UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

MASTER'S THESIS

DESIGN AND IMPLEMENTATION OF NON-POWER-OF-TWO FFT ARCHITECTURES FOR 5G AND BEYOND

VÍCTOR MANUEL BAUTISTA LOZA

JULY 2022

MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

MASTER'S THESIS

Title:	DESIGN AND IMPLEMENTATION OF NON-POWER-OF-TWO FFT ARCHITECTURES FOR 5G AND BEYOND
Author:	VÍCTOR MANUEL BAUTISTA LOZA
Supervisor:	MARIO GARRIDO GÁLVEZ
Cosupervisor:	MARÍA LUISA LÓPEZ-VALLEJO
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 AND IMPLEMENTATION OF NON-POWER-OF-TWO FFT ARCHITECTURES FOR 5G AND BEYOND

AUTHOR: VÍCTOR MANUEL BAUTISTA LOZA

SUPERVISOR: MARIO GARRIDO GÁLVEZ

COSUPERVISOR: MARÍA LUISA LÓPEZ-VALLEJO

JULY 2022

A mi familia, pareja y amigos.

"Even if a samurai's head were to be suddenly cut off, he should still be able to perform one more action with certainty", Yamamoto Tsunetomo.

Abstract

In modern communication systems, hardware architectures are used to process the fast Fourier transform (FFT), which is an essential component of the transmitters and receivers of these systems. In the new generation of 5G communication, it is established that the size of the FFTs does not need to be power-of-two. This opens the door to the possibility of processing FFTs whose size is a product of powers of two, three and five. The design of these architectures is a field that has not been explored enough to achieve a high degree of optimization, unlike architectures whose size is power-of-two, which are currently quite efficient.

In this Master Thesis, efficient pipelined FFT architectures for non-power-of-two sizes have been designed. They process serial data at a rate of one sample per clock cycle. This is achieved by reducing the number of hardware resources while maximizing the use of them.

This Master Thesis starts from previous designs of butterflies for serial processing, which are essential components of the architectures. This Master Thesis is divided in the following steps: Analyze the FFT algorithm for sizes that are non-powers-of-two, determine the input and output data orders, design the permutation circuits of the architectures, implement the architectures by connecting the butterflies and the permutation circuits, and obtain experimental results on FPGAs.

The experimental results obtained by the proposed architectures improve the power-of-two SDF FFT architectures and represents a step forward in the development of efficient architectures for FFTs used in 5G and beyond.

Key Words

Digital circuits, FPGA, VHDL, ASIC, SDF, FFT, DFT, pipelined, butterfly, non-power-of-two, buffer.

Resumen

En los sistemas de comunicación modernos, se utilizan arquitecturas de hardware para procesar la transformada rápida de Fourier (FFT), que es un componente esencial de los transmisores y receptores de estos sistemas. En la nueva generación de comunicaciones 5G, se establece que el tamaño de las FFT no tiene por qué ser de potencia de dos. Esto abre la puerta a la posibilidad de procesar FFTs cuyo tamaño sea un producto de potencias de dos, tres y cinco. El diseño de estas arquitecturas es un campo que no se ha explorado lo suficiente como para lograr un alto grado de optimización, a diferencia de las arquitecturas cuyo tamaño es potencia de dos, que sí son bastante eficientes actualmente.

En este TFM se han diseñado arquitecturas FFT en pipeline eficientes para tamaños que no son potencia de dos. Procesan datos en serie a una velocidad de una muestra por ciclo de reloj. Esto se consigue reduciendo el número de recursos hardware y maximizando el uso de los mismos.

Este TFM parte de diseños previos de mariposas para el procesamiento en serie, que son componentes esenciales de las arquitecturas. Este TFM se divide en los siguientes pasos: analizar el algoritmo FFT para tamaños que no sean potencias de dos, determinar los órdenes de datos de entrada y salida, diseñar los circuitos de permutación de las arquitecturas, implementar las arquitecturas conectando las mariposas y los circuitos de permutación y obtener resultados experimentales en FPGAs.

Los resultados experimentales obtenidos por las arquitecturas propuestas mejoran las arquitecturas SDF de potencia de dos y representan un paso adelante en el desarrollo de arquitecturas eficientes para FFTs utilizadas en 5G y otras tecnologías futuras.

Palabras clave

Circuitos digitales, FPGA, VHDL, ASIC, SDF, FFT, DFT, pipeline, mariposa, no potencias de dos, buffer.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We are living in the era of 5G communications and start to hear about 6G. The advances of these technologies will be traduced in benefits for the entire society. In fact, the deployment of 5G networks is enabling cutting-edge applications such as autonomous vehicles [1] and telesurgery [2], which are expected to change our lives.

The Fourier transform [3] is a key component in 5G technologies. This mathematical operation transforms a signal from the time domain into the frequency domain. In 5G, it is used for calculating the modulation/demodulation of the symbols in the physical layer [4]. The discrete version of the Fourier transform is called discrete Fourier transform (DFT). In this case, the input signal is composed of complex-valued discrete samples and the output signal is composed of complex-valued discrete samples that represent the spectrum of the signal.

The fast Fourier transform (FFT) [5] is an algorithm that calculates the DFT more efficiently, as it reduces the number of operations from $\mathcal{O}(N^2)$ in the DFT to $\mathcal{O}(N \cdot \log N)$ in the FFT. The FFT algorithm was proposed by Cooley-Tukey [5] in 1965 and it has been used since then as a basic function for signal processing.

Multiple hardware FFT architectures for 5G, and even 6G, have been proposed so far [6, 7, 8]. These architectures consist of butterflies, rotators and memory [9]. Butterflies calculate additions and subtractions, rotators carry out rotations in the complex plane, and memory is used to store data and rotation coefficients. The hardware architectures that are used to process the FFT can be iterative [10, 11, 12] or pipelined [13]. Iterative hardware architectures [11] are based on memories, whereas pipelined architectures process a continuous flow of data. Pipelined FFTs can be divided into parallel architectures and serial architectures. The difference between both types is the throughput: Serial architectures process one sample per clock cycle, whereas parallel ones process more than one sample per clock cycle. The single-path delay feedback (SDF) [14, 15] is the most common hardware architecture for serial pipelined FFTs. So far, the utilization of the butterflies in SDF architectures is 50%.

In 5G, it is established that the size of the FFT does not have to be power-of-two, as it is described in the physical layer description for 5G [4]. The field of power-of-two sizes for FFTs has already reached a high grade of optimization [16]. However, FFT architectures for non-power-of-two sizes have been barely explored so far, due to their higher complexity. As a result, current designs for communication systems tend to accommodate the FFT size to a power of two.

The aim of this Master Thesis is to design serial pipelined FFT hardware architectures with non-power-of-two sizes and a high grade of optimization. The FFT sizes in this Master Thesis have been selected to match the requirements in the description of the physical layer of 5G. The starting point for the design of these architectures was the design of the butterflies, which was accomplished in my Bachelor Thesis [17]. The proposed butterflies achieved high efficiency and 100% utilization for processing serial data.

In this Master Thesis, the goal is to understand the FFT algorithms for non-power-of-two sizes, such as the prime factor algorithm (PFA) [18], and design efficient architectures for non-power-of-two sizes. These architectures use the butterflies in [17], as well as new permutation circuits in between them. As a result, the new architectures that are proposed are suitable for non-powers-of-two sizes and require less area than previous serial pipelined architectures.

This Master Thesis is organized as follows. In Chapter 2, a review of the state-of-the-art is provided. It starts with the role of FFTs in 5G. Then, it continues explaining the basic concepts of FFTs. This is followed by the butterflies that are going to be used in the architectures and their benefits compared to the SDF butterflies. Then the operating principles of optimum permutation circuits are summarized. Finally, it ends by analyzing SDF architectures. In Chapter 3, the proposed approach is presented. It starts with the study of the data order for non-power-of-two sizes, followed by another way to understand the PFA, and ends by studying algorithms for different FFT sizes. In Chapter 4, the hardware implementation of these designs and experimental results on FPGA are provided. The results are compared to SDF architectures with similar FFT sizes, proving that the proposed architectures reduce hardware resources. Finally, in Chapter 5, the main conclusions of this Master Thesis are summarized and future work is discussed.

Chapter 2

Background

2.1 FFTs in 5G and beyond

FFTs appears in 5G during modulation and demodulation transactions for orthogonal frequency division multiplexing (OFDM) systems [19]. The FFT module transforms a complex-valued symbol in the domain into complex-valued symbols that compose the constellations.

The physical layer description for 5G [4] establishes that the size of the FFT, N, can be a product of powers of 2, 3 and 5, i.e.,

$$N = 2^{\alpha_2} \cdot 3^{\alpha_3} \cdot 5^{\alpha_5}, \tag{2.1}$$

with α_2, α_3, $\alpha_5 \in \mathbb{Z}^+$. So far, hardware designers used to adapt their FFTs designs to power-of-to sizes, because these architectures have reached a higher grade of optimization. Designs of FFTs with non-power-of-two (NP2) sizes are far from reaching the same grade of optimization as power-of-two (P2) ones. In this Master Thesis, the goal is to tackle this issue and develop efficient FFT architectures for NP2.

Additionally, new communication generations have strict latency requirements. In the case of 5G, the system must communicate from point to point with a latency lower than 1 ms. If the system is expected to be implemented in 6G, the latency must be lower than 100 μs. As FFT hardware architectures are directly involved in the global system, latency is an important parameter to take into account.

2.2 FFT algorithm

As it was told in the introduction, the FFT is an efficient algorithm to calculate the DFT. An N-point DFT of a discrete signal $x[n]$ is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{nk}, \qquad k = 0, 1, \ldots, N-1, \tag{2.2}$$

where $W_N^{nk} = e^{-j\frac{2\pi}{N}nk}$ is called twiddle factor. Twiddle factors represent the phase that the signal must be rotated by in the complex plane. The complexity of this operation is $\mathcal{O}(N^2)$ where $\mathcal{O}(\cdot)$ represents the complexity of an operation. That means that its complexity increases quadratically with N. This implies that a lot of hardware resources must be dedicated just to compute the DFT. The FFT algorithm obtains the result of the DFT dedicating a lower amount of hardware resources. This is possible by reducing the number of operations that must been calculated. As a result, the new complexity that the FFT based on the Cooley-Tukey algorithms achieves is $\mathcal{O}(N \cdot \log N)$ [5].

The FFT algorithm can be represented by a flow graph, such as that in Figure 2.1 for an 8-point radix-2 FFT. FFTs are essentially composed of butterflies, rotators and memory along $\log_2 N$ stages [13]. A butterfly of radix-ρ calculates a DFT of ρ points. Radix-2 butterflies are commonly used, because they are as simple as calculating an addition and a subtraction of two complex samples. The rotations appear between the stages and are represented by the twiddle factors.

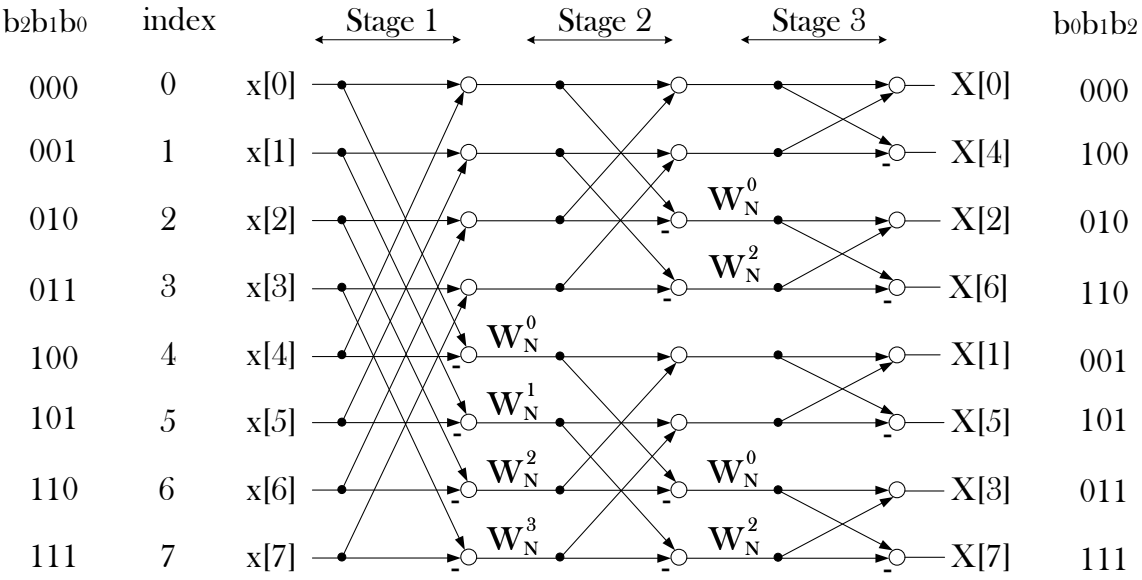

Figure 2.1: An 8-point radix-2 FFT flow graph.

In Figure 2.1, the first radix-2 stage divides the FFT into 2 FFTs of 4 points.

Using another radix-2 butterfly in the second stage means that in the last stage 4 FFTs of 2-points must be calculated.

The bits $b_2b_1b_0$ at the input of the flow graph correspond to the binary representation of the index, I. The FFT is composed of n stages. At ach stage $s = 1 \dots n$, butterflies operate on pairs of samples, where the index of each pair of samples only differs in bit b_{n-s}. As an example, the first stage operates on (0,4), (1,5), (2,6) and (3,7). By observing their binary representation (000,100), (001,101), (010,110), (011,111), it can be noticed that their value is the same except for the most significant bit $b_2 = b_{3-1}$. Likewise, for stage 2, pairs of data into a butterfly only differ in $b_{3-2} = b_1$, and at stage 3 they differ in $b_{3-3} = b_0$

As it can be seen in Figure 2.1, the outputs have a special order called bit-reversed order. This means that the output index is calculated by reversing the bits of the input index. The most significant bit of the input index becomes the least significant bit of the output one. For example, input with index (001) becomes (100). This holds for the rest of indexes with power-of-two sizes of FFTs. In NP2 FFTs, ternary bits appear, which makes more difficult the analysis of the bit reversal. Nevertheless, NP2 algorithms can take advantage of other properties that can simplify their flow graph.

Cooley-Tukey algorithm

Figure 2.2 represents the flow graph of a 6-point FFT based on the Cooley-Tukey algorithm. It consists of two stages of radix-3 and radix-2 butterflies, respectively. In this flow graph, the inputs of each stage are directly connected to the butterflies and the butterflies take consecutive inputs in the flow graph. Thus, this flow graph is an alternative to the flow graph in Figure 2.1 to represent FFT algorithms.

