TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY
Julkaisu 481 • Publication 481

Jari Nikara

Application-Specific Parallel Structures for Discrete Cosine Transform and Variable Length Decoding

Tampere 2004

Tampereen teknillinen yliopisto. Julkaisu 481
Tampere University of Technology. Publication 481

Jari Nikara

Application-Specific Parallel Structures for Discrete Cosine Transform and Variable Length Decoding

Thesis for the degree of Doctor of Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB104, at Tampere University of Technology, on the 18th of June 2004, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2004

ISBN 952-15-1196-6 (printed)
ISBN 952-15-1405-1 (PDF)
ISSN 1459-2045

ABSTRACT

This Thesis considers the design of application-specific parallel structures for digital signal processing. Due to wideness of the subject, the discussion has been restricted to the studies of the discrete cosine transform and variable length decoding.

New area-efficient parallel structures, which process data in a sequential form at data rate, are developed for the discrete cosine transform. The development of the structures begins with the derivation of novel regular fast algorithms. The algorithms lend themselves for vertical mapping resulting in modular cascaded structures that can be freely pipelined due to the loop-free structure. In order to prove the feasibility and estimate the performance, the unified transform kernel for discrete cosine transform and its inverse is implemented on a standard cell CMOS technology with a data path synthesis. Finally, the comparison to a state-of-the-art design reveals up to 15% smaller estimated area than in the reference design.

For the variable length decoding, a novel multiple-symbol decoding scheme is proposed. The critical path of the resulting decoder is minimized by introducing a new multiplexed add unit. In order to prove the feasibility and determine the limiting factors of the scheme, the decoder has been implemented on an FPGA technology. When applied to MPEG-2 standard benchmark scenes, on average 4.8 codewords are decoded per cycle resulting in the throughput of 106 million symbols per second. Although, a straightforward and fair comparison of variable length decoders is extremely difficult due to different implementation approaches, the performance of the decoder can be considered promising with 16–100 % better throughput at 2–3.6 times lower frequencies than the reference designs on the same FPGA technology.

In both the case studies, the discrete cosine transform and variable length decoding, the modularity and achievable high speed operation provide flexibility for the design re-use in the current and future applications.

PREFACE

This research work has been carried out during the years 2000 – 2004 at the Institute of Digital and Computer Systems, former Digital and Computer Systems Laboratory, of Tampere University of Technology, Tampere, Finland as a part of the wider related research projects of which one included a one-year visit to Computer Engineering Laboratory, Delft University of Technology, Delft, The Netherlands.

I am grateful to my supervisors, Prof. *Jarmo Takala* and Prof. *Stamatis Vassiliadis*, for guiding and encouraging me to study and work towards doctoral degree. Furthermore, my sincere thanks to my Thesis reviewers, Prof. *Olli Silven* and Prof. *Jorma Skyttä*, for their constructive comments on the manuscript. Special thanks to *Tuomas Järvinen*, M.Sc., *Perttu Salmela*, M.Sc., Mr. *Harri Sorokin*, *Vesa Lahtinen*, M.Sc., *Georgi Kuzmanov*, M.Sc., for sharing ideas, knowledge, and opinions about those various different topics related, more or less if at all, to this Thesis.

I would like to thank all the co-authors: Prof. *David Akopian*, *Mihai Sima*, M.Sc., *Petri Liuha*, M.Sc., Prof. *Jaakko Astola*, *Jukka Saarinen*, Dr. Tech., *Konsta Punkka*, M.Sc.. In addition, *Rami Rosendahl*, M.Sc., deserves special thanks for co-operation. My warmest thanks to personnel in Tampere and Delft for nice working atmosphere.

This Thesis was financially supported by the Graduate School in Electronics, Telecommunications, and Automation (GETA), Academy of Finland, National Technology Agency (TEKES), Nokia Foundation, Jenny and Antti Wihuri Foundation, Ulla Tuominen Foundation, and the Foundation of Advancement of Technology, which are gratefully acknowledged.

Finally, I would like to express my gratitude to my parents *Erkki* and *Marjatta Nikara*, sisters *Virpi* and *Kirsi*, and my wife *Anu* for their support and love during these years.

Tampere, May 2004

Jari Nikara

TABLE OF CONTENTS

Abstract i

Preface iii

Table of Contents v

List of Publications ix

List of Figures xi

List of Tables xv

List of Abbreviations xvii

List of Symbols xxi

1. Introduction 1

1.1 Objective and Scope of Research 3

1.2 Main Contributions 4

1.2.1 Author's Contribution 5

1.3 Thesis Outline 6

2. Discrete Cosine Transform 7

2.1 Definitions and Properties 8

2.2 One-Dimensional Fast Algorithms 9

2.2.1 Well-Known Pioneer Algorithms 10

2.2.2 Regular Algorithms 12

2.3 Two-Dimensional Algorithms 16

2.4 Hardware Structures 18

2.4.1 One-Dimensional Structures 19

2.4.2 Two-Dimensional Structures with Matrix Transpose 21

2.4.3 Direct Two-Dimensional Structures 25

2.5 Summary 27

3. Perfect Shuffle Topology DCT Algorithms 29

3.1 Preliminaries 29

3.2 One-Dimensional Transform 35

3.2.1 Processing Columns 37

3.2.2 Interconnection Permutations 38

3.2.3 Final Algorithm 41

3.3 Two-Dimensional Transform 43

3.3.1 Row-Column Decomposition 43

3.3.2 Direct Computation 45

3.4 Summary 48

4. Pipeline Structures for Perfect Shuffle Topology DCT 49

4.1 Basic Processing Units 49

4.2 Basic Permutation Units 52

4.3 Final Structures 54

4.4 Case Study: Unified DCT and IDCT 57

4.4.1 Algorithms and Structures 57

4.4.2 Implementation of Direct 8×8 DCT/IDCT 61

4.5 Summary 64

5. Variable Length Coding 67

5.1 Definitions and Properties 67

5.2 Variable Length Decoders 70

5.2.1 Serial Decoders 71
5.2.2 Parallel Decoders 73
5.2.3 Multiple-Symbol Decoders 78
5.3 Summary 82
6. Variable Length Decoding Scheme 85
6.1 Algorithm 85
6.1.1 Decoding Example 88
6.2 General Structure 89
6.2.1 Decoder example 92
6.3 Critical Path 93
6.4 Summary 94
7. MPEG-2 Variable Length Decoding 95
7.1 Overview to MPEG-2 Standard 95
7.2 Decoder Specification 96
7.3 Hardware Model 97
7.3.1 Codeword Detector 98
7.3.2 Chrominance Format Counter 98
7.3.3 Multiplexed Add 99
7.3.4 Memory Address Generator 99
7.3.5 Symbol Fetch 101
7.3.6 Entire Decoder 101
7.4 Experimental Results 103
7.5 Discussion and Comparison 106
7.5.1 Reference Decoders 107
7.5.2 Comparison Results 109
7.6 Summary 111

8. Conclusions 113

Bibliography 115

LIST OF PUBLICATIONS

This Thesis is a monograph which contains some unpublished material. However, the thesis is based on the work already published during the research studies in the following international publications. In the text, these publications are referred to as [P1], [P2], . . . , [P7].

[P1] J. Nikara, J. Takala, D. Akopian, J. Astola, and J. Saarinen, "Sequential architecture for discrete cosine transform," in *Proceedings of the 18th NORCHIP Conference*, Turku, Finland, Nov. 6–7 2000, pp. 53–58.

[P2] J. Nikara, J. Takala, D. Akopian, and J. Saarinen, "Pipeline architecture for DCT/IDCT," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 4, Sydney, Australia, May 6–9 2001, pp. 902–905.

[P3] J. Nikara and J. Takala, "Unified pipeline architecture for in-order 8x8 DCT and IDCT," in *Proceedings of the 5th World Multiconference in Systemics, Cybernetics, and Informatics*, vol. 4, Orlando, FL, U.S.A., July 22–25 2001, pp. 30–35.

[P4] J. Takala, J. Nikara, and K. Punkka, "Pipeline architecture for two-dimensional discrete cosine transform and its inverse," in *Proceedings of the 9th IEEE International Conference on Electronics, Circuits and Systems*, vol. 3, Dubrovnik, Croatia, Sept. 15–18 2002, pp. 947 – 950.

[P5] J. Nikara, S. Vassiliadis, J. Takala, M. Sima, and P. Liuha, "Parallel multiple-symbol variable-length decoding," in *Proceedings of the IEEE International Conference on Computer Design*, Freiburg, Germany, Sept. 16–18 2002, pp. 126–131.

[P6] J. Nikara, S. Vassiliadis, J. Takala, and P. Liuha, "FPGA-based variable length decoders," in *Proceedings of the IFIP WG 10.5 International Con-*

ference on Very Large Scale Integration of System-on-Chip, Darmstadt, Germany, Dec. 1–3 2003, pp. 437 – 441.

[P7] J. Nikara, S. Vassiliadis, J. Takala, and P. Liuha, “Multiple-symbol parallel decoding for variable length codes,” to appear in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 7, July 2004.

LIST OF FIGURES

1 Signal flow graph to compute the DCT via real parts of the DFT . . 10

2 Signal flow graph of the DCT based on the sparse matrix factorization 11

3 Signal flow graph of the DCT based on the rotations 12

4 Signal flow graph of the in-place DCT with separated irregularities . 13

5 Signal flow graph of the in-place DCT with distributed irregularities 14

6 Signal flow graph of the constant geometry DCT algorithm 15

7 Signal flow graph of the direct 8×8 DCT by utilizing an 8-point DCT 17

8 Block diagram of the pipeline structure with double buffering 19

9 Block diagram of the fully sequential DCT pipeline 21

10 Register-based matrix transpose network by using double buffering . 22

11 Block diagrams of the basic switching units 22

12 Example of the iterative 8×8 matrix transpose 23

13 Block diagram of the column parallel 8×8 matrix transpose 24

14 Block and timing diagrams of the sequential matrix transpose 25

15 Block diagram of the 8×8 DCT/IDCT with matrix transpose 26

16 Block diagram of the fully sequential 8×8 direct DCT 27

17 Simplification of the interconnection permutation 39

18 Signal flow graph of the regular perfect shuffle topology DCT . . . 43

19 Signal flow graph of the regular perfect shuffle topology 8×8 DCT 47

20 Block diagrams of the basic data processing units 50

21 Unified butterfly unit 51
22 Direct two-dimensional butterfly unit 52
23 Block diagrams of the basic data permutation units 53
24 Sequential matrix transpose network for $2^k \times 2^k, k \leq 3$, matrix . . . 54
25 Pipeline structures for regular perfect shuffle topology DCT 56
26 Signal flow graphs of the 8-point DCT and IDCT 58
27 Block diagrams of the 8-point DCT kernels 59
28 Unified input/output permutation network 60
29 Block diagrams of the unified two-dimensional DCT/IDCT 61
30 Block diagram of the verification environment of the DCT/IDCT . . 63
31 Example of the Huffman coding 69
32 Block diagram of the generalized variable length decoding. 71
33 Variable length decoding as a finite-state machine 72
34 Direct mapping of the VLD onto demultiplexers and storage locations 73
35 Block diagram of the parallel constant output rate decoder 74
36 Block diagram of the loop-free parallel decoder 75
37 Block diagram of the parallel group-based decoder 77
38 Block diagram of the variable I/O rate multiple-symbol decoder . . 79
39 Block diagram of the group-based multiple-symbol decoder 80
40 Block diagram of the 2-symbol decoder with an additional shifter . . 80
41 Block diagram of the 2-symbol decoder with parallel symbol look-ups 81
42 Block diagram of the 2-symbol decoder with length prediction . . . 82
43 Principle of the proposed variable length decoding scheme 90
44 Block diagram of the generalized codeword detection 90
45 Block diagram of the 8-to-1 multiplexed add unit 91

46 Principal structure of entire variable length decoder. 92

47 Block diagram of the MPEG-2 modified codeword detector 98

48 Block diagrams of the MPEG-2 specific units 99

49 Block diagram of the MPEG-2 variable length decoder 102

50 Throughput of the proposed approach 104

51 Distribution of symbols over the decoder outputs 105

LIST OF TABLES

1 Characteristics of the 8×8 DCT/IDCT pipeline implementation . . 62

2 Comparison of the 8×8 DCT/IDCT structures 63

3 Gate count estimates for the basic units 64

4 Comparison of the fully sequential unified DCT/IDCT pipelines . . 64

5 Summary of the Huffman coding example 69

6 Variable length decoding example 89

7 Properties of the MPEG-2 benchmark scenes 97

8 Memory address generation in the demonstration implementation . . 100

9 Experimental results of the MPEG-2 variable length decoder 106

10 Characteristics of the MPEG-2 variable length decoder 107

11 Comparison of the FPGA-based variable length decoders 110

LIST OF ABBREVIATIONS

ACM	Association for Computing Machinery
AO	And-Or
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction-set Processor
BU	Butterfly Unit
CCITT	the International Telegraph and Telephone Consultative Committee
CD	Codeword Detector
CFC	Chrominance Format Counter
CLA	Carry Look-Ahead
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CW	Codeword
D	Delay register
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DHT	Discrete Hartley Transform
DIF	Decimation-In-Frequency

DIT	Decimation-In-Time
DHT	Discrete Hartley Transform
DRU	Data Reordering Unit
DSD	Delay-Switch-Delay unit
DSP	Digital Signal Processor
DWHT	Discrete Walsh-Hadamard Transform
EOB	End-Of-Block
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FSM	Finite-State Machine
IDCT	Inverse Discrete Cosine Transform
IDRU	Inverse Data Reordering Unit
IEC	International Electrotechnical Commission
IEE	the Institution of Electrical Engineers
IEEE	the Institute of Electrical and Electronics Engineers
IEICE	the Institute of Electronics, Information and Communication Engineers
I/O	Input/Output
IRE	Institute of Radio Engineers
ISO	International Organization for Standardization
ITU	International Telecommunication Union
ITU-T	the ITU Telecommunication Standardization Sector
JPEG	Joint Photographic Experts Group

LEU	Local Exchange Unit
LIFO	Last-In-First-Out
LSB	Least Significant Bit
LSU	Local Subtraction Unit
LUT	Look-Up Table
M	Multiplexer
MA	Multiplexed Add
MAG	Memory Address Generator
MPEG	Moving Picture Experts Group
MSB	Most Significant Bit
MT	Matrix Transpose
MVM	Matrix-Vector Multiplier
PLA	Programmable Logic Array
PP	Post-Processor
PU	Post-processing Unit
PS	Perfect Shuffle
RAM	Random Access Memory
ROM	Read Only Memory
S	Switch
SEU	Shift-Exchange Unit
SF	Symbol Fetch
SoC	System on Chip
UBU	Unified Butterfly Unit

VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLC	Variable Length Coding
VLD	Variable Length Decoding
VLSI	Very Large Scale Integration
WSES	World Scientific and Engineering Academy and Society

LIST OF SYMBOLS

Discrete Cosine Transform

C_N^{i}	discrete cosine transform matrix of type i of order N
N	input sequence size
b_m	DCT scaling factor of mth output
I_N	identity matrix of order N
$\overline{I}_N$	anti-diagonal identity matrix of order N
P_N	permutation matrix of order N
$P_{N,R}$	stride-by-R permutation matrix of order N
X	sample vector
x_i	ith sample value
P_N^H	Hadamard permutation matrix of order N
$h_N(\cdot)$	Hadamard permutation function for N-point vector
J_N	J permutation matrix of order N
F_N	radix-2 butterfly matrix of order N
D_N^*	DCT scaling matrix of order N
$d(\cdot)$	DCT coefficient function of order N
$C_N^{(0)}$	Hadamard ordered DCT matrix
$M_N^{(s)}$	local subtraction matrix of order N of sth processing column

$D_N^{(s)}$	DCT coefficient matrix of order N of sth processing column
$A_N^{(s)}$	sth processing column matrix of order N
$U_N^{(s)}$	output permutation matrix of order N of the sth processing column
$V_N^{(s)}$	input permutation matrix of order N of the sth processing column
$\mu_n(\cdot)$	binary-valued parametrization function to unify representation of the processing columns
$H_N^{(s)}$	local exchange matrix of order N of the sth processing column
$\tau_i(\cdot)$	binary-valued parametrization function to unify representation of the DCT coefficient matrices
$C_{N\times N}$	two-dimensional DCT transform matrix of order N^2
P_N^O	output permutation matrix of order N
K_N	DCT kernel matrix of order N
P_N^I	input permutation matrix of order N
$A_{N\times N}^{(s)}$	sth processing column matrix of order N^2
$D_{N\times N}^{(s)}$	DCT coefficient matrix of order N^2 of the sth processing column
$P_{N\times N}^I$	input permutation matrix of order N^2
$P_{N\times N}^O$	output permutation matrix of order N^2

Variable Length Decoding

s_k	kth symbol
p_k	probability of the symbol s_k
S	set of symbols
H	entropy
c_k	codeword for the symbol s_k

l_k	length of the codeword c_k
l_{av}	average codeword length
c_i^k	ith code symbol of the codeword c_k
L	set of codeword lengths
l_{min}	minimum codeword length
l_{max}	maximum codeword length
d_{max}	maximum number of codewords with equal length
X	input data stream
x_i	ith input data stream element
B	sliding window over input data stream
b_i	ith data element in the sliding window
idx	index to the first undecoded bit in the input data stream
N	length of the sliding window
M	maximum number of the codewords in the sliding window
W_i	ith codeword in the sliding window
L_i	length of the ith codeword in the sliding window
j_i	index to starting location of the ith codeword in the sliding window
J_i	set of possible starting points for the ith codeword in the sliding window
J	set of possible starting points for the codewords in the sliding window
m_i	codeword length at position i in the sliding window
A_i	index to symbol table for ith codeword in the sliding window

τ	delay of 3-4 AND-OR
t_M	delay of the multiplexer
t_{MA}	delay of the multiplexed add
T	codeword table parameter

1. INTRODUCTION

In general, advances in technologies allow the implementation of more complex systems. On the other hand, they also encourage engineers to design complex systems requiring advanced solutions and technologies. Likewise, the development creates new markets for several novel systems. As a result, the amount of all kind of information processing is increasing around us all the time. Consequently, more computational power—more speed is required in future [24]. However, the cost of design is the greatest threat to continuation of the semiconductor roadmap [46]. Therefore, the main issue is to support real-time applications with making, however, the compromise between performance and cost.

History has proven the cyclic behaviour between hardware and software implementations of signal processing applications [85]. While an application is implemented initially on software, its first real-time implementation requires typically application-specific hardware. Later, the advances in technology make the software implementation again possible. At that time, however, newer and more aggressive algorithms have been studied, causing the cycle to repeat itself.

The main advantage of software implementations is flexibility; modifications on the functionality can be realized by re-programming without physical changes on hardware. Therefore, the software implementations are often preferred. On software, however, there is always some overhead due to instruction fetch, instruction decode, and perhaps inappropriate instruction set. In addition, different data and processing rates require buffering of data. All these facts imply either reduced speed, increased area, or power consumption. Altogether, the software implementation is always a trade-off between flexibility, speed, area, and power consumption. Despite of the rapid advances in processor architectures and integrated circuit technologies, there are always applications requiring higher performance than provided by a state-of-the-art programmable processor at reasonable cost.

In general, higher performance can be achieved by exploiting parallelism. The resulting increased costs are kept reasonable by adding just some application-specific features, i.e., special instructions or even customized functional units on hardware, as done in digital signal processors (DSP). Another approach is application-specific instruction-set processors (ASIP), where the instruction-set and hardware are tailored according to the requirements of the given application. Customization level in ASIPs varies on three architectural levels depending on the approach: instruction extension, inclusion or exclusion of predefined blocks, and parametrization of cache sizes, number of registers, etc. [32]. One actively studied solution nowadays is reconfigurable computing, which achieves potentially higher performance than software but adopts a higher level of flexibility than hard-wired hardware. The reconfigurable systems contain usually a general-purpose processor managing data-dependent control and possible memory accesses, while the computational tasks are mapped onto the reconfigurable hardware [18]. The extreme level of customization is represented by application-specific integrated circuits (ASIC) based on standard cell or full custom design, which are designed to perform only specified computation with very limited control.

All the previously discussed approaches offer distinct advantages and drawbacks in applications. Therefore, there is no single ultimate solution suitable for all possible design cases but they are exploited as combinations. Furthermore, when considering the hardware design its abstraction level has changed from logic design via standard cell design to block based design and there seems to be no sign to end of this trend—systems keep on becoming more and more complex. The number of processing units will increase while the overall gate count for custom logic is decreasing [32]. This arises a demand for packing know-how into intellectual property (IP) blocks which may be re-used. This ongoing change introduces two major problems: where to get the components from, and how to verify that they work together as desired [71].

Although the systems have become more complex, the life cycles of applications have shortened due to rapid advances in technologies and various applications. This implies less time for the design and test and thus, emphasizes flexibility for the design re-use. The flexibility enables covering a wide range of applications. E.g., the structure with achievable high-speed operation covers high data rate applications but it can be also exploited as a shared resource in low data rate applications. Simple interfaces and control make the structure easy to integrate into larger systems, e.g., as an accelerator in parallel with the DSP, building block of the ASIP, or configurable unit in a reconfigurable system. In any case, due to the cyclic behaviour between hardware and software implementations, acceleration with application-specific hardware, and changes in abstraction level and complexity of the design, there is continuous need for the studies of design and implementation of signal processing functions on hardware.

The studies in this Thesis consider the development of application-specific parallel structures for the discrete cosine transform (DCT) and variable length decoding (VLD). The attractiveness of the DCT is based on its predominant use in transform coding [86]. It has been applied actively to data compression applications, e.g., in speech, image, and video coding. It can be exploited in filtering and subsequently, in transmultiplexer systems based on filter banks as in [116]. Other applications that involve the DCT are, e.g., data analysis, classification, and pattern recognition [86]. Variable length coding (VLC) and subsequently the VLD have been utilized either alone for data compression or as a part of compression application, e.g., for text, speech, image and video compression. Since both the DCT and VLD have an essential role in image and video coding [31, 74, 94] they have been adopted as a part of various current standards, e.g., JPEG [42, 44], MPEG-1 [41], and MPEG-2 [43, 45]. Although the real-time applications based on the previous standards can nowadays be implemented on software, the performance is a serious bottleneck in some applications, e.g., in multichannel coding, transcoders, video servers, and related professional applications processing multiple video streams.

1.1 Objective and Scope of Research

The general objective of this Thesis is to develop area-efficient realizations of digital signal processing functions. Software implementations introduce overhead and typically instruction level parallelism is limited implying lower utilization of arithmetic resources. Furthermore, data memory bandwidth is often a serious bottleneck in high data rate applications. Therefore, in this Thesis, only application-specific parallel structures are considered. The structures are developed in such a way that the throughput can be tailored according to the input data rate of an application. Furthermore, the aim is to create general methodology to construct modular structures from the given algorithms. In general, modularity is considered to describe the granularity of the algorithm or structure.

In this Thesis, two types of systems are considered: constant and variable output rate systems. Constant output rate systems are illustrated with the computation of the discrete cosine transform. In general, the DCT algorithms possess irregularity, which restricts the area-efficient exploitation of inherent parallelism. Regularity defines the similarities in nodes and interconnections. Therefore, the objective is to derive the regular fast algorithm for the DCT. Regularity allows the utilization of linear mapping methods [84] for reducing the dimensionality of the problem and mapping the algorithm onto a parallel structure. Furthermore, since the data is often in a sequential form, another objective is to develop a single rate system, i.e., data rate equals to clock rate.

In this Thesis, the variable output rate systems are exemplified with variable length decoding. The major design problem is to break the recursive data dependencies in input data, which complicate substantially the design of parallel variable length decoder. Therefore, the objective is to develop a multiple-symbol VLD scheme and parallel structure that will at least partially break the recursive dependency related to the VLD.

1.2 Main Contributions

In this Thesis, the high-performance computational platforms are developed by making use of inherent parallelism of the given applications. In the first application, discrete cosine transform, the temporal parallelism is utilized while in the second application, variable length decoding, spatial parallelism is exploited. The resulting hardware structures are modular lending themselves to very large scale integration (VLSI) implementations. To summarize, the main contributions are the following:

- Up-to-date survey of related work in respect of the pipeline computation of the discrete cosine transform and multiple-symbol variable length decoding, which provides motivation and bases for the work presented in this Thesis.
- Novel regular fast algorithms for the one- and two-dimensional discrete cosine transforms, which do not reach the lower bound on arithmetic complexity but the regularity allows efficient utilization of temporal parallelism.
- New modular pipeline structures for computing the discrete cosine transform, which can be extended to support larger transform sizes by replicating the basic processing units.

- So far the most area-efficient unified pipeline structure supporting both the 8×8 discrete cosine transform and its inverse.

- Demonstration implementation of the unified structure supporting both the 8×8 discrete cosine transform and its inverse, which is synthesized onto a standard cell CMOS technology.

- Novel variable length decoding scheme for decoding multiple symbols in parallel.

- New multiplexed add unit solution, which reduces the number of logic levels in the critical path of the codeword detection in variable length decoder.

- Demonstration implementation of MPEG-2 variable length decoder, which is synthesized onto an FPGA technology.

1.2.1 Author's Contribution

Let us first consider the studies of DCT. The derivation of the fast DCT algorithms is continuation for the algorithm development reported earlier, e.g., in [2, 5, 104, 105]. Author rescheduled the constant geometry algorithm and found possibilities to optimize interconnections for minimizing the area of permutations in pipeline structures. The author had an essential role in deriving and formalizing the regular perfect shuffle topology algorithms. The development of computational structures for the derived algorithms was done by the author. In addition, the author has supervised the analysis of word length requirements, modeling, and synthesis of the structure.

The author was responsible for the deriving and verifying the multiple-symbol variable length decoding scheme. The author was also responsible for developing the structure, as well as applying the scheme for MPEG-2 variable length decoding. Furthermore, the author analyzed the pre-processed MPEG-2 data streams and analyzed the performance of the proposed decoder with different design parameters.

The work reported in this Thesis has been reported earlier in seven publications [P1–P7] and in six of them, the author has been the main author. Consequently, some chapters contain verbatim extracts from the publications. With respect to the extracts, copyrights are retained by the respective copyright holders.

The co-authors of the publications [P1–P7] have seen this clarification and agree with the author. In addition, none of the publications have been used in another person's academic thesis or dissertation.

1.3 Thesis Outline

The first three chapters cover the studies on discrete cosine transform. In Chapter 2, an introduction to the DCT and the related work is given in order to provide background and motivation for work reported ensuing two chapters. Starting with the definitions and properties of the DCT the discussion continues with a glance at some popular algorithms and implementations close to pipeline structures. The chapter is concluded with a short summary. Subsequently, the derivation of the novel regular fast algorithms for the one- and two-dimensional DCT is studied in Chapter 3. After providing the preliminaries for the derivation, the formulation of the algorithms, which are exploited and partially reported earlier in [P1 – P4], is described in details. Chapter 4 covers the derivation of the new pipeline structures for the DCT, IDCT, and both the transforms, which are published earlier in [P1 – P4]. First, the operational columns in the algorithms are mapped vertically onto basic units which are then cascaded to construct the pipelines. The chapter is concluded with a case study on the unified 8×8 DCT/IDCT implementation and comparative discussion.

Then, the topic of the discussion is changed to variable length decoding on hardware in Chapter 5. Before going into the details of decoding, the variable length coding is studied briefly. Subsequently, the related work of the decoders is outlined in order to have basis for the further research of the VLD. Next, in Chapter 6, a novel VLD scheme is described, which has been proposed for the first time in [P5] and studied more carefully in [P7]. Furthermore, the structure of the decoder is presented before discussion on the performance in general. In Chapter 7, the presented VLD scheme and structure are applied to MPEG-2 video coding standard, which has been reported earlier in [P7]. First, the standard is outlined briefly in order to understand the fundamentals that affect to decoder design. Before modeling the structure, the specifications are determined according to the statistics of the benchmark scenes. Then, the performance of the resulting decoder is analyzed with different design parameters. The chapter is concluded with the comparison and the discussion on related problems reported in [P6]. Finally, Chapter 8 concludes Thesis.

2. DISCRETE COSINE TRANSFORM

By applying appropriate transforms, the complexity of a mathematical problem may be reduced, e.g., differential and integral equations may be replaced with the easier algebraic ones [86]. Representing a waveform having relatively complex variations in a signal amplitude with a sum of the oscillatory cosine function is called a cosine transform. When the waveform and cosine functions are sampled at certain intervals, the transform becomes a discrete cosine transform [74]. The DCT has been considered one of the best tools in digital signal processing and therefore, it has many applications, e.g., in the area of multimedia and telecommunications. Especially, in image and video coding it has been employed as the main tool for data compression.

Traditionally the objective in the development of fast algorithms in the field of digital signal processing has been the minimization of arithmetic complexity, i.e., the number of arithmetic operations and especially the number of multiplications. However, in hardware realizations, the number of arithmetic units depends on mapping methods, thus there are also other properties in the algorithms reflecting on the cost of a specific implementation. Especially, when targeting at structures consisting of cascaded processing units, there are specific properties, which make certain algorithms efficient for implementations.

In this chapter, an introduction to the DCT and the related work is given in order to provide background and motivation for our work. Rao and Yip's book [86] gives a good baseline for the discussion but thereafter several algorithms and structures have been reported. However, without taking restricted publication-specific comparisons into account extensive up-to-date surveys are missing. In addition, our objective is to develop a cascaded structure, which, in general, is referred to as a pipeline. The principal idea is to reduce the dimensionality of the signal flow graph of the algorithm by applying vertical mapping, i.e., the two-dimensional signal flow graph is collapsed or folded into a one-dimensional data path. Consequently, each processing column is to be mapped onto a single processing unit. Therefore, in the following survey, some popular fast algorithms are introduced and their suitability for pipeline computations are discussed. Similarly, some DCT implementations close to pipeline structures are briefly outlined.

2.1 Definitions and Properties

The orthogonal DCT is classified into four different types: DCT of type I, II, III, and IV of which transform matrices are defined as [118]

$$\begin{aligned}
\left[C^{\mathrm{I}}_{N+1}\right]_{mn} &= \sqrt{\frac{2}{N}}\left[b_m b_n \cos\left(\frac{mn\pi}{N}\right)\right], \quad m,n=0,1,\ldots,N && (1)\\
\left[C^{\mathrm{II}}_{N}\right]_{mn} &= \sqrt{\frac{2}{N}}\left[b_m \cos\left(\frac{m(n+\frac{1}{2})\pi}{N}\right)\right], \quad m,n=0,1,\ldots,N-1 && (2)\\
\left[C^{\mathrm{III}}_{N}\right]_{mn} &= \sqrt{\frac{2}{N}}\left[b_n \cos\left(\frac{(m+\frac{1}{2})n\pi}{N}\right)\right], \quad m,n=0,1,\ldots,N-1 && (3)\\
\left[C^{\mathrm{IV}}_{N}\right]_{mn} &= \sqrt{\frac{2}{N}}\left[\cos\left(\frac{(m+\frac{1}{2})(n+\frac{1}{2})\pi}{N}\right)\right], \quad m,n=0,1,\ldots,N-1 && (4)
\end{aligned}$$

where b_m is a scaling factor defined as

$$b_m = \begin{cases} \frac{1}{\sqrt{2}} & \text{, if } m=0 \text{ or } m=N \\ 1 & \text{, if } m\neq 0 \text{ and } m\neq N \end{cases} . \qquad (5)$$

The DCT of type I (DCT-I) was introduced by Wang and Hunt in [120]. The first definitions of the DCT and its inverse, which according to previous classification are known as the DCT of type II (DCT-II) and DCT of type III (DCT-III), respectively, were given by Ahmed *et al.* in [1]. The DCT of type IV (DCT-IV) was introduced by Jain in [47].

Let us next summarize the main properties of the DCT matrices given in (1)–(4). Since the DCT matrices are orthogonal the inverse transform matrices are obtained with a matrix transpose. In addition, the DCT-I and DCT-IV matrices are symmetric which means that the inverse transform is the transform itself. On the contrary, the DCT-II and DCT-III are transposes for each others. These relations can be formulated as

$$\left[C^{\mathrm{I}}_{N+1}\right]^{-1} = \left[C^{\mathrm{I}}_{N+1}\right]^{T} = C^{\mathrm{I}}_{N+1}, \qquad \left[C^{\mathrm{II}}_{N}\right]^{-1} = \left[C^{\mathrm{II}}_{N}\right]^{T} = C^{\mathrm{III}}_{N},$$
$$\left[C^{\mathrm{III}}_{N}\right]^{-1} = \left[C^{\mathrm{III}}_{N}\right]^{T} = C^{\mathrm{II}}_{N}, \qquad \left[C^{\mathrm{IV}}_{N}\right]^{-1} = \left[C^{\mathrm{IV}}_{N}\right]^{T} = C^{\mathrm{IV}}_{N}. \tag{6}$$

From the multidimensional point of view, an essential property is separability which allows the decomposition of the multidimensional transform into successive one-dimensional transforms. [86]

In image and video processing, especially the two-dimensional DCT-II and its inverse, i.e., DCT-III, have gained popularity due to good energy compaction properties. The computation of the DCT-II as a matrix product is computationally expensive thus several fast algorithms for the DCT-II have been suggested over the years. Before taking an overview into the fast DCT-II algorithms and their properties, let us note that in this Thesis, we concentrate only on the DCT-II and DCT-III. Therefore, from now on, the DCT and its inverse refer to DCT-II and DCT-III, respectively.

2.2 One-Dimensional Fast Algorithms

The DCT can be computed via other discrete trigonometric transforms, e.g., discrete Fourier transform (DFT), discrete Walsh-Hadamard transform (DWHT) and discrete Hartley transforms (DHT) [70, 113, 115]. However, such approaches result in additional computational complexity. The fast algorithms with lower arithmetic complexity can be obtained by considering the direct factorization of the DCT matrix. When the factorization results in sparse component factors, the decomposition represents a direct fast algorithm for the DCT. Since the matrix factorization is not unique, the different types of the fast algorithms can be derived. Some of the proposed fast algorithms derived with the matrix factorization can be categorized into decimation-in-time or decimation-in-frequency algorithms as in [126, 127].

In the following, some fast algorithms are described. First, well-known pioneer algorithms are introduced, which have advantages in some realizations but introduce critical drawbacks when targeting at the area-efficient pipeline computation at data rate. In such cases, regularity in particular is a beneficial property. Therefore, a viewpoint is focused on regular algorithms, which represents novelty in the survey.

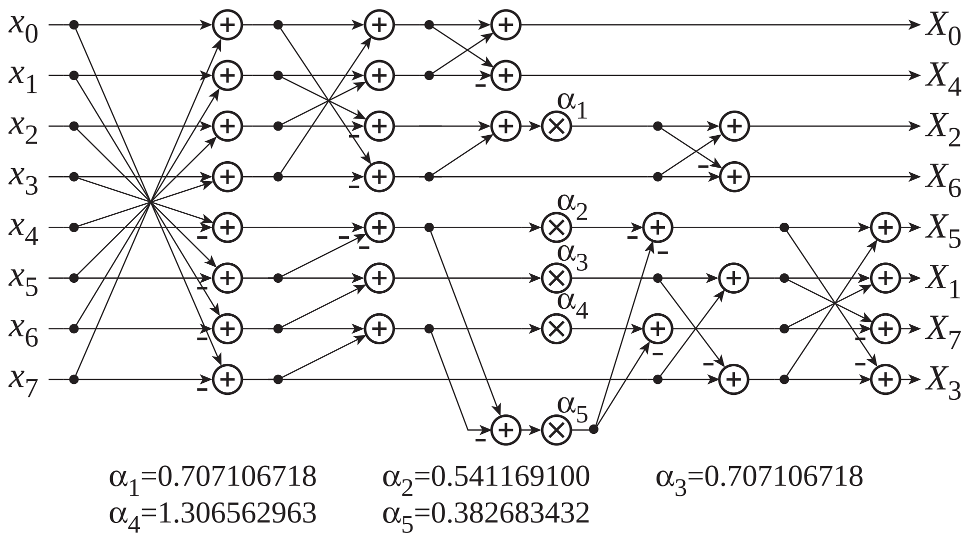


Fig. 1. *Signal flow graph to compute the DCT via real parts of the DFT* [3].

2.2.1 Well-Known Pioneer Algorithms

The natural basis for developing the fast algorithms for the DCT is its relation to a discrete Fourier transform (DFT). In addition to introducing the discrete cosine transform, Ahmed *et al.* in [1] suggested the computation of the DCT by using one double length fast Fourier transform (FFT). In [28], Haralick accelerated computation by taking two N-point FFTs instead one double length FFT. Tseng and Miller in [111] proved, however, that the DCT can be computed more efficiently with the aid of one double length modified FFT and especially exploiting only its real parts. As an example, a signal flow graph to compute the scaled DCT via real parts of the 16-point DFT as proposed by Arai *et al.* in [3] is illustrated in Fig. 1. In any case, the relation between the FFT and DCT can be utilized in various different ways as described, e.g., in [69, 79, 115].

The first fast DCT algorithm based on the sparse matrix factorization of the DCT matrix has been reported by Chen *et al.* in [11] and Wang in [117]. Since the presented factorization of the algorithm consists of a matrix which cannot be recursively factorized, the factorization is only partially recursive. However, the non-recursive matrix possesses some regularity; it can be decomposed into the product of sparse matrices, which are of five distinct types, all having at most two non-zero elements in each row [86]. The resulting signal flow graph of the 8-point DCT algorithm is illustrated in Fig. 2.

