A New Representation of FFT Algorithms Using Triangular Matrices

Mario Garrido Gálvez

Journal Article

Mario Garrido Gálvez , A New Representation of FFT Algorithms Using Triangular Matrices, IEEE Transactions on Circuits and Systems Part 1, 2016. 63(10), pp.1737-1745.
http://dx.doi.org/10.1109/TCSI.2016.2587822
Postprint available at: Linköping University Electronic Press

http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-132528

A new Representation of FFT Algorithms using Triangular Matrices

Mario Garrido, *Member, IEEE*

***Abstract*—In this paper we propose a new representation for FFT algorithms called the triangular matrix representation. This representation is more general than the binary tree representation and, therefore, it introduces new FFT algorithms that were not discovered before. Furthermore, the new representation has the advantage that it is simple and easy to understand, as each FFT algorithm only consists of a triangular matrix. Besides, the new representation allows for obtaining the exact twiddle factor values in the FFT flow graph easily. This facilitates the design of FFT hardware architectures. As a result, the triangular matrix representation is an excellent alternative to represent FFT algorithms and it opens new possibilities in the exploration and understanding of the FFT.**

***Index Terms*—Binary tree, Cooley-Tukey, fast Fourier transform (FFT)**

I. Introduction

It was 50 years ago when Cooley and Tukey proposed the fast Fourier transform (FFT) algorithm [1]. The FFT is a way to compute the discrete Fourier transform (DCT) that reduces the amount of computations from $O(N^2)$ to $O(N \log N)$ [2].

The Cooley-Tukey algorithm led to two main algorithms: Radix-2 decimation in frequency (DIF) and radix-2 decimation in time (DIT). The radix-2 DIF FFT is a decomposition in which the output sequence is separated into even and odd samples iteratively. Analogously, the radix-2 DIT FFT separates the input sequence into even and odd samples iteratively.

After the Cooley-Tukey algorithm, new FFT algorithms were presented. The radix-4 FFT algorithm [3] reduces the rotations of the radix-2 algorithm. The same occurs with radix-8 and higher radices. For further information about the first FFT algorithms, [4] provides historical notes on the first FFT algorithms.

Later, the radices were combined in mixed radix algorithms [5], [6]. Mixed radix is generally used when the number of points of the FFT, N, is not a power of the radix. For instance, $N = 128$ is not a power of 4 and, therefore, it cannot be implemented using radix-4. However, it can be implemented with mixed radix-4 and 2, being all the stages of radix-4 except the last one.

An evolution of the FFT algorithms happened with radix-2^2 FFT [7], [8]. This radix led to new FFT architectures with less rotators than the radix-2 ones and less adders than the radix-4 ones. Later, radix-2^3 [9], [10], radix-2^4 [11] and in general radix-2^k [9], [12], [13] algorithms were proposed.

Nowadays we can talk about a large number of FFT algorithms. This is thanks to the binary tree representation [14], [15]. This representation is a generalization of the Cooley-Tukey algorithm, which allows for decimating an FFT at any stage in each iteration of the decimation. The binary tree representation can represent all the algorithms mentioned previously. A comparison of FFT algorithms can be found in [16].

In this paper we propose a new representation of the FFT algorithms. It is called triangular matrix, as the FFT algorithms are represented by a triangular matrix. The triangular matrix representation has four main advantages. First, the representation is simple and easy to understand. Second, it support any radix, including mixed radix algorithms. Third, it is a more general representation and allows for representing significantly more algorithms than the binary tree one. Third, it allows for generating the twiddle factors at each stage of the FFT easily. As a result, the triangular matrix representation opens new possibilities in the research of the FFT.

The paper is organized as follows. Section II reviews previous FFT representations. Section III explains how to move rotations among FFT stages. This section also refutes some common beliefs related to the FFT that are shown to be wrong. Section IV presents the triangular matrix representation, relates it to the binary tree representation and shows how to obtain the twiddle factors. Section V shows the application to FFT implementation. Finally, Section VI provides the main conclusions of the paper.

II. Review of previous FFT representations

A. The Cooley-Tukey algorithm

The Cooley-Tukey algorithm [17], [18] is based on the observation that multiple operations can be shared when calculating the output frequencies of the FFT. This is done by decomposing the equation of the DFT:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}nk} \tag{1}$$

The most common decompositions are decimation in time (DIT) and decimation in frequency (DIF) [18]. The DIT decomposition separates the sequence $x[n]$ into its even and odd samples, whereas the DIF decomposition is applied on the output sequence $X[k]$.

In both decompositions, the N-point DFT is transformed into two $N/2$-point DFTs. By applying the procedure iteratively, each step halves the number of points of the DFTs, which finally leads to 2-point DFTs.

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

This work was supported by the the Swedish ELLIIT Program.

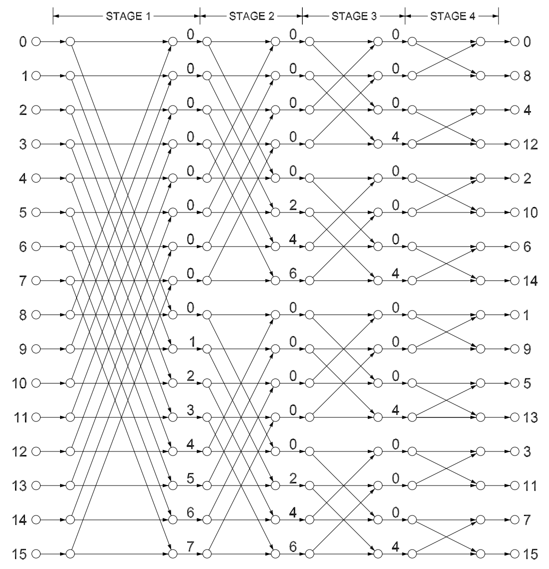


Fig. 1. 16-point radix-2 DIF FFT flow graph.

B. FFT representation using flow graphs

In order to understand the operations performed by an FFT, the FFT is represented most times by its flow graph. An example of an FFT flow graph is shown in Fig. 1. The numbers at the input represent the indexes of the input sequence, $x[n]$, whereas those at the output are the frequencies, k, of the output signal $X[k]$. It can be observed that the inputs are ordered, and the outputs are provided in the so-called bit-reversed order [19].

The flow graph consists of a series of n stages, $s \in \{1 \dots n\}$, where additions, subtractions and complex rotations are calculated. Additions and subtractions come in pairs, forming a structure called butterfly. The flow graph in Fig. 1 assumes that the lower edge of each butterfly is always multiplied by -1. These -1 are usually not depicted in order to simplify the graphs.

