Generation of All Radix-2 Fast Fourier Transform Algorithms Using Binary Trees

Fahad Qureshi and Oscar Gustafsson
Department of Electrical Engineering, Linköping University
SE-581 83 Linköping, Sweden
E-mail: {fahadq, oscarg}@isy.liu.se

***Abstract*—In this work a systematic method to generate all possible fast Fourier transform (FFT) algorithms is proposed based on the relation to binary trees. The binary tree is used to represent the decomposition of a discrete Fourier transform (DFT) into sub-DFTs. The radix is adaptively changed according to compute sub-DFTs in proposed decomposition. In this work we determine the number of possible algorithms for 2^n-point FFTs with radix-2 butterfly operation and propose a simple method to determine the twiddle factor indices for each algorithm based on the binary tree representation.**

I. Introduction

In many DSP algorithms the discrete Fourier transform (DFT) and inverse DFT is used. Examples include orthogonal frequency-division multiplexing (OFDM) communication systems and spectrometers. An N-point DFT is expressed as

$$X(k) = \sum_{t=0}^{N-1} x(t) W_N^{tk}, \ k = 0, 1, ...N-1, \qquad (1)$$

where $W_N = e^{-j\frac{2\pi}{N}}$ is the twiddle factor, the N:th primitive root of unity with its exponent being evaluated modulo N, and t and k are time and frequency indices, respectively. Various methods for efficiently computing (1) have been the subject of a large body of published literature. They are commonly referred to as fast Fourier transform (FFT) algorithms. Most of these are based on decomposing the N-point DFT in $b = \log_r N$ stages, where r is the radix, leading to an asymptotic complexity reduction from $O(N^2)$ for a direct computation of (1) to $O(N \log N)$ for the FFT.

Usually, a signal flow graph (SFG) is used to illustrate the arithmetic operations of FFT. Each stage of the SFG consists of butterfly and multiplication operations. In radix-r, a butterfly has r inputs and r outputs, and it computes an r-point DFT. For example, the SFG of a 32-point FFT with radix-2 butterflies is shown in Fig. 1. In Fig. 1, twiddle factor multiplication are shown based on stage and row number of the SFG. For example, $W_{3,6}$, where 3 and 6 are stage and row, respectively. The selection of r has great influence on the resulting FFT algorithm. For smaller values of r more processing stages are required, but the butterfly operations are simpler.

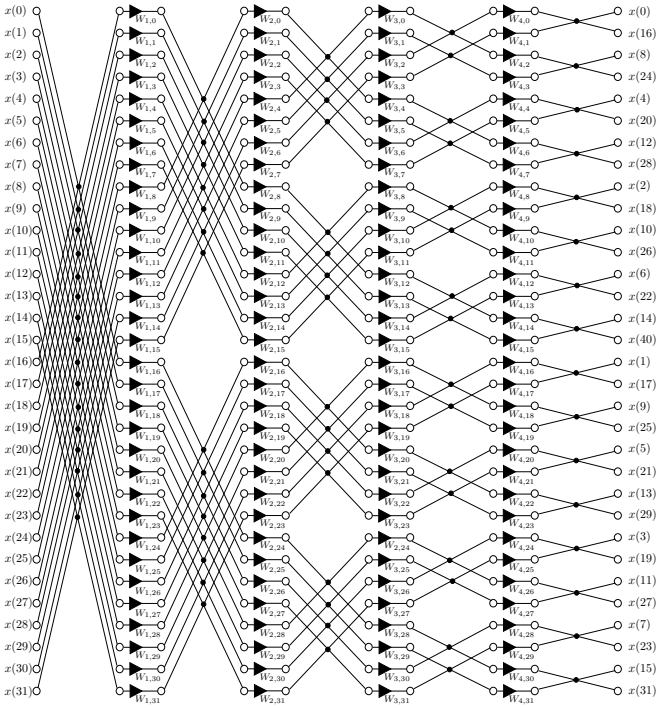

Fig. 1. Generalized radix-2 32-point FFT signal flow graph.

Different methods have been proposed to carry out the decomposition. The initially proposed ones are decimation in time (DIT) and decimation in frequency (DIF). DIT is based on decomposing the input sequence into smaller sub-sequences while in DIF the output sequence is decomposes into smaller sub-sequences [1], [2].