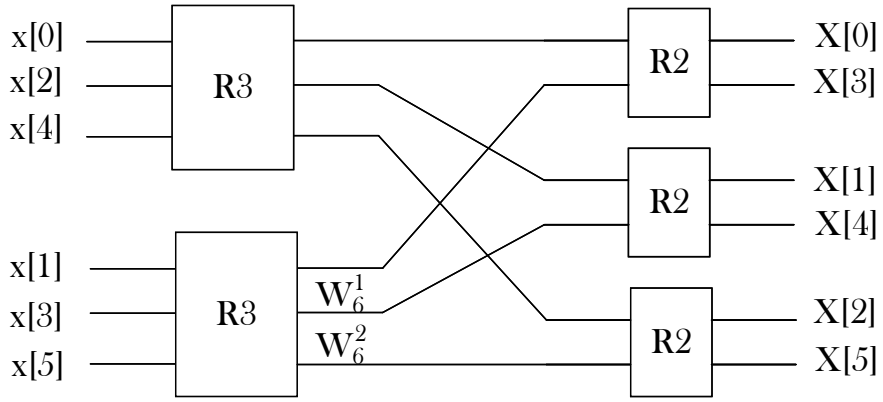

Figure 2.2: 6-point FFT using Cooley-Tukey algorithm.

The Cooley-Tukey algorithm is based on index mapping. This means that there is a mathematical method whereby the input and output indexes can be calculated,

as appears in Figure 2.2. The FFT size, N, can be expressed as a product of two integers $N = N_1 \cdot N_2$. The time-domain index, $n = 0 \dots N-1$, and the frequency-domain index, $k = 0 \dots N-1$, can also be expressed as a combination of two integers as

$$n = n_1 + N_1 \cdot n_2, \qquad 0 \leq n_1 \leq N_1 - 1, \quad 0 \leq n_2 \leq N_2 - 1, \tag{2.3}$$

$$k = N_2 \cdot k_1 + k_2, \qquad 0 \leq k_1 \leq N_1 - 1, \quad 0 \leq k_2 \leq N_2 - 1. \tag{2.4}$$

As the example is a 6-point FFT, then $N_1 = 3$ and $N_2 = 2$.

Prime factor algorithm

The prime factor algorithm (PFA) [18] is an alternative to the Cooley-Tukey algorithm for the specific case when the radices are coprime.

The PFA changes input and output index order with respct to the Cooley-Tukey algorithm and, as a result, the twiddle factors are removed. For this purpose, the size of the FFT must be expressed as $N = N_1 \cdot N_2$ where N_1 and N_2 are coprime. The input and output orders can be expressed with the next index mapping for any N:

$$n = < A \cdot n_1 + B \cdot n_2 >_N, \qquad 0 \leq n_1 \leq N_1 - 1, \quad 0 \leq n_2 \leq N_2 - 1, \tag{2.5}$$

$$k = < C \cdot k_1 + D \cdot k_2 >_N, \qquad 0 \leq k_1 \leq N_1 - 1, \quad 0 \leq k_2 \leq N_2 - 1, \tag{2.6}$$

where A, B, C and $D \in \mathbb{Z}$. One possible solution for the coefficients was proposed in [20]:

$$A = N_2, \tag{2.7}$$

$$B = N_1, \tag{2.8}$$

$$C = N_2 \cdot < N_2^{-1} >_{N_1}, \tag{2.9}$$

$$D = N_1 \cdot < N_1^{-1} >_{N_2} . \tag{2.10}$$

The operation $< N_1^{-1} >_{N_2} = \alpha$ can also be expressed as $< N_1 \cdot \alpha >_{N_2} = 1$, which is a property from the congruent theory [21]. Following the previous example, where

we have a 6-point FFT and $N = N_1 \cdot N_2$, the values of the terms whose product is N are $N_1 = 3$ and $N_2 = 2$. The reason is that N_1 corresponds to the size of the first butterfly (radix-3) and N_2 corresponds to the second one (radix-2). The results for this index mapping are shown in Table 2.1.

Table 2.1: Values for the PFA coefficients in a 6-point FFT

Coeff	Value
N_1	3
N_2	2
A	2
B	3
C	4
D	3

Once the coefficients A, B, C and D are calculated, it is possible to represent the index mapping into the flow graph. Figure 2.3 shows the PFA flow graph for a 6-point FFT. The advantage is clear if we think in a hardware implementation: Rotators do not have to be implemented. Consequently PFA reduces area and logic with respect to the Cooley-Tukey algorithm.

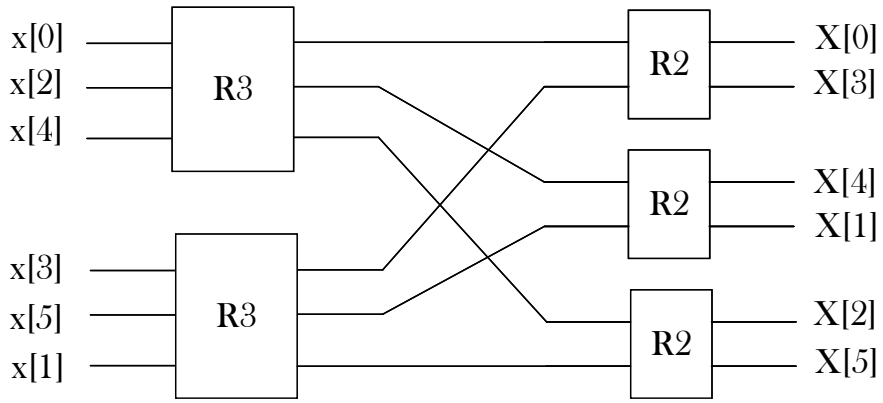

Figure 2.3: 6-point FFT using the prime factor algorithm.

2.3 Serial butterflies for NP2 FFTs

Every FFT architecture needs butterflies to work. In many architectures, radix-2 is used, among other reasons because its hardware implementation is simple. Larger butterflies have higher complexity, specially when the radix is not a power of two.

This Section presents efficient butterflies for radices 2, 3, 4 and 5 that process data in series. These butterflies have been proposed in my Bachelor Thesis [17] with the aim to use them to design the FFT architectures proposed in this Master Thesis. These butterflies are serial butterflies, which calculate a FFT of N samples

arriving in N consecutive clock cycles. The hardware implementation corresponds to a pipelined architecture that processes one sample each clock cycle. The intention of designing these butterflies is to increase their utilization up to 100%. Other FFT implementations such as SDF architectures use butterflies of size N that have N parallel inputs and N parallel outputs. Consequently, their number of components is roughly multiplied by N and their utilization decreases down to $1/N$.

Radix-2 butterfly

Radix-2 butterflies are the most simple butterflies that an FFT can use. The flow graph of a radix-2 butterfly is shown in Figure 2.4. The operations consist of one addition and one subtraction.

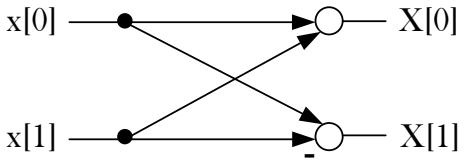

Figure 2.4: Flow graph of a radix-2 butterfly.

The hardware circuit that it is going to be used in this Master Thesis is shown in Figure 2.5.

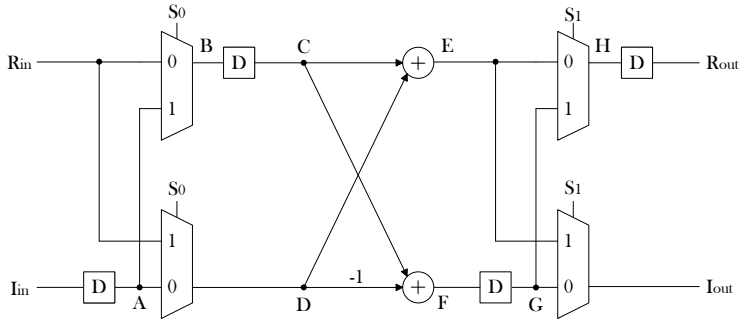

Figure 2.5: Hardware implementation of the radix-2 butterfly.

The butterfly has been designed with two branches. The reason is that both the input and the output are complex-valued samples. Thus, R_{in} and I_{in} are the real and imaginary parts of the input, respectively. Likewise, R_{out} and I_{out} are the real and imaginary parts of the output, respectively. Along the pipeline there are serial-parallel permutation circuits [22] that change data between both branches. The butterfly calculates the sum and subtraction of the real parts of two samples that arrive in consecutive cycles, and the sum and subtraction between the imaginary

part of the first sample and the second sample. The serial-parallel permutation circuit allows both real parts to be operated in the same clock cycle. Likewise, it allows the imaginary parts of both samples to be operated in the next clock cycle. The design of this butterfly is based on a serial commutator architecture [23].

Radix-3 butterfly

The radix-3 flow graph [24] is represented in Figure 2.6. In contrast to the radix-2 flow graph of Figure 2.4, in the radix-3 flow graph one real multiplication must be calculated.

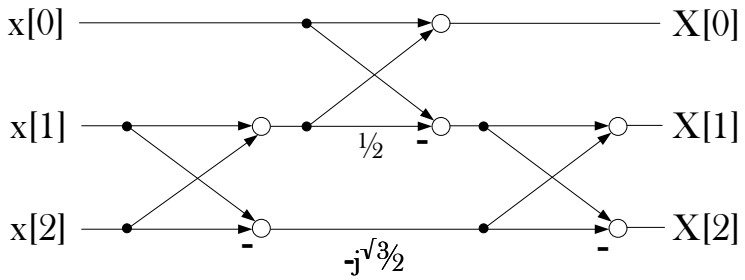

Figure 2.6: Flow graph of a radix-3 butterfly.

Figure 2.7 shows the hardware implementation of the radix-3 butterfly. It follows the datapath of the flow graph by using three stages with adders. The data that do not have to be operated in the adders are bypassed via multiplexer. As a result, the implementation only requires 6 real adders, 1 real multiplier and several multiplexers.

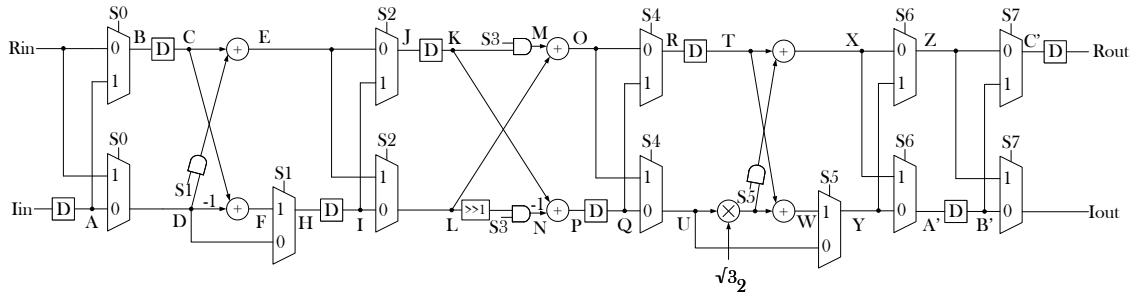

Figure 2.7: Hardware implementation of the radix-3 butterfly.

Radix-4 butterfly

The radix-4 flow graph is represented in Figure 2.8. In the literature this butterfly size is commonly used due to the fact that its rotation is trivial [25, 26]. Trivial rotations are rotations by 0^{o}, 90^{o}, 180^{o} and 270^{o} and can be calculated by changing

the real/imaginary parts of the input and/or changing their sign. Thus, no multiplier is required to calculate them.

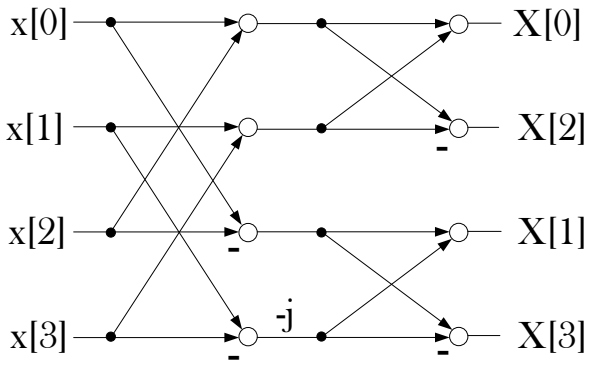

Figure 2.8: Flow graph of a radix-4 butterfly.

The hardware implementation of the radix-4 circuit is shown in Figure 2.9.

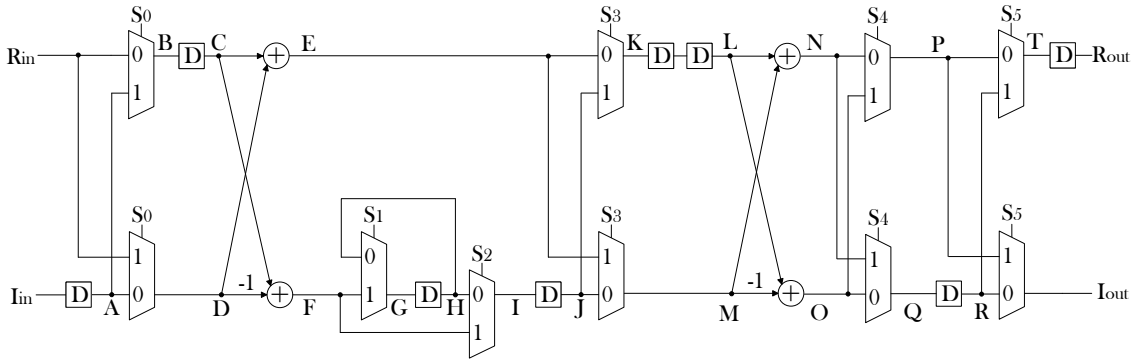

Figure 2.9: Hardware implementation of the radix-4 butterfly.

Radix-5 butterfly

The radix-5 flow graph [24] is represented in Figure 2.10. Several multiplications by constants must be carried out. The values of these constants are included in Table 2.2. The derivation of the flow graph is similar to the Winograd [27] and Rader [28] algorithms.

The hardware implementation of the radix-5 butterfly is shown in Figure 2.11.

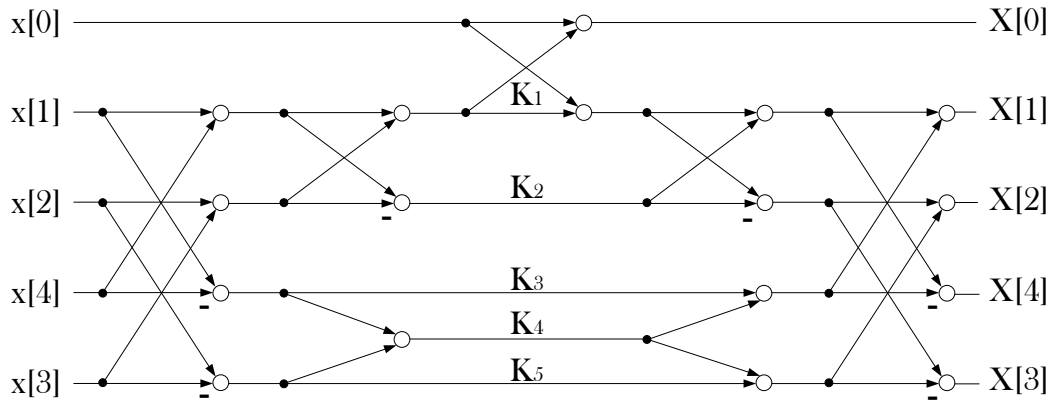

Figure 2.10: Flow graph of a radix-5 butterfly.

