This is a preprint document. To cite the work or retrieve its final Open Access version, follow the DOI below:

Digital Object Identifier 10.1109/ACCESS.2024.3414928

Optimized 4-Parallel 1024-Point MSC FFT

ZEYNEP KAYA[1], and MARIO GARRIDO[2], (Senior Member, IEEE)

[1]Department of Electricity and Energy, Osmaneli Vocational School, Bilecik Seyh Edebali University, 11500 Bilecik, Türkiye (e-mail: zeynep.kaya@bilecik.edu.tr)

[2]Department of Electronic Engineering, ETSI de Telecomunicación, Universidad Politécnica de Madrid, 28040 Madrid, Spain (e-mail: mario.garrido@upm.es)

Corresponding author: Zeynep Kaya (e-mail: zeynep.kaya@bilecik.edu.tr).

This work was supported in part by Comunidad de Madrid through the call "Ayudas de Estímulo a la Investigación de Jóvenes Doctores de la Universidad Politécnica de Madrid" under Project APOYO-JOVENES-21-TL23SB-116-I4FOMC; and in part by MCIN/AEI/10.13039/501100011033 and "ESF Investing in your future" under Grant RYC2018-025384-I.

ABSTRACT This paper presents a 4-parallel 1024-point multi-path delay commutator (MSC) fast Fourier transform (FFT) architecture. The aim of this work is to provide multiple optimizations of this type of FFT at the architectural level. This results in a highly optimized FFT architecture that uses significantly fewer resources than previous ones. One advantage of the proposed approach is that it uses two identical processing modules, which simplifies the development of architecture. Between the modules, instead of permutation circuits, we use matrix transposition with memories, which results in a more compact and hardware-efficient solution. Additionally, a tailor-made design of the rotations and the application of shift-and-add techniques lead to a reduction in their number and complexity. Experimental results show that the proposed architecture requires significantly fewer hardware components than previous comparable parallel pipeline FFT architectures.

INDEX TERMS MSC, FFT, parallel pipeline, matrix transposition, shift-and-add.

I. INTRODUCTION

NOWADAYS, fast Fourier transform (FFT) architectures are key components in multiple signal processing applications. This includes areas as relevant as 5G and 6G communications [1], [2], radio atronomy [3], [4], and satellite systems [5], [6].

FFT architectures are classified into memory-based [7]–[10] and pipelined [4], [11]–[34]. Memory-based FFTs carry out the calculations iteratively and intermediate results are stored in the same memory or bank of memories. By contrast, pipelined FFTs compute the algorithm by means of a series of stages connected in a pipeline, which allows for continuous flow computations.

Pipelined FFTs consider a continuous flow of data arriving in series or in parallel. Traditionally, serial pipelined FFT architectures included single-path delay feedback (SDF) [4], [11], [12] and single-path delay commutator (SDC) [13], [14] FFTs. Analogously, parallel pipelined FFT architectures considered the cases of multi-path delay feedback (MDF) [15]–[17] and multi-path delay commutator (MDC) [18]–[25].

In this context, the serial commutator (SC) FFT [26] appeared as an alternative to the traditional architectures, being the first serial pipelined FFT architecture that achieved full utilization of butterflies. Additionally, the SC FFT was also the first FFT architecture that used circuits to permute serial data [35]. This allows for placing data that must be rotated every other clock cycle, which provides two clock cycles to calculate each rotation and translates into a reduction of the complexity of the rotators. As the rotation can be carried out in two clock cycles instead of one, only half of the hardware resources are necessary for calculating them. As a consequence, the SC FFT represented a key step in the development of FFT architectures.

After the original SC FFT, numerous works have continued this research line by proposing alternatives that improve the original design. Among them, most of the papers in the literature have considered real-valued serial commutator (RSC) FFT architectures [27]–[31]. These RSC approaches provided new architectural modifications [27]–[30] or focused on evaluating the accuracy in terms of SQNR calculation [31]. Furthermore, a recent paper presented an interesting approach to reduce the number of multiplexers in SC FFTs [32].

Apart from developing the SC FFT itself, a natural evolution of this research field has been to use the principles in the SC FFT to calculate FFTs on parallel data. This led to the multi-path serial commutator (MSC) FFT, whose first version was presented in [33]. This work only covered the particular

case of a 128-point MSC FFT. Thus, a more completed study on MSC FFT architectures was presented later in [34]. This work considered MSC architectures for any radix-2^k and presented efficient modules that can be combined to derive the architectures for any FFT size and radix.

In this work, we present an evolution of the MSC FFT in which we have applied multiple optimization techniques to the architecture with the goal of developing a new architecture that requires significantly fewer resources to calculate the FFT than previous approaches. Although the proposed techniques are general and can be applied to FFT architectures with different sizes, radices, and parallelization, we consider for the study the case of a 4-parallel radix-2^5 1024-point MSC FFT. One advantage of the proposed FFT is that it uses the same radix-2^5 module twice in the architecture. This makes the design simple, as only a single module must be developed. Furthermore, the proposed architecture is designed so that most of the permutations are placed between the radix-2^5 modules. Instead of using registers and multiplexers for these permutations, we substitute them with an efficient circuit based on memories. Additionally, the proposed radix-2^5 modules use more efficient rotators and include a simplification of the permutation circuits between stages. All of this results in a new and more optimized version of the MSC FFT that requires significantly less hardware resources than previous parallel pipelined FFT architectures with the same size and parallelization.

We have organized the rest of the paper as follows: In Section II, we review the related concepts to understand the proposed approach. In Section III, we present the proposed radix-2^5 1024-point MSC FFT architecture. In section IV, we provide experimental results on a field-programmable gate array (FPGA) and compare our design to previous studies. Finally, in Section V, we summarize the main conclusion of this paper.

II. BACKGROUND

A. THE FFT

An N-point discrete Fourier transform (DFT) of a complex signal $x[n]$, $n = 0, \ldots, N-1$, is calculated as

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

where k is the frequency index and the terms $W_N^{nk} = e^{-j\frac{2\pi}{N}nk}$ are the twiddle factors that correspond to the rotations. The FFT is an efficient way to compute the DFT. The Cooley-Tukey algorithm [36] calculates the FFT in $n = \log_2 N$ stages and reduces the computational cost of the DFT from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log_2 N)$.

To define the data order at the FFT stages $s = 1 \ldots n$, the index I is defined [37], being its binary representation $b_{n-1} \ldots b_1 b_0$. Throughout the paper, we use the symbol ($\equiv$) to relate a number and its binary representation. Thus, for the index it is fulfilled $I \equiv b_{n-1} \ldots b_1 b_0$. Based on this index, the FFT algorithm has the property that butterflies at stage s operate on pairs of data that differ in b_{n-s} [37]. We will use this property for the design of the proposed architecture.

B. BIT-DIMENSION PERMUTATIONS

Bit-dimension permutations are used to provide the data order required along the FFT stages. This data order is defined by its position. For a set of $N = 2^n$ data, the position is obtained as [35]

$$\mathcal{P} = \sum_{i=0}^{n-1} x_i 2^i, \tag{2}$$

where $x_{n-1}x_{n-2} \ldots x_0$ are called dimensions.

When P-parallel data arrive at different terminals, $p = \log_2 P$ dimensions are parallel and correspond to $x_{p-1} \ldots x_0$. The remaining $n - p$ dimensions define data arriving in series and correspond to $x_{n-1} \ldots x_p$. Note that $\mathcal{P}$ is used for the position, whereas P stands for the number of parallel branches. For P-parallel data, the terminals are defined in the range $T = 0 \ldots 2^p - 1$ and are related to the dimensions according to

$$T = \sum_{i=0}^{p-1} x_i 2^i, \tag{3}$$

being their binary representation

$$T \equiv x_{p-1} \ldots x_0 = T_{p-1} \ldots T_0. \tag{4}$$

Likewise, the arrival time of the data coming in consecutive clock cycles is defined in the range $t = 0 \ldots 2^{n-p} - 1$ as

$$t = \sum_{i=p}^{n-1} x_i 2^{i-p}, \tag{5}$$

being its binary representation

$$t \equiv x_{n-1} \ldots x_{n-p} = t_{n-p-1} \ldots t_0. \tag{6}$$

Therefore, (2) can be expressed as

$$\mathcal{P} = t|T \equiv t_{n-p-1} \ldots t_0 | T_{p-1} \ldots T_0, \tag{7}$$

where a vertical bar (|) separates the serial and parallel dimensions.

A bit-dimension permutation σ transforms data from an input position $\mathcal{P}_0 = u_{n-1} \ldots u_p | u_{p-1} \ldots u_0$ into an output position $\mathcal{P}_1 = \sigma(\mathcal{P}_0) = u'_{n-1} \ldots u'_p | u'_{p-1} \ldots u'_0$. In this regard, a bit-dimension permutation σ that permutes two dimensions of the data flow, i.e., $\sigma : x_j \leftrightarrow x_k$, is called elementary bit-exchange (EBE) [38]. Here, if both x_j and x_k are serial dimensions, the permutation is called serial-serial (ss) permutation. If both x_j and x_k are parallel dimensions, a parallel-parallel (pp) permutation is performed, which only routes input data to different terminals. Likewise, when the permutation is done between a serial and a parallel dimension, it is called serial-parallel (sp) permutation [35].

