Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2018

One Million-Point FFT

Hans Kanders and Tobias Mellqvist

LINKÖPING
UNIVERSITY

Master of Science Thesis in Electrical Engineering

One Million-Point FFT

Hans Kanders and Tobias Mellqvist

LiTH-ISY-EX--18/5111--SE

Supervisor: **Mario Garrido**
ISY, Linköpings universitet

Examiner: **Kent Palmkvist**
ISY, Linköpings universitet

Division of Computer Engineering
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Sammanfattning

Målet med detta examensarbete har varit att konstruera ett digitalt system som beräknar den snabba Fouriertransformen (FFT) av en signal som är samplad i en miljon datapunkter och som skall implementeras på en FPGA. Arkitekturen har designats som ett seriellt återkopplat system (single-delay feedback, SDF) med rotatorer och adderare, däribland en trestegs-rotator med en miljon vinklar. Systemet har implementeras på en FPGA med en datatakt om 233 miljoner punkter i sekunden. Den beräknade transformen har hög noggranhet med ett signal-brusförhållande (SQNR) om 95.6 dB.

Abstract

The goal of this thesis has been to implement a hardware architecture for FPGA that calculates the fast Fourier transform (FFT) of a signal using one million samples. The FFT has been designed using a single-delay feedback architecture with rotators and butterflies, including a three-stage rotator with one million rotation angles. The design has been implemented onto a single FPGA and has a throughput of 233 Msamples/s. The calculated FFT has high accuracy with a signal to quantization noise ratio (SQNR) of 95.6 dB.

Acknowledgments

We want to direct a huge thanks to our supervisor Mario Garrido who has devoted a lot of time to help us and to whom we refer to as the "FFT-man". When we first visited him to discuss the subject of the thesis he was full of ideas and seemed to be really passionate about the subject.

We also want to thank our examiner Kent Palmkvist who not only has helped us with the thesis but also has been helping us a lot with both software and hardware related problems. When the computers break down you run to Kents office.

Jag, Hans, vill rikta ett enormt tack till mina föräldrar, Lena Kanders och Lars-Erik Larsson, som under hela min studietid stöttat mig på flera sätt. När studierna känts tunga och jag tvivlat på min egen förmåga har ni aldrig gjort det och ni har sagt att jag kommer att klara mig. Ni hade ju rätt hela tiden...

Jag, Tobias, vill tacka mina föräldrar Maria och Enar samt min tvillingbror Jonatan oerhört mycket för allt stöd de har gett mig under hela studietiden. Ni har hjälpt mig att hålla siktet framåt och underlättat så enormt i tunga och kämpiga stunder. Tack! Sedan vill jag också rikta ett stort tack alla mina vänner som gjort studietiden fantastisk. Utan er hade jag vetat varken fram eller bak.

Linköping februari 2018,
i en vinter som har svårt att bestämma sig

Hans Kanders och Tobias Mellqvist

Contents

Notation

Abbreviations

Abbreviation	Meaning
AGU	Adress Generating Unit
ASIC	Application-Specific Intergrated Circuit
BRAM	Block Random Access Memory
CCSSI	Combined Coefficient Selection and Shift-and-add Implementation
DFT	Discrete Fourier Transform
DIF	Decimation-In-frequency
DIT	Decimation-In-time
FFT	Fast Fourier Transform
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
IDFT	Inverse Discrete Fourier Transform
LSB	Least Significant Bit
MDC	Multi-Path Delay Commutator
MDF	Multi-Path Delay Feedback
MSB	Most Significant Bit
SDC	Single-Path Delay Commutator
SDF	Single-Delay Feedback
SFFT	Sparse Fast Fourier Transform
SFG	Signal Flow Graph
SQNR	Signal-to-Quantization-Noise Ratio
VHDL	Very high speed integrated circuit Hardware Description Language

1 Introduction

The ever increasing number of transistors that can be fit into an integrated circuit has revolutionized the modern world and made way for new innovations such as powerful computers and the smartphone that almost everyone carry around in their pockets. However, this digital revolution would not have been possible without the fast Fourier transform (FFT) [7], which enables the ability to analyze a signals frequency composition.

This transistor density explosion has also had an effect on field programmable gate arrays (FPGAs) which are now capable of being configured with larger designs running at higher speeds than ever before. This, combined with a thorough research and development in the area of FFT algorithms and architectures [10, 14, 17, 18, 20, 21, 24, 51], has made it possible to fit very large FFT architectures into a single FPGA chip, cutting the need for using multiple FPGAs [27] or expensive application specific integrated circuits (ASIC).

These very large FFT architectures can be of great value in areas such as radio astronomy [2, 27, 39] where a large portion of the electromagnetic spectrum has to be analyzed at high speed to get a fundamental understanding about celestial objects. Other areas include computational biology, medical imaging, etc. [2].

Today, implementations of very large FFTs can be very expensive in terms of used hardware resources. The goal in this thesis has been to design and implement a one million-point FFT, 1M-point FFT for short, with the focus on using less hardware. The 1M-point FFT is actually a $2^{20} = 1\,048\,576$-point FFT. This implementation should fit onto a single FPGA chip using only on-chip memory and also be able to run accurately with a throughput of at least 200 Msamples/s, which is reasonable for a serial architecture of this type.

One task has been to find an architecture that is suitable for the large amount of memory required to implement a 1M-point FFT. This also includes selecting an algorithm that will minimize the complexity of the rotators to reduce their hardware utilization. Another task has been to design a one million angle rotator, called the W_{1M} rotator. This rotator is a part of the 1M-point FFT architecture, but has been listed as a separate task since its large size presents challenges in itself. The complexity of the rotator requires very high precision at the same time as the reduction of hardware utilization is in focus. The third task has been to find a suitable FPGA that is large enough to fit the whole implementation, especially in terms of on-chip memory.

The outline of the thesis is described in the following section. Chapter 2 gives a background on the Fourier transform and the derivation of the FFT algorithm. This chapter also covers what operations are used in the calculation and how this is translated onto digital systems using fixed-point arithmetic. Chapter 3 presents methods and approaches regarding FFT design and implementation that is relevant for this work. The proposed design and the separate design steps are presented in chapter 4 and is followed the hardware implementation in chapter 5. Experimental results and conclusions are found in chapter 6 and 7, respectively. The final chapter gives an insight of possible future work.

2

Background

2.1 The Fast Fourier Transform

Just like any color can be decomposed into a number of primary colors, any signal can be decomposed into simple frequencies that together make up the signal. The signal can be anything from a sound wave from a musical instrument to a traveling electromagnetic wave from a distant star. The study of Fourier analysis is the study of how functions are made up of a sum of simpler trigonometric functions and has many applications in various fields. In signal processing, Fourier analysis is an indispensable tool for everything from image processing to radio applications.

2.1.1 Origin

The history of the Fourier transform dates back to the early 19th century and the French mathematician Jean-Baptiste-Joseph Fourier [28]. He discovered that any function that is defined in a finite interval can be expressed as a sum of sinusoids with various frequencies. This led to the concept of Fourier series for representation of periodic signals and to its extension, the Fourier integral, for aperiodic signals. The Fourier integral is found in the right-hand side of equation (2.2).

When analyzing continuous-time signals, the Fourier transform allows transformation between the time domain and the frequency domain. Let $x(t)$ be a continuous signal in the time domain, then its frequency domain specification or spectra, $X(f)$, is obtained from

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi tf}\,dt, \tag{2.1}$$

where $X(f)$ is also continuous and is the direct Fourier transform of $x(t)$, f is the frequency in Hz. The signal $x(t)$ can be restored from $X(f)$ by using the inverse Fourier transform

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi f t} df. \tag{2.2}$$

2.1.2 The Discrete Fourier Transform

In digital systems it is not feasible to store or perform operations on continuous-time signals. Instead, these continuous signals are represented by sequences of discrete data samples equally spaced in time. To be able to perform the Fourier transform on these discrete signal representations, the discrete Fourier transform (DFT) was derived. It transforms a sampled signal sequence in the time domain to its corresponding components in the frequency domain. The definition is

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}nk} = \sum_{n=0}^{N-1} x[n] W_N^{nk}, \quad k = 0, 1, ..., N-1, \tag{2.3}$$

where $x[n]$, $n = 0, 1, ..., N-1$, is the sampled signal sequence in the time domain and $X[k]$, $k = 0, 1, ..., N-1$, is the resulting frequency components, also called bins. N denotes the size of the DFT. The factor W_N^{nk} is called twiddle factor and is defined as

$$W_N^{nk} = e^{-j\frac{2\pi}{N}nk} = \cos\left(\frac{2\pi}{N}nk\right) - j\sin\left(\frac{2\pi}{N}nk\right). \tag{2.4}$$

A multiplication with a twiddle factor corresponds to a rotation in the complex plane. With an N sample data input, the DFT produces an equally sized N sample data output. As for the case of the Fourier transform on continuous-time signals, the DFT can be reversed with the inverse discrete Fourier transform (IDFT), which is defined as

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{nk}, \quad n = 0, 1, ..., N-1. \tag{2.5}$$

The computational complexity of an N-point DFT is $\mathcal{O}(N^2)$, which soon becomes very large as N increases. For that reason, it is of great interest to optimize the DFT algorithm to reduce the the number of computations.

2.1.3 The FFT Algorithm

The fast Fourier transform (FFT) is actually a collective name for many different algorithms. What they have in common is the reduced computational complexity in comparison to the direct calculation of the DFT. The most common FFT algorithm is the Cooley-Tukey algorithm [7], where the principle is to decompose the original DFT into a set of smaller DFTs. Two ways of decomposition are decimation-in-frequency (DIF) and decimation-in-time (DIT) [40]. The derivation of the DIF is given next.

Starting with the definition in equation (2.3), the input sequence is split in half forming the two separate sequences

$$x[n_1]\,,\quad n_1 = 0, 1, \ldots, \frac{N}{2} - 1,$$
$$x[n_2]\,,\quad n_2 = \frac{N}{2}, \ldots, N-2, N-1.$$

Equation (2.3) then becomes

$$X[k] = \sum_{n=0}^{N/2-1} x[n]W_N^{nk} + \sum_{n=N/2}^{N-1} x[n]W_N^{nk}. \tag{2.6}$$

In the next step the second term is rewritten as

$$\sum_{n=N/2}^{N-1} x[n]W_N^{nk} = W_N^{(N/2)k} \sum_{n=0}^{N/2-1} x\left[n + \frac{N}{2}\right] W_N^{nk}, \tag{2.7}$$

and together with $W_N^{N/2} = e^{-j\frac{2\pi}{N}\cdot N/2} = -1$, the following is obtained:

$$X[k] = \sum_{n=0}^{N/2-1} \left(x[n] + (-1)^k x\left[n + \frac{N}{2}\right]\right) W_N^{nk}. \tag{2.8}$$

The frequency components $X[k]$ are split into even and odd indices. $(-1)^k = 1$ for even indices $k = 2m$, $m = 0, 1, \ldots, \frac{N}{2} - 1$, and $(-1)^k = -1$ for odd $k = 2m + 1$. This yields

$$X[2m] = \sum_{n=0}^{N/2-1} \left(x[n] + x\left[n + \frac{N}{2}\right]\right) W_N^{n(2m)}, \tag{2.9a}$$

$$X[2m+1] = \sum_{n=0}^{N/2-1} \left(x[n] - x\left[n + \frac{N}{2}\right]\right) W_N^{n} W_N^{n(2m)}. \tag{2.9b}$$

The fact that $W_N^2 = e^{-j\frac{2\pi}{N/2}} = W_{N/2}$ is used to obtain

$$X[2m] = \underbrace{\sum_{n=0}^{N/2-1} a[n]W_{N/2}^{nm}}_{N/2\text{–}point\ DFT} \tag{2.10a}$$

$$X[2m+1] = \underbrace{\sum_{n=0}^{N/2-1} \left(b[n] \underbrace{W_N^n}_{twiddle\ factor} \right) W_{N/2}^{nm}}_{N/2\text{–}point\ DFT}. \tag{2.10b}$$

Thus, the N-point DFT in equation (2.3) is now described as two $\frac{N}{2}$-point DFTs, where

$$a[n] = x[n] + x\left[n + \frac{N}{2}\right], \tag{2.11}$$

$$b[n] = x[n] - x\left[n + \frac{N}{2}\right], \tag{2.12}$$

for $n = 0, 1, \dots, \frac{N}{2} - 1$. The two separate $\frac{N}{2}$-point DFTs in equation (2.10a) and equation (2.10b) can be split further in the same way. By iterating this process, the original DFT is broken down into $\frac{N}{2}$ number of 2-point DFTs. A 2-point DFT is calculated as

$$X[k] = \sum_{n=0}^{1} x[n] W_2^{nk} \Rightarrow \underbrace{\begin{cases} X[0] = x[0] + x[1] \\ X[1] = x[0] - x[1] \end{cases}}_{radix-2\ butterfly\ operation}. \tag{2.13}$$

All iterations includes a twiddle factor in the odd portion of every split which is shown in equation (2.10b). Figure 2.1 illustrates this using a signal flow graph (SFG) representation, with the corresponding SFG operations seen in figure 2.2. The final result is shown in figure 2.1c, where the 8-point FFT calculation in the SFG is divided into three stages. Each stage consists of a butterfly operation and a complex multiplication. For an N-point FFT, its SFG representation consists of n stages, where $n = \log_2 N$.

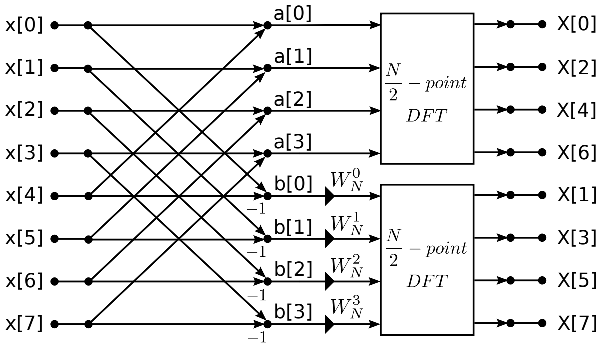
The number of twiddle factors in the SFG corresponds to the number of rotations that needs to be performed. For example, the first iteration uses the factors W_N^n, $n = 0, 1, \dots, \frac{N}{2} - 1$. The number of complex multiplications this corresponds to is $\frac{N}{2}$. The second iteration needs $2 \cdot \frac{N}{4}$ number of complex multiplications and so on. The resulting total number of complex multiplications that is required is

$$\#_{mult} = \underbrace{1 \cdot \frac{N}{2}}_{1st\ iteration} + \underbrace{2 \cdot \frac{N}{4}}_{2nd\ iteration} + \underbrace{4 \cdot \frac{N}{8}}_{3rd\ iteration} + \dots + \underbrace{\frac{N}{2} \cdot 1}_{last\ iteration}$$

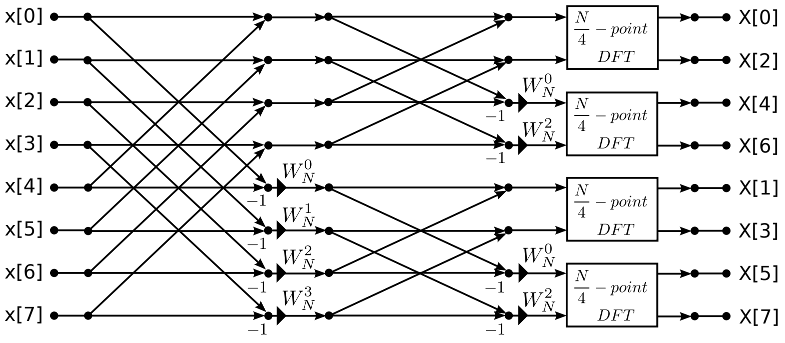
$$= N \underbrace{\left(\frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \dots + \frac{1}{2}\right)}_{\log_2 N\ iterations} = \frac{N}{2} \log_2 N. \tag{2.14}$$

Compared to the original DFT that requires N^2 rotations, the above method has reduced it to $\frac{N}{2} \log_2 N$. The significance of this becomes very obvious for larger FFTs. For $N = 2^{20}$, a comparison is shown in table 2.1 which compares the number of rotations of the DFT and FFT with 2^{20} points.

