New Identical Radix-2^k Fast Fourier Transform Algorithms

Fahad Qureshi
Pervasive Computing Department
Tampere University of Technology
Tampere, Finland
Email: fahad@tut.fi

Jarmo Takala
Pervasive Computing Department
Tampere University of Technology
Tampere, Finland
jarmo.takala@tut.fi

***Abstract*—The radix-2^k fast Fourier transform (FFT) algorithm is used to achieve at the same time both a radix-2 butterfly and a reduced number of twiddle factor multiplication. In this paper we present a new identical radix-2^k FFT algorithms, which has same number of butterfly and twiddle factor multiplication. The difference is only in twiddle factor stage location in signal flow graph (SFG). Further, analyze these algorithms and is shown that the round-off noise of identical radix-2^2, radix-2^3, and radix-2^4 FFT algorithms at output is reduced 27%, 8%, 3% respectively.**

***Keywords*-Fast Fourier Transform (FFT), Round-off noise, Radix-2^k.**

I. Introduction

The discrete Fourier transform (DFT) is an essential digital signal processing algorithm to convert the signal between time and frequency domain. The DFT is part of numerous system in a large variety of applications, such as biomedical processing, image processing, telecommunication, and wireless communication systems.

In order to calculate DFT efficiently, Cooley and Tukey rediscovered the algorithm called FFT. The FFT reduces a complexity from $O(N^2)$ to $O(N \log N)$ by using of symmetry and periodicity properties of the twiddle factors [1]. Since last decade, numerous FFT algorithms have been proposed, such as radix-2, radix-4, radix-8, mixed radix, and split radix [2]–[5]. Among them radix-2 FFT algorithm is one of most popular solution because it requires simple butterfly operation, but higher number of twiddle factor multiplications. Nevertheless higher radix requires less number of twiddle factor multiplication as compared to radix-2 at cost of complex architecture of radix. [6], the first radix-2^2FFT algorithm was proposed, which has same complexity as radix-4. In addition, radix-2^3, radix-2^4, and radix-2^k FFT algorithms were proposed in [7]–[9] to get the advantage of higher radix by using radix-2. Similarly, the number of possible radix-2 FFT algorithms using binary tree have been proposed in [10], which included all the previous FFT algorithms as well.

These FFT algorithms are getting much attention, when implemented on pipelined architecture. There are two basic type of pipelined architectures: Single delay feedback (SDF) and single delay commutator (SDC). However, SDF FFT pipelined architecture is most popular among them because it has moderate complexity with simple control [6]–[9]. The accuracy of architecture is an important metric for system performance [11]. Usually, FFT algorithms are also implemented on hardware with finite word length [12]. Because it is not possible to keep infinite word length of operations and coefficients. As a result, coefficients and internal signals are quantized with certain bits in binary format. The number of bits are depending upon on the requirement of accuracy which is directly proportional to the hardware complexity. In fixed point arithmetic, generally a multiplication may produce an error due to rounding or truncation, which are often called rounding errors even if truncation is used [12]. The rounding or truncation error happens when the twiddle factors have different value then $\{\pm 1, \pm j\}$, so called non-trivial multiplication. Which leads to an increased intermediate word length after multiplication, then it must be reduced for next stage. A number of papers have been published to analyze the fixed point arithmetic effects of FFT algorithms [13]–[15].

In this work, identical radix-2^k FFT algorithms are presented. These algorithms have similar butterfly structure and data input/output sequence as radix-2^k. The difference is twiddle factor stage location in a signal flow graph except one case, where the complexity of twiddle factor multiplication is also reduced. This variation of the FFT algorithm is affect on the overall round-off error at output. Following the method used in [15], we calculate the number of non-trivial multiplications and round-off noise for analysis of identical radix-2^k FFT algorithms. The proposed method applies on radix-2^2, radix-2^3, and radix-2^4 FFT algorithms as a case studies.