Table 2.2: Values for the coefficients of radix-5 flow graph.

Coeff	Equation	Value
K_1	$-\frac{1}{4}$	-0.25
K_2	$\frac{1}{2}\left(\cos\left(\frac{2\pi}{5}\right) - \cos\left(\frac{4\pi}{5}\right)\right)$	0.56
K_3	$j\left(\sin\left(\frac{4\pi}{5}\right) - \sin\left(\frac{2\pi}{5}\right)\right)$	$-j \cdot 0.36$
K_4	$-j\sin\left(\frac{4\pi}{5}\right)$	$-j \cdot 0.59$
K_5	$j\left(\sin\left(\frac{4\pi}{5}\right) + \sin\left(\frac{2\pi}{5}\right)\right)$	$j \cdot 1.54$

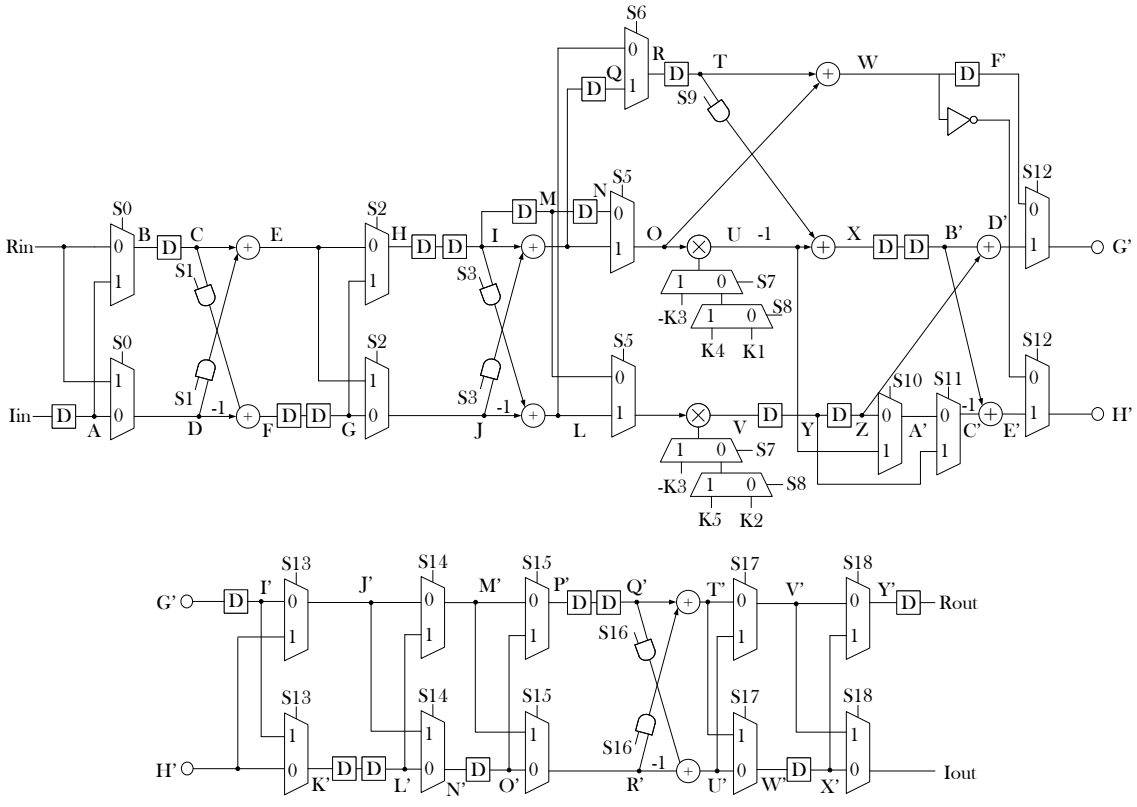

Figure 2.11: Hardware implementation of a radix-5 butterfly.

Theoretical limits and comparison between serial and parallel butterflies

Theoretically, there are some inferior limits that must be taken into account in order to calculate the minimum number of elements to implement in a pipelined circuit with only one input. In a butterfly that processes data in series there is a sample each clock cycle at the input and the butterfly must not take more than N cycles to do the calculations. Thus, an adder/multiplier can calculate at most N additions/multiplications. This results in the following equations:

$$\text{Minimum number of real adders} = \frac{\text{Total of real adds operations}}{N}, \tag{2.11}$$

$$\text{Minimum number of real multipliers} = \frac{\text{Total of real multiplications}}{N}. \tag{2.12}$$

For a radix-5 butterfly the minimum number of real adders is 7 and the proposed circuit has 10. The minimum number of multipliers is 2 real multipliers, which is reached in the proposed architecture.

As a result, serial butterflies can reduce area compared to a parallel implementation. Table 2.3 shows the comparison of hardware elements between both types of butterflies.

Table 2.3: Comparison between serial and parallel butterflies.

N	2		3		4		5	
Elements	Parallel	Serial	Parallel	Serial	Parallel	Serial	Parallel	Serial
Real multipliers	0	0	2	1	0	0	10	2
Real adders	4	2	8	6	16	4	34	10
Multiplexers	0	4	0	12	0	10	0	25
Registers	0	4	0	8	0	8	0	23
Logic gates	0	0	0	4	0	0	0	7
Utilization	50%	100%	33%	100%	25%	100%	20%	100%

2.4 SDF architectures

Single-path delay feedback architectures are the most used architectures for processing serial pipelined FFTs [13], specially for power-of-two sizes. Figure 2.12 shows an 8-point SDF FFT using radix-2 butterflies, which calculates the flow graph in Figure 2.1. In SDF FFTs, data arrive to each stage of the architecture from the top to the bottom of the flow graph in consecutive clock cycles. Therefore, the first input of the butterfly must wait for the second one 2^{n-s} clock cycles. For this purpose, a buffer is used in each stage.

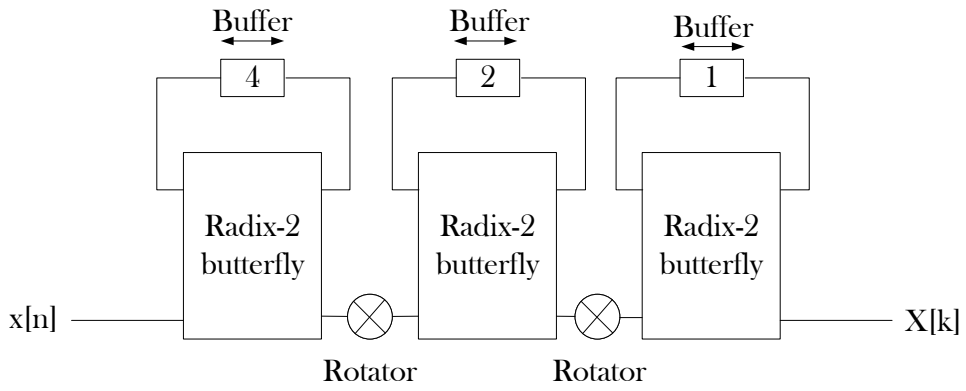

Figure 2.12: Architecture of an 8-point SDF FFT using radix-2 butterflies.

Figure 2.13 shows the internal structure of a stage of an SDF FFT. There are multiplexers that bypass the first samples and introduce them in the buffer. When data leave the buffer, they are operated together with the data arriving at the input of the stage. The outputs of the addition in the butterfly are sent to the output of the stage, whereas the outputs of the subtractions in the butterfly are transferred to the buffer as a previous step before leaving the circuit.

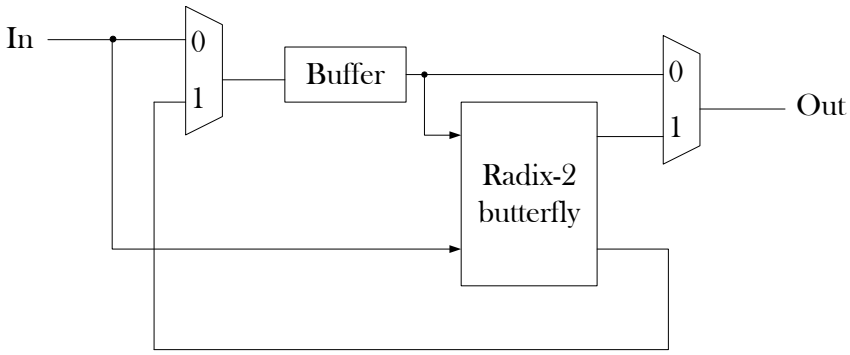

Figure 2.13: Architecture of a radix-2 SDF butterfly.

As a result, SDF butterflies are not working every clock cycle. Their utilization is $1/N$. This means that, in the best case, the utilization can reach the 50% of the time when using a radix-2 butterfly. This value is even lower when using non-power-of-two size butterflies. For a radix-3 and radix-5 butterflies their utilization decreases to 33% and 20% respectively.

2.5 Serial permutation circuits

Serial permutation circuits [22] are used to calculate permutations on data arriving in series, such as the bit reversal of serial data [29]. Likewise, they are the essence of serial commutator architectures [30, 31].

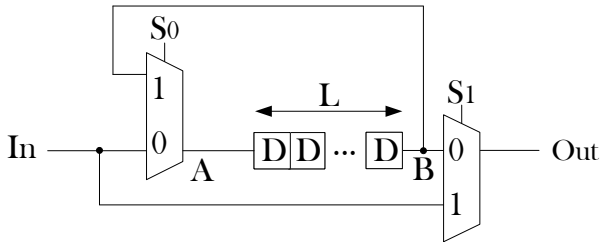

Figure 2.14: Serial permutation circuit.

A serial permutation circuit is shown in Figure 2.14. It consists in two multiplexer and a buffer of length L, where L is the temporal distance between the two samples that are permuted. A control circuit provides the signals for the multiplexers. When these control signals are set to 1, the input is directly transferred to the output with no cycles of latency. At the same moment, the sample at the output of the buffer returns to the input of the buffer. The rest of the time the second multiplexer extracts the samples that the buffer provides. As a result, the output provides the input sequence L clock cycles later with some samples interchanged depending on to the control signal values.

Chapter 3

Proposed approach

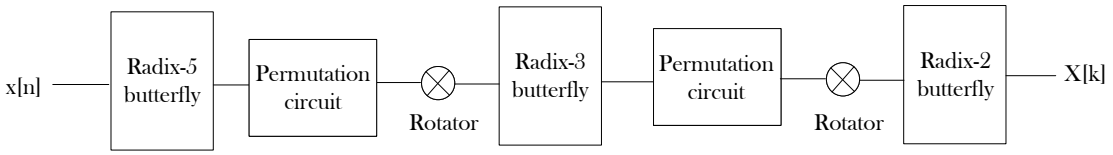

Figure 3.1: Hardware architecture of a serial pipelined 30-point FFT.

The aim of this Chapter is to analyze NP2 FFT algorithms and derive new approaches that are suitable for a hardware implementation. The final objective is to design a 30-point FFT as a proof of concept. As a starting point, using the serial butterflies already explained in Chapter 2, one possible implementation is shown in Figure 3.1. The input data indexes must be the ones that appear in the flow graph when it is read from top to bottom. In this figure, the index data is expected to be the one that is described in Cooley-Tukey algorithm. It is composed by the serial butterflies, permutation circuits and rotators. However, there are some modifications that can be done in order to optimize the algorithm.

Algorithms families such as prime factor reach efficient solutions using the index mapping methodology. The mathematical development of these solutions are far from being easy to understand. Along this Chapter, a new solution for PFA is developed, which can lead to new architectures with input and output orders different from those in [20].

In Section 3.1 the data order in NP2 architectures is studied. In Section 3.2 some useful properties for butterflies are proposed. With these new properties a new solution for the prime factor algorithm is proposed in Section 3.3. New algorithms for different NP2 sizes are designed in Section 3.4 by developing a new methodology with the advances described. The proposed algorithms have a high grade of optimization and can be an inspiration source for future NP2 designs.

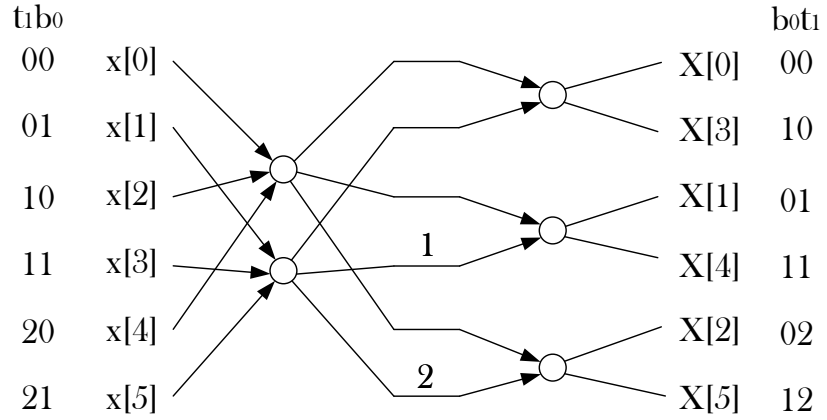

Figure 3.2: Flow graph of a 6-point FFT.

3.1 Model for NP2 algorithms

Figure 3.2 shows the decimation in frequency (DIF) flow graph of a 6-point FFT according to the Cooley-Tukey algorithm. This is a different way to represent flow graphs compared to Figure 2.2. In this flow graph the inputs of each stage are directly connected to spread butterflies, whereas in the previous flow graph all the inputs connected to a butterfly appear consecutively. The reason to use the flow graph in Figure 3.2 is that the binary representation of the index is easier to understand: Inputs are in natural order and outputs in bit-reversed order.

The order in which the radices are taken affects the FFT algorithm. In this example, as the the radix-3 butterflies are in first place, the inputs of the first pair of butterflies are taken every other sample. Then, the FFT is divided into 3 2-point FFTs. As there are 2 samples for each FFT and the radix is 2, samples are chosen every sample.

As N is a non-power-of-two number, the binary representation for each index can not be expressed with only binary bits. For instance, in Figure 3.2 the binary representation of the indexes uses binary bit (b_0) and a ternary bit (t_1). This way, we can also have quaternary bits for radix-4 and quinary bits for radix-5.

Following the previous example of the 6-point FFT, where the input indexes are read from the top to the bottom in natural order, input indexes are represented as

$$I_0 = 2 \cdot t_1 + b_0, \tag{3.1}$$

where t_1 is the ternary bit. Each bit is multiplied by a weight. Notice that there is a parallelism between power-of-two and NP2 flow graphs. Again, the samples that are inputs of the same butterfly have the same binary representation with the only exception of the most significant bit. The output indexes are not as trivial as the power-of-two ones that are used for radix-2 butterflies. For NP2 there are more

operations besides the reversal of the bits. Regarding the example of a 6-point FFT, output indexes are

$$I_1 = 3 \cdot b_0 + t_1. \tag{3.2}$$

The output binary representation consider the reversal of the bits and the multiplication by other weights which are different to the input ones.