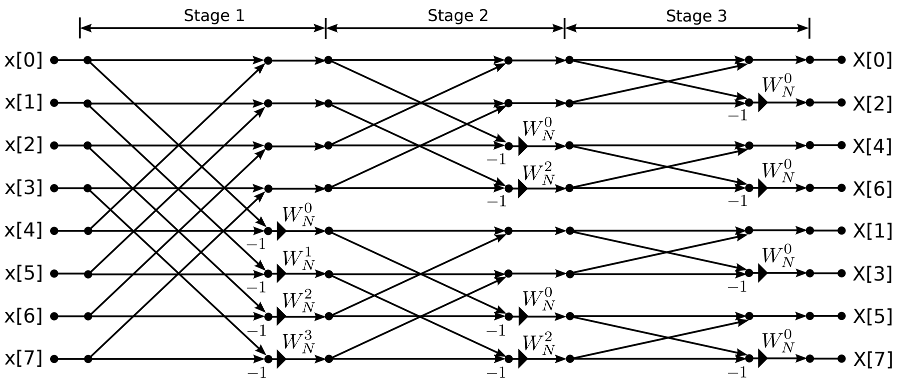
It is clear that using DIF simplifies the computation of the DFT. It takes advantage of the fact that the same rotation is performed on more than one input sample, which comes from the symmetry and periodicity in the twiddle factors.



(a) First iteration



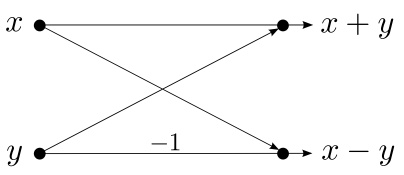
(b) Second iteration



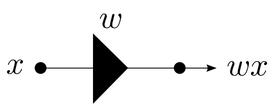
(c) Final result

Figure 2.1: *Signal flow graphs of an 8-point FFT.*

The FFT algorithm derived above is called a radix-2 [24] DIF FFT. Radix-2 means that the smallest decomposition consists of 2-point DFT. Its expansion in equation (2.13) is called a radix-2 butterfly operation. Furthermore, radix-2 also implies that the size N needs to be a multiple of two. In a similar fashion, the



***(a)** Radix-2 butterfly*



***(b)** Rotation*

***Figure 2.2:** Signal flow graph operations.*

Table 2.1

N	Algorithm	Number of rotations	Ratio
2^{20}	DFT	$\approx 1.1 \cdot 10^{12}$	104 858
	DIF FFT	$\approx 2.1 \cdot 10^{7}$	1

radix-2 DIT FFT is obtained, but instead of the frequency components the time domain sequence is decomposed.

2.1.4 Rotations in the Complex Plane

A multiplication by a twiddle factor W_N^{nk} [20, 26] corresponds to a rotation in the complex plane by an angle α. It is calculated as

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}, \tag{2.15}$$

where $x + jy$ and $X + jY$ are the complex input and output, respectively. The exact coefficient to multiply with depends on the value of n and k. For clarity, let $\phi := nk$, and express the twiddle factor as

$$W_N^{\phi} = e^{-j\frac{2\pi}{N}\phi}. \tag{2.16}$$

This gives an angle resolution of $\frac{2\pi}{N}$ radians, which is the smallest rotation angle. The number ϕ indicates how many smallest angles the rotation consists of. Often, the value of ϕ itself is used in the SFG to represent a certain twiddle factor.

A set of twiddle factors W_L^{ϕ}, where $\phi = 0, 1, \ldots, L-1$, is called a W_L rotator. Such a rotator is capable of performing L rotations with the resolution $\frac{2\pi}{L}$ radians, resulting in the angle set $\alpha \in \left\{0, \frac{2\pi}{L}, \frac{2\cdot 2\pi}{L}, \ldots, \frac{(L-1)2\pi}{L}\right\}$.

From equation (2.16), the rotation angle is defined in a counter-clockwise, or positive, direction, as $\alpha = -\frac{2\pi}{N}\phi$. For a positive ϕ, the resulting rotation is in a clockwise, or negative, direction. This is shown in figure 2.3 for a W_4 rotator. The angle α is expressed in radians together with its corresponding rotation.

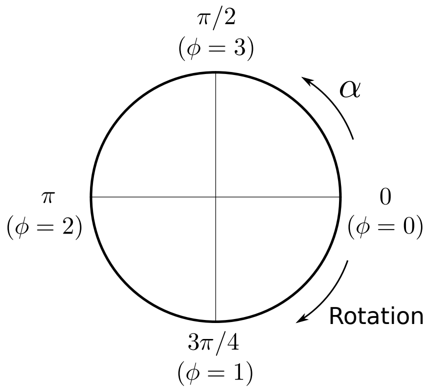


Figure 2.3: *Illustration of the rotation direction, using a W_4 rotator.*

The number of angles in a rotator decides the rotator complexity. Naturally, the larger number of angles, the more complex the rotator. The numbers $\phi \in \{0, N/4, N/2, 3N/4\}$ corresponds to the rotation angles 0°, 270°, 180° and 90°. This can be described as a W_4 rotator, with 4 angles and the angle resolution of $\frac{2\pi}{N} = \frac{2\pi}{4} = \frac{\pi}{2}$. In particular, the W_4 rotator is also called a trivial rotator since its angle set corresponds to complex multiplications by 1, $-j$, -1 or j. These are simply performed by swapping and/or negating the real and imaginary components.

As a side note, it can also be mentioned that a radix-2 butterfly operation in practice holds the functionality of a W_2 rotator. It performs rotations by 0° and 180°, which correspond to the complex multiplications by 1 and -1.

2.2 The Field Programmable Gate Array

A field programmable gate array (FPGA) [43] is an integrated circuit with logic blocks and interconnects that can be configured to the customers need. This is usually done by specifying the configuration with a hardware description language (HDL). The advantage of using an FPGA over an application specific integrated circuit (ASIC) that cannot be reconfigured, is that many FPGAs can be produced in advance and be configured in many different ways, cutting costs and time to market. This can be compared to a box of Legos (the FPGA) that can be configured in many shapes (the configuration) as long as you do not run out of Lego blocks (logic blocks). Like the box of Legos, the FPGA can be reconfigured many times.

2.3 Error Estimation

This section discusses various approaches of error estimation.

2.3.1 Rotations in Fixed-Point Arithmetic

When calculating rotations in digital systems an error will always be introduced [13]. The reason is that the number of bits that are used to represent a value is finite. A twiddle factor W_N^{ϕ}, that corresponds to an angle α, is described with a complex coefficient $C + jS$, where C and S are the real and imaginary components, respectively. The calculation of a rotation is obtained through

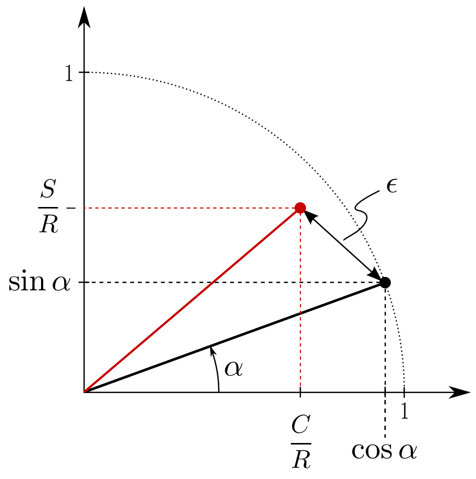
$$\begin{bmatrix} X_D \\ Y_D \end{bmatrix} = \begin{bmatrix} C & -S \\ S & C \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}, \tag{2.17}$$

where $x + jy$ is the complex input, and $X_D + jY_D$ the output.

Due to finite word length effects in fixed-point arithmetic the rotation coefficients C and S can not be described exactly to represent all angles α. The coefficients $C, S \in [-2^{b-1}, 2^{b-1} - 1]$, where b is the coefficient word length, are obtained from

$$\begin{aligned} C &= R \cdot (\cos\alpha + \epsilon_c), \\ S &= R \cdot (\sin\alpha + \epsilon_s). \end{aligned} \tag{2.18}$$

ϵ_c and ϵ_s are the relative quantization errors of the coefficients and R is the scaling factor. The rotation error for an angle α can be calculated as $\epsilon = \sqrt{\epsilon_c^2 + \epsilon_s^2}$, which is the distance between the exact and the quantized rotation. This is depicted in figure 2.4.



(a)

Figure 2.4: *The rotation error ϵ*

A set of coefficients $C_i + jS_i, i = 1, ..., M$ is called a kernel set. For a set of rotation angles α_i, $i = 1, \ldots, M$, the error for each of the corresponding coefficients $C_i + jS_i$ is expressed as

$$\epsilon_i = \left(\frac{\sqrt{(C_i - R \cdot \cos\alpha_i)^2 + (S_i - R \cdot \sin\alpha_i)^2}}{R} \right). \tag{2.19}$$

The rotation error of the entire set is then defined as the maximum rotation error of the individual angles,

$$\epsilon_{max} = \max_i(\epsilon_i)\,. \tag{2.20}$$

2.3.2 Estimation of the Scaling Factor

The radius [13] of a rotator with a kernel set $C_i + jS_i$, $i = 1, \ldots, M$ where $C, S \in [-2^{b-1}, 2^{b-1} - 1$ refers to the radius R that minimizes the function

$$\Delta R_{max} = \max_i\left(\left|\sqrt{C_i^2 + S_i^2} - R\right|\right)$$

Different kernels $C_i + jS_i$ are at different distances $\sqrt{C_i^2 + S_i^2}$ from origo. A rotation by an angle $\alpha_i = \arctan\left(\frac{S_i}{C_i}\right)$ will scale the result differently. The selected radius R is the radius that will give the smallest error $\left|\sqrt{C_i^2 + S_i^2} - R\right|$ when downscaling the result.

The selected radius is found in the middle between the points closest and furthest away from origo, see figure 2.5.

$$R = \frac{\min_i\left(\sqrt{C_i^2 + S_i^2}\right) - \max_i\left(\sqrt{C_i^2 + S_i^2}\right)}{2}$$

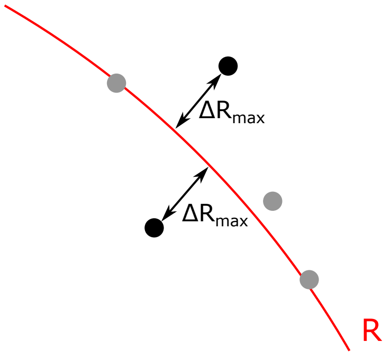


Figure 2.5: *Radius selection*

2.3.3 Signal-to-Quantization-Noise Ratio (SQNR)

The signal-to-quantization-noise ratio (SQNR) can be used to measure the accuracy of an FFT. It is usually expressed in decibels (dB) as

$$\text{SQNR (dB)} = 10 \cdot \log_{10}\left(\frac{E\{|X_Q|^2\}}{E\{|X_Q - X_I|^2\}}\right). \tag{2.21}$$

E is the expected value, X_Q is the quantized FFT output and X_I is the the ideal output where no quantization occurs. The distance $|X_Q - X|$ represents the error, also called the quantization noise, due to finite word lengths in digital systems.

3 State of the Art

The area of fast Fourier transform research is vast and the results are used in implementations in various fields. This section aims to compile results relevant to this thesis.

3.1 FFT Algorithms

There exists multiple ways of representing FFT algorithms. In section 2.1.3 the signal flow graph representation was presented. This section will present two more representations, both of which are useful in different ways.

3.1.1 The Binary Tree Representation

In the binary tree representation of an $N = 2^n$-point DFT, [29, 41] N is first split into two factors, $N = P \cdot Q$. Then, P and Q are in turn split into two factors each. This is repeated, resulting in a tree where each branch is divided in two. The rooted node has the value n, where $N = 2^n$. Its branches have the values p and q, where $P = 2^p$ and $Q = 2^q$. Hence, the value of a node equals the sum of its subnodes, i.e., $n = p + q$. This can be seen in figure 3.1. In a binary tree, each of the nodes refer to a rotator, where the node numbers describe the rotator complexity. The position of the node decides in which stage of the FFT the rotator should be placed. Since each node can be split in many ways, a large number of algorithms is possible.

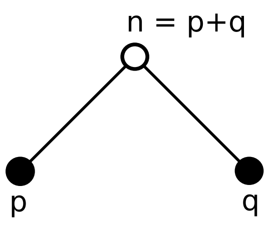


Figure 3.1: *Illustration of how a node is split in the binary tree representation*

Once a binary tree is formed, the rotators at each FFT stage can be obtained directly by following the branches from left to right. Figure 3.2 shows an example of two different algorithms for a 2^5-point FFT represented with binary trees.

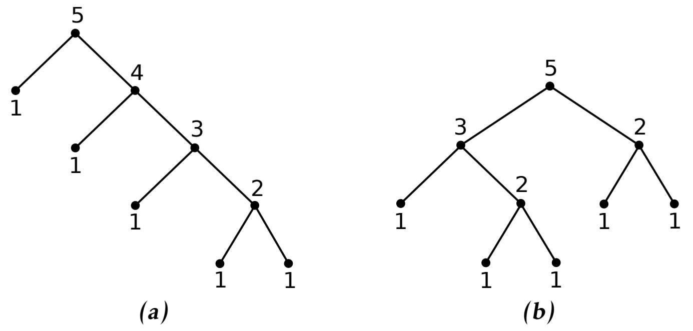


Figure 3.2: *Example of two algorithms for a 2^5-point FFT using binary trees*

In figure 3.2a the order of the twiddle factors at the FFT stages becomes 5-4-3-2. The end branches with value 1 correspond to radix-2 DFT or butterfly operations. In the same way the algorithm in figure 3.2b yields the order 3-2-5-2. The number of angles of the twiddle factors is equal to two to the power of these values. Hence, the two algorithms above yield the rotator order W_{32}-W_{16}-W_8-W_4 and W_8-W_4-W_{32}-W_4, respectively.

3.1.2 The Triangular Matrix Representation

Another useful approach to represent the FFT algorithm is the triangular matrix representation [17]. For an FFT algorithm with $N = 2^n$ points, the triangular matrix has $n-1$ rows and columns. The algorithm in figure 3.2b is represented by using a triangular matrix in figure 3.3. The upper right corner represents the rotation W_{32}^1, the second diagonal $y - x = 2$ represents W_{32}^2, the third $y - x = 1$ represents W_{32}^4 and so on. The last diagonal represents trivial rotations.

The numbers in each element group represents the stage where the rotation is carried out. Rotations that belong to the same stage have been grouped into large rectangles in figure 3.3 for clarity.

The triangular matrix can be used to easily obtain the twiddle factors for each stage in the FFT architecture. The twiddle factors can be calculated as [11]

$$\phi_s(I) = \sum_{M_{xy}=s} b_{n-x} \cdot b_{n-y-1} \cdot 2^{n+(x-y)-2} \tag{3.1}$$

where $\phi_s(I)$ is the twiddle factor at stage s with index I, and x and y are the rows and columns of the triangular matrix and b is the index number in binary format. The index number relates to in what order the signal sample arrives at the serial input.

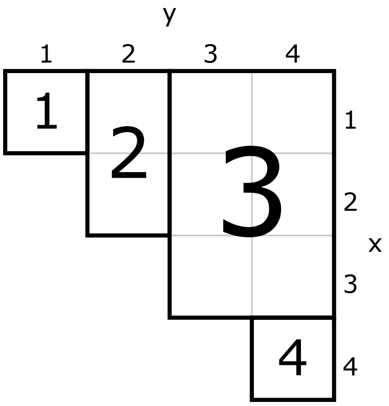


Figure 3.3: Example of a triangular matrix representation

3.2 Pipelined FFT Architectures

The high performance requirements of today's real-time applications make pipelined FFT architectures [5, 6, 8, 9, 12, 16, 18, 20, 22–25, 30–35, 37, 38, 42, 44, 45, 49–51] useful. They provide high throughput and reasonably low area and power consumption. Feedback [20] and feedforward [18] are the two main types of pipelined architectures. Both types can be implemented with serial or parallel structures. In the case of feedback, these are called single-path delay feedback (SDF) [8, 9, 23, 24, 42, 51] and multi-path delay feedback (MDF) [6, 30, 31, 34, 35, 45, 49, 50], respectively. For feedforward architectures, these are known as single-path delay commutator (SDC) [3, 4, 36, 47] and multi-path delay commutator (MDC) [24].

