UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

MASTER'S THESIS

DESIGN OF PARALLEL PIPELINED FFT ARCHITECTURES FOR 6G ON FPGAS

JEZAEL PÉREZ HERRERA

SEPTEMBER 2022

MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

MASTER'S THESIS

Title:	DESIGN OF PARALLEL PIPELINED FFT ARCHITECTURES FOR 6G ON FPGAS
Author:	JEZAEL PÉREZ HERRERA
Supervisor:	MARIO GARRIDO GÁLVEZ
Ponente:	PEDRO MALAGÓN MARZO
Department:	DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

EVALUATION COMMITTEE

President: .....

Vocal: .....

Secretary: .....

The evaluation committee members agree to grant the grade: .....

Madrid, on ....., 2022

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

MASTER'S THESIS

DESIGN OF PARALLEL PIPELINED FFT ARCHITECTURES FOR 6G ON FPGAS

AUTHOR: JEZAEL PÉREZ HERRERA

SUPERVISOR: MARIO GARRIDO GÁLVEZ

PONENTE: PEDRO MALAGÓN MARZO

SEPTEMBER 2022

A mi familia.

Abstract

The goal of this Master's Thesis is the design of a parallel pipelined fast Fourier transform (FFT) architecture with reduced area and low power consumption for its implementation on field programmable gate arrays (FPGAs) for 6G applications.

To achieve this purpose, a first the research about the state of the art of the different components of an FFT and the different existing FFT architectures has been carried out.

Then, the architecture proposed in this Master's Thesis is a 1024-point radix-2^4 4-parallel FFT, based on existing multi-path delay feedback (MDF) architecture and a new parallel version of the single-stream feedforward (SFF) architecture, the multi-stream feedforward (MFF). This design is implemented on a Xilinx Virtex 7 FPGA using VHDL. The detailed explanation of the proposed approach is presented in this Thesis, including the specific mapping on the FPGA and the different techniques used to achieve an optimized architecture implementation. Finally, the experimental results and conclusions are provided, including a comparison with other architectures where the proposed architecture achieves low power consumption and a small usage of look-up tables (LUTs) and flip-flops (FFs) in the FPGA.

Key Words

FFT, MDF, MFF, radix, FPGA, VHDL

Resumen

El objetivo de este Trabajo Fin de Máster es el diseño de una arquitectura hardware paralela de la FFT en pipeline con área reducida y bajo consumo de potencia para su implementación en *field programmable gate arrays* (FPGAs) en aplicaciones 6G.

Para lograr este propósito, se ha llevado a cabo una primera investigación sobre el estado del arte de los diferentes componentes de la *fast Fourier transform* (FFT) y las arquitecturas FFT existentes.

La arquitectura propuesta en este Trabajo de Fin de Máster es una FFT 1024 puntos con radix-2^4 y una paralelización de 4, basada en la arquitectura *multi-path delay feedback* (MDF) y una nueva versión paralela de la arquitectura *single-stream feedforward* (SFF), la *multi-stream feedforward* (MFF). Este diseño se ha implementado en un FPGA Xilinx Virtex 7 usando VHDL. En este trabajo, se presenta una explicación detallada del diseño propuesto incluyendo el mapeo sobre la FPGA y las diferentes técnicas utilizadas para lograr una implementación optimizada de la arquitectura. Por último, se exponen los resultados experimentales y conclusiones, incluyendo una comparación con otras arquitecturas donde se observa que la arquitectura propuesta obtiene bajo consumo de potencia y un uso muy reducido de *look-up tables* (LUTs) y *flip-flops* (FFs).

Palabras clave

FFT, MDF, MFF, radix, FPGA, VHDL

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the arrival of 1G around 1980, the scientific and technological innovations in the field of mobile communication have never stopped. The rapid development of various emerging applications, such as artificial intelligence (AI) [2, 3], virtual reality (VR) [4, 5], three-dimensional (3D) media [6, 7], and the Internet of Everything (IoE) [8, 9] has led to a massive volume of traffic. The recent fifth-generation (5G) wireless networks [10, 11] has been deployed to support these applications, achieving data transfer speeds of up to 10 Gb/s with a latency of 1 ms.

However, 5G networks will not have the capacity to deliver a completely automated and intelligent network that provides everything as a service and a completely immersive experience. New applications in the future such as holography [12] or connected autonomous vehicles (CAV) [13, 14] may require a latency below 100 μs and data rates up to Terabits per second. This motivates the need for developing the sixth-generation (6G) wireless network [15, 12]. To achieve the desired performance, all parts of the 6G system must be optimized, which includes the calculation of the fast Fourier transform (FFT) for the data modulation and demodulation [16, 17].

One of the most relevant FFT algorithms was proposed more than 50 years ago by Cooley and Tukey [18] as an efficient method for calculating the discrete Fourier transform (DFT) [19]. The FFT is a mathematical algorithm that allows to transform a signal in the time domain to the frequency domain. The Cooley-Tukey algorithm can carry out this transformation using two different methods [19]: Decimation in time (DIF) and decimation in frequency (DIT). Radix-2 DIF FFT repeatedly separates the output sequence into odd and even samples, while in radix-2 DIT FFT this decomposition is performed on the input sequence [19].

In 1970, the first FFT pipelined hardware architectures emerged [20, 21]. Nowadays, pipeline architectures [22] are widely used in modern applications that require high performance and low latency at a reasonable area cost and power consumption.

There are three main types of pipelined FFT architectures: Feedback, feedfor-

ward and serial commutator. Additionally, these pipeline FFT architectures can be serial or parallel. This Master's Thesis focuses on the development of a parallel FFT architecture based on feedback and feedforward architectures. In feedback architectures, the single-path delay feedback (SDF) architecture [21, 23, 24] represents the serial FFT architecture and the multi-path delay feedback (MDF) [25, 26, 27] is the parallel version. Likewise, the single-stream feedforward architecture (SFF) [28] is considered, but a parallel version of this architecture has not been developed so far.

In this Master's Thesis, an FFT architecture has been designed and implemented on a field programmable gate array (FPGA). Due to its future application in 6G technology, the architecture is designed based on parallel FFT architectures that allow us to obtain higher data rates and low latency. In addition, low resources usage and low power consumption have been some of the main objectives. To achieve this, we propose a specific mapping for all components of the architecture in terms of look-up tables (LUTs), memory and digital signal processing (DSP) slices. Additionally, the architecture is divided in modules depending on the FFT stage. For this Master's Thesis, a new innovative multi-stream feedforward (MFF) architecture has been developed, which has not been explored so far. By analyzing the viability of this new architecture, we could observe a better result in terms of area than other architectures in certain cases. Therefore, some modules are based on this new MFF and other ones are based on MDF, depending on the memory size and the rotation needed at each stage of the FFT. Then, all these modules are integrated together, with a dedicated rotation memory control and generation of other control signals.

This document is divided into five chapters. Chapter 2 describes the importance of the Cooley-Tukey algorithm, with the main FFT representations and rotators that we can distinguish, in addition to an explanation of the most significant pipelined FFT hardware architectures. In chapter 3, the design and implementation of the proposed approach of this Master's Thesis is presented. Chapter 4 describes the experimental results of our architecture and a comparison with other pipeline architectures of the state-of-the-art. Finally, the conclusions of this work and possible future research lines are explained in Chapter 5.

Chapter 2

State of the Art

In this chapter, a review of basic concepts about FFT algorithms and architectures is provided. This includes the representation of FFT algorithms, the design of rotators, and serial and parallel pipelined FFT architectures.

2.1 FFT in 6G applications

6G networks will be able to process data at higher rates than 5G networks and provide substantially higher capacity and much lower latency. To achieve this, FFT architectures need to have a high throughput, low latency and low power consumption. High-throughput and low latency requirements can be achieved by increasing the parallelization and pipelining the FFT architecture. This is critical in several 6G applications such as ultra-reliable low-latency communications or connected (CAV) [13, 14], where a reduction in latency will make cars react faster and, thus, reduce the number of accidents.

Finally, the power consumption is another relevant factor in 6G technology, which is influenced by the area required. For example, if we implement an FFT architecture with a parallelization of 32 branches, the latency of the system would be low, but the necessary FPGA resources to implement it would be high, leading to a higher power consumption. Therefore, the development of the FFT architecture in this Master's Thesis will focus on maintaining a balance between the latency and the area usage.

2.2 FFT algorithms

The discrete Fourier transform (DFT) [19] is one of the most powerful tools in digital signal processing. It allows us to obtain the spectrum of a finite-duration signal. The DFT is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{nk}, \quad k = 0, 1, \ \ldots\ , N-1, \tag{2.1}$$

where N is the DFT size, $x[n]$ is the complex input data in time domain, $X[k]$ is the complex output value in the frequency domain, n is the input time index, k is the output frequency index, and the term $W_N^{nk} = e^{-j\frac{2\pi}{N}nk}$ is called twiddle factor and corresponds to a rotation in the complex plane.

According to (2.1), the computation of each output frequency requires approximately N multiplications and N sums. Then, for calculating all N output frequencies there will be approximately N^2 multiplications and N^2 sums, i.e., the operation order of the DFT is $\mathcal{O}(N^2)$. The fast Fourier transform (FFT) was born from the need to calculate the DFT efficiently. The most used FFT algorithm was proposed by Cooley and Tukey [18]. The Cooley-Tukey algorithm is based on the fact that the calculation of the output frequencies in the DFT includes operations that can be shared among those outputs. Therefore, by decomposing the equation of the DFT, the Cooley-Tukey algorithm reduces the number of operations from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log(N))$. This can be clearly seen in the example of figure 2.1, where the sums and subtractions shared among the output frequencies are indicated.

$$\begin{aligned} X[0] &= x[0] + x[2] + x[1] + x[3] \\ X[2] &= x[0] + x[2] - (x[1] + x[3]) \\ X[1] &= x[0] - x[2] - j(x[1] - x[3]) \\ X[3] &= x[0] - x[2] + j(x[1] - x[3]) \end{aligned}$$

Figure 2.1: Redundant operations in a 4-point FFT.

There are two typical ways to decompose the FFT: DIF and DIT decomposition [19]. The DIF FFT applies the decomposition on the output sequence separating it into even and odd samples iteratively, while in DIT FFT this decomposition is applied on the input sequence. As it was observed in [29], DIF and DIT FFT algorithms based on the Cooley-Tukey algorithm only differ in the rotations at the FFT stages. Throughout this chapter, the DIF FFT will be used as an example.

2.2.1 FFT representations

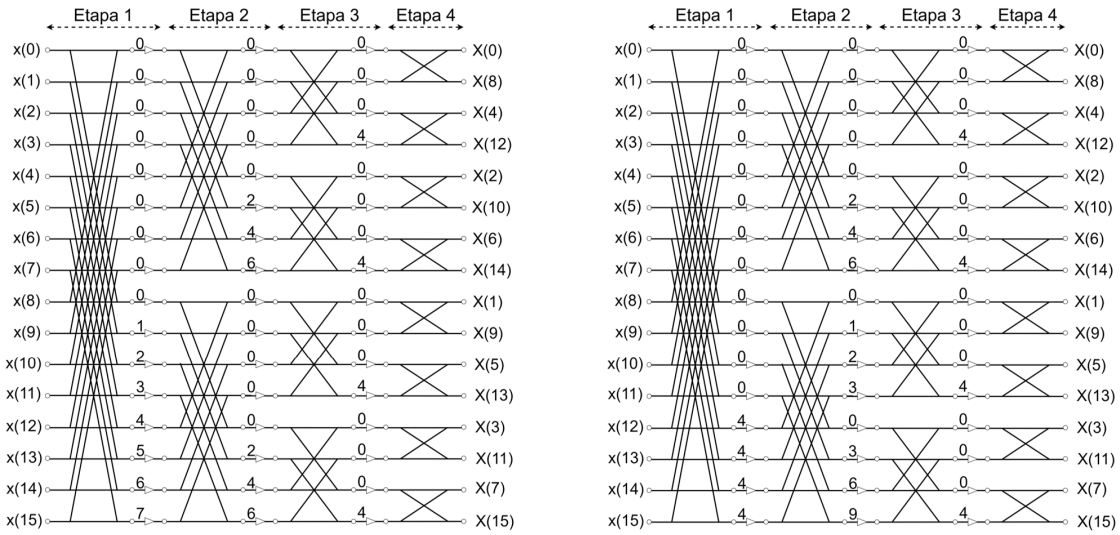
In order to represent the FFT algorithm, several representations have been developed. They include the flow graph, the binary tree and the triangular matrix.

Flow graph

The most common representation of the FFT is the flow graph. Figure 2.2 show two examples of flow graphs. At each stage of the flow graph, there are pairs of additions and subtractions, named butterflies. The upper part of the butterfly represents the addition and the lower one the subtraction. At the output of each butterfly, a number can be observed. It corresponds to the value ϕ and represents a rotation by

$$W_N^{\phi} = e^{-j\frac{2\pi}{N}\phi}. \tag{2.2}$$

A rotation by the angles 0°, 90°($\phi = \frac{3N}{4}$), 180°($\phi = \frac{N}{2}$) or 270°($\phi = \frac{N}{4}$) is considered as a trivial rotation. This is due to its hardware-efficient implementation, which consists of exchanging the real and imaginary parts and/or changing the sign of the data.



(a) (b)

Figure 2.2: Flow graph examples. (a) 16-point radix-2 DIF FFT. (b) 16-point radix-2^2 DIF FFT.

Both figures 2.2(a) and 2.2(b) show the flow graph of a 16-point DIF FFT for radices 2 and 2^2 respectively. For any radix-ρ^k, the base of the radix indicates the size of the butterflies and the exponent relates to the rotation distribution among FFT stages, placing the most complex rotations every k stage. For instance, the

figure 2.2(b) shows a 16-point radix-2^2 algorithm where the most complex rotations are placed every 2 stages, keeping the odd stages only with trivial rotations.

Binary tree

The binary tree representation [30, 31] is used to represent the FFT algorithms for a 2^n-point radix-2^k FFT. The binary tree representation splits the initial DFT into smaller DFTs. Initially, a N-point DFT (2.1) is divided into Q DFTs of P-point and P DFTs of Q-point, which is described mathematically as

$$X\left[k_1 + Pk_2\right] = \sum_{t_2=0}^{Q-1} \left[\left(\sum_{t_1=0}^{P-1} x\left[Qt_1 + t_2\right] W_P^{t_1 k_1} \right) W_N^{t_2 k_1} \right] W_Q^{t_2 k_2}. \tag{2.3}$$

This division results in an inner P-point DFT and an outer Q-point DFT. For a 16-point FFT, the stages that compose the FFT are shown in figure 2.3.

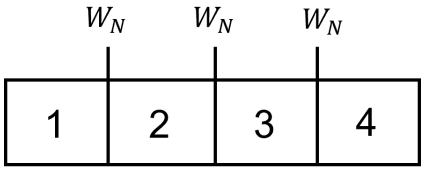


Figure 2.3: Stages of a 16-point FFT.

The selection of the values P and Q determines where the DFT is split. Each splitting can be done in multiple different ways, which correspond to the possible combinations of $P = 2^p$ and $Q = 2^q$, whose product is equal to $N = 2^n$. The value of n indicates the number of stages before the split, which result in p and q leafs ($n = p + q$). Thus, the twiddle factor in the first split will always be equal to the size of the FFT, in this case W_{16}.

Two examples are illustrated in figures 2.4(d) and 2.4(e). These examples are composed by doing the first split between the 3^{rd} and 4^{th} stage, respectively. With this, 3 stages are obtained on the left and 1 on the right, which corresponds to $p = 3$ and $q = 1$. Then, the second division takes place between stages 1 and 2 or between 2 and 3. This election will determine the values of p and q, which leads to a different algorithm in each case, $p = 1$ and $q = 2$ for figure 2.4(d) and $p = 2$ and $q = 1$ for figure 2.4(e).

For the binary tree representation, the number of algorithms to compute a 2^n-point FFT using different Cooley-Tukey decomposition schemes is [31]

$$\frac{(2(n-1))!}{n!(n-1)!}. \tag{2.4}$$

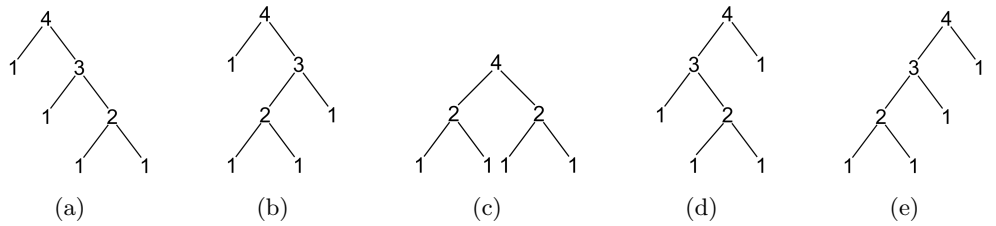


Figure 2.4: Binary tree of all possible algorithms for $N = 16$.

Figure 2.4 shows all the possible algorithms for $N = 16$ according to the binary tree representation.

Triangular matrix

Another way to represent the FFT is by using the triangular matrix representation [29]. Figure 2.5 shows the triangular matrix representation of the algorithms in figure 2.4.

In the triangular matrix representation, each element in the matrix represents a set of rotations that can be moves along several stages. The lowest stage where these rotations can be placed is indicated by the number of the row at the right of the matrix, whereas the highest stage is indicated by the column of the matrix, i.e., by the numbers on top of the matrix.

For example, let us start from the distribution illustrated in figure 2.6, resulting in the triangular matrix of figure 2.5(d). First, the highest twiddle factor must be placed, in this case W_{16}. The procedure for each twiddle factor is the following:

1. We have to look for higher twiddle factors at the left of the stage, where the actual twiddle factor is located. If a higher twiddle factor is found, the stage where it is placed is incremented by 1, determining the row of the triangular matrix where we may place the stage being analyzed. The column is selected by the stage being analyzed. If there is not any higher twiddle factor, the row is related to the first stage. Therefore, in the case of stage 3 (W_{16}), a number 3 is placed in {1,3}. In the case of stage 1 (W_8), a number 1 is placed on {1,1}. In the case of stage 2 (W_4), a number 2 is placed on {2,2}.