Figures 1(a) and 1(b) show the serial-serial and serial-parallel permutation circuits, respectively. The circuits consist of buffers of length L and multiplexers. The buffer length for the serial-serial permutation is calculated as

$$L_{ss} = \frac{2^j - 2^k}{2^p}, \tag{8}$$

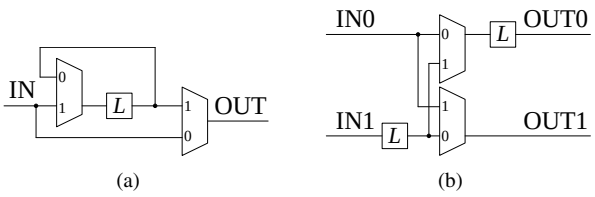


FIGURE 1. **Permutation circuits. (a) Serial-serial permutation. (b) Serial-parallel permutation.**

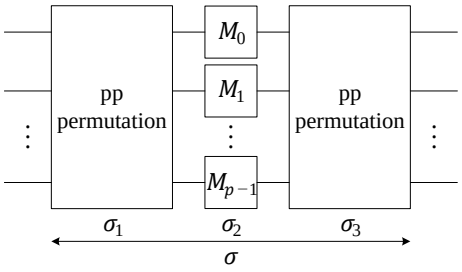


FIGURE 2. **Matrix transposition of parallel data using memories.**

whereas the buffer length for the serial-parallel permutation is

$$L_{sp} = \frac{2^j}{2^p}. \tag{9}$$

A comprehensive explanation of the circuits in Fig. 1 can be found in [35].

C. MATRIX TRANSPOSITION OF PARALLEL DATA USING MEMORIES

Let us consider that a continuous flow of $N = 2^n$ data provides P-parallel data at each clock cycle, where $P = 2^p$. In this way, the entire data set is received in $N/P = 2^{n-p}$ clock cycles. As the time of arrival is $t = 0 \dots 2^{n-p} - 1$, a control counter with $n - p$ bits can be used to count this time, i.e., $c_{n-p-1} \dots c_0$.

In this context, we can define the transposition of a square matrix for the case $n/2 > p$ as [39]

$$\begin{aligned}&\sigma(u_{n-1} \dots u_{n/2} u_{n/2-1} \dots u_p | u_{p-1} \dots u_0)\\ &= u_{n/2-1} \dots u_0 u_{n-1} \dots u_{n/2+p} | u_{n/2+p-1} \dots u_{n/2}.\end{aligned} \tag{10}$$

As shown in Fig. 2, this matrix transposition can be calculated as a sequence of permutations of the form pp-ss-pp. The matrix transposition is then obtained as [39]

$$\sigma = \sigma_3 \circ \sigma_2 \circ \sigma_1, \tag{11}$$

being

$$\begin{aligned}&\sigma_1(u_{n-1} \dots u_{n/2} u_{n/2-1} \dots u_p | u_{p-1} \dots u_0) =\\ &= u_{n-1} \dots u_p | (u_{p-1} \oplus u_{n/2+p-1}) \dots (u_0 \oplus u_{n/2}),\\ &\sigma_2(u_{n-1} \dots u_{n/2} u_{n/2-1} \dots u_p | u_{p-1} \dots u_0) =\\ &= u_{n/2-1} \dots u_p (u_{p-1} \oplus u_{n/2+p-1}) \dots\\ &\dots (u_0 \oplus u_{n/2}) u_{n-1} \dots u_{n/2+p} | u_{p-1} \dots u_0,\\ &\sigma_3(u_{n-1} \dots u_{n/2} u_{n/2-1} \dots u_p | u_{p-1} \dots u_0) =\\ &= u_{n-1} \dots u_p | (u_{p-1} \oplus u_{n/2+p-1}) \dots (u_0 \oplus u_{n/2}),\end{aligned} \tag{12}$$

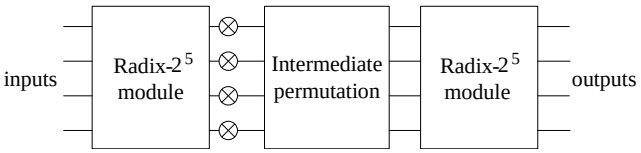


FIGURE 3. **Overview of the proposed architecture.**

where ($\oplus$) is the XOR operation. Note that the permutations σ_1 and σ_3 are the same parallel-parallel permutation. For both of them, the input terminal is $T_{in} = u_{p-1} \dots u_0$ and the output terminal is $T_{out} = (u_{p-1} \oplus c_{n/2-1}) \dots (u_0 \oplus c_{n/2-p})$.

The permutation σ_2 is a serial-serial permutation and is calculated in the memories based on their read and write addresses. These addresses are obtained from the counter. By setting the write address of all memories for the first set of N data to $W_1 = c_{n-1} \dots c_0$, the read address is obtained as

$$\begin{aligned}R_1 = {} & c_{n/2-p-1} \dots c_0 (u_{p-1} \oplus c_{n/2-1}) \dots\\ & \dots (u_0 \oplus c_{n/2-p}) c_{n-p-1} \dots c_{n/2}.\end{aligned} \tag{13}$$

For continuous flow processing, new data are written in the memory addresses that are being emptied when reading the previous data set. Thus, for the i-th set of data it is fulfilled that $W_i = R_{i-1}$, leading to [39]

$$W_i = \begin{cases} c_{n-1} \dots c_0, & \text{if } i \text{ odd},\\ c_{n/2-p-1} \dots c_0 (u_{p-1} \oplus c_{n/2-1}) \dots & \\ \dots (u_0 \oplus c_{n/2-p}) c_{n-p-1} \dots c_{n/2}, & \text{if } i \text{ even}, \end{cases} \tag{14}$$

$$R_i = \begin{cases} c_{n/2-p-1} \dots c_0 (u_{p-1} \oplus c_{n/2-1}) \dots & \\ \dots (u_0 \oplus c_{n/2-p}) c_{n-p-1} \dots c_{n/2}, & \text{if } i \text{ odd},\\ c_{n-1} \dots c_0. & \text{if } i \text{ even}. \end{cases} \tag{15}$$

III. PROPOSED 1024-POINT MSC FFT ARCHITECTURE

Figure 3 shows the overview of the proposed 4-parallel 1024-point MSC FFT architecture. It consists of two identical radix-2^5 modules, rotators ($\otimes$), and an intermediate permutation circuit that interconnects the radix-2^5 modules. In this Section, we explain all the design steps and optimizations of the proposed architecture.

A. ENABLING THE USE OF TWO IDENTICAL RADIX-2^5 MODULES

Table 1 shows the data order of the proposed $N = 2^n = 1024$-point MSC FFT at the different stages. The architecture has $n = \log_2 N = 10$ stages, 5 for each radix-2^5 module. Initially, the input data index is ordered as $\mathcal{P}_1 \equiv b_3 b_2 b_4 b_0 b_1 b_8 b_7 b_9 | b_5 b_6$. As there are two parallel dimensions ($p = 2$), the number of parallel branches in the architecture is $P = 2^p = 4$. Regarding the output data, note that they are provided in natural order at the last stage, which means that the FFT outputs are in bit-reversed order [40].

To be able to have two identical radix-2^5 modules we have designed the data order in such a way that the permutations of the index bits are the same for stages 1 to 5 and stages 6 to 10. For instance, between stages 1 to 2, the dimensions x_4 and x_2

TABLE 1. Bit-dimension index for the proposed architecture.

Stage	$x_9x_8x_7x_6x_5x_4x_3x_2 \mid x_1x_0$
1	$b_3b_2b_4b_0b_1b_8b_7\boldsymbol{b_9} \mid b_5b_6$
2	$b_3b_2b_4b_0b_1b_9b_7\boldsymbol{b_8} \mid b_5b_6$
3	$b_3b_2b_4b_0b_1b_9b_8\boldsymbol{b_7} \mid b_5b_6$
4	$b_3b_2b_4b_0b_1b_9b_8b_7 \mid b_5\boldsymbol{b_6}$
5	$b_3b_2b_4b_0b_1b_9b_8b_7 \mid b_6\boldsymbol{b_5}$
6	$b_9b_8b_7b_6b_5b_3b_2\boldsymbol{b_4} \mid b_0b_1$
7	$b_9b_8b_7b_6b_5b_4b_2\boldsymbol{b_3} \mid b_0b_1$
8	$b_9b_8b_7b_6b_5b_4b_3\boldsymbol{b_2} \mid b_0b_1$
9	$b_9b_8b_7b_6b_5b_4b_3b_2 \mid b_0\boldsymbol{b_1}$
10	$b_9b_8b_7b_6b_5b_4b_3b_2 \mid b_1\boldsymbol{b_0}$

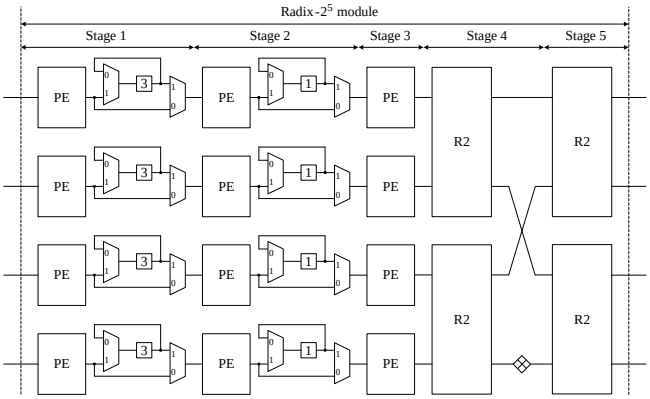


FIGURE 4. Original radix-2^5 module.

are swapped, which also occurs between stages 6 and 7. As the permutations, data order, and FFT algorithm in stages 1 to 5 and 6 to 10 are exactly the same, then the circuit is also the same.

