Rotators in Fast Fourier Transforms

Fahad Qureshi, Jarmo Takala, and Shuvra Bhattacharyya

Abstract This chapter discusses architectures for computing the rotations in fast Fourier transforms. There are two principal methods, which can be exploited: general complex multipliers or multiplier-less techniques. We describe different architectures, each with different advantages, indicating that the final selection depends on the requirements of the application at hand.

1 Introduction

A rotation is a multiplication by a complex number whose magnitude is one, i.e., a transformation that describes a circular movement with respect to a point [7, 11, 30, 32]. The rotation of a complex number $(x + iy)$ by an angle α can be defined as:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \tag{1}$$

where X and Y are the real and imaginary parts of the result, respectively. Thus, the rotation can be written as:

$$X + iY = (x\cos\alpha - y\sin\alpha) + i(y\cos\alpha + x\sin\alpha). \tag{2}$$

F. Qureshi (✉) · J. Takala
Tampere University, Tampere, Finland
e-mail: fahad@tuni.fi; jarmo.takala@tuni.fi

S. Bhattacharyya (✉)
University of Maryland, College Park, MD, USA

Tampere University, Tampere, Finland
e-mail: ssb@umd.edu

S. S. Bhattacharyya et al. (eds.), *Embedded, Cyber-Physical, and IoT Systems*,
https://doi.org/10.1007/978-3-030-16949-7_11

Let C and S be representations of $\cos\alpha$ and $\sin\alpha$ with a finite number of bits. A rotation of complex number, i.e., a rotation in complex plane, can be presented as

$$\begin{bmatrix} X_Q \\ Y_Q \end{bmatrix} = \begin{bmatrix} C & -S \\ S & C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \tag{3}$$

where X_Q and Y_Q are the result of the rotation, which includes the error resulting from quantization of the coefficients. Many digital signal processing algorithms require to carry out rotations of complex numbers by the given angles with respect to the origin. This is the case with *fast Fourier transforms* (FFT) [26], discrete cosine transforms, and lattice filters [23].

2 Rotations in FFT

The fast Fourier transform is one of the most important tools in digital signal processing. It is based on discrete Fourier transform and used to convert time domain signals to frequency domain [3, 26, 28]. The Fourier transform is defined for continuous signals, while the modern signal processing mainly considers digital systems, and the *discrete Fourier transform* (DFT) is used instead. The DFT transforms a finite sequence of equally spaced samples to a corresponding frequency domain representation as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \qquad k = 0, 1 \ldots, N-1, \tag{4}$$

where N denotes the length of DFT, i.e., the number of points of the DFT, $x[n]$ and $X[k]$ are the input and output samples, respectively. Note that both the signals are discrete in nature. The complex-valued coefficients W_N are called as *twiddle factors* and are defined as

$$W_N = e^{-j2\pi/N} = \cos(2\pi/N) - j\sin(2\pi/N), \tag{5}$$

where j denotes the imaginary unit.

The original signal $x[n]$ can be recovered from $X[k]$ with the aid of an *inverse discrete Fourier transform* (IDFT):

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{kn}, \qquad k = 0, 1, \ldots, N-1. \tag{6}$$

The arithmetic complexity of the DFT in (4) is $O(N^2)$. However, DFT contains redundant computations and several methods have been introduced for avoiding

such a redundancy, thus reducing the complexity. Any algorithm computing DFT with less than $O(N^2)$ complexity is called as a fast Fourier transform. The most popular FFT is the Cooley–Tukey algorithm [3], which uses divide-and-conquer paradigm to decompose DFT into a set of smaller DFTs. Especially, the Cooley–Tukey principle says that an N-point DFT ($N = PQ$) can be computed with the aid of a P-point DFT and a Q-point DFT. By exploiting the periodicity of the twiddle factors:

$$W_N^{kQ} = W_P^k;\ W_N^{kP} = W_Q^k;\ W_N^{kPQ} = 1,$$

the radix-Q FFT can be expressed as:

$$\begin{aligned} X(Qk_1 + k_2) &= \sum_{n_1=0}^{P-1} \sum_{n_2=0}^{Q-1} x\left(n_1 + Pn_2\right) W_P^{n_1 k_1} W_N^{n_1 k_2} W_Q^{n_2 k_2} \\ &= \underbrace{\sum_{n_1=0}^{P-1} \left[\underbrace{\left(\sum_{n_2=0}^{Q-1} x(n_1 + Pn_2) W_Q^{n_2 k_2} \right)}_{Q-point\ DFT} \underbrace{W_N^{n_1 k_2}}_{twiddle factor} \right] W_P^{n_1 k_1}}_{P-point\ DFT}. \end{aligned} \tag{7}$$

The two summations indexed by n_1 and n_2 are referred to as inner and outer DFTs. As a result, an N-point DFT is broken down into P-point and Q-point DFTs. The output of the inner DFT is multiplied by $W_N^{n_1 k_2}$, which is called as a twiddle factor multiplication. The scheme of decomposition is shown in Fig. 1, where the left and right sides represent the P-point inner DFT and Q-point outer DFT, respectively. Between those DFTs, a twiddle factor multiplication indicates a rotation by $W_N^{\phi} = e^{-j\frac{2\pi}{N}\phi}$. The arithmetic complexity is reduced from $O(N^2)$ of the DFT in (4) to $O(N \log N)$.