3.2.1 Single-Path Delay Feedback

Allowing for high throughput and a low amount of hardware resources makes the SDF architecture [20] one of the most attractive and frequently used architectures. Figure 3.4 illustrates the structure of a 16-point radix-2 SDF FFT. It consists of a series of $n = \log_2(16) = 4$ interconnected stages, where each stage

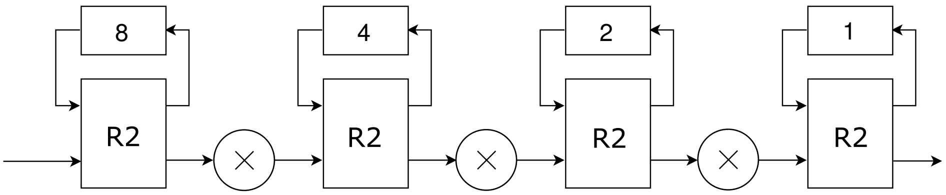


Figure 3.4: *Structure of a 16-point radix-2 SDF FFT*

is composed of a radix-2 butterfly and a rotator. The output from the butterflies are connected to first-in-first-out (FIFO) registers in a feedback loop, which allows for the single-path input and processing of one sample per clock cycle.

The calculations performed in stage s, where $s \in \{1, ..., n\}$, corresponds to that of the same stage in the SFG, and can therefore be easily be translated. The same way goes for the binary tree representation. The rotator order, which is explained in section 3.1.1, describes in what stage in the SDF architecture the different rotators are placed.

To increase throughput in the SDF architecture, more pipeline registers can be inserted to reduce the critical path.

3.3 Large FFTs

Radio astronomy, computational biology and medical imaging are some areas that increasingly demand the applications of large FFT computations [2]. In particular, some recent works have explored FFT sizes of up to a million-point range [1, 2, 27].

Implementations of very large FFTs for frequency-sparse signals are proposed in [1, 2]. A sparse FFT (SFFT) handles signals where only a few frequencies have energy and the rest are noise, which comes with some simplifications compared to the standard FFT. In [2], a million-point SFFT is implemented on a single Virtex-6 FPGA, capable of performing real-time computations with an input sample rate of 0.86 Gsamples/s. At the time of the publication, the authors claimed that efficient million-point FFTs were not practical due to high energy consumption and large area requirement in hardware. However, in this thesis it is shown that the implementation of a hardware efficient 1M-point FFT is feasible.

In [39], a million-point FFT is implemented on several FPGAs.

3.4 Rotator Architectures

A large portion of the resources used in a FFT hardware implementation is used by the rotators. The topic of rotator architectures [10, 13, 15, 19, 21, 26] is a vast research area in itself and is used for many applications apart from the FFT. Depending on the complexity and application of a certain rotator, there are different architectures that can be utilized.

Rotators are divided into two main types. These are constant rotators [18, 24] and general rotators [11]. Constant rotators carry out rotations by specific angles, while general rotators are able to rotate by any angle which is given as an input [19]. Typical for FFT applications is that the rotation angles are known beforehand. In this case, constant rotators allow for hardware simplifications that can not be applied in the case of general rotators. However, when rotators become larger, constant rotators do not provide enough simplification and general rotators are used instead.

3.4.1 Complex Multiplier

One way to carry out a rotation is to use the standard complex multiplier [11]. It consists of four real-valued multipliers and two real-valued adders; this architecture is derived from the calculation of a complex multiplication. A complex multiplier classifies as a general rotator due to the fact that the architecture allows for a rotation with any angle fed to its input. In FFT implementations, the rotation coefficients are usually provided from a memory. As mentioned before, this approach is suitable for larger rotators.

There are methods to reduce the number of real-valued multipliers to three instead of four, based on rewriting the original equations. Some of these structures are presented in [48]. However, this is out of the scope of the thesis.

3.4.2 Shift-and-Add Implementations

Implementations of multiplications can be simplified to using only shifts and additions and, hence, become multiplierless. This is useful for rotators with few rotation angles.

One technique, proposed in [19], is the combined coefficient selection and shift-and-add implementation (CCSSI). Where in previous approaches most effort lies in the shift-and-add implementation, this method puts the coefficient selection to equal importance. This means that there are no restrictions set to $C + jS$ beforehand, and the coefficient values are optimized together with the shift-and-add implementation.

3.4.3 Multi-Stage Rotators

Rotators such as the CORDIC [10, 46] and CORDIC II [15] rotators, and the W_{1M} rotator implemented in this thesis, use a multi-stage rotator approach. The main principle in these rotators is to break down the rotation into a set of so called microrotations that lead to the rotation angle.

3.4.4 Nanorotators

There also exists so called nanorotators [15] which are suitable for handling the smallest angles in a multi-stage rotator architecture. It works accurately on the premise that the input angle from the previous rotator stage is small enough, typically $\alpha_{in} \leq 1°$. The coefficient set is defined as $P_k = C + jk$, where C is a constant and $k = 0, \ldots, N$. Since $N \ll C$, the scaling of all the angles is approximately the same, and the angle set can be obtained with the use of the small-angle approximation,

$$\alpha_k = \tan^{-1}\left(\frac{k}{C}\right) \approx \frac{k}{C}. \tag{3.2}$$

This makes the angles equally distributed. The angle set of the nanorotator is depicted in figure 3.5.

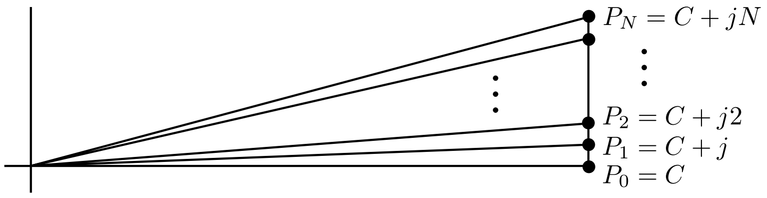


Figure 3.5: *Angle set of the nanorotator*

4 Proposed Design

This chapter presents the design flow and methodology of the proposed 1M-point FFT.

First, a desired architecture will be motivated, followed by a selection of a suitable FFT algorithm to see the complexity of the required rotators. This is then translated into the chosen architecture. The twiddle factor generation using the triangular matrix representation will be presented. Thereafter, efforts to optimize the SQNR by adapting the signal scaling throughout the whole system will be discussed. Lastly, the separate FFT building blocks are presented, starting with the design of the W_{1M} rotator.

4.1 Architecture

The architecture used for the 1M-point FFT is the SDF-architecture presented in section 3.4. The SDF-architecture is suitable for the 1M-point FFT because it is resource efficient and has high throughput.

4.2 Selection of an FFT algorithm

The FFT algorithm has an impact on the rotator placement in an SDF architecture. Since the W_{1M} rotator has the highest complexity of all the rotators, it is preferred to place it as late as possible. The reason for this is that the high complexity requires a larger amount of bits for good precision. This in turn needs to be stored in buffers in each stage when applying an SDF architecture. Since the size of the buffers is cut in half for each stage in the FFT, it is preferable to place the W_{1M} rotator as late as possible to decrease the amount of required memory.

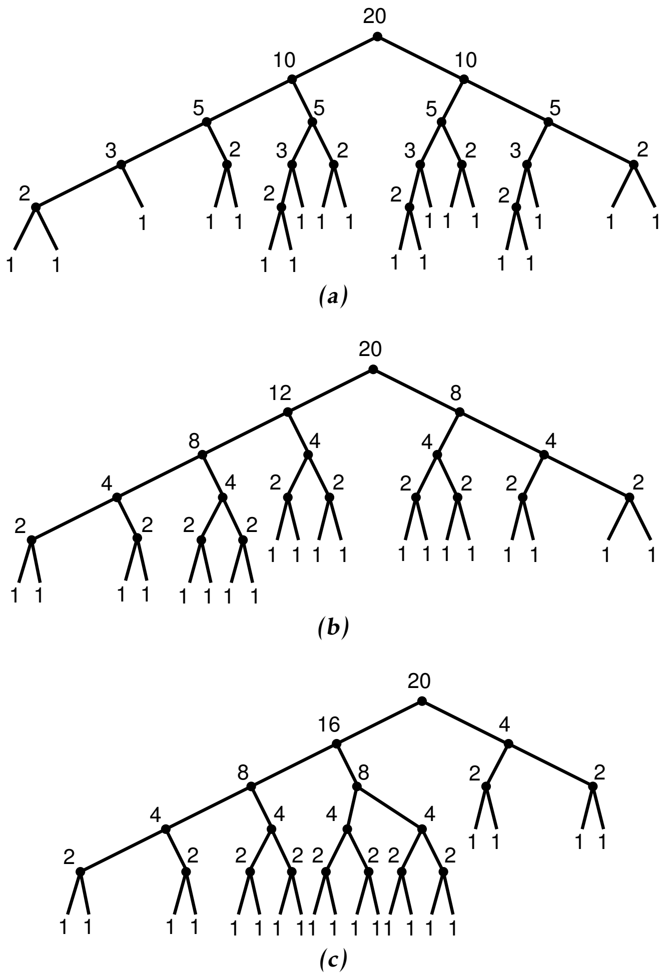


Figure 4.1: *FFT algorithms represented using binary trees. (a), (b) and (c) illustrate the different splits 10-20-10, 12-20-8 and 16-20-4, respectively.*

On the other hand, it is desired to use as many trivial rotators as possible. And also, reduce the complexity of the rest of the rotators. These requirements lead to the three following alternatives in figure 4.1 using binary tree representations. They are denoted *10-20-10*, *12-20-8* and *16-20-4*, which is derived from their individual split in the root node. As explained earlier, the numbers in a binary tree is related to the complexity of the corresponding rotator. For example, the number 20 represents the W_{1M} rotator which has 2^{20} rotation angles. Table 4.1 shows the rotators that are required for each of the three candidates. The first column shows the stages of the FFT. The second through fourth columns show the rotators at each stage. The trivial W_4 rotators are highlighted in gray and the placement of the W_{1M} rotator in red.

Firstly, it is obvious that 10-20-10 contains more non-trivial rotators than 12-

Table 4.1: *Required rotators and their order at the FFT stages for each algorithm.*

	Algorithm		
Stage	10-20-10	12-20-8	16-20-4
1	W_4	W_4	W_4
2	W_8	W_{16}	W_{16}
3	W_{32}	W_4	W_4
4	W_4	W_{256}	W_{256}
5	W_{1024}	W_4	W_4
6	W_4	W_{16}	W_{16}
7	W_8	W_4	W_4
8	W_{32}	W_{4096}	W_{65K}
9	W_4	W_4	W_4
10	W_{1M}	W_{16}	W_{16}
11	W_4	W_4	W_4
12	W_8	W_{1M}	W_{256}
13	W_{32}	W_4	W_4
14	W_4	W_{16}	W_{16}
15	W_{1024}	W_4	W_4
16	W_4	W_{256}	W_{1M}
17	W_8	W_4	W_4
18	W_{32}	W_{16}	W_{16}
19	W_4	W_4	W_4
Trivial rotators	8	10	10

20-8 and 16-20-4 which is not desirable. In 10-20-10, the W_{1M} rotator also appear the earliest in the pipeline out of all three algorithms. This algorithm is therefore discarded because it would use more memory. Between 12-20-8 and 16-20-4, both have the same number of trivial rotators. In 16-20-4 the W_{1M} rotator appears the latest in the pipeline, but another very large rotator, W_{65K} , is required. Hence, this algorithm is also discarded because it would require the implementation of another large rotator. This leaves us with 12-20-8. This algorithm is selected since it has the least number of non-trivial rotators, places W_{1M} later than 10-20-10 and does not introduce another very large rotator as the 16-20-4 architecture does.

The selected algorithm, 12-20-8, is translated to the SDF architecture depicted in figure 4.2. As mentioned earlier, a stage consists of a radix-2 butterfly, a buffer and a rotator. Trivial rotators are depicted with a diamond shape, W_{16} rotators with squares, complex multipliers with a circle and the W_{1M} rotator with an octagon. The last stage has no rotator since the twiddle factors here corresponds to a rotation by 0°. The sizes of the FIFO buffers denote the number of complex samples that they should be able to store.

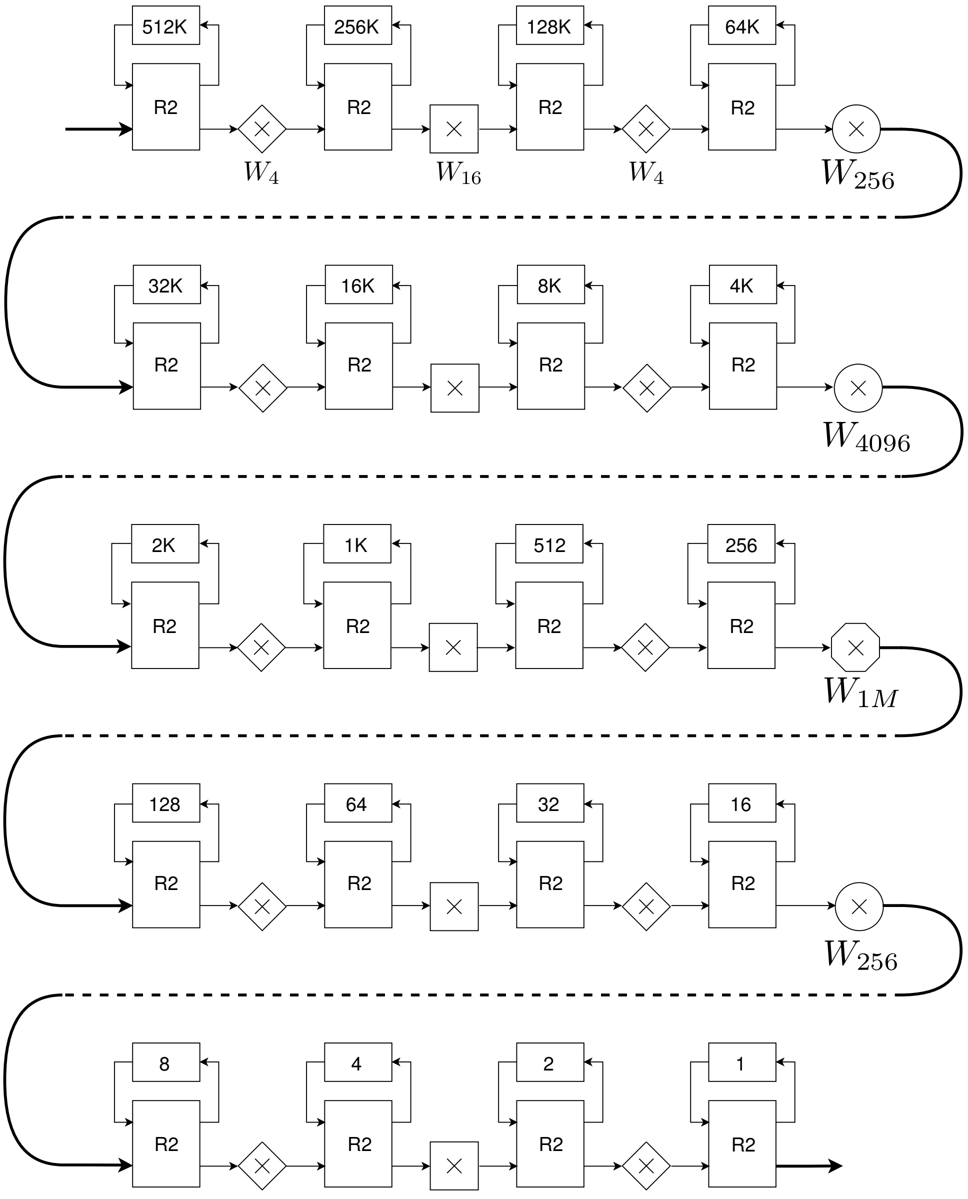


Figure 4.2: *Structure of the 1M-point SDF FFT.*

4.3 Obtaining the Twiddle Factors

As discussed in section 3.1.2, the triangular matrix representation can be used to obtain the twiddle factors for each stage in the FFT architecture. The twiddle factors can be calculated as [11]

$$\phi_s(I) = \sum_{M_{xy}=s} b_{n-x} \times b_{n-y-1} \times 2^{n+(x-y)-2} \tag{4.1}$$

where $\phi_s(I)$ is the twiddle factor at stage s with index I, x and y are the rows and columns of the triangular matrix and b is the index number in binary format.