The paper is organized as follows. Section II briefly describes the background. Section III presents the related research in the field. Section IV generates the identical radix-2^k FFT algorithms. Section V provides a mechanism to generate the exponent of the twiddle factor. Section VI explains the model of round-off noise. Section VII shows quantitative results for analysis of identical radix-2^k FFT. Finally, Section VIII summarizes the main conclusions of paper.

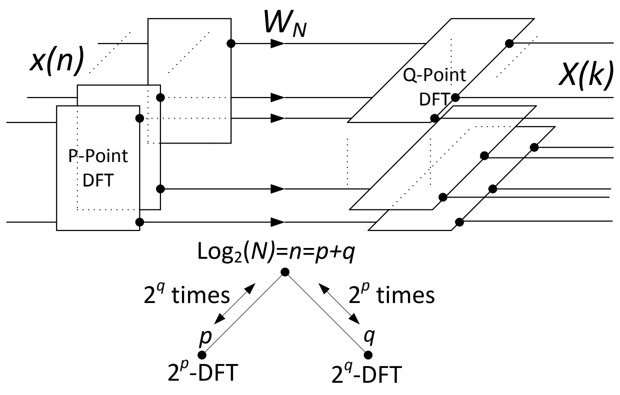


Figure 1. Decomposition scheme and binary tree of Cooley-Tukey FFT algorithm.

II. Background

An N-point DFT can be expressed as

$$X(k)=\sum_{n=0}^{N-1} x\left(n\right) W_N^{\phi},\ k=0,1,\ldots,N-1, \tag{1}$$

where n is the time index, k is the frequency index and $W_N = e^{-j\frac{2\pi}{N}}$ is the twiddle factor and defined as

$$W_N^{\phi}=\cos\left(\frac{2\pi\phi}{N}\right)-j\sin\left(\frac{2\pi\phi}{N}\right). \tag{2}$$

The Cooley-Tukey FFT algorithms are based on unified approach, which is called divide and conquer. It can be expressed as

$$X\left[Qk_1+k_2\right] = \underbrace{\sum_{n_1=0}^{P-1}\left[\underbrace{\left(\sum_{n_2=0}^{Q-1} x\left[n_1+Pn_2\right]W_Q^{\phi_2}\right)}_{Q-point FFT}\underbrace{W_N^{\phi}}_{Twiddle factor}\right]W_P^{\phi_1}}_{P-point\ DFT},$$
$$0\le n_1,k_1\le P-1; 0\le n_2,k_2\le Q-1, \tag{3}$$

where the two summations are indexed by n_1 and n_2, which 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^{ϕ}, which is called 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 $W_N^{\phi}=e^{-j\frac{2\pi}{N}\phi}$.

A. Binary Tree

A binary tree representation of the FFT algorithm was first proposed in [16]. Furthermore, this representation has been used to generate the number of possible FFT algorithms [10]. In binary tree representation, a node with a value n represents a 2^n-point DFT. Each node has at most two leaves which represent the inner and outer DFTs. An example of a binary tree is shown in Fig. 1 corresponding to the scheme of DFT. The left leaf corresponds to number of 2^q DFTs of 2^p-point whereas the right leaf represents set of 2^p DFTs of 2^q-point. This decomposition applies recursively until the leaf node matches the radix. When the binary tree is finalized, all leaf nodes correspond to the radix and internal nodes correspond to twiddle factor multiplication stages (s), which connect the two decomposed DFTs.

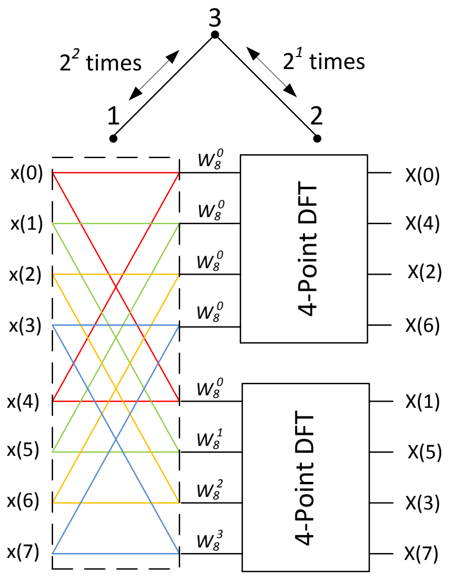


Figure 2. First decomposition of 8-point FFT.

