Feedforward FFT Hardware Architectures Based on Rotator Allocation

Mario Garrido, *Member, IEEE*, Shen-Jui Huang, and Sau-Gee Chen

***Abstract*— In this paper, we present new feedforward FFT hardware architectures based on rotator allocation. The rotator allocation approach consists in distributing the rotations of the FFT in such a way that the number of edges in the FFT that need rotators and the complexity of the rotators are reduced. Radix-2 and radix-2^k feedforward architectures based on rotator allocation are presented in this paper. Experimental results show that the proposed architectures reduce the hardware cost significantly with respect to previous FFT architectures.**

***Index Terms*— Fast Fourier transform (FFT), multi-path delay commutator (MDC), pipelined architecture, radix-2, radix-2^k.**

I. Introduction

THE fast Fourier transform (FFT) is one of the most important algorithms in the field of digital signal processing. It is used to calculate the discrete Fourier transform (DFT) efficiently. In order to meet the high performance and real-time requirements of modern applications, hardware designers have always tried to implement efficient architectures for the computation of the FFT. In this context, pipelined hardware architectures [1]–[27] are widely used, because they provide high throughput and low latency suitable for real time, as well as a reasonably low area and power consumption.

There are three main types of pipelined FFT architectures: feedback (FB), feedforward (FF) and serial commutator (SC). First, feedback architectures [1]–[14] 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) [1]–[5], which process a continuous flow of one sample per clock cycle, and multi-path delay feedback (MDF) or parallel feedback [6]–[14], which process several samples in parallel. Second, feedforward architectures [3], [4], [15]–[21], which mainly consist of multi-path delay commutator (MDC) FFTs [3], do not have feedback loops and each stage passes the processed data to the next stage. MDC architectures process several samples in parallel. A comparison of parallel FFT architectures is provided in [16]. Finally, SC FFT architectures [22] are characterized by the use of circuits for bit-dimension permutation of serial data.

If the purpose is to improve parallel FFT architectures, MDC FFTs already achieve the minimum number of butterflies with 100% usage and the minimum amount of total memory [16]. However, there is still room for improvement regarding rotators: Different radices lead to different amount of rotators, as shown in [16]. For instance, radix-2^3 and radix-2^4 architectures have less general rotators than radix-2 and radix-2^2. However, instead of general rotators, they have a larger number of W_8 and W_{16} rotators. Therefore, there is a trade-off among rotators.

This paper presents a new idea called rotator allocation. The purpose is to reduce the number and the complexity of the rotators even further. The number of rotators is reduced by only having rotations by 0° in some of the FFT edges. The complexity of the rotators is reduced by distributing the rotations of the FFT so that similar rotations are calculated by the same rotator. For instance, a rotation by 45° and a rotation by 135° can be easily calculated by the same rotator as it only consists of a rotation by 45° plus a trivial rotation by 0° or 90°. New feedforward FFT architectures based on rotator allocation are proposed in this paper. These architectures use radices that lead to a small amount of hardware resources.

The paper is organized as follows. Section II reviews the FFT algorithm. Section III summarizes the types of rotators used in FFT architectures and assigns a cost to them in terms of equivalent adders. Section IV provides general guidelines to design an FFT hardware architecture. Section V explains the design of FFT hardware architectures using rotator allocation. Section VI presents the proposed architectures both for radix-2 and radix-2^k. Section VII compares the proposed architectures to the state-of-the-art in terms of the number of multipliers and equivalent adders. Section VIII provides experimental results and compares them to to previous works in the literature. Finally, Section IX summarizes the main conclusions of the paper.

II. The FFT Algorithm

The N-point DFT of an input sequence $x[n]$ is defined as:

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

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

Manuscript received February 9, 2017; revised May 12, 2017 and June 9, 2017; accepted June 27, 2017. Date of publication August 15, 2017; date of current version January 25, 2018. This work was supported by the Swedish ELLIIT Program. This paper was recommended by Associate Editor G. J. Dolecek. *(Corresponding author: Mario Garrido.)*

M. Garrido is with the Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden (e-mail: mario.garrido.galvez@liu.se).

S.-J. Huang is with Novatek Corporation, Hsinchu 300, Taiwan (e-mail: shenray.ee95g@g2.nctu.edu.tw).

S.-G. Chen is with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu 300, Taiwan (e-mail: sgchen@cc.nctu.edu.tw).

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

Digital Object Identifier 10.1109/TCSI.2017.2722690

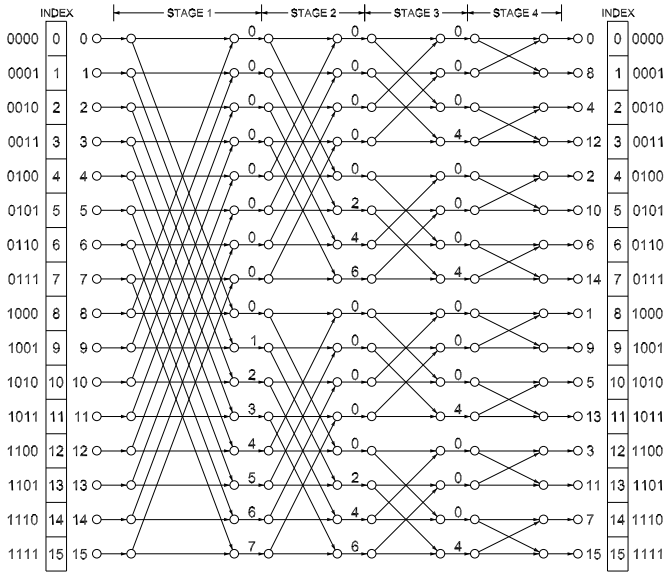

Fig. 1. Flow graph of a 16-point DIF FFT.

To calculate the DFT, the FFT based on the Cooley-Tukey algorithmm [28] is mostly used. The Cooley-Tukey algorithm reduces the number of operations from $O(N^2)$ for the DFT to $O(N \cdot \log_2 N)$ for the FFT.

Figure 1 shows the flow graph of a 16-point radix-2 FFT according to the Cooley-Tukey algorithm, decomposed using decimation in frequency (DIF) [29]. The FFT consists of $n = \log_2 N$ stages. At each stage of the graphs, $s \in \{1, \ldots, n\}$, 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, samples for which $\phi = 0$ do not need to be rotated. Likewise, if $\phi \in \{0, N/4, N/2, 3N/4\}$ the samples must be rotated by 0°, 270°, 180° or 90°, which correspond to complex multiplications by 1, $-j$, -1 and j, respectively. These rotations are considered trivial, because they can be carried out 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 graph, where $b_{n-1} \ldots b_0$ is the binary representation of I. This index is used in the explanations throughout the paper.

Different radices only differ in the rotations at the different FFT stages [30], whereas the butterflies are the same. Thus, different algorithms lead to different distributions of rotations, which may influence the design of hardware architectures, as shown throughout this paper.

III. Hardware Rotators for the FFT

This section defines some terms used later in the paper.

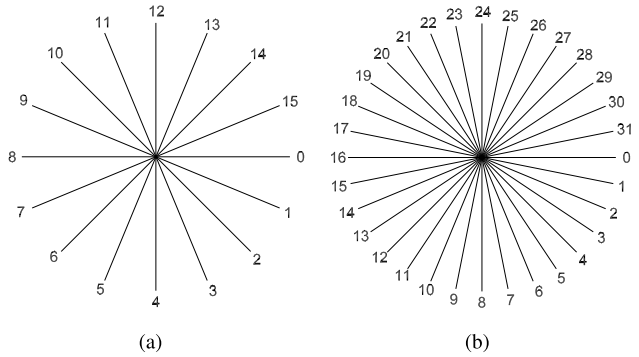

Fig. 2. Rotation angles used in the FFT for different values of ϕ. (a) 16-point FFT. (b) 32-point FFT.

TABLE I
Symmetric Angle Sets for the 16-Point FFT

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

TABLE II
Symmetric Angle Sets for the 32-Point FFT

Set 0	Set 1	Set 2	Set 3	Set 4
0	1	2	3	4
	7	6	5	
8	9	10	11	12
	15	14	13	
16	17	18	19	20
	23	22	21	
24	25	26	27	28
	31	30	29	

A *general rotator* is a rotator that can carry out a rotation by any angle, which is provided as an input. A *single constant rotator* (SCR) is a rotator that carries out a rotation by a single constant angle [31]. A *multiple constant rotator* (MCR) is a rotator that rotates by an angle selected among several constant angles.

A *symmetric angle set* (SAS) is a set of angles $n\pi/2 \pm \alpha$, where $n = 0, \ldots, 3$ and $\alpha \in [0, \pi/4]$. Any rotation in a symmetric angle set can be calculated as a rotation by $\alpha \in [0, \pi/4]$, a trivial rotation and a exchange of the real and imaginary part of the rotation coefficient. Figure 2 shows the angles used in a 16-point FFT ($\phi = 0, \ldots, 15$) and in a 32-point FFT ($\phi = 0, \ldots, 31$) which correspond to dividing the circumference into 16 and 32 equal parts, respectively. These angles form several symmetric angle sets, which are shown in Tables I and II.