The triangular matrix representation for the 1M-point FFT is illustrated in figure 4.3. It consists of 19 rows and columns. The square in the right upper corner corresponds to the rotation W_{1M}^{1}, the squares on the diagonal $y - x = 16$ corresponds to the rotation W_{1M}^{2} and so on. The three first diagonals are marked with red lines in figure 4.3.

Storing the twiddle factors for each stage would require a lot of memory [10]. A better approach is to calculate the twiddle factors on the fly. Lets consider stage 4 which is represented in the triangular matrix as a 4×4 matrix at position $y = 4 \ldots 7$, $x = 1 \ldots 4$. According to equation 4.1, $\phi_4(I)$ is calculated as the sum of 16 power of 2 numbers. The elements belonging to column $y = 4$ in the triangular matrix results in different powers of 2 and can therefore be represented in a 4-bit vector with the outcome of the boolean equation $a_{x,y}(I) = b_{20-x}(I)$ AND $b_{19-y}(I)$ at position $18 + (x - y)$. The twiddle factor is then calculated as the sum of these vectors, as in figure 4.4

This can be done for all rotator stages, where the number of columns in the triangular matrix for a stage determines the number of vectors that should be added together and the number of rows is the individual vector length. For stage 12, this corresponds to 8 vectors of length 12.

4.4 Maximizing the SQNR by Adjusting the Signal Amplitude

To maximize the SQNR of the signal it is important to take into consideration how much the signal is scaled at each stage. Each butterfly adds one bit to the signal and each rotator first adds a number of bits and then truncates a number of bits according to table 4.2. The scaling $R_{scale,S}$ of a stage s due to truncation is defined as

$$R_{scale,s} = \frac{R}{2^{WL_{s+1}-WL_s}}$$

where R is the radius of the rotation carried out in that stage and WL_s is the input word length in stage s.

Figure 4.5 shows why it is important to adjust the signal amplitude. Let us consider a signal in the upper right corner of the red square that are about to be rotated with 45° in the counterclockwise direction. The red square represents

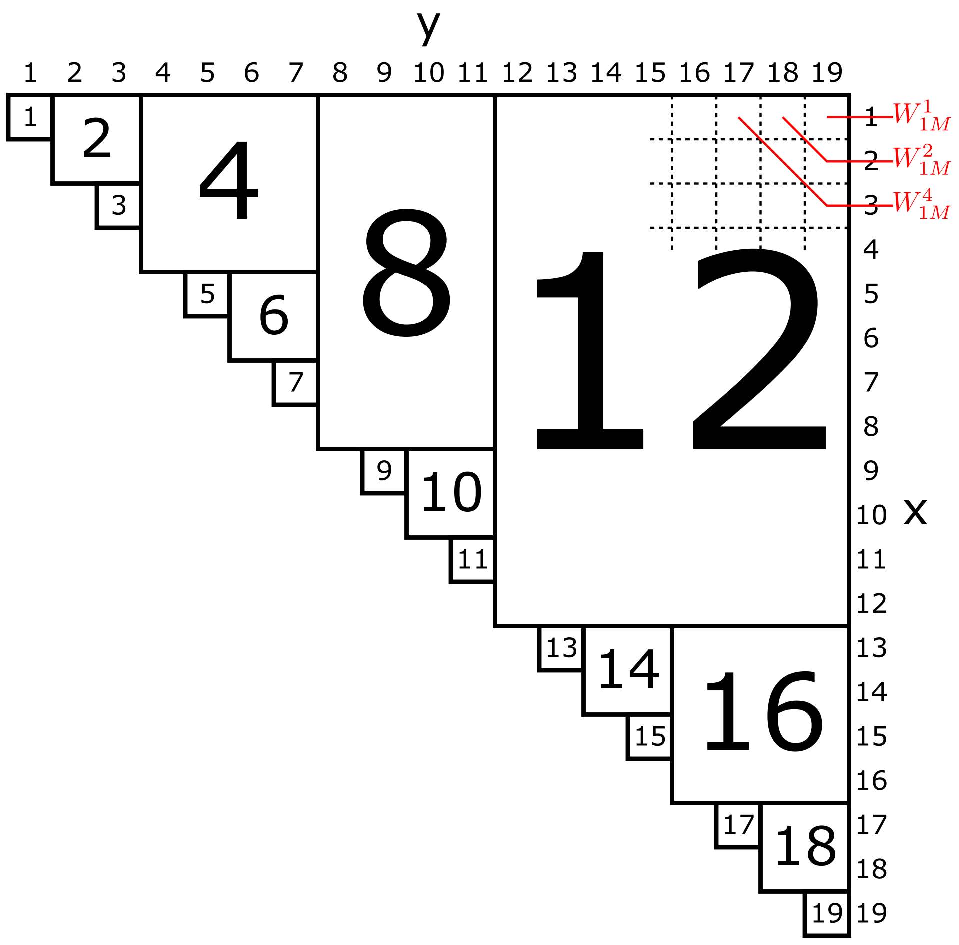


Figure 4.3: *The 1M-point FFT triangular matrix representation*

the set of all values that can be expressed with a given word length for a signed complex number. The rotated signal would end up outside this square which in hardware terms is analogous to an integer overflow. This can be prevented by adjusting the signal so that the maximum signal amplitude stays inside the black circle.

The amplitude ratio $R_{ar,s}$ at stage s, is a measurement of how many bits that are used in the output signal. It can be calculated as

$$R_{ar,s} = \frac{WL_1 \cdot \prod_{i=1}^{s} R_{scale,i}}{2^{WL_{s+1}}}$$

An amplitude ratio of 1 means that all the bits are used to a maximum and the rotation could cause an overflow. For a non-trivial rotator, the amplitude ratio should never exceed $\frac{1}{\sqrt{2}} > 0.7$. Referring to figure 4.5, this means that the signal

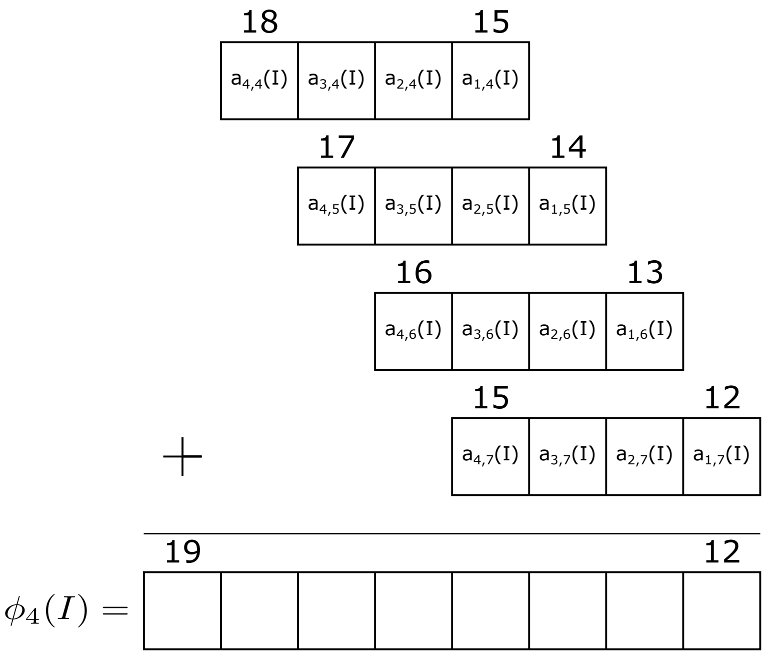

18
15
$a_{4,4}(I)$ $a_{3,4}(I)$ $a_{2,4}(I)$ $a_{1,4}(I)$
17
14
$a_{4,5}(I)$ $a_{3,5}(I)$ $a_{2,5}(I)$ $a_{1,5}(I)$
16
13
$a_{4,6}(I)$ $a_{3,6}(I)$ $a_{2,6}(I)$ $a_{1,6}(I)$
15
12
$a_{4,7}(I)$ $a_{3,7}(I)$ $a_{2,7}(I)$ $a_{1,7}(I)$
+
19
12
$\phi_4(I) =$

Figure 4.4: *Calculation of stage 4 twiddle factors*

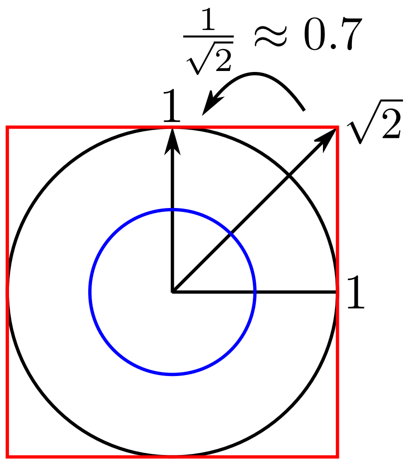


Figure 4.5: *A signal should stay in the area between the two circles. The square contains all values that can be expressed with a signed complex number with a given word length*

is inside the black circle. From table 4.2 it is clear that the amplitude ratio is 1 in the first stage, which is acceptable since stage 1 has a trivial rotator. For all the next stages the amplitude ratio never exceeds 0.7. The maximum input word length is 30, the reason for this is exlplained in section 6.2.

A low amplitude ratio will have detrimental effect on the SQNR of the signal. This can be prevented by truncating one MSB instead of one LSB when truncation should be done. This doubles the amplitude ratio and should always be done when the amplitude ratio is about to drop below $\frac{0.7}{2} = 0.35$. In figure 4.5, this means that the signal should always stay outside the blue circle.

Table 4.2: *Signal scaling. The red strike-through values indicate that the amplitude ratio has fallen below 0.35. To improve the SQNR of the signal, 1 MSB has been truncated instead of an LSB*

	Signal scaling			
Stage	Rotator size	Input word length	Truncated bits	Amplitude ratio
1	W_4	16	0	1.00
2	W_{16}	17	15	0.64
3	W_4	18	0	0.64
4	W_{256}	19	18	0.52
5	W_4	20	0	0.52
6	W_{16}	21	14 LSB, 1 MSB	0.67 ~~0.34~~
7	W_4	22	0	0.67
8	W_{4096}	23	18	0.41
9	W_4	24	0	0.41
10	W_{16}	25	14 LSB, 1 MSB	0.53 ~~0.27~~
11	W_4	26	0	0.53
12	W_{1M}	27	37 LSB, 1 MSB	0.53 ~~0.27~~
13	W_4	28	0	0.53
14	W_{16}	29	14 LSB, 1 MSB	0.68 ~~0.34~~
15	W_4	30	1	0.68
16	W_{256}	30	19	0.55
17	W_4	30	1	0.55
18	W_{16}	30	15 LSB, 1 MSB	0.70 ~~0.35~~
19	W_4	30	1	0.70
20	-	30	0	0.70

The radius of the W_{256} rotators in stage 4 and 16 and the W_{4096} rotator in stage 8 has been chosen to maximize the SQNR of the 1M-point FFT. The values has been obtained by first fixating the radius of the W_{256} rotators to some value $2^{17} < R_{W_{256}} < 2^{18}$ and then calculating

$$R_{4096} = \frac{2^{162}}{\sqrt{2} \cdot \prod_{s \neq 8} R_s}$$

For values of R_{4096} in the same range as $R_{W_{256}}$, the total scaling of all the rotators is in the order of 2^{162}, not taking any truncation into consideration.

The resulting amplitude ratio at the last stage then becomes 0.7 which is as large as possible without causing an overflow.

4.5 Design of the $\mathrm{W}_{1\mathrm{M}}$ Rotator

As discussed in section 3.4, a common strategy for large rotators is to use a series of multiple rotation stages to carry out the rotation. The W_{1M} rotator uses this approach. The design of the rotation stages and the control unit are discussed in the following sections below.

It is a good idea to have a trivial rotator as the first rotation stage since it is simple and does not introduce any error to the signal. Since the angle resolution of the W_{1M} rotator is really high, it is good to use a nanorotator architecture as the last rotation stage. The nanorotator makes use of simplifications that can be applied when the angles are really small, typically $\alpha_{in} < 1°$.

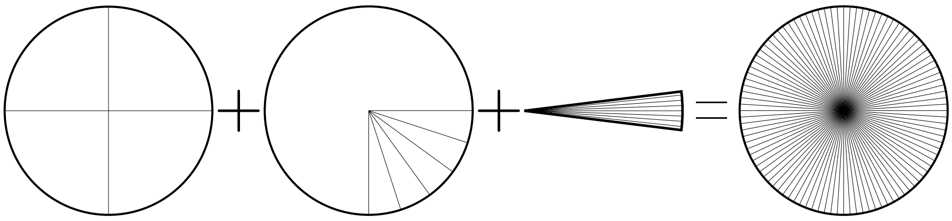


Figure 4.6: *The W_{1M} rotator uses multiple rotator stages*

4.5.1 The Trivial W_4 Rotator

A good choice as a first rotator stage is the trivial W_4 rotator. It is a simple rotator, hence the name, and performs four different rotations in 90° intervals, see figure 4.7. The rotator does not introduce any error to the signal and can be implemented with simple hardware.

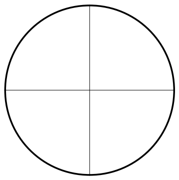


Figure 4.7: *The four rotation angles of the trivial W_4 rotator*

4.5.2 The Nanorotator

The nanorotator makes use of the small-angle approximation. Consider the coefficient set $P_k = C + jk$ where C is a constant, $k = 0, ..., N$ and $N \ll C$. The angle set is determined by $\alpha_k = \tan^{-1}\left(\frac{k}{C}\right)$. Since α_k is small the relation $\alpha_k \approx \tan(\alpha_k)$ holds and can be simplified to $\alpha_k \approx \frac{k}{C}$. This simplified rotation can be described as

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} C & -k \\ k & C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} .$$

Since C is constant, Cx and Cy can be calculated using shift and add techniques instead of using expensive general multipliers. The products $-ky$ and kx still requires general multipliers but no rotation coefficients need to be stored in memory.

The constant C can be calculated by considering the smallest rotation that the nanorotator can carry out. Refer to figure 4.8 and let C be the side adjacent to the angle α_{min} and the opposite side have a length of 1. Then C can be calculated as

$$\tan(\alpha_{min}) = \frac{1}{C} \Leftrightarrow C = \frac{1}{\tan\left(\frac{360}{2^{20}}\right)} = 166\,886 .$$

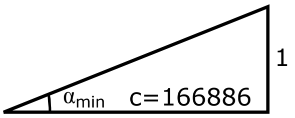


Figure 4.8: *The geometry of the nanorotator in the case of the smallest rotation angle*

The nature of the nanorotator makes it suitable as the last stage in the W_{1M} rotator. The error introduced by this rotator is generally considered to be acceptable when $\alpha_{in} < 1°$ [15] and this has to be taken into consideration when designing the rotator stage before the nanorotator.

The radius of the nanorotator is

$$R_{nano} \approx 166\,888 .$$

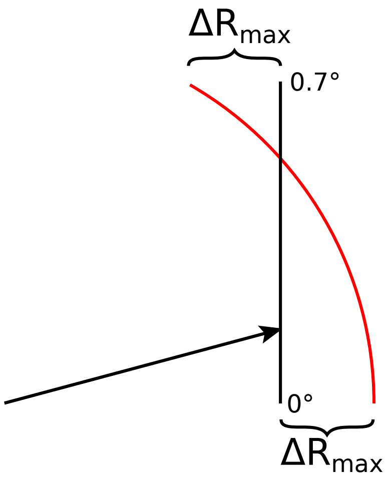
This value will be used to determine the total scaling factor of the W_{1M} rotator.