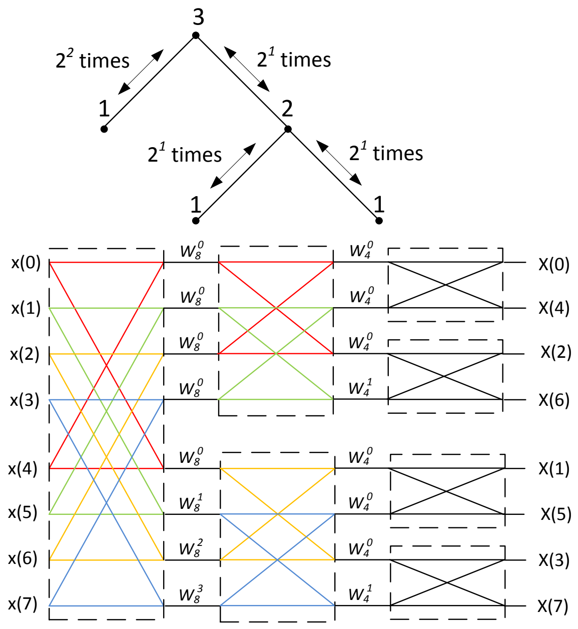
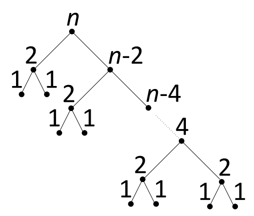
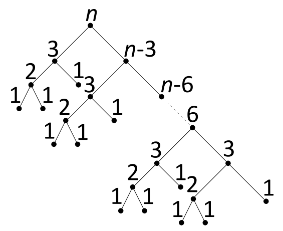
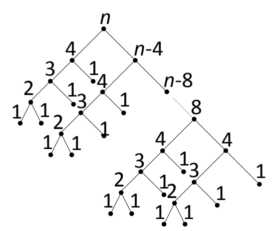


Figure 3. Second decomposition of 8-point FFT.


(a) Radix-2^2


(b) Radix-2^3


(c) Radix-2^4

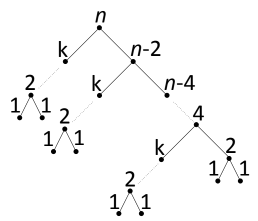

(d) Radix-2^k

Figure 4. Binary tree representation of radix-2^k FFT algorithms for k=2, 3, 4.

B. Binary Tree Relation with Signal Flow Graph

A signal flow graph (SFG) is a one of the most popular graphical representations of FFT algorithms. The flow graph consists of a series of stages, where each stage includes additions, subtractions and complex multiplications. An example of 8-point FFT decomposition is shown in two steps. At first, it decomposes into 2-point and 4-point, this ends up with a four times 2-point computation, twiddle factor multiplication, and two times 4-point in next stage. All these information is mapped on the binary tree and SFG are shown in Fig. 2, where four different of butteries shows the four times 2-point FFT. At second stage, 4-point FFT is further decomposed in 2-point and 2-point. However, there are two 4-point blocks so it applies on both blocks as shown in Fig. 3.

III. Related Work

Generally, a selection of radix has a large impact on the complexity of FFT algorithm. Higher radices help in reducing the number of stages. However, complexity of the butterflies increases [17]. The radix-2^2 FFT algorithm is one of most popular FFT algorithm, where the number of non trivial multiplications is exactly the same as the radix-4 [6]. The binary tree diagram of radix-2^2 is depicted in Fig. 4(a), which shows that the alternate twiddle factor multiplication stages have the W_4 twiddle factor multiplication. The W_4 twiddle factor multiplication involves only trivial multiplications.

