MULTIPLIERLESS DFT, DCT APPROXIMATIONS FOR MULTI-BEAM RF
APERTURE AND HEVC HD VIDEO APPLICATIONS: DIGITAL SYSTEMS
IMPLEMENTATION

A Thesis
Presented to
The Graduate Faculty of The University of Akron

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

Sunera Kulasekera
May, 2016
The discrete Fourier transform (DFT) and the discrete cosine transform (DCT) are two of the most commonly used algorithms in the field of digital signal processing. Hence significant effort is been made to realize the DFT and the DCT computation utilizing minimum resources in terms of hardware, at acceptable accuracy. Fast Fourier transforms (FFT’s) are commonly used in realizing the DFT as they reduce the complexity of the ideal DFT from $N^2$ to $O(N \log N)$. This thesis explores the use of approximate DFT (a-DFT) and approximate DCT (a-DCT) algorithms which closely approximates the ideal DFT and the ideal DCT. Review of the approximate algorithms, mathematical derivations of the proposed architectures and digital hardware implementations are provided. All designs are realized and tested on-chip using field programmable gate array (FPGA) technology and also synthesized up to place & route level for application specific integrated circuits (ASICs). Obtained accuracy and hardware metrics are presented and compared with the current state-of-the-art. Application specific examples are provided for both a-DFT and a-DCT where when simulated, the a-DFT closely resemble the antenna array patterns of an FFT-based beamformer. The a-DCT algorithm in next generation video codecs are also analysed by integrating it into high efficiency video coding standard (HEVC).
ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to my supervisor, Dr. Arjuna Madanayake, for all his continuous guidance, regular advice and encouragement throughout my course of study. His support and continuous belief in me enabled me to overcome difficult research problems and this journey would not have been possible without him.

I am also grateful to my thesis committee members Dr. Shiva Sastry and Dr. Nghi Tran for their helpful advice and encouragement. I would also like to thank department chair, Dr. Abbas Omar and other faculty members of the electrical and computer engineering department for their support and advice. I would like to thank external collaborators Dr. Renato J. Cintra and Dr. Fabio M. Bayer for their valuable ideas and guidance. My sincere gratitude goes out to my friends, specially Tharindu, Nilan, Viduneth, Nilanka and Chamith of the Advanced Signal Processing Circuits Group at the University of Akron, for their continuous help and support.

Finally I would specially like to thank my dear parents, Kumudinie Perera and D.P. Kulaskera, my wife Lawani Senanayake, my sister Sulari Kulasekera and all my relatives for all they’ve done for me throughout the years and i dedicate this thesis to them as a token of gratitude.
TABLE OF CONTENTS

| LIST OF TABLES                        | viii |
| LIST OF FIGURES                      | x    |

CHAPTER

I. INTRODUCTION                       | 1    |
   1.1 Contributions in this Thesis    | 4    |
   1.2 Publications                    | 6    |
   1.3 Primary Collaborators           | 8    |
   1.4 Thesis Outline                  | 8    |

II. REVIEW OF DFT COMPUTATION: FAST ALGORITHMS & HARDWARE | 11 |
   2.1 Mathematical Definition         | 11   |
   2.2 Properties of the DFT           | 13   |
   2.3 Fast Fourier Transforms (FFTs)  | 14   |

III. REVIEW OF DIGITAL HARDWARE FPGA SYSTEMS | 22 |
   3.1 ROACH-2 Hardware Processing Platform | 23 |
   3.2 BEE3 multi-FPGA based rapid prototyping platform. | 24 |
   3.3 ML605 FPGA evaluation board      | 25   |
IV. INTRODUCTION TO A-DFT ........................................... 30
  4.1 Proposed a-DFT: Mathematical Definition .................... 32
  4.2 Arithmetic Complexity .......................................... 40
  4.3 a-DFT: Digital Architectures and Realizations ............... 40
  4.4 Hardware Resource Consumption Comparison & Discussion .... 47
V. A-DFT APPLICATIONS: INTRODUCTION TO MULTI-BEAM RF
   APERTURES .......................................................... 49
  5.1 Multi-Beamforming using 1-D DFT: Theory ....................... 53
  5.2 Multi-Beam RF Aperture using 1-D a-DFT: Simulations & Results. 54
  5.3 a-DFT Rectangular Aperture ..................................... 57
VI. 8-POINT & 16-POINT LOW-COMPLEXITY MULTIPLIERLESS
     DCT APPROXIMATIONS FOR LOW-POWER HEVC DIGITAL IP
     CORES .................................................................. 64
  6.1 Mathematical Review of DCT ....................................... 66
  6.2 Fast Algorithms for Computing DCT .............................. 66
  6.3 Proposed DCT Approximations: Mathematical Definition ...... 68
  6.4 a-DCT: Digital Architectures & Realizations ................. 73
  6.5 a-DCT for HEVC ..................................................... 81
VII. A MULTIPLIERLESS PRUNED DCT-LIKE TRANSFORMATION
     FOR IMAGE AND VIDEO COMPRESSION .......................... 86
  7.1 Pruned a-DCT: Mathematical Definition ........................ 87
  7.2 Pruned a-DCT: Digital Architectures & Realizations .......... 91
  7.3 HEVC Software Simulation ....................................... 100
### VIII. ERROR-FREE COMPUTATION OF 8-POINT DCT BASED ON THE LOEFFLER FACTORIZATION AND ALGEBRAIC INTEGERS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Algebraic Integer Representation</td>
<td>104</td>
</tr>
<tr>
<td>8.2</td>
<td>AI-based 8-point Loeffler DCT</td>
<td>110</td>
</tr>
<tr>
<td>8.3</td>
<td>Final Reconstruction Step</td>
<td>115</td>
</tr>
<tr>
<td>8.4</td>
<td>AI Based Loeffler DCT: Digital Architectures &amp; Realizations</td>
<td>119</td>
</tr>
</tbody>
</table>

### IX. CONCLUSIONS & FUTURE WORK

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Future Work</td>
<td>125</td>
</tr>
</tbody>
</table>

### BIBLIOGRAPHY

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>129</td>
</tr>
</tbody>
</table>

### APPENDIX

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>ROACH-2 FPGA Specifications.</td>
</tr>
<tr>
<td>3.2</td>
<td>BEE3 FPGA Specifications.</td>
</tr>
<tr>
<td>3.3</td>
<td>ML605 FPGA Specifications.</td>
</tr>
<tr>
<td>3.4</td>
<td>Comparison of digital FPGA prototyping boards.</td>
</tr>
<tr>
<td>4.1</td>
<td>Arithmetic Complexity of each FFT algorithm compared with the proposed a-DFT.</td>
</tr>
<tr>
<td>4.2</td>
<td>Hardware resource consumption using Xilinx Virtex-6 XC6VLX-240T 1FFG1156 device and power consumption using a Xilinx Virtex-6 XC6VLX760-1FF1760 device.</td>
</tr>
<tr>
<td>4.3</td>
<td>Hardware resource consumption using CMOS 45nm ASIC synthesis.</td>
</tr>
<tr>
<td>4.4</td>
<td>Hardware resource consumption using Xilinx Virtex-6 XC6VSX-475T 1FF1759 on ROACH-2 device.</td>
</tr>
<tr>
<td>4.5</td>
<td>Hardware resource consumption for CMOS 180nm ASIC place &amp; route.</td>
</tr>
<tr>
<td>6.1</td>
<td>Hardware resource consumption using Xilinx Virtex-6 XC6VLX-240T-1FFG1156 device.</td>
</tr>
<tr>
<td>6.2</td>
<td>Hardware resource consumption for CMOS 45nm ASIC place-route realization.</td>
</tr>
<tr>
<td>6.3</td>
<td>Hardware resource consumption comparison of the 2-D 8-point approximations using Xilinx Virtex-6 XC6VLX240T-1FFG1156 device.</td>
</tr>
<tr>
<td>6.4</td>
<td>Hardware resource consumption comparison for CMOS 45 nm ASIC synthesis realization of the 2-D 8-point approximations.</td>
</tr>
</tbody>
</table>
6.5 Hardware resource consumption comparison of the 1-D 16-point approximations using Xilinx Virtex-6 XC6VLX240T-1FFG1156 device. . . 80
6.6 Hardware resource consumption comparison for CMOS 45nm ASIC synthesis realization of the 1-D 16-point approximations. . . . . . . . . 80
7.1 Resource consumption on Xilinx XC6VLX240T-1FFG1156 device. . . 93
7.2 Resource consumption for CMOS 45nm ASIC synthesis. . . . . . . . . 94
7.3 Percentage reduction in area and dynamic power for FPGA. . . . . . 95
7.4 Hardware resource consumption using Xilinx Virtex-5 XC5VSX-95T-2FF1136 device. . . . . . . . . . . . . . . . . . . . . . . . . . . . 99
8.1 Encoding of AI basis elements. . . . . . . . . . . . . . . . . . . . . . 107
8.2 Quantities required by Loeffler algorithm for 8-point DCT and their respective products by an arbitrary algebraic integer. . . . . . . . 109
8.3 Multiplicative complexity comparison for 8-point DCT computation. . 115
8.4 12-bit CSD encoding of the AI bases elements. . . . . . . . . . . . . 117
8.5 CSD encoding error. . . . . . . . . . . . . . . . . . . . . . . . . . . . 118
8.6 Scale factors and maximum/minimum relatives error. . . . . . . . . . 119
8.7 Hardware resource consumption using Xilinx Virtex-6 XC6VSX-475T 1FF1759 device. . . . . . . . . . . . . . . . . . . . . . . . . . . . 122
8.8 Hardware resource consumption for CMOS 180nm ASIC synthesis. . . 123


<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Signal flow graph of the radix-2 8-point Cooley-Tukey algorithm. page 18</td>
</tr>
<tr>
<td>2.2</td>
<td>Signal flow graph of the radix-2 16-point Cooley-Tukey algorithm. page 19</td>
</tr>
<tr>
<td>3.1</td>
<td>ROACH-2 hardware processing platform board. page 23</td>
</tr>
<tr>
<td>3.2</td>
<td>ROACH-2 hardware processing platform with the ADCs connected. page 25</td>
</tr>
<tr>
<td>3.3</td>
<td>BEE3 device used to implement digital architectures. Labels A, B, C, and D indicate the four Xilinx Virtex-5 XC5VSX95T-2FF1136 FPGA devices. page 26</td>
</tr>
<tr>
<td>3.4</td>
<td>ML605 device used to implement digital architectures. page 27</td>
</tr>
<tr>
<td>4.1</td>
<td>Signal flow graph for the factorization of ( \hat{F}_8 ). Input data ( v_i, i = 0, 1, \ldots, 7 ), relates to the output ( V_k, k = 0, 1, \ldots, 7 ). Dotted arrows represent multiplications by (-1). page 36</td>
</tr>
<tr>
<td>4.2</td>
<td>Signal flow graph for the factorization of ( \hat{F}_{16} ). Input data ( v_i, i = 0, 1, \ldots, 15 ), relates to the output ( V_k, k = 0, 1, \ldots, 15 ). page 39</td>
</tr>
<tr>
<td>4.3</td>
<td>1-D architecture of the proposed 8-point DFT Approximation. page 42</td>
</tr>
<tr>
<td>4.4</td>
<td>1-D architecture of the proposed 16-point DFT Approximation. page 43</td>
</tr>
<tr>
<td>4.5</td>
<td>Two-dimensional approximate algorithm for 8-point and 16-point DFT by means of 1-D approximate transform. Signal ( v_{j,0}, v_{j,1}, \ldots ) corresponds to the rows of the input; ( V_{j,0}, V_{j,1}, \ldots ) indicates the transformed rows; ( V_{0,k}, V_{1,k}, \ldots ) indicates the columns of the transposed row-wise transformed image; and ( V_{0,k}^{(2-D)}, V_{1,k}^{(2-D)}, \ldots ) indicates the columns of the final 2-D transformed input. page 45</td>
</tr>
</tbody>
</table>
5.1 Application of N-point spatial DFT for N-beam radar sensor. ....... 51
5.2 Example of high-bandwidth connections over relatively short distances. 52
5.3 Simplified ULA-based multi-beamformer using a spatial FFT. ......... 53
5.4 Real output of each bin of the a-DFT. ........................................... 56
5.5 Imaginary output of each bin of the a-DFT. ................................. 56
5.6 Normalized power with respect to time. ....................................... 57
5.7 Multi-beams using 8-point a-DFT. Array with omni-antennas, \( \Delta x = \lambda/2 \) using 8-point exact DFT (left), proposed a-DFT (middle) and deviation/error (right) .............................. 57
5.8 Multi-beams using 16-point a-DFT. Top: Array with omni-antennas, \( \Delta x = \lambda/2 \) using 16-point exact DFT (left), proposed a-DFT (middle) and deviation/error (right). Bottom: Sparse multi-beams for elements with \( \cos^3 \psi \) pattern, with \( \Delta = \lambda \) using 16-point exact DFT (left), proposed a-DFT (middle) and deviation/error (right). ............. 58
5.9 ULA based multi-beam RF aperture application using a spatial DFT. .. 59
5.10 Signal flow graph implementation of a real-time 64 beam RF aperture. 62
5.11 Normalized array beams for the 8-beams. ................................. 63
5.12 Array pattern from the Ideal DFT, array pattern from the a-DFT and the absolute difference. ................................................. 63
6.1 Signal flow graph for the factorization of \( \mathbf{T}^* \). Input data \( x_i, i = 0,1,...,7 \), relates to the output \( X_k, k = 0,1,...,7 \). Dotted arrows represent multiplications by \(-1\). .............................................. 71
6.2 Signal flow graph for the factorization of \( \mathbf{T} \). Input data \( x_i, i = 0,1,...,15 \), relates to the output \( X_k, k = 0,1,...,15 \). Dotted arrows represent multiplications by \(-1\). ............................................ 74
6.3 Architecture of the proposed 8-point DCT approximation. ............. 75
6.4 Architecture of the proposed 16-point DCT approximation. ............ 76
6.5 Two-dimensional approximate transform for 8-point and 16-point by means of 1-D approximate transform. Signal $x_{j,0}, x_{j,1}, \ldots$ corresponds to the rows of the input image; $X_{j,0}, X_{j,1}, \ldots$ indicates the transformed rows; $X_{0,k}, X_{1,k}, \ldots$ indicates the columns of the transposed row-wise transformed image; and $X_{0,k}^{(2-D)}, X_{1,k}^{(2-D)}, \ldots$ indicates the columns of the final 2-D transformed image.

6.6 RD curves for ‘BasketballPass’ test sequence.

6.7 RD curves for ‘BasketballPass’ test sequence.

6.8 Selected frames from ‘BasketballPass’ test video coded by means of the Chen DCT and the 16-point and 8-point DCT approximations for QP = 0 (a-b-c), QP = 32 (d-e-f), and QP = 50 (g-h-i).

7.1 Signal flow graph for the a-DCT matrix and pruned a-DCT matrices [1,2].

7.2 Fast algorithm for pruned LODCT.

7.3 1-D architecture of the proposed 8-point pruned LODCT.

7.4 1-D architecture of the proposed 8-point pruned a-DCT.

7.5 Selected frame from ‘BasketballPass’ test video coded by means of (a) the Chen’s DCT algorithm (PSNR 37.62 dB), (b) the modified RDCT (PSNR 37.42 dB), and (c) the proposed pruned a-DCT ($k = 4$), with 76.2% less arithmetic operations than Chen’s DCT (PSNR 37.41 dB).

7.6 RD curves for ‘BasketballPass’ test sequence.

7.7 PSNR values related to QP.

8.1 Loeffler algorithm for the 8-point DCT computation, where dashed lines represent product by $-1$.

8.2 Multiplication rules required by the Loeffler DCT over AI: (a) rotation block computation and (b) multiplication by $\sqrt{2}$.

8.3 The 8-point DCT algorithm for a real quantized input sequence.

8.4 Digital architecture of the Loeffler algorithm for the 8-point DCT computation.
8.5 Digital implementation of several outputs using the FRS. (a) Computation of the final outputs $X_0, X_1, X_2$; (b) computation of $c_0, c_1, \ldots, c_7$ using the CSD method as shown in Table 8.4, and (c) computation of $c_0, c_1, \ldots, c_7$ using the EF method. 121

9.1 RF-front end for receiver for one element in the array. I-Q fed to real and imaginary DFT inputs. 126

9.2 Prototype FPGA realization of 16-point a-DFT and 16-channel ADC on ROACH-2. 127
CHAPTER I

INTRODUCTION

Digital very-large-scale integration (VLSI) sector has been subjected to explosive growth in the past decade or so as it’s fueled by new innovations in physics at the device level with the advancements in technology. Parallel to technology developments, systems performance is rapidly improving at the algorithms level by sophisticate designs which increasingly take into non-conventional mathematical techniques and optimization schemes that are specifically tailored for high-performance digital VLSI devices. In particular, the application of number theoretic and approximation techniques for the design of fast algorithms having lower hardware complexity, lower power consumption, and/or higher accuracy is of critical interest [3,4].

The DFT and the DCT are used in a number of emerging digital signal processing (DSP) applications, such as, HD video compression, biomedical imaging, and smart antenna beamformers for wireless communications and radar. The FFT stands out as the pivotal algorithm to calculate the DFT more conveniently which in turn will save huge chunks of hardware resources.
Of late, there has been much interest on fast algorithms for the computation of the above transforms using multiplier-free approximations because they result in low power and low complexity systems. Approximate methods rely on the trade-off of accuracy for lower power and/or circuit complexity/chip-area.

FFT [5] is considered the most important algorithm of our generation; it is essential to a huge number of applications, such as wireless communications [6, 7], networking [8], sensor networks [9,10], cognitive radio [11–13], radar [14], imaging [15, 16], filtering [17], correlation [18], and radio-astronomy [19]. N-point FFTs are fast algorithms that efficiently compute an N-point DFT. The DFT is an $N \times N$ linear transform (LT) matrix that splits $N$-samples of a signal to its frequency components.

The DCT is a pivotal tool in almost all real-time video compression systems. In particular, the DCT is a major tool in image processing. This is mainly because the DCT is asymptotically equivalent to the Karhunen-Loève transform (KLT) [20, 21], which possesses optimal decorrelation and energy compaction properties [20–25].

The DFT and DCT are two important LT operations in digital signal processing (DSP) [26,27]. The DCT has applications in video processing, such as H.265/HEVC, MIT SoftCast [28–30] and four-dimensional (4-D) light-field compression [31–37].

The hardware complexity of FFTs/DCTs play a critical role where the size, weight and power (SWaP) must be minimized. Traditional fast algorithms do not take into account the circuit-level trade-offs (power, chip area, quantization) during algorithm design. Typically, fast algorithms aim to minimize floating point operation (FLOP) counts. This approach is sub-optimal for hardware algorithms aimed at application-specific computing architectures.
What if we can come up with algorithms which are faster than the current fast algorithms? It clearly will have a major impact in the field [38–40]. But if they need to work on all possible input signals and to perfectly realize an exact DFT or an exact DCT, then we strongly suspect that we are unlikely to beat the current fast algorithm implementations because their lower bounds are well known [41]. However, if we now allow a vanishingly small (and bounded) computational error in computing these algorithms, under such a scenario, all known lower bounds of such fast algorithms that aim for the exact $DFT/DCT$ are no longer relevant! This is the key point of this research. We propose to trade a small amount of computational error to achieve a massive reduction in hardware complexity and power consumption.

This thesis is aimed at exploring approximation of DFT and DCT transforms, such that substantial savings in hardware complexity can be achieved by trading-off accuracy. We propose provably-accurate multiplierless a-DFT and a-DCT digital hardware implementations that offer almost DFT/DCT like performance at significantly smaller hardware complexity and power consumption. The circuit complexity and power consumption of multipliers exceed that of adders [42]. Replacing a given LT with an approximation that does not require multiplication is a tremendous improvement in terms of SWaP because multiplication is a very expensive operation in digital arithmetic.

In this thesis, approximation algorithms for the DFT and the DCT are investigated, both of which are the backbone of communications [6,43,44], beamforming [45,46], video processing [47,48]. Novel digital architectures are introduced for the proposed approximate algorithms and are realized in FPGA and ASIC along with
popular fast DFT/DCT algorithms available in signal processing literature and the amount of SWaP reductions that can be achieved using the proposed approximation algorithms are shown. Furthermore these novel approximation algorithms are simulated in application specific examples (multi-beam radio frequency (RF) aperture applications in the case of the DFT & HEVC high definition (HD) video encoding applications in the case of DCT) to show that the results we obtain are almost identical to the ones we get using the above mentioned fast algorithms.

1.1 Contributions in this Thesis

1. Digital implementation of a novel 1-D DFT approximation algorithm is proposed. This approximation is realized in both FPGA & ASIC along with the well known FFT’s in literature and the results are presented. Furthermore, multi-beam RF aperture application simulation and the resulting beam patterns are provided where the proposed algorithm was used instead of the traditional FFT’s. This work is published in [49, 50]

2. A novel 2-D DFT approximation algorithm is also implemented in digital hardware and steps are also provided in creating the 2-D algorithm using the 1-D DFT approximation. Digital implementation details are also presented along with the FPGA & ASIC results for the real-time row parallel 2-D digital implementation of the algorithm using 16 blocks of the 1-D digital architecture for the 8-point a-DFT. Resulting 3-D beams from the simulation for multi-beam rect-
angular apertures are also provided where beam patterns show almost identical results and minimum error when compared with the ideal DFT [51].

3. Digital architecture for both 1-D and 2-D 16-point DCT approximation algorithms are also proposed. SWaP values were taken from both FPGA & ASIC realizations of the algorithm and compared with one of the most popular DCT approximation algorithms in literature, the BAS-2010 algorithm [52]. Furthermore, both 8-point and 16-point versions of the proposed algorithm were implemented in a real time HEVC video encoder software [53] and corresponding PSNR values and bit-rate values were compared with the software’s inbuilt algorithm, the Chen’s Algorithm [54] using rate-distortion (RD) curves [55–58].

4. Digital architectures for an 8-point algorithm is proposed based on pruning state-of-the-art DCT approximations. Digital architectures for different pruning stages are provided. The algorithm is realized in both FPGA & ASIC along with another DCT approximation algorithm in the literature, the BAS-2013 [59]. Compared to the mentioned algorithm, the proposed show vast improvements in both chip area and power consumption. The digital architecture of one such pruning stage ($K = 6$) is realized on the Berkerly Emulation Engine (BEE3) along with the LODCT [60] and hardware resource consumption values are compared. Another pruning stage ($K = 4$) is implemented the HEVC reference software [53] and the RD curves are compared with the Chen DCT. This work is published in [1, 2].
5. Finally an algebraic integer (AI) based 1-D DCT architecture using a novel expansion factor based final reconstruction step (FRS) as well as a canonical signed digit based FRS is proposed and its digital implementation is presented. The algorithm is compared with the Shen’s algorithm [61] using FPGA & ASIC realizations.

