Using Rotator Transformations to Simplify FFT Hardware Architectures

Rikard Andersson and Mario Garrido, *Senior Member, IEEE*

***Abstract*— In this paper, we present a new approach to simplify fast Fourier transform (FFT) hardware architectures. The new approach is based on a group of transformations called decimation, reduction, center, move and merge. By combining them it is possible to transform the rotators at different FFT stages, move them to other stages and merge them in such a way that the resulting rotators are simpler than the original ones. The proposed approach can be combined with other existing techniques such coefficient selection and shift-and-add implementation, or rotator allocation in order to obtain low-complexity FFT hardware architectures. To show the effectiveness of the proposed approach, it has been applied to single-path delay feedback (SDF) FFT hardware architectures, where it is observed that the complexity of the rotators is reduced up to 33%.**

***Index Terms*— Center, decimation, fast Fourier transform (FFT), merge, move, pipelined architecture, reduction, shift-and-add implementation, twiddle factor.**

I. Introduction

IN TODAY'S digital signal processing world, there exists a need for converting signals between time and frequency domains. For this reason, the fast Fourier transform (FFT) has become one of the most important algorithms in the field.

To calculate the FFT in hardware, various architectures have been proposed. Among them, the single-path delay feedback (SDF) [1]–[11] FFT is one of the most popular ones, as it processes data in pipeline with high throughput and using a small amount of hardware resources. In an SDF FFT, each stage consists of a butterfly, a buffer and a rotator. The butterfly calculates additions and subtractions, the buffer delays the data and the rotator rotates the data in the complex plane. At each time instant, the rotation angle may be different. Thus, the rotator must be capable of rotating by different angles. The set of all these angles is called *twiddle factor*.

As a rotation is an expensive operation in terms of hardware resources, different techniques to implement rotators have been proposed with the goal of reducing the hardware cost. The most straightforward approach is to implement the rotator by using a complex multiplier, which consists of four real multipliers and two additions. A more efficient alternative is to implement the rotator as shift-and-add operations. The coordinate rotation digital computer (CORDIC) [12]–[15] algorithm exploits this idea and calculates the rotation by a series of micro-rotation stages that include a small number of adders. An alternative to the CORDIC is to simplify a complex multiplier into shift-and-add operations. This approach is useful for rotators that have to rotate among a small number of angles. In this approach, the multiplications by the real and imaginary parts of the rotation coefficient are transformed into shift-and-add operations by using single constant multiplication (SCM) [16], [17], multiple constant multiplication (MCM) [18]–[22] or constant matrix multiplication (CMM) [23]–[25] techniques. Then, the rotator is created by merging the shift-and-add multiplications by the different coefficients [26], [27]. Finally, rotator allocation [28] is another existing technique to simplify FFT rotators. It is used in parallel FFT architectures to distribute the rotations efficiently among the parallel branches.

This paper proposes a new approach to simplify rotators in FFT hardware architectures based on a set of transformations called decimation, reduction, center, move and merge. These techniques take advantage of three main ideas. First, a rotator can be split into several rotators of smaller complexity. Second, by extracting a constant angle, the angles of a rotator can be rotated in order to place them in a more favorable position. And third, constant rotations can be moved among FFT stages and merged with other rotators. All these ideas are combined in the proposed approach, leading to simpler rotators. Furthermore, contrary to rotator allocation [28], the proposed approach does not alter the data order in the architecture. Finally, the proposed approach is compatible with previous approaches used to simplify rotators. This allows for further complexity reduction when combining several approaches.

The paper is organized as follows. In Section II, we review previous concepts related to the FFT that are later used in this paper. In Section III, we present the proposed approach as a set of transformations that are applied to rotators. In Section IV, we show the application of the proposed approach to simplify rotators in FFT hardware architectures. In Section V, we show the hardware implementation of one of these architectures. In Section VI, we compare the proposed approach to previous approaches. Finally, in Section VII, we summarize the main conclusions of the paper.

Manuscript received March 30, 2020; revised June 16, 2020; accepted June 27, 2020. Date of publication July 9, 2020; date of current version December 1, 2020. This work was supported by the Spanish Ministry of Science, Innovation and Universities under the Ramón y Cajal Grant RYC2018-025384-I. This article was recommended by Associate Editor P. K. Meher. *(Corresponding author: Mario Garrido.)*

Rikard Andersson is with the Department of Electrical Engineering, Linköping University, 58183 Linköping, Sweden (e-mail: rikan842@gmail.com).

Mario Garrido is with the Department of Electronic Engineering, ETSI de Telecomunicación, Universidad Politécnica de Madrid, 28040 Madrid, Spain (e-mail: mario.garrido@upm.es).

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

Digital Object Identifier 10.1109/TCSI.2020.3006253

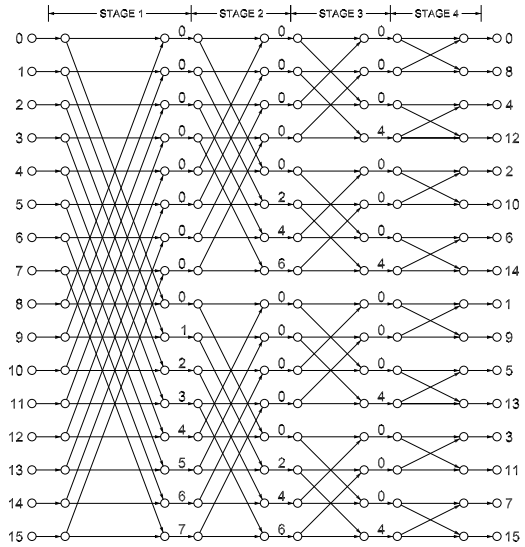


Fig. 1. Flow graph of a 16-point radix-2 DIF FFT.

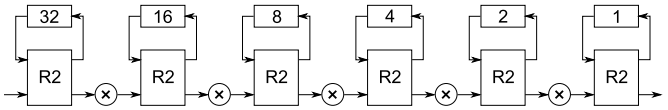


Fig. 2. A 64-point SDF FFT architecture.

II. Background

A. The FFT Algorithm

The FFT algorithm is used to reduce the computation complexity of the discrete Fourier transform (DFT) from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log_2 N)$ operations [29]. Fig. 1 shows the flow graph of a 16-point radix-2 FFT decomposed by using decimation in frequency (DIF) [1], [30]. The FFT consists of $n = \log_2 N$ stages. At each stage of the flow graph, $s \in \{1, \ldots, n\}$, butterflies and rotations are calculated. The lower edges of the butterflies are always multiplied by -1. These -1 are not depicted in order to simplify the graphs.