In a general case, the index is defined as $I \equiv \chi_{n-1} ... \chi_0$, where χ corresponds to b, t, q in case of binary, ternary and quinary bits, respectively. Here, n is the number of stages and $s = 1, 2 \dots n$ is the specific stage. A generic input weight that characterizes n stages can be represented as

$$w_0 = \{w_{0,n-1} \ \dots \ w_{0,1} \ w_{0,0}\}. \tag{3.3}$$

Weights depend on the size of the butterflies (radix) used in the FFT and the order in which they appear. Each stage is composed of only one radix r_i and the size of the FFT is the product of all r_i. The set of radices along the FFT, R, can be expressed as

$$R = \{r_{n-1} \ \dots \ r_1 \ r_0\}, \tag{3.4}$$

so it is possible to describe each input weight exactly as the product of the next radices,

$$w_{0,s} = \prod_{i=1}^{n-s-1} r_i. \tag{3.5}$$

Once the input weights are calculated, the input index I_0 can be calculated as

$$I_0 = \sum_{i=0}^{n-1} \chi_i \cdot w_{0,i}. \tag{3.6}$$

The way in which the output weights are calculated considers that the radices are inverted. Thus, output weights can be represented as

$$w_{1,s} = \prod_{i=s+1}^{n-1} r_i. \tag{3.7}$$

In other words, an output weight is the product of all the radices before that stage. The output binary representation is the reverse version of the input one. Therefore, output indexes are calculated as

$$I_1 = \sum_{i=0}^{n-1} \chi_{n-1-i} \cdot w_{1,i}. \tag{3.8}$$

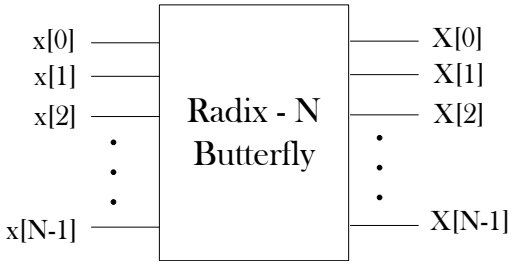

Figure 3.3: N-point FFT using a butterfly of radix-N.

3.2 Butterfly properties

The proposed solution to the prime factor algorithm is based on shifting the input and output data. The shifts are done in first place by considering a single butterfly. Later, this is generalized by combining more butterflies at different stages. In this Section, a formal mathematical demonstration of the index rotations of a single butterfly is proved. The intention of this Section is to model the behavior of the butterflies in order to derive general properties.

Let us consider a generic N-point FFT that uses only a butterfly, as is shown in Figure 3.3. This is a radix-N butterfly. The expressions of the outputs are calculated according to the DFT formula. For a generic radix-N butterfly the expressions are

$$X[0] = x[0] + x[1] + x[2] + \cdots + x[N-1], \tag{3.9}$$

$$X[1] = x[0] + x[1] \cdot W_N^1 + x[2] \cdot W_N^2 + \cdots + x[N-1] \cdot W_N^{N-1}, \tag{3.10}$$

$$\vdots$$

$$X[k] = x[0] + x[1] \cdot W_N^k + x[2] \cdot W_N^{2k} + \cdots + x[N-1] \cdot W_N^{(N-1)k}, \tag{3.11}$$

$$\vdots$$

$$X[N-1] = x[0] + x[1] \cdot W_N^{N-1} + x[2] \cdot W_N^{2\cdot(N-1)} + \cdots + x[N-1] \cdot W_N^{(N-1)^2}. \tag{3.12}$$

Previous equations can be rewritten by extracting a common factor, i.e.,

$$X[0] = (x[0] + x[1] + x[2] + \cdots + x[N-1]), \tag{3.13}$$

$$X[1] = (x[0] \cdot W_N^{-1} + x[1] + x[2] \cdot W_N^1 + \cdots + x[N-1] \cdot W_N^{N-2}) \cdot W_N^1, \tag{3.14}$$

$$X[2] = (x[0] \cdot W_N^{-2} + x[1] + x[2] \cdot W_N^2 + \cdots + x[N-1] \cdot W_N^{2(N-2)}) \cdot W_N^2, \tag{3.15}$$

$$\vdots$$

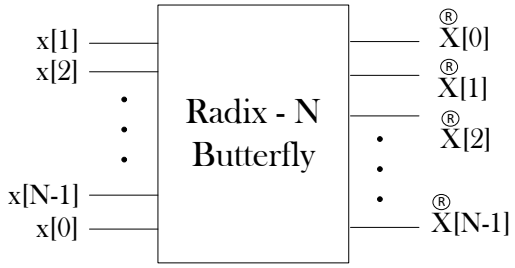

Figure 3.4: N-point FFT using a butterfly of radix-N with 1-shifted inputs.

$$X[N-1] = (x[0]\cdot W_N^{-(N-1)} + x[1] + x[2]\cdot W_N^{(N-1)} + \cdots + x[N-1]\cdot W_N^{(N-1)(N-2)})\cdot W_N^{N-1}. \tag{3.16}$$

Notice that the common factor is not constant for all the equations. It has been chosen such as $x[1]$ is multiplied by 1 inside the parenthesis. In equations (3.13-3.16), the common factors were the twiddle factors that were multiplying $x[1]$. Thus, a generic output can be described as

$$X[k] = (x[1] + x[2]\cdot W_N^k + \cdots + x[N-1]\cdot W_N^{k\cdot(N-2)} + x[0]\cdot W_N^{k\cdot(N-1)})\cdot W_N^k. \tag{3.17}$$

Notice that the term inside the parenthesis has the same form as equations (3.10) to (3.12). with the exception that the decimal indexes are shifted one position. In (3.17), $x[1]$ appears instead of $x[0]$, $x[2]$ instead of $x[1]$, etc.

Let us assume that $\overset{①}{X}$ represents the term inside the parenthesis in (3.17). It actually conforms another DFT solution with 1-shifted inputs. Therefore, it is fulfilled that

$$X[k] = \overset{①}{X}[k]\cdot W_N^k. \tag{3.18}$$

Figure 3.4 shows a generic butterfly with 1-shifted inputs. The index 0 is circular-shifted, so it becomes the last input of the butterfly. If more than one position is shifted, the previous operations can be done multiple times. Extending this property to an M-position shift, equation (3.18) leads to

$$\overset{Ⓜ}{X}[k] = X[k]\cdot W_N^{-Mk}. \tag{3.19}$$

For all of this demonstration, M is considered as a positive integer when the circular shift is done from bottom to top in the butterfly. If the shift is done from top to bottom there are two ways to interpret the value of M that lead to the same solution. The first way is to interpret that it is a negative integer. The second way is to interpret that it represent a positive shift of $N - M$ positions. As a result, for an input shifting operation, the outputs are directly multiplied by an exponential factor whose value depends of the index k and the number of positions shifted M. As a reminder, $W_N^{-Mk} = e^{j\frac{2\pi}{N}Mk}$.

Now, the effect that appears at the input when the outputs are shifted will be analyzed. Let us consider the starting point of Figure 3.3. The outputs are calculated as

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{nk}, \quad k = 0, 1, \ldots, N-1. \tag{3.20}$$

Let us also consider the shift frequency property [3], i.e.,

$$X[< k - k_0 >_N] = x[n] \cdot W_N^{nk_0}. \tag{3.21}$$

The parameter k_0 is replaced by $-M$ because in the proposed model M is positive when the shift is done from bottom to top. By substituting this equation into (3.20) and replacing k_0 by $-M$ leads to

$$X[k + M] = \sum_{n=0}^{N-1} (x[n] \cdot W_N^{nM}) \cdot W_N^{nk}, k = 0, 1, \ldots, N-1. \tag{3.22}$$

The term inside the parenthesis can be renamed as $\overset{\text{(M)}}{x}[n]$, leading to

$$\overset{\text{(M)}}{x}[n] = x[n] \cdot W_N^{nM}. \tag{3.23}$$

As a result, $\overset{\text{(M)}}{x}[n]$ represents the rotation that appears at the input when the outputs are shifted. Figure 3.5 shows the butterfly when the outputs are shifted one position, i.e. $M = 1$.

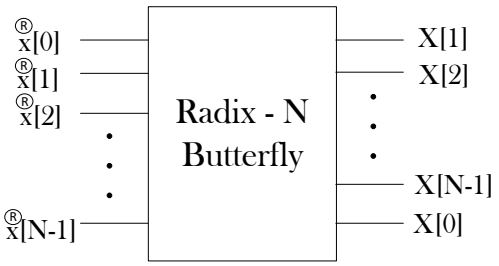

Figure 3.5: N-point FFT using a butterfly of radix-N with outputs shifted one position.

3.3 New solution for PFA

In the literature there are solutions that show how to calculate the PFA FFT [20]. However, these solutions are far from being intuitive or easy to understand. In this Section, a new method to understand the prime factor algorithm is used and a new solution for it is proposed. In order to derive the proposed solution, the butterfly properties provided in Section 3.2 are used.

The proposed approach starts from the Cooley-Tukey algorithm. Then, the inputs and outputs of the butterflies are shifted in such a way that the twiddle factors are avoided. The main difference with the original prime factor algorithms is that in the proposed approach, the input and output order is not based on an index mapping as in the common PFA.

Figure 3.6 represents a flow graph that is divided at some point by a vertical line. Let us assume that this flow graph corresponds to an N-point FFT using the Cooley-Tukey algorithm, where N is a product of two integers $N = N_1 \cdot N_2$. The Cooley-Tukey algorithm forces to calculate a multiplication by a twiddle factor $W_{N_1 N_2}^{nk}$ [9]. The objective is to find values for the number of positions that the input and output must be shifted, such that the twiddle factors are compensated with other rotations.

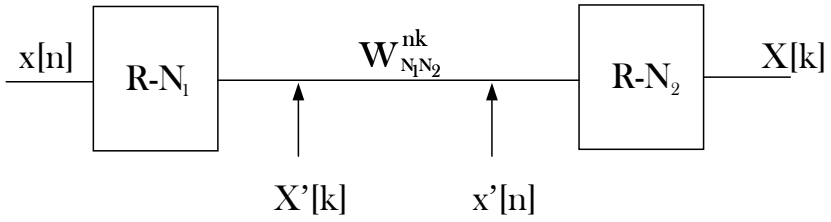

Figure 3.6: Vertical cut of a branch in a generic flow graph of the Cooley-Tukey algorithm.

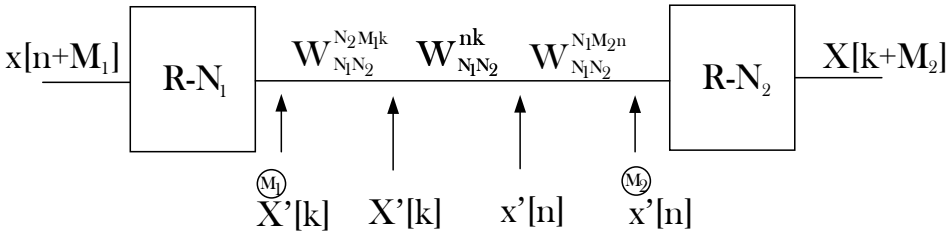

Figure 3.7: Effects of input and output movements on a branch in a flow graph.

Figure 3.7 shows all the external factors that affect the branch once the input and output data are shifted. The parameters M_1 and M_2 represent the number of positions that the input and output data of the first and the second butterfly are shifted, respectively. Both parameters have a positive value when the shift is done

upwards. As the first butterfly was input-shifted, it expects at the output $\overset{\text{M1}}{X'}[k]$. However, this model is based on the Cooley-Tukey architecture. This means that for making the algorithm work, the line expects $X'[k]$ represented as

$$X'[k] = \overset{\text{M1}}{X'}[k] \cdot W_N^{N_2 M_1 k}. \tag{3.24}$$

Notice that the factor N_2 appears in the exponent of $W_N^{N_2 M_1 k}$. The multiplication that appears on the output should be $W_{N_1}^{M_1 k}$. In order to normalize the base of the multiplication N_1 to the base of the twiddle factor N, the factor N_2 multiplies both the base and the exponent. Regarding the second butterfly, in order to achieve a shifted output, the multiplication by $W_N^{N_1 M_2 n}$ appears in its input. As it was explained, it is normalized to the base N of the twiddle factor by multiplying the base and the exponent by N_1, leading to

$$\overset{\text{M2}}{X'}[n] = X'[n] \cdot W_N^{N_1 M_2 n}. \tag{3.25}$$

Three exponential factors with the same base are multiplying the value in the branch that connects both butterflies. The objective is to get M_1 and M_2 such that

$$W_N^{N_2 M_1 k} \cdot W_N^{N_1 M_2 n} \cdot W_N^{nk} = 1, \tag{3.26}$$

which will remove the twiddle factor from this branch. As the equation consists of exponential factors of the same base, the equation can be expressed as a sum of their exponents, leading to

$$N_2 M_1 k + N_1 M_2 n + nk = 0. \tag{3.27}$$

If the term nk is moved to the other side of the equation, it can be observed that the movements of the input and output of the butterflies try to avoid the twiddle factor by cancelling it with its inverse form: i.e.,

$$N_2 M_1 k + N_1 M_2 n = -nk. \tag{3.28}$$

The election of M_1 and M_2 must hold for every index n and k in the equation (3.28). Consequently, those values are not constant for every butterfly of each stage. One conclusion that can be drawn from the previous equation is that N_1 and N_2 must be coprime by each other in order to fulfill equation (3.28). A simpler expression can be derived by solving previous equation for $nk = 1$, which represents the first twiddle factor, and then generalize for the rest of butterflies. Thus, if

$$N_2 M_1 + N_1 M_2 = -1, \tag{3.29}$$

solving for M_1 leads to

$$M_1 = \frac{-1 - N_1 M_2}{N_2}. \tag{3.30}$$

Regarding equation (3.28), the exponent of the twiddle factor increases with both n and k. The butterflies of the same stage increment the value of the exponents of their twiddle factors in a factor of k if the flow graph is read from top to bottom. This means that for the the first butterfly, $k = 0$, for the second one $k = 1$, etc. This value of k holds constant for the same butterfly, where the twiddle factors depend on the branch, n. Consequently, M_1 and M_2 must also increment gradually according to the specific butterfly along the stage:

$$M_{1,\alpha} = M_1 \cdot \alpha, \qquad \alpha = 0, 1, ..., N_2 - 1. \tag{3.31}$$

$$M_{2,\beta} = M_2 \cdot \beta, \qquad \beta = 0, 1, ..., N_1 - 1. \tag{3.32}$$

3.4 NP2 FFT optimum designs using proposed PFA

In this Section, FFT algorithms for 6, 15 and 30 points are proposed. The algorithms use the proposed solution for the prime factor algorithm. The starting point is the Cooley-Tukey algorithm and, then, the values for M_1 and M_2 are calculated.