The rotations are represented by the numbers between the stages, ϕ. Each of these values corresponds to a rotation by the twiddle factor:

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

As general criterion, throughout the paper, it is considered that the rotations of each stage s are those placed to the right of the butterflies of that stage.

Whereas W_N^{ϕ} represents a single rotation value, W_L (without exponent) represents a set of rotations, where

$$W_L = W_L^k, \quad k = 0, \dots, L-1 \tag{3}$$

Rotations by the angles 0°, 90°, 180° and 270°, are called trivial rotations. These rotations correspond to multiplications by 1, j, -1 and $-j$, respectively. They are called trivial rotations because they can be easily implemented by interchanging the real and imaginary parts and/or changing the sign of the data. Note that the twiddle factor $W_4 = W_4^k = W_N^{kN/4}$, $k \in \{0, \dots, 3\}$ only includes trivial rotations, whereas larger twiddle factor include other rotations as well.

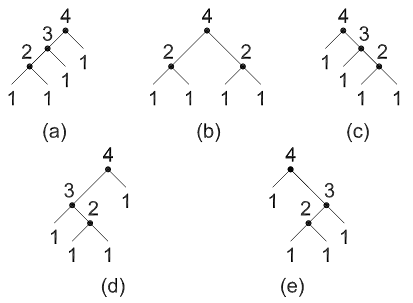


Fig. 2. Binary tree diagram of all possible algorithm for $N = 16$.

C. Binary-tree representation

The binary tree representation [14], [15] is a generalization of the Cooley-Tukey algorithm. The difference among them lies in the fact that the binary tree representation allows for decimating an FFT of any size at each of the iterations in the decomposition, not only a 2-point DFT.

A binary tree diagram is an effective way to represent various FFT algorithms [14]. Fig. 2 shows all the possible binary tree diagrams for $N = 16$. In binary tree representation, each node is assigned a value m to represent a 2^m-point DFT. Each node has at most two leaves which represent the inner and outer DFTs.

Mathematically, the binary tree decomposition is described as

$$\begin{aligned} &X[k_1 + Pk_2] = \\ &= \sum_{t_2=0}^{Q-1} \left[\left(\sum_{t_1=0}^{P-1} x[Qt_1 + t_2] W_P^{t_1 k_1} \right) W_N^{t_2 k_1} \right] W_Q^{t_2 k_2}, \end{aligned} \tag{4}$$

where $0 \leq k_1 \leq P-1$ and $0 \leq k_2 \leq Q-1$.

Here, an N-point DFT is decomposed into Q DFTs of P-point and P DFTs of Q-point. These are named as inner and outer DFTs, respectively. Each output of the inner DFT is multiplied by a twiddle factor, which has a resolution $N = 2^{p+q}$ and index $t_2 k_1$.

The decimation of an N point DFT is done by choosing the values P and Q. Here, it is important to realized that $N = P \cdot Q$. Thus, the first iteration of the binary tree decomposition allows for splitting an N-point DFT in multiple different ways, which correspond to the possible combinations of P and Q whose product is equal to N.

After the first iteration, the remaining P and Q-point DFTs are again divided into smaller DFTs by using the same procedure. Again, these DFTs can be divided arbitrarily by factors whose product is equal to the corresponding DFT size. The procedure continues until all the branches have been reduced to radix-2 DFTs. When the binary tree is finalized, all leaf nodes correspond to the required radix and internal nodes correspond to twiddle factor multiplication stages, which connect the two decomposed DFTs.

The number of radix-2 2^n-point FFT algorithms generated using binary trees is [15]

$$\frac{(2(n-1))!}{n!(n-1)!}. \tag{5}$$

This number comes from all the possible selections that can be done at each iteration. For instance, for $N = 16$,

$n = \log_2 N = 4$. According to (5) this leads to 5 different algorithms, as shown in Fig. 2.

The twiddle factors at each stages are directly obtained from the binary tree. This is done by going across the binary tree from left to right. In this process, the final leafs of the tree, represented by 1, correspond to radix-2 DFT operations. For example, the sequence of numbers in the binary tree in Fig. 2(a) is 2, 3, 4. Given this sequence, the number of angles of the corresponding twiddle factors is the power of these numbers. Thus, for this example the twiddle factors are W_4, W_8 and W_{16}. This corresponds to the DIT decomposition.

D. Matrix factorization

Another way to represent the FFT is by using the matrix factorization [12], [20]. This method represents the FFT in an algebraic way as a product of matrices. For instance, the equation

$$T_N = P_N^{(r)} \cdot [I_r \otimes T_{n/r}] \cdot D_N^{(r)} \cdot [T_r \otimes I_{N/r}], \qquad (6)$$

is a matricial representation of the recursion used in the Cooley-Tukey algorithm [12]. In the equation, $\otimes$ represents a tensor product or Kronecker product and the matrices represent permutations and twiddle factor multiplications.

E. Comparison of representations

Each FFT representation has its utility. The Cooley-Tukey algorithm and the matrix factorization approaches represent the FFT algorithm algebraically. The matrix factorization is a powerful mathematical and computational tool to represent not only the FFT algorithms but also the permutations involved in FFT architectures. Although they are useful approaches for the generation of FFTs, in terms of visualization and representation the Cooley-Tukey algorithm and the matrix factorization are difficult to visualize and understand. This makes it difficult to observe different algorithms and draw conclusions about them.

If we look for a representation of FFT algorithms we want to get an idea of the algorithm from its visualization. From this perspective, the representation using flow graphs provides more clarity than the Cooley-Tukey and the matrix factorization approaches. The representation using flow graphs is excellent for FFTs of small sizes. However, for large FFTs the size of the graph increases drastically and it becomes cumbersome.

Finally, the binary tree representation is clear, intuitive, compact and can represent small and large FFTs easily. If the purpose is to represent and compare FFT algorithms, the binary tree representation is the most attractive representation so far.