An *M-rotator* or *M-rot* is a rotator that can rotate any number of angles in M different symmetric angle sets. For instance, a rotator that rotates by 0°, 45° and 135° is a 2-rot, as it rotates angles in the symmetric angle sets $n\pi/2$ and $n\pi/2 \pm \pi/4$. Likewise, the FFT twiddle factor W_8 is a 2-rot, the twiddle factor W_{16} is a 3-rot (note the 3 SAS in Table I) and the twiddle factor W_{32} is a 5-rot (note the

TABLE III
ESTIMATION OF EQUIVALENT NUMBER OF ADDERS IN FFT ROTATORS

Rotator		Hardware Cost				
Type	Approach	Add. Rot.	Multiplexers	Add. Mux.	Scaling Compensation	Total Equivalent Adders
ARBITRARY SCALING						
1-rot	CCSSI [31]: SCR arbitrary scaling	6	0	0	-	6
2-rot	CCSSI [31]: MCR uniform scaling	6	6	1.5	-	7.5
3-rot	CCSSI [31]: MCR uniform scaling	8	24	6	-	14
General	CORDIC II [32]	16	33	8.25	-	24.25
General	Memoryless CORDIC [33]	21	26	6.5	-	27.5
General	Complex Multiplier: Booth encoding	34	0	0	-	34
UNITY SCALING						
1-rot	CCSSI [31]: SCR with unity scaling	10	0	0	0	10
2-rot	CCSSI [31]: MCR with unity scaling	10	10	2.5	0	12.5
3-rot	CCSSI [31]: MCR with unity scaling	10	30	7.5	0	17.5
General	CORDIC II [32]	16	33	8.25	6	30.25
General	Memoryless CORDIC [33]	21	26	6.5	4	31.5
General	Complex Multiplier: Booth encoding	34	0	0	0	34

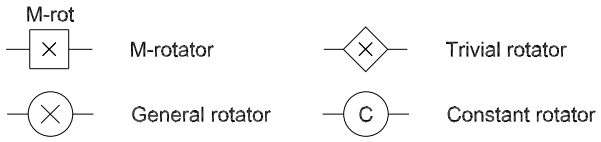

Fig. 3. Symbols used in the paper for the different types of rotators.

5 SAS in Table II). The symbols used in this paper for the different types of rotators are shown in Fig. 3.

Table III shows an estimation of the hardware cost of different types of rotators in terms of adders. The table is divided into rotators for arbitrary scaling and unity scaling. *Arbitrary scaling* means that any scaling is accepted and *unity scaling* means that the scaling must be a power of two [31]. M-rots are obtained using the CCSSI approach [31], whereas general rotators are obtained either using the CORDIC algorithm [33], [32] or complex multipliers. The table shows the adders (Add. Rot.) and multiplexers involved in the rotation. Considering that a multiplexer is equivalent to 1/4 adders [34], the column Add. Mux. shows the equivalent number of adders of the multiplexers involved in the rotation. The next column is the number of adders needed for scaling compensation. This is not needed in case of arbitrary scaling. The scaling compensation factor in the CORDIC II is $K = 1/(1.563 \cdot 1.008) = 0.6347 \approx 0.6348 = 2^{-1} + 2^{-3} + 2^{-7} + 2^{-9}$. Thus, it needs 6 adders, 3 for the real part and three for the imaginary part. The scaling compensation in the memoryless CORDIC needs 4 adders, 2 for the real part and 2 for the imaginary part [33]. The last column of the table shows the total number of equivalent adders, which is the result of adding the adders involved in the rotation, the equivalent adders of the multiplexers and the adders of the scaling compensation. We use the results in Table III to compute the number of equivalent adders of the proposed FFT architectures.

IV. DESIGNING AN FFT HARDWARE ARCHITECTURE

In order to design an FFT hardware architecture, we have to be aware of the FFT properties introduced in [16]. The first property, which is general for any FFT architecture and any N, is that at any FFT stage, butterflies operate on data whose index I differ in b_{n-s}, where n is the number of FFT stages and s is the specific stage that we are considering. This fact can be observed in the flow graph of Fig. 1. In this flow graph, the index has $n = 4$ bits, i.e., $I \equiv b_3\ b_2\ b_1\ b_0$. At the first stage, the butterflies operate on samples whose indexes differ in $b_{n-s} = b_{4-1} = b_3$. This happens for samples with indexes 0 and 8, 1 and 9, etc. For the second stage, the different bit is $b_{n-s} = b_{4-2} = b_2$. Note for instance, that the data with indexes 0 and 4 are operated together in the butterfly at stage 2. For the third and fourth stages, the correponding bits are b_1 and b_0, respectively.

Therefore, if we want to design an FFT hardware architecture, we have to assure that at each stage s, the indexes of the inputs to any butterfly at any time instant differ in b_{n-s}. Note that the term butterfly refers now to a hardware component of the architecture, not to the mathematical operation of the algorithm in the flow graph. If we consider the example of Fig. 3 in [16], we observe that this property is fulfilled at all the stages. In this figure b_{n-s} is at the lowest parallel dimension, which corresponds to the pair of samples that flow into the butterflies at the same clock cycle.

The property of b_{n-s} is the only requirement set by the butterflies in FFT hardware architecture. As long as this property is met, we can have any data order at the different FFT stages. This allows for exploring a variety of data orders at the FFT stages. This is what is done in the current paper, as explained later in Section V.

The second FFT property refers to the rotations at the FFT stages. At each stage, any sample with index I must be rotated according to equation (2) by a value ϕ that depends on the index I and on the stage, s. We can represent it as $\phi_s(I)$. The work [30] explains how to calculate $\phi_s(I)$, which only depends on the FFT algorithm that is used.

By combining these ideas, we lead to the following conclusions: On the one hand, any order at any stage of any FFT hardware architecture is possible as long as the property of b_{n-s} is met at the input of the butterflies. On the other hand, to any index I at any FFT stage corresponds a specific rotation $\phi_s(I)$. As a result, we can play with the data order at different FFT stages to look for patterns that allow for a more optimized distribution of rotations. This is the idea behind the proposed rotator allocation approach.

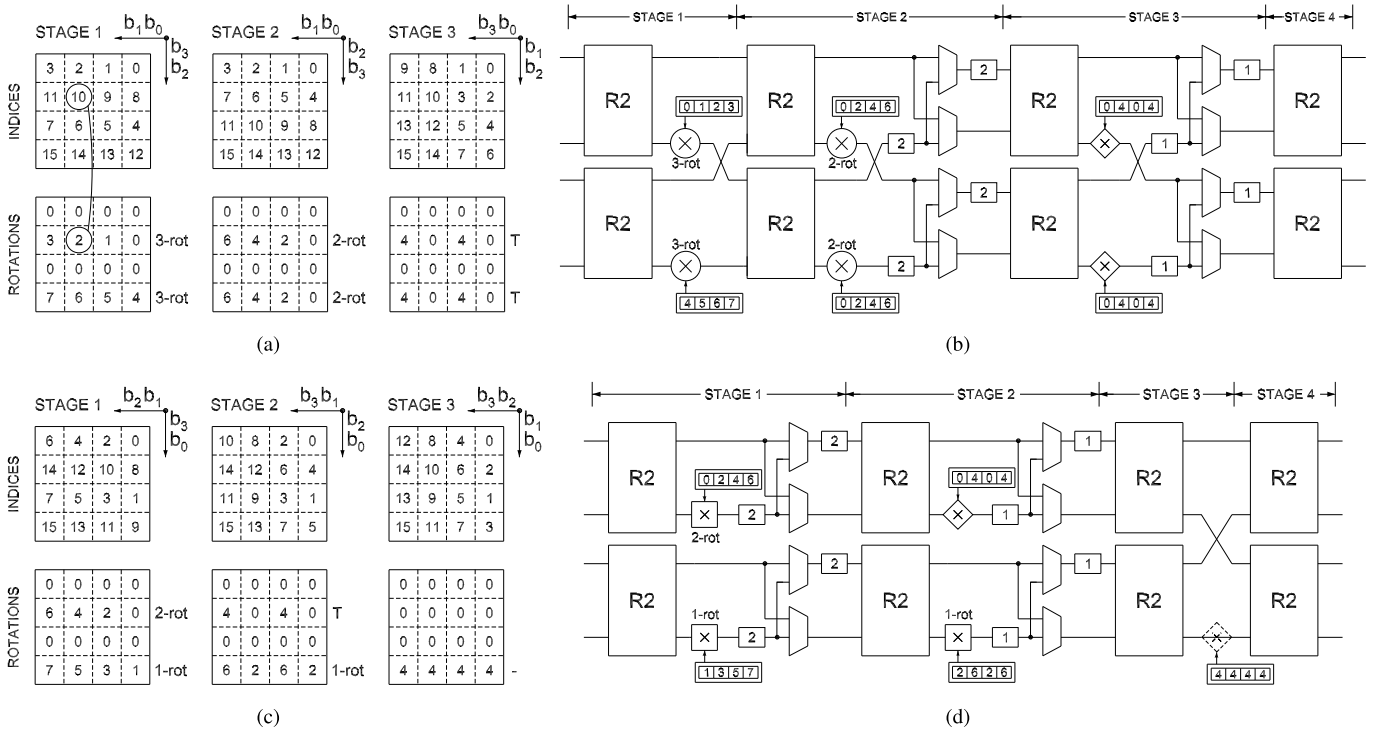

Fig. 4. Rotator allocation of a 16-point 4-parallel FFT. (a) Layout not aware of the rotator allocation. (b) FFT architecture for the layout in Fig. 4(a). (c) Layout aware of the rotator allocation. (d) FFT architecture for the layout in Fig. 4(c).

