

國立交通大學

電子工程學系電子研究所

博士論文

新式位元層次設計方法及其應用於離散弦轉換

A Novel Bit-level Design Approach and its Application to Discrete
Sinusoidal Transforms

研究生：陳漢臣

指導教授：任建葳
張添烜

中華民國九十五年一月

A Novel Bit-level Design Approach and its Application to Discrete
Sinusoidal Transforms

新式位元層次設計方法及其應用於離散弦轉換

研究生：陳漢臣

Student: Hun-Chen Chen

指導教授：任建葳 博士
張添烜 博士

Advisors: Prof. Chein-Wei Jen
Prof. Tian-Sheuan Chang



A Dissertation

Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in
Electronics Engineering
January 2006
Hsinchu, Taiwan, Republic of China

推薦函

事由：推薦電子研究所博士班研究生陳漢臣提出論文以參加國立交通大學博士班論文口試。

說明：本校電子研究所博士班學生陳漢臣已經完成本校電子研究所所規定之學科課程及論文研究之訓練。有關學科方面，陳君已修畢十八學分(請查閱學籍資料)，通過學科考試。有關論文研究部份，陳君已完成「新式位元層次設計方法及其應用於離散弦轉換」(A Novel Bit-level Design Approach and its Application to Discrete Sinusoidal Transforms)初稿。相關論文發表情形如下：

- [1]. **H. C. Chen**, J. I. Guo, T. S. Chang, and C. W. Jen, "A Memory Efficient Realization of Cyclic Convolution and its Application to Discrete Cosine Transform," *IEEE Tran. Circuits and Systems for Video Technology*, vol. 15, no. 3, pp. 445-453, March 2005.
- [2]. **H. C. Chen**, T. S. Chang, J. I. Guo, and C. W. Jen, "The Long Length DHT Design with a New Hardware Efficient Distributed Arithmetic Approach and Cyclic Preserving Partitioning," *IEICE Tran. Electronics*, Vol. E88-C, No. 5, pp. 1061-1069, May 2005.
- [3]. **H. C. Chen**, J. I. Guo, C. W. Jen, and T. S. Chang, "Distributed Arithmetic Realisation of Cyclic Convolution and Its DFT Application," *IEE Proceedings Circuits, Devices and Systems*, Vol. 152, No. 6, pp. 615-629, December 2005.
- [4]. **H. C. Chen**, T. S. Chang, and C. W. Jen, "A Low Power and Memory Efficient Distributed Arithmetic Design and its DCT Application," *Proc. 2004 IEEE Asia-Pacific Conference on Circuits and Systems*, Tainan Taiwan, pp. 805-808, 2004.
- [5]. **H. C. Chen**, J. I. Guo, and C. W. Jen, "A Memory Efficient Realization of Cyclic Convolution and its Application to Discrete Cosine Transform," *Proc. 2003 IEEE International Symposium on Circuits and Systems*, Bangkok Thailand, pp. IV-33-IV-36, 2003.
- [6]. **H. C. Chen**, J. I. Guo, and C. W. Jen, "A New Group Distributed Arithmetic Design for the One Dimensional Discrete Fourier Transform," *Proc. 2002 IEEE International Symposium on Circuits and Systems*, Arizona USA, pp. I-421-I-424, 2002.
- [7]. **H. C. Chen**, J. I. Guo, and C. W. Jen, "Low Power Module Designs for Video Codec Systems," *Proc. The 10th VLSI Design/CAD Symposium*, Nantou Taiwan, pp. 275-278, 1999.
- [8]. **H. C. Chen**, T. S. Chang, J. I. Guo, and C. W. Jen, "A Power-of-Two Variable Length DFT Processor using Group Distributed Arithmetic for Communication Applications," submitted to *IEEE Tran. Circuits and Systems I* in Dec. 2005.

總言之，陳君具備國立交通大學電子研究所應有之教育及訓練水準，因此推薦陳君參加國立交通大學電子研究所博士班論文口試。

此致

國立交通大學 電子研究所

電子研究所教授_____

任建箴

張添烜

中華民國九十四年十一月



新式位元層次設計方法及其應用於離散弦轉換

研究生：陳漢臣

指導教授：任建葳 博士

張添烜 博士

國立交通大學電子工程學系暨研究所

摘要

離散弦轉換已被廣泛的應用於數位信號處理，諸如：影像處理、數位濾波器及數位通訊等...。於低成本架構設計的研究，文獻中雖已有許多的設計，但都因只考慮到係數的常數特性而未真正有效的著眼於不同演算法中這些係數的數值特性，因此高效能低成本離散弦轉換之架構設計仍有極大的著墨空間。對此，本論文提出了一低記憶體成本之位元層次設計方法並應用於高效能低成本之離散弦轉換架構設計上。

本論文以迴旋疊積離散弦轉換演算法為基礎，同時利用分散式算數將輸入資料分解至位元層次進而去除潛在的冗餘而提出名為群組式分散式算數之低記憶體成本之位元層次設計方法；對於一個 N 點的迴旋疊積運算，所提出之新的分散式算數設計方法僅僅使用了一組遠小於傳統式分散式算數設計方法的記憶體、一組 N 位元之移位暫存器、及 N 個累加器。跟據輸入資料之迴旋特性，我們重新安排了分散式算數架構中記憶體的內容進而消除了原先儲存於記憶體中重複出現之係數和而達到降低硬體成本之目的。與傳統之分散式算數設計比較，所提出之群組式分散式算數設計可使記憶體成本由 $O(2^N)$ 降至 $O(2^{N-\log_2 N})$ ；若考慮額外付出的硬體代價，硬體成本則由 $O(2^N)$ 改善至 $O(2^{N-\log_2 N} + N + 2)$ 。此外，為了使所提出之群組式分散式算數設計方法可應用於長點數之設計，群組式分散式算數之分割問題是在提出一種新的分散式算數方法時必須面對的。對質數點數及非質數點數我們分別結合了 Agarwal-Cooley 及 Pseudocirculant matrix factorization 等分割演算法進行迴旋疊積之分割，這樣的結合使得群組式分散式算數設計方法在低成本長點數的迴旋疊積設計上一併得到了解決方案，也進而提升了此二分割演算法在實際應用上之價值。

在離散弦轉換的實現上，為使能夠更進一步降低硬體成本，在離散傅利葉轉

換的設計中我們進一步利用了其中係數之對稱性，使得在群組式分散式算數之離散傅利葉轉換設計上可再降低一半的記憶體成本。而在離散餘絃轉換的設計中，由於迴旋疊積的不完美，為使群組式分散式算數方法能順利的應用於離散餘絃轉換的設計之中，我們亦利用了離散餘絃轉換中係數之對稱性將原來的迴旋疊積演算法轉換成一完美的迴旋疊積演算法，進而使得一個低成本的群組式分散式算數離散餘絃轉換架構得以實現，這樣的一個處理也使得群組式分散式算數在離散餘絃轉換的實現上亦減少了一半的記憶體成本。與現存的心脈式陣列架構及其他分散式算數架構之離散絃轉換設計比較，所提出之群組式分散式算數架構可節省超過 29% 的延遲時間-硬體成本乘積值。

考慮在通訊系統上的應用，本研究最後嘗試使用所提出之低硬體成本群組式分散式算數設計方法來實現長點數且為可變點數之二的次方長度之離散傅利葉轉換。我們使用 Cooley-Tukey 演算法先對離散傅利葉轉換進行分解，再使用 pseudocirculant matrix factorization 演算法對分解後的離散傅利葉轉迴旋疊積式進行進一步的分割，使得一長點數的問題仍可利用低硬體成本之群組式分散式算數加以實現。所提出之以群組式分散式算數設計為基礎的可變點數離散傅利葉轉換架構可適用於 64/128/256/512/1024/2048/4096 等長度之離散傅利葉轉換。此外，所提出之架構亦適用於任意長度之離散傅利葉轉換實現。與現存的長點數及可變點數 FFT 架構比較，除了潛在延遲較短及高硬體使用率的優點外，在單位產出率下，當長度小於 256 時，本架構可節省超過 9.6% 的硬體成本；因此，所提出的是一個具相當競爭力的硬體架構實現。除了上述有關離散弦轉換的應用外，本論文所提出之設計方法亦適用於任何有關迴旋運算的數位信號處理方面的應用上。

A Novel Bit-level Design Approach and its Application to Discrete Sinusoidal Transforms

Student: Hun-Chen Chen

Advisors: Prof. Chein-Wei Jen

Prof. Tian-Sheuan Chang

Department of Electronics Engineering and Institute of Electronics,
National Chiao-Tung University

Abstract

The Discrete Sinusoidal transform (DSST's) have been widely used in many digital signal processing applications such as image processing, digital filtering, digital communication, and etc. Although many designs of the DSST's have been proposed in the literatures, their designs are still not efficient enough since they exploit only the constant property of the transform coefficients without considering the numerical property of these coefficients in the reformulated algorithms to further optimize the hardware cost. This dissertation proposes a novel bit-level hardware-efficient group distributed arithmetic (GDA) design and its applications for Discrete Sinusoidal transform (DSST's) designs.

In the proposed GDA design approach, first we formulate the algorithm of DSST's into cyclic convolution form in algorithm level. Then we use the distributed arithmetic to decompose the input data into bit-level in architecture level. Thus, the data redundancy due to the cyclic convolution can be efficiently removed within the bit-level input context to facilitate a hardware efficient DA realization. The proposed GDA approach rearranges the contents of DA memory according to its cyclic property such that redundancy of the contents can be eliminated and only a few groups of data are needed. Thus, compared with the conventional DA design, the memory cost of the proposed GDA design can be reduced from $O(2^N)$ to $O(2^{N-\log_2 N})$, and accounting with the necessary overhead, the overall complexity is improved from $O(2^N)$ to $O(2^{N-\log_2 N} + N + 2)$. To further extend its applications to long length designs, we further combine the Agarwal-Cooley algorithm and Pseudocirculant matrix factorization algorithm. This can partition the long length cyclic convolution into short ones while can still maintain its cyclic property, which avoids the non-cyclic problem of direct partitioning. Thus the proposed GDA design can efficiently be

applied to realize each of the shortened cyclic convolution blocks to achieve low hardware cost.

The proposed GDA design approach has been applied successfully to the DFT, DHT and DCT designs. For DFT design, we further combine the symmetrical property of the DFT coefficients with the proposed GDA design approach such that this design requires only half the contents to be stored. This further reduces the memory size by a factor of two. For the DCT design, in addition to the symmetry property of DCT coefficients, we further reformulate the non-cyclic DCT kernel into two perfect cyclic forms such that the DCT can be implemented by the GDA design approach with less hardware of $(N-1)/2$ adders or subtractors, one much small memory module, a $(N-1)/2$ -bit barrel shifter, and $(N-1)/2+1$ accumulators. Compared with the existing systolic array designs and DA-based designs, the realizations of 1-D DFT, DHT, and DCT with the proposed GDA design approach reduce the delay-area product more than 29% according to a 0.35 μm CMOS cell library.

In addition to the prime length design, we also apply the GDA approach to the long length power-of-two DFT design commonly used in the communication system. We combine the proposed hardware efficient GDA approach with the Cooley-Tukey algorithm on DFT decomposition, and pseudocirculant matrix factorization algorithm on cyclic convolution partitioning to facilitate the long- and variable-length DFT design with low hardware cost. The proposed design can be flexibly used to compute the 1-D 64/128/256/512/1024/2048/4096-point DFT by cascading two 1-D short length DFTs and summing up the partitioned short length cyclic convolutions for each stage of the cascaded DFT. Besides, the proposed hardware efficient design approach can also be adopted in the design with the length beyond power of two. Compared with the existing long-length and variable-length FFT design, in addition to the advantages of short latency and high hardware utilization efficiency, under the same throughput rate, the proposed variable-length DFT can be a competitive design, and save the hardware cost more than 9.6% while the transform length is smaller than 256. In summary, the presented GDA-based design approach provides a solution to efficiently implement not only the DSST's but also the DSP applications involving convolution and correlation.

CONTENTS

Chapter 1 Introduction.....	1
1.1 Motivation.....	1
1.2 Current status of DSST's designs	2
1.3 Review of DA-based designs.....	3
1.4 Overview of the proposed design approach.....	4
1.5 Considerations to the DSST's designs	5
1.6 Outline of this dissertation	6
Chapter 2 The Group Distributed Arithmetic (GDA) Design Approach.....	8
2.1 Algorithm point of view.....	8
2.2 Architecture point of view	10
2.2.1 Memory-based Group Distributed Arithmetic design.....	10
2.2.2 Analysis of Barrel shifter	11
2.2.3 Evaluation of hardware cost.....	16
2.3 Consideration of low power design.....	17
2.3.1 Analysis of transition activity	17
2.3.2 Address morphing approach	19
2.3.3 Exploration of dynamic range of the input data.....	20
2.3.4 Low Power Design with pre-computation scheme	24
2.3.5 Evaluation of power cost	28
2.4 Partitioning of cyclic convolution.....	29
2.4.1 Agarwal-Cooley algorithm.....	29
2.4.2 Pseudocirculant matrix factorization algorithm.....	30
2.4.3 Long length cyclic convolution design.....	32
2.4.4 Evaluation of long length cyclic convolution GDA design	37
Chapter 3 GDA-based Design for 1-D DSST's.....	39
3.1 Design of 1-D DFT	39
3.1.1 Cyclic Convolution Formulation	39
3.1.2 CORDIC (CO-ordinate Rotation Digital Computer).....	41

3.1.3 Symmetry exploration of the DFT in cyclic convolution	45
3.1.4 Architecture design and evaluation	49
3.2 Design of 1-D DHT	57
3.2.1 Cyclic Convolution Formulation	57
3.2.2 Numerical stability	59
3.2.3 Symmetry exploration of the DHT in cyclic convolution.....	60
3.2.4 Architecture design and evaluation	64
3.3 Design of 1-D DCT.....	66
3.3.1 Cyclic Convolution Formulation	66
3.3.2 Numerical stability	71
3.3.3 Architecture design and evaluation	72
3.3.4 Chip implementation.....	79
Chapter 4 Long-length DSST's designs	80
4.1 Decomposition of long-length DSST's.....	80
4.1.1 Cooley-Tukey Algorithm.....	80
4.1.2 Prime Factor Algorithm	81
4.1.3 Rader's Algorithm.....	83
4.2 Long length DHT Design and Evaluation.....	85
4.3 Variable-length DFT Design to Communication System Application.....	96
4.3.1 Overview of Communication system.....	96
4.3.2 Hardware Cost Analysis.....	97
4.3.3 GDA-based Variable Length DFT Design and Evaluation.....	102
Chapter 5 Conclusion	118
5.1 Contributions.....	118
5.2 Future Research Directions.....	119
Bibliography	121

List of Figures

Fig. 1.1: Outline of this research.	7
Fig. 2.1: The proposed GDA architecture and the associated memory content arrangement in realizing the cyclic convolution example shown in (2.2).....	10
Fig. 2.2: Multiplexer-based barrel shifter design.....	12
Fig. 2.3: Multiplier-based barrel shifter design.	12
Fig. 2.4: Logarithmic number of multiplexer barrel shifter design.	13
Fig. 2.5: Barrel shifter with N^2 transistors.....	14
Fig. 2.6: Comparison of the four barrel shifters in (a) hardware cost, (b) power consumption, and (c) delay time.	15
Fig. 2.7: The delay-area product comparison in the proposed GDA design and the traditional memory-based DA design with 16-bit data word length.	17
Fig. 2.8: Trend of the improvement of transition probability versus the number of input-data bit.....	19
Fig. 2.9: The description of architecture transformation from DA to GDA.....	20
Fig. 2.10: The test image with the size of 252 * 252 pixels.	21
Fig. 2.11: gray-level of the pixels in the image of Fig. 2.10.	22
Fig. 2.12: histogram of the gray-level distribution in the image of Fig. 2.10.	22
Fig. 2.13: The preprocessed gray-level of the image in Fig. 2.10.	23
Fig. 2.14: Histogram of the preprocessed data used in the example of 7-point DCT design.....	23
Fig. 2.15: The skipped bits in DA computation for the even outputs.....	25
Fig. 2.16: The skipped bits in DA computation for the odd outputs.	26
Fig. 2.17: Power consumption of the GDA-based 1-D DCT designs.	28
Fig. 2.18: The low cost version of BGDA design realizing the cyclic convolution example shown in (2.19).	33
Fig. 2.19: The BGDA design on realizing the cyclic convolution example shown in (2.19) with high performance.	34
Fig. 2.20: The low cost version of GDA realization of the example shown in (2.21).	36
Fig. 2.21: The high performance version of GDA realization of the example shown in (2.21).	37
Fig. 3.1: Realization of CORDIC iterations and scaling iterations.	44
Fig. 3.2: Comparison of (a) area cost and (b) power consumption for the complex multiplications realized with serial multiplier and CORDIC.....	45
Fig. 3.3: Architecture design of the 1-D 11-point DFT with GDA approach.	50

Fig. 3.4: Comparison of the area cost of the existing DFT designs and the proposed GDA design in realizing the 1-D N-point DFT.	53
Fig. 3.5: Comparison of the ACT for the existing designs and the proposed GDA design in realizing the 1-D N-point DFT.....	54
Fig. 3.6: Comparison of the delay-area product for the existing designs and the proposed GDA design in realizing the 1-D N-point DFT.....	54
Fig. 3.7: The architecture of the GDA design realizing the 1-D 11-point DHT.	65
Fig. 3.8: The area reduction of the memory cost when applying the symmetry property of DCT coefficients or not.	69
Fig. 3.9: Block diagram of the proposed pipeline architecture for computing the 1-D N-point DCT.....	73
Fig. 3.10: Design of the preprocessing stage in the 1-D 7-point DCT.	74
Fig. 3.11: Design of the DA processing stage that is used to compute the kernel of $T((3^k)_7)$ in the 1-D 7-point DCT.	74
Fig. 3.12: Design of the post-processing stage in the 1-D 7-point DCT including (a) the post-processing, and (b) the output buffer.	76
Fig. 3.13: The delay-area product of the proposed design and the existing DCT designs [33]-[35][52] in realizing the 1-D DCT.....	78
Fig. 3.14: Layout view of the 1-D 7-point GDA-based DCT design.	79
Fig. 4.1: The GDA-based architecture design for 1-D 29-point DHT example	90
Fig. 4.2: Comparison of the normalized area cost in the realization of 1-D N-point DHT using the proposed design and the existing designs.....	93
Fig. 4.3: Comparison of the normalized cycle time in the realization of 1-D N-point DHT using the proposed design and the existing designs.....	93
Fig. 4.4: Comparison of the normalized area-delay product in the realization of 1-D N-point DHT using the proposed design and the existing designs	94
Fig. 4.5: Average improvement of the normalized area-delay product in the designs of 841-point DHT, 1653-point DHT, and 3249-point DHT using the proposed design approach	95
Fig. 4.6: Transceiver /Receiver architecture in the communication system.....	97
Fig. 4.7: Hardware cost of the original FFT versus the proposed GDA-based DFT	101
Fig. 4.8: Delay-area product of the FFT versus the proposed GDA-based DFT.....	102
Fig. 4.9: Block diagram of the proposed variable-length DFT architecture.....	104
Fig. 4.10: Architecture of 2-D DFT with real input.....	104
Fig. 4.11: Architecture design of the 2-D DFT in cyclic convolution formulation. .	104
Fig. 4.12: Version 1 of the reduced architecture of 2-D DFT in cyclic convolution formulation.	105
Fig. 4.13: Version 2 of the reduced architecture of 2-D DFT in cyclic convolution formulation.	105

Fig. 4.14: Folding of the computation of each eight row blocks in 64-point cyclic convolution.	106
Fig. 4.15: Folding of the computation of each four row blocks in 32-point cyclic convolution.	107
Fig. 4.16: Detail architecture of (a) the row 1-D DFT with input buffer and (b) the column 1-D DFT with output buffer.	109
Fig. 4.17: Detail design of (a) input buffer groups, (b) PISO groups, and (c) output buffer groups in the proposed 1-D DFT architecture.	110
Fig. 4.18: (a) design of the 1-bit three dimensional rotator and the routing for (b) 2-bit BRG in stage 2, (c) 4-bit BRG in stage 3, and (d) 8-bit BRG in stage 4.	112
Fig. 4.19: Detail design of variable-length GDA-based module used for the computation of $T_{ij}()$ in the proposed 1-D DFT architecture.	113
Fig. 4.20: Data-flow of the adder-group tree follows the GDAUs in the proposed variable-length DFT design.	114
Fig. 4.21: Detail design of serial multiplier groups in the proposed 1-D DFT architecture.	115
Fig. 4.22: The transpose memory with the specific address generator.	115

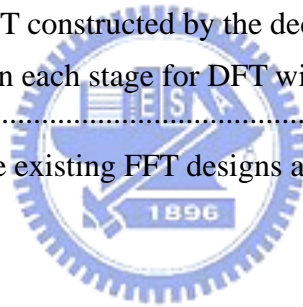




List of Tables

Table 2.1: The rule of group mapping.....	11
Table 2.2: Comparison of memory size in both the traditional memory-based DA and the proposed GDA designs for different values of N.	16
Table 2.3: Transformation of transition probability for the input data of the 4-input data-path.	18
Table 2.4: The relation ship of the address morphing.....	20
Table 2.5: Relationship between the sum of primary inputs and the even outputs.....	25
Table 2.6: Analysis for the covered lengths of cyclic convolution can be decomposed.	30
Table 2.7: Comparison of the hardware cost of the design examples shown in low-cost BGDA, high performance BGDA, and conventional DA in the case of non-coprime partitioning.	38
Table 3.1: Table for θ_i	43
Table 3.2: Determination of the s_i sequence at the θ of 56.	43
Table 3.3: Hardware cost comparison of direct realization and CORDIC realization for a complex multiplication.	45
Table 3.4: The 8 groups of memory content used for computing the 5-point cyclic convolution in GDAUc.....	51
Table 3.5: The 8 groups of memory content used for computing the 5-point cyclic convolution in GDAUs.....	51
Table 3.6: Area cost models to estimate the 1-D N-point DFT modules in the existing systolic array designs, DA-based designs, and the proposed GDA design with real input data.	55
Table 3.7: Area cost models to estimate the 1-D N-point DFT modules with the partitioned cyclic convolution in the existing systolic array designs, DA-based designs, and the proposed BGDA design with real input data.	56
Table 3.8: Average cycle time (ACT) models to estimate the not partitioned and partitioned 1-D N-point DFT modules in the existing systolic array designs, DA-based designs, and the proposed GDA design with real input data.....	57
Table 3.9: The 8 groups of memory content used for computing the 5-point cyclic convolution in GDAUc.....	66
Table 3.10: The 8 groups of memory content used for computing the 5-point cyclic convolution in GDAUs.....	66
Table 3.11: The seed-value, group address, and rotating factor used in the design of group address decoder of 1-D 7-point DCT.	75
Table 3.12: The partial products distribution for different DCT outputs under the same input value.....	75

Table 3.13: 8-word memory contents arranged into groups.	76
Table 3.14: The comparison of the proposed design and the existing DCT designs [33]-[35][52] in realizing the 1-D N-point DCT in terms of delay and silicon area.	78
Table 4.1: Function of the address decoders in the 1-D 29-point DHT design.....	91
Table 4.2: The performance comparison of different designs for computing the 1-D N-point DHT	92
Table 4.3: Length of 1-D DHT constructed by the decomposed short length DHTs..	94
Table 4.4: The evaluation result of GDA-based DHT designs	95
Table 4.5: DFT lengths for several communication systems	97
Table 4.6: The computation complexity of various DFT algorithms.....	98
Table 4.7: The estimation of hardware costs of the FFT and the proposed GDA-DFT	100
Table 4.8: The estimation of hardware costs of the FFT with actual complexity and the proposed GDA-DFT	100
Table 4.9: The estimation of cycle times of the FFT and the proposed GDA-DFT for each sample	101
Table 4.10: Length of 1-D DFT constructed by the decomposed short length DFTs	103
Table 4.11: Condition of BR in each stage for DFT with the lengths of 64, 256, 1024, and 4096.	111
Table 4.12: Comparison of the existing FFT designs and our DFT design	117



Chapter 1

Introduction

In this chapter, we illustrate the motivation, current status of DSST's designs, review of the existing DA-based designs, overview of the proposed memory efficient bit-level design approach, considerations to the DSST's designs, and outline of this dissertation. The details of the proposed design approach and associated advantage as well as the application in DSST's will illustrate in the following chapters.

1.1 Motivation

The Discrete Sinusoidal transforms (DSST's), including discrete Fourier transform (DFT), discrete Hartley transform (DHT), and discrete cosine transform (DCT), have been widely used in many digital signal processing applications such as image processing, digital filtering, digital communication, etc. There are two main solutions for realizing the high complexity of the DSST's in real-time. One is based on the fast algorithms that aim at reducing the complexity of DSST's to speed up the computation. The other is to directly realize the DSST's formulations or their reformulations, such as the convolution, with hardware for accelerating the DSST's computation.

The designs with fast algorithms are attractive for low computational complexity. However, hardware design of the algorithm is communication intensive and computation intensive to complicate the realizations of controller and arithmetic operation. In addition, most of the designs with fast algorithms exploit a butterfly datapath and a global memory in storing all of input/output data as well as the intermediate results. The mass data access from the global memory wastes a large percentage of power in this kind of designs. Besides, the cascaded structure in the fast algorithm makes the designs have poor numerical accuracy such that longer data word length in the datapath is needed. This fact will reduce the low complexity advantages of the fast algorithm and thus increase the hardware cost of the designs with fast algorithm, especially in the design with the length of non-power of two.

On the designs with direct manner, many existing architectures, such as the systolic array, are still severely suffered from large hardware cost because most of the

existing designs use area-consuming multipliers as the fundamental computing elements. Besides, these designs are not efficient enough since they only exploit the constant property of the transform coefficients without considering the possibility on further hardware optimization. Thus, efficient hardware design of the DSST's is still a challenging problem due to its high computational complexity and the requirement of real-time processing.

The other popular architecture based on the distributed arithmetic (DA) has been adopted in DSP applications. In the case of short length, with less hardware cost, the memory-based DA design can instead of area-consuming multiplier for the computation of multiple-in-multiple-out (MIMO) inner product. Thus, trading the required performance, the DA technique shall be a hardware efficient method for the realization with direct manner. Combining with the good feature of DA, we explore the existing DSST's algorithms to develop a hardware efficient DA design approach for real-time realization of the main modules in the multimedia and communication systems.

1.2 Current status of DSST's designs

In this subsection, we will illustrate the current status of DSST's designs with fast algorithms and the direct manner respectively.

For the DFT designs, the designs [1]-[5] exploited the feature of low computation complexity in fast Fourier transform (FFT) algorithms to achieve the goal of reducing the number of computation. However in these design, the global interconnection usually complicates the realization of controller. Since most of the FFT-based designs exploit a butterfly datapath and a global memory in storing all of input/output data as well as the intermediate results, the mass data access from the global memory wastes a large amount of power. Besides, the cascaded structure of FFT algorithm makes these designs have poor numerical accuracy such that longer data word-length in the data-path is needed. This fact will reduce the low complexity advantages of the FFT algorithm and thus increase the hardware cost of the FFT-based designs. On the exploration of hardware solution, the systolic array designs for DFT [6]-[11] were the major trend of realizing DFT in the past decades due to the promising VLSI features of modularity, locality, and regularity. However, these designs are still severely suffered from large hardware cost because most systolic array designs for DFT use

area-consuming multipliers as the fundamental processing elements (PEs).

For the Hartley transform (DHT) designs, since it is a good alternative to the discrete Fourier transform (DFT) for its real-number operations [12][13], the discrete Hartley transform (DHT) [14][15] also plays an important role in many DSP applications. There are many high-speed communication applications [16]-[21] that address the use of dedicated hardware designs for the DHT computation. For instance of the discrete multitone modulation (DMT)-based ADSL transceiver realization, the modulator and demodulator need to respectively compute the DFT and IDFT. The DFT and IDFT computation can be realized effectively by using DHT and IDHT computation for its inherent real-number operations [14]-[15]. The efficiency of using DHT to compute the DFT/FFT becomes more apparent in the applications encapsulating real input data than those encapsulating the complex input data. Many hardware implementations of the DHT have been proposed, including multiplier-based designs [22][23], Coordinate rotation digital computer (CORDIC)-based designs [14]-[29], memory-based designs [30][31], and hardwired multiplier-based design [32]. The design [22] uses a time recursive lattice structure to compute the 1-D DHT. The design [24] uses a fast algorithm to compute 1-D DHT. The designs [23][25]-[29] use direct matrix-vector multiplication algorithm to compute the 1-D DHT. The designs [30][32] use cyclic convolution based matrix-vector multiplication algorithm to compute the 1-D DHT.

For the DCT designs, due to playing a key function in image and signal processing, especially for the demanding multi-media and portable applications, the efficient hardware implementation of DCT is still a challenging problem for the requirements of high computational complexity and real-time processing. To achieve efficient hardware realization, except for the multiplier-based systolic array designs, many researches have been done on realizing the multiplications needed in the DCT through memory. One is the memory-based systolic array design [33] in which the proposed cyclic convolution based architecture possesses the features of simple I/O behavior and removes the data redundancy in the DCT coefficients.

1.3 Review of DA-based designs

To remedy the problems in the DFT, DHT, and DCT realizations with the designs mentioned above, many researches have realized the multiplications needed in

the DSST's through memory [33]-[37]. One of the popular techniques is distributed arithmetic (DA). It has been widely used in many DSP applications such as the DSST's, convolution, and digital filters [34]-[37]. The DA technique is an efficient method for computing inner products by using table look-up, shifting, and accumulations. Therefore, some existing designs are great interests in reducing the memory size required in the implementation of the DA-based architectures [34]-[36], such as the partial sum techniques and the Offset Binary Coding (OBC) techniques [34][35]. Besides, there is a different DA-based design denoted as adder-based DA design that realizes the multiplications by using adders instead of memories [32][38]-[40]. Chang [38]-[39] took advantage of the shared partial sum-of-products and sparse nonzero bits in the fixed input data to reduce the computational complexity. Guo [32][40] exploited the feature of cyclic convolution to simplify the computation of DHT and DFT, so that the multiplications and additions can be realized by using a small number of adders. On the algorithm point of view, these existing designs mentioned above, cyclic convolution-based designs have the good features of simple I/O behavior and reduction of coefficients redundancy in the 1-D DFT, DHT, and DCT. However, since they only exploit the constant property of the transform coefficients without considering the possibility of further hardware optimization with different DSST's algorithms, they are still not efficient enough.

1.4 Overview of the proposed design approach

In this dissertation, we propose a new hardware efficient DA approach for the 1-D DSST's design. The proposed approach can further reduce the memory size required in the traditional DA technique [34]. For a glance of the proposed DA design approach, first we formulate the algorithm of DSST's into cyclic convolution form in algorithm level, and then exploit the distributed arithmetic to decompose the input data into bit-level in architecture level. Thus, the data redundancy due to the cyclic convolution can be efficiently removed within the bit-level input context to facilitate a hardware efficient DA realization.

Observing the cyclic convolution realized by DA technique, we find that different DSST's outputs can be computed using the same DSST's coefficients and the same input data samples with rotated order. If we directly realize the DSST's in cyclic convolution using traditional DA technique, we find that N identical memory modules are used. It reveals a message that the redundancy still exists in the contents of the

memory, which implies that the memory utilization in this case is not good enough. Therefore, we intend to reduce the memory size by re-arranging the memory contents in different way. Combining with the cyclic property, we first group the candidates of DA inputs with rotated order as the same candidate, and then arrange the memory contents in this manner that the partial products for accumulating different DSST's outputs according to the candidates being grouped together, and accessed simultaneously for the different outputs of DSST's. The partial products arranged in a group should be rotated suitably before accumulating. With this way, the memory module contains only few groups of contents and only one memory module, instead of N identical memory modules needed in the computation of 1-D N-point DSST's in conventional DA design. We named this proposed new DA design approach, Group Distributed Arithmetic (GDA).

Because of the inherent issue of DA-based design that the memory size increases exponentially as the length of input data increases, the partition issue must be regarded for long length DA design. In the conventional DA design, we can arbitrarily partition the input data of DA, and then sum up the partial sums from the different memory modules to achieve low hardware cost. Because of the necessity of cyclic preserving, the manner of arbitrarily partitioning cannot be applied to the proposed GDA design. Otherwise, the benefit of low hardware cost in GDA design will not exist. To solve the problems mentioned above, we combine several algorithms to decompose the long length DSST's and partition the DA design in each of the shortened DSST's into smaller ones, which is still preserving the property of cyclic, such that the DSST's can efficiently be realized with GDA design. In the proposed decomposition approach, we decompose the long length DSST's into the short ones with prime factor algorithm (PFA) or Cooley Turkey algorithm, and further partition each of them by using Agarwal-Cooley algorithm [41] or pseudocirculant matrix factorization algorithm (PMFA) [42] such that all the partitioned short DSST's are still composed of the shortened cyclic-convolution blocks. For such long-length computations, dedicated hardware designs can meet both the real-time and low hardware cost requirements in the various high-speed data communication applications.

1.5 Considerations to the DSST's designs

For the DFT design, we further explore the symmetrical property of DFT

coefficients for further reducing the hardware cost of the memory by a factor of two. Compared with the existing systolic array designs and DA-based designs, the DFT design with the proposed GDA design approach can reduce the delay-area product from 29% to 68% according to the 0.35 μm CMOS cell library for short lengths. As compared with the existing designs, the DHT design with the proposed GDA design approach possesses better performance in reducing the area-delay product from 52% to 91%. For the DCT design, due to the rotated input data in the input-data matrix of DCT possess different signs, it is not easy to apply the GDA approach directly to DCT realization. Exploiting the symmetry property of DCT coefficients, we merge the elements in the matrix of DCT kernel, and separate the matrix to two perfect cyclic forms. Then these two smaller perfect cyclic convolution forms can be realized with the proposed GDA approach. This realization facilitates reducing the memory size significantly. As compared with the existing DA-based designs, for an example of 1-D 7-point DCT with 16-bit coefficients; the proposed design can save more than 57% of the delay-area product. Besides, the 1-D DCT chip was implemented to illustrate the efficiency associated with the proposed approach.

As for the popular application of DFT with the length of power of two in the communication system, combining the proposed low cost GDA design with the suggested long-length transform decomposition methodology, a variable-length DFT design has been proposed and implemented in our studies. The proposed design can flexibly be used to compute the 1-D 64/128/256/512/1024/2048/4096-point DFT by cascading two 1-D short length DFTs and summing up the partitioned short length cyclic convolutions for each stage of the cascaded DFT. Besides, the proposed hardware efficient design approach can also be adopted in the design with the length beyond power of two. Compared with the existing long-length and variable-length FFT design [67]-[70], in addition to the advantages of short latency and high hardware utilization efficiency (HUE), the proposed variable-length DFT design can achieve competitive hardware cost under the same throughput rate.

1.6 Outline of this dissertation

The dissertation is organized following the research outline as Fig. 1.1. In chapter 2 we illustrate the proposed GDA design approach for cyclic convolution in detail, including the issue of cyclic convolution partitioning, and its advantages compared with the traditional memory-based DA approach on hardware cost and power

consumption points of view. Chapter 3 illustrates GDA for 1-D DSST's designs, where the optimization on algorithm level for further reducing the hardware cost is involved. Chapter 4 illustrates long-length issues for DSST's design and the proposed variable-length DFT design to communication Application. Finally, we conclude this dissertation in chapter 5, including contributions in this research and some future research directions.

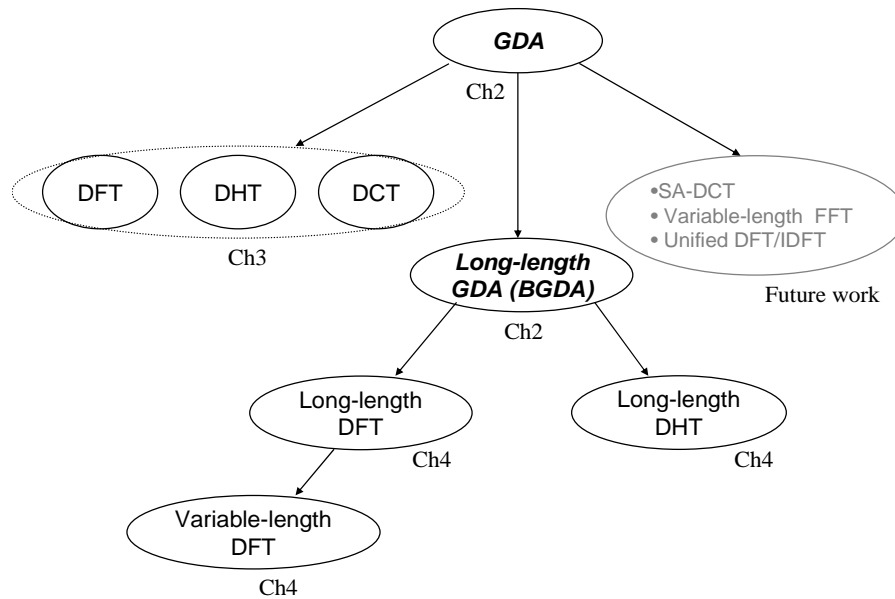


Fig. 1.1: Outline of this research.

Chapter 2

The Group Distributed Arithmetic (GDA) Design Approach

The presented Group Distributed Arithmetic (GDA) design approach mainly consists of cyclic convolution and memory-based DA technique. The algorithm in cyclic convolution can significantly reduce the complexity for the inner product computation with multiple inputs and multiple outputs (MIMO). In the following, we illustrate the proposed GDA design approach from algorithm-level to architecture-level involving the solution of cyclic convolution partitioning for GDA design and the evaluations of hardware cost and power consumption for design with this approach.

2.1 Algorithm point of view

Let us first consider a cyclic convolution example:

$$U = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ d & a & b & c \\ c & d & a & b \\ b & c & d & a \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}, \quad (2.1)$$

where $\{v_1, v_2, v_3, v_4\}$ are input data, $\{a, b, c, d\}$ are coefficients, and $\{u_1, u_2, u_3, u_4\}$ are output data. Using the commutative property of convolution, we can rewrite (2.1) as follow:

$$U = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \\ v_2 & v_3 & v_4 & v_1 \\ v_3 & v_4 & v_1 & v_2 \\ v_4 & v_1 & v_2 & v_3 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}. \quad (2.2)$$

Observing (2.2), we find that different outputs in vector U can be computed using the same input data with rotated order and the same set of coefficients $\{a, b, c, d\}$. According to the DA technique [34], using the same set of coefficients implies that identical memory modules are used to compute all the different outputs. And using the

same inputs with rotated order implies that we can arrange the partial products generated by them as a group and these partial products can be accessed simultaneously in accumulating all the outputs.

For facilitating utilization of the GDA design approach, the general form of GDA shows as

$$\mathbf{u}_k = -\mathbf{u}_{(k-1+R_0)_N+1,0} + \sum_{q=1}^{L-1} \mathbf{u}_{(k-1+R_q)_N+1,q} \cdot 2^{-q}, k = 1, 2, \dots, N \quad (2.3)$$

Where $\bar{\mathbf{u}}_{R_0,0} = R(\bar{\mathbf{u}}_0)$,

$$\bar{\mathbf{u}}_{R_0,0} = \{u_{(0+R_0)_N+1,0}, u_{(1+R_0)_N+1,0}, \dots, u_{(k-1+R_0)_N+1,0}, \dots, u_{(N-1+R_0)_N+1,0}\},$$

$$\bar{\mathbf{u}}_0 = \{u_{1,0}, u_{2,0}, \dots, u_{k,0}, \dots, u_{N,0}\}$$

and

$$\bar{\mathbf{u}}_{R_q,q} = R(\bar{\mathbf{u}}_q),$$

$$\bar{\mathbf{u}}_{R_q,q} = \{u_{(0+R_q)_N+1,q}, u_{(1+R_q)_N+1,q}, \dots, u_{(k-1+R_q)_N+1,q}, \dots, u_{(N-1+R_q)_N+1,q}\},$$

$$\bar{\mathbf{u}}_q = \{u_{1,q}, u_{2,q}, \dots, u_{k,q}, \dots, u_{N,q}\},$$

and

$$u_{k,0} = \sum_{n=1}^N v_{((n-1)+(k-1)+R_0)_N+1,0} \cdot c_n \quad \text{and}$$

$$u_{k,q} = \sum_{n=1}^N v_{((n-1)+(k-1)+R_q)_N+1,q} \cdot c_n .$$

where L denotes the word length of the input data v , N denotes the length of cyclic convolution, R_q denotes the rotating factor for q th bit that is used for indicating the number of position of the partial products in DA input and output should be rotated, and c_n are the coefficients. The rotation function $R(\cdot)$ is used to rotate the elements in

the output vector $\bar{\mathbf{u}}_{R_q,q}$ from the input vector $\bar{\mathbf{v}}_{R_q,q}$ by R_q for the q th bit of DA computation. In the example of 4-point cyclic convolution mentioned above, the coefficient vector $\{c_1, c_2, c_3, c_4\}$ is given as $\{a, b, c, c, d\}$.

2.2 Architecture point of view

2.2.1 Memory-based Group Distributed Arithmetic design

Fig. 2.1 shows the proposed GDA architecture for computing the vector U in (2.2). We arrange the memory contents (16 words) into six groups in this example. The candidate of DA input in the q -th bit, i.e. vector V_q , is first fed into an address decoder to generate the group address V_q' and the corresponding rotating factor R_q according to the rule of group mapping shown in Table 2.1 that performed by the specific address decoder in the proposed GDA design when realizing the cyclic convolution example shown in (2.2). Here, the group address G_q denotes which group the candidate of DA input belongs to. If the candidate is the seed value of a group V_q' , the rotating factor is equal to 00. That means the partial products accessed from the group memory is directly fed into the accumulators for computing the DA outputs without performing any rotation. If the candidate is different from the seed value but belongs to the same group, the rotating factor is the value indicating how many positions the partial products accessed from the group memory should be rotated before entering the accumulators.

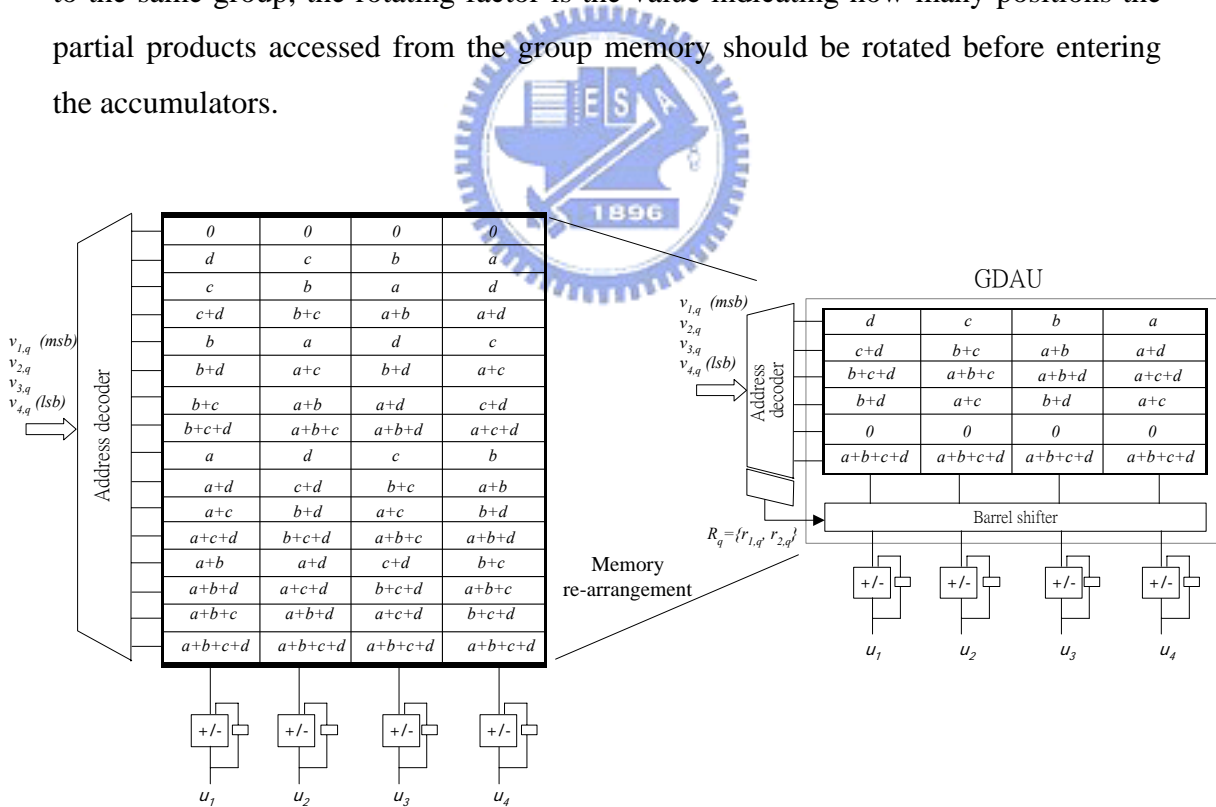


Fig. 2.1: The proposed GDA architecture and the associated memory content arrangement in realizing the cyclic convolution example shown in (2.2).

Table 2.1: The rule of group mapping.

Grouped candidates of DA input (V_q) $\{v_{1,q}, v_{2,q}, v_{3,q}, v_{4,q}\}$	Seed value (V'_q) $\{v'_{1,q}, v'_{2,q}, v'_{3,q}, v'_{4,q}\}$	¹ Rotating factor (R_q) $\{r_{1,q}, r_{2,q}\}$	Group address (G_q) $\{g_{1,q}, g_{2,q}, g_{3,q}\}$
0001	0001	0	000
0010		1	
0100		2	
1000		3	
0011	0011	0	001
0110		1	
1100		2	
1001		3	
0111	0111	0	010
1110		1	
1101		2	
1011		3	
0101	0101	0	011
1010		1	
0000	0000	0	100
1111	1111	0	101

Note:

1. Rotating factor denotes the number of position of the output data, corresponding to the candidate of DA input value in a group, needs to rotate.



2.2.2 Analysis of Barrel shifter

In this subsection, we will illustrate the hardware cost of barrel shifter in the design of overhead. Four barrel shifter designs are respectively analyzed and evaluated in the following. Fig. 2.2 shows the architecture realized with multiplexer. This straight forward design adopts the multiplexers that switch the input data to the selected outputs by the control signals as a rotation operation. The hardware required of this design is N times of $N \log_2 N + 1$ -input AND gates and one N -input OR gates. Thus the complexity of hardware is $O(N^2 \log_2(\log_2 N + 1) + \log_2 N)$ in gate count. It reveals that the design with this approach is not hardware efficient. Besides, the number of level of the multiplexer logic will increase while the number of input is increased. Then the delay time in this design will be not a constant.