1.2 Publications

A list of publications resulting from this thesis are as follows:


9. V. Coutinho, R. Cintra, F. Bayer, S. Kulasekera, and A. Madanayake, ”Low-complexity pruned 8-point DCT approximations for image encoding,” in Inter-
1.3 Primary Collaborators

1. Dr. R.J. Cintra from Universidade Federal de Pernambuco, Brazil and Dr. F.M. Bayer from Universidade Federal de Santa Maria, Brazil, proposed the a-DFT algorithm for both 8-point and 16-point versions [49]. Furthermore they extended their idea on the 8-point a-DCT which was proposed in [62] to propose a novel 16-point a-DCT algorithm [56]. Their continues research on a-DCT resulted in proposing the pruning algorithm for the 8-point a-DCT [1, 2]. The FRS methods using the canonical signed digits and expansion factor methods for the Loeffler AI based algorithm was also proposed by Dr. Cintra and Dr. Bayer.

1.4 Thesis Outline

The remainder of the thesis is organized as follows. Chapter 2 reviews the DFT and its mathematical definition along with the properties of the DFT. Introduction to FFTs are presented with two of the most popular FFT algorithms in the literature, The radix-2 Cooley-Tukey FFT and the Winograd FFT. Mathematical definition of both algorithms are provided with the matrix versions and the signal flow graphs for the 8-point and 16-point versions.
Chapter 3 reviews the hardware platforms that are being used to realize the digital designs. Three main FPGA prototyping boards have been used throughout this research and all three are introduced in this chapter along with their FPGA specifications and comparison of each board with another.

Chapter 4 introduces the proposed DFT approximation algorithm and its mathematical framework and the matrix form of the 8-point and 16-point versions. The signal flow graphs of the above two versions are presented with computational complexities are compared with the FFT algorithms discussed in Chapter 2. Digital architectures are presented while FPGA & ASIC results are shown and compared with the FFT algorithms and the results are discussed.

Chapter 5 provides application specific examples where the proposed algorithm is used in realizing multi-beam RF apertures for uniformly linear array (ULA) as well as rectangular aperture arrays. Multi-beamformer theory is discussed for both 1-D and 2-D cases. Simulation information is provided along with resulting beam patterns for both 8-point and 16-point cases and the beam patterns are compared with the ideal-DFT patterns and their error characteristics are also discussed.

Chapter 6 introduces an a-DCT algorithm along with its mathematical definitions and corresponding matrices for both 8-point and 16-point versions. The digital architectures are presented with FPGA & ASIC realization results for the 16-point version. These hardware realizations are compared with the BAS-2010 algorithm and the SWaP matrices are discussed. Finally both 8-point and 16-point versions are implemented in HEVC reference software and the corresponding RD curves are obtained.
Chapter 7 begins by introducing a pruned DCT algorithm which is an extension of the 8-point version discussed in Chapter 6. The signal flow graphs for all stages of the algorithm is provided and the digital architectures are presented and compared with the BAS-2013 algorithm [59] in both FPGA & ASIC. The stage where \( K = 6 \) is compared with the LODCT algorithm using the hardware resource consumption numbers taken from the BEE3 implementation. The chapter concludes with the implementation details of the \( K = 4 \) stage on HEVC with corresponding RD Curves.

In Chapter 8, the AI based Loeffler algorithms discussed starting with mathematical framework for both FRS methods, canonical signed digit method [63] and the expansion factor [63] method. Then digital architectures are presented and the algorithm is also compared to the Shen’s algorithm [54] by realizing both in FPGA & ASIC.

Finally, Chapter 9 concludes the thesis with a description of possible future work and architecture and application level research on DFT/DCT approximation.
CHAPTER II
REVIEW OF DFT COMPUTATION: FAST ALGORITHMS & HARDWARE

The DFT is an $N \times N$ linear transform (LT) matrix that splits $N$-samples of a signal to its frequency components. Its the most basic building block in many highly-deployed digital systems. The DFT is fundamental to the computation of correlation [18], equalization [64], channelization [65], fractional delay [66], convolution [67, 68], filtering [69], deconvolution [70], multiplication of large integers [71], spectral estimation [72], energy detection [73], and feature-detection [74]. Hence efficient realization of the DFT is of immense importance.

2.1 Mathematical Definition

The sequence of $N$ complex numbers $x_0, x_1, \ldots, x_{N-1}$ is transformed into an $N$-periodic sequence of complex numbers:

$$X_k \overset{\text{def}}{=} \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}, \quad k \in \mathbb{Z}.$$
Each $X_k$ is a complex number that encodes both amplitude and phase of a sinusoidal component of function $x_n$. The sinusoid’s frequency is $k$ cycles per $N$ samples. Its amplitude and phase are:

$$|X_k|/N = \sqrt{\text{Re}(X_k)^2 + \text{Im}(X_k)^2}/N$$

$$\text{arg}(X_k) = \text{atan2}(\text{Im}(X_k), \text{Re}(X_k)) = -i \ln \left( \frac{X_k}{|X_k|} \right),$$

where $\text{atan2}$ is the two-argument form of the $\text{arctan}$ function. Assuming periodicity, the customary domain of $k$ actually computed is $[0, N - 1]$. That is always the case when the DFT is implemented via the FFT algorithm. But other common domains are $[-N/2, N/2 - 1]$ ($N$ even) and $[-(N - 1)/2, (N - 1)/2]$ ($N$ odd), as when the left and right halves of an FFT output sequence are swapped.

The DFT can also be expressed as a Vandermonde matrix [75].

$$F = \begin{bmatrix}
\omega_N^0 & \omega_N^1 & \ldots & \omega_N^{N-1} \\
\omega_N^1 & \omega_N^2 & \ldots & \omega_N^{N} \\
\vdots & \vdots & \ddots & \vdots \\
\omega_N^{N-1} & \omega_N^{N-2} & \ldots & \omega_N^{2(N-1)} \\
\end{bmatrix}$$

where $\omega_N = e^{-2\pi i/N}$ is a primitive $N^{th}$ root of unity.

The 8-point DFT which has been used during this research can be represented in the matrix form as,
\[ W = \frac{1}{\sqrt{8}} \begin{bmatrix}
    \omega^0 & \omega^0 & \omega^0 & \ldots & \omega^0 \\
    \omega^0 & \omega^1 & \omega^2 & \ldots & \omega^7 \\
    \omega^0 & \omega^2 & \omega^4 & \ldots & \omega^{14} \\
    \omega^0 & \omega^3 & \omega^6 & \ldots & \omega^{21} \\
    \omega^0 & \omega^4 & \omega^8 & \ldots & \omega^{28} \\
    \omega^0 & \omega^5 & \omega^{10} & \ldots & \omega^{35} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    \omega^0 & \omega^7 & \omega^{14} & \ldots & \omega^{49}
\end{bmatrix}, \text{where } \omega = e^{-\frac{2\pi i}{8}}. \]

The DFT is (or can be, through appropriate selection of scaling) a unitary transform, i.e., one that preserves energy. The appropriate choice of scaling to achieve unitarity is \( 1/\sqrt{N} \), so that the energy in the physical domain will be the same as the energy in the Fourier domain.

### 2.2 Properties of the DFT

In this section, main properties of the DFT such as completeness, orthogonality and periodicity are introduced.

#### 2.2.1 Completeness

The DFT is an invertible linear transformation denoted as \( F : \mathbb{C}^N \to \mathbb{C}^N \) with \( \mathbb{C} \) denoting the set of complex numbers. In other words, for any \( "N" > 0 \), an \( N \)-dimensional complex vector has a DFT and an IDFT which are in turn \( N \)-dimensional complex vectors.
2.2.2 Orthogonality

The vectors $u_k = \left[ e^{\frac{2\pi i}{N} kn} | n = 0, 1, \ldots, N - 1 \right]^T$ form an orthogonal basis over the set of $N$-dimensional complex vectors:

$$u_k^T u_{k'} = \sum_{n=0}^{N-1} \left( e^{\frac{2\pi i}{N} kn} \right) \left( e^{\frac{2\pi i}{N} (-k')n} \right) = \sum_{n=0}^{N-1} e^{\frac{2\pi i}{N} (k-k')n} = N \delta_{kk'}$$

where $\delta_{kk'}$ is the Kronecker delta [76]. This orthogonality condition can be used to derive the formula for the IDFT from the definition of the DFT.

2.2.3 Periodicity

The periodicity can be shown directly from the definition:

$$X_{k+N} \overset{\text{def}}{=} \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} (k+N)n} = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} e^{-2\pi i n} = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} = X_k.$$ 

Similarly, it can be shown that the IDFT formula leads to a periodic extension.

2.3 Fast Fourier Transforms (FFTs)

FFT refers to fast algorithms that efficiently compute the DFT, which in turn, is a LT that splits $N$-samples of a signal to its frequency-domain representation. The computational complexity of FFTs depend on the underlying signal flow graph (SFG). The computational (time) complexity of an $N$–point FFT is $O(N \log N)$, which is dominated by multiplications. Direct computation of a DFT requires $N^2$ complex multiplications and $N^2 N$ complex additions [77]. An $N$ point FFT factorizes the DFT [38, 78] matrix into a product of sparse factors. The FFT is the most important algorithm of our lifetime. Its efficient implementation at low-complexity and low
power consumption is a matter of fundamental importance. The FFT complexity is $O(N \log N)$. For large $N$, the improvement in speed by using FFTs is significant, is proportional to $N/\log N$, and is typically orders of magnitude. The radix-2 Cooley-Tukey FFT algorithm [5], when $N = 2^k, k \in \mathbb{Z}$ gives the DFT output using $\frac{3N}{2} \log N$ real multiplications $2N \log N$ real additions. The search for FFTs for the exact computation of the DFT is well-established. There are many FFT algorithms, each having different complexities and trade-offs. FFT algorithms by Winograd [79], which factorizes an $N^{th}$ order complex polynomial $z^N - 1$ into cyclotomic polynomials (a unique irreducible polynomial with integer coefficients) [79] that have coefficients limited to 1, 0, or $-1$.

2.3.1 The Cooley-Tukey FFT Algorithm

The Cooley-Tukey algorithm, named after J.W. Cooley and J. Tukey, is the most common FFT algorithm. It re-expresses the DFT of an arbitrary composite size $N = N_1N_2$ in terms of smaller DFTs of sizes $N_1$ and $N_2$, recursively, in order to reduce the computation time to $O(N \log N)$ for highly composite $N$. The algorithm, along with its recursive application, was invented by Carl Friedrich Gauss. Cooley and Tukey independently rediscovered and popularized it 160 years later.

Radix-2 Cooley-Tukey Algorithm

The publication by Cooley and Tukey [5] in 1965 of an efficient algorithm for the calculation of the DFT was a major turning point in the development of digital signal processing. During the five or so years that followed, various extensions and modifications were made to the original algorithm [5]. By the early 1970’s the practical
programs were basically in the form used today. Radix-2 divides a DFT of size N into two interleaved DFTs (hence the name “radix-2”) of size N/2 with each recursive stage. It first computes the DFTs of the even-indexed inputs \(x_{2m} = x_0, x_2, \ldots, x_{N-2}\) and of the odd-indexed inputs \(x_{2m+1} = x_1, x_3, \ldots, x_{N-1}\), and then combines those two results to produce the DFT of the whole sequence. This idea can then be performed recursively to reduce the overall runtime to \(O(N \log N)\). This simplified form assumes that \(N\) is a power of two hence the number of sample points \(N\) can usually be chosen freely by the application, this is often not an important restriction.

The radix-2 algorithm rearranges the DFT of the function \(x_n\) into two parts: a sum over the even-numbered indices \(n = 2m\) and a sum over the odd-numbered indices \(n = 2m + 1\):

\[
X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N} (2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N} (2m+1)k}
\]

If we factor a common multiplier \(e^{-\frac{2\pi i k}{N}}\) out of the second sum, as shown in the equation below. It is then clear that the two sums are the DFT of the even-indexed part \(x_{2m}\) and the DFT of odd-indexed part \(x_{2m+1}\) of the function \(x_n\). If we denote the DFT of the even-indexed inputs \(x_{2m}\) by \(E_k\) and the DFT of the odd-indexed inputs \(x_{2m+1}\) by \(O_k\) and we obtain:

\[
X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2} mk} + e^{-\frac{2\pi i k}{N}} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2} mk} = E_k + e^{-\frac{2\pi i k}{N}} O_k.
\]

Using periodicity property of the DFT, we know that
\[ E_{k + \frac{N}{2}} = E_k \quad \text{and} \quad O_{k + \frac{N}{2}} = O_k. \] Therefore, we can rewrite the above equation as

\[
X_k = \begin{cases} 
E_k + e^{-\frac{2\pi i}{N}k}O_k & \text{for } 0 \leq k < \frac{N}{2} \\
E_{k-N/2} + e^{-\frac{2\pi i}{N}k}O_{k-N/2} & \text{for } \frac{N}{2} \leq k < N.
\end{cases}
\]

The twiddle factor \( e^{-2\pi ik/N} \) obeys the following relation:

\[
e^{-\frac{2\pi i}{N}(k+N/2)} = e^{-\frac{2\pi i}{N}k}e^{-\pi i} = e^{-\pi i}e^{-\frac{2\pi i}{N}k} = -e^{-\frac{2\pi i}{N}k}
\]

This allows us to cut the number of “twiddle factor” calculations in half, for \( 0 \leq k < \frac{N}{2} \), we have

\[
X_k = E_k + e^{-\frac{2\pi i}{N}k}O_k \\
X_{k+N/2} = E_k - e^{-\frac{2\pi i}{N}k}O_k
\]

The above result, expressing the DFT of length \( N \) recursively in terms of two DFTs of size \( N/2 \), is the core of the radix-2 FFT. The algorithm gains its speed by re-using the results of intermediate computations to compute multiple DFT outputs. The final outputs are obtained by a \( \pm \) combination of \( E_k \) and \( O_k \exp(-2\pi ik/N) \). The signal flow graph for radix-2 8-point Cooley-Tukey FFT is shown in Fig. 2.1 and the radix-2 16-point Cooley-Tukey is shown in Fig. 2.2.
2.3.2 The Winograd Algorithm

The Winograd DFT algorithm is another popular method of calculating the DFT with lower circuit complexity. Its advantage over the FFT algorithms lies in its significant reduction in the number of multiplications, particularly for higher $N$. This is obtained, however, at the cost of a more complex structure. Most of the FFT algorithms that are available in the literature can easily calculate the DFT for powers of 2. The Winograd algorithm constructs the DFT of any permissible order $N$ through the sequential application of special generalized DFT algorithm for the following 8 orders, 2, 3, 4, 5, 6, 7, 8, 9, 16.
Using an operator notation where $F_1$ represents taking row DFT’s and $F_2$ represents column DFT’s, the two-factor Prime Factor Algorithm (PFA) is represented by [79]

$$X = F_2 F_1 x$$

This will be as same as

$$X = F_1 F_2 x$$

because each operator represents identical operations on each row or column, they commute [79]. In Winograd’s algorithm each $F$ is represented by three operators and
$F$ can be factorized as two-factor Prime Factor Algorithm (PFA) is represented by

$$F = A^T D A$$

where $A$ represents the set of additions done on each row or column that performs the residue reduction [77]. It is the first operator on $x$ and it is called a preweave operator. $D$ is the set of $M$ multiplications and $A^T$ is the reconstruction operator called the postweave. So the equation becomes

$$X = A_2^T D_2 A_2 A_1^T D_1 A_1 x$$

Because these operators commute, it can be written as

$$X = A_2^T A_1^T D_2 D_1 A_2 A_1 x$$

but the two adjacent multiplication operators can be premultiplied and the result can be represented by one operator $D = D_2 D_1$ which is no longer the same for each row or column and the equation becomes

$$X = A_2^T A_1^T D A_2 A_1 x$$

This is the basic idea of the Winograd DFT. The commuting of the multiplication operators together in the center of the algorithm is called nesting and it results in a significant decrease in the number of multiplications that must be done at the execution of the algorithm.

2.3.3 The 8-point and 16-point Winograd Algorithms

The Winograd small FFT algorithms are in the form of

$$V = C B A v$$
Where $v$ and $V$ are the inputs and the outputs and $A$ and $C$ are $8 \times 8$ matrices while $B$ is a $8 \times 8$ diagonal matrix [77]. Hence the 8-point algorithm can be represented in the matrix form where

$$
A_8 = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\
1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 1 & 0 & 1 & 0 & -1 & 0 & -1
\end{bmatrix}
$$

and

$$
C_8 = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & -j & -j & 0 \\
0 & 0 & 1 & 0 & 0 & -j & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & 0 & j & -j \\
0 & 0 & 0 & 1 & j & -j & 0 & 0 \\
0 & 0 & 1 & 0 & j & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

and

$$
B_8 = \text{diag}(1, 1, 1, 1, \cos \theta, 1, 1, \sin \theta)
$$

The 16-point version of the Winograd algorithm can be represented in the matrix form where [77]

$$
A_{16} = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\
1 & 0 & 0 & -1 & 0 & 0 & 0 & -1 \\
1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\
0 & 1 & 0 & 1 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
$$

and

$$
C_{16} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

and

$$
B_{16} = \text{diag}(1, 1, 1, 1, \cos 2\theta, \cos 2\theta, \cos 3\theta, \cos \theta + \cos 3\theta, -\cos \theta + \cos 3\theta, 1, 1, 1, - \sin 2\theta, - \sin 2\theta, - \sin 3\theta, - \sin \theta + \sin 3\theta, - \sin \theta - \sin 3\theta)
$$
CHAPTER III

REVIEW OF DIGITAL HARDWARE FPGA SYSTEMS

A FPGA is an integrated circuit designed to be programmed in the field after manufacture. FPGAs are similar in principle to, but have vastly wider potential application than, programmable read-only memory (PROM) chips. The FPGA configuration is specified using a hardware description language (HDL), similar to that used in ASIC. A FPGA contains an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together", like many logic gates that can be inter-wired in different configurations. Logic blocks can be configured to perform complex functions, or merely simple logic gates like AND and XOR. In most FPGAs, logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory. FPGAs are used by engineers in the design of ICs that can later be produced hard-wired in large quantities for distribution to computer manufacturers and end users. Ultimately, FPGAs might allow computer users to tailor microprocessors to meet their own individual needs. During this study three main FPGA boards have been used to realize digital designs corresponding to the DFT and DCT. The board that was mainly used to realize designs related to DFT was a Reconfigurable Open Architecture Computing Hardware-2 (ROACH-2) board. The designs related to a-DCT were realized in a ML605 board and a BEE3 multi-FPGA based rapid prototyping system.
3.1 ROACH-2 Hardware Processing Platform

The ROACH-2 [80, 81] is a system that was developed with major universities and research labs throughout the world. The ROACH-2 board is widely used in radio astronomy to process signals over a wide frequency range. It is the implementation platform of choice for many modern radio telescopes [82–85] that require extensive digital signal processing for FX-correlations, multi-beamforming and transient event detection. ROACH-2 [80, 81] is an open source FPGA platform designed in Cape Town, South Africa, together with CASPER collaborators as part of the Square Kilometer Array (SKA) radio telescope project [86, 87]. The core processing module
of ROACH-2 is an Xilinx Virtex-6 sx475t FPGA (XC6VSX475T-1FFG1759C) chip, equipped with on board inter-FPGA links interfaced with 10Gb Ethernet interfaces using SFP+ mezzanine cards for all cross-FPGA communications. Key peripherals of a ROACH-2 consists of two multi-gigabit transceiver card slots supporting 4x10Ge links (SFP+), four 72Mb QDR II+ SRAMs and a 72-bit DDR3 RDIMM slot connected to the FPGA along with two ZDok+ interfaces supporting high speed digital to analog converter (DAC)/analog to digital converter (ADC). A wide range of ADC boards are available to be interfaced with ROACH-2 with 1, 4, 8 and 16 ADCs [88,89] with sampling rates upto 240, 480 and 960 MHz per output. Fig. 3.1 and Fig. 3.2 shows the actual ROACH-2 board that we used to realize digital architectures and the table 3.1 depicts the specifications of the Virtex 6 FPGA included in the ROACH-2 board.

3.2 BEE3 multi-FPGA based rapid prototyping platform.

BEE3 is a full-speed FPGA prototyping platform which is used primarily for mixed-signal research and development, system-level real-time prototyping, high-speed RTL verification, system integration, and system validation. The BEE3 system consists of a 2U chassis with a tightly-coupled four FPGA system, widely employed in academia and industry. The main printed circuit Board (PCB) and a control & I/O PCB supports four Xilinx Virtex 5 FPGAs (XC5VSX95T-2FF1136), up to 16 DDR2 DIMMs, eight 10GBase-CX4 Interfaces, four PCI-Express slots, four USB ports, four 1GbE
RJ45 Ports, and one Xilinx USB-JTAG Interface, as illustrated in Fig. 3.3. Table 3.2 depicts the specifications of the Virtex 5 FPGA included in BEE3.

3.3 ML605 FPGA evaluation board

The ML605 fpga system is hardware tool which is heavily used in systems that demand high-performance, serial connectivity and advanced memory interfacing. The board consists of a Xilinx Virtex-6 XC6VLX240T-1FFG1156 FPGA and has serial connectivity through PCIe Gen2x4, Gen1x8, 1 SFP, 1 SMA Pair, and UART and Ethernet connectivity with Tri-Speed Ethernet (RJ-45) slot. And also it consists of

Figure 3.2: ROACH-2 hardware processing platform with the ADCs connected.
a DDR3 SO-DIMM (512 MB) memory and can expand I/O with the FPGA Mezzanine Card (FMC) interface. Fig. 3.4 shows a ML605 board while table 3.3 depicts the specifications of the XC6VLX240T FPGA. Finally table 4.1 compares all three boards, their pros and cons and specific scenarios each board will be useful more than the others.

![Figure 3.3: BEE3 device used to implement digital architectures. Labels A, B, C, and D indicate the four Xilinx Virtex-5 XC5VSX95T-2FF1136 FPGA devices.](image)

Figure 3.3: BEE3 device used to implement digital architectures. Labels A, B, C, and D indicate the four Xilinx Virtex-5 XC5VSX95T-2FF1136 FPGA devices.
Table 3.1: ROACH-2 FPGA Specifications.

