New Approaches for the Design of Low-Complexity Radix-Based FFT and FHT Algorithms

Saad Bouguezel

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montreal, Quebec, Canada

September 2004

Your file Votre référence
ISBN: 0-612-96958-4
Our file Notre référence
ISBN: 0-612-96958-4

ABSTRACT

New Approaches for the Design of Low-Complexity Radix-Based FFT and FHT Algorithms

Saad Bouguezel, Ph. D.

Concordia University, 2004

The discrete Fourier transform (DFT) and discrete Hartley transform (DHT) play a crucial role in one- and multi-dimensional digital signal processing applications. Traditionally, the main concern in the design of fast Fourier transform (FFT) and fast Hartley transform (FHT) algorithms has been the reduction of the arithmetic complexity. However, with the recent advances in the digital technology and the present demands of such transforms in low-power high-performance real-time applications, a more comprehensive treatment of the computational and structural complexities must be considered in the design of the algorithms. The objective of this thesis is to design one- and multi-dimensional FFT and FHT algorithms that address the problem of reducing the number of arithmetic operations, data transfers, address generations, and twiddle factor evaluations or accesses to the lookup table, while possessing features such as simplicity, regularity, modularity, easy indexing scheme, and butterfly-style and in-place computations that are highly desirable characteristics for software or hardware implementations of the algorithms. To achieve these objectives, radix-based algorithms are proposed by introducing new decomposition strategies, efficient index mappings, and by an appropriate use of the Kronecker product.

A general decomposition method, which is based on the radix-2 approach, valid for any dimension and applicable to both the DHT and DFT, and which significantly reduces the complexity of the FHT algorithms, is proposed. This method enables us to develop multidimensional FHT and FFT algorithms. A new approach for computing the DFT and DHT using a unified structure is proposed by establishing a close relationship, valid for any dimension, between the radix-2 based FHT and FFT algorithms.

An efficient method, based on the radix-2 approach, for pruning output samples of a 1-D or 2-D DFT is proposed by grouping in its 1-D or 2-D FFT algorithm all the stages that involve unnecessary operations into a single stage and by introducing a new recursive technique for the computations required in the resulting stage.

A technique is presented to improve the performance of the radix-4, radix-8 and radix-16 FFT algorithms in terms of the number of twiddle factor evaluations or accesses to the lookup table without any increase in the computational or structural complexities of the algorithms.

In order to take advantage of the lowest structural complexity provided by the radix-2 approach and reduced computational complexity offered by the radix-4 approach, a technique suitable for combining these two approaches is introduced in order to develop efficient 3-D FFT and FHT algorithms.

A radix-2/8 approach for reducing the complexity in the computation of the 1-D DFT and DHT of lengths $N = q \times 2^m$ is proposed by appropriately mixing the radix-2 and radix-8 index maps. This approach is extended to 2-D and 3-D DFTs. It is shown that the proposed radix-2/8 approach is superior to all the other existing radix-based approaches in providing low-complexity 1-D, 2-D and 3-D FFT, and 1-D FHT algorithms.

To my loving family

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude and heartfelt thanks to my thesis supervisors, Dr. M. Omair Ahmad and Dr. M.N.S. Swamy, for their invaluable guidance and mentorship throughout the span of this research. Without their insight, inspiration, valuable advice and encouragement, it would not have been possible to complete this thesis. I feel honored and privileged for having the opportunity to work under their supervision.

It is my great pleasure to thank my friends and colleagues at the Center for Signal Processing and Communications, Department of Electrical and Computer Engineering, for the many fruitful discussions which we have had during the course of this work.

I owe special thanks to my family members for their encouragement, love, patience and sacrifices. Their support and understanding has contributed a great deal for completing this work.

Finally, I gratefully acknowledge the support from the Natural Sciences and Engineering Research Council of Canada, through research grants awarded to Dr. M. Omair Ahmad and Dr. M.N.S. Swamy.

Contents

1 Introduction **1**

1.1 General 1

1.2 Structural and Computational Complexities 2

1.3 Scope and Organization of the Thesis 4

2 Radix-2 Approach for the Computation of DFT and DHT of Arbitrary Dimensions **7**

2.1 Introduction 7

2.2 Proposed M-D Radix-$(2 \times 2 \times ... \times 2)$ DIF FHT Algorithm 9

2.3 Proposed M-D Radix-$(2 \times 2 \times ... \times 2)$ DIT FHT Algorithm 21

2.4 Computational Complexities of the Proposed M-D Radix-$(2 \times 2 \times ... \times 2)$ FHT Algorithms 27

2.5 Relationship Between the Structures of the M-D Radix-$(2 \times 2 \times ... \times 2)$ Complex-Valued FFT Algorithms and the Proposed M-D Radix-$(2 \times 2 \times ... \times 2)$ FHT Algorithms 34

2.5.1 An M-D Radix-$(2 \times 2 \times ... \times 2)$ DIF Complex-Valued FFT Algorithm 35

2.5.2 An M-D Radix-$(2 \times 2 \times ... \times 2)$ DIT Complex-Valued FFT Algorithm 39

2.6 Summary 42

3 Output-Pruning of DFTs **44**

3.1 Introduction 44

3.2 Output-Pruning of the 1-D DFT 45

3.2.1 Proposed Radix-2 DIT FFT Pruning Algorithm 47

3.2.2 Proposed Radix-2 DIF FFT Pruning Algorithm 53

3.3 Output-Pruning of the 2-D DFT 54

3.3.1 Proposed Radix-(2×2) DIT FFT Pruning Algorithm 55

3.4 Summary 62

4 Higher Radix Approach for the Computation of DFTs **63**

4.1 Introduction 63

4.2 An Improved Radix-4 DIF FFT Algorithm 64

4.3 An Improved Radix-8 DIF FFT Algorithm 66

4.4 An Improved Radix-16 DIF FFT Algorithm 71

4.5 Summary 76

5 Radix-2/4 Approach for the Computation of 3-D DFT and DHT **77**

5.1 Introduction 77

5.2 Radix-2/4 Approach for the Computation of 3-D DFT 78

5.2.1 Proposed Radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ DIF FFT Algorithm 79

5.2.2 Computational Complexity 87

5.2.2.1 Arithmetic Complexity 87

5.2.2.2 Data Transfers 92

5.2.2.3 Twiddle Factors 94

5.3 Radix-2/4 Approach for the Computation of 3-D DHT 95

5.3.1 Proposed Radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ DIF FHT Algorithm 97

5.3.2 Proposed Radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ DIT FHT Algorithm 107
5.3.3 Computational Complexity 114
5.3.3.1 Arithmetic Complexity 115
5.3.3.2 Data Transfers 121
5.3.3.3 Twiddle factors 122
5.4 Summary 124

6 A New radix-2/8 Approach **125**
6.1 Introduction 125
6.2 Radix-2/8 Approach for the Computation of Length-$q \times 2^m$ 1-D DFT . . . 126
6.2.1 Proposed Radix-2/8 DIF FFT Algorithm 127
6.2.2 Computational Complexity 138
6.2.2.1 Arithmetic Complexity 138
6.2.2.2 Data Transfers 144
6.2.2.3 Twiddle Factors 146
6.2.2.4 Example 149
6.3 Radix-2/8 Approach for the Computation of Length-$q \times 2^m$ 1-D DHT . . . 150
6.3.1 Proposed Radix-2/8 DIF FHT Algorithm 151
6.3.2 Computational Complexity 168
6.3.2.1 Arithmetic Complexity 168
6.3.2.2 Data Transfers 173
6.3.2.3 Twiddle Factors 175
6.4 Summary 178

7 Extension of the Radix-2/8 Approach for the Computation of DFTs of Higher Dimensions **179**
7.1 Introduction 179

7.2 Proposed Radix-$(2 \times 2)/(8 \times 8)$ DIF FFT Algorithm 181
7.2.1 Computational Complexity 190
7.2.1.1 Arithmetic Complexity 190
7.2.1.2 Data Transfers 192
7.2.1.3 Twiddle Factors 193
7.3 Proposed Radix-$(2 \times 2 \times 2)/(8 \times 8 \times 8)$ DIF FFT Algorithm 195
7.3.1 Computational Complexity 202
7.3.1.1 Arithmetic Complexity 202
7.3.1.2 Data Transfers 206
7.3.1.3 Twiddle Factors 207
7.4 Summary 209

8 Conclusion **210**
8.1 Concluding Remarks 210
8.2 Scope for Further Investigation 214

References **216**

List of Figures

2.1 General butterfly of the proposed 2-D DIF FHT algorithm, $c_i = \cos(\theta k_i)$, $s_i = \sin(\theta k_i)$, $c_{12} = \cos(\theta(k_1 + k_2))$, $s_{12} = \sin(\theta(k_1 + k_2))$, $\theta = 2\pi/N$. . . . 17

2.2 General butterfly of the proposed 4-D DIF FHT algorithm, $c_i = \cos(\theta k_i)$, $s_i = \sin(\theta k_i)$, $c_{ij} = \cos\left(\theta\left(k_i + k_j\right)\right)$, $s_{ij} = \sin\left(\theta\left(k_i + k_j\right)\right)$, $c_{ijl} = \cos(\theta(k_i + k_j + k_l))$, $s_{ijl} = \sin\left(\theta\left(k_i + k_j + k_l\right)\right)$, $c_{1234} = \cos(\theta(k_1 + k_2 + k_3 + k_4))$, $s_{1234} = \sin(\theta(k_1 + k_2 + k_3 + k_4))$, $\theta = 2\pi/N$. 18

2.3 General butterfly of the proposed M-D DIF FHT algorithm, $c_i = \cos(\theta k_i)$, $s_i = \sin(\theta k_i)$, $c_{ij} = \cos\left(\theta\left(k_i + k_j\right)\right)$, $s_{ij} = \sin\left(\theta\left(k_i + k_j\right)\right)$, $c_{ij\ldots l} = \cos(\theta(k_i + k_j + \ldots + k_l))$, $s_{ij\ldots l} = \sin\left(\theta\left(k_i + k_j + \ldots + k_l\right)\right)$, $c_{ij\ldots lh} = \cos(\theta(k_i + k_j + \ldots + k_l + k_h))$, $s_{ij\ldots lh} = \sin\left(\theta\left(k_i + k_j + \ldots + k_l + k_h\right)\right)$, $\theta = 2\pi/N$. 19

2.4 General butterfly of the proposed 4-D DIT FHT algorithm, $e = -1$, $c_i = \cos(\theta n_i)$, $s_i = \sin(\theta n_i)$, $c_{ij} = \cos\left(\theta\left(n_i + n_j\right)\right)$, $s_{ij} = \sin\left(\theta\left(n_i + n_j\right)\right)$, $c_{ijl} = \cos\left(\theta\left(n_i + n_j + n_l\right)\right)$, $s_{ijl} = \sin\left(\theta\left(n_i + n_j + n_l\right)\right)$, $c_{1234} = \cos(\theta(n_1 + n_2 + n_3 + n_4))$, $s_{1234} = \sin(\theta(n_1 + n_2 + n_3 + n_4))$, $\theta = 2\pi/N$. 25

2.5 General butterfly of the proposed M-D DIT FHT algorithm, $e = -1$, $c_i = \cos(\theta n_i)$, $s_i = \sin(\theta n_i)$, $c_{ij} = \cos\left(\theta\left(n_i + n_j\right)\right)$, $s_{ij} = \sin\left(\theta\left(n_i + n_j\right)\right)$, $c_{ij\ldots l} = \cos\left(\theta\left(n_i + n_j + \ldots + n_l\right)\right)$, $s_{ij\ldots l} = \sin\left(\theta\left(n_i + n_j + \ldots + n_l\right)\right)$, $c_{ij\ldots lh} = \cos(\theta(n_i + n_j + \ldots + n_l + n_h))$, $s_{ij\ldots lh} = \sin\left(\theta\left(n_i + n_j + \ldots + n_l + n_h\right)\right)$, $\theta = 2\pi/N$. . . 26

6.1 Decomposition diagram of a length-$8q$ DFT. 132

6.2 General butterfly of the proposed radix-2/8 FFT algorithm. 135
6.3 First stage of the proposed radix-2/8 FFT algorithm. 137
6.4 Decomposition of a length-$2q$ DHT. 153
6.5 Decomposition of a length-$4q$ DHT. 154
6.6 Decomposition of a length-$8q$ DHT. 160
6.7 General butterfly of the proposed radix-2/8 FHT algorithm, $\theta = 2\pi qk/N$. . . 162
6.8 Decomposition scheme of the proposed radix-2/8 FHT algorithm. 167

List of Tables

2.1 Number of arithmetic operations required for the computation of the 4-D DHT using the 4mult-2add scheme for $s = 0$ 30

2.2 Number of non-trivial operations required for the computation of the 4-D DHT using the 4mult-2add scheme 32

2.3 Number of non-trivial operations required for the computation of the 5-D DHT using the 4mult-2add scheme 32

3.1 Computational complexities of the different algorithms in the computation of a subset of output samples of the 1-D DFT 46

3.2 Computational complexities of the different algorithms in the computation of a subset of output samples of the 2-D DFT 55

5.1 Number of arithmetic operations required for the computation of the 3-D DFT using the 4mult-2add scheme 90

5.2 Number of arithmetic operations required for the computation of the 3-D DFT using the 3mult-3add scheme 90

5.3 Number of non-trivial operations required for the computation of the 3-D DFT using the 4mult-2add scheme 91

5.4 Number of non-trivial operations required for the computation of the 3-D DFT using the 3mult-3add scheme 91

5.5 Comparison of the number of data transfers 93
5.6 Comparison of the number of twiddle factor evaluations or accesses to the lookup table 95
5.7 Number of arithmetic operations required for the computation of the 3-D DHT using the 4mult-2add scheme 118
5.8 Number of arithmetic operations required for the computation of the 3-D DHT using the 3mult-3add scheme 118
5.9 Number of non-trivial operations required for the computation of the 3-D DHT using the 4mult-2add scheme 119
5.10 Number of non-trivial operations required for the computation of the 3-D DHT using the 3mult-3add scheme 119
5.11 Number of non-trivial operations required for the computation of the 3-D DHT using the 4mult-2add scheme 120
5.12 Comparison of the number of data transfers 122
5.13 Comparison of the number of twiddle factor evaluations or accesses to the lookup table 123

6.1 Arithmetic complexities of the $8q$-, $4q$- and $2q$-point DFTs 138
6.2 Number of arithmetic operations required for the computation of the 1-D DFT using the 3mult-3add scheme for $q = 1$ 141
6.3 Number of arithmetic operations required for the computation of the 1-D DFT using the 4mult-2add scheme for $q = 1$ 141
6.4 Number of arithmetic operations required for the computation of the 1-D DFT using the 4mult-2add scheme for $q = 3$ 142
6.5 Number of arithmetic operations required for the computation of the 1-D DFT using the 3mult-3add scheme for $q = 3$ 142

6.6 Number of arithmetic operations required for the computation of the 1-D DFT using the 4mult-2add scheme for $q = 9$ 143
6.7 Number of arithmetic operations required for the computation of the 1-D DFT using the 4mult-2add scheme for $q = 15$ 143
6.8 Number of multiplications and additions required by 3-, 9- and 15-point DFTs 143
6.9 Comparison of the number of data transfers operations for $q = 1$ 146
6.10 Comparison of the number of twiddle factor evaluations or accesses to the lookup table for $q = 1$ 148
6.11 Comparison of the number of twiddle factor evaluations or accesses to the lookup table for $q = 3$ 148
6.12 Comparison of the arithmetic complexities and twiddle factor requirements for the proposed 2^n- and 3×2^m-point FFT algorithms 150
6.13 Number of arithmetic operations required for the computation of the 1-D DHT for $q = 1$ 170
6.14 Number of arithmetic operations required for the computation of the 1-D DHT for $q = 3$ 171
6.15 Number of arithmetic operations required for the computation of the 1-D DHT for $q = 9$ 171
6.16 Number of arithmetic operations required for the computation of the 1-D DHT for $q = 15$ 172
6.17 Number of multiplications and additions required by 3-, 9- and 15-point DHTs 172
6.18 Comparison of the number of data transfer operations for $q = 1$ 175
6.19 Comparison of the number of twiddle factor evaluations or accesses to the lookup table for $q = 1$ 177

6.20 Comparison of the number of twiddle factor evaluations or accesses to the lookup table for $q = 3$ 177

7.1 Number of arithmetic operations required for the computation of the 2-D DFT using the 3mult-3add scheme 191

7.2 Number of arithmetic operations required for the computation of the 2-D DFT using the 4mult-2add scheme 192

7.3 Comparison of the number of data transfers, and twiddle factor evaluations or accesses to the lookup table 194

7.4 Number of arithmetic operations required for the computation of the 3-D DFT using the 3mult-3add scheme 205

7.5 Number of arithmetic operations required for the computation of the 3-D DFT using the 4mult-2add scheme 205

7.6 Comparison of the number of data transfers, and twiddle factor evaluations or accesses to the lookup table 208

List of Abbreviations

3mult-3add	Three-multiplication-three-addition scheme
4mult-2add	Four-multiplication-two-addition scheme
A.	Number of additions
Adds.	Number of additions
DFT	Discrete Fourier transform
DHT	Discrete Hartley transform
DIF	Decimation-in-frequency decomposition
DIT	Decimation-in-time decomposition
FFT	Fast Fourier transform
FHT	Fast Hartley transform
M.	Number of multiplications
M-D	Multi-dimensional
Mults.	Number of multiplications
RAM	Random access memory
ROM	Read only memory
VLSI	Very large scale integration

List of Symbols

A_q	Number of additions required in the computation of a length-q DHT
$\widetilde{A}_q$	Number of additions required in the computation of a length-q DFT
$\mathbf{H}_L$	Hadamard matrix of order L
$\mathbf{I}_L$	Identity matrix of order L
k, k_i	Indices or variables for time (space) domain
$L, L \times L$	Number of samples in the desired spectrum band
M	Number of dimensions in a sequence or transform
M_q	Number of multiplications required in the computation of a length-q DHT
$\widetilde{M}_q$	Number of multiplications required in the computation of a length-q DFT
M_q^s	Number of multiplications required in the computation of a scaled length-q DHT
$\widetilde{M}_q^s$	Number of multiplications required in the computation of a scaled length-q DFT
N	Number of samples in each dimension
n, n_i	Indices or variables for frequency domain
$\mathbf{O}$	Null matrix
$\otimes$	Kronecker product
q	Arbitrary or odd integer
$X(.)$	Real-valued output sequence of a DHT
$x(.)$	Real-valued input sequence of a DHT
$\widetilde{X}(.)$	Complex-valued output sequence of a DFT
$\widetilde{x}(.)$	Complex-valued input sequence of a DFT

Chapter 1

Introduction

1.1 General

The discrete Fourier transform (DFT) plays a central role in many applications such as frequency-domain analysis and design of signals and systems in communications, image processing, multi-dimensional digital signal processing, and in many other areas of science and engineering [1], [2]. Since the discovery of the radix-2 fast Fourier transform (FFT) algorithm by Cooley and Tukey in 1965 [3], intensive research efforts have been made by several investigators resulting in a number of fast algorithms for the computation of the one-dimensional (1-D) DFT [4]-[21] and the multi-dimensional (M-D) DFT [22]-[35]. All these FFT algorithms have been derived specifically for computing complex-valued DFTs.

There are many practical applications that involve only real data. The direct use of the complex-valued FFT algorithms in such applications is time consuming and memory intensive. To overcome this problem, specific FFT algorithms have been derived to compute real-valued DFTs [36]-[41]. An alternative solution is through the use of the discrete Hartley transform (DHT) introduced by Bracewell in 1983 [42], which has now become an important tool in image and signal processing. The main difference between the DHT and

the DFT is that the former is a real-to-real transform, whereas the latter is, in general, a complex-to-complex transform. In other words, the kernel of the DHT is real, whereas that of the DFT is complex. An important property of the DHT is that the forward and inverse transforms have the same form except for the scaling factor. This property is also preserved by the M-D DHT introduced by Buneman [43] and can be used as an alternative to the M-D DFT for applications that involve real data. Almost all the approaches that have been used for deriving complex-valued FFT algorithms have been applied to derive fast Hartley transform (FHT) algorithms. Since the introduction of the FHT algorithm by Bracewell in 1984 [44], considerable research has been conducted resulting in a number of algorithms for computing the 1-D DHT [45]-[57] and the M-D DHT [58]-[68]. It has been shown in [45] that the structure of a real-valued FFT algorithm is complicated compared to that of an FHT algorithm and, in general, the forward and inverse versions of the former are completely different, which is not the case with the latter. Due to these drawbacks of the real-valued FFT and due to the fact that the DHT is an efficient alternative to the DFT for real-data applications, development of real-valued FFT algorithms is not attractive.

Until the nineties, the DFT and DHT have found their practical applications mainly in those that involve 1-D and 2-D signals, thus motivating researchers to concentrate on 1-D and 2-D transform algorithms. However, in recent years, there is also a growing interest in applications that involve 3-D and 4-D DFTs [69]-[76].

1.2 Structural and Computational Complexities

As mentioned in Section 1.1, a large number of algorithms, based on a number of different approaches, have been proposed in the literature for the computation of the DFT and DHT; however, only a few of them are popular and widely used. One of the most interesting approaches is that of Cooley-Tukey [3], which has been used to develop radix-2 [3], [10],

[21], [38], [41], [45], radix-4 [12], [45], radix-8 [4], [36], mixed radix [5], and vector radix [22]-[24], [26], [32], [33], [58], [65], [66] transform algorithms. Although the arithmetic complexities of these algorithms are not optimum, the algorithms have been found to be more attractive for software and hardware implementations than all the other ones. This is mainly due to their features of simplicity, regularity, modularity, easy indexing scheme, and butterfly-style and in-place computations, which comprise the *structural complexity* of the algorithms. The split-radix (or radix-2/4) approach introduced by Duhamel and Hollmann in [11] has provided attractive algorithms for one [11], [17], [45], [56], [57] and higher [29], [30], [39], [64] dimension transforms. These algorithms possess a good compromise between the arithmetic and structural complexities. All the other approaches that have been proposed so far are less popular than the radix-based ones, since the resulting algorithms [6], [7], [9], [19], [25], [34], [35], [47], [48], [50], [51], [59], [61], [63], [67], [68] have structures that are rather complicated. Consequently, the structural complexity is an important factor that should be considered in the development of an efficient algorithm.

From the mid-sixties to the mid-eighties, the main concern of the researchers in this areas was to optimize the number of multiplications [77]-[79], since the most costly operation in hardware or software implementation of an algorithm at that time was multiplication. This led to algorithms requiring minimum number of multiplications [6], [7], [9], [25], and any further reduction in the number of multiplications became very difficult. Due to the rapid advances in the VLSI design and technology, the cost of a multiplication reduced significantly and became comparable to that of an addition in many computer architectures [80]-[84]. As a consequence, it became important to consider the optimization of the number of arithmetic operations (multiplications + additions) rather than optimizing the number of multiplications alone [11], [15], [19], [34], [38], [45], [47], [68], [85], [86]. Further reduction in the number of arithmetic operations has become difficult, especially for the case of 1-D transform algorithms. Due to the recent demands for the applications of the

DFT and DHT in wireless communication, radar and portable computation, the reduction of other operations such as data transfers, address generations, and twiddle factor evaluations or accesses to the lookup table has become significantly important. Such reductions are also crucial in the design of low-power digital signal processors [87]-[97]. Therefore, in addition to the structural complexity, the factors that should be considered to determine the overall complexity of an algorithm must include the numbers of the 1) multiplications, 2) additions, 3) data transfers, 4) address generations, and 5) twiddle factor evaluations or accesses to the lookup table. We shall, henceforth, refer to the complexity of these five operations as *computational complexity*. To this date, no approach exists in the literature that addresses the problem of simultaneously reducing the complexity of the above five operations.

Recently, some attention has been given to reducing the number of twiddle factor accesses to the lookup table and address generations in the computation of the DFT [88], [89], [91]-[96]. This reduction has been achieved at the implementation level by suitable modifications of the existing FFT algorithms. However, in general, these modifications introduce additional complexity, and the improvements achieved are not that significant. In order to achieve a significant reduction in the computational complexity, it is essential to focus on this aspect at the algorithm design level itself.

1.3 Scope and Organization of the Thesis

The objective of this thesis is to devise techniques for the design of one- and multi-dimensional FFT and FHT algorithms that address the problem of reducing their computational and structural complexities. The emphasis is to introduce certain attributes in the algorithms at the design level itself so that the feature of reduced complexity is naturally achieved when the algorithms are implemented. The thesis is organized as follows.

Chapter 2 addresses the problem of reducing the complexity in the computation of the DHT and DFT of arbitrary dimensions and introduces a general decomposition method for these transforms by exploring the most popular approach, namely, the radix-2 approach, and by using appropriate index mappings and the Kronecker product. This chapter also introduces a new approach for a unified computation of the DHT and DFT.

There are many applications where a significant part of the output samples of a DFT are not needed, for example, where only a narrow band of the spectrum is of interest. Chapter 3 considers this case and introduces an efficient method for pruning the output samples of the 1-D and 2-D FFT algorithms that are based on the radix-2 approach.

Even though the radix-2 approach leads to algorithms having the lowest structural complexities, the computational complexities of these algorithms are still higher than those provided by a higher radix approach. Chapter 4 presents an efficient technique for improving the radix-4, radix-8, and radix-16 approaches for the computation of the DFT. This is achieved by introducing new indices for some of the sub-sequences resulting from the conventional decompositions in the radix-4, radix-8, and radix-16 FFT algorithms.

Algorithms requiring the lowest number of arithmetic operations have been obtained in the 1-D case [11], [17], [45], [56], [57] by a simultaneous use of the radix-2 and radix-4 approaches, which has lead to the so-called radix-2/4 approach. This approach has been exploited in developing effective algorithms for the computation of 2-D transforms [29], [30], [64]. However, this approach has not yet been exploited in the case of three or higher dimensions. This is mainly due to the fact that the simultaneous use of the radix-2 and radix-4 approaches in the case of higher dimensions is neither obvious nor straightforward. This problem, for the 3-D case, is efficiently solved in Chapter 5 by introducing a new decomposition strategy and appropriate mathematical tools.

Chapter 6 proposes a new approach that provides a substantial reduction in the number of data transfers, address generations, and twiddle factor evaluations or accesses to the lookup table in the computation of the DFT and DHT, while maintaining the lowest number of arithmetic operations. This approach is based on an appropriate simultaneous use of the radix-2 and radix-8 approaches, and provides a natural reduction in the number of these operations.

Chapter 7 proposes efficient 2-D and 3-D FFT algorithms by exploiting the approach introduced in Chapter 6.

Finally, Chapter 8 concludes the thesis by highlighting the contributions made in this investigation and suggesting some possible future research work.

Chapter 2

Radix-2 Approach for the Computation of DFT and DHT of Arbitrary Dimensions

2.1 Introduction

The growing interest in applications involving higher-dimension signals has spurred a great deal of interest in the challenging problem of the computation of M-D DFT and DHT. Since the M-D DFT is separable and the M-D DHT is not, the use of the existing approaches in developing algorithms for fast computation of the former is much easier than for the case of the latter. Two approaches that have been used for computing the M-D DFT have also been applied to compute the M-D DHT. One is the conventional row-column decomposition approach, which requires matrix transposition operations and high computational complexity. Another approach, based on polynomial transforms, provides FHT algorithms [67], [68] with reduced arithmetic complexities at the expense of very complicated structures. Other algorithms [59], [61], [63] that have been specifically proposed for the M-D DHT have also

complicated structures and, in general, are not suitable for software and hardware implementations. To cope with the problems of prohibitively large computational complexity and the large volume of data for the computation of M-D DHT, there is a need to develop FHT algorithms that are more efficient and valid for any dimension, possessing highly desirable properties such as regularity, simplicity and in-place computation similar to the ones of the existing M-D radix-$(2 \times 2 \times ... \times 2)$ complex-valued FFT algorithms [24], [26], [32].

The most regular and simple algorithm among the ones reported for the fast computation of the one-dimensional (1-D) DHT is the radix-2 FHT algorithm [45]. It has been extended for the 2-D case using decimation-in-time (DIT) [58] and, recently, for the 3-D case using DIT [65] as well as decimation-in-frequency (DIF) [66]. These FHT algorithms have been specifically derived for the DHT of a particular dimension and their extension to other higher dimensions is neither obvious nor straightforward. However, it is desirable to generalize the radix-2 approach for computing the DHT of any dimension.

In this chapter, the problem of reducing the complexity in the computation of the DHT of an arbitrary dimension is addressed, and efficient M-D radix-$(2 \times 2 \times ... \times 2)$ DIF and DIT FHT algorithms are proposed [98], [99] by exploring the radix-2 approach and making use of an appropriate index mapping and the Kronecker product. The resulting butterflies of the proposed algorithms are characterized by simple closed-form expressions that allow easy implementations of these algorithms for any dimension.

Since the 1-D radix-2 complex-valued DIT and DIF FFT algorithms and their extensions to the M-D case are very popular and frequently used in practice for the computation of DFTs, it is highly desirable to investigate all the existing hardware and software implementations of these FFT algorithms in the computation of the M-D DHTs. By reformulating the existing M-D radix-$(2 \times 2 \times ... \times 2)$ complex-valued DIF and DIT FFT algorithms using an approach similar to that used for deriving the proposed M-D radix-$(2 \times 2 \times ... \times 2)$ DIF and DIT FHT algorithms, a close relationship between the former and the latter algorithms,

valid for any dimension, is established in this chapter [98], [99]. This type of relationship is of great importance for software and hardware implementations of the algorithms, since such a relationship along with the fact that the DHT is an alternative to the DFT for real data would make it possible for a single kind of software or hardware module to be used, possibly with some minor modifications, for the computation of the forward and inverse M-D DFTs for real- or complex-valued data and M-D DHTs. Thus, the same module (with a little or no modification) can be used to cover all domains of applications that involve DFT or DHT.

In Sections 2.2 and 2.3, we propose M-D radix-$(2 \times 2 \times ... \times 2)$ DIF and DIT FHT algorithms and show how to map the mathematical expressions, derived for these algorithms, into actual structures for implementations. In Section 2.4, we study the performance of the proposed M-D FHT algorithms by analyzing their computational and structural complexities and comparing them with those of the existing M-D FHT algorithms. In Section 2.5, we show that there exists a close relationship between the M-D radix-$(2 \times 2 \times ... \times 2)$ complex-valued FFT algorithms and the proposed M-D radix-$(2 \times 2 \times ... \times 2)$ FHT algorithms.

2.2 Proposed M-D Radix-$(2 \times 2 \times ... \times 2)$ DIF FHT Algorithm

Let $x(k_1, ..., k_i, ..., k_M)$, where $1 \leq i \leq M$, be a real data sequence of size $(N_1 \times N_2 \times \ ... \times N_M)$; its M-D DHT $X(n_1, ..., n_i, ..., n_M)$ is also a real sequence of size $(N_1 \times N_2 \times \ ... \times N_M)$ defined by

$$X(n_1, ..., n_i, ..., n_M) = \sum_{k_1=0}^{N_1-1} ... \sum_{k_i=0}^{N_i-1} ... \sum_{k_M=0}^{N_M-1} x(k_1, ..., k_i, ..., k_M) \operatorname{cas}\left(2\pi \sum_{i=1}^{M} \frac{n_i k_i}{N_i}\right),$$
$$0 \leq n_i \leq N_i - 1, \ 1 \leq i \leq M \quad (2.1)$$

where $\text{cas}(\theta) = \cos(\theta) + \sin(\theta)$. Without loss of generality, we assume that $N_i = N$, for $1 \leq i \leq M$, where N is an integral power of two. Then, (2.1) becomes a size-$(N \times N \times ... \times N)$ M-D DHT given by

$$X(n_1, ..., n_i, ..., n_M) = \sum_{k_1=0}^{N-1} ... \sum_{k_i=0}^{N-1} ... \sum_{k_M=0}^{N-1} x(k_1, ..., k_i, ..., k_M) \text{cas}\left(\frac{2\pi}{N} \sum_{i=1}^{M} n_i k_i\right), \quad 0 \leq n_i \leq N-1, \ 1 \leq i \leq M \tag{2.2}$$

Let us start by deriving in matrix form, using the Kronecker product, the M-D DHT for the interesting case of $N = 2$. Then, the size-$(2 \times 2 \times ... \times 2)$ M-D DHT given by (2.2) can be expressed in a matrix form as

$$\mathbf{X} = \mathbf{H}_{2^M} \mathbf{x} \tag{2.3}$$

where $\mathbf{x}$ and $\mathbf{X}$ are, respectively, the input and output vectors. Their mth components are related to the input and output sequences, respectively, by

$$x(m) = x(m_1, ..., m_i, ..., m_M) \tag{2.4}$$

and

$$X(m) = X(m_1, ..., m_i, ..., m_M) \tag{2.5}$$

where m varies from 0 to $2^M - 1$ and m_i, for $1 \leq i \leq M$, are the binary bits of m, that is,

$$m = m_1 2^{M-1} + m_2 2^{M-2} + ... + m_i 2^{M-i} + ... + m_M 2 + m_M \tag{2.6}$$

The transform operator in (2.3) is a Hadamard matrix of order 2^M, which can be generated from the second-order Hadamard matrix given by

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

using the Kronecker product [32], [100]

$$\mathbf{H}_{2^M} = \mathbf{H}_2 \otimes \mathbf{H}_2 \otimes \ldots \otimes \mathbf{H}_2 \tag{2.7}$$

The matrix $\mathbf{H}_{2^M}$ in (2.7) can be efficiently factored as

$$\mathbf{H}_{2^M} = \prod_{i=1}^{M} (\mathbf{I}_{2^{i-1}} \otimes \mathbf{H}_2 \otimes \mathbf{I}_{2^{M-i}}) \tag{2.8}$$

where $\mathbf{I}_L$ is an identity matrix of order L. Since the right side of (2.8) can be considered as a product of M matrices each requiring 2^M additions, the matrix $\mathbf{H}_{2^M}$ introduces $M2^M$ additions in the computation of the size-$(2 \times 2 \times \ldots \times 2)$ M-D DHT given by (2.3).

Let us now consider the general case of (2.2). Since $\text{cas}(\theta) = \text{Re}[(1+j)\exp(-j\theta)]$, $j = \sqrt{-1}$, (2.2) can be written as

$$X(n_1, \ldots, n_i, \ldots, n_M) = \sum_{k_1=0}^{N-1} \cdots \sum_{k_i=0}^{N-1} \cdots \sum_{k_M=0}^{N-1} x(k_1, \ldots, k_i, \ldots, k_M) \text{Re}\left[(1+j)\exp\left(-j\frac{2\pi}{N}\sum_{i=1}^{M} n_i k_i\right)\right], \quad 0 \le n_i \le N-1 \tag{2.9}$$

In order to exploit the matrix representation derived above for the special case of $N = 2$, we

change the variables k_i and n_i in (2.9), respectively, by

$$k_i + q_i \frac{N}{2},\ 0 \le k_i \le \frac{N}{2} - 1,\ q_i = 0, 1 \tag{2.10}$$

and

$$2n_i + p_i,\ 0 \le n_i \le \frac{N}{2} - 1,\ p_i = 0, 1 \tag{2.11}$$

Then, (2.9) becomes

$$\begin{aligned} & X(2n_1 + p_1, ..., 2n_i + p_i, ..., 2n_M + p_M) \\ & = \sum_{k_1=0}^{N/2-1} \cdots \sum_{k_i=0}^{N/2-1} \cdots \sum_{k_M=0}^{N/2-1} \left(\sum_{q_1=0}^{1} \cdots \sum_{q_i=0}^{1} \cdots \sum_{q_M=0}^{1} \right. \\ & \quad x\left(k_1 + q_1 \frac{N}{2}, ..., k_i + q_i \frac{N}{2}, ..., k_M + q_M \frac{N}{2}\right) \\ & \quad \mathrm{Re}\left[(-1)^{\sum_{i=1}^{M} p_i q_i} \exp\left(-j \frac{2\pi}{N} \sum_{i=1}^{M} p_i k_i \right) \right. \\ & \quad \left. \left. (1 + j) \exp\left(-j \frac{2\pi}{N/2} \sum_{i=1}^{M} n_i k_i \right) \right] \right) \end{aligned} \tag{2.12}$$

The quantity Re[•] in (2.12) can be expressed as

$$\begin{aligned} & (-1)^{\sum_{i=1}^{M} p_i q_i} \left(\cos\left(\frac{2\pi}{N} \sum_{i=1}^{M} p_i k_i \right) \mathrm{cas}\left(\frac{2\pi}{N/2} \sum_{i=1}^{M} n_i k_i \right) \right. \\ & \quad \left. + \sin\left(\frac{2\pi}{N} \sum_{i=1}^{M} p_i k_i \right) \mathrm{cas}\left(\frac{-2\pi}{N/2} \sum_{i=1}^{M} n_i k_i \right) \right) \end{aligned} \tag{2.13}$$

Substituting (2.13) in (2.12) and using the fact that

$$\sum_{k_1=0}^{N/2-1}\cdots\sum_{k_i=0}^{N/2-1}\cdots\sum_{k_M=0}^{N/2-1} x\left(k_1+q_1\frac{N}{2},\ldots,k_i+q_i\frac{N}{2},\ldots,k_M+q_M\frac{N}{2}\right)$$
$$\sin\left(\frac{2\pi}{N}\sum_{i=1}^{M}p_ik_i\right)\operatorname{cas}\left(\frac{-2\pi}{N/2}\sum_{i=1}^{M}n_ik_i\right)$$
$$=\sum_{k_1=0}^{N/2-1}\cdots\sum_{k_i=0}^{N/2-1}\cdots\sum_{k_M=0}^{N/2-1} x\left(\frac{N}{2}-k_1+q_1\frac{N}{2},\ldots,\frac{N}{2}-k_i+q_i\frac{N}{2},\ldots,\frac{N}{2}-k_M+q_M\frac{N}{2}\right)$$
$$\sin\left(\frac{2\pi}{N}\sum_{i=1}^{M}p_i\left(\frac{N}{2}-k_i\right)\right)\operatorname{cas}\left(\frac{2\pi}{N/2}\sum_{i=1}^{M}n_ik_i\right) \tag{2.14}$$

we see that (2.12) corresponds to a set of 2^M size-$\left(\frac{N}{2}\times\frac{N}{2}\times\ldots\times\frac{N}{2}\right)$ M-D DHTs given by

$$X(2n_1+p_1,\ldots,2n_i+p_i,\ldots,2n_M+p_M)=\sum_{k_1=0}^{N/2-1}\cdots\sum_{k_i=0}^{N/2-1}\cdots\sum_{k_M=0}^{N/2-1}$$
$$F_{k_1,\ldots,k_i,\ldots,k_M}(p_1,\ldots,p_i,\ldots,p_M)\operatorname{cas}\left(\frac{2\pi}{N/2}\sum_{i=1}^{M}n_ik_i\right),$$
$$0\le n_i\le\frac{N}{2}-1,\ p_i=0,1 \tag{2.15}$$

The 2^M input sequences of (2.15) are given by

$$F_{k_1,\ldots,k_i,\ldots,k_M}(p_1,\ldots,p_i,\ldots,p_M)=\cos\left(\frac{2\pi}{N}\sum_{i=1}^{M}p_ik_i\right)\left[\sum_{q_1=0}^{1}\cdots\sum_{q_i=0}^{1}\cdots\sum_{q_M=0}^{1}\right.$$
$$\left.x\left(k_1+q_1\frac{N}{2},\ldots,k_i+q_i\frac{N}{2},\ldots,k_M+q_M\frac{N}{2}\right)(-1)^{\sum_{i=1}^{M}p_iq_i}\right]$$
$$+\sin\left(\frac{2\pi}{N}\sum_{i=1}^{M}p_i\left(\frac{N}{2}-k_i\right)\right)\left[\sum_{q_1=0}^{1}\cdots\sum_{q_i=0}^{1}\cdots\sum_{q_M=0}^{1}\right.$$
$$\left.x\left(\frac{N}{2}-k_1+q_1\frac{N}{2},\ldots,\frac{N}{2}-k_i+q_i\frac{N}{2},\ldots,\frac{N}{2}-k_M+q_M\frac{N}{2}\right)(-1)^{\sum_{i=1}^{M}p_iq_i}\right],$$
$$p_i=0,1 \tag{2.16}$$

It is clear that (2.16) can be considered, for a given combination of k_i, $1\le i\le M$, as a sum

of two size-$(2 \times 2 \times ... \times 2)$ M-D DHTs that are multiplied by twiddle factors. Therefore, using (2.3), (2.16) can be expressed in a matrix form as

$$\begin{aligned} \mathbf{F}_{k_1,...,k_i,...,k_M} &= \mathbf{D}^c_{k_1,...,k_i,...,k_M}\mathbf{H}_{2^M}\mathbf{x}_{k_1,...,k_i,...,k_M} \\ &\quad +\mathbf{D}^s_{\frac{N}{2}-k_1,...,\frac{N}{2}-k_i,...,\frac{N}{2}-k_M}\mathbf{H}_{2^M}\mathbf{x}_{\frac{N}{2}-k_1,...,\frac{N}{2}-k_i,...,\frac{N}{2}-k_M} \end{aligned} \tag{2.17}$$

where the mth component of the input vector $\mathbf{x}_{k_1,...,k_i,...,k_M}$ is given by

$$x_{k_1,...,k_i,...,k_M}(m) = x\left(k_1 + m_1\frac{N}{2}, ..., k_i + m_i\frac{N}{2}, ..., k_M + m_M\frac{N}{2}\right), \tag{2.18}$$

the (m, m)th component of the diagonal matrix $\mathbf{D}^c_{k_1,...,k_i,...,k_M}$ by

$$d^c_{k_1,...,k_i,...,k_M}(m, m) = \cos\left(\frac{2\pi}{N}\sum_{i=1}^{M} m_i k_i\right), \tag{2.19}$$

and the (m, m)th component of the diagonal matrix $\mathbf{D}^s_{k_1,...,k_i,...,k_M}$ by

$$d^s_{k_1,...,k_i,...,k_M}(m, m) = \sin\left(\frac{2\pi}{N}\sum_{i=1}^{M} m_i k_i\right), \tag{2.20}$$

The components of the vector $\mathbf{F}_{k_1,...,k_i,...,k_M}$ in (2.17) are related to the input sequence of (2.15) by

$$F_{k_1,...,k_i,...,k_M}(m) = F_{k_1,...,k_i,...,k_M}(m_1, ..., m_i, ..., m_M). \tag{2.21}$$

Since the in-place computation property is highly desirable to reduce the storage, especially for the multidimensional case, we use the same input samples of (2.17) in the computation of the vector $\mathbf{F}_{\frac{N}{2}-k_1,...,\frac{N}{2}-k_i,...,\frac{N}{2}-k_M}$. This can be achieved by replacing k_i by $\frac{N}{2} - k_i$ in (2.17). Then, we obtain

$$\mathbf{F}_{\frac{N}{2}-k_1,\ldots,\frac{N}{2}-k_i,\ldots,\frac{N}{2}-k_M} = \mathbf{D}^{s}_{k_1,\ldots,k_i,\ldots,k_M}\mathbf{H}_{2^M}\mathbf{x}_{k_1,\ldots,k_i,\ldots,k_M} + \mathbf{D}^{c}_{\frac{N}{2}-k_1,\ldots,\frac{N}{2}-k_i,\ldots,\frac{N}{2}-k_M}\mathbf{H}_{2^M}\mathbf{x}_{\frac{N}{2}-k_1,\ldots,\frac{N}{2}-k_i,\ldots,\frac{N}{2}-k_M} \quad (2.22)$$

It is seen that the application of the Hadamard matrix on the input vector is required in the computation of the vectors given by (2.17) and (2.22). Some of these operations are redundant and can be avoided by combining the two equations. Therefore, we may obtain a closed-form expression for the general butterfly of the proposed M-D radix-$(2 \times 2 \times \ldots \times 2)$ DIF FHT algorithm as

$$\begin{bmatrix} \mathbf{F}_{k_1,\ldots,k_i,\ldots,k_M} \\ \mathbf{F}_{\frac{N}{2}-k_1,\ldots,\frac{N}{2}-k_i,\ldots,\frac{N}{2}-k_M} \end{bmatrix} = \mathbf{D}^{t}_{k_1,\ldots,k_i,\ldots,k_M} \begin{bmatrix} \mathbf{H}_{2^M} & \mathbf{O} \\ \mathbf{O} & \mathbf{H}_{2^M} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{k_1,\ldots,k_i,\ldots,k_M} \\ \mathbf{x}_{\frac{N}{2}-k_1,\ldots,\frac{N}{2}-k_i,\ldots,\frac{N}{2}-k_M} \end{bmatrix} \quad (2.23)$$

where $\mathbf{O}$ is a null matrix and the twiddle factor matrix $\mathbf{D}^{t}_{k_1,\ldots,k_i,\ldots,k_M}$ is the transpose of the matrix given by

$$\mathbf{D}_{k_1,\ldots,k_i,\ldots,k_M} = \begin{bmatrix} \mathbf{D}^{c}_{k_1,\ldots,k_i,\ldots,k_M} & \mathbf{D}^{s}_{k_1,\ldots,k_i,\ldots,k_M} \\ \mathbf{D}^{s}_{\frac{N}{2}-k_1,\ldots,\frac{N}{2}-k_i,\ldots,\frac{N}{2}-k_M} & \mathbf{D}^{c}_{\frac{N}{2}-k_1,\ldots,\frac{N}{2}-k_i,\ldots,\frac{N}{2}-k_M} \end{bmatrix} \quad (2.24)$$

In order to avoid the redundant computations in (2.23), one of the indices k_i $(1 \leq i \leq M)$ varies from 0 to $\left(\frac{N}{4}-1\right)$, whereas the others vary from 0 to $\left(\frac{N}{2}-1\right)$. It is interesting to note some useful properties of the elements of the twiddle factor matrix given by (2.24). It can be seen from (2.19) that

$$d^{c}_{\frac{N}{2}-k_1,\ldots,\frac{N}{2}-k_i,\ldots,\frac{N}{2}-k_M}(m,m) = (-1)^{\sum_{i=1}^{M} m_i} d^{c}_{k_1,\ldots,k_i,\ldots,k_M}(m,m) \quad (2.25)$$

and

$$d^c_{k_1,\ldots,k_i,\ldots,k_M}(0,0) = d^c_{\frac{N}{2}-k_1,\ldots,\frac{N}{2}-k_i,\ldots,\frac{N}{2}-k_M}(0,0) = 1 \tag{2.26}$$

Similarly, from (2.20), we have

$$d^s_{\frac{N}{2}-k_1,\ldots,\frac{N}{2}-k_i,\ldots,\frac{N}{2}-k_M}(m,m) = -(-1)^{\sum_{i=1}^{M} m_i} d^s_{k_1,\ldots,k_i,\ldots,k_M}(m,m) \tag{2.27}$$

and

$$d^s_{k_1,\ldots,k_i,\ldots,k_M}(0,0) = d^s_{\frac{N}{2}-k_1,\ldots,\frac{N}{2}-k_i,\ldots,\frac{N}{2}-k_M}(0,0) = 0 \tag{2.28}$$

It is seen from (2.25) and (2.27) that the transposition operation of the matrix given by (2.24) to obtain the twiddle factor matrix $\mathbf{D}^t_{k_1,\ldots,k_i,\ldots,k_M}$ corresponds to just changing the sign of the off-diagonal sub-matrices in (2.24). In addition, it is clear from (2.25)-(2.28) that, for a given combination of k_i's, the computation of the operation given by (2.23) requires only $2 \times \left(2^M - 1\right)$ twiddle factors, including trivial factors, to be evaluated or loaded from a lookup table. Using (2.25) and (2.27), the twiddle factor matrix for the butterfly, given by (2.23), of the proposed M-D radix-$(2 \times 2 \times \ldots \times 2)$ DIF FHT algorithm can be expressed as

$$\mathbf{D}^t_{k_1,\ldots,k_i,\ldots,k_M} = \mathbf{T}^t_{k_1,\ldots,k_i,\ldots,k_M} \begin{bmatrix} \mathbf{I}_{2^M} & \mathbf{O} \\ \mathbf{O} & \mathbf{E}_{2^M} \end{bmatrix} \tag{2.29}$$

where

$$\mathbf{T}_{k_1,\ldots,k_i,\ldots,k_M} = \begin{bmatrix} \mathbf{D}^c_{k_1,\ldots,k_i,\ldots,k_M} & \mathbf{D}^s_{k_1,\ldots,k_i,\ldots,k_M} \\ -\mathbf{D}^s_{k_1,\ldots,k_i,\ldots,k_M} & \mathbf{D}^c_{k_1,\ldots,k_i,\ldots,k_M} \end{bmatrix} \tag{2.30}$$

and $\mathbf{E}_{2^M}$ is a diagonal matrix whose elements are given by

$$e(m,m) = (-1)^{\sum_{i=1}^{M} m_i} \tag{2.31}$$

Let us first consider the particular cases of $M = 1$, 2, 3 and 4. By setting $M = 1$ and

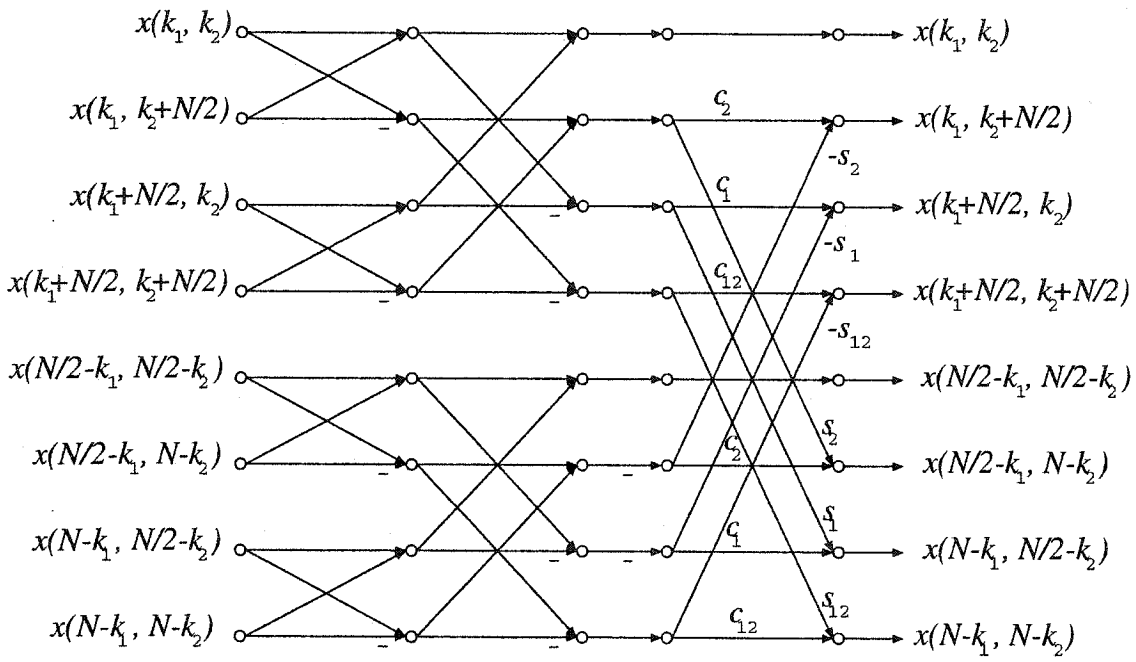

Figure 2.1: General butterfly of the proposed 2-D DIF FHT algorithm, $c_i = \cos(\theta k_i)$, $s_i = \sin(\theta k_i)$, $c_{12} = \cos(\theta(k_1 + k_2))$, $s_{12} = \sin(\theta(k_1 + k_2))$, $\theta = 2\pi/N$.

3 in (2.23), the proposed M-D radix-$(2 \times 2 \times ... \times 2)$ DIF FHT algorithm reduces to the existing 1-D [45] and 3-D [66] DIF FHT algorithms, respectively. For $M = 2$ and 4, the butterflies of the proposed 2-D and 4-D DIF FHT algorithms are depicted in Figs. 2.1 and 2.2, respectively. In these figures, we have adopted the same notation for the input and output samples, since the proposed algorithms have the property of in-place computation. They are obtained by a direct mapping of the expression given by (2.23) for $M = 2$ and 4, respectively. It is noted that at present no radix-$(2 \times 2 \times ... \times 2)$ DIF FHT algorithm exists in the literature for $M \geq 4$.

This direct mapping can be easily generalized; the butterfly for the proposed DIF FHT algorithm for any dimension M is illustrated in Fig. 2.3. In this figure, the gain denoted by * depends only on the dimension M and its value is either +1 or -1. The value of the gain can be obtained from the matrix $\mathbf{E}_{2^M}$ given by (2.31). Due to the closed-form expression of the butterfly given by (2.23), the proposed DIF FHT algorithm can be mapped very

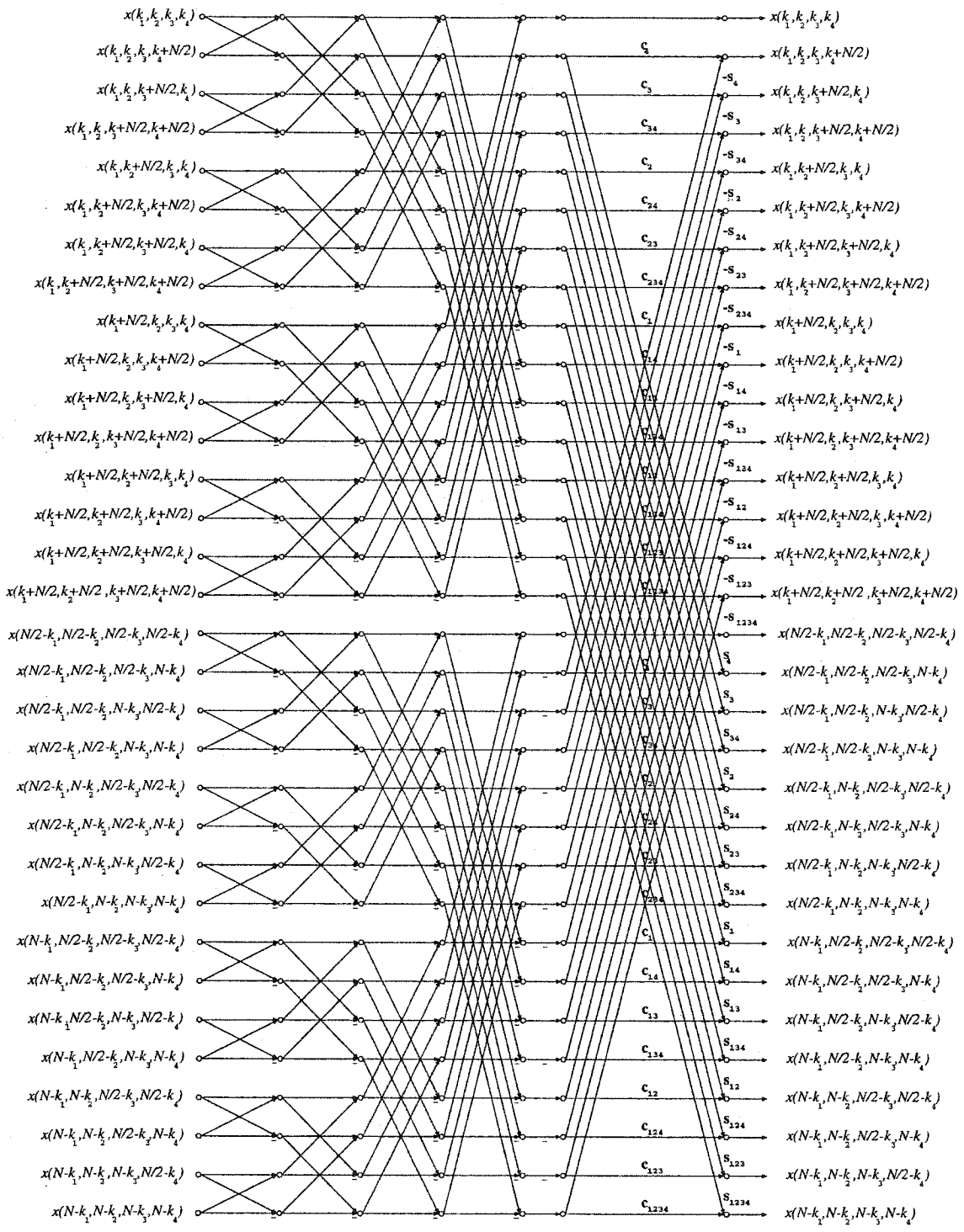

Figure 2.2: General butterfly of the proposed 4-D DIF FHT algorithm, $c_i = \cos(\theta k_i)$, $s_i = \sin(\theta k_i)$, $c_{ij} = \cos\left(\theta\left(k_i + k_j\right)\right)$, $s_{ij} = \sin\left(\theta\left(k_i + k_j\right)\right)$, $c_{ijl} = \cos(\theta(k_i + k_j + k_l))$, $s_{ijl} = \sin\left(\theta\left(k_i + k_j + k_l\right)\right)$, $c_{1234} = \cos\left(\theta\left(k_1 + k_2 + k_3 + k_4\right)\right)$, $s_{1234} = \sin\left(\theta\left(k_1 + k_2 + k_3 + k_4\right)\right)$, $\theta = 2\pi/N$.

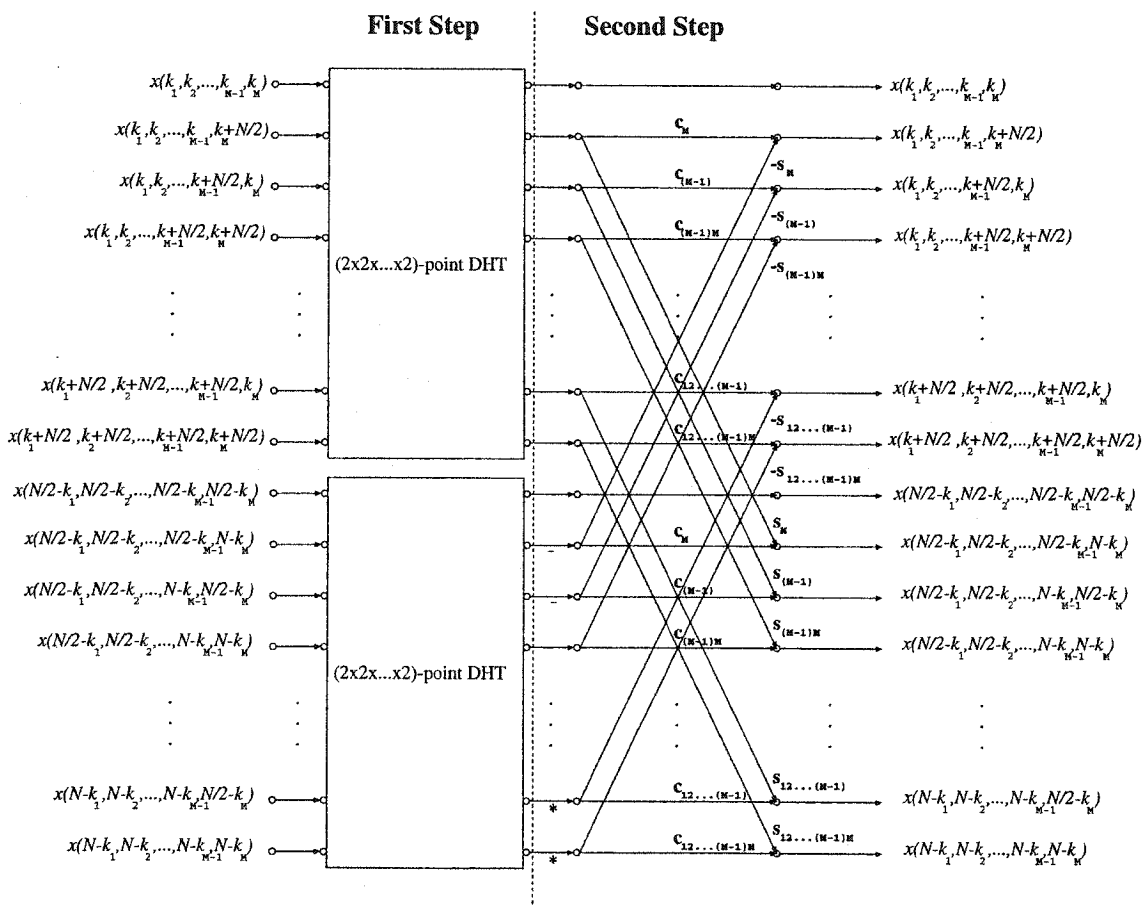

Figure 2.3: General butterfly of the proposed M-D DIF FHT algorithm, $c_i = \cos(\theta k_i)$, $s_i = \sin(\theta k_i)$, $c_{ij} = \cos\left(\theta\left(k_i + k_j\right)\right)$, $s_{ij} = \sin\left(\theta\left(k_i + k_j\right)\right)$, $c_{ij\ldots l} = \cos(\theta(k_i + k_j + \ldots + k_l))$, $s_{ij\ldots l} = \sin\left(\theta\left(k_i + k_j + \ldots + k_l\right)\right)$, $c_{ij\ldots lh} = \cos(\theta(k_i + k_j + \ldots + k_l + k_h))$, $s_{ij\ldots lh} = \sin\left(\theta\left(k_i + k_j + \ldots + k_l + k_h\right)\right)$, $\theta = 2\pi/N$.

easily into structures for implementation for any dimension. It is clear that the operation given by (2.23) can be performed in two steps. In the first step, a pair of 2^M samples that are regularly selected from the input sequence according to (2.18) are processed by applying the operator $\mathbf{H}_{2^M}$. As seen earlier, $\mathbf{H}_{2^M}$, a Hadamard matrix of order 2^M, is a size-$(2 \times 2 \times ... \times 2)$ M-D DHT operator. The mapping of this matrix (which introduces only additions and subtractions) into a structure can be easily obtained according to the decomposition given by (2.8). Addition and subtraction operations of two samples are mapped using the well-known radix-2 butterfly. The second step consists of multiplying the results obtained from the first step by the matrix given by (2.29). In this step, a change of signs of some of the results obtained from the first step is performed according to the diagonal matrix $\mathbf{E}_{2^M}$. Then, multiplication by the transpose of the matrix given by (2.30) is carried out. Each row of this matrix contains only two non-zero elements, except for the first and $(2^M + 1)$th rows. These non-zero elements are twiddle factors and are obtained from (2.19) and (2.20). Hence, the basic operation required in this step consists of multiplying two samples, one by cosine and the other by sine, and adding (or subtracting) the results of these multiplications. It is clear from (2.26) and (2.28) that the first and $(2^M + 1)$th rows of the matrix given by (2.30) contain only one non-zero element that is unity. Hence, the additions (or subtractions) and multiplications of the two samples corresponding to these two rows are not needed.

Finally, the proposed M-D radix-$(2 \times 2 \times ... \times 2)$ DIF FHT algorithm corresponds to decomposing a size-$(N \times N \times ... \times N)$ M-D DHT into 2^M size-$\left(\frac{N}{2} \times \frac{N}{2} \times ... \times \frac{N}{2}\right)$ M-D DHTs given by (2.15) in the first stage of the decomposition. This is achieved by repeating $\frac{N^M}{2 \times 2^M}$ times the butterfly given by (2.23) and illustrated in Fig. 2.3. It is clear from (2.15) and (2.23) that the input sequence of the resulting ith size-$\left(\frac{N}{2} \times \frac{N}{2} \times ... \times \frac{N}{2}\right)$ M-D DHT is obtained from the ith and $(2^M + i)$th rows of the butterfly. This decomposition scheme can be repeated successively for each of the new resulting DHTs until size-$(2 \times 2 \times ... \times 2)$ M-D

DHTs result in the last stage. The entire algorithm requires $\log_2 N$ stages. It can be seen that the operations such as multiplications, additions, and twiddle factor evaluations or accesses to a lookup table introduced by the twiddle factor matrix of the butterfly of the proposed algorithm in the last two stages are all trivial.

