STATUS OF THESIS

Title of thesis: Development of Lifting-based VLSI Architectures for Two-Dimensional Discrete Wavelet Transform

I, IBRAHIM SAEED MOHAMED KOKO

hereby allow my thesis to be placed at the Information Resource Center (IRC) of Universiti Teknologi PETRONAS (UTP) with the following conditions:

1. The thesis becomes the property of UTP.
2. The IRC of UTP may make copies of the thesis for academic purposes only.
3. This thesis is classified as
   - ☐ Confidential
   - ☑ Non-confidential

If this thesis is confidential, please state the reason:

The contents of the thesis will remain confidential for ____________ years.

Remarks on disclosure:

____

Signature of Author

Permanent address:
Sudan U. of Science & Technology
Dept. of Electronics Engineering
Sudan, Khartoum

Date: 1/9/2010

Endorsed by

Signature of Supervisor

Name of the Supervisor

____

Date: 1/9/2010

UNIVERSITI TEKNOLOGI PETRONAS

DEVELOPMENT OF LIFTING-BASED VLSI ARCHITECTURES FOR TWO-DIMENSIONAL DISCRETE WAVELET TRANSFORM

By

IBRAHIM SAEED MOHAMED KOKO

The undersigned certify that they have read, and recommend to the Postgraduate Studies Programme for acceptance this thesis for the fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical and Electronic Engineering.

Signature :
Main Supervisor : Dr. Nor Hisham Bin Hamid
Signature :
Co-Supervisor : Dr. Fawnizu Azmadi Hussin
Signature :
Head of Department : Dr. Nor Hisham Bin Hamid
Date : 1/9/2010

DEVELOPMENT OF LIFTING-BASED VLSI ARCHITECTURES FOR TWO-DIMENSIONAL DISCRETE WAVELET TRANSFORM

By

IBRAHIM SAEED MOHAMED KOKO

A thesis
Submitted to the Postgraduate Studies Programme
as a Requirement for the degree of

DOCTOR OF PHILOSOPHY

IN ELECTRICAL AND ELECTRONIC ENGINEERING

UNIVERSITI TEKNOLOGI PETRONAS

BANDAR SERI ISKANDAR,

PERAK

August, 2010

DECLARATION OF THESIS

Title of thesis: Development of Lifting-based VLSI Architectures for Two-Dimensional Discrete Wavelet Transform

I BRAHIM SAEED MOHAMED KOKO

hereby declare that the thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTP or other institutions.

Signature of Author

Witnessed by

Signature of Supervisor

Permanent address:
Sudan U. of Science & Technology
Dept of Electronics Engineering
Sudan, Khartoum

Name of Supervisor

Date : 1/9/2010

Date : 1/9/2010

ACKNOWLEDGMENTS

I would like to express my most gratitude to my supervisor Dr. Nor Hisham Bin Hamid and co-supervisor Dr. Fawnizu Azmadi for their guidance, encouragement, and valuable advises during the course of this research. I would like also to acknowledge my former supervisor Dr. Herman Agustiawan for his guidance, support, and great help during the two years period before he left the university.

I wish to thank the University for giving me the opportunity to undertake this research at Universiti Teknologi PETRONAS. This work would have been impossible without the valuable database resources and facilities provided by the university.

My sincere gratitude also to my family for their unlimited support and patience thought out the period of this work. Finally, I would like to dedicate this work to the memory of my mother for her unconditional love, support, and motivation of us during her life.

ABSTRACT

Two-dimensional discrete wavelet transform (2-D DWT) has evolved as an essential part of a modern compression system. It offers superior compression with good image quality and overcomes disadvantage of the discrete cosine transform, which suffers from blocks artifacts that reduces the quality of the image. The amount of computations involve in 2-D DWT is enormous and cannot be processed by general-purpose processors when real-time processing is required. Therefore, high speed and low power VLSI architecture that computes 2-D DWT effectively is needed. In this research, several VLSI architectures have been developed that meets real-time requirements for 2-D DWT applications. This research initially started off by implementing a software simulation program that decorrelates the original image and reconstructs the original image from the decorrelated image. Then, based on the information gained from implementing the simulation program, a new approach for designing lifting-based VLSI architectures for 2-D forward DWT is introduced. As a result, two high performance VLSI architectures that perform 2-D DWT for 5/3 and 9/7 filters are developed based on overlapped and nonoverlapped scan methods. Then, the intermediate architecture is developed, which aim at reducing the power consumption of the overlapped areas without using the expensive line buffer. In order to best meet real-time applications of 2-D DWT with demanding requirements in terms of speed and throughput parallelism is explored. The single pipelined intermediate and overlapped architectures are extended to 2-, 3-, and 4-parallel architectures to achieve speed factors of 2, 3, and 4, respectively. To further demonstrate the effectiveness of the approach single and parallel VLSI architectures for 2-D inverse discrete wavelet transform (2-D IDWT) are developed. Furthermore, 2-D DWT memory architectures, which have been overlooked in the literature, are also developed. Finally, to show the architectural models developed for 2-D DWT are simple to control, the control algorithms for 4-parallel architecture based on the first scan method is developed. To validate architectures developed in this work five architectures are implemented and simulated on Altera FPGA.

In compliance with the terms of the Copyright Act 1987 and the IP Policy of the university, the copyright of this thesis has been reassigned by the author to the legal entity of the university,

Institute of Technology PETRONAS Sdn bhd.

Due acknowledgement shall always be made of the use of any material contained in, or derived from, this thesis.

© Name of candidate, Year of Thesis submission
Institute of Technology PETRONAS Sdn Bhd
All rights reserved

TABLE OF CONTENTS

Status of Thesis....i

Approval Page....ii

Title Page....iii

Declaration....iv

Acknowledgements....v

Abstract....v i

List of Figures....xiv

List of Tables....xxi

CHAPTER ONE: INTRODUCTION 1

1.1 Background ---- 1

1.2 JPEG2000 Image Compression ---- 4

1.3 Realization of 2-D DWT ---- 5

1.4 Separable and nonseparable transform ---- 6

1.5 Problem statement ----6

1.6 Research objectives and approach ---- 8

1.7 Contributions ----9

1.8 Organization of the thesis ----9

CHAPTER TWO: LITERATURE REVIEW 11

2.1 Introduction ----11

2.2 RAM-based architectures ----14

2.2.1 Direct architecture ---- 15

2.2.2 Row-column and column-row architecture ----16

2.2.3 1-level line-based architecture ------------------------------------17
2.2.4 Multi-level line-based architecture -----------------------------18
2.3 Discussion ---18
2.4 Review of the 1-level line-based architectures -------------------------19
2.5 Conclusion -- 25

CHAPTER THREE : ARCHITECTURE DEVELOPMENT 27

3.1 Introduction --27
3.2 Lifting-based 5/3 and 9/7 algorithms
and architectures development --28
3.3 Data dependency graphs (DDGs) for 5/3 and 9/7 algorithms ---------32
3.4 External Architecture Development and Refinement ------------------32
3.5 Overlapped and Nonoverlapped Scan Methods ------------------------ 38
3.6 Scan Based Architectures --39
3.7 Intermediate Architectures ---44
3.7.1 Generalized Overlapped Scan method ------------------------- 44
3.7.2 Proposed External Intermediate Architecture ----------------- 47
3.7.3 Second Dataflow ---48
3.8 Processors Datapath Architectures Development --------------------------52
3.8.1 5/3 Processor's Datapath Architecture Development --------53
3.8.2 9/7 Processor's Datapath Architecture Development --------54
3.8.3 Row and Column Processors for 5/3 and 9/7 -----------------56
3.9 Evaluation of architectures -- 65
3.10 Combined 5/3 and 9/7 Architecture ---------------------------------------74
3.11 conclusions ---75

CHAPTER FOUR: PARALLEL ARCHITECTURES DEVELOPMENT 76

4.1 Introduction --76

4.2 parallel architectures based on first scan method ------------- ---------------77

4.2.1 2-parallel pipelined external architecture ----------------------77

4.2.2 3-parallel pipelined architecture -------------------------------81

4.2.3 4-parrallel pipelined architecture ------------------------------89

4.2.4 Evaluations of architectures ------------------------------------97

4.3 Parallel form of the intermediate architectures -----------------------------101

4.3.1 2-parallel pipelined intermediate architecture ---------------102

4.3.2 Transition to the last run---------------------------------------108

4.3.3 3-parallel pipelined intermediate architecture ---------------111

4.3.4 Scale factor multipliers reduction ----------------------------119

4.3.5 Evaluation of performance -------------------------------------120

4.4 Conclusions --- 124

CHAPTER FIVE: DWT MEMORY ARCHITECTURES 125

5.1 Introduction --- 125

5.2 The LL-RAM architecture development ----------------------------------- 126

5.2.1 The LL-RAM read operations ---------------------------------129

5.2.2 The LL-RAM write operations --------------------------------130

5.2.3 RAM architecture modifications for

higher scan methods --131

5.2.4 RAM architecture using banks ------------------------------- 134

5.3 Subband memory architecture development -------------------------------137

5.3.1 The bank structure used in forming subband memory -----138

5.3.2 Details of the subband memory architecture ----------------143

5.3.3 Subband memory architecture for higher scan methods ---150

5.4 Control Design for 4-parallel Architecture -------------------------------- 154
5.4.1 Main Control Unit ---157
5.4.2 Processors Control Unit ---------------------------------------164
5.4.3 Read LL-RAM Control Unit ---------------------------------174
5.4.4 Write RAM/Subband Memory Control Unit ----------------178
5.5 Conclusions ---187

CHAPTER SIX: 2-DIMENSIONAL INVERSE DISCRETE WAVELETS TRANSFORM ARCHITECTURE DEVELOPMENT 188

6.1 Introduction --188
6.2 Lifting-based 5/3 and 9/7 synthesis algorithms
and data dependency graphs --189
6.3 Scan methods --191
6.4 Proposed External Architecture --193
6.5 Processors' architecture development -----------------------------------198
6.5.1 Inverse 5/3 processor's architecture development --------198
6.5.2 Inverse 9/7 processor's datapath architecture ------------200
6.5.3 Combined inverse 9/7 and
5/3 processors architecture ----------------------------------200
6.5.4 Modified row and column processors for 5/3 and 9/7
external architecture --201
6.6 Performance Evaluation ---209
6.7 Parallel Architecture Development -------------------------------------- 211
6.7.1 Proposed 2-parallel external architecture ---------------- 211
6.7.2 Modified CPs and RPs for 5/3 and 9/7 2-parallel
external architecture --214
6.8 Proposed 4-parallel external architecture --------------------------------217

6.8.1 Column and row processors for 5/3 and 9/7
4-parallel external architecture ---------- -----------------220
6.8.2 Modified CPs for 4-parallel architecture ----------------220
6.8.3 Modified RPs for 4-parallell architecture ---------------222
6.9 Performance evaluation -- 224
6.10 Conclusions -- 227

CHAPTER SEVEN: EXPERIMENTAL RESULTS 228
7.1 Performance analysis --228
7.2 Performance evaluations and comparisons ----------------------------229
7.3 Experimental results and comparisons ---------------------------------231
7.4 Conclusions ---241

CHAPTER EIGHT: CONCLUSIONS AND RECOMMENDATIONS 242
8.1 Conclusions--242
8.2 Recommendations --246
REFERENCES ---248

APPENDIX A: Software simulation program development -------------------------253
APPENDIX B: Dataflow and control signals tables-----------------------------------270
APPENDIX C: FPGA compilation and synthesis results----------------------------293
APPENDIX D: Publications---298

List of Figures

1.1.1 (a) The original image (b) decorrelated image (c) reconstructed image ---------2

1.1.4 A simplified compression system ---3

1.2.1 JPEG 2000 encoding---4

1.3.1 Lifting-based tree-structured filter bank ---6

2.1.1 One-dimensional tree-structured filter bank and Subband structure -----------11

2.1.2 Tree-structured filter bank for 2-D DWT for D levels decomposition ---------12

2.1.3 3-level of Wavelet decomposition of an image ----------------------------------13

2.2.1 Direct 2-D implementation --15

2.2.2 RCCR 2-D implementation --- 16

2.2.3 1-level line-based implementation --- 17

2.2.4 Multi-level line-based implementation -- 18

3.1.1 Lifting-based tree-structured filter bank with processors -----------------------28

3.2.1 Block diagram representation of Eq (4.2.4) ------------------------------------30

3.2.2 Three processors based 2-D DWT architecture -------------------------------- 31

3.3.1 5/3 algorithm's DDGs for (a) odd and (b) even length signals ----------------33

3.3.2 9/7 algorithm's DDG for odd (a) and even (b) length signals -----------------33

3.4.1 Architecture for 2-D DWT -- 36

3.5.1 Overlapped scan method for 5/3 --39

3.5.2 Non-overlapped scan method for 5/3 --- 40

3.5.3 Overlapped scan method for 9/7 --40

3.6.1 Proposed overlapped scan architecture --- 41

3.6.2 Proposed non-overlapped scan architecture -------------------------------------42

3.7.1 The third overlapped scan method (a) for 5/3 and (b) for 9/7 ---------------- 45

3.7.2 Proposed external intermediate architecture ------------------------------------ 49

3.8.1 5/3 processor's datapath architecture with symmetric extension -------------54

3.8.2 The 9/7 processor's datapath architecture with extension ---------------------55

3.8.3 Modified stage 2 of the 5/3 CP for overlapped and nonoverlapped ----------58

3.8.4(a) Modified first 9/7 CP for overlapped and nonoverlapped architectures------ 58

3.8.4(b) Modified second 9/7 CP for overlapped and nonoverlapped architectures --59

3.8.5 Modified stage 2 of 5/3 for intermediate CP --------------------------------------59

3.8.6 Incorporation of a TLB in stage 2 of the RP ------------------------------------ 60

3.8.7 TLB in a separate pipeline stage ---60

3.8.8(a) Modified first 9/7 RP for overlapped and nonoverlapped architectures ---62

3.8.8(b) Modified fsecond 9/7 RP for overlapped and nonoverlapped architectures ---63

3.8.9 Modified RP datapath for 5/3 and 9/7 intermediate architectures ------------64

3.8.10 Incorporation of a TLB in stage 3 of Figure 3.9.2 to form the 9/7 RP for Intermediate architecture --- 64

3.9.1 Pipelined overlapped parallel scan architecture ----------------------------------68

3.9.2 Pipelined nonoverlapped parallel scan architecture ---------------------------- 68

3.9.3 Pipelined intermediate parallel scan architecture ---------------------------- ---- 69

3.10.1 Combined 9/7 and 5/3 processors datapath architecture -------------------------74

4.2.1 2-parallel pipelined external architecture ---78

4.2.2 Modified RP --81

4.2.3 3-parallel pipelined architecture -- 82

4.2.4 Waveforms of the 2 clocks used in 3-parallel --------------------------------------83

4.2.5 Modified CPs datapath architecture --87

4.2.6 Modified stage 3 of CPs -- 89

4.2.7 4-parallel pipelined architecture -- 90

4.2.8 Waveforms of the 3 clocks used in 4-parallel-------------------------------------- 91

4.2.9 Modified stage 2 of the RPs datapath architecture --------------------------------93

4.2.10 Control signals carried by CST and the block diagram ---------------------- 96
4.2.11 CP1 and CP2 exchange high coefficients -- 97
4.3.1 2-parallel pipelined intermediate architecture -----------------------------------102
4.3.2 Modified stage 2 of 5/3 RPS datapath architecture ---------------------------105
4.3.3 Modified 9/7 RPs datapath for 2-parallel intermediate architecture ---------106
4.3.4 Control circuit that determines the last run ------------------------------------110
4.3.5 (a) Modified 5/3 CP for 2-parallel intermediate architecture ----------------111
4.3.5 (b) Modified 9/7 CP for 2-parallel intermediate architecture ----------------112
4.3.6 3-parallel pipelined intermediate architecture -----------------------------------113
4.3.7 Waveforms of the three clocks --113
4.3.8 (a) Modified 5/3 RPs datapath for 3-parallel intermediate architecture --115
4.3.8 (a, b) Modified 9/7 RPs datapath for 3-parallel intermediate architecture --116
5.1.1 General structure of a compression system------------------------------------ 126
5.2.1 Block diagram of the memory module. --127
5.2.2 RAM architecture using modules --128
5.2.3 Incorporation of register XR ---134
5.2.4 Bank architecture with 8 modules and its block diagram -------------------- 135
5.2.5 RAM architecture using bank --- 136
5.3.1 Subband memory architecture --139
5.3.2 Structure of the first bank and its block diagram ------------------------------ 140
5.3.3 Structure of the second bank and its block diagram --------------------------141
5.3.4 Subband memory architecture built using the block diagram of the second bank --142
5.3.5 Architecture of the subband memory --145
5.3.6 Architecture of subband memory ---146

5.3.7 Incorporation of register XR --151

5.3.8 Flowchart for subband memory write control algorithm ----------------------153

5.4.1 Subband memory interconnections to 4-parallel ----------------------------- 155

5.4.2 DWT Control Unit --- 156

5.4.3 C-unit -- 158

5.4.4 ASM flowchart for A-unit and its block diagram --------------------------------161

5.4.5 ASM chart for B-unit and its block diagram ------------------------------------163

5.4.6 ASM flowchart for TLB control unit and its block diagram.------------------ 166

5.4.7 ASM flowchart for Extension Control Unit and its block diagram.-----------169

5.4.8 ASM flowchart for CPs control unit and its block diagram ------------------ 172

5.4.9 ASM chart for Read RAM Control Unit of the RAM architecture

using modules and its block diagram --175

5.4.10 ASM chart for Read RAM Control Unit of the

RAM architecture using banks and its block diagram ------------------------177

5.4.11 ASM chart for write subband memory control

unit and its block diagram ---180

5.4.12 ASM chart for write RAM control unit of the RAM architecture

using modules, the block diagram, and the proposed clock signal ---------- 184

5.4.13 SM flowchart for write RAM control unit of the RAM

architecture using banks and block diagram -----------------------------------186

6.2.1 5/3 synthesis algorithm's DDGs for (a) odd and (b) even length signals ----190

6.2.2 9/7 synthesis algorithm's DDGs for (a) odd and (b) even length signals ----190

6.3.1 5/3 CP scan method -- 192

6.3.2 9/7 CP scan method -- 192

6.3.3 5/3 RP scan method -- 193

6.3.4 9/7 RP scan method -- 193

6.4.1 roposed external architecture for 5/3 and 9/7 and combined

5/3 and 9/7 for 2-D IDWT and Waveforms for clocks *f* and *f/2* ------------- 195

6.5.1 Inverse 5/3 processor datapath architecture with symmetric extension ------199

6.5.2 Inverse 9/7 processor datapath architecture with symmetric extension ------202

6.5.3 Combined Inverse 9/7 and 5/3 processor datapath architecture --------------203

6.5.4 Modified inverse 5/3 CP datapath architecture with symmetric extension --203

6.5.5 Modified CP for 9/7 and combined 5/3 and 9/7 datapath architecture -------206

6.5.6 Modified inverse 5/3 RP datapath architecture with symmetric extension --205

6.5.7 Modified RP for 9/7 and combined 5/3 and 9/7datapath architecture --------206

6.7.1 Proposed 2-parallel pipelined external architecture for 5/3 and 9/7

and combined 5/3 and 9/7 for 2-D IDWT and waveforms of the clocks----212

6.7.2 Modified inverse 5/3 CP for 2-parallel External architecture --------------- 215

6.7.3 Modified RP for 2-parallel architecture (a) 5/3 and (a) & (b) 9/7 ----------- 216

6.8.1 (a) Proposed 2-D IDWT 4-parallel pipelined external architecture

or 5/3 and 9/7 and combined 5/3 and 9/7. (b) Waveforms of the clocks -----218

6.8.2 Modified CPs for 5/3 CP 1 & 3 and 2 &4 for 4-parallel architecture ------- 221

6.8.3 (a) Modified 5/3 RPs 1 and 3 for 4-parallel External Architecture ---------222

6.8.3 (a, b) Modified 9/7 RPs 1 and 3 for 4-parallel External Architecture --------223

7.3.1 Simulation Report – Simulation Waveforms

for 5/3 "decorrelate_processor"---236

7.3.2 Simulation Report – Simulation Waveforms for 5/3

module "reconst_processor" ---236

7.3.3 Simulation Report – Simulation Waveforms

for first 9/7 "decrrelation2_processor" --237

7.3.4 Simulation Report – Simulation Waveforms

for second 9/7 "decorelation_processor" --------------------------------------238

7.3.5 Simulation Report – Simulation Waveforms

for 5/3 2-parallel module "decorelation_process -----------------------------239

A.3.1 (a) Main program (b) Forward program --260

A.3.1 (c) Horizontal highpass decomposition flowchart ------------------------------261

A.3.1 (d) Horizontal lowpass decomposition flowchart --------------------------------262

A.3.2 (a) Inverse program --263

A.3.2 (b) Vertical lowpass flowchart ---264

A.3.2 (c)Vertical highpass reconstruction flowcharts ---------------------------------.265

A.3.2 (d) Horizontal lowpass flowchart ---266

A.3.2 (e) Horizontal highpass reconstruction flowcharts -----------------------------.267

A.3.3 (a) The original image (b) decorrelated image

(c) reconstructed image ---269

A.3.4 Original image pixels highly correlated ---269

A.3.5 Decorrelated image pixels decorrelated --269

B.11 Circuit --275

C.1.1 Compilation Report – Flow Summary for

5/3 module "decorrelate_processor"---293

C.1.2 Compilation Report – Power Analyzer summary

for 5/3 "decorrelate_processor"---293

C.1.3 Compilation Report – Timing Analyzer Summary

for 5/3 "decorrelate_processor"---293

C.2.1 Compilation Report – Flow Summary for

5/3 module "reconst_processor"---294

C.2.2 Compilation Report – Power Analyzer

summary for 5/3 "reconst_processor"--294

C.2.3 Compilation Report–Timing Analyzer

Summary for 5/3 "reconst_processor"---------------------------------------294

C.3.1 Compilation Report – Flow Summary

for first 9/7 "decrrelation2_processor"-------------------------------------295

C.3.2 Compilation Report – Power Analyzer
summary for 9/7 “decrrelation2_processor”----------------------------------295

C.3.3 Compilation Report – Timing Analyzer
Summary for 9/7 “decrrelation2_processor”----------------------------------295

C.4.1 Compilation Report – Flow Summary
for second 9/7 “decorelation_processor”--296

C.4.2 Compilation Report – Power Analyzer
summary for 9/7 “decorelation_processor”-------------------------------------296

C.4.3 Compilation Report – Timing Analyzer
Summary for 9/7 “decrrelation2_processor”----------------------------------296

C.5.1 Compilation Report – Flow Summary
for 5/3 2-parallel module “two-parallel_DWT”-------------------------------297

C.5.2 Compilation Report – Power Analyzer
summary for 5/3 2-parallel module “two-parallel_DWT”--------------------297

C.5.3 Compilation Report – Timing Analyzer
Summary for 5/3 2-parallel module “two-parallel_DWT”-------------------297

List of Tables

2.1 Summary of the RAM-based 2-D architecture ---------------------------------------19

3.1 Control signal values ---43

3.2 Reduce control signals --44

3.3 Symmetric extension's control signals for 5/3 ------------------------------------ 54

3.4 Symmetric extension's control signals for 9/7 ------------------------------------ 56

4.1 Shows scheduling patterns for CPs and registers involved.----------------------- 85

4.2 Control signal values -- 85

4.3 Shows how and when CPs exchange high coefficients ----------------------------88

4.4 Control signal values for signal sc --88

4.5 Control signal values for *s3, SL0,* and *SL1*--------------------------------------- 104

4.6 Control signal values for signals in stage 2 of both RP1 and RP2 --------------107

4.7 Control signal values for s2, sl0, and sl1 in the last run.-------------------------109

4.8 Control signal values ---118

6.1 Control signal values for *eth, etl, and sr* ---------------------------------- ------- 198

6.2 Extension's control signals --- 200

6.3 Control signal values for 9/7 RP --- 208

6.4 Dataflow of the 5/3 RP1 ---217

7.1 Comparisons of several1-level (9/7) 2-D DWT architectures-------------------- 230

7.2 Experimental results and comparisons --- 234

B.1 Dataflow for Figures 4.6.1 and 4.6.2 ---270

B.2 (a) Dataflow of the second 9/7 pipelined overlapped

architecture for even N--271

B.2 (b) Dataflow of the second 9/7 pipelined overlapped architecture for odd N---272

B.2 (c) Control signal values --272

B.3 Dataflow of the intermediate architecture --273

B.4 Second dataflow for the architecture --274

B.5 (a) Control signal values ------------------------ ------------------------------------275

B.5 (b) Control signal values for sre2 --275

B.6 5/3 Dataflow for overlapped and nonoverlapped parallel scan architecture --275

B.7 5/3 Dataflow for intermediate parallel scan architecture -----------------------276

B.8 Dataflow for 2-parallel architecture ---276

B.9 Dataflow of the architecture ---277

B.10 5/3 4-parallel architecture's dataflow ---278

B.11 4-parallel's TLBs read and write dataflow --------------------------------------279

B.12 Dataflow for 2-parallel intermediate architecture -------------------------------280

B.13 Dataflow of the last run for cases 4 and 3 when N is even ---------------------281

B.14 Dataflow of the last run for cases 4 and 3 when N is odd ----------------------282

B.15 Dataflow of the last run for cases 2 and 1 when N is even ---------------------283

B.16 Dataflow of the last run for cases 2 and 1 when N is odd ----------------------284

B.17 Dataflow of the 3-parallel intermediate architecture ----------------------------285

B.18 Dataflow of the 5/3 architecture -- 287

B.19 (a) dataflow for 9/7 architecture from CP side ---------------------------------288

B.19 (b) dataflow for 9/7 architecture from RP side --------------------------------- 289

B.20 Dataflow for 2-parallel 5/3 architecture --290

B.21 Dataflow for 4-parallel 5/3 architecture --291

CHAPTER 1

INTRODUCTION

1.1 Background

Image compression plays an important role in real-time applications especially in the bandwidth limited applications such as internet, mobile phone, and telemedicine. Images are compressed for fast transmission over a network and efficient storage. Image compression takes advantage of the redundant information contained in the original image. The redundancy exists in the form of statistical dependencies among pixels especially neighboring pixels. However, neighboring or adjacent pixels are highly correlated, which implies that it would be very difficult to immediately compress the original image pixels. Applying a compression algorithm directly to the original image pixels would yield poor compression ratio. Therefore, Transforms such as Fast Fourier Transform (FFT), Discrete Cosine Transform (DCT), and Discrete Wavelet Transform (DWT) are utilized to decorrelate the original image pixels in order to be amenable to compression. Two-dimensional discrete wavelet transform (2-D DWT) compared to DCT is very efficient in decorrelating an image pixels and thus leading to a superior compression performance. DWT naturally as indicated in Figure 1.1.1(b) supports progressive transmission, which is somewhat very difficult to implement in DCT-based compression. 2-DWT has evolved as an effective and powerful tool in many applications especially in image processing and compression [1, 2].

To show the correlation property of original image pixels I have plotted in Figure A.3.4 the pixels of the original image shown in Figure 1.1.1 (a). It shows that the original image pixels are highly correlated. But, when the pixels of the original image are applied to the forward discrete wavelet transform (FDWT) software simulation program that we have developed which is listed in Appendix A, the result was the decorrelated image shown in Figure 1.1.1 (b). The pixels of the decorrelated image

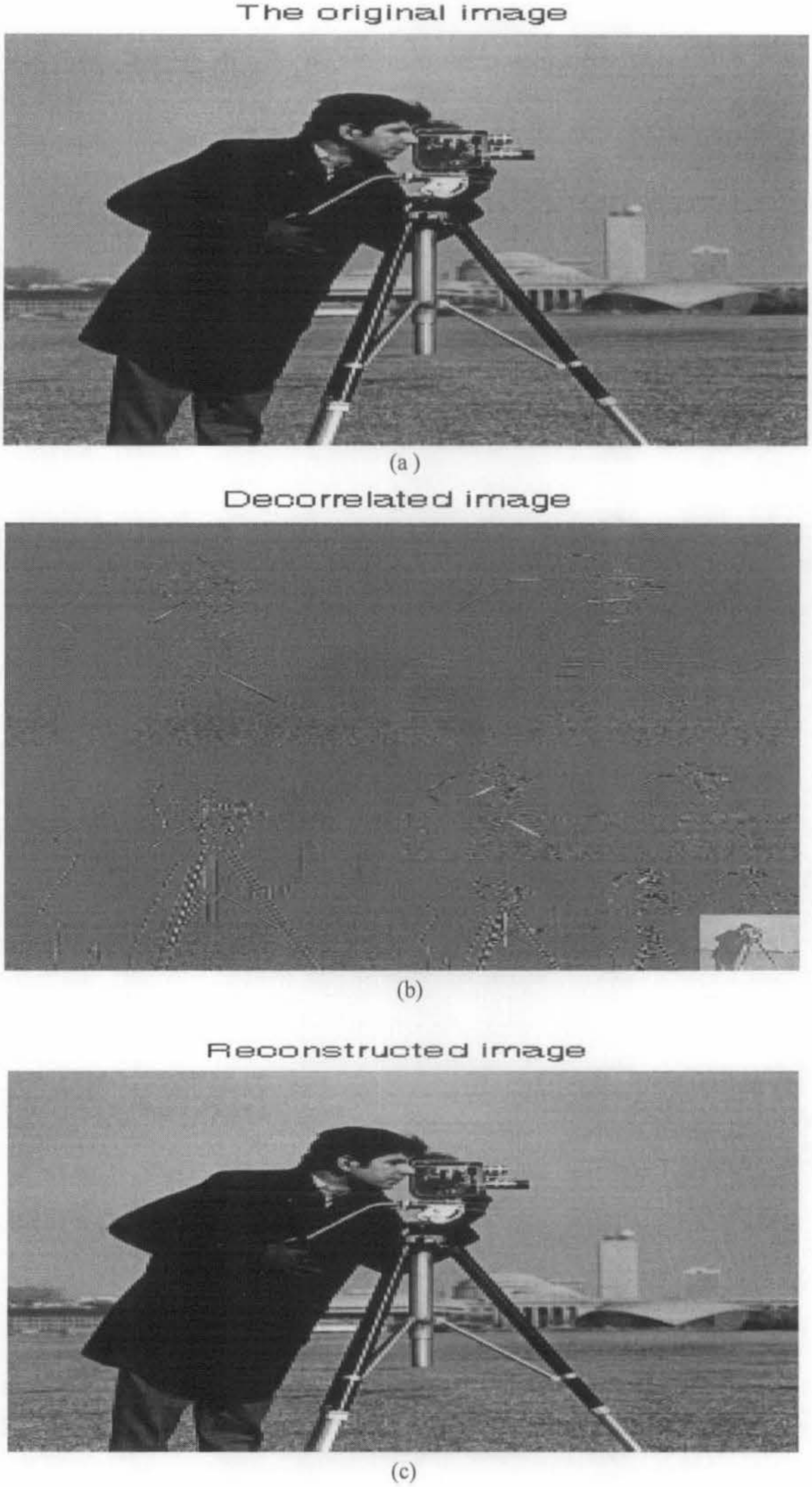


Figure 1.1.1 (a) The original image (b) Decorretated image (c) Reconstructed image

shown in Figure 1.1.1 (b) are then plotted in Figure A.3.5 where it displays a flat image indicating that the amount of correlation among pixels has been greatly reduced.

The 2-D DWT considered in this research is part of a compression system based on wavelet such as JPEG2000, as shown in Figure 1.1.4. The function of the forward discrete wavelet transform (FDWT) in a compression system is to decorrelate the image pixels prior to the compression step [3]. Thus, the DWT is used to effectively decorrelate the image pixels to achieve higher compression rates [4, 5]. Decorrelation step can be thought of introducing distortion to the original image pixels so that they can be amenable to compression.

After transmitting to a remote site, the original image must be reconstructed from the decorrelated image. The task for reconstructing and completely recovering the original image are performed by the inverse discrete wavelet transforms (IDWT).

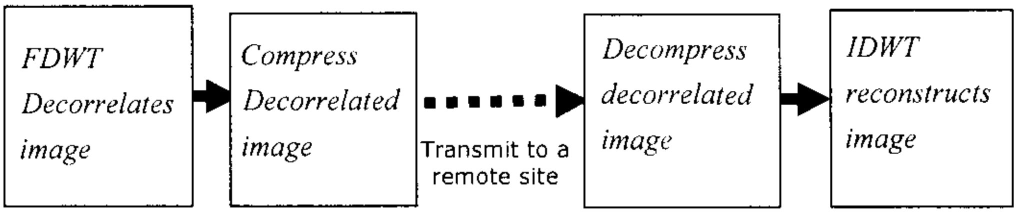

FDWT Decorrelates image
Compress Decorrelated image
Transmit to a remote site
Decompress decorrelated image
IDWT reconstructs image

Figure 1.1.4 A simplified Compression System

The amount of computations involves in both decorrelation and reconstruction steps are enormous, which required very high processing power that can't be achieved by general-purpose processors, especially when real-time processing is required. Therefore, high speed, low power, and low memory VLSI architectures that compute 2-D DWT effectively are needed. The objective of this research is to develop such architectures based on the lifting scheme [4, 5, 6] that meets real-time requirements for 2-D DWT applications. Lifting-based, compared with convolution-based, involves less computation and lower memory and facilitates high speed and efficient implementation of wavelet transform and it is attractive for high throughput and low power applications.

1.2 JPEG2000 Image Compression

JPEG2000 was developed to provide high rates of compression with good image quality and overcome the disadvantages of previous JPEG that uses DCT based image compression [7, 8] which suffers from blocks artifacts that reduce the quality the image.

The JPEG2000 standard uses 2-dimentional, separable, non expansive, symmetric extension wavelet transforms. In this process the whole image is transformed into different resolution levels using the DWT. In case of a large image size, the image is optionally decomposed (divided) into a number of non-overlapping rectangular blocks called tiles and DWT is applied inside each tile independently. The DWT performs either reversible 5/3 filter, which provides lossless coding, or nonreversible 9/7 filter, which provides higher compression ratio with lossy coding. The DWT decomposes an image into subbands, then coefficients of each subband is partitioned into rectangular code block as illustrated in Figure 1.2.1, which are then coded independently using EBCOT (Embedded Block Code with Optimized Truncation). EBCOT is the name given to the entropy encoder in the JPEG2000 and it differs from JPEG's encoder in that the division into independent non-overlapping code-blocks is done after the transform instead of before the transform. EBCOT, which contains tier-1 and tier-2 coding, relies upon independent coding of relatively small blocks of subband samples (e.g., 64 x 64 or 32 x 32 samples). In tier-1 each code-block is independently entropy coded and in tier-2 each encoded bit-stream is optimally truncated such that an overall desired bit rate is achieved. Tier-2 is implemented in software whereas tier-1 is implemented in hardware [8].

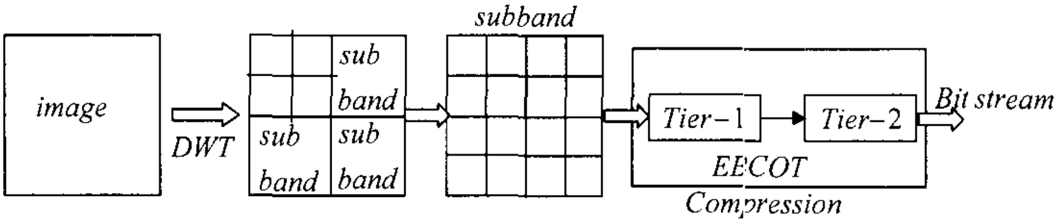


Figure 1.2.1 JPEG 2000 encoding

1.3 Realization of 2-D DWT

The realization of DWT filter bank can be classified into two categories: one is based on the convolution operation [10], [11], [12], and the other is based on the lifting scheme [13], [14], [15]. The tree structure filter bank is the realization of 2-D DWT based on convolution operation. The high-pass and low-pass filters of the filter bank are usually FIR (finite impulse response) filters and FIR involves convolution operation. This direct realization is termed convolution-based DWT. Convolution based DWT is computationally intensive and requires a large number of registers – features that are not desirable in high-speed and low-power VLSI implementation.

On the other hand, lifting-based scheme proposed by Daubechies [4, 5, 6] involves less computation and lower memory. The basic principle of lifting scheme is to factorize the polyphase matrix of the wavelet filters into a sequence of alternating upper and lower triangular matrices and a diagonal matrix called lifting steps [4, 5]. Polyphase divide the filters into even and odd parts as follows [16]:

$$\tilde{h}(z) = \tilde{h}e(z^2) + z^{-1}\tilde{h}o(z^2)\ ,\ \tilde{g}(z) = \tilde{g}e(z^2) + z^{-1}\tilde{g}o(z^2) \qquad (1.1)$$

where $\tilde{h}(z)$ and $\tilde{g}(z)$ are the low-pass and high-pass analysis filters. $\tilde{h}e(z)$ and $\tilde{h}o(z)$ are the even and odd parts of $\tilde{h}(z)$, whereas $\tilde{g}e(z)$ and $\tilde{g}o(z)$ are the even and odd parts of $\tilde{g}(z)$. Eq(1.1) can be represented in a matrix form, called, polyphase matrix, $\tilde{P}(z)$:

$$\tilde{P}(z) = \begin{bmatrix} \tilde{h}e(z) & \tilde{h}o(z) \\ \tilde{g}e(z) & \tilde{g}o(z) \end{bmatrix} \qquad (1.2)$$

If the determinant of $\tilde{P}(z)$ is one, then polyphase matrix can be factorized into lifting steps [4], as follows:

$$\tilde{P}(z) = \prod_{i=1}^{m} \begin{bmatrix} 1 & s_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ t_i(z) & 1 \end{bmatrix} \begin{bmatrix} k & 0 \\ 0 & 1/k \end{bmatrix} \qquad (1.3)$$

It is a well known result in matrix algebra that any matrix with polynomial entries and determinant one can be factored into such elementary matrices. Figure 1.3.1 shows the lifting-based tree-structured filter bank representation of 2-D DWT. The new representation leads to a faster implementation of the wavelet transform and it is attractive for both high throughput and low-power applications. In addition, the computational complexity of the lifting algorithm is half of that of convolution algorithm [4]. Therefore, the lifting-based DWT becomes the preferred scheme for VLSI implementation and it has been selected as the transform coder for image compression in the released JPEG2000 standard.

1.4 Separable and nonseparable transforms

There are two approaches to compute the 2-D DWT: separable and nonseparable [12]. A key practical advantage of separable transforms is that they may be implemented by applying the one dimensional transform first to the rows of the image and then to its columns. The inverse transform is implemented in an analogous manner. A nonseparable approach for the 2-D DWT directly decomposes an image into four subimages without row and column processes one after another [17]. However, the dedicated four 2-D filters require considerably more hardware resources.

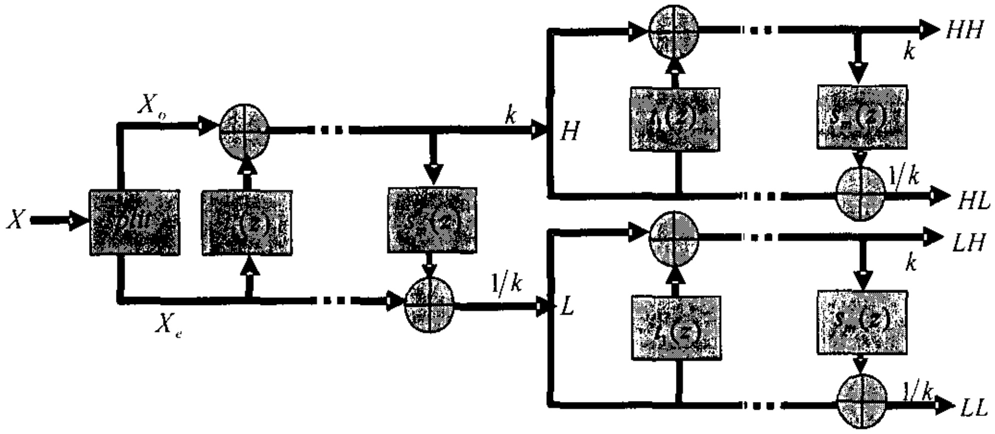


Figure 1.3.1 Lifting-based tree-structured filter bank

1.5 Problem statement

VLSI architecture for 2-D DWT has not yet been completely and accurately developed that meet real-time requirements for 2-D DWT applications. There is need for comprehensive and detailed study to understand the 2-D DWT algorithms in order to develop more accurate architectures. Thorough understanding of DWT algorithms can be gained through developing a software simulation program for both decorrelation and reconstruction processes. Developing a simulation program will give the hardware architecture designer available opportunity to learn in details the behavior of the algorithm and acquire a firm understanding, which in turn will enable him to develop more accurate architecture.

Furthermore, the internal memory of the 2-D DWT processor, which dominates the hardware cost and the complexity of the architecture, is still high, while external memory consumes the most power. Therefore, the research would focus on reducing effectively the internal memory or temporary line buffer (TLB) requirements for 2-D DWT architecture. In addition, novel and accurate architectures for 2-D DWT would be developed that meet high speed and low memory requirements. Furthermore, a specific architecture would be developed that aims at reducing the external memory power consumption, which consumes the most power. The intermediate architecture developed in chapter 3 addresses this issue and 22% reduction in power consumption has been achieved.

DWT decomposes an *NxM* image into subbands. These subbands must be stored by DWT unit in a memory unit in a specific order that preserves the subbands boundaries such that these subbands can be manipulated effectively by both DWT and compression units. This would require developing specific VLSI memory architectures for 2-D DWT. DWT memory architectures have been usually overlooked in the literature. Since, 2-D DWT memory architectures are equally important as DWT processor architectures commonly covered in the literature, in this work, two novel VLSI architectures for LL-RAM and subband memory would be developed. Furthermore, to show the architectures developed in this research are simple to control, one of the architecture would be selected and its control algorithms will be developed. Both pipelining and parallelism will be explored to further improve performance in terms of speed and throughput to best meet real-time applications of 2-D DWT with demanding requirements.

1.6 Research objectives and approach

The objective of the research is to develop VLSI architectures for both decorrelation and reconstruction processors that meet real-time requirements for 2-D DWT applications. In developing VLSI architectures for 2-D DWT processors, our goals are to achieve high speed, low power, low memory, and complete hardware utilization.

In this work, specifically, VLSI architectures for lossless 5/3 and lossy 9/7 algorithms, explicitly defined by the JPEG2000 image compression standard, will be used for the development of the 2-D DWT decorrelation and reconstruction processors. In addition, symmetric extension algorithm recommended by JPEG2000 for boundary treatment will be incorporated into 5/3 and 9/7 data dependency graphs (DDGs) and will be implemented by the architectures developed in this research.

To verify the architectures developed in this research are efficient and accurately perform their intended functions, some selected architectures, which are representative of the other architectures, will be implemented on FPGA and a timing simulation will be performed to validate the logical operations of the designs.

The approach or the strategy adopted in the development of 2-D DWT architectures is based on the observation that the DDGs for 5/3 and 9/7 algorithms are identical when they are looked at from outside, taking into consideration only inputs and outputs requirements, but differ in the internal details. Based on this observation, the first level of the architecture, call it, the external architecture, which is identical for both 5/3 and 9/7, is developed. Then, the internal details of the DDGs is considered for developing separately the processors' datapath architectures for each 5/3 and 9/7 filters that can be incorporated into the external architecture, since DDGs internally define and specify the structure of the processors.

This new approach not only can be effectively used in 5/3 and 9/7 based architectures development, but can be used also in architecture development for any 2-D DWT algorithms and it is certain to yield very efficient architectures in terms of hardware complexity, speed, and power consumption with manageable control complexity.

1.7 Contributions

This research has contributed with several novels VLSI architectural models developed specifically for 2-D DWT as follows. First, a software simulation program is developed that perform both decorrelation and reconstruction of an *MxN* image. Then, two single pipelined architectures based on overlapped and nonoverlapped scan methods are developed for both 5/3 and 9/7 followed by the single pipelined intermediate architecture. The above 3 single pipelined architectures are then extended to 2-, 3-, and 4-parrallel architectures. In addition, modified datapath processor architectures that can be incorporated into single and parallel architectures are also developed.

The research also has addressed one of the critical issues overlooked in the literature, the 2-D DWT memory architectures, and has developed two novel VLSI architectures for LL-RAM and subband memory. Furthermore, to show that the architectures developed in this research are simple to control, the control model and its algorithms for 4-parallel architecture based on the first scan method is developed.

Finally, to show the effectiveness of the approach, the inverse DWT architectures for single and parallel 5/3, 9/7, and combined 5/3 and 9/7 are developed.

Significant parts of this research had been published in international conferences and journals were listed in Appendix D.

1.8 Organization of the thesis

Chapter 2 introduces tree structured filter bank for 1-D and 2-D DWT and classification of 2-D DWT architectures. Then 1-level line-based architectures, which adopt level-by-level approach to achieve multi-level decompositions, are reviewed.

In chapter 3, the data dependency graphs (DDGs) of the algorithms are derived. Based on the DDGs, the overlapped and nonoverlapped single pipelined architectures are developed. The intermediate architecture which is an alternative form of reducing the power consumption of the overlapped areas is also developed.

In chapter 4, in order to best meet real-time applications of DWT with demanding requirements, the parallel architectures based on the first scan method and parallel form of the intermediate architectures are developed

In chapter 5, DWT Memory architectures for LL_RAM and subband memory, which have overlooked in the literature, are developed. To show the architectures developed in this work are easy to control, the control algorithms of the 4-parallel architecture are developed.

In chapter 6, to show the effectiveness of the approach and techniques adopted in the forward architectures, the single and parallel architectures for inverse 5/3 and 9/7 are developed.

In chapter 7, performance evaluations and experimental results for 5 architectures developed in this research are implemented on Altera FPGA and then simulated for validation.

In chapter 8, conclusions are drawn and recommendations for future work are stated.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The basic operation of a discrete wavelet transform is as follow. Applied to a discrete signal containing N samples, a pair of filters low-pass (h_0) and high-pass (h_1) derived from wavelet is applied to the signal to decompose it into a low frequency band (L) and a high frequency band (H). Each band is subsampled (decimated) by a factor of two, so that the two frequency bands each contain N/2 samples. A tree-structured transform is obtained by applying the L band again to a pair of low- and high-pass filters [15]. The one dimensional case is illustrated in Figure 2.1.1. The recursive subdivision is continued for J levels, yielding a total of (J+1) subbands. The low frequency subband L_J contains $N/2^J$ samples, while the remaining subbands contain $N/2^j$ samples for $0 < j \leq J$.

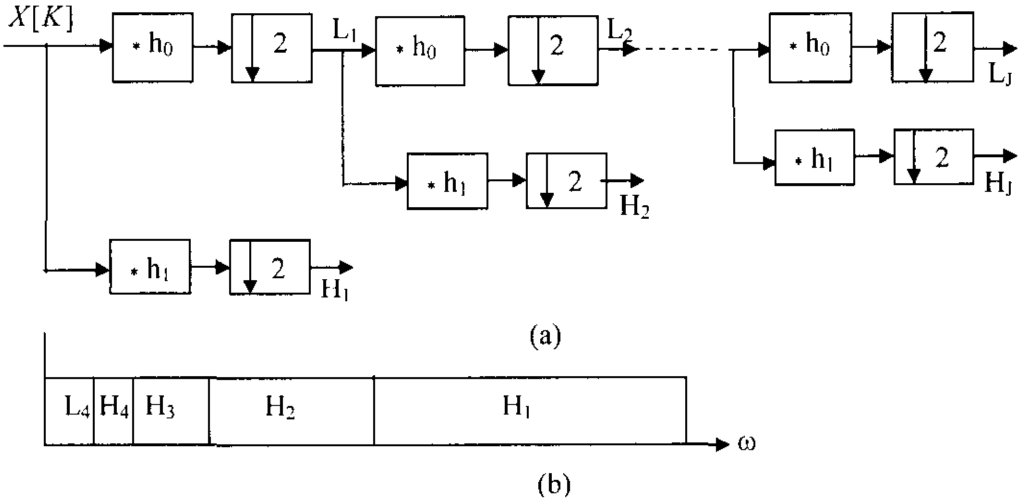


Figure 2.1.1 (a) one-dimensional tree-structured filter bank; (b) Subband structure for J= 4 levels decomposition.

A two dimensional transform is constructed by "separable extension" of one dimensional transform. In this approach each row of 2-J image is filtered with a low-pass (h_0) and high-pass (h_1) filters and the output of each filter is down-sampled (decimated) by a factor of two to produce the intermediate images L and H, as shown in Figure 2.1.2. L is the original image low-pass filtered and down-sampled in the horizontal direction and H is the original image high-pass filtered and down-sampled in the horizontal direction. Next, each column of these new images is filtered with low- and high-pass filters in the vertical direction and down-sampled by a factor of two to produce four sub-images (LL, LH, HL, and HH). These four subband images can be combined to create an output image with same number of samples as the original. The four subband images contain all of the information present in the original image but the sparse nature of the LH, HL, and HH subbands (many samples in these subbands are zeros or close to zeros) makes them amenable to compression.

In an image compression application, the two-dimensional wavelet decomposition described above is applied again to the 'LL' image, forming four new subband images. The resulting low-pass image is iteratively filtered to create a tree of subband images filter bank as shown in Figure 2.1.2. The subband structure is shown in Figure 2.1.3 for

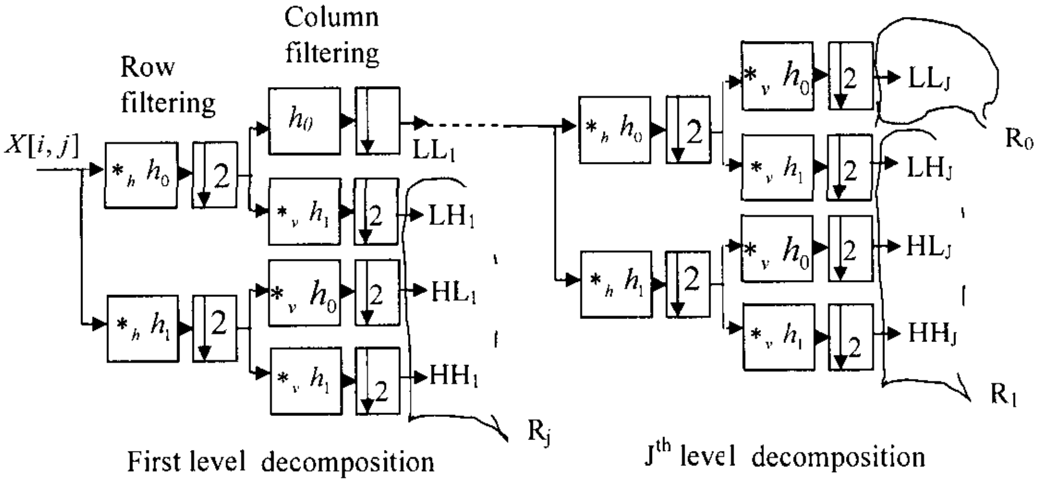


Figure 2.1.2 Tree-structured filter bank for 2-D DWT for J levels decomposition.

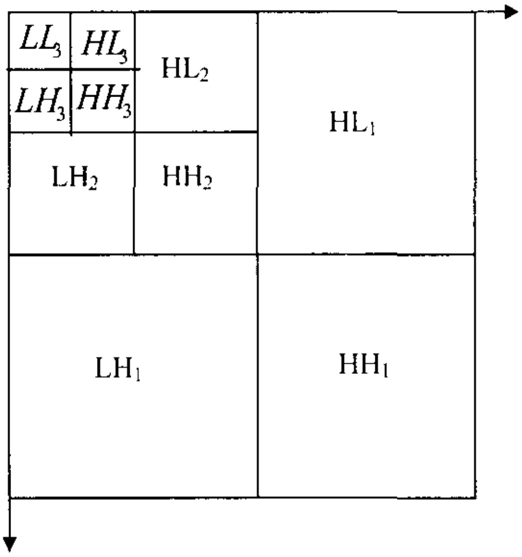


Figure 2.1.3 3-level of Wavelet decomposition of an image

3-level decomposition of an image. In Figure 2.1.2 the notations $*_h$ and $*_v$ denote horizontal and vertical convolution along rows and columns of the image, respectively. And $\downarrow_2$ denote horizontal and vertical decimation by 2 (down sampled by 2). Note that only one of the four subbands, the LL band, is recursively decomposed into further subbands. If the recursive subdivision is continued for J levels, it yields a total of (3J + 1) subbands, with non-uniformly spaced passbands.

The LL subband of nonuniform subband decomposition is a low resolution version of the original image. Therefore, it follows that the lowpass subbands, identified as LL_j in Figure 2.1.2, represent a family of successively lower resolution versions of the original image. The sampling density for LL_j is 2^{-2j} times that of the original image in each direction, where d = 1,2,...,J. However, all these low resolution images are intermediate results; only LL_J is actually one of the subbands of the final tree-structured transform. And each of the images in this multiresolution family may be recovered by partial application of the synthesis system. LL_{J-1}, for example, may be synthesized from subbands LL_J, LH_J, HL_J, and HH_J, while LL_{J-2} may be synthesized from these subbands, together with LH_{J-1}, HL_{J-1}, and HH_{J-1}.

This multiresolution property is particularly interesting for image compression applications. It provides a mechanism whereby a compressed bit-stream may be partially decompressed to obtain successively higher resolution versions of the original image. To be more specific, let R_j be the set containing of subbands LH_{J+1-j}, HL_{J+1-j} and HH_{J+1-j} for $0 < j \leq J$ and let R_0 be the set consisting of only subband LL_J. These groupings are also identified in Figure 2.1.2. We refer to the R_j as resolution levels, since R_0 contains the lowest resolution image and each successive resolution level, R_j, contains the additional information required to reconstruct the next member of the multiresolution family. Suppose now that the elements of each set, R_j, $0 \leq j \leq J$, are compressed independently and their compressed representations are separately identifiable within the compressed image representation. Then, the compressed representation has a property known as "resolution scalability," whereby a compressed representation of any member of the multiresolution family may be obtained simply by discarding those pieces corresponding to the irrelevant resolution levels, R_j. For image compression applications, the interest in dyadic decompositions and hence two channels subband transform is driven primarily by the significance of resolution scalability.

In the literature, 2-dimensional discrete wavelet transform (2-D DWT) architectures are classified into two categories [1, 13]: convolution-based and lifting-based. Convolution-based implements the two-channel filter bank directly. Such an implementation demands intensive computations and a large number of storage — features that are not desirable for either high speed or low power applications [13]. On the other hand, lifting-based involves less computation and lower memory and facilitates high speed and efficient implementation of wavelet transform and it is attractive for both high throughput and low power applications.

2.2 RAM-based architectures

There have been many VLSI architecture proposed for 2-D DWT in literature [13, 27, 29, 34]. Nevertheless, only RAM-based architectures are most practical for real-life designs because of their greater regularity, density of storage, and simple control circuits [1]. However, according to [51], the memory issue dominates the hardware cost and complexity of the architecture and is the most critical part for 2-D DWT architecture. Instead of number of multipliers that decide the performance of one-dimensional (1-D) DWT architectures. Thus, for 2-D DWT architectures, the memory issues, including internal memory size and external frame memory access, are the most critical problems. The internal memory generally dominates the hardware cost, whereas the external frame memory access consumes the most power [51]. In [1], RAM-based architectures for 2-D DWT are categorized as follows.

2.2.1 Direct Architecture

The most straightforward implementation is to perform 1-D DWT in one direction and store the intermediate coefficients in the same frame memory, and then to perform 1-D DWT with these intermediate coefficients in the other direction to complete 1-level 2-D DWT, as illustrated in Figure 2.2.1. For the other decomposition levels, the lowpass-lowpass (LL) subband of the current level is treated as the input signals of the next level and the above steps are then performed recursively.

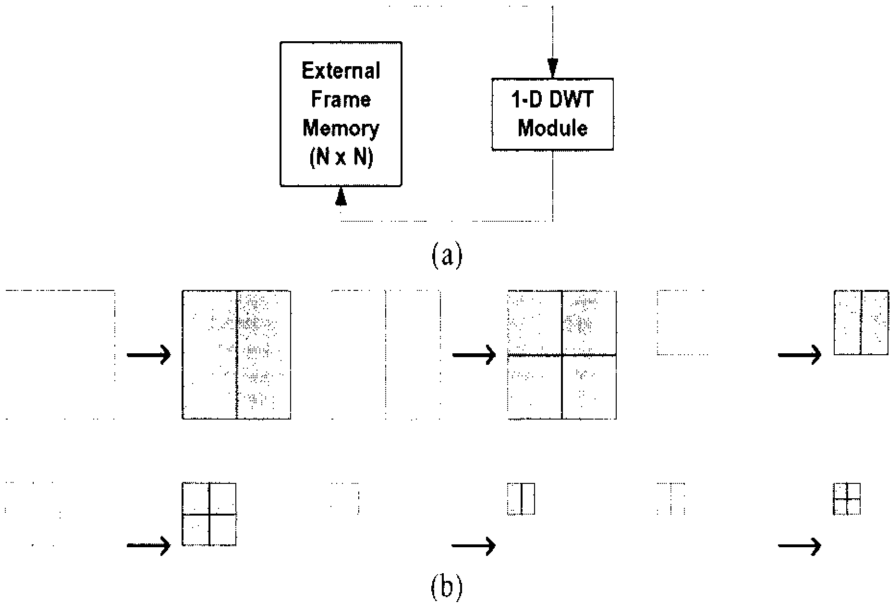


Figure 2.2.1 Direct 2-D implementation. (a) System architecture. (b) Data flow of external memory access (J = 3; white and grey parts represent external frame memory reads and writes, respectively).

2.2.2 Row-column and column-row (RCCR) architecture

The direct architecture processes row coefficients first in every decomposition level all the time. Whereas, RCCR architecture processes row-column for odd-level decompositions and column-row for even-level decompositions [50], then the successive two row-wise or column-wise 1-D DWT decompositions can be performed simultaneously, as illustrated in Figure 2.2.2. The DWT module of the RCCR architecture can be implemented by folding two successive decompositions into 1-D DWT module and store the coefficients in a line buffer of size N/2, and then performs the latter level decomposition with the stored coefficients. The merging of two successive decompositions in the same direction can decrease the external memory access bandwidth by one half for every level, except the first level decomposition.

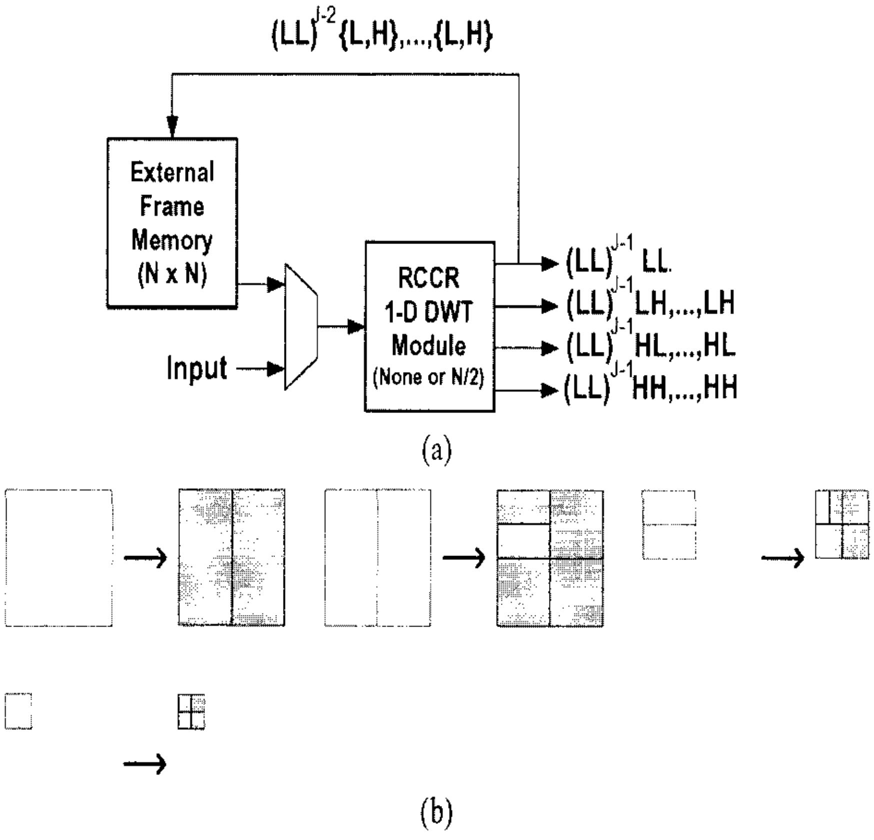
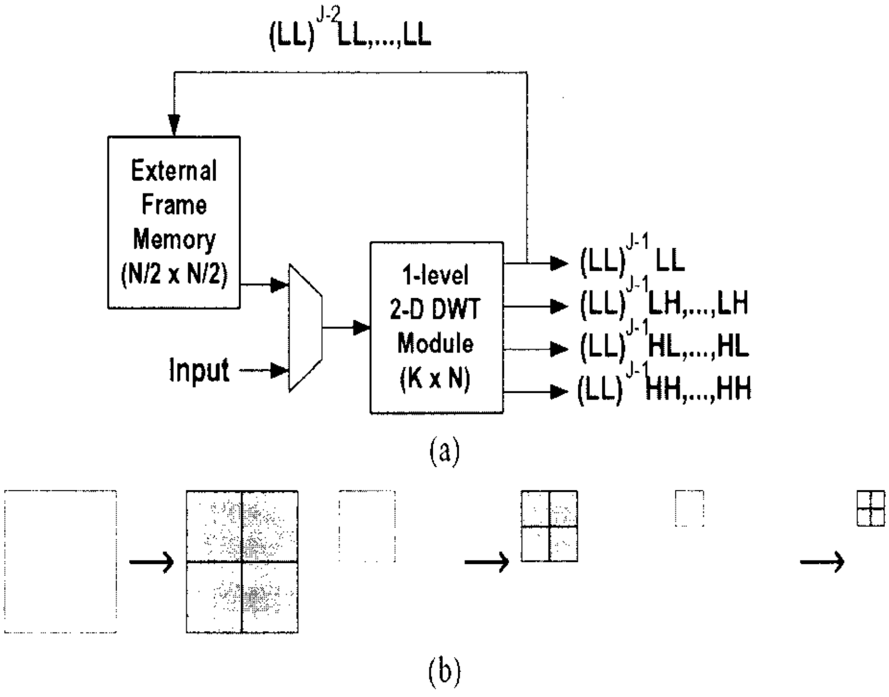


Figure 2.2.2 RCCR 2-D implementation (a) System architecture.
(b) Data flow of external memory access (J = 3).

2.2.3 1-level Line-Based Architecture

Unlike the direction-by-direction approach of direct and RCCR architectures, each level of the DWT decomposition can be performed at a time, and the multi-level decompositions can be achieved by using the level-by-level approach as illustrated in Figure 2.2.3. However, this approach may require some internal memory, whose size is proportional to the image width, to store the intermediate DWT coefficients of one direction and to supply the input signals for the DWT decomposition in the other direction [11].

The external memory bandwidth of the 1-level line-based architecture is exactly one half of that of the direct architecture. This is due to the utilization of internal buffers. Furthermore, unlike the direct architecture that uses the whole frame buffer of size N^2 as the intermediate coefficient buffer, the 1-level line-based architecture only uses one-quarter of the frame buffer.



(a)

(b)

Figure 2.2.3 1-level line-based implementation. (a) System architecture. (b) Data flow of external memory access (J = 3).

2.2.4 Multi-Level Line-Based Architecture

Instead of level-by-level approach, multi-level line-based architecture performs all of the decomposition levels simultaneously, as illustrated in Figure 2.2.4. However, using cascaded J 1-level line-based architectures to implement directly will result in very low hardware utilization. In addition, multi-level 2-D architecture requires more internal buffer and suitable task assignment for 1-D DWT modules; but it reduces the external memory access bandwidth to the minimum $2N^2$.

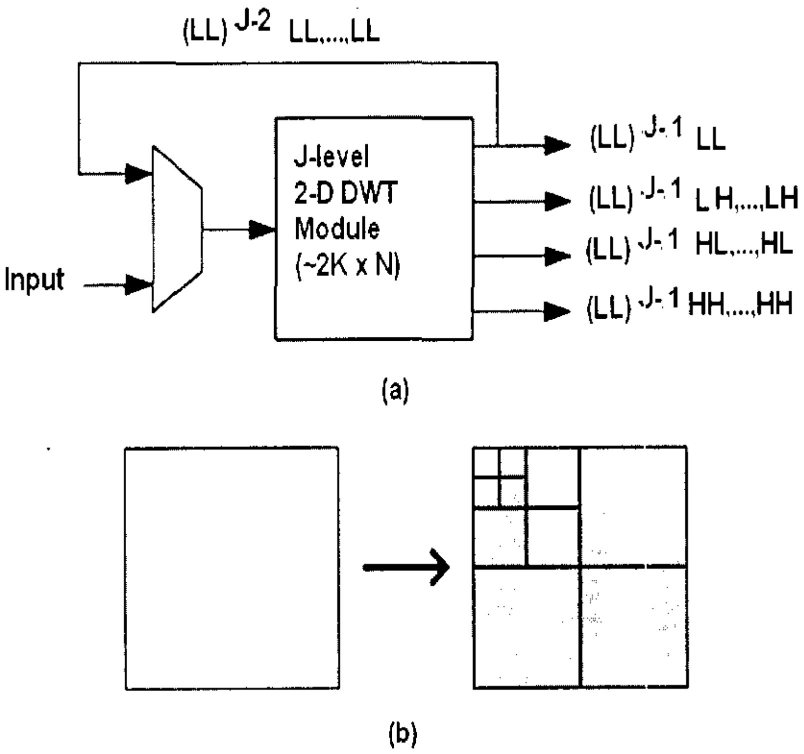


Figure 2.2.4 Multi-level line-based implementation. (a) System architecture. (b) Data flow of external memory access (J = 3).

2.3 Discussion

Based on the Table 2.1, [1] the multi-level line-based architecture requires the most hardware cost, including the internal line buffer, multiple 1-D DWT modules, and complex control circuits. In addition, simultaneously interleaving of the first decomposition level computations with all subsequent levels computations is somewhat a very complex mechanism to control, which makes this approach

Table 2.1 Summary of the RAM-based 2-D architecture [1]

Architecture	External Memory Access (words/image)	Line Buffer (words)	Intermediate Frame Buffer (words)	Control Complexity	System Integration
Direct	$5.33N^2$	-	N^2	Simple	Difficult
RCCR (RPA)	$4.67N^2$	-	N^2	Medium	Difficult
RCCR (N/2)	$4.67N^2$	0.5N	N^2	Simple	Difficult
1-level	$2.67N^2$	kN	$N^2/4$	Medium	Medium
Multi-level	$2N^2$	2kN	-	Complex	Simple

impractical for real-time implementation. However, it requires the least external memory bandwidth without using the external frame buffer to store intermediate data.