V. FFT Design Using Rotator Allocation

The purpose of the FFT design using rotator allocation is to distribute the rotations of the FFT in such a way that the number and complexity of the required rotators is reduced.

Fig. 4(a) shows an example of a layout for the first three stages of a 16-point 4-parallel FFT. The indexes in the figure show how data flows at the different stages. Each element in the matrices of indexes is the index value according to Fig. 1. Data flows from left to right. Thus, values in the same column are data that flow in parallel and values in the same row flow through the same path in consecutive clock cycles. The matrices of rotations show the value $\phi_s(I)$ that corresponds to each of the indexes according to the flow graph in Fig. 1. For instance, the rotation corresponding to the index 10 at stage 1 is $\phi_s(I) = \phi_1(10) = 2$ according to Fig. 1. This case is highlighted in Fig. 4(a).

As for the indexes, each column in the matrices of rotations are rotations that are calculated in parallel at the same clock cycle, whereas rotations in the same row are calculated in consecutive clock cycles by the same rotator. According to this, each row of the matrices of rotations are the rotations that must be calculated by a single rotator in consecutive clock cycles. For instance, stage 1 needs a rotator by $\phi = \{0, 1, 2, 3\}$ and a rotator by $\phi = \{4, 5, 6, 7\}$, which are 3-rots according to Table I. Stage 2 includes 2 2-rots and stage 3 includes 2 trivial rotators (T).

The layout in Fig. 4(a) translates into the FFT architecture in Fig. 4(b), which shows the rotators at each stage, as well as the content of the rotation memories.

By applying rotator allocation we aim to reduce the number of rotators and their complexity. Rotator allocation simply consists of reorganizing the matrices of indexes and, therefore, the matrices of rotations, in such a way that the matrices of rotations have less rotators (if possible) and the complexity of the rotators is lower. On the one hand, fewer rotators are achieved when there are more rows in the matrices of rotations whose elements are $\phi = 0$. On the other hand, the complexity of the rotators is reduced when the rotations in the same row are in less SAS.

The procedure of rotator allocation consists in distributing the bits $b_{n-1} \dots b_0$ of the index I into serial and parallel dimensions. Serial dimensions correspond to data arriving at the same input terminal in series and parallel dimensions refers to data arriving at parallel terminals. Depending on the FFT size $N = 2^n$ and the number of parallel data in the FFT $P = 2^p$, the number of serial bits is $n-p = \log_2(N) - \log_2(P)$ and the number of parallel ones is $p = \log_2(P)$. For the example in Fig. 4, $N = 16$ and $P = 4$, so there are $n - p = \log_2(16) - \log_2(4) = 2$ serial dimensions and $p = \log_2(P) = 2$ parallel ones. The alternatives to allocate the bits correspond to all the possible permutations, i.e.,

$$\begin{array}{c} \text{Serial|Parallel} \\ \begin{array}{cccc} b_3b_2|b_1b_0 & b_2b_3|b_1b_0 & b_3b_2|b_0b_1 & b_2b_3|b_0b_1 \\ b_3b_1|b_2b_0 & b_1b_3|b_2b_0 & b_3b_1|b_0b_2 & b_1b_3|b_0b_2 \\ b_3b_0|b_2b_1 & b_0b_3|b_2b_1 & b_3b_0|b_1b_2 & b_0b_3|b_1b_2 \\ b_0b_1|b_2b_3 & b_1b_0|b_2b_3 & b_0b_1|b_3b_2 & b_1b_0|b_3b_2 \\ b_0b_2|b_1b_3 & b_2b_0|b_1b_3 & b_0b_2|b_3b_1 & b_2b_0|b_3b_1 \\ b_1b_2|b_0b_3 & b_2b_1|b_0b_3 & b_1b_2|b_3b_0 & b_2b_1|b_3b_0 \end{array} \end{array} \quad (3)$$

However, neither the order of the serial bits nor the order of the parallel bits affect the complexity of the

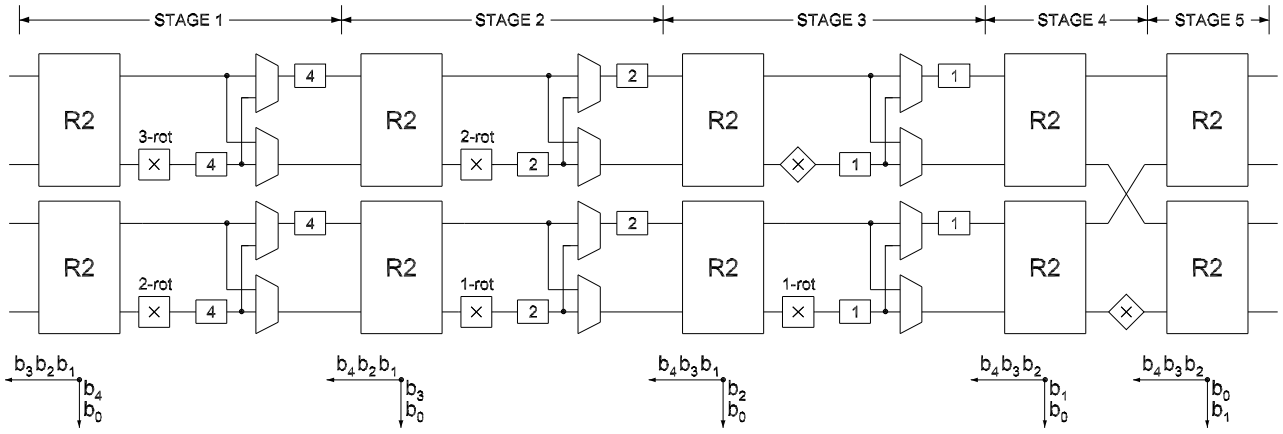

Fig. 5. Proposed 32-point 4-parallel radix-2 MDC DIF FFT architecture.

rotators: A different order of the serial bits changes the order of the rotations, but the same rotations are calculated by the rotators. A different order of the parallel bits changes the edges in which the rotators are placed, but the rotators are the same. Therefore, the alternatives in each row of equation (3) have the same complexity, so only one alternative per row needs to be evaluated. According to this, the number of alternatives that need to be evaluated at each FFT stage is

$$\binom{n}{p} = \frac{n!}{p!(n-p)!} \tag{4}$$

Note that we can choose any order in equation (3) and design the data management of the FFT to achieve the desired order. However, it is in general a good idea to place the rotators close to the butterflies of the same state. This requires that the data order at the rotator respects the property of b_{n-s} explained in previous section. Alternatively, if the property of b_{n-s} is not met at the rotator, then additional permutation circuits are needed between the butterfly and the rotator. The architectures in this paper correspond to the case of placing the rotator just after the butterfly.

Figure. 4(c) shows a layout of indexes and rotations that is aware of the rotator allocation, and the corresponding FFT architecture is shown in Fig. 4(d). The bits of the serial and parallel dimensions at each stage are shown in Fig. 4(c). In this layout, stage 1 requires a 2-rot and a 1-rot, stage 2 a trivial rotator and a 1-rot, and stage 3 a trivial rotator. The trivial rotator in stage 3 can be hard wired due to the fact that all the rotations have the same value. Comparing this example with the example in Figs. 4(a) and 4(b), it can be observed that the number of rotators and their complexity have been reduced. Instead of 2 3-rots, 2 2-rots and 2 trivial rotators, the design aware of rotator allocation only needs a 2-rot, 2 1-rot and 1 trivial rotator.

As a conclusion, the rotator allocation can simplify significantly the complexity of the rotators at the different stages of the FFT, while achieving the same complexity in terms of butterflies and shuffling circuits.