The rotation error ϵ_{nano} increases as α_{nano} increases. To keep the maximum α_{nano} small, the nanorotator uses both positive and negative angles, see figure 4.9. This reduces the maximum distance between the radius and the points $C + jS$. Rotations with negative angles are compensated by the complex multiplier rotator. This is explained in section 4.5.4.

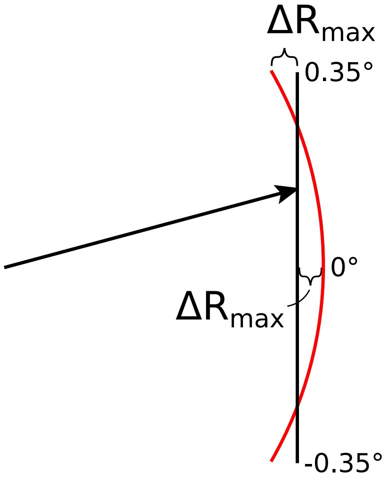
4.5.3 The W$_{512}$ Complex Multiplier Rotator

The second stage in the W_{1M} rotator works as a bridge between the trivial rotator and the nanorotator. The input angle is determined by the output of the trivial rotator and is $\alpha_{in} = 90°$. To achieve good accuracy in the nanorotator, the second stage is designed to have an output angle $\alpha_{out} < 1°$. A W_{512} rotator will have an angle resolution $\alpha_{min} \approx 0.70$ °, which is sufficient.

It is important to remember that this rotator is preceded by a trivial rotator. This means that the W_{512} rotator is only partial, rotating in a single quadrant of



(a) Only positive angles used. ΔR_{max} is large



(b) Use of both positive and negative angles. ΔR_{max} is small

Figure 4.9: *Illustration of use of only positive angles in (a) and both positive and negative angles in (b). The distance ΔR_{max} between the points $C + jS$ (black line) and the radius (red curve) is smaller in (b)*

the complex plane, but with the same resolution as a full W_{512} rotator. The resolution of $\frac{360°}{512}$ yields 128 angles in a quadrant. The rotator is however designed using 129 angles, where $\alpha \in [0, ..., \frac{\pi}{2}]$. The reason behind this is that the nanorotator operates in both a positive and negative range. This will be explained more in detail in section 4.5.4.

Since the W_{512} rotator is a medium-sized rotator, it is a good choice to implement it as a complex multiplier rotator. The coefficients are stored in memory, the principle for this is illustrated in figure 4.10.

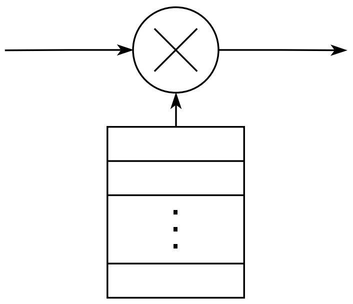


Figure 4.10: *General complex multiplier with a coefficient memory.*

This architecture is straight-forward and easy to implement. Furthermore, the memory is not very large, especially considering that the rotation coefficients can be reused. This will be discussed later in this section.

The coefficients C and S are obtained through

$$\begin{aligned} C &= [R_{CM}^T \cdot \cos(\alpha_{CM})], \\ S &= [R_{CM}^T \cdot \sin(\alpha_{CM})], \end{aligned}$$

where R_{CM}^T is the target radius and α_{CM} represents all 129 angles in the angle set $\{0, \frac{2\pi}{512}, \frac{2\cdot 2\pi}{512}, ..., \frac{126\cdot 2\pi}{512}, \frac{127\cdot 2\pi}{512}, \frac{\pi}{2}\}$. The symbol $[\cdot]$ denotes a rounding operation. The target radius is determined by first setting the total scaling factor of the W_{1M} rotator, $R_{W_{1M}}$, to equal a factor of two which will make W_{1M} have unity-gain, i.e.,

$$R_{W_{1M}} = R_{CM}^T \cdot R_{Nano} = 2^k,$$

where k is a positive integer. The trivial W_4 rotator does not scale the signal. Having unity-gain enables easy down-scaling of the output by simply shifting the signal. Hence, R_{CM}^T is obtained from

$$R_{CM}^T = \frac{2^k}{R_{Nano}}.$$

Figure 4.11 shows that the rotation error of the W_{1M} rotator, $\epsilon_{W_{1M}}$, does not improve much for $k > 37$. Choosing $k = 37$, R_{CM}^T can be calculated to

$$\left.\begin{aligned} k &= 37 \\ R_{nano} &\approx 166888 \end{aligned}\right\} \implies R_{CM}^T \approx 823542,$$

which corresponds to a coefficient word length of $WL_C = \left\lceil \log_2 R_{CM}^T \right\rceil = 20$ bits, where $\lceil\cdot\rceil$ denotes a ceiling operation. This is the unsigned representation, but since the multiplications are signed the coefficients will use a 21-bit two's complement representation.

Once the coefficients are retrieved, the radius R_{CM} of the rotator can be calculated according to section 2.3.2. This value is approximately equal to R_{CM}^T.

Due to symmetry in the complex plane, not all coefficients need to be stored. Figure 4.12 illustrate this. Here, $C' + jS'$ represents the coefficients that are actually fed into the multipliers, and $C + jS$ are the values stored in memory. The only unique coefficient values is found in the first half of a quadrant, where $\alpha \in [0, \ldots, \frac{\pi}{4}]$. This means that out of 129 different angles, only 65 angle coefficients need to be stored in memory. The rest of the angles are obtained by swapping the values of C and S, see figure 4.12.

The W_{1M} rotator is summarized in table 4.3.

4.5.4 Control Unit

The rotation angle $\alpha_{W_{1M}}$ of the W_{1M} rotator has to be translated to the rotation angles α_{W_4}, $\alpha_{W_{512}}$, α_{nano} for each rotator stage to fulfill

$$\alpha_{w_{1M}} = \alpha_{W_4} + \alpha_{W_{512}} + \alpha_{nano},$$

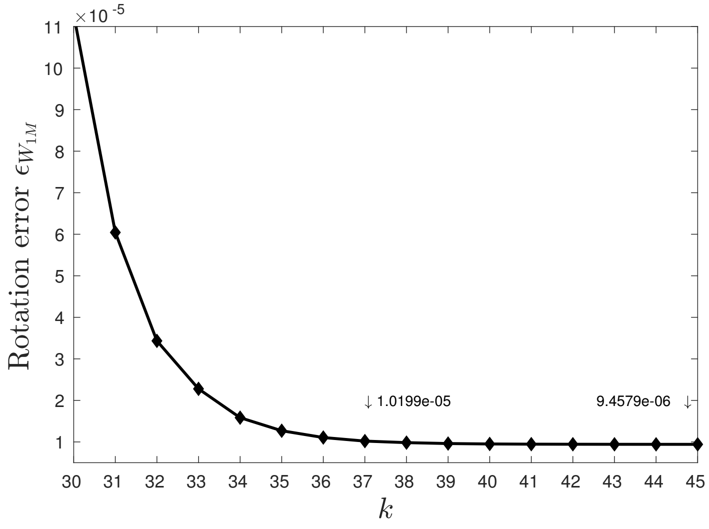


Figure 4.11: *Rotation error as a function of the number of output bits*

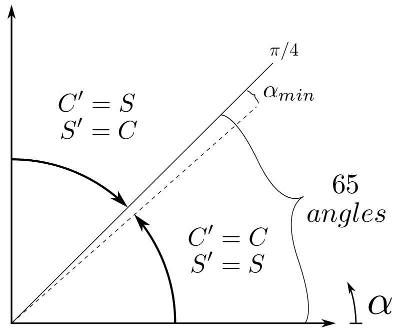


Figure 4.12: *Illustration of the coefficient symmetry in* W_{512}.

where

$$\begin{cases} \alpha_{W_4} &= 45^\circ \cdot i, & i \in \{0,1,2,3\} \\ \alpha_{W_{512}} &= 0.703^\circ \cdot j, & j \in \{0,1,2,\ldots,128\} \\ \alpha_{nano} &= \frac{360^\circ}{2^{20}} \cdot k, & k \in \{-1023,-1022,\ldots,1024\} \end{cases}$$

and i, j, k are the control signals to the rotator stages. The negative angles in the nanorotator are discussed in section 4.5.2. The rotator stage angles are

$$\begin{cases} \alpha_{W_4} &= \left\lfloor \frac{\alpha_{w_{1M}}}{45^\circ} \right\rfloor \\ \alpha_{W_{512}} &= \left[\frac{\alpha_{w_{1M}}}{360^\circ/512} \right] \\ \alpha_{nano} &= \alpha_{W_{1M}} - \left[\frac{\alpha_{W_{1M}}}{360^\circ/512} \right] \times \frac{360^\circ}{512} \end{cases}$$

where $\lfloor \cdot \rfloor$ and $[\cdot]$ denotes floor and rounding operations respectively.

***Table 4.3:** Summary of the W_{1M} rotator*

Rotator	Architecture	Rotation error ϵ	Radius
W_4	Trivial	0	1
W_{512}	Complex multiplier	7.87×10^{-7}	823 542
Nano	Nano	9.41×10^{-6}	166 888
W_{1M}	Cascaded rotator stages	1.02×10^{-5}	2^{37}

4.6 Design of the W_{16} CCSSI Rotators

For rotators with small angle sets, such as the W_{16} rotator, a complex multiplier rotator, although easy to implement, would be an overly complex implementation with its four real multipliers. A better approach for small angle sets is to use a multiplierless rotator that uses shifters and adders to implement the multiplications.

Table 7 in [19] contains coefficients chosen wisely to minimize rotation error and use of resources. The coefficients used in the W_{16} rotator were chosen from the table in such a way to have a rotation error in the same order of magnitude as the rotation error in the W_{1M} rotator. The coefficients are repeated here in table 4.4 for convenience.

Table 4.4

	Coefficients used in the W_{16} rotator		
	0°	22.5°	45°
C	21 059	19 456	14 891
S	0	8 059	14 891

The four products are calculated by progressively shifting and adding intermediate results. Figure 4.13 shows an example for a signal x multiplied by 827. The rotation error is given by table 7 and is $\epsilon_{W16} = 2.67 \times 10^{-5}$.

4.7 Design of the W_{256} and W_{4096} Complex Multiplier Rotators

The W_{256} and W_{4096} rotators are not as small as the W_{16} rotator, and not as large as the W_{1M} rotator. This makes the complex multiplier architecture suitable for these two rotators. The rotators are similar to the W_{512} rotator stage in the W_{1M} rotator. The difference is that the W_{256} and W_{4096} rotators carry out rotations in the full 360° range, since they are not preceded by a trivial rotator. However, only coefficients for the range $\alpha = [0°, \ldots, 45°]$ need to be stored because of rotation symmetry. It can be said that the W_{256} and W_{4096} rotators have trivial rotators

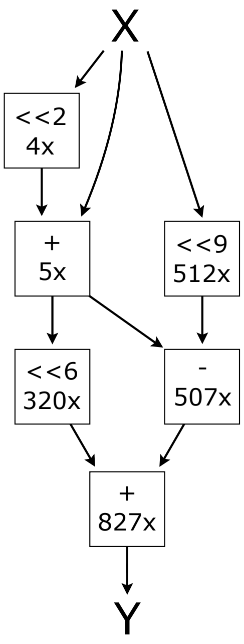


Figure 4.13: *Shift and add constant multiplication*

built-in to them. The radius of the two rotators has been tweaked to maximize the SQNR of the signal. This is explained in section 4.4.

What differs from W_{512} is that these rotators use all angles in the interval $[0, 2\pi)$, which makes the importance of coefficient re-usability more apparent in terms of memory saving.

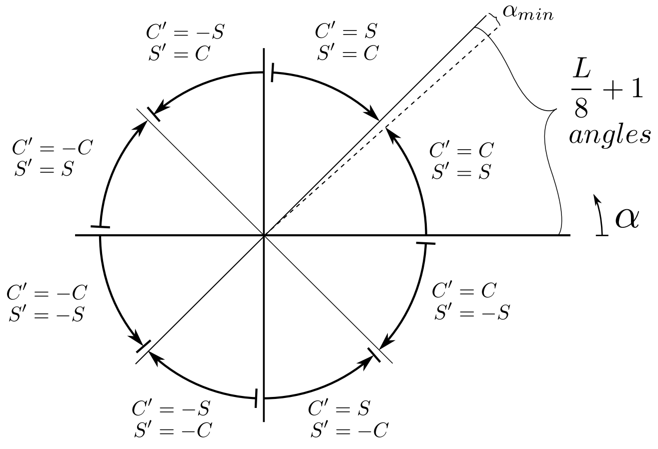


Figure 4.14: *Illustration of the coefficient symmetry used in W_{256} and W_{4096}.*

Figure 4.14 shows that, as for the W_{512} rotator, only half a quadrant consists of all unique coefficient values needed to express all rotation angles in the complex plane. For a W_L rotator the number of coefficients that needs to be stored is $\frac{L}{8}+1$. Hence the total amount of memory, expressed in bits, that is used for storing the coefficients equals

$$mem_{coeff} = 2\left(\frac{L}{8}+1\right)WL_C\,, \tag{4.2}$$

where WL_C is the number of bits used to represent C and S. Table 4.5 shows the memory usage of W_{256} and W_{4096}.

Table 4.5

	Memory usage (bits)	
L	All angles	Coefficients reused
256	$512 \cdot WL_C$	$66 \cdot WL_C$
4096	$8192 \cdot WL_C$	$1026 \cdot WL_C$

5 Implementation

This chapter presents the hardware implementation of the one million-point FFT building blocks along with the FFT itself discussed in chapter 4. All modules are described in VHDL and their functionality verified through simulations using ModelSim SE-64 6.4a from Mentor Graphics. Matlab R2016b is used to generate input stimuli and to verify the simulation output.

The different modules will be discussed in the order they have been implemented. First, the implementation of the W_{1M} rotator will be discussed, followed by the other rotators. Then the implementation of the FFT stage and FFT control unit will be explained. Lastly, everything is put together in the FFT top module.

5.1 Target Device

Although the SDF architecture is resource-efficient, the size of the 1M-point FFT makes demands on the target device where the design is going to be implemented. The feedback FIFO buffers use a lot of memory, which has to be accessed with no more than one clock cycle of delay. This means that the buffers needs to use a lot of on-chip memory. The memory used by the buffers can be estimated to

$$2\sum_{s=1}^{20}\left(2^{20-s} \times (16+s)\right) \approx 34 \text{ Mbit}$$

assuming 16 bit inputs and that the signal is allowed to grow by 1 bit every stage. This does not include the memory used by some rotators to store coefficients. However, rotator memories are comparatively small. Memory capacity was the main factor when choosing the target device.

The task fell on the mighty Xilinx Virtex Ultrascale evaluation board with the xcvu095-ffvd1924-2-e-es1 FPGA. Some device specific coding techniques has been used when implementing the design. They are described in each section.

5.2 Implementation of the W_{1M} Rotator

The architecture of the W_{1M} rotator is illustrated in figure 5.1. It consists of the three rotator stages and the control unit.