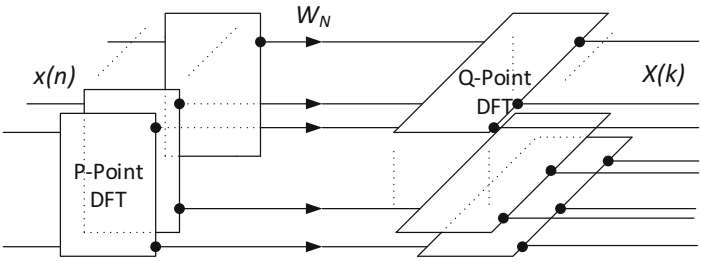

Fig. 1 Decomposition scheme of Cooley–Tukey FFT algorithm

If the length N is not a prime, the Cooley–Tukey principle can be applied iteratively and then the DFT is computed with the aid of several smaller DFTs. In particular, if the DFT length is a power of a prime, i.e., $N = P^q$, then the N-point DFT can be computed with the aid of P-point DFTs constructed in q computing stages. As the resulting fast algorithm contains only P-point DFTs, it is called as a *radix-P FFT*.

The most popular approach is radix-2 FFT algorithm, where the DFT is decomposed recursively until the entire algorithm is computed with the aid of two-point DFTs. The advantage is that that the two-point DFT can be computed with trivial twiddle factors, thus multiplications can be avoided. Another popular algorithm is radix-4 FFT as twiddle factors contain only 1, -1, j, and $-j$, thus again no multiplications are needed.

The recursive application of Cooley–Tukey principle can be done by starting from the time domain sequence, which results in a decimation-in-time (DIT) algorithm as shown in Fig. 2. In a similar fashion, the decomposition process can be started from the frequency domain sequence resulting in a decimation-in-frequency (DIF) algorithm as shown in Fig. 3. The numbers between each stage represent the rotations. Each of these values (ϕ) corresponds to a rotation by the twiddle factor [6]. It should be noted that $W_{16}^2 = W_8^1$; $W_{16}^4 = W_4^1$.

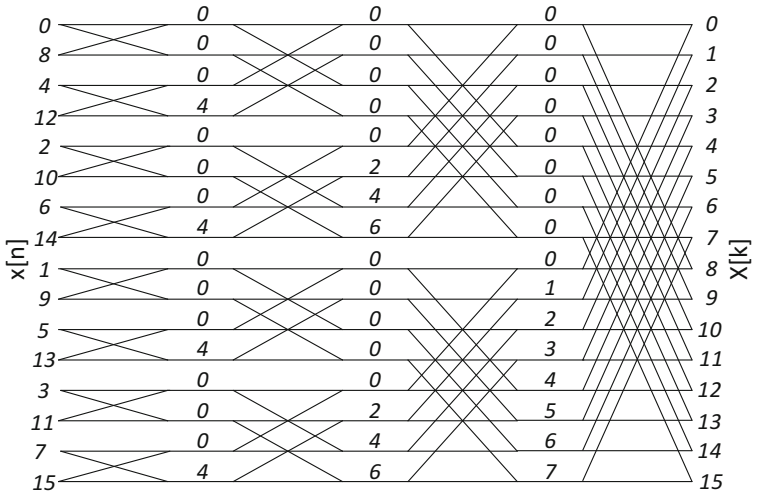

Fig. 2 16-point radix-2 DIT FFT flow graph. The number between the stages indicates rotation, i.e., value ϕ in twiddle factor multiplication W_{16}^{ϕ}

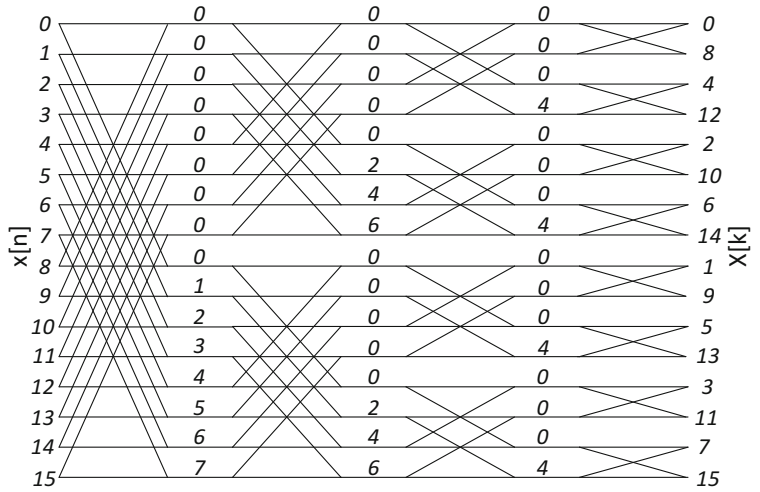

Fig. 3 16-point radix-2 DIF FFT flow graph. The number between the stages indicates rotation, i.e., value ϕ in twiddle factor multiplication W_{16}^{ϕ}

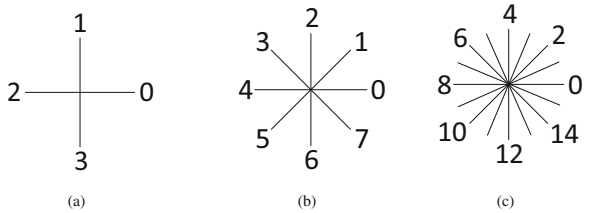

Fig. 4 Twiddle factors: (**a**) $W_4(W_4^0, W_4^1, W_4^2, W_4^3)$, (**b**) $W_8(W_8^0, W_8^1, \ldots, W_8^7)$, and (**c**) $W_{16}(W_{16}^0, W_{16}^1, \ldots, W_{16}^{15})$

2.1 Twiddle Factors

Each input of an FFT stage is rotated by a different angle, which is determined by the twiddle factor $W_L^{\phi} = e^{-j2\pi\phi/L}$. The parameter L is a constant for each stage and determines the number of possible rotations in that stage. These angles are based on the division of the circumference into L equal parts. The exact angle is determined by the parameter ϕ, which is a natural number $\{0, \ldots, L-1\}$ [7, 11, 30, 32].

The complexity of the rotator is determined by the value of L. A small value is desirable, because it results in a simple twiddle factor architecture. An example of twiddle factors W_4, W_8, and W_{16} is shown in Fig. 4.