2. We have to look for higher twiddle factors at the right of the stage, where the actual twiddle factor is located. If a higher twiddle factor is found, the stage where it is placed is decremented by 1, determining the column of the triangular matrix where we may place the stage being analyzed. The row is selected by the stage being analyzed. If there is not any higher twiddle factor, the column is related to the last stage. Therefore, in the case of stage 3 (W_{16}),

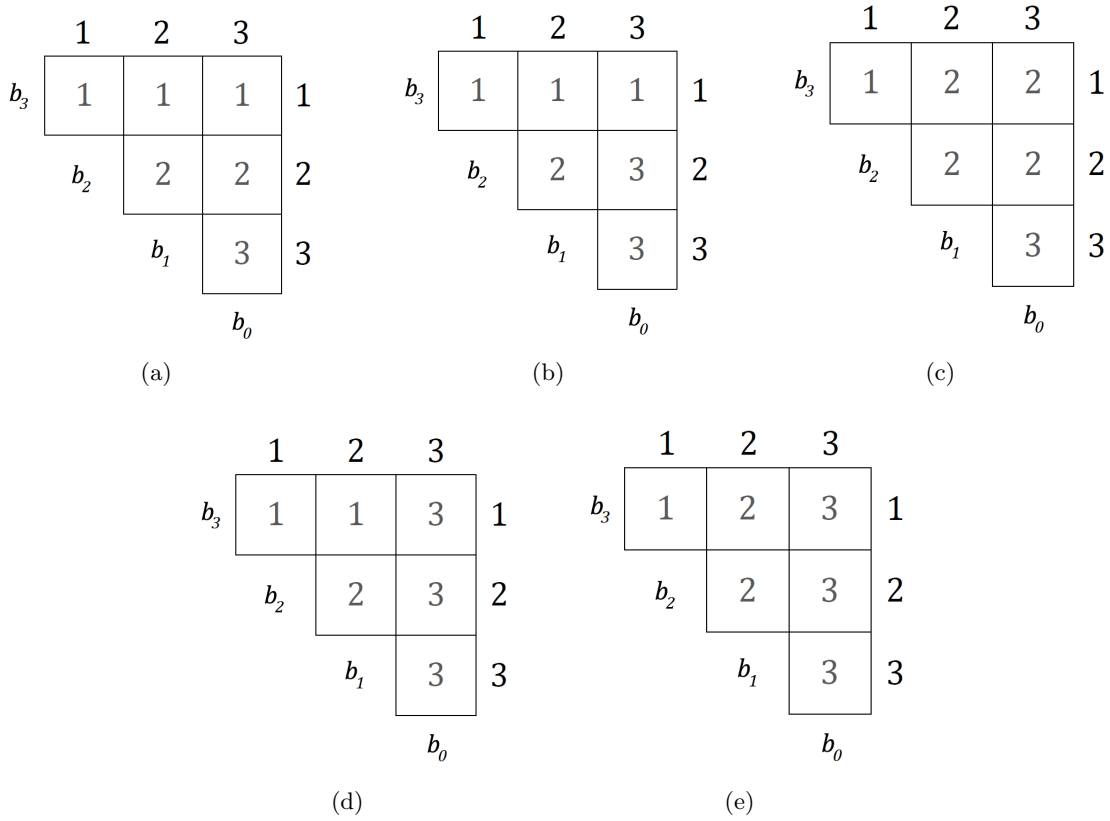


Figure 2.5: Triangular matrix representation of all possible algorithms for $N = 16$.

a number 3 is placed in {3,3}. In the case of stage 1 (W_8), a number 1 is placed on {1,2}. In the case of stage 2 (W_4), a number 2 is placed on {2,2}.

3. All locations between the positions set, are filled in with the value of the stage in question. For this example, a value of 3 is placed between {1,3} and {3,3}.

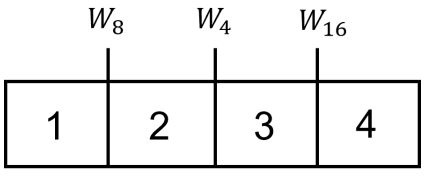


Figure 2.6: A possible twiddle factor distribution among stages of a 16-point FFT

When the stage of the first twiddle factor is placed, the same procedure is repeated for the next twiddle factor and so on, until the triangular matrix is filled.

The total number of algorithms that can be represented by the triangular matrix

representation is

$$\prod_{k=1}^{n-1}(n-k)^k. \tag{2.5}$$

Apart from allowing us to visualize the distribution of the rotations along the FFT stages, the triangular matrix representation is used as a starting point to obtain the twiddle factors for any FFT algorithm. For this purpose, the bits of the index are included in figure 2.5 in order to obtain the values of the twiddle factors using the expression

$$\phi_s(I) \equiv \sum_{x_{ij}=s} [b(i)\underbrace{0\ldots0}_{n-1-i}] \cdot [b(j)\underbrace{0\ldots0}_{j}], \tag{2.6}$$

where the sum is for all the elements of the matrix x_{ij}, s is the stage, $b(i)$ is the bit in row i, and $b(j)$ is the bit in column j.

For instance, from the triangular matrix in figure 2.5(a) and the equation (2.6), we can obtain the different rotations for each stage by the following expressions:

$$\begin{aligned}
\phi_1(I) &\equiv b_3 \cdot [b_2 00] + b_3 \cdot [b_1 0] + b_3 \cdot b_0 = b_3 \cdot [b_2 b_1 b_0]\,, \\
\phi_2(I) &\equiv [b_2 0] \cdot [b_1 0] + [b_2 0] \cdot [b_0] = [b_2 0] \cdot [b_1 b_0]\,, \\
\phi_3(I) &\equiv [b_1 00] \cdot [b_0]\,.
\end{aligned} \tag{2.7}$$

2.3 Rotators

A rotation of a complex number $x + jy$ by an angle α can be described as

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} C & -S \\ S & C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \tag{2.8}$$

where $X + jY$ is the result of the rotation and $C + jS$ is the rotation coefficient that approximates the angle α, i.e., $\alpha \simeq \tan^{-1}(S/C)$. This approximation has a certain error, as can be seen in the following expression for calculating the coefficients C and S:

$$\begin{aligned}
C &= R\,(\cos\alpha + \epsilon_c)\,, \\
S &= R\,(\sin\alpha + \epsilon_s)\,,
\end{aligned} \tag{2.9}$$

where ϵ_c and ϵ_s are the relative approximation errors of the cosine and sine component, respectively, and R is the scaling factor [32].

In order to calculate the rotations, there are two main types of rotators: General rotators and constant rotators. General rotators can carry out a rotation by any input angle, which is provided as an input, while constant rotators calculate rotations by a specific angles. Both rotators can be implemented by a complex multiplier or by the CORDIC algorithm. Additionally, constant rotators can be implemented using shift-and-add operations by applying the combined coefficient selection and shift-and-add implementation (CCSSI) [33].

All of these types of rotator implementations are explained next.

2.3.1 Multiplier-based implementation

The use of complex multipliers is the most straightforward approach to implement a rotation. The resulting rotation expression is

$$\begin{aligned} X &= xC - yS, \\ Y &= xS + yC, \end{aligned} \tag{2.10}$$

which requires four multipliers and two additions, as shown in figure 2.7(a).

This approach is based on the supply of the C and S coefficients at the input of the rotator. These coefficients may be stored in a ROM, as in [34, 35]. In addition, (2.10) may be rewritten in order to reduce the number of multipliers from four to three [36], according to

$$\begin{aligned} X &= x(C + S) - (x + y)S, \\ Y &= y(C - S) + (x + y)S. \end{aligned} \tag{2.11}$$

This approach is shown in figure 2.7(b). In addition, this last complex multiplier can be implemented with only 3 adders, by replacing the adders that add coefficients by ROM memories.

2.3.2 CORDIC algorithm

The CORDIC algorithm [37, 38, 39] is another alternative to calculate general rotations. It is based on breaking down the rotation angle into a series of micro-rotations by specific angles α_k, according to

$$\theta = \sum_{k=0}^{M} \alpha_k + \epsilon_\phi, \tag{2.12}$$

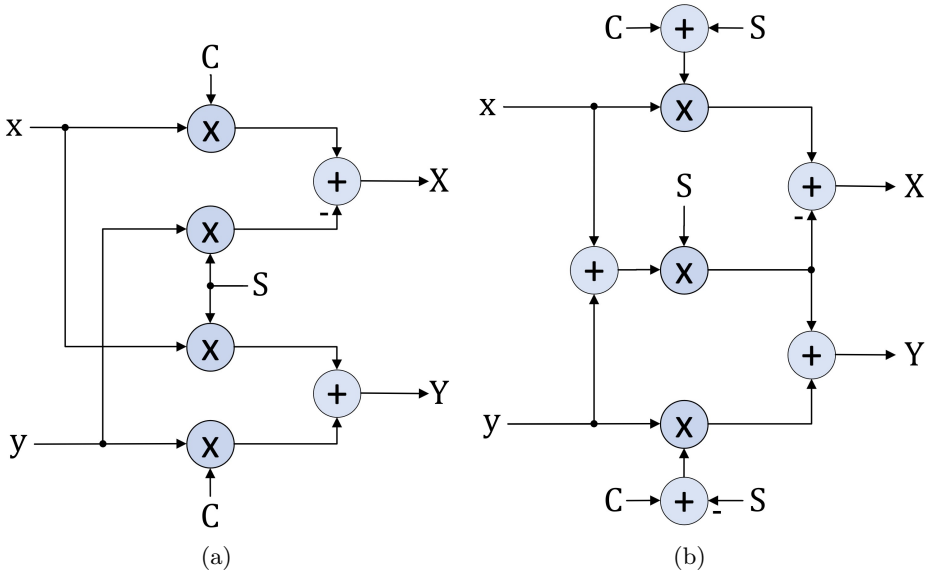


Figure 2.7: Multiplier-based rotators. (a) Standard complex multiplier based on (2.10). (b) Modified complex multiplier based on (2.11).

where $\alpha_k = \pm \tan^{-1}(2^{-k})$ and ϵ_ϕ is the remaining angular error.

The rotation coefficients are $P = C + jS = 2^k + j\delta_k$, where $\delta_k \in \{-1, 1\}$, and $k = 0, \ldots, M$ is the micro-rotation stage. Each micro-rotation stage calculates

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 2^k & -\delta_k \\ \delta_k & 2^k \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \tag{2.13}$$

where δ_k determines the direction of the rotation and the scaling factor of the stage is $R(k) = \sqrt{2^{2k} + 1}$.

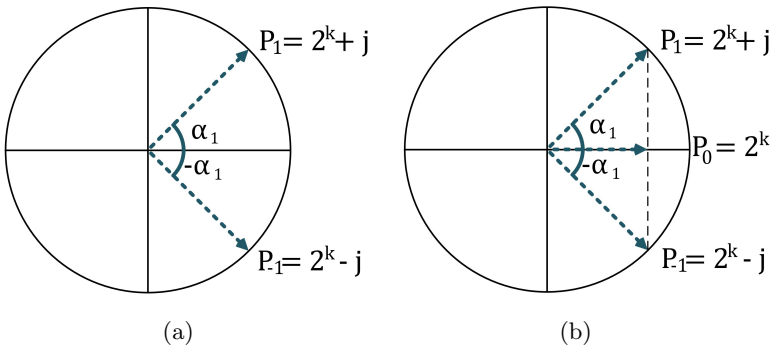


Figure 2.8: CORDIC algorithm. (a) Standard CORDIC algorithm. (b) Redundant CORDIC algorithm.

An alternative to the CORDIC algorithm is the so-called redundant CORDIC

algorithm. It has the same set of coefficients as the standard CORDIC, with the particularity that $\delta_k \in \{-1, 0, 1\}$. Thus, a increased number of angles per stage are allowed. By contrast, it needs special stages to compensate the different scaling of the angles of the kernel. The differences between both algorithms can be clearly seen in figure 2.8.

Regarding constant rotators, CORDIC-based approaches are based on selecting stages of the conventional CORDIC [40, 41, 42] or increasing the amount of micro-rotation angles [43, 44, 45].

2.3.3 CCSSI

For constant rotators, it can be distinguished between single constant rotator (SCR) and multiple constant rotator (MCR). SCR refers to a rotation by a single angle. Conversely, MCR has three types: A rotator that rotates by multiple angles, multiple rotators with one rotation angle for each of them, and multiple rotators with multiple rotation angles for each of them.

The CCSSI method [33] provides solutions to both types of constant rotators. The design process will be explained with two examples at the same time: An SCR for the angle $\alpha = 40°$ and an MCR for the angles $\alpha_1 = 17°$ and $\alpha_2 = 40°$. For MCR, both angles shall have the same scaling. These examples will consider a word length of 4 bits, a maximum error $\epsilon_{max} = 5 \times 10^{-2}$ and a maximum of four adders for the rotator.

- *Step 1.* The complete design space is initialized, as illustrated in figure 2.9(a). This design space will consist in $\left(2^{2b}\right)^M$ coefficients for each angle, where b is the word length and M is the number of angles, i.e., $M = 1$ for the SCR rotator and $M = 2$ for the MCR rotator.

- *Step 2.* The design space is reduced according to an angle $\delta = \sin^{-1}(\epsilon_{max})$, i.e, only the coefficients which differ at most δ from the required angle will be considered:

$$\left|\tan^{-1}\left(\frac{S_i}{C_i}\right) - \alpha_i\right| < \delta. \tag{2.14}$$

  After this step, the number of alternative coefficients for each angle is reduced as shown in figure 2.9(b).

- *Step 3.* The reduction in this step is performed according to the maximum number of adders that it must be used for each rotation. Coefficients that require more than the allowed number of adders are discarded. In this example, with 4 maximum adders, the resulting space design is shown in figure 2.9(c).

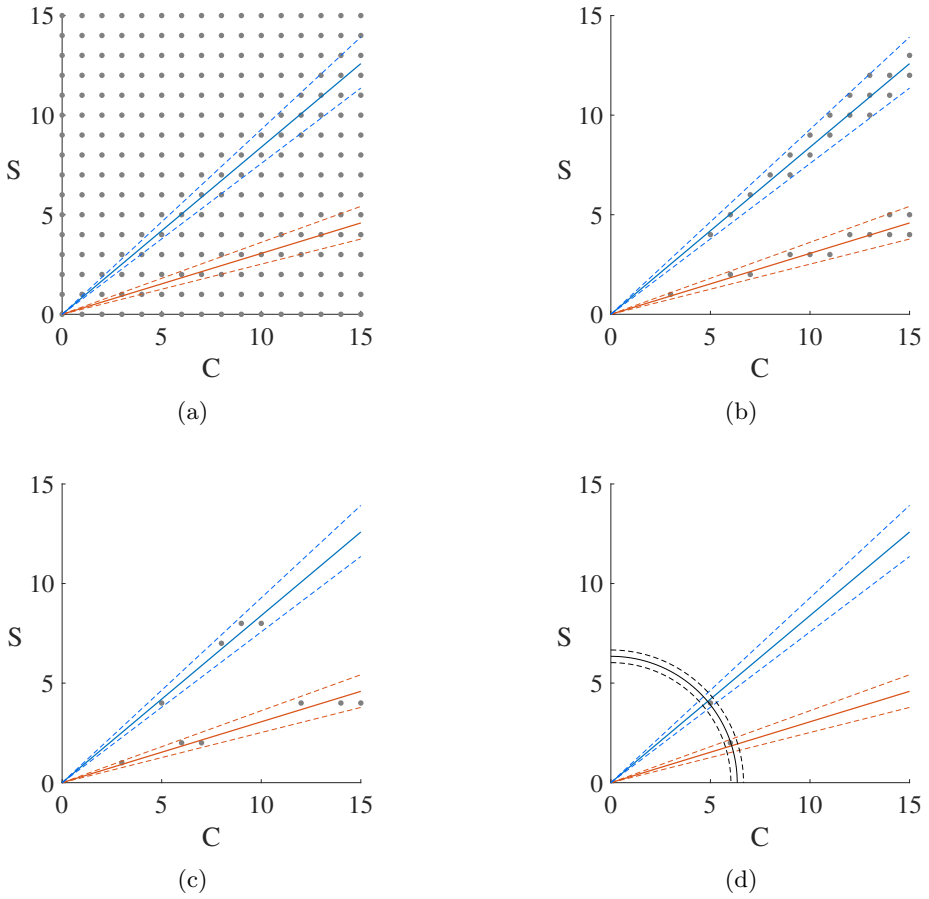


Figure 2.9: Design process of CCSSI example. (a) Step 1. (b) Step 2. (c) Step 3. (d) Step 4.

- *Step 4.* At this point, the SCR rotator may already have several coefficients that satisfy the specifications. The coefficient with the smallest rotation error is usually selected as the best one.

  For the MCR case, combinations of coefficient close to the rotation angles that have similar radius are considered. The resulting coefficients are shown in figure 2.9(d). These selected coefficients are $5 + 4j$ for 40° and $6 + 2j$ for 17°.

- *Step 5.* Finally, the rotator is implemented. For it, a shift-and-add implementation is used. Both examples are illustrated in figure 2.10, where the MCR case is a merged implementation of both rotators by 17° and 40°, shown in figure 2.10(c).

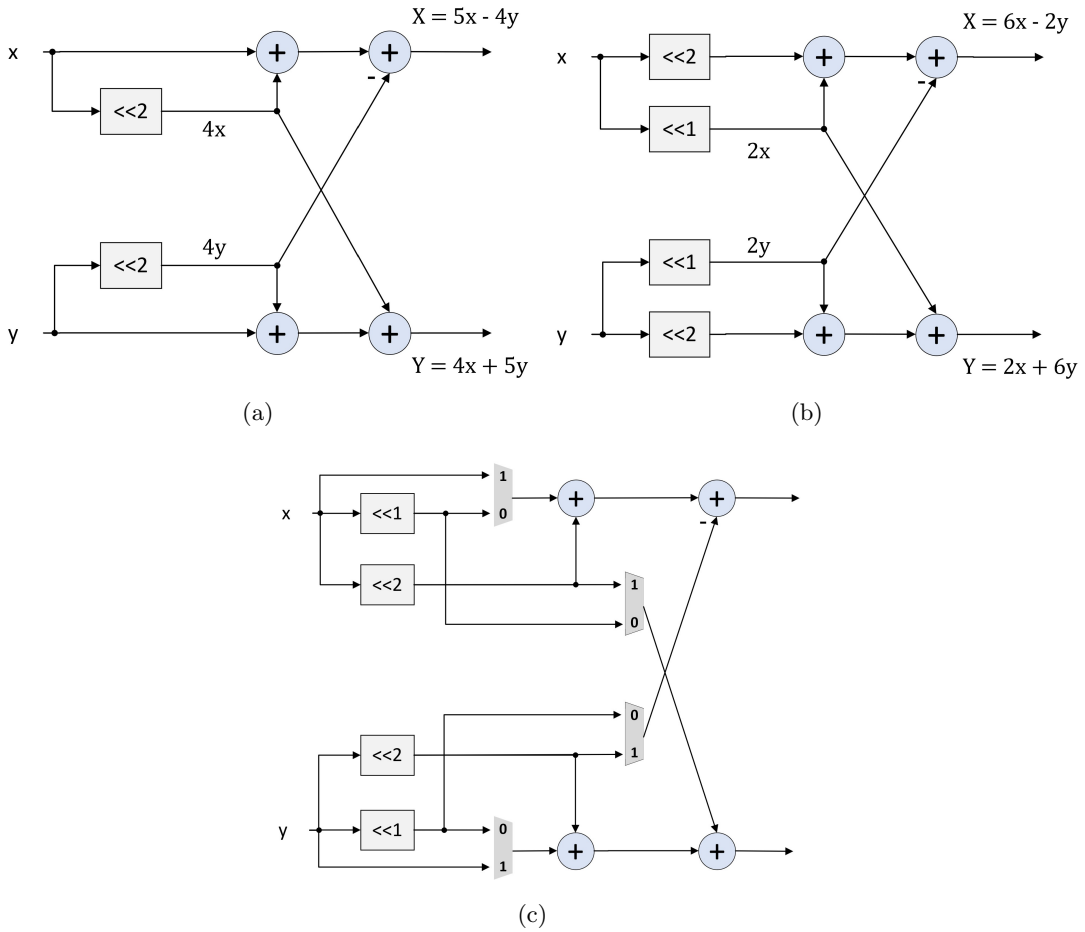


Figure 2.10: Rotator implementation of the example. (a) Rotator by 40° using 5 + 4j. (b) Rotator by 17° using 6 + 2j. (c) Rotator by 17° and 40° merged.