The complexity of an algorithm depends on the number of trivial and non-trivial twiddle factor multiplications. The twiddle factors, which perform actually a rotation of the phase, can be implemented by using various methods, e.g., general complex multipliers with look-up tables storing the coefficient values, complex constant multiplication [3], or CORDIC (COordinate Rotation DIgital Computer) [4]. To improve the DIF and DIT algorithm, the Radix-2^2 algorithm was proposed [5]. The algorithm has the same number of non-trivial multiplications as a radix-4 algorithm, but it can be

mapped to radix-2 butterflies. This was further on extended to Radix-2^3 [5] and 2^4 [6], with the goal to reduce the resolution of the twiddle factor multiplications in certain stages. Recently, a general approach for Radix-2^i, up to $i = 14$, was proposed in [7].

Also recently, an algorithm aiming at minimizing the size of the twiddle factor look-up tables was proposed in [8]. This method is based on a binary tree representation, where the goal was to find a balanced binary tree. However, as shown in [8] DIF, DIT and R-2^2 can also be represented with a binary tree.

All the algorithms mentioned above can be mapped to an SFG as shown in Fig. 1. The difference is the twiddle factors for the different multipliers. The twiddle factors will determine several things for the resulting realization. In a pipelined architectures, it will determine the size and the switching activity [9] of the twiddle factor memories. It will also determine the possible use of constant complex multipliers, which can be used if the resolution is low. In addition, it will impact the round-off noise (relating to the number of non-trivial multiplications) and the accuracy of the transform [10]. However, to the best of our knowledge, there has been no work on obtaining all algorithms for 2^n-point FFTs with radix-2 (or any other size) butterflies. Depending on the cost measure for the architecture used, it is possible to simply select the algorithm with the minimum cost once all are available.

In this paper we propose an approach using a binary tree to find all 2^n-point FFT algorithms with radix-2 butterfly. We also present a generalized approach to calculate a twiddle factors of all the stages of the FFT for any algorithm which is represented using a binary tree.

The paper is organized as follows. In Section II, we discuss the binary tree representation of the Cooley-Tukey decomposition and explain the number of possible algorithms using binary trees for 2^n-point FFTs and Section III presents the twiddle factor index computation. Section IV illustrates the index generation with an example. Finally, some conclusions are drawn in Section V.

II. Binary Tree Representation of Cooley-Tukey Algorithm

The Cooley-Tukey FFT algorithm is based on the decomposition of an N-point FFT into smaller FFTs of length P and Q. This can be expressed as

$$X[k_1 + Pk_2] = \sum_{t_2=0}^{Q-1} \left[\left(\sum_{t_1=0}^{P-1} x[Qt_1 + t_2] W_P^{t_1 k_1} \right) W_N^{t_2 k_1} \right] W_Q^{t_2 k_2}, \quad (2)$$

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

In this work, N, P, and Q are considered to be powers of 2, i.e., $N = 2^n = 2^{p+q}$, $P = 2^p$, and $Q = 2^q$, where p and q are positive integers. Here, an N-point DFT is decomposed into Q P-point DFTs of and P Q-point DFTs. 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 P and Q-point DFTs are again divided into smaller DFTs, down to the required radix.

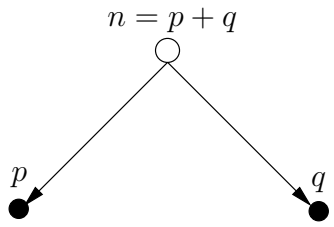

Fig. 2. Illustration of binary tree corresponding to (2).

A binary tree representation is used to represent the FFT algorithms [8]. Each node has at most two leaves which represents the inner and outer FFTs. An example of a binary tree is shown in Fig. 2 corresponding to (2). Each rooted node in the binary tree, represents the computation of an N-point DFT and also has maximum twiddle factor resolution, i.e., a W_N multiplier. The rooted node is labeled by the value $n = \log_2(N)$. The two leaves of this node are labeled by decomposition of n into two positive integer p and q. Hence, the left branch corresponds to $P = 2^p$-point DFTs and the right branch to $Q = 2^q$-point DFTs. The decomposition then continues until the leaf node matches the radix. In the binary tree, the nodes, except for leaf nodes, corresponds to twiddle factor multiplication stages of FFT algorithm. The leaf nodes corresponds to butterfly operations.