<table>
<thead>
<tr>
<th>Part Number</th>
<th>XC6VSX475T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>74,400</td>
</tr>
<tr>
<td>Logic Cells</td>
<td>476,160</td>
</tr>
<tr>
<td>DSP48E1 Slices</td>
<td>2,016</td>
</tr>
<tr>
<td>Max Block RAM</td>
<td>38,304</td>
</tr>
<tr>
<td>Max User I/O</td>
<td>840</td>
</tr>
</tbody>
</table>

Figure 3.4: ML605 device used to implement digital architectures.
Table 3.2: BEE3 FPGA Specifications.

<table>
<thead>
<tr>
<th>Part Number</th>
<th>XC5VSX95T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtex 5 Slices</td>
<td>14720</td>
</tr>
<tr>
<td>Max Distributed RAM (Kb)</td>
<td>1,520</td>
</tr>
<tr>
<td>DSP48E Slices</td>
<td>640</td>
</tr>
<tr>
<td>Max Block RAM</td>
<td>8,784</td>
</tr>
<tr>
<td>Max User I/O</td>
<td>640</td>
</tr>
</tbody>
</table>

Table 3.3: ML605 FPGA Specifications.

<table>
<thead>
<tr>
<th>Part Number</th>
<th>XC6VLX240T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>37,680</td>
</tr>
<tr>
<td>Logic Cells</td>
<td>241,152</td>
</tr>
<tr>
<td>DSP48E1 Slices</td>
<td>768</td>
</tr>
<tr>
<td>Max Block RAM</td>
<td>14,976</td>
</tr>
<tr>
<td>Max User I/O</td>
<td>720</td>
</tr>
</tbody>
</table>
### Table 3.4: Comparison of digital FPGA prototyping boards.

<table>
<thead>
<tr>
<th>Property</th>
<th>ROACH-2</th>
<th>BEE3</th>
<th>ML605</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of the FPGA hardware</td>
<td>Type Virtex-6 and the biggest of them all but the board has one FPGA</td>
<td>Type-Virtex 5 and the smallest of them all but has four FPGA’s in one BEE3</td>
<td>Larger than the BEE3 FPGA and smaller than the ROACH-2 FPGA.</td>
</tr>
<tr>
<td>Parallel Connectivity</td>
<td>ROACH-2 boards can be connected parallel using multi-gigabit SFP+ cards</td>
<td>Intraconnect or interconnect is possible using the CX4 or RJ45 interfaces</td>
<td>Parallel connectivity is possible using multi-gigabit SFP and ethernet slots</td>
</tr>
<tr>
<td>Block RAM Size</td>
<td>Has the biggest block RAM out of the three boards</td>
<td>Has the smallest RAM but BEE3 system has four of such block rams hence bigger than the Virtex-5 on the BEE3 but smaller than the Virtex-6 on the ROACH-2</td>
<td>Block RAM is bigger than the ML605</td>
</tr>
<tr>
<td>Applications</td>
<td>Useful for systems that need extensive digital signal processing (Radio Telescopes)</td>
<td>Useful for parallel processing and mainly used in computer architecture research</td>
<td>Used for designs that need high-performance and serial connectivity</td>
</tr>
</tbody>
</table>
CHAPTER IV
INTRODUCTION TO A-DFT

Direct computation of a DFT requires $N^2$ complex multiplications and $N^2N$ complex additions [77]. An $N$ point FFT factorizes the DFT [38, 78] matrix into a product of sparse factors. Sparse factorization reduces the complexity of the DFT to $O(N \log N)$. However, the arithmetic complexity of all possible FFT algorithms have been bound to a lower theoretical limit $O(N)$ by Heidemann [41]. This bound is the lowest possible arithmetic complexity for computing the DFT regardless of the FFT used. That is, no matter how deeply one searches for better FFT algorithms, it is provably impossible to reduce arithmetic complexity beyond $O(N)$. When an $N$—point discrete sequence is highly sparse in the frequency domain, the limiting complexity $O(N)$ can be beaten using sparse FFTs. For example, for high sparsity, there has been progress in the design of iterative algorithms, such as MIT’s “Faster Than Fast-Fourier Transform” [39, 90–94].

Although the FFT itself transforms perfectly in theory, it is a component of a lossy scheme or hardware implementation. What if the DFT itself were allowed to be lossy? What if we can create an “almost-DFT” that, if we accept a tiny deviation from the exact DFT (that has no appreciable effect in the application) can achieve circuit complexities and power consumptions that are tremendously smaller than that
of the current state-of-the-art FFT blocks? Such an improvement will open up new possibilities, such as solar-powered systems, ultra-low power wireless systems, energy-harvesting powered systems, ultra-long battery life and body-embedded low-power systems.

Algorithms are explored for optimizing the size, weight and power (SWaP) for DFT and other linear transforms (LT) operations of the form $y = Ax$ where $x$ and $y$ are $N$ element vectors, and $A_{N \times N}$ is a matrix. Direct “brute-force computation” of LT algorithms require $O(N^2)$ multiplications, which is expensive in both hardware complexity and power consumption. The field of fast algorithms find matrix factorizations of $A$, such that $A = A_1 A_2 \cdots A_m$ where $A_k, k = 1, 2, \ldots, m$ are sparse matrices (butterfly structures), leading to tremendously reduced multiplier complexity. Traditional fast algorithms do not take into account circuit-level trade-offs (e.g., power, chip area, gate-count and quantization) during algorithm design.

Replacing a given LT with an approximation that does not require multiplication is a tremendous improvement in terms of SWaP, because multiplication is an expensive operation in hardware implementation. The hardware complexity of FFTs play a critical role where SWaP must be minimized. The circuit complexity and power consumption of multipliers exceed that of adders. In fact, in this research we explore a-DFTs, such that substantial savings in SWaP is achieved by slightly trading-off accuracy. To wit, we propose provably-accurate algorithms that offer **a-DFT performance albeit at ultra low SWaP** by removing multiplications.
4.1 Proposed a-DFT: Mathematical Definition

If an exact matrix $A$ can be approximated such that a small computational inaccuracy is allowed, we can create LT operations $\hat{A} = DT = DT_1T_2...T_m \approx A$, where $D$ is a diagonal matrix, so that the approximation can be implemented using fast algorithms that are realized as a factored product of $m$ sparse matrices $T_k$, $k = 1, 2, .., m$, corresponding to a cascade of $m$ low-complexity butterflies, which are simple to realize.

A matrix approximation is a LT $\hat{A}$ that—according to a specified metric—behaves similarly to an exact LT $A$. Let $x$ be the input to the linear system $A$. The output is $y = Ax$. The output of the approximate linear system $\hat{A}$ is $\hat{y} = \hat{A}x$. There are many ways to derive an approximation for $A$. The goal is to find an approximation $A^*$ that is optimal in a well-defined sense. Mathematically, the approximation problem can be stated as a constraint optimization problem: $A^* = \arg \min_{\hat{A}} \text{error}(y, \hat{y})$, where $\text{error}(\cdot, \cdot)$ is the objective function (figure of merit) imposed by the particular context where the LT is relevant. The above problem must be subject to the constraint that the computational cost of computing $\hat{y}$—in both theory and implementation—are much less than computing $y$. The low-cost constraint translates into deriving methodologies to obtain approximate matrices $\hat{A}$. The optimization problem can be recast as: $A^* = \arg \min_{\hat{A}} \text{error}_1(A, \hat{A})$, where $\text{error}_1(\cdot, \cdot)$ is a matrix-based objective function that captures the error in a relevant sense. Possible error functions are 1 matrix norms (e.g., Frobenius norm) [95], 2 mean squared error [96], 3 trans-
form efficiency, and 4 coding gain [97], among others. The low computation cost constraint imposed on \( \hat{A} \) leads to technical challenges towards solving the above problem. In fact, matrix approximation is often a non-linear optimization problem requiring sophisticated solving techniques. For instance, this property can be satisfied by restricting the entries of an approximate matrix to the set of powers of two \( \{0, \pm 1, \pm 2, \pm 4, \pm 8, \ldots \} \), which can be implemented by only bit-shifting operations or to the set of small integers \( \{0, \pm 1, \pm 2, \pm 3, \ldots \} \), which can be represented with an extremely reduced number of additions. Additionally, several objective functions can be considered simultaneously, resulting in a multicriteria optimization problem as follows: \( A^* = \arg \min_{\hat{A}} \{ \text{error}_1(A, \hat{A}), \text{error}_2(A, \hat{A}), \ldots, \text{error}_n(A, \hat{A}) \} \), where \( \text{error}_i(\cdot, \cdot), i = 1, 2, \ldots, n \) form a set objective functions. Thus, the inclusion of additional objective functions increases the difficulty of the problem as it becomes more sophisticated. However, the results are expected to be more meaningful, as the obtained approximations satisfy more desirable properties. An approximation matrix \( \hat{A} \) is usually based on an LT transformation matrix \( T \) of low computational complexity. Indeed, matrix \( T \) is the key component of a given LT approximation. Brute force direct exhaustive search over the set of matrices with low-complexity entries is an unfeasible task.

8-point DFT Approximation Algorithm

The DFT is a linear orthogonal transformation relating an \( N \)-point input vector \( v = [v_0 \ v_1 \ \ldots \ v_N]^\top \) to an output vector denoted by \( V = [V_0 \ V_1 \ \ldots \ V_N]^\top \)
according to \( V_k = \sum_{n=0}^{N-1} v_n \cdot \omega_n^{kn}, \ k = 0, 1, \ldots, N - 1 \), where \( \omega_N = \exp\{-2\pi j/N\} \) is the \( N \)th root of unity and \( j = \sqrt{-1} \). In matrix formalism, the above expression reduces to \( V = F_N v \), where \( F_N \) is the DFT matrix, whose \((i,k)\)th element is given by \( f_{i,k} = \omega_N^{ik} \), for \( i, k = 0, 1, \ldots, N - 1 \). For \( N = 8 \), the DFT matrix presents the following structure \([49, 50, 98]\):

\[
F_8 = \begin{bmatrix}
\alpha_0 & \alpha_0 & \alpha_0 & \alpha_0 & \alpha_0 & \alpha_0 & \alpha_0 & \alpha_0 \\
\alpha_0 & \alpha_1 & -\alpha_1^* & -\alpha_0 & -\alpha_0 & \alpha_0 & \alpha_0 & \alpha_0 \\
\alpha_0 & -\alpha_0 & \alpha_0 & -\alpha_1 & -\alpha_0 & \alpha_0 & \alpha_0 & \alpha_0 \\
\alpha_0 & -\alpha_1 & -\alpha_0 & \alpha_0 & -\alpha_0 & \alpha_0 & \alpha_0 & \alpha_0 \\
\alpha_0 & \alpha_1 & -\alpha_1^* & -\alpha_0 & -\alpha_0 & \alpha_0 & \alpha_0 & \alpha_0 \\
\alpha_0 & -\alpha_0 & \alpha_0 & -\alpha_0 & \alpha_0 & -\alpha_0 & \alpha_0 & \alpha_0 \\
\alpha_0 & -\alpha_1 & -\alpha_0 & \alpha_0 & -\alpha_0 & \alpha_0 & -\alpha_0 & \alpha_0 \\
\alpha_0 & -\alpha_0 & \alpha_0 & -\alpha_0 & \alpha_0 & -\alpha_0 & \alpha_0 & -\alpha_0 \\
\end{bmatrix}
\]

(4.1)

where \( \alpha_0 = 1 \), \( \alpha_1 = (1 - j)/\sqrt{2} \), \( \alpha_2 = -j \), and the superscript * denotes the conjugation operation. The 8-point DFT matrix \( F_8 \) was submitted to the parametric-based optimization method described in \([99]\) to derive a matrix approximation. Two major constraints were imposed on the sought approximations: 1) near-orthogonality and 2) low-complexity. Thus, the optimal elements for the parametric approximation of \( F_8 \) are 1, \((1 - j)/2\), and \(-j\). Such parameters result in the following matrix approximation \([49]\).

\[
\hat{F}_8 = \frac{1}{2} \begin{bmatrix}
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
\end{bmatrix}
\]

Compared to the exact DFT matrix, the above approximation has a mean squared error of 0.686, which is considered very low. Although not exactly orthogonal, the proposed approximation is very close to orthogonality.

The proposed 8-point transform \( \hat{F}_8 \) by Dr. Renato and his team preserves the symmetry of the DFT and has zero multiplicative complexity requiring only 64
real additions and 32 bit shifting operations. A tailored fast algorithm based on the matrix factorization methods suggested in [100], was used to further reduce the additive complexity and the fast algorithm can be given:

\[ \hat{F}_8 = Y_4 \times Y_3 \times D_1 \times Y_2 \times D_2 \times Y_1 \times P, \]

where

\[
Y_4 = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & -1
\end{bmatrix},
\]

\[
Y_3 = \begin{bmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix},
\]

\[
Y_2 = \begin{bmatrix}
1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix},
\]

\[
Y_1 = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix},
\]

\[
D_1 = \text{diag}(1, 1, 1, 1, 1/2, 1, 1/2),
\]

\[
D_2 = \text{diag}(1, 1, 1, j, j, j, j),
\]

\[
P = \begin{bmatrix}
e_1 & e_5 & e_3 & e_7 & e_2 & e_4 & e_8 & e_6
\end{bmatrix}^T
\] is a permutation matrix, and \(e_i\) is the 8-point column vector with element 1 at the \(i\)th position and 0 elsewhere. The above proposed 8-point algorithm has reduced the required real additions to 26 along with 2 bit shifting operations. Figure 4.1 depicts the signal flow graph of the 1-D 8-point ADFT algorithm. Traditionally, the 2-D version of the DFT is computed using a row-column (RC) decomposition, where initially the 1-D DFTs are computed along the rows followed by 1-D DFTs along the columns to get the 2-D version of the algorithm.
Figure 4.1: Signal flow graph for the factorization of $\hat{F}_8$. Input data $v_i$, $i = 0, 1, \ldots, 7$, relates to the output $V_k$, $k = 0, 1, \ldots, 7$. Dotted arrows represent multiplications by $-1$.

For the $8 \times 8$ block $A_8$ has its 2-D transform mathematically expressed by [101]:

$$B_8 = \hat{F}_8 \cdot A_8 \cdot \hat{F}_8^\top,$$

(4.2)

where $B_8$ is the $8 \times 8$ block output matrix in frequency.

16-point DFT Approximation Algorithm

Similarly, the proposed 16-point ADFT matrix $\hat{F}_{16}$ was derived by Dr. Renato and Dr. Fabio, to satisfy the constrains of near-orthogonality and low-complexity and the optimal elements for the parametric approximation of $\hat{F}_{16}$ is given by $\alpha_{16}^*$ where [102]

$$\alpha_{16}^* = 1/2[2 \quad 2j \quad 1 - j \quad 1 - 2j \quad 2j].$$

The resulting 16-point ADFT matrix is given by [102]:

$$\hat{F}_{16} = \frac{1}{2} \begin{pmatrix} A_{0,0} & A_{0,1} \\ A_{1,0} & A_{1,1} \end{pmatrix}$$

(4.3)
where each $A_{i,j}$ is an $8 \times 8$ matrix and can be shown by [102]:

$$
A_{0,0} = \left[
\begin{array}{cccccccc}
2 & 2 & -1i & 1 & -2i & 2 & -2 & 2 \\
2 & 1-1i & -2i & -1-1i & -2 & -1+1i & +2i & 1+1i \\
2 & 1-2i & -1-1i & -2+1i & 2 & 1+1i & -1 & -1-2i \\
2 & -1-2i & -1+1i & 2+1i & -2 & -2+1i & 1+1i & 1-2i \\
2 & -1-1i & +2i & 1-1i & -2 & 1+1i & -2 & -1+1i \\
2 & -2-1i & 1+1i & -1-2i & 2 & -1+2i & 1-1i & 2-1i \\
\end{array}
\right],
$$

$$
A_{0,1} = \left[
\begin{array}{cccccccc}
2 & 2 & -2 & 2 & 2 & 2 & 2 & 2 \\
2 & -2 & -2 & 2 & -2 & 2 & 2 & 2 \\
2 & 1-1i & -2i & -1+1i & 2 & 1+1i & -2 & -1+2i \\
2 & -2i & -2 & +2i & 2 & -2 & -2 & +2i \\
-2 & 1+2i & 1+1i & -2 & 1+1i & -2 & -1+2i & 1-1i \\
2 & -1-1i & +2i & 1-1i & -2 & 1+1i & -2 & -1+1i \\
2 & 2+1i & -1-1i & 1+2i & -2 & -1+2i & 1-1i & 2-1i \\
\end{array}
\right],
$$

$$
A_{1,0} = \left[
\begin{array}{cccccccc}
2 & 2 & -2 & 2 & -2 & 2 & 2 & 2 \\
2 & -2 & -2 & 2 & -2 & 2 & 2 & 2 \\
2 & 1+1i & -2i & -1+1i & 2 & 1+1i & -2 & -1+2i \\
2 & -2i & -2 & +2i & 2 & -2 & -2 & +2i \\
2 & 1+2i & -1+1i & 2+1i & -2 & -1+2i & 1+1i & 1-2i \\
2 & -2 & -2i & 2 & +2i & 2 & +2i & 2 \\
2 & 1+1i & -2i & -1+1i & 2 & -1+2i & 1+1i & 1-1i \\
\end{array}
\right],
$$

$$
A_{1,1} = \left[
\begin{array}{cccccccc}
2 & 2 & -2 & 2 & -2 & 2 & 2 & 2 \\
2 & -2 & -2 & 2 & -2 & 2 & 2 & 2 \\
-2 & 1-2i & 1+1i & -2i & -2 & 1+1i & -2 & -2 \\
2 & -2i & -2 & +2i & 2 & -2 & -2 & +2i \\
-2 & 1+2i & 1+1i & -2i & -2 & 1+1i & -2 & -2 \\
2 & -2i & -2 & +2i & 2 & -2 & -2 & +2i \\
2 & 1+1i & +2i & 1+1i & -2 & -1+2i & 1+1i & 1-1i \\
\end{array}
\right].
$$

The 16-point matrix can be further factorized to reduce the complexity of the algorithm by using the decimation in frequency methods [100]. The fast algorithm can be given by [102]:

$$
\hat{F}_{16} = W_5 \times W_4 \times W_3 \times W_2 \times D_1 \times W_1,
$$

(4.4)

where [102]

$$
W_5 = \left[
\begin{array}{cccccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\right].
$$

37
\[
W_4 = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
\end{bmatrix},
\]
\[
W_3 = \begin{bmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix},
\]
\[
W_2 = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -10 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -10 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -10 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{bmatrix},
\]
\[
W_1 = \begin{bmatrix}
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix},
\]

and

\[
D = \frac{1}{2} \text{diag}(1, 1, 1, 1, 1, 1, 1, j, j, j, j, j, j, j, j, j).
\]
Similar to the 8-point 2-D approximation, the 16-point 2-D approximation can be expressed for a $16 \times 16$ block $A_{16}$ as

$$B_{16} = \hat{F}_{16} \cdot A_{16} \cdot \hat{F}_{16}^\top. \quad (4.5)$$

Fig. 4.2 depicts the signal flow graph of the proposed 1-D 16-point algorithm.

Figure 4.2: Signal flow graph for the factorization of $\hat{F}_{16}$. Input data $v_i, i = 0, 1, \ldots, 15$, relates to the output $V_k, k = 0, 1, \ldots, 15$. 

39
4.2 Arithmetic Complexity

In order to properly assess and quantify the gains in SWaP the proposed a-DFTs compared to regular FFT algorithms in the literature, we carried out a research to evaluate the arithmetic complexity of each algorithm for complex input/outputs. We calculated multiplications and additions each algorithm needed and tabled them for fair comparison. Since these are complex multipliers, the Gaussian complex multiplier algorithm was used to properly convert the complex adders into hardware realizable real adders. And the important thing to note is that even though some algorithms consider multiplying by $\omega_0$ which is $e^{-2\pi 0/N} = 1$, such multiplications by one or zero were not considered as multipliers. So in table 4.1, we believe a fair comparison can be made with the proposed a-DFT for both 8-point and 16-point cases. In table 4.1 column two and three depicts the complex adders and multipliers while column four and five depicts the number of real adders and multipliers. While the last column shows the theoretical lower bounds according do Heidermann [103].

4.3 a-DFT: Digital Architectures and Realizations

In this section, hardware architectures for the proposed 8-point and 16-point a-DFT algorithms are detailed for both 1-D and 2-D versions. Introduced architectures were submitted to (i) FPGA implementations, (ii) ROACH-2 implementations,(iii) Complementary metal oxide semiconductor (CMOS) 45 nm ASIC implementation up
Table 4.1: Arithmetic Complexity of each FFT algorithm compared with the proposed a-DFT.

<table>
<thead>
<tr>
<th>Approximation</th>
<th>Algorithm</th>
<th>Complex Adders/Subtracts</th>
<th>Complex Multipliers</th>
<th>Real Adders/Subtracts</th>
<th>Real Multipliers</th>
<th>Lower Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-point</td>
<td>Radix-8</td>
<td>24</td>
<td>2</td>
<td>58</td>
<td>6</td>
<td>Heidermann</td>
</tr>
<tr>
<td></td>
<td>Cooley-Tukey Radix-2</td>
<td>24</td>
<td>5</td>
<td>73</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Winograd (8-point)</td>
<td>26</td>
<td>2</td>
<td>62</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>a-DFT (8-point)</td>
<td>26</td>
<td>0</td>
<td>52</td>
<td>0</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>16-point</td>
<td>Radix-2</td>
<td>64</td>
<td>17</td>
<td>213</td>
<td>51</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Radix-4</td>
<td>64</td>
<td>10</td>
<td>178</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Split-Radix</td>
<td>64</td>
<td>10</td>
<td>178</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Winograd (16-point)</td>
<td>74</td>
<td>10</td>
<td>198</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>a-DFT (16-point)</td>
<td>72</td>
<td>0</td>
<td>144</td>
<td>0</td>
<td>Not Applicable</td>
</tr>
</tbody>
</table>
to synthesis level and (iv) CMOS 18 nm ASIC implementation up to place & route level using AMS standard cells.

4.3.1 Digital Architectures for the 8-point and 16-point DFT Approximations

Initially, 1-D architectures of the approximation algorithms for the 8-point and 16-point DFT were implemented. The algorithms deal with complex numbers hence both had a real section and an imaginary section. The real section and the imaginary section were implemented separately using two identical architectures and outputs were combined together to furnish the final algorithms. Fig. 4.3 depicts the architecture for the 8-point 1-D approximate DFT, while Fig. 4.4 depicts the architecture for the 16-point a-DFT.