2.3 Proposed M-D Radix-$(2 \times 2 \times \ldots \times 2)$ DIT FHT Algorithm

To derive a M-D radix-$(2 \times 2 \times \ldots \times 2)$ FHT algorithm using the DIT decomposition, we change the variables k_i and n_i, $1 \leq i \leq M$, in (2.9), respectively, by

$$2k_i + q_i,\ 0 \leq k_i \leq \frac{N}{2} - 1,\ q_i = 0, 1 \tag{2.32}$$

and

$$n_i + p_i \frac{N}{2},\ 0 \leq n_i \leq \frac{N}{2} - 1,\ p_i = 0, 1 \tag{2.33}$$

Then, (2.9) can be written as

$$\begin{aligned} X\left(n_1 + p_1\frac{N}{2}, \ldots, n_i + p_i\frac{N}{2}, \ldots, n_M + p_M\frac{N}{2}\right) \\ = \sum_{q_1=0}^{1} \cdots \sum_{q_i=0}^{1} \cdots \sum_{q_M=0}^{1} \left(\sum_{k_1=0}^{N/2-1} \cdots \sum_{k_i=0}^{N/2-1} \cdots \sum_{k_M=0}^{N/2-1} \right. \\ x(2k_1 + q_1, \ldots, 2k_i + q_i, \ldots, 2k_M + q_M) \\ \left. \mathrm{Re}\left[\exp\left(-j\frac{2\pi}{N}\sum_{i=1}^{M} n_i q_i\right)(1+j)\right. \right. \\ \left. \left. \exp\left(-j\frac{2\pi}{N/2}\sum_{i=1}^{M} n_i k_i\right)\right]\right) (-1)^{\sum_{i=1}^{M} p_i q_i} \end{aligned} \tag{2.34}$$

Carrying out the operation of taking the real part in the above equation and using an appropriately modified form of (2.14), (2.34) can be expressed as

$$X\left(n_1+p_1\frac{N}{2},\ldots,n_i+p_i\frac{N}{2},\ldots,n_M+p_M\frac{N}{2}\right)$$
$$=\sum_{q_1=0}^{1}\cdots\sum_{q_i=0}^{1}\cdots\sum_{q_M=0}^{1}G_{n_1,\ldots,n_i,\ldots,n_M}(q_1,\ldots,q_i,\ldots,q_M)$$
$$\cos\left(\frac{2\pi}{N}\sum_{i=1}^{M}n_iq_i\right)(-1)^{\sum_{i=1}^{M}p_iq_i}$$
$$+\sum_{q_1=0}^{1}\cdots\sum_{q_i=0}^{1}\cdots\sum_{q_M=0}^{1}G_{\frac{N}{2}-n_1,\ldots,\frac{N}{2}-n_i,\ldots,\frac{N}{2}-n_M}(q_1,\ldots,q_i,\ldots,q_M)$$
$$\sin\left(\frac{2\pi}{N}\sum_{i=1}^{M}n_iq_i\right)(-1)^{\sum_{i=1}^{M}p_iq_i},$$
$$p_i=0,1 \qquad (2.35)$$

where

$$G_{n_1,\ldots,n_i,\ldots,n_M}(q_1,\ldots,q_i,\ldots,q_M)=\sum_{k_1=0}^{N/2-1}\cdots\sum_{k_i=0}^{N/2-1}\cdots\sum_{k_M=0}^{N/2-1}$$
$$x(2k_1+q_1,\ldots,2k_i+q_i,\ldots,2k_M+q_M)\,\mathrm{cas}\left(\frac{2\pi}{N/2}\sum_{i=1}^{M}n_ik_i\right),$$
$$0\le n_i\le\frac{N}{2}-1,\ q_i=0,1 \qquad (2.36)$$

For all combinations of q_i's, (2.36) represents a set of 2^M size-$\left(\frac{N}{2}\times\frac{N}{2}\times\ldots\times\frac{N}{2}\right)$ M-D DHTs. It is clear that for a given combination of n_i's, (2.35) represents a sum of two size-$(2\times2\times\ldots\times2)$ M-D DHTs whose input sequences are obtained from (2.36) after the multiplications by the twiddle factors ($\cos(\bullet)$ and $\sin(\bullet)$). Therefore, exploiting the matrix representation given by (2.3), (2.35) can be expressed in a matrix form as

$$\begin{aligned}\mathbf{X}_{n_1,\ldots,n_i,\ldots,n_M} &= \mathbf{H}_{2^M}\mathbf{D}^{c}_{n_1,\ldots,n_i,\ldots,n_M}\mathbf{G}_{n_1,\ldots,n_i,\ldots,n_M} \\ &\quad +\mathbf{H}_{2^M}\mathbf{D}^{s}_{n_1,\ldots,n_i,\ldots,n_M}\mathbf{G}_{\frac{N}{2}-n_1,\ldots,\frac{N}{2}-n_i,\ldots,\frac{N}{2}-n_M}\end{aligned} \tag{2.37}$$

where the components of the vector $\mathbf{G}_{n_1,\ldots,n_i,\ldots,n_M}$ are obtained from the output samples of the DHTs given by (2.36) as

$$G_{n_1,\ldots,n_i,\ldots,n_M}(m) = G_{n_1,\ldots,n_i,\ldots,n_M}(m_1,\ldots,m_i,\ldots,m_M) \tag{2.38}$$

The desired output samples given by (2.35) are related to the component of the output vector $\mathbf{X}_{n_1,\ldots,n_i,\ldots,n_M}$ of (2.37) by

$$X_{n_1,\ldots,n_i,\ldots,n_M}(m) = X\left(n_1+m_1\frac{N}{2},\ldots,n_i+m_i\frac{N}{2},\ldots,n_M+m_M\frac{N}{2}\right) \tag{2.39}$$

In order to derive an M-D radix-$(2\times 2\times\ldots\times 2)$ DIT FHT algorithm, which possesses the property of in-place computation, we replace n_i by $\frac{N}{2}-n_i$ in (2.37), thus yielding

$$\begin{aligned}\mathbf{X}_{\frac{N}{2}-n_1,\ldots,\frac{N}{2}-n_i,\ldots,\frac{N}{2}-n_M} &= \mathbf{H}_{2^M}\mathbf{D}^{s}_{\frac{N}{2}-n_1,\ldots,\frac{N}{2}-n_i,\ldots,\frac{N}{2}-n_M}\mathbf{G}_{n_1,\ldots,n_i,\ldots,n_M} \\ &\quad +\mathbf{H}_{2^M}\mathbf{D}^{c}_{\frac{N}{2}-n_1,\ldots,\frac{N}{2}-n_i,\ldots,\frac{N}{2}-n_M}\mathbf{G}_{\frac{N}{2}-n_1,\ldots,\frac{N}{2}-n_i,\ldots,\frac{N}{2}-n_M}\end{aligned} \tag{2.40}$$

By combining (2.37) and (2.40), we may obtain a closed-form expression for the general butterfly of the proposed M-D radix-$(2\times 2\times\ldots\times 2)$ DIT FHT algorithm as

$$\begin{bmatrix} \mathbf{X}_{n_1,\ldots,n_i,\ldots,n_M} \\ \mathbf{X}_{\frac{N}{2}-n_1,\ldots,\frac{N}{2}-n_i,\ldots,\frac{N}{2}-n_M} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{2^M} & \mathbf{O} \\ \mathbf{O} & \mathbf{H}_{2^M} \end{bmatrix} \mathbf{D}_{n_1,\ldots,n_i,\ldots,n_M} \begin{bmatrix} \mathbf{G}_{n_1,\ldots,n_i,\ldots,n_M} \\ \mathbf{G}_{\frac{N}{2}-n_1,\ldots,\frac{N}{2}-n_i,\ldots,\frac{N}{2}-n_M} \end{bmatrix} \tag{2.41}$$

Just as for the proposed M-D radix-$(2\times 2\times\ldots\times 2)$ DIF FHT algorithm derived in Section

2.2, in (2.41), one of the indices n_i $(1 \leq i \leq M)$ must vary from 0 to $\left(\frac{N}{4}-1\right)$ and the others from 0 to $\left(\frac{N}{2}-1\right)$, in order to avoid the redundant computations. The twiddle factor matrix for the butterfly, given by (2.41), of the proposed M-D radix-$(2 \times 2 \times ... \times 2)$ DIT FHT algorithm can also be expressed as

$$\mathbf{D}_{n_1,...,n_i,...,n_M} = \begin{bmatrix} \mathbf{I}_{2^M} & \mathbf{O} \\ \mathbf{O} & \mathbf{E}_{2^M} \end{bmatrix} \mathbf{T}_{n_1,...,n_i,...,n_M} \tag{2.42}$$

Let us first consider the particular cases of $M = 1, 2, 3$ and 4. By setting $M = 1, 2$ and 3 in (2.41), the proposed M-D DIF FHT algorithm reduces, respectively, to the existing 1-D [45], 2-D [58] and 3-D [65] DIT FHT algorithms. For $M = 4$, the butterfly of the proposed 4-D DIT FHT algorithm is depicted in Fig. 2.4. This is obtained by a direct mapping of the expression given by (2.41) for $M = 4$. The direct mapping can be easily generalized as was done in the case of the DIF algorithm; the butterfly for the proposed DIT FHT algorithm for any dimension M is illustrated in Fig. 2.5. It is to be noted that at present no radix-$(2 \times 2 \times ... \times 2)$ DIT FHT algorithm exits in the literature for $M \geq 4$.

Finally, in the proposed M-D radix-$(2 \times 2 \times ... \times 2)$ DIT FHT algorithm, the computation of the size-$(N \times N \times ... \times N)$ M-D DHT given by (2.2) consists of computing the 2^M size-$\left(\frac{N}{2} \times \frac{N}{2} \times ... \times \frac{N}{2}\right)$ M-D DHTs given by (2.36) whose output sequences are used in (2.41) for computing the desired output samples. The desired M-D DHT is obtained by repeating $\frac{N^M}{2 \times 2^M}$ times the butterfly given by (2.41) and illustrated in Fig. 2.5. This computation process is applied recursively for the computation of each of the required DHTs until only size-$(2 \times 2 \times ... \times 2)$ M-D DHTs need to be computed. The whole algorithm requires $\log_2 N$ stages. It can be seen that the operations such as multiplications, additions, and twiddle factor evaluations or accesses to a lookup table introduced by the twiddle factor matrix of the butterfly of the proposed algorithm in the first two stages of the computation are all trivial.

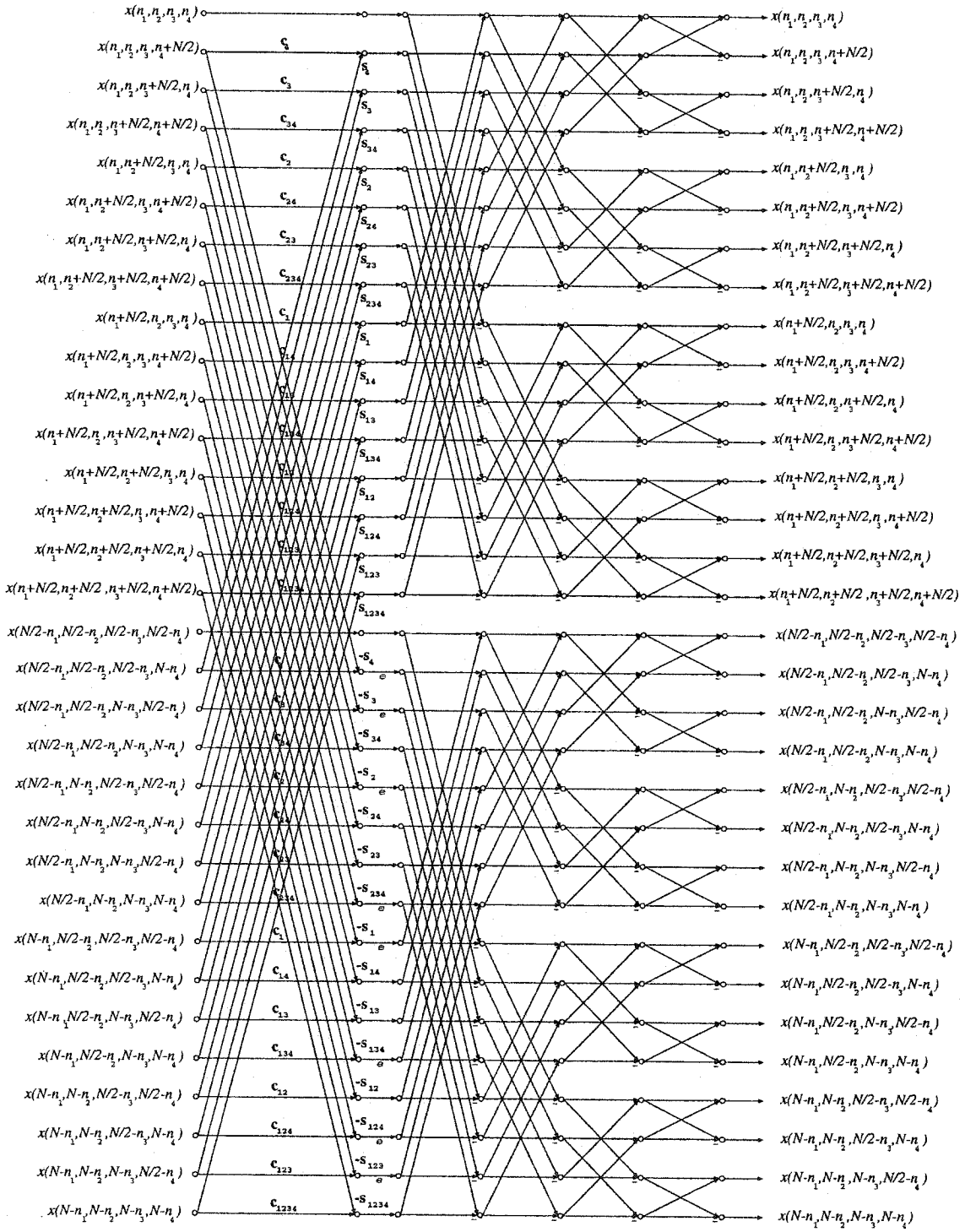

Figure 2.4: General butterfly of the proposed 4-D DIT FHT algorithm, $e = -1$, $c_i = \cos(\theta n_i)$, $s_i = \sin(\theta n_i)$, $c_{ij} = \cos(\theta(n_i + n_j))$, $s_{ij} = \sin(\theta(n_i + n_j))$, $c_{ijl} = \cos(\theta(n_i + n_j + n_l))$, $s_{ijl} = \sin(\theta(n_i + n_j + n_l))$, $c_{1234} = \cos(\theta(n_1 + n_2 + n_3 + n_4))$, $s_{1234} = \sin(\theta(n_1 + n_2 + n_3 + n_4))$, $\theta = 2\pi/N$.

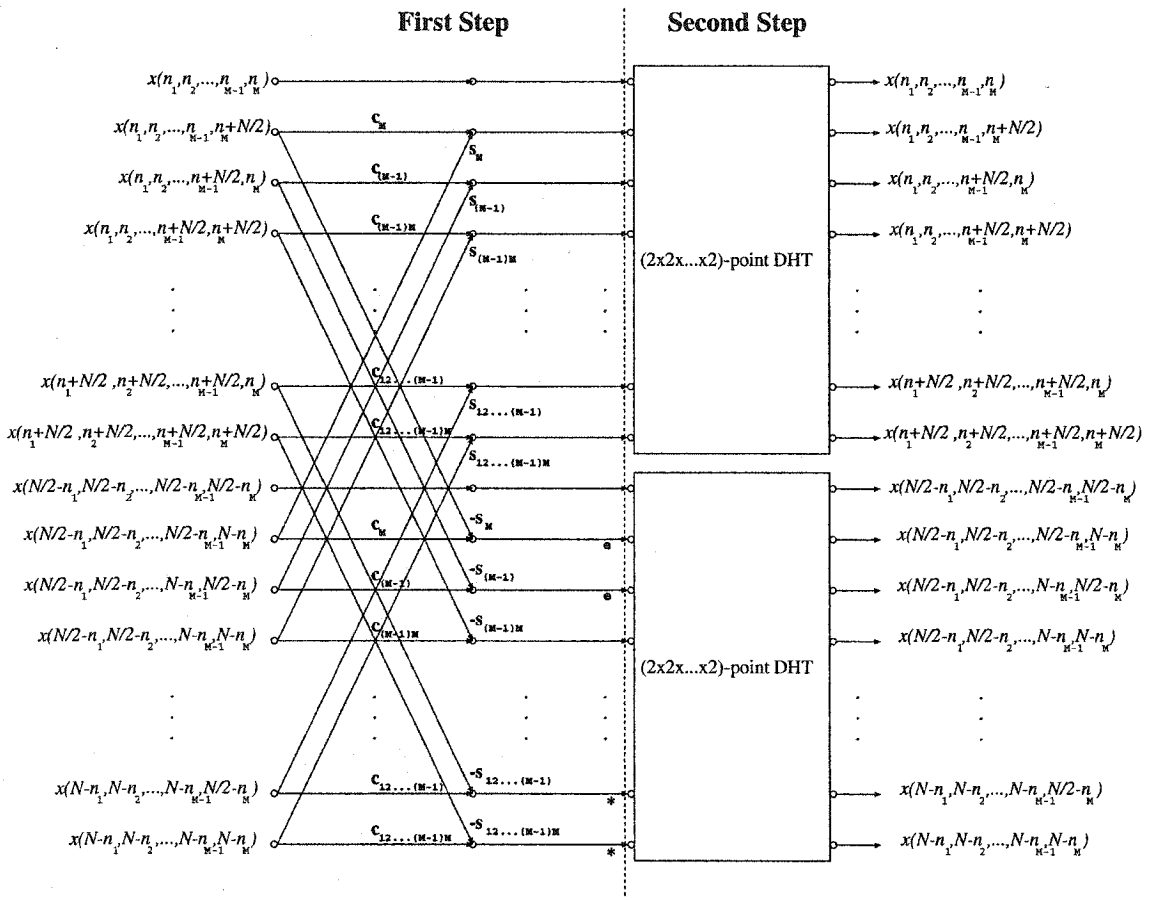

Figure 2.5: General butterfly of the proposed M-D DIT FHT algorithm, $e = -1$, $c_i = \cos(\theta n_i)$, $s_i = \sin(\theta n_i)$, $c_{ij} = \cos\left(\theta\left(n_i + n_j\right)\right)$, $s_{ij} = \sin\left(\theta\left(n_i + n_j\right)\right)$, $c_{ij\ldots l} = \cos\left(\theta\left(n_i + n_j + \ldots + n_l\right)\right)$, $s_{ij\ldots l} = \sin\left(\theta\left(n_i + n_j + \ldots + n_l\right)\right)$, $c_{ij\ldots lh} = \cos(\theta(n_i + n_j + \ldots + n_l + n_h))$, $s_{ij\ldots lh} = \sin\left(\theta\left(n_i + n_j + \ldots + n_l + n_h\right)\right)$, $\theta = 2\pi/N$.

2.4 Computational Complexities of the Proposed M-D Radix-$(2 \times 2 \times ... \times 2)$ FHT Algorithms

In this section, we study the performance of the proposed M-D FHT algorithms by analyzing their computational and structural complexities and comparing them with those of the existing M-D FHT algorithms. Since the proposed DIF and DIT algorithms are based on the same decomposition philosophy, their structural and computational complexities are exactly the same. Therefore, the analysis of the computational complexity of only one would be sufficient. Let us consider the computational complexity of the proposed DIF algorithm. It is clear from (2.26) and (2.28) that the twiddle factor matrix given by (2.24) has for its first and $(2^M + 1)$th rows, all the elements as zero except for one element which is unity. Therefore, in the computation of the right side of (2.23), the twiddle factor matrix introduces $4(2^M - 1)$ multiplications and $2(2^M - 1)$ additions. It is also observed that in the first step of this computation, 2×2^M samples of the input sequence $x(k_1, ..., k_i, ..., k_M)$ are operated upon by the second-order Hadamard matrix $\mathbf{H}_2$. This step requires $2 \times M2^M$ additions. Therefore, the butterfly based on (2.23), which computes 2×2^M samples, requires $4(2^M - 1)$ multiplications and $[2(2^M - 1) + 2 \times M2^M]$ additions. The first stage of the algorithm consists of decomposing the size-$(N \times N \times ... \times N)$ M-D DHT given by (2.2) into 2^M size-$(\frac{N}{2} \times \frac{N}{2} \times ... \times \frac{N}{2})$ M-D DHTs given by (2.15). This is achieved by repeating $\frac{N^M}{2 \times 2^M}$ times the butterfly given by (2.23). This decomposition process is repeated for each of the new resulting DHTs until the problem is reduced to size-$(2 \times 2 \times ... \times 2)$ M-D DHTs. The desired output is obtained after performing $\log_2 N$ such stages. It can be seen from the twiddle factor matrix given by (2.24) that some savings can be achieved in the number of operations for special combinations of k_i, $1 \leq i \leq M$. Specifically, the operations introduced by the twiddle factor matrix in the last two stages are trivial for any combination of k_i. Consequently, the number of multiplications and additions required by the proposed M-D DIF

FHT algorithm are given by

$$\mathrm{M}_{\mathrm{r2}}(M,N) = a\frac{2^M-1}{2^{M-1}}N^M\log_2\frac{N}{2^s} \tag{2.43}$$

$$\mathrm{A}_{\mathrm{r2}}(M,N) = \left(M+b\frac{2^M-1}{2^M}\right)N^M\log_2\frac{N}{2^s}+sMN^M \tag{2.44}$$

where $s=0$ if no special butterfly is used in the implementation, $s=1$ if we can avoid the trivial operations only at the last stage, and $s=2$ if we can avoid the trivial operations at the last two stages. In (2.43) and (2.44), $a=b=1$, if the twiddle factor matrix is implemented using the four-multiplication-two-addition (4mult-2add) scheme, whereas $a=3/4$ and $b=3/2$, if it is implemented using the three-multiplication-three-addition (3mult-3add) scheme.

The arithmetic complexities of the proposed algorithms can be further dramatically reduced by avoiding all trivial operations. Let us consider the 4mult-2add scheme. Then, by removing the trivial operations in the first or last stage of the proposed DIF or DIT algorithm, respectively, and applying the decomposition scheme derived in Section 2.2 or 2.3, it is seen that the expressions for the numbers of non-trivial multiplications and additions required by the proposed algorithms are given by

$$\begin{aligned}&\widetilde{\mathrm{M}}_{\mathrm{r2}}(M,N)\\ &\quad=\begin{cases}\frac{2^M-1}{2^{M-1}}N^M-3\left(2^M-1\right)2^{2M-1}\left(2^{M-1}\right)^{\log_2 N-3}+2^M\widetilde{\mathrm{M}}_{\mathrm{r2}}(M,\frac{N}{2}), & N\geq 8\\ 0, & N<8\end{cases}\end{aligned} \tag{2.45}$$

$$\widetilde{\mathrm{A}}_{\mathrm{r2}}(M,N) = \begin{cases} \left(M+\frac{2^M-1}{2^M}\right)N^M - \left(2^M-1\right)2^{2M-1}\left(2^{M-1}\right)^{\log_2 N-3} + 2^M\widetilde{\mathrm{A}}_{\mathrm{r2}}(M,\frac{N}{2}), N \geq 8 \\ \\ MN^M\log_2 N,\ N<8 \end{cases} \tag{2.46}$$

In the row-column approach, besides computing MN^{M-1} 1-D DHTs, MN^M extra additions are required in the computation of the M-D DHT. Therefore, if the simplest and the most regular existing 1-D radix-2 FHT algorithm is used to compute the 1-D DHTs, then the numbers of multiplications and additions required by the row-column approach are given by

$$\mathrm{M}_{\mathrm{rc}}(M,N) = aMN^M\log_2\frac{N}{2^s} \tag{2.47}$$

$$\mathrm{A}_{\mathrm{rc}}(M,N) = M\left(\frac{b+2}{2}\right)N^M\log_2\frac{N}{2^s} + (s+1)MN^M \tag{2.48}$$

By removing all trivial operations and considering the 4mult-2add scheme, the numbers of non-trivial multiplications and additions required by the row-column approach are given by

$$\widetilde{\mathrm{M}}_{\mathrm{rc}}(M,N) = MN^{M-1}\widetilde{\mathrm{M}}_{\mathrm{r2}}(1,N) \tag{2.49}$$

$$\widetilde{\mathrm{A}}_{\mathrm{rc}}(M,N) = MN^{M-1}\widetilde{\mathrm{A}}_{\mathrm{r2}}(1,N) + MN^M \tag{2.50}$$

Note that the bit-shift operations (multiplications by 1/2) required by the row-column approach in computing the M-D DHT are not included in (2.47) and (2.49). Table 2.1 presents the arithmetic complexities required by the row-column approach and proposed algorithms in the computation of the 4-D DHT. The results in this table are obtained by letting $M=4$ and $s=0$ in (2.43), (2.44), (2.47) and (2.48). It can be verified using (2.43)-(2.44)-(2.47)-

Table 2.1: Number of arithmetic operations required for the computation of the 4-D DHT using the 4mult-2add scheme for $s = 0$

	Row-Column method			Proposed Radix-$(2 \times 2 \times 2 \times 2)$ FHTs		
Transform Size $N \times N \times N \times N$	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point
$2^3 \times 2^3 \times 2^3 \times 2^3$	12	22	34	5.62	14.81	20.43
$2^4 \times 2^4 \times 2^4 \times 2^4$	16	28	44	7.50	19.75	27.25
$2^5 \times 2^5 \times 2^5 \times 2^5$	20	34	54	9.37	24.68	34.06
$2^6 \times 2^6 \times 2^6 \times 2^6$	24	40	64	11.25	29.62	40.87
$2^7 \times 2^7 \times 2^7 \times 2^7$	28	46	74	13.12	34.56	47.68
$2^8 \times 2^8 \times 2^8 \times 2^8$	32	52	84	15.00	39.5	54.50
$2^9 \times 2^9 \times 2^9 \times 2^9$	36	58	94	16.87	44.43	61.31
$2^{10} \times 2^{10} \times 2^{10} \times 2^{10}$	40	64	104	18.75	49.37	68.12
$2^{11} \times 2^{11} \times 2^{11} \times 2^{11}$	44	70	114	20.62	54.31	74.93
$2^{12} \times 2^{12} \times 2^{12} \times 2^{12}$	48	76	124	22.50	59.25	81.75

(2.48) or (2.45)-(2.46)-(2.49)-(2.50) that significant savings in terms of the arithmetic operations are achieved by the proposed algorithms for any set of values of M and N. In addition, these savings increase with the dimension M. Moreover, the matrix transposition and bit-shift operations required in the row-column method are not required in the proposed algorithms.

The row-column scheme has been exploited in [63] to derive an approach different from the traditional row-column method for the M-D DHT. This approach is based on a successive use of 1-D complex-valued FFT algorithms and special retrograde indexing manipulations. In addition, it requires matrix transposition operations and has a computational complexity comparable to that of the traditional row-column method for the M-D DHT. The only advantage of this approach over the traditional row-column method is the possibility of exploiting the existing hardware and software modules of the 1-D complex-valued FFT algorithm in the computation of the M-D DHT. This, however, is not of much importance compared to advantages offered by the proposed algorithms, since there exists a

close relationship between the structures of the M-D radix-$(2 \times 2 \times ... \times 2)$ complex-valued FFT algorithms and the proposed M-D radix-$(2 \times 2 \times ... \times 2)$ FHT algorithms, as shown in Section 2.5.

An approach for computing 2-D and 3-D DHTs has been proposed in [61], which requires a simultaneous use of 1-D FFT and FHT algorithms and a special interstage scheme for addition operations. Further, the number of arithmetic operations (additions + multiplications) needed is greater than those required by the 2-D radix-(2×2) and 3-D radix-$(2 \times 2 \times 2)$ FHT algorithms. Hence, the extension of the approach in [61] to higher dimensions is not attractive.

In [59], an approach based on Fermat number transforms (FNTs) has been proposed for the computation of 2-D and 3-D DHTs. It requires the implementation of 1-D, 2-D, and 3-D forward FNTs, 2-D and 3-D inverse FNTs, and a very large number of bit-shift operations. In addition, the number of arithmetic operations needed is too large compared to those required by the 2-D radix-(2×2) and 3-D radix-$(2 \times 2 \times 2)$ FHT algorithms. All these drawbacks would become more pronounced if the approach in [59] is extended to higher dimensions.

The only approach left for comparison is the one recently reported in [68] for the computation of the M-D DHT. It requires $\frac{1}{2}N^M \log_2 N - \frac{2^{M+1}-1}{2^{M+1}-2}N^M + \frac{2^{M+1}-1}{2^{M+1}-2} + \frac{1}{2}\left(2^M - 1\right)$ multiplications, $\left(M + \frac{1}{2}\right)N^M \log_2 N - \frac{2^{M+2}-3}{2^{M+1}-2}N^M + \frac{2^{M+2}-3}{2^{M+1}-2} + \frac{3}{2}\left(2^M - 1\right)$ additions, and a special sequence reordering. This reordering requires $\left(\frac{1}{2} - \frac{1}{N}\right)N^M$ multiplications, $\left(\frac{1}{2} - \frac{1}{N}\right)N^M$ additions, $\left(\frac{1}{2} - \frac{1}{N}\right)N^M$ modulo operations, and $\left(\frac{1}{2} - \frac{1}{N}\right)N^M$ bit-shift operations. These requirements along with those of the proposed algorithms are given in Tables 2.2 and 2.3 for $M = 4$ and 5. In these tables, m_p and a_p denote the total number of multiplications and additions required by the algorithm in [68]. The advantage of a slight reduction in the number of arithmetic operations provided by the approach in [68] compared to that of the proposed algorithms is achieved at the expense of certain severe drawbacks, as explained below.

Table 2.2: Number of non-trivial operations required for the computation of the 4-D DHT using the 4mult-2add scheme

	Algorithm in [68]					Proposed Radix-$(2\times2\times2\times2)$ FHTs		
Transform Size $N\times N\times N\times N$	Mults. /point	Adds. /point	M.+A. /point	Modulo /point	Bit-shift /point	Mults. /point	Adds. /point	M.+A. /point
$2^3\times2^3\times2^3\times2^3$	0.84	11,84	12.69	0.37	0.37	0.46	12.46	12.93
$2^4\times2^4\times2^4\times2^4$	1.40	16.40	17.80	0.43	0.43	1.64	17.17	18.81
$2^5\times2^5\times2^5\times2^5$	1.93	20.93	22.87	0.47	0.47	3.16	21.99	25.15
$2^6\times2^6\times2^6\times2^6$	2.45	25.45	27.90	0.48	0.48	4.86	26.87	31.73
$2^7\times2^7\times2^7\times2^7$	2.95	29.95	32.91	0.49	0.49	6.65	31.78	38.42
$2^8\times2^8\times2^8\times2^8$	3.46	34.46	37.92	0.49	0.49	8.48	36.70	45.18
$2^9\times2^9\times2^9\times2^9$	3.96	38.96	42.93	0.50	0.50	10.33	41.63	51.96
$2^{10}\times2^{10}\times2^{10}\times2^{10}$	4.46	43.46	47.93	0.50	0.50	12.19	46.56	58.76
$2^{11}\times2^{11}\times2^{11}\times2^{11}$	4.96	47.96	52.93	0.50	0.50	14.06	51.50	65.56
$2^{12}\times2^{12}\times2^{12}\times2^{12}$	5.46	52.46	57.93	0.50	0.50	15.94	56.43	72.37

Table 2.3: Number of non-trivial operations required for the computation of the 5-D DHT using the 4mult-2add scheme

	Algorithm in [68]					Proposed Radix-$(2\times2\times2\times2\times2)$ FHTs		
Transform Size $N\times N\times N\times N$	Mults. /point	Adds. /point	M.+A. /point	Modulo /point	Bit-shift /point	Mults. /point	Adds. /point	M.+A. /point
$2^3\times2^3\times2^3\times2^3\times2^3$	0.85	14.86	15.71	0.37	0.37	0.48	15.48	15.96
$2^4\times2^4\times2^4\times2^4\times2^4$	1.42	20.42	21.84	0.43	0.43	1.69	21.21	22.90
$2^5\times2^5\times2^5\times2^5\times2^5$	1.95	25.95	27.90	0.47	0.47	3.26	27.05	30.32
$2^6\times2^6\times2^6\times2^6\times2^6$	2.46	31.46	33.93	0.48	0.48	5.02	32.96	37.99
$2^7\times2^7\times2^7\times2^7\times2^7$	2.97	36.97	39.95	0.49	0.49	6.87	38.90	45.77
$2^8\times2^8\times2^8\times2^8\times2^8$	3.48	42.48	45.96	0.49	0.49	8.76	44.85	53.62
$2^9\times2^9\times2^9\times2^9\times2^9$	3.98	47.98	51.96	0.50	0.50	10.67	50.82	61.49

The main reason behind the importance of the DHT is the fact that the software or hardware implementation of only a single FHT algorithm is sufficient to compute both the forward and inverse transforms. Unfortunately, this very useful property of the DHT in requiring only a single algorithm is lost when we consider the approach reported in [68], since the latter requires the implementation of two different complicated algorithms, one for computing the 1-D type-II-DWTs and the other for the M-D polynomial transforms. Moreover, the approach in [68] requires a special sequence reordering, thus necessitating extra multiplications, additions, modulo operations, and bit-shift operations. Further, these time-consuming overhead operations contribute significantly to an increase in the overall structural complexity of the algorithm. Another drawback of this approach is that it does not have the butterfly-style and in-place computation properties, thereby requiring a large memory. Hence, even though this approach has the advantage of a slight reduction in the number of arithmetic operations compared to that required by the proposed algorithms, its complicated structure and overhead operations make it unsuitable for software or hardware implementation. However, the proposed algorithms possess attractive and highly desirable properties such as the regularity, modularity, simplicity, easy indexing scheme, and the butterfly-style and in-place computations. Thus, they are more suitable for software or hardware implementations.

2.5 Relationship Between the Structures of the M-D Radix-$(2 \times 2 \times ... \times 2)$ Complex-Valued FFT Algorithms and the Proposed M-D Radix-$(2 \times 2 \times ... \times 2)$ FHT Algorithms

In this section, we reformulate the existing M-D radix-$(2 \times 2 \times ... \times 2)$ complex-valued FFT algorithms using an approach similar to that used in Sections 2.2 and 2.3 for deriving the proposed M-D radix-$(2 \times 2 \times ... \times 2)$ FHT algorithms in order to establish a relationship between the M-D radix-$(2 \times 2 \times ... \times 2)$ complex-valued FFT algorithms and the proposed M-D radix-$(2 \times 2 \times ... \times 2)$ FHT algorithms. Such a relationship would be highly desirable in order to investigate the possibility of using a single software or hardware module to compute the DHT as well as the forward and inverse complex-valued DFTs.

Let $\tilde{x}(k_1, ..., k_i, ..., k_M)$ be a complex data sequence of size $(N \times N \times ... \times N)$. Its M-D DFT $\tilde{X}(n_1, ..., n_i, ..., n_M)$, which is also a complex sequence of the same size, is defined by

$$\tilde{X}(n_1, ..., n_i, ..., n_M) = \sum_{k_1=0}^{N-1} ... \sum_{k_i=0}^{N-1} ... \sum_{k_M=0}^{N-1} \tilde{x}(k_1, ..., k_i, ..., k_M) W_N^{\sum_{i=1}^{M} n_i k_i},$$
$$0 \le n_i \le N-1, \ 1 \le i \le M \qquad (2.51)$$

where $W_N = \exp(-j2\pi/N)$. The input and output sequences in (2.51) can be expressed in terms of their real and imaginary parts, respectively, by

$$\tilde{x}(k_1, ..., k_i, ..., k_M) = x^{re}(k_1, ..., k_i, ..., k_M) + jx^{im}(k_1, ..., k_i, ..., k_M) \qquad (2.52)$$

and

$$\tilde{X}(n_1, ..., n_i, ..., n_M) = X^{re}(n_1, ..., n_i, ..., n_M) + jX^{im}(n_1, ..., n_i, ..., n_M) \qquad (2.53)$$

2.5.1 An M-D Radix-$(2 \times 2 \times ... \times 2)$ DIF Complex-Valued FFT Algorithm

To derive the M-D radix-$(2 \times 2 \times ... \times 2)$ complex-valued FFT algorithm using the DIF decomposition, we replace k_i and n_i in (2.51), respectively, by the expressions given in (2.10) and (2.11). Then, we see that (2.51) corresponds to a set of 2^M size-$\left(\frac{N}{2} \times \frac{N}{2} \times ... \times \frac{N}{2}\right)$ M-D DFTs given by

$$\widetilde{X}(2n_1+p_1,...,2n_i+p_i,...,2n_M+p_M) = \sum_{k_1=0}^{N/2-1} ... \sum_{k_i=0}^{N/2-1} ... \sum_{k_M=0}^{N/2-1} \widetilde{F}_{k_1,...,k_i,...,k_M}(p_1,...,p_i,...,p_M) W_{N/2}^{\sum_{i=1}^{M} n_i k_i}, \quad 0 \le n_i \le \frac{N}{2}-1,\ p_i = 0 \tag{2.54}$$

where their 2^M input sequences are given by

$$\widetilde{F}_{k_1,...,k_i,...,k_M}(p_1,...,p_i,...,p_M) = W_N^{\sum_{i=1}^{M} p_i k_i} \left[\sum_{q_1=0}^{1} ... \sum_{q_i=0}^{1} ... \sum_{q_M=0}^{1} \widetilde{x}\left(k_1+q_1\frac{N}{2},...,k_i+q_i\frac{N}{2},...,k_M+q_M\frac{N}{2}\right)(-1)^{\sum_{i=1}^{M} p_i q_i} \right], \quad p_i = 0, 1 \tag{2.55}$$

For a given combination of k_i's, (2.55) is recognized as a size-$(2 \times 2 \times ... \times 2)$ M-D DFT whose output samples are multiplied by the twiddle factors. This DFT is similar to the size-$(2 \times 2 \times ... \times 2)$ M-D DHT. Hence, using (2.3), (2.55) can be expressed in a matrix form as

$$\widetilde{\mathbf{F}}_{k_1,...,k_i,...,k_M} = \mathbf{D}^{w}_{k_1,...,k_i,...,k_M} \mathbf{H}_{2^M} \widetilde{\mathbf{x}}_{k_1,...,k_i,...,k_M} \tag{2.56}$$

where the mth component of the input vector $\widetilde{\mathbf{x}}_{k_1,\ldots,k_i,\ldots,k_M}$ is given by

$$\begin{aligned}\widetilde{x}_{k_1,\ldots,k_i,\ldots,k_M}(m) &= \widetilde{x}\left(k_1+m_1\frac{N}{2},\ldots,k_i+m_i\frac{N}{2},\ldots,k_M+m_M\frac{N}{2}\right)\\ &= x^{re}\left(k_1+m_1\frac{N}{2},\ldots,k_i+m_i\frac{N}{2},\ldots,k_M+m_M\frac{N}{2}\right)\\ &\quad +jx^{im}\left(k_1+m_1\frac{N}{2},\ldots,k_i+m_i\frac{N}{2},\ldots,k_M+m_M\frac{N}{2}\right)\end{aligned} \tag{2.57}$$

and the (m, m)th component of the diagonal matrix $\mathbf{D}^w_{k_1,\ldots,k_i,\ldots,k_M}$ by

$$\begin{aligned}d^w_{k_1,\ldots,k_i,\ldots,k_M}(m,m) &= \exp\left(-j\frac{2\pi}{N}\sum_{i=1}^{M}m_ik_i\right)\\ &= d^c_{k_1,\ldots,k_i,\ldots,k_M}(m,m)-jd^s_{k_1,\ldots,k_i,\ldots,k_M}(m,m),\end{aligned} \tag{2.58}$$

The components of the vector $\widetilde{\mathbf{F}}_{k_1,\ldots,k_i,\ldots,k_M}$ in (2.56) are related to the input sequence of (2.54) by

$$\begin{aligned}\widetilde{F}_{k_1,\ldots,k_i,\ldots,k_M}(m) &= \widetilde{F}_{k_1,\ldots,k_i,\ldots,k_M}(m_1,\ldots,m_i,\ldots,m_M)\\ &= F^{re}_{k_1,\ldots,k_i,\ldots,k_M}(m_1,\ldots,m_i,\ldots,m_M)\\ &\quad +jF^{im}_{k_1,\ldots,k_i,\ldots,k_M}(m_1,\ldots,m_i,\ldots,m_M)\end{aligned} \tag{2.59}$$

The input vector $\widetilde{\mathbf{x}}_{k_1,\ldots,k_i,\ldots,k_M}$, the vector $\widetilde{\mathbf{F}}_{k_1,\ldots,k_i,\ldots,k_M}$ and the matrix $\mathbf{D}^w_{k_1,\ldots,k_i,\ldots,k_M}$ are complex-valued, which can be expressed in terms of their real and imaginary parts, respectively, by

$$\widetilde{\mathbf{x}}_{k_1,\ldots,k_i,\ldots,k_M} = \mathbf{x}^{re}_{k_1,\ldots,k_i,\ldots,k_M} + j\mathbf{x}^{im}_{k_1,\ldots,k_i,\ldots,k_M}, \tag{2.60}$$

$$\widetilde{\mathbf{F}}_{k_1,\ldots,k_i,\ldots,k_M} = \mathbf{F}^{re}_{k_1,\ldots,k_i,\ldots,k_M} + j\mathbf{F}^{im}_{k_1,\ldots,k_i,\ldots,k_M}, \tag{2.61}$$

and

$$\mathbf{D}^w_{k_1,\ldots,k_i,\ldots,k_M} = \mathbf{D}^c_{k_1,\ldots,k_i,\ldots,k_M} - j\mathbf{D}^s_{k_1,\ldots,k_i,\ldots,k_M} \tag{2.62}$$

Substituting (260)-(2.62) in (2.56), we get a matrix form for the butterfly of the M-D radix-$(2 \times 2 \times ... \times 2)$ DIF complex-valued FFT algorithm

$$\begin{bmatrix} \mathbf{F}^{re}_{k_1,...,k_i,...,k_M} \\ \mathbf{F}^{im}_{k_1,...,k_i,...,k_M} \end{bmatrix} = \mathbf{T}_{k_1,...,k_i,...,k_M} \begin{bmatrix} \mathbf{H}_{2^M} & \mathbf{O} \\ \mathbf{O} & \mathbf{H}_{2^M} \end{bmatrix} \begin{bmatrix} \mathbf{x}^{re}_{k_1,...,k_i,...,k_M} \\ \mathbf{x}^{im}_{k_1,...,k_i,...,k_M} \end{bmatrix},$$
$$0 \le k_i \le \frac{N}{2} - 1, \ 1 \le i \le M \qquad (2.63)$$

The M-D radix-$(2 \times 2 \times ... \times 2)$ DIF FFT algorithm consists of decomposing a size-$(N \times N \times ... \times N)$ M-D DFT into 2^M size-$\left(\frac{N}{2} \times \frac{N}{2} \times ... \times \frac{N}{2}\right)$ M-D DFTs given by (2.54) in the first stage. This is achieved by repeating $\frac{N^M}{2^M}$ times the butterfly given by (2.63). This decomposition scheme is repeated recursively for each of the new resulting DFTs until size-$(2 \times 2 \times ... \times 2)$ M-D DFTs result in the last stage. The entire algorithm requires $\log_2 N$ stages. It can be seen that the operations such as multiplications, additions, and twiddle factor evaluations or accesses to a lookup table introduced by the twiddle factor matrix of the butterfly given by (2.63) in the last two stages are all trivial. Therefore, it can be shown that the numbers of real multiplications, real additions, storage locations for real data, data transfers, and twiddle factor evaluations or accesses to the lookup table required by the M-D radix-$(2 \times 2 \times ... \times 2)$ DIF FFT algorithm are exactly twice the corresponding numbers required by the proposed M-D radix-$(2 \times 2 \times ... \times 2)$ DIF FHT algorithm. This result is obvious, since the M-D DIF FFT algorithm, actually, computes $2N^M$ real output samples (real and imaginary parts of the output sequence) from $2N^M$ real input samples (real and imaginary parts of the input sequence), whereas the proposed M-D DIF FHT algorithm computes N^M real output samples from N^M real input samples.

Let us now discuss the relationship between the structures of the M-D DIF FHT and FFT algorithms. It is seen that the butterfly given by (2.63) of the DIF FFT algorithm is very similar to the butterfly given by (2.23) of the proposed DIF FHT algorithm; both

the butterflies have the same size and in-place computation property, and are processed by applying the Hadamard operator matrix to the input samples followed by a twiddle factor matrix multiplication. In addition, all the twiddle factors required to be evaluated or read from a lookup table in a given stage of the DIF FHT algorithm are also required in the corresponding stage of the DIF FFT algorithm and the twiddle factor matrices of the two butterflies differ only in the sign of some elements. This difference in sign depends only on the dimension M. The special butterflies that can be used to further reduce the arithmetic operations in the DIF FHT algorithm can also be used in the DIF FFT algorithm for the same objective. It is interesting to note that, for computing the inverse DFT, we use the twiddle factor matrix $\mathbf{T}^t_{k_1,\ldots,k_i,\ldots,k_M}$ instead of $\mathbf{T}_{k_1,\ldots,k_i,\ldots,k_M}$ in (2.63). Therefore, the butterfly of the DIF FHT algorithm is more similar to that used for computing the inverse DFT than to that used for the forward DFT. Moreover, the similarity between the butterflies of the DIF FHT and FFT algorithms is comparable to the similarity between the butterflies used for the forward and inverse DFTs. The decomposition processes in the DIF FHT and FFT algorithms are very similar; both the algorithms have the same number of stages and perform similar operations. However, the number of butterflies required in each stage for the DIF FFT is twice that needed by the DIF FHT. This is due to the fact that in the DIF FFT algorithm all the indices k_i's vary from 0 to $\left(\frac{N}{2}-1\right)$ as indicated in (2.63), whereas in the DIF FHT algorithm, one of these indices must vary only from 0 to $\left(\frac{N}{4}-1\right)$. In order to make the indexing process of the DIF FHT algorithm identical to that of the DIF FFT algorithm, the first half of the data that can be indexed by $(k_1,\ldots,k_i,\ldots,k_M)$ in the FHT algorithm is stored at the locations designated for the real part in the FFT algorithm and the second half of the data that can be indexed by $\left(\frac{N}{2}-k_1,\ldots,\frac{N}{2}-k_i,\ldots,\frac{N}{2}-k_M\right)$ is stored at the locations designated for the imaginary part. Finally, this close relationship between the two algorithms, which we have established, can be exploited to develop a single software or hardware module for the implementation of the algorithms to compute the forward and

inverse M-D complex-valued DFT as well as the M-D DHT. It should be noted that the M-D DHT can be used for efficiently computing the M-D forward and inverse real-valued DFTs as well.

2.5.2 An M-D Radix-$(2 \times 2 \times ... \times 2)$ DIT Complex-Valued FFT Algorithm

To derive the M-D radix-$(2 \times 2 \times ... \times 2)$ complex-valued FFT algorithm using the DIT decomposition, we replace k_i and n_i in (2.51) by the expressions given in (2.32) and (2.33), respectively. Then, we obtain

$$\widetilde{X}\left(n_1 + p_1\frac{N}{2}, ..., n_i + p_i\frac{N}{2}, ..., n_M + p_M\frac{N}{2}\right) = \sum_{q_1=0}^{1} ... \sum_{q_i=0}^{1} ... \sum_{q_M=0}^{1} \left[\widetilde{G}_{n_1,...,n_i,...,n_M}(q_1, ..., q_i, ..., q_M) W_N^{\sum_{i=1}^{M} n_i q_i}\right] (-1)^{\sum_{i=1}^{M} p_i q_i},$$
$$p_i = 0, 1 \quad (2.64)$$

where

$$\widetilde{G}_{n_1,...,n_i,...,n_M}(q_1, ..., q_i, ..., q_M) = \sum_{k_1=0}^{N/2-1} ... \sum_{k_i=0}^{N/2-1} ... \sum_{k_M=0}^{N/2-1} \widetilde{x}(2k_1 + q_1, ..., 2k_i + q_i, ..., 2k_M + q_M) W_{N/2}^{\sum_{i=1}^{M} n_i k_i},$$
$$0 \le n_i \le \frac{N}{2} - 1, \; q_i = 0, 1 \qquad (2.65)$$

For all combinations of q_i's, (2.65) corresponds to a set of 2^M size-$\left(\frac{N}{2} \times \frac{N}{2} \times ... \times \frac{N}{2}\right)$ M-D DFTs. It is clear that for a given combination of n_i's, (2.64) is recognized as a size-$(2 \times 2 \times ... \times 2)$ M-D DFT whose input sequences are obtained from (2.65) after multiplications by the twiddle factors. Therefore, exploiting the representation given by (2.3),

(2.64) can be expressed in a matrix form as

$$\widetilde{\mathbf{X}}_{n_1,\ldots,n_i,\ldots,n_M} = \mathbf{H}_{2^M}\mathbf{D}^{w}_{n_1,\ldots,n_i,\ldots,n_M}\widetilde{\mathbf{G}}_{n_1,\ldots,n_i,\ldots,n_M} \tag{2.66}$$

where the components of the vector $\widetilde{\mathbf{G}}_{n_1,\ldots,n_i,\ldots,n_M}$ are obtained from the output samples of the DHTs given by (2.65) as

$$\begin{aligned}\widetilde{G}_{n_1,\ldots,n_i,\ldots,n_M}(m) &= \widetilde{G}_{n_1,\ldots,n_i,\ldots,n_M}(m_1,\ldots,m_i,\ldots,m_M)\\ &= G^{re}_{n_1,\ldots,n_i,\ldots,n_M}(m_1,\ldots,m_i,\ldots,m_M)\\ &\quad + jG^{im}_{n_1,\ldots,n_i,\ldots,n_M}(m_1,\ldots,m_i,\ldots,m_M)\end{aligned} \tag{2.67}$$

The desired output samples are related to the components of the output vector $\widetilde{\mathbf{X}}_{n_1,\ldots,n_i,\ldots,n_M}$ of (2.66) as given by

$$\begin{aligned}\widetilde{X}_{n_1,\ldots,n_i,\ldots,n_M}(m) &= \widetilde{X}\left(n_1+m_1\frac{N}{2},\ldots,n_i+m_i\frac{N}{2},\ldots,n_M+m_M\frac{N}{2}\right)\\ &= X^{re}\left(n_1+m_1\frac{N}{2},\ldots,n_i+m_i\frac{N}{2},\ldots,n_M+m_M\frac{N}{2}\right)\\ &\quad + jX^{im}\left(n_1+m_1\frac{N}{2},\ldots,n_i+m_i\frac{N}{2},\ldots,n_M+m_M\frac{N}{2}\right),\end{aligned} \tag{2.68}$$

The input and output vectors of (2.66) can be expressed in terms of their real and imaginary parts by

$$\widetilde{\mathbf{X}}_{n_1,\ldots,n_i,\ldots,n_M} = \mathbf{X}^{re}_{n_1,\ldots,n_i,\ldots,n_M} + j\mathbf{X}^{im}_{n_1,\ldots,n_i,\ldots,n_M} \tag{2.69}$$

and

$$\widetilde{\mathbf{G}}_{n_1,\ldots,n_i,\ldots,n_M} = \mathbf{G}^{re}_{n_1,\ldots,n_i,\ldots,n_M} + j\mathbf{G}^{im}_{n_1,\ldots,n_i,\ldots,n_M} \tag{2.70}$$

respectively. Using (2.62), (2.69) and (2.70) in (2.66), we have a matrix form for the butterfly of the M-D radix-$(2 \times 2 \times ... \times 2)$ DIT complex-valued FFT algorithm, as given by

$$\begin{bmatrix} \mathbf{X}^{re}_{n_1,...,n_i,...,n_M} \\ \mathbf{X}^{im}_{n_1,...,n_i,...,n_M} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{2^M} & \mathbf{O} \\ \mathbf{O} & \mathbf{H}_{2^M} \end{bmatrix} \mathbf{T}_{n_1,...,n_i,...,n_M} \begin{bmatrix} \mathbf{G}^{re}_{n_1,...,n_i,...,n_M} \\ \mathbf{G}^{im}_{n_1,...,n_i,...,n_M} \end{bmatrix},$$

$$0 \le n_i \le \frac{N}{2} - 1, \ 1 \le i \le M \qquad (2.71)$$

In the M-D DIT FFT algorithm, the computation of the size-$(N \times N \times ... \times N)$ M-D DFT given by (2.51) consists of computing 2^M size-$\left(\frac{N}{2} \times \frac{N}{2} \times ... \times \frac{N}{2}\right)$ M-D DFTs given by (2.65) whose output sequences are used in (2.64) for computing the desired output samples. The desired M-D DFT is obtained by repeating $\frac{N^M}{2^M}$ times the butterfly given by (2.71). This computation process can be applied recursively for the computation of each of the required DFTs until only size-$(2 \times 2 \times ... \times 2)$ M-D DFTs need to be computed. The entire algorithm requires $\log_2 N$ stages. It can be seen that the operations such as multiplications, additions, and twiddle factor evaluations or accesses to a lookup table introduced by the twiddle factor matrix of the butterfly given by (2.71) in the first two stages of the computation are all trivial. It can be shown that the complexity of the M-D DIT FFT algorithm is identical to that of the M-D DIF FFT algorithm.

A discussion similar to that carried out above for establishing a relationship between the M-D DIF FHT and FFT algorithms can be carried out to establish a relationship between the M-D DIT radix-$(2 \times 2 \times ... \times 2)$ FHT and FFT algorithms. The similarity between the proposed DIT FHT algorithm and the DIT FFT algorithm can be clearly seen from Figs. 2.4 and 2.5. If the first half of the data that can be indexed by $(n_1, ..., n_i, ..., n_M)$ in the FHT algorithm is stored at the locations designated for the real part in the FFT algorithm and the second half of the data that can be indexed by $\left(\frac{N}{2} - n_1, ..., \frac{N}{2} - n_i, ..., \frac{N}{2} - n_M\right)$ is stored at the locations designated for the imaginary part, then the butterflies of the DIT FHT and

FFT algorithms for the corresponding dimension differ only in the value of e. That is, $e = 1$ for the DIT FFT algorithms and $e = -1$ for the DIT FHT algorithms. This similarity is due to the fact that the twiddle factor matrix of the butterfly given by (2.71) of the M-D DIT FFT algorithm and the twiddle factor matrix given by (2.42) of the butterfly given by (2.41) of the proposed M-D DIT FHT algorithm differ only in the diagonal matrix $\mathbf{E}_{2M}$. Note that, $\mathbf{E}_{2M}$ depends only on the dimension M and its elements are +1 or -1.

It is interesting to note that such a close relationship between the proposed M-D FHT algorithms and the corresponding complex-valued FFT algorithms does not exist between the existing real-valued [36]-[41] and complex-valued FFT algorithms. Moreover, the existing real-valued FFT algorithms have complicated structures compared to those of the proposed FHT algorithms and, in general, their forward and inverse versions are completely different.

2.6 Summary

In this chapter, M-D radix-$(2 \times 2 \times ... \times 2)$ DIF and DIT FHT algorithms have been proposed for efficiently computing the DHT of any dimension by using an appropriate index mapping and the Kronecker product. It has been shown that the proposed algorithms are more effective and highly suitable for hardware and software implementations compared to all the existing M-D FHT algorithms. The butterflies of the proposed algorithms are characterized by simple closed-form expressions that allow easy implementations of these algorithms for any dimension. The butterfly for a given dimension M can be readily obtained by assigning the corresponding value to M in the butterflies of the proposed FHT algorithms. For example, the butterflies of the existing 1-D radix-2 [45], 2-D radix-(2×2) [58] and 3-D radix-$(2 \times 2 \times 2)$ [65] DIT FHT algorithms, can be easily obtained by letting $M = 1$, 2 and 3, respectively, in the general butterfly of the proposed M-D radix-$(2 \times 2 \times ... \times 2)$ DIT FHT algorithm. Similarly, for $M = 1$ and 3, the general butterfly of the proposed M-D

radix-$(2 \times 2 \times ... \times 2)$ DIF FHT algorithm reduces to those of the existing 1-D radix-2 [45] and 3-D radix-$(2 \times 2 \times 2)$ [66] DIF FHT algorithms, respectively. Moreover, the proposed algorithms possess properties such as high regularity, simplicity and in-place computation that are highly desirable characteristics for software and hardware implementations, especially for multidimensional applications. A close relationship between the proposed M-D radix-$(2 \times 2 \times ... \times 2)$ FHT algorithms and the M-D radix-$(2 \times 2 \times ... \times 2)$ complex-valued FFT algorithms has been established. This relationship along with the fact that the DHT is an alternative to the DFT for real data, can enable one to use a single software or hardware module for computing the forward and inverse M-D DFTs for real- or complex-valued data and M-D DHTs with a little or no modifications.

One of the most interesting results that have been presented in this chapter is the general decomposition method, which is based on the radix-2 approach, valid for any dimension, applicable either for the case of the DHT or DFT, and significantly reducing the complexity of the algorithms. This method will be further exploited in developing efficient FFT and FHT algorithms in the succeeding chapters.

Chapter 3

Output-Pruning of DFTs

3.1 Introduction

Chapter 2 was devoted to the fast computation of full-length (size) transforms, by exploring the radix-2 approach. For the case of the Fourier transform, there are some applications where a significant part of the output samples of the DFT are not needed, for example, where only a narrow spectrum band is desired. The direct use of a full-length FFT algorithm in such applications is not advantageous, and in order to reduce the arithmetic complexity, methods for pruning 1-D [101]-[106] and 2-D [107]-[109] DFTs have been proposed in the literature.

In this chapter, the problem of increasing the computational and structural efficiencies of FFT algorithms for such applications is considered and efficient methods for pruning output samples of the DFT, based on the radix-2 approach, are proposed for both the 1-D [110] and 2-D [111] cases. These two cases are considered in detail in Sections 3.2 and 3.3, respectively.

3.2 Output-Pruning of the 1-D DFT

One of the methods of increasing the computational efficiency of an FFT algorithm in applications, where only a narrow spectrum band is of interest, is to remove unnecessary computations from the FFT algorithm. This case was studied by Markel, who proposed the well-known pruning algorithm [101] based on a modification of the radix-2 DIT FFT algorithm. Using (2.51), the 1-D DFT of length N is given by

$$\widetilde{X}(n) = \sum_{k=0}^{N-1} \widetilde{x}(k) W_N^{nk}, \; 0 \leq n \leq N-1 \tag{3.1}$$

Assuming that only the first L, $(L < N = 2^r)$, output points are needed and is restricted to be a power of two, $L = 2^d$, Markel's algorithm retains the first d stages of the radix-2 DIT FFT algorithm and alters the remaining $(r-d)$ stages by avoiding the computations of the undesired $(N-L)$ points. The ith stage, $(i = d+1$ to $r)$, contains $2^{r-i}L$ half butterflies each requiring 4 real multiplications, 4 real additions, and 2 RAM read and 1 RAM write operations of complex numbers. Note that the complex multiplication can be computed by using 4 real multiplications and 2 real additions (4mult-2add scheme) or by using 3 real multiplications and 3 real additions (3mult-3add scheme). In this chapter, it is assumed that the 4mult-2add scheme is considered. The different requirements of the Markel's algorithm are given in Table 3.1. Skinner proposed a method to prune the input samples by modifying the radix-2 DIT FFT algorithm [102]. Sorensen and Burrus [103] showed how this idea can be adapted to prune the output samples by modifying the radix-2 DIF FFT algorithm and called it as the Skinner algorithm. This Skinner's algorithm retains the first d stages of the radix-2 DIF FFT algorithm and alters the remaining $(r-d)$ stages by avoiding the twiddle factors, and using only additions. The ith stage, $(i = d+1$ to $r)$, requires $2 \times 2^{r-i}L$ real additions, and $2 \times 2^{r-i}L$ RAM read and $2^{r-i}L$ RAM write operations of complex numbers. The requirements of the Skinner's algorithm are also given in Table 3.1. Sorensen and

Table 3.1: Computational complexities of the different algorithms in the computation of a subset of output samples of the 1-D DFT

	Output-pruned radix-2 DIT FFT	
	Markel's algorithm	Proposed algorithm
Real multiplications	$2Nlog_2L-4L$	$2Nlog_2L-\frac{5}{4}N-\frac{7N}{2L}-4L$
Real additions	$3Nlog_2L+2N-4L$	$3Nlog_2L+\frac{3}{2}N-\frac{7N}{4L}-4L$
Address generations of complex numbers	$Nlog_2L+2N-2L$	$Nlog_2L$
Read operations of complex numbers	$Nlog_2L+2N-2L$	$Nlog_2L$
Write operations of complex numbers	$Nlog_2L+N-L$	$Nlog_2L-(N-L)$
	Output-pruned radix-2 DIF FFT	
	Skinner's algorithm	Proposed algorithm
Real multiplications	$2Nlog_2L$	$2Nlog_2L-\frac{5}{4}N-2L$
Real additions	$3Nlog_2L+2N-2L$	$3Nlog_2L+\frac{3}{2}N-3L$
Address generations of complex numbers	$Nlog_2L+2N-2L$	$Nlog_2L$
Read operations of complex numbers	$Nlog_2L+2N-2L$	$Nlog_2L$
Write operations of complex numbers	$Nlog_2L+N-L$	$Nlog_2L-(N-L)$

Burrus proposed another method [103], that seems to be better than the algorithm of Markel or that of Skinner in terms of arithmetic complexity. This is due to the use of the radix-2/4 FFT in the computation of the sub-transforms. However, the overall structural complexity of the algorithm in [103] is much larger than those of the algorithms of Markel and Skinner, which are based on the simple and regular structure of the radix-2 FFT. These three pruning algorithms decrease the number of arithmetic operations relative to the full-length FFT. However, no attempt has been made to minimize the number of data transfers, address generations, and twiddle factor evaluations or accesses to the lookup table.

In this section, we propose efficient algorithms for the pruning of the output samples based on both the DIT and DIF versions of the radix-2 FFT. The main difference between the proposed pruning algorithms and those mentioned above is in the decimation process. In the decomposition of the proposed method, the decimation is carried out only on a few stages, all the other stages being grouped by an appropriate recursive process so as to minimize the number of arithmetic operations, data transfers, address generations and twiddle

factor evaluations or accesses to the lookup table.

3.2.1 Proposed Radix-2 DIT FFT Pruning Algorithm

The proposed method is similar to that of Markel in the sense that both the methods are based on the radix-2 DIT FFT algorithm. The main difference, however, is that in the proposed method only d stages are needed in pruning the output samples. Further, the remaining $(r-d)$ stages are grouped and incorporated in the dth stage of the radix-2 DIT FFT. This grouping provides a significant reduction in the overall structural and computational complexities.

For the case of $L = 2^d < N = 2^r$, the DFT given by (3.1) can be written as

$$\widetilde{X}(m) = \sum_{k=0}^{N-1} \widetilde{x}(k) W_N^{mk}, \qquad m = 0, 1, 2, ..., L-1 \tag{3.2}$$

Therefore, the index k can be expressed by r binary bits, whereas for the index m, d bits are sufficient. Thus, the product mk can be written as

$$mk = AB + CD \tag{3.3}$$

where

$A = 2^{d-1}m_{d-1} + 2^{d-2}m_{d-2} + ... + 2m_1 + m_0,$

$B = 2^{r-1}k_{r-1} + 2^{r-2}k_{r-2} + ... + 2^{r-d+1}k_{r-d+1} + 2^{r-d}k_{r-d},$

$C = 2^{d-1}m_{d-1} + 2^{d-2}m_{d-2} + ... + 2m_1 + m_0,$

$D = 2^{r-d-1}k_{r-d-1} + ... + 2k_1 + k_0,$

m_l, $k_l = 0, 1$ and $l \in [0, r-1]$. We observe that the factor CD in (3.3) cannot provide the advantage of redundant operations when it is used to carry out the DIT decomposition of (3.2). Consequently, in order to compute (3.2) by using a minimum number of stages,

we apply the DIT only when we use the first factor AB in (3.3), and group in one stage all the stages resulting from the factor CD. Hence, we can perform the pruning operation using only the d stages corresponding to the expressions given by

$$\widetilde{X}_i(m_0...m_{i-1}k_{r-i-1}...k_0) = \sum_{k_{r-i}=0}^{1} \widetilde{X}_{i-1}(m_0...m_{i-2}k_{r-i}...k_0)\, W_N^{\left(2^{i-1}m_{i-1}+...+m_0\right)2^{r-i}k_{r-i}} \tag{3.4}$$

and

$$\widetilde{X}_d(m_0...m_{d-1}0...0) = \sum_{k_0=0}^{1} ... \sum_{k_{r-d-1}=0}^{1} \sum_{k_{r-d}=0}^{1} \widetilde{X}_{d-1}(m_0...m_{d-2}k_{r-d}...k_0)$$
$$W_N^{m2^{r-d}k_{r-d}} W_N^{m\left(2^{r-d-1}k_{r-d-1}+...+k_0\right)} \tag{3.5}$$

where $i = 1, 2, ..., d-1$ and $\widetilde{X}_0(k) = \widetilde{x}(k)$. The input is taken in the natural order so as to obtain the output in the bit-reversed sequence. Thus, the desired narrow spectrum is obtained by the bit-reversal of only L points of the dth stage. The set of equations given by (3.4) represents the first $(d-1)$ stages of the radix-2 DIT FFT [32], and (3.5) represents the dth stage obtained by a grouping of the last $(r-d+1)$ stages. Therefore, the problem of pruning the output samples using a radix-2 DIT FFT can be divided into two distinct parts. After expressing the indices by a decimal notation, the first part that contains the $(d-1)$ stages corresponding to (3.4) can be efficiently calculated by the following recursive relation

$$\widetilde{\mathbf{X}}_{m,k,i}^{ou} = \mathbf{H}_2 \begin{bmatrix} 1 & 0 \\ 0 & W_N^{m2^{r-i}} \end{bmatrix} \widetilde{\mathbf{X}}_{m,k,i}^{in},\ i = 1, 2, ..., d-1,$$
$$m = 0, 1, ..., (L/2)-1,\ k = 0, 1, ..., 2^{r-i}-1 \tag{3.6}$$

where

$$\widetilde{\mathbf{X}}^{ou}_{m,k,i} = \begin{bmatrix} \widetilde{X}_i\left(c^0_{m,i}(k)\right) \\ \widetilde{X}_i\left((c^1_{m,i}(k)\right) \end{bmatrix}$$

$$\widetilde{\mathbf{X}}^{in}_{m,k,i} = \begin{bmatrix} \widetilde{X}_{i-1}\left(c^0_{m,i}(k)\right) \\ \widetilde{X}_{i-1}\left((c^1_{m,i}(k)\right) \end{bmatrix},$$

$c^0_{m,i}(k) = 2^{r-i+1}\overleftarrow{m} + k$, $c^1_{m,i}(k) = C^0_{m,i}(k) + 2^{r-i}$, and $\overleftarrow{m}$ denotes the bit-reversed value of m. For each value of i, (3.6) corresponds to a stage. However, for a given stage and for a specific value of m and of k, (3.6) represents a butterfly. For each stage there are $N/2$ such butterflies. Each butterfly requires 2 RAM address generations of complex numbers, 4 real multiplications, 6 real additions, and 2 RAM read and 2 RAM write operations of complex numbers. In order to simplify the procedure for the address generation, we do the bit-reversal of the exponents of the twiddle factors, which are generally precomputed and stored in a lookup table. Recall that the first stage of the radix-2 DIT FFT does not contain any twiddle factor and also the second stage requires only multiplications by $-j$. Therefore, the first $(d-1)$ stages require $(Nlog_2L - N)$ RAM address generations of complex numbers, $(2Nlog_2L - 6N)$ real multiplications, $(3Nlog_2L - 5N)$ real additions, and $(Nlog_2L - N)$ RAM read and $(Nlog_2L - N)$ RAM write operations of complex numbers.

It is seen from (3.6) that the indexing process introduced in the above development not only simplifies the address generation, but has two other advantages. First, it allows the processing of all the butterflies of the ith stage having the same twiddle factors, and this can be easily done by fixing m and varying k from 0 to $\left(2^{r-i}-1\right)$, $i = 1, 2, ..., d-1$. This technique provides a significant reduction in the number of address generations and twiddle factor evaluations or accesses to the lookup table. The second advantage is that, by this indexing process, it is easy to avoid trivial multiplications in (3.6) by starting each stage with $m = 0$ and varying k. Additional savings are possible if the implementation is

designed to take advantage of the twiddle factors at the third stage.