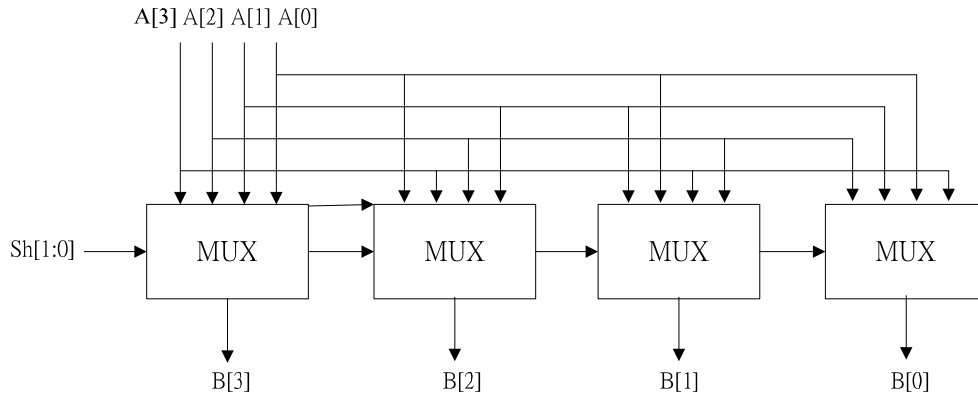


Fig. 2.2: Multiplexer-based barrel shifter design.

Fig. 2.3 shows the second design of barrel shifter. It adopts the multiplier with double length of input data. The duplicated input data is multiplied by the control signals, and then select out the 2^{nd} N-bit of the result of multiplier as the shifted result. Although implementation with this algorithm uses only one multiplier and one-to-four demultiplexer, the word length in them is the drawback in hardware implementation. The required hardware in this design is one 2N-bit multiplier and one N-bit one-to-four demultiplexer. It is equivalent to $10^3 * (-0.039 + 0.457 * 2N + 0.001 * 2N + 0.263 * 4N^2) / 58$ and 2N 2-input gates (i.e., $N * 2$ 2-input gate). Thus the complexity of hardware is $O(N^2 + 2N)$ in gate count.

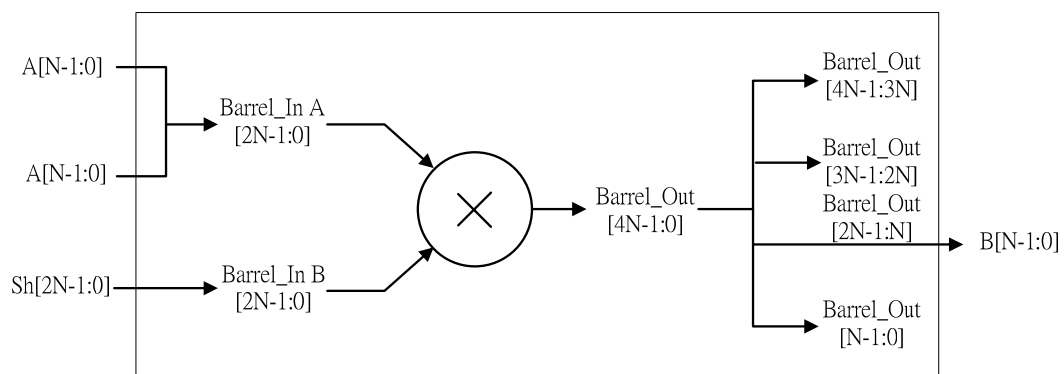


Fig. 2.3: Multiplier-based barrel shifter design.

Fig. 2.4 shows the third design of barrel shifter. This design consists of $\log_2 N$ rotators and $\log_2 N$ N-bit two-to-one multiplexer. The length of these rotators are respectively $2^0, 2^1, \dots, 2^{\log_2 N-1}$. If the length of the barrel shifter is not power of two, the length of most significant rotator is $N - (2^{\log_2 N-2} + 2^{\log_2 N-3} + \dots + 2^0)$. Since each of the rotators can be realized with the manner of wiring, there is no hardware cost on these rotators. Therefore, the hardware cost of this barrel shifter design is $\log_2 N$ N-bit two-to-one multiplexer. It is equivalent to $\log_2 N$ times of $2N$ 2-input AND gates and N 2-input OR gates. Thus the complexity of hardware is $O(3N \log_2 N)$ in gate count.

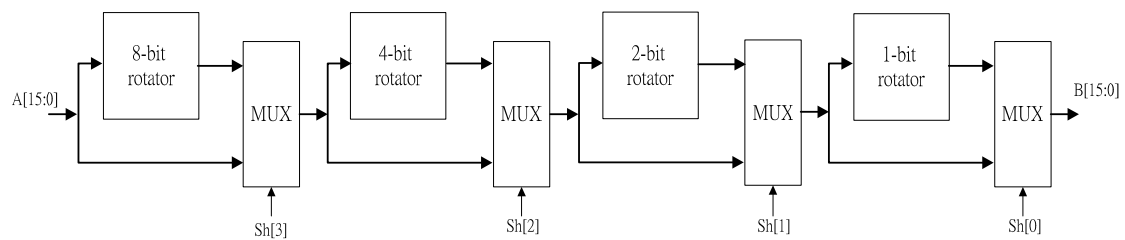


Fig. 2.4: Logarithmic number of multiplexer barrel shifter design.

Fig. 2.5 shows the fourth design of barrel shifter. This design consists of N^2 transistors and N inverter gates. The hardware cost of this barrel shifter design is equivalent to $N^2/4 + N/2$ in gate count. Thus the complexity of hardware of this design is $O(N^2/4 + N/2)$ in gate count. Compared with the designs mentioned above, it reveals that this design is the most efficient choice for the case that the length of input data is smaller than 64.

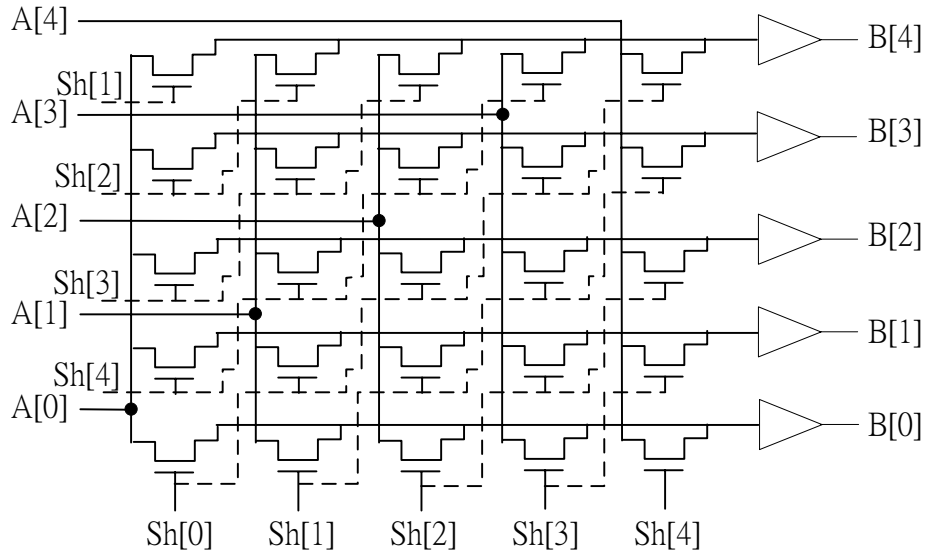
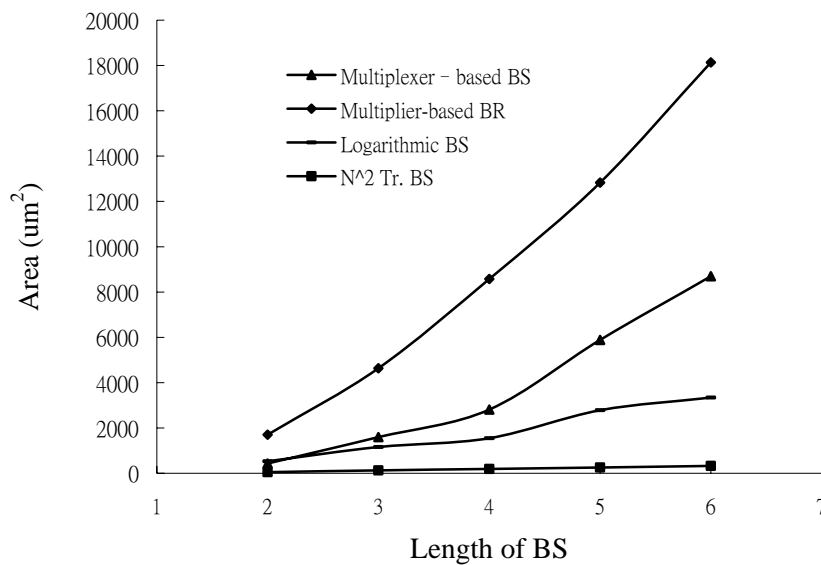
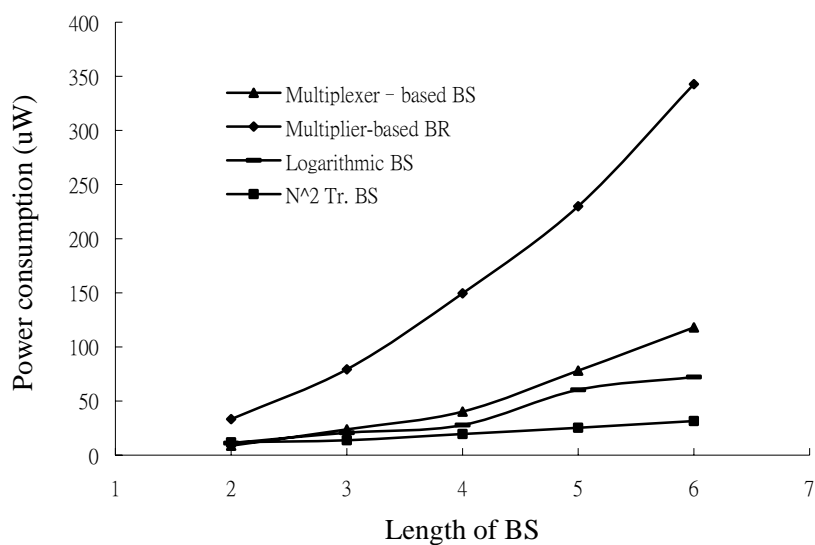


Fig. 2.5: Barrel shifter with N^2 transistors

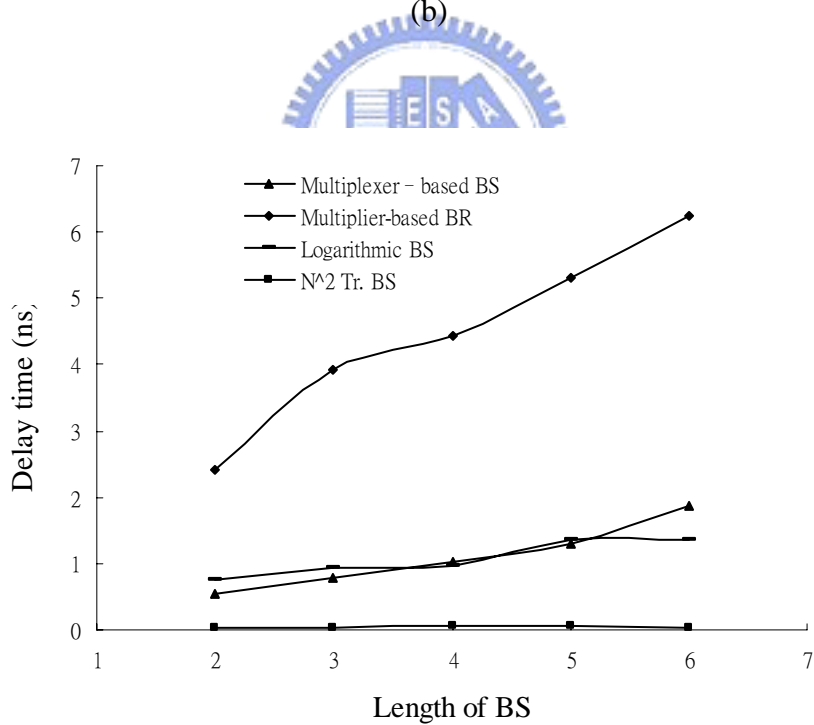
Fig. 2.6 (a), (b), and (c) show the comparisons of the four barrel shifters in hardware cost, power consumption, and delay time, respectively. It is seen that the area cost, power consumption, and delay time of N^2 -transistor barrel shifter are almost smaller than the others. However, this design is hard to implement by synthesis in the cell-based design flow. Thus the alternative of logarithmic barrel shifter is chosen, and synthesized in the implementation of the proposed GDA design and its applications.



(a)



(b)



(c)

Fig. 2.6: Comparison of the four barrel shifters in (a) hardware cost, (b) power consumption, and (c) delay time.

2.2.3 Evaluation of hardware cost

In the following, we evaluate the delay time and hardware cost of the designs with the proposed GDA approach and the traditional DA approach for illustrating the advantages of the proposed approach. For a fair comparison, we adopt Avant 0.35 μ m CMOS cell-library [43] in the performance evaluation. The delay time for accessing a partial product from a memory module is $t_{addr_dec} + t_{rom_acc}$ in the traditional DA designs, and $t_{addr_dec} + t_{rom_acc} + t_{bar_shf}$ in the GDA design, where t_{addr_dec} denotes the delay time of address decoder, t_{rom_acc} denotes the access time of memory, and t_{bar_shf} denotes the delay time of the barrel shifter. Since the memory size required in the GDA design is much smaller than that in the traditional DA design, the delay time of address decoder and access time of memory in the GDA design are accordingly much smaller than that in the traditional DA design. However, the extra delay time of the barrel shifter must be counted in the GDA design. As a result, the total delay of the GDA design is almost similar to that of the traditional DA design. As for the hardware cost evaluation, the hardware for accessing a partial product is A_{rom} in the traditional DA design, and is $A_{grp_rom} + A_{bar_shf}$ in the proposed GDA design, where A_{grp_rom} denotes the area cost of Group memory, and A_{bar_shf} denotes the area cost of a barrel shifter.

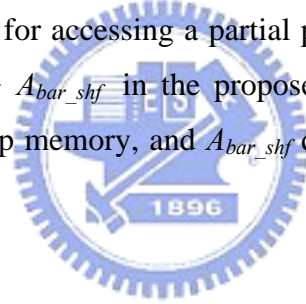


Table 2.2: Comparison of memory size in both the traditional memory-based DA and the proposed GDA designs for different values of N.

<i>Length of cyclic convolution (N)</i>	3	4	5	6	7	8	9	10	11	12	13	14	...
Traditional DA	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	...
GDA (# of group: G(N))	4	6	8	14	20	36	60	108	188	352	632	1197	...
memory size reduction ratio (DA / GDA)	2	2.7	4	4.6	6.4	7.1	8.5	9.5	10.9	11.6	13	13.7	...

Table 2.2 shows the comparison of memory size required in the two designs under different N. We can see that the proposed GDA design is much more hardware efficient than the traditional DA design. Fig. 2.7 shows the measure of delay-area product to evaluate the performance for the proposed GDA design and the traditional DA design. We find that the delay-area product of the proposed GDA design is much

smaller than that of the traditional DA design as N increases, which illustrates that the proposed GDA design possesses better performance than the traditional DA designs in terms of delay-area product.

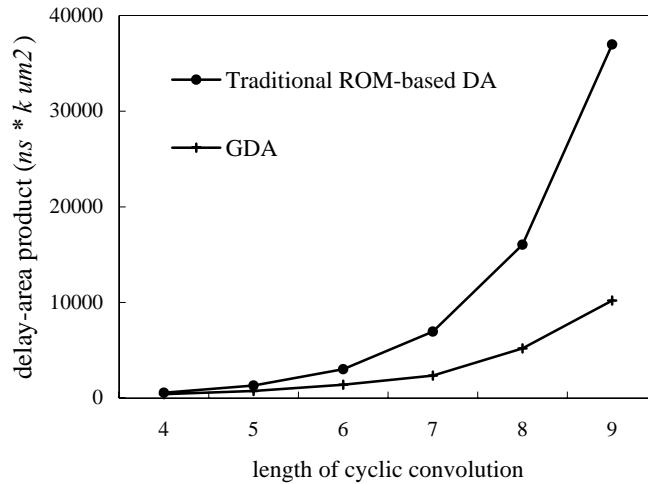


Fig. 2.7: The delay-area product comparison in the proposed GDA design and the traditional memory-based DA design with 16-bit data word length.

2.3 Consideration of low power design

With the approach of address grouping in GDA design, the number of address appears on DA input has been reduced significantly such that the transition activity on the word-line of memory in original DA design is reduced. And due to reduction of the memory size in GDA design, the bit-line loading as well as the transition activity on the bit-line is also reduced. Besides, the barrel-shifter is with higher driving strength than ROM in conventional DA. On the power consumption point of view, the proposed GDA design should be not only the low hardware cost design but also a low power design. In the following, we will analysis and evaluate the GDA design to be a low power design.

2.3.1 Analysis of transition activity

In general, transition activity at the output of circuitry depends on the transition activity at the inputs and the circuitry function. The transition probability of a node

from 0 to 1 (i.e., $\alpha_{0 \rightarrow 1}$) is $p_0 p_1$, where p_0 and p_1 denote the probability of signal is settled on logic-0 and logic-1, respectively [44]. The transition probability appeared in the input of data-path have affected power consumption of the followed circuitry. Considering a design example of 4-input data-path, Table 2.3 shows the comparisons of transition probability and Hamming distance, respectively. Since grouped binary is a subset of the complete binary, we can select to construct a subset as the distribution of group addresses with lowest Hamming distance. Thus the transition activity on the input nodes will be much smaller than that of complete binary such that the power consumption of the data-path can be reduced significantly. Fig. 2.8 shows the trend of sum of transition probability against the number of input-data bit.

Table 2.3: Transformation of transition probability for the input data of the 4-input data-path.

Input data	Complete binary				Grouped binary			
	v_3	v_2	v_1	v_0	v_3	v_2	v_1	v_0
	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0	1
	0	0	1	0				
	0	1	0	0				
	1	0	0	0				
	0	0	1	1	0	0	1	0
	0	1	1	0				
	1	1	0	0				
	1	0	0	1				
	0	1	0	1	0	1	0	0
	1	0	1	0				
	0	1	1	1	1	0	0	0
	1	1	1	0				
	1	1	0	1				
	1	0	1	1				
	1	1	1	1	0	0	1	1
Transition probability	64/256	64/256	64/256	64/256	5/36	5/36	8/36	8/36
Sum of transition probability	1				0.722			
Average Hamming distance	2				1.444			
improvement	27.8 %							

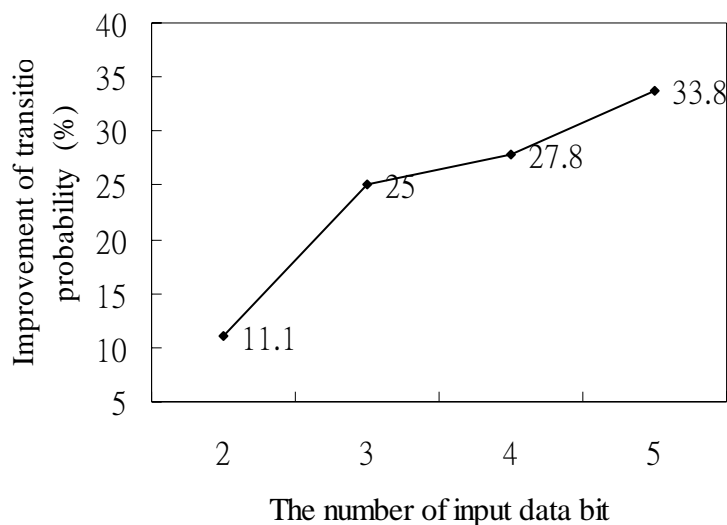


Fig. 2.8: Trend of the improvement of transition probability versus the number of input-data bit.

2.3.2 Address morphing approach

With the GDA design approach, the distribution of DA input address is reduced into few groups. Shown as Table 2.4, we can realize the cyclic convolution by using the scheme of address morphing that converts the distribution of DA input address into a subset of it with minimal transition activity such that the transition activity on the word lines of memory is reduced. Even in the case of never removing the unused entries of memory, due to the lower input activity, the power consumption of memory shown in Fig. 2.9 is reduced. Actually due to the number of memory entry is reduced in Fig. 2.9; the bit-line loading of memory and transition activity on the bit-lines are also reduced. Thus the power consumption of memory in the GDA design is reduced significantly. However, the barrel shifter in the overhead of GDA design consumes extra power such that the overall power consumption of GDA design with short length is improved inconspicuously.

Table 2.4: The relation ship of the address morphing.

<i>DA input address (V_q)</i> $\{v_{1,q}, v_{2,q}, v_{3,q}, v_{4,q}\}$	<i>Rotating factor (R_q)</i> $\{r_{1,q}, r_{2,q}\}$	<i>Morphed address</i>
0001	0	0001
0010	1	
0100	2	
1000	3	
0011	0	0011
0110	1	
1100	2	
1001	3	
0111	0	0111
1110	1	
1101	2	
1011	3	
0101	0	0101
1010	1	
0000	0	0000
1111	0	1111

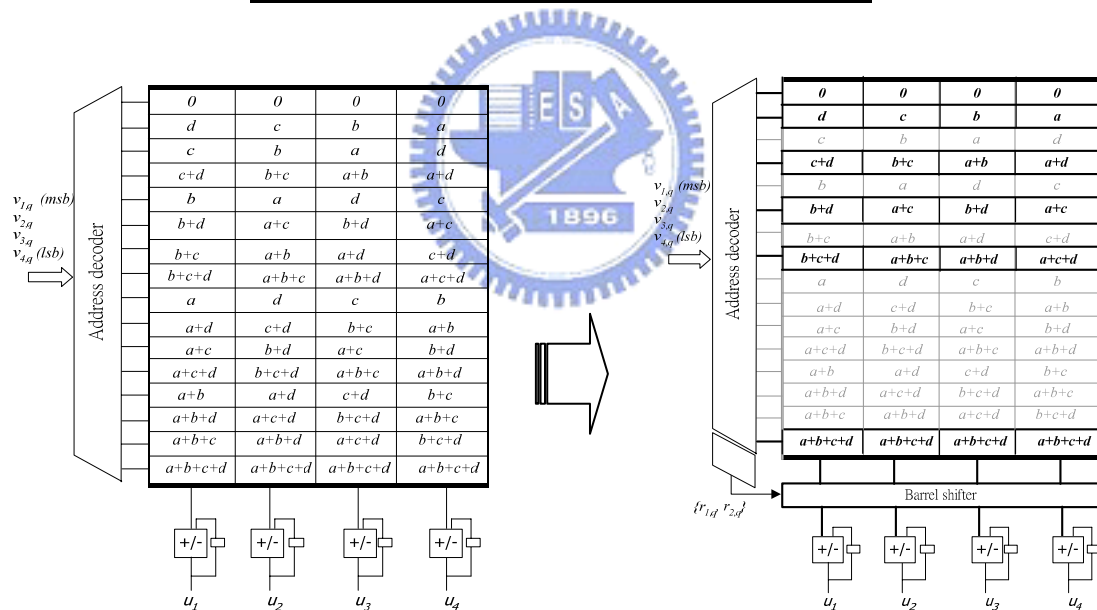


Fig. 2.9: The description of architecture transformation from DA to GDA

2.3.3 Exploration of dynamic range of the input data

The power consumption of a circuit highly depends on the transition activity of input data. In some video codec systems, the data to be processed is the difference of the adjacent frames such as the inter frame used in the video codec standards of

MPEG-2, MPEG-4, H.26X, and etc. As most of the pixels in the inter frame, the difference is with smaller value such that some of the higher bit in DA computation can be omitted to achieve lower power consumption. On the second concept, with the choice of DSP algorithm, sometimes the data fed into the processing unit needs to be processed previously such as the difference of input data. It means that we can exploit the property of correlation for the local data such that the dynamic range of these being processed data is reduced significantly. For example of 7-point DCT in cyclic convolution formulation, the data on the input of processing unit is not the direct input data. These data need to be computed previously with the combination of subtractions and additions. Fig. 2.10, Fig. 2.11, and Fig. 2.12 show the test image, gray-level of the pixels in this image, and histogram of the gray-level distribution, respectively. Fig. 2.13 and Fig. 2.14 respectively show the preprocessed gray-level and the histogram of the input data of processing unit in DCT design. It reveals that most of the preprocessed data values is small than the original one. Thus, the dynamic range is reduced for most of the input data. Combined with the second concept above, the number of cycle of DA computation in the DCT design will be reduced to achieve lower power consumption.

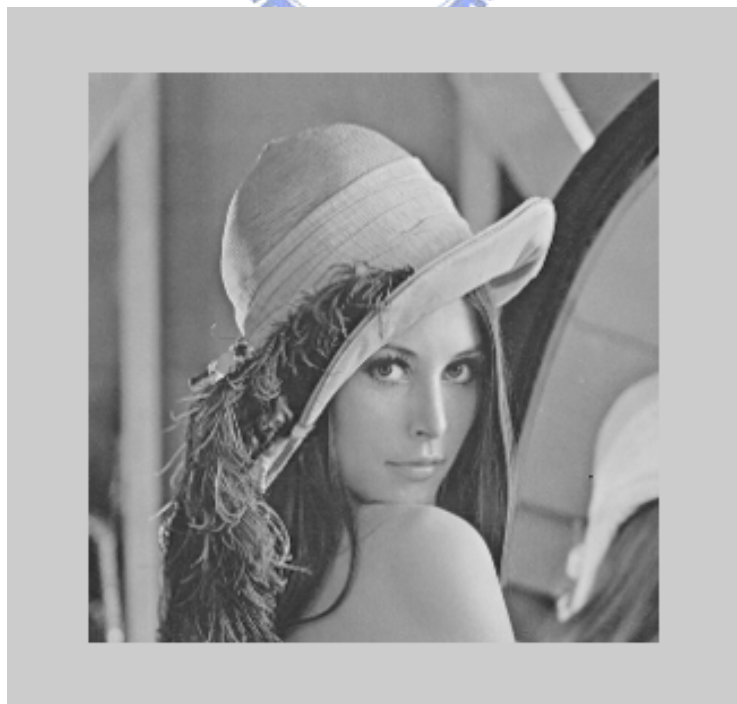


Fig. 2.10: The test image with the size of 252 * 252 pixels.

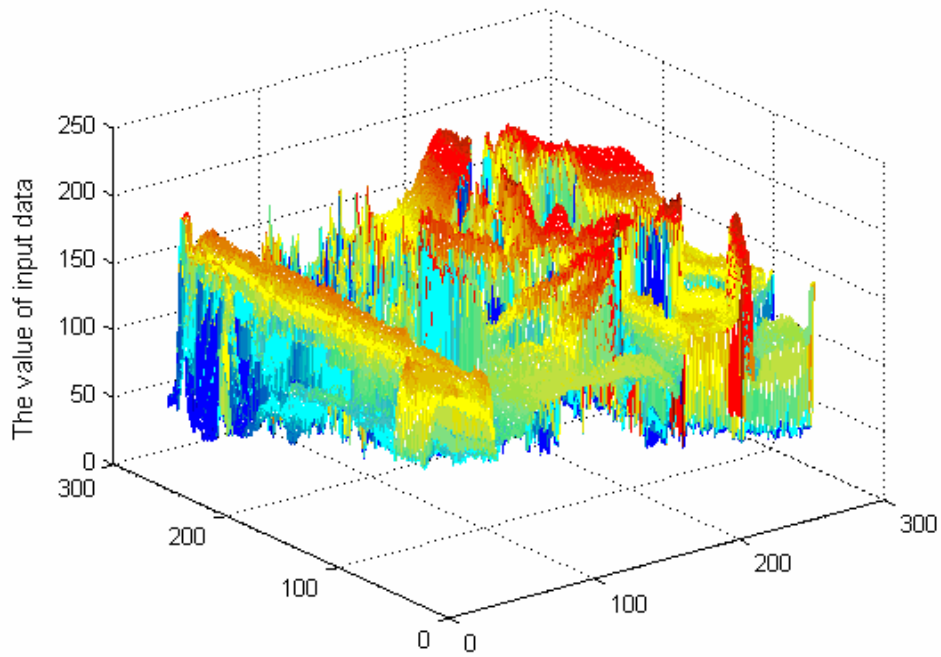


Fig. 2.11: gray-level of the pixels in the image of Fig. 2.10.

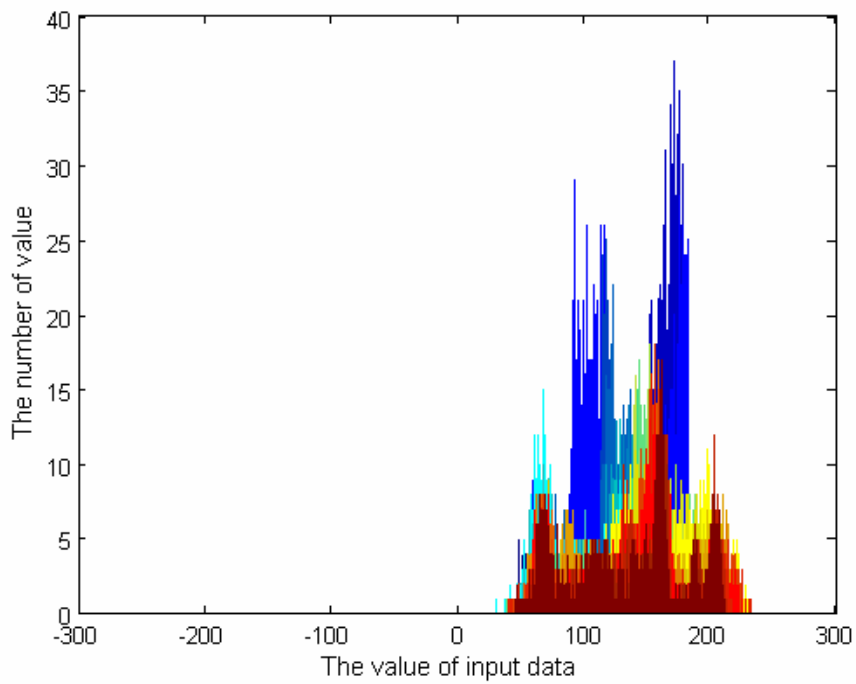


Fig. 2.12: histogram of the gray-level distribution in the image of Fig. 2.10.

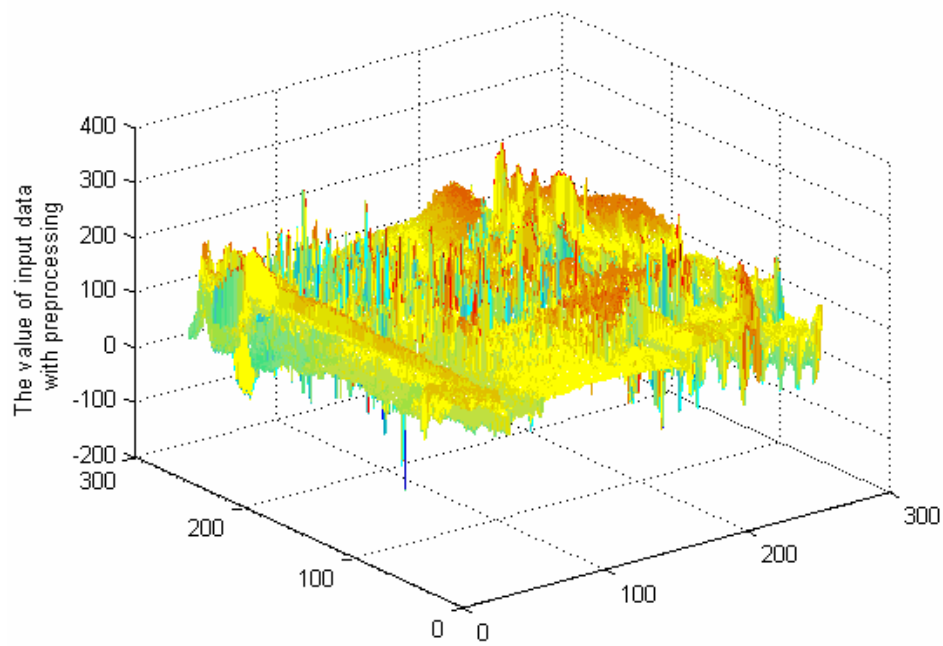


Fig. 2.13: The preprocessed gray-level of the image in Fig. 2.10.

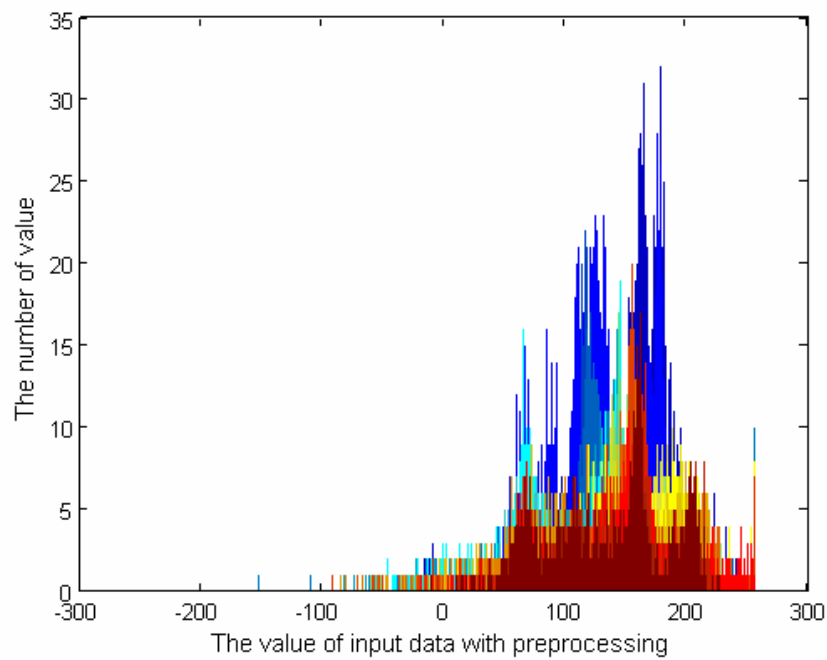


Fig. 2.14: Histogram of the preprocessed data used in the example of 7-point DCT design.

2.3.4 Low Power Design with pre-computation scheme

Exploiting the property of spatial correlation in natural images, for the algorithm with the inputs formed as sum and difference of the primary inputs, the sum of inputs are likely to have a number of equal high-order bits, and the difference inputs are likely to have small dynamic range. Then for some cases, such as the 8-point 1-D DCT, the cycles of DA computation for the high-order bits of sum inputs can be skipped. On the other hand, since most of the bits in high-order bits of the difference of inputs are the sign-extension bits, with the manner of bit-serial and word-parallel, the cycles in DA computation for these extended sign-bits can also be skipped to achieve lower computation power [45]. In the following, we will illustrate the high-order bits rejection technique briefly, where this technique named most significant bit rejection (MSBR) in [45], and explore the distribution of pre-computed input data for the cyclic convolution formulation of prime-length DCT. For the realization of prime-length DCT, combining the proposed GDA design with the MSBR technique facilitates not only reducing the memory size, but also improving the power consumption.



MSBR technique

Considering the even and odd outputs of the reformulated 8-point 1-D DCT as

$$\begin{bmatrix} X_0 \\ X_2 \\ X_4 \\ X_6 \end{bmatrix} = \begin{bmatrix} A & A & A & A \\ B & C & -C & -B \\ A & -A & -A & A \\ C & -B & B & -C \end{bmatrix} \cdot \begin{bmatrix} x_0 + x_7 \\ x_1 + x_6 \\ x_2 + x_5 \\ x_3 + x_4 \end{bmatrix}, \quad (2.4)$$

$$\begin{bmatrix} X_1 \\ X_3 \\ X_5 \\ X_7 \end{bmatrix} = \begin{bmatrix} D & E & F & G \\ E & -G & -D & -F \\ F & -D & G & E \\ G & -F & E & -D \end{bmatrix} \cdot \begin{bmatrix} x_0 - x_7 \\ x_1 - x_6 \\ x_2 - x_5 \\ x_3 - x_4 \end{bmatrix} \quad (2.5)$$

Observing the Table 2.5, we can see that some of candidates of DA input, i.e., 0000 and 1111, cause the even output to be zero. It means that the computation, shown as the rejected bits in Fig. 2.15, can be skipped in DA computation.

Table 2.5: Relationship between the sum of primary inputs and the even outputs.

$\{x_0+x_7, x_1+x_6, x_2+x_5, x_3+x_4\}$	X_0	X_2	X_4	X_6
0000	0	0	0	0
0001	A	-B	A	-C
0010	A	-C	-A	B
0011	2A	-(B+C)	0	B-C
0100	A	C	-A	-B
0101	2A	-(B-C)	0	-(B+C)
0110	2A	0	-2A	0
0111	3A	-B	-A	-C
1000	A	B	A	C
1001	2A	0	2A	0
1010	2A	B-C	0	B+C
1011	3A	-C	A	B
1100	2A	B+C	0	-(B-C)
1101	3A	C	A	-B
1110	3A	B	-A	C
1111	4A	0	0	0

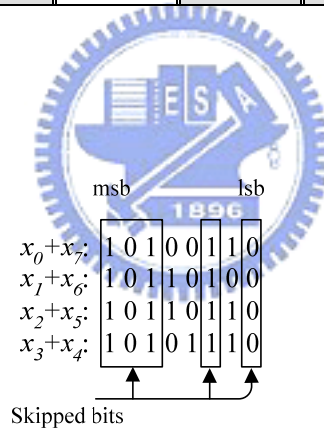


Fig. 2.15: The skipped bits in DA computation for the even outputs.

As for the computation of odd outputs, with the property of high spatial correlation for the pixels in an image, shown as Fig. 2.16 the difference of primary inputs reveals the property of small dynamic range, and thus most of the high-order bits in these difference inputs are the sign-extension bits. Then we need only computing for the least significant bit of sign-extension bits to have the exactly final result of DA computation. Similar to the sum inputs, the number of cycles in DA computation for these extended sign-bits can also be reduced significantly. However, due to the huge amount of overhead for skippable bits detection, development of the

efficient detection scheme is still the issue of low power GDA-based design with MSBR technique.

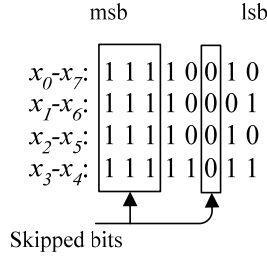


Fig. 2.16: The skipped bits in DA computation for the odd outputs.

Exploration of the input data for prime-length DCT in cyclic convolution

In the following, we illustrate how the MSBR technique can apply to the prime-length DCT design with the example of 7-point DCT. Considering the kernel of DCT $T((3^k)_7)$ in (2.7), where $x((3^{n-k+1})_7)$ denotes the indirect inputs pre-computed from the primary input $y(n)$ as (2.8).

$$Y(0) = \sum_{n=0}^6 y(n)$$

$$Y((3^k)_7) = [2 \cdot T((3^k)_7) + x(0)] \cdot \cos\left(\frac{\pi}{14} \cdot ((3^k)_7)\right); k = 1, \dots, 6 \quad (2.6)$$

$$T((3^k)_7) = \sum_{n=1}^6 x((3^{n-k+1})_7) \cdot (-1)^m \cdot \cos\left(\frac{\pi}{7} \cdot (3^{n+1})_7\right) \quad (2.7)$$

where $(3^k)_7$ denotes the result of “ 3^k modulo 7” for short,

$x((3^{n-k+1})_7) = \begin{cases} x((3^{n-k+1})_7); & \text{if } n - k + 1 \geq 0 \\ x((3^{6+(n-k+1)})_7); & \text{if } n - k + 1 < 0 \end{cases}$, the value of m is determined by

$(3^{n+1})_7 + m \cdot 7 = (3^{n-k+1})_7 \cdot (3^k)_7; n, k = 1, \dots, 6$, and the sequence $\{x(n)\}$ is defined

as

$$\left\{ \begin{array}{l} x(6) = y(6) \\ x(n) = y(n) - x(n+1); n = 0, \dots, 5 \end{array} \right\}. \quad (2.8)$$

We can write the kernel $T((3^k)_7)$ as the matrix form

$$\begin{bmatrix} T(3) \\ T(2) \\ T(6) \\ T(4) \\ T(5) \\ T(1) \end{bmatrix} = \begin{bmatrix} -x(3) & x(2) & x(6) & -x(4) & x(5) & x(1) \\ x(1) & x(3) & x(2) & -x(6) & -x(4) & -x(5) \\ x(5) & x(1) & x(3) & -x(2) & -x(6) & -x(4) \\ x(4) & x(5) & x(1) & -x(3) & -x(2) & -x(6) \\ x(6) & x(4) & -x(5) & x(1) & x(3) & -x(2) \\ x(2) & x(6) & x(4) & x(5) & x(1) & x(3) \end{bmatrix} \times \begin{bmatrix} \cos(2a) \\ \cos(6a) \\ \cos(4a) \\ \cos(5a) \\ \cos(1a) \\ \cos(3a) \end{bmatrix}, \quad (2.9)$$

where a denotes $\frac{\pi}{7}$.

And then exploiting the symmetry property of DCT coefficients, (2.9) is reformulated as

$$\begin{bmatrix} T(3) \\ T(2) \\ T(6) \\ T(4) \\ T(5) \\ T(1) \end{bmatrix} = \begin{bmatrix} -x(3) & x(2) & x(6) & -x(4) & x(5) & x(1) \\ x(1) & x(3) & x(2) & -x(6) & -x(4) & -x(5) \\ x(5) & x(1) & x(3) & -x(2) & -x(6) & -x(4) \\ x(4) & x(5) & x(1) & -x(3) & -x(2) & -x(6) \\ x(6) & x(4) & -x(5) & x(1) & x(3) & -x(2) \\ x(2) & x(6) & x(4) & x(5) & x(1) & x(3) \end{bmatrix} \times \begin{bmatrix} \cos(2a) \\ \cos(6a) \\ \cos(4a) \\ -\cos(2a) \\ -\cos(6a) \\ -\cos(4a) \end{bmatrix} \quad (2.10)$$

, and

$$\begin{bmatrix} T(3) \\ T(2) \\ T(6) \\ T(4) \\ T(5) \\ T(1) \end{bmatrix} = \begin{bmatrix} x(4) - x(3) & x(2) - x(5) & x(6) - x(1) \\ x(6) + x(1) & x(4) + x(3) & x(2) + x(5) \\ x(2) + x(5) & x(6) + x(1) & x(4) + x(3) \\ x(4) + x(3) & x(2) + x(5) & x(6) + x(1) \\ x(6) - x(1) & x(4) - x(3) & x(2) - x(5) \\ x(2) - x(5) & x(6) - x(1) & x(4) - x(3) \end{bmatrix} \times \begin{bmatrix} \cos(2a) \\ \cos(6a) \\ \cos(4a) \end{bmatrix}. \quad (2.11)$$

To separate the even and odd outputs, two smaller perfect cyclic forms are shown as

$$\begin{bmatrix} T(2) \\ T(6) \\ T(4) \end{bmatrix} = \begin{bmatrix} x(6) + x(1) & x(4) + x(3) & x(2) + x(5) \\ x(2) + x(5) & x(6) + x(1) & x(4) + x(3) \\ x(4) + x(3) & x(2) + x(5) & x(6) + x(1) \end{bmatrix} \times \begin{bmatrix} \cos(2a) \\ \cos(6a) \\ \cos(4a) \end{bmatrix} \quad (2.12)$$

and

$$\begin{bmatrix} T(5) \\ T(1) \\ T(3) \end{bmatrix} = \begin{bmatrix} x(6) - x(1) & x(4) - x(3) & x(2) - x(5) \\ x(2) - x(5) & x(6) - x(1) & x(4) - x(3) \\ x(4) - x(3) & x(2) - x(5) & x(6) - x(1) \end{bmatrix} \times \begin{bmatrix} \cos(2a) \\ \cos(6a) \\ \cos(4a) \end{bmatrix}. \quad (2.13)$$

With the property of spatial correlation, the difference of the indirect inputs will

remain most of the high-order bits as sign-extension bits such that the cycles of DA computation for most of the bits can be skipped. Similar to the benefit of MSBR technique in 8-point DCT design, combining this technique with the proposed GDA approach for the prime-length DCT design facilitates not only low hardware cost but also low power consumption.

2.3.5 Evaluation of power cost

We have synthesized and verified the power consumption of 1-D 5-point to 13-point DCT designs at the clock frequency of 166MHz by using respectively DesignCompiler and PrimePower with the UMC 0.18um cell-library and the test benches of Lena, Babon, and Peper. As shown in Fig. 2.17, the simulation result shows that power consumption of the 1-D prime-length DCT with GDA design is lower than that of the conventional DA design for the test benches with different characteristics of content. With the power consumption point of view, it reveals that the proposed GDA design is also a low power design.

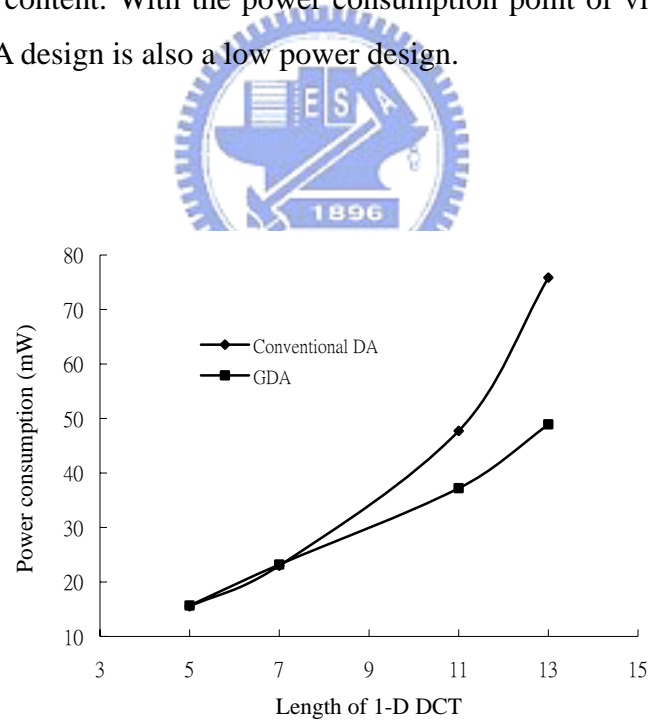
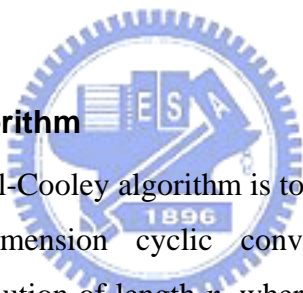


Fig. 2.17: Power consumption of the GDA-based 1-D DCT designs.