Figure 4.3: 1-D architecture of the proposed 8-point DFT Approximation.
Figure 4.4: 1-D architecture of the proposed 16-point DFT Approximation.
The 2-D version of the 8-point and 16-point DFT approximation architectures was realized using two 1-D transforms and a transpose buffer. For both 8-point and 16-point approximations, the first instantiation of the a-DFT block furnishes a row-wise transform computation of the input, while the second implementation furnishes a column-wise transformation of the previous intermediate result. A real time row-parallel transposition buffer circuit is required in between the 1-D transformation blocks. The transposition buffer block orders the data by converting the row-transformed data from the first DFT approximation circuit into a transposed format, which will be fed to the second DFT approximation block. Both 1-D transformation blocks and the transposition buffer were modeled and tested in Matlab/Simulink; then they were combined to complete the 2-D approximate transform. Fig. 4.5 shows the implementation of the 2-D transform by means of the 1-D transforms for both 8-point and 16-point algorithms.

4.3.2 FPGA and ASIC realizations and results

The above 2-D architectures were physically realized on an FPGA-based rapid prototyping system and tested using on-chip hardware-in-the-loop co-simulation. Xilinx Virtex-6 XC6VLX240T-1FFG1156 device was employed to physically realize the architectures on FPGA with fine-grain pipelining for increased throughput. The realizations were verified on FPGA chip using a Xilinx ML605 board at a clock frequency of 50 MHz. Input precision level was varied to compare the performance in terms of digital logic resource consumptions at varied degrees of numerical accuracy and
dynamic range. Adopting system word length $L \in \{4, 8, 12, 16\}$, we applied 10,000 random 8-point input test vectors using hardware co-simulation. Test vectors were generated from within the MATLAB environment and routed to the physical FPGA device using JTAG based hardware co-simulation. Then measured data from the FPGA was routed back to MATLAB memory space. Evaluation of hardware complexity and real time performance considered the following metrics: the number of configurable logic blocks (CLB), flip-flop (FF) count, critical path delay ($T_{cpd}$), and the maximum operating frequency ($F_{max}$) in MHz. The xflow.results report file was accessed to obtain the above results. Dynamic ($D_p$) and static power ($Q_p$) consumptions were estimated using the Xilinx XPower Analyzer for the Xilinx Virtex-6 XC6VLX760-1FF1760 FPGA device and the results are shown in Table 4.2. For the ASIC synthesis, the hardware description language code was ported to 45 nm CMOS

Figure 4.5: Two-dimensional approximate algorithm for 8-point and 16-point DFT by means of 1-D approximate transform. Signal $v_{j,0}, v_{j,1}, \ldots$ corresponds to the rows of the input; $V_{j,0}, V_{j,1}, \ldots$ indicates the transformed rows; $V_{0,k}, V_{1,k}, \ldots$ indicates the columns of the transposed row-wise transformed image; and $V^{(2-D)}_{0,k}, V^{(2-D)}_{1,k}, \ldots$ indicates the columns of the final 2-D transformed input.
Table 4.2: Hardware resource consumption using Xilinx Virtex-6 XC6VLX-240T 1FFG1156 device and power consumption using a Xilinx Virtex-6 XC6VLX760-1FF1760 device.

<table>
<thead>
<tr>
<th>Approximation</th>
<th>L</th>
<th>CLB</th>
<th>FF</th>
<th>$T_{cpd}$ (ns)</th>
<th>$F_{max}$ (MHz)</th>
<th>$D_p$ (mW/MHz)</th>
<th>$Q_p$ (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-point</td>
<td>4</td>
<td>7121</td>
<td>1702</td>
<td>5.786</td>
<td>172.83</td>
<td>10.29</td>
<td>4.54</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8853</td>
<td>1973</td>
<td>5.386</td>
<td>185.66</td>
<td>11.16</td>
<td>4.56</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>10581</td>
<td>2717</td>
<td>9.134</td>
<td>109.48</td>
<td>20.18</td>
<td>4.57</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>12309</td>
<td>3205</td>
<td>9.519</td>
<td>105.05</td>
<td>22.90</td>
<td>4.56</td>
</tr>
<tr>
<td>16-point</td>
<td>4</td>
<td>24317</td>
<td>5031</td>
<td>7.429</td>
<td>134.60</td>
<td>24.44</td>
<td>4.54</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>30143</td>
<td>5326</td>
<td>8.379</td>
<td>119.34</td>
<td>34.02</td>
<td>4.56</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>35967</td>
<td>7187</td>
<td>8.688</td>
<td>115.10</td>
<td>37.67</td>
<td>4.56</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>41791</td>
<td>8628</td>
<td>12.126</td>
<td>82.46</td>
<td>57.57</td>
<td>4.57</td>
</tr>
</tbody>
</table>

technology and subject to synthesis using Cadence Encounter. The FreePDK, a free open-source ASIC standard cell library at the 45 nm CMOS node, was used for this purpose. The supply voltage of the CMOS realization was fixed at $V_{DD} = 1.1$ V during estimation of power consumption and logic delay. The adopted figures of merit for the ASIC synthesis were: area ($A$) in mm$^2$, area-time complexity ($AT$) in mm$^2$·ns, area-time-squared complexity ($AT^2$) in mm$^2$·ns$^2$, dynamic ($D_p$) and static ($Q_p$) power consumption in watts, critical path delay ($T_{cpd}$) in ns, and maximum operating frequency ($F_{max}$) in MHz. Results are displayed in Table 4.3.
4.4 Hardware Resource Consumption Comparison & Discussion

To properly assess the SWaP characteristics of the new algorithm, we implemented the 1-D design of the 8-point and the 16-point versions along with the most popular fast algorithms in the literature for 32-bit input word lengths (twiddle-factors @ 16-bit precision). 8-point and 16-point Radix-2 Cooley-Tukey algorithm along with the Winograd 8-point and 16-point algorithms were realized in both FPGA and ASIC as they are the most popular and efficient FFT’s available. The FPGA designs were pipelined to increase throughput and were realized in a ROACH-2 Board. Table 4.4 shows the hardware resource comparison for the algorithms on ROACH-2.

Table 4.3: Hardware resource consumption using CMOS 45nm ASIC synthesis.

<table>
<thead>
<tr>
<th>Approximation</th>
<th>( L )</th>
<th>Area ((\text{mm}^2))</th>
<th>( AT )</th>
<th>( AT^2 )</th>
<th>( T_{\text{cpd}} ) (ns)</th>
<th>( F_{\text{max}} ) (MHz)</th>
<th>( D_p ) (mW/MHz)</th>
<th>( Q_p ) (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.097</td>
<td>0.095</td>
<td>0.093</td>
<td>0.98</td>
<td>1015</td>
<td>0.165</td>
<td>0.720</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.141</td>
<td>0.153</td>
<td>0.167</td>
<td>1.09</td>
<td>917.43</td>
<td>0.234</td>
<td>1.028</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.190</td>
<td>0.209</td>
<td>0.229</td>
<td>1.10</td>
<td>909.09</td>
<td>0.311</td>
<td>1.379</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.240</td>
<td>0.276</td>
<td>0.317</td>
<td>1.15</td>
<td>869.56</td>
<td>0.384</td>
<td>1.729</td>
<td></td>
</tr>
<tr>
<td>16-point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.409</td>
<td>0.449</td>
<td>0.494</td>
<td>1.10</td>
<td>909.09</td>
<td>0.710</td>
<td>3.057</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.560</td>
<td>0.660</td>
<td>0.779</td>
<td>1.18</td>
<td>847.45</td>
<td>0.971</td>
<td>4.184</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.732</td>
<td>0.871</td>
<td>1.036</td>
<td>1.19</td>
<td>840.33</td>
<td>1.243</td>
<td>5.397</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.877</td>
<td>1.096</td>
<td>1.370</td>
<td>1.25</td>
<td>800</td>
<td>1.504</td>
<td>6.524</td>
<td></td>
</tr>
</tbody>
</table>
Table 4.4: Hardware resource consumption using Xilinx Virtex-6 XC6VSX-475T 1FF1759 on ROACH-2 device.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>CLB</th>
<th>FF</th>
<th>$T_{cpd}$ (ns)</th>
<th>$F_{max}$ (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a-DFT</td>
<td>1498</td>
<td>4153</td>
<td>4.817</td>
<td>207.59</td>
</tr>
<tr>
<td>Radix-2 Cooley-Tukey</td>
<td>1643</td>
<td>5357</td>
<td>11.753</td>
<td>85.08</td>
</tr>
<tr>
<td>Winograd</td>
<td>1351</td>
<td>4715</td>
<td>4.323</td>
<td>231.32</td>
</tr>
</tbody>
</table>

For the ASIC synthesis, the hardware description language code was ported to 180 nm CMOS technology using AMS standard cells and subject to synthesis and place & route using Cadence Encounter tool. Table 4.5 summarizes the CMOS hardware resource consumption results and the 8-point algorithm show reductions of 35% in chip-area and 48% in power compared to the radix-2 Cooley-Tukey and 16% in chip-area and 35% in power compared to the Winograd algorithm.

Table 4.5: Hardware resource consumption for CMOS 180nm ASIC place & route.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Area (mm$^2$)</th>
<th>$AT$</th>
<th>$AT^2$</th>
<th>$T_{cpd}$ (ns)</th>
<th>$F_{max}$ (MHz)</th>
<th>$D_p$ (mW/MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a-DFT</td>
<td>0.709</td>
<td>3.694</td>
<td>19.252</td>
<td>5.211</td>
<td>191.90</td>
<td>0.323</td>
</tr>
<tr>
<td>Radix-2 Cooley-Tukey</td>
<td>1.094</td>
<td>6.860</td>
<td>47.062</td>
<td>6.271</td>
<td>159.46</td>
<td>0.630</td>
</tr>
<tr>
<td>Winograd</td>
<td>0.849</td>
<td>4.928</td>
<td>28.609</td>
<td>5.805</td>
<td>172.26</td>
<td>0.497</td>
</tr>
</tbody>
</table>
Antenna array based radio frequency (RF) applications, such as radar, wireless communications, localization, remote sensing, signal intelligence, radio astronomy, search for extraterrestrial intelligence (SETI), and imaging requires the fundamental operation of receive mode beamforming. To wit, beamforming is precisely the directional enhancement of propagating electromagnetic planar-waves based on their directions of arrival (DOA), whilst suppressing undesired noise and interference that impinge on the antenna array. There are many digital beamforming algorithms for achieving independent RF beams, at specified angles from array broadside, such as adaptive beamformers \[104–107\], delay-and-sum beamformers \[108, 109\], Frost beamformers \[110\], phased-arrays implemented in the temporal frequency domain \[111, 112\], filter-and-sum beamformers using finite-impulse response (FIR) spatial digital filters \[113, 114\], and spatial DFT based beamformers \[115\].

The ability to form multiple receiver beams is known as “multi-beamforming” \[115, 116\], where multi-beamformers produce multiple beams with the use of antenna array and signal processing techniques. If we consider a particular temporal frequency \(f_0\), multi-beamformer can have multiple orthogonal beams with different DOAs equal
to the number of antenna elements. Multiple RF beams, each having a unique “look direction”—the direction of maximum sensitivity—is needed for multiple visibilities. Multiple simultaneous beams are also needed for search-and-track radar, which in volume-scan mode, continuously monitor airborne threats, such as aircraft, warheads and cruise missiles, across a given range of angles. From the standpoint of high-capacity wireless communications, simultaneous receiver beams are of importance to multi-input multi-output (MIMO) systems.

The application of the DFT along the spatial array of receiver antennas, usually in a uniformly spaced linear array configuration, leads to \( N \) receive mode RF beams. The application of a \( N \)-point spatial-DFT at each time frame leads to \( N \) independent beams having array factors such as shown in Fig. 5.1. Traditional FFT blocks are used in achieving multi-beams across a ULA of antennas and in this chapter we consider the multiplierless 8-point DFT approximation [49] and 16-point DFT approximation in achieving RF multi-beam apertures very similar to those achieved using conventional FFT based algorithms. Such low-cost computation trades the beamformer array pattern accuracy, for significantly lower digital arithmetic complexity. The a-DFT [49] allows beamforming with zero power consumption related to power demanding multiplication operations. Thus, the computational framework of associated system is essentially constituted of adders only.

The main goal is the reduction of digital arithmetic complexity of the spatial DFT-based multi-beam receive apertures using the a-DFT. Significant reductions in circuit complexity and power consumption in the digital implementation—with a
small cost in terms of performance—can be achieved. To wit, by adopting a multiplierless approximation to the ideal DFT, multi-beams can be obtained at a fraction of the arithmetic (and circuit) complexity of a traditional DFT based multi-beam RF digital aperture array receiver.

A particularly important emerging application for multiple beams would be in the area of massively MIMO wireless systems that are attempting to utilize unused and currently unlicensed bandwidth in the 25-150 GHz region of the electromagnetic spectrum [117]. The exploitation of mm-wave frequencies at these high frequency bands is particularly suited for establishing a large number of high-bandwidth connections over relatively short distances (within a room, for example), to connect a base station with a multitude of mobile wireless devices as shown in Fig. 5.2. As a design specific ratio, we assume our need is to realize a miniaturized digital antenna aperture array receiver providing 64 RF beams arranged in an $8 \times 8$ grid in the angular domain. The operational (center) frequency is assumed at 70 GHz, for purposes of explanation.

![Figure 5.1: Application of $N$-point spatial DFT for $N$-beam radar sensor.](image)
Figure 5.2: Example of high-bandwidth connections over relatively short distances.

The bandwidth per beam is assumed to be 500 MHz, which is sufficient for maintaining 500 Mbps data rates, as a minimum requirement, using even the basic binary phase shift keying (BPSK) modulation method. In practice, advanced modulation and coding schemes support better data rates over each RF beam. We chose 500 MHz of bandwidth per RF beam so that the required digital architecture is easily achievable using relatively low-cost VLSI CMOS fabrication techniques. At an RF frequency of $f = 70$ GHz, the wavelength is $\lambda = 4.28$ mm, which makes the Nyquist spacing of the array $\Delta x = \lambda/2 = 2.14$ mm. For an $8 \times 8$ element planar square antenna array, the total array size is $15 \times 15$ mm in size, making it small enough to fit on a variety of wearable/mobile devices.

The need for such massive connectivity within a small geographic area is an expected outcome of the rapidly expanding topic of the internet of things (IoT) [118,
The IoT may involve wearable technologies, body area networks (BANs), consumer electronics, network infrastructure, smart home technologies, security, smart connected health, and vehicular technologies and networks, such as vehicle to vehicle (V2V) and urban unmanned aerial systems (UAS), for example [120,121].

As shown in Fig. 5.3, consider a 2-D plane wave $w_d(x, ct)$ having a desired DOA given by $\psi_d$, where $ct$ is time $t$ normalized by the speed of light $c$. Received signals from a ULA of antennas, oriented along the $x$ axis, with uniform inter-element spacing $\Delta x$, can be denoted as $x(n_x, ct) = w_d(n_x \Delta x, ct)$, where $n_x = 1, 2, 3, \ldots, N$ is the antenna
index. The signal is amplified using a low noise amplifier (LNA) and the real ($I$, in-phase, $v_{\text{real}}$) and the imaginary ($Q$, quadrature, $v_{\text{im}}$) parts are low-pass filtered and sampled using ADCs. The resulting is a baseband signal $w_k, k = 0, 1, ..., N - 1$, where $\Delta T = F_{\text{clock}}^{-1}$, and where $F_{\text{clock}}$ is the sample-rate. Let the vector of sample values from the array, for every time instance $n_t$, be denoted $w = [w_0 \ w_1 \ ... \ w_{N-1}]^T$. The spatial DFT operation, computed across the array outputs leads to a time sequence of spatial frequency coefficients $W_k$ where $W = [W_1 \ W_2 \ ... \ W_{N-1}]^T$ found via $W = Aw$, where $A$ is the DFT matrix. The computation of $W_k$ gives $N$ directional beams in the array pattern [115, 116, 122, 123].

5.2 Multi-Beam RF Aperture using 1-D a-DFT: Simulations & Results

Simulations were carried out in realizing Multi-beam RF apertures using the 8-point a-DFT algorithm instead of the traditional FFT for a $N = 8$ element antenna array. The normalized center frequency $\omega_{ct0} = \pi$ was utilized and the input planar wave started out at a DOA of $-90^\circ$ and was slowly incremented to the DOA of $90^\circ$, at a rate of $0.1^\circ$ per 256 time samples on the linear array. The slowly changing DOA of the received waves on the aperture antenna progressively falls onto all of the eight RF beam main lobes furnished by the proposed a-DFT. The eight independent beams are pointed at angles $\psi_d = 0.00, \pm 14.47, \pm 30.00, \pm 48.59, 90.00$ in degrees measured from array broadside direction. The array patterns are given by:

$$P_i(\psi; \hat{F}_8) = \frac{|H_i(-\omega_{ct0} \sin(\psi); \hat{F}_8)|}{\beta_i},$$
where $|H_i(-\omega_{ct0}\sin(\psi); \hat{F}_k)|$ is the transfer function of the discrete filter and $\beta_i = \max_\psi |H_i(-\omega_{ct0}\sin(\psi))|$, for $i = 0, 1, \ldots, 7$, is a normalization factor [49]. The corresponding real and imaginary outputs of each bin of the a-DFT are shown in Figs. 5.4 and 5.5, respectively. The signal power with respect to time is depicted in Fig. 5.6 where each color represents the beam associated to the corresponding a-DFT bin. The power plots are computed by squaring and summing the real and imaginary outputs (that is, the in-phase $I$ and quadrature $Q$ components of each a-DFT bin) as a function of time. The eight traces are color coded to match the beam patterns of the multi-beamformer, and has a maximum on the time frame where the incident planar waves had a DOA directly on the main lobe of each of the eight array factors for the eight beams of the receive aperture. Fig 5.7 shows the corresponding polar plots that were taken using the a-DFT and the absolute error when compared with the ideal DFT. For the 16-point case, the same simulation was carried out but this time for $N = 16$ element antenna array. The corresponding results are shown in Fig. 5.8
Figure 5.4: Real output of each bin of the a-DFT.

Figure 5.5: Imaginary output of each bin of the a-DFT.
Figure 5.6: Normalized power with respect to time.

Figure 5.7: Multi-beams using 8-point a-DFT. Array with omni-antennas, $\Delta x = \lambda/2$ using 8-point exact DFT (left), proposed a-DFT (middle) and deviation/error (right)

5.3 a-DFT Rectangular Aperture

The two-dimensional (2-D) DFT is widely used in digital signal processing and computing applications, such as multi-beamforming RF and ultrasonic apertures, image
watermarking [124], fingerprint recognition [125], texture feature extraction [126], medical imaging [127,128], synthetic aperture radar (SAR) processing [129], and dig-

Figure 5.8: Multi-beams using 16-point a-DFT. Top: Array with omni-antennas, $\Delta x = \lambda/2$ using 16-point exact DFT (left), proposed a-DFT (middle) and deviation/error (right). Bottom: Sparse multi-beams for elements with $\cos^3\psi$ pattern, with $\Delta = \lambda$ using 16-point exact DFT (left), proposed a-DFT (middle) and deviation/error (right).
ital holographic imaging [130]. The main goal of this section is to exploit multiplierless DFT approximations and their fast algorithm realizations for furnishing high-speed multi-beam RF receive and transmit-mode apertures for wireless communications, radar, RF sensing, and microwave/mm-wave imaging systems.

The 2-D version of the a-DFT can be used to closely-approximate the 2-D $8 \times 8$-point DFT. By completely eliminating multiplications in the proposed a-DFT, we furnish major reductions in circuit complexity, chip area, and power consumption in any digital system that utilizes the 2-D DFT. Fig. 5.9 provides a multi-beam RF aperture application example, where we can use the 2-D DFT approximation to closely resemble antenna array patterns. The antenna grid will consist of 64 independent beam patterns. Several of them have been graphically shown in the figure.

![Figure 5.9: ULA based multi-beam RF aperture application using a spatial DFT.](image)

59
5.3.1 Multi-Beam RF Aperture using 2-D a-DFT: Simulations & Results

The 2-D DFT can be computed for a $8 \times 8$ idealized antenna array. The multiple beams on the uniform rectangular array (URA) aperture of elements is obtained by first applying the a-DFT along rows, resulting in row-transformed outputs, which in turn, are again transformed column-wise to effect the approximation of 2-D DFT using row-column separable transforms. Fig. 5.10 show the signal flow graph implementations of a real-time 64 beam RF aperture. Simulation results were performed for 64 idealized antennas (omnidirectional patterns) in a URA and the corresponding normalized array factors for the resulting 8-beams along a particular elevation are shown in Fig. 5.11. The use of the 2-D a-DFT to obtain a total of 64 simultaneous beams is possible. We show four cases of these 64 beams in Fig. 5.12. In our simulation setup, we modeled the complex filter bank responses of both DFT and a-DFT and found the complex error between each of the bins. The magnitude of this error was plotted as a function of frequency, in polar format. The error characteristics appear as a small deviation of the minor lobes in the array factor of the aperture antenna, where it is clear that the adoption of the a-DFT in place of the DFT leads to only minor differences in the beam personalities.

The 8-point FFT requires 15 real multiplier cores, assuming the use of Gauss complex multiplication algorithm. The proposed approximation does not require any multipliers at all. From a hardware complexity standpoint, the use of 64 simultaneous beams requires 16 parallel instantiations of 8-point FFT blocks in VLSI hardware. Therefore, the use of the proposed implementation for a $8 \times 8$ URA for 64-beams
requires a much reduced hardware count, because the number of parallel multipliers has reduced from $16 \times 15 = 240$ down to zero, while the parallel adder count for both designs are approximately the same. A key point to remember is that when we adopt the proposed approximation in place of the exact DFT/FFT, the resulting beams are scaled by a constant. This has no effect on the enhancement of a directional signal in the presence of noise and interference because both desired and undesired components of the signal are equally scaled by the same constant.

In this chapter application specific examples are provided for the new class of multiplierless hardware algorithm that were introduced in chapter 2, consisting only of arithmetic adder circuits that closely approximates the 1-D and 2-D version of the 8-point and 16-point DFT. By eliminating multiplication operations in the proposed a-DFT, unprecedented improvements in circuit complexity, chip area and power consumption were achieved and the results are shown in chapter 4. Design specifications are discussed in establishing a large number of high-bandwidth connections over relatively short distances and miniaturized digital antenna aperture array receivers. Simulation results that closely resemble the antenna array patterns for the 1-D and 2-D DFT and several beam examples for the ideal DFT, a-DFT and the absolute difference are provided.
Figure 5.10: Signal flow graph implementation of a real-time 64 beam RF aperture.
Figure 5.11: Normalized array beams for the 8-beams.