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

$$W_N^{\phi} = e^{-j\frac{2\pi}{N}\phi}. \tag{1}$$

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

B. The SDF FFT Architecture

The FFT algorithm allows for a variety of hardware architectures. When high throughput and small area are required, the SDF FFT is an attractive choice. Fig. 2 shows a 64-point radix-2 SDF FFT architecture. It consists of $n = \log_2(N) = \log_2(64) = 6$ stages and each of them includes a butterfly (R2), a buffer of length 2^{n-s} and a rotator.

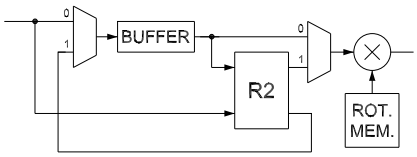


Fig. 3. Internal structure of a stage in an SDF FFT architecture.

The internal structure of a stage in an SDF FFT is shown in Fig. 3. First, the buffer collects 2^{n-s} data. Then, these data are added and subtracted with the incoming data in the butterfly. The results of the additions pass to the rotator, while the results of the subtractions are fed back to the buffer. The latter are passed to the rotator after the former, at the same time that new data arrive. This allows for continuous flow processing.

It is worth noting that all data pass through the rotator. This means that the same rotator calculates all the rotations at the same stage of the flow graph in Fig. 1 and, therefore, the rotator must be configurable to rotate by different angles at different clock cycles.

C. Twiddle Factors

In FFT architectures that use conventional FFT algorithms [31], each rotator in an FFT architecture rotates by a twiddle factor. A twiddle factor, W_L, is a set of L angles evenly distributed along the circumference, i.e.,

$$W_L = W_L^{\phi} = e^{-j\frac{2\pi}{L}\phi}, \qquad \text{for} \quad \phi = 0 \ldots L-1. \tag{2}$$

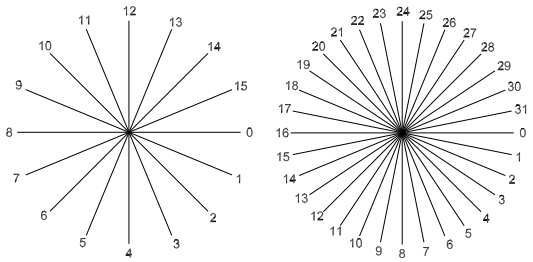
For the FFT architecture in Fig. 2, if the radix-2 DIF algorithm is used, the twiddle factors at stages 1 to 5 in are W_{64}, W_{32}, W_{16}, W_8 and W_4, respectively. For radix-2^2 DIF, the twiddle factors are W_4, W_{64}, W_4, W_{16} and W_4, and for radix-2^3 they are W_4, W_8, W_{64}, W_4 and W_8. Note that radix-2^k divides the n stages of the FFT in $\lceil n/k \rceil$ groups, where $\lceil n/k \rceil - 1$ of them calculate a 2^k-point FFT and one of them calculates the FFT of the remaining stages.

D. Symmetric Angle Sets

Some of the angles in a twiddle factor are symmetric with respect to 0°, 45°, 90° or 135°. Based on these symmetries, a symmetric angle set (SAS) [32] is defined as 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/or an exchange of the real and imaginary parts of the rotation coefficient. Fig. 4 shows the angles used in a W_{16} twiddle factor ($\phi = 0, \ldots, 15$) and in a W_{32} twiddle factor ($\phi = 0, \ldots, 31$), which correspond to dividing the circumference into 16 and 32 equal parts, respectively. These angles form several SAS, which are shown in Tables I and II.

E. M-Rotator

The smaller the number of SAS in a rotator, the smaller the number of independent coefficients. To quantize this number,



(a) (b)

Fig. 4. Rotation angles ϕ in twiddle factors. (a) W_{16}. (b) W_{32}.

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	

an *M-rotator* or *M-rot* [28] is defined as a rotator that can rotate a 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 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 5 SAS in Table II). In general, a twiddle factor W_L is an $L/8+1$-rot. This means that there are $L/8+1$ angles in the range $[0, \pi/4]$ and the rest of the angles can be obtained by symmetries.

III. PROPOSED APPROACH

The proposed approach consists of a group of transformations that can be applied to the rotators in FFT hardware architectures. These transformations are called decimation, reduction, center, move and merge, and are explained in the steps of Sections III-A to III-F. These steps suggest the order in which the transformations must be applied.

A. Step 1: Decimate Twiddle Factors

Decimation transforms a twiddle factor W_L into a $W_{L/2}$ in series with a 2-rot according to

$$W_L = W_{L/2} \cdot e^{-j\frac{2\pi}{L}\phi_0}, \tag{3}$$

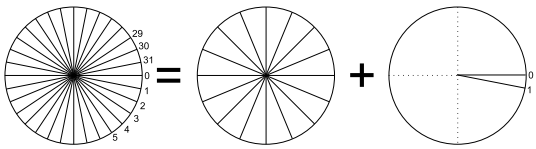


Fig. 5. Decimation of a W_{32} into a W_{16} and a 2-rot.

which is equal to

$$e^{-j\frac{2\pi}{L}\phi_L} = e^{-j\frac{2\pi}{L/2}\phi_{L/2}} \cdot e^{-j\frac{2\pi}{L}\phi_0}, \tag{4}$$

where

$$\phi_L = 2\phi_{L/2} + \phi_0, \tag{5}$$

for $\phi_L = \{0, 1, \ldots, L-1\}$, $\phi_{L/2} = \{0, 1, \ldots, L/2-1\}$ and $\phi_0 = \{0, 1\}$.

The specific case of decimating a W_{32} is shown in Fig. 5. It can be observed that the W_{32} is decimated into a W_{16} plus a 2-rot.

For any L, decimation halves the number of angles of the original twiddle factor W_L, which is an $(L/8+1)$-rot, providing a $W_{L/2}$, which is an $(L/16+1)$-rot, and a 2-rot. As a result, the combination of the the twiddle factor $W_{L/2}$ and the 2-rot calculates the same rotations as the original twidddle factor W_L.

Decimation can be applied repeatedly to obtain a simpler twiddle factor in each iteration. By applying decimation twice, a W_L is transformed into a $W_{L/4}$ plus two 2-rots according to

$$W_L = W_{L/4} \cdot e^{-j\frac{2\pi}{L}\phi_0} \cdot e^{-j\frac{2\pi}{L}\phi_1}, \tag{6}$$

which is equal to

$$e^{-j\frac{2\pi}{L}\phi_L} = e^{-j\frac{2\pi}{L/4}\phi_{L/4}} \cdot e^{-j\frac{2\pi}{L}\phi_0} \cdot e^{-j\frac{2\pi}{L}\phi_1}, \tag{7}$$

where

$$\phi_L = 4\phi_{L/4} + \phi_0 + \phi_1, \tag{8}$$

for $\phi_L = \{0, 1, \ldots, L-1\}$, $\phi_{L/4} = \{0, 1, \ldots, L/4-1\}$, $\phi_0 = \{0, 1\}$ and $\phi_1 = \{0, 2\}$.