The simplest direct architecture has the least hardware cost but requires the most external memory bandwidth. The RCCR architecture can decrease the external memory bandwidth of the direct architecture by using one small line buffer.

The 1-level line-based architecture which adopts level-by-level approach to achieve multi-level decompositions is a simple mechanism to control. In addition, 1-level line-based architecture is the most practical for real-time implementation because of its greater regularity, which suit well for VLSI implementation. Therefore, the research would focus on 1-level line-based architectures and the related work in literature would be reviewed in the next section.

2.4 Review of 1-level line-based architectures

In the following, line-based architectures recently proposed in literature are reviewed. Bing-Fie *et al.* [43] proposed a pipelined architecture for 2-D lifting-based DWT of the 5/3 and the 9/7 filters by merging predict and update stages into one stage (step). The overall architecture includes three main components: the column processor, the transposing buffer, and the row processor. The modified algorithm was derived to shorten the data path but it decreases the throughput of the pipelined architecture. The architecture based on this modified algorithm is more complex and may require a complex control circuits. The transposing buffer is a drawback because it is a very expensive memory component and increases the complexity and the cost of the hardware without any performance advantage. In addition, the architecture requires a total memory of size 3.5N and 5.5N for 5/3 and 9/7, respectively.

Cheng-Yi et al. [40] proposed an architecture which is a combination of a 1-level architecture block and a multilevel architecture block. The 1-level architecture block consists of 4 processors, while the recursive architecture block consists of 2 processors. The 1-level architecture performs the first level of decomposition of the original image and generates four subbands coefficients LL, LH, HL, and HH every clock cycle. The LL coefficients are further pipelined to the recursive architecture block for performing the next levels of decomposition. However, this architecture requires considerable hardware resources with limited utilization, 6 processors and a total of line buffer of size 5.5 and it is definitely slow.

Hongyu *et al.* [59] proposed an architecture called two-dimensional dual scan architecture in which two consecutive rows are scanned simultaneously that allows two pixels to be read per clock cycle from memory and applied to the row processor. In this architecture the FIFO memories had been eliminated and the interleaving mechanism was substituted by adding an intermediate memory of size $N^2/2$ to store LL coefficients for the next levels decompositions. However, the scan method adopted requires a total of line buffer of size 2N and 6N for 5/3 and 9/7 architectures, respectively.

Several lifting-based architectures resembling the architecture in [59] were also proposed in [3], [28], [29], and [35] in which the datapath (the row and the column processors) was pipelined to increase the throughput of the computations. In [30] and [16] very efficient methods were developed that implement the multipliers in DWT data path using arithmetic shift operation, which provide better area-power-operating frequency.

In [25] and [26] line-based VLSI architectures for 9/7 and 5/3 based on lifting scheme were proposed, respectively. The proposed architecture mainly includes a row transform module and a column transform module, working in parallel and

pipeline. The embedded decimation technique based on fold and time multiplexing is exploited to optimize the design of the architecture. The "so-called" embedded decimation technique is defined as that, the samples are input in sequence, then the prediction (dual) lifting and update (primal) lifting operations are performed at the same processing element (PE) by fold and time multiplexing, so that the decimation operation is completed in embedded fashion.

The authors of [25] and [26] claim that by adopting decimation technique they have reduced significantly the required number of multipliers, adders, and registers, as well as the size of the buffer memory and the amount of the RAM access. However, since the two architectures use the raster scan order (RSO) for scanning the external frame memory there would be no significant reduction in the line buffer size. In addition, use of the same processing element (PE) to perform both predict lifting and update lifting operations increase the hardware complexity by requiring introduction of several multiplexers which in turn slow the computations.

In the efficient pipelined architecture presented in [61], a critical path delay of Tm +Ta and a reduction in the number of multipliers are achieved through optimized data flow graph. However, this architecture requires a total line buffer of size 10*N*, which is a very expensive memory component.

The architecture presented in [24] is an attempt to exploit the parallel nature of the 5/3 algorithm through parallel operation of independent units. The design is further optimized by introducing pipeline stages. Input samples are accessed through a window of four samples, allowing two concurrent predict operations and two concurrent update operations. Four coefficients can be calculated in one clock cycle once the pipeline is populated. The major drawback is that the pipeline requires four clock cycles to read new values from external memory and how the architecture is pipelined is not evidence. In addition, predict and update modules including the whole architecture are poorly structured.

In [62], architecture called, deeply parallel architecture is proposed. The architecture requires a buffer memory (BM) of size 5*N*, several FIFO buffers, and a main memory (MM) of size 4*N*, which are very expensive memory components. In addition, writing the results into MM and then switching them out to external memory (EM) is really a drawback, since external memory usually consumes the most power [47].

Chengyi et al. [64] proposed a line-based architecture for 2-D DWT where an embedded decimation technique is exploited to optimize the architecture. The architecture is mainly constituted of an input data buffer unit (IDBU) implemented as (FIFO) RAMs, and a wavelet transform (WT) module. The WT module includes two horizontal filters HF1 and HF2 for row-transform and one vertical filter module VF for column-transform. The image is scanned into HF1 and HF2 in a raster format. Two lines of sample are required to input simultaneously to the transform module, therefore, the two FIFOs are used first to store the required input data before they are sent out to the row-transform module. The architecture requires excessive hardware resources; two FIFOs and two row-processors. In addition, scanning using a raster format is a drawback. The architecture also suffers from long latency of N/2 and 2N for 5/3 and 9/7, respectively. The architecture requires a total memory of size 3.5N and 5.5N for 5/3 and 9/7, respectively.

Chih et at. [66] proposed based on new algorithms architectures for 5/3 and 9/7 which aim at improving the critical issues of the 2-D DWT. The architecture consists of four parts, two sets of the first stage 1-D DWT, two sets of the second 1-D DWT, control unit, and Mac unit. The new algorithm, however, increases the hardware complexity of the architecture and does not decrease the transpose memory requirement. In fact, the architecture requires a transpose memory of size 2N and 4N for 5/3 and 9/7, respectively, in addition to internal memories. The architecture also suffers from long latency, 3/2N +3 cycles.

Wei et at. [68] proposed architecture for 2-D DWT, which reduces the internal memory required for 5/3 and 9/7 to 2N and 4N, respectively. However, the row and the column processors are not pipelined and require considerable hardware resources which lead to longer critical path delay. In addition, scheduling coefficient, generated by the row processor, to the column processor and registers used are not shown in the architecture. The architecture requires a latency of 3/2N +3 clock cycles, which implies the architecture need an additional transpose memory at least of size 1.5N and that increases the total memory required for 5/3 and 9/7 to 3.5N and 5.5N, respectively.

Jie et al. [67] proposed a modified interger-to-interger wavelet transform architecture based on fixed-point manipulation. The architecture consists of horizontal and vertical transform processors, intermediate buffer, control module, and output control module. Image is input line-by-line to the horizontal processor to perform horizontal filtering. Vertical processor employs row-wise coefficients and simultaneously fetches data via intermediate buffers to execute column-wise transform. The latency of the architecture is too long, 5N clock cycles. Intermediate memory buffer of size 5N, in addition, to several memories which are internal to the vertical processor are required in order for the architecture to perform its task. Furthermore, the fixed-point manipulation actually increases the computational complexity of the architecture, which leads to longer critical path delay.

The 5/3 architecture proposed in [69], consists of five key modules: data choose module, the row DWT module, the column DWT module, DWT control unit, and external RAM. The architecture requires a transpose memory of size 2N and internal memory of size 2N, a total of 4N memory which is considered a large memory for 5/3 architecture. The data choose module is a drawback since it constitutes an extra module, in addition, its structure is not drawn and how it operates is not described.

In [70], VLSI architecture for the 2-D 9/7 float discrete wavelet transform (DWT) for the Consultative Committee for Space Data Systems image data compression is proposed. The proposed architecture mainly consists of five parts: row processor, column processor, intermediate buffer, controller, and external memory. The row processor calculates the horizontal DWT of each row of the external memory image data. Then, the resulting decomposed high-pass and low-pass coefficients are stored in the intermediate buffers. The column processor calculates the vertical DWT as soon as five rows have been processed. That means, the architecture would require a latency of 5N clock cycles which is a very long latency. In addition, the row and the column processors require large hardware resources and the internal memory requirement is too large, 22N, which makes this architecture very expensive.

One of the serious limitations of the lifting-based architecture is its potentially long critical path [2]. This problem was addressed in [2] and [21] and these papers proposed architectures which aim at shorting the critical path of the lifting-based 1-D architectures. Huang et al. [2], proposed an efficient VLSI architecture, called flipping structure, in which the problem of serious timing accumulation for lifting-based architectures is addressed by flipping some computing units with the inverse of multiplier coefficients such that the critical path can be greatly reduced. However, this architecture requires a total line buffer of size 11N, which is a very expensive memory component. A modified view of the flipping structure is presented in [21]. Compared with Huang's method, the method proposed in [21] is more efficient in reducing critical path and memory requirement for one processor is 4N. But, usually 2-D DWT architectures consist of 2 processors, which would require more line buffers. Furthermore, reducing the critical path delay to one multiplier is no longer a critical issue, since coefficients and scaling factors of the 9/7 can be implemented in hardware with only 2 adders using arithmetic shift method [23].

In [60], by reordering the lifting-based DWT of the 9/7, the critical path delay of the pipelined architecture has been reduced to one multiplier delay. But the architecture requires a total line buffer of size 5.5N, which is a very expensive memory component. In addition, it requires real multipliers with long delay that can't be implemented by using arithmetic shift method. Moreover, the fold architecture which uses one module to perform both predictor and update steps in fact increases the hardware complexity, e.g., use of several multiplexers, and the control complexity. Use of one module to perform both predictor and update steps implies both steps have to be sequenced, which will definitely slow down the computation process.

In [63], a line-based pipelined architecture for the 5/3 and the 9/7 2-D DWT is proposed. The architecture consists of three key modules: the row DWT module, the data buffer, and the column DWT module. The row module performs row-wise DWT and the output data is stored in the data buffer. When enough rows are processed the column module starts to perform the column-wise transform as soon as possible and stores the intermediate results in the temporal buffer memory. The folding technique is employed to reduce the hardware cost, which achieves a critical path of one multiplier delay. The folding technique even though it reduces the arithmetic resources, it require, besides increasing number of multiplexers used, the used of real multipliers which leads to longer critical path delay and more hardware resources. In addition, the temporal buffers, which hold the intermediate results generated by the column DWT module, are not incorporated into the column module's architecture, thus, the architecture is not complete. Furthermore, the architecture requires a total memory of size 3.5 N and 5.5N for 5/3 and 9/7, respectively.

Chung-Fu et al. [71] proposed a pipeline architecture for the 9/7 2-D DWT. The proposed architecture is composed of column and row processors to perform the separable 2-D DWT. Based on a rescheduling algorithm, which merges the computation of each lifting step, a critical path of one multiplier and two full-adders delay is achieved. The architecture is generally complex and requires more hardware resources such as Wallace tree multipliers. In addition, the architecture requires a total memory of size 5.5N.

JPEG2000 allows (optionally) an image to be divided into a number of smaller non-overlapping rectangular blocks known as “tiles” and 2-D DWT is applied inside each tile independently. Tiling provides a simple mechanism for controlling the amount of working memory used to compute 2-D DWT of a large image [8]. Papers reviewed so far have proposed non-tile-based architectures, i. e.; they process the whole image as one tile. Srikar et al. [27] and Dimitroutakos et al. [36] proposed tile-based architectures for computing 2-D DWT. These architectures are somewhat too complex and memory requirement is high which make them impractical. Nevertheless, tiling is a useful mechanism to use for computing 2-D DWT of a large image independent of its size with the use of the smaller intermediate memory size to store “LL” values for next level decomposition.

2.5 Conclusion

I conclude that the most critical part of 2-D DWT architectures is the memory issue, especially internal memory of the processors, which dominates the hardware cost and complexity of the architecture, while, external memory access consumes the most power. Most of the architectures proposed in the literature managed to reduce internal memory (line buffers) requirements of the processors between 5.5N to 11N, which is still a large memory. In addition, no architectures were developed on purpose that address directly the problem of reducing the power consumption of the 2-D DWT. Other architectures, on the other hand, have focused on reducing the critical path delay of the processor to one multiplier delay. However, this issue becomes less critical after the fact that scales factors and coefficients of the 9/7 filters can be implemented in hardware using only two adders. In addition, these architectures are largely inaccurate and incomplete. Furthermore, two very important issues have been overlooked in the literature, which will be addressed in this research, the DWT memory architectures and control algorithms for 2-D DWT processor architectures.

CHAPTER 3

ARCHITECTURE DEVELOPMENT

3.1 Introduction

This research is started off by developing a software simulation program for both decorrelation and reconstruction processes. The objective of developing the software program is to learn in depth the behavior of the algorithm and in the process to acquire a firm understanding, which would enable us to develop more accurate architectures. The software program is listed in Appendix A.

Then, equipped with information gained from developing the software program, in this chapter, novel VLSI architectures based on lifting scheme that compute 2-D DWT in an image compression system and meet the high speed requirement for real time applications of 2-D DWT will be developed.

As a starting point consider the general lifting-based tree-structured filter bank for the first level decomposition shown in Figure 3.1.1. The figure suggests that 2-D DWT can be implemented by three processors as indicated by dotted lines in the figure. The processors are row-processor, column-processor-H, and column-processor-L. The row-processor (RP) computes DWT row wise i.e., the RP applies one-dimensional DWT algorithm in each row of an image to produce the YH and YL decompositions. The two column processors each compute DWT column wise by applying one-dimensional DWT algorithm in every column of *YH* and *YL*. The column-processor-H takes as an input *YH* and produces subbands *HL* and *HH*, while the column-processor-L takes as an input *YL* and produces the *LH* and *LL* subbands.

Since the tree-structure shown in Figure 3.1.1 is a general representation of 2-D DWT, it would be necessary now to determine the wavelet algorithm that would be used by the three processors to compute DWT. As a matter of fact, any wavelet algorithm could be chosen and the processors hardware architecture could be designed based on it. At this point it is also clear that each processor should be designed to execute one-dimensional DWT algorithm applied either to all rows or all columns of an image. Therefore, to be specific in the architectures development, the one-dimensional lifting-based 5/3 and 9/7 wavelet transform algorithms are selected to be implemented by the three processors.

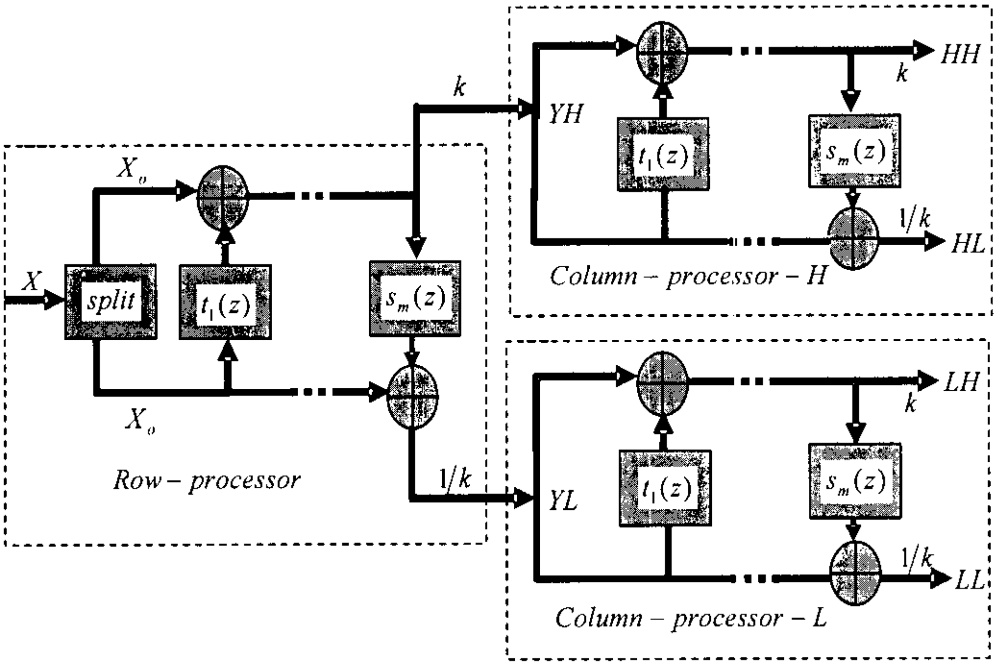


Figure 3.1.1 Lifting-based tree-structured filter bank

3.2 Lifting-based 5/3 and 9/7 algorithms and architectures development

The lossless 5/3 and lossy 9/7 discrete wavelet transforms algorithms are defined by the JPEG2000 image compression standard for 1-D signal X containing N samples, as follow [27, 29]:

<u>5/3 analysis algorithm</u>

$$step1: Y(2j+1) = X(2j+1) - \left\lfloor \frac{X(2j) + X(2j+2)}{2} \right\rfloor$$

$$step2: Y(2j) = X(2j) + \left\lfloor \frac{Y(2j-1) + Y(2j+1) + 2}{4} \right\rfloor$$

9/7 analysis algorithm

$$
\begin{aligned}
&step1: Y''(2j+1) = X(2j+1) + \alpha\big(X(2j) + X(2j+2)\big)\\
&step2: Y''(2j) = X(2j) + \beta\big(Y''(2j-1) + Y''(2j+1)\big)\\
&step3: Y'(2j+1) = Y''(2j+1) + \gamma\big(Y''(2j) + Y''(2j+2)\big)\\
&step4: Y'(2j) = Y''(2j) + \delta\big(Y'(2j-1) + Y'(2j+1)\big)\\
&step5: Y(2j+1) = 1/k\, Y'(2j+1)\\
&step6: Y(2j) = kY'(2j)
\end{aligned}
$$

where $j = 0, 1, 2 \ldots\ldots\ldots, N-1$.

For the RP to compute 2-D FDWT for an $N \times M$ image, the 5/3 algorithm can be written as follows.

```
for $i = 0$ to $N-1$ do
    for $j = 0$ to $M-1$ do
        $Y(i,2j+1) = X(i,2j+1) - \left\lfloor \frac{X(i,2j) + X(i,2j+2)}{2} \right\rfloor$
        $Y(i,2j) = X(i,2j) + \left\lfloor \frac{Y(i,2j-1) + Y(i,2j+1) + 2}{4} \right\rfloor$
    end
end
```

Where $Y(i,2j+1)$ and $Y(i,2j)$ are the high and low decompositions that would result when the image $X(i, j)$ is applied to the algorithm above. This algorithm implies that the high and the low output coefficients are stored in the same memory Y with the high coefficients occupying the odd indexed locations and the low coefficients occupying the even indexed locations. However, I prefer to store high and low coefficients each in a separate memory, so the algorithm above is rewritten as

```
for $i = 0$ to $N-1$ do
    for $j = 0$ to $M-1$ do
        $YH(i,j) = X(i,2j+1) - \left\lfloor \frac{X(i,2j) + X(i,2j+2)}{2} \right\rfloor$
        $YL(i,j) = X(i,2j) + \left\lfloor \frac{YH(i,j-1) + YH(i,j) + 2}{4} \right\rfloor$
    end
end
```

In this representation *X (i, j)* is interpreted as a two-dimensional array in a software implementation and a physical memory in a hardware implementation containing the original image pixels. The algorithm takes as an input *X (i, j)* and decomposes it into high (H) and low (L) decompositions, which are stored in the memories denoted by *YH (i, j)* and *YL (i, j)*, respectively. This algorithm can be represented in a block diagram as shown in Figure 3.2.1. The block diagram consists of a row-processor (RP) and an external memory *X (N, M)* that contains the original image. The processor reads the contents of the memory labeled *X (N, M)* line by line and computes the high and low coefficients of the image and stores the results in the memories labeled *YH* and *YL*, respectively.

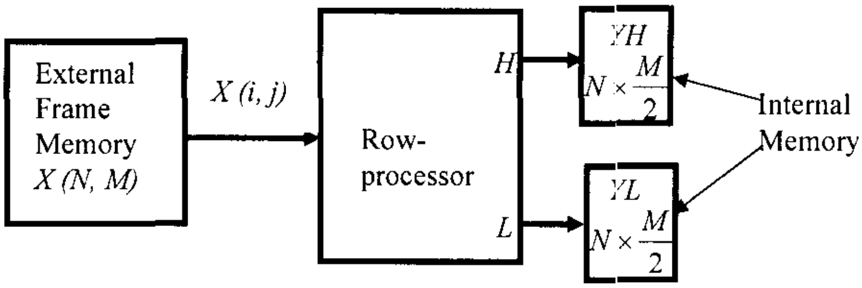


Figure 3.2.1 Block diagram representation of the algorithm

By slightly modifying the indexes of the last algorithm, algorithms for the column-processor-H and the column-processor-L are obtained, respectively. The column-processor-H reads the contents of the memory labeled *YH* as input and yields subbands *HH* and *HL*. Whereas, the column-processor-L reads contents of the memory labeled *YL* and yields subbands *LH* and *LL*.

Column-processor-H

```
for j = 0 to M - 1 do
    for i = 0 to N - 1 do
        $YHH(i, j) = YH(2i+1, j) - \left\lfloor \frac{YH(2i, j) + YH(2i-2, j)}{2} \right\rfloor$
        $YHL(i, j) = YH(2i, j) + \left\lfloor \frac{YHH(i-1, j) + YHH(i, j) + 2}{4} \right\rfloor$
    end
end
```

Column-processor-L

```
for $j = 0$ to $M-1$ do
    for $i = 0$ to $N-1$ do
        $YLH(i,j) = YL(2i+1,j) - \left\lfloor \frac{YL(2i,j)+YL(2i+2,j)}{2} \right\rfloor$
        $YLL(i,j) = YL(2i,j) + \left\lfloor \frac{YLH(i-1,j)+YLH(i,j)+2}{4} \right\rfloor$
    end
end
```

When the two column-processors are combined with the architecture shown in Figure 3.2.1, the architecture shown in Figure 3.2.2 is obtained, which computes the first level DWT decomposition for an *NxM* image. To obtain *J* levels decomposition the LL subband coefficients of each successive level are stored in the memory labeled LL-RAM for further decompositions as shown in Figure 3.2.2. This implies the architecture decomposes 2-D images into the desired number of decomposition levels, level by level.

Similar procedure can be applied to transform the 9/7 algorithm. A careful examination of the last 3 algorithms shows that they are basically identical algorithms, which imply that their processor architectures would also be identical. In addition, the architecture is modular, since it consists of three modules one row-processor and two column-processors and regular because the modules are identical.

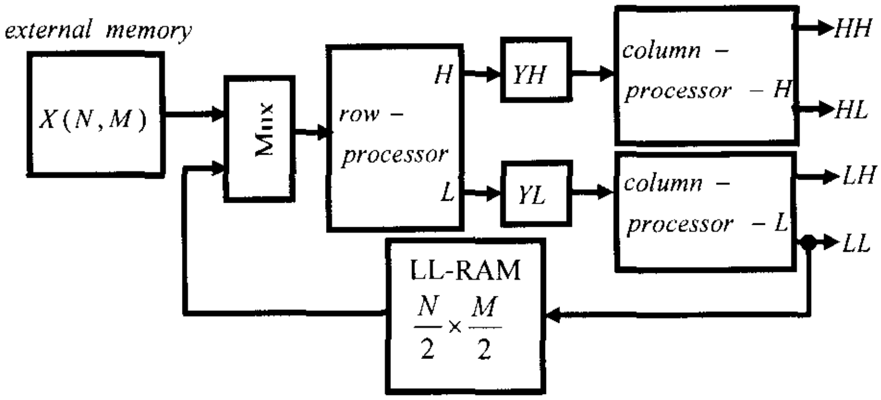


Figure 3.2.2 2-D DWT architecture formed using 3 processors.

3.3 Data dependency graphs (DDGs) for 5/3 and 9/7 algorithms

The data dependency graphs (DDGs) for the 5/3 and the 9/7 algorithms derived from their respective algorithm are shown in Figures 3.3.1 and 3.3.2, respectively. In the DDGs, a node circled with a number represents a computation. All *step1* computations in 5/3 algorithm are performed by the nodes circled with odd numbers (first level) in the DDGs of Figure 3.3.1. On the other hand, *step 2* computations are performed by the nodes circled with even numbers in the second level labeled *Y(2j)* in the DDGs. The symmetric extension algorithm is incorporated in the DDGs to handle the boundary problems. The symmetric extension is represented in the DDGs by dotted lines. The boundary treatment is necessary to keep the number of wavelet coefficients same as that of the original input. The boundary treatment is only applied at the beginning and ending of the process [3]. That means in 2-D images, it will be applied at the beginning and the ending of each row or column. The nodes circled with the same numbers in the DDGs are considered redundant computations, which will be computed once and used thereafter. In addition, note that the symmetric extension algorithm behaves differently for even and odd length signals when it is applied to the data dependency graph. Therefore, two DDGs are provided for each algorithm, one for even and another for odd length signals. The data dependency graph would be a useful tool in architecture development and enhancement.

3.4 External Architecture Development and refinement

In the architecture shown in Figure 3.2.2, the row-processor scans (reads) the external memory, which contains the original image pixels, row-by-row and decomposes the image into high (H) and low (L) coefficients which are stored in the memories labeled *YH* and *YL* respectively. Then, the two column processors simultaneously each reads its respective memory, *YH* and *YL*, and compute subbands *HH, HL, LH,* and *LL* coefficients in parallel.

In order to reduce the size of the internal memories *YH* and *YL* and to allow the two column processors to work in parallel with the row-processor, the DDGs are considered. The DDGs show that, to ease the development of architectures the strategy would be to divide the details of the development into two steps, each having less information to handle. In the first step, the DDGs are looked at from the outside,

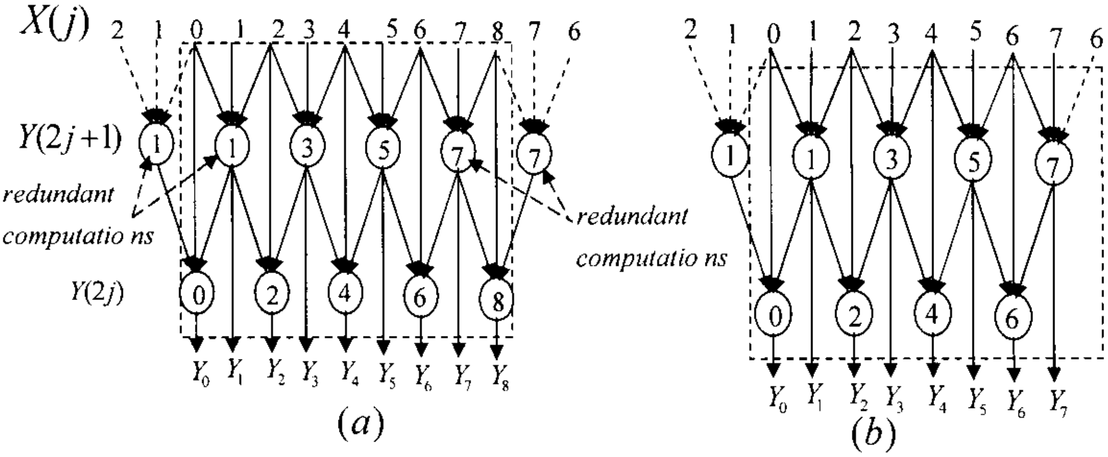

X(j)
Y(2j+1)
redundant computations
Y(2j)
redundant computations
Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7 Y_8
(a)
Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7
(b)

Figure 3.3.1 5/3 algorithm's DDGs for (a) odd and (b) even length signals

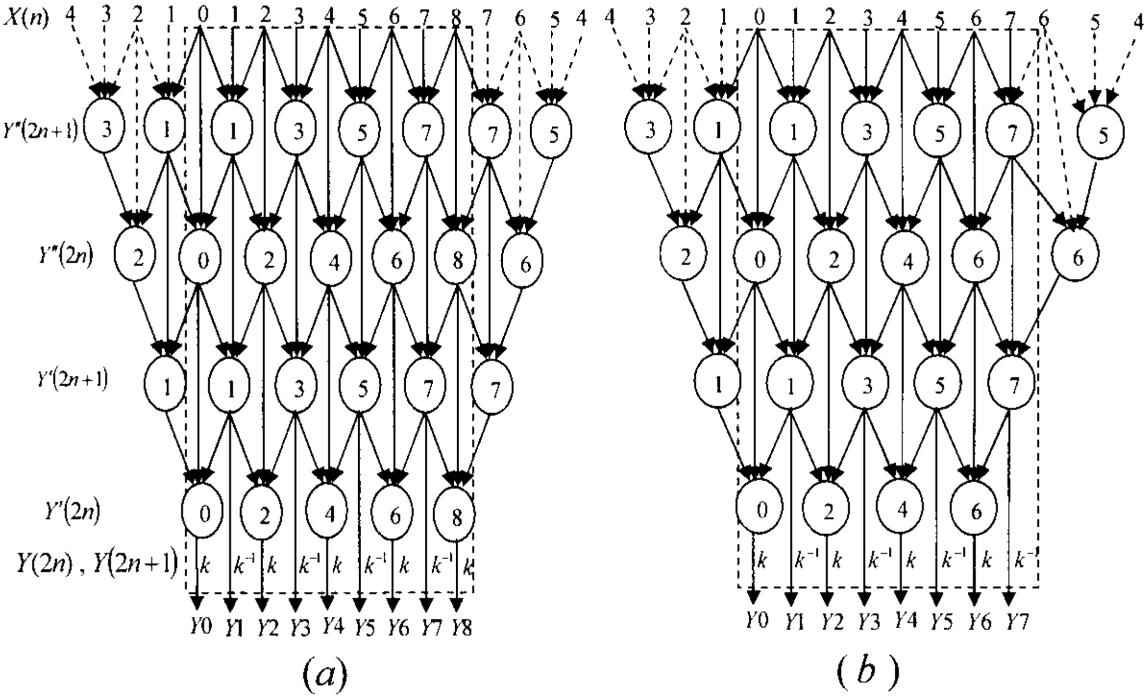


Figure 3.3.2 9/7 algorithm's DDG for odd (a) and even (b) length signals

which is specified by the dotted boxes in the DDGs, in terms of the input and output requirements. We have observed that the DDGs for 5/3 and 9/7 are identical when they are looked at from outside, taking into consideration only the input and output requirements; but differ in the internal details. Based on this observation the first level of the architecture, the external architecture, is developed. In the second step, the internal details of the DDGs are considered for the development of processors' datapath architectures, since DDGs internally define and specify the internal structure of the processors.

The advantage of this new approach along with scan methods developed in the next section can be used not only in the forward 2-D DWT architecture development but in inverse and any DWT algorithm and it is certain to yield very efficient architectures in terms of hardware complexity, speedup, and power consumption with manageable control complexity.

The DDGs of Figures 3.3.1 and 3.3.2 show that to compute one high and one low coefficient at anytime, the processor needs three pixels as an input. Thus, for the two column processors to work in parallel with the row-processor, the row-processor must compute DWT for the first two rows. Then, the two column processors can start computing as soon as the result of the first operation in the third row is available. After that the three processors proceed computing in parallel until the row-processor (RP) performs the last operation in the third row. The two column processors then go into idle states, while the RP works on the fourth row. When the RP reaches the fifth row and as soon as the result of the first operation in the row is available, the two column processors again resume computing in parallel with the RP using the results of the third, fourth, and fifth rows, until the last operation in the fifth row is performed. Then, the two column processors again go into idle states, while RP operates on the sixth row to repeat the process. It is obvious the two column processors would be in idle states or under utilized half of the time. But, the advantage is that the sizes of the two column processors memories labeled *YH* and *YL* can each be reduced to M instead of $N \times M/2$, which is a considerable reduction in a very expensive memory component. In addition, since the two column-processors (CPs) are under utilized half of the time, it is possible to remove one of the CPs and keep only one to compute the four subbands *HH, HL, LH,* and *LL*. When these changes are made to the architecture shown in Figure 3.2.2, the architecture shown in Figure 3.4.1 is obtained and the hardware utilization is 100%. In this architecture, the internal memories *YH* and *YL* each can be considered as consisting of two memory banks of size *M/2*.

To evaluate the performance of the two architectures shown in Figures 3.2.2 and 3.4.1 in term of speedup, consider the following. Assume the RP of the architecture in

Figure 3.2.2 takes T clock cycles to perform one level of decomposition. Then the two column processors, working in parallel; each would need T/2 clock cycles for a total of $T+T/2=3/2T$ cycles to perform one level of decomposition by the three processors. On other hand, the architecture shown in Figure 3.4.1 only requires a total of T cycles to compute one level of decomposition which is a gain in speedup factor of 3/2 as compared with the architecture shown in Figure 3.2.2.

Let us now explain the dataflow of the architecture shown in Figure 3.4.1. Specifically, how data would flow from the outputs of the RP, through the internal memories *YH* and *YL*, to the inputs of the CP. The RP scans the external memory row-by-row, by reading every cycle 3 pixels and placing them into the registers labeled *Rt0*, *Rt1*, and *Rt2* to initiate an operation, and produces as output coefficients of the high (H) and low (L) decompositions, according to the DDGs. The results of the first row computations, which are placed on output lines labeled *H* and L, are stored in the memory banks *B0* of *YH* and *B0* of *YL* respectively. The results of the second row computations are stored in the memory banks *B1* of *YH* and *B1* of *YL*. The CP would start its computations as soon as the results of the first operation in the third row are computed and placed into registers *Rt3* and *Rt4*. The CP performs its computations by reading two coefficients data from the memory banks of *YH* and the third from register *Rt3*. Data in register *Rt3* follows the path that leads to Mux2 , to register Rt6 and finally to the column-processor input labeled *Ic2*. While, data from banks *B0* and *B1* of *YH* follow the paths that lead to Mux0 and Mux1 to be loaded into *Rt7* and *Rt5*, respectively. The CP repeats this process every clock cycle until it consumes the data in the two banks of the *YH* memory including the immediate data coming through *Rt3*. According to the DDGs, the low and high coefficients produced as a result of processing the third row by the RP are needed not only in the current but also in the next calculations involving the 4th and the 5th rows of the *YL* and *YH* decompositions. Therefore, these high and low coefficients are stored in the memory banks B_0 and *B1* of *YH*, respectively, while the CP retrieves data from memory *YH* banks. Of course, that would require reading and writing the same memory location of *YH* in the same clock cycle, which is a problem. One might think as a solution

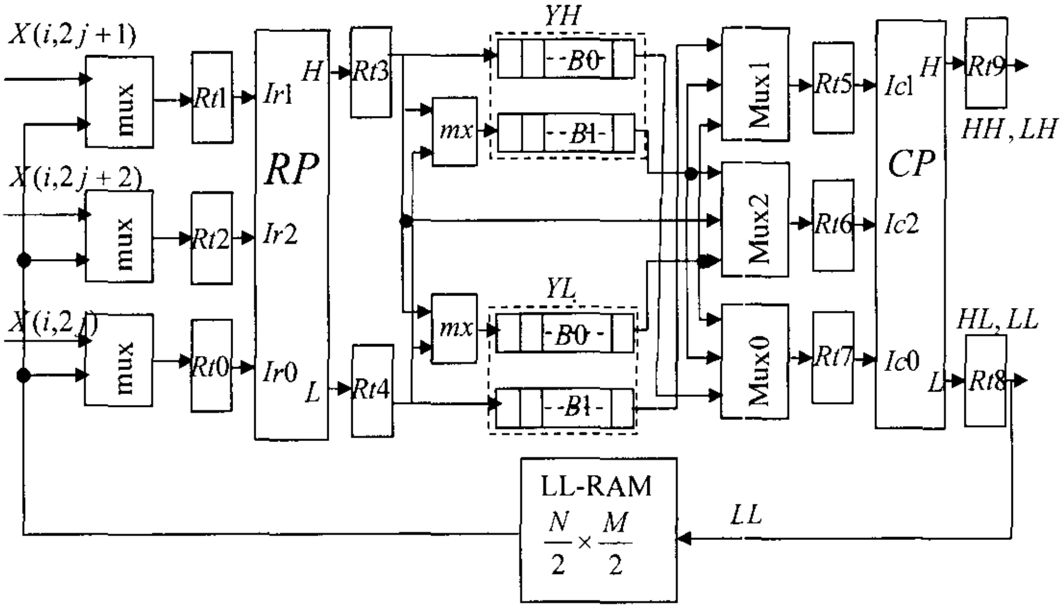


Figure 3.4.1 Architecture for 2-D DWT

implementing the memory banks of *YH* and *YL* as FIFO queues. That sounds logically correct, but practically would require a large number of registers for 2-D images and that would be a very expensive solution which we prefer to avoid. Therefore, we prefer that the memory banks of *YH* and *YL* be implemented as RAM. Then, read and write conflict can be resolved with careful timing by allowing read to be performed in the first half cycle and writing in the second half.

As soon as the CP is done with the data stored in memory *YH* it turns to memory *YL* and starts its second batch by operating on the data stored there. Each clock cycle, two data one from bank *B1* which takes the path that leads to mux1 and the other from bank *B0* that takes the path leading to Mux0. The third data is read at the same time from bank *B*1 of *YH* to complete the three inputs requirement for an operation. While the CP is retrieving and operating on the data stored in the memory banks of *YL* and *B*1 of *YH*, the high and low coefficients, generated by the RP as a result of applying DWT to the pixels of the fourth row in the external memory, are stored in banks *B0* and *B1* of *YL*, respectively. The third batch of computations take place by reading the high coefficients stored in bank *B0* of *YH* and in bank *B0* of *YL*, while the high coefficients, generated by the RP using data of the fifth row, are passed from register *Rt3* through the path leading to mux2 to CP as a third input. At the same time, the high and low coefficients computed using the fifth row's data are stored in bank *B0* of

YH and in bank *B0* of *YL*, respectively, since they are needed in the computations of the next two batches. The fourth batch is a low coefficients processing begins by reading the data stored in banks *B0* and *B1* of *YL* and *B1* of *YH*, which follow the path leading to Mux0, to *Rt7* register, and finally enters the CP through the input labeled *Ic0*. Meanwhile, the high and low coefficients computed by the RP using the data of the sixth row are routed to *B1* of *YH* and *B1* of *YL*, respectively. Data read from bank *B0* of *YL* enter the CP through the input labeled *Ic2*.

A careful examination shows that after the fourth batch is processed, the dataflow or scheduling of batches repeat the same patterns described above for the four batches. That means the next 4 batches would also exhibit the same scheduling patterns of the first four batches and so on. Furthermore, with the pipeline registers *Rt1, Rt2, Rt0, Rt3, Rt4, Rt5, Rt6, Rt7, Rt8*, and *Rt9* are in place not only the RP works in parallel with the CP but the whole architecture are now fully pipelined. The pipeline consists of three stages: the RP stage, the *YH* and *YL* memory stage, and the CP stage. Pipelining improves the performance of the architecture in terms of speedup and throughput as compared with non-pipelined architecture. It is possible to attain maximum speedup and throughput in this architecture because 2-D DWT computations involve a large number of operations. The larger the number of pipeline stages, the higher the speedup.

Even though we have managed to reduce the hardware complexity to a great extend from 3 processors and a total internal memory of size $N \times M$ consisting of *YH* and *YL* in the architecture shown in Figure 3.2.2, to two processors and a total memory of size *2M* for *YL* and *YH* in the architecture shown in Figure 3.4.1 and in the process have gained a speedup factor of 3/2 as compared with the architecture in Figure 3.2.2, the disadvantage of the architecture shown in Figure 3.4.1 is that it requires a very complex control circuitry to govern the dataflow across the memory banks of *YH* and *YL*. In addition, the internal memory requirement is still high. However, it is possible to eliminate the internal memories labeled *YH* and *YL* entirely and use instead a few registers and reduce the control complexity to a great deal by adopting a different scan strategy for scanning the external memory, as would be illustrated in the following section.

3.5 Overlapped and Nonoverlapped Scan Methods

I believe that minimization of the internal memory, and hence the hardware complexity in general for 2-D DWT architectures, depends on the proper scan method adopted for scanning the external frame memory. Therefore, in this section two scan methods are illustrated and will be adopted instead of the row-by-row scan method used so far, to further refine the architecture and obtain novel architectures that best meet real-time applications of 2-D DWT requirements.

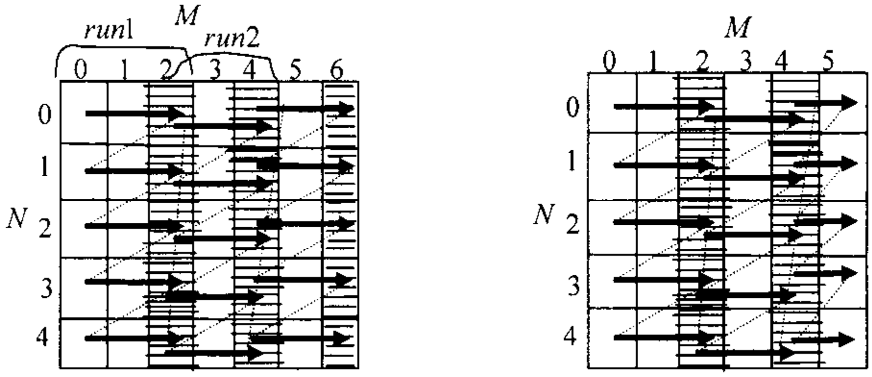
The two scan methods, overlapped and nonoverlapped, are illustrated in Figures 3.5.1 and 3.5.2, respectively. The pixels in the overlapped areas, indicated by the dark lines in Figure 3.5.1, are scanned twice. For an $N \times M$ image, the overlapped scan method requires $NM + N(\lfloor (M-1)/2 \rfloor)$ clock cycles to scan the external memory for the first level decomposition, whereas in the nonoverlapped method, the overlapped areas are eliminated to reduce the external memory access cycles to *NM* clock cycles only and hence reduce the power consumption. The external memory access usually consumes the most power [33, 51].

The scan method shown in Figures 3.5.1 and 3.5.2 are appropriate for both 5/3 and 9/7 algorithms. But, when this scan method is used in 9/7, it would not yield any output coefficients in the first run, according to the 9/7 DDGs. Thus, to allow the 9/7 to generate output coefficients starting from the first run, we propose the overlapped scan method shown in Figure 3.5.3. This scan method differs from 5/3 in the first run only, which requires scanning of 5 pixels from each row. These two scan methods are developed mainly with two objectives to achieve, that is, to make the external architecture for both algorithms identical and to reduce the internal memory between RP and CP to a few registers.

The following two observations, regarding the two scan methods would be necessary in order to develop precise architectures for computing 2-D DWT. First, in the case when the row length of an image is odd, pixels of the last column (*M-1*) are considered overlapped and are scanned twice. In the first scan, according to the DDG for odd length signals shown in Figures 3.3.1 and 3.3.2, they are used in the calculation of the last high coefficient in each row, whereas in the second scan, they are used in the calculation of the last low coefficient in each row. On the other hand, when the row length of an image is even, only the last two pixels in each row (columns *M-2* and *M-1*) are scanned and are used by the RP in the calculations of the last low and high coefficients, as required by the DDG for even length signals.

3.6 Scan Based Architectures

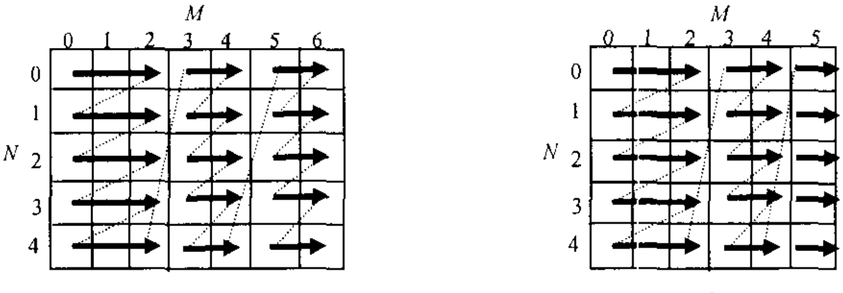
Based on the scan methods and the DDGs for 5/3 and 9/7 shown in Figures 3.3.1 and 3.3.2, when they looked at from outside, the architectures shown in Figures 3.6.1 and 3.6.2 are proposed for overlapped and non-overlapped scan methods, respectively. The architectures operate in a pipeline fashion, consisting of two stages, the RP stage and the CP stage. The two architectures are basically identical. The main difference is that the nonoverlapped architecture contains a line buffer *(LB)* of size *N.* This line buffer is added to hold *N* pixels that lay in each overlapped areas in Figure 3.5.1 in order to reduce the external memory access and hence the power consumption. Pixels in an overlapped area such as column 2 are also required in the next *N* operations. According to the DDGs, each operation performed by either RP or CP would require three inputs. For example, the inputs labeled 0, 1, and 2 in DDG of Figure 3.5.2 initiate the first operation to yield the coefficients labeled *Y0* and *Y1*, whereas inputs 2, 3, and 4 initiate the second operation which yields *Y2* and *Y3* and so on. Fig. 3.6.2 shows the nonoverlapped architecture from the RP side only, since its remaining parts are the same as in Fig. 3.6.1.



(*a*) (*b*)

Figure 3.5.1 Overlapped scan method for 5/3 (a) Odd length signals

(b) Even length signals



(a) (b)

Figure 3.5.2 Non-overlapped scan method for 5/3(a) Odd length signals

(b) Even length signals

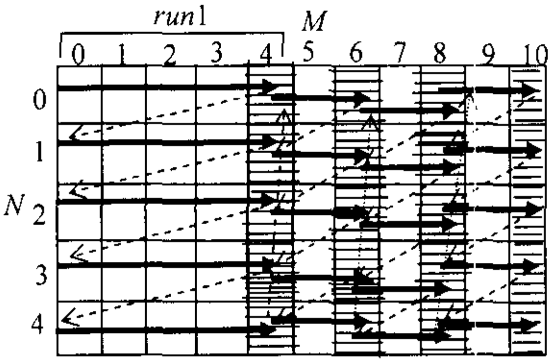


Figure 3.5.3 Overlapped scan method for 9/7

If external memory is scanned with frequency f, both architectures shown in Figures 3.6.1 and 3.6.2 should operate with frequency $f/3$. The dataflow for both architectures is given in Table B.1 (Appendix B). Note that this dataflow is derived based on the 5/3 scan methods shown in Figures 3.5.1 and 3.5.2 and it is identical to the 9/7 architecture's dataflow, based on the same scan methods, in all runs except the first run where 9/7 does not yield any output coefficients. The dataflow of the 9/7 architecture based on the scan method of Figure 3.5.3 is shown in Table B.2 (Appendix B).

Looking at the DDGs shown in Figures 3.3.1 and 3.3.2 from the outside, it can be observed that in the last high and low coefficients calculations, where the row length of an image is even, only the last two pixels in a row, r, at locations X(r, M-2) and X(r, M-1) are read from external memory. In addition, the DDG for even length,

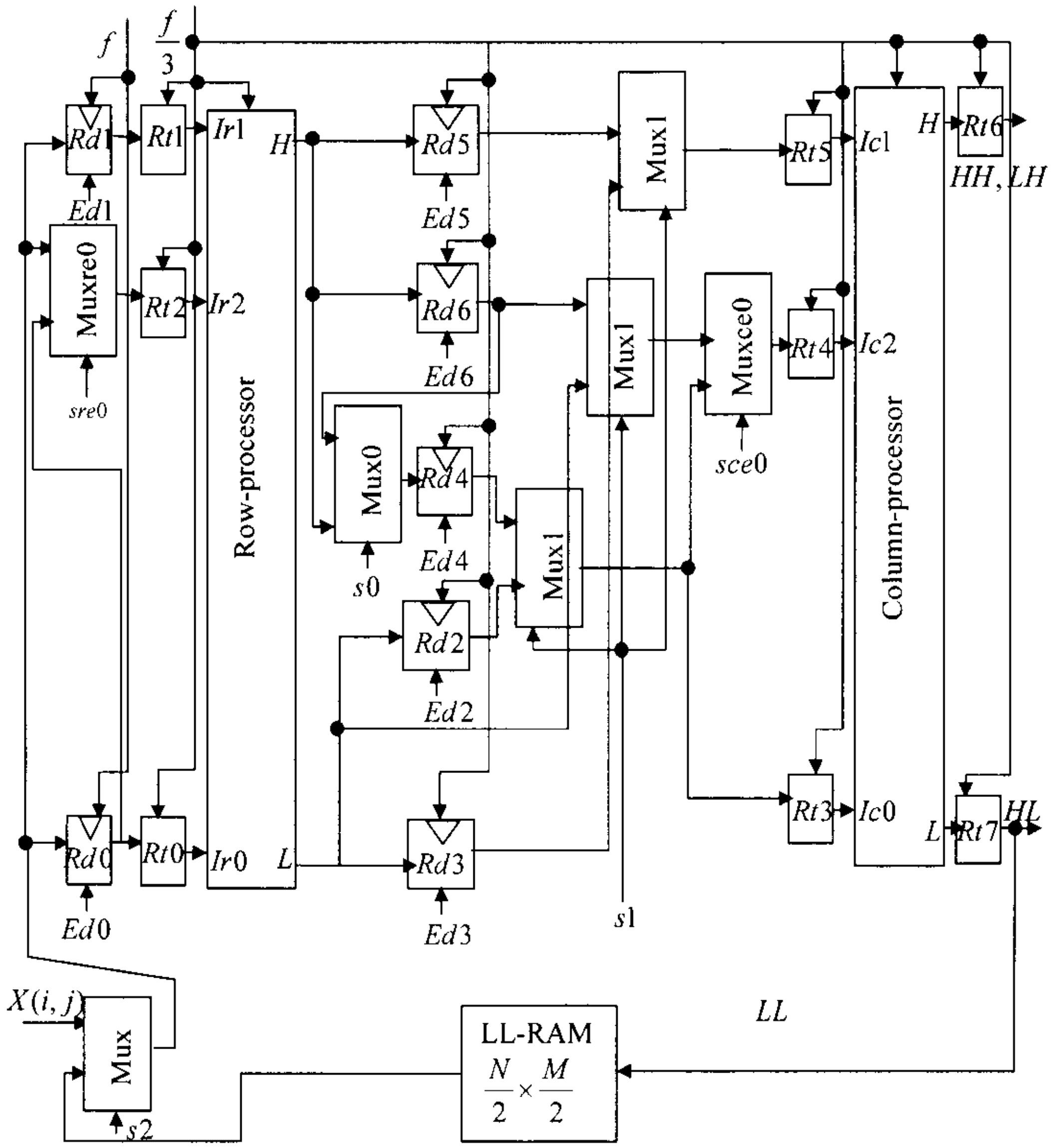


Figure 3.6.1 Proposed overlapped scan architecture

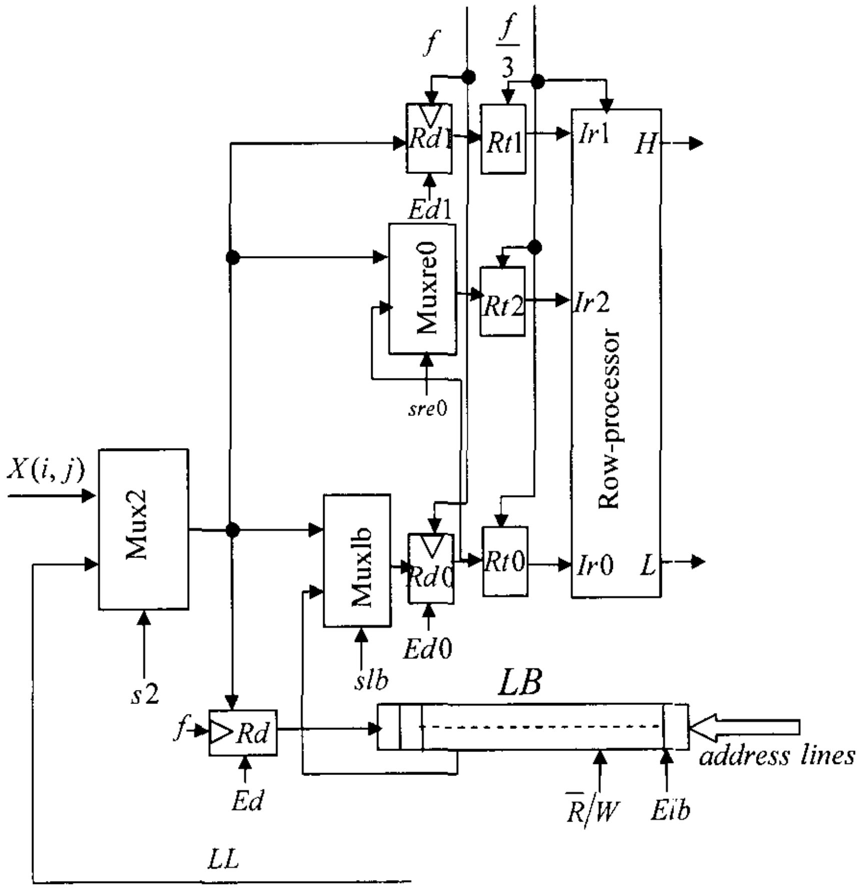


Figure 3.6.2 Proposed non-overlapped scan architecture (RP-side only).

implementing the extension part, requires the pixel located at *X(r, M-2)* to be considered as the first and the third inputs. This must be passed to the RP with the second input pixel from location *X(r, M-1)*, to compute the last high and low coefficients in the row *r*. Thus, the function of the multiplexer labeled Muxre0 is to pass the pixel read from location *X(r, M-2)* after it has been transferred to register *Rd0*, to the row-processor's latch, *Rt2*, as the third input. Register *Rd1* holds the second inputs, pixel from location *X(r, M-1)*. Similarly, the multiplexer labeled Muxce0 performs the same function, when the CP applies DWT to columns. In other words, Muxre0 and Muxce0, which are extension multiplexers, are used only in calculation of the last coefficient in even row or even column images.

On the other hand, when the row length of an image is odd, according to the DDGs for the odd length shown in Figure 3.3.1 and 3.3.2, to calculate the last low coefficient only one pixel the last one at location *X(r, M-1)* should be passed to the row-processor. This pixel is loaded into *Rd0* and then passed to the row-processor where it is used in the computation of the last low coefficient.

In the architecture based on the nonoverlapped scan method, starting from the second run, the dataflow or scheduling of pixels to RP and *LB* should be as follows. Assume the cycle where the last three pixels that are scanned from the last row in the first run are loaded into the RP's latches by the pulse ending, say, cycle *n*. Cycle *n* also transfers the pixel from location *X(N-1,2)* into *Rd*. In cycle *n+1*, the second run begins and the first pixel for the first operation is read from location *X(0,3)* and is loaded into *Rd1* by the pulse ending the cycle. In addition, during cycle *n +1*, contents of register *Rd* are written into the last location of the *LB*. In cycle *n+2*, the first location of the *LB* is loaded into *Rd0* by the pulse ending the cycle and it is the only event that takes place during the cycle. Cycle *n+3* transfers the second pixel from location *X(0,4)* to both *Rd* and *Rt2* and contents of *Rd0* and *Rd1* to *Rt0* and *Rt1* by the pulse ending the cycle, respectively. In cycle *n+4*, *Rd*'s contents are written in the first location of the *LB*. In addition, the first pixel of the second operation which is in location *X(1,3)* is loaded into *Rd1* by the pulse ending the cycle. This pattern of scheduling is repeated until the whole image is scanned.

The control signal values that must be issued by the control unit for the signals labeled *Ed2, Ed3,S0, Ed4, Ed5, Ed6,* and *S1* in the architecture shown in Figure 3.6.1 can be derived, reference to clock *f*, from Table B.1 and starting from clock cycle 6 as shown in Table 3.1. Note that the pattern included in the dotted box repeats after cycle 9. In addition, the number of control signals in Table 3.1 can be reduced further, as shown in Table 3.2, by observing that signals *Ed2=S1=Ed6=S0* and signals *Ed3=Ed5*.

Table 3.1 Control signal values

Cycle	*Ed2*	*Ed3*	*S0*	*Ed4*	*Ed5*	*Ed6*	*S1*
6	1	X	1	1	X	X	X
9	0	1	X	0	1	X	X
12	1	X	X	0	0	1	1
15	0	1	0	1	1	X	0
18	1	X	1	0	0	1	1
21	0	1	0	1	1	X	0

Table 3.2 Reduced control signals

Cycle	Ed2	Ed4	Ed5
6	1	1	X
9	0	0	1
12	1	0	0
15	0	1	1
18	1	0	0
21	0	1	1

3.7 Intermediate Architectures

Two lifting-based VLSI architectures for 2-D DWT for the 5/3 and the 9/7 algorithms were proposed in the previous section based on two scan methods, overlapped and nonoverlaped. In the architecture based on the overlapped scan method, the maximum power consumption occurs due to overlap external frame memory access. On the other hand, in the nonoverlapped architecture, the power consumption was reduced to minimum by eliminating the overlapped areas which requires the addition of a line buffer of size *N*. In this section, we developed a new architecture, called intermediate architecture, for 5/3 and 9/7 algorithms, which aim at reducing the power consumption of the overlapped areas, without using the expensive line buffer, to somewhat between the two extreme architectures proposed in the previous section and hence the name intermediate. The intermediate architectures are based on the generalization of the overlapped scan method which is introduced next.

3.7.1 Generalized Overlapped Scan method

Suppose the overlapped scan method shown in Figure 3.5.1 is termed as the first scan method, since three pixels are scanned from each row. The second method scans 5 pixels from each row. The third scans 7 pixels and the fourth scans 9 pixels and so on. In general, the i^{th} scan method scans $2i+1$ pixels from each row and the number of overlapped areas in the i^{th} scan method can be written as $\lfloor (M-1)/2i \rfloor$. Similarly, consider the overlapped scan method shown in Figure 3.5.3 for 9/7 as the first scan method. Then successive scan methods for 9/7 will differ from that of the 5/3 only in the first run, which requires scanning of $3+2i$ pixels from each row, while scanning in the remaining runs remain the same. These scan methods reduce the excess memory access and hence the power consumption by a factor of $1/i$ as compared with the first scan method. In addition, the internal memory between the row and column processors increases by $5i$ registers, where $i = 1,2,3,\cdots\cdots$ denote the first, the second, and the third scan methods and so on. The excess memory access is due to scanning pixels in the overlapped areas twice. Figures 3.7.1 (a) and (b) show the third overlapped scan method for 5/3 and 9/7, respectively, where the external memory access due to overlapped areas scanning is reduced by a factor of 1/3. Thus, by adopting a higher scan method it is possible to obtain an intermediate architecture, since the external memory access due to scanning of the overlapped areas will be somewhat between the two extreme architectures proposed based overlapped and nonoverlapped scan methods.

To appreciate and have more insight into the excess memory access, which is due to scanning of the overlapped areas twice, consider the following. The architecture based on the first overlapped method, the total external memory access time T_{mo} in clock cycles for J levels of decomposition can be estimated as follows.

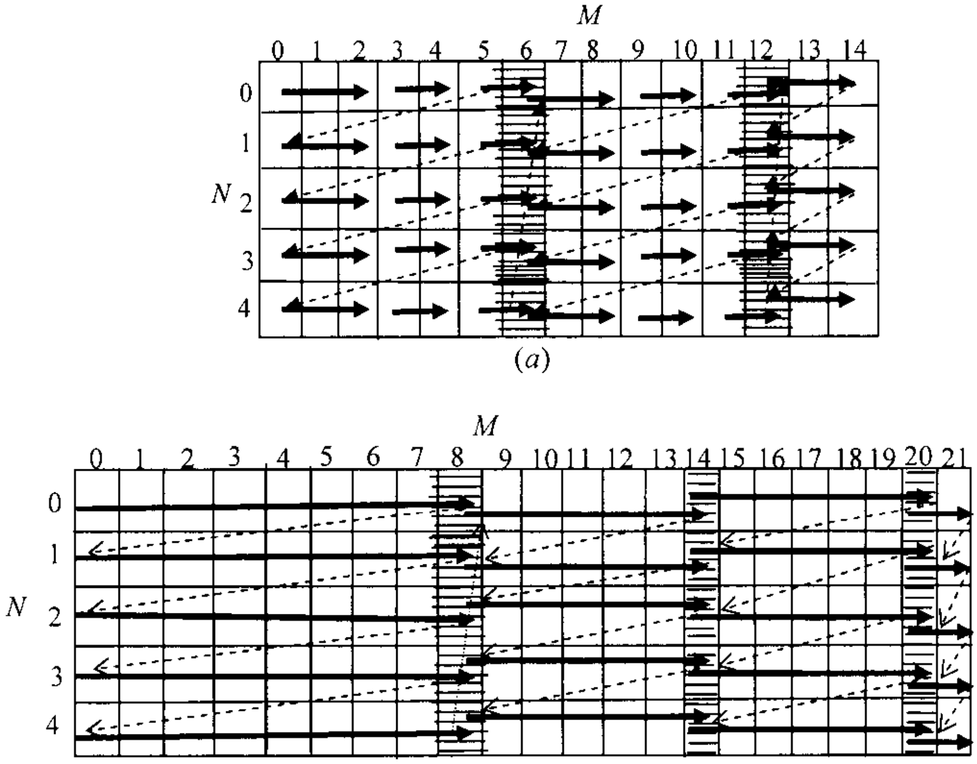


Figure 3.7.1 The third overlapped scan method (a) for 5/3 and (b) for 9/7

$$T_{mo} = NM + N\left(\frac{M-1}{2}\right) + \frac{NM}{4} + \frac{N}{2}\left(\left(\frac{M}{2}-1\right)\Big/2\right) + \frac{NM}{16} + \frac{N}{4}\left(\left(\frac{M}{4}-1\right)\Big/2\right) + \cdots\cdots + \frac{NM}{4^{J-1}} + \frac{N}{2^{J-1}}\left(\left(\frac{M}{2^{J-1}}-1\right)\Big/2\right) \tag{3.1}$$

$$T_{mo} = NM\left(1+\frac{1}{4}+\frac{1}{16}+\cdots\cdot+\left(\frac{1}{4}\right)^{J-1}\right) + \frac{NM}{2} - \frac{N}{2} + \frac{NM}{8} - \frac{N}{4} + \frac{NM}{32} + \cdots\cdot + \frac{NM}{2^{2J-1}} - \frac{N}{2^{J}} \tag{3.2}$$

$$= NM\left(1+\frac{1}{4}+\frac{1}{16}+\cdots\cdot+\left(\frac{1}{4}\right)^{J-1}\right) + \frac{1}{2}NM\left(1+\frac{1}{4}+\frac{1}{16}+\cdots\cdot+\left(\frac{1}{4}\right)^{J-1}\right) - N\left(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}\cdots\cdot+\left(\frac{1}{2}\right)^{J}\right) \tag{3.3}$$

$$= \frac{3}{2}NM\left(1+\frac{1}{4}+\frac{1}{16}+\cdots\cdots+\left(\frac{1}{4}\right)^{J-1}\right) - N\left(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots\cdots+\left(\frac{1}{2}\right)^{J}\right) \tag{3.4}$$

$$T_{mo} = \frac{3}{2}NM\sum_{J=1}^{n}\left(\frac{1}{4}\right)^{J-1} - N\sum_{J=1}^{n}\left(\frac{1}{2}\right)^{J} \tag{3.5}$$

Then using geometric series summation formula $\sum_{k=n1}^{n2} a^{k} = \frac{a^{n1} - a^{n2+1}}{1-a}$, obtain

$$T_{mo} = \frac{3}{2}NM\left(\frac{4-\left(\frac{1}{4}\right)^{J-1}}{3}\right) - N\left(1-\left(\frac{1}{2}\right)^{J}\right) = N\left[\frac{1}{2}M\left(4-\left(\frac{1}{4}\right)^{J-1}\right) - \left(1-\left(\frac{1}{2}\right)^{J}\right)\right] \tag{3.6}$$

$$T_{mo} \cong \frac{1}{2}NM\left(4-\left(\frac{1}{4}\right)^{J-1}\right) \tag{3.7}$$

Since the term $\left(\frac{1}{4}\right)^{J-1}$ will be very small, the above equation can be reduced to

$$T_{mo} \cong 2NM \text{ Clock cycles} \tag{3.8}$$

This equation can be used also to estimate the computation time of 2-D DWT architectures.

On the other hand, for the architecture based on nonoverlapped scan method shown in Figure 3.5.2, the total external memory access time, T_{mn}, in clock cycles for J levels of decomposition can be estimated as

$$T_{mn} = NM\left(1+\frac{1}{4}+\frac{1}{16}+\cdots\cdots+\left(\frac{1}{4}\right)^{J-1}\right) = NM\sum_{J=1}^{n}\left(\frac{1}{4}\right)^{J-1} \tag{3.9}$$

$$T_{mn} = \frac{1}{3}NM\left(4-\left(\frac{1}{4}\right)^{J-1}\right) \cong \frac{4}{3}NM \tag{3.10}$$

Thus, the excess memory access time, T_{me}, due to overlapped areas scanning for J levels of decomposition is given by

$$T_{me} = T_{mo} - T_{mn} = 2NM - 4/3NM = 2/3NM \tag{3.11}$$

which is significant. In the architecture shown in Figure 3.6.2, T_{me} is eliminated and minimum access time T_{mn} and hence minimum power is obtained by nonoverlapped scan method. But, the method requires the addition of a very expensive memory component, a line buffer, in the architecture. The intermediate architectures are alternative form for reducing the power consumption of the overlapped areas, expressed in Eq(3.11), without a line buffer.

3.7.2 Proposed External Intermediate Architecture

Based on the scan method shown in Figure 3.7.1 and DDGs for 5/3 and 9/7 shown in Figures 3.3.1 and 3.3.2, the architecture shown in Figure 3.7.2 is developed. The architecture is valid for both 5/3 and 9/7 algorithms, since it is developed based on the observation that the DDGs for 5/3 and 9/7 are identical when they are looked at from outside, taking into consideration only inputs and outputs requirements. The architecture operates in a pipelined fashion consisting of two stages, the row-processor (RP) and the column-processor (CP). If external memory is scanned with frequency f, then registers Rd0 and Rd1 should operate with frequency f and the rest of the architecture should operate with frequency $f/3$ as indicated in Figure 3.7.2. The dataflow of the architecture, derived based on 5/3 scan method shown in Figure 3.7.1 (a), is shown in Table B.3 (Appendix B). The dataflow is identical to the 9/7 dataflow in all runs except in the first run where 9/7 scans 9 pixels, whereas 5/3 scans 7 pixels from each row.

The clock period τ and hence frequency f of the proposed overlapped, nonoverlapped, and intermediate architectures can be determined by the following statement. f_m is the external memory frequency of operation, f_p is the processor frequency and I is the number of input pixels that are required for an operation. $I = 3$ for 5/3 and 9/7 algorithms.