Figure 5.12: Array pattern from the Ideal DFT, array pattern from the a-DFT and the absolute difference.
CHAPTER VI

8-POINT & 16-POINT LOW-COMPLEXITY MULTIPLIERLESS DCT APPROXIMATIONS FOR LOW-POWER HEVC DIGITAL IP CORES

Recent years have experienced a significant demand for high dynamic range systems that operate at high resolutions [131]. In particular, high-quality digital video in multimedia devices [132] and video-over-Internet protocol networks [133] are prominent areas where such requirements are evident. Other noticeable fields are geospatial remote sensing [134], traffic cameras [135], automatic surveillance [131], homeland security [136], automotive industry [137], and multimedia wireless sensor networks [63], to name but a few. Often hardware capable of significant throughput is necessary; as well as allowable area-time complexity [63].

Of the many different trigonometric and wavelet transforms used in signal processing, the DCT plays an especially important role for image and video compression applications [3, 63, 138, 139]. In particular, the DCT has been adopted in several image and video coding schemes [140], such as JPEG [141], MPEG-1 [142], MPEG-2 [143], H.261 [144], H.263 [145], H.264 [146], and the recent HEVC [147]. For instance, H.264 and HEVC standards employ 8-point DCT algorithms [147–154] among other different block lengths, such as 4, 16, and 32 points [155, 156]. In [157], the 8-point DCT stage of the HEVC was optimized. The distinctive char-
acteristic of HEVC is, its capability of achieving high compression performance at approximately half the bit rate required by H.264/AVC with same image quality [147, 155, 158]. Also HEVC was demonstrated to be especially effective for high-resolution video applications [158]. However, HEVC possesses a significant computational complexity in terms of arithmetic operations [153, 155, 158]. In fact, HEVC can be 2–4 times more computationally demanding when compared to H.264/AVC [155]. Because of this, several algorithms for the efficient computation of the 8-point DCT have been proposed [159, 160]. In [20, 23], comprehensive surveys on DCT algorithms are detailed. Noteworthy DCT methods include the following procedures: Wang factorization [161], Lee DCT for power-of-two block lengths [162], Arai DCT scheme [163], Loeffler algorithm [164], and Feig-Winograd factorization [165]. All these algorithms are classical results in the field and have been considered for practical applications [142, 166, 167]. For instance, the Arai DCT scheme was employed in various recent hardware implementations of the DCT [3, 63, 168].

Because the DCT is a widely used mathematical tool in image and video compression, it follows that it is a core component in contemporary media standards such as JPEG and MPEG [169]. Indeed, the DCT is known for its properties of decorrelation, energy compaction, symmetry, and orthogonality [170]. The DCT enjoys its prominent role in compression because of its energy compaction property, which is very close to the statistically optimal Karhunen–Loève transform. Low-complexity approximations for the DCT, whether this is achieved via integer approximations or other fast algorithms, are of immense importance in the digital video industry.
6.1 Mathematical Review of DCT

There are several variants of the DCT depending on the type of boundary conditions used to define the finite length transform. The \( N \)-point DCT is algebraically represented by the \( N \times N \) transformation matrix \( C_N \) whose elements are given by [20,23]

\[
c_{m,n} = \frac{1}{\sqrt{N}} \beta_{m-1} \cos \left( \frac{\pi(m-1)(2n-1)}{2N} \right),
\]

where

\( m, n = 1, 2, ..., N, \beta_0 = 1, \) and \( \beta_k = \sqrt{2}, \) for \( k \neq 0. \) Let \( \mathbf{x} = [x_0 \ x_1 \ \cdots \ x_{N-1}]^\top \) be an input vector, where the superscript \( ^\top \) denotes the transposition operation. The one-dimensional (1-D) DCT of \( \mathbf{x} \) is the \( N \)-point vector \( \mathbf{X} = [X_0 \ X_1 \ \cdots \ X_{N-1}]^\top \) given by \( \mathbf{X} = C_N \cdot \mathbf{x} \). Because \( C_N \) is an orthogonal matrix, the inverse transformation can be written according to \( \mathbf{x} = C_N^\top \cdot \mathbf{X} \). Let \( \mathbf{A} \) and \( \mathbf{B} \) be square matrices of size \( N \).

For two-dimensional (2-D) signals, we have the following expressions that relate the forward and inverse 2-D DCT operations, respectively:

\[
\mathbf{B} = C_N \cdot \mathbf{A} \cdot C_N^\top \quad \text{and} \quad \mathbf{A} = C_N^\top \cdot \mathbf{B} \cdot C_N.
\]

(6.1)

6.2 Fast Algorithms for Computing DCT

Designing fast algorithms for the DCT is a mature area of research [24,103,164,168]; thus it is not realistic to expect major advances by means of standards techniques. On the other hand, the development of low-complexity approximations for DCT is an open field of research. In particular, the 8-point DCT was given several approximations, such as the signed DCT [25], the level 1 DCT approximation [60], the
Bouguezel-Ahmad-Swamy (BAS) series of transforms [52, 59, 171–173], the rounded DCT (RDCT) [174], the modified RDCT [62], the multiplier-free DCT approximation for RF imaging [139], and the improved a-DCT proposed in [157]. Such approximations reduce the computational demands of the DCT evaluation, leading to low-power consumption and high-speed hardware realizations [157]. At the same time, approximate transforms can provide adequate numerical accuracy for image and video processing.

In response to the growing need for higher compression rates related to real time applications [175], the HEVC video compression format [147] was proposed. Different from its predecessors, the HEVC employs not only $8 \times 8$ size blocks, but also $4 \times 4$, $16 \times 16$, and $32 \times 32$. Several approximations for 16-point DCT based on the integer cosine transform [176] were proposed in [177], [178] and [179]. These transformations are derived from the exact DCT after scaling the elements of the DCT matrix and approximating the resulting real-numbered entries to integers [178]. Therefore, real multiplications can be completely eliminated, at the expense of a noticeable increase in both the additive complexity and the number of required bit-shifting operations [20].

A more restricted class of DCT approximations prescribe transformation matrices with entries defined on the set $C = \{0, \pm 1/2, \pm 1, \pm 2\}$. Because the elements of $C$ imply almost null arithmetic complexity, resulting transformations defined over $C$ have very low-complexity, requiring no multiplications and a reduced number of bit-shifting operations. In this context, methods providing 16-point low-cost orthogonal
transforms include the Walsh–Hadamard transform (WHT) [180,181], BAS-2010 [52], BAS-2013 [59], and the approximate transform proposed in [138], here referred to as BCEM approximation.

6.3 Proposed DCT Approximations: Mathematical Definition

6.3.1 8-point DCT Approximation

In deriving a novel low-complexity a-DCT, a search over the $8 \times 8$ matrix space is proposed in order to find candidate matrices that possess low computation cost. The cost of a transformation matrix can be defined as the number of arithmetic operations required for its computation. One way to guarantee good candidates is to restrict the search to matrices whose entries do not require multiplication operations. Thus the following optimization problem arises:

$$T^* = \arg \min_T \text{cost}(T),$$ (6.2)

where $T^*$ is the sought matrix and cost$(T)$ returns the arithmetic complexity of $T$.

Additionally, the following constraints were adopted:

1. Elements of matrix $T$ must be in $\{0, \pm 1, \pm 2\}$ to ensure that resulting multiplicative complexity is null;

2. The following form for matrix $T$ is imposed:

$$T = \begin{bmatrix}
a_3 & a_3 & a_3 & a_3 & a_3 & a_3 & a_3 & a_3 \\
a_0 & a_2 & a_4 & a_6 & -a_6 & -a_4 & -a_2 & -a_0 \\
a_1 & a_5 & -a_5 & -a_1 & -a_5 & a_5 & a_1 & a_3 \\
a_2 & -a_6 & -a_0 & -a_4 & a_4 & a_0 & a_6 & -a_2 \\
a_3 & -a_3 & -a_3 & a_3 & -a_3 & a_3 & -a_3 & a_3 \\
a_4 & -a_0 & a_6 & a_2 & -a_2 & -a_6 & a_0 & -a_4 \\
a_5 & -a_1 & a_1 & -a_5 & a_5 & -a_1 & -a_1 & a_5 \\
a_6 & -a_4 & a_2 & -a_0 & a_0 & -a_2 & a_4 & -a_6
\end{bmatrix},$$

where $a_i \in \{0, 1, 2\}$, for $i = 0, 1 \ldots, 6;$

68
3. All rows of $\mathbf{T}$ are non-null;

4. Matrix $\mathbf{T} \cdot \mathbf{T}^\top$ must be a diagonal matrix to ensure orthogonality of the resulting approximation [182].

Constraint 2) is required to preserve the DCT-like matrix structure. Recalling that the exact 8-point DCT matrix is given by [165]:

$$
\mathbf{C} = \frac{1}{2} \cdot \begin{bmatrix}
\gamma_0 & \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 & \gamma_5 & \gamma_6 & \gamma_7 \\
\gamma_0 & \gamma_1 & \gamma_2 & -\gamma_6 & -\gamma_4 & -\gamma_2 & -\gamma_0 & 0 \\
\gamma_1 & \gamma_2 & -\gamma_6 & -\gamma_4 & -\gamma_2 & -\gamma_0 & 0 & 0 \\
\gamma_2 & -\gamma_6 & -\gamma_4 & -\gamma_2 & -\gamma_0 & 0 & 0 & 0 \\
\gamma_3 & \gamma_1 & \gamma_2 & -\gamma_6 & -\gamma_4 & -\gamma_2 & -\gamma_0 & 0 \\
\gamma_4 & \gamma_5 & \gamma_6 & \gamma_2 & -\gamma_6 & -\gamma_4 & -\gamma_2 & -\gamma_0 \\
\gamma_5 & \gamma_6 & \gamma_2 & -\gamma_6 & -\gamma_4 & -\gamma_2 & -\gamma_0 & 0 \\
\gamma_6 & -\gamma_4 & \gamma_2 & -\gamma_6 & -\gamma_4 & -\gamma_2 & -\gamma_0 & 0 \\
\end{bmatrix},
$$

where $\gamma_k = \cos(2\pi(k+1)/32)$, $k = 0, 1, \ldots, 6$.

Above optimization problem is algebraically intractable. Therefore we resorted to exhaustive computational search. As a result, eight candidate matrices were found, including the transform matrix proposed in [62]. Among these minimal cost matrices, we separated the matrix that presents the best performance in terms of image quality of compressed images according the JPEG-like technique employed in [52, 62, 171–174, 183, 184].

An important parameter in the image compression routine is the number of retained coefficients in the transform domain. In several applications, the number of retained coefficients is very low. For instance, considering $8 \times 8$ image blocks, (i) in image compression using support vector machine, only the first 8–16 coefficients were considered [185]; (ii) Mandyam et al. proposed a method for image reconstruction based on only three coefficients; and Bouguezel et al. employed only 10 DCT coefficients when assessing image compression methods [172, 183]. Retaining a very
small number of coefficients is common for other image block sizes. In high speed face recognition applications, Pan et al. demonstrated that just 0.34%–24.26% out of 92×112 DCT coefficients are sufficient [186, 187]. Therefore, as a compromise, the number of retained coefficients equal to 10 is adopted, as suggested in the experiments by Bouguezel et al. [172, 183].

The solution of (6.2) is the following DCT approximation:

\[ C^* = D^* \cdot T^* = D^* \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \end{bmatrix}, \]

where \( D^* = \text{diag}\left( \frac{1}{\sqrt{8}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{8}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{8}}, \frac{1}{\sqrt{2}}, \frac{1}{2}, \frac{1}{2} \right) \). Matrix \( T^* \) has entries in \( \{0, \pm 1\} \) and it can be given a sparse factorization according to:

\[ T^* = P \cdot A_3 \cdot A_2 \cdot A_1, \]

where

\[ A_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \]

and

\[ P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \]

The signal flow graph of the 8-point DCT approximation is shown on fig 6.1.

6.3.2 16-point DCT Approximation

Several fast algorithm for the DCT allow recursive structures, for which the computation of the \( N \)-point DCT can be split into the computation of \( \frac{N}{2} \)-point DCT [20, 23, 54, 100, 164, 188]. This is usually the case for algorithms based on decimation-in-frequency methods [100, 103].
In account of the above observation and judiciously considering permutations and signal changes, Dr. Cintra and his team designed a 16-point transformation that splits itself into two instantiations of the low-complexity matrix associated to the 8-point RDCT [174]. The proposed transformation, denoted as $T$, is given by:

$$
T = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & -1 & -1 \\
1 & 0 & 0 & -1 & -1 & 0 & 1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\
1 & 0 & -1 & -1 & 1 & 1 & 0 & -1 & -1 & 0 & -1 & -1 & 1 & -1 & 0 & 1 \\
0 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 \\
1 & -1 & 1 & 1 & 1 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & -1 & -1 & 1 & -1 \\
1 & -1 & -1 & 1 & 1 & 1 & 0 & -1 & -1 & 0 & -1 & -1 & 1 & 1 & -1 & 1 \\
0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & -1 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 1 & -1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & -1 \\
1 & -1 & 1 & -1 & -1 & 1 & 0 & -1 & -1 & 0 & -1 & -1 & 1 & -1 & 1 & -1 \\
0 & -1 & 1 & -1 & -1 & 1 & 1 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
0 & -1 & -1 & 1 & 1 & 1 & 1 & 0 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & 1 \\
0 & -1 & -1 & -1 & 1 & -1 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & -1 & 1 & -1 \\
1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & -1 & 1 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & 0 & 0 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & -1 & 0 & 0 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1
\end{bmatrix}
$$

Because the entries of $T$ are in $\{0, \pm 1\} \subset \mathbb{C}$, the proposed matrix is a multiplierless operator. Bit-shifting operations are also unnecessary; only simple additions are re-

Figure 6.1: Signal flow graph for the factorization of $T^*$. Input data $x_i, i = 0, 1, \ldots, 7$, relates to the output $X_k, k = 0, 1, \ldots, 7$. Dotted arrows represent multiplications by $-1$. 

71
quired. Additionally, the above matrix obeys the condition: \( \mathbf{T} \mathbf{T}^\top = \text{[diagonal matrix]} \), where superscript \( \top \) denotes matrix transposition. Thus, the necessary conditions for orthogonalizing it according to the methods described in [182], [174] and [189] are satisfied. Such procedure yields the following orthogonal 16-point DCT approximation matrix:

\[
\hat{\mathbf{C}} = \mathbf{S} \cdot \mathbf{T},
\]

where

\[
\mathbf{S} = \text{diag} \left( \frac{1}{4}, \frac{1}{4}, \frac{1}{\sqrt{12}}, \frac{1}{\sqrt{8}}, \frac{1}{\sqrt{8}}, \frac{1}{\sqrt{12}}, \frac{1}{\sqrt{12}}, \frac{1}{\sqrt{12}}, \frac{1}{4}, \frac{1}{\sqrt{12}}, \frac{1}{\sqrt{12}}, \frac{1}{\sqrt{8}}, \frac{1}{\sqrt{8}}, \frac{1}{\sqrt{12}}, \frac{1}{\sqrt{12}}, \frac{1}{\sqrt{12}} \right).
\]

In the context of image and video compression, the diagonal matrix \( \mathbf{S} \) can be absorbed into the quantization step [52, 168, 171, 173, 174, 189, 190]. Therefore, under these conditions, the complexity of the approximation \( \hat{\mathbf{C}} \) can be equated to the complexity of the low-complexity matrix \( \mathbf{T} \) [138, 190].

Matrix-based fast algorithm design techniques yield a sparse matrix factorization of \( \mathbf{T} \) as given below:

\[
\mathbf{T} = \mathbf{P}_2 \cdot \mathbf{M}_4 \cdot \mathbf{M}_3 \cdot \mathbf{M}_2 \cdot \mathbf{P}_1 \cdot \mathbf{M}_1,
\]
where

\[
M_1 = \begin{bmatrix} I_8 & I_8 \\ I_8 & -I_8 \end{bmatrix}, \quad P_1 = \text{diag} \left( I_9, \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \right),
\]

\[
M_2 = \text{diag} \left( \begin{bmatrix} I_4 & I_4 \\ I_4 & -I_4 \end{bmatrix}, \begin{bmatrix} I_4 & I_4 \\ I_4 & -I_4 \end{bmatrix} \right),
\]

\[
M_3 = \text{diag} \left( \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ -1 & -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ -1 & -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 0 & 1 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & -1 & 1 \\ 1 & 0 & -1 & 1 \end{bmatrix} \right),
\]

\[
M_4 = \text{diag} \left( \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, I_6, \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, I_6 \right)
\]

and matrix \( P_2 \) performs the simple permutation \((0)(1 \ 8)(2 \ 4 \ 3 \ 11 \ 10 \ 7 \ 12 \ 2)(5 \ 9 \ 13 \ 14 \ 6 \ 5)(15)\) in cyclic notation \([191, p. \ 77]\). Matrices \( I_n \) and \( \bar{I}_n \) denote the identity and counter-identity matrices of order \( n \), respectively. Fig 6.2 shows the signal flow graph for the 16-point DCT approximate algorithm.

6.4 a-DCT: Digital Architectures & Realizations

In this section, hardware architectures for both 8-point and 16-point a-DCT algorithms are detailed. Both 1-D and 2-D transformations are addressed. Introduced architectures were submitted to (i) FPGA realizations and (ii) CMOS 45 nm ASIC synthesis up to place & route level.

The 2-D version of the 8-point and 16-point DCT approximation architectures were realized using two 1-D transforms and a transpose buffer. This is possible because the proposed approximation inherits the separable kernel property of the exact DCT \([192]\). The first instantiation of the a-DCT block furnishes a row-wise transform computation of the input image, while the second implementation furnishes
a column-wise transformation of the previous intermediate result. A real time row-
parallel transposition buffer circuit is required in between the 1-D transformation
blocks. Such block ensures data ordering for converting the row-transformed data
from the first DCT approximation circuit to a transposed format as required by
the second DCT approximation circuit. Both 1-D transformation blocks and the
transposition buffer were initially modeled and tested in Matlab Simulink; then they
were combined to furnish the complete 2-D approximate transform. Fig. 6.4 depicts
the architecture for the proposed 1-D 8-point a-DCT. We emphasize in dashed boxes
the blocks $A_1$, $A_2$ and $A_4$ which correspond to the realization of sparse matrices $A_1$,

![Signal flow graph for the factorization of T](image)

Figure 6.2: Signal flow graph for the factorization of $T$. Input data $x_i$, $i = 0, 1, \ldots, 15$,
relates to the output $X_k$, $k = 0, 1, \ldots, 15$. Dotted arrows represent multiplications
by $-1$. 

74
A_2, and A_3, respectively, while Fig. 6.4 depicts the architecture for the proposed 1-D 16-point a-DCT. We emphasize in dashed boxes the blocks M_1, M_2, M_4, and M_4, which correspond to the realization of sparse matrices M_1, M_2, M_3, and M_4, respectively, as shown in the equation set (6.3). Fig. 6.5 shows the implementation of the 2-D transform by means of the 1-D transforms.

6.4.1 FPGA and ASIC Realizations and Results

The architectures were physically realized on a FPGA based rapid prototyping system for various register sizes and tested using on-chip hardware-in-the-loop co-simulation.

The architectures were designed for digital realization within the MATLAB environment using the Xilinx System Generator. Xilinx Virtex-6 XC6VLX240T-1FFG1156 device was employed to realize the architectures on FPGA with fine-grain pipelining.

Figure 6.3: Architecture of the proposed 8-point DCT approximation.
for increased throughput. The realizations were verified on FPGA chip using a Xilinx ML605 board at a clock frequency of 50 MHz. The realizations were tested with 10,000 random 8-point and 16-point input test vectors using hardware co-simulation. Test vectors were generated from within the MATLAB environment and routed to the physical FPGA device using JTAG based hardware co-simulation. Then measured data from the FPGA was routed back to MATLAB memory space.

Evaluation of hardware complexity and real time performance considered the following metrics: the number of used configurable logic blocks (CLB), flip-flop (FF) count, critical path delay ($T_{cpd}$), and the maximum operating frequency ($F_{max}$)

Figure 6.4: Architecture of the proposed 16-point DCT approximation.
in MHz. The `xflow.results` report file was accessed to obtain the above results. Dynamic ($D_p$) and static power ($Q_p$) consumptions were estimated using the Xilinx XPower Analyzer. Results for the 2-D digital architectures of the 8-point and 16-point are shown in Table 6.1.

For the ASIC realization, the hardware description language code was ported to 45 nm CMOS technology and subject to synthesis and place-and-route steps using the Cadence Encounter. The FreePDK, a free open-source ASIC standard cell library at the 45 nm node, was used for this purpose. The supply voltage of the CMOS realization was fixed at $V_{DD} = 1.1$ V during estimation of power consumption and logic delay. The adopted figures of merit for the ASIC synthesis were: area ($A$) in mm$^2$, area-time complexity ($AT$) in mm$^2$·ns, area-time-squared complexity ($AT^2$) in mm$^2$·ns$^2$, dynamic ($D_p$) power in (mW/MHz) and static ($Q_p$) power consumption.

Figure 6.5: Two-dimensional approximate transform for 8-point and 16-point by means of 1-D approximate transform. Signal $x_{j,0}, x_{j,1}, \ldots$ corresponds to the rows of the input image; $X_{j,0}, X_{j,1}, \ldots$ indicates the transformed rows; $X_{0,k}, X_{1,k}, \ldots$ indicates the columns of the transposed row-wise transformed image; and $X_{0,k}^{(2-D)}, X_{1,k}^{(2-D)}, \ldots$ indicates the columns of the final 2-D transformed image.
in watts, critical path delay \( (T_{cpd}) \) in ns, and maximum operating frequency \( (F_{max}) \) in MHz. Results are shown in Table 6.2.

Table 6.1: Hardware resource consumption using Xilinx Virtex-6 XC6VLX-240T-1FFG1156 device.