The simplest rotator is $W_L = W_4$, which includes only trivial rotations (0, $\frac{\pi}{4}$, π, and $\frac{3\pi}{4}$). Trivial rotations are characterized by the fact that they can be calculated by simply exchanging the real and imaginary parts of the input and/or changing their sign [11]. Thus, the criterion used to select the algorithm is to minimize the number

of large twiddle factors and maximize the number of trivial rotators (W_4), which are the cheapest ones. There are also trade-offs between the number of large (W_{64} and larger) and small (W_8, W_{16}, and W_{32}) twiddle factors.

3 Rotation in Fixed-Point Arithmetic

Ideally the arithmetic operations in the FFT algorithm should be represented with infinite precision. However, in practice, temporary values are stored in registers with a finite capacity. There exist different ways to approximate real numbers using a finite number of digits. One of the most common ways for embedded systems is two's complement fixed-point (FxP) arithmetic [5, 27].

For angle α, it is always true that: $\cos\alpha \in [-1, 1]$ and $\sin\alpha \in [-1, 1]$. Therefore, it is assumed that C and S are also numbers in the range $[-1, 1]$ with rounding to a certain number of fractional bits, b. According to the general interpretation of the coefficients, the magnitude of the rotation is always one, independently on the number of bits, b, as it happens in the definition of the rotation.

As C and S contain b bits, they can also be considered integers in two's complement representation in the range of $[-2^{b-1}, 2^{b-1}]$. Accordingly, the values of the cosine and sine components of the angle α will be

$$\begin{aligned} C &= R(\cos\alpha + \epsilon_c); \\ S &= R(\sin\alpha + \epsilon_s), \end{aligned} \tag{8}$$

where ϵ_c and ϵ_s are the relative quantization errors of the cosine and sine components, respectively, and R is the scaling factor of the coefficients being the error rotation [9].

4 Rotations in FFT Architectures

Closer investigation of signal flow graphs of FFT algorithms indicates that the rotations in the signal flow graph have some systematic properties, which can be exploited in implementations. The way how these properties can be exploited depends on the mapping from the signal flow graph onto the processing units. The main differences are that how many samples are processed in parallel and how many different twiddle factors each rotator must support. We can identify three principal types:

1. Single Branch with Multiple Rotations. A general scenario is to compute rotations on the data that flow through a single branch, where different pieces of data rotated by different coefficients are shown in Fig. 5a. This scenario is mostly used in single/multiple feedback pipelined and iterative FFT architectures [11].

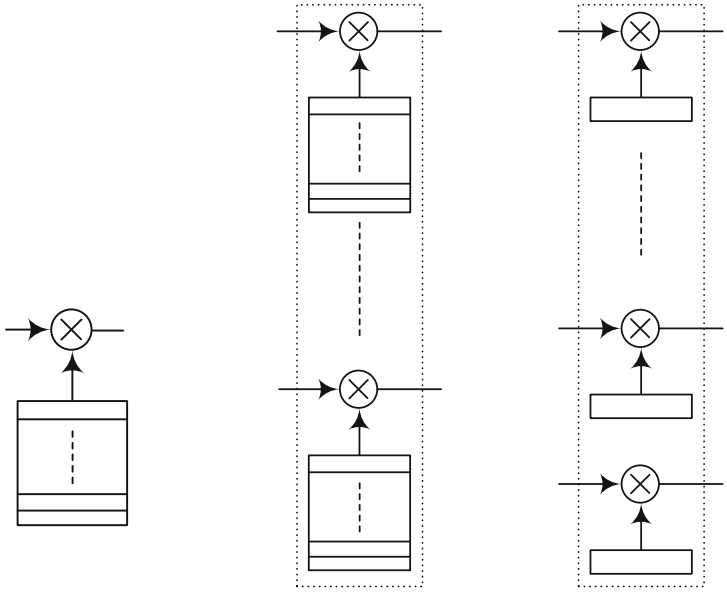

Fig. 5 Rotator scenarios: (**a**) Type 1, (**b**) Type 2, and (**c**) Type 3

2. Multiple Branches, Multiple Rotations for Each Branch. The most typical case consists of several branches with several rotations for each of them, as shown in Fig. 5b. This scenario is part of feed forward pipelined FFT architectures [11].
3. Multiple Branches, Single Rotation for Each Branch. This scenario is applied on fully parallel FFT architecture, where data flow is parallel and each branch may carry out a rotation by a different coefficient [11] as shown in Fig. 5c.

The Type 1 and Type 2 are also referred to as *multiple constant rotations* (MCR) and Type 3 is called as a *single constant rotation* (SCR) [11].

5 Rotator Units

As discussed earlier, the FFT signal flow graphs can be mapped on processing elements with different methods indicating different properties for the units. In this section, we discuss the implications to the rotators due to these mappings and identify three different classes: rotators based on general complex multipliers, constant rotators, and CORDIC rotators.