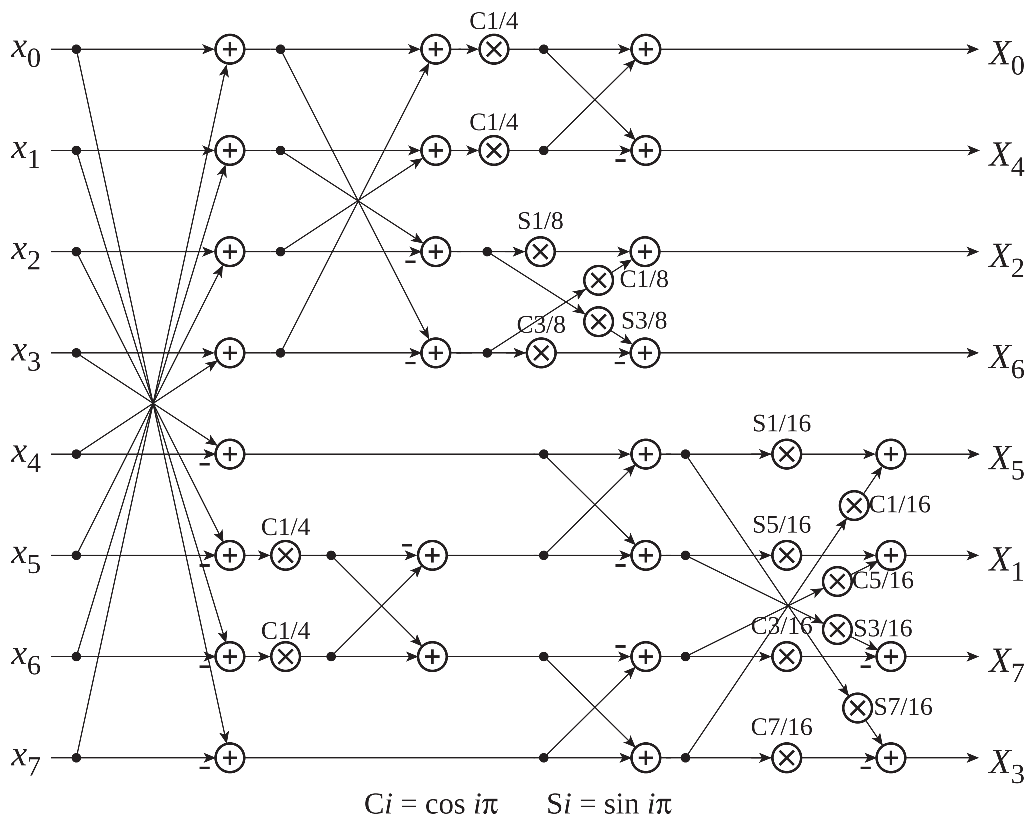
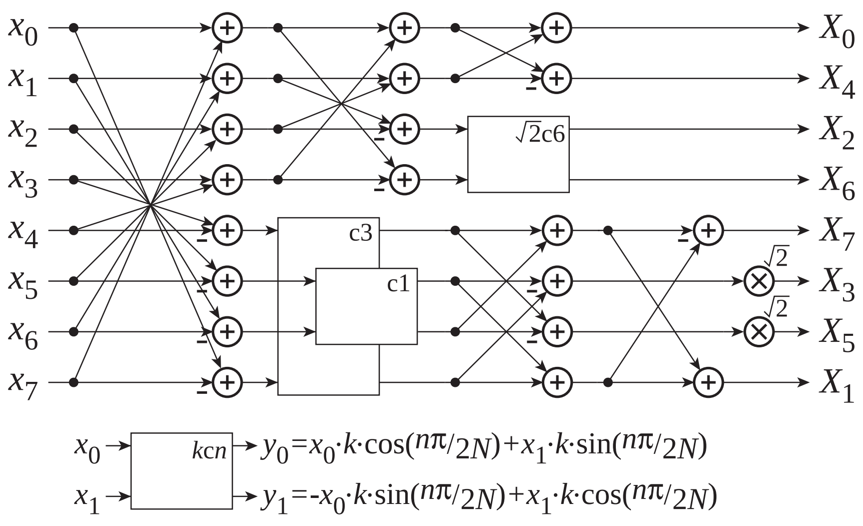


Fig. 2. *Signal flow graph of the DCT based on the sparse matrix factorization* [11].

Loeffler *et al.* in [67] presented a class of the 8-point DCT algorithms based on planar rotations. The algorithms require only 11 multiplications, which is shown to be the theoretical minimum number of the multiplications for the 8-point transform [22]. The number of the multiplications is decreased from the traditional approaches by interpreting the multiplications as rotations, which can be performed with three multiplications and three additions by introducing two new coefficients instead of four multiplications and two additions. For example, consider the following formulation [67]

$$\begin{aligned} y_0 &= a \cdot x_0 + b \cdot x_1 &&= (b-a) \cdot x_1 + a \cdot (x_0 + x_1) \\ y_1 &= -b \cdot x_0 + a \cdot x_1 &&= -(a+b) \cdot x_0 + a \cdot (x_0 + x_1). \end{aligned} \tag{7}$$

An example of Loeffler's 8-point DCT algorithm is depicted in Fig. 3. The 8-point algorithm can be extended in a recursive way to support larger transform sizes but the resulting algorithms do not reach anymore the theoretical lower bound on the number of the multiplications.



x_0
x_1
x_2
x_3
x_4
x_5
x_6
x_7
X_0
X_4
X_2
X_6
X_7
X_3
X_5
X_1
$\sqrt{2}$c6
c3
c1
$\sqrt{2}$
$\sqrt{2}$
kcn
$y_0=x_0 \cdot k \cdot \cos(n\pi/2N)+x_1 \cdot k \cdot \sin(n\pi/2N)$
$y_1=-x_0 \cdot k \cdot \sin(n\pi/2N)+x_1 \cdot k \cdot \cos(n\pi/2N)$

Fig. 3. *Signal flow graph of the DCT based on the rotations* [67].

2.2.2 Regular Algorithms

In general, the fast algorithms for the DCT do not possess the regularity found, e.g., in Cooley-Tukey FFT algorithms. For example, the previously discussed fast DCT algorithms, Chen's sparse matrix factorization and Loeffler's algorithm reaching the theoretical lower bound on the number of multiplications with planar rotations, do not result in area-efficient pipeline structures due to their high control complexity. In addition, the algorithms consist of the variable number of operations at operational columns implying that the resulting pipeline structure would be a multirate system, i.e., the pipeline would require a higher clock rate than a sample rate. Nevertheless, some regular fast algorithms for the computation of the DCT have been reported over the years.

In-Place Algorithms

One category of the regular fast DCT algorithms is well-known in-place algorithms where the computations can be performed in-place, i.e., the results of intermediate computations can be stored into the same locations as the operands. Consequently, the in-place algorithms lend themselves especially to software implementations due to efficient memory utilization. However, these algorithms have also good properties for vertical mapping due to simple and regular processing columns.

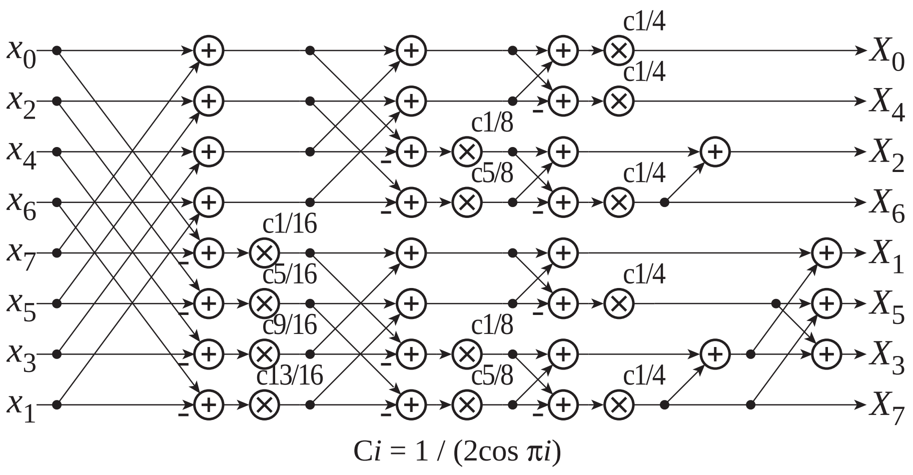


Fig. 4. *Signal flow graph of the in-place DCT with separated irregularities* [34].

The basis for the derivation of the first reported in-place algorithms is the reordering of the input sequence into even and odd indices as proposed earlier in [79]. In addition, Lee in [56] reordered both halves in bit-reversed order when considering the matrix factorization of the IDCT matrix. With the aid of this rearrangement, the N-point IDCT has been decomposed into a sum of two $N/2$-point IDCTs. The transform size can be further reduced by repeating the decomposition. The procedure results in a simple structure with recursive modularity improving the regularity but the drawback is the secant coefficients introducing round off errors with finite word length.

In [19], Cvetković and Popović derived an in-place DCT algorithm which does not require the additional bit-reversed permutation of the input halves. Instead, the output of the algorithm illustrated in Fig. 4 is in bit-reversed order. The same algorithm and the matrix decomposition of the algorithm with the output in natural order have been reported by Hsiao *et al.* in [34]. In order to avoid round off problems with the coefficients of secants, Hou in [33] proposed an in-place algorithm with pure sine or cosine coefficients. However, the approach results in additional multiplications by two. By using the different derivation technique, Lee and Huang in [57] entered corresponding in-place algorithm with a perfect shuffle output permutation [102] which will be defined in Section 3.1.

In all the in-place algorithms discussed so far, the computations are performed with a kernel of real-valued radix-2 butterfly operations and a separate post-processing stage, or alternatively a pre-processing stage, containing all the DCT-related irregular

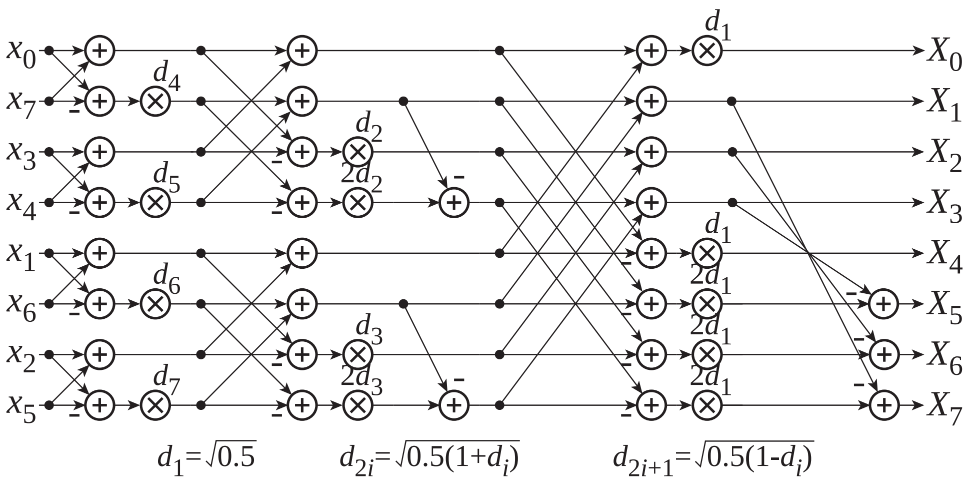


Fig. 5. *Signal flow graph of the in-place DCT with distributed irregularities* [119].

additions and permutations. Instead, the separate pre- or post-processing is avoided in a regular in-place algorithm proposed by Wang in [119] where irregularities are distributed between the butterfly columns as illustrated in Fig. 5. The matrix factorization is based on the successive order reduction of the Hadamard ordered DCT matrix. The algorithm can be characterized as simple as the previous in-place algorithms and its coefficients are pure cosines, which can be generated with the recursive equations depicted below the signal flow graph in Fig. 5. The input sequence is Hadamard ordered, which will be defined in Section 3.1, and the output is obtained in natural order.

Constant Geometry Algorithms

Another category of the regular fast DCT algorithms is constant geometry algorithms where the interconnection topology and geometry between the computation columns are the same, i.e., constant. The constant geometry algorithms have been used in software implementations where sequential data access and storage are required. In addition, the constant geometry algorithms lend themselves to VLSI implementations. Full-column and partial-column structures are typically based on such algorithms. In these structures, the computations are performed iteratively one column of the signal flow graph or part of the column at a time.

A constant geometry algorithm based on Wang's in-place algorithm with the distributed irregularities has been presented by Takala *et al.* in [5, 105]. The derivation

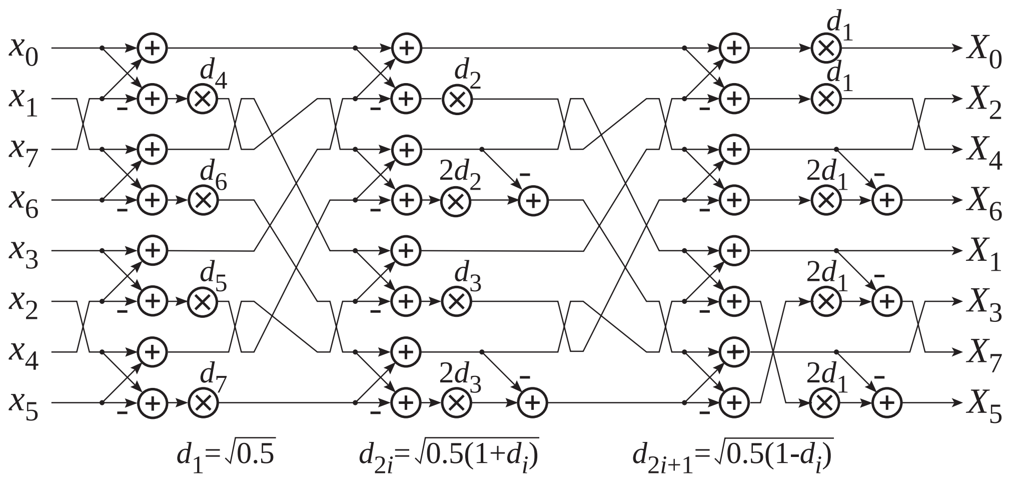


Fig. 6. *Signal flow graph of the constant geometry DCT algorithm* [105].

is based on a method to localize irregularities into block matrices of order four as proposed in [5]. Therefore, the achieved constant geometry algorithm illustrated in Fig. 6 can be interpreted to be a rescheduled version of the in-place algorithm in Fig. 5. The corresponding derivative constant geometry algorithm of Hou's in-place algorithm in [33] has been presented by Kwak and You in [55]. In this algorithm, a computational kernel is performed with consecutive stages of the radix-2 butterfly operations and all the irregularities have been separated into a post-processing column containing only additions.

In general, the post-processing is a disadvantage in unified pipeline realizations, when both forward and inverse transforms need to be supported. The post-processing in the forward transform maps as a pre-processing in the inverse transform, thus in the unified approach both pre- and post-processing need to be realized implying additional hardware cost. In Wang's in-place algorithm in [119], the irregular additions are local, thus they map in similar fashion in forward and inverse transforms. On the other hand, the representation of the algorithm contains anti-diagonal matrices, which imply complex data permutations requiring large data storage in VLSI implementations. Such problems are avoided in Takala's constant geometry algorithm in [105]. The drawback is that the interconnections between the operation columns in an N-point algorithm are N-point permutations implying higher area cost in pipeline structure.

2.3 Two-Dimensional Algorithms

The previous discussion on the categorization and features of the one-dimensional DCT algorithms applies also to two-dimensional DCT algorithms, which can be derived by using either a row-column decomposition or a direct computation. The row-column decomposition exploits the separability property of the DCT, i.e., the two-dimensional transform is computed by performing the one-dimensional DCT first over the rows and then over the columns or vice versa. On the contrary, the direct computation produces the result of the two-dimensional transform at once, i.e., the algorithms operates directly over the two-dimensional data set. In other words, the dimensions are not separated and, therefore, the computation of the one-dimensional transforms over the rows and columns cannot be identified from the direct two-dimensional transform. Let us remark, that the direct two-dimensional algorithm can be, however, derived with the aid of the one-dimensional transform as will be done later in this Thesis.

The direct two-dimensional algorithms are based on the same derivation approaches as the one-dimensional algorithms. Without going into details but outlining the development of the algorithms, the relation between the DFT and DCT has been exploited, e.g., by Makhoul in [69] and Vetterli in [114]. From the authors mentioned previously when considering the one-dimensional DCT algorithms, Lee and Huang in [57], Kwak and You in [55], and Hsiao *et al.* in [35] have extended their approaches to support also the direct two-dimensional computation. Kamangar and Rao in [51] presented non-recursive as well as recursive algorithms for the two-dimensional DCT, which have been derived with an approach very similar to Chen's approach to derive the one-dimensional DCT in [11]. In [27], Haque presented the two-dimensional extension of Lee's algorithm in [56]. Correspondingly, Hou's one-dimensional DCT algorithm in [33] has been extended to two-dimensional DCT by Chan and Ho in [7] and Wu and Paoloni in [124].

In [13–15], Cho *et al.* presented an approach to compute the two-dimensional DCT algorithm by exploiting the one-dimensional algorithm but, however, operating directly over the two-dimensional data set . In addition to any available one-dimensional algorithm only permutations, pre-, and post-additions are required as illustrated in Fig. 7. The presented approach introduces systematic expressions for the algorithms and in that sense, it can be considered to possess some regularity. Furthermore, the

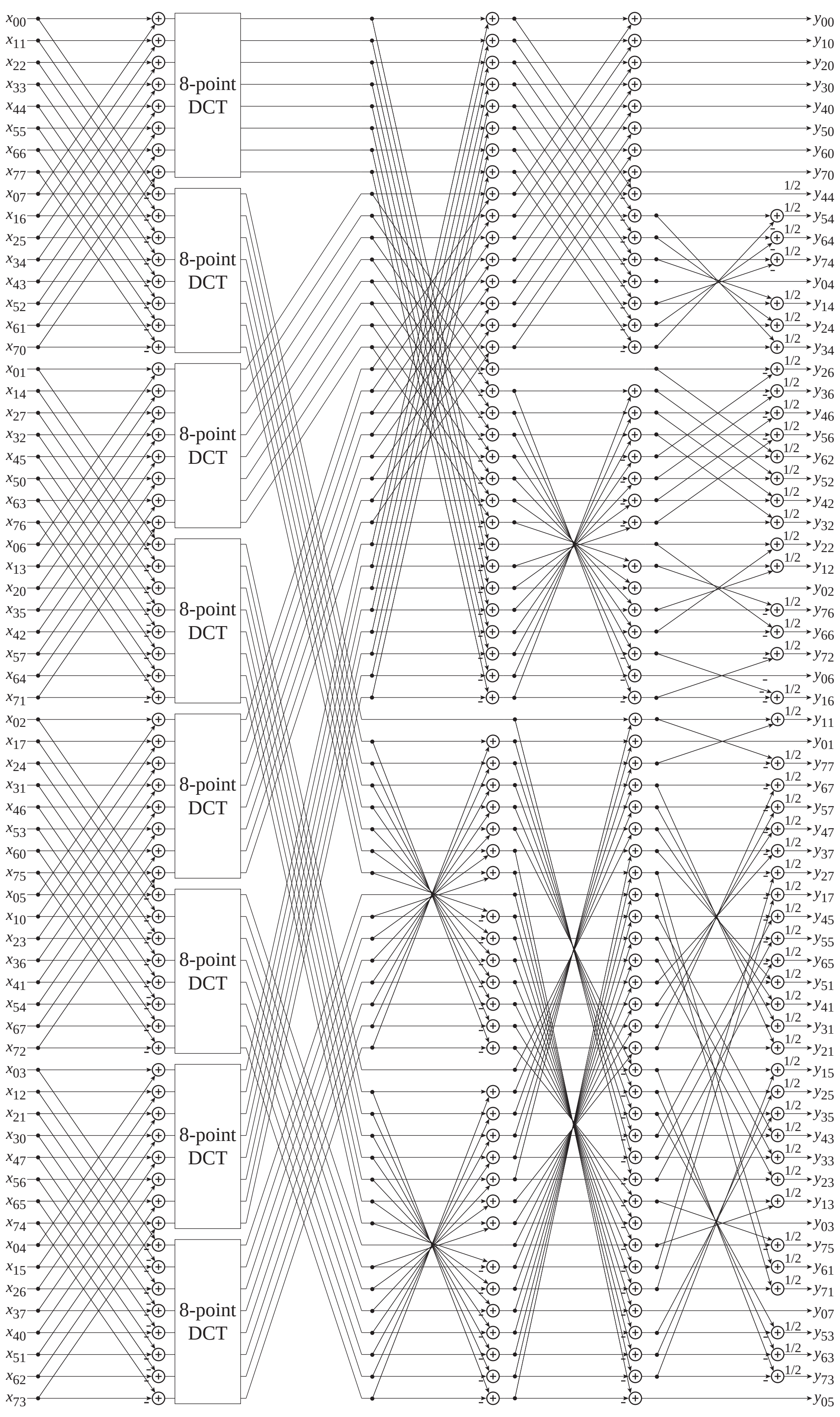


Fig. 7. *Signal flow graph of the direct* 8×8 *DCT by utilizing an* 8*-point DCT* [13].

approach results in the optimal two-dimensional algorithm from the multiplicative complexity theory point of view when Loeffler's DCT algorithm in [67] is used to compute the 8-point DCT [123]. Nevertheless, the resulting algorithm do not possess regularity that would be advantageous when mapping onto hardware. The representation of the post-additions in the algorithm contains anti-diagonal matrices requiring the complex data permutations and large data storage.

In general, the direct computation is considered to produce algorithms with a lower arithmetic complexity and especially with the lower number of multiplications than the row-column decomposition. Typically the control complexity of algorithms derived with the row-column decomposition is lower implying regularity and modularity. Since these properties are preferred especially in VLSI implementations, the row-column decomposition has gained popularity in hardware realizations. It is also possible to obtain regular and modular algorithms with the direct computation but at the expense of arithmetic complexity. However, modularity and regularity may be the key properties to efficient implementations.

2.4 Hardware Structures

The main objectives in realizing the algorithms is to achieve required system performance with minimum cost. In general, the direct implementation or so called one-to-one mapping of the algorithm leads to an expensive realization. Therefore, the mapping methods described, e.g., in [84, 87, 92] can be utilized for reducing the dimensionality of a signal flow graph in different directions; horizontal, vertical, or in both directions. In principle, the exploitation of spatial parallelism, i.e., horizontal mapping, results in a column structure where the computations are performed recursively on parallel data, i.e., nodes at a single processing stage are computed at a time. In such a structure, the throughput is limited by the delay of the basic arithmetic units used to realize the nodes. The exploitation of temporal parallelism with the aid of vertical mapping, in turn, results in sequential structures, where the computations are performed over data in sequential form and the overall structure can be considered as a pipeline. In such a structure, the throughput can be tailored with additional pipeline registers if the data dependencies, i.e., the feedback loops in the structure, can be avoided. In addition, data is often in sequential form, thus structures operating over sequential data are advantageous. In the following, computational structures for such a data format are described.

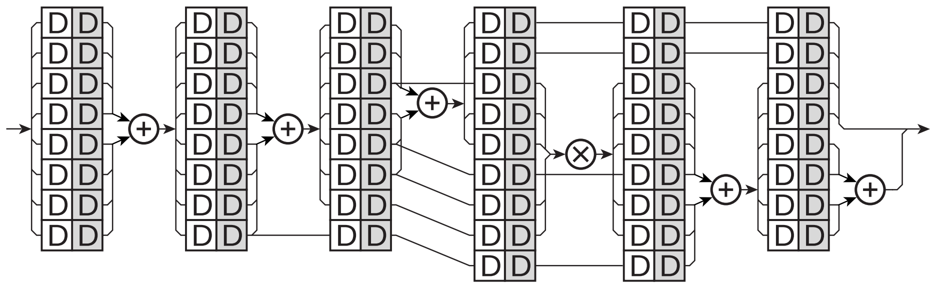


Fig. 8. *Block diagram of the pipeline structure with double buffering* [54]. *D: Delay register. Control logic is omitted for clarity.*

2.4.1 One-Dimensional Structures

In [54], Kovać and Ranganathan presented a six-stage pipeline structure based on the modified Arai's DCT algorithm in Fig. 1. With the aid of the modifications the number of subtractions has been decreased but they do not affect the structure, i.e., Arai's original algorithm could be mapped onto the same structure. In any case, the resulting pipeline structure is illustrated in Fig. 8. Each operational column in the algorithm has been mapped onto a corresponding stage in the pipeline. The operational stages are separated with register-based double buffering stages. In other words, a left column of the registers is first filled and then copied onto a right column for computation. During the computation over the values in the right register column the left column is refilled. Altogether, the structure requires high number of register and introduces complicated control. Let us remark that the control and selection resources are not included into the figure.

The irregularities in the DCT algorithms restrict the exploitation of the linear mapping methods. Hence the structures based on Chen's fast DCT algorithm [11, 117] have been fully parallel and consequently expensive. The reduction in the amount of hardware has been achieved by using advanced circuit techniques and optimizations that are very close to technology and, in general, time demanding. E.g., in [112], Uramoto *et al.* have exploited multiplier accumulators based on distributed arithmetic in order to have critical path of adders instead of multipliers. For minimizing the routing area and its parasitics, a column-interleaved memory structure has been employed. In addition, the number of memory cells has been halved with the aid of dual-plane feature and dual-port configuration. Correspondingly, Matsui *et al.* in [72] achieved improvements in chip area and speed by introducing and applying a sense-amplifying pipeline flip-flop circuit technique. Instead of the conventional synchronous design style, an asynchronous DCT processor has been presented by Johnson *et al.* in [50].

The linear mapping methods can be efficiently applied to the regular fast DCT algorithms although the irregularities may limit the exploitation of the parallelism or decrease the utilization rate of resources. In principle, applying the vertical mapping to the in-place algorithms corresponds to serializing the computation. The level of the mapping can be varied to have different level of parallelism. E.g., Cheng *et al.* in [12] have mapped the in-place algorithm with regular butterfly kernel and post-processing stage managing the irregularities only half. Instead in [110], Tan *et al.* have applied vertical mapping to Hou's algorithm [33] in order to have fully sequential structure for sequentially represented data. The relation between the DCT and DFT can also be utilized in the structural derivation. E.g., in [108], Takala *et al.* have mapped Wang's in-place DCT algorithm with the distributed irregularities [119] onto a pipeline structure by using an efficient mapping technique, which has been proposed earlier by Groginsky and Works in [26].

In [34], Hsiao *et al.* mapped their DCT algorithm efficiently onto a pipeline structure depicted in Fig. 9. The structure has single input and output interface and each computational stage has been mapped vertically onto a corresponding unit, i.e., butterfly stages onto processing elements BU_4, BU_2, and BU_1 with single arithmetic unit, multiplications onto multipliers and post-processing stages onto post-processors PP_1 and PP_2. With such an arrangement, the amount of hardware is minimized. In addition, the required throughput rate can be achieved with the aid of additional pipelining, since the structure does not consist of feedback paths. Furthermore, due to regularity the structure is comparable easy to extend to support larger transform sizes.

The vertical mapping can also be applied to the constant geometry algorithms in [55, 105] in order to select an optimal level of parallelism with respect to performance requirements and resources. On the other hand, the constant interconnections may increase the hardware cost in pipeline structures when the constant interconnection network is repeated between every stage. Instead, the constant interconnections are, in principle, advantageous for horizontal mapping. However, the pure column structures for computing the DCT are rare due to irregularities in the fast algorithms but the principal concept can be found, e.g., in [48,105]. Likewise, the applying mapping in both directions, i.e., multiprojection, has not been reported when considered only the DCT structures.

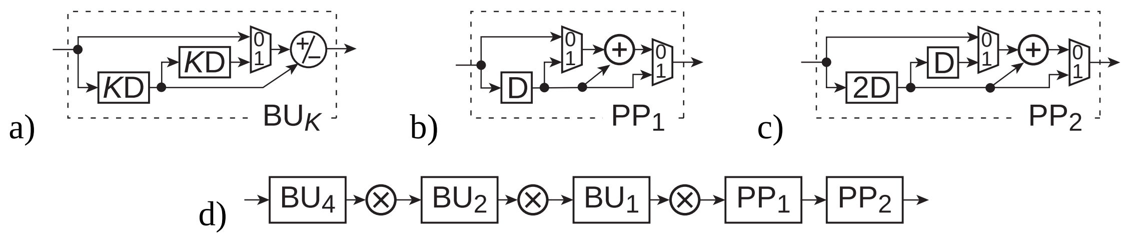


Fig. 9. *Block diagram of the fully sequential DCT pipeline* [34]*: (a) radix-2 butterfly unit having operands K samples apart (BU$_K$), (b) post-processor 1 (PP$_1$), and (c) post-processor 2 (PP$_2$), and (d) entire pipeline structure. KD: Shift register of size K. Clock and control signals are omitted for clarity.*

2.4.2 Two-Dimensional Structures with Matrix Transpose

In the two-dimensional transforms based on the row-column decomposition, large silicon area may be consumed into the realization of the matrix transpose. The most straightforward realization of the matrix transpose is to exploit double buffering according to the direct interpretation of the transpose, i.e., rows in, columns out. The implementation can be memory-based as in [50, 112] or register-based as in [54]. The realization based on the double buffering is, however, expensive; $2N^2$ storage locations are needed for an $N \times N$ matrix transpose as illustrated with the aid of a register-based transpose network in Fig. 10. Furthermore, latency is increased since all the samples must be stored before reading. If the $N \times N$ transpose is to be performed with N^2 memory locations like in [68], either dual-port memory or higher write/read-rate is needed since the data is in sequential form, i.e., when a new sample is written in, a transposed sample need to be read out. This implies higher cost for memory-based transpose units.

In the advanced register-based transpose networks, two principal switching units, i.e., a 2-port delay-switch-delay unit (DSD) and 1-port shift-exchange unit (SEU) introduced in [98], are exploited to perform the reordering of the data elements. In principle, the delay-switch-delay unit of size K (DSD$_K$) depicted in Fig. 11(a) exchanges the first K data elements entering to the lower port with the K elements entering to the upper port. Correspondingly, the shift-exchange unit of size K (SEU$_K$) illustrated in Fig. 11(b) is capable of exchanging elements in a serial sequence K elements apart. The latencies of these units depend on the size of the shift registers; the latency of DSD$_K$ or SEU$_K$ is K cycles.

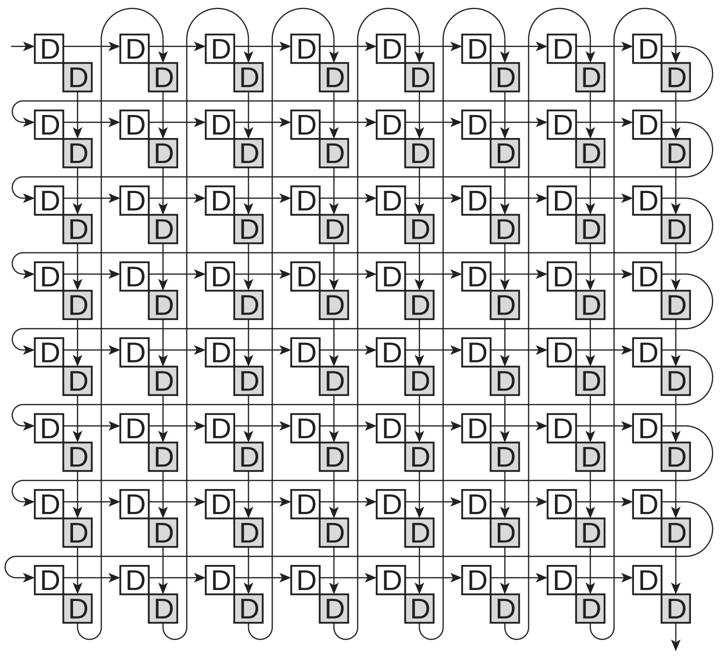


Fig. 10. *Register-based matrix transpose network by using double buffering* [54].

In [98], Shung *et al.* proposed a sequential permutation network which can be used to perform any arbitrary permutation over sequentially represented data. In general, the arbitrary permutations of an N-point sequence, $N = 2^k$, can be realized as the sequential permutation network of the cascaded SEUs arranged in increasing and decreasing order: $\text{SEU}_{2^0}, \text{SEU}_{2^1}, \ldots, \text{SEU}_{2^{k-1}}, \ldots, \text{SEU}_{2^1}, \text{SEU}_{2^0}$. Such a realization requires less registers than the conventional double buffering approach but still more than the original data.

In [6], Carlach *et al.* proposed an iterative method for an 8×8 matrix transpose, which has been generalized for an $N \times N, N = 2^k$, matrix by Takala *et al.* in [109]. In principle, the entire $N \times N$ matrix is divided into $(N/2)^2$ submatrices of order

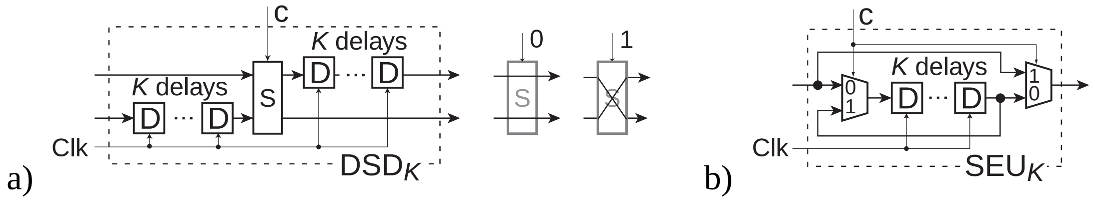


Fig. 11. *Block diagrams of the basic switching units: (a) delay-switch-delay unit of size K (DSD$_K$) and (b) shift-exchange unit of size K (SEU$_K$). S: Switch. c: Control signal. Clk: Clock signal.*

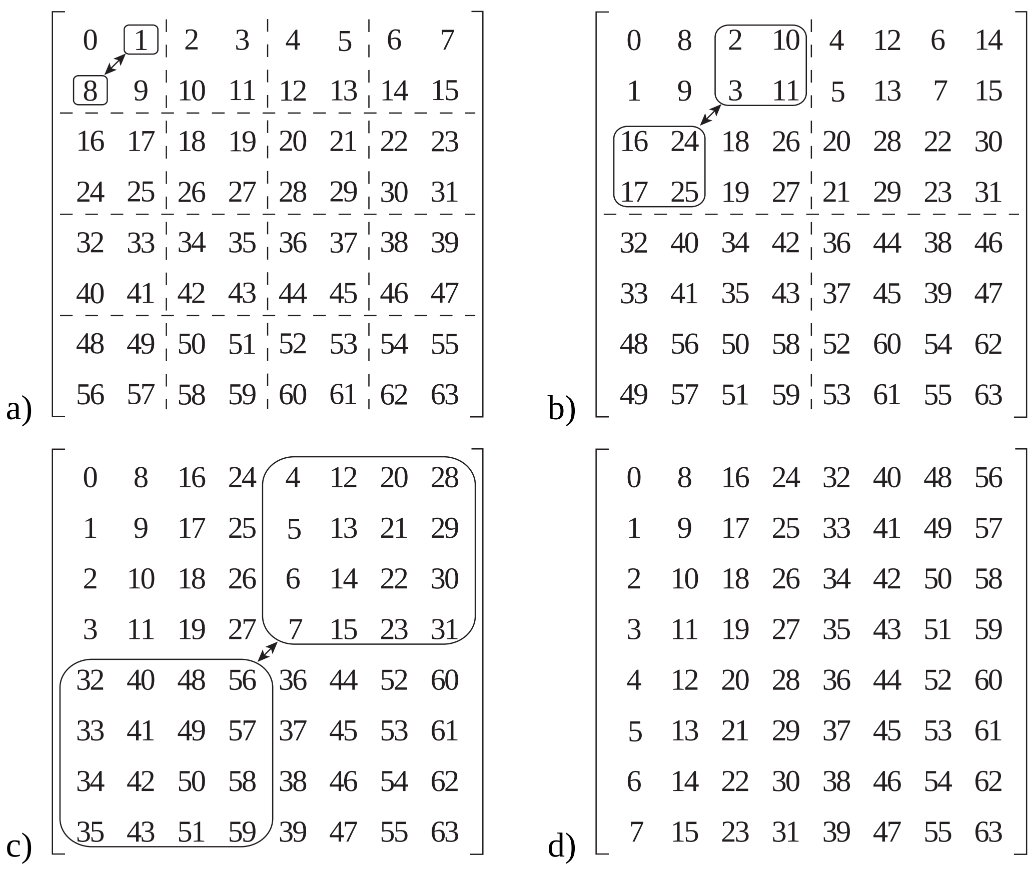


Fig. 12. *Example of the iterative* 8×8 *matrix transpose: (a) the first iteration, (b) the second iteration, (c) the third iteration, and (d) the transposed matrix.*

two and each submatrix is transposed. The resulting $N \times N$ matrix is then divided into submatrices of order four which are transposed as 2-by-2 matrices of 2-by-2 blocks. This operation is iterated until the entire $N \times N$ matrix is divided into four $N/2 \times N/2$ submatrices and the resulting 2-by-2 block matrix is transposed in order to have finally the transposed matrix. The described method is illustrated with the aid of an 8×8 matrix in Fig. 12, and the resulting column parallel 8×8 matrix transpose unit proposed by Carlach *et al.* in [6] is depicted in Fig. 13.

In general, the previously described iterative method for the matrix transpose of an $N \times N, N = 2^k$, matrix requires k steps. At each step, the transpose is performed within 2-by-2 blocks. This implies that the distance of data elements to be exchanged at a single step is constant. Therefore, the corresponding structure can be constructed by cascading the SEUs in increasing order of size: SEU_{N-1}, $\mathrm{SEU}_{2(N-1)}$, $\ldots$, $\mathrm{SEU}_{(N/2)(N-1)}$ as reported by Takala *et al.* in [108, 109]. The latency of the resulting generalized sequential matrix transpose network is $\sum_{i=0}^{k-1}(2^i)(N-1) = (N-1)^2$

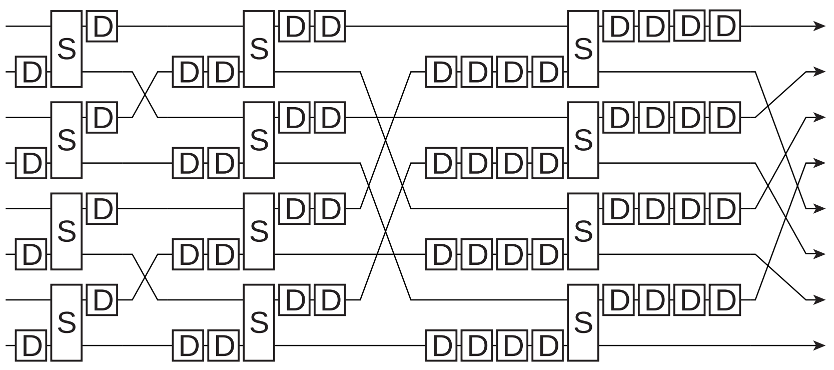


Fig. 13. *Block diagram of the column parallel matrix transpose* [6].

cycles, which equals to the maximum distance of single data element to be moved in the $N \times N$ matrix transpose in a vector form. The sequential 8×8 matrix transpose network is illustrated in Fig. 14. Let us remark that the diagram in the figure illustrates only the functionality, not the timing.

Although any one-dimensional structure can be exploited for the two-dimensional structure based on the row-column decomposition, let one two-dimensional structure, which has been proposed by Madisetti and Willson in [68], be introduced in more details. The basis for the structural derivation is the DCT algorithm presented by Chen *et al.* in [11]. When the transform matrix is decomposed into even and odd rows, the DCT as well as its inverse requires the computation of two 4×4 by 4×1 matrix-vector products. Such operations can be performed in parallel with two matrix-vector multipliers. The matrix-vector multiplier performing the product of the odd rows $\mathrm{MVM_o}$ requires four multipliers in addition to four accumulators as illustrated in Fig. 15(a). Instead, the product of the even rows can be computed with three multipliers and four accumulators.

In addition to the matrix-vector products, the DCT requires several additions and subtractions before the matrix-vector product whereas the IDCT requires the grouping of even and odd samples. These operations can be performed in a data reordering unit DRU depicted in Fig. 15(b). Correspondingly, the DCT requires the regrouping of even and odd coefficients after the matrix-vector product while the IDCT requires some additions and subtractions, which are performed in an inverse reordering unit IDRU. The objective has been to minimize the core area and keep input and output requirements simple with the single-pixel interfaces. The resulting structure is illustrated in Fig. 15(c).