B. RADIX-2^5 MODULE

Figure 4 shows the original structure of the 4-parallel radix-2^5 module used twice in the architecture. Through the paper, we apply multiple optimizations to this module. The original module in Fig. 4 consists of processing elements (PEs), which include butterflies and rotators, radix-2 (R2) butterflies, a trivial rotator (diamond-shaped), and serial-serial permutation circuits. In Fig. 4, all the wires carry complex-valued data. As the PEs have to operate on pairs of the data in consecutive clock cycles, in any stage s that uses PEs, the bit b_{n-s} must be placed in the least significant bit (LSB) of the serial dimensions, which is x_2. Alternatively, R2 butterflies operate on consecutive parallel branches. Therefore, in stages with R2 butterflies, the bit b_{n-s} has to be in the LSB bit of the parallel dimensions, i.e., x_0. These requirements are the same for both radix-2^5 modules. To highlight this fact, in Table 1 we have written the bit b_{n-s} at each stage in bold.

The circuit of the PEs is shown in Fig. 5, which has already been used in previous SC [26] and MSC [33], [34] FFT architectures. The input data of the PE arrive in consecutive clock

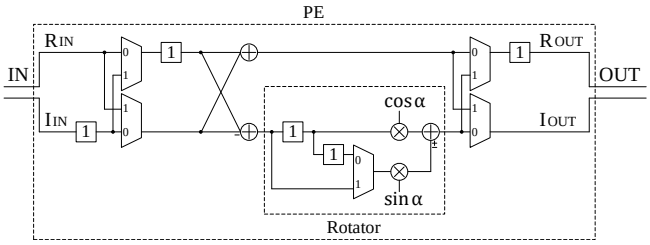


FIGURE 5. Processing element.

cycles and are split into their real and imaginary parts. After the serial-parallel permutation circuit, the butterfly operates first on the real parts of the consecutive inputs and then on their imaginary parts. This leads to using one real adder and one real subtractor instead of complex ones. Likewise, the rotation operation only appears at the lower output of the butterfly. This means that only the lower path of the PE needs to be rotated. The rotator uses two real multipliers and one adder. Thus, the total number of adders and multipliers in the PEs is halved concerning conventional butterflies and rotators.

C. OPTIMIZATION OF THE PERMUTATIONS IN THE RADIX-2^5 MODULES

In our design, two identical radix-2 modules are used. Each module has 12 PEs and 8 serial-serial permutation circuits. Each PE has 5 multiplexers and each serial-serial permutation circuit has 2 multiplexers. Taking into account both identical modules, the total number of real multiplexers in PEs is $2 \times 12 \times 5 = 120$, and in permutation circuits, it is $2 \times 8 \times 2 \times 2 = 64$ by taking into account the real and imaginary parts of the data. Thus, before optimization, the radix-2^5 modules require a total of 184 real multiplexers. Regarding real delays, each radix-2^5 module in Fig. 4 requires 92, leading to a total of $92 \times 2 = 184$.

Figure 6 shows the optimization process for the permutations between processing elements, which is inspired by the ideas in [32]. The upper part of Fig. 6 shows the detailed circuit of the first two consecutive PEs of any of the parallel branches of the original radix-2^5 module in Fig. 4 and the serial-serial permutation circuit that connects these PEs. Note that the permutation circuit has been split into two circuits to process the real and imaginary parts of the data. The lower part of Fig. 6 shows the optimized version of this circuit. In the upper part of the figure, the interconnection between PEs is carried out by a set of three permutations of the form sp-ss-sp. As these permutations only affect three serial dimensions and a parallel one, following the theory in [35], we can model them as

$$\begin{aligned} \sigma_1(u_3u_2u_1|u_0) &= u_3u_2u_0|u_1, \\ \sigma_2(u_3u_2u_1|u_0) &= u_1u_2u_3|u_0, \\ \sigma_3(u_3u_2u_1|u_0) &= u_3u_2u_0|u_1. \end{aligned} \tag{16}$$

Note that the lower input to the first permutation includes a delay by one clock cycle, which is carried out in the rotator.

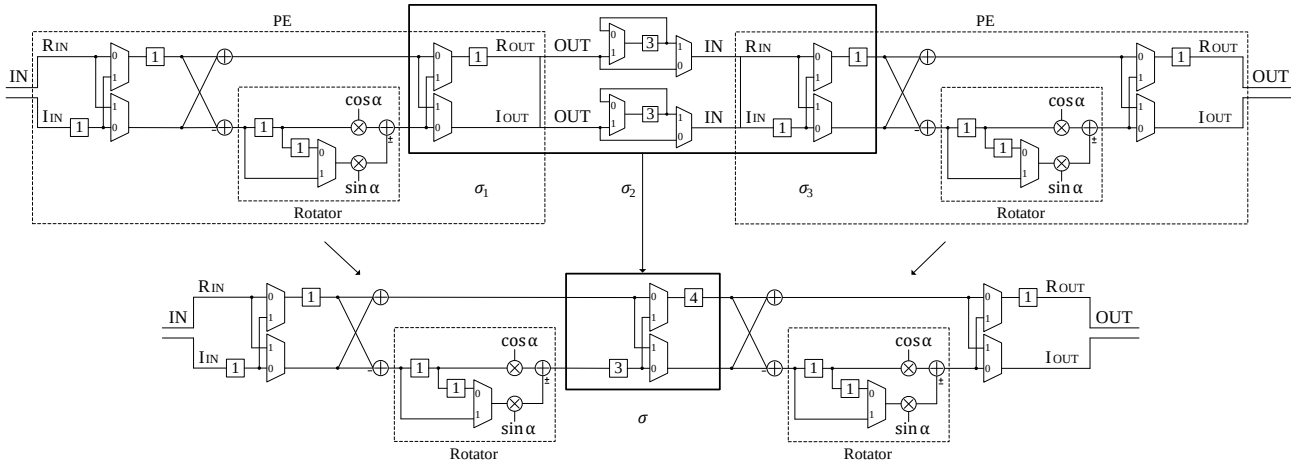


FIGURE 6. **Optimized model of the radix-2^5 module.**

If we calculate the total permutation carried out between the PEs as $\sigma = \sigma_3 \circ \sigma_2 \circ \sigma_1$, we obtain

$$\sigma(u_3u_2u_1|u_0) = u_0u_2u_1|u_3. \tag{17}$$

This permutation is the EBE $\sigma : x_3 \leftrightarrow x_0$, which can be calculated with the circuit in Fig. 1(b) with buffers of length $L = 4$, according to (9). Therefore, the permutations in the upper circuit of Fig. 6 are substituted by the simple circuit in the lower part of Fig. 6. Again, one of the delays of the lower branch of the input of the circuit is accomplished in the rotator.

By comparing the permutations in the upper and lower parts of Fig. 6, it can be observed that the number of real multiplexers has been reduced from 8 to 2 and the number of delays from 9 to 7. This improvement can be applied to the connection between the first and second stages in all the parallel branches of the radix-2^5 module in Fig. 4. Likewise, the same approach can be applied to the connections between the second and third stages. In this case, the serial-serial permutation only has one delay, the number of real multiplexers is reduced from 8 to 2, and the number of delays from 5 to 3. Thus, the total saving in multiplexers in the entire FFT architecture is $6 \times 4 \times 2 \times 2 = 96$. This reduces their number from 184 down to 88, which represents a saving of more than 50%. Regarding the number of real registers, it is reduced by $2 \times 4 \times 2 \times 2 = 32$, from 184 down to 152, which is a saving of 17%.

As a result of this optimization, the proposed version of the radix-2^5 module is shown in Fig. 7. Input data and data from the output of stage 3 to the output of the module are complex-valued, whereas data connecting stages 1, 2, and 3 are real-valued, which can be deduced from the optimization in Fig. 6.

D. INTERMEDIATE PERMUTATION CIRCUIT

From Sections III-B and III-C, it can be observed that the radix-2^5 modules of the architecture use short buffers. This

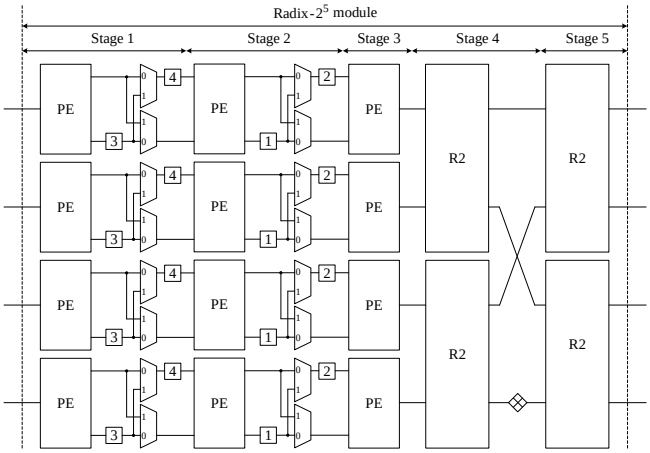


FIGURE 7. **Optimized radix-2^5 module.**

is because we have designed the proposed architecture in such a way that all the longer buffers are placed between the radix-2^5 modules, which creates a permutation between stages 5 and 6 where multiple dimensions are exchanged. This not only simplifies the radix-2^5 modules but also has the advantage that we can find a way to carry out the intermediate permutation in an optimized way, as described next.