Statement 1

$$
\begin{aligned}
&Case\ 1: If\ t_m \geq t_p\ then \\
&\qquad \tau = t_m \\
&Case\ 2: else\ \ if\ \frac{t_p}{I} \geq t_m\ then \\
&\qquad\qquad \tau = \frac{t_p}{I} \\
&\qquad\quad else\ \ \tau = t_m
\end{aligned}
$$

To this point the processor critical path delay $(t_p = 1/f_p)$ is expected to be much larger than that of the external frame memory scan delay, $t_m = 1/f_m$. Therefore, the processor delay t_p would be the determining factor of the frequency f. In other words, *case2* will be always true. The situation would change when the processors are pipelined later.

3.7.3 Second Dataflow

The dataflow given in Table B.3 (Appendix B) is justified by the fact that each operation performed by the RP and the CP requires three input data. In addition, since the processor delay t_p determines the scanning frequency f_1 then

$$f_1 = 1/\tau_1 = 3/t_p = 3f_p \tag{3.12}$$

That is, the scanning frequency f_1 should be at least three times faster than the processor frequency f_p in order to allow the scanning of the three pixels during the

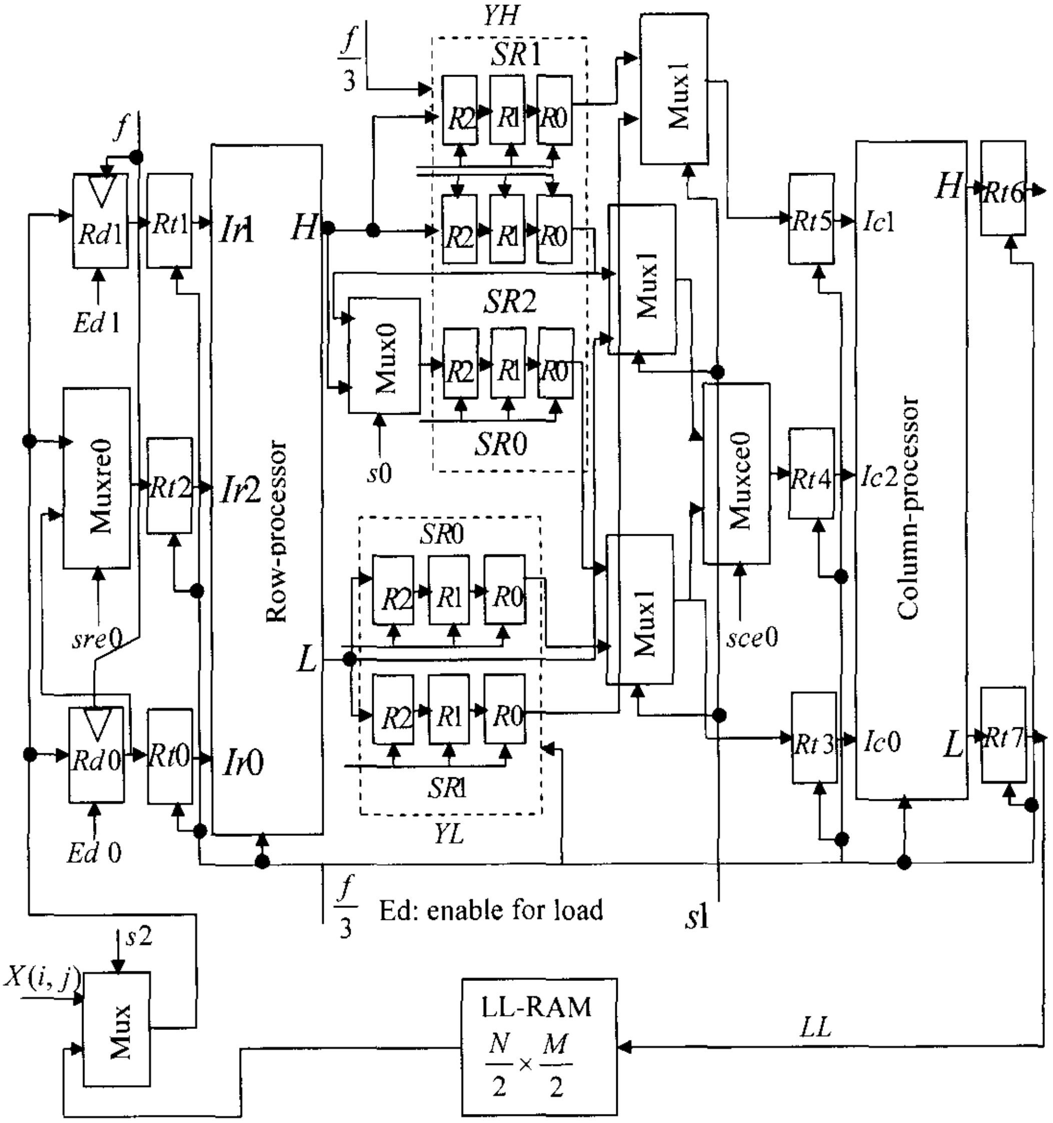


Figure 3.7.2 proposed external intermediate architecture

time specified by t_p. Nevertheless, it is possible to obtain a different dataflow with different frequency by realizing that after the first operation in each row, the second and the third operations in the same row need only 2 pixels to be scanned. This is because the third input pixel of the previous operation which is also the first input in the next operation is already scanned and is available in register *Rd0*. This implies, a new scanning frequency, f_2 can be used, which is given by

$$f_2 = 1/\tau_2 = 1/(t_p/2) = 2/t_p = 2f_p \qquad (3.13)$$

The scanning frequency f_2 is two times faster than the processor's frequency of operation f_p. Thus, with the second scanning frequency, f_2, it is possible to achieve a great reduction in the external memory power consumption but with a drop in speed. The second dataflow is illustrated in Table B.4 (Appendix B).

To compare the performance of the two dataflow in terms of power consumption and speed consider the following. In the first dataflow shown in Table B.3 $\rho_1 = 27$ clock cycles are needed to yield the first pair of output. The remaining *(n – 1)* outputs require *3(n - 1)* cycles. Thus, the total time, *T1*, required to yield *n* paired outputs is given by

$$T1 = [\rho_1 + 3(n-1)]\tau_1 \tag{3.14}$$

Similarly, the second dataflow shown in Table B.4 requires $\rho_2 = 21$ cycles to yield the first pair of output. According to Table B.4, the remaining *(n – 1)* outputs require *7/3(n – 1)* clock cycles. Thus, the total time, *T2*, required to produce *n* paired outputs is given by

$$T2 = [\rho_2 + 7/3(n-1)]\tau_2 \tag{3.15}$$

The speedup factor is then given by

$$S = \frac{T2}{T1} = \frac{[\rho_2 + 7/3(n-1)]\tau_2}{[\rho_1 + 3(n-1)]\tau_1} \tag{3.16}$$

$$S = \frac{7/3(n-1)(t_p/2)}{3(n-1)(t_p/3)} = \frac{7}{6} \tag{3.17}$$

That means the first dataflow is 7/6 times faster than the second. In other words, the total execution time of the second dataflow is increased by 16.7% as compared with the total execution time of the first dataflow.

The power consumption of VLSI architectures can be estimated [17] as

$$P = C_{total} \cdot V_0^2 \cdot f \tag{3.18}$$

where C_{total} denotes the total capacitance of the architecture, V_0 is the supply voltage, f is the clock frequency.

To determine the amount of power reduction in the external memory that can be achieved; when the second dataflow with frequency f_2 is used, consider the following. First, determine the power consumption due to scanning the external memory, when the nonoverlapped scan method is used with frequencies f_1 and f_2. Thus, if P_1 and P_2 denote the power consumed by the external memory for both f_1 and f_2, respectively, then P_1 and P_2 can be written as.

$$P_1 = \beta \cdot C_{total} \cdot V_0^2 \cdot f_1 = \beta \cdot C_{total} V_0^2 / \tau_1 \tag{3.19}$$

$$P_1 = 3 \cdot \beta \cdot C_{total} \cdot V_0^2 / t_p = 3 \cdot \beta \cdot C_{total} \cdot V_0^2 \cdot f_p \tag{3.20}$$

$$P_2 = \beta \cdot C_{total} \cdot V_0^2 \cdot f_2 = \beta \cdot C_{total} \cdot V_0^2 / \tau_2 \tag{3.21}$$

$$P_2 = 2\beta \cdot C_{total} \cdot V_0^2 / t_p = 2\beta \cdot C_{total} \cdot V_0^2 \cdot f_p \tag{3.22}$$

Where $C_{total} \cdot V_0^2 \cdot f_1$ and $C_{total} \cdot V_0^2 \cdot f_2$ are the external memory power consumption due to first overlapped scan method for f_1 and f_2, respectively and $\beta = T_{mn} / T_{mo} = 2/3$. Second, taking into account the fact that the scan method shown in Figure 3.7.1 reduces the power consumption of overlapped areas by a factor of 1/3, then the power consumption due to scanning the overlapped areas using the first and the second dataflow, $Po1$ and $Po2$, respectively are given by

$$Po1 = \beta_0 \cdot C_{total} \cdot V_0^2 \cdot f_1 / 3 \tag{3.23}$$

$$Po1 = 3\beta_0 \cdot C_{total} \cdot V_0^2 / 3t_p = \beta_0 \cdot C_{total} \cdot V_0^2 \cdot f_p \tag{3.24}$$

$$Po2 = \beta_0 \cdot C_{total} \cdot V_0^2 \cdot f_2 / 3 = 2\beta_0 \cdot C_{total} \cdot V_0^2 / 3t_p \tag{3.25}$$

$$Po2 = 2/3\beta_0 \cdot C_{total} \cdot V_0^2 \cdot f_p \tag{3.26}$$

Where $\beta_0 = T_{me} / T_{mo} = 1/3$. Thus, the total power consumption due to external memory access for the first and the second dataflow, $P1_{total}$ and $P2_{total}$ are

$$P1_{total} = P_1 + Po1 = C_{total} \cdot V_0^2 \cdot f_p \cdot (3\beta + \beta_0) \tag{3.27}$$

$$P2_{total} = P_2 + Po2 = 2C_{total} \cdot V_0^2 \cdot f_p (\beta + \beta_0 / 3) \tag{3.28}$$

and

$$\frac{P1_{total}}{P2_{total}} = \frac{C_{total} \cdot V_0^2 \cdot f_p \cdot (3\beta + \beta_0)}{2 \cdot C_{total} \cdot V_0^2 \cdot f_p \cdot (\beta + \beta_0 / 3)} = \frac{3}{2} \tag{3.29}$$

Eq (3.29) implies that power consumption due to external memory scanning in the second dataflow is 2/3 of the first dataflow. In other words, the second dataflow reduces the power consumption by 33.3% over the first dataflow.

On the other hand, the percent of power reduction achieved in the intermediate architecture shown in Figure 3.7.2 for the first and the second dataflow as compared with the architecture based on the overlapped scan method can be obtained as follows.

$$\frac{P1_{total}}{P_{total}} = \frac{C_{total} \cdot V_0^2 \cdot f_p \cdot (3\beta + \beta_0)}{3 \cdot C_{total} \cdot V_0^2 \cdot f_p} = \frac{7}{9} \tag{3.30}$$

Where P_{total} is the total power consumption of scanning the external memory for the architecture based on the overlapped scan method. Eq(3.30) implies that the power consumed due to scanning the external memory in the intermediate architecture based on the first dataflow is reduced by 22.22% as compared with the architecture based on the first scan method. Whereas,

$$\frac{P2_{total}}{P_{total}} = \frac{P2_{total}}{P1_{total}} \cdot \frac{P1_{total}}{P_{total}} = \frac{14}{27} \tag{3.31}$$

implies that the power consumption of the external memory in the intermediate architecture based on the second dataflow is 14/27 of the architecture based on the first scanning method. In other words, the external memory power consumption in the intermediate architecture is decreased by 48% as compared with the architecture based on the first scan method.

3.8 Processors Datapath Architectures Development

To complete the architectures for 2-D DWT, the last phase is to design the row and column processors datapath architectures for 5/3 and 9/7 algorithms separately that can fit into the three architectures shown in Figures 3.6.1, 3.6.2, and 3.7.2. The three architectures are valid architectures for both 5/3 and 9/7 algorithms, since they were developed based on the observation that the DDGs for 5/3 and 9/7 are identical, when they are looked at from outside, taking into consideration only the input and output requirements.

3.8.1 5/3 Processor's Datapath Architecture Development

Based on the 5/3 algorithm and its DDGs shown in Figure 3.3.1, the 5/3 processor datapath architecture is shown in Figure 3.8.1. The multiplexers labeled muxe0, muxe1, and muxe2 implement the symmetric extension. This 3-stage pipelined processor is formed by mapping the two lifting steps of the 5/3 algorithm into two pipeline stages. Stage 3 is added to reduce the critical path delay of stage 2; specifically the path connecting the adders in stage2 to the RP's output L, to muxce0 through mux1, and end at Rt4. Suppose t_a and t_x denote adder and multiplexer delays, respectively. Then, the critical path of stage 2 becomes large, $3t_a + 3t_x$, when the processor datapath is incorporated into the architecture. The addition of stage 3, which is obtained by splitting stage 2, reduces the critical path of stage 2 to $2t_a + t_x$ and that of stage 3 to $t_a + 2t_x$.

Stage 1 computes the high coefficients (*step1*) and sends results to the output labeled H, whereas stages 2 and 3 compute the low coefficients (*step2*) and send results to the output labeled L. According to the DDGs in Figure 3.3.1, each high coefficient calculated in stage 1 enters not only in the calculation of the current low coefficient in stage 2 but also in the next low coefficient calculation in stage 2. Therefore, *Rt1* output of stage 3, which holds the high coefficient, is fed back into Muxe1 and Muxe2 to be considered in the next low coefficient calculation. Stage 2 of the pipeline is a little bit complicated because it implements part of the extension. So in the following, the dataflow of stage 2 is explained. First, according to the DDGs for 5/3, in the calculation of the first low coefficient Y0, the high coefficient value Y1, calculated in stage1, must be allowed to pass through the multiplexers, labeled Muxe1 and Muxe2, to the adder in stage 2. Second, in the calculation of the last coefficient, for example, Y8 in the DDG of odd length signals in Figure 3.3.1(a), the high coefficient (Y7) in *RT1* of stage 3 must be allowed to pass through both Muxe1 and Muxe2 to the adder. During normal computations that occur between the first and last coefficients calculations, the current high coefficient calculated in stage 1 and the previous high coefficient in *Rt1*of stage 3 are allowed to pass through Muxe1 and Muxe2 to the adder, respectively. Note, in even length signals, the last high and low coefficients calculations occur normally. Table 3.3 shows the values of the control signals that have to be issued by the control unit so that the extension multiplexers perform the required functions. Note also, the shift operations that are indicated on the figure by the symbol >> are implemented in hardwire.

3.8.2 *9/7 Processor's Datapath Architecture Development*

A 6-stage pipelined datapath architecture for 9/7 processor is shown in Figure 3.8.2. It is formed using both the 9/7 algorithm and its DDGs shown in Figure 3.3.2. In this architecture the pipeline stages 1, 2, 4, and 5 represent the first 4 steps in the 9/7 algorithm. The implementation of step5 and step6 are incorporated in stage 6 to allow the two steps to operate in parallel. Stage 3, which connects stage 2 with stage 4, is

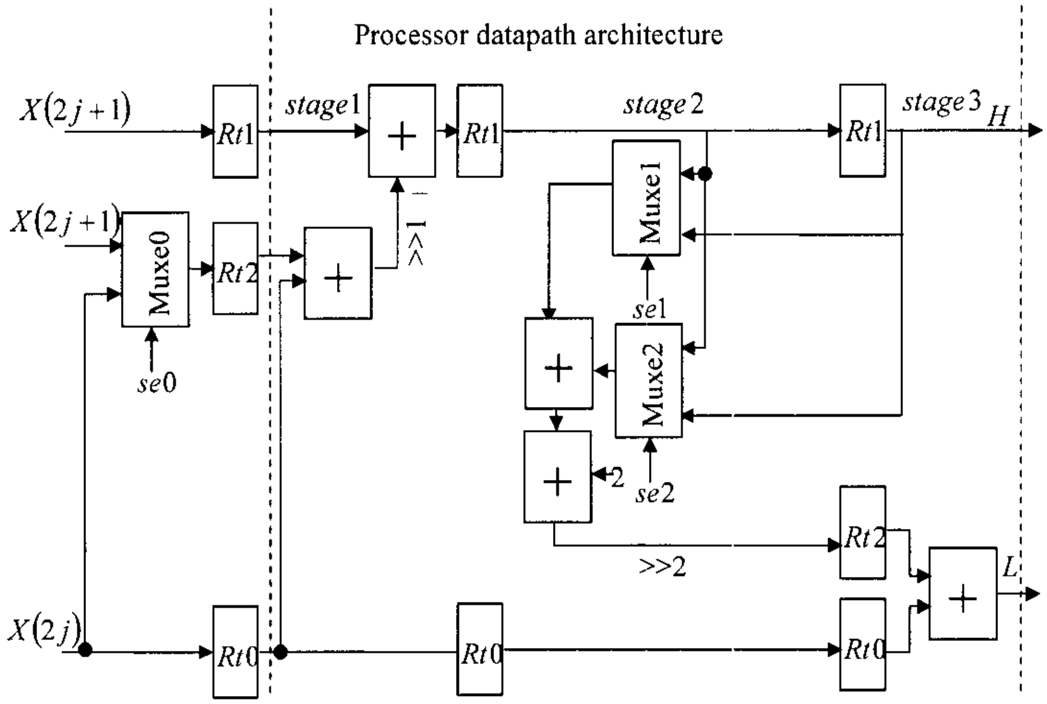


Figure 3.8.1 5/3 processor's datapath architecture with symmetric extension

Table 3.3 symmetric extension's control signals for 5/3

	se0	se1	se2
First	0	0	0
Normal	0	0	1
Last	1	0	1

a) Even length signal

	se0	se1	se2
First	0	0	0
Normal	0	0	1
Last	0	1	1

b) odd length signal

added because stage 4 requires two successive low coefficients that must be produced by stage 2 in order to perform an operation. When the first coefficient produced by stage 2 is in Rt of stage 4 the second coefficient will in Rt of stage 3 and will be applied to stage 4 through the path labeled *forward.* The 9/7 processor shown in Figure 3.8.2, can be thought formed by connecting together two 5/3 processors through stage 3, assuming the 5/3 is a 2-stage pipelined processor.

The multiplexers in stages 2, 4 and 5 including the one labeled Muxe0 implement the symmetric extension algorithm that is part of the DDGs in Figure 3.3.2. Table 3.4 shows the appropriate values of the control signals that must be issued by the control unit to the extension multiplexers so that they perform the required functions. The extension multiplexers in stages 2 and 5 function exactly the same way as that of the 5/3, described earlier. The normal function of the extension multiplexer labeled muxe0 is to pass the input signal *X(2n + 2)* to the latch, whereas function of the extension multiplexer labeled muxe3 in stage 4, is to pass the forward signal, $Y''(2n+2)$ to the adder. Only in the even length signals and in the calculation of the last coefficient, muxe0 passes the input signal X(*2n*) to the latch and Muxe3

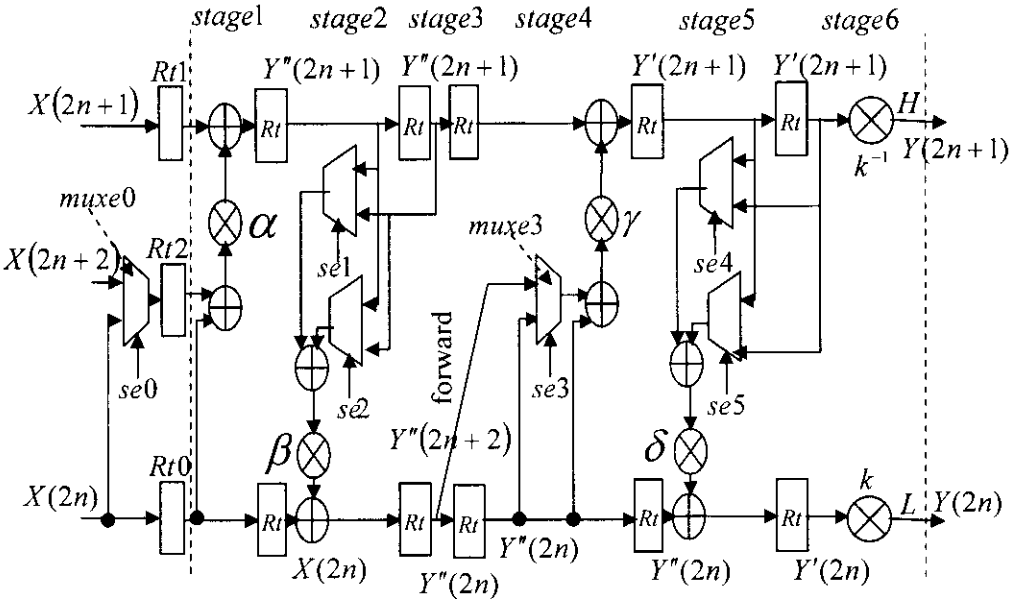


Figure 3.8.2 The 9/7 processor's datapath architecture with extension

Table 3.4 symmetric extension's control signals for 9/7

	step1	step2		step3	step4	
	se0	se1	se2	se3	se4	se5
First	0	0	0	0	0	0
Normal	0	0	1	0	0	1
Last	1	0	1	1	0	1

a) Odd length signals

	step1	step2		step3	step4	
	se0	se1	se2	se3	se4	se5
First	0	0	0	0	0	0
Normal	0	0	1	0	0	1
Last	0	1	1	0	1	1

b) Even length signals

passes the delay signal $Y''(2n)$ to the adder instead of the forward signal $Y''(2n+2)$. Note that multiplication operations in Figure 3.8.2 can be implemented by only two adders as illustrated in [23].

3.8.3 Row and Column Processors for 5/3 and 9/7

The 5/3 and 9/7 processor datapath architectures shown in Figures 3.8.1 and 3.8.2 were developed assuming the external memory is scanned either row-by-row or column-by-column. The CPs in the two architectures shown in Figures 3.6.1 and 3.6.2 for overlapped and nonoverlapped scan methods, respectively, scan the high and the low coefficients generated by RP column-by-column. But, since the CPs alternate, in an interleave fashion, between the high and the low coefficients calculations as indicated in Table B.1, therefore, the 5/3 CP's datapath and both 9/7 CPs' datapath based on the scan method shown in Figures 3.5.1 and 3.5.3, must be modified to allow interleaving in execution. The modified 5/3 and 9/7 CPs' datapath are shown in Figures 3.8.3 and 3.8.4, respectively.

In the 5/3 CP shown in Figure 3.8.3, registers *Rd0* and *Rd1* are added to allow interleaving in execution. The first 9/7 CP shown in Figure 3.8.4(a), which is based on the scan method of Figure 3.5.1, is obtained by splitting stage 3 of the 9/7 processor's datapath shown in Figure 3.8.2 into two stages to allow also interleaving of two columns coefficients in execution. On the other hand, the second 9/7 CP shown in Figure 3.8.4(b), which is based on the scan method shown in Figure 3.5.3, is obtained by splitting stage 3 of the 9/7 processor's datapath of Figure 3.8.2 into four stages and adding 4 registers labeled *R0, R1, R2,* and *R3* in stage 5. The multiplexers labeled *mux*, control the interleaving operations. In the first run, the control signals, sc, of the multiplexers are set 0, to allow in execution the interleaving pattern of run1, as illustrated in the dataflow Table B.2 (a). In all subsequent runs, the multiplexers' control signals are set 1 to allow normal interleaving of two columns.

As for the 5/3 CP in the intermediate architecture shown in Figure 3.7.2, it should be modified as shown in Figure 3.8.5. This is necessary, since the intermediate CP scans three columns in each H and L decomposition in a run as illustrated in the dataflow shown in Table B.3 and alternates between executing 3 high and 3 low operations in H and L decompositions.

On the other hand, the row-processors in the proposed overlapped and nonoverlapped architectures for 5/3 and 9/7 scan the external memory according to one of the scan methods illustrated in Figs 3.5.1, 3.5.2 and 3.5.3. A careful examination of the scan methods and the DDGs shows that the N high coefficients of step1 in the 5/3 and steps 1, 2, and 3 in the 9/7 that were calculated during a run must be kept, in order to be used in the N operations of the next run. This requires the addition of a temporary line buffer (*TLB)* of size N in stage 2 of the 5/3 and in each of stages 2, 3, and 5 of the 9/7. Thus, the RP's datapath that fit into the two proposed architectures is obtained when a *TLB* is incorporated into stage 2 of the 5/3 and in each of stages 2, 3, and 5 of the 9/7 as shown in Figure 3.8.6. The inclusion of the *TLB* may decrease the speed of the architectures. To maintain the speed, the *TLB* can be placed in a separate pipeline stage as shown in Figure 3.8.7. However, inclusion of a TLB causes a problem because the same *TLB's* location must be read and written in the same clock cycle. To solve this problem, the signal labeled $\overline{R}/W$ is connected to the clock *f/3* so that the *TLB* can be read in the first half cycle and written in the second half. The register labeled *TLBAR* (*TLB* address register) generates addresses for the *TLB*. Initially, *TLBAR* is cleared to zero by asserting signal *incar* (increment address register) low to point at the first location. Then to address the next location, after each read and write, register *TLBAR* is incremented by one by asserting *incar* high.

Figure 3.8.7 is appropriate for 5/3 RP in overlapped and nonoverlapped architectures. To obtain the first and the second 9/7 RPs' datapath based on the scan methods of Figures 3.5.1 and 3.5.3, respectively, the 9/7 datapath shown in Figure

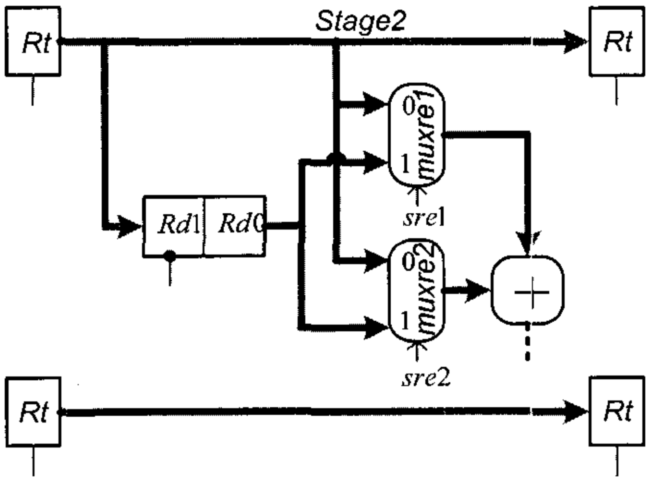


Figure 3.8.3 Modified the 5/3 CP for overlapped and nonoverlapped architectures

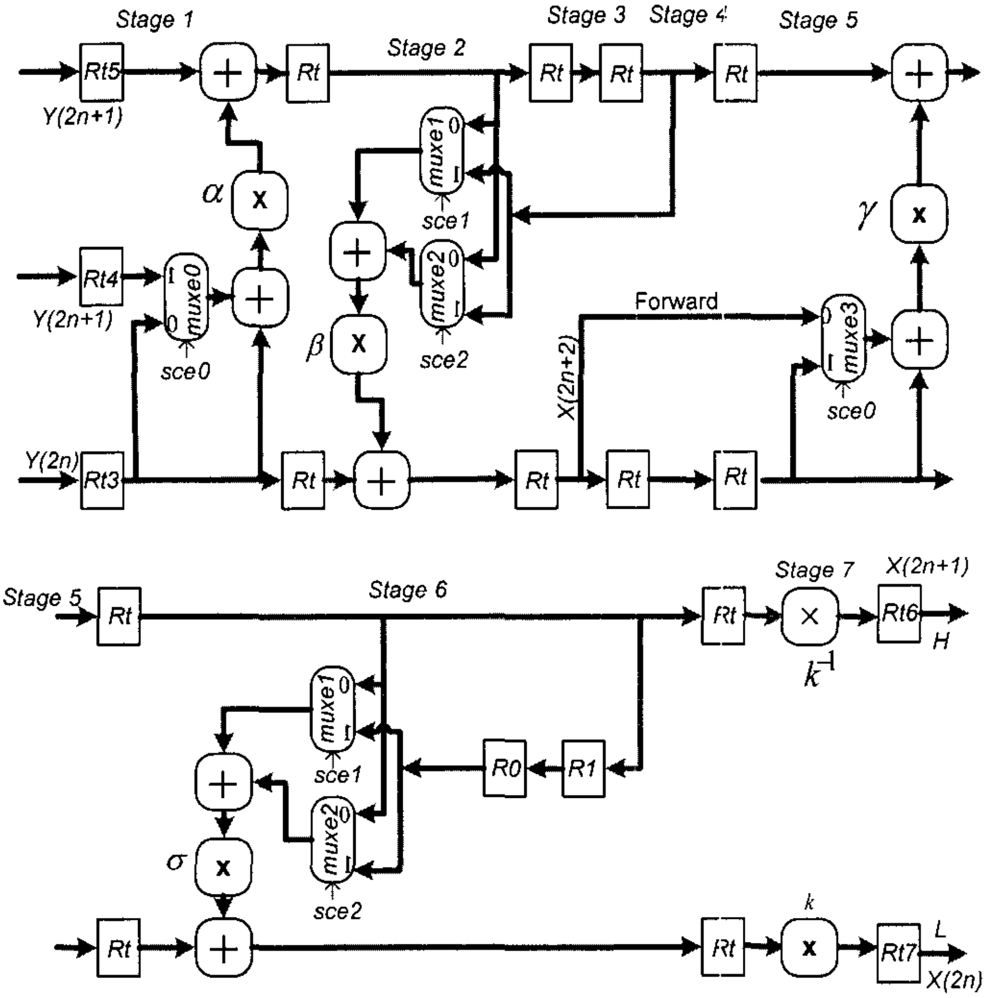


Figure 3.8.4 (a) Modified first 9/7 CP based on the scan method of Figure 3.5.1 for overlapped and nonoverlaped architectures

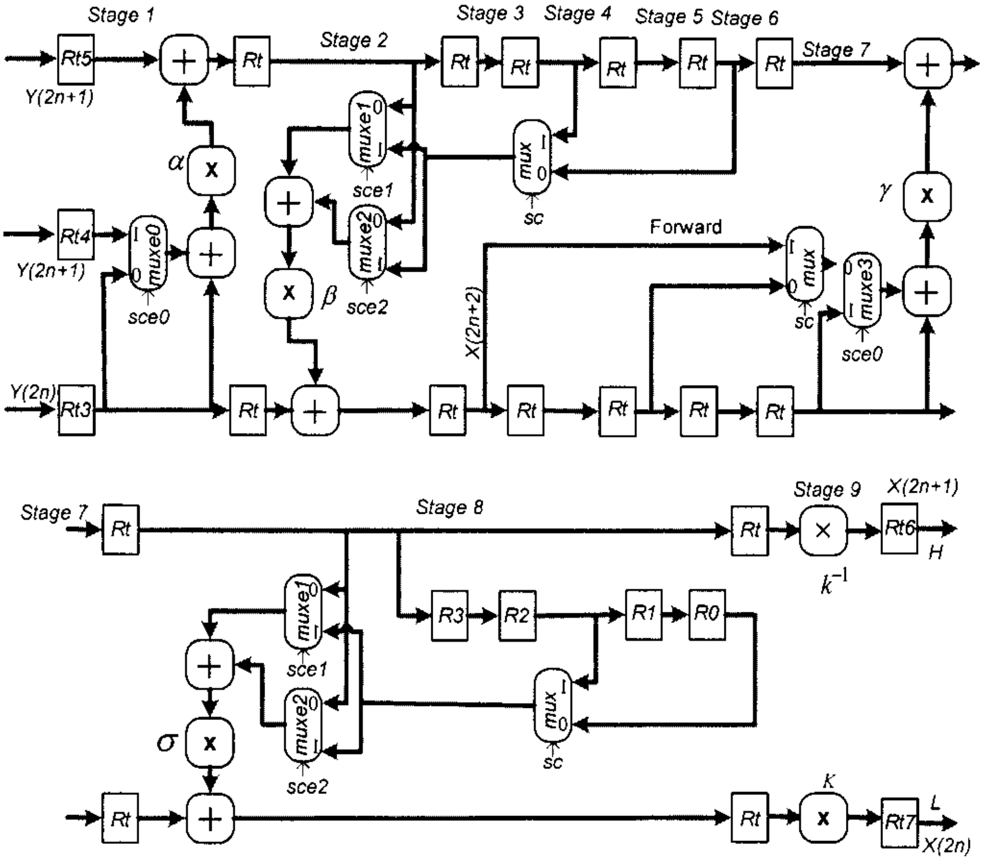


Figure 3.8.4 (b) Modified second 9/7 CP based on the scan method of Figure 3.5.3 for overlapped and nonoverlapped architectures

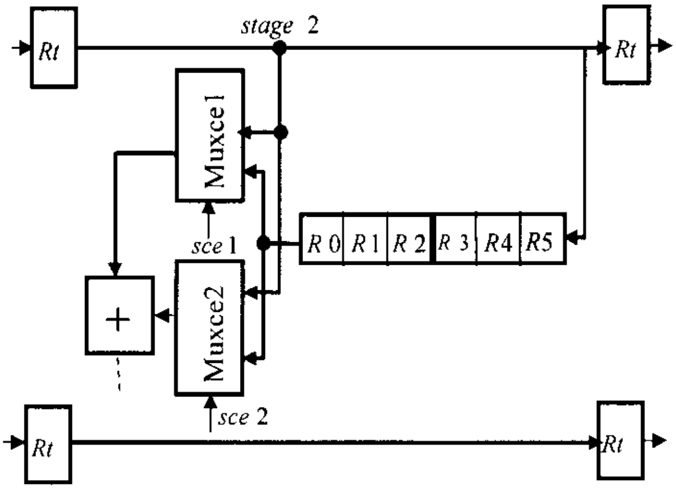


Figure 3.8.5 modified stage 2 of 5/3 CP for intermediate architecture

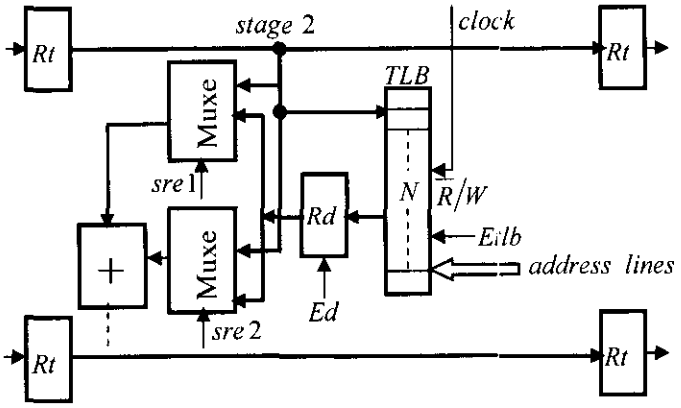


Figure 3.8.6 Incorporation of a TLB in stage 2 of the RP

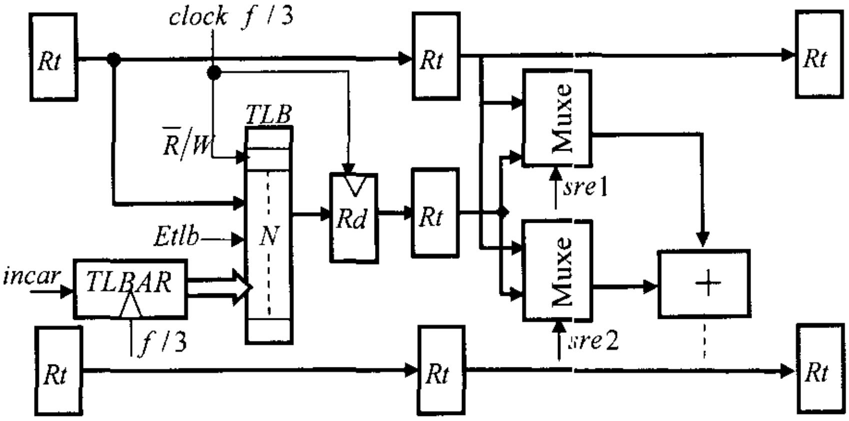


Figure 3.8.7 TLB in a separate pipeline stage

3.8.2 should be modified as shown in Figures 3.8.8 (a) and (b), respectively. The operations of the multiplexers labeled *mux* in Figure 3.8.8 (b) can be controlled by setting the select signals, *sr*, of the multiplexers 0 during the first run and 1 in all subsequent runs.

A careful examination of the 9/7 DDGs shows that when the last run's computations are executed they would not yield all required output coefficients. Thus, to get the remaining output coefficients, the control unit should be instructed to execute one more run, call it, the extra run. In addition, examination of the last run's portion of the 9/7 DDG for odd length signals shows that the extension signal labeled sre1 is required to be set 1 in order to compute the operation in the level labeled $Y''(2n)$ in the DDG. But, when the computation reaches level $Y'(2n)$, the operation in that level requires signal sre1 to be set 0. Furthermore, in the extra run, the operation at level $Y'(2n)$ requires signal *sre1* to be set 1. Therefore, a circuit consisting of an AND gate and an inverter is inserted into stages 6 and 7 of Figures 3.8.8 (a) and (b), respectively. The circuit operates according to Table B.5 (a). However, in the case of even length signals, according to the DDG of the 9/7, both *sre1* and Q1 are set 0 in all runs.

Similarly, examination of signal *sre0*, in the last and extra runs, for both even and odd signals, reveals that this signal should be set also according to Table B.5(a) and the circuit consisting of the AND gate and the inverter should be inserted into stages 4 and 5 of Figures 3.8.8 (a) and (b), respectively. For the architecture developed based on the scan method of Figure 3.5.1, signal *sre2* should be set according to Table B.5 (b) and the circuit consisting of the AND gate the inverter should be inserted into stage 6 of Figure 3.8.8 (a).

Furthermore, to allow TLB3 of Figure 3.8.8 (b) to store coefficients generated by stage 6 in the first run, a circuit consisting of a multiplexer and an inverter is inserted into stage 6 of Figure 3.8.8 (b). In addition, to allow register TLBAR3 to address the first location of the TLB3, when a transition is made from run1 to run2, a circuit consisting of a multiplexer, two inverters, and an AND gate is inserted into stage 5 of Figure 3.8.8 (b).

On the other hand, to obtain the RP datapath for 5/3 and 9/7 intermediate architectures, stage 2 of the 5/3 and stages 2 and 5 of the 9/7 datapath architectures shown in Figures 3.8.1 and 3.8.2 should be modified as shown in Figure 3.8.9. The advantage of this arrangement is that the TLB is not required to be read and written in the same clock cycle.

Furthermore, examination of step2 ($Y''(2n)$) in the 9/7 DDGs shows that the fourth low coefficients labeled $Y''(6)$ calculated for each row in a run using the third intermediate scan method should be stored in a buffer of size N, since they are required in the N operations of the next run. This requires the addition of another *TLB* in stage 3 of the 9/7 datapath architecture shown in Figure 3.8.2. Figures 3.8.10 shows how this *TLB* can be incorporated into stage 3 of Figure 3.8.2 to form the required 9/7 RP for intermediate architecture. The TLB in Figure 3.8.10 is also not required to

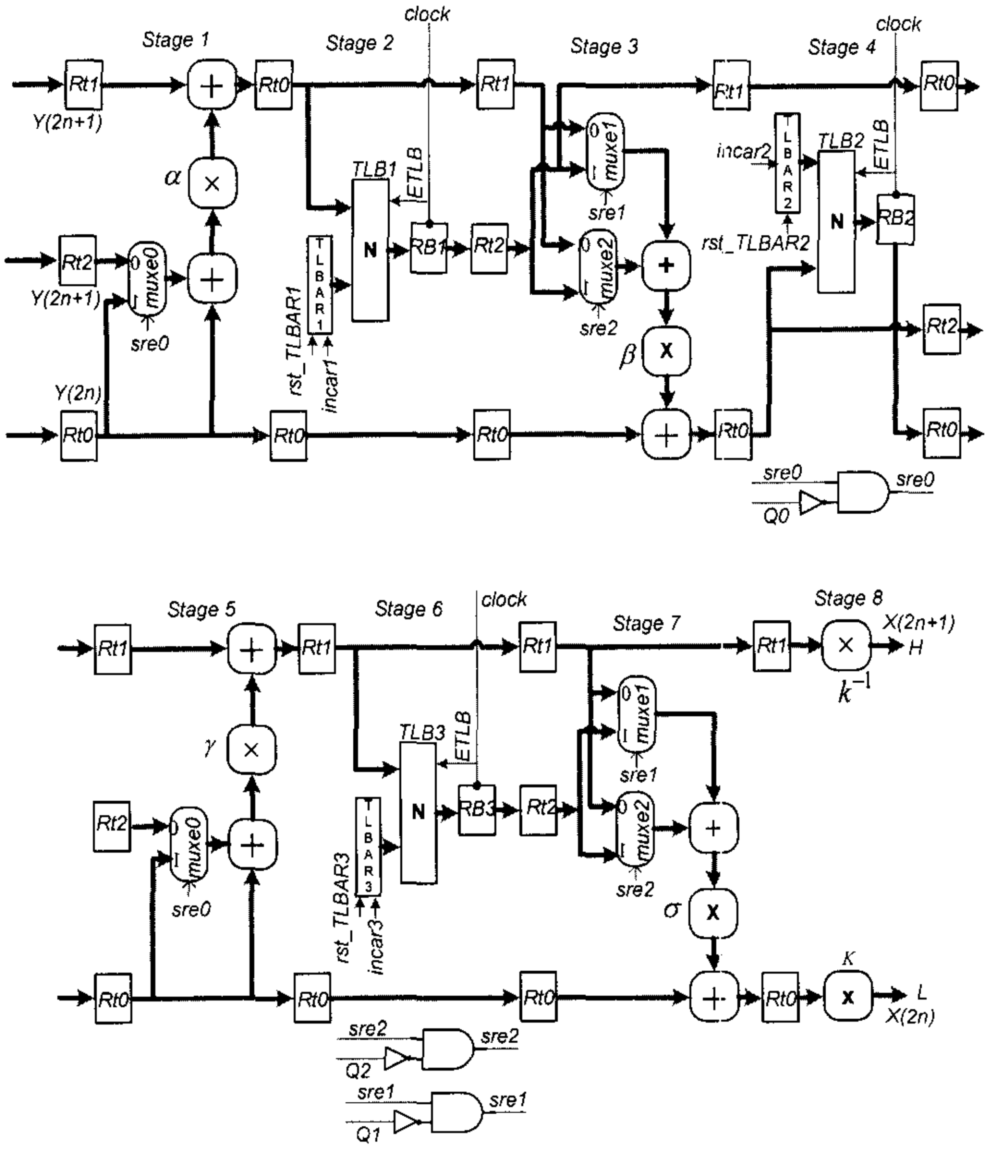


Figure 3.8.8 (a) Modified first 9/7 RP based on scan method 3.5.1 for overlapped and nonoverlapped architectures

be read and written in the same clock cycle. Figures 3.8.9 and3.8.10 form the first 5 stages of the modified 7-stage 9/7 RP for intermediate architecture and the remaining 2 stages are identical to stages 2 and 3 of Figure 3.8.9.

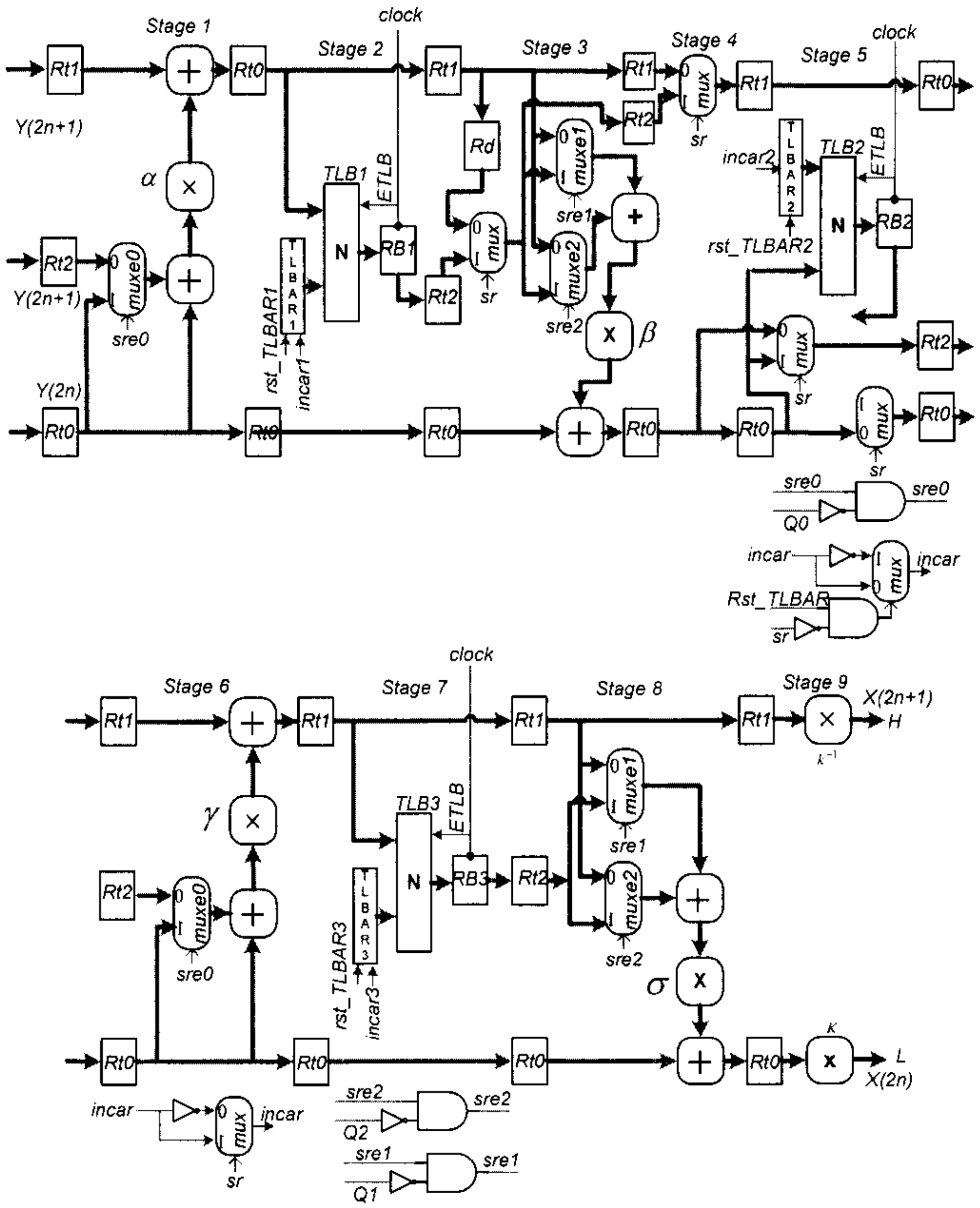


Figure 3.8.8 (b) Modified second 9/7 RP based on scan method 3.5.3 for overlapped and nonoverlapped architectures

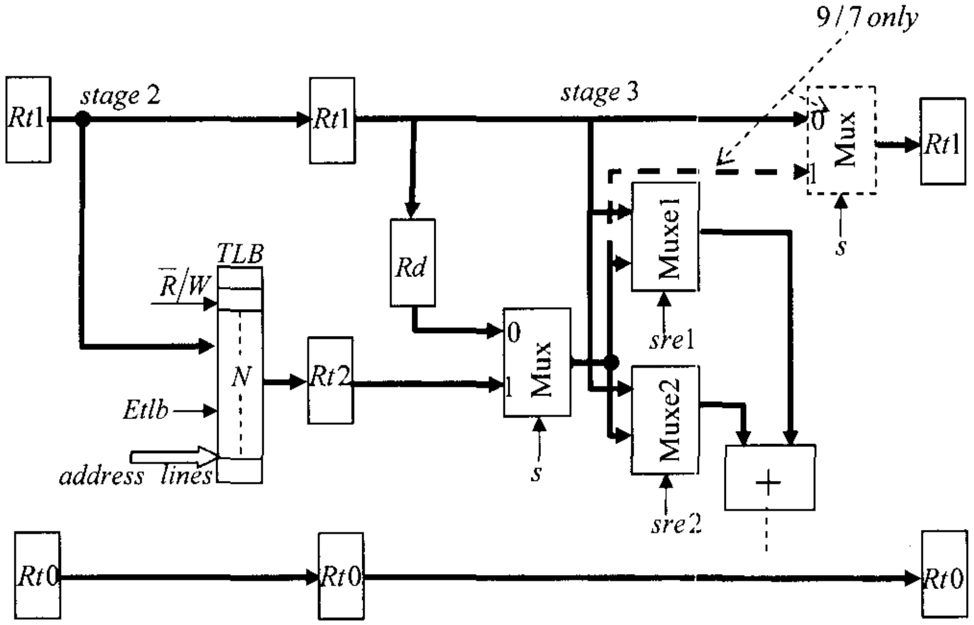


Figure 3.8.9 Modified RP datapath for 5/3 and 9/7 intermediate architectures

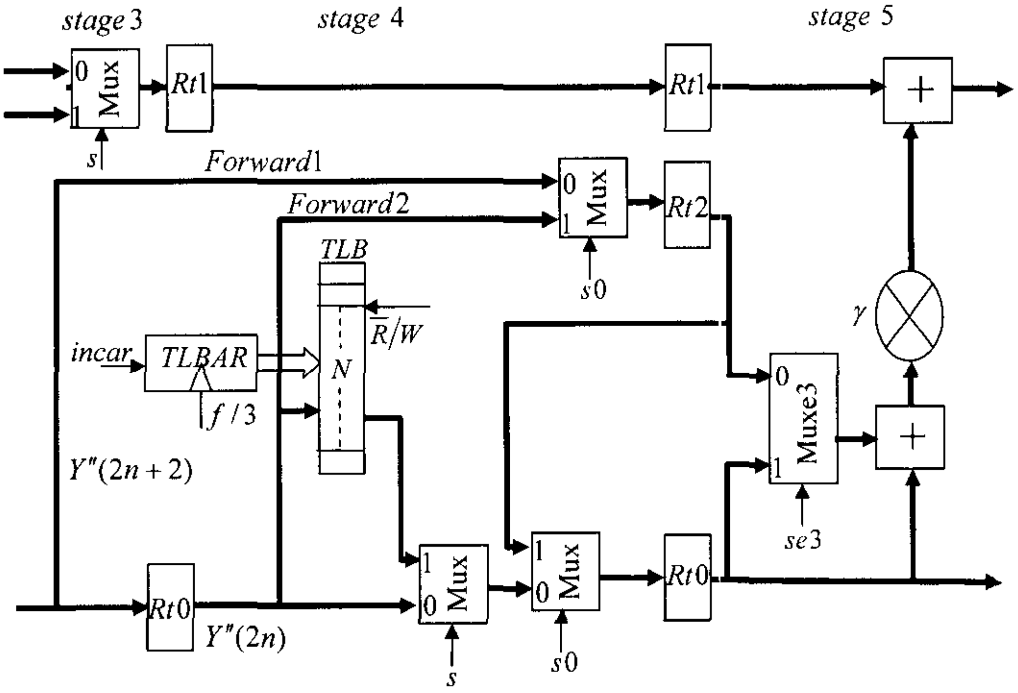


Figure 3.8.10 Incorporation of a TLB in stage 3 of Figure 3.8.2 to form the 9/7 RP for Intermediate architecture

3.9 Evaluation of architectures

In section 3.7.2, it is mentioned that *statement1* can be used to determine the frequency f of the architectures. Pipelining the processors to k stages changes the frequency f, which can be determined by the following statement which is a slight modification of *statement1.*

Statement2

$$
\begin{aligned}
&case1: If\ t_m \geq \frac{t_p}{k}\ then\\
&\qquad \tau = t_m\\
&case2: Else\ if \frac{t_p}{I \cdot k} \geq t_m\ then\\
&\qquad \tau = \frac{t_p}{I \cdot k}\\
&\qquad else\quad \tau = t_m
\end{aligned}
$$

Where $\tau = 1/f$, $t_m = 1/f_m$, and $t_p = 1/f_p$ are the clock period, the critical path delay of the external frame memory and the processors, respectively.

In the algorithm stated above either case 1 or case 2 can be true. Case 2 implies the availability of a very high speed scan that can scan the three pixels required for an operation during the specified time limit given by t_p/k. If that is the case, the architectures shown in Figures 3.6.1, 3.6.2 and 3.7.2 with their processors pipelined, the hardware utilization is 100% and the architectures are complete. Now, suppose τ_1 and τ_2 denote the clock periods of the architectures before and after pipelining, respectively. Then from *statement1*, case2

$$\tau_1 = \frac{t_p}{I}. \tag{3.32}$$

And from *statement2* case2

$$\tau_2 = \frac{t_p}{I \cdot k} = \frac{I \cdot \tau_1}{I \cdot k} = \frac{\tau_1}{k}. \tag{3.33}$$

The speedup factor S is given by

$$S = \frac{\tau_1}{\tau_2} = \frac{\tau_1}{\tau_1/k} = k \tag{3.34}$$

The efficiency E of a k-stage pipeline is defined in [58] as

$$E = \frac{S}{k} = \frac{k}{k} = 1 \tag{3.35}$$

Thus, the architectures with pipelined processors are k times faster than the architectures with nonpipelined processors with efficiency 1.

On the other hand, case 1 implies low scanning frequency. That means the time required to scan the three pixels for an operation will take at least $3t_p/k$ seconds or three clock cycles, where t_p/k is the stage critical path delay of the pipelined processor. In that case, the architectures with pipelined processors will be under utilized 2/3 of the time, since every three clock cycles yield one output. In addition, the speedup due to pipelining is proportional to k. To determine that consider the following. From *statement2* case1,

$$\tau_3 = \frac{t_p}{k} = \frac{I \cdot \tau_1}{k}. \tag{3.36}$$

The speedup factor S is then given by

$$S = \frac{\tau_1}{\tau_3} = \frac{\tau_1}{I \cdot \tau_1/k} = \frac{k}{I} \tag{3.37}$$

The efficiency $$E = \frac{S}{k} = \frac{k/I}{k} = \frac{1}{I} \tag{3.38}$$

Thus, in 9/7 architectures, a gain in speedup factor of 2 can be achieved since $k = 6$ and $I = 3$ but no gain in speedup can be achieved in the case of 5/3 architectures, since $k = 3$, by pipelining the processors and the efficiency is very low, 1/3.

The under utilization and speedup problems can be alleviated, and the entire architecture can be made to operate with frequency $f = k/t_p$ and fully utilized, producing outputs every cycle. If the architecture is allowed to read from the external memory the required three pixels for an operation in parallel every clock cycle instead of one pixel at time. Of course, that will require three buses instead of one to scan the external frame memory. The parallel scan architectures can be obtained by slight modifications of the architectures shown in Figures 3.6.1, 3.6.2, and 3.7.2 from RP side only as shown in Figures 3.9.1, 3.9.2, and 3.9.3, respectively, since modifications only affect this part of the architecture and the other parts remain the same. The 5/3 dataflow of the pipelined parallel scan architectures for overlapped and nonoverlapped in Figures 3.9.1 and 3.9.2, respectively, is shown in Table B.6, whereas the dataflow of the pipelined intermediate parallel scan architecture, Figure 3.9.3, is shown in Table B.7. Tables B.6 and B.7 are derived assuming the RP and the CP are 4- and 3-stage pipelined processor, respectively.

A problem occurs in the line buffer (LB) of Figure 3.9.2 because the same memory location in the line buffer must be read and written in the same clock cycle. To solve this problem, the LB is read in the first half cycle and is written in the second half. To perform this operation the clock line is connected to the control signal labeled $\overline{R}/W$ of the LB. When the clock is low, read takes place and the result is loaded into Rd by the positive transition of the clock and when it is high write operation takes place, as illustrated in Figure 3.9.2. The signal labeled *Elb* (enable LB), when it is asserted high, read and write take place, otherwise, no read and write take place.

To compare the performances of the pipelined parallel scan architectures with the nonpipelined sequential scan architectures shown in Figures 3.6.1 and 3.6.2, consider the following. In the architectures shown in Figures 3.6.1 and 3.6.2, $\rho_1 = 15$ clock cycles (Table B.1) are needed to complete the execution of the first operation, whereas $\rho_1 = 27$ is needed in the intermediate architecture shown in Figure 3.7.2 (Table B.3). The remaining *(n–1)* operations require *I(n*-1*)* cycles, where *I* = *3* for 5/3 and 9/7. Thus, the total time required to perform *(n)* operations or tasks is

$$T(non)_{seq} = [\rho_1 + I \cdot (n-1)]\tau_1 \tag{3.39}$$

where $\tau_1 = 1/f_1$ is the clock period. On the other hand, the pipelined overlapped and nonoverlapped parallel scan architectures shown in Figures 3.9.1 and 3.9.2 require $\rho_3 = 10$ cycles for 5/3 (Table B.6) to complete the execution of the first task, whereas

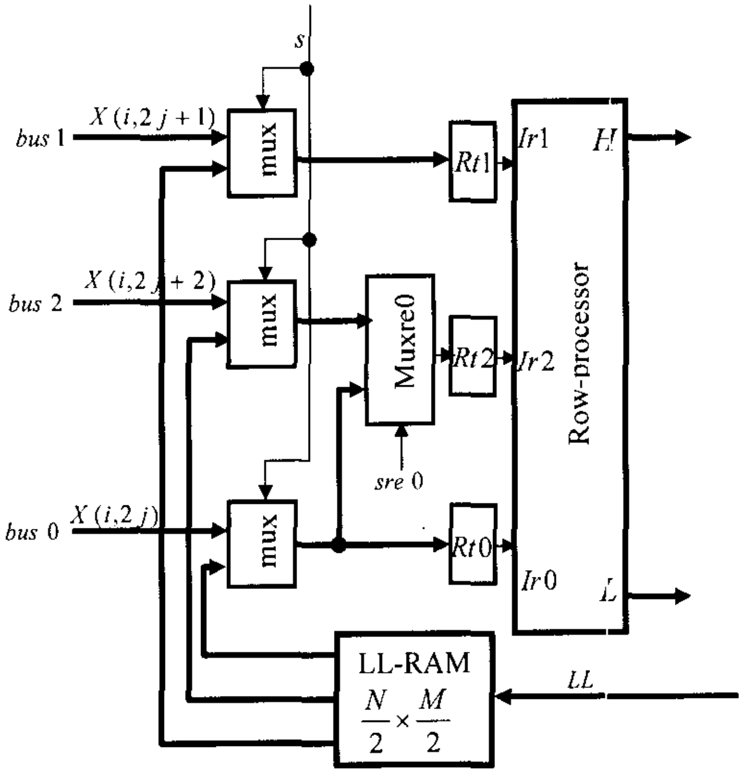


Figure 3.9.1 Pipelined overlapped parallel scan architecture

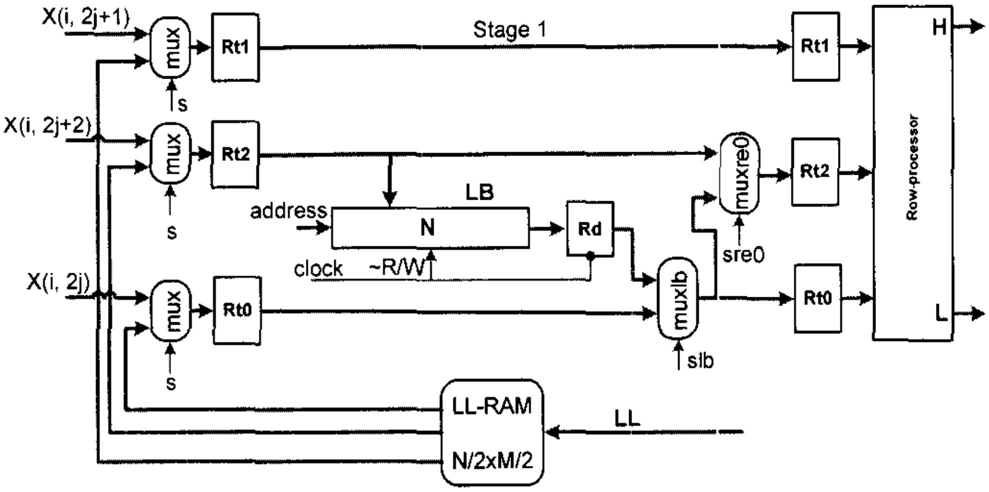


Figure 3.9.2 Pipelined nonoverlapped parallel scan architecture

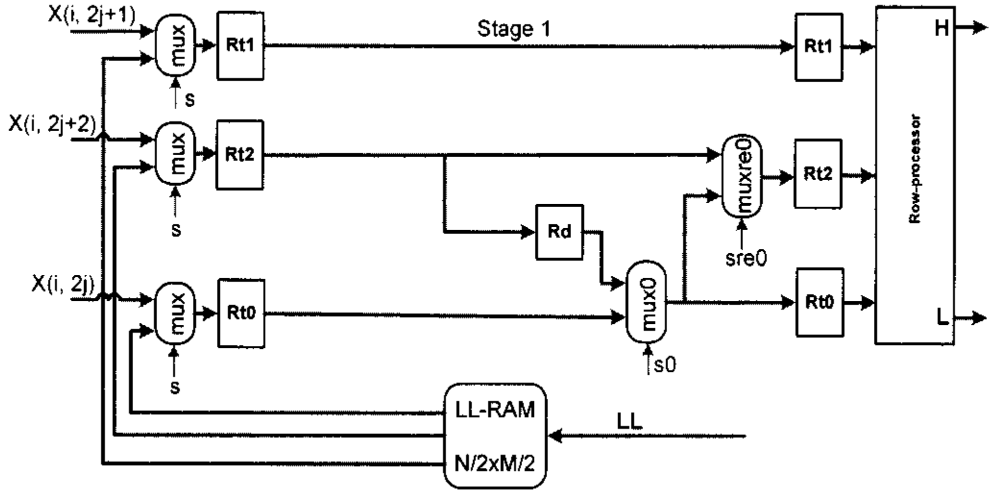


Figure 3.9.3 Pipelined intermediate parallel scan architecture

$\rho_3 = 14$ for 5/3 (Table B.7) is needed in the pipelined intermediate parallel scan architecture shown in Figure 3.9.3. The remaining (*n* – 1*) tasks* require *(n* – 1*)* cycles. The total time required to execute *n* tasks is given by

$$T(pipe)_{par} = [\rho_3 + (n-1)]\tau_3 \tag{3.40}$$

The speedup factor is then given by

$$S = \frac{T(non)_{seq}}{T(pipe)_{par}} = \frac{[\rho_1 + I \cdot (n-1)]\tau_1}{[\rho_3 + (n-1)]\tau_3} \tag{3.41}$$

For large *n*, the above equation reduces to

$$S = \frac{(n-1)(I \cdot \tau_1)}{(n-1)\tau_3} \cong I\frac{\tau_1}{\tau_3} = I\frac{\tau_1}{I \cdot \tau_1/k} = k \tag{3.42}$$

The efficiency

$$E = \frac{S}{k} = \frac{k}{k} = 1 \tag{3.43}$$

That is the pipelined parallel scan architectures are *k* times faster than the nonpipelined sequential scan architectures with efficiency 1.

The throughput, H, which is defined as the number of tasks (operations) performed per unit time, can be written as

$$H(non)_{seq} = \frac{n}{\left[\rho_1 + I \cdot (n-1)\right]\tau_1} = \frac{nf_1}{\rho_1 + I \cdot (n-1)} \qquad (3.44)$$

$$H(pipe)_{seq} = \frac{n}{\left[\rho_2 + I \cdot (n-1)\right]\tau_2} = \frac{nk}{[\rho_2 + I \cdot (n-1)]\tau_1} \qquad (3.45)$$

$$= \frac{nkf_1}{\rho_2 + I \cdot (n-1)} \qquad (3.46)$$

$$H(pipe)_{par} = \frac{n}{[\rho_3 + (n-1)]\tau_3} = \frac{n}{[\rho_3 + (n-1)]I \cdot \tau_1 / k} \qquad (3.47)$$

$$= \frac{nkf_1 / I}{\rho_3 + (n-1)} \qquad (3.48)$$

The maximum throughput, H^{max}, occur when n is very large $(n \rightarrow \infty)$and in these architectures the maximum throughput is attainable, since n is expected to be very large. Thus,

$$H(non)_{seq}^{\max} = f_1 / I \qquad (3.49)$$

and

$$H(pipe)_{seq}^{\max} = H(pipe)_{par}^{\max} = kf_1 / I \qquad (3.50)$$

The pipelined parallel and sequential scan architectures' throughputs have increased by a factor of k as compared with the nonpipelined architectures.

Based on the above evaluations, we can conclude that both pipelined sequential and parallel scan architectures achieve the same performance in terms of speedup, efficiency, and throughput.

To evaluate the power consumption of the pipelined parallel scan architectures shown in Figures 3.9.1, 3.9.2, and 3.9.3 and that of the pipelined sequential scan architectures shown in Figures 3.6.1, 3.6.2, and 3.7.2 consider the following. First, consider the power consumption of the pipelined parallel and sequential scan architectures without external memory. From Eq (3.36) the frequency of the pipelined parallel scan architectures is

$$f_3 = k/t_p \tag{3.51}$$

Whereas from Eq (3.33) the frequency of the pipelined sequential scan architectures is

$$f_2/I = k/t_p \tag{3.52}$$

If the total the capacitance, $C_{total,}$ of parallel and sequential scan architectures are equal, then that implies they are also consume the same power.