6-point FFT

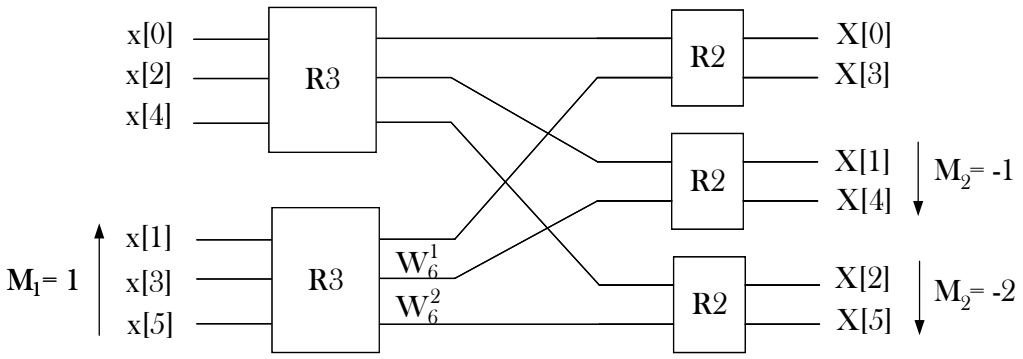

Figure 3.8: 6-point FFT using Cooley-Tukey algorithm.

Figure 3.8 shows the Cooley-Tukey algorithm for a 6-point FFT. Let us consider the connection where the twiddle factor W_6^1 appears. In this branch, $n = k = 1$ so that equation (3.28) can be rewritten as

$$2M_1 + 3M_2 = -1. \tag{3.33}$$

By solving the previous equation for M_1, we obtain

$$M_1 = \frac{-1 - 3M_2}{2}. \tag{3.34}$$

The simpler method to achieve the values for M_1 and M_2 is giving integers values for M_2 until the value of M_1 is integer too. Choosing $M_2 = -1$ makes $M_1 = 1$. Consequently, each butterfly of the first stage must be input-shifted as

$$M_{1,\alpha} = 1 \cdot \alpha, \qquad \alpha = 0, 1, ..., N_2 - 1. \tag{3.35}$$

Each butterfly of the second stage must be output-shifted as

$$M_{2,\beta} = -1 \cdot \beta, \qquad \beta = 0, 1, ..., N_1 - 1. \tag{3.36}$$

As a result, Figure 3.9 shows the optimized algorithm including the effect of the input and output shifting. Notice that the product of the twiddle factors in the same branch is always 1, which removes the twiddle factors. Figure 3.10 shows the optimized algorithm once the twiddle factors have been removed. The same solution is achieved by the PFA using the index mapping in [20].

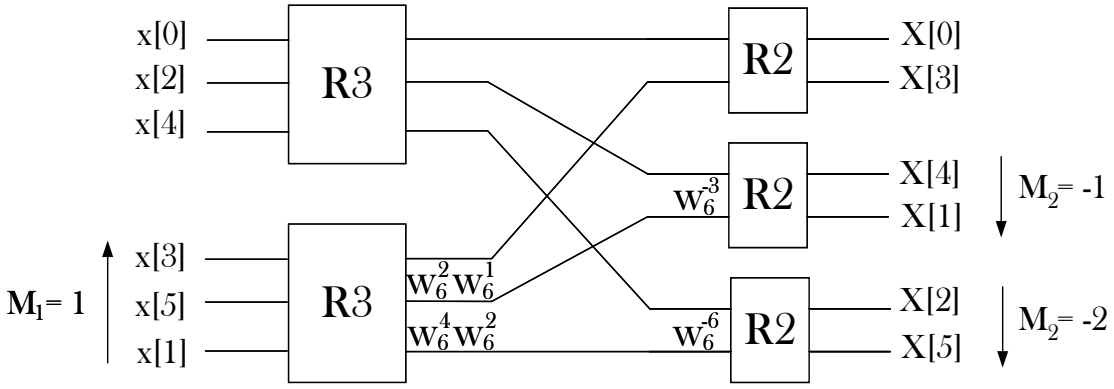

Figure 3.9: 6-point FFT using proposed approach for PFA.

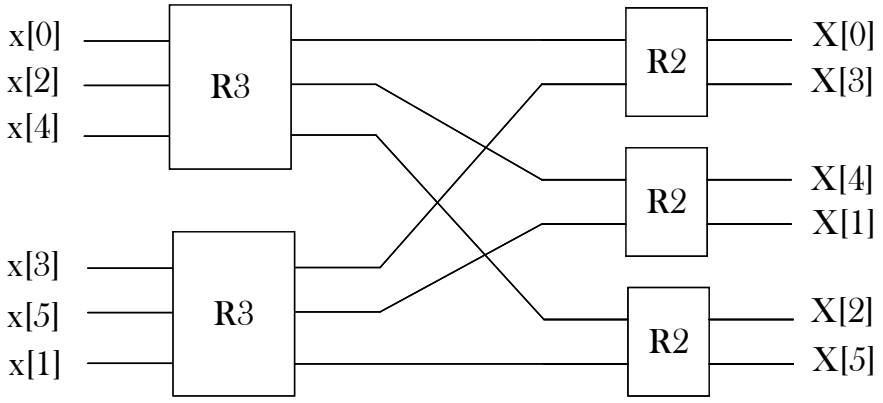

Figure 3.10: 6-point FFT using proposed approach for PFA.

15-point FFT

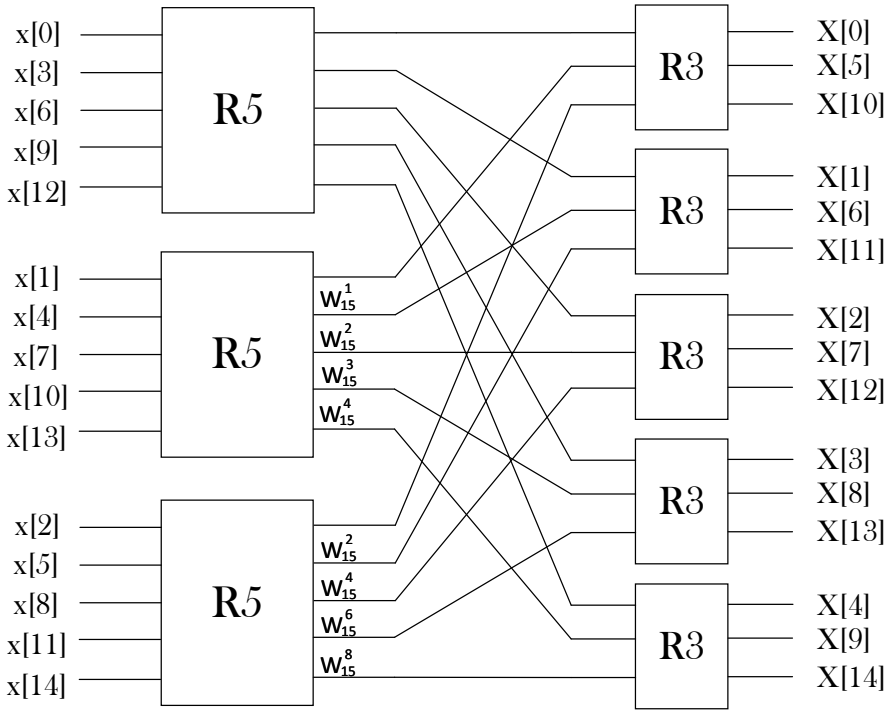

Figure 3.11: 15-point FFT using Cooley-Tukey algorithm.

Figure 3.11 shows a 15-point FFT algorithm using the Cooley-Tukey algorithm. It consists of radix-5 butterflies and radix-3 butterflies. Therefore $N_1 = 5$ and $N_2 = 3$. As it was done with the 6-point FFT,

$$M_1 = \frac{-1 - 5M_2}{3}. \tag{3.37}$$

Using the value $M_2 = 1$ then $M_1 = -2$ and, as a result, Figure 3.12 shows the optimized algorithm including the effect of the input and output shifting. Notice that the product of the values that share any branch is 1, which removes the twiddle factors. Figure 3.13 shows the proposed algorithm once the twiddle factors have been removed. This algorithm differs from the conventional PFA [21], which is shown in Figure 3.14.

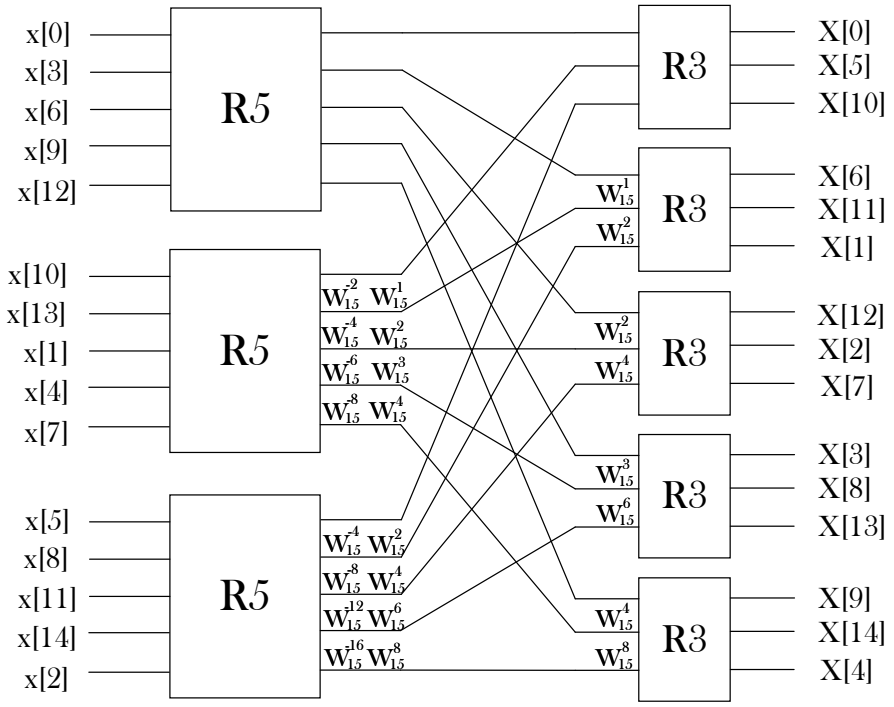

Figure 3.12: 15-point FFT using proposed approach for PFA showing the effects of input and output movements.

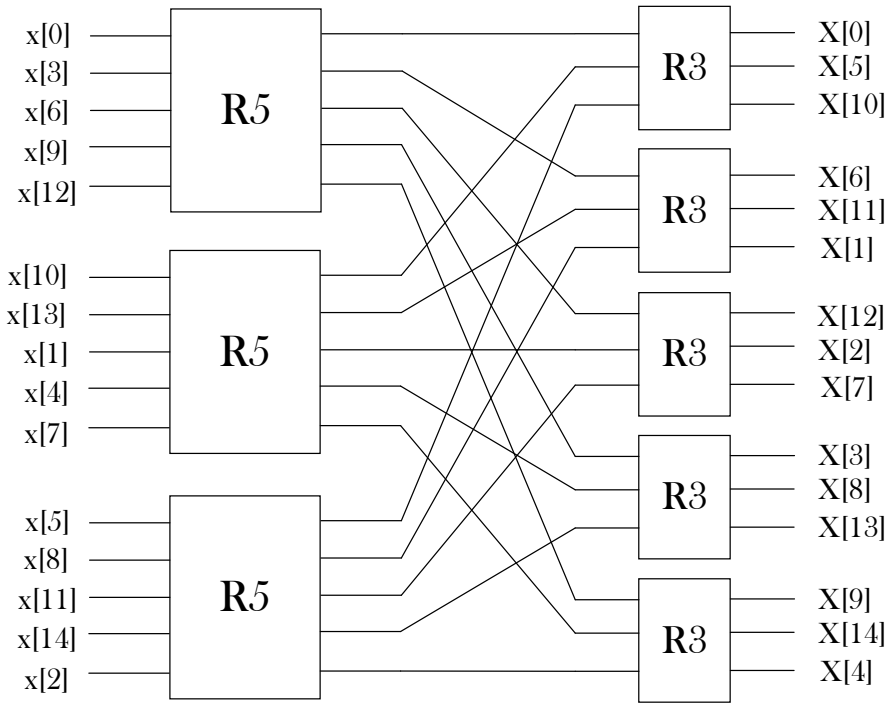

Figure 3.13: 15-point FFT using proposed approach for PFA.

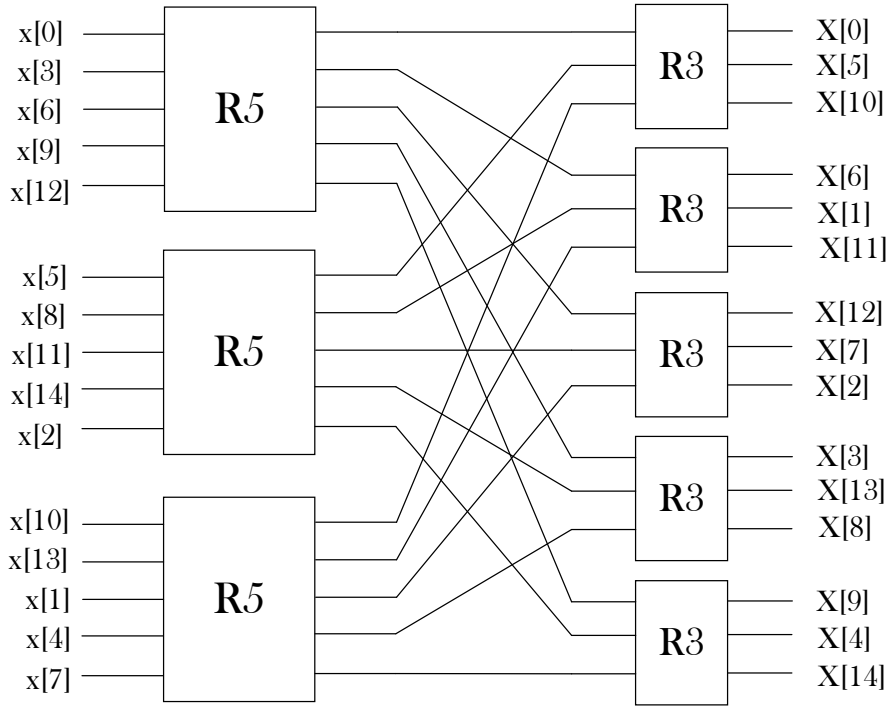

Figure 3.14: 15-point FFT based on index mapping of PFA.