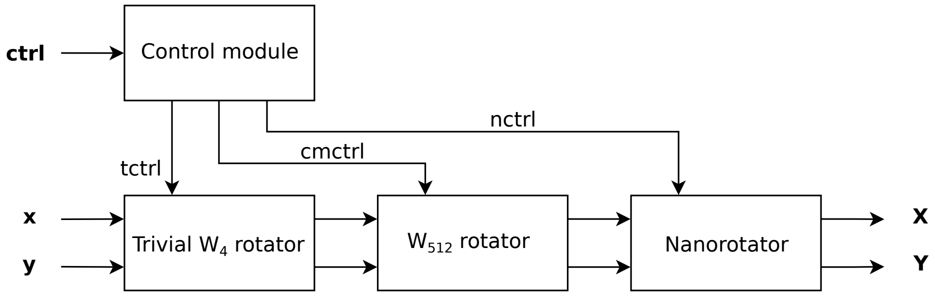
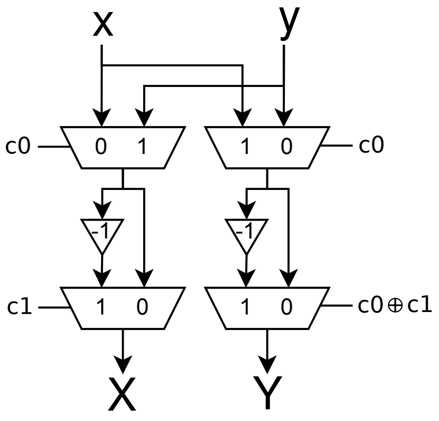


Figure 5.1: *Hardware architecture of the W_{1M} rotator.*

5.2.1 The Trivial W_4 Rotator

The implementation of the trivial rotator is rather straightforward since the operation only performs a swap and/or negation on the real and imaginary parts of the input. Figure 5.2 depicts the hardware architecture as implemented, consisting of four 2-input multiplexers and two negators.



Control table

c1	c0	$X + jY$	α
0	0	$x + jy$	0°
0	1	$y - jx$	270°
1	0	$-x - jy$	180°
1	1	$-y + jx$	90°

Figure 5.2: *Hardware architecture and control table of the trivial rotator.*

5.2.2 The Complex Multiplier Rotator

The partial W_{512} rotator used in the middle stage of the W_{1M} rotator is implemented as a standard complex multiplier, seen in figure 5.3a. This is the easiest implementation directly derived from the calculation in (2.17). The memory unit where the coefficients are stored is illustrated in figure 5.3b.

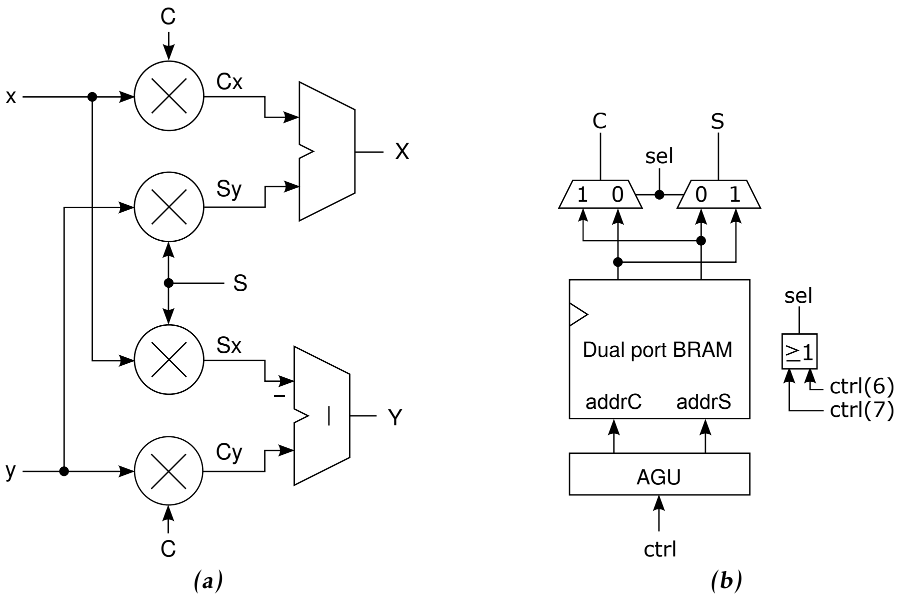


Figure 5.3: *Architectures of the W_{512} complex multiplier in (a), and its coefficient memory unit in (b).*

The architecture requires four real multipliers, two adders and a memory to store the complex coefficients $C_i + jS_i$, $i = 0, 1, ..., 128$. The memory is implemented with a dual port read-only functionality using block-RAM (BRAM). C and S are stored in an order which follows $\alpha : 0 \rightarrow \frac{\pi}{4}$. All the values of C are put first followed by all values of S. This is illustrated in figure 5.4. An 8-bit control signal is used to generate the memory address. The address generation unit (AGU) is illustrated in 5.5.

5.2.3 Nanorotator

Figure 5.6 illustrates the hardware architecture of the nanorotator. The architecture consists of two real multipliers, two adders and two scale units which perform a constant multiplication by the factor 166886. No coefficient memory is required since k is the two's complement interpretation of the control signal. Figure 5.8 illustrates the principle of shift-and-add computation, but with the values used in the W_{16} rotator.

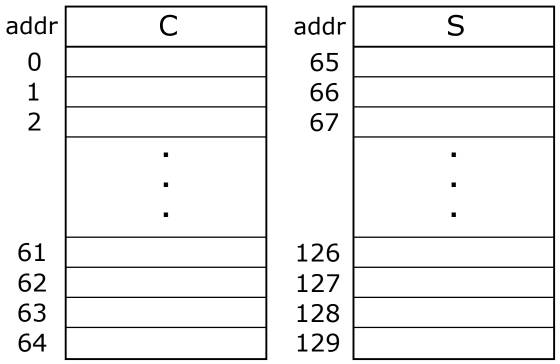


Figure 5.4: *W_{512} coefficient storage order.*

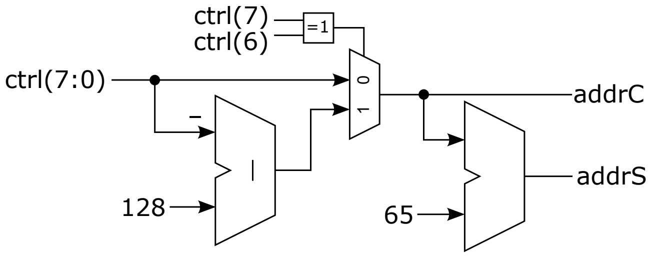


Figure 5.5: *W_{512} address generation unit.*

5.3 Implementation of the W_{16} Rotator

The implemented architecture of the W_{16} rotator is illustrated in figure 5.7. The architecture is hardware-efficient as it does not require any general multipliers or memory. The submodules scale C and scale S carry out constant multiplications and are implemented with adders. Pipeline registers added to the submodules are not shown in the figures. The architecture of the scale C submodule is found in figure 5.8.

5.4 Implementation off the W_{256} and W_{4096} Complex Multiplier Rotators

In the same way as the W_{512} rotator, the W_{256} and W_{4096} rotators are implemented as general complex multipliers by using the architecture depicted in figure 5.3a. Since these rotators cover the full 360°range, the memory unit and its AGU have slightly different hardware implementations compared to the complex multiplier rotator stage in the W_{1M} rotator.

Figure 5.9 illustrates the memory unit of both rotators, where a negation function of the memory output is added. The swapping of C and S is incorporated in the AGU, seen in figure 5.10. The order of the coefficients in the memory follows

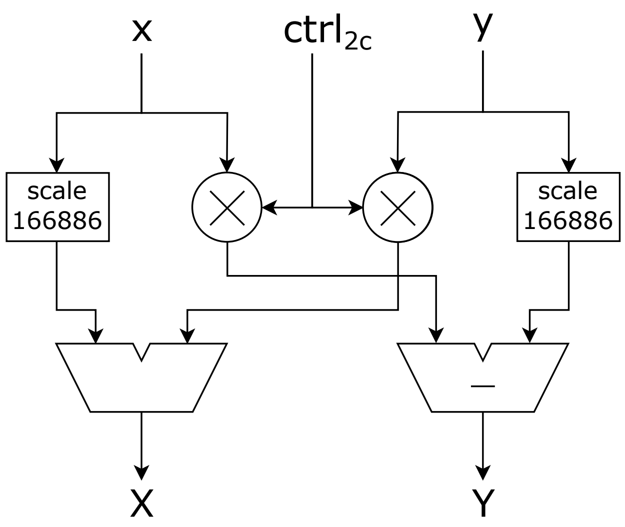


Figure 5.6: *Hardware architecture of the nanorotator.*

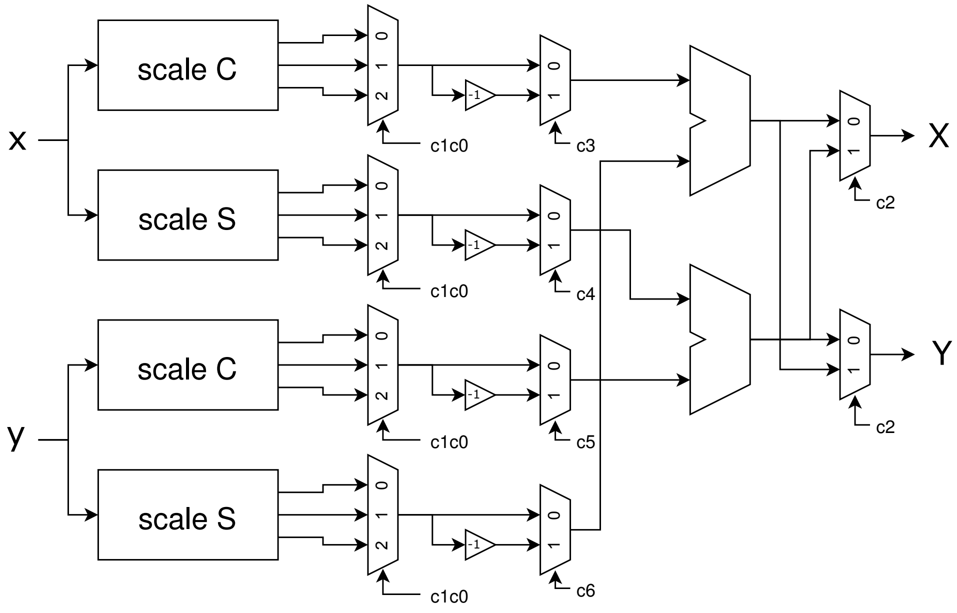


Figure 5.7: *W_{16} rotator architecture.*

the same principle as for the W_{512} rotator.

For this implementation, the size of the control signal for a W_L rotator is $\lceil \log_2 L \rceil$ bits, since it divides the circumference into L number of angles. The 3 most significant bits of the control signal will be denoted ctrl(MSB). This is independent of the rotator size, and divides the circumference into 8 equally sized

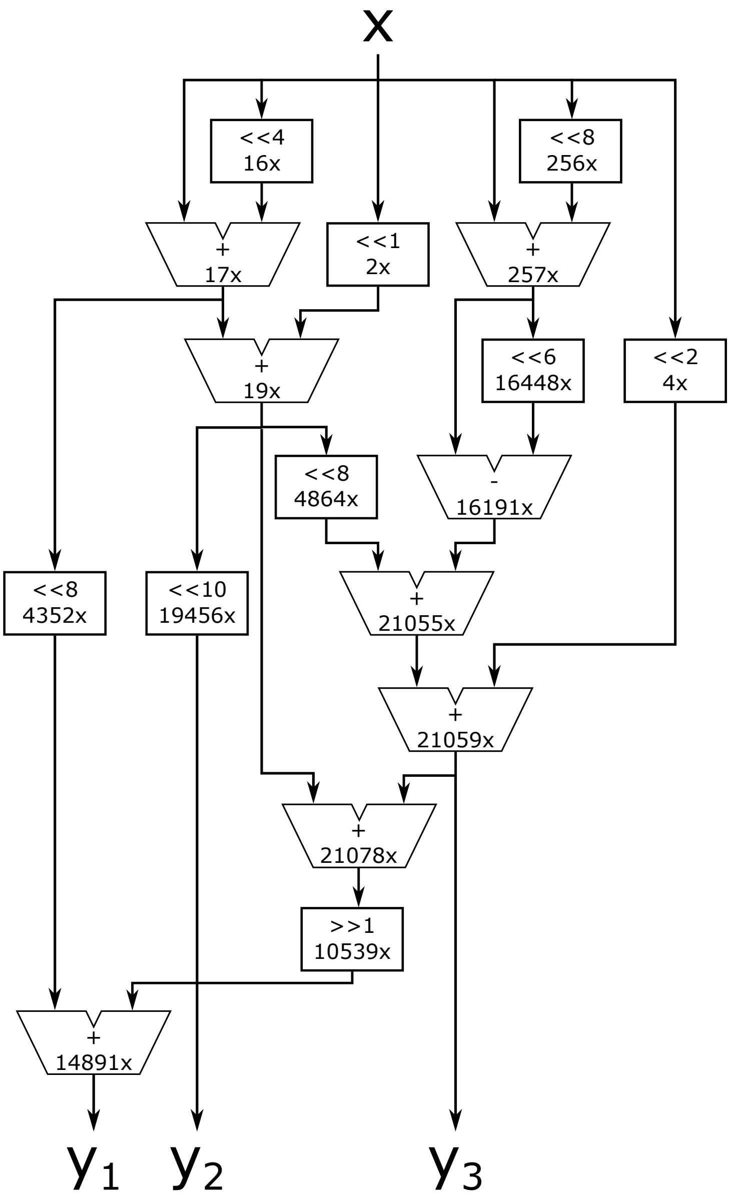


Figure 5.8: *Architecture of the scale C submodule.*

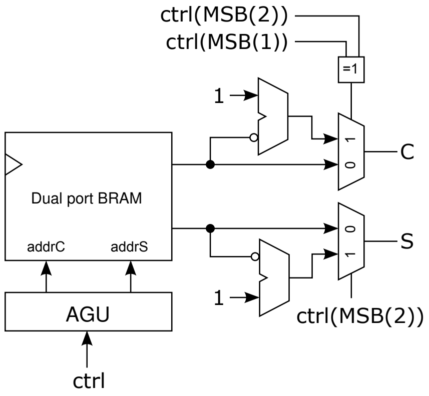


Figure 5.9: *Memory unit of the W_{256} and W_{4096} rotators.*

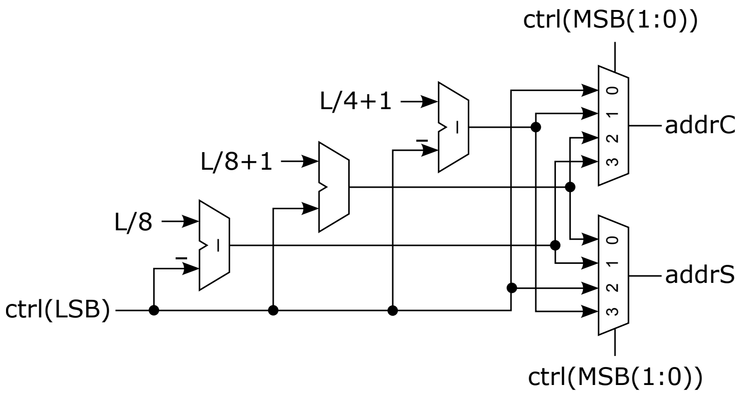


Figure 5.10: *AGU for the W_{256} and W_{4096} memory units.*

areas, seen in figure 4.14, and has to do with the coefficient symmetry. ctrl(MSB) decides whether the coefficients need to swap and/or be negated. The rest of the control signal, the least significant part, is denoted ctrl(LSB), and point to a certain angle in an area. For W_{256} and W_{4096}, the control signal sizes are 8 bits and 12 bits, respectively.

5.5 Implementation of the SDF stage

The architecture of the SDF stage is illustrated in figure 5.11. The size of the buffer and the type of rotator will vary depending on the stage number. The muxes are controlled with the buffSel signal from the FFT control unit.

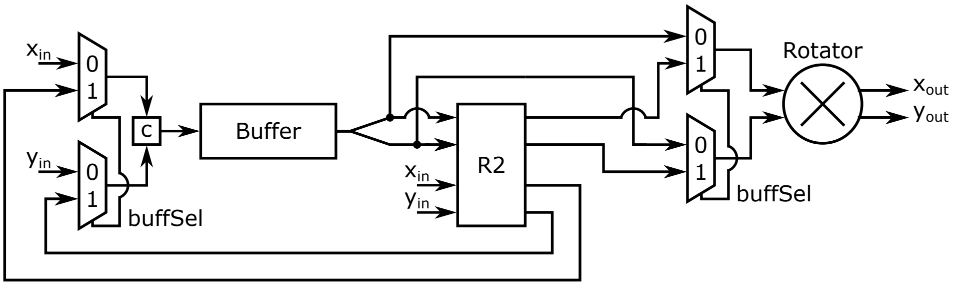


Figure 5.11: *The architecture of the SDF stage*

5.6 Implementation of the FFT Control Unit

An FFT control unit has been implemented to control the rotators and the muxes leading to the buffers. The architecture of the top module of the FFT control unit is depicted in figure 5.12.