All algorithms which are based on systematic decomposition as in (2), where each inner and outer DFT is decomposed in the same way for all DFTs, can be represented using a binary tree. These include DIT, DIF, R-2^i, and the balanced binary tree among others.

A. Number of Possible FFT Algorithms for 2^n-Point FFTs

The number of algorithms to compute a radix-2 2^n-point FFT using different Cooley-Tukey decomposition schemes is the same as the number of rooted binary trees with $n-1$ internal nodes. This is given by the $n-1$:th Catalan number as [11]

$$\#_{\text{FFTs}}(n) = C_{n-1}, \quad (3)$$

where

$$C_n = \frac{(2n)!}{(n+1)!n!}. \quad (4)$$

Consider $n = 5$ ($N = 32$), which has fourteen different algorithms to implement a 32-point FFT. The binary tree representations of all possible algorithms are shown in Fig. 3. From the DFT calculation point of view, all possible decompositions represent the same transform. The generated algorithms have the same number of butterfly operations with a different

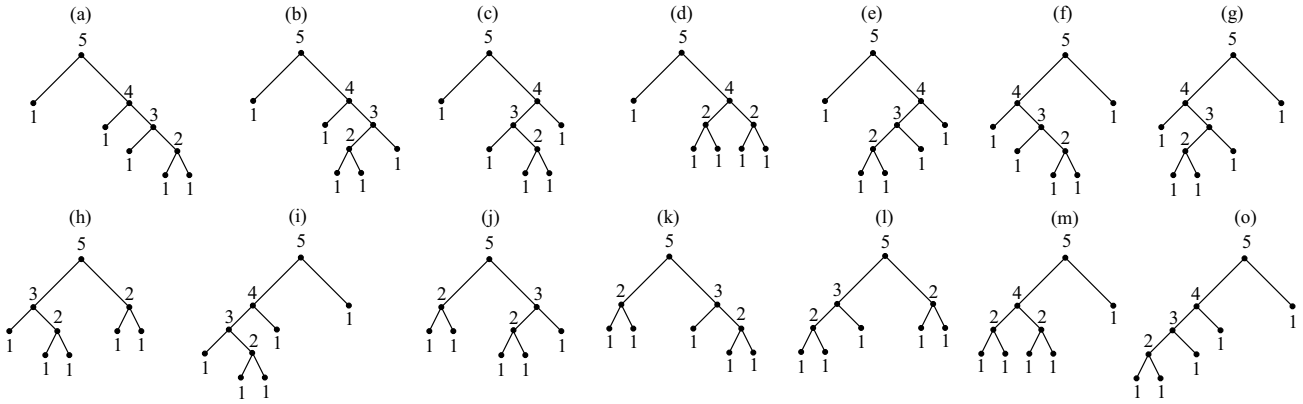

Fig. 3. Binary tree diagram of all possible algorithm for $N = 32$.

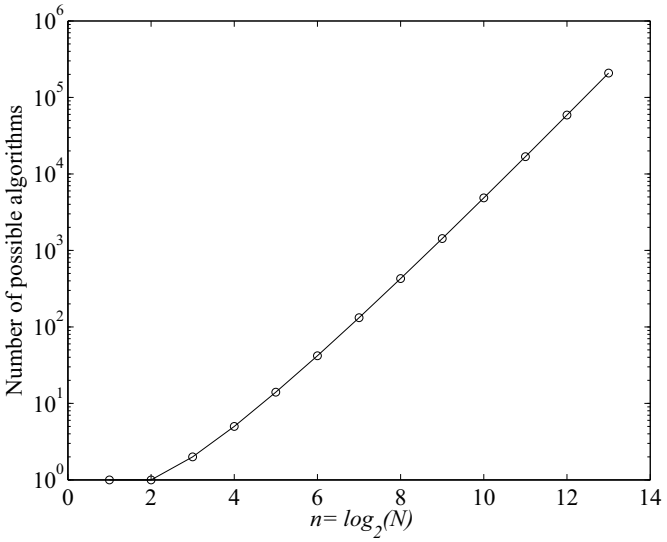

Fig. 4. Number of possible algorithms for 2^n-point FFTs.