TABLE IV
TWIDDLE FACTORS VALUES (ϕ) FOR THE 32-POINT 4-PARALLEL RADIX-2 MDC DIF FFT ARCHITECTURE IN FIG. 5

Stage 1	0	0	0	0	0	0	0	0	
	0	2	4	6	8	10	12	14	3-rot
	0	0	0	0	0	0	0	0	
	1	3	5	7	9	11	13	15	2-rot
Stage 2	0	0	0	0	0	0	0	0	
	0	4	8	12	0	4	8	12	2-rot
	0	0	0	0	0	0	0	0	
	2	6	10	14	2	6	10	14	1-rot
Stage 3	0	0	0	0	0	0	0	0	
	0	8	0	8	0	8	0	8	T
	0	0	0	0	0	0	0	0	
	4	12	4	12	4	12	4	12	1-rot
Stage 4	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	
	8	8	8	8	8	8	8	8	-

VI. PROPOSED FEEDFORWARD FFT ARCHITECTURES BASED ON ROTATOR ALLOCATION

A. Radix-2 Feedforward FFTs Based on Rotator Allocation

Fig. 5 shows the proposed 32-point 4-parallel radix-2 DIF FFT architecture based on rotator allocation. The architecture consists of five stages that include radix-2 butterflies (R2), rotators and circuits for data management. In total, the architecture requires one 3-rot, two 2-rot, two 1-rot and a trivial rotator. As for the 16-point FFT architecture in Fig. 4(d), the trivial rotator in the last stage can be hard wired. Note also that the 32-point architecture is equal to the 16-point architecture plus an extra first stage. The twiddle factor values (ϕ) for the different stages of the architecture are shown in Table IV.

Fig. 6 shows the proposed 32-point 8-parallel radix-2 DIF FFT architecture based on rotator allocation. This architecture can be derived from the 4-parallel radix-2 FFT in Fig. 5 in the following way: Samples that arrive in even clock cycles in the 4-parallel architecture are processed by the upper part of the 8-parallel architecture, whereas samples that arrive in odd clock cycles in the 4-parallel architecture are processed by the lower part of the 8-parallel architecture. In the 8-parallel architecture, the rotators are simplified compared to the

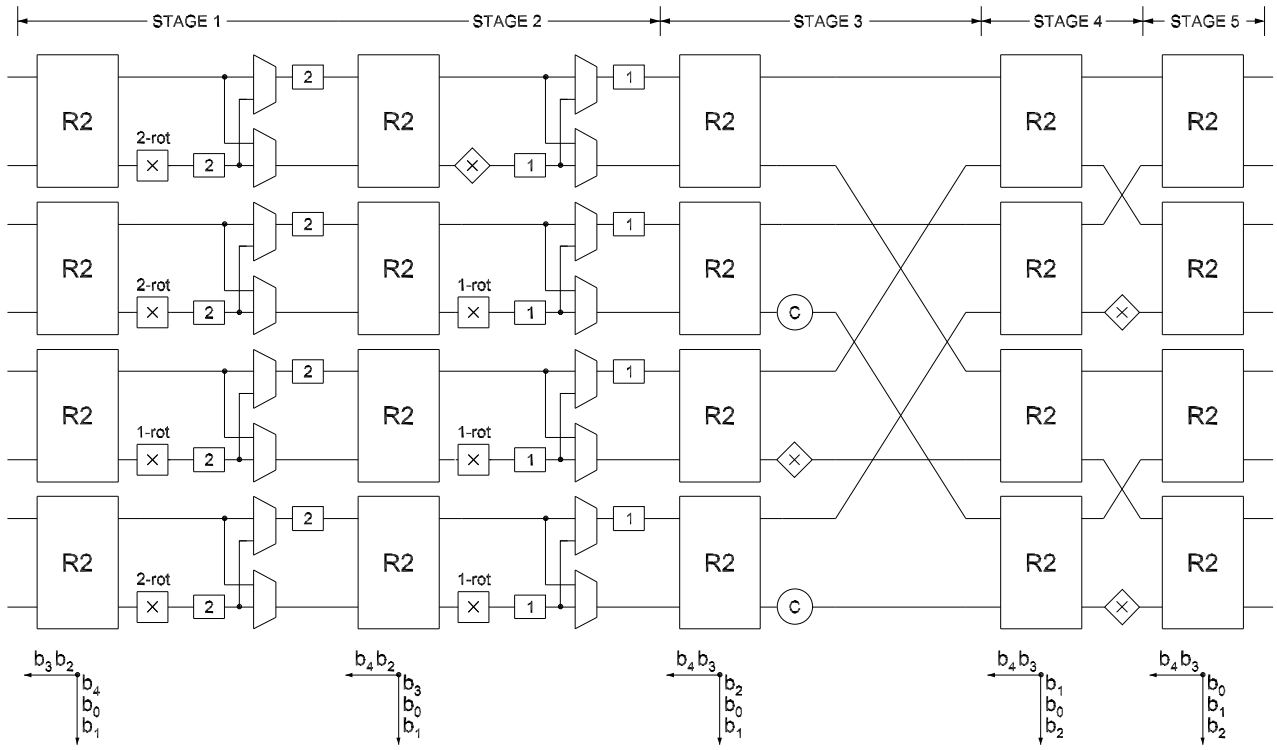

Fig. 6. Proposed 32-point 8-parallel radix-2 MDC DIF FFT architecture.

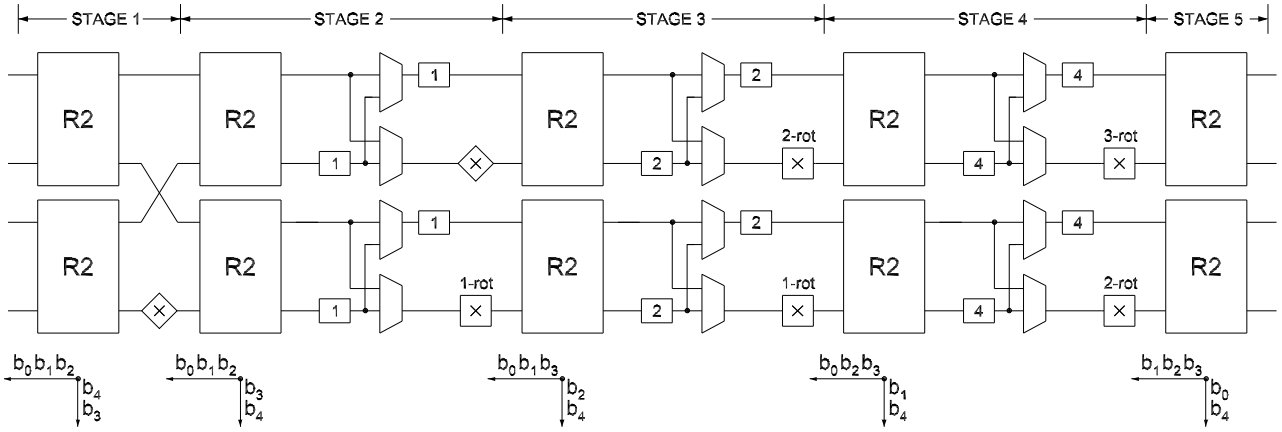

Fig. 7. Proposed 32-point 4-parallel radix-2 MDC DIT FFT architecture.

4-parallel architecture. Therefore, the FFT hardware does not duplicate when duplicating the number of samples in parallel. Only the number of butterflies is duplicated. The memory is reduced and the rotators are almost duplicated in number, but simplified in complexity.

Figure 7 shows the proposed 4-parallel radix-2 FFTs for decimation in time (DIT) [29]. It can be noted that this architecture is symmetric with respect to the DIF one: Whereas for the DIF case the length of the buffers and the complexity of the rotators decrease with the number of the stage, for the DIT case it is the opposite. As a result, the amount of hardware resources for the DIF and DIT versions is the same. The 8-parallel radix-2 DIT FFT is obtained in a similar way.

For larger N, any N-point P-parallel radix-2 DIF FFT architectures based on rotator allocation is obtained by adding one extra stage at the beginning of the $N/2$-point P-parallel radix-2 DIF architecture, as can be observed when comparing the 32-point 4-parallel radix-2 FFT in Fig. 5 and the 16-point 4-parallel radix-2 FFT in Fig 4(d). Analogously, for the DIT case any N-point P-parallel DIT FFT is obtained by adding one extra stage at the end of the $N/2$-point P-parallel radix-2 DIT architecture. In both DIF and DIT cases, further stages added to the 32-point FFT require general rotators. Thus, the 64-point radix-2 FFT requires the same rotators as the 32-point radix-2 FFT plus two general rotators. For any radix-2 4-parallel (DIF or DIT) FFT based on rotator

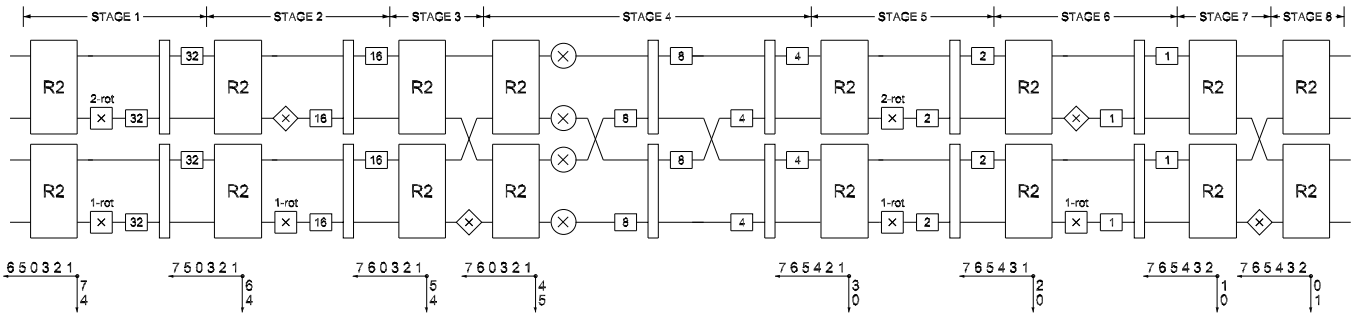

Fig. 8. 256-point 4-parallel radix-2^4 MDC DIF FFT architecture.

