The Constant Multiplier FFT

Mario Garrido, *Senior Member, IEEE*, and Pedro Malagón

***Abstract*— In this paper, we present a new fast Fourier transform (FFT) hardware architecture called constant multiplier (CM) FFT. Whereas rotators in previous architectures must rotate among several different angles, the CM FFT exploits the use of constant multipliers to calculate the rotations. The paper explores the 4-parallel and 8-parallel radix-2 CM FFT, which calculates the FFT entirely by using constant multipliers. Later, radices 2^k are presented, with emphasis in radix-2^4 and radix-2^5 as the best alternatives. Experimental results for a 1024-point radix-2^5 CM FFT show that the proposed approach reduces the number of block random-access memories (BRAM) and digital signal processing (DSP) slices with respect to previous approaches, while achieving the highest clock frequency for a 4-parallel FFT architecture on field-programmable gate arrays (FPGAs) reported so far.**

***Index Terms*— Fast Fourier Transform (FFT), pipelined architecture, constant multiplier (CM).**

I. Introduction

THE fast Fourier transform (FFT) is one of the most important algorithms in the field of digital signal processing. It is a key component in applications such as digital communications [1]–[5], radio astronomy [6], [7] and medical imaging [8].

In order to meet the high performance and real-time requirements of modern applications, hardware engineers aim for designing efficient architectures for the computation of the FFT. In this context, pipelined hardware architectures [1]–[24] are widely used, because they provide high throughput and low latency suitable for real-time processing, as well as a reasonably low area and power consumption.

There are three main types of pipelined FFT architectures: Feedback, feedforward and serial commutator (SC). First, feedback architectures [3]–[6], [9]–[12] are characterized by their feedback loops, i.e., some outputs of the butterflies are fed back to the memories at the same stage. Feedback architectures are divided into single-path delay feedback (SDF) architectures [6], [9], [10], which process a continuous flow of one sample per clock cycle, and multi-path delay feedback (MDF) architectures [2]–[5], [11]–[14], which process several samples in parallel. Second, feedforward architectures [1], [10], [15]–[18] do not have feedback loops and each stage passes the processed data to the next stage. Among feedforward architectures, multi-path delay commutator (MDC) architectures [1], [10], [15]–[18] are the most common ones and process several samples in parallel. Finally, SC FFT architectures [19], [20] are characterized by the use of circuits for bit-dimension permutation of serial data. They can process either serial data [19] or parallel data [20] in a continuous flow.

In pipelined FFT architectures, the amount of hardware resources is measured in terms of adders, rotators and memory. Among them, rotators are generally the most costly components. Consequently, the design of efficient rotators is a fundamental challenge for parallel pipelined FFT architectures.

FFT rotators can be implemented in various ways depending on the target twiddle factor, which is the set of rotations that a rotator can calculate. Large twiddle factors are usually implemented by using general multipliers [25] or CORDIC rotators [26]–[30]. Twiddle factors of intermediate size (W_8 and W_{16}) are generally implemented as shift-and-add operations [4], [31]–[38]. Conversely, constant rotators [39]–[44] are not used in FFT architectures except in the case of a fully parallel implementation of the FFT [45].

W_8, W_{16} and general rotators are configured to rotate by a set of angles. This means that data flowing through the path of any of those rotators demand different rotation angles at different clock cycles. This fact has double impact in the area of the FFT architecture. On the one hand, the capacity of rotating by several angles increases the complexity of the rotator itself. On the other hand, the twiddle factors need to be stored in a rotation memory. For the entire FFT architecture, this memory may take up an area equivalent to the data memory.

Therefore, it would be desired to use constant rotators for the FFT. They have smaller area usage than other rotators and do not need to store any twiddle factor in rotation memory, because they rotate by a single angle and there is no need to choose among several of them. However, in order to have a constant rotation in an FFT path, it is needed that all the data flowing through that path are rotated by the same angle, which is not the case in common FFT architectures.

In this work we present the constant multiplier (CM) FFT. The proposed CM FFT has the property that it uses constant multipliers for calculating the rotations. This is achieved by grouping data that are rotated by the same angle, and making them follow the same path in the architecture. The CM FFT is proposed for 4-parallel and 8-parallel data and for decimation in frequency (DIF) and decimation in time (DIT)

Manuscript received July 28, 2020; revised September 23, 2020; accepted October 8, 2020. Date of publication October 27, 2020; date of current version December 21, 2020. This work was supported in part by the Spanish Ministry of Science, Innovation and Universities under the Ramón y Cajal Grant RYC2018-025384-I; and in part by the Spanish MINECO under Grant PID2019-110866RB-I00. This article was recommended by Associate Editor G. Jovanovic Dolecek. *(Corresponding author: Mario Garrido.)*

The authors are with the Department of Electronic Engineering, ETSI de Telecomunicación, Universidad Politécnica de Madrid, 28040 Madrid, Spain (e-mail: mario.garrido@upm.es; malagon@die.upm.es).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2020.3031688

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

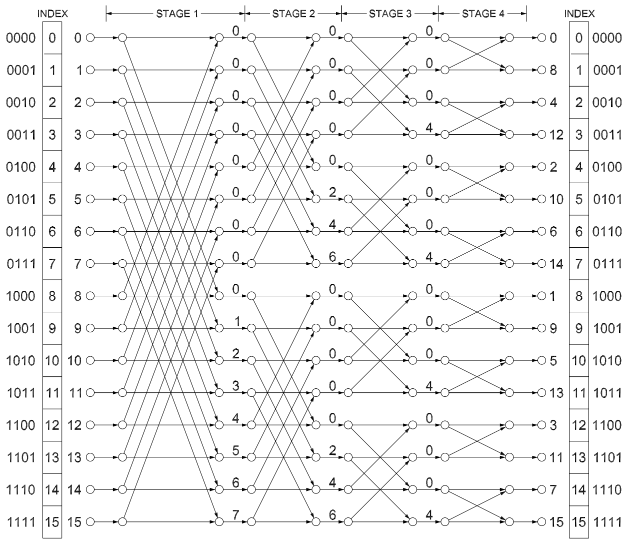

Fig. 1. Flow graph of a 16-point radix-2 DIF FFT.

decompositions. Furthermore, both radix-2 and radix-2^k CM FFT architectures are explored. As a proof of concept, a 1024-point radix-2^5 CM FFT has been implemented on a field-programmable gate array (FPGA). Compared to previous approaches, the CM FFT is shown to be more efficient in terms of block random access memories (BRAMs) and digital signal processing (DSP) slices than previous MDF and MDC FFTs, while achieving a higher clock frequency. This makes the CM FFT an attractive solution for processing a continuous dataflow at high rates.

This paper is organized as follows. In Section II, we review the FFT algorithm. In Section III, we explain how to extract constant multiplications from the FFT flow graph. In Section IV, we present the CM FFT and develop it for different radices, parallelization and decomposition. In Section V, we show the hardware implementation of a 1024-point radix-2^5 CM FFT. In Section VI, we present the experimental results and compare them to previous parallel pipelined FFTs. In Section VII, we discuss the main findings of the proposed approach. Finally, in Section VIII, we summarize the main conclusions of the paper.

II. The FFT Algorithm

The N-point discrete Fourier transform (DFT) of an input sequence $x[n]$ is defined as:

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

where $W_N^{nk} = e^{-j\frac{2\pi}{N}nk}$.

To calculate the DFT, the FFT based on the Cooley-Tukey algorithm [46] is mostly used. The Cooley-Tukey algorithm reduces the number of operations from $\mathcal{O}(N^2)$ for the DFT to $\mathcal{O}(N \log_2 N)$ for the FFT.

Figure 1 shows the flow graph of a 16-point radix-2 FFT according to the Cooley-Tukey algorithm, decomposed according to the DIF decomposition [22], [47]. The FFT consists of $n = \log_2 N$ stages. At each stage of the graph, $s \in \{1, \ldots, n\}$, butterflies and rotations are calculated. The lower edges of the butterflies are always multiplied by -1. These -1 are not depicted in order to simplify the graphs.

The numbers at the input represent the index of the input sequence, whereas those at the output are the frequencies, k, of the output signal $X[k]$. Finally, each number, ϕ, in between the stages indicates a rotation by

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

As a consequence, data for which $\phi = 0$ do not need to be rotated. Likewise, if $\phi \in [0, N/4, N/2, 3N/4]$, data must be rotated by 0°, 270°, 180° and 90°, which correspond to complex multiplications by 1, $-j$, -1 and j, respectively. These rotations are considered to be trivial, because they can be calculated by interchanging the real and imaginary components and/or changing the sign of the data.

An index $I \equiv b_{n-1} \ldots b_0$ is also added to the left and to the right of the flow graph, where $b_{n-1} \ldots b_0$ is the binary representation of I. It can be observed that the butterflies at stage s operate on pairs of data that differ in bit b_{n-s} [15].

III. Obtaining Constant Multiplications in the FFT

The twiddle factor multiplications in the flow graph of Fig. 1 are constant complex multiplications. However, in FFT architectures it is common that several data flowing through the same path must be rotated by different angles. Consequently, rotators that can rotate among a set of different angles are needed. This includes general rotators and rotators by W_8 and W_{16}. Contrarily to previous FFT architectures, in this paper we are interested in using only constant complex multipliers in the FFT, as explained next.

It is known [47] that rotations for a radix-2 DIF FFT are calculated as

$$\phi_s(I) \equiv b_{n-s} \cdot 2^{s-1} \cdot [b_{n-s-1} \ldots b_0], \tag{3}$$

where b_i are the bits of the index I associated to the rotation, as shown in the flow graph of Fig. 1. For instance, in the first stage of the flow graph ($s = 1$), a rotation by $\phi = 3$ is associated with the index $I = 11 \equiv 1011 = b_3\ b_2\ b_1\ b_0$. According to equation (3), $\phi_1(11) \equiv b_{4-1} \cdot 2^{1-1} \cdot [b_{4-1-1} \ldots b_0] = b_3 \cdot [b_2\ b_1\ b_0] = 1 \cdot [011] \equiv 3$, as expected.