Furthermore, radix-2^3 and radix-2^4 can be used to implement the radix-8 and radix-16 butterflies by radix-2 [7] as shown in Fig. 4(b) and (c) respectively. The radix-2^3 and radix-2^4 FFT algorithms have the same twiddle factor multiplication stages as the radix-8 and radix-16 butterfly FFT algorithm respectively. Finally, the more generalized FFT algorithms based on radix-2^k was proposed as shown in Fig. 4(d) [8].

These algorithms have same butterfly architecture regardless of value of k. However, only the twiddle factor multiplication is varied by a value of k. Table I shows the twiddle factor multiplication stages of different k value, which includes radix-2^2, 2^3, 2^4 and 2^5.

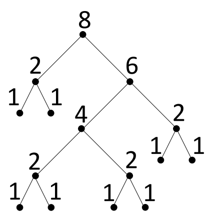
IV. New Identical Radix-2^k FFT Algorithms

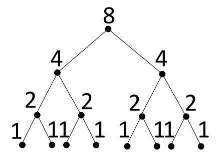
This section presents the identical radix-2^k FFT algorithms, which are generated by using binary tree representation. Now based on the binary tree presentation of Cooley- Tukey FFT algorithm, we generate the number of FFT algorithms which are identical to radix-2^k. These algorithms can be generated by splitting N-point FFT into 2^k-point and $N-2^k$-point. Apply this decomposition recursively until all sizes will become 2^k. This decomposition can apply in different ways to get identical radix-2^k FFT algorithms. Then apply radix-2 decomposition on all 2^k-point FFTs as shown in Fig 4(d). With this explanation, the binary trees of the identical

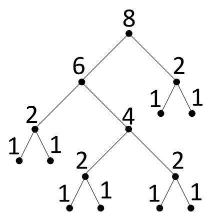
Table I
Twiddle factor multiplication stages of radix-2^k FFT algorithms.

Radix	Stage number 1	2	3	4	5	6	...	s
2^2	W_4	W_N	W_4	$W_{\frac{N}{4}}$	W_4	$W_{\frac{N}{16}}$	...	W_4
2^3	W_4	W_8	W_N	W_4	W_8	$W_{\frac{N}{8}}$	...	W_8
2^4	W_4	W_8	W_{16}	W_N	W_4	W_8	...	W_{16}
2^5	W_4	W_8	W_{16}	W_{32}	W_N	W_4	...	W_{32}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

radix-2^2 FFT algorithms for considering $N = 256$ are shown in Fig. 5. These are four cases of FFT algorithms having the same butterfly structure, where the only difference is in twiddle factor multiplication stages which is tabulated in Table II. It is clearly shown in Table II that every second stage of twiddle factor multiplication is W_4, which is similar to radix-2^2.


(a) Case-I


(b) Case-II


(c) Case-III

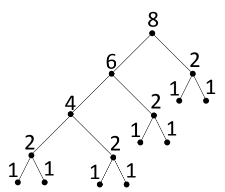
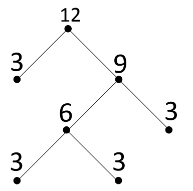

(d) Case-IV

Figure 5. Binary tree representation of identical radix-2^2 for N=256.

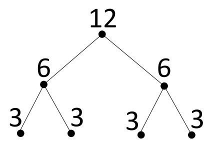
Similarly, identical radix-2^3 and radix-2^4 FFT algorithms are generated for given transform length. Figures 6 and 7 illustrate the binary tree of four different cases for each FFT algorithms respectively. The figures show the decomposition iteration up to length 2^k FFT because rest of the decomposition is similar in all

Table II
TWIDDLE FACTOR MULTIPLICATIONS OF IDENTICAL RADIX-2^2 FOR N=256.