Figure 8 shows the intermediate permutation between the radix-2^5 modules, which results from moving all the permutations with long buffers out of the radix-2^5 modules. In the proposed architecture, this permutation tranforms the order $b_3b_2b_4b_0b_1b_9b_8b_7|b_6b_5$ in the fifth stage to the order $b_9b_8b_7b_6b_5b_3b_2b_4|b_0b_1$ in the sixth stage, according to Table 1. It can be noted that the intermediate permutation is fulfilled by multiple exchanges in dimensions. Some of these exchanges are done between serial dimensions such as $x_9 \leftrightarrow x_4$, $x_8 \leftrightarrow x_3$, $x_7 \leftrightarrow x_2$ and the others are done between serial and parallel dimensions, i.e., $x_6 \leftrightarrow x_1$, $x_5 \leftrightarrow x_0$. In Fig. 8, the length of the buffers is obtained by applying (8) and (9). According to them, each branch has serial-serial

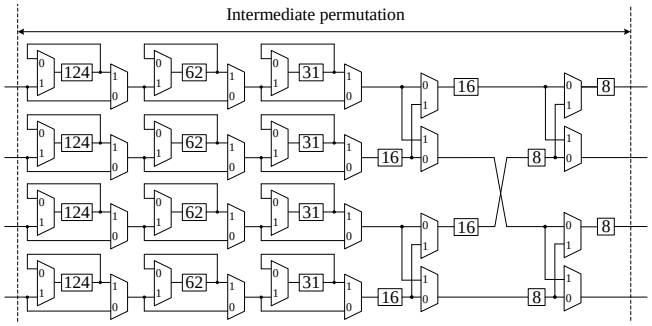


FIGURE 8. **Intermediate permutation before optimization.**

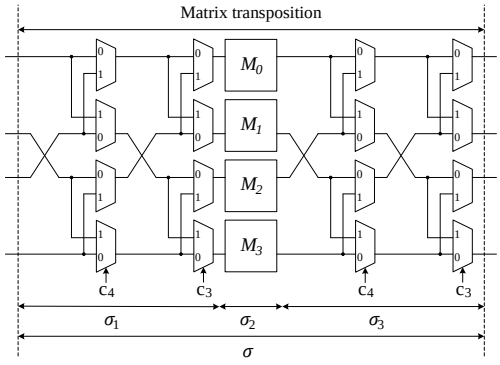


FIGURE 9. **Optimized intermediate permutation by calculating a matrix transposition using memories.**

permutation circuits with 124, 62, and 31 delays, and serial-parallel permutation circuits with 16 and 8 delays. In total, the circuit requires 20 buffers with a total of $4 \times 241 = 964$ delays and 32 multiplexers.

In the proposed architecture, the data orders at stages 5 and 6 have not been designed arbitrarily. In fact, they have been defined in such a way that the permutation between these two stages is a matrix transposition. Specifically, from the orders in stages 5 and 6 in Table 1 we can deduce that the permutation is

$$\sigma(u_9 \dots u_2|u_1u_0) = u_4u_3u_2u_1u_0u_9u_8u_7|u_6u_5. \tag{18}$$

Therefore, we can apply the theory of Section II-C to obtain a circuit that carries out this permutation using memories in an optimized way. The resulting circuit is shown in Fig. 9 and calculates the permutation σ in (18) as a combination of three permutations according to (11).

To derive the circuit in Fig. 9, we apply (12) to the specific case of (18), which leads to

$$\begin{aligned}
\sigma_1(u_9 \dots u_0) &= u_9u_8u_7u_6u_5u_4u_3u_2|(u_1 \oplus u_6)(u_0 \oplus u_5),\\
\sigma_2(u_9 \dots u_0) &= u_4u_3u_2(u_1 \oplus u_6)(u_0 \oplus u_5)u_9u_8u_7|u_1u_0,\\
\sigma_3(u_9 \dots u_0) &= u_9u_8u_7u_6u_5u_4u_3u_2|(u_1 \oplus u_6)(u_0 \oplus u_5).
\end{aligned} \tag{19}$$

Note that the permutations σ_1 and σ_3 are the same and permute the data only in parallel dimensions. It means that these permutations route the data from a certain terminal at the input of the permutation to another terminal at the

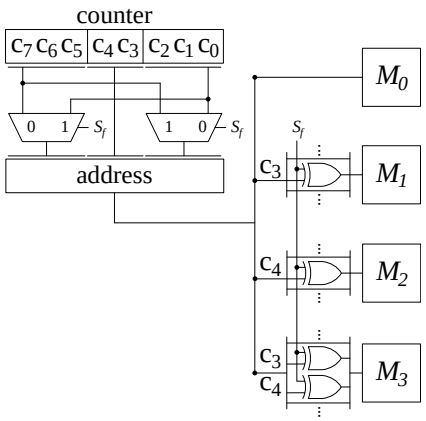


FIGURE 10. **Address generation circuit for the memories in the intermediate permutation.**

output. For the permutations σ_1 and σ_3, the input terminal is $T_{in} = u_1u_0 = T_1T_0$ and the output terminal is $T_{out} = (u_1 \oplus u_6)(u_0 \oplus u_5) = (T_1 \oplus t_4)(T_0 \oplus t_3)$. This leads to the shuffling carried out by the multiplexers before the memories for σ_1 and after the memories for σ_3. Note that they are controlled by the c_3 and c_4 bits of the counter, which correspond to t_3 and t_4, respectively.

Regarding σ_2, it permutes data arriving in series by using the memories $M_0 - M_3$. By applying (14) and (15), σ_2 the address signals for the memories become

$$\begin{aligned}
M_0 : W_i = R_{i-1} &= \begin{cases} c_7c_6c_5c_4c_3c_2c_1c_0, & \text{if } S_f = 0,\\ c_2c_1c_0c_4c_3c_7c_6c_5, & \text{if } S_f = 1,\end{cases}\\
M_1 : W_i = R_{i-1} &= \begin{cases} c_7c_6c_5c_4c_3c_2c_1c_0, & \text{if } S_f = 0,\\ c_2c_1c_0c_4\bar{c}_3c_7c_6c_5, & \text{if } S_f = 1,\end{cases}\\
M_2 : W_i = R_{i-1} &= \begin{cases} c_7c_6c_5c_4c_3c_2c_1c_0, & \text{if } S_f = 0,\\ c_2c_1c_0\bar{c}_4c_3c_7c_6c_5, & \text{if } S_f = 1,\end{cases}\\
M_3 : W_i = R_{i-1} &= \begin{cases} c_7c_6c_5c_4c_3c_2c_1c_0, & \text{if } S_f = 0,\\ c_2c_1c_0\bar{c}_4\bar{c}_3c_7c_6c_5, & \text{if } S_f = 1,\end{cases}
\end{aligned} \tag{20}$$

where the control signal S_f toggles for every incoming FFT.

Figure 10 shows the address generation circuit for memories M_0 to M_3, where an 8-bit counter is used to obtain the read and write addresses. The multiplexers after the counter are used to swap their upper and lower bits when $S_f = 1$, whereas the bits are unchanged if $S_f = 0$. Likewise, the signal S_f is used to negate some of the bits before memories M_1 to M_3 to obtain the cases for which $S_f = 1$ in (20).

As a result, the proposed approach to calculate the intermediate permutation converts the circuit with 20 buffers and 32 multiplexers in Fig. 8 into the compact solution with 4 memories and 16 multiplexers shown in Fig. 9. Furthermore, the read and write addresses are easily generated by a simple control circuit based on a counter, according to Fig. 10.

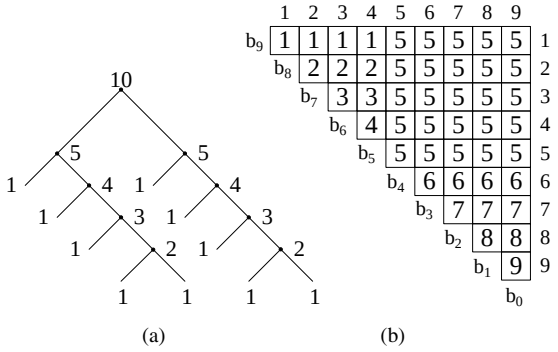


FIGURE 11. Algorithm used in the proposed radix-2^5 1024-point FFT architecture. (a) Binary tree representation. (b) Triangular matrix representation.

E. OBTAINING THE TWIDDLE FACTORS IN THE MSC FFT

FFT algorithms are usually described by their binary tree representation [41]–[43] and their triangular matrix representation [40], [43]. As can be seen in the binary tree representation in Fig. 11(a), the upper node is divided into two equal 2^5-point FFTs. These FFTs are calculated in the radix-2^5 modules of the proposed architecture. Note also that each 2^5-point FFT is decomposed according to the radix-2 decimation in frequency (DIF) FFT algorithm [43].