TABLE V
ROTATORS IN THE PROPOSED 4-PARALLEL RADIX-2 MDC FFT ARCHITECTURES

FFT Algorithm		Rotators			
N	Radix	General	3-rot	2-rot	1-rot
4	2	0	0	0	0
8	2	0	0	0	1
16	2	0	0	1	2
32	2	0	1	2	2
64	2	2	1	2	2
128	2	4	1	2	2
256	2	6	1	2	2
512	2	8	1	2	2
1024	2	10	1	2	2
2048	2	12	1	2	2
4096	2	14	1	2	2

TABLE VI
ROTATORS IN THE PROPOSED 8-PARALLEL RADIX-2 MDC FFT ARCHITECTURES

FFT Algorithm		Rotators			
N	Radix	General	3-rot	2-rot	1-rot
8	2	0	0	0	2
16	2	0	0	0	5
32	2	0	0	3	6
64	2	2	1	4	6
128	2	6	1	4	6
256	2	10	1	4	6
512	2	14	1	4	6
1024	2	18	1	4	6
2048	2	22	1	4	6
4096	2	26	1	4	6

allocation, the type and number of rotators are shown in Table V. For the 8-parallel case, the type and number of rotators are shown in Table VI. Both in the 4-parallel and 8-parallel architectures, the number of general rotators increases with N. It would be desired that the additional rotators are simpler than a general rotator. This motivates the use of radix-2^k instead of radix-2, as shown in the next section.

B. Radix-2^k Feedforward FFTs Based on Rotator Allocation

Radix-2^k feedforward FFTs are obtained by combining the radix-2 FFTs in previous section. For instance, the architecture in Fig. 8 is a 256-point 4-parallel radix-2^4 DIF FFT. It consists of two 16-point FFTs connected by the rotators and the shuffling circuits in stage 4. The FFT algorithm used in this architecture is represented by the binary tree in Fig. 9(i).

TABLE VII
PROPOSED 4-PARALLEL RADIX-2^k MDC FFT ARCHITECTURES

FFT Algorithm			Rotators				Eqiv.
N	Radix	Tree	Gen.	3-rot	2-rot	1-rot	Adders
SELECTED ARCHITECTURES							
4	2	(a)	0 / 0	0	0	0	16
8	2	(b)	0 / 0	0	0	1	36
16	2	(c)	0 / 0	0	1	2	68.5
32	2	(d)	0 / 0	1	2	2	108.5
64	2^3	(f)	0 / 4	0	0	2	173
128	2^4,2^3	(h)	0 / 4	0	1	3	205.5
256	2^4	(i)	0 / 4	0	2	4	238
512	2^5,2^4	(k)	0 / 4	1	3	4	278
1024	2^5	(m)	0 / 4	2	4	4	318
2048	2^4,2^4,2^3	(o)	0 / 8	0	2	5	375
4096	2^4	(q)	0 / 8	0	3	6	407.5
OTHER ALTERNATIVES WITH HIGHER HW COST							
64	2	(e)	2 / 0	1	2	2	179
128	2	(g)	4 / 0	1	2	2	249.5
256	2^5,2^3	(j)	0 / 4	1	2	3	245.5
512	2^3	(l)	0 / 8	0	0	3	310
2048	2^6,2^5	(n)	2 / 4	2	4	4	385.5
4096	2^6	(p)	4 / 4	2	4	4	435
4096	2^3	(r)	0 / 12	0	0	4	447

The 16-point FFTs are the same as that in Fig. 4(d), with the only difference in the length of the buffers of the first 16-point FFT. The connection stage has 4 general rotators. This is the same for all radix-2^k FFTs presented next. As a result, the radix-2^k feedforward FFTs have $P/2 \cdot \log_2 N$ butterflies, a total memory size $N - P$, and the rotators of the radix-2 sub-FFTs plus P general rotators per connection stage.

Table VII presents the proposed radix-2^k feedforward FFTs based on rotator allocation for 4-parallel samples. The table is divided into *Selected architectures* and *Other alternatives with higher hardware cost.* The selected architectures are the ones that achieve smallest hardware cost. The first two columns of the table show the number of points, N, and the radix. The third column indicates the FFT algorithms used in these architectures, which are represented by the binary trees in Fig. 9. Note that the binary tree provides the values of the rotation angles at each FFT stage [35]. The fourth to the seventh columns show the number of rotators that the architecture requires. The general rotators are separated into two groups: The general rotators in the radix-2 sub-FFTs and

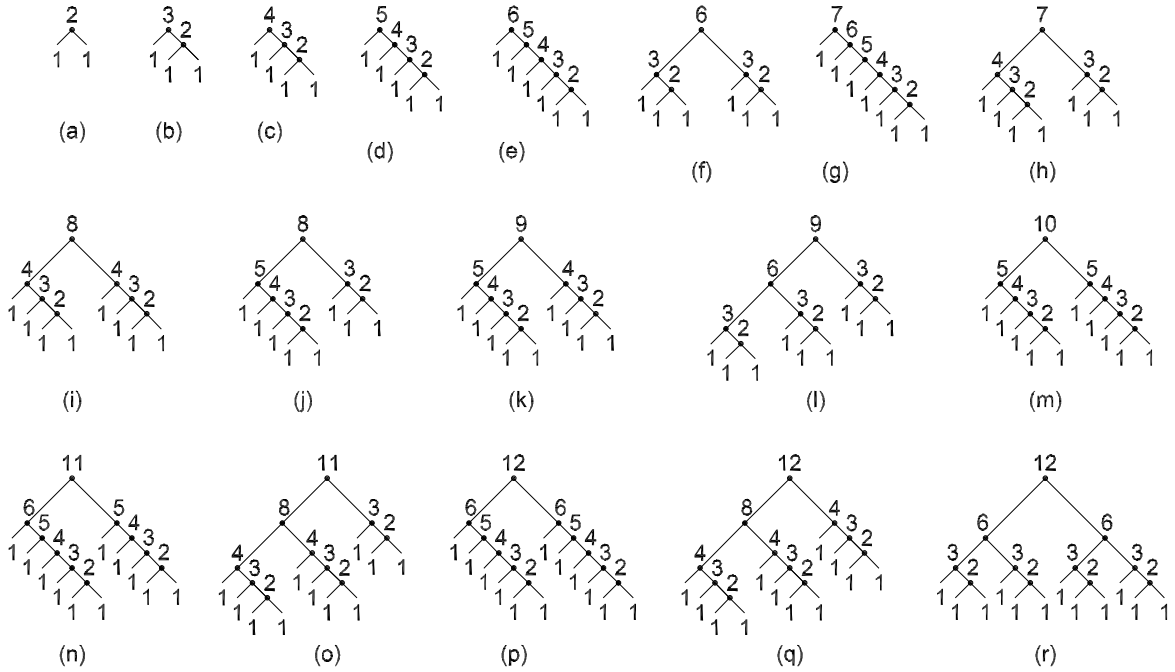

Fig. 9. Binary trees of the proposed feedforward FFTs.

the general rotators used to interconnect sub-FFTs. Finally, the last column indicates the total number of equivalent real adders of the entire architecture, including the adders of the butterflies and rotators, and the equivalent number of real adders of the multiplexers:

$$\text{Equiv. Adders} = 2P \cdot \log_2 N + \text{AddersRot} + 2P/4 \cdot \log_2(N/P), \quad (5)$$

where the adders for the rotators (AddersRot) are computed according to the estimations in Table III.

In Table VII, radix-2^k starts to get better results than radix-2 for $N = 64$, whereas radix-2 is preferable for sizes smaller than or equal to 32.

Analogously, Table VIII shows the proposed radix-2^k feedforward FFTs based on rotator allocation for 8-parallel samples. In this case, radix-2^k starts to obtain better results than radix-2 for $N = 128$.