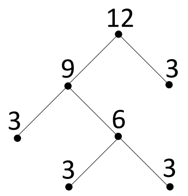
	Stage number						
Case	1	2	3	4	5	6	7
I	W_4	W_{256}	W_4	W_{16}	W_4	W_{64}	W_4
II	W_4	W_{16}	W_4	W_{256}	W_4	W_{16}	W_4
III	W_4	W_{64}	W_4	W_{16}	W_4	W_{256}	W_4
IV	W_4	W_{16}	W_4	W_{64}	W_4	W_{256}	W_4



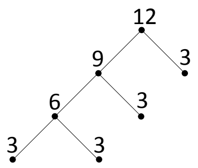
(a) Case-I



(b) Case-II

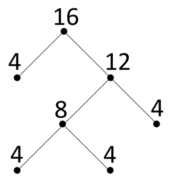


(c) Case-III

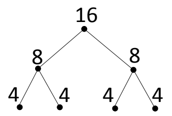


(d) Case-IV

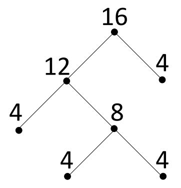
Figure 6. Binary tree representation of identical radix-2^3 for N=4096.



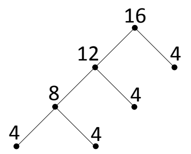
(a) Case-I



(b) Case-II



(c) Case-III



(d) Case-IV

Figure 7. Binary tree representation of identical radix-2^4 for $N = 64K$.

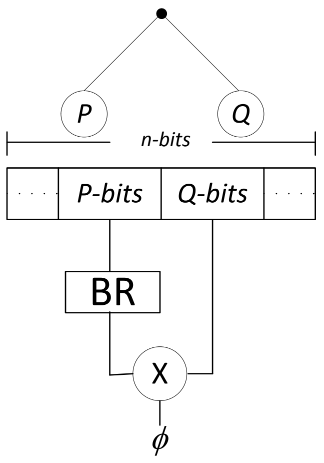


Figure 8. Block diagram of twidde factor exponent generation.

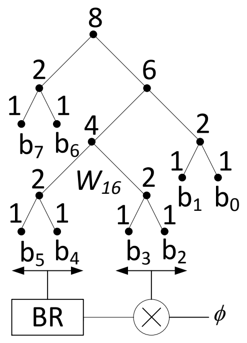


Figure 9. Binary tree and twiddle factor exponent generation.

cases.

V. TWIDDLE FACTOR EXPONENT GENERATION

When designing an FFT architecture, twiddle factor exponent (ϕ) is a key parameter for twiddle factor generator. There are two basic types: on the one hand, run time generator, which generates the rotation at run-time. On the other hand, off-line generator computes the angles and stores in memory called twiddle factor coefficient memory. However both types requires the exponent of twiddle factor even it is also used for reduction of twiddle factor coefficient memory [12]. This normally ends up the importance of exponent of twiddle factor. As discussed earlier, the $n = p + q$ root node, which is decomposed into two child nodes, p and q, then these leaves are further decomposed into child nodes. The decomposition continues until the child node matches the radix. The above observation leads to the following method to determine the twiddle factor exponent (ϕ) based on a binary tree representation.

Each row number is written as binary number of length $n = log_2(N)$, which is equal to number of leaf node. Each leaf node assigns to single bit of this binary number. The bit assigning starts from the left leaf node of binary tree with the most significant bit and moves towards right leaf node as shown in Fig. 9. Now, P and Q bits are required for exponent calculation. These bits are decided by the position of twiddle factor in binary tree. In order to calculate exponent value, the Q binary bits are multiplied with the bit-reversed (BR) of P binary bits as shown in Fig. 8. The BR means that swap the most significant bits (MSB) with least significant bits (LSB).

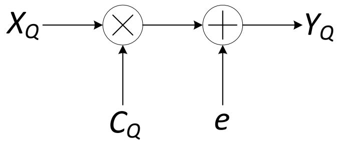


Figure 10. Model for round-off noise.