The rotation values $\phi_s(I)$ can be obtained from the index $I \equiv b_9 \dots b_0$ at any stage s from the triangular matrix representation in Fig. 11(b). In the triangular matrix representation, rows and columns are numbered as $x = 1 \dots n-1$ and $y = 1 \dots n-1$ [40], respectively, and the numbers in the matrix cells represent the stages. According to this, the rotations at any stage s are calculated as [43]

$$\phi_s(I) = \sum_s b_{n-x} \cdot b_{n-y-1} \cdot 2^{n+(x-y)-2}. \tag{21}$$

For instance, in Fig. 11(b), $s = 2$ for the cells of the matrix where $x = 2$ and $y = \{2, 3, 4\}$. If we apply (21), we get $\phi_2(I) = b_8 \cdot b_7 \cdot 2^8 + b_8 \cdot b_6 \cdot 2^7 + b_8 \cdot b_5 \cdot 2^6 = b_8 \cdot [b_7 b_6 b_5] \cdot 2^6$, where values inside $[\cdot]$ are the digits of a binary number.

The rotations for all the stages of the proposed FFT architecture are derived in the same way, which results in

$$\begin{aligned}
\phi_1(I) &= b_9 \cdot [b_8 b_7 b_6 b_5] \cdot 2^5, \\
\phi_2(I) &= b_8 \cdot [b_7 b_6 b_5] \cdot 2^6, \\
\phi_3(I) &= b_7 \cdot [b_6 b_5] \cdot 2^7, \\
\phi_4(I) &= b_6 \cdot b_5 \cdot 2^8, \\
\phi_5(I) &= [b_5 b_6 b_7 b_8 b_9] \cdot [b_4 b_3 b_2 b_1 b_0], \\
\phi_6(I) &= b_4 \cdot [b_3 b_2 b_1 b_0] \cdot 2^5, \\
\phi_7(I) &= b_3 \cdot [b_2 b_1 b_0] \cdot 2^6, \\
\phi_8(I) &= b_2 \cdot [b_1 b_0] \cdot 2^7, \\
\phi_9(I) &= b_1 \cdot b_0 \cdot 2^8.
\end{aligned} \tag{22}$$

To design the rotators in the architecture, we have to rewrite (22) in terms of time and terminals as given in (7). As a consequence, by replacing the b_i bits in (22) with the corresponding t_i and T_i bits in (7) we obtain

$$\begin{aligned}
\phi_1(I) &= \phi_6(I) = t_0 \cdot [t_2 t_1 T_0 T_1] \cdot 2^5, \\
\phi_2(I) &= \phi_7(I) = t_0 \cdot [t_1 T_0 T_1] \cdot 2^6, \\
\phi_3(I) &= \phi_8(I) = t_0 \cdot [T_0 T_1] \cdot 2^7, \\
\phi_4(I) &= \phi_9(I) = T_0 \cdot T_1 \cdot 2^8, \\
\phi_5(I) &= [T_0 T_1 t_0 t_1 t_2] \cdot [t_5 t_7 t_6 t_3 t_4].
\end{aligned} \tag{23}$$

Note that $\phi_i(I) = \phi_{i+5}(I)$ for $i = 1 \dots 4$. This is due to the fact that the radix-2^5 modules are identical in our architecture. Furthermore, the t_i and T_i bits give the information about the time of arrival data and the terminals. To obtain the twiddle factors at the four parallel branches of each stage, we set the terminal bits $T_1 T_0$ to the corresponding terminal, which takes the values $\{00, 01, 10, 11\}$ from the upper to the lower branch.

As stages 6 to 9 are the same as stages 1 to 4, we only focus on the latter ones. In the first stage, the rotations are

$$\begin{aligned}
\phi_{10}(I) &= t_0 \cdot [t_2 t_1 00] \cdot 2^5 \\
\phi_{11}(I) &= t_0 \cdot [t_2 t_1 10] \cdot 2^5 \\
\phi_{12}(I) &= t_0 \cdot [t_2 t_1 01] \cdot 2^5 \\
\phi_{13}(I) &= t_0 \cdot [t_2 t_1 11] \cdot 2^5,
\end{aligned} \tag{24}$$

where $\phi_{sj}(I)$ is the value of the rotations at stage s and branch j. The value $j = 0$ corresponds to the upper branch in the architecture and $j = 3$ corresponds to the lower branch. In the second stage, the rotations for all branches are calculated as

$$\begin{aligned}
\phi_{20}(I) &= t_0 \cdot [t_1 00] \cdot 2^6 \\
\phi_{21}(I) &= t_0 \cdot [t_1 10] \cdot 2^6 \\
\phi_{22}(I) &= t_0 \cdot [t_1 01] \cdot 2^6 \\
\phi_{23}(I) &= t_0 \cdot [t_1 11] \cdot 2^6.
\end{aligned} \tag{25}$$

In the third stage, the rotations are

$$\begin{aligned}
\phi_{30}(I) &= t_0 \cdot [00] \cdot 2^7 \\
\phi_{31}(I) &= t_0 \cdot [10] \cdot 2^7 \\
\phi_{32}(I) &= t_0 \cdot [01] \cdot 2^7 \\
\phi_{33}(I) &= t_0 \cdot [11] \cdot 2^7.
\end{aligned} \tag{26}$$

Likewise, the rotations at stage four are

$$\begin{aligned}
\phi_{40}(I) &= 0 \\
\phi_{41}(I) &= 0 \\
\phi_{42}(I) &= 0 \\
\phi_{43}(I) &= 2^8.
\end{aligned} \tag{27}$$

The twiddle factors W_{32}^{ϕ} as a function of the time are obtained by simply substituting the values $t_2 t_1 t_0$ in equations (24) to (27). These twiddle factors are summarized in Table 2. For the first stage, the rotations in (24) depend on the binary values of $t_2 t_1 t_0$. As the t_i values represent the time of arrival and the rotations only depend on the 3 LSBs of t, the rotation values repeat every 8 consecutive samples. For stage 2, the rotations in (25) only depend on $t_1 t_0$, so they repeat every 4 clock cycles. For stage 3, they repeat every other clock cycle and for stage 4 they are constant, as they do not depend on any t_i.

In equations (24) to (26) it can also be observed that all the rotations are obtained through a multiplication by t_0. Thus,

TABLE 2. Twiddle factors of the radix-2^5 module.

Stage 1				
$t_2t_1t_0$	$\phi_{10}(I)$	$\phi_{11}(I)$	$\phi_{12}(I)$	$\phi_{13}(I)$
0	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}
1	W_{32}^{0}	W_{32}^{2}	W_{32}^{1}	W_{32}^{3}
2	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}
3	W_{32}^{4}	W_{32}^{6}	W_{32}^{5}	W_{32}^{7}
4	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}
5	W_{32}^{8}	W_{32}^{10}	W_{32}^{9}	W_{32}^{11}
6	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}
7	W_{32}^{12}	W_{32}^{14}	W_{32}^{13}	W_{32}^{15}
Stage 2				
t_1t_0	$\phi_{20}(I)$	$\phi_{21}(I)$	$\phi_{22}(I)$	$\phi_{23}(I)$
0	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}
1	W_{32}^{0}	W_{32}^{4}	W_{32}^{2}	W_{32}^{6}
2	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}
3	W_{32}^{8}	W_{32}^{12}	W_{32}^{10}	W_{32}^{14}
Stage 3				
t_0	$\phi_{30}(I)$	$\phi_{31}(I)$	$\phi_{32}(I)$	$\phi_{33}(I)$
0	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}
1	W_{32}^{0}	W_{32}^{8}	W_{32}^{4}	W_{32}^{12}
Stage 4				
t	$\phi_{40}(I)$	$\phi_{41}(I)$	$\phi_{42}(I)$	$\phi_{43}(I)$
All	W_{32}^{0}	W_{32}^{0}	W_{32}^{0}	W_{32}^{8}

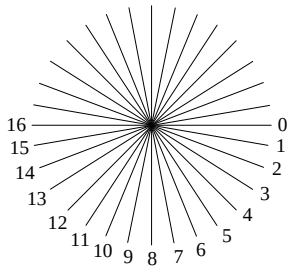


FIGURE 12. Rotation angles for W_{32}^{ϕ}.

when $t_0 = 0$, no rotation needs to be calculated. Therefore, the twiddle factor W_{32}^{0} appears every other clock cycle in Table 2 for the stages 1 to 3.

F. DESIGNING THE ROTATORS OF THE PROPOSED MSC FFT

Figure 12 shows the angles used for W_{32}^{ϕ}, which are the result of dividing the circumference into 32 equal parts. It can be observed that some angles are symmetric regarding $0°$, $-45°$ and $-90°$. In this context, a symmetric angle set (SAS) is the group of angles that can be obtained by symmetries from other angles in the group [44]. These symmetries are achieved by applying a trivial rotation by $0°$, $\pm 90°$ or $180°$ and/or changing the real and imaginary parts of the rotation coefficient. Table 3 shows all the symmetric angle sets in the W_{32}^{ϕ} twiddle factor. For instance, W_{32}^{3}, W_{32}^{5}, W_{32}^{11}, and W_{32}^{13} are in the same SAS.