TABLE VIII

PROPOSED 8-PARALLEL RADIX-2^k MDC FFT ARCHITECTURES

FFT Algorithm			Rotators				Eqiv.
N	Radix	Tree	Gen.	3-rot	2-rot	1-rot	Adders
SELECTED ARCHITECTURES							
8	2	(b)	0 / 0	0	0	2	68
16	2	(c)	0 / 0	0	0	5	118
32	2	(d)	0 / 0	0	3	6	185.5
64	2	(e)	2 / 0	1	4	6	296
128	2^4,2^3	(h)	0 / 8	0	0	7	392
256	2^4	(i)	0 / 8	0	0	10	442
512	2^5,2^4	(k)	0 / 8	0	3	11	509.5
1024	2^5	(m)	0 / 8	0	6	12	577
2048	2^6,2^5	(n)	2 / 8	1	7	12	687.5
4096	2^4	(q)	0 / 16	0	0	15	766
OTHER ALTERNATIVES WITH HIGHER HW COST							
64	2^3	(f)	0 / 8	0	0	4	342
128	2	(g)	6 / 0	1	4	6	437
256	2^5,2^3	(j)	0 / 8	0	3	9	469.5
512	2^3	(l)	0 / 16	0	0	6	616
2048	2^4,2^4,2^3	(o)	0 / 16	0	0	12	716
4096	2^6	(p)	4 / 8	2	8	12	798
4096	2^3	(r)	0 / 24	0	0	6	870

VII. COMPARISON

Table IX compares the proposed architectures to previous ones in terms of complex adders and rotators. The table includes the rotators used in previous architectures as a function of $n = \log_2(N)$. For the proposed architectures there is no generalized expression. Instead, the number of rotators is provided by the *Selected architectures* in Tables VII and VIII. For a better comparison of the number of rotators, Table IX includes the examples of $N = 1024$ and $N = 4096$. The architectures are sorted out by the exponent $k = 0, \ldots, 4$ of the radix-2^k. It can be observed that radix-2 architectures lead to the largest number of complex rotators, and this number decreases as we increase k until radix-2^4. For $N = 1024$ the proposed architectures reduce this number even further, as they require half the general rotators of previous radix-2^4 FFT architectures, both for 4-parallel and 8-parallel designs. Note also that when not using rotator allocation, a 1024-point FFT that uses radix-2^5 requires 8 general rotators (4 W_{1024} and 4 W_{32}), 4 3-rot and 4 2-rot, and a 1024-point FFT that uses radix-2^4 [16] requires 8 general rotators (4 W_{1024} and 4 W_{32}) and 6 3-rot, as shown in Table IX. Thus, radix-2^4 and a radix-2^5 FFTs without rotator allocation are comparable in area. However, when using rotator allocation, these numbers are improved noticeably.

TABLE IX
COMPARISON OF PIPELINE PARALLEL FFT ARCHITECTURES IN TERMS OF COMPLEX ADDERS AND ROTATORS

PIPELINED ARCHITECTURE	AREA											
	Complex Adders	Rotators			Rotators (N=1024)				Rotators (N=4096)			
		General	3-rot	2-rot	Gen.	3-rot	2-rot	1-rot	Gen.	3-rot	2-rot	1-rot
4-PARALLEL ARCHITECTURES												
R2 MDC, [19]	$4n+4$	$2n-4$	0	0	16	0	0	0	20	0	0	0
R4 MDC, [4]	$4n$	$3\lceil n/2\rceil-3$	0	0	12	0	0	0	15	0	0	0
R4 MDC, [15]	$4n$	$3\lceil n/2\rceil-3$	0	0	12	0	0	0	15	0	0	0
$R2^2$ MDC, [16]	$4n$	$3\lceil n/2\rceil-3$	0	0	12	0	0	0	15	0	0	0
$R2^3$ MDC, [16]	$4n$	$4\lceil n/3\rceil-4$	0	$2\lfloor n/3\rfloor$	12	0	4	0	12	0	8	0
$R2^4$ MDF, [6]	$4n$	$4\lceil (n-2)/4\rceil-1$	$4\lfloor (n-2)/4\rfloor$	0†	7	8	0	0	11	8	0	0
$R2^4$ MDF, [8]	$8n$	$4\lceil n/4\rceil-4$	$4\lfloor n/4\rfloor$	0††	8	8	0	0	8	12	0	0
$R2^4$ MDF, [12]	$8n$	$4\lceil n/4\rceil-4$	$4\lfloor n/4\rfloor$	0††	8	8	0	0	8	12	0	0
$R2^4$ MDC, [16]	$4n$	$4\lceil n/4\rceil-4$	$3\lfloor n/4\rfloor$	0⋆	8	6	0	0	8	12	0	0
Proposed, MDC	$4n$	Table VII (Selected architectures)			4	2	4	4	8	0	3	6
8-PARALLEL ARCHITECTURES												
R2 MDC, [36]	$8n$	$4n-8$	0	0	32	0	0	0	40	0	0	0
R2 MDF, [37]	$16n$	$4n-8$	0	0	32	0	0	0	40	0	0	0
$R2^2$ MDC, [16]	$8n$	$6\lceil n/2\rceil-6$	0	0	24	0	0	0	30	0	0	0
R8 MDC, [4]	$8n$	$7\lceil n/3\rceil-7$	0	$2\lfloor n/3\rfloor$	21	0	6	0	21	0	8	0
$R2^3$ MDC, [16]	$8n$	$7\lceil n/3\rceil-7$	0	$2\lfloor n/3\rfloor$	21	0	6	0	21	0	8	0
$R2^4$ MDF, [6]	$8n$	$8\lceil (n-3)/4\rceil-1$	$8\lfloor (n-3)/4\rfloor$	2†	15	8	6	0	23	16	2	0
$R2^4$ MDF, [7]	$16n$	$8\lceil n/4\rceil-8$	$8\lfloor n/4\rfloor$	0††	16	16	0	0	16	24	0	0
$R2^4$ MDC, [16]	$8n$	$8\lceil n/4\rceil-8$	$6\lfloor n/4\rfloor$	0⋆	16	12	0	0	16	18	0	0
Proposed, MDC	$8n$	Table VIII (Selected architectures)			8	0	6	12	16	0	0	15

†: It requires 2 additional 2-rot for 4-parallel if $\text{mod}(n-2,4)=3$, and 4 2-rot for 8-parallel if $\text{mod}(n-3,4)=3$, to calculate the W_8 rotations.
††: If $\text{mod}(n,4)=3$, it requires 1 2-rot for 4-parallel and 2 2-rot for 8-parallel to calculate the W_8 rotations.
⋆: If $\text{mod}(n,4)=3$, it requires 2 2-rot for 4-parallel and 4 2-rot for 8-parallel to calculate the W_8 rotations.

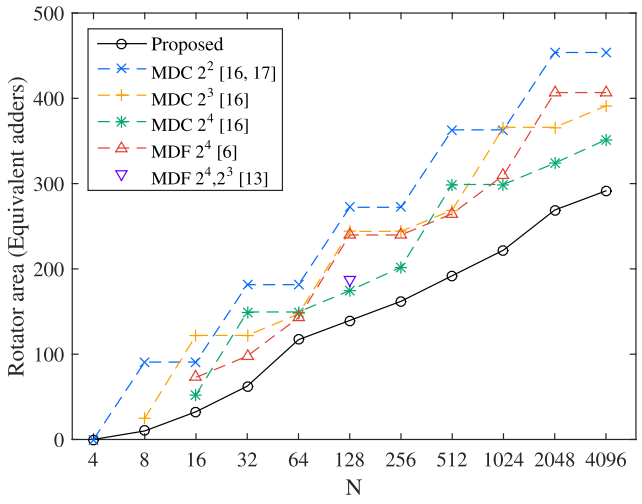

Fig. 10. Rotator area of 4-parallel feedforward FFT architectures in terms of equivalent adders.

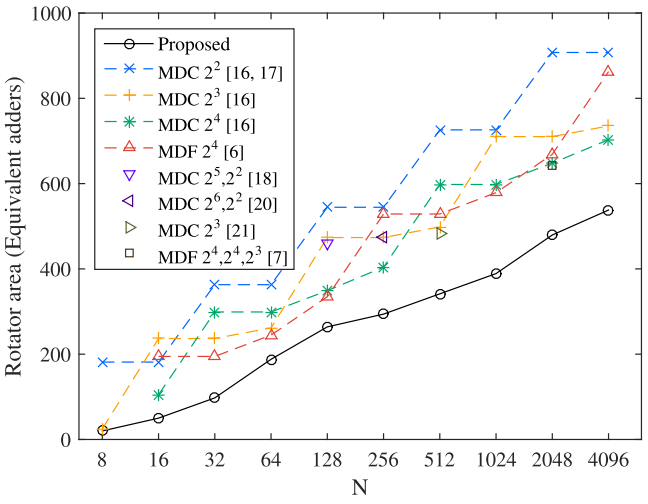

Fig. 11. Rotator area of 8-parallel feedforward FFT architectures in terms of equivalent adders.

For the case of $N=4096$ in Table IX, the number of general rotators of the proposed architectures is the same as those in previous radix-2^4 FFTs. However, the number of non-general non-trivial rotators and their complexity is reduced significantly by using the proposed approach.

The total area occupied by the rotators is compared in Figs. 10 and 11 for 4-parallel and 8-parallel FFTs, respectively. The area is measured in terms of equivalent adders, considering the adder cost of each type of rotator according to Table III. Figs. 10 and 11 show that the rotator area of the proposed architectures is smaller than that in previous approaches for all FFT sizes. For 4-parallel architectures, the rotator area reduction ranges from 17% to 36%. For 8-parallel architectures, it ranges from 23% for to 50%. This highlights the significant improvements of using the rotator allocation approach.