In general, the decimation of a W_L twiddle factor m times results in a $W_{L/2^m}$ twiddle factor in series with m 2-rots by $\phi_i = \{0, 2^i\}$ for $i = 0, \ldots, m-1$. This way, an $(L/8+1)$-rot is transformed into an $(L/(8 \cdot 2^m)+1)$-rot plus m 2-rots.

B. Step 2: Reduce Twiddle Factors

The aim of reduction is to decrease the number of SAS in a twiddle factor by extracting a constant angle from the rotator. Given a twiddle factor W_L, the extraction of a constant angle ϕ_0 results in

$$W_L^{\phi_L} = e^{-j\frac{2\pi}{L}(\phi_L+\phi_0)} \cdot e^{j\frac{2\pi}{L}\phi_0}, \quad \phi_L = 0, \ldots, L-1. \tag{9}$$

If $\phi_0 = 1/2$, this leads to

$$\begin{aligned} e^{-j\frac{2\pi}{L}\phi_L} &= e^{-j\frac{2\pi}{L}\left(\phi_L+\frac{1}{2}\right)} \cdot e^{j\frac{2\pi}{L}\left(\frac{1}{2}\right)} = \\ &= e^{-j\frac{2\pi}{2L}(2\phi_L+1)} \cdot e^{j\frac{2\pi}{2L}}, \end{aligned} \tag{10}$$

being

$$\phi_{2L[\text{odd}]} = 2\phi_L + 1. \tag{11}$$

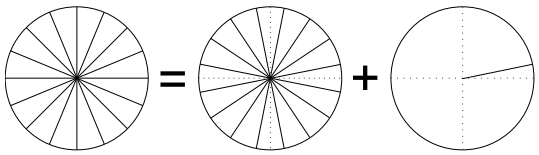


Fig. 6. Reduction of a W_{16} twiddle factor. The rotator to the left is a W_{16} twiddle factor. It is transformed into a 2-rot plus a constant rotator.

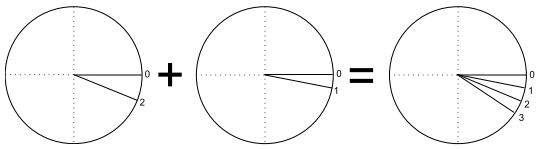


Fig. 7. Example of merging two 2-rots into a 4-rot.

As $\phi_L = \{0, 1, \ldots, L-1\}$, then $\phi_{2L[\text{odd}]} = \{1, 3, 5, \ldots, 2L-1\}$, which corresponds to the odd coefficients of the twiddle factor W_{2L}. This allows to represent the reduction transformation as

$$W_L = W_{2L[\text{odd}]} \cdot e^{j\frac{2\pi}{2L}}. \tag{12}$$

As a result, reduction transforms the twiddle factor W_L, which is an $L/8+1$-rot, into the cascade of two rotators. These rotators are $W_{2L[\text{odd}]}$, which is an $L/8$-rot, and a constant rotator by the angle $\alpha = 2\pi/2L$. Fig. 6 shows the reduction of a W_{16} twiddle factor. This rotator is a 3-rot, which is reduced to a 2-rot and a constant rotator.

Additionally, any constant angle of the form $\alpha = 2\pi/2L + n2\pi/L$, where $n \in \mathbb{Z}$, can be extracted when reducing a twiddle factor. The result is also a $W_{2L[\text{odd}]}$ plus the constant rotator, with the particularity that the angles of $W_{2L[\text{odd}]}$ will be rotated by $n2\pi/L$. This rotation does not change the angles in the $W_{2L[\text{odd}]}$.

C. Step 3: Merge 2-Rots

The 2-rots generated by decimation (step 1) can be merged together into larger rotators. For two 2-rots we get

$$e^{-j\frac{2\pi}{L}\phi_p} \cdot e^{-j\frac{2\pi}{L}\phi_q} = e^{-j\frac{2\pi}{L}\phi_r}, \tag{13}$$

with $\phi_p = \{0, 2^p\}$, $\phi_j = \{0, 2^q\}$, and $\phi_r = \{0, 2^p, 2^q, 2^p+2^q\}$. This case is shown in Fig. 7.

In general, m 2-rots generated by decimation can be merged together to create a single 2^m-rot.

D. Step 4: Center Merged 2^m-Rots

The 2^m-rots obtained from merging 2-rots (step 3) have the property that the angles start in 0° and continue clockwise or counterclockwise, covering a range smaller than 45°. Due to this, all the 2^m angles of the rotator are in a different SAS. This is why the rotator is a 2^m-rot. In this context, we can center the rotator in order to half the number of SAS. This is achieved by extracting the constant angle that lies in the center of the set. For a set of merged 2-rots where each of them corresponds to $\phi_i = \{0, 2^i\}$, the angle in the center corresponds to

$$\phi_C = \frac{1}{2}\sum_i 2^i. \tag{14}$$

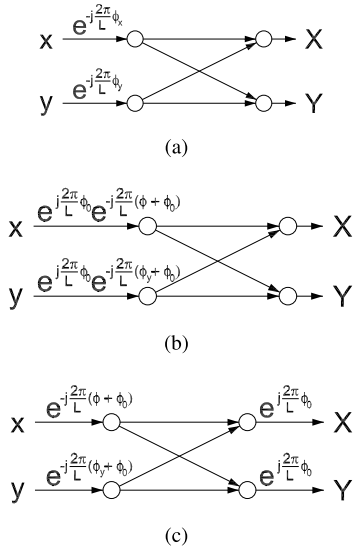


Fig. 8. Movement of an extracted angle. (a) Original butterfly with rotations at a given stage. (b) Structure after extracting $e^{j\frac{\pi}{L}}$. (c) Structure after moving the extracted angle to the next stage.

For instance, for $i = \{0, 3, 4\}$,

$$\phi_C = \frac{1}{2}(2^0 + 2^3 + 2^4) = 12.5. \tag{15}$$

The centered rotator has a symmetry with respect to 0° and each positive angle has a symmetric negative angle. Therefore, centering the rotator halves the number of SAS, i.e., a non-centered 2^m-rot is transformed into a centered 2^{m-1}-rot.