On the other hand, the external memory power consumption of the pipelined sequential and parallel scan architectures can be obtained as follow. The total power consumption of the external memory for the pipelined overlapped sequential scan architecture, $P_m(over)_{seq}$ is written as

$$P_m(over)_{seq} = C_{total}^m \cdot V_0^2 \cdot f_2 = C_{total}^m \cdot V_0^2 \cdot I \cdot k/t_p = I \cdot k \cdot C_{total}^m \cdot V_0^2 \cdot f_p \tag{3.53}$$

Where C_{total}^m is the total capacitance of the external memory. The total external memory power consumption for the pipelined nonoverlapped sequential scan architecture, $P_m(nonover)_{seq}$ is written as

$$P_m(nonover)_{seq} = \beta \cdot I \cdot k \cdot C_{total}^m \cdot V_0^2 \cdot f_p \tag{3.54}$$

Whereas the total external memory power consumption of the pipelined intermediate sequential scan architecture, $P_m(int)_{seq}$ can be obtained as follow. If $P_o(int)_{seq}$ is the power consumption due to scanning the overlapped areas of the external memory sequentially is give by

$$p_o(\text{int})_{seq} = \beta_0 \cdot I \cdot k \cdot C_{total}^m \cdot V_0^2 \cdot f_p/3 \tag{3.55}$$

Where $I \cdot k \cdot C_{total}^m \cdot V_0^2 \cdot f_p$ is the external memory power consumption of the pipelined overlapped sequential scan architecture Eq(3.53), then

$$P_m(\text{int})_{seq} = P_m(nonover)_{seq} + P_o(\text{int})_{seq} \tag{3.56}$$

$$= I \cdot k \cdot \beta \cdot C_{total}^m \cdot V_0^2 \cdot f_p + \beta_0 \cdot I \cdot k \cdot C_{total}^m \cdot V_0^2 \cdot f_p/3 \tag{3.57}$$

$$= I \cdot k \cdot f_p \cdot C_{total}^m \cdot V_0^2 (\beta + \beta_0/3) \tag{3.58}$$

On the other hand, the total external power consumption of the pipelined overlapped and nonoverlaped parallel scan architectures $P_m(over)_{par}$ and $P_m(nonover)_{par}$, respectively, are written as

$$P_m(over)_{par} = I \cdot C_{total}^m \cdot V_0^2 \cdot f_3 = I \cdot C_{total}^m \cdot V_0^2 \cdot k/t_p \tag{3.59}$$

$$= I \cdot k \cdot C_{total}^m \cdot V_0^2 \cdot f_p \tag{3.60}$$

$$P_m(nonover)_{par} = \beta \cdot I \cdot k \cdot C_{total}^m \cdot V_0^2 \cdot f_p \tag{3.61}$$

Whereas, the total external power consumption of the pipelined intermediate parallel scan architecture, $P_m(int)_{par}$ can be obtained as follow. If $P_o(int)_{par}$ is the power consumption due to scanning the overlapped areas of the external memory by parallel scan architecture is give by

$$p_o(\text{int})_{par} = \beta_0 \cdot I \cdot k \cdot C_{total}^m \cdot V_0^2 \cdot f_p/3 \tag{3.62}$$

Where $I \cdot k \cdot C_{total}^m \cdot V_0^2 \cdot f_p = P_m(over)_{par}$, then

$$P_m(\text{int})_{par} = P_m(nonover)_{par} + P_o(\text{int})_{par} \tag{3.63}$$

$$= I \cdot k \cdot \beta \cdot C_{total}^m \cdot V_0^2 \cdot f_p + \beta_0 \cdot I \cdot k \cdot C_{total}^m \cdot V_0^2 \cdot f_p/3 \tag{3.64}$$

$$= I \cdot k \cdot f_p \cdot C_{total}^m \cdot V_0^2 \cdot (\beta + \beta_0/3) \tag{3.65}$$

The above evaluations show that the external memory power consumption of the sequential and parallel overlapped architectures are equal (Eqs 3.53 and 3.60) and that of the sequential and parallel nonoverlapped (Eqs 3.54 and 3.61) and the sequential and parallel intermediate (Eqs 3.58 and 3.65).

In the following, an estimate for the total number of operations performed by the row-processor for j levels of decomposition is derived. Number of operations performed by the row-processor in each level of decomposition can be written as

$$n1 = N\left(\left\lceil \frac{M+1}{2} \right\rceil\right) \tag{3.66}$$

$$n2 = \lfloor N/2 \rfloor \left(\left\lceil \frac{\lfloor M/2 \rfloor + 1}{2} \right\rceil\right) \tag{3.67}$$

$$n3 = \lfloor N/4 \rfloor \left(\left\lceil \frac{\lfloor M/4 \rfloor + 1}{2} \right\rceil\right) \tag{3.68}$$

$$n4 = \lfloor N/8 \rfloor \left(\left\lceil \frac{\lfloor M/8 \rfloor + 1}{2} \right\rceil\right) \tag{3.69}$$

.

.

$$nj = \lfloor N/2^{j-1} \rfloor \left(\left\lceil \frac{\lfloor M/2^{j-1} \rfloor + 1}{2} \right\rceil\right) \tag{3.70}$$

Then the total number of operations *(n)* performed by the RP for *j* levels of decomposition can be estimated as

$$n = N\left(\left\lceil \frac{M+1}{2} \right\rceil\right) + \frac{N}{2}\left(\left\lceil \frac{\frac{M}{2}+1}{2} \right\rceil\right) + \frac{N}{4}\left(\left\lceil \frac{\frac{M}{4}+1}{2} \right\rceil\right) + \cdots\cdots + \frac{N}{2^{j-1}}\left(\left\lceil \frac{\frac{M}{2^{j-1}}+1}{2} \right\rceil\right) \tag{3.71}$$

$$n = \frac{1}{2}NM\left(1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \cdots\cdots + \left(\frac{1}{4}\right)^{j-1}\right) + N\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots\cdots + \left(\frac{1}{2}\right)^{j}\right) \tag{3.72}$$

$$n = \frac{1}{2}NM\left(\frac{4 - \left(\frac{1}{4}\right)^{j-1}}{3}\right) + N\left(1 - \left(\frac{1}{2}\right)^{j}\right) = N\left[\frac{1}{2}M\left(\frac{4 - \left(\frac{1}{4}\right)^{j-1}}{3}\right) + \left(1 - \left(\frac{1}{2}\right)^{j}\right)\right] \tag{3.73}$$

$$n \cong \frac{1}{2}NM\left(\frac{4 - \left(\frac{1}{4}\right)^{j-1}}{3}\right) = \frac{1}{6}NM\left(4 - \left(\frac{1}{4}\right)^{j-1}\right) \tag{3.74}$$

Since the term $\left(\frac{1}{4}\right)^{j-1}$ *will be very small the above equation can be reduced to*

$$n \cong \frac{2}{3}NM \tag{3.75}$$

Eq (3.75) also estimates the total number of operations performed by the CP and the total number of paired outputs for *j* levels of decomposition.

3.10 Combined 5/3 and 9/7 Architecture

The 9/7 processor datapath architecture of Figure 3.8.2 can be viewed as formed by connecting two 5/3 processors through stage 3, assuming 5/3 is a 2-stage pipelined processor. That suggests the possibility of modifying the 9/7 processor datapath architecture shown in Figure 3.8.2 such that it performs both 9/7 and 5/3 algorithms. To obtain such processor architecture the 5/3 algorithm is incorporated in stages 1, 2, and 3 of Figure 3.8.2 as shown in Figure 3.10.1. The control signal value of the signal labeled $\overline{lossless}/lossy$ determines which function the architecture would perform. If $\overline{lossless}/lossy$ is 0, the architecture performs the lossless 9/7, otherwise, performs the lossy 5/3. The combined architecture is useful and very efficient in situations where the encoder in one site is required to perform either lossless or lossy image compression. The advantage of the combined architecture is that a substantial saving in silicon area could be achieved.

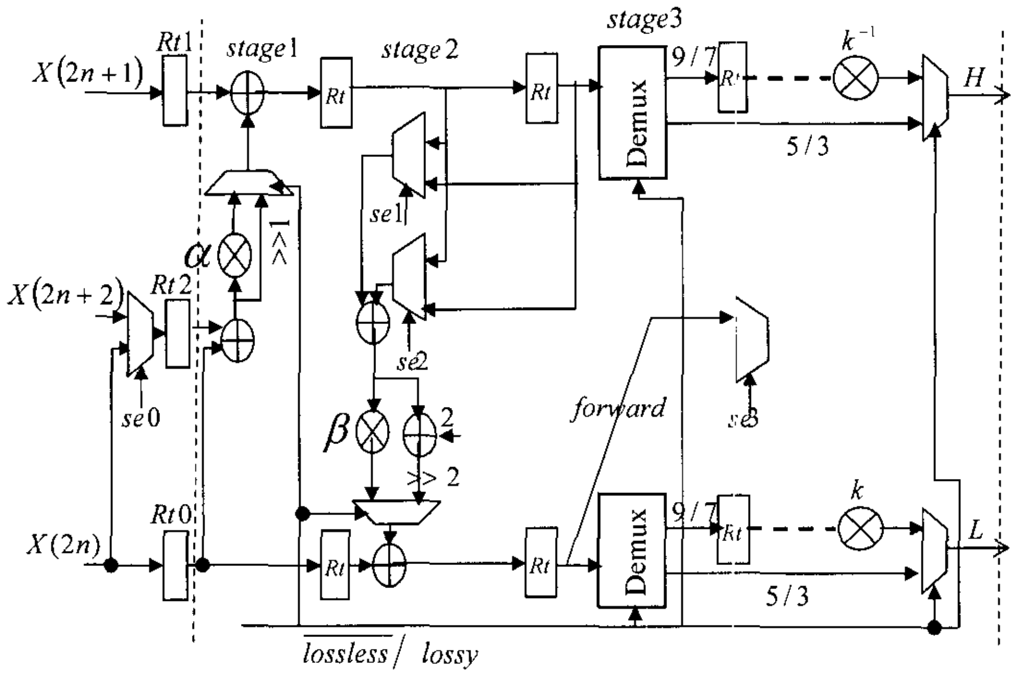


Figure 3.10.1 Combined 9/7 and 5/3 processors datapath architecture

3.11 Conclusions

In this chapter, 3 high-speed and novel pipelined VLSI architectures, overlapped, nonoverlapped, and intermediate architectures were developed for 5/3 and 9/7, respectively. Pipelining technique is utilized to achieve high-speed performance. The advantage of the overlapped and intermediate architectures is that they only require a total temporary line buffer (TLB) of size N and 3N for 5/3 and 9/7, respectively. The intermediate architecture, which is an alternative form for reducing the power consumption of the overlapped areas of the external memory expressed in Eq(3.9), reduces the external memory power consumption by 22.22 % as compared with the external memory power consumption of the architecture based on the first overlapped scan method. However, the intermediate architecture with the second dataflow Table B.4 reduces the power consumption of the external memory by 48%. Therefore, intermediate architecture could be a very good candidate in applications where power consumption is a serious issue.

CHAPTER 4

PARALLEL ARCHITECTURES DEVELOPMENT

4.1 Introduction

In chapter 3, three pipelined architectures were developed. The first architecture, which is based on the first overlapped scan method, the maximum power consumption occurs due to overlapped external memory access. The second architecture, which is based on the nonoverlapped scan method, the power consumption of the external memory has been reduced to minimum by eliminating the overlapped areas but requires the addition of a line buffer (LB) to the architecture. The intermediate architecture, which is based on the generalized overlapped scan method, is introduced to reduce the power consumption of the external memory access, without using the expensive line buffer, to somewhat between that based on the first scan method and that based on the nonoverlapped scan method.

In this chapter, to further increase the performance in order to closely meet real-time applications of DWT with demanding requirements, the parallel architectures based on the first scan method and the parallel form of the intermediate architectures will be designed. First, the parallel architectures based on the first overlapped scan method will be developed followed by the intermediate parallel architectures.

In general, the scan frequency f_l and hence the period $\tau_l = 1/f_l$ of parallel pipelined architectures can be determined by the following statement, when the required pixels I of an operation are scanned simultaneously in parallel. Suppose t_p and t_m are the processor and the external memory critical path delays, respectively.

<u>Statement3</u>

$$\begin{aligned} &\textit{If } \ t_p/l \cdot k \geq t_m \ \ \textit{then} \\ &\qquad \tau_l = t_p/(l \cdot k) \\ &\textit{else} \ \ \tau_l = t_m \end{aligned}$$

Where l = 2, 3, 4 ... denote 2, 3, and 4-parallel and t_p/k is the stage critical path delay of a k- stage pipelined processor.

4.2 parallel architectures based on first scan method

In this section, three parallel architectures based on the first overlapped scan method will be developed for 5/3 and 9/7 2-D DWT algorithms. These three parallel architectures will be referred to as

- 2-parallel pipelined architecture.
- 3-parallel pipelined architecture.
- 4-parallel pipelined architecture.

The 2-parallel, the 3-parallel, and the 4-parallel architectures each increases the speedup by a factor of 2, 3, and 4, respectively, as compared with the single pipelined architecture based on the first scan method developed in chapter 3.

4.2.1 2-parallel pipelined external architecture

Based on the first overlapped scan methods shown in Figures 3.5.1 and 3.5.3 and DDGs for 5/3 and 9/7, respectively, the 2-parallel architecture shown in Fig. 4.2.1 is developed for 5/3 and 9/7. The architecture is valid for both 5/3 and 9/7 algorithms, since it is developed based on the observation that the DDGs for 5/3 and 9/7 are identical when they are looked at from outside, taking into consideration only inputs and outputs requirements.

The architecture consists of 2 k-stage pipelined row-processors labeled RP1 and RP2 and 2 k-stage pipelined column-processors labeled CP1 and CP2. The architecture scans external memory with frequency f_2 and it operates with frequency $f_2/2$. The buses labeled *bus0, bus1,* and *bus2* are used for transferring in every clock cycle 3 pixels from external memory to RP's latches *Rt0, Rt1,* and *Rt2.* The RP1's latches load data every time clock $f_2/2$ makes a positive transition, whereas RP2's latches load data every time a negative transition occurs as indicated in Figure 4.2.1, assuming the first half pulse of the clocks f_2 and $f_2/2$ are low.

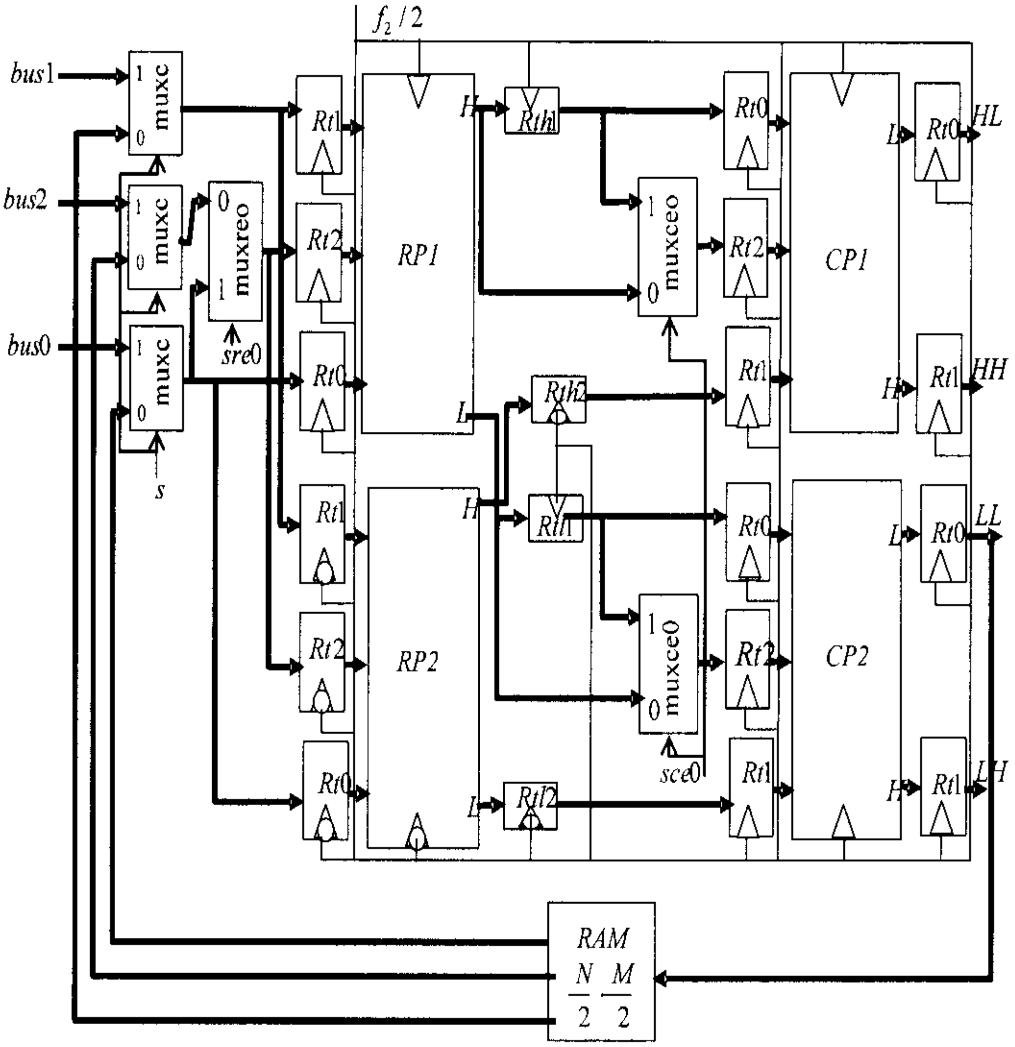


Figure 4.2.1 2-parallel pipelined external architecture

On the other hand, the column-processors CP1 and CP2 and their associated latches load new data every time clock $f_2/2$ makes a positive transition.

The DDGs for even length signals show that in the last high and low coefficients calculations, only the last two pixels in a row, r, at locations X(r, M-2) and X(r, M-1) are read from external memory. In addition, the extension part of the DDGs for even length requires the pixel located at X(r, M-2) to be considered as the first and the third inputs. This pixel must be passed to the RP2 with the second input pixel from location X(r, M-1), to compute the last high and low coefficients in row r. Thus, the multiplexer labeled *muxre0*, which is an extension multiplexer, passes in all cases data coming through bus2, except when the row length *(M)* of an image is even and only in the calculations of the last high and low coefficients in a row *r*, the pixel of location *X (r,M-2)*, which will be read into bus0, must be allowed to pass through *muxre0* and then loaded into *Rt2* as well as *Rt0*. The two multiplexers labeled *muxce0*, attached to CPs, are also extension multiplexers and operate similar to *muxre0* when DWT is applied column-wise by CPs. The three multiplexers labeled *muxc* allow either the external memory or the LL-RAM data to be passed to the RP's latches *Rt0, Rt1,* and *Rt2*.

On the other hand, when the row length of an image is odd, according to the DDGs for odd length signals, to calculate the last low coefficient only one pixel the last one at location X(r, M-1) should be passed to the RP1.

The dataflow of the architecture is shown in Table B.8. This dataflow table is derived based on the 5/3 scan method shown in Figure 3.5.1 and it is identical to 9/7 dataflow except in the first run, where 9/7 scan method shown in Fig. 3.5.3 requires scanning of 5 pixels from each row. The 5/3 scan method shown in Figure 3.5.1 is also a valid scan method for 9/7 and the dataflow for 5/3 shown in Table B.8 would be identical to 9/7 dataflow derived using 5/3 scan method except in the first run where 9/7, according to its DDGs, would not be able to yield any output coefficients. The 9/7 RPs in the first run will be able to compute only two coefficients labeled $Y''(1)$ and $Y''(0)$ in the DDGs for each row of run 1 and these coefficients can be stored in TLBs so that they can be used in the next run computations. Inclusion of TLBs will be discussed later when modified RP datapath architecture is developed.

The utilization of the 5/3 scan method as a unified scan method for both 5/3 and 9/7 gives many advantages:

- Similar control algorithms, if not identical, can be used for both 5/3 and 9/7.
- Ease of integration of the 5/3 into the 9/7 processor datapath architecture for combined 5/3 and 9/7 architecture.

For these two reasons, the 5/3 scan method as unified scan method for both 5/3 and 9/7 is preferred and therefore, will be used in all parallel architectures developed in this chapter.

Note that according to the first overlapped scan method shown in Figure 3.5.1, in any particular time 3 columns are considered for scanning and in every clock cycle 3 pixels are scanned one from each column until end of the columns are reached, say, to complete a run. Then a transition is made to the beginning of the next 3 columns to initiate another run. In the clock cycle where a transition occurs, especially when column length of an image is odd, the external memory should not be scanned since during that cycle the two CPs each will compute the last low coefficient as required by the DDGs for odd length signals. That is, during that cycle no pixel is loaded into RP2 latches while the control is allowed to return to RP1 by the pulse ending the cycle. This also implies that each run will begin at RP1 and the high coefficients generated during a run, which are required in the next run computations, will be stored in the TLB of the RP that generated them.

Figure 4.2.2 shows how stage 2 of the pipelined 5/3 RP and stages 2, 3 and 5 of the pipelined 9/7 RP should be modified when they are incorporated into the 2-parallel architecture processors. The modifications require addition of a *TLB* size of *N/2* in each stage mentioned. The *TLB* is necessary, according to the DDGs, to keep N coefficients calculated during a run in each of stages 1, 2, and 4 of Figure 3.8.2 that are also needed in the N operations of the next run. Signal $\overline{R}/w$ (read/write) is connected to the clock $f_2/2$ in Figure 4.2.2 so that the *TLB* can be read in the first half cycle and written in the second half as required. The data read in the first half cycle, for example, from *TLB1*, is stored in register Rd1 by the negative edge of the clock. Then the positive edge of the clock loads it into the latch of the next stage. Note that each of the 2-parallel 9/7 RP is identical to the RP shown in Figure 3.8.8 (a).

The register labeled *TLBAR* (TLB address register) generates addresses for the *TLB*. Initially, register *TLBAR* is cleared to zero by asserting signal *incar* low to point at the first location in the TLB. Then to address the next location after each read and write, register TLBAR is incremented by one by asserting *incar* (increment address register) high.

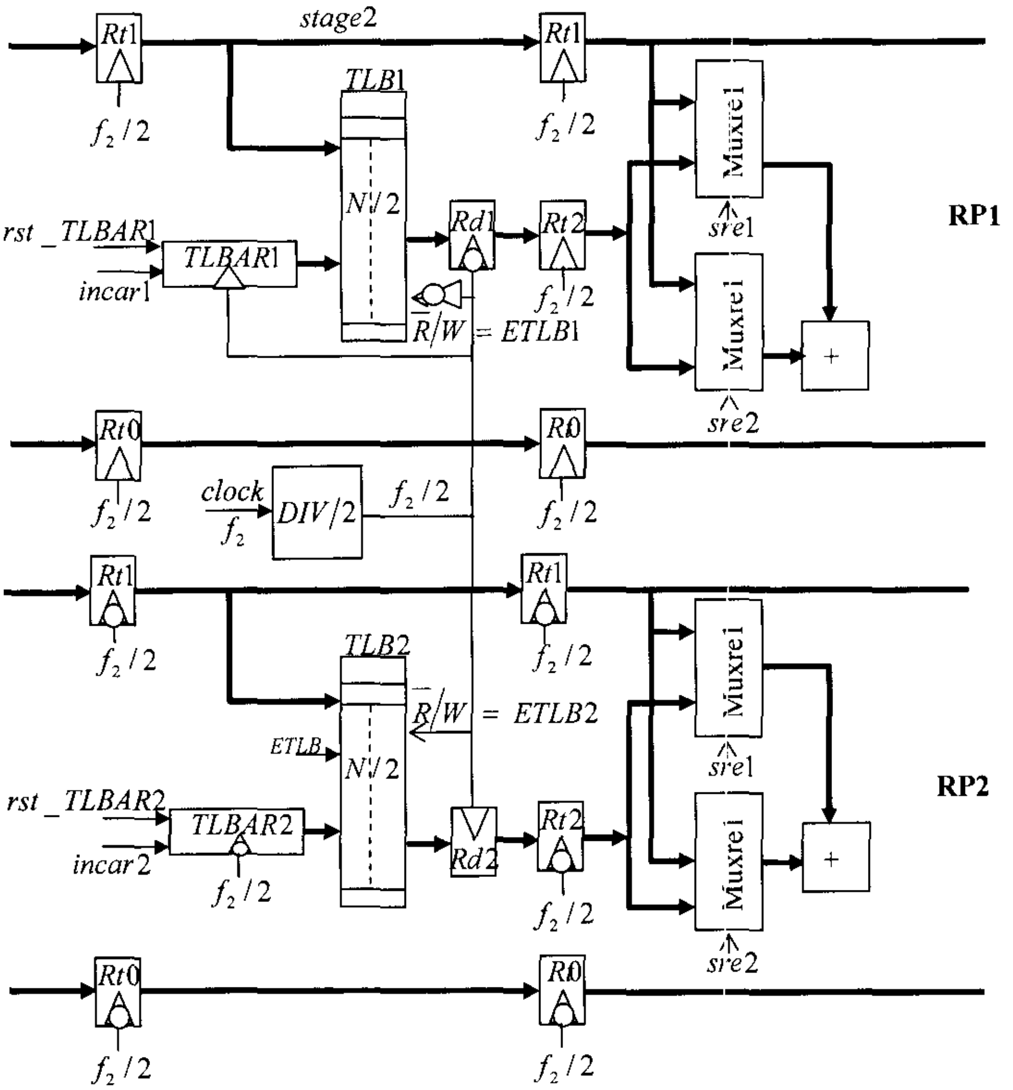


Figure 4.2.2 modified 2-parallel RPs

4.2.2 3-parallel pipelined architecture

The 3-parallel pipelined architecture is shown in Figure 4.2.3 and its dataflow based on 5/3 scan method shown in Figure 3.5.1 is given in Table B.9. The architecture has two more processors, labeled RP3 and CP3, than the 2-parallel architecture shown in Figure 4.2.1. The architecture operates with frequency $f_3/3$ and scans the external memory with frequency f_3.

Figure 4.2.4 shows two waveforms for the frequency $f_3/3$ labeled f_{3a} and f_{3b}.

The RP1 and its associated latches use the clock f_{3a} , whereas the RP2 and the RP3 and their associated latches use the clock f_{3b} as indicated in Figure 4.2.3.

In every clock cycle, 3 pixels are scanned from external memory and are loaded into the latches of one of the RPs. First, RP1 latches are loaded then RP2 latches followed by RP3 latches and then the process repeats. The 3 row-processors latches should be loaded with the required data during the time limit specified by t_p/k before

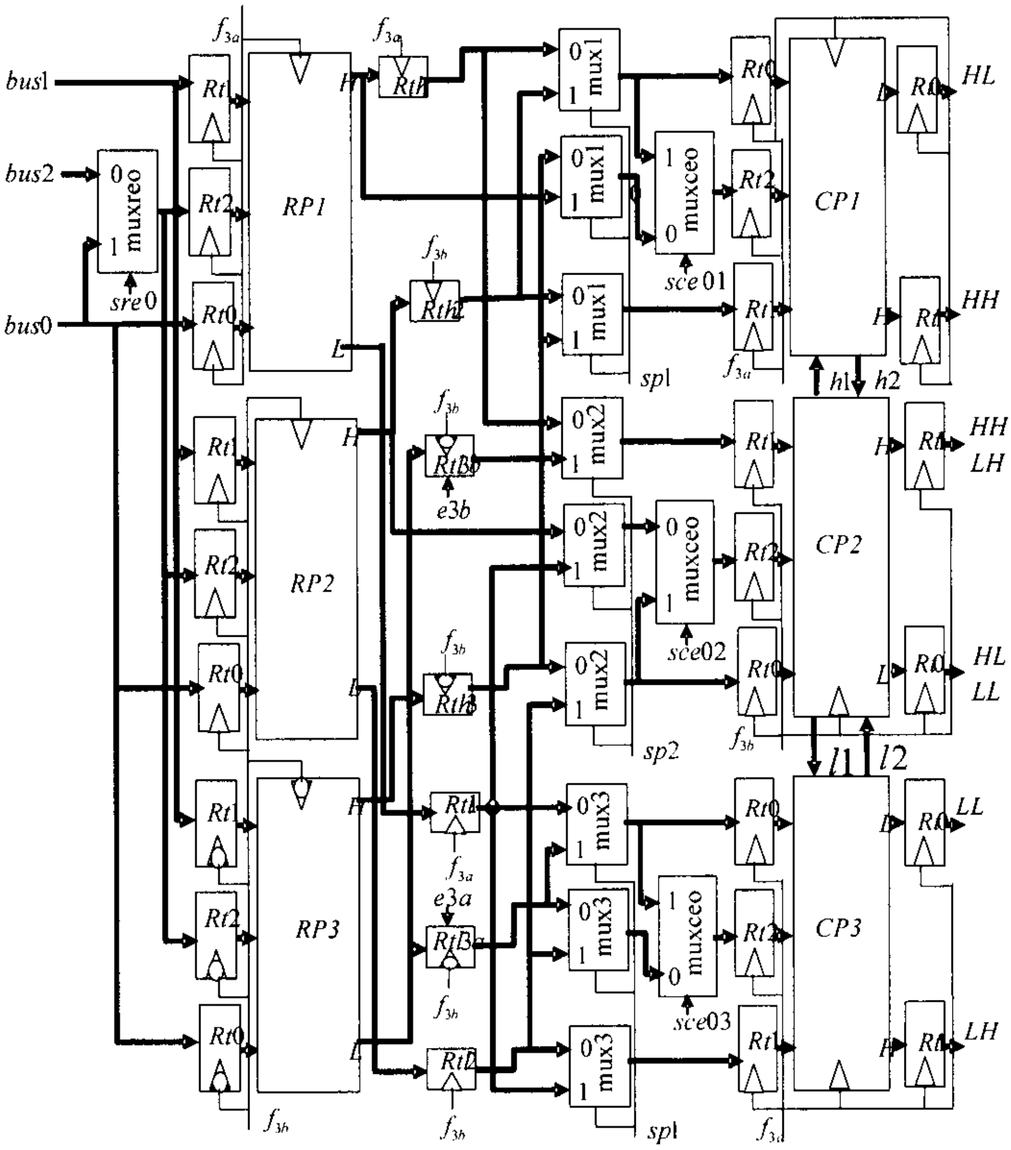


Figure 4.2.3 3-parallel pipelined architecture

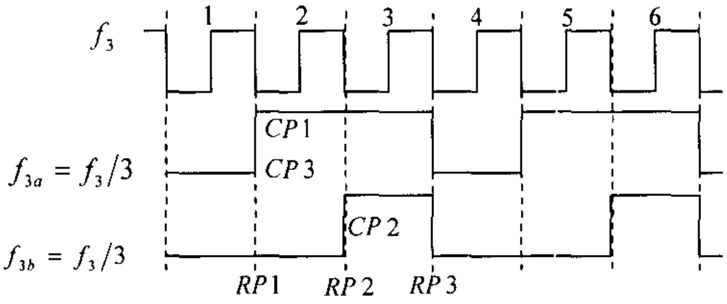


Figure 4.2.4 waveforms of the 2 clocks used in 3-parallel

it repeats. The RP1 and RP2 latches are loaded every time clocks f_{3a} and f_{3b} make a positive transition, respectively, whereas RP3 latches are loaded each time clock f_{3b} makes a negative transition.

The extension multiplexer's labeled *muxre0* and *muxce0* in Figure 4.2.3, function the same way as in the 2-parallel extension multiplexers described in section 4.2.1. In addition, note that the RP3 has two *Rtl* output latches labeled *Rtl3a* and *Rtl3b* instead of one because the dataflow in Table B.9 requires the presence of such latches. These latches are required to hold its contents sometime for more than one clock cycle with respect to clock f_{3b}. Therefore, the control signals *e3a* and *e3b* are added to control the loading of these two latches

The strategy adopted in this architecture is that each run must begin at RP1. The advantage of the strategy is that it will not require any modifications to the RPs datapath architecture shown in Figure 4.2.2 except the 3 RPs in the 3-parallel each will has a TLB of size $\lceil N/3 \rceil$, while any other strategy will complicate very much the RPs datapath and the control circuitry. Application of this strategy requires that if a run ends at RP1, then the next run should begin after 2 clocks cycles during which the external memory is not scanned whether the column length *(N)* is even or odd. But, if a run ends at RP2, then the next run must begin after one clock cycle. The external memory is not scanned also during this cycle whether N is even or odd.

On the other hand, if a run ends at RP3 and *N* is even, then the next run can begin immediately, otherwise, if *N* is odd, then 3 clock cycles must elapse before the next run can begin. These guidelines are necessary in order to avoid any conflict in the dataflow. To identify at which RP a run would end, a 2-bit register can be used. The register is initially set to 0 and then is incremented by one every clock cycle to count from 1 to 3 and repeats. When a run ends the 2-bit register will contain the RP number.

Now, let's move to the CPs side to see how this part of the architecture works. According to the dataflow shown in Table B.9, CP1 and CP3 work in parallel starting from cycle 13. However, CP1 executes high coefficients stored in *Rth1, Rth2,* and *Rth3*, while CP3 executes low coefficients stored *Rtl1, Rtl2,* and *Rtl3*. Whereas, starting from clock cycle 14, the CP2 alternates between executing high and low coefficients. Moreover, both CP1 and CP3 are run by the clock labeled f_{3_a} and every time it makes a positive transition new data are loaded simultaneously into both CP1 and CP3 latches *Rt0, Rt1,* and *Rt2*. CP2 is run by the clock f_{3_b} and loads new data into its latches *Rt0, Rt1,* and *Rt2* every time the clock makes a positive transition.

In order to understand and appreciate why the 3 sets of the multiplexers labeled *mux1, mux2,* and *mux3* are included, why they are interconnected in that way, and finally, how they operate, consider Table 4.1. Table 4.1 is obtained from Table B.9 and it lists groups of RPs' output latches, identified in the table as patterns, and shows how they are scheduled for the CPs. As shown in Table B.9 in cycle 13, pattern1 latches are scheduled for CP1and CP3. In cycle 14, pattern2 latches are scheduled for CP2. In cycle 16, pattern 3 latches are scheduled for CP1 and CP3, whereas in cycle 17, pattern4 latches are scheduled for CP2. These scheduling patterns again repeat starting from pattern1 and so on. Thus, looking at pattern 1 and pattern 3 latches, the presence and interconnections of the three CP1 multiplexers labeled *mux1* and the three CP3 multiplexers labeled *mux3* can be justified. In Figure 4.2.3, pattern1 latches are connected to the inputs of the multiplexers labeled 0, whereas pattern3 latches are connected to inputs labeled 1.The operation of the two set of the multiplexers can be controlled by one signal labeled $sp1$. First, $sp1$ is set to 0 to schedule pattern1 and then is set to 1 to schedule pattern 3 and so on.

Similarly, looking at pattern 2 and pattern 4 latches, which are used by CP2, the inclusion of the three multiplexers, labeled *mux2* and their interconnections can be

Table 4.1 Shows scheduling patterns for CPs and registers involved.

Pattern	RP's output latches	CP
1	*Rth1* *Rtl1* *Rth2* *Rtl2* *Rth3* *Rtl3a*	1 & 3
2	*Rth3* *Rth1* *H2*	2
3	*Rth2* *Rtl3a* *Rth3* *Rtl1* *H1* *Rtl2*	1 & 3
4	*Rtl2* *Rtl3b* *Rtl1*	2

verified. In the architecture, pattern 2 latches are connected to the inputs of the multiplexers labeled 0, whereas pattern4 latches are connected to the inputs labeled 1. The operations of these multiplexers are controlled by one signal labeled *sp2*. First, *sp2* is asserted low to schedule pattern 2 and then high to schedule pattern 4 and so on.

On the other hand, examination of tables B.9 and 4.1 starting cycle 12 until cycle 17 shows that the control signal values for signals *e3a*, *e3b*, *sp1*, and *sp2* can be derived as shown in Table 4.2. These signal values repeat every 6 clock cycles. In addition, as indicated in the table, signals *sp1*and *sp2* can be combined into one signal *sp*.

According to the DDGs for 5/3 and 9/7, a high coefficient calculated in a previous operation is also required in the calculation of the next operation. This implies, since

Table 4.2 Control signal values

Cycle number	*e3a*	*e3b*	*sp1*	*sp2*	*Sp*
12	1	0	X	X	0
13	0	0	0	X	0
14	0	0	X	0	0
15	0	1	X	X	0
16	0	0	1	X	1
17	0	0	X	1	1

CP2 interleave in execution coefficients of both H and L decomposition generated by the RPs, then it should be able to pass the high coefficients it generates to CP1 and CP3, and receive high coefficients generated by CP1 and CP3. Therefore, the paths, labeled *h*1, *h2, l*1, and *l2,* are added in Fig. 4.2.3 to serve this purpose.

In order for the CPs to exchange these high coefficients properly, the CPs datapath architecture, specifically stage 2 of the 5/3 and stages 2 and 5 of the 9/7 should be modified as shown in Figure 4.2.5. Table 4.3 provides the information necessary for passing high coefficients between CPs. This table is used as mean in implementing the modifications shown in Figure 4.2.5. Therefore, understanding of Table 4.3 is essential to appreciate the changes that have been incorporated into Figure 4.2.5.

Table 4.3 shows that in cycle 16, CP1 and CP3 generate the high coefficients HH0,0 and LH0,0, which are placed in *Rt*1 and *Rt3*, respectively, by the pulse ending the cycle. The pulse ending cycle 17 loads HH0,0 into *Rd2* of the CP2 as indicated by the arrow labeled 1. Similarly, the pulse ending cycle 20 transfers the high coefficient LH1,0 stored in *Rt3* of the CP3 to *Rd2* of the CP2 as indicated by the arrow labeled 2. Note that this pattern of scheduling high coefficients to *Rd2* repeats again in cycles 23 and 26. Thus, since *Rd2* accept data either from *Rt*1 of the CP1 or *Rt3* of the CP3, the multiplexer labeled *muxc2* is added in Figure 4.2.5 to allow *Rd2* to select between these two inputs. Similarly, the inclusion of the multiplexers, labeled *muxc1* and *muxc3* attached to *Rd*1 of the CP1 and *Rd3* of the CP3, respectively, can be verified.

Another point that needs to be addressed is that Figure 4.2.5 shows that the operations of *muxc1* and *muxc3* can be controlled by only one signal labeled *sc*1. This can be verified also with the aid of Table 4.3. For instance, the two arrows labeled 3 and 5 in Table 4.3 indicate that two data transfers take place at the same time; one is going to *Rd*1 of the CP1 and the other to *Rd2* of the CP3. This implies that the two data transfers can be accomplished if the data pointed by arrow 3 and that pointed by arrow 5 are connected to input 0 of *muxc1* and *muxc3*, respectively. On the other hand, the second data transfer indicated by the two arrows labeled 4 and 6 can be accomplished by connecting the data pointed by arrow 4 and that pointed by arrow 6 to input 1 of the multiplexers *muxc1* and *muxc3*, respectively. Furthermore, Table 4.3

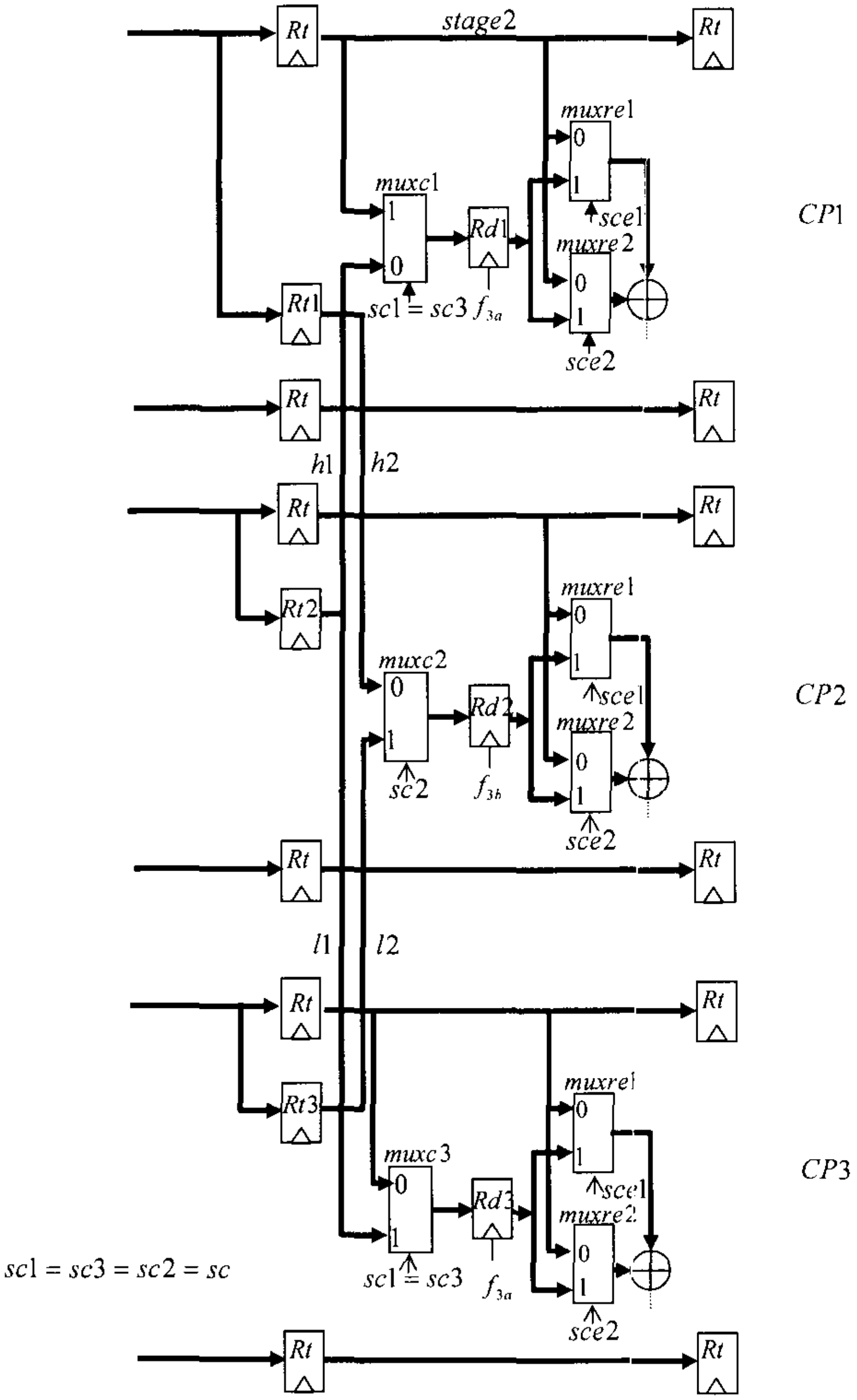
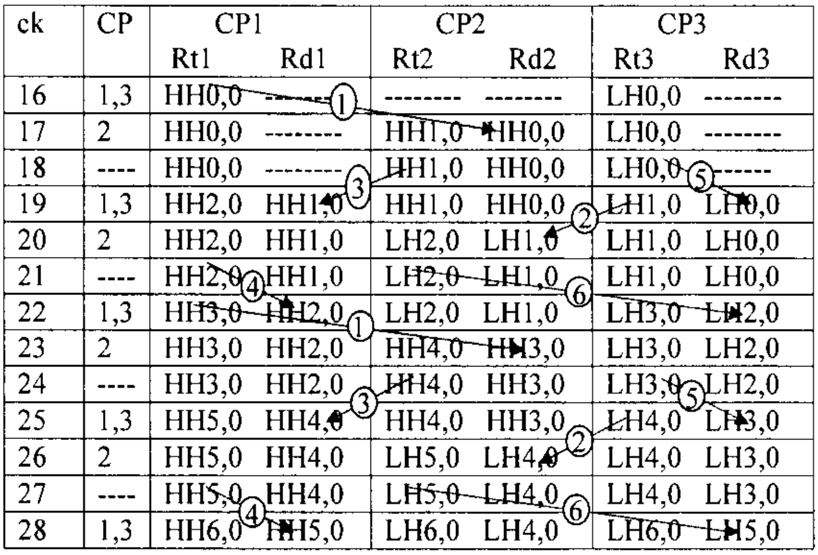


Figure 4.2.5 Modified CPs datapath architecture

Table 4.3 shows how and when CPs exchange high coefficients



ck	CP	CP1 Rt1	CP1 Rd1	CP2 Rt2	CP2 Rd2	CP3 Rt3	CP3 Rd3
16	1,3	HH0,0	-------- ①	--------	--------	LH0,0	--------
17	2	HH0,0	--------	HH1,0	HH0,0	LH0,0	--------
18	----	HH0,0	-------- ③	HH1,0	HH0,0	LH0,0 ⑤	-----
19	1,3	HH2,0	HH1,0	HH1,0	HH0,0 ②	LH1,0	LH0,0
20	2	HH2,0	HH1,0	LH2,0	LH1,0	LH1,0	LH0,0
21	----	HH2,0 ④	HH1,0	LH2,0	LH1,0 ⑥	LH1,0	LH0,0
22	1,3	HH3,0	HH2,0 ①	LH2,0	LH1,0	LH3,0	LH2,0
23	2	HH3,0	HH2,0	HH4,0	HH3,0	LH3,0	LH2,0
24	----	HH3,0	HH2,0 ③	HH4,0	HH3,0	LH3,0 ⑤	LH2,0
25	1,3	HH5,0	HH4,0	HH4,0	HH3,0 ②	LH4,0	LH3,0
26	2	HH5,0	HH4,0	LH5,0	LH4,0	LH4,0	LH3,0
27	----	HH5,0 ④	HH4,0	LH5,0	LH4,0 ⑥	LH4,0	LH3,0
28	1,3	HH6,0	HH5,0	LH6,0	LH4,0	LH6,0	LH5,0

can be used for deriving control signal values for signals *sc1=sc3* and *sc2* as shown in Table 4.4. These signal values repeat every 6 clock cycles. As indicated in the table these two signals can be further combined into one signal *sc*.

A careful examination of 9/7 DDGs shows that stage 3 of the 9/7 CPs in the 3-parallel architecture should be also modified as shown in Figure 4.2.6. This figure can be verified using 9/7 DDGs. The operations of the 3 multiplexers, labeled *mux* in Figure 4.2.6 can be controlled simply by setting the control signal *s* repeatedly 3 consecutive cycles low and 3 cycles high as soon as stage 3 latches of the CP2 are loaded, as shown in Table 4.4 for signal sc. Figures 4.2.5 and 4.2.6 form the first 3 stages of the 6-stage 9/7 CP and the remaining two stages are identical to stages 1 to 2 and the last stage is the scale factor.

Table 4.4 Control signal values for signal sc

Cycle number	*sc1=sc3*	*sc2*	*Sc*
16	X	0	0
17	X	X	0
18	0	X	0
19	X	1	1
20	X	X	1
21	1	X	1

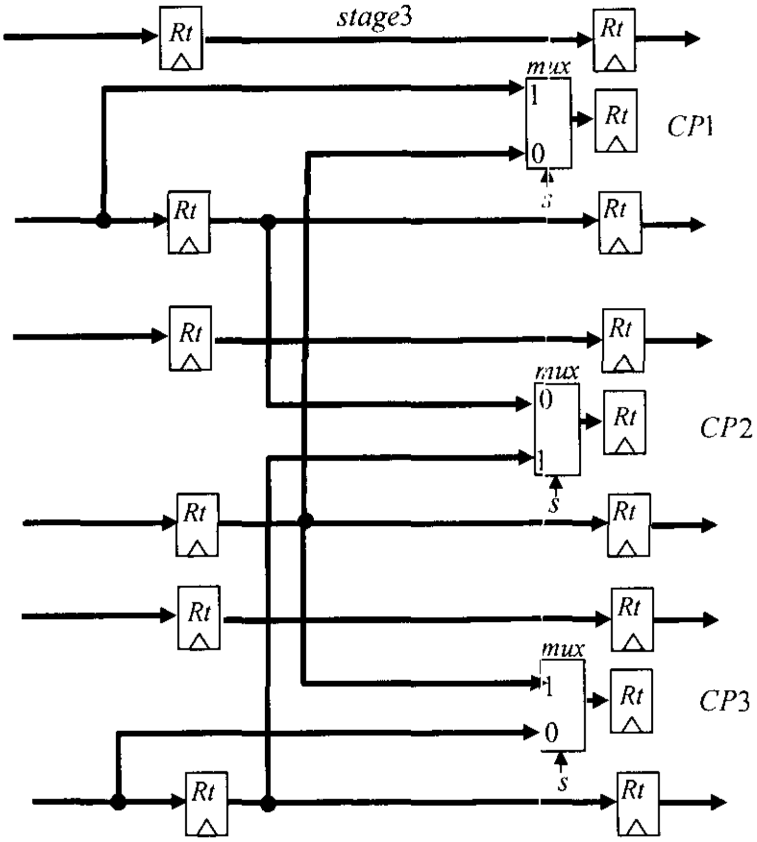


Figure 4.2.6 modified stage 3 of the 9/7 CPs

4.2.3 4-parrallel pipelined architecture

The 4-parallel pipelined architecture is shown in Figure 4.2.7 and its dataflow is given in Table B.10. This architecture closely resembles the 2-parallel architecture shown in Figure 4.2.1. The main difference is that the 2-parallel architecture consists of two pipelined processors, whereas the 4-parallel consist of 4 pipelined processors. Each pipelined processor contains one RP and one CP.

The architecture scans the external memory with frequency f_4 and itself operates with frequency. The clock frequency f_4 can be obtained from *statement3* as

$$f_4 = 4k/t_p \qquad (4.1)$$

Note that when degree of parallelism increases from 2 to 3 e.g., the scanning frequency f_l also increases, while the architecture frequency of operation, which is the

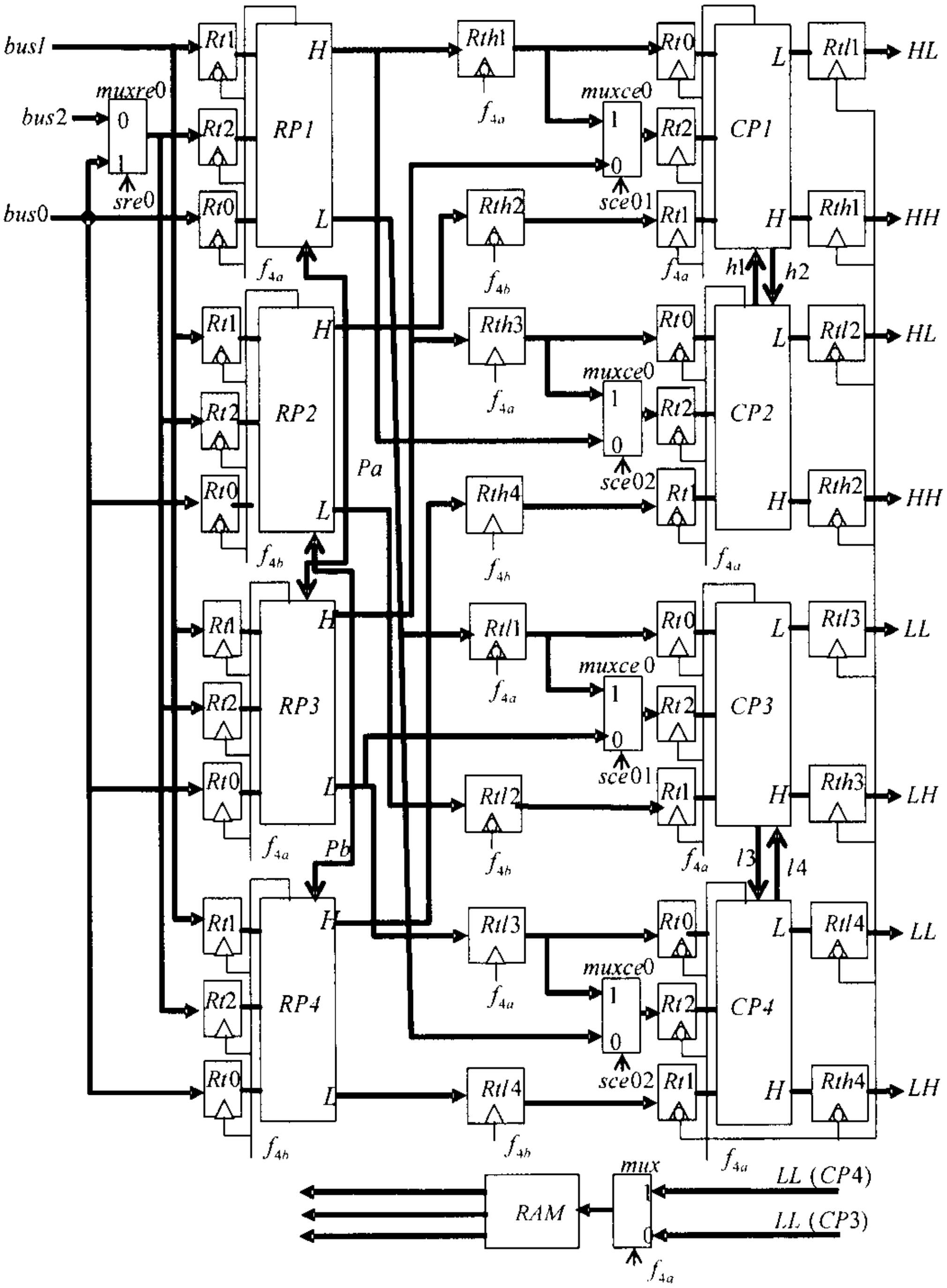


Figure 4.2.7 4-parallel pipelined architecture

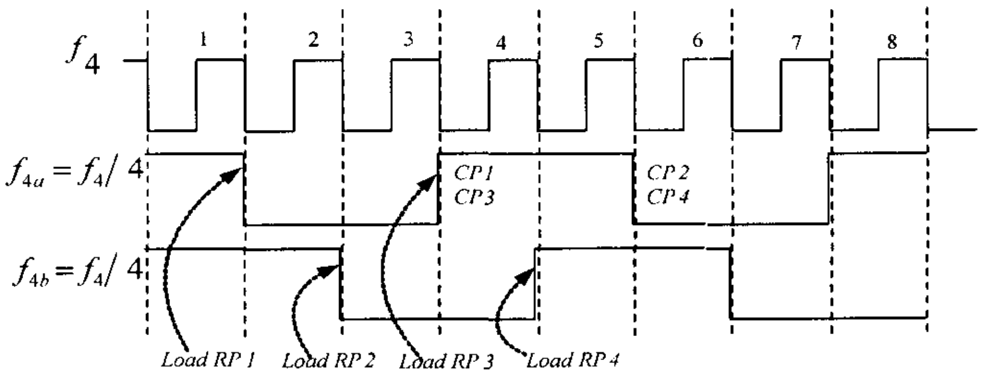


Figure 4.2.8 Waveforms of the 3 clocks used in 4-parallel

reciprocal of the stage critical path delay of the pipelined processors, remains unchanged.

Two waveforms of the frequency labeled f_{4a} and f_{4b} that can be generated from f_4 are shown in Figure 4.2.8. In the architecture, RP1 and RP3, and their associate latches employ the clock labeled f_{4a}, whereas RP2 and RP4 and their associate latches employ the clock labeled f_{4b} as shown in Figure 4.2.7.

As shown in Table B.10, in every clock cycle, three pixels are scanned from external frame memory and are loaded into the latches of one of the RPs. First, RP1 latches are loaded followed by RP2 latches then RP3 latches followed by RP4 latches, and then the process repeats. When the scanning process return to RP1 to initiate another operation, the RP1 should have completed its current operation in the time specified by t_p/k, and should be ready to accept the pixels of the next operation. As indicated, in the architecture, RP1 latches will be loaded with new data every time clock f_{4a} makes a negative transition, while RP3 latches will be loaded at the positive transition. Whereas, RP2 and RP4 latches will be loaded at the negative and the positive transitions of clock f_{4b}, respectively.

In the 3-parallel architecture, the strategy adopted was to allow each run to begin at RP1. This strategy was preferred over the one that allows each new run to start its computations in the RP that immediately comes after the RP where the previous run end, mainly because with the later it is very difficult to come up with a simple scheme that allows us to decide which TLB a high coefficient needed in the next run should be stored and when it can be retrieved. However, the situation is quit different in the

4-parallel architecture because there can be found a simple and very efficient scheme that encourages the adoption of the later strategy.

The scheme, which can be reasoned from Table B.10, is summarized as follows. The decision, where to store each high coefficient calculated in the previous run that are needed in the calculations of low coefficients in the next run, can be made by examining the two least significant bits of N . Case one; if the two least significant bits of N are 00 or 11 then the high coefficients should be stored in the TLBs of the RPs that generate them. Case two; if the two least significant bits of N are either 01 or 10, then the high coefficients of RP1 should be stored in the TLB of RP3 and vice versa, and the high coefficients of RP2 should be stored in the TLB of RP4 and vice versa. Symbolically, case two can be written as

$$\begin{aligned} RP1 &\xleftrightarrow{Pa} RP3 \\ RP2 &\xleftrightarrow{Pb} RP4 \end{aligned} \tag{4.2}$$

Therefore, the paths labeled *Pa* and *Pb* are added in Figure 4.2.7.

Not that the following fact is used also to arrive at the above result. In the clock cycle where a transition from a run to the next occurs, especially when the column length (N) of an image is odd, the external memory is not scanned and no pixels are loaded into the RP latches. Since, during this cycle two CPs (CP1 and CP3) or (CP2 and CP4) each will compute the last low coefficient using the last high and the last low coefficients in H and L columns, respectively, as required by DDGs for odd length signals. In Table B.10, the columns labeled Rth and Rtl represent H and L columns, respectively.

The above scheme only affects stage 2 of the four 5/3 RPs and stages 2, 3, and 5 of the four 9/7 RPs and it can be implemented as shown in Figure 4.2.9. Signal *(zs)* which control the operations of the four multiplexers labeled *mux1, mux2, mux3,* and *mux4* can be generated by use of a simple 2-input XNOR gate with its two inputs connected to the two least significant bits of N. Thus, if the input to the XNOR are either00 or 11 (case one), *zs* is asserted high to pass the high coefficient generated in stage 1 of the same RP. Otherwise (case two) it is asserted low to pass the high coefficient stored in each register *BIR* (buffer input register) that have been generated

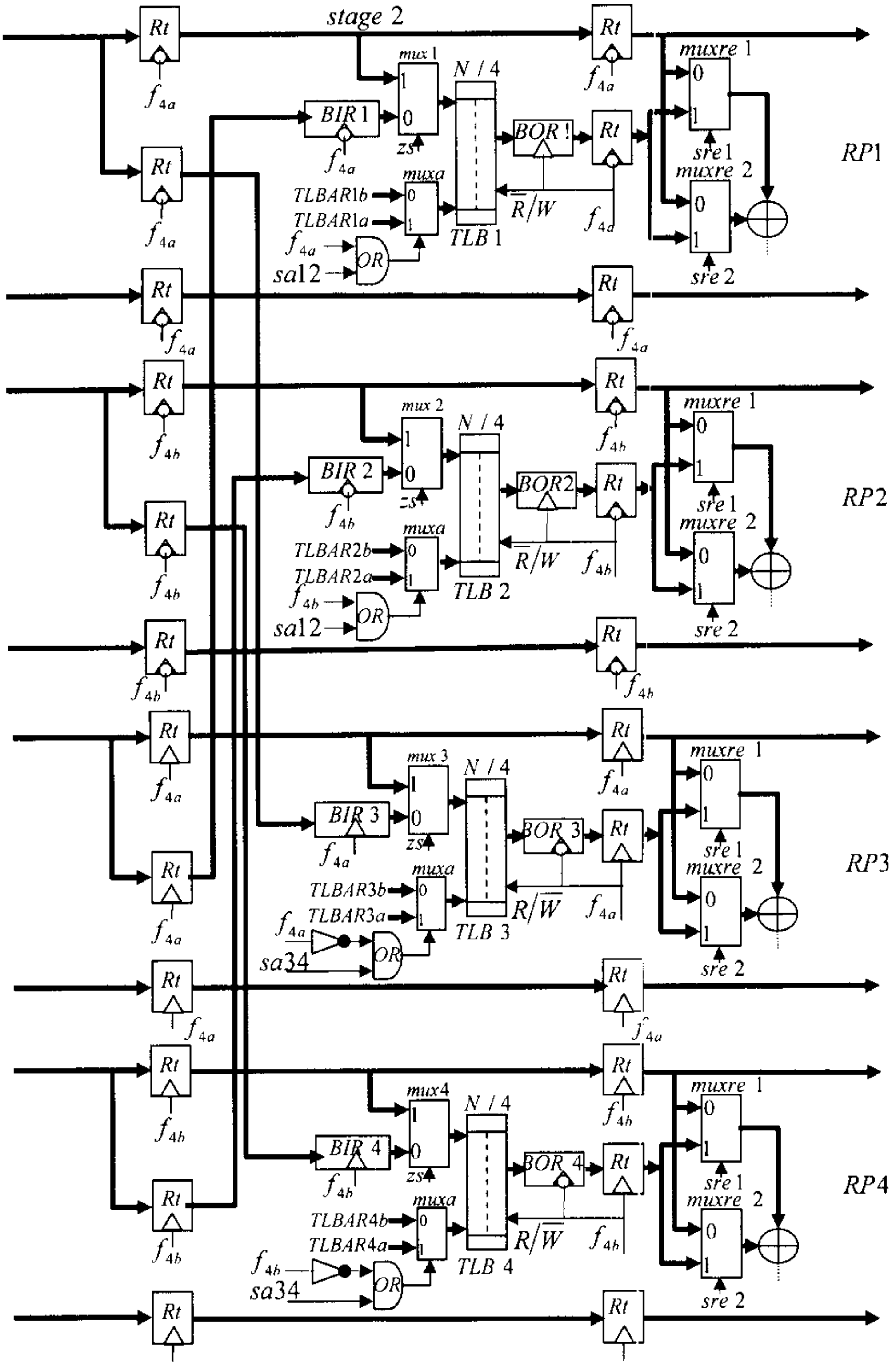


Figure 4.2.9 Modified stage 2 of the RPs datapath architecture

by one of the RP. Note that signal *zs* will only have one value during each level of decomposition. For example, during the whole period of the first level decomposition, *zs* may be equal to 1 or 0, but not both.

This scheme, even though it optimizes the performance in term of number of clock cycles that are needed for j-level decomposition, but, it complicates very much the operations of the 4 RPs which would require a very complex control circuitry. In addition, it needs more hardware and long buses. The alternative scheme would be to allow each run to begin at RP1, as in case 1. The advantage of this scheme is that it would reduce the hardware and the control complexities to the level of case 1 which is less complex and manageable. In addition, it will eliminate the long buses, the four BIR registers, and the four multiplexers labeled *mux1, mux2, mux3,* and *mux4*. The disadvantage of the alternative scheme is that it will increase the execution time by $M/2^{j-1}$ cycles for each decomposition level, when case2 occurs. However, since, the hardware complexity is less; the alternative scheme will operate with higher frequency which would compensate for the performance lost.

Read and write operations in the 4 TLBs for case2 is somewhat complex. Therefore, Table B.11 is provided to illustrate how read and write operations take place in the TLBs during each run of case2. Table B.11 shows read and write operations for RP1 and RP3, which is also identical to that, take place in TLBs of RP2 and RP4, respectively. Table B.11 shows that in the first run, RP1 and RP3 each uses its *TLBARa* for addressing its TLB and in each cycle, reference to clock f_{4a}, the same location is read in the first half cycle and is written in the second half cycle starting from the first location. In the second run, as in the first run, RP1 uses only *TLBAR1*a to address its TLB, while RP3 uses both *TLBARa* and *TLBARb* to address its TLB, which take place as follows. In each cycle two successive locations are accessed. The first location is accessed by *TLBAR3a*, while the second is accessed by *TLBAR3b*. In the first half cycle, reference to clock f_{4a}, *TLBAR3b* reads its location and loads the result into register *BOR3* by the negative transition of f_{4a}, whereas, during the second half cycle, *TLBAR3a* write contents of register BIR3 into the location it addressing. This writing completes by the positive transition of clock f_{4a}. For example, Table B.11 shows that in cycles 35 and 37, *TLBAR3a* is addressing location 2, while *TLBAR3b* is addressing location 3.

In addition, note that in cycle 23, where run2 begins, and cycle 25, Table B.11 shows that *TLBAR3a* is addressing location 4 to write the last coefficient of run1, while *TLBAR3b* is addressing location 0 to read the first location which contains the first coefficient needed in run2 first operation.

Finally, note when the control signals *sa12* or *sa34* of the multiplexers, labeled *muxa* are set 0 in a run, each OR gate passes the clock signal to the multiplexer *muxa* control signal. The clock signals of f_{4a} or f_{4b} allow both *TLBARa* and *TLBARb* to be used for addressing TLBs, as shown in RP3's run2 in Table B.11. On the other hand, when, *sa12* and *sa34* are set 1 in a run only *TLBARa* is used for addressing TLBs, as shown in run1 of Table B.11. In case1, signals *sa12* and *sa34* are set 1 in all runs and only *TLBARa* of each RP is used for addressing *TLB.*

The control signals such as *zs*, *incar,* and *sre2* etc., which are generated by the control unit can be arranged as shown in Figure 4.2.10 (a) and its block diagram is shown in Figure 4.2.10 (b). The control signal values issued in each clock cycle by control unit are transferred to the first stage of the pipeline and are loaded into the control signal latches (CSTs) to carry these signal values from stage-to-stage. When a stage where a signal(s) is used is reached, the signal value carried by its CST is applied, while the remaining signals are carried on to the next stage.

Now, let's move to the CPs side to see how this part of the architecture works. The 4 CPs run by the clock labeled f_{4a}. According to the dataflow shown in Table B.10, both CP1 and CP3 execute in parallel starting from cycle 15 and load new data every time clock f_{4a} makes a positive transition. Similarly, both CP2 and CP4 execute in parallel starting from cycle 17 and load new data every time clock f_{4a} makes a negative transition. Thereafter, all RPs and CPs in the architecture work in parallel. However, both CP1 and CP2 execute high coefficients stored in *Rth1, Rth2, Rth3* and *Rth4*, whereas CP3 and CP4 execute low coefficients stored in *Rtl1, Rtl2, Rtl3,* and *Rtl4.*

The two paths labeled *h1* and *h2* between CP1 and CP2, and that labeled l_3 and l_4 between CP3 and CP4 are used for passing high coefficients among CPs, since each high coefficient generated by a CP is also required in the next operation that will be executed by another CP. Passing high coefficients occur between stages 2

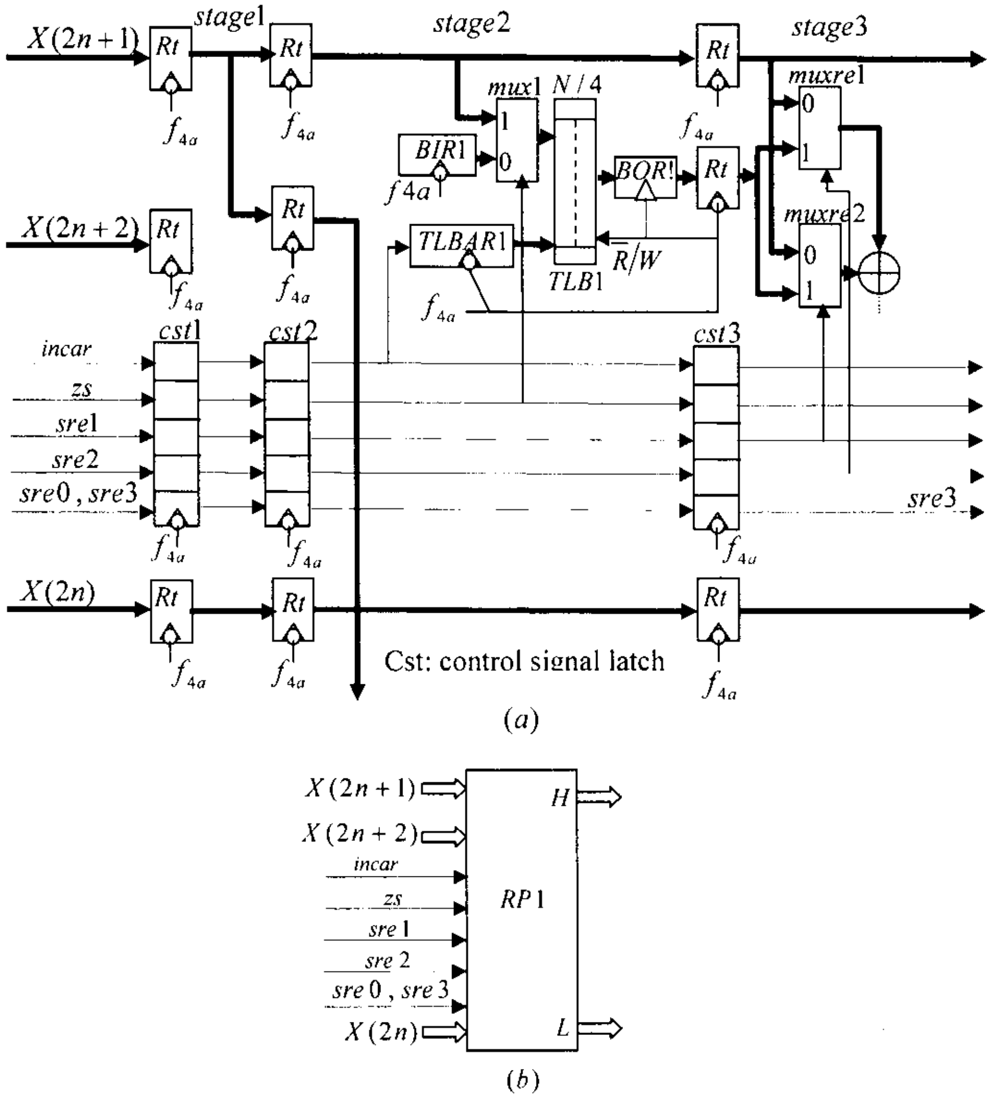


Figure 4.2.10 (a) Control signals carried by CST and (b) the block diagram

of both CP1 and CP2 or CP3 and CP4, in case of 5/3, and between stages 2 and between stages 5 of both CP1 and CP2 or CP3 and CP4, in case of 9/7, as illustrated in Fig. 4.2.11 for CP1 and CP2. The first 2 stages of Figure 4.2.11 represent modified 5/3 CP1 and CP3, while, stages 1 to 3 represent the first 3 stages of the modified 6-stage 9/7 CP1 and CP3 and the following two stage are identical to stages 1 to 2.