2.4 Partitioning of cyclic convolution

Because of the inherent issue of DA-based design that the memory size increases exponentially as the length of input data increases, the partition issue must be regarded. In the conventional DA design, we can arbitrarily partition the input data of DA, and then sum up the partial sums from the different memory modules to achieve low hardware cost. However, because of the necessity of cyclic preserving, the manner of arbitrarily partitioning cannot be applied to the proposed GDA design. Otherwise, the benefit of low hardware cost in GDA design will not exist. To solve the problems mentioned above, we combine applicably the proposed GDA approach with the partition methods for prime length and non-prime length cyclic convolutions respectively such that the case of long length GDA can be partitioned, and composed of the short cyclic-convolution blocks. It facilitates that we can still realize each of the shortened cyclic convolution blocks with the proposed GDA design to achieve low hardware cost.

2.4.1 Agarwal-Cooley algorithm



The approach of Agarwal-Cooley algorithm is to convert one-dimensional cyclic convolution into a multidimension cyclic convolution [41]. In essence, a one-dimensional cyclic convolution of length n , where $n = n_1 * n_2$, and n_1 and n_2 are relatively prime, can be expressed as a two-dimensional cyclic convolution of length n_1 and n_2 , respectively. The extension of the idea to convert one-dimensional cyclic convolution to a d-dimensional cyclic convolution when n has d relatively co-prime factors, that is $n = n_1 * n_2 * \dots * n_d$, and n_i and n_j are relatively prime, $i \neq j$, is straightforward. The Agarwal-Cooley algorithm consists in the application of Chinese remainder theorem for integers (CRT-I) [46] to the indices of sequences being convoluted. Therefore, it is valid for data sequences defined over any arbitrary number system. A major advantage of the Agarwal-Cooley algorithm is that the long length cyclic convolution can be constructed from short length cyclic convolution. Table 2.6 shows the covered lengths that the cyclic convolution can be decomposed with Agarwal-Cooley algorithm.

Table 2.6: Analysis for the covered lengths of cyclic convolution can be decomposed.

Length of cyclic convolution	Decomposition factors	Length of cyclic convolution	Decomposition factors	Length of cyclic convolution	Decomposition factors	Length of cyclic convolution	Decomposition factors
7	7	20	4*5	33	3*11	43	43
10	2*5	21	3*7	34	2*17	44	4*11
11	11	22	2*11	35	5*7	45	9*5
12	4*3	23	23	36	4*9	46	2*13
13	13	24	8*3	37	37	47	47
14	2*7	26	2*13	38	2*19	48	3*16
15	3*5	28	4*7	39	3*13	50	5*10
17	17	29	29	40	8*5	51	51
18	2*9	30	6*5	41	41	⋮	⋮
19	19	31	31	42	6*7		

Note: Power of two and power of prime-value cannot be covered.

2.4.2 Pseudocirculant matrix factorization algorithm

Since the partitioning factors for cyclic convolution are not relatively co-prime, the Chinese Remainder Theorem I (CRT-I) cannot be used in the indices of sequences being convoluted. Thus for preserving the cyclic property for GDA design, we use the pseudocirculant matrix factorization algorithm [42] for further partitioning the long-length cyclic convolution. With this algorithm, shown as (2.14) and (2.15), the cyclic convolution with the length of N can be factorized as the factors of N/r and r .

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 & \cdots & c_{N-1} & c_N \\ c_N & c_1 & c_2 & c_3 & \cdots & c_{N-1} \\ c_{N-1} & c_N & c_1 & c_2 & c_3 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ c_3 & \cdots & c_{N-1} & c_N & c_1 & c_2 \\ c_2 & c_3 & \cdots & c_{N-1} & c_N & c_1 \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{N-1} \\ v_N \end{bmatrix} \quad (2.14)$$

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_{r-1} \\ U_r \end{bmatrix} = \begin{bmatrix} C_1 & S_{N/r}C_2 & S_{N/r}C_3 & \cdots & S_{N/r}C_{r-1} & S_{N/r}C_r \\ C_r & C_1 & S_{N/r}C_2 & S_{N/r}C_3 & \cdots & S_{N/r}C_{r-1} \\ C_{r-1} & C_r & C_1 & S_{N/r}C_2 & S_{N/r}C_3 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ C_3 & \cdots & C_{r-1} & C_r & C_1 & S_{N/r}C_2 \\ C_2 & C_3 & \cdots & C_{r-1} & C_r & C_1 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ \vdots \\ V_{r-1} \\ V_r \end{bmatrix} \quad (2.15)$$

where $\{v_1, v_2, v_3, v_4, \dots, v_N\}$ are input data, $\{c_1, c_2, c_3, c_4, \dots, c_N\}$ are coefficients, and $\{u_1, u_2, u_3, u_4, \dots, u_N\}$ are output data. The cyclic shift operator $S_{N/r}$ can be written in form as

$$S_{N/r} = \begin{bmatrix} 0 & 0 & 0 & \vdots & 0 & 1 \\ 1 & 0 & 0 & \vdots & 0 & 0 \\ 0 & 1 & 0 & \vdots & 0 & 0 \\ 0 & 0 & 1 & \vdots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \vdots & 1 & 0 \end{bmatrix}$$

Using the commutative property of convolution, we can rewrite (2.14) and (2.15) as follow:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 & \dots & v_{N-1} & v_N \\ v_2 & v_3 & \dots & v_{N-1} & v_N & v_1 \\ v_3 & \dots & v_{N-1} & v_N & v_1 & v_2 \\ \vdots & & \ddots & \vdots & \vdots & \vdots \\ v_{N-1} & v_N & v_1 & \dots & v_{N-3} & v_{N-2} \\ v_N & v_1 & \dots & v_{N-3} & v_{N-2} & v_{N-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{N-1} \\ c_N \end{bmatrix} \quad (2.16)$$

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_{r-1} \\ U_r \end{bmatrix} = \begin{bmatrix} V_1 & S_{N/r}V_2 & S_{N/r}V_3 & S_{N/r}V_4 & \dots & S_{N/r}V_r \\ V_2 & S_{N/r}V_3 & S_{N/r}V_4 & \dots & S_{N/r}V_r & V_1 \\ V_3 & S_{N/r}V_4 & \dots & S_{N/r}V_r & V_1 & V_2 \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ V_{r-1} & S_{N/r}V_r & V_1 & \dots & V_{r-3} & V_{r-2} \\ V_r & V_1 & \dots & V_{r-3} & V_{r-2} & V_{r-1} \end{bmatrix} \cdot \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ \vdots \\ C_{r-1} \\ C_r \end{bmatrix} \quad (2.17)$$

$$= \begin{bmatrix} S_{N/r}V_2 & S_{N/r}V_3 & S_{N/r}V_4 & \dots & S_{N/r}V_r & V_1 \\ S_{N/r}V_3 & S_{N/r}V_4 & \dots & S_{N/r}V_r & V_1 & V_2 \\ S_{N/r}V_4 & \dots & S_{N/r}V_r & V_1 & V_2 & V_3 \\ \vdots & & & \ddots & & \vdots \\ S_{N/r}V_r & V_1 & V_2 & \dots & V_{r-2} & V_{r-1} \\ V_1 & V_2 & \dots & V_{r-2} & V_{r-1} & V_r \end{bmatrix} \cdot \begin{bmatrix} C_2 \\ C_3 \\ C_4 \\ \vdots \\ C_r \\ C_1 \end{bmatrix}$$

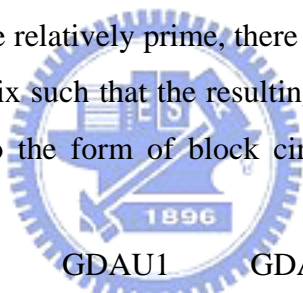
2.4.3 Long length cyclic convolution design

The case of the partitioning factors is relatively co-prime

Consider the example of computing the cyclic convolution example shown in (2.18), where $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ are input data, $\{a, b, c, d, e, f\}$ are coefficients, and $\{u_1, u_2, u_3, u_4, u_5, u_6\}$ are output data.

$$U = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\ v_2 & v_3 & v_4 & v_5 & v_6 & v_1 \\ v_3 & v_4 & v_5 & v_6 & v_1 & v_2 \\ v_4 & v_5 & v_6 & v_1 & v_2 & v_3 \\ v_5 & v_6 & v_1 & v_2 & v_3 & v_4 \\ v_6 & v_1 & v_2 & v_3 & v_4 & v_5 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}, \quad (2.18)$$

In the case of cyclic formulation with 6 input data, it can be factorized into 2 and 3. Since the factors of 2 and 3 are relatively prime, there exists a data permutation on the rows and columns of the matrix such that the resulting matrix of input data shown in (2.19) can be partitioned into the form of block circulants of 2×2 with circulant blocks of 3×3 .



GDAU1 GDAU2

$$U = \begin{bmatrix} u_1 \\ u_5 \\ u_3 \\ u_4 \\ u_2 \\ u_6 \end{bmatrix} = \begin{bmatrix} v_1 & v_5 & v_3 & v_4 & v_2 & v_6 \\ v_5 & v_3 & v_1 & v_2 & v_6 & v_4 \\ v_3 & v_1 & v_5 & v_6 & v_4 & v_2 \\ v_4 & v_2 & v_6 & v_1 & v_5 & v_3 \\ v_2 & v_6 & v_4 & v_5 & v_3 & v_1 \\ v_6 & v_4 & v_2 & v_3 & v_1 & v_5 \end{bmatrix} \begin{bmatrix} a \\ e \\ c \\ d \\ b \\ f \end{bmatrix} \quad (2.19)$$

$$= \begin{bmatrix} U_A \\ U_B \end{bmatrix} = \begin{bmatrix} V_A & V_B \\ V_B & V_A \end{bmatrix} \cdot \begin{bmatrix} A \\ B \end{bmatrix}$$

where

$$U_A = \begin{bmatrix} u_1 \\ u_5 \\ u_3 \end{bmatrix}, \quad U_B = \begin{bmatrix} u_4 \\ u_2 \\ u_6 \end{bmatrix}, \quad V_A = \begin{bmatrix} v_1 & v_5 & v_3 \\ v_5 & v_3 & v_1 \\ v_3 & v_1 & v_5 \end{bmatrix}, \quad V_B = \begin{bmatrix} v_4 & v_2 & v_6 \\ v_2 & v_6 & v_4 \\ v_6 & v_4 & v_2 \end{bmatrix}, \quad A = \begin{bmatrix} a \\ e \\ c \end{bmatrix}, \text{ and}$$

$$B = \begin{bmatrix} d \\ b \\ f \end{bmatrix}.$$

From the hardware point of view, we have partitioned the original memory module into two smaller ones. One memory module stores the combination of the coefficients $\{a, e, c\}$ for $u_1, u_5,$ and u_3 as well as $u_4, u_2,$ and u_6 . Similarly, the other one stores the combination of the coefficients $\{d, b, f\}$ for $u_1, u_5,$ and u_3 as well as $u_4, u_2,$ and u_6 . In performing the memory access, we can access the partial products for $u_1, u_5,$ and u_3 by using the memory address generated from $\{v_1, v_5, v_3\}$ through first memory module and access the partial products for $u_1, u_5,$ and u_3 by using the memory address generated from $\{v_4, v_2, v_6\}$ through second memory module at the same time. Then we sum up the two partial products to have $u_1, u_5,$ and u_3 using the extra adders. With the identical hardware and extra input-data-rotator, we can compute $u_4, u_2,$ and u_6 in the next iteration. The cyclic convolution realized with this partitioning scheme and GDA approach is named block-based group distributed arithmetic (BGDA) in our research.

Fig. 2.18 shows low-cost hardware architecture to realize the design example illustrated in (2.19) based on the proposed BGDA design approach. To meet the requirement of high performance, we can easily duplicate the BGDA modules to construct the high performance version of BGDA design as shown in Fig. 2.19.

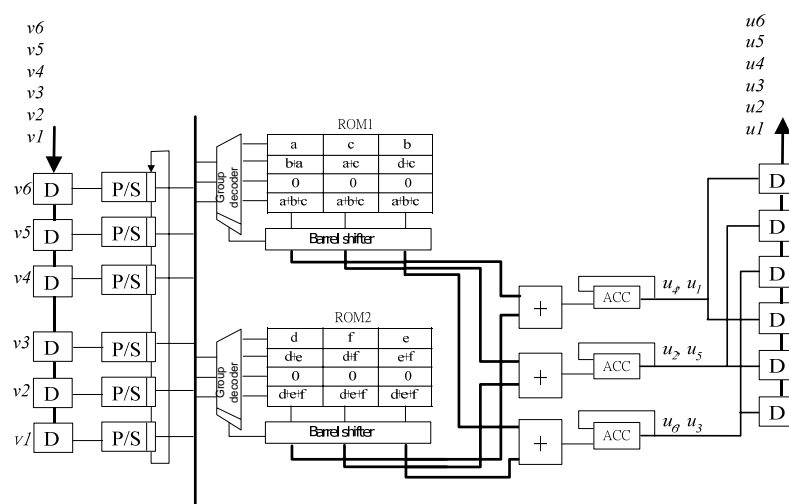


Fig. 2.18: The low cost version of BGDA design realizing the cyclic convolution example shown in (2.19).

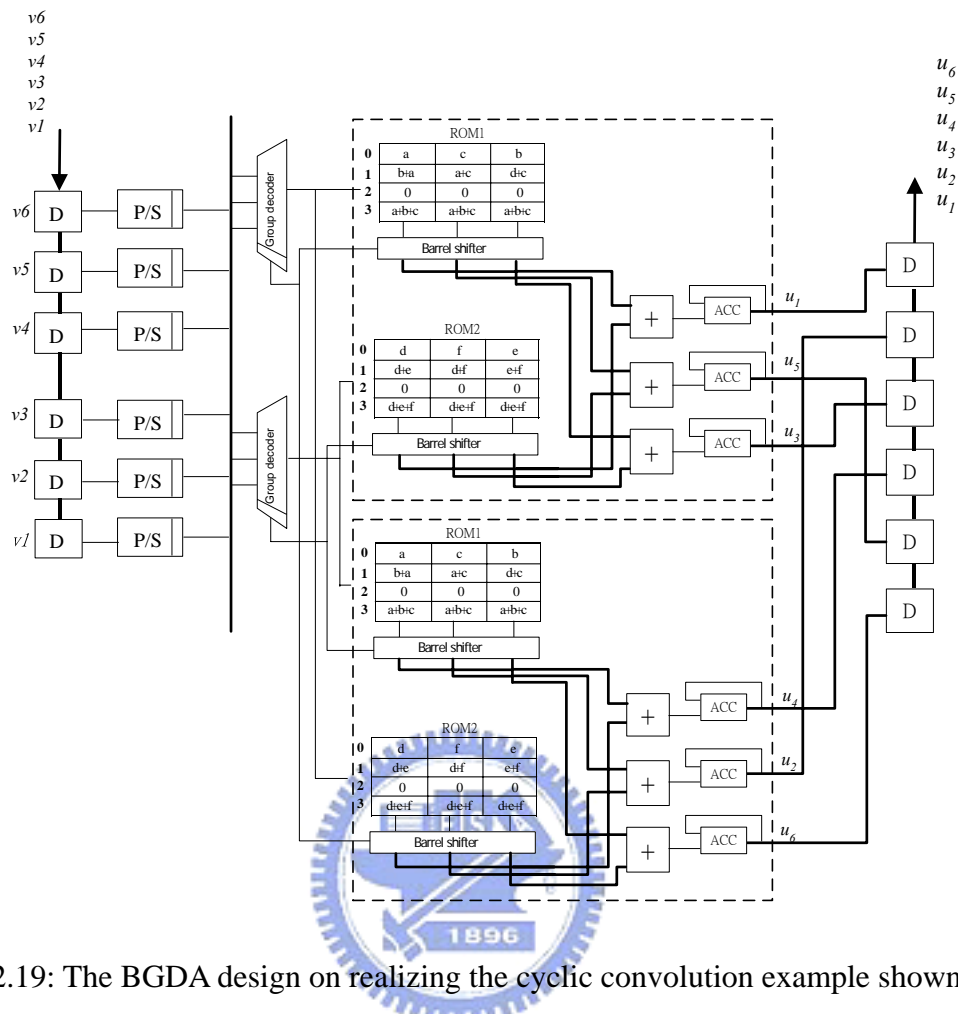


Fig. 2.19: The BGDA design on realizing the cyclic convolution example shown in (2.19) with high performance.

The case of the partitioning factors is not relatively co-prime

Considering the example of computing the cyclic convolution shown in (2.20), $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ denote the input data, $\{a, b, c, d, e, f, g, h\}$ denote the coefficients, and $\{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8\}$ denote the output data.

$$U = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \\ v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_1 \\ v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_1 & v_2 \\ v_4 & v_5 & v_6 & v_7 & v_8 & v_1 & v_2 & v_3 \\ v_5 & v_6 & v_7 & v_8 & v_1 & v_2 & v_3 & v_4 \\ v_6 & v_7 & v_8 & v_1 & v_2 & v_3 & v_4 & v_5 \\ v_7 & v_8 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\ v_8 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}, \quad (2.20)$$

In the case of cyclic formulation with eight input data, we permute the data on the rows and columns of the matrix such that the resulting matrix of input data shown in (2.21) can be partitioned into the form of block pseudocirculants of 2×2 with circulant blocks of 4×4 .

$$U = \begin{bmatrix} u_1 \\ u_3 \\ u_5 \\ u_7 \\ u_2 \\ u_4 \\ u_6 \\ u_8 \end{bmatrix} = \begin{bmatrix} v_1 & v_3 & v_5 & v_7 & v_2 & v_4 & v_6 & v_8 \\ v_3 & v_5 & v_7 & v_1 & v_4 & v_6 & v_8 & v_2 \\ v_5 & v_7 & v_1 & v_3 & v_6 & v_8 & v_2 & v_4 \\ v_7 & v_1 & v_3 & v_5 & v_8 & v_2 & v_4 & v_6 \\ v_2 & v_4 & v_6 & v_8 & v_3 & v_5 & v_7 & v_1 \\ v_4 & v_6 & v_8 & v_2 & v_5 & v_7 & v_1 & v_3 \\ v_6 & v_8 & v_2 & v_4 & v_7 & v_1 & v_3 & v_5 \\ v_8 & v_2 & v_4 & v_6 & v_1 & v_3 & v_5 & v_7 \end{bmatrix} \cdot \begin{bmatrix} a \\ c \\ e \\ g \\ b \\ d \\ f \\ h \end{bmatrix} \quad (2.21)$$

GDAU1 GDAU2

$$= \begin{bmatrix} U_A \\ U_B \end{bmatrix} = \begin{bmatrix} V_A & V_B \\ V_B & V_{A'} \end{bmatrix} \cdot \begin{bmatrix} A \\ B \end{bmatrix}$$

where

$$U_A = \begin{bmatrix} u_1 \\ u_3 \\ u_5 \\ u_7 \end{bmatrix}, \quad U_B = \begin{bmatrix} u_2 \\ u_4 \\ u_6 \\ u_8 \end{bmatrix}, \quad V_A = \begin{bmatrix} v_1 & v_3 & v_5 & v_7 \\ v_3 & v_5 & v_7 & v_1 \\ v_5 & v_7 & v_1 & v_3 \\ v_7 & v_1 & v_3 & v_5 \end{bmatrix}, \quad V_B = \begin{bmatrix} v_2 & v_4 & v_6 & v_8 \\ v_4 & v_6 & v_8 & v_2 \\ v_6 & v_8 & v_2 & v_4 \\ v_8 & v_2 & v_4 & v_6 \end{bmatrix}, \quad V_{B'} = \begin{bmatrix} v_3 & v_5 & v_7 & v_1 \\ v_5 & v_7 & v_1 & v_3 \\ v_7 & v_1 & v_3 & v_5 \\ v_1 & v_3 & v_5 & v_7 \end{bmatrix},$$

$$A = \begin{bmatrix} a \\ c \\ e \\ g \end{bmatrix}, \text{ and } B = \begin{bmatrix} b \\ d \\ f \\ g \end{bmatrix}.$$

From the hardware point of view, we have partitioned the original memory module into two smaller ones. One memory module stores the combination of the coefficients $\{a, c, e, g\}$ for $u_1, u_3, u_5,$ and u_7 as well as $u_2, u_4, u_6,$ and u_8 . Similarly, the other one stores the combination of the coefficients $\{b, d, f, g\}$ for $u_1, u_3, u_5,$ and u_7 as well as $u_2, u_4, u_6,$ and u_8 . In performing the memory access, we can access the partial products for $u_1, u_3, u_5,$ and u_7 by using the memory address generated from $\{v_1, v_3, v_5, v_7\}$ through first memory module and access the partial products for $u_1, u_3, u_5,$ and u_7 by using the memory address generated from $\{v_2, v_4, v_6, v_8\}$ through second memory module at the same time. Then we sum up these two partial products for obtaining $u_1, u_3, u_5,$ and u_7 by using extra adders respectively. With the help of the identical hardware and extra input-data-rotator, we can compute $u_2, u_4, u_6,$ and u_8 in the same way. However, for the operation of input data rotation, in the case of partitioning factors is not relatively co-prime, the number of rotated bit for V_B is larger than V_A by one bit. Fig. 2.20 and Fig. 2.21 show the low-cost and high performance GDA architectures to realize this design example illustrated in (2.21).

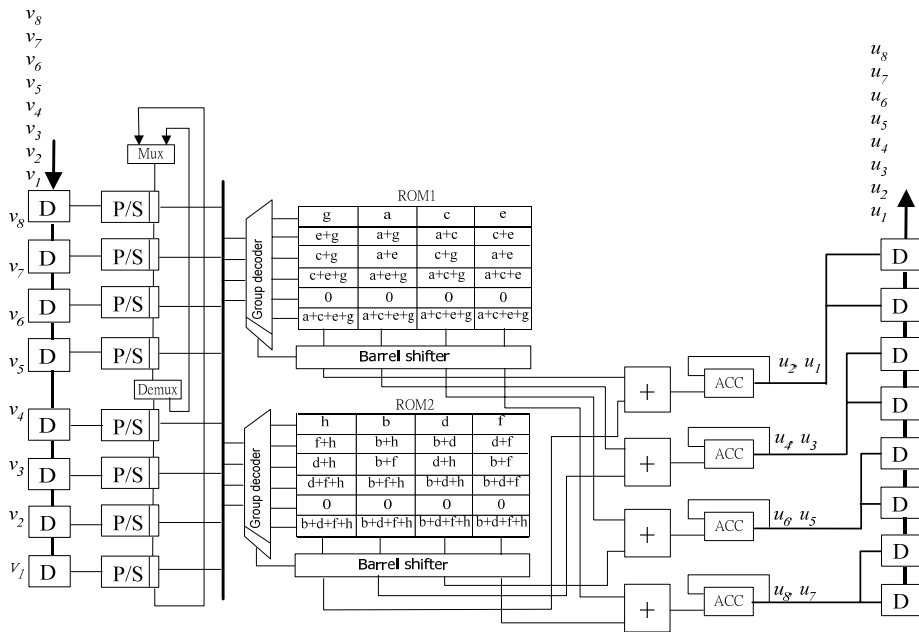


Fig. 2.20: The low cost version of GDA realization of the example shown in (2.21).

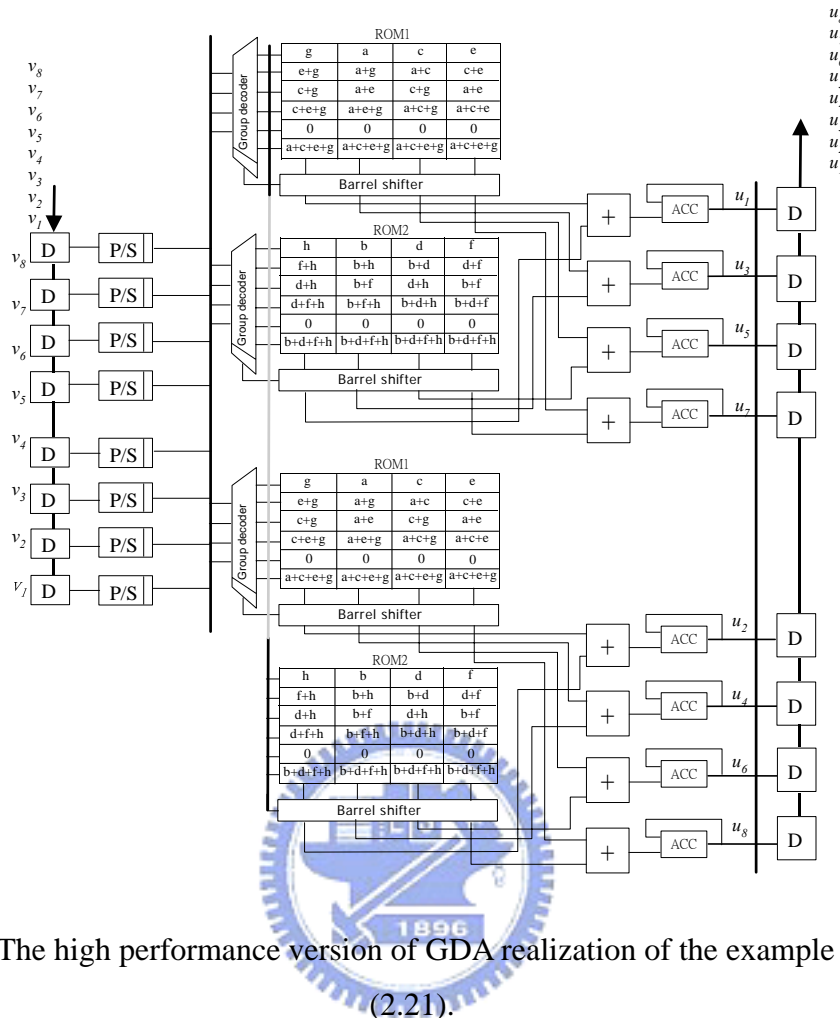


Fig. 2.21: The high performance version of GDA realization of the example shown in (2.21).

2.4.4 Evaluation of long length cyclic convolution GDA design

As shown in Fig. 2.18, the proposed low-cost BGDA design with co-prime only requires two small memory modules to compute all the output samples. It saves 72 words of memory (i.e., 75% of memory cost), 6 adders, and 3 registers at the cost of introducing the extra barrel rotator and input-vector rotator circuitries as well as halving the throughput rate as compared with the traditional DA-based design. For the requirement of high performance, the proposed BGDA design (shown in Fig. 2.19) can save 48 words of memory (i.e., 50% of memory cost) and operate at the same throughput rate as compared with the traditional DA design at the cost of one extra barrel rotator. As to the non-coprime partitioning, the proposed low-cost BGDA design shows in Fig. 2.20, similar to the case of co-prime partitioning, this design only requires two small memory modules to compute all the output samples. It saves

208 words of memory (i.e., 81.25% of memory cost), 8 adders, and 4 registers at the cost of introducing the extra barrel rotator and input-vector rotator circuitries as well as halving the throughput rate as compared with the traditional DA-based design. For the requirement of high performance, the proposed BGDA design (shown in Fig. 2.21) can save 160 words of memory (i.e., 62.5% of memory cost) and operate at the same throughput rate as compared with the traditional DA design at the cost of one extra barrel rotator. Table 2.7 summarizes the hardware cost in the architectures of low-cost BGDA, high performance BGDA, and traditional DA. It is concluded that the proposed BGDA design approach provides a hardware efficient scheme to realize the long-length cyclic convolution.

Table 2.7: Comparison of the hardware cost of the design examples shown in low-cost BGDA, high performance BGDA, and conventional DA in the case of non-coprime

partitioning.

	Address decoder (coded addresses)	memory size (words)	4-bit Barrel shifter (words)	Adder (words)	SR (words)	P/S (words)	Rotator (words)	Normalized Throughput
Conventional DA design	$2*2^4$	256	0	16	24	8	0	1
Proposed BGDA design (high performance version)	$2*(16+2^4)$	96	³ 4	16	24	8	0	1
Proposed BGDA design (low cost version)	$2*(6+4)$	48	2	8	20	8	6	0.5

Note:

- 1 denotes the number of group-address.
- 2 denotes the number of rotate-left factor.
- 3 denotes equivalent area of $4*4^2$ memory words.

Chapter 3

GDA-based Design for 1-D DSST's

In this chapter, we illustrate the GDA-based designs of 1-D DSST's with prime-length and any-length, including DFT, DHT, and DCT, from algorithm to architecture, respectively. The optimizations on algorithm level of DSST's for further reducing the hardware cost are involved. Besides, we have evaluated each of the DSST's designs in the corresponding subsection.

3.1 Design of 1-D DFT

3.1.1 Cyclic Convolution Formulation

Prime-length case

The 1-D N-point DFT of an input sequence $\{x(n), n = 0, 1, \dots, N-1\}$ is defined as

$$Y(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}, k = 0, 1, \dots, N-1 \quad (3.1)$$

If N is prime, we can rewrite (3.1) in a cyclic convolution by exploiting the property of input/output (I/O) data permutation as

$$Y(0) = \sum_{n=0}^{N-1} x(n) \quad (3.2)$$

$$\begin{aligned} Y((g^k)_N) &= \left[\sum_{n=1}^{N-1} x((g^{n-k})_N) \cdot W_N^{((g^n)_N)} \right] + x(0); k = 1, \dots, N-1 \\ &= T((g^k)_N) + x(0) \end{aligned} \quad (3.3)$$

$$T((g^k)_N) = \sum_{n=1}^{N-1} x((g^{n-k})_N) \cdot W_N^{(g^n)_N} \quad (3.4)$$

, where $(g^k)_N$ denotes the result of “ g^k modulo N ” for short and g is a primitive element. $T((g^k)_N)$ in (3.4) is the kernel of the N-point DFT that is written in cyclic convolution formulation. For facilitating the utilization of the GDA design approach, the GDA formulation of $T((g^k)_N)$ shows as

$$T((g^k)_N) = -T_0((g^{k+R_0})_N) + \sum_{q=1}^{L-1} T_q((g^{k+R_q})_N) \cdot 2^{-q} \quad (3.5)$$

where $\overline{T}_0(R_0) = R(\overline{T}_0)$,

$$\overline{T}_0(R_0) = \{T_0((g^{1+R_0})_N), T_0((g^{2+R_0})_N), \dots, T_0((g^{k+R_0})_N), \dots, T_0((g^{N+R_0})_N)\},$$

$$\overline{T}_0 = \{T_0((g^1)_N), T_0((g^2)_N), \dots, T_0((g^k)_N), \dots, T_0((g^N)_N)\}$$

and

$$\overline{T}_q(R_q) = R(\overline{T}_q),$$

$$\overline{T}_q(R_q) = \{T_q((g^{1+R_q})_N), T_q((g^{2+R_q})_N), \dots, T_q((g^{k+R_q})_N), \dots, T_q((g^{N+R_q})_N)\},$$

$$\overline{T}_q = \{T_q((g^1)_N), T_q((g^2)_N), \dots, T_q((g^k)_N), \dots, T_q((g^N)_N)\},$$

and

$$T_0((g^k)_N) = \sum_{n=1}^{N-1} x_0((g^{n-k-R_0})_N) \cdot W_N^{(g^n)_N} \text{ and}$$

$$T_q((g^k)_N) = \sum_{n=1}^{N-1} x_q((g^{n-k-R_q})_N) \cdot W_N^{(g^n)_N}.$$

where L denotes the data word length of the variable x , N denotes the transform length, R_q denotes the rotating factor for q th bit that is used for indicating the number of position of the partial products in DA input and output should be rotated, and $W_N^{(g^n)_N}$ are the DFT coefficients. The rotation function $R(\cdot)$ is used to rotate the elements in the output vector $\overline{T}_q(R_q)$ from the input vector \overline{T}_q by R_q for the q th bit of DA computation.

Non-prime length case

For the case of non-prime length, the 1-D N -point DFT of an input sequence $\{y(n), n=0, 1, \dots, N-1\}$ is defined as

$$Y(k) = \sum_{n=0}^{N-1} y(n) \cdot W_N^{nk}; k = 0, 1, \dots, N-1, \quad (3.6)$$

where W_N^{nk} denotes $e^{\frac{-2nk\pi}{N}}$.

Using the identity

$$n \times k = \frac{1}{2} \times [n^2 + k^2 - (n - k)^2] \quad (3.7)$$

we can express (3.6) as

$$Y(k) = W_N^{\frac{k^2}{2}} \times T(k); k = 0, 1, \dots, N - 1, \quad (3.8)$$

where

$$T(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{\frac{-1}{2}(n-k)^2}; k = 0, 1, \dots, N - 1, \quad (3.9)$$

and

$$x(n) = y(n) \times W_N^{\frac{1}{2}n^2}. \quad (3.10)$$

The $T(k)$ in (3.8) is expressed as a cyclic convolution. To facilitate the GDA design of $T(k)$, we expressed $T(k)$ in a commutative form as

$$T(k) = \sum_{n=0}^{N-1} x((n+k)_N) \cdot W_N^{\frac{-1}{2}(n)^2} \quad (3.11)$$

$W_N^{\frac{1}{2}n^2}$ in (3.10) denotes the complex multiplication for the input sample, and the $W_N^{\frac{k^2}{2}}$ in (3.8) denotes the complex multiplication for the result of cyclic convolution operation. Hence the extra pre-processing and post-processing are needed for the cyclic convolution of any length DFT. Since the GDA design is based on bit serial approach, with the stage-balance point of view in pipeline architecture, the CORDIC (CO-ordinate Rotation Digital Computer) complex multiplier should be an proper combination. The detail of CORDIC is illustrated as the following.

3.1.2 CORDIC (CO-ordinate Rotation Digital Computer)

For properly combining with the feature of bit-serial in DA computation, we hope to realize the complex multiplication in serial manner for pre-processing and post-processing of the DFT in cyclic convolution. The existing realizations of complex multiplication have either direct manner or rotated transformation algorithm.

The realization of complex multiplication with direct manner needs four multipliers and two adders, but realization with the rotated transformation algorithm, such as the CORDIC, needs only a sequence of identical arithmetic shift-and-addition operations. With the feature of serial manner, CORDIC should be a proper choice of serial complex multiplication for low hardware cost in the bit-level design. So, combining the GDA approach with CORDIC facilitates a hardware efficient design for any-length cyclic convolution DFT.

The CORDIC was developed by Volder in 1959 as a technique for solving the coordinate rotation problem [47] and later generalized to solve other elementary functions by Walther [48]. It can be applied to the rotations in three coordinates systems: the linear, circular, and hyperbolic coordinate systems. A complex multiplication with the rotation operation in the circular mode can be shown as (3.12). The basic concept of CORDIC computation is to decompose the desired rotation angle of coefficient into the weighted sum of a set of predefined elementary rotation angles in (3.13) so that the rotation through each of them can be accomplished with simple shift-and-add operations for two stages. As shown in Fig. 3.1, the architecture design of CORDIC is more hardware efficient than the direct realization of complex multiplications, which needs four multipliers and two adders.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R(\theta) \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}. \quad (3.12)$$

Where $[x, y]$ denotes the input vector with real part of x and imaginary part of y . $R(\theta)$ denotes the complex coefficient to be multiplied.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R(\theta) \times \begin{bmatrix} x \\ y \end{bmatrix} = K_m \cdot \prod_{i=0}^{m-1} \begin{bmatrix} 1 & s_i 2^{-i} \\ -s_i 2^{-i} & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} \quad (3.13)$$

Where $K_m = \prod_{i=0}^{m-1} \cos(\theta_i)$

Table 3.1: Table for θ_i

i	θ_i (degree)
0	45
1	26.56
2	14.03
3	7.12
4	3.58
5	1.79
6	0.89
7	0.45
8	0.22
9	0.11
10	0.06

Table 3.2: Determination of the s_i sequence
at the θ of 56.

i	s_i	$Sum(\theta_i)$
0	1	45
1	1	71.5
2	-1	57.5
3	-1	50.4
4	1	53.9
5	1	55.7
6	1	56.6
7	-1	56.2
8	-1	56

Since the DFT algorithm θ has been given, with the table for $\theta_{i in}$ in Table 3.1, the corresponding set of s_i can be computed and stored in memory in advance. Table 3.2 shows the example to determine the sequence of s_i at the θ of 56. In the two stages computation of CORDIC, the multiplication of the scaling factor in second stage imposes significant overhead. Fortunately, if $|s_i|$ equals 1, and i is given, K_m can be computed in advance, and converted into a canonical sign-digit representation [49] as (3.14) so that the same processing unit shown in Fig. 3.1 can be used for the two stages of CORDIC computation.

$$K_m = \sum_{p=1}^p k_p 2^{-i_p} \quad (3.14)$$

where $k_p = \pm 1$, i_p are positive integers. Multiplication for scaling then will take $p-1$ shift-and-add operations.

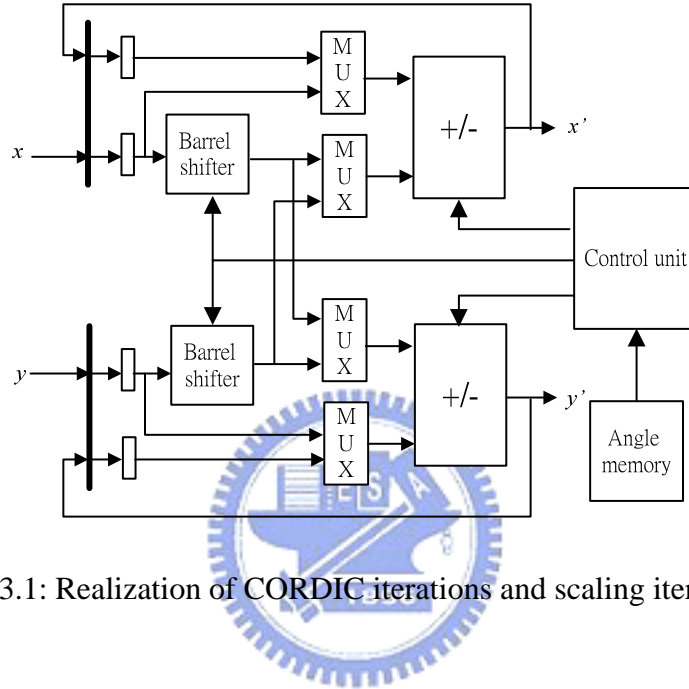


Fig. 3.1: Realization of CORDIC iterations and scaling iterations.

Hardware cost analysis of the complex multiplication realization with direct manner and CORDIC is addressed as the following. Table 3.3 shows the comparison of the hardware cost for the two realizations. In the direct realization, since the two product terms are respectively formed of the real part and imaginary part such that $L-1$ shift operations and $L-1$ accumulation operations are needed for each computation of the product term. Then $4(L-1)$ shift operations and $4(L-1)$ accumulation operations are needed for the complex multiplication, where the parameter L denotes the word length. For the CORDIC realization, $2m$ shift and additions operations are needed for the first stage, and $2(p-1)$ shift and accumulation operations are needed for the second stage. Consequently, the total number of shift and accumulation operations needed for the direct realization are $4(L-1)$ and $4(L-1)$ as well as $2m+2(p-1)$ and $2m+2(p-1)$ for the CORIC realization. Additionally, two additional additions in the direct realization are needed for summing up two terms of real part and imaginary part for output. In general, the word length of the input value is larger than the number of iteration in

each of the CORDIC stages. With the UMC 0.18um cell-library and the same constrained speed, Fig. 3.2 shows the comparison of area cost and power consumption for the complex multiplications realized with serial multiplier and CORDIC, respectively. As a result of the simulation result, the CORDIC realization should be better than the direct realization for hardware cost.

Table 3.3: Hardware cost comparison of direct realization and CORDIC realization for a complex multiplication.

	<i>Shift</i>	<i>Accumulation</i>	<i>Adder</i>
Direct manner	$4(L-1)$	$4(L-1)$	2
CORDIC	$2m+2(p-1)$	$2m+2(p-1)$	0

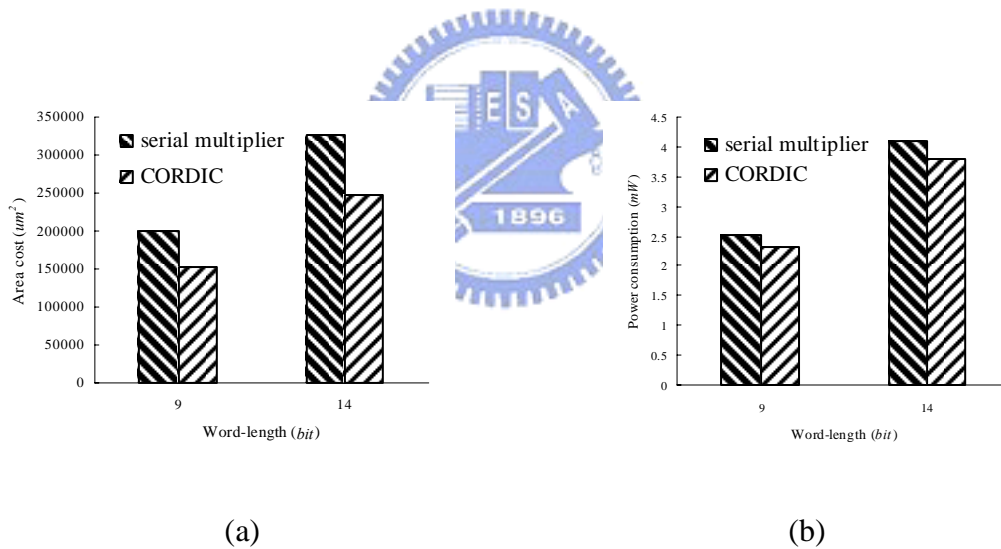


Fig. 3.2: Comparison of (a) area cost and (b) power consumption for the complex multiplications realized with serial multiplier and CORDIC.