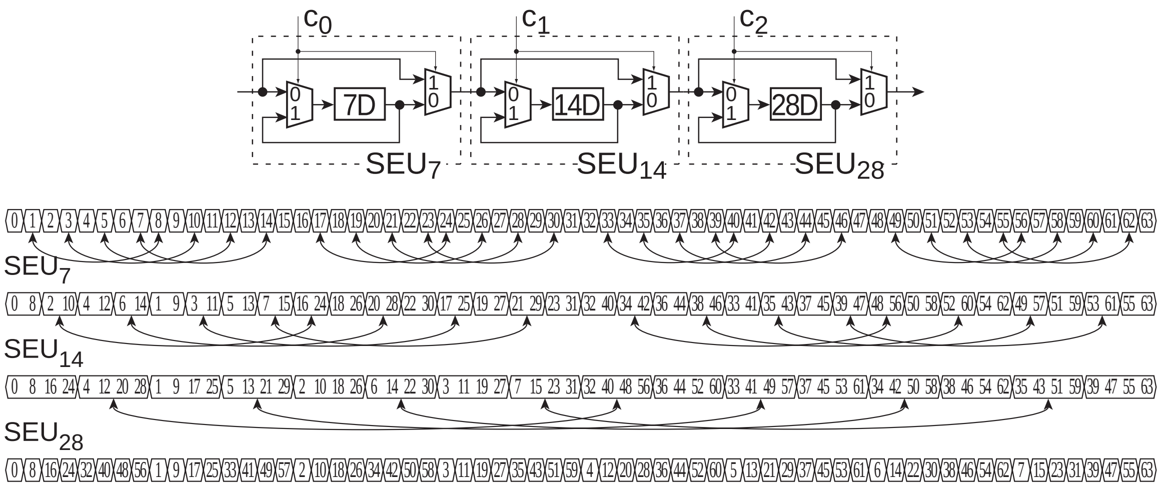


Fig. 14. *Block and timing diagrams of the sequential matrix transpose.*

The feature that makes the structure in Fig. 15(c) different from the other row-column structures is the exploitation of the parallelism and time multiplexed computation of the DCT over dimensions. In other words, for four clock cycles, the matrix-vector multipliers operate on a column at the input X and stores samples from Y into last-in-first-out (LIFO) buffer. For the next four cycles, multipliers operate on a row Y while samples from X are stored. Katayama *et al.* in [52] presented a corresponding structure but instead of the line parallel operation, the structure operates line-by-line. With such a modification, the number of accumulators is reduced from eight to two.

2.4.3 Direct Two-Dimensional Structures

Instead of applying the row-column decomposition, Lee *et al.* in [59] proposed a direct two-dimensional DCT structure. In order to reduce computational complexity with the aid of rotation techniques, real input values are mapped into complex numbers in the $N \times N$ DCT algorithm presented by Duhamel and Guillemot in [21]. Furthermore, the algorithm is modified for increasing the regularity and applying the vertical mapping. In any case, the resulting structure still possess high degree of parallelism; 16 values are processed in parallel, which reflect on the amount of hardware resources. On the other hand, the parallel processing provides a good throughput rate. Another highly parallel direct two-dimensional DCT structure is presented by Lim *et al.* in [63]. The structure is based on the implementation of the matrix multiplication as a systolic array. The interconnection complexity is minimized by using bit-serial interfaces between processing elements. However, the amount of hardware is huge.

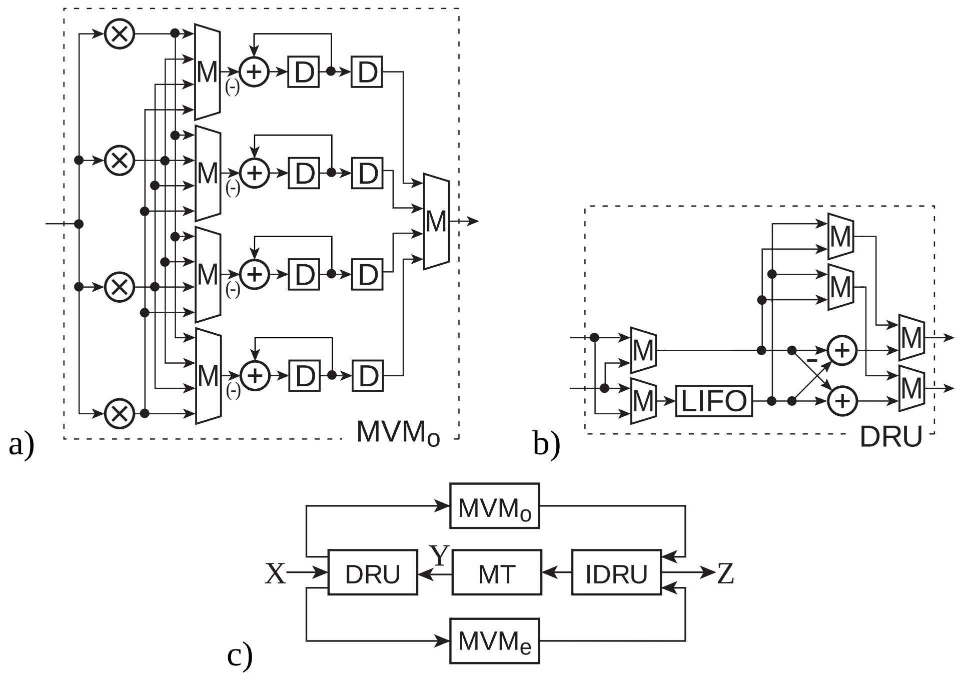


Fig. 15. *Block diagram of the* 8×8 *DCT/IDCT with matrix transpose: (a) matrix-vector multiplier performing the product of the odd rows (MVM*$_o$*), (b) data reordering unit (DRU), and (c) entire structure* [68]. *M: Multiplexer. LIFO: Last-in-first-out buffer. MVM*$_e$*: matrix-vector multiplier for even rows. IDRU: Inverse data reordering unit.*

Kwak and You in [55] proposed a VLSI implementation methodology for their constant geometry algorithms. Due to the very regular structure the linear mapping methods are applicable. Consequently, several structures with a flexible degree of parallelism can be constructed from the algorithms. The good regularity yields to high modularity, thus structures are made up of identical blocks. While the hardware resources can be optimized with respect to application requirements, the throughput rate can be adjusted to meet application requirements with the pipelining. In [35], Hsiao *et al.* extended correspondingly their sequential one-dimensional DCT structure presented in [34] to support the direct two-dimensional DCT. The structure resembles the one-dimensional structure depicted in Fig. 9 but each processing element is replicated having operands from N times more apart. Let us remark that the replicating multiplications can be combined and mapped still onto the single multiplier. Therefore, the number of multipliers in the two-dimensional transform is the same as in the one-dimensional transform. The resulting pipeline is illustrated in Fig. 16.

In principle, a unified structure can be constructed by providing additional data paths to reverse the data flow of the DCT pipeline for the IDCT computation as described,

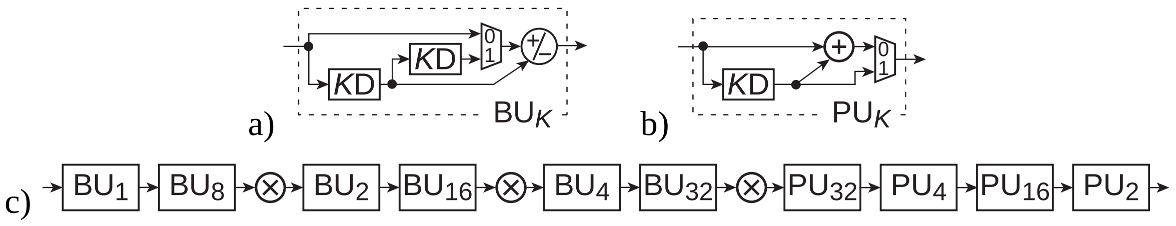


Fig. 16. *Block diagram of the fully sequential* 8×8 *direct DCT* [35]*: (a) radix-2 butterfly unit, (b) post-processing unit having operands K samples apart (PU*$_K$*), and (c) entire pipeline.*

e.g., in [12]. Such an approach will, however, introduce high routing cost and complicated control. Therefore, it is desirable that both the modes share a common data path wherever possible [68]. Consequently, the similarities in the computations of the DCT and the IDCT are typically utilized to develop a common structure. As discussed with the properties of the algorithms, the post-processing is a disadvantageous in the unified VLSI realizations; the post-processing in the forward transform results in a pre-processing in the inverse transform, thus in the unified approach both the pre- and post-processing need to be realized implying additional hardware cost as, e.g., in [34].

2.5 Summary

In this chapter, the DCT algorithms and structures have been surveyed for having bases for upcoming work. The survey has been carried out with a new viewpoint; the algorithms and structures are discussed with respect to their suitability for the pipeline computation. In addition, the survey has been extended to cover also the related work after the publication of the Rao and Yip's book [86] which has been exploited as a starting point. To conclude this chapter, let us summarize the features and limiting factors arisen during the survey.

The DCT algorithms based on the computation of other discrete trigonometric transforms introduce additional computational complexity. In order to achieve lower arithmetic complexity, the direct factorization of the DCT matrix has been considered. In general, the resulting DCT algorithms do not possess the regularity found, e.g., in Cooley-Tukey FFT algorithms. On the other hand, in the reported regular algorithms, the secant coefficients introduce round off errors with finite word length, the factorization contains anti-diagonal matrices or the irregularities related to the DCT are separated into pre- or post-processing stages.

When aiming at pipeline structures, the irregular algorithms restrict the efficient utilization of the linear mapping methods. Consequently, structures introduce complicated control or exploit application-specific solutions with high level of parallelism and without modularity. Instead, when using the regular algorithms as a basis, round off errors affect word length requirements and anti-diagonal matrices introduce irregular permutations requiring extra storage resources. Furthermore, pre- or post-processing stages introduce additional hardware when implementing the unified pipeline structure for the forward and inverse DCT. Moreover, the constant interconnection permutations in the constant geometry algorithms result in larger area than the permutations in the in-place algorithms where the size of the permutations is getting either smaller or larger stage by stage.

3. PERFECT SHUFFLE TOPOLOGY DCT ALGORITHMS

Since we are targeting at pipeline computation at data rate, our intention is to derive novel regular fast DCT algorithms which lend themselves for the vertical mapping. According to the previous survey, we should achieve the algorithms with comparable arithmetic complexity and reduced interconnection complexity. Therefore, our objective is not to minimize the arithmetic complexity but to derive algorithms possessing regularity in operational columns implying reduced control complexity. In addition, the interconnection permutations should be minimized and large anti-diagonal sparse matrices should be avoided in the algorithms for minimizing storage requirements. In order to avoid additional hardware in a unified pipeline structure, we avoid algorithms with the pre- or post-processing operations. Instead the irregularities are to be distributed over the operational columns in the signal flow graph. Moreover, the coefficients should be cosine coefficients for minimizing the round off errors.

In this chapter, the sparse matrix decomposition of the novel perfect shuffle topology algorithms for the one- and two-dimensional DCT is described. Before going into the details of the proposed fast algorithms, we provide the preliminaries needed during the derivation. Subsequently, the derivation of the algorithms for the one- and two-dimensional DCTs is described. The chapter is concluded by summarizing the benefits of the derived algorithms.

3.1 Preliminaries

The formulation used in this Thesis is based on tensor (or Kronecker) products denoted by $\otimes$. Operator $\oplus$ is used to denote the matrix direct sum [122]:

$$\bigoplus_{i=0}^{n} A_i = diag(A_0, A_1, \ldots, A_n). \tag{8}$$

For ordinary products, left evaluation is used, i.e.,

$$\prod_{i=0}^{n} A_i = A_0 \cdot A_1 \cdot A_2 \cdot \ldots \cdot A_n. \tag{9}$$

The floor function and modulus operation are denoted by $\lfloor \cdot \rfloor$ and $\bmod$, respectively. The identity matrix of order N is denoted by I_N and the anti-diagonal identity matrix of order N by $\bar{I}_N$, i.e.,

$$[\bar{I}_N]_{mn} = \begin{cases} 1, & m = N - n - 1 \\ 0, & \text{otherwise} \end{cases}, n, m = 0, 1, \ldots, N-1. \tag{10}$$

Permutation matrices are used in the formulation for reordering the data arrays. In the following, the permutation matrices are defined and three permutation types used in this Thesis are described. In addition, some properties related to these permutations are presented.

Definition 1. *A permutation matrix P_N is an $N \times N$ matrix with all elements either* 0 *or* 1, *with exactly one* 1 *at each row and column* [75].

The permutation matrices are orthogonal; if P_N is a permutation, then $P_N^{-1} = P_N^T$. The product of the permutation matrices is another permutation matrix. [75]

Stride permutations can be described like matrix transposes over sequentially represented data [25]. In stride-by-R permutation of an N-element vector, an $R \times (N/R)$ matrix is first constructed out of the vector in column wise and then returned back to the vector form in row wise. The stride-by-R permutation matrix of order N is denoted by $P_{N,R}$ and defined as [107]

$$[P_{N,R}]_{mn} = \begin{cases} 1, & \text{iff } n = (mR \bmod N) + \lfloor mR/N \rfloor \\ 0, & \text{otherwise} \end{cases}, m, n = 0, 1, \ldots, N-1. \tag{11}$$

As an example, the stride-by-R permutation reorders the data elements of an array $X = (x_0, x_1, \ldots, x_{N-1})^T$ as

$$P_{N,R}X = (x_0, x_R, x_{2R}, \ldots, x_{N-R+1}, x_1, x_{R+1}, \ldots, x_{N-1})^T. \tag{12}$$

A special case of the stride permutation is perfect shuffle, which interleaves the data elements in the first half of an array with the elements in the second half, i.e., perfect shuffle performs the stride-by-$N/2$ permutation, $P_{N,N/2}$, for an N-point array. Stride-by-2 permutation is the inverse permutation of perfect shuffle, thus the perfect shuffle permutation matrix can also be denoted by $P_{N,2}^T$.

The Hadamard permutation matrix of order N denoted by P_N^H is defined as

$$\left[P_N^H\right]_{mn} = \begin{cases} 1, & \text{iff } n = h_N(m) \\ 0, & \text{otherwise} \end{cases} \quad ,m,n = 0,1,\ldots,N-1 \tag{13}$$

where $h_N(i)$ is the Hadamard permutation function defined recursively as [119]

$$h_1(0) = 0;\ h_{2N}(2i) = h_N(i);\ h_{2N}(2i+1) = 2N-1-h_N(i),\ i = 0,1,\ldots,N-1. \tag{14}$$

Hadamard permutation reorders the elements in a vector X as

$$P_N^H X = \left(x_{h_N(0)}, x_{h_N(1)}, x_{h_N(2)}, \ldots, x_{h_N(N-1)}\right)^T. \tag{15}$$

The third permutation matrix to be exploited is defined as [106]

$$J_N = \left(I_2 \otimes P_{N/2,N/4}\right) P_{N,2}. \tag{16}$$

In this permutation, the odd elements in the first half of a vector are exchanged with the even elements of the last half of the vector.

In addition, let us define the following matrices:

$$F_1 = I_1;\ F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix};\ F_N = I_{N/2} \otimes F_2 \tag{17}$$

$$Q_1 = I_1;\ Q_2 = I_2;\ Q_4 = P_{4,2} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix};\ Q_N = I_{N/4} \otimes Q_4 \tag{18}$$

$$R_1 = I_1;\ R_2 = I_2;\ R_4 = Q_4 P_4^H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix};\ R_N = I_{N/4} \otimes R_4 \tag{19}$$

$$\tilde{I}_N = \text{diag}(-1^i),\ \ i = 0,1,\ldots,N-1. \tag{20}$$

In the following, some properties of tensor product and the previous permutations are presented. The proofs for the following theorems can be found, e.g., from [20, 25].

Theorem 1. *Consider matrices A, B, C, and D. The following properties hold true with the corresponding operations:*

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C \tag{21}$$

$$(A \otimes B)(C \otimes D) = (AC \otimes BD) \tag{22}$$

$$(A \otimes B)^T = A^T \otimes B^T \tag{23}$$

Corollary 1.

$$A_N \otimes B_N = (A_N \otimes I_N)(I_N \otimes B_N) \tag{24}$$

The proof for this is evident by referring to (22).

Theorem 2. *Factorizations of stride permutations*

$$P_{a,bc} = P_{a,b}P_{a,c} \tag{25}$$

$$P_{abc,c} = (P_{ac,c} \otimes I_b)(I_a \otimes P_{bc,c}) \tag{26}$$

Theorem 3. *Commutativity property of tensor product. If A_a and B_b are matrices of order a and b, respectively, then,*

$$A_a \otimes B_b = P_{ab,a}(B_b \otimes A_a)P_{ab,b} \tag{27}$$

Corollary 2. *Relation of stride-by-2 permutations*

$$P_{N,2}(I_2 \otimes P^T_{N/2,2}) = P_{4,2} \otimes I_{N/4} \tag{28}$$

Proof. By noting that $P_{N,2}P^T_{N,2} = I_N$ and applying (26) to $P_{N,2}$, we obtain $P_{N,2}(I_2 \otimes P^T_{N/2,2}) = (P_{4,2} \otimes I_{N/4})(I_2 \otimes P_{N/2,2})(I_2 \otimes P^T_{N/2,2}) = P_{4,2} \otimes I_{N/4}$. ∎

Corollary 3. *Factorization of stride-by-2 permutation*

$$P_{2^k,2} = \prod_{i=0}^{k-2} (I_{2^i} \otimes P_{4,2} \otimes I_{2^{k-i-2}}), \quad k > 1 \tag{29}$$

Proof. According to (26), the stride-by-2 permutation matrix of order $N, N = 2^k$, can be decomposed as $P_{N,2} = \left(P_{4,2} \otimes I_{N/4}\right)\left(I_2 \otimes P_{N/2,2}\right)$. By applying the decomposition to the term $P_{N/2,2}$, we obtain $P_{N,2} = \left(P_{4,2} \otimes I_{N/4}\right)\left(I_2 \otimes P_{4,2} \otimes I_{N/8}\right)\left(I_4 \otimes P_{N/4,2}\right)$. This recursion can be continued until the rightmost term is $I_{N/4} \otimes P_{4,2}$, i.e., the recursion can be applied $k-1$ times. ■

Corollary 4. *Factorization of perfect shuffle permutation*

$$P_{2^k,2^{k-1}} = \prod_{i=0}^{k-2} \left(I_{2^{k-i-2}} \otimes P_{4,2} \otimes I_{2^i}\right), \quad k > 1 \tag{30}$$

The proof is straightforward from Corollary 3 by noting that $P_{N,N/2} = P_{N,2}^T$.

The proofs for the following theorems are shown in [5].

Theorem 4. *Factorization of Hadamard permutation*

$$P_{2^k}^H = \prod_{i=0}^{k-2} \left[\left(I_{2^{k-2-i}} \otimes P_{2^{i+2},2}\right) R_{2^k}\right], \quad k > 1 \tag{31}$$

Definition 2. *The matrix A of order N is cross-diagonal of type 1 if it can be presented as*

$$A = D_1 + D_2 \bar{I} \tag{32}$$

where D_1 and D_2 are diagonal matrices of order N.

Definition 3. *The matrix A of order N is cross-diagonal of type 2 if it can be presented as*

$$A = D_1 + D_2 \bar{P} \tag{33}$$

where $\bar{P} = 0 \oplus \bar{I}_{N-1}$.

Theorem 5. *Let A_N be a type 1 cross-diagonal matrix of order $N = 2^k, k \geq 2$. Then, it can be represented as*

$$A_N = R_N P_{N,2}^T \begin{pmatrix} A_{N/2}^1 & 0 \\ 0 & A_{N/2}^2 \end{pmatrix} P_{N,2} R_N \tag{34}$$

where $A_{N/2}^1$ and $A_{N/2}^2$ are cross-diagonal matrices of type 1.

Theorem 6. *Let A_N be a type 2 cross-diagonal matrix of order $N = 2^k, k \geq 2$. Then, it can be represented as*

$$A_N = P_{N,2}^T \begin{pmatrix} A_{N/2}^2 & 0 \\ 0 & A_{N/2}^1 \end{pmatrix} P_{N,2} \tag{35}$$

where $A_{N/2}^1$ is a cross-diagonal matrix of type 1 and $A_{N/2}^2$ is a cross-diagonal matrix of type 2.

The fast algorithms for DCT derived in this Thesis are based on the results presented in [119] where the proofs for the following theorems can be found from.

Theorem 7. *The DCT matrix of order $N = 2^k$, C_N^{II}, can be represented as*

$$C_N^{II} = \sqrt{\frac{2}{N}} D_N^* C_N^{(0)} P_N^H \tag{36}$$

where D_N^* is a scaling matrix

$$D_N^* = d(1) \oplus I_{N-1} \tag{37}$$

and the matrix $C_N^{(0)}$ is a Hadamard ordered DCT matrix defined as

$$C_J^{(i)} = L_J^{(i)} B_J \begin{pmatrix} C_{J/2}^{(2i)} & 0 \\ 0 & C_{J/2}^{(2i+1)} \end{pmatrix} \tag{38}$$

where $L_J^{(i)}$ and B_J are the following:

$$L_J^{(i)} = \begin{pmatrix} 1 & & & \\ & I_{J/2-1} & & 0 \\ & & d(N/J+i) & \\ & -\overline{I}_{J/2-1} & & 2d(N/J+i)I_{J/2-1} \end{pmatrix} \tag{39}$$

$$B_J = \begin{pmatrix} I_{J/2} & I_{J/2} \\ I_{J/2} & -I_{J/2} \end{pmatrix}. \tag{40}$$

The coefficients $d(i)$ are all cosines defined as

$$d(i) = \cos\left((h_K(t) + 1/2)\pi/2K\right), K = 2^{\lfloor \log_2(i) \rfloor}, t = i - K. \tag{41}$$

or they can also be generated recursively as

$$d(1) = \sqrt{0.5};\ \ d(2i) = \sqrt{0.5(1 + d(i))};\ \ d(2i+1) = \sqrt{0.5(1 - d(i))}. \tag{42}$$

Theorem 8. *The Hadamard ordered DCT matrix $C_{2^k}^{(0)}$ can be computed as*

$$C_{2^k}^{(0)} = \prod_{s=k-1}^{0} \bigoplus_{i=0}^{2^{k-s-1}-1} \left(L_{2^{s+1}}^{(i)} B_{2^{s+1}} \right). \tag{43}$$

3.2 One-Dimensional Transform

In order to derive an algorithm suitable for the vertical mapping, we exploit the decomposition of the DCT matrix with Hadamard ordered input in Theorem 7. Let us first consider the matrix $L_J^{(i)}$ of order J, $J = 2^k$, in (39) which is actually a cross-diagonal matrix of type 2, thus Theorem 6 applies. Recursive application of the theorem $k-3$ times results in the following decomposition:

$$L_J^{(i)} = G_J E_J'^{(i)} G_J^T \tag{44}$$

where G_J is defined as

$$G_{2^k} = \prod_{i=0}^{k-3} \left[(I_{2^{k-i}} \oplus R_{2^k - 2^{k-i}}) \left(I_{2^i} \otimes P_{2^{k-i},2}^T \right) \right] \tag{45}$$

and $E_J'^{(i)}$ is the following block diagonal matrix consisting of blocks of order 4

$$\begin{aligned} E_J'^{(i)} = & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & d(N/J+i) & 0 \\ 0 & -1 & 0 & 2d(N/J+i) \end{pmatrix} \\ & \oplus \left[I_{J/4-1} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 2d(N/J+i) & 0 \\ -1 & 0 & 0 & 2d(N/J+i) \end{pmatrix} \right]. \end{aligned} \tag{46}$$

The block diagonal matrix $E_J'^{(i)}$ consists of a cross-diagonal matrix of type 2 of order 4 in the top-left corner and a block diagonal of cross-diagonal matrices of type 1 of order 4, thus Theorem 6 can still be applied for reducing the order of the blocks as

$$L_J^{(i)} = G_J (I_4 \oplus R_{J-4})(I_{J/4} \otimes P_{4,2}^T) E_J^{(i)} (I_{J/4} \otimes P_{4,2})(I_4 \oplus R_{J-4}) G_J^T \tag{47}$$

where $E_J^{(i)}$ is

$$E_J^{(i)} = \begin{pmatrix} 1 & 0 \\ 0 & d(N/J+i) \end{pmatrix} \oplus \left[I_{J/2-1} \otimes \begin{pmatrix} 1 & 0 \\ -1 & 2d(N/J+i) \end{pmatrix} \right]$$
$$= \left[I_2 \oplus \left(I_{J/2-1} \otimes \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \right) \right]$$
$$\cdot \left[\begin{pmatrix} 1 & 0 \\ 0 & d(N/J+i) \end{pmatrix} \oplus \left(I_{J/2-1} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 2d(N/J+i) \end{pmatrix} \right) \right]. \tag{48}$$

By defining two matrices $M_J'^{(i)}$ and $D_J''^{(i)}$ as

$$M_J'^{(i)} = I_2 \oplus \left[I_{J/2-1} \otimes \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \right] \tag{49}$$

$$D_J''^{(i)} = \begin{pmatrix} 1 & 0 \\ 0 & d(N/J+i) \end{pmatrix} \oplus \left[I_{J/2-1} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 2d(N/J+i) \end{pmatrix} \right] \tag{50}$$

we may rewrite $L_J^{(i)}$ as

$$L_J^{(i)} = G_J(I_4 \oplus R_{J-4})Q_J M_J'^{(i)} D_J''^{(i)} Q_J (I_4 \oplus R_{J-4}) G_J^T. \tag{51}$$

Next we consider the matrix B_J in (40), which according to Theorem 3 can be rewritten as

$$B_J = (F_2 \otimes I_{J/4}) \otimes I_2 = P_{J,2}^T (I_2 \otimes F_2 \otimes I_{J/4}) P_{J,2}. \tag{52}$$

The matrix $(I_2 \otimes F_2 \otimes I_{J/4})$ is a block diagonal matrix with blocks of order $J/2$ where each block consists of four diagonal matrices as

$$I_2 \otimes F_2 \otimes I_{J/4} = \begin{pmatrix} I_{J/4} & I_{J/4} & & \\ I_{J/4} & -I_{J/4} & & \\ & & I_{J/4} & I_{J/4} \\ & & I_{J/4} & -I_{J/4} \end{pmatrix}. \tag{53}$$

This matrix can be reordered with $I_{J/2} \oplus R_{J/2}$ without any effect since the diagonal elements to be exchanged are identical (either 1 or -1). Therefore, we may rewrite B_J as

$$B_J = P_{J,2}^T (I_{J/2} \oplus R_{J/2})(I_2 \otimes F_2 \otimes I_{J/4})(I_{J/2} \oplus R_{J/2}) P_{J,2}. \tag{54}$$

Applying Theorem 3 to the term $(F_2 \otimes I_{J/4})$ results in

$$B_J = P_{J,2}^T (I_{J/2} \oplus R_{J/2})(I_2 \otimes P_{J/2,2}^T)(I_4 \otimes F_2 \otimes I_{J/8})(I_2 \otimes P_{J/2,2})(I_{J/2} \oplus R_{J/2}) P_{J,2}. \tag{55}$$

The term $(I_{J/4} \oplus R_{3J/4})$ can be again included since R_4 matrices are exchanging identical diagonal elements, thus

$$\begin{aligned} B_J =& P^T_{J,2}(I_{J/2} \oplus R_{J/2})(I_2 \otimes P^T_{J/2,2})(I_{J/4} \oplus R_{3J/4})(I_4 \otimes F_2 \otimes I_{J/8}) \\ & \cdot (I_{J/4} \oplus R_{3J/4})(I_2 \otimes P_{J/2,2})(I_{J/2} \oplus R_{J/2})P_{J,2}. \end{aligned} \tag{56}$$

The recursive application of this two phase addition of terms can be continued $k-3$ times, which leads into form

$$B_J = G_J(I_{J/4} \otimes F_2 \otimes I_2)G^T_J. \tag{57}$$

In order to simplify the innermost term, Theorem 3 provides us the following form:

$$B_J = G_J(I_{J/2} \otimes P^T_{4,2})(I_{J/2} \otimes F_2)(I_{J/2} \otimes P_{4,2})G^T_J = G_J Q_J F_J Q_J G^T_J. \tag{58}$$

Now the decompositions of L_J in (51) and B_J in (58) can be replaced into (43), which results in

$$C^{(0)}_{2^k} = \prod_{s=k-1}^{0} \left(U^{(s)}_{2^k} A'^{(s)}_{2^k} V^{(s)}_{2^k} \right) \tag{59}$$

where

$$V^{(s)}_{2^k} = \bigoplus_{i=0}^{2^{k-s-1}-1} \left(Q_{2^{s+1}} G^T_{2^{s+1}} \right) \tag{60}$$

$$U^{(s)}_{2^k} = \bigoplus_{i=0}^{2^{k-s-1}-1} \left[G_{2^{s+1}} \left(I_4 \oplus R_{2^{s+1}-4} \right) Q_{2^{s+1}} \right] \tag{61}$$

$$A'^{(s)}_{2^k} = \bigoplus_{i=0}^{2^{k-s-1}-1} \left[M'^{(i)}_{2^{s+1}} D''^{(i)}_{2^{s+1}} Q_{2^{s+1}} (I_4 \oplus R_{2^{s+1}-4}) Q_{2^{s+1}} F_{2^{s+1}} \right]. \tag{62}$$

3.2.1 Processing Columns

The block diagonal matrix $A'^{(s)}_N$ defines all the arithmetic operations performed at the sth processing column. The representation of these columns can be unified by using a binary-valued parametrization function defined as

$$\mu_n(i) = \begin{cases} 0, & i \bmod 2^n = 0 \\ 1, & i \bmod 2^n \neq 0 \end{cases}. \tag{63}$$

Hence, the processing column $A_N'^{(s)}$ can be represented as

$$A_N'^{(s)} = M_N^{(s)} D_N'^{(s)} H_N^{(s)} F_N \tag{64}$$

where

$$M_{2^k}^{(s)} = \bigoplus_{i=0}^{2^{k-1}-1} \begin{pmatrix} 1 & 0 \\ -\mu_s(i) & 1 \end{pmatrix} \tag{65}$$

$$D_{2^k}'^{(s)} = \bigoplus_{i=0}^{2^{k-1}-1} \begin{pmatrix} 1 & 0 \\ 0 & 2^{\mu_s(i)} d(2^{k-s-1} + \lfloor i/2^s \rfloor) \end{pmatrix} \tag{66}$$

$$H_{2^k}^{(s)} = \bigoplus_{i=0}^{2^{k-2}-1} (Q_4 R_4 Q_4)^{\mu_{s-1}(i)}. \tag{67}$$

3.2.2 Interconnection Permutations

According to (59), $C_N^{(0)}$ can be factorized into the processing columns interconnected with the permutations. The permutation between the processing columns $A_N^{(s-1)}$ and $A_N^{(s)}$ is defined as

$$\begin{aligned} V_{2^k}^{(s)} U_{2^k}^{(s-1)} &= \bigoplus_{i=0}^{2^{k-s-1}-1} \left(Q_{2^{s+1}} G_{2^{s+1}}^T\right) \bigoplus_{i=0}^{2^{k-s}-1} \left[G_{2^s} \left(I_4 \oplus R_{2^s-4}\right) Q_{2^s}\right] \\ &= \bigoplus_{i=0}^{2^{k-s-1}-1} \left[Q_{2^{s+1}} G_{2^{s+1}}^T \left(I_2 \otimes G_{2^s} \left(I_4 \oplus R_{2^s-4}\right) Q_{2^s}\right)\right] = \bigoplus_{i=0}^{2^{k-s-1}-1} P_{2^{s+1}}^P. \end{aligned} \tag{68}$$

In order to simplify the representation, it is sufficient to consider only the matrix P_N^P of order N, $N = 2^k$, which can be written as

$$\begin{aligned} P_N^P &= Q_N G_N^T \left[I_2 \otimes \left(G_{N/2} \left(I_4 \oplus R_{N/2-4}\right) Q_{N/2}\right)\right] \\ &= Q_N G_N^T \left(I_2 \otimes G_{N/2}\right) \left(I_2 \otimes \left(I_4 \oplus R_{N/2-4}\right)\right) Q_N. \end{aligned} \tag{69}$$

An example of such a 32-point interconnection permutation, P_{32}^P, is illustrated in Fig. 17(a) where Q_N is represented with the aid of relation $Q_4 = P_{4,2} = P_{4,2}^T$.

Let us first consider the term $G_N^T(I_2 \otimes G_{N/2})$ in (69). According to the definition of G_N in (45), this can represented as

$$\begin{aligned} G_N^T(I_2 \otimes G_{N/2}) = &\prod_{i=0}^{k-3} \left[\left(I_{2^{k-i-3}} \otimes P_{2^{i+3},2}\right)\left(I_{2^{i+3}} \oplus R_{2^k-2^{i+3}}\right)\right] \\ &\cdot \left(I_2 \otimes \prod_{i=0}^{k-4} \left[\left(I_{2^{k-i-1}} \oplus R_{2^{k-1}-2^{k-i-1}}\right)\left(I_{2^i} \otimes P_{2^{k-i-1},2}^T\right)\right]\right). \end{aligned} \tag{70}$$

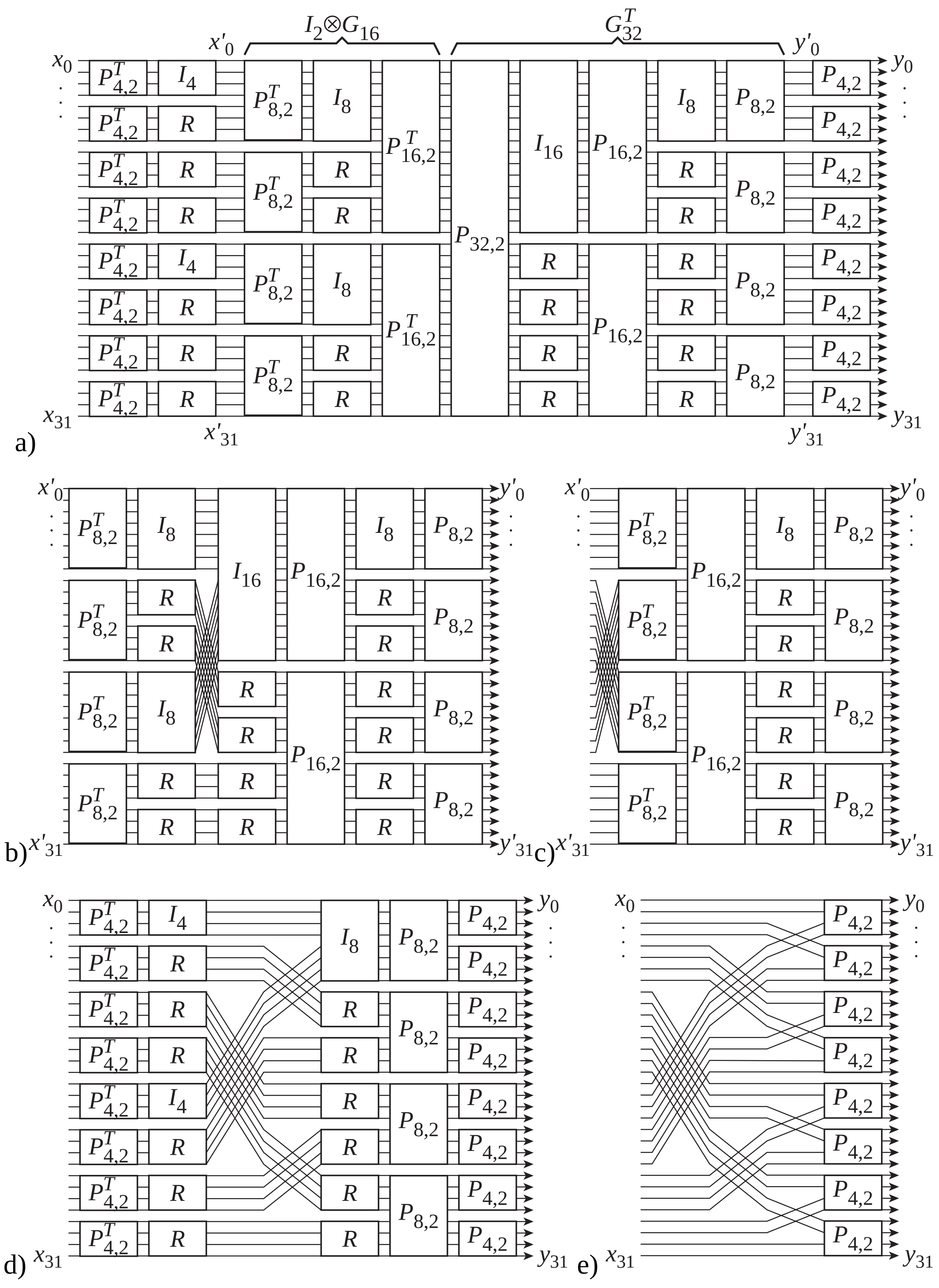


Fig. 17. *Simplification of the interconnection permutation: structure of 32-point interconnection permutation (a) in (69), (b) in (71), (c) after (73), (d) in (76), and (e) in (77).*

By taking two permutation matrices out of the products and applying Corollary 2, we obtain

$$\begin{aligned} G_N^T(I_2 \otimes G_{N/2}) = &\prod_{i=0}^{k-5} \left[\left(I_{2^{k-i-3}} \otimes P_{2^{i+3},2}\right)\left(I_{2^{i+3}} \oplus R_{2^k-2^{i+3}}\right)\right]\left(I_2 \otimes P_{N/2,2}\right) \\ &\cdot \left(I_{N/2} \oplus R_{N/2}\right)\left(P_{4,2} \otimes I_{N/4}\right)\left(I_2 \otimes (I_{N/4} \oplus R_{N/4})\right)\left(I_4 \otimes P_{N/4,2}^T\right) \\ &\cdot \left(I_2 \otimes \prod_{i=2}^{k-4}\left[\left(I_{2^{k-i-1}} \oplus R_{2^{k-1}-2^{k-i-1}}\right)\left(I_{2^i} \otimes P_{2^{k-i-1},2}^T\right)\right]\right). \end{aligned} \tag{71}$$

The resulting permutation in the 32-point example is depicted in Fig. 17(b).