The second part of the proposed algorithm is concerned with the last stage, represented by (3.5). The $(r-d+1)$ summations in (3.5) are replaced by a single summation, and is expressed, after using the decimal notation of the indices, by

$$\widetilde{\mathbf{X}}^{ou}_{m,0,d} = \sum_{k=0}^{(N/L)-1} \left(\mathbf{D}_{m,k} \mathbf{H}_2 \mathbf{D}_m \widetilde{\mathbf{X}}^{in}_{m,k,d} \right), \; m = 0, 1, ..., (L/2)-1 \tag{3.7}$$

where

$$\mathbf{D}_{m,k} = \begin{bmatrix} W_N^{mk} & 0 \\ 0 & W_N^{((L/2)+m)k} \end{bmatrix}$$

and

$$\mathbf{D}_m = \begin{bmatrix} 1 & 0 \\ 0 & W_L^m \end{bmatrix}$$

Since modern processors possess sufficient number of internal fast registers to store the intermediate results, (3.7) can be efficiently implemented without requiring intermediate storing in and loading from the RAM. This reduction in the number of RAM address generations, and read and write operations is one of the advantages of the equation given by (3.7). There are only L points to be computed using (3.7), hence this part of pruning requires only L write operations, and can be implemented as follows. For each value of m, $m = 0, 1, ..., (L/2)-1$, vary k in decreasing order from $(N/L)-1$ to 0, generate the two addresses for reading, read two points from the RAM, process the general butterfly of the radix-2 DIT FFT, multiply the butterfly outputs by the twiddle factors, store the results in registers, repeat the process for the second value of k, add these results to the previous ones and store in the same registers, and repeat the whole process until $k = 0$. Write the final results in RAM using the last two address generated for reading. It is clear from (3.7) that a large number of trivial multiplications can be easily avoided when $m = k = 0$. Thus, it

can be easily deduced that the second part of our decomposition requires N RAM address generations of complex numbers, $6N-(2N/L)-4L$ real multiplications, $7N-(N/L)-4L$ real additions, and N RAM read and L RAM write operations of complex numbers. If we combine the computational complexities of the both the parts, we find that our decomposition presents a significant improvement in comparison with Markel's algorithm, especially in the address generation and data transfer; however, the number of real multiplications and additions is only slightly reduced. In order to further decrease this number, we rearrange (3.7) as

$$\widetilde{\mathbf{X}}^{ou}_{m,0,d} = \sum_{k=0}^{(N/4L)-1} \mathbf{D}_{m,k}\overline{\mathbf{X}}^{in}_{m,k},\ m=0,1,...,(L/2)-1 \tag{3.8}$$

where

$$\begin{aligned}\overline{\mathbf{X}}^{in}_{m,k} &= \mathbf{H}_2\mathbf{D}_m\widetilde{\mathbf{X}}^{in}_{m,k,d}+\mathbf{J}_2\mathbf{H}_2\mathbf{D}^2_m\widetilde{\mathbf{X}}^{in}_{m,k+\frac{N}{2L},d} \\ &\quad +\mathbf{c}_2\left(\mathbf{H}_2\mathbf{D}^1_m\widetilde{\mathbf{X}}^{in}_{m,k+\frac{N}{4L},d}+\mathbf{J}_2\mathbf{H}_2\mathbf{D}^3_m\widetilde{\mathbf{X}}^{in}_{m,k+\frac{3N}{4L},d}\right),\end{aligned} \tag{3.9}$$

$$\mathbf{c}_2 = \begin{bmatrix} 1 & 0 \\ 0 & \frac{(1-j)}{\sqrt{2}} \end{bmatrix}, \tag{3.10}$$

$$\mathbf{J}_2 = \begin{bmatrix} 1 & 0 \\ 0 & -j \end{bmatrix}, \tag{3.11}$$

$$\mathbf{D}^1_m = \begin{bmatrix} W^m_{4L} & 0 \\ 0 & W^{5m}_{4L} \end{bmatrix},$$

$$\mathbf{D}^2_m = W^m_{4L}\mathbf{D}^1_m$$

and

$$\mathbf{D}^3_m = W^{2m}_{4L}\mathbf{D}^1_m.$$

A close examination of the computational complexity of (3.8), taking into account the number of operations introduced by each matrix in the calculation of (3.9) and the number of trivial multiplications that can be avoided when $m = k = 0$, shows that the second part of pruning requires only $\{(19/4)N - (7N/2L) - 4L\}$ real multiplications and $\{(13/2)N - (7N/4L) - 4L\}$ real additions.

The overall computational complexity of the proposed algorithm is summarized in Table 3.1. The comparison of this complexity is made only with that of Markel's pruning algorithm, since the both these are based on the radix-2 DIT FFT algorithm. It is clear that the proposed algorithm is far better than Markel's, since all the operations that contribute significantly to the execution time are reduced.

There remains another point, which needs to be discussed. It concerns the number of twiddle factor evaluations or accesses to the lookup table. It can be shown that the number of complex twiddle factors needed to compute (3.8) is $\{(9N/4) - (3N/2L) - 2L - 1\}$. To reduce this number, we introduce a technique to implement (3.8) with a minimum number of twiddle factor evaluations or accesses to the lookup table. Since $\mathbf{D}_{m,k}$ is a diagonal matrix and $W_N^{m(k+1)} = W_N^{mk} W_N^m$, we get $\mathbf{D}_{m,k+1} = \mathbf{D}_{m,k}\mathbf{D}_{m,1}$. Hence, by performing an appropriate factorization, (3.8) can be efficiently implemented by expressing it as

$$\begin{aligned}\widetilde{\mathbf{X}}^{ou}_{m,0,d} = \overline{\mathbf{X}}^{in}_{m,0} + \mathbf{D}_{m,1}(\overline{\mathbf{X}}^{in}_{m,1} + \mathbf{D}_{m,1}(\overline{\mathbf{X}}^{in}_{m,2} + \ldots \\ \ldots + \mathbf{D}_{m,1}(\overline{\mathbf{X}}^{in}_{m,\frac{N}{4L}-2} + \mathbf{D}_{m,1}(\overline{\mathbf{X}}^{in}_{m,\frac{N}{4L}-1}))\ldots))\end{aligned} \tag{3.12}$$

It is seen from (3.12) that for a given value of m, $m = 0, 1, \ldots, (L/2) - 1$, it requires only 9 complex twiddle factors. Note that for $m = 0$, one complex twiddle factor is enough. Thus, the number of complex twiddle factor evaluations or accesses to the lookup table is only $(9L/2) - 8$.

3.2.2 Proposed Radix-2 DIF FFT Pruning Algorithm

In this section, a second efficient algorithm for the pruning of the output samples is proposed. It is similar to that of Skinner in the sense that both the algorithms are based on the radix-2 DIF FFT algorithm. This algorithm is derived by using techniques similar to those in the previous section. It can be shown that it consists of two parts. The first part can be efficiently calculated by the recursive equation given by

$$\widetilde{\mathbf{X}}^{ou}_{m,k,i} = \begin{bmatrix} 1 & 0 \\ 0 & W_N^{2^{i-1}k} \end{bmatrix} \mathbf{H}_2 \widetilde{\mathbf{X}}^{in}_{m,k,i} \; i = 1, 2, ..., d-1,$$
$$k = 0, 1, ..., 2^{r-i}-1, \; m = 0, 1, ..., (L/2)-1 \tag{3.13}$$

The second part is given by the expression

$$\widetilde{\mathbf{X}}^{ou}_{m,0,d} = \overline{\mathbf{X}}^{in}_{m,0} + \mathbf{D}_{0,1}(\overline{\mathbf{X}}^{in}_{m,1} + \mathbf{D}_{0,1}(\overline{\mathbf{X}}^{in}_{m,2} + ...$$
$$... + \mathbf{D}_{0,1}(\overline{\mathbf{X}}^{in}_{m,\frac{N}{4L}-2} + \mathbf{D}_{0,1}(\overline{\mathbf{X}}^{in}_{m,\frac{N}{4L}-1}))...)) \tag{3.14}$$

where $m = 0, 1, ..., (L/2)-1$. Since the entire second part needs only one complex twiddle factor, this decomposition not only reduces the number of twiddle factor evaluations or accesses to the lookup table, but also is suitable for the lookup table reduction techniques. Note that the first d stages of Skinner's algorithm can be computed by any radix-2 DIF FFT. Assuming now that they are computed by (3.13), the dth stage requires $(N/L) - 1$ complex twiddle factors, since in general $N \gg L$. Other advantages of the proposed algorithm compared to that of Skinner are included in Table 3.1.

3.3 Output-Pruning of the 2-D DFT

Pruning algorithms are extremely important to avoid unnecessary computations in an FFT algorithm, especially in the 2-D case where the task is more complex and the volume of data larger. Using (2.51), the 2-D DFT of size $(N \times N)$ is given by

$$\widetilde{X}(n_1, n_2) = \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} \widetilde{x}(k_1, k_2) W_N^{n_1 k_1 + n_2 k_2}, \quad 0 \le n_1, n_2 \le N-1 \tag{3.15}$$

Some of the existing 1-D FFT pruning methods reviewed in Section 3.2 have been extended in the literature to the 2-D case, and there are two widely used approaches for this purpose [107]-[109]. One of them is based on the row-column decomposition and the other on a logical extension of the pruned 1-D FFT algorithm. It has been shown in [107] and [108] that the latter approach is better than the former in terms of the arithmetic complexity. For the case where only the first $L \times L$, $\left(L = 2^d < N = 2^r\right)$, output points of the DFT given by (3.15) are needed, the latter approach [107], [108] retains the first d stages of the radix-(2×2) DIT FFT algorithm and alters the remaining $(r-d)$ stages by avoiding the computations of the undesired $\left(N^2 - L^2\right)$ output points. The ith stage, $i = d+1$, ..., r, contains $4^{r-i}L^2$ partial butterflies each requiring 3 complex multiplications, 3 complex additions, 4 RAM read and 1 RAM write operations of complex numbers. Note that the number of RAM address generations of complex numbers is equal to the corresponding number of RAM read operations of complex numbers. These requirements of the radix-(2×2) DIT FFT pruning algorithm [107], [108] are summarized in Table 3.2. Just as in the case of the existing radix-2 DIT FFT pruning algorithm, the existing radix-(2×2) DIT FFT pruning algorithm [107], [108] has been derived mainly to reduce the number of arithmetic operations relative to the full-size 2-D FFT. However, efficient 2-D FFT pruning algorithms are not available to minimize the computational and structural complexities.

In this section, we explore the pruning method introduced in Section 3.2.1 for the case

Table 3.2: Computational complexities of the different algorithms in the computation of a subset of output samples of the 2-D DFT

	Output-pruned radix-(2×2) DIT FFT	
	Algorithm in [107], [108]	Proposed algorithm
Real multiplications	$3N^2log_2L+4N^2-4L^2$	$3N^2log_2L+\frac{11}{8}N^2-4L^2$
Real additions	$\frac{11}{2}N^2log_2L+4N^2-4L^2$	$\frac{11}{2}N^2log_2L+\frac{89}{32}N^2-4L^2$
Address generations of complex numbers	$N^2log_2L+\frac{4}{3}N^2-\frac{4}{3}L^2$	N^2log_2L
Read operations of complex numbers	$N^2log_2L+\frac{4}{3}N^2-\frac{4}{3}L^2$	N^2log_2L
Write operations of complex numbers	$N^2log_2L+\frac{1}{3}N^2-\frac{1}{3}L^2$	$N^2log_2L-N^2+L^2$

of the radix-2 DIT FFT algorithm to develop an algorithm for efficiently pruning the output samples of the radix-(2×2) DIT FFT.

3.3.1 Proposed Radix-(2×2) DIT FFT Pruning Algorithm

Without loss of generality, we assume that only the first $L\times L$, $\left(L=2^d<N=2^r\right)$, output points are needed. Then, the 2-D DFT given by (3.15) can be expressed as

$$\widetilde{X}(m_1,m_2)=\sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1}\widetilde{x}(k_1,k_2)W_N^{m_1k_1+m_2k_2},\ m_1,m_2=0,1,2,...,L-1 \tag{3.16}$$

Therefore, each of the indices k_1 and k_2 can be expressed by r binary bits, whereas for each of the indices m_1 and m_2, d bits are sufficient. Thus, the product m_qk_q, $q=1,2$, can be written as

$$m_qk_q=A_qB_q+C_qD_q \tag{3.17}$$

where

$$A_q=2^{d-1}(m_q)_{d-1}+2^{d-2}(m_q)_{d-2}+...+2(m_q)_1+(m_q)_0,$$
$$B_q=2^{r-1}(k_q)_{r-1}+2^{r-2}(k_q)_{r-2}+...+2^{r-d+1}(k_q)_{r-d+1}+2^{r-d}(k_q)_{r-d},$$
$$C_q=2^{d-1}(m_q)_{d-1}+2^{d-2}(m_q)_{d-2}+...+2(m_q)_1+(m_q)_0,$$
$$D_q=2^{r-d-1}(k_q)_{r-d-1}+...+2(k_q)_1+(k_q)_0,$$

$(m_q)_l$, $(k_q)_l = 0, 1$ for $q = 1, 2$ and $l \in [0, r-1]$. We observe that the factor $C_q D_q$ in (3.17) cannot provide the advantage of redundant operations when it is used to carry out the DIT decomposition of (3.16). Consequently, in order to compute (3.16) by using a minimum number of stages, we apply the DIT only when we use the first factor $A_q B_q$ in (3.17), and group in one stage all the stages resulting from the factor $C_q D_q$. Hence, we can perform the pruning operation using only the d stages corresponding to the expressions given by

$$\widetilde{X}_i((m_1)_0...(m_1)_{i-1}(k_1)_{r-i-1}...(k_1)_0, (m_2)_0...(m_2)_{i-1}(k_2)_{r-i-1}...(k_2)_0) =$$
$$\sum_{(k_1)_{r-i}=0}^{1} \sum_{(k_2)_{r-i}=0}^{1} \widetilde{X}_{i-1}((m_1)_0...(m_1)_{i-2}(k_1)_{r-i}...(k_1)_0, (m_2)_0...(m_2)_{i-2}(k_2)_{r-i}...(k_2)_0)$$
$$W_N^{\sum_{q=1}^{2}(2^{i-1}(m_q)_{i-1}+...+(m_q)_0)2^{r-i}(k_q)_{r-i}} \tag{3.18}$$

and

$$\widetilde{X}_d((m_1)_0...(m_1)_{d-1}0...0, (m_2)_0...(m_2)_{d-1}0...0) =$$
$$\sum_{(k_1)_0=0}^{1} ... \sum_{(k_1)_{r-d-1}=0}^{1} \sum_{(k_2)_0=0}^{1} ... \sum_{(k_2)_{r-d-1}=0}^{1}$$
$$\sum_{(k_1)_{r-d}=0}^{1} \sum_{(k_2)_{r-d}=0}^{1} \widetilde{X}_{d-1}((m_1)_0...(m_1)_{d-2}(k_1)_{r-d}...(k_1)_0, (m_2)_0...(m_2)_{d-2}(k_2)_{r-d}...(k_2)_0)$$
$$W_N^{\sum_{q=1}^{2}(2^{d-1}(m_q)_{d-1}+...+(m_q)_0)2^{r-d}(k_q)_{r-d}}$$
$$W_N^{\sum_{q=1}^{2}(2^{d-1}(m_q)_{d-1}+...+(m_q)_0)(2^{r-d-1}(k_q)_{r-d-1}+...+(k_q)_0)} \tag{3.19}$$

where $i = 1, 2, ..., d-1$, $\widetilde{X}_0(k_1, k_2) = \widetilde{x}(k_1, k_2)$. The input is taken in the natural order so as to obtain the output in the bit-reversed sequence. Thus, the desired narrow 2-D spectrum is obtained by the bit-reversal of only $L \times L$ points of the dth stage. The set of equations given by (3.18) may be recognized as corresponding to the first $(d-1)$ stages of the radix-(2×2) DIT FFT [32]. The remaining $(r-d+1)$ stages of the FFT represented by (3.19) may be

grouped together into a single stage, the dth stage. Therefore, the problem of pruning the output samples using a radix-(2×2) DIT FFT can be divided into two distinct parts. After expressing the indices by a decimal notation, the first part that contains the $(d-1)$ stages corresponding to (3.18) can be efficiently calculated by using the recursive relation given by

$$\widetilde{\mathbf{X}}^{ou}_{(m_1,k_1)(m_2,k_2)i} = \mathbf{H}_{2^2} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & W_N^{m_2 2^{r-i}} & 0 & 0 \\ 0 & 0 & W_N^{m_1 2^{r-i}} & 0 \\ 0 & 0 & 0 & W_N^{m_1 2^{r-i} + m_2 2^{r-i}} \end{bmatrix} \widetilde{\mathbf{X}}^{in}_{(m_1,k_1)(m_2,k_2)i},$$

$$i = 1, 2, ..., d-1,\ m_1, m_2 = 0, 1, ..., 2^{i-1}-1,\ k_1, k_2 = 0, 1, ..., 2^{r-i}-1 \quad (3.20)$$

where

$$\widetilde{\mathbf{X}}^{ou}_{(m_1,k_1)(m_2,k_2)i} = \begin{bmatrix} \widetilde{X}_i\left(c^0_{m_1,i}(k_1), c^0_{m_2,i}(k_2)\right) \\ \widetilde{X}_i\left(c^0_{m_1,i}(k_1), c^1_{m_2,i}(k_2)\right) \\ \widetilde{X}_i\left(c^1_{m_1,i}(k_1), c^0_{m_2,i}(k_2)\right) \\ \widetilde{X}_i\left(c^1_{m_1,i}(k_1), c^1_{m_2,i}(k_2)\right) \end{bmatrix},$$

$$\widetilde{\mathbf{X}}^{in}_{(m_1,k_1)(m_2,k_2)i} = \begin{bmatrix} \widetilde{X}_{i-1}\left(c^0_{m_1,i}(k_1), c^0_{m_2,i}(k_2)\right) \\ \widetilde{X}_{i-1}\left(c^0_{m_1,i}(k_1), c^1_{m_2,i}(k_2)\right) \\ \widetilde{X}_{i-1}\left(c^1_{m_1,i}(k_1), c^0_{m_2,i}(k_2)\right) \\ \widetilde{X}_{i-1}\left(c^1_{m_1,i}(k_1), c^1_{m_2,i}(k_2)\right) \end{bmatrix},$$

$c^0_{m_q,i}(k_q) = 2^{r-i+1}\overleftarrow{m_q} + k_q$, $c^1_{m_q,i}(k_q) = C^0_{m_q,i}(k_q) + 2^{r-i}$, and $\overleftarrow{m_q}$ denotes the bit-reversed value of m_q. The second part of the proposed algorithm is concerned with the dth stage, represented by (3.19). After using the decimal notation of the indices, the two $(r-d+1)$-

summations in (3.19) may be replaced by the relation given by

$$\widetilde{\mathbf{X}}^{ou}_{(m_1,0)(m_2,0)d} = \sum_{k_1=0}^{(N/L)-1} \sum_{k_2=0}^{(N/L)-1} \mathbf{D}_{(m_1,k_1)(m_2,k_2)} \mathbf{H}_{22} \mathbf{D}_{m_1 m_2} \widetilde{\mathbf{X}}^{in}_{(m_1,k_1)(m_2,k_2)d},$$
$$m_1, m_2 = 0, 1, ..., (L/2) - 1 \qquad (3.21)$$

where

$$\mathbf{D}_{(m_1,k_1)(m_2,k_2)} = \begin{bmatrix} W_N^{m_1k_1+m_2k_2} & 0 & 0 & 0 \\ 0 & W_N^{m_1k_1+\left(\frac{L}{2}+m_2\right)k_2} & 0 & 0 \\ 0 & 0 & W_N^{\left(\frac{L}{2}+m_1\right)k_1+m_2k_2} & 0 \\ 0 & 0 & 0 & W_N^{\left(\frac{L}{2}+m_1\right)k_1+\left(\frac{L}{2}+m_2\right)k_2} \end{bmatrix}$$

$$\mathbf{D}_{m_1m_2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & W_L^{m_2} & 0 & 0 \\ 0 & 0 & W_L^{m_1} & 0 \\ 0 & 0 & 0 & W_L^{m_1+m_2} \end{bmatrix}$$

In order to further improve the computation of the second part of our decomposition, let us rearrange (3.21) as

$$\widetilde{\mathbf{X}}^{ou}_{(m_1,0)(m_2,0)d} = \sum_{k_1=0}^{(N/4L)-1} \sum_{k_2=0}^{(N/4L)-1} \mathbf{D}_{(m_1,k_1)(m_2,k_2)} \widehat{\mathbf{X}}^{in}_{(m_1,k_1)(m_2,k_2)},$$
$$m_1, m_2 = 0, 1, ..., (L/2) - 1 \qquad (3.22)$$

The expression of the input vector $\widehat{\mathbf{X}}^{in}_{(m_1,k_1)(m_2,k_2)}$ in (3.22) is given by

$$\begin{aligned}\widehat{\mathbf{X}}^{in}_{(m_1,k_1)(m_2,k_2)} = &\left\{\overline{\mathbf{X}}^{in}_{00} + (\mathbf{I}_2 \otimes \mathbf{J}_2)\,\overline{\mathbf{X}}^{in}_{02} + (\mathbf{J}_2 \otimes \mathbf{I}_2)\left(\overline{\mathbf{X}}^{in}_{20} + (\mathbf{I}_2 \otimes \mathbf{J}_2)\,\overline{\mathbf{X}}^{in}_{22}\right)\right.\\ &\left. + (\mathbf{I}_2 \otimes \mathbf{c}_2)\left[\overline{\mathbf{X}}^{in}_{01} + (\mathbf{I}_2 \otimes \mathbf{J}_2)\,\overline{\mathbf{X}}^{in}_{03} + (\mathbf{J}_2 \otimes \mathbf{I}_2)\left(\overline{\mathbf{X}}^{in}_{21} + (\mathbf{I}_2 \otimes \mathbf{J}_2)\,\overline{\mathbf{X}}^{in}_{23}\right)\right]\right\}\\ &+ (\mathbf{c}_2 \otimes \mathbf{I}_2)\left\{\overline{\mathbf{X}}^{in}_{10} + (\mathbf{I}_2 \otimes \mathbf{J}_2)\,\overline{\mathbf{X}}^{in}_{12} + (\mathbf{J}_2 \otimes \mathbf{I}_2)\left(\overline{\mathbf{X}}^{in}_{30} + (\mathbf{I}_2 \otimes \mathbf{J}_2)\,\overline{\mathbf{X}}^{in}_{32}\right)\right.\\ &\left. + (\mathbf{I}_2 \otimes \mathbf{c}_2)\left[\overline{\mathbf{X}}^{in}_{11} + (\mathbf{I}_2 \otimes \mathbf{J}_2)\,\overline{\mathbf{X}}^{in}_{13} + (\mathbf{J}_2 \otimes \mathbf{I}_2)\left(\overline{\mathbf{X}}^{in}_{31} + (\mathbf{I}_2 \otimes \mathbf{J}_2)\,\overline{\mathbf{X}}^{in}_{33}\right)\right]\right\} \quad (3.23)\end{aligned}$$

The input vectors $\overline{\mathbf{X}}^{in}_{\alpha\beta}$, $\alpha, \beta = 0, 1, 2, 3$, in (3.23) are expressed as

$$\overline{\mathbf{X}}^{in}_{\alpha\beta} = \mathbf{H}_{22}\mathbf{D}^{\alpha\beta}\widetilde{\mathbf{X}}^{in}_{(m_1,k_1+\alpha\frac{N}{4L})(m_2,k_2+\beta\frac{N}{4L})d} \quad (3.24)$$

where $\mathbf{D}^{00} = \mathbf{D}_{m_1m_2}$, $\mathbf{D}^{0\beta} = \mathbf{D}^{00}W_{4L}^{\beta m_2}$, and $\mathbf{D}^{\alpha\beta} = \mathbf{D}^{0\beta}W_{4L}^{\alpha m_1}$.

Finally, to implement (3.22) with a minimum number of twiddle factor evaluations or accesses to the lookup table, a recursive technique is introduced. Since $\mathbf{D}_{(m_1,k_1)(m_2,k_2)}$ is a diagonal matrix and $W_N^{m_q(k_q+1)} = W_N^{m_qk_q}W_N^{m_q}$, $q = 1, 2$, we get

$$\begin{aligned}\mathbf{D}_{(m_1,k_1)(m_2,k_2)} &= \mathbf{D}^1_{(m_1,k_1)}\mathbf{D}^2_{(m_2,k_2)},\\ \mathbf{D}^q_{(m_q,k_q+1)} &= \mathbf{D}^q_{(m_q,k_q)}\mathbf{D}^q_{(m_q,1)}\end{aligned} \quad (3.25)$$

where

$$\mathbf{D}^1_{(m_1,k_1)} = \begin{bmatrix} W_N^{m_1k_1} & 0 \\ 0 & W_N^{(\frac{L}{2}+m_1)k_1}\end{bmatrix} \otimes \mathbf{I}_2$$

and

$$\mathbf{D}^2_{(m_2,k_2)} = \mathbf{I}_2 \otimes \begin{bmatrix} W_N^{m_2k_2} & 0 \\ 0 & W_N^{(\frac{L}{2}+m_2)k_2}\end{bmatrix}.$$

Hence, by performing an appropriate factorization, (3.22) can be efficiently implemented

by expressing it as

$$\begin{aligned}\widetilde{\mathbf{X}}^{ou}_{(m_1,0)(m_2,0)d} &= \overline{\mathbf{X}}^{ou}_{m_1m_2,0}+\mathbf{D}^1_{(m_1,1)}(\overline{\mathbf{X}}^{ou}_{m_1m_2,1} \\ &\quad +\mathbf{D}^1_{(m_1,1)}(\overline{\mathbf{X}}^{ou}_{m_1m_2,2}+.....+\mathbf{D}^1_{(m_1,1)}(\overline{\mathbf{X}}^{ou}_{m_1m_2,\frac{N}{4L}-2} \\ &\quad +\mathbf{D}^1_{(m_1,1)}(\overline{\mathbf{X}}^{ou}_{m_1m_2,\frac{N}{4L}-1}))...)) \end{aligned} \tag{3.26}$$

where

$$\begin{aligned}\overline{\mathbf{X}}^{ou}_{m_1m_2,k_1} &= \widehat{\mathbf{X}}^{in}_{(m_1,k_1)(m_2,0)}+\mathbf{D}^2_{(m_2,1)}(\widehat{\mathbf{X}}^{in}_{(m_1,k_1)(m_2,1)} \\ &\quad +\mathbf{D}^2_{(m_2,1)}(\widehat{\mathbf{X}}^{in}_{(m_1,k_1)(m_2,2)}+...+\mathbf{D}^2_{(m_2,1)}(\widehat{\mathbf{X}}^{in}_{(m_1,k_1)(m_2,\frac{N}{4L}-2)} \\ &\quad +\mathbf{D}^2_{(m_2,1)}(\widehat{\mathbf{X}}^{in}_{(m_1,k_1)(m_2,\frac{N}{4L}-1)}))...)) \end{aligned} \tag{3.27}$$

$m_1, m_2 = 0, 1, ..., (L/2)-1$ and $k_1 = 0, 1, ..., (N/4L)-1$.

We now consider the performance of the proposed pruning algorithm for the 2-D DFT computation of a subset of output samples, when it is implemented using (3.20) and (3.26). This is done by comparing its computational complexity with that of the existing 2-D pruning algorithm [107], [108], since both these are based on the radix-(2×2) DIT FFT algorithm. Note that when the desired output has a square support region ($L \times L$), the algorithm in [109] reduces to the existing algorithm [107], [108].

The proposed algorithm reduces the number of stages from $r = \log_2 N$ to $d = \log_2 L$. The advantage of this is that the overall structural complexity of the algorithm is significantly decreased in addition to the reduction of the data transfers. This is achieved by dividing the pruning operation into two distinct parts. The first part is concerned with the first $(d-1)$ stages obtained by varying i in (3.20) from 1 to $(d-1)$. Each of these stages is performed by repeating $(N^2/4)$ times the butterfly based on (3.20). Each butterfly operation requires 3 complex multiplications, 8 complex additions, 4 RAM read and 4 RAM write operations

of complex numbers. It is seen from (3.20) that the indexing process introduced in the above development not only simplifies the address generation, but has two other advantages. First, it allows the processing of all the butterflies of the ith stage having the same twiddle factors, and this can be easily done by fixing m_1 and m_2, and varying k_1 and k_2 from 0 to $(2^{r-i}-1)$, $i=1,2,...,d-1$. This technique provides a significant reduction in the number of address generations and twiddle factor evaluations or accesses to the lookup table. The second advantage is that, by this indexing process, it is easy to avoid trivial multiplications in (3.20) by starting each stage with m_1 or $m_2=0$ and varying k_1 and k_2. The first and second stages (corresponding to $i=1,2$ in (3.20)) do not require multiplications any more. Additional savings are possible if the implementation is designed to take advantage of the twiddle factors at the third stage.

The second part of the proposed algorithm is concerned with the dth stage corresponding to (3.26). The purpose of the recursive computation technique introduced in (3.26) is to provide further savings in the data transfers, address generations, arithmetic operations, and twiddle factor evaluations or accesses to the lookup table. From (3.26), it can be seen that the second part of the proposed algorithm requires $L^2[(N/4L)-1]$ complex multiplications, $L^2[(N/4L)-1]$ complex additions, L^2 RAM write operations of complex numbers and to perform $(LN/16)$ times the operation given by (3.27). The expression given by (3.27) requires $4[(N/4L)-1]$ complex multiplications, $4[(N/4L)-1]$ complex additions and to perform $(N/4L)$ times the operation given by (3.23). Finally, (3.23) requires 63 complex multiplications, 6 multiplications by $(1-j)/\sqrt{2}$, 188 complex additions and 64 RAM read operations of complex numbers. It is shown that the number of complex twiddle factor evaluations or accesses to the lookup table needed by (3.26) is $(65L^2/4)+L$. In counting this number and that of the data transfers, it is assumed that sufficient number of internal fast registers are available in the processor to store the intermediate results and to keep the twiddle factors for the succeeding operations.

The overall computational complexity including the trivial multiplications (multiplications by ± 1) of the proposed algorithm as well as that of the existing output-pruned radix-(2×2) DIT FFT algorithm [107], [108] is summarized in Table 3.2. It is assumed that the 4mult-2add scheme is considered. It is clear from this table that the proposed algorithm is far better than the algorithm of [107], [108], since all the operations that contribute significantly to the execution time are reduced.

3.4 Summary

In this chapter, we have proposed an efficient method, applicable for both DIT and DIF decompositions, for pruning the output samples in a radix-2 or radix-(2×2) FFT algorithm. It has been achieved by grouping in a radix-2 or radix-(2×2) FFT algorithm the last $(r-d+1)$ stages that involve unnecessary operations into a single stage and introducing a new recursive technique for computing the resulting dth stage. Due to this grouping and the efficient indexing process introduced in this chapter, the implementation of the proposed pruning algorithms requires only d stages, whereas the existing pruning algorithms requires r $(r > d)$ stages. Therefore, the overall structural complexities of the algorithms are substantially reduced by using the proposed method. In addition, it has been shown that significant reduction in the computational complexity can be achieved using the proposed method, yet retaining all the features of the Cooley-Tukey FFT algorithms.

Chapter 4

Higher Radix Approach for the Computation of DFTs

4.1 Introduction

The previous two chapters dealt with the exploitation of the radix-2 approach in developing new attractive algorithms and methods that provide significantly reduced computational and structural complexities compared to those of the existing ones. The idea of the radix-2 approach was originally introduced by Cooley and Tukey in 1965 [3] for computing a length-N DFT, where N is an integral power of two. Later, it has been used to develop more efficient algorithms, namely radix-4 [12], [95], radix-8 [4], [92], [112] and radix-16 [32], [84] FFT algorithms. The number of stages required by the radix-16 FFT algorithm is three-quarters, one-half and one-quarter of that required respectively by the radix-8, radix-4 and radix-2 FFT algorithms. If the algorithms are implemented on processors that possess sufficient internal registers to perform an entire butterfly without extra transfers, a higher radix FFT algorithm reduces the number of all the operations such as arithmetic operations, data transfers, address generations, and twiddle factor evaluations or accesses to the lookup

table compared to the corresponding numbers required by lower radix FFT algorithms.

In this chapter, an efficient technique for further improving these higher radix FFT algorithms is proposed [113], [114]. It is achieved by introducing new indices for some of the sub-sequences resulting from the conventional decompositions in these algorithms. This is realized without any increase in the computational or structural complexities of the algorithms. Sections 4.2, 4.3 and 4.4 consider respectively the cases of radix-4, radix-8 and radix-16 FFT algorithms.

4.2 An Improved Radix-4 DIF FFT Algorithm

If N is assumed to be an integral power of four, the DIF decomposition of the length-N DFT given by (3.1) can provide in the first stage four length-$N/4$ DFTs given by

$$\widetilde{X}(4n) = \sum_{k=0}^{N/4-1} \widetilde{g}_0(k) W_{N/4}^{nk},\ n = 0, 1, ..., (N/4-1) \tag{4.1}$$

$$\widetilde{X}(4n+1) = \sum_{k=0}^{N/4-1} \widetilde{g}_1(k) W_{N/4}^{nk},\ n = 0, 1, ..., (N/4-1) \tag{4.2}$$

$$\widetilde{X}(4n+2) = \sum_{k=0}^{N/4-1} \widetilde{g}_2(k) W_{N/4}^{nk},\ n = 0, 1, ..., (N/4-1) \tag{4.3}$$

$$\widetilde{X}((N+4n-1)\mathrm{mod}N) = \sum_{k=0}^{N/4-1} \widetilde{g}_{-1}(k) W_{N/4}^{nk},\ n = 0, 1, ..., (N/4-1) \tag{4.4}$$

where the input sequences $\widetilde{g}_0(k)$, $\widetilde{g}_1(k)$, $\widetilde{g}_2(k)$ and $\widetilde{g}_{-1}(k)$ are expressed in a matrix form as

$$\begin{bmatrix} \widetilde{g}_0(k) \\ \widetilde{g}_1(k) \\ \widetilde{g}_2(k) \\ \widetilde{g}_{-1}(k) \end{bmatrix} = \mathbf{F}_{4,k}\mathbf{Z}_4\mathbf{D}_4\mathbf{R}_4 \begin{bmatrix} \widetilde{x}(k) \\ \widetilde{x}(k+N/4) \\ \widetilde{x}(k+N/2) \\ \widetilde{x}(k+3N/4) \end{bmatrix},\ k = 0, 1, ..., (N/4-1) \tag{4.5}$$

In (4.5), the twiddle factor matrix $\mathbf{F}_{4,k}$ is given by

$$\mathbf{F}_{4,k} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & W_N^k & 0 & 0 \\ 0 & 0 & W_N^{2k} & 0 \\ 0 & 0 & 0 & W_N^{-k} \end{bmatrix} \tag{4.6}$$

and the constant matrices $\mathbf{Z}_4$, $\mathbf{D}_4$ and $\mathbf{R}_4$ by

$$\mathbf{Z}_4 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \tag{4.7}$$

$$\mathbf{D}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -j \end{bmatrix} \tag{4.8}$$

$$\mathbf{R}_4 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \tag{4.9}$$

This decomposition scheme is identical to that of the conventional radix-4 DIF FFT algorithm, except for a slight difference, yet with a significant implication. This difference is in the indexing of the output sub-sequence given by (4.4) and leads to the twiddle factor matrix given by (4.6). The radix-4 DIF FFT algorithm resulting from this new decomposition will, henceforth, be referred as the improved radix-4 DIF FFT algorithm. It is

seen from (4.6) that only 4 real twiddle factors (cosine and sine) need to be evaluated or loaded from the lookup table during the processing of the improved algorithm, since $W_N^k = \cos\left(\frac{2\pi}{N}k\right) - j\sin\left(\frac{2\pi}{N}k\right)$ and $W_N^{-k} = \cos\left(\frac{2\pi}{N}k\right) + j\sin\left(\frac{2\pi}{N}k\right)$. However, the general butterfly of the conventional radix-4 DIF FFT algorithm requires 6 real twiddle factors. Consequently, a saving of 33% is achieved by the improved algorithm in terms of the twiddle factor evaluations or accesses to the lookup table. Note that, when the lookup table is used, similar savings are obtained in the address generation. This improvement is achieved without any increase in the computational and structural complexities of the algorithm. The generation of the new address $(N+4n-1)\text{mod}N$ introduced in (4.4) does not require any additional complexity as compared to its corresponding address $(4n+3)$ of the conventional radix-4 DIF FFT algorithm, since

$$(N+4n-1)\text{mod}N = \begin{cases} N-1, \text{ for } n=0 \\ 4n-1 \text{ elsewhere} \end{cases} \tag{4.10}$$

4.3 An Improved Radix-8 DIF FFT Algorithm

If the length N is assumed to be an integral power of eight, the length-N DFT given by (3.1) can be decomposed in the first stage, using the DIF scheme, into eight length-$N/8$ DFTs given by

$$\widetilde{X}(8n) = \sum_{k=0}^{N/8-1} \widetilde{h}_0(k) W_{N/8}^{nk}, \; n = 0, 1, ..., (N/8-1) \tag{4.11}$$

$$\widetilde{X}(8n+1) = \sum_{k=0}^{N/8-1} \widetilde{h}_1(k) W_{N/8}^{nk}, \; n = 0, 1, ..., (N/8-1) \tag{4.12}$$

$$\widetilde{X}(8n+2) = \sum_{k=0}^{N/8-1} \widetilde{h}_2(k) W_{N/8}^{nk}, \; n = 0, 1, ..., (N/8-1) \tag{4.13}$$

$$\widetilde{X}(8n+3) = \sum_{k=0}^{N/8-1} \widetilde{h}_3(k) W_{N/8}^{nk}, \; n = 0, 1, ..., (N/8-1) \tag{4.14}$$

$$\widetilde{X}(8n+4) = \sum_{k=0}^{N/8-1} \widetilde{h}_4(k) W_{N/8}^{nk},\ n = 0, 1, ..., (N/8-1) \tag{4.15}$$

$$\widetilde{X}((N+8n-3)\mathrm{mod}N) = \sum_{k=0}^{N/8-1} \widetilde{h}_{-3}(k) W_{N/8}^{nk},\ n = 0, 1, ..., (N/8-1) \tag{4.16}$$

$$\widetilde{X}((N+8n-2)\mathrm{mod}N) = \sum_{k=0}^{N/8-1} \widetilde{h}_{-2}(k) W_{N/8}^{nk},\ n = 0, 1, ..., (N/8-1) \tag{4.17}$$

$$\widetilde{X}((N+8n-3)\mathrm{mod}N) = \sum_{k=0}^{N/8-1} \widetilde{h}_{-1}(k) W_{N/8}^{nk},\ n = 0, 1, ..., (N/8-1) \tag{4.18}$$

where the input sequences in (4.11)-(4.18) are expressed in a matrix form as

$$\begin{bmatrix} \widetilde{h}_0(k) \\ \widetilde{h}_1(k) \\ \widetilde{h}_2(k) \\ \widetilde{h}_3(k) \\ \widetilde{h}_4(k) \\ \widetilde{h}_{-3}(k) \\ \widetilde{h}_{-2}(k) \\ \widetilde{h}_{-1}(k) \end{bmatrix} = \mathbf{F}_{8,k}\mathbf{Z}_8\mathbf{D}_8\mathbf{R}_8\mathbf{T}_8\mathbf{P}_8 \begin{bmatrix} \widetilde{x}(k) \\ \widetilde{x}(k+N/8) \\ \widetilde{x}(k+N/4) \\ \widetilde{x}(k+3N/8) \\ \widetilde{x}(k+N/2) \\ \widetilde{x}(k+5N/8) \\ \widetilde{x}(k+3N/4) \\ \widetilde{x}(k+7N/8) \end{bmatrix},\ k = 0, 1, ..., (N/8-1) \tag{4.19}$$

The twiddle factor matrix $\mathbf{F}_{8,k}$ in (4.19) is given by

$$\mathbf{F}_{8,k} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & q & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & q^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & q^3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & q^4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & q^{-3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & q^{-2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & q^{-1} \end{bmatrix} \tag{4.20}$$

and the constant matrices $\mathbf{Z}_8$, $\mathbf{D}_8$, $\mathbf{R}_8$, $\mathbf{T}_8$ and $\mathbf{P}_8$ by

$$\mathbf{Z}_8 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix} \tag{4.21}$$

$$\mathbf{D}_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -j & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -j \end{bmatrix} \tag{4.22}$$

$$\mathbf{R}_8 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix} \tag{4.23}$$

$$\mathbf{T}_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \alpha & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -j & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta \end{bmatrix} \tag{4.24}$$

$$\mathbf{P}_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{4.25}$$

where $q = W_N^k$, $\alpha = \frac{\sqrt{2}}{2}(1-j)$ and $\beta = -\frac{\sqrt{2}}{2}(1+j)$.

This decomposition scheme is identical to that of the conventional radix-8 DIF FFT algorithm, except for a slight difference, yet with a significant implication. This difference is in the indexing of the output sub-sequences given by (4.16)-(4.18) and leads to the twiddle factor matrix given by (4.20). The radix-8 DIF FFT algorithm resulting from this new decomposition will, henceforth, be referred as the improved radix-8 DIF FFT algorithm. It is seen from (4.20) that only 8 real twiddle factors need to be evaluated or loaded from the lookup table during the processing of the improved algorithm, since $W_N^{pk} = \cos\left(\frac{2\pi}{N}pk\right) - j\sin\left(\frac{2\pi}{N}pk\right)$ and $W_N^{-pk} = \cos\left(\frac{2\pi}{N}pk\right) + j\sin\left(\frac{2\pi}{N}pk\right)$ for $p = 1, 2, 3$. However, since the elements of the corresponding diagonal twiddle factor matrix of the conventional radix-8 DIF FFT algorithm are W_N^{qk}, for $q = 1, 2, 3, 4, 5, 6, 7$, the general butterfly of the conventional radix-8 DIF FFT algorithm requires 14 real twiddle factors. Consequently, a saving of 42% is achieved by the improved algorithm in terms of the twiddle factor evaluations or accesses to the lookup table. Note that, when the lookup table is used, similar savings are obtained in the address generation. This improvement is achieved without any increase in the computational and structural complexities of the algorithm. The generation of the new indices $(N+8n-1)\text{mod}N$, $(N+8n-2)\text{mod}N$ and $(N+8n-1)\text{mod}N$

introduced, respectively, in (4.16)-(4.18) do not require any additional complexity as compared to the corresponding indices $(8n+5)$, $(8n+6)$ and $(8n+7)$ of the conventional radix-8 DIF FFT algorithm, since

$$(N+8n-p)\text{mod}N = \begin{cases} N-p, \text{ for } n=0 \\ 8n-p \text{ elsewhere} \end{cases}, p=1,2,3 \tag{4.26}$$

4.4 An Improved Radix-16 DIF FFT Algorithm

Let us first give a brief review of the conventional radix-16 DIF FFT algorithm. By assuming that N is an integral power of 16 and changing the variables n and k in (3.1) by

$$16n+p,\ 0 \le n \le \frac{N}{16}-1,\ 0 \le p \le 15 \tag{4.27}$$

and

$$k+q\frac{N}{16},\ 0 \le k \le \frac{N}{16}-1,\ 0 \le q \le 15, \tag{4.28}$$

respectively, the DFT given by (3.1) can be expressed as

$$\widetilde{X}(16n+p) = \sum_{k=0}^{N/16-1} \widetilde{y}_p(k) W_N^{pk} W_{N/16}^{nk},\ 0 \le n \le \frac{N}{16}-1 \tag{4.29}$$

where

$$\widetilde{y}_p(k) = \sum_{q=0}^{15} \widetilde{x}(k+q\frac{N}{16}) W_{16}^{pq},\ 0 \le p \le 15 \tag{4.30}$$

For a given value of p, (4.29) is recognized as a length-$\frac{N}{16}$ DFT, whose input sequence is obtained from (4.30) after multiplying it by the twiddle factors. For a given value of k, (4.30) is recognized as a 16-point DFT that can be expressed in a matrix form as

$$\widetilde{\mathbf{Y}}_k = \mathbf{W}_{16}\widetilde{\mathbf{X}}_k \tag{4.31}$$

where $\mathbf{W}_{16}$ is the operator matrix of the DFT of length 16. The components of the input and output vectors of (4.31) are related to the input and output sequences of (4.30) by

$$\widetilde{X}_k(p) = \widetilde{x}(k + q\frac{N}{16})$$

and

$$\widetilde{Y}_k(p) = \widetilde{y}_p(k),$$

respectively. From (4.29) and (4.31), the butterfly of the conventional radix-16 DIF FFT algorithm is given by

$$\widehat{\mathbf{Y}}_k = \mathbf{F}_{16,k}^{\text{conv}} \mathbf{W}_{16} \widetilde{\mathbf{X}}_k \tag{4.32}$$

where

$$\begin{aligned} \mathbf{F}_{16,k}^{\text{conv}} \;=\; & diag\Big(1, W_{16}^{k}, W_{16}^{2k}, W_{16}^{3k}, W_{16}^{4k}, W_{16}^{5k}, W_{16}^{6k}, W_{16}^{7k}, \\ & W_{16}^{8k}, W_{16}^{9k}, W_{16}^{10k}, W_{16}^{11k}, W_{16}^{12k}, W_{16}^{13k}, W_{16}^{14k}, W_{16}^{15k}\Big) \end{aligned} \tag{4.33}$$

and the components of the vector $\widehat{\mathbf{Y}}_k$ are related to the input sequences of (4.29) by

$$\widehat{Y}_k(p) = \widetilde{y}_p(k) W_N^{pk}$$

Thus, the radix-16 DIF FFT algorithm decomposes the length-N DFT into 16 length-$\frac{N}{16}$ DFTs given by (4.29) in the first stage. This is achieved by repeating $\frac{N}{16}$ times the butterfly given by (4.32). This decomposition process is applied successively for each of the new resulting DFTs until the size is reduced to 16-point DFTs. The entire algorithm requires $\log_{16} N$ stages.

In order to further decrease the number of twiddle factor evaluations or accesses to the lookup table in the conventional radix-16 DIF FFT algorithm, we introduce a slight

modification in the indexing process of some of the DFTs given by (4.29). This leads to a new algorithm that provides in its first stage 16 length-$\frac{N}{16}$ DFTs given by

$$\widetilde{X}(16n) = \sum_{k=0}^{N/16-1} \widetilde{y}_0(k) W_{N/16}^{nk}, \; 0 \le n \le \frac{N}{16} - 1 \tag{4.34}$$

$$\widetilde{X}(16n+1) = \sum_{k=0}^{N/16-1} \widetilde{y}_1(k) W_N^k W_{N/16}^{nk}, \; 0 \le n \le \frac{N}{16} - 1 \tag{4.35}$$

$$\widetilde{X}(16n+2) = \sum_{k=0}^{N/16-1} \widetilde{y}_2(k) W_N^{2k} W_{N/16}^{nk}, \; 0 \le n \le \frac{N}{16} - 1 \tag{4.36}$$

$$\widetilde{X}(16n+3) = \sum_{k=0}^{N/16-1} \widetilde{y}_3(k) W_N^{3k} W_{N/16}^{nk}, \; 0 \le n \le \frac{N}{16} - 1 \tag{4.37}$$

$$\widetilde{X}(16n+4) = \sum_{k=0}^{N/16-1} \widetilde{y}_4(k) W_N^{4k} W_{N/16}^{nk}, \; 0 \le n \le \frac{N}{16} - 1 \tag{4.38}$$

$$\widetilde{X}(16n+5) = \sum_{k=0}^{N/16-1} \widetilde{y}_5(k) W_N^{5k} W_{N/16}^{nk}, \; 0 \le n \le \frac{N}{16} - 1 \tag{4.39}$$

$$\widetilde{X}(16n+6) = \sum_{k=0}^{N/16-1} \widetilde{y}_6(k) W_N^{6k} W_{N/16}^{nk}, \; 0 \le n \le \frac{N}{16} - 1 \tag{4.40}$$

$$\widetilde{X}(16n+7) = \sum_{k=0}^{N/16-1} \widetilde{y}_7(k) W_N^{7k} W_{N/16}^{nk}, \; 0 \le n \le \frac{N}{16} - 1 \tag{4.41}$$

$$\widetilde{X}(16n+8) = \sum_{k=0}^{N/16-1} \widetilde{y}_8(k) W_N^{8k} W_{N/16}^{nk}, \; 0 \le n \le \frac{N}{16} - 1 \tag{4.42}$$

$$\widetilde{X}((N+16n-7)\mathrm{mod}N) = \sum_{k=0}^{N/16-1} \widetilde{y}_9(k) W_N^{-7k} W_{N/16}^{nk}, \; 0 \le n \le \frac{N}{16} - 1 \tag{4.43}$$

$$\widetilde{X}((N+16n-6)\mathrm{mod}N) = \sum_{k=0}^{N/16-1} \widetilde{y}_{10}(k) W_N^{-6k} W_{N/16}^{nk}, \; 0 \le n \le \frac{N}{16} - 1 \tag{4.44}$$

$$\widetilde{X}((N+16n-5)\mathrm{mod}N) = \sum_{k=0}^{N/16-1} \widetilde{y}_{11}(k) W_N^{-5k} W_{N/16}^{nk}, \; 0 \le n \le \frac{N}{16} - 1 \tag{4.45}$$

$$\widetilde{X}((N+16n-4)\mathrm{mod}N) = \sum_{k=0}^{N/16-1} \widetilde{y}_{12}(k) W_N^{-4k} W_{N/16}^{nk},\ 0 \le n \le \frac{N}{16}-1 \tag{4.46}$$

$$\widetilde{X}((N+16n-3)\mathrm{mod}N) = \sum_{k=0}^{N/16-1} \widetilde{y}_{13}(k) W_N^{-3k} W_{N/16}^{nk},\ 0 \le n \le \frac{N}{16}-1 \tag{4.47}$$

$$\widetilde{X}((N+16n-2)\mathrm{mod}N) = \sum_{k=0}^{N/16-1} \widetilde{y}_{14}(k) W_N^{-2k} W_{N/16}^{nk},\ 0 \le n \le \frac{N}{16}-1 \tag{4.48}$$

$$\widetilde{X}((N+16n-1)\mathrm{mod}N) = \sum_{k=0}^{N/16-1} \widetilde{y}_{15}(k) W_N^{-k} W_{N/16}^{nk},\ 0 \le n \le \frac{N}{16}-1 \tag{4.49}$$

The radix-16 DIF FFT algorithm resulting from this modification in the indexing process will be referred as the improved radix-16 DIF FFT algorithm. The decomposition scheme of this improved algorithm is identical to that of the conventional radix-16 DIF FFT algorithm.

Similar to the conventional radix-16 DIF FFT algorithm, the butterfly of the improved radix-16 DIF FFT algorithm can be constructed from (4.34)-(4.49) as

$$\widehat{\mathbf{Y}}_k = \mathbf{F}_{16,k}^{\mathrm{imp}} \mathbf{W}_{16} \widetilde{\mathbf{X}}_k \tag{4.50}$$

where

$$\begin{aligned} \mathbf{F}_{16,k}^{\mathrm{imp}} \;=\; & diag\Big(1, W_{16}^{k}, W_{16}^{2k}, W_{16}^{3k}, W_{16}^{4k}, W_{16}^{5k}, W_{16}^{6k}, W_{16}^{7k}, \\ & W_{16}^{8k}, W_{16}^{-7k}, W_{16}^{-6k}, W_{16}^{-5k}, W_{16}^{-4k}, W_{16}^{-3k}, W_{16}^{-2k}, W_{16}^{-k}\Big) \end{aligned} \tag{4.51}$$

It is clear that the DFTs given by (4.34)-(4.42) in the improved radix-16 DIF FFT algorithm are identical to the corresponding DFTs in the conventional radix-16 DIF FFT algorithm. However, the remaining DFTs given by (4.43)-(4.49) slightly differ from the corresponding DFTs given by (4.29). The modification of the indexing process introduced in (4.43)-(4.49) leads to the twiddle factor matrix given by (4.51) of the butterfly of the improved radix-16 DIF FFT algorithm, which is different from that given by (4.33) of the

butterfly of the conventional radix-16 DIF FFT algorithm. This slight difference has a significant implication. It is seen from (4.51) that, since $W_N^{pk} = \cos\left(\frac{2\pi}{N}pk\right) - j\sin\left(\frac{2\pi}{N}pk\right)$ and $W_N^{-pk} = \cos\left(\frac{2\pi}{N}pk\right) + j\sin\left(\frac{2\pi}{N}pk\right)$ for $p = 1, 2, 3, 4, 5, 6, 7$, only 16 real twiddle factors need to be evaluated or loaded from the lookup table during the processing of the butterfly given by (4.50) of the improved FFT algorithm. However, since the elements of the matrix given by (4.33) are W_N^{qk}, for $q = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$, the butterfly given by (4.32) of the conventional FFT algorithm requires 30 real twiddle factors. Consequently, savings of more than 46% are achieved by the improved algorithm in terms of the twiddle factor evaluations or accesses to the lookup table. Note that when the lookup table is used, similar savings are obtained in the address generation. This improvement is achieved without imposing any additional computational or structural complexity in the algorithm.

The generation of the new indices $(N + 16n - p)\text{mod}N$, for $p = 1, 2, 3, 4, 5, 6, 7$, introduced in (4.43)-(4.49), of the improved radix-16 DIF FFT algorithm does not impose any additional complexity as compared to the corresponding indices $(16n + q)$, for $q = 9, 10, 11, 12, 13, 14, 15$, of the conventional radix-16 DIF FFT algorithm, since

$$(N + 16n - p)\text{mod}N = \begin{cases} N - p, \text{ for } n = 0 \\ 16n - p \text{ elsewhere} \end{cases}, \quad p = 1, 2, 3, 4, 5, 6, 7 \tag{4.52}$$

4.5 Summary

In this chapter, improved algorithms for radix-4, radix-8 and radix-16 FFTs have been proposed by introducing new indices for some of the sub-sequences resulting from the conventional decompositions in the radix-4, radix-8 and radix-16 FFT algorithms. It has been shown that savings of more than 33%, 42% and 46% in the number of twiddle factor evaluations or accesses to the lookup table can be achieved using the proposed radix-4, radix-8 and radix-16 algorithms over the corresponding conventional FFT algorithms without imposing any additional computational or structural complexity. In addition, it should be mentioned that the hardware implementation of the architectures [115] employing the proposed FFT algorithms can result in a reduced number of ROM modules.

The technique introduced in this chapter for improving the higher radix approach will also be used in Chapter 6.

Chapter 5

Radix-2/4 Approach for the Computation of 3-D DFT and DHT

5.1 Introduction

The radix-2 approach was used in Chapter 2 to reduce the complexity in the computation of the DFT and DHT of arbitrary dimensions. As discussed in Chapter 4, the radix-4 approach provides a lower computational complexity compared to that provided by the radix-2 approach. It has been shown in the literature that the simultaneous use of these two approaches leads to 1-D [11], [17], [45], [56], [57] and 2-D [29], [30], [64] FFT and FHT algorithms having a good compromise between the arithmetic and structural complexities. The numbers of data transfers, address generations, and twiddle factor evaluations or accesses to the lookup table required by these algorithms are lower than those required by the corresponding algorithms that are based on the radix-2 approach. However, in the case of three or higher dimensions, the simultaneous use of the radix-2 and radix-4 approaches that may provide efficient solutions has not yet been studied. This is mainly due to the fact that the simultaneous use of these two approaches in the case of higher dimensions is neither

obvious nor straightforward.

In this chapter, the problem of reducing the complexity in the computation of the 3-D DFT and DHT is addressed and a method proposed for the simultaneous use of the radix-2 and radix-4 approaches in the 3-D case. This method consists of two steps. In the first step, the general decomposition method proposed in Chapter 2 is used to decompose the 3-D DFT or DHT using the radix-$(2 \times 2 \times 2)$ index maps. In the second step, a further decomposition is carried out to introduce radix-$(4 \times 4 \times 4)$ index maps. The introduction of these two steps and an appropriate use of the Kronecker product enable us to design 3-D FFT [116], [117] and FHT [118], [119] algorithms whose butterflies are characterized by simple closed-form expressions. The proposed solutions for the 3-D DFT and DHT are given in Sections 5.2 and 5.3, respectively.

5.2 Radix-2/4 Approach for the Computation of 3-D DFT

The conventional method of reducing the computational complexity in the computation of the 3-D DFT is by applying sequentially a 1-D FFT algorithm to each of the three dimensions of the input sequence. This is called the row-column method [26], [35]. Another popular algorithm is the 3-D radix-$(2 \times 2 \times 2)$ FFT algorithm [24], [26], [32], which is based on a philosophy similar to that of the 1-D radix-2 FFT algorithm [3]. The radix-$(2 \times 2 \times 2)$ FFT substantially reduces the number of arithmetic operations over the row-column method. Although the polynomial transform algorithm [25], [34] requires the lowest number of multiplications, it is not widely used in view of its complicated structure compared to the existing radix-based FFT algorithms. Duhamel and Hollmann have shown that in the 1-D case the radix-2/4 approach provides the best compromise between the arithmetic and structural complexities [11]. This approach has been extended to the 2-D DFT case using DIT [39]. The DIF version of the algorithm in [39] has been formulated in

[28] using a matrix form. Alternate versions of these 2-D FFT algorithms have been given in [29] using an algebraic formulation. However, no algorithm exists for three or higher dimensions using the radix-2/4 approach.

In this section, we propose a new radix-$(2\times 2\times 2)/(4\times 4\times 4)$ DIF FFT algorithm for computing the 3-D DFT, based on a mixture of radix-$(2\times 2\times 2)$ and radix-$(4\times 4\times 4)$ index maps, coupled with an appropriate use of the Kronecker product.

5.2.1 Proposed Radix-$(2\times 2\times 2)/(4\times 4\times 4)$ DIF FFT Algorithm

Using (2.51), the 3-D DFT of size $(N\times N\times N)$ is given by

$$\widetilde{X}(n_1,n_2,n_3)=\sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1}\sum_{k_3=0}^{N-1}\widetilde{x}(k_1,k_2,k_3)W_N^{\sum_{i=1}^{3}n_ik_i},$$
$$0\le n_i\le N-1,\ i=1,2,3 \tag{5.1}$$

In this section, we introduce a new DIF decomposition of the 3-D DFT given by (5.1) using the radix-2/4 approach. This decomposition is performed using two steps. In the first step, we use radix-$(2\times 2\times 2)$ index maps to decompose the DFT given by (5.1) into eight DFTs. In the second step, we introduce radix-$(4\times 4\times 4)$ index maps by further decomposing some of these resulting DFTs.

Let us first introduce radix-$(2\times 2\times 2)$ index maps by changing the variables k_i and n_i in (5.1) by

$$k_i+\frac{N}{2}l_i,\ 0\le k_i\le\frac{N}{2}-1,\ l_i=0,1 \tag{5.2}$$

and

$$2n_i+m_i,\ 0\le n_i\le\frac{N}{2}-1,\ m_i=0,1 \tag{5.3}$$

respectively. Then, (5.1) is decomposed into eight $\left(\frac{N}{2}\times\frac{N}{2}\times\frac{N}{2}\right)$-point DFTs given by

$$\widetilde{X}(2n_1,2n_2,2n_3)=\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1}\widetilde{y}_{000}(k_1,k_2,k_3)\,W_{N/2}^{\sum_{i=1}^{3}n_ik_i},$$
$$0\le n_i\le\frac{N}{2}-1 \tag{5.4}$$

$$\widetilde{X}(2n_1,2n_2,2n_3+1)=\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1}\widetilde{y}_{001}(k_1,k_2,k_3)\,W_N^{k_3}W_{N/2}^{\sum_{i=1}^{3}n_ik_i},$$
$$0\le n_i\le\frac{N}{2}-1 \tag{5.5}$$

$$\widetilde{X}(2n_1,2n_2+1,2n_3)=\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1}\widetilde{y}_{010}(k_1,k_2,k_3)\,W_N^{k_2}W_{N/2}^{\sum_{i=1}^{3}n_ik_i},$$
$$0\le n_i\le\frac{N}{2}-1 \tag{5.6}$$

$$\widetilde{X}(2n_1,2n_2+1,2n_3+1)=\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1}\widetilde{y}_{011}(k_1,k_2,k_3)\,W_N^{k_2+k_3}W_{N/2}^{\sum_{i=1}^{3}n_ik_i},$$
$$0\le n_i\le\frac{N}{2}-1 \tag{5.7}$$

$$\widetilde{X}(2n_1+1,2n_2,2n_3)=\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1}\widetilde{y}_{100}(k_1,k_2,k_3)\,W_N^{k_1}W_{N/2}^{\sum_{i=1}^{3}n_ik_i},$$
$$0\le n_i\le\frac{N}{2}-1 \tag{5.8}$$

$$\widetilde{X}(2n_1+1,2n_2,2n_3+1)=\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1}\widetilde{y}_{101}(k_1,k_2,k_3)\,W_N^{k_1+k_3}W_{N/2}^{\sum_{i=1}^{3}n_ik_i},$$

$$0\le n_i\le\frac{N}{2}-1 \qquad (5.9)$$

$$\widetilde{X}(2n_1+1,2n_2+1,2n_3)=\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1}\widetilde{y}_{110}(k_1,k_2,k_3)\,W_N^{k_1+k_2}W_{N/2}^{\sum_{i=1}^{3}n_ik_i},$$

$$0\le n_i\le\frac{N}{2}-1 \qquad (5.10)$$

$$\widetilde{X}(2n_1+1,2n_2+1,2n_3+1)=\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1}\widetilde{y}_{111}(k_1,k_2,k_3)\,W_N^{k_1+k_2+k_3}W_{N/2}^{\sum_{i=1}^{3}n_ik_i},$$

$$0\le n_i\le\frac{N}{2}-1 \qquad (5.11)$$

The sequences $\widetilde{y}_{...}(k_1,k_2,k_3)$ in (5.4)-(5.11) are obtained from the original input sequence $\widetilde{x}(k_1,k_2,k_3)$ using

$$\widetilde{\mathbf{y}}_{k_1,k_2,k_3}=\mathbf{H}_{2^3}\widetilde{\mathbf{x}}_{k_1,k_2,k_3},\ 0\le k_i\le\frac{N}{2}-1 \qquad (5.12)$$

where

$$\widetilde{\mathbf{x}}_{k_1,k_2,k_3}=\begin{bmatrix}\widetilde{x}(k_1,k_2,k_3)\\ \widetilde{x}(k_1,k_2,k_3+N/2)\\ \widetilde{x}(k_1,k_2+N/2,k_3)\\ \widetilde{x}(k_1,k_2+N/2,k_3+N/2)\\ \widetilde{x}(k_1+N/2,k_2,k_3)\\ \widetilde{x}(k_1+N/2,k_2,k_3+N/2)\\ \widetilde{x}(k_1+N/2,k_2+N/2,k_3)\\ \widetilde{x}(k_1+N/2,k_2+N/2,k_3+N/2)\end{bmatrix}, \qquad (5.13)$$

$$\widetilde{\mathbf{y}}_{k_1,k_2,k_3} = \begin{bmatrix} \widetilde{y}_{000}(k_1,k_2,k_3) \\ \widetilde{y}_{001}(k_1,k_2,k_3) \\ \widetilde{y}_{010}(k_1,k_2,k_3) \\ \widetilde{y}_{011}(k_1,k_2,k_3) \\ \widetilde{y}_{100}(k_1,k_2,k_3) \\ \widetilde{y}_{101}(k_1,k_2,k_3) \\ \widetilde{y}_{110}(k_1,k_2,k_3) \\ \widetilde{y}_{111}(k_1,k_2,k_3) \end{bmatrix} \tag{5.14}$$

and the Hadamard matrix $\mathbf{H}_{2^3}$ in (5.12) can be factored using (2.8). The decomposition of the DFT given by (5.1) into the DFTs given by (5.4)-(5.11) represents the first stage of the decomposition in the radix-$(2\times 2\times 2)$ DIF FFT algorithm presented in Section 2.5.1.