3.1.3 Symmetry exploration of the DFT in cyclic convolution

Let us take an example of 1-D 11-point DFT with the real input sequence $\{x(n), n=0, 1, \dots, 10\}$. The cyclic convolution form of $T((g^k)_N)$ can be expressed as

$$\begin{bmatrix} T(2) \\ T(4) \\ T(8) \\ T(5) \\ T(10) \\ T(9) \\ T(7) \\ T(3) \\ T(6) \\ T(1) \end{bmatrix} = \begin{bmatrix} x(1) & x(2) & x(4) & x(8) & x(5) & x(10) & x(9) & x(7) & x(3) & x(6) \\ x(6) & x(1) & x(2) & x(4) & x(8) & x(5) & x(10) & x(9) & x(7) & x(3) \\ x(3) & x(6) & x(1) & x(2) & x(4) & x(8) & x(5) & x(10) & x(9) & x(7) \\ x(7) & x(3) & x(6) & x(1) & x(2) & x(4) & x(8) & x(5) & x(10) & x(9) \\ x(9) & x(7) & x(3) & x(6) & x(1) & x(2) & x(4) & x(8) & x(5) & x(10) \\ x(10) & x(9) & x(7) & x(3) & x(6) & x(1) & x(2) & x(4) & x(8) & x(5) \\ x(5) & x(10) & x(9) & x(7) & x(3) & x(6) & x(1) & x(2) & x(4) & x(8) \\ x(8) & x(5) & x(10) & x(9) & x(7) & x(3) & x(6) & x(1) & x(2) & x(4) \\ x(4) & x(8) & x(5) & x(10) & x(9) & x(7) & x(3) & x(6) & x(1) & x(2) \\ x(2) & x(4) & x(8) & x(5) & x(10) & x(9) & x(7) & x(3) & x(6) & x(1) \end{bmatrix} \times \begin{bmatrix} W_{11}^2 \\ W_{11}^4 \\ W_{11}^8 \\ W_{11}^5 \\ W_{11}^{10} \\ W_{11}^9 \\ W_{11}^7 \\ W_{11}^3 \\ W_{11}^6 \\ W_{11}^1 \end{bmatrix}. \quad (3.15)$$

As shown in (3.11), the coefficient matrix in (3.15) can be expanded as the even symmetries of cosine function $c_N^i = c_N^{N-i}, i=1,2,\dots,N-1$, where $c_N^i = \cos(2i\pi/N)$, and the odd symmetries of sine function $s_N^i = -s_N^{N-i}, i=1,2,\dots,N-1$, where $s_N^i = \sin(2i\pi/N)$.

$$\begin{bmatrix} W_{11}^2 \\ W_{11}^7 \\ W_{11}^8 \\ W_{11}^6 \\ W_{11}^{10} \\ W_{11}^9 \\ W_{11}^4 \\ W_{11}^3 \\ W_{11}^5 \\ W_{11}^1 \end{bmatrix} = \begin{bmatrix} c_{11}^2 - js_{11}^2 \\ c_{11}^7 - js_{11}^7 \\ c_{11}^8 - js_{11}^8 \\ c_{11}^6 - js_{11}^6 \\ c_{11}^{10} - js_{11}^{10} \\ c_{11}^9 - js_{11}^9 \\ c_{11}^4 - js_{11}^4 \\ c_{11}^3 - js_{11}^3 \\ c_{11}^5 - js_{11}^5 \\ c_{11}^1 - js_{11}^1 \end{bmatrix} = \begin{bmatrix} c_{11}^2 - js_{11}^2 \\ c_{11}^4 + js_{11}^4 \\ c_{11}^8 - js_{11}^8 \\ c_{11}^6 - js_{11}^6 \\ c_{11}^{10} - js_{11}^{10} \\ c_{11}^2 + js_{11}^2 \\ c_{11}^4 - js_{11}^4 \\ c_{11}^8 + js_{11}^8 \\ c_{11}^6 + js_{11}^6 \\ c_{11}^{10} + js_{11}^{10} \end{bmatrix} \quad (3.16)$$

Then, we can re-write $T((2^k)_{11})$ in (3.15) as follows:

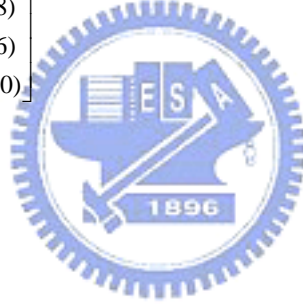
$$\begin{aligned}
& \begin{bmatrix} T(2) \\ T(7) \\ T(8) \\ T(6) \\ T(10) \\ T(9) \\ T(4) \\ T(3) \\ T(5) \\ T(1) \end{bmatrix} = \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) & x(10) & x(2) & x(7) & x(8) & x(6) \\ x(5) & x(1) & x(9) & x(4) & x(3) & x(6) & x(10) & x(2) & x(7) & x(8) \\ x(3) & x(5) & x(1) & x(9) & x(4) & x(8) & x(6) & x(10) & x(2) & x(7) \\ x(4) & x(3) & x(5) & x(1) & x(9) & x(7) & x(8) & x(6) & x(10) & x(2) \\ x(9) & x(4) & x(3) & x(5) & x(1) & x(2) & x(7) & x(8) & x(6) & x(10) \\ x(10) & x(2) & x(7) & x(8) & x(6) & x(1) & x(9) & x(4) & x(3) & x(5) \\ x(6) & x(10) & x(2) & x(7) & x(8) & x(5) & x(1) & x(9) & x(4) & x(3) \\ x(8) & x(6) & x(10) & x(2) & x(7) & x(3) & x(5) & x(1) & x(9) & x(4) \\ x(7) & x(8) & x(6) & x(10) & x(2) & x(4) & x(3) & x(5) & x(1) & x(9) \\ x(2) & x(7) & x(8) & x(6) & x(10) & x(9) & x(4) & x(3) & x(5) & x(1) \end{bmatrix} \times \begin{bmatrix} c_{11}^2 - js_{11}^2 \\ c_{11}^4 + js_{11}^4 \\ c_{11}^8 - js_{11}^8 \\ c_{11}^6 - js_{11}^6 \\ c_{11}^{10} - js_{11}^{10} \\ c_{11}^2 + js_{11}^2 \\ c_{11}^4 - js_{11}^4 \\ c_{11}^8 + js_{11}^8 \\ c_{11}^6 + js_{11}^6 \\ c_{11}^{10} + js_{11}^{10} \end{bmatrix} \\
& = \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) & x(10) & x(2) & x(7) & x(8) & x(6) \\ x(5) & x(1) & x(9) & x(4) & x(3) & x(6) & x(10) & x(2) & x(7) & x(8) \\ x(3) & x(5) & x(1) & x(9) & x(4) & x(8) & x(6) & x(10) & x(2) & x(7) \\ x(4) & x(3) & x(5) & x(1) & x(9) & x(7) & x(8) & x(6) & x(10) & x(2) \\ x(9) & x(4) & x(3) & x(5) & x(1) & x(2) & x(7) & x(8) & x(6) & x(10) \\ x(1) & x(9) & x(4) & x(3) & x(5) & x(10) & x(2) & x(7) & x(8) & x(6) \\ x(5) & x(1) & x(9) & x(4) & x(3) & x(6) & x(10) & x(2) & x(7) & x(8) \\ x(3) & x(5) & x(1) & x(9) & x(4) & x(8) & x(6) & x(10) & x(2) & x(7) \\ x(4) & x(3) & x(5) & x(1) & x(9) & x(7) & x(8) & x(6) & x(10) & x(2) \\ x(9) & x(4) & x(3) & x(5) & x(1) & x(2) & x(7) & x(8) & x(6) & x(10) \end{bmatrix} \begin{bmatrix} c_{11}^2 \\ c_{11}^4 \\ c_{11}^8 \\ c_{11}^6 \\ c_{11}^{10} \\ c_{11}^2 \\ c_{11}^4 \\ c_{11}^8 \\ c_{11}^6 \\ c_{11}^{10} \end{bmatrix} \\
& - j \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) & x(10) & x(2) & x(7) & x(8) & x(6) \\ x(5) & x(1) & x(9) & x(4) & x(3) & x(6) & x(10) & x(2) & x(7) & x(8) \\ x(3) & x(5) & x(1) & x(9) & x(4) & x(8) & x(6) & x(10) & x(2) & x(7) \\ x(4) & x(3) & x(5) & x(1) & x(9) & x(7) & x(8) & x(6) & x(10) & x(2) \\ x(9) & x(4) & x(3) & x(5) & x(1) & x(2) & x(7) & x(8) & x(6) & x(10) \\ -x(1) & -x(9) & -x(4) & -x(3) & -x(5) & -x(10) & -x(2) & -x(7) & -x(8) & -x(6) \\ -x(5) & -x(1) & -x(9) & -x(4) & -x(3) & -x(6) & -x(10) & -x(2) & -x(7) & -x(8) \\ -x(3) & -x(5) & -x(1) & -x(9) & -x(4) & -x(8) & -x(6) & -x(10) & -x(2) & -x(7) \\ -x(4) & -x(3) & -x(5) & -x(1) & -x(9) & -x(7) & -x(8) & -x(6) & -x(10) & -x(2) \\ -x(9) & -x(4) & -x(3) & -x(5) & -x(1) & -x(2) & -x(7) & -x(8) & -x(6) & -x(10) \end{bmatrix} \begin{bmatrix} s_{11}^2 \\ -s_{11}^4 \\ s_{11}^8 \\ s_{11}^6 \\ s_{11}^{10} \\ -s_{11}^2 \\ +s_{11}^4 \\ -s_{11}^8 \\ -s_{11}^6 \\ -s_{11}^{10} \end{bmatrix} \tag{3.17}
\end{aligned}$$

From (3.17), we see that

$$\begin{bmatrix} T_R(2) \\ T_R(7) \\ T_R(8) \\ T_R(6) \\ T_R(10) \end{bmatrix} = \begin{bmatrix} T_R(9) \\ T_R(4) \\ T_R(3) \\ T_R(5) \\ T_R(1) \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} T_I(2) \\ T_I(7) \\ T_I(8) \\ T_I(6) \\ T_I(10) \end{bmatrix} = - \begin{bmatrix} T_I(9) \\ T_I(4) \\ T_I(3) \\ T_I(5) \\ T_I(1) \end{bmatrix}. \tag{3.18}$$

Then, we can respectively express $T_R(\cdot)$ and $T_I(\cdot)$ in (14) as

$$\begin{aligned}
\begin{bmatrix} T_R(2) \\ T_R(7) \\ T_R(8) \\ T_R(6) \\ T_R(10) \end{bmatrix} &= \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) & x(10) & x(2) & x(7) & x(8) & x(6) \\ x(5) & x(1) & x(9) & x(4) & x(3) & x(6) & x(10) & x(2) & x(7) & x(8) \\ x(3) & x(5) & x(1) & x(9) & x(4) & x(8) & x(6) & x(10) & x(2) & x(7) \\ x(4) & x(3) & x(5) & x(1) & x(9) & x(7) & x(8) & x(6) & x(10) & x(2) \\ x(9) & x(4) & x(3) & x(5) & x(1) & x(2) & x(7) & x(8) & x(6) & x(10) \end{bmatrix} \cdot \begin{bmatrix} c_{11}^2 \\ c_{11}^4 \\ c_{11}^8 \\ c_{11}^6 \\ c_{11}^{10} \\ c_{11}^2 \\ c_{11}^4 \\ c_{11}^8 \\ c_{11}^6 \\ c_{11}^{10} \end{bmatrix} \\
&= \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) \\ x(5) & x(1) & x(9) & x(4) & x(3) \\ x(3) & x(5) & x(1) & x(9) & x(4) \\ x(4) & x(3) & x(5) & x(1) & x(9) \\ x(9) & x(4) & x(3) & x(5) & x(1) \end{bmatrix} \cdot \begin{bmatrix} c_{11}^2 \\ c_{11}^4 \\ c_{11}^8 \\ c_{11}^6 \\ c_{11}^{10} \end{bmatrix} + \begin{bmatrix} x(10) & x(2) & x(7) & x(8) & x(6) \\ x(6) & x(10) & x(2) & x(7) & x(8) \\ x(8) & x(6) & x(10) & x(2) & x(7) \\ x(7) & x(8) & x(6) & x(10) & x(2) \\ x(2) & x(7) & x(8) & x(6) & x(10) \end{bmatrix} \cdot \begin{bmatrix} c_{11}^2 \\ c_{11}^4 \\ c_{11}^8 \\ c_{11}^6 \\ c_{11}^{10} \end{bmatrix} \\
&= \begin{bmatrix} T_{R1}(2) \\ T_{R1}(7) \\ T_{R1}(8) \\ T_{R1}(6) \\ T_{R1}(10) \end{bmatrix} + \begin{bmatrix} T_{R2}(2) \\ T_{R2}(7) \\ T_{R2}(8) \\ T_{R2}(6) \\ T_{R2}(10) \end{bmatrix}
\end{aligned} \tag{3.19}$$



$$\begin{aligned}
\begin{bmatrix} T_I(2) \\ T_I(7) \\ T_I(8) \\ T_I(6) \\ T_I(10) \end{bmatrix} &= \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) & x(10) & x(2) & x(7) & x(8) & x(6) \\ x(5) & x(1) & x(9) & x(4) & x(3) & x(6) & x(10) & x(2) & x(7) & x(8) \\ x(3) & x(5) & x(1) & x(9) & x(4) & x(8) & x(6) & x(10) & x(2) & x(7) \\ x(4) & x(3) & x(5) & x(1) & x(9) & x(7) & x(8) & x(6) & x(10) & x(2) \\ x(9) & x(4) & x(3) & x(5) & x(1) & x(2) & x(7) & x(8) & x(6) & x(10) \end{bmatrix} \cdot \begin{bmatrix} -s_{11}^2 \\ s_{11}^4 \\ -s_{11}^8 \\ -s_{11}^6 \\ -s_{11}^{10} \\ -s_{11}^2 \\ s_{11}^4 \\ -s_{11}^8 \\ -s_{11}^6 \\ -s_{11}^{10} \end{bmatrix} \\
&= \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) \\ x(5) & x(1) & x(9) & x(4) & x(3) \\ x(3) & x(5) & x(1) & x(9) & x(4) \\ x(4) & x(3) & x(5) & x(1) & x(9) \\ x(9) & x(4) & x(3) & x(5) & x(1) \end{bmatrix} \cdot \begin{bmatrix} -s_{11}^2 \\ s_{11}^4 \\ -s_{11}^8 \\ -s_{11}^6 \\ -s_{11}^{10} \end{bmatrix} - \begin{bmatrix} x(10) & x(2) & x(7) & x(8) & x(6) \\ x(6) & x(10) & x(2) & x(7) & x(8) \\ x(8) & x(6) & x(10) & x(2) & x(7) \\ x(7) & x(8) & x(6) & x(10) & x(2) \\ x(2) & x(7) & x(8) & x(6) & x(10) \end{bmatrix} \cdot \begin{bmatrix} -s_{11}^2 \\ s_{11}^4 \\ -s_{11}^8 \\ -s_{11}^6 \\ -s_{11}^{10} \end{bmatrix} \\
&= \begin{bmatrix} T_{I1}(2) \\ T_{I1}(7) \\ T_{I1}(8) \\ T_{I1}(6) \\ T_{I1}(10) \end{bmatrix} - \begin{bmatrix} T_{I2}(2) \\ T_{I2}(7) \\ T_{I2}(8) \\ T_{I2}(6) \\ T_{I2}(10) \end{bmatrix}
\end{aligned} \tag{3.20}$$

Observing (3.19) and (3.20), we find that the real part of $T((2^k)_{II})$ is composed of the same upper and lower halves, and the imaginary part of $T((2^k)_{II})$ is composed of the upper and the lower halves with the same absolute value, but different signs. Hence, only the unique constant multiplications in $\{T(i), i=1, 2, \dots, (N-1)/2\}$ need to be calculated. Therefore, we can calculate two output values simultaneously through (3.17) with the same hardware. This feature facilitates the hardware sharing in computing $T((g^k)_N)$ with even and odd indices such that only half the hardware is needed as compared with the direct realization on (3.15).

3.1.4 Architecture design and evaluation

Architecture design

By exploiting the symmetrical properties of both the cosine and sine functions shown in (3.17) in the DFT computation, we find that the output with odd indices can easily be obtained by means of hardwiring, which facilitates the reduction of memory cost by a factor of two. Considering the example of 1-D 11-point DFT and referring to the reformulation of 1-D DFT in (3.19) and (3.20), we can realize the 10-point cyclic convolution required in 1-D 11-point DFT through the hardware architectures designed for the 5-point cyclic convolution as shown in Fig. 3.3. The proposed GDA architecture is composed of the group distributed arithmetic units (GDAU), address decoder, adders/subtractors, accumulators, and parallel-to-serial (P/S) converters. According to the rule of group mapping shown in Table 2.1, the candidate of DA input $X_q = \{x_q(1), x_q(9), x_q(4), x_q(3), x_q(5)\}$ or $\{x_q(10), x_q(2), x_q(7), x_q(8), x_q(6)\}$ is first fed into the address decoder to determine which group it should belong to, and then compute the group address $G_q = \{g_q(1), g_q(2), g_q(3)\}$ and the rotating factor $R_q = \{r_q(1), r_q(2), r_q(3)\}$ used for the GDAU. The GDAUc and GDAUs are used to respectively realize the operations specified in (3.19) and (3.20) for computing 5-point cyclic convolution. The contents of the memory modules corresponding to GDAUc and GDAUs are shown in Table 3.4 and Table 3.5 respectively, which illustrate the distribution of the partial products when computing different DFT outputs according to the candidate of DA input.

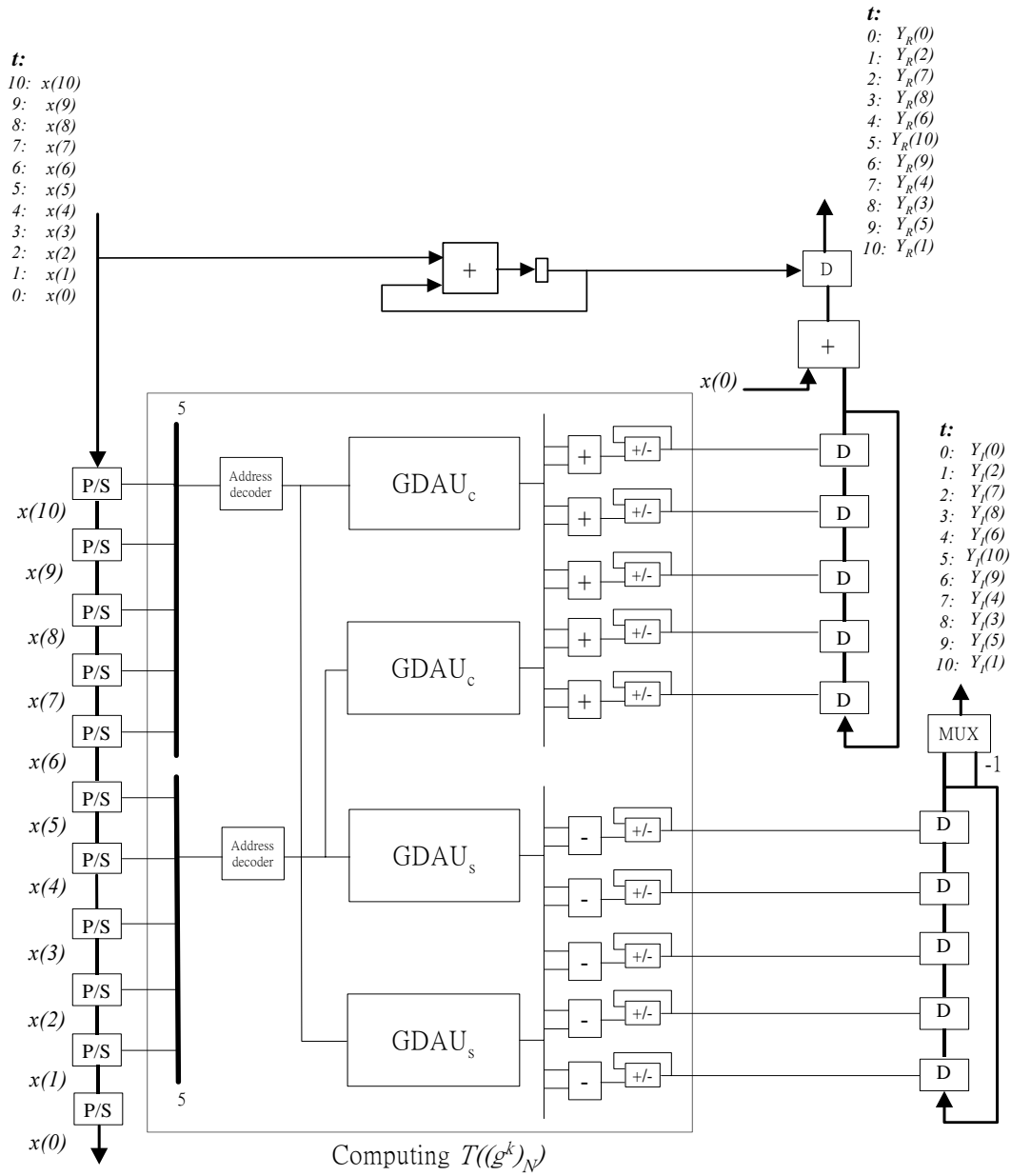


Fig. 3.3: Architecture design of the 1-D 11-point DFT with GDA approach.

Table 3.4: The 8 groups of memory content used for computing the 5-point cyclic convolution in GDAUc.

Grouped candidates of DA input (X_q)	Group address (G_q)	$T_{R1}(2)/T_{R2}(2)/T_{R1}(9)/T_{R2}(9)$	$T_{R1}(7)/T_{R2}(7)/T_{R1}(4)/T_{R2}(4)$	$T_{R1}(8)/T_{R2}(8)/T_{R1}(3)/T_{R2}(3)$	$T_{R1}(6)/T_{R2}(6)/T_{R1}(5)/T_{R2}(5)$	$T_{R1}(10)/T_{R2}(10)/T_{R1}(1)/T_{R2}(1)$
0	0	0	0	0	0	0
1, 2, 4, 8, 16	1	c_{11}^{10}	c_{11}^2	c_{11}^4	c_{11}^8	c_{11}^6
3, 6, 12, 24, 17	2	$c_{11}^6+c_{11}^{10}$	$c_{11}^{10}+c_{11}^2$	$c_{11}^2+c_{11}^4$	$c_{11}^4+c_{11}^8$	$c_{11}^8+c_{11}^6$
5, 10, 20, 9, 18	3	$c_{11}^8+c_{11}^{10}$	$c_{11}^6+c_{11}^2$	$c_{11}^{10}+c_{11}^4$	$c_{11}^2+c_{11}^8$	$c_{11}^4+c_{11}^6$
7, 14, 28, 25, 19	4	$c_{11}^8+c_{11}^6+c_{11}^{10}$	$c_{11}^6+c_{11}^{10}+c_{11}^2$	$c_{11}^{10}+c_{11}^2+c_{11}^4$	$c_{11}^2+c_{11}^4+c_{11}^8$	$c_{11}^4+c_{11}^8+c_{11}^6$
11, 22, 13, 26, 21	5	$c_{11}^4+c_{11}^6+c_{11}^{10}$	$c_{11}^2+c_{11}^8+c_{11}^{10}$	$c_{11}^2+c_{11}^4+c_{11}^6$	$c_{11}^4+c_{11}^8+c_{11}^{10}$	$c_{11}^2+c_{11}^8+c_{11}^6$
15, 30, 29, 27, 23	6	$c_{11}^4+c_{11}^8+c_{11}^6+c_{11}^{10}$	$c_{11}^8+c_{11}^6+c_{11}^2+c_{11}^{10}$	$c_{11}^6+c_{11}^{10}+c_{11}^2+c_{11}^4$	$c_{11}^{10}+c_{11}^2+c_{11}^4+c_{11}^8$	$c_{11}^2+c_{11}^4+c_{11}^8+c_{11}^6$
31	7	$c_{11}^2+c_{11}^4+c_{11}^6+c_{11}^{10}$	$c_{11}^2+c_{11}^4+c_{11}^8+c_{11}^{10}$	$c_{11}^2+c_{11}^4+c_{11}^6+c_{11}^8+c_{11}^{10}$	$c_{11}^2+c_{11}^4+c_{11}^8+c_{11}^6+c_{11}^{10}$	$c_{11}^2+c_{11}^4+c_{11}^8+c_{11}^6+c_{11}^{10}$

Table 3.5: The 8 groups of memory content used for computing the 5-point cyclic convolution in GDAUs.

Grouped candidates of DA input (X_q)	Group address (G_q)	$T_{11}(2)/T_{12}(2)/T_{11}(9)/T_{12}(9)$	$T_{11}(7)/T_{12}(7)/T_{11}(4)/T_{12}(4)$	$T_{11}(8)/T_{12}(8)/T_{11}(3)/T_{12}(3)$	$T_{11}(6)/T_{12}(6)/T_{11}(5)/T_{12}(5)$	$T_{11}(10)/T_{12}(10)/T_{11}(1)/T_{12}(1)$
0	0	0	0	0	0	0
1, 2, 4, 8, 16	1	$-s_{11}^{10}$	$-s_{11}^2$	s_{11}^4	$-s_{11}^8$	$-s_{11}^6$
3, 6, 12, 24, 17	2	$-s_{11}^6-s_{11}^{10}$	$-s_{11}^{10}-s_{11}^2$	$-s_{11}^2+s_{11}^4$	$s_{11}^4-s_{11}^8$	$-s_{11}^8-s_{11}^6$
5, 10, 20, 9, 18	3	$-s_{11}^8-s_{11}^{10}$	$-s_{11}^6-s_{11}^2$	$-s_{11}^{10}+s_{11}^4$	$-s_{11}^2-s_{11}^8$	$s_{11}^4-s_{11}^6$
7, 14, 28, 25, 19	4	$-s_{11}^8-s_{11}^6-s_{11}^{10}$	$-s_{11}^6-s_{11}^{10}-s_{11}^2$	$-s_{11}^{10}-s_{11}^2+s_{11}^4$	$-s_{11}^2+s_{11}^4-s_{11}^8$	$s_{11}^4-s_{11}^8-s_{11}^6$
11, 22, 13, 26, 21	5	$s_{11}^4-s_{11}^6-s_{11}^{10}$	$-s_{11}^2-s_{11}^8-s_{11}^{10}$	$-s_{11}^2+s_{11}^4-s_{11}^6$	$s_{11}^4-s_{11}^8-s_{11}^{10}$	$-s_{11}^2-s_{11}^8-s_{11}^6$
15, 30, 29, 27, 23	6	$s_{11}^4-s_{11}^8-s_{11}^{10}$	$-s_{11}^8-s_{11}^6-s_{11}^{10}$	$-s_{11}^6-s_{11}^{10}$	$-s_{11}^{10}-s_{11}^2$	$-s_{11}^2+s_{11}^4$
31	7	$-s_{11}^2+s_{11}^4-s_{11}^8-s_{11}^6-s_{11}^{10}$	$-s_{11}^8-s_{11}^6-s_{11}^{10}$	$-s_{11}^2+s_{11}^4-s_{11}^8-s_{11}^6-s_{11}^{10}$	$-s_{11}^2+s_{11}^4-s_{11}^8-s_{11}^6-s_{11}^{10}$	$-s_{11}^2+s_{11}^4-s_{11}^8-s_{11}^6-s_{11}^{10}$

Design evaluation

In this section, we will illustrate the performance evaluation on the proposed GDA design and some existing DFT designs. The existing DFT designs in the evaluation include systolic array designs [10][33], memory-based DA designs [34][35], and adder-based DA designs [38]. For a fair comparison, we evaluate the hardware cost and average cycle time (ACT) of these existing designs and the proposed design based on Avant 0.35 μm , 3.3-volt CMOS cell-library [43]. Besides, we adopt the logic synthesis to obtain the measures of hardware cost and ACT for the component whose measures cannot be found from the cell-library, such as the address decoders, the specific memory cells, and RAM cells. According to the measures of area cost and ACT for the used components, we can fairly evaluate the performance of these designs in terms of delay-area products with respect to different values of N.

Table 3.6 and Table 3.7 respectively show the models to estimate the area cost of 1-D N-point DFT modules with or without partitioned cyclic convolution. Table 3.8 shows the corresponding models to estimate the ACT for the existing systolic arrays, DA-based designs, and the proposed GDA design with real input data. The ACT denotes the time needed to perform a 1-D N-point DFT. Besides, we carefully decide the data word-length of the components for evaluating the different designs, respectively.

In the case of 8-bit real input data and complex coefficients, the existing systolic array design [10] requires $2(N+1)$ PEs to process the real part and imaginary part of 1-D N-point DFT, where each PE requires one 16-bit multiplier, one 20-bit adder, one 8-bit register, and two 20-bit registers. The design in [33] is a memory-based systolic array design, which uses a different way to implement the multipliers. It needs an 8-bit multiplexer and demultiplexer for the preprocessing, and each PE is composed of two 12-bit memories, 8-bit 2-to-1 multiplexers, and one 20-bit adder. The designs in [34][35] are the DA-based designs. The design [35] uses the technique of offset binary coding (OBC) to reduce the memory size required in the design [34]. Due to the fact that the two designs are constructed by the same DA architecture, they are composed of 8-bit and 20-bit registers respectively for the input buffer and output buffer, 12-bit memory modules, 20-bit adders, and 20-bit registers for processing stage. The extra XOR gates and 16-bit 2-to-1 multiplexers are needed in the design [35].

As for the adder-based DA design [38], the issue in this design is how to find the common terms from the nonzero sub-expressions in order to reduce the hardware cost of the summation network. Extracting the common terms is similar to the problem of logic optimization. Since this is a NP complete problem [39], it is almost impossible to exactly estimate the hardware cost of the adder-based DA design. Thus the worst-case estimation for the common terms of the adder-based DA design has been adopted here.

Fig. 3.4, Fig. 3.5, and Fig. 3.6 respectively show the comparison of area cost, ACT, and area-delay product of the existing designs and the proposed GDA design in realizing the 1-D DFT, where the 5-point and 7-point 1-D DFTs are realized by using the GDA design, and the 11-point 1-D DFT is realized by using the BGDA design with the partition factors of 2×5 for the 10-point cyclic convolution required in the

1-D 11-point DFT. As shown in Fig. 3.6, the delay-area product of the proposed design is much smaller than the traditional memory-based DA design. Precisely, the proposed GDA design can save averagely 68%, 49%, and 29% of the delay-area product, respectively, as compared with the systolic array designs [10][33], memory-based DA designs [34][35], and adder-based DA design [38] in the case that the length of cyclic convolution is smaller than nine. Generally, the length of cyclic convolution should be smaller than nine for obtaining a reasonable memory size in DA-based designs.

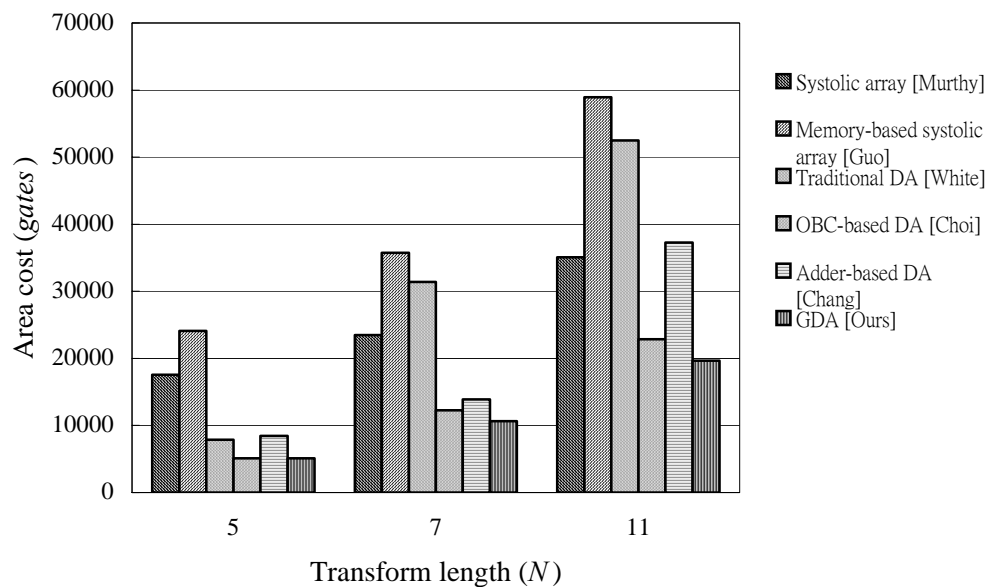


Fig. 3.4: Comparison of the area cost of the existing DFT designs and the proposed GDA design in realizing the 1-D N-point DFT.

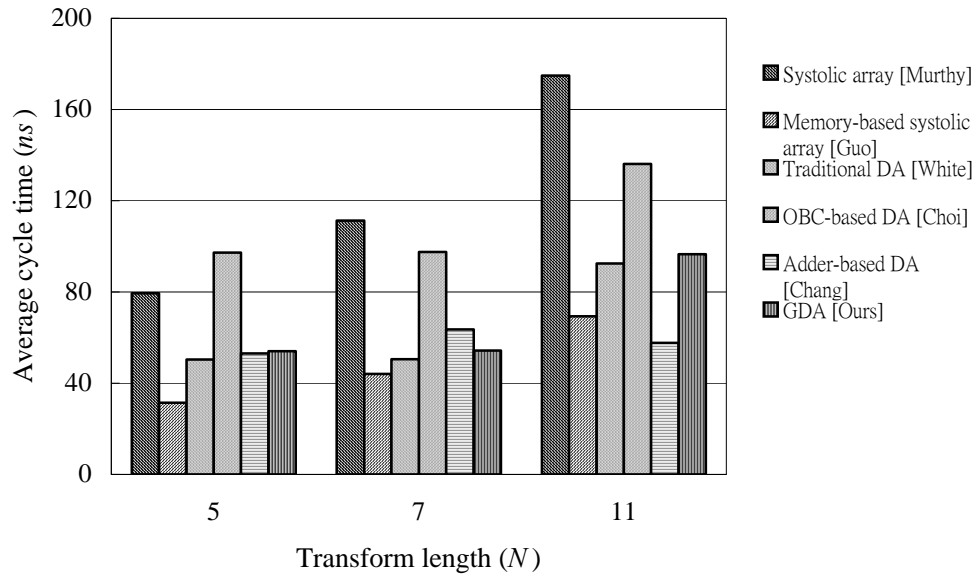


Fig. 3.5: Comparison of the ACT for the existing designs and the proposed GDA design in realizing the 1-D N -point DFT.

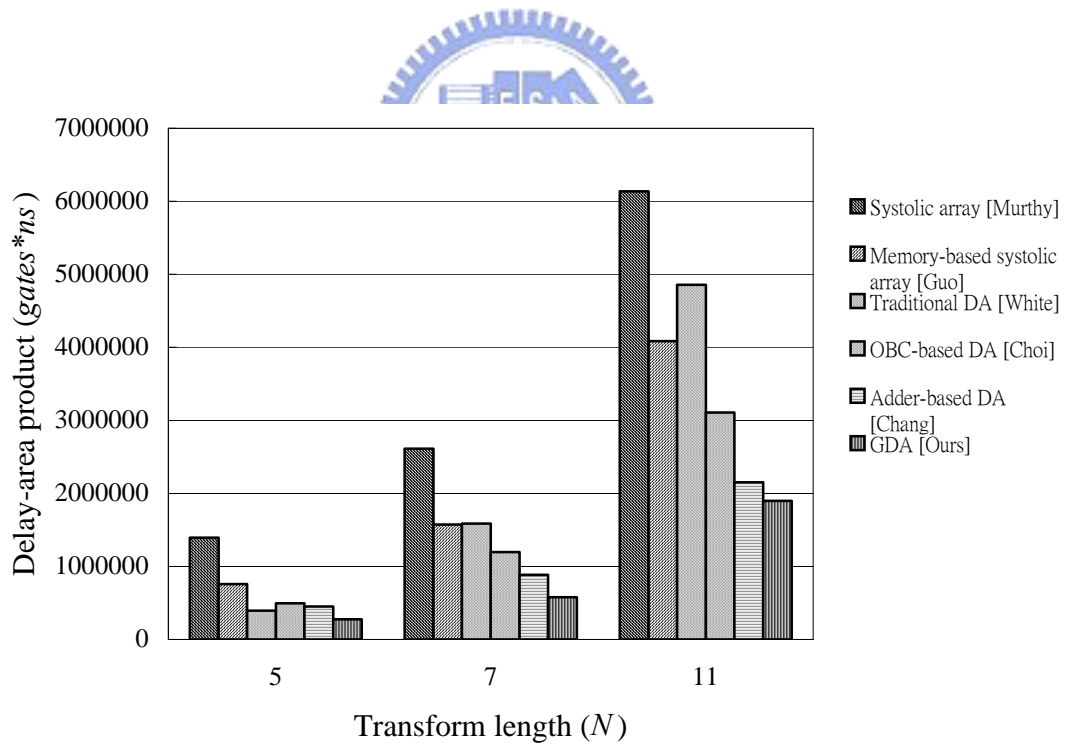


Fig. 3.6: Comparison of the delay-area product for the existing designs and the proposed GDA design in realizing the 1-D N -point DFT.

Table 3.6: Area cost models to estimate the 1-D N-point DFT modules in the existing systolic array designs, DA-based designs, and the proposed GDA design with real input data.

	Module name	Address decoder (coded addresses)	XOR (bit)	Mux/Demux (words)	RAM (words)	memory (double-words)	(N-1)-bit Barrel rotator (double-words)	Adder (double-words)	Mul (double-words)	Reg (words)	P/S (words)
Murthy [10] (Systolic array)	Array processing stage							2(N+1)	2(N+1)	3(N+1)	
Guo [33] (Memory-based systolic array)	Input-buffer, output-buffer, and Preprocessing stage	N		2-to-1: 2 1-to-2: 1	N					2N	
	Array processing stage	$2^*[(N-1)*2^{L/2}*2]$		2 to 1: $2^*[(N-1)+3]$		$2^*[(N-1)*2^{L/2}*2]$		$2^*[2(N-1)+2]$			
	totally	$N+2^*[(N-1)*2^{L/2}*2]$		2N+7	N	$(N-1)*2^{L/2}*4$		4N		2N	
White [34] (Traditional DA)	Input and output buffers							1		2N	
	DA processing stage	$2^{(N-1)}$				$2^*2^{(N-1)* (N-1)/2}$		$2^*(N-1)/2$		$2^*(N-1)/2$	N-1
	Y(0) computation							1		1	
	totally	$2^{(N-1)}$				$2^{(N-1)* (N-1)}$		N+1		3N	N-1
Choi [35] (OBC-based DA)	Input and output buffers							1		2N	
	DA processing stage (coeff-add, sum, acc)	$2^{(N-1)-2}$	$2(2(N-2)+2)$	2 to 1: $2^*2^*(N-1)/2 + 2^*(N-1)/2$		$2^*2^{(N-1)-2} * (N-1)/2$		$2^*(N-1)/2$		$2^*(N-1)/2$	N-1
	Y(0) computation							1		1	
	totally	$2^{(N-1)-2}$	4(N-1)	3(N-1)		$2^{(N-1)-2} * (N-1)$		N+1		3N	N-1
Chang [38] (Adder-based DA)	Input and output buffers							1		2N	
	DA processing stage (DA, sum)							$2^*[(N-1)+L+1]* (N-1)/2$		$2^*(N-1)/2$	
	Y(0) computation							1		1	
	totally							$N^2+N(L-1)-L$		3N	
Proposed GDA design	Input and output buffers							1		2N	
	DA processing stage	G(N-1)				$2^*G(N-1)* (N-1)/2$	4_2	$2^*(N-1)/2$		$2^*(N-1)/2$	N-1
	Y(0) computation							1		1	
	totally	G(N-1)				$G(N-1)* (N-1)$	2	N+1		3N	N-1

Note:

1. L denotes the word length that equals to 16 in the design example, and N denotes the transform length.

Table 3.7: Area cost models to estimate the 1-D N-point DFT modules with the partitioned cyclic convolution in the existing systolic array designs, DA-based designs, and the proposed BGDA design with real input data.

	Module name	Address decoder (coded addresses)	XOR (bit)	Mux/Demux (words)	RAM (words)	memory (double-words)	(N-1)/2-bit Barrel rotator (double-words)	Adder (double-words)	Mul (double-words)	Reg (words)	P/S (words)
Murthy [10] (Systolic array)	Array processing stage							2(N+1)	2(N+1)	3(N+1)	
Guo [33] (Memory-based systolic array)	Input-buffer, output-buffer, and Preprocessing stage	N		2-to-1: 2 1-to-2: 1	N					2N	
	Array processing stage	$2^*[(N-1)*2^{L/2}*2]$		2 to 1: $2^*[(N-1)+3]$		$2^*[(N-1)*2^{L/2}*2]$		$2^*[2(N-1)+2]$			
	totally	$N+2^*[(N-1)*2^{L/2}*2]$		2N+7	N	$(N-1)*2^{L/2}*4$		4N		2N	
White [34] (Traditional DA)	Input and output buffers							1		2N	
	DA processing stage	$2^*2^{(N-1)/2}$				$2^*2^*2^{(N-1)/2} * (N-1)/2$		$2^*(N-1)/2 + 2^*(N-1)/2$		$2^*(N-1)/2$	N-1
	Y(0) computation							1		1	
	totally	$2^{(N+1)/2}$				$2^{(N+1)/2} * (N-1)$		2N		3N	N-1
Choi [35] (OBC-based DA)	Input and output buffers							1		2N	
	DA processing stage (coeff-add, sum, acc)	$2^*2^{(N-1)/2-2}$	$2^*2^*(2^{L/2} * ((N-1)/2-2) + 2)$	2 to 1: $2^*2^*2^*(N-1)/2 + 2^*(N-1)/2$		$2^*2^*2^{(N-1)/2-2} * (N-1)/2$		$2^*(N-1)/2 + 2^*(N-1)/2$		$2(N-1)/2$	N-1
	Y(0) computation							1		1	
	totally	$2^{(N-1)/2-1}$	4(N-3)	5(N-1)		$2^*2^{(N-1)/2-2} * (N-1)$		2(N-1)		3N	N-1
Chang [38] (Adder-based DA)	Input and output buffers							1		2N	
	DA processing stage (DA, sum)							$2^*[2^*[(N-1)/2 + L] + 1] * (N-1)/2$		$2^*(N-1)/2$	
	Y(0) computation							1		1	
	totally							$N^2 + N(2L-1) - 2L$		3N	
Proposed BGDA design	Input and output buffers							1		2N	
	DA processing stage	$2^*G((N-1)/2)$				$2^*2^*G((N-1)/2) * (N-1)/2$	4^*2^*2	$2^*(N-1)/2 + 2^*(N-1)/2$		$2^*(N-1)/2$	N-1
	Y(0) computation							1		1	
	totally	$2^*G((N-1)/2)$				$2G((N-1)/2) * (N-1)$	4	2N		3N	N-1

Table 3.8: Average cycle time (ACT) models to estimate the not partitioned and partitioned 1-D N-point DFT modules in the existing systolic array designs, DA-based designs, and the proposed GDA design with real input data.

	Not partitioned	Partitioned
Murthy [10] (Systolic array)	$N * (T_{mul} + T_{add} + T_{latch})$	$N * (T_{mul} + T_{add} + T_{latch})$
Guo [33] (Memory-based systolic array)	$N * (T_{rom1} + 2T_{add} + T_{latch})$	$N * (T_{rom1} + 2T_{add} + T_{latch})$
White [34] (Traditional DA)	$L * (T_{rom2} + T_{add} + T_{latch})$	$L * (T_{rom2} + 2T_{add} + T_{latch})$
Choi [35] (OBC-based DA)	$L * (T_{xor} + 2T_{mux} + T_{rom3} + 2T_{add} + T_{latch})$	$L * (T_{xor} + 2T_{mux} + T_{rom3} + 3T_{add} + T_{latch})$
Chang [38] (Adder-based DA)	$((N-1)+2\log_2L) * T_{add} + T_{latch}$	$((N-1)/2-1)+2\log_2L+1) * T_{add} + T_{latch}$
Proposed GDA design	$L * (T_{rom5} + T_{bar} + T_{add} + T_{latch})$	$L * (T_{rom5} + T_{bar} + 2T_{add} + T_{latch})$

Note:

1. T_{mul} denotes the delay time of multiplier, T_{mux} denotes the delay time of multiplexer, T_{add} denotes the delay time of adder, T_{rom} denotes the access time of memory, and T_{bar} denotes the delay time of Barrel shifter with N -word width.
2. Since the required memory sizes of the designs except for the adder-based DA are different, the access time of memory in these designs is also different.
3. The timing costs of memory are the sum of delay in both the address decoder and memory-cell.
4. L denotes the word length of the candidate of DA input, and N denotes the transform length of 1-D DFT.

3.2 Design of 1-D DHT

3.2.1 Cyclic Convolution Formulation

Prime-length case

The 1-D N-point DHT of an input sequence $\{x(n), n = 0, 1, \dots, N-1\}$ is defined as

$$Y(k) = \sum_{n=0}^{N-1} x(n) \cdot H_N^{nk}; \quad k = 0, 1, \dots, N-1 \quad (3.21)$$

where $H_N^{nk} = \cos(2nk\alpha) + j\sin(2nk\alpha)$, $\alpha = \pi/N$, and N denotes the transform length. If N is prime, we can rewrite (3.21) in a cyclic convolution by exploiting the property of input/output (I/O) data permutation as

$$Y(0) = \sum_{n=0}^{N-1} x(n),$$

$$Y((g^k)_N) = \left[\sum_{n=1}^{N-1} x((g^{n-k})_N) \cdot \text{cas}\left(\frac{2\pi}{N}(g^n)_N\right) \right] + x(0); \quad k = 1, \dots, N-1$$

$$= T((g^k)_N) + x(0) \quad (3.22)$$

$$T((g^k)_N) = \sum_{n=1}^{N-1} x((g^{n-k})_N) \cdot \text{cas}\left(\frac{2\pi}{N}(g^n)_N\right) \quad (3.23)$$

where $(g^k)_N$ denotes the result of “ g^k modulo N ” for short and g is a primitive element. $T((g^k)_N)$ in (3.22) is the kernel of the N -point DHT that is written in cyclic convolution formulation.

Non-prime length case

According to (3.21) and utilizing the Chirp-Z transform [50][51], we illustrate the derivation of cyclic convolution algorithm for non-prime length DHT in the following. By introducing two sequences $\{C(k)\}$ and $\{S(k)\}$ defined as follows:

$$C(k) = \sum_{n=0}^{N-1} x(n) \cdot \cos[(k^2 - 2nk) \cdot \alpha]$$

$$S(k) = \sum_{n=0}^{N-1} x(n) \cdot \sin[(k^2 - 2nk) \cdot \alpha] \quad (3.24)$$

we can express DHT equation in (3.21) as

$$Y(k) = \frac{1}{2 \cdot \cos(k^2 \alpha)} [C(k) + C(2N-k) + S(2N-k) - S(k)]; k = 0, 1, \dots, N-1 \quad (3.25)$$

Then, suitably evaluating the term $C(k) + C(2N-k) + S(2N-k) - S(k)$ yields

$$Y(k) = \frac{1}{2 \cdot \cos(k^2 \alpha)} [T_c(k) + T_s(k)]; k = 0, 1, \dots, N-1 \quad (3.26)$$

where

$$\begin{aligned}
T_c(k) &= \sum_{n=0}^{N-1} x_f((n+k)_N) \cdot \cos(n^2 \alpha); k = 0, 1, \dots, N-1 \\
T_s(k) &= \sum_{n=0}^{N-1} y_f((n+k)_N) \cdot \sin(n^2 \alpha); k = 0, 1, \dots, N-1
\end{aligned} \tag{3.27}$$

and

$$\begin{aligned}
x_f((n+k)_N) &= x'((n+k)_N) + (-1)^N \cdot x'((n+k+N)_N) \\
y_f((n+k)_N) &= y'((n+k)_N) + (-1)^N \cdot y'((n+k+N)_N)
\end{aligned} \tag{3.28}$$

$$x'(n) = \begin{cases} x_p(n), & n = 0, 1, \dots, N-1 \\ (-1)^N \cdot x_n(0), & n = N \\ x_n(2N-n) & n = N+1, \dots, 2N-1 \end{cases} \tag{3.29}$$

$$y'(n) = \begin{cases} -x_n(n), & n = 0, 1, \dots, N-1 \\ (-1)^N \cdot x_p(0), & n = N \\ x_p(2N-n) & n = N+1, \dots, 2N-1 \end{cases} \tag{3.30}$$

$$\begin{aligned}
x_n(n) &= x(n) \cdot [\cos(n^2 \cdot \alpha) - \sin(n^2 \cdot \alpha)] \\
x_p(n) &= x(n) \cdot [\cos(n^2 \cdot \alpha) + \sin(n^2 \cdot \alpha)]
\end{aligned} \tag{3.31}$$

In the above equations, $(A)_N$ denotes the result of A modulo N operation for short. It is seen that both DHT kernel operations, i.e. $T_c(k)$ and $T_s(k)$ in (3.27), are expressed in cyclic convolution forms and thus can be efficiently implemented by GDA. However, the non-prime length DHT algorithm requires pre-processing as indicated in (3.28) ~ (3.31) and post-processing as indicated in (3.26). This algorithm is useful in realizing the DHT with any length, which can cover the applications with broader ranges in the transform length than the fast algorithms being developed for 2^p -point DHT and other prime-length DHT algorithms.

3.2.2 Numerical stability

For the above mentioned algorithms with non-prime length, the issue of division operation involves in them to evaluate the transform values. This will cause numerical instability of some results as the denominator in division operation may equal to zero for specific values of k . In the following, we illustrate the remedy for this issue to ensure the correctness of non-prime length 1-D DHT algorithm.

Since $\cos(\frac{k^2\pi}{N}) = 0$ implies $\sin(\frac{k^2\pi}{N}) = 1$ or -1 , we can overcome this issue by first reformulating (3.25) as

$$Y(k) = \frac{1}{2 \cdot \sin(k^2\alpha)} [C(k) - C(2N-k) + S(2N-k) + S(k)]; k = 0, 1, \dots, N-1 \quad (3.32)$$

Then, evaluating the term $C(k) - C(2N-k) + S(2N-k) + S(k)$ based on the same procedure shown in before yields

$$Y(k) = \frac{1}{2 \cdot \sin(k^2\alpha)} [U_c(k) + U_s(k)]; k = 0, 1, \dots, N-1 \quad (3.33)$$

where

$$\begin{aligned} U_c(k) &= \sum_{n=0}^{N-1} -y_f((n+k)_N) \cdot \cos(n^2\alpha); k = 0, 1, \dots, N-1 \\ U_s(k) &= \sum_{n=0}^{N-1} x_f((n+k)_N) \cdot \sin(n^2\alpha); k = 0, 1, \dots, N-1 \end{aligned} \quad (3.34)$$

Compared with the procedure mentioned before, the sequences $\{U_c(k)\}$ and $\{U_s(k)\}$ are similar to $\{T_c(k)\}$ and $\{T_s(k)\}$, and the operands $y_f((N+k)_N)$ and $x_f((N+k)_N)$ are exchanged with different signs. This phenomenon reveals that this issue of numerical instability can be solved by using simple control in hardware realization.