Since $\left(I_2 \otimes (I_{N/4} \oplus R_{N/4})\right)$ is actually a block-diagonal matrix with four blocks of order $N/4$ and $\left(P_{4,2} \otimes I_{N/4}\right)$ performs exchange of middle blocks of order $N/4$ in a block diagonal matrix, the following is valid

$$\left(P_{4,2} \otimes I_{N/4}\right)\left(I_2 \otimes (I_{N/4} \oplus R_{N/4})\right) = \left(I_{N/2} \oplus R_{N/2}\right)\left(P_{4,2} \otimes I_{N/4}\right). \tag{72}$$

When using this in (71), the terms $\left(I_{N/2} \oplus R_{N/2}\right)$ cancel each other. Furthermore, $\left(I_4 \otimes P_{N/4,2}^T\right)$ is also a block diagonal matrix with identical blocks of order $N/4$ thus it commutes with $\left(P_{4,2} \otimes I_{N/2}\right)$, i.e.,

$$\left(P_{4,2} \otimes I_{N/2}\right)\left(I_4 \otimes P_{N/4,2}^T\right) = \left(I_4 \otimes P_{N/4,2}^T\right)\left(P_{4,2} \otimes I_{N/2}\right), \tag{73}$$

resulting the permutation shown in Fig. 17(c). When the commutativity in (73) is used in (71), we have a term $\left(I_2 \otimes P_{N/2,2}\right)\left(I_4 \otimes P_{N/4,2}^T\right)$, which according to Corollary 2 is $\left(I_2 \otimes P_{4,2} \otimes I_{N/8}\right)$ and, therefore,

$$\begin{aligned} G_N^T(I_2 \otimes G_{N/2}) = &\prod_{i=0}^{k-5} \left[\left(I_{2^{k-i-3}} \otimes P_{2^{i+3},2}\right)\left(I_{2^{i+3}} \oplus R_{2^k-2^{i+3}}\right)\right] \\ &\cdot \left(I_2 \otimes P_{4,2} \otimes I_{N/8}\right)\left(P_{4,2} \otimes I_{N/4}\right) \\ &\cdot \left(I_2 \otimes \prod_{i=2}^{k-4}\left[\left(I_{2^{k-i-1}} \oplus R_{2^{k-1}-2^{k-i-1}}\right)\left(I_{2^i} \otimes P_{2^{k-i-1},2}^T\right)\right]\right). \end{aligned} \tag{74}$$

By recursively applying the previous procedure, we obtain

$$\begin{aligned} G_N^T(I_2 \otimes G_{N/2}) = &\left(I_{N/8} \otimes P_{8,2}\right)\left(I_8 \oplus R_{N-8}\right)\left(I_{N/16} \otimes P_{4,2} \otimes I_4\right) \\ &\cdot \left(I_{N/32} \otimes P_{4,2} \otimes I_8\right) \ldots \left(I_2 \otimes P_{4,2} \otimes I_{N/8}\right)\left(P_{4,2} \otimes I_{N/4}\right). \end{aligned} \tag{75}$$

By replacing this term into (69), we end up a permutation illustrated in Fig. 17(d) with the following generalized representation:

$$\begin{aligned} P_N^P = &\left(I_{N/4} \otimes P_{4,2}\right)\left(I_{N/8} \otimes P_{8,2}\right)\left(I_8 \oplus R_{N-8}\right)\left(I_{N/16} \otimes P_{4,2} \otimes I_4\right) \\ &\cdot \left(I_{N/32} \otimes P_{4,2} \otimes I_8\right) \dots \left(I_2 \otimes P_{4,2} \otimes I_{N/8}\right)\left(P_{4,2} \otimes I_{N/4}\right) \\ &\cdot \left(I_2 \otimes \left(I_4 \oplus R_{N/2-4}\right)\right)\left(I_{N/4} \otimes P_{4,2}^T\right). \end{aligned} \tag{76}$$

According to Corollary 4, the permutation terms $\left(I_{N/16} \otimes P_{4,2} \otimes I_4\right)\left(I_{N/32} \otimes P_{4,2} \otimes I_8\right) \dots \left(P_{4,2} \otimes I_{N/4}\right)$ perform the perfect shuffle permutation of blocks of order four over $N/4$ blocks and, therefore, the term $\left(I_2 \otimes \left(I_4 \oplus R_{N/2-4}\right)\right)$ at the end of (76) can be moved next to the third term $(I_8 \oplus R_{N-8})$ in (76), simply by permuting the blocks of order four. Such a permutation results in term $(I_8 \oplus R_{N-8})$, thus the terms containing R matrices cancel each other and disappear. In similar fashion, we may move the term $\left(I_{N/4} \otimes P_{4,2}^T\right)$ at the end of (76) next to the term $\left(I_{N/8} \otimes P_{8,2}\right)$, which results in term $\left(I_{N/8} \otimes P_{4,2} \otimes I_2\right)$ according to Corollary 2. After this we may rewrite P_N^P with the aid of Corollary 4 as

$$\begin{aligned} P_N^P &= \left(I_{N/4} \otimes P_{4,2}\right)\left(I_{N/8} \otimes P_{4,2} \otimes I_2\right)\left(I_{N/16} \otimes P_{4,2} \otimes I_4\right) \dots \\ &\quad \cdot \left(I_2 \otimes P_{4,2} \otimes I_{N/8}\right)\left(P_{4,2} \otimes I_{N/4}\right) \\ &= \prod_{i=0}^{k-2} \left(I_{2^{k-i-2}} \otimes P_{4,2} \otimes I_{2^i}\right) = P_{N,N/2} = P_{N,2}^T, \end{aligned} \tag{77}$$

which is illustrated in Fig. 17(e).

With the aid of Corollary 4, we have found that the permutation between the processing columns is based on perfect shuffle permutation and, therefore, the permutation matrix between the processing columns $A_{2^k}^{(s)}$ and $A_{2^k}^{(s-1)}$ in (68) can be defined as

$$V_{2^k}^{(s)} U_{2^k}^{(s-1)} = \bigoplus_{i=0}^{2^{k-s-1}-1} P_{2^{s+1},2}^T. \tag{78}$$

3.2.3 Final Algorithm

Since $V_N^{(0)} = I_N$, the representation of the DCT matrix $C_N^{(0)}$ in (59) can be reduced with the aid of the simplified interconnection in (78) to

$$C_{2^k}^{(0)} = U_{2^k}^{(k-1)} \prod_{s=k-1}^{1} \left[A_{2^k}^{\prime(s)} \left(I_{2^{k-s-1}} \otimes P_{2^{s+1},2}^T\right)\right] A_{2^k}^{\prime(0)} \tag{79}$$

where $U_{2^k}^{(k-1)}$ represents the output permutation defined as

$$U_{2^k}^{(k-1)} = \prod_{i=0}^{k-2} \left[(I_{2^{k-i}} \oplus R_{2^k - 2^{k-i}}) \left(I_{2^i} \otimes P_{2^{k-i},2}^T \right) \right]. \tag{80}$$

The complete DCT in (36) contains a diagonal matrix D_N^* defined in (37), which has only a single non-trivial element at the top-left corner. Let us remark that matrices $U_N^{(k-1)}$ as a permutation matrix and $M_{2^k}^{(k-1)}$ defined in (65) contain only single 1 in the first row and in the first column, at the top-left corner and thus, D_N^* commutes with them. This implies that $D_{2^k}^*$ can be included into the diagonal matrix $D_{2^k}^{\prime(k-1)}$ defined in (66) and we accomplish that with the aid of a binary-valued parametrization function $\tau_i(s)$,

$$\tau_i(s) = \begin{cases} 0, & s = i \\ 1, & s \neq i \end{cases}, \tag{81}$$

and defining a diagonal matrix $D_N^{(s)}$ based on (66) as

$$D_{2^k}^{(s)} = \operatorname{diag}\left(g_k(i,s)\right), \quad i = 0, 1, \ldots, 2^k - 1 \tag{82}$$

$$g_k(i,s) = \left(2^{\mu_s(\lfloor i/2 \rfloor)} d(2^{k-s-1} + \lfloor i/2^{s+1} \rfloor) \right)^{f_k(i,s)} \tag{83}$$

$$f_k(i,s) = (i \bmod 2) + (1 - \tau_0(i))(1 - \tau_{k-1}(s)). \tag{84}$$

Consequently, a new matrix representation for the sth processing column, $A_N^{(s)}$, can be defined as

$$A_N^{(s)} = M_N^{(s)} D_N^{(s)} H_N^{(s)} F_N. \tag{85}$$

Finally, the entire DCT matrix can be formulated as

$$C_{2^k}^{\mathrm{II}} = \sqrt{\frac{2}{2^k}} U_{2^k}^{(k-1)} \prod_{s=k-1}^{1} \left[A_{2^k}^{(s)} \left(I_{2^{k-s-1}} \otimes P_{2^{s+1},2}^T \right) \right] A_{2^k}^{(0)} P_{2^k}^H. \tag{86}$$

In order to emphasize the regularity of the proposed perfect shuffle topology algorithm, the signal flow graph of the 16-point DCT is illustrated in Fig. 18 where the normalization factors are not included. In general, the regular perfect shuffle topology algorithm in (86) yields arithmetic complexity of $N\log(N)/2 + 1$ multiplications and $3N\log(N)/2 - N + 1$ additions for an N-point DCT, $N = 2^k$, including the scaling factor b_m but without the normalization factor $\sqrt{2/N}$ in (2).

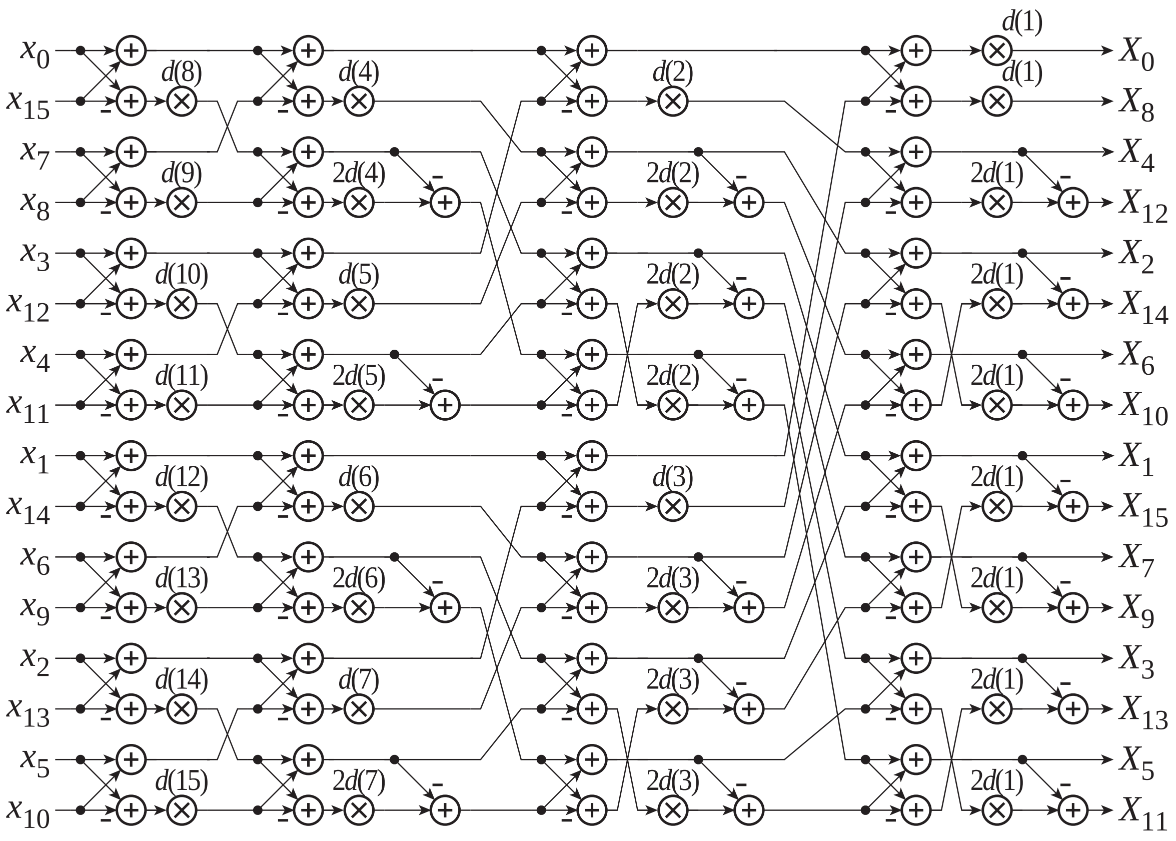


Fig. 18. *Signal flow graph of the regular perfect shuffle topology DCT in (86).*

3.3 Two-Dimensional Transform

As discussed in the previous chapter, there are two principal methods for deriving fast algorithms for the two-dimensional DCT: a row-column decomposition and direct computation. In the ensuing two sections, both the row-column decomposition and direct computation are used to derive two-dimensional DCT algorithms based on the proposed one-dimensional perfect shuffle topology algorithm.

3.3.1 Row-Column Decomposition

Due to the fact that the DCT transform matrix is separable, the DCT of an $N \times N$ matrix x defined as

$$X = C_N^{\mathrm{II}} x {C_N^{\mathrm{II}}}^T \tag{87}$$

can be reduced into a one-dimensional transform by concatenating the columns of the two-dimensional input data matrix x; i.e., x is represented as an array, resulting in an N^2-point one-dimensional array $\hat{x}$. Then the two-dimensional transform can be computed as

$$\hat{X} = \left(C_N^{\mathrm{II}} \otimes C_N^{\mathrm{II}}\right) \hat{x} = C_{N\times N}\hat{x} \tag{88}$$

where $\hat{X}$ is the transformed data matrix in the array format and $C_{N\times N}$ is the corresponding two-dimensional transform matrix of order N^2. By utilizing Corollary 1 and the commutativity property of tensor product in Theorem 3, $C_{N\times N}$ can be written as

$$C_{N\times N} = \left(C_N^{\mathrm{II}} \otimes I_N\right)\left(I_N \otimes C_N^{\mathrm{II}}\right) = P_{N^2,N}\left(I_N \otimes C_N^{\mathrm{II}}\right) P_{N^2,N}\left(I_N \otimes C_N^{\mathrm{II}}\right). \tag{89}$$

If a fast algorithm exists for computing the one-dimensional DCT, the algorithm in (89) is a fast algorithm for the two-dimensional DCT. According to the definition of the stride permutation in [25], $P_{N^2,N}$ performs the transpose of an $N \times N$ matrix arranged in a vector form and, therefore, (89) describes the traditional row-column decomposition of the two-dimensional DCT.

The fast algorithms for the one-dimensional DCT can often be represented as

$$C_N^{\mathrm{II}} = P_N^O K_N P_N^I \tag{90}$$

where K_N represents a transform kernel matrix of order N containing all the arithmetic operations and P_N^I and P_N^O are input and output permutation matrices of order N, respectively. This implies that in the two-dimensional transform in (89), the permutation between the consecutive one-dimensional DCT kernels consists of the output permutation, matrix transpose, and input permutation. Such a permutation increases the complexity of hardware implementations considerably in the cases where a programmable processor with efficient addressing modes could take care of data permutations. However, when a one-dimensional algorithm having the form of (90) is used in the two-dimensional transform, the transform matrix $C_{N\times N}$ can be formulated as

$$\begin{aligned} C_{N\times N} &= \left(P_N^O \otimes P_N^O\right)\left(K_N \otimes K_N\right)\left(P_N^I \otimes P_N^I\right) \\ &= \left(P_N^O \otimes P_N^O\right) P_{N^2,N}\left(I_N \otimes K_N\right) P_{N^2,N}\left(I_N \otimes K_N\right)\left(P_N^I \otimes P_N^I\right). \end{aligned} \tag{91}$$

This implies that in the two-dimensional transform, the input permutation from the consecutive one-dimensional transforms can be combined as a single permutation matrix $P_{N\times N}^I$ of order N^2. Similarly the output permutations and additional matrix transpose can also be combined as a matrix $P_{N\times N}^O$ of order N^2. Therefore, the entire two-dimensional DCT can be interpreted to consist of the matrix transpose between two consecutive transform kernels and the input and output permutations as

$$C_{N\times N} = P^{O}_{N\times N}\left(I_N \otimes K_N\right) P_{N^2,N}\left(I_N \otimes K_N\right)\left(P^{I}_{N\times N}\right). \tag{92}$$

Such an arrangement provides efficiency in hardware accelerator type of implementations where the permutation can be performed with a programmable processor and only the transform kernels and matrix transpose need to be realized in hardware.

When the one-dimensional DCT algorithm in (86) is used as a basis for the row-column approach, the two-dimensional DCT realization yields arithmetic complexity of $N(N\log(N)+2)$ multiplications and $N(3N\log(N)-2N+2)$ additions for an $N\times N$ DCT, $N=2^k$ including the DC factor b_m but without the normalization factor $\sqrt{2/N}$ in (2).

3.3.2 Direct Computation

The one-dimensional DCT algorithm in (86) can be used as a basis to derive a regular algorithm for the direct computation of the two-dimensional DCT. The two-dimensional transform matrix $C_{N\times N}$ can be derived by substituting the fast algorithm in (86) for C_N^{II} in (88). This results in

$$\begin{aligned} C_{2^k\times 2^k} =& 2^{1-k}\left[U_{2^k}\prod_{s=k-1}^{1}\left(Q_{2^k}^{(s)}\left(I_{2^{k-s-1}}\otimes P^T_{2^{s+1},2}\right)\right)Q_{2^k}^{(0)}P_{2^k}^H\right] \\ &\otimes\left[U_{2^k}\prod_{s=k-1}^{1}\left(Q_{2^k}^{(s)}\left(I_{2^{k-s-1}}\otimes P^T_{2^{s+1},2}\right)\right)Q_{2^k}^{(0)}P_{2^k}^H\right], \end{aligned} \tag{93}$$

which by applying Theorem 1 can be formulated as

$$\begin{aligned} C^{\mathrm{II}}_{2^k\times 2^k} =& 2^{1-k}\left(U_{2^k}^{(k-1)}\otimes U_{2^k}^{(k-1)}\right) \\ &\cdot\prod_{s=k-1}^{1}\left[\left(A_{2^k}^{(s)}\otimes A_{2^k}^{(s)}\right)\left(I_{2^{k-s-1}}\otimes P^T_{2^{s+1},2}\otimes I_{2^{k-s-1}}\otimes P^T_{2^{s+1},2}\right)\right] \\ &\cdot\left(A_{2^k}^{(0)}\otimes A_{2^k}^{(0)}\right)\left(P_{2^k}^H\otimes P_{2^k}^H\right). \end{aligned} \tag{94}$$

The matrix representation of the processing column in the two-dimensional DCT can be formulated with the aid of Corollary 1 as

$$\begin{aligned} A^{(s)}_{N\times N} =& A_N^{(s)}\otimes A_N^{(s)} = \left(M_N^{(s)}\otimes M_N^{(s)}\right)\left(H_N^{(s)}\otimes H_N^{(s)}\right)\left(D_N^{(s)}\otimes D_N^{(s)}\right)\left(F_N\otimes F_N\right) \\ =&\left(M_N^{(s)}\otimes I_N\right)\left(I_N\otimes M_N^{(s)}\right)\left(H_N^{(s)}\otimes I_N\right)\left(I_N\otimes H_N^{(s)}\right)D^{(s)}_{N\times N}\left(F_N\otimes I_N\right)F_{N^2} \end{aligned} \tag{95}$$

where $D^{(s)}_{N\times N}$ can be defined with the aid of the coefficients $g_k(i)$ in (84) as

$$\begin{aligned} D^{(s)}_{2^k\times 2^k} &= D^{(s)}_{2^k} \otimes D^{(s)}_{2^k} \\ &= \operatorname{diag}\left(g_k(i \bmod 2^k, s)\, g_k(\lfloor i/2^k \rfloor, s)\right), \quad i = 0,1,\ldots,2^{2k}-1. \end{aligned} \tag{96}$$

With the same analogy, the permutation between the processing columns s and $s-1$ can be written as

$$I_{2^{k-s-1}} \otimes P^T_{2^{s+1},2} \otimes I_{2^{k-s-1}} \otimes P^T_{2^{s+1},2} = \left(I_{2^{k-s-1}} \otimes P^T_{2^{s+1},2} \otimes I_{2^k}\right)\left(I_{2^{2k-s-1}} \otimes P^T_{2^{s+1},2}\right). \tag{97}$$

Similarly, the input and output permutation matrices of order N^2 denoted by $P^I_{N\times N}$ and $P^O_{N\times N}$, respectively, are formulated as

$$P^I_{2^k\times 2^k} = \left(P^H_{2^k} \otimes I_{2^k}\right)\left(I_{2^k} \otimes P^H_{2^k}\right) \tag{98}$$

$$P^O_{2^k\times 2^k} = \left(U^{(k-1)}_{2^k} \otimes I_{2^k}\right)\left(I_{2^k} \otimes U^{(k-1)}_{2^k}\right). \tag{99}$$

Finally, with the aid of (95)–(99), we can formulate the matrix representation of the fast algorithm for the two-dimensional DCT as

$$\begin{aligned} C^{\mathrm{II}}_{2^k\times 2^k} =& 2^{1-k} P^O_{2^k\times 2^k} \prod_{s=k-1}^{1} \left[A^{(s)}_{2^k\times 2^k}\left(I_{2^{k-s-1}} \otimes P^T_{2^{s+1},2} \otimes I_{2^k}\right)\left(I_{2^{2k-s-1}} \otimes P^T_{2^{s+1},2}\right)\right] \\ & \cdot A^{(0)}_{2^k\times 2^k} P^I_{2^k\times 2^k}. \end{aligned} \tag{100}$$

As an example, the signal flow graph of the resulting regular perfect shuffle topology algorithm for the 8×8 DCT without the normalization factor is depicted in Fig. 19. This example illustrates that the structure of the two-dimensional signal flow graph follows the structure of the previously proposed one-dimensional algorithm but all the processing and permutation columns are followed by another similar in-place column where the operation is over distributed data. The only exception is the columns with multiplications. In this sense, the algorithm is a hybrid; interconnection topology is perfect shuffle but the processing columns are in in-place configuration.

The two-dimensional DCT algorithm defined in (100) yields arithmetic complexity of $N(5N\log(N)/4+1)-1$ multiplications and $N(3N\log(N)-2N+2)$ additions for an $N\times N$ DCT, $N=2^k$, including the scaling factor b_m but without the normalization factor 2^{1-k}. It should be noted that the non-trivial coefficients are localized into $\log(N)$ diagonal matrices of order N^2, which implies that the signal flow graph

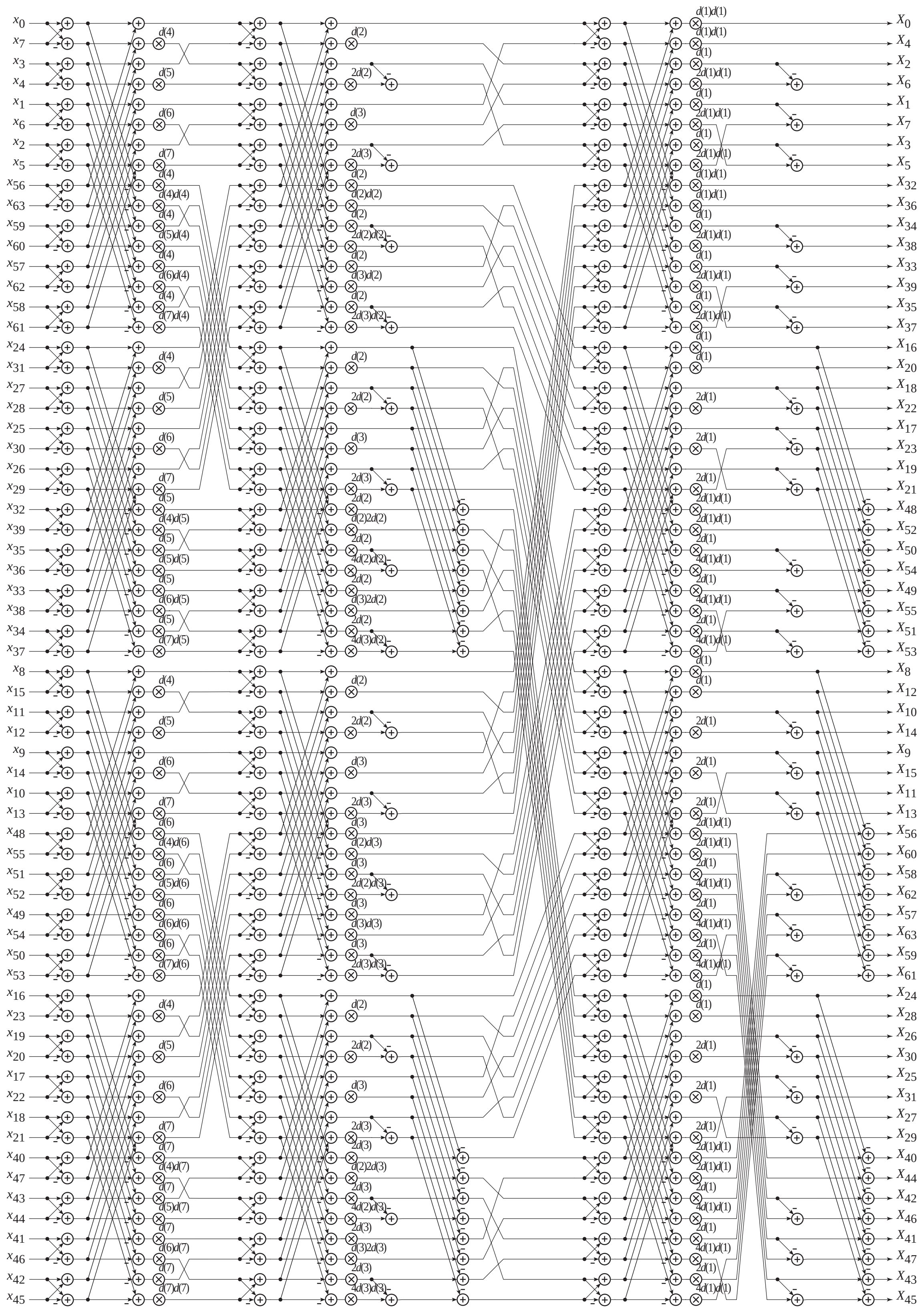


Fig. 19. *Signal flow graph of the regular perfect shuffle topology* 8×8 *DCT in (100).*

contains $\log(N)$ columns of multiplications, i.e., as many as in the previous one-dimensional transform, while in the row-column approach there are $2\log(N)$ columns of multiplications. This provides advantage in pipeline architectures, as described in the next chapter of this Thesis.

3.4 Summary

In the derived regular perfect shuffle topology algorithms, a straightforward design strategy has been to exploit the advantages and get rid of or minimize the effects of the drawbacks arisen in the survey of the related work. The algorithms possess regularity in the operational columns implying reduced control complexity. Although the irregularities related to the DCT are distributed between the butterfly columns they are localized into rather small node functions. Simultaneously, the pre- and post-processing stages as well as the large anti-diagonal sparse matrices are avoided. The regular interconnections with smaller and smaller permutations from column to column reduces the complexity of data permutation. All the previous features minimize the need for storage and provide area-efficiency when targeting at the single-rate pipeline structures.

On the other hand, the algorithms can be interpreted to combine the regular in-place and constant geometry algorithms. The interconnection topology is perfect shuffle as in the constant geometry algorithms but the size of the permutation is smaller from column to column as in the in-place algorithms. The relation to the in-place and constant geometry algorithms is emphasized especially in the direct two-dimensional DCT algorithms in which interconnection topology is perfect shuffle but the processing columns are in in-place configuration. In this sense, the algorithms are hybrids.

4. PIPELINE STRUCTURES FOR PERFECT SHUFFLE TOPOLOGY DCT

The regularity in all the novel perfect shuffle topology DCT algorithms derived in the previous chapter allows efficient utilization of temporal parallelism. Therefore, in the following, our purpose is to reduce the dimensionality of the signal flow graph by applying vertical mapping, i.e., the two-dimensional signal flow graph is collapsed or folded into a one-dimensional data path. In other words, each node found in the matrix factorizations in (86) and (100) is mapped onto a corresponding sequential unit with appropriate control signals resulting in new cascaded or pipeline structures.

The proposed DCT algorithms consist of the input permutation, processing columns interconnected with the perfect shuffle permutations, and output permutation. In this chapter, these processing and permutation columns are first mapped onto basic processing and permutation units. Then, the resulting processing and permutation units are cascaded for constructing the new pipeline structures. In order to prove the potentiality and feasibility of the structures, the unified 8×8 DCT/IDCT pipeline structure is compared to other reported structures and its demonstration implementation is described. Finally, the benefits of the developed structures are summarized.

4.1 Basic Processing Units

Basic processing units for the pipeline computation of an algorithm are obtained by defining the minimum set of data-dependent arithmetic operations in each operational column. The processing columns $A_N^{(s)}$ in the proposed algorithm contain operational columns of butterflies F_N, multiplications $D_N^{(s)}$, and local subtractions $M_N^{(s)}$.

In the operational column F_N, the principal operation is a real-valued radix-2 butterfly computation which can be realized with a single adder/subtracter as depicted in Fig. 20(a) [34]. This requires that input operands x_0 and x_1 are stored into registers

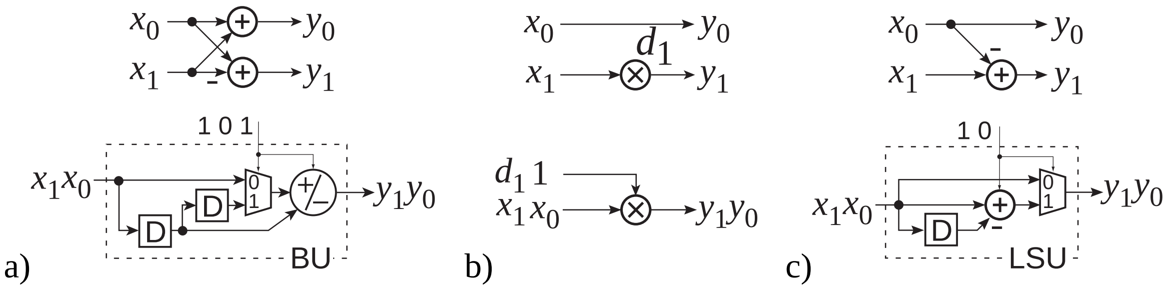


Fig. 20. *Block diagrams of the basic data processing units: (a) butterfly unit (BU), (b) multiplier, and (c) local subtraction unit (LSU).*

for computing both a sum and difference at consecutive clock cycles. When x_0 is read from the first delay register, x_1 is entering the unit and, therefore, the addition $y_0 = x_0 + x_1$ can be performed and the result is directed to the output. In the next clock cycle, x_0 proceeds to the second delay register and x_1 is read from the first one thus the subtraction $y_1 = x_0 - x_1$ can be performed. The adder/subtracter in this structure is fully utilized at the expense of additional delay registers. Due to the operand storage the latency of the butterfly unit (BU) is one.

The basic operation in the column $D_N^{(s)}$ is multiplication and especially, if the trivial multiplication with one is included, each operand can be interpreted to be multiplied. Therefore, the implementation is a single multiplier as illustrated in Fig. 20(b). Since all the intermediate values are fed through a multiplier, the signal levels can be scaled arbitrary allowing the efficient utilization of numeric range without additional hardware cost.

The column $M_N^{(s)}$ is parametrized and contains irregular subtractions performed over the neighboring data samples in a sequence as depicted in Fig. 20(c). These operations can be realized with a local subtraction unit (LSU). The operands, which do not need any operation, are directed to the output through a multiplexer. When the subtraction is needed, the first operand x_0 is forwarded to output without computation, i.e., $y_0 = x_0$ and, at the same time, stored into a register. In the next clock cycle, when x_1 is entering the unit, the operands x_0 and x_1 can be subtracted and the difference $y_1 = x_1 - x_0$ is directed to the output. Since both the operands are available for the subtraction when the result is needed, the LSU does not introduce additional latency.

Let us remark that the local subtraction in Fig. 20(c) can be interpreted as a half of the butterfly operation. Therefore, the local subtraction can be realized with the previous

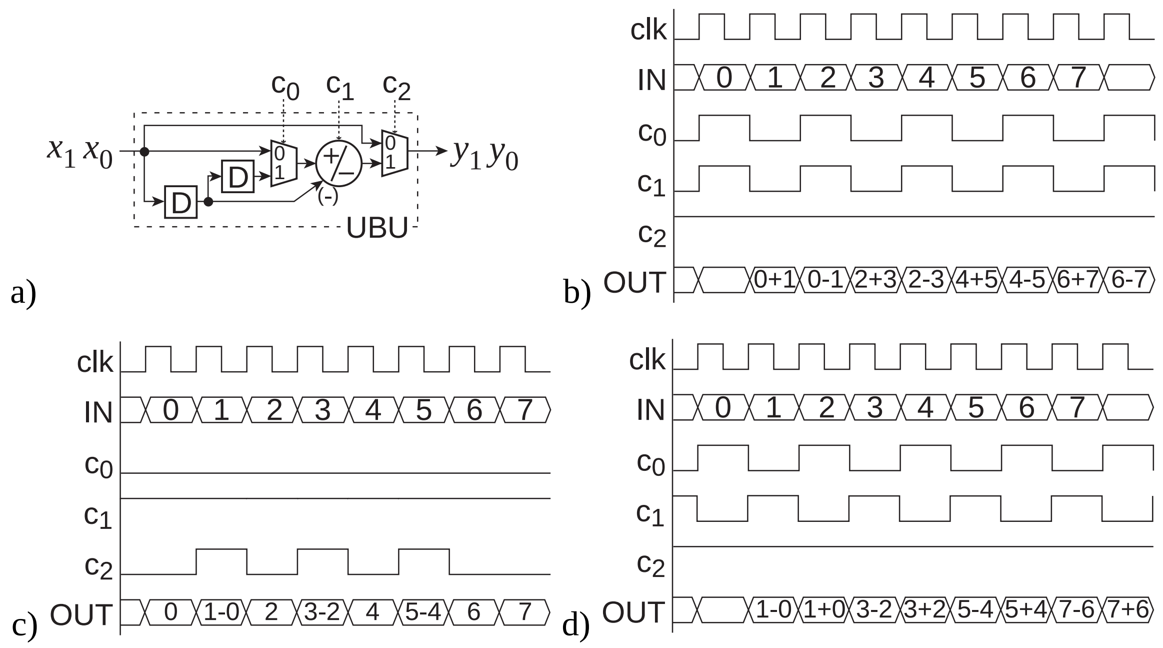
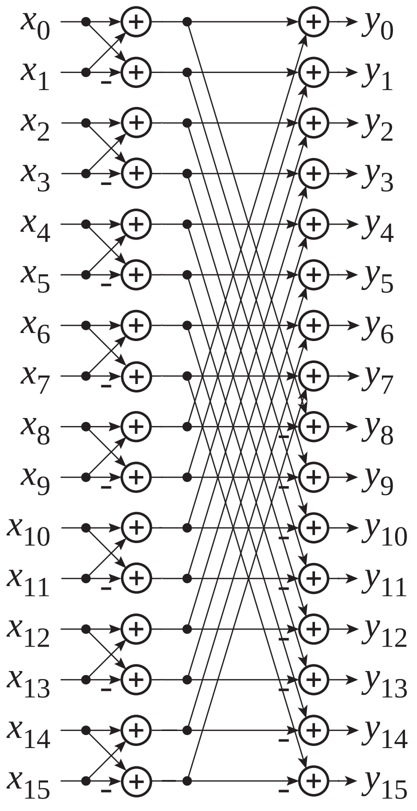


Fig. 21. *Unified butterfly unit: (a) block diagram and timing diagrams when performing (b) butterfly, (c) local subtraction, and (d) flipped butterfly.*

butterfly unit but an additional multiplexer is needed to bypass the arithmetic operation. The resulting multifunction arithmetic unit realizing the butterfly operations and local subtractions is referred to as a unified butterfly unit (UBU), which is illustrated with the aid of timing diagrams in Fig. 21. Note, that a flipped butterfly operation illustrated in Fig. 21(d) is exploited in the inverse DCT, and it will be introduced later in this chapter.

Processing units for computing the two-dimensional DCT can be constructed in a similar fashion from the two-dimensional DCT algorithm in (100). The only difference is that in the $N \times N$ DCT, the operational columns have a form $(X_N \otimes I_N)(I_N \otimes X_N)$, i.e., each column of the arithmetic operations is followed by another column with operands N times more apart. Consequently, such two-dimensional columns can be mapped onto cascade of two basic units where the shift registers in the latter unit are replaced with N times longer shift registers. As an example, a direct 8-point two-dimensional butterfly operation corresponding $(F_2 \otimes I_8)(I_8 \otimes F_2)$ is illustrated in Fig. 22(a). The first part of the direct two-dimensional unified butterfly unit operates exactly as the unit in Fig. 21(a) while the second part computes same butterfly operation but over operands eight elements apart. The block diagram of the direct two-dimensional unified butterfly unit UBU^2 is depicted in Fig. 22(b).

a)


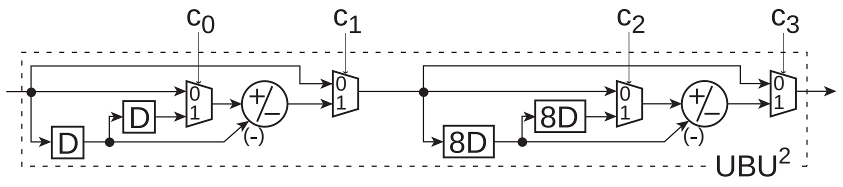
b)


Fig. 22. *Direct two-dimensional butterfly unit: (a) signal flow graph and (b) block diagram.*

4.2 Basic Permutation Units

Similarly to the columns of the arithmetic operations, data permutations in the algorithms need to be mapped onto sequential units, i.e., sequential permutation networks. The processing columns $A_N^{(s)}$ contain local permutation columns $H_N^{(s)}$. In addition, the processing columns are interconnected with the perfect shuffle permutations $P_{N,2}^T$. Furthermore, if input and output data sequences are required in natural order, the input and output permutations need to be realized. All the permutation networks are based on the shift-exchange units shown in Fig. 11(b) [98].

The basic permutation in the column $H_N^{(s)}$ is a local 4-point permutation illustrated in Fig. 23(a). Let us remark that, in the proposed algorithms, this local permutation $H_N^{(s)}$

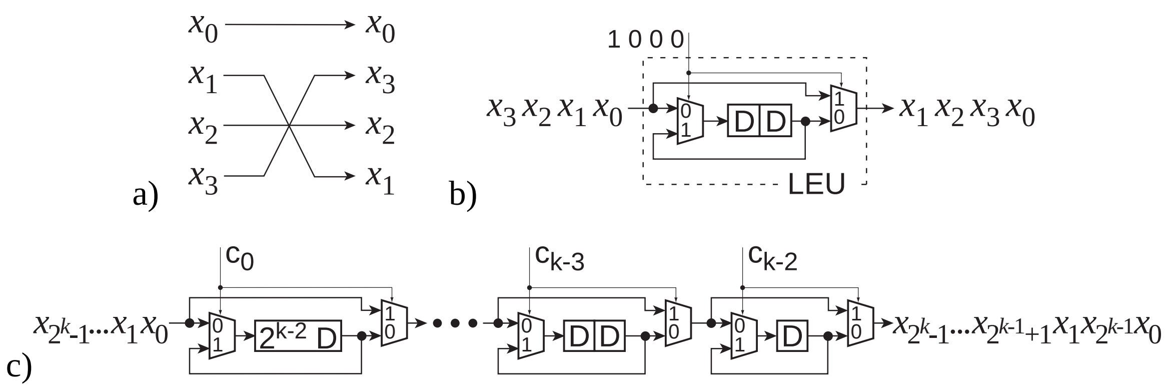


Fig. 23. Block diagrams of the basic data permutation units: (a) signal flow graph of local exchange operation, (b) local exchange unit (LEU), and (c) sequential 2^k*-point perfect shuffle permutation network.*

in (67) always exchanges the elements two elements apart regardless of the transform size. Such an exchange can be performed effectively with a local exchange unit (LEU), which is a single SEU_2 as shown in Fig. 23(b). The exchange operation is performed when an appropriate control signal is provided. Consequently, the LEU has the latency of two cycles.

The perfect shuffle permutations $P^T_{N,2}$ between the processing columns can also be realized with the aid of the SEUs. The 4-element perfect shuffle $P^T_{4,2}$ exchanges consecutive elements, thus a single SEU_1 can be used for this permutation. Respectively, the 8-element perfect shuffle can be performed with the cascade of SEU_2 and SEU_1. In general, the sequential perfect shuffle permutation of a 2^k-element sequence can be performed with a structure where $k-1$ SEUs are cascaded in decreasing order of size: $\mathrm{SEU}_{2^{k-2}}$, $\mathrm{SEU}_{2^{k-3}}$, ..., SEU_{2^0} [97]. The latency of the resulting network is $\sum_{i=0}^{k-2}(2^i) = 2^{k-1}-1$. The generalized block diagram of the sequential perfect shuffle permutation network is illustrated in Fig. 23(c).