In a control design it would be necessary to determine the clock cycle ($C1$) where the first input data are loaded into the CPs latches and the clock cycle ($C2$) where the first output coefficients are loaded into the CPs output latches. The following two equations can be used to determine $C1$ and $C2$.

$$C1 = l \cdot k_r + 2i + 1 \tag{4.3}$$

$$C2 = C1 + l \cdot k_c \tag{4.4}$$

Where l = 2, 3, 4... denote 2-, 3-, 4-parallel (degree of parallelism) and i = 1, 2, 3 ... denotes the first, the second, the third scan method and so on. K_r and k_c are the number of pipeline stages in a RP and a CP, respectively. Note that Eqs (4.4) and (4.5) are also valid for parallel intermediate architectures developed in section 4.3.

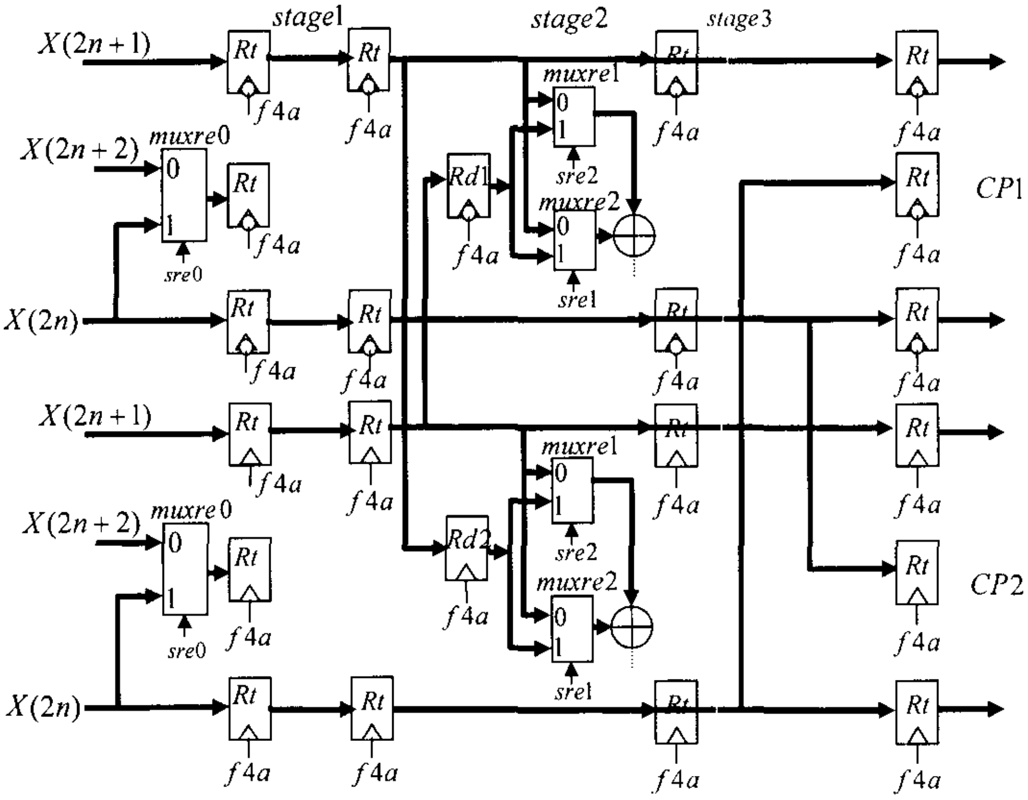


Figure 4.2.11 CP1 and CP2 are modified to exchange high coefficients

4.2.4 Evaluations of architectures

To evaluate the performances of the three parallel architectures developed in this section, in terms of speedup, efficiency, hardware utilization, and power consumption consider the following. In the single pipelined processor architecture based on the first overlapped scan method developed in chapter 3, the total time $T1$ required to execute n operations for j-level decomposition of an NxM image is given by Eq (3.14) as

$$T1 = [\rho 1 + 3(n-1)]\tau_1 \tag{4.5}$$

From *statement2*, case 2,

$$\tau_1 = t_p/(I \cdot k) \tag{4.6}$$

Where $I = 3$ for 5/3 and 9/7. Thus,

$$T1 = [\rho 1 + 3(n-1)]t_p/3k \tag{4.7}$$

On the other hand, the total time, *T2*, required for executing *n* operations for j-level decomposition of an *NxM* image on the 2-parallel pipelined architecture shown in Figure 4.2.1, can be estimated using Table B.8 as

$$T2 = \frac{[\rho 2 + 2(n-1)]\tau_2}{2} \tag{4.8}$$

From *statement3*, $\tau_2 = t_p/2k$ (4.9)

Therefore, $T2 = \frac{[\rho 2 + 2(n-1)]t_p/2k}{2}$ (4.10)

The speedup factor (*S2*) is then given by

$$S2 = \frac{T1}{T2} = \frac{[\rho 1 + 3(n-1)]t_p/3k}{[\rho 2 + 2(n-1)]t_p/4k} \tag{4.11}$$

For large *n*, the above equation reduces to

$$S2 = \frac{3(n-1)t_p/3k}{2(n-1)t_p/4k} = 2 \tag{4.12}$$

Eq (4.12) indicates that the 2-parallel architecture is 2 times faster than the single pipelined architecture.

The efficiency (E_l) of an *l*-parallel processors system is defined by [58] as

$$E_l = S_l/l \tag{4.13}$$

The efficiency measures the useful portion of the total work performed by *l* processors. The lowest efficiency corresponds to the case of an entire *NxM* image being decomposed on a single pipelined processor (consisting of a RP and CP). The maximum efficiency is achieved when all *l* pipelined processors are fully utilized throughout the execution period. Thus, the efficiency of the 2-parallel pipelined architecture can be written as

$$E_2 = S_2/2 = 1 \tag{4.14}$$

Hardware utilization indicates the extent to which resources (e.g. processors) are utilized during a parallel computation [58]. Since in parallel architectures, hardware utilization can be measured by efficiency [40], therefore, it can be concluded that hardware utilization in the 2-parallel architecture is 100%.

The total time (*T3*) required to perform *n* operations, in j-level decomposition of an *NxM* image on the 3-parallel pipelined architecture, can be written as

$$T3 = \frac{[\rho 3 + 3(n-1)]\tau_3}{3} \tag{4.15}$$

From *statement3,* $\tau_3 = t_p/3k$ (4.16)

Thus, $T3 = \frac{[\rho 3 + 3(n-1)]t_p/3k}{3}$ (4.17)

The speedup factor (*S3*) is given by

$$S3 = \frac{T1}{T3} = \frac{[\rho 1 + 3(n-1)]t_p/3k}{[\rho 3 + 3(n-1)]t_p/9k} \tag{4.18}$$

For large *n*, *S3* reduces to

$$S3 = \frac{27(n-1)}{9(n-1)} = 3 \tag{4.19}$$

The efficiency $E_3 = S_3/3 = 1$ (4.20)

Eq (4.19) indicates that the 3-parallel architecture is 3 time faster than the single pipelined architecture with efficiency 1.

Similarly from Table B.10, the total time (*T4*) require to execute *n* operations for j levels of decomposition of an *NxM* image on the 4-parallel pipelined architecture can be written as

$$T4 = \frac{[\rho 4 + 2(n-1)]\tau_4}{2} \tag{4.21}$$

From *statement3,* $\tau_4 = t_p/4k$ (4.22)

Thus, $$T4 = \frac{[\rho 4 + 2(n-1)]t_p/4k}{2} \tag{4.23}$$

The speedup factor (*S4*) is then given by

$$S4 = \frac{T1}{T4} = \frac{[\rho 1 + 3(n-1)]t_p/3k}{[\rho 4 + 2(n-1)]t_p/8k} \tag{4.24}$$

For large *n*, the above equation reduces to

$$S4 = \frac{24(n-1)}{6(n-1)} = 4 \tag{4.25}$$

The efficiency $E_4 = S_4/4 = 1$ (4.26)

Equations (4.25) and (4.26) imply that the 4-parallel architecture is 4 times faster than the single pipelined architecture and the efficiency is 1, respectively.

On the other hand, the power consumption of *l*-parallel pipelined architecture as compared with the single pipelined architecture can be obtained as follows. Let P_1 and P_l denote the power consumption of the single and *l*-parallel architectures without the external memory, and P_{m1} and P_{ml} denote the power consumption of the external memory for the single and *l*-parallel architectures, respectively. Then,

$$P_1 = C_{total} \cdot V_0^2 \cdot f_1/3 \ , \quad P_l = l \cdot C_{total} \cdot V_0^2 \cdot f_l/l \tag{4.27}$$

and $$\frac{P_l}{P_1} = \frac{l \cdot C_{total} \cdot V_0^2 \cdot f_l / l}{C_{total} \cdot V_0^2 \cdot f_1 / 3} = \frac{3f_l}{f_1} = 3\left(\frac{l \cdot k}{t_p}\right) \Big/ 3k/t_p = l\ . \tag{4.28}$$

where C_{total} is the total capacitance of single pipelined architecture.

On the other hand, P_{m1} and P_{ml} can be estimated as

$$P_{m1} = C^m_{total} \cdot V_0^2 \cdot f_1 \quad , \quad P_{ml} = I \cdot C^m_{total} \cdot V_0^2 \cdot f_l , \tag{4.29}$$

and $$\frac{P_{ml}}{P_{m1}} = \frac{I \cdot f_l}{f_1} = \frac{I \cdot l \cdot k/t_p}{3k/t_p} = l \tag{4.30}$$

where C^m_{total} is the total capacitance of the external memory and $I=3$ *is* number of buses.

From the above evaluations, it can be concluded that as the degree of parallelism increases the speedup and the power consumption of the architecture, without external memory, and the power consumption of the external memory increase by a factor of l, as compared with single pipelined architecture.

4.3 Parallel form of the intermediate architectures

As mentioned before, the rational behind developing intermediate architecture is to reduce the excess power consumption of the external memory, due to scanning overlapped areas, to somewhat between the architecture based on the first overlapped scan method and that based on the nonoverlapped scan method developed in chapter 3. In this section, the single pipelined intermediate architecture shown in Figure 3.7.2 will be extended to 2- and 3-parallel pipelined architectures to achieve speedup factors of 2 and 3, respectively. The two proposed parallel architectures are intended for used in real-time applications of 2-D DWT, where very high speed and throughput are required.

4.3.1 2-parallel pipelined intermediate architecture

Based on the DDGs for 5/3 and 9/7 filters shown in Figures 3.3.1 and 3.3.2, respectively, and the scan method shown in Fig. 3.7.1 (a), the 2-parallel pipelined intermediate architecture shown in Figure 4.3.1 is developed. The dataflow of the architecture is given in Table B.12. The architecture consists of 2 k-stage pipelined row-processors labeled RP1 and RP2 and 2 k-stage pipelined column-processors labeled CP1 and CP2. In the previous chapter, the RP and the CP for the 5/3 were pipelined into 4 and 3 stages, respectively, whereas, the RP and the CP for the 9/7 were pipelined into 8 and 6 stages, respectively.

The architecture scans the external memory with frequency f_2 and operates with frequency $f_2/2$. The buses labeled *bus*0, *bus1*, and *bus2* are used for transferring every clock cycle pixels from external memory to one of the RPs latches labeled Rt0, Rt1, and Rt2, according to the scan method in Figure 3.7.1 (a). This scan method requires that in the first clock cycle, the 3 buses should be used for scanning the first 3 pixels from the first row of the external memory, whereas in the second and

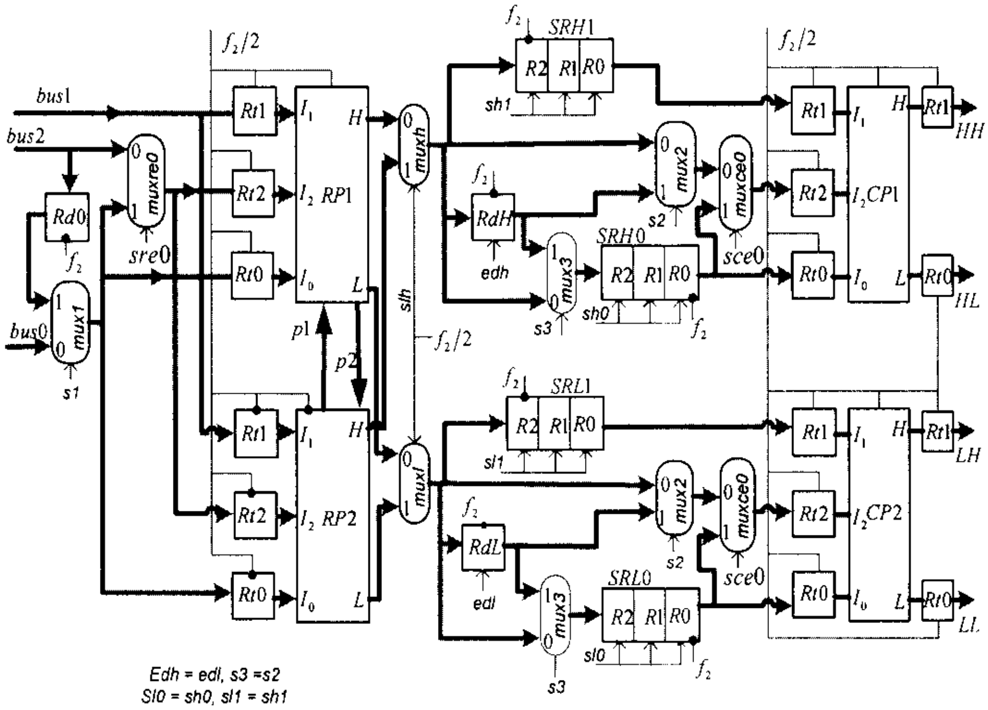


Figure 4.3.1 2-parallel pipelined intermediate architecture

third cycles each scans two pixels through *bus1* and *bus2*. Then the scan moves to the second row to repeat the process. The RP1 latches load new data (pixels) every time clock $f_2/2$ makes a positive transition, whereas RP2 latches load new data when a negative transition occurs. Assume the first half cycle of the clocks f_2 and $f_2/2$ are low. On the other hand, both CP1 and CP2 and their associate latches load new data every time clock $f_2/2$ makes a positive transition.

Furthermore, since in every clock cycle, 3 pixels are required to initiate an operation and the third pixels, according to the DDGs, is always needed in the next operation, therefore, register Rd0 is added to hold the third pixel for the next operation. The multiplexer labeled *mux1* passes *Rd0* to either *Rt0* of RP1 or *Rt0* of RP2. Register *Rd0* loads a new pixel from *bus*2 every time clock f_2 makes a negative transition.

The control signal *s*1 of the multiplexer labeled *mux1* is set to 0 in the first clock cycle of f_2 to pass data in *bus*0 and is set to 1 in the second and third clock cycles to pass *Rd0* contents. The above steps are repeated in cycles 4, 5, and 6 and so on, when scan moves to the second row.

The multiplexer labeled *muxre0* is an extension multiplexer, passes in all cases data coming through *bus2*. Except when the row length *(M)* of an image is even and only in the calculations of the last high and low coefficients in a row *r*, according to the DDGs, the pixels at location *X(r,M-2)*, which will be placed in *bus0*, must be allowed to pass through *muxre0* and then be loaded into *Rt2* as well as *Rt0*. The two multiplexers labeled *muxce0*, located at the CPs side, are also extension multiplexers and perform the same function as that of *muxre0* when DWT is applied column-wise by the CPs.

The registers labeled *SRH1*, *SRH0*, *SRL1*, and *SRL0* are FIFO shift registers each holds at any time 3 coefficients. Registers *SRH1*, *SRH0*, and *RdH* are used for storing high coefficients generated by RP1 and RP2, whereas *SRL1*, *SRL0*, and *RdL* are used for storing low coefficients. These registers all operate with frequency f_2. In addition, the control signals *sl0=sh0* and *sl1=sh1* control the operation of the FIFO registers. When they are high, the FIFOs shift in new data, otherwise, no shift take place. The high coefficients stored in *SRH0* and *SRH1* are executed by CP1, while CP2 executes low coefficients stored in *SRL0* and *SRL1*.

The operations of the two multiplexers, labeled *muxh* and *muxl*, can be controlled by one control signal labeled *slh*. This control signal is connected to the clock $f_2/2$.

When $f_2/2$ is low, both multiplexers pass coefficients generated by RP1, otherwise, pass that generated by RP2.

Observe that the dataflow pattern between cycles 13 and 18 in Table B.12, especially in the 4 FIFO registers including *RdH* and *RdL*, repeats each 6 clock cycles. A careful investigation of Table B.12 from cycles 13 to 18 shows that the control signals of the two multiplexers labeled *mux2* and two multiplexers labeled *mux3* including the control signals *(edh and edl)* of the registers labeled *RdH* and *RdL* can all be combined into one signal, *s2*. Moreover, examination of Table B.12 shows that the control signals values for signals *s2*, *sl0=sh0,* and *sl1=sh1* starting from cycles 13 to 18 can be as shown in Table 4.5. These control signal values repeat every 6 clock cycles.

Table 4.5 Control signal values for *s3*, *sl0*, and *sl1*

Cycle number	*s2*	*sl0*	*sl1*
13	0	1	1
14	1	0	0
15	1	1	0
16	0	0	1
17	1	1	1
18	0	0	1

According to the 5/3 DDGs shown in Figure 3.3.1, each coefficient calculated in the first level (step1) is also required in the calculations of two coefficients in the second level (step 2). That implies a high coefficient calculated by RP1 in stage 1 should be passed to stage 2 of RP2 and vice versa. The 9/7 DDGs shown in Figure 3.3.2 also shows similar dependencies that exist among coefficients of two levels or steps. Therefore, the path labeled *P1* and *P2* have been added in Fig. 4.3.1 so that the two RPs can pass high coefficients to each other. However, this would require the two RPs datapath architectures for 5/3 and 9/7 to be modified as shown in Figures 4.3.2 and 4.3.3, respectively.

In addition, if the third high coefficient of the first row labeled *Y(5)* in the 5/3 DDGs is stored in the first location in *TLB1* of RP1, then the third high coefficient of the second row should be stored in the first location in *TLB1* of RP2 and so on.

Similarly, the 9/7 coefficients labeled $Y''(5)$, $Y''(4)$, and $Y'(3)$ in the DDGs generated by processing the first row of the first run should be stored in the first locations of each *TLB1, TLB2*, and *TLB3* of RP1, respectively, whereas the same coefficients generated by processing the second row of the first run should be stored in the first locations of each *TLB1, TLB2*, and *TLB3* of RP2, respectively, and so on. The same process also applies in all other runs.

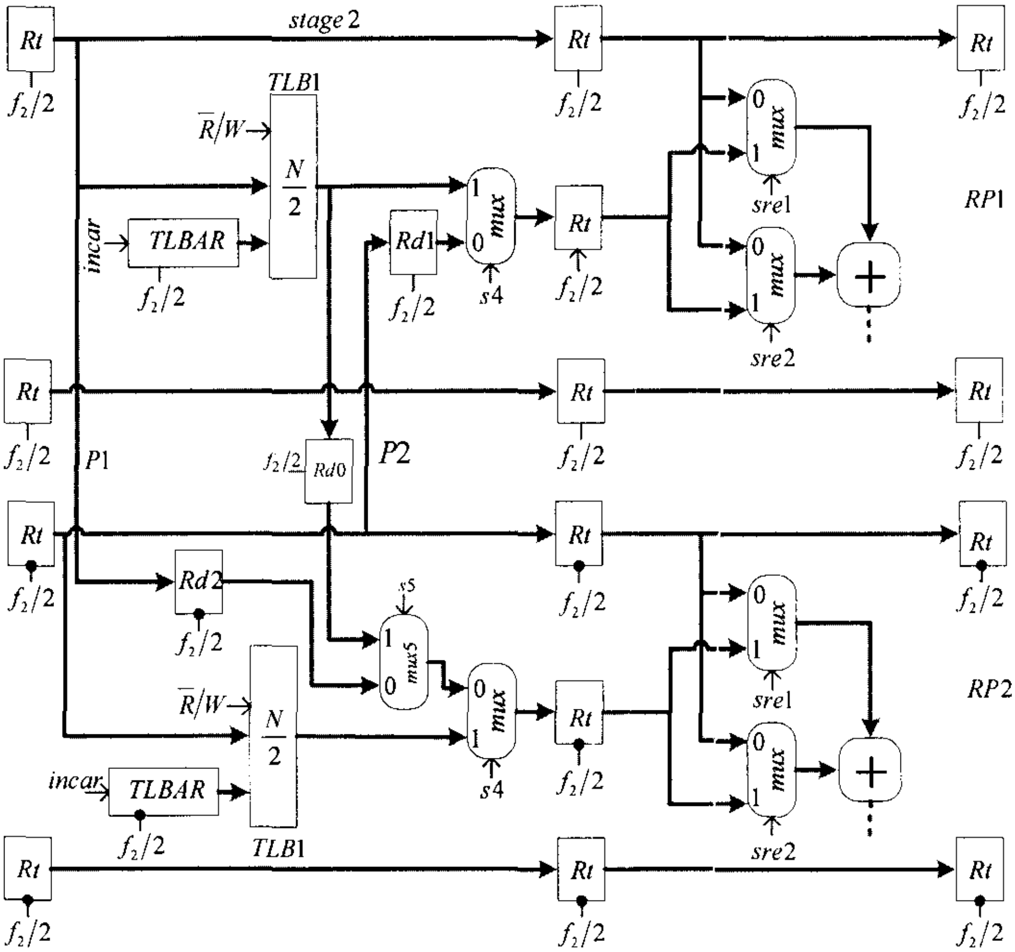


Figure 4.3.2 Modified 5/3 RPs datapath architecture

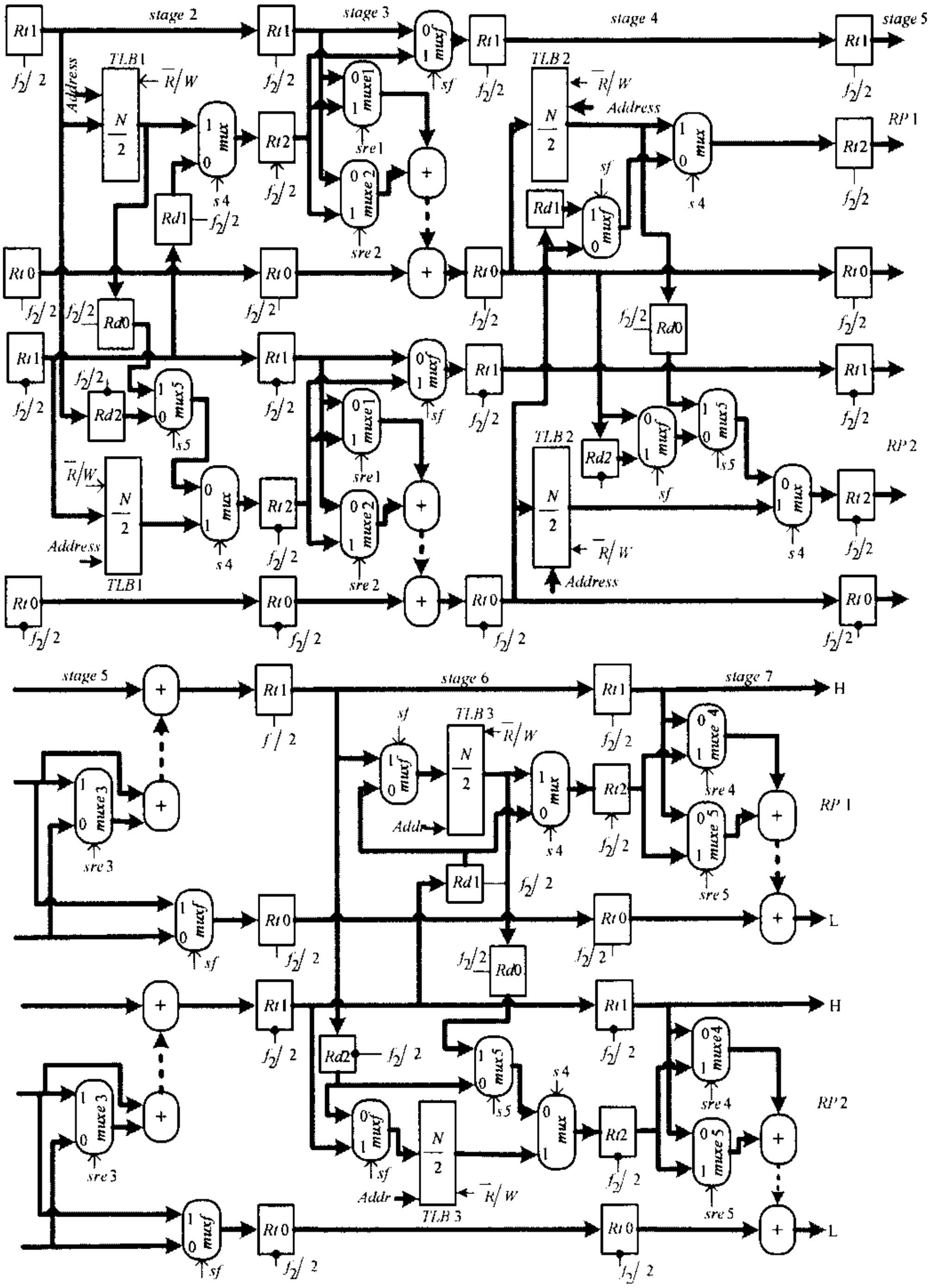


Figure 4.3.3 Modified 9/7 RPs datapath for 2-parallel intermediate architecture

The control signal *sf* of the 8 multiplexers labeled *muxf* in Figure 4.3.3 can be set 0 in the first run and 1 in all other runs. It is very important to note that, especially in the first run, the scan method in Figure 3.7.1 (a) allows 5/3 RPs to yield 6 coefficients, where half belong to the first 3 columns of H decomposition and the other half to L decomposition, each time it processes 7 pixels of a row, while 9/7 yield only 4 coefficients, 2 high and 2 low coefficients by processing the same number of pixels in a row. This implies that in the first run each 5/3 CP would process 3 columns in an interleave fashion as shown in Table B.12, whereas each 9/7 CP would process in the first run only two columns in an interleave fashion. However, in all other runs, except the last, both 9/7 and 5/3 CPs would process 3 columns at a time. This interleaving process, however, would require 9/7 and 5/3 CPs to be modified in order to allow interleaving in execution to take place.

The advantage of this organization is that the TLBs in Figures 4.3.2 and 4.3.3 are not required to be read and written in the same clock cycle, since, according to the scan method shown in Figure 3.7.1 (a), 7 pixels are scanned from each row to initiate 3 successive operations and the TLB is read in the first operation and is written in the third operation starting from the second run. Furthermore, the fact that 7 pixels are scanned from each row to initiate 3 consecutive operations and the TLB is read in the first operation and written in the third can be used to derive, for all runs except the last one, the control signal values for the signals labeled $\overline{R}/W$ and *incar* in both TLBs including *s4*, as shown in Table 4.6. These signal values repeat every 3 cycles starting from the first cycle. However, since in the first run TLBs are only written then signal s4 can be set 0 in the first run, whereas, in all subsequent runs it is set according to Table 4.6. Signals in Table 4.6 including the extension multiplexers control signals which will be generated by a separate control unit can be carried by latches, similar to pipeline latches, from the control unit to the first stage of the pipeline then to the next stage and so on. When a stage where a signal(s) will be used is reached that signal(s) can be dropped and the rest are carried on to the next stage and so on until they are all used.

Table 4.6 Control signal values for signals in stage 2 of both RP1 and RP2

Cycle Number	RP number	$\overline{R}/W$	*incar*	*s4*
1	1	0	0	1
2	2	0	0	0
3	1	1	1	0

4.3.2 Transition to the last run

The description given so far including the control signal values in Tables 4.5 and 4.6 apply to all runs except the last run, which requires special handling. The last run in any decomposition level can be determined and detected by subtracting after each run 6 from the width *(M)* of an image. The last run is reached when M becomes less than or equal to 6 $(M \leq 6)$ and M can have one of the six different values 6, 5, 4, 3, 2, or 1, which imply 6 different cases. These values give number of external memory columns that will be considered for scanning in the last run.

According to the scan method, in each run 7 columns in the external memory are considered for scanning and each 7 pixels scanned, one from each column, initiate 3 consecutive operations. Thus, since cases 6 and 5 initiate 3 operations they can be handled as normal runs.

On the other hand, cases 4 and 3 initiate 2 operations and the dataflow in the last run will differ from the normal dataflow given in Table B.12. Therefore, 2 dataflow are provided in Tables B.13 and B.14 for even and odd N, respectively, so that they can be applied when either of the two cases occurs. The dataflow shown in Table B.13 is derived for case 4 but it can be used also for case3. Similarly, Table B.14 is derived for case3 but it can be used also for case 4. Moreover, examination of Tables B.13 and B.14, especially signals $s2$, $sl0$ and $sl1$, show that after $2k+2$ cycles from the last empty cycle, where k is the number of pipeline stages of the RPs, the control signal values of signals $s2$, $sl0$, and $sl1$, which repeat every 4 clock cycles, should be as shown in Table 4.7 for the rest of the decomposition level. However, during the $2k+2$ and the empty cycles, the control signal values for $s2$, $sl0$ and $sl1$ follow Table 4.5. Therefore, cases 4 and 3 can be considered as one case. Only at the beginning of the transition to the last run, if N is even, then one empty cycle is inserted, otherwise, 4 cycles are inserted, according to Table B.13 and B.14, respectively. During an Empty cycle external memory is not scanned.

On the other hand, cases 2 and 1, each initiate one operation. Case 2 initiates an operation each time 2 pixels, one from each column, are scanned, whereas case 1 initiate an operation each time a pixel is scanned from the last column. Therefore, dataflow of the last run in the two cases will differ from the normal dataflow given in

Table 4.7 Control signal values for s2, sl0, and sl1 in the last run.

Cycle number	s2	*Sl0*	*sl1*
34	0	0	1
35	1	1	1
36	1	1	1
37	1	1	0

Table B.12. For this reason, two dataflow are given in Tables B.15 and B.16 for even and odd *N*, respectively, in order to be used when either of the two cases occurs. The dataflow in Table B.15 is derived for case 2, even *N*, but it can be also applied in case 1 for even *N* as well. Similarly, Table B.16 is derived for case 1, odd *N*, but it can be applied in case 2 for odd *N*. Furthermore, study of Tables B.15 and B.16 shows that in the last run the control signal values for *s2, sl0* and *sl1* follow Table 4.5 until the clock cycle that is *2k+1* cycles away from the last empty cycle is reached. In that clock cycle, change the control signal value of signal *sl0* to zero instead of one. Then, for all subsequence cycles and to the end of the decomposition level, the control signal values for signals *sl0, sl1,* and *s2=s3* should remain at one and *edl=edh* should alternate between 0 and 1. Therefore, cases 2 and 1 can be treated as one case. Only at the beginning of the transition to the last run, even *N* requires insertion of two empty cycle and odd *N* requires insertion of five cycles, according to Tables B.15 and B.16, respectively.

Figure 4.3.4 shows the block diagram of the control unit that generates signals *s2, sl0,* and *sl1* along with the circuits that detect the occurrence of the last run and the 6 cases. First, *M* is loaded into register *RM*, then register *R6*, which contain the 2's complement of 6, is subtracted from *RM* through the 2's complement adder circuit and the result of the subtraction is loaded back into *RM*. If *Lr* is 1, then that implies the last run is reached and the result of the subtraction is not transferred to *RM*. The 3 least significant bits of register *RM* is then examined by the control unit to determine which of the 6 cases has occurred. First *z1* is examined. If *z1* is 1, that implies the occurrence of either cases 6 or 5 and the control unit proceeds as usual. But, if *z1* is 0, then *z2* is examined. If *z2* is 1, then cases 4 and 3 are applied, otherwise cases 2 and 1.

The above description can be generalized for determining the last run in any scan method (first, second, or third scan method and so on) used in designing single or l-parallel architectures. Thus, in general, the last run in any scan method can be determined and detected by subtracting after each run $2i$ from the width (M) of an image. The last run is reached when M becomes less than or equal to $2i$ $(M \leq 2i)$, where $i=1, 2, 3...$ denote the first, the second, and the third scan method and so on. M can have one of $2i$ different values, when last is reached, as follows: $2i$, $2i-1$, $2i-2$ ... 2, 1, which implies $2i$ cases.

These values give number of external memory columns that would be considered for scanning in the last run. In addition, cases $2i$ and $2i-1$ can always be handled as normal runs.

According to the 5/3 DDGs, each 5/3 CP should also interleave in execution 3 columns, if case 5 or case 6 is the last run. But, if case 3 or case 4 is the last run, according to Tables B.13 and B.14, each CP should process 2 columns in interleave fashion, whereas, if case 1 or case 2 is the last run, according to Tables B.15 and B.16, each CP should process one column. On the other hand, each 9/7 CP, according to the DDGs, should also interleave in execution 3 columns, if either (cases 3 and 4) or (cases 5 and 6) is the last run. However, if case 1 or case 2 is the last run, then each CP should interleave 2 columns in execution, as shown in Tables B.13 and B.14.

Furthermore, a careful l examination of the 9/7 DDGs, when last run is case 5 or

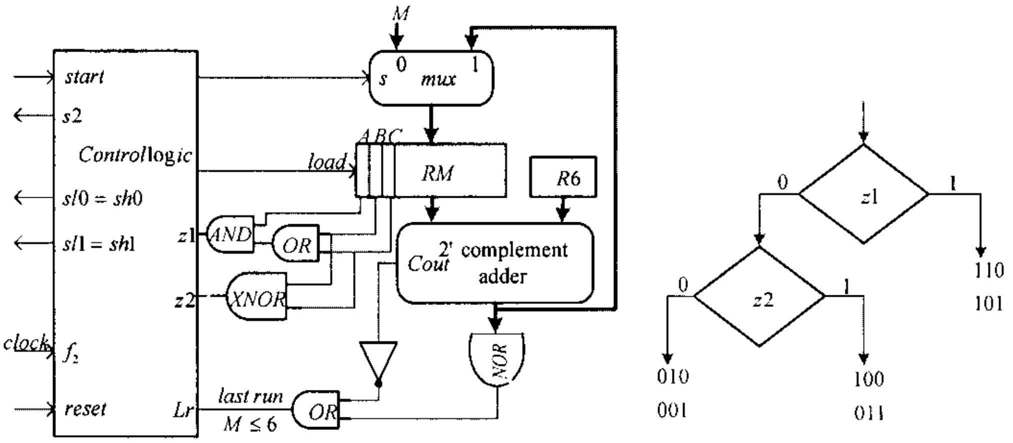


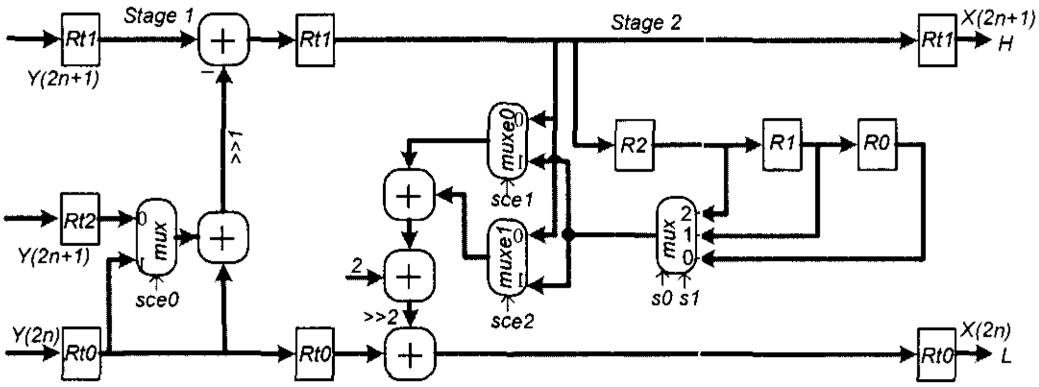
Figure 4.3.4 Control circuit that determines the last run

6, shows that the 2-parallel RPs would not be able to yield all required output coefficients. Thus, to get the remaining coefficients the 4 RPs should be instructed to execute one extra run. In the extra run, each CP would only process one column, as shown in Tables B.15 and B.16. Signal s5 of the multiplexers labeled *mux5* in Figs 4.3.2 and 4.3.3 should be set 1 only in the computations involving cases 3 and 4 of the 5/3 and cases 1 and 2 of the 9/7, otherwise, it remains at 0.

To enable each CP to process single column and interleave in execution 3 and 2 columns, each of the 5/3 and 9/7 processor's datapath should be modified as shown in Figures 4.3.5 (a) and (b), respectively. Through the multiplexers labeled *mux* the CP control the process of executing single column, interleaving 2 or 3 columns.

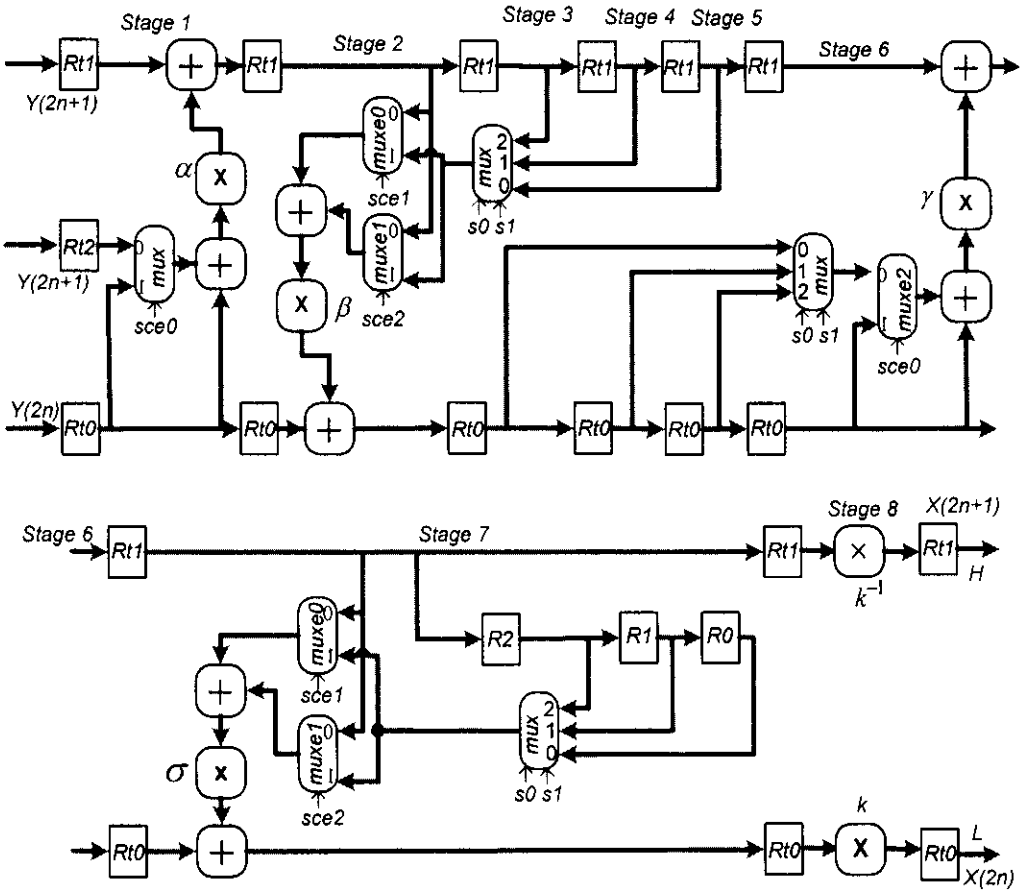
4.3.3 3-parallel pipelined intermediate architecture

The 2-parallel pipelined intermediate architecture developed in section 4.3.1 can be extended to 3-parallel pipelined intermediate architecture as shown in Figure 4.3.6. This architecture increases the speed up by a factor of 3 as compared with single pipelined architecture. The architecture performs its computations according to the dataflow given in Table B.17. It operates with frequency $f_3/3$ and scans the external memory with



s0	s1	
0	0	interleave 3 columns (run1 to the run before last + last run of cases 5 & 6)
0	1	interleave 2 columns (if last run is cases 3 or 4)
1	x	single column (if last is cases 1 or 2)

Figure 4.3.4 (a) Modified 5/3 CP for 2-parallel intermediate architecture



s0	s1	
0	0	interleave 3 columns (run1 to the run before last + last run of cases 3 & 4 and cases 5 & 6)
0	1	interleave 2 columns (if last run is cases 1 or 2)
1	x	single column (Extra run for cases 6 & 5)

Figure 4.3.5 (b) Modified 9/7 CP for 2-parallel intermediate architecture

frequency f_3. The clock frequency f_3 can be obtained from *statement3* as

$$f_3 = 3k/t_p \tag{4.31}$$

The waveform of the frequency f_3 including two waveforms of the frequency f_3 /3 labeled f_{3a} and f_{3b} that can be generated from f_3 are shown in Figure 4.3.7.

The RP2 loads new data into its latches every time clock f_{3b} makes a positive transition, whereas RP1 and RP3 load when clock f_{3a} makes a positive and a negative transition, respectively. On the other hand, CP1 and CP3 loads simultaneously new data every time clock f_{3a} makes a positive transition and CP2 loads every time clock f_{3b} makes

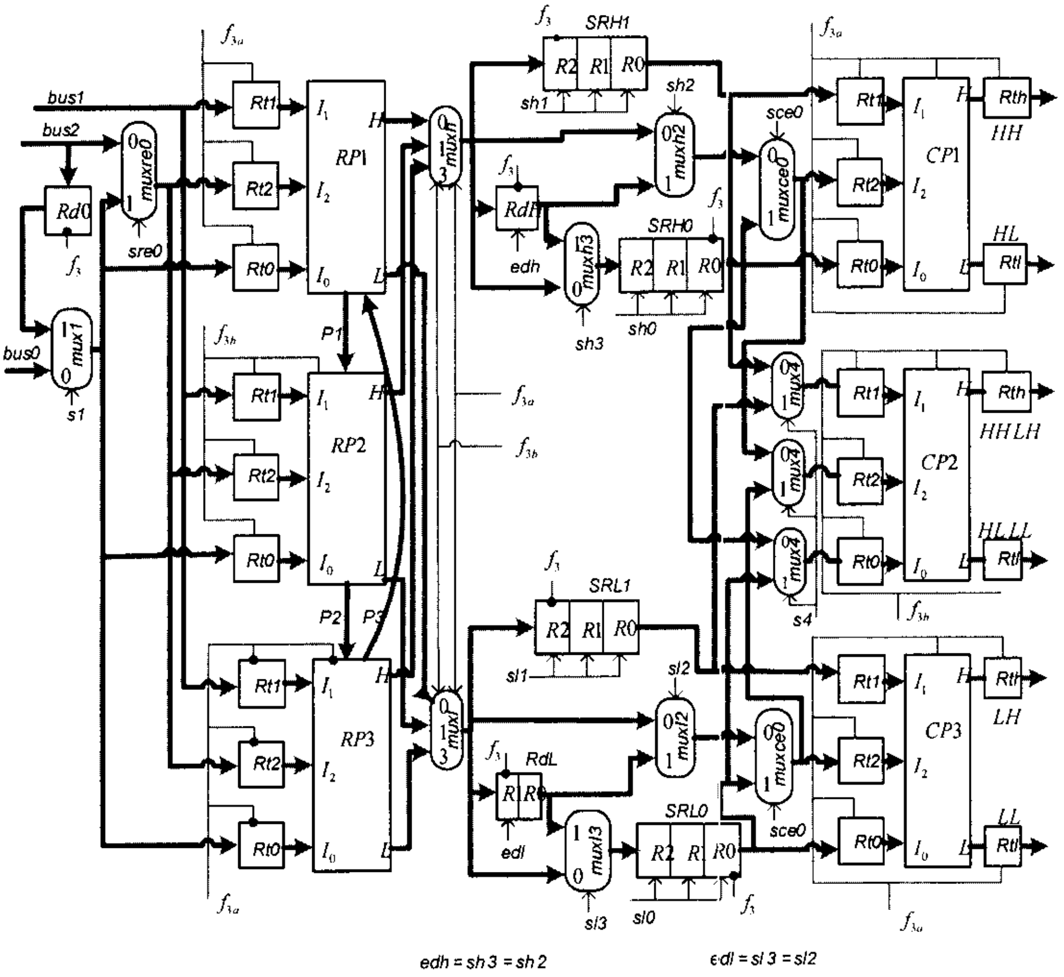


Figure 4.3.6 3-parallel pipelined intermediate architecture

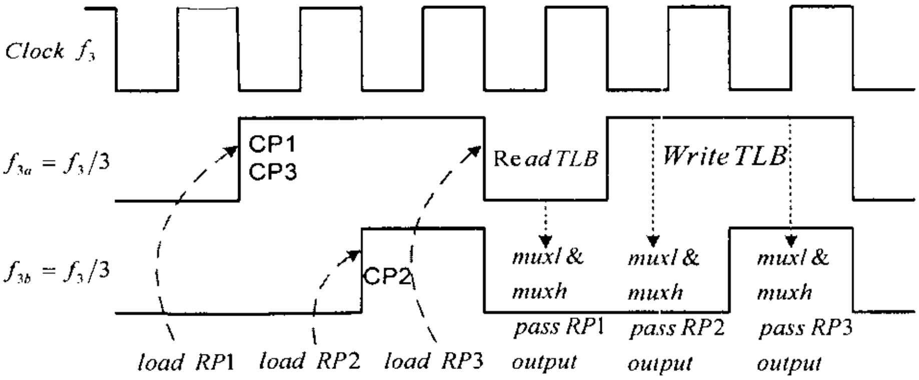


Figure 4.3.7 waveforms of the three clocks

a positive transition Furthermore, for the architecture to operate properly, it is essential the three clocks labeled f_3, f_{3a}, and f_3 be synchronized as shown in Figure 4.3.7. Clock f_{3a} and f_{3b} can be generated from f_3 using a 2-bit register clocked by f_3 and with a synchronous control signal *clear*. In order to obtain the divide-by-3 frequency, the register should be designed to count from 0 to 2 and then repeats. The synchronization can then be achieved by the control unit simply by asserting the clear signal high just before the first cycle where the external memory scanning begins.

The buses labeled *bus0, bus1*, and *bus2* are used for transferring, in every clock cycle, 3 pixels from external memory to one of the RPs latches labeled Rt0, Rt1, and Rt2. In the first clock cycle, 3 pixels are scanned from external memory, locations *X(0,0), X(0,1)* and *X(0,2)*, and are loaded into RP1 latches to initiate the first operation. While the third pixel *(X(0,2))* in *bus2,* which is required in the next operation, is also loaded into Rd0. The second clock cycle scans 2 pixels from external memory, locations *X(0,3)* and *X(0,4)*, through *bus1* and *bus2* , respectively, and loads them into RP2 latches along with the pixel in register Rd0 by the pulse ending the cycle. This cycle also stores pixel carried by *bus2* in register Rd0. Similarly, the third clock cycle transfers 2 pixels from external memory, locations *X(0,5)* and *X(0,6)*, including the pixel in register Rd0 to RP3 latches to initiate the third operation. The scan then moves to the second row

The paths labeled *P1, P2*, and *P3* in Figure 4.3.6 are used for passing coefficients between the three RPs, since a coefficient calculated in one stage of a RP is always required in the next stage of another RP. This will require the combined three RPs datapath architectures for 5/3 and 9/7 to be modified as shown in Figure 4.3.8 (a) and (a, b), respectively, so that they can fit into RPs of the 3-parallel architecture shown in Figure 4.3.6. Note that Figures 4.3.8 (a) and (b) together form the 9/7 RPs datapath architecture. This architecture can be verified using the 9/7 DDGs. The control signal *sf* of the 9 multiplexers, labeled *muxf* in Figure 4.3.8 is set 1 in the first run and 0 in all other runs.

In the 5/3 datapath architecture shown in Figure 4.3.8 (a), all high coefficients, calculated in stage 1 of the RP3 in a run, are stored in TLB of stage 2 so that they can

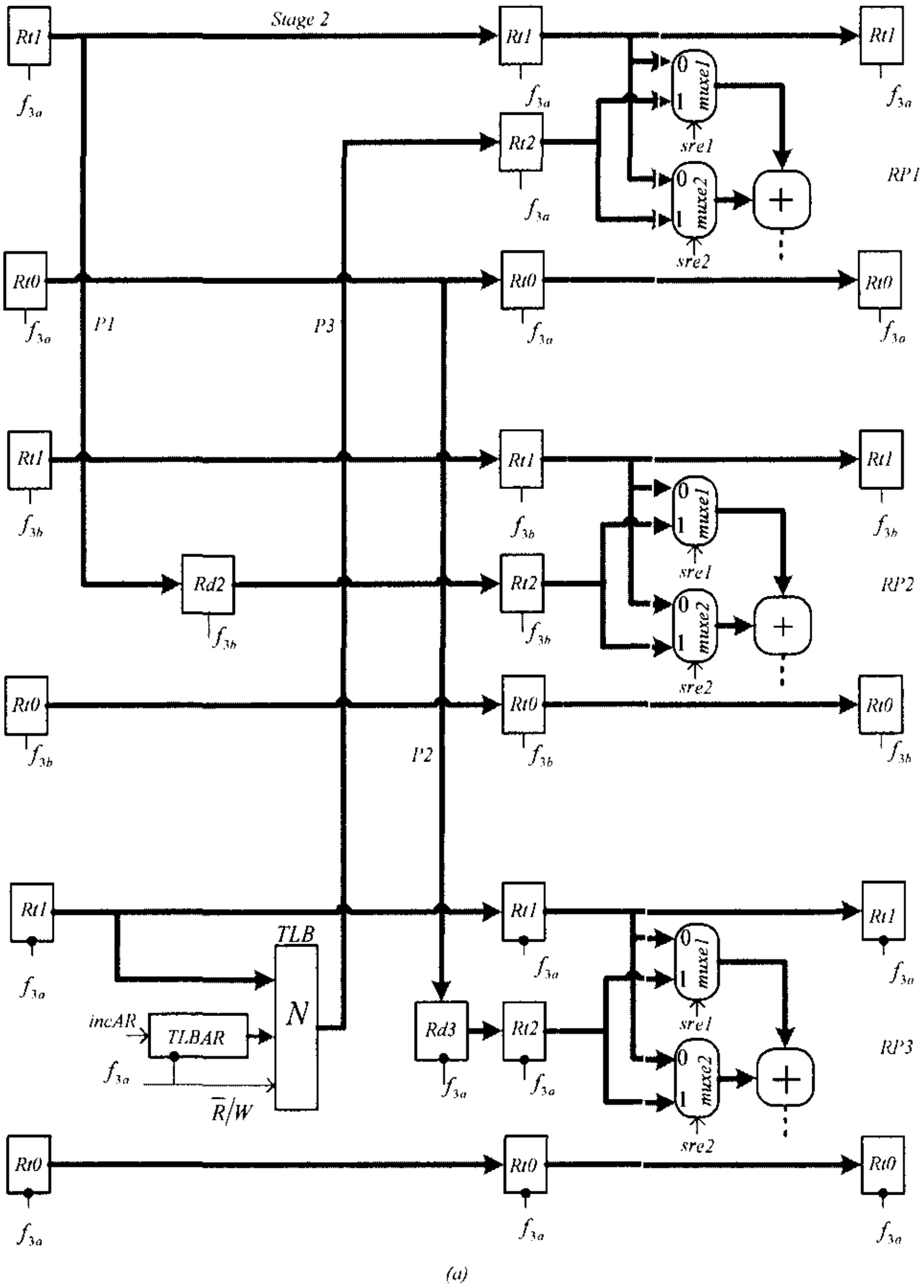
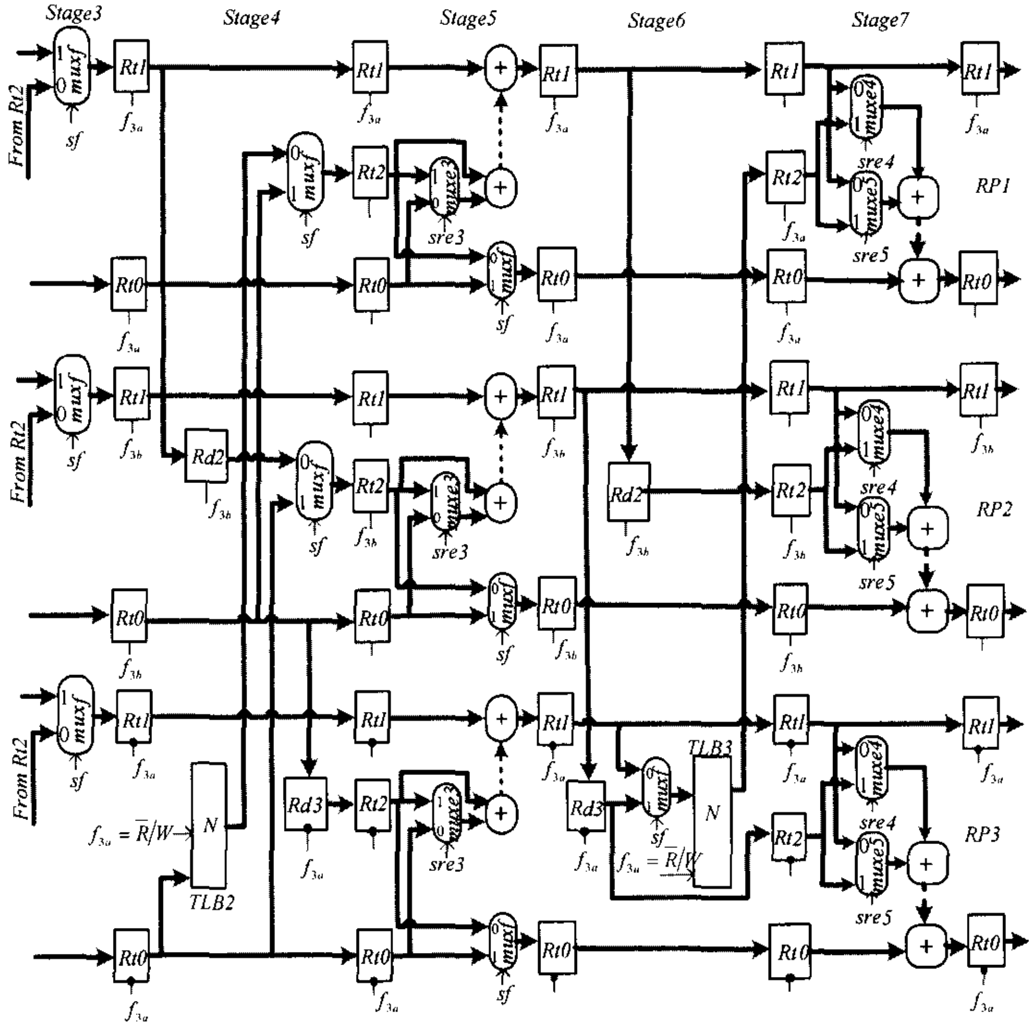


Figure 4.3.8 (a) Modified 5/3 RPs datapath architecture



(b)

Figure 4.3.8 (a,b) Modified 9/7 RPs datapath for 3-parallel intermediate architecture

be used by RP1 in the calculations of low coefficients in the next run. On the other hand, the 9/7 datapath stores, the coefficients labeled $Y''(5)$, $Y''(4)$, and $Y'(3)$ in the DDGs that can be generated as a result of processing the first 7 pixels of every row in the first run, in TLB1, TLB2, and TLB3, respectively. Similarly, all other runs can be handled.

For the same reason mentioned in the 2-parallel, the 5/3 RPs will generate 6 coefficients each time they process 7 pixels of a row, while 9/7 RPs will generate 4 coefficients by processing the same number of pixels in the first run. Each 4 coefficients will be generated by RP1 and RP2, while RP3 will generate invalid coefficients during the first run. As shown in Table B.17, each CP in the 3-parallel architecture processes, in a run, 2 columns coefficients in an interleave fashion. This interleave processing will also require each CP to be modified as shown in Figures 3.8.3 and 3.8.4 (a) for 5/3 and 9/7, respectively.

In the first run the TLB is only written. However, starting from the second run until the run before last, the TLB is read and written in the same clock cycle, with respect to clock f_{3a}.

The negative transition of clock f_{3a} always brought a new high coefficient from stage 1 into stage 2 of the RP3. During the low pulse of clock f_{3a} the TLB is read and the result, which is placed in the path labeled *P3*, is loaded by the positive transition into latch Rt2 in stage 3 of RP1 where it will be used in the calculation of the low coefficient. On the other hand, during the high pulse, as indicated in Figure 4.3.7, the high coefficient in *Rt1* which is needed in the next run will be stored in the TLB.

The register labeled *TLBAR* (TLB address register) generates addresses for the *TLB*. Initially, register TLBAR is cleared to zero by asserting signal *incar* low to point at the first location in the *TLB*. Then to address the next location after each read and write, register *TLBAR* is incremented by asserting *incar* high. Each time a run is complete, register *TLBAR* is cleared zero to start a new run and the process is repeated.

The two multiplexers labeled *muxh* and *muxl* are used for passing every clock cycle, reference to clock f_3, the high and low coefficients, respectively, generated by the three RPs. The two control signals of the two multiplexers are shown in Figure 4.3.6 connected to clocks f_{3a} and f_{3b}. When the two pulses of the clock f_{3a} and f_{3b} are low, the two multiplexers would pass the output coefficients generated by RP1, whereas when a high pulse of the clock f_{3a} and a low pulse of the clock f_{3b} occur, the two multiplexers would pass the output coefficients generated by RP2 as indicated in Figure 4.3.7. Finally, when the two pulses are high, the two multiplexers would pass the output coefficient of RP3. In addition, note that the path extending from the inputs of the multiplexer *muxh,* passing through *muxh2, muxce0,* and ending at Rt2 may form a critical path, since signals through this path should reach Rt2 during one cycle of clock f_3.

The registers labeled *SRH1, SRH0, SRL1,* and *SRL0,* including *RdH* and *RdL* operate with frequency f_3. Registers *SRH1, SRH0,* and *RdH* store high coefficients, while registers *SRL1, SRL0,* and *RdL* store low coefficients. New coefficients are loaded simultaneously into both CP1 and CP3 latches every time clock f_{3a} makes a positive transition, whereas CP2 latches are loaded when clock f_{3b} makes a positive transition. Furthermore, each time a transition from a run to the next is made, when the column length *(N)* of an image is odd, the external memory should not be scanned for 3 clock cycles, since during this period the CPs will process the last high and low coefficients in each of the 3 columns of H and L decompositions, as required by the DDGs for odd signals. This is also true for 2-parallel intermediate architecture. No such situation occurs when the column length of an image is even.

It can be reasoned from Table B.17, the control signals of the two multiplexer's labeled *muxh2, muxh3,* and register RdH can all be combined into one signal, *sh2.* Similarly, the control signal of the two multiplexer's *muxl2, muxl3,* and register RdL can be combined into one signal, *sl2.* Furthermore, a careful examination of Table B.17 shows that the control signal values that must be issued by the control unit for signals *sh1, sh0, sl1, sl0, sh2,* and *sl2,* starting from cycles 16 to 21 and repeat every 6 cycles, should be as shown in Table 4.8

Table 4.8 control signal values

Cycle	*Sh1*	*Sh0*	*sl1*	*sl0*	*Sh2*	*sl2*
16	1	1	1	1	0	0
17	1	1	0	0	0	1
18	0	0	0	0	1	1
19	1	1	1	1	1	1
20	1	0	1	1	0	1
21	1	0	1	0	0	0

Moreover, if it is necessary to extend the 2-parallel architecture to 4-parallel architecture, from the experience gained in designing 2- and 3-parallel architectures, the best architecture for 4-parallel would be obtained if the fourth overlapped scan method is used and 5-parallel if the fifth scan method is used and so on. Then the architecture design for a higher degree parallelism becomes similar to that experienced in the 3-parallel intermediate architecture. While an attempt, e.g., to design 4-parallel intermediate architecture using the third scan method would require very complex modifications in the datapath architecture of the combined 4 RPs and complex control logic. However, the objective for choosing a higher scan method in the first place is to reduce the power consumption due to overlapped areas scanning of external memory. Therefore, it makes sense if 4-parallel is designed with fourth scan method and 5-parallel with fifth scan method and so on.

4.3.4 Scale factor multipliers reduction

In the lifting-based tree-structured filter bank for 2-D DWT shown in Figure 3.1.1, it can be observed that the high output coefficients, which form H decomposition, each is multiplied by the scale factor k in the first pass. In the second pass, the high output coefficients, which form HH subband, each is multiplied by k. This implies the first multiplication can be eliminated and the output coefficients of HH subband can be multiplied by k^2 using one multiplier after the second pass. While, the high output coefficients, which form HL subband, each is multiplied by $1/k$. This implies no multiplications are required and scale multipliers along this path can be eliminated, since HL subband coefficients are formed by multiplying each coefficient in the first pass by k and then in the second by pass by $1/k$.

On the other hand, the low output coefficients of the first pass, which form L decomposition, each is multiplied by $1/k$. Then in the second pass, the output coefficients, which form LH subband, each is multiplied by k, which implies no multiplications are required along this path. While, the output coefficients of the second pass, which form LL subband, each is multiplied by $1/k$. Thus, instead of performing two multiplications, one multiplication can be performed by $1/k^2$ after the second pass [22, 23, 59]. However, note that the simple computations involve in each lifting step of the 5/3 and 9/7 algorithms have made arriving at these results possible.

This process reduces number of multipliers used for scale factor multiplications in the tree-structured filter bank to 2 instead of 6 multipliers. When it applied to single pipelined architectures, it reduces number of scale multipliers to 2 instead of 4, whereas, in 2- and 3-parallel pipelined architectures, it reduces number of scale multipliers to 2 and 4 instead of 8 and 12, respectively.

In [23], it has been illustrated that the multipliers used for scale factor k and coefficients α, β, γ, and δ of the 9/7 filter can be implemented in hardware using only two adders.

4.3.5 Evaluation of performance

To evaluate the performance of the two proposed parallel architectures in terms of speedup, throughput, and power consumption as compared with the single pipelined intermediate architecture consider the following. In the single pipelined intermediate architecture, the total time, *T1*, required to yield n paired outputs for j-level decomposition of an *NxM* image is given by

$$T1 = [\rho_1 + 3(n-1)]\tau_1 = [\rho_1 + 3(n-1)]t_p/3k \tag{4.32}$$

The dataflow of the 2-parallel architecture in Table B.12 shows that $\rho_2 = 19$ clock cycles are needed to yield the first 2-pair of output. The remaining *(n-2)/2* outputs require *2(n-2)/2* cycles. Thus, the total time, *T2*, required to yield n paired outputs is given by

$$T2 = [\rho_2 + (n-2)]\tau_2 \tag{4.33}$$

From *statement3*, $\tau_2 = t_p/2k$ then

$$T2 = [\rho_2 + (n-2)]t_p/2k \tag{4.34}$$

The speedup factor S is then given by

$$S_2 = \frac{T1}{T2} = \frac{[\rho_1 + 3(n-1)]t_p/3k}{[\rho_2 + (n-2)]t_p/2k} \tag{4.35}$$

For large n, the above equation reduces to

$$S_2 = \frac{3(n-1)(2k)}{(n-2)(3k)} = 2 \tag{4.36}$$

Eq (4.36) implies that the proposed 2-parallel intermediate architecture is 2 times faster than the single pipelined intermediate architecture.

On the other hand, to estimate the total time, *T3*, required for j-level decomposition of an *NxM* image on the 3-parallel pipelined intermediate architecture, assume the output generated by CP2 in Table B.17 are shifted up one clock cycle so that it parallel that of CP1 and CP3. Then, $\rho_2 = 25$ clock cycles are needed to yield the first 3-pair of output. The remaining *(n-3)/3* 3-paired outputs require *3(n-3)/3* clock cycles. Thus, the total time, *T3*, required to yield *n* paired outputs is given by

$$T3 = [\rho_3 + (n-3)]\tau_3 = [\rho_3 + (n-3)]t_p/3k \tag{4.37}$$

The speedup factor *S* is then given by

$$S_3 = \frac{T1}{T3} = \frac{[\rho_1 + 3(n-1)]t_p/3k}{[\rho_3 + (n-3)]t_p/3k} \tag{4.38}$$

$$S_3 = \frac{3(n-1)}{(n-3)} = 3 \tag{4.39}$$

Eq (4.39) implies that the proposed 3-parallel pipelined intermediate architecture is 3 times faster than the single pipelined intermediate architecture.