E. Step 5: Move Constant Angles

Both reduction (step 2) and center (step 4) produce constant rotators as a result of the transformation. The advantage of constant angles is that they can be moved freely between FFT stages. To move a rotator from a certain stage to an adjacent stage, we have to pass the butterfly in the middle. Let us consider a butterfly with a rotation previous to it, as shown in Fig. 8(a). The equation of the structure in Fig. 8(a) is

$$\begin{aligned} X &= x \cdot e^{-j\frac{2\pi}{L}\phi_x} + y \cdot e^{-j\frac{2\pi}{L}\phi_y}, \\ Y &= x \cdot e^{-j\frac{2\pi}{L}\phi_x} - y \cdot e^{-j\frac{2\pi}{L}\phi_y}. \end{aligned} \tag{16}$$

By extracting $e^{j\frac{2\pi}{L}\phi_0}$, we obtain Fig. 8(b), which calculates

$$\begin{aligned} X &= x \cdot e^{-j\frac{2\pi}{L}(\phi_x+\phi_0)} \cdot e^{j\frac{2\pi}{L}\phi_0} + y \cdot e^{-j\frac{2\pi}{L}(\phi_y+\phi_0)} \cdot e^{j\frac{2\pi}{L}\phi_0}, \\ Y &= x \cdot e^{-j\frac{2\pi}{L}(\phi_x+\phi_0)} \cdot e^{j\frac{2\pi}{L}\phi_0} - y \cdot e^{-j\frac{2\pi}{L}(\phi_y+\phi_0)} \cdot e^{j\frac{2\pi}{L}\phi_0}. \end{aligned} \tag{17}$$

Finally, by moving $e^{j\frac{2\pi}{L}\phi_0}$ to the next stage, we obtain Fig. 8(c), which calculates

$$\begin{aligned} X &= e^{j\frac{2\pi}{L}\phi_0} \cdot (x \cdot e^{-j\frac{2\pi}{L}(\phi_x+\phi_0)} + y \cdot e^{-j\frac{2\pi}{L}(\phi_y+\phi_0)}), \\ Y &= e^{j\frac{2\pi}{L}\phi_0} \cdot (x \cdot e^{-j\frac{2\pi}{L}(\phi_x+\phi_0)} - y \cdot e^{-j\frac{2\pi}{L}(\phi_y+\phi_0)}). \end{aligned} \tag{18}$$

As a result, the constant angle is moved to the next stage. Alternatively, the constant angle can be moved to the previous stage. As all data of the stage are multiplied by the factor $e^{j\frac{2\pi}{L}\phi_0}$, the move operation can be repeated several times. This allows for placing the constant rotator at any stage of the FFT.

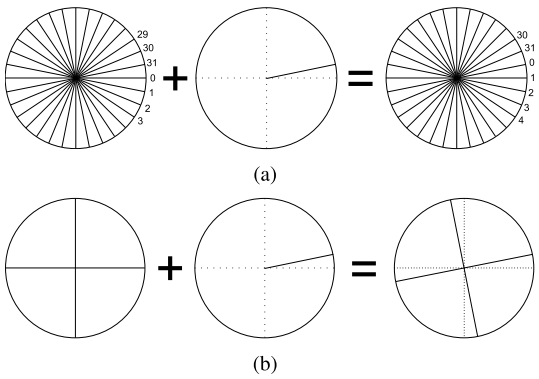


Fig. 9. Examples for merging a constant rotator with a twiddle factor. (a) W_{32} absorbing the angle $\frac{\pi}{16}$ at no cost. Only the control changes. (b) Complication of a W_4 when merging it with the angle $\frac{\pi}{16}$.

F. Step 6: Merge Constant Angles

1) Merging a Constant Rotation With a Twiddle Factor: An extracted constant angle can be merged with the rotators in the stage where it has been moved. If this stage contains a twiddle factor

$$W_L = e^{-j\frac{2\pi}{L}\phi} \quad \phi = 0, \ldots, L-1, \tag{19}$$

the result of merging it with a constant rotator by $e^{j\frac{2\pi}{L}\phi_0}$ is

$$e^{-j\frac{2\pi}{L}\phi} \cdot e^{j\frac{2\pi}{L}\phi_0} = e^{-j\frac{2\pi}{L}(\phi-\phi_0)} \quad \phi = 0, \ldots, L-1. \tag{20}$$

When this method is applied, it is important that merging the rotators does not increase the total amount of SAS. This depends on the value of the term ϕ_0. If $\phi_0 \geq 1$ is an integer number, the rotation angles of the merged twiddle factor are the same as those of W_L. This case is shown in Fig. 9(a), where a constant angle is absorbed by the twiddle factor without increasing its complexity.

If $\phi_0 = 1/2$, the rotator W_L is reduced, as explained in Section III-B. This not only eliminates the constant angle, but also simplifies the twiddle factor.

Finally, in case ϕ_0 does not meet any of the previous conditions, the merged rotator becomes more complicated. An example of this is shown in Fig. 9(b), where merging the angle $\frac{\pi}{16}$ with a W_4 complicates the W_4.

Note that, in general, it is a good practice to move the extracted constant angles to a stage where there exists a large rotator, as large rotators can generally absorbe the constant angles without increasing their complexity.

2) Merging a Constant Rotation With an M-Rot: A second case is to merge a constant rotation with an M-rot. For instance, merging a constant rotation by $e^{-j\frac{2\pi}{L}\phi_0}$ with a 2-rot with $\phi_a = \{0, a\}$ results in

$$e^{-j\frac{2\pi}{L}\phi_a} \cdot e^{-j\frac{2\pi}{L}\phi_0} = e^{-j\frac{2\pi}{L}\phi_b}, \quad \phi_b = \{\phi_0, a+\phi_0\}. \tag{21}$$

Fig. 10 shows an example where merging simplifies the M-rot. In this case, a constant rotation by the angle $e^{-j\frac{\pi}{16}}$ is merged with a 4-rot. This reduces the amount of SAS, as $\phi = 1$ and $\phi = 31$ are in the same SAS. As a result, the 4-rot is simplified to a 3-rot.

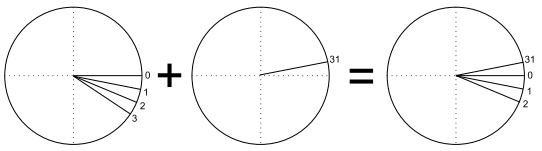


Fig. 10. Example of merging a constant rotator with an M-rot, where a 4-rot is merged with a constant angle $e^{-j\frac{\pi}{16}}$. As a result, the 4-rot is simplified to a 3-rot.

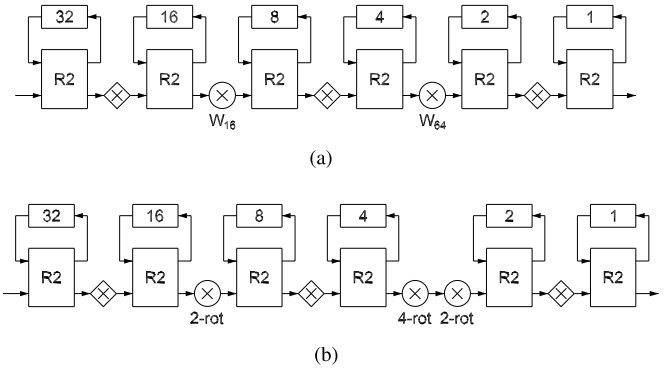


Fig. 11. Simplification of a 64-point radix-2^4 SDF FFT by using the proposed methods. (a) Original architecture. (b) Simplified architecture.