The pipe counter module keeps track of when the rotCtrl module should start each counter used for generating twiddle factors. It also keeps track of when the buffCtrl module should start its counters used to control the muxes before the buffers. The pipe counter module is illustrated in figure 5.13.

The buffCtrl submodule consists of 20 counters, one for each buffer. One of the counters for stage S is shown in figure 5.14. The rotCtrl submodule consists of 19 counters, one for each rotator. The value of each counter refers to the binary index number $b(I)$ of the SFG, see figure 2.1. The twiddle factor is generated in a submodule from the binary index number. The counter and phigen submodule for the W_{4096} rotator are illustrated in figure 5.15 and the phigen submodule for the W_{4096} rotator is illustrated in figure 5.16.

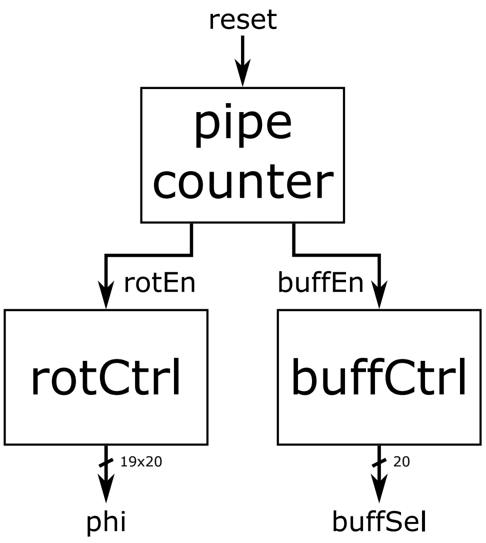


Figure 5.12: *FFT control unit top module*

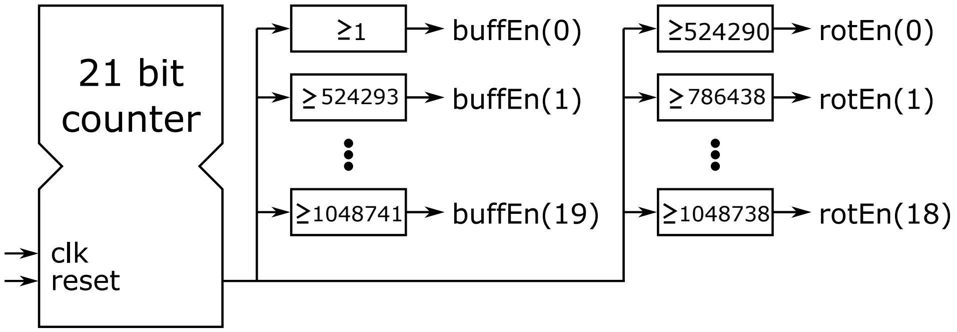


Figure 5.13: *The architecture of the pipe counter submodule*

5.7 Implementation of the FFT Top Module

The implemented architecture of the FFT top module is illustrated in figure 5.17. It consists of the FFT control unit, all 19 rotators and all 20 SDF stages connected together. The calculated FFT appears in bit-reversed order on the output.

5.8 Test Methodology

The functionality of every submodule has been verified with test benches. The overall strategy is to generate long vectors of input signals, simulating the system using ModelSim, and then comparing the result with the expected behavior in Matlab. The rotators have been tested by varying the input signals and rotation angles then calculating the error ζ_{max}, see section 2.3.

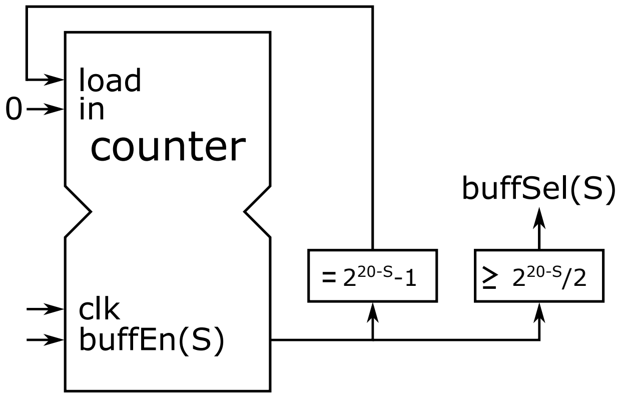


Figure 5.14: *The buffer counter for stage S in the buffCtrl submodule*

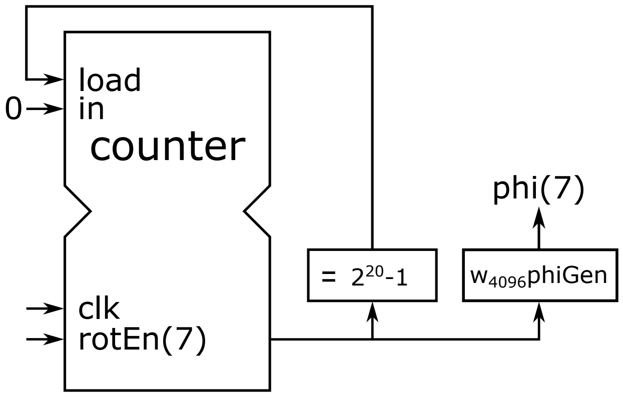


Figure 5.15: *The index counter for the W_{4096} rotator in the rotCtrl submodule*

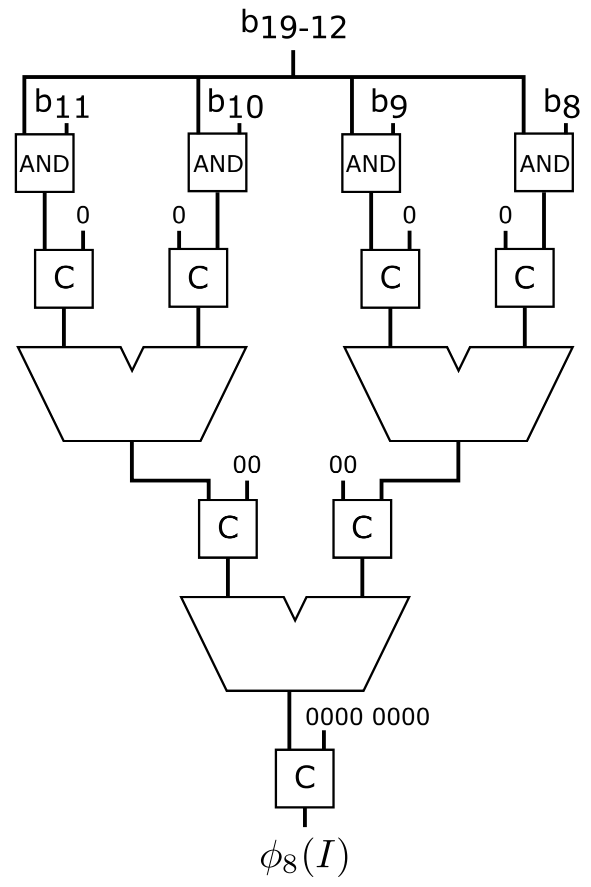


Figure 5.16: *The architecture for generating twiddle factors to the W_{4096} rotator. The boxes marked with C represents a concatenation operation.*

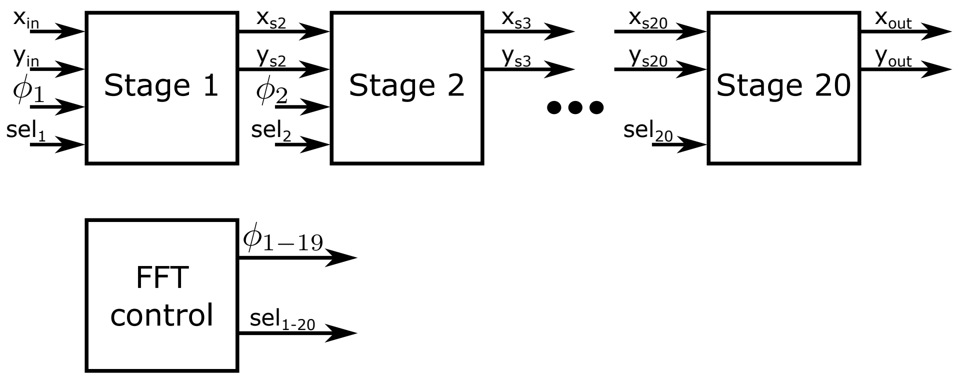


Figure 5.17: *FFT top module*

Running a lot of hardware simulations is very time-consuming. For this reason, a model of the 1M-point FFT has been developed in Matlab. Although slower than Matlab's built-in FFT by the Matlab model generates exactly the same results as the hardware FFT. This has been verified by comparing it to the simulated hardware FFT, running 2 consecutive FFT cycles on 10 different computers.

The rest of the testing has been performed on the Matlab model. To verify that the hardware actually performs an FFT calculation, it has been tested by stimulating the input with different sinusoids, one for each FFT cycle, and checking that the output corresponds to the expected behavior. It would take a lot of time (over a month!) to test the Matlab model with all 2^{20} possible harmonics. Instead, the first, last, and middle 128 harmonics, and a couple of thousand arbitrary harmonics in-between, have been tested.

The last test calculates the SQNR of the signal to make sure that the hardware produces a strong signal with low quantization noise. The result from this test is described further in section 6.2.

6 Experimental Results

This chapter presents various results of the 1M-point FFT. It includes W_{1M} rotator error measurements, FFT performance results and extracts from the Vivado place and route report.

6.1 W_{1M} Rotator Error

As mentioned in section 2.3, the accuracy of the W_{1M} rotator can be measured using ϵ, which is the distance between where the rotated signal ended up and where it should be in the ideal case. When determining ϵ_{max}, a lot of input signals are randomized and used together with all rotation angles in the angle set.

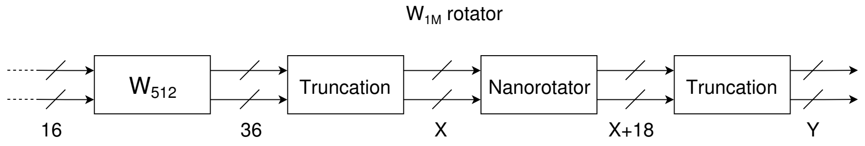


Figure 6.1: *W_{1M} truncation setup*

To determine the bit growth throughout the FFT it is important to know the error characteristics of the W_{1M} rotator when the signal is truncated. The W_{512} rotator and the nanorotator both scale the signal, which will increase the number of bits. According to the setup illustrated in figure 6.1, the maximum total error, ϵ_{max}, is calculated for different output word lengths from these two rotators. During this simulation, the input word length of the W_{1m} rotator has been set to 16 bits instead of the 27 used in the FFT architecture. The result is shown in figure 6.2. The inputs are randomized 16-bit numbers in the range $[-2^{15}, -1]$. This will capture only cases for which all bits are used. 1000 different input samples is evaluated for every rotation angle and the largest error is extracted from each truncation setting.

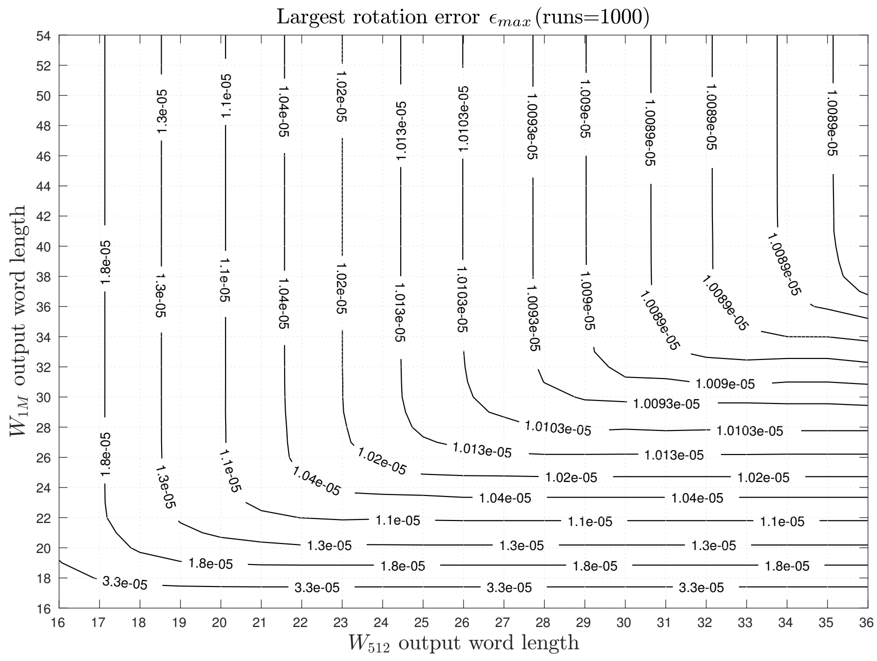


Figure 6.2: *Error estimation of the W_{1M} rotator for different truncation settings*

The result indicates that the truncation should occur after the nanorotator and not after the W_{512} rotator. Therefore, the output from the W_{512} rotator is kept as it is. In the case of the nanorotator, the output can be truncated significantly without losing much of the precision. The smallest error measured in the current design is

$$\epsilon_{max} \approx 1.0089 \cdot 10^{-5}.$$

6.2 FFT Performance Results

The signal to quantization noise ratio (SQNR) is the relationship between the output signal strength and the quantization error caused by the non-ideal FFT. It is measured in decibels and a higher value means that the FFT is more accurate.

Figure 6.3 illustrates the trade-off between the SQNR and the amount of used BRAM when increasing the output word length by one bit every stage up to a maximum number of bits. These calculations has been done with the Matlab model. A higher SQNR demands for using more bits, which will increase the amount of used BRAM.

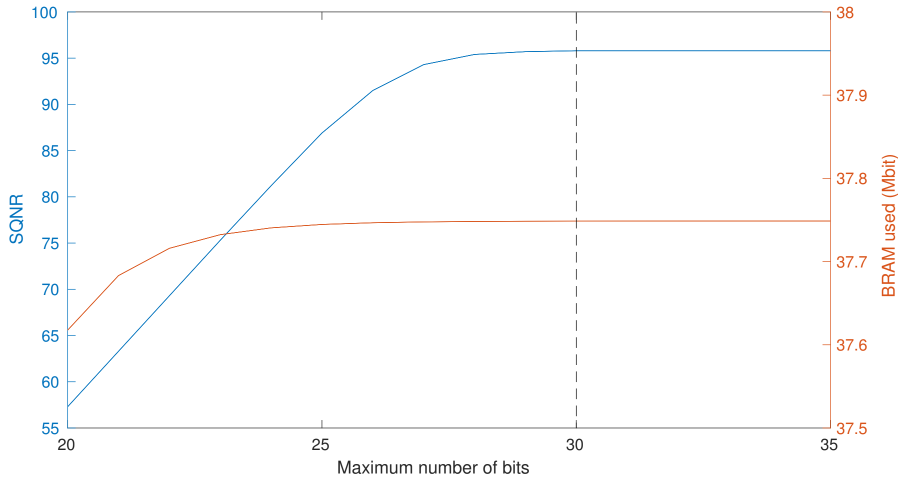


Figure 6.3: *SQNR vs amount of used BRAM*

Using a maximum number of more than 30 bits would not improve the SQNR or decrease the amount of used BRAM.