30-point FFT

The 30-point FFT that has been designed consist of 3 stages of radix-5, radix-3 and radix-2 butterflies. Figure 3.15 shows a 30-point FFT algorithm flow graph using the Cooley-Tukey algorithm. As the previous methodology only considers two butterflies, one more step is needed. First it can be considered that there are only two butterflies: One radix-5 butterfly and other radix-6 butterfly. The condition for applying the previous methodology is that both sizes are relatively primes. $N_1 = 5$ and $N_2 = 6$ hold this condition and, consequently,

$$M_1 = \frac{-1 - 5M_2}{6}. \tag{3.38}$$

Figure 3.17 shows the 30-point FFT avoiding the twiddle factors by fixing $M_2 = 1$ and consequently $M_1 = -1$ and shifting the inputs and outputs of both butterflies respectively. Figure 3.16 also represents the internal connections of those radix-6 butterflies and the effect that appears in the algorithm when the input and output data are shifted. As the algorithm consist of radix-3 and radix-2 butterflies besides the radix-5 butterfly, the proposed algorithm of a 6-point FFT that was described in this Section can be reused. Figure 3.17 shows the same algorithm as in Figure 3.16 with the multiplications simplified. Figure 3.18 shows the same 30-point FFT algorithm once the butterflies have been unraveled. As a result, the proposed algorithm represents the same connection as the Cooley-Tukey algorithm, but avoiding the twiddle factors.

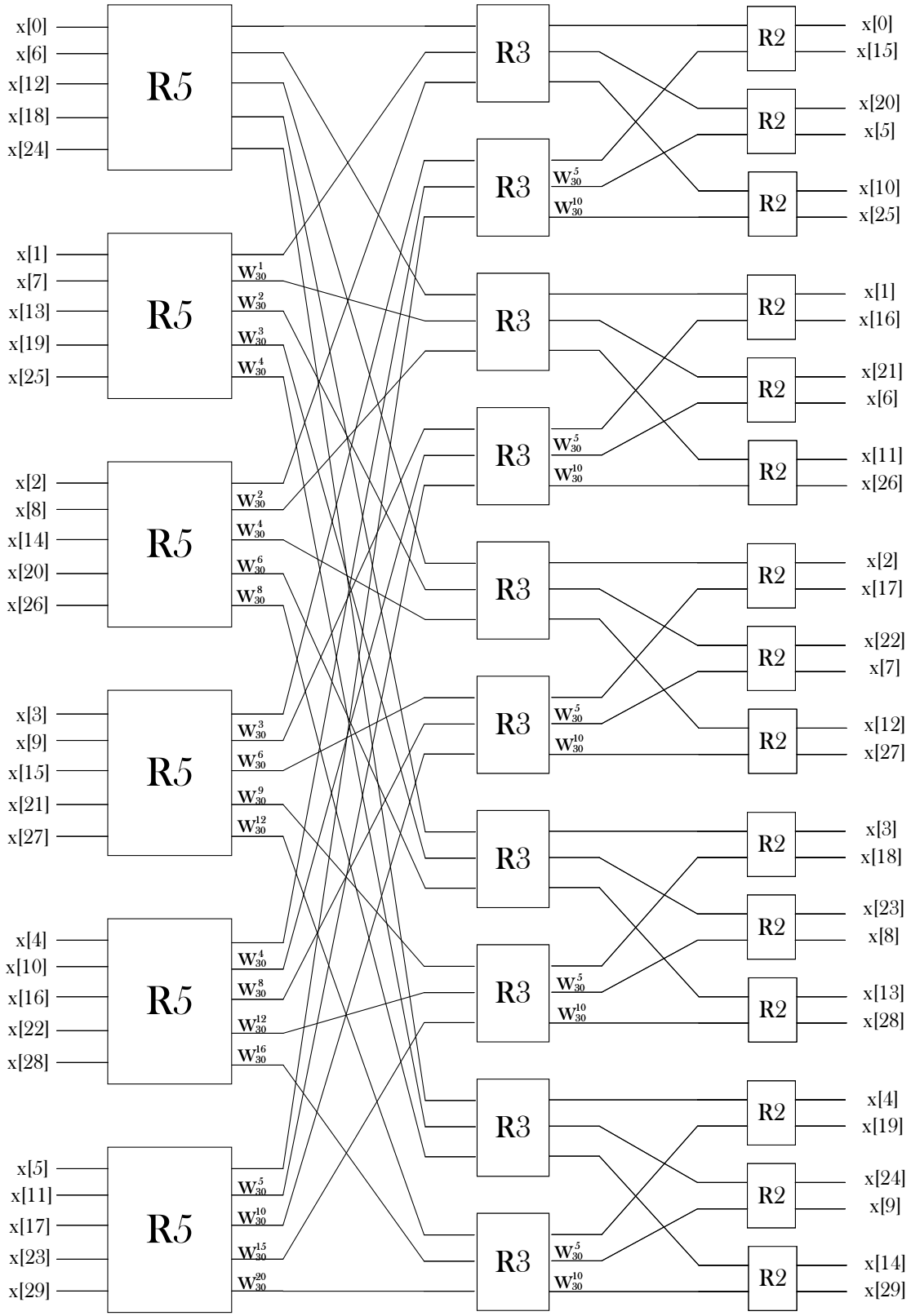

Figure 3.15: 30-point FFT using the Cooley-Tukey algorithm.

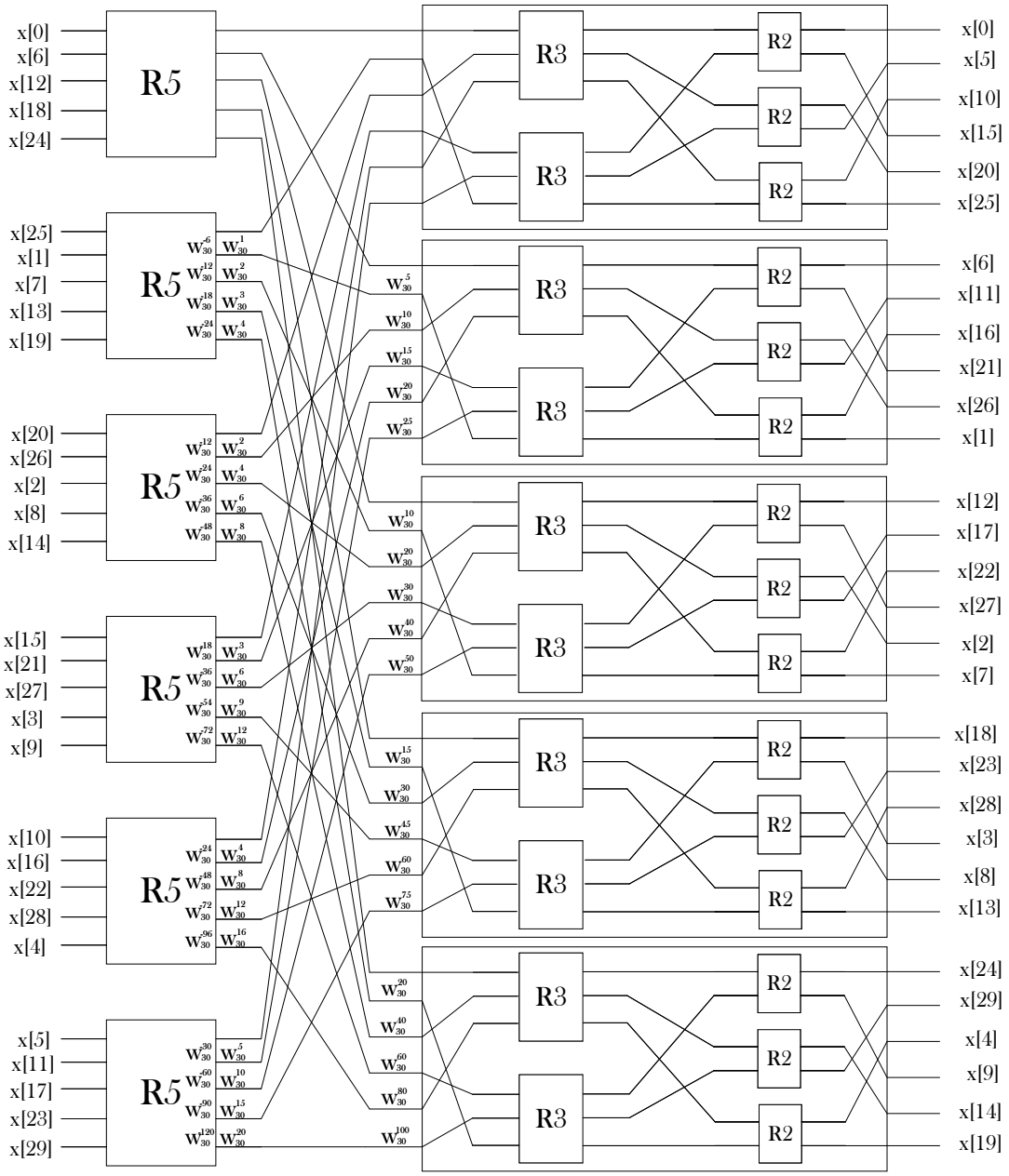

Figure 3.16: 30-point FFT using radix-5 and radix-6 butterflies showing the effects of the input and output movements.

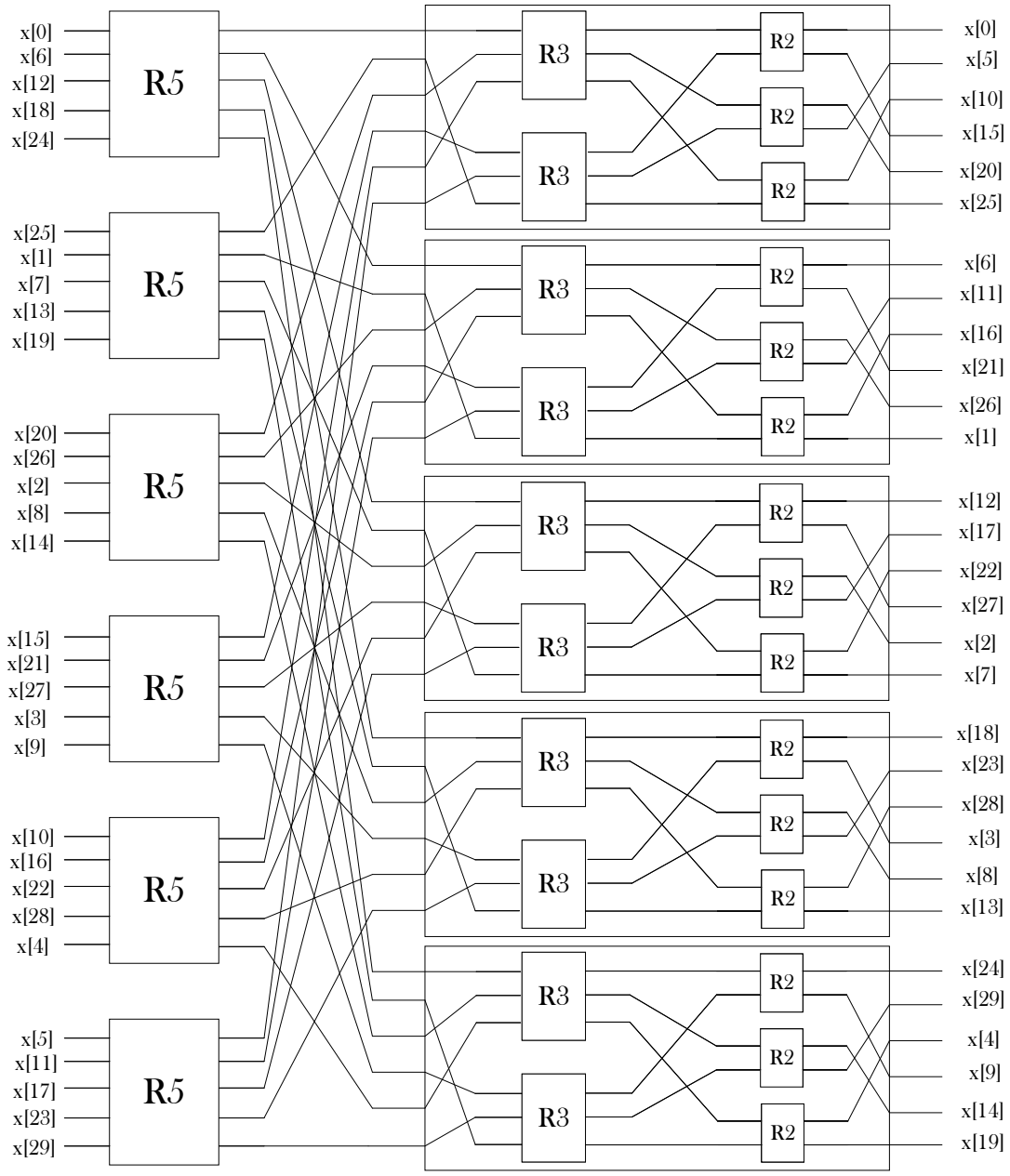

Figure 3.17: 30-point FFT using radix-5 and radix-6 butterflies.

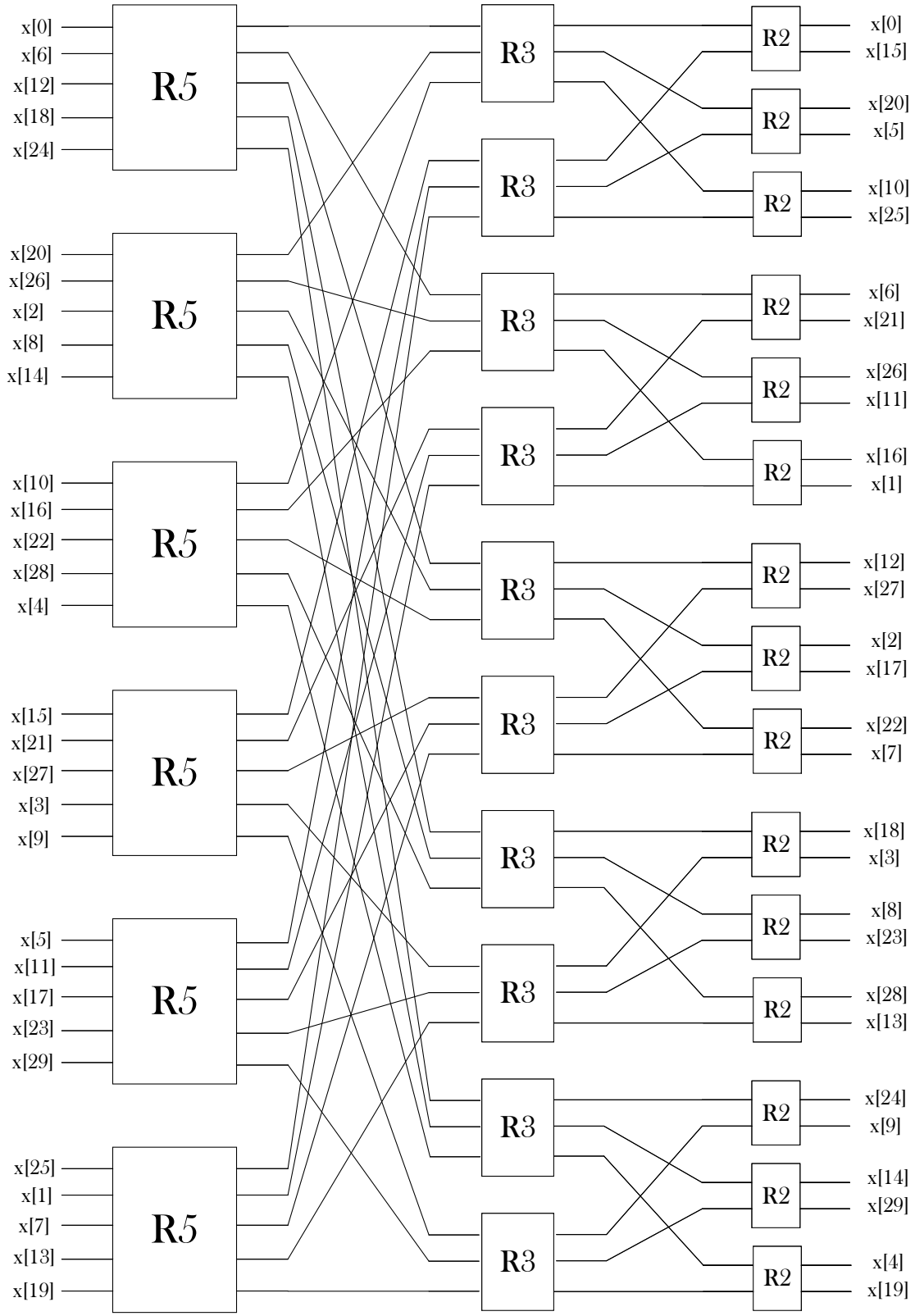

Figure 3.18: Proposed algorithm for a 30-point FFT.