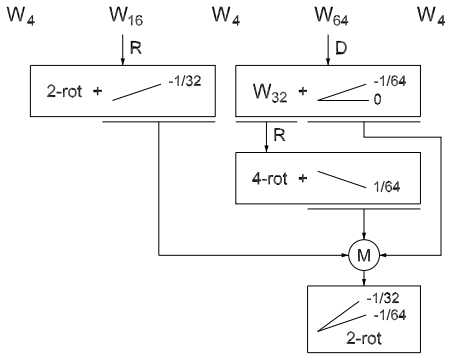


Fig. 12. Transformation of the rotators in a 64-point radix-2^4 FFT.

IV. Application of the Proposed Method

This section shows how to apply the proposed approach to simplify radix-2^4 SDF FFT architectures with sizes 64 to 1024.

A. Transforming a 64-Point FFT

Figure 11(a) shows a 64-point radix-2^4 SDF FFT. It includes a W_{16} in stage 2, a W_{64} in stage 4 and a W_4 in stages 1, 3 and 5.

The rotators in the architecture are transferred to the top of Fig. 12. This figure shows the transformations that are carried out. In order to identify the transformation, letters corresponding to the different transformations are added to the figure: Decimation (D), reduction (R), merge (M). As the figure includes twiddle factors of different sizes, the angles are indicated as ϕ/L. This represents the ϕ-th angle in a circumference divided in L parts, where ϕ increases clockwise. This criterion is used in Figs. 12 to 16.

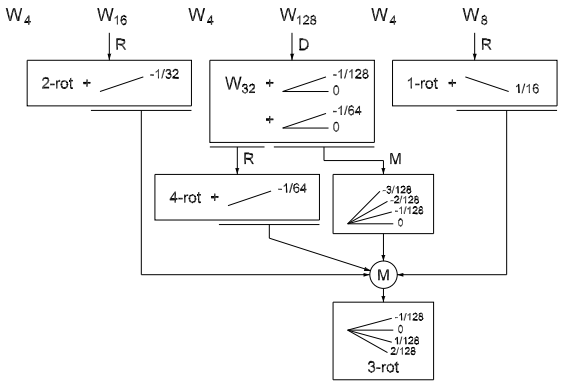


Fig. 13. Transformation of the rotators in a 128-point radix-2^4 FFT.

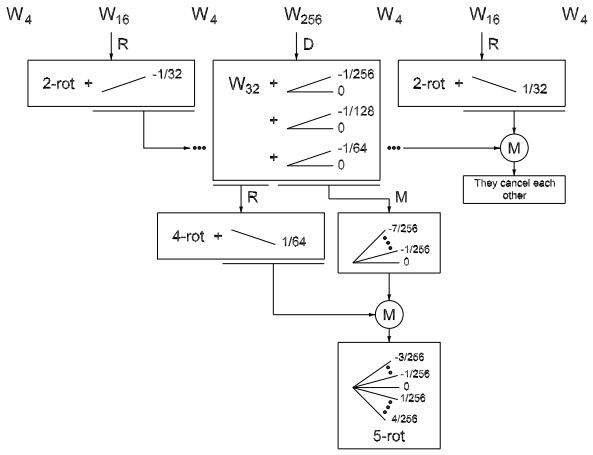


Fig. 14. Transformation of the rotators in a 256-point radix-2^4 FFT.

To start the transformation, decimation is applied to the W_{64} rotator. This turns it into a W_{32} in series with a 2-rot with the angles $\{0, -1/64\}$. Then, the W_{32} and the W_{16} are reduced by extracting the angles $1/64$ and $-1/32$, respectively. This reduces the W_{16} from a 3-rot down to a 2-rot with the angles $\{1/32, 3/32\}$ in $[0, -45°]$, and the W_{32} from a 5-rot to a 4-rot with the angles $\{1/64, 3/64, 5/64, 7/64\}$ in $[0, -45°]$. Next, the constant angles are moved to stage 4, where they are merged with the 2-rot resulting from decimation. This changes its angle set to $\{0, -1/64\} - 1/32 + 1/64 = \{-1/32, -1/64\}$.

Fig. 11(b) shows the resulting architecture. It requires a 2-rot in stage 2, and a 4-rot and a 2-rot in stage 4.

From this example, it is worth noting that the the decimation of W_{64} has provided a 2-rot that later is used to absorb all the constant rotations. Indeed, while applying the proposed approach it is a good practice to decide and keep in mind which rotator in the architecture is going to absorb the constant rotators.

B. Transforming a 128-Point FFT

Figure 13 describes the transformation of the rotators in a 128-point FFT. The original twiddle factors correspond to radix-2^4, where the first 4 stages calculate a 16-point FFT and the last three stages calculate an 8-point FFT.

The first step in the transformation consist in applying decimation to the W_{128} rotator. In this case, we extract two 2-rots to decimate the twiddle factor down to a W_{32}. This is due to the fact that a W_{32} is a 5-rot, which is a reasonably small rotator. This W_{32} is then reduced to a 4-rot plus a constant rotation. Likewise, the W_{16} and the W_8 are reduced to a 2-rot plus a constant rotation by $-1/32$, and a 1-rot plus a constant rotation by $1/16$. Next, the 2-rots obtained by decimating the W_{128} are merged together to form a 4-rot. Finally, this 4-rot absorbs the constant rotators obtained in the reductions of the three twiddle factors, leading to a rotator by $\{0, -1/128, -2/128, -3/128\} + (-1/32) + 1/16 + (-1/64) = \{2/128, 1/128, 0, -1/128\}$. The result is a 3-rot, as it includes 4 angles where 2 of them correspond to the same SAS. As a result, the rotators in the 128-point FFT are simplified to a 4-rot, a 3-rot, a 2-rot and a 1-rot.

With this example we want to highlight the importance of the selection of the extracted constant angles in the reduction process. In this case, some of the extracted angles are positive, whereas other ones are negative. This is due to the fact that this selection results in a simple rotator when these angles are merged. Conversely, if the reduction of the W_8 had been done with the angle $-1/16$, this would have changed the final rotator into $\{0, -1/128, -2/128, -3/128\} + (-1/32) + (-1/16) + (-1/64) = \{-14/128, -15/128, -16/128, -17/128\}$, which is a 4-rot instead of a 3-rot.

C. Transforming a 256-Point FFT

Figure 14 shows the transformation of the rotators in a 256-point FFT. The original twiddle factors correspond to the radix-2^4 algorithm.