In the literature, the input and output permutations are often omitted in structures, especially in sequential structures, and only the transform kernels are considered. When the DCT is targeted at an embedded system containing a programmable processor, the input and output permutations can be realized efficiently with the index addressing modes of the processor. Therefore, for the hardware structure, it is sufficient to consider only the kernel operation of the DCT. On the other hand, when the structure is utilized without external data permutation capabilities, the additional input and output permutations are also required. However, let us omit the input and

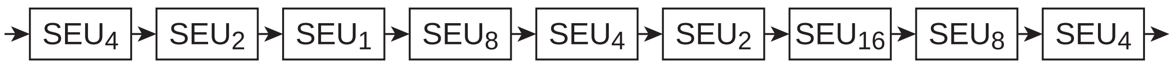


Fig. 24. *Sequential matrix transpose network for* $2^k \times 2^k, k \leq 3$, *matrix.*

output permutations in details by noting that in the cases where the in-order input and output are required, additional sequential permutation networks can be constructed. E.g., for the Hadamard input permutation, the factorization in Theorem 4 can be applied while arbitrary permutations can be performed with a cascaded SEUs arranged in increasing and decreasing order [98].

In the two-dimensional DCT based on the row-column decomposition, the matrix transpose $P_{N^2,N}$ can be performed with the sequential matrix transpose network illustrated in Fig. 14. In order to support also smaller powers of two matrix transposes with the same network, the factorization in (26) can be applied k times in order to have a decomposition

$$P_{N^2,N} = \prod_{i=0}^{k-1} \left(I_{2^i} \otimes P_{2^{k+1},2^k} \otimes I_{2^{k-i-1}} \right). \tag{101}$$

With such an arrangement, the matrix transpose is performed as k perfect shuffle permutations of $2N$-item vectors. The size of items is doubled at each cycle, i.e., during the first perfect shuffle permutation single elements are reordered, while the next perfect shuffle permutation reorders the items of two elements and so on, until in the kth perfect shuffle permutation, the item size is $N/2$ elements. Since each perfect shuffle permutation is realized with the aid of k cascaded SEUs, the decomposition in (101) results in network of k^2 cascaded SEUs as illustrated with an 8×8 matrix transpose network in Fig. 24. The proposed transpose network possesses the same latency as the network in Fig. 14 but requires more multiplexers. Consequently, smaller matrix transposes can be performed by bypassing extra SEUs with appropriate control.

4.3 Final Structures

The pipeline structure for the one-dimensional DCT can be constructed by mapping each operational column in (86) onto the corresponding sequential unit described previously. According to (85) the first processing column in the 2^k-point DCT is

$$A_{2^k}^{(0)} = D_{2^k}^{(0)} F_{2^k}. \tag{102}$$

Therefore, the corresponding realization would be a cascade of the butterfly unit and multiplier. The first operation column is followed by 4-element perfect shuffles, which map onto a 4-element perfect shuffle network consisting of SEU_1. The second processing column is

$$A_{2^k}^{(1)} = M_{2^k}^{(1)} D_{2^k}^{(1)} F_{2^k}, \tag{103}$$

which is realized as a cascade of the butterfly unit, multiplier, and local subtraction unit. This is followed by the 8-element perfect shuffles mapped onto the 8-point perfect shuffle network. The remaining $(k-2)$ processing columns in (86) are realized with a cascade of the butterfly unit, local exchange unit, multiplier, and local subtraction unit. The interconnections between the processing columns are in increasing order and the last permutation is the 2^k-element perfect shuffle.

The resulting pipeline structure without the input and output permutations is illustrated in Fig. 25(a). The structure yields arithmetic complexity of $\log(N)$ multipliers, $\log(N)$ adder/subtracters, and $\log(N)-1$ adders. The latency of the proposed kernel structure due to the data permutations is $N+2\log(N)-4$ cycles for an N-point DCT. It should be noted that the structure supports also all the smaller transform sizes of powers of two. However, this requires bypassing of certain arithmetic units and thus exploitation of unified butterfly units UBUs instead of BUs. Moreover, the structure can be freely pipelined since all the arithmetic units are in feedforward paths. In any case, additional pipelining increases latency although improves the throughput.

According to the matrix decomposition in (89), the structure for the two-dimensional DCT can be constructed by cascading two one-dimensional DCT structures with an intermediate matrix transpose network. When the two-dimensional DCT is targeted at the embedded system containing a programmable processor, the input and output permutations as well as the latter matrix transpose can be combined as in (92) into one permutation, which can be realized efficiently with the index addressing modes of the programmable processor. Such an arrangement saves additional permutations between the one-dimensional transforms and, therefore, it is sufficient to consider only the kernel operation of the one-dimensional DCT, i.e., C_N^K, and matrix transpose $P_{N^2,N}$. The resulting structure for the embedded system is depicted in Fig. 25(b).

When the structure is targeted at the system requiring the complete DCT, the integration of the input and output permutations of the one-dimensional transforms is not reasonable. Let us assume an arbitrary permutation of an 8-element sequence of

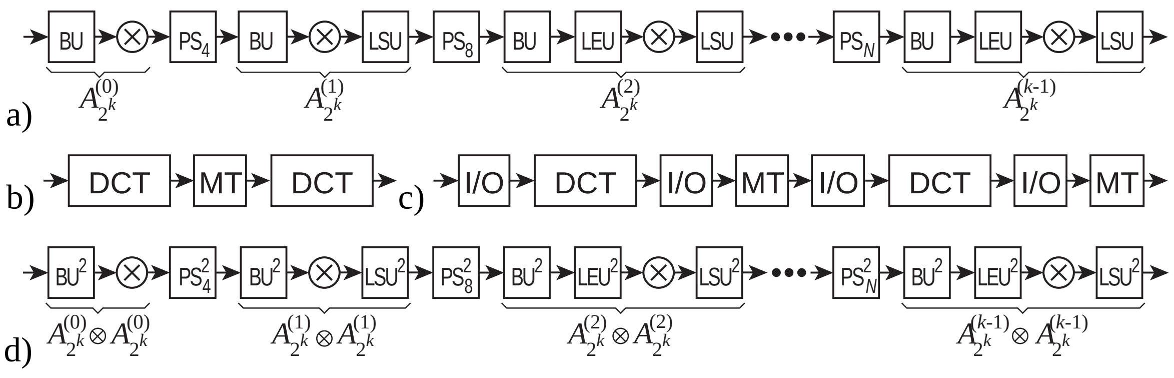


Fig. 25. *Pipeline structures for regular perfect shuffle topology DCT: (a) one-dimensional DCT, (b) two-dimensional DCT kernel based on row-column decomposition, (c) complete two-dimensional DCT based on row-column decomposition, and (d) direct two-dimensional DCT kernel.*

which permutation matrix is denoted by P_8. In this case, the maximum distance that a single element is to be moved is less than eight. If two such permutations are combined as in (92), the complexity of the permutation can be seen from the following:

$$(P_8 \otimes P_8) = (P_8 \otimes I_8)\,(I_8 \otimes P_8)\,, \tag{104}$$

i.e., the maximum distance a single data element to be moved in a two-dimensional case with a 64-element sequence will be eight times greater than in a one-dimensional case with an 8-element sequence. Since the data in the target structure is in sequential form, changing locations of data elements in the data stream requires intermediate storage. The minimum number of storage locations needed to perform the permutation depends on the maximum distance a single element needs to be moved. Therefore, in order to minimize the number of the storage locations in the input and output permutation units, the input and output permutations are not combined. The structure for the complete in-order DCT is illustrated in Fig. 25(c).

The development of the two-dimensional pipeline structure based on the direct two-dimensional DCT algorithm follows the same principles as the development of the one-dimensional structure. By comparing the matrix decompositions of the one-dimensional algorithm in (86) to the two-dimensional algorithm in (100), it can be obtained that the structures will resemble each others. I.e., the number of stages is the same but each stage $A_N^{(s)}$ in the one-dimensional DCT is now replaced with $\left(A_N^{(s)} \otimes A_N^{(s)}\right)$ as stated in (95). The decomposition of these stages can be mapped vertically onto cascaded sequential units similarly to the stages of the one-dimensional transform. Therefore, the final pipeline structure for the two-dimensional DCT kernel in (100) can be constructed by cascading the basic modules in correct order as shown in Fig. 25(d) and providing appropriate control signals. Let us remark that in order to support smaller powers of two transforms, the intermediate outputs are required into shift registers. The structure yields arithmetic complexity of $\log(N)$ multipliers, $2\log(N)$ adder/subtracters, and $2\log(N)-2$ adders. The latency of the pipeline due to data permutations is $(N+1)(N+2\log(N)-4)$ for an $N \times N$ DCT.

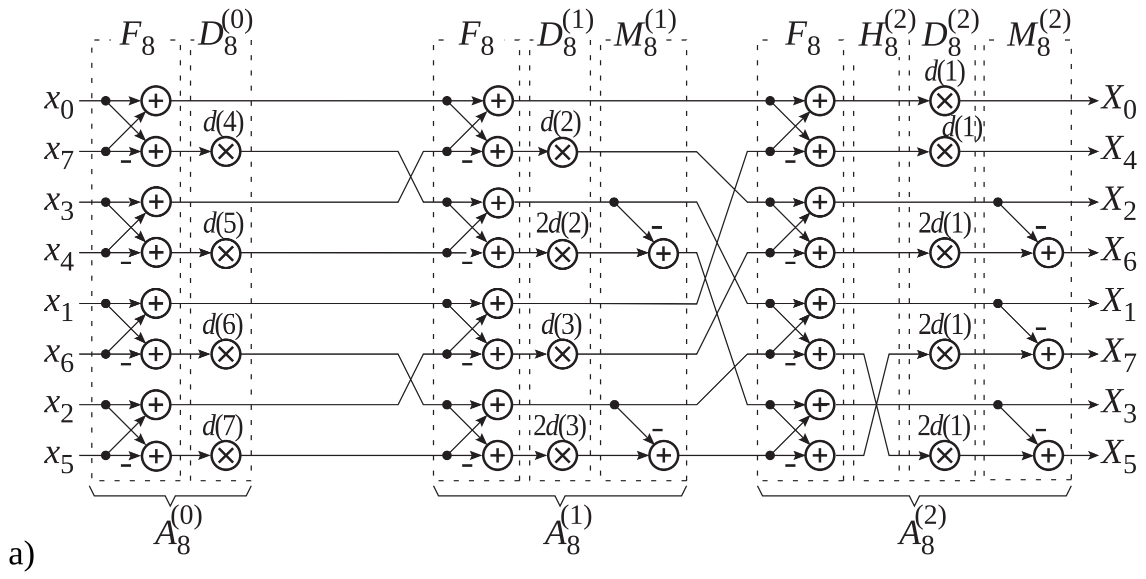
4.4 Case Study: Unified DCT and IDCT

In several applications, where transforms are utilized, also the inverse transforms are required. Therefore, a unified structure supporting both the forward and inverse transforms is preferable. Due to the popularity of the 8×8 DCT in the current multimedia applications let us consider only the unified 8-point and 8×8 DCT and its inverse. In the following, the structure based on row-column decomposition is introduced briefly whereas the unified structure based on the direct computation is also implemented and compared to other corresponding state-of-the-art solutions.

4.4.1 Algorithms and Structures

The unified pipeline structure supporting both the DCT and its inverse can be constructed by considering first the individual one-dimensional transforms. The basis for the derivation is the DCT algorithm defined in (86) of which 8-point signal flow graph is illustrated in Fig. 26(a). The corresponding inverse DCT algorithm is obtained by reversing the signal flow graph of the forward algorithm. In addition, the signal flow graph is flipped vertically in order to minimize the latency of the implementation, i.e., to have the operands in sequential form available when the result is needed. Finally, the algorithm is rescheduled for achieving the interconnections in increasing order. The resulting signal flow graph of the 8-point IDCT is depicted in Fig. 26(b). The previously described procedure for deriving algorithm for the inverse transform applies also to the direct 2-D DCT algorithm in (100).

The vertical mapping of the processing columns in the signal flow graphs of the DCT and IDCT algorithms in Fig. 26 results in pipeline structures illustrated in Fig. 27(a)


a)
F_8
$D_8^{(0)}$
$D_8^{(1)}$
$M_8^{(1)}$
$H_8^{(2)}$
$D_8^{(2)}$
$M_8^{(2)}$
$A_8^{(0)}$
$A_8^{(1)}$
$A_8^{(2)}$
x_0 x_7 x_3 x_4 x_1 x_6 x_2 x_5
X_0 X_4 X_2 X_6 X_1 X_7 X_3 X_5
$d(4)$ $d(5)$ $d(6)$ $d(7)$
$d(2)$ $2d(2)$ $d(3)$ $2d(3)$
$d(1)$ $d(1)$ $2d(1)$ $2d(1)$ $2d(1)$

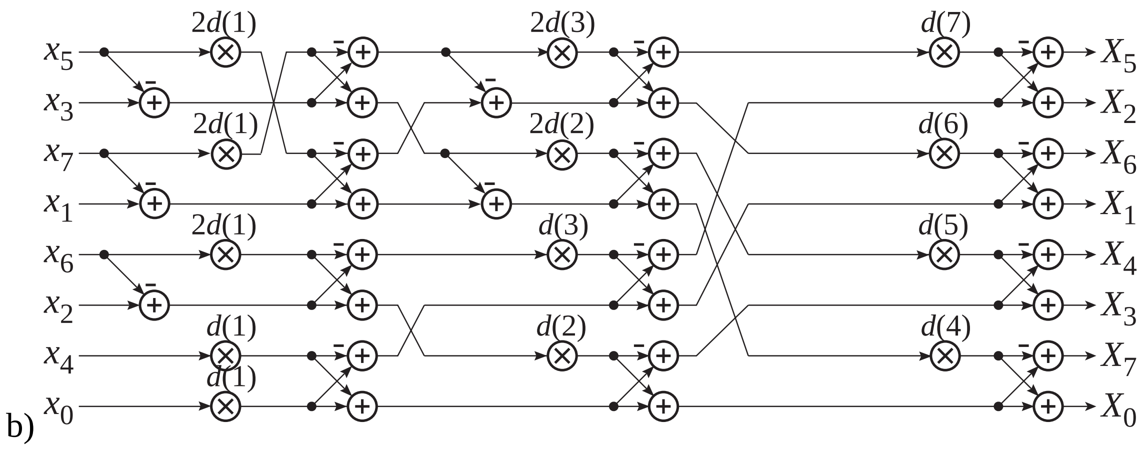

b)
x_5 x_3 x_7 x_1 x_6 x_2 x_4 x_0
X_5 X_2 X_6 X_1 X_4 X_3 X_7 X_0
$2d(1)$ $2d(1)$ $2d(1)$ $d(1)$ $d(1)$
$2d(3)$ $2d(2)$ $d(3)$ $d(2)$
$d(7)$ $d(6)$ $d(5)$ $d(4)$

Fig. 26. *Signal flow graphs of the 8-point (a) DCT and (b) IDCT.*

and (b), respectively. These separate structures imply that a unified structure supporting both transforms could be constructed by providing additional data paths to reverse the data flow of the DCT pipeline for the IDCT computation. Such an approach would, however, introduce high routing costs and complicated control. A more efficient solution is obvious when comparing the structures in Fig. 27(a) and (b); the DCT and IDCT pipelines can be mapped onto a unified processing pipeline. Such a structure supporting both the 8-point DCT and IDCT is depicted in Fig. 27(c).

The type of the transform can be easily selected by providing appropriate control signals to the different units. According to the signal flow graph in Fig. 26(a), the 8-point DCT can be computed by using the first UBU for the butterfly operation, bypassing the first SEU_2 unit and the second butterfly unit. The 4-point perfect shuffle

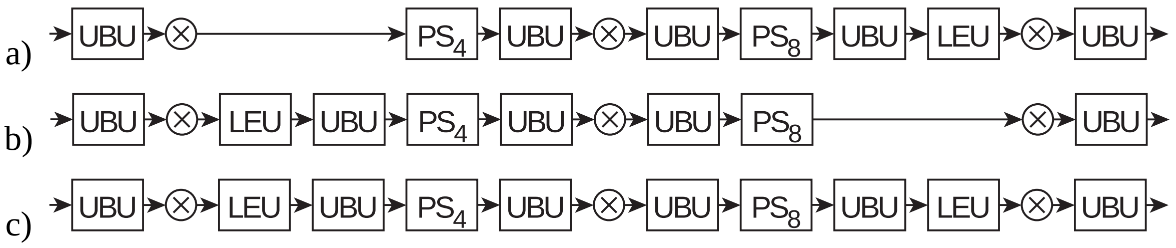


Fig. 27. *Block diagrams of the 8-point DCT kernels: (a) DCT, (b) IDCT, and (c) unified DCT/IDCT.*

is realized with a single SEU_1 denoted as PS_4. The third and fifth UBUs are used for the butterfly operations, while the fourth and sixth UBUs compute local subtractions. The 8-point IDCT, in turn, can be computed by bypassing the last SEU_2 unit and the second to last butterfly unit. The other UBUs are controlled according to the signal flow graph in Fig. 26(b).

In some cases in-order data may be preferred, and therefore, the additional input and output permutations are required. The permutation network should be capable of performing required input and output permutations for both DCT and IDCT. By comparing the input and output permutations of DCT and IDCT, it can be noticed that an application-specific permutation network depicted in Fig. 28(a) can perform all needed permutations. The only drawback of the unified network is increased silicon area, since each SEU can be bypassed with appropriate control without extra latency. Therefore, the latency of the unified input and output permutation network varies from four sample cycles to seven cycles depending on its function. The operation of the unified input and output permutation network is illustrated with the aid of timing diagrams in Fig. 28(b)–(e).

When the unified two-dimensional structure based on the row-column decomposition is targeted at the embedded system containing a programmable processor, it is constructed by cascading DCT/IDCT kernel in Fig. 27(c), the matrix transpose network MT and DCT/IDCT kernel as illustrated in Fig. 29(a). The only difference to pure two-dimensional DCT structure is the unified transform kernel. When the structure is targeted at system requiring the complete DCT or IDCT, the input and output permutations of the one-dimensional transforms are included into pipeline structure as illustrated in Fig. 29(b). Correspondingly, the structure utilizing the direct computation of the two-dimensional transform resembles the unified one-dimensional DCT/IDCT kernel structure in Fig. 27(c), i.e., each processing unit is only replaced with the

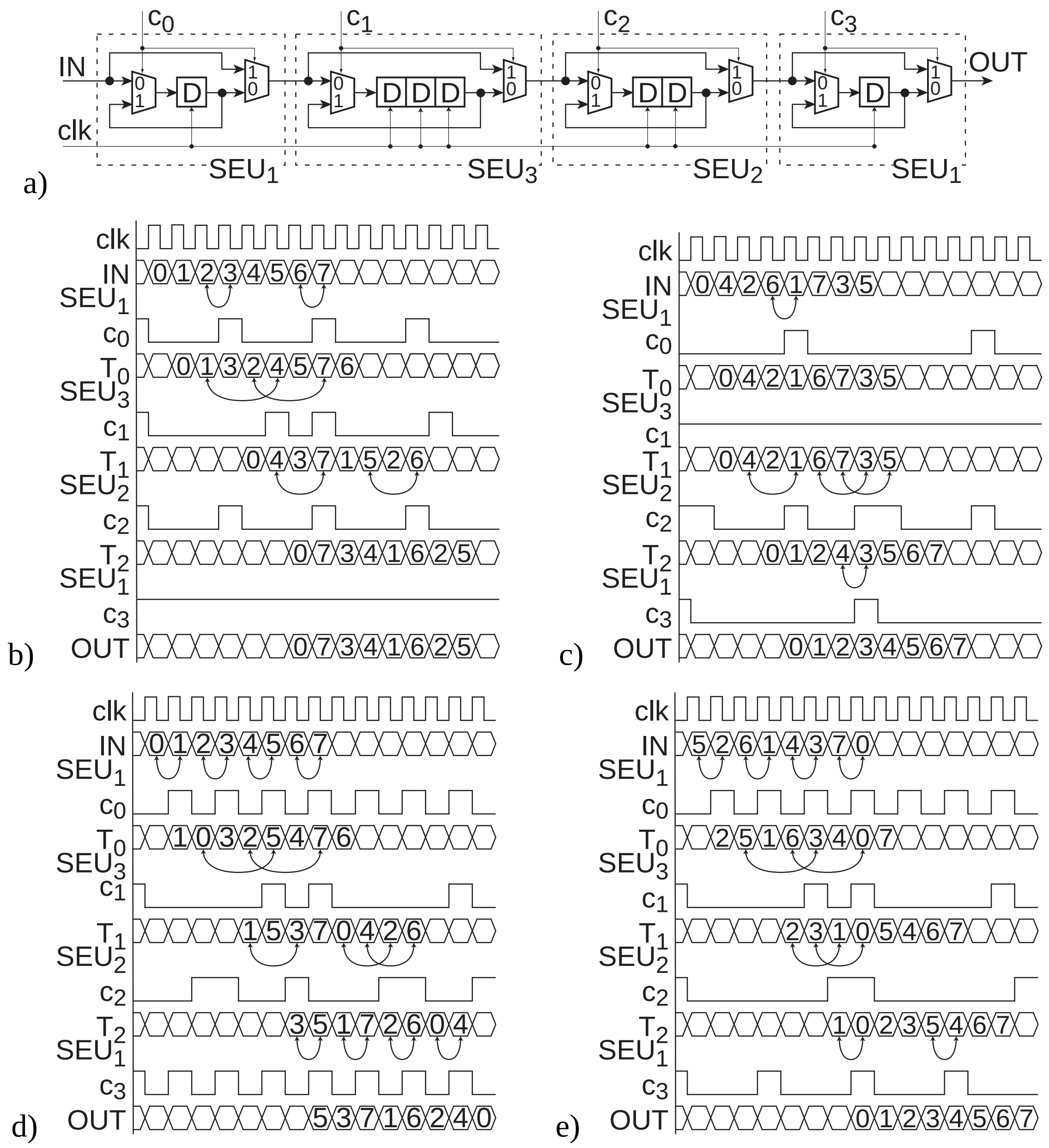


Fig. 28. *Unified input/output permutation network: (a) block diagram and timing diagrams for (b) DCT input, (c) DCT output, (d) IDCT input, and (e) IDCT output permutation.*

corresponding direct two-dimensional units as shown in Fig. 29(c). By providing appropriate control sequences and coefficients, the final pipeline can be used to compute either 8×8 DCT or IDCT as well as the corresponding one-dimensional transforms.

The previously described approach can be used to design the unified pipeline structures for the larger transform sizes as well. However, the transforms larger than 16 points require more redundancy since interconnections of the forward and in-

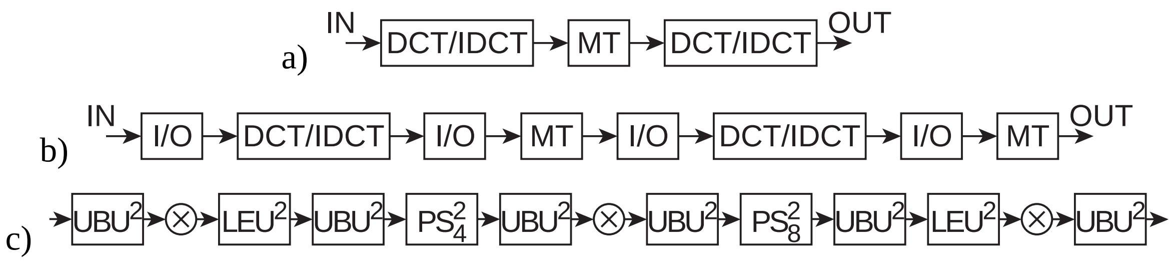


Fig. 29. *Block diagrams of the unified two-dimensional DCT/IDCT: (a) kernel based on the row-column decomposition, (b) complete transform based on the row-column decomposition, and (c) kernel based on the direct computation.*

verse transforms cannot be mapped onto a common interconnection topology. Let us remark that the one-dimensional structures supporting N-point, $N = 2^k$, transforms supports also smaller powers of two transform sizes. In the two-dimensional structures exploiting the row-column decomposition, this requires that the matrix transpose network based on the perfect shuffle decomposition in (101) is used for transpose. Instead in the structures using the direct computation, the intermediate outputs are required into the shift registers.

4.4.2 *Implementation of Direct* 8×8 *DCT/IDCT*

In order to determine an internal word length for the fixed-point implementation of the two-dimensional pipeline structure in Fig. 29(c), the error behaviour of the structure is analyzed against an IEEE specification [40] with Matlab. The simulations are performed with different word lengths and using rounding and truncation of two's complement quantization methods. The coefficients d_i are given as long as the internal word length but rounded down in magnitude. Briefly, the required word lengths for the implementation are 19 and 23 bits for rounding and truncation of two's complement, respectively.

The pipeline structure based on the direct computation of the two-dimensional transform in Fig. 29(c) has been described as a data path with Module Compiler Language and synthesized with Synopsys Module Compiler onto a 0.11 μm standard cell CMOS technology. The adder/subtracters in the UBU have been implemented as carry-look-ahead (CLA) adders, because the data width being 19 bits CLA type of adders are the fastest and the area of CLA-adders is only slightly larger than the area of 19-bit ripple-carry adders. The multipliers have been implemented as non-

Table 1. *Characteristics of the* 8×8 *DCT/IDCT pipeline implementation.*

Technology	0.11 μm CMOS
Function	8×8 DCT or IDCT
Internal word length	19 bits
Frequency	253 MHz
Latency	94 cycles
Gate count	39 424

booth multipliers, which include one 2-to-1 multiplexer at the output. With the aid of the multiplexer multiplier can be bypassed and thus, multiplication with one is also possible with fractional numbers. The data permutations correspond exactly to the presented block diagrams. The characteristics of the implementation are summarized into Table 1 where the gate count is given as equivalent 2-input NAND gates.

The functionality of the data path has been verified at a register transfer level and gate level with the aid of structure illustrated in Fig. 30. Since Module Compiler enables optimization of high performance data path captured at the structural level of abstraction but is not suitable for synthesizing general logic, e.g., random Boolean logic or state machines, the control for the DCT/IDCT data path is described with VHDL. The control is realized trivially as a 64-state state machine, which generates all the control signal and coefficients to the data path.

In order to compare the proposed pipeline structure to other previously proposed structures, we have used the unified 8×8 DCT/IDCT as the reference. The complexity of the structures is estimated based on the number of arithmetic units, multiplexers, and delay registers. Multiplexers are estimated as equivalent 2-to-1 multiplexers. The comparison is summarized in Table 2. In general, the complexity of the data permutations is reflected by the number of the multiplexers and registers. In this sense, the proposed two-dimensional algorithm has lower overall permutation complexity than the algorithms used in the other structures. The DCT algorithms used in [36, 59] possess a regular kernel followed by post-processing columns, which in inverse transform are mapped onto pre-processing columns. This implies that additional adders are needed to support both the transforms. In the proposed algorithms, the irregularities are localized into processing columns, thus the same arithmetic units can be used to realize the irregularities in both the forward and inverse transforms.

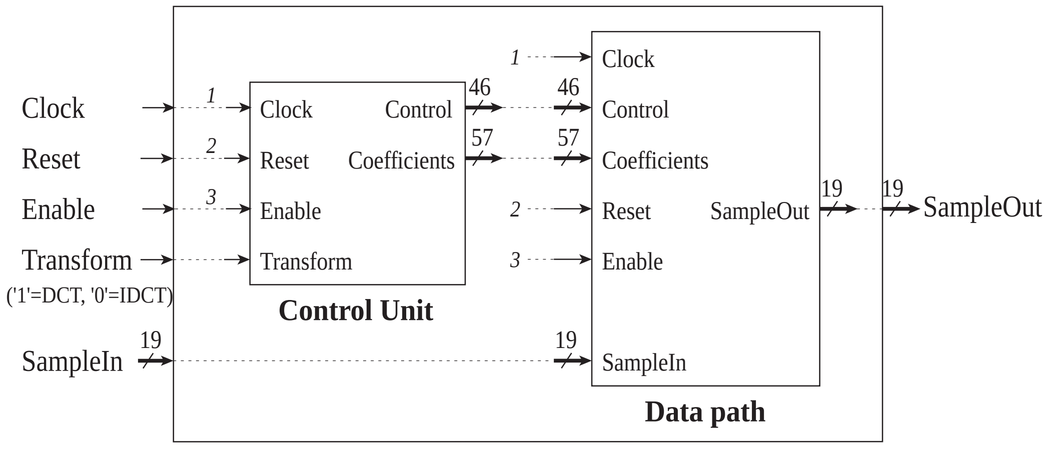


Fig. 30. *Block diagram of the verification environment of the DCT/IDCT.*

This is reflected by the number of adders or adder/subtracters. Finally, it should be noted that the proposed pipeline includes also the scaling factors b_m in (36), which are often omitted as, e.g., in [36].

In general, fully sequential, unified DCT/IDCT pipeline structures based on direct computation over the two-dimensional data are really rarely reported. However, based on the complexity estimates presented in Table 2, the sequential pipeline structures represent state-of-the-art from the hardware complexity point of view. Therefore, the proposed pipeline implementation is compared more carefully only to the most relevant reference design presented in [36]. In order to be fair in comparison, all the multiplexers are estimated as equivalent 2-to-1 multiplexers and pipeline re-

Table 2. *Comparison of the* 8×8 *DCT/IDCT structures.*

Structure	$\times$	$+/-$	R	M	mem	principle
Madisetti and Willson [68]	7	12	24	38	64	recursive, R-C
Lee *et al.* [59], folded	28	134	256	0	128	pipeline, D
Lee *et al.* [59]	24	58	128	66	128	pipeline, R-C
Katayama *et al.* [52]	7	5	>10	>25	64	recursive, R-C
Lim *et al.* [63]	64	64	492	0	0	systolic, R-C
Hsiao *et al.* [36]	4	14	216	18	0	pipeline, D
proposed	3	12	180	44	0	pipeline, D

$\times$: Number of multipliers. $+/-$: Number of adder/subtracters. R: Number of registers. M: Number of 2-to-1 multiplexers. mem: Number of memory words. D: Direct two-dimensional approach. R-C: Row-column computation.

Table 3. *Gate count estimates for the basic units (area optimized / speed optimized).*

W	$\times$	$+/-$	R	M
19	2088 / 3288	290 / 1279	95 / 109	39 / 151

$\times$: Multiplier. $+/-$: Adder/subtracter. R: Register. M: 2-to-1 multiplexer.

gisters are not included. The error behaviour of the reference structure is analyzed with exactly similar procedure as the presented structure. In addition, all resources required for implementation are assumed to be similar from the same standard cell technology. The assumed gate counts of area and speed optimized resources with the required word length are collected into Table 3. Altogether, the presented implementation exhibits improvement in estimated gate count as summarized in Table 4. Let us remark that both structures can be designed for the same throughput rates due to the fact that arithmetic units are not located in feedback loops in either of the structures.

4.5 Summary

Since regularity of the perfect shuffle topology DCT algorithms allows efficient utilization of temporal parallelism, each operational column can be mapped onto a corresponding sequential unit. By cascading these basic modules in correct order and providing appropriate control signals, the new modular single-rate pipeline structures can be constructed. Due to recursive description of the DCT algorithms and correspondence between operational stages and basic modules, the structures can be extended to support larger transforms.

The structures can be freely pipelined since all the arithmetic units are in feedforward paths. Furthermore, the one-dimensional structures support also all the smaller

Table 4. *Comparison of the fully sequential unified DCT/IDCT pipelines.*

Structure	$\times$	$+/-$	R	M	W	Gate Count
Hsiao *et al.* [36]	4	14	216	18	19	33 634 / 57 320
proposed	3	12	180	44	19	28 560 / 51 476

$\times$: Number of multipliers. $+/-$: Number of adder/subtracters. R: Number of registers.
M: Number of 2-to-1 multiplexers. W: Required word length

transform sizes of powers of two with appropriate control when basic arithmetic units with a bypass path are exploited for computation. In the two-dimensional structures exploiting the row-column decomposition, this requires also that the matrix transpose network based on the perfect shuffle decomposition is employed for transpose. The two-dimensional structures exploiting the direct algorithm support one-dimensional transform but in order to support smaller powers of two transforms, the intermediate outputs are required into shift registers.

The regularity of the algorithm and the modularity of the structure have been exploited when mapping the 8×8 DCT and its inverse onto a common pipeline structure with minimal redundancy in the amount of hardware. Due to distribution of the irregularities between butterfly columns, both the DCT and IDCT algorithms can be mapped onto same resources. The resulting structure for the 8×8 DCT/IDCT has been proven to be area-efficient compared to other reported solutions. In order to prove the feasibility, the demonstration pipeline implementation of the 8×8 DCT/IDCT is based on the data path model of the structure. When synthesized onto a 0.11 μm standard cell CMOS technology, the DCT kernel occupies 39 424 equivalent 2-input NAND gates achieving the operation frequency of 253 MHz.

5. VARIABLE LENGTH CODING

In general, any data can be interpreted to consist of symbols, whatever they are. The ultimate purpose of compression is to represent the set of symbols in source data with minimum number of bits. This is achieved by representing frequently occurring symbols with shorter codewords. Such a coding method results in variable codeword lengths hence the name variable length coding (VLC). An example of such a coding technique is the well-known Huffman coding. The inverse procedure for VLC is variable length decoding (VLD).

In this chapter, the variable length coding process and the properties of the variable length codes are introduced briefly before going into the actual topic of the chapter, i.e., the variable length decoding on hardware or in short, variable length decoders. Our objective is to develop a novel scheme for decoding multiple symbols in parallel. Since only a few multiple-symbol decoders have been reported, surveys gathering them up are missing. Therefore, in the following, some variable length decoders are collected together and reviewed especially with respect to parallel multiple-symbol decoding. The chapter is closed with a brief summary.

5.1 Definitions and Properties

Let us assume an information source that consists of symbols s_k with probabilities p_k in a set $S = \{s_1, \ldots, s_n\}$. The theoretical lower bound on the average number of bits required to represent a symbol in the given set is defined by entropy H [93]

$$H = -\sum_{S} p_k \log_2 p_k. \tag{105}$$

In other words, entropy defines the average amount of information contained by a symbol. In order to reach entropy, noninteger codeword lengths are needed. A variable length coding method approaching the lower bound with noninteger codeword lengths is arithmetic coding [23, 88, 89], which is, however, omitted in this Thesis.

Instead, suboptimal compression can be obtained with integer codeword lengths and a coding method providing the shortest integer length codewords is Huffman coding [39]. Huffman presented a method to construct a minimum-redundancy code, i.e., the average number of coding digits per message is minimized. Huffman's coding method is based on the following five theorems:

Theorem 9. *Each codeword c_k is unique.*

Theorem 10. *Codewords are constructed in such a way that boundary information is not needed to specify the beginning or end of the codeword once the starting point of the encoded data stream is known.*

Theorem 11. *The more probable symbol, the shorter codeword;*

$$p_1 \geq p_2 \geq \ldots \geq p_{n-1} \geq p_n \;\;\Rightarrow\;\; l_1 \leq l_2 \leq \ldots \leq l_{n-1} = l_n \tag{106}$$

where l_k is used to denote the length of the codeword c_k corresponding symbol s_k.

Theorem 12. *Two least probable symbols should have the same codeword with the exception of the last bits.*

Theorem 13. *Each possible sequence of length $l_n - 1$ bits must be used either as a codeword or must have one of its prefixes used as a codeword.*

The Huffman coding procedure, i.e., the creation of a Huffman tree is illustrated in Fig. 31 where the notation for the symbols is $s_k(p_k)$. First, all the source symbols A–F on the left are ordered in decreasing order according to their probabilities given as percentages in parentheses. Then, two least probable symbols E and F are connected with edges 1 and 0, respectively, to a new node of which probability is equal to the sum of the probabilities of two least probable symbols, i.e., 10%. Subsequently, the resulting node is interpreted as a new symbol. This procedure is repeated until the root node with probability of 100% is formed. Finally, the codewords are obtained by

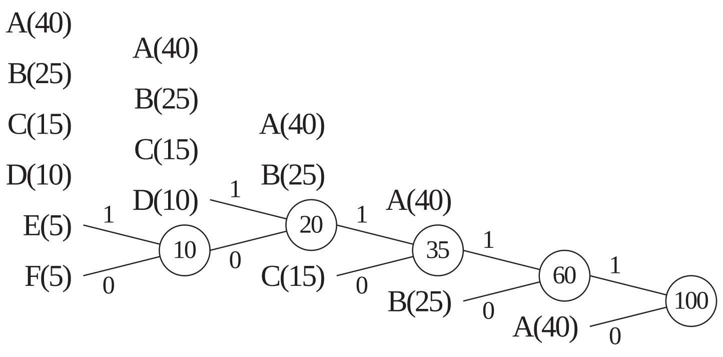


Fig. 31. *Example of the Huffman coding.*

tracing the tree back to source symbol. Huffman coding results in a codeword table in which average codeword length l_{av} is

$$l_{av} = \sum_S p_k l_k. \tag{107}$$

The resulting codeword table and the related statistics for the encoding example in Fig. 31 are summarized in Table 5.

To summarize the main features, the Huffman coding is a block-based coding method meaning that each symbol has a fixed codeword. The resulting encoded data stream consist of consecutive codewords without explicit boundary information. This introduces a sequential dependency which complicates a decoding process. The Huffman code is nonsingular and instantaneous, i.e., all the codewords are distinct and no codeword is the prefix of some other codeword. It is a lossless coding technique, meaning that data can be fully reconstructed without any distortion or noise. On the other hand, it is error sensitive in the sense that a possible bit error may propagate

Table 5. *Summary of the Huffman coding example.*

s_k	p_k	$-p_k \log_2 p_k$	c_k	l_k	$p_k l_k$
A	40	0.53	0	1	0.40
B	25	0.50	10	2	0.50
C	15	0.41	110	3	0.45
D	10	0.33	1111	4	0.40
E	5	0.22	11101	5	0.25
F	5	0.22	11100	5	0.25
		$H = 2.20$			$l_{av} = 2.25$

far; decoding may be continued but nearly all symbols are incorrect until the last bit of a decoded data sequence coincides with the last bit of the correct codeword in sequence [73]. Loss in coding efficiency with respect to entropy is due to the fact that the Huffman coding always encodes a source symbol with an integer number of bits. However, the average codeword length approaches the entropy with infinite block size. On the other hand, the complexity of the coding procedure will be out of control.

In principle, the Huffman coding is adaptive coding, i.e., the codeword table is dependent on source data and its statistics. However, by analyzing huge amount of typical data, the achieved statistics can be exploited to approximate data of the same type. Consequently, the fixed codeword table with reduced number of codewords based on the Huffman procedure can be defined for coding and decoding. The possible symbols that do not have codeword are passed with the aid of an exceptional fixed length code. As a result, the compression efficiency is also sacrificed. In any case, the corresponding approach have been utilized in several applications, e.g., in MPEG-2 [45]. Also in this Thesis, the discussion is limited to the decoding of prespecified variable length codes.

5.2 Variable Length Decoders

In this Thesis, the objective is the decoding of variable length codes, i.e., the variable length decoding on hardware. The principal block diagram of the VLD is illustrated in Fig. 32. A codeword is detected from a block of the variable length coded input stream and this codeword is used to determine the actual symbol with the aid of predefined codeword values, i.e., a codeword table. The input stream is then aligned according to codeword length for the next decoding iteration. In general, there is no explicit boundary information for detecting the end or beginning of the codeword in the encoded data stream. Therefore, the length of the current codeword should be known before the next codeword can be decoded. This feature complicates the decoder design substantially and limits the performance.