<table>
<thead>
<tr>
<th>Transform</th>
<th>CLB</th>
<th>FF</th>
<th>( T_{cpd} ) (ns)</th>
<th>( F_{max} ) (MHz)</th>
<th>( D_p ) (mW/MHz)</th>
<th>( Q_p ) (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-point</td>
<td>381</td>
<td>1056</td>
<td>3.407</td>
<td>293.51</td>
<td>2.31</td>
<td>3.44</td>
</tr>
<tr>
<td>16-point</td>
<td>1408</td>
<td>4600</td>
<td>3.737</td>
<td>267.59</td>
<td>6.98</td>
<td>3.48</td>
</tr>
</tbody>
</table>

Table 6.2: Hardware resource consumption for CMOS 45nm ASIC place-route realization.

<table>
<thead>
<tr>
<th>Transform</th>
<th>Area (mm(^2))</th>
<th>( AT )</th>
<th>( AT^2 )</th>
<th>( T_{cpd} ) (ns)</th>
<th>( F_{max} ) (MHz)</th>
<th>( D_p ) (mW/MHz)</th>
<th>( Q_p ) (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-point</td>
<td>0.128</td>
<td>0.673</td>
<td>3.54</td>
<td>5.263</td>
<td>190</td>
<td>0.093</td>
<td>78.7</td>
</tr>
<tr>
<td>16-point</td>
<td>0.585</td>
<td>4.896</td>
<td>40.98</td>
<td>8.37</td>
<td>119.47</td>
<td>0.368</td>
<td>143.6</td>
</tr>
</tbody>
</table>
Table 6.3: Hardware resource consumption comparison of the 2-D 8-point approximations using Xilinx Virtex-6 XC6VLX240T-1FFG1156 device.

<table>
<thead>
<tr>
<th>Transform</th>
<th>CLB</th>
<th>FF</th>
<th>(T_{cpd}) (ns)</th>
<th>(F_{\text{max}}) (MHz)</th>
<th>(D_p) (mW/MHz)</th>
<th>(Q_p) (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAS-2013</td>
<td>517</td>
<td>1910</td>
<td>3.200</td>
<td>294.98</td>
<td>2.74</td>
<td>3.44</td>
</tr>
<tr>
<td>Proposed</td>
<td>445</td>
<td>1696</td>
<td>3.390</td>
<td>312.50</td>
<td>5.07</td>
<td>3.44</td>
</tr>
</tbody>
</table>

Table 6.4: Hardware resource consumption comparison for CMOS 45 nm ASIC synthesis realization of the 2-D 8-point approximations.

<table>
<thead>
<tr>
<th>Transform</th>
<th>Area (mm(^2))</th>
<th>(AT)</th>
<th>(AT^2)</th>
<th>(T_{cpd}) (ns)</th>
<th>(F_{\text{max}}) (GHz)</th>
<th>(D_p) (mW/MHz)</th>
<th>(Q_p) (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAS-2013</td>
<td>0.057</td>
<td>0.057</td>
<td>0.058</td>
<td>1.008</td>
<td>0.992</td>
<td>0.084</td>
<td>0.357</td>
</tr>
<tr>
<td>Proposed</td>
<td>0.046</td>
<td>0.051</td>
<td>0.057</td>
<td>1.103</td>
<td>0.906</td>
<td>0.084</td>
<td>0.357</td>
</tr>
</tbody>
</table>
Table 6.5: Hardware resource consumption comparison of the 1-D 16-point approximations using Xilinx Virtex-6 XC6VLX240T-1FFG1156 device.

<table>
<thead>
<tr>
<th>Transform</th>
<th>CLB</th>
<th>FF</th>
<th>$T_{cpd}$ (ns)</th>
<th>$F_{max}$ (MHz)</th>
<th>$D_p$ (mW/MHz)</th>
<th>$Q_p$ (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAS-2010</td>
<td>430</td>
<td>1440</td>
<td>1.950</td>
<td>512.82</td>
<td>4.54</td>
<td>3.49</td>
</tr>
<tr>
<td>Proposed</td>
<td>421</td>
<td>1372</td>
<td>1.900</td>
<td>526.31</td>
<td>4.22</td>
<td>3.49</td>
</tr>
</tbody>
</table>

Table 6.6: Hardware resource consumption comparison for CMOS 45nm ASIC synthesis realization of the 1-D 16-point approximations.

<table>
<thead>
<tr>
<th>Transform</th>
<th>Area (mm$^2$)</th>
<th>$AT$</th>
<th>$AT^2$</th>
<th>$T_{cpd}$ (ns)</th>
<th>$F_{max}$ (MHz)</th>
<th>$D_p$ (mW/MHz)</th>
<th>$Q_p$ (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAS-2010</td>
<td>0.169</td>
<td>0.843</td>
<td>4.21</td>
<td>4.994</td>
<td>200.24</td>
<td>0.093</td>
<td>70.47</td>
</tr>
<tr>
<td>Proposed</td>
<td>0.183</td>
<td>0.895</td>
<td>4.38</td>
<td>4.895</td>
<td>204.29</td>
<td>0.095</td>
<td>78.73</td>
</tr>
</tbody>
</table>

Among the considered competitors, the BAS-2013 [59] and BAS-2010 [52] showed arithmetic complexity and coding performance similar to the proposed 8-point and 16-point transforms respectively. For comparison purposes the 1-D versions of the BAS-2010 approximations and the proposed 16-point approximation and the 2-D versions of the BAS-2013 and the proposed 8-point approximation were realized on a Xilinx Virtex-6 XC6VLX240T-1FFG1156 device as well as were ported to
45 nm CMOS technology and subject to synthesis using RTL compiler. The FPGA comparison for the 8-point is shown in Table 6.3 and ASIC comparison is shown in Table 6.4. For the 16-point approximation, the FPGA and ASIC comparisons are shown in Table 6.5 and Table 6.6. Compared to the BAS-2013 and BAS-2010, the proposed 8-point and 16-point transforms are faster when both the FPGA realization and CMOS synthesis is considered while having similar performance in hardware usage and dynamic power consumption.

6.5 a-DCT for HEVC

In order to assess the performance of the DCT approximations in real time video coding, the 8 point and the 16 point DCT approximations were embedded into an open source HEVC [193,194] standard reference software by the Fraunhofer Heinrich Hertz Institute [53]. The original transform prescribed in the selected HEVC reference software is the scaled approximation of Chen DCT algorithm [54] and the software can process image block sizes of $4 \times 4$, $8 \times 8$, $16 \times 16$ and $32 \times 32$. Our methodology consists of replacing the $8 \times 8$ and the $16 \times 16$ DCT algorithms of the reference software by the respective approximate algorithm. The algorithms were evaluated for their effect on the overall performance of the encoding process by obtaining rate-distortion (RD) curves for standard video sequences. The quantization point (QP) was varied from 0 to 50 to obtain the curves and the resulting PSNR values along with the bits/frame values were recorded for both approximation algorithm and Chen DCT algorithm which was already implemented in the reference software.
Fig. 6.6 depicts the obtained RD curves for the ‘BasketballPass’ test sequence.

Fig. 6.8 shows particular $416 \times 240$ frames for the test video sequence ‘BasketballPass’ with $QP \in \{0, 32, 50\}$ when the 8-point a-DCT, 16-point a-DCT, and the Chen DCT are considered.

![RD curves for 'BasketballPass' test sequence.](image)

Figure 6.6: RD curves for ‘BasketballPass’ test sequence.
Figure 6.7: RD curves for ‘BasketballPass’ test sequence.
To compare the performance of the proposed 16-point a-DCT with BAS-2010 [52] algorithm, our methodology consists of replacing the $16 \times 16$ DCT algorithm of the reference software by the proposed 16-point and BAS-2010 algorithms. Algorithms
were evaluated for their effect on the overall performance of the encoding process. For such, we obtained rate-distortion (RD) curves for standard video sequences [195]. The quantization point (QP) varied from 0 to 50 to obtain the curves and the resulting PSNR values with the bit rate values measured in bits per frame were recorded for the proposed 16-point algorithm, the Chen DCT algorithm, and the BAS-2010 [52] algorithm. Fig. 6.7 depicts the obtained RD curves for the ‘BasketballPass’ test sequence. The RD curves reveal that the difference between the BAS-2010 and the implementation of the proposed approximation is negligible.
CHAPTER VII

A MULTIPLIERLESS PRUNED DCT-LIKE TRANSFORMATION FOR IMAGE AND VIDEO COMPRESSION

Recently, low-complexity DCT approximations have been considered for image and video processing [25, 59, 62, 157, 171, 172, 174, 189, 196, 197]. Such approximate transforms can offer meaningful DCT estimations at the expense of small errors. Such trade-off is often acceptable leading to low-power, high-speed digital hardware realizations [157], while ensuring adequate numerical accuracy.

Furthermore, it is a well-known fact that in many DCT applications, the most useful signal information tends to be concentrated in the low-frequency coefficients [23, 198, 199]. This is because the DCT presents good energy compaction properties, which are closely related to the Karhunen-Loève transform [20, 21]. Therefore, only the low-frequency DCT components are necessary to be computed in these applications. A typical example of this situation occurs in data compression applications [23], where high-frequency components are often zeroed by the quantization process [200, p. 586]. Then, only the quantities that are likely to be significant should be computed [201]. This approach is called frequency-domain pruning and has been employed for computing the DFT [202–206]. Such methodology was originally applied in the DCT context in [207] and [208]. In [198, 209], the two-dimensional (2-D)
version of the pruned DCT was proposed. In the context of low-powered wireless vision sensor networks, a pruned DCT was proposed in [210] based on the binary DCT [59].

In [211], Meher et al. proposed a HEVC architecture where the wordlength was maintained fixed by means of discarding least significant bits. In that context, the goal was the minimization of the arithmetic complexity at the expense of wordlength truncation. Such approach was also termed ‘pruning’. However, it is fundamentally different from the approach discussed in the current chapter. This terminology distinction is worth observing.

Thus, in response to the growing need for high compression of image and moving pictures for various applications [212], we propose a further reduction of the computational cost of the 8-point DCT computation in the context of JPEG-like compression and HEVC processing. In this work, a pruned DCT approximations for image and video compression is introduced. Essentially, DCT-like pruning consists of extracting from a given a-DCT matrix a submatrix that aims at furnishing similar mathematical properties.

7.1 Pruned a-DCT: Mathematical Definition

Essentially, DCT pruning consists of extracting from the 8×8 DCT matrix \( C \) a submatrix that aims at furnishing similar mathematical properties as \( C \). Pruning is often realized on the transform-domain by means of computing fewer transform coefficients than prescribed by the full transformation. Usually, only the \( K < N \) coefficients that
retain more energy are preserved. For the DCT, this corresponds to the first $K$ rows of the DCT matrix. Therefore, this particular type of pruning implies the following $K \times 8$ matrix:

\[
C_K = \begin{bmatrix}
c_{0,0} & c_{0,1} & \cdots & c_{0,7} \\
c_{1,0} & c_{1,1} & \cdots & c_{1,7} \\
\vdots & \vdots & \ddots & \vdots \\
c_{K-1,0} & c_{K-1,1} & \cdots & c_{K-1,7}
\end{bmatrix},
\]

\tag{7.1}

where $0 < K \leq 8$ and $c_{m,n}$, $m, n = 0, 1, \ldots, 7$, are the entries of $C$. The case $K = 8$ corresponds to the original transformation. Such procedure was proposed in [198,209] for the DCT in the context of wireless sensor networks. For the 2-D case, we have that the pruned DCT is given by: $\hat{B} = C_K \cdot A \cdot C_K^T$. Notice that $\hat{B}$ is a $K \times K$ matrix over the transform-domain. Lecuire et al. [209] showed that retaining the transform-domain coefficients in a $K \times K$ square pattern at the upper-right corner leads to a better energy-distortion trade-off when compared to the alternative triangle pattern [198]. The pruning approach can be applied to DCT approximations. By discarding the lower rows of the low-complexity matrix $T$, we obtain the following $K \times N$ pruned matrix transformation:

\[
T_K = \begin{bmatrix}
t_{0,0} & t_{0,1} & \cdots & t_{0,7} \\
t_{1,0} & t_{1,1} & \cdots & t_{1,7} \\
\vdots & \vdots & \ddots & \vdots \\
t_{K-1,0} & t_{K-1,1} & \cdots & t_{K-1,7}
\end{bmatrix},
\]

\tag{7.2}
Considering the orthogonalization method described in [189], the $K \times 8$ pruned a-DCT is given by:

$$\hat{C}_K = S_K \cdot T_K,$$  \hspace{1cm} (7.3)

where $S_K = \sqrt{\text{diag}\{(T_K \cdot T_K^\top)^{-1}\}}$ is a $K \times K$ diagonal matrix and $\text{diag}(\cdot)$ returns a diagonal matrix with the diagonal elements of its argument. If $T$ is orthogonal, then $T_K$ satisfies semi-orthogonality [213, p. 84]. The 2-D pruned DCT of a matrix $A$ is given by

$$\tilde{B} = T_K \cdot A \cdot T_K^\top.$$  \hspace{1cm} (7.4)

Resulting transform-domain matrix $\tilde{B}$ is sized $K \times K$.

7.1.1 Arithmetic Complexity

Because all considered a-DCT are natively multiplierless operators, the pruned DCT approximation inherits such property. Therefore, the arithmetic complexity of the pruned approximations is simply given by the number of additions and bit-shifting operations required by their respective fast algorithms. To illustrate the complexity assessment, we focus on the a-DCT [62], whose fast algorithm signal flow graph (SFG) is shown in Figure 7.1(a). The full computation of the a-DCT requires 14 additions. By judiciously considering the computational cost of only the first $K$ transform-domain components, we derived fast algorithms for the pruned a-DCT matrices as shown in Figure 7.1.
Figure 7.1: Signal flow graph for the a-DCT matrix and pruned a-DCT matrices [1,2].
7.2 Pruned a-DCT: Digital Architectures & Realizations

The proposed pruned a-DCT was realized in a separable 2-D block transform using two 1-D transform blocks with a transpose buffer between them. Such blocks were designed and simulated, using bit-true cycle-accurate modeling, in Matlab/Simulink. Thereafter, the proposed architecture was ported to Xilinx Virtex-6 FPGA as well as to custom CMOS standard-cell integrated circuit (IC) design. The transform was applied in a row-parallel fashion to the blocks of data and all blocks were $8 \times 8$, irrespective of pruning. When $K$ decreases, the number of null elements in the blocks increases. The row-transformed data were subject to transposition and then the same pruned algorithm was applied, albeit for column direction.

7.2.1 FPGA Rapid Prototypes

The pruned architectures were physically realized on a Xilinx Virtex-6 XC6VLX240T-1FFG1156 FPGA device with fine-grain pipelining for increased throughput. The FPGA realizations were verified using hardware-in-the-loop testing, which was achieved through a JTAG interface. Proposed approximations were verified using more than 10000 test vectors with complete agreement with theoretical values. Evaluation of hardware complexity and real-time performance considered the following metrics: the number of employed configurable logic blocks (CLB), flip-flop (FF) count, critical path delay ($T_{cpd}$), and the maximum operating frequency ($F_{max}$) in MHz. The xflow.results report file, from the Xilinx FPGA tool flow, led to the reported results. Frequency normalized dynamic power ($D_p$, in mW/MHz) and static power
\( (Q_p, \text{ in } W) \) consumption were estimated using the Xilinx XPower Analyzer software tool. Above measurements are shown in Table 7.1 for the proposed pruned a-DCT (highlighted in green) and the pruned BAS-2013 introduced in [210]. The transform in [210] is the only pruned a-DCT competitor in literature.

7.2.2 ASIC Synthesis

For the ASIC synthesis, the hardware description language code from the Xilinx System Generator FPGA design flow was ported to 45 nm CMOS technology and subject to synthesis using Cadence Encounter. Standard ASIC cells from the FreePDK, which is a free open-source cell library at the 45 nm node, was used for this purpose. The supply voltage of the CMOS realization was fixed at \( V_{DD} = 1.1 \text{ V} \) during estimation of power consumption and logic delay. The adopted figures of merit for the ASIC synthesis were: area \((A)\) in mm\(^2\), area-time complexity \((AT)\) in mm\(^2\) \(\cdot\) ns, area-time-squared complexity \((AT^2)\) in mm\(^2\) \(\cdot\) ns\(^2\), frequency normalized dynamic power \((D_p, \text{ in mW/MHz})\) and static power \((Q_p, \text{ in mW})\), critical path delay \((T_{cpd})\) in ns, and maximum operating frequency \((F_{max})\) in GHz. ASIC synthesis results for the proposed pruned a-DCT (highlighted in green) and the pruned BAS-2013 algorithm are displayed in Table 7.2.

The FPGA realization of the proposed pruned a-DCT showed a drastic reductions in both area (measured from the number of CLBs) and frequency normalized dynamic power consumption, compared to the full a-DCT. Table 7.3 shows the percentage reduction of area and frequency-normalized dynamic power for both FPGA
implementation and CMOS synthesis for different pruning values. All metrics indicate lower hardware resource consumption when the number of outputs are reduced.

Table 7.1: Resource consumption on Xilinx XC6VLX240T-1FFG1156 device.

<table>
<thead>
<tr>
<th></th>
<th>CLB</th>
<th>FF</th>
<th>$T_{cpd}$</th>
<th>$F_{max}$</th>
<th>$D_p$</th>
<th>$Q_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>107</td>
<td>376</td>
<td>2.263</td>
<td>441.89</td>
<td>0.67</td>
<td>3.43</td>
</tr>
<tr>
<td></td>
<td>107</td>
<td>376</td>
<td>2.263</td>
<td>441.89</td>
<td>0.67</td>
<td>3.43</td>
</tr>
<tr>
<td>2</td>
<td>136</td>
<td>568</td>
<td>2.300</td>
<td>434.78</td>
<td>0.97</td>
<td>3.43</td>
</tr>
<tr>
<td></td>
<td>204</td>
<td>751</td>
<td>2.450</td>
<td>408.10</td>
<td>1.45</td>
<td>3.44</td>
</tr>
<tr>
<td>3</td>
<td>210</td>
<td>783</td>
<td>2.509</td>
<td>398.56</td>
<td>0.87</td>
<td>3.43</td>
</tr>
<tr>
<td></td>
<td>263</td>
<td>978</td>
<td>2.534</td>
<td>394.63</td>
<td>2.11</td>
<td>3.44</td>
</tr>
<tr>
<td>4</td>
<td>247</td>
<td>961</td>
<td>2.946</td>
<td>339.44</td>
<td>1.35</td>
<td>3.43</td>
</tr>
<tr>
<td></td>
<td>339</td>
<td>1216</td>
<td>2.900</td>
<td>344.82</td>
<td>2.50</td>
<td>3.43</td>
</tr>
<tr>
<td>5</td>
<td>290</td>
<td>1123</td>
<td>2.877</td>
<td>347.58</td>
<td>1.70</td>
<td>3.43</td>
</tr>
<tr>
<td></td>
<td>377</td>
<td>1374</td>
<td>2.902</td>
<td>344.58</td>
<td>3.13</td>
<td>3.43</td>
</tr>
<tr>
<td>6</td>
<td>350</td>
<td>1286</td>
<td>2.735</td>
<td>365.63</td>
<td>2.07</td>
<td>3.44</td>
</tr>
<tr>
<td></td>
<td>382</td>
<td>1557</td>
<td>2.784</td>
<td>359.19</td>
<td>3.80</td>
<td>3.44</td>
</tr>
<tr>
<td>7</td>
<td>424</td>
<td>1487</td>
<td>3.300</td>
<td>303.03</td>
<td>2.21</td>
<td>3.44</td>
</tr>
<tr>
<td></td>
<td>445</td>
<td>1720</td>
<td>3.432</td>
<td>291.37</td>
<td>3.87</td>
<td>3.44</td>
</tr>
<tr>
<td>8</td>
<td>445</td>
<td>1696</td>
<td>3.390</td>
<td>294.98</td>
<td>2.74</td>
<td>3.44</td>
</tr>
<tr>
<td></td>
<td>517</td>
<td>1910</td>
<td>3.200</td>
<td>312.5</td>
<td>5.07</td>
<td>3.44</td>
</tr>
</tbody>
</table>

93
from 8 to 1. In particular, for $K = 6$, we notice a power consumption reduction for approximately 20–25%.

Table 7.2: Resource consumption for CMOS 45 nm ASIC synthesis.

<table>
<thead>
<tr>
<th>$K$</th>
<th>Area</th>
<th>AT</th>
<th>$AT^2$</th>
<th>$T_{cpd}$</th>
<th>$F_{max}$</th>
<th>$D_p$</th>
<th>$Q_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.011</td>
<td>0.011</td>
<td>0.010</td>
<td>0.961</td>
<td>1.040</td>
<td>0.018</td>
<td>0.088</td>
</tr>
<tr>
<td></td>
<td>0.011</td>
<td>0.011</td>
<td>0.010</td>
<td>0.961</td>
<td>1.040</td>
<td>0.018</td>
<td>0.088</td>
</tr>
<tr>
<td>2</td>
<td>0.017</td>
<td>0.016</td>
<td>0.015</td>
<td>0.962</td>
<td>1.039</td>
<td>0.028</td>
<td>0.125</td>
</tr>
<tr>
<td></td>
<td>0.022</td>
<td>0.022</td>
<td>0.022</td>
<td>0.995</td>
<td>1.005</td>
<td>0.036</td>
<td>0.167</td>
</tr>
<tr>
<td>3</td>
<td>0.022</td>
<td>0.021</td>
<td>0.020</td>
<td>0.963</td>
<td>1.038</td>
<td>0.038</td>
<td>0.167</td>
</tr>
<tr>
<td></td>
<td>0.029</td>
<td>0.028</td>
<td>0.027</td>
<td>0.981</td>
<td>1.019</td>
<td>0.047</td>
<td>0.213</td>
</tr>
<tr>
<td>4</td>
<td>0.027</td>
<td>0.027</td>
<td>0.026</td>
<td>0.970</td>
<td>1.030</td>
<td>0.047</td>
<td>0.208</td>
</tr>
<tr>
<td></td>
<td>0.037</td>
<td>0.036</td>
<td>0.036</td>
<td>0.997</td>
<td>1.003</td>
<td>0.059</td>
<td>0.269</td>
</tr>
<tr>
<td>5</td>
<td>0.032</td>
<td>0.034</td>
<td>0.037</td>
<td>1.075</td>
<td>0.930</td>
<td>0.057</td>
<td>0.246</td>
</tr>
<tr>
<td></td>
<td>0.041</td>
<td>0.041</td>
<td>0.041</td>
<td>1.007</td>
<td>0.993</td>
<td>0.068</td>
<td>0.303</td>
</tr>
<tr>
<td>6</td>
<td>0.038</td>
<td>0.038</td>
<td>0.037</td>
<td>0.995</td>
<td>1.005</td>
<td>0.067</td>
<td>0.288</td>
</tr>
<tr>
<td></td>
<td>0.046</td>
<td>0.046</td>
<td>0.046</td>
<td>1.008</td>
<td>0.992</td>
<td>0.077</td>
<td>0.340</td>
</tr>
<tr>
<td>7</td>
<td>0.043</td>
<td>0.047</td>
<td>0.051</td>
<td>1.085</td>
<td>0.921</td>
<td>0.079</td>
<td>0.333</td>
</tr>
<tr>
<td></td>
<td>0.051</td>
<td>0.054</td>
<td>0.057</td>
<td>1.050</td>
<td>0.952</td>
<td>0.087</td>
<td>0.378</td>
</tr>
<tr>
<td>8</td>
<td>0.046</td>
<td>0.051</td>
<td>0.057</td>
<td>1.103</td>
<td>0.906</td>
<td>0.084</td>
<td>0.357</td>
</tr>
<tr>
<td></td>
<td>0.057</td>
<td>0.057</td>
<td>0.058</td>
<td>1.008</td>
<td>0.992</td>
<td>0.097</td>
<td>0.424</td>
</tr>
</tbody>
</table>

94
In order to compare the hardware resource consumption of the introduced pruned DCT approximation with competing transforms, the pruned BAS-2013 algorithm [210] was physically realized on the same Xilinx Virtex-6 XC6VLX240T-1FFG1156 device and submitted it to synthesis using ASIC 45 nm CMOS technology. By comparing the results in Table 7.1 and 7.2, it can be seen that the proposed transform discussed here outperforms the pruned BAS-2013 in terms of hardware resource consumption, and power consumption while is in par in terms of speed as well.