2.4 The bit b_{n-s}

The concept of the bit b_{n-s} must be taken into account when designing FFT architectures. In order to understand it, the flow graph of figure 2.11 will be considered, where the index I of data arriving to the FFT is indicated. This index I depends on the bits of the binary representation of itself, according to

$$I \equiv b_{n-1} \dots b_0, \tag{2.15}$$

being the b_i bits, the binary representation of I.

In the flow graph, it can be observed that pairs of data operated in each butterfly only differ in a bit of the index. For instance, at the first stage $x[1]$ and $x[9]$ are operated together in a butterfly and $I = 1$ and $I = 9$ only differ in a bit b_3. Figure 2.12 shows the indexes of pairs of data that must be processed in the butterflies at

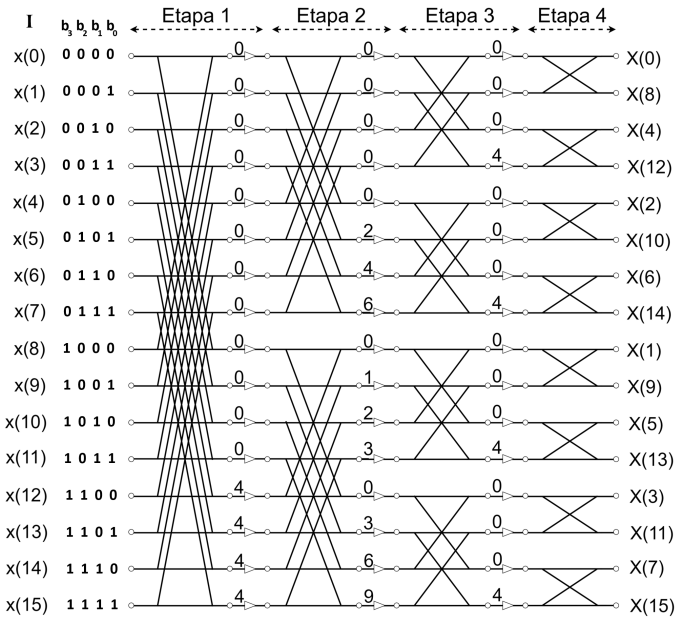


Figure 2.11: Flow graph example with index bits

the different stages. It can be observed that in the first stage pairs of data only differ in b_3, in the second stage only differ in b_2, in the third stage only differ in b_1 and in the fourth stage only differ in b_0.

Thus, the rule that must be considered is: For any N-point FFT, the indexes of pairs of data that are processed in butterflies at stage s only differ in the bit b_{n-s}, where $n = \log_2 N$ is the number of FFT stages. Therefore, in any FFT architecture, pairs of data that are provided at the inputs of a butterfly at the same clock cycle, must always differ in the bit b_{n-s}.

2.5 Pipelined FFT hardware architectures

FFT hardware architectures are designed to meet the requirements of the most demanding applications in terms of performance, circuit area and power consumption. There are three main types of hardware architectures: Iterative, fully parallel and pipelined. This Master's Thesis deals only with pipelined FFT hardware architectures [22]. Pipelined architectures can be serial or parallel.

Serial pipelined FFT architectures are classified into single-path delay feedback (SDF) [21, 23, 24], single-stream feedforward (SFF) [28], single-path delay commutator (SDC) [20] and serial commutator (SC) [46]. Parallel pipelined FFT architectures are classified into multi-path delay feedback (MDF) [25, 26, 27], multi-path delay commutator (MDC) [47, 48, 49] and multi-path serial commutator (MSC) [50]. Additionally, based on those parallel pipelined FFT architectures, other approaches

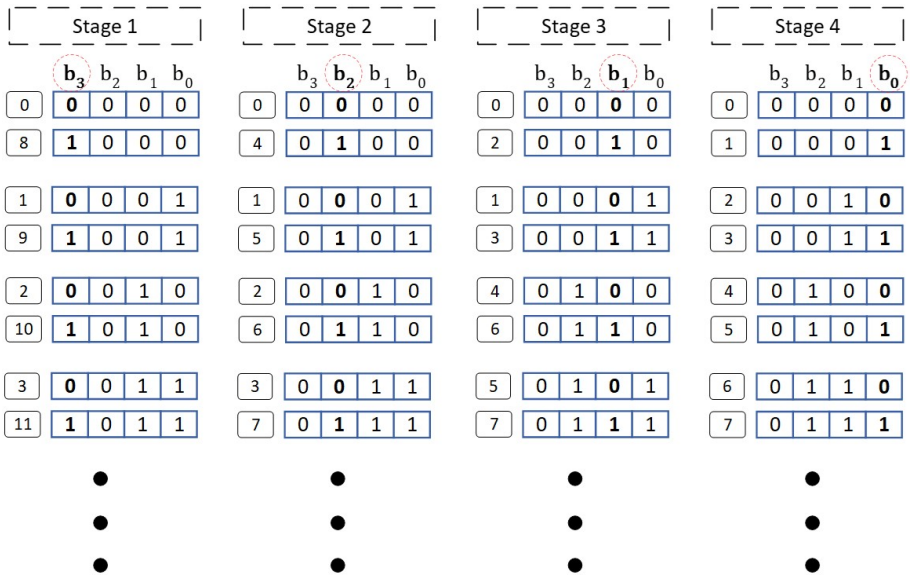


Figure 2.12: Index differing in the bit b_{n-s}

have been developed with significant improvements in terms of area, throughput and power consumption [51, 52, 53, 54, 55, 56].

The architectures related to this Master's Thesis are the SFF, SDF and MDF. These architectures are explained next. A more extended survey of these architectures is provided in [22].

2.5.1 Serial architectures

Serial pipelined FFT architectures receive data in series in a continuous flow. These architectures are characterized by their relatively low number of components (adders, rotators and memory) and a throughput of 1 sample per clock cycle.

SDF

The SDF FFT architecture consists of $n = \log_2 N$ stages with radix-2 butterflies (R2), rotators ($\otimes$) or trivial rotators (diamond-shaped), and buffers. With this architecture, any radix-2^k FFT algorithm can be implemented [57], where the difference is reflected in the rotations between stages. This is illustrated in figures 2.13(a) and 2.13(b) respectively, where the only difference is the rotator at the stage 1, which is a trivial rotator in the case of radix-2^2.

At each stage, the SDF receives the data from top to bottom according to the flow graph. As explained in Section 2.4, pairs of data computed in the butterflies

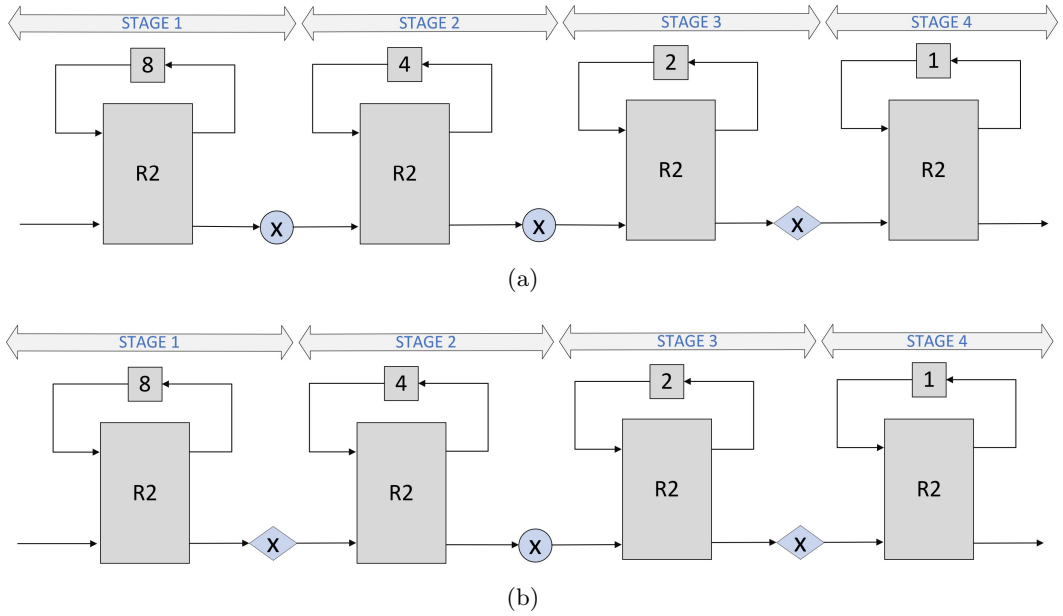


Figure 2.13: 16-point SDF FFT architectures. (a)With radix-2. (b)With radix-2^2.

must always differ in the bit b_{n-s}. These pairs of data that differ in b_{n-s} arrive with a difference of 2^{n-s} clock cycles. Thus, a buffer of length $L = 2^{n-s}$ is situated in each stage.

The data flow is easily understandable by looking at figure 2.14, which represents the internal structure of an SDF stage. The first $L = 2^{n-s}$ values are stored in the buffer. When 2^{n-s} clock cycles have passed, the first value stored in the buffer is computed in the butterfly together with the actual input of the stage. From this moment, a pair of data is processed every clock cycle. The upper outputs of the butterfly are sent later to the rotator, whereas the lower outputs are fed back to the buffer to be sent to the rotator. The output of the rotator is connected to the input of the next stage.

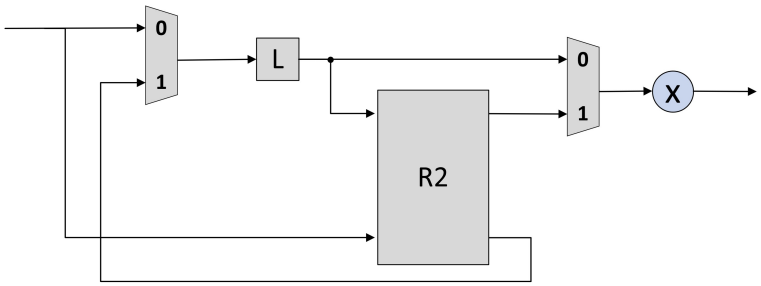


Figure 2.14: Stage of a radix-2 SDF FFT architecture

The butterflies are only used when they process the pairs of data, which only occurs the 50% of the time. Thus, the SDF has a 50% utilization of the butterflies.

As a radix-2^k SDF FFT uses one butterfly per stage, $2\log_2 N$ complex adders are used for the entire FFT. In addition, the buffers at each stage result in a total memory of $N-1$. Finally, the use of different FFT algorithms is related to the complexity of the rotators. For the 16-point SDF FFTs of figure 2.13, it can be observed that the radix-2^2 architecture has half complexity of the rotators compared with the radix-2 one, being the number of rotators $1/2 \cdot \log_2 N - 1$ for radix-2^2 and $\log_2 N - 2$ for radix-2.

SFF

The SFF FFT architecture is shown in figure 2.15. This architecture shares with the SDF FFT the characteristic that data arrive in natural order at the input of the architecture. This means that x[0], x[1], x[2], ..., arrive in order. The main difference between these architectures is that the SFF FFT calculates the addition and the subtraction of the butterfly at different time instants. This allows for reducing the adder complexity with respect to the SDF, by using the same adder/subtractor for both operations of the butterfly. To achieve this, the SFF FFT needs two buffers of length $L = 2^{n-s}$ at each stage. Thus, the SFF FFT reduces the number of adders by half compared to the SDF, at the expense of using twice as much memory. The rotators in an SFF FFT are the same as in an SDF FFT, i.e., the SFF allows for the same use of the radices as the SDF.

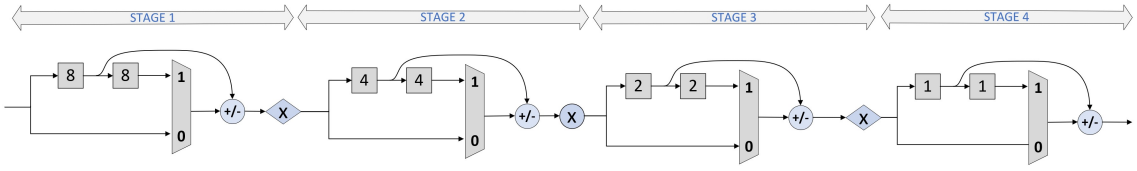


Figure 2.15: 16-point radix-2^2 SFF FFT architecture.

To understand the workflow, let us explain an SFF stage. The data arrive and are stored in the first buffer. When 2^{n-s} clock cycles have passed, the addition of the pairs of values from the input of the stage and the output of the buffer is calculated. At the same time, the second buffer is filling with data from the first buffer. Then, when the bit b_{n-s} is located at the output of the first buffer (when 2^{n-s} clock cycles have passed), the subtraction between both output buffers is calculated. Finally, the rotation is calculated, which leads to the input of the next stage.

2.5.2 Parallel architectures

Parallel pipelined FFT architectures are characterized by their high throughput. They can process P parallel samples in a continuous flow and achieve a throughput that is P times the clock frequency.

MDF

The MDF FFT architecture is the parallel version of the SDF FFT. In an MDF FFT architecture, the first stages consist of P SDF structures in parallel, whereas the last stages are used to combine the P branches. An example for $P = 2$ and $N = 16$ is shown in figure 2.16. In this case, the data order is obtained by separating the odd-indexed data and even-indexed data, as is shown in figure 2.17. This places data that differ in b_{n-s} closer in the pipeline, which allows reducing the length of the buffers.

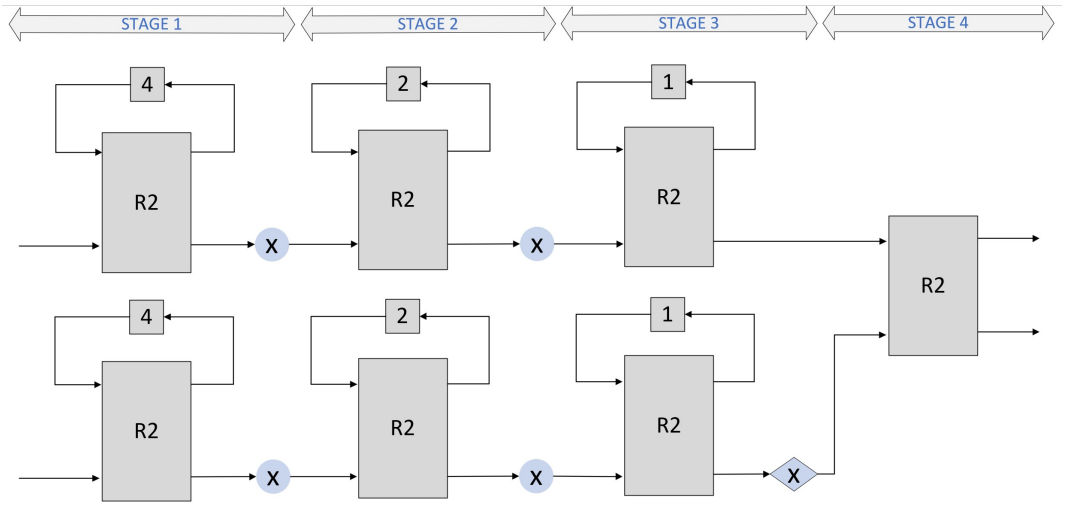


Figure 2.16: 16-point 2-parallel radix-2 MDF FFT architecture.

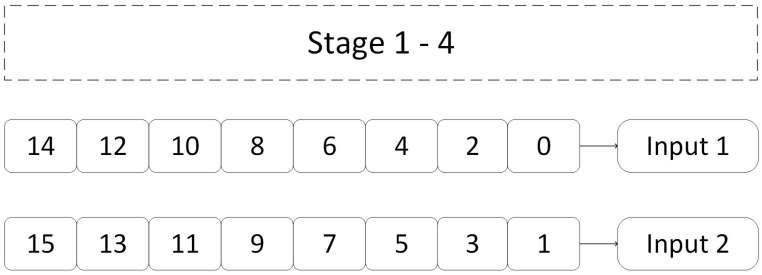


Figure 2.17: Timing of a 16-point 2-parallel radix-2 MDF FFT architecture.

MDF FFTs with higher parallelization are also possible, and they have been developed using different radices [58, 26].

In general, a P-parallel radix-2 MDF FFT uses $2P \log_2 N - P \log_2 P$ complex adders in butterflies, $P \log_2 N - P/2 \cdot \log_2 P - P$ non-trivial rotators and a total memory of $N - P$ [22]. As in SDF FFT architectures, radix-2^2 used in MDF FFT transforms the rotators in odd stages to trivial rotators, which halves the rotator complexity.

M^{2}DF

The M^{2}DF FFT architecture [26] arises as an alternative to the MDF FFT in order to increase the utilization of the butterflies and rotators by creating two data flows in opposite directions. These two data flows share the computations of the butterflies, which reach 100% utilization. To achieve this, we need to place the correct bit order in each stage, granting the data from both data flows to arrive simultaneously at stage 4 to be processed in parallel. This is possible due to the bit reversal (BR) [59] calculated for both data flows in different time moments.

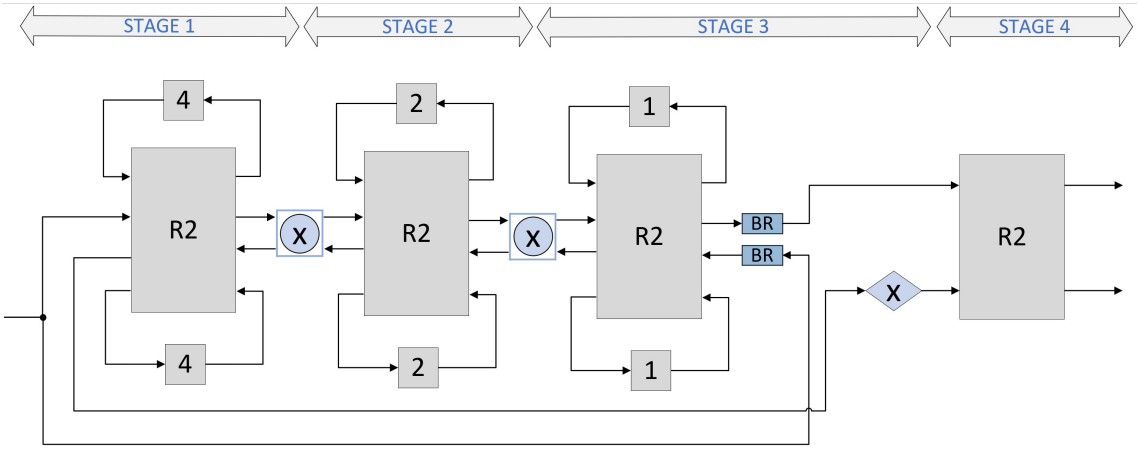


Figure 2.18: 16-point 2-parallel radix-2 M^{2}DF FFT architecture.