III. Moving FFT rotations

A. Basic movement of twiddle factors

In an FFT, rotations can be moved among various stages. Fig. 3 shows how rotations are moved in a flow graph. Fig. 3(a) considers the rotations previous to a butterfly, $\phi_x = a$ and $\phi_y = b$, and those after the butterfly, $\phi_X = c$ and $\phi_Y = d$. The equation of this structure is:

$$\begin{aligned} X &= W_N^c \cdot (x \cdot W_N^a + y \cdot W_N^b) \\ Y &= W_N^d \cdot (x \cdot W_N^a - y \cdot W_N^b) \end{aligned} \qquad (7)$$

If we divide the inputs by the factor W_N^q, we can pass this factor to the outputs, leading to the equation:

$$\begin{aligned} X &= W_N^{c+q} \cdot (x \cdot W_N^{a-q} + y \cdot W_N^{b-q}) \\ Y &= W_N^{d+q} \cdot (x \cdot W_N^{a-q} - y \cdot W_N^{b-q}) \end{aligned} \qquad (8)$$

This equation is represented in Fig. 3(b). As a result, q is subtracted from the ϕ values at the input and is added to the ϕ values at the output.

B. Moving rotations in the FFT flow graph

If we apply the movement of rotations to the flow graph in Fig. 1, we obtain new flow graphs. The flow graph in Fig. 4 represents a 16-point radix-2^2 FFT. If we compare Figs. 1 and 4, we observe that the radix-2^2 FFT is the result of moving the factor W_N^1 from stage 1 to stage 2 in the butterflies with odd twiddle factors. In general for any N, the radix-2^2 FFT algorithm is the result of moving all the factors except $W_N^{N/4}$ from each odd stage to the next even stage in a radix-2 flow graph. The twiddle factor $W_N^{N/4}$ is a trivial rotation that remains in odd stages.

If we move twiddle factors in Fig. 4 from stage 2 to stage 3, we obtain the flow graph in Fig. 5. This flow graph is a 16-point radix-2 DIT FFT. Therefore, by moving twiddle

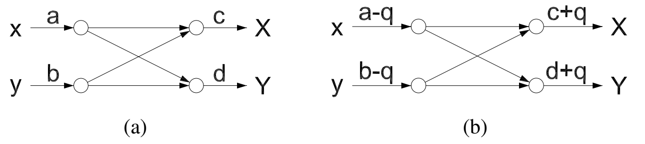


Fig. 3. Movement of rotations in the FFT. (a) Original structure. (b) Structure after movement.

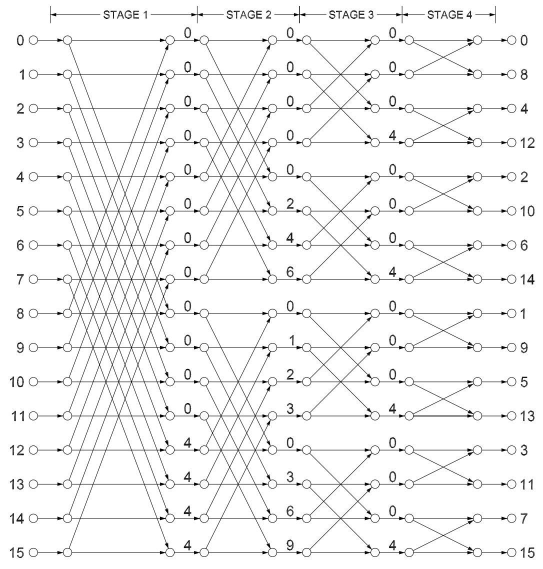


Fig. 4. 16-point radix-2^2 DIF FFT flow graph.

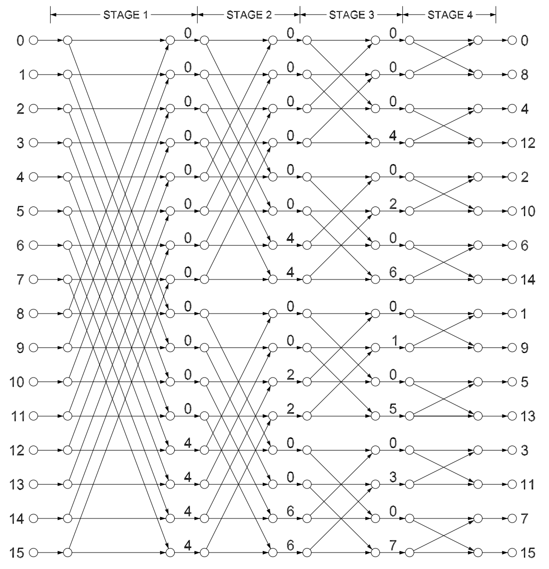


Fig. 5. 16-point radix-2 DIT FFT flow graph with inputs in natural order.

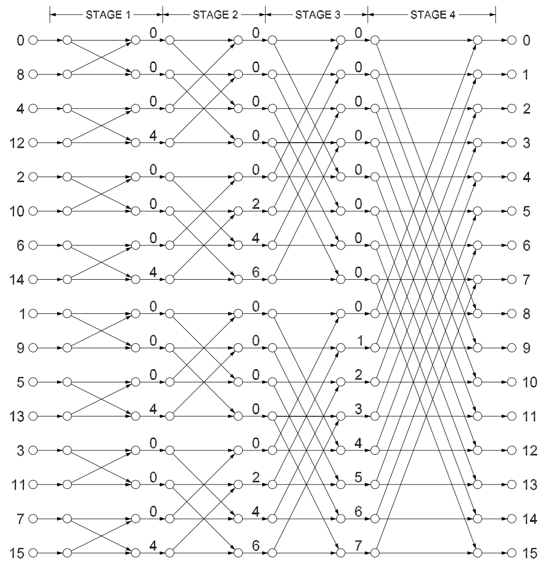


Fig. 6. 16-point radix-2 DIT FFT flow graph.

factors we can obtain different FFT algorithms. Indeed, the FFT algorithms only differ on the values of the twiddle factors.

A more usual representation of the 16-point radix-2 DIT FFT is shown in Fig. 6. In this case, the inputs are depicted in bit-reversed order and the outputs in natural order [18], whereas in Fig. 5 the inputs are in natural order and the outputs are in bit-reversed order. It can be easily checked that both figures are the same algorithm, i.e., the operations in both of them are exactly the same. This refutes the common belief that the inputs of the DIF decomposition are in natural order and those of the DIT decomposition are in bit-reversed order. In fact, the order of the inputs and outputs is always independent of the FFT algorithm: Each FFT algorithm can be represented with any input and output orders.