3.2.3 Symmetry exploration of the DHT in cyclic convolution

Let us take an example of 1-D 11-point DHT with the real input sequence $\{x(n), n=0, 1, \dots, 10\}$. The cyclic convolution form of $T((g^k)_N)$ can be expressed as

$$\begin{bmatrix} T(2) \\ T(4) \\ T(8) \\ T(5) \\ T(10) \\ T(9) \\ T(7) \\ T(3) \\ T(6) \\ T(1) \end{bmatrix} = \begin{bmatrix} x(1) & x(2) & x(4) & x(8) & x(5) & x(10) & x(9) & x(7) & x(3) & x(6) \\ x(6) & x(1) & x(2) & x(4) & x(8) & x(5) & x(10) & x(9) & x(7) & x(3) \\ x(3) & x(6) & x(1) & x(2) & x(4) & x(8) & x(5) & x(10) & x(9) & x(7) \\ x(7) & x(3) & x(6) & x(1) & x(2) & x(4) & x(8) & x(5) & x(10) & x(9) \\ x(9) & x(7) & x(3) & x(6) & x(1) & x(2) & x(4) & x(8) & x(5) & x(10) \\ x(10) & x(9) & x(7) & x(3) & x(6) & x(1) & x(2) & x(4) & x(8) & x(5) \\ x(5) & x(10) & x(9) & x(7) & x(3) & x(6) & x(1) & x(2) & x(4) & x(8) \\ x(8) & x(5) & x(10) & x(9) & x(7) & x(3) & x(6) & x(1) & x(2) & x(4) \\ x(4) & x(8) & x(5) & x(10) & x(9) & x(7) & x(3) & x(6) & x(1) & x(2) \\ x(2) & x(4) & x(8) & x(5) & x(10) & x(9) & x(7) & x(3) & x(6) & x(1) \end{bmatrix} \times \begin{bmatrix} H_{11}^2 \\ H_{11}^4 \\ H_{11}^8 \\ H_{11}^5 \\ H_{11}^{10} \\ H_{11}^9 \\ H_{11}^7 \\ H_{11}^3 \\ H_{11}^6 \\ H_{11}^1 \end{bmatrix}. \quad (3.35)$$

As shown in (3.36), the coefficient matrix in (3.35) can be expanded as the even symmetries of cosine function $c_N^i = c_N^{N-i}, i=1,2,\dots,N-1$, where $c_N^i = \cos(2i\pi/N)$, and the odd symmetries of sine function $s_N^i = -s_N^{N-i}, i=1,2,\dots,N-1$, where $s_N^i = \sin(2i\pi/N)$.

$$\begin{bmatrix} H_{11}^2 \\ H_{11}^7 \\ H_{11}^8 \\ H_{11}^6 \\ H_{11}^{10} \\ H_{11}^9 \\ H_{11}^4 \\ H_{11}^3 \\ H_{11}^5 \\ H_{11}^1 \end{bmatrix} = \begin{bmatrix} c_{11}^2 + s_{11}^2 \\ c_{11}^7 + s_{11}^7 \\ c_{11}^8 + s_{11}^8 \\ c_{11}^6 + s_{11}^6 \\ c_{11}^{10} + s_{11}^{10} \\ c_{11}^9 + s_{11}^9 \\ c_{11}^4 + s_{11}^4 \\ c_{11}^3 + s_{11}^3 \\ c_{11}^5 + s_{11}^5 \\ c_{11}^1 + s_{11}^1 \end{bmatrix} = \begin{bmatrix} c_{11}^2 + s_{11}^2 \\ c_{11}^4 - s_{11}^4 \\ c_{11}^8 + s_{11}^8 \\ c_{11}^6 + s_{11}^6 \\ c_{11}^{10} + s_{11}^{10} \\ c_{11}^2 - s_{11}^2 \\ c_{11}^4 + s_{11}^4 \\ c_{11}^8 - s_{11}^8 \\ c_{11}^6 - s_{11}^6 \\ c_{11}^{10} - s_{11}^{10} \end{bmatrix} \quad (3.36)$$

Then, we can re-write $T((2^k)_{11})$ in (3.35) as follows:



$$\begin{aligned}
& \begin{bmatrix} T(2) \\ T(7) \\ T(8) \\ T(6) \\ T(10) \\ T(9) \\ T(4) \\ T(3) \\ T(5) \\ T(1) \end{bmatrix} = \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) & x(10) & x(2) & x(7) & x(8) & x(6) \\ x(5) & x(1) & x(9) & x(4) & x(3) & x(6) & x(10) & x(2) & x(7) & x(8) \\ x(3) & x(5) & x(1) & x(9) & x(4) & x(8) & x(6) & x(10) & x(2) & x(7) \\ x(4) & x(3) & x(5) & x(1) & x(9) & x(7) & x(8) & x(6) & x(10) & x(2) \\ x(9) & x(4) & x(3) & x(5) & x(1) & x(2) & x(7) & x(8) & x(6) & x(10) \\ x(10) & x(2) & x(7) & x(8) & x(6) & x(1) & x(9) & x(4) & x(3) & x(5) \\ x(6) & x(10) & x(2) & x(7) & x(8) & x(5) & x(1) & x(9) & x(4) & x(3) \\ x(8) & x(6) & x(10) & x(2) & x(7) & x(3) & x(5) & x(1) & x(9) & x(4) \\ x(7) & x(8) & x(6) & x(10) & x(2) & x(4) & x(3) & x(5) & x(1) & x(9) \\ x(2) & x(7) & x(8) & x(6) & x(10) & x(9) & x(4) & x(3) & x(5) & x(1) \end{bmatrix} \times \begin{bmatrix} c_{11}^2 + s_{11}^2 \\ c_{11}^4 - s_{11}^4 \\ c_{11}^8 + s_{11}^8 \\ c_{11}^6 + s_{11}^6 \\ c_{11}^{10} + s_{11}^{10} \\ c_{11}^2 - s_{11}^2 \\ c_{11}^4 + s_{11}^4 \\ c_{11}^8 - s_{11}^8 \\ c_{11}^6 - s_{11}^6 \\ c_{11}^{10} - s_{11}^{10} \end{bmatrix} \\
& = \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) & x(10) & x(2) & x(7) & x(8) & x(6) \\ x(5) & x(1) & x(9) & x(4) & x(3) & x(6) & x(10) & x(2) & x(7) & x(8) \\ x(3) & x(5) & x(1) & x(9) & x(4) & x(8) & x(6) & x(10) & x(2) & x(7) \\ x(4) & x(3) & x(5) & x(1) & x(9) & x(7) & x(8) & x(6) & x(10) & x(2) \\ x(9) & x(4) & x(3) & x(5) & x(1) & x(2) & x(7) & x(8) & x(6) & x(10) \\ x(1) & x(9) & x(4) & x(3) & x(5) & x(10) & x(2) & x(7) & x(8) & x(6) \\ x(5) & x(1) & x(9) & x(4) & x(3) & x(6) & x(10) & x(2) & x(7) & x(8) \\ x(3) & x(5) & x(1) & x(9) & x(4) & x(8) & x(6) & x(10) & x(2) & x(7) \\ x(4) & x(3) & x(5) & x(1) & x(9) & x(7) & x(8) & x(6) & x(10) & x(2) \\ x(9) & x(4) & x(3) & x(5) & x(1) & x(2) & x(7) & x(8) & x(6) & x(10) \end{bmatrix} \begin{bmatrix} c_{11}^2 \\ c_{11}^4 \\ c_{11}^8 \\ c_{11}^6 \\ c_{11}^{10} \\ c_{11}^2 \\ c_{11}^4 \\ c_{11}^8 \\ c_{11}^6 \\ c_{11}^{10} \end{bmatrix} \\
& + \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) & x(10) & x(2) & x(7) & x(8) & x(6) \\ x(5) & x(1) & x(9) & x(4) & x(3) & x(6) & x(10) & x(2) & x(7) & x(8) \\ x(3) & x(5) & x(1) & x(9) & x(4) & x(8) & x(6) & x(10) & x(2) & x(7) \\ x(4) & x(3) & x(5) & x(1) & x(9) & x(7) & x(8) & x(6) & x(10) & x(2) \\ x(9) & x(4) & x(3) & x(5) & x(1) & x(2) & x(7) & x(8) & x(6) & x(10) \\ -x(1) & -x(9) & -x(4) & -x(3) & -x(5) & -x(10) & -x(2) & -x(7) & -x(8) & -x(6) \\ -x(5) & -x(1) & -x(9) & -x(4) & -x(3) & -x(6) & -x(10) & -x(2) & -x(7) & -x(8) \\ -x(3) & -x(5) & -x(1) & -x(9) & -x(4) & -x(8) & -x(6) & -x(10) & -x(2) & -x(7) \\ -x(4) & -x(3) & -x(5) & -x(1) & -x(9) & -x(7) & -x(8) & -x(6) & -x(10) & -x(2) \\ -x(9) & -x(4) & -x(3) & -x(5) & -x(1) & -x(2) & -x(7) & -x(8) & -x(6) & -x(10) \end{bmatrix} \begin{bmatrix} s_{11}^2 \\ -s_{11}^4 \\ s_{11}^8 \\ s_{11}^6 \\ s_{11}^{10} \\ s_{11}^2 \\ -s_{11}^4 \\ s_{11}^8 \\ s_{11}^6 \\ s_{11}^{10} \end{bmatrix} \\
& \tag{3.37}
\end{aligned}$$

From (3.37), we see that

$$\begin{bmatrix} T_C(2) \\ T_C(7) \\ T_C(8) \\ T_C(6) \\ T_C(10) \end{bmatrix} = \begin{bmatrix} T_C(9) \\ T_C(4) \\ T_C(3) \\ T_C(5) \\ T_C(1) \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} T_S(2) \\ T_S(7) \\ T_S(8) \\ T_S(6) \\ T_S(10) \end{bmatrix} = - \begin{bmatrix} T_S(9) \\ T_S(4) \\ T_S(3) \\ T_S(5) \\ T_S(1) \end{bmatrix}. \tag{3.38}$$

Then, we can respectively express $T_R(\cdot)$ and $T_I(\cdot)$ in (3.38) as

$$\begin{aligned}
\begin{bmatrix} T_C(2) \\ T_C(7) \\ T_C(8) \\ T_C(6) \\ T_C(10) \end{bmatrix} &= \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) & x(10) & x(2) & x(7) & x(8) & x(6) \\ x(5) & x(1) & x(9) & x(4) & x(3) & x(6) & x(10) & x(2) & x(7) & x(8) \\ x(3) & x(5) & x(1) & x(9) & x(4) & x(8) & x(6) & x(10) & x(2) & x(7) \\ x(4) & x(3) & x(5) & x(1) & x(9) & x(7) & x(8) & x(6) & x(10) & x(2) \\ x(9) & x(4) & x(3) & x(5) & x(1) & x(2) & x(7) & x(8) & x(6) & x(10) \end{bmatrix} \cdot \begin{bmatrix} c_{11}^2 \\ c_{11}^4 \\ c_{11}^8 \\ c_{11}^6 \\ c_{11}^{10} \\ c_{11}^2 \\ c_{11}^4 \\ c_{11}^8 \\ c_{11}^6 \\ c_{11}^{10} \end{bmatrix} \\
&= \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) \\ x(5) & x(1) & x(9) & x(4) & x(3) \\ x(3) & x(5) & x(1) & x(9) & x(4) \\ x(4) & x(3) & x(5) & x(1) & x(9) \\ x(9) & x(4) & x(3) & x(5) & x(1) \end{bmatrix} \cdot \begin{bmatrix} c_{11}^2 \\ c_{11}^4 \\ c_{11}^8 \\ c_{11}^6 \\ c_{11}^{10} \end{bmatrix} + \begin{bmatrix} x(10) & x(2) & x(7) & x(8) & x(6) \\ x(6) & x(10) & x(2) & x(7) & x(8) \\ x(8) & x(6) & x(10) & x(2) & x(7) \\ x(7) & x(8) & x(6) & x(10) & x(2) \\ x(2) & x(7) & x(8) & x(6) & x(10) \end{bmatrix} \cdot \begin{bmatrix} c_{11}^2 \\ c_{11}^4 \\ c_{11}^8 \\ c_{11}^6 \\ c_{11}^{10} \end{bmatrix} \\
&= \begin{bmatrix} T_{C_1}(2) \\ T_{C_1}(7) \\ T_{C_1}(8) \\ T_{C_1}(6) \\ T_{C_1}(10) \end{bmatrix} + \begin{bmatrix} T_{C_2}(2) \\ T_{C_2}(7) \\ T_{C_2}(8) \\ T_{C_2}(6) \\ T_{C_2}(10) \end{bmatrix}
\end{aligned}$$



(3.39)

$$\begin{aligned}
\begin{bmatrix} T_S(2) \\ T_S(7) \\ T_S(8) \\ T_S(6) \\ T_S(10) \end{bmatrix} &= \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) & x(10) & x(2) & x(7) & x(8) & x(6) \\ x(5) & x(1) & x(9) & x(4) & x(3) & x(6) & x(10) & x(2) & x(7) & x(8) \\ x(3) & x(5) & x(1) & x(9) & x(4) & x(8) & x(6) & x(10) & x(2) & x(7) \\ x(4) & x(3) & x(5) & x(1) & x(9) & x(7) & x(8) & x(6) & x(10) & x(2) \\ x(9) & x(4) & x(3) & x(5) & x(1) & x(2) & x(7) & x(8) & x(6) & x(10) \end{bmatrix} \begin{bmatrix} s_{11}^2 \\ -s_{11}^4 \\ s_{11}^8 \\ s_{11}^6 \\ s_{11}^{10} \\ -s_{11}^2 \\ s_{11}^4 \\ -s_{11}^8 \\ -s_{11}^6 \\ -s_{11}^{10} \end{bmatrix} \\
&= \begin{bmatrix} x(1) & x(9) & x(4) & x(3) & x(5) \\ x(5) & x(1) & x(9) & x(4) & x(3) \\ x(3) & x(5) & x(1) & x(9) & x(4) \\ x(4) & x(3) & x(5) & x(1) & x(9) \\ x(9) & x(4) & x(3) & x(5) & x(1) \end{bmatrix} \begin{bmatrix} s_{11}^2 \\ -s_{11}^4 \\ s_{11}^8 \\ s_{11}^6 \\ s_{11}^{10} \end{bmatrix} - \begin{bmatrix} x(10) & x(2) & x(7) & x(8) & x(6) \\ x(6) & x(10) & x(2) & x(7) & x(8) \\ x(8) & x(6) & x(10) & x(2) & x(7) \\ x(7) & x(8) & x(6) & x(10) & x(2) \\ x(2) & x(7) & x(8) & x(6) & x(10) \end{bmatrix} \begin{bmatrix} s_{11}^2 \\ -s_{11}^4 \\ s_{11}^8 \\ s_{11}^6 \\ s_{11}^{10} \end{bmatrix} \\
&= \begin{bmatrix} T_{S1}(2) \\ T_{S1}(7) \\ T_{S1}(8) \\ T_{S1}(6) \\ T_{S1}(10) \end{bmatrix} - \begin{bmatrix} T_{S2}(2) \\ T_{S2}(7) \\ T_{S2}(8) \\ T_{S2}(6) \\ T_{S2}(10) \end{bmatrix}
\end{aligned} \tag{3.40}$$



Observing (3.39) and (3.40), similar to the DFT, we find that the cosine part of $T((2^k)_{II})$ is composed of the same upper and lower halves, and the sine part of $T((2^k)_{II})$ is composed of the upper and the lower halves with the same absolute value, but different signs. We can also calculate two output values simultaneously through (3.37) with the same hardware. This feature facilitates the hardware sharing in computing $T((g^k)_N)$ with even and odd indices such that only half the hardware is needed as compared with the direct realization on (3.35).

3.2.4 Architecture design and evaluation

Architecture design

By exploiting the symmetrical properties of both the cosine and sine functions shown in (3.37) in the DHT computation, the outputs with odd indices can also be obtained by means of hardwiring to achieve the reduction of memory cost by a factor

of two. With the example of 1-D 11-point DHT and referring to the reformulation of 1-D DHT in (3.39) and (3.40), we can realize the 10-point cyclic convolution in 11-point DHT through the architectures designed for the 5-point cyclic convolution as shown in Fig. 3.7. The architecture is composed of GDAUs, address decoder, adders/subtractors, accumulators, and parallel-to-serial converters. The $GDAU_c$ and $GDAU_s$ are used to respectively realize the operations specified in (3.39) and (3.40) for computing 5-point cyclic convolution. According to the rule of group mapping shown in Table 2.1, the contents of the memory corresponding to $GDAU_c$ and $GDAU_s$ are shown in Table 3.9 and Table 3.10 respectively, which illustrate the distribution of partial products in the memory of GDA design.

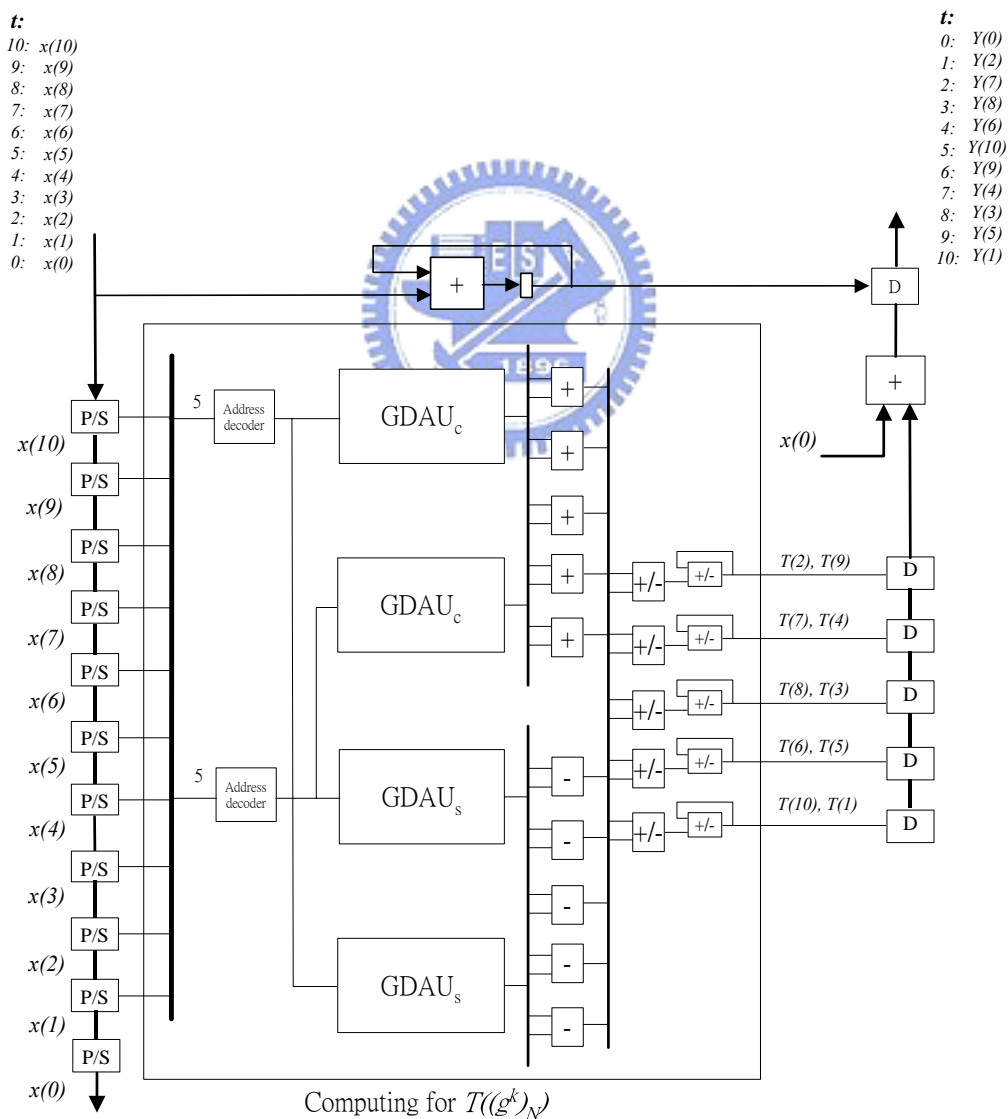


Fig. 3.7: The architecture of the GDA design realizing the 1-D 11-point DHT.

Table 3.9: The 8 groups of memory content used for computing the 5-point cyclic convolution in GDAUc.

Grouped candidates of DA input (X_q)	Group address (G_q)	$T_{C1}(2)/T_{C2}(2)/T_{C1}(9)/T_{C2}(9)$	$T_{C1}(7)/T_{C2}(7)/T_{C1}(4)/T_{C2}(4)$	$T_{C1}(8)/T_{C2}(8)/T_{C1}(3)/T_{C2}(3)$	$T_{C1}(6)/T_{C2}(6)/T_{C1}(5)/T_{C2}(5)$	$T_{C1}(10)/T_{C2}(10)/T_{C1}(1)/T_{C2}(1)$
0	0	0	0	0	0	0
1, 2, 4, 8, 16	1	c_{11}^{10}	c_{11}^2	c_{11}^4	c_{11}^8	c_{11}^6
3, 6, 12, 24, 17	2	$c_{11}^6+c_{11}^{10}$	$c_{11}^{10}+c_{11}^2$	$c_{11}^2+c_{11}^4$	$c_{11}^4+c_{11}^8$	$c_{11}^8+c_{11}^6$
5, 10, 20, 9, 18	3	$c_{11}^8+c_{11}^{10}$	$c_{11}^6+c_{11}^2$	$c_{11}^{10}+c_{11}^4$	$c_{11}^2+c_{11}^8$	$c_{11}^4+c_{11}^6$
7, 14, 28, 25, 19	4	$c_{11}^8+c_{11}^6+c_{11}^{10}$	$c_{11}^6+c_{11}^{10}+c_{11}^2$	$c_{11}^{10}+c_{11}^2+c_{11}^4$	$c_{11}^2+c_{11}^4+c_{11}^8$	$c_{11}^4+c_{11}^8+c_{11}^6$
11, 22, 13, 26, 21	5	$c_{11}^4+c_{11}^6+c_{11}^{10}$	$c_{11}^2+c_{11}^8+c_{11}^{10}$	$c_{11}^2+c_{11}^4+c_{11}^6$	$c_{11}^4+c_{11}^8+c_{11}^{10}$	$c_{11}^2+c_{11}^8+c_{11}^6$
15, 30, 29, 27, 23	6	$c_{11}^4+c_{11}^8+c_{11}^6+c_{11}^{10}$	$c_{11}^8+c_{11}^6+c_{11}^2+c_{11}^{10}$	$c_{11}^6+c_{11}^{10}+c_{11}^2+c_{11}^4$	$c_{11}^{10}+c_{11}^2+c_{11}^4+c_{11}^8$	$c_{11}^2+c_{11}^4+c_{11}^8+c_{11}^6$
31	7	$c_{11}^2+c_{11}^4+c_{11}^8+c_{11}^6+c_{11}^{10}$	$c_{11}^2+c_{11}^4+c_{11}^8+c_{11}^6+c_{11}^{10}$	$c_{11}^2+c_{11}^4+c_{11}^8+c_{11}^6+c_{11}^{10}$	$c_{11}^2+c_{11}^4+c_{11}^8+c_{11}^6+c_{11}^{10}$	$c_{11}^2+c_{11}^4+c_{11}^8+c_{11}^6+c_{11}^{10}$

Table 3.10: The 8 groups of memory content used for computing the 5-point cyclic convolution in GDAUs.

Grouped candidates of DA input (X_q)	Group address (G_q)	$T_{S1}(2)/T_{S2}(2)/T_{S1}(9)/T_{S2}(9)$	$T_{S1}(7)/T_{S2}(7)/T_{S1}(4)/T_{S2}(4)$	$T_{S1}(8)/T_{S2}(8)/T_{S1}(3)/T_{S2}(3)$	$T_{S1}(6)/T_{S2}(6)/T_{S1}(5)/T_{S2}(5)$	$T_{S1}(10)/T_{S2}(10)/T_{S1}(1)/T_{S2}(1)$
0	0	0	0	0	0	0
1, 2, 4, 8, 16	1	s_{11}^{10}	$-s_{11}^2$	$-s_{11}^4$	s_{11}^8	s_{11}^6
3, 6, 12, 24, 17	2	$s_{11}^6+s_{11}^{10}$	$s_{11}^{10}+s_{11}^2$	$s_{11}^2-s_{11}^4$	$-s_{11}^4+s_{11}^8$	$s_{11}^8+s_{11}^6$
5, 10, 20, 9, 18	3	$s_{11}^8+s_{11}^{10}$	$s_{11}^6+s_{11}^2$	$s_{11}^{10}-s_{11}^4$	$s_{11}^2+s_{11}^8$	$-s_{11}^4+s_{11}^6$
7, 14, 28, 25, 19	4	$s_{11}^8+s_{11}^6+s_{11}^{10}$	$s_{11}^6+s_{11}^{10}+s_{11}^2$	$s_{11}^{10}+s_{11}^2-s_{11}^4$	$s_{11}^2-s_{11}^4+s_{11}^8$	$-s_{11}^4+s_{11}^8+s_{11}^6$
11, 22, 13, 26, 21	5	$-s_{11}^4+s_{11}^6+s_{11}^{10}$	$s_{11}^2+s_{11}^8+s_{11}^{10}$	$s_{11}^2-s_{11}^4+s_{11}^6$	$-s_{11}^4+s_{11}^8+s_{11}^{10}$	$s_{11}^2+s_{11}^8+s_{11}^6$
15, 30, 29, 27, 23	6	$-s_{11}^4+s_{11}^8+s_{11}^6+s_{11}^{10}$	$s_{11}^8+s_{11}^6+s_{11}^2+s_{11}^{10}$	$s_{11}^6+s_{11}^{10}+s_{11}^2-s_{11}^4$	$s_{11}^{10}+s_{11}^2+s_{11}^8$	$s_{11}^2-s_{11}^4+s_{11}^6$
31	7	$s_{11}^2-s_{11}^4+s_{11}^8+s_{11}^6+s_{11}^{10}$	$s_{11}^2-s_{11}^4+s_{11}^8+s_{11}^6+s_{11}^{10}$	$s_{11}^2-s_{11}^4+s_{11}^8+s_{11}^6+s_{11}^{10}$	$s_{11}^2-s_{11}^4+s_{11}^8+s_{11}^6+s_{11}^{10}$	$s_{11}^2-s_{11}^4+s_{11}^8+s_{11}^6+s_{11}^{10}$

Design evaluation

The evaluation of GDA-based DHT design and some existing DHT designs involves in the subsequent section of long length DSST's designs.

3.3 Design of 1-D DCT

3.3.1 Cyclic Convolution Formulation

Prime-length case

If transform length N is prime, we can write the 1-D N -point DCT of an input sequence $\{y(n), n = 0, 1, \dots, N-1\}$ in cyclic convolution form by exploiting the property of I/O data permutation as

$$Y(0) = \sum_{n=0}^{N-1} y(n)$$

$$Y((g^k)_N) = [2 \cdot T((g^k)_N) + x(0)] \cdot \cos\left(\frac{\pi}{2N} \cdot ((g^k)_N)\right); k = 1, \dots, N-1 \quad (3.41)$$

$$T((g^k)_N) = \sum_{n=1}^{N-1} x((g^{n-k+1})_N) \cdot (-1)^m \cdot \cos\left(\frac{\pi}{N} \cdot (g^{n+1})_N\right)$$

, where $(g^k)_N$ denotes the result of “ g^k modulo N ” for short, g is a primitive element, and the sequence $\{x(n)\}$ is defined as

$$\left\{ \begin{array}{l} x(N-1) = y(N-1) \\ x(n) = y(n) - x(n+1); n = 0, \dots, N-2 \end{array} \right\}. \quad (3.42)$$

By using the symmetry property of cosine kernel as

$$\cos\left(\frac{\pi}{N} \cdot (g^k)_N\right) = \cos\left(\frac{\pi}{N} \cdot (N - (g^k)_N)\right) = -\cos\left(\frac{\pi}{N} \cdot (g^{n+\frac{N-1}{2}})_N\right), \quad (3.43)$$

we can re-write the $T((g^k)_N)$ in (3.41) as

$$T((g^k)_N) = \sum_{n=1}^{(N-1)/2} [x((g^{n-k+1})_N) \cdot (-1)^m + x((g^{(n-k+1+\frac{N-1}{2}})_N) \cdot (-1)^{m+\frac{N-1}{2}})] \times \cos\left(\frac{\pi}{N} \cdot (g^{n+1})_N\right); k = 1, \dots, N-1 \quad (3.44)$$

To describe the proposed algorithm in more detail, we can write the kernel $T((3^k)_7)$ in a design example of 1-D 7-point DCT in matrix form as

$$\begin{bmatrix} T(3) \\ T(2) \\ T(6) \\ T(4) \\ T(5) \\ T(1) \end{bmatrix} = \begin{bmatrix} -x(3) & x(2) & x(6) & -x(4) & x(5) & x(1) \\ x(1) & x(3) & x(2) & -x(6) & -x(4) & -x(5) \\ x(5) & x(1) & x(3) & -x(2) & -x(6) & -x(4) \\ x(4) & x(5) & x(1) & -x(3) & -x(2) & -x(6) \\ x(6) & x(4) & -x(5) & x(1) & x(3) & -x(2) \\ x(2) & x(6) & x(4) & x(5) & x(1) & x(3) \end{bmatrix} \times \begin{bmatrix} \cos(2a) \\ \cos(6a) \\ \cos(4a) \\ \cos(5a) \\ \cos(1a) \\ \cos(3a) \end{bmatrix}, \quad (3.45)$$

where a denotes $\frac{\pi}{7}$. However, the input data elements of the kernel possess different signs so that it is not easy to apply the proposed memory efficient approach directly to DCT realization. According to the symmetry property of DCT coefficients as that we show in (3.43), we can write (3.45) as

$$\begin{bmatrix} T(3) \\ T(2) \\ T(6) \\ T(4) \\ T(5) \\ T(1) \end{bmatrix} = \begin{bmatrix} -x(3) & x(2) & x(6) & -x(4) & x(5) & x(1) \\ x(1) & x(3) & x(2) & -x(6) & -x(4) & -x(5) \\ x(5) & x(1) & x(3) & -x(2) & -x(6) & -x(4) \\ x(4) & x(5) & x(1) & -x(3) & -x(2) & -x(6) \\ x(6) & x(4) & -x(5) & x(1) & x(3) & -x(2) \\ x(2) & x(6) & x(4) & x(5) & x(1) & x(3) \end{bmatrix} \times \begin{bmatrix} \cos(2a) \\ \cos(6a) \\ \cos(4a) \\ -\cos(2a) \\ -\cos(6a) \\ -\cos(4a) \end{bmatrix}, \quad (3.46)$$

and the data elements in the matrix of (3.46) can be merged as

$$\begin{bmatrix} T(3) \\ T(2) \\ T(6) \\ T(4) \\ T(5) \\ T(1) \end{bmatrix} = \begin{bmatrix} x(4) - x(3) & x(2) - x(5) & x(6) - x(1) \\ x(6) + x(1) & x(4) + x(3) & x(2) + x(5) \\ x(2) + x(5) & x(6) + x(1) & x(4) + x(3) \\ x(4) + x(3) & x(2) + x(5) & x(6) + x(1) \\ x(6) - x(1) & x(4) - x(3) & x(2) - x(5) \\ x(2) - x(5) & x(6) - x(1) & x(4) - x(3) \end{bmatrix} \times \begin{bmatrix} \cos(2a) \\ \cos(6a) \\ \cos(4a) \end{bmatrix}. \quad (3.47)$$

To separate the even and odd outputs in (3.47), we can obtain two smaller perfect cyclic convolution forms as following:

$$\begin{bmatrix} T(2) \\ T(6) \\ T(4) \end{bmatrix} = \begin{bmatrix} x(6) + x(1) & x(4) + x(3) & x(2) + x(5) \\ x(2) + x(5) & x(6) + x(1) & x(4) + x(3) \\ x(4) + x(3) & x(2) + x(5) & x(6) + x(1) \end{bmatrix} \times \begin{bmatrix} \cos(2a) \\ \cos(6a) \\ \cos(4a) \end{bmatrix} \quad (3.48)$$

and

$$\begin{bmatrix} T(5) \\ T(1) \\ T(3) \end{bmatrix} = \begin{bmatrix} x(6) - x(1) & x(4) - x(3) & x(2) - x(5) \\ x(2) - x(5) & x(6) - x(1) & x(4) - x(3) \\ x(4) - x(3) & x(2) - x(5) & x(6) - x(1) \end{bmatrix} \times \begin{bmatrix} \cos(2a) \\ \cos(6a) \\ \cos(4a) \end{bmatrix}. \quad (3.49)$$

From (3.46), (3.47), (3.48), and (3.49), we find that exploiting the symmetry property of the DCT coefficient can help merging the input data elements in the DCT kernel and separating the kernel into two perfect cyclic forms, which facilitates the efficient realization of the DCT through the proposed design approach. Fig. 3.8 shows the area reduction of the memory cost when applying the symmetry property of the DCT coefficients (shown in (3.48) and (3.49)) or not (shown in (3.45)). We find that it is helpful in reducing the memory size greatly when using the symmetry property of the DCT coefficients.

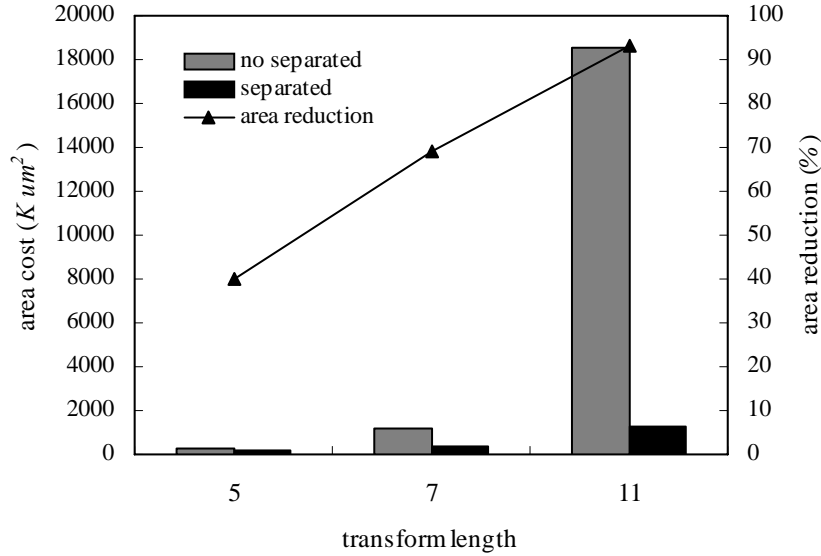


Fig. 3.8: The area reduction of the memory cost when applying the symmetry property of DCT coefficients or not.

For facilitating the proposed memory efficient design approach, we further formulate the $T((g^k)_N)$ specified in (3.44) as

$$T((g^{k+1})_N) = \left. \begin{aligned} & \sum_{j=1}^L \left(\sum_{n=1}^{(N-1)/2} G_1(x_j((g^{n-k})_N)) \cdot \cos\left(\frac{\pi}{N} \cdot (g^{n+1})_N\right) \cdot 2^{-j}; k = 1, \dots, \frac{N-1}{2} \right. \\ & \left. \sum_{j=1}^L \left(\sum_{n=1}^{(N-1)/2} G_2(x_j((g^{n-k})_N)) \cdot \cos\left(\frac{\pi}{N} \cdot (g^{n+1})_N\right) \cdot 2^{-j}; k = \frac{N-1}{2} + 1, \dots, N-1 \right) \right\} \end{aligned} \right\} \quad (3.50)$$

where L denotes the data word length of the variable x , N denotes the transform length, the variable $G(x_j((g^{n-k})_N))$ denotes the j^{th} -bit group address of the memory access operations, and the preprocessed input sequence $\{x(n)\}$ is defined as

$$x((g^{n-k+1})_N) = \left\{ \begin{aligned} & x((g^{n-k+1})_N); \text{if } n - k + 1 \geq 0 \\ & x((g^{(N-1)+(n-k+1)})_N); \text{if } n - k + 1 < 0 \end{aligned} \right\}, \quad (3.51)$$

$$x((g^{n-k+1+\frac{N-1}{2}})_N) = \left\{ \begin{aligned} & x((g^{n-k+1+\frac{N-1}{2}})_N); \text{if } n - k + 1 + \frac{N-1}{2} \geq 0 \\ & x((g^{(N-1)+(n-k+1+\frac{N-1}{2})})_N); \text{if } n - k + 1 + \frac{N-1}{2} < 0 \end{aligned} \right\}. \quad (3.52)$$

The value of m is determined by

$$(g^{n+1})_N + m \cdot N = (g^{n-k+1})_N \cdot (g^k)_N; n, k = 1, \dots, N-1. \quad (3.53)$$

Non-prime length case

By introducing a indirect sequence $\{x(n)\}$, we can express the non-prime length 1-D DCT as

$$Y(k) = [2 \cdot T(k) + x(0)] \cdot \cos\left(\frac{\pi k}{2N}\right); k = 1, \dots, N-1 \quad (3.54)$$

$$T(k) = \sum_{n=1}^{N-1} x(n) \cdot \cos\left(\frac{\pi nk}{N}\right)$$

$$Y(0) = \sum_{n=0}^{N-1} y(n)$$

The sequence $\{x(n)\}$ is defined as

$$\left\{ \begin{array}{l} x(N-1) = y(N-1) \\ x(n) = y(n) - x(n+1); n = 0, \dots, N-2 \end{array} \right\}. \quad (3.55)$$

By introducing a new sequence $\{C(k)\}$ as

$$C(k) = \sum_{n=1}^{N-1} x(n) \cdot \cos[(k^2 - 2nk)\beta] \quad (3.56)$$

with $\beta = \frac{\pi}{2N}$, we can express $\{T(k)\}$ as

$$T(k) = \frac{1}{2 \cdot \cos(k^2 \beta)} [C(k) + C(2N-k)]; k = 1 \dots, N-1 \quad (3.57)$$

Now appropriately evaluating the term $C(k) + C(2N-k)$

$$T(k) = \frac{1}{2 \cdot \cos(k^2 \beta)} [T_c(k) + T_s(k)]; k = 1 \dots, N-1 \quad (3.58)$$

where

$$\begin{aligned} T_c(k) &= \sum_{n=0}^{N-1} c_f((n+k)_N) \cdot \cos(n^2 \beta); k = 1, \dots, N-1 \\ T_s(k) &= \sum_{n=0}^{N-1} s_f((n+k)_N) \cdot \sin(n^2 \beta); k = 1, \dots, N-1 \end{aligned} \quad (3.59)$$

and

$$\left\{ \begin{array}{l} c_f((n+k)_N) = c'((n+k)_{2N}) + (-1)^{m+n} \cdot c'((n+k+N)_{2N}) \\ s_f((n+k)_N) = s'((n+k)_{2N}) + (-1)^{m+n} \cdot s'((n+k+N)_{2N}) \end{array} \right\}; \text{if } N = 2m \quad (3.60)$$

$$\left\{ \begin{array}{l} c_f((n+k)_N) = c'((n+k)_{2N}) + (-1)^{m+n} \cdot c'((n+k+N)_{2N}) \\ s_f((n+k)_N) = s'((n+k)_{2N}) + (-1)^{m+n+1} \cdot s'((n+k+N)_{2N}) \end{array} \right\}; \text{if } N = 2m+1 \quad (3.61)$$

where

$$c'(n) = \left\{ \begin{array}{ll} x_c(n), & n = 1, \dots, N-1 \\ 0, & n = 0, N \\ x_c(2N-n) & n = N+1, \dots, 2N-1 \end{array} \right\} \quad (3.62)$$

$$s'(n) = \left\{ \begin{array}{ll} x_s(n), & n = 1, \dots, N-1 \\ 0, & n = 0, N \\ x_s(2N-n) & n = N+1, \dots, 2N-1 \end{array} \right\} \quad (3.63)$$

and the sequences $\{x_c(n)\}$ and $\{x_s(n)\}$ are defined respectively as

$$\begin{aligned} x_c(n) &= x(n) \cdot \cos(n^2 \beta) \\ x_s(n) &= x(n) \cdot \sin(n^2 \beta) \end{aligned} \quad (3.64)$$

In the above equations, the both DCT kernel operations $T_c(k)$ and $T_s(k)$, are expressed in cyclic convolution forms and thus can be efficiently implemented by GDA. However, the non-prime length DCT algorithm also requires pre-processing as indicated in (3.60) ~ (3.64) and post-processing as indicated in (3.54). This algorithm is useful in realizing the DCT with any length, which can cover the applications with broader ranges in the transform length than the algorithms being developed for 2^p -point DCT.

3.3.2 Numerical stability

Similar to the non-prime length 1-D DHT, the issue of numerical instability is also involved in the non-prime length 1-D DCT algorithm, and causes numerical instability of some results as the denominator in division operation may equal to zero for specific values of k . For ensuring the correctness of non-prime length 1-D DCT, we first introduce a sequence $\{S(k)\}$ that is different from the $\{C(k)\}$ mentioned above. That is,

$$S(k) = \sum_{n=0}^{N-1} x(n) \cdot \sin[(k^2 - 2nk)\beta]; k = 0, 1, \dots, N-1$$

Using this sequence, we can first express the sequence $\{T(k)\}$ as

$$T(k) = \frac{1}{2 \cdot \sin(k^2 \beta)} [S(k) + S(2N - k)]; k = 0, 1, \dots, N-1$$

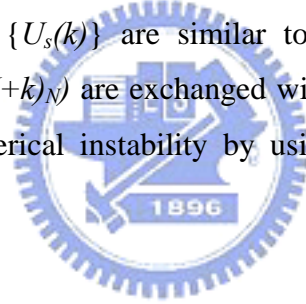
Then, based on the similar procedures shown in before, we can rewrite the $T(k)$ as

$$T(k) = \frac{1}{2 \cdot \sin(k^2 \beta)} [U_c(k) + U_s(k)]; k = 0, 1, \dots, N-1$$

where

$$\begin{aligned} U_c(k) &= \sum_{n=0}^{N-1} -s_f((n+k)_N) \cdot \cos(n^2 \beta); k = 0, 1, \dots, N-1 \\ U_s(k) &= \sum_{n=0}^{N-1} c_f((n+k)_N) \cdot \sin(n^2 \beta); k = 0, 1, \dots, N-1 \end{aligned} \quad (3.65)$$

The sequences $\{U_c(k)\}$ and $\{U_s(k)\}$ are similar to $\{T_c(k)\}$ and $\{T_s(k)\}$, and the operands $c_f((N+k)_N)$ and $s_f((N+k)_N)$ are exchanged with different signs. Thus we also can solve this issue of numerical instability by using simple control in hardware realization.



3.3.3 Architecture design and evaluation

Architecture design

Fig. 3.9 shows the proposed pipeline architecture that realizes the 1-D N-point DCT. It consists of the pre-processing stage, DA processing stage, and post-processing stage. For the 1-D 7-point DCT design example, the input buffer and pre-processing in the preprocessing stage shown as Fig. 3.10 are designed by using the bidirectional shift registers and an accumulator, which is used to generate the data sequence $x(n)$ from input sequence $y(n)$. The detail cycle information shows that the latency consumed by input data sampling and $x(n)$ computation is 14 cycles. The DA processing stage shown as Fig. 3.11, named group distributed arithmetic unit (GDAU), is designed with the proposed memory efficient approach to carry out the computation of $T((3^k)_7)$ in the design example of 1-D 7-point DCT. Due to the same content of group memory, only one group memory in the GDAU is required to compute the outputs of the

separated cyclic operations. In Fig. 3.11, the candidate of DA input for q th bit $X_q = \{x_{3,q}, x_{2,q}, x_{1,q}\}$ is first fed into an address decoder to determine which group it should belong to. The address decoder will compute the seed-value $X'_q = \{x'_{3,q}, x'_{2,q}, x'_{1,q}\}$, group address $G_q = \{g_{2,q}, g_{1,q}\}$, and the rotating factor $R_q = \{r_{2,q}, r_{1,q}\}$ by decoding the input vector according to Table 3.11. Table 3.12 shows the original partial product distributions for computing the outputs of DCT kernel under the same input value. From Table 3.12, the rotation relationship between these partial products is also visible, and then the memory content arrangement in the proposed design is shown in Table 3.13. It is noted that we only need one small group memory module of size $(N-1)/2 \times G_{\text{num}}$ words for computing $T((3^k)_7)$. In above, G_{num} denotes the number of groups in the group memory modules, which is dependent on the transform length N . The post-processing stage shown in Fig. 3.12(a) is used to perform the post computation for GDAU outputs, including the operations of multiplying by two, adding with $x(0)$, and multiplying serially by the cosine coefficients in formulation. Since the operation of multiplying by two is performed by the manner of hardwiring, it has no hardware cost required. The output buffer shown in Fig. 3.12(b) in post-processing stage is used to perform the operations of pre-loadable shifting for serially generating the results of DCT in order.

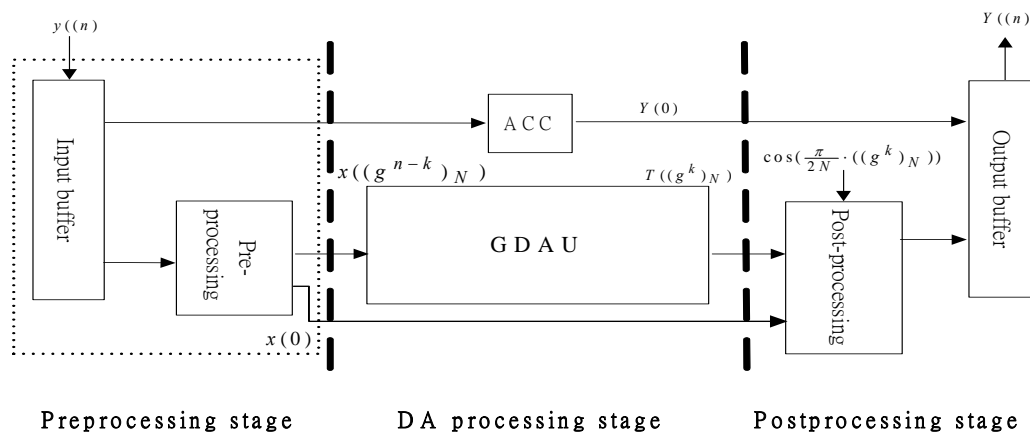
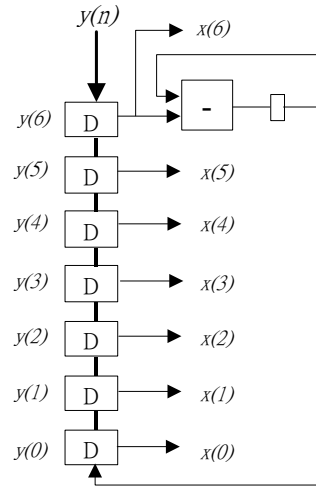


Fig. 3.9: Block diagram of the proposed pipeline architecture for computing the 1-D N-point DCT.