Table 7.3: Percentage reduction in area and dynamic power for FPGA.

<table>
<thead>
<tr>
<th></th>
<th>FPGA</th>
<th>ASPIC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area</td>
<td>Dp</td>
</tr>
<tr>
<td>1</td>
<td>71.65</td>
<td>83.11</td>
</tr>
<tr>
<td>2</td>
<td>54.59</td>
<td>72.29</td>
</tr>
<tr>
<td>3</td>
<td>44.88</td>
<td>62.33</td>
</tr>
<tr>
<td>4</td>
<td>30.18</td>
<td>51.94</td>
</tr>
<tr>
<td>5</td>
<td>19.16</td>
<td>34.63</td>
</tr>
<tr>
<td>6</td>
<td>3.14</td>
<td>20.77</td>
</tr>
<tr>
<td>7</td>
<td>1.57</td>
<td>12.12</td>
</tr>
</tbody>
</table>
7.2.3 Hardware Realization of Pruned LODCT

Presenting good performance at image compression and energy compaction, the LODCT [60] is associated to the following transformation matrix:

\[
W = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & -1 & -1 & -1 \\
1 & \frac{1}{2} & -\frac{1}{2} & -1 & -1 & -\frac{1}{2} & \frac{1}{2} & 1 \\
1 & 0 & -1 & -1 & 1 & 1 & 0 & -1 \\
1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 \\
\frac{1}{2} & -1 & 1 & -\frac{1}{2} & -\frac{1}{2} & 1 & -1 & \frac{1}{2} \\
0 & -1 & 1 & -1 & 1 & -1 & 1 & 0
\end{bmatrix}.
\] (7.5)

The implied approximation matrix is furnished by \( \hat{C} = S \cdot W \), where

\[
S = \text{diag}\left(\frac{1}{\sqrt{8}}, \frac{1}{\sqrt{6}}, \frac{1}{\sqrt{6}}, \frac{1}{\sqrt{8}}, \frac{1}{\sqrt{6}}, \frac{1}{\sqrt{8}}, \frac{1}{\sqrt{6}}\right).
\]

The fast algorithm for \( W \) requires 24 additions and 2 bit-shift operations [60].

By analyzing fifty 512 × 512 standard images from a public bank [214], we noticed that the LODCT can concentrate \( \approx 98.98\% \) of the average total image energy in the transform-domain upper-left square of size \( K = 4 \). Thus, computational savings can be achieved by computing only the 16 lower frequency coefficients. Above facts
lead to the following pruned transformation based on the LODCT:

\[
W_{(4)} = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & -1 & -1 & -1 \\
1 & \frac{1}{2} & -\frac{1}{2} & -1 & -1 & -\frac{1}{2} & \frac{1}{2} & 1 \\
1 & 0 & -1 & -1 & 1 & 1 & 0 & -1
\end{bmatrix}.
\] (7.6)

The corresponding pruned approximation is: \( \hat{C}_{(4)} = S_{(4)} \cdot W_{(4)} \), where \( S_{(4)} = \text{diag} \left( \frac{1}{\sqrt{8}}, \frac{1}{\sqrt{2}}, \frac{1}{2}, \frac{1}{\sqrt{2}} \right) \). A fast algorithm flow graph for the pruned 1-D LODCT is shown in Figure 7.2, requiring only 18 additions and 1 bit-shift operation.

7.2.4 BEE3 Realization

The 1-D version of the pruned LODCT and the a-DCT where \( k = 6 \) were initially modeled and tested in Matlab Simulink. Figure 7.3 and 7.4 depict the resulting architectures for the pruned LODCT and pruned a-DCT where \( k = 6 \), respectively.

![Figure 7.2: Fast algorithm for pruned LODCT.](image-url)
The architectures were realized on BEE3 [215], a multi-FPGA based rapid prototyping system and was tested using on-chip hardware-in-the-loop co-simulation. Such a device was employed to physically realize the above architectures with fine-grain pipelining for increased throughput. FPGA realizations were tested with 10,000 random 8-point input test vectors. Test vectors were generated from the MATLAB environment and routed to the BEE3 device through the USB ports and then measured data from the BEE3 device was routed back to MATLAB memory space.

![1-D architecture of the proposed 8-point pruned LODCT.](image1.png)

Figure 7.3: 1-D architecture of the proposed 8-point pruned LODCT.

![1-D architecture of the proposed 8-point pruned a-DCT.](image2.png)

Figure 7.4: 1-D architecture of the proposed 8-point pruned a-DCT.
Evaluation of hardware complexity and real time performance considered the following metrics: the number of used configurable logic blocks (CLB), flip-flop (FF) count, critical path delay ($T_{cpd}$), and the maximum operating frequency ($F_{max}$) in MHz. The `xflow.results` report file was accessed to obtain the above results. Dynamic ($D_p$) and static power ($Q_p$) consumptions were estimated using the Xilinx XPower Analyzer for the Xilinx Virtex-5 XC5VSX95T-2FF1136 device. Results are shown in Table 7.4.

Table 7.4: Hardware resource consumption using Xilinx Virtex-5 XC5VSX-95T-2FF1136 device.

<table>
<thead>
<tr>
<th>Hardware metric</th>
<th>Method</th>
<th>Pruned LODCT</th>
<th>Pruned a-DCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLB</td>
<td></td>
<td>298</td>
<td>232</td>
</tr>
<tr>
<td>FF</td>
<td></td>
<td>1054</td>
<td>895</td>
</tr>
<tr>
<td>$T_{cpd}$ (ns)</td>
<td></td>
<td>3.578</td>
<td>3.588</td>
</tr>
<tr>
<td>$F_{max}$ (MHz)</td>
<td></td>
<td>279.48</td>
<td>278.70</td>
</tr>
<tr>
<td>$D_p$ (mW/MHz)</td>
<td></td>
<td>3.141</td>
<td>3.620</td>
</tr>
<tr>
<td>$Q_p$ (W)</td>
<td></td>
<td>1.50</td>
<td>1.50</td>
</tr>
</tbody>
</table>
7.3 HEVC Software Simulation

The original transform included in the HEVC reference software is a scaled approximation of Chen’s DCT algorithm. Our methodology consists of replacing the 8×8 DCT algorithm of the reference software by the a-DCT and the proposed pruned a-DCT where \( k = 4 \). Fig. 7.5 shows three 416×240 frames of the ‘BasketballPass’ test sequence [216] obtained from the HEVC simulation. Resulting frames were coded using the Chen’s DCT algorithm (Fig. 7.5(a)), the modified RDCT (Fig. 7.5(b)), and the proposed pruned a-DCT \((k = 4)\) (Fig. 7.5(c)). The PSNR values for these three frames are shown in Fig. 7.5. The pruned approximation effected minimal image degradation—less than 0.25 dB. On the other hand, arithmetic complexity of the 8-point DCT was significantly reduced—76.2% less arithmetic operations when compared with the original Chen’s DCT algorithm. We have also computed rate distortion (RD) curves for both a-DCT and the proposed pruned a-DCT using standard video sequences [216]. For such, we varied the quantization point (QP) from 0 to 50 and computed the PSNR of the proposed pruned approximate with reference to the RDCT along with the bits/frame of the encoded video. As a result, we obtained the curves shown in Fig. 7.6. The PSNR values related to QP are shown in Figure 7.7. The difference in the rate points between the RDCT and the proposed pruned approximation is less than 0.57 dB, which is smaller than 1.3%.
Figure 7.5: Selected frame from ‘BasketballPass’ test video coded by means of (a) the Chen’s DCT algorithm (PSNR 37.62 dB), (b) the modified RDCT (PSNR 37.42 dB), and (c) the proposed pruned a-DCT ($k = 4$), with 76.2% less arithmetic operations then Chen’s DCT (PSNR 37.41 dB).

Figure 7.6: RD curves for ‘BasketballPass’ test sequence.
Figure 7.7: PSNR values related to QP.
Numerous efficient algorithms for the 8-point DCT computation have been proposed [20]. A particularly successful method is the Loeffler DCT algorithm [164] which is capable of computing the 8-point at minimal multiplicative cost [103,164,217] and is often regarded as a reference method for comparison and analysis of 8-point DCT algorithms. In [218,219], Cozzens and Finkelstein proposed a method based on algebraic integer (AI) representation for the computation of the $2^n$-point DFT. Algebraic integer representation has the distinction of being capable of exact computation, numerical calculation without error propagation, and arbitrary precision [63]. Besides the DFT, AI methods have been applied for the DCT computation [220–223]. Recently, Shen et al. [61] introduced a two-dimensional algebraic encoding representation for the Loeffler [164]. Shen’s design could not attain an error-free architecture, due to the partial encoding of selected rotational blocks required by the Loeffler DCT.
As a consequence, the resulting design requires a number of multiplications. In this chapter, a fully error-free AI-based scheme for the 8-point DCT computation by means of the Loeffler algorithm is introduced. Moreover, this algorithm aims at a multiplication-free architecture, where only simple additions and bit-shifting operations are necessary. For such, a suitable AI-based representation for the DCT multipliers is sought. Resulting algorithms are designed in a multiplication-free framework. Decoding architectures for mapping AI encoded quantities back to usual fixed point arithmetic are also proposed. For resource comparison purposes, the proposed algorithm and the Shen’s algorithm [61], have been realized in hardware along with proposed decoding architectures.

8.1 Algebraic Integer Representation

In number theory, an algebraic integer is a complex number that is a root of some monic polynomial with coefficients in $\mathbb{Z}$

8.1.1 Mathematical Preliminaries

An algebraic integer is any root of a monic polynomial with integer coefficients [224, p. 178]. In [218,219], Cozzens and Finkelstein proposed the use of a ring of algebraic integers for numerical representation. Such representation was based on the subring of the field of complex numbers generated by the primitive root of the polynomial $\omega^{R/2} + 1 = 0$, where $R$ is a power of two. The implied subring is denoted by $\mathbb{Z}[\omega]$, where $\omega = e^{2\pi j/R}$, and the set \{1, $\omega, \omega^2, \ldots, \omega^{R/2-1}\} forms an integral basis for $\mathbb{Z}[\omega]$ [219,225].
8.1.2  8-point DCT AI Basis

The 8-point DCT is a linear orthogonal transformation given by [20, 226]:

\[
X_k = \frac{1}{2} \sum_{n=0}^{7} \beta_k x_n \cos \left[ \pi \frac{(2n + 1)k}{16} \right],
\]

where \( k = 0, 1, \ldots, 7 \) and \( \beta_0 = 1/\sqrt{2} \) and \( \beta_k = 1 \) for \( k = 1, 2, \ldots, 7 \). Our goal is to characterize a ring suitable for the computation of the 8-point DCT computation. Since the 8-point DCT requires the quantities \( \cos \left[ \pi k \frac{(2n + 1)}{16} \right] \), \( n, k = 0, 1, \ldots, N-1 \) [20], then, for \( R = 32 \), the irreducible polynomial \( \omega^{16} + 1 \) furnishes the necessary algebraic structure. Thus, the ring \( \mathbb{Z} \left[ e^{j\pi/16} \right] \), which spans the following subset of complex numbers is [227]:

\[
\mathbb{Z} \left[ e^{j\pi/16} \right] = \left\{ z \in \mathbb{C} : z = \sum_{k=0}^{15} a_k \cdot \left( e^{j\pi/16} \right)^k, \text{ for } a_k \in \mathbb{Z} \right\}.
\]

However, because the DCT kernel is real, focus is on the purely real elements of \( \mathbb{Z} \left[ e^{j\pi/16} \right] \). Hence, it is imposed that \( \Im \{z\} = \Im \{ \sum_{k=0}^{15} a_k \cdot \sin \left( k\pi/16 \right) \} = 0 \), where \( \Im \{ \cdot \} \) denotes the imaginary part operator. This condition is equivalent to \( a_k + a_{16-k} = 0 \) [228, p. 72]. Therefore, because \( \cos(k\pi/16) = \cos((N - k)\pi/16) \), the purely real elements of \( \mathbb{Z} \left[ e^{j\pi/16} \right] \) can be represented as:

\[
x = a_0 + 2 \cdot \sum_{k=1}^{7} a_k \cdot \cos \left( k \frac{\pi}{16} \right). \tag{8.1}
\]

Thus, a set that spans \( x \) over \( \mathbb{Z} \) is

\[
\mathbb{Z} = \{ 1, c_1, c_2, c_3, c_4, c_5, c_6, c_7 \},
\]

where \( c_k = 2 \cos(k\pi/16), \) for \( k = 1, 2, \ldots, 7 \). Notice that span(\( \mathbb{Z} \)) \( \subset \mathbb{Z} \left[ e^{j\pi/16} \right] \), where span(\( \cdot \)) is the vector space generated by linear combination of the elements of \( \mathbb{Z} \).
8.1.3 Encoding and Decoding

The AI encoding of a given real number \( x \) over a particular basis is denoted by:

\[
f_{\text{enc}}(x; \zeta) = x,
\]

where \( \zeta \) is a vector with the basis elements, \( x = [a_0, a_1, a_2, a_3, a_4, a_5, a_6]^\top \) is a vector of integers representing the encoding of \( x \) over the AI domain, and \( ^\top \) denotes transposition. For the discussed set \( \mathbb{Z} \), we have

\[
\zeta = [1, c_1, c_2, c_3, c_4, c_5, c_6]^\top.
\]  \( (8.2) \)

The decoding operation is furnished by the dot product operation [229]:

\[
f_{\text{dec}}(x; \zeta) = x^\top \cdot \zeta = a_0 + \sum_{i=1}^{7} a_k \cdot c_k = \hat{x},
\]  \( (8.3) \)

It can be shown that the above presentation is dense and can provide arbitrary precision, i.e., it is always possible to determine an integer vector \( x \) such that \( |x - \hat{x}| < \varepsilon \), for any \( \varepsilon > 0 \). For example, the real number \( x = 1 + \sqrt{3} \) can be represented by \( [1, -2, 1, -1, 2, 1, 2, 0]^\top \) or \( [3, 0, 1, 1, -3, -2, 3, 1]^\top \), considering representation errors of \( 7.07 \times 10^{-5} \) and \( 7.10 \times 10^{-6} \), respectively. However, some particular real numbers are well-suited for the proposed basis, allowing error-free representation: \( |x - \hat{x}| = 0 \). For instance, the real number \( x = 1 - \sqrt{2 + \sqrt{2 + \sqrt{2}}} \) has the following error-free representation over the discussed basis:

\[
f_{\text{enc}}(x; \zeta) = [1, -1, 0, 0, 0, 0, 0, 0]^\top.
\]

In fact, by construction, the basis elements and their combinations can be given error-free integer representation. Table 8.1 displays the proposed encoding of the basis elements.
In actual applications, the input data is discrete and usually quantized [69] in the form of an integer [20]. Therefore, input data can be understood as a sequence of integers [100]. In this particular but realistic case, the proposed encoding can be trivially performed. Indeed, any integer quantity $m \in \mathbb{Z}$ can be represented by [229]:

$$f_{\text{enc}}(m; \zeta) = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T.$$  \hfill (8.4)

Thus, under above assumptions, the encoding operation is effortless. In the current work, only trivial encoding is required.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$f_{\text{enc}}(x; \zeta)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$[1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T$</td>
</tr>
<tr>
<td>$c_1$</td>
<td>$[0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$[0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0]^T$</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$[0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]^T$</td>
</tr>
<tr>
<td>$c_4$</td>
<td>$[0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0]^T$</td>
</tr>
<tr>
<td>$c_5$</td>
<td>$[0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0]^T$</td>
</tr>
<tr>
<td>$c_6$</td>
<td>$[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0]^T$</td>
</tr>
<tr>
<td>$c_7$</td>
<td>$[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1]^T$</td>
</tr>
</tbody>
</table>
8.1.4 Arithmetic Operations

In this subsection, the basic arithmetic operations over the proposed AI representation is discussed. Only addition/subtraction and multiplication are analyzed here, which are the only elementary operations required for the DCT evaluation via the Loeffler DCT algorithm. Because the AI representation consists of an array of coefficients linked to a particular basis, the addition/subtraction operations follow usual vector sum rule. Let \( u = \begin{bmatrix} u_0 & u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 \end{bmatrix}^\top \) and
\[ v = \begin{bmatrix} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{bmatrix}^\top \]
be two AI encoded numbers. Thus, \( w = \begin{bmatrix} w_0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 \end{bmatrix}^\top \) = \( u + v \) satisfies \( w_k = u_k + v_k \), where \( k = 0, 1, \ldots, 7 \). Considering the multiplication operation, will restrict our analysis to operations involving encoded basis elements (cf. Table 8.1). In fact, only this type of multiplication is present in the Loeffler algorithm. Thus, considering (8.1) and trigonometric identities [228, p. 72], the results shown in Table 8.2 are established. Notice that all required multiplications are trivial in the sense that only element permutations and additions are required. In terms of hardware realization, these operations can be implemented by wiring and adder blocks.

For quantized real input, the multiplication can be trivially computed. Indeed, the multiplication between an integer \( m \) and a basis element is simply the basis element itself scaled by \( m \). Thus, for instance, it can be shown that
\[
f_{\text{enc}}(m \cdot c_1; \zeta) = m \cdot \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top = \begin{bmatrix} 0 & m & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top.
\]
8.1.5 Representable Interval

Practical considerations require a limited dynamic range for the coefficients \( a_k, k = 1, 2, \ldots, 7 \). Therefore, representable numbers are limited to the elements defined in the following subset:

\[
\left\{ x \in \mathbb{R} : x = a_0 + \sum_{k=1}^{7} a_k \cdot c_k, \quad |a_k| \leq M \right\},
\]

Table 8.2: Quantities required by Loeffler algorithm for 8-point DCT and their respective products by an arbitrary algebraic integer.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( f_{\text{enc}}(x; \zeta) \cdot \mathbf{u} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( [u_0 \ u_1 \ u_2 \ u_3 \ u_4 \ u_5 \ u_6 \ u_7]^\top )</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>( [2u_1 \ u_0 + u_2 \ u_1 + u_3 \ u_2 + u_4 \ u_3 + u_5 \ u_4 + u_6 \ u_5 + u_7 \ u_6]^\top )</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>( [2u_2 \ u_3 \ u_0 + u_4 \ u_1 + u_5 \ u_2 + u_6 \ u_3 + u_7 \ u_4 \ u_5 - u_7]^\top )</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>( [2u_3 \ u_2 + u_4 \ u_1 + u_5 \ u_0 + u_6 \ u_1 + u_7 \ u_2 \ u_3 - u_7 \ u_4 \ u_5 - u_6]^\top )</td>
</tr>
<tr>
<td>( c_4 )</td>
<td>( [2u_4 \ u_3 + u_5 \ u_2 + u_6 \ u_1 + u_7 \ u_0 \ u_1 - u_7 \ u_2 - u_6 \ u_3 - u_5]^\top )</td>
</tr>
<tr>
<td>( c_5 )</td>
<td>( [2u_5 \ u_4 + u_6 \ u_3 + u_7 \ u_2 \ u_1 - u_7 \ u_0 - u_6 \ u_1 - u_5 \ u_2 - u_4]^\top )</td>
</tr>
<tr>
<td>( c_6 )</td>
<td>( [2u_6 \ u_5 + u_7 \ u_4 \ u_3 - u_7 \ u_2 - u_6 \ u_1 - u_5 \ u_0 - u_4 \ u_1 - u_3]^\top )</td>
</tr>
<tr>
<td>( c_7 )</td>
<td>( [2u_7 \ u_6 \ u_5 + u_7 \ u_4 - u_6 \ u_3 - u_5 \ u_2 - u_4 \ u_1 - u_3 \ u_0 - u_2]^\top )</td>
</tr>
</tbody>
</table>
where $M$ is the largest representable integer. It can be shown that the largest representable number in the proposed AI representation is:

$$x_{\text{max}} = \frac{\sin\left(\frac{15\pi}{32}\right)}{\sin\left(\frac{\pi}{32}\right)} \cdot M \approx 10.2 \cdot M.$$ 

Thus, representable numbers are confined to the interval $[-x_{\text{max}}, x_{\text{max}}]$.