The first step is to decimate the W_{256} twiddle factor. As in previous examples, the goal is to obtain a W_{32}, which is a relatively simple twiddle factor. This is achieved by extracting three 2-rots. Next, the W_{32} and both W_{16} are reduced. For the W_{16} twiddle factors, the angles are extracted in opposite directions, which makes them cancel each other. Next, the three 2-rots resulting from decimation are merged together into an 8-rot. Finally, this 8-rot is merged with the constant rotation that results from reducing the W_{32} rotator, leading to a 5-rot. As a result, the rotators in the 256-point FFT are simplified to a 5-rot, a 4-rot and two 2-rots.

This example shows the fact that two twiddle factors of the same size W_L, which are $L/8 + 1$-rots, can be reduced by extracting opposite angles that cancel each other. This results into two $L/8$-rots with no additional constant angles.

D. Transforming a 512-Point FFT

Figure 15 shows the transformation of the rotators in a 512-point radix-2^4 FFT. As in previous examples, the largest rotator is first decimated down to a W_{32}. In this case, four 2-rots are obtained apart from the W_{32}. Next, the W_{16}, W_8 and the two W_{32} are reduced and the corresponding constants are extracted. Then, the 2-rots obtained from decimation are separated in two groups of two and each of the groups is

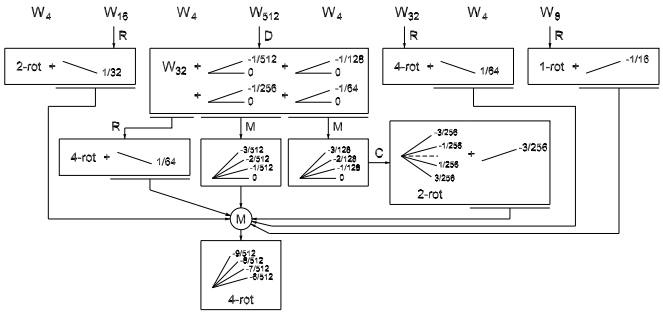


Fig. 15. Transformation of the rotators in a 512-point radix-2^4 FFT.

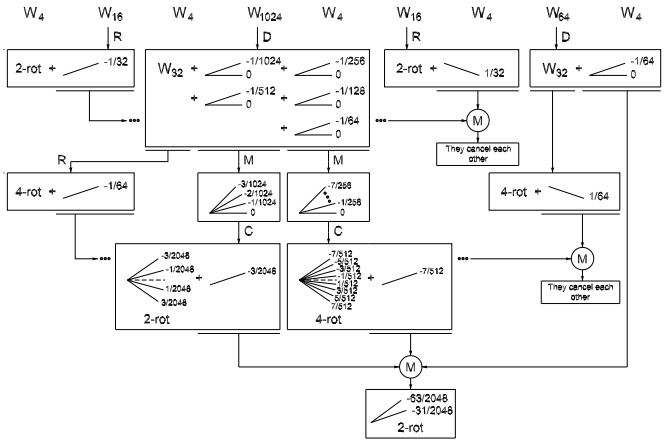


Fig. 16. Transformation of the rotators in a 1024-point radix-2^4 FFT.

merged to form a 4-rot. One of these rotators is then centered (C) to a 2-rot. The other 4-rot is used to absorb all the constants that result from reduction and centering. As a result, the rotators in the 512-point FFT are simplified to three 4-rots, two 2-rots and a 1-rot.

This example has shown how to group the four 2-rots obtained after decimation. By grouping them in pairs, small rotators are obtained. Conversely, the alternative to group the four 2-rots together would have resulted in a 16-rot. This rotator is too complex even after centering it into an 8-rot, which makes this option inefficient. Another observation from this example is that it centers one of the rotators. This step transforms an M-rot into an $M/2$-rot.

E. Transforming a 1024-Point FFT

Figure 16 shows the transformation of the rotators in a 1024-point radix-2^4 FFT. First, the W_{1024} is decimated to a W_{32} plus five 2-rots, and the W_{64} is decimated to a W_{32} and a 2-rot. Then, both W_{32} and both W_{16} are reduced. For each type of twiddle factor, the selection of opposite angles allows for canceling the constants. Next, the five 2-rots obtained from decimating the W_{1024} are divided into a group of two 2-rots and a group of three 2-rots. The rotators in each of these groups are then merged, which results in a 4-rot and an 8-rot. Next, they are centered, which transforms them into a 2-rot plus a constant angle and a 4-rot plus a constant angle, respectively. Finally, the constant angles obtained from centering are merged with the 2-rot that was extracted in the decimation of the W_{64}. As a result, the rotators in the 1024-point FFT are simplified to three 4-rots and four 2-rots.

This example has shown the advantage of having pairs of twiddle factors of the same size. When this occurs, the constants obtained from reduction cancel each other. Likewise, this example has shown the centering of an 8-rot, which results in a 4-rot. Finally, contrary to the 512-point FFT in section IV-D, in the current example both groups of 2-rots obtained from decimating the largest rotator are centered after merging the rotators in each group. This is done because the 2-rot resulting from decimating the W_{64} is reserved to absorb the constant angles. Conversely, in the case of the 512-point FFT one of the two groups must absorb the constant rotations and, therefore, to center it would not have any advantage.

TABLE III
ROTATORS FOR THE PROPOSED 64-POINT FFT

Stage	Rot.	ϕ/L	Angle	Coefficient	WL_E	#Add
2	2-rot	$1/32$ $3/32$	$-\pi/16$ $-3\pi/16$	$60-j12$ $51-j34$	11.0	6
4	4-rot	$1/64$ $3/64$ $5/64$ $7/64$	$-\pi/32$ $-3\pi/32$ $-5\pi/32$ $-7\pi/32$	$515-j51$ $495-j150$ $456-j244$ $400-j328$	12.2	8
4	2-rot	$-1/64$ $-1/32$	$\pi/32$ $\pi/16$	$61+j6$ $60+j12$	11.2	6

TABLE IV
FIGURES OF MERIT OF THE PROPOSED 64-POINT SDF FFT ON A VIRTEX-7 XC7VX330T FPGA

Parameter	Value
N	64
WL (bits)	16+16
Latency (ns)	308
Throughput (MS/s)	250
f_{CLK} (MHz)	250
Slices	556 (1 %)
Slice LUT	1807 (1 %)
Slice FF	1406 (1 %)
DSP Slices	0 (0 %)
BRAM	0 (0 %)

V. IMPLEMENTATION

The proposed 64-point SDF FFT in Fig. 11(b) has been implemented in hardware. The internal structure of each stage is the same as in Fig. 3.

All the rotators have been implemented by using the combined coefficient selection and shift-and-add implementation (CCSSI) method [26]. This method obtains the rotation coefficients for a set of input angles and a minimum effective word length (WL_E). In the implementation, the effective word length of the rotators has been chosen to be $WL_E \geq 11$.