Chapter 4

Implementation

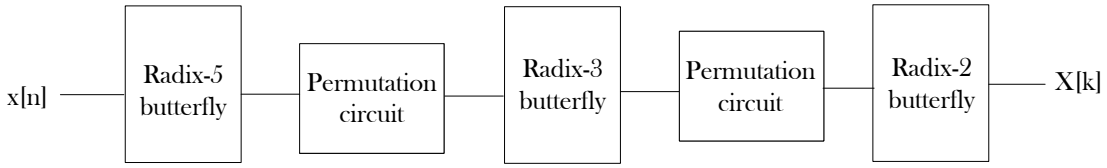

Figure 4.1: Hardware architecture of a serial pipelined 30-point FFT using input and output order of PFA.

In this Chapter, new serial pipelined NP2 architectures are presented. These architectures are based on the algorithms developed in Chapter 3. Figure 4.1 shows an overview of a 30-point FFT architecture. It consists of butterflies and permutation circuits. Compared to Figure 3.1, the architecture in Figure 4.1 does not include any rotator. This is due to the fact that the algorithms in Chapter 3 have been able to remove all the rotations in between butterflies.

As a result, the architecture only consists of butterflies and permutation circuits. The butterflies have already been described in Section 2.3, whereas the design of the permutations circuits is discussed along this Chapter. The proposed architectures work as follows: The architectures receive at their input one sample per clock cycle, which means that the circuit must process one sample per clock cycle. Consequently, each butterfly and each permutation circuit have a 100% utilization. All mathematical operations are calculated in the butterflies and permutation circuits reorder the output data of a butterfly before they arrive to the next one.

4.1 6-point FFT hardware implementation

Let us consider the flow graph in Figure 4.2. Based on it, we can assume that, in the architecture, data are received and provided from top to bottom in the flow

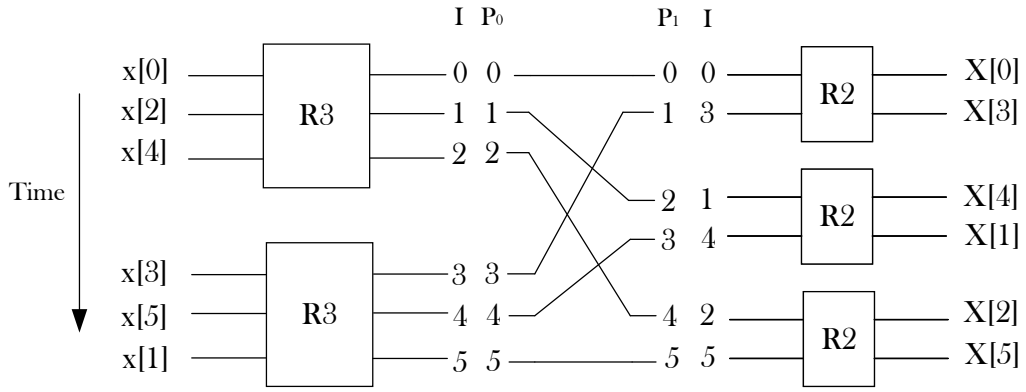

Figure 4.2: Flow graph of a 6-point FFT showing the permutation indexes.

graph. This way, each butterfly receives data in series in consecutive clock cycles and provides them also in series in consecutive clock cycles.

Figure 4.2 shows the connections between both butterflies. The order at the output of the radix-3 butterflies must be transformed into the order at the input of the radix-2 butterflies. For instance, the value with index $I = 3$ at the output of the radix-3 butterfly must be reordered so that it is connected to the branch of the radix-2 butterfly whose index is $I = 3$. Given the input and output orders, the number of cycles that a value must be moved can be calculated as [22]

$$\Delta = P_1 - P_0. \tag{4.1}$$

The values of P_1 and P_0 must be chosen according to the same indexes in Figure 4.2. For example, index 3 has $P_0 = 3$ and $P_1 = 1$ and, as a result, $\Delta = -2$. A negative value of Δ represents an advance forward in the data flow, and consecutively in time. A positive value of Δ represents a delay of the value. Table 4.1 shows the permutations that are done along the permutation circuit. The first column represents the index. The second column represents the number of positions that each value must be moved, Δ. A positive value means that the branch must be connected with an input branch of the second butterfly that is lower than the first one. In the table, a +2 movement means that this index have to go down 2 positions to reach its destiny. The methodology that has been follow to design the permutation circuit is to try movements between these values until all of them have reached their destination. The numbers that are in bold are interchanged. An important rule is that the number of positions in which the numbers can be permuted is constant in each stage. Thus, in the second column the bold numbers that differs in 1 positions are permuted leading to the third column. The third column permutes bold numbers that differ also in 1 position. The last column indicates that every sample has reached their destination and no more permutations are needed.

The minimum number of delays is the maximum absolute value among numbers in the second column and leads to the minimum latency L_{min}. Following the example of Figure 4.2, $L_{min} = 2$ and a possible circuit is shown in Figure 4.3. This circuit

Table 4.1: Design of the permutation circuit for a 6-points FFT.

P			
0	0	0	0
1	+1	**+1**	0
2	**+2**	**-1**	0
3	**-2**	**+1**	0
4	-1	**-1**	0
5	0	0	0
L		± 1	± 1

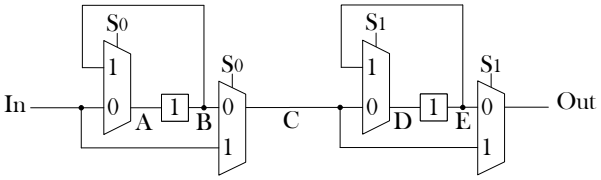

Figure 4.3: Permutation circuit for a 6-point FFT.

consists of two serial permutation circuits with buffers of length 1. Table 4.2 shows a timing diagram of the circuit in Figure 4.3. Notice that the output has 2 clock cycles of latency regarding the input. In fact, the circuit achieves the minimum latency.

Table 4.2: Timing diagram for the permutation circuit in Figure 4.3

Time	S0	S1	In	C	Out
0	0	x	0		
1	0	0	1	0	
2	0	0	2	1	0
3	1	1	3	3	3
4	0	0	4	2	1
5	0	1	5	4	4
6	0	0		5	2
7	0	0			5

As a result, Figure 4.4 shows the proposed hardware implementation of a 6-point FFT.

The proposed architecture is compared to an 8-point SDF FFT architecture in Table 4.3. The table shows the number of real multipliers, real adders, registers, latency and throughput of both architectures. It is considered that the SDF architecture is implemented with radix-2 butterflies and only has one rotator implemented as a complex multiplier. It is also considered that the complex multiplier consists of 4 real multipliers and 2 real adders. Both architectures are compared as if they were implemented using the minimum number of elements. This means that pipeline registers are not taken into account. Notice that the proposed architecture reduces the

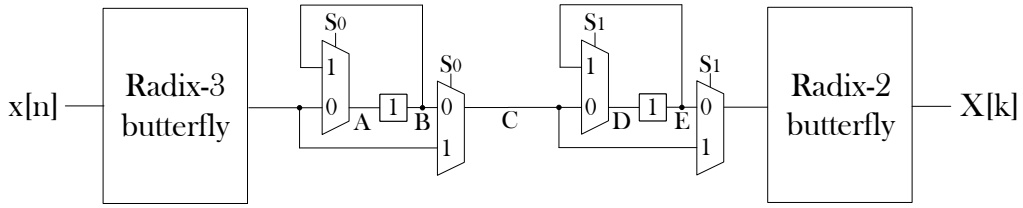

Figure 4.4: Hardware implementation of the serial pipelined 6-point FFT using the proposed input and output orders.

Table 4.3: Comparison between the proposed 6-point FFT and an 8-point radix-2 SDF architecture.

Architecture	Proposed	SDF
N	6	8
Real multipliers	1	4
Real adders	8	14
Registers	16	14
Latency (clock cycles)	8	7
Throughput (samples/cycle)	1	1

number of components with respect to the SDF architecture, due to the fact that the SDF architectures uses more adders and multipliers than the proposed architecture. The proposed architecture uses only one real multiplier in the radix-3 butterfly.

4.2 15-point FFT hardware implementation

Let us consider the flow graph of a 15-point FFT in Figure 3.13. The architecture uses a radix-5 butterfly in the first stage and a radix-3 butterfly in the second one. The design of the permutation circuit that connects both butterflies is done as it was derived in Section 4.1. Table 4.4 shows the methodology used to design the permutation circuits in the 15-point architecture, which follows the same procedure as in the 6-point FFT.

As a result, 3 serial permutation circuits have to be implemented. Two of them with a buffer length of 2 delays and the last one with a buffer length of 4 delays. The minimum number of delays was achieved. Figure 4.5 shows the proposed 15-point FFT architecture using the permutation circuit that results from the permutation methodology. The input and output orders that the proposed architecture expects are the ones shown in Figure 3.13, so that no rotators are needed to implement this FFT.

The proposed architecture is compared to a 16-point SDF FFT architecture in Table 4.5. The table shows the number of real multipliers, real adders, registers,

Table 4.4: Design of the permutation circuit for a 15-points FFT.

P				
0	0	0	0	0
1	+2	**+2**	0	0
2	+4	+4	**+4**	0
3	**+6**	**-2**	0	0
4	**+8**	0	0	0
5	**-4**	+4	**+4**	0
6	**-2**	**+6**	**-4**	0
7	0	0	0	0
8	**+2**	**-6**	**+4**	0
9	**+4**	-4	**-4**	0
10	**-8**	0	0	0
11	**-6**	**+2**	0	0
12	-4	-4	**-4**	0
13	-2	**-2**	0	0
14	0	0	0	0
L		± 2	± 2	± 4

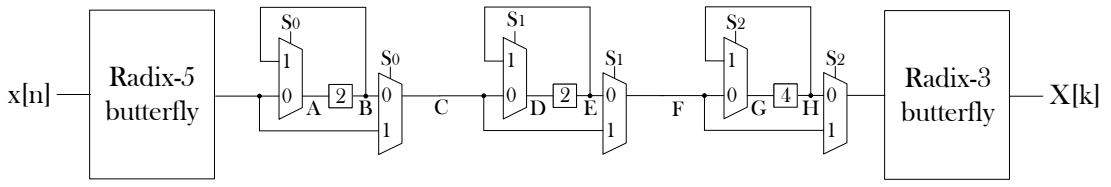

Figure 4.5: Hardware implementation of the serial pipelined 15-point FFT using the proposed input and output orders.

latency and throughput of both architectures. It is consider that the SDF architecture is implemented with radix-2 butterflies and only has one rotator implemented as a complex multiplier. It is also considered that the complex multiplier consists of 4 real multiplier and 2 real adders. Both architectures are compared as if they were implemented using the minimum number of elements. This means that pipeline registers are not taken into account. The proposed architecture uses a lower number of real multipliers and real adders than the SDF architecture. Consequently less logic is needed. However, it uses more registers than the SDF architecture and more latency is expected.

Table 4.5: Comparison between the proposed 15-point FFT and a 16-point radix-2 SDF architecture.

Architecture	Proposed	SDF
N	15	16
Real multipliers	3	4
Real adders	16	18
Registers	47	30
Latency (clock cycles)	21	15
Throughput (samples/cycle)	1	1

Table 4.6: Design of the permutation circuit for a 30-points FFT.

P							
0	**0**	+6	+6	**+6**	**+3**	**+1**	0
1	**+5**	-1	-1	-1	-1	**-1**	0
2	+10	**+10**	-1	-1	**-1**	**+1**	0
3	+15	+15	**+15**	-1	-1	**-1**	0
4	**+20**	**-3**	+8	**+8**	0	0	0
5	**-4**	**+19**	**+14**	+3	**+3**	**+1**	0
6	+1	+1	+1	+1	**+1**	**-1**	0
7	**+6**	**+12**	**+17**	**-4**	**-1**	**+1**	0
8	**+11**	+5	+5	**+5**	**-3**	**-1**	0
9	+16	**+16**	**+5**	0	0	0	0
10	-8	-8	**-8**	**+8**	0	0	0
11	**-3**	**+3**	**+14**	**-7**	+1	**+1**	0
12	**+2**	-4	**-4**	**+7**	-1	**-1**	0
13	+7	+7	**+7**	+2	**+2**	0	0
14	**+12**	-11	**-11**	**+10**	**+2**	0	0
15	**-12**	+11	**+11**	**-10**	**-2**	0	0
16	-7	-7	**-7**	-2	**-2**	0	0
17	**-2**	+4	**+4**	**-7**	+1	**+1**	0
18	**+3**	**-3**	**-14**	**+7**	-1	**-1**	0
19	+8	+8	**+8**	**-8**	0	0	0
20	-16	**-16**	**-5**	0	0	0	0
21	**-11**	-5	-5	**-5**	**+3**	**+1**	0
22	**-6**	**-12**	**-17**	**+4**	**+1**	**-1**	0
23	-1	-1	-1	-1	**-1**	**+1**	0
24	**+4**	**-19**	-14	-3	**-3**	**-1**	0
25	**-20**	**+3**	-8	**-8**	0	0	0
26	-15	-15	**-15**	+1	+1	**+1**	0
27	-10	**-10**	+1	+1	**+1**	**-1**	0
28	**-5**	+1	+1	+1	+1	**+1**	0
29	**0**	-6	-6	**-6**	**-3**	**-1**	0
L		±1	±2	±7	±7	±2	±1

4.3 30-point FFT hardware implementation

Let us consider the 30-point FFT flow graph in Figure 3.18. The order of the radices is 5, 3 and 2. This means that after the first stage, the FFT is divided into a 6-point FFT like the circuit that has already been implemented in hardware in Section 4.1. The design of the permutation circuit that connects the radix-5 butterfly and the radix-3 butterfly could not be solved manually as in the 6-point and 15-point FFTs. In this case, the complexity of the design is high, which made it necessary to develop a software program to explore a wide number of permutation patterns. The software can be seen as a function in which the parameters are the values of the buffers of each permutation circuit and it returns the optimum path that data must follow regarding the parameters if exists. The software returned a path using buffers of length {1,2,7,7,2,1}. The resulting data movement is shown in Table 4.6.