Likewise, for the radix-2 DIT FFT, the rotations are obtained as [47]

$$\phi_s(I) \equiv b_{n-s-1} \cdot 2^{n-s-1} \cdot [b_{n-s} \ldots b_{n-1}]. \tag{4}$$

Continuing with the radix-2 DIF case, if we particularize equation (3) for the 16-point FFT in Fig. 1, the rotations for the first three stages are obtained as

$$\begin{aligned}
\phi_1(I) &\equiv b_3 \cdot 2^0 \cdot [b_2 b_1 b_0] = 4 \cdot b_3 b_2 + 2 \cdot b_3 b_1 + 1 \cdot b_3 b_0, \\
\phi_2(I) &\equiv b_2 \cdot 2^1 \cdot [b_1 b_0] = 4 \cdot b_2 b_1 + 2 \cdot b_2 b_0, \\
\phi_3(I) &\equiv b_1 \cdot 2^2 \cdot [b_0] = 4 \cdot b_1 b_0.
\end{aligned} \tag{5}$$

This highlights that $\phi_s(I)$ is obtained as a combination of constant multiplications. For instance, rotations in the first

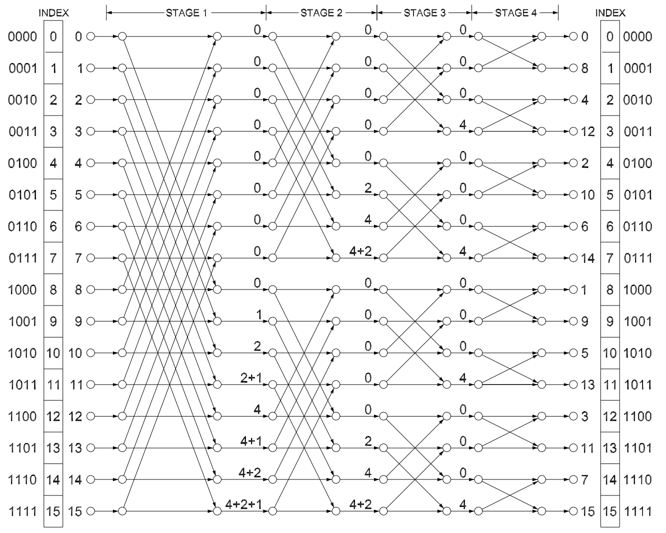

Fig. 2. Flow graph of a 16-point radix-2 DIF FFT where twiddle factors are decomposed into constant rotations.

stage are a combination of constant multiplications by $\phi = 4$, $\phi = 2$ and $\phi = 1$. The exact value of the rotation depends on the bits b_i of the index. Again for the case of $\phi_1(11)$ we obtain $\phi_1(11) \equiv 4{\cdot}b_3\, b_2{+}2{\cdot}b_3\, b_1{+}1{\cdot}b_3\, b_0 = 4{\cdot}1{\cdot}0{+}2{\cdot}1{\cdot}1{+}1{\cdot}1{\cdot}1 = 3$. This sum of constant rotations is represented in Fig. 2.

For the general case of any N, from (3) we obtain

$$\begin{aligned}\phi_s(I) \equiv\; & 2^{n-2} \cdot b_{n-s}b_{n-s-1} + \\ & +2^{n-3} \cdot b_{n-s}b_{n-s-2} + \\ & +2^{n-4} \cdot b_{n-s}b_{n-s-3} + \\ & + \ldots + \\ & +2^{2} \cdot b_{n-s}b_{2} + \\ & +2^{1} \cdot b_{n-s}b_{1} + \\ & +2^{0} \cdot b_{n-s}b_{0}. \end{aligned} \quad (6)$$

Therefore, the rotation by $\phi_s(I)$ is decomposed into a sum of constant rotations by $\phi = 2^i, i\ = 0 \ldots n-2$. In radians, this corresponds to the angles

$$\alpha_i = \frac{2\pi}{N}\phi = \frac{2\pi}{N}2^i = \frac{2\pi}{2^{n-i}}, \qquad i = 0, \ldots, n-2. \quad (7)$$

The next step is to design the architecture. For this purpose, we have to map the rotations to rotators. Fig. 3 shows the first stage of a 16-point 4-parallel radix-2 DIF CM FFT architecture. It consists of butterflies (R2), constant rotators (⊗), trivial rotators (diamond-shaped) and shuffling circuits. The shuffling circuits include multiplexers and buffers represented by boxes. The number inside each buffer indicates the length of the buffer. The data order at each place of the circuit is represented at the bottom of the figure by using matrices. These matrices include the data indexes according to Fig. 2. Each row of the matrix includes the indexes that flow through one of the paths of the circuit. This way, the first row includes the data flowing through the upper path and the lower row includes the data flowing through the lower path. Each column of the matrix indicates the instant at which each datum arrives. The values to the right (column 4) arrive first, followed by the values at columns 3, 2 and 1. As an example, data with indexes 9, 13, 11 and 15 arrive to the butterflies of the first stage at consecutive clock cycles at the lowest path of the architecture.

The bits b_i close to the matrices are a more compact way to represent the orders in the matrix. They show which bit b_i of the index varies at each dimension of the matrix. This representation is described in [48] (Section III).

The buffers and multiplexers in the architecture carry out the data shuffling. Figure 4 shows how they work for a buffer size of length L. First, the multiplexers are set to 0 for L clock cycles. Thus, the first L samples from the upper path (set A) are stored in the output buffer. At the same time, the first L samples from the lower path (set C) are stored in the input buffer. Next, the multiplexer is set to 1 for L clock cycles. This makes C pass to the output buffer and D is stored in the input buffer. At the same time, sets A and B are provided in parallel at the output. When the multiplexer commutes again to 0, sets C and D are provided in parallel. As a result, sets B and C are interchanged. More detail on the shuffling circuits can be found in [48].

The rotators at the FFT stage depend on the data that flow through each path. In fact, the rotator at each path must be able to rotate by all the rotations applied to the data flowing through that path. In the CM FFT, the rotations at the first stage are split into three rotators by W_{16}^1, W_{16}^2 and W_{16}^4, which are marked as 1, 2 and, 4 in Figs. 2 and 3. According to Fig. 2, the rotation by 1 at the first stage occurs only for indexes 9, 11, 13 and 15. In Fig. 3 these indexes flow through the lowest path, as can be observed from the matrix related to the rotator by 1. This is why the rotator by 1 is placed at the lowest path of the architecture. Likewise, rotations by 2 in Fig. 2 occur for indexes 10, 11, 14 and 15, which are at the lowest path after the shuffling circuits with buffer length equal to 2. As a result, the rotator by 2 only needs to be placed at the lowest path at this place of the circuit. Finally, rotation by 4 occurs for data with indexes 12, 13, 14 and 15 and corresponds to the lowest path before the butterflies of the second stage. As a result, the rotations by the angles of the first stage are calculated by means of three constant rotators. This is possible thanks to the shuffling circuits at the first stage, which place data that must be rotated by the same constant angle at the same path of the architecture.

IV. Proposed Constant Multiplier FFT

A. 4-Parallel Radix-2 DIF CM FFT

Fig. 5 shows the proposed 16-point 4-parallel radix-2 DIF CM FFT architecture. It consists of four stages with radix-2 butterflies (R2), constant complex multipliers, trivial multipliers and shuffling circuits.

The first stage of the architecture in Fig. 5 is similar to that in Fig. 3. The only difference is that extra registers have been removed. Note that in Fig. 3 there are four registers in parallel in a vertical cut after the rotation by $\phi = 2$. Whereas the pipelining technique introduces registers in a vertical cut of a hardware architecture with the goal of reducing the critical path, in this case we apply the same principle in the opposite direction, i.e., instead of adding registers, we remove registers

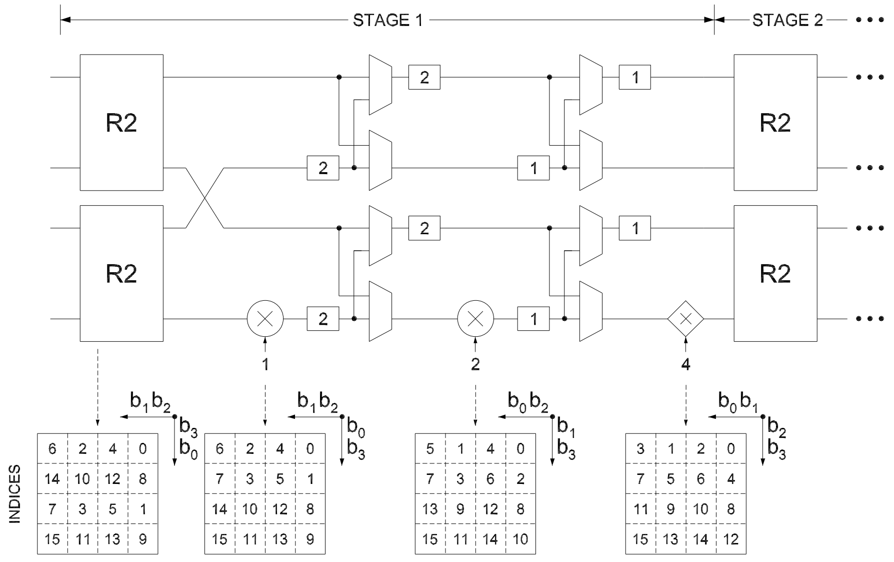

Fig. 3. First stage of the 16-point 4-parallel radix-2 DIF CM FFT architecture. The stage consists of radix-2 butterflies (R2), constant complex multipliers (⊗), trivial multipliers (diamond shaped) and circuits for data permutation that consist of buffers and multiplexers. The data indexes of the dataflow are shown at the bottom of the figure.