In order to develop a radix-$(2\times 2\times 2)/(4\times 4\times 4)$ FFT algorithm, we further decompose the DFTs given by (5.5)-(5.11) to introduce radix-$(4\times 4\times 4)$ index maps. Let us start by the decomposition of the even-even-odd indexed terms, which correspond to (5.5). First, we change the variables k_i's by

$$k_i + \frac{N}{4} l_i, \; 0 \le k_i \le \frac{N}{4} - 1, \; l_i = 0, 1 \tag{5.15}$$

Then, the DFT given by (5.5) becomes

$$\widetilde{X}(2n_1, 2n_2, 2n_3+1) = \sum_{k_1=0}^{N/4-1} \sum_{k_2=0}^{N/4-1} \sum_{k_3=0}^{N/4-1}$$
$$\left[\sum_{l_1=0}^{1} \sum_{l_2=0}^{1} \sum_{l_3=0}^{1} \widetilde{y}_{001}\left(k_1 + \frac{N}{4} l_1, k_2 + \frac{N}{4} l_2, k_3 + \frac{N}{4} l_3\right) (-j)^{l_3} (-j)^{\sum_{i=1}^{3} n_i l_i} \right] W_N^{k_3} W_{N/2}^{\sum_{i=1}^{3} n_i k_i},$$
$$0 \le n_i \le \frac{N}{2} - 1 \tag{5.16}$$

Now, we change the frequency variables n_i's in (5.16) by

$$2n_i+m_i,\ 0\le n_i\le \frac{N}{4}-1,\ m_i=0,1 \tag{5.17}$$

Then, (5.16) becomes

$$\widetilde{X}(4n_1+p,4n_2+q,4n_3+\gamma)=\sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1}\widehat{y}_{001}^{pq\gamma}(k_1,k_2,k_3)\,W_{N/4}^{\sum_{i=1}^{3}n_ik_i},$$
$$0\le n_i\le \frac{N}{4}-1,\ p,q=0,2,\ \gamma=1,3 \tag{5.18}$$

It is clear that (5.18) represents a set of eight $\left(\frac{N}{4}\times\frac{N}{4}\times\frac{N}{4}\right)$-point DFTs that are obtained by further decomposition of the even-even-odd index terms given by (5.5) using the radix-$(2\times2\times2)$ index maps given by (5.15) and (5.17). By a similar technique, the other odd-indexed terms given by (5.6)-(5.11) can be expressed, respectively, as

$$\widetilde{X}(4n_1+p,4n_2+\beta,4n_3+q)=\sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1}\widehat{y}_{010}^{p\beta q}(k_1,k_2,k_3)\,W_{N/4}^{\sum_{i=1}^{3}n_ik_i},$$
$$0\le n_i\le \frac{N}{4}-1,\ p,q=0,2,\ \beta=1,3 \tag{5.19}$$

$$\widetilde{X}(4n_1+p,4n_2+\beta,4n_3+\gamma)=\sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1}\widehat{y}_{011}^{p\beta\gamma}(k_1,k_2,k_3)\,W_{N/4}^{\sum_{i=1}^{3}n_ik_i},$$
$$0\le n_i\le \frac{N}{4}-1,\ p=0,2,\ \beta,\gamma=1,3 \tag{5.20}$$

$$\widetilde{X}(4n_1+\alpha,4n_2+p,4n_3+q)=\sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1}\widehat{y}_{100}^{\alpha pq}(k_1,k_2,k_3)\,W_{N/4}^{\sum_{i=1}^{3}n_ik_i},$$
$$0\le n_i\le \frac{N}{4}-1,\ p,q=0,2,\ \alpha=1,3 \tag{5.21}$$

$$\widetilde{X}(4n_1+\alpha,4n_2+p,4n_3+\gamma)=\sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1}\widetilde{y}_{101}^{\alpha p\gamma}(k_1,k_2,k_3)\,W_{N/4}^{\sum_{i=1}^{3}n_ik_i},$$

$$0\le n_i\le\frac{N}{4}-1,\ p=0,2,\ \alpha,\gamma=1,3 \qquad (5.22)$$

$$\widetilde{X}(4n_1+\alpha,4n_2+\beta,4n_3+p)=\sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1}\widetilde{y}_{110}^{\alpha\beta p}(k_1,k_2,k_3)\,W_{N/4}^{\sum_{i=1}^{3}n_ik_i},$$

$$0\le n_i\le\frac{N}{4}-1,\ p=0,2,\ \alpha,\beta=1,3 \qquad (5.23)$$

$$\widetilde{X}(4n_1+\alpha,4n_2+\beta,4n_3+\gamma)=\sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1}\widetilde{y}_{111}^{\alpha\beta\gamma}(k_1,k_2,k_3)\,W_{N/4}^{\sum_{i=1}^{3}n_ik_i},$$

$$0\le n_i\le\frac{N}{4}-1,\ \alpha,\beta,\gamma=1,3 \qquad (5.24)$$

The input sequences $\widetilde{y}_{\cdots}^{\cdots}(k_1,k_2,k_3)$ of the DFTs given by (5.18)-(5.24) are, respectively, the components of the vectors given by

$$\widehat{\mathbf{Y}}_{k_1,k_2,k_3}^{001}=\mathbf{F}_{k_1,k_2,k_3}^{001}\mathbf{H}_{2^3}(\mathbf{I}_2\otimes\mathbf{I}_2\otimes\mathbf{J}_2)\,\widetilde{\mathbf{Y}}_{k_1,k_2,k_3}^{001},\ 0\le k_i\le\frac{N}{4}-1 \qquad (5.25)$$

$$\widehat{\mathbf{Y}}_{k_1,k_2,k_3}^{010}=\mathbf{F}_{k_1,k_2,k_3}^{010}\mathbf{H}_{2^3}(\mathbf{I}_2\otimes\mathbf{J}_2\otimes\mathbf{I}_2)\,\widetilde{\mathbf{Y}}_{k_1,k_2,k_3}^{010},\ 0\le k_i\le\frac{N}{4}-1 \qquad (5.26)$$

$$\widehat{\mathbf{Y}}_{k_1,k_2,k_3}^{011}=\mathbf{F}_{k_1,k_2,k_3}^{011}\mathbf{H}_{2^3}(\mathbf{I}_2\otimes\mathbf{J}_2\otimes\mathbf{J}_2)\,\widetilde{\mathbf{Y}}_{k_1,k_2,k_3}^{011},\ 0\le k_i\le\frac{N}{4}-1 \qquad (5.27)$$

$$\widehat{\mathbf{Y}}_{k_1,k_2,k_3}^{100}=\mathbf{F}_{k_1,k_2,k_3}^{100}\mathbf{H}_{2^3}(\mathbf{J}_2\otimes\mathbf{I}_2\otimes\mathbf{I}_2)\,\widetilde{\mathbf{Y}}_{k_1,k_2,k_3}^{100},\ 0\le k_i\le\frac{N}{4}-1 \qquad (5.28)$$

$$\widehat{\mathbf{Y}}_{k_1,k_2,k_3}^{101}=\mathbf{F}_{k_1,k_2,k_3}^{101}\mathbf{H}_{2^3}(\mathbf{J}_2\otimes\mathbf{I}_2\otimes\mathbf{J}_2)\,\widetilde{\mathbf{Y}}_{k_1,k_2,k_3}^{101},\ 0\le k_i\le\frac{N}{4}-1 \qquad (5.29)$$

$$\widehat{\mathbf{Y}}_{k_1,k_2,k_3}^{110}=\mathbf{F}_{k_1,k_2,k_3}^{110}\mathbf{H}_{2^3}(\mathbf{J}_2\otimes\mathbf{J}_2\otimes\mathbf{I}_2)\,\widetilde{\mathbf{Y}}_{k_1,k_2,k_3}^{110},\ 0\le k_i\le\frac{N}{4}-1 \qquad (5.30)$$

$$\widehat{\mathbf{Y}}_{k_1,k_2,k_3}^{111}=\mathbf{F}_{k_1,k_2,k_3}^{111}\mathbf{H}_{2^3}(\mathbf{J}_2\otimes\mathbf{J}_2\otimes\mathbf{J}_2)\,\widetilde{\mathbf{Y}}_{k_1,k_2,k_3}^{111},\ 0\le k_i\le\frac{N}{4}-1 \qquad (5.31)$$

In (5.25)-(5.31), the matrix $\mathbf{J}_2$ is given by (3.11), while the components of the input and output vectors $\widetilde{\mathbf{Y}}^{\cdots}_{k_1,k_2,k_3}$ and $\widehat{\mathbf{Y}}^{\cdots}_{k_1,k_2,k_3}$ and those of their corresponding diagonal twiddle factor matrices $\mathbf{F}^{\cdots}_{k_1,k_2,k_3}$ are given by

$$\begin{cases} \widetilde{Y}^{001}_{k_1,k_2,k_3}(a) = \widetilde{y}_{001}\left(k_1 + p\frac{N}{8}, k_2 + q\frac{N}{8}, k_3 + (\gamma-1)\frac{N}{8}\right) \\ \widehat{Y}^{001}_{k_1,k_2,k_3}(a) = \widetilde{y}^{pq\gamma}_{001}(k_1,k_2,k_3) \\ F^{001}_{k_1,k_2,k_3}(a,a) = W_N^{pk_1+qk_2+\gamma k_3} \\ \text{where } a = 2p+q+\frac{\gamma-1}{2},\ p,q=0,2,\ \gamma=1,3 \end{cases} \tag{5.32}$$

$$\begin{cases} \widetilde{Y}^{010}_{k_1,k_2,k_3}(b) = \widetilde{y}_{010}\left(k_1 + p\frac{N}{8}, k_2 + (\beta-1)\frac{N}{8}, k_3 + q\frac{N}{8}\right) \\ \widehat{Y}^{010}_{k_1,k_2,k_3}(b) = \widetilde{y}^{p\beta q}_{010}(k_1,k_2,k_3) \\ F^{010}_{k_1,k_2,k_3}(b,b) = W_N^{pk_1+\beta k_2+qk_3} \\ \text{where } b = 2p+\beta-1+\frac{q}{2},\ p,q=0,2,\ \beta=1,3 \end{cases} \tag{5.33}$$

$$\begin{cases} \widetilde{Y}^{011}_{k_1,k_2,k_3}(c) = \widetilde{y}_{011}\left(k_1 + p\frac{N}{8}, k_2 + (\beta-1)\frac{N}{8}, k_3 + (\gamma-1)\frac{N}{8}\right) \\ \widehat{Y}^{011}_{k_1,k_2,k_3}(c) = \widetilde{y}^{p\beta\gamma}_{011}(k_1,k_2,k_3) \\ F^{011}_{k_1,k_2,k_3}(c,c) = W_N^{pk_1+\beta k_2+\gamma k_3} \\ \text{where } c = 2p+\beta-1+\frac{\gamma-1}{2},\ p=0,2,\ \beta,\gamma=1,3 \end{cases} \tag{5.34}$$

$$\begin{cases} \widetilde{Y}^{100}_{k_1,k_2,k_3}(d) = \widetilde{y}_{100}\left(k_1 + (\alpha-1)\frac{N}{8}, k_2 + p\frac{N}{8}, k_3 + q\frac{N}{8}\right) \\ \widehat{Y}^{100}_{k_1,k_2,k_3}(d) = \widetilde{y}^{\alpha pq}_{100}(k_1,k_2,k_3) \\ F^{100}_{k_1,k_2,k_3}(d,d) = W_N^{\alpha k_1+pk_2+qk_3} \\ \text{where } d = 2(\alpha-1)+p+\frac{q}{2},\ p,q=0,2,\ \alpha=1,3 \end{cases} \tag{5.35}$$

$$\begin{cases} \widetilde{Y}^{101}_{k_1,k_2,k_3}(e) = \widetilde{y}_{101}\left(k_1 + (\alpha-1)\frac{N}{8}, k_2 + p\frac{N}{8}, k_3 + (\gamma-1)\frac{N}{8}\right) \\ \widehat{Y}^{101}_{k_1,k_2,k_3}(e) = \widetilde{y}^{\alpha p\gamma}_{101}(k_1,k_2,k_3) \\ F^{101}_{k_1,k_2,k_3}(e,e) = W_N^{\alpha k_1+pk_2+\gamma k_3} \\ \text{where } e = 2(\alpha-1)+p+\frac{\gamma-1}{2},\ p=0,2,\ \alpha,\gamma=1,3 \end{cases} \tag{5.36}$$

$$\begin{cases} \widetilde{Y}^{110}_{k_1,k_2,k_3}(f) = \widetilde{y}_{110}\left(k_1+(\alpha-1)\frac{N}{8}, k_2+(\beta-1)\frac{N}{8}, k_3+p\frac{N}{8}\right) \\ \widehat{Y}^{110}_{k_1,k_2,k_3}(f) = \widetilde{y}^{\alpha\beta p}_{110}(k_1,k_2,k_3) \\ F^{110}_{k_1,k_2,k_3}(f,f) = W_N^{\alpha k_1+\beta k_2+pk_3} \\ \text{where } f = 2(\alpha-1)+\beta-1+\frac{p}{2},\ p=0,2,\ \alpha,\beta=1,3 \end{cases} \tag{5.37}$$

and

$$\begin{cases} \widetilde{Y}^{111}_{k_1,k_2,k_3}(g) = \widetilde{y}_{111}\left(k_1+(\alpha-1)\frac{N}{8}, k_2+(\beta-1)\frac{N}{8}, k_3+(\gamma-1)\frac{N}{8}\right) \\ \widehat{Y}^{111}_{k_1,k_2,k_3}(g) = \widetilde{y}^{\alpha\beta\gamma}_{111}(k_1,k_2,k_3) \\ F^{111}_{k_1,k_2,k_3}(g,g) = W_N^{\alpha k_1+\beta k_2+\gamma k_3} \\ \text{where } g = 2(\alpha-1)+\beta-1+\frac{\gamma-1}{2},\ \alpha,\beta,\gamma=1,3 \end{cases} \tag{5.38}$$

For a given combination of (k_1,k_2,k_3), each of the seven general sub-butterflies given by (5.25)-(5.31) computes 8 points using (5.12). In order to compute, for a given combination of (k_1,k_2,k_3), all the points required in the computation of these sub-butterflies, we rearrange (5.12) as

$$\widetilde{\mathbf{y}}_{k_1+l_1\frac{N}{4},k_2+l_2\frac{N}{4},k_3+l_3\frac{N}{4}} = \mathbf{H}_{2^3}\widetilde{\mathbf{x}}_{k_1+l_1\frac{N}{4},k_2+l_2\frac{N}{4},k_3+l_3\frac{N}{4}},$$
$$0 \le k_i \le \frac{N}{4}-1,\ l_i=0,1 \tag{5.39}$$

It is clear from (5.13) and (5.14) that for a given combination of (k_1,k_2,k_3), (5.39) computes 64 points from the initial input sequence $\widetilde{x}(k_1,k_2,k_3)$, where 8 points are used to form the input sequence $\widetilde{y}_{000}(k_1,k_2,k_3)$ of the $\left(\frac{N}{2}\times\frac{N}{2}\times\frac{N}{2}\right)$-point 3-D DFT given by (5.4) and the other 56 points are used by the general sub-butterflies given by (5.25)-(5.31). The output of these sub-butterflies are used to form the input sequences of the corresponding $\left(\frac{N}{4}\times\frac{N}{4}\times\frac{N}{4}\right)$-point 3-D DFTs given by (5.18)-(5.24). These operations are repeated for all the combinations of (k_1,k_2,k_3), $k_i=0,1,\ldots,(N/4)-1$, $i=1,2,3$. This completes the first stage of the proposed decomposition.

Thus, the proposed radix-$(2\times2\times2)/(4\times4\times4)$ DIF FFT algorithm corresponds to decomposing the $(N\times N\times N)$-point 3-D DFT given by (5.1) into one $\left(\frac{N}{2}\times\frac{N}{2}\times\frac{N}{2}\right)$-point 3-D DFT given by (5.4) and 56 $\left(\frac{N}{4}\times\frac{N}{4}\times\frac{N}{4}\right)$-point 3-D DFTs given by (5.18)-(5.24), in the first stage. This is achieved by repeating $\frac{N^3}{64}$ times the general butterfly based on (5.39) and (5.25)-(5.31). This decomposition scheme is repeated successively for each of the new resulting DFTs, until the problem is reduced to some $(2\times2\times2)$-point 3-D DFTs.

5.2.2 Computational Complexity

In this section, we consider the performance of the proposed radix-$(2\times2\times2)/(4\times4\times4)$ FFT algorithm by analyzing its computational complexity and comparing it with that of the radix-$(2\times2\times2)$ FFT algorithms presented in Chapter 2.

5.2.2.1 Arithmetic Complexity

The general butterfly of the proposed 3-D DIF FFT algorithm computes 64 points using (5.39) and (5.25)-(5.31). It requires 56 complex multiplications and 360 complex additions. There are $\frac{N^3}{64}$ general butterflies in the first stage of the decomposition. This means that the first stage requires $7\frac{N^3}{8}$ complex multiplications and $45\frac{N^3}{8}$ complex additions. The decomposition requires a special butterfly to compute the $(2\times2\times2)$-point 3-D DFTs involved in the last stage. Therefore, it is seen that the expressions for the numbers of multiplications and additions required by the two-butterfly implementation (one general and one for computing $(2\times2\times2)$-point DFTs) of the proposed FFT algorithm are, respectively,

$$\mathrm{M}^{42}_{\mathrm{r2}/4}(3,N)=3.5N^3+\mathrm{M}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right)+56\mathrm{M}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N>2,$$
$$\mathrm{M}^{42}_{\mathrm{r2}/4}(3,2)=\mathrm{M}^{42}_{\mathrm{r2}/4}(3,1)=0 \qquad (5.40)$$

and

$$\mathrm{A}^{42}_{\mathrm{r2}/4}(3,N) = 13N^3 + \mathrm{A}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right) + 56\mathrm{A}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N>2,$$
$$\mathrm{A}^{42}_{\mathrm{r2}/4}(3,2) = 48,\ \mathrm{A}^{42}_{\mathrm{r2}/4}(3,1) = 0, \tag{5.41}$$

if the 4mult-2add scheme is considered. Similarly, if the 3mult-3add scheme is considered, the corresponding expressions for the numbers of multiplications and additions are

$$\mathrm{M}^{33}_{\mathrm{r2}/4}(3,N) = 2.625N^3 + \mathrm{M}^{33}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right) + 56\mathrm{M}^{33}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N>2,$$
$$\mathrm{M}^{33}_{\mathrm{r2}/4}(3,2) = \mathrm{M}^{33}_{\mathrm{r2}/4}(3,1) = 0 \tag{5.42}$$

and

$$\mathrm{A}^{33}_{\mathrm{r2}/4}(3,N) = 13.875N^3 + \mathrm{A}^{33}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right) + 56\mathrm{A}^{33}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N>2,$$
$$\mathrm{A}^{33}_{\mathrm{r2}/4}(3,2) = 48,\ \mathrm{A}^{33}_{\mathrm{r2}/4}(3,1) = 0 \tag{5.43}$$

The expressions for the numbers of non-trivial multiplications and additions required by the proposed FFT algorithm are, respectively,

$$\mathrm{M}^{42}_{\mathrm{r2}/4}(3,N) = 3.5N^3 - 21N^2 + \mathrm{M}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right) + 56\mathrm{M}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N>4,$$
$$\mathrm{M}^{42}_{\mathrm{r2}/4}(3,4) = \mathrm{M}^{42}_{\mathrm{r2}/4}(3,2) = 0 \tag{5.44}$$

and

$$\mathrm{A}^{42}_{\mathrm{r2}/4}(3,N) = 13N^3 - 7N^2 + \mathrm{A}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right) + 56\mathrm{A}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N>4,$$
$$\mathrm{A}^{42}_{\mathrm{r2}/4}(3,4) = 768,\ \mathrm{A}^{42}_{\mathrm{r2}/4}(3,2) = 48, \tag{5.45}$$

in the case of the 4mult-2add scheme, and

$$\mathrm{M}^{33}_{\mathrm{r2}/4}(3,N) = 2.625N^3 - 14N^2 + \mathrm{M}^{33}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right) + 56\mathrm{M}^{33}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N>4,$$
$$\mathrm{M}^{33}_{\mathrm{r2}/4}(3,4) = \mathrm{M}^{33}_{\mathrm{r2}/4}(3,2) = 0 \quad (5.46)$$

and

$$\mathrm{A}^{33}_{\mathrm{r2}/4}(3,N) = 13.875N^3 - 14N^2 + \mathrm{A}^{33}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right) + 56\mathrm{A}^{33}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N>4,$$
$$\mathrm{A}^{33}_{\mathrm{r2}/4}(3,4) = 768,\ \mathrm{A}^{33}_{\mathrm{r2}/4}(3,2) = 48, \quad (5.47)$$

in the case of the 3mult-3add scheme.

To carry out a fair and comprehensive comparison, we also consider the numbers of multiplications and additions required by the one-butterfly, two-butterfly and multiple-butterfly implementations of the radix-$(2\times 2\times 2)$ FFT algorithms. The arithmetic complexities required by these algorithms along with those required by the proposed one for various transform sizes $(N\times N\times N)$ are given in Tables 5.1-5.4. We see from these tables that the proposed FFT algorithm reduces substantially the number of multiplications as well as the number of additions. For example, if the 3mult-3add scheme is considered, savings of about more than 40% in the number of multiplications and about more than 10% in the number of additions can be easily achieved using the two-butterfly implementation of the proposed FFT algorithm over the two-butterfly implementation of the radix-$(2\times 2\times 2)$ FFT algorithm.

Table 5.1: Number of arithmetic operations required for the computation of the 3-D DFT using the 4mult-2add scheme

	Radix-$(2\times2\times2)$ FFTs						Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FFT		
	Using one butterfly			Using two butterflies			Using two butterflies		
Transform size $N\times N\times N$	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point
$2^3\times2^3\times2^3$	10.5	23.25	33.75	7	21.5	28.5	3.93	19.96	23.90
$2^4\times2^4\times2^4$	14	31	45	10.5	29.25	39.75	7.05	27.52	34.58
$2^5\times2^5\times2^5$	17.5	38.75	56.25	14	37	51	7.82	33.91	41.74
$2^6\times2^6\times2^6$	21	46.5	67.5	17.5	44.75	62.25	10.65	41.32	51.97
$2^7\times2^7\times2^7$	24.5	54.25	78.75	21	52.5	73.5	11.68	47.84	59.52
$2^8\times2^8\times2^8$	28	62	90	24.5	60.25	84.75	14.27	55.13	69.41
$2^9\times2^9\times2^9$	31.5	69.75	101.25	28	68	96	15.50	61.75	77.25
$2^{10}\times2^{10}\times2^{10}$	35	77.5	112.5	31.5	75.75	107.25	17.93	68.96	86.89
$2^{11}\times2^{11}\times2^{11}$	38.5	85.25	123.75	35	83.5	118.5	19.30	75.65	94.96
$2^{12}\times2^{12}\times2^{12}$	42	93	135	38.5	91.25	129.75	21.60	82.80	104.40

Table 5.2: Number of arithmetic operations required for the computation of the 3-D DFT using the 3mult-3add scheme

	Radix-$(2\times2\times2)$ FFTs						Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FFT		
	Using one butterfly			Using two butterflies			Using two butterflies		
Transform size $N\times N\times N$	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point
$2^3\times2^3\times2^3$	7.87	25.87	33.75	5.25	23.25	28.5	2.95	20.95	23.90
$2^4\times2^4\times2^4$	10.5	34.5	45	7.875	31.87	39.75	5.29	29.29	34.58
$2^5\times2^5\times2^5$	13.12	43.12	56.25	10.5	40.5	51	5.87	35.87	41.74
$2^6\times2^6\times2^6$	15.75	51.75	67.5	13.12	49.12	62.25	7.98	43.98	51.97
$2^7\times2^7\times2^7$	18.37	60.37	78.75	15.75	57.75	73.5	8.76	50.76	59.52
$2^8\times2^8\times2^8$	21	69	90	18.37	66.37	84.75	10.70	58.70	69.41
$2^9\times2^9\times2^9$	23.62	77.62	101.25	21	75	96	11.62	65.62	77.25
$2^{10}\times2^{10}\times2^{10}$	26.25	86.25	112.5	23.62	83.62	107.25	13.44	73.44	86.89
$2^{11}\times2^{11}\times2^{11}$	28.87	94.87	123.75	26.25	92.25	118.5	14.48	80.48	94.96
$2^{12}\times2^{12}\times2^{12}$	31.5	103.5	135	28.87	100.87	129.75	16.20	88.20	104.40

Table 5.3: Number of non-trivial operations required for the computation of the 3-D DFT using the 4mult-2add scheme

	Radix-$(2\times2\times2)$ FFTs			Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FFT		
Transform size $N\times N\times N$	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point
$2^3\times2^3\times2^3$	0.875	18.875	19.75	0.875	18.875	19.75
$2^4\times2^4\times2^4$	3.0625	26.1875	29.25	2.2969	25.4219	27.7188
$2^5\times2^5\times2^5$	5.9062	33.7188	39.625	3.8965	32.4746	36.3711
$2^6\times2^6\times2^6$	9.0781	41.3594	50.4375	5.6687	39.1941	44.8628
$2^7\times2^7\times2^7$	12.4141	49.0547	61.4688	7.4539	46.2599	537138
$2^8\times2^8\times2^8$	15.832	56.7773	72.6094	9.3098	53.05	62.3598
$2^9\times2^9\times2^9$	19.291	64.5137	83.8047	11.1449	60.0949	71.2399
$2^{10}\times2^{10}\times2^{10}$	22.7705	72.2568	95.0273	13.0187	66.9238	79.9425
$2^{11}\times2^{11}\times2^{11}$	26.2603	80.0034	106.2637	14.8689	73.9451	88.814
$2^{12}\times2^{12}\times2^{12}$	29.7551	87.7517	117.5068	16.7449	80.7997	97.5446

Table 5.4: Number of non-trivial operations required for the computation of the 3-D DFT using the 3mult-3add scheme

	Radix-$(2\times2\times2)$ FFTs			Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FFT		
Transform size $N\times N\times N$	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point
$2^3\times2^3\times2^3$	0.2188	19.5312	19.75	0.8750	18.875	19.75
$2^4\times2^4\times2^4$	1.6406	27.6094	29.25	1.8594	25.8594	27.7188
$2^5\times2^5\times2^5$	3.6641	35.9609	39.625	3.1855	33.1855	36.3711
$2^6\times2^6\times2^6$	5.9883	44.4492	50.4375	4.4314	40.4314	44.8628
$2^7\times2^7\times2^7$	8.4629	53.0059	61.4688	5.8569	47.8569	53.7138
$2^8\times2^8\times2^8$	11.0127	61.5967	72.6094	7.1799	55.1799	62.3598
$2^9\times2^9\times2^9$	13.6001	70.2046	83.8047	8.6199	62.6199	71.2399
$2^{10}\times2^{10}\times2^{10}$	16.2063	78.821	95.0273	9.9712	69.9712	79.9425
$2^{11}\times2^{11}\times2^{11}$	18.8219	87.4418	106.2637	11.407	77.4070	88.814
$2^{12}\times2^{12}\times2^{12}$	21.4422	96.0646	117.5068	12.7723	84.7723	97.5446

5.2.2.2 Data Transfers

An appropriate use of the internal registers (on-chip memory) of a processor is becoming an important strategy in the exploitation of the modern computer architectures, since the on-chip memory can be accessed faster than the external memory (off-chip memory). We assume that a sufficient on-chip memory is available to perform an entire butterfly without using any intermediate transfer operations between the processor and the off-chip memory. We first give a brief implementation scheme of the proposed radix-$(2\times 2\times 2)/(4\times 4\times 4)$ FFT algorithm by considering the implementation of the butterfly given by (5.39) and (5.25)-(5.31). For a given combination of (k_1,k_2,k_3), by reading eight points from the off-chip memory according to (5.13) to compute the operation given by (5.12), we obtain the eight points given by (5.14). The first point $\tilde{y}_{000}(k_1,k_2,k_3)$ of (5.14) is returned to the off-chip memory, whereas the other seven points are kept in the on-chip memory since they are used by the seven sub-butterflies given by (5.25)-(5.31). These operations are repeated 8 times to complete the computation of the operation given by (5.39). The seven 8-point results kept in the on-chip memory are used to process the seven sub-butterflies. This completes the process of the butterfly. For all the combinations of (k_1,k_2,k_3), $k_i = 0, 1, ..., (N/4)-1$, the results returned to the off-chip memory are grouped to form the input sequence $\tilde{y}_{000}(k_1,k_2,k_3)$ of the $\left(\frac{N}{2}\times\frac{N}{2}\times\frac{N}{2}\right)$-point 3-D DFT given by (5.4). The 56 input sequences of the $\left(\frac{N}{4}\times\frac{N}{4}\times\frac{N}{4}\right)$-point 3-D DFTs given by (5.18)-(5.24) are formed by grouping separately the results of each line of the output vectors given by (5.25)-(5.31). This completes the first stage of the proposed decomposition.

Then, it can be shown that the expression for the number of data transfers (real and imaginary parts), not including the read operations for the twiddle factors, required by the proposed FFT algorithm is

Table 5.5: Comparison of the number of data transfers

	Radix-$(2\times2\times2)$ FFTs	Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FFT	Savings
Transform size $N\times N\times N$	$D_{r2}(3,N)/N^3$	$D_{r2/4}(3,N)/N^3$	(%)
$2^3\times2^3\times2^3$	6	4.0312	32.81
$2^4\times2^4\times2^4$	8	4.4727	44.09
$2^5\times2^5\times2^5$	10	6.0864	39.13
$2^6\times2^6\times2^6$	12	6.6744	44.38
$2^7\times2^7\times2^7$	14	8.1599	41.71
$2^8\times2^8\times2^8$	16	8.8601	44.62
$2^9\times2^9\times2^9$	18	10.2474	43.07
$2^{10}\times2^{10}\times2^{10}$	20	11.0335	44.83
$2^{11}\times2^{11}\times2^{11}$	22	12.3457	43.88
$2^{12}\times2^{12}\times2^{12}$	24	13.1975	45.01

$$D_{r2/4}(3,N) = 2N^3 + D_{r2/4}\left(3,\frac{N}{2}\right) + 56D_{r2/4}\left(3,\frac{N}{4}\right),\ N>2,$$
$$D_{r2/4}(3,2) = 16,\ D_{r2/4}(3,1) = 0 \tag{5.48}$$

The radix-$(2\times2\times2)$ FFT algorithm requires $\log_2 N$ stages each requiring $2N^3$ transfer operations. Thus, the number of data transfers required by the radix-$(2\times2\times2)$ FFT algorithm is

$$D_{r2}(3,N) = 2N^3\log_2 N \tag{5.49}$$

The numbers of data transfers for the two FFT algorithms are compared in Table 5.5, for various values of N. It is clear from this table that the radix-$(2\times2\times2)/(4\times4\times4)$ FFT algorithm requires about 40% less data transfer operations than that required in the radix-$(2\times2\times2)$ FFT algorithm. Hence, similar reduction is also obtained in the case of the address generation.

5.2.2.3 Twiddle Factors

In counting the number of real twiddle factor (cosine and sine) evaluations or accesses to the lookup table required by the radix-$(2\times2\times2)/(4\times4\times4)$ and radix-$(2\times2\times2)$ FFT algorithms, it is assumed that the 4mult-2add scheme is considered. Hence, the general butterfly of the radix-$(2\times2\times2)/(4\times4\times4)$ FFT algorithm requires 112 twiddle factors, whereas that of the radix-$(2\times2\times2)$ FFT algorithm requires 14 twiddle factors. Therefore, it is seen that the expression for the number of twiddle factors required by the two-butterfly implementation of the radix-$(2\times2\times2)/(4\times4\times4)$ FFT algorithm is given by

$$\mathrm{T}^{42}_{\mathrm{r2}/4}(3,N) = 112\frac{N^3}{64} + \mathrm{T}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right) + 56\mathrm{T}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N>2,$$
$$\mathrm{T}^{42}_{\mathrm{r2}/4}(3,2) = \mathrm{T}^{42}_{\mathrm{r2}/4}(3,1) = 0 \tag{5.50}$$

The corresponding expression for the one-butterfly ($s=0$) or two-butterfly ($s=1$) implementation of the radix-$(2\times2\times2)$ FFT algorithm is given by

$$\mathrm{T}^{42}_{\mathrm{r2}}(3,N) = 14\frac{N^3}{8}\log_2\frac{N}{2^s} \tag{5.51}$$

The numbers of twiddle factor evaluations or accesses to the lookup table for the two algorithms are compared in Table 5.6. It is clear from this table that savings of about 40% in the number of twiddle factor evaluations or accesses to the lookup table can be achieved using the proposed radix-$(2\times2\times2)/(4\times4\times4)$ FFT algorithm over the one-butterfly and two-butterfly implementations of the radix-$(2\times2\times2)$ FFT algorithm. Note that, when the lookup table is used, similar savings are obtained by the proposed FFT algorithm in the address generation for reading the twiddle factors.

Table 5.6: Comparison of the number of twiddle factor evaluations or accesses to the lookup table

Transform size	Radix-$(2\times2\times2)$ FFTs		Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FFT	Savings (%)	
	Using one butterfly	Using two butterflies	Using two butterflies		
$N\times N\times N$	$T^{42}_{r2}(3,N)/N^3$ (A)	$T^{42}_{r2}(3,N)/N^3$ (B)	$T^{42}_{r2/4}(3,N)/N^3$ (C)	in C over A	in C over B
$2^3\times2^3\times2^3$	5.25	3.5	1.9688	62.49	43.74
$2^4\times2^4\times2^4$	7	5.25	3.5273	49.61	32.71
$2^5\times2^5\times2^5$	8.75	7	3.9136	55.27	44.09
$2^6\times2^6\times2^6$	10.5	8.75	5.3256	49.28	39.13
$2^7\times2^7\times2^7$	12.25	10.5	5.8401	52.32	44.38
$2^8\times2^8\times2^8$	14	12.25	7.1399	49.00	41.71
$2^9\times2^9\times2^9$	15.75	14	7.7526	50.77	44.62
$2^{10}\times2^{10}\times2^{10}$	17.5	15.75	8.9665	48.76	43.06
$2^{11}\times2^{11}\times2^{11}$	19.25	17.5	9.6543	49.84	44.83
$2^{12}\times2^{12}\times2^{12}$	21	19.25	10.8025	48.55	43.88

5.3 Radix-2/4 Approach for the Computation of 3-D DHT

A number of FHT algorithms have been developed in [45] for the fast computation of the 1-D DHT. The most attractive ones are those that employ radix-2 or radix-2/4 approach, since the former results in algorithms that are highly regular and simple, whereas the latter in algorithms that provide a good compromise between the arithmetic and structural complexities. The radix-2 approach has been extended for the 2-D DHT case using DIT [58] and, recently, for the 3-D DHT case using DIT [65] as well as DIF [66]. However, the radix-2/4 approach has been extended only for the 2-D DHT case [60], [62], [64] and to this date, no attempt has been made for its extension to the case of three or higher dimensions. This is mainly due to the fact that the kernel of the multidimensional DHT is not separable, mak-

ing the extension of the radix-2/4 approach neither obvious nor straightforward. Therefore, new decomposition strategies and appropriate mathematical tools are necessary in order to develop multidimensional FHT algorithms that are based on the radix-2/4 approach.

We now give a brief review of other approaches proposed so far for the computation of the 3-D DHT. The 3-D DHT can be computed using 1-D FHT algorithms in a row-column fashion. This traditional row-column method requires matrix transposition operations and high computational complexity as well as bit-shift operations. The 3-D DHT can also be computed using 1-D complex-valued FFT algorithms in a row-column fashion [63]. This approach requires special retrograde indexing manipulations, matrix transposition operations and a computational complexity comparable to that required by the row-column method. Another approach has been proposed in [61]. However, it requires the use of 1-D FFT and FHT algorithms and a special interstage scheme for addition operations. In [59], an approach based on Fermat number transforms (FNTs) has been proposed for the computation of the 3-D DHT. It requires the implementation of 1-D, 2-D, and 3-D forward FNTs, 3-D inverse FNT, and a large number of bit-shift operations. It has been shown in [65] that the 3-D radix-$(2 \times 2 \times 2)$ FHT algorithm is more efficient than all the algorithms reported in [59], [61], [63] including the row-column method. Recently, an approach, based on polynomial transforms, has been proposed in [68]. It provides FHT algorithms with reduced arithmetic complexities at the expense of very complicated structures.

In this section, new 3-D radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ DIF and DIT FHT algorithms are proposed for efficient computation of the 3-D DHT. The decomposition strategy used to develop the two algorithms is based on a mixture of radix-$(2 \times 2 \times 2)$ and radix-$(4 \times 4 \times 4)$ index maps, coupled with an appropriate use of the Kronecker product. The butterflies of the proposed FHT algorithms are characterized by simple closed-form expressions.

5.3.1 Proposed Radix-$(2\times2\times2)/(4\times4\times4)$ DIF FHT Algorithm

Using (2.2), the 3-D DHT of size $(N\times N\times N)$ is given by

$$X(n_1,n_2,n_3)=\sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1}\sum_{k_3=0}^{N-1}x(k_1,k_2,k_3)\,\mathrm{cas}\left(\frac{2\pi}{N}\sum_{i=1}^{3}n_ik_i\right),$$
$$0\le n_i\le N-1,\ i=1,2,3 \tag{5.52}$$

Then, using some trigonometric properties, (5.52) can be expressed as

$$X(n_1,n_2,n_3)=\sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1}\sum_{k_3=0}^{N-1}\mathrm{Re}\left[x(k_1,k_2,k_3)\,(1+j)W_N^{\sum_{i=1}^{3}n_ik_i}\right],$$
$$0\le n_i\le N-1 \tag{5.53}$$

In this section, we develop a 3-D radix-$(2\times2\times2)/(4\times4\times4)$ FHT algorithm by employing the radix-2/4 DIF approach. The decomposition of the $(N\times N\times N)$-point 3-D DHT is performed using two steps. In the first step, we introduce radix-$(2\times2\times2)$ index maps by decomposing (5.53) into the following eight indexed terms.

$$\begin{aligned}X(2n_1,2n_2,2n_3)&=\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1}\mathrm{Re}\left[y_{000}(k_1,k_2,k_3)\,(1+j)W_{N/2}^{\sum_{i=1}^{3}n_ik_i}\right]\\&=\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1}y_{000}(k_1,k_2,k_3)\,\mathrm{cas}\left(\frac{2\pi}{N/2}\sum_{i=1}^{3}n_ik_i\right),\\&\qquad 0\le n_i\le\frac{N}{2}-1\end{aligned} \tag{5.54}$$

$$X(2n_1,2n_2,2n_3+1)=\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1}\mathrm{Re}\left[y_{001}(k_1,k_2,k_3)\,W_N^{k_3}(1+j)W_{N/2}^{\sum_{i=1}^{3}n_ik_i}\right],$$
$$0\le n_i\le\frac{N}{2}-1 \tag{5.55}$$

$$X(2n_1, 2n_2+1, 2n_3) = \sum_{k_1=0}^{N/2-1} \sum_{k_2=0}^{N/2-1} \sum_{k_3=0}^{N/2-1} \mathrm{Re}\left[y_{010}(k_1,k_2,k_3)\, W_N^{k_2} (1+j) W_{N/2}^{\sum_{i=1}^{3} n_i k_i} \right],$$

$$0 \le n_i \le \frac{N}{2} - 1 \quad (5.56)$$

$$X(2n_1, 2n_2+1, 2n_3+1)$$

$$= \sum_{k_1=0}^{N/2-1} \sum_{k_2=0}^{N/2-1} \sum_{k_3=0}^{N/2-1} \mathrm{Re}\left[y_{011}(k_1,k_2,k_3)\, W_N^{k_2+k_3} (1+j) W_{N/2}^{\sum_{i=1}^{3} n_i k_i} \right],$$

$$0 \le n_i \le \frac{N}{2} - 1 \quad (5.57)$$

$$X(2n_1+1, 2n_2, 2n_3)$$

$$= \sum_{k_1=0}^{N/2-1} \sum_{k_2=0}^{N/2-1} \sum_{k_3=0}^{N/2-1} \mathrm{Re}\left[y_{100}(k_1,k_2,k_3)\, W_N^{k_1} (1+j) W_{N/2}^{\sum_{i=1}^{3} n_i k_i} \right],$$

$$0 \le n_i \le \frac{N}{2} - 1 \quad (5.58)$$

$$X(2n_1+1, 2n_2, 2n_3+1)$$

$$= \sum_{k_1=0}^{N/2-1} \sum_{k_2=0}^{N/2-1} \sum_{k_3=0}^{N/2-1} \mathrm{Re}\left[y_{101}(k_1,k_2,k_3)\, W_N^{k_1+k_3} (1+j) W_{N/2}^{\sum_{i=1}^{3} n_i k_i} \right],$$

$$0 \le n_i \le \frac{N}{2} - 1 \quad (5.59)$$

$$X(2n_1+1, 2n_2+1, 2n_3)$$

$$= \sum_{k_1=0}^{N/2-1} \sum_{k_2=0}^{N/2-1} \sum_{k_3=0}^{N/2-1} \mathrm{Re}\left[y_{110}(k_1,k_2,k_3)\, W_N^{k_1+k_2} (1+j) W_{N/2}^{\sum_{i=1}^{3} n_i k_i} \right],$$

$$0 \le n_i \le \frac{N}{2} - 1 \quad (5.60)$$

$$X(2n_1+1,2n_2+1,2n_3+1)$$
$$= \sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1} \mathrm{Re}\left[y_{111}(k_1,k_2,k_3)\,W_N^{k_1+k_2+k_3}(1+j)W_{N/2}^{\sum_{i=1}^{3} n_i k_i}\right],$$
$$0 \le n_i \le \frac{N}{2}-1 \quad (5.61)$$

The sequences $y_{abc}(k_1,k_2,k_3)$, for a, b, $c = 0$, 1, in (5.54)-(5.61) are obtained from the original input sequence $x(k_1,k_2,k_3)$ as

$$\mathbf{y}_{k_1,k_2,k_3} = \mathbf{H}_{2^3}\mathbf{x}_{k_1,k_2,k_3},\ 0 \le k_i \le \frac{N}{2}-1 \quad (5.62)$$

where

$$\mathbf{x}_{k_1,k_2,k_3} = \begin{bmatrix} x(k_1,k_2,k_3) \\ x(k_1,k_2,k_3+N/2) \\ x(k_1,k_2+N/2,k_3) \\ x(k_1,k_2+N/2,k_3+N/2) \\ x(k_1+N/2,k_2,k_3) \\ x(k_1+N/2,k_2,k_3+N/2) \\ x(k_1+N/2,k_2+N/2,k_3) \\ x(k_1+N/2,k_2+N/2,k_3+N/2) \end{bmatrix} \quad (5.63)$$

$$\mathbf{y}_{k_1,k_2,k_3} = \begin{bmatrix} y_{000}(k_1,k_2,k_3) \\ y_{001}(k_1,k_2,k_3) \\ y_{010}(k_1,k_2,k_3) \\ y_{011}(k_1,k_2,k_3) \\ y_{100}(k_1,k_2,k_3) \\ y_{101}(k_1,k_2,k_3) \\ y_{110}(k_1,k_2,k_3) \\ y_{111}(k_1,k_2,k_3) \end{bmatrix} \quad (5.64)$$

The even-even-even term given by (5.54) is recognized as a 3-D DHT of size $\left(\frac{N}{2}\times\frac{N}{2}\times\frac{N}{2}\right)$, whose input sequence is obtained from the original input sequence without any twiddle factor. In the second step, we introduce radix-$(4\times4\times4)$ index maps by further decomposing the terms given by (5.55)-(5.61).

Let us start by decomposing the even-even-odd term given by (5.55). This is achieved by changing the variables n_i and k_i by

$$2n_i+p_i,\ 0\le n_i\le\frac{N}{4}-1,\ p_i=0,1 \tag{5.65}$$

and

$$k_i+\frac{N}{4}q_i,\ 0\le k_i\le\frac{N}{4}-1,\ q_i=0,1 \tag{5.66}$$

respectively. Then, (5.55) becomes

$$\begin{aligned}&X(4n_1+2p_1,4n_2+2p_2,4n_3+2p_3+1)\\&=\sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1}\mathrm{Re}\left[\sum_{q_1=0}^{1}\sum_{q_2=0}^{1}\sum_{q_3=0}^{1}y_{001}\left(k_1+\frac{N}{4}q_1,k_2+\frac{N}{4}q_2,k_3+\frac{N}{4}q_3\right)\right.\\&\qquad\left.(-j)^{q_3}(-1)^{\sum_{i=1}^{3}p_iq_i}W_N^{k_3}W_{N/2}^{\sum_{i=1}^{3}p_ik_i}(1+j)W_{N/4}^{\sum_{i=1}^{3}n_ik_i}\right]\end{aligned} \tag{5.67}$$

Using the fact that

$$\begin{aligned}&\sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1}g(k_1,k_2,k_3)\cos\left(-\frac{2\pi}{N/4}\sum_{i=1}^{3}n_ik_i\right)\\&\qquad=\sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1}g\left(\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3\right)\cos\left(\frac{2\pi}{N/4}\sum_{i=1}^{3}n_ik_i\right)\end{aligned} \tag{5.68}$$

for any 3-D sequence $g(k_1,k_2,k_3)$, we see that (5.67) corresponds to a set of eight 3-D

DHTs, each of size $\left(\frac{N}{4}\times\frac{N}{4}\times\frac{N}{4}\right)$, given by

$$X(4n_1+2p_1,4n_2+2p_2,4n_3+2p_3+1) = \sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1} f^{001}_{k_1,k_2,k_3}(p_1,p_2,p_3)\,\mathrm{cas}\left(\frac{2\pi}{N/4}\sum_{i=1}^{3}n_ik_i\right), \quad 0\le n_i\le\frac{N}{4}-1,\ p_i=0,1 \tag{5.69}$$

where the eight input sequences of (5.69) are given by

$$f^{001}_{k_1,k_2,k_3}(p_1,p_2,p_3) = \mathrm{Re}\left[W_N^{k_3}W_{N/2}^{\sum_{i=1}^{3}p_ik_i}\left(\sum_{q_1=0}^{1}\sum_{q_2=0}^{1}\sum_{q_3=0}^{1} y_{001}\left(k_1+\frac{N}{4}q_1,k_2+\frac{N}{4}q_2,k_3+\frac{N}{4}q_3\right)(-j)^{q_3}(-1)^{\sum_{i=1}^{3}p_iq_i}\right)\right.$$
$$\left.+jW_N^{\frac{N}{4}-k_3}W_{N/2}^{\sum_{i=1}^{3}p_i\left(\frac{N}{4}-k_i\right)}\left(\sum_{q_1=0}^{1}\sum_{q_2=0}^{1}\sum_{q_3=0}^{1} y_{001}\left(\frac{N}{4}-k_1+\frac{N}{4}q_1,\frac{N}{4}-k_2+\frac{N}{4}q_2,\frac{N}{4}-k_3+\frac{N}{4}q_3\right)(-j)^{q_3}(-1)^{\sum_{i=1}^{3}p_iq_i}\right)\right] \tag{5.70}$$

For a given combination of (k_1,k_2,k_3), the quantity contained within brackets in (5.70) can be considered as a sum of two 3-D DFTs, each of size $(2\times2\times2)$, multiplied by twiddle factors. The input sequences of these DFTs are obtained using (5.62). Now, using matrix representation, (5.70) can be expressed as

$$\mathbf{f}^{001}_{k_1,k_2,k_3} = \mathrm{Re}\left[\left(\mathbf{C}^{001}_{k_1,k_2,k_3}-j\mathbf{S}^{001}_{k_1,k_2,k_3}\right)\mathbf{H}_{2^3}\left(\mathbf{J}^{001}_{\mathrm{re}}-j\mathbf{J}^{001}_{\mathrm{im}}\right)\mathbf{y}^{001}_{k_1,k_2,k_3}\right.$$
$$\left.+\left(\mathbf{C}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3}-j\mathbf{S}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3}\right)\mathbf{H}_{2^3}\left(\mathbf{J}^{001}_{\mathrm{im}}+j\mathbf{J}^{001}_{\mathrm{re}}\right)\mathbf{y}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3}\right] \tag{5.71}$$

where

$$\mathbf{J}^{001}_{\mathrm{re}} = \mathrm{Re}\left[\mathbf{I}_2\otimes\mathbf{I}_2\otimes\mathbf{J}_2\right], \tag{5.72}$$

$$\mathbf{J}^{001}_{\mathrm{im}} = -\mathrm{Im}\left[\mathbf{I}_2\otimes\mathbf{I}_2\otimes\mathbf{J}_2\right] \tag{5.73}$$

the pth component of the vector $\mathbf{y}^{001}_{k_1,k_2,k_3}$ is given by

$$y^{001}_{k_1,k_2,k_3}(p) = y_{001}\left(k_1+\frac{N}{4}p_1, k_2+\frac{N}{4}p_2, k_3+\frac{N}{4}p_3\right), \tag{5.74}$$

the (p,p)th component of the diagonal matrix $\mathbf{C}^{001}_{k_1,k_2,k_3}$ by

$$C^{001}_{k_1,k_2,k_3}(p,p) = \cos\left(\frac{2\pi}{N}(2p_1k_1+2p_2k_2+(2p_3+1)k_3)\right), \tag{5.75}$$

the (p,p)th component of the diagonal matrix $\mathbf{S}^{001}_{k_1,k_2,k_3}$ by

$$S^{001}_{k_1,k_2,k_3}(p,p) = \sin\left(\frac{2\pi}{N}(2p_1k_1+2p_2k_2+(2p_3+1)k_3)\right) \tag{5.76}$$

and

$p = 4p_1+2p_2+p_3$, $p_i(i=1,2,3) = 0$ or 1.

The components of the vector $\mathbf{f}^{001}_{k_1,k_2,k_3}$ in (5.71) are related to the input sequence of (5.69) by

$$f^{001}_{k_1,k_2,k_3}(p) = f^{001}_{k_1,k_2,k_3}(p_1,p_2,p_3) \tag{5.77}$$

It can be seen from (5.75) that

$$C^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3}(p,p) = (-1)^{p_1+p_2+p_3}S^{001}_{k_1,k_2,k_3}(p,p) \tag{5.78}$$

Similarly, from (5.76), we have

$$S^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3}(p,p) = (-1)^{p_1+p_2+p_3}C^{001}_{k_1,k_2,k_3}(p,p) \tag{5.79}$$

Using (5.78) and (5.79), it can be shown that

$$\mathbf{C}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3} = \mathbf{S}^{001}_{k_1,k_2,k_3}\mathbf{R} \tag{5.80}$$

and

$$\mathbf{S}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3} = \mathbf{C}^{001}_{k_1,k_2,k_3}\mathbf{R} \tag{5.81}$$

where

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{5.82}$$

Now, by substituting (5.80) and (5.81) in (5.71), we obtain

$$\begin{aligned} \mathbf{f}^{001}_{k_1,k_2,k_3} = \mathbf{C}^{001}_{k_1,k_2,k_3}\left(\mathbf{H}_{2^3}\mathbf{J}^{001}_{\text{re}}\mathbf{y}^{001}_{k_1,k_2,k_3} + \mathbf{R}\mathbf{H}_{2^3}\mathbf{J}^{001}_{\text{re}}\mathbf{y}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3}\right) \\ -\mathbf{S}^{001}_{k_1,k_2,k_3}\left(\mathbf{H}_{2^3}\mathbf{J}^{001}_{\text{im}}\mathbf{y}^{001}_{k_1,k_2,k_3} - \mathbf{R}\mathbf{H}_{2^3}\mathbf{J}^{001}_{\text{im}}\mathbf{y}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3}\right), \\ 0 \le k_i \le \frac{N}{4}-1 \end{aligned} \tag{5.83}$$

In order to perform the in-place computation, we use the same input vectors of (5.83) in the computation of the vector $\mathbf{f}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3}$. This can be achieved by replacing k_i by $\frac{N}{4}-k_i$ in (5.83) and using (5.80) and (5.81). Then we obtain

$$\mathbf{f}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3} = \mathbf{S}^{001}_{k_1,k_2,k_3}\left(\mathbf{H}_{2^3}\mathbf{J}^{001}_{\text{re}}\mathbf{y}^{001}_{k_1,k_2,k_3} + \mathbf{R}\mathbf{H}_{2^3}\mathbf{J}^{001}_{\text{re}}\mathbf{y}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3}\right)$$
$$+\mathbf{C}^{001}_{k_1,k_2,k_3}\left(\mathbf{H}_{2^3}\mathbf{J}^{001}_{\text{im}}\mathbf{y}^{001}_{k_1,k_2,k_3} - \mathbf{R}\mathbf{H}_{2^3}\mathbf{J}^{001}_{\text{im}}\mathbf{y}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3}\right),$$
$$0 \le k_i \le \frac{N}{4}-1 \quad (5.84)$$

The redundant operations involved in the computation of $\mathbf{f}^{001}_{k_1,k_2,k_3}$ and $\mathbf{f}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3}$ can be avoided by combining (5.83) and (5.84). Therefore, we may obtain a sub-butterfly for computing the input sequences of (5.69) as

$$\begin{bmatrix} \mathbf{f}^{001}_{k_1,k_2,k_3} \\ \mathbf{f}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3} \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{C}^{001}_{k_1,k_2,k_3} & -\mathbf{S}^{001}_{k_1,k_2,k_3} \\ \mathbf{S}^{001}_{k_1,k_2,k_3} & \mathbf{C}^{001}_{k_1,k_2,k_3} \end{bmatrix} \begin{bmatrix} \mathbf{H}_{2^3}\mathbf{J}^{001}_{\text{re}} & \mathbf{R}\mathbf{H}_{2^3}\mathbf{J}^{001}_{\text{re}} \\ \mathbf{H}_{2^3}\mathbf{J}^{001}_{\text{im}} & -\mathbf{R}\mathbf{H}_{2^3}\mathbf{J}^{001}_{\text{im}} \end{bmatrix} \begin{bmatrix} \mathbf{y}^{001}_{k_1,k_2,k_3} \\ \mathbf{y}^{001}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3} \end{bmatrix} \quad (5.85)$$

where one of the indices k_1, k_2, and k_3 varies from 0 to $\left(\frac{N}{8}-1\right)$, whereas the others vary from 0 to $\left(\frac{N}{4}-1\right)$. Let us assume that $0 \le k_1 \le \frac{N}{8}-1$ and $0 \le k_2, k_3 \le \frac{N}{4}-1$.

In order to introduce radix-$(4\times4\times4)$ index maps for the other terms given by (5.56)-(5.61), we apply similar decomposition and techniques to those used for deriving the sub-butterfly given by (5.85) of the even-even-odd term. Then, it is seen that (5.55)-(5.61) can be mapped to a set of 56 DHTs, each of size $\left(\frac{N}{4}\times\frac{N}{4}\times\frac{N}{4}\right)$, given by

$$X(4n_1+2p_1+a, 4n_2+2p_2+b, 4n_3+2p_3+c)$$
$$= \sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1} f^{abc}_{k_1,k_2,k_3}(p_1,p_2,p_3)\,\text{cas}\left(\frac{2\pi}{N/4}\sum_{i=1}^{3} n_i k_i\right),$$
$$0 \le n_i \le \frac{N}{4}-1,\ p_i, a, b, c = 0, 1,\ (a,b,c) \ne (0,0,0) \quad (5.86)$$

The input sequences $f^{abc}_{k_1,k_2,k_3}(p_1,p_2,p_3)$ in (5.84) are obtained from a general sub-butterfly of the proposed radix-$(2\times2\times2)/(4\times4\times4)$ DIF FHT algorithm that is given by a closed-form expression as

$$\begin{bmatrix} \mathbf{f}^{abc}_{k_1,k_2,k_3} \\ \mathbf{f}^{abc}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3} \end{bmatrix} = \begin{bmatrix} \mathbf{C}^{abc}_{k_1,k_2,k_3} & -\mathbf{S}^{abc}_{k_1,k_2,k_3} \\ \mathbf{S}^{abc}_{k_1,k_2,k_3} & \mathbf{C}^{abc}_{k_1,k_2,k_3} \end{bmatrix} \mathbf{A}^{abc} \begin{bmatrix} \mathbf{y}^{abc}_{k_1,k_2,k_3} \\ \mathbf{y}^{abc}_{\frac{N}{4}-k_1,\frac{N}{4}-k_2,\frac{N}{4}-k_3} \end{bmatrix},$$
$$0 \le k_1 \le \frac{N}{8}-1,\ 0 \le k_2,k_3 \le \frac{N}{4}-1,\ a,b,c=0,1,(a,b,c)\neq(0,0,0) \quad (5.87)$$

where

$$\mathbf{A}^{abc} = \begin{cases} \begin{bmatrix} \mathbf{H}_{2^3}\mathbf{J}^{abc}_{\mathrm{re}} & (-1)^{\frac{a+b+c-1}{2}}\mathbf{R}\mathbf{H}_{2^3}\mathbf{J}^{abc}_{\mathrm{re}} \\ \mathbf{H}_{2^3}\mathbf{J}^{abc}_{\mathrm{im}} & -(-1)^{\frac{a+b+c-1}{2}}\mathbf{R}\mathbf{H}_{2^3}\mathbf{J}^{abc}_{\mathrm{im}} \end{bmatrix}, & \text{when } a+b+c \text{ is odd} \\ \\ \begin{bmatrix} \mathbf{H}_{2^3}\mathbf{J}^{abc}_{\mathrm{re}} & (-1)^{\frac{a+b+c}{2}}\mathbf{R}\mathbf{H}_{2^3}\mathbf{J}^{abc}_{\mathrm{im}} \\ \mathbf{H}_{2^3}\mathbf{J}^{abc}_{\mathrm{im}} & (-1)^{\frac{a+b+c}{2}}\mathbf{R}\mathbf{H}_{2^3}\mathbf{J}^{abc}_{\mathrm{re}} \end{bmatrix}, & \text{when } a+b+c \text{ is even} \end{cases}, \quad (5.88)$$

$$\mathbf{J}^{abc}_{\mathrm{re}} = \mathrm{Re}\left[\mathbf{J}^a \otimes \mathbf{J}^b \otimes \mathbf{J}^c\right],\ \mathbf{J}^0 = \mathbf{I}_2,\ \mathbf{J}^1 = \mathbf{J}_2, \quad (5.89)$$

$$\mathbf{J}^{abc}_{\mathrm{im}} = -\mathrm{Im}\left[\mathbf{J}^a \otimes \mathbf{J}^b \otimes \mathbf{J}^c\right], \quad (5.90)$$

$$\mathbf{y}^{abc}_{k_1,k_2,k_3} = \begin{bmatrix} y_{abc}(k_1,k_2,k_3) \\ y_{abc}(k_1,k_2,k_3+N/4) \\ y_{abc}(k_1,k_2+N/4,k_3) \\ y_{abc}(k_1,k_2+N/4,k_3+N/4) \\ y_{abc}(k_1+N/4,k_2,k_3) \\ y_{abc}(k_1+N/4,k_2,k_3+N/4) \\ y_{abc}(k_1+N/4,k_2+N/4,k_3) \\ y_{abc}(k_1+N/4,k_2+N/4,k_3+N/4) \end{bmatrix}, \quad (5.91)$$

$$C^{abc}_{k_1,k_2,k_3}(p,p) = \cos\left(\frac{2\pi}{N}\left((2p_1+a)k_1+(2p_2+b)k_2+(2p_3+c)k_3\right)\right), \tag{5.92}$$

$$S^{abc}_{k_1,k_2,k_3}(p,p) = \sin\left(\frac{2\pi}{N}\left((2p_1+a)k_1+(2p_2+b)k_2+(2p_3+c)k_3\right)\right) \tag{5.93}$$

and the components of the vector $\mathbf{f}^{abc}_{k_1,k_2,k_3}$ are related to the input sequences of (5.86) by

$$\mathbf{f}^{abc}_{k_1,k_2,k_3} = \begin{bmatrix} f^{abc}_{k_1,k_2,k_3}(0,0,0) \\ f^{abc}_{k_1,k_2,k_3}(0,0,1) \\ f^{abc}_{k_1,k_2,k_3}(0,1,0) \\ f^{abc}_{k_1,k_2,k_3}(0,1,1) \\ f^{abc}_{k_1,k_2,k_3}(1,0,0) \\ f^{abc}_{k_1,k_2,k_3}(1,0,1) \\ f^{abc}_{k_1,k_2,k_3}(1,1,0) \\ f^{abc}_{k_1,k_2,k_3}(1,1,1) \end{bmatrix} \tag{5.94}$$

For given combinations of (k_1,k_2,k_3) and (a,b,c), the sub-butterfly given by (5.87) computes 16 points using 16 points obtained from (5.62). For a given combination of (k_1,k_2,k_3), the sub-butterflies corresponding to the seven combinations of (a,b,c) require 112 points to be computed using (5.62). In order to compute, for a fixed combination of (k_1,k_2,k_3), all the points required by (5.87), we rearrange (5.62) as

$$\begin{bmatrix} \mathbf{y}_{k_1+\frac{N}{4}p_1,k_2+\frac{N}{4}p_2,k_3+\frac{N}{4}p_3} \\ \mathbf{y}_{\frac{N}{4}-k_1+\frac{N}{4}p_1,\frac{N}{4}-k_2+\frac{N}{4}p_2,\frac{N}{4}-k_3+\frac{N}{4}p_3} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{2^3} & 0 \\ 0 & \mathbf{H}_{2^3} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{k_1+\frac{N}{4}p_1,k_2+\frac{N}{4}p_2,k_3+\frac{N}{4}p_3} \\ \mathbf{x}_{\frac{N}{4}-k_1+\frac{N}{4}p_1,\frac{N}{4}-k_2+\frac{N}{4}p_2,\frac{N}{4}-k_3+\frac{N}{4}p_3} \end{bmatrix},$$
$$0 \le k_1 \le \frac{N}{8}-1,\ 0 \le k_2,k_3 \le \frac{N}{4}-1,\ p_i = 0,1 \tag{5.95}$$

It is clear that, for a fixed combination of (k_1,k_2,k_3), (5.95) computes 128 points from the initial input sequence $x(k_1,k_2,k_3)$, where 16 points are used to form the input sequence $y_{000}(k_1,k_2,k_3)$ of the $\left(\frac{N}{2}\times\frac{N}{2}\times\frac{N}{2}\right)$-point 3-D DHT given by (5.54) and all the other points are used by the sub-butterfly given by (5.87). The results of (5.87) are used to form the input sequences of the corresponding $\left(\frac{N}{4}\times\frac{N}{4}\times\frac{N}{4}\right)$-point 3-D DHTs given by (5.86).

In summary, the proposed radix-$(2\times2\times2)/(4\times4\times4)$ DIF FHT algorithm corresponds to decomposing the $(N\times N\times N)$-point 3-D DHT into one $\left(\frac{N}{2}\times\frac{N}{2}\times\frac{N}{2}\right)$-point 3-D DHT given by (5.54) and 56 $\left(\frac{N}{4}\times\frac{N}{4}\times\frac{N}{4}\right)$-point 3-D DHTs given by (5.86), in the first stage. This is achieved by repeating $\frac{N^3}{128}$ times the butterfly based on (5.95) and (5.87). This decomposition scheme is repeated successively for each of the new resulting DHTs, until the problem is reduced to some $(2\times2\times2)$-point 3-D DHTs in the last stage.

5.3.2 Proposed Radix-$(2\times2\times2)/(4\times4\times4)$ DIT FHT Algorithm

In this section, we develop a 3-D radix-$(2\times2\times2)/(4\times4\times4)$ FHT algorithm by employing the radix-2/4 DIT approach. The decomposition of the $(N\times N\times N)$-point 3-D DHT is performed using the following two steps.