2.6 FPGA resources

The architecture proposed in this Master's Thesis is developed for its implementation on an FPGA. Specifically, the 7 series FPGA devices from Xilinx have been chosen, which includes the Spartan and the Virtex families [60].

2.6.1 CLBs, slices and LUTs

The logic of the 7 series FPGAs is distributed in configurable logic blocks (CLBs) [61]. Each CLB is formed by two slices. Each slice contains four LUTs and eight flip-flops, as well as multiplexers and arithmetic carry logic.

The LUTs in 7 series FPGAs can be configured as either a 6-input LUT with one output, or as two 5-input LUTs with separate outputs. Each 5-input LUT output can optionally be registered in a flip-flop. Furthermore, the LUTs can be used as distributed 64-bit RAM, as 32-bit shift registers (SRL32) or as two SRL16.

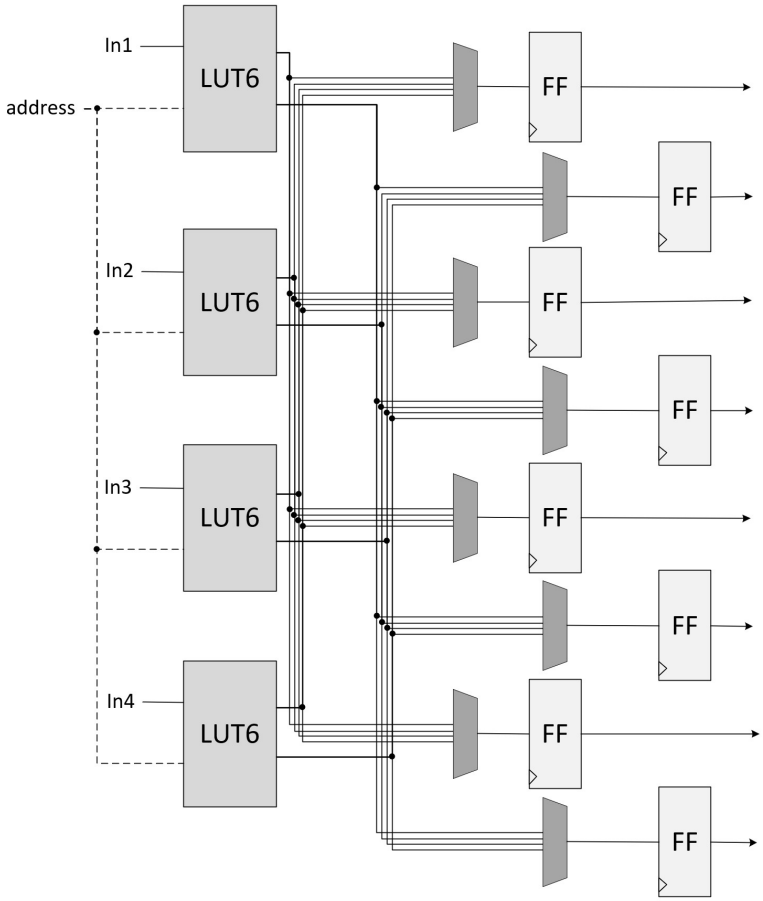


Figure 2.19: Simplified scheme of an 7 series FPGA SLICE.

This feature will be key to the architecture module selection, as will be discussed in Section 3.1.2.

However, all LUTs are not able to be used as memory. There are two types of slices: SLICEM and SLICEL. The difference between them is only the LUT as memory option, which is available in the SLICEM, but not in the SLICEL.

In figure 2.19, we can observe a simplified scheme of a slice. The LUT6s located in SLICEMs have two inputs, the data input and the read/write address for locating the data when the LUT is configured as memory. However, in the case of SLICEL, this second input is not available. Due to this, in figure 2.19 the input address is connected with dashed lines. Additionally, the SLICEM introduces more arithmetic logic than SLICEL to configure the LUTs and to control the data storing when LUTs are shift register or distributed RAM. For more detailed information and the complete figure of the slices, the user guide [61] is provided.

2.6.2 Block RAM

The block RAM in Xilinx 7 series FPGAs stores up to 36 kbits of data and can be configured as either two independent 18 Kb RAMs, or one 36 Kb RAM. Each 36 Kb block of RAM can be configured as a 64K x 1, 32K x 1, 16K x 2, 8K x 4, 4K x 9, 2K x 18, 1K x 36, or 512 x 72 in simple dual-port mode. This last configuration will be used in our architecture.

2.6.3 DSPs

The 7 series FPGAs also include DSP48E1 slices [1] to calculate more complex mathematical operations such as multiplications, pattern detection or bitwise logic functions. From the perspective of the FFT, DSPs are commonly used to calculate rotations by means of a complex multiplication. In figure 2.20, a simplified scheme of the DSP slice is shown. It consists of a multiplier followed by an accumulator. At least three pipeline registers are required for both multiply and multiply-accumulate operations to run at full speed. The multiply operation in the first stage generates two partial products that need to be added together in the second stage. Furthermore, in this second stage an additional input port (C) that can be included in the operation.

The DSP48E1 slice additionally contains a pre-adder and two dual registers that allow us to have more pipelining. In case of use them, the DSP would take 5 cycles to obtain the output.

The DSP has three control signals: INMODE, OPMODE and ALUMODE. The control bits of the INMODE select the functionality of the pre-adder, the A, B and D input registers. Therefore, the dual A and B registers, can be configured with one or two pipeline registers, just changing the value of one bit on the control signal. On the other hand, the OPMODE and ALUMODE are used to control the multiplexer output and operation of the second stage add/sub/logic unit.

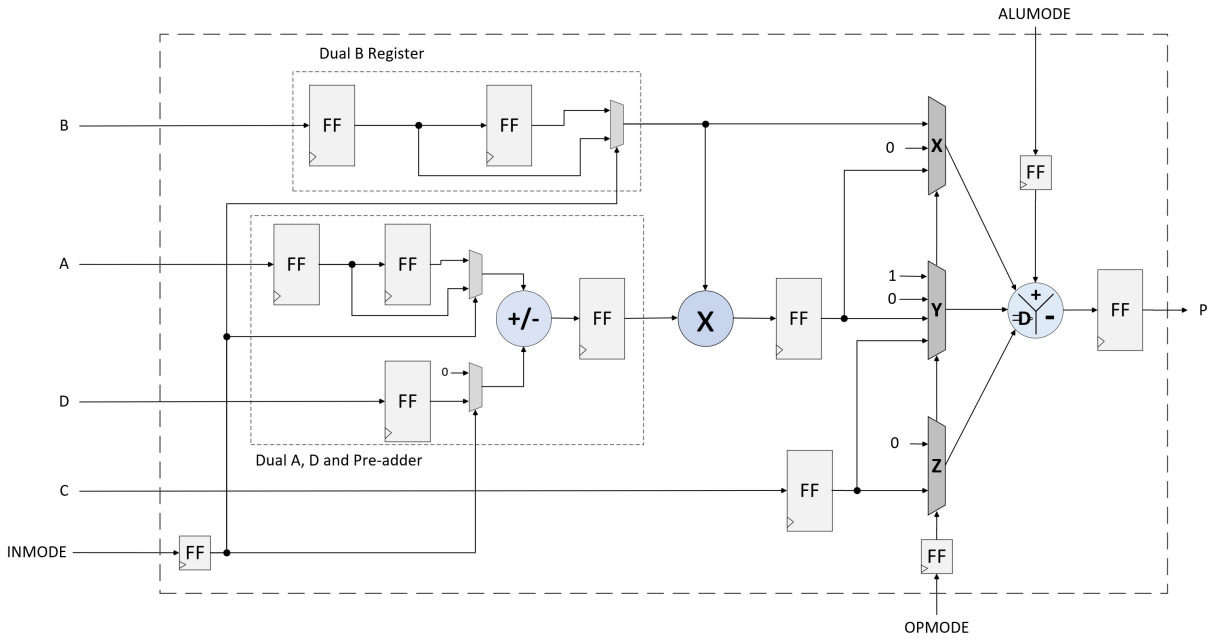


Figure 2.20: Simplified scheme of a DSP48E1.

Chapter 3

Proposed approach

This chapter explains the proposed design of a parallel pipelined FFT architecture based on SFF and SDF. The architecture has been developed for 1024 points, radix-2^4 and a parallelization of 4.

First, in section 3.1 a proposed mapping is provided to allocate all the FFT components into the 7 series FPGAs in order to obtain an architecture with a reduced area and low power consumption. In section 3.2, the architecture is separated in modules depending on the rotation type and register length and their implementation is explained in section 3.3. Therefore, the architecture will be swapping between MDF and MFF, using different type of memory. In section 3.4, the radix and the specific algorithm to implement the FFT is selected, depending on the distribution and complexity of the rotations. After that, the coefficient's memory is designed and integrated together with all modules previously implemented. Finally, the implementation of control signals needed in each module and the memory control are explained.

3.1 Proposed mapping

3.1.1 Adders and multiplexers

In Xilinx 7 series FPGAs, each bit in an adder is implemented partly in a LUT and partly in a dedicated logic at the LUT output, within a slice. This output logic is used to control the carry between LUT adders. The 7 series FPGAs architecture is based on six input LUTs (LUT6) that are composed by two LUT5.

In figure 3.1 we can observe that for an addition/subtraction between A and B, some of the inputs are unused. As it has been demonstrated in [24], some other logic can be merged on the input side of the adder by using these free inputs of the LUT.

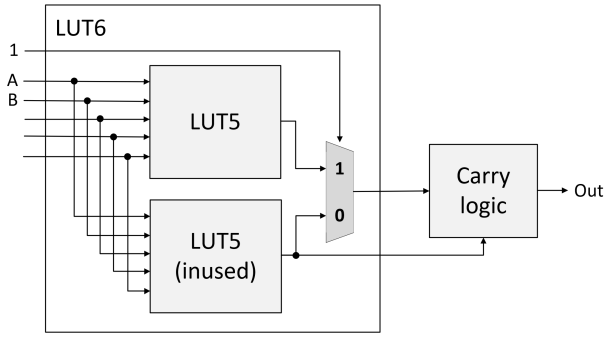


Figure 3.1: Simplified scheme of a full adder implementation in 7 series FPGA

Thus, it is possible to merge one adder with 2:1 multiplexers on both its inputs to a LUT6.

3.1.2 Memory

Each FFT stage needs memory to hold the data arriving at the stage until the expected index is at the input and the butterfly can do the calculations. This memory can be implemented in the FPGA by using LUTs or block RAMs. As we discussed in chapter 2.6, two buffers to delay up to 16 clock cycles can be mapped in the same LUT. For buffers of length 32, a single LUT has to be used. These options are per output bit, e.g., a delay of 4 clock cycles for data with a word length of 16 bits would be mapped in 8 LUTs. However, for a delay of 32 clock cycles with the same word length of 16 bits, 16 LUTs are needed, one per output bit.

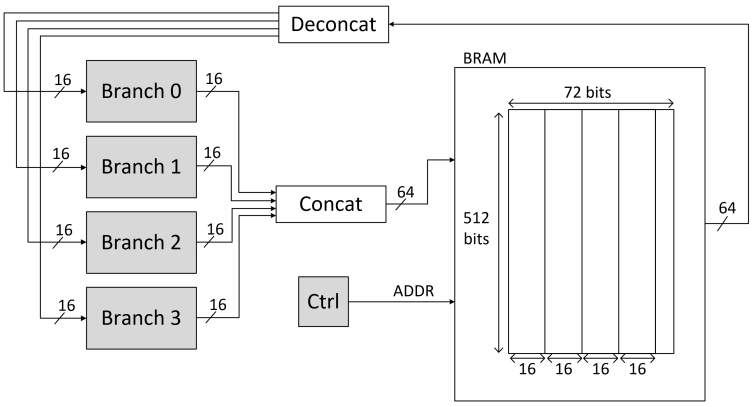


Figure 3.2: Implementation of 4 branches buffers on BRAM 512x72 bits

For larger buffers, several LUTs are necessary to implement each register, and it would produce a high consumption of resources. For this reason, for buffer length 32 and larger, block RAMs are considered instead . The 512x72 configuration of the 7 series block RAM is used in this Master's Thesis. Taking into account that the

proposed architecture process 4 data in parallel and the data has 16 bits, we can map the memory for the 4 branches in the same BRAM. For example, for a buffer of length 256 on each branch, the BRAM will be implemented as is illustrated in figure 3.2, taking the data of the 4 branches data in serie and allocating them in the same address of the BRAM. This address is incremented each cycle and reset after L cycles. After the first L cycles, the address is reset to 0 and the data in each address is read and overwritten with new data every clock cycle. The possible addresses are from 0 to $L-1$, in this example, 0 to 255. Thus, four branches that need a buffer up to 512 cycles each one will be mapped in two BRAMs, one for the real part and one for the imaginary part.

3.1.3 Rotations

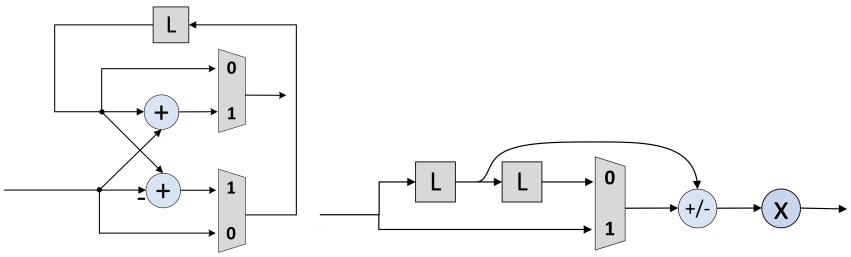
To take advantage of all the resources of the FPGA, it has been decided to implement the FFT complex rotations by using complex multipliers. In this case, DSP blocks are used to implement the rotators and are optimized in this Master's Thesis to allocate other logic of the architecture, reducing the total area of our system. For its implementation, the Xilinx 7 series FPGAs have DSP48E1 blocks. A Virtex-7 DSP48E1 block contains a hard 18x25 bit multiplier followed by a 48-bit accumulator. Thus, each complex rotation is implemented using four real multipliers requiring four DSP48E1 units, following the description in section 2.3.1. In addition, due to the architecture of these DSP blocks with a pre-adder before the multiplier, some logic of the proposed architecture is mapped into the DSP. The CORDIC and CCSSI algorithms explained in section 2.3.1 are proposed for future works to compare and update the architecture proposed in this Master's Thesis.

Trivial rotations are implemented with LUTs, taking into account that they consist in exchanging the real and imaginary parts and changing the sign of the data.

3.2 Architecture modules selection

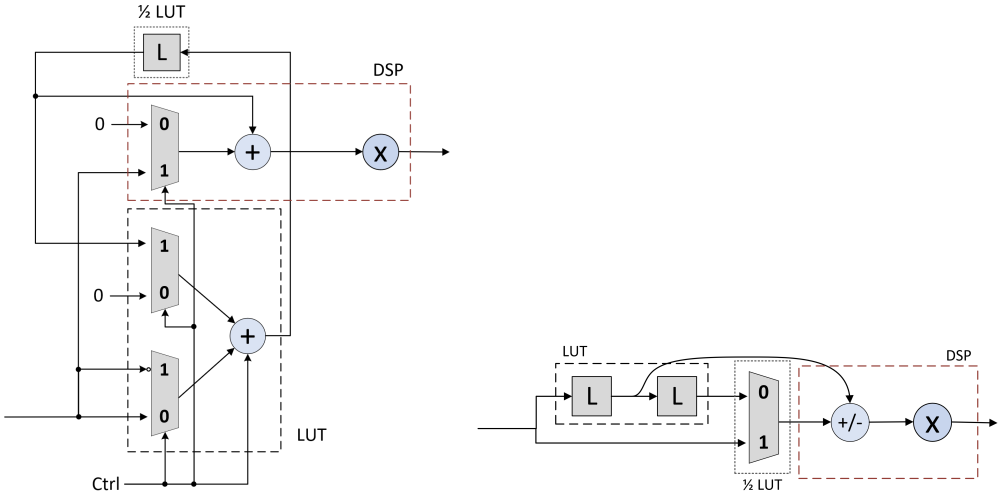
In order to simplify the development of the architecture, the design is divided into several modules. Each module is based on MDF or MFF (the proposed parallel approach for the SFF architecture). This module selection is determined by the memory size and the rotation needed at each stage of the FFT architecture.

As we can observe in figure 3.3, an SDF stage needs more logic in terms of multiplexers and adders/subtractors than the SFF stage. However, in terms of memory, the SFF needs twice the memory of the SDF. The four parallel branches of MDF are based on SDF', an optimized version of conventional SDF [24]. This SDF' mapping allows us to integrate the multiplexers into a LUT together with the



(a) (b)

Figure 3.3: Single stage of SDF and SFF. (a) SDF. (b) SFF.



(a) (b)

Figure 3.4: Mapping of a single stage of complex SDF' and SFF. (a) SDF'. (b) SFF.

adder, as is shown in figure 3.4. For more information about this optimization, refer to [24].

In stages with trivial rotations, for memory sizes from 1 to 16, the MFF is chosen. For this case, only 5 LUTs are needed per output bit, 2 LUTs for SRLs, 2 LUTs for adders-mux of the butterfly and 1 LUT for two muxes of the trivial rotation. In the case of SDF, only one LUT is needed for the buffers, but 4 LUTs are needed to map the 4 adders-mux. For memory sizes of 32 bits, the results with both architectures are similar, so the proposed MFF is selected. However, for memory sizes higher than 32 bits, the BRAM is needed and, prioritizing the minimum use of them, the MDF is selected.

In stages with complex rotations, as it can be seen in figure 3.4, both architectures can reduce the number of logic by mapping some logic into the pre-adder on DSP block. In both cases, this mapping gets an stage with one LUT and a half per output bit when shift registers up to 16 bits are used (SRL16). However, for higher buffer lengths the MDF' needs less logic, so it is selected. Taking into account that

Table 3.1: INMODE functions for the DSP48E1.

INMODE[3]	INMODE[2]	INMODE[1]	INMODE[0]	Multiplier A Port
0	0	0	0	A2
0	0	0	1	A1
0	0	1	0	Zero
0	0	1	1	Zero
0	1	0	0	D+A2
0	1	0	1	D+A1
0	1	1	0	D
0	1	1	1	D
1	0	0	0	−A2
1	0	0	1	−A1
1	0	1	0	Zero
1	0	1	1	Zero
1	1	0	0	D−A2
1	1	0	1	D−A1
1	1	1	0	D
1	1	1	1	D

the architecture implementation would only change in the buffer mapping, MDF is selected for all memory sizes.