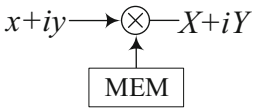

Fig. 6 Rotation architecture based on complex multiplier

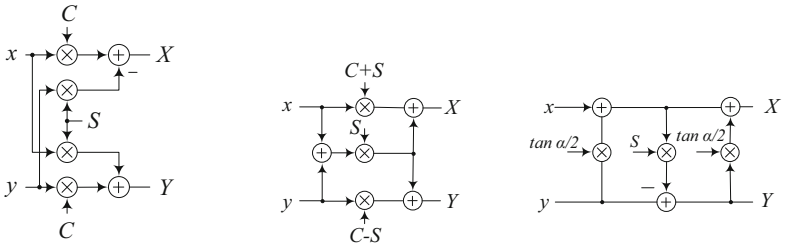

Fig. 7 Rotation based on complex multipliers: (**a**) standard, (**b**) modified I, and (**c**) lifting-based

5.1 Rotators Based on General Complex Multiplier

The most popular approach to implement the rotation is to use complex multiplier and lookup table for storing the sine and cosine components of the rotation angle as shown in Fig. 6 [30]. A straightforward method is to implement the complex multiplication with four real multipliers and two adders as depicted in Fig. 7a. The complex multiplication can be realized more efficiently by exploiting the fact that this is a rotation. Such methods reduce the number of real multipliers from four to three [30]. Among them, the most alluring ones are the following:

$$\begin{aligned} X &= x(\cos\alpha + \sin\alpha) - (x+y)\cdot\sin\alpha; \\ Y &= x(\cos\alpha + \sin\alpha) + (y+x)\cdot\sin\alpha \end{aligned} \tag{9}$$

and

$$\begin{aligned} X &= (x+y)\cos\alpha - y\cdot(\cos\alpha + \sin\alpha); \\ Y &= (x+y)\cos\alpha - x\cdot(\cos\alpha - \sin\alpha). \end{aligned} \tag{10}$$

Both these cases consist of a common term in the equations for X and Y that only need to be computed once. Thus, when $(C+S)$ and $(C-S)$ are precomputed, these cases require three real-valued multiplications and five real-valued additions. The architecture for (9) is shown in Fig. 7b.

Another approach can be applied only in the case of rotation, which is called lifting-based rotators [2]. The rotation in (1) can be written in matrix form as follows:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}. \tag{11}$$

We can apply the lifting approach to the previous equation and the following form is obtained:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -\tan\frac{\alpha}{2} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ \sin\alpha & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & \tan\frac{\alpha}{2} \\ 0 & -1 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}. \tag{12}$$

By further computing the matrices, the following equations are obtained:

$$\begin{aligned} X &= x - x\sin\alpha\tan\frac{\alpha}{2} + 2y\tan\frac{\alpha}{2} - y\tan^2\frac{\alpha}{2}\sin\alpha; \\ Y &= y - x\sin\alpha - y\sin\alpha\tan\frac{\alpha}{2}. \end{aligned} \tag{13}$$

These equations can be realized with the structure depicted in Fig. 7c. This approach requires also three real-valued multiplications and three real-valued additions. However, there is no need for an additional coefficient as it simply replaces the $\cos\alpha$ with $\tan\frac{\alpha}{2}$ in memory.

This type of rotator can be applied in all types of FFT architecture scenarios mentioned earlier. Figure 5b and c requires more than one rotation, which are used in signal flow graphs and parallel pipelined FFT architectures, respectively. In those scenarios, the complexity of rotation memory can be reduced by applying appropriate techniques. Rotation memory is used to store the coefficients C and S of rotation. There are a number of techniques to store the coefficients in the memory according to the requirement of FFT architectures [31]. The simplest approach is to store all the coefficients of rotations in the memory without considering any optimization technique as shown in Fig. 8. This results in a large rotations memory especially for large FFTs. It should be noted that this scheme possibly stores the same rotations in several locations as the mapping is from the computing stages of FFT algorithms.

A possible simplification is to use an address generator that generates the row address for the corresponding angle. As a result, we only need to store the coefficients once in the memory as shown in Fig. 9. For the case $L = N$, we will need to store many but not all values, still using N possible words even though many

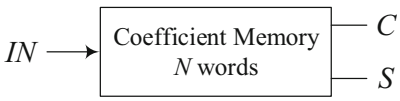

Fig. 8 Coefficient memory for storing all rotations of twiddle factor

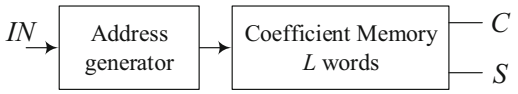

Fig. 9 Coefficient memory with address generation

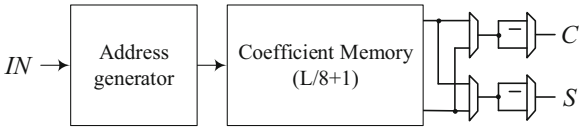

Fig. 10 Coefficient memory with address generation and octave symmetry circuit

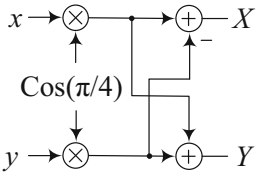

Fig. 11 Single constant rotation by $\cos(\frac{\pi}{4})$

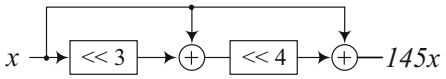

Fig. 12 Simplified constant multiplication

can be set as "don't cares." Due to this fact, one can expect the resources used to the lookup table to be reduced compared to the previous approach, given that the synthesis tool can benefit from it.

Another modification, proposed in [16], is to use the well-known octave symmetry to only store twiddle factors for $0 \leq \phi \leq \frac{\pi}{4}$. The additional cost is an address mapping circuit as discussed in the previous section as well as multiplexers to interchange the real and imaginary parts and possible negations. The main benefit is that only $\frac{L}{8} + 1$ words are required to be stored. The resulting architecture is illustrated in Fig. 10.

5.2 *Constant Rotators*

Typically the real-valued multipliers discussed in the previous section can be replaced by shift-and-add multiplication as shown in [15, 22, 39]. This is especially useful when the rotator needs to support only a single rotation angle.

Let us consider a rotation by $\frac{\pi}{4}$, which requires only one coefficient as $\cos\frac{\pi}{4} = \sin\frac{\pi}{4}$. Hence, each real, x, and imaginary, y, parts of the operand are multiplied separately by $\cos\frac{\pi}{4}$ and the multiplier outputs are added and subtracted as depicted in Fig. 11. When an input signal is multiplied by one constant, an optimal single constant multiplier from [13] can be used for the implementation. The internal architecture of a constant multiplier is shown in Fig. 12, where the input operand x is multiplied with a coefficient 145, the result being $145x$.