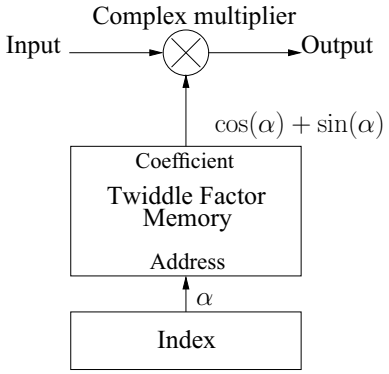

Fig. 5. General twiddle factor multiplication.

twiddle factor multiplier for each stage. However, they differ in the measures listed in the introduction. The number of possible algorithms increase rapidly as the number of FFT points are increased. Figure 4 shows the number of possible algorithms from 4-point to 8192-point FFTs.

III. Twiddle Factor Index Generation

A general twiddle factor multiplication relies on index generation, a twiddle factor memory, and a complex multiplier which as shown in Fig. 5.

Consider the $n = p + q$ root node, which is decomposed into two leaves, p and q, then these leaves are further decomposed into sub-leaves. The decomposition continues until the leaf node matches the radix. The output sequence of a DFT is the bit-reversed input sequence of the DFT. The above observation leads to the following method to determine the twiddle factor indices based on a binary tree representation.

The row number is determined by an n-bit number, which can represent all numbers between 0 to $N-1$. Each leaf node corresponds to one bit of this number. The leaf node mapping starts from the left leaf node with the most significant bit and moves towards right leaf node.

In the index generation, the n-bit number determining the row in Fig. 1 is for each node decomposed into four bit groups, M, P, Q and O, with lengths m, p, q, and o, respectively, such that $m + p + q + o = n$. The length of each group can be determined from the binary tree representation of the FFT algorithm. Consider an arbitrary node in a binary tree, illustrated in Fig. 6. P and Q are simply determined by the decomposition of that node. M is the number of leaf nodes to the left of the leaf nodes in the current subgraph, while O is the corresponding number of leaf nodes to the right.

First, the stage in relation to the SFG in Fig. 1 is determined by $m + p$. The resolution of the twiddle factor multiplier is determined by 2^{p+q}. Finally, the index can be determined by multiplying the bit-reversed bits in P with the bits in Q as illustrated in Fig. 7.

IV. Design Example

To illustrate the properties of the proposed method, we initially consider the generation of twiddle factor indices for

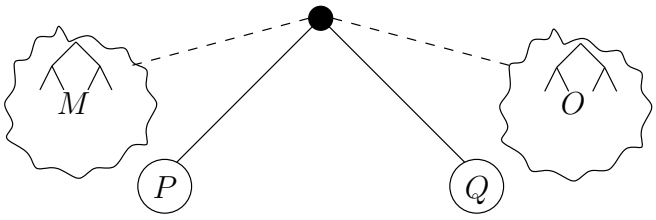

Fig. 6. An arbitrary node in a binary tree representation with relations used for index mapping.

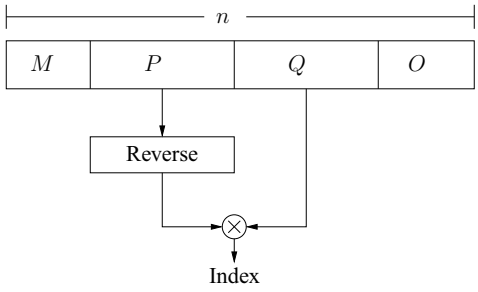

Fig. 7. Block diagram of the proposed index generation.

the 14 possible algorithms for $N = 32$. In Table I, the resulting bits of the row counter to consider in the different stages for the algorithms in Fig. 3 are shown. The bits corresponding to the gray dots (the P-bits) are bit-reversed and multiplied with the bits corresponding to the black dots (the Q-bits) to obtain the twiddle factor index. The white dots corresponds to M- and O-bits and are not used for the index mapping. In the final column, the resulting algorithm is related to the previously discussed algorithms.

To calculate the index of $W_{3,6}$ twiddle factor multiplication in the SFG in Fig. 1 for the algorithm in Fig. 3(g) the following is done. As we consider the third stage, we should consider a node that have three leaf nodes to the left ($m+p = 3$). This is the node with the number 3 in Fig. 3(g). For this node we have $m = 1$, $p = 2$, $q = 1$, and $o = 1$. The resolution of twiddle factor is $2^{p+q} = 8$. The index is determined be writing the row number in binary form with $n = 5$ bits as $6_{10} = 00110_2$. This is then decomposed in $P = 01_2$ and $Q = 1_2$. Bit-reversing P and multiplying with Q, we obtain the index as $10_2 \times 1_2 = 2$. Hence, the twiddle factor $W_{3,6}$ for the algorithm in Fig. 3(g) is $W_8^2 = -j$. This can also be obtained from the (g)-row in Table I.