Table III shows the coefficients that are obtained for the three non-trivial rotators of the architecture. The first columns shows the stage of the FFT where the rotator is placed. The second column shows the number of SAS of the rotator. The third and fourth columns show the rotation angles of the

rotators represented as ϕ/L and in radians, respectively. The fifth column shows the coefficients obtained by CCSSI for these angles. And the last two columns show the WL_E and the number of adders of the rotators. It can be observed that the rotators require 6, 8 and 6 adders, respectively, leading to a total of 20 adders for the rotators in the architecture.

A consequence of the shift-and-add implementation of the rotators according to [26] is the removal of the rotation memories. These rotators do not need to store the real and imaginary parts of the coefficient in a memory. Instead, they only require a set of control signals that select the rotation that is carried out at each time instant. These control signals are obtained directly from the control counter $c_5c_4c_3c_2c_1c_0$ of the FFT architecture. Let us consider the case of the 2-rot at stage 4 in Table III. This rotator comes from the rotator by $\{0, -1/64\}$. By applying [30], the rotation by 0 is calculated when $c_5 \cdot c_0 = 0$ and the rotation by $-1/64$ when $c_5 \cdot c_0 = 1$. When a constant angle $-1/64$ is added to the rotator, the 2-rot by $\{-1/64, -1/32\}$ is obtained. This changes the angles and the rotation coefficients, but the control signals do not change. Thus, the control signal that is used to select the rotation angle in the 2-rot at stage 4 is calculated directly from the control counter and the rotator does not need any rotation memory, as an AND gate is enough to obtain the control signal.

Table IV shows post implementation results for the 64-point SDF FFT on a Virtex-7 XC7VX330T FPGA. The design runs at 250 MHz and uses 556 Slices that include 1807 LUTs and 1406 FFs. Furthermore, the proposed 64-point SDF FFT does not require any DSP Slice nor BRAM.

TABLE V
ROTATOR COMPLEXITY FOR DIFFERENT RADICES AND FFT SIZES

Radix	N				
	64	128	256	512	1024
2	1x 9-rot 1x 5-rot 1x 3-rot 1x 2-rot	1x 17-rot 1x 9-rot 1x 5-rot 1x 3-rot 1x 2-rot	1x 33-rot 1x 17-rot 1x 9-rot 1x 5-rot 1x 3-rot 1x 2-rot	1x 65-rot 1x 33-rot 1x 17-rot 1x 9-rot 1x 5-rot 1x 3-rot 1x 2-rot	1x 129-rot 1x 65-rot 1x 33-rot 1x 17-rot 1x 9-rot 1x 5-rot 1x 3-rot 1x 2-rot
	42 Add	64 Add	86 Add	108 Add	130 Add
2^2	1x 9-rot 1x 3-rot	1x 17-rot 1x 5-rot 1x 2-rot	1x 33-rot 1x 9-rot 1x 3-rot	1x 65-rot 1x 17-rot 1x 5-rot 1x 2-rot	1x 129-rot 1x 33-rot 1x 9-rot 1x 3-rot
	30 Add	34 Add	52 Add	56 Add	74 Add
2^3	1x 9-rot 2x 2-rot	1x 17-rot 1x 3-rot 2x 2-rot	1x 33-rot 1x 5-rot 2x 2-rot	1x 65-rot 1x 9-rot 3x 2-rot	1x 129-rot 1x 9-rot 1x 3-rot 3x 2-rot
	30 Add	38 Add	38 Add	56 Add	64 Add
2^4	1x 9-rot 1x 3-rot	1x 17-rot 1x 3-rot 1x 2-rot	1x 33-rot 2x 3-rot	1x 65-rot 1x 5-rot 2x 3-rot	1x 129-rot 1x 9-rot 2x 3-rot
	30 Add	34 Add	38 Add	46 Add	60 Add
Prop.	1x 4-rot 2x 2-rot	1x 4-rot 1x 3-rot 1x 2-rot 1x 1-rot	1x 5-rot 1x 4-rot 2x 2-rot	3x 4-rot 2x 2-rot 1x 1-rot	3x 4-rot 4x 2-rot
	20 Add	28 Add	32 Add	44 Add	52 Add
Savings	33%	18%	15%	4%	13%

VI. COMPARISON

Table V compares the proposed approach to other radices. The table includes the number of M-rots for each of the alternatives. For the cases in the proposed approach, they correspond to the rotators obtained in Section IV. Note that the second column of the table, where a 64-point FFT is considered, corresponds to the architecture in Section V. For radix-2, 2^2 2^3 and 2^4, the type of each rotator is obtained by considering that a W_L rotator is a $L/8 + 1$-rot.

By comparing the M-rots in all the cases, it can be observed that the proposed approach reduces the complexity of the rotators. This is supported by the fact that the largest rotator for the proposed approach is a 5-rot, whereas other radices use larger rotators.

Table V also compares the rotator complexity in terms of the number of adders when the rotators are implemented as shift-and-add according to [26] with an effective word length $WL_E \geq 12$ bits. To achieve this accuracy, the rotators require the following number of adders [32]. For radix-2, 2^2, 2^3 and 2^4, a W_8 uses 4 adders, a W_{16} uses 8 adders, a W_{32} uses 8 adders, and larger rotators require a CORDIC rotator with 11 stages, which uses 22 adders. For the proposed approach, all the rotators require 6 or 8 adders.

The comparison in terms of the number of adders shows that radix-2^4 achieves the best results among radix-2, 2^2, 2^3 and 2^4, and the proposed approach achieves the best results among all the alternatives. The percentage of savings of the proposed approach with respect to radix-2^4 when the rotators are implemented as shift-and-add is reported in the last row of Table V and it ranges from 4% to 33%.

Regarding experimental results, Table VI compares the proposed 64-point SDF FFT to previous 64-point SDF FFT architectures. We have also implemented a 64-point radix-2^4 SDF FFT that uses complex multipliers for the rotations and obtained implementation results for this architecture when using distributed logic and when using DSP slices for the multipliers. The table includes the approach, device, word length (WL), number of slices, look-up tables (LUTs), flip-flops (FFs), block random access memories (BRAMs), digital signal processing (DSP) slices, and clock frequency (f_{CLK}).

The proposed architecture and the implemented radix-2^4 architecture only differ in the rotators, the control for these rotators and the fact that the latter needs rotation memories and the proposed one does not. As a result, the proposed architecture save 233 slices, i.e., 41 % of the area of the entire FFT. Likewise, with respect to [6], the proposed approach saves 30% of the slices.

The equivalence between DSP slices and distributed logic can be obtained by comparing the implemented radix-2^4 FFT with and without DSPs, leading to the fact that each DSP slice is equivalent to 197 LUTs and 183 FFs. By applying this equivalence to [7], it can be deduced that the proposed approach saves around 22% of the area with respect to [7].