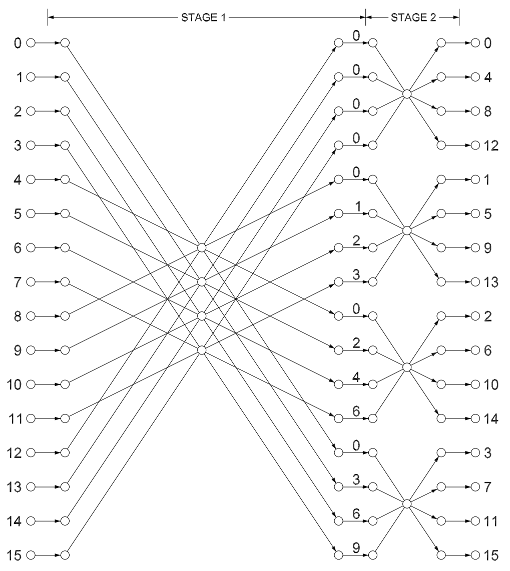


Fig. 7. 16-point radix-4 DIT FFT flow graph.

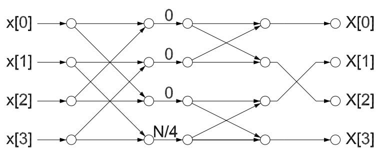


Fig. 8. Radix-4 butterfly.

C. Radix-4, radix-8, etc. algorithms

Fig. 7 shows the 16-point radix-4 DIF FFT flow graph. Each dot in the middle of the stages with four inputs and four outputs represents a radix-4 butterfly, like the one shown in Fig. 8. If we substitute the dots by the corresponding butterfly, we obtain the flow graph shown in Fig. 9. By comparing it to the 16-point radix-2^2 DIF FFT flow graph in Fig. 4 we observe that the radix-2^2 and radix-4 algorithms are indeed the same algorithm, i.e., the operations in both of them are exactly the same. The same holds for radix-8 and radix-2^3, for radix-16 and radix-2^4, etc.

IV. The triangular matrix representation

A. Rotations in an FFT flow graph

The movements presented in previous section can be classified into sets. In any FFT algorithm there is one set of W_N^1 rotations that can be move in all stages except the last one. There are also two set of W_N^2. One of them can be moved between stages 1 and $n-2$, and the other one between stages 2 and $n-1$. And in general, there are m sets of $W_N^{2^{m-1}}$. Each set p, $p = 0 \dots m-1$ can be moved between stages $1+p$ and $n-m+p$. This is shown in Fig. 10 for the case of $N = 16$ points.

In case of the 16-point FFT flow graph, Fig. 11 shows the flow graph of the FFT with decomposed rotations. The algorithm represented in the figure is the radix-2 DIF FFT. Note that the first stage includes the set of W_N^1, one of the 2 sets of W_N^2 and one of the 3 sets of W_N^4. The second stage

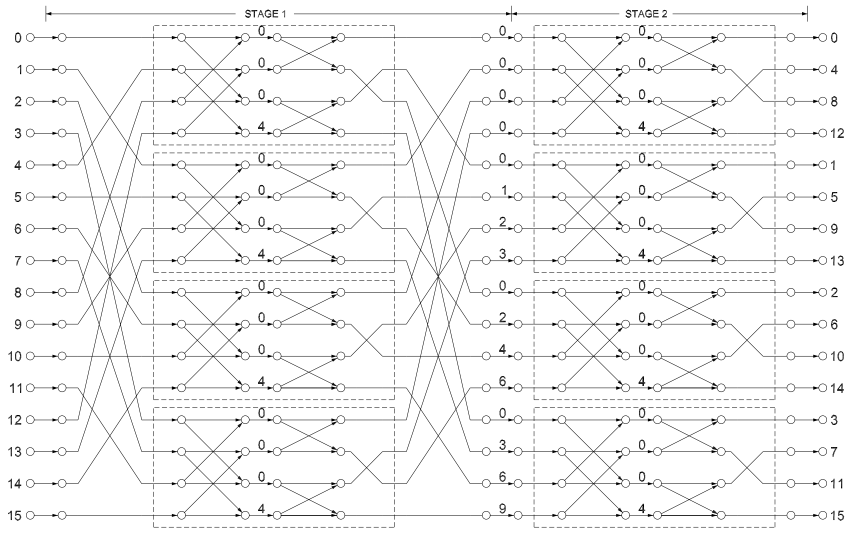


Fig. 9. 16-point radix-4 DIT FFT flow graph with detailed butterflies.

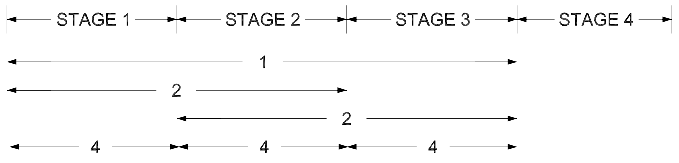


Fig. 10. Rotations in a 16-point FFT.

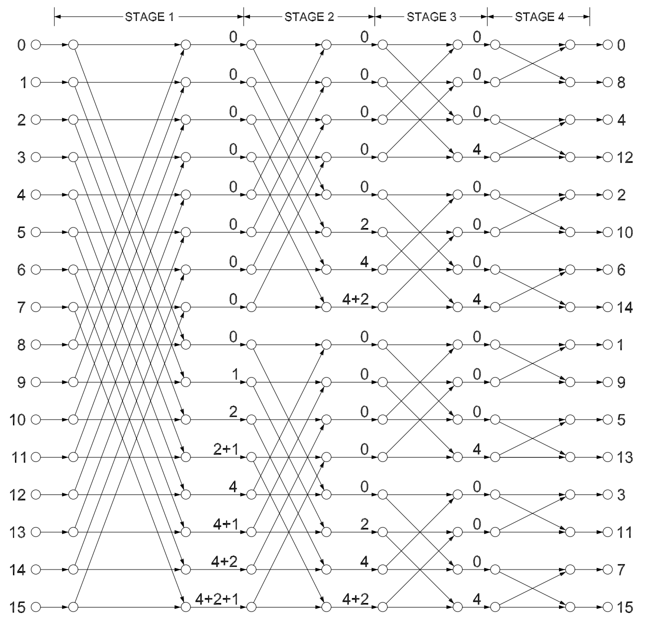


Fig. 11. 16-point radix-2 DIF FFT flow graph with decomposed twiddle factors.

includes the second set of W_N^2 and one set of W_N^4. And the third stage includes one set of W_N^4.

It can be observed that the set of W_N^1 can be placed in the first three stages of the FFT by just moving it through the butterflies. The set W_N^2 in the first stage can be placed in stages 1 and 2, but not in stage 3. The set W_N^2 in the second stage can be placed in stages 2 and 3. And the sets W_N^4 cannot be

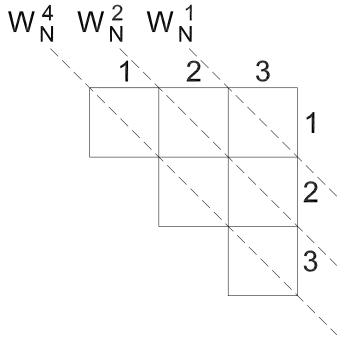


Fig. 12. Triangular matrix representation for FFT algorithms.

moved. Note that these sets of rotations are formed according to their movement possibilities.