TABLE 3. Symmetric angle sets for W_{32}^{ϕ} twiddle factors.

SAS 0	SAS 1	SAS 2	SAS 3	SAS 4
0	1	2	3	4
8	7	6	5	
	9	10	11	12
16	15	14	13	

The rotator $\phi_{10}(I)$ in Table 2 must rotate by the angles $\phi = \{0, 4, 8, 12\}$. According to Table 3, $\phi = 0$ and $\phi = 8$ are in SAS 0, and $\phi = 4$ and $\phi = 12$ are in SAS 4. A rotator that can rotate multiple angles in M different SAS is called an M-rot [45]. Thus, the rotator $\phi_{10}(I)$ is a 2-rot.

Figure 13 shows the circuit for the rotator $\phi_{10}(I)$. It receives the real and imaginary parts of data to be rotated in consecutive clock cycles and considers that $W_{32}^{0} = 1$, $W_{32}^{4} = (181 - j181)/256$, $W_{32}^{8} = -j$, and $W_{32}^{12} = (-181 - j181)/256$. The rotation is divided into two parts. The first part scales the inputs by $256/256$ or $181/256$ depending on the control signal s_0. The second part carries out a multiplication by 1, $(1 - j)$, $-j$, or $(-1 - j)$, which is controlled by s_1, s_2, s_3, and s_4. Table 4 shows the values of the control signals depending on the rotation that must be carried out. Here, c_0 represents the LSB of the control counter.

The rotator $\phi_{11}(I)$ in Table 2 computes $\phi = \{2, 6, 10, 14\}$. As all the rotations are in SAS 2 in Table 3, $\phi_{11}(I)$ is a 1-rot. Conversely, the rotators $\phi_{12}(I)$ and $\phi_{13}(I)$ are 2-rot, as they rotatate by $\phi = \{1, 5, 9, 13\}$ and $\phi = \{3, 7, 11, 15\}$, respectively. In the proposed architecture, $\phi_{12}(I)$ and $\phi_{13}(I)$ are optimized by exchanging some rotations between them according to the circuit in Fig. 14(a). This circuit makes it possible to calculate rotations $\phi = \{1, 7, 9, 15\}$ in $\phi_{12}(I)$ and $\phi = \{3, 5, 11, 13\}$ in $\phi_{13}(I)$, so both rotators become 1-rot.

Figure 14(b) shows the structure of the rotators $\phi_{11}(I), \phi_{12}(I)$ and $\phi_{13}(I)$. This circuit has already been described in [34] and the details on how it works can be found there. It receives the real and imaginary parts of the input in consecutive clock cycles. In the figure, C and S refer to the real and imaginary parts of the rotation coefficient $C + jS$. In this work, we improve the calculation of $C + jS$ with respect to [34], by using optimized coefficients. These coefficientes are $C + jS = (473 + j196)/512$, $C + jS = (251 + j50)/256$, and $C + jS = (213 + j142)/256$ for $\phi_{11}(I), \phi_{12}(I)$ and $\phi_{13}(I)$, respectively. Their shift-and-add implementations are shown in Figs. 14(c), 14(d), and 14(e), respectively. The multiplication by 473 and 196 requires only 4 adders. Likewise, the multiplication by 251 and 50, and the multiplication by 213 and 142 requires only 3 adders. This makes the rotators hardware-efficient.

In the second stage, $\phi_{20}(I)$ involves rotations by $\phi = 0$ and $\phi = 8$ from Table 2, which are in SAS 0 according to Table 3 and correspond to trivial rotations. Figure 15(a) shows the circuit of the rotator $\phi_{20}(I)$. To rotate by $\phi = 0$, data simply passes through the circuit by setting $s = 0$. To rotate

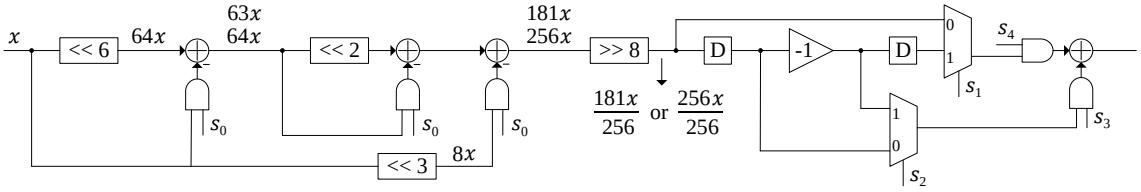


FIGURE 13. Rotator $\phi_{10}(I)$.

TABLE 4. Control signals for the rotator $\phi_{10}(I)$.

Control Signal	Rotation Angle			
	1	1-j	-j	-1-j
s_0	0	1	0	1
s_1	c_0	c_0	c_0	c_0
s_2	0	0	Any	1
s_3	1	1	0	1
s_4	0	1	1	1

by $\phi = 8$, which is a multiplication by $-j$, first $s = 0$ to let x pass to the register. Then, $s = 1$ so that y passes to the output while x is multiplied by -1 and stored back in the register, which is output in the next clock cycle.

The rotator ϕ_{21} rotates by $\phi = 4$ or $\phi = 12$, which corresponds to multiplying by $(1-j)$ or $(-1-j)$. These angles are a subset of the angles for $\phi_{10}(I)$, so ϕ_{21} can be calculated with the rotator in Fig. 13. For ϕ_{21}, we only have to use the control signals s_1 and s_2, whereas the other signals are fixed to $s_0 = 0$, $s_3 = 1$, and $s_4 = 1$, which allows for simplifying the circuit. Similarly, the angles in $\phi_{22}(I)$ and $\phi_{23}(I)$ are a subset of the angles in ϕ_{11}, so the rotator in Fig. 14(b) can be used to calculate them. For $\phi_{22}(I)$, $s_0 = 0$, whereas for $\phi_{23}(I)$, $s_0 = 1$. As the control signal is constant in both cases, the corresponding multiplexers can be removed.

In the third stage, $\phi_{30}(I)$ does not calculate any rotation, $\phi_{31}(I)$ is a trivial rotator as $\phi_{20}(I)$, and both $\phi_{32}(I)$ and $\phi_{33}(I)$ can use a simplified version of the rotator $\phi_{10}(I)$.

In the fourth stage, the first three branches do not calculate any rotation, whereas ϕ_{43} is a constant trivial rotation calculated with the circuit in Fig. 15(b). Contrary to the trivial rotator in Fig. 15(a), in the trivial rotator in Fig. 15(b) the real and imaginary parts of the data arrive at the same clock cycle. Note that swapping inputs and multiplying x by -1 results in multiplying the inputs by $-j$. In fact, this trivial rotator can be integrated into the butterfly previous to it by changing the real and imaginary parts of the output of the subtraction and also changing the inputs of one of the subtractors.

G. NUMBER OF COMPONENTS

The proposed 4-parallel 1024-point radix-2^5 MSC FFT architecture uses a total data memory size of 1088. This includes four memories with the size of 256 in the matrix transposition and 64 registers in the permutation circuits.

Regarding adders in butterflies, each PE has 2 real adders, which is equivalent to 1 complex adder. Three stages of PEs with 4-parallel branches in two radix-2^5 modules lead to $1 \times 3 \times 4 \times 2 = 24$ real adders in PEs. Similarly, the butterflies in R2 have $2 \times 2 \times 2 = 8$ complex adders that correspond to 16 real adders. This leads to a total of 40 complex adders in the architecture.

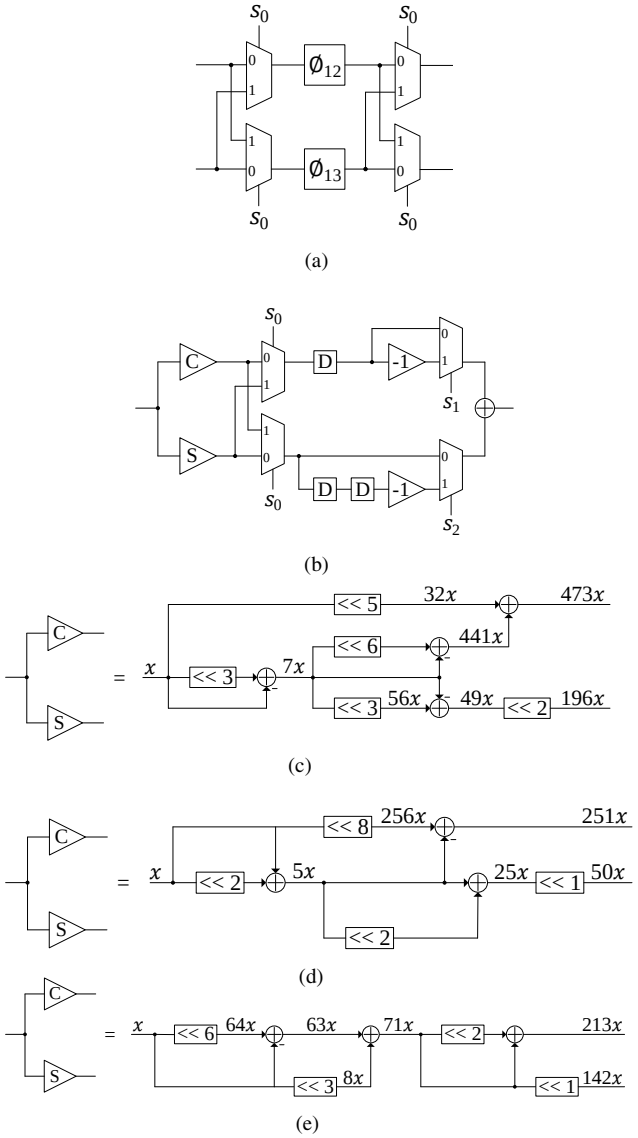
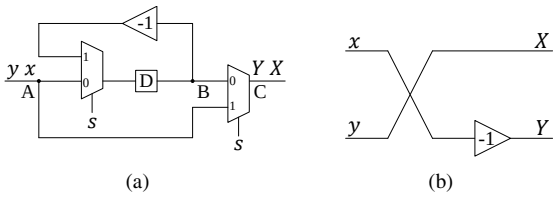


FIGURE 14. Rotators for $\phi_{11}(I)$, $\phi_{12}(I)$, and $\phi_{13}(I)$. (a) Rotators structure. (b) Exchange circuit for $\phi_{12}(I)$ and $\phi_{13}(I)$. (c) Multiplication by coefficient $C + jS = 473 + j196$ in $\phi_{11}(I)$. (d) Multiplication by $C + jS = 251 + j50$ in $\phi_{12}(I)$. (e) Multiplication by coefficient $C + jS = 213 + j142$ in $\phi_{13}(I)$.