Nevertheless, there is a single case where the MFF can be used to implement stages with complex rotations. By analysing the DSP block, we can see that the pipelining registers on the pre-adder block can be controlled and exploited to map into them the registers of the SFF stage. This case is possible when the SFF stage needs registers of size 1. Thus, as it can be observed in figure 3.5, we can map both registers of an SFF stage into the A1 and A2 registers by controlling the control signal INMODE and clock enable signals of these registers. The INMODE signal controls the operation that is made in the pre-adder step of the DSP, as is shown in table 3.1. This will be clearly explained in section 3.3.3 and 3.3.4.

Finally, in table 3.2 the resources necessary to implement the complex and trivial stages of MDF and MFF architectures with different memory sizes are exposed. The details for these modules are explained in the next section.

3.3 Implementation of MFF and MDF modules

3.3.1 MFF module for $L \leq 32$ and trivial rotations

Figure 3.6 shows the SFF mapping for trivial stages with shift registers of length L in the range 1-32. First, the buffer starts to fill. When the buffer is full after L

Table 3.2: MFF and MDF resources consumption comparison.

Architecture	Stage rotation	Buffer size	LUTs	BRAM	DSPs
MFF	Trivial	1-16	$5P$	-	-
		32	$7P$	-	-
		>32	$3P$	4	-
	Complex	1	0	-	P
		2-16	$3P$	-	P
		32	$5P$	-	P
		>32	$2P$	4	P
MDF	Trivial	1-16	$6P$	-	-
		32	$7P$	-	-
		>32	$5P$	2	-
	Complex	1	$3P$	-	P
		2-16	$3P$	-	P
		32	$4P$	-	P
		>32	$2P$	2	P

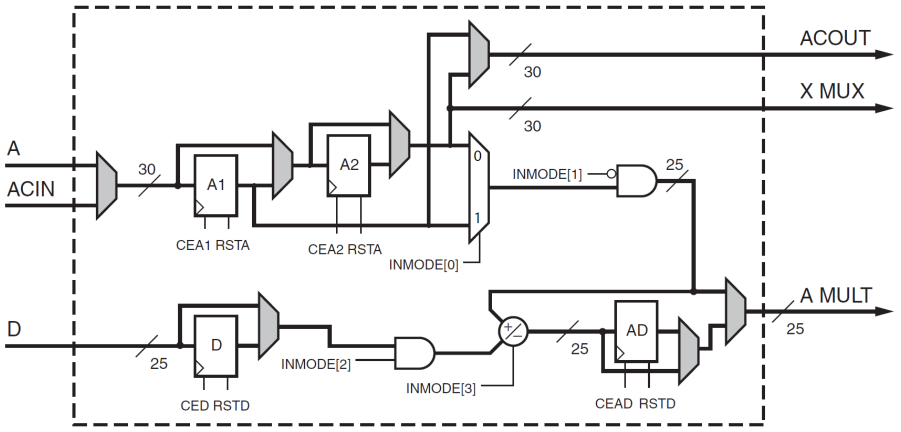


Figure 3.5: Dual A, D and Pre-adder logic from [1].

clock cycles, the addition starts to take place between the buffer r1 and the input, setting the MUX1 input control signal ctrl_S to '1'. Then, after other L clock cycles, the subtraction between the buffers r2 and r1 is calculated by setting ctrl_S to '0'.

In the FFT, trivial rotations take place at the second part of the subtraction. These rotations consist of exchanging the real and imaginary parts and changing the sign of the data. Therefore, the rotation can be implemented by multiplying the real part by -1 and swapping the lines. However, to reduce the use of logic required by this multiplication, we can integrate the exchange of the real and imaginary part and the sign of data change by including an extra multiplexer per line, as is shown in figure 3.7.

After $2L + L/2$ clock cycles (second half of the subtraction), the rotation control signal *ctrl_rot* is set to '1'. This control signal is also included as an input of the

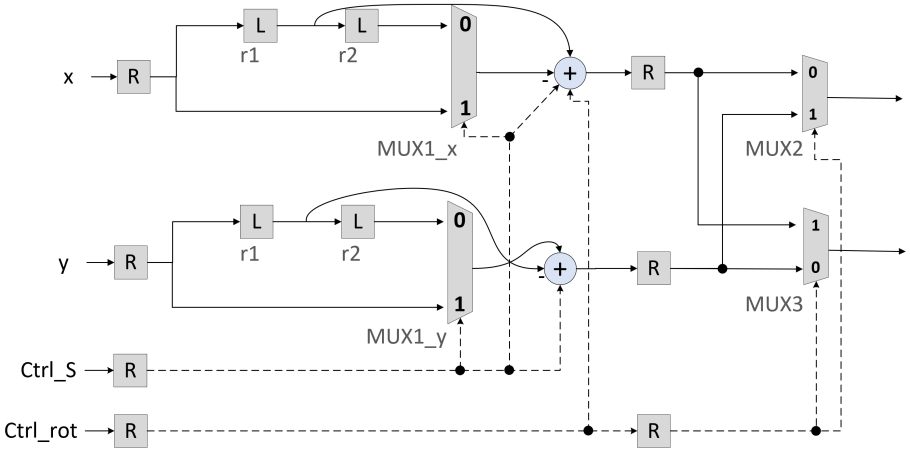


Figure 3.6: MFF branch with trivial rotations for $L \leq 32$.

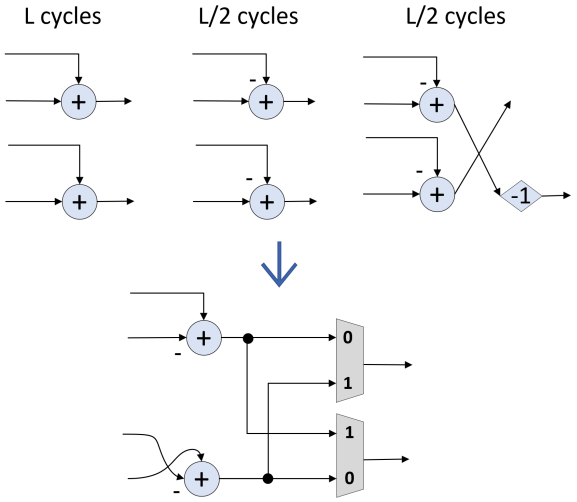


Figure 3.7: Trivial rotation on MFF FFT.

adder/subtractor of the real part, in order to change the sign of data. Thereby, when *ctrl_rot* is '1', the positive input and the input to be subtracted are exchanged. As a result, the trivial rotation only requires one more multiplexer per line.

Finally, in equation (3.1) the latency is calculated. This latency defines the clock cycles needed to obtain the first valid value.

$$\text{Latency } (\text{MFF}_{triv}) = 1 + L + 1 \text{ clock cycles.} \tag{3.1}$$

This latency corresponds to the buffer length plus the delays of the input register and the register placed after the butterfly operation.

3.3.2 MDF module for $L > 32$ and trivial rotations

In figure 3.8, a branch of the MDF module for $L > 32$ and trivial rotations is illustrated. First, the control signal *ctrl_S* is '0', so the buffer is filled with inputs through MUX3. After L cycles, *ctrl_S* is set to '1', so the addition between the input (through MUX1) and the output of the buffer takes place on the Adder2 (Adder2_x for real data and Adder2_y for imaginary data). At the same time, the buffer is filled by the subtraction performed in Adder1. Then, after L cycles, when the sum is over, the control signal goes low, so the buffer is filled again with the input while the subtraction stored in the buffer is passed through the Adder2.

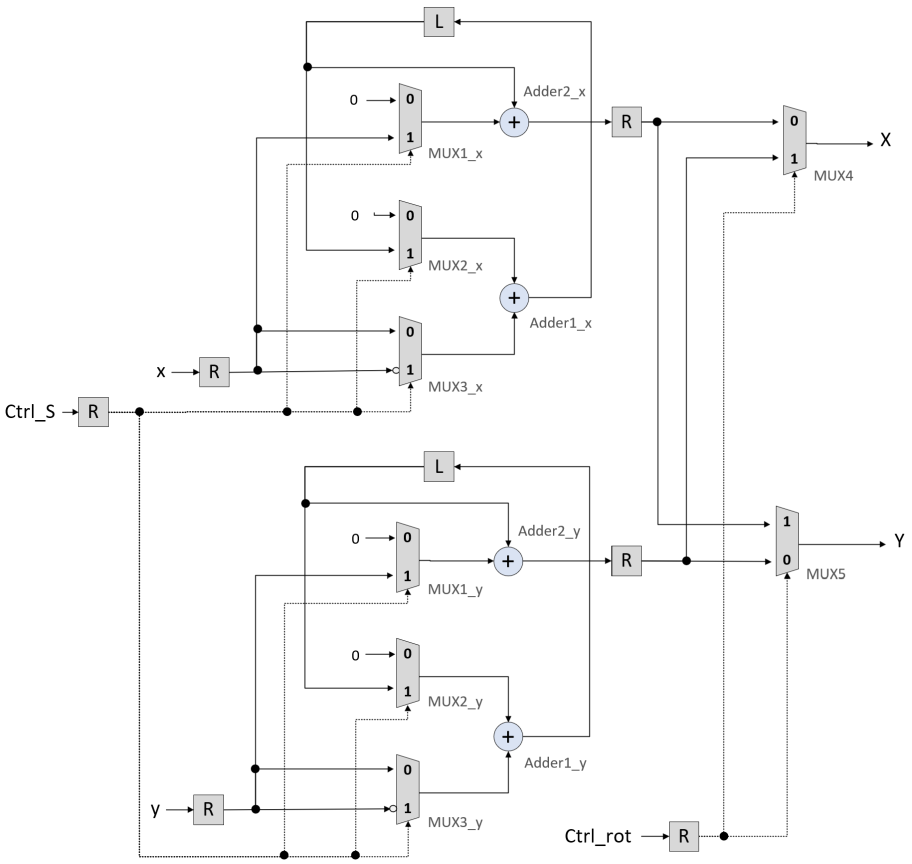


Figure 3.8: MDF branch with trivial rotations and $L > 32$.

Then, when the second half of the subtraction passes through the Adder2, the *ctrl_rot* signal is set to '1'. With this, the subtraction of the real part is sign changed in MUX4 and the lines between real and imaginary part are swapped.

In this module, the buffers are mapped into two BRAMs, for real and imaginary lines. The Adder1 output of each line is concatenated with other parallel branches of the same line and stored in a BRAM. Additionally, to read from the BRAM, each value stored in each RAM address is separated and connected to its corresponding branch. Thus, every clock cycle, we read from an address and write in the same

address.

Finally, the latency is calculated as

$$\text{Latency } (\text{MDF}_{triv}) = 1 + L + 1 \text{ clock cycles.} \tag{3.2}$$

This latency corresponds to the buffer length plus the delays of the input register and the register placed after the Adder2 operation.

3.3.3 MDF module for $L \geq 2$ and complex rotations

In figure 3.9, the MDF module for complex stages and buffer length equal or larger than 2 is illustrated. In this case, the upper adder and MUX1 are mapped into the pre-adder phase of the DSP block, which is used to allocate the complex rotation. To do this, the adder and mux have to be duplicated. This does not produce more logic consumption due to the use of the whole DSP anyway.

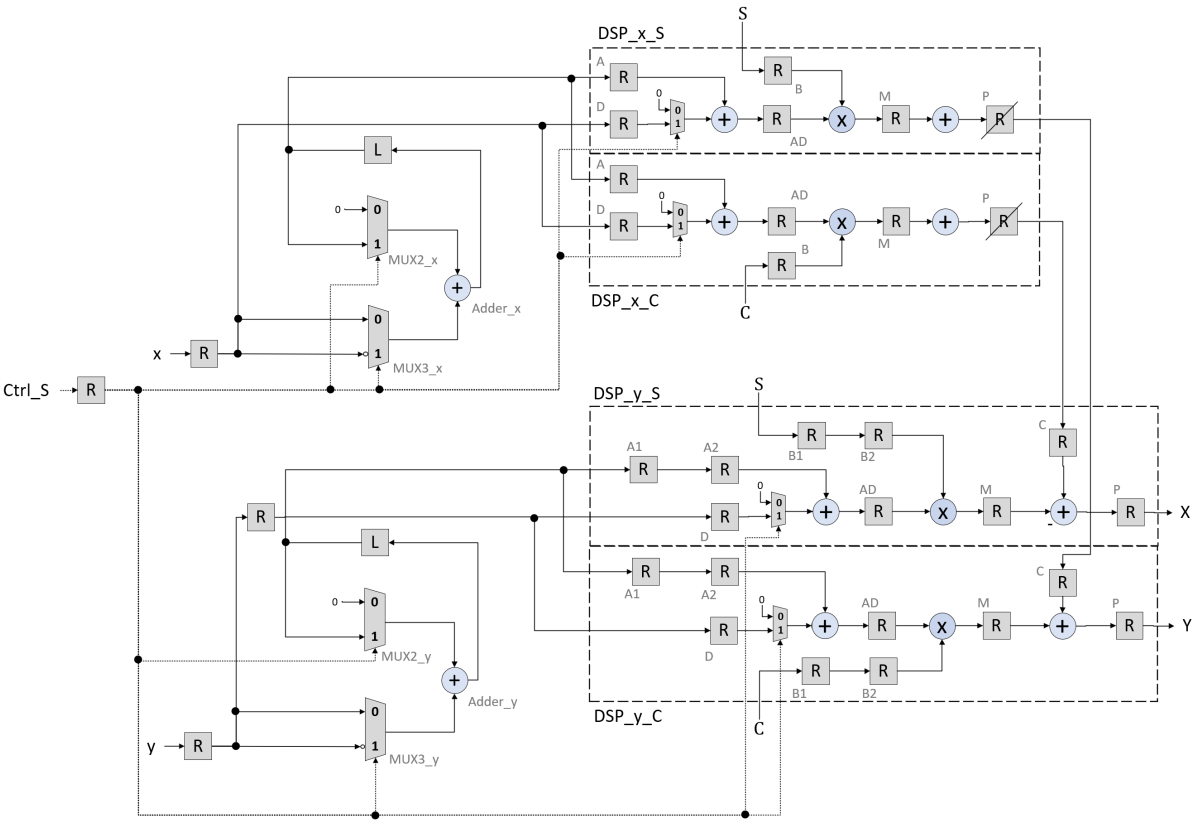


Figure 3.9: MDF branch with complex rotations and $L \geq 2$.

As it has been mentioned in section 2.3.1, the complex rotations are implemented using four DSP blocks. To obtain the corresponding real and imaginary outputs,

it is necessary to add and subtract the results obtained from the DSPs. Therefore, to adjust the timing, the two upper DSPs use only one pipeline stage on input A and on input B, where the rotation coefficients are found. In addition, in order to synchronize the upper DSPs with lower DSPs, the P register is disabled, registering the result of DSP_x_S and DSP_x_C directly on the register C of DSP_y_S and DSP_y_C, where the result of the complex multiplier is calculated.

The circuit works as follows.n First, the buffer starts to fill by maintaining the control signal $Ctrl\_S$ to zero. Then, after L clock cycles, the $Ctrl\_S$ signal is set to one, causing the subtraction to be carried out and stored inside the buffer. At the same time, the addition is taking place within the DSP. To achieve this, the addition between A2 and D is calculated. Thus, as is detailed in table 3.1, INMODE needs to be "0100". When the addition is finished, so L cycles have passed, $Ctrl\_S$ is set to 0, the buffer starts to fill with input values again and the subtraction takes place at the output of the buffer. This subtraction is connected to the A port of the DSP, so the operation in the pre-adder needs to be a bypass for this input. To do this, INMODE has the value 0000.

By analysing the used INMODE values, the only bit that varies depending on the operation to be carried out is bit 2, being '1' when the addition is calculated and 0 when the subtraction must be obtained by bypassing this entry. Therefore, to control the operation in the pre-adder step of DSPs, $ctrl\_S$ is directly connected with bit 2 of INMODE.

After the pre-adder step, the multiplication is carried out in two stages. In the first stage, the multiply operation between the pre-adder result and the multiplier coefficient allocated in the input register B generates two partial products. In the second stage, those partial products need to be added together. This second stage is controlled by control signals OPMODE and ALUMODE. For the two upper DSPs, the OPMODE and ALUMODE are adjusted to simply obtain the addition of these partial products. In the case of the DSP_y_S, which obtains the real part output, the ALUMODE is adjusted to calculate the subtraction illustrated in figure 3.9.

In terms of memory, for buffer lengths larger than 2 and less or equal to 32, the SRLs are used. For buffer lengths larger than 32, the implementation of the buffer is equal to the buffer implementation in the trivial MDF for $L > 32$, using a BRAM for each line (real and imaginary part), as could be observed in figure 3.2.

Finally, the latency of this module is

$$\text{Latency}(\text{MDF}_{complex}) = 1 + L + 5. \tag{3.3}$$

This latency corresponds to the buffer length plus the delays of the input register and the DSP latency, which is 5 clock cycles.

3.3.4 MFF module for $L = 1$ and complex rotations

In figure 3.10, a branch of the MFF with complex rotations is illustrated. For this module, all necessary resources are mapped into the four DSPs used to implement the complex rotation. To do this, the four DSPs operate with the two pipeline registers of input registers A and B. The registers A1 and A2 are used to map the two registers of the SFF architecture that control the order of data to be added or subtracted.

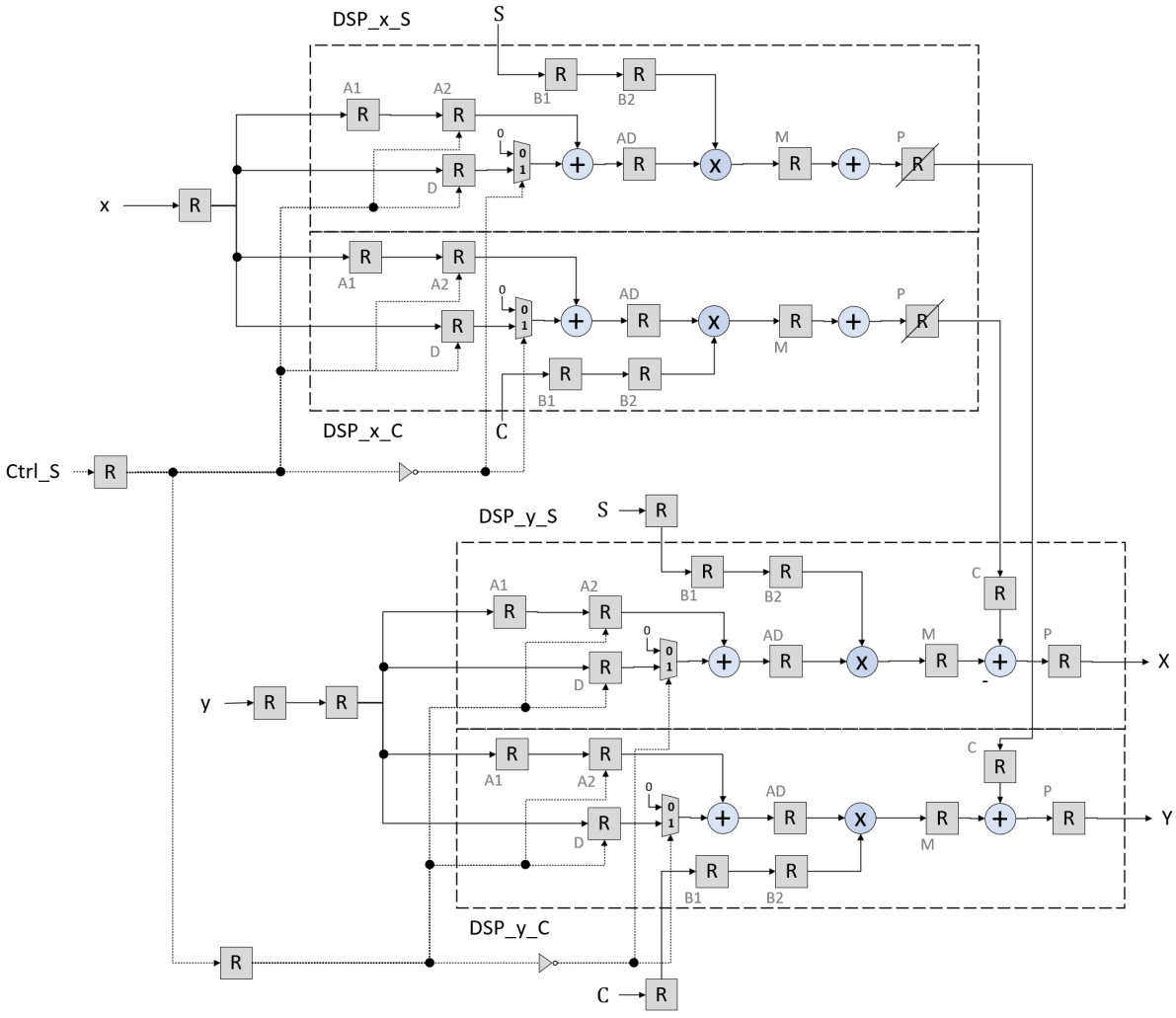


Figure 3.10: MFF branch with complex rotations and $L = 1$.