As a result, 6 serial permutation circuits have to be implemented. The minimum number of delays was achieved. Figure 4.6 shows the proposed architecture of a 30-point FFT using the permutation circuit that results from Table 4.6. The input and output order that the proposed architecture expects are the ones shown in Figure 3.13. Therefore, no rotators are needed at any stage.

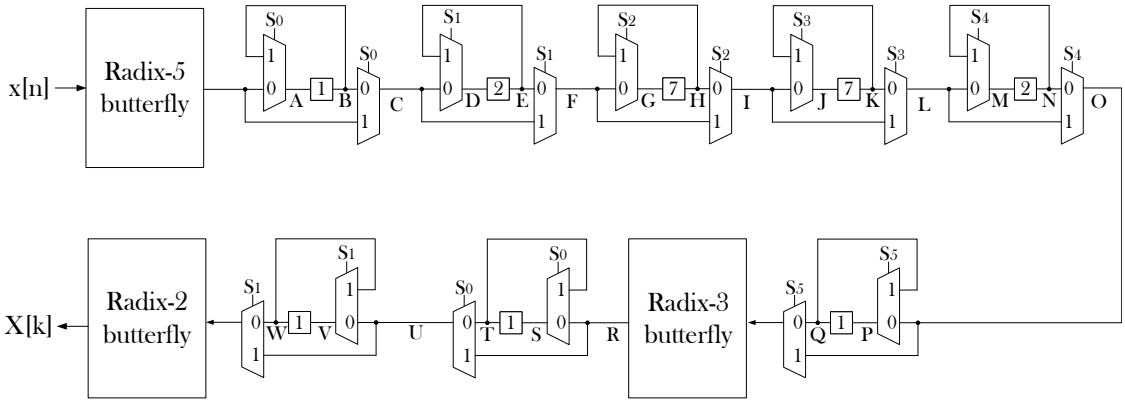

Figure 4.6: Hardware implementation of the serial pipelined 30-point FFT using the proposed input and output orders.

The proposed architecture is compared to a 32-point SDF FFT architecture. The table shows the number of real multipliers, real adders, registers, latency and throughput of both architectures. It is consider that the SDF architecture is implemented with radix-2 butterflies and has two rotators implemented as complex multipliers. It is also considered that a complex multiplier consists of 4 real multiplier and 2 real adders. Both architectures are compared as if they were implemented using the minimum number of elements. This means that pipeline registers are not taken into account. The proposed architecture uses only 3 real multipliers whereas the SDF architecture uses 8. Consequently less logic is required. However it uses

Table 4.7: Comparison between proposed architecture of a 30-point FFT and a 32-point radix-2 with an SDF architecture.

Architecture	Proposed	SDF
N	30	32
Real multipliers	3	8
Real adders	18	24
Registers	79	62
Latency (clock cycles)	37	31
Throughput (samples/cycle)	1	1

more registers than the SDF architecture and more latency is expected due to the latency of the permutation circuits that interconnect the butterflies.

4.4 Experimental results

The hardware circuits of the three proposed FFTs have been implemented on a Virtex Ultrascale+ HBM xcvu37p-fsvh2892-2L-e FPGA. NP2 butterflies were modified to add pipeline registers, so they reduce the combinational time and, consequently, increase the maximum operating frequency. The objective was to reach at least 500 MHz in each implemented FFT. Table 4.8 shows the comparison between the proposed architectures and power-of-two SDF FFTs from Xilinx [32]. The elements that are compared are CLBs, LUTs, registers, maximum operating frequency and power consumption at the frequency of 500 MHz. The power-of-two FFTs were implemented by using IP cores of Vivado. They were synthesized with a word length of 16 bits in a fixed point representation, and no growth in bits after an arithmetic operation was considered. The implementation was done using a clock constraint of $T = 2$ ns, so the power results are conditioned by that clock. The multipliers and the adders have been implemented as distributed logic, so DSP slices were not considered in any architecture.

Table 4.8: Comparison between the proposed architectures and Xilinx architectures.

Arquitecture	Proposed	Xilinx [32]	Proposed	Xilinx [32]	Proposed	Xilinx [32]
N	6	8	15	16	30	32
CLBs	91	219	255	256	322	396
CLB LUTs	462	1065	1360	1257	1767	2080
CLB REG	489	1678	1250	1984	1515	3130
f_{max}(MHz)	579	675	507	670	507	684
Latency (ns)	24	72	71	114	104	200
Power(mW) @500 MHz	62	130	181	172	231	268

As it can be seen in Table 4.8, the proposed NP2 FFT architectures improve the logic characteristics compared to the P2 ones as it was expected regarding Tables 4.3, 4.5 and 4.7. The CLBs used are significantly reduced in the 6-point FFT architecture compared to the 8-point FFT architecture. It improves the CLB resources by 58%, and CLB REGs are improved by 71%. Its power consumption has been reduced by 52%. A reason for this improvement is that only one real multiplier is used in the proposed architecture whereas the SDF architecture implements 4. The latency has been reduced by 71%.

The 15-point FFT architecture does not reduce the area, but it improves the latency by 54% compared to the 16-point FFT. Both architectures have similar elements regarding adders and multipliers, so that no difference in area is contemplated.

Finally, the proposed 30-point FFT implementation reduces the area by 19% and the power consumption by 24% compared to the 32-point FFT. This is the reduction expected, because in the SDF architecture 2 rotators are implemented, which corresponds to 8 real multipliers.

Conversely, only requires 3 real multipliers. As a result, the proposed NP2 FFT architectures improve P2 FFT architectures of similar sizes, specially in the case of the 6-point and 30-point FFTs.

Chapter 5

Conclusions and future work

5.1 Conclusions

This Master Thesis had the objective of progressing in the research line of non-power-of-two FFTs for 5G and beyond. The data order in NP2 FFTs has been studied and formalized. The proposed approach leads to an easier methodology than the common prime factor algorithm and allows to determine the order of input and output data. As in PFA, the proposed approach removes the twiddle factors between butterflies. The limitation for these optimizations is that the radices of the FFT butterflies must be coprime. FFTs with sizes 6, 15 and 30 were designed.

The butterflies of the proposed architectures were designed and implemented during my TFG. In order to connect the butterflies, serial permutation circuits have been designed. They are used to change the data order along a serial path. The design of these permutation circuits was done manually for the 6-point and 15-point FFTs. For the 30-point FFT a software was developed. The permutation circuits use the optimum number of registers and a low number of multiplexers.

The proposed hardware architectures have been implemented on an FPGA and were compared to power-of-two radix-2 SDF FFT architectures. This means that the proposed 6-point FFT was compared to an 8-point FFT, the 15-point to a 16-point one, and the 30-point to a 32-point FFT. As the proposed designs avoid twiddle factors, complex multipliers are not needed between stages. This leads to a reduction of area and power consumption when $N = 6$ and $N = 30$. Latency was also improved in every proposed architecture. Consequently it can be concluded that the proposed architectures have a high grade of optimization and represent a significant step towards the development of efficient FFT architectures for 5G and beyond.

5.2 Future work

The advances in this line of research suggest that non-power-of-two sizes for FFT designs can reach a high grade of optimization such as that in power-of-two ones or even more. The only condition is that the sizes of the butterflies have to be relatively prime. In order to continue scaling the size of the FFTs without using rotators, butterflies whose sizes are prime numbers can be developed such as a radix-7 butterfly.

An important future work is the development of larger FFTs for 5G, such as a 1200-point FFT. In this case, rotators will be required in some stages, but in some other stages they will be avoided.

Another future step of this TFM is the design of parallel FFT architectures for NP2. This will allow for increasing the throughput of the architectures towards the high performance expected in communication systems.

The experimental results of the circuits implemented on a FPGA were satisfactory, so that it is possible to continue with the implementation in ASIC, expecting good results as the FPGA ones in terms of area and power consumption.

Appendix A

Budget

LABOUR COSTS		Hours	Cost/hour	TOTAL
Junior Telecommunication Engineer		350	20 €	**7000,00 €**
MATERIAL COSTS	Cost (€)	Use (months)	Amort. (years)	TOTAL
Computer	800,00	9	5	**120,00 €**
VIVADO	2.995,00	9	5	**449,25 €**
FPGA Virtex Ultraescale+ HBM	10.262,00	9	5	**1.539,30 €**
MATLAB	2.000,00	9	5	**300,00 €**
SUBTOTAL DIRECT COSTS				**9.408,55 €**
INDIRECT COSTS		15%	of DC	**1.411,28 €**
INDUSTRIAL PROFIT		6%	of DC+IC	**649,19 €**
FUNGIBLE MATERIAL				
Office material				**25,00 €**
SUBTOTAL				**11.494,02 €**
IVA		21%		**2.413,74 €**
TOTAL BUDGET				**13.907,76 €**

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 Thesis. The general context of the proposed approach is the one that links electronic systems with telecommunication systems and signal processing.

B.1 Ethical and social impacts

The proposed approach for non-power-of-two FFT architectures can be used in new communications generations, because the sizes fix in the physical layer description of 5G. The ultimate goal of these systems is to achieve a better connected society.

B.2 Economic impact

The proposed approach tackles a key problem in 5G technologies: Although 5G expects FFT sizes that are a product of powers of 2, 3 and 5, NP2 FFTs were not optimized. As a result, FFT sizes in 5G were adapted to use powers of two. In this Master Thesis it has been shown that efficient architectures for NP2 can be designed. This can result in cutting-edge 5G systems that improve the current capabilities and create a technological advantage that brings profit to companies that use it.

B.3 Environmental impact

The extracted results from implementing in the FPGA the proposed designs says that the power consumption and the area was improved in most of the cases. As the proposed technology may be used in many communications devices, the power savings will scale significantly.

Bibliography

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

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

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

[4] 3GPP. 3GPP TS 38.211 - Physical channels and modulation V16.3.0. , September 2020.

[5] 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.

[6] J. Dai and H. Yin. Design and realization of non-radix-2 FFT prime factor processor for 5G broadcasting in release 16. In *Proc. Int. Symp. Comput. Intell. Design*, pages 406–409, December 2020.

[7] D.-Z. Qin, J.-A. Ren, and Y.-H. Xu. An efficient pruning algorithm for IFFT/FFT based on NC-OFDM in 5G. In *Proc. Int. Conf. Inventive Comm. Comput. Tech.*, pages 432–435, April 2018.

[8] Manuel Viqueira. Design of a low-power FFT architecture for 6G. Master's thesis, Dpt. of Electronic Engineering, Technical University of Madrid, July 2021.

[9] M. Garrido, F. Qureshi, J. Takala, and O. Gustafsson. Hardware architectures for the fast Fourier transform. In S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, editors, *Handbook of Signal Processing Systems*. Springer, third edition, 2019.

[10] M. Garrido, M.A. Sánchez, M.L. López-Vallejo, and J. Grajal. A 4096-point radix-4 memory-based FFT using DSP slices. *IEEE Trans. VLSI Syst.*, 25(1):375–379, January 2017.

[11] N. Bhagat, D. Valencia, A. Alimohammad, and F. Harris. High-throughput and compact FFT architectures using the Good–Thomas and Winograd algorithms. *IET Commun.*, 12(8):1011–1018, May 2018.

[12] Z.-G. Ma, X.-B. Yin, and F. Yu. A novel memory-based FFT architecture for real-valued signals based on a radix-2 decimation-in-frequency algorithm. *IEEE Trans. Circuits Syst. II*, 62(9):876–880, Sept 2015.

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

[14] M. Garrido, R. Andersson, F. Qureshi, and O. Gustafsson. Multiplierless unity-gain SDF FFTs. *IEEE Trans. VLSI Syst.*, 24(9):3003–3007, September 2016.

[15] M. Bansal and S. Nakhate. High speed pipelined 64-point FFT processor based on radix-2^2 for wireless LAN. In *Proc. Int. Conf. Signal Process. Integrated Networks*, pages 607–612, February 2017.

[16] P. Paz. Design spaced exploration of FFT architectures using rotator allocation. Master's thesis, Dpt. of Electronic Engineering, Universidad Politécnica de Madrid, July 2020.

[17] V. M. Bautista. Design and implementation of FFT architectures with non-power-of-two sizes for 5G. Bachelor's thesis, Dpt. of Electronic Engineering, Universidad Politécnica de Madrid, July 2021.

[18] I. J. Good. The interaction algorithm and practical Fourier analysis. *J. Roy. Statist. Soc. Ser. B*, 20(2):361–372, July 1958.

[19] S. B. Weinstein. The history of orthogonal frequency-division multiplexing. *IEEE Comm. Magazine*, 47(11):26–35, 2009.

[20] C Burrus. Index mappings for multidimensional formulation of the DFT and convolution. *IEEE Trans. Acoust., Speech, Signal Process.*, 25(3):239–242, June 1977.

[21] S. K. Mitra. *Discrete Signal Processing: A Computer Based Approach.* McGraw-Hill, 4 edition, 2011.

[22] M. Garrido, J. Grajal, and O. Gustafsson. Optimum circuits for bit-dimension permutations. *IEEE Trans. VLSI Syst.*, 27(5):1148–1160, May 2019.

[23] 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.

[24] J. Löfgren and P. Nilsson. On hardware implementation of radix-3 and radix-5 FFT kernels for LTE systems. In *Proc. NORCHIP*, pages 1–4, November 2011.

[25] A. M. Despain. Fourier transform computers using CORDIC iterations. *IEEE Trans. Comput.*, C-23:993–1001, October 1974.

[26] A. M. Despain. Very fast Fourier transform algorithms hardware for implementation. *IEEE Trans. Comput.*, C-28(5):333–341, May 1979.

[27] S. Winograd. On computing the discrete Fourier transform. *Math. Comput.*, 32(141):175–199, January 1978.

[28] C. M. Rader. Discrete Fourier transform when the number of data samples is prime. *Proc. IEEE*, 56(6):1107–1108, June 1968.

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

[30] M. Garrido, N. K. Unnikrishnan, and K. K. Parhi. A serial commutator fast Fourier transform architecture for real-valued signals. *IEEE Trans. Circuits Syst. II*, 65(11):1693–1697, November 2018.

[31] 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.

[32] Xilinx - FFT Logicore 9.1, May 2022. https://china.xilinx.com/content/dam/xilinx/support/documents/ip_documentation/xfft/v9_1/pg109-xfft.pdf.