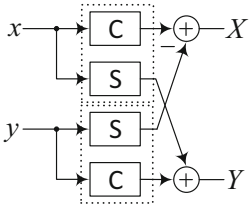

Fig. 13 Rotation based on constant multiplication

In terms of complexity, the shift operations are free as they only reduce the number of adders in the implementation of the constant multiplications. A general block diagram of the rotator is shown in Fig. 13. This consists of two constant multiplication blocks by dashed blocks. Each dashed block consists of two constants, C and S, which are implemented by shift-and-add. Hence, both the multipliers sharing the same input can be simultaneously realized using a *multiple constant multiplication* (MCM) technique [4].

The constant rotators can be applied for all the FFT rotation scenarios. However, this architecture is no more efficient for large values of L; the higher the value of L, the greater number of constant coefficients are needed implying more complex shift-and-add circuits. As a result, this architecture can be used efficiently for twiddle factors W_8, W_{16}, and W_{32} [11, 25, 32]. The maximum number of rotations in the twiddle factor set is $0, \ldots, L-1$. As discussed earlier, thanks to the symmetries of the angles on the complex plane, for L-point twiddle factor only the $L/8+1$ angles in the range $[0, \pi/4]$ need to be considered. This is due to the fact that the rest of rotations of twiddle factor can be formed from these angles by interchanging the real and imaginary parts of the input and output data and/or the signs of the outputs. The design of constant rotators can be divided into two main parts: the generation of the rotation coefficients and the implementation of shift-and-add circuit.

5.2.1 Rotation Coefficients

In hardware implementations, it is not only desirable to reduce the computation error but also the area on the circuit. As mentioned earlier: $\cos\alpha \in [-1, 1]$ and $\sin\alpha \in [-1, 1]$. Therefore, in general it is assumed that C and S are also numbers in the range of $[-1, 1]$ with rounding a certain number of fractional bits, b. There is a number of methods to reduce the coefficient error without increasing the addition cost.

A method based on searching the coefficients by allowing word length increase by E fractional bits is introduced in [14]. The approximation error can be guaranteed to meet $\epsilon \leq 2^{-(N+E+1)}$ with a brute force method by searching coefficients, which fulfill this condition such that the word length increase is at most E fractional bits. Another way of using the additional fractional bits is to realize that there are exactly 2^E different representable coefficients for which $\epsilon \leq 2^{-(N+1)}$, including

the one obtained by rounding to b bits. The basic idea is to search these 2^E and select the coefficient value that has the smallest approximation error for allowed complexity called as the addition aware quantization [14]. The allowed complexity is typically assumed to be the same number of additions as required by the coefficient rounded to a coefficient with b fractional bits. This is a generic method and it can be applied on rotation coefficients. Another method, which is well suited especially to rotations, is the so-called *combined coefficient selection and shift-and-add implementation* (CCSSI) [11]. This method refers to a set of rotations that must be optimized together. This joint optimization happens when there is a dependency on the scaling of the rotations. Different optimization problems can be defined for SCR and MCR depending on the scaling that is required and on the hardware layout. Thus, the scaling can be fixed, unity, or arbitrary depending on the freedom to choose the scaling factor. Unity scaling is a particular case of fixed scaling, where the rotation has magnitude of one or, in more general terms, $R = 2^b$. This is equivalent to considering that the binary point is in a different position in the binary representation. Conversely, arbitrary scaling means that R can take any value, i.e., no restriction is set to R. For arbitrary scaling the approximation error is equal to the angular error only, since R will always take the optimal value. However, the scaling for multiple angles is classified based on the relation among the scaling factors of the rotations. More details of the design process can be found from [11].

In pipelined FFT architectures, uniform scaling can be applied on sets of rotations, which means that R is the same for all the rotations of each FFT stage. The purpose is to select the best coefficients of rotations. Thus, when fixed or arbitrary scaling factor is applied, the output of FFT is shifted by a certain factor.

5.2.2 Shift-and-Add Circuit Implementation

This section describes the methods to design the constant rotators circuit for FFT, especially for MCR. There are two main methods to design rotation circuits for FFT. One is based on using combinations of rotation coefficients for other rotation with the aid of additional multiplexers. Thus, it is reducing the coefficients of rotations. Other is merging the rotation and sharing the adders among them by using additional multiplexers.

Regarding the first technique, trigonometric identities are used to reduce the number of required coefficients. This techniques is applied on twiddle factors of W_{16} and W_{32} to reduce the required coefficients from three to two and seven to three, respectively [32]. Thus, the equivalent expression for all the coefficients in twiddle factors of W_{16} and W_{32} is tabulated in Tables 1 and 2, respectively [25, 32].

The architecture for W_{16} twiddle factor is shown in Fig. 14, where a single input is multiplied with any of the coefficient pairs $\{(1, 0), (\cos\frac{\pi}{8}, \sin\frac{\pi}{8}), (\cos\frac{\pi}{4}, \sin\frac{\pi}{4})\}$.