The proposed architecture also reduces the number of slices with respect to [2]–[5], although we must be aware that these works use older FPGAs.

TABLE VI
COMPARISON OF 64-POINT SDF FFTS ON FPGAS

Approach	Device (FPGA)	N (points)	Arch. Type	Radix (-)	WL (bits)	Area					f_{CLK} (MHz)
						Slices	LUT	FF	BRAM	DSP Slices	
Kolovos [2]	XCV 300	64	SDF	2^4	16	1566	-	-	-	-	56
Sánchez [3]	XC2V4000	64	SDF	4	16	2452	-	-	0	0	251
Zhou [4]	XC4VSX25	64	SDF	2^2	16	779	-	-	2	8	236
Wang [5]	XC4VLX100	64	SDF	-	16	1303	-	-	0	0	111
Bansal [6]	XC7VX330T	64	SDF	2^2	-	725	-	-	0	0	159
Milovanović [7]	XC7S50	64	SDF	2^2	16	-	1023	689	0	6	100
Radix-2^4	XC7VX330T	64	SDF	2^4	16	789	2474	2066	0	0	250
Radix-2^4 with DSPs	XC7VX330T	64	SDF	2^4	16	280	893	600	0	8	250
Proposed	XC7VX330T	64	SDF	NA	16	556	1807	1406	0	0	250

NA: Not applicable.

Finally, the clock frequency of 250 MHz in the proposed architecture is the highest clock frequency among 64-point SDF FFTs.

VII. CONCLUSIONS

In this paper we have presented an approach to simplify FFT hardware architectures that is based on transforming the rotators in the architecture. These transformations are decimation, reduction, center, move and merge. Decimation transforms a twiddle factor into a simpler twiddle factor plus a 2-rot; reduction extracts a constant angle from a twiddle factor, which simplifies the twiddle factor; center halves the number of SAS by rotating the angles of a 2^m-rot; move is used to move constant rotations among FFT stages; and merge combines extracted rotators into a single one. The use of these transformations leads to simpler rotators in FFT hardware architectures. By comparing the proposed approach to radix-2^3 and 2^4 algorithms, it has been shown that the proposed approach reduces the rotator complexity between 4% and 33% in FFTs from 64 to 1024 points, and the area of a 64-point SDF FFT is reduced 41% with respect to using complex multipliers.

REFERENCES

[1] 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.

[2] P. Kolovos, E. Fotopoulou, and T. Stouraitis, "Comparison of VLSI architectures for a WLAN OFDM transmitter with interpolation filters," in *Proc. 14th IEEE Int. Conf. Electron., Circuits Syst.*, Dec. 2007, pp. 451–454.

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

[4] B. Zhou, Y. Peng, and D. Hwang, "Pipeline FFT architectures optimized for FPGAs," *Int. J. Reconfigurable Comput.*, vol. 2009, pp. 1–9, Sep. 2009.

[5] H.-Y. Wang, J.-J. Wu, C.-W. Chiu, and Y.-H. Lai, "A modified pipeline FFT architecture," in *Proc. Int. Conf. Electr. Control Eng.*, Jun. 2010, pp. 4611–4614.

[6] M. Bansal and S. Nakhate, "High speed pipelined 64-point FFT processor based on radix-22 for wireless LAN," in *Proc. 4th Int. Conf. Signal Process. Integr. Netw. (SPIN)*, Feb. 2017, pp. 607–612.

[7] V. M. Milovanovic and M. L. Petrovic, "A highly parametrizable chisel HCL generator of single-path delay feedback FFT processors," in *Proc. IEEE 31st Int. Conf. Microelectron. (MIEL)*, Sep. 2019, pp. 247–250.

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

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

[10] 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.

[11] A. Cortes, I. Velez, 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.

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

[13] M. Garrido, P. Kallstrom, 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.

[14] P. K. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 9, pp. 1893–1907, Sep. 2009.

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

[16] O. Gustafsson, A. G. Dempster, K. Johansson, M. D. Macleod, and L. Wanhammar, "Simplified design of constant coefficient multipliers," *Circuits, Syst. Signal Process.*, vol. 25, no. 2, pp. 225–251, Apr. 2006.

[17] J. Thong and N. Nicolici, "Time-efficient single constant multiplication based on overlapping digit patterns," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 9, pp. 1353–1357, Sep. 2009.

[18] A. G. Dempster and M. D. Macleod, "Multiplication by two integers using the minimum number of adders," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, May 2005, pp. 1814–1817.

[19] O. Gustafsson, "A difference based adder graph heuristic for multiple constant multiplication problems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 1097–1100.

[20] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algorithms*, vol. 3, no. 2, pp. 1–39, May 2007.

[21] L. Aksoy, E. O. Günes, and P. Flores, "Search algorithms for the multiple constant multiplications problem: Exact and approximate," *Microprocessors Microsyst.*, vol. 34, no. 5, pp. 151–162, Aug. 2010.

[22] M. Kumm, P. Zipf, M. Faust, and C. H. Chang, "Pipelined adder graph optimization for high speed multiple constant multiplication," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2012, pp. 49–52.

[23] A. Dempster, O. Gustafsson, and J. Coleman, "Towards an algorithm for matrix multiplier blocks," in *Proc. Eur. Conf. Circuit Theory Design*, Sep. 2003, pp. 1–4.

[24] O. Gustafsson, H. Ohlsson, and L. Wanhammar, "Low-complexity constant coefficient matrix multiplication using a minimum spanning tree approach," in *Proc. IEEE Nordic Signal Process. Symp.*, Feb. 2004, pp. 141–144.

[25] M. Kumm, M. Hardieck, and P. Zipf, "Optimization of constant matrix multiplication with low power and high throughput," *IEEE Trans. Comput.*, vol. 66, no. 12, pp. 2072–2080, Dec. 2017.

[26] 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.

[27] K. Moller, M. Kumm, M. Garrido, and P. Zipf, "Optimal shift reassignment in reconfigurable constant multiplication circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 3, pp. 710–714, Mar. 2018.

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

[29] 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.

[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] F. Qureshi and O. Gustafsson, "Generation of all radix-2 fast Fourier transform algorithms using binary trees," in *Proc. 20th Eur. Conf. Circuit Theory Design (ECCTD)*, Aug. 2011, pp. 677–680.

[32] R. Andersson, "FFT hardware architectures with reduced twiddle factor sets," M.S. thesis, Dept. Elect. Eng., Linköping Univ., Linköping, Sweden, Jun. 2014.

Rikard Andersson received the M.Sc. degree in applied physics and electronics from Linköping University in 2014. Since then, he has worked as an Application Engineer at SICK IVP AB, Mjärdevi, Linköping. There, he is working with 2D and 3D machine vision solutions for industrial applications.

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

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