V. Conclusions

In this work, we have proposed a method to generate all possible algorithms for 2^n-point FFTs using a binary trees. The generated algorithms have different properties, which affects complexity, accuracy and twiddle factor memory switching activity. A method for obtaining the twiddle factor indices based on the binary tree representation is also proposed. Therefore, the proposed twiddle factor indices method leads to generate the address of twiddle factor memory.

TABLE I
Bit representation of address generation.

Fig. 3	First	Second	Third	Fourth	Comments
(a)	W_{32} ●●●●●	W_{16} ○●●●●	W_8 ○○●●●	W_4 ○○○●●	R-2 DIF
(b)	W_{32} ●●●●●	W_{16} ○●●●●	W_4 ○○●●○	W_8 ○○●●●	
(c)	W_{32} ●●●●●	W_8 ○●●●○	W_4 ○○●●○	W_{16} ○●●●●	
(d)	W_{32} ●●●●●	W_4 ○●●○○	W_{16} ○●●●●	W_4 ○○○●●	
(e)	W_{32} ●●●●●	W_4 ○●●○○	W_8 ○●●●○	W_{16} ○●●●●	
(f)	W_{16} ●●●●○	W_8 ○●●●○	W_4 ○○●●○	W_{32} ●●●●●	
(g)	W_{16} ●●●●○	W_4 ○●●○○	W_8 ○●●●○	W_{32} ●●●●●	
(h)	W_8 ●●●○○	W_4 ○●●○○	W_{32} ●●●●●	W_4 ○○○●●	R-2^2 DIT
(i)	W_8 ●●●○○	W_4 ○●●○○	W_{16} ●●●●○	W_{32} ●●●●●	
(j)	W_4 ●●○○○	W_{32} ●●●●●	W_4 ○○●●○	W_8 ○○●●●	
(k)	W_4 ●●○○○	W_{32} ●●●●●	W_8 ○○●●●	W_4 ○○○●●	R-2^2 DIF
(l)	W_4 ●●○○○	W_8 ●●●○○	W_{32} ●●●●●	W_4 ○○○●●	Balance binary tree
(m)	W_4 ●●○○○	W_{16} ●●●●○	W_4 ○○●●○	W_{32} ●●●●●	
(n)	W_4 ●●○○○	W_8 ●●●○○	W_{16} ●●●●○	W_{32} ●●●●●	R−2 DIT

References

[1] S. J. Mitra, *Digital Signal Processing*, McGraw-Hill, 2006.

[2] L. Wanhammar, *DSP Integrated Circuits*, Academic Press, 1999.

[3] F. Qureshi and O. Gustafsson, "Low-complexity reconfigurable complex constant multiplication for FFTs," in *Proc. IEEE Int. Symp. Circuits Syst.*, Taipei, Taiwan, May 24–27, 2009.

[4] M. Garrido and J. Grajal, "Efficient Memoryless CORDIC for FFT Computation," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 2, Apr. 2007, pp. II-113–116.

[5] S. He and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)Modulation," in *Proc. IEEE URSI Int. Symp. Signals, Systems and Elect.*, 1998, pp. 257–262.

[6] J. -E. Oh and M. -S. Lim, "New radix-2 to the 4th power pipeline FFT processor," *IEICE Trans. Elect.*, vol. E88-C, no. 8, pp. 694–697, Aug. 2005.

[7] 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, July 2009.

[8] H. -Y Lee and I. -C Park, "Balanced binary-tree decomposition for area-efficient pipelined FFT processing," *IEEE Trans. Circuits and Syst. I*, vol. 54, no. 4, pp. 889–900, April 2009.

[9] F. Qureshi and O. Gustafsson, "Twiddle factor memory switching activity analysis of Radix-2^2 and equivalent FFT algorithms," in *Proc. IEEE Int. Symp. Circuits Syst.*, Paris, France, May 30-June 2 2010, pp. 4145–4148.

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

[11] R. P. Stanley and S. Fomin, *Enumerative Combinatorics*, vol. 2 Cambridge university press, 1999.