(a) (b)

FIGURE 15. Trivial rotators. (a) Trivial rotator $\phi_{20}(I)$ for serial data. (b) Trivial rotator $\phi_{43}(I)$ for parallel data.

TABLE 5. Comparison of 4-parallel 1024-point pipelined FFT architectures

Parameter	[19]	[24]	[23]	[16]	[20]	Prop.
Type	MDC	MDC	MDC	M^{2}DF	CM	MSC
Radix	2^5	2^2	2	2^3	2^5	2^5
N	1024	1024	1024	1024	1024	1024
P	4	4	4	4	4	4
Adders	40	40	44	40	40	40
G-Rots	4	12	16	11	4	4
3-Rots	2	0	0	0	0	0
2-Rots	4	0	0	0	0	1
1-Rots	4	0	0	0	0	6
C-Rots	0	0	0	4	22	2
Data Mem.	1020	1024	3068	2048	1116	1088

Regarding multipliers, there are 4 general rotators (G-rot) after the first radix-2^5 module as shown in Fig. 3. In each module, there is a half 2-rot, 6 half 1-rot, and 2 half constant rotators (C-rot), leading to a total of 1 2-rot, 6 1-rot, and 2 C-rot in both modules.

Regarding multiplexers, with the help of the optimization in the radix-2^5 module shown in Fig. 6, 16 real multiplexers are used in stages 1 and 3, while there is no multiplexer in stage 2. In the permutation circuits between the PEs, 16 real multiplexers are used. Additionally, 16 complex multiplexers are used in the matrix transposition. Thus, the total usage of real multiplexers is 96, which is equivalent to 48 complex multiplexers.

IV. EXPERIMENTAL RESULTS AND COMPARISON

Table 5, compares the proposed architecture to previous radix-2^k pipeline FFT architectures. For a fair comparison, we have only selected previous FFT architectures with the same size and number of parallel branches, i.e., $N = 1024$ and $P = 4$. The comparison in Table 5 compares different approaches from an architectural point of view in terms of adders, rotators, and memory.

By comparing complex adders, the proposed architecture achieves the minimum number of adders, which is also the same as in [16], [19], [20], [24] and less than in [23]. Regarding the use of memory, [19], [20], [24] have pretty close results to the proposed approach, whereas we get significantly better results than [16], [23]. Regarding rotators, our architecture improves the number of rotators and their complexity. The number of general rotators is 12 in [24], 16 in [23], and 11 in [16]. The proposed approach uses only 4 G-rots,

TABLE 6. Comparison of 4-parallel 1024-point pipelined FFT implemented on FPGA

Parameter	[19]	[24]	[23]	[16]	[20]	Prop.
Type	MDC	MDC	MDC	M^{2}DF	CM	MSC
Radix	2^5	2^2	2	2^3	2^5	2^5
N	1024	1024	1024	1024	1024	1024
P	4	4	4	4	4	4
WL	16	16	16	16	16	16
FPGA	V6	V6	V5	V6	V7	V7
Slices	1420	1351	-	-	2631	1615
LUTs	-	-	4116	3796	-	4682
FFs	-	-	1920	4432	-	5910
DSP Slices	16	48	72	45	12	12
BRAMs	12	12	0	10	0	4
f_{CLK} (MHz)	253	227	380	292	680	420
Th. (MS/s)	1012	910	1520	1168	2720	1680
Lat. (cycles)	265	285	767	538	394	300
Lat. (μs)	1.04	1.25	2.02	1.84	0.58	0.71
SQNR	40.30	-	-	-	-	50.16
P (W)	-	-	-	-	1.68	0.98
NP $\left(\frac{\text{mW}}{\text{MHz}}\right)$	-	-	-	-	2.47	2.33

-: Not provided.

which represents a saving of at least 63%. Compared to approaches with the same number of general rotators [19], [20], our approach reduces the number of non-general rotators. Specifically, [20] requires 22 non-general rotators, whereas the proposed architecture uses only 9. Despite the proposed approach using 1-rots and 2-rots, the savings in total rotator complexity are significant. Compared to [19], the proposed approach reduces the number of total rotators from 10 to 9 and, at the same time, reduces the complexity of the non-general rotators by using mostly 1-rots and C-rots. As a result, thanks to all the architectural optimizations carried out in the proposed approach, the architecture presented in this paper is the most resource-efficient one among comparable state-of-the-art approaches.

Regarding experimental results, the proposed architecture has been implemented on a Virtex 7 XC7VX330T -3 FFG1157 FPGA. It works at 420 MHz with a latency of 300 clock cycles and a throughput of 1680 MS/s. The signal-to-quantization-noise ratio (SQNR) is 50.16 dB with 16 bits of word length. Additionally, the power consumption is 0.98 W and the normalized power consumption is 2.33 mW/MHz.

Table 6 compares the experimental results to previous FFT architecture. The table only includes FFT architecture with $N = 1024$ points, $P = 4$ parallel branches, and a word length $WL = 16$, so that the architectures are comparable. All approaches in the table are implemented on Virtex 6 (V6) or Virtex 7 (V7) FPGAs and the comparison is carried out in terms of area, clock frequency (f_{CLK}), throughput (Th.), latency (Lat.), signal-to-quantization-noise ration (SQNR), power consumption (P), and normalized power consumption (NP). From the results in the table, the proposed architecture uses a total of 12 DSP slices for the G-rots and 4 BRAMs for

the matrix transposition. This represents a huge reduction in terms of hardware resources concerning most previous FFT architectures [16], [19], [23], [24]. Note that the proposed architecture reduces the number of DSP Slices by 25 % and the number of BRAMs by 66 % with respect to [19], which is the most resource-efficient implementation among [16], [19], [23], [24]. Thus, the area savings with respect to [16], [23], [24] are even more significant.

The only previous work with area results that are closer to our work is [20]. Compared to it, our approach reduces the number of slices by 38 % at the cost of adding only 4 BRAMs. Furthermore, the proposed approach has 24 % lower latency in terms of clock cycles and 4 % less power consumption when normalizing it in terms of mW/MHz.

As a result, the proposed approach offers a very efficient solution in terms of hardware resources, as it requires a small number of rotators with low complexity. This leads to a hardware implementation with a small number of slices, LUTs, FFs, DSP slices, and BRAMs.

V. CONCLUSIONS

In this paper, we have proposed an optimized 4-parallel 1024-point radix-2^5 MSC FFT architecture that reduces the number of hardware resources significantly concerning previous parallel pipelined FFT architectures.

The proposed design carries out multiple optimizations over previous MSC FFT architectures. First, the architecture makes use of a radix-2^5 module. This module has been optimized by simplifying the permutations between PEs and optimizing the rotators. As this module is used twice in the architecture, the optimizations are also applied twice.

Apart from optimizing the radix-2^5 modules, the most complex permutation circuits in the architecture have been placed between the modules and implemented as a matrix transposition, which reduces the complexity of the circuit.

As a result, the proposed architecture achieves the lowest complexity in terms of hardware resources among FFT architectures with the same size and parallelization so far.

REFERENCES

[1] V. M. Bautista, M. Garrido, and M. López-Vallejo, "Serial butterflies for non-power-of-two FFT architectures in 5G and beyond," *IEEE Trans. Circuits Syst. I*, vol. 70, no. 10, pp. 3992–4003, Oct. 2023.

[2] M. Mahdavi, O. Edfors, V. Öwall, and L. Liu, "A low latency and area efficient FFT processor for massive MIMO systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2017, pp. 1–4.

[3] T. Kamazaki, S. K. Okumura, Y. Chikada, T. Okuda, Y. Kurono, S. Iguchi, S. Mitsuishi, Y. Murakami, N. Nishimuta, H. Mita, and R. Sano, "Digital spectro-correlator system for the Atacama compact array of the Atacama large millimeter/submillimeter array," *Publ. Astron. Soc. Japan*, vol. 64, no. 2, p. 29, Apr. 2012.