Step 1

In this step, we introduce radix-$(2\times2\times2)$ index maps by changing the variables n_i and k_i in (5.53) by

$$n_i+\frac{N}{2}p_i,\ 0\le n_i\le\frac{N}{2}-1,\ p_i=0,1 \tag{5.96}$$

and

$$2k_i+q_i,\ 0\le k_i\le\frac{N}{2}-1,\ q_i=0,1 \tag{5.97}$$

respectively. Then, (5.53) becomes

$$X\left(n_1+\frac{N}{2}p_1, n_2+\frac{N}{2}p_2, n_3+\frac{N}{2}p_3\right) = \sum_{q_1=0}^{1}\sum_{q_2=0}^{1}\sum_{q_3=0}^{1}\left(\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1}\right.$$
$$\left.\mathrm{Re}\left[x(2k_1+q_1, 2k_2+q_2, 2k_3+q_3)\, W_N^{\sum_{i=1}^{3} n_i q_i}(1+j) W_{N/2}^{\sum_{i=1}^{3} n_i k_i}\right]\right)(-1)^{\sum_{i=1}^{3} p_i q_i},$$
$$0 \le n_i \le \frac{N}{2}-1,\ p_i = 0, 1 \quad (5.98)$$

According to the definition of the 3-D DHT, (5.98) is recognized, for a fixed combination of (n_1, n_2, n_3), as a $(2\times 2\times 2)$-point 3-D DHT that can be expressed in a matrix form as

$$\mathbf{X}_{n_1,n_2,n_3} = \mathbf{H}_{2^3}\mathbf{Y}_{n_1,n_2,n_3},\ 0 \le n_i \le \frac{N}{2}-1 \quad (5.99)$$

where

$$\mathbf{Y}_{n_1,n_2,n_3} = \begin{bmatrix} Y_{000}(n_1,n_2,n_3) \\ Y_{001}(n_1,n_2,n_3) \\ Y_{010}(n_1,n_2,n_3) \\ Y_{011}(n_1,n_2,n_3) \\ Y_{100}(n_1,n_2,n_3) \\ Y_{101}(n_1,n_2,n_3) \\ Y_{110}(n_1,n_2,n_3) \\ Y_{111}(n_1,n_2,n_3) \end{bmatrix} \quad (5.100)$$

and

$$\mathbf{X}_{n_1,n_2,n_3} = \begin{bmatrix} X(n_1,n_2,n_3) \\ X(n_1,n_2,n_3+N/2) \\ X(n_1,n_2+N/2,n_3) \\ X(n_1,n_2+N/2,n_3+N/2) \\ X(n_1+N/2,n_2,n_3) \\ X(n_1+N/2,n_2,n_3+N/2) \\ X(n_1+N/2,n_2+N/2,n_3) \\ X(n_1+N/2,n_2+N/2,n_3+N/2) \end{bmatrix} \tag{5.101}$$

The components of the vector given by (5.100) are expressed as

$$Y_{abc}(n_1,n_2,n_3) = \sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1} \operatorname{Re}\left[x(2k_1+a,2k_2+b,2k_3+c)\,W_N^{an_1+bn_2+cn_3}(1+j)W_{N/2}^{\sum_{i=1}^{3}n_ik_i}\right], \quad 0 \le n_i \le \frac{N}{2}-1,\ a,b,c=0,1 \tag{5.102}$$

For the case of $a=b=c=0$, (5.102) reduces to a $\left(\frac{N}{2}\times\frac{N}{2}\times\frac{N}{2}\right)$-point 3-D DHT given by

$$Y_{000}(n_1,n_2,n_3) = \sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\sum_{k_3=0}^{N/2-1} x(2k_1,2k_2,2k_3)\operatorname{cas}\left(\frac{2\pi}{N/2}\sum_{i=1}^{3}n_ik_i\right), \quad 0 \le n_i \le \frac{N}{2}-1 \tag{5.103}$$

Step 2

In this step, we introduce radix-$(4\times4\times4)$ index maps by further decomposing (5.102), for $(a,b,c) \neq (0,0,0)$. This is achieved by changing the variables n_i and k_i in (5.102) by

$$n_i + \frac{N}{4}p_i,\ 0 \le n_i \le \frac{N}{4}-1,\ p_i = 0,1 \tag{5.104}$$

and

$$2k_i+q_i,\ 0\le k_i\le \frac{N}{4}-1,\ q_i=0,1 \tag{5.105}$$

respectively. Then, using the facts that $\cos\left(-\frac{2\pi}{N/4}\sum_{i=1}^{3}n_ik_i\right)=\cos\left(\frac{2\pi}{N/4}\sum_{i=1}^{3}\left(\frac{N}{4}-n_i\right)k_i\right)$ and

$(1+j)W_{N/4}^{\sum_{i=1}^{3}n_ik_i}=\cos\left(\frac{2\pi}{N/4}\sum_{i=1}^{3}n_ik_i\right)+j\cos\left(-\frac{2\pi}{N/4}\sum_{i=1}^{3}n_ik_i\right)$, (5.102) can be written as

$$Y_{abc}\left(n_1+\frac{N}{4}p_1,n_2+\frac{N}{4}p_2,n_3+\frac{N}{4}p_3\right)=\mathrm{Re}\left[(-j)^{ap_1+bp_2+cp_3}\sum_{q_1=0}^{1}\sum_{q_2=0}^{1}\sum_{q_3=0}^{1}\right.$$
$$\left(W_N^{an_1+bn_2+cn_3}W_{N/2}^{\sum_{i=1}^{3}n_iq_i}F^{abc}_{n_1,n_2,n_3}(q_1,q_2,q_3)\right)(-1)^{\sum_{i=1}^{3}p_iq_i}+j(-j)^{ap_1+bp_2+cp_3}$$
$$\left.\sum_{q_1=0}^{1}\sum_{q_2=0}^{1}\sum_{q_3=0}^{1}\left(W_N^{an_1+bn_2+cn_3}W_{N/2}^{\sum_{i=1}^{3}n_iq_i}F^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}(q_1,q_2,q_3)\right)(-1)^{\sum_{i=1}^{3}p_iq_i}\right],$$
$$0\le n_i\le\frac{N}{4}-1\ p_i=0,1,\ a,b,c=0,1,(a,b,c)\neq(0,0,0) \tag{5.106}$$

where

$$F^{abc}_{n_1,n_2,n_3}(q_1,q_2,q_3)=\sum_{k_1=0}^{N/4-1}\sum_{k_2=0}^{N/4-1}\sum_{k_3=0}^{N/4-1}$$
$$x(4k_1+2q_1+a,4k_2+2q_2+b,4k_3+2q_3+c)\cos\left(\frac{2\pi}{N/4}\sum_{i=1}^{3}n_ik_i\right),$$
$$0\le n_i\le\frac{N}{4}-1,\ q_i=0,1,\ a,b,c=0,1,(a,b,c)\neq(0,0,0) \tag{5.107}$$

It is clear that (5.107) represents a set of 56 $\left(\frac{N}{4}\times\frac{N}{4}\times\frac{N}{4}\right)$-point 3-D DHTs. For fixed combinations of (n_1,n_2,n_3) and (a,b,c), the quantity between brackets in (5.106) can be recognized as a sum of two $(2\times2\times2)$-point 3-D DFTs that are multiplied by some trivial twiddle factors 1, -1, j or $-j$. The input sequences of these two DFTs are obtained using

(5.107). Now, using matrix form representation, (5.106) can be expressed as

$$\begin{aligned}
\mathbf{Y}^{abc}_{n_1,n_2,n_3} = \quad & \mathrm{Re}\Big[\Big(\mathbf{J}^{abc}_{\mathrm{re}} - j\mathbf{J}^{abc}_{\mathrm{im}}\Big)\mathbf{H}_{2^3}\left(\mathbf{C}^{abc}_{n_1,n_2,n_3} - j\mathbf{S}^{abc}_{n_1,n_2,n_3}\right)\mathbf{F}^{abc}_{n_1,n_2,n_3} \\
& + \Big(\mathbf{J}^{abc}_{\mathrm{im}} + j\mathbf{J}^{abc}_{\mathrm{re}}\Big)\mathbf{H}_{2^3}\left(\mathbf{C}^{abc}_{n_1,n_2,n_3} - j\mathbf{S}^{abc}_{n_1,n_2,n_3}\right)\mathbf{F}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}\Big] \\
= \quad & \mathbf{J}^{abc}_{\mathrm{re}}\mathbf{H}_{2^3}\left(\mathbf{C}^{abc}_{n_1,n_2,n_3}\mathbf{F}^{abc}_{n_1,n_2,n_3} + \mathbf{S}^{abc}_{n_1,n_2,n_3}\mathbf{F}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}\right) \\
& + \mathbf{J}^{abc}_{\mathrm{im}}\mathbf{H}_{2^3}\left(-\mathbf{S}^{abc}_{n_1,n_2,n_3}\mathbf{F}^{abc}_{n_1,n_2,n_3} + \mathbf{C}^{abc}_{n_1,n_2,n_3}\mathbf{F}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}\right), \\
& 0 \le n_i \le \tfrac{N}{4}-1,\ a,b,c = 0,1,\ (a,b,c) \neq (0,0,0)
\end{aligned} \tag{5.108}$$

where

$$\mathbf{F}^{abc}_{n_1,n_2,n_3} = \begin{bmatrix} F^{abc}_{n_1,n_2,n_3}(0,0,0) \\ F^{abc}_{n_1,n_2,n_3}(0,0,1) \\ F^{abc}_{n_1,n_2,n_3}(0,1,0) \\ F^{abc}_{n_1,n_2,n_3}(0,1,1) \\ F^{abc}_{n_1,n_2,n_3}(1,0,0) \\ F^{abc}_{n_1,n_2,n_3}(1,0,1) \\ F^{abc}_{n_1,n_2,n_3}(1,1,0) \\ F^{abc}_{n_1,n_2,n_3}(1,1,1) \end{bmatrix} \tag{5.109}$$

and

$$\mathbf{Y}^{abc}_{n_1,n_2,n_3} = \begin{bmatrix} Y_{abc}(n_1,n_2,n_3) \\ Y_{abc}(n_1,n_2,n_3+N/4) \\ Y_{abc}(n_1,n_2+N/4,n_3) \\ Y_{abc}(n_1,n_2+N/4,n_3+N/4) \\ Y_{abc}(n_1+N/4,n_2,n_3) \\ Y_{abc}(n_1+N/4,n_2,n_3+N/4) \\ Y_{abc}(n_1+N/4,n_2+N/4,n_3) \\ Y_{abc}(n_1+N/4,n_2+N/4,n_3+N/4) \end{bmatrix} \tag{5.110}$$

Since the in-place computation property is highly desirable to reduce the storage, especially for the 3-D case, we use the same input samples of (5.108) in the computation of the vector $\mathbf{Y}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}$. This can be achieved by replacing n_i by $\frac{N}{4}-n_i$ in (5.108). Then, we obtain

$$\begin{aligned}
&\mathbf{Y}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3} \\
&= \mathbf{J}^{abc}_{\mathrm{re}}\mathbf{H}_{2^3}\left(\mathbf{C}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}\mathbf{F}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}+\mathbf{S}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}\mathbf{F}^{abc}_{n_1,n_2,n_3}\right) \\
&\quad +\mathbf{J}^{abc}_{\mathrm{im}}\mathbf{H}_{2^3}\left(-\mathbf{S}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}\mathbf{F}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}+\mathbf{C}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}\mathbf{F}^{abc}_{n_1,n_2,n_3}\right), \\
&\qquad\qquad 0\le n_i\le \frac{N}{4}-1,\ a,b,c=0,1,(a,b,c)\neq(0,0,0) \qquad (5.111)
\end{aligned}$$

From (5.92), it is seen that

$$\mathbf{C}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}=\begin{cases}(-1)^{\frac{a+b+c-1}{2}}\mathbf{RS}^{abc}_{n_1,n_2,n_3}, & \text{when } a+b+c \text{ is odd}\\ (-1)^{\frac{a+b+c}{2}}\mathbf{RC}^{abc}_{n_1,n_2,n_3}, & \text{when } a+b+c \text{ is even}\end{cases} \qquad (5.112)$$

Similarly, from (5.93), we obtain

$$\mathbf{S}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3}=\begin{cases}(-1)^{\frac{a+b+c-1}{2}}\mathbf{RC}^{abc}_{n_1,n_2,n_3}, & \text{when } a+b+c \text{ is odd}\\ -(-1)^{\frac{a+b+c}{2}}\mathbf{RS}^{abc}_{n_1,n_2,n_3}, & \text{when } a+b+c \text{ is even}\end{cases} \qquad (5.113)$$

Now, in order to avoid the redundant operations in (5.108) and (5.111), we use (5.112) and (5.113) in (5.111), combine the result of (5.111) with (5.108), and vary one of the indices out of n_1, n_2, and n_3 from 0 to $\left(\frac{N}{8}-1\right)$ and the remaining ones from 0 to $\left(\frac{N}{4}-1\right)$. Without loss of generality, let us assume that $0\le n_1\le\frac{N}{8}-1$ and $0\le n_2,n_3\le\frac{N}{4}-1$. Therefore, a closed-form expression for the general sub-butterfly of the proposed radix-

$(2\times2\times2)/(4\times4\times4)$ DIT FHT algorithm is given by

$$\begin{bmatrix} \mathbf{Y}^{abc}_{n_1,n_2,n_3} \\ \mathbf{Y}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3} \end{bmatrix} = \mathbf{A}^{abc}_t \begin{bmatrix} \mathbf{C}^{abc}_{n_1,n_2,n_3} & \mathbf{S}^{abc}_{n_1,n_2,n_3} \\ -\mathbf{S}^{abc}_{n_1,n_2,n_3} & \mathbf{C}^{abc}_{n_1,n_2,n_3} \end{bmatrix} \begin{bmatrix} \mathbf{F}^{abc}_{n_1,n_2,n_3} \\ \mathbf{F}^{abc}_{\frac{N}{4}-n_1,\frac{N}{4}-n_2,\frac{N}{4}-n_3} \end{bmatrix},$$
$$0 \le n_1 \le \frac{N}{8}-1,\ 0 \le n_2, n_3 \le \frac{N}{4}-1,\ a,b,c = 0,1,(a,b,c) \ne (0,0,0) \quad (5.114)$$

where the matrix $\mathbf{A}^{abc}_t$ is the transpose of the matrix $\mathbf{A}^{abc}$ given by (5.88). For given combinations of (n_1,n_2,n_3) and (a,b,c), the sub-butterfly given by (5.114) computes 16 points that require the knowledge of the 16 points of the right hand side vector, which in turn are obtained from the DHTs given by (5.107). For a given combination of (n_1,n_2,n_3), the sub-butterflies corresponding to the seven combinations of (a,b,c) require 112 points to be obtained from (5.107). In order to use, for a fixed combination of (n_1,n_2,n_3), all the points computed by (5.114), we rearrange (5.99) as

$$\begin{bmatrix} \mathbf{X}_{n_1+\frac{N}{4}p_1,n_2+\frac{N}{4}p_2,n_3+\frac{N}{4}p_3} \\ \mathbf{X}_{\frac{N}{4}-n_1+\frac{N}{4}p_1,\frac{N}{4}-n_2+\frac{N}{4}p_2,\frac{N}{4}-n_3+\frac{N}{4}p_3} \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{H}_{2^3} & 0 \\ 0 & \mathbf{H}_{2^3} \end{bmatrix} \begin{bmatrix} \mathbf{Y}_{n_1+\frac{N}{4}p_1,n_2+\frac{N}{4}p_2,n_3+\frac{N}{4}p_3} \\ \mathbf{Y}_{\frac{N}{4}-n_1+\frac{N}{4}p_1,\frac{N}{4}-n_2+\frac{N}{4}p_2,\frac{N}{4}-n_3+\frac{N}{4}p_3} \end{bmatrix},$$
$$0 \le n_1 \le \frac{N}{8}-1,\ 0 \le n_2, n_3 \le \frac{N}{4}-1,\ p_i = 0,1 \quad (5.115)$$

It is clear that, for a fixed combination of (n_1,n_2,n_3), (5.115) computes 128 points of the desired output sequence $X(n_1,n_2,n_3)$ using 16 points obtained from the output sequence $Y_{000}(n_1,n_2,n_3)$ of the $\left(\frac{N}{2}\times\frac{N}{2}\times\frac{N}{2}\right)$-point 3-D DHT given by (5.103) and 112 points obtained from the sub-butterfly given by (5.114). The input points of (5.114) are obtained from the output sequences of the corresponding $\left(\frac{N}{4}\times\frac{N}{4}\times\frac{N}{4}\right)$-point 3-D DHTs given by (5.107).

In summary, in the proposed radix-$(2\times2\times2)/(4\times4\times4)$ DIT FHT algorithm, the computation of the $(N\times N\times N)$-point 3-D DHT consists of computing one $\left(\frac{N}{2}\times\frac{N}{2}\times\frac{N}{2}\right)$-point 3-D DHT given by (5.103) and 56 $\left(\frac{N}{4}\times\frac{N}{4}\times\frac{N}{4}\right)$-point 3-D DHTs given by (5.107). The output points of the DHTs given by (5.107) are used to process the sub-butterfly given by (5.114). The results obtained from (5.114) and the output points of the DHT given by (5.103) are used in (5.115) to compute the samples of the desired output sequence $X(n_1,n_2,n_3)$. The desired 3-D DHT is obtained by repeating $\frac{N^3}{128}$ times the butterfly given by (5.114) and (5.115). This computation process is applied successively for the computation of each of the required DHTs until only some $(2\times2\times2)$-point 3-D DHTs need to be computed.

5.3.3 Computational Complexity

In this section, we consider the performance of the proposed radix-$(2\times2\times2)/(4\times4\times4)$ DIF and DIT FHT algorithms by analyzing their computational complexities and comparing them with those of the existing 3-D FHT algorithms. This comparison is made with the 3-D radix-$(2\times2\times2)$ FHT algorithms of [65] and [66]. Note that the computational complexities of the algorithms of [65] and [66] are exactly the same, and the algorithm in [65] has been shown to be more efficient than all the other existing algorithms appearing before 2001 and reported in [59], [61], [63] including the traditional row-column method. The comparison is also made with a subsequent algorithm reported in [68]. Since the proposed radix-$(2\times2\times2)/(4\times4\times4)$ DIF and DIT FHT algorithms are based on the same decomposition philosophy, their computational complexities are exactly the same. Therefore, the analysis of the computational complexity of only one would be sufficient. Let us consider the computational complexity of the proposed radix-$(2\times2\times2)/(4\times4\times4)$ DIF FHT algorithm.

5.3.3.1 Arithmetic Complexity

From (5.89) and (5.90), we observe that four of the elements of the diagonal matrix $\mathbf{J}_{\text{re}}^{abc}$ or $\mathbf{J}_{\text{im}}^{abc}$ are zero, whereas the remaining four elements are either 1 or -1. Hence, the matrix $\mathbf{H}_{2^3}\mathbf{J}_{\text{re}}^{abc}$ or $\mathbf{H}_{2^3}\mathbf{J}_{\text{im}}^{abc}$ introduces only 8 additions in (5.87). If the 4mult-2add scheme is considered, the general butterfly of the proposed 3-D DIF FHT algorithm based on (5.95) and (5.87) requires 224 multiplications and 832 additions. The decomposition process of the algorithm consists of dividing the $(N \times N \times N)$-point DHT into one $\left(\frac{N}{2} \times \frac{N}{2} \times \frac{N}{2}\right)$-point DHT and 56 $\left(\frac{N}{4} \times \frac{N}{4} \times \frac{N}{4}\right)$-point DHTs. This is achieved by performing $\frac{N^3}{128}$ general butterflies. The decomposition process is repeated successively for each of the new resulting DHTs, until the problem is reduced to some $(2 \times 2 \times 2)$-point 3-D DHTs in the last stage. Therefore, it is seen that the expressions for the numbers of multiplications and additions required by the two-butterfly implementation (one general and one for computing $(2 \times 2 \times 2)$-point DHTs) of the proposed FHT algorithm are, respectively,

$$\mathrm{M}_{\mathrm{r2}/4}^{42}(3,N) = 1.75N^3 + \mathrm{M}_{\mathrm{r2}/4}^{42}\left(3,\frac{N}{2}\right) + 56\mathrm{M}_{\mathrm{r2}/4}^{42}\left(3,\frac{N}{4}\right),\ N > 2,$$
$$\mathrm{M}_{\mathrm{r2}/4}^{42}(3,2) = \mathrm{M}_{\mathrm{r2}/4}^{42}(3,1) = 0 \qquad (5.116)$$

and

$$\mathrm{A}_{\mathrm{r2}/4}^{42}(3,N) = 6.5N^3 + \mathrm{A}_{\mathrm{r2}/4}^{42}\left(3,\frac{N}{2}\right) + 56\mathrm{A}_{\mathrm{r2}/4}^{42}\left(3,\frac{N}{4}\right),\ N > 2,$$
$$\mathrm{A}_{\mathrm{r2}/4}^{42}(3,2) = 24,\ \mathrm{A}_{\mathrm{r2}/4}^{42}(3,1) = 0 \qquad (5.117)$$

Similarly, if the 3mult-3add scheme is considered, it can be shown that the corresponding expressions for the numbers of multiplications and additions are

$$\mathrm{M}^{33}_{\mathrm{r2/4}}(3,N) = 1.3125N^3 + \mathrm{M}^{33}_{\mathrm{r2/4}}\left(3,\frac{N}{2}\right) + 56\mathrm{M}^{33}_{\mathrm{r2/4}}\left(3,\frac{N}{4}\right),\ N>2,$$
$$\mathrm{M}^{33}_{\mathrm{r2/4}}(3,2) = \mathrm{M}^{33}_{\mathrm{r2/4}}(3,1) = 0 \qquad (5.118)$$

and

$$\mathrm{A}^{33}_{\mathrm{r2/4}}(3,N) = 6.9375N^3 + \mathrm{A}^{33}_{\mathrm{r2/4}}\left(3,\frac{N}{2}\right) + 56\mathrm{A}^{33}_{\mathrm{r2/4}}\left(3,\frac{N}{4}\right),\ N>2,$$
$$\mathrm{A}^{33}_{\mathrm{r2/4}}(3,2) = 24,\ \mathrm{A}^{33}_{\mathrm{r2/4}}(3,1) = 0 \qquad (5.119)$$

The expressions for the numbers of non-trivial multiplications and additions required by the proposed FHT algorithm are, respectively,

$$\mathrm{M}^{42}_{\mathrm{r2/4}}(3,N) = 1.75N^3 - 10.5N^2 + \mathrm{M}^{42}_{\mathrm{r2/4}}\left(3,\frac{N}{2}\right) + 56\mathrm{M}^{42}_{\mathrm{r2/4}}\left(3,\frac{N}{4}\right),\ N>4,$$
$$\mathrm{M}^{42}_{\mathrm{r2/4}}(3,4) = \mathrm{M}^{42}_{\mathrm{r2/4}}(3,2) = 0 \qquad (5.120)$$

and

$$\mathrm{A}^{42}_{\mathrm{r2/4}}(3,N) = 6.5N^3 - 3.5N^2 + \mathrm{A}^{42}_{\mathrm{r2/4}}\left(3,\frac{N}{2}\right) + 56\mathrm{A}^{42}_{\mathrm{r2/4}}\left(3,\frac{N}{4}\right),\ N>4,$$
$$\mathrm{A}^{42}_{\mathrm{r2/4}}(3,4) = 384,\ \mathrm{A}^{42}_{\mathrm{r2/4}}(3,2) = 24, \qquad (5.121)$$

in the case of the 4mult-2add scheme, and

$$\mathrm{M}^{33}_{\mathrm{r2/4}}(3,N) = 1.3125N^3 - 7N^2 + \mathrm{M}^{33}_{\mathrm{r2/4}}\left(3,\frac{N}{2}\right) + 56\mathrm{M}^{33}_{\mathrm{r2/4}}\left(3,\frac{N}{4}\right),\ N>4,$$
$$\mathrm{M}^{33}_{\mathrm{r2/4}}(3,4) = \mathrm{M}^{33}_{\mathrm{r2/4}}(3,2) = 0 \qquad (5.122)$$

and

$$A^{33}_{r2/4}(3,N) = 6.9375N^3 - 7N^2 + A^{33}_{r2/4}\left(3,\frac{N}{2}\right) + 56A^{33}_{r2/4}\left(3,\frac{N}{4}\right),\ N > 4,$$
$$A^{33}_{r2/4}(3,4) = 384,\ A^{33}_{r2/4}(3,2) = 24, \quad (5.123)$$

in the case of the 3mult-3add scheme.

To carry out a fair and comprehensive comparison, we also consider the numbers of multiplications and additions required by the one-butterfly, two-butterfly and multiple-butterfly implementations of the existing 3-D radix-$(2 \times 2 \times 2)$ FHT algorithms. The arithmetic complexities required by these algorithms along with those required by the proposed ones for various transform sizes $(N \times N \times N)$ are given in Tables 5.7-5.10. We see from these tables that the proposed FHT algorithms reduce substantially the number of multiplications as well as the number of additions. For example, if the 3mult-3add scheme is considered, savings of about more than 40% in the number of multiplications and about more than 10% in the number of additions can be easily achieved using the two-butterfly implementation of the proposed FHT algorithm over the two-butterfly implementation of the radix-$(2 \times 2 \times 2)$ algorithm.

The 3-D DHT can also be computed using the approach recently reported in [68]. This approach requires $\frac{1}{2}N^3\log_2 N - \frac{15}{14}N^3 + \frac{64}{14}$ non-trivial multiplications, $\frac{7}{2}N^3\log_2 N - \frac{29}{14}N^M + \frac{176}{14}$ non-trivial additions, and a special sequence reordering. This reordering requires $\left(\frac{1}{2} - \frac{1}{N}\right)N^3$ multiplications, $\left(\frac{1}{2} - \frac{1}{N}\right)N^3$ additions, $\left(\frac{1}{2} - \frac{1}{N}\right)N^3$ modulo operations, and $\left(\frac{1}{2} - \frac{1}{N}\right)N^3$ bit-shift operations. These requirements along with those of the proposed algorithm are given in Table 5.11. It is clear from this table that the two algorithms have comparable arithmetic complexities. However, as discussed below, the overall structural and computational complexity of the algorithm in [68] is larger than those of the proposed FHT algorithms.

Table 5.7: Number of arithmetic operations required for the computation of the 3-D DHT using the 4mult-2add scheme

	Radix-$(2\times2\times2)$ FHTs in [65], [66]						Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FHTs		
	Using one butterfly			Using two butterflies			Using two butterflies		
Transform size $N\times N\times N$	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point
$2^3\times2^3\times2^3$	5.25	11.62	16.87	3.5	10.75	14.25	1.96	9.98	11.95
$2^4\times2^4\times2^4$	7	15.5	22.5	5.25	14.62	19.87	3.52	13.76	17.29
$2^5\times2^5\times2^5$	8.75	19.37	28.12	7	18.5	25.5	3.91	16.95	20.87
$2^6\times2^6\times2^6$	10.5	23.25	33.75	8.75	22.37	31.12	5.32	20.66	25.98
$2^7\times2^7\times2^7$	12.25	27.12	39.37	10.5	26.25	36.75	5.84	23.92	29.76
$2^8\times2^8\times2^8$	14	31	45	12.25	30.12	42.37	7.13	27.57	34.70
$2^9\times2^9\times2^9$	15.75	34.87	50.62	14	34	48	7.75	30.87	38.62
$2^{10}\times2^{10}\times2^{10}$	17.5	38.75	56.25	15.75	37.87	53.62	8.96	34.48	43.44
$2^{11}\times2^{11}\times2^{11}$	19.25	42.62	61.87	17.5	41.75	59.25	9.65	37.82	47.48
$2^{12}\times2^{12}\times2^{12}$	21	46.5	67.5	19.25	45.62	64.87	10.80	41.40	52.20

Table 5.8: Number of arithmetic operations required for the computation of the 3-D DHT using the 3mult-3add scheme

	Radix-$(2\times2\times2)$ FHTs in [65], [66]						Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FHTs		
	Using one butterfly			Using two butterflies			Using two butterflies		
Transform size $N\times N\times N$	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point
$2^3\times2^3\times2^3$	3.93	12.93	16.87	2.62	11.62	14.25	1.47	10.47	11.95
$2^4\times2^4\times2^4$	5.25	17.25	22.5	3.93	15.93	19.87	2.64	14.64	17.29
$2^5\times2^5\times2^5$	6.56	21.56	28.12	5.25	20.25	25.5	2.93	17.93	20.87
$2^6\times2^6\times2^6$	7.87	25.87	33.75	6.56	24.56	31.12	3.99	21.99	25.98
$2^7\times2^7\times2^7$	9.18	30.18	39.37	7.87	28.87	36.75	4.38	25.38	29.76
$2^8\times2^8\times2^8$	10.5	34.5	45	9.18	33.18	42.37	5.35	29.35	34.70
$2^9\times2^9\times2^9$	11.81	38.81	50.62	10.5	37.5	48	5.81	32.71	38.62
$2^{10}\times2^{10}\times2^{10}$	13.12	43.12	56.25	11.81	41.81	53.62	6.72	36.72	43.44
$2^{11}\times2^{11}\times2^{11}$	14.43	47.43	61.87	13.12	46.12	59.25	7.24	40.24	47.48
$2^{12}\times2^{12}\times2^{12}$	15.75	51.75	67.5	14.43	50.43	64.87	8.10	44.10	52.20

Table 5.9: Number of non-trivial operations required for the computation of the 3-D DHT using the 4mult-2add scheme

	Radix-$(2\times2\times2)$ FHTs in [65], [66]			Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FHTs		
Transform size $N\times N\times N$	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point
$2^3\times2^3\times2^3$	0.4375	9.4375	9.8750	0.4375	9.4375	9.8750
$2^4\times2^4\times2^4$	1.5312	13.0938	14.6250	1.1484	12.7109	13.8594
$2^5\times2^5\times2^5$	2.9531	16.8594	19.8125	1.9482	16.2373	18.1855
$2^6\times2^6\times2^6$	4.5391	20.6797	25.2188	2.8344	19.5970	22.4314
$2^7\times2^7\times2^7$	6.2070	24.5273	30.7344	3.7270	23.1299	26.8569
$2^8\times2^8\times2^8$	7.9160	28.3887	36.3047	4.6549	26.5250	31.1799
$2^9\times2^9\times2^9$	9.6455	32.2568	41.9023	5.5725	30.0475	35.6199
$2^{10}\times2^{10}\times2^{10}$	11.3853	36.1284	47.5137	6.5094	33.4619	39.9712
$2^{11}\times2^{11}\times2^{11}$	13.1301	40.0017	53.1318	7.4344	36.9726	44.4070
$2^{12}\times2^{12}\times2^{12}$	14.8776	43.8759	58.7534	8.3724	40.3999	48.7723

Table 5.10: Number of non-trivial operations required for the computation of the 3-D DHT using the 3mult-3add scheme

	Radix-$(2\times2\times2)$ FHTs in [65], [66]			Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FHTs		
Transform size $N\times N\times N$	Mults. /point	Adds. /point	M.+A. /point	Mults. /point	Adds. /point	M.+A. /point
$2^3\times2^3\times2^3$	0.1094	9.7656	9.8750	0.4375	9.4375	9.8750
$2^4\times2^4\times2^4$	0.8203	13.8047	14.6250	0.9297	12.9297	13.8594
$2^5\times2^5\times2^5$	1.8320	17.9805	19.8125	1.5928	16.5928	18.1855
$2^6\times2^6\times2^6$	2.9941	22.2246	25.2188	2.2157	20.2157	22.4314
$2^7\times2^7\times2^7$	4.2314	26.5029	30.7344	2.9285	23.9285	26.8569
$2^8\times2^8\times2^8$	5.5063	30.7983	36.3047	3.5899	27.5899	31.1799
$2^9\times2^9\times2^9$	6.8000	35.1023	41.9023	4.3100	31.3100	35.6199
$2^{10}\times2^{10}\times2^{10}$	8.1031	39.4105	47.5137	4.9856	34.9856	39.9712
$2^{11}\times2^{11}\times2^{11}$	9.4109	43.7209	53.1318	5.7035	38.7035	44.4070
$2^{12}\times2^{12}\times2^{12}$	10.7211	48.0323	58.7534	6.3861	42.3861	48.7723

Table 5.11: Number of non-trivial operations required for the computation of the 3-D DHT using the 4mult-2add scheme

	Polynomial transform algorithm in [68]					Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FHTs		
Transform Size $N\times N\times N$	Mults. /point	Adds. /point	M.+A. /point	Modulo /point	Bit-shift /point	Mults. /point	Adds. /point	M.+A. /point
$2^3\times2^3\times2^3$	0.81	8.82	9.64	0.37	0.37	0.43	9.43	9.87
$2^4\times2^4\times2^4$	1.36	12.36	13.73	0.43	0.43	1.14	12.71	13.85
$2^5\times2^5\times2^5$	1.89	15.89	17.79	0.46	0.46	1.94	16.23	18.18
$2^6\times2^6\times2^6$	2.41	19.41	21.82	0.48	0.48	2.83	19.59	22.43
$2^7\times2^7\times2^7$	2.92	22.92	25.84	0.49	0.49	3.72	23.12	26.85
$2^8\times2^8\times2^8$	3.42	26.42	29.84	0.49	0.49	4.65	26.52	31.17
$2^9\times2^9\times2^9$	3.92	29.92	33.85	0.49	0.49	5.57	30.04	35.61
$2^{10}\times2^{10}\times2^{10}$	4.42	33.42	37.85	0.49	0.49	6.50	33.46	39.97
$2^{11}\times2^{11}\times2^{11}$	4.92	36.92	41.85	0.49	0.49	7.43	36.97	44.40
$2^{12}\times2^{12}\times2^{12}$	5.42	40.42	45.85	0.49	0.49	8.37	40.39	48.77

The popularity of the DHT is mainly due to the fact that the forward and inverse transforms can be computed using only a single FHT algorithm. Unfortunately, this very useful property of the DHT in requiring only a single algorithm is lost when we consider the approach reported in [68], since the algorithm requires the implementation of two different complicated algorithms, one for computing the 1-D type-II-DWTs and the other for the 3-D polynomial transforms. Moreover, the approach in [68] requires a special sequence reordering, thus necessitating extra multiplications, additions, modulo operations, and bit-shift operations. Further, these time-consuming overhead operations contribute significantly to an increase in the overall structural complexity of the algorithm. Another drawback of this approach is that it does not have the butterfly-style and in-place computation properties, thereby requiring a large memory. Due to these drawbacks, the algorithm in [68] would not be attractive for software or hardware implementation. However, the proposed FHT algorithms have more regular and simple structures using butterfly-style and in-place computations.

5.3.3.2 Data Transfers

We first give a brief implementation scheme of the proposed radix-$(2\times2\times2)/(4\times4\times4)$ DIF FHT algorithm by considering the implementation of the butterfly given by (5.95) and (5.87). For a given combination of (k_1,k_2,k_3), by reading eight points from the off-chip memory according to (5.63) to compute the operation given by (5.62), we obtain the eight points given by (5.64). The first point $y_{000}(k_1,k_2,k_3)$ of (5.64) is returned to the off-chip memory, whereas the other seven points are kept in the on-chip memory since they are used according to (5.91) by the general sub-butterfly given by (5.87). These operations are repeated 16 times to complete the computation of the operation given by (5.95). The seven 16-point results kept in the on-chip memory are used to process the general sub-butterfly. This completes the process of the butterfly. For all the combinations of (k_1,k_2,k_3), $0\le k_1\le\frac{N}{8}-1,\ 0\le k_2,k_3\le\frac{N}{4}-1$, the results returned to the off-chip memory are grouped to form the input sequence $y_{000}(k_1,k_2,k_3)$ of the $\left(\frac{N}{2}\times\frac{N}{2}\times\frac{N}{2}\right)$-point 3-D DHT given by (5.54). The 56 input sequences of the $\left(\frac{N}{4}\times\frac{N}{4}\times\frac{N}{4}\right)$-point 3-D DHTs given by (5.86) are formed by grouping appropriately the results of the output vector given by (5.87). This completes the first stage of the proposed decomposition.

Then, it can be shown that the expression for the number of data transfers, not including the read operations for the twiddle factors, required by the proposed DIF FHT algorithm is given by

$$\mathrm{D}_{\mathrm{r2/4}}(3,N)=N^3+\mathrm{D}_{\mathrm{r2/4}}\left(3,\frac{N}{2}\right)+56\mathrm{D}_{\mathrm{r2/4}}\left(3,\frac{N}{4}\right),\ N>2,$$
$$\mathrm{D}_{\mathrm{r2/4}}(3,2)=8,\ \mathrm{D}_{\mathrm{r2/4}}(3,1)=0 \qquad (5.124)$$

The corresponding expression for the radix-$(2\times2\times2)$ FHT algorithm is given by

$$\mathrm{D}_{\mathrm{r2}}(3,N)=N^3\log_2 N \qquad (5.125)$$

Table 5.12: Comparison of the number of data transfers

Transform size $N \times N \times N$	Radix-$(2 \times 2 \times 2)$ FHTs $D_{r2}(3,N)/N^3$	Proposed radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ FHTs $D_{r2/4}(3,N)/N^3$	Savings (%)
$2^3 \times 2^3 \times 2^3$	3	2.0156	32.81
$2^4 \times 2^4 \times 2^4$	4	2.2363	44.09
$2^5 \times 2^5 \times 2^5$	5	3.0432	39.13
$2^6 \times 2^6 \times 2^6$	6	3.3372	44.38
$2^7 \times 2^7 \times 2^7$	7	4.08	41.71
$2^8 \times 2^8 \times 2^8$	8	4.43	44.62
$2^9 \times 2^9 \times 2^9$	9	5.1237	43.07
$2^{10} \times 2^{10} \times 2^{10}$	10	5.5167	44.83
$2^{11} \times 2^{11} \times 2^{11}$	11	6.1728	43.88
$2^{12} \times 2^{12} \times 2^{12}$	12	6.5988	45.01

The numbers of data transfers for the two FHT algorithms are compared in Table 5.12, for various values of N. It is clear from this table that the radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ FHT algorithm requires about 40% less data transfer operations than that required in the radix-$(2 \times 2 \times 2)$ FHT algorithm. Hence, similar reduction is also obtained in the case of the address generation.

5.3.3.3 Twiddle factors

The general butterfly of the radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ DIF FHT algorithm requires 112 twiddle factors (cosine and sine), whereas that of the radix-$(2 \times 2 \times 2)$ FHT algorithm requires 14 twiddle factors. Therefore, it is seen that the expression for the number of twiddle factors required by the two-butterfly implementation of the radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ FHT algorithm is given by

$$T^{42}_{r2/4}(3,N) = 112\frac{N^3}{128} + T^{42}_{r2/4}\left(3,\frac{N}{2}\right) + 56T^{42}_{r2/4}\left(3,\frac{N}{4}\right),\ N > 2,$$
$$T^{42}_{r2/4}(3,2) = T^{42}_{r2/4}(3,1) = 0 \qquad (5.126)$$

Table 5.13: Comparison of the number of twiddle factor evaluations or accesses to the lookup table

	Radix-$(2\times2\times2)$ FHTs		Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FHTs	Savings (%)	
Transform size	Using one butterfly	Using two butterflies	Using two butterflies		
$N\times N\times N$	$T_{r2}^{42}(3,N)/N^3$ (A)	$T_{r2}^{42}(3,N)/N^3$ (B)	$T_{r2/4}^{42}(3,N)/N^3$ (C)	in C over A	in C over B
$2^3\times2^3\times2^3$	2.625	1.75	0.9844	62.49	43.74
$2^4\times2^4\times2^4$	3.5	2.625	1.7637	49.61	32.71
$2^5\times2^5\times2^5$	4.375	3.5	1.9568	55.27	44.09
$2^6\times2^6\times2^6$	5.25	4.375	2.6628	49.28	39.13
$2^7\times2^7\times2^7$	6.125	5.25	2.92	52.32	44.38
$2^8\times2^8\times2^8$	7	6.125	3.57	49.00	41.71
$2^9\times2^9\times2^9$	7.875	7	3.8763	50.77	44.62
$2^{10}\times2^{10}\times2^{10}$	8.75	7.875	4.4833	48.76	43.06
$2^{11}\times2^{11}\times2^{11}$	9.625	8.75	4.8272	49.84	44.83
$2^{12}\times2^{12}\times2^{12}$	10.5	9.625	5.4012	48.55	43.88

The corresponding expression for the one-butterfly ($s=0$) or two-butterfly ($s=1$) implementation of the radix-$(2\times2\times2)$ FHT algorithm is given by

$$T_{r2}^{42}(3,N)=14\frac{N^3}{16}\log_2\frac{N}{2^s} \tag{5.127}$$

The numbers of twiddle factor evaluations or accesses to the lookup table for the two algorithms are compared in Table 5.13. It is clear from this table that savings of about 40% in the number of twiddle factor evaluations or accesses to the lookup table can be achieved using the proposed radix-$(2\times2\times2)/(4\times4\times4)$ FHT algorithms over the one-butterfly and two-butterfly implementations of the radix-$(2\times2\times2)$ FHT algorithms. Note that, when the lookup table is used, similar savings are obtained by the proposed FHT algorithms in the address generation for reading the twiddle factors.

5.4 Summary

In this chapter, new radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ FFT and FHT algorithms for computing, respectively, the 3-D DFT and DHT have been developed using DIF and/or DIT decompositions. The proposed algorithms are based on the radix-2/4 approach that has been introduced by using an appropriate index mapping to carry out the decomposition of the 3-D DFT or DHT in two distinct steps. In the first step, the decomposition has been carried out by using radix-$(2 \times 2 \times 2)$ index maps, and in the second step further decomposition has been carried out by introducing radix-$(4 \times 4 \times 4)$ index maps. The introduction of these two steps and the use of the Kronecker product enable us to develop 3-D FFT and FHT algorithms whose butterflies are characterized by simple closed-form expressions. It has been shown that the proposed 3-D FFT (or FHT) algorithm substantially reduces the number of arithmetic operations, data transfers, address generations, and twiddle factor evaluations or accesses to the lookup table compared to that required by the existing 3-D radix-$(2 \times 2 \times 2)$ FFT (or FHT) algorithm. The proposed 3-D FFT and FHT algorithms have significantly reduced structural complexities compared to the corresponding polynomial transform algorithms.

Chapter 6

A New radix-2/8 Approach

6.1 Introduction

It has been shown in the literature that the 1-D and 2-D FFT and FHT algorithms, based on the radix-2/4 approach, are more efficient than the other existing ones [11], [17], [29], [30], [45], [56], [57], [64]. In Chapter 5, we have shown that the radix-2/4 approach is superior to the other existing approaches even in the 3-D case. The efficiency of this approach is obtained by taking advantage of the lowest structural complexity provided by the radix-2 approach and reduced computational complexity provided by the radix-4 approach. Even though the radix-2/4 algorithms are the simplest algorithms having the lowest number of arithmetic operations, there is still a scope for further improvement in terms of the number of data transfers, address generations, and twiddle factor evaluations or accesses to the lookup table, which also significantly increase the complexity of the algorithms.

In this chapter, the problem of reducing the complexity in the computation of the 1-D DFT and DHT of lengths $N = q \times 2^m$, where q and m are integers is addressed, and a new radix-2/8 approach is proposed by a simultaneous use of the radix-2 and radix-8 approaches [120]-[123]. This new approach takes advantage offered by the radix-2 approach discussed

in Chapter 1 and that offered by the radix-8 approach discussed in Chapter 4. This approach will be extended to 2-D and 3-D DFTs in the next chapter.

6.2 Radix-2/8 Approach for the Computation of Length-$q \times 2^m$ 1-D DFT

The radix-2/4 approach of Duhamel-Hollmann differs from that of Cooley-Tukey in the decomposition process. For example, in the Cooley-Tukey radix-2 FFT, radix-2 index maps are used to decompose both the even and odd indexed terms of the DFT, while in the radix-2/4 FFT, radix-4 index maps are used in the decomposition of the odd indexed terms. In other words, the radix-2/4 FFT consists of dividing a length-N DFT into one length-$N/2$ DFT and two length-$N/4$ DFTs in the first stage and repeating successively the process until the size is reduced to a 2-point DFT without twiddle factors. This significantly reduces the number of arithmetic operations as compared with the fixed-radix Cooley-Tukey FFT [4], [36], [32]. In 2000, Grigoryan and Agaian [19] proposed another algorithm that reduces the number of arithmetic operations for DFTs of lengths larger than 256 at the expense of a more complicated structure compared to that of the radix-2/4 FFT algorithm; however, it is computationally less efficient for smaller lengths. Even though the radix-2/4 FFT algorithm is the simplest algorithm having the lowest number of arithmetic operations, the number of data transfers, address generations, and twiddle factor evaluations or accesses to the lookup table is still high.

The radix-2/4 FFT algorithm was mainly proposed for length-2^m DFTs. In order to explore the performance of the split-radix approach in the other DFTs, the idea of the radix-2/4 FFT proposed in [11] has been extended to the length-p^m [14] and length-$q \times 2^m$ DFTs [17]. Specifically, the algorithm in [17] consists of dividing a length-N DFT into one length-$N/2$ DFT and two length-$N/4$ DFTs in the first stage and repeating successively the

process until the size is reduced to a $2q$-point or a q-point DFT. It has been shown that this algorithm reduces the number of arithmetic operations compared with other reported algorithms [2], [5], [6], [13], [14].

In order to address the problem of reducing the number of data transfers, address generations, and twiddle factor evaluations or accesses to the lookup table, we explore the possibility of using radix-8 index maps in the decomposition process. In [11], the authors claim that the computation of the odd terms of the split-radix DFT through a radix-8 does not improve the algorithm. In [124], the author states "It can easily be checked out that a $\frac{2}{8}$-split-radix algorithm is worse than a $\frac{2}{4}$-split-radix algorithm from an arithmetic complexity point of view". In 2001, Takahashi [20] directly used the radix-8 in the decomposition of the odd terms. This, of course, led to an algorithm with an increased arithmetic complexity.

In this section, we develop a new radix-2/8 FFT algorithm [120], [121] by an appropriate simultaneous use of the radix-2 and radix-8 index maps and show that, for the case of the DFT of length $N = 2^m$, we can substantially reduce the number of data transfers, address generations and twiddle factor evaluations or accesses to the lookup table, without increasing the number of arithmetic operations from that of the radix-2/4 FFT algorithm. We also show that these results hold for the case of the DFT of length $N = q \times 2^m$.

6.2.1 Proposed Radix-2/8 DIF FFT Algorithm

From (3.1), the 1-D DFT of length N is given by

$$\widetilde{X}(n) = \sum_{k=0}^{N-1} \widetilde{x}(k) W_N^{nk}, \ 0 \leq n \leq N-1 \tag{6.1}$$

The sequence length N is assumed to be $q \times 2^m$, where q is an odd integer and $m > 0$. Let us first consider the case when $m = 1$. Then, the decomposition of the DFT coefficients $\widetilde{X}(n)$

in (6.1) into even-indexed and odd-indexed terms provides

$$\widetilde{X}(2n) = \sum_{k=0}^{q-1} (\widetilde{x}(k) + \widetilde{x}(k+q)) W_q^{nk},\ n = 0, 1, ..., q-1 \tag{6.2}$$

$$\widetilde{X}((2n+q) \bmod 2q) = \sum_{k=0}^{q-1} (\widetilde{x}(k) - \widetilde{x}(k+q)) (-1)^k W_q^{nk},\ n = 0, 1, ..., q-1 \tag{6.3}$$

This decomposition consists of dividing the length-$2q$ DFT into two length-q DFTs given by (6.2) and (6.3). In a similar way, when $m = 2$, the decomposition of (6.1) provides

$$\widetilde{X}(2n) = \sum_{k=0}^{2q-1} (\widetilde{x}(k) + \widetilde{x}(k+2q)) W_{2q}^{nk},\ n = 0, 1, ..., (2q-1) \tag{6.4}$$

for the even-indexed terms, and

$$\widetilde{X}((4n+q) \bmod 4q) = \sum_{k=0}^{q-1} (\widetilde{a}(k) + (-j)^q \widetilde{a}(k+q)) (-j)^k W_q^{nk},\ n = 0, 1, ..., q-1 \tag{6.5}$$

$$\widetilde{X}((4n+3q) \bmod 4q) = \sum_{k=0}^{q-1} (\widetilde{a}(k) + (j)^q \widetilde{a}(k+q)) (j)^k W_q^{nk},\ n = 0, 1, ..., q-1 \tag{6.6}$$

for the odd-indexed terms, where $\widetilde{a}(k) = \widetilde{x}(k) - \widetilde{x}(k+2q)$. The length-$4q$ DFT is thus decomposed into one length-$2q$ DFT given by (6.4) and two length-q DFTs given by (6.5) and (6.6). We now propose the following new decomposition of (6.1) so that we can compute a length-$q \times 2^m$ ($m > 2$) DFT using a mixture of radix-2 and radix-8 index maps.

$$\widetilde{X}(2n) = \sum_{k=0}^{N/2-1} \widetilde{a}_e(k) W_{N/2}^{nk},\ n = 0, 1, ..., (N/2-1) \tag{6.7}$$

for the even-indexed terms, and

$$\widetilde{X}((8n+q) \bmod N) = \sum_{k=0}^{N/8-1} \left(\widetilde{b}_e(k) + e^{-j\frac{q\pi}{4}}\widetilde{b}_e(k+N/8)\right) W_N^{qk} W_{N/8}^{nk},$$
$$n = 0, 1, ..., (N/8-1) \qquad (6.8)$$

$$\widetilde{X}((N+8n-3q) \bmod N) = \sum_{k=0}^{N/8-1} \left(\widetilde{b}_e(k) - e^{-j\frac{q\pi}{4}}\widetilde{b}_e(k+N/8)\right) W_N^{-3qk} W_{N/8}^{nk},$$
$$n = 0, 1, ..., (N/8-1) \qquad (6.9)$$

$$\widetilde{X}((8n+3q) \bmod N) = \sum_{k=0}^{N/8-1} \left(\widetilde{b}_o(k) + e^{-j\frac{3q\pi}{4}}\widetilde{b}_o(k+N/8)\right) W_N^{3qk} W_{N/8}^{nk},$$
$$n = 0, 1, ..., (N/8-1) \qquad (6.10)$$

$$\widetilde{X}((N+8n-q) \bmod N) = \sum_{k=0}^{N/8-1} \left(\widetilde{b}_o(k) - e^{-j\frac{3q\pi}{4}}\widetilde{b}_o(k+N/8)\right) W_N^{-qk} W_{N/8}^{nk},$$
$$n = 0, 1, ..., (N/8-1) \qquad (6.11)$$

for the odd-indexed terms. The sequences $\widetilde{b}_e(k)$ in (6.8) and (6.9), and $\widetilde{b}_o(k)$ in (6.10) and (6.11) can be expressed in a matrix form as

$$\begin{bmatrix} \widetilde{b}_e(k) \\ \widetilde{b}_o(k) \end{bmatrix} = \mathbf{H}_2\mathbf{J} \begin{bmatrix} \widetilde{a}_o(k) \\ \widetilde{a}_o(k+N/4) \end{bmatrix} \qquad (6.12)$$

where

$$\mathbf{J} = \begin{bmatrix} 1 & 0 \\ 0 & (-j)^q \end{bmatrix}$$

The sequences $\widetilde{a}_e(k)$ in (6.7) and $\widetilde{a}_o(k)$ in (6.12) can be obtained by

$$\begin{bmatrix} \widetilde{a}_e(k) \\ \widetilde{a}_o(k) \end{bmatrix} = \mathbf{H}_2 \begin{bmatrix} \widetilde{x}(k) \\ \widetilde{x}(k+N/2) \end{bmatrix} \tag{6.13}$$

which can be considered as the first step of the new decomposition. This first step is processed by applying a second-order Hadamard matrix $\mathbf{H}_2$ to the input sequence $\widetilde{x}(k)$ without requiring the twiddle factors. Since the operation of multiplication of $(a+jb)$ by $(-j)$ converts the complex number $(a+jb)$ to the complex number $(b-ja)$, the matrix $\mathbf{J}$ introduced in (6.12) can be implemented within the address generation process, and hence, $\mathbf{J}$ does not require any additional processing time.

Let us use this decomposition to efficiently compute the length-$8q$ DFT with a minimum number of arithmetic operations. We let $N = 8q$ in (6.7)-(6.11). Then, (6.7) reduces to

$$\widetilde{X}(2n) = \sum_{k=0}^{4q-1} \widetilde{a}_e(k) W_{4q}^{nk}, \; n = 0, 1, ..., (4q-1) \tag{6.14}$$

By adding (6.9) and (6.8) and dividing the result by two, we can form a new sequence $\widetilde{g}_a(n)$ given by

$$\widetilde{g}_a(n) = \sum_{k=0}^{q-1} \left[e^{-j\frac{k\pi}{4}} \frac{\left(1+(-1)^k\right)}{2} \widetilde{b}_e(k) + e^{-j\frac{(q+k)\pi}{4}} \frac{\left(1-(-1)^k\right)}{2} \widetilde{b}_e(k+q) \right] W_q^{nk}, \quad n = 0, 1, ..., q-1 \tag{6.15}$$

Similarly, subtracting (6.9) from (6.8) and dividing by two, we can form another sequence $\widetilde{g}_s(n)$ given by

$$\widetilde{g}_s(n) = \sum_{k=0}^{q-1} \left[e^{-j\frac{k\pi}{4}} \frac{\left(1-(-1)^k\right)}{2} \widetilde{b}_e(k) + e^{-j\frac{(q+k)\pi}{4}} \frac{\left(1+(-1)^k\right)}{2} \widetilde{b}_e(k+q) \right] W_q^{nk}, \quad n = 0, 1, ..., q-1 \tag{6.16}$$

Since q is an odd integer and k takes only even or odd integer values, the quantity within the brackets in (6.15) reduces either to $(-j)^{k/2}\widetilde{b}_e(k)$ or $(-j)^{(q+k)/2}\widetilde{b}_e(k+q)$. Also, the corresponding quantity in (6.16) reduces to $\frac{\sqrt{2}}{2}(\pm 1 \pm j)\widetilde{b}_e(k)$ or $\frac{\sqrt{2}}{2}(\pm 1 \pm j)\widetilde{b}_e(k+q)$. Hence, the length-$q$ DFT given by (6.15) does not require to perform any pre-additions or pre-multiplications on the sequence $\widetilde{b}_e(k)$ or $\widetilde{b}_e(k+q)$; however, (6.16) can be considered as a length-q DFT scaled by a real factor $\frac{\sqrt{2}}{2}$, whose input sequence is either $(\pm 1 \pm j)\widetilde{b}_e(k)$ or $(\pm 1 \pm j)\widetilde{b}_e(k+q)$. The final transform output sequences corresponding to (6.8) and (6.9) are obtained, respectively, by

$$\widetilde{X}\left((8n+q) \bmod 8q\right) = \widetilde{g}_a(n) + \widetilde{g}_s(n),\ n = 0, 1, ..., q-1 \tag{6.17}$$

and

$$\widetilde{X}\left((8n+5q) \bmod 8q\right) = \widetilde{g}_a(n) - \widetilde{g}_s(n),\ n = 0, 1, ..., q-1 \tag{6.18}$$

Similarly, using (6.10) and (6.11) we can form two more new sequences given by

$$\widetilde{h}_a(n) = \sum_{k=0}^{q-1}\left[e^{-j\frac{3k\pi}{4}}\frac{\left(1+(-1)^k\right)}{2}\widetilde{b}_o(k) + e^{-j\frac{3(q+k)\pi}{4}}\frac{\left(1-(-1)^k\right)}{2}\widetilde{b}_o(k+q)\right]W_q^{nk},$$
$$n = 0, 1, ..., q-1 \tag{6.19}$$

$$\widetilde{h}_s(n) = \sum_{k=0}^{q-1}\left[e^{-j\frac{3k\pi}{4}}\frac{\left(1-(-1)^k\right)}{2}\widetilde{b}_o(k) + e^{-j\frac{3(q+k)\pi}{4}}\frac{\left(1+(-1)^k\right)}{2}\widetilde{b}_o(k+q)\right]W_q^{nk},$$
$$n = 0, 1, ..., q-1 \tag{6.20}$$

For even values of k, the quantities within the brackets in (6.19) and (6.20) reduce to $(j)^{k/2}\widetilde{b}_o(k)$ and $\frac{\sqrt{2}}{2}(\pm 1 \pm j)\widetilde{b}_o(k+q)$, respectively. The corresponding quantities for odd values of k reduce to $(j)^{(q+k)/2}\widetilde{b}_o(k+q)$ and $\frac{\sqrt{2}}{2}(\pm 1 \pm j)\widetilde{b}_o(k)$. Therefore, the computation of (6.19) and (6.20) is similar to that of (6.15) and (6.16), respectively. The final transform

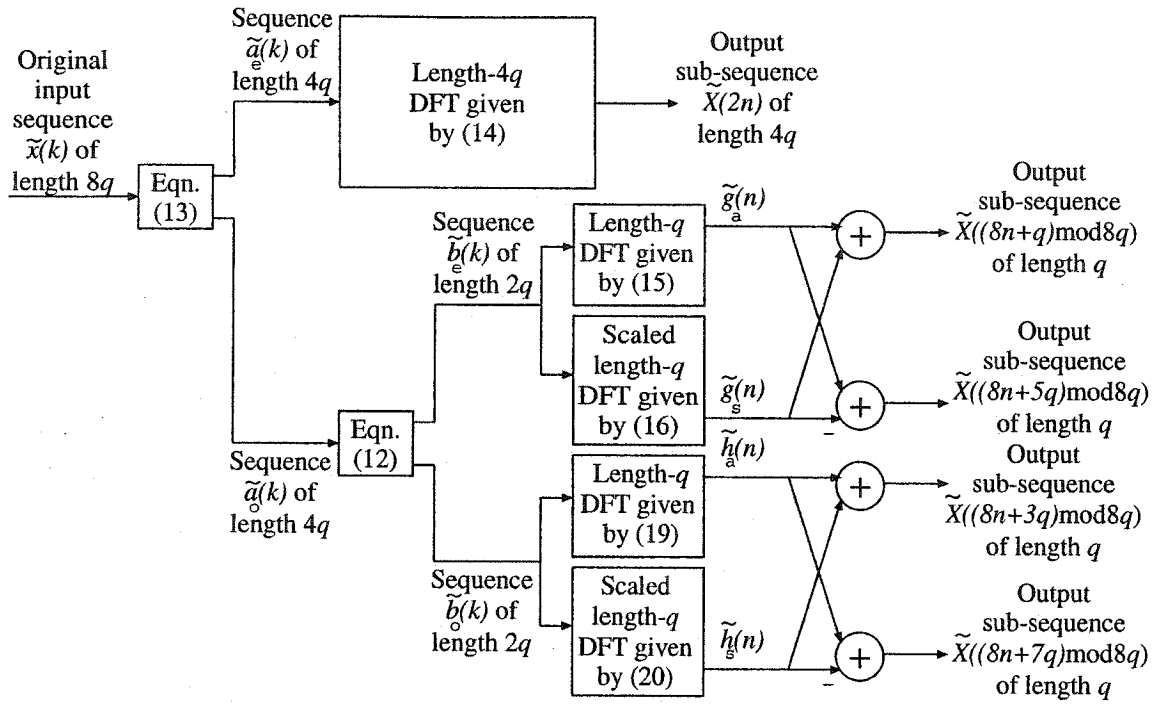

Figure 6.1: Decomposition diagram of a length-$8q$ DFT.

output sequences corresponding to (6.10) and (6.11) are obtained, respectively, by

$$\tilde{X}((8n+3q) \bmod 8q) = \tilde{h}_a(n) + \tilde{h}_s(n), \; n = 0, 1, ..., q-1 \tag{6.21}$$

and

$$\tilde{X}((8n+7q) \bmod 8q) = \tilde{h}_a(n) - \tilde{h}_s(n), \; n = 0, 1, ..., q-1 \tag{6.22}$$

Thus, the computation of a length-$8q$ DFT may be thought of as having being decomposed into five DFTs, as given by (6.14), (6.17), (6.18), (6.21) and (6.22). This decomposition is illustrated in Fig. 6.1. It is clear from this figure that the computation of a length-$8q$ DFT requires one length-$4q$ DFT, two length-q DFTs, two scaled length-q DFTs and some extra additions.

Let us now consider the general case when $N > 8q$. We denote the input sequences

of the four length-$N/8$ DFTs given by (6.8)-(6.11), respectively, by $\widetilde{c}_1(k)$, $\widetilde{c}_{-3}(k)$, $\widetilde{c}_3(k)$ and $\widetilde{c}_{-1}(k)$. After certain mathematical transformations, these input sequences may be expressed in a compact form as

$$\begin{bmatrix} \widetilde{c}_1(k) \\ \widetilde{c}_{-3}(k) \\ \widetilde{c}_3(k) \\ \widetilde{c}_{-1}(k) \end{bmatrix} = \mathbf{A}_k \begin{bmatrix} \widetilde{b}_e(k) \\ \widetilde{b}_e(k+N/8) \\ \widetilde{b}_o(k) \\ \widetilde{b}_o(k+N/8) \end{bmatrix}, \quad k = 0, 1, ..., (N/8-1) \tag{6.23}$$

where the matrix $\mathbf{A}_k$ is given by

$$\mathbf{A}_k = \mathbf{E}_k \mathbf{A} \tag{6.24}$$

In (6.24), the twiddle factor matrix $\mathbf{E}_k$ is a diagonal matrix given by

$$\mathbf{E}_k = \begin{bmatrix} W_N^{qk} & 0 & 0 & 0 \\ 0 & W_N^{-3qk} & 0 & 0 \\ 0 & 0 & W_N^{3qk} & 0 \\ 0 & 0 & 0 & W_N^{-qk} \end{bmatrix} \tag{6.25}$$

and the constant matrix $\mathbf{A}$ is given by

$$\mathbf{A} = (\mathbf{I}_2 \otimes \mathbf{H}_2)\,\mathbf{D}\,(\mathbf{I}_2 \otimes \mathbf{C}) \tag{6.26}$$

where

$$\mathbf{D} = \begin{bmatrix} \mathbf{I}_2 & 0 \\ 0 & \mathbf{J} \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & e^{-jq\pi/4} \end{bmatrix},$$

Finally, the general butterfly of the proposed radix-2/8 FFT algorithm is constructed using

$$\begin{bmatrix} \widetilde{a}_e(k) \\ \widetilde{a}_e(k+N/8) \\ \widetilde{a}_e(k+N/4) \\ \widetilde{a}_e(k+3N/8) \\ \widetilde{a}_o(k) \\ \widetilde{a}_o(k+N/8) \\ \widetilde{a}_o(k+N/4) \\ \widetilde{a}_o(k+3N/8) \end{bmatrix} = (\mathbf{H}_2 \otimes \mathbf{I}_2 \otimes \mathbf{I}_2) \begin{bmatrix} \widetilde{x}(k) \\ \widetilde{x}(k+N/8) \\ \widetilde{x}(k+N/4) \\ \widetilde{x}(k+3N/8) \\ \widetilde{x}(k+N/2) \\ \widetilde{x}(k+5N/8) \\ \widetilde{x}(k+3N/4) \\ \widetilde{x}(k+7N/8) \end{bmatrix}, \; k=0, 1, ..., (N/8-1) \quad (6.27)$$

$$\begin{bmatrix} \widetilde{b}_e(k) \\ \widetilde{b}_e(k+N/8) \\ \widetilde{b}_o(k) \\ \widetilde{b}_o(k+N/8) \end{bmatrix} = (\mathbf{H}_2 \otimes \mathbf{I}_2)(\mathbf{J} \otimes \mathbf{I}_2) \begin{bmatrix} \widetilde{a}_o(k) \\ \widetilde{a}_o(k+N/8) \\ \widetilde{a}_o(k+N/4) \\ \widetilde{a}_o(k+3N/8) \end{bmatrix}, \; k=0, 1, ..., (N/8-1) \quad (6.28)$$

and the general sub-butterfly given by (6.23). The flowgraph of this general butterfly is illustrated in Fig. 6.2.

It is clear that for a sequence of length N, the required number of butterflies to perform the first stage of this new decomposition is $\frac{N}{8}$. The index k in Fig. 6.2 varies from 0 to $\left(\frac{N}{8}-1\right)$ as indicated in (6.27), (6.28) and (6.23). The appearance of the parameter q in the exponents of the elements of the twiddle factor matrix (6.25), allows us to define $2q$ special butterflies that can be indexed by $k = p\frac{N}{16q}$, where $p = 0, 1, ..., (2q-1)$. Then, (6.25) can be expressed as

$$\mathbf{E}_{p\frac{N}{16q}} = \left(\mathbf{I}_2 \otimes \mathbf{J}'\right)(\mathbf{E} \otimes \mathbf{I}_2), \; p = 0, 1, ..., (2q-1) \quad (6.29)$$

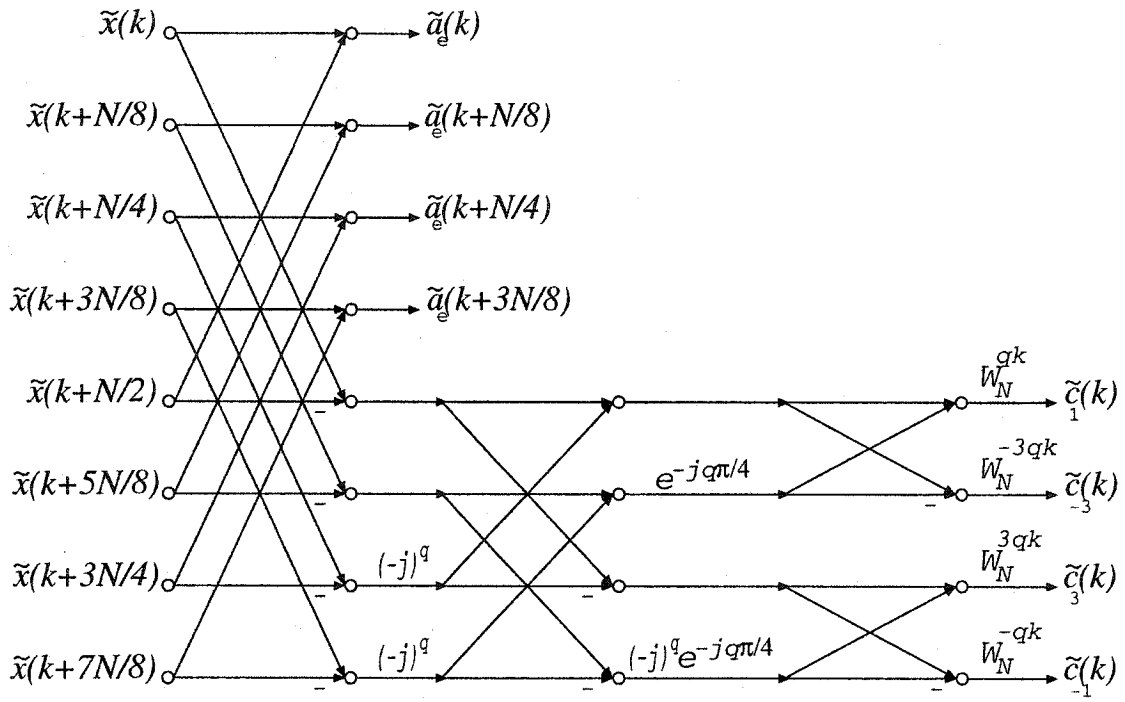

Figure 6.2: General butterfly of the proposed radix-2/8 FFT algorithm.

where

$$\mathbf{J}' = \begin{bmatrix} 1 & 0 \\ 0 & (j)^p \end{bmatrix}$$

and

$$\mathbf{E} = \begin{bmatrix} e^{-j\frac{p\pi}{8}} & 0 \\ 0 & e^{-j\frac{3p\pi}{8}} \end{bmatrix}.$$

By substituting (6.29) in (6.24), using the fact that $(\mathbf{E} \otimes \mathbf{I}_2)(\mathbf{I}_2 \otimes \mathbf{H}_2) = (\mathbf{I}_2 \otimes \mathbf{H}_2)(\mathbf{E} \otimes \mathbf{I}_2)$, and combining the twiddle factor matrices $(\mathbf{E} \otimes \mathbf{I}_2)$ and $\mathbf{D}(\mathbf{I}_2 \otimes \mathbf{C})$, the matrix $\mathbf{A}_{p\frac{N}{16q}}$ can be expressed as

$$\mathbf{A}_{p\frac{N}{16q}} = (\mathbf{I}_2 \otimes \mathbf{J}')(\mathbf{I}_2 \otimes \mathbf{H}_2)\mathbf{T}, \quad p = 0, 1, ..., (2q-1) \tag{6.30}$$

where

$$\mathbf{T} = \begin{bmatrix} e^{-j\frac{p\pi}{8}} & 0 & 0 & 0 \\ 0 & e^{-j\frac{(2q+p)\pi}{8}} & 0 & 0 \\ 0 & 0 & e^{-j\frac{3p\pi}{8}} & 0 \\ 0 & 0 & 0 & (-j)^q e^{-j\frac{(2q+3p)\pi}{8}} \end{bmatrix}, \; p = 0, 1, ..., (2q-1) \qquad (6.31)$$

It can be shown that four real multiplications and four real additions are introduced by the matrix $\mathbf{T}$ in each of the q special butterflies corresponding to the even values of p, and four complex multiplications are introduced in each of the other q special butterflies that correspond to the odd values of p. The number of operations to process these special butterflies is less than that needed to process the general butterfly, as will be seen in Section 6.2.2.

We now summarize the scheme of the proposed radix-2/8 FFT algorithm for computing a length-$q \times 2^m$ DFT. By performing $\left(\frac{N}{8} - 2q\right)$ general butterflies and $2q$ special butterflies defined above, the initial input sequence $\tilde{x}(k)$ of length N is decomposed into five sub-sequences. The first sub-sequence of length $N/2$ is constructed by the first four outputs obtained from (6.27). The other four sub-sequences each of length $N/8$ represent the four outputs of the sub-butterfly given by (6.23). This process is repeated successively for each of the new resulting sub-sequences, until the size is reduced to a $4q$- , $2q$- or q-point DFT. This scheme is illustrated for the first stage in Fig. 6.3. Note that, for simplicity, the gains (weights) are omitted in Fig. 6.3 and can easily be obtained from Fig. 6.2.

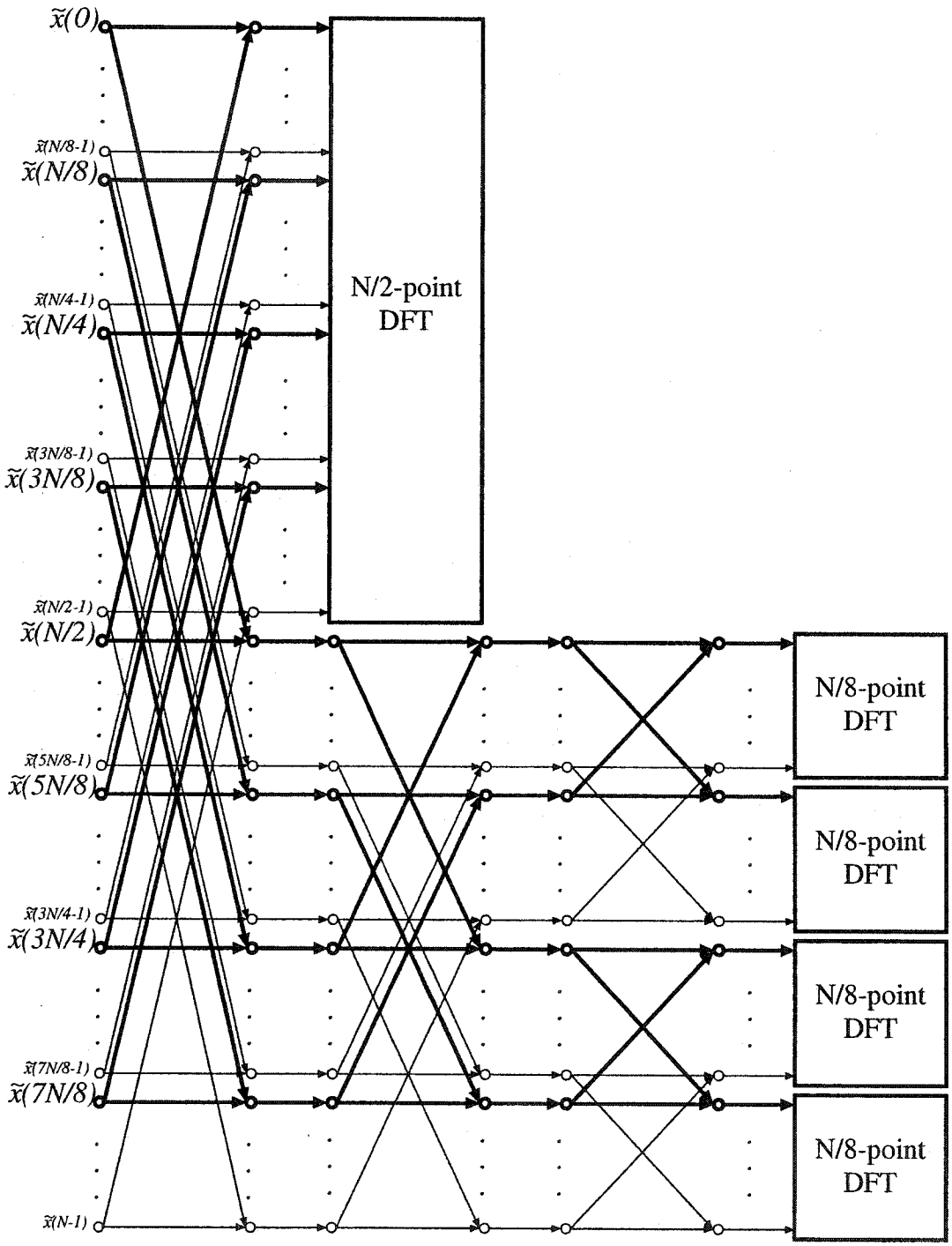

Figure 6.3: First stage of the proposed radix-2/8 FFT algorithm.

Table 6.1: Arithmetic complexities of the $8q$-, $4q$- and $2q$-point DFTs

N	Multiplications	Additions
$2q$	$2\widetilde{M}_q$	$2\widetilde{A}_q+4q$
$4q$	$4\widetilde{M}_q$	$4\widetilde{A}_q+16q$
$8q$	$6\widetilde{M}_q+2\widetilde{M}_q^s$	$8\widetilde{A}_q+52q$

6.2.2 Computational Complexity

In this section, we consider the performance of the proposed radix-2/8 FFT algorithm for length-$q \times 2^m$ DFT by analyzing its computational complexity and comparing it with that of the existing radix-2/4 FFT algorithm reported in [17].