To illustrate the design, Fig. 9 shows a binary tree of 256-point FFT for calculating the exponent of W_{16} twiddle factor. This presents the P and Q bits of W_{16}, however, these bits will be changed according to the stage location of twiddle factor.

VI. Round-off Noise

Error due to quantization, referred to as round-off errors are introduced when the word length is finite. In fixed point arithmetic, the multiplications must be quantized using rounding or truncation. However, round-off noise at output is used for both rounding and truncation. This is often modeled as a stochastic signal source, producing a noise signal with certain statistical properties as shown in Fig. 10 [12]. In FFTs, it is introduced due to non-trivial multiplications as it is not possible to keep the increased word length after each multiplication. The amount of introduced round-off noise will be used to determine suitable data word lengths in the FFT architecture.

Following [15], a round-off noise model based on uncorrelated noise sources after each non-trivial rotation is adapted as shown in Fig. 10. The round-off noise is then propagated to the outputs, where it is possible to accumulate the number of noise sources propagating to that output as shown in Fig 11. This figure shows the propagation path from error generation to the output. The propagation path passes through the arithmetic operation, which increases the effect of error at output. It is further assumed that the total noise power accumulated over all the outputs is a valid measure. This can be argued, since some outputs may receive a significantly larger contribution than others, although it simplifies the analysis. In addition, the commonly used scaling by two after each butterfly is not taken into consideration. This is, lacking more advanced dynamic data scaling strategies such as [18], a straight forward way to guarantee that overflows will not happen.

The resulting round-off noise measure for a generalized algorithm as in Fig. 11 for an N-point FFT with $N = 2^n$ can be obtained by observing that a round-off noise source in stage i will propagate to 2^{n-i} outputs. This leads to the total number of round-off noise sources at the outputs being

$$\sum_{i=0}^{n-1} 2^{n-i-1} F_i \sigma^2, \tag{4}$$

where F_i is the number of non-trivial multiplications in stage i and σ^2 is the round-off noise variance.

VII. Results

To illustrate the advantage of the identical radix-2^k FFT algorithms, the number of non-trivial multiplication and the previously discussed round-off noise are calculated for radix-2^2, radix-2^3 and radix-2^4.

From the results shown in Table III, we can observe that number of non-trivial multiplication almost same in all cases of algorithm. However, the effect of error varies because it depends upon on the stage where the non-trivial multiplication is located.

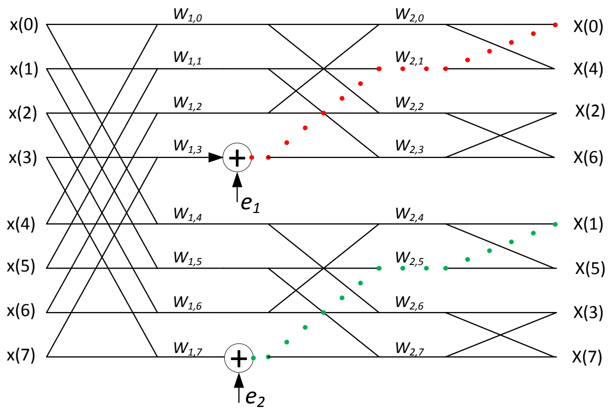


Figure 11. Generalized radix-2 8-point FFT SFG with two error source.

Table III
Number of non-trivial multiplication and round-off noise of radix-2^2 and identical radix-2^2 FFT Algorithms for $N = 256$.

Algorithms	Non-Trivial Multiplication	Round-off noise (σ_c^2)
Radix-2^2	276	15232
Case-I	276	14784
Case-II	272	14064
Case-III	288	12288
Case-IV	276	11760

Regarding the variation in numbers, we can observe that up to 27% reduction gain at output without any additional hardware cost.

Similar behavior can be observed for radix-2^3 and radix-2^4 FFT algorithms in the Tables IV and V respectively. In these cases, savings are up to 8% and 3% respectively. From the above, we can conclude that the overall saving decreases with value of k. This is expected behavior because the non-trivial multiplication increases at the initial stages of SFG with the value of k.