[4] 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*, vol. 66, no. 10, pp. 3863–3873, Oct 2019.

[5] N. Linty and L. Lo Presti, "Doppler frequency estimation in GNSS receivers based on double FFT," *IEEE Trans. Veh. Technol.*, vol. 65, no. 2, pp. 509–524, Feb. 2016.

[6] T. Yu, E. Wang, S. Jin, Y. Wang, J. Huang, X. Liu, and W. Zhan, "Responses of GNSS ZTD variations to ENSO events and prediction model based on FFT-LSTME," *IEEE Trans. Geosci. Remote Sens.*, vol. 61, no. 4101417, pp. 1–17, Mar. 2023.

[7] Z. Kaya, M. Garrido, and J. Takala, "Memory-based FFT architecture with optimized number of multiplexers and memory usage," *IEEE Trans. Circuits Syst. II*, vol. 70, no. 8, pp. 3084–3088, Aug. 2023.

[8] Z. Kaya and M. Garrido, "Low-latency 64-parallel 4096-point memory-based FFT for 6G," *IEEE Trans. Circuits Syst. I*, vol. 70, no. 10, pp. 4004–4014, Oct. 2023.

[9] S. Liu and D. Liu, "A high-flexible low-latency memory-based FFT processor for 4G, WLAN, and future 5G," *IEEE Trans. VLSI Syst.*, vol. 27, no. 3, pp. 511–523, Mar. 2019.

[10] Y. Guo, Z. Wang, Q. Hong, H. Luo, X. Qiu, and L. Liang, "A 60-mode high-throughput parallel-processing FFT processor for 5G/4G applications," *IEEE Trans. VLSI Syst.*, vol. 31, no. 2, pp. 219–232, Dec. 2023.

[11] V. M. Bautista and M. Garrido, "An automatic generator of non-power-of-two SDF FFT architectures for 5G and beyond," in *Proc. Conf. Design Circuits Integrated Syst.*, Nov. 2023, pp. 61–66.

[12] X.-Y. Shih, H.-R. Chou, and Y.-Q. Liu, "Design and implementation of flexible and reconfigurable SDF-based FFT chip architecture with changeable-radix processing elements," *IEEE Trans. Circuits Syst. I*, vol. 65, no. 11, pp. 3942–3955, Nov. 2018.

[13] Y.-N. Chang, "An efficient VLSI architecture for normal I/O order pipeline FFT design," *IEEE Trans. Circuits Syst. II*, vol. 55, no. 12, pp. 1234–1238, Dec. 2008.

[14] X. Liu, F. Yu, and Z. Wang, "A pipelined architecture for normal I/O order FFT," *J. of Zhejiang Univ. Sci. C*, vol. 12, no. 1, pp. 76–82, Jan. 2011.

[15] X. Zhou, X. Chen, Y. He, and X. Mou, "A flexible-channel MDF architecture for pipelined radix-2 FFT," *IEEE Access*, vol. 11, no. 4, pp. 38 023–38 033, Apr. 2023.

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

[17] 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.

[18] M. Garrido, J. Grajal, M. A. Sánchez, and O. Gustafsson, "Pipelined radix-2^k feedforward FFT architectures," *IEEE Trans. VLSI Syst.*, vol. 21, no. 1, pp. 23–32, Jan. 2013.

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

[20] M. Garrido and P. Malagón, "The constant multiplier FFT," *IEEE Trans. Circuits Syst. I*, vol. 68, no. 1, pp. 322–335, Jan. 2021.

[21] N. Le Ba and T. T. Kim, "An area efficient 1024-point low power radix-2^2 FFT processor with feed-forward multiple delay commutators," *IEEE Trans. Circuits Syst. I*, vol. 65, no. 10, pp. 3291–3299, Oct. 2018.

[22] P. Paz and M. Garrido, "A 5.2-GSps 8-parallel 1024-point MDC FFT," in *Proc. Conf. Design Circuits Integrated Syst.*, Nov. 2023, pp. 55–60.

[23] 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. VLSI Syst.*, vol. 24, no. 6, pp. 2402–2406, Jun. 2016.

[24] M. Garrido, M. Acevedo, A. Ehliar, and O. Gustafsson, "Challenging the limits of FFT performance on FPGAs," in *Proc. Int. Symp. Integrated Circuits*, Dec. 2014, pp. 172–175.

[25] W.-L. Tsai, S.-G. Chen, and S.-J. Huang, "Reconfigurable radix-$2^k \times 3$ feedforward FFT architectures," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2019, pp. 1–5.

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

[27] S. Park and D. Jeon, "A modified serial commutator architecture for real-valued fast Fourier transform," in *Proc. IEEE Workshop Signal Process. Syst.*, Oct. 2020, pp. 1–6.

[28] J. Hazarika, M. T. Khan, S. R. Ahamed, and H. B. Nemade, "Energy efficient VLSI architecture of real-valued serial pipelined FFT," *IET Comp. Digital Tech.*, vol. 13, no. 6, pp. 461–469, Jun. 2019.

[29] 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*, vol. 65, no. 11, pp. 1693–1697, Nov. 2018.

[30] H. Fang, B. Zhang, F. Yu, B. Zhao, and Z. Ma, "A pipelined algorithm and area-efficient architecture for serial real-valued FFT," *IEEE Trans. Circuits Syst. I*, vol. 69, no. 11, pp. 4533–4537, Nov. 2022.

[31] N. K. Unnikrishnan, M. Garrido, and K. K. Parhi, "Effect of finite word-length on SQNR, area and power for real-valued serial FFT," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2019, pp. 1–5.

[32] H. Fang, Z. Ma, F. Yu, B. Zhao, and B. Zhang, "Optimised serial commutator FFT architecture in terms of multiplexers," *IEEE Trans. Circuits Syst. II*, vol. 71, no. 1, pp. 445–449, Jan. 2024.
[33] 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.*, Oct. 2020, pp. 1–5.
[34] G.-T. Deng, M. Garrido, S.-G. Chen, and S.-J. Huang, "Radix-2^k MSC FFT architectures," *IEEE Access*, vol. 11, pp. 81 497–81 510, Jul. 2023.
[35] M. Garrido, J. Grajal, and O. Gustafsson, "Optimum circuits for bit-dimension permutations," *IEEE Trans. VLSI Syst.*, vol. 27, no. 5, pp. 1148–1160, May 2019.
[36] 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.
[37] M. Garrido, "A survey on pipelined FFT hardware architectures," *J. Signal Process. Syst.*, vol. 94, no. 11, pp. 1345–1364, Nov. 2022.
[38] A. Edelman, S. Heller, and L. Johnsson, "Index transformation algorithms in a linear algebra framework," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 12, pp. 1302–1309, Dec. 1994.
[39] M. Garrido and P. Pirsch, "Continuous-flow matrix transposition using memories," *IEEE Trans. Circuits Syst. I*, vol. 67, no. 9, pp. 3035–3046, Sep. 2020.
[40] M. Garrido, "A new representation of FFT algorithms using triangular matrices," *IEEE Trans. Circuits Syst. I*, vol. 63, no. 10, pp. 1737–1745, Oct. 2016.
[41] H. Lee and I. Park, "Balanced binary-tree decomposition for area-efficient pipelined FFT processing," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 4, pp. 889–900, Apr. 2007.
[42] F. Qureshi and O. Gustafsson, "Generation of all radix-2 fast Fourier transform algorithms using binary trees," in *Proc. European Conf. Circuit Theory Design*, Aug. 2011, pp. 677–680.
[43] 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. Springer, 2019.
[44] R. Andersson and M. Garrido, "Using rotator transformations to simplify FFT hardware architectures," *IEEE Trans. Circuits Syst. I*, vol. 67, no. 12, pp. 4784–4793, Dec. 2020.
[45] R. Andersson, "FFT hardware architectures with reduced twiddle factor sets," Master's Thesis, Dpt. of Electrical Engineering, Linköping University, Jun. 2014.

ZEYNEP KAYA received her M.Sc. and Ph.D. degrees in Electrical and Electronics Engineering from Eskisehir Osmangazi University, Turkey, in 2015 and 2021. Since 2021, she has been an Assistant Professor at Bilecik Seyh Edebali University. From September 2022 to September 2023 she joined Universidad Politécnica de Madrid (UPM) as a Post-Doctoral Researcher.

Her current research interests are optimized hardware architectures for the fast Fourier transform (FFT) including data management and memory addressing schemes. Her research includes high-performance circuits, designs for small area, low latency, and low power consumption.

MARIO GARRIDO (M'07-SM'19) received the M.Sc. and Ph.D. degrees in electrical engineering from Universidad Politécnica de Madrid (UPM), Spain, in 2004 and 2009, respectively. In 2010 he moved to Sweden to work as a postdoctoral researcher at the Department of Electrical Engineering at Linköping University. From 2012 to 2019 he was Associate Professor in the same department. In 2019 he moved back to UPM, where he holds a Ramón y Cajal Research Fellowship. So far, he has been the author of more than 50 scientific publications, and he appeared in the "World's Top 2% Scientists List" elaborated by Standford University both in 2022 and 2023.

His research focuses on optimized hardware design for signal-processing applications. This includes the design of hardware architectures for the fast Fourier transform (FFT), circuits for data management, the CORDIC algorithm, neural networks, 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.