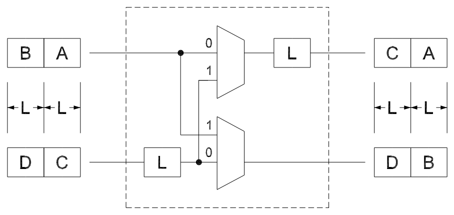

Fig. 4. Shuffling circuit used in the CM FFT architecture.

that appear in all the paths of a vertical cut of the architecture. This results in a reduction of the number of registers. Indeed, removing extra registers between serial-parallel permutation is always possible when the serial-parallel permutations share the parallel dimension [48].

Fig. 6 shows the proposed 32-point 4-parallel radix-2 DIF CM FFT architecture. As the 16-point one, it only uses constant complex multipliers. By comparing the 16-point and 32-point architectures, it can be observed that, at each stage, the 32-point architecture has one permutation circuit and one rotator more than those in the 16-point FFT.

For a general N, the amount of resources of the proposed 4-parallel radix-2 CM FFT can be calculated as follows. First, the shuffling circuits with buffers of length 2^i appear at $n-1-i$ stages and their total memory is $4 \cdot 2^i = 2^{i+2}$ due to the four parallel paths. The buffer lengths range from $2^0 = 1$ to 2^{n-3} and, thus, the index ranges from $i = 0$ to $i = n-3$. Furthermore, shuffling circuits of sizes 2^i and 2^{i-1} are connected, leading to $4 \cdot 2^{i-1} = 2^{i+1}$ registers that appear in parallel, 2^i in each parallel path. This forms pipeline sections that can be removed. These sections range from $i = 1$ to $i = 3$ and appear at $n-1-i$ stages. As a result, the memory of the architecture for any N is

$$\text{Total Mem.} = \sum_{i=0}^{n-3}(n-1-i)2^{i+2} - \sum_{i=1}^{n-3}(n-1-i)2^{i+1}, \tag{8}$$

where the first sum is the total number of registers and the second sum is the number of registers in parallel paths that can be removed. This leads to a total memory for the entire FFT of

$$\text{Total Mem.} = \frac{3}{2}N - 4. \tag{9}$$

Second, each radix-2 butterfly includes two complex adders and processes two parallel paths. Therefore, at each stage there is one complex adder per parallel path, being the number of complex adders in the architecture

$$\text{Total Adders} = 4n = 4\log_2 N. \tag{10}$$

Third, there are constant complex rotators at stages $i = 1$ to $i = n-2$ and the number of them at the i-th stage is $n-1-i$, leading to a total number of constant complex rotators for the entire architecture of

$$\text{Total Constant Rot.} = \sum_{i=1}^{n-2}(n-1-i) = \frac{(n-1)(n-2)}{2} \tag{11}$$

Trivial rotators are not considered in this calculation, because they can be merged with the butterflies before or after them.

Finally, the number of multiplexers is calculated considering that each stage $i \in [1, n-1]$ includes $4 \cdot (n-i)$ multiplexers, except the first stage, which includes four less. This leads to a total number of multiplexers

$$\text{Total Mux.} = 4\left(\sum_{i=1}^{n-1}(n-i)\right) - 4 = 2n(n-1) - 4. \tag{12}$$

B. 4-Parallel Radix-2 DIT CM FFT

Fig. 7 shows the 16-point 4-parallel radix-2 DIT CM FFTs. As can be observed, the DIT FFT architecture is symmetric with respect to its DIF version in Fig. 5: In the DIF CM FFT the first stages include the largest number of constant

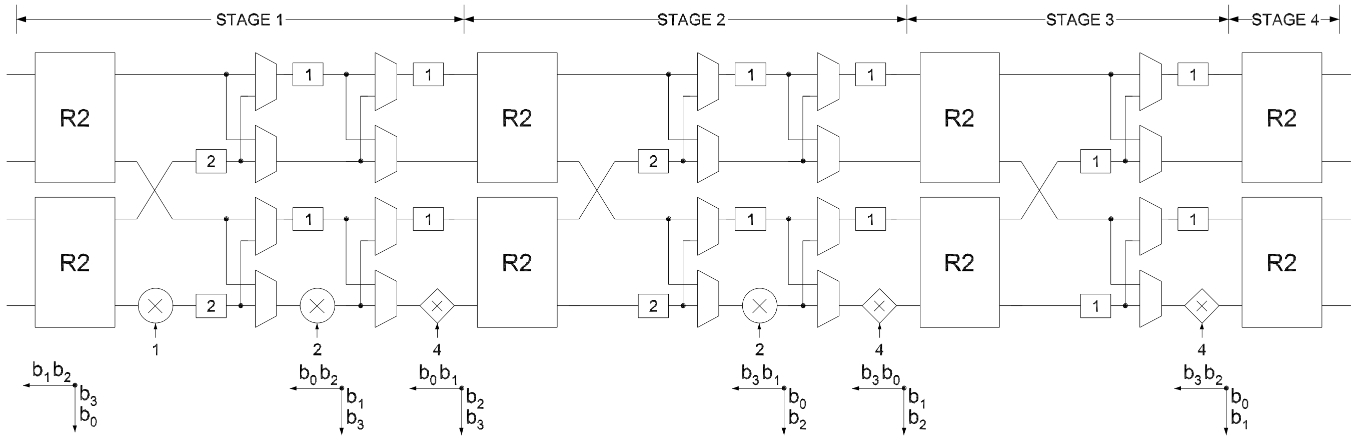

Fig. 5. Proposed 16-point 4-parallel radix-2 DIF CM FFT architecture.

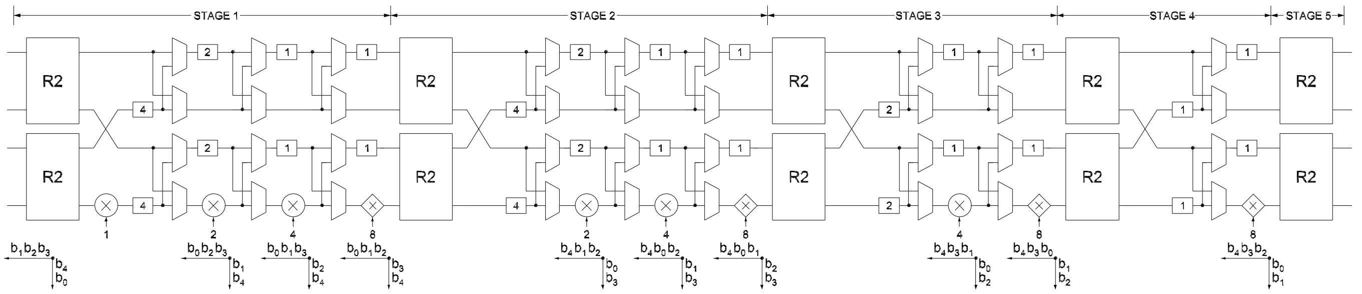

Fig. 6. Proposed 32-point 4-parallel radix-2 DIF CM FFT architecture.

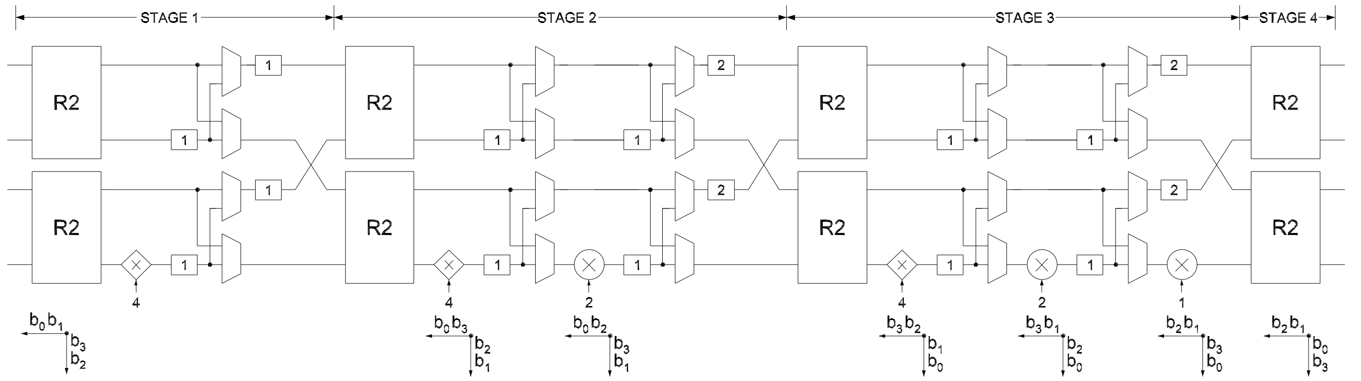

Fig. 7. Proposed 16-point 4-parallel radix-2 DIT CM FFT architecture.

complex multipliers, whereas in the DIT case these multiplications appear towards the last stages. Due to this symmetry, the amount of resources in the DIT case is the same as in the DIF one. The 32-point 4-parallel radix-2 DIT CM FFTs can be obtained in a similar way from its DIF version in Fig. 5 by using symmetry. This also holds for other FFT sizes.

C. 8-Parallel Radix-2 cm FFTs

The 8-parallel radix-2 CM FFT is shown in Fig. 8 for the case of 16 points and DIF. It replicates the 4-parallel version in Fig. 5 while including some simplifications in terms of memory and constant complex multipliers.