Another implementation technique is based on CCSSI. The coefficient selection has been explained in Sect. 5.2.1. The implementation consists of two steps: first,

Table 1 Trigonometric identities used for W_{16} twiddle factors

Coefficient	Used expression
$\sin\frac{\pi}{4}$	$2\sin\frac{\pi}{8}\cos\frac{\pi}{8}$
$\sin\frac{\pi}{8}$	$\sin\frac{\pi}{8}$
$\cos\frac{\pi}{8}$	$\cos\frac{\pi}{8}$

Table 2 Trigonometric identities used for W_{32} twiddle factor

Coefficient	Used expression
$\sin\frac{\pi}{4}$	$4\cos\frac{\pi}{8}\cos\frac{\pi}{16}\sin\frac{\pi}{16}$
$\sin\frac{\pi}{8}$	$2\cos\frac{\pi}{16}\sin\frac{\pi}{16}$
$\cos\frac{\pi}{8}$	$\cos\frac{\pi}{8}$
$\sin\frac{3\pi}{16}$	$\sin\frac{\pi}{16}\left(2\cos\frac{\pi}{8}+1\right)$
$\cos\frac{3\pi}{16}$	$\cos\frac{\pi}{16}\left(2\cos\frac{\pi}{8}-1\right)$
$\sin\frac{\pi}{16}$	$\sin\frac{\pi}{16}$
$\cos\frac{\pi}{16}$	$\cos\frac{\pi}{16}$

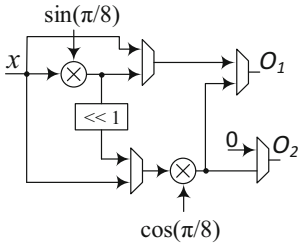

Fig. 14 Architecture for W_{16} twiddle factors [25]

obtain the implementation of a rotation by each SCR and then merge together all the rotations that are carried out by the MCR.

The rotation by an SCR includes the multiplication of the input by C and S, and addition of the products. This corresponds to Eq. (1). The multiplication by C and S is carried out by means of shifts and additions according to the MCM representation of the numbers. Similarly, layout the architecture of all the rotations that are carried by the MCR. Finally, the rotations carried out by the same rotator must be merged together. This is done by adding multiplexers to the inputs of the adders. Figure 15 illustrates an architecture for the computing W_8 twiddle factor and Fig. 15a and b shows the multiplication by 1 and $\frac{\pi}{4}$ rotation, respectively.

The architecture in Fig. 15c is a result of merging together the architectures in Fig. 15a and b. The control signal of the multiplexer controls the multiplication of $\{(1, 0), (\cos\frac{\pi}{4}, \sin\frac{\pi}{4})$ [11]. In order to obtain an efficient realization of the rotator, reconfigurable single [13, 35] and multiple constant multiplication [4, 12] techniques can be used. Alternatively, when the number of coefficients is small, which is true in most of the practical cases, the selection of an efficient implementation can be found manually.

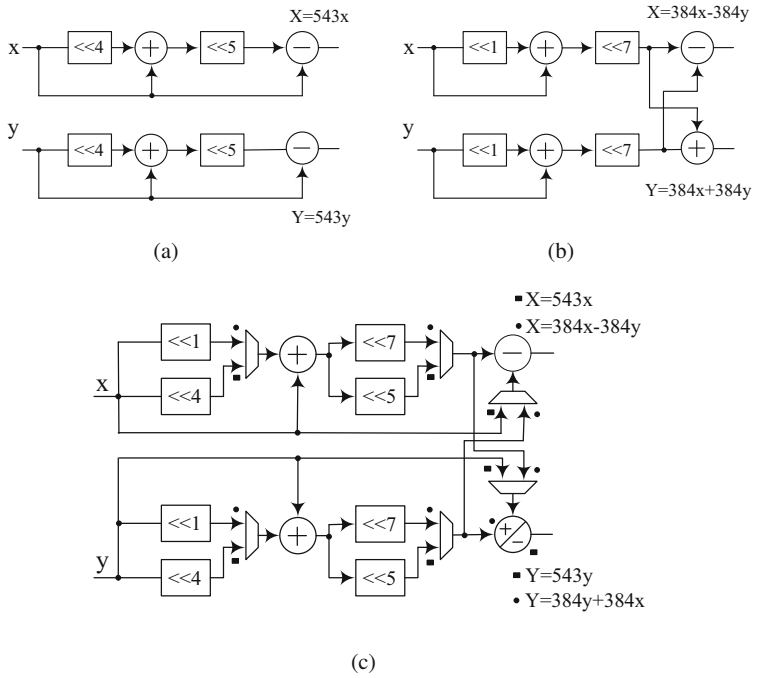

Fig. 15 W_8 twiddle factor: (**a**) multiplication by 1, (**b**) rotation by $\frac{\pi}{4}$, and (**c**) $\{(1), (\cos\frac{\pi}{4}, \sin\frac{\pi}{4})\}$

5.3 CORDIC Rotator

COordinate Rotation DIgital Computer (CORDIC) is one of the most popular algorithms for implementing multiplier-less rotations [1, 7, 8, 36]. It realizes rotation by means of a series of shifts and additions, which reduces the amount of hardware. It is also suitable for cases, where multipliers are not available. However, it may affect the accuracy since it is based on an approximation. The CORDIC algorithm decomposes the angle that has to be rotated, θ, into a sum of M predefined angles, α_i, according to:

$$\theta = \sum_{i=0}^{M-1} \delta_i \alpha_i + \epsilon, \tag{14}$$

where ϵ is the error of the approximation, δ_i indicates the direction of the so-called micro-rotation and

$$\alpha_i = \tan^{-1}(2^{-i}). \tag{15}$$

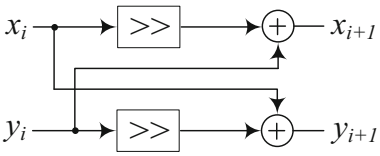

Fig. 16 CORDIC micro-rotation

These angles that define the micro-rotations have the property that they can be rotated by shifts and additions, which reduces significantly the hardware resources. These micro-rotations are carried out as follows:

$$\begin{aligned} x_{i+1} &= x_i - y_i\ \delta_i\ 2^{-i}; \\ y_{i+1} &= y_i + x_i\ \delta_i\ 2^{-i}. \end{aligned} \tag{16}$$

The hardware circuit for calculating the case of $\delta_i = 1$ is depicted in Fig. 16; input samples are rotated with an angle α_i, which is chosen by setting the number of bits that are shifted before the additions and subtractions are carried out.