Input buffer	Cycle-based timing information for input data sampling and preprocessing													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
D	$y(0)$	$y(1)$	$y(2)$	$y(3)$	$y(4)$	$y(5)$	$y(6)$	$y(5)$	$y(4)$	$y(3)$	$y(2)$	$y(1)$	$y(0)$	$x(6)$
D		$y(0)$	$y(1)$	$y(2)$	$y(3)$	$y(4)$	$y(5)$	$y(4)$	$y(3)$	$y(2)$	$y(1)$	$y(0)$	$x(6)$	$x(5)$
D			$y(0)$	$y(1)$	$y(2)$	$y(3)$	$y(4)$	$y(3)$	$y(2)$	$y(1)$	$y(0)$	$x(6)$	$x(5)$	$x(4)$
D				$y(0)$	$y(1)$	$y(2)$	$y(3)$	$y(2)$	$y(1)$	$y(0)$	$x(6)$	$x(5)$	$x(4)$	$x(3)$
D					$y(0)$	$y(1)$	$y(2)$	$y(1)$	$y(0)$	$x(6)$	$x(5)$	$x(4)$	$x(3)$	$x(2)$
D						$y(0)$	$y(1)$	$y(0)$	$x(6)$	$x(5)$	$x(4)$	$x(3)$	$x(2)$	$x(1)$
D							$y(0)$	$x(6)$	$x(5)$	$x(4)$	$x(3)$	$x(2)$	$x(1)$	$x(0)$

Fig. 3.10: Design of the preprocessing stage in the 1-D 7-point DCT.

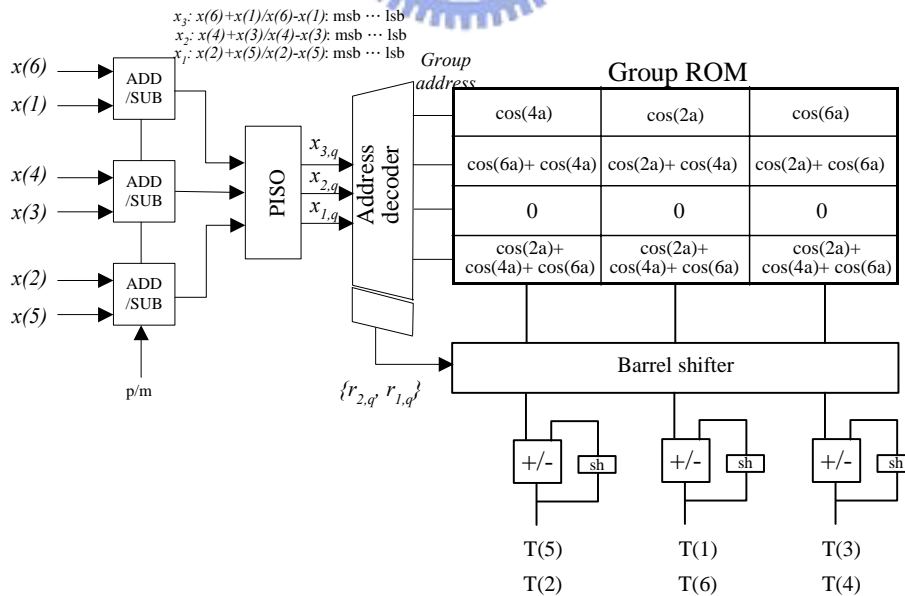


Fig. 3.11: Design of the DA processing stage that is used to compute the kernel of $T((3^k)_7)$ in the 1-D 7-point DCT.

Table 3.11: The seed-value, group address, and rotating factor used in the design of group address decoder of 1-D 7-point DCT.

Grouped DA input value (X_q) $\{x_{3,q}, x_{2,q}, x_{1,q}\}$	Seed-value (X'_q) $\{x'_{3,q}, x'_{2,q}, x'_{1,q}\}$	¹ Rotating factor (R_q) $\{r_{2,q}, r_{1,q}\}$	Group address (G_q) $\{g_{2,q}, g_{1,q}\}$
001	001	0	0
010		1	
100		2	
011	011	0	1
110		1	
101		2	
000	000	0	2
111	111	0	3

Note:

1. Rotating factor denotes the number of position of the output data, corresponding to the candidate of DA input value in a group, should be rotated.

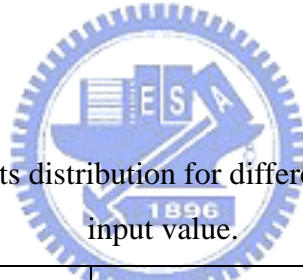


Table 3.12: The partial products distribution for different DCT outputs under the same input value.

Input ($X_{p,q}$)/($X_{m,q}$)	$T(2)/T(5)$	$T(6)/T(1)$	$T(4)/T(3)$	Group address
000	0	0	0	2
001	$\cos(4a)$	$\cos(2a)$	$\cos(6a)$	0
010	$\cos(6a)$	$\cos(4a)$	$\cos(2a)$	0
011	$\cos(6a)+$ $\cos(4a)$	$\cos(2a)+$ $\cos(4a)$	$\cos(2a)+$ $\cos(6a)$	1
100	$\cos(2a)$	$\cos(6a)$	$\cos(4a)$	0
101	$\cos(2a)+$ $\cos(4a)$	$\cos(2a)+$ $\cos(6a)$	$\cos(6a)+$ $\cos(4a)$	1
110	$\cos(2a)+$ $\cos(6a)$	$\cos(6a)+$ $\cos(4a)$	$\cos(2a)+$ $\cos(4a)$	1
111	$\cos(2a)+\cos(4a)+$ $\cos(6a)$	$\cos(2a)+\cos(4a)+$ $\cos(6a)$	$\cos(2a)+\cos(4a)+$ $\cos(6a)$	3

Note:

1. $X_{p,q}$ denotes the sum of inputs for the q th bit.
2. $X_{m,q}$ denotes the difference of inputs for the q th bit.

Table 3.13: 8-word memory contents arranged into groups.

Original address	Group address			
1, 2, 4	0	$\cos(4a)$	$\cos(2a)$	$\cos(6a)$
3, 5, 6	1	$\cos(6a) + \cos(4a)$	$\cos(2a) + \cos(4a)$	$\cos(2a) + \cos(6a)$
0	2	0	0	0
7	3	$\cos(2a) + \cos(4a) + \cos(6a)$	$\cos(2a) + \cos(4a) + \cos(6a)$	$\cos(2a) + \cos(4a) + \cos(6a)$

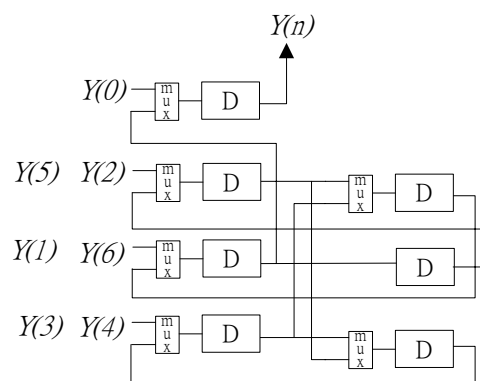
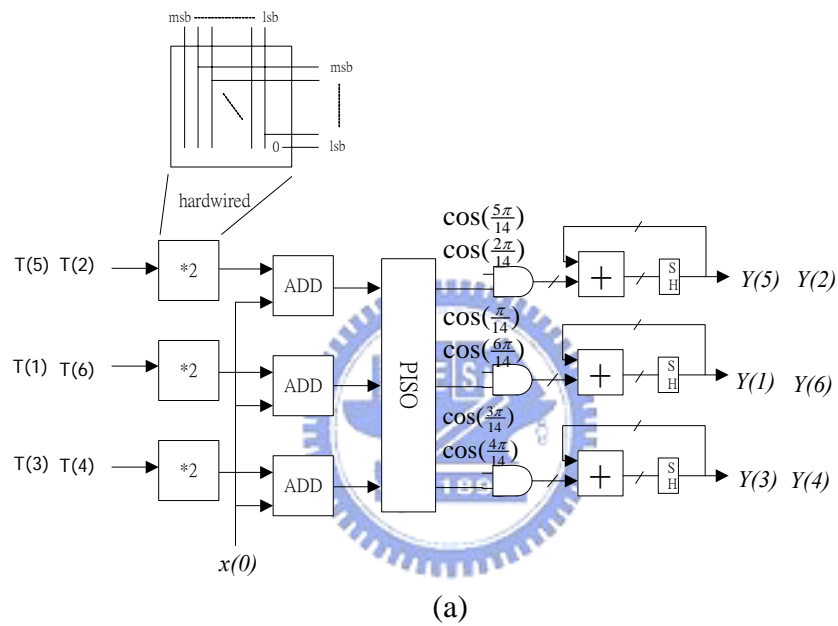


Fig. 3.12: Design of the post-processing stage in the 1-D 7-point DCT including (a) the post-processing, and (b) the output buffer.

Design evaluation

In this section, we will illustrate the performance evaluation of the design using the proposed design approach and some existing DCT designs. The existing DCT designs used in this evaluation include memory-based systolic array design [33], direct DA design [34], OBC DA design [35] and adder-based DA design [52]. For a fair comparison, we also adopt the Avanti 0.35 μm , 3.3-volt CMOS cell-library [43] in the performance evaluation in terms of the delay time and area cost. According to these two measures, we can evaluate these designs in terms of delay-area product with respect to different values of N . Table 3.14 shows the comparisons of these designs. The design in [33] is a memory-based systolic array design. It needs about N adders, $N-1$ 16-bit Flip-Flop and $(N-1) \cdot 2^{(L/2)}$ words of memory if the memory tables in the design are partitioned once. The silicon area of this design is equal to $1237N-1217 \text{ Kum}^2$. The design in [34] is the conventional memory-based DA design; it requires about N 16-bit adders, N 16-bit Flip-Flops used for PISO and $2^N \cdot N$ words of memory. The silicon area of this design is equal to $13.7N+4.75 \cdot 2^N \cdot N \text{ Kum}^2$. The design in [35] is the modified memory-based DA design using the reduction technique of OBC, it requires about $2N$ 16-bit adders, N 16-bit Flip-Flop and $2^{(N-2)} \cdot N$ words of memory. The silicon area of this design is equal to $19.6N+4.75 \cdot 2^{(N-2)} \cdot N \text{ Kum}^2$. The design in [52] is the adder-based DA design; it requires about four 16-bit multipliers, $2N+2L+5$ 16-bit adders, $4N+3$ 16-bit Flip-Flops and $4N$ words of RAM. The silicon area of this design is equal to $73N+380 \text{ Kum}^2$. Fig. 3.13 shows the delay-area product of the proposed design and the existing designs [33]-[35][52] in realizing the 1-D DCT with various values of N . As shown in Fig. 3.13, in case of 16-bit data word-length, the delay-area product of the proposed design is much smaller than the memory-based systolic array DCT design [33] and the other DA-based designs [34][35][52].

Table 3.14: The comparison of the proposed design and the existing DCT designs [33]-[35][52] in realizing the 1-D N-point DCT in terms of delay and silicon area.

	Cycle time (T)	Mul (16-bit)	Adder (16-bit)	FF (16-bit)	memory (16-bit)	Barrel shifter (16-bit)	RAM (16-bit)	Delay*Area ($ns * K \mu m^2$)
Guo [33] (Memory-based systolic array)	$T = tmux + trom + tadd + tadd$		$5.9N$	$7.8(N-1)$	$1216 \cdot (N-1)$		$7.4 \cdot (N-1)$	$[(N-1)T/N] (1237N-1217)$
White [34] (directly DA)	$T = trom + tadd$		$5.9N$	$23.4N$	$4.75 \cdot 2^N \cdot N$			$[(16T)/N] * (29.3N + 4.75 \cdot 2^N \cdot N)$
Choi [35] (OBC-based DA)	$T = trom + 2 tadd$		$11.8N$	$23.4N$	$4.75 \cdot 2^{(N-2)} \cdot N$			$[(16T)/N] * (35.2N + 4.75 \cdot 2^{(N-2)} \cdot N)$
Guo [52] (Adder-based DA)	$T = \max\{tmul, tadd + tff\}$	$34.6 * 4$	$5.9(2N+37)$	$7.8(4N+3)$			$7.4(4N)$	$[(NT)/N] * (73N+380)$
The proposed design	$T = trom + tbr + tadd$		$11.8N$	$23.4(N-1)$	$4.75 \cdot 2^{\frac{6(N-1)}{7-2}} \cdot \frac{(N-1)}{2}$	$12 * [-0.072 + 0.435 * (N-1) + 0.053 * (N-1)^2]$		$[(32T)/N] * [-28.8 + 39.1N + 0.64N^2 + [4.75 \cdot 2^{\frac{6(N-1)}{7-2}} \cdot \frac{(N-1)}{2}]]$

Note:

1. $tmul$ denotes the delay time of multiplier.
2. $tmux$ denotes the delay time of multiplexer.
3. $tadd$ denotes the delay time of adder.
4. $trom$ denotes the access time of memory associated with N for DA-based design or associated with wordlength for memory-based systolic array design.
5. tbr denotes the delay time of barrel shifter associated with N .

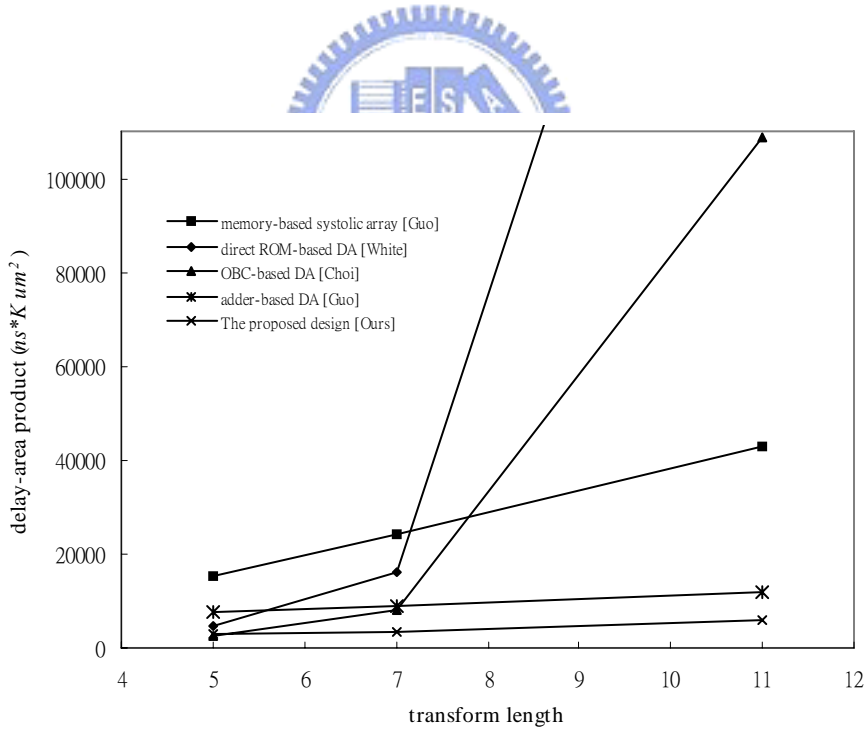


Fig. 3.13: The delay-area product of the proposed design and the existing DCT designs [33]-[35][52] in realizing the 1-D DCT.

3.3.4 Chip implementation

We have verified the proposed design for 1-D 7-point DCT in VERILOG modeling. According to the synthesis result with Avanti 0.35um cell-library, this design consumes 7485 gates, and possesses the maximum path delay of 12.1ns. The working frequency of the chip is above 82.6 MHz. In other words, the chip can maintain the throughput rate of 18.1 M samples/second, i.e., $(82.6 \text{ MHz} / 32 \text{ cycles}) * 7 \text{ samples}$. Fig. 3.14 shows the layout view of the 1-D 7-point GDA-based DCT design fabricated using TSMC 0.35um CMOS 1P4M process. The core size of proposed DCT design is equal to $1734 * 1732 \text{ um}^2$.

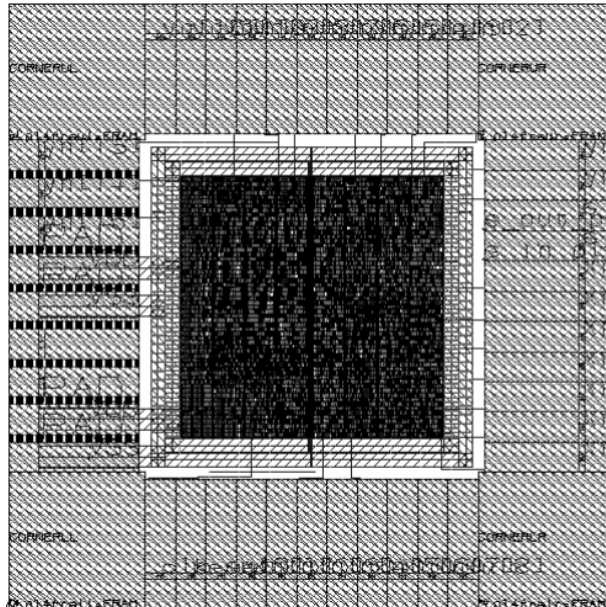


Fig. 3.14: Layout view of the 1-D 7-point GDA-based DCT design.

Chapter 4

Long-length DSST's designs

Regarding the long length DSST's design, we adopt the methodology of two level decomposition for realization of the long length DSST's with short ones. In our research, we not only partition the cyclic convolution in DSST's kernel with Agwal-Cooley algorithm and pseudocirculant factorization algorithm for the cases of prime-length and non-prime length respectively but also combine with the other decomposition algorithms [53]-[56] for different cases of transform length, such as the Cooley-Tukey algorithm for the case of $(a, b) > 1$, where the transform length $N = a \cdot b$, prime factor algorithm (PFA) for the case of $(a, b) = 1$, and Rader's algorithm for the case of $N = p^c$, where p is prime, and $c > 1$, to decompose the long length DSST's for short ones.

4.1 Decomposition of long-length DSST's

4.1.1 Cooly-Tukey Algorithm

For the long-length 1-D DFT with non-prime length, firstly we can apply Cooly-Tukey FFT Algorithm to decompose a 1-D N-point DFT into a 2-D DFT with the lengths of N_1 by N_2 . Based on the common factor map (CFM) [57], this algorithm can map the time index n and frequency index k in 1-D DFT into the time indices n_1, n_2 and frequency indices k_1, k_2 as

$$\begin{aligned} n &= N_2 \cdot n_1 + n_2 \\ k &= k_1 + N_1 \cdot k_2 \end{aligned} \tag{4.1}$$

where $0 \leq n_1, k_1 \leq N_1 - 1$, $0 \leq n_2, k_2 \leq N_2 - 1$. Thus, the 1-D N-point DFT can be decomposed into a 2-D $N_1 \times N_2$ DFT as shown in the following

$$\begin{aligned}
Y(k) &= \sum_{n=0}^{N-1} x(n) \cdot W_N^{nk}; k = 0, 1, \dots, N-1 \\
&= Y(k_1 + N_1 k_2) \\
&= \sum_{n_2=0}^{N_2-1} \left[\sum_{n_1=0}^{N_1-1} x(N_2 n_1 + n_2) \cdot W_{N_1}^{n_1 k_1} \right] \cdot W_N^{n_2 k_1} \cdot W_{N_2}^{n_2 k_2} \\
&= \sum_{n_2=0}^{N_2-1} [G(n_2, k_1) \cdot W_N^{n_2 k_1}] \cdot W_{N_2}^{n_2 k_2} \\
&= \sum_{n_2=0}^{N_2-1} \tilde{G}(n_2, k_1) \cdot W_{N_2}^{n_2 k_2}
\end{aligned} \tag{4.2}$$

where $G(n_2, k_1) = \sum_{n_1=0}^{N_1-1} x(N_2 n_1 + n_2) \cdot W_{N_1}^{n_1 k_1}$, $\tilde{G}(n_2, k_1) = G(n_2, k_1) \cdot W_N^{n_2 k_1}$

In the above derivation, we can see that $Y(k)$ can be viewed as a 1-D N_2 -point DFT with input $\tilde{G}(n_2, k_1)$, and $G(n_2, k_1)$ can be viewed as a 1-D N_1 -point DFT with input $x(N_2 n_1 + n_2)$. We can obtain $\tilde{G}(n_2, k_1)$ by multiplying $G(n_2, k_1)$ with a twiddle factor $W_N^{n_2 k_1}$. These twiddle factors multiplications can be absorbed into the post-processing in the cyclic convolution formulation of 1-D N_1 -point DFT. By realizing the computation of $\tilde{G}(n_2, k_1)$ and $G(n_2, k_1)$ based on the proposed GDA approach, we can achieve the design of long-length 1-D DFT

4.1.2 Prime Factor Algorithm

Prime Factor Decomposition of 1D DFT

A design example of decomposing 1-D N -point DFT into 2-D $N_0 \times N_1$ DFT is illustrated in the following. Based on prime factor map (PFM) [57], the mapping of 1-D indices n and k to 2-D indices n_1, n_2, k_1 , and k_2 are typically given by:

$$\begin{aligned}
n &= (N_1 \cdot n_0 + N_0 \cdot n_1)_N \\
k &= (N_1(N_1^{-1})_{N_0} \cdot k_0 + N_0(N_0^{-1})_{N_1} \cdot k_1)_N
\end{aligned} \tag{4.3}$$

where $0 \leq n_0, k_0 \leq N_0 - 1$, $0 \leq n_1, k_1 \leq N_1 - 1$, with N_0 and N_1 the relatively prime factors of the transform length N . Then, the 1-D N -point DFT shown in (4.2) can be

decomposed into the 2-D $N_0 \times N_1$ DFT as

$$Y[k_1, k_0] = \sum_{n_1=0}^{N_1-1} \sum_{n_0=0}^{N_0-1} x[n_1, n_0] \cdot W_{N_0}^{n_0 k_0} \cdot W_{N_1}^{n_1 k_1} \quad (4.4)$$

Using the index mappings, we can express the DFT in (4.4) as

$$Y((N_1(N_1^{-1})_{N_0} \cdot k_0 + N_0(N_0^{-1})_{N_1} \cdot k_1)_N) = \sum_{n_1=0}^{N_1-1} \left(\sum_{n_0=0}^{N_0-1} x((N_1 \cdot n_0 + N_0 \cdot n_1)_N) \cdot W_{N_0}^{n_0 k_0} \right) \cdot W_{N_1}^{n_1 k_1} \quad (4.5)$$

Now, according to the PFA, the 1-D N -point DFT is decomposed into a 2-D $N_0 \times N_1$ DFT with no twiddle factor such that the GDA design can directly be applied to realize each of the 1-D N_0 -point DFT and N_1 -point DFT computation.

Prime Factor Decomposition of 1D DHT

For the long length DHT design, we firstly exploit the prime factor algorithm (PFA) to decompose the long length DHT into shortened ones and then implement each of the shortened DHT [53][56][58][59]. Based on PFA, the computation of the long length DHT can be effectively achieved, whereby a 1-D DHT of $N = N_1 \times N_2$ samples can be formulated into a separable 2-D $N_1 \times N_2$ DHT. The decomposition of 1-D N -point DHT is briefly illustrated in the following.

For input index n and output index k , the mapping of 1-D indices n and k to 2-D indices n_1, n_2, k_1 , and k_2 are typically given by

$$\begin{aligned} n &= (N_2 \cdot n_1 + N_1 \cdot n_2)_N \\ k &= (N_2(N_2^{-1})_{N_1} \cdot k_1 + N_1(N_1^{-1})_{N_2} \cdot k_2)_N \end{aligned} \quad (4.6)$$

where $0 \leq n_1, k_1 \leq N_1 - 1$, $0 \leq n_2, k_2 \leq N_2 - 1$, with N_1 and N_2 the prime factors of the transform length N . Thus, the 1-D N -point DHT can be decomposed into a 2-D $N_1 \times N_2$ DHT as shown in the following

$$\Psi(k_1, k_2) = \frac{1}{2} [Y_1(k_1, k_2) + Y_2(N_1 - k_1, k_2) + Y_3(k_1, N_2 - k_2) + Y_4(N_1 - k_1, N_2 - k_2)] \quad (4.7)$$

$$\text{where } Y_i(k_1, k_2) = \sum_{n_2=0}^{N_2-1} \left\{ \sum_{n_1=0}^{N_1-1} x(n_1, n_2) \cdot \text{cas}\left(\frac{2\pi}{N_1} n_1 k_1\right) \right\} \cdot \text{cas}\left(\frac{2\pi}{N_2} n_2 k_2\right),$$

and $\text{cas}\theta = \cos\theta + \sin\theta$.

Now, the 1-D N -point DHT is decomposed into a 2-D $N_1 \times N_2$ DHT. In this case, the shortened DHTs are much more efficient in the hardware realization than the direct realization of long length DHT.

4.1.3 Rader's Algorithm

[53][61] have shown that the DFT/DHT can be converted to convolution when the transform length N is a power of the odd prime, i.e., $N = p^r$ for a prime $p \neq 2$. For the conversion with this algorithm, we must first remove all integers which contain a factor p from the set $\{1, 2, \dots, N-1\}$ to get a cyclic group with $p^{r-1}(p-1)$ elements. This cyclic group leads to a circular convolution of length $p^{r-1}(p-1)$ as before. The remaining computation consists of two DFT's of length p^{r-1} . The generalized algorithm shows that if $N = p^r$ the length N transform is computed with one length $p^{r-1}(p-1)$ circular convolution, two $p^{r-2}(p-1)$ circular convolutions, four $p^{r-3}(p-1)$ circular convolutions, ..., terminating $2^{r-1}(p-1)$ circular convolutions. An example of DFT with the length $N = 9 = 3^2$ illustrates this algorithm in detail begin with the matrix representation as

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \\ Y(4) \\ Y(5) \\ Y(6) \\ Y(7) \\ Y(8) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W_9^1 & W_9^2 & W_9^3 & W_9^4 & W_9^5 & W_9^6 & W_9^7 & W_9^8 \\ 1 & W_9^2 & W_9^4 & W_9^6 & W_9^8 & W_9^1 & W_9^3 & W_9^5 & W_9^7 \\ 1 & W_9^3 & W_9^6 & 1 & W_9^3 & W_9^6 & 1 & W_9^3 & W_9^6 \\ 1 & W_9^4 & W_9^8 & W_9^3 & W_9^7 & W_9^2 & W_9^6 & W_9^1 & W_9^5 \\ 1 & W_9^5 & W_9^1 & W_9^6 & W_9^2 & W_9^7 & W_9^3 & W_9^8 & W_9^4 \\ 1 & W_9^6 & W_9^3 & 1 & W_9^6 & W_9^3 & 1 & W_9^6 & W_9^3 \\ 1 & W_9^7 & W_9^5 & W_9^3 & W_9^1 & W_9^8 & W_9^6 & W_9^4 & W_9^2 \\ 1 & W_9^8 & W_9^7 & W_9^6 & W_9^5 & W_9^4 & W_9^3 & W_9^2 & W_9^1 \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \\ x(8) \end{bmatrix} \quad (4.8)$$

We remove rows and columns corresponding to the indices of 0, 3, and 6, and compute the remaining length six transform

$$\begin{bmatrix} Y'(1) \\ Y'(2) \\ Y'(4) \\ Y'(5) \\ Y'(7) \\ Y'(8) \end{bmatrix} = \begin{bmatrix} W_9^1 & W_9^2 & W_9^4 & W_9^5 & W_9^7 & W_9^8 \\ W_9^2 & W_9^4 & W_9^8 & W_9^1 & W_9^5 & W_9^7 \\ W_9^4 & W_9^8 & W_9^7 & W_9^2 & W_9^1 & W_9^5 \\ W_9^5 & W_9^1 & W_9^2 & W_9^7 & W_9^8 & W_9^4 \\ W_9^7 & W_9^5 & W_9^1 & W_9^8 & W_9^4 & W_9^2 \\ W_9^8 & W_9^7 & W_9^5 & W_9^4 & W_9^2 & W_9^1 \end{bmatrix} \cdot \begin{bmatrix} x(1) \\ x(2) \\ x(4) \\ x(5) \\ x(7) \\ x(8) \end{bmatrix} \quad (4.9)$$

using the permutation

$$n = 2^m \bmod 9, \quad \begin{matrix} n = 1,2,4,8,7,5 \\ m = 0,1,2,3,4,5 \end{matrix}$$

to obtain the circular convolution (with input reversed as before)

$$\begin{bmatrix} Y'(1) \\ Y'(2) \\ Y'(4) \\ Y'(8) \\ Y'(7) \\ Y'(5) \end{bmatrix} = \begin{bmatrix} W_9^1 & W_9^5 & W_9^7 & W_9^8 & W_9^4 & W_9^2 \\ W_9^2 & W_9^1 & W_9^5 & W_9^7 & W_9^8 & W_9^4 \\ W_9^4 & W_9^2 & W_9^1 & W_9^5 & W_9^7 & W_9^8 \\ W_9^8 & W_9^4 & W_9^2 & W_9^1 & W_9^5 & W_9^7 \\ W_9^7 & W_9^8 & W_9^4 & W_9^2 & W_9^1 & W_9^5 \\ W_9^5 & W_9^7 & W_9^8 & W_9^4 & W_9^2 & W_9^1 \end{bmatrix} \cdot \begin{bmatrix} x(1) \\ x(5) \\ x(7) \\ x(8) \\ x(4) \\ x(2) \end{bmatrix} \quad (4.10)$$

and

$$\begin{bmatrix} Y(0) \\ Y(3) \\ Y(6) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W_9^3 & W_9^6 & 1 & W_9^3 & W_9^6 & 1 & W_9^3 & W_9^6 \\ 1 & W_9^6 & W_9^3 & 1 & W_9^6 & W_9^3 & 1 & W_9^6 & W_9^3 \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \\ x(8) \end{bmatrix} \quad (4.11)$$

$$= \begin{bmatrix} 1 & 1 & 1 \\ 1 & W_3^1 & W_3^2 \\ 1 & W_3^2 & W_3^1 \end{bmatrix} \cdot \begin{bmatrix} x(0) + x(3) + x(6) \\ x(1) + x(4) + x(7) \\ x(2) + x(5) + x(8) \end{bmatrix}$$

where $W_9^3 = W_3^1, W_9^6 = W_3^2$.

For the deleted column we have

$$\begin{bmatrix} Y''(0) \\ Y''(1) \\ Y''(2) \\ Y''(3) \\ Y''(4) \\ Y''(5) \\ Y''(6) \\ Y''(7) \\ Y''(8) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & W_9^3 & W_9^6 \\ 1 & W_9^6 & W_9^3 \\ 1 & 1 & 1 \\ 1 & W_9^3 & W_9^6 \\ 1 & W_9^6 & W_9^3 \\ 1 & 1 & 1 \\ 1 & W_9^3 & W_9^6 \\ 1 & W_9^6 & W_9^3 \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(3) \\ x(6) \end{bmatrix} \quad (4.12)$$

and the equivalent form as

$$\begin{bmatrix} Y''(0) \\ Y''(1) \\ Y''(2) \end{bmatrix} = \begin{bmatrix} Y''(3) \\ Y''(4) \\ Y''(5) \end{bmatrix} = \begin{bmatrix} Y''(6) \\ Y''(7) \\ Y''(8) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & W_3^1 & W_3^2 \\ 1 & W_3^2 & W_3^1 \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(3) \\ x(6) \end{bmatrix} \quad (4.13)$$

Only the last two entries $Y''(1)$ and $Y''(2)$ are needed from (4.13) to compute the rest of (4.8) as

$$\begin{bmatrix} Y(1) \\ Y(2) \\ Y(4) \\ Y(8) \\ Y(7) \\ Y(5) \end{bmatrix} = \begin{bmatrix} Y'(1) \\ Y'(2) \\ Y'(4) \\ Y'(8) \\ Y'(7) \\ Y'(5) \end{bmatrix} + \begin{bmatrix} Y''(1) \\ Y''(2) \\ Y''(1) \\ Y''(2) \\ Y''(1) \\ Y''(2) \end{bmatrix} \quad (4.14)$$

As for the DHT with the length of power of odd prime, the derivation of cyclic convolution formulation is similar to that of the DFT.

4.2 Long length DHT Design and Evaluation

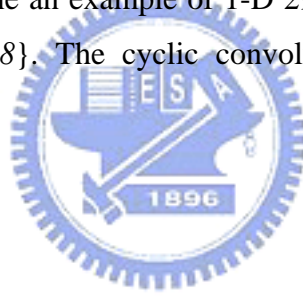
Architecture design

To facilitate the GDA realization of the shortened DHT without suffering from exponential memory size, after algorithm decomposition of the long-length DHT, we need to further partition these shortened DHTs, where the cyclic property must be preserved in each partition of them. We recall the kernel $T((g^k)_N)$ of 1-D DHT in (3.23), and further partition it into short ones by the Agarwal-Cooley algorithm. With this algorithm, we can preserve the cyclic property in each of the partitions and thus

apply GDA design approach efficiently to the implementation of the shortened 1-D DHT. It means that the original $(N-1)$ -point cyclic convolution can be partitioned into $s * s$ short-length cyclic convolutions with the size of $t * t$, where s and t denote the partitioning factors, i.e., $N-1 = s * t$. Thus, the permuted $T((g^k)_N)$ in 1-D DHT formulation can be written as the sum of some short-length cyclic convolutions. That is,

$$\begin{aligned}
 T((g^{k'})_N) &= T_1((g^{k'})_N) + T_2((g^{k'})_N) + \dots + T_i((g^{k'})_N) + \dots + T_s((g^{k'})_N) \\
 &= \sum_{n_1=1}^{(N-1)/s} x((g^{n_1-k'})_N) \cdot H_N^{(g^{n_1})_N} + \sum_{n_2=(N-1)/s+1}^{2(N-1)/s} x((g^{n_2-k'})_N) \cdot H_N^{(g^{n_2})_N} \dots \dots \\
 &\quad + \sum_{n_i=(i-1)(N-1)/s+1}^{i(N-1)/s} x((g^{n_i-k'})_N) \cdot H_N^{(g^{n_i})_N} \dots \dots + \sum_{n_s=(s-1)(N-1)/s+1}^{N-1} x((g^{n_s-k'})_N) \cdot H_N^{(g^{n_s})_N}
 \end{aligned} \tag{4.31}$$

where $n'_i = 1 + (n_i - 1)_t + t(n_i - 1)_t + t(s - \text{int}((n_i - 1)/t))_s$ and $k' = 1 + (k - 1)_t + t(k - 1)_t + t(s - \text{int}((k - 1)/t))_s$ denote the mapped indices for maintaining the partitioned matrix still preserves the cyclic property. Let us examine an example of 1-D 29-point DHT with the real input sequence $\{x(n), n=0, 1, \dots, 28\}$. The cyclic convolution form of $T((g^k)_N)$ can be expressed as



2	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	2
4	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	4
8	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	8
16	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	16
3	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	3
6	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	6
12	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	12
24	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	24
19	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	19
9	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	9
18	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	18
7	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	7
14	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	14
28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	28
27	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	14	27
25	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	7	25
21	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	18	21
13	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	9	13
26	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	19	26
23	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	24	23
17	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	12	17
5	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	6	5
10	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	3	10
20	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	16	20
11	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	8	11
22	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	4	22
15	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	2	15
1	2	4	8	16	3	6	12	24	19	9	18	7	14	28	27	25	21	13	26	23	17	5	10	20	11	22	15	1	1

$T()$

$x()$

$H_N^{()}$

(4.32)

Exploiting the Agarwal-Cooley algorithm, we can convert the cyclic convolution with long length to a four by four short-length cyclic convolutions, and express (4.32) as

GDAU1

GDAU2

GDAU3

GDAU4

2	1	24	25	20	16	7	23	17	2	19	21	11	3	14	28	5	4	9	13	22	6	12	27	10	8	18	26	15	2
19	23	1	24	25	20	16	7	14	17	2	19	21	11	3	6	28	5	4	9	13	22	15	12	27	10	8	18	26	19
21	7	23	1	24	25	20	16	3	14	17	2	19	21	11	22	6	28	5	4	9	13	26	15	12	27	10	8	18	21
11	16	7	23	1	24	25	20	11	3	14	17	2	19	21	13	22	6	28	5	4	9	18	26	15	12	27	10	8	11
3	20	16	7	23	1	24	25	21	11	3	14	17	2	19	9	13	22	6	28	5	4	8	18	26	15	12	27	10	3
14	25	20	16	7	23	1	24	19	21	11	3	14	17	2	4	9	13	22	6	28	5	10	8	18	26	15	12	27	14
17	24	25	20	16	7	23	1	2	19	21	11	3	14	17	5	4	9	13	22	6	28	27	10	8	18	26	15	12	17
5	12	27	10	8	18	26	15	1	24	25	20	16	7	23	17	2	19	21	11	3	14	28	5	4	9	13	22	6	5
4	15	12	27	10	8	18	26	23	1	24	25	20	16	7	14	17	2	19	21	11	3	6	28	5	4	9	13	22	4
9	26	15	12	27	10	8	18	7	23	1	24	25	20	16	3	14	17	2	19	21	11	22	6	28	5	4	9	13	9
13	18	26	15	12	27	10	8	16	7	23	1	24	25	20	11	3	14	17	2	19	21	13	22	6	28	5	4	9	13
22	8	18	26	15	12	27	10	20	16	7	23	1	24	25	21	11	3	14	17	2	19	9	13	22	6	28	5	4	22
6	10	8	18	26	15	12	27	25	20	16	7	23	1	24	19	21	11	3	14	17	2	4	9	13	22	6	28	5	6
28	27	10	8	18	26	15	12	24	25	20	16	7	23	1	2	19	21	11	3	14	17	5	4	9	13	22	6	28	28
27	28	5	4	9	13	22	6	12	27	10	8	18	26	15	1	24	25	20	16	7	23	17	2	19	21	11	3	14	27
10	6	28	5	4	9	13	22	15	12	27	10	8	18	26	23	1	24	25	20	16	7	14	17	2	19	21	11	3	10
8	22	6	28	5	4	9	13	26	15	12	27	10	8	18	7	23	1	24	25	20	16	3	14	17	2	19	21	11	8
18	13	22	6	28	5	4	9	18	26	15	12	27	10	8	16	7	23	1	24	25	20	11	3	14	17	2	19	21	18
26	9	13	22	6	28	5	4	8	18	26	15	12	27	10	20	16	7	23	1	24	25	21	11	3	14	17	2	19	26
15	4	9	13	22	6	28	5	10	8	18	26	15	12	27	25	20	16	7	23	1	24	19	21	11	3	14	17	2	15
12	5	4	9	13	22	6	28	27	10	8	18	26	15	12	24	25	20	16	7	23	1	2	19	21	11	3	14	17	12
24	17	2	19	21	11	3	14	28	5	4	9	13	22	6	12	27	10	8	18	26	15	1	24	25	20	16	7	23	24
25	14	17	2	19	21	11	3	6	28	5	4	9	13	22	15	12	27	10	8	18	26	23	1	24	25	20	16	7	25
20	3	14	17	2	19	21	11	22	6	28	5	4	9	13	26	15	12	27	10	8	18	7	23	1	24	25	20	16	20
16	11	3	14	17	2	19	21	13	22	6	28	5	4	9	18	26	15	12	27	10	8	16	7	23	1	24	25	20	16
7	21	11	3	14	17	2	19	9	13	22	6	28	5	4	8	18	26	15	12	27	10	20	16	7	23	1	24	25	7
23	19	21	11	3	14	17	2	4	9	13	22	6	28	5	10	8	18	26	15	12	27	25	20	16	7	23	1	24	23
1	2	19	21	11	3	14	17	5	4	9	13	22	6	28	27	10	8	18	26	15	12	24	25	20	16	7	23	1	1

 $T()$ $x()$ $H_N()$

(4.33)

For facilitating the utilization of GDA approach, we can express each of the shortened cyclic convolutions in (4.33) as

$$T_i((g^{k'})_N) = -T_0((g^{k'+R_0})_N) + \sum_{q=1}^{L-1} T_q((g^{k'+R_q})_N) \cdot 2^{-q}; \quad i=1 \cdots s \quad (4.34)$$

$$\text{where } T_q((g^{k'+R_q})_N) = \sum_{n_i=[(i-1)(N-1)/s]+1}^{i(N-1)/s} x_q((g^{n_i-k'-R_q})_N) \cdot H_N^{(g^{n_i})_N}. \quad (4.35)$$

where L denotes word length of the input data x , N denotes the transform length, and R_q denotes the rotating factor.

In the following, we intend to illustrate the hardware realization in detail through a 1-D 29-point DHT. We make use of the partitioning scheme of cyclic convolution such that the length of cyclic convolution can be partitioned into the

composition of short ones that can be realized efficiently by the proposed GDA design for achieving low hardware cost. Referring to the reformulation of 1-D DHT in (4.33), we can realize the 28-point cyclic convolution used in 1-D 29-point DHT through the summation of four 7-point cyclic convolutions four times since four sets of the outputs in the 28-point cyclic convolution can be computed by using identical four 7-point GDA units (i.e., GDAU1~GDAU4), where the blocks of input data should be rotated for each of the summation computations. The idea of exploiting computation sharing on the content of memory not only efficiently reduces the memory cost with the trade-off in slowing down the data throughput rate, but also achieves good performance of the proposed design in terms of the hardware cost and average computation time as we shall illustrate later.



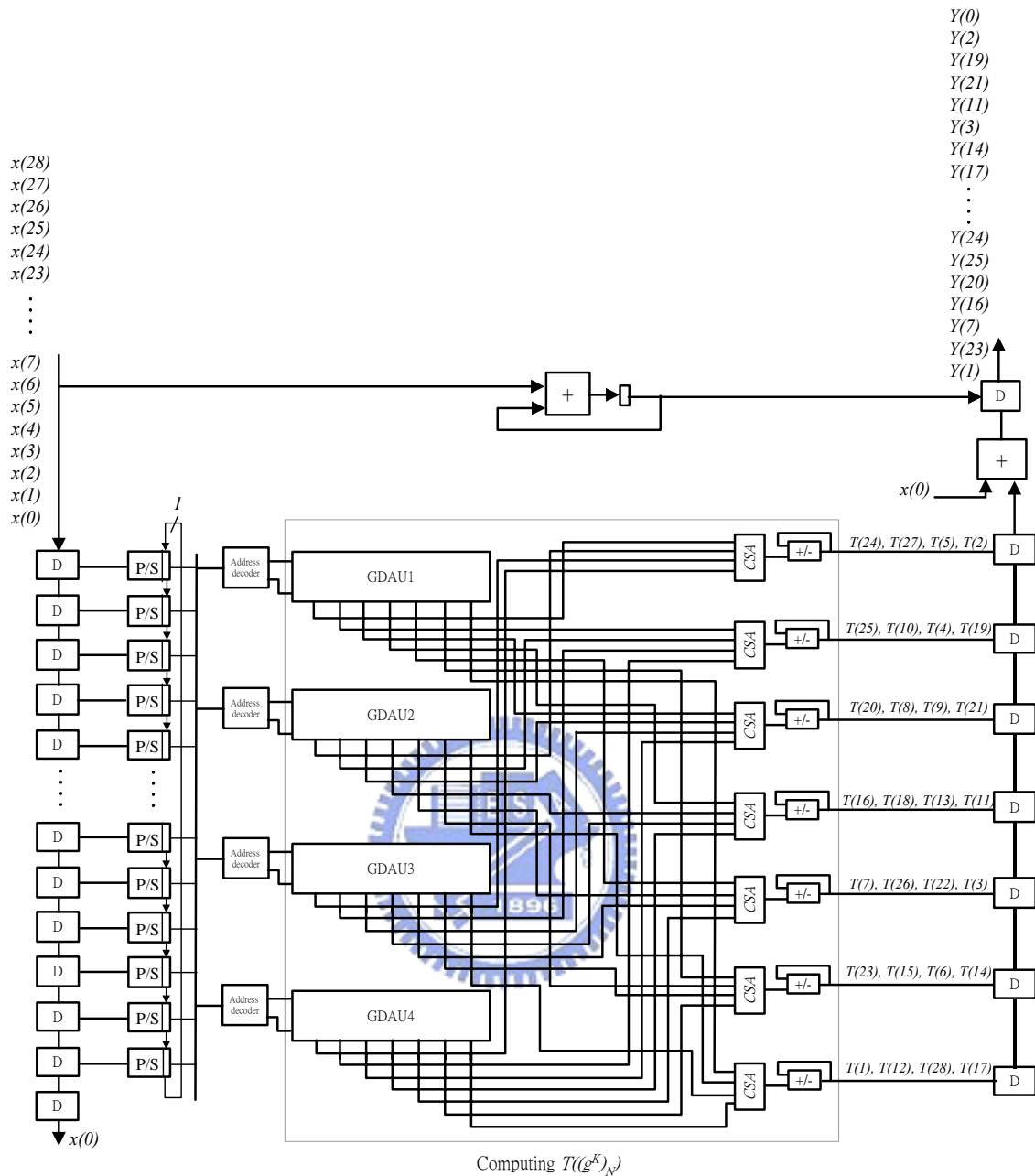


Fig. 4.1: The GDA-based architecture design for 1-D 29-point DHT example

With the derivation mentioned above, Fig. 4.1 shows the GDA-based architecture of 1-D 29-point DHT. It is composed of the GDA unit (GDAU), address decoders, adders/subtractors, accumulators, and parallel-to-serial (P/S) converters. The computation of this design is illustrated as follows. The input vector X_q , which can be $\{x_q(1), x_q(24), x_q(25), x_q(20), x_q(16), x_q(7), x_q(23)\}, \{x_q(17), x_q(2), x_q(19), x_q(21), x_q(11), x_q(3), x_q(14)\}, \{x_q(28), x_q(5), x_q(4), x_q(9), x_q(13), x_q(22), x_q(6)\}$, or

$\{x_q(12), x_q(27), x_q(10), x_q(8), x_q(18), x_q(26), x_q(15)\}$, is first fed into the address decoder to determine which group the DA input belongs to and how many positions the outputs should be rotated. The rotating factor $R_q = \{r_q(1), r_q(2), r_q(3)\}$ decoded from the address decoder is used to control how many bits the barrel shifter should be rotated left, where L_{ROM} 7-bit barrel shifters are involved in the GDAU and L_{ROM} denotes the word length of memory. Table 4.1 shows the relationship between the original DA input address and the group address as well as the rotating factor.