Note also that the radix-2 DIF FFT algorithm is the case in which the sets of rotations are placed at the earliest possible stage. If we move the rotations to the most left stage we obtain the radix-2 DIT FFT algorithm, as shown in Fig. 5. And if we move the rotations by W_N^1 and W_N^2 in the first stage to the second one, we obtain the radix-2^2 FFT, as shown in Fig. 4.

Apart from the radix-2 DIF, radix-2 DIT and radix-2^2 algorithms, we can perform other movements in the flow graph to obtain new FFT algorithms. For instance, a new algorithm is obtained when we move the set of W_N^2 in the first stage to the second one and keep the set of W_N^1 in the first stage.

As a result, the number of FFT algorithms that can be represented is large and we need a representation that includes all of them. This representation is presented in the next section.

B. The Triangular Matrix Representation

The triangular matrix representation for FFT algorithms is shown in Fig. 12 for the case of $N = 16$ points. For any N, the triangular matrix representation has a number of rows and columns equal to $n-1$. The upper right corner is the rotation W_N^1, which represents the smallest rotation angle. Starting from the corner, each diagonal corresponds to a twiddle factor.

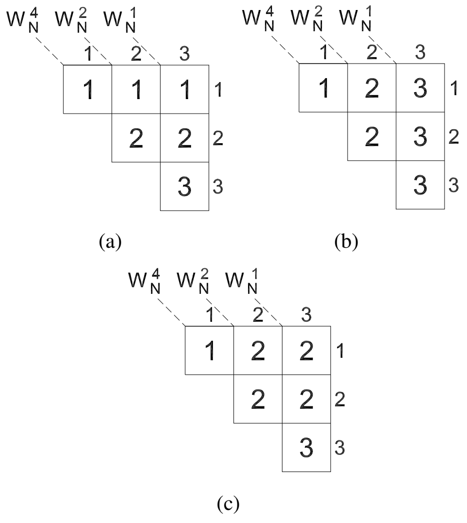


Fig. 13. Triangular matrix representation of typical 16-point FFT algorithms. (a) Radix-2 DIF. (b) Radix-2 DIT. (c) Radix-2^2 DIF.

The second diagonal is the factor W_N^2. The third diagonal is the factor W_N^4 and so on. The last diagonal is the lowest one and corresponds to the trivial rotations $W_N^{N/4}$.

The number of the column represents the highest FFT stage where the rotation can be moved, whereas the number of the row indicates the lowest FFT stage where the rotation can be moved. According to this, W_N^1 can be placed between stages 1 and $n-1$. There is one set of rotations by W_N^2 that can be placed in stages 1 and 2, and another set that can be placed in stages 2 and 3. Finally, for the trivial rotations in the main diagonal the number of the column and the number of the row are the same. This means that each set of trivial rotations is placed in an specific stage and cannot be moved. All of this agrees with the explanation in Section IV-A.

The radix-2 DIF FFT is the case where all the rotations are in the lowest possible stage. This means that the values inside the matrix are equal to the row in which they are. This is shown in Fig. 13(a). Analogously, the radix-2 DIT FFT is the case where all the rotations are in the highest possible stage. This means that the values inside the matrix are equal to the column in which they are. This is shown in Fig. 13(b). Finally, The radix-2^2 DIF FFT is the case in which the rotations of the radix-2 DIF algorithm in odd stages are moved to the next even stage. This is shown in Fig. 13(c).

C. Number of representations

The number of algorithm of the triangular matrix representation can be calculated easily. As we know the number of alternatives for each element in the matrix, the total number of algorithms as a function of n is obtained as

$$\text{Number of algorithms} = \prod_{k=1}^{n-1} (n-k)^k \qquad (9)$$

The total number of algorithms as a function of the FFT size is shown in Table I. Note that the number grows extremely fast, achieving more than 34 thousands for $N = 64$ and more than 24 million for $N = 128$.

TABLE I
NUMBER OF FFT ALGORITHMS OF THE TRIANGULAR MATRIX REPRESENTATION.

N	Number of algorithms
2	1
4	1
8	2
16	12
32	288
64	34560
128	24883200
256	$1.2541 \cdot 10^{11}$
512	$5.0566 \cdot 10^{15}$
1024	$1.8349 \cdot 10^{21}$
2048	$6.6586 \cdot 10^{27}$
4096	$2.6579 \cdot 10^{35}$
8192	$1.2731 \cdot 10^{44}$
16384	$7.9279 \cdot 10^{53}$

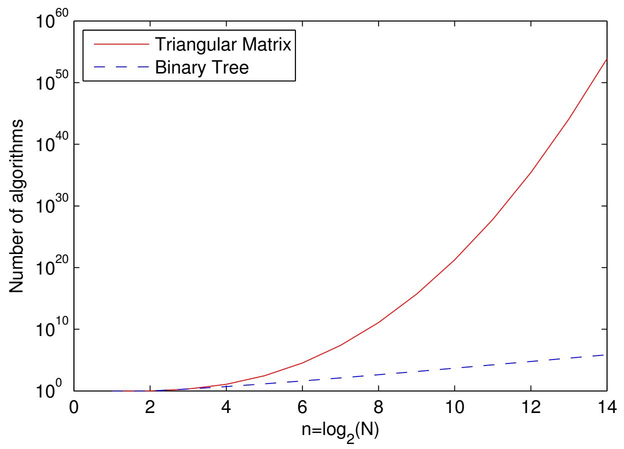


Fig. 14. Number of FFT algorithm as a function of the number of FFT stages.

Fig. 14 compares the number of algorithms representable by binary trees as shown in equation (5) with the number of algorithms representable by triangular matrix as shown in equation (9). Fig. 14 shows that the number of algorithms representable by triangular matrix is significantly larger than that of binary tree. In fact, the binary tree representation is a subset of the triangular matrix representation. This is explained more in detail in the next section.