Usually $\delta \in \{-1, 1\}$. This forces all the micro-rotations to be computed either clockwise or counterclockwise and assures a constant gain for the CORDIC computations, which can be compensated by multiplying the outputs by:

$$K = \prod_{i=0}^{M} \cos(\alpha_i) = \prod_{i=0}^{M} \cos(\tan^{-1}(2^{-i})) = 0.6073. \tag{17}$$

This option is preferable when the circuit is used for rotating several different angles and a constant gain for all of them is required, as happens in the rotators for the FFT. However, in a constant rotator, only a single angle θ can be rotated. In this case, it is better to consider $\delta_i \in \{-1, 0, 1\}$. This approach is called as redundant CORDIC [19], which allows certain micro-rotations to be removed, reducing the number of adders.

There are multiple variations of the CORDIC algorithm. Some of the main modifications are introduced in the following. Surveys on CORDIC techniques can be found, e.g., in [1, 24]. For some of the approaches, it is not straightforward to determine the rotation parameters at run time. Hence, for these methods the design is carried out offline and the control signals are stored in memory rather than the angles. This approach is naturally possible for all techniques and, as the sequence of angles is often known beforehand, most likely advantageous compared to storing the angle values.

The redundant CORDIC considers that $\delta_i \in \{-1, 0, 1\}$ [34] or even $\delta_i \in \{-2, -1, 0, 1, 2\}$ [20]. This allows several rotation angles at each CORDIC stage. However, the scaling for different angles is different, which implies a need for a specific circuit for scaling compensation. The *extended elementary angle set* (EEAS) CORDIC [38] and *mixed-scaling-rotation* (MSR) CORDIC [21, 29] also follow the idea of increasing the number of rotation angles per rotation stage.

The memoryless CORDIC [10] removes the need for rotation memory to store the FFT rotation angles. Instead, the control signals δ_i are generated from a counter. This is advantageous for large FFTs, where the butterfly stages have a large number of rotations. The *modified vector rotational* (MRV) RORDIC [37] allows skipping and repeating CORDIC stages, whereas the hybrid CORDIC [17, 33] divides the rotations into a coarse and a fine rotations. These techniques reduce the number of stages and, therefore, the latency of the CORDIC. The CORDIC II [10] proposes new types of rotation stages: friend angles, uniformly scaled redundant (USR) CORDIC, and nano-rotations. These result in both a low latency and a small number of adders. Finally, the base-3 rotators [18] consider an elementary angle set that is different to that of the CORDIC. All the rotations are generated by combining a small set of FFT angles. This set fits better the rotation angles of the FFT than that of the CORDIC, which results in a reduction in the rotation error, the number of adders, and latency of the circuit.

6 Conclusions

Rotation architecture has an important role in the design of an FFT architecture and has a large effect on the cost of the architecture. This chapter provided an overview over different existing architectures of the rotations especially for FFT. These can be implemented using complex-valued multipliers, constant multipliers, and the CORDIC. Architecture based on the CORDIC and constant multiplication uses shift-and-add circuit, whereas the complex multiplication generally uses complex multiplier and memory to store the coefficients of rotation.

References

1. Andraka, R.: A survey of CORDIC algorithms for FPGA based computers. In: Proc. ACM/SIGDA Int. Symp. FPGAs, pp. 191–200 (1998)
2. Chan, S.C., Yiu, P.M.: An efficient multiplierless approximation of the fast Fourier transform using sum-of-powers-of-two (SOPOT) coefficients. IEEE Signal Process. Lett. **9**(10), 322–325 (2002)
3. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19**, 297–301 (1965)
4. Dempster, A.G., Macleod, M.D.: Multiplication by two integers using the minimum number of adders. In: Proc. IEEE Int. Symp. Circuits Syst., vol. 2, pp. 1814–1817 (2005). https://doi.org/10.1109/ISCAS.2005.1464962
5. Finley, T.: Two's complement. Cornell University lecture notes (2000)
6. Garrido, M.: A new representation of FFT algorithms using triangular matrices. IEEE Trans. Circuits Syst. I **63**(10), 1737–1745 (2016)
7. Garrido, M., Andersson, R., Qureshi, F., Gustafsson, O.: Multiplierless unity-gain SDF FFTs. IEEE Trans. VLSI Syst. **24**(9), 3003–3007 (2016)
8. Garrido, M., Grajal, J.: Efficient memoryless CORDIC for FFT computation. In: Proc. IEEE Int. Conf. Acoust. Speech Signal Process., vol. 2, pp. 113–116 (2007)