The decoding structures for variable length codes can be characterized according to the parallelism of the data processing. The bit-serial processing refers to the sequential, bit-by-bit processing of the encoded bit stream at constant rate. Correspondingly, the bit-parallel processing refers to the processing over several bits at a time

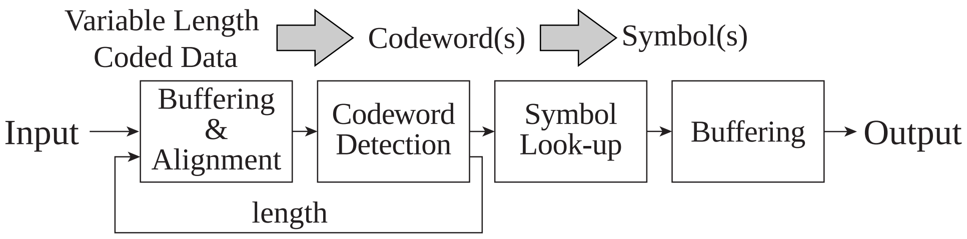


Fig. 32. *Block diagram of the generalized variable length decoding.*

in a parallel manner. Similar characteristics can be given according to the number of codewords to be decoded or symbols obtained from the output. A traditional VLD method is to decode one symbol at a time in a symbol-serial fashion. Instead, the symbol-parallel or multiple-symbol decoding refers to decoding of several symbols at a time. In the following, existing variable length decoders are overviewed according to previous categorization.

5.2.1 Serial Decoders

Bit-serial decoders, also referred to as tree-based decoders [9], decode an input data stream sequentially bit-by-bit. The encoded input stream is compared to a binary tree starting at the root of the tree until the entire codeword is detected in the leaf node corresponding the symbol. In [65, 66], Lin and Messerschmitt considered variable length decoder as the implementation of a special finite-state machine (FSM) which does not always return output. Such a FSM for the Huffman tree given in the example in Fig. 31 is illustrated in Fig. 33(a). The sequential FSM can be realized with a read only memory (ROM) as depicted in Fig. 33(b) where codeword table is assumed to consist of 256 codewords for 8-bit symbols [10]. Briefly, each node is stored into the ROM as the states of the FSM. In addition, the ROM consists of status bits indicating whether the state corresponds to a terminal node in the Huffman tree, i.e., a symbol, or is used as a basis to the next address. The same status bit is used to reset the address register when the symbol is found. In other words, the read address is constructed from the current state and the incoming bit.

Another way to realize the decoder as the inverse interpretation of the Huffman tree is to implement branches as demultiplexers and terminal nodes as storage locations containing corresponding symbols. In [9], Chang and Messerschmitt presented the structure illustrated in Fig. 34. A token, i.e., a pointer to the current stage, is for-

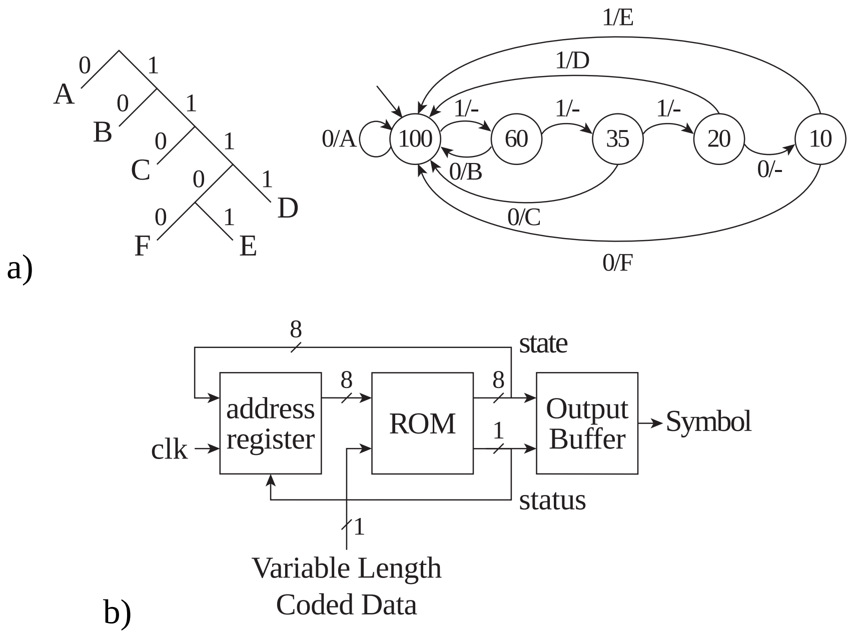


Fig. 33. Variable length decoding as a finite-state machine: (a) Huffman tree with its state diagram and (b) block diagram of the FSM-based variable length decoder [10].

warded through the demultiplexers according to the encoded input stream. When the token enters the storage location, the symbol is returned. The similar approach has been used in the decoder proposed by Mukherjee *et al.* in [77] where individual storage location for the symbols have been replaced with a common ROM or a balanced binary tree module returning the symbol. Drawback in these approaches is that they are not programmable but the decoder needs to be redesigned for different codeword table. Also the support for adaptive codes would require too much hardware [76].

Instead of the direct implementation of the Huffman tree, the variable length decoding can be performed with a single processing unit consisting of control logic and a arithmetic unit in addition to a symbol memory. In principle, the symbols are mapped onto the memory according to a mapping scheme in such a way that the address for reading a symbol can be determined with the aid of basic arithmetic. E.g., the nodes are numbered and symbol is found by tracing the tree back by accumulating the offsets given to edges [76, 78]. Since such mapping schemes can be applied to different codeword tables and the contents of the memory can be changed, the memory-based structures are considered to be programmable. In other words, the same hardware can be used for any type of tree based codes. Consequently, the structure is also applicable for adaptive codes.

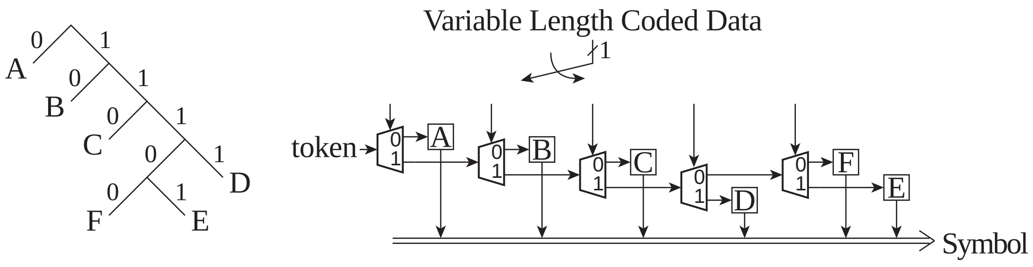


Fig. 34. *Direct mapping of the VLD onto demultiplexers and storage locations* [9].

In [81], Park and Prasanna proposed a memory-efficient scheme for mapping the Huffman tree onto a memory in which nodes are numbered starting from the root and proceeding from left to right at the each level. The terminal nodes, i.e., symbols are mapped onto one memory in increasing order according to node numbers. Furthermore, the numbers of the terminal nodes at each level are stored into another memory. The read address during the decoding is determined according to the input, current level, position of the node from the leftmost node at the current level, and the accumulated number of the symbols at the previous levels.

In the bit-serial decoders discussed so far, the comparison is performed with a constant input rate, one bit per cycle, until the entire codeword is detected in the corresponding leaf node. Due to the variable codeword lengths, the serial processing results in a variable output rate. Short decoding time is achieved only with short codewords. However, under hard real-time constraints, the required output rate should be fulfilled also with long codewords, thus the performance is defined by the latency of the long codeword processing. Furthermore, the serial processing is not applicable for multiple-symbol decoding with a single data stream due to the recursive dependencies between the codewords.

5.2.2 Parallel Decoders

For improving the decoding performance, bit parallelism can be increased by comparing simultaneously several bits of the incoming bit stream to the Huffman tree. Such an approach with a memory-based codec structure has been introduced by Mukherjee *et al.* in [76, 78]. The approach requires some overhead in control logic compared to the bit-serial approach but the average throughput is directly proportional to the number of bits to be processed at a time. In other words, the input rate is maintained

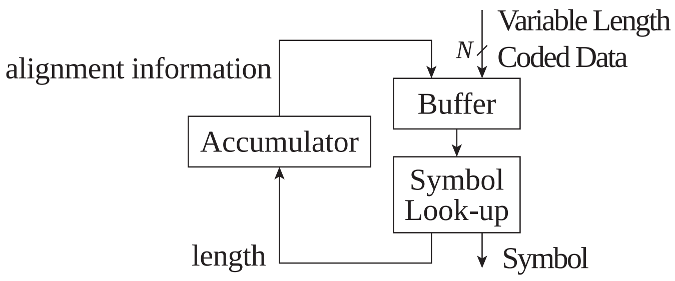


Fig. 35. *Block diagram of the parallel constant output rate decoder.*

constant, several bits per cycle, but the output rate is still variable as in the bit-serial approach. Therefore, the limitations of the bit-serial decoders apply also to this kind of parallel variable output rate decoders.

In order to guarantee a symbol for each cycle, Lei and Sun in [62] presented a parallel constant output rate decoder in which the number of bits buffered for the decoding at a time is equal to the longest codeword length. The principal block diagram of the decoder is depicted in Fig. 35. Briefly, a block of the encoded bit stream is buffered and matched with the stored prespecified codewords in a symbol look-up of which look-up table (LUT) can be implemented with a ROM, programmable logic array (PLA), or random access memory (RAM). Consequently, the corresponding symbol is returned and the codeword length is accumulated with the accumulator of which value is used as a pointer to the correct location in the buffered bit stream. Furthermore, the accumulator triggers the reloading of the buffer with new data. In [103], Sun improved the structure by removing the accumulator from the feedback loop.

Both the variable and constant output rate decoders have been reported during the years. Nevertheless, a common feature in the parallel decoders has been that they use shifter for aligning the input stream. Such an arrangement forms a feedback loop. Instead in [90,91], Rudberg and Wanhammar replaced the shifter with a shift register while otherwise the decoder resembles other parallel pipelined decoders. However, the decoder, which is illustrated in Fig. 36, can be characterized as a hybrid between the serial and parallel decoder, i.e., it is a parallel pipelined decoder with the behaviour of the serial decoder. In short, the shift register receives sequentially the encoded data stream. A pipelined length decoder checks codeword lengths until the matching length is found and returned. The feedback from the length decoder to the shift register is not needed but it is replaced by a synchronous reset signal to a counter. In principle, the maximum operation frequency is limited by the delay of a single logic gate.

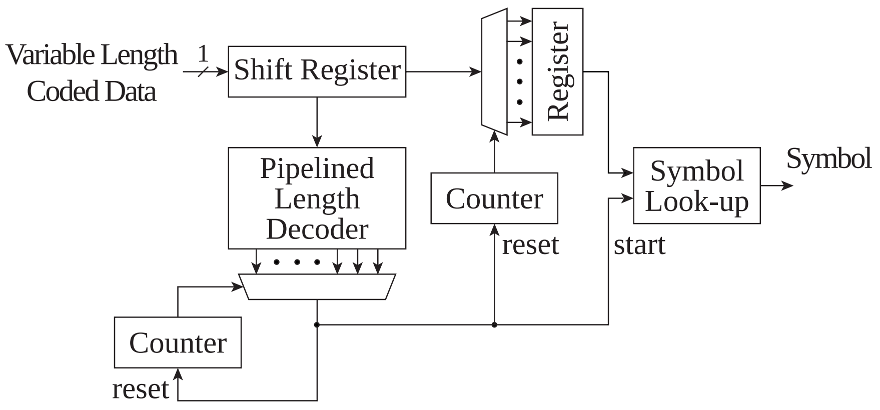


Fig. 36. *Block diagram of the loop-free parallel decoder.*

Despite of the output rate, variable or constant, the main contributions to development of the parallel variable length decoders are related to the partition of the codeword table into groups and mapping them onto memory with the best possible memory space utilization. In [76, 78], Mukherjee *et al.* proposed a memory mapping scheme for the bit-parallel tree-based decoding in which a distinct memory location is determined for each node in such a way that each edge is considered as an offset to its parent node. Consequently, the location of the symbol is determined by accumulating the offsets. Lee *et al.* in [60] continued the memory mapping scheme development along the same direction with a 2-bit tree-based decoding. In this approach, the nodes have been coded with 3-bit node codes to indicate the type of the node, i.e., if the node has all child nodes labeled with two bits, it has at least one child node labeled with a single bit, or it is a terminal node. In addition, the nodes of the same parent node are merged into a group. The node codes of the group and the symbols are stored into own memories. The approach yields about 20% reduction in the total memory space compared to Mukherjee's approach [61].

In [80], Ooi *et al.* presented a codeword table segmentation technique in which couple of bits are traced at a time. A variable length decoding data table consists of three separate data fields: 1) the next address for the node having children or a symbol for the terminal node, 2) a shift quantity for the incoming bit stream, and 3) a termination status. Consequently, either the symbol or continuation to the next segment can be concluded according to the traced bits. Correspondingly, Hashemian in [29, 30] presented a tree-based decoder applying a technique where variable length codewords are ordered and grouped according to their lengths. Instead in [17], Choi and Lee presented an approach to partition codewords in a single-side growing tree into two clusters: regular bit pattern like a prefix code and second cluster identifying codeword. Similarly in [82], Park *et al.* exploited leading zeros for partition of the codeword table.

While smaller memories through the efficient mapping schemes provide faster accesses to symbol memory, another key issue in the decoder design is the fastest possible search. Traditionally, codewords are detected with a pattern matching based on logical functions as presented, e.g., by Lei and Sun in [62]. Choi and Lee in [17] achieved improvement with the clustering approach and performing the parallel pattern matchings with smaller patterns. Correspondingly, Hsieh and Kim in [37] accelerated pattern matching by presenting a technique to exploit maximum likely bit patterns for grouping the codewords. By storing some additional data for the nodes, Lee *et al.* in [61] introduced a technique to perform codeword prediction. With such an arrangement, the next node information can be accessed simultaneously with the incoming bit stream. However, the approach will increase the hardware complexity and therefore, it is more applicable in a processor-based platform reported by Shieh *et al.* in [95].

In [100], Sima *et al.* presented a variable length decoding approach to improve the performance of a general purpose processor by augmenting it with an FPGA-based variable length decoder. The decoding has been implemented as parallel look-ups into codeword groups followed by the selection of the valid symbol. The resulting variable length decoder is illustrated in Fig. 37. Briefly, a codeword table is partitioned into groups according to characteristics of the design platform, i.e., the used FPGA, and these groups are mapped onto LUTs referred as group decoders. The decoding is started by forwarding bit fields from a block of the encoded bit stream to the parallel group decoders and performing parallel matching with codewords. Each group decoder returns the symbol value as if the generated symbol was valid. The selection of the proper symbol is done according to leading bits in a group detector which determine the correct group.

In addition to utilization of leading characters, Wei and Meng in [121] exploited also the numerical properties of the variable length coding tables specified in JPEG standard [44] and replaced the traditional pattern matching with arithmetic operations. The approach is based on the property that codes of the same codeword length are numerically sequential. Hence, the symbol can be returned with the aid of a minimum code

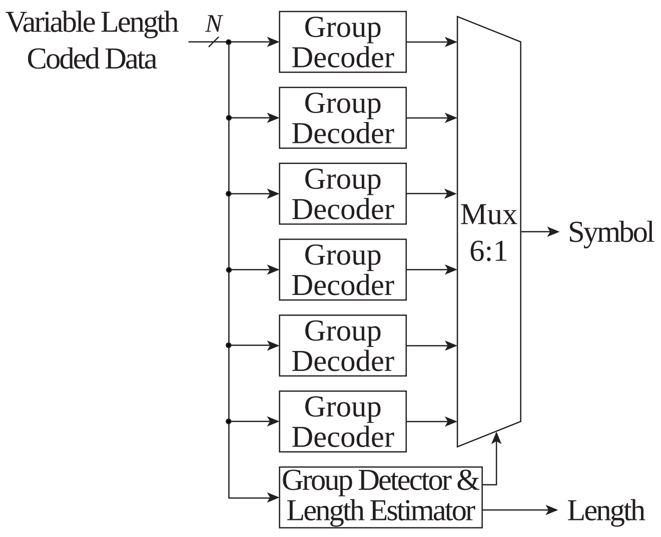


Fig. 37. *Block diagram of the parallel group-based decoder.*

defining the length and an offset with respect to the minimum code. In [96], Shieh *et al.* continued the development of the VLD based on the arithmetic operations by introducing a pseudo constant length code and inter-group symbol memory mapping. Briefly, the codeword lengths are equalized by adding zeros after the codeword. The resulting pseudo constant length codewords can be identified unambiguously due to instantaneous nature of variable length codes. Consequently, the pseudo constant length codewords having the same length as the longest variable length codeword can be treated as individual binary numbers and ordered numerically in sequential order.

So far, we have been discussing on high throughput variable length decoding without paying any attention to power consumption. Let us next outline roughly the main approaches used in a low-power design. For the data alignment, a low-power barrel shifter has been proposed by Lin and Jen in [64]. Since LUTs consume considerable amount of power in the VLD, the size of the LUTs can be decreased, e.g., with the aid of prefix decoding in the codeword detection like Choi and Lee in [17] or partitioning of the symbol look-up as Rudberg and Wanhammar in [91]. By using the property that the short codewords are more probable, the average power consumption can be decreased by breaking long codewords into parts and consequently reducing the size of the codeword detection as reported by Cho *et al.* in [16]. Furthermore, the frequencies of the codewords and the energy consumption estimates for the LUT partitions have been used as a basis for the partition of the codeword table into variable size LUTs. The approach can be further improved by taking the probabilities of successive short codewords into account and employing additional small LUTs as proposed by Lee and Park in [58].

The bit-parallel processing offers the improved throughput at the price of increased hardware and control complexity while the data dependencies between codewords still exist and limit the overall performance. In other words, in the bit-parallel decoding, the encoded data stream needs to be buffered and shifting is always required after the determination of the codeword boundaries. Moreover, according to the properties of the variable length codes, the shorter codewords are more probable than the longer ones. Hence, most probably a block of bits in the input stream contains more than one codeword. Therefore, decoding more codewords at a time and decreasing the need for shifting may further improve the decoding performance. These facts encourage studies towards multiple-symbol decoding to be discussed in the following.

5.2.3 Multiple-Symbol Decoders

In multiple-symbol or symbol-parallel decoding schemes, the major design issue is to break the data dependencies between the codewords. Another issue is the management of the increasing hardware and control complexity, especially when large codeword tables and long codewords are used. Let us remark that the memory mapping and search techniques discussed with the parallel decoders in the previous section are also applicable to the multiple-symbol decoding. Moreover, a parallel decoder, which operates on a buffer whose size is equal to the longest codeword, is the natural basis for the multiple-symbol decoder design since often a block of bits in the input stream contains more than one codeword.

Chang and Messerschmitt in [9] presented a multiple-symbol variable length decoder for short codewords. The constant output rate variable length decoder presented earlier by Lei and Sun in [62] has been extended to decode more than one codewords per cycle when possible. In the resulting decoder illustrated in Fig. 38, the output field of the PLA-based symbol look-up contains now several symbols and an additional field indicating the number of the symbols that are returned. Consequently, the decoding performance of this variable input / variable output rate multiple-symbol decoding scheme is constrained by the increased complexity and incurred delay penalty of the PLA [10].

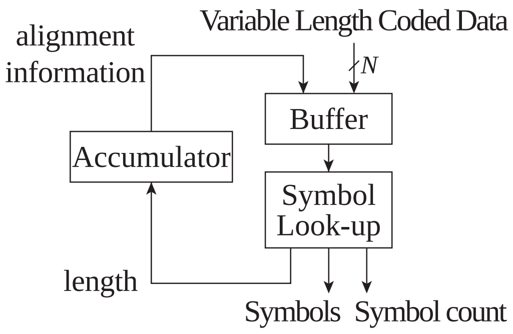


Fig. 38. *Block diagram of the variable I/O rate multiple-symbol decoder.*

In [37], Hsieh and Kim presented a multiple-symbol VLD algorithm in which objective is to speed up the decoding process by matching two or more shorter or, in other words, more probable codewords in parallel. The principal structure of the resulting decoder is illustrated in Fig. 39. The first pattern matching unit includes all the codewords, while the rest of the units consist of the shorter and more probable codewords. Each pattern matching unit returns group and length information of the possible codeword and therefore, combinatorial logics and barrel shifters are required to manage the RAM-based symbol look-up and provide the alignment information for the next decoding cycle. The method can be extended to decode also long codewords in parallel but it is often impractical due to too many possible combinations. In order to find a compromised solution for the decoder in terms of system requirements like decoding performance and cost, a systematic approach has been outlined by Hsieh in [38].

In addition to decoding only short codewords in parallel, the increasing complexity can be managed by restricting the number of the symbols per cycle. E.g., in the illustration in Fig. 39, at most two symbols are returned at a time. If there are several consecutive long codewords in the data stream, the decoding approach yields the same throughput as the parallel decoding. However, the approach decodes short codewords simultaneously, which will not result in the tremendously high cost of hardware but will increase the average throughput.

The multiple-symbol decoders discussed so far have been based on the fact that often a block of bits in the input stream contains more than one codeword. In general, this kind of decoders have variable input and output rates. The increasing complexity has been managed by restricting the buffer size, the number of symbols or decoding only short codewords simultaneously. An alternative approach is to keep the output

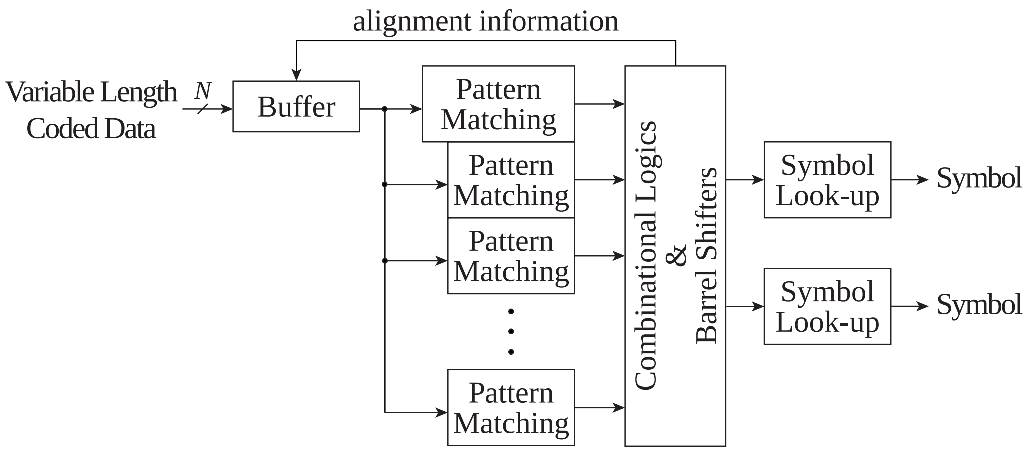


Fig. 39. *Block diagram of the group-based multiple-symbol decoder.*

rate constant but to produce more than one symbol in every cycle. Such an approach results in variable input / constant output rate multiple-symbol decoding. Let us remark that in order to return symbols at constant rate, the number of bits to be decoded at a time should be equal to the product of the longest codeword length and number of the symbols at the output. Consequently, the whole input buffer is not exploited when it consist of short codewords.

Instead of expanding the output field of the single symbol look-up, Park in [83] increased parallelism with another barrel shifter and symbol look-up. With such an approach, the operational delay in an accumulator can be reduced for achieving high speed decoding. The resulting 2-symbol decoder is depicted in Fig. 40. In short, the buffered encoded data stream in latches is fed to the barrel shifter which provides a decoding window for the first PLA-based symbol look-up. The symbol look-up re-

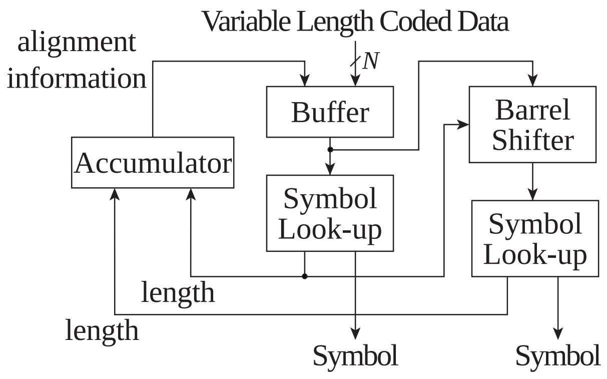


Fig. 40. *Block diagram of the 2-symbol decoder with an additional shifter.*

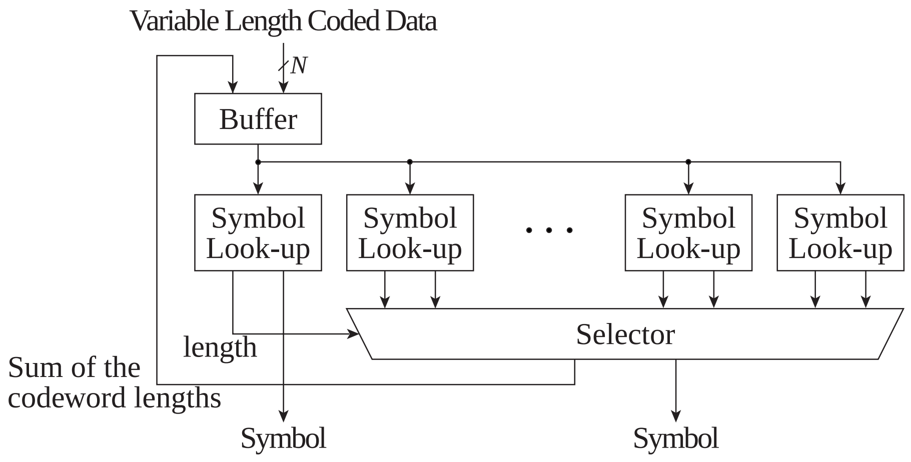


Fig. 41. *Block diagram of the 2-symbol decoder with parallel symbol look-ups.*

turns the codeword length and symbol as in the conventional parallel decoder. Instead of forwarding the codeword length only to the accumulator, the length is also fed to the second barrel shifter having an input from the first barrel shifter. The second barrel shifter in turn provides the decoding window for the second symbol look-up independently of the accumulator. Consequently, two variable length codewords are decoded without an operational delay in an accumulator for shifting the decoding window of the first barrel shifter.

In [53], Kinouchi and Sawada proposed a constant output rate multiple-symbol decoder returning two symbols at a time. The principal structure of the decoder is shown in Fig. 41. The encoded bit-stream is forwarded to parallel symbol look-ups. Each symbol look-up performs matching with all codewords and returns a symbol as if the symbol was valid. In addition, the first symbol look-up returns the length of the codeword while the rest symbol look-ups return the sum of the codeword lengths. The symbol from the first symbol look-up is returned and its length is fed to a selector which selects the second codeword. The sum of the codewords is forwarded as alignment information to the buffer in which the leading bit position determined for the next decoding cycle.

In [101], Sima *et al.* extended their decoding scheme proposed in [100] to support the constant output rate multiple-symbol decoding on the FPGA-augmented processor. The principal block diagram of the decoder returning two symbols per cycle is illustrated in Fig. 42. Using the terms previous, current, and next in chronological order, the main idea is to determine the symbol and the length for the current codeword, and

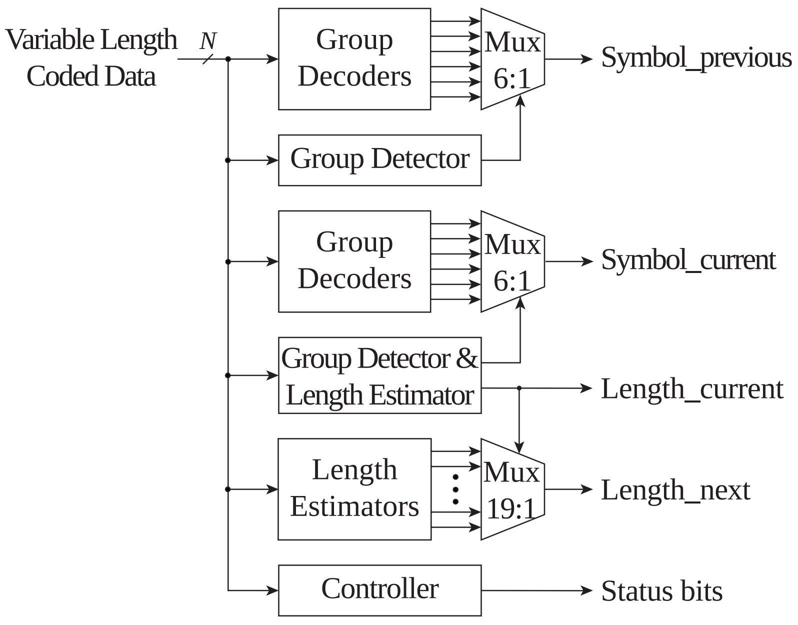


Fig. 42. *Block diagram of the 2-symbol decoder with length prediction.*

only the length for the next codeword during the same VLD call. Concurrently, the symbol for the previous codewords are determined. In more details, each group decoder operates as in Fig. 37. Simultaneously, length estimators determine the length of the current codeword and the length candidates for the next codeword. The valid length is selected according to the current length. In other words, the generation of the next symbol is postponed to the subsequent cycle and the series of decoding cycles results in symbol-parallel decoding.

To summarize the current multiple-symbol approaches, the performance is partly limited due to the fact that the arbitrary length input buffers are not exploited. According to the previous discussion, current variable input / output rate decoders operate on a buffer whose size is equal or very close to the longest codeword length. In other words, the complexity has been managed by decoding only short codewords concurrently. Furthermore, the number of symbols may be limited, which is done especially for achieving a constant output rate.

5.3 Summary

Based on the above updated survey, let us conclude the chapter with a brief summary. In general, there is no explicit boundary information for detecting the end or beginning of the codeword in the variable length coded data stream. Therefore, the length of the current codeword should be known before the next codeword can be decoded. This feature complicates the decoder design substantially and limits the performance. Consequently, a traditional VLD method is to decode one symbol at time in symbol-serial fashion. Two principal approaches exist: the bit-serial tree-based processing and the bit-parallel approach. In bit-serial decoders, the performance is defined by the latency of the long codeword processing. Furthermore, the serial processing is not applicable for multiple-symbol decoding with a single data stream. The bit-parallel processing offers the improved throughput at the price of increased hardware and control complexity. On the other hand, although a block of input data is buffered, the data is not exploited when it consists of short codewords.

In multiple-symbol decoding or symbol-parallel schemes, the major design issue is to break the data dependencies between codewords. Another issue is the management of the increasing hardware and control complexity, especially when large codeword tables and long codewords are used. However, the current multiple-symbol approaches, the performance is partly limited due to the fact that the arbitrary length input buffers are not exploited. In other words, the complexity has been managed by decoding only short codewords concurrently or the number of symbols is limited.

6. VARIABLE LENGTH DECODING SCHEME

The main challenge in the multiple-symbol parallel VLD is to break the recursive dependencies between the codewords or at least to minimize their effects to the throughput. Since the encoded data stream is buffered in any case in variable length decoders decoding at least one codeword at a time and most probably, a block of bits contains more than one codeword, our objective is to develop a novel scheme for decoding all the codewords in a block of input data stream simultaneously. In order to have a flexible structure for design re-use, we are aiming at modular structure. The block size of the input data stream and the number of symbols are not to be set up fixed. Instead, they are to be parametrized for tailoring the structure according to application. E.g., for longer block of the input stream, more modules can be employed for decoding, and thus decode more symbols at a time.

In this chapter, a novel variable length decoding scheme is first introduced and illustrated with an example. Subsequently, a general hardware structure for the scheme is proposed with an illustration corresponding to the preceding example. Then, the studies are continued with a discussion on the evaluation of the critical path and performance of the resulting decoder. Finally, a brief summary about the features of the scheme concludes the chapter.

6.1 Algorithm

Let us assume K symbols and the corresponding binary codewords are collected into a codeword table as

$$c_k = (c_0^k, \ldots, c_{l_k-1}^k) | c_i^k \in \{0,1\}, k = 0, \ldots, K-1. \tag{108}$$

All the different codeword lengths in the codeword table can be combined into a set L defined as

$$L = \bigcup_{k=0}^{K-1} \{l_k\}. \tag{109}$$

Let the minimum and maximum codeword lengths be denoted by l_{min} and l_{max}, respectively. In addition, the maximum number of codewords with equal length is denoted by d_{max}. We use a group-based approach for storing the symbols into a symbol table; the symbols are grouped according to the length of the corresponding codeword and each group is stored into one page in the table. The size of the page is defined by d_{max}. In such an arrangement, the page where the symbol s_k is stored is determined by the length of its codeword, l_k. The symbols within a page are arranged in such a way that the offset within the page is determined by the least significant bits (LSB) of the codeword, $(c^k_{l_k - log_2 d_{max}}, \ldots, c^k_{l_k - 1})$.

The input data stream for the decoding process is an encoded binary vector X, i.e.,

$$X = (x_0, x_1, x_2, \ldots),\ x_i \in \{0, 1\}. \tag{110}$$

An N-bit sliding window B is used to extract bits from the input stream as

$$B = (b_0, b_1, \ldots, b_{N-1}),\ b_i = x_{idx+i},\ i = 0, \ldots, N-1 \tag{111}$$

where idx is the index to the first undecoded bit in the input stream X. Throughout the discussion, the sliding window B is assumed to be greater than the longest codeword, i.e., $N \geq l_{max}$.

We start the derivation of the algorithm by determining the maximum number of variable length codewords, M, in an N-bit sliding window B as

$$M = \lfloor N / l_{min} \rfloor. \tag{112}$$

Let us denote the codewords in the window by W_i where $i = 0, 1, \ldots, (M-1)$ and the length of codeword W_i by L_i. Moreover, let an index j_i, $0 \leq j_i \leq (N-1)$, define a location where the codeword W_i starts, i.e.,

$$W_i = (b_{j_i}, \ldots, b_{j_i + L_i - 1}). \tag{113}$$

Without losing generality, we may assume that the first codeword W_0 is always located at the beginning of the window, thus $j_0 = 0$. The second codeword W_1 is located immediately after the first L_0-bit codeword and, therefore, W_1 can be found starting from the index $j_1 = L_0$. This implies that the start index of the codeword W_i in B is the sum of the previous codeword lengths, i.e.,

$$j_i = \sum_{k=0}^{i-1} L_k. \tag{114}$$

However, the lengths of the codewords are not known in advance.

In order to avoid the recursive dependencies, a parallel search is needed to find codewords from “arbitrary” positions in the window. In general, all the candidates for indices j_i for the codeword W_i can be represented with the aid of set J_i defined recursively as

$$J_0 = 0;\ J_i = \{j_i | j_i = q + l, \forall q \in J_{i-1}, \forall l \in L\}, \tag{115}$$

which implies that a codeword can lie in any location in the window defined by a set J defined as

$$J = \bigcup_{k=0}^{M-1} J_k \tag{116}$$

Since the maximum length of the codeword, l_{max}, is known, we need to extract at most l_{max}-bit fields from the window B starting from all the locations defined by set J. In each bit field, the possible codeword is searched after by matching the bit field with all the possible codewords. When a match is found, the length of the codeword at position i in the window B, m_i, is returned as

$$m_i = \begin{cases} l_k\text{, if } \exists\, k : (b_i, b_{i+1}, \ldots, b_{i+l_k-1}) = (c_0^k, c_1^k, \ldots, c_{l_k-1}^k) \\ 0\text{ , otherwise} \end{cases} \tag{117}$$

where $i \in J, k = 0, 1, \ldots, K-1$.

The start index, j_i, of the each valid codeword W_i in the window can be defined with the aid of the lengths of the detected codewords. Correspondingly, the length of W_i is $L_i = m_{j_i}$. The symbol look-up is performed from the symbol table according to index A_i, which is formed by concatenating the length of the codeword and its LSBs. By returning the sum of all the valid codeword lengths, the input stream can be aligned for the next decoding iteration by updating the sliding window index, $idx' = idx + j_M$. The described procedure is iterated until the entire input stream is decoded.

Before giving a decoding example, let us summarize the described variable length decoding scheme. First, determine the parameters for decoding: the maximum number of codewords M that the N-bit window can hold and the set J, i.e., the locations where a codeword may lie. The decoding iteration can be outlined briefly as follows.

Codeword Detection. Extract $|J|$ bit fields of size at most l_{max} from the locations defined by set J. Detect a codeword from the beginning of each bit field and return the length m_i. Find the lengths of the valid codewords W_i according to indices j_i obtained by computing the sum of the previous valid lengths, and return sum of the valid codeword lengths L_i.

Symbol Look-Up. Form the index A_i and fetch the symbols from the symbol table.

Data Alignment. Align the input stream for the next decoding iteration.

6.1.1 Decoding Example

Let us assume that a codeword table depicted in Table 6(a) is used, thus the set of codeword lengths is defined as $L = \{2,3,4,5,6,7,8\}$ and the maximum number of codewords in a 16-bit window B is $M = 8$. In principle, the proposed approach would result in a 5-bit index to symbol table. However, the size of the symbol table can be easily decreased by noting that four LSBs are sufficient for each individual index. The resulting symbol table consisting of seven pages of two symbols is illustrated in Table 6(b). In the example case, a codeword can lie in 14 bit fields starting at locations $J = \{0,2,3,4,\ldots,14\}$ as illustrated with the aid of boxes below the window in Fig. 43. The fields at the end of the window are shorter than the others since the number of available bits in the window is less than $l_{max} = 8$.

All the fields are matched with all the codewords and the length and LSB of each detected codeword are returned. The detected codeword in the bit field is shown inside the corresponding box in Fig. 43. In the example case, the lengths of the codewords at the positions seven and eight in the window B are zero, which implies that the codewords were not detected. The fields containing a valid codeword are determined recursively using start indices j_i defined in (114). The first valid codeword W_0 is found from the first bit field at the beginning of the window, i.e., the first start index is $j_0 = 0$. The second codeword W_1 can be found in one of the seven fields starting at locations $J_1 = \{2,3,4,5,6,7,8\}$. Since the length of W_0 is $L_0 = 5$, the start index of W_1 is $j_1 = 5$. In Fig. 43, the detected valid codewords are marked with grey colour.

Table 6. *VLD decoding example: a) Codeword table and b) resulting symbol table.*

a)

s_k	c_k	l_k
A	10	2
B	11	2
C	011	3
D	0100	4
E	0101	4
F	00101	5
G	00110	5
H	000110	6
I	000111	6
J	0000110	7
K	00100101	8

b)

A: Page & Offset	s_k
000 0	
000 1	**K**
010 0	**A**
010 1	**B**
011 0	
011 1	**C**
100 0	**D**
100 1	**E**
101 0	**G**
101 1	**F**
110 0	**H**
110 1	**I**
111 0	**J**
111 1	

Index A_i for the symbol look-up is formed by concatenating the length and the LSB of the valid codeword. E.g., the length of W_1 is $L_1 = 4$ and the LSB of the W_1 is 0 and, therefore, index A_1 is 1000 and **D** is fetched from the symbol table.

6.2 General Structure

The design of the variable length decoder is started by considering the codeword detection of the valid codewords in a block of the input stream. The previously discussed sliding window B is realized as a N-bit codeword buffer and the codeword detection is performed by $|J|$ parallel codeword detector (CD) units. The input for each CD is a bit field of at most l_{max} bits, which is obtained from the codeword buffer locations in the set J defined in (116). All the CDs detect codewords simultaneously and return the length of the detected codeword. With this arrangement, the leftmost CDs up to location $N - l_{max}$ search after all the codewords in the codeword table while, for the remaining CDs, it is sufficient to detect only shorter codewords.