The DSPs are configured by the control signal *Ctrl_S* to decide the operation to be calculated every clock cycle. Taking into account that this module is for stages with a single delay registers, the operation is swapping between addition and subtraction every clock cycle. As we can observe in table 3.3, the registers A2 and D need to hold the data during two cycles to calculate both addition and subtraction operations. To achieve this, both registers are continuously enabled/disabled using

Ctrl_S. When *Ctrl_S* is '1', in the next cycle the buffer A2 is updated with data stored in A1, and D is updated with the input data, so they are added. When the addition is being calculated, the *Ctrl_S* signal is '0', so the next cycle the data is unchanged, and the subtraction is carried out.

Table 3.3: Timing of the MFF module for $L = 1$ and complex rotations.

Time	Input	*Ctrl_S*	A1	A2	D	Output operation
t0	0	-	-	-	-	-
t1	1	1	0	0	0	-
t2	2	0	1	0	1	A2+D
t3	3	1	2	0	1	A2−D
t4	4	0	3	2	3	A2+D
t5	5	1	4	2	3	A2−D
t6	6	0	5	4	5	A2+D
t7	7	1	6	4	5	A2−D

As for the complex MDF, *Ctrl_S* is directly connected to the bit 2 of INMODE, but with a negation on it. This determines the calculation of the addition or subtraction. Observing the figure 3.10, *Ctrl_S* is used one cycle earlier for A2 and D than for INMODE, represented with a multiplexer. Therefore, taking advantage of the fact that *Ctrl_S* toggles every cycle, instead of adding an extra register to delay *Ctrl_S*, the control signal is negated at the input of the INMODE.

ALUMODE and OPMODE work in the same way as in MDF module for complex data. However, in the case of the proposed MFF module for complex data, the four DSPs have full pipelining on inputs A and B. Therefore, the delay needed between the upper and lower DSPs on figure 3.10 to obtain the last operation of the complex multiplier is obtained by adding an extra register on the imaginary input y and coefficient inputs C and S of DSP_y_S and DSP_y_C.

Finally, the latency for the MFF complex is calculated as

$$\text{Latency}(\text{MFF}_{complex}) = 2 + L + 5. \tag{3.4}$$

3.4 MDF-MFF architecture implementation

3.4.1 Radix selection

The first step to obtain the complete architecture of the MDF-MFF is the selection of the radix. Starting from a 1024-point FFT (10 stages), radix 2^4 and 2^5 are the options with the lowest number of rotations of greater complexity. The exponent indicates every how many stages the more complex rotations appear. Therefore,

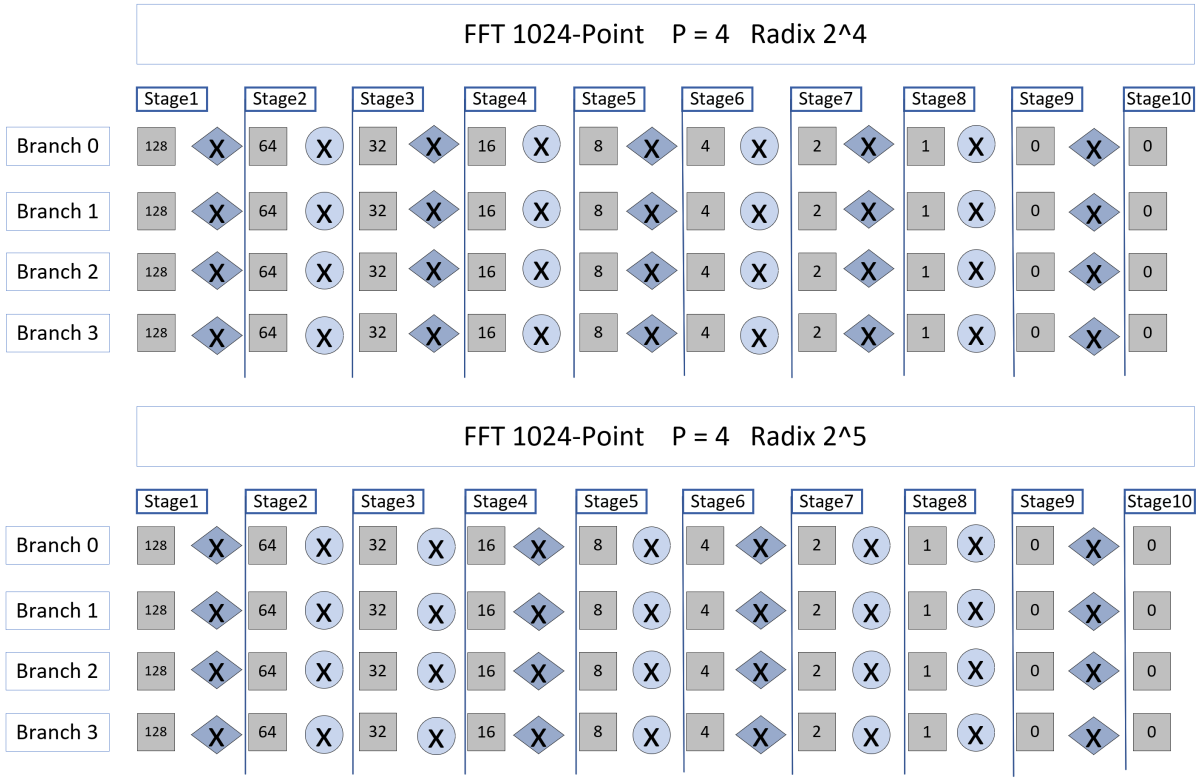


Figure 3.11: Rotation and buffer distribution for FFT 1024-point with radices 2^4 and 2^5.

radices like 2^2 or 2^3 would imply a larger number of stages with more complex rotations within the 10 stages of our FFT.

Figure 3.11 shows a comparison of the distribution of rotations in a 1024-point FFT using a radix-2^4 and radix-2^5, where $\otimes$ represent complex rotators and diamond-shaped rotators represent trivial rotators. In the figure, we can observe that radix-2^4 presents fewer stages with complex rotations and, therefore, will reduce the complexity and resource consumption of our architecture. For this reason, radix-2^4 is selected for the implementation.

However, there are several possible algorithms to implement the radix-2^4 FFT, depending on the stages where the more complex rotation are placed. Following the methodology explained in section 2.2.1, we can obtain the triangular matrices and then the rotations to be calculated at each stage. Based on this, we obtain four possible algorithms for the radix-2^4, as is illustrated in the triangular matrices of figure 3.12.

By obtaining the rotation equations at each complex stage, we can select the algorithm that is better adapted to our architecture. To analyse the four options, we can separate the four algorithms in two groups, one group with the most complex rotations in stage 4 and the other group with the most complex rotations in stage

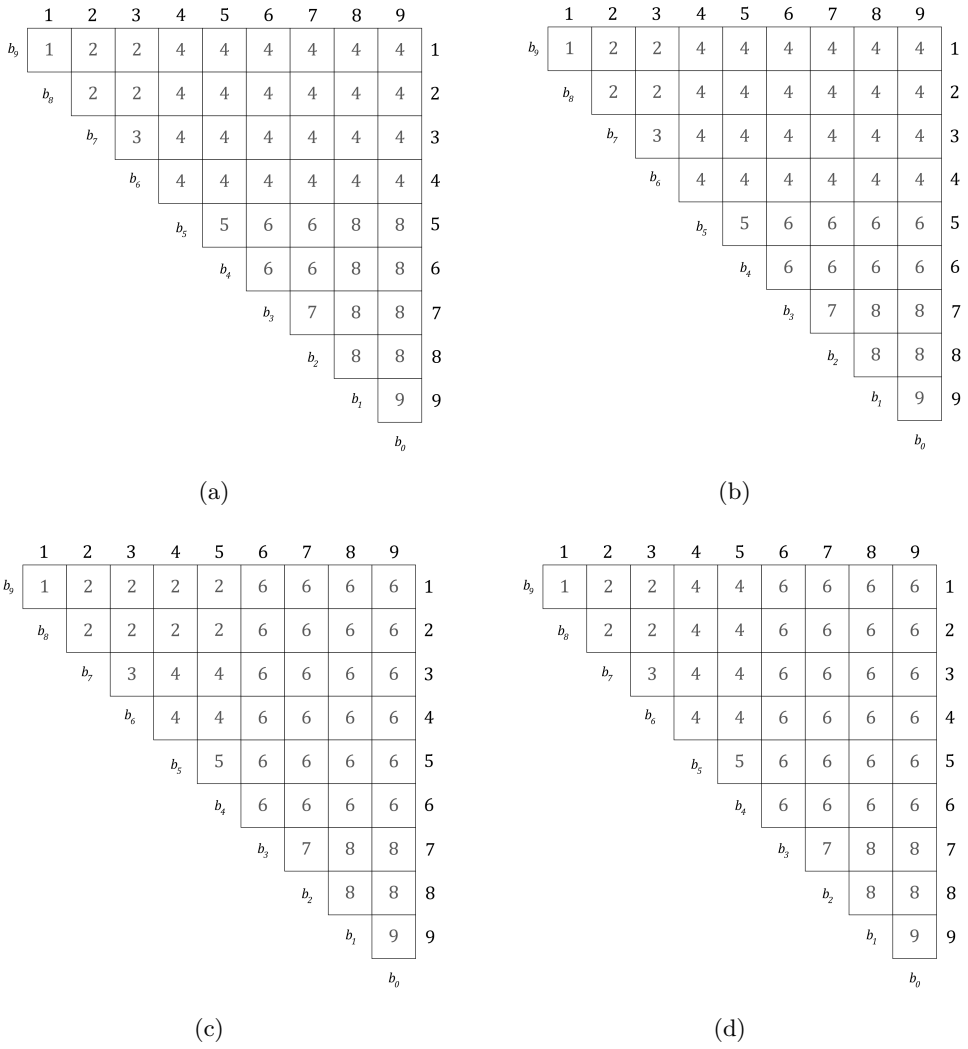


Figure 3.12: Triangular matrices for radix-2^4 1024-point FFT.

6. The rotation equations for the first group (figures 3.12(a) and 3.12(b)) are:

$$\begin{aligned}
&\phi_2(a,b) \equiv [b_8 b_9] \cdot [b_7 b_6 000000] \rightarrow W_{16} \\
&\phi_4(a,b) \equiv [b_6 b_7 b_8 b_9] \cdot [b_5 b_4 b_3 b_2 b_1 b_0] \rightarrow W_{1024} \\
&\phi_6(a) \equiv [b_4 b_5 0000] \cdot [b_3 b_2 00] \rightarrow W_{16} \\
&\phi_6(b) \equiv [b_4 b_5 0000] \cdot [b_3 b_2 b_1 b_0] \rightarrow W_{64} \\
&\phi_8(a) \equiv [b_2 b_3 b_4 b_5 0000] \cdot [b_1 b_0] \rightarrow W_{64} \\
&\phi_8(b) \equiv [b_2 b_3 000000] \cdot [b_1 b_0] \rightarrow W_{16}.
\end{aligned} \tag{3.5}$$

The rotation equations for the second group (figures 3.12(c) and 3.12(d)) are:

$$\begin{aligned}
&\phi_2(c) \equiv [b_8 b_9] \cdot [b_7 b_6 b_5 b_4 0000] \rightarrow W_{64} \\
&\phi_2(d) \equiv [b_8 b_9] \cdot [b_7 b_6 000000] \rightarrow W_{16} \\
&\phi_4(c) \equiv [b_6 b_7 00] \cdot [b_5 b_4 0000] \rightarrow W_{16} \\
&\phi_4(d) \equiv [b_6 b_7 b_8 b_9] \cdot [b_5 b_4 0000] \rightarrow W_{64} \\
&\phi_6(c,d) \equiv [b_6 b_7 b_8 b_9] \cdot [b_5 b_4 b_3 b_2 b_1 b_0] \rightarrow W_{1024} \\
&\phi_8(c,d) \equiv [b_2 b_3 000000] \cdot [b_1 b_0] \rightarrow W_{16}.
\end{aligned} \tag{3.6}$$

The twiddle factor indicated in both equations (3.5) and (3.6) is directly related to the coefficients needed in that stage. For example, the four algorithms have a stage with a twiddle factor W_{1024} of 1024 that corresponds to four memories of 1024 coefficients, one per branch. However, due to the parallelization of 4, the memory needed for rotations that depend on bits b_0 and b_1 is reduced. The data indexes in each branch have the b_0 and b_1 bits as constants, so the rotation obtained by the equations exposed depends on less bit variations and, consequently, those stages need fewer coefficients to do the rotations. This is represented in figure 3.13.

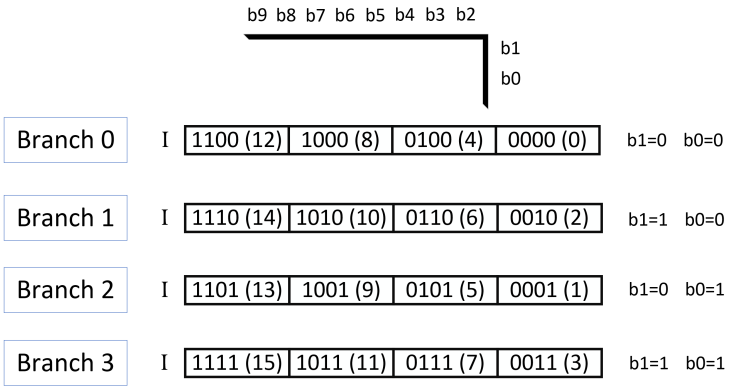


Figure 3.13: b_0 and b_1 indexes in each parallel branch of the proposed architecture.

In stage 8, only 3 branches need the complex rotation in both groups of algorithms. Observing the figure 3.13 and equations (3.5) and (3.6), we can see that the rotation in the first branch is always 0. This is because the right operand of the multiplication of the complex rotation only depends on index bits b_0 and b_1, which are 0.

Taking into account the possible optimizations due to bits b_0 and b_1, the algorithms (a) and (b) allow us to implement the coefficients for twiddle factors of 64 with memories of just 16 addresses. This is the case of $\phi_8(a)$ and $\phi_6(b)$, where the 64 possible coefficients values due to the first 6 bits (b_5, b_4, b_3, b_2, b_1 and b_0) are converted in only 16 coefficients, taking b_1 and b_0 as constants and taking only 4 bits to determine the rotation to be performed each cycle. Since this optimization is not possible in the second group of algorithms, algorithms (c) and (d) are discarded.

Then, by analysing the complex rotations in which algorithm (a) and algorithm (b) differ (stages 6 and 8), we can determine the best option for our architecture in terms of memory. The memory needed in each case is the following:

- Algorithm (a):
  - The rotation $\phi_6(a)$ only depends on bits b_5, b_4, b_3 and b_2, which are equal in the four branches. Therefore, the rotation coefficients will be the same in all branches. Because of that, only one memory of 16 coefficients is needed, which will be shared by all branches.
  - The rotation $\phi_8(a)$ depends on 6 bits, but due to the constants b_1 and b_0 in each branch, 16 different coefficients would be possible for the last 3 branches. The first branch does not need any rotation as explained before.
- Algorithm (b):
  - The rotation $\phi_6(b)$ depends on 6 bits, but due to the constants b_1 and b_0 in each branch, only 16 coefficients are needed for all branches.
  - The rotation $\phi_8(b)$ depends on 4 bits, but due to the constants b_1 and b_0 in each branch, only 4 possible coefficients are needed for the last 3 branches. The first branch does not need any rotation, as explained before.

Therefore, for stages 6 and 8 algorithm (a) needs 64 coefficients that would be implemented as 4 different memories (16x1 + 16x3), while algorithm (b) needs 76 coefficients that would be implemented as 7 different memories (16x4 + 4x3). For this reason, algorithm (a) is the selected to implement the complex rotations in the MFF-MDF architecture. A comparison between all options is summarized in table 3.4.

Table 3.4: Memory needed in the four radix-2^4 FFT algorithms.

Stage	Algorithm a	Algorithm b	Algorithm c	Algorithm d
ϕ_2	16x1 (W_{16})	16x1 (W_{16})	64x1 (W_{64})	16x1 (W_{16})
ϕ_4	1024x4 (W_{1024})	1024x4 (W_{1024})	16x4 (W_{16})	64x4 (W_{64})
ϕ_6	16x1 (W_{16})	16x4 (W_{64})	1024x4 (W_{1024})	1024x4 (W_{1024})
ϕ_8	16x3 (W_{64})	4x3 (W_{16})	4x3 (W_{16})	4x3 (W_{16})

3.4.2 Coefficients memory

For the generation of coefficients in the rotation memories, a Matlab script is used. With this script, a read only memory (ROM) is generated for each memory, which has 32-bit word length, 16 bits for the real part and 16 bits for the imaginary part. As is illustrated in table 3.4, we have 9 different ROMs: One for stage 2, four for stage 4, one for stage 6 and three for stage 8. These memories are:

- **Stage 2**. As explained in the previous section, the rotation coefficients in the four branches are the same, depending on bits b_8, b_9, b_7 and b_6. After generating the 256 coefficients for each stage, it was observed that, in addition to being the same coefficients between branches, the coefficients present the expected relation. Each coefficient is repeated 16 times before it changes, because the bits b_2, b_3, b_4 and b_5 of the input data index are changing during those 16 cycles, which does not influence the rotation calculation. Therefore, the ROM memory for stage 2 is reduced to 16 coefficients. To go through it, each coefficient must be read 16 times before continuing with the next one, since each coefficient is the same for each group of 16 input data.