With 30 bits, the SQNR is

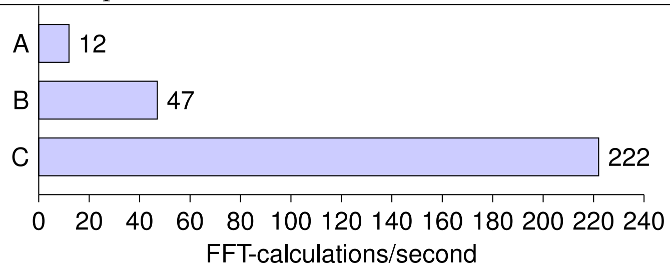
$$\text{SQNR} = 95.6 \text{ dB}$$

This estimation has been made by comparing the 1M-point FFT with the Matlab FFT. The throughput of the 1M-point FFT is 1 sample every clock cycle. Table 6.1 compares the performance of the 1M-point FFT with Matlab's FFT, running on two different computers. It can be said that the 1M-point FFT hardware implementation approximately equals the performance of 20 CPU-cores when calculating the FFT.

Table 6.1

FFT Performance Comparison

A:	Matlab, Xeon E5-1607, 2 cores
B:	Matlab, Xeon E5-E5620, 4 cores
C:	1M-point FFT @ 233 MHz



6.3 Synthesis Results

The 1M-point FFT has been implemented on the Xilinx Virtex Ultrascale VCU107 Evaluation Board which hosts the xcvu095-ffvd1924-2-e-es1 FPGA. En extract from the Xilinx Vivado 2014.3.1 place and route report can be found in table 6.2. The report shows that the implementation mostly uses a lot of BRAMs, but there is still space available on the FPGA. The large FIFOs in the first stages are the main factor that holds back the clock frequency as the critical path between the FIFO outputs and the input of the following butterflies is long.

Table 6.2

Place and Route Summary
Device: xcvu095-ffvd1924-2-e-es1

Utilization			
Resource	Used	Available	Utilization %
Flip Flops	30 299	1 075 200	3
LUTs	16 827	537 600	3
I/O	96	832	12
BRAMs	1 157	1 728	67
DSPs	30	768	4
Clock Frequency	233 MHz		
On-chip Power	3.436 W		
Junction Temperature	27.8 °C		

7 Conclusions

In this thesis, a one million-point FFT with a single-delay feedback architecture has been implemented on a Xilinx Virtex Ultrascale FPGA. The single-delay feedback architecture was chosen as it offers high throughput and efficient resource utilization. The architecture consists of 20 stages with butterflies, FIFO buffers and rotators.

The FFT architecture makes use of trivial-, CCSSI-, complex multiplier- and staged rotators. The staged rotator is an W_{1M} rotator with a trivial-, complex multiplier- and nanorotator stage. The FFT algorithm has been chosen to minimize the number of large rotators and to place the largest W_{1M} rotator late in the architecture to minimize FIFO buffer memory utilization. The twiddle factors for the rotators are calculated on-demand. This removes the need for large memories to store the twiddle factors.

The million-point FFT has high accuracy with an SQNR of 95.6 dB. The system is able to process 233 Msamples/s, which is equal to 222 full FFT calculations/s.

8 Future Work

The 1M-point FFT can be further developed in various ways to improve accuracy, throughput and clock frequency.

The SDF-architecture can be changed to a parallel one, improving throughput with an implementation area-trade-off. Several parallel architectures exists and some are mentioned briefly in section 3.2.

The design can to a greater extent be adapted to the target device to improve the clock frequency. The large FIFO buffers can be better designed to reduce the critical path. The outputs of the FIFOs has registers added to them, however, it is unclear how many registers that are required to obtain the highest clock frequency. By gaining deeper knowledge about the target device and the synthesis and implementation tools, a higher clock frequency might be achievable.

The complex multiplier rotators can be changed to more resource-efficient rotators. However, the place and route report clearly shows that there is a lot of resources still available on the FPGA, and not many DSPs has been used. There is no clear benefit in reducing anything other than the amount of used BRAMs.

The word length of each rotator and buffer can be changed easily as they are implemented using generics. This can either improve the accuracy or reduce the implementation area.

9 Contributions

This chapter will breifly explain the various tasks that have been carried out during the thesis, and how the workload of these has been distributed within the group.

In the beginning of the thesis, Hans started working on the trivial rotator and the nanorotator, while Tobias got busy designing the W_{512} rotator stage. After the rotator stage had been implemented, the same person took responsibility to thoroughly test the functionality of the rotator stage in a test bench. Hans has also been responsible for implementing the control module of the W_{1M} rotator, making sure that the W_{512} rotator and nanorotator worked in the expected way. Tobias has calculated the truncation characteristics of the W_{1M} rotator to investigate where truncation should occur in the design.

Tobias has been responsible for both the W_{256} and the W_{4096} rotators whilst Hans has been working with the W_{16} rotator. The stage architecture has mainly been developed by Hans, but Tobias has also been working in this field. The FFT control signal generation to the buffers and the rotators has been developed by Hans, the twiddle factor generation has also been part of this area. Tobias has been working a lot with integration of the top module.

The Matlab model has been developed in the same way as the hardware architecture, with the same responsibility areas.

The testing of the complete system and the work done in Vivado has been carried out equally on both parts.

Overall one can say that both participants have been working interchangeably with the thesis. The work has been characterized by the fact that both parties mainly focused on different areas, while both had a good understanding and influence over the work of the other.

Bibliography

[1] Omid Abari, Ezz Hamed, Haitham Hassanieh, Abhinav Agarwal, Dina Katabi, Anantha P. Chandrakasan, and Vladimir Stojanovic. A 0.75-million-point fourier-tranform chip for frequency-sparse signals. *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 458 – 460, Feb 2014. Cited on page 16.

[2] A. Agarwal, H. Hassanieh, O. Abari, E. Hamed, D. Katabi, and Arvind. High-throughput implementation of a million-point sparse Fourier transform. pages 1–6, September 2014. Cited on pages 1 and 16.

[3] Y.-N. Chang. An efficient VLSI architecture for normal I/O order pipeline FFT design. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 55(12):1234–1238, December 2008. Cited on page 15.

[4] Yun-Nan Chang. Design of an 8192-point sequential I/O FFT chip. In *Proc. World Congress Eng.Comp. Science*, volume II, October 2012. Cited on page 15.

[5] Chao Cheng and Keshab K. Parhi. High-throughput VLSI architecture for FFT computation. *IEEE Transactions on Circuits and Systems—Part II: Express Briefs*, 54(10):863–867, October 2007. Cited on page 15.

[6] Sang-In Cho, Kyu-Min Kang, and Sang-Sung Choi. Implemention of 128-point fast Fourier transform processor for UWB systems. In *Proc. Int. Wireless Comm. Mobile Comp. Conf.*, pages 210–213, August 2008. Cited on page 15.

[7] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, 19:297–301, April 1965. Cited on pages 1 and 4.

[8] A. Cortés, I. Vélez, and J. F. Sevillano. Radix r^k FFTs: Matricial representation and SDC/SDF pipeline implementation. *IEEE Transactions on Signal Processing*, 57(7):2824–2839, July 2009. Cited on page 15.

[9] A. M. Despain. Fourier transform computers using CORDIC iterations. *IEEE Transactions on Computers*, C-23:993–1001, October 1974. Cited on page 15.

[10] M. Garrido and J. Grajal. Efficient memoryless CORDIC for FFT computation. volume 2, pages 113–116, April 2007. Cited on pages 1, 16, 17, and 23.

[11] M. Garrido, F. Qureshi, J. Takala, and O. Gustafsson. Hardware architectures for the fast Fourier transform. Cited on pages 15, 17, and 23.

[12] M. Garrido, Keshab K. Parhi, and J. Grajal. A pipelined FFT architecture for real-valued signals. *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, 56(12):2634–2643, December 2009. Cited on page 15.

[13] M. Garrido, O. Gustafsson, and J. Grajal. Accurate rotations based on coefficient scaling. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 58(10):662–666, October 2011. Cited on pages 10, 11, and 16.

[14] M. Garrido, M. Acevedo, A. Ehliar, and O. Gustafsson. Challenging the limits of FFT performance on FPGAs. In *Int. Symp. Integrated Circuits*, pages 172–175, December 2014. Cited on page 1.

[15] M. Garrido, P. Källström, M. Kumm, and O. Gustafsson. CORDIC II: A new improved CORDIC algorithm. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 63(2):186–190, February 2016. ISSN 1549-7747. doi: 10.1109/TCSII.2015.2483422. Cited on pages 16, 17, and 28.

[16] Mario Garrido. *Efficient hardware architectures for the computation of the FFT and other related signal processing algorithms in real time*. PhD thesis, Universidad Politécnica de Madrid, December 2009. Cited on page 15.

[17] Mario Garrido. A new representation of FFT algorithms using triangular matrices. *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, 63(10):1737–1745, October 2016. Cited on pages 1 and 14.

[18] Mario Garrido, J. Grajal, M. A. Sánchez, and Oscar Gustafsson. Pipelined radix-2^k feedforward FFT architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(1):23–32, January 2013. Cited on pages 1, 15, and 17.

[19] Mario Garrido, F. Qureshi, and O. Gustafsson. Low-complexity multiplierless constant rotators based on combined coefficient selection and shift-and-add implementation (CCSSI). *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, 61(7):2002–2012, July 2014. Cited on pages 16, 17, and 32.

[20] Mario Garrido, Rikard Andersson, Fahad Qureshi, and Oscar Gustafsson. Multiplierless unity-gain SDF FFTs. *IEEE Transactions on Very Large Scale*

Integration (VLSI) Systems, 24(9):3003–3007, September 2016. Cited on pages 1, 8, and 15.

[21] Mario Garrido, Shen-Jui Huang, and Sau-Gee Chen. Feedforward FFT hardware architectures based on rotator allocation. *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, 2017. Accepted for publication. Cited on pages 1 and 16.

[22] B. Gold and T. Bially. Parallelism in fast Fourier transform hardware. *IEEE Transactions on Audio and Electroacoustics*, 21(1):5–16, February 1973. Cited on page 15.

[23] H.L. Groginsky and G.A. Works. A pipeline fast Fourier transform. *IEEE Transactions on Computers*, C-19(11):1015–1019, October 1970. Cited on page 15.

[24] S. He and M. Torkelson. Design and implementation of a 1024-point pipeline FFT processor. pages 131–134, May 1998. Cited on pages 1, 7, 15, and 17.

[25] J.A. Johnston. Parallel pipeline fast fourier transformer. In *IEE Proc. F Comm. Radar Signal Process.*, volume 130, pages 564–572, October 1983. Cited on page 15.

[26] P. Källström, M. Garrido, and O. Gustafsson. Low-complexity rotators for the FFT using base-3 signed stages. pages 519–522, December 2012. Cited on pages 8 and 16.

[27] Takeshi Kamazaki, Sachiko K. Okumura, Yoshihiro Chikada, Takeshi Okuda, Yasutaka Kurono, Satoru Iguchi, Shunji Mitsuishi, Yuji Murakami, Naomitsu Nishimuta, and Haruo Mita Ryo Sano. Digital spectro-correlator system for the atacama compact array of the atacama large millimeter/submillimeter array. *PASJ*, April 2012. Cited on pages 1 and 16.

[28] B. P. Lathi. *Linear Systems and Signals*. Oxford University Press, 2010. Cited on page 3.

[29] Hyun-Yong Lee and In-Cheol Park. Balanced binary-tree decomposition for area-efficient pipelined FFT processing. *IEEE Transactions on Circuits and Systems—Part I: Fundamental Theory and Applications*, 54(4):889–900, April 2007. ISSN 1549-8328. doi: 10.1109/TCSI.2006.888764. Cited on page 13.

[30] Jeesung Lee, Hanho Lee, Sang in Cho, and Sang-Sung Choi. A high-speed, low-complexity radix-2^4 FFT processor for MB-OFDM UWB systems. In *Proc. IEEE Int. Symp. Circuits Syst.*, pages 210–213, May 2006. Cited on page 15.

[31] Nuo Li and N.P. van der Meijs. 'A Radix 2^2 based parallel pipeline FFT processor for MB-OFDM UWB system'. pages 383 – 386, September 2009. Cited on page 15.

[32] S. Li, H. Xu, W. Fan, Y. Chen, and X. Zeng. A 128/256-point pipeline FFT/IFFT processor for MIMO OFDM system IEEE 802.16e. In *Proc. IEEE Int. Symp. Circuits Syst.*, pages 1488–1491, June 2010. Not cited.

[33] Y.-W. Lin and C.-Y. Lee. Design of an FFT/IFFT processor for MIMO OFDM systems. *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, 54(4):807–815, April 2007. Not cited.

[34] H. Liu and H. Lee. A high performance four-parallel 128/64-point radix-2^4 FFT/IFFT processor for MIMO-OFDM systems. In *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, pages 834–837, November 2008. Cited on page 15.

[35] L. Liu, J. Ren, X. Wang, and F. Ye. Design of low-power, 1GS/s throughput FFT processor for MIMO-OFDM UWB communication system. In *Proc. IEEE Int. Symp. Circuits Syst.*, pages 2594–2597, May 2007. Cited on page 15.

[36] Xue Liu, Feng Yu, and Zeke Wang. A pipelined architecture for normal I/O order FFT. *Journal of Zhejiang University - Science C*, 12(1):76–82, January 2011. Cited on page 15.

[37] J. H. McClellan and R. J. Purdy. *Applications of Digital Signal Processing*, chapter 5, Applications of Digital Signal Processing to Radar. Prentice-Hall, 1978. Cited on page 15.

[38] P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel. Formal datapath representation and manipulation for implementing DSP transforms. In *Design Automation Conf.*, pages 385–390, June 2008. Cited on page 15.

[39] Ryan Monroe. Improving the performance and resource utilization of the CASPER FFT and polyphase filterbank. *J. Astron. Instrum.*, 5(4), April 2016. Cited on pages 1 and 16.

[40] A.V. Oppenheim and R.W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall, 1989. Cited on page 4.

[41] Fahad Qureshi and Oscar Gustafsson. Generation of all radix-2 fast Fourier transform algorithms using binary trees. pages 677–680, August 2011. Cited on page 13.

[42] M.A. Sánchez, M. Garrido, M.L. López, and J. Grajal. Implementing FFT-based digital channelized receivers on FPGA platforms. *IEEE Transactions on Aerospace and Electronic Systems*, 44(4):1567–1585, October 2008. Cited on page 15.

[43] Kenneth L. Short. *VHDL for Engineers*. Pearson, 2014. Cited on page 9.

[44] E. E. Swartzlander, W. K. W. Young, and S. J. Joseph. A radix 4 delay commutator for fast Fourier transform processor implementation. *IEEE Journal of Solid-State Circuits*, 19(5):702–709, October 1984. Cited on page 15.

[45] Song-Nien Tang, Jui-Wei Tsai, and Tsin-Yuan Chang. A 2.4-GS/s FFT processor for OFDM-based WPAN applications. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 57(6):451–455, June 2010. Cited on page 15.

[46] Jack E. Volder. The CORDIC trigonometric computing technique. *IRE Trans. Electronic Computing*, EC-8:330–334, September 1959. Cited on page 17.

[47] Zeke Wang, Xue Liu, Bingsheng He, and Feng Yu. A combined SDC-SDF architecture for normal I/O pipelined radix-2 FFT. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(5):973–977, May 2015. ISSN 1063-8210. doi: 10.1109/TVLSI.2014.2319335. Cited on page 15.

[48] A. Wenzler and E. Luder. New structures for complex multipliers and their noise analysis. volume 2, pages 1432–1435, April 1995. Cited on page 17.

[49] E.H. Wold and A.M. Despain. Pipeline and parallel-pipeline FFT processors for VLSI implementations. *IEEE Transactions on Computers*, C-33(5):414–426, May 1984. Cited on page 15.

[50] W. Xudong and L. Yu. Special-purpose computer for 64-point FFT based on FPGA. In *Proc. Int. Conf. Wireless Comm. Signal Process.*, pages 1–3, November 2009. Cited on page 15.

[51] Liang Yang, Kewei Zhang, Hongxia Liu, Jin Huang, and Shitan Huang. An efficient locally pipelined FFT processor. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 53(7):585–589, July 2006. Cited on pages 1 and 15.