Table 4.1: Function of the address decoders in the 1-D 29-point DHT design

Group number	Grouped candidates of DA input (X_q) $\{x_q(1), x_q(24), x_q(25), x_q(20), x_q(16), x_q(7), x_q(23)\},$ $\{x_q(17), x_q(2), x_q(19), x_q(21), x_q(11), x_q(3), x_q(14)\},$ $\{x_q(28), x_q(5), x_q(4), x_q(9), x_q(13), x_q(22), x_q(6)\},$ or $\{x_q(12), x_q(27), x_q(10), x_q(8), x_q(18), x_q(26), x_q(15)\}$	Rotating-left factor (R_q) $\{r_q(1), r_q(2), r_q(3)\}$	Group address (G_q) $\{g_q(1), g_q(2), g_q(3), g_q(4), g_q(5)\}$	Group number	Grouped candidates of DA input (X_q) $\{x_q(1), x_q(24), x_q(25), x_q(20), x_q(16), x_q(7), x_q(23)\},$ $\{x_q(17), x_q(2), x_q(19), x_q(21), x_q(11), x_q(3), x_q(14)\},$ $\{x_q(28), x_q(5), x_q(4), x_q(9), x_q(13), x_q(22), x_q(6)\},$ or $\{x_q(12), x_q(27), x_q(10), x_q(8), x_q(18), x_q(26), x_q(15)\}$	Rotating-left factor (R_q) $\{r_q(1), r_q(2), r_q(3)\}$	Group address (G_q) $\{g_q(1), g_q(2), g_q(3), g_q(4), g_q(5)\}$
0	10000000	0	00000	10	0001111 ,0011110,0111100,1000111,1100011,1110001,1111000	0,1,2,6,5,4,3	01010
1	0000001 ,0000010,0000100,0001000,0010000,0100000,1000000	0,1,2,3,4,5,6	00001	11	0010111 ,0101110,0111001,1001011,1011100,1100101,1110010	0,1,3,6,2,5,4	01011
2	0000011 ,0000110,0001100,0011000,0110000,1000001,1100000	0,1,2,3,4,6,5	00010	12	0011011 ,0110011,0110110,1001101,1011001,1100110,1101100	0,4,1,6,3,5,2	01100
3	0000101 ,0001010,0010100,0100001,0101000,1000010,1010000	0,1,2,5,3,6,4	00011	13	0011101 ,0100111,0111010,1001110,1010011,1101001,1110100	0,5,1,6,4,3,2	01101
4	0001001 ,0010010,0010001,0100010,0100100,1000100,1001000	0,1,4,5,2,6,3	00100	14	0101011 ,0101101,0110101,1010101,1010110,1011010,1101010	0,2,4,6,1,3,2	01110
5	0001111 ,0001110,0011100,0111000,1000011,1100001,1110000	0,1,2,3,6,5,4	00101	15	0011111 ,0111110,1001111,1100111,1110011,1111001,1111100	0,1,6,5,4,3,2	01111
6	0001011 ,0010110,0101100,0110001,1000101,1011000,1100010	0,1,2,4,6,3,5	00110	16	0101111 ,0111101,1010111,1011110,1101011,1110101,1111010	0,2,6,1,5,4,3	10000
7	0001101 ,0011010,0100011,0110100,1000110,1010001,1101000	0,1,5,2,6,4,3	00111	17	0110111 ,0111011,1011011,1011101,1101101,1101110,1110110	0,3,6,2,5,1,4	10001
8	0010011 ,0100110,0011001,0110010,1001001,1001100,1100100	0,1,3,4,6,2,5	01000	18	0111111 ,1011111,1101111,1110111,1111011,1111101,1111110	0,6,5,4,3,2,1	10010
9	0010101 ,0100101,0101001,0101010,1001010,1010010,1010100	0,5,3,1,6,4,2	01001	19	1111111	0	10011

Note:

1. Binary value with Boldface font denotes the seed-value of the group

Design evaluation

Table 4.2 lists the comparison of performance of the proposed design with the existing designs [22][26]-[29]. The I/O channel of the proposed design is just a single input/output and independent of the transform length N. Based on Avant 0.35 μ m cell-library [43], we respectively show the comparison of area, cycle time, and area-delay product in Fig. 4.2, Fig. 4.3 and Fig. 4.4 to illustrate the advantages of the

proposed design. The results show that the normalized area cost of the proposed design is not always improved significantly if the length of partitioned cyclic convolution is not short enough. However, when we consider the cycle time effect together with the hardware cost, we find in Fig. 4.4 that the proposed design possesses better performance than the other designs [22][26]-[29] in term of reducing the normalized area-delay product from 52% to 91%. Table 4.3 shows the decomposition of 1-D DHT designs of different lengths in terms of short-length DHTs realized by the proposed design approach. For the DHT designs with lengths longer than 121, we use both the Agarwal-Cooley algorithm and prime-factor algorithm in decomposing them into short-length DHTs with cyclic convolution formulation. However, if the lengths of DHT are not long enough, like 49, 77, and 121 shown in Table 4.3, we need only to decompose the DHT to short ones by prime-factor algorithm (PFA), and realize them through GDA design approach directly. Due to the shortened DHTs with lengths that are short enough (i.e., 6-point cyclic convolution for 7-point DHT and 10-point cyclic convolution for 11-point DHT), the process of cyclic convolution partitioning can be omitted.

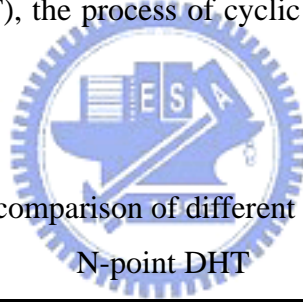


Table 4.2: The performance comparison of different designs for computing the 1-D N-point DHT

Designs	Adder (words)	MUL (words)	memory (words)	Barrel shifter (n ₂ -bit)	I/O No.	Cycle time
Liu [22]	5N-2	4N	0	0	(N+1)L	N*(T _{mul} + 2T _{add})
Kumar [26]	9N/4-6	N-4	0	0	(N+1)L	3N*(T _{mul} + 2T _{add})
Dhar [27]	6N-8	8(N-1)	0	0	(N+1)L	2N*(T _{mul} + 2T _{add})
Fang (DIT) [28]	$4^{\lceil \frac{(N+1)}{2} \rceil} + 1$	$4^{\lceil \frac{(N+1)}{2} \rceil}$	0	0	4NL	$[\lceil \frac{(N+1)}{2} \rceil + (N-1)]$ *(T _{mul} + 2T _{add})
Chang [29]	5N	8N	0	0	(N+2)L	2N*(T _{mul} + 2T _{add})
Proposed design	N+1	0	G _{num} (n ₂)*(N-1)	n ₁ *L _{ROM}	2L	n ₁ *L*(T _{rom} + T _{bar} + T _{csa} + T _{add})

Note:

1. A CORDIC processor is equivalent to four multipliers and two adders.
2. L denotes word-length of the input data.
3. G_{num} denotes the number of groups contained in the group memory modules. Usually, G_{num} is linearly related to N for small N values.
4. N-1 equals to n₁*n₂.
5. L_{ROM} denotes the word-length of memory.
6. T_{mul} denotes the delay time of a multiplier, T_{add} denotes the delay time of an adder, T_{rom} denotes the access time of memory, T_{bar} denotes the delay time of a Barrel shifter with n₂-bit width, and T_{csa} denotes the delay time of a carry save adder, where T_{csa} is equivalent to the delay time of n₁-1 adders.

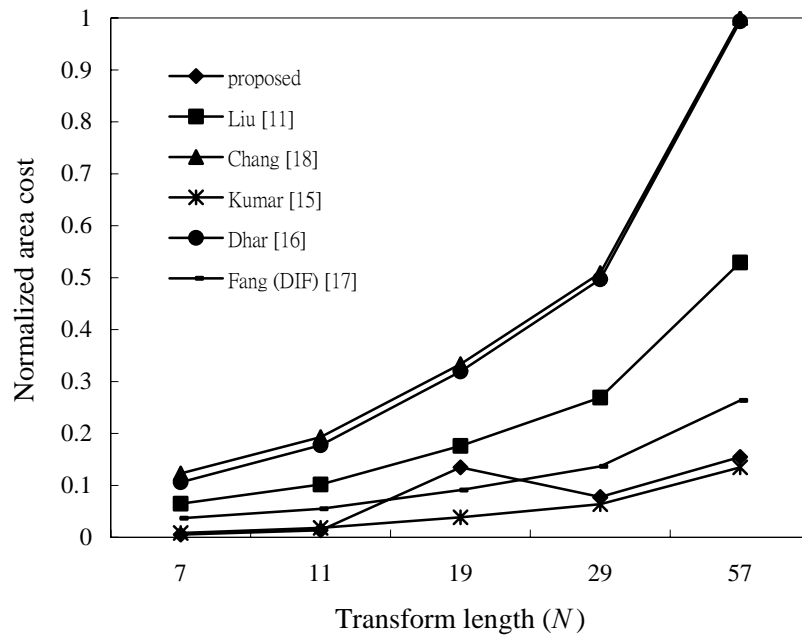


Fig. 4.2: Comparison of the normalized area cost in the realization of 1-D N-point DHT using the proposed design and the existing designs

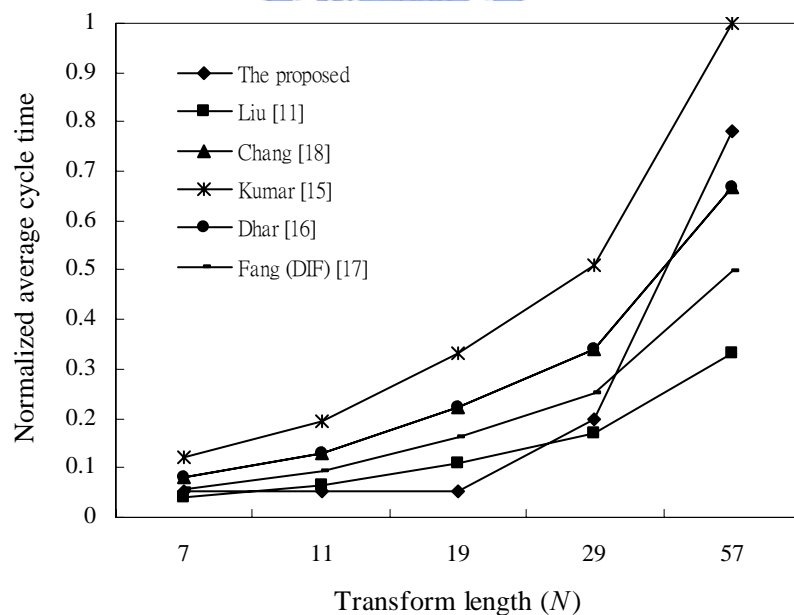


Fig. 4.3: Comparison of the normalized cycle time in the realization of 1-D N-point DHT using the proposed design and the existing designs

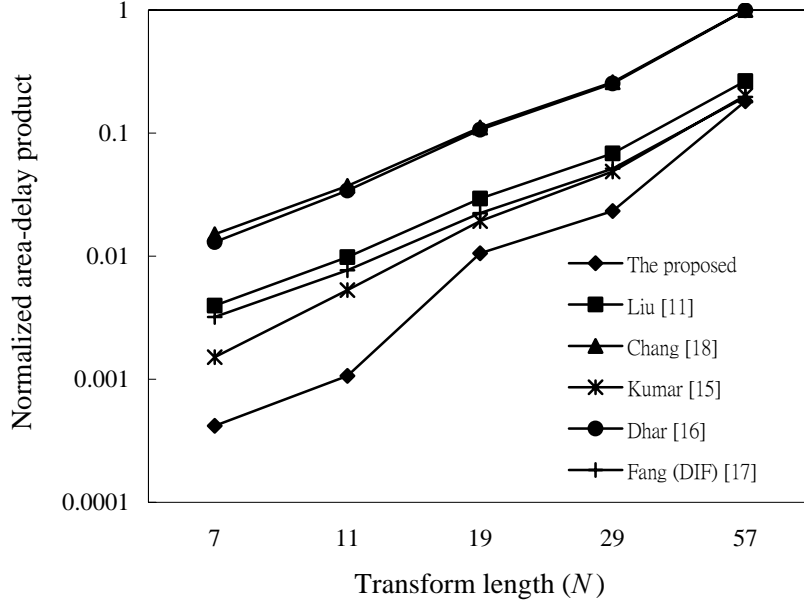


Fig. 4.4: Comparison of the normalized area-delay product in the realization of 1-D N-point DHT using the proposed design and the existing designs

Table 4.3: Length of 1-D DHT constructed by the decomposed short length DHTs

		Length of the decomposed DHT				
		7	11	19	29	57
Length of the decomposed DHT	7	49	77	133	203	399
	11	77	121	209	319	627
	19	133	209	361	551	1083
	29	203	319	551	841	1653
	57	399	627	1083	1653	3249

For the evaluation of long length DHT decomposed by the scheme mentioned above, we firstly evaluate the hardware cost and cycle time for the shortened DHT, and then estimate overall architecture of the long length DHT, which is composed of the shortened DHT, in terms of the hardware cost and cycle time parameters of the evaluated shortened DHT. Based on the result of high level synthesis for the 1-D 29-point and 57-point DHT designs shown in Table 4.4, we can further evaluate the DHT designs with the lengths of 841 (i.e., $29 * 29$), 1653 (i.e., $29 * 57$), and 3249 (i.e., $57 * 57$), respectively. Since the cycle times consumed by both the stage of 29-point

DHT and the stage of 57-point DHT in the 1653-point DHT design are not the same, we should take the largest one of them in these two stages. Comparing with the manner of directly partitioning in the conventional DA, we show the effectiveness of the proposed design in Fig. 4.5 in terms of the normalized area-delay products, where DP denotes directly partitioning for conventional DA and AC denotes partitioning with Agarwal-Cooley algorithm for GDA. In the DHT design examples of 841-point, 1653-point, and 3249-point, the proposed GDA approach combining with Agarwal-Cooley algorithm can efficiently remove the data redundancy to achieve 66.1% better in terms of area-delay product averagely.

Table 4.4: The evaluation result of GDA-based DHT designs

The length of DHT	Area cost ¹ (gates)	Cycle time ² (ns)
29	44890	702.7
57	89539	2749.4
841	89780	20378.3
1653	134429	79732.6
3249	179078	156715.8

Note:

1. The area cost of the DHT with composed length (i.e. 841-point, 1653-point, and 3249-point) does not include the transpose memory.
2. The cycle time denotes the time consumed by the computation of N DHT outputs.

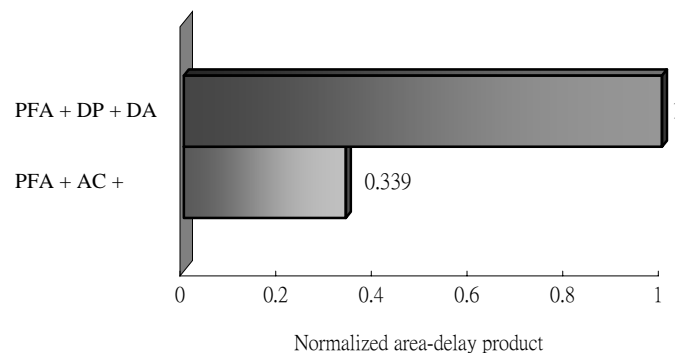


Fig. 4.5: Average improvement of the normalized area-delay product in the designs of 841-point DHT, 1653-point DHT, and 3249-point DHT using the proposed design approach

4.3 Variable-length DFT Design to Communication System Application

4.3.1 Overview of Communication system

The orthogonal frequency division multiplexing (OFDM) technique has been widely adopted in high-speed data transmission, such as asymmetry digital subscriber lines (ADSL), very high speed digital subscriber lines (VDSL), and digital audio/video broadcasting (DAB/DVB) systems. In these systems, the discrete Fourier transform (DFT) plays a main role. Table 4.5 shows the lengths of DFT required in these systems, where the required length of DFT is proportional to the data-rate as well as the distance. Thus a configurable dedicated hardware for the DFT computation with variable length would be desired in the various high data-rate communication applications.

There are many high-speed applications [16][17][19][21][62][63] that address the use of dedicated hardware designs for computing the long length DFT/IDFT. The designs with fast algorithms are attractive for low computational complexity. However, hardware design of the algorithm is communication intensive and computation intensive to complicate the realizations of controller and arithmetic operation. In addition, most of the designs with fast algorithms exploit a butterfly datapath and a global memory in storing all of input/output data as well as the intermediate results. The mass data access from the global memory wastes a large percentage of power in this kind of designs. Besides, the cascaded structure in the fast algorithm makes the designs have poor numerical accuracy such that longer data wordlength in the datapath is needed. This fact will reduce the low complexity advantages of the fast algorithm and thus increase the hardware cost of the designs with fast algorithm. Thus, the efficient hardware design of DFT is still a challenging problem due to its high computational complexity and the requirement of real-time processing. The popular designs based on the distributed arithmetic (DA) have the benefit to exploit both constant and bit-level computation. However, the traditional DA technique suffers from large memory cost for long length designs. To solve this problem, we have proposed the GDA design approach that further reduces the

memory cost efficiently with the numerical property. In this part of research, we intend to extend the GDA design approach to long- and any-length design, and its application to the popular power-of-two variable-length DFT.

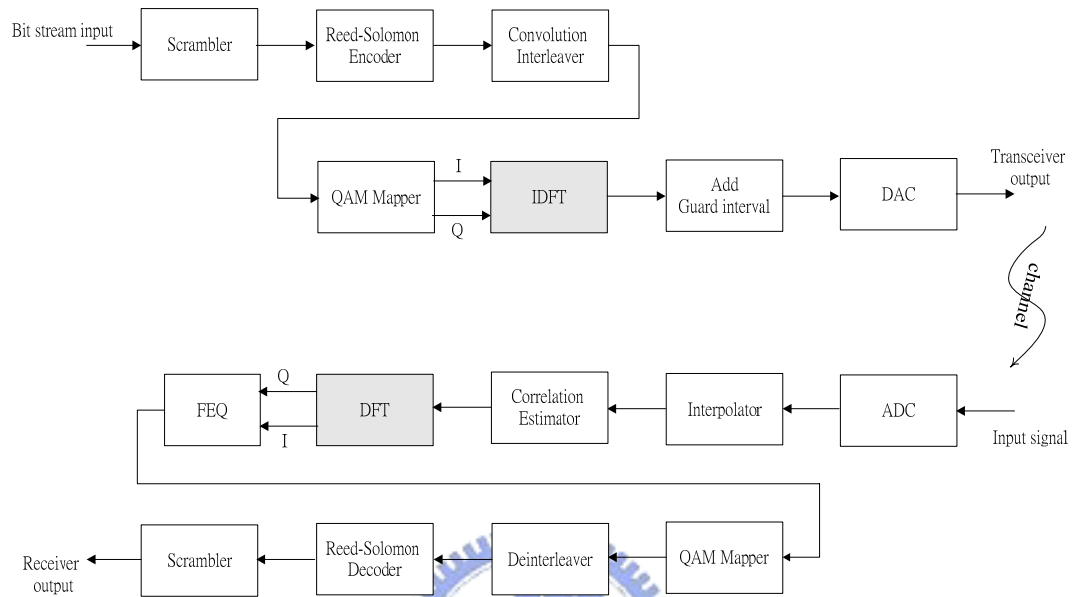


Fig. 4.6: Transceiver /Receiver architecture in the communication system

Table 4.5: DFT lengths for several communication systems

Communication system	DFT length	application
IEEE 802.11a	64	Wireless Ethernet
HIPERLAN/2	64	Wireless ATM
ADSL	256/512	Internet access
VDSL	512/1024/2048/4096/8192	Internet access
DAB	256/512/1024/2048	Digital Audio Broadcasting
DVB-T	8192/2048	Digital Video Broadcasting

4.3.2 Hardware Cost Analysis

Before designing the DFT architecture with GDA approach, we intend to analyze the complexities of the FFT algorithm and the proposed GDA algorithm first. With the Cooley-Tukey algorithm, the computation complexity of FFT algorithm is

around $N \log_2^N$ [64], including $1/2 N \log_2^N$ computations of complex-multiplication and $N \log_2^N$ computations of complex-addition, that is equivalent to $2 N \log_2^N$ computations of real-multiplication and $2 N \log_2^N$ computations of real-addition. Based on Cooley-Tukey algorithm, there are two popular architecture designs. One is single processing element (PE) design which provides adequate performance with low hardware cost. The other is pipeline based design for the application with high throughput. Because of regularity, modularity, locality, and high throughput with moderate hardware cost, one dimensional linear array is more popular [65]. Besides, in order to compute DFT via FFT, the input data and the intermediate results need to be buffered and reordered by using some memory buffers, where the size of the memory buffers is around $N(N-1)$ words. There are two existing buffering strategies proposed for the pipeline FFT architecture [66]. One is delay-commutator (DC) architecture. The other is delay-feedback (DF) architecture.

Table 4.6: The computation complexity of various DFT algorithms

Algorithm	Complexity
DFT (Mul-Add)	N^2
Cooley-Tukey DFT (Mul-Add)	$2N^{\frac{3}{2}}$
DA-based decomposed Cooley-Tukey DFT with cyclic partitioning (memory-word)	$N^{\frac{5}{4}} \cdot 2^{\sqrt[3]{N}+1}$
GDA-based decomposed Cooley-Tukey DFT with cyclic partitioning (memory -word)	$N \cdot 2^{\sqrt[4]{N}+1}$

Based on Cooley-Tukey algorithm for DFT decomposition, cyclic convolution and pseudocirculant matrix factorization algorithm for cyclic convolution partitioning, and GDA design approach, Table 4.6 shows the derivation of computation complexity of the proposed long length DFT algorithm from the original DFT algorithm. The

complexity of original DFT is N^2 . With Cooley-Tukey decomposition, the complexity is reduced to $2N^{\frac{3}{2}}$ (i.e., $2\sqrt{N}(\sqrt{N})^2$). And then combining with the pseudocirculant matrix factorization algorithm, we can realize the long length DFT with conventional DA, and the complexity can be changed into $N^{\frac{5}{4}}2^{\sqrt[4]{N}+1}$ (i.e., $2\sqrt{N} \cdot [(2^{\sqrt[4]{N}} \cdot \sqrt[4]{N}) \cdot (\sqrt[4]{N} \cdot \sqrt[4]{N})]$). If we replace the conventional DA with the proposed GDA in the DFT design, the complexity can be reduced to $N2^{\sqrt[4]{N}+1}$ (i.e., $2\sqrt{N} \cdot [(\frac{2^{\sqrt[4]{N}}}{\sqrt[4]{N}} \cdot \sqrt[4]{N}) \cdot (\sqrt[4]{N} \cdot \sqrt[4]{N})]$). Thus it is possible that the hardware cost of DFT with the proposed DFT algorithm is smaller than the existing FFT algorithms. For example of 4096-point DFT, according the Table 4.7, the estimated hardware costs of FFT and proposed GDA-based DFT are shown as Fig. 4.7. We can see that the hardware cost of the proposed design is better than FFT when the length of DFT is smaller than 4096, where the multiplier is four times the hardware cost of adder, and the transistor count of memory is proportional to memory word-length. However, actually due to some of the multiplications in FFT butterfly can be omitted, the hardware complexity in Table 4.7 should be changed into Table 4.8. As for the estimations of the delay time consumed by each sample, shown as Table 4.9, they are respectively sum of the delay time of multiplier, adder, and memory access in FFT, and L_2 times the sum of the delay time of memory access, barrel-shifter, and accumulator divided by the length of cyclic convolution in GDA-based DFT, where L_2 denotes the maximal one of the word-length of parallel-in-serial-out module in the two stages of GDA-based DFT design. As shown in Fig. 4.8, the area-delay product of GDA-based DFT is smaller than that of FFT when the transform length is smaller than 256, where sum of memory access time and barrel-shifter delay time is around half delay time of the adder for the partitioned small size memory in GDAU, and the delay time of multiplier in FFT is assumed as four times delay time of the adder. Thus the GDA-DFT takes around 0.32 time delay time of FFT for each sample, where L_2 and $N^{1/4}$ equal 12 and 8 respectively in our design).

Table 4.7: The estimation of hardware costs of the FFT and the proposed GDA-DFT

Algorithm	Hardware cost
FFT	$0.5N\log_2^N A_{mul} + N\log_2^N A_{add}$
GDA-based DFT	$2 \cdot N \cdot 2^{\sqrt[4]{N}+1} A_{ROM-word}$

Note:

1. Assume overall GDAU is two times hardware cost of ROM while $N^{1/4}$ equals 8.
2. A_{mul} denotes the hardware cost of multiplier in unit of equivalent gate count.
3. A_{add} denotes the hardware cost of adder in unit of equivalent gate count.
4. $A_{ROM-word}$ denotes the hardware cost of the word of ROM in unit of equivalent gate count.



Table 4.8: The estimation of hardware costs of the FFT with actual complexity and the proposed GDA-DFT

Algorithm	Hardware cost (gates)
FFT (radix-2 SDF)	$3(\log_2^N - 2)A_{mul} + 6(\log_2^N - 2)A_{add} + 2(N-1)A_{mem} + 9(\log_2^N - 2)A_{mux}$
GDA-based DFT	$2 \cdot N \cdot 2^{\sqrt[4]{N}+1} A_{ROM-word}$

Table 4.9: The estimation of cycle times of the FFT and the proposed GDA-DFT for each sample

Algorithm	Delay time
FFT	$t_{add} + 2 t_{mux} + t_{mul}$
GDA-based DFT	$L_2 (t_{acc} + t_{br} + t_{add})/N^{1/4}$

Note:

1. t_{mul} denotes delay time of the multiplier.
2. t_{add} denotes delay time of the adder.
3. t_{acc} denotes access time of the memory used in GDA-based DFT design.
4. t_{br} denotes delay time of the barrel-shifter used in GDA-based DFT design.
5. L_2 denotes the word-length of input data.

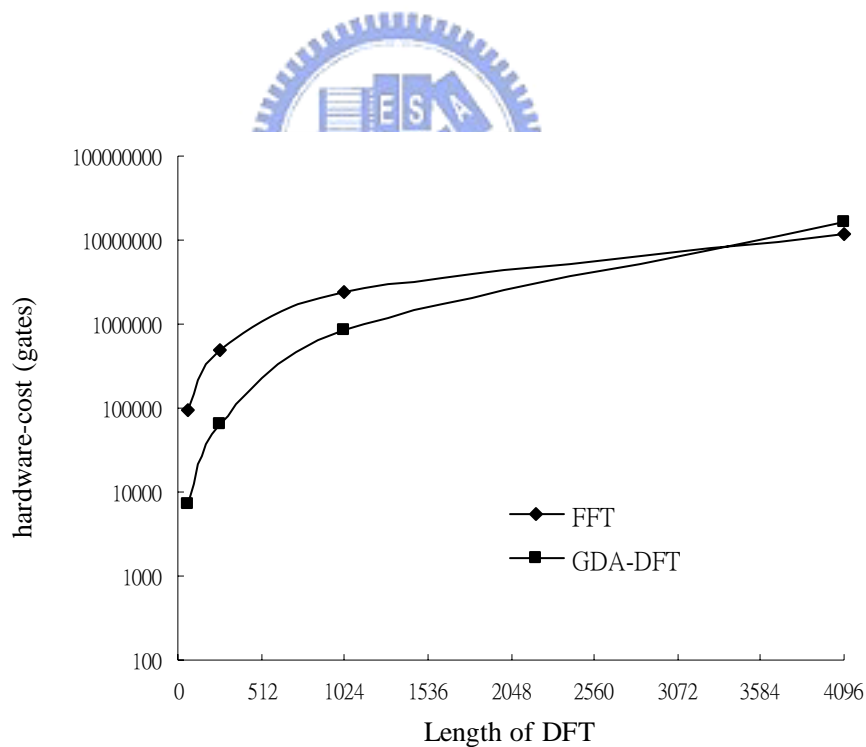


Fig. 4.7: Hardware cost of the original FFT versus the proposed GDA-based DFT

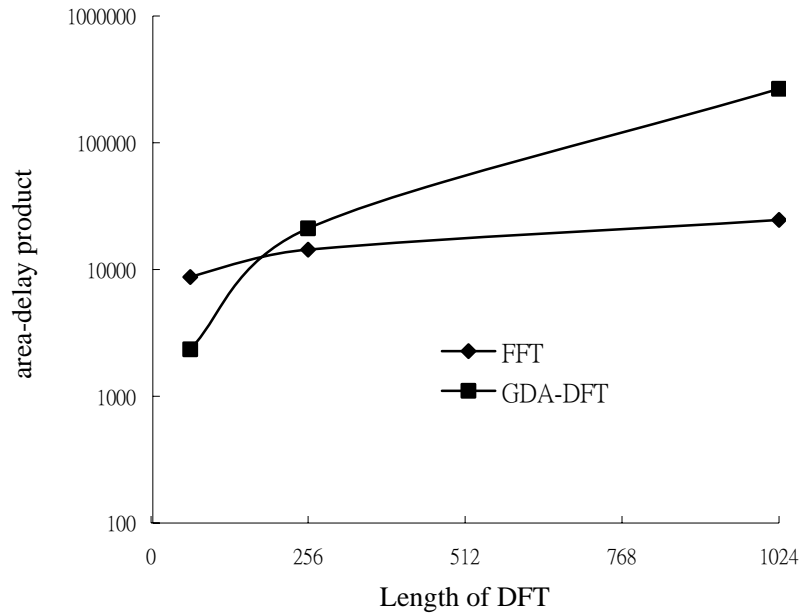


Fig. 4.8: Delay-area product of the FFT versus the proposed GDA-based DFT



4.3.3 GDA-based Variable Length DFT Design and Evaluation

Exploiting the Cooley-turkey decomposition algorithm, we first decompose the long length 1-D DFT into 2-D short length DFT, and form the shortened DFTs in each dimension in cyclic convolution. Then, with the pseudocirculant factorization algorithm, we factorize the cyclic convolutions as the sum of the shortened cyclic convolutions, and apply the proposed GDA design to realize of the short-length cyclic convolutions for achieving a hardware efficient long-length DFT design. Table 4.10 shows the proposed design can flexibly be used to compute the 1-D 64/128/256/512/1024/2048/4096-point DFT by cascading the decomposed short length DFT.

Table 4.10: Length of 1-D DFT constructed by the decomposed short length DFTs

		Length of the decomposed DFT			
		8	16	32	64
Length of the decomposed DFT	8	64	128	256	512
	16	128	256	512	1024
	32	256	512	1024	2048
	64	512	1024	2048	4096

Architecture design

Fig. 4.9 shows the block diagram of the proposed GDA-based DFT architecture with variable length with the Cooley-Turkey decomposition. This architecture consists of two configurable GDA units for respectively computing the row and column 1-D 8/16/32/64-point DFT, a multiplier for performing the twiddle factor multiplications serially, and a transpose memory for data transposition. Fig. 4.10 shows the block diagram more detail with real input data and complex output data. For efficiently realizing the twiddle factor multiplications, the complex number multiplier with serial manner, such as CORDIC processor or the serial multiplier set, can be a proper choice combined with DA-based design. In cyclic convolution formulation, the architecture in Fig. 4.10 can be redrawn as Fig. 4.11. It is composed of serial multiplication for preprocessing, GDA computation for $T_{ij}()$, and serial multiplication for post-processing. Each the $T_{ij}()$ block can be configured for the 1-D DFT computation with different length, where i, j denote the computation with real part of input data and real part of DFT coefficient (i.e., RR), imaginary part of input data and imaginary part of DFT coefficient (i.e., II), real part of input data and imaginary part of DFT coefficient (i.e., RI), or imaginary part of input data and real part of DFT coefficient (i.e., IR). In Fig. 4.11, we can see that the output data of $T_{ij}()$ is sequentially multiplied by the post-processing coefficient of row 1-D DFT, the twiddle factor, and preprocessing coefficient of column 1-D DFT. Thus we can combine the three multiplications, and replace with one multiplication only. According to the tradeoff between word-length of the transpose memory and word-length of the multiplier, as shown in Fig. 4.12 and Fig. 4.13, this multiplication can selectively be located in front or rear of the transpose memory.

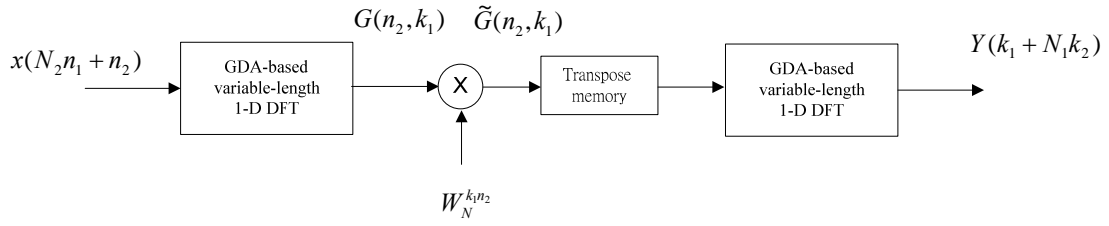


Fig. 4.9: Block diagram of the proposed variable-length DFT architecture.

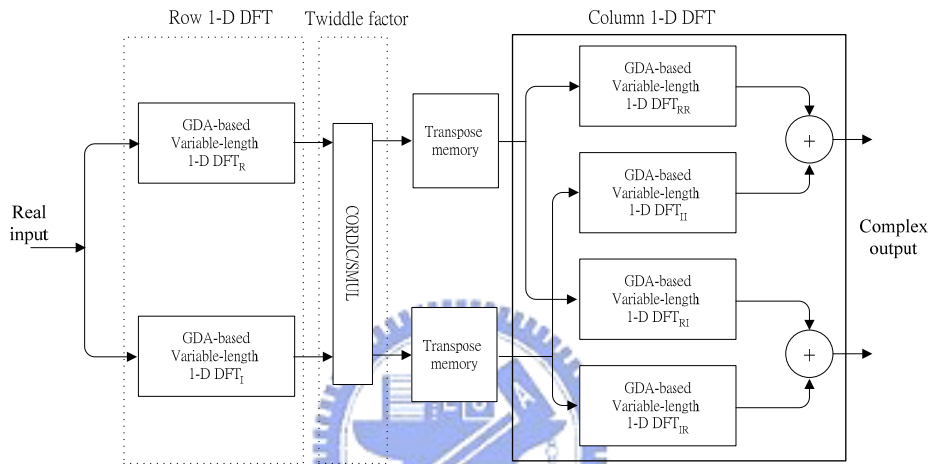


Fig. 4.10: Architecture of 2-D DFT with real input.

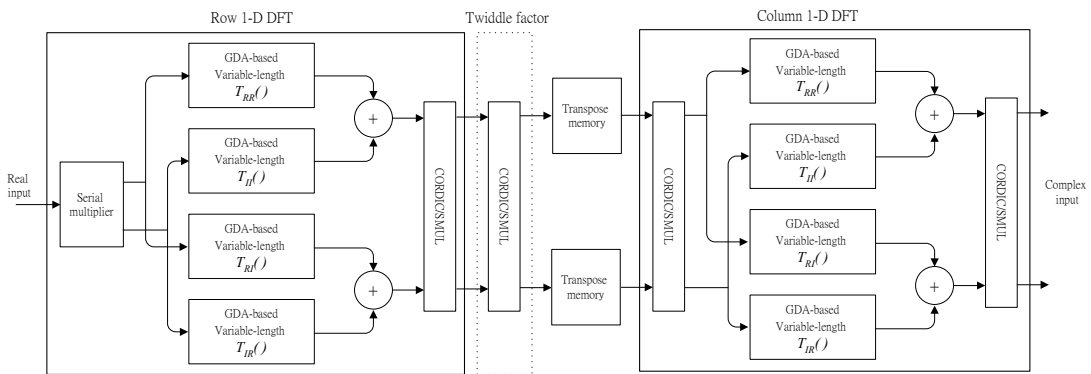


Fig. 4.11: Architecture design of the 2-D DFT in cyclic convolution formulation.

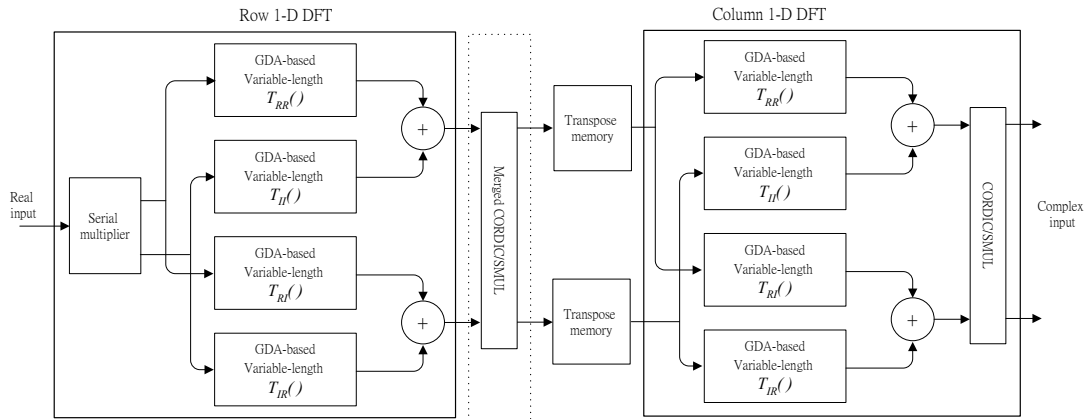


Fig. 4.12: Version 1 of the reduced architecture of 2-D DFT in cyclic convolution formulation.

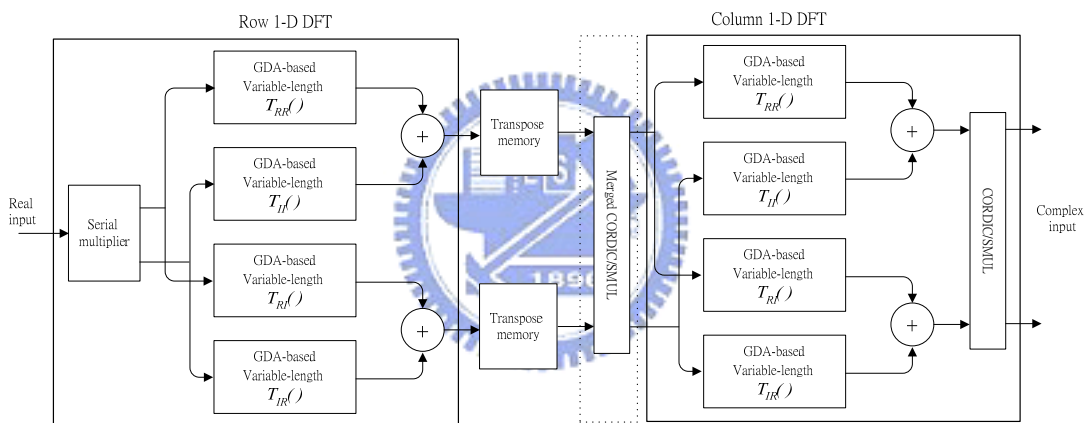


Fig. 4.13: Version 2 of the reduced architecture of 2-D DFT in cyclic convolution formulation.

For the purpose of performing the variable-length DFT computation with identical hardware, we adopt the pseudocirculant matrix factorization algorithm to factorize the cyclic convolution $T_{ij}(\cdot)$ in 1-D DFT with different length as the composition of 8-point cyclic convolutions. For the case of 64-point cyclic convolution, as shown in Fig. 4.14, the matrix of input data can be decomposed as an eight by eight blocked matrix. Since each block in the matrix has preserved as an 8-point cyclic convolution, we can allocate the computation of every eight row blocks into eight 8-point GDAU and sum up the outputs of GDAUs to have eight outputs of

the 64-point cyclic convolution. Observing the matrix form in left side of the Fig. 4.14, we can see that each computation of eight row blocks with rotated order can be folded onto the identical eight 8-point GDAUs. Totally, eight iterations are needed to compute all the outputs of 64-point cyclic convolution. For the case of 32-point cyclic convolution, due to it is composed of four by four blocked matrix with pseudocirculant, as shown in Fig. 4.15, we can compute every eight outputs of the 32-point cyclic convolution by summing up the results of four 8-point cyclic convolution. With the same amount of GDA computation hardware resource, it needs two iterations to compute all the 32 outputs of 32-point cyclic convolution. With the same way, the case of 16-point cyclic convolution can also be composed of two by two blocked matrix with pseudocirculant. In the proposed design, we have constructed the hardware with eight 8-point cyclic convolution modules for the computation of cyclic convolution in the variable-length DFT. This hardware can compute the 64 outputs of 64-point cyclic convolution by eight iterations, the 32 outputs of 32-point cyclic convolution by two iterations, the 16 outputs of 16-point cyclic convolution by one-second iteration, and the 8 outputs of 8-point cyclic convolution by one-eighth iteration. Thus for the computation of 64/256/1024/4096-point 1-D DFT, the lengths of row DFT and column DFT are respectively 8/16/32/64, and the number of iterations with the identical hardware is 1/8/64/512.

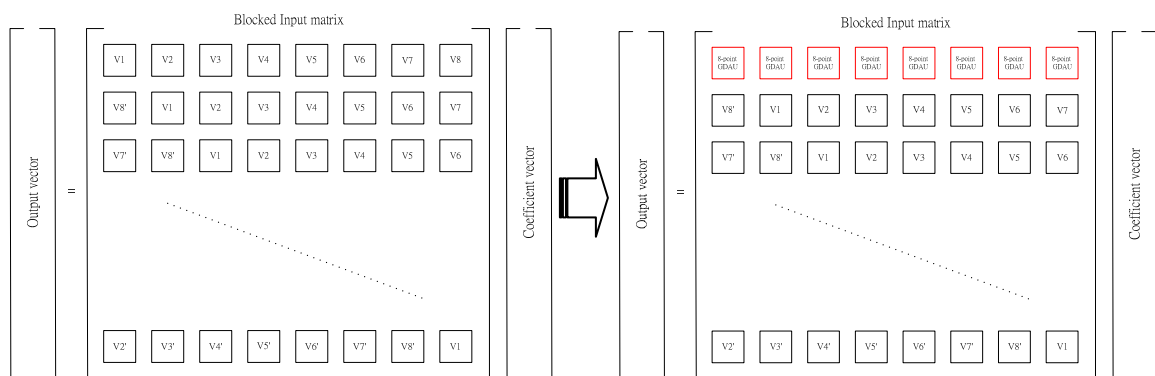


Fig. 4.14: Folding of the computation of each eight row blocks in 64-point cyclic convolution.

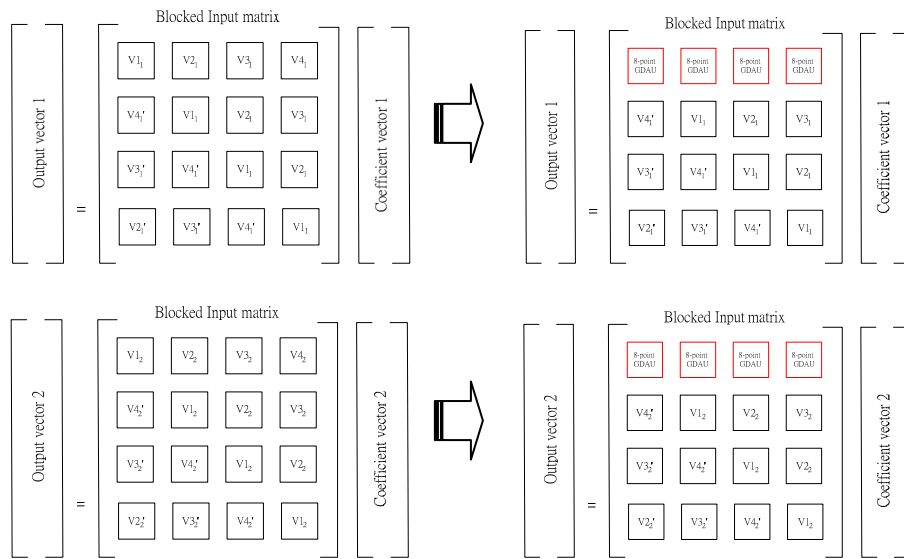
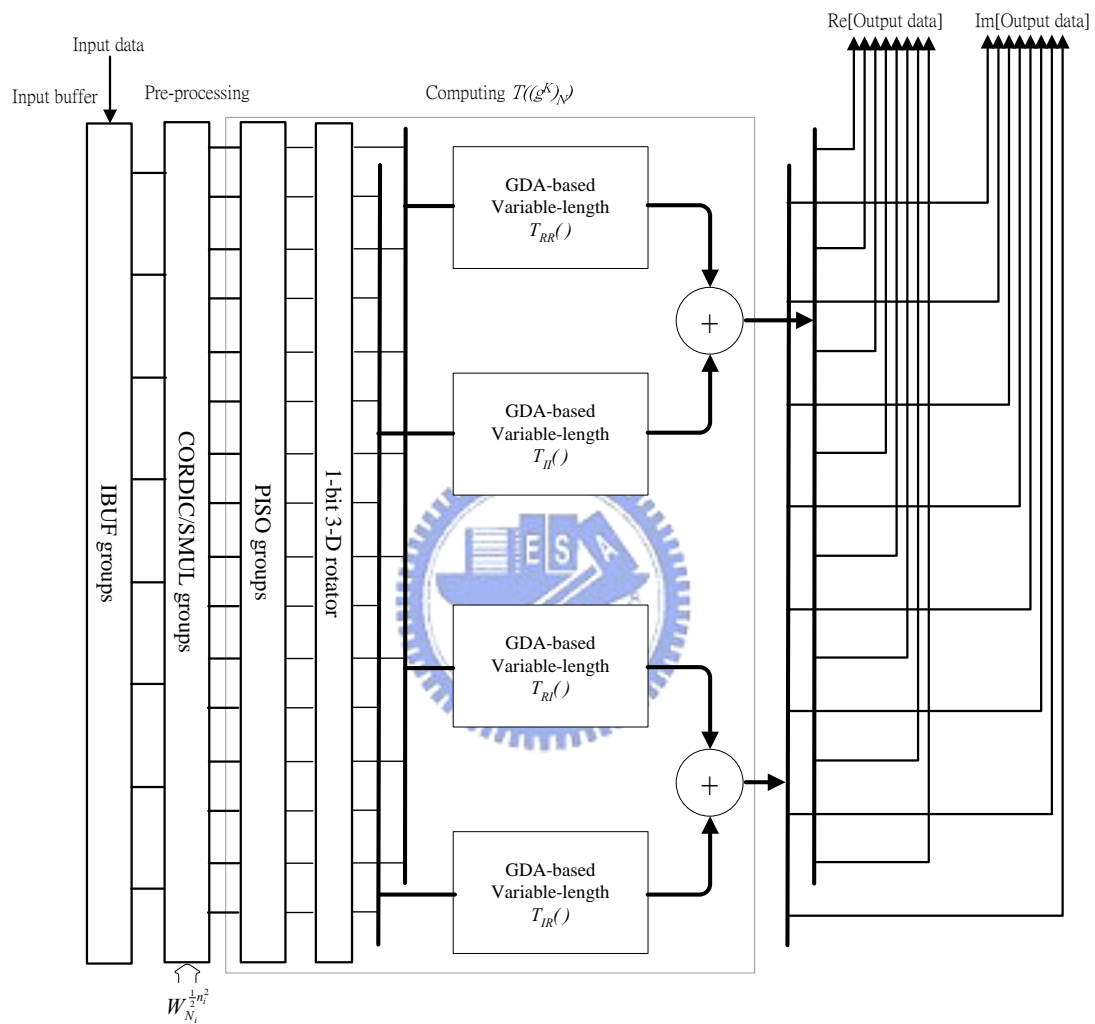


Fig. 4.15: Folding of the computation of each four row blocks in 32-point cyclic convolution.