For the area of the entire FFT architecture, Figs. 12 and 13 compare the proposed feedforward FFTs based on rotator

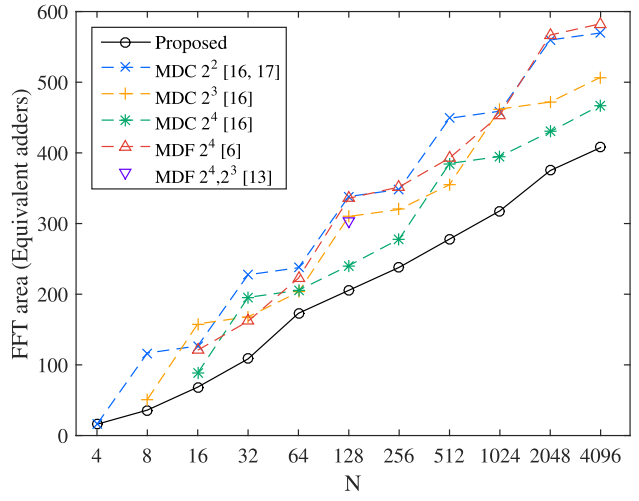

Fig. 12. FFT area of 4-parallel feedforward FFT architectures in terms of equivalent adders (including butterflies, rotators and multiplexers and excluding memories).

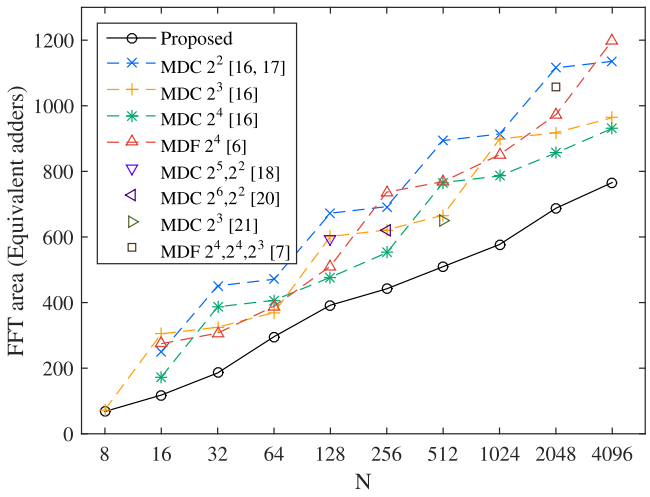

Fig. 13. FFT area of 8-parallel feedforward FFT architectures in terms of equivalent adders (including butterflies, rotators and multiplexers and excluding memories).

allocation with previous feedforward FFT architectures for 4-parallel and 8-parallel, respectively. In all the architectures the total memory is the same, $N - P$. Therefore, the comparison is done in terms of equivalent adders of rotators, butterflies and shuffling circuits. For the proposed architectures, the number of equivalent adders comes from Tables VII and VIII, and equation (5). For the previous architectures, we calculate the number of equivalent adders in the same way as for the proposed architectures, i.e., considering the equivalent adders in butterflies and shuffling circuits, and an optimized implementation of the rotators according to Table III.

Figs. 12 and 13 show that the proposed architectures reduce the number of equivalent adders with respect to previous approaches. For 4-parallel and $N = 4096$ points, the savings of using rotator allocation are around 15% with respect to radix-2^4 [16], and 40% with respect to radix-2^2 [16], [17]. For 8-parallel and $N = 4096$, the savings are around 22% with respect to radix-2^4 architectures [16], and 48% with respect to radix-2^2 [16], [17].

TABLE X
COMPARISON OF 4-PARALLEL 1024-POINT FFTS ON FPGAS

FFT Parametes	Ours14 [16], [17]	Glittas16 [19]	Wang16 [6]	Spiral17 [26]	Proposed -
N	1024	1024	1024	1024	1024
P	4	4	4	4	4
Radix	2^2	2	2^2	2	2^5
Word lenth	16	16	16	16	16
FPGA	Virtex6	Virtex5	Virtex6	Virtex6	Virtex6
Slices	1341	4116	4359	1443	1420
BRAM	12	0	10	44	12
DSP slices	48	72	45	64	16
Clk (MHz)	227	380	305	289	253
Th (MS/s)	910	1520	1220	1157	1012

VIII. IMPLEMENTATION

In order to verify the advantages of the proposed architectures, we have implemented the proposed 1024-point 4-parallel radix-2^5 DIF FFT based on rotator allocation, shown in Table VII. The input data word length is chosen to be 16 bits. Similar to the case of the 256-point 4-parallel radix-2^4 MDC DIF FFT in Fig. 8, the 1024-point 4-parallel radix-2^5 DIF FFT can be derived by cascading two 32-point MDC DIF FFTs as that in Fig. 5, changing the length of the buffers of one of them, and including rotators and shuffling circuits between both of them.

A. FPGA Results

Table X compares 4-parallel 1024-point FFT architectures on FPGAs. The references to previous works include the first author and the year of the work. For instance, 'Wang16' means 'Wang, 2016'. The table includes the area in terms of slices, BRAM and DSP slices, and the performance in terms of clock frequency (Clk) and throughput (Th). The proposed results are obtained for a Virtex-6 XC6VSX475T-1-FF1156 FPGA. The comparison shows that the proposed architecture reduces by 64% or more the number of DSP slices with respect to previous approaches. This is due to the fact that the proposed architecture has only 4 general rotators, which are implemented in the DSP slices. The rest of rotators are implemented as distributed logic (slices). This, however, does not increase significantly the amount of distributed logic: The proposed architecture requires approximately the same distributed logic as [16], [17], which also require 12 BRAM.

Figure 14 shows the area comparison as a bar diagram. It can be observed that our approach saves a significant amount of DSP slices used for rotators with respect to previous approaches, while keeping a small number of slices and BRAM.

Figure 15 compares the throughput versus the FPGA utilization for 1024-point FFTs on FPGAs. The FPGA utilization is calculated as the percentage of FPGA hardware resources used of a a Virtex-6 XC6VSX475T-1-FF1156 FPGA, according to the definition in [17].

Fig. 15 includes both serial FFTs (1P) and parallel FFTs (2P, 4P, 8P). The architectures improve as they increase in throughput and/or decrease in FPGA utilization, towards the upper left corner. Both the throughput and the FPGA utilization increase with the parallelization of the FFT.

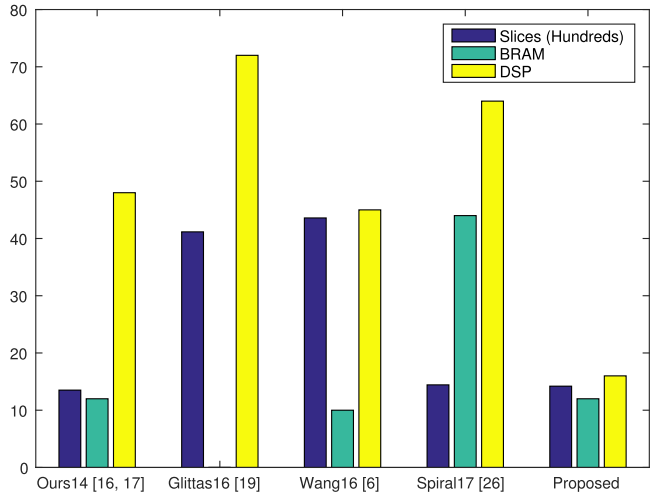

Fig. 14. Area comparison of 4-parallel 1024-point FFTs on FPGAs.

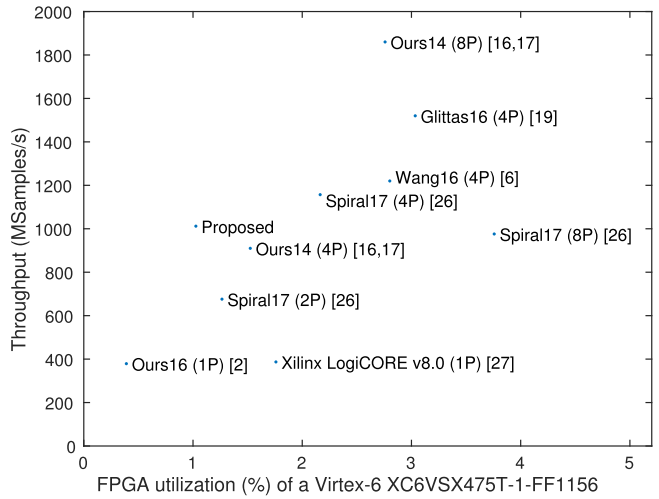

Fig. 15. Throughput vs. FPGA utilization of 1024-point FFT hardware architectures.

Serial FFTs [2], [27] appear in the lower left corner and 8-parallel FFTs appear towards the upper right one. The proposed FFT provides a good trade-off between throughput and resources. Its FPGA utilization is 1.03% and the throughput is 1012 MS/s. This is 33% less area and 11% more throughput than the closest FFT [16], [17]. Furthermore, it has significantly smaller FPGA utilization than previous parallel FFTs, while keeping a high throughput.