The memory of the 8-parallel CM FFT is calculated by taking into account that the shuffling circuits with buffers of length 2^i appear at $n-1-i$ stages and their total memory is $8 \cdot 2^i = 2^{i+3}$ due to the eight parallel paths. The buffer lengths range from $2^0 = 1$ to 2^{n-4} and, therefore, the index ranges from $i = 0$ to $i = n-4$. Furthermore, shuffling circuits of sizes 2^i and 2^{i-1} are connected, from which $8 \cdot 2^{i-1} = 2^{i+2}$ can be removed. For these registers that can be removed, i ranges from $i = 1$ to $i = n-4$. As a result, the total memory

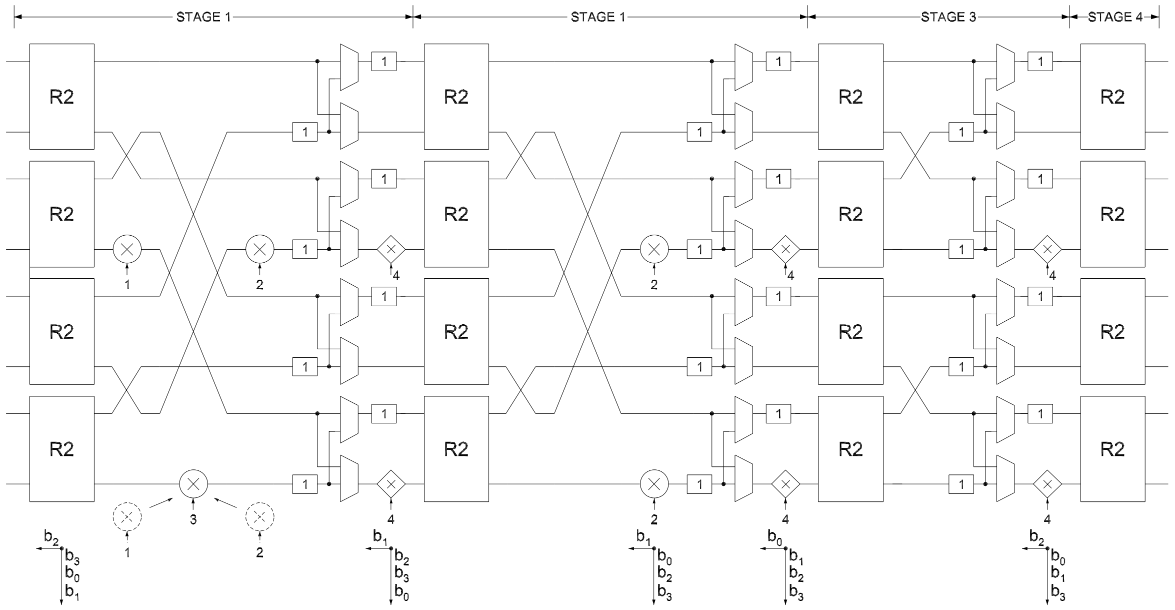

Fig. 8. Proposed 16-point 8-parallel radix-2 DIF CM FFT architecture.

for the 8-parallel CM FFT is calculated as

$$\text{Total Mem.} = \sum_{i=0}^{n-4}(n-1-i)2^{i+3} - \sum_{i=1}^{n-4}(n-1-i)2^{i+2}, \tag{13}$$

which results in

$$\text{Total Mem.} = 2N - 8. \tag{14}$$

As in the 4-parallel case, the number of adders is the product of the number of parallel paths, P, and the number of stages, n. This leads to

$$\text{Total Adders} = P \cdot n = 8n = 8\log_2 N. \tag{15}$$

The number of constant rotators is twice the number in equation (11), with the exception that two of the constant complex rotators can be merged together. This reduces the number by one. This can be observed in the lowest edge of the first stages in Fig. 8. The total number of constant rotators is then

$$\text{Total Constant Rot.} = 2 \cdot \left(\sum_{i=1}^{n-2} i\right) - 1 = (n-1)(n-2) - 1. \tag{16}$$

Finally, the number of multiplexers is twice the number in the 4-parallel case in equation (12) minus 8 for each of the first 2 stages, which are removed in the parallelization, i.e,

$$\text{Total Mux.} = 2 \cdot \left(4\left(\sum_{i=1}^{n-1} i\right) - 4\right) - 8 \cdot 2 = 4n(n-1) - 24. \tag{17}$$

D. Higher Radices

Radix-2 CM FFTs are efficient for small FFT sizes. However, as N increases, the extra cost of additional radix-2 stages is higher and higher. This is due to the fact that in radix-2 the number of constant rotators has an order $\mathcal{O}(n^2)$, which is quadratic with respect to the number of stages. The way to circumvent this problem is to resort to higher radices.

A radix-2^k FFT breaks down an N-point FFT into n/k FFTs of 2^k points. For instance, a 1024-point 4-parallel radix-2^5 FFT architecture consists of two blocks that calculate an FFT of $2^5 = 32$ points connected by a set of general rotators and shuffling circuits. This allows for implementing the 32-point blocks efficiently as radix-2 CM FFTs.

In the selection of the radix, there are two considerations to take into account. First, if we consider a radix-2^k where k is large, then the 2^k-point radix-2 CM FFTs that it is split into would be complex due to the quadratic increase of the constant multipliers. For this reason, large values of k are not advisable. Second, k should not be small, because small values of k break down the FFT into a larger number of radix-2 CM FFTs, which results in a larger number of general rotators in between these blocks. Based on these considerations, the most suitable radices are radix-2^4 and radix-2^5. For them, both the number of the radix-2 CM FFTs blocks and the hardware complexity of these blocks is small.

E. 4-Parallel Radix-2^k CM FFTs

Fig. 9 shows the proposed 1024-point 4-parallel radix-2^5 CM FFT. It consists of $n/k = 2$ 32-point 4-parallel radix-2 CM FFTs. One of them involves stages 1 to 5, and the other one involves stages 6 to 10. Additionally, stage 5 includes a set of 4 general rotators in the 4-parallel paths and stages 2 to 5 include additional shuffling circuits. Apart from the 4 general rotators, the rest of rotators in the FFT are constant

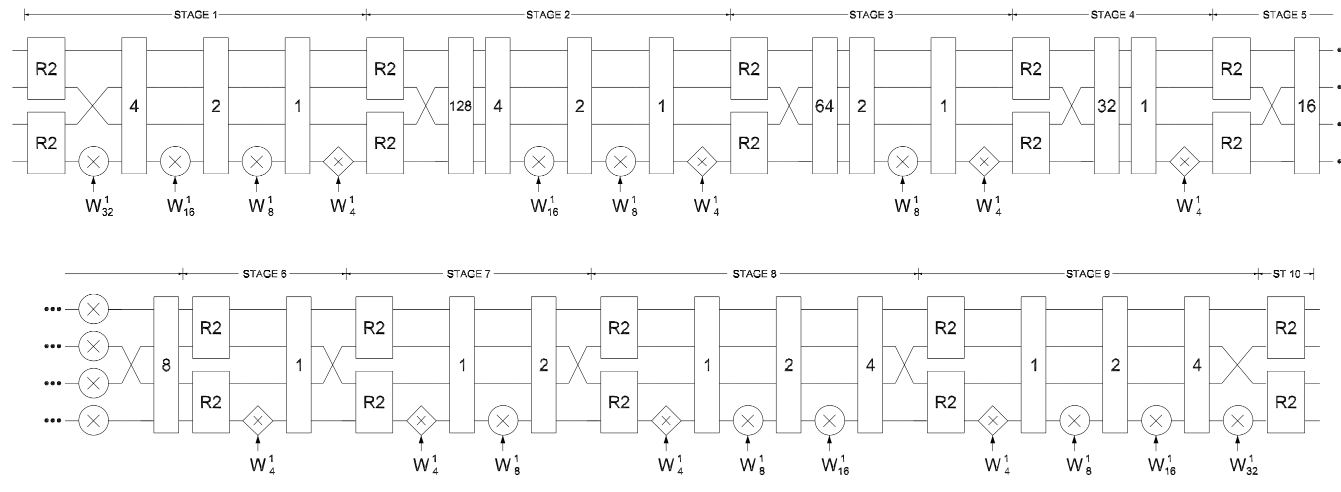

Fig. 9. Proposed 1024-point 4-parallel radix-2^5 CM FFT architecture. For simplicity of the figure, permutation circuits are represented by rectangles with a number that represent the number of registers in each parallel path. As for other CM FFT architectures in this paper, registers that appear in all parallel paths are removed, as explained in [48].

rotators. In the figure, shuffling circuits are represented with a vertical rectangle and a number inside, which allows for a more compact drawing of the architecture.

The number of hardware components can be calculated for a general N and radix-2^k. The total memory is calculated as the sum of the memory of the radix-2 CM FFTs and the memory of the additional shuffling circuits. On the one hand, the memory for the radix-2 CM FFTs is taken from (9), where N is substituted by 2^k, and this memory is multiplied by n/k, which is the number of radix-2^k FFT blocks. On the other hand, the additional shuffling circuits are in charge of accessing all the data in the data flow, so they have to complete the buffer sizes that are not considered in the radix-2^k FFT blocks. These shuffling circuits have a 4 buffers of memory 2^{i-2} from $i = k$ to $i = n - 1$. As a result, the total memory for a 4-parallel radix-2^k CM FFT is

$$\text{Total Mem.} = \frac{n}{k}\left(\frac{3}{2}2^k - 4\right) + \sum_{i=k}^{n-1}\left(4 \cdot 2^{i-2}\right), \tag{18}$$

which results in

$$\text{Total Mem.} = N + 2^k\left(\frac{3}{2}\cdot\frac{n}{k} - 1\right) - 4\cdot\frac{n}{k}. \tag{19}$$

This total memory has an upper bound of

$$\text{Total Mem.} < N + 2\sqrt{N}, \tag{20}$$

where the term $2\sqrt{N}$ is small compared to N. Therefore, the complexity of the total memory has order $\mathcal{O}(N)$, which reduces the complexity with respect to radix-2 CM FFT architectures.