- **Stage 4**. For this case, all coefficients are independent and there is not a possible optimization. For this stage, four ROMs are used, one ROM of 256 coefficients for each branch.

- **Stage 6**. Only one ROM of 16 coefficients is needed to implement the rotations of this stage, analogously to stage 2. However, in this case the rotation coefficients depend on the bits b_5, b_4, b_3 and b_2, so the coefficients change every clock cycle during 16 cycles. After 16 cycles, these four bits are 0 and the coefficients are repeated. For this reason, this ROM of 16 coefficients to all branches is read as a loop, sequentially getting the 16 coefficients and then starting from the beginning.

- **Stage 8**. In this case, three ROMs of 16 coefficients are needed for the last 3 branches. As for stage 6, the first 16 coefficients are different and the same sequence of 16 coefficients are repeated, so the ROM is read as a loop of 16 iterations.

3.4.3 Submodule assignation and integration

Taking into account the conditions in which each submodule is designed, the MDF-MFF architecture is built as illustrated in figure 3.14. In this figure the type of rotator ($\otimes$ for complex and diamond-shaped for trivial rotations) and the size of the buffer at each stage are shown.

To determine which designed submodule is implemented at each stage, the buffer size and the type of rotation are considered. This way, the four different submodules are placed in our architecture:

- MDF trivial with BRAM for stage 1.
- MDF complex with BRAM for stage 2.
- MFF trivial with SRLs for stages 3, 5 and 7.
- MDF complex with SRLs for stages 4 and 6.

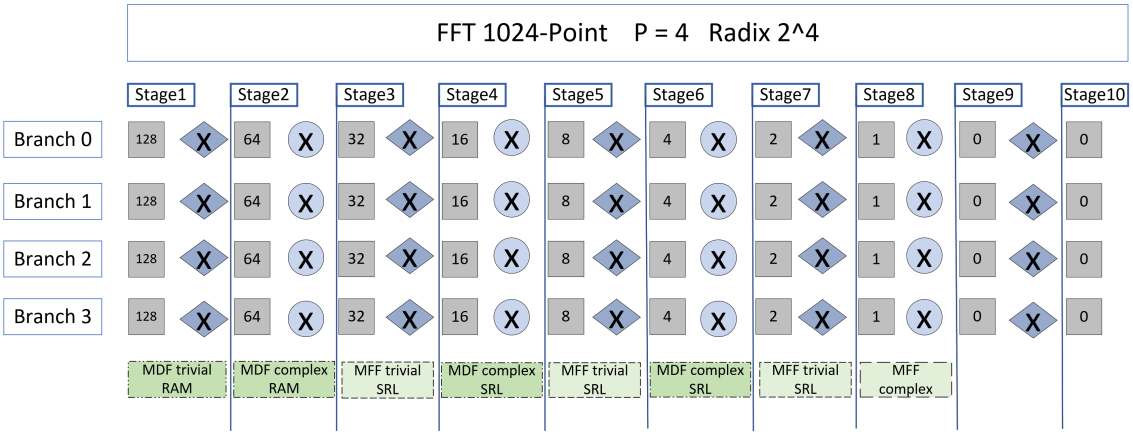


Figure 3.14: Architecture modules by stage.

- MFF complex for stage 8.

As discussed in section 3.3, all submodules need some control signals for their correct behaviour. For modules with complex rotators, a control signal *ctrl_S* is used to determine the operation to be carried out in the DSPs and, additionally, in the case of MDF, to control the external adder operation. For modules with trivial rotators, there are two control signals needed, *ctrl_S* for adders and multiplexers and *ctrl_rot* for rotator multiplexers.

To have those control signals correctly synchronized, the latency at each stage must be determined. By using the equations (3.1), (3.2), (3.4) and (3.3), the latency at each stage and in the whole system is calculated. This is illustrated in figure 3.15. Therefore, the system has a latency of 289 clock cycles.

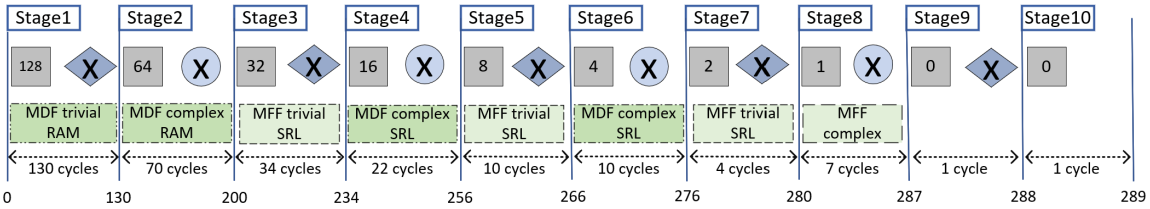


Figure 3.15: Architecture latency.

3.4.4 Control signals

To obtain all the signals needed at each stage, a counter is used. Due to the nature of this binary counter, each counter bit will toggle after 2^b clock cycles, where b is the position of the bit being analysed.

The *ctrl_S* signal toggles every L cycle in all stages. In the case of stages with trivial rotations, *ctrl_rot* is set to 1 after $2L + L/2$ cycles in order to implement the trivial rotation at the second part of the butterfly subtraction. To achieve this, the control signals will take the value of some of the bits of the counter depending on the buffer length, resulting in

$$ctrl\_S = cnt[\log_2 L], \tag{3.7}$$

$$ctrl\_rot = \text{NOT}(cnt[\log_2 L]) \text{ AND } cnt[\log_2 (L) - 1]. \tag{3.8}$$

Therefore, to take the correct counter bit at each stage for *ctrl_S* and *ctrl_rot*, the equations (3.7) and (3.8) are used. For example, in stage 5, *ctrl_S* must be set to 1 after 8 clock cycles. This occurs when the counter bit 3 is set. The *ctrl_rot* signal must be set to 1 after 20 clock cycles ($2L + L/2$) and hold high during $L/2$ cycles. This is achieved by negating the value of counter bit 3 and operating it with the counter bit 2 through an AND logic gate. Therefore, *ctrl_rot* is set to 1 after 20 clock cycles as expected and after 4 cycles ($L/2$).

However, due to the latency of each module, the beginning of every stage does not occur when the counter bits are 0. Because of it, we need to place delays for the counter bits according to the timing of the system shown in figure 3.15. The delay for *ctrl_S* at each stage is listed below:

- **Stage 1.** The counter starts at 0. Therefore, no delays are needed.
- **Stage 2.** The stage starts to work after 130 cycles. At this point, we would need to hold the control signal during 130 cycles. Nevertheless, only the bit 6 is taken into account in this stage, so the bit 7 is ignored. Because of it, only a delay of 2 cycles is needed (130-128=2).
- **Stage 3.** The stage starts to work after 200 cycles. In this case, the bits 6 and 7 are ignored. Therefore, a delay of 8 cycles is performed (200-128-64=8).
- **Stage 4.** The stage starts to work after 234 cycles. In this case, the bits 5, 6 and 7 are ignored. Therefore, a delay of 10 cycles is performed (234-128-64-32=10).
- **Stage 5.** The stage starts to work after 256 cycles. In this case, only the bit 8 is set. Therefore, from the point of view of the rest of bits, the counter is 0. Because of this, the *ctrl_S* signal at this stage takes the value of the bit 3 without any delays.
- **Stage 6.** The stage starts to work after 266 cycles. In this case, as the bits taken into account would be the 3 LSBs, which are 0 after 266 cycles, no delays are needed.

- **Stage 7.** The stage starts to work after 276 cycles. In this case, as the bits taken into account would be the 2 LSBs, which are 0 after 276 cycles, no delays are needed.
- **Stage 8.** The stage starts to work after 280 cycles. In this case, the *ctrl_S* signal toggles every cycle, so the bit 0 is taken. After 280 cycles, this bit is 0, so no delays are needed.

The delay for *ctrl_rot* is the same as for *ctrl_S*. Thus, on stages 1, 5 and 7 there are no delays. Nevertheless, a delay of 8 clock cycles will be applied to obtain *ctrl_rot* at stage 3.

3.4.5 Memory control

The memories use the same counter used for the control signals. This is used for both the RAM on the first two stages and the ROM coefficients on complex stages.

In the case of the address to be accessed by the RAM at each clock cycle, the $\log_2 L - 1$ bits of the counter are directly connected to the module. Therefore, the 7 LSBs of the counter are used as RAM address at stage 1, and the 6 LSBs are used as RAM address at stage 2.

On the other hand, an address control for each ROM with the stored coefficients is needed. The address signal configuration for each stage with complex rotations is explained below:

- **Stage 2.** In this case, each coefficient must be read 16 times before reading the next one. To achieve this, the bit 4 of the counter must be the LSB of the address control, because it toggles every 16 cycles. Then, as the ROM has 16 coefficients, four bits of the counter are used, from the seventh bit to the fourth. However, the first coefficient to be read is needed after 194 cycles, so a delay of 194 cycles would be needed. To avoid this resource consumption, the read address signal is obtained by equation (3.9). With this, the ROM address signal starts from 0 when the counter value is 192 and the reading is performed in the correct order. However, the stage 2 starts at cycle 194, so a delay register of two cycles is placed on this address signal.

$$ROM\_addr_{st2} = \{(cnt[7] \text{ XOR } cnt[6]),\ \text{NOT}(cnt[6]), cnt[5], cnt[6]\}. \quad (3.9)$$

  For a better understanding of equation (3.9), the truth table 3.5 is provided.

- **Stage 4.** The address at this stage is obtained from the 8 LSBs of the counter to go through the whole ROM. The first rotation coefficient on the four parallel branches is needed after 250 cycles, when the 16 cycles of the buffer have already passed. Because of this, a delay of 250 cycles would be needed. However,

Table 3.5: Truth table of stage 2 ROM address.

cnt (b7 to b4)	b7 xor b6	not(b6)	b5	b4	address
0000	0	1	0	0	4
0001	0	1	0	1	5
0010	0	1	1	0	6
0011	0	1	1	1	7
0100	1	0	0	0	8
0101	1	0	0	1	9
0110	1	0	1	0	10
0111	1	0	1	1	11
1000	1	1	0	0	12
1001	1	1	0	1	13
1010	1	1	1	0	14
1011	1	1	1	1	15
1100	0	0	0	0	0
1101	0	0	0	1	1
1110	0	0	1	0	2
1111	0	0	1	1	3

to avoid the additional logic related to these delay buffers, a rotation on the ROM is performed, i.e., the 256 coefficients of each branch memory coefficient are shifted 250 addresses. Thus, the first coefficient is stored in address 249, so when the counter is equal to 249 the first coefficient is read.

- **Stage 6.** In this case, the memory only has 16 coefficients that are read sequentially every clock cycles, so the 4 LSBs of the counter are used as the read address of the ROM. The first rotation coefficient of the stage 6 ROM is needed after 270 cycles, so the 4 LSBs of the counter will be equal to 14. As in stage 4, the coefficients are shifted 14 positions to avoid the delay needed of 14 clock cycles.

- **Stage 8.** The reading of the ROMs of this stage works as that in stage 6, so the 4 LSBs of the counter are used. The first rotation coefficient to be read is needed after 280 cycles. Therefore, the coefficients are shifted 8 positions to avoid the delay needed of 8 clock cycles.

Chapter 4

Experimental results and comparison

In this chapter, the latency, resources and power consumption of the proposed architecture are analysed. Additionally, a comparison with other architectures is provided.

4.1 Experimental results

In the first place, it is necessary to point out the basic conditions of the architecture for which the results that are going to be presented have been obtained. The architecture has been designed for a 1024-point radix-2^4 with a parallelization of 4. The platform for which the architecture is developed is for any 7 series FPGA devices from Xilinx. Specifically, the selected board for which the results have been obtained is part xc7vx330tffv1157-1 from Virtex 7.

The system frequency has been set to 200 MHz, obtaining the slack (WNS) that is presented in figure 4.1, extracted from Vivado. This positive value indicates how much the system clock period can be decreased to achieve a higher frequency of operation. That is, the closer this value is to 0, the closer our system will be to the maximum possible frequency. Therefore, the maximum frequency at which the proposed architecture could work is approximately 206 MHz.

In terms of latency, as shown in section 3.3, the architecture gets results after 290 clock cycles. Therefore, the latency is 1.45 μs, taking into account the working frequency of the architecture (equation (4.1)).

$$\text{Latency (us)} = \frac{\text{Latency (cycles)}}{f_{CLK}\ \text{(MHz)}} = \frac{290}{200} = 1.45\ \mu s \tag{4.1}$$

Setup

Worst Negative Slack (WNS):	0,154 ns
Total Negative Slack (TNS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	9798

Figure 4.1: Slack obtained at 200 MHz.

Additionally, the throughput provides the number of samples that the architecture is able to output each second. This throughput is calculated as

$$\text{Th (MS/s)} = P \cdot f_{CLK}(\text{MHz}) = 4 \cdot 200 = 800 \text{ MS/s}. \tag{4.2}$$

Regarding the resources consumed by the FPGA, the architecture uses 1193 slices with a total of 3164 LUTs, where 675 correspond to LUTs as memory. These 675 LUTs are related to shift registers needed in MFF and MDF after the stage 2. For the first two stages, where the buffer lengths are 128 and 64 respectively, 2 BRAMs are used at each stage, one for the real part and one for the imaginary part. Then, 2073 flip-flops (FF) are used to implement all registers placed at the start of each stage and after addition/subtraction operations.

Resource	Utilization	Available	Utilization %
LUT	3164	204000	1.55
LUTRAM	675	70200	0.96
FF	2703	408000	0.66
BRAM	4	750	0.53
DSP	60	1120	5.36
IO	258	600	43.00

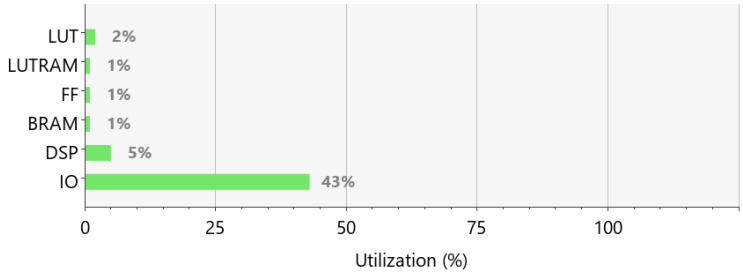


Figure 4.2: Resources utilization summary.

Additionally, 60 DSPs are used to implement the complex rotations. These 60 DSPs correspond to the 16 DSPs (4 per branch) at stages 2, 4 and 6, and the 12 DSPs used at stage 8, where the first branch does not calculate any rotation.

In figure 4.2 the resource's utilization for the FPGA selected is shown. As we can

observe, the implemented architecture consumes a very low percentage of available resources in the Virtex 7 FPGA. The I/O resources do not have to be taken into account, as in a real system the FFT is expected to be an internal component of the system, not connected to the I/O of the board.

Finally, the analysis of the power consumed by our design is of great interest in order to better understand the performance of the system developed. The total power consumed by our architecture, as can be seen in figure 4.3, is the sum of the dynamic power and the static power. On the one hand, the static power depends exclusively on the FPGA technology used, since it refers to the power consumed by the device when it is on, regardless of the implemented design. On the other hand, the power consumed by the developed system is separated by the different resources used by the architecture that are: clocks, signals, logic, BRAM and DSPs. Therefore, our architecture has a power consumption of 0.494 W. This low power consumption is considered one of the strengths of the proposed architecture.

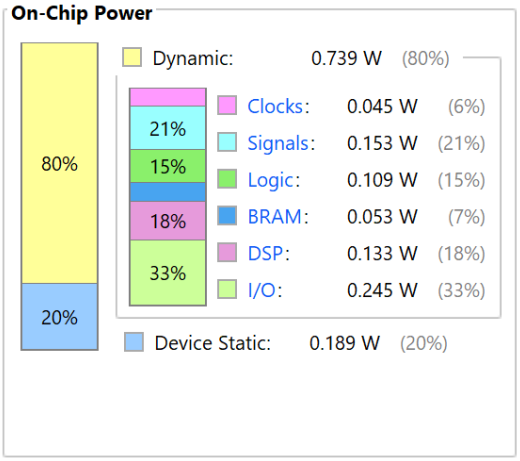


Figure 4.3: Power consumption.

4.2 Comparison

In order to estimate the positive impact of our architecture in the field of FFT hardware architectures, table 4.1 compares 4-parallel 1024-point FFT architectures. All designs have $N = 1024$, $P = 4$, and $WL = 16$, which makes them comparable. In addition, all the architectures are implemented in Virtex FPGAs of Xilinx, specifically the series 6 and 7. The cells with a dash in the table indicate that the paper does not include this specific value.

The proposed architecture presents a solution with a low resource consumption in terms of slices, LUTs and FFs, compared with other architectures. The architecture designed in this Master's Thesis reduces by 15% or more the number of slices and by 60% or more the number of BRAMs with respect to the first four architectures

Table 4.1: Comparison of 4-parallel 1024-point FFTs on FPGAs.

FFT	Garrido (2014) [51, 52]	Wang (2016) [26]	Spiral (2017) [53, 54]	Garrido (2018) [55]	Garrido (2021) [56]	Proposed
FPGA	V6	V6	V6	V6	V7	V7
N	1024	1024	1024	1024	1024	1024
P	4	4	4	4	4	4
Radix	2^2	2^2	2	2^5	2^5	2^4
WL	16	16	16	16	16	16
Architecture	MDC	MDF	-	MDC	CM	MDF-MFF
Slices	1341	-	1443	1420	2631	1146
LUTs	-	3671	-	-	8478	3159
FFs	-	4359	-	-	11832	2703
BRAM	12	10	44	12	0	4
DSP slices	48	45	64	16	12	60
Latency (ns)	-	-	-	-	376	1450
Latency (clks)	-	-	-	-	256	290
Clk (MHz)	227	305	289	253	680	200
Th (MS/s)	910	1220	1157	1012	2720	800
P(W)	-	-	-	-	1.68	0.49
P_{NORM}(W)	-	-	-	-	0.49	0.49

presented in the table. Unfortunately, the power consumption of these architectures is not specified in their respective papers to make a comparison with our approach.

Comparing our architecture with the last approach, published in 2021, we can observe the resource's reduction in terms of logic that occurs at the cost of increasing the number of DSPs and BRAM. Our approach, reduces the number of slices LUTs by 63% and the number of flip-flops by 77%. However, there is an increase in the number of DSPs and BRAMs needed. Due to this, each approach can be selected as a better option depending on the specifications of the system where the architecture is used and the FPGA resources.

In terms of frequency and throughput, the proposed design achieves lower speed than in previous approaches. However, the obtained results are considered acceptable and may be improved in future works.