6.2.2.1 Arithmetic Complexity

Let $\widetilde{M}_q^s$ be the number of multiplications required by a scaled length-q DFT. Let $\widetilde{M}_q$ and $\widetilde{A}_q$ be, respectively, the number of multiplications and number of additions required by a length-q DFT. The arithmetic complexity of a length-$2q$ DFT can be computed using (6.2) and (6.3), whereas that of a length-$4q$ DFT can be obtained using (6.4)-(6.6). Finally, the required arithmetic complexity of a length-$8q$ DFT can be calculated from (6.14)-(6.22). These arithmetic complexities are given in Table 6.1.

In order to carry out a complete comparison of the arithmetic complexity of the proposed radix-2/8 FFT algorithm with that of the existing radix-2/4 FFT algorithm [17], we count the number of operations required by each algorithm whether the complex multiplication is performed using the 3mult-3add scheme or using the 4mult-2add scheme.

For the 4mult-2add scheme, the general butterfly of the proposed radix-2/8 FFT algorithm based on (6.27), (6.28) and (6.23) requires 20 real multiplications and 44 real additions. The algorithm has $2q$ special butterflies that can be indexed by $k = p\frac{N}{16q}$, where $p = 0, 1, ..., (2q-1)$. For a given even value of p, the corresponding special butterfly

requires 4 real multiplications and 36 real additions. For a given odd value of p, the corresponding special butterfly requires 16 real multiplications and 40 real additions. The proposed decomposition consists of dividing a length-N DFT to one length-$N/2$ DFT and four length-$N/8$ DFTs in the first stage. This is achieved by performing $(N/8)-2q$ general butterflies and the $2q$ special butterflies. The decomposition process is repeated successively for each of the resulting DFT until the size is reduced to a $8q$-, $4q$-, $2q$- or q-point DFT. Therefore, it is seen that the expressions for the number of real multiplications and real additions of the proposed FFT algorithm are, respectively,

$$\mathrm{M}_{\mathrm{r2}/8}^{42}(1,N)=\frac{5}{2}N-20q+\mathrm{M}_{\mathrm{r2}/8}^{42}\left(1,\frac{N}{2}\right)+4\mathrm{M}_{\mathrm{r2}/8}^{42}\left(1,\frac{N}{8}\right),\ N>8q \tag{6.32}$$

$$\mathrm{A}_{\mathrm{r2}/8}^{42}(1,N)=\frac{11}{2}N-12q+\mathrm{A}_{\mathrm{r2}/8}^{42}\left(1,\frac{N}{2}\right)+4\mathrm{A}_{\mathrm{r2}/8}^{42}\left(1,\frac{N}{8}\right),\ N>8q \tag{6.33}$$

Similarly, if the 3mult-3add scheme is considered, it can be shown that the corresponding numbers are

$$\mathrm{M}_{\mathrm{r2}/8}^{33}(1,N)=2N-16q+\mathrm{M}_{\mathrm{r2}/8}^{33}\left(1,\frac{N}{2}\right)+4\mathrm{M}_{\mathrm{r2}/8}^{33}\left(1,\frac{N}{8}\right),\ N>8q \tag{6.34}$$

$$\mathrm{A}_{\mathrm{r2}/8}^{33}(1,N)=6N-16q+\mathrm{A}_{\mathrm{r2}/8}^{33}\left(1,\frac{N}{2}\right)+4\mathrm{A}_{\mathrm{r2}/8}^{33}\left(1,\frac{N}{8}\right),\ N>8q \tag{6.35}$$

It can be verified that the radix-2/4 FFT algorithm of [17] also has $2q$ special butterflies that are precisely indexed by $p\frac{N}{8q}$, where $p=0,1,...,(2q-1)$. For a given even value of p, the corresponding special butterfly requires 12 real additions. For a given odd value of p, the corresponding special butterfly requires 4 real multiplications and 16 real additions. When the 3mult-3add scheme is used, the general butterfly of the algorithm in [17] requires 6 real multiplications and 18 real additions. Thus, it can be seen that the expressions for the number of real multiplications and real additions of the radix-2/4 FFT algorithm of [17]

are, respectively

$$\mathrm{M}^{33}_{\mathrm{r2/4}}(1,N) = \frac{3}{2}N - 8q + \mathrm{M}^{33}_{\mathrm{r2/4}}\left(1,\frac{N}{2}\right) + 2\mathrm{M}^{33}_{\mathrm{r2/4}}\left(1,\frac{N}{4}\right),\ N > 8q \tag{6.36}$$

$$\mathrm{A}^{33}_{\mathrm{r2/4}}(1,N) = \frac{9}{2}N - 8q + \mathrm{A}^{33}_{\mathrm{r2/4}}\left(1,\frac{N}{2}\right) + 2\mathrm{A}^{33}_{\mathrm{r2/4}}\left(1,\frac{N}{4}\right),\ N > 8q \tag{6.37}$$

Similarly, when the 4mult-2add scheme is used, the corresponding numbers are [17]

$$\mathrm{M}^{42}_{\mathrm{r2/4}}(1,N) = 2N - 12q + \mathrm{M}^{42}_{\mathrm{r2/4}}\left(1,\frac{N}{2}\right) + 2\mathrm{M}^{42}_{\mathrm{r2/4}}\left(1,\frac{N}{4}\right),\ N > 8q \tag{6.38}$$

$$\mathrm{A}^{42}_{\mathrm{r2/4}}(1,N) = 4N - 4q + \mathrm{A}^{42}_{\mathrm{r2/4}}\left(1,\frac{N}{2}\right) + 2\mathrm{A}^{42}_{\mathrm{r2/4}}\left(1,\frac{N}{4}\right),\ N > 8q \tag{6.39}$$

The arithmetic complexities of the proposed radix-2/8 FFT algorithm, and the radix-2/4 FFT algorithm reported in [17] for complex data for various values of N are given in Tables 6.2-6.7. In Tables 6.4-6.7, the values used for $\widetilde{M}^s_q$, $\widetilde{M}_q$ and $\widetilde{A}_q$ for the proposed FFT algorithm are shown in Table 6.8, for $q = 3, 9$ and 15. These values are the same as the ones used in [17] for the radix-2/4 FFT algorithm for the corresponding values of q. It can be seen from Tables 6.2-6.7 that, in general, the total number of arithmetic operations is the same for both the proposed radix-2/8 FFT and the existing radix-2/4 FFT algorithms. For the interesting case of $q = 1$, the total number of operations (i.e., multiplications+additions) is exactly the same for the proposed FFT algorithm and the algorithms in [11], [120] and [17] for both the 3mult-3add and 4mult-2add schemes. Note that this number is less than the corresponding numbers for the algorithms in [16] and [20].

Table 6.2: Number of arithmetic operations required for the computation of the 1-D DFT using the 3mult-3add scheme for $q = 1$

		Radix-2/4 FFT in [17]			Proposed radix-2/8 FFT		
m	N	Mults.	Adds.	M.+A.	Mults.	Adds.	M.+A.
3	8	4	52	56	4	52	56
4	16	20	148	168	20	148	168
5	32	68	388	456	68	388	456
6	64	196	964	1160	196	964	1160
7	128	516	2308	2824	516	2308	2824
8	256	1284	5380	6664	1284	5380	6664
9	512	3076	12292	15368	3076	12292	15368
10	1024	7172	27652	34824	7172	27652	34824
11	2048	16388	61444	77832	16388	61444	77832
12	4096	36868	135172	172040	36868	135172	172040

Table 6.3: Number of arithmetic operations required for the computation of the 1-D DFT using the 4mult-2add scheme for $q = 1$

		Radix-2/4 FFT in [17]			Proposed radix-2/8 FFT		
m	N	Mults.	Adds.	M.+A.	Mults.	Adds.	M.+A.
3	8	4	52	56	4	52	56
4	16	24	144	168	24	144	168
5	32	84	372	456	84	372	456
6	64	248	912	1160	240	920	1160
7	128	660	2164	2824	636	2188	2824
8	256	1656	5008	6664	1592	5072	6664
9	512	3988	11380	15368	3812	11556	15368
10	1024	9336	25488	34824	8896	25928	34824
11	2048	21396	56436	77832	20364	57468	77832
12	4096	48248	123792	172040	45832	126208	172040

Table 6.4: Number of arithmetic operations required for the computation of the 1-D DFT using the 4mult-2add scheme for $q = 3$

		Radix-2/4 FFT in [17]			Proposed radix-2/8 FFT		
m	N	Mults.	Adds.	M.+A.	Mults.	Adds.	M.+A.
1	6	4	36	40	4	36	40
2	12	8	96	104	8	96	104
3	24	24	252	276	24	252	276
4	48	100	624	724	100	624	724
5	96	304	1500	1804	312	1500	1812
6	192	852	3504	4356	828	3528	4356
7	384	2192	8028	10220	2128	8100	10228
8	768	5396	18096	23492	5236	18288	23524
9	1536	12816	40284	53100	12328	40812	53140

Table 6.5: Number of arithmetic operations required for the computation of the 1-D DFT using the 3mult-3add scheme for $q = 3$

		Radix-2/4 FFT in [17]			Proposed radix-2/8 FFT		
m	N	Mults.	Adds.	M.+A.	Mults.	Adds.	M.+A.
1	6	4	36	40	4	36	40
2	12	8	96	104	8	96	104
3	24	24	252	276	24	252	276
4	48	88	636	724	88	636	724
5	96	256	1548	1804	264	1548	1812
6	192	696	3660	4356	696	3660	4356
7	384	1760	8460	10220	1768	8460	10228
8	768	4280	19212	23492	4312	19212	23524
9	1536	10080	43020	53100	10120	43020	53140

Table 6.6: Number of arithmetic operations required for the computation of the 1-D DFT using the 4mult-2add scheme for $q = 9$

		Radix-2/4 FFT in [17]			Proposed radix-2/8 FFT		
m	N	Mults.	Adds.	M.+A.	Mults.	Adds.	M.+A.
1	18	32	204	236	32	204	236
2	36	64	480	544	64	480	544
3	72	140	1140	1280	140	1140	1280
4	144	448	2640	3088	448	2640	3088
5	288	1196	6036	7232	1244	6036	7280
6	576	3136	13584	16720	3064	13656	16720
7	1152	7724	30228	37952	7556	30444	38000
8	2304	18496	66576	85072	18112	67152	85264

Table 6.7: Number of arithmetic operations required for the computation of the 1-D DFT using the 4mult-2add scheme for $q = 15$

		Radix-2/4 FFT in [17]			Proposed radix-2/8 FFT		
m	N	Mults.	Adds.	M.+A.	Mults.	Adds.	M.+A.
1	30	60	396	456	60	396	456
2	60	120	912	1032	120	912	1032
3	120	264	2124	2388	264	2124	2388
4	240	804	4848	5652	804	4848	5652
5	480	2112	10956	13068	2184	10956	13140
6	960	5460	24432	29892	5340	24552	29892
7	1920	13344	53964	67308	13056	54324	67380

Table 6.8: Number of multiplications and additions required by 3-, 9- and 15-point DFTs

q	$\widetilde{M}_q$	$\widetilde{M}_q^s$	$\widetilde{A}_q$
3	2	6	12
9	16	22	84
15	30	42	168

6.2.2.2 Data Transfers

In counting the number of data transfer operations between the processor and the external memory (RAM), we assume that sufficient registers are available in the processor to perform an entire butterfly without using any intermediate transfer operations. Let us give a brief implementation scheme of the proposed FFT algorithm to show how an expression for the number of data transfers, not including the read operations for the twiddle factors, is obtained. The implementation of the butterfly for a given value of k, based on (6.27), (6.28) and (6.23), of the proposed FFT algorithm consists, of reading two points from the external memory of the processor and performing the operations of addition and subtraction using these two points as the operands. The result of the addition is returned to the external memory, whereas that of the subtraction is kept in an internal register of the processor. This process is repeated four times to compute $\widetilde{a}_e$ and $\widetilde{a}_o$ given by (6.27) of the butterfly. The four points kept in the processor are used to compute (6.28), whose outputs are used to process the sub-butterfly given by (6.23). The results of the additions for $k = 0, 1, ..., (N/8) - 1$ returned to the external memory are grouped to form the first sub-sequence of length $N/2$. The four sub-sequences of length $N/8$ are formed by grouping separately the results of each line of the output vector of (6.23). This scheme of implementation reduces significantly the number of data transfers and address generations. To compare the number of data transfers and address generations of the proposed radix-2/8 and the radix-2/4 FFT algorithm of [17], we consider the case of $q = 1$ as an example. For this case, both the algorithms require four types of butterflies: one type to compute length-2 DFTs, two types of special butterflies and one general butterfly. However, the radix-2/8 FFT requires one additional type of butterfly to compute length-4 DFTs. It can be shown that the expression for the number of data transfer operations (real and imaginary parts) for the implementation of the radix-2/8 FFT

algorithm using all the five types of butterflies is given by

$$\begin{aligned} D_{r2/8}(1,N) &= 2N + D_{r2/8}\left(1,\frac{N}{2}\right) + 4D_{r2/8}\left(1,\frac{N}{8}\right),\ N \geq 8, \\ D_{r2/8}(1,4) &= 8, D_{r2/8}(1,2) = 4, D_{r2/8}(1,1) = 0 \end{aligned} \tag{6.40}$$

whereas the corresponding expression for the implementation of the radix-2/4 FFT algorithm using the first four types of butterflies is given by

$$\begin{aligned} D_{r2/4}(1,N) &= 2N + D_{r2/4}\left(1,\frac{N}{2}\right) + 2D_{r2/4}\left(1,\frac{N}{4}\right),\ N \geq 4, \\ D_{r2/4}(1,2) &= 4, D_{r2/4}(1,1) = 0 \end{aligned} \tag{6.41}$$

If in the implementation of the latter algorithm, all the five types of butterflies are used, then the corresponding expression is

$$\begin{aligned} D_{r2/4}(1,N) &= 2N + D_{r2/4}\left(1,\frac{N}{2}\right) + 2D_{r2/4}\left(1,\frac{N}{4}\right),\ N \geq 8, \\ D_{r2/4}(1,4) &= 8, D_{r2/4}(1,2) = 4, D_{r2/4}(1,1) = 0 \end{aligned} \tag{6.42}$$

The number of data transfer operations for the two algorithms are compared in Table 6.9. The 4-butterfly and 5-butterfly implementations of the radix-2/4 FFT algorithms [17] or [11] require about 20% more data transfer operations compared to the implementation of the proposed FFT algorithm. During the data transfer process the address generation operations are required to read/write the data from/in memory, and hence, similar savings are also obtained by the proposed FFT algorithm in the case of the address generation. Note that the proposed FFT algorithm and the algorithms in [20] and [120] have the same number of data transfer operations.

Table 6.9: Comparison of the number of data transfers operations for $q = 1$

		4-butterfly implementation of the radix-2/4 FFT	5-butterfly implementation of the radix-2/4 FFT	Implementation of the proposed radix-2/8 FFT	Savings in C over A	Savings in C over B
m	N	$D_{r2/4}(1,N)$ (A)	$D_{r2/4}(1,N)$ (B)	$D_{r2/8}(1,N)$ (C)	%	%
3	8	36	32	24	33.33	25.00
4	16	92	80	72	21.73	10.00
5	32	228	208	168	26.31	19.23
6	64	540	496	392	27.40	20.96
7	128	1252	1168	936	25.23	19.86
8	256	2844	2672	2120	25.45	20.65
9	512	6372	6032	4712	26.05	21.88
10	1024	14108	13424	10504	25.54	21.75
11	2048	30948	29584	23080	25.42	21.98
12	4096	67356	64624	50120	25.58	22.44

6.2.2.3 Twiddle Factors

A complex multiplication using the 3mult-3add scheme is performed as

$$(c+js)(x_R+jx_I) = (c+s)x_R - s(x_R+x_I) + j[s(x_R+x_I)+(c-s)x_I] \tag{6.43}$$

where c and s are, respectively, the real and imaginary parts of a twiddle factor. Hence, one complex multiplication requires three real coefficients, s, $c+s$ and $c-s$, to be calculated or loaded from a lookup table. It is obvious that two real coefficients are required when the 4mult-2add scheme is used. We discuss here only the cases of $q = 1$ and $q = 3$. In counting the number of real coefficient evaluations or accesses to the lookup table required by the proposed radix-2/8 and the radix-2/4 FFT algorithm of [17], it is assumed that the coefficients required by the special butterflies, such as $\sqrt{2}/2$, $\sqrt{3}/2$, $\cos(\pi/8)$ and $\sin(\pi/8)$ are initialized and kept in the internal registers of the processor during the processing time of the corresponding algorithm. However, for the radix-2/8 and radix-2/4 FFT algorithms, 4 and 2 real coefficients are respectively required to be evaluated or loaded from the lookup

table by the corresponding general butterfly using the 4mult-2add scheme; the corresponding numbers are 6 and 3 for the 3mult-3add scheme. Therefore, by counting the required number of the general butterflies in the radix-2/8 and radix-2/4 FFT algorithms, the number of real coefficient evaluations or accesses to the lookup table for the two algorithms are, respectively

$$T^{42}_{r2/8}(1,N) = \frac{N}{2} - 8q + T^{42}_{r2/8}\left(1,\frac{N}{2}\right) + 4T^{42}_{r2/8}\left(1,\frac{N}{8}\right),\ N > 8q \tag{6.44}$$

and

$$T^{42}_{r2/4}(1,N) = \frac{N}{2} - 4q + T^{42}_{r2/4}\left(1,\frac{N}{2}\right) + 2T^{42}_{r2/4}\left(1,\frac{N}{4}\right),\ N > 4q \tag{6.45}$$

using the 4mult-2add scheme, and

$$T^{33}_{r2/8}(1,N) = \frac{3}{4}N - 12q + T^{33}_{r2/8}\left(1,\frac{N}{2}\right) + 4T^{33}_{r2/8}\left(1,\frac{N}{8}\right),\ N > 8q \tag{6.46}$$

and

$$T^{33}_{r2/4}(1,N) = \frac{3}{4}N - 6q + T^{33}_{r2/4}\left(1,\frac{N}{2}\right) + 2T^{33}_{r2/4}\left(1,\frac{N}{4}\right),\ N > 4q \tag{6.47}$$

using the 3mult-3add scheme.

For various values of N, the number of real coefficient evaluations or accesses to the lookup table of the proposed radix-2/8 FFT algorithm is compared to that required by the radix-2/4 FFT algorithm when $q = 1$ in Table 6.10 and $q = 3$ in Table 6.11. It is seen from these tables that savings of over 30% in the evaluation of twiddle factors or in the access to the lookup table can easily be achieved by the proposed FFT algorithm. Note that, when the lookup table is used, identical savings are obtained by the proposed FFT algorithm in the address generation for reading the twiddle factors. Specifically, for the case of $q = 1$, the proposed FFT algorithm achieves savings of 50% as compared with the algorithm in [120], over 50% as compared with the algorithm of [20] and over 65% as compared with the

Table 6.10: Comparison of the number of twiddle factor evaluations or accesses to the lookup table for $q = 1$

		Radix-2/4 FFT	Proposed radix-2/8 FFT	Savings	Radix-2/4 FFT	Proposed radix-2/8 FFT	Savings
m	N	$T^{42}_{r2/4}(1,N)$	$T^{42}_{r2/8}(1,N)$	%	$T^{33}_{r2/4}(1,N)$	$T^{33}_{r2/8}(1,N)$	%
3	8	0	0	0	0	0	0
4	16	4	0	100	6	0	100
5	32	16	8	50	24	12	50
6	64	52	32	38.46	78	48	38.46
7	128	144	88	38.88	216	132	38.38
8	256	372	240	35.48	558	360	35.48
9	512	912	616	32.45	1368	924	32.45
10	1024	2164	1472	31.97	3246	2208	31.97
11	2048	5008	3448	31.15	7512	5172	31.15
12	4096	11380	7952	30.12	17070	11928	30.12

Table 6.11: Comparison of the number of twiddle factor evaluations or accesses to the lookup table for $q = 3$

		Radix-2/4 FFT	Proposed radix-2/8 FFT	Savings	Radix-2/4 FFT	Proposed radix-2/8 FFT	Savings
m	N	$T^{42}_{r2/4}(1,N)$	$T^{42}_{r2/8}(1,N)$	%	$T^{33}_{r2/4}(1,N)$	$T^{33}_{r2/8}(1,N)$	%
3	24	0	0	0	0	0	0
4	48	12	0	100	18	0	100
5	96	48	24	50	72	36	50
6	192	156	96	38.46	234	144	38.46
7	384	432	264	38.88	648	396	38.38
8	768	1116	720	35.48	1674	1080	35.48
9	1536	2736	1848	32.45	4104	2772	32.45
10	3072	6492	4416	31.97	9738	6624	31.97
11	6144	15024	10344	31.15	22536	15516	31.15
12	12288	34140	23856	30.12	51210	35784	30.12

standard radix-2/4 FFT algorithm of [11]. Note that the proposed radix-2/8 FFT algorithm for $q = 1$ can be considered as an improved version of the algorithm reported in [120].

6.2.2.4 Example

The purpose of this example is to show the importance of a $q \times 2^m$-point FFT algorithm in providing a large range of choices for the length N. To illustrate this point, we use the algorithm proposed here since we have already shown that the complexity of this algorithm is less than those of the existing ones. Consider the case of filtering, which plays a central role in signal processing. In general, the length of the impulse response of the filter is much lower than the length of the input signal and is not a power of two. Instead of implementing the linear convolution directly, it is more efficient to carry out a block convolution using the FFT [1], [2]. If we assume that the length of the impulse response is P and the length of the blocks of the input signal is L, then the possible choice for the length N of the FFT is $N \geq (L+P-1)$. Since $(L+P-1)$ is, in general, not a power of two, the use of a 2^n-point FFT algorithm requires a zero padding technique to augment the lengths P and L to the closest $N = 2^n$ that satisfies $2^n \geq (L+P-1)$. However, when using a $q \times 2^m$-point FFT algorithm, for $q > 1$, it is sufficient to choose the closest $N = q \times 2^m$ that satisfies $q \times 2^m \geq (L+P-1)$. Table 6.12 gives the arithmetic complexities and twiddle factor requirements using the 3×2^m-point and 2^n-point FFT algorithms. It is clear from this table that the 3×2^m-point FFT algorithm substantially reduces the arithmetic complexity and the number of twiddle factor evaluations or accesses to the lookup table compared to those required by the 2^n-point FFT algorithm. In addition, it reduces the number of data transfers. Similar comparisons may be carried out for different values of q to show the importance of the $q \times 2^m$-point FFT algorithm in providing many other interesting choices for the length N different from the usual one of $N = 2^n$.

Table 6.12: Comparison of the arithmetic complexities and twiddle factor requirements for the proposed 2^n- and 3×2^m-point FFT algorithms

Possible choice of length N	2^n-point FFT algorithm				3×2^m-point FFT algorithm			
	N	M.+A.	$T^{42}_{r2/8}$	$T^{33}_{r2/8}$	N	M.+A.	$T^{42}_{r2/8}$	$T^{33}_{r2/8}$
$N \geq 23$	32	456	8	12	24	276	0	0
$N \geq 47$	64	1160	32	48	48	724	0	0
$N \geq 95$	128	2824	88	132	96	1812	24	36
$N \geq 191$	256	6664	240	360	192	4356	96	144
$N \geq 383$	512	15368	616	924	384	10228	264	396
$N \geq 767$	1024	34824	1472	2208	768	23524	720	1080
$N \geq 1535$	2048	77832	3448	5172	1536	53140	1848	2772

6.3 Radix-2/8 Approach for the Computation of Length-$q \times 2^m$ 1-D DHT

The idea of the radix-2/4 approach proposed in [11] to compute a length-2^m DFT has been exploited in [45] and [60] to compute a length-2^m DHT. Recently, an alternative radix-2/4 FFT algorithm proposed for computing a length-$q \times 2^m$ DFT [17], has also been used for computing a length-$q \times 2^m$ DHT [55]-[57]. It is known that these radix-2/4 FHT algorithms are the simplest algorithms having the lowest number of arithmetic operations [45], [55]-[57].

We now propose a new radix-2/8 FHT algorithm for efficiently computing a length-$q \times 2^m$ DHT, where q is an odd integer, by introducing a mixture of radix-2 and radix-8 index maps in the decomposition of the DHT [122], [123].

6.3.1 Proposed Radix-2/8 DIF FHT Algorithm

Using (2.2), the 1-D DHT of length N is given by

$$X(n) = \sum_{k=0}^{N-1} x(k)\text{cas}\left(\frac{2\pi}{N}nk\right),\ n = 0, 1, ..., N-1 \tag{6.48}$$

The sequence length N is assumed to be $q \times 2^m$, where q is an odd integer and $m > 0$. Using the trigonometric identities, cas $\left(\frac{2\pi}{N}nk\right)$ can be expressed as

$$\begin{aligned}\text{cas}\left(\frac{2\pi}{N}nk\right) &= \left(\frac{1}{2}+\frac{1}{2}j\right)W_N^{nk} + \left(\frac{1}{2}-\frac{1}{2}j\right)W_N^{-nk} \\ &= \text{Re}\left[(1+j)W_N^{nk}\right]\end{aligned} \tag{6.49}$$

where $W_N = exp(-j2\pi/N)$, $j = \sqrt{-1}$ and Re [y] denotes the real part of y. Another expression for computing the DHT can be obtained from (6.48) using (6.49) as

$$X(n) = \sum_{k=0}^{N-1} \text{Re}\left[x(k)(1+j)W_N^{nk}\right],\ n = 0, 1, ..., N-1 \tag{6.50}$$

Let us first consider the case when $m = 1$. Then, the decomposition of the DHT coefficients $X(n)$ in (6.50) into even-indexed and odd-indexed terms provides

$$\begin{aligned}X(2n) &= \sum_{k=0}^{q-1} \text{Re}\left[r_e(k)(1+j)W_q^{nk}\right] \\ &= \sum_{k=0}^{q-1} r_e(k)\text{cas}\left(\frac{2\pi}{q}nk\right),\ n = 0, 1, ..., q-1\end{aligned} \tag{6.51}$$

$$\begin{aligned}X((2n+q) \bmod 2q) &= \sum_{k=0}^{q-1} \text{Re}\left[r_o(k)(-1)^k(1+j)W_q^{nk}\right] \\ &= \sum_{k=0}^{q-1} r_o(k)(-1)^k\text{cas}\left(\frac{2\pi}{q}nk\right),\ n = 0, 1, ..., q-1\end{aligned} \tag{6.52}$$

where the sequences $r_e(k)$ in (6.51) and $r_o(k)$ in (6.52) are given by

$$\begin{bmatrix} r_e(k) \\ r_o(k) \end{bmatrix} = \mathbf{H}_2 \begin{bmatrix} x(k) \\ x(k+q) \end{bmatrix} \tag{6.53}$$

This decomposition reduces the computation of the length-$2q$ DHT to the computation of two length-q DHTs given by (6.51) and (6.52), whose input sequences are obtained from the original input sequence $x(k)$ using (6.53). The decomposition process of a length-$2q$ DHT is shown in Fig. 6.4. In a similar way, when $m = 2$, the decomposition of (6.50) provides

$$X(2n) = \sum_{k=0}^{2q-1} d_e(k)\text{cas}\left(\frac{2\pi}{2q}nk\right), \; n = 0, 1, ..., (2q-1) \tag{6.54}$$

for the even-indexed terms, and

$$X\left((4n+q) \bmod 4q\right) = \sum_{k=0}^{q-1} \text{Re}\left[(d_o(k) + (-j)^q d_o(k+q)) W_4^k (1+j) W_q^{nk}\right], \quad n = 0, 1, ..., q-1 \tag{6.55}$$

$$X\left((4n+3q) \bmod 4q\right) = \sum_{k=0}^{q-1} \text{Re}\left[(d_o(k) + (j)^q d_o(k+q)) W_4^{-k} (1+j) W_q^{nk}\right], \quad n = 0, 1, ..., q-1 \tag{6.56}$$

for the odd-indexed terms, where the sequences $d_e(k)$ in (6.54) and $d_o(k)$ in (6.55) and (6.56) are given by

$$\begin{bmatrix} d_e(k) \\ d_o(k) \end{bmatrix} = \mathbf{H}_2 \begin{bmatrix} x(k) \\ x(k+2q) \end{bmatrix} \tag{6.57}$$

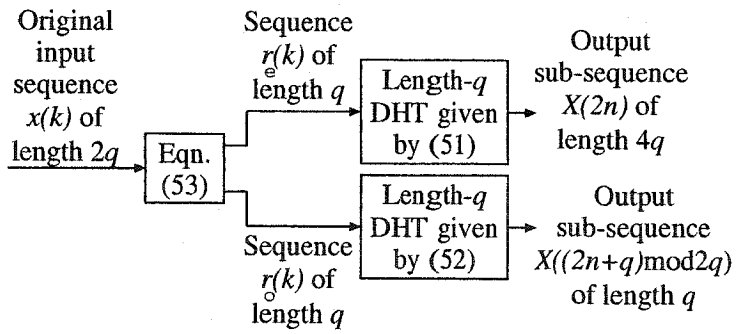

Figure 6.4: Decomposition of a length-$2q$ DHT.

After computing the real parts in (6.55) and (6.56), the odd-indexed terms can be expressed as

$$\begin{bmatrix} X((4n+q) \bmod 4q) \\ X((4n+3q) \bmod 4q) \end{bmatrix} = \mathbf{H}_2 \begin{bmatrix} F(n) \\ G(q-n) \end{bmatrix}, \; n = 0, 1, ..., q-1 \tag{6.58}$$

where

$$F(n) = \sum_{k=0}^{q-1} f(k)\text{cas}\left(\frac{2\pi}{q}nk\right), \; f(k) = \begin{cases} (-1)^{\frac{k}{2}} d_o(k), & k \text{ even} \\ (-1)^{\frac{q+1}{2}}(-1)^{\frac{k-1}{2}} d_o(k+q), & k \text{ odd} \end{cases} \tag{6.59}$$

and

$$G(n) = \sum_{k=0}^{q-1} g(k)\text{cas}\left(\frac{2\pi}{q}nk\right), \; g(k) = \begin{cases} -(-1)^{\frac{q+1}{2}}(-1)^{\frac{k}{2}} d_o(k+q), & k \text{ even} \\ (-1)^{\frac{k-1}{2}} d_o(k), & k \text{ odd} \end{cases} \tag{6.60}$$

Therefore, the decomposition of the length-$4q$ DHT leads to the computation of the length-$2q$ DHT given by (6.54) and the two length-q DHTs given by (6.59) and (6.60). This decomposition is illustrated in Fig. 6.5.

By introducing a mixture of radix-2 and radix-8 index maps, we now propose the following novel decomposition of (6.50) so that we can compute a length-$q \times 2^m$ ($m > 2$)

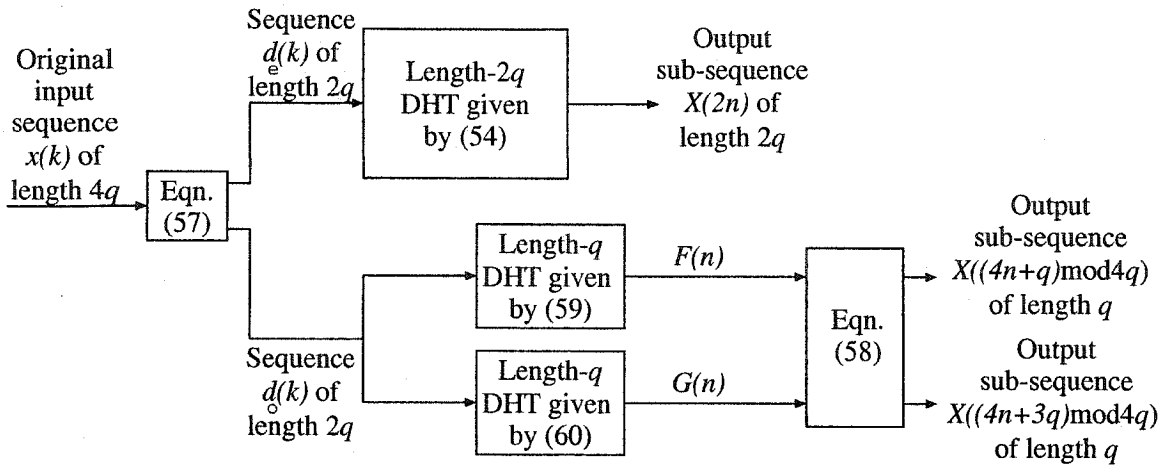

Figure 6.5: Decomposition of a length-$4q$ DHT.

DHT

$$X(2n) = \sum_{k=0}^{N/2-1} a_e(k)\text{cas}\left(\frac{2\pi}{N/2}nk\right), \; n = 0, 1, ..., (N/2-1) \tag{6.61}$$

for the even-indexed terms, as

$$X((8n+q) \bmod N) = \sum_{k=0}^{N/8-1} \text{Re}\left[\left((a_o(k) + (-j)^q a_o(k+N/4)) + e^{-j\frac{q\pi}{4}}(a_o(k+N/8) + (-j)^q a_o(k+3N/8))\right) W_N^{qk}(1+j)W_{N/8}^{nk}\right], \quad n = 0, 1, ..., (N/8-1) \tag{6.62}$$

$$X((8n+3q) \bmod N) = \sum_{k=0}^{N/8-1} \text{Re}\left[\left((a_o(k) + (j)^q a_o(k+N/4)) + e^{-j\frac{3q\pi}{4}}(a_o(k+N/8) + (j)^q a_o(k+3N/8))\right) W_N^{3qk}(1+j)W_{N/8}^{nk}\right], \quad n = 0, 1, ..., (N/8-1) \tag{6.63}$$

$$X((N+8n-3q) \bmod N) = \sum_{k=0}^{N/8-1} \mathrm{Re}\left[\left((a_o(k)+(-j)^q a_o(k+N/4))\right.\right.$$
$$\left.\left.-e^{-j\frac{q\pi}{4}}\left(a_o(k+N/8)+(-j)^q a_o(k+3N/8)\right)\right) W_N^{-3qk}(1+j)W_{N/8}^{nk}\right],$$
$$n = 0, 1, ..., (N/8-1) \quad (6.64)$$

$$X((N+8n-q) \bmod N) = \sum_{k=0}^{N/8-1} \mathrm{Re}\left[\left((a_o(k)+(j)^q a_o(k+N/4))\right.\right.$$
$$\left.\left.-e^{-j\frac{3q\pi}{4}}\left(a_o(k+N/8)+(j)^q a_o(k+3N/8)\right)\right) W_N^{-qk}(1+j)W_{N/8}^{nk}\right],$$
$$n = 0, 1, ..., (N/8-1) \quad (6.65)$$

for the odd-indexed terms. The sequences $a_e(k)$ in (6.61) and $a_o(k)$ in (6.62)-(6.65) can be obtained by

$$\begin{bmatrix} a_e(k) \\ a_o(k) \end{bmatrix} = \mathbf{H}_2 \begin{bmatrix} x(k) \\ x(k+N/2) \end{bmatrix} \quad (6.66)$$

which can be considered as the first step of the new decomposition. This step is processed by applying a second-order Hadamard matrix $\mathbf{H}_2$ to the input sequence $x(k)$ without requiring any twiddle factor. Some of the properties that will be used throughout this section are given in the following lemmas, without proof.

Lemma 1: Let $\cos\left(\beta\frac{\pi}{4}\right) = \frac{\sqrt{2}}{2}c_q$ and $\sin\left(\beta\frac{\pi}{4}\right) = \frac{\sqrt{2}}{2}s_q$, where β is an odd integer. Then

(a) For $\beta = q$, $c_q s_q = (-1)^{\frac{q-1}{2}}$

(b) For $\beta = 3q$, $c_{3q} = -c_q$ and $s_{3q} = s_q$

Lemma 2: Let β be an odd integer. Then

(a) $\cos\left(3\beta\frac{\pi}{8}\right) = (-1)^{\frac{\beta-1}{2}} \sin\left(\beta\frac{\pi}{8}\right)$

(b) $\sin\left(3\beta\frac{\pi}{8}\right) = (-1)^{\frac{\beta-1}{2}} \cos\left(\beta\frac{\pi}{8}\right)$

Now, computing the real parts in (6.62)-(6.65) and using Lemma 1, the odd-indexed terms of the proposed decomposition can be expressed in a compact form as

$$\begin{bmatrix} X((8n+q) \bmod N) \\ X((8n+3q) \bmod N) \\ X((N+8n-3q) \bmod N) \\ X((N+8n-q) \bmod N) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \\ 1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} F_1(n) \\ G_1(\frac{N}{8}-n) \\ F_3(n) \\ G_3(\frac{N}{8}-n) \end{bmatrix},$$

$$n = 0, 1, ..., (N/8-1) \qquad (6.67)$$

where

$$F_1(n) = \sum_{k=0}^{N/8-1} \left[\left(a_o(k) + \frac{\sqrt{2}}{2} c_q b_o(k+N/8) \right) \cos\left(\frac{2\pi}{N} qk\right) + (-1)^{\frac{q+1}{2}} \left(a_o(k+N/4) + \frac{\sqrt{2}}{2} c_q b_e(k+N/8) \right) \sin\left(\frac{2\pi}{N} qk\right) \right] \operatorname{cas}\left(\frac{2\pi}{N/8} nk\right) \qquad (6.68)$$

$$G_1(n) = \sum_{k=0}^{N/8-1} \left[\left(a_o(k) + \frac{\sqrt{2}}{2} c_q b_o(k+N/8) \right) \sin\left(\frac{2\pi}{N} qk\right) - (-1)^{\frac{q+1}{2}} \left(a_o(k+N/4) + \frac{\sqrt{2}}{2} c_q b_e(k+N/8) \right) \cos\left(\frac{2\pi}{N} qk\right) \right] \operatorname{cas}\left(\frac{2\pi}{N/8} nk\right) \qquad (6.69)$$

$$F_3(n) = \sum_{k=0}^{N/8-1} \left[\left(a_o(k) - \frac{\sqrt{2}}{2} c_q b_o(k+N/8) \right) \cos\left(\frac{2\pi}{N} 3qk\right) - (-1)^{\frac{q+1}{2}} \left(a_o(k+N/4) - \frac{\sqrt{2}}{2} c_q b_e(k+N/8) \right) \sin\left(\frac{2\pi}{N} 3qk\right) \right] \operatorname{cas}\left(\frac{2\pi}{N/8} nk\right) \qquad (6.70)$$

$$G_3(n) = \sum_{k=0}^{N/8-1} \left[\left(a_o(k) - \frac{\sqrt{2}}{2} c_q b_o(k+N/8) \right) \sin\left(\frac{2\pi}{N} 3qk \right) \right.$$
$$\left. +(-1)^{\frac{q+1}{2}} \left(a_o(k+N/4) - \frac{\sqrt{2}}{2} c_q b_e(k+N/8) \right) \cos\left(\frac{2\pi}{N} 3qk \right) \right] \text{cas}\left(\frac{2\pi}{N/8} nk \right) \quad (6.71)$$

for $n = 0, 1, ..., (N/8-1)$. The sequences $b_e(k)$ and $b_o(k)$ in (6.68)-(6.71) can be obtained by

$$\begin{bmatrix} b_e(k) \\ b_o(k) \end{bmatrix} = \mathbf{H}_2 \begin{bmatrix} a_o(k) \\ a_o(k+N/4) \end{bmatrix} \quad (6.72)$$

Let us now use this new decomposition to efficiently compute the length-$8q$ DHT. We let $N = 8q$ in (6.61) and (6.66)-(6.72). Then, (6.61) becomes

$$X(2n) = \sum_{k=0}^{4q-1} a_e(k) \text{cas}\left(\frac{2\pi}{4q} nk \right), \quad n = 0, 1, ..., (4q-1) \quad (6.73)$$

Adding (6.70) and (6.68), and dividing the result by two, we can form a new sequence $F_a(n)$ given by

$$F_a(n) = \sum_{k=0}^{q-1} f_a(k) \text{cas}\left(\frac{2\pi}{q} nk \right) \quad (6.74)$$

where

$$f_a(k) = \begin{cases} (-1)^{\frac{k}{4}} a_o(k), & k = 0, 4, 8, 12, ... \\ -(-1)^{\frac{q+1}{2}} (-1)^{\frac{k+2}{4}} a_o(k+2q), & k = 2, 6, 10, 14, ... \\ c_q (-1)^{\frac{k-1}{4}} \left[\frac{1+(-1)^{\frac{q+1}{2}}}{2} a_o(k+q) - \frac{1-(-1)^{\frac{q+1}{2}}}{2} a_o(k+3q) \right], & k = 1, 5, 9, 13, ... \\ c_q (-1)^{\frac{k+1}{4}} \left[\frac{1-(-1)^{\frac{q+1}{2}}}{2} a_o(k+q) - \frac{1+(-1)^{\frac{q+1}{2}}}{2} a_o(k+3q) \right], & k = 3, 7, 11, 15, ... \end{cases} \quad (6.75)$$

Subtracting (6.70) from (6.68) and dividing by two, we can form another sequence $F_s(n)$

given by

$$F_s(n) = \frac{\sqrt{2}}{2} \sum_{k=0}^{q-1} f_s(k) \text{cas}\left(\frac{2\pi}{q} nk\right) \tag{6.76}$$

where

$$f_s(k) = \begin{cases} (-1)^{\frac{k}{4}} c_q b_o(k+q), & k = 0, 4, 8, 12, \ldots \\ -(-1)^{\frac{q+1}{2}} (-1)^{\frac{k+2}{4}} c_q b_e(k+q), & k = 2, 6, 10, 14, \ldots \\ (-1)^{\frac{k-1}{4}} \left[a_o(k) + (-1)^{\frac{q+1}{2}} a_o(k+2q)\right], & k = 1, 5, 9, 13, \ldots \\ (-1)^{\frac{k+1}{4}} \left[a_o(k) - (-1)^{\frac{q+1}{2}} a_o(k+2q)\right], & k = 3, 7, 11, 15, \ldots \end{cases} \tag{6.77}$$

Similarly, using (6.69) and (6.71), we can form two more new sequences given by

$$G_a(q-n) = \frac{\sqrt{2}}{2} \sum_{k=0}^{q-1} g_a(k) \text{cas}\left(\frac{2\pi}{q} nk\right) \tag{6.78}$$

and

$$G_s(q-n) = \sum_{k=0}^{q-1} g_s(k) \text{cas}\left(\frac{2\pi}{q} nk\right) \tag{6.79}$$

where

$$g_a(k) = \begin{cases} -(-1)^{\frac{q+1}{2}} (-1)^{\frac{k}{4}} c_q b_e(k+q), & k = 0, 4, 8, 12, \ldots \\ -(-1)^{\frac{k+2}{4}} c_q b_o(k+q), & k = 2, 6, 10, 14, \ldots \\ (-1)^{\frac{k-1}{4}} \left[a_o(k) - (-1)^{\frac{q+1}{2}} a_o(k+2q)\right], & k = 1, 5, 9, 13, \ldots \\ -(-1)^{\frac{k+1}{4}} \left[a_o(k) + (-1)^{\frac{q+1}{2}} a_o(k+2q)\right], & k = 3, 7, 11, 15, \ldots \end{cases} \tag{6.80}$$

and

$$g_s(k) = \begin{cases} -(-1)^{\frac{q+1}{2}}(-1)^{\frac{k}{4}}a_o(k+2q), & k = 0, 4, 8, 12, \ldots \\ -(-1)^{\frac{k+2}{4}}a_o(k), & k = 2, 6, 10, 14, \ldots \\ c_q(-1)^{\frac{k-1}{4}}\left[\frac{1-(-1)^{\frac{q+1}{2}}}{2}a_o(k+q) - \frac{1+(-1)^{\frac{q+1}{2}}}{2}a_o(k+3q)\right], & k = 1, 5, 9, 13, \ldots \\ -c_q(-1)^{\frac{k+1}{4}}\left[\frac{1+(-1)^{\frac{q+1}{2}}}{2}a_o(k+q) - \frac{1-(-1)^{\frac{q+1}{2}}}{2}a_o(k+3q)\right], & k = 3, 7, 11, 15, \ldots \end{cases} \tag{6.81}$$

Finally, using (6.74), (6.76), (6.78) and (6.79), the four input sequences in (6.67), for the case of $N = 8q$, can be obtained by

$$\begin{bmatrix} F_1(n) \\ G_1(q-n) \\ F_3(n) \\ G_3(q-n) \end{bmatrix} = (\mathbf{H}_2 \otimes \mathbf{I}_2) \begin{bmatrix} F_a(n) \\ G_a(q-n) \\ F_s(n) \\ G_s(q-n) \end{bmatrix}, \quad n = 0, 1, \ldots, q-1 \tag{6.82}$$

It is clear from this decomposition that a length-$8q$ DHT requires one length-$4q$ DHT given by (6.73), two length-q DHTs given by (6.74) and (6.79), two scaled length-q DHTs given by (6.76) and (6.78), and the additions needed in (6.66), (6.72), (6.82) and (6.67). This computation scheme is illustrated in Fig. 6.6.

Let us now consider the general case when $N > 8q$. We denote the input sequences of the four length-$N/8$ DHTs given by (6.68)-(6.71), respectively, by $f_1(k)$, $g_1(k)$, $f_3(k)$ and $g_3(k)$. After certain mathematical transformations, these input sequences may be expressed

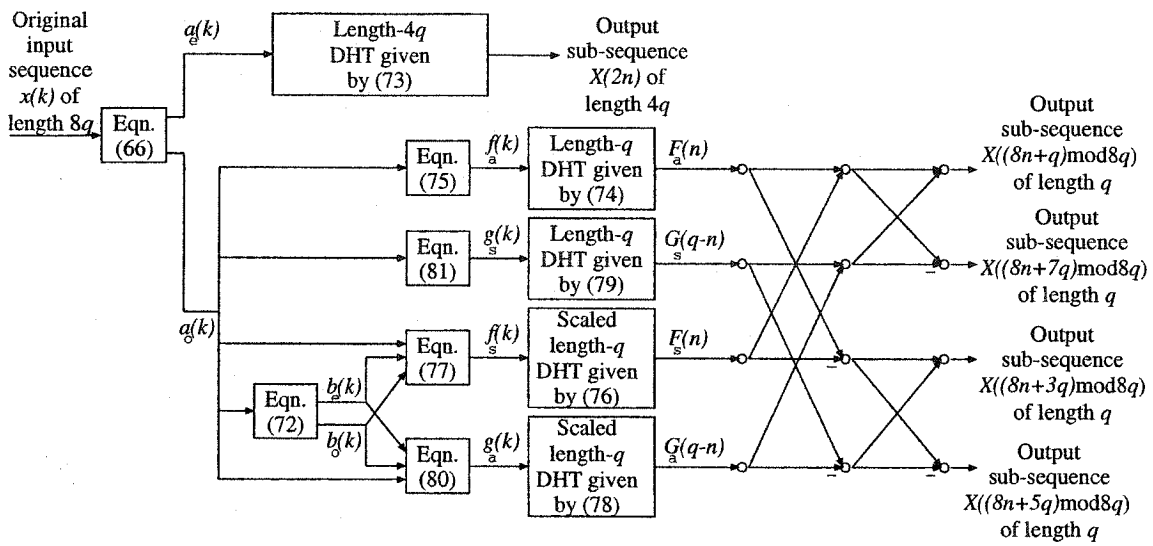

Figure 6.6: Decomposition of a length-$8q$ DHT.

in a matrix form as

$$\begin{bmatrix} f_1(k) \\ g_1(k) \\ f_3(k) \\ g_3(k) \end{bmatrix} = \mathbf{B}_k \begin{bmatrix} a_o(k) \\ a_o(k+N/4) \\ a_o(k+N/8) \\ a_o(k+3N/8) \end{bmatrix}, \quad k = 0, 1, ..., (N/8-1) \tag{6.83}$$

where the matrix $\mathbf{B}_k$ is given by

$$\mathbf{B}_k = \mathbf{F}_k \mathbf{Q} (\mathbf{H}_2 \otimes \mathbf{I}_2) \mathbf{S} \mathbf{Z}, \quad k = 0, 1, ..., (N/8-1) \tag{6.84}$$

In (6.84), the twiddle factor matrix $\mathbf{F}_k$ is given by

$$\mathbf{F}_k = \begin{bmatrix} \cos\left(\frac{2\pi}{N}qk\right) & \sin\left(\frac{2\pi}{N}qk\right) & 0 & 0 \\ \sin\left(\frac{2\pi}{N}qk\right) & -\cos\left(\frac{2\pi}{N}qk\right) & 0 & 0 \\ 0 & 0 & \cos\left(\frac{2\pi}{N}3qk\right) & \sin\left(\frac{2\pi}{N}3qk\right) \\ 0 & 0 & \sin\left(\frac{2\pi}{N}3qk\right) & -\cos\left(\frac{2\pi}{N}3qk\right) \end{bmatrix} \tag{6.85}$$

and the other constant matrices are given by

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & (-1)^{\frac{q+1}{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -(-1)^{\frac{q+1}{2}} \end{bmatrix} \tag{6.86}$$

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2}c_q & 0 \\ 0 & 0 & 0 & \frac{\sqrt{2}}{2}c_q \end{bmatrix} \tag{6.87}$$

$$\mathbf{Z} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{6.88}$$

Finally, the general butterfly of the proposed radix-2/8 FHT algorithm is constructed using

$$\begin{bmatrix} a_e(k) \\ a_e(k+N/8) \\ a_e(k+N/4) \\ a_e(k+3N/8) \\ a_o(k) \\ a_o(k+N/8) \\ a_o(k+N/4) \\ a_o(k+3N/8) \end{bmatrix} = (\mathbf{H}_2 \otimes \mathbf{I}_2 \otimes \mathbf{I}_2) \begin{bmatrix} x(k) \\ x(k+N/8) \\ x(k+N/4) \\ x(k+3N/8) \\ x(k+N/2) \\ x(k+5N/8) \\ x(k+3N/4) \\ x(k+7N/8) \end{bmatrix}, \; k = 0, 1, ..., (N/8-1) \tag{6.89}$$

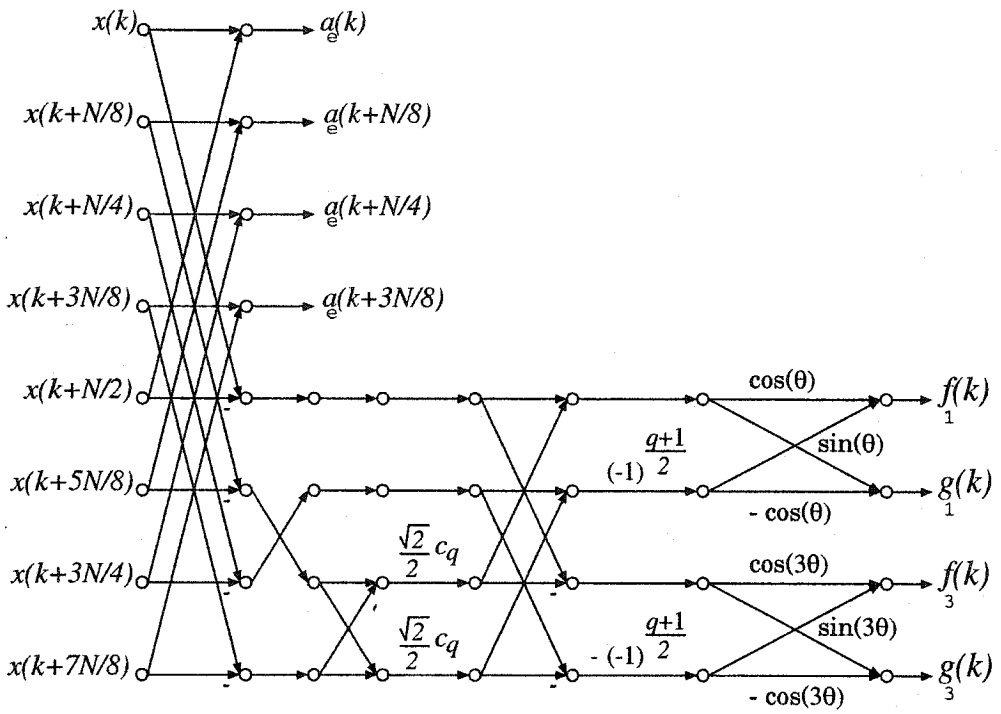

Figure 6.7: General butterfly of the proposed radix-2/8 FHT algorithm, $\theta = 2\pi qk/N$.

and the general sub-butterfly given by (6.83). The flowgraph of this general butterfly is illustrated in Fig. 6.7.

The appearance of the parameter q in the elements of the twiddle factor matrix given by (6.85), allows us to define $2q$ special butterflies that can be indexed by $k = p\frac{N}{16q}$, where $p = 0, 1, ..., 2q-1$. Then, for these cases, (6.85) reduces to

$$\mathbf{F}_{p\frac{N}{16q}} = \begin{bmatrix} \cos\left(p\frac{\pi}{8}\right) & \sin\left(p\frac{\pi}{8}\right) & 0 & 0 \\ \sin\left(p\frac{\pi}{8}\right) & -\cos\left(p\frac{\pi}{8}\right) & 0 & 0 \\ 0 & 0 & \cos\left(p\frac{3\pi}{8}\right) & \sin\left(p\frac{3\pi}{8}\right) \\ 0 & 0 & \sin\left(p\frac{3\pi}{8}\right) & -\cos\left(p\frac{3\pi}{8}\right) \end{bmatrix}, \; p = 0, 1, ..., (2q-1) \tag{6.90}$$

It is clear that the operations introduced by (6.90) are trivial for the special butterflies corresponding to $p = 0, 4, 8, 12, ...$ For the special butterflies corresponding to other even

values of p, matrix $\mathbf{B}_k$ given by (6.84) can be rearranged as

$$\mathbf{B}_{p\frac{N}{16q}} = (-1)^{\frac{p-2}{8}} \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}(-1)^{\frac{q+1}{2}} & c_q\frac{1+(-1)^{\frac{q+1}{2}}}{2} & -c_q\frac{1-(-1)^{\frac{q+1}{2}}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2}(-1)^{\frac{q+1}{2}} & c_q\frac{1-(-1)^{\frac{q+1}{2}}}{2} & -c_q\frac{1+(-1)^{\frac{q+1}{2}}}{2} \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2}(-1)^{\frac{q+1}{2}} & c_q\frac{1+(-1)^{\frac{q+1}{2}}}{2} & -c_q\frac{1-(-1)^{\frac{q+1}{2}}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2}(-1)^{\frac{q+1}{2}} & -c_q\frac{1-(-1)^{\frac{q+1}{2}}}{2} & c_q\frac{1+(-1)^{\frac{q+1}{2}}}{2} \end{bmatrix} \tag{6.91}$$

for $p = 2, 10, 18, ...,$ and

$$\mathbf{B}_{p\frac{N}{16q}} = (-1)^{\frac{p-6}{8}} \begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}(-1)^{\frac{q+1}{2}} & -c_q\frac{1-(-1)^{\frac{q+1}{2}}}{2} & c_q\frac{1+(-1)^{\frac{q+1}{2}}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}(-1)^{\frac{q+1}{2}} & c_q\frac{1+(-1)^{\frac{q+1}{2}}}{2} & -c_q\frac{1-(-1)^{\frac{q+1}{2}}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2}(-1)^{\frac{q+1}{2}} & -c_q\frac{1-(-1)^{\frac{q+1}{2}}}{2} & c_q\frac{1+(-1)^{\frac{q+1}{2}}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}(-1)^{\frac{q+1}{2}} & -c_q\frac{1+(-1)^{\frac{q+1}{2}}}{2} & c_q\frac{1-(-1)^{\frac{q+1}{2}}}{2} \end{bmatrix} \tag{6.92}$$

for $p = 6, 14, 22, ...$ Now, for the special butterflies corresponding to odd values of p, using Lemma 2 and carrying out some mathematical manipulations, matrix $\mathbf{B}_k$ given by (6.84) can be factored as

$$\mathbf{B}_{p\frac{N}{16q}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & (-1)^{\frac{p-1}{2}} & 0 \\ 0 & 0 & 0 & (-1)^{\frac{p-1}{2}} \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \cos(\theta) & i_q\sin(\theta) & 0 & 0 \\ \sin(\theta) & -i_q\cos(\theta) & 0 & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2}c_q\left(\cos(\theta)+i_q\sin(\theta)\right) & -\frac{\sqrt{2}}{2}c_q\left(\cos(\theta)-i_q\sin(\theta)\right) \\ 0 & 0 & \frac{\sqrt{2}}{2}c_q\left(\sin(\theta)-i_q\cos(\theta)\right) & -\frac{\sqrt{2}}{2}c_q\left(\sin(\theta)+i_q\cos(\theta)\right) \end{bmatrix} \tag{6.93}$$

where $\theta = p\frac{\pi}{8}$ and $i_q = (-1)^{\frac{q+1}{2}}$. The number of operations to process each of the special butterflies defined above is less than that needed to process the general butterfly, as will be seen in Section 6.3.2.

It is seen from (6.85) that the general butterfly of the proposed FHT algorithm requires an operation of the type

$$\begin{bmatrix} D_1 \\ D_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(\theta) & -\cos(\theta) \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \tag{6.94}$$

which needs four multiplications and two additions (4mult-2add scheme). The implementation of (6.94) using the 3mult-3add scheme can be achieved by rearranging (6.94) in the form

$$\begin{bmatrix} D_1 \\ D_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \text{cas}(-\theta) & 0 & 0 \\ 0 & \sin(\theta) & 0 \\ 0 & 0 & \text{cas}(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \tag{6.95}$$

In order to perform the general butterfly of the proposed FHT algorithm using the 3mult-3add scheme given by (6.95), we factor the twiddle factor matrix given by (6.85) as

$$\mathbf{F}_k = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} \cas(-\theta) & 0 & 0 & 0 & 0 & 0 \\ 0 & \sin(\theta) & 0 & 0 & 0 & 0 \\ 0 & 0 & \cas(\theta) & 0 & 0 & 0 \\ 0 & 0 & 0 & \cas(-3\theta) & 0 & 0 \\ 0 & 0 & 0 & 0 & \sin(3\theta) & 0 \\ 0 & 0 & 0 & 0 & 0 & \cas(3\theta) \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \theta = \frac{2\pi}{N} qk \tag{6.96}$$

For the special butterflies corresponding to the even values of p, we use the matrices given by (6.90), (6.91) and (6.92). However, for the butterflies corresponding to the odd values of p, we rearrange the matrix given by (6.93) as

$$
\mathbf{B}_{p\frac{N}{16q}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & (-1)^{\frac{p-1}{2}} & 0 \\ 0 & 0 & 0 & (-1)^{\frac{p-1}{2}} \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix}
$$

$$
\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \text{cas}(-\theta) & 0 & 0 & 0 & 0 & 0 \\ 0 & \sin(\theta) & 0 & 0 & 0 & 0 \\ 0 & 0 & \text{cas}(\theta) & 0 & 0 & 0 \\ 0 & 0 & 0 & A & 0 & 0 \\ 0 & 0 & 0 & 0 & B & 0 \\ 0 & 0 & 0 & 0 & 0 & C \end{bmatrix}
$$

$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & i_q & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i_q \end{bmatrix} \tag{6.97}
$$

where

$\theta = p\frac{\pi}{8}$, $i_q = (-1)^{\frac{q+1}{2}}$, $A = \frac{\sqrt{2}}{2}c_q\left(\left(1+i_q\right)\cos(\theta) - \left(1-i_q\right)\sin(\theta)\right)$,

$B = \frac{\sqrt{2}}{2}c_q\left(\sin(\theta) - i_q\cos(\theta)\right)$ and $C = \frac{\sqrt{2}}{2}c_q\left(\left(1-i_q\right)\cos(\theta) + \left(1+i_q\right)\sin(\theta)\right)$.

We now summarize the scheme of the proposed radix-2/8 FHT algorithm for computing a length-$q \times 2^m$ DHT. In this decomposition, the computation of a length-N DHT may be considered as being decomposed into five DHTs, one of length $N/2$ given by (6.61) to compute the even-indexed terms and the others of length $N/8$ given by (6.68)-(6.71) to be used in (6.67) for computing the odd-indexed terms. This is achieved by using the

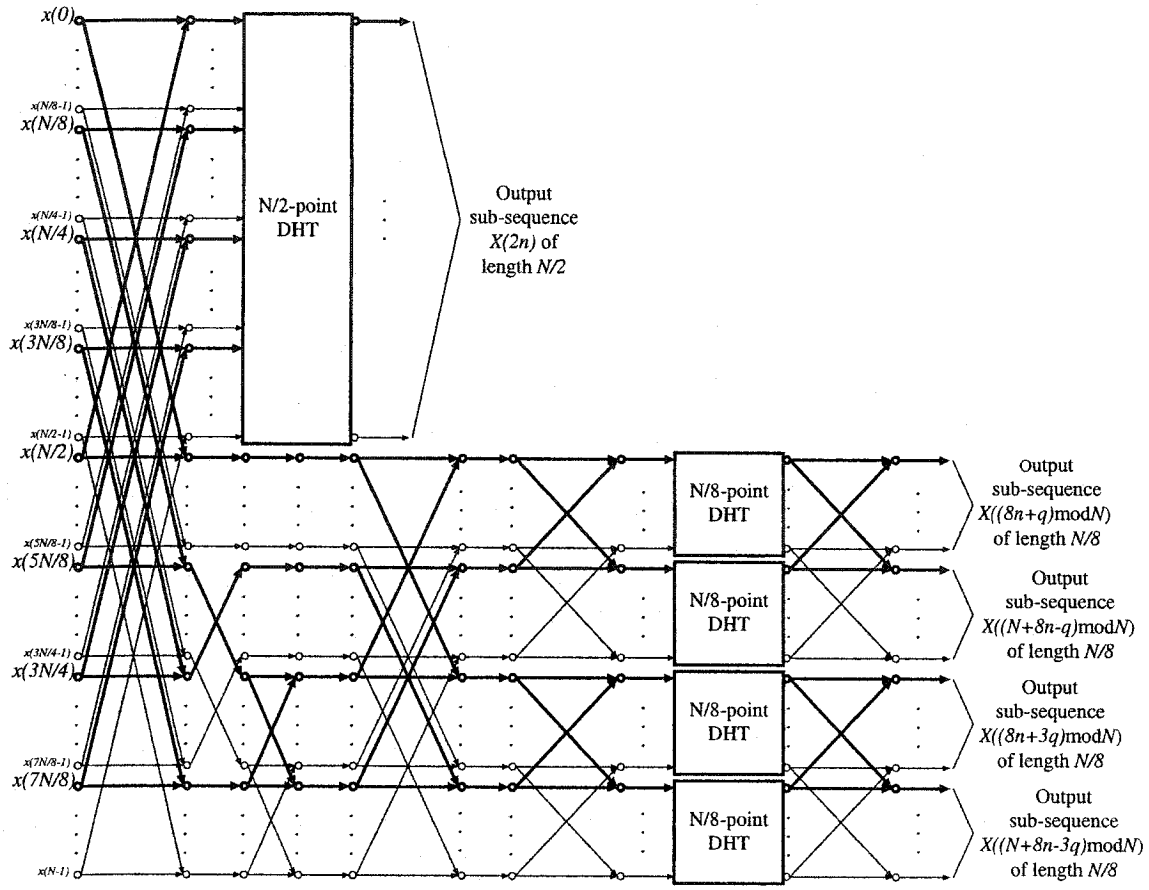

Figure 6.8: Decomposition scheme of the proposed radix-2/8 FHT algorithm.

butterfly based on (6.89) and (6.83). The required number of butterflies is $\left(\frac{N}{8}-2q\right)$ general butterflies and $2q$ special butterflies corresponding to $k = p\frac{N}{16q}$, where $p = 0, 1, ..., 2q-1$. This process is repeated successively for each of the new resulting DHTs, until the size is reduced to a $4q$- , $2q$- or q-point DHT. This scheme is illustrated in Fig. 6.8. Note that, for simplicity, the gains (weights) are omitted in Fig. 6.8 and can easily be obtained from Fig. 6.7.

6.3.2 Computational Complexity

In this section, we consider the performance of the proposed radix-2/8 FHT algorithm for the computation of a length-$q \times 2^m$ DHT by analyzing its computational complexity and comparing it with that of the existing radix-2/4 FHT algorithm reported in [55]-[57].

6.3.2.1 Arithmetic Complexity

The arithmetic complexities of $2q$-, $4q$-, and $8q$-point DHTs can easily be obtained, from Figs. 6.4, 6.5 and 6.6 respectively. These arithmetic complexities are respectively $2M_q$, $4M_q$ and $6M_q+2M_q^s$ multiplications, and $2A_q+2q$, $4A_q+8q$ and $8A_q+26q$ additions, where M_q^s denotes the number of multiplications required by a scaled length-q DHT, and M_q and A_q denote, respectively, the numbers of multiplications and additions required by a length-q DHT.