The number of adders for 4-parallel radix-2^k CM FFTs is not modified with respect to radix-2 and fulfills equation (10).

The general rotators appear in between the n/k radix-2^k CM FFT blocks. As there are 4 rotators in parallel in each connection between radix-2 blocks, the total number of rotators in the architecture is

$$\text{Total General Rot.} = 4\left(\frac{n}{k} - 1\right), \tag{21}$$

and the number of constant rotators is equal to the number of 2^k-point radix-2 CM FFT blocks times the number of rotators in each of these blocks according to (11), i.e.,

$$\text{Total Constant Rot.} = \frac{n}{k}\cdot\frac{(k-1)(k-2)}{2}. \tag{22}$$

Finally, the number of multiplexers is the sum of the multiplexers in the radix-2 CM FFT blocks and those in the additional shuffling circuits. Specifically, there are n/k 2^k-point radix-2 blocks and each of them has $2k(k-1) - 4$ multiplexers according to (12), plus $n - k$ addition shuffling circuits with 4 multiplexers each. This leads to

$$\text{Total Mux.} = \frac{n}{k}\cdot(2k(k-1) - 4) + 4(n-k), \tag{23}$$

which result in

$$\text{Total Mux.} = 2nk - 4\cdot\frac{n}{k} + 2n - 4k. \tag{24}$$

F. 8-Parallel Radix-2^k CM FFTs

An 8-parallel radix-2^k CM FFT consists of n/k 2^k-point 8-parallel radix-2 CM FFTs plus additional shuffling circuits and general rotators. As for the 4-parallel case, the total memory is calculated by adding the memory of the radix-2 CM FFTs and the memory of the additional shuffling circuits. In this case, the memory for the radix-2 CM FFT blocks comes from (14) and the memory of the additional shuffling circuits includes 8 buffers of size 2^{i-3} from $i = k$ to $i = n - 1$. This leads to a total memory of

$$\text{Total Mem.} = \frac{n}{k}\left(2\cdot 2^k - 8\right) + \sum_{i=k}^{n-1}\left(8\cdot 2^{i-3}\right), \tag{25}$$

which results in

$$\text{Total Mem.} = N + 2^k\left(2\cdot\frac{n}{k} - 1\right) - 8\cdot\frac{n}{k}. \tag{26}$$

This total memory has an upper bound of

$$\text{Total Mem.} < N + 3\sqrt{N}. \tag{27}$$

TABLE I

CONSTANT MULTIPLICATIONS BY THE FFT ANGLES

Rotation	Angle	Coefficient	R	Adders	WL_E
W_8^1	$-2\pi/8$	$181 - j181$	255.97	8	14.69
W_{16}^1	$-2\pi/16$	$473 - j196$	512.00	7	14.31
W_{32}^1	$-2\pi/32$	$251 - j50$	255.93	8	12.84

As for 4-parallel radix-2^k CM FFTs, the complexity of the total memory has order $\mathcal{O}(N)$, which reduces the complexity with respect to radix-2 CM FFT architectures.

The number of adders for an 8-parallel radix-2^k CM FFT is the same as that in equation (15).

The number of general rotators is one per parallel branch per connection between radix-2 CM FFT blocks, i.e.,

$$\text{Total General Rot.} = 8\left(\frac{n}{k} - 1\right), \tag{28}$$

The number of constant rotators is the result of multiplying the number of radix-2 CM FFT blocks times the number of constant rotators in each of these block according to (16), i.e.,

$$\text{Total Constant Rot.} = \frac{n}{k} \cdot ((k-1)(k-2) - 1). \tag{29}$$

Finally, the number of multiplexers is calculated by adding the multiplexers of all the radix-2 CM FFT blocks according to (17) plus the multiplexers of the additional shuffling circuits, as was done for (23), i.e.,

$$\text{Total Mux.} = \frac{n}{k} \cdot (4k(k-1) - 24) + 8(n-k), \tag{30}$$

which results in

$$\text{Total Mux.} = 4nk - 24 \cdot \frac{n}{k} + -4n - 8k. \tag{31}$$

V. IMPLEMENTATION

As a proof of concept, we have implemented the 4-parallel radix-2^5 1024-point CM FFT in Fig. 9. In this section, we highlight the main details related to the implementation of the architecture.

A. Implementation of the Constant Complex Rotators

The proposed 1024-point 4-parallel radix-2^5 CM FFT calculates constant rotations by W_8^1, W_{16}^1 and W_{32}^1. To obtain a shift-and-add implementation for these rotations we use the combined coefficient scaling and shift-and-add implementation (CCSSI) method [31]. Specifically, we consider the case of single constant rotation (SCR) with unity scaling. The results are shown in Table I. The first two columns show the rotation and the angle, respectively. The third column shows the coefficient $P = C + jS$ that results from the CCSSI method. The fourth column shows the scaling factor (R). The fifth column shows the number of adders that are needed to calculate the rotation. And the last column shows the accuracy of the rotation in terms of effective word length (WL_E) [31].

For all the coefficients, the scaling factor is close to a power of two. Therefore, this scaling is compensated by removing the least significant bits (LSB) of the output of the rotator, without any additional hardware cost. Note that the scaling of the rotators in the CM FFT needs to be compensated to meet the scaling of $R = 1$ in other parallel paths of the architecture. This is why a scaling that is close to a power of two and, therefore, can easily be compensated, has been chosen.

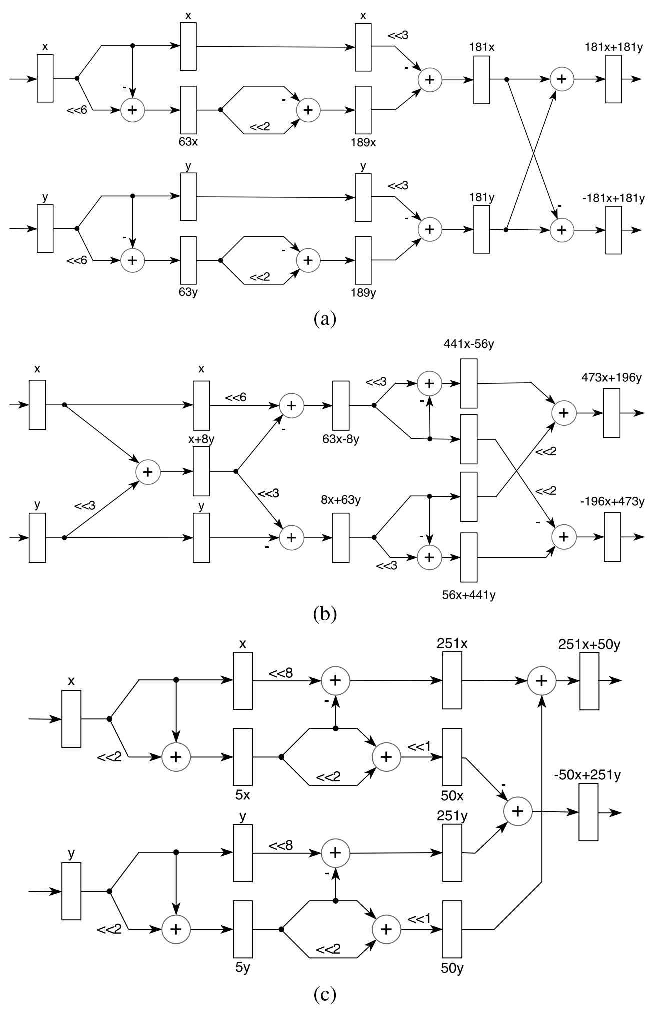

Fig. 10. Constant rotator circuits for the CM FFT. (a) Constant rotation by the coefficient $181 - j181$ for the angle W_8^1. (b) Constant rotation by the coefficient $473 - j196$ for the angle W_{16}^1. (c) Constant rotation by the coefficient $251 - j50$ for the angle W_{32}^1.

Fig. 10 shows the circuits used to calculate the constant rotations by the angles in Table I. The operation that is calculated is $(x + jy) \cdot (C + jS)$ where $x + jy$ is the input signal and $C + jS$ is the rotation coefficient according to Table I.

The rotators consist of adders and registers. The registers are used for pipelining, which allows for reducing the critical path of the architecture to only one adder and, therefore, increase the clock frequency of the FFT architecture. To compensate for the pipelining of the rotators, extra registers are added to the other parallel paths of the FFT architecture in order to keep the same delay in all the paths.

The shift operations in the circuits are hard-wired, so they do not have any hardware cost. It can also be observed that the constant rotators do not include any rotation memory. The rotation is carried out by fixed shift-and-add operations, so there is no need to store the coefficient. Likewise, previous

W_8 and W_{16} rotators require multiplexers in order to select among the rotation angles. Conversely, the proposed constant rotators do not need any multiplexer.

B. Merging Butterflies and Trivial Rotators

In the implemented architecture shown in Fig. 9, trivial rotators, i.e., diamond-shaped ones, appear either before (stages 2, 3, 4 and 5) or after the butterflies (stages 6, 7, 8 and 9). Furthermore, they are constant rotators, meaning that all the data passing through the rotator are rotated by W_4^1. In the implementation of the architecture, we have taken advantage of these facts in order to simplify the architecture.