With the identical hardware, due to the numbers of iterations for the computations of DFT with different lengths are not the same, the variable-length DFT design must be worked with different control states. Since the hardware resource in the proposed design can compute eight 8-point 1-D DFTs in each iteration, the 64-point 1-D DFT needs only one iteration to compute all the output data in row and column DFT. For the computation of 256-point 1-D DFT, each of the iterations can be used for the computation of two 16-point DFTs in each dimension so that 16 16-point DFT computations need totally eight iterations, as well as 64 iterations needed for 1024-point 1-D DFT and 512 iterations needed for 4096-point 1-D DFT. Due to the coefficients of 8, 16, 32, and 64-point DFT are different, we use RAM instead of ROM for replacing the contents of memory needed for computing the variable-length DFT. The partial products stored in this memory for DA computation can be downloaded in the initialization phase from the main frame. Since there are thirty-six memory entries in the 8-point GDAUs, thirty-six write cycles are consumed in each of the initial phases. Due to the data rate and the length of DFT in a communication system is fixed while the condition of environment is remained, once for loading coefficients of the DFT with decided length into the memory of variable-length DFT core is required. However, if the length of DFT is decided larger than 64, there are

required respectively 4, 16, and 64 initial phases for 256-, 1024-, and 4096-point DFT. All the coefficients of DFTs with different lengths can be stored previously in the low cost memory of main frame.



(a)

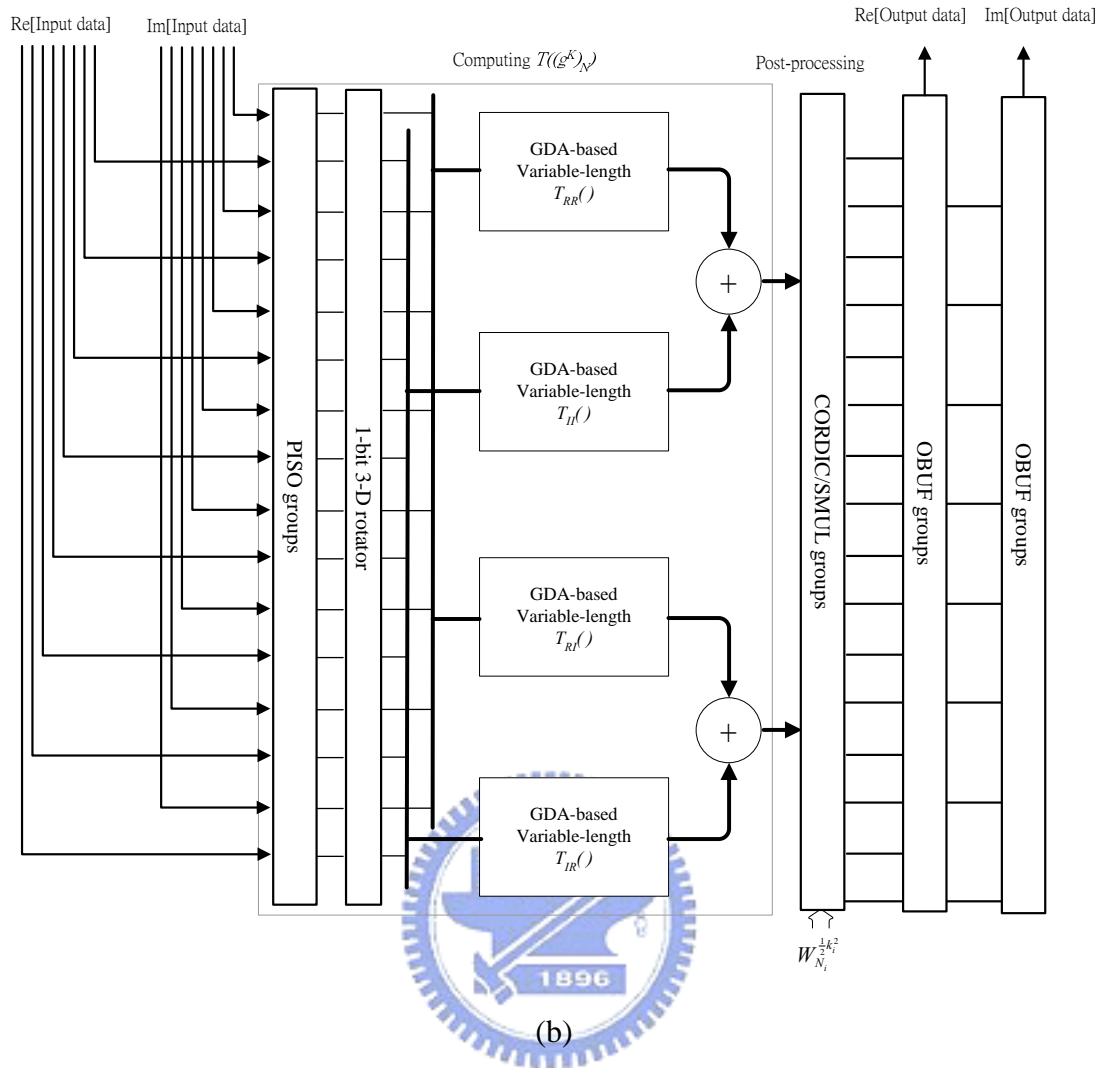


Fig. 4.16: Detail architecture of (a) the row 1-D DFT with input buffer and (b) the column 1-D DFT with output buffer.

Fig. 4.16 shows the proposed variable-length DFT design more detail in row stage and column stage, including input buffer (IBUF), serial multiplier (SMUL), parallel-in-serial-out (PISO), 1-bit three-dimension (3-D) rotator, variable-length GDA-based module, and output buffer (OBUF). The length of DFT in each stage can be configured with 8/16/32/64-point. In the following, we will illustrate detail design of the modules in the proposed variable-length DFT.

Similar to most of the DA-based designs, Fig. 4.17 (a), (b), and (c) show the input buffer for serially storing input data, the parallel-in serial-out (PISO) module for issuing the input data of DA with word-parallel-bit-serial manner, and the output buffer for serially outputting the output data.

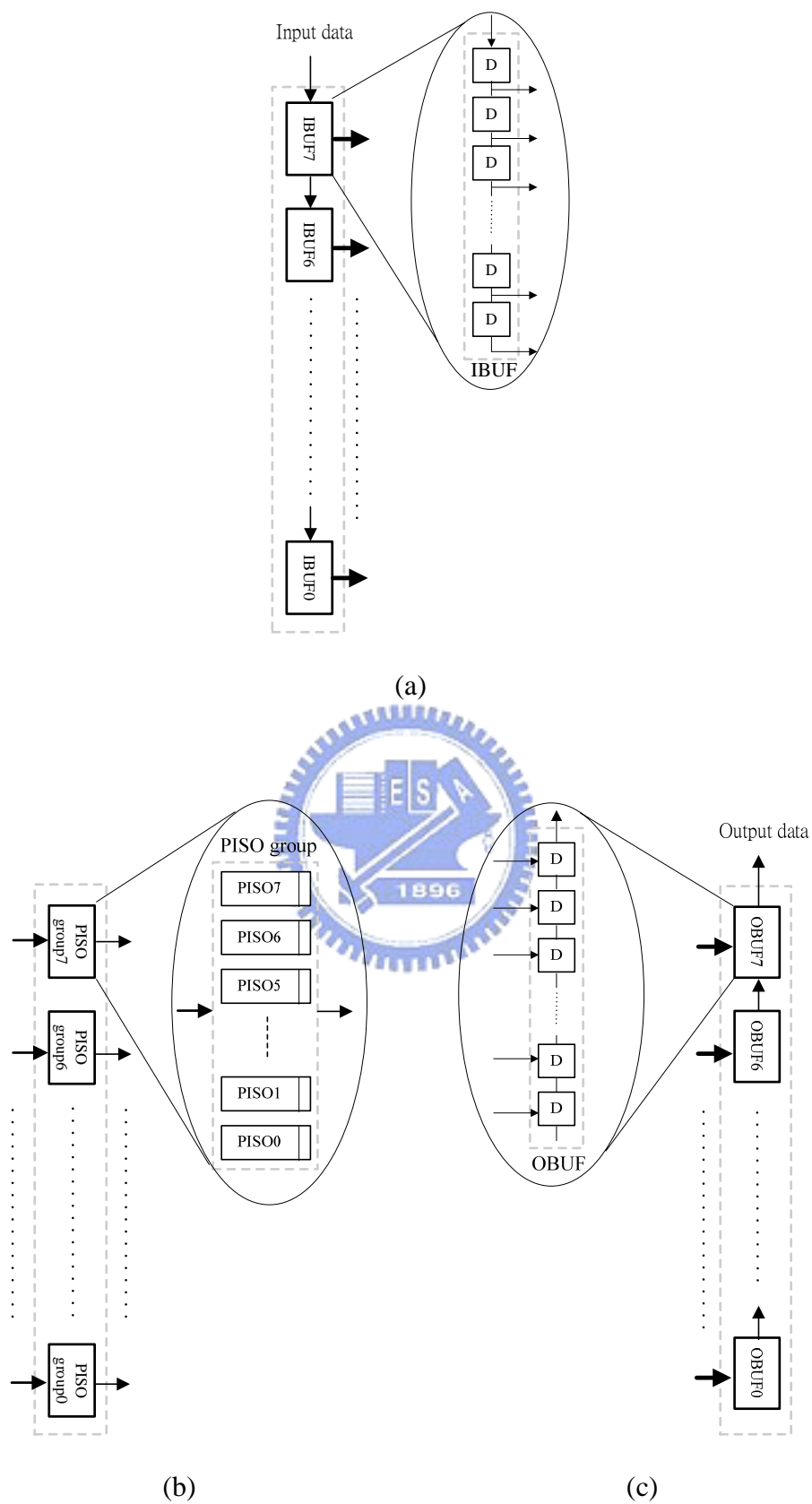


Fig. 4.17: Detail design of (a) input buffer groups, (b) PISO groups, and (c) output buffer groups in the proposed 1-D DFT architecture.

On the consideration of input data permutation for GDAUs, according to the formulation of any-length cyclic convolution in (2.6), the input data of the eight 8-point GDAUs in each of the iterations is block rotated and in-block rotated. Then a 1-bit rotator is needed for preparing the exact data on the inputs of GDAUs. Since the rotator needs to work with different lengths for variable-length DFT, a specific 1-bit 3-D barrel rotator is designed as Fig. 4.18 (a). The mode of 1-bit 3-D rotator can be decided by three variables for how many bits are rotated in a block, how many blocks are rotated in cyclic convolution for the chosen length of DFT, and which length of DFT is chosen. It performs the in-block rotation with 8-bit barrel rotator (BR) in stage 1. For the block rotation, in the stage 2, the barrel rotator group (BRG) with eight 2-bit barrel rotators is used in 16-point DFT in each dimension of the 256-point DFT. In the stage 3, the barrel rotator group (BRG) with eight 4-bit barrel rotators is used in 32-point DFT in each dimension of the 1024-point DFT. In the stage 4, the barrel rotator group (BRG) with eight 8-bit barrel rotators is used in 64-point DFT in each dimension of the 4096-point DFT. Table 4.11 shows the condition of BR in each stage for DFT with the lengths of 64, 256, 1024, and 4096. This specific 1-bit 3-D rotator design provides to permute the exact data on the inputs of GDAUs for computation of the proposed variable-length DFT design.

Table 4.11: Condition of BR in each stage for DFT with the lengths of 64, 256, 1024, and 4096.

length of DFT	stage 1	stage 2	stage 3	stage 4
64	P	P	P	P
256	R	R	P	P
1024	R	P	R	P
4096	R	P	P	R

Note:

1. *D* denotes the BR works on bypass mode.
2. *R* denotes the BR works on rotation mode.

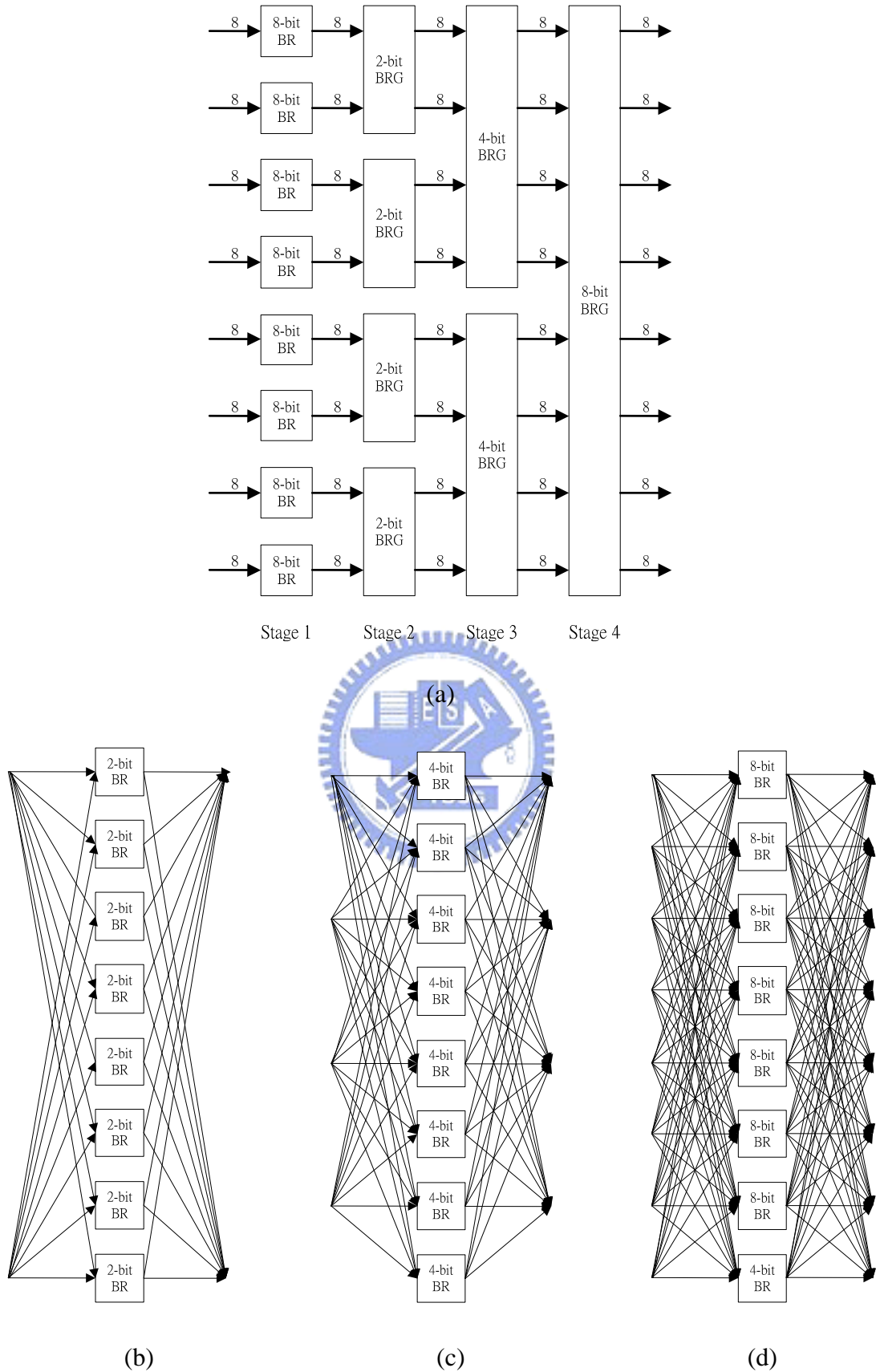


Fig. 4.18: (a) design of the 1-bit 3-D rotator and the routing for (b) 2-bit BRG in stage 2, (c) 4-bit BRG in stage 3, and (d) 8-bit BRG in stage 4.

Following the 1-bit 3-D barrel-rotator, with identical hardware, the module with GDAUs is used to compute all the output data or part of the output data in each of the iterations for DFT with variable length. As shown in Fig. 4.19, each of the GDAUs performs the computation of 8-point cyclic convolution. In the following stage, shown in Fig. 4.20, an adder-group tree is used to sum up the partial outputs from these GDAUs for the shortened cyclic convolutions in case of the length of row or column DFT is larger than eight, where the different dash lines respectively denote the data-flows in the row or column DFT with different lengths. In each of the iterations for DFT computation, the numbers of output data computed by the identical computation resource for the 1-D DFT with lengths of 64/256/1024/4096 are 64/32/16/8. With the limitation of the number of GDAUs, we place the multiplexers with different width to select out the different number of output data for the 1-D DFT with different length.

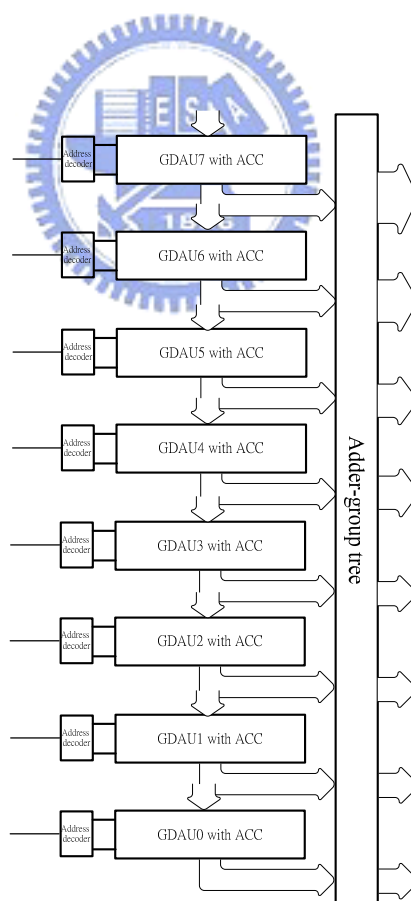


Fig. 4.19: Detail design of variable-length GDA-based module used for the computation of $T_{ij}()$ in the proposed 1-D DFT architecture.

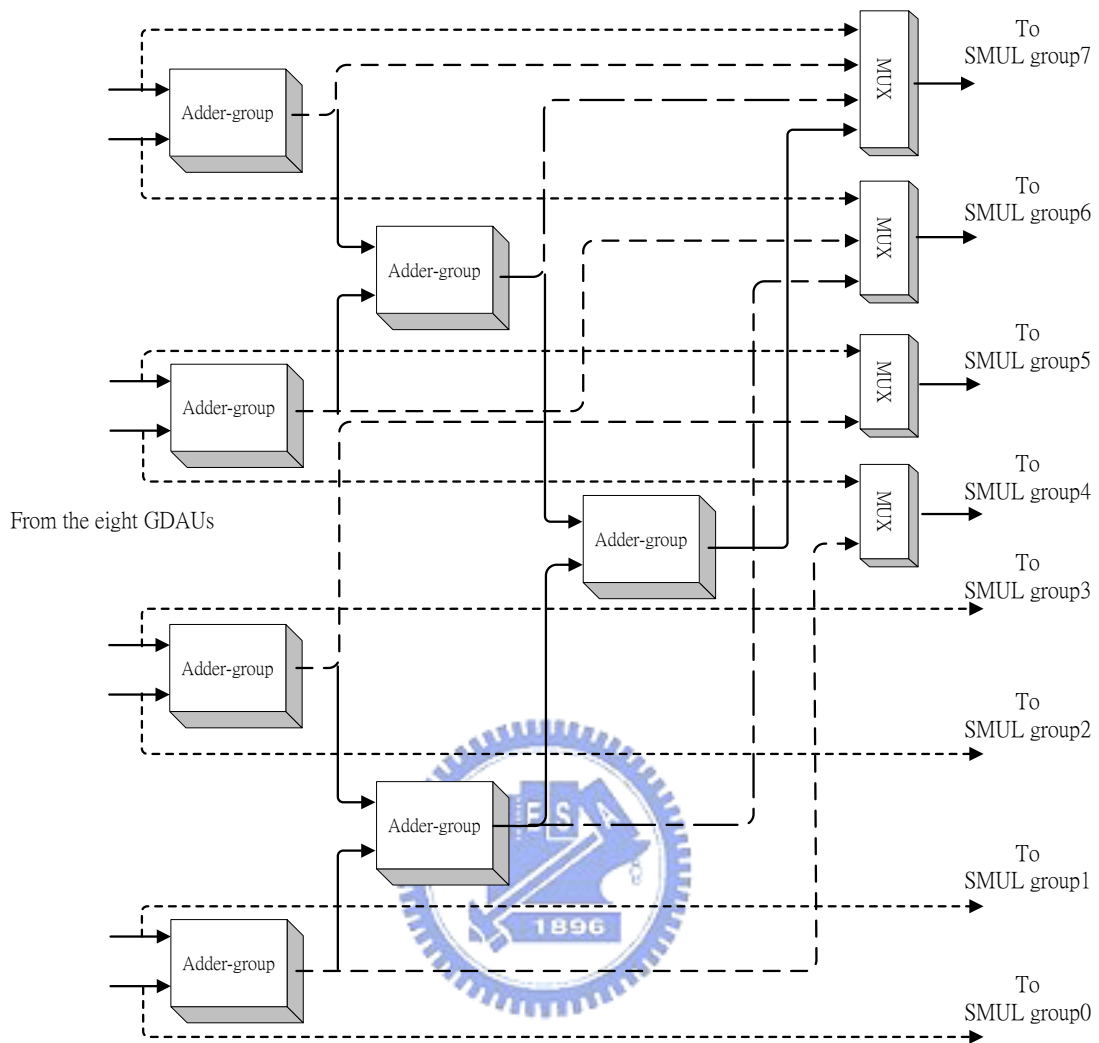


Fig. 4.20: Data-flow of the adder-group tree follows the GDAUs in the proposed variable-length DFT design.

As the formulation mentioned in the chapter 3, the multiplications need for pre- and post- processing of the 1-D DFT in cyclic convolution. For reducing hardware cost of the multiplications, we combine the multiplication of pre-processing in row DFT and the multiplication of post-processing in column DFT with the multiplication of twiddle-factor processing such that only one multiplier is remained between row DFT and column DFT. With the feature of serial manner in DA computation, the complex multiplier with serial manner should be a proper choice for the multiplication.

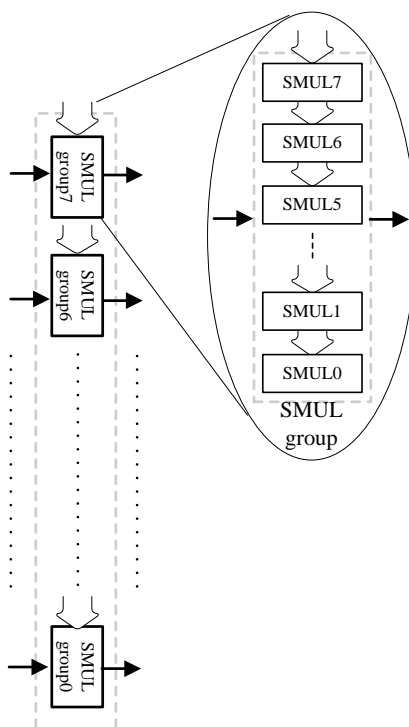


Fig. 4.21: Detail design of serial multiplier groups in the proposed 1-D DFT architecture.



Since the output data is out of order in the row DFT, shown as Fig. 4.22, for the usage of column DFT, we can reorder these data while writing them into the transpose memory by using a specific address generator.

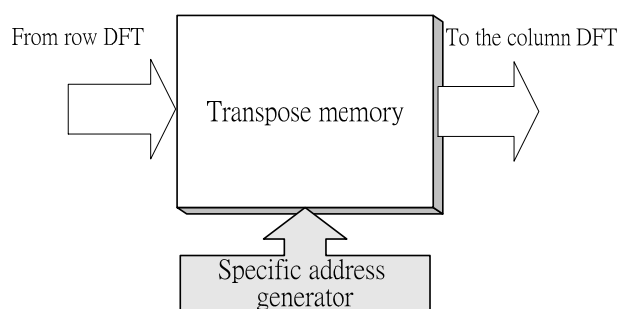


Fig. 4.22: The transpose memory with the specific address generator

Design evaluation

Based on the proposed GDA-based 1-D variable length DFT architecture as Fig. 4.16, the number of cycle consumed for computing the 64/256/1024/4096-point DFT with the 8/16/32/64-point DFT in two dimensions is proportional to $O(8^{\log_2 \sqrt{N}-3} \times L)$, where N denotes the length of 1-D DFT, and L denotes the word-length of GDA input data. Referring to the simulation results of the DFT with lengths of 8, 16, 32, and 64, we can further evaluate the DFT designs with the lengths of 128 (i.e., $8 * 16$), 512 (i.e., $16 * 32$), and 2048 (i.e., $32 * 64$), respectively. However, since the cycle count consumed in two stages of 8-point DFT and 16-point DFT in the 128-point DFT design as well as in the 512- and 2048-point DFT designs, are not the same, we must take the largest one of the two stages.

We have evaluated the proposed design with UMC 0.18um cell-library. For fairly compared with the existing long-length and variable-length FFT designs [67]-[71], we eliminate the factor of different technology by normalizing all the design areas with the normalized index [72] as (4.36). As the simulation result, except for the advantages of short latency and high hardware utilization efficiency in the GDA-based design, checked with the hardware cost analysis mentioned above, Table 4.12 also reveals that the power of two variable-length DFT realized with the proposed decomposition approach and GDA design can achieve competitive hardware cost under the same throughput rate, especially the length of DFT is ranged between 64 and 512. Thus the proposed variable-length DFT can be a more efficient dedicated design to the application of ADSL system.

$$\text{Normalized Area} = \frac{\text{Area}}{(\text{Technology} / 0.18\mu\text{m})^2} \quad (4.36)$$

Table 4.12: Comparison of the existing FFT designs and our DFT design

	Bidgt [67]	Jia [68]	Kuo [69]	Pao [70]	Lin [71]	ours
DFT size	8192	8192	64 ~ 2048	512 ~ 8192	512~2048	64 ~ 4096
Algorithm	Radix-4 FFT	Radix-2/4/8 FFT	Cached FFT	Radix-4 DHT-based FFT	Radix-2/4/8 FFT	Cooly-Turkey/ cyclic convolution/ Pseudocirculant factorization/GDA DFT
Word-length (bit)	12	12	16	22	12	20
Process (um)	0.5	0.6	0.35	0.25	0.35	0.18
Clock rate (MHz)	20	20	60	35	45	85
Throughput (sample/cycle)	1	1	1	1	1	5.33 ~ 0.67
Latency (cycle)	N	N	N	N	N	60
Area (mm ²)	100	107	12.25	25	13.05	7.79
Normalized area	12.96	13.87	3.24	12.96	3.45	7.79
Normalized area/throughput	12.96	13.87	3.24	12.96	3.45	1.46 ~ 11.62

Chapter 5

Conclusion

In this chapter, we summarize with some useful results and contributions presented in this dissertation, and point out some future research directions.

5.1 Contributions

In this dissertation, an entire bit-level hardware-efficient group distributed arithmetic (GDA) design approach has been presented for Discrete Sinusoidal transform (DSST's). A new hardware-efficient GDA datapath and the essential partitioning schemes are involved in the development of the proposed new DA design approach for long-length cyclic convolution of the DSST's, where Agarwal-Cooley algorithm and Pseudocirculant matrix factorization algorithm are respectively adopted for the cyclic convolution with prime length and non-prime length. Furthermore, for the long-length DSST's designs, we combine the proposed design approach with the fast transform algorithms, such as Cooley-Tukey algorithm and prime factor algorithm, to achieve the low hardware cost.

In the proposed bit-level design approach, we adopt the way of distributed arithmetic (DA) computation and exploit the good features of the cyclic convolution to facilitate an efficient DA realization of 1-D N-point DSST,s using a very small memory module, a barrel shifter, and N accumulators. The proposed GDA design is achieved by re-arranging the contents of the memory into few groups such that all the elements in a group can be accessed simultaneously in accumulating all the DSST's outputs for increasing the memory utilization. This design reveals that the complexity of DA design is improved from $O(2^N)$ to $O(2^{N-\log_2 N} + N + 2)$.

For the purpose of further reducing the hardware cost in DSST's design, we exploit the symmetrical property of DFT coefficients with the proposed GDA design approach such that the DFT requires only half the contents to be stored, which further reduces the memory size by a factor of two. For the DCT design, we exploit the symmetry property of DCT coefficients, merge the elements in the matrix of DCT kernel, separate the kernel of DCT to be two perfect cyclic forms, and partition the content of memory into groups to facilitate an efficient realization of 1-D N-point

DCT kernel using $(N-1)/2$ adders or subtractors, one small memory module, a $(N-1)/2$ -bit barrel shifter, and $(N-1)/2+1$ accumulators. Compared with the existing systolic array designs and DA-based designs, the realizations of 1-D DFT, DHT, and DCT with the proposed GDA-based design approach reduce the delay-area product more than 29% according to Avanti 0.35 μm CMOS cell library. However, observing the DCT and DHT in cyclic convolution algorithm with non-prime length, there exists the inherent overhead for handling the issue of numerical instability such that the proposed design approach is not efficient for design with this case.

Finally, combining the proposed GDA design approach with the suggested long-length transform decomposition methodology, a variable-length DFT design has been proposed and implemented in our studies for the popular application of DFT with the length of power of two in the communication system. The proposed design can flexibly be used to compute the 1-D 64/128/256/512/1024/2048/4096-point DFT by cascading the 1-D short length DFTs and summing up the partitioned short length cyclic convolutions for each stage of the cascaded DFT. Besides, the proposed hardware efficient design approach can be applied to the design with any length beyond power of two. Compared with the existing long-length and variable-length FFT design, in addition to the advantages of short latency and high hardware utilization efficiency, the proposed power of two variable-length DFT design can achieve competitive hardware cost under the same throughput rate.

5.2 Future Research Directions

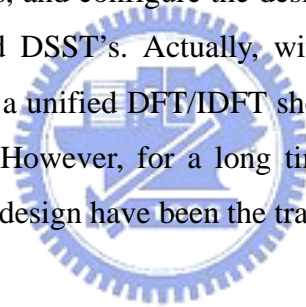
The presented GDA design approach involves cyclic convolution, its partitioning scheme, and hardware efficient GDA implementation. Since the linear convolution and correlation own similar characteristics to cyclic convolution, if any DSP algorithm can be expressed as cyclic convolution, we can apply the proposed GDA design approach to achieve efficient hardware cost for applications.

On the power consumption point of view, with the approach of address grouping in the proposed GDA design, we will further explore how to decide the set of seed partial products for groups in the memory module of GDA design to have optimal transition activity on the bit-line of memory and achieve lower power consumption. However, since the optimal arrangement of these seed partial products depends on the characteristic of image sequences as well as the distribution of input

data, there should exist an optimal arrangement of seed partial products for each kind of image sequences.

For the application of prime-length DCT, since the prime length cyclic convolution DCT algorithm has less overhead in the pre- and post- processing, the GDA-based variable-length DCT design should be an alternative hardware-efficient DA solution for the shape adaptive discrete cosine transform (SA-DCT) in MPEG-4 codec application. However, since there exist more overhead in non-prime length cyclic convolution DCT, this part of realization in SA-DCT must be combined with the existing DA design or the other efficient design.

Based on the derivation of DSST's in cyclic convolution, a unified GDA-based design of DSST's should be a considerable approach for the hybrid system with the requirements of multimedia and communication such as the portable devices. With a commonly used memory module in the GDA design, we can preload the corresponding partial products, and configure the design with different data flow for computations of the involved DSST's. Actually, with the acceptable overhead in cyclic convolution algorithm, a unified DFT/IDFT should be the possible design for communication applications. However, for a long time, the approaches of general purpose design and dedicated design have been the traded-off between flexibility and hardware cost.



Bibliography

- [1] T. M. Pytosh and A. M. Magnasi, "A new parallel 2-D FFT architecture," *Proc. ICASSP1990*, pp. 905-908, 1990.
- [2] J. Choi and V. Boriakoff, "A new linear systolic array for FFT computation," *IEEE Transaction on Circuits and Systems-II: Analog and Digital Signal Processing*, Vol. 39, pp. 236-239, April 1992.
- [3] J. You and S. S. Wong, "Serial-parallel FFT array processor," *IEEE Transaction on Signal Processing*, Vol. 41, pp. 1472-1476, March 1993.
- [4] V. Boriakoff, "FFT computation with systolic arrays, a new architecture," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Vol. 41, pp. 278-284, April 1994.
- [5] H. E. Shousheng and M. Torkelson, "A new approach to pipeline FFT processor," *Proc. IPSP1996*, pp. 766-770, 1996.
- [6] H. T. Kung, "Why systolic architectures?" *Computer Magazines*, 15, pp. 37-45, Jan. 1982.
- [7] L. W. Chan and M. Y. Chen, "A new systolic array for discrete Fourier transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36, pp. 1665-1666, Oct. 1988.
- [8] J. A. Beraldin, T. Aboulnasr, and W. Steenaart, "Efficient one-dimensional systolic array realization of the discrete Fourier transform," *IEEE Transactions on Circuits and Systems*, Vol. 36, No. 1, pp. 95-100, Jan. 1989.
- [9] E. Chan and S. Panchanathan, "A VLSI architecture for DFT," *Proc. the 36th Midwest Symposium on Circuits and Systems*, Vol. 1, pp. 292-295, 1993.
- [10] N. R. Murthy and M. N. S. Swamy, "On the real-time computation of DFT and DCT through systolic architectures," *IEEE Transactions on Signal Processing*, Vol. 42, No. 4, pp. 988-991, Apr. 1994.
- [11] W. H. Fang and M. L. Wu, "An efficient unified systolic architecture for the computation of discrete trigonometric transforms," *Proc. ISCAS1997*, Vol. 3, pp. 2092-2095, 1997.

- [12] C. H. Paik and M. D. Fox, "Fast Hartley transform for image processing," *IEEE Transactions on Med. Imaging*, Vol. 7, No. 6, pp. 149-153, 1988.
- [13] P. Duhamel and M. Vetterli, "Improved Fourier and Hartley transform algorithms: application to cyclic convolution of real data," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASP-35, No. 6, pp. 818-824, 1987.
- [14] R. N. Bracewell, "Discrete Hartley transform," *J. Opt. Soc. Amer.*, Vol.73, No.12, pp. 1832-1835, 1983.
- [15] R. N. Bracewell, "The fast Hartley transform," *Proc. IEEE*, Vol. 72, No. 8, pp. 1010-1018, 1984.
- [16] J. A. C. Bingham, "Multicarrier modulation for data transmission: An idea whose time has come," *IEEE Communications Magazine*, pp. 5-14, May 1990.
- [17] J. S. Chow, J. C. Tu, and J. M. Cioffi, "A discrete multi-tone transceiver system for HDSL applications," *IEEE Journals on Selected Areas and Communications*, Vol. 9, pp. 895-908, Aug. 1991.
- [18] C. L. Wang and C. H. Chang, "A Novel DHT-based FFT/IFFT Processor for ADSL Transceivers," *Proc. IEEE International Symposium on Circuits and Systems*, Vol. 1, pp. 51-54, 1999.
- [19] C. L. Wang and C. H. Chang, "A DHT-based FFT/IFFT Processor for VDSL Transceivers," *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2, pp. 1213-1216, 2001.
- [20] C. L. Wang, C. H. Chang, J. L. Fan, and J. M. Cioffi, "Discrete Hartley transform based multicarrier modulation," *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 5, pp. 2513-2516, 2000.
- [21] H. Bogucka, "Effective implementation of the OFDM/CDMA base station transmitter using joint FHT and IFFT," *Proc. IEEE Workshop on Signal Processing Advances in Wireless Communications*, pp. 162-165, 1999.
- [22] K. J. R. Liu and C. T. Chiu, "Unified parallel lattice structures for time-recursive discrete cosine/sine/Hartley transforms," *IEEE Transactions on*

- Acoustics, Speech, and Signal Processing*, Vol. 41, No. 3, pp. 1357-1377, March 1993.
- [23] S. B. PAN and R. H. Park, "Unified Systolic Arrays for computation of Discrete Hartley Transform," *IEEE Trans. on Circuits and Systems Video Technology*, Vol. 7, No. 2, pp. 413-419, Apr. 1997.
- [24] J. H. Hsiao, L. G. Chen, T. D. Chiueh, and C. T. Chen, "Novel systolic array design for the discrete Hartley transform with high throughput rate," *Proc. IEEE International Conference on Circuits and Systems*, Chicago, IL, U.S.A, pp. 1567-1570, 1993.
- [25] J. I. Guo, C. M. Liu, and C. W. Jen, "A novel CORDIC-based array architecture for the multi-dimensional discrete Hartley transform," *IEEE Transactions on Circuits and Systems*, Vol. 42, No. 5, pp. 349-355, 1995.
- [26] S. P. Kumar and K. M. M. Prabhu, "Novel CORDIC-based systolic arrays for the DFT and the DHT," *Proc. Asia High Performance Computing on the Information Superhighway*, pp. 547-551, 1997.
- [27] A. S. Dhar and S. Banerjee, "An array architecture for fast computation of discrete Hartley transform," *IEEE Transactions on Circuits and Systems*, Vol. 38, No. 9, pp. 1095-1098, 1991.
- [28] W. H. Fang and J. D. Lee, "Efficient CORDIC-based systolic architectures for the discrete Hartley transform," *IEE Proceedings, Computers and Digital Techniques*, Vol. 142, No. 3, pp. 201-207, May 1995.
- [29] L. W. Chang and S. W. Lee, "Systolic arrays for the discrete Hartley transform," *IEEE Transactions on Signal Processing*, Vol. 39, No. 11, pp. 2411-2418, 1991.
- [30] J. I. Guo, C. M. Liu, and C. W. Jen, "A novel VLSI array design for the discrete Hartley transform using cyclic convolution," *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, Adelaide, SA, Australia, pp. II501-II504, 1994.
- [31] J. I. Guo, "A New DA-Based Array for One Dimensional Discrete Hartley Transform," *Proc. 2001 IEEE International Symposium on Circuits and Systems*, Sydney, Australia, pp. IV662-IV665, May 2001.

- [32] J. I. Guo, "An Efficient Design for One Dimensional Discrete Hartley Transform Using Parallel Additions," *IEEE Transactions on Signal Processing*, Vol. 48, No. 10, pp. 2806-2813, Oct. 2000.
- [33] J. I. Guo, C. M. Liu, and C. W. Jen, "The efficient memory-based VLSI array designs for DFT and DCT," *IEEE Trans. Circuits Syst. II*, Vol. 39, pp. 723-733, Oct. 1992.
- [34] S.A. WHITE, "Applications of distributed arithmetic to digital sequence processing: a tutorial review," *IEEE ASSP Magazines*, Vol. 6, No. 3, pp. 5-19, 1989.
- [35] J. P. Choi, S. C. Shin, and J.G. Chung, "Efficient ROM size reduction for distributed arithmetic," *Proc. ISCAS2000*, pp. II61-II64, May 2000.
- [36] K. Nourji and N. Demassieux, "Optimal VLSI Architecture for Distributed Arithmetic-based Algorithm," *ICASSP1994*, Vol. 2, pp. 509-512, 1994.
- [37] M. T. SUN, T. C. Chen, and A. M. Gotlieb, "VLSI implementation of a 16 x 16 discrete cosine transform," *IEEE Transactions on Circuits and Systems*, CAS-36, pp. 610-617, Apr. 1989.
- [38] T. S. Chang, J. I. Guo, and C. W. Jen, "Hardware Efficient DFT Designs with Cyclic Convolution and Subexpression Sharing," *IEEE Transactions on Circuits and Systems II*, Vol. 47, No. 9, pp. 886-892, Sep. 2000.
- [39] T. S. Chang, C. Chen, and C. W. Jen, "New distributed arithmetic algorithm and its application to IDCT," *IEE Proc. on Circuits, Devices, and Systems*, Vol. 146, No. 4, pp. 159-163, 1999.
- [40] J. I. Guo, "An Efficient Parallel Adder Based Design for One Dimensional Discrete Fourier Transform," *Proceedings of the National Science Council, ROC, Part A*, Vol. 24, No. 3, pp. 195-204, May 2000.
- [41] R. C. Agarwal and J. W. Cooley, "New Algorithms for Digital Convolution," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-25, pp. 392-410, Oct. 1977.
- [42] M. Teixeira and D. Rodriguez, "A class of fast cyclic convolution algorithms based on block pseudocirculant," *IEEE Signal Processing Letters*, Vol. 2, No. 5, pp. 92-94, May 1995.

- [43] AVANT “0.35 micron 3.3-volt high performance standard cell library,” 1996.
- [44] A. P. Chandrakasan and R. W. Brodersen, “Minimizing power consumption in digital CMOS circuit,” *Proceeding of the IEEE*, Vol. 83, No. 4, pp. 498-523, April, 1995.
- [45] T. Xanthopoulos and A. P. Chandrakasan, “A low power DCT core using adaptive bandwidth and arithmetic activity exploiting signal correlations and quantization,” *IEEE Journal of Solid-State Circuits*, Vol. 35, No. 5, pp. 740-750, 2000.
- [46] H. K. Garg, “Digital signal processing algorithms - number theory, convolution, fast fourier transforms, and application,” CRC Press, 1998.
- [47] J. E. Volder, “The CORDIC trigometric computation technique,” *IRE Tran. Electron. Comput.*, Vol. EC-8, pp. 330-334, Sep. 1959.
- [48] J. S. Walther, “A unified algorithm for elementary functions,” *AFIPS Spring Joint Comput. Conf.*, pp. 379-385, 1971.
- [49] K. Hwang, “Computer Arithmetic principles, architecture, and design,” John Wiley & Sons, Inc., New York, 1979.
- [50] A. V. Oppenheim and R. W. Schaffer, “Discrete-time Signal Processing,” Prentice-Hall, Englewood Cliffs, NJ, U.S.A, 1989.
- [51] Y. H. Chan and W. C. Siu, “Generalized approach for the realization of discrete cosine transform using cyclic convolution,” *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, Minneapolis, MN, U.S.A, Vol. 3, pp. III277-III280, 1993.
- [52] J. I. Guo, "Efficient parallel adder based design for one dimensional discrete cosine transform," *IEE Proceedings Circuits, Devices, and Systems*, Vol. 147, No. 5, pp. 276-282, Oct. 2000.
- [53] J. H. McClellan, and C. M. Rader, “Number Theory in Digital Signal Processing,” Prentice-Hall, 1979.
- [54] B. Arambepola, “Discrete Fourier transform processor based on the prime-factor algorithm,” *IEE Proc.*, 130, Pt. G, No. 4, pp. 138-144, 1983.

- [55] H. Lim, and E. E. Swartzlander, "Multidimensional systolic arrays for the implementation of discrete Fourier transforms," *IEEE Transactions on Signal Processing*, Vol. 47, No. 5, pp. 1359-1370, May 1999.
- [56] C. S. Burrus, "Index mappings for multidimensional formulation of the DFT and convolution," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-25, pp. 239-242, 1977.
- [57] C. S. Burrus and T. W. Parks, "DFT/FFT and Convolution Algorithms," John Wiley & Sons, 1985.
- [58] H. V. Sorensen, D. L. Jones, C. S. Burrus, and M. T. Heideman, "On Computing the Discrete Hartley Transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-33, pp. 239-242, Oct. 1985.
- [59] C. Chakrabarti and J. Ja'Ja', "Systolic Architectures for the Computation of the Discrete Hartley and the Discrete Cosine Transforms Based on Prime Factor Decomposition," *IEEE Transactions on Computer*, Vol.39, No.11, pp. 1359-1368, Nov. 1990.
- [60] B. G. Lee, "Input and output mappings for a prime-factor-decomposed computation of discrete cosine transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 37, No. 2, pp. 237-244, Feb. 1989
- [61] J. McClellan and C. M. Rader, "There is something much faster than the fast Fourier transform," Seminar Notes, Oct. 21, 1976.
- [62] C. H. Chang, C. L. Wang, and Y. T. Chang, "Efficient VLSI architectures for fast computation of the discrete Fourier transform and its inverse," *IEEE Transactions on Signal Processing*, Vol. 48, No. 11, pp. 3206-3216, Nov. 2000.
- [63] S. F. Hsiao and W. R. Shiue, "Design of low-cost and high-throughput linear arrays for DFT computations: algorithms, architectures, and implementations," *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing*, Vol. 47, No. 11, pp.1188-1203, Nov. 2000.
- [64] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall, Inc. 1975.

- [65] E. H. Wold and A. M. Despain, "Pipeline and Parallel pipeline FFT processors for VLSI implementation," *IEEE Transaction on Computers*, Vol. C-33, No. 5, pp. 414-426, 1984.
- [66] S. He and M. Torkelson, "Designing Pipeline FFT Processor for OFDM (de)Modulation," 1998 URSI International Symposium on Signals, Systems, and Electronics, pp. 257 -262, 1998.
- [67] E. Bidet, D. Castelain, C. Joanblanq, and P. Senn, "A fast single-chip implementation of 8192 complex point FFT," *IEEE Journal of Solid-State Circuits*, Vol. 30, No. 3, pp. 300-305, Mar. 1995.
- [68] L. Jia, "A new VLSI-oriented FFT algorithm and implementation," *IEEE ASIC Conference*, pp. 337-341, 1998.
- [69] J. C. Kuo, C. H. Wen, C. H. Lin, and A. Y. Wu, "VLSI Design of a Variable-Length FFT/IFFT Processor for OFDM-based Communication Systems," in *Special Issue on "Signal Processing for Broadband Access Systems: Techniques and Implementations," EURASIP Journal on Applied Signal Processing*, No. 13, pp. 1306-1316, Dec. 2003
- [70] T. C. Pao, C. C. Chang, and C. K. Wang, "A variable-length DHT-based FFT/IFFT processor for VDSL/ADSL systems," *IEEE Asia-Pacific Conference on Circuits and Systems*, pp. 381-384, 2004.
- [71] Y. T. Lin, P. Y. Tsai, and T. D. Chiueh, "Low-power variable-length fast Fourier transform processor," *IEE Proc. Comput. Digit. Tech.*, Vol. 152, No. 4, pp. 499-506, 2005.
- [72] B. M. Bass, "A low-power high performance, 1024-point FFT processor," *IEEE Journal of Solid-State Circuit*, Vol. 34, No. 3, pp. 380-387, Mar. 1999.



VITA

Hun-Chen Chen was born in Taiwan in 1961. He received the B.S. and M.S degrees, all in electronics engineering, from National Taiwan Technology University, and National Chiao-Tung University, Taiwan, in 1990 and 1998, respectively. He is currently pursuing the Ph.D. degree in low-cost bit-level DSP VLSI design and its applications to multimedia and communication systems. His research interests include VLSI digital signal processing and computer architecture.