Table IV
Number of non-trivial multiplication and round-off noise of radix-2^3 and identical radix-2^3 FFT algorithms for $N = 4096$.

Algorithms	Non-Trivial Multiplication	Round-off noise (σ_c^2)
Radix-2^3	14408	3278848
Case-I	14408	3253760
Case-II	14216	3243760
Case-III	14464	3237824
Case-IV	14408	3024832

Table V
Number of non-trivial multiplication and round-off noise of radix-2^4 and identical radix-2^4 FFT algorithms for $N = 64K$.

Algorithms	Non-Trivial Multiplication	Round-off noise (σ_c^2)
Radix-2^4	377104	769245184
Case-I	337104	768323584
Case-II	337044	767344384
Case-III	377344	753516544
Case-IV	337104	752598784

VIII. Conclusion

The binary tree approach can be applied to generate FFT algorithms quickly based on Cooley Tukey method. In this paper, a new identical radix-2^k FFT algorithms are proposed. However, the hardware cost is similar to radix-2^k FFT algorithm. We also analyze those FFT algorithms based on the error sources and their impact at the output of FFT. The quantitative results show that error at the output is reduced 27%, 8%, and 3% in identical radix-2^2, radix-2^3, and radix-2^4 FFT algorithms respectively.

Acknowledgment

This work is partially supported by Tekes - the Finnish Funding Agency for Innovation under funding decision $40142/14$ (StreamPro) in the FiDiPro program.

References

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

[2] P. Duhamel and M. Vetterli, "Fast Fourier transforms: A tutorial review and a state of the art," *Signal Processing*, vol. 19, no. 4, pp. 259 – 299, 1990.

[3] S. Bouguezel, M. Ahmad, and M. Swamy, "A new radix$-2/8$ FFT algorithm for length$-q\ \times 2^m$ DFTs," *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 51, no. 9, pp. 1723–1732, Sep. 2004.

[4] L. Jia, Y. Gao, and H. Tenhunen, "Efficient VLSI implementation of radix-8 FFT algorithm," in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 1999*, 1999, pp. 468–471.

[5] J. Takala and K. Punkka, "Butterfly unit supporting radix-4 and radix-2 FFT," in *Proc. International TICSP Workshop on Spectral Methods and Multirate Signal Process.*, vol. 30, 2005, pp. 47–54.

[6] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proc. Int. Parallel Processing Symp.*, 1996, pp. 766–770.

[7] J.-Y. Oh and M.-S. Lim, "New radix-2 to the 4th power pipeline FFT processor," *IEICE Trans. Electron.*, vol. E88-C, no. 8, pp. 1740–1746, Aug. 2005.

[8] A. Cortes, I. Velez, and J. 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.

[9] M. Garrido, J. Grajal, M. A. Sanchez, and O. Gustafsson, "Pipelined radix-2^k feedforward FFT architectures," *IEEE Trans. VLSI Syst.*, vol. 21, no. 1, pp. 23–32, Jan 2013.

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

[11] Y.-N. Chang, "An efficient VLSI architecture for normal I/O order pipeline FFT design," *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 55, no. 12, pp. 1234–1238, Dec. 2008.

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

[13] P. Welch, "A fixed-point fast Fourier transform error analysis," *IEEE Trans. Audio Electroacoust.*, vol. 17, no. 2, pp. 151–157, Jun 1969.

[14] Tran-Thong and B. Liu, "Fixed-point fast Fourier transform error analysis," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 24, no. 6, pp. 563–573, Dec 1976.

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

[16] 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, Apr. 2007.

[17] A. V. Oppenheim and R. W. Schafer, *Dicrete-Time Signal Processing*. Prentice Hall, 1989.

[18] T. Lenart and V. Öwall, "Architectures for dynamic data scaling in 2/4/8K pipeline FFT cores," *IEEE Trans. VLSI Syst.*, vol. 14, no. 11, pp. 1286–1290, Nov. 2006.