A regular butterfly adds two complex numbers according to

$$\begin{aligned} X &= x + y = (x_r + y_r) + j(x_i + y_i), \\ Y &= x - y = (x_r - y_r) + j(x_i - y_i), \end{aligned} \tag{32}$$

where $x = x_r + jx_i$ is the upper input, $y = y_r + jy_i$ is the lower input, $X = X_r + jX_i$ is the upper output and $Y = Y_r + jY_i$ is the lower output.

When a trivial rotator precedes the butterfly, the lower input y is multiplied by $-j$, which results in

$$\begin{aligned} X &= x + (-j)y = (x_r + y_i) + j(x_i - y_r), \\ Y &= x - (-j)y = (x_r - y_i) + j(x_i + y_r). \end{aligned} \tag{33}$$

As can be observed, the trivial rotator is embedded in the butterfly by simply changing some inputs and transforming additions into subtractions or vice versa. As a result, the number and complexity of the operation is the same as in a regular butterfly.

When the trivial rotator appears after the butterfly, the lower output is multiplied by $-j$, i.e.,

$$\begin{aligned} X &= x + y = (x_r + y_r) + j(x_i + y_i), \\ Y &= -j(x - y) = (x_i - y_i) + j(y_r - x_r). \end{aligned} \tag{34}$$

In this case, merging the trivial rotator with the butterfly does not increase the complexity of the butterfly either.

C. Implementation of the Shuffling Circuits

The implementation of the shuffling circuits includes several improvements. First, they are pipelined by adding a register previous to the shuffling circuit in order to avoid long critical paths through the multiplexers. This allows for increasing the clock frequency of the FFT architecture.

Second, the delays of the shuffling circuits can be implemented using BRAMs or look-up tables (LUTs). We have evaluated both alternatives for all the shuffling circuits in the architecture and observed that the implementation using LUTs obtains the best results, both in area and clock frequency. As a result, the proposed architecture does not use BRAMs.

Third, the control of the shuffling circuits has been simplified. The control signals for the shuffling circuits can be obtained from the bits of the control counter. They are periodical with a duty cycle of 50% and a period that is twice the length of the permutation circuit. For instance, a shuffling circuit with buffer length 2 is controlled by a signal with 2 consecutive 0s, and then 2 consecutive 1s. This signal corresponds to the bit c_2 of the control counter $c_7\, c_6 \ldots c_2\, c_1\, c_0$. However, the control signal has to be delayed a number of clock cycles that makes it start when the first four data arrive in parallel to the shuffling circuit, i.e., the phases of the bit of the counter and the control signal may be different. This makes it necessary to add a shift register for each bit of the counter to adjust the phase of the control signal.

TABLE II
CALCULATION OF THE PHASE OF THE SIGNAL CONTROLLING EACH SHUFFLING CIRCUIT AND THE LENGTH OF THE SHIFT REGISTERS

Stage	Shuffling Circuit Size (Delays)	Latency (cycles)	Phase (cycles)	Shift Register Size (cycles)
1	4	6	6	6
1	2	17	1	3
1	1	26	0	1
2	128	30	30	30
2	4	160	0	6
2	2	171	3	3
2	1	180	0	1
3	2	250	2	3
3	1	259	1	1
3	64	184	56	56
4	32	263	7	7
4	1	297	1	1
5	16	301	13	13
5	8	325	5	5
6	1	336	0	1
7	1	340	0	1
7	2	348	0	3
8	1	353	1	1
8	2	361	1	3
8	4	370	2	6
9	1	377	1	1
9	2	385	1	3
9	4	394	2	6

Table II shows the calculation of the phase of the signal controlling each shuffling circuit and the length of the shift registers, according to the previous explanation. Each row of the table corresponds to one of the shuffling circuits of the architecture according to Fig. 9. The first column indicates the stage in which the shuffling circuit is placed. The second column indicates the size of the buffers of the shuffling circuit. The third column is the accumulated latency of the architecture at the input of the shuffling circuit. The fourth column is the phase of the control signal for the shuffling circuit, which is equal to the accumulated latency in column 3 modulo the period of the shuffling circuit, where the period is twice the size of the buffers of the shuffling circuit in column 2. Finally, there is one shift register for each group of shuffling circuits with the same size. The size of the shift register is then the maximum among the phases of the shuffling circuits with the same buffer size.

D. Implementation of the General Rotators

The proposed architecture includes a general rotator at stage 5 in each of the four parallel paths. These rotators are implemented by using complex multipliers and read-only

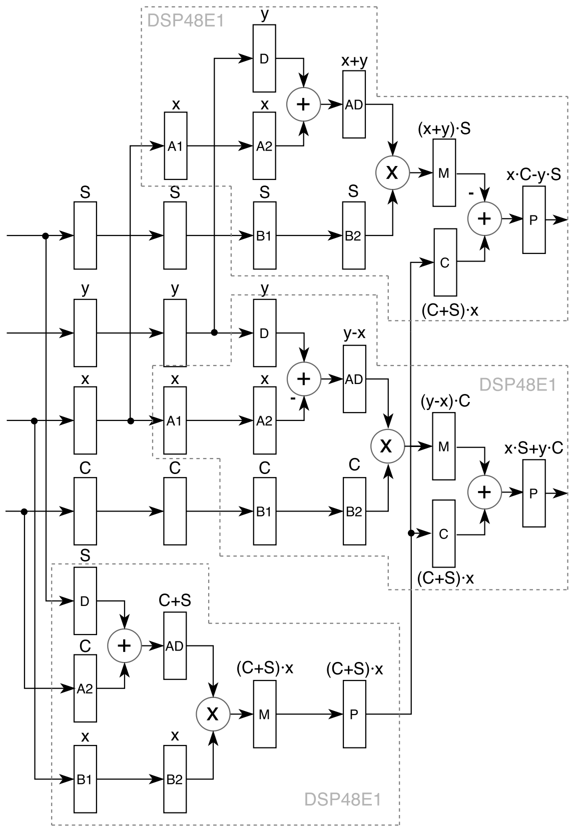

Fig. 11. General rotator using 3 DSP slices.

memories (ROM) to store the rotation coefficients. Each of the four memories has size 256 and the 8-bit counter that controls the permutation circuits is also used as the address of the ROMs. As for the shuffling circuits, the counter should be delayed according to the latency at this place of the circuit. However, we have implemented a more efficient solution that consists in storing the coefficients in the ROM with an offset according to the latency. As the latency at this point of the circuit is 318 and the size of the memories is 256, coefficients are stored in the memories with an offset of 62 addresses. For instance, the coefficient for $\phi = 0$ is stored in address 62 of the memory. This removes the need for delaying the control counter and, therefore, saves hardware resources.

The general rotators are implemented using the Complex Multiplier IP generator from Xilinx [49]. The circuit is shown in Fig. 11. This solution uses 3 real multipliers in 3 DSP slices (DSP48E1) to calculate a complex multiplication according to

$$\begin{aligned} X &= x(C+S) - (x+y)S \\ Y &= x(C+S) + (y-x)C, \end{aligned} \tag{35}$$

where $x + jy$ is the input, $C + jS$ is the rotation coefficient and $X + jY$ is the output. This reduces the complexity with respect to the common alternative of using 4 real multipliers to calculate a complex multiplication.

The latency of the proposed solution is 6 clock cycles. In Fig. 11, it can be observed that the signals pass through 6 registers. Some of these registers are internal to the DSPs, which are labeled as A1, A2, D, AD, B1, B2, M, C and P. The other registers, which do not have any label, are external to the DSP. The pipelining of the general rotators leads to calculating only one mathematical operation per clock cycle, which allows for a high clock frequency.

VI. Comparison and Experimental Results

Table III compares the proposed CM FFT to previous approaches for the computation of an N-point FFT, for the cases of 4-parallel and 8-parallel architectures. The first column shows the type of FFT architecture and the radix. Columns two to four show the number of rotators, separated into general rotators, rotators by W_8 & W_{16}, and constant rotators. The value n in the table is $n = \log_2(N)$. The fifth column shows the number of complex adders in butterflies. The sixth and seventh columns show the sizes of the complex data and the rotation memories. Finally, columns eight and nine show the performance in terms of latency and throughput, respectively.

In Table III, it can be observed that the proposed radix-2 CM FFT is the only architecture that only uses constant rotators, and does not use any general, W_8 or W_{16} rotators. Furthermore, it does not use any rotation memory.

The radix-2^4 and radix-2^5 CM FFTs include constant rotators and a small number of general rotators. By comparing the 4-parallel radix-2^4 CM FFT to previous radix-2^4 MDC architectures [15], it can be observed that the number of general rotators is the same, whereas the W_8 and W_{16} rotators in [15] are transformed into constant rotators in the proposed approach, which reduces the complexity.

Furthermore, the proposed radix-2^5 CM FFT has smaller number of general rotators compared to previous parallel pipelined architectures.

Table IV compares the proposed architecture to previous 4-parallel 1024-point FFTs in the literature. The table includes the following parameters: FPGA type, FFT size (N), parallelization (P), radix, word length (WL), number of slices, block random access memories (BRAM), digital signal processing (DSP) slices, clock frequency, throughput and power consumption. It can be observed that $N = 1024$, $P = 4$, and $WL = 16$ for all the designs in the table, which makes them comparable. Furthermore, most of them provide experimental results for a Virtex-6 FPGA.

For the proposed approach post implementation results have been obtained for a Virtex-6 XC6VSX475T FPGA (V6 in the table) and for a Virtex-7 XC7VX330T FPGA (V7 in the table). By comparing the results for V6 to previous approaches, it can be observed that the proposed approach achieves not only the highest clock frequency but also the smallest number of BRAM and DSP slices. The highest clock frequency is due to the deep pipelining in the proposed design, which is enabled by the use of constant multipliers. Due to the simplicity of these multipliers, they are easily pipelined. The small number of BRAM is due to using distributed logic for the shuffling circuits, as explained in Section V-C. This is also why the number of slices is higher in the proposed approach. Finally, the smaller number of DSP slices is due to the use of 3 DSP slices per general rotator, as explained in Section V-D.