The throughput, *H*, which can be defined as number of output coefficients generated per unit time, can be written for each architectures as

$$H(\sin gle) = n/(\rho_1 + 3(n-1))t_p/3k \tag{4.40}$$

The maximum throughput, H^{max}, occurs when *n* is very large ($n \rightarrow \infty$). Thus,

$$\begin{aligned} H^{\max}(\sin gle) &= H(\sin gle)_{n\to\infty} \\ &\cong 3\cdot n\cdot k\cdot f_p/3\cdot n = k\cdot f_p \end{aligned} \tag{4.41}$$

$$H(2-parallel) = n/(\rho_2 + (n-2))t_p/2k \tag{4.42}$$

$$\begin{aligned} H^{\max}(2-parallel) &= H(2-parallel)_{n\to\infty} \\ &\cong 2\cdot n\cdot k\cdot f_p/n = 2\cdot k\cdot f_p \end{aligned} \tag{4.43}$$

$$H(3-parallel) = n/(\rho_3 + (n-3))t_p/3k \tag{4.44}$$

$$\begin{aligned} H^{\max}(3-parallel) &= H(3-parallel)_{n\to\infty} \\ &\cong 3\cdot n\cdot k\cdot f_p/n = 3\cdot k\cdot f_p \end{aligned} \tag{4.45}$$

Hence, the throughputs of the 2-parallel and 3-parallel pipelined architectures have increased by a factor of 2 and 3, respectively, as compared with single pipelined architecture.

To determine the amount of power reduction achieved in the external memory of the intermediate parallel architecture as compared with first scan method based parallel architecture, consider the following. If the power consumption of VLSI architectures can be estimated as

$$P = C_{total} \cdot V_o^2 \cdot f \tag{4.46}$$

where C_{total} denotes the total capacitance of the architecture, V_o is the supply voltage, and f is the clock frequency, then the power consumption due to scanning external memory of the single pipelined architecture based on nonoverlapped scan method can be written as

$$P_s(non) = \beta \cdot C_{total} \cdot V_o^2 \cdot f_1 \tag{4.47}$$

where $C_{total} \cdot V_o^2 \cdot f_1$ is the external memory power consumption due to first overlapped scan method, f_1 is the external memory scan frequency, and $\beta = T_{mn}/T_{mo} = 2/3$. T_{mo} and T_{mn} denote total external memory access time in clock cycle for J levels of decomposition for architecture based on the first overlapped and nonoverlapped scan methods, respectively.

Using the fact that the scan method shown in Figure 3.7.1 (a) reduces the power consumption of the overlapped areas by a factor of 1/3, the power consumption due to scanning the overlapped areas of Figure 3.7.1 (a) can be written as

$$P_o(areas) = \left(\beta_0 \cdot C_{total} \cdot V_o^2 \cdot f_1/3\right) \tag{4.48}$$

where $\beta_0 = T_{me}/T_{mo} = 1/3$ and T_{me} is the excess memory access time due to overlapped areas scanning for J levels of decomposition. Thus, the external memory power consumption of the single pipelined intermediate, $P_s(int)$, is

$$\begin{aligned}
P_s(\text{int}) &= P_s(non) + P_o(areas) && (4.49)\\
&= \beta \cdot C_{total} \cdot V_o^2 \cdot f_1 + \beta_0 \cdot C_{total} \cdot V_o^2 \cdot f_1/3 && (4.50)\\
&= C_{total} \cdot V_o^2 \cdot f_1(\beta_0/3 + \beta) && (4.51)\\
&= 3 \cdot k \cdot C_{total} \cdot V_o^2 \cdot f_p(\beta_0/3 + \beta) && (4.52)
\end{aligned}$$

where $f_1 = 3 \cdot k \cdot f_p$, and f_p is processor's frequency.

The external memory power consumption of l-parallel pipelined intermediate architecture, $P_l(\text{int})$ can be written as

$$P_l(\text{int}) = I \cdot C_{total} \cdot V_o^2 \cdot f_l \cdot (\beta_0/3 + \beta) \tag{4.53}$$

From *statement3*, $1/\tau_l = f_l = l \cdot k/t_p$, then

$$P_l(\text{int}) = l \cdot k \cdot I \cdot C_{total} \cdot V_o^2 \cdot f_p \cdot (\beta_0/3 + \beta) \tag{4.54}$$

where (I) is number of input buses and is 3 in the parallel architecture.

Similarly, the external memory power consumption of l-parallel pipelined architecture based on the first scan, $P_l(first)$ can be written as

$$P_l(first) = I \cdot C_{total} \cdot V_o^2 \cdot f_l = l \cdot k \cdot I \cdot C_{total} \cdot V_o^2 \cdot f_p \tag{4.55}$$

Thus,

$$\frac{P_l(\text{int})}{P_l(first)} = \frac{l \cdot k \cdot I \cdot C_{total} \cdot V_o^2 \cdot f_p \cdot (\beta_0/3 + \beta)}{l \cdot k \cdot I \cdot C_{total} \cdot V_o^2 \cdot f_p} \tag{4.56}$$

$$= \beta_0/3 + \beta = 7/9 \tag{4.57}$$

implies that the intermediate parallel architecture based on scan method shown in Figure 3.7.1 (a) reduces power consumption of the external memory by a factor of 7/9 as compared with parallel architecture based on the first scan method. On the other hand,

$$\frac{P_l(\text{int})}{P_s(\text{int})} = \frac{l \cdot k \cdot I \cdot C_{total} \cdot V_o^2 \cdot f_p \cdot (\beta_0/3 + \beta)}{3 \cdot k \cdot C_{total} \cdot V_o^2 \cdot f_p \cdot (\beta_0/3 + \beta)} = l \tag{4.58}$$

implies that as the degree of parallelism increases the external memory power consumption of the intermediate parallel architecture based on the scan method in Figure 3.7.1 (a) also increases by a factor of l as compared with single pipelined intermediate architecture's external memory power consumption.

4.4 Conclusions

In this chapter, the single pipelined overlapped architecture is extended to 2-parallel, 3-parallel, and 4-parallel architectures to achieve speedup factors of 2, 3, and 4, respectively, according to the evaluation given in section 4.2.4. Similarly, the single pipeline intermediate architecture is extended to 2-parallel and 3-parallel architectures. According to the evaluation given in section 4.3.5, the 2-parallel and 3-parallel intermediate architectures achieve speedup factors of 2 and 3, respectively. The intermediate parallel architecture reduces the power consumption of the external memory by a factor of 7/9 as compared with the overlapped parallel architecture, Eq(4.57). The advantage of the parallel architectures developed in this chapter, is that the total temporary line buffer (TLB) requirement does not increase from the single pipelined architectures.

CHAPTER 5

DWT MEMORY ARCHITECTURES

5.1 Introduction

DWT memory architectures have been usually overlooked in the literature. However, since 2-D DWT memory architectures are equally important as DWT processor architectures commonly covered in the literature, in this chapter, two novel VLSI architectures for LL-RAM and subband memory are developed.

The general structure of a compression system is shown in Figure 5.1.1. The DWT unit generally consists of a row-processor (RP) and a column-processor (CP). RP reads LL-RAM, while CP writes into LL-RAM and subband memory.

DWT decomposes an *NxM* image into subbands, as shown in Figure 2.1.3 for 3 decomposition levels. These subbands must be stored by DWT unit in a memory such that they can be manipulated effectively by compression unit for compression purposes. Therefore, a memory architecture, which allows DWT unit to perform efficiently both, reads and writes and compression unit to perform reads is necessary.

Figure 2.1.3 shows that the first decomposition generates 4 subbands labeled HL1, HH1, LH1, and LL1. The coefficients of the first 3 subbands would be stored in a memory, call it subband memory, which would contain memory blocks HL1, *HH1*, and *LH1*. The compression unit can then read the 3 subbands and compress each independently, while subband LL1 would be stored in another memory, call it, LL-RAM or just RAM, for further decompositions.

The second decomposition generates 4 subbands, labeled HL2, HH2, LH2, LL2, by reading subband LL1 coefficients stored in the LL-RAM. The coefficients of the 3 subbands HL2, HH2, and LH2 would be stored also in the subband memory blocks labeled *HL2*, *HH2*, and *LH2*, while subband LL2 would be stored in the RAM for further decomposition.

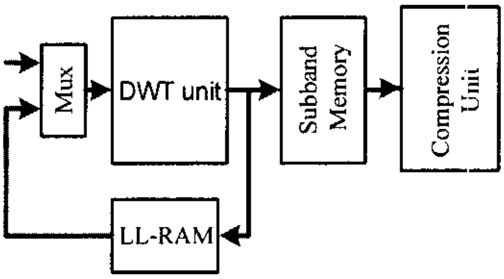


Figure 5.1.1 General structure of a compression system.

In the discussion above, two memory components have been identified, the LL-RAM and the subband memory, that need to be designed such that DWT unit can perform effectively both read and write operations in the LL-RAM and write only into subband memory, while compression can read subband memory. Thus, in this chapter, the architectures of the LL-RAM and subband memory would be developed. First, the LL-RAM architecture will be developed followed by subband memory architecture.

5.2 The LL-RAM architecture development

The LL-RAM is used by the DWT unit to store the coefficients of the LL subband that it generates in each decomposition level, for further decompositions. In the DWT unit, the RP scans (reads) the LL-RAM, and the CP writes the LL subband coefficients in the LL-RAM. The generalized scan method requires the RAM to be read in every clock cycle with frequency f_l, where $l=1,2,3$ denote single, 2- or 3-parallel, and to be written according to the order in which each scan method generates its output coefficients. Which implies that reads operations will coincide with writes operations. Therefore, the RAM architecture should be designed such that both read and write can take place in the same clock cycle. Thus, the first half cycle of clock f_l will be reserved for read and the second half cycle for write.

The RAM, which can be viewed as a 2-dimensional memory of size $N/2xM/2$, where $N = 2^n$ *and* $M = 2^m$, can be readily constructed from $M/2$ modules with each module having $N/2$ locations.

The block diagram of the memory module that would be used in forming the RAM architecture is shown in Figure 5.2.1. The E signal, which is active high, enables the module for reads and writes. The module is read and the result is placed in the output bus when the signal labeled $\overline{R}/W$ is low, otherwise, it is written. The address bus is used in addressing each location in the module for read or write. The control signal labeled *MS* (module select) is useful when several modules are used in forming a memory. It allows through a decoder one module to be selected for read or write. The module can be read or written only when both signal *E* and *MS* are asserted high.

The complete architecture of the RAM that facilitates both reads and writes is shown in Figure 5.2.2 (a) and (b). This architecture is based on the first scan method. However, the RAM architecture can be easily modified to handle other (or higher) scan methods, as will be explained later. The decoder labeled *dcodms* is responsible for selecting modules for reads or writes. When the architecture performs read operations, the register labeled *RMSR* (read module select register) determines through *muxs* which modules to be enabled. When it performs write operations, register *WMSR* (write module select register) is used for selecting modules. Both registers are (*m-2*)-bit counters with control signal *clr* (clear) and *inc* (increment) and operate with frequency f_l.

The multiplexer labeled *muxs*, its control signal is shown connected to clock f_l. When f_l is low, read operation takes place and *RMSR* controls the decoder. On the

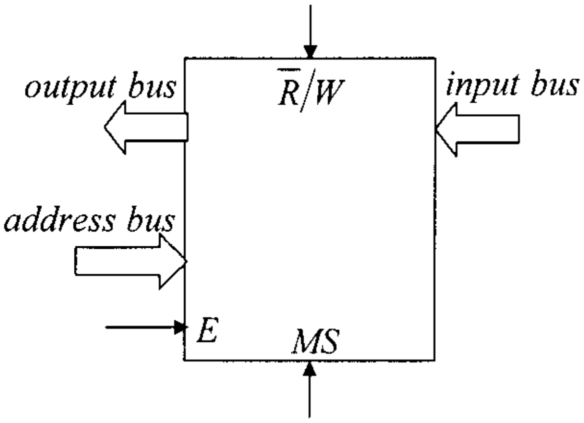


Figure 5.2.1 Block diagram of the memory module.

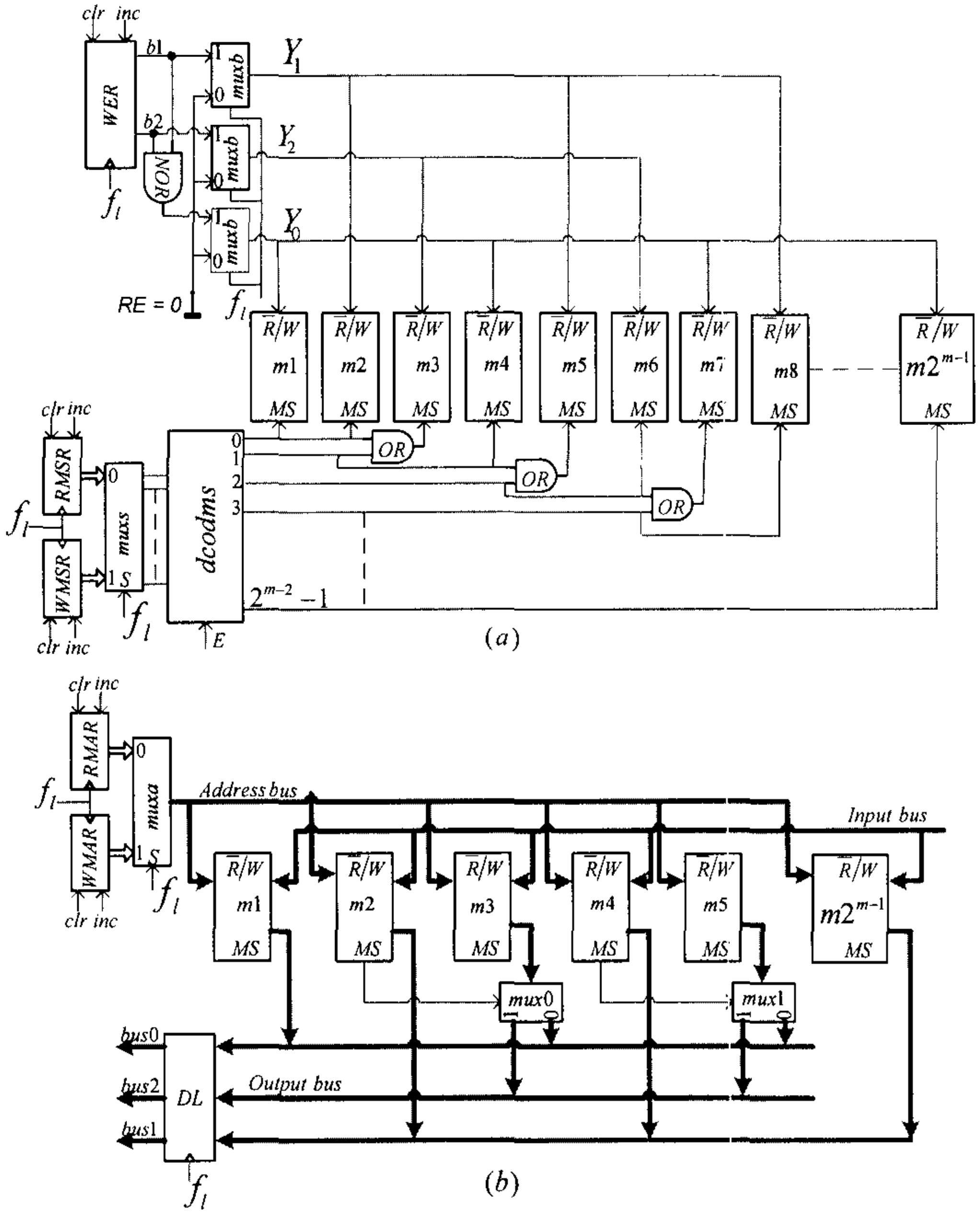


Figure 5.2.2 (a) and (b) RAM architecture using modules

other hand, when f_l is high, write operation takes place and *WMSR* controls the decoder.

The circuit in the upper left corner of Figure 5.2.2 (a), consisting of 3 multiplexers labeled *muxb*, a *NOR* gate, signal *RE* and the register labeled *WER* (write enable register), is in charge of generating signal values for the read and write signal labeled $\overline{R}/W$. The control signals of the 3 multiplexers are also shown connected to the clock f_l. When f_l is low, modules are enabled for read, otherwise, are enabled for write. Signals generated by this circuit will be described later in details.

The address bus is managed by two registers labeled RMAR (read module address register) and WMAR (write module address register) through *muxa*. When the multiplexer control signal is driven low by clock f_l, read operation takes place and RMAR provides addresses to modules, otherwise, write operation takes place and WMAR provides address to modules.

In the following, read and write operations will be described in details. First, read operations will be described followed by write operations.

5.2.1 The LL-RAM read operations

The LL-RAM is read according to the scan method shown in Figure 3.5.1, the first overlapped scan method. This scan method requires reading every clock cycle 3 pixels simultaneously, one from each module as follows. When f_l is low, the 3 multiplexers labeled *muxb* pass signal *RE*, which is active low, to the output signals *Y0*, *Y2*, and *Y1*. The three output signals enable all memory modules for read. However, the scan method requires that in every run 3 modules should be enabled for read as follows. First, modules 1, 2, and 3 should be enabled then modules 3, 4, and 5 followed by modules 5, 6, and 7 and so on. Thus, the role of the decoder labeled *dcodms* is to guarantee that modules are enabled in the order specified above. First, the output of the decoder labeled 0 will be activated to enable modules 1, 2, and 3. Then, using the address bus, the first location of each enabled module is read into the output buses. When f_l makes a positive transition, the 3 pixels in the output buses are loaded into a temporary register labeled *DL* (Data latch). Then the negative transition of the clock f_l loads the 3 pixels into the RPs latches. To address the second location in each module, the negative transition of f_l increments also register RMAR. This process is repeated until the 3 enabled modules are read.

To enable the next 3 modules, register RMSR is incremented by one, which asserts the second output of the decoder high. The decoder output labeled 1 enables modules 3, 4, and 5 for read. When all 3 modules are read, the decoder output line labeled 2 is activated by incrementing RMSR again by one to enable the set that contains modules 5, 6, and 7. This process is repeated until the whole RAM is scanned.

In Figure 5.2.2 (b), the output of modules 3 and 5 are shown connected to *mux0* and *mux1*, respectively. These multiplexers are necessary because all modules with odd numbers, except the first, are scanned twice. For example, in the first run, when locations of module 3 are scanned they are placed in the bus labeled *bus2*, whereas in the second run they are placed in another bus labeled *bus0*. Thus, to allow these multiplexers to switch between *bus2* and *bus0* their control signals are connected to decoder dcodms output lines labeled 0 and 1 and so on.

5.2.2 The LL-RAM write operations

How the LL-RAM should be written can be determined by examining the scan method or the dataflow table of the DWT architecture under consideration. For example, examination of the first scan method shows that the CP would generate output coefficients column-by-column, which implies that the RAM should be written module-by-module.

In general, the RAM can be written as follows. When f_l is high, the outputs *b1* and *b2* of the register labeled *WER* (write enable register), which is initially cleared to zero, and the output of the *NOR* gate are passed through multiplexers to the outputs labeled *Y1*, *Y3*, and Y0, respectively. Since *WER* is initially zero, only *Y0* will be asserted high, which enable for write all modules $1+3i$, where $i = 0,1, 2 \ldots, m-1$. For example, if $m=3$, then modules 1, 4, and 7 will be enabled. However, the RAM is required to be written module-by-module and in order, i.e., first module 1, then 2 followed by 3 and so on and the function of the decoder labeled *dcodms* is to provide this module-by-module control. Thus, the decoder output labeled 0 will be first asserted high through *WMSR* to enable only module number 1 for write. Note that a module is enabled for write when its both signals *MS* and $\overline{R}/W$ are asserted high and all modules are disabled when signal *E* of *dcodms* is low.

When all locations of module 1 are written, *WER* is incremented by one to assert only *Y1* high. *Y1* enables all modules labeled $2+3i$, but since the first output of the decoder is still high, only module 2 will be selected for write. When all locations of module 2 are written, *WER* is incremented again by one to assert this time *Y2* high. *Y2* enables all modules labeled *3+3i* but since the first output of the decoder is still high, only modules 3 will be selected for write. When all locations of module 3 are written, *WER* is cleared to zero to set *Y0* high, and *WMSR* is incremented by one to assert the decoder second output labeled 1 high. Assertion of both *Y0* and the second output of the decoder enable only module 4 for write. This process is repeated until all modules are written.

Note that *WER* is a 2-bit register that count from 0 to 2 and repeats. Furthermore, the amount of data to be written in each decomposition level including number of modules and number of locations to be written in each module, can be determined in advance from the knowledge of the height and width of the image that will be processed.

5.2.3 RAM architecture modifications for higher scan methods

The RAM architecture shown in Figure 5.2.2 can be easily modified to handle other scan method. The circuits in the upper corner of the RAM architecture, consisting of register *WER* and multiplexers labeled *muxb*, remain unchanged. However, modifications for a specific scan method in general, can be obtained by eliminating some of the OR gates whose outputs are connected to signal *MS*, as follows. For example, the second scan method, which requires 5 modules to be considered for read and two modules for write at a time, would require eliminating the first OR gate and connecting the first output of the decoder labeled *dcodms* to signal *MS* of each modules *m1, m2,* and *m3.* While, connections to modules *m4* and *m5,* remain unchanged. Then, the connection pattern of the first 5 modules *m1, m2, m3, m4,* and *m5* is repeated in the next 5 modules *m5, m6, m7, m8,* and *m9* and so on.

Similarly, the third scan method, which requires 7 modules to be considered for read and 3 modules for write at a time, would require eliminating the first and the second OR gates and then connecting the first output of the decoder to *MS* signal of modules *m1, m2,* and *m3* and that of the second output to signal *MS* of modules *m4,* and *m5.* While connections to modules *m6* and_*m7,* remain the same. The connection patterns of the first 7 modules is repeated in the next 7 modules *m7, m8, m9, m10, m11, m12,* and *m13* and so on.

Now, let's see how read operations are performed on the RAM architecture based on the second scan method. Since, the second scan method requires 5 modules to be considered for read at a time, the modules labeled *m1, m2, m3, m4,* and *m5* will be considered first. Thus, to read these modules location-by-location, registers *RMAR* and *RMSR* are reset 0. This will allow register *RMAR* to address the first location in each module and register *RMSR* to enable modules *m1, m2,* and *m3* through the decoder *dcodms*. Then, in the first clock cycle, when f_l is low, the $\overline{R}/W$ signals of modules *m1, m2,* and *m3* are activated for read. This will allow the first location of each modules *m1, m2,* and *m3* to be read into the buses labeled *bus0, bus1,* and *bus2,* respectively. Then register *RMSR* is incremented by 1 to enable modules *m4,* and *m5* for read. When f_l is low, again in the second clock cycle, the first location in each modules *m4* and *m5* are read into *bus1* and *bus2*, respectively. When this is done, register *RMAR* is incremented by 1 to point at the second location in each module. Register *RMSR* is reset 0 to enable again modules *m1, m2,* and *m3*. When f_l is low in the third cycle, the second location in each modules *m1, m2,* and *m3* are read into the buses. Then, register *RMSR* is incremented by 1 to enable modules *m4* and *m5* and disable *m1, m2,* and *m3* though the decoder labeled *dcodms*. Again, when f_l is low in the fourth clock cycle, the second location of each modules *m4* and *m5* are read into bus1 and bus2, respectively. Then, register *RMAR* is incremented by one to address the third location of each module and register *RMSR* is reset 0 to enable again modules *m1, m2,* and *m3*. This process is repeated until the first 5 modules are read. Then the same process is applied on the next 5 modules *m5, m6, m7, m8,* and *m9* and so on.

Similarly, the RAM architectures for third and fourth scan method etc. can be read in the same manner described above. Note that, in the read operations described above for the second scan method, after each read operation performed on modules *m4* and *m5*, the control should return to module *m1* and repeat the process. The same situation also occurs when the next 5 modules *m5, m6, m7, m8,* and *m9* are considered for read and so on. That is, returning to module *m5* from module *m9* should be remembered by the control. Therefore, register XR is added to serve this purpose and it can be connected to register *RMSR* as shown in Figure 5.2.3. A similar problem occurs with write operations using registers *WMSR* and *WER*, and the solution shown in Figure 5.2.3 can be used, which is described in details in section 5.3.4.

This RAM architecture would work well in DWT architectures, where pixels are scanned in parallel, such as in the parallel architectures developed in chapter 5. But, if a DWT architecture is required to scan RAM pixel-by-pixel, then in that case all OR gates in Fig. 5.2.2 (a) are eliminated and each output of decoder *dcodms* is connected only to signal *MS* of one module and the output buses are reduced to one bus.

On the other hand, how the RAM should be written would depend on the scan method adopted. The first scan method, as described earlier, requires the RAM to be written module-by-module. Whereas, the second scan method requires considering 2 modules for write at a time, as follows. Initially, registers WER, WMSR, and WMAR are set 0. Setting WER and WMBR 0 while f_l is high enable module 1 for write, and WMAR addresses the first location of module 1. This will let the first output coefficient, LL0,0 to be stored in the first location of module 1. When the negative transition of clock f_l ending the cycle occurs, it will increment WER by one to enable module 2 for write. During the high pulse of the second cycle of clock f_l, the second coefficient labeled LL0,1 is stored in the first location of module 2, while the negative transition of clock f_l ending the cycle clears WER to enable again module 1 for write and increments WMAR to address the second location of module 1. In this location, the third output coefficient, LL1,0 is stored during the high pulse of the third cycle of clock f_l. The negative transition of clock f_l ending the third cycle, increments WER by one to enable module 2 again for write. During the high pulse of the fourth cycle, the fourth output coefficient, LL1,1 is stored in the second location of module 2. This process is repeated until all required locations in the two modules are written. Then the same process is applied on the next 2 modules *m3* and *m4* and so on. Note that writing into the RAM does not take place every clock cycle as reading but when it occurs it coincides with reading and the order of writing coefficients occur as described above.

Similarly, the third scan method requires writing into 3 modules at a time. In general, the *ith* scan method would require writing into *i* modules at a time.

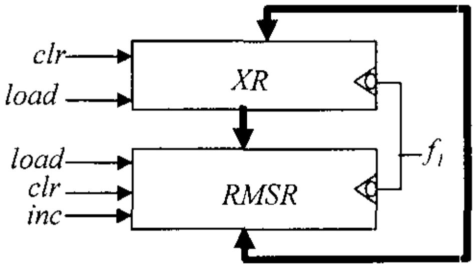


Figure 5.2.3 Incorporation of register XR

5.2.4 RAM architecture using banks

The decoder labeled *dcodms*, in the RAM architecture shown in Figure 5.2.2, is a very large decoder. This large decoder can slow down the LL-RAM's operations and can degrade its performance in terms of speed and power. Therefore, it is necessary to reduce the size of the decoder to a practical level. Furthermore, the signal labeled *Y0, Y1,* and *Y2,* each is shown in Figure 5.2.2 connected to drive read/ write signal labeled $\overline{R}/W$ of several modules. Driving this large capacitive load in this way can also negatively affect the performance of the RAM. For these reasons, the bank method is introduced in Fig. 5.2.4 (a) to alleviate these problems.

Figure 5.2.4 (a) shows a bank structure with 8 modules. The bank can contain any number of 2^b modules where $b = 1, 2, ... m-2$. Read and write operations in the bank can be performed in the same way as described for Figure 5.2.2. Figure 5.2.4 (b) shows the block diagram of the bank. This block diagram is used in building the RAM architecture shown in Figure 5.2.5. This architecture can be thought formed by dividing the architecture in Figure 5.2.2, which can be considered as one big bank holding 2^{m-1} modules, into several smaller independent banks each holding 2^b modules. Inside the smaller banks reads and writes are performed as in the big bank but faster and more efficient.

The architecture performs read or write operations bank-by-bank and in order, *b1* first, *b2* second followed by *b3* and so on. In the architecture shown in Figure 5.2.2, the decoder labeled *dcodms* is used for selecting modules, whereas the decoder labeled *dcodbs* in Figure 5.2.5 is used for selecting banks. When read operation takes place, the register labeled *RBSR* (read bank select register) controls the decoder but when write operation takes place, the register labeled *WBSR* (write bank select

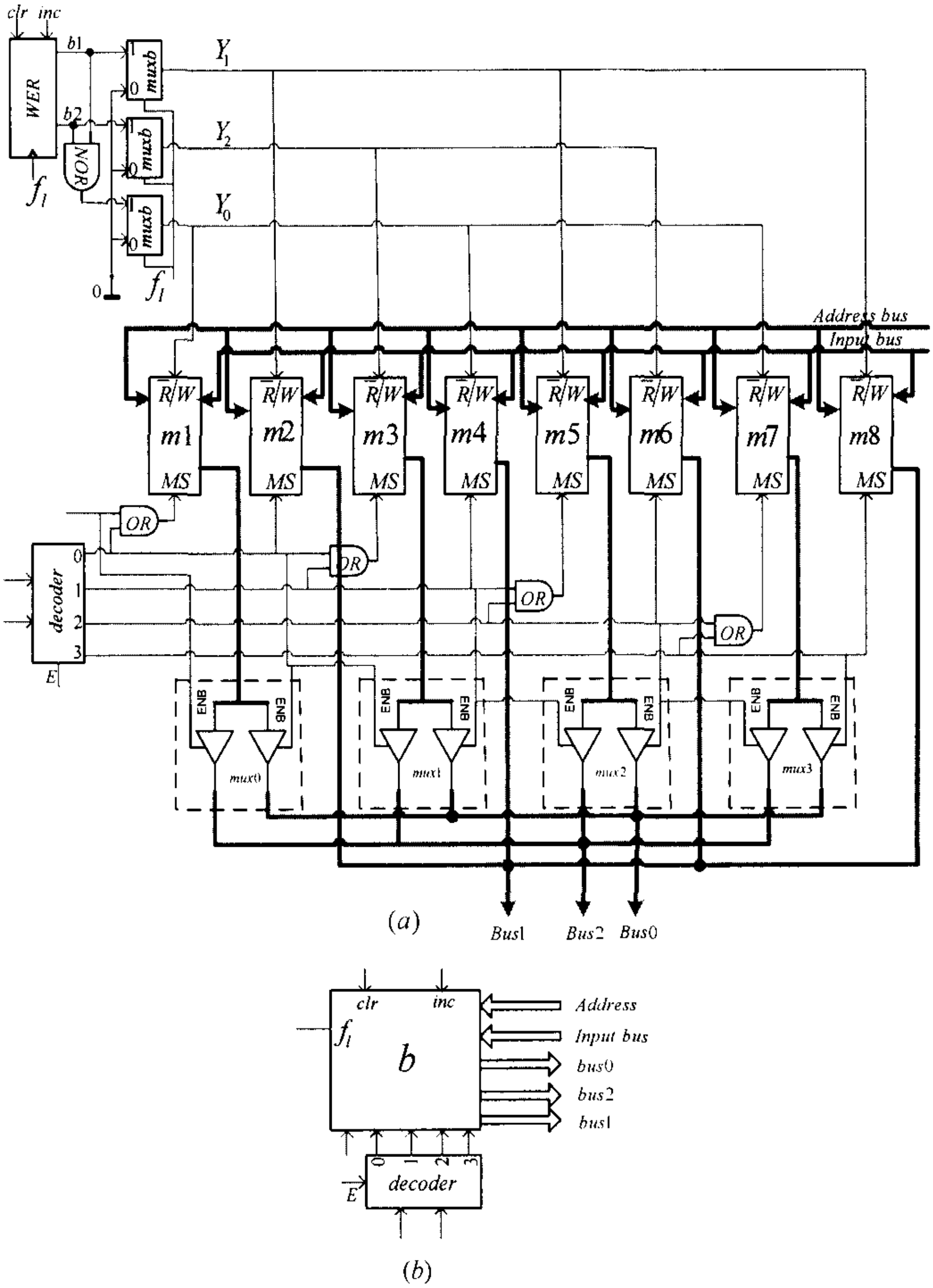
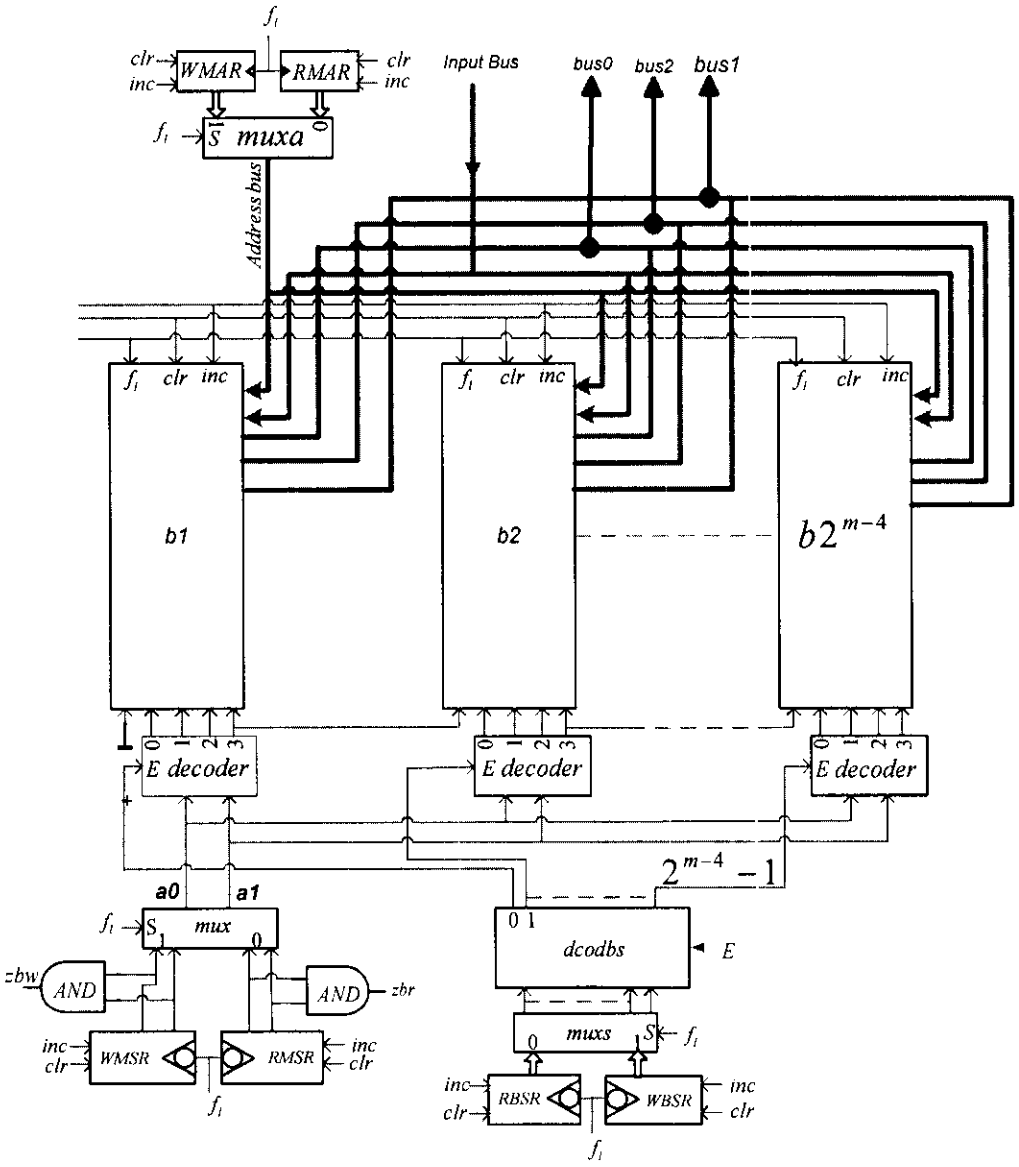


Figure 5.2.4 (a) bank architecture with 8 modules (b) its block diagram



zbw : all modules in a bank are written
zbr : all modules in a bank are read

Figure 5.2.5 RAM architecture using bank

register) controls the decoder. Both registers are *(m-4)*-bit counters with control signals *clr* (clear) and *inc* (increment).

The decoders which are attached to the banks labeled *b1* and *b2* etc. in the RAM architecture shown in Figure 5.2.5, each is responsible for selecting modules when its bank is enabled by decoder *dcodbs*. When the architecture performs read operations in bank *b1*, for example, the register labeled *RMSR* (read module select register) controls the decoder output through *mux*. When it performs write operations, the register labeled *WMSR* (write module select register) controls output of the enabled decoder. Registers RMSR *and* WMSR both are 2-bit counters that count from 0 to 3 and repeats. When all modules in a bank are read or written, the signals labeled *zbr* or *zbw* will be asserted high, respectively, indicating that the next bank can now be enabled by *dcodbs*. To see how effective the bank method in reducing the decoder size, consider the following. Suppose, $M=2^m$ is the largest image width that can be processed by the DWT unit. Then, the maximum number of modules in the RAM will be (2^{m-1}) modules with decoder size $m-2$: 2^{m-2}. Now, if each bank is structured to contain 2^b modules, then

$$2^{m-1}/2^b = 2^{m-b-1} \tag{5.1}$$

represents number of banks and number of decoder *dcodbs* outputs. Whereas,

$$2^{m-b-1}/2^{m-2} = 2^{-b+1} \tag{5.1}$$

gives the reduction in the decoder size. Thus, if $b=3$, the decoder size decreases by a factor of 4.

5.3 Subband memory architecture development

The basic architecture of the subband memory is shown in Figure 5.3.1. The architecture is developed with two objectives in mind to achieve, that is, write operations by the DWT unit and read operations by compression unit, which are somewhat complex operations, should be performed effectively.

The strategy adopted for managing subband memory architecture for an *NxM* image is as follow. The first decomposition, which consist of subbands HL1, HH1, and LH1, are stored in the memory blocks labeled *HL1*, *HH1*, and *LH1*, respectively. Then, the compression unit is informed to read these memory blocks. The compression unit can read each subband memory block code-block by code-block for EBCOT (Embedded Block Coded with Optimized Truncation) coding as required by JPEG2000 standard [7]. The compression unit applies compression algorithm on each code-block independently. The compression unit first reads contents of *HL1*, then *HH1*, and last *LH1*, while, the LL1 subband coefficients, which are stored in the RAM, are scanned by the RPs for further decomposition.

Subbands of the second decomposition HL2, HH2, and LH2, are stored in the subband memory blocks labeled *HL2, HH2*, and *LH2*, whereas, subbands of the third decomposition are stored in the subband memory blocks labeled *HL3, HH3*, and *LH3*, and so on. However, subbands of the last decomposition are stored in the subband memory labeled HL_{jmax}, HH_{jmax}, LH_{jmax}, and LL_{jmax}.

When the LL1 subband is decomposed into the required number of decomposition levels, the compression unit is again informed. Thus, the compression unit is informed twice during the whole decomposition process. First, when subbands of the first decomposition are available in subband memory blocks *HL1, HH1*, and *LH1*. Second, when all subsequence decompositions of LL1 subband are completed and are stored in their respective subband memory blocks.

5.3.1 The bank structure used in forming subband memory

In Figure 5.3.1, each block of the subband memory labeled *HL1, HH1*, etc. is a 2-dimensional memory block, size $2^{n-j}x2^{m-j}$, where $j = 1, 2, 3 ... jmax$ and *jmax* is the maximum number of decomposition levels allowed. Two methods of forming a bank containing modules are shown Figures 5.3.2 and 5.3.3. The first bank shown in Figure 5.3.2 contains 2^h modules. When signal *EM* is asserted high, it enables the bank for both read and writes operations. Whereas, which module to read or write is determined by the decoder and the address lines are used to address each location in the selected module starting from location zero to location $2^{n-j}-1$. The block diagram of the bank is shown in Figure 5.3.2 (b).

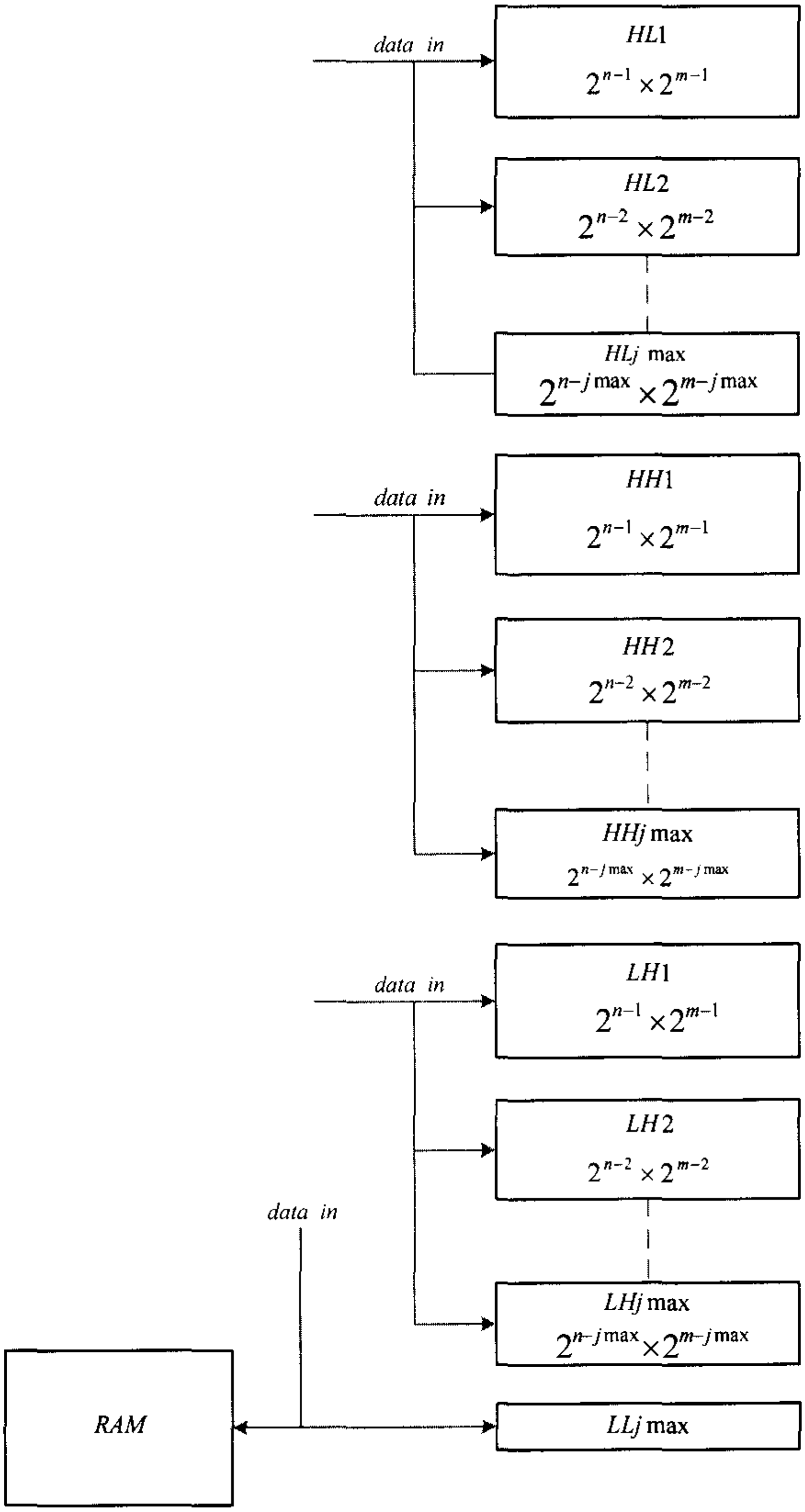
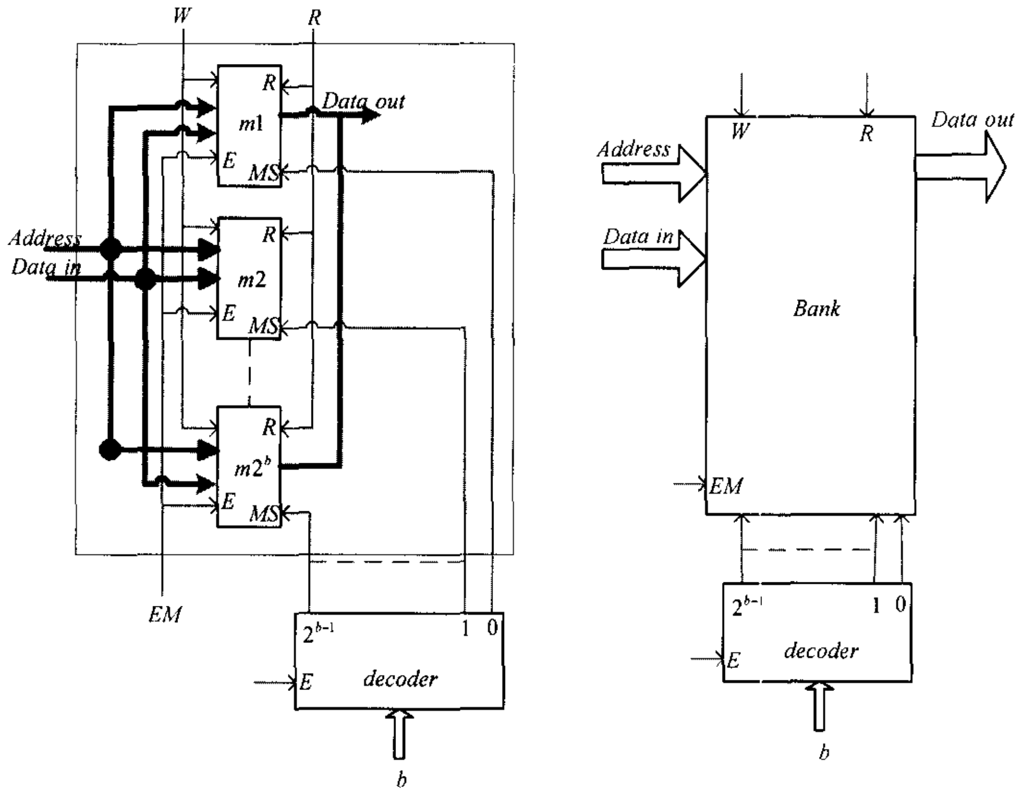


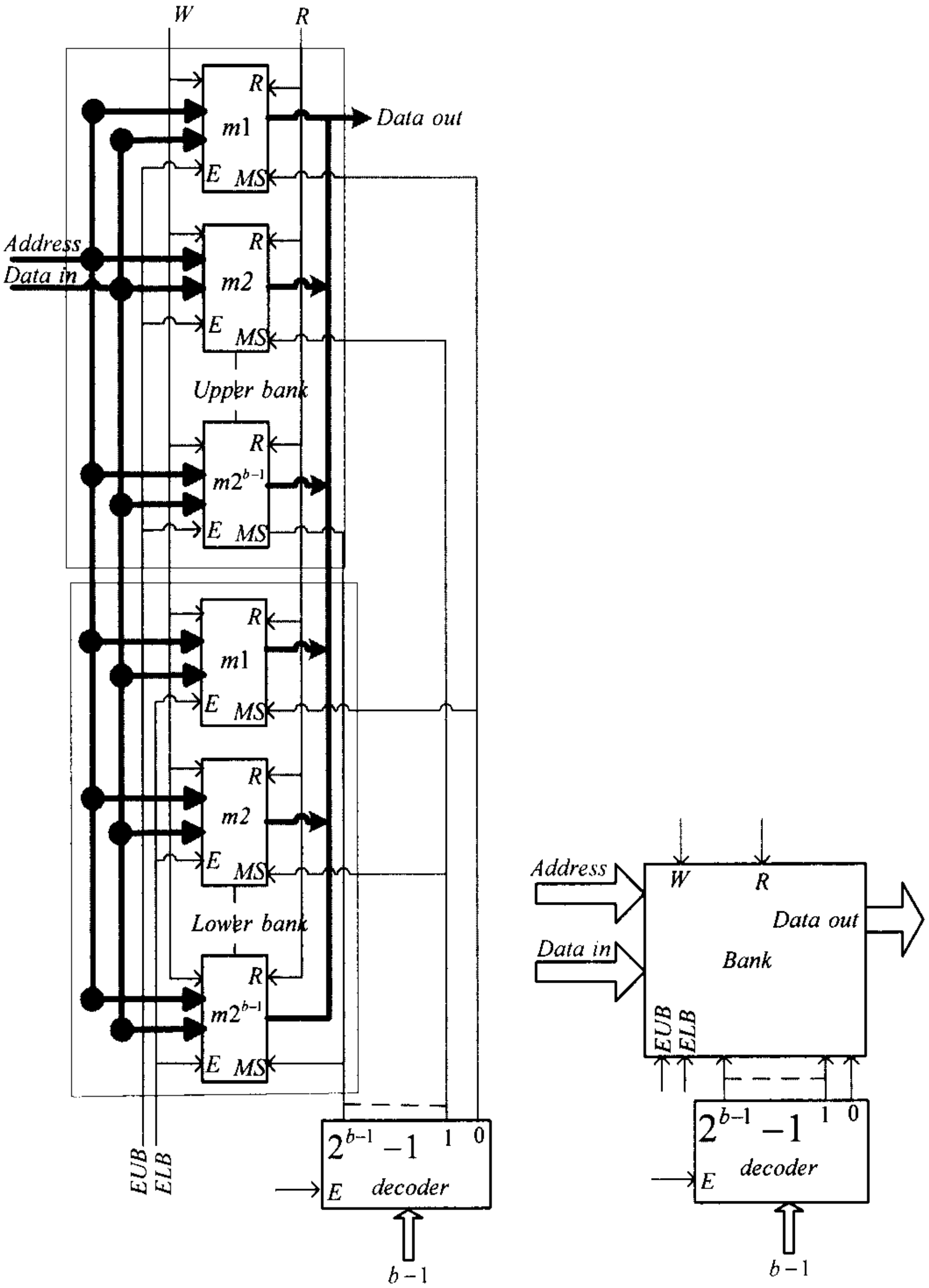
Figure 5.3.1 Subband memory architecture

The second bank and its block diagram are shown in Figure 5.3.3 (a) and (b), respectively. It consists of two small banks, the upper and the lower banks, which in turn form a larger bank. The second bank method reduces the decoder size by ½ as compared with the first bank method, and allows more packing of modules into a bank. The number of modules in the larger bank is 2^b, while the lower and upper banks each contains (2^{b-1}) modules as indicated in Figure 5.3.3(a). Reads or writes into the bank take place module-by-module. Modules in the upper bank are read (or written) first followed by the lower bank modules. When signal *E* is enabled, the upper bank is selected by asserting the signal *EUB* (enable upper bank), whereas the lower bank is selected by asserting the signal *ELB* (enable lower bank). Modules in the upper or lower banks are selected by the decoder. Modules are selected in the order specified by the decoder, which selects a module at a time.



(*a*) (*b*)

Figure 5.3.2 (a) structure of the first bank (b) its block diagram



(a)

(b)

Figure 5.3.3 (a) Structure of the second bank (b) its block diagram

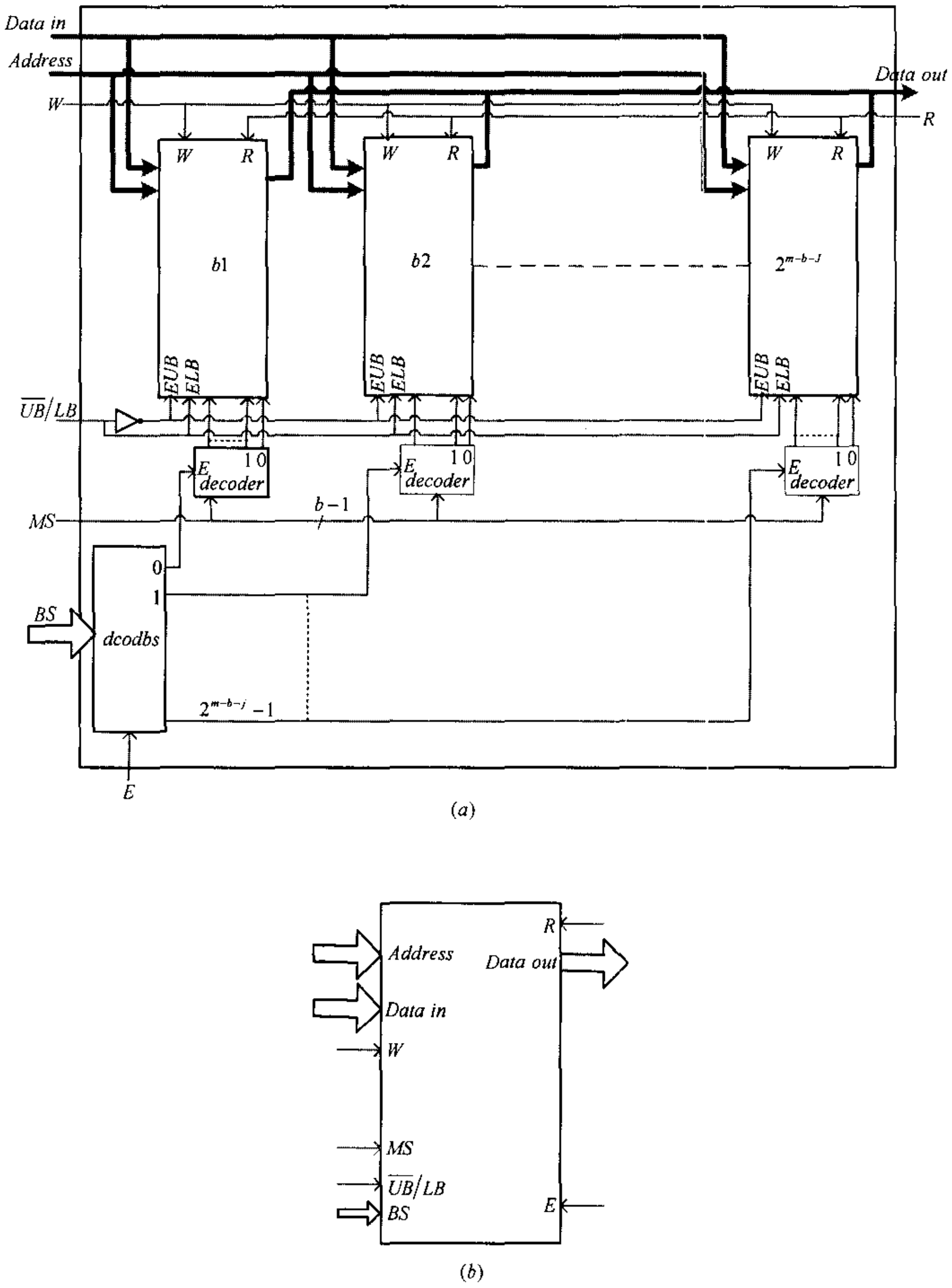


Figure (a) subband memory block architecture formed using the block diagram of the second bank (b) its block diagram

Using the block diagram of the second bank, the subband memory block architecture shown Figure 5.3.4 (a) is formed. The architecture consists of 2^{m-b-j}

banks, each bank contains 2^b memory modules and each module contains 2^{n-j} locations. The decoder labeled *dcodbs* in Figure 5.3.4(a) selects one bank at a time for reads or writes. Banks are selected in order, first *b1*, and second *b2* and so on. The modules inside a selected bank are enabled one at a time through the lines labeled *MS* (module select). The line labeled $\overline{UB}/LB$ enables the upper bank when asserted low and the lower bank when asserted high. Reads or writes occur when signal *E* of decoder *dcodbs* is asserted high.

The block diagram of the architecture is shown in Figure 5.3.4(b). This block is used further for forming the subband memory architecture shown Figure 5.3.1. That means, each block in Figure 5.3.1 is replaced by the block diagram shown in Figure 5.3.4(b).

Suppose, for instant, the largest image size that can be processed is $N=M=2^{10}$, *b* is *3*, and the maximum number of decomposition levels, *jmax* is 7. Then, this implies that the subband memory blocks labeled *HL1, HH1,* and *LH1* for j=1, should each be designed to contain 64 banks and each memory module in a bank should contain 2^9 locations. The blocks of the second level labeled *HL2, HH2,* and *LH2* for j=2, each should contain 32 banks and each module in a bank should contain 2^8 memory locations. Similarly, the sizes of the subband memory blocks for third and forth and so on to *jmaxth* level can be determined. Note that the blocks of the last level labeled HL_{jmax}, HH_{jmax}, LH_{jmax}, and LL_{jmax} for $j=jmax=7$, each must be designed with one bank with each module in the bank having 2^3 memory locations. That is each block should be 8x8.

5.3.2 Details of the subband memory architecture

The details of the subband memory architecture and its interconnections are shown in Figures 5.3.5 and 5.3.6. These two figures together give the complete architecture of the subband memory. The architecture is designed to allow the DWT unit to write into subband memory and the compression unit to read it.

The two sets of registers labeled *MAR1* and *MAR2* in Figure 5.3.5 supply address to modules that are selected for reads or writes. *MAR1*, which is an *(n-1)*-bit counter, provides addresses to modules of the first level memory blocks labeled *HL1, HH1,* and *LH1*, whereas, *MAR2*, an *(n-2)*-bit counter, provides addresses to all memory blocks that lay below the first level. Note that in Figure 5.3.6, the 3 signals labeled $BS, \overline{UB}/LB$, and *MS* are grouped together and are connected to the output of the register labeled *SMSR* (subband module select register), where *BS* and *MS* occupy the most and the least significant bits positions, respectively. Grouping of these 3 signals in this way facilitate banks and modules within a bank to be accessed successively. These signals can be generated by register *SMSR*, which is a simple counter. This register will drive these signals and will determine their values by simply counting from 0 to 2^{m-j}, where 2^{m-j} represents number of modules to be written (or read) in each subband memory block. The value in the *SMSR* gives, when a block of the subband memory is enabled for reads (or writes), the bank number and the module number selected in the upper or lower bank. *SMSR1* is an *(m-1)*-bit register and is used along with *MAR1* to address only subband memory blocks of the first level. Whereas SMSR2, which is an *(m-2)*-bit register, is used along with *MAR2* to address all subband memory blocks that lay after the first level.

Figures 5.3.5 and 5.3.6 also show two groups of registers labeled *A* and *B*. These registers make it possible to control storing of output coefficients in the subband memory by either single or parallel pipelined 2-D DWT architectures. Single pipelined architectures generate two output coefficients each clock cycle, reference to the processor's clock. The two output coefficients might belong to either subbands LH and LL or subbands HL and HH. In the first case, one coefficient (the high coefficient) is stored in the subband memory block *LH* using group *B* registers, while the other coefficient (low coefficient) is passed to LL-RAM where it is stored. In the second case, simultaneously, the low and high coefficients are stored in the subband memory blocks *HL* and *HH,* respectively, using group *A* registers. On the other hand, the parallel architectures generate 4 output coefficients every clock cycle that belong to subbands HL, HH, LH, and LL. The 3 coefficients of subbands HL, HH, and LH are stored in the subband memory blocks *HL, HH,* and *LH* using both groups A and *B* registers, while coefficient of subband LL is passed to LL-RAM.

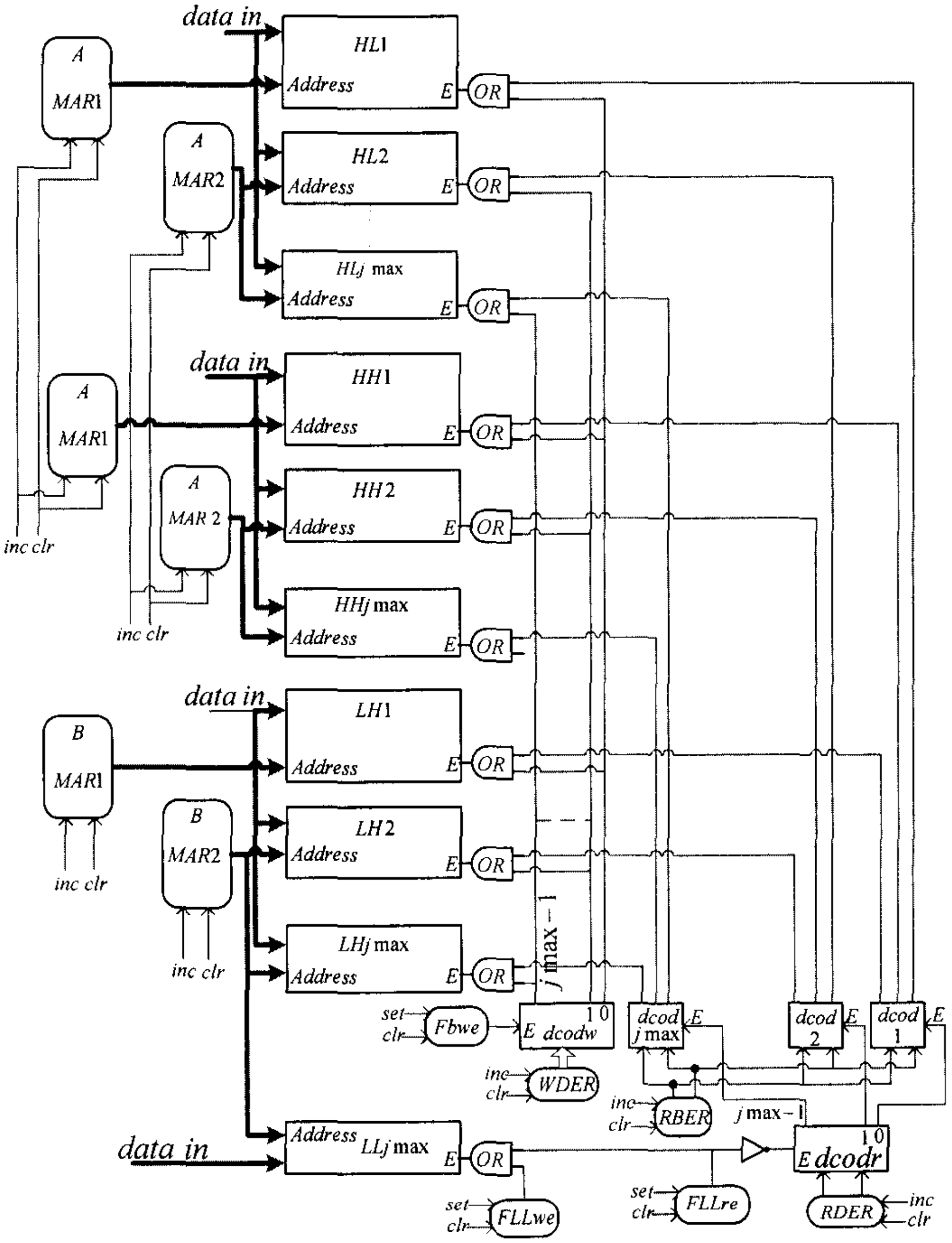


Figure 5.3.5 Architecture of the subband memory

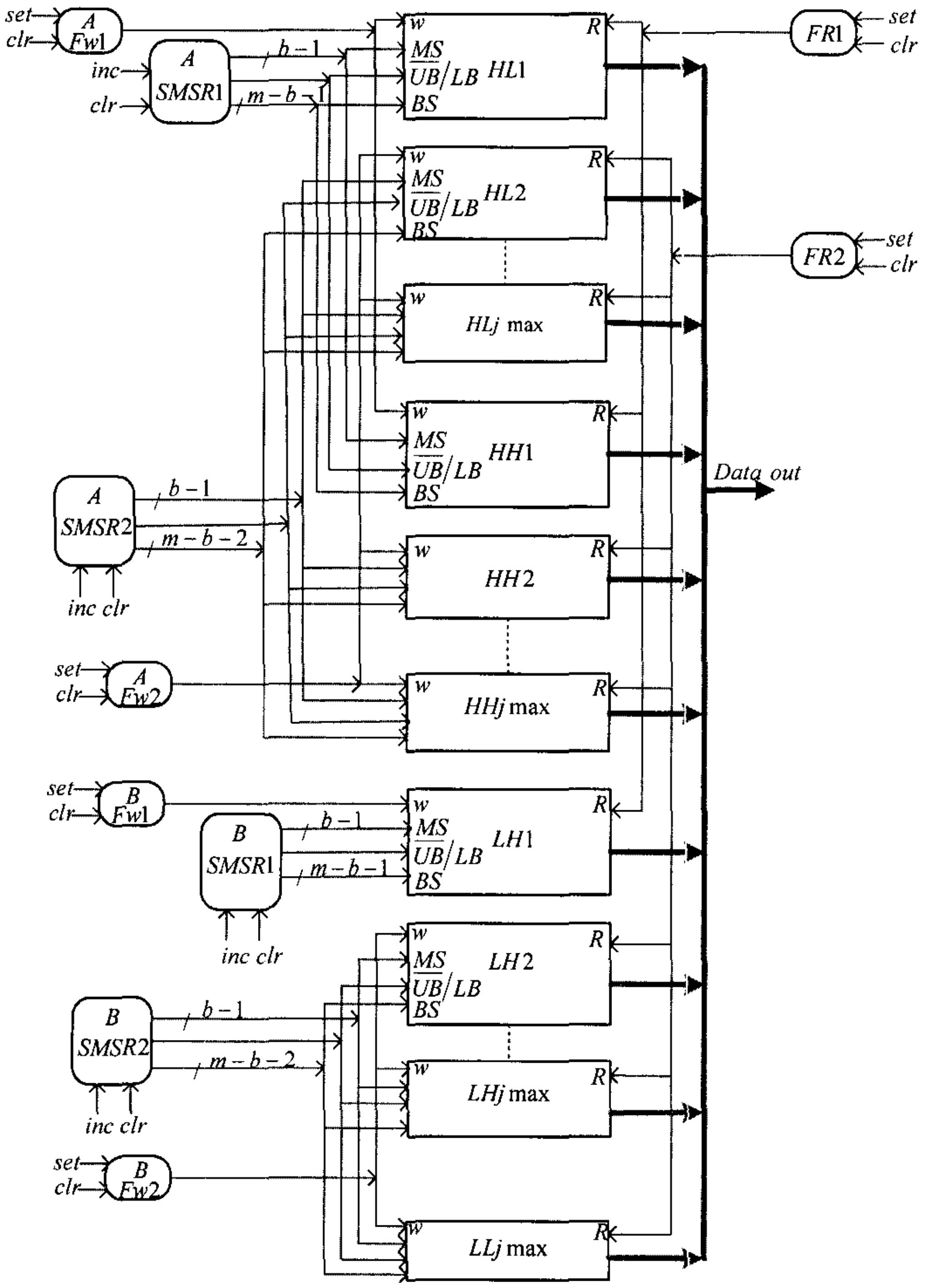


Figure 5.3.6 Architecture of the subband memory

Suppose, now the DWT unit is requested to process, for example, a 256x200 image and to decompose it into 5 levels of decomposition. The first decomposition will generate 4 subbands, each of size 128x100. The 3 subbands HL1, HH1, and LH1 will be written into the subband memory blocks *HL1, HH1,* and *LH1*. That is, in each subband memory blocks *HL1, HH1,* and *LH1*, 100 modules will be written and each module addresses range from 0 to 127. *SMSR1* selects a bank and a module in the bank to be written, while *MAR1* generates addresses for accessing locations in the selected module.

The second decomposition generates also 4 subbands images, each of size 64x50. The 3 subbands HL2, HH2, and LH2 will be written into the subband memory blocks *HL2, HH2,* and *LH2*. In each subband memory block, *SMSR2* is used for selecting a bank and a module in the bank and *MAR2* is used for generating addresses for accessing each location in the module.

The third decomposition generates 4 subbands HL3, HH3, LH3, and LL3 each of size 32x25. The first 3 subbands are stored in the subband memory blocks *HL3, HH3,* and *LH3*, respectively.