D. Equivalence to the binary tree representation

Fig. 15 shows various 256-point FFT algorithms in the triangular matrix representation, as well as its equivalent binary tree. Fig. 15(a) is the 256-point radix-2 DIF FFT. As mentioned before, it is characterized by the fact that each element of the matrix is equal to its row. Fig. 15(b) is the 256-point radix-2 DIT FFT. It is characterized by the fact that each element of the matrix is equal to its column. Fig. 15(c) is the 256-point radix-2^2 DIF FFT. It is characterized by the fact that elements in even rows are equal to the row number and elements in odd rows are equal to the next row, except for the trivial twiddle factors $W_N^{N/4}$. Fig 15(d) is the 256-point mixed radix-2^5-2^3 DIF FFT. Fig. 15(e) is the 256-point radix-2^4 FFT. For a 256-point FFT, radix-2^4 forms a balanced binary tree. In a balanced binary tree the DIF and DIT decompositions are

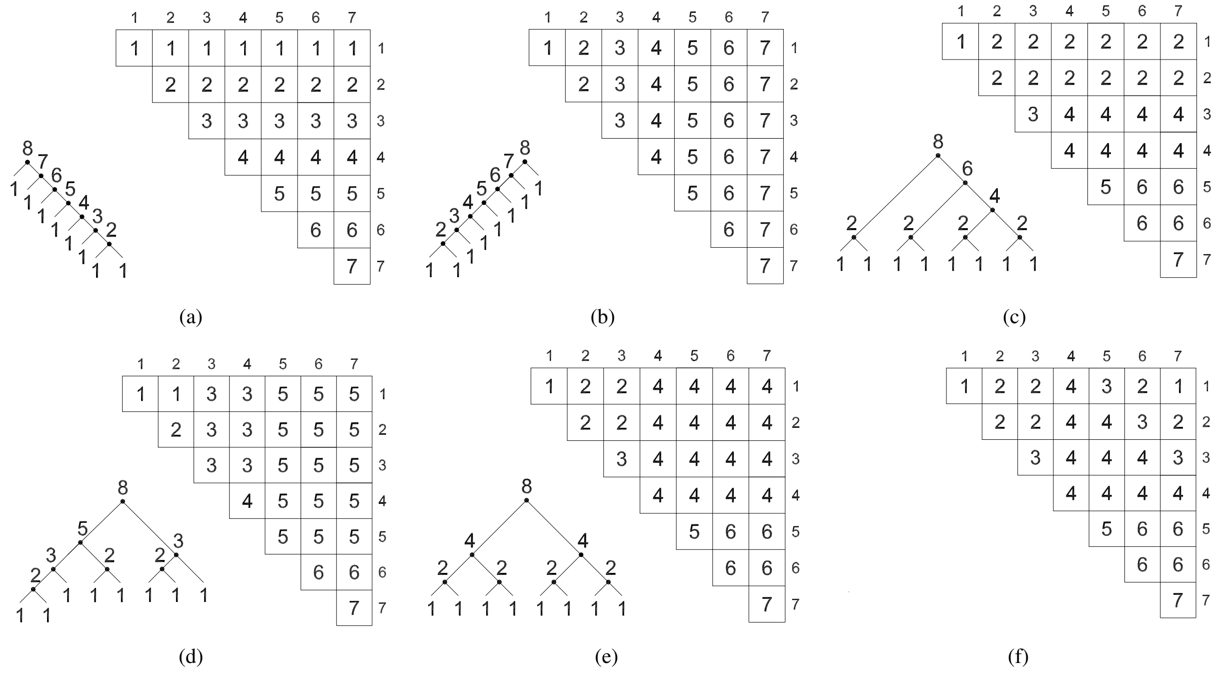


Fig. 15. (a) 256-point radix-2 DIF FFT. (b) 256-point radix-2 DIT FFT. (c) 256-point radix-2^2 DIF FFT. (d) 256-point mixed radix-2^5-2^3 DIF FFT. (e) 256-point radix-2^4 FFT (balanced binary tree). (f) 256-point FFT algorithm not representable with a binary tree.

the same algorithm. The balanced binary tree is characterized by the fact that it is formed by squares of numbers. It includes a big square of $n/2 \times n/2$ elements for the stage $n/2$, a square of $n/4 \times n/4$ elements for stages $n/4$ and $3n/4$, and so on. And it ends with squares of 1 elements for odd stages.

Finally, Fig. 15(f) shows an algorithm that is not representable by a binary tree. The criterion to know if a triangular matrix representation can also be represented by a binary tree is simple: A binary tree is an algorithm in which the triangular matrix representation consists of $n-1$ rectangles or squares of numbers, one for each stage. If a shape in the triangular matrix representation is not a rectangle or a square, or if the number of them is larger than $n-1$, then this representation cannot be represented as a binary tree. Note that, indeed, most algorithms cannot be represented as a binary tree. This highlights the fact that the triangular matrix representation can represent significantly more algorithms than the binary tree representation.

E. Obtaining the twiddle factor values

Fig. 16 shows the flow graph of a 16-point radix-2 DIF FFT that includes the index $I \equiv b_{n-1}b_{n-2}\dots b_1b_0$, where $(\equiv)$ relates the decimal and binary representations of the index. To define the index we always consider the inputs in natural order and the outputs in bit-reversed order. According to this, $\phi_s(I)$ is the value of the rotation at stage s and index I. For instance, in Fig. 16 $\phi_1(9) = 1$ and $\phi_2(15) = 6$.

The values of $\phi_s(I)$ can be obtained directly from the bits of the index $b_{n-1}b_{n-2}\dots b_1b_0$. For this purpose, we include the bits of the index in the triangular matrix representation, as shown in Fig. 17.

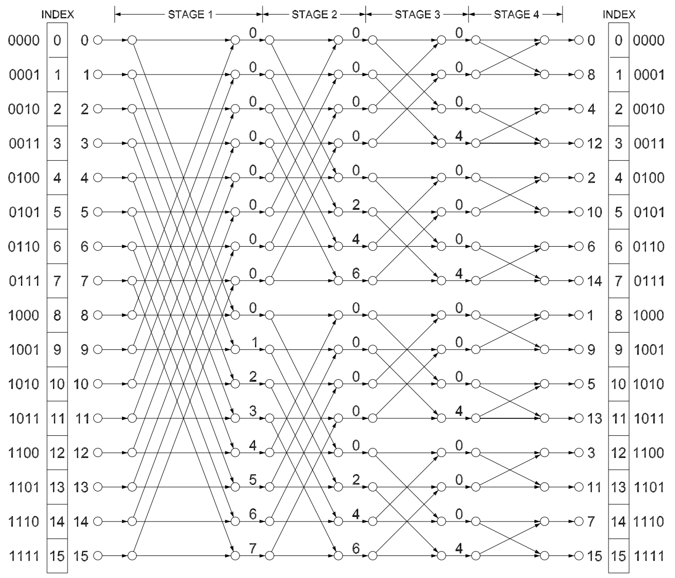


Fig. 16. 16-point radix-2 DIF FFT flow graph with the bits of the index depicted.