Finally, the low power consumption of our approach can only be compared with [56], because most of the other previous approaches do not provide this parameter. As we can observe in table 4.1, the architecture designed in this work consumes 71% less power than [56]. However, the power consumption if directly proportional to the clock frequency of the system. Therefore, to compare two designs that work at

different frequencies, a normalization must be made to have both at the same clock frequency, by applying the equation (4.3). With this, both designs are similar in terms of power.

$$\mathrm{P_{NORM}(W)} = \frac{\mathrm{P(W)}}{\left(\frac{f_{CLK}}{200\ \mathrm{MHz}}\right)} \tag{4.3}$$

The low power consumption achieved represents a great advance for the implementation of future 6G networks. After all, 6G technology will need to deliver much more data at faster rates than today's networks, while still fulfilling very stringent energy-efficiency goals. To obtain higher data rates, a parallelization is needed in all parts of the 6G networks that translates to larger area and higher power consumption. For this reason, the low FPGA resources and power consumption obtained in this proposed Master's Thesis is also remarkable for the future implementation of 6G technology.

Chapter 5

Conclusions and future works

5.1 Conclusions

In this Master's Thesis, a 1024 point FFT 4-parallel architecture has been designed and implemented on a Xilinx Virtex 7 FPGA. First, the design has been separated into several modules based on MDF and MFF architectures. This latter architecture is the parallel version of the SFF and is the first work in the state of the art where it is implemented, which is an added value to our project. The developed modules have been selected pursuing the least FPGA resources and power consumption as possible. To achieve this objective, an exhaustive study of the FPGA Series 7 has been carried out, which has allows us to design and map the architecture in the FPGA optimally.

Depending on the buffer length needed for the butterfly operation and the type of rotation that is performed, 5 modules have been developed: MFF for trivial rotations stages and $L \leq 32$, MDF for trivial rotations stages and $L > 32$, MFF for complex rotations stages and $L = 1$, MDF for complex rotations stages and $2 \leq L \geq 32$ and MDF for complex rotations stages and $L > 32$. For buffer sizes less than or equal to 32, the buffers have been implemented with SRLs that are mapped into a few slice LUTs. In the case of buffer sizes higher than 32, to reduce the number of consumed LUTs that implementing the design with SRLs would entail, the buffers have been developed by sharing a BRAM between the four parallel branches of the architecture.

Additionally, the complex rotations have been mapped into DSP blocks. In the case of the complex MFF for $L = 1$, a special approach was designed, allowing us to implement the whole stage into the 4 DSPs needed for the complex rotation. This has been possible by using the DSP internal pipeline input registers to map the buffers needed for the butterfly addition and subtraction.

Then, the radix-2^4 for the FFT architecture was selected by pursuing the least

rotation complexity. Subsequently, the specific radix-2^4 algorithm that determines the stages where the complex rotations are placed was chosen. With this, just 8 ROMs are required to store the coefficients of the architecture complex rotators.

To finish the architecture implementation, the modules and rotator coefficient memories were integrated together. To allow this, the control signals required at each stage were implemented by using one counter and some delays. Additionally, the coefficient memories were rotated to synchronize the read address at each point of the architecture. With this technique, all necessary delay registers were avoided and the FPGA resources consumption was greatly reduced.

Finally, the experimental results have been compared with other architectures on the state-of-the-art. In this comparison, we could observe that the resource consumption of our design was low, and the architecture power consumption was considerably low. In terms of clock frequency, the design result was acceptable, taking into account that some improvements could be implemented in future works, as is explained in the next section.

5.2 Future works

For this Master's Thesis, the complex rotators were implemented by using DSP blocks in all the cases. In future works, it may be interesting to explore the possibility of implementing the rotators with the CORDIC algorithm or the CCSSI approach. This way, we may compare the results and an upgraded version of this work may be achieved. Additionally, with these upgrades and some other improvements that can be performed in the memory reading (RAM and ROM), the system frequency may be increased. With this, it would be possible to make a scientific publication, where the expected throughput should be higher.

Additionally, this Master's Thesis may be adapted to other FPGA families. For example, in UltraScale Xilinx FPGAs, the DSP block (DSP48E2) has some improvements over the DSP48E1 of Series 7 FPGA as the addition of a fourth operand in the ALU operation, which can be interesting to explore.

Finally, the proposed architecture is presented as a good starting point for the design of advanced FFTs for 6G. For this reason, by continuing the same research thread of the present work, focusing on the low power consumption, low resources consumption, low latency and high throughput the capability can be improved, which may give rise to commercial solutions for 6G.

Appendix A

Budget

LABOUR COSTS		Hours	Cost/hour	TOTAL
Junior Engineer in Electronics		500	16 €	**8.000 €**
MATERIAL COSTS	Cost (€)	Use (months)	Amort. (years)	TOTAL
Computer and sofware	1.500,00	10	5	**250,00 €**
Xilinx Virtex-7 FPGA VC707	5244,00	10	5	**874,00 €**
SUBTOTAL DIRECT COSTS				**9.124,00 €**
INDIRECT COSTS		15%	of DC	**1.368,60 €**
INDUSTRIAL PROFIT		6%	of DC+IC	**629,56 €**
FUNGIBLE MATERIAL				
Office material				**70,00 €**
SUBTOTAL				**11.192,16 €**
IVA		21%		**2.350,35 €**
TOTAL BUDGET				**13.542,51 €**

Table A.1: Budget

Appendix B

Ethical, social, economic and environmental aspects

This annex discusses ethical, economic, social and environmental aspects related to this Master's Thesis.

B.1 Ethical and social impacts

Due to the future 6G applications of this Master's Thesis, the impact in society is evident. The 6G communications will be used in many fields of the daily life as it will focus on the interconnection of everything, from cars to buildings. For example, in ultra-reliable low-latency communications or connected autonomous vehicles (CAV) would reduce the number of accidents due to the reduction in latency that make cars react faster.

In the ethical context, new applications and automations due to the wireless technology development would take away some humans jobs, making society increasingly dependent on machines. However, most applications improve solutions or present solution to problems that are not actually contemplated. For example, in the health field, 6G improvements will allow to hospitals a major easiness and faster detection and reporting of diseases, in addition to applications such as telesurgery [62].

B.2 Economic impact

Achieving 6G standards and reducing the power consumption in telecommunications will open a world of new technologies in several fields. For example, the

CAV vehicles are revolutionizing the motor sector even if there are restrictions that 6G technology will avoid. Another example resides in communications and radio astronomy, where the improvements achieved though 6G will allow us to increase the number of scientific discoveries and the competence in the space race [63].

B.3 Environmental impact

Increasingly, the environmental problems due to human technologies is affecting our planet. The carbon footprint of electronic devices and recent technologies and digital processes, such as blockchain cryptocurrency or training large AI models, is getting bigger and bigger. Because of this, the importance of reducing the power consumption and number of devices needed in a specific task contributes to slightly curbing this negative impact in the environment.

Not only is there a strong societal expectation to reduce technology's carbon footprint, but there are also commercial incentives. This means that the required energy for transmitting a bit must be significantly reduced. In this Master's Thesis, these objectives are achieved by proposing a low power consumption and reduced area FFT architecture, which will play an important role within 6G networks implementation.

Bibliography

[1] Xilinx. *7 Series DSP48E1 Slice: User Guide*. Xilinx, March 2018.

[2] M. G. Kibria, K. Nguyen, G. P. Villardi, O. Zhao, K. Ishizu, and F. Kojima. Big data analytics, machine learning, and artificial intelligence in next-generation wireless networks. *IEEE Access*, 6:32328–32338, May 2018.

[3] C. Zhang, Y.-L. Ueng, C. Studer, and A. Burg. Artificial intelligence for 5G and beyond 5G: Implementations, algorithms, and optimizations. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(2):149–163, June 2020.

[4] Z. Zhang, B. Cao, J. Guo, D. Weng, Y. Liu, and Y. Wang. Inverse virtual reality: Intelligence-driven mutually mirrored world. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 735–736, March 2018.

[5] Y. Liu, Q. Sun, Y. Tang, Y. Li, W. Jiang, and J. Wu. Virtual reality system for industrial training. In *2020 International Conference on Virtual Reality and Visualization (ICVRV)*, pages 338–339, November 2020.

[6] W.-C. Chang and W.-C. Chang. Real-time 3D rendering based on multiple cameras and point cloud. In *2014 7th International Conference on Ubi-Media Computing and Workshops*, pages 121–126, July 2014.

[7] L. Guo. 3D image optimization model of new media immersive display platform of digital traditional culture. In *2022 6th International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 1570–1574, April 2022.

[8] H. Zhang, N. Shlezinger, F. Guidi, D. Dardari, M. F. Imani, and Y. C. Eldar. Near-field wireless power transfer for 6G internet of everything mobile networks: Opportunities and challenges. *IEEE Communications Magazine*, 60(3):12–18, March 2022.

[9] V. Avula, R. Nanditha, S. Dhuli, and P. Ranjan. The internet of everything: A survey. In *2021 13th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 72–79, September 2021.

[10] C.-X. Wang, F. Haider, X. Gao, X.-H. You, Y. Yang, D. Yuan, H. M. Aggoune, H. Haas, S. Fletcher, and E. Hepsaydir. Cellular architecture and key

technologies for 5G wireless communication networks. *IEEE Communications Magazine*, 52(2):122–130, February 2014.

[11] Q. Zhao and M. Gerla. Energy efficiency enhancement in 5G mobile wireless networks. In *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 1–3, June 2019.

[12] E. C. Strinati, S. Barbarossa, J. L. Gonzalez, D. Ktenas, N. Cassiau, L. Maret, and C. Dehos. 6G: The next frontier: From holographic messaging to artificial intelligence using subterahertz and visible light communication. *IEEE Vehicular Technology Magazine*, 14(3):42–50, August 2019.

[13] J. He, K. Yang, and H.-H. Chen. 6G cellular networks and connected autonomous vehicles. *IEEE Network*, pages 1–7, November 2020. Early Access.

[14] S. Lins, K. V. Cardoso, C. B. Both, L. Mendes, J. F. De Rezende, A. Silveira, N. Linder, and A. Klautau. Artificial intelligence for enhanced mobility and 5G connectivity in UAV-based critical missions. *IEEE Access*, 9:111792–111801, August 2021.

[15] E. C. Strinati, M. Peeters, C. R. Neve, M. D. Gomony, A. Cathelin, M. R. Boldi, M. Ingels, A. Banerjee, P. Chevalier, B. Kozicki, and D. Belot. The hardware foundation of 6G: The NEW-6G approach. In *2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 423–428, June 2022.

[16] Yunho J., Yonji T., Jaeseok K., Junhyun P., Dongkyu K., and Hyuncheol P. Efficient FFT algorithm for OFDM modulation. In *Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology. TENCON 2001 (Cat. No.01CH37239)*, volume 2, pages 676–678 vol.2, August 2001.

[17] D.-S Kim, S.-Y Lee, and D.-J Chung. A partially operated FFT/IFFT processor for low complexity OFDM modulation and demodulation of wibro in-car entertainment system. *IEEE Trans. on Consumer Electronics*, 54(2):431–436, May 2008.

[18] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, 19(90):297–301, April 1965.

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

[20] G. O'Leary. Nonrecursive digital filtering using cascade fast Fourier transformers. *IEEE Trans. Audio Electroacoust.*, 18(2):177–183, June 1970.

[21] H. L. Groginsky and G. A. Works. A pipeline fast Fourier transform. *IEEE Trans. Comput.*, C-19(11):1015–1019, October 1970.

[22] M. Garrido. A survey on pipelined FFT hardware architectures. *J. Signal Process. Syst., Survey Papers*, pages 1–20, July 2021.

[23] M. Santhi, S. A. Kumar, G. S. P. Kalish, K. Murali, S. Siddharth, and G. Lakshminarayanan. A modified radix-2^4 SDF pipelined OFDM module for FPGA based MB-OFDM UWB systems. In *Proc. Int. Conf. Comp. Comm. Networking*, pages 1–5, December 2008.

[24] C. Ingemarsson, P. Källström, F. Qureshi, and O. Gustafsson. Efficient FPGA mapping of pipeline SDF FFT cores. *IEEE Trans. on VLSI Systems*, 25(9):2486–2497, June 2017.

[25] E. H. Wold and A. M. Despain. Pipeline and parallel-pipeline FFT processors for VLSI implementations. *IEEE Trans. Comput.*, C-33(5):414–426, May 1984.

[26] J. Wang, C. Xiong, K. Zhang, and J. Wei. A mixed-decimation MDF architecture for radix- 2^k parallel FFT. *IEEE Trans. VLSI Syst.*, 24(1):67–78, January 2016.

[27] J.-F Tang, X.-J Li, G. Zhang, and Z.-S Lai. Design of high-throughput mixed-radix MDF FFT processor for ieee 802.11.3c. In *2012 IEEE 11th International Conference on Solid-State and Integrated Circuit Technology*, pages 1–3, October 2012.

[28] C. Ingemarsson and O. Gustafsson. SFF—The single-stream FPGA-optimized feedforward FFT hardware architecture. *J. Signal Process. Syst.*, 90:1583–1592, November 2018.

[29] M. Garrido. A new representation of FFT algorithms using triangular matrices. *IEEE Trans. Circuits Syst. I*, 63(10):1737–1745, October 2016.

[30] H. Lee and I. Park. Balanced binary-tree decomposition for area-efficient pipelined FFT processing. *IEEE Trans. Circuits Syst. I*, 54(4):889–900, April 2007.

[31] F. Qureshi and O. Gustafsson. Generation of all radix-2 fast Fourier transform algorithms using binary trees. In *Proc. European Conf. Circuit Theory Design*, pages 677–680, August 2011.

[32] M. Garrido, O. Gustafsson, and J. Grajal. Accurate rotations based on coefficient scaling. *IEEE Trans. Circuits Syst. II*, 58(10):662–666, October 2011.

[33] M. Garrido, F. Qureshi, and O. Gustafsson. Low-complexity multiplierless constant rotators based on combined coefficient selection and shift-and-add implementation (CCSSI). *IEEE Trans. Circuits Syst. I*, 61(7):2002–2012, July 2014.

[34] W. Han, A. T. Erdogan, T. A., and Mohd. Hasan. High-performance low-power FFT cores. *ETRI J.*, 30(3):451–460, June 2008.

[35] C.-H. Yang, T.-H. Yu, and D. Markovic. Power and area minimization of reconfigurable FFT processors: A 3GPP-LTE example. *IEEE J. Solid-State Circuits*, 47(3):757–768, March 2012.

[36] A. Wenzler and E. Luder. New structures for complex multipliers and their noise analysis. In *Proc. IEEE Int. Symp. Circuits Syst.*, volume 2, pages 1432–1435, April 1995.

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

[38] M. Garrido and J. Grajal. Efficient memoryless CORDIC for FFT computation. In *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, volume 2, pages 113–116, April 2007.

[39] P. K. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna. 50 years of CORDIC: Algorithms, architectures, and applications. *IEEE Trans. Circuits Syst. I*, 56(9):1893–1907, September 2009.

[40] Y.H. Hu and S. Naganathan. An angle recoding method for CORDIC algorithm implementation. *IEEE Trans. Comput.*, 42(1):99–102, January 1993.

[41] C.-H. Wu and A.-Y. Wu. Modified vector rotational CORDIC (MVR-CORDIC) algorithm and architecture. *IEEE Trans. Circuits Syst. II*, 48(6):548–561, June 2001.

[42] P. K. Meher and S. Y. Park. CORDIC designs for fixed angle of rotation. *IEEE Trans. VLSI Syst.*, 21(2):217–228, February 2013.

[43] C.-H. Wu, A.-Y. Wu, and C.-H. Lin. A high-performance/low-latency vector rotational CORDIC architecture based on extended elementary angle set and trellis-based searching schemes. *IEEE Trans. Circuits Syst. II*, 50(9):589–601, September 2003.

[44] C.-H. Lin and A.-Y. Wu. Mixed-scaling-rotation CORDIC (MSR-CORDIC) algorithm and architecture for high-performance vector rotational DSP applications. *IEEE Trans. Circuits Syst. I*, 52(11):2385–2396, November 2005.

[45] S. Y. Park and Y. J. Yu. Fixed-point analysis and parameter selections of MSR-CORDIC with applications to FFT designs. *IEEE Trans. Signal Process.*, 60(12):6245–6256, December 2012.

[46] M. Garrido, S.-J. Huang, S.-G. Chen, and O. Gustafsson. The serial commutator (SC) FFT. *IEEE Trans. Circuits Syst. II*, 63(10):974–978, October 2016.

[47] B. Gold and T. Bially. Parallelism in fast Fourier transform hardware. *IEEE Trans. Audio Electroacoust.*, 21(1):5–16, February 1973.

[48] M. Garrido and P. Paz. Optimum MDC FFT hardware architectures in terms of delays and multiplexers. *IEEE Trans. Circuits Syst. II*, 68(3):1003–1007, March 2021.

[49] M. G. Kim, S. K. Shin, and M. H. Sunwoo. New parallel MDC FFT processor with efficient scheduling scheme. In *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, pages 667–670, November 2014.

[50] S.-C. Hsu, S.-J. Huang, S.-G. Chen, S.-C. Lin, and M. Garrido. A 128-point multi-path SC FFT architecture. In *Proc. IEEE Int. Symp. Circuits Syst.*, pages 1–5, October 2020.

[51] M. Garrido, J. Grajal, M. A. Sánchez, and O. Gustafsson. Pipelined radix-2^k feedforward FFT architectures. *IEEE Trans. VLSI Syst.*, 21(1):23–32, January 2013.

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

[53] P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel. Formal datapath representation and manipulation for implementing DSP transforms. In *Proc. IEEE Design Automation Conf.*, pages 385–390, July 2008.

[54] Spiral DFT/FFT IP core generator, May 2017. http://spiral.net/hardware/dftgen.html.

[55] M. Garrido, S. J. Huang, and S. G. Chen. Feedforward FFT hardware architectures based on rotator allocation. *IEEE Trans. Circuits Syst. I*, 65(2):581–592, February 2018.

[56] M. Garrido and P. Malagón. The constant multiplier FFT. *IEEE Trans. Circuits Syst. I*, 68(1):322–335, January 2021.

[57] S. He and M. Torkelson. Design and implementation of a 1024-point pipeline FFT processor. In *Proc. IEEE Custom Integrated Circuits Conf.*, pages 131–134, May 1998.

[58] 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.

[59] M. Garrido, J. Grajal, and O. Gustafsson. Optimum circuits for bit reversal. *IEEE Trans. Circuits Syst. II*, 58(10):657–661, October 2011.

[60] Xilinx. *7 Series FPGAs Data Sheet: Overview*. Xilinx, September 2020.

[61] Xilinx. *7 Series FPGAs Configurable Logic Block: User Guide*. Xilinx, September 2016.

[62] S. Nayak and R. Patgiri. 6G communication technology: A vision on intelligent healthcare. In R. Patgiri, A. Biswas, and P. Roy, editors, *Health Informatics: A Computational Perspective in Healthcare*, pages 1–18. Springer Singapore, January 2021.

[63] P. P. Ray. A review on 6G for space-air-ground integrated network: Key enablers, open challenges, and future direction. *Journal of King Saud University - Computer and Information Sciences*, August 2021.