The fourth decomposition generates 4 subbands HL4, HH4, LH4, and LL4. Subbands HL4 and HH4 each is of size 16x12, while subbands LH4 and LL4 each is of size 16x13. The first 3 subbands are stored in the subband memory blocks *HL4, HH4,* and *LH4*, respectively.

The fifth decomposition, which is the last decomposition, generates 4 subbands HL5, HH5, LH5, and LL5. Subbands HL5 and HH5 each is of size 8x6, while subbands LH5 and LL5 each is of size 8x7. These 4 subbands are stored in the subband memory blocks *HL5, HH5, LH5,* and LL_{jmax}, respectively. Note that the LL_j subband of the last decomposition should always be stored in the subband memory block labeled LL_{jmax}.

The decoder labeled *dcodw* along with the register labeled *WDER* and the FFs labeled *Fbwe, Fw1, Fw2,* and *FLLwe* are used for enabling subband memory for writes. Whereas the decoder labeled *dcodr* along with the two registers labeled *RDER* and *RBER*, and the FFs labeled *FR1, FR2,* and *FLLre* are used by compression unit for enabling subband memory for reads.

The two registers labeled *WDER* (write decomposition register) and *RDER* (read decomposition register) both are counter that count from 0 to *j-1*. These registers are initially designed to count from 0 to *jmax-1*, where *Jmax* is the maximum number of decomposition allowed. In a decomposition process, the required number of decompositions, *j* desired should be provided by loading *j* into a register. Moreover, the order of writing into the subband memory blocks are controlled by *WDER*, whereas the order of reading them by compression unit are controlled by the two registers labeled *RDER* and *RBER*.

To write subbands coefficients of the first level decomposition into subband memory, the DWT unit initially clears registers *SMSR1, MAR1, WDER*, and the flip-flop (FF) labeled *FLLre* to zero and sets the FFs labeled *Fw1* and *Fbwe* 1. *Fbwe* enables the decoder *dcodw* and since *WDER* is 0, the first output of the decoder labeled 0 is activated. Activation of this output signal enables subband memory blocks *HL1, HH1*, and *LH1* for write. The value in register *SMSR1* determines the bank number and the module number to be written in each enabled subband memory block. While register *MAR1* is used for addressing each location in the 3 selected modules. When all locations of the 3 modules are written, register *SMSR1* is incremented by one to select the next 3 modules, one from each enabled blocks. This process is repeated until all modules in the 3 enabled subband memory blocks are written. The DWT unit resets FF *Fw1* 0 and then informs the compression unit, say, by asserting a FF high. The compression unit responds by reading contents of the subband memory blocks *HL1, HH1*, and *LH1*, and compresses them independently.

Meanwhile, the DWT unit moves to the second level in the subband memory by incrementing register *WDER* and setting *Fw2* 1. This allows the DWT unit to write subbands coefficients of the second decomposition into the subband memory. Incrementing register *WDER* by one activates the second output of the decoder labeled *dcodw*. This output enables subband memory blocks labeled *HL2, HH2*, and *LH2* for write. In addition, registers *SMSR2* and *MAR2* are reset zero. Resetting *SMSR2* zero, selects the first bank in each one of the 3 enabled blocks and enables the first module in each selected bank for write. Register *MAR2* is used for addressing each location in the 3 enabled modules. The process of writing into these modules proceeds as that of the first level. When all modules in the 3 enabled subband memory blocks are written, the third level in the subband memory is enabled by incrementing *WDER* by one. This activates the third output of the decoder, which enables blocks *HL3*, *HH3*, and *LH3* for write. This process is continued until the last decomposition level is reached. When all subbands coefficients of the last decomposition are written, the DWT unit will inform again the compression unit. It will also reset *Fbwe* and *Fw2* zero to disable subband memory for writes, until it read by compression unit.

On the other hand, reading of subband memory by compression unit proceeds as follows. As soon as the compression unit receives the first signal from DWT unit, confirming that the first level decomposition is completed and its subbands coefficients are available in the subband memory blocks *HL1,HH1,* and *LH1,* the compression unit clears registers *RDER, RBER, SMSR1,*and *MAR1*to zero and sets *FF FR1 1.* Resetting *RDER* and *RBER* zero enable the subband memory block labeled *HL1* for read. While resetting *SMSR1* selects the first bank in block *HL1* and enables the first module in the bank. Then *MAR1* is used for addressing each location in the module for read. The next module is enabled by incrementing *SMSR1* by one. The compression unit continues in this fashion until all *HL1* modules are read. Then *RBER* is incremented by one to enable *HH1* for read and *SMSR1* and *MAR1* are reset zero to select the first bank and enable the first module in the bank. Then, reading of block *HH1* proceeds as that of *HL1.*

To enable block *LH1*, the compression unit increments again *RBER* by one and resets *SMSR1* and *MAR1* zero. When all modules in *LH1* are read and the second signal from DWT unit is received to confirm that all subband coefficients, starting from the second level decomposition, are available in their respective subband memory blocks, register *RDER* is incremented by one to enable the second decoder (*dcod2)* and *RBER* is reset zero to activate the first output of the decoder. In addition, *FR1* is reset zero and *FR2* is set 1. Activation of the first output of the second decoder enables block *HL2* for read. Then compression unit uses registers *SMSR2* and *MAR2* to read block *HL2* module-by-module as described in the first level. After *HL2* is read, *HH2* is enabled for read then *LH2.* The compression unit reads subband memory level-by-level and each level is read block-by-block and each block is read bank-by-bank and each bank is read module-by-module until it reaches the last subband memory block labeled LL_{jmax}. To read block LL_{jmax}, the compression unit sets FLLre 1 to enable this block for read and then uses registers *SMSR2* and *MAR2* to read its contents.

5.3.3 Subband memory architecture for higher scan methods

With first scan method, writing into each subband memory block takes place module-by-module. That means, only one module in each block will be enabled for write at a time. The second and the third scan methods require writing into 2 and 3 modules at time in each block, respectively. In general, the *ith* scan method requires writing into *i* modules in each subband memory block.

To see how this can take place consider, for example, the dataflow for the 2-parallel intermediate architecture shown in Table B.12. The dataflow table shows that the architecture yields 4 output coefficients every clock cycle, reference to clock $f_2/2$. The 3 output coefficients labeled HH0,0, HL0,0, and LH0,0 in Table B.12 should be stored in the first location of the first module in each subband memory blocks *HH1, HL1,* and *LH1*, respectively. The second output coefficients HH0,1, HL0,1, and LH0,1 should be stored in the first location of the second module in each subband memory blocks HH1, HL1, and LH1, respectively. The third output coefficients HH0,2, HL0,2, and LH0,2 should be stored in the first location of the third module in each subband memory blocks *HH1, HL1*, and *LH1*, respectively. The fourth output coefficients HH1,0, HL1,0, and LH1,0 should be stored in the second location of the first module in each subband memory blocks *HH1, HL1,* and *LH1*, respectively.

It is obvious, after the third output coefficients are stored, the process of storing coefficients returns to the first module in each block to repeat the process until the first 3 modules in each subband memory blocks *HH1, HL1*, and *LH1* are written. Similarly, the next 3 modules in each subband memory blocks *HH1, HL1,* and *LH1* are written and so on. When all modules in the subband memory blocks *HH1, HL1*, and *LH1* are written, the process moves to the second level of the subband memory blocks *HH2, HL2*, and *LH2* to store subbands coefficients of the second decomposition level. However, in order for the control to move effectively between 3 modules, the first module number ought to be remembered by the control. For this reason, register *XR* is added and is connected to register *SMSR* as shown in Figure 5.3.7.

Initially, registers *SMSR, MAR* and *XR* are reset 0. When *SMSR* is reset, *BS* enables the first bank in each subband memory blocks *HH1, HL1,* and *LH1*, while $\overline{UB}$ and MS enable the upper bank and the first module in each bank, respectively. This will allow the first 3 output coefficients HH0,0, HL0,0, and LH0,0 to be stored in the first location of each module in blocks *HH1, HL1,* and *LH1*, respectively, addressed by *MAR*. Then register *SMSR* is incremented by one to enable the second module in each subband memory blocks *HH1, HL1,* and *LH1*. This will allow the second output coefficients HH0,1, HL0,1, and LH0,1 to be stored in the first location of the second module in each subband memory blocks *HH1, HL1,* and *LH1*, respectively. To store the third output coefficients HH0,2, HL0,2, and LH0,2 in the first location of the third module in each block, register *SMSR* is again incremented by one.

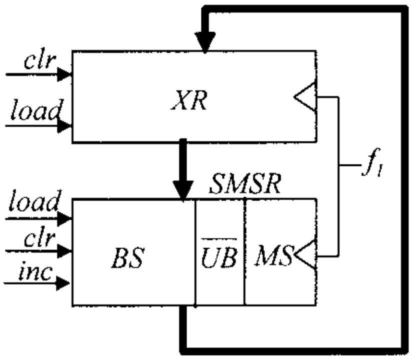


Figure 5.3.7 Incorporation of register XR

Since, the fourth output coefficients HH1,0, HL1,0, and LH1,0 should be stored in the second location of the first module in each subband memory blocks *HH1, HL1,* and *LH1*, respectively, register *XR*, which is 0, is loaded into *SMSR* while *MAR* is incremented by one to address the second location in each module. This process is repeated until the first 3 modules in each block are written. At that point, where run 2 begins, SMSR will be 2, indicating that the third module is the last module written in each block. To enable the fourth module in each block, register *SMSR* is incremented by one and the result is loaded into *XR* so that this module number can be remembered, while *MAR* is reset 0 to address the first location in each module. This will allow the first 3 output coefficients of run 2 to be stored in the first location of each module enabled in the subband memory blocks HH1, HL1, and LH1. Then, register *SMSR* is incremented by one to enable the fifth module in each block. When the first location of each module is written, register *SMSR* is incremented again by one to enable the sixth module in each block. When the first location of each module is written, register *MAR* is incremented by 1 and register *XR* is loaded into *SMSR* to enable again the fourth module in each block and the process repeats. When all modules in the first level are written, the subband memory blocks *HH2, HL2*, and *LH2*, in the second level, are enabled and writing into these blocks proceeds as in the first level.

A flowchart, which describes the control algorithm that can be used to control subband memory write operations, is shown in Figure 5.3.8. In the flowchart, the following 3 registers are used. Register *RN3* holds number of locations to be written in a module. Register *RM3* holds number of modules to be written in a subband memory block, while *RS* holds the scan method number. Thus, if DWT architecture is based on the third scan method, e.g., 3 is loaded into *RS* to indicate number of modules that will be considered for write in each subband memory block at a time. *Flast* is a *FF*, when it is set 1, indicates the last run.

The flowchart remains in state *S0* as long as the status input signal *wsub* is low. When *wsub* is asserted high, the process of storing subbands of the first decomposition level begins. As the flowchart moves from states *S0* to *S1* it resets registers *SMSR, MAR, WDER, XR*, and FF *Flast* 0, sets *FFs FW1* and *Fbwe* 1, loads *i* into *RS*, while number of modules and number of locations are loaded into *RM3* and *RN3*, respectively. In state *S1*, register *RS* is examined. As long as it is not 1, the loop consisting of states *S1* and *S2* is executed, during which write operations take place in the modules enabled in each subband memory blocks *HH1, HL1*, and *LH1*. When *RS* becomes 1, register *RN3* is examined. If *RN3* is not equal 1, the control moves to state *S3*. As the control moves from states *S3* to *S1*, register *MAR* is incremented and register *RN3* is decremented, while register *XR* and *i* are loaded into *SMSR* and *RS*, respectively. If *RN3* is 1, it indicates the last location is reached and the flowchart moves to state *S4*. As it moves from states *S1* to *S4*, it loads *SMSR* into *XR* and

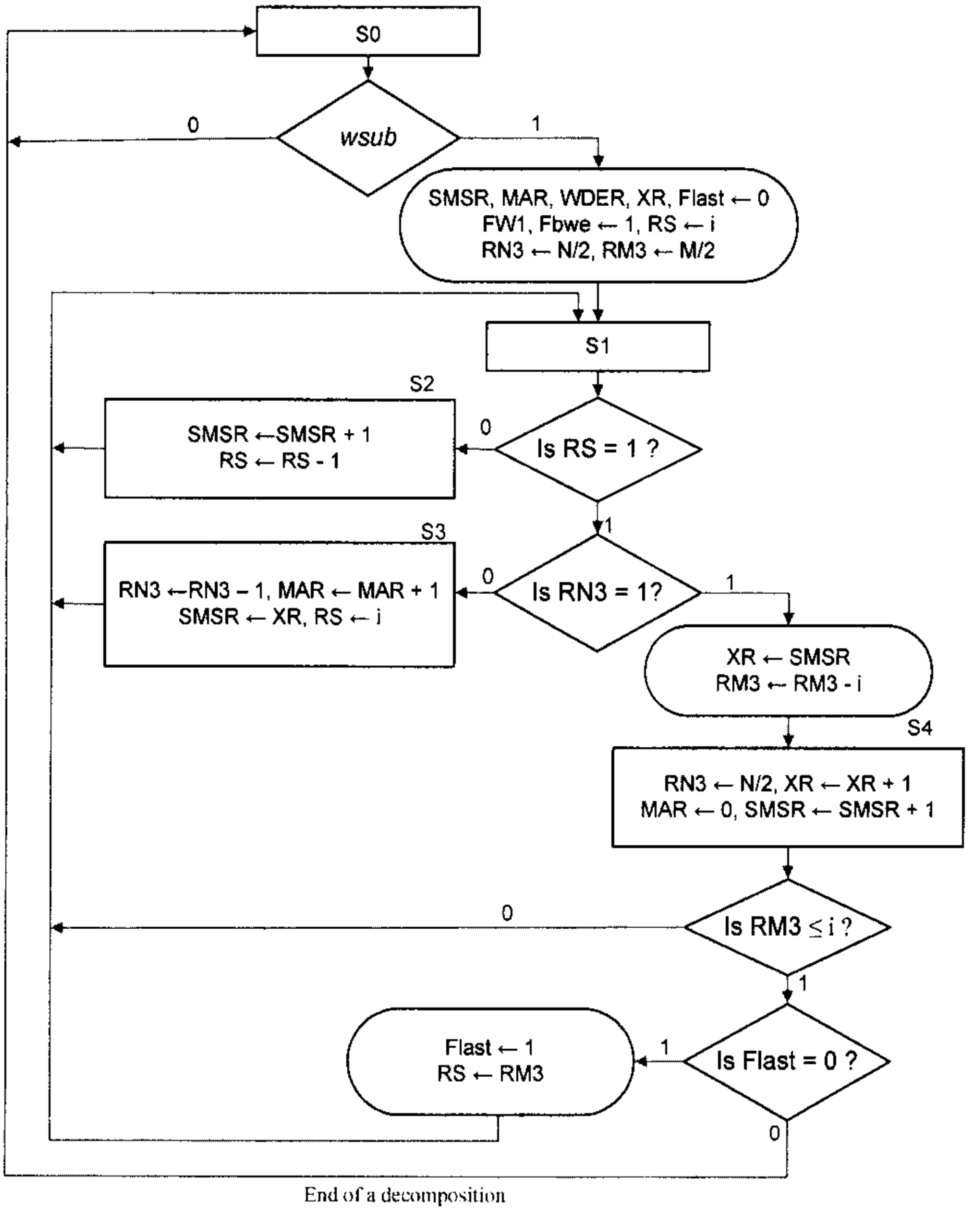


Figure 5.3.8 Flowchart for subband memory write control algorithm

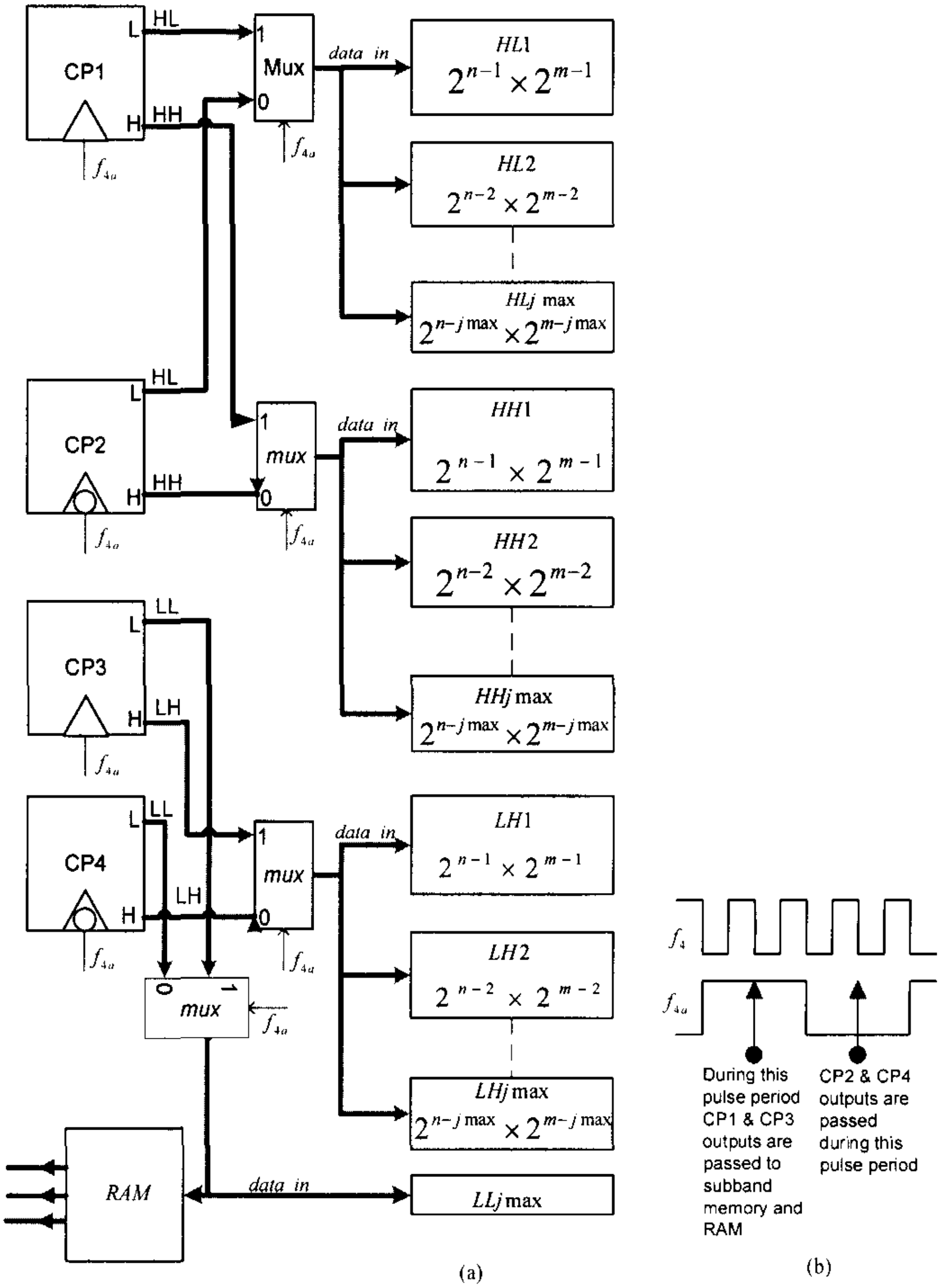
subtracts *i* from *RM3* to reflect number of modules that remain to be written in the subband memory blocks that are under consideration.

In state *S4*, a signal would be issued to reset *MAR* 0, to increment *SMSR* and *XR*, and to load *RN3* with number of locations, while register *RM3* is examined. If *RM3* > i, the flowchart moves to state *S1* to consider the next *i* modules in each subband memory block for write. But, if ($RM3 \leq i$), then the last run is reached and *RM3* contains number of modules that are remain in each subband memory block which will be considered for write in the last run. Number of modules that will be considered in the last run will be *i*, *i*-1, *i-2*...*or 1* depending on the image width *M*. For example, if the architecture is based on the third scan method , then number of modules that will be consider in the last run will be either 3, 2, or 1. In addition, if $RM3 \leq i$, the status of the next input is examined. If *Flast* is 0, then the control moves to state *S1* to begin storing the output coefficients that will be generated in the last run and as it moves to state *S1*, it set *Flast* 1 and loads *RM3* into *RS*. When *Flast* is 1, the flowchart returns to its initial state *S0* and remains in that state until activated for the second level decomposition. The algorithm given in Figure 5.3.8 is general and is intended to illustrate in a broad sense how subband memory is written. However, the algorithm can be modified to fit any specific architecture requirements.

5.4 Control Design for 4-parallel Architecture

In this section, to demonstrate that the controls for the architectures developed in chapter 4 and 5 are simple to design, the control algorithms for the 4-parallel architecture shown in Figure 4.2.7 including the LL-RAM and subband memory architectures will be developed. Control unit is responsible for issuing proper control signals, in respond to a clock pulse, to the components of the architecture where data processing take place.

Figure 5.4.1 (a) shows the interconnection between subband memory of Figure 5.3.1 and the 4-parallel pipelined architecture shown in Figure 4.2.7. The interconnection between the two entities is accomplished through four multiplexers, labeled *mux*. Furthermore, since CP1 and CP3, and CP2 and CP4 load into their output latches four new coefficients each time clock f_{4a} makes a positive and a



(a)

(b)

Figure 5.4.1 (a) Subband memory interconnections to 4-parallel (b) Control input signal waveform of the multiplexer labeled *mux*

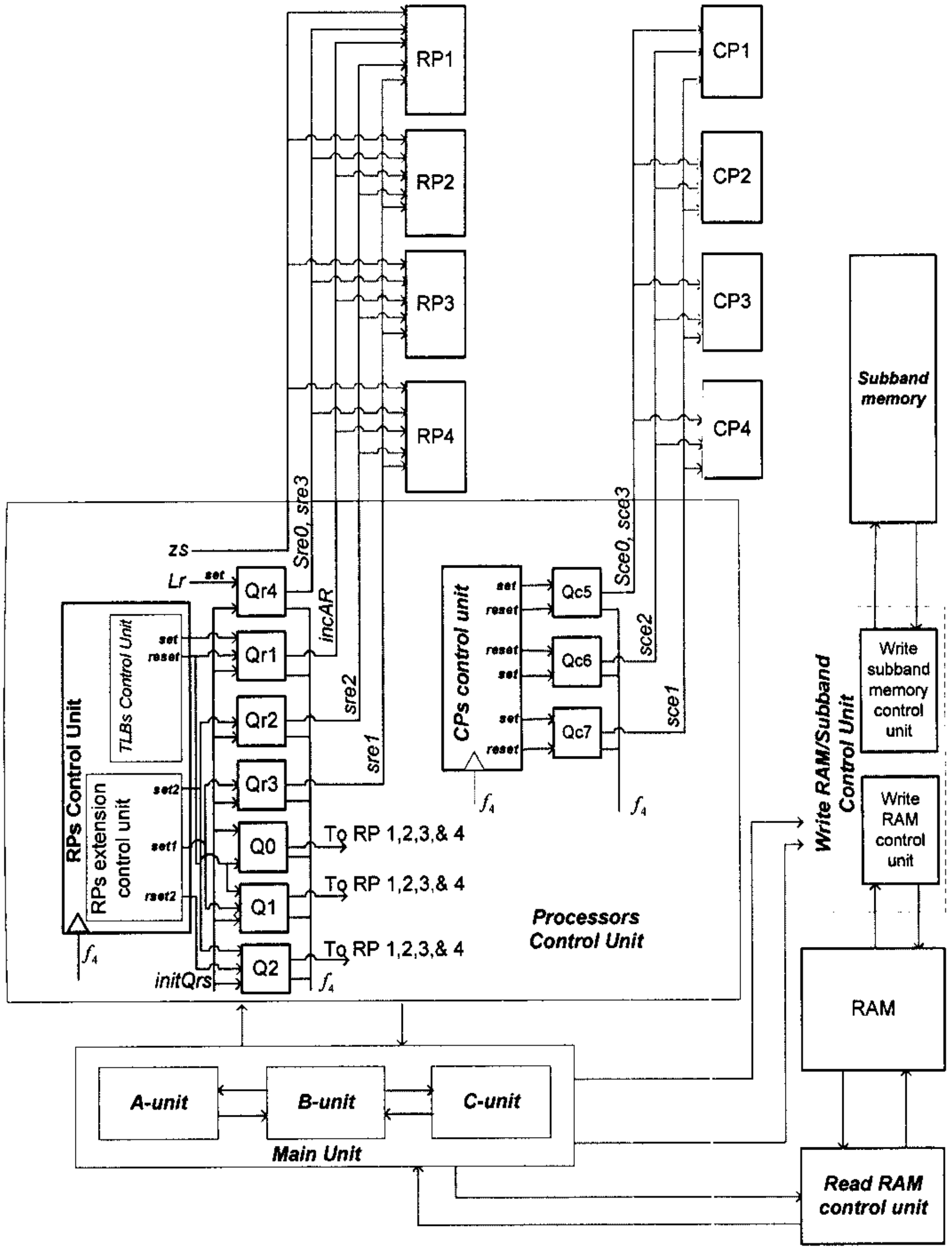


Figure 5.4.2 DWT Control Unit

negative transition, respectively; therefore, the clock f_{4a} is connected to the input control signal of the four multiplexers. When f_{4a} is high, the four multiplexers will pass the four output coefficients generated by CP1 and CP3 to subband memory and LL-RAM for storage. Otherwise, the four multiplexers will pass the output coefficients of CP2 and CP4 to subband memory and LL-RAM for storage, as illustrated in Figure 5.4.1 (b).

In Figure 5.4.2, which represent the overall DWT control unit, four control units have been identified and labeled main control unit, processors control unit, read RAM control unit, and write RAM/subband memory control unit. The main control unit consists of 3 units, *A-unit, B-unit,* and *C-unit.*

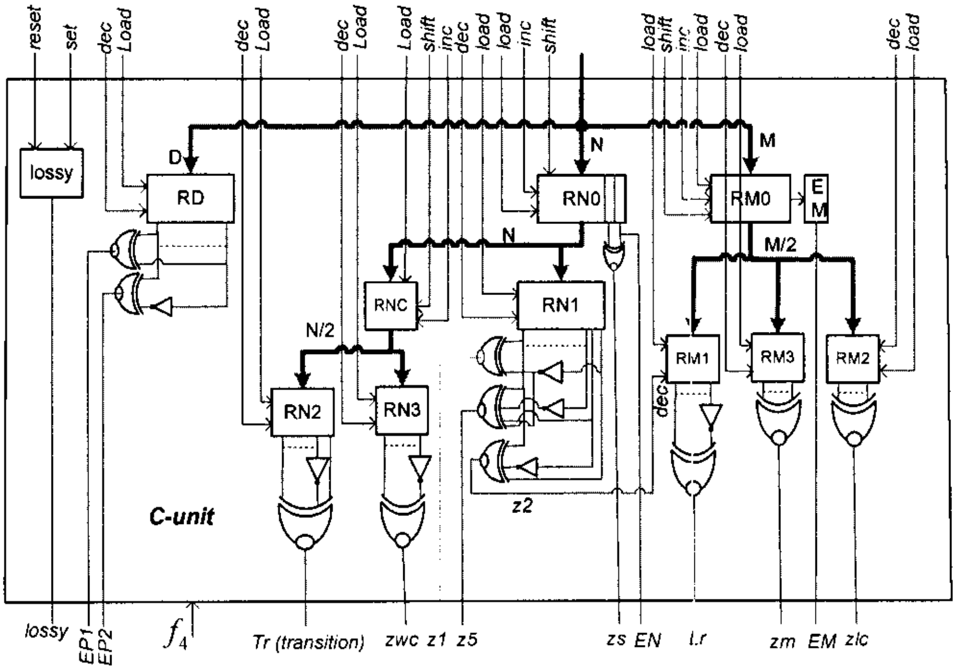
In the following, a description of each control unit function will be given along with its algorithmic state machine (ASM). The ASM is a special flowchart, which precisely specifies the control algorithm that can be used for deriving the hardware of the control.

5.4.1 Main Control Unit

a) C-unit

This unit is basically consists of various registers, as shown in Figure 5.4.3. These registers functions are to generate control signals, which will be used by all other control units as input control signals. At the start of a decomposition process, the height *(N)* and the width *(M)* of an image along with the desired number of decomposition levels *(J)* must be loaded into registers RN0, RM0, and RD, respectively. The loading of these registers should be handled by an entity other than the DWT unit, for example, microprocessor. Then DWT unit is activated by asserting the start signal of A-unit.

The signals labeled *EN* and *EM* in Figure 5.4.3 are examined by the control units to determine whether *N and M* are even or odd. In section 4.2.3, two cases where identified regarding storage of high coefficients. In the first case, if the two least significant bits of *N* are either 00 or 11, then the high coefficients should be stored in the TLBs of the RPs that generate them. In the second case, if the two least significant bits of *N* are either 01 or 10, then the high coefficients of RP1 should be stored in the TLB of RP3 and vice versa, while the high coefficients of RP2 should be stored in the TLB of RP4 and vice versa. Thus, the signal labeled *zs* is formed to



RN2 : holds number of operations in a column

RN3 : holds number of locations to be written in a module

RN1 : holds number of locations to be read from each module in a run

RM1 : holds number of runs (each run activates 3 modules)

RM2 : holds number of columns to be scheduled for CPs

RD : holds number of decomposition levels desired

RM3 : holds number of modules to be written in the RAM & in subband memory

Zwc : all locations in enabled modules are written

Lr : last run in a decomposition is reached

Zm : *last module is written*

Zlc : *the last operation in the last column is reached*

EP1 : End of decomposition process

Z1 : End of a run

EP2 : last decomposition level

lossy : is a FF, if zero, performs 5/3, otherwise, 9/7

Figure 5.4.3 C-unit

detect occurrence of these two cases. If zs is 1, it signifies occurrence of the first case, otherwise, the second case.

Figure 5.4.3 shows that contents of RN0 should be transferred to both registers RN1 and RNC. However, if RNC is odd, which can be determined by examining *EN*, it is first shifted to right (divided by 2) and then is incremented by one, otherwise, it shifted to right only. These operations are controlled by A-unit. The result is then loaded into two registers labeled RN2 and RN3. Register RN2 holds number of operations in a column when DWT is applied column wise by CPs and each operation requires 3 pixels or coefficients except the last operation, while register RN3 holds number of locations to be written in a module.

On the other hand, contents of the register labeled RM0 is examined by the *B-unit* to determine whether it is even or odd. If signal *EM* is 1, then RM0 is odd and it is shifted to right and then is incremented by one, otherwise, it is shifted only to right. The result is then transferred to the three registers labeled RM1, RM2, and RM3.

Registers RN1 and RM1 are used by the read RAM control unit. Register RM1 holds number of runs required in a level decomposition, where each run activates 3 modules for read except the last run. When the signal labeled *Lr* (last run) is asserted high it indicates that the run before the last has completed. On the other hand, register RN1 holds number of locations to be read from each module in a run. The signal labeled *z2*, which is generated by an XNOR gate attached to RN1, is shown connected to RM1's signal labeled *dec* (decrement). When register RN1 is counted down to 2, signal *z2* is asserted high, which in the next clock cycle will decrement register RM1 by one to reflect number of runs remaining. Signal *z1* is similar to *z2*, but it is asserted high when RN1 is counted down to 1 and it indicates a run has completed. Then the next run can be initiated by reloading register RN1 from RN0. Signal *z5* is asserted high when RN1 is counted down to 5. This signal will be made clear when TLB control unit is introduced later.

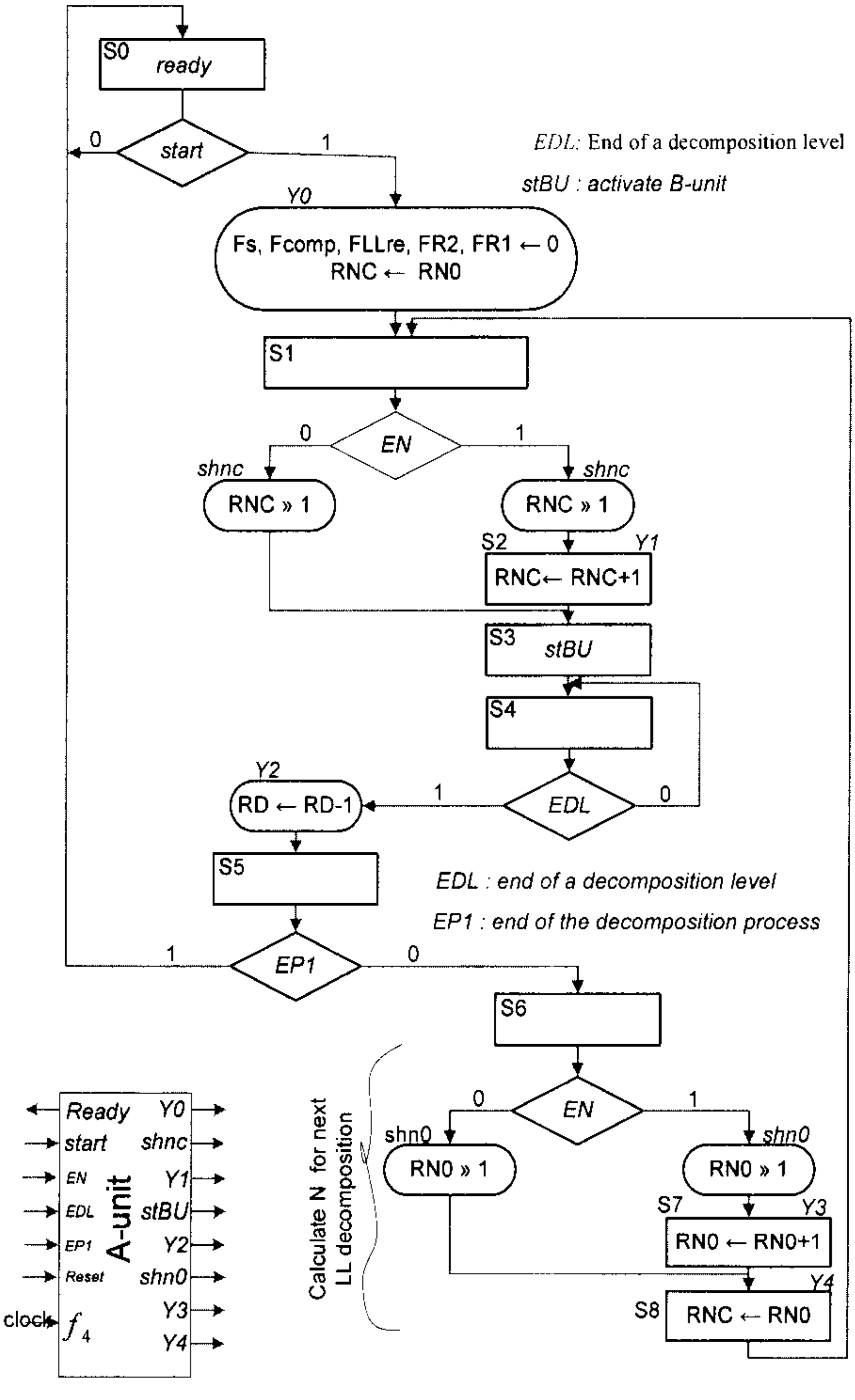
The registers labeled RM3 and RN3 are used by both write control units of the LL- RAM and subband memory to control write operations in the two memories. Register RM3 function is to hold number of modules to be written in the RAM and in each subband memory block enabled for write in a level decomposition. When RM3 is counted down to zero, signal *zm* is asserted high to indicate all modules for this decomposition have been written and the next decomposition level can be initiated. On the other hand, register RN3 function is to hold number of locations to be written in a module. When all locations in a module are written, the signal labeled *zwc* is asserted high and RN3 can be then reloaded from RNC for the next module to be written. This process is repeated until all modules in a decomposition level are written. The occurrence of this event will be signified by assertion of signal *zm*.

The registers labeled RM2 and RN2 are used by the CPs control unit, which is part of the processors control unit. Register RM2 holds number of columns, in L and H decompositions, to be scheduled for CPs. When all columns in L and H decompositions are scheduled, the signal labeled *zlc* is asserted high to indicate that this is the last cycle where the coefficients of the last operation in the last column will be transferred to CPs input latches. On the other hand, register RN2 holds number of operations in a column, where each operation requires 3 coefficients except the last one. Each time an operation is scheduled, RN2 is decremented by one. When all operations in a column of L and a column of H are scheduled, signal *Tr* (transition) is asserted high. That is when RN2 is counted down to 2. Assertion of signal *Tr* indicates that in the clock cycle after next, the last operation in a column, before a transition is made to the next column, will be scheduled.

The final register in *C-unit* is the register labeled RD. Register RD holds number of decomposition levels *(J)* desired for an *(NxM)*-image decomposition. Each time a decomposition level is completed, RD is decremented by one. When all *J* levels of decomposition are completed, that is, when RD is counted down to zero, the signal labeled *EP1* is asserted high signifying end of the process. The second signal labeled *EP2* is asserted high when RD is counted down to 1 to indicate this is the last decomposition.

b) A-unit

The ASM flowchart and the block diagram for *A-unit* are shown in Figures 5.4.4 (a) and (b), respectively. The ASM chart describes the control function of the *A-unit*,



(b)

(a)

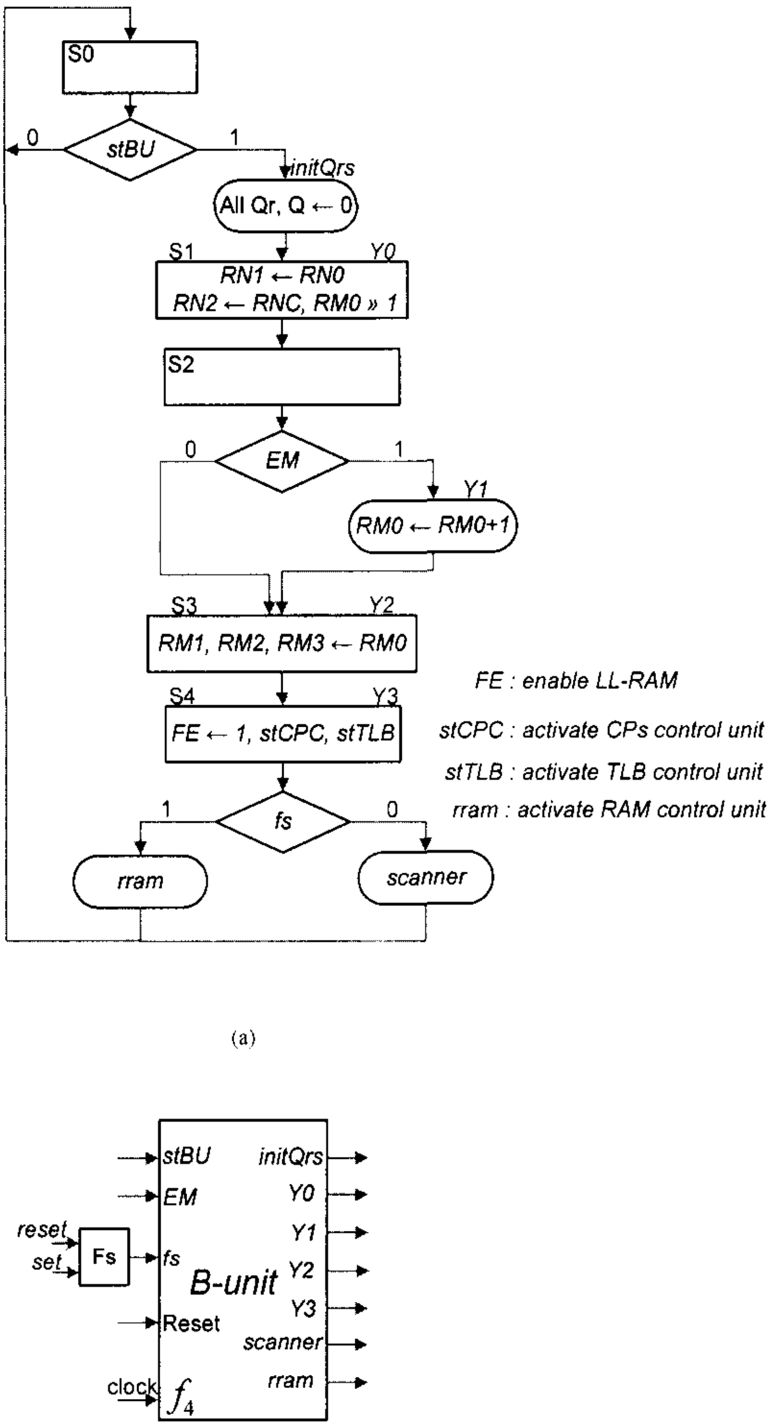
Figure 5.4.4 (a) ASM flowchart for A-unit (b) Block diagram while the block diagram displays the input and output control signals. As soon as registers RN0, RM0, and RD are loaded with *N, M,* and *J*, respectively, *A-unit* is activated by asserting the *start* signal. As long as the *start* signal is low the A-unit remains in the initial state S0. The activation of *A-unit* starts the decomposition process.

When *start* signal is asserted high, the *A-unit* first initializes several registers and flip-flops (FFs) by asserting its output signal labeled *Y0* and then it moves to state S1 at the clock event. In state S1, it examines signal *EN to* determine whether register RNC is even or odd. If *EN* is 1, then RNC is odd and the ASM asserts the conditional output signal labeled *shnc*. At the clock event, RNC is shifted to the right. In state S2, RNC is incremented by one. If *EN* is 0, register RNC is shifted to the right only. Register RNC now holds the number that will be loaded into register RN2 and RN3. In state S3, the *B-unit* is activated by asserting signal *stBU* high. In state S4, signal *EDL* (end of a decomposition level) is examined. If *EDL* is 0, the ASM remains in state S4 until *EDL* is 1. When *EDL* becomes 1, register RD is decremented by one and the ASM moves to state S5. In state S5, the status input signal labeled *EP1* is examined. If *EP1* is 1, then this indicates the decomposition process has completed and the control returns to its initial state S0 at the clock event. Otherwise, the control executes the loop consisting of states S6, S7, S8, and S1. Inside the loop a new value for RN0 is computed. This value gives the height of the LL-image to be decomposed next.

c) B-unit

The *B-unit* is represented by the ASM flowchart and the block diagram shown in Figures 5.4.5 (a) and (b), respectively. When *B-unit* is activated, by asserting its input signal labeled *stBU* high, it immediately initializes all FFs labeled Qr, in the processors control unit, to zero by asserting the output signal labeled *initQrs* and then moves to state S1. In state S1, registers RN2 and RN1 are loaded from RNC and RN0, respectively, while register RM0 is shifted to the right one position.

In state S2, a decision is made based on signal *EM*, the least significant bit of RM0. IF *EM* is 1, RM0 is incremented by one; otherwise, RM0 is left unchanged. In state S3, the new value in register RM0 is loaded into registers RM2, RM1, and RM3.



FE : enable LL-RAM

stCPC : activate CPs control unit

stTLB : activate TLB control unit

rram : activate RAM control unit

(a)

(b)

Figure 5.4.5 (a) ASM chart for B-unit (b) Block diagram

In state S4, the FF FE is set 1 to enable the LL-RAM for read and write. The RAM is enabled when signal *E* of *dcodms or dcodbs* are high. In addition, the TLB control unit and the CPs control unit are activated by asserting the input signals *stTLB* and *stCPC*, respectively. Furthermore, while the ASM is in state S4, signal *fs* is examined. If *fs* is 0, the scanner control unit is activated to scan the original image pixels; otherwise, read RAM control unit is activated to scan the LL-RAM.

In a decomposition process, the original image pixels are scanned first through an image scanner. Thus, in the first level decomposition the scanner control unit is activated to scan the original image pixels. Then in all subsequence decompositions, read LL-RAM control unit is activated. This process is controlled by signal *fs* of FF Fs. First, Fs is cleared to zero by *A-unit* and then examined by B-unit in state S4. The scanner control unit sets Fs 1 at the end of the scan to allow in all subsequence decompositions the LL-RAM to be scanned. Signal *fs* can also be used to control the operations of the multiplexers that would be needed in Figure 4.2.7 to select between passing the scanner or the LL-RAM data. If signal *fs* is 0, the multiplexers should pass to RPs the pixels that will be scanned by the scanner, otherwise, should pass data that will be read from the LL-RAM.

5.4.2 Processors Control Unit

The processors control unit consists of two control units, the RPs control unit and the CPs control unit, which are in charge of issuing control signals to RPs and CPs, respectively. The RPs control unit generates the following signals labeled *zs, sre0, sre3, sre1, sre2,* and *incAR* for the RPs. These signals are generated by the RPs control unit by setting or resetting each of the FFs labeled Qr0, Qr1, Qr2, and Qra shown in Figure 5.4.2. These signals are then transferred to the first stages of the RPs and loaded into the latches labeled CST (control signal latches). These latches then carry these signals from stage-to-stage. Each time a stage is reached; signals that are used in that stage can be dropped from the *CST* and the rest are carried on until the last stage is reached. These signals are used in both 5/3 and 9/7 processors. For example, signal *incAR* which is used in stage 2 of the 5/3 is also used in stages 2 and 5 of the 9/7. This is also true for other control signals. Thus, the control developed here can be used in both 5/3 and 9/7 architectures. Similarly, the CPs control unit generates four extension signals labeled *sce0, sce3, sce2,* and *sce1* by setting or resetting each of the FFs labeled Qc5, Qc6, and Qc7 shown in Figure 5.4.2.

a) The RPs Control Unit

The RPs control unit is further divided into two units, the TLB control unit and the extension control unit.

i) The TLB Control Unit

The TLB control unit is in charge of the reads and writes operations that take place in the 4 RPs' TLBs. The control unit generates the control signal *incar* (increment address register) for both *TLBARa* and *TLBARb* registers shown in Figure 4.2.9. Both *TLBARs* are *(n-2)*-bit counters.

The ASM chart, which represents the control algorithm of the TLB control unit, is shown in Figure 5.4.6. The control unit is activated when its status input signal *stTLB* is asserted high by *B-unit*. Then at the clock event, the ASM moves to state S1. In state S1, FF FEXR is set 0 and signals ETLB, sa12, and sa34 are set 1, while a decision is made based on the input signal labeled zs. If zs is 1, the control takes the path labeled case1 and in every clock cycle each location of a TLB is read in the first half cycle and written in the second half using only *TLBARa* as address register. But if zs is 0, the control takes the path labeled case2 and read and write operations take place according to Table B.11.

As explained in chapter 4, signal *zs* will be 1, when the two least significant bits of *N* are either 00 or 11, which implies that the high coefficients of stage1 will be stored in the TLB of the RP that will generate them, starting from the TLB of RP1. This would require FF Qra, which drives signal *incar* of each *TLBARa* in the 4 RPs shown Figure 4.2.9, to be set 1 a clock cycle before external memory scanning begins, as shown in Figure 5.4.6 (a). In state S2, where scanning of the external memory begins, the extension control unit is activated by asserting signal *stEX* high. When the ASM moves to state S3, the first three pixels and content of Qra are loaded into the three RP1 latches and CSTa, respectively.

In state S3, the control examines signal z5 and will continue executing the loop

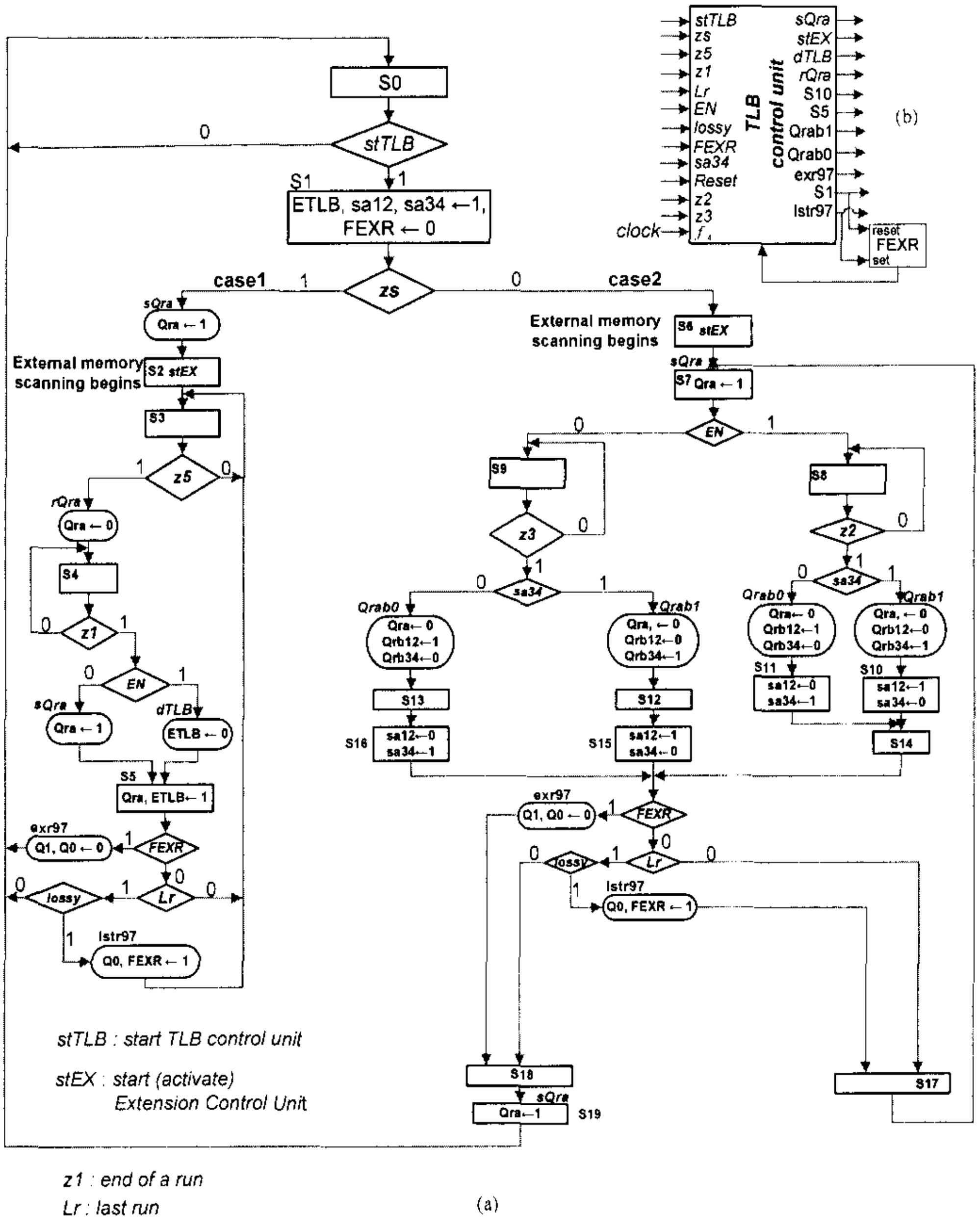


Figure 5.4.6 (a) ASM flowchart for TLB control unit (b) The block diagram

consisting only of S3 as long as z5 is 0. Each time this loop is executed three pixels and Qra are loaded into one of the RP latches until z5 is asserted high. Assertion of z5 allows Qra to insert zero in each of the last 4 operations that will be scheduled for the 4 RPs. The insertion of zeros occurs while the control is in state S4. These zero values of signal *incar* are necessary to reset register TLBAR of each TLB zero so that it addresses the first location at the start of the next run. The control remains in state S4 until *z1* becomes 1. When *z1* is 1, the control examines signal *EN*. If *EN* is 1, then N is odd and the external memory will not be scanned in the next cycle.

Therefore, the control sets signal *ETLB* 0 to disable TLB so that read and write can not take place during the next cycle and then moves to state S5. But, if *EN* is 0, the control sets FF Qra 1, which asserts signal *incar* high, and then moves to state S5.

In state S5, the control sets Qra 1 and examines FF FEXR and signal *Lr* (last run). If both are 0, then the next run is initiated by executing the loop consisting of states S3, S4, and S5. This loop usually will be executed for several times and each time it executed, a new run will be initiated until signal *Lr* becomes 1. Signal *Lr* will be 1 only when last run is initiated. When signal *Lr* becomes 1, signal *lossy* is examined. If *lossy* is 0, the operation is 5/3 last run and the control returns to its initial state S0 and remains in that state until activated. Otherwise, the operation is 9/7, which requires extra run, and the control set both FFs Q0 and FEXR 1 and moves to state S3 to initiate the last run. When the control reaches state S5 again, it examines FEXR. At this time FEXR should be 1 and the control sets both FFs Q1 and Q0 0, as required by Table B.5 (a), to initiate the extra run. Then the control moves to its initial state S0.

On the other hand, when the two least significant bits of N are either 01 or 10, signal zs becomes 0 and the control takes the path labeled case2 to state S6. When this path is taken, high coefficients generated by stage 1 of each RP will be stored according to Eq(4.3) starting from TLB of RP3. Therefore, setting of Qra is delayed until state S7.

In state S6, where scanning of the external memory begins, the extension control unit is activated by asserting signal *stEX* high. When the control moves to state S7, the first 3 pixels scanned from the external memory are loaded into RP1 latches. In state S7, Qra is set 1 and signal EN is examined to determine whether N is even or odd. If N is 1, then N is odd and the control moves to state S8, where it examines signal z2. As long as z2 is 0, the control executes the loop consisting only of S8. Signal z2 will be 1 when register RN1 is counted down to 2 by read RAM control unit and it indicates that in the next cycle the last operation of the current run will be scheduled for computation. When z2 becomes 1, the control examines signal sa34. According to Table B.11, signal sa34 will alternate between 1 and 0 values. Therefore it has been used here to indicate whether the current run sequence is even or odd. Signal sa34 will be 1 when a run sequence is odd and it will be 0 when the sequence is even. Thus, at the end of the first run, sa34 will be 1 and the conditional output signal *Qrab1* will be asserted high and at the end of the second run, it will be 0 to assert signal *Qrab0* high and so on. In both cases, *Qrab0* and *Qrab1* set FFs Qra, Qrb12, and Qrb34 according to Table B.11 so that *TLBARa* and *TLBARb* of each RP address the first location in the TLB each time a transition to a new run is made. FF Qrb12 drives signal *incar* of both *TLBAR1b* and *TLBAR2b* of RP1 and RP2, whereas, FF Qrb34 drives signal *incar* of both *TLBAR3b* and *TLBAR4b* in RP3 and RP4 shown in Figure 4.2.9. FF Qra drives signal *incar* of all *TLBARa* of the 4 RPs.

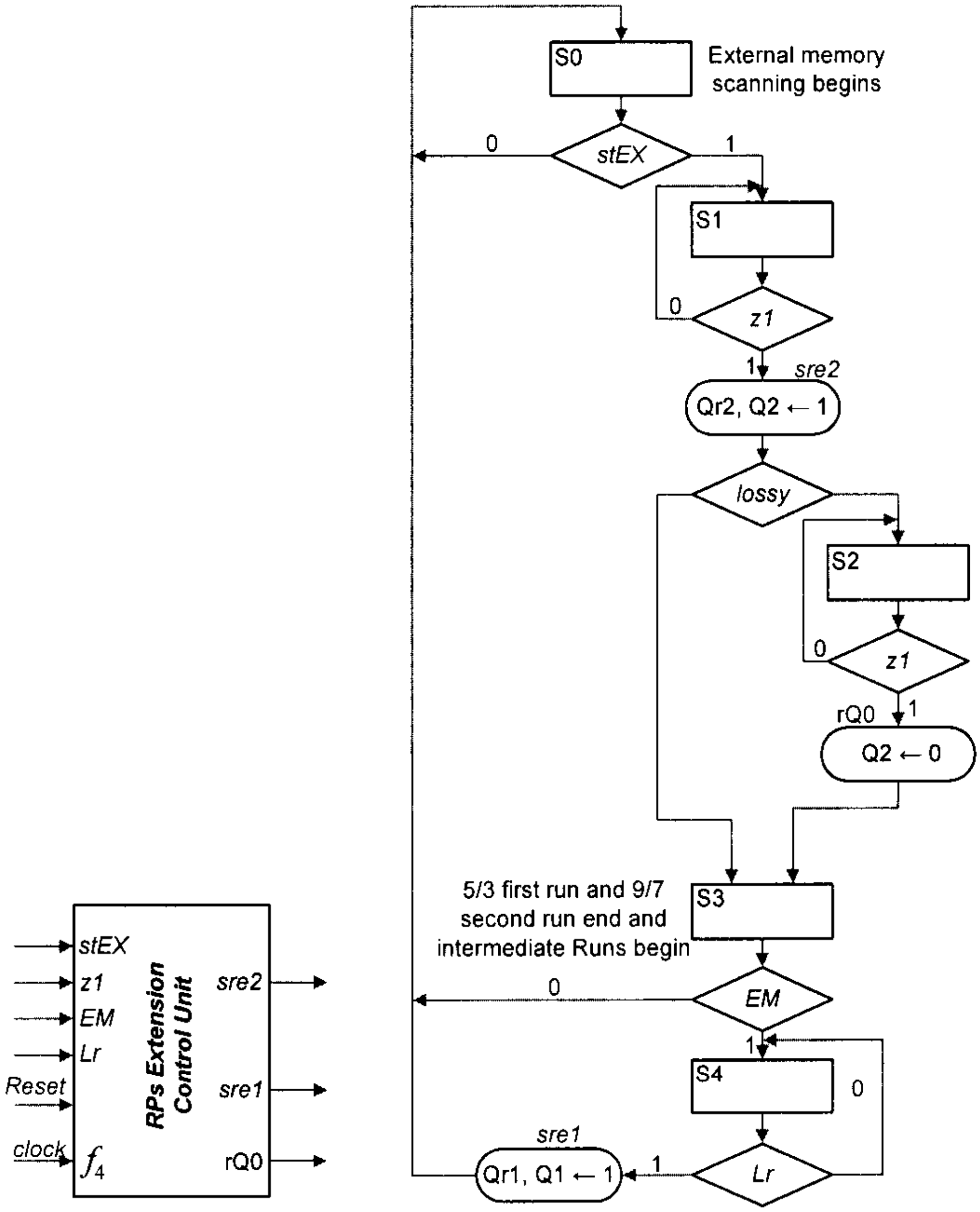
In states S10 and S11, signals sa12 and sa34 are also set according to Table B.11. State S14 is parallel to state S5 when the control takes the path labeled case1. Thus, every thing said there is also true here.

On the other hand, if EN is 0, then N is even and the control moves to state S9 where it examines signal z3. As long as z3 is 0, the control executes the loop consisting only of S9, until z3 is 1. Signal z3 will be 1 when register RN1 is counted down to 3 and it indicates that in the next two clock cycles, the last two operations of the current run will be scheduled and a new run then can be initiated. From this point on every thing that has been said when the control takes the path $EN = 1$ is also true for $EN = 0$.

ii) The Extension Control Unit

The extension control unit controls the operation of the two extension multiplexers found in stage 3 of the four 5/3 RPs and stages 3 and 7 of the four 9/7 RPs, through the two signals labeled *sre1* and *sre2*. The extension control unit generates these two signals by setting or resetting each of the two FFs labeled Qr1 and Qr2 in Figure 5.4.2.

The ASM chart for the extension control unit is shown in Figure 5.4.7 (a) and the control block diagram is shown in Figure 5.4.7 (b). The TLB control unit activates, by asserting its output signal *stEX* (start extension), the extension control unit in the clock cycle where external memory scan begins. At the clock event, the ASM moves from states S0 to S1. In state S1, the ASM examines signal *z1* and remains in that state as long as *z1* is 0. During this period where the first run takes place, Qr2 and Qr1 are left unchanged (retain zero values). The reason for this is that the first run requires the two multiplexers to pass in each clock cycle the current


stEX
z1
EM
Lr
Reset
clock
f_4
RPs Extension Control Unit
sre2
sre1
rQ0
S0
External memory scanning begins
stEX
0
1
S1
z1
0
1
sre2
Qr2, Q2 ← 1
lossy
S2
z1
0
rQ0
1
Q2 ← 0
S3
5/3 first run and 9/7 second run end and intermediate Runs begin
EM
0
1
S4
0
Lr
1
sre1
Qr1, Q1 ← 1

(b) (a)

Figure 5.4.7 (a) ASM flowchart for Extension Control Unit (b) The block diagram.

high coefficient required in the calculation of the current low coefficient and inserting zeros by Qr2 and Qr1 during this period will guarantee the proper operation of the multiplexers. When *z1* becomes 1, the control asserts its conditional output signal *sre2* to set Qr2 and Q2 1, as required by Table B.5 (b) for run2 of the 9/7, and examines signal *lossy*. If *lossy* is 0, the control moves to state S3 to initiate run2 of the 5/3, otherwise, it moves to state S2 to initiate run2 of the 9/7. In state S2, the control examines signal *z1* again and remains in that state until *z1* becomes 1, which indicates end of run2. As the control moves from states S2 to S3 it set FF Q2 0, as required by Table B.5 (b) for run3 and all subsequent runs of the 9/7.

In state S3, the first run of the 5/3 or the second run of the 9/7 end and the intermediate runs begin. Intermediate refers to the runs that are between the first and last run. During intermediate runs the two multiplexers are required to pass both the current high coefficient and the previous high coefficient read from TLB. Thus, for the multiplexers to be able to accomplish this task, Qr2 is set 1 while Qr1 is left unchanged (zero) during the whole intermediate period. In addition, in state S3, a decision is made based on signal *EM*, the least significant bit of register RM0, to determine whether the width *M* of the image is even or odd. If *EM* is 0, then *M* is even and the control returns to its initial state S0, since, as in the intermediate runs, even *M* requires Qr2 and Qr1 to be set 1 and 0, respectively, in the last run.

On the other hand, if *EM* is 1, then *M* is odd and the last run would require both Qr2 and Qr1 to be 1. Therefore, in state S4, the ASM waits in a loop controlled by *Lr* until the last run is reached. The last run is reached when *Lr* equals 1. Then, the ASM sets Qr1 and Q1 1 and returns to the initial state S0.

Finally, note that the output of the XNOR gate attached to register RN0 will generate the control signal *zs*, whereas signals *sre0* and *sre3* will be obtained by directly connecting signal *set* of Qr0 to signal *Lr*, as indicated in Figure 5.4.2.

b) The CPs Control Unit

The CPs control unit is in charge of issuing the four extension signals labeled *sce0, sce3, sce2,* and *sce1* that control the operations of the extension multiplexers in the four pipelined CPs. The CPs control unit generates these signals by setting each of the FFs labeled Qc5, Qc6, and Qc7 in Figure 5.4.2 either 1 or 0. According to Tables 3.3

and 3.4, since CPs compute DWT column-by-column, Qc5 which drives both signals sce0 and sce3 should be set to 1 every time the last operation in a column is scheduled for execution; otherwise, it remains at zero. On other hand, the two signals *sce1* and *sce2,* which control the two multiplexers in stage 3 of the 5/3 and stages 3 and 7 of the 9/7 processors, according to Tables 3.3 and 3.4, should be set as follows. Every time the first operation in a column is scheduled, both Qc6 and Qc7 should be set zero. All operations between the first and last operations in a column require Qc6 and Qc7 to be set 1 and 0, respectively. The last operation in each column requires Qc6 and Qc7 to be set 1 if the column length is odd, otherwise, Qc6 and Qc7 are set 1 and 0, respectively.

The cycle number (C1) at which the first input data are loaded into both CP1 and CP3 latches for both 5/3 and 9/7 is given by Eq (4.4). For 5/3 C1 is 19, since its RPs are pipelined into 4 stages, whereas C1 is 35 for 9/7, since its RPs are pipelined into 8 stages. In order to detect occurrence of this event, register RC is added to the CPs control unit as shown in Figure 5.4.8 (b). Register RC is a down counter with control signals *set* and *dec* (decrement). Initially, RC is set to 18 or 34 by asserting signal *set* high. Register RC then is decremented by one every clock cycle starting from the cycle where scanning of external memory begins. When RC becomes 0, it sets signal *zc* high to indicate that the pulse ending this cycle will load CP1 and CP3 latches with data for the first time.

The ASM chart for the CPs control unit and its block diagram are shown in Figure 5.4.8, respectively. The CPs control unit is activated when its input signal *stCPC* is asserted high by the TLB control unit. As the ASM moves from states S0 to S1, register RC is set to its initial value. In state S1, FFs Qc5, Qc6, and Qc7 are set 0. In state S2, where scanning of external memory begins, register RC is decremented by one.

In state S3, the ASM executes the loop consisting only of state S3 and controlled by signal *zc*. Each time this loop is executed, RC is decremented by one. When *zc* is 1, the control exits the loop and moves to state S4. As the control moves from states S3 to state S4, it activates the write subband memory control unit by asserting the output control signal labeled *wsub* and checks the input signal *EP2*. If *EP2* is 0, the

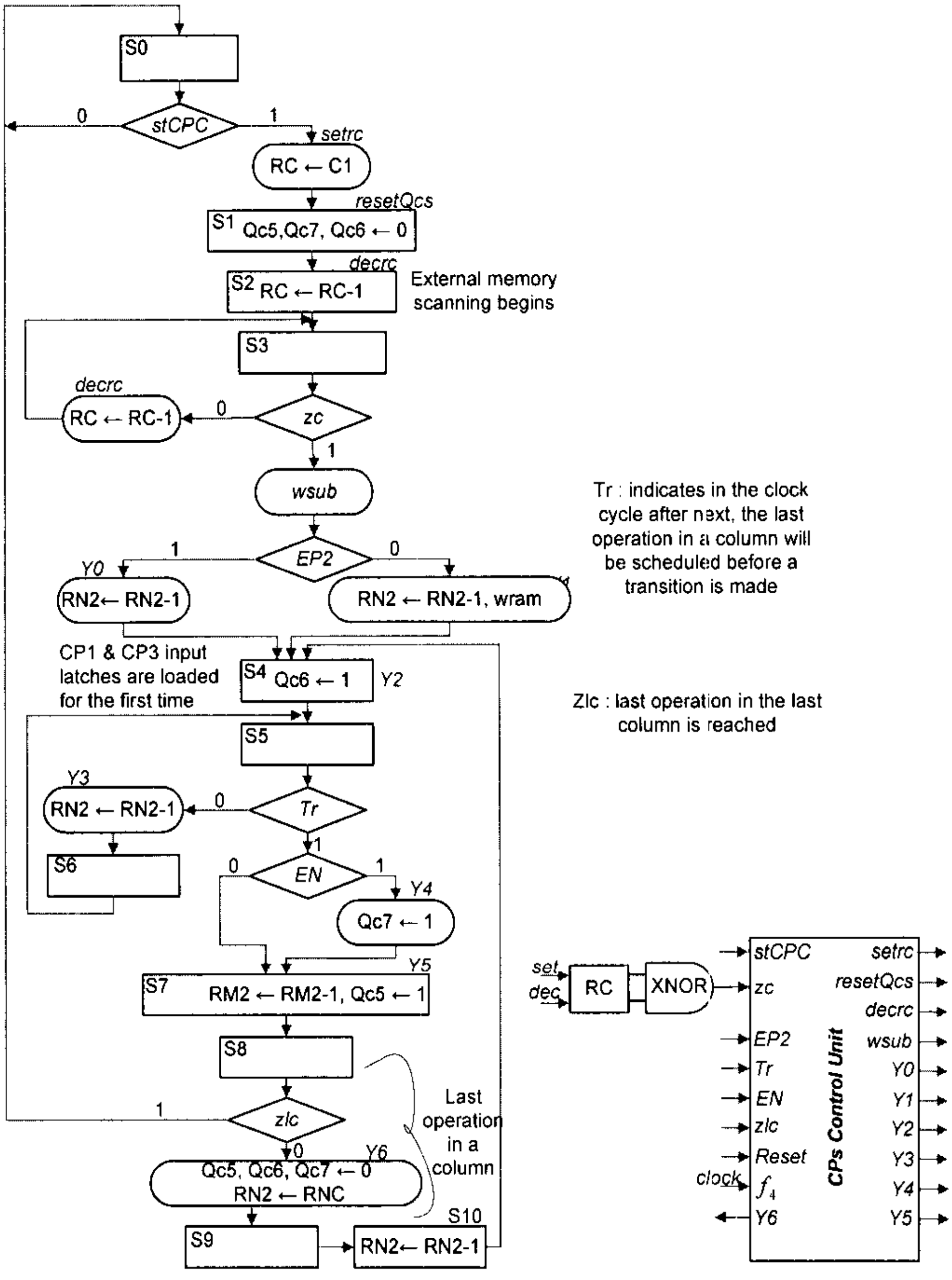


Figure 5.4.8 (a) ASM flowchart for CPs Control Unit (b) The block diagram.