According to this, the values of the twiddle factor are obtained from the triangular matrix representation as

$$\phi_s(I) = \sum_{x_{ij}=s} b(i) \cdot 2^{n-1-i} \cdot b(j) \cdot 2^j \qquad (10)$$

where the sum is for all the elements of the matrix x_{ij} whose stage is equal to s, $b(i)$ is the bit in row i and $b(j)$ is the bit in column j. We can also represent equation (10) in binary as

$$\phi_s(I) \equiv \sum_{x_{ij}=s} [b(i)\underbrace{0\dots0}_{n-1-i}] \cdot [b(j)\underbrace{0\dots0}_{j}] \qquad (11)$$

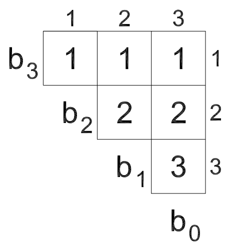


Fig. 17. Triangular matrix representation of the 16-point radix-2 DIF FFT algorithm with the bits of the index.

For the triangular matrix representation in Fig. 17 this leads to

$$\begin{array}{l} \phi_1(I) \equiv b_3 \cdot [b_2 00] + b_3 \cdot [b_1 0] + b_3 \cdot b_0 = b_3 \cdot [b_2 b_1 b_0] \\ \phi_2(I) \equiv [b_2 0] \cdot [b_1 0] + [b_2 0] \cdot [b_0] = [b_2 0] \cdot [b_1 b_0] \\ \phi_3(I) \equiv [b_1 00] \cdot [b_0] \end{array} \tag{12}$$

It can be checked that the values of $\phi_s(I)$ correspond to the twiddle factors in Fig. 16. For instance $\phi_1(9) \equiv b_3 \cdot [b_2 b_1 b_0] = 1 \cdot [001] \equiv 1$ and $\phi_2(15) \equiv [b_2 0] \cdot [b_1 b_0] = [10] \cdot [11] = 110 \equiv 6$.

In the same way, the twiddle factors for the balanced binary tree in Fig. 15(e) are

$$\begin{array}{l} \phi_1(I) \equiv b_7 \cdot [b_6 000000] \\ \phi_2(I) \equiv [b_6 b_7] \cdot [b_5 b_4 0000] \\ \phi_3(I) \equiv [b_5 00] \cdot [b_4 0000] \\ \phi_4(I) \equiv [b_4 b_5 b_6 b_7] \cdot [b_3 b_2 b_1 b_0] \\ \phi_5(I) \equiv [b_3 0000] \cdot [b_2 00] \\ \phi_6(I) \equiv [b_2 b_3 0000] \cdot [b_1 b_0] \\ \phi_7(I) \equiv [b_1 000000] \cdot b_0 \end{array} \tag{13}$$

Note that stages 1, 3, 5 and 7 only take the values 0 and 64, where $W_N^{N/4} = W_N^{64}$ is a trivial rotation. And stage 4 is the stage where most rotation sets are allocated.

In this way it is very easy to obtain the twiddle factors of any FFT algorithm.

F. Twiddle factors of common algorithms

Based on the procedure in previous sections, we can calculate the twiddle factors of different FFT algorithms. For some common radices, the twiddle factor calculations can be represented by simple equations. For radix-2 DIF, the twiddle factors at any stage s and for any n are

$$\phi_s(I) \equiv b_{n-s} \cdot 2^{s-1} \cdot [b_{n-s-1} \dots b_0] \tag{14}$$

Thus, a single equation defines all the twiddle factors of the FFT.

Analogously, for the radix-2 DIT, the twiddle factors are obtained by

$$\phi_s(I) \equiv b_{n-s-1} \cdot 2^{n-s-1} [b_{n-s} \dots b_{n-1}] \tag{15}$$

For radix-2^2 the twiddle factor calculation is reduced to two equations, one for odd stages and one for even stages. For radix-2^2 DIF the twiddle factors at odd stages are

$$\phi_s(I) \equiv b_{n-s} \cdot b_{n-s-1} \cdot 2^{n-2} \tag{16}$$

and those at even stages are

$$\phi_s(I) \equiv [b_{n-s} b_{n-s+1}] \cdot [b_{n-s-1} \dots b_0] \cdot 2^{s-2} \tag{17}$$

TABLE II
ALGORITHMS WITH THE MINIMUM NUMBER OF NON-TRIVIAL ROTATIONS.

N	Algorithm	Non-trivial rotations
16	Radix 2^2 (Fig. 13(c))	8
32	Mixed radix 2^3-2^2 (in its four variants)	28
64	Radix-2^2 DIF and radix-2^2 DIT	76
128	Mixed radix-2^4-2^3	200
256	Balanced binary tree (Fig. 15(e))	480

Finally, for the radix-2^2 DIT algorithm, twiddle factors at odd stages are

$$\phi_s(I) \equiv b_{n-s} \cdot b_{n-s-1} \cdot 2^{n-2} \tag{18}$$

and twiddle factors at even stages are

$$\phi_s(I) \equiv [b_{n-s-1} b_{n-s-2}] \cdot [b_{n-s} \dots b_{n-1}] \cdot 2^{n-s-2} \tag{19}$$

V. APPLICATION TO FFT IMPLEMENTATION

A. Minimum number of non-trivial rotations

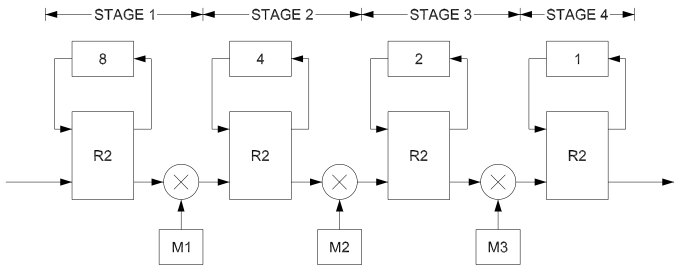
The algorithms with the minimum number of non-trivial rotations are relevant for both software implementation and direct implementation in hardware. In software, these algorithms lead to the smallest number of operations. In a direct hardware implementation these algorithms lead to the implementation with the smallest number of rotators.