B. ASIC Results

Table XI shows the ASIC results of the proposed FFT in two different versions: Low power and high performance. The difference between these two versions is that the high performance approach includes additional pipelining. The experimental results show that the low power version only requires 8.88 mW at 180 Mhz. The high performance version calculates the 1024-point FFT in 0.83 μs at 1.28 GS/s. In both versions, the SQNR [38], [39] is 40.3 dB, which meets the SQNR needed in applications such as UWB [18], Wi-Fi [18] and gigabit WPAN [40]. Finally, the area of the proposed FFTs is small, around 0.2 mm^2. This meets the main purpose of the paper, i.e., achieving a small area for the FFT.

TABLE XI
PROPOSED 4-PARALLEL 1024-POINT FFTS ON ASICS

FFT Parametes	Proposed (low power)	Proposed (high performance)
N	1024	1024
P	4	4
Radix	2^5	2^5
Word length	16	16
Technology (nm)	55	55
Voltage (V)	0.9	0.9
Clk (MHz)	180	320
Th (MS/s)	720	1280
Latency (cycles)	263	265
Latency (μs)	1.46	0.83
Area (mm^2)	0.198	0.212
SQNR (dB)	40.3	40.3
Power (mW)	8.88 @ 180 MHz	17.02 @ 320 MHz

IX. CONCLUSIONS

In this paper, we have presented the rotation allocation approach. It consists in finding an efficient distribution of FFT rotations that reduces the number of rotators and their complexity. This leads to new radix-2 and radix-2^k MDC FFT architectures. These architectures require the same memory and butterflies as previous MDC FFTs, but fewer and/or simpler rotators. For 4-parallel FFTs, the area of rotators is reduced by a factor from 17% to 36% with respect to previous approaches. For 8-parallel FFTs, the rotator area reduction ranges from 23% to 50%. These savings are confirmed with the implementation of a 4-parallel 1024-point radix-2^5 MDC FFT based on rotation allocation. Experimental results on FPGAs show significant reduction in area with respect to previous 4-parallel 1024-point FFTs, with 64% less DSP slices than previous FFT architectures. This leads to 33% less total FFT area. Experimental results on ASICs lead to a very small area, together with low power consumption or high performance.

ACKNOWLEDGMENT

The authors would like to thank Dr. Martin Kumm for his help with the calculation of the adder cost of the rotators.

REFERENCES

[1] 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.

[2] M. Garrido, R. Andersson, F. Qureshi, and O. Gustafsson, "Multiplier-less unity-gain SDF FFTs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 9, pp. 3003–3007, Sep. 2016.

[3] S. He and M. Torkelson, "Design and implementation of a 1024-point pipeline FFT processor," in *Proc. IEEE Custom Integr. Circuits Conf.*, May 1998, pp. 131–134.

[4] M. Sänchez, M. Garrido, M. López, 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.

[5] A. Cortés, I. Vélez, and J. F. Sevillano, "Radix r^k FFTs: Matricial representation and SDC/SDF pipeline implementation," *IEEE Trans. Signal Process.*, vol. 57, no. 7, pp. 2824–2839, Jul. 2009.

[6] 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.

[7] 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. I, Reg. Papers*, vol. 57, no. 6, pp. 451–455, Jun. 2010.

[8] H. Liu and H. Lee, "A high performance four-parallel 128/64-point radix-2^4 FFT/IFFT processor for MIMO-OFDM systems," in *Proc. IEEE Asia–Pacific Conf. Circuits Syst.*, Nov. 2008, pp. 834–837.

[9] 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.

[10] 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.

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

[12] 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.

[13] S.-I. Cho and K.-M. Kang, "A low-complexity 128-point mixed-radix FFT processor for MB-OFDM UWB systems," *ETRI J.*, vol. 32, no. 1, pp. 1–10, Feb. 2010.

[14] W. Xudong and L. Yu, "Special-purpose computer for 64-point FFT based on FPGA," in *Proc. Int. Conf. Wireless Commun. Signal Process.*, Nov. 2009, pp. 1–3.

[15] K.-J. Yang, S.-H. Tsai, and G. C. H. Huang, "MDC FFT/IFFT processor with variable length for MIMO-OFDM systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 4, pp. 720–731, Apr. 2013.

[16] M. Garrido, J. Grajal, M. A. Sänchez, 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.

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

[18] M. G. Kim, S. K. Shin, and M. H. Sunwoo, "New parallel MDC FFT processor with efiicient scheduling scheme," in *Proc. IEEE Asia–Pacific Conf. Circuits Syst.*, Nov. 2014, pp. 667–670.

[19] 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.

[20] J. K. Jang, M. G. Kim, and M. H. Sunwoo, "Efficient scheduling scheme for eight-parallel MDC FFT processor," in *Proc. Int. SoC Design Conf. (ISOCC)*, Nov. 2015, pp. 277–278.

[21] 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.

[22] M. Garrido, S.-J. Huang, S.-G. Chen, and O. Gustafsson, "The serial commutator FFT," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 63, no. 10, pp. 974–978, Oct. 2016.

[23] Y. W. Lin and C. Y. Lee, "Design of an FFT/IFFT processor for MIMO OFDM systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 4, pp. 807–815, Apr. 2007.

[24] S. Li, H. Xu, W. Fan, Y. Chen, and X. Zeng, "A 128/256-point pipeline FFT/IFFT processor for MIMO OFDM system IEEE 802.16e," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jun. 2010, pp. 1488–1491.

[25] M. Garrido, "Efficient hardware architectures for the computation of the FFT and other related signal processing algorithms in real time," Ph.D. dissertation, Dept. Signals, Syst. Radiocommun., Univ. Politécnica de Madrid, Madrid, Spain, Dec. 2009.

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

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

[28] 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, 1965.

[29] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, 1989.

[30] 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.

[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] M. Garrido, P. Källström, M. Kumm, and O. Gustafsson, "CORDIC II: A new improved CORDIC algorithm," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 63, no. 2, pp. 186–190, Feb. 2016.

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

[34] J. Chen and C. H. Chang, "High-level synthesis algorithm for the design of reconfigurable constant multiplier," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 28, no. 12, pp. 1844–1856, Dec. 2009.

[35] F. Qureshi and O. Gustafsson, "Generation of all radix-2 fast Fourier transform algorithms using binary trees," in *Proc. Eur. Conf. Circuit Theory Design*, Aug. 2011, pp. 677–680.

[36] J. A. Johnston, "Parallel pipeline fast Fourier transformer," *IEE Proc. F Commun. Radar Signal Process.*, vol. 130, no. 6, pp. 564–572, Oct. 1983.

[37] E. H. Wold and A. M. Despain, "Pipeline and parallel-pipeline FFT processors for VLSI implementations," *IEEE Trans. Comput.*, vol. C-33, no. 5, pp. 414–426, May 1984.

[38] W. H. Chang and T. Q. Nguyen, "On the fixed-point accuracy analysis of FFT algorithms," *IEEE Trans. Signal Process.*, vol. 56, no. 10, pp. 4673–4682, Oct. 2008.

[39] I. B. Dhaou, "Hardware architecture for an anti-traffic noise system," *Microelectron. J.*, vol. 46, no. 5, pp. 370–376, May 2015.

[40] Y. Chen, Y. C. Tsao, Y. W. Lin, C. H. Lin, and C. Y. Lee, "An indexed-scaling pipelined FFT processor for OFDM-based WPAN applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 2, pp. 146–150, Feb. 2008.

Mario Garrido (M'07) received the M.S. degree in electrical engineering and the Ph.D. degree from the Technical University of Madrid, 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 has been an Associate Professor since 2012.

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, 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 and designs for small area and low power consumption.

Shen-Jui Huang received the M.S. degree from the Department of Electrical Engineering, National Taiwan University, in 1997, and the Ph.D. degree in electronics from National Chiao Tung University in 2012. He is currently a Deputy Manager with Novatek Corporation, Hsinchu, Taiwan. His research interests include baseband signal processing and circuit implementation.

Sau-Gee Chen received the B.S. degree from National Tsing Hua University, Taiwan, in 1978, and the M.S. degree and the Ph.D. degree in electrical engineering from the State University of New York at Buffalo, Buffalo, NY, USA, in 1984 and 1988, respectively. He was the Director of the Institute of Electronics, National Chiao Tung University (NCTU), Taiwan, from 2003 to 2006. He was the Director of the Honors Program at the College of Electrical and Computer Engineering/College of Computer Science, NCTU, from 2011 to 2012. He was the Associate Dean with the Office of International Affairs in 2011. He was the Department Chair of the Department of Electronics Engineering, NCTU, from 2012 to 2015. He is currently a Professor with the Department of Electronics Engineering, NCTU. He has authored over 100 conference and journal papers, and holds 19 U.S. and Taiwan patents. His research interests include digital communication, multi-media computing, digital signal processing, and VLSI signal processing. He is a member of the Board of Governor, IEEE Taipei Section. He has been appointed as the Coordinator of the IEEE VTS Asia-Pacific Chapters, since 2014. He was the Chair of the IEEE Vehicular Technology Society, Taipei Chapter, from 2012 to 2013. From 2004 to 2006, he served as an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I.