Regarding the implementation on the V7 FPGA, it uses a small amount of hardware resources and obtains a clock

TABLE III

COMPARISON OF THE PROPOSED CM FFT ARCHITECTURES TO PREVIOUS APPROACHES FOR THE COMPUTATION OF AN N-POINT FFT

PIPELINED	AREA						PERFORMANCE	
ARCHITECTURE	Rotators			Complex	Complex Mem.		Latency	Th.
Type, Radix	General	W_8 & W_{16}	Constant	Adders	Data	Rots.	(cycles)	(data/cyc.)
4-PARALLEL ARCHITECTURES								
MDC, radix-4 [10]	$3\lceil n/2\rceil-3$	0	0	$4n$	$8N/3$	N	$N/3$	4
MDF, radix-2^4 [4]	$4\lceil n/4\rceil-4$	$4\lfloor n/4\rfloor$	0	$8n$	N	N	$N/4$	4
MDF, radix-2^4 [14]	$4\lceil n/4\rceil-4$	$4\lfloor n/4\rfloor$	0	$8n$	N	N	$N/4$	4
MDF, radix-2^4 [3]	$4\lceil (n-2)/4\rceil-1$	$4\lfloor (n-2)/4\rfloor$	0	$4n$	N	N	$N/2$	4
MDC, radix-2 [1]	$2n-4$	0	0	$4n+4$	N	N	$3N/4$	4
MDC, radix-2^2 [15]	$3\lceil n/2\rceil-3$	0	0	$4n$	N	N	$N/4$	4
MDC, radix-2^3 [15]	$4\lceil n/3\rceil-4$	$2\lfloor n/3\rfloor$	0	$4n$	N	N	$N/4$	4
MDC, radix-2^4 [15]	$4\lceil n/4\rceil-4$	$3\lfloor n/4\rfloor$	0	$4n$	N	N	$N/4$	4
Proposed, CM, radix-2	0	0	$(n-1)(n-2)/2$	$4n$	$3N/2-4$	0	$3N/8$	4
Proposed, CM, radix-2^4	$4\lceil n/4\rceil-4$	0	$3\lfloor n/4\rfloor$	$4n$	$< N+2\sqrt{N}$	N	$\approx N/4$	4
Proposed, CM, radix-2^5	$4\lceil n/5\rceil-4$	0	$6\lfloor n/5\rfloor$	$4n$	$< N+2\sqrt{N}$	N	$\approx N/4$	4
8-PARALLEL ARCHITECTURES								
MDC, radix-8 [10]	$7\lceil n/3\rceil-7$	$2\lfloor n/3\rfloor$	0	$8n$	$16N/7$	N	$2N/7$	8
MDF, radix-2^4 [11]	$8\lceil n/4\rceil-8$	$8\lfloor n/4\rfloor$	0	$16n$	N	N	$N/8$	8
MDF, radix-2^4 [3]	$8\lceil (n-3)/4\rceil-1$	$8\lfloor (n-3)/4\rfloor$	0	$8n$	N	N	$N/4$	8
MDC, radix-2^2 [15]	$6\lceil n/2\rceil-6$	0	0	$8n$	N	N	$N/8$	8
MDC, radix-2^3 [15]	$7\lceil n/3\rceil-7$	$2\lfloor n/3\rfloor$	0	$8n$	N	N	$N/8$	8
MDC, radix-2^4 [15]	$8\lceil n/4\rceil-8$	$6\lfloor n/4\rfloor$	0	$8n$	N	N	$N/8$	8
Proposed, CM, radix-2	0	0	$(n-1)(n-2)-1$	$8n$	$2N-8$	0	$N/4$	8
Proposed, CM, radix-2^4	$8\lceil n/4\rceil-8$	0	$5\lfloor n/4\rfloor$	$8n$	$< N+3\sqrt{N}$	N	$\approx N/8$	8
Proposed, CM, radix-2^5	$8\lceil n/5\rceil-8$	0	$11\lfloor n/5\rfloor$	$8n$	$< N+3\sqrt{N}$	N	$\approx N/8$	8

TABLE IV

COMPARISON OF 4-PARALLEL 1024-POINT FFTS ON FPGAS

FFT	Ours (2014)	Glittas (2016)	Wang (2016)	Spiral (2017)	Ours (2018)	Proposed	
Parametes	[15], [16]	[1]	[3]	[21], [23]	[17]	V6	V7
FPGA	V6	V5	V6	V6	V6	V6	V7
N	1024	1024	1024	1024	1024	1024	1024
P	4	4	4	4	4	4	4
Radix	2^2	2	2^2	2	2^5	2^5	2^5
WL	16	16	16	16	16	16	16
Slices	1341	1029†	1090†	1443	1420	2576	2631
BRAM	12	0	10	44	12	0	0
DSP slices	48	72	45	64	16	12	12
Clk (MHz)	227	380	305	289	253	475	680
Th (MS/s)	910	1520	1220	1157	1012	1900	2720
P(W)	-	-	-	-	-	-	1.68

†: Obtained by dividing the maximum number among look-up tables (LUTs) and flip-flops (FF) by 4. These values have been corrected with respect to the numbers reported in [17].

frequency of 680 MHz and a throughput of 2.72 GS/s. To the best of our knowledge, this is the highest throughput reported for a 4-parallel pipelined FFT on an FPGA so far.

The power consumption of the proposed architecture on the V7 is 1.68 W, which is divided into 0.328 W for the clocks, 0.705 W for the signals, 0.642 W for the logic, and 0.004 W for the DSP slices. These numbers have been obtained from the post-implementation power report in Vivado. Previous works in Table IV do not report power metrics that could be used for a comparison. Indeed, only the most recent FFT architectures on FPGAs have started to report power consumption [6], [45].

In order to provide the results graphically, Fig. 12 shows the throughput versus FPGA utilization of the 1024-point pipelined FFT hardware architectures in the literature, including all the designs in Table IV. In the figure, $(1P)$, $(2P)$, $(4P)$ and $(8P)$ indicate that the architectures are serial, 2-parallel, 4-parallel and 8-parallel respectively.

The throughput in Fig. 12 is measured in MS/s and the FPGA utilization is calculated according to the definition

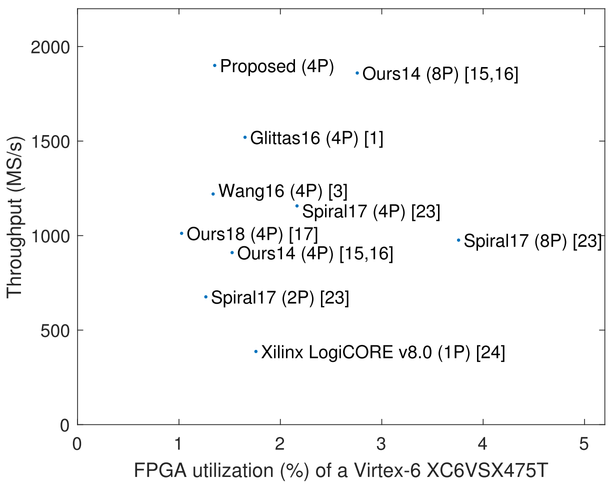

Fig. 12. Throughput versus FPGA utilization of 1024-point FFT hardware architectures on a Virtex-6 FPGA.

in [16]. In the figure, it can be observed that all the designs are efficient in terms of FPGA utilization, as they require less than 4% of the resources of the FPGA.

The most efficient designs appear towards the upper left corner, which represents the highest throughput and the lowest FPGA utilization. According to this, the proposed architecture achieves the highest throughput and one of the lowest values of FPGA utilization. Compared to [3], the FPGA utilization of the proposed approach is similar, whereas the throughput is 36% higher in the proposed approach. Compared to [1], which is the 4-parallel FFT with highest throughput among previous works, the proposed approach increases the throughput by 20% and requires less FPGA resources. Compared to [17], the proposed approach almost doubles the throughput, but has higher FPGA utilization. This higher FPGA utilization is mainly due to the deep pipelining in the proposed FFT, which is used to increase the clock frequency. Thus, the architecture in [17] and the proposed architecture present a trade-off between throughput and FPGA utilization: If the main goal is to increase the throughput, the proposed architecture is the best alternative, whereas the architecture in [17] provides lower utilization with lower throughput. Finally, it is interesting to compare the proposed design to the 8-parallel architectures in [15], [16]. The throughput of both designs is similar. However, the proposed architecture is 4-parallel, whereas the architecture in [15], [16] is 8-parallel, and the FPGA utilization of the proposed approach is approximately half of the FPGA utilization in [15], [16]. Therefore, in 6 years from 2014, we have been able to derive FFT architectures that half the amount of resources while achieving a similar, or even higher, throughput.

VII. Discussion

The proposed CM FFT architectures surge from the idea of using constant rotators to calculate the FFT instead of the typical W_8 and W_{16} and general rotators. To make this possible, the proposed approach does two things. First, it decomposes the rotations at the FFT stages into a sum of constant rotators. Second, it allocate data that must be rotated by the same constant rotation at the same path in the architecture. To achieve this, additional shuffling circuits are needed in order to guarantee that each constant rotator receives the corresponding data.

The use of constant multipliers at all or almost all the stages of the architecture reduces the complexity of the rotators. Contrary to W_8 and W_{16} rotators, constant rotators only rotate by a single constant angle. Therefore, they do not require multiplexers to select among different angles. However, the use of constant rotators does not necessarily reduce the area of the entire FFT because, despite their lower complexity, a larger number of shuffling circuits is used in the architecture.