Let us now discuss the arithmetic complexity of a length-N DHT, when $N > 8q$. If the 4mult-2add scheme is considered, the general butterfly of the proposed radix-2/8 FHT algorithm based on (6.89) and (6.83) requires 10 multiplications and 18 additions; 10 multiplications and 10 additions are introduced by the matrix (6.84) in the computation of (6.83), whereas the matrix $(\mathbf{H}_2 \otimes \mathbf{I}_2 \otimes \mathbf{I}_2)$ introduces only 8 additions in the computation of (6.89). The algorithm has $2q$ special butterflies that can be indexed by $k = p\frac{N}{16q}$, where $p = 0, 1, ..., (2q-1)$. The reduction in the computation achieved by these special butterflies is due to the fact that the matrix used for computing (6.83) is different from that used by the general butterfly. Since (6.90) does not introduce any operation in the computation of (6.84) for $p = 0, 4, 8, 12, ...$, each of the corresponding special butterflies requires 2 multiplications and 14 additions. The special butterflies corresponding to $p = 2, 10, 18, ...$ and $p = 6, 14, 22, ...$ use, respectively, (6.91) and (6.92) to compute (6.83). Since q is an odd integer, if $\frac{1+(-1)^{\frac{q+1}{2}}}{2}$ equals one, then $\frac{1-(-1)^{\frac{q+1}{2}}}{2}$ equals zero and vis versa. Using this fact, (6.91) and (6.92) each requires only 2 multiplications and 6 additions. Therefore, each of

the special butterflies corresponding to $p = 2, 10, 18, \ldots$ and $p = 6, 14, 22, \ldots$ requires 2 multiplications and 14 additions. Since for odd values of p, the matrix given by (6.93) is used for computing (6.83), each of the corresponding special butterflies requires 8 multiplications and 16 additions. The proposed decomposition consists of dividing a length-N DHT to one length-$N/2$ DHT and four length-$N/8$ DHTs. This is achieved by performing $\left(\frac{N}{8} - 2q\right)$ general butterflies and the $2q$ special butterflies. The odd-indexed terms are obtained by performing the extra $N/2$ additions needed by (6.67). The decomposition process is repeated successively for each of the new resulting DHTs until the size is reduced to a $8q$-, $4q$-, $2q$- or q-point DHT. Therefore, it is seen that the expressions for the number of multiplications and additions of the proposed FHT algorithm are, respectively,

$$\mathrm{M}^{42}_{\mathrm{r2}/8}(1,N) = \frac{5}{4}N - 10q + \mathrm{M}^{42}_{\mathrm{r2}/8}\left(1,\frac{N}{2}\right) + 4\mathrm{M}^{42}_{\mathrm{r2}/8}\left(1,\frac{N}{8}\right),\ N > 8q \tag{6.98}$$

$$\mathrm{A}^{42}_{\mathrm{r2}/8}(1,N) = \frac{11}{4}N - 6q + \mathrm{A}^{42}_{\mathrm{r2}/8}\left(1,\frac{N}{2}\right) + 4\mathrm{A}^{42}_{\mathrm{r2}/8}\left(1,\frac{N}{8}\right),\ N > 8q \tag{6.99}$$

Similarly, if the 3mult-3add scheme is considered, the general butterfly uses (6.96) in the computation of (6.84). Then, it requires 8 multiplications and 20 additions. The special butterflies corresponding to the even values of p have the same number of multiplications and additions as in the case of the 4mult-2add scheme. However, for the odd values of p, the matrix given by (6.97) is used to compute (6.83). This involves the computation of each of the corresponding special butterflies using 6 multiplications and 18 additions. It can be shown that, for this scheme, the number of multiplications and additions of the proposed FHT algorithm are, respectively,

$$\mathrm{M}^{33}_{\mathrm{r2}/8}(1,N) = N - 8q + \mathrm{M}^{33}_{\mathrm{r2}/8}\left(1,\frac{N}{2}\right) + 4\mathrm{M}^{33}_{\mathrm{r2}/8}\left(1,\frac{N}{8}\right),\ N > 8q \tag{6.100}$$

$$\mathrm{A}^{33}_{\mathrm{r2}/8}(1,N) = 3N - 8q + \mathrm{A}^{33}_{\mathrm{r2}/8}\left(1,\frac{N}{2}\right) + 4\mathrm{A}^{33}_{\mathrm{r2}/8}\left(1,\frac{N}{8}\right),\ N > 8q \tag{6.101}$$

Table 6.13: Number of arithmetic operations required for the computation of the 1-D DHT for $q = 1$

		Radix-2/4 FHT algorithm in [55]-[57]					Proposed radix-2/8 FHT algorithm				
		Using 3mult-3add scheme		Using 4mult-2add scheme		Total	Using 3mult-3add scheme		Using 4mult-2add scheme		Total
m	N	Mults.	Mults.	Mults.	Adds.	M.+A.	Mults.	Adds.	Mults.	Adds.	M.+A.
3	8	2	26	2	26	28	2	26	2	26	28
4	16	10	74	12	72	84	10	74	12	72	84
5	32	34	194	42	186	228	34	194	42	186	228
6	64	98	482	124	456	580	98	482	120	460	580
7	128	256	1154	330	1082	1412	256	1154	318	1094	1412
8	256	642	2690	828	2504	3332	642	2690	796	2536	3332
9	512	1538	6146	1994	5690	7684	1538	6146	1906	5778	7684
10	1024	3586	13826	4668	12744	17412	3586	13826	4448	12964	17412
11	2048	8194	30722	10698	28218	38916	8194	30722	10182	28734	38916

The arithmetic complexities of the proposed radix-2/8 FHT algorithm and the existing radix-2/4 FHT algorithm reported in [55]-[57] for various values of N are given in Tables 6.13-6.16. In Tables 6.14-6.16, the values used for M_q^s, M_q and A_q for the proposed FHT algorithm are shown in Table 6.17, for $q = 3, 9$ and 15. These values are the same as the ones used in [55] and [56] for the radix-2/4 FHT algorithm for the corresponding values of q. We see from Tables 6.13-6.16 that, in general, the total number of arithmetic operations is the same for both the proposed radix-2/8 FHT and the radix-2/4 FHT algorithms. It can be verified that the radix-2/4 FHT algorithms of [55]-[57] also have exactly $2q$ special butterflies that are precisely indexed by $p\frac{N}{8q}$, where $p = 0, 1, ..., (2q-1)$.

Table 6.14: Number of arithmetic operations required for the computation of the 1-D DHT for $q = 3$

		Radix-2/4 FHT algorithm in [55]-[57]					Proposed radix-2/8 FHT algorithm				
		Using 3mult-3add scheme		Using 4mult-2add scheme		Total	Using 3mult-3add scheme		Using 4mult-2add scheme		Total
m	N	Mults.	Mults.	Mults.	Adds.	M.+A.	Mults.	Adds.	Mults.	Adds.	M.+A.
1	6	2	18	2	18	20	2	18	2	18	20
2	12	4	48	4	48	52	4	48	4	48	52
3	24	12	126	12	126	138	12	126	12	126	138
4	48	44	318	50	312	362	44	318	50	312	362
5	96	128	774	152	750	902	132	774	156	750	906
6	192	348	1830	426	1752	2178	348	1830	414	1764	2178
7	384	880	4230	1096	4014	5110	884	4230	1064	4050	5114
8	768	2140	9606	2698	9048	11746	2156	9606	2618	9144	11762
9	1536	5040	21510	6408	20142	26550	5060	21510	6164	20406	26570

Table 6.15: Number of arithmetic operations required for the computation of the 1-D DHT for $q = 9$

		Radix-2/4 FHT algorithm in [55]-[57]					Proposed radix-2/8 FHT algorithm				
		Using 3mult-3add scheme		Using 4mult-2add scheme		Total	Using 3mult-3add scheme		Using 4mult-2add scheme		Total
m	N	Mults.	Mults.	Mults.	Adds.	M.+A.	Mults.	Adds.	Mults.	Adds.	M.+A.
1	18	20	98	20	98	118	20	98	20	98	118
2	36	40	232	40	232	272	40	232	40	232	272
3	72	86	554	86	554	640	86	554	86	554	640
4	144	238	1306	256	1288	1544	238	1306	256	1288	1544
5	288	590	3026	662	2954	3616	614	3026	686	2954	3640
6	576	1462	6898	1696	6664	8360	1462	6898	1660	6700	8360
7	1152	3470	15506	4118	14858	18976	3494	15506	4034	14966	19000
8	2304	8086	34450	9760	32776	42536	8182	34450	9568	33064	42632

Table 6.16: Number of arithmetic operations required for the computation of the 1-D DHT for $q = 15$

		Radix-2/4 FHT algorithm in [55]-[57]					Proposed radix-2/8 FHT algorithm				
		Using 3mult-3add scheme		Using 4mult-2add scheme		Total	Using 3mult-3add scheme		Using 4mult-2add scheme		Total
m	N	Mults.	Mults.	Mults.	Adds.	M.+A.	Mults.	Adds.	Mults.	Adds.	M.+A.
1	30	36	192	36	192	228	36	192	36	192	228
2	60	72	444	72	444	516	72	444	72	444	516
3	120	156	1038	156	1038	1194	156	1038	156	1038	1194
4	240	420	2406	450	2376	2826	420	2406	450	2376	2826
5	480	1032	5502	1152	5382	6534	1068	5502	1188	5382	6570
6	960	2532	12414	2922	12024	14946	2532	12414	2862	12084	14946
7	1920	5976	27678	7056	26598	33654	6012	27678	6912	26778	33690

Table 6.17: Number of multiplications and additions required by 3-, 9- and 15-point DHTs

q	M_q	M_q^s	A_q
3	1	3	6
9	10	13	40
15	18	24	81

6.3.2.2 Data Transfers

The butterfly, shown in Fig. 6.7, of the proposed FHT algorithm, is based on (6.89) and (6.83). We assume that sufficient registers are available in the processor to compute all the operations required by the butterfly without using any intermediate transfer operations between the processor and the external memory (RAM). The implementation of the butterfly for a given value of k, consists of reading two points from the external memory of the processor and performing the operations of addition and subtraction using these two points as the operands. The result of the addition is returned to the external memory, whereas that of the subtraction is kept in an internal register of the processor. This process is repeated four times to compute a_e and a_o given by (6.89) of the butterfly. The four points of a_o kept in the processor are used to process the sub-butterfly given by (6.83). Therefore, the butterfly requires 8 transfer operations (read and write). The results of the additions for $k = 0, 1, ..., (N/8) - 1$ returned to the external memory are grouped to form the first sub-sequence $a_e(k)$ of length $N/2$. The four sub-sequences $f_1(k)$, $g_1(k)$, $f_3(k)$ and $g_3(k)$ of length $N/8$ are formed by grouping separately the results of each row of the output vector of (6.83). This is achieved by using $N/8$ butterflies. This scheme of implementation reduces significantly the number of data transfers and address generations. To compare the number of data transfers and address generations of the proposed radix-2/8 FHT algorithm with that of the radix-2/4 FHT algorithm of [55]-[57], we consider the case of $q = 1$ as an example. Since, for a length-8 DHT, the DHTs given by (6.68)-(6.71) become length-1 DHTs, the four resulting points can be kept in the processor to compute (6.67) without any extra transfer operation. However, for a length-N DHT ($N > 8$), the extra $N/2$ transfer operations of (67) are required. The proposed decomposition and that reported in [55]-[57] require four types of butterflies: one type to compute length-2 DHTs, two types of special butterflies and one general butterfly. However, the proposed decomposition requires one additional type of butterfly to compute a length-4 DHT. It can be shown that the expression

for the number of data transfer operations, not including the read operations for the twiddle factors, for the implementation of the proposed radix-2/8 FHT algorithm is given by

$$\begin{aligned} D_{r2/8}(1,N) &= \frac{3N}{2} + D_{r2/8}\left(1,\frac{N}{2}\right) + 4D_{r2/8}\left(1,\frac{N}{8}\right),\ N > 8, \\ D_{r2/8}(1,8) &= 12, D_{r2/8}(1,4) = 4, D_{r2/8}(1,2) = 2, D_{r2/8}(1,1) = 0 \end{aligned} \tag{6.102}$$

whereas the corresponding expression for the implementation of the radix-2/4 FHT algorithm using the first four types of butterflies is given by

$$\begin{aligned} D_{r2/4}(1,N) &= \frac{3N}{2} + D_{r2/4}\left(1,\frac{N}{2}\right) + 2D_{r2/4}\left(1,\frac{N}{4}\right),\ N \geq 8, \\ D_{r2/4}(1,4)\,D_4^{2/4} &= 6, D_{r2/4}(1,2) = 2, D_{r2/4}(1,1) = 0 \end{aligned} \tag{6.103}$$

If in the implementation of the latter algorithm, all the five types of butterflies are used, then the corresponding expression is given by

$$\begin{aligned} D_{r2/4}(1,N) &= \frac{3N}{2} + D_{r2/4}\left(1,\frac{N}{2}\right) + 2D_{r2/4}\left(1,\frac{N}{4}\right),\ N \geq 8, \\ D_{r2/4}(1,4) &= 4, D_{r2/4}(1,2) = 2, D_{r2/4}(1,1) = 0 \end{aligned} \tag{6.104}$$

The number of data transfer operations for the two algorithms are compared in Table 6.18. It is seen from this table that the implementation of the proposed FHT algorithm provides around 25% savings over the 4-butterfly and 5-butterfly implementations of the radix-2/4 FHT algorithm of [55]-[57]. During the data transfer process, the address generation operations are required to read/write the data from/in memory, and hence, similar savings are also obtained by the proposed FHT algorithm in the case of the address generation.

Table 6.18: Comparison of the number of data transfer operations for $q = 1$

		4-butterfly implementation of the radix-2/4 FHT	5-butterfly implementation of the radix-2/4 FHT	Implementation of the proposed radix-2/8 FHT	Savings in C over A	Savings in C over B
m	N	$D_{r2/4}(1,N)$ (A)	$D_{r2/4}(1,N)$ (B)	$D_{r2/8}(1,N)$ (C)	%	%
3	8	22	20	12	45.45	40.00
4	16	58	52	44	24.13	15.38
5	32	150	140	108	28.00	22.85
6	64	362	340	252	30.38	25.88
7	128	854	812	620	27.40	23.64
8	256	1962	1876	1436	26.80	23.45
9	512	4438	4268	3212	27.62	24.74
10	1024	9898	9556	7228	26.97	24.36
11	2048	21846	21164	16044	26.55	24.19
12	4096	47786	46420	35036	26.68	24.52

6.3.2.3 Twiddle Factors

We discuss here only the cases of $q = 1$ and $q = 3$. In counting the number of twiddle factor evaluations or accesses to the lookup table required by the proposed radix-2/8 FHT algorithm and the radix-2/4 FHT algorithm of [55]-[57], it is assumed that the twiddle factors required by the special butterflies, such as $\sqrt{2}/2$, $\sqrt{3}/2$, $\cos(\pi/8)$, $\sin(\pi/8)$, $\text{cas}(-\pi/8)$ and $\text{cas}(\pi/8)$ are initialized and kept in the internal registers of the processor during the processing time of the corresponding algorithm. However, for the radix-2/8 and radix-2/4 FHT algorithms, 4 and 2 twiddle factors are respectively required to be evaluated or loaded from the lookup table by the corresponding general butterfly using the 4mult-2add scheme. The corresponding numbers are 6 and 3 for the 3mult-3add scheme. Therefore, by counting the required number of the general butterflies in the radix-2/8 and radix-2/4 FHT algorithms, the number of twiddle factor evaluations or accesses to the lookup table for the

two algorithms are, respectively

$$T^{42}_{r2/8}(1,N) = \frac{N}{2} - 8q + T^{42}_{r2/8}\left(1,\frac{N}{2}\right) + 4T^{42}_{r2/8}\left(1,\frac{N}{8}\right),\ N > 8q \tag{6.105}$$

and

$$T^{42}_{r2/4}(1,N) = \frac{N}{2} - 4q + T^{42}_{r2/4}\left(1,\frac{N}{2}\right) + 2T^{42}_{r2/4}\left(1,\frac{N}{4}\right),\ N > 4q \tag{6.106}$$

using the 4mult-2add scheme, and

$$T^{33}_{r2/8}(1,N) = \frac{3}{4}N - 12q + T^{33}_{r2/8}\left(1,\frac{N}{2}\right) + 4T^{33}_{r2/8}\left(1,\frac{N}{8}\right),\ N > 8q \tag{6.107}$$

and

$$T^{33}_{r2/4}\left(1,\frac{N}{2}\right) = \frac{3}{4}N - 6q + T^{33}_{r2/4}\left(1,\frac{N}{2}\right) + 2T^{33}_{r2/4}\left(1,\frac{N}{4}\right),\ N > 4q \tag{6.108}$$

using the 3mult-3add scheme. For various values of N, the number of twiddle factor evaluations or accesses to the lookup table of the proposed radix-2/8 FHT algorithm is compared with that of the radix-2/4 FHT algorithm when $q = 1$ in Table 6.19 and $q = 3$ in Table 6.20. It is seen from these tables that savings of over 30% in the evaluation of twiddle factors or in the access to the lookup table can easily be achieved by the proposed FHT algorithm. Note that, when the lookup table is used, identical savings are obtained by the proposed FHT algorithm in the address generation for reading the twiddle factors.

Table 6.19: Comparison of the number of twiddle factor evaluations or accesses to the lookup table for $q = 1$

		Radix-2/4 FHT	Proposed radix-2/8 FHT	Savings	Radix-2/4 FHT	Proposed radix-2/8 FHT	Savings
m	N	$T^{42}_{r2/4}(1,N)$	$T^{42}_{r2/8}(1,N)$	%	$T^{33}_{r2/4}(1,N)$	$T^{33}_{r2/8}(1,N)$	%
3	8	0	0	00.00	0	0	00.00
4	16	4	0	100	6	0	100
5	32	16	8	50.00	24	12	50.00
6	64	52	32	38.46	78	48	38.46
7	128	144	88	38.88	216	132	38.38
8	256	372	240	35.48	558	360	35.48
9	512	912	616	32.45	1368	924	32.45
10	1024	2164	1472	31.97	3246	2208	31.97
11	2048	5008	3448	31.15	7512	5172	31.15
12	4096	11380	7952	30.12	17070	11928	30.12

Table 6.20: Comparison of the number of twiddle factor evaluations or accesses to the lookup table for $q = 3$

		Radix-2/4 FHT	Proposed radix-2/8 FHT	Savings	Radix-2/4 FHT	Proposed radix-2/8 FHT	Savings
m	N	$T^{42}_{r2/4}(1,N)$	$T^{42}_{r2/8}(1,N)$	%	$T^{33}_{r2/4}(1,N)$	$T^{33}_{r2/8}(1,N)$	%
3	24	0	0	00.00	0	0	00.00
4	48	12	0	100	18	0	100
5	96	48	24	50.00	72	36	50.00
6	192	156	96	38.46	234	144	38.46
7	384	432	264	38.88	648	396	38.38
8	768	1116	720	35.48	1674	1080	35.48
9	1536	2736	1848	32.45	4104	2772	32.45
10	3072	6492	4416	31.97	9738	6624	31.97
11	6144	15024	10344	31.15	22536	15516	31.15
12	12288	34140	23856	30.12	51210	35784	30.12

6.4 Summary

In this chapter, we have proposed a new radix-2/8 approach for computing the DFT and DHT of arbitrary lengths $N = q \times 2^m$, where q and m are integers. It is based on the use of a mixture of radix-2 and radix-8 index maps. This mixture and the use of an efficient indexing process provide decompositions different from those of the existing radix-2/4 FFT and FHT algorithms, since the existing ones are all based on the use of a mixture of radix-2 and radix-4 index maps. This new approach has been used to design efficient radix-2/8 FFT and FHT algorithms. It has been shown that, in most cases, the number of arithmetic operations required in the proposed radix-2/8 FFT (or FHT) algorithm is the same as that in the existing radix-2/4 FFT (or FHT) algorithm. However, the other operations such as data transfer, address generation, and twiddle factor evaluation or access to the lookup table, which increase significantly the complexity of FFT and FHT algorithms, are all substantially reduced in the proposed radix-2/8 algorithms. This is of particular importance in the case of $q = 1$, when the data length becomes an integral power of two, and the number of arithmetic operations required by the proposed radix-2/8 FFT (or FHT) is identical to that required by the existing radix-2/4 FFT (or FHT) algorithm. The proposed algorithms are expressed in simple matrix forms, thereby facilitating easy implementations of the algorithms, and allowing for extensions to the multidimensional case.

Chapter 7

Extension of the Radix-2/8 Approach for the Computation of DFTs of Higher Dimensions

7.1 Introduction

In the conventional row-column method, the 2-D and 3-D DFTs are computed by a series of 1-D FFT algorithms applied successively in each of the dimensions [26]. The 2-D and 3-D DFTs can also be computed using the polynomial transform [25], [34]. Although the polynomial transform algorithms require the lowest number of multiplications, they are not popular in view of their complicated structures compared to the existing radix-based FFT algorithms. We now give a brief review of the radix-2 and radix-2/4 approaches that have been exploited in developing two and higher dimension FFT algorithms.

The radix-2 approach of Cooley-Tukey [3] has been used to develop the radix-(2×2) FFT algorithm for computing the 2-D DFT [22]. This has been generalized for the radix-$(r_1 \times r_2)$ FFT algorithm [23], where r_1 and r_2 are arbitrary integers. Later, the approach of

Cooley-Tukey was extended for the computation of the DFT of an arbitrary dimension [24], [26], [32]. These algorithms, known as vector radix FFT algorithms, substantially reduce the number of arithmetic operations over the row-column method.

The radix-2/4 approach of Duhamel-Hollmann [11] has been extended for the 2-D DFT case using DIT decomposition [39]. This provides the 2-D radix-$(2 \times 2)/(4 \times 4)$ DIT FFT algorithm, which is based on the use of a mixture of radix-(2×2) and radix-(4×4) index maps. The 2-D radix-$(2 \times 2)/(4 \times 4)$ DIF version of the DIT algorithm in [39] has been formulated in a matrix form in [28]. Alternate versions of these radix-$(2 \times 2)/(4 \times 4)$ FFT algorithms have been given in [29] using an algebraic formulation. In Chapter 5, we have proposed an efficient radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ DIF FFT algorithm for computing the 3-D DFT. All the above FFT algorithms that are based on the radix-2/4 approach significantly reduce the computational complexity as compared to the corresponding vector radix FFT algorithms.

It has been shown that the new radix-2/8 approach introduced in Chapter 6 is superior to all the other existing radix-based approaches in providing efficient 1-D FFT and FHT algorithms. In this chapter, by exploring the radix-2/8 approach, the problem of reducing the complexity in the computation of the 2-D and 3-D DFTs is addressed, and efficient radix-$(2 \times 2)/(8 \times 8)$ [125] and radix-$(2 \times 2 \times 2)/(8 \times 8 \times 8)$ [117] FFT algorithms are proposed. An appropriate use of the Kronecker product coupled with an efficient index mapping enable us to characterize the butterflies of the proposed algorithms by simple closed-form expressions that facilitate easy software or hardware implementations of the algorithms. Sections 7.2 and 7.3 consider the cases of the 2-D and 3-D DFTs, respectively

7.2 Proposed Radix-$(2 \times 2)/(8 \times 8)$ DIF FFT Algorithm

From (3.15), the 2-D DFT of size $(N \times N)$, is given by

$$\widetilde{X}(n_1, n_2) = \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} \widetilde{x}(k_1, k_2) W_N^{n_1 k_1 + n_2 k_2}, \; 0 \leq n_1, n_2 \leq N-1 \tag{7.1}$$

In this section, we introduce a new DIF decomposition of the 2-D DFT given by (7.1) using the radix-2/8 approach. This decomposition is performed using two steps. In the first step, we use radix-(2×2) index maps to decompose the DFT given by (7.1) into four DFTs. In the second step, we introduce radix-(8×8) index maps by further decomposing some of these resulting DFTs.

Let us first introduce radix-(2×2) index maps by changing the variables k_i and n_i in (7.1) by

$$k_i + \frac{N}{2} l_i, \; 0 \leq k_i \leq \frac{N}{2} - 1, \; l_i = 0, 1 \tag{7.2}$$

and

$$2n_i + m_i, \; 0 \leq n_i \leq \frac{N}{2} - 1, \; m_i = 0, 1 \tag{7.3}$$

respectively. Then, (7.1) is decomposed into four $\left(\frac{N}{2} \times \frac{N}{2}\right)$-point DFTs given by

$$\widetilde{X}(2n_1, 2n_2) = \sum_{k_1=0}^{N/2-1} \sum_{k_2=0}^{N/2-1} \widetilde{y}_{00}(k_1, k_2) W_{N/2}^{\sum_{i=1}^{2} n_i k_i}, \; 0 \leq n_i \leq \frac{N}{2} - 1 \tag{7.4}$$

$$\widetilde{X}(2n_1, 2n_2 + 1) = \sum_{k_1=0}^{N/2-1} \sum_{k_2=0}^{N/2-1} \widetilde{y}_{01}(k_1, k_2) W_N^{k_2} W_{N/2}^{\sum_{i=1}^{2} n_i k_i}, \; 0 \leq n_i \leq \frac{N}{2} - 1 \tag{7.5}$$

$$\widetilde{X}(2n_1 + 1, 2n_2) = \sum_{k_1=0}^{N/2-1} \sum_{k_2=0}^{N/2-1} \widetilde{y}_{10}(k_1, k_2) W_N^{k_1} W_{N/2}^{\sum_{i=1}^{2} n_i k_i}, \; 0 \leq n_i \leq \frac{N}{2} - 1 \tag{7.6}$$

$$\widetilde{X}(2n_1+1,2n_2+1)=\sum_{k_1=0}^{N/2-1}\sum_{k_2=0}^{N/2-1}\widetilde{y}_{11}(k_1,k_2)W_N^{k_1+k_2}W_{N/2}^{\sum_{i=1}^{2}n_ik_i},\ 0\le n_i\le\frac{N}{2}-1 \qquad (7.7)$$

The sequences $\widetilde{y}_{..}(k_1,k_2)$ in (7.4)-(7.7) are obtained from the original input sequence $\widetilde{x}(k_1,k_2)$ using

$$\widetilde{\mathbf{y}}_{k_1,k_2}=\mathbf{H}_{2^2}\widetilde{\mathbf{x}}_{k_1,k_2},\ 0\le k_i\le\frac{N}{2}-1 \qquad (7.8)$$

where

$$\widetilde{\mathbf{x}}_{k_1,k_2}=\begin{bmatrix}\widetilde{x}(k_1,k_2)\\ \widetilde{x}(k_1,k_2+N/2)\\ \widetilde{x}(k_1+N/2,k_2)\\ \widetilde{x}(k_1+N/2,k_2+N/2)\end{bmatrix} \qquad (7.9)$$

and

$$\widetilde{\mathbf{y}}_{k_1,k_2}=\begin{bmatrix}\widetilde{y}_{00}(k_1,k_2,k_3)\\ \widetilde{y}_{01}(k_1,k_2,k_3)\\ \widetilde{y}_{10}(k_1,k_2,k_3)\\ \widetilde{y}_{11}(k_1,k_2,k_3)\end{bmatrix} \qquad (7.10)$$

The transform operator $\mathbf{H}_{2^2}$ in (7.8) can be factored using (2.8). The decomposition of the DFT given by (7.1) into the DFTs given by (7.4)-(7.7) represents the first stage of the decomposition in the radix-(2×2) DIF FFT algorithm.

In order to develop a radix-$(2\times 2)/(8\times 8)$ FFT algorithm, we further decompose the DFTs given by (7.5)-(7.7) to introduce radix-(8×8) index maps. Let us start by the decomposition of the even-odd indexed terms, which correspond to (7.5). First, we change the variables k_i's by

$$k_i+\frac{N}{8}l_i,\ 0\le k_i\le\frac{N}{8}-1,\ l_i=0,1,2,3 \qquad (7.11)$$

Then, the DFT given by (7.5) becomes

$$\widetilde{X}(2n_1,2n_2+1)=\sum_{k_1=0}^{N/8-1}\sum_{k_2=0}^{N/8-1}\left[\sum_{l_1=0}^{3}\sum_{l_2=0}^{3}\widetilde{y}_{01}\left(k_1+\frac{N}{8}l_1,k_2+\frac{N}{8}l_2\right)e^{-j\frac{\pi}{4}l_2}(-j)^{\sum_{i=1}^{2}n_il_i}\right]W_N^{k_2}W_{N/2}^{\sum_{i=1}^{2}n_ik_i},$$

$$0\le n_i\le\frac{N}{2}-1 \quad (7.12)$$

Now, we change the frequency variables n_i's in (7.12) by

$$4n_i+m_i,\ 0\le n_i\le\frac{N}{8}-1,\ m_i=0,1,2,3 \quad (7.13)$$

Then, (7.12) becomes

$$\widetilde{X}(8n_1+p,8n_2+\beta)=\sum_{k_1=0}^{N/8-1}\sum_{k_2=0}^{N/8-1}\widetilde{y}_{01}^{p\beta}(k_1,k_2)\,W_{N/8}^{\sum_{i=1}^{2}n_ik_i},$$

$$0\le n_i\le\frac{N}{8}-1,\ p=0,2,4,6,\ \beta=1,3,5,7 \quad (7.14)$$

It is clear that (7.14) represents a set of 16 $\left(\frac{N}{8}\times\frac{N}{8}\right)$-point DFTs that are obtained by a further decomposition of the even-odd index terms given by (7.5) using radix-(8×8) index maps. By a similar technique, the odd-even and odd-odd indexed terms given by (7.6) and (7.7) can be expressed, respectively, as

$$\widetilde{X}(8n_1+\alpha,8n_2+p)=\sum_{k_1=0}^{N/8-1}\sum_{k_2=0}^{N/8-1}\widetilde{y}_{10}^{\alpha p}(k_1,k_2)\,W_{N/8}^{\sum_{i=1}^{2}n_ik_i},$$

$$0\le n_i\le\frac{N}{8}-1,\ p=0,2,4,6,\ \alpha=1,3,5,7 \quad (7.15)$$

$$\widetilde{X}(8n_1+\alpha,8n_2+\beta)=\sum_{k_1=0}^{N/8-1}\sum_{k_2=0}^{N/8-1}\widetilde{y}_{11}^{\alpha\beta}(k_1,k_2)\,W_{N/8}^{\sum_{i=1}^{2}n_ik_i},$$
$$0\le n_i\le\frac{N}{8}-1,\ \alpha,\beta=1,3,5,7 \tag{7.16}$$

The input sequences $\widetilde{y}_{\cdot\cdot}^{\cdot\cdot}(k_1,k_2)$ of the DFTs given by (7.14)-(7.16) are, respectively, the components of the vectors given by

$$\widehat{\mathbf{Y}}_{k_1,k_2}^{01}=\mathbf{F}_{k_1,k_2}^{01}\mathbf{W}_{4^2}(\mathbf{I}_4\otimes\mathbf{S}_4)\widetilde{\mathbf{Y}}_{k_1,k_2}^{01},\ 0\le k_i\le\frac{N}{8}-1 \tag{7.17}$$

$$\widehat{\mathbf{Y}}_{k_1,k_2}^{10}=\mathbf{F}_{k_1,k_2}^{10}\mathbf{W}_{4^2}(\mathbf{S}_4\otimes\mathbf{I}_4)\widetilde{\mathbf{Y}}_{k_1,k_2}^{10},\ 0\le k_i\le\frac{N}{8}-1 \tag{7.18}$$

$$\widehat{\mathbf{Y}}_{k_1,k_2}^{11}=\mathbf{F}_{k_1,k_2}^{11}\mathbf{W}_{4^2}(\mathbf{S}_4\otimes\mathbf{S}_4)\widetilde{\mathbf{Y}}_{k_1,k_2}^{11},\ 0\le k_i\le\frac{N}{8}-1 \tag{7.19}$$

where the components of the input vectors $\widetilde{\mathbf{Y}}_{k_1,k_2}^{\cdot\cdot}$, output vectors $\widehat{\mathbf{Y}}_{k_1,k_2}^{\cdot\cdot}$ and those of their corresponding diagonal twiddle factor matrices $\mathbf{F}_{k_1,k_2}^{\cdot\cdot}$ are given by

$$\begin{cases}\widetilde{Y}_{k_1,k_2}^{01}(a)=\widetilde{y}_{01}\left(k_1+p\frac{N}{16},k_2+(\beta-1)\frac{N}{16}\right)\\ \widehat{Y}_{k_1,k_2}^{01}(a)=\widetilde{y}_{01}^{p\beta}(k_1,k_2)\\ F_{k_1,k_2}^{01}(a,a)=W_N^{pk_1+\beta k_2}\\ \text{where } a=2p+\frac{\beta-1}{2},\ p=0,2,4,6,\ \beta=1,3,5,7\end{cases} \tag{7.20}$$

$$\begin{cases}\widetilde{Y}_{k_1,k_2}^{10}(b)=\widetilde{y}_{10}\left(k_1+(\alpha-1)\frac{N}{16},k_2+p\frac{N}{16}\right)\\ \widehat{Y}_{k_1,k_2}^{10}(b)=\widetilde{y}_{010}^{\alpha p}(k_1,k_2)\\ F_{k_1,k_2}^{10}(b,b)=W_N^{\alpha k_1+pk_2}\\ \text{where } b=2\alpha-2+\frac{p}{2},\ p=0,2,4,6,\ \alpha=1,3,5,7\end{cases} \tag{7.21}$$

and

$$\begin{cases} \widetilde{Y}^{11}_{k_1,k_2}(c) = \widetilde{y}_{11}\left(k_1+(\alpha-1)\frac{N}{16}, k_2+(\beta-1)\frac{N}{16}\right) \\ \widehat{Y}^{11}_{k_1,k_2}(c) = \widetilde{y}^{\alpha\beta}_{11}(k_1,k_2) \\ F^{11}_{k_1,k_2}(c,c) = W_N^{\alpha k_1+\beta k_2} \\ \text{where } c = 2\alpha-2+\frac{\beta-1}{2},\ \alpha,\beta = 1,3,5,7 \end{cases} \tag{7.22}$$

In (7.17)-(7.19), the matrix $\mathbf{S}_4$ is given by

$$\mathbf{S}_4 = (\mathbf{I}_2 \otimes \mathbf{C}_2)(\mathbf{J}_2 \otimes \mathbf{I}_2),\ \mathbf{C}_2 = \begin{bmatrix} 1 & 0 \\ 0 & e^{-j\frac{\pi}{4}} \end{bmatrix},\ \mathbf{J}_2 = \begin{bmatrix} 1 & 0 \\ 0 & -j \end{bmatrix} \tag{7.23}$$

and the matrix $\mathbf{W}_{4^2}$, the operator of the (4×4)-point 2-D DFT, can be factored using the Kronecker product as [32]

$$\begin{aligned} \mathbf{W}_{4^2} &= \mathbf{W}_4 \otimes \mathbf{W}_4 \\ &= \prod_{l=1}^{2} (\mathbf{I}_{4^{l-1}} \otimes \mathbf{Z}_4 \otimes \mathbf{I}_{4^{2-l}})(\mathbf{I}_{4^{l-1}} \otimes \mathbf{D}_4 \otimes \mathbf{I}_{4^{2-l}})(\mathbf{I}_{4^{l-1}} \otimes \mathbf{R}_4 \otimes \mathbf{I}_{4^{2-l}}) \end{aligned} \tag{7.24}$$

where the matrices $\mathbf{Z}_4$, $\mathbf{D}_4$ and $\mathbf{R}_4$ are given by (4.7), (4.8) and (4.9), respectively.

For a given combination of (k_1,k_2), each of the three general sub-butterflies given by (7.17)-(7.19) requires 16 input points to be calculated using (7.8). Hence, the operation given by (7.8) is repeated 16 times, since $\widetilde{y}_{00}$ is no longer required by the sub-butterflies and the other three output points are used directly by the corresponding sub-butterflies. In order to compute, for a given combination of (k_1,k_2), all the points required in the computation of these three sub-butterflies, we rearrange (7.8) as

$$\begin{aligned} \widetilde{\mathbf{y}}_{k_1+l_1\frac{N}{8},k_2+l_2\frac{N}{8}} &= \mathbf{H}_{2^2}\widetilde{\mathbf{x}}_{k_1+l_1\frac{N}{8},k_2+l_2\frac{N}{8}}, \\ 0 \le k_i \le \frac{N}{8}-1,\ l_i &= 0,1,2,3 \end{aligned} \tag{7.25}$$

Using (7.9) and (7.10), it is clear that for a given combination of (k_1, k_2), (7.25) computes 64 points from the initial input sequence $\widetilde{x}(k_1, k_2)$, where 16 points are used to form the input sequence $\widetilde{y}_{00}(k_1, k_2)$ of the $\left(\frac{N}{2} \times \frac{N}{2}\right)$-point 2-D DFT given by (7.4) and all the other points are used by the general sub-butterflies given by (7.17)-(7.19). The output of these sub-butterflies are used to form the input sequences of the corresponding $\left(\frac{N}{8} \times \frac{N}{8}\right)$-point 2-D DFTs given by (7.14)-(7.16). These operations are repeated for all the combinations of (k_1, k_2), $k_i = 0, 1, ..., (N/8) - 1$, $i = 1, 2$. This completes the first stage of the proposed decomposition.

Thus, the proposed radix-$(2 \times 2)/(8 \times 8)$ FFT algorithm corresponds to decomposing the $(N \times N)$-point 2-D DFT given by (7.1) into one $\left(\frac{N}{2} \times \frac{N}{2}\right)$-point 2-D DFT given by (7.4) and 48 $\left(\frac{N}{8} \times \frac{N}{8}\right)$-point 2-D DFTs given by (7.14)-(7.16), in the first stage. This is achieved by repeating $\frac{N^3}{64}$ times the general butterfly based on (7.25) and (7.17)-(7.19). This decomposition scheme is repeated successively for each of the new resulting DFTs, until the problem is reduced to some DFTs without twiddle factors, namely (4×4)- or (2×2)-point 2-D DFTs.

Let us now count the operations required by the general butterfly of the proposed 2-D FFT algorithm. This is achieved by counting all the necessary operations for a given combination of (k_1, k_2). According to (2.8), the matrix $\mathbf{H}_{22}$ introduces 8×16 complex additions in the computation of the operation given by (7.25). According to (7.24), the matrix $\mathbf{W}_{4^2}$ introduces $16 \times 2 \times 2$ complex additions in the computation of each of the three general sub-butterflies given by (7.17)-(7.19). The computation of each of these sub-butterflies also requires 16 complex multiplications that are introduced by the corresponding twiddle factor matrix $\mathbf{F}_{k_1,k_2}$, and 16 real multiplications and 16 real additions that are introduced by the corresponding diagonal matrix generated using $\mathbf{S}_4$. Consequently, the general butterfly of the proposed 2-D FFT algorithm requires 240 real multiplications and 784 real additions, if the 4mult-2add scheme is considered, and 192 real multiplications and 832 real additions,

if the 3mult-3add scheme is considered.

In order to further reduce the number of operations, we consider some special combinations of (k_1, k_2) that involve some trivial operations. It is clear from (7.20)-(7.22) that each of the twiddle factor matrices $\mathbf{F}_{k,0}^{01}$ and $\mathbf{F}_{0,k}^{10}$ requires only 12 complex multiplications, for $k = 1, 2, ..., (N/8) - 1$ and $k \neq s, s/2$, where $s = \frac{N}{16}$. It is also clear that the twiddle factor matrices $\mathbf{F}_{s,2s}^{01}$, $\mathbf{F}_{3s,2s}^{01}$, $\mathbf{F}_{s,0}^{01}$, $\mathbf{F}_{2s,s}^{10}$, $\mathbf{F}_{2s,3s}^{10}$, $\mathbf{F}_{0,s}^{10}$, $\mathbf{F}_{s,s}^{11}$, $\mathbf{F}_{3s,s}^{11}$, $\mathbf{F}_{s,3s}^{11}$ and $\mathbf{F}_{3s,3s}^{11}$ involve some trivial operations. Since the sub-butterflies given by (7.17)-(7.19) require pre-multiplications (due to $\mathbf{S}_4$) and post-multiplications (due to $\mathbf{F}_{k_1,k_2}^{\cdot\cdot}$), a technique for combining twiddle factors is introduced to reduce the number of operations for some special combinations of (k_1, k_2). This technique is based on the following theorem.

***Theorem* 1**: Let two matrices $\mathbf{Z}_{4^M}$ and $\mathbf{R}_{4^M}$ of order 4^M be defined as

$$\mathbf{Z}_{4^M} = \mathbf{Z}_4 \otimes \mathbf{Z}_4 \otimes ... \otimes \mathbf{Z}_4 = \prod_{i=1}^{M} (\mathbf{I}_{4^{i-1}} \otimes \mathbf{Z}_4 \otimes \mathbf{I}_{4^{M-i}}) \tag{7.26}$$

$$\mathbf{R}_{4^M} = \mathbf{R}_4 \otimes \mathbf{R}_4 \otimes ... \otimes \mathbf{R}_4 = \prod_{i=1}^{M} (\mathbf{I}_{4^{i-1}} \otimes \mathbf{R}_4 \otimes \mathbf{I}_{4^{M-i}}) \tag{7.27}$$

Let $\mathbf{A}_{4^M}$ and $\widetilde{\mathbf{A}}_{4^M}$ be two diagonal matrices generated by an arbitrary diagonal matrix $\mathbf{A}_2$ of order two, as

$$\mathbf{A}_{4^M} = \mathbf{I}_{4^{M-1}} \otimes (\mathbf{I}_2 \otimes \mathbf{A}_2) \tag{7.28}$$

$$\widetilde{\mathbf{A}}_{4^M} = \mathbf{I}_{4^{M-1}} \otimes (\mathbf{A}_2 \otimes \mathbf{I}_2) \tag{7.29}$$

Then,

(i) $\mathbf{A}_{4^M}\mathbf{Z}_{4^M} = \mathbf{Z}_{4^M}\mathbf{A}_{4^M}$

(ii) $\mathbf{R}_{4^M}\mathbf{A}_{4^M} = \widetilde{\mathbf{A}}_{4^M}\mathbf{R}_{4^M}$.

Proof:

(i) For $i = 1, 2, ..., M-1$, the product of the matrix $\mathbf{A}_{4^M}$ by the ith term $(\mathbf{I}_{4^{i-1}} \otimes \mathbf{Z}_4 \otimes \mathbf{I}_{4^{M-i}})$

of the right side of (7.26) is commutative. That is,

$$\begin{aligned}
\mathbf{A}_{4M}\left(\mathbf{I}_{4i-1}\otimes\mathbf{Z}_4\otimes\mathbf{I}_{4M-i}\right) &= \left(\mathbf{I}_{4M-1}\otimes\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)\right)\left(\mathbf{I}_{4i-1}\otimes\mathbf{Z}_4\otimes\mathbf{I}_{4M-i}\right)\\
&= \left(\mathbf{I}_{4i-1}\otimes\left(\mathbf{I}_{4M-i}\otimes\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)\right)\right)\left(\mathbf{I}_{4i-1}\otimes\left(\mathbf{Z}_4\otimes\mathbf{I}_{4M-i}\right)\right)\\
&= \mathbf{I}_{4i-1}\otimes\left[\left(\mathbf{I}_{4M-i}\otimes\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)\right)\left(\mathbf{Z}_4\otimes\mathbf{I}_{4M-i}\right)\right]\\
&= \mathbf{I}_{4i-1}\otimes\left[\left(\mathbf{I}_4\otimes\left(\mathbf{I}_{4M-i-1}\otimes\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)\right)\right)\left(\mathbf{Z}_4\otimes\mathbf{I}_{4M-i}\right)\right]\\
&= \mathbf{I}_{4i-1}\otimes\left[\left(\mathbf{I}_4\mathbf{Z}_4\right)\otimes\left(\left(\mathbf{I}_{4M-i-1}\otimes\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)\right)\mathbf{I}_{4M-i}\right)\right]\\
&= \mathbf{I}_{4i-1}\otimes\left[\left(\mathbf{Z}_4\mathbf{I}_4\right)\otimes\left(\mathbf{I}_{4M-i}\left(\mathbf{I}_{4M-i-1}\otimes\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)\right)\right)\right]\\
&= \mathbf{I}_{4i-1}\otimes\left[\left(\mathbf{Z}_4\otimes\mathbf{I}_{4M-i}\right)\left(\mathbf{I}_4\otimes\mathbf{I}_{4M-i-1}\otimes\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)\right)\right]\\
&= \left(\mathbf{I}_{4i-1}\otimes\mathbf{Z}_4\otimes\mathbf{I}_{4M-i}\right)\left(\mathbf{I}_{4M-1}\otimes\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)\right)\\
&= \left(\mathbf{I}_{4i-1}\otimes\mathbf{Z}_4\otimes\mathbf{I}_{4M-i}\right)\mathbf{A}_{4M} \qquad (7.30)
\end{aligned}$$

For $i = M$,

$$\begin{aligned}
\mathbf{A}_{4M}\left(\mathbf{I}_{4M-1}\otimes\mathbf{Z}_4\right) &= \left(\mathbf{I}_{4M-1}\otimes\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)\right)\left(\mathbf{I}_{4M-1}\otimes\mathbf{Z}_4\right)\\
&= \mathbf{I}_{4M-1}\otimes\left(\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)\mathbf{Z}_4\right)\\
&= \left(\mathbf{I}_{4M-1}\otimes\mathbf{Z}_4\right)\left(\mathbf{I}_{4M-1}\otimes\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)\right)\\
&= \left(\mathbf{I}_{4M-1}\otimes\mathbf{Z}_4\right)\mathbf{A}_{4M} \qquad (7.31)
\end{aligned}$$

where in the last step we have used the fact that $\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)\mathbf{Z}_4 = \mathbf{Z}_4\left(\mathbf{I}_2\otimes\mathbf{A}_2\right)$. Combining (7.30) and (7.31), we see that $\mathbf{A}_{4M}\mathbf{Z}_{4M} = \mathbf{Z}_{4M}\mathbf{A}_{4M}$.

(ii) Similarly, for $i = 1, 2, ..., M-1$, it can be shown that the product of the matrix $\widetilde{\mathbf{A}}_{4M}$ by the ith term $\left(\mathbf{I}_{4i-1}\otimes\mathbf{R}_4\otimes\mathbf{I}_{4M-i}\right)$ of the right side of (7.27) is also commutative. That is,

$$\widetilde{\mathbf{A}}_{4M}\left(\mathbf{I}_{4i-1}\otimes\mathbf{R}_4\otimes\mathbf{I}_{4M-i}\right) = \left(\mathbf{I}_{4i-1}\otimes\mathbf{R}_4\otimes\mathbf{I}_{4M-i}\right)\widetilde{\mathbf{A}}_{4M} \qquad (7.32)$$

Also, for $i = M$, since $\mathbf{R}_4 (\mathbf{I}_2 \otimes \mathbf{A}_2) = (\mathbf{A}_2 \otimes \mathbf{I}_2) \mathbf{R}_4$, we have

$$\begin{aligned} \widetilde{\mathbf{A}}_{4M} (\mathbf{I}_{4M-1} \otimes \mathbf{R}_4) &= (\mathbf{I}_{4M-1} \otimes (\mathbf{A}_2 \otimes \mathbf{I}_2)) (\mathbf{I}_{4M-1} \otimes \mathbf{R}_4) \\ &= \mathbf{I}_{4M-1} \otimes ((\mathbf{A}_2 \otimes \mathbf{I}_2) \mathbf{R}_4) \\ &= (\mathbf{I}_{4M-1} \otimes \mathbf{R}_4) (\mathbf{I}_{4M-1} \otimes (\mathbf{I}_2 \otimes \mathbf{A}_2)) \\ &= (\mathbf{I}_{4M-1} \otimes \mathbf{R}_4) \mathbf{A}_{4M} \end{aligned} \tag{7.33}$$

Combining (7.32) and (7.33), we see that $\mathbf{R}_{4M} \mathbf{A}_{4M} = \widetilde{\mathbf{A}}_{4M} \mathbf{R}_{4M}$.

Now, for $M = 2$, by an appropriate application of this theorem in the expressions given by (7.17)-(7.19), we can show the following:

a) The sub-butterfly given by (7.17) requires only 16 real multiplications and 144 real additions for each of the combinations $(0, 0)$ and $(2s, 0)$ of (k_1, k_2), whereas it needs 16 complex multiplications and 64 complex additions for each of the combinations $(0, 2s)$ and $(2s, 2s)$.

b) The sub-butterfly given by (7.18) requires only 16 real multiplications and 144 real additions for each of the combinations $(0, 0)$ and $(0, 2s)$, whereas it needs 16 complex multiplications and 64 complex additions for each of the combinations $(2s, 0)$ and $(2s, 2s)$.

c) The sub-butterfly given by (7.19) requires only 16 real multiplications and 144 real additions for each of the combinations $(0, 0)$ and $(2s, 2s)$, whereas it needs 16 complex multiplications and 64 complex additions for each of the combinations $(0, 2s)$ and $(2s, 0)$.

7.2.1 Computational Complexity

In this section we consider the efficiency of the proposed radix-$(2\times 2)/(8\times 8)$ FFT algorithm when it is implemented using the general butterfly based on (7.25) and (7.17)-(7.19), and the special sub-butterflies corresponding to the special combinations of (k_1,k_2) defined in Section 7.1.1. This is done by comparing its computational complexity with that of the existing radix-$(2\times 2)/(4\times 4)$ FFT algorithm [29]. The following computational complexity of the proposed 2-D FFT algorithm is found by a close examination of the scheme described in Section 7.1.1.

7.2.1.1 Arithmetic Complexity

It can be seen that the expressions for the numbers of real multiplications and real additions required by the proposed 2-D FFT algorithm are, respectively,

$$\begin{aligned} \mathrm{M}^{33}_{\mathrm{r2/8}}(2,N) &= 3N^2-3N-472+\mathrm{M}^{33}_{\mathrm{r2/8}}\left(2,\frac{N}{2}\right)+48\mathrm{M}^{33}_{\mathrm{r2/8}}\left(2,\frac{N}{8}\right),\ N\geq 32,\\ &\mathrm{M}^{33}_{\mathrm{r2/8}}(2,16)=432,\ \mathrm{M}^{33}_{\mathrm{r2/8}}(2,8)=48,\ \mathrm{M}^{33}_{\mathrm{r2/8}}(2,4)=0 \end{aligned} \quad (7.34)$$

and

$$\begin{aligned} \mathrm{A}^{33}_{\mathrm{r2/8}}(2,N) &= 13N^2-3N-472+\mathrm{A}^{33}_{\mathrm{r2/8}}\left(2,\frac{N}{2}\right)+48\mathrm{A}^{33}_{\mathrm{r2/8}}\left(2,\frac{N}{8}\right),\ N\geq 32,\\ &\mathrm{A}^{33}_{\mathrm{r2/8}}(2,16)=4528,\ \mathrm{A}^{33}_{\mathrm{r2/8}}(2,8)=816,\ \mathrm{A}^{33}_{\mathrm{r2/8}}(2,4)=128 \end{aligned} \quad (7.35)$$

if the 3mult-3add scheme is considered, and

$$\begin{aligned} \mathrm{M}^{42}_{\mathrm{r2/8}}(2,N) &= \frac{15}{4}N^2-4N-624+\mathrm{M}^{42}_{\mathrm{r2/8}}\left(2,\frac{N}{2}\right)+48\mathrm{M}^{42}_{\mathrm{r2/8}}\left(2,\frac{N}{8}\right),\ N\geq 32,\\ &\mathrm{M}^{42}_{\mathrm{r2/8}}(2,16)=528,\ \mathrm{M}^{42}_{\mathrm{r2/8}}(2,8)=48,\ \mathrm{M}^{42}_{\mathrm{r2/8}}(2,4)=0 \end{aligned} \quad (7.36)$$

Table 7.1: Number of arithmetic operations required for the computation of the 2-D DFT using the 3mult-3add scheme

Transform size	Radix- $(2\times 2)/(4\times 4)$ FFT in [29]		Proposed radix- $(2\times 2)/(8\times 8)$ FFT		Savings (%)	
$N\times N$	Mults.	Adds.	Mults.	Adds.	Mults.	Adds.
$2^2\times 2^2$	0	128	0	128	0	0
$2^3\times 2^3$	48	816	48	816	0	0
$2^4\times 2^4$	560	4654	432	4528	22.85	2.70
$2^5\times 2^5$	3312	23792	2936	23416	11.35	1.58
$2^6\times 2^6$	18992	117296	16864	115168	11.20	1.81
$2^7\times 2^7$	95088	553840	85896	544648	9.66	1.66
$2^8\times 2^8$	469424	2566576	422192	2519344	10.06	1.84
$2^9\times 2^9$	2198256	11635440	2016088	11453272	8.28	1.56
$2^{10}\times 2^{10}$	10186544	52129584	9281280	51224320	8.88	1.73

and

$$\mathrm{A}^{42}_{\mathrm{r2/8}}(2,N)=\frac{49}{4}N^2-2N-320+\mathrm{A}^{42}_{\mathrm{r2/8}}\left(2,\frac{N}{2}\right)+48\mathrm{A}^{42}_{\mathrm{r2/8}}\left(2,\frac{N}{8}\right),\ N\geq 32,$$
$$\mathrm{A}^{42}_{\mathrm{r2/8}}(2,16)=4432,\ \mathrm{M}^{42}_{\mathrm{r2/8}}(2,8)=816,\ \mathrm{M}^{42}_{\mathrm{r2/8}}(2,4)=128 \quad (7.37)$$

if the 4mult-2add scheme is considered.

The arithmetic complexities of the proposed radix-$(2\times 2)/(8\times 8)$ and the existing radix-$(2\times 2)/(4\times 4)$ [29] FFT algorithms for complex data for various values of N are given in Tables 7.1 and 7.2. It is clear from these tables that the proposed 2-D FFT algorithm reduces significantly the number of arithmetic operations, especially the number of real multiplications.

Table 7.2: Number of arithmetic operations required for the computation of the 2-D DFT using the 4mult-2add scheme

Transform size	Radix- $(2\times2)/(4\times4)$ FFT in [29]		Proposed radix- $(2\times2)/(8\times8)$ FFT		Savings (%)	
$N\times N$	Mults.	Adds.	Mults.	Adds.	Mults.	Adds.
$2^2\times2^2$	0	128	0	128	0	0
$2^3\times2^3$	48	816	48	816	0	0
$2^4\times2^4$	720	4496	528	4432	26.66	1.42
$2^5\times2^5$	4176	22928	3616	22736	13.41	0.83
$2^6\times2^6$	24720	111568	20400	111632	17.47	-0.05
$2^7\times2^7$	123216	525712	106048	524496	13.93	0.23
$2^8\times2^8$	614928	2421072	523728	2417808	14.83	0.13
$2^9\times2^9$	2876880	10956816	2483296	10986064	13.68	-0.26
$2^{10}\times2^{10}$	13395600	48920528	11501040	49004560	14.14	-0.17

7.2.1.2 Data Transfers

We assume that a sufficient on-chip memory is available to perform an entire butterfly without using any intermediate transfer operations between the processor and the off-chip memory. For a given combination of (k_1,k_2), the implementation of the butterfly based on (7.25) and (7.17)-(7.19) of the proposed 2-D FFT algorithm consists of reading four points from the off-ship memory of the processor and computing the operation given by (7.25) using these four points as operands, for $l_1 = l_2 = 0$. The result $\widetilde{y}_{00}(k_1,k_2)$ is returned to the off-ship memory, whereas the other three results $\widetilde{y}_{01}(k_1,k_2)$, $\widetilde{y}_{10}(k_1,k_2)$ and $\widetilde{y}_{11}(k_1,k_2)$ are kept in the on-ship memory of the processor. This process is repeated 16 times to compute the operation given by (7.25) for all combinations of (l_1,l_2). The three 16-point results kept in the on-ship memory are used to process the three sub-butterflies given by (7.17)-(7.19). The above operations are repeated by taking all the combinations of (k_1,k_2), $k_1, k_2 = 0, 1, ..., (N/8)-1$. Now, all the results of $\widetilde{y}_{00}$ returned to the off-ship memory are grouped to form the first sub-sequence of size $\left(\frac{N}{2}\times\frac{N}{2}\right)$, and the 48 sub-sequences of size $\left(\frac{N}{8}\times\frac{N}{8}\right)$ are formed by grouping separately the results of each line of the output vectors of (7.17)-(7.19). This scheme of implementation reduces significantly the number of data transfers

and index generations. To compare the number of data transfers and address generations of the proposed radix-$(2\times 2)/(8\times 8)$ and the existing radix-$(2\times 2)/(4\times 4)$ FFT algorithms, we assume that for a given size, the corresponding repeated butterfly operations of both the algorithms are applied in the same way. Then, it can be shown that the expressions for the number of data transfers (real and imaginary parts) of the radix-$(2\times 2)/(8\times 8)$ and radix-$(2\times 2)/(4\times 4)$ FFT algorithms are respectively

$$\begin{aligned} \mathrm{D}_{\mathrm{r2/8}}(2,N) &= 2N^2 + \mathrm{D}_{\mathrm{r2/8}}\left(2,\frac{N}{2}\right) + 48\mathrm{D}_{\mathrm{r2/8}}\left(2,\frac{N}{8}\right),\ N \geq 8, \\ \mathrm{D}_{\mathrm{r2/8}}(2,4) &= 32,\ \mathrm{D}_{\mathrm{r2/8}}(2,2) = 8,\ \mathrm{D}_{\mathrm{r2/8}}(2,1) = 0 \end{aligned} \tag{7.38}$$

and

$$\begin{aligned} \mathrm{D}_{\mathrm{r2/4}}(2,N) &= 2N^2 + \mathrm{D}_{\mathrm{r2/4}}\left(2,\frac{N}{2}\right) + 12\mathrm{D}_{\mathrm{r2/4}}\left(2,\frac{N}{4}\right),\ N \geq 4, \\ \mathrm{D}_{\mathrm{r2/4}}(2,2) &= 8,\ \mathrm{D}_{\mathrm{r2/4}}(2,1) = 0 \end{aligned} \tag{7.39}$$

The number of data transfers for the two algorithms are compared in Table 7.3. The proposed 2-D FFT algorithm requires about 25% less data transfer operations than that in the existing one. Hence, similar savings are also achieved in the address generation. The use of vector radix-(8×8) Cooley-Tukey FFT can bring about even more savings; however, it will have a higher arithmetic complexity compared to that of the proposed 2-D FFT algorithm and imposes more restrictions on the choice of the transform size.

7.2.1.3 Twiddle Factors

In counting the number of twiddle factor (sine and cosine) evaluations or accesses to the lookup table required by the two algorithms, it is assumed that the constant factor $1/\sqrt{2}$ for the algorithm in [29], and the constant factors $1/\sqrt{2}$, $\cos(\pi/8)$ and $\sin(\pi/8)$ for the pro-

Table 7.3: Comparison of the number of data transfers, and twiddle factor evaluations or accesses to the lookup table

	Radix-$(2\times2)/(4\times4)$ FFT in [29]		Proposed radix-$(2\times2)/(8\times8)$ FFT		Savings (%)	
Transform size $N\times N$	$D_{r2/4}(2,N)$ (A)	$T^{42}_{r2/4}(2,N)$ (B)	$D_{r2/8}(2,N)$ (C)	$T^{42}_{r2/8}(2,N)$ (D)	in C over A	in D over B
2×2	8	0	8	0	0	0
$2^2\times2^2$	40	0	32	0	20	0
$2^3\times2^3$	264	0	160	0	39.39	0
$2^4\times2^4$	1256	256	1056	0	15.92	-
$2^5\times2^5$	6472	1600	4640	0	28.30	-
$2^6\times2^6$	29736	10496	20512	5180	31.01	50.64
$2^7\times2^7$	140168	53696	103968	28376	25.82	47.15
$2^8\times2^8$	628072	276864	457760	124468	27.11	55.04
$2^9\times2^9$	2834376	1312320	1966624	762448	30.61	41.90
$2^{10}\times2^{10}$	12468392	6203392	9054240	3690156	27.38	40.51

posed 2-D FFT algorithm, are initialized and kept in the on-ship memory of the processor during the processing time of the algorithms. Therefore, they are respectively

$$T^{42}_{r2/8}(2,N)=\frac{3}{2}N^2-\frac{13}{2}N-548+T^{42}_{r2/8}\left(2,\frac{N}{2}\right)+48T^{42}_{r2/8}\left(2,\frac{N}{8}\right),\ N\geq64,$$
$$T^{42}_{r2/8}(2,N)=0,\ N\leq32, \quad (7.40)$$

$$T^{42}_{r2/4}(2,N)=\frac{3}{2}N^2-4N-64+T^{42}_{r2/4}\left(2,\frac{N}{2}\right)+12T^{42}_{r2/4}\left(2,\frac{N}{4}\right),\ N\geq16,$$
$$T^{42}_{r2/4}(2,N)=0,\ N\leq8 \quad (7.41)$$

It is seen from Table 7.3 that a saving of over 40% in the evaluation of twiddle factors or in the access to the lookup table can easily be achieved by the proposed 2-D FFT algorithm. Note that, when the lookup table is used, similar savings are obtained by the proposed 2-D FFT algorithm in the address generation for reading the twiddle factors.