9. Garrido, M., Gustafsson, O., Grajal, J.: Accurate rotations based on coefficient scaling. IEEE Trans. Circuits Syst. II **58**(10), 662–666 (2011)
10. Garrido, M., Källström, P., Kumm, M., Gustafsson, O.: CORDIC II: A new improved CORDIC algorithm. IEEE Trans. Circuits Syst. II **63**(2), 186–190 (2016). https://doi.org/10.1109/TCSII.2015.2483422
11. Garrido, M., Qureshi, F., Gustafsson, O.: Low-complexity multiplierless constant rotators based on combined coefficient selection and shift-and-add implementation (CCSSI). IEEE Trans. Circuits Syst. I **61**(7), 2002–2012 (2014)
12. Gustafsson, O.: A difference based adder graph heuristic for multiple constant multiplication problems. In: Proc. IEEE Int. Symp. Circuits Syst., pp. 1097–1100 (2007). https://doi.org/10.1109/ISCAS.2007.378201
13. Gustafsson, O., Dempster, A.G., Johansson, K., Macleod, M.D., Wanhammar, L.: Simplified design of constant coefficient multipliers. Circuits Syst. Signal Process. **25**(4), 225–251 (2006)
14. Gustafsson, O., Qureshi, F.: Addition aware quantization for low complexity and high precision constant multiplication. IEEE Signal Process. Lett. **17**(2), 173–176 (2010)
15. Han, W., Erdogan, A.T., Arslan, T., Hasan, M.: High-performance low-power FFT cores. ETRI J. **30**(3), 451–460 (2008). https://doi.org/10.4218/etrij.08.0107.0189
16. Hasan, M., Arslan, T.: Scheme for reducing size of coefficient memory in FFT processor. Elect. Letters **38**(4), 163–164 (2002)
17. Hsiao, S.F., Lee, C.H., Cheng, Y.C., Lee, A.: Designs of angle-rotation in digital frequency synthesizer/mixer using multi-stage architectures. In: Proc. Asilomar Conf. Signals Syst. Comput., pp. 2181–2185 (2011). https://doi.org/10.1109/ACSSC.2011.6190418
18. Källström, P., Garrido, M., Gustafsson, O.: Low-complexity rotators for the FFT using base-3 signed stages. In: Proc. IEEE Asia-Pacific Conf. Circuits Syst., pp. 519–522 (2012)
19. Lee, J.A., Lang, T.: Constant-factor redundant CORDIC for angle calculation and rotation. IEEE Trans. Comput. **41**(8), 1016–1025 (1992). https://doi.org/10.1109/12.156544
20. Li, C.C., Chen, S.G.: A radix-4 redundant CORDIC algorithm with fast on-line variable scale factor compensation. In: Proc. IEEE Int. Conf. Acoust. Speech Signal Process., vol. 1, pp. 639–642, vol. 1 (1997). https://doi.org/10.1109/ICASSP.1997.599849
21. Lin, Z.X., Wu, A.Y.: Mixed-scaling-rotation CORDIC (MSr-CORDIC) algorithm and architecture for scaling-free high-performance rotational operations. In: Proc. IEEE Int. Conf. Acoust. Speech Signal Process., vol. 2 (2003). https://doi.org/10.1109/ICASSP.2003.1202451
22. Liu, H., Lee, H.: 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., pp. 834–837 (2008)
23. Loeffler, C., Ligtenberg, A., Moschytz, G.: Practical fast 1-D DCT algorithms with 11 multiplications. In: Proc. IEEE Int. Conf. Acoust. Speech Signal Process., vol. 2, pp. 988–991 (1989). https://doi.org/10.1109/ICASSP.1989.266596
24. Meher, P.K., Valls, J., Juang, T.B., Sridharan, K., Maharatna, K.: 50 years of CORDIC: Algorithms, architectures, and applications. IEEE Trans. Circuits Syst. I **56**(9), 1893–1907 (2009). https://doi.org/10.1109/TCSI.2009.2025803
25. Oh, J.Y., Lim, M.S.: New radix-2 to the 4th power pipeline FFT processor. IEICE Trans. Electron. **E88-C**(8), 1740–1746 (2005)
26. Oppenheim, A., Schafer, R.: Discrete-Time Signal Processing. Prentice Hall (1989)
27. Padgett, W.T., Anderson, D.V.: Fixed-point signal processing. Synthesis Lectures on Signal Processing **4**(1), 1–133 (2009)
28. Parhi, K.K.: VLSI Digital Signal Processing Systems, Design and Implementation. Wiley-Interscience (1999)
29. Park, S.Y., Yu, Y.J.: Fixed-point analysis and parameter selections of MSR-CORDIC with applications to FFT designs. IEEE Trans. Signal Process. **60**(12), 6245–6256 (2012). https://doi.org/10.1109/TSP.2012.2214218
30. Qureshi, F.: Optimization of rotations in FFTs. Ph.D. thesis, Linköping University (2012)
31. Qureshi, F., Gustafsson, O.: Analysis of twiddle factor memory complexity of radix-2^i pipelined FFTs. In: Proc. Asilomar Conf. Signals Syst. Comput., pp. 217–220 (2009). https://doi.org/10.1109/ACSSC.2009.5470121

32. Qureshi, F., Gustafsson, O.: Low-complexity constant multiplication based on trigonometric identities with applications to FFTs. IEICE Trans. Fundamentals **E94-A**(11), 324–326 (2011)
33. Shukla, R., Ray, K.: Low latency hybrid CORDIC algorithm. IEEE Trans. Comput. **63**(12), 3066–3078 (2014). https://doi.org/10.1109/TC.2013.173
34. Takagi, N., Asada, T., Yajima, S.: Redundant CORDIC methods with a constant scale factor for sine and cosine computation. IEEE Trans. Comput. **40**(9), 989–995 (1991). https://doi.org/10.1109/12.83660
35. Thong, J., Nicolici, N.: An optimal and practical approach to single constant multiplication. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **30**(9), 1373–1386 (2011). https://doi.org/10.1109/TCAD.2011.2153853
36. Volder, J.E.: The CORDIC trigonometric computing technique. IRE Trans. Electronic Computing **EC-8**, 330–334 (1959)
37. Wu, C.S., Wu, A.Y.: Modified vector rotational CORDIC (MVR-CORDIC) algorithm and architecture. Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on **48**(6), 548–561 (2001). https://doi.org/10.1109/82.943326
38. Wu, C.S., Wu, A.Y., Lin, C.H.: A high-performance/low-latency vector rotational CORDIC architecture based on extended elementary angle set and trellis-based searching schemes. IEEE Trans. Circuits Syst. II **50**(9), 589–601 (2003). https://doi.org/10.1109/TCSII.2003.816923
39. Yang, C.H., Yu, T.H., Markovic, D.: Power and area minimization of reconfigurable FFT processors: A 3GPP-LTE example. IEEE J. Solid-State Circuits **47**(3), 757–768 (2012). https://doi.org/10.1109/JSSC.2011.2176163