The algorithms with the minimum number of non-trivial rotations are shown in Table II for FFT sizes from 16 to 256. For even powers of two the best alternative is to decompose the FFT in groups of radix-2^2 FFTs, while trying to make the tree as balanced as possible. For odd powers of two, mixed radix is used, where one of the radices is 2^3 and the other radix includes the rest of stages. These stages are then decomposed as in the case of even powers of two.

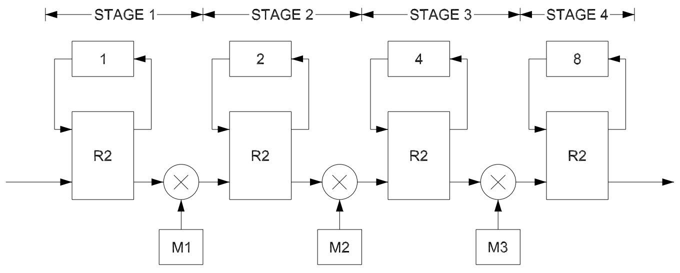
B. SDF FFTs

The proposed triangular matrix representation has direct application to the design of FFT hardware architectures. In this section we present the example of a single-delay feedback (SDF) architecture, shown in Fig. 18(a). This architecture receives input samples in natural order and provides the outputs in bit-reversed order. The architecture is the same for any FFT algorithm. The only difference among algorithms is the content of the memories $M1$, $M2$ and $M3$. The memory at each stage, s, includes the rotations by $\phi_s(I)$, as obtained in the previous section. For instance, in order to implement the radix-2 DIT FFT the twiddle factors in the memories will be those defined by equation (15), and for a radix-2^2 DIF the memories must store the twiddle factors according to equations (16) and (17).

In general, the order of the twiddle factors in memory must respect the order of the data input to the multiplier. In a second example, we may want an architecture in which inputs are in bit-reversed order and outputs in natural order, as shown in Fig. 18(b). In this architecture, the inputs to the multipliers are in bit-reversed order. According to this, the twiddle factors in the memories must be stored in the memories or read from the memories in bit-reversed order. Again, any FFT



(a)



(b)

Fig. 18. Single delay feedback FFT architectures. (a) For inputs in natural order and outputs in bit-reversed order. (b) For inputs in bit-reversed order and outputs in natural order.

algorithm representable by triangular matrices can be mapped to the architecture, being the only difference the content of the memories.

In order to achieve optimized FFT architectures, several previous works [21], [22] show how to proceed. The main idea behind the design is to combine the algorithm selection and the rotator implementation. This leads to simpler rotators at different FFT stages.

VI. Conclusions

This paper has presented the triangular matrix representation. It is based on considering groups of rotations and moving them among FFT stages. As the name of the representation suggests, FFT algorithms are represented by a triangular matrix, where the elements of the matrix indicate in which FFT stage the rotations are allocated. The triangular matrix representation adds degrees of freedom with respect to the binary tree representation. As a result, it can represent significantly more algorithms than the binary tree representation. The triangular matrix representation has direct application to the implementation of the FFT in software and hardware. The algorithms with the minimum number of operations are highlighted in the paper. And in order to obtain optimized hardware designs, the triangular matrix representation can be combined with the rotator implementation.

VII. Acknowledgments

The author wants to thank Dr. Oscar Gustafsson for the valuable discussions about this work.

References

[1] J. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. of Comput.*, vol. 19, pp. 297–301, Apr. 1965.

[2] G. Bergland, "A guided tour of the fast fourier transform," *Spectrum, IEEE*, vol. 6, no. 7, pp. 41–52, July 1969.

[3] W. M. Gentleman and G. Sande, "Fast Fourier transforms -for fun and profit," in *Joint Computer Conf.*, vol. 29, Nov. 1966, pp. 563–578.

[4] J. Cooley, P. Lewis, and P. Welch, "Historical notes on the fast Fourier transform," *Proceedings of the IEEE*, vol. 55, no. 10, pp. 1675–1677, Oct. 1967.

[5] R. C. Singleton, "An algorithm for computing the mixed radix fast Fourier transform," *IEEE Trans. Audio Electroacoustics*, vol. 17, no. 2, pp. 93–103, Jun 1969.

[6] Y. jin Moon and Y. il Kim, "A mixed-radix 4-2 butterfly with simple bit reversing for ordering the output sequences," in *Int. Conf. Advanced Comm. Tech.*, vol. 3, Feb 2006, pp. 1771–1774.

[7] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Parallel Process. Symp.*, Apr. 1996, pp. 766–770.

[8] ——, "Design and implementation of a 1024-point pipeline FFT processor," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 1998, pp. 131–134.

[9] ——, "Designing pipeline FFT processor for OFDM (de)modulation," in *Signals, Systems, and Electronics*, Sep 1998, pp. 257–262.

[10] T. Adiono, M. Irsyadi, Y. Hidayat, and A. Irawan, "64-point fast efficient FFT architecture using radix-2^3 single path delay feedback," in *Int. Conf. Electrical Engineering Informatics*, vol. 2, Aug. 2009, pp. 654–658.

[11] C.-P. Fan, M.-S. Lee, and G.-A. Su, "A low multiplier and multiplication costs 256-point FFT implementation with simplified radix-2^4 SDF architecture," in *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, Dec. 2006, pp. 1935 – 1938.

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

[13] M. Garrido, J. Grajal, M. A. Sánchez, and O. Gustafsson, "Pipelined radix-2^k feedforward FFT architectures," *IEEE Trans. VLSI Syst.*, vol. 21, pp. 23–32, Jan. 2013.

[14] H.-Y. Lee and I.-C. Park, "Balanced binary-tree decomposition for area-efficient pipelined FFT processing," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 4, pp. 889–900, April 2007.

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

[16] F. Qureshi, "Optimization of rotations in FFTs," Ph.D. dissertation, Linköping University, Mar. 2012.

[17] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, 1965.

[18] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*. Prentice Hall, 1989.

[19] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*. Prentice Hall, 1975.

[20] C. V. Loan, *Computational Frameworks for the Fast Fourier Transform*. Siam, 1992.

[21] M. Garrido, R. Andersson, F. Qureshi, and O. Gustafsson, "Multiplierless unity-gain SDF FFTs," *IEEE Trans. VLSI Syst.*, 2016, Accepted for publication.

[22] R. Andersson, "FFT hardware architectures with reduced twiddle factor sets," Master's Thesis, Linköping University, Jun. 2014.

Mario Garrido received the M.S. 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 at Linköping University. Since 2012 he is Associate Professor at the same department.

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.