As a result, the advantage of the proposed architectures is not observed in the area of the architecture, but in the throughput. The reason for this is that the smaller complexity of the rotators allows for deep pipelining that reduces significantly the critical path of the architecture, which leads to an increase of the clock frequency and the throughput.

Experimental results for a 1024-point 4-parallel radix-2^5 CM FFT show that this architecture achieves the highest throughput among 4-parallel FFT hardware architectures, which is 20% higher than the highest throughput among previous works. Furthermore, the FPGA utilization of this design is very competitive and only the 4-parallel architecture in [17] presents a noticeable reduction in area with respect to the proposed approach.

VIII. Conclusions

In this paper, we have presented the constant multiplier FFT. This is a new concept that is based on using constant multipliers to calculate the FFT rotations. When using radix-2, it is possible to calculate the entire FFT by using constant multipliers. However, as the increase in the number of complex multipliers is quadratic with the number of stages, radix-2 is only feasible for small FFTs. For large FFTs, better alternatives are radix-2^4 or radix-2^5.

The use of constant multipliers allows for deep pipelining that result in high throughput. Experimental results for a 1024-point 4-parallel radix-2^5 CM FFT provide a throughput that is 20% higher than the highest throughput among previous 1024-point 4-parallel pipelined FFT architectures. As a consequence, the proposed 1024-point radix-2^5 CM FFT achieves the highest throughput reported for a 4-parallel pipelined FFT on an FPGA so far.

References

[1] A. X. Glittas, M. Sellathurai, and G. Lakshminarayanan, "A normal I/O order Radix-2 FFT architecture to process twin data streams for MIMO," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 6, pp. 2402–2406, Jun. 2016.

[2] J. Lee, H. Lee, S. Cho, and S.-S. Choi, "A high-speed, low-complexity radix-2^4 FFT processor for MB-OFDM UWB systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2006, pp. 210–213.

[3] J. Wang, C. Xiong, K. Zhang, and J. Wei, "A mixed-decimation MDF architecture for radix-2^k parallel FFT," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 1, pp. 67–78, Jan. 2016.

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

[5] 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.*, May 2007, pp. 2594–2597.

[6] H. Kanders, T. Mellqvist, M. Garrido, K. Palmkvist, and O. Gustafsson, "A 1 million-point FFT on a single FPGA," *IEEE Trans. Circuits Syst. I. Reg. Papers*, vol. 66, no. 10, pp. 3863–3873, Oct. 2019.

[7] T. Kamazaki *et al.*, "Digital spectro-correlator system for the Atacama compact array of the Atacama large millimeter/submillimeter array," *Publications Astronomical Soc. Jpn.*, vol. 64, no. 2, p. 29, Apr. 2012.

[8] L. Li and A. M. Wyrwicz, "Parallel 2D FFT implementation on FPGA suitable for real-time MR image processing," *Rev. Sci. Instrum.*, vol. 89, no. 9, pp. 1–9, Sep. 2018.

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

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

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

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

[13] D. Wold, "Pipeline and parallel-pipeline FFT processors for VLSI implementations," *IEEE Trans. Comput.*, vol. C-33, no. 5, pp. 414–426, May 1984.

[14] S.-I. Cho, K.-M. Kang, and S.-S. Choi, "Implemention of 128-point fast Fourier transform processor for UWB systems," in *Proc. Int. Wireless Commun. Mobile Comput. Conf.*, Aug. 2008, pp. 210–213.

[15] M. Garrido, J. Grajal, M. A. Sanchez, and O. Gustafsson, "Pipelined radix-2^k feedforward FFT architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 1, pp. 23–32, Jan. 2013.

[16] M. Garrido, M. Acevedo, A. Ehliar, and O. Gustafsson, "Challenging the limits of FFT performance on FPGAs (Invited paper)," in *Proc. Int. Symp. Integr. Circuits (ISIC)*, Dec. 2014, pp. 172–175.

[17] M. Garrido, S.-J. Huang, and S.-G. Chen, "Feedforward FFT hardware architectures based on rotator allocation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 2, pp. 581–592, Feb. 2018.

[18] T. Ahmed, M. Garrido, and O. Gustafsson, "A 512-point 8-parallel pipelined feedforward FFT for WPAN," in *Proc. Conf. Rec. 45th Asilomar Conf. Signals, Syst. Comput. (ASILOMAR)*, Nov. 2011, pp. 981–984.

[19] M. Garrido, S.-J. Huang, S.-G. Chen, and O. Gustafsson, "The serial commutator FFT," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 63, no. 10, pp. 974–978, Oct. 2016.

[20] 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. (ISCAS)*, Oct. 2020, pp. 1–5.

[21] P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Formal datapath representation and manipulation for implementing DSP transforms," in *Proc. 45th Annu. Conf. Design Autom. (DAC)*, 2008, pp. 385–390.

[22] M. Garrido, F. Qureshi, J. Takala, and O. Gustafsson, "Hardware architectures for the fast Fourier transform," in *Handbook of Signal Processing Systems*, 3rd ed., S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds. Cham, Switzerland: Springer, 2019.

[23] *Spiral DFT/FFT IP Core Generator*. Accessed: May 2017. [Online]. Available: http://spiral.net/hard-ware/dftgen.html

[24] *Xilinx—FFT Logicore 8.0*. Accessed: Jul. 2012. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ds808_xfft.pdf

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

[26] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.

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

[28] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proc. ACM/SIGDA 6th Int. Symp. Field Program. Gate Arrays (FPGA)*, Feb. 1998, pp. 191–200.

[29] 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, Reg. Papers*, vol. 56, no. 9, pp. 1893–1907, Sep. 2009.

[30] C.-Y. Chen and C.-Y. Lin, "High-resolution architecture for CORDIC algorithm realization," in *Proc. Int. Conf. Commun., Circuits Syst.*, vol. 1, Jun. 2006, pp. 579–582.

[31] 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, Reg. Papers*, vol. 61, no. 7, pp. 2002–2012, Jul. 2014.

[32] 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*, vol. 47, no. 3, pp. 757–768, Mar. 2012.

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

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

[35] F. Qureshi and O. Gustafsson, "Low-complexity constant multiplication based on trigonometric identities with applications to FFTs," *IEICE Trans. Fundamentals*, vol. E94-A, no. 11, pp. 324–326, Nov. 2011.

[36] Y.-E. Kim, K.-J. Cho, and J.-G. Chung, "Low power small area modified booth multiplier design for predetermined coefficients," *IEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vol. E90-A, no. 3, pp. 694–697, Mar. 2007.

[37] V. Karkala, J. Wanstrath, T. Lacour, and S. P. Khatri, "Efficient arithmetic sum-of-product (SOP) based multiple constant multiplication (MCM) for FFT," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2010, pp. 735–738.

[38] P. Kallstrom, M. Garrido, and O. Gustafsson, "Low-complexity rotators for the FFT using base-3 signed stages," in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, Dec. 2012, pp. 519–522.

[39] Y. H. Hu and S. Naganathan, "An angle recoding method for CORDIC algorithm implementation," *IEEE Trans. Comput.*, vol. 42, no. 1, pp. 99–102, Jan. 1993.

[40] C.-S. Wu and A.-Y. Wu, "Modified vector rotational CORDIC (MVR-CORDIC) algorithm and architecture," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 6, pp. 548–561, Jun. 2001.

[41] P. K. Meher and S. Y. Park, "CORDIC designs for fixed angle of rotation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 2, pp. 217–228, Feb. 2013.

[42] C.-S. Wu, A.-Y. Wu, and C.-H. Lin, "A high-performance/low-latency vector rotational CORDIC architecture based on extended elementary angle set and trellis-based searching schemes," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 50, no. 9, pp. 589–601, Sep. 2003.

[43] 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, Reg. Papers*, vol. 52, no. 11, pp. 2385–2396, Nov. 2005.

[44] S. Yoon Park and Y. Jun Yu, "Fixed-point analysis and parameter selections of MSR-CORDIC with applications to FFT designs," *IEEE Trans. Signal Process.*, vol. 60, no. 12, pp. 6245–6256, Dec. 2012.

[45] M. Garrido, K. Moller, and M. Kumm, "World's fastest FFT architectures: Breaking the barrier of 100 GS/s," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 4, pp. 1507–1516, Apr. 2019.

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

[47] M. Garrido, "A new representation of FFT algorithms using triangular matrices," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 10, pp. 1737–1745, Oct. 2016.

[48] M. Garrido, J. Grajal, and O. Gustafsson, "Optimum circuits for bit-dimension permutations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 5, pp. 1148–1160, May 2019.

[49] *Xilinx—Complex Multiplier V6.0*. Accessed: Nov. 2015. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/cmpy/v6_0/pg104-cmpy.pdf

Mario Garrido (Senior Member, IEEE) received the M.Sc. degree in electrical engineering and the Ph.D. degree from the Technical University of Madrid (UPM), Madrid, Spain, in 2004 and 2009, respectively.

In 2010, he moved to Sweden to work as a Post-Doctoral Researcher with the Department of Electrical Engineering, Linköping University, where he was an Associate Professor from 2012 to 2019. In 2019, he moved back to UPM, where he holds a Ramón y Cajal Research Fellowship. His research focuses on optimized hardware design for signal processing applications. This includes the design of hardware architectures for the calculation of transforms, such as the fast Fourier transform (FFT), circuits for data management, the CORDIC algorithm, and circuits to calculate statistical and mathematical operations. His research covers high-performance circuits for real-time computation, as well as designs for small area and low power consumption.

Pedro Malagón received the Ph.D. degree in telecommunication engineering from the Technical University of Madrid (UPM) in 2015. He has been an Associate Professor with UPM since 2019. His main research interests are reconfigurable hardware for 5G networks and security, and energy-aware computing.