### 8.2 AI-based 8-point Loeffler DCT

The original Loeffler DCT algorithm has the 4-stage signal flow graph (SFG) shown in Figure 8.1 [164]. An initial inspection reveals that the Loeffler DCT requires the following six multiplicands: \{c_1, \sqrt{2}c_2, c_3, c_4, c_5, \sqrt{2}c_6, c_7\}. Most of these constants are basis elements of the discussed AI representation as shown in Table 8.1. After cascading the multiplicands through the SFG, a more detailed analysis shows that only six multiplicands are required by the Loeffler algorithm: $c_4 \cdot c_2$, $c_4 \cdot c_6$, $c_4 \cdot c_3$, $c_4 \cdot c_5$, $c_4 \cdot c_1$, and $c_4 \cdot c_7$. Notice that $c_4 = \sqrt{2}$. Either by considering the multiplication rules in Table 8.2 or by employing standard trigonometric rules, we maintain that $c_i \cdot c_k = c_{i+k} + c_{i-k}$, for any $i, k \in \mathbb{Z}$ and $c_i = -c_{16-i}$, for $i = 8, 9, \ldots, 16$. Therefore, the above six multiplicands can be written as additive combinations of the AI basis:

\[
\begin{align*}
    c_4 \cdot c_2 &= c_2 + c_6, & c_4 \cdot c_6 &= c_2 - c_6, \\
    c_4 \cdot c_3 &= c_1 + c_7, & c_4 \cdot c_5 &= c_1 - c_7, \\
    c_4 \cdot c_1 &= c_3 + c_5, & c_4 \cdot c_7 &= c_3 - c_5.
\end{align*}
\]
For instance, we obtain:

\[
\begin{align*}
    f_{\text{enc}}(c_4 \cdot c_2; \zeta) &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}^T, \\
    f_{\text{enc}}(c_4 \cdot c_6; \zeta) &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}^T.
\end{align*}
\]

Thus, all necessary quantities for the Loeffler DCT computation possess simple, multiplierless representations over the introduced representation.

The set of equations (8.5) reveals another relevant fact. When written as sums of basis elements the Loeffler multiplicands do not require the basis element \( c_4 \). This means that the ring implied by the Cozzens-Finkelstein formalism results in an overcomplete basis with more elements than necessary for the DCT computation. Thus, we propose the following reduced AI basis for the Loeffler DCT:

\[
Z' = \{1, c_1, c_2, c_3, c_5, c_6, c_7\} \subset \mathbb{Z}.
\]

The 8-point representation can be maintained for the sake of clarity. However, considering actual implementations only the seven relevant components should be taken into account.

Additionally, the original Loeffler DCT algorithm contains rotation blocks (butterfly sections with multiplications) that require four multiplications via direct computation. This cost can be reduced to three multiplications as shown in [100, 164]. Despite the fact that such multiplication saving scheme is particularly successful for the computation over usual arithmetic, this is not the case for the proposed AI-based representation. Indeed, all multiplications in the rotation blocks are trivial over the AI-based encoding. Therefore, the rotation blocks are directly computed.
8.2.1 Real Quantized Input Data

Considering real quantized input, the AI-encoded data can be represented according to \( x_i \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top, x_i \in \mathbb{Z}, i = 0, 1, \ldots, 7 \). Therefore, all three rotation blocks in the AI-based Loeffler DCT have their input data encoded in the format described in (8.4). In this case, the multiplication rules can be dramatically simplified. For input data \( u_0 \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top \) and \( v_0 \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top \), Figure 8.2(a) depicts the resulting operations. For the multiplication by \( \sqrt{2} \), the input data is always in the format \( \begin{bmatrix} 0 & u_1 & 0 & u_3 & 0 & u_5 & 0 & u_7 \end{bmatrix}^\top \), for \( u_1, u_3, u_5, u_7 \in \mathbb{Z} \). Figure 8.2 details the required multiplication rules.

The application of the proposed encoding scheme into the 8-point Loeffler DCT furnishes the multiplierless algorithm shown in Figure 8.3. The total arithmetic cost of the proposed algorithm consists of 20 additions. In fact, the algorithm output

Figure 8.1: Loeffler algorithm for the 8-point DCT computation, where dashed lines represent product by \(-1\).
is a scaled DCT spectrum with scaling factor of 2. If necessary, the scaling can be removed by simple bit-shifting or it can be embedded in the decoding stage. Thus, such scaling does not contribute to any increase of the computational cost.

Figure 8.2: Multiplication rules required by the Loeffler DCT over AI: (a) rotation block computation and (b) multiplication by $\sqrt{2}$. 
Input: $x_n \in \mathbb{Z}$ for $n = 0, 1, \ldots, 7$

Output: $X_k \in \text{span}(\mathbb{Z})$, for $k = 0, 1, \ldots, 7$

\[ A_0 = x_0 + x_7 \quad A_4 = x_3 - x_4 \]

Additions in Stage 1:
\[ A_1 = x_1 + x_6 \quad A_5 = x_2 - x_5 \]
\[ A_2 = x_2 + x_5 \quad A_6 = x_1 - x_6 \]
\[ A_3 = x_3 + x_4 \quad A_7 = x_0 - x_7 \]

Additions in Stage 2:
\[ B_0 = A_0 + A_3 \quad B_2 = A_1 - A_2 \]
\[ B_1 = A_1 + A_2 \quad B_3 = A_0 - A_3 \]

Additions in Stage 3:
\[ C_0 = B_0 + B_1 \quad C_2 = B_2 + B_3 \]
\[ C_1 = B_0 - B_1 \quad C_3 = B_2 - B_3 \]

Additions in Stage 4:
\[ D_0 = -A_5 + A_6 \quad D_2 = A_4 + A_7 \]
\[ D_1 = A_4 - A_7 \quad D_3 = -A_5 - A_6 \]

Output:
\[ X_0 = [2C_0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T \]
\[ X_1 = [0 \ -D_3 \ 0 \ D_2 \ 0 \ -D_1 \ 0 \ D_0]^T \]
\[ X_2 = [0 \ 0 \ C_2 \ 0 \ 0 \ 0 \ -C_3 \ 0]^T \]
\[ X_3 = [0 \ -D_1 \ 0 \ D_3 \ 0 \ D_0 \ 0 \ D_2]^T \]
\[ X_4 = [2C_1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T \]
\[ X_5 = [0 \ D_2 \ 0 \ -D_0 \ 0 \ D_3 \ 0 \ D_1]^T \]
\[ X_6 = [0 \ 0 \ -C_3 \ 0 \ 0 \ 0 \ -C_2 \ 0]^T \]
\[ X_7 = [0 \ -D_0 \ 0 \ -D_1 \ 0 \ -D_2 \ 0 \ -D_3]^T \]

Figure 8.3: The 8-point DCT algorithm for a real quantized input sequence.
Table 8.3 compares the proposed method with the original Loeffler algorithm [164] and Shen et al. [61] in terms of precision and arithmetic cost over the AI representation.

8.3 Final Reconstruction Step

The AI decoding is mathematically described in (8.3) and it is performed at the final reconstruction step (FRS) block, which maps the AI represented quantities into usual fixed point representation. In this section, we consider two methods for AI decoding: (i) the canonical-signed-digit (CSD) method [230] and (ii) the expansion factor method [20,231]. The CSD method is suitable for exact spectrum computation, whereas the expansion factor method grants a scaled spectrum.

Table 8.3: Multiplicative complexity comparison for 8-point DCT computation.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Precision</th>
<th>Multiplications</th>
<th>Additions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Real</td>
<td>Integer</td>
</tr>
<tr>
<td>Loeffler [164]</td>
<td>Finite</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Shen et al. [61]</td>
<td>Finite</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Proposed</td>
<td>Arbitrary</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
8.3.1 Canonical Signed Digit Method

The CSD representation is capable of representing fixed point numbers according to a combination of powers-of-two with a minimum number of adders [20, 230]. Therefore, the multiplications required in the decoding operation described in (8.3) can be performed with few additions and bit-shifting operations.

Considering 12-bit representation, Table 8.4 shows the CSD representation of the proposed AI basis elements $\zeta$ as in (8.2), where 1 and $\bar{1}$ represent additive and subtractive powers of two, respectively [230]. The basis element $c_4$ was included for completeness; however, it is not necessary for the decoding stage as discussed in Section ??.

Table 8.5 shows the minimum and maximum absolute error of the CSD representation for basis elements at several wordlengths $N$.

8.3.2 Expansion Factor

The expansion factor method consists of finding a real number $\alpha^* > 1$ such that $\alpha^* \cdot \zeta$ is close to an integer vector. This allows decoding by means of integer arithmetic operations according to:

$$\alpha^* \cdot f_{\text{dec}}(x; \zeta) = x^T \cdot (\alpha^* \cdot \zeta),$$

where $\alpha^* \cdot \zeta$ is approximately a vector of integers. On the other hand, the decoded quantities are scaled by $\alpha^*$; thus this method offers the scaled DCT. The expansion factor $\alpha^*$ satisfies the following optimization problem:

$$\alpha^* = \arg \min_{\alpha > 1} \| \alpha \cdot \zeta - \text{round}(\alpha \cdot \zeta) \| .$$  \hspace{1cm} (8.6)
Above problem is non-linear and an algebraic closed-form solution may not be simple. Thus, we resort to exhaustive search methods to numerically solve (8.6). Because the basis component $c_4$ is not required for decoding (cf. Section ??), we solve (8.6) considering the reduced vector $\zeta' = \begin{bmatrix} 1 & c_1 & c_2 & c_3 & c_5 & c_6 & c_7 \end{bmatrix}^\top$ instead of $\zeta$.

For instance, considering the search space $[0, 1023]$ (10-bit wordlength) and

<table>
<thead>
<tr>
<th>Basis element</th>
<th>CSD Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_0$</td>
<td>01.0000000000</td>
</tr>
<tr>
<td>$c_1$</td>
<td>10.0000101000</td>
</tr>
<tr>
<td>$c_2$</td>
<td>10.0010100100</td>
</tr>
<tr>
<td>$c_3$</td>
<td>10.1010101010</td>
</tr>
<tr>
<td>$c_4$</td>
<td>10.1010101000</td>
</tr>
<tr>
<td>$c_5$</td>
<td>01.0010101001</td>
</tr>
<tr>
<td>$c_6$</td>
<td>01.0100100011</td>
</tr>
<tr>
<td>$c_7$</td>
<td>00.1010010001</td>
</tr>
</tbody>
</table>
a step size of $10^{-2}$, we obtain a optimal value of $\alpha^* = 719.86$, leading to

$$341.01 \cdot \zeta = \begin{bmatrix} 341.01 \\ 668.92 \ldots \\ 630.10 \ldots \\ 567.08 \ldots \\ 482.26 \ldots \\ 378.91 \ldots \\ 261.00 \ldots \\ 133.06 \ldots \end{bmatrix} \approx \begin{bmatrix} 341 \\ 669 \\ 630 \\ 567 \\ 482 \\ 379 \\ 261 \\ 133 \end{bmatrix}.$$  

Table 8.5: CSD encoding error.

<table>
<thead>
<tr>
<th>$N$</th>
<th>Error</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$1.52 \cdot 10^{-2}$</td>
<td>$1.11 \cdot 10^{-1}$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$1.52 \cdot 10^{-2}$</td>
<td>$4.86 \cdot 10^{-2}$</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$4.01 \cdot 10^{-3}$</td>
<td>$2.41 \cdot 10^{-2}$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$1.77 \cdot 10^{-3}$</td>
<td>$1.54 \cdot 10^{-2}$</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>$6.33 \cdot 10^{-4}$</td>
<td>$7.55 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$1.03 \cdot 10^{-4}$</td>
<td>$3.65 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>$1.03 \cdot 10^{-4}$</td>
<td>$1.77 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>$1.03 \cdot 10^{-4}$</td>
<td>$8.3 \cdot 10^{-4}$</td>
<td></td>
</tr>
</tbody>
</table>

118
Notice that the element 482, which corresponds to $c_4$, is never used and is not submitted to implementation. Table 8.6 presents optimal expansion factors for several wordlengths $N$ with precision of $10^{-2}$. Minimum and maximum relative errors are also shown.

Table 8.6: Scale factors and maximum/minimum relatives error.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$\alpha^*$</th>
<th>$\zeta - \frac{\text{round}(\alpha^* \cdot \zeta)}{\alpha^*}$</th>
<th>min(·)</th>
<th>max(·)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>25.98</td>
<td>$5.27 \cdot 10^{-4}$</td>
<td>$7.18 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>25.98</td>
<td>$5.27 \cdot 10^{-4}$</td>
<td>$7.18 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>107.10</td>
<td>$7.44 \cdot 10^{-5}$</td>
<td>$2.01 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>107.10</td>
<td>$7.44 \cdot 10^{-5}$</td>
<td>$2.01 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>341.01</td>
<td>$4.84 \cdot 10^{-13}$</td>
<td>$3.06 \cdot 10^{-4}$</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>341.01</td>
<td>$4.84 \cdot 10^{-13}$</td>
<td>$3.06 \cdot 10^{-4}$</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1844.96</td>
<td>$2.13 \cdot 10^{-7}$</td>
<td>$7.38 \cdot 10^{-5}$</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1844.96</td>
<td>$2.13 \cdot 10^{-7}$</td>
<td>$7.38 \cdot 10^{-5}$</td>
<td></td>
</tr>
</tbody>
</table>

8.4 AI Based Loeffler DCT: Digital Architectures & Realizations

The 1-D version of the proposed algorithm was initially modeled and tested in MATLAB Simulink using Xilinx block set. Two separate designs were used to realize both
the CSD and the expansion factor approach. For hardware comparison purposes, the competing AI algorithm proposed in [61] was also realized. All three methods were designed using bit-shifting operations and reinterpretation blocks. Hence no multipliers were needed in implementation of the FRS. Figure 8.4 depicts the architecture for the proposed 1-D Loeffler DCT. Figure 8.5 displays the FRS implementation of hardware using zero multipliers.

Figure 8.4: Digital architecture of the Loeffler algorithm for the 8-point DCT computation.
8.4.1 FPGA Realization and Results

Architectures for the CSD, expansion factor, and Shen’s methods were physically realized on Xilinx Virtex-6 XC6VSX475T-1FF1759 ROACH-2 Board. A single layer of pipelining was included in all architectures for increased throughput. Realizations were designed using 32-bit software registers, both on input and output. The CSD method was physically realized considering the 12-bit representation depicted in Table 8.4; whereas the expansion factor method was implemented considering the discussed 10-bit representation. Shen’s algorithm was implemented using the 16-bit CSD
method as provided in [61]. All three designs were evaluated for hardware complexity and real-time performance using metrics such as configurable logic blocks (CLB) and flip-flop (FF) count, critical path delay ($T_{cpd}$) in ns, and maximum operating frequency ($F_{max}$) in MHz. Values were obtained from the Xilinx FPGA synthesis and place-route tools by accessing the xflow.results report file. Table 8.7 depicts the FPGA hardware resource consumption for the CSD, the expansion factor and Shen’s algorithm designs.

<table>
<thead>
<tr>
<th>FRS Method</th>
<th>CLB</th>
<th>FF</th>
<th>$T_{cpd}$ (ns)</th>
<th>$F_{max}$ (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSD Method</td>
<td>1573</td>
<td>1572</td>
<td>9.958</td>
<td>100.42</td>
</tr>
<tr>
<td>EF Method</td>
<td>1565</td>
<td>1455</td>
<td>9.960</td>
<td>100.40</td>
</tr>
<tr>
<td>Shen’s Method [61]</td>
<td>1573</td>
<td>1890</td>
<td>17.196</td>
<td>58.15</td>
</tr>
</tbody>
</table>

8.4.2 ASIC Synthesis and Results

The ASIC implementation was realized by porting the hardware description language code to 18 nm CMOS technology and was subjected to synthesis using the AMS Encounter Digital Implementation (EDI) libraries. Libraries for the best case scenario for gate voltage 1.8 V were used in getting the synthesis results. The adopted figures of merit for the ASIC synthesis were: area ($A$) in mm$^2$, area-time complexity ($AT$)
Table 8.8: Hardware resource consumption for CMOS 180nm ASIC synthesis.

<table>
<thead>
<tr>
<th>Method</th>
<th>Area (mm²)</th>
<th>$AT$ (ns)</th>
<th>$AT^2$ (ns²)</th>
<th>$T_{cpd}$ (ns)</th>
<th>$F_{max}$ (MHz)</th>
<th>$D_p$ (mW/MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSD Method</td>
<td>0.628</td>
<td>1.879</td>
<td>5.625</td>
<td>2.993</td>
<td>334</td>
<td>1.60</td>
</tr>
<tr>
<td>EF Method</td>
<td>0.776</td>
<td>2.584</td>
<td>8.610</td>
<td>3.310</td>
<td>302</td>
<td>2.18</td>
</tr>
<tr>
<td>Shen’s Method [61]</td>
<td>0.989</td>
<td>4.007</td>
<td>16.238</td>
<td>4.052</td>
<td>246</td>
<td>4.01</td>
</tr>
</tbody>
</table>

in mm² · ns, area-time-squared complexity ($AT^2$) in mm² · ns², dynamic ($D_p$) power consumption in mW/MHz, critical path delay ($T_{cpd}$) in ns, and maximum operating frequency ($F_{max}$) in MHz. Results for all three methods are displayed in Table 8.8.
CHAPTER IX

CONCLUSIONS & FUTURE WORK

A new class of multiplierless hardware algorithms are presented consisting only of arithmetic adder circuits that closely approximates the 8-point and 16-point DFT. By eliminating multiplication operations in the proposed a-DFT, unprecedented improvements in circuit complexity, chip area, and power consumption were achieved. The architectures were physically realized in FPGA and and synthesized in ASIC along with the FFT algorithms that are being heavily used in DSP systems and the results show reductions of 35% in chip-area and 48% in power compared to the radix-2 Cooley-Tukey and 16% in chip-area and 35% in power compared to the Winograd algorithm. Furthermore, simulation results for the 1-D and 2-D antenna array patterns for a receive mode RF multi-beamforming using a ULA and a URA of antennas were provided along with the absolute error when compared to the ideal DFT.

A new orthogonal DCT approximation algorithm is presented. The introduced transform offers low computational cost, outperforming to the best of our knowledge all competing methods as it consists only of arithmetic adder circuits. Moreover, the proposed transform performed well by means of (i) hardware implementation (both in FPGA and ASIC), and (ii) software embedding, we demonstrated the adequacy and efficiency of the proposed method, which is suitable for codec schemes, like the HEVC. The algorithm was implemented in a HEVC reference soft-
ware and the RD curves were compared with the Chen’s DCT algorithm. A set of 8-point pruned DCT approximations derived from state-of-the-art methods. The pruned transform based on the 8-point a-DCT presented the lowest arithmetic complexity and showed competitive performance. Thus, the pruned approximations were digitally implemented using both Xilinx FPGA tools and CMOS 45 nm ASIC technology. Additionally the most common DCT approximation algorithms, the BAS-2013 was also realized in FPGA and ASIC to compare hardware resource consumption and the pruned approximation outperformed both algorithms in Chip area as well as power.

A low complexity digital VLSI architecture for the computation of AI based 8-point Loeffler DCT is presented. AI encoding is used on the Loeffler fast algorithm for DCT computation, and a novel FRS structures based on canonical signed digits and the expansion factor are employed. The digital architectures are realized in both 18 nm CMOS using AMS standard cells and Xilinx Virtex-6 XC6VSX475T FPGA technology using the ROACH-2.

9.1 Future Work

The a-DFT designs were mainly realized in the ROACH-2 FPGA board. The ROACH-2 consists of two ZDok+ interfaces supporting high speed DAC/ADC. During this research multi-beamforming has been used as an application in testing out the proposed a-DFT algorithms but the simulations were limited to Matlab. The next step of this research is to create a complete front-end which consists of antenna arrays,
LNAs, bandpass filters, mixers and low pass filters as illustrated on Fig. 9.1. This can be used to capture signals coming from different directions and then can be send through the 16-ADC channels of the ROACH-2.

Since the ROACH-2 board which was used in this research consists of 16 channel ADC ports, the a-DFT can be realized in the FPGA and then can be connected 

Figure 9.1: RF-front end for receiver for one element in the array. I-Q fed to real and imaginary DFT inputs.
to the ADC ports. CASPER blocks can be used to realize the ADC connectivity and the a-DFT algorithm on the ROACH-2 as shown in Fig. 9.2.

After having a working end to end system where you can get the frequency response for a particular signal, the hardware realization can be changed from 8-point to 16-point and so on. During this research the 8-point a-DFT and the 16-point a-DFT was used primarily but it can be extended for larger a-DFT algorithms as well. Since these a-DFT algorithms can be used in power critical devices as they closely approximates the ideal DFT using less amount of hardware resources, this thesis provide evidence that reductions upto 30% in chip area and power can be achieved in the case of the 16-point a-DFT.

For large Fourier transforms ($N > 128$), the percentage reductions are asymp-
totic to $\gamma = \frac{\beta}{1 + \beta}$ where $\beta$ is the relative chip area of a complex multiplier with reference to an adder. This was measured in Cadence as $\beta \approx 10$, implying $\gamma \approx 0.91$: that is, we can project an order of magnitude reduction in chip area and power using the a-DFT in place of FFTs. So coming up with large approximation algorithms for the DFT (i.e. 64-point, 128-point) show immense promise hence can be considered in the future.

Same principal can be used in the case of the a-DCT as well. As future research a-DCT algorithms for larger $N$ can be realized in hardware as well as in software and their resource consumption as well as coding standards can be compared. HEVC is the new standard when it comes to image processing algorithms and it uses the latest h.265 encoding standard. The software uses the scaled approximation of Chen DCT algorithm and the software can process image block sizes of $4 \times 4$, $8 \times 8$, $16 \times 16$, and $32 \times 32$. So extending the 8-point and 16-point a-DCT algorithms to 32-point is of great importance.
BIBLIOGRAPHY


[38] Kit Eaton. How speeding the "most important algorithm of our life time" could change this modern world, Jan 2012.


[75] Vandermonde matrix.


[80] Berkeley wireless research center. The University of California, Berkeley.

[81] Casper ROACH2 revision 2. The University of California, Berkeley.

[82] Murchison widefield array (MWA) telescope.


[193] Ron Wilson. The tv studio becomes a system.


[214] USC-SIPI Image Database.


APPENDIX

LIST OF ABBREVIATIONS

1-D One-dimensional
2-D Two-dimensional
3-D Three-dimensional
AI Algebraic integer
ADC Analog to digital converter
ASIC Application specific integrated circuit
CMOS Complementary metal oxide semiconductor
DCT Discrete cosine transform
DFT Discrete fourier transform
DOA Direction of arrival
DSP Digital signal processing
FFT Fast fourier transform
FPGA Field programmable gate array
FRS Final reconstruction step
HEVC High efficiency video coding
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTAG</td>
<td>Joint Test Action Group</td>
</tr>
<tr>
<td>LNA</td>
<td>Low-noise amplifier</td>
</tr>
<tr>
<td>RF</td>
<td>Radio frequency</td>
</tr>
<tr>
<td>ROACH</td>
<td>Reconfigurable Open Architecture Computing Hardware</td>
</tr>
<tr>
<td>ULA</td>
<td>Uniform linear array</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very large scale integration</td>
</tr>
</tbody>
</table>