7.3 Proposed Radix-$(2 \times 2 \times 2)/(8 \times 8 \times 8)$ DIF FFT Algorithm

In the radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ DIF FFT algorithm developed in Section 5.2, radix-$(4 \times 4 \times 4)$ index maps are introduced by further decomposing the DFTs given by (5.5)-(5.11) using radix-$(2 \times 2 \times 2)$ index maps. In order to develop a radix-$(2 \times 2 \times 2)/(8 \times 8 \times 8)$ DIF FFT algorithm, we introduce radix-$(8 \times 8 \times 8)$ index maps by further decomposing the DFTs given by (5.5)-(5.11) using radix-$(4 \times 4 \times 4)$ index maps. Let us start by decomposing the even-even-odd indexed terms, which correspond to (5.5). First, we change the variables k_i's by

$$k_i + \frac{N}{8} l_i, \; 0 \le k_i \le \frac{N}{8} - 1, \; l_i = 0, 1, 2, 3 \tag{7.42}$$

Then, the DFT given by (5.5) becomes

$$\widetilde{X}(2n_1, 2n_2, 2n_3 + 1) = \sum_{k_1=0}^{N/8-1} \sum_{k_2=0}^{N/8-1} \sum_{k_3=0}^{N/8-1} \left[\sum_{l_1=0}^{3} \sum_{l_2=0}^{3} \sum_{l_3=0}^{3} \widetilde{y}_{001}\left(k_1 + \frac{N}{8} l_1, k_2 + \frac{N}{8} l_2, k_3 + \frac{N}{8} l_3\right) e^{-j\frac{\pi}{4} l_3} (-j)^{\sum_{i=1}^{3} n_i l_i} \right] W_N^{k_3} W_{N/2}^{\sum_{i=1}^{3} n_i k_i},$$

$$0 \le n_i \le \frac{N}{2} - 1 \tag{7.43}$$

Now, we change the frequency variables n_i's in (7.43) by

$$4n_i + m_i, \; 0 \le n_i \le \frac{N}{8} - 1, \; m_i = 0, 1, 2, 3 \tag{7.44}$$

Then, (7.43) becomes

$$\widetilde{X}(8n_1+p,8n_2+q,8n_3+\gamma)=\sum_{k_1=0}^{N/8-1}\sum_{k_2=0}^{N/8-1}\sum_{k_3=0}^{N/8-1}\widetilde{h}_{001}^{pq\gamma}(k_1,k_2,k_3)\,W_{N/8}^{\sum_{i=1}^{3}n_ik_i},$$

$$0\le n_i\le\frac{N}{8}-1,\ p,q=0,2,4,6,\ \gamma=1,3,5,7 \qquad (7.45)$$

It is clear that (7.45) represents a set of 64 $\left(\frac{N}{8}\times\frac{N}{8}\times\frac{N}{8}\right)$-point DFTs that are obtained by further decomposition of the even-even-odd index terms given by (5.5) using the radix-$(4\times4\times4)$ index maps given by (7.42) and (7.44). By a similar technique, the other odd-indexed terms given by (5.6)-(5.11) can be expressed, respectively, as

$$\widetilde{X}(8n_1+p,8n_2+\beta,8n_3+q)=\sum_{k_1=0}^{N/8-1}\sum_{k_2=0}^{N/8-1}\sum_{k_3=0}^{N/8-1}\widetilde{h}_{010}^{p\beta q}(k_1,k_2,k_3)\,W_{N/8}^{\sum_{i=1}^{3}n_ik_i},$$

$$0\le n_i\le\frac{N}{8}-1,\ p,q=0,2,4,6,\ \beta=1,3,5,7 \qquad (7.46)$$

$$\widetilde{X}(8n_1+p,8n_2+\beta,8n_3+\gamma)=\sum_{k_1=0}^{N/8-1}\sum_{k_2=0}^{N/8-1}\sum_{k_3=0}^{N/8-1}\widetilde{h}_{011}^{p\beta\gamma}(k_1,k_2,k_3)\,W_{N/8}^{\sum_{i=1}^{3}n_ik_i},$$

$$0\le n_i\le\frac{N}{8}-1,\ p=0,2,4,6,\ \beta,\gamma=1,3,5,7 \qquad (7.47)$$

$$\widetilde{X}(8n_1+\alpha,8n_2+p,8n_3+q)=\sum_{k_1=0}^{N/8-1}\sum_{k_2=0}^{N/8-1}\sum_{k_3=0}^{N/8-1}\widetilde{h}_{100}^{\alpha pq}(k_1,k_2,k_3)\,W_{N/8}^{\sum_{i=1}^{3}n_ik_i},$$

$$0\le n_i\le\frac{N}{8}-1,\ p,q=0,2,4,6,\ \alpha=1,3,5,7 \qquad (7.48)$$

$$\widetilde{X}(8n_1+\alpha,8n_2+p,8n_3+\gamma)=\sum_{k_1=0}^{N/8-1}\sum_{k_2=0}^{N/8-1}\sum_{k_3=0}^{N/8-1}\widetilde{h}_{101}^{\alpha p\gamma}(k_1,k_2,k_3)\,W_{N/8}^{\sum_{i=1}^{3}n_ik_i},$$

$$0\le n_i\le\frac{N}{8}-1,\ p=0,2,4,6,\ \alpha,\gamma=1,3,5,7 \qquad (7.49)$$

$$\widetilde{X}(8n_1+\alpha,8n_2+\beta,8n_3+p)=\sum_{k_1=0}^{N/8-1}\sum_{k_2=0}^{N/8-1}\sum_{k_3=0}^{N/8-1}\widetilde{h}_{110}^{\alpha\beta p}(k_1,k_2,k_3)\,W_{N/8}^{\sum_{i=1}^{3}n_ik_i},$$
$$0\le n_i\le\frac{N}{8}-1,\ p=0,2,4,6,\ \alpha,\beta=1,3,5,7 \qquad (7.50)$$

$$\widetilde{X}(8n_1+\alpha,8n_2+\beta,8n_3+\gamma)=\sum_{k_1=0}^{N/8-1}\sum_{k_2=0}^{N/8-1}\sum_{k_3=0}^{N/8-1}\widetilde{h}_{111}^{\alpha\beta\gamma}(k_1,k_2,k_3)\,W_{N/8}^{\sum_{i=1}^{3}n_ik_i},$$
$$0\le n_i\le\frac{N}{8}-1,\ \alpha,\beta,\gamma=1,3,5,7 \qquad (7.51)$$

The input sequences $\widetilde{h}_{\cdots}^{\cdots}(k_1,k_2,k_3)$ of the DFTs given by (7.45)-(7.51) are, respectively, the components of the vectors given by

$$\widehat{\mathbf{H}}_{k_1,k_2,k_3}^{001}=\mathbf{B}_{k_1,k_2,k_3}^{001}\mathbf{W}_{4^3}(\mathbf{I}_4\otimes\mathbf{I}_4\otimes\mathbf{S}_4)\widetilde{\mathbf{H}}_{k_1,k_2,k_3}^{001},\ 0\le k_i\le\frac{N}{8}-1 \qquad (7.52)$$

$$\widehat{\mathbf{H}}_{k_1,k_2,k_3}^{010}=\mathbf{B}_{k_1,k_2,k_3}^{010}\mathbf{W}_{4^3}(\mathbf{I}_4\otimes\mathbf{S}_4\otimes\mathbf{I}_4)\widetilde{\mathbf{H}}_{k_1,k_2,k_3}^{010},\ 0\le k_i\le\frac{N}{8}-1 \qquad (7.53)$$

$$\widehat{\mathbf{H}}_{k_1,k_2,k_3}^{011}=\mathbf{B}_{k_1,k_2,k_3}^{011}\mathbf{W}_{4^3}(\mathbf{I}_4\otimes\mathbf{S}_4\otimes\mathbf{S}_4)\widetilde{\mathbf{H}}_{k_1,k_2,k_3}^{011},\ 0\le k_i\le\frac{N}{8}-1 \qquad (7.54)$$

$$\widehat{\mathbf{H}}_{k_1,k_2,k_3}^{100}=\mathbf{B}_{k_1,k_2,k_3}^{100}\mathbf{W}_{4^3}(\mathbf{S}_4\otimes\mathbf{I}_4\otimes\mathbf{I}_4)\widetilde{\mathbf{H}}_{k_1,k_2,k_3}^{100},\ 0\le k_i\le\frac{N}{8}-1 \qquad (7.55)$$

$$\widehat{\mathbf{H}}_{k_1,k_2,k_3}^{101}=\mathbf{B}_{k_1,k_2,k_3}^{101}\mathbf{W}_{4^3}(\mathbf{S}_4\otimes\mathbf{I}_4\otimes\mathbf{S}_4)\widetilde{\mathbf{H}}_{k_1,k_2,k_3}^{101},\ 0\le k_i\le\frac{N}{8}-1 \qquad (7.56)$$

$$\widehat{\mathbf{H}}_{k_1,k_2,k_3}^{110}=\mathbf{B}_{k_1,k_2,k_3}^{110}\mathbf{W}_{4^3}(\mathbf{S}_4\otimes\mathbf{S}_4\otimes\mathbf{I}_4)\widetilde{\mathbf{H}}_{k_1,k_2,k_3}^{110},\ 0\le k_i\le\frac{N}{8}-1 \qquad (7.57)$$

$$\widehat{\mathbf{H}}_{k_1,k_2,k_3}^{111}=\mathbf{B}_{k_1,k_2,k_3}^{111}\mathbf{W}_{4^3}(\mathbf{S}_4\otimes\mathbf{S}_4\otimes\mathbf{S}_4)\widetilde{\mathbf{H}}_{k_1,k_2,k_3}^{111},\ 0\le k_i\le\frac{N}{8}-1 \qquad (7.58)$$

The components of the input and output vectors $\widetilde{\mathbf{H}}_{k_1,k_2,k_3}^{\cdots}$ and $\widehat{\mathbf{H}}_{k_1,k_2,k_3}^{\cdots}$ and those of their corresponding diagonal twiddle factor matrices $\mathbf{B}_{k_1,k_2,k_3}^{\cdots}$ in (7.52)-(7.58) are given by

$$\begin{cases} \widetilde{H}_{k_1,k_2,k_3}^{001}(a) = \widetilde{y}_{001}\left(k_1 + p\frac{N}{16}, k_2 + q\frac{N}{16}, k_3 + (\gamma-1)\frac{N}{16}\right) \\ \widehat{H}_{k_1,k_2,k_3}^{001}(a) = \widetilde{h}_{001}^{pq\gamma}(k_1,k_2,k_3) \\ B_{k_1,k_2,k_3}^{001}(a,a) = W_N^{pk_1+qk_2+\gamma k_3} \\ \text{where } a = 8p+2q+\frac{\gamma-1}{2},\ p,q = 0,2,4,6,\ \gamma = 1,3,5,7 \end{cases} \tag{7.59}$$

$$\begin{cases} \widetilde{H}_{k_1,k_2,k_3}^{010}(b) = \widetilde{y}_{010}\left(k_1 + p\frac{N}{16}, k_2 + (\beta-1)\frac{N}{16}, k_3 + q\frac{N}{16}\right) \\ \widehat{H}_{k_1,k_2,k_3}^{010}(b) = \widetilde{h}_{010}^{p\beta q}(k_1,k_2,k_3) \\ B_{k_1,k_2,k_3}^{010}(b,b) = W_N^{pk_1+\beta k_2+qk_3} \\ \text{where } b = 8p+2\beta-2+\frac{q}{2},\ p,q = 0,2,4,6,\ \beta = 1,3,5,7 \end{cases} \tag{7.60}$$

$$\begin{cases} \widetilde{H}_{k_1,k_2,k_3}^{011}(c) = \widetilde{y}_{011}\left(k_1 + p\frac{N}{16}, k_2 + (\beta-1)\frac{N}{16}, k_3 + (\gamma-1)\frac{N}{16}\right) \\ \widehat{H}_{k_1,k_2,k_3}^{011}(c) = \widetilde{h}_{011}^{p\beta\gamma}(k_1,k_2,k_3) \\ B_{k_1,k_2,k_3}^{011}(c,c) = W_N^{pk_1+\beta k_2+\gamma k_3} \\ \text{where } c = 8p+2\beta-2+\frac{\gamma-1}{2},\ p = 0,2,4,6,\ \beta,\gamma = 1,3,5,7 \end{cases} \tag{7.61}$$

$$\begin{cases} \widetilde{H}_{k_1,k_2,k_3}^{100}(d) = \widetilde{y}_{100}\left(k_1 + (\alpha-1)\frac{N}{16}, k_2 + p\frac{N}{16}, k_3 + q\frac{N}{16}\right) \\ \widehat{H}_{k_1,k_2,k_3}^{100}(d) = \widetilde{h}_{100}^{\alpha pq}(k_1,k_2,k_3) \\ B_{k_1,k_2,k_3}^{100}(d,d) = W_N^{\alpha k_1+pk_2+qk_3} \\ \text{where } d = 8(\alpha-1)+2p+\frac{q}{2},\ p,q = 0,2,4,6,\ \alpha = 1,3,5,7 \end{cases} \tag{7.62}$$

$$\begin{cases} \widetilde{H}_{k_1,k_2,k_3}^{101}(e) = \widetilde{y}_{101}\left(k_1 + (\alpha-1)\frac{N}{16}, k_2 + p\frac{N}{16}, k_3 + (\gamma-1)\frac{N}{16}\right) \\ \widehat{H}_{k_1,k_2,k_3}^{101}(e) = \widetilde{h}_{101}^{\alpha p\gamma}(k_1,k_2,k_3) \\ B_{k_1,k_2,k_3}^{101}(e,e) = W_N^{\alpha k_1+pk_2+\gamma k_3} \\ \text{where } e = 8(\alpha-1)+2p+\frac{\gamma-1}{2},\ p = 0,2,4,6,\ \alpha,\gamma = 1,3,5,7 \end{cases} \tag{7.63}$$

$$\begin{cases} \widetilde{H}^{110}_{k_1,k_2,k_3}(f) = \widetilde{y}_{110}\left(k_1+(\alpha-1)\frac{N}{16}, k_2+(\beta-1)\frac{N}{16}, k_3+p\frac{N}{16}\right) \\ \widehat{H}^{110}_{k_1,k_2,k_3}(f) = \widetilde{h}^{\alpha\beta p}_{110}(k_1,k_2,k_3) \\ B^{110}_{k_1,k_2,k_3}(f,f) = W_N^{\alpha k_1+\beta k_2+pk_3} \\ \text{where } f = 8(\alpha-1)+2\beta-2+\frac{p}{2},\ p=0,2,4,6,\ \alpha,\beta=1,3,5,7 \end{cases} \tag{7.64}$$

and

$$\begin{cases} \widetilde{H}^{111}_{k_1,k_2,k_3}(g) = \widetilde{y}_{111}\left(k_1+(\alpha-1)\frac{N}{16}, k_2+(\beta-1)\frac{N}{16}, k_3+(\gamma-1)\frac{N}{16}\right) \\ \widehat{H}^{111}_{k_1,k_2,k_3}(g) = \widetilde{h}^{\alpha\beta\gamma}_{111}(k_1,k_2,k_3) \\ B^{111}_{k_1,k_2,k_3}(g,g) = W_N^{\alpha k_1+\beta k_2+\gamma k_3} \\ \text{where } g = 8(\alpha-1)+2\beta-2+\frac{\gamma-1}{2},\ \alpha,\beta,\gamma=1,3,5,7 \end{cases} \tag{7.65}$$

In (7.52)-(7.58), the matrix $\mathbf{W}_{4^3}$, the operator of the $(4\times4\times4)$-point 3-D DFT, can be factored using the Kronecker product as [32]

$$\mathbf{W}_{4^3} = \prod_{l=1}^{3}\left(\mathbf{I}_{4^{l-1}}\otimes\mathbf{Z}_4\otimes\mathbf{I}_{4^{3-l}}\right)\left(\mathbf{I}_{4^{l-1}}\otimes\mathbf{D}_4\otimes\mathbf{I}_{4^{3-l}}\right)\left(\mathbf{I}_{4^{l-1}}\otimes\mathbf{R}_4\otimes\mathbf{I}_{4^{3-l}}\right) \tag{7.66}$$

For a given combination of (k_1,k_2,k_3), each of the seven general sub-butterflies given by (7.52)-(7.58) computes 64 points using (5.12). In order to compute, for a given combination of (k_1,k_2,k_3), all the points required in the computation of these sub-butterflies, we rearrange (5.12) as

$$\begin{aligned} \widetilde{\mathbf{y}}_{k_1+l_1\frac{N}{8},k_2+l_2\frac{N}{8},k_3+l_3\frac{N}{8}} &= \mathbf{H}_{2^3}\widetilde{\mathbf{x}}_{k_1+l_1\frac{N}{8},k_2+l_2\frac{N}{8},k_3+l_3\frac{N}{8}}, \\ 0\le k_i\le\frac{N}{8}-1,\ l_i&=0,1,2,3 \end{aligned} \tag{7.67}$$

It is clear from (5.13) and (5.14) that for a given combination of (k_1,k_2,k_3), (7.67) computes 512 points from the initial input sequence $\widetilde{x}(k_1,k_2,k_3)$, where 64 points are used to form the input sequence $\widetilde{y}_{000}(k_1,k_2,k_3)$ of the $\left(\frac{N}{2}\times\frac{N}{2}\times\frac{N}{2}\right)$-point 3-D DFT given by (5.4) and

all the other points are used by the general sub-butterflies given by (7.52)-(7.58). The output of these sub-butterflies are used to form the input sequences of the corresponding $\left(\frac{N}{8} \times \frac{N}{8} \times \frac{N}{8}\right)$-point 3-D DFTs given by (7.45)-(7.51). These operations are repeated for all the combinations of (k_1, k_2, k_3), $k_i = 0, 1, ..., (N/8) - 1$, $i = 1, 2, 3$. This completes the first stage of the proposed decomposition.

Thus, the proposed radix-$(2 \times 2 \times 2)/(8 \times 8 \times 8)$ FFT algorithm corresponds to decomposing the $(N \times N \times N)$-point 3-D DFT given by (5.1) into one $\left(\frac{N}{2} \times \frac{N}{2} \times \frac{N}{2}\right)$-point 3-D DFT given by (5.4) and 448 $\left(\frac{N}{8} \times \frac{N}{8} \times \frac{N}{8}\right)$-point 3-D DFTs given by (7.45)-(7.51), in the first stage. This is achieved by repeating $\frac{N^3}{512}$ times the general butterfly based on (7.67) and (7.52)-(7.58). This decomposition scheme is repeated successively for each of the new resulting DFTs, until the problem is reduced to some DFTs without twiddle factors, namely $(4 \times 4 \times 4)$- or $(2 \times 2 \times 2)$-point 3-D DFTs.

Let us now count the operations required by the general butterfly of the radix-$(2 \times 2 \times 2)/(8 \times 8 \times 8)$ FFT algorithm. This is achieved by counting all the necessary operations for a given combination of (k_1, k_2, k_3). According to (2.8), the matrix $\mathbf{H}_{2^3}$ introduces 24×64 complex additions in the computation of the operation given by (7.67). According to (7.66), the matrix $\mathbf{W}_{4^3}$ introduces $64 \times 2 \times 3$ complex additions in the computation of each of the seven general sub-butterflies given by (7.52)-(7.58). The computation of each of these sub-butterflies also requires 64 complex multiplications that are introduced by the corresponding twiddle factor matrix $\mathbf{B}^{\cdots}_{k_1,k_2,k_3}$, and 64 real multiplications and 64 real additions that are introduced by the corresponding diagonal matrix generated using $\mathbf{S}_4$. We count the operations required by the butterfly whether the complex multiplication is performed using the 4mult-2add scheme or using the 3mult-3add scheme. Consequently, the general butterfly of the radix-$(2 \times 2 \times 2)/(8 \times 8 \times 8)$ FFT algorithm requires 2240 real multiplications and 9792 real additions, if the 4mult-2add scheme is considered, and 1792 real multiplications and 10240 real additions, if the 3mult-3add scheme is considered.

In order to further reduce the number of operations, we consider some special combinations of (k_1, k_2, k_3) that involve some trivial operations. It is clear from (7.59)-(7.64) that each of the twiddle factor matrices $\mathbf{B}_{k,0,0}^{001}$, $\mathbf{B}_{0,k,0}^{001}$, $\mathbf{B}_{k,0,0}^{010}$, $\mathbf{B}_{0,0,k}^{010}$, $\mathbf{B}_{k,0,0}^{011}$, $\mathbf{B}_{0,0,k}^{100}$, $\mathbf{B}_{0,k,0}^{100}$, $\mathbf{B}_{0,k,0}^{101}$ and $\mathbf{B}_{0,0,k}^{110}$ requires only 48 complex multiplications, whereas each of the twiddle factor matrices $\mathbf{B}_{k,l,0}^{001}$, $\mathbf{B}_{k,0,l}^{010}$ and $\mathbf{B}_{0,k,l}^{100}$ requires 60 complex multiplications, for $k, l = 1, 2, ..., (N/8) - 1$ and $k, l \neq s$, where $s = \frac{N}{16}$. Since the sub-butterflies given by (7.52)-(7.58) require pre-multiplications (due to $\mathbf{S}_4$) and post-multiplications (due to $\mathbf{B}_{k_1,k_2,k_3}^{\cdots}$), Theorem 1 can be used effectively to combine the twiddle factors for the purpose of reducing the number of operations. For $M = 2$, it has been efficiently exploited in Section 7.2. Now, for $M = 3$, by appropriate application of this theorem in the expressions given by (7.52)-(7.58), we can show the following:

1) Only 64 real multiplications and 832 real additions are required for each of the following combinations of (k_1, k_2, k_3):

(a) $(0, 0, 0)$, $(0, s, 0)$, $(s, 0, 0)$ and $(s, s, 0)$ for the sub-butterfly given by (7.52)

(b) $(0, 0, 0)$, $(0, 0, s)$, $(s, 0, 0)$ and $(s, 0, s)$ for the sub-butterfly given by (7.53)

(c) $(0, 0, 0)$, $(s, 0, 0)$, $(0, s, s)$ and (s, s, s) for the sub-butterfly given by (7.54)

(d) $(0, 0, 0)$, $(0, 0, s)$, $(0, s, 0)$ and $(0, s, s)$ for the sub-butterfly given by (7.55)

(e) $(0, 0, 0)$, $(0, s, 0)$, $(s, 0, s)$ and (s, s, s) for the sub-butterfly given by (7.56)

(f) $(0, 0, 0)$, $(0, 0, s)$, $(s, s, 0)$ and (s, s, s) for the sub-butterfly given by (7.57)

(g) $(0, 0, 0)$, $(0, s, s)$, $(s, s, 0)$ and $(s, 0, s)$ for the sub-butterfly given by (7.58)

2) Only 64 complex multiplications and 384 complex additions are required for each of the following combinations of (k_1, k_2, k_3):

(a) $(0, 0, s)$, $(0, s, s)$, $(s, 0, s)$ and (s, s, s) for the sub-butterfly given by (7.52)

(b) $(0, s, 0)$, $(0, s, s)$, $(s, s, 0)$ and (s, s, s) for the sub-butterfly given by (7.53)

(c) $(0, 0, s)$, $(s, 0, s)$, $(0, s, 0)$ and $(s, s, 0)$ for the sub-butterfly given by (7.54)

(d) $(s,0,0)$, $(s,0,s)$, $(s,s,0)$ and (s,s,s) for the sub-butterfly given by (7.55)

(e) $(0,0,s)$, $(0,s,s)$, $(s,0,0)$ and $(s,s,0)$ for the sub-butterfly given by (7.56)

(f) $(0,s,0)$, $(0,s,s)$, $(s,0,0)$ and $(s,0,s)$ for the sub-butterfly given by (7.57)

(g) $(0,0,s)$, $(0,s,0)$, $(s,0,0)$ and (s,s,s) for the sub-butterfly given by (7.58)

7.3.1 Computational Complexity

In this section, we consider the performance of the proposed radix-$(2\times 2\times 2)/(8\times 8\times 8)$ FFT algorithm by analyzing its computational complexity and comparing it with that of the proposed radix-$(2\times 2\times 2)/(4\times 4\times 4)$ FFT algorithm introduced in Section 5.2.

7.3.1.1 Arithmetic Complexity

For both the radix-$(2\times 2\times 2)/(8\times 8\times 8)$ and radix-$(2\times 2\times 2)/(4\times 4\times 4)$ FFT algorithms, we count the total number of operations required by each algorithm using both the 3mult-3add and 4mult-2add schemes. In addition, we remove the trivial operations involved only in the special combinations of (k_1,k_2,k_3) defined in Section 7.3 for $s=N/16$ for the radix-$(2\times 2\times 2)/(8\times 8\times 8)$ algorithm, and similarly, for $s=N/8$ for the radix-$(2\times 2\times 2)/(4\times 4\times 4)$ algorithm.

Then, it can be shown that the expressions for the numbers of real multiplications and real additions required by the radix-$(2\times 2\times 2)/(8\times 8\times 8)$ FFT algorithm are, respectively,

$$\begin{aligned}&\mathrm{M}_{\mathrm{r2}/8}^{33}(3,N)\\&=\frac{7}{2}N^3-\frac{9}{16}N^2-36N-6448+\mathrm{M}_{\mathrm{r2}/8}^{33}\left(3,\frac{N}{2}\right)+448\mathrm{M}_{\mathrm{r2}/8}^{33}\left(3,\frac{N}{8}\right),\ N>8,\\&\qquad\qquad\mathrm{M}_{\mathrm{r2}/8}^{33}(3,8)=448,\ \mathrm{M}_{\mathrm{r2}/8}^{33}(3,4)=\mathrm{M}_{\mathrm{r2}/8}^{33}(3,2)=0\end{aligned}\quad(7.68)$$

and

$$\begin{aligned}
&\mathrm{A}^{33}_{\mathrm{r2}/8}(3,N) \\
&= 20N^3 - \frac{9}{16}N^2 - 36N - 6448 + \mathrm{A}^{33}_{r2/8}\left(3,\frac{N}{2}\right) + 448\mathrm{A}^{33}_{r2/8}\left(3,\frac{N}{8}\right),\ N>8, \\
&\qquad \mathrm{A}^{33}_{\mathrm{r2}/8}(3,8) = 9664,\ \mathrm{A}^{33}_{\mathrm{r2}/8}(3,4) = 768,\ \mathrm{A}^{33}_{\mathrm{r2}/8}(3,2) = 48, \quad (7.69)
\end{aligned}$$

if the 3mult-3add scheme is considered. The corresponding numbers are

$$\begin{aligned}
&\mathrm{M}^{42}_{\mathrm{r2}/8}(3,N) \\
&= \frac{35}{8}N^3 - \frac{3}{4}N^2 - 48N - 8000 + \mathrm{M}^{42}_{\mathrm{r2}/8}\left(3,\frac{N}{2}\right) + 448\mathrm{M}^{42}_{\mathrm{r2}/8}\left(3,\frac{N}{8}\right),\ N>8, \\
&\qquad \mathrm{M}^{42}_{\mathrm{r2}/8}(3,8) = 448,\ \mathrm{M}^{42}_{\mathrm{r2}/8}(3,4) = \mathrm{M}^{42}_{\mathrm{r2}/8}(3,2) = 0 \quad (7.70)
\end{aligned}$$

and

$$\begin{aligned}
&\mathrm{A}^{42}_{\mathrm{r2}/8}(3,N) \\
&= \frac{153}{8}N^3 - \frac{3}{8}N^2 - 24N - 4896 + \mathrm{A}^{42}_{\mathrm{r2}/8}\left(3,\frac{N}{2}\right) + 448\mathrm{A}^{42}_{\mathrm{r2}/8}\left(3,\frac{N}{8}\right),\ N>8, \\
&\qquad \mathrm{A}^{42}_{\mathrm{r2}/8}(3,8) = 9664,\ \mathrm{A}^{42}_{\mathrm{r2}/8}(3,4) = 768,\ \mathrm{A}^{42}_{\mathrm{r2}/8}(3,2) = 48, \quad (7.71)
\end{aligned}$$

if the 4mult-2add scheme is considered.

It can be further shown that the expressions for the numbers of real multiplications and real additions required by the radix-$(2\times 2\times 2)/(4\times 4\times 4)$ FFT algorithm are, respectively

$$\begin{aligned}
&\mathrm{M}^{33}_{\mathrm{r2}/4}(3,N) \\
&= \frac{21}{8}N^3 - \frac{9}{8}N^2 - 9N - 752 + \mathrm{M}^{33}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right) + 56\mathrm{M}^{33}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N>4, \\
&\qquad \mathrm{M}^{33}_{\mathrm{r2}/4}(3,4) = \mathrm{M}^{33}_{\mathrm{r2}/4}(3,2) = 0 \quad (7.72)
\end{aligned}$$

and

$$\begin{aligned}
&\mathrm{A}^{33}_{\mathrm{r2}/4}(3,N) \\
&\quad = \frac{111}{8}N^3 - \frac{9}{8}N^2 - 9N - 752 + \mathrm{A}^{33}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right) + 56\mathrm{A}^{33}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N > 4, \\
&\qquad\qquad \mathrm{A}^{33}_{\mathrm{r2}/4}(3,4) = 768,\ \mathrm{A}^{33}_{\mathrm{r2}/4}(3,2) = 48, \quad (7.73)
\end{aligned}$$

if the 3mult-3add scheme is considered. The corresponding numbers are

$$\begin{aligned}
&\mathrm{M}^{42}_{\mathrm{r2}/4}(3,N) \\
&\quad = \frac{7}{2}N^3 - \frac{3}{2}N^2 - 12N - 1152 + \mathrm{M}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right) + 56\mathrm{M}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N > 4, \\
&\qquad\qquad \mathrm{M}^{42}_{\mathrm{r2}/4}(3,4) = \mathrm{M}^{42}_{\mathrm{r2}/4}(3,2) = 0 \quad (7.74)
\end{aligned}$$

and

$$\begin{aligned}
&\mathrm{A}^{42}_{\mathrm{r2}/4}(3,N) \\
&\quad = 13N^3 - \frac{3}{4}N^2 - 6N - 352 + \mathrm{A}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{2}\right) + 56\mathrm{A}^{42}_{\mathrm{r2}/4}\left(3,\frac{N}{4}\right),\ N > 4, \\
&\qquad\qquad \mathrm{A}^{42}_{\mathrm{r2}/4}(3,4) = 768,\ \mathrm{A}^{42}_{\mathrm{r2}/4}(3,2) = 48, \quad (7.75)
\end{aligned}$$

if the 4mult-2add scheme is considered.

The arithmetic complexities of the two algorithms for complex data for various values of N are given in Tables 7.4 and 7.5. It is clear from these tables that the radix-$(2\times2\times2)$/$(8\times8\times8)$ FFT algorithm reduces the number of arithmetic operations, especially the number of real multiplications, compared to that required by the radix-$(2\times2\times2)$/$(4\times4\times4)$ FFT algorithm. For example, if the 4mult-2add scheme is considered, savings of about 14% in the number of multiplications can be achieved using the radix-$(2\times2\times2)$/$(8\times8\times8)$ FFT algorithm. It should be pointed out that additional reductions in

Table 7.4: Number of arithmetic operations required for the computation of the 3-D DFT using the 3mult-3add scheme

	Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FFT		Proposed radix-$(2\times2\times2)/(8\times8\times8)$ FFT		Savings	
Transform size $N\times N\times N$	Mults. /point	Adds. /point	Mults. /point	Adds. /point	Mults. (%)	Adds. (%)
$2^3\times2^3\times2^3$	0.8750	18.8750	0.8750	18.8750	0	0
$2^4\times2^4\times2^4$	2.4453	26.4453	1.8594	25.8594	23.96	2.21
$2^5\times2^5\times2^5$	3.6294	33.6294	3.4829	33.4829	4.03	0.43
$2^6\times2^6\times2^6$	5.1957	41.1957	4.6588	40.6588	10.33	1.30
$2^7\times2^7\times2^7$	6.4405	48.4405	5.6996	47.6996	11.50	1.52
$2^8\times2^8\times2^8$	7.9717	55.9717	7.2569	55.2569	8.96	1.27
$2^9\times2^9\times2^9$	9.2546	63.2546	8.4823	62.4823	8.34	1.22
$2^{10}\times2^{10}\times2^{10}$	10.7560	70.7560	9.5469	69.5469	11.24	1.70
$2^{11}\times2^{11}\times2^{11}$	12.0668	78.0668	11.0428	77.0428	8.48	1.31
$2^{12}\times2^{12}\times2^{12}$	13.5445	85.5445	12.3022	84.3022	9.17	1.45

Table 7.5: Number of arithmetic operations required for the computation of the 3-D DFT using the 4mult-2add scheme

	Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FFT		Proposed radix-$(2\times2\times2)/(8\times8\times8)$ FFT		Savings	
Transform size $N\times N\times N$	Mults. /point	Adds. /point	Mults. /point	Adds. /point	Mults. (%)	Adds. (%)
$2^3\times2^3\times2^3$	0.8750	18.8750	0.8750	18.8750	0	0
$2^4\times2^4\times2^4$	3.1875	25.7031	2.2969	25.4219	27.94	1.09
$2^5\times2^5\times2^5$	4.5703	32.6885	4.3477	32.6182	4.87	0.21
$2^6\times2^6\times2^6$	6.8296	39.5618	5.6301	39.6875	17.56	-0.31
$2^7\times2^7\times2^7$	8.3397	46.5412	7.0759	46.3233	15.15	0.46
$2^8\times2^8\times2^8$	10.5122	53.4312	9.0596	53.4542	13.81	-0.04
$2^9\times2^9\times2^9$	12.1083	60.4010	10.4321	60.5325	13.84	-0.21
$2^{10}\times2^{10}\times2^{10}$	14.2103	67.3017	11.8697	67.2241	16.45	0.11
$2^{11}\times2^{11}\times2^{11}$	15.8703	74.2632	13.7854	74.3002	13.13	-0.04
$2^{12}\times2^{12}\times2^{12}$	17.9174	81.1717	15.2261	81.3784	15.02	-0.25

the number of arithmetic operations in the two algorithms can be obtained by removing all the trivial operations.

7.3.1.2 Data Transfers

We first give a brief implementation scheme of the proposed radix-$(2 \times 2 \times 2)/(8 \times 8 \times 8)$ FFT algorithm by considering the implementation of the butterfly given by (7.67) and (7.52)-(7.58). For a given combination of (k_1, k_2, k_3), by reading eight points from the off-chip memory according to (5.13) to compute the operation given by (5.12), we obtain the eight points given by (5.14). The first point $\widetilde{y}_{000}(k_1, k_2, k_3)$ of (5.14) is returned to the off-chip memory, whereas the other seven points are kept in the on-chip memory since they are used by the seven sub-butterflies given by (7.52)-(7.58). These operations are repeated 64 times to complete the computation of the operation given by (7.67). The seven 64-point results kept in the on-chip memory are used to process the seven sub-butterflies. This completes the process of the butterfly. For all the combinations of (k_1, k_2, k_3), $k_i = 0, 1, ..., (N/8) - 1$, the results returned to the off-chip memory are grouped to form the input sequence $\widetilde{y}_{000}(k_1, k_2, k_3)$ of the $\left(\frac{N}{2} \times \frac{N}{2} \times \frac{N}{2}\right)$-point 3-D DFT given by (5.4). The 448 input sequences of the $\left(\frac{N}{8} \times \frac{N}{8} \times \frac{N}{8}\right)$-point 3-D DFTs given by (7.45)-(7.51) are formed by grouping separately the results of each line of the output vectors given by (7.52)-(7.58). This completes the first stage of the proposed decomposition.

To count the numbers of data transfers required by the radix-$(2 \times 2 \times 2)/(8 \times 8 \times 8)$ and radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ FFT algorithms, we assume that the $(2 \times 2 \times 2)$- and $(4 \times 4 \times 4)$-point DFTs are implemented separately for the two algorithms. In addition, we assume that the general butterflies of the two algorithms are implemented without additional transfers. Then, it can be shown that the expressions for the number of data transfers (real and imaginary parts), not including the read operations for the twiddle factors, required by

these two algorithms are

$$\begin{aligned} \mathrm{D}_{\mathrm{r2/8}}(3,N) &= 2N^3 + \mathrm{D}_{\mathrm{r2/8}}\left(3,\frac{N}{2}\right) + 448\mathrm{D}_{\mathrm{r2/8}}\left(3,\frac{N}{8}\right),\ N>4, \\ \mathrm{D}_{\mathrm{r2/8}}(3,4) &= 128,\ \mathrm{D}_{\mathrm{r2/8}}(3,2) = 16,\ \mathrm{D}_{\mathrm{r2/8}}(3,1) = 0 \end{aligned} \tag{7.76}$$

and

$$\begin{aligned} \mathrm{D}_{\mathrm{r2/4}}(3,N) &= 2N^3 + \mathrm{D}_{\mathrm{r2/4}}\left(3,\frac{N}{2}\right) + 56\mathrm{D}_{\mathrm{r2/4}}\left(3,\frac{N}{4}\right),\ N>4, \\ \mathrm{D}_{\mathrm{r2/4}}(3,4) &= 128,\ \mathrm{D}_{\mathrm{r2/4}}(3,2) = 16,\ \mathrm{D}_{\mathrm{r2/4}}(3,1) = 0 \end{aligned} \tag{7.77}$$

respectively. The numbers of data transfers for the two algorithms are compared in Table 7.6, for various values of N. It is clear from this table that the radix-$(2\times2\times2)/(8\times8\times8)$ FFT algorithm requires about 25% less data transfer operations than that required in the radix-$(2\times2\times2)/(4\times4\times4)$ FFT algorithm. Hence, similar reduction is also achieved in the address generation.

7.3.1.3 Twiddle Factors

In the radix-$(2\times2\times2)/(8\times8\times8)$ and radix-$(2\times2\times2)/(4\times4\times4)$ FFT algorithms, 896 and 112 real coefficients (cosine and sine) are respectively required to be evaluated or loaded from the lookup table by the corresponding general butterfly using the 4mult-2add scheme. Therefore, by counting the required number of the general butterflies in the two algorithms, and assuming that the coefficients required in the special combinations of (k_1,k_2,k_3), such as $\sqrt{2}/2$, $\cos(\pi/8)$ and $\sin(\pi/8)$ are initialized and kept in the internal registers of the processor during the processing time of the corresponding algorithm, the number of real coefficient evaluations or accesses to the lookup table for the radix-

Table 7.6: Comparison of the number of data transfers, and twiddle factor evaluations or accesses to the lookup table

	Proposed radix-$(2\times2\times2)/(4\times4\times4)$ FFT		Proposed radix-$(2\times2\times2)/(8\times8\times8)$ FFT		Savings (%)	
Transform size $N\times N\times N$	$D_{r2/4}(3,N)/N^3$ (A)	$T^{42}_{r2/4}(3,N)/N^3$ (B)	$D_{r2/8}(3,N)/N^3$ (C)	$T^{42}_{r2/8}(3,N)/N^3$ (D)	in C over A	in D over B
$2^3\times2^3\times2^3$	4	0	2.25	0	43.75	0
$2^4\times2^4\times2^4$	4.25	1.4844	4.0312	0	5.14	100
$2^5\times2^5\times2^5$	6.0312	1.8818	4.2539	1.5107	29.46	19.72
$2^6\times2^6\times2^6$	6.4727	3.2678	4.5005	1.9016	30.46	41.80
$2^7\times2^7\times2^7$	8.0864	3.7985	6.0899	1.9801	24.68	47.87
$2^8\times2^8\times2^8$	8.6744	5.0811	6.4834	3.3172	25.25	34.71
$2^9\times2^9\times2^9$	10.1599	5.7073	6.7484	3.8277	33.57	32.93
$2^{10}\times2^{10}\times2^{10}$	10.8601	6.9086	8.1722	3.9607	24.75	42.67
$2^{11}\times2^{11}\times2^{11}$	12.2474	7.6071	8.6945	5.1474	29.00	32.33
$2^{12}\times2^{12}\times2^{12}$	13.0335	8.7458	8.9916	5.7426	31.01	34.33

$(2\times2\times2)/(8\times8\times8)$ and radix-$(2\times2\times2)/(4\times4\times4)$ FFT algorithms are, respectively

$$\begin{aligned}&T^{42}_{r2/8}(3,N)\\&\quad=\frac{7}{4}N^3-\frac{3}{8}N^2-24N-6688+T^{42}_{r2/8}\left(3,\frac{N}{2}\right)+448T^{42}_{r2/8}\left(3,\frac{N}{8}\right),\ N>8,\\&\qquad\qquad T^{42}_{r2/8}(3,8)=T^{42}_{r2/8}(3,4)=T^{42}_{r2/8}(3,2)=0\end{aligned}\quad(7.78)$$

and

$$\begin{aligned}&T^{42}_{r2/4}(3,N)\\&\quad=\frac{7}{4}N^3-\frac{3}{4}N^2-6N-800+T^{42}_{r2/4}\left(3,\frac{N}{2}\right)+56T^{42}_{r2/4}\left(3,\frac{N}{4}\right),\ N>4,\\&\qquad\qquad T^{42}_{r2/4}(3,4)=T^{42}_{r2/4}(3,2)=0\end{aligned}\quad(7.79)$$

For various values of N, the numbers of real coefficient evaluations or accesses to the lookup table required by the two algorithms are compared in Table 7.6. It is seen from this

table that savings of over 30% in the evaluation of twiddle factors or in the access to the lookup table can be easily achieved using the radix-$(2 \times 2 \times 2)/(8 \times 8 \times 8)$ algorithm over the radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ FFT algorithm. Note that, when the lookup table is used, similar savings are obtained in the address generation for reading the twiddle factors.

7.4 Summary

In this chapter, the radix-2/8 approach, proposed in Chapter 6 for the computation of the 1-D transforms, has been extended for the computation of the 2-D and 3-D DFTs. The two-step decomposition strategy, presented in Chapter 5, has been advanced in this chapter to introduce radix-(2×2), radix-(8×8), radix-$(2 \times 2 \times 2)$ and radix-$(8 \times 8 \times 8)$ index maps in the decomposition of the 2-D and 3-D DFTs. This has enabled the design of radix-$(2 \times 2)/(8 \times 8)$ and radix-$(2 \times 2 \times 2)/(8 \times 8 \times 8)$ FFT algorithms whose butterflies are characterized by simple closed-form expressions that facilitate easy implementations of the algorithms. The two new algorithms have been compared, respectively, with the existing radix-$(2 \times 2)/(4 \times 4)$ FFT algorithm and the radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ FFT algorithm proposed in Chapter 5. It has been shown that the radix-$(2 \times 2)/(8 \times 8)$ and radix-$(2 \times 2 \times 2)/(8 \times 8 \times 8)$ FFT algorithms provide better performance in terms of the number of arithmetic operations, data transfers, address generations, and twiddle factor evaluations or accesses to the lookup table, thus making the radix-2/8 approach superior to all the other existing radix-based approaches in providing efficient 2-D and 3-D FFT algorithms.

Chapter 8

Conclusion

8.1 Concluding Remarks

The DFT and DHT play a key role in one- and multi-dimensional signal processing applications. Due to the recent advances in the digital technology and the present demands of such transforms in low-power high-performance real-time applications, fast and efficient computation of these transforms is extremely crucial. This thesis has been concerned with developing new techniques and approaches for the design of FFT and FHT algorithms with a view to reducing the computational and structural complexities globally and comprehensively. With this objective in mind, this thesis has proposed a number of one and multi-dimensional radix-based FFT and FHT algorithms, which in most cases significantly reduce the operations such as multiplications, additions, data transfers, address generations, and twiddle factor evaluations or accesses to the lookup table. Further, the proposed algorithms possess features such as simplicity, regularity, modularity, easy indexing scheme, and butterfly-style and in-place computations that are highly desirable characteristics for software or hardware implementations of the algorithms.

Due to the growing interest in applications that involve three and higher dimension

signals, the M-D transforms are expected to be widely used and hence, their efficient computations are highly desirable. Although the kernel of the M-D DHT is not separable, it has been shown that by using an appropriate index mapping and the Kronecker product, the generalization of the radix-2 approach for the computation of the DHT of an arbitrary dimension is possible. For this purpose, a general decomposition method has been proposed to develop M-D radix-$(2 \times 2 \times ... \times 2)$ DIF and DIT FHT algorithms that have been shown to be more effective and highly suitable for hardware and software implementations compared to all the existing M-D FHT algorithms. The proposed method has also been used to reformulate the existing M-D radix-$(2 \times 2 \times ... \times 2)$ complex-valued DIF and DIT FFT algorithms. All the butterflies of the proposed algorithms are based on simple closed-form expressions that allow for easy implementations of these algorithms for any dimension. A new approach for computing the DFT and DHT using a unified structure has been introduced and a close relationship between these FHT and FFT algorithms, valid for any dimension, has been established. This type of relationship is of great significance for software and hardware implementations of the algorithms, since it has been shown that because of this relationship and the fact that the DHT is an alternative to the DFT for real data, a single module with a little or no modification can be used to carry out the forward and inverse M-D DFTs for real- or complex-valued data and M-D DHTs. Thus, the same module, with a little or no modification, can be used to cover all domains of applications that involve the DFTs or DHTs.

Based on the radix-2 approach, a method has been proposed for reducing significantly the complexity in the computation of the 1-D and 2-D DFTs, where only a sub-set of output samples of the transforms are needed. This has been achieved by grouping in a radix-2 or radix-(2×2) FFT algorithm all the stages that involve unnecessary operations into a single stage and introducing a new recursive technique for the computations required in the resulting stage. Due to this grouping and an efficient indexing process, the implementation

of the proposed pruning DIT or DIF FFT algorithm requires a minimum number of stages. It has been shown that the proposed 1-D and 2-D FFT algorithms substantially reduce the structural and computational complexities compared to those of the corresponding existing ones.

Even though the radix-2 approach introduced by Cooley and Tukey in 1965 has led to algorithms having the lowest structural complexities, the computational complexities of these algorithms are still higher than those provided by a higher radix approach. In order to further improve the performance of the higher-radix FFT algorithms, namely radix-4, radix-8 and radix-16 FFT algorithms, an efficient technique has been proposed by introducing new indices for some of the sub-sequences resulting from the conventional decompositions in the radix-4, radix-8, and radix-16 FFT algorithms. It has been shown that significant savings in the number of twiddle factor evaluations or accesses to the lookup table can be achieved using the proposed algorithms compared to those of the conventional ones. This has been achieved without imposing any additional computational or structural complexities in the algorithms.

The radix-2/4 approach introduced by Duhamel and Hollmann in 1984 has led to 1-D and 2-D FFT and FHT algorithms having a good compromise between the arithmetic and structural complexities. However, its extension to three or higher dimensions is neither obvious nor straightforward. In order to overcome this problem and reduce the complexity in the computation of the 3-D DFT and DHT, a two-step decomposition strategy, suitable for the extension of the radix-2/4 approach, coupled with an efficient index mapping and the Kronecker product has been developed. This has led to new radix-$(2 \times 2 \times 2)/(4 \times 4 \times 4)$ DIF FFT, and DIF and DIT FHT algorithms whose butterflies are characterized by simple closed-form expressions that facilitate easy implementations of the algorithms. It has been shown that these algorithms substantially reduce the complexities as compared with those of the respective existing 3-D FFT and FHT algorithms.

The radix-2/4 approach leads to 1-D FFT and FHT algorithms requiring the lowest number of arithmetic operations, while maintaining simple and regular structures. This is due to the fact that the radix-2/4 approach takes advantage of the lowest structural complexity provided by the radix-2 approach and reduced computational complexity provided by the radix-4 approach. In order to take advantage of the lower computational complexity offered by the radix-8 approach in reducing the complexity in the computation of the 1-D DFT and DHT of lengths $N = q \times 2^m$, a new radix-2/8 approach has been proposed. This has been achieved by an appropriate simultaneous use of the radix-2 and radix-8 approaches, which has led to new radix-2/8 FFT and FHT algorithms. It has been shown that the proposed radix-2/8 algorithms maintain all the advantages of the radix-2/4 algorithms and minimize the number of data transfers, address generations and twiddle factor evaluations or accesses to the lookup table.

Motivated by the success of the proposed radix-2/8 approach in providing efficient 1-D algorithms, the problem of reducing the complexity in the computation of the 2-D and 3-D DFTs has been addressed by exploiting the concept of the proposed approach. A two-step decomposition strategy, suitable for the extension of the radix-2/8 approach, coupled with an efficient index mapping and the Kronecker product has been introduced to develop new radix-(2×2)/(8×8) and radix-$(2 \times 2 \times 2)$/$(8 \times 8 \times 8)$ DIF FFT algorithms whose butterflies are characterized by simple closed form expressions. It has been shown that the proposed radix-(2×2)/(8×8) and radix-$(2 \times 2 \times 2)$/$(8 \times 8 \times 8)$ FFT algorithms not only significantly reduce the number of data transfers, address generations and twiddle factor evaluations or accesses to the lookup table, but also the number of arithmetic operations as compared to those of the existing radix-(2×2)/(4×4) FFT algorithm and the radix-$(2 \times 2 \times 2)$/$(4 \times 4 \times 4)$ FFT algorithm presented in Chapter 5. Therefore, the proposed radix-2/8 approach is superior to all the other existing radix-based approaches in providing efficient 1-D, 2-D and 3-D FFT, and 1-D FHT algorithms.

8.2 Scope for Further Investigation

One important contributions of this thesis has been the development of a general decomposition method, which is based on the radix-2 approach, valid for any dimension and applicable to the cases of both the DHT and DFT, and which significantly reduces the complexity of the FHT algorithms. Similar generalizations of the radix-2/4 and radix-2/8 approaches can be investigated for further reducing the complexity of the algorithms. The philosophy behind computing the DFT and DHT using a unified structure introduced in this thesis for the case of the radix-2 approach could be used for the cases of the radix-2/4 and radix-2/8 approaches.

One way of computing a very high resolution spectrum is to pad the input sequence with a large number of zeros before computing the DFT. In this case, the direct use of the FFT is not attractive. Therefore, an input-pruning method similar to the one for the output-pruning proposed in this thesis would be highly desirable to increase the efficiency of FFT algorithms in such a case. Moreover, the proposed method may be adapted for pruning input and output samples of the DFT simultaneously.

The basic idea for improved higher-radix FFT algorithms proposed in this thesis can also be applied to other higher radices DIT and DIF FFT and FHT algorithms as well as to M-D FFT and FHT algorithms.

The lowest number of arithmetic operations has been achieved in the 1-D case using the radix-2/4 and radix-2/8 approaches. However, in the case of higher dimensions, the radix-2/8 approach provides a reduced number of arithmetic operations, but not the lowest one. Therefore, further work can be undertaken to develop new radix-based approaches that provide the lowest number of arithmetic operations in the case of higher dimensions, while maintaining the advantages offered by the radix-2/8 approach.

The problem of reducing the complexity in the computation of other interesting discrete transforms, such as the discrete cosine, sine, Walsh, and wavelet transforms, can be undertaken by using some of the approaches and techniques introduced in this thesis.

The focus of the thesis has been the development of reduced-complexity FFT and FHT algorithms by directing our effort at the algorithm design level so that such reductions are achieved naturally when the algorithms are implemented. It would be a worthwhile exercise to investigate the DSP and VLSI implementations of the proposed FFT and FHT algorithms.

References

[1] E. O. Brigham, *The fast Fourier transform and its applications, Englewood Cliffs.* NJ: Prentice-Hall, 1988.

[2] O. K. Ersoy, *Fourier-Related Transforms, Fast Algorithms and Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1997.

[3] J. W. Cooley and J. W. Tukey, "An algorithm for machine computation of complex Fourier series," *Math. Compt.*, vol. 9, pp. 297-301, 1965.

[4] G. D. Bergland, "A fast Fourier transform algorithm using base 8 iterations," *Math. Comput.*, vol. 22, pp. 275-279. Apr. 1968.

[5] R. C. Singleton, "An algorithm for computing the mixed-radix fast Fourier transform," *IEEE Trans. Audio Electroacoust.*, vol. 17, pp. 93-103, June 1969.

[6] D. P. Kolba and T. W. Parks, "A prime factor FFT algorithm using high-speed convolution," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 25, pp. 281-294, Aug. 1977.

[7] S. Winograd, "On computing the discrete Fourier transform," *Math. Comput.*, vol. 32, pp. 175-199, Jan. 1978.

[8] E. Dubois, and A. Venetsanopoulos, "A new algorithm for the radix-3 FFT," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 26, pp. 222-225, Jun 1978.

[9] C.S. Burrus, and P.W. Eschenbacher, "An in-place, in-order prime factor FFT algorithm," *IEEE Trans. Audio Electroacoust.*, vol. 29, pp. 806-817, Aug. 1981.

[10] R. D. Preuss, " Very fast computation of the radix-2 discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 30, pp. 595-607, Aug. 1982.

[11] P. Duhamel and H. Hollmann, "Split radix FFT algorithm," *Electron. Lett.*, vol. 20, pp. 14-16, Jan. 1984.

[12] E. E. Swartzlander, W. K. W. Young and S. J. Joseph, "A radix-4 delay commutator for fast Fourier transform processor implementation," *IEEE J. Solid-State Circuits*, vol. 19, Oct. 1984.

[13] Y. Suzuki, T. Sone and K. Kido, "A new algorithm of radix 3, 6, and 12," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 34, pp. 380-383, Feb. 1986.

[14] M. Vetterli and P. Duhamel, "Split-radix algorithms for length-p^m DFT's," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 57-64, Jan. 1989.

[15] P. Duhamel and M. Vetterli, "Fast Fourier transforms: A tutorial review and a state of the art," *Signal Processing,* vol. 19, pp. 259-299, 1990.

[16] D. Sundararajan, M. Omair Ahmad and M. N. S. Swamy, "Vector computation of the discrete Fourier transform," *IEEE Trans. Circuits and Syst. II: Analog and Digital Signal Processing*, vol. 45, pp. 449-461, Apr. 1998.

[17] G. Bi and Y. Q. Chen, "Fast DFT algorithms for length $N = q * 2^m$," *IEEE Trans. Circuits and Syst. II: Analog and Digital Signal Processing*, vol. 45, pp. 685-690, June 1998.

[18] D. Sundararajan, M. Omair Ahmad, "Index mapping approach of deriving the PM DFT algorithms," *IEEE Trans. Computers*, vol. 47, pp. 1418-1424, Dec. 1998.

[19] A. M. Grigoryan and S. S. Agaian, "Split manageable efficient algorithm for Fourier and Hadamard transforms," *IEEE Trans. Signal Processing*, vol. 48, pp. 172-182, Jan. 2000.

[20] D. Takahashi, "An extended split-radix FFT algorithm," *IEEE Signal Processing Letters*, vol. 8, pp. 145-147, May 2001.

[21] V. P. Rodriguez, "A radix-2 FFT algorithm for modern single instruction multiple data (SIMD) architectures," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, vol. 3, May 2002, pp. 3220-3223.

[22] G.E. Rivard, "Algorithm for direct fast Fourier transform of bivariant functions," *1975 Annual Meeting of the Optical Society of America, Boston, Massachusetts*, Oct. 1975.

[23] D. B. Harris and J. H. McClellan, "Vector radix fast Fourier transform," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, vol. 2, May 1977 , pp. 548-551.

[24] R. Mersereau and T. Speake, "A unified treatment of Cooley-Tukey algorithms for the evaluation of the multidimensional DFT," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 29, pp. 1011-1017, Oct. 1981.

[25] H. J. Nussbaumer, *Fast Fourier transform and convolution algorithms*. New York: Springer-Verlag, 1981.

[26] D. Dudgeon and R. Mersereau, *Multidimensional digital signal processing*. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[27] S. C. Pei and J. L. Wu, "split vector-radix 2D fast Fourier transform," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, vol. 12, Apr. 1987, pp. 1987-1990.

[28] H. R. Wu, and F. J. Paolomi, "On the two-dimensional vector split-radix FFT algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1302-1304, Aug. 1989.

[29] S. C. Chan and K. L. Ho, " Split vector-radix fast Fourier transform," *IEEE Trans. Signal Processing*, vol. 40 , pp. 2029-2039, Aug. 1992.

[30] G. Bi, and Y. Chen, "Split-radix algorithm for 2D DFT," *Electron. Lett.*, vol. 33, pp. 203-205, Jan. 1997.

[31] D. Sevic, "On computing 2-D FFT," *IEEE Trans. Signal Processing*, vol. 47, pp. 1428-1431, May. 1999.

[32] S. Bouguezal, D. Chikouche and A Khellaf, " An efficient algorithm for the computation of the multidimensional discrete Fourier transform," *Multidimensional Systems and Signal processing*, vol. 10, pp. 275-304, July 1999.

[33] D. Chikouche, A. Khellaf an S. Bouguezel, " A new proposed algorithm of arbitrary radix for the computation of the 2D DFT," *International Journal for Numerical Methods in Engineering.*, vol. 46, pp.103-115, Sep. 1999.

[34] C. Lizhi and Z. Yonghong, "Multidimensional polynomial transform algorithm for multidimensional DFT," *Electron. Lett.*, vol. 36, pp. 990-991, May 2000.

[35] R. Bernardini, "A new multidimensional FFT based on one-dimensional decomposition," *IEEE Trans. Circuits and Syst. II: Analog and Digital Signal Processing*, vol. 47, pp. 1123-1126, Oct. 2000.

[36] G. D. Bergland, "A radix-eight fast Fourier transform subroutine for real-valued series," *IEEE Trans. Audio Electroacoust.*, vol. 17, pp. 138-144, June 1969.

[37] I. Pitas and M. Strintzis, "General in-place calculation of discrete Fourier transforms of multidimensional sequences," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 34, pp. 565-572, June 1986.

[38] H. V. Sorensen, D. L. Jones, M. T. Heideman, and C. S. Burrus, "Real-valued fast Fourier transform algorithms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 35, pp. 849-863, June 1987.

[39] Z. -J. Mou and P. Duhamel, "In-place butterfly-style FFT of 2-D real sequences," *IEEE Trans. Signal Processing*, vol. 36, pp. 1642-1650, Oct. 1988.

[40] D. Sundararajan, M. Omair Ahmad and M. N. S. Swamy, " Fast computation of the discrete Fourier transform of real data," *IEEE Trans. Signal Processing*, vol. 45, pp. 2010-2022, Aug. 1997.

[41] B. R. Sekhar and K. M. M. Prabhu, "Radix-2 decimation-in-frequency algorithm for the computation of the real-valued FFT," *IEEE Trans. Signal Processing, vol.* 47, pp. 1181-1184, Apr. 1999.

[42] R. N. Bracewell, "Discrete Hartley transform," *J. Opt. Soc. Amer.*, vol. 73, pp. 1832-1835, Dec. 1983.

[43] O. Buneman, "Multi-dimensional Hartley transform," *Proc. IEEE*, vol. 75, pp. 267, 1987.

[44] R. N. Bracewell, "The fast Hartley transform," *Proc. IEEE*, vol. 72, pp. 1010-1018, Aug. 1984.

[45] H. V. Sorensen, D. L. Jones, C. S. Burrus and M. T. Heideman, "On computing the discrete Hartley transform," *IEEE Trans. Acoust. Speech Signal Processing*, vol. 33, pp. 1231-1238, Oct. 1985.

[46] H. V. Hou, "The fast Hartley transform algorithm," *IEEE Trans. Comput.*, vol. 36, pp. 147-156, Feb. 1987.

[47] P. Duhamel and M. Vetterli, "Improved Fourier and Hartley transform algorithms: Applications to cyclic convolution of real data," *IEEE Trans. Signal Processing*, vol. 35, pp. 818-824, Jun 1987.

[48] Y. Dekun, "Prime factor fast Hartley transform," *Electron. Lett.*, vol. 26, pp. 119-121, Jan. 1990.

[49] N. Aupindi, S. B. Narayanan and K. M. M. Prabhu, "New radix-3 FHT algorithm," *Electron. Lett.*, vol. 26, pp. 1537-1538, Aug. 1990.

[50] D. P. K. Lun and W. -C. Siu, "On prime factor mapping for the discrete Hartley transform," *IEEE Trans. Signal Processing*, vol. 40, pp. 1399-1411, June 1992.

[51] P. K. Meher, J. K. Satapathy and G. Panda, "New high-speed prime-factor algorithm for discrete Hartley transform," *IEE Proc. F., Radar Signal Processing,* vol. 140, pp. 63-70, Feb. 1993.

[52] D. P. -K. Lun and W. -C. Siu, "Fast radix-3/9 discrete Hartley transform," *IEEE Trans. Acoust. Speech Signal Processing*, vol. 41, pp. 2494-2499, July 1993.

[53] G. Bi., "Split-radix algorithm for the discrete Hartley transform," *Electron. Lett.*, vol. 30, pp. 1833-1835, Oct. 1994.

[54] G. Bi, "Radix-3/9 FHT algorithm," *Electron. Lett.*, vol. 31, pp. 166-168, Feb. 1995.

[55] G. Bi., "New split-radix algorithm for the discrete Hartley transform," *IEEE Trans. Signal Processing*, vol. 45, pp. 297-302, Feb. 1997.

[56] G. Bi and Y. Q. Chen, "Fast DHT algorithms for length $N = q * 2^m$," *IEEE Trans. Signal Processing*, vol. 47, pp. 900-903, Mar. 1999.

[57] N. Vijayakumar and K. M. M. Prabhu, "FHT algorithm for length $N = q * 2^m$," *Electron. Lett.*, vol. 35, pp. 966-968, Jun. 1999.

[58] R. Kumaresan and P. K. Gupta, "Vector-radix algorithm for 2-D discrete Hartley transform," *Proc. IEEE*, vol. 74, pp. 755-757, May 1986.

[59] S. Boussakta and A. G. J. Holt, "Fast multidimensional discrete Hartley transform using Fermat number transform," *IEE Proc. G., Circuits, Devices, Syst.*, vol. 135, pp. 253-257, Dec. 1988.

[60] E. A. Jonckheere and C. Ma, "Split-radix fast Hartley transform in one and two dimensions," *IEEE Trans. Acoust. Speech Signal Processing*, vol 39, pp. 499-503, Feb. 1991.

[61] P. K. Meher, J. K. Satapathy, and G. Panda, "Fast computation of multidimensional discrete Hartley transform," *Electron. Lett.*, vol.28, pp. 1077-1078, June 1992.

[62] J.-L. Wu, and S.-C. Pei, "The vector split-radix algorithm for 2-D DHT," *IEEE Trans. Signal Processing*, vol. 41, pp. 960-965, Feb. 1993.

[63] T. Bortfield and W. Dinter, "Calculation of multidimensional Hartley transforms using one-dimensional Fourier transforms," *IEEE Trans. Signal Processing*, vol. 43, pp. 1306-1310, May 1995.

[64] G. Bi, A.C. Kot and Z. Meng, "Computation of 2d discrete Hartley transform," *Electron. Lett.*, vol. 34, pp. 1058-1059, May 1998.

[65] S. Boussakta, O. Alshibami and M. Y. Aziz, "Radix-$2 \times 2 \times 2$ algorithm for the 3-D discrete Hartley transform," *IEEE Trans. Acoust. Speech Signal Processing*, vol. 49, pp. 3145-3156, Dec. 2001.

[66] O. Alshibami and S. Boussakta, "Fast 3-D decimation-in-frequency algorithm for 3-D Hartley transform," *Signal Processing*, vol. 82, pp. 121-126, Jan. 2002.

[67] Y. Zeng, G. Bi, and A.C. Kot, "Fast algorithm for multi-dimensional discrete Hartley transform with size $q^{l_1} \times q^{l_2} \times ... \times q^{l_r}$," *Signal Processing*, vol. 82, pp. 497-502, Mar. 2002.

[68] Y. Zeng, G. Bi, and A. R. Leyman, "New algorithms for multidimensional discrete Hartley transform," *Signal Processing*, vol. 82, pp. 1086-1095, Aug. 2002.

[69] J.R. Phillips, and J.K. White, "A precorrected-FFT method for electrostatic analysis of complicated 3-D structures," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 1059-1072, Oct. 1997.

[70] S. Schaller, T. Flohr, and P. Steffen, "An efficient Fourier method for 3-D radon inversion in exact cone-beam CT reconstruction," *IEEE Trans. Medical Imaging*, vol. 17, pp. 244-250, April 1998.

[71] K. Taylor, and D.P. Taylor, "Analyzing television coding techniques using three-dimensional signal analysis," *IEEE Trans. Circuits Syst. II: Analog and Digital Signal Processing*, vol. 46, pp. 414-427, April 1999.

[72] R.W. Cox, and Tong Raoqiong, "Two- and three-dimensional image rotation using the FFT," *IEEE Trans. Image Processing*, vol. 8, pp. 1297-1299, Sept. 1999.

[73] M.A. Westenberg, and J.B.T.M. Roerdink, "Frequency domain volume rendering by the wavelet X-ray transform," *IEEE Trans. Image Processing*, vol. 9, pp. 1249-1261, July 2000.

[74] D.B. Trizna, "Errors in bathymetric retrievals using linear dispersion in 3-D FFT analysis of marine radar ocean wave imagery," *IEEE Trans. Geosci. Remote Sensing*, vol. 39, pp. 2465-2469, Nov. 2001.

[75] Z.M. Benenson, A.B Elizarov, T.V. Yakovleva, and Jr. W.D. O'Brien, "Approach to 3-D ultrasound high resolution imaging for mechanically moving large-aperture transducer based upon Fourier transform," *IEEE Trans. Ultrason., Ferroelect., Freq. Contr.*, vol. 49, pp. 1665-1685, Dec. 2002.

[76] D. Brasse, P.E. Kinahan, R. Clackdoyle, M. Defrise, C. Comtat, and D.W. Townsend, "Fast fully 3-D image reconstruction in PET using planograms," *IEEE Trans. Medical Imagine*, vol. 23, pp. 413-425, April 2004.

[77] S. Winograd, "On the multiplicative complexity of the discrete Fourier transform," *Adv. Math.*, vol. 32, pp. 83-117, May 1979.

[78] S. Winograd, "Signal processing and complexity of computation," in *Proc., IEEE Int. Conf. Acoust., Speech, Signal Processing*, vol. 5, Apr. 1980, pp. 94-101.

[79] B. Mescheder, "On the number of active *-operations needed to compute the discrete Fourier transform," *Acta Inform.*, vol. 13, pp. 383-408, May 1980.

[80] Z. Li, H.V. Sorensen, and C.S. Burrus, "FFT and convolution algorithms on DSP microprocessors," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, vol. 11, Apr. 1986, pp. 289-292.

[81] C. Lu, "Implementation of 'multiply-add' FFT algorithms for complex and real data sequences," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol.1, June 1991, pp. 480-483.

[82] E.N. Linzer, and E. Feig, "Implementation of efficient FFT algorithms on fused multiply-add architectures," *IEEE Trans. Singal Processing*, vol. 41, pp. 93-107, Jan. 1993.

[83] C. Lu, J.W. Cooley, and R. Tolimieri, "FFT algorithms for prime transform sizes and their implementations on VAX, IBM3090VF, and IBM RS/6000," *IEEE Trans. Signal Processing*, vol. 41, pp. 638-648, Feb. 1993.

[84] D. Takahashi, "A radix-16 FFT algorithm suitable for multiply-add instruction based on Goedecker method," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, vol. 2, April 2003, pp. 665-668.

[85] M.T. Heideman, and C.S. Burrus, "On the number of multiplications necessary to compute a length-2^n DFT," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 34, pp. 91-95, Feb. 1986.

[86] P. Duhamel, "Algorithms meeting the lower bounds on the multiplicative complexity of length-2^n DFTs and their connection with practical algorithms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 1504-1511, Sept. 1990.

[87] B.M. Baas, "*An approach to low-power, high-performance, fast Fourier transform processor Design*," Ph.D. Dissertation, Stanford University, USA, Feb. 1999.

[88] Y. Ma, "An effective memory addressing scheme for FFT processors," *IEEE Trans. Signal Processing*, vol. 47, pp.907-911, Mar. 1999.

[89] Y. Ma and L. Wanhammar, "A hardware efficient control of memory addressing for high-performance FFT processors," *IEEE Trans. Signal Processing*, vol. 48, pp.917-921, Mar. 2000.

[90] C.-L. Wang, and C.-H. Chang, "A DHT-based FFT/IFFT processor for VDSL transceivers," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, vol. 2, May 2001, pp. 1213-1216.

[91] Y. Jiang, T. Zhou, Y. Tang and Y. Wang, "Twiddle-factor-based FFT algorithm with reduced memory access," in *Proc. IEEE Int. Symp. Parallel and Distributed Processing*, April 2002, pp. 70-77.

[92] Q. Li, N. Wang, B. Shi and C. Zheng, "Extendible look-up table of twiddle factors and radix-8 based Fourier transform," *Signal Processing*, vol. 82, pp. 643-648, April 2002.

[93] M. Hasan and T. Arslan, " A coefficient memory addressing scheme for VLSI implementation of FFT processors," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 4, May 2002, pp. 850-853.

[94] Y. Tang, Y. Jiang and Y. Wang, "Reduce FFT memory reference for low power applications," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, vol. 3, May 2002, pp. 3204-3207.

[95] M. Hasan, T. Arslan and J. S. Thompson, "A novel coefficient ordering based low power pipelined radix-4 FFT processor for wireless LAN applications," *IEEE Trans. Consumer Electronics*, vol. 49,, pp. 128-134, Feb. 2003.

[96] Y. Tang, L. Qian, Y. Wang and Y. Savaria, "A new memory reference reduction method for FFT implementation on DSP," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, vol. 4, May 2003, pp. 496-499.

[97] P.K. Meher, and T. Srikanthan, "A scalable and multiplier-less fully-pipelined architecture for VLSI implementation of discrete Hartley transform," in *Proc. IEEE Int. Symp. Signals, Circuits, Systems*, vol. 2, July 2003, pp. 393-396.

[98] Saad Bouguezel, M.N.S. Swamy, and M. Omair Ahmad, "Multidimensional vector radix FHT algorithms," submitted to *IEEE Transactions on Circuits and Systems Part I: Regular Papers.*

[99] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "A relationship between the structures of the radix-2 DIT FHT and complex-valued FFT algorithms," in *Proc. IEEE Canadian Conference on Electrical and Computer Engineering*, Niagara Falls, Ontario, Canada, May 2004, pp. 1111-1114.

[100] J. Granata, M. Conner and R. Tolimieri, "The tensor product: A Mathematical programming language for FFT's and other fast DSP operations," *IEEE Signal Processing Magazine*, pp. 40-48, Jan. 1992.

[101] J. D. Markel, "FFT Pruning," *IEEE Trans. Audio Electroacoust.*, vol. 19, pp. 305-311, Dec. 1971.

[102] D. P. Skinner, "Pruning the decimation-in-time FFT algorithm," *IEEE Tans. Acoust., Speech, Signal Processing*, vol. 24, pp. 193-194, Apr. 1976.

[103] H. V. Sorensen and C. S. Burrus, "Efficient computation of the DFT with only a subset of input/output points," *IEEE Trans. Signal Processing*, vol. 41, pp.1184-1200, Mar. 1993.

[104] R. G. Alves, P. L. Osorio and M. N. S. Swamy, "General FFT pruning algorithm," in *Proc. 43rd Midwest Symposium on Circuits and Systems,* Lansing MI, Aug. 2000, pp. 1192-1195.

[105] S. R. Rangarajan and S. Srinivasan, "Generalised method for pruning an FFT type of transform," *IEE Proc. Vision, Image and Signal Processing*, vol. 144, pp. 189-192, Aug. 1997.

[106] C.D. Murphy, "Low-complexity FFT structures for OFDM transceivers,"*IEEE Trans. Communications,* vol. 50, pp. 1878-1881, Dec. 2002.

[107] L. Capodiferro, "Two-dimensional FFT and FFT-pruned algorithms in the context of HDTV images," in *Signal Processing of HDTV*, L. Chiariglione, Ed. New York: Elsevier, 1988.

[108] R. Cusani and L. Capodiferro, "Pruning the vector-radix 2-D FFT," in *Proc. Int. Conf. on Mini and Microcomputers,* Barcelona, Spain, June 1988, pp. 461-464.

[109] K. S. Knudsen and L. T. Bruton, "Recursive pruning of the 2-D DFT with 3-D signal processing applications," *IEEE Trans. Signal Processing*, vol. 41, pp. 1340-1356, Mar. 1993.

[110] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "Efficient pruning algorithms for the DFT computation for a subset of output samples," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 4, Bangkok, Thailand, May 2003, pp. 97-100.

[111] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "Efficient output-pruning of the 2-D FFT algorithm," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 3, Vancouver, Canada, May 2004, pp. 285-288.

[112] L. Jia, Y. Gao and H. Tenhumen, "Efficient VLSI implementation of radix-8 FFT algorithm," in *Proc. IEEE Pacific Rim Conf., Communications, Computers and Signal Processing*, Aug. 1999, pp. 468-471.

[113] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "Improved radix-4 and radix-8 FFT algorithms," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 3, Vancouver, Canada, May 2004, pp. 561-564.

[114] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "An improved radix-16 FFT algorithm," in *Proc. IEEE Canadian Conference on Electrical and Computer Engineering*, Niagara Falls, Ontario, Canada, May 2004, pp. 1089-1092.

[115] Y. N. Chang and K. K. Parhi, "An efficient pipelined FFT architecture," *IEEE Trans. Circuits Systems II: Analog and Digital Signal Processing*, vol. 50, pp. 322-325, June 2003.

[116] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "An efficient FFT algorithm based on the radix-2/4 DIF approach for computing 3-D DFT," in *Proc. IEEE Canadian Conference on Electrical and Computer Engineering*, Niagara Falls, Ontario, Canada, May 2004, pp. 1131-11134.

[117] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "New radix-$(2\times2\times2)/(4\times4\times4)$ and radix-$(2\times2\times2)/(8\times8\times8)$ DIF FFT algorithms for 3-D DFT," submitted to *IEEE Transactions on Circuits and Systems Part I: Regular Papers.*

[118] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "An efficient three-dimensional decimation-in-time FHT algorithm based on the radix-2/4 approach," submitted to *IEEE Int. Symp. Signal Processing and Information Technology*, Rome, Italy, December 2004.

[119] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "Split vector-radix algorithms for the 3-D discrete Hartley transform," submitted to *IEEE Trans. on Circuits and Systems Part I: Regular Papers*.

[120] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "An efficient split-radix FFT algorithm," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 4, Bangkok, Thailand, May 2003, pp. 65-68.

[121] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "A new radix-2/8 FFT algorithm for length-$q \times 2^m$ DFTs," *IEEE Trans. Circuits and Systems Part I: Regular Papers,* vol. 51, pp. 1723-1732, Sept. 2004.

[122] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "An efficient split-radix FHT algorithm," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 3, Vancouver, Canada, May 2004, pp. 565-568.

[123] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "A new split-radix FHT algorithm for length-$q * 2^m$ DHTs," *IEEE Trans. Circuits and Systems Part I: Regular Papers*, vol. 51, Oct. 2004.

[124] P. Duhamel, "Implementation of 'split-radix' FFT algorithms for complex , real, and real-symmetric data," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 34, pp. 285-295, April. 1986.

[125] Saad Bouguezel, M. Omair Ahmad and M.N.S. Swamy, "A split-radix algorithm for 2-D DFT," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 3, Bangkok, Thailand, May 2003, pp. 698-701.