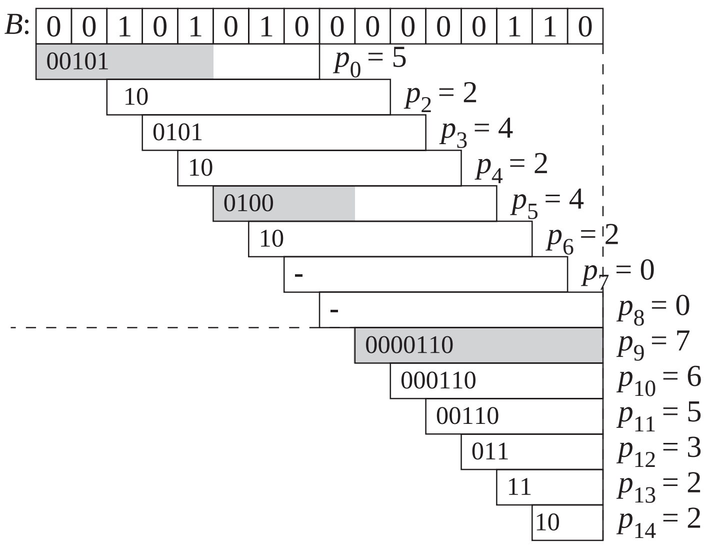


Fig. 43. *Principle of the proposed variable length decoding scheme.*

In order to select the valid codeword lengths, i.e., L_i, from the lengths of all the detected codewords, a cascade of multiplexers is employed as depicted in Fig. 44. Each multiplexer should have inputs, i.e., codeword lengths from all the CDs in the locations specified by J_i defined in (115). The first codeword length L_0 obtained from the leftmost CD starting at bit location $j_0 = 0$ controls the first multiplexer selecting the second valid codeword length L_1. Moreover, the output of the leftmost CD can be used to provide the decoding status, i.e., if the codeword length is zero, either the decoding is completed or an error has encountered. The other multiplexers are controlled by the sum of the previous codeword lengths according to (114). Hence, the computation of the sum of the valid codeword lengths creates the critical path as shown in Fig. 44.

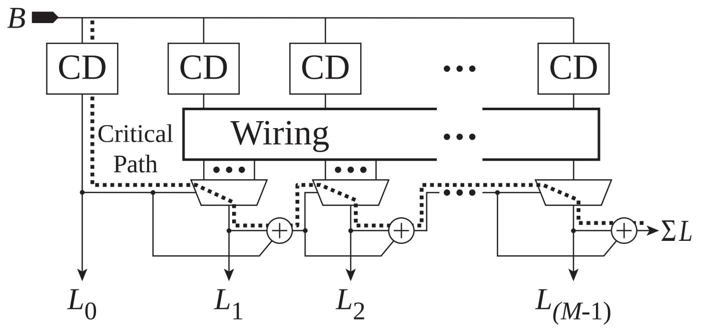


Fig. 44. *Block diagram of the generalized codeword detection.*

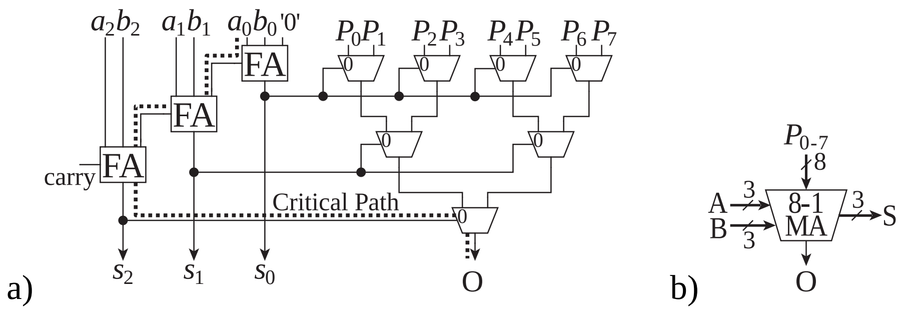


Fig. 45. Block diagram of the 8-to-1 multiplexed add unit: (a) structure and (b) corresponding symbol.

The critical path is shortened with a new multiplexed add (MA) unit shown in Fig. 45. In principle, the MA computes the sum of two input operands, A and B, and the resulting sum, S, is used to control a multiplexer which selects one of alternative inputs, P_i, to output O. In order to illustrate the operation of the MA, let us assume two 3-bit numbers $A = (a_2, a_1, a_0)$ and $B = (b_2, b_1, b_0)$. The sum denoted by $S = (s_2, s_1, s_0)$ controls the selection of the output O from inputs $P_0 - P_7$. Consequently, the output O can be expressed with the aid of sum of products as

$$\begin{aligned}
O =& P_0 \overline{s_2}\,\overline{s_1}\,\overline{s_0} + P_1 \overline{s_2}\,\overline{s_1} s_0 + P_2 \overline{s_2} s_1 \overline{s_0} + P_3 \overline{s_2} s_1 s_0 \\
& + P_4 s_2 \overline{s_1}\,\overline{s_0} + P_5 s_2 \overline{s_1} s_0 + P_6 s_2 s_1 \overline{s_0} + P_7 s_2 s_1 s_0 \\
=& \left(P_0 \overline{s_1}\,\overline{s_0} + P_1 \overline{s_1} s_0 + P_2 s_1 \overline{s_0} + P_3 s_1 s_0\right) \overline{s_2} \\
& + \left(P_4 \overline{s_1}\,\overline{s_0} + P_5 \overline{s_1} s_0 + P_6 s_1 \overline{s_0} + P_7 s_1 s_0\right) s_2 \\
=& \left[\left(P_0 \overline{s_0} + P_1 s_0\right) \overline{s_1} + \left(P_2 \overline{s_0} + P_3 s_0\right) s_1\right] \overline{s_2} \\
& + \left[\left(P_4 \overline{s_0} + P_5 s_0\right) \overline{s_1} + \left(P_6 \overline{s_0} + P_7 s_0\right) s_1\right] s_2 .
\end{aligned} \tag{118}$$

Closer examination of this decomposition reveals that each sum of products can be performed with the aid of 2-to-1 multiplexers. When MA is applied to the proposed VLD approach, the accumulated sum of the valid codeword lengths, i.e., the start index j_i, can be computed concurrently with the selection of current codeword length, L_i.

When the codeword length is known the symbol look-up is performed, i.e., a symbol corresponding to the valid codeword is fetched from the symbol table. The symbol table is mapped into a symbol memory. Consequently, the symbol look-up can be decomposed into two phases: address generation and symbol fetch as shown in Fig. 46. Briefly, the address generation is used to form an address to symbol table,

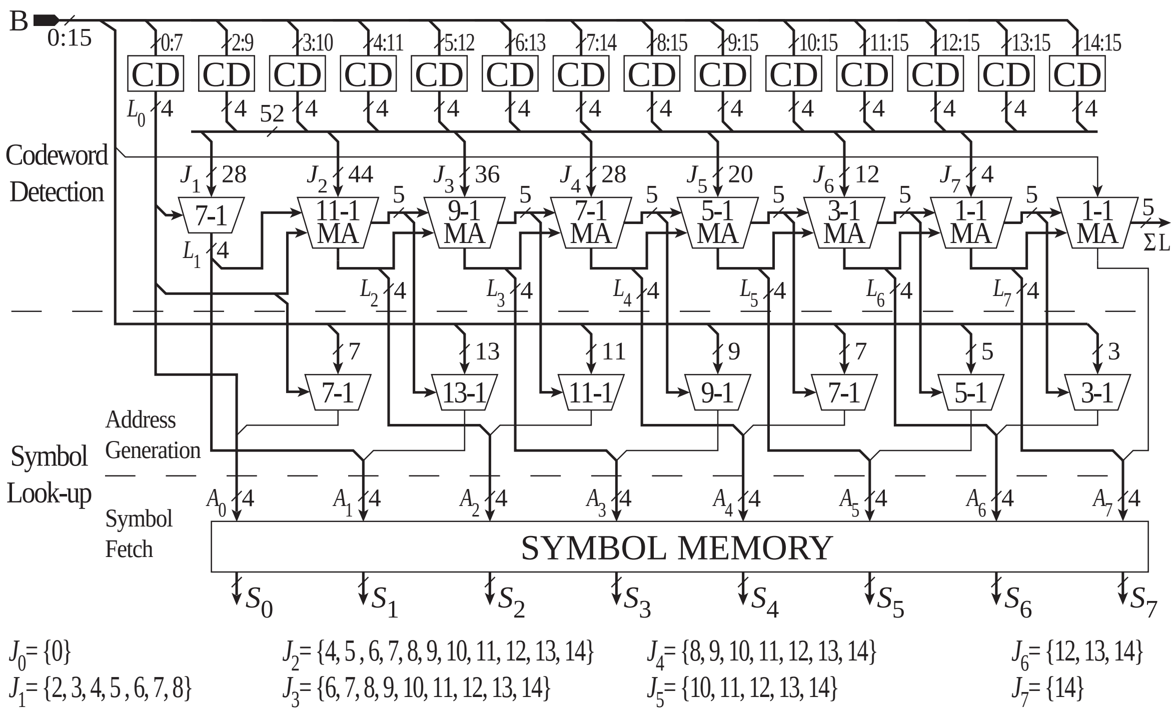


***Fig. 46.** Principal structure of entire variable length decoder.*

A_i, corresponding codeword W_i. The address A_i consists of the page and offset where page forms most significant part of A_i. The page is the length of W_i, L_i, obtained from MA units as seen in Fig. 46. The offset consists of the LSBs of the codeword, which can be determined according to start index of the next valid codeword j_{i+1}. If complex codeword tables are used, additional logic may be needed to form the page and offset. Finally, the symbol fetch is a trivial read memory operation. In order to support parallel symbol fetches, the symbol memory consists of separate parallel memory blocks, one for each decoder output, S_i.

6.2.1 Decoder example

The principal structure of the entire variable length decoder corresponding to the VLD example illustrated in Table 6 and Fig. 43 is depicted in Fig. 46. In the codeword detection, all the codewords in the 16-bit codeword buffer B are detected with 14 parallel CDs in defined locations. Each CD returns only the length of the detected codeword. The lengths of the valid codewords are selected with a 7-to-1 multiplexer and six cascaded 5-bit MAs. Each unit selecting L_i has lengths from the locations defined by set J_i. These locations are depicted on the left side of the input bus of the corresponding unit in Fig. 46. It should be noted that if no codeword matches the obtained bit field the MA returns zero, which is not, however, included into the number of alternatives denoted in the symbol of the MA. In the symbol look-up, the length of the valid codeword, L_i is used as a page. Since, the LSB of the codeword is enough to identify the codeword in Fig. 43(a), the LSB is extracted from the location $j_{i+1}-1$ and used as an offset. Note that the extraction of the offsets resembles the selection of valid codewords: multiplexing controlled by accumulated length. Due to this similarity, the MA can be used not only to compute the final sum but to select the offset corresponding to the last codeword $W_{(M-1)}$. Finally, the symbol S_i can be fetched from the memory according to address A_i.

6.3 Critical Path

According to Fig. 32, the length of the detected codewords is used to align the data in the codeword buffer. This feedback path forms the critical path since the alignment and codeword detection should be performed in a single cycle. Therefore, the critical path of the decoder in Fig. 46 consists of a CD unit, $|J_1|$-to-1 multiplexer, and a cascade of MA units. In order to approximate the critical path independent of technology, we use the interpretation from [8] where the delay is estimated with the aid of logical stages. A logical stage is assumed to be equivalent to 3-4 AND-OR (AO) and its delay is denoted by τ.

The number of AO stages in the CD unit is defined by the codeword table, which is application-specific. However, it is independent of N. Therefore, the delay of CD unit, t_D, is constant. The $|J_1|$-to-1 multiplexer contains $\lceil \log_2(|J_1|) \rceil$ AO stages, thus the corresponding delay can be estimated as

$$t_M = \lceil \log_2(|J_1|) \rceil \tau < \lceil \log_2(N+1) \rceil \tau. \tag{119}$$

The codeword buffer may contain at most M codewords, thus the complete decoder contains $(M-1)$ cascaded MA units. The critical path through MA as seen in Fig. 45 consists of $\lceil \log_2(N+1) \rceil$ full adders and a 2-to-1 multiplexer, thus the delay of MA is $t_{MA} = (\lceil \log_2(N+1) \rceil + 1)\tau$. Therefore, the delay of the critical path of the decoder,

t, is

$$\begin{aligned} t &\approx t_D + [\lceil \log_2(N+1) \rceil + (M-1)(\lceil \log_2(N+1) \rceil + 1)]\,\tau \\ &\approx t_D + [M(\lceil \log_2(N+1) \rceil + 1)]\,\tau. \end{aligned} \tag{120}$$

Although the variable M according to the definition in (112) is dependent on N, we may interpret that M defines the number of outputs of the decoder, i.e., the maximum number of codewords, which can be detected from the codeword buffer. Therefore, by decreasing M we may reduce the delay of the decoder. This implies that sometimes the codeword buffer may contain more codewords than we can decode, thus reducing the decoding rate. However, the loss of performance may be negligible since the probability that the codeword buffer contains the maximum number of codewords is low. The number of decoder outputs can be optimized for the given application, if the statistics of encoded data is available. This approach is used in our MPEG-2 demonstration discussed in the following chapter. Furthermore, if M is decreased and fixed, we find that the delay of the critical path is constant when $2^{n-1} \leq N < 2^n$ where n is an integer. This implies that the length of the codeword buffer should be chosen such that $N = 2^n - 1$. In this case, MA units are equipped with n full adders.

6.4 Summary

In this chapter, a novel multiple-symbol variable length decoding scheme has been proposed with the following properties; the scheme is parallel, decodes multiple symbols at a time, and exploits arbitrary codelength buffers and variable output rate. The proposed VLD scheme has been mapped onto a modular structure of which critical path has been optimized by reducing the number of logic levels with the aid of a new specific hardware mechanism called multiplexed add unit. Due to high modularity, the structure can be easily tailored according to the requirements of the application.

7. MPEG-2 VARIABLE LENGTH DECODING

In order to prove feasibility and estimate the performance and limiting factors of the proposed variable length decoding scheme, it has been applied to MPEG-2 video coding standard [45]. The previously proposed decoding scheme results in a variable input / variable output rate decoder and, therefore, the buffering resources are needed in the input as well as in the output. Our demonstration is targeted at an embedded system assuming external buffering and alignment resources. Therefore, only the kernel decoder design consisting of codeword detection and symbol look-up is considered.

In this chapter, the MPEG-2 demonstration implementation on an FPGA is described. First, the standard is outlined briefly in order to understand the fundamentals that affect to decoder design. Before modeling the structure, the specifications are determined according to the statistics of the benchmark scenes. Then, the performance of the resulting decoder is analyzed with different design parameters. The demonstration is concluded with the comparison to other FPGA-based variable length decoders and the discussion on related problems. Finally, the chapter is closed with a brief summation of the demonstration.

7.1 Overview to MPEG-2 Standard

Let us next outline briefly MPEG-2 standard from the variable length decoding point of view. According to the standard a video stream is constructed out of a sequence of pictures and each picture is processed in 8×8 blocks of pixels. Color is expressed in terms of luminance and chrominance components. There are three different chrominance formats: 4:2:0, 4:2:2, and 4:4:4 where four luminance blocks are followed by two, four, or eight chrominance blocks, respectively. These luminance and chrominance blocks construct a macroblock.

Compression in video is achieved by coding the pictures using information either from the same picture only or from neighboring pictures. These alternative compression techniques are referred to as intra and non-intra coding, respectively. Each block is transformed with the DCT resulting in a block of DCT coefficients. The first coefficient in a block is referred to as DC coefficient. The other coefficients are called AC coefficients. After the DCT, the blocks are quantized for reducing the number of bits required to represent the pixel values in frequency domain. The quantized values are then serialized and reordered into an one-dimensional array form with the aid of zig-zag scanning in order to construct longer sequences of zeros. The resulting sequence is coded with run-level symbols: the number of zero coefficients *run* preceding a non-zero coefficient *level*. In addition, a special symbol *end-of-block (EOB)* is used to denote the end of the one-dimensional array. Finally, the resulting stream of the symbols is ready for the VLC.

MPEG-2 defines two codeword tables B.12 and B.13 for coding the intra DC coefficients in the luminance and chrominance blocks, respectively. The codeword itself is a variable length code *dct_dc_size* and it is followed by a code of *dct_dc_size* bits indicating *dct_dc_differential* value which is returned from the decoder. All the other DCT coefficients are coded with the codeword table B.14 or B.15. Let us remark that the *level* is coded as a unsigned value and the sign is given as the LSB of the codeword. The correct codeword table is specified to the decoder with parameters *intra_vlc_format* and *macroblock_intra*. The *run* and *level* combinations without prespecified variable length code are coded with the aid of a 24-bit escape codeword *ESC* which is identified from a 6-bit prefix. The prefix is followed by a 6-bit fixed length code giving *run* and a 12-bit fixed length code providing the signed *level*.

7.2 Decoder Specification

Continuous preprocessed MPEG-2 data strings, which consist only of the variable length code of the DCT coefficients, have been chosen as the input for our implementation. Several encoded MPEG-2 data streams were analyzed and the obtained statistics are summarized in Table 7. This information has been used to derive the requirements for the demonstration decoder.

The minimum size for the codeword buffer is the length of the longest codeword, i.e., 24 bits in MPEG-2, which implies that the MA units must be equipped with at

Table 7. *Properties of the MPEG-2 benchmark scenes.*

Benchmark	Block type	b	W	B	b/W	W/31b
bat_327_334	I (B.15)	905 241	172 745	23 298	5.2	5.9
	NI	1 506 680	266 485	38 940	5.7	5.5
popplen	I (B.15)	242 795	47 003	4 572	5.2	6.0
	NI	153 265	28 069	4 139	5.5	5.7
sarnoff	I (B.14)	429 065	80 563	8 418	5.3	5.8
	NI	169 567	36 408	8 447	4.7	6.7
tennis	I (B.14)	57 741	12 345	2 718	4.7	6.6
	I (B.15)	613 066	120 754	9 504	5.1	6.1
	NI	989 235	137 756	25 524	7.2	4.3
t1cheer	I (B.15)	415 873	80 818	8 244	5.1	6.0
	NI	255 433	51 680	9 432	4.9	6.3
Total		5 737 961	1 034 626	143 236	5.5	5.6

b: bits. W: codewords. B: block. b/W: bits per codeword. W/31b: codewords in 31 bits.

least five full adders, i.e., $n = 5$. In the demonstration, we have used this minimum requirement. Therefore, the optimum size for the codeword buffer from the critical path point of view is $N = 31$. The 31-bit codeword buffer may contain at most 15 codewords but according to statistics in Table 7, 31-bit buffer can contain 5.6 codewords on average and, therefore, the number of decoder outputs, M, can be decreased for shortening the critical path. In our case, the average is rounded upwards and the number of outputs is $M = 6$.

7.3 Hardware Model

The variable length decoder supporting MPEG-2 has been described with behavioural-VHDL. Although we target at an FPGA technology, the VHDL description has been kept as technology independent as possible. The structure of demonstration implementation follows the general organization, i.e., the codeword detection and symbol look-up have been realized as illustrated in Fig. 46 but some MPEG-2-specific modifications were included. These modifications are described in the following.

vlcf	*format*
00	NI
01	NI
10	I/B.14
11	I/B.15

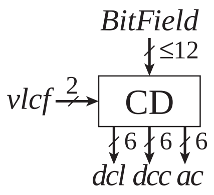


Fig. 47. *Block diagram of the MPEG-2 modified codeword detector.*

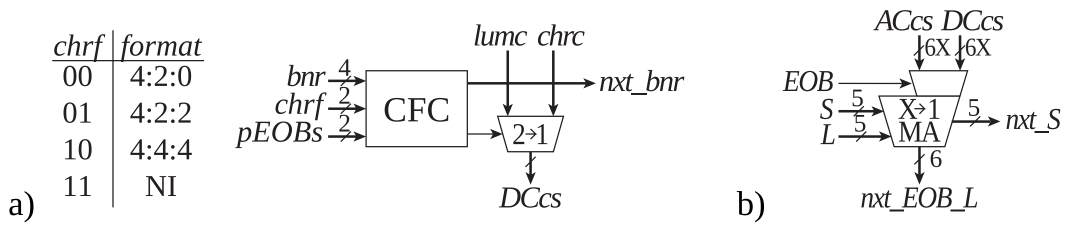
7.3.1 Codeword Detector

A codeword detector (CD) unit has at most a 12-bit input which is sufficient to detect all the MPEG-2 codewords from the minimum length of two bits to 24 bits. The CD returns three 6-bit values of a 5-bit codeword length and a 1-bit EOB status: two values for the DC coefficient and one value for the AC coefficient. The MPEG-2 standard defines four codeword tables, B.12 - B.15, and the selection of codeword table is controlled by a 2-bit VLC control signal *vlcf* which is the concatenation of the parameters *intra_vlc_format* and *macroblock_intra* defined in [45]. The symbol for the modified CD is depicted in Fig. 47.

The input *BitField* is checked for a possible codeword. If detected codeword represents EOB, the EOB status is set “true”. In the intra decoding, two DC values, *dcl* for luminance and *dcc* for chrominance are returned according to codeword tables B.12 and B.13, respectively. In the non-intra decoding, *dcc* represents the value of the DC coefficient. If a codeword is not detected from the bit field, zero-lengths are returned and EOB status is maintained as follows. The codeword represents a DC coefficient only if the previous codeword is EOB, thus EOB status is forced to “true”. Correspondingly, the codeword is an AC coefficient only if the previous codeword is not EOB and, therefore, EOB status is forced to “false”.

7.3.2 Chrominance Format Counter

A chrominance format counter (CFC) is used to select the correct group of the DC candidates out of two possible groups, i.e., chrominance candidates *chrc* and luminance candidates *lumc*. The realization is trivial; a counter returns a chrominance control signal *chr_ctrl* for the next block according to a current block number *bnr* in a macroblock as specified in [45]. The maximum block number is controlled by a parameter chrominance format *chrf*. The block number is upgraded when EOB is



chrf	format
00	4:2:0
01	4:2:2
10	4:4:4
11	NI

Fig. 48. *Block diagrams of the MPEG-2 specific units: (a) selection of DC coefficient and (b) modified MA.*

detected. In order to prevent the increase in the block number when EOB status is maintained, two previous EOB statuses given with *prEOBs* are checked. The schematic of the CFC is shown in Fig. 48(a) where *DCcs* denotes the correct DC candidates.

7.3.3 Multiplexed Add

The multiplexed add unit is modified to select also between the AC candidates *ACcs* and DC candidates *DCcs*. The candidates consist of the values from all the CDs defined by set J_i. The 2-to-1 multiplexing between AC and DC candidates is controlled by the previous EOB status *EOB* and it can be performed in parallel with the full adder computing the sum of the LSBs of the input operands. The symbol of the modified MA is illustrated in Fig. 48(b). Otherwise, the operation of the MA is similar to the principal operation, i.e., output *nxt_EOB_L* is selected according to the sum *nxt_S* of the previous sum *S* and the previous length *L*.

7.3.4 Memory Address Generator

A memory address generator (MAG) unit returns an 11-bit *MAG_code*, which may contain memory address or bits that are required for returning the symbol, for each codeword. In order to decode DC coefficient in intra decoding, 11 bits are extracted from the codeword buffer. The bits to be extracted are located according to intermediate sums.

The extracted bits are processed depending on the length and the interpretation of the codeword. If the codeword represents DC coefficient in intra decoding, it specifies the number of bits to be selected according to table B.12 or B.13 in [45]. The selected bits are extended to 11-bit *MAG_code* as a two's complement number. Otherwise, the

Table 8. *Memory address generation in the demonstration implementation.*

Length	Page	Offset
2, 3	000	01001
4	001	1100 & EB(7)
5	010	01 & T & EB(8:9)
6	010	0 & T & EB(6) & EB(8:9)
7	001	01 & T & EB(8:9)
8	010	T & EB(6:9)
9	001	T & EB(6:9)
10	000	01 & EB(8:9) & 0
11	000	0 & T & EB(7:9)
13	000	1 & EB(6:9)
14	011	0 & EB(6:9)
15	011	1 & EB(6:9)
16	100	0 & EB(6:9)
17	100	1 & EB(6:9)

extracted bits contain a complete or partial codeword, which is used to generate the address to the symbol memory.

In order to describe the memory mapping and address generation method used in the demonstration, let the extracted bits be enumerated from the left to the right and denoted as EB(0:10). Both the tables, B.14 and B.15, include at most 16 different codewords of a specific length and consequently, the identification of the codeword requires four bits. However, when combining the codeword tables and mapping them into unified memory, the chosen bits may identify two different codewords depending on the table. In order to distinguish the codewords in different tables, a table bit T defined as

$$T = \begin{cases} 1 & \text{, if B.15} \\ 0 & \text{, otherwise} \end{cases} \tag{121}$$

is used to specify the table. Altogether, a 3-bit page as well as the 5-bit offset are generated according to length as shown in Table 8. Although the sign bit is not needed to point the magnitude of the symbol stored into the memory, it should be propagated further for determining the correct level. Therefore, the memory address and sign are embedded into *MAG_code*.

Since only one codeword per cycle can represent symbol ESC in a 31-bit codeword buffer, a shared unit is utilized for extracting ESC and forwarding the 18-bit *ESC_Sym* consisting of possible symbol whose value is not predefined. Similarly, the EOB statuses are propagated further.

7.3.5 Symbol Fetch

A symbol fetch (SF) consist of three parallel dual-port memory banks and the resources to return the correct symbol. The symbols in the tables B.14 and B.15 excluding EOB and ESC are mapped into each memory bank. *MAG_code*s are read in rising clock edge. If the EOB status is true, it is returned and *run* and *level* are forced to zero. If the length of the codeword is equal to 24 implying ESC, a 6-bit *run* followed by a 12-bit signed *level* in *ESC_Sym* are returned. For the DC coefficient in the intra decoding mode, *run* is forced to zero and *MAG_code* is returned as a *level*. Otherwise, the symbol is read from the memory location defined by the address which is embedded into *MAG_code*. The predefined symbols stored in the memory can be represented with 11 bits, i.e., 5-bit *run* and the 6-bit unsigned value of *level*. Therefore, the *run* is extended to six bits and *level* is converted to 12-bit signed value before returning the actual 18-bit symbol.

7.3.6 Entire Decoder

The block diagram of the entire MPEG-2 decoder is illustrated in Fig. 49. The codeword detection consists of 29 CD units, which have inputs from buffer locations shown above the CDs. The seven leftmost CDs can detect all the possible codewords, next three CDs detect up to 21-bit codewords, and the remaining CDs detect only shorter codewords until the last or the rightmost CD detects only 2-bit codewords.

The first valid EOB and length, $EOB\&L_0$, is obtained from the leftmost CD but selection between the two DC candidates is needed introducing a 2-to-1 multiplexer controlled by chrominance control *pre_chr_ctrl* from the previous cycle. Similarly, a 2-to-1 multiplexer controlled by the EOB status *pre_EOB* from the previus cycle is employed to select between AC and DC candidates. The other $EOB\&L_i$ values are selected from CDs in buffer locations J_i. The correct DC candidates out of luminance and chrominance candidates are selected according the control provided by the corresponding CFC. A 2-to-1 multiplexer and one 21-to-1 multiplexer select $EOB\&L_1$

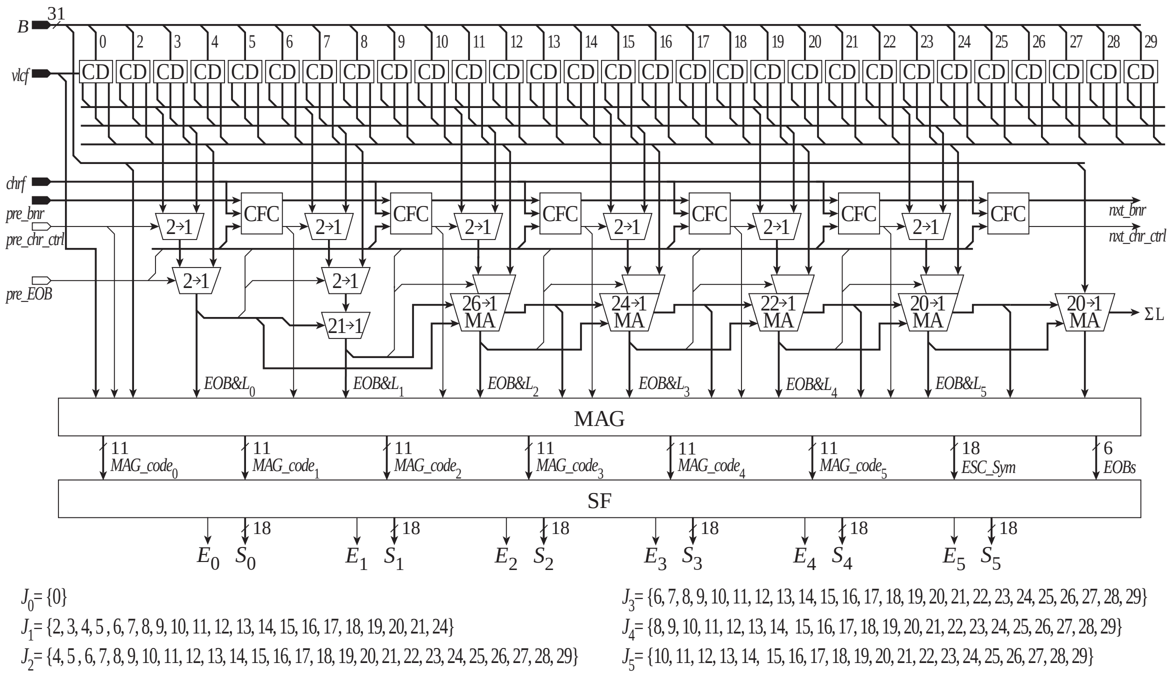

B
31
vlcf
CD
chrf
pre_bnr
pre_chr_ctrl
pre_EOB
2→1
CFC
21→1
26→1 MA
24→1 MA
22→1 MA
20→1 MA
nxt_bnr
nxt_chr_ctrl
ΣL
EOB&L0
EOB&L1
EOB&L2
EOB&L3
EOB&L4
EOB&L5
MAG
MAG_code0
MAG_code1
MAG_code2
MAG_code3
MAG_code4
MAG_code5
ESC_Sym
EOBs
SF
E0 S0
E1 S1
E2 S2
E3 S3
E4 S4
E5 S5
J0= {0}
J1= {2, 3, 4, 5 , 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 24}
J2= {4, 5 , 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29}
J3= {6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29}
J4= {8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29}
J5= {10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29}

Fig. 49. *Block diagram of the MPEG-2 variable length decoder.*

from AC and DC candidates. For the outputs $EOB\&L_2$–$EOB\&L_5$, the modified MAs are used to select valid values. The MA for the third output $EOB\&L_2$ is the most complex having candidates from 26 CDs. Let us remark that the rightmost MA is used to provide the extracted bits for the last codeword W_5 and to compute the final sum of the detected codeword lengths.

For the symbol look-up, the variable length coding format *vlcf*, chrominance controls, the EOB statuses, and lengths of the codewords are forwarded to the MAG with the intermediate sums in order to generate the *MAG_codes* for each codeword. Apart from *MAG_codes*, the MAG returns possible escape value *ESC_Sym* and the EOB statuses *EOBs*. During the symbol fetch, the EOB is interpreted according to the EOB status, which is also returned. The codeword representing the intra DC coefficient is determined from the most significant bit (MSB) of *vlcf* and the EOB status of the preceding codeword. The ESC can be identified from the MSBs of the length. Otherwise, the actual symbol is fetched from the symbol memory.

In general, the MPEG-2-specific modifications are not in the critical path, thus the discussion on decoder delay in the previous section applies to the demonstration. The generation of *MAG_codes*, except the last one, can be performed in parallel with the MAs and, therefore, the MAG is not a separate pipeline stage. However, the symbol fetch is pipelined since synchronous memories have been used.

7.4 Experimental Results

The proposed VLD scheme has been experimented with a parametrizable simulation model in Matlab and with an FPGA implementation. The simulation model is exploited to analyze the dependencies and behaviour of the scheme. The results are given in cycle domain meaning that information on timing or required resources is not considered. The FPGA implementation is used to prove the feasibility of the scheme and estimate the hardware complexity. The results are obtained by using Modelsim HDL simulator and Exemplar LeonardoSpectrum. The performance figures of the demonstration implementation are estimated in time domain.

The highest input rate is obtained when the codeword buffer can be completely updated at each cycle, i.e., if the accumulated length of the complete codewords in the buffer is equal to the buffer size. Assuming such an ideal data stream, the theoretical upper bound for the throughput is equal to buffer size divided by the average codeword length given in column ”W/31b” in Table 7. In practice, however, the buffer may contain a partial codeword, which cannot be detected at the current cycle. Therefore, it should be kept in the buffer and processed at the next cycle when the remaining bits are fetched into the buffer. When applying the proposed scheme to our benchmarks in Table 7, the effect of the buffer size to the throughput is illustrated in Fig. 50.

The number of outputs has been decreased in the demonstration implementation based on statistics and by recognizing the fact that the shorter codewords may not be decoded although they may exist in the buffer. The distribution of the codewords over decoder outputs, i.e., the proportion of cycles returning certain number of symbols, with different decoder configurations is illustrated in Fig. 51. The leftmost group “15 outputs” represents the theoretical approach, i.e., the scheme with a 31-bit codeword buffer and 15 decoder outputs. When experimented with the benchmark data, the proportion of cycles returning more than nine symbols is negligible. Therefore, the experimental results support the statistical conclusion to decrease the number of the decoder outputs.

In Fig. 51, the remarkable drop in proportion can be obtained after seven outputs. The resulting distribution over outputs “7 outputs” is balanced to return from 4 to 7 symbols but, on the other hand, the cycles with the largest proportion are returning five symbols, although with small difference. The balanced proportion between cycles is

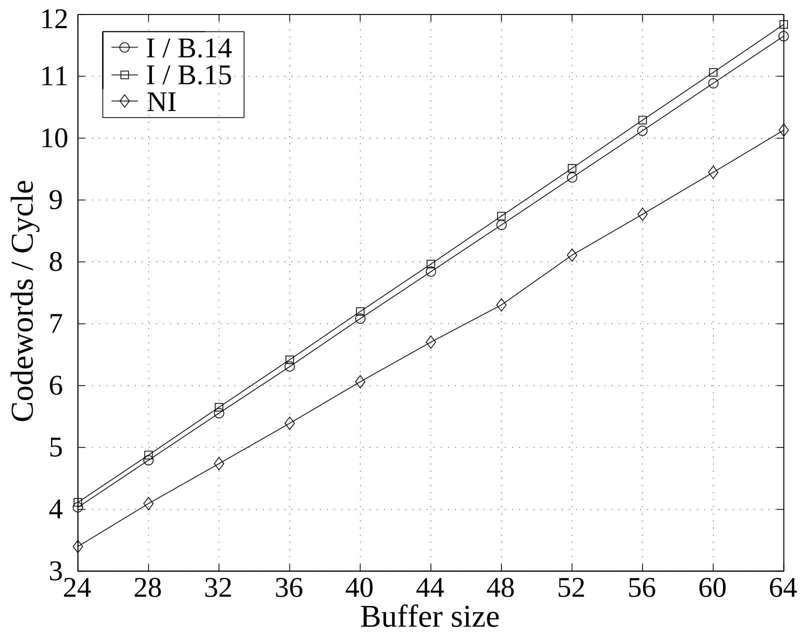


Fig. 50. *Throughput of the proposed approach.*

advantageous if the cycle time is predefined and seven codewords can be detected in the given cycle time. However, the detection of the seventh codeword may increase the critical path and the given cycle time is exceeded.

The distribution with six outputs, noted as “6 outputs”, represents our demonstration. The cycles with largest proportion are returning the maximum number of symbols, i.e., six symbols and the difference to the second largest proportion is already remarkable. Furthermore, the most of the cycles are decoding five or six codewords. In order to decode maximum number of codewords during the most of the cycles, the number of outputs should be restricted to five as shown with the group “5 outputs” in Fig. 51. When the number of outputs is decreased further, it is obvious that the largest proportion is increasing until symbol-serial decoders are returning one symbol per cycle with proportion of one. However, it should be noted that also the number of cycles required to complete decoding is increased and utilization of codeword buffer is decreased. Altogether, these effects are against our original objective.

The experimental results with scheme and demonstration implementation in cycle domain are summarized into Table 9. Column “Scheme” contains the practical upper bounds for the performance of the scheme with a 31-bit buffer and 15 outputs. The required cycles and achieved throughput for the implementation with a 31-bit buffer and 6 outputs are depicted in column “FPGA”. On average, 4.8 codewords per cycle

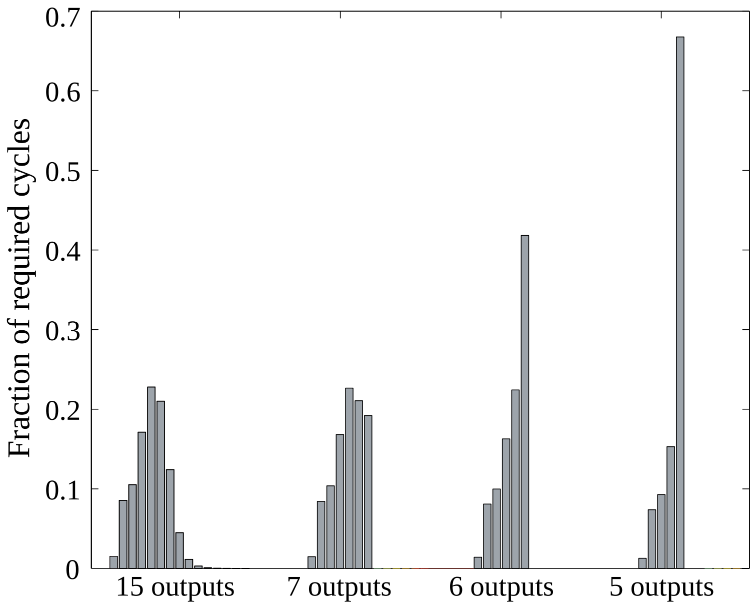


Fig. 51. *Distribution of symbols over the decoder outputs.*

are detected and decoded while the theoretical and practical throughputs in cycle domain are 5.6 and 5.0 codewords per cycle, respectively.

The previous discussion is based on behavioural models and the timing accuracy on unit cycles. However, the critical path defining the cycle time is an important measure for determining the absolute throughput, i.e., the amount of data processed in a time unit. In order to estimate the maximum clock frequency, the VHDL model of the demonstration implementation with the 31-bit buffer and 6 outputs has been synthesized on Xilinx Virtex-II FPGA (device 2V4000bf957) [125]. The CD units turn out to be application-specific pattern recognizers based on LUTs. The CFC is also based on LUTs while each MA is synthesized onto a 5-bit ripple carry adder parallel with a multiplexer tree. Consequently, the delay of each MA is about the same, i.e., the delays of five full adders and one 2-to-1 multiplexer, although the size of the multiplexer tree varies depending the number of candidates. When the entire design has been synthesized, 2 940 configurable logic blocks (CLB) out of 23 040 were allocated.

Three dual-port Block SelectRAM memories with 160 rows of 11 bits are generated using Xilinx CORE Generator for symbol memories. In an ideal memory mapping, each symbol has location of its own and the number of non-used locations and replicated symbols are zero. In such an case, a 7-bit address space is enough for 111

Table 9. *Experimental results of the MPEG-2 variable length decoder.*

		Scheme		FPGA	
Benchmark	Block type	C	W/C	C	W/C
bat_327_334	I (B.15)	32 104	5.4	33 526	5.2
	NI	54 843	4.9	56 780	4.7
popplen	I (B.15)	8 686	5.4	9 182	5.1
	NI	5 555	5.1	5 746	4.9
sarnoff	I (B.14)	15 296	5.3	16 198	5.0
	NI	6 052	6.0	6 823	5.3
tennis	I (B.14)	2 039	6.1	2 339	5.3
	I (B.15)	21 721	5.6	22 897	5.3
	NI	38 238	3.6	38 682	3.6
t1cheer	I (B.15)	14 730	5.5	15 500	5.2
	NI	9 113	5.7	9 824	5.3
Total		208 377	5.0	217 497	4.8

C: cycles. W/C: codewords per cycle.

different predefined symbols. In practice, however, many mapping functions results in non-used locations and some symbols are located in two different locations due to two different codewords representing same symbol. In order to ease the design work, the 8-bit address space has been used in the demonstration implementation. The synthesized design resulted in a critical path of 45.11 ns. The characteristics of the implementation are summarized in Table 10.

7.5 Discussion and Comparison

We would like to note that straightforward and fair comparison with other reported decoders is extremely difficult due to different implementation approaches. Standards, like JPEG [44] or MPEG-2 [45] with different codeword tables, set their own requirements for the decoder and, therefore, distinguish decoders from each other. Furthermore, input data with different compression ratios affect decoders with variable output rate; the less compression, the longer codewords resulting in decreased throughput. One main issue in comparing performance of different decoder implementations is how to equalize the effects of the different ASIC technologies or how to make different FPGAs and their specific features equivalent to each other or even to ASICs. However, the characteristic figures about the performance of the decoder

Table 10. *Characteristics of the MPEG-2 variable length decoder.*

Design platform	Xilinx Virtex-II FPGA
Application	MPEG-2
CLB count	4 940 / 23 040
Memory	3 dual-port memories with 160 rows of 11 bits
Virtex-II specifics	Block SelectRAM Memory
Frequency	22 MHz
Throughput	Variable 1-6 S/C Average 4.8 S/C 105 MS/s 585 Mbits/s

S/C: Symbols per cycle. MS/s: Million symbols per second.

are mostly given only for the chosen technology and without detailed variables. In other words, the results are technology dependent and consequently the performance of the chosen decoding technique is hidden behind technology. Altogether the decoding performance of the used technique is not directly comparable although all the previous aspects can be considered as critical design issues as well.

On the other hand, reconfigurable platforms provide fast design iteration times to change the design variables. Therefore, the variable input / output rate multiple-symbol decoder on the FPGA has been compared to other FPGA-based decoders presented in literature by using uniform implementation approaches. In other words, short design iteration times on FPGA allow the configuration of the proposed decoder to match the reference decoders or to provide at least the same features with the reference decoders. The behavioural non-optimized VHDL model of the decoder has been mapped onto the FPGAs used in the references in order to guarantee same technological features. The uniformity is guaranteed with same codeword tables, compression ratios, implementation platform, and synchronous design style.

7.5.1 Reference Decoders

Before moving on to discussion on the results of the comparison, let us briefly summarize the reference FPGA-based variable length decoders. All the decoders process data in a bit-parallel manner but from the symbol parallelism point of view they are different. In order to emphasize the difficulties in comparison with different technologies, let us remark the huge variation in the descriptive figures of the following decoders.

Aspar *et al.* in [4] reported a symbol-serial variable length decoder implemented on Altera's Flex 10K20RC240-4 and Flex 10K20RC240-3 FPGAs. The decoder is based on the decoder structure presented earlier by Lei and Sun in [62], which is illustrated in Fig. 35. The decoder has been designed to support up to 16-bit codewords according to JPEG standard [44]. The achieved operation frequencies are 9.91 MHz for 10K20RC240-4 and 11.54 MHz for 10K20RC240-3. In both platforms, the utilization of logic cells is 1145 out of 1152 logic cells.

Another FPGA-based symbol-serial decoder based on Lei and Sun's work is proposed by Jeon *et al.* in [49]. In order to reduce the processing time in the critical path, the decoder exploits a plane separation technique where input plane and OR plane performing the data buffering operate in parallel. The consecutive PLA-based matching process uses exactly the same method as previous decoder, i.e., the block of encoded bit stream is matched with all possible codewords stored into LUT. The decoder using the plane separation technique has been realized on Altera's Flex 8000 FPGA. When applying the presented technique to MPEG-2 intra-frame decoding [45], the throughput of 15 million symbols per second has been achieved. From logical resources point of view, about 30 % performance improvement from Lei and Sun's approach doubles the required resources [49].

Sima *et al.* in [101] considered an FPGA-augmented TriMedia processor running at 200 MHz [99]. When their approach presented in [100, 101] has been mapped onto Altera's ACEX EP1K100 FPGA, the decoder returning one symbol in Fig. 37 exhibits seven TriMedia cycles while two symbols can be returned in eight TriMedia cycles with the decoder in Fig. 42. Hence, the MPEG-2 compliant two-symbol decoder with the constant output rate yields the throughput of about 50 Msymbols/s or 275 Mbits/s assuming the data summarized in Table 7. The implementation requires all 12 Embedded Array Blocks, i.e., RAM blocks, and 51 % of the logic cells supporting either codeword table B.14 or B.15 in MPEG-2 standard.

In order to compare the proposed variable output rate multiple-symbol decoding technique, the non-optimized model of the decoder in [P7], where the specific features of implementation platform are *not* exploited, is mapped onto the same FPGA technologies as the references but without parallel symbol memories. The characteristics of the variable length decoders presented by Aspar *et al.* in [4], Jeon *et al.* in [49], and Sima *et al.* in [101] and the achieved results of the decoder [P7], which are obtained by using Exemplar LeonardoSpectrum, are summarized into three columns labeled as Altera Flex10K, Altera Flex8K, and Altera ACEX1K100 according to used FPGA in Table 11.

7.5.2 Comparison Results

Although our original objective has been to uniform the FPGA-based decoders based on different decoding approaches, clear differences and restrictions still exist. Both the multiple-symbol decoders, decoder presented by Sima *et al.* in [101] and the proposed decoder in [P7], require clearly more hardware resources than the symbol-serial decoders reported by Aspar *et al.* in [4] and Jeon *et al.* in [49]. Furthermore, the decoders in [4, 49] are independent whereas both the multiple-symbol decoders are clearly targeted at embedded system providing external resources for data buffering and alignment. The proposed decoder does not compete in hardware resources with other decoders due to the high degree of parallelism and it does not fit into all FPGAs that are used in the references. On the other hand, there are already enough resources available in the state of the art FPGAs and the integration density is increasing.

In the multiple-symbol decoding approaches used in the decoders in [101] and [P7], the critical path can be adjusted with requirements of the application by processing more data in a cycle; more time, more symbols per cycle. Therefore, they are advantageous when cycle time is specified according to environment. The critical path in the proposed decoder is dominated by the recursive selection of proper codewords and therefore, codeword properties reflecting to codeword detection delay have minor effect in total cycle time. In other words, the increase in the delay of the codeword detection has relatively small part in total cycle time.

The decoders in [4, 49, 101] have constant output rate resulting in constant throughput in terms of symbols whereas source data statistics reflect to throughput in the decoder [P7]. The decoder is sensitive to the compression ratio due to variable processing rate, i.e., the worse compression ratio implies longer codewords and therefore, less codewords in the buffer returning less symbols per cycle. The throughput values in Table 11 are estimated by assuming the average codeword length of 5.5 bits.

Table 11. *Comparison of the FPGA-based variable length decoders.*

	Altera Flex10K		Altera Flex8K		Altera ACEX1K100	
	Aspar *et al.* [4]	M[P7]	Jeon *et al.* [49]	[P7]	Sima *et al.* [101]	[P7]
Standard	JPEG	JPEG	MPEG-2 Intra	MPEG-2	MPEG-2	MPEG-2
CWT	K.5	K.3-K.6	Not known	B.12-B.15*	B.14 or B.15	B.14 or B.15
Logic cells	1 145	5 833	Not known	6 397	51 %	35 %
Frequency	11.54 MHz	4.8 MHz	15 MHz	4.2 MHz	25 MHz	12.1 MHz
Throughput	Constant 1 S/C 11.54 MS/s 63 Mbits/s	1-6 S/C avg. 4.8 S/C 23.04 MS/s 127 Mbits/s	Constant 1 S/C 15 MS/s 82 Mbits/s	1-6 S/C avg. 4.8 S/C 20 MS/s 111 Mbits/s	Constant 2 S/C 50 MS/s 275 Mbits/s	1-6 S/C avg. 4.8 S/C 58 MS/s 319 Mbits/s

CWT: Codeword tables. S/C: Symbols per cycle. MS/s: Million symbols per second. * Support for 4:2:0, 4:2:2, and 4:4:4 chrominance formats.

The properties of the decoders on Altera Flex10K supporting JPEG standard are collected into column labeled as Altera Flex10K in Table 11. The decoder in [P7] is configured to support JPEG standard and is referred just as M[P7] in the column. The structure of the decoder follows the structure of the MPEG-2 decoder in Fig. 49, only codeword tables are assumed to be typical Huffman tables from the standard [44]. The decoder requires about five times more hardware without symbol memories providing double throughput in time domain. In addition, the decoder supports four codeword tables.

The characteristics of the MPEG-2 decoders on Altera's Flex8K are summarized into the column Altera's Flex8K in Table 11. The MPEG-2 compliant multiple-symbol decoder illustrated in Fig. 49 is used in comparison although decoder in [49] supports only intra-frame decoding. This difference reflects to the critical path due to the different complexities of the codeword tables in LUT and CDs and larger multiplexers and MA units. However, the symbol-parallel decoder results in better throughput than symbol-serial decoder assuming 5.5-bit codewords on average, i.e., compression ratio of about 30%.

When comparing symbol-parallel decoders on ACEX EP1K100 FPGAs supporting codeword table B.14 or B.15 in MPEG-2, we assume again the compression ratio of 30%. The decoder in [P7] is simplified to support only a single codeword table, which is reflected in the required resources. The achieved results are given in the column ACEX EP1K100 FPGAs in Table 11. The throughput of the decoder in [101] is constant, two symbols per cycle. Instead, the decoder in [P7] is capable of returning up to six symbols per cycle although less symbols would imply shorter cycle time.

7.6 Summary

In this chapter, the proposed new VLD scheme and structure have been applied to MPEG-2 benchmark scenes for experimenting and estimating the behaviour and performance. The MPEG-2 variable length decoder demonstration has been described in VHDL and mapped onto Xilinx Virtex-II FPGA. The evaluated results indicate that 4.8 symbols out of the 5.6 average symbols present in the 31-bit buffer can be detected per cycle. The critical path of 45 ns proves the feasibility and potential of the approach. In order to illustrate the behaviour of the proposed decoder, its performance is analyzed with different design parameters.

The straightforward and fair comparison with other reported decoders is extremely difficult since the variable length decoders and consequently the reported figures are really application-specific. Besides, the multiple-symbol decoders are rarely reported. Nevertheless, we compared the demonstrated decoder to other FPGA-based decoders. With slight changes but without optimizing the decoder, the experimental results indicate that the proposed approach provides 16–100% better throughput at 2–3.6 times lower frequencies than referenced decoders. The achieved results exhibit also capability to cover a wide range of applications with this new decoding approach.

For concluding the chapter, one quite obvious golden rule can be emphasized; when presenting the method and estimating its performance with the aid of certain implementation, the performance of the method is extremely difficult to distinguish from the technological performance. Therefore, exact design variables or characteristics that are independent from technology should be also provided. Without these independent facts, the results can always be speculated. However, according to experimental results, the performance of the variable rate symbol-parallel approach can be considered promising for future applications.

8. CONCLUSIONS

In this Thesis, new application-specific parallel structures for the DCT and VLD have been described. The studies have been opened with a case study on DCT where the objectives have been to derive hardware oriented algorithm and map it onto an area-efficient parallel structure processing data in a sequential form at data rate. Therefore, the related work has been surveyed with respect to the cascaded pipeline computation of the DCT.

The guidelines to develop the algorithms have been outlined according to observations made in the survey. The derived novel regular perfect shuffle topology DCT algorithms for N-point and $N \times N$ transforms, $N = 2^k$, do not reach the lower bound on arithmetic complexity but the regularity introduces essential properties for the area-efficient hardware implementation and flexible use. The regular interconnections with smaller and smaller permutations from column to column reduces the complexity of data permutation. The distributed irregularities provide area-efficiency when unified pipelines are designed although, on the other hand, makes the node functions less regular.

Due to the regular topology, the algorithms lend themselves for vertical mapping resulting in area-efficient sequential structures with high modularity. The structures can be freely pipelined since all the arithmetic units are in feed forward paths. The additional pipelining allows the critical path to be shortened thus higher clock frequency can be used implying higher throughput. Let us remark that a sequential pipeline structure for an N-point transform supports also all the smaller transform sizes of powers of two and the structure for two-dimensional transform can also be used to compute corresponding one-dimensional transform.

The regularity of the algorithm and the modularity of the structure have been exploited when mapping the 8×8 DCT and its inverse onto a new common pipeline structure. The resulting structure for the 8×8 DCT/IDCT has been proven to be area-efficient compared to other reported solutions. The sequential demonstration implementation of the cascaded pipeline 8×8 DCT/IDCT is based on the data path model of the structure. When synthesized onto a 0.11 μm standard cell CMOS technology, the DCT/IDCT kernel occupies 39 424 equivalent 2-input NAND gates achieving the operation frequency of 253 MHz.

When considering the VLD, the objectives have been to break at least partially the recursive dependency related to the VLD, decode multiple symbols parallel at a time and exploit arbitrary code length buffers. Therefore, variable length decoders have been surveyed from the objectives point of view. Based on the made observations, the novel multiple-symbol VLD scheme has been derived. The scheme is capable of decoding all the complete codewords in an arbitrary length block of input data. The proposed VLD scheme has been mapped onto a general structure of which critical path has been optimized by reducing the number of logic levels with the aid of the new multiplexed add unit.

The proposed scheme is applied to MPEG-2 benchmark scenes for experimenting and estimating its behaviour and performance. It has been shown that the throughput rate of the scheme is proportional to the size of the codeword buffer. In order to prove the feasibility, structure has been demonstrated with an MPEG-2 compliant variable length decoder and its technology independent VHDL description is mapped onto Xilinx Virtex-II FPGA. The evaluated results indicate that 4.8 symbols out of the 5.6 average symbols present in the 31-bit buffer can be detected per cycle. The critical path of 45 ns proves the potential of the approach.

BIBLIOGRAPHY

[1] N. Ahmed, T. Natarajan, and K. R. Rao, “Discrete cosine transform,” *IEEE Transactions on Computers*, vol. 23, no. 1, pp. 90–93, Jan. 1974.

[2] D. Akopian, “Systematic approach to parallel architectures for DSP algorithms,” Dr.Tech. Thesis, Tampere University of Technology, Tampere, Finland, Sept. 1997.

[3] Y. Arai, T. Agui, and M. Nakajima, “A fast DCT-SQ scheme for images,” *Transactions of the IEICE*, vol. E71, no. 11, pp. 1095–1097, Nov. 1988.

[4] Z. Aspar, Z. M. Yusof, and I. Suleiman, “Parallel Huffman decoder with an optimized look up table option on FPGA,” in *Proc. TENCON 2000*, vol. 1, Kuala Lumpur, Malaysia, Sep. 24–27 2000, pp. 73–76.

[5] J. Astola and D. Akopian, “Architecture-oriented regular algorithms for discrete sine and cosine transforms,” *IEEE Transactions on Signal Processing*, vol. 47, no. 4, pp. 1109–1124, Apr. 1999.

[6] J. C. Carlach, P. Penard, and J. L. Sicre, “TCAD: a 27 MHz 8x8 discrete cosine transform chip,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Glasgow, UK, May 23–26 1989, pp. 2429–2432.

[7] S. C. Chan and K. L. Ho, “A new two-dimensional fast cosine transform algorithm,” *IEEE Transactions on Signal Processing*, vol. 39, no. 2, pp. 481–485, Feb. 1991.

[8] C.-J. Chang, S. Vassiliadis, and J. G. Delgado-Frias, “An investigation of binary CLA and ripple CMOS adder designs,” *Microprocessing and Microprogramming J.*, vol. 40, no. 1, pp. 1–21, Jan. 1994.

[9] S.-F. Chang and D. G. Messerschmitt, “VLSI designs for high-speed Huffman decoder,” in *Proceedings of the IEEE International Conference on Computer Design*, Cambridge, MA, U.S.A., Oct. 14–16 1991, pp. 500–503.

[10] ——, “Designing high-throughput VLC decoder Part I - Concurrent VLSI architectures,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 2, pp. 187–196, June 1992.

[11] W. H. Chen, C. H. Smith, and S. C. Fralick, “A fast computational algorithm for the discrete cosine transform,” *IEEE Transactions on Communications*, vol. 25, no. 9, pp. 1004–1009, Sept. 1977.

[12] K.-H. Cheng, C.-S. Huang, and C.-P. Lin, “The design and implementation of DCT/IDCT chip with novel architecture,” in *Proceedings of the IEEE International Symposium on Circuits and Systems*, Geneva, Switzerland, May 28–31 2000, pp. 741–744.

[13] N. I. Cho and S. U. Lee, “Fast algorithm and implementation of 2-D discrete cosine transform,” *IEEE Transactions on Circuits and Systems*, vol. 38, no. 3, pp. 297–305, Mar. 1991.

[14] ——, “A fast 4×4 DCT algorithm for the recursive 2-D DCT,” *IEEE Transactions on Signal Processing*, vol. 40, no. 9, pp. 2166–2173, Sept. 1992.

[15] N. I. Cho, I. D. Yun, and S. U. Lee, “On the regular structure for the fast 2-D DCT algorithm,” *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, vol. 40, no. 4, pp. 259–266, Apr. 1993.

[16] S. H. Cho, T. Xanthopoulos, and A. P. Chandrakasan, “A low power variable length decoder for MPEG-2 based on nonuniform fine-grain table partitioning,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 2, pp. 249–257, June 1999.

[17] S. B. Choi and M. H. Lee, “High speed pattern matching for a fast Huffman decoder,” *IEEE Transactions on Consumer Electronics*, vol. 41, no. 1, pp. 97–103, Feb. 1995.

[18] K. Compton and S. Hauck, “Reconfigurable computing: A survey of systems and software,” *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, June 2002.

[19] Z. Cvetković and M. V. Popović, “New fast recursive algorithms for the computation of discrete cosine and sine transforms,” *IEEE Transactions on Signal Processing*, vol. 40, no. 8, pp. 2083–2086, Aug. 1992.

[20] M. Davio, “Kronecker products and shuffle algebra,” *IEEE Transactions on Computers*, vol. 30, no. 2, pp. 116–125, Feb. 1981.

[21] P. Duhamel and C. Guillemot, “Polynomial transform computation of the 2-D DCT,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, Albuquerque, NM, U.S.A., Apr. 3–6 1990, pp. 1515–1518.

[22] P. Duhamel and H. H’Mida, “New 2^n DCT algorithms suitable for VLSI implementation,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 12, Dallas, TX, U.S.A., Apr. 6–9 1987, pp. 1805–1808.

[23] J. G. G. Langdon, “An introduction to arithmetic coding,” *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135–149, Mar. 1984.

[24] L. Geppert, “Devices and circuits [Technology 2000 analysis and forecast],” *IEEE Spectrum*, vol. 37, no. 1, pp. 63–69, Jan. 2000.

[25] J. Granata, M. Conner, and R. Tolimieri, “Recursive fast algorithms and the role of the tensor product,” *IEEE Transactions on Signal Processing*, vol. 40, no. 12, pp. 2921–2930, Dec. 1992.

[26] H. L. Groginsky and G. A. Works, “A pipeline fast Fourier transform,” *IEEE Transactions on Computers*, vol. 19, no. 11, pp. 1015–1019, Nov. 1970.

[27] M. A. Haque, “Two-dimensional fast cosine transform,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 33, no. 6, pp. 1532–1539, Dec. 1985.

[28] R. M. Haralick, “A storage efficient way to implement the discrete cosine transform,” *IEEE Transactions on Computers*, vol. 25, no. 7, pp. 764–765, July 1976.

[29] R. Hashemian, “High speed search and memory efficient Huffman coding,” in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 1, Chicago, IL, U.S.A., May 3–6 1993, pp. 287–290.

[30] ——, “Design and hardware implementation of a memory efficient Huffman decoding,” *IEEE Transactions on Consumer Electronics*, vol. 40, no. 3, pp. 345–352, Aug. 1994.

[31] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital video: an introduction to MPEG-2*. Norwell, MA, U.S.A.: Kluwer Academic Publishers, 1997.

[32] J. Henkel, “Closing the SoC design gap,” *IEEE Computer*, vol. 36, no. 9, pp. 119–121, Sept. 2003.

[33] H. S. Hou, “A fast recursive algorithm for computing the discrete cosine transform,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 10, pp. 1455–1461, Oct. 1987.

[34] S.-F. Hsiao, W.-R. Shiue, and J.-M. Tseng, “A cost-efficient fully-pipelinable architecture for DCT/IDCT,” *IEEE Transactions on Consumer Electronics*, vol. 45, no. 3, pp. 515–525, Aug. 1999.

[35] S.-F. Hsiao and J.-M. Tseng, “Parallel, pipelined and folded architectures for computation of 1-D and 2-D DCT in image and video codec,” *Journal of VLSI Signal Processing*, vol. 28, no. 3, pp. 205–220, Jul. 2001.

[36] ——, “New matrix formulation for two-dimensional DCT/IDCT computation and its distributed-memory VLSI implementation,” *IEE Proceedings - Vision, Image and Signal Processing*, vol. 149, no. 2, pp. 97–107, Apr. 2002.

[37] C.-T. Hsieh and S. P. Kim, “A concurrent memory-efficient VLC decoder for MPEG applications,” *IEEE Transactions on Consumer Electronics*, vol. 42, no. 3, pp. 439–446, Aug. 1996.

[38] C.-T. T. Hsieh, “The systematic approach for concurrent VLC decoder,” *IEEE Transactions on Consumer Electronics*, vol. 43, no. 3, pp. 918–924, Aug. 1997.

[39] D. A. Huffman, “A method for the construction of minimum-redundancy codes,” *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, Sept. 1952.

[40] IEEE Std 1180-1990, “IEEE standard specification for the implementations of 8x8 inverse discrete cosine transform,” Institute of Electrical and Electronics Engineers, New York, USA, International Standard, Dec. 1990.

[41] International Organization for Standardization, “Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 2: Video , ISO/IEC International Standard 11172-2:1993,” 1993.

[42] ——, “Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines, ISO/IEC International Standard 10918-1:1994,” 1994.

[43] ——, “Information technology – Generic coding of moving pictures and associated audio information: Video, ISO/IEC International Standard 13818-2:2000,” 2000.

[44] International Telecommunication Union, “Information technology – Digital compression and coding of continuous-tone still images – requirements and guidelines, CCITT Recommendation T.81,” Sept. 1992.

[45] ——, “Information technology – Generic coding of moving pictures and associated audio information: Video, ITU-T Recommendation H.262,” Feb. 2000.

[46] (2003) International Technology Roadmap for Semiconductors 2003 Edition. [Online]. Available: http://public.itrs.net

[47] A. K. Jain, “A sinusoidal family of unitary transforms,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, no. 4, pp. 356–365, Oct. 1979.

[48] T. Järvinen, J. Takala, and J. Saarinen, “Unified architecture for discrete fourier and cosine transform,” in *Advances in System Science: Measurements, Circuits and Control*, N. E. Mastorakis and L. A. Pecorelli-Peres, Eds. New York, NY, U.S.A.: WSES Press, 2001, pp. 301–306.

[49] J. H. Jeon, Y. S. Park, and H. W. Park, “A fast variable-length decoder using plane separation,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 5, pp. 806–812, Aug. 2000.

[50] D. Johnson, V. Akella, and B. Stott, “Micropipelined asynchronous discrete cosine transform (DCT) processor,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 4, pp. 731–740, Dec. 1998.

[51] F. A. Kamangar and K. R. Rao, “Fast algorithms for the 2-D discrete cosine transform,” *IEEE Transactions on Computers*, vol. 31, no. 9, pp. 899–906, Sept. 1982.

[52] Y. Katayama, T. Kitsuki, and Y. Ooi, “A block processing unit in single-chip MPEG-2 video encoder LSI,” *Journal of VLSI Signal Processing*, vol. 22, no. 1, pp. 59–64, Aug. 1999.

[53] S. Kinouchi and A. Sawada, “Huffman code decoding circuit,” U.S. Patent 5 617 089, Apr. 1 1997.

[54] M. Kovac and N. Ranganathan, “JAGUAR: A fully pipelined VLSI architecture for JPEG image compression standard,” *Proceedings of the IEEE*, vol. 83, no. 2, pp. 247–258, Feb. 1995.

[55] J. Kwak and J. You, “One- and two-dimensional constant geometry fast cosine transform algorithms and architectures,” *IEEE Transactions on Signal Processing*, vol. 47, no. 7, pp. 2023–2034, July 1999.

[56] B. G. Lee, “A new algorithm for the discrete cosine transform,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 6, pp. 1243–1245, Dec. 1984.

[57] P.-Z. Lee and F.-Y. Huang, “Restructured recursive DCT and DST algorithms,” *IEEE Transactions on Signal Processing*, vol. 42, no. 7, pp. 1600–1609, July 1994.

[58] S.-W. Lee and I.-C. Park, “A low-power variable length decoder for MPEG-2 based on successive decoding of short codewords,” *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, vol. 50, no. 2, pp. 73–82, Feb. 2003.

[59] Y.-P. Lee, T.-H. Chen, L.-G. Chen, M.-J. Chen, and C.-W. Ku, “A cost-effective architecture for 8 $\times$ 8 two-dimensional DCT/IDCT using direct method,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 3, pp. 459–467, June 1997.

[60] Y.-S. Lee, J.-J. Jong, T.-S. Perng, L.-C. Hsu, M.-Y. Jaw, and C.-Y. Li, “A memory-based architecture for very-high-throughput variable length codec design,” in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 3, Hong Kong, June 9–12 1997, pp. 2096–2099.

[61] Y.-S. Lee, B.-J. Shieh, and C.-Y. Lee, “A generalized prediction method for modified memory-based high throughput VLC decoder design,” *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, vol. 46, no. 6, pp. 742–754, June 1999.

[62] S. M. Lei and M. T. Sun, “An entropy coding system for digital HDTV applications,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 1, no. 1, pp. 147–155, Mar. 1991.

[63] H. Lim, V. Piuri, and E. E. Swartzlander, “A serial-parallel architecture for two-dimensional discrete cosine and inverse discrete cosine transforms,” *IEEE Transactions on Computers*, vol. 49, no. 12, pp. 1297–1309, Dec. 2000.

[64] C.-H. Lin and C. W. Jen, “Low power parallel Huffman decoding,” *Electronics Letters*, vol. 34, no. 3, pp. 240–241, Feb. 1998.

[65] H.-D. Lin and D. G. Messerschmitt, “High throughput reconstruction of Huffman-coded images,” in *Proceedings of the IEEE International Conference on Computer Design*, Cambridge, MA, U.S.A., Oct. 2–4 1989, pp. 172–175.

[66] ——, “Designing a high-throughput VLC decoder Part II - Parallel decoding methods,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 2, pp. 197–206, June 1992.

[67] C. Loeffler, A. Ligtenberg, and G. S. Moschytz, “Practical fast 1-D DCT algorithms with 11 multiplications,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, Glasgow, UK, May 23–26 1989, pp. 988–991.

[68] A. Madisetti and A. N. Wilson, “A 100 MHz 2-D 8$\times$8 DCT/IDCT processor for HDTV applications,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 2, pp. 158–165, Apr. 1995.

[69] J. Makhoul, “A fast cosine transform in one and two dimensions,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 1, pp. 27–34, Feb. 1980.

[70] H. S. Malvar, “Fast computation of the discrete cosine transform through the fast Hartley transform,” *Electronics Letters*, vol. 22, no. 7, pp. 352–353, Mar. 1986.

[71] B. Martin, “Electronic design automation [1999 technology analysis and forecast],” *IEEE Spectrum*, vol. 36, no. 1, pp. 57–61, Jan. 1999.

[72] M. Matsui, H. Hara, Y. Uetani, L.-S. Kim, T. Nagamatsu, Y. Watanabe, A. Chiba, K. Matsuda, and T. Sakurai, “A 200 MHz 13 mm^2 2-D DCT macrocell using sense-amplifying pipeline flip-flop scheme,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 29, no. 12, pp. 1482–1490, Dec. 1994.

[73] J. C. Maxted and J. P. Robinson, “Error recovery for variable length codes,” *IEEE Transactions on Information Theory*, vol. 31, no. 6, pp. 794–801, Nov. 1985.

[74] J. L. Mitchell, W. B. Pennebaker, C. H. Fogg, and D. J. LeGall, *MPEG video compression standard*. Norwell, MA, U.S.A.: Kluwer Academic Publishers, 1996.

[75] T. K. Moon and W. C. Stirling, *Mathematical Methods and Algorithms for Signal Processing*. Upper Saddle River, NJ, U.S.A.: Prentice Hall, Inc., 2000.

[76] A. Mukherjee, H. Bheda, M. A. Bassiouni, and T. Acharya, “Multibit decoding/encoding of binary codes using memory based architectures,” in *Proceedings of the IEEE Data Compression Conference*, Snowbird, UT, U.S.A., Apr. 8–11 1991, pp. 352–361.

[77] A. Mukherjee, N. Ranganathan, and M. Bassiouni, “Efficient VLSI designs for data transformation of tree-based codes,” *IEEE Transactions on Circuits and Systems*, vol. 38, no. 2, pp. 306–314, Mar. 1991.

[78] A. Mukherjee, N. Ranganathan, J. W. Flieder, and T. Acharya, “MARVLE: A VLSI chip for data compression using tree-based codes,” *IEEE Transactions*

on Very Large Scale Integration (VLSI) Systems, vol. 1, no. 2, pp. 203–214, June 1993.

[79] M. J. Narasimha and A. M. Peterson, “On the computation of the discrete cosine transform,” *IEEE Transactions on Communications*, vol. 26, no. 6, pp. 934–936, June 1978.

[80] Y. Ooi, A. Taniguchi, and S. Demura, “A 162Mbit/s variable length decoding circuit using an adaptive tree search technique,” in *Proceedings of the IEEE Custom Integrated Circuits Conference*, San Diego, CA, U.S.A., May 1–4 1994, pp. 107–110.

[81] H. Park and V. K. Prasanna, “Area efficient VLSI architectures for Huffman coding,” *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, vol. 40, no. 9, pp. 568–575, Sept. 1993.

[82] H. Park, J.-C. Son, and S.-R. Cho, “Area efficient fast Huffman decoder for multimedia applications,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, Detroit, MI, U.S.A., May 9–12 1995, pp. 3279–3281.

[83] Y.-G. Park, “High speed variable length code decoding apparatus,” U.S. Patent 5 561 690, Oct. 1 1996.

[84] P. Pirsch, *Architectures for Digital Signal Processing*. Chichester, United Kingdom: John Wiley & Sons, Ltd., 1998.

[85] J. M. Rabaey, W. Gass, R. Brodersen, and T. Nishitani, “VLSI design and implementation fuels the signal-processing revolution,” *IEEE Signal Processing Magazine*, vol. 15, no. 1, pp. 22–37, Jan. 1998.

[86] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, and Applications*. San Diego, CA, U.S.A.: Academic Press, 1990.

[87] S. K. Rao and T. Kailath, “Regular iterative algorithms and their implementation on processor arrays,” *Proceedings of the IEEE*, vol. 76, no. 3, pp. 259–269, Mar. 1988.

[88] J. Rissanen and J. G. G. Langdon, “Arithmetic coding,” *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, Mar. 1979.

[89] J. J. Rissanen, "Generalized Kraft inequality and arithmetic coding," *IBM Journal of Research and Development*, vol. 20, no. 3, pp. 198–203, May 1976.

[90] M. K. Rudberg and L. Wanhammar, "New approaches to high speed Huffman decoding," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 2, Atlanta, U.S.A., May 12–15 1996, pp. 149–152.

[91] ——, "High speed pipelined parallel Huffman decoding," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 3, Hong Kong, June 9–12 1997, pp. 2080–2083.

[92] C. L. Seitz, "Concurrent VLSI systems," *IEEE Transactions on Computers*, vol. 33, no. 12, pp. 1247–1265, Dec. 1984.

[93] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical J.*, vol. 27, pp. 379–423, 623–656, July, Oct. 1948.

[94] Y. Q. Shi and H. Sun, *Image and video compression for multimedia engineering: fundamentals, algorithms, and standards*. Boca Raton, FL, U.S.A.: CRC Press LLC, 1999.

[95] B.-J. Shieh, Y.-S. Lee, and C.-Y. Lee, "A high-throughput memory-based VLC decoder with codeword boundary prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 8, pp. 1514–1521, Dec. 2000.

[96] ——, "A new approach of group-based VLC codec system with full table programmability," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 2, pp. 210–221, Feb. 2001.

[97] C. B. Shung, H.-D. Lin, R. Cybher, P. H. Siegel, and H. K. Thapar, "Area-efficient architectures for Viterbi algorithm II. Applications," *IEEE Transactions on Communications*, vol. 41, no. 5, pp. 802–807, May 1993.

[98] C. B. Shung, H.-D. Lin, R. Cypher, P. H. Siegel, and H. K. Thapar, "Area-efficient architectures for the Viterbi algorithm I. Theory," *IEEE Transactions on Communications*, vol. 41, no. 4, pp. 636–644, Apr. 1993.

[99] M. Sima, personal correspondence, Apr. 2003.

[100] M. Sima, S. Cotofana, S. Vassiliadis, J. T. J. van Eijndhoven, and K. Visser, “MPEG macroblock parsing and pel reconstruction on an FPGA-augmented TriMedia processor,” in *Proceedings of the IEEE International Conference on Computer Design*, Austin, Texas, U.S.A., Sep. 24–26 2001, pp. 425–430.

[101] ——, “MPEG-compliant entropy decoding on FPGA-augmented TriMedia/CPU64,” in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, U.S.A., Apr. 21–24 2002.

[102] H. Stone, “Parallel processing with perfect shuffle,” *IEEE Transactions on Computers*, vol. 20, no. 2, pp. 153–161, Feb. 1971.

[103] M.-T. Sun, “VLSI architecture and implementation of a high-speed entropy decoder,” in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 1, Singapore, June 11–14 1991, pp. 200–203.

[104] J. Takala, “Real-time digital signal processing systems: parallel algorithms and architectures,” Dr.Tech. Thesis, Tampere University of Technology, Tampere, Finland, Aug. 1999.

[105] J. Takala, D. Akopian, J. Astola, and J. Saarinen, “Constant geometry algorithm for discrete cosine transform,” *IEEE Transactions on Signal Processing*, vol. 48, no. 6, pp. 1840–1843, June 2000.

[106] ——, “Scalable interconnection networks for partial column array processor architectures,” in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. IV, Geneva, Switzerland, May 28–31 2000, pp. 513–516.

[107] J. Takala and T. Järvinen, “Stride permutation access in interleaved memory systems,” in *Domain-Specific Multiprocessors - Systems, Architectures, Modeling, and Simulation*, S. S. Bhattacharyya, E. F. Deprettere, and J. Teich, Eds. New York, NY, U.S.A.: Marcel Dekker, Inc., 2004, ch. 4, pp. 63–84.

[108] J. Takala, J. Nikara, D. Akopian, J. Astola, and J. Saarinen, “Pipeline architecture for 8×8 discrete cosine transform,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, Istanbul, Turkey, June 5–9 2000, pp. 3303–3306.

[109] J. H. Takala, T. S. Järvinen, and J. A. Nikara, “Register-based reordering networks for matrix transpose,” in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 4, Phoenix, AZ, U.S.A., May 26–29 2002, pp. 874–877.

[110] T. C. Tan, G. Bi, Y. Zeng, and H. N. Tan, “DCT hardware structure for sequentially presented data,” *Signal Processing*, vol. 81, no. 11, pp. 2333–2342, Nov. 2001.

[111] B. D. Tseng and W. C. Miller, “On computing the discrete cosine transform,” *IEEE Transactions on Computers*, vol. 27, no. 10, pp. 966–968, Oct. 1978.

[112] S. Uramoto, Y. Inoue, A. Takabatake, J. Takeda, Y. Tamashita, H. Terane, and M. Yoshimoto, “A 100-MHz 2-D discrete cosine transform core processor,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 27, no. 4, pp. 492–499, Apr. 1992.

[113] S. Venkataraman, V. R. Kanchan, K. R. Rao, and M. Mohanty, “Discrete transforms via the Walsh-Hadamard transform,” *Signal Processing*, vol. 14, no. 4, pp. 371–382, June 1988.

[114] M. Vetterli, “Fast 2-D discrete cosine transform,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 10, Tampa, FL, U.S.A., Mar. 26–29 1985, pp. 1538–1541.

[115] M. Vetterli and H. Nussbaumer, “Simple FFT and DCT algorithms with reduced number of operations,” *Signal Processing*, vol. 6, no. 4, pp. 267–278, Aug. 1984.

[116] A. Viholainen, J. Alhava, and M. Renfors, “Implementation of parallel cosine and sine modulated filter banks for equalized transmultiplexer systems,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, Salt Lake City, UT, U.S.A., May 7–11 2001, pp. 3625–3628.

[117] Z. Wang, “Reconsideration of a fast computational algorithm for the discrete cosine transform,” *IEEE Transactions on Communications*, vol. 31, no. 1, pp. 121–123, Jan. 1983.

[118] ——, “Fast algorithms for the discrete W transform and for the discrete Fourier transform,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 4, pp. 803–816, Aug. 1984.

[119] ——, “Pruning the fast discrete cosine transform,” *IEEE Transactions on Communications*, vol. 39, no. 5, pp. 640–643, May 1991.

[120] Z. Wang and B. Hunt, “The discrete cosine transform – a new version,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 8, Apr. 1983, pp. 1256–1259.

[121] B. W. Y. Wei and T. H. Meng, “A parallel decoder of programmable Huffman codes,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 2, pp. 175–178, Apr. 1995.

[122] E. W. Weisstein. (2004) Matrix direct sum. From MathWorld – A Wolfram Web Resource. [Online]. Available: http://mathworld.wolfram.com/MatrixDirectSum.html

[123] H. R. Wu and Z. Man, “Comments on “Fast algorithms and implementation of 2-D discrete cosine transform”,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 2, pp. 128–129, Apr. 1998.

[124] H. R. Wu and F. J. Paoloni, “A two-dimensional fast cosine transform algorithm based on Hou’s approach,” *IEEE Transactions on Signal Processing*, vol. 39, no. 2, pp. 544–546, Feb. 1991.

[125] Xilinx, Inc., *Virtex-II Platform FPGA Handbook*, UG002 (v1.0) Dec. 2000.

[126] P. Yip and K. R. Rao, “Fast DIT algorithms for DST’s and DCT’s,” *Circuits, Systems, and Signal Process*, vol. 3, no. 4, pp. 387–408, 1984.

[127] ——, “The decimation-in-frequency algorithms for a family of discrete sine and cosine transforms,” *Circuits, Systems, and Signal Process*, vol. 7, no. 1, pp. 3–19, 1988.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O. Box 527
FIN-33101 Tampere, Finland