ASM asserts its conditional output signal labeled *Y1* to activate the write RAM control unit and decrement register RN2 by one. The control will execute this path and activate the write LL-RAM control unit in all decomposition levels except in the last level decomposition. The reason is that, the LL-subband of the last level decomposition should be stored in the subband memory block labeled LL_{jmax}, not in the LL-RAM. When *EP2* becomes 1, it indicates that the last level decomposition is in process.

In addition, note that when the ASM makes a transition from states S3 to S4, CP1 and CP3 latches will be loaded for the first time with high and low coefficients of the first operations, respectively. In state S4, Qc6 is set 1, since all operations between the first and last operations in a column, as explained before, require Qc6 and Qc7 to be set 1 and 0, respectively.

In state S5, a decision is made based on signal *Tr*, which is the output of the XNOR gate attached to register RN2. As long as, *Tr* is 0, the loop consisting of states S5 and S6 is executed and register RN2, which hold number of operations in a column, is decremented by one to reflect number of operations left. Register RN2 is decremented each time a high and a low operation are scheduled from H and L decompositions, respectively. Note that, the actual scheduling of operations is done internally by clock *f4a*, as indicated in the architecture shown in Figure 4.2.7, and during execution of the above loop. However, all operations scheduled for CPs during this loop execution are that between the first and last operation in a column.

Signal *Tr* becomes 1 when RN2 is decremented to 2. When *Tr* is 1, the decision box with input signal *EN* is examined to determine whether *N* is even or odd. If *EN* is 1, then *N* is odd and Qc7 is set 1 in order to satisfy the requirement that both Qc7 and Qc6 must be1 in the last operation. Otherwise, Qc7 is left unchanged. Then the control moves to state S7.

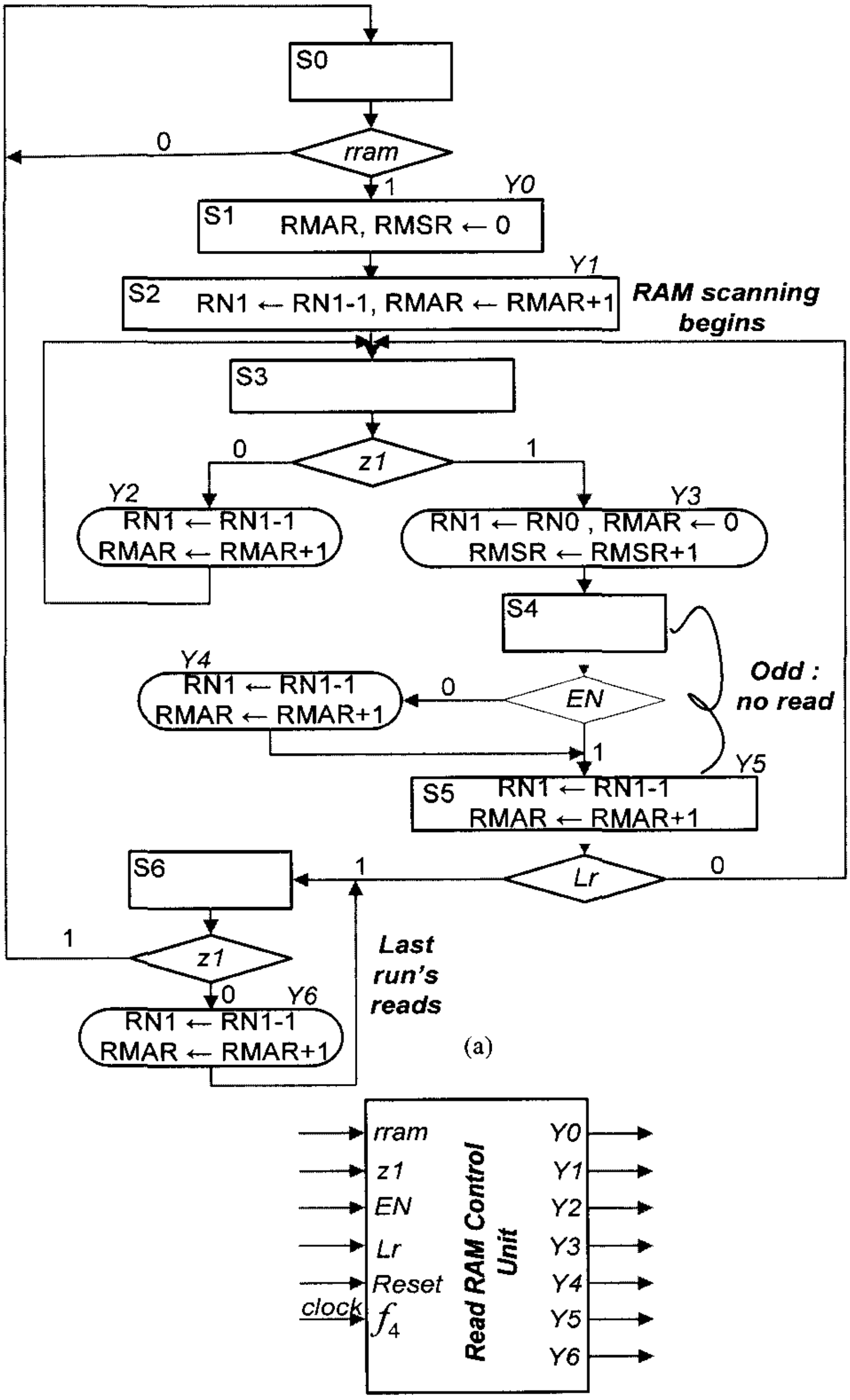
In state S7, the ASM asserts the output signal labeled *Y5*. This output signal decrements register RM2, which holds number of column to be scheduled for CPs, by one and sets Qc5 1. Setting Qc5 1 for the last operation in a column, which will be scheduled in the next state (S8), will allow the extension multiplexers controlled by signals *sce0* and *sce3* to pass data of the bus connected to the input of the extension multiplexers labeled 1 instead of 0 to *Rt2* as a third input, as required when *N* is even.

In state S8, where the last operation in a column is scheduled for execution, a decision is made based on signal *zlc,* which is the output of the XNOR gate attached to register RM2. If *zlc* is 1, it indicates that all columns in L and H decompositions have been scheduled and the control returns to its initial state S0. On the other hand, if *zlc* is 0, the control moves to state S9 to initiate processing of the next column. As the control moves from states S8 to S9, it loads again register RN2 and clears FFs Qc5, Qc6 and Qc7 to zero by asserting its conditional output signal labeled *Y6.* When the control moves from S10 to S4 it loads coefficients of the first operation of the next column in each H and L decomposition into CP1 and CP3 or CP2 and CP4 latches.

5.4.3 Read LL-RAM Control Unit

Read LL-RAM control unit is responsible for reading LL-RAM memory according to the scan method shown in Figure 3.5.1. Two control algorithms (or ASM charts) will be developed, one for the RAM architecture designed using modules shown in Figure 5.2.2 and the other for the RAM architecture designed using banks shown in Figure 5.2.5. Remember, the LL-RAM architecture is designed to allow both read and write to take place in the same clock cycle. Read takes place in the first half cycle and write in the second half cycle.

The ASM chart for read RAM control unit and its block diagram that controls the read operations of the RAM architecture shown in Figure 5.2.2 are given in Figures 5.4.9 The ASM chart of the control unit is activated when its input signal *rram* is asserted high by B-unit. As a result, the control moves from states S0 to S1. In state S1, both registers *RMAR* (read module address register) and *RMSR* (read model select register) are set zero. Register *RMSR* enables the first 3 modules for read, while register *RMAR* points to the first location in each module. Then, the control moves unconditionally to state S2, where the process of scanning the RAM begins. When the control moves from states S2 to S3 three pixels are scanned, one from each module, and then are loaded into the RP1's latches. In addition, register *RMAR* is incremented by one so that it addresses the second location in each module, while register RN1 is decremented by one to reflect that one read operation has been performed. Register RN1 hold number of locations to be read from each module in a run.



(b)

Figure 5.4.9 (a) ASM chart for Read RAM Control Unit of the RAM architecture using modules (b) The block diagram.

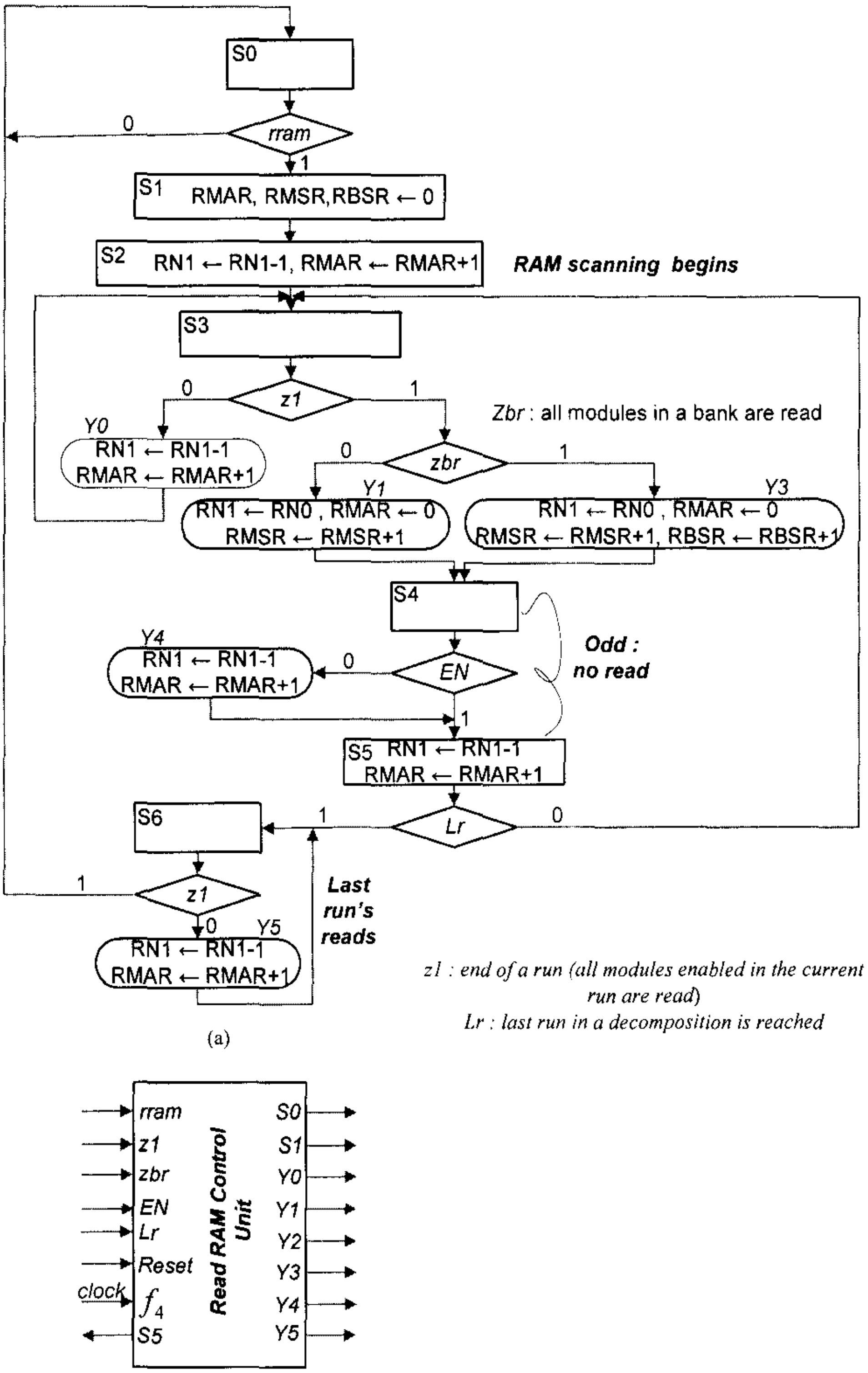
In state S3, the control executes the loop consisting only of state S3 and controlled by signal *z1*. This loop allows the control to continue reading the enabled RAM modules. Each time the loop is executed, register *RMAR* is incremented so that it points to the next location, while register RN1 is decremented by one. When RN1 is decremented to 1, it asserts signal *z1* high to indicate the three modules enabled in the current run all have been read and the next 3 modules for next run can be initiated. As the ASM moves from states S3 to S4, to get ready for the next run, register RN1 is again loaded with the same value, register *RMAR* is set 0, and register *RMSR* is incremented by one to select the next three modules that would be read in the next run.

In state S4, where a run ends and another begins, signal *EN* is examined, the least significant bit of RN0. If *EN* is 1, then *N* is odd and no read will take place when the control moves to S5. This will satisfy the condition required by the 4-parallel architecture, when a transition is made from a run to the next and if *N* is odd, no data is read from external memory. Otherwise, *N* is even and the first read operation in the new run is immediately performed. In both cases, the next state is S5.

In state S5, signal *Lr* (last run) is examined to determine whether the last run is reached. As long as, *Lr* is 0, the last run is not reached and the ASM executes the loop consisting of states S3, S4, and S5 until *Lr* becomes 1. When *Lr* becomes 1, it indicates that the run before the last one is now completed and the last run is in progress. Then, the ASM moves to state S6 to continue with the last run. Signal *Lr*, which is the output of the XNOR gate attached to register RM1, becomes 1 when RM1 is decremented to 1. Note that register RM1 is decremented internally by the signal labeled *z2* in *C-unit.*

In state S6, the ASM chart executes the loop consisting only of state S6 and controlled by signal *z1*. As long as, signal *z1* is 0, this loop will be executed and read operations required in the last run will be performed. When *z1* becomes 1, it indicates that all required reads in the last run have been performed. Then at the clock event, the control returns to its initial state S0.

The ASM chart for the second read RAM control unit and its block diagram, which controls the read operations of the RAM architecture (Figure 5.2.5) designed



RAM scanning begins

Zbr : all modules in a bank are read

Odd : no read

Last run's reads

z1 : end of a run (all modules enabled in the current run are read)

Lr : last run in a decomposition is reached

Figure 5.4.10 (a) ASM chart for Read RAM Control Unit of the RAM architecture using banks (b) The block diagram using banks, are given in Figure 5.4.10. The ASM chart shown in Figure 5.4.10 (a) is basically identical in every aspect to the one shown in Figure 5.4.9 (a). Except, it has one extra decision box between states S3 and S4 with the control input signal labeled *zbr* (see Figure 5.2.5). When all modules in a bank are read, signal *zbr* becomes 1. When *zbr* becomes 1, register RN1 is loaded again with the same value, register *RMAR* is set 0, and register RBSR (read bank select register) and RMSR are incremented to select the first three modules in the new bank. Otherwise, the control will continue reading the same bank. In both cases, the next state is S4.

5.4.4 Write RAM/Subband Memory Control Unit

Write RAM/subband memory control unit consist of two control units, write RAM control unit and write subband memory control unit. Write RAM and subband memory control units are responsible for performing write operations in the LL-RAM and subband memory, respectively. Both control units are activated at the same time, when signals *wsub* and *wram* are asserted high by the CPs control unit and are terminated at the same time. However, in the last level decomposition, only write subband memory control unit will be activated, since the LL-subband of the last decomposition is required to be stored in the subband memory block labeled LL_{jmax} not in the LL-RAM.

On the other hand, number of clock cycles that would elapse between the cycle, where the first inputs are loaded into CP1 and CP3 latches and the cycle where the first output coefficients generated CP1 and CP3 are loaded into the output latches, can be obtained from Eqs (4.4) and (4.5) as follows.

$$C2 - C1 = 4k_c \qquad (5.3)$$

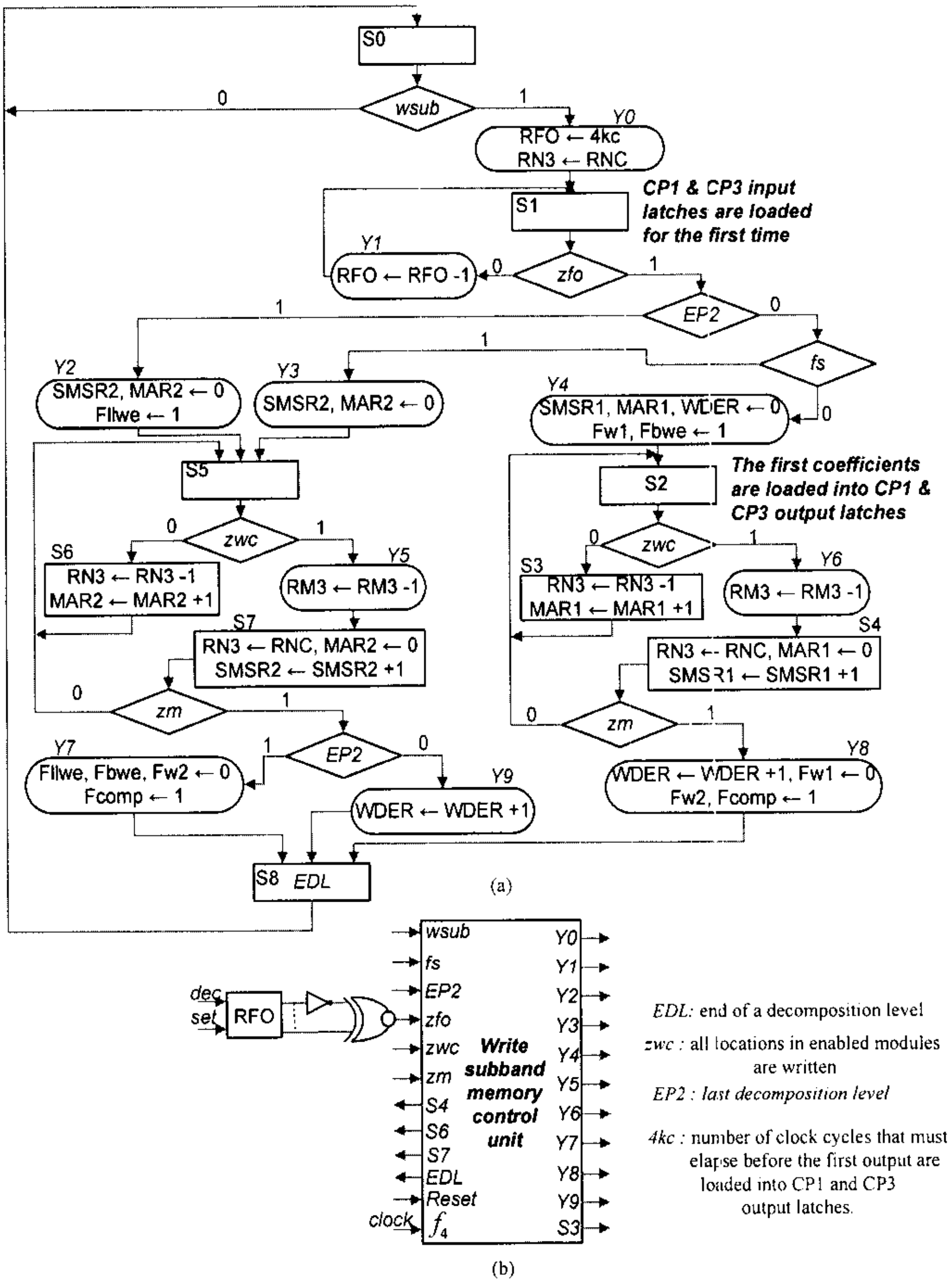
In order to detect occurrence of this event, register RFO is added to write subband memory control unit shown in Figure 5.4.11 (b). Register RFO is a down counter with control signals *set* and *dec* (decrement). Initially, RFO is set equal to $4k_c$ by asserting signal *set* high. This register is then decremented by one every clock cycle. When RFO is decremented to 1, it will assert signal *zfo* high to indicate that the first output coefficients will be available in CP1 and CP3 output latches at the end of the cycle. According to the dataflow table of the 4-parallel architecture, once the first four output coefficients are produced, then in every other clock cycle four new output coefficients will be produced until the process of decomposing a level into subbands is completed.

a) Write Subband Memory Control Unit

The ASM chart that describes write subband memory control unit is shown in Figure 5.4.11 (a) and its block diagram is shown in Figure 5.4.11 (b). The ASM chart is derived such that the control unit can write into subband memory according to the strategy explained in section 5.3.2, which can be summarized as follows. The strategy begins by storing the first three subbands of the first level decomposition in the subband memory blocks labeled HL1, HH1, and LH1. As soon as, the three subbands are written, the compression unit is informed by setting the FF labeled Fcomp high. Then the compression unit can read each subband block and compress it independently, while the DWT unit continues to further decompose the LL-subband of the first level decomposition. First, the compression unit will reset Fcomp zero and then will go on with compression process. When all levels after the first are decomposed and their subbands are stored in their respective subband memory blocks, the compression unit is again informed by asserting FF Fcomp high.

Write subband memory control unit, represented by the ASM chart shown in Figure 5.4.11 (a), is activated when the input signal *wsub* of the ASM is asserted high by the CPs control unit. Then the ASM moves from its initial state S0 to state S1. As the control moves from state S0 to S1, register RN3 is loaded with number of locations to be written in a module and register RFO is set equal $4k_c$, while the input latches of CP1 and CP3 are loaded internally with data of the first operation.

In state S1, the ASM execute the loop controlled by signal *zfo,* which consists of state S1 and the conditional output labeled *Y1*. As long as *zfo* is 0, this loop is executed and register RFO is decremented by one, while the control remains in the same state, S1. When register RFO is decremented to 1, it asserts its output signal *zfo* high, which indicates that the first output coefficients generated by CP1 and CP3 will be loaded into the output latches by the pulse ending the cycle (when the control moves from states S1 to S2). In addition, when signal *zfo* is 1, two status input signals *EP2* and *fs* are examined. If both signals are 0, which will be true only if this is the first level decomposition, the ASM will follow the path leading to state S2. But, if



EDL: end of a decomposition level

zwc : all locations in enabled modules are written

EP2 : last decomposition level

4kc : number of clock cycles that must elapse before the first output are loaded into CP1 and CP3 output latches.

Figure 5.4.11 (a) ASM chart for write subband memory control unit
(b) The block diagram

EP2 is 1, then it implies that the final decomposition is in progress and the conditional output labeled *Y2* is executed as the control moves from states S1 to S5. Execution of *Y2* sets FF Fllwe 1, which enables the subband memory block labeled LL_{jmax} to store the last subband LL-image. However, in all decomposition levels that are between the first and the last decomposition, signal *EP2* and *fs* will be 0 and 1, respectively, and the path leading to state S5 through the conditional output labeled *Y3* will be executed.

In state S2, the ASM executes the loop consisting of states S2 and S3. Each time this loop is executed three coefficients from CPs output latches will be simultaneously transferred to subband memory, where each coefficient will be stored in the first module of each subband blocks labeled HL1, HH, and LH1, starting from the first location. In addition, register RN3, which holds number of locations to be written in a module, is decremented by one and register *MAR1* is incremented by one so that it points to the next location in the three enabled modules that will be written next.

When register RN3 is decremented to 1, it asserts signal *zwc* high to indicate that all locations in the three enabled modules are written and the next three modules can be enabled for write. Then the ASM moves from states S2 to S4. As the ASM moves from states S2 to S4, register RM3, which holds number of modules to be written in each subband memory block, is decremented by one. In state S4, register RN3 is loaded again with the same value and register *MAR1* is reset 0, while register SMSR is incremented by one to select the next 3 modules, one from each subband memory blocks labeled HL1, HH1, and LH1 that will be written next.

In state S4, a decision is made based on signal *zm*. If *zm* is 0, the loop consisting of states S2, S3, and S4 is executed. This loop will execute several times before *zm* becomes 1. Signal *zm* becomes 1, when register RM3 is decremented to 0, which confirms that all modules in the first level are written. Then the control moves from states S4 to S8 during which register WDER is incremented by one to enable the next 3 subband memory blocks labeled HL2, HH2, and LH2 for writing the second level decomposition. In addition, Fw1 is reset 0 and Fw2 is set 1 to prevent further writing in the first level of the subband memory and to enable the second level for write, respectively. Furthermore, FF Fcomp is set 1 to inform the compression unit that the first level decomposition is completed and its subbands are now available in the subband memory blocks HL1, HH1, and LH1 for compression.

In state S8, the output signal labeled *EDL* (end of a decomposition level) is asserted high to inform the *A-unit* that the first level decomposition has completed and the next level decomposition can be initiated. Then at the clock event, the control returns to its initial state S0 and remains in that state until it is activated for the next level decomposition.

In all decomposition levels except the first, the second path leading to state S5 is executed. The second path executes a loop identical to the one in the first path. So every thing that has been said for the loop in the first path is also true for the loop in the second path.

At the end of the second loop, when signal *zm* is 1, the status input signal *EP2* is examined again, this time to determine if the last decomposition is completed. Signal *EP2* becomes 1 only when register RD is decremented to 1. Thus, the path labeled 0 leading to state S8 through the conditional output signal labeled *Y9* is always executed until the last decomposition is completed. When the last decomposition completes, signal *EP2* will be still 1. Then, at the clock event as the ASM moves to state S8, FFs Fllwe, Fbwe, and Fw2 are reset 0 to disable subband memory so that no further writes take place until it is read by the compression unit and the compression unit is informed by setting Fcomp 1.

In state S8, the output signal *EDL* is asserted high and at the clock event, the control returns to its initial state S0 and remains in that state until activated for decomposition of another image.

b) Write LL-RAM Control Unit

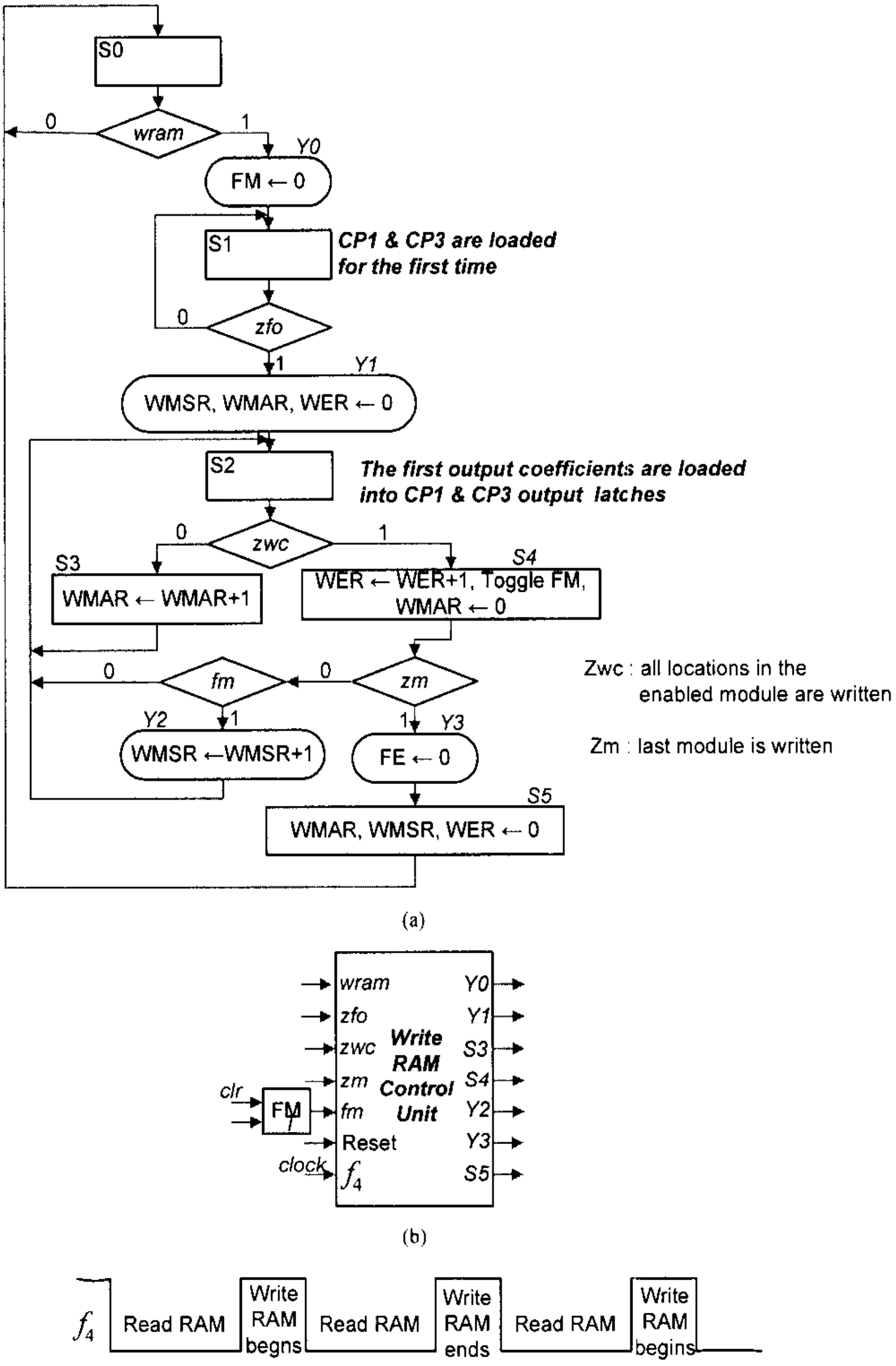
In following, two ASM charts for write LL-RAM control unit will be derived, one for the RAM architecture designed using modules shown in Figure 5.2.2 and the other for the RAM architecture designed using banks shown in Figure 5.2.5.

The first ASM chart that describes write RAM control unit for the RAM architecture shown in Figure 5.2.2 is given in Figure 5.4.12 (a) and its block diagram is shown in

Figure 5.4.12 (b). This control unit is activated when its input signal *wram* is asserted high by the CPs control unit. As the control moves from states S0 to S1 the FF labeled FM is set 0. FM is a FF with two signals *clr* (clear) and *T* (toggle). This FF is initially cleared to 0 and each time signal *T* is high it toggles. Since, the decoder labeled *dcodms* enables at a time 3 modules and writing is required to take place module-by-module, FM is used for determining the time at which register WMSR should be incremented such that the next 3 modules are enabled by the decoder at appropriate time, while writing into only one module at a time is still possible. Looking at the architecture in Figure 5.2.2 it can be determined that as soon as module number *(2m)* is written, where *m=1, 2, 3,...,* register WMSR can be incremented so that the decoder can safely select the next 3 modules. In other words, register WMSR will be incremented first after module number 2 is written then after module number 4 is written and so on. Thus, FM is used to serve this purpose.

In state S1, the ASM executes a loop exactly identical to the one in state S1 of the write subband memory control unit. This might suggest the possibility of eliminating this loop and the control can be activated from write subband memory control unit instead. Any way, as the control moves from states S1 and S2 register *WMSR*, *WMAR*, and WER are reset 0. Registers WMSR and WER together determine which module will be enabled for write, whereas register WMAR is used to address each location in the enabled module.

In state S2, two loops are executed, the inner loop which is controlled by signal *zwc* and the outer loop which is controlled by signal *zm*. These two loops are similar to the two loops that are in states S2 and S4 of the ASM chart for write subband memory control unit. The inner loop writes into the enabled module through register WMAR, which serves as address pointer starting from the first location. On the other hand, the outer loop selects the next module to be written through registers WER and WMSR. When all modules are written, signal *zm* becomes 1. Then, at the clock event the control moves to state S5. As the control moves to state S5, FF FE, which its output should be connected to the enable signal of the decoder labeled *dcodms* in Figure 5.2.2 (a), is set 0 to disable the LL-RAM so that it safeguard its contents until next level decomposition is initiated.



(a)

(b)

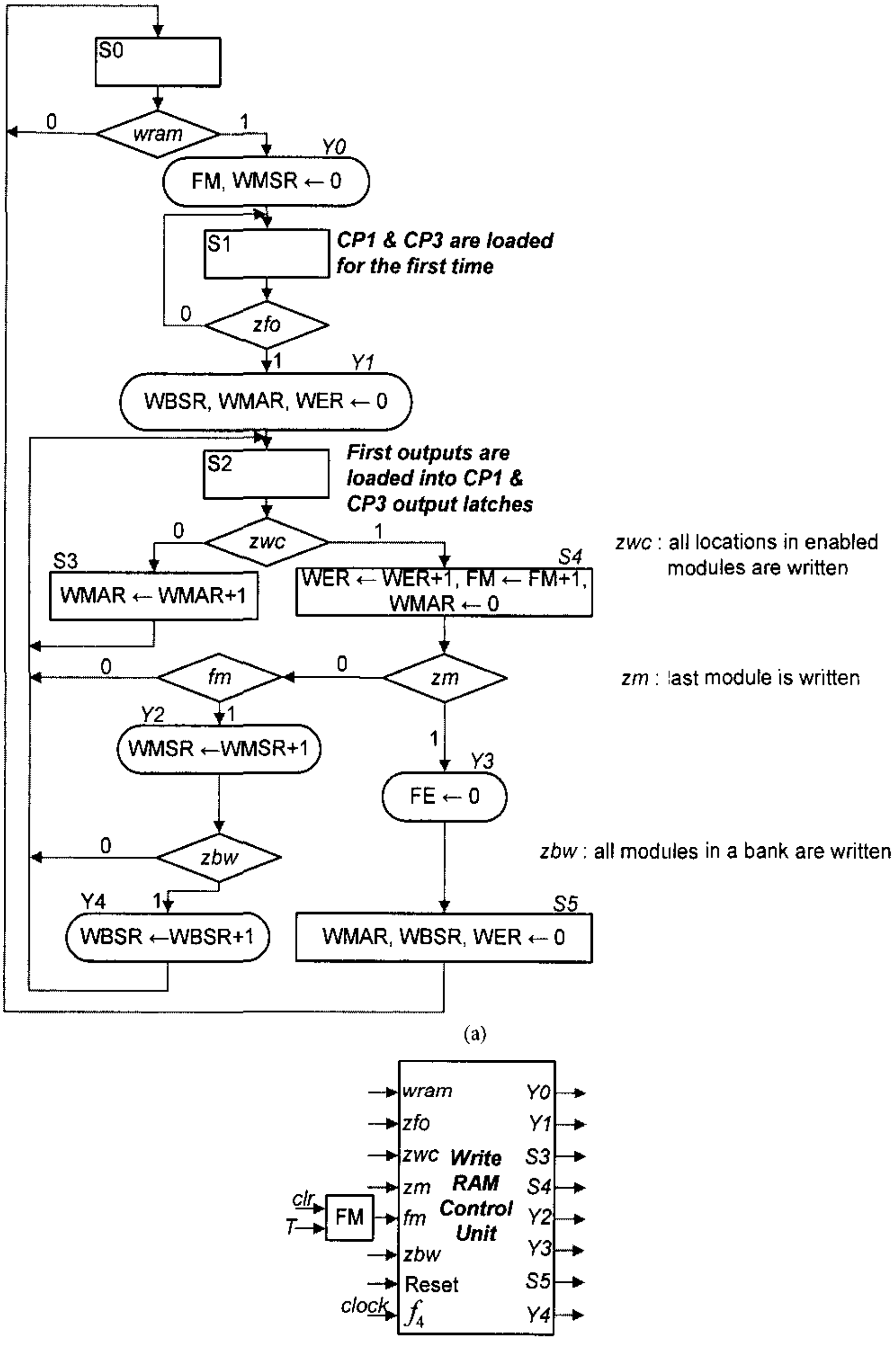
(c)

Figure 5.4.12 (a) ASM chart for write RAM control unit of the RAM architecture using modules (b) The block diagram (c) Proposed clock signal

In state S5, registers *WMAR*, *WMSR*, and *WER* are reset 0. This step is necessary to prevent modification of stored data by illegal writes during the period where the RAM is enabled and only read operations are taking place. This occurs always at the beginning of each decomposition level, since the LL-RAM is designed to allow both read and write to take place in the same clock cycle. This step will force the first module to be enabled and register *WMAR* to point at the first location. Thus, during this period all illegal writes will occur in the first location of the first module which will be read before the first illegal write takes place. Then, at the clock event the ASM moves from states S5 to S0 and remains in that state until it is activated again.

The second ASM chart shown in Figure 5.4.13 (a) describes the write RAM control unit for the RAM architecture designed using bank shown in Figure 5.2.5. The block diagram of the control unit is shown in Figure 5.4.13 (b). This ASM is basically identical in every part to the one shown in Figure 5.4.12 (a). Except that it has one extra decision box with a status input signal *zbw* (see Figure 5.2.5) and one conditional output box labeled *Y4* immediately inserted after the conditional output labeled *Y2*. When all modules in a bank are written, signal *zbw* is asserted high. Thus, every time signal *zbw* is 1, register WBSR (write bank select register) is incremented by one to enable the next bank for write and the control moves to state S2.

Finally, before closing this section, a very important issue regarding clock f_4 would be addressed. As mentioned before, the LL-RAM architecture is designed to support both reading and writing operations to take place in the same clock cycle. Read occurs in the first half cycle and write in the second half cycle. This might suggest the low and high pulses of clock f_4 should be equal. But, from the dataflow given in Table B.10 it can be seen that the CPs yield four output coefficients every other clock cycle, reference to clock f_4. That means these output coefficients remain in the output latches for two clock cycles before the next output coefficients are loaded. Thus, using a clock with equal pulses will be definitely inefficient. For example, if read is performed during the time where the first pulse of the clock is low and write is performed during the time where the second pulse of the clock is high, then in every two clock cycles, the second pulse of the first cycle will be used for writing, but the second pulse of the second cycle will be unused. Thus, in order to use the whole


S0
wram
0
1
Y0
FM, WMSR ← 0
S1
CP1 & CP3 are loaded for the first time
zfo
0
1
Y1
WBSR, WMAR, WER ← 0
S2
First outputs are loaded into CP1 & CP3 output latches
zwc
0
1
S3
WMAR ← WMAR+1
S4
WER ← WER+1, FM ← FM+1, WMAR ← 0
zwc : all locations in enabled modules are written
fm
0
zm
0
zm : last module is written
Y2
1
WMSR ←WMSR+1
1
Y3
FE ← 0
zbw
0
zbw : all modules in a bank are written
Y4
1
WBSR ←WBSR+1
S5
WMAR, WBSR, WER ← 0
wram
zfo
zwc
zm
fm
zbw
Reset
f_4
Write RAM Control Unit
Y0
Y1
S3
S4
Y2
Y3
S5
Y4
clr
T
FM
clock

(a)

(b)

Figure 5.4.13 (a) ASM flowchart for write RAM control unit of the RAM architecture using banks (b) The block diagram period effectively, a clock signal of the form shown in Figure 5.4.12 (c) is proposed. In this clock, the low pulse width is longer than the high pulse width and write operation which starts at a high pulse is allowed to complete in the next high pulse of the clock as indicated in Figure 5.4.12 (c). In addition, the fact that memory read operation takes more time than write operation makes this solution more attractive.

5.5 Conclusions

In this chapter, two novel VLSI memory architectures for 2-D DWT architectures for 5/3 and 9/7 are developed. Banking technique is utilized to form more efficient DWT memory architectures in term of speed. The advantage of the two proposed architectures is that they can be easily incorporated into single or parallel DWT architectures. Furthermore, to show that the architectures developed in this research are simple to control, the control algorithms for 4-parallel architecture including the LL-RAM and the subband memory were developed. To ease the control development, the overall system control is divided into several smaller units. Then, the algorithmic state machine (ASM) for each unit is developed. The control algorithms developed here can be used to derive the hardware of the control.

CHAPTER 6

2-DIMENSIONAL INVERSE DISCRETE WAVELETS TRANSFORM ARCHITECTURE DEVELOPMENT

6.1 Introduction

In chapter 3, architectures for 2-dimensional forward discrete wavelet transform (2-D FDWT) for 5/3 and 9/7 algorithms were developed. In this chapter, architectures for 2-dimensional inverse discrete wavelet transform (2-D IDWT) for 5/3 and 9/7 algorithms will be developed.

The function of the 2-D FDWT in a compression system is to decorrelate image pixels prior to compression step, whereas the function of the 2-D IDWT is to reconstruct and completely recover the original image from the decorretated image.

The 2-DFDWT decomposes an *NxM* image into subbands as shown in Figure 6.1.1 for 3-level decomposition. The decorrelated image shown in Figure 6.1.1 can be reconstructed by using 2-D IDWT as follows. First, it reconstructs in the column direction subbands LL3 and LH3 column-by-column to recover L3 decompostion. Similarly, subbands HL3 and HH3 are reconstructed to obtain H3 decomposition. Then L3 and H3 decompositions are combined row-wise to reconstruct subband LL2. This process is repeated in each level until the whole image is reconstructed.

The reconstruction process described above implies that the task of the reconstruction can achieved by using 2 processors. The first processor (the column-processor) computes column-wise to combine subbands LL and LH into L and subbands HL and HH into H, while the second processor (the row-processor) computes row-wise to combine L and H into the next level subband. The decorrelated image represented in Figure 6.1.1 is assumed to be residing with the same format in an external memory.

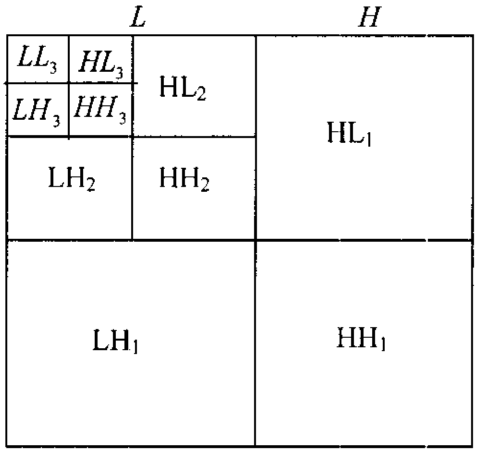


Figure 6.1.1 Subband decomposition of an *NxM* image into 3 levels.

6.2 Lifting-based 5/3 and 9/7 synthesis algorithms and data dependency graphs

The 5/3 and the 9/7 inverse discrete wavelet transforms algorithms are defined by the JPEG2000 image compression standard for 1-D signal *Y(n)* containing *N* samples as follow:

5/3 synthesis algorithm

$$step1: X(2n) = Y(2n) - \left\lfloor \frac{Y(2n-1) + Y(2n+1) + 2}{4} \right\rfloor$$

$$step2: X(2n+1) = Y(2n+1) + \left\lfloor \frac{X(2n) + X(2n+2)}{2} \right\rfloor \quad where\ n = 0,1,2....N-1$$

9/7 synthesis algorithm

Step1: $Y'(2n) = 1/k \cdot Y(2n)$
Step2: $Y'(2n+1) = k \cdot Y(2n+1)$
Step3: $Y''(2n) = Y'(2n) - \delta(Y'(2n-1) + Y'(2n+1))$
Step4: $Y''(2n+1) = Y'(2n+1) - \gamma(Y''(2n) + Y''(2n+2))$
Step5: $X(2n) = Y''(2n) - \beta(Y''(2n-1) + Y''(2n+1))$
Step6: $X(2n+1) = Y''(2n+1) - \alpha(X(2n) + X(2n+2))$

The data dependency graphs (DDGs) for 5/3 and 9/7 derived from the synthesis algorithms are shown in Figures 6.2.1 and 6.2.2, respectively. The DDGs are very useful tools in architecture development and provide the information necessary for the designer to develop more accurate architectures. The symmetric extension algorithm recommended by JPEG2000 is incorporated into the DDGs to handle the boundaries problems. The boundary treatment is necessary to keep number of wavelet coefficient

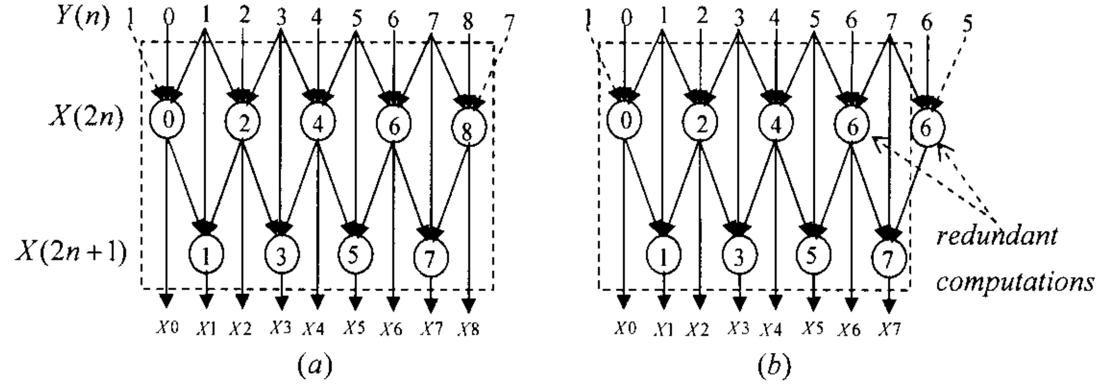


Figure 6.2.1 5/3 synthesis algorithm's DDGs for (a) odd and (b) even length signals

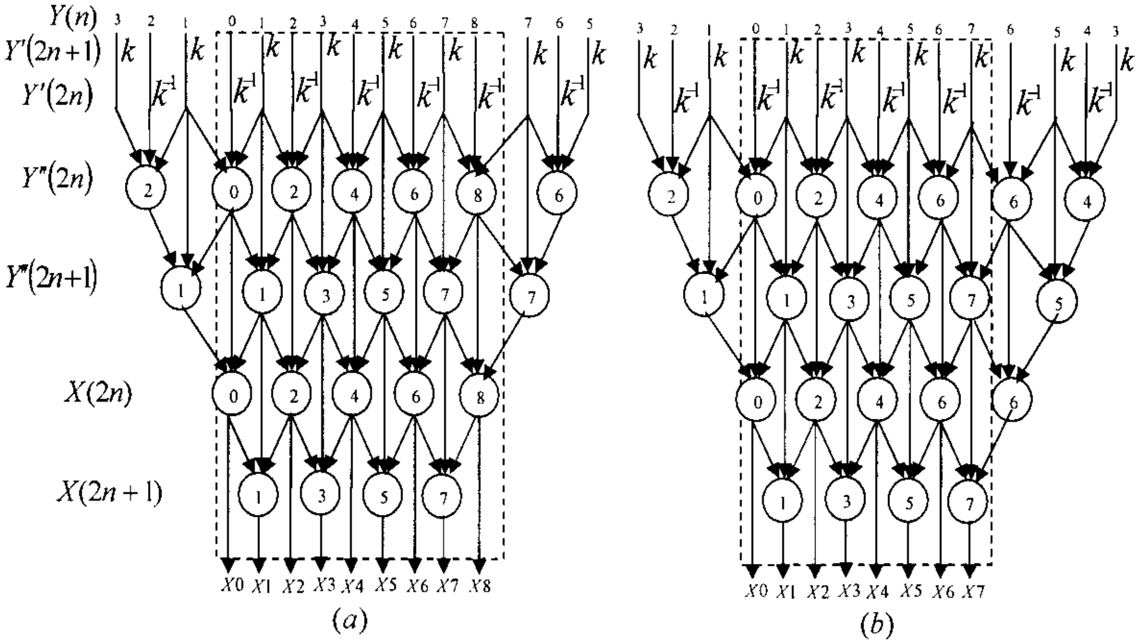


Figure 6.2.2 9/7 synthesis algorithm's DDGs for (a) odd and (b) even length signals

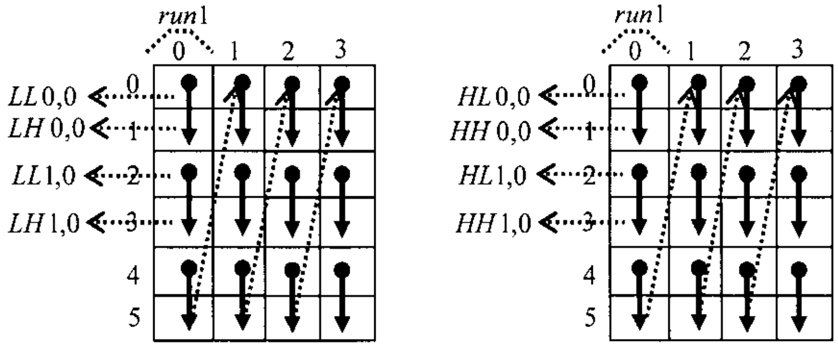
the same as that of the original input. The boundary treatment is only applied at the beginning and ending of the process. The nodes circled with the same numbers are considered redundant computations, which will be computed once and used thereafter. Note that the inputs coefficients with even numbers in the DDGs are low coefficients and that with odd numbers are high coefficients.

The strategy or the approach used in chapter 4 for developing 2-D FDWT architectures can be also used in 2-D IDWT architectures development. To ease the architecture development, the strategy divides the details of the development into two parts or steps each having less information to handle. In the first step, the DDGs are looked at from the outside, which is specified by the dotted boxes in the DDGs, in terms of the inputs and outputs requirements. It can be observed that the DDGs for 5/3 and 9/7 are identical when they are looked at from outside, taking into consideration only the input and output requirements, which can be specified for each algorithm by adopting appropriate scan method; but differ in the internal details Based on this observation, the first level of the architecture, call it, the external architecture is developed. In the second step, the internal details of the DDGs are considered for the development of the processors' datapath architectures, since the DDGs internally define and specify the internal structure of the processors.

6.3 Scan methods

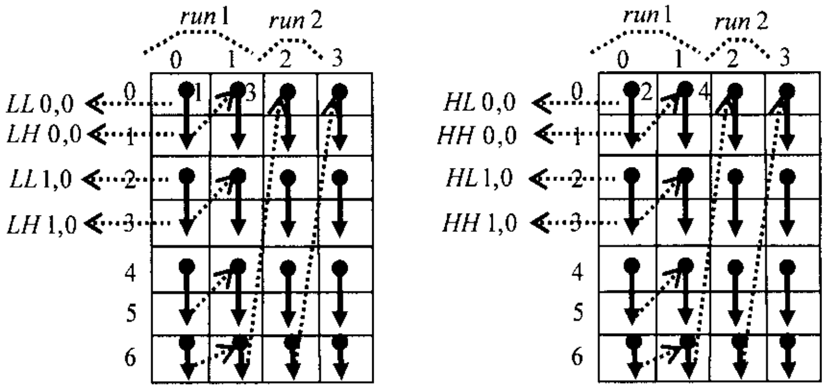
The first step in developing external architecture for 5/3 and 9/7, which would consist of a column-processor (CP) and a row-processor (RP), is to specify an appropriate scan method for each processor. Therefore, in Figures 6.3.1 and 6.3.2, two scan methods for 5/3 and 9/7 CP are illustrated, respectively. Similarly, two scan methods are illustrated in Figures 6.3.3 and 6.3.4 for 5/3 and 9/7 RP, respectively. These scan methods are developed mainly with one objective in mind to achieve, that is, to make the external architecture for both 5/3 and 9/7 algorithms identical. Note that the boxes labeled (*a*) in Figures 6.3.1 and 6.3.2 are formed for illustration purposes by merging together subbands LL and LH, where LL-subband coefficients occupy even rows and LH-subband coefficients occupy odd rows. Similarly, the boxes labeled *(b)* in Figures 6.3.1 and 6.3.2 are formed by merging HL and HH together.

The 5/3 CP scans the external memory column-by-column according to the scan method shown in Figure 6.3.1. The scan method illustrated in Figure 6.3.1 *(a)* scans the sections of the external memory labeled LL and LH as follows. First, the low coefficient, LL0,0 is scanned followed by the high coefficient, LH0,0 to initiate the first operation. The second operation is initiated by scanning coefficient LL1,0 followed by LH1,0 and so on. Note that coefficient LH0,0 is also required in the second operation. This process is repeated until the first column in both LL and LH are scanned. Then the scan moves to the second column in both LL and LH to repeat the process and so on. Similarly, sections HL and HH of the external memory are scanned.



(a) (b)

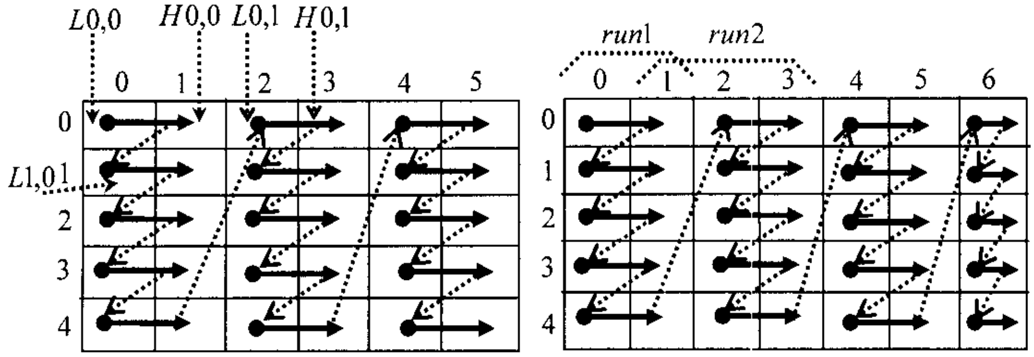
Figure 6.3.1 5/3 CP scan method (a) merging of LL and LH
(b) merging of HL and HH



(a) (b)

Figure 6.3.2 9/7 CP scan method(a) merging of LL and LH
(b) merging of HL and HH

However, in order to allow the RP, which operates on data generated by the CP, to work in parallel with the CP as soon as possible, the (a)'s (LL+LH) first column coefficients are interleaved in execution with the (b)'s (HL+HH) first column coefficients. Then the second column coefficients in both (a) and (b) are interleaved and so on. This columns coefficients interleaving process take place as follow. First, two coefficients LL0,0 and LH0,0 are scanned from the first column of (a) followed by another two coefficients HL0,0 and HH0,0 from the first column of (b). Then the scan moves to (a)'s first column and scans LL1,0 and LH1,0 followed by HL1,0 and HH1,0 from the first column of (b). This is repeated until the two columns are processed, say, to complete a run. The second run, similarly, processes the second column in both (a) and (b) and so on. The advantage of interleaving process not only it speedups the computations by allowing the two processors to work in parallel



(*a*) (*b*)

Figure 6.3.3 5/3 RP scan method (a) Even length row (b) Odd length row

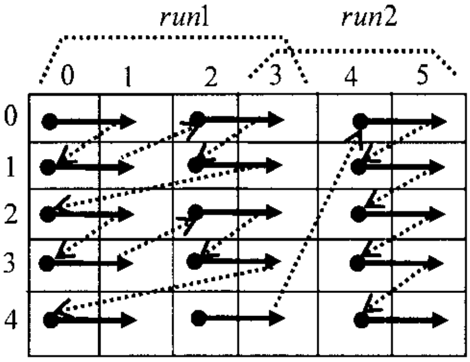


Figure 6.3.4 9/7 RP scan method

earlier during the computations, but also reduces the internal memory requirement between CP and RP to a few registers.

The scan method for 5/3 CP and the DDGs suggest that the 5/3 RP should scan its coefficients, which are generated by CP, according to the scan method illustrated in Figure 6.3.3. This figure is formed, for illustration purposes, by merging L and H decompositions, even though they are actually separate. In Figure 6.3.3, *L*'s coefficients occupy even columns, while *H*'s coefficients occupy odd columns. In the first run, coefficients of columns 0 and 1 are scanned by RP as shown in Figure 6.3.3. In the second run, coefficients of columns 2 and 3 are scanned and so on.

The scan method shown in Figure 6.3.2 for the 9/7 CP is basically identical in all runs to that of the 5/3 CP except in the first run which requires, according to 9/7 DDGs, interleaving of 4 columns; two from each *(a)* and *(b)* of Figure 6.3.2 as follows. First, coefficients LL0,0, HL0,0 from the first column of *(a)* are scanned. Second, coefficients HL0,0 and HH0,0 from the first column of *(b)* are scanned, then

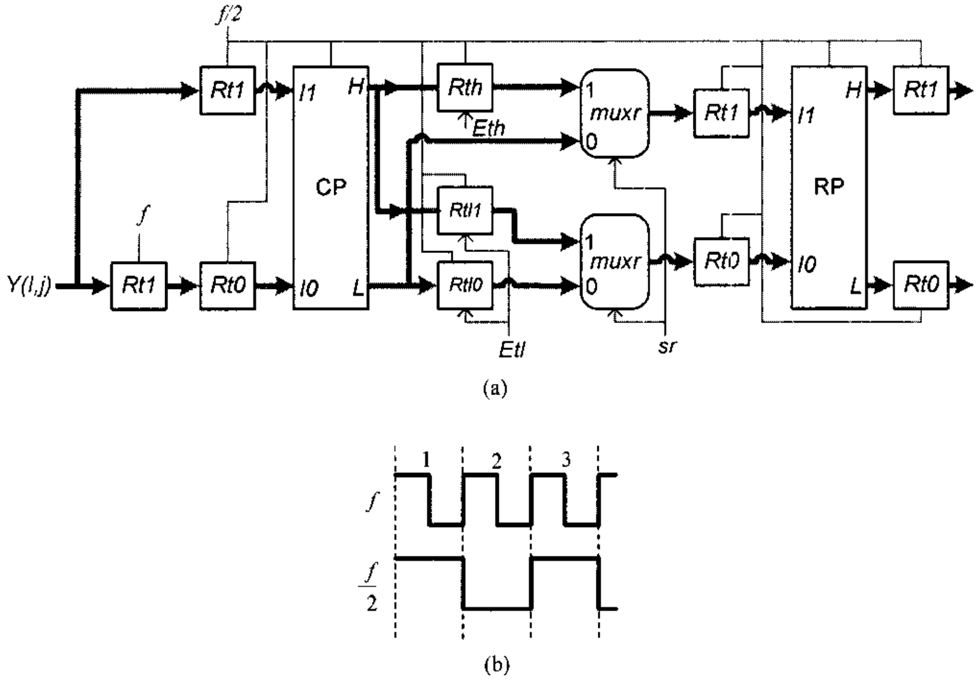
LL0,1 and LH0,1 from the second column of *(a)* followed by HL0,1 and HH0,1 from the second column of *(b)* are scanned. The scanning process then returns to the first column of *(a)* to repeat the process and so on.

The scan method for 9/7 RP is illustrated in Figure 6.3.4, which is basically also identical to the 5/3 RP scan method except in the first run. In the first run, the 9/7 RP's scan method requires considering the first four columns for scanning as follows. First, coefficients L0,0 and H0,0 from row 0 followed by L1,0 and H1,0 from row 1 are scanned. Then the scan returns to row 0 and scans coefficients L0,1 and H0,1 followed by L1,1 and H1,1. This process is repeated as shown in Figure 6.3.4 until the first run completes.

6.4 Proposed External Architecture

Based on the scan methods and the DDGs for 5/3 and 9/7, the architecture shown in Figure 6.4.1 (a) is proposed for 2-D IDWT. This architecture is also valid for combined 5/3 and 9/7 architecture. The architecture consists of two fully pipelined processor labeled CP and RP which will be developed later. The proposed architecture scans the external memory with frequency *f*, while the architecture operates with frequency *f/2* as indicated in Fig. 6.4.1 (a). The waveforms of the two clocks are shown in Figure 6.4.1 (b). The CP and the RP latches load new data every time clock *f/2* makes a positive transition.

The CP in the proposed architecture scans the external memory according to the scan methods shown in Figures 6.3.1 and 6.3.2 for 5/3 and 9/7, respectively, whereas RP scans the output latches of the CP labeled *Rtl0, Rtl1*, and *Rth* according to scan method illustrated in Figure 6.3.3 and 6.3.4 for 5/3 and 9/7, respectively. The architecture reconstructs a decorrelated image stored in the external memory such as the one shown in Figure 6.1.1 as follows. The CP begins the reconstruction process by scanning column-by-column the external memory's sections labeled LL3 and LH3.and that labeled HL3 and HH3 in an interleave manner to yield L3 and H3 decomposition, which are passed to RP through the latches labeled *Rtl0, Rtl1*, and *Rth*. L3's coefficients are stored in Rtl0 and Rtl1, whereas H3's coefficients are stored in Rth before they are read by RP.



(a)

(b)

Figure 6.4.1 (a) Proposed external architecture for 5/3 and 9/7 and combined 5/3 and 9/7 2-D IDWT (b) Waveform for clock *f* and *f/2*.

To be specific consider the dataflow of the architecture when it executes 5/3 algorithm. In the first clock cycle, coefficient LL0,0 from the first column of LL3 in the external memory, is scanned and is loaded into *Rd0* by the positive transition of clock *f*. The second clock cycle scans coefficient LH0,0 from the first column of LH3 and places it in the path labeled *Y(i,j)*. Then the positive transition of clock *f/2* loads Rd0 and LH0,0 into CP latches Rt0 and Rt1, respectively.

In the third clock cycle, coefficient HL0,0, from the first column of HL3, is scanned and is loaded into Rd0 by the positive transition of the clock *f*. The fourth clock cycle scans coefficient HH0,0 from the first column of HH3 in the external memory and places it in the path labeled *Y(i,j)*. Then the positive transition of clock *f/2* loads contents of *Rd0* and HH0,0 into the CP's latches labeled *Rt0* and *Rt1*, respectively. The scanning process then returns to subband LL3 in the external memory to repeat this interleaving process.

The CP generates every clock cycle two output coefficients. The first two output coefficients, L0,0 and L1,0 which belong to L3 decomposition are loaded into *Rtl0* and *Rtl1*, respectively, by the positive transition of clock *f/2*. During the next clock cycle, say, cycle *n* coefficients H0,0 and H1,0 which belong to H3 decomposition, will be placed in the output paths labeled L and H, respectively. Then the positive transition of the clock ending the cycle transfers Rtl0 and H0,0 in the output path, L, to the RP's latches labeled *Rt0* and *Rt1*, respectively, through the two multiplexers labeled *muxr*, while H1,0 in the output path labeled H is loaded int *Rth*. The second two output coefficients of L3, L2,0 and L3,0 are loaded into *Rtl0* and Rtl1, respectively, by the positive transition of the clock ending cycle *n*+1, while contents of *Rtl1* and *Rth* are transferred to RP latches *Rt0* and *Rt1*, respectively. This process is repeated according to the scan method illustrated in Figure 6.3.3.

On the other hand, the dataflow of the 9/7 architecture, which differs mainly in the first run from that of the 5/3 by requiring interleaving of 4 columns instead of two, is as follow. However, since the dataflow of the 9/7 CP is same as that of the 5/3 up to the fourth clock cycle, the dataflow description would continue from the fifth cycle. In the fifth clock cycle, the scanning process returns to LL3 and scans coefficient LL0,1 from the second column and loads it into *Rd0* by the positive transition of the clock ending the cycle. The sixth clock cycle scans coefficients LH0,1 from the second column of LH3 and places it in the path labeled *Y(i,j)*. Then the positive transition of the clock *f/2* loads *Rd0* and LH0,1 into CP's latches *Rt0* and *Rt1*, respectively. In the seventh clock cycle, the scan moves to HL3 in the external memory and scans coefficient HL0,1 from the second column and loads it into Rd0 by the pulse ending the cycle. The eighth clock cycle, scans coefficient HH0,1 from the second column of HH3 and places it in the path labeled *Y(i,j)*. Then the positive transition of the clock *f/2* loads Rd0 and H0,1 into CP's latches *Rt0* and *Rt1*, respectively. The scanning process then returns to subband LL3 in the external memory to repeat the process until the first run completes. In the second run, the third column in both (a) and (b) of Figure 6.3.2 are consider for processing and proceeds as that of the 5/3 described earlier. Remember, in Figure 6.3.2 (a), coefficients of subband LL occupy even rows, while subband LH coefficients occupy odd row. Similarly, in Figure 6.3.2 (b), coefficients of subband HL occupy even row, while subband HH coefficients occupy odd rows.

Now, let's look at the dataflow of the 9/7 from RP side. The CP yields every clock cycle two output coefficients. The first two output coefficients, L0,0 and L1,0 from

L3 decomposition are loaded into *Rtl0* and *Rtl1*, respectively, by the positive transition of clock *f/2*. During the next clock cycle, say, cycle *n*, coefficients H0,0 and H1,0 from H3 decomposition will be placed in the output path labeled L and H, respectively. Then, the positive transition of the clock ending the cycle, transfers *Rtl0* and coefficient H0,0 in the output path labeled L, to RP's latches *Rt0* and *Rt1*, respectively, while H1,0 in path H is loaded into Rth. In cycle *n*+1, coefficients in *Rtl1* and Rth are transferred to RP's latches *Rt0* and *Rt1*, respectively, while the two output coefficients L0,1 and L1,1 from L3 decomposition are loaded into *Rtl0* and *Rtl1*, respectively, by the positive transition of the clock ending the cycle. During cycle *n*+2, two output coefficients H0,1 and H1,1 from H3 decomposition will be placed in the output path labeled L and H, respectively. Then the positive transition of the clock ending the cycle, transfers *Rtl0* and H0,1 in path L to RP latches *Rt0* and *Rt1*, respectively, while H1,1 in path H is loaded into *Rth*. Cycle *n+3* transfers contents of *Rtl1* and *Rth* to RP latches *Rt0* and *Rt1*, respectively, while the two new output coefficients, L2,0 and L3,0 from L3 decomposition generated by CP are loaded into *Rtl0* and *Rtl1*, respectively. This process is repeated according to the scan method shown in Figure 6.3.4. The dataflow table of the architecture will be given later after the two processor, labeled CP and RP in Figure 6.4.1 are developed.

One important point, if number of columns in *(a)* and *(b)* of Figures 6.3.1 and 6.3.2 are not equal, then the last run will consist of only one column of *(a)*. In that case, scan the last column of *(a)* every other clock cycle, reference to clock *f/2,* so that CP yields a valid pair of output coefficients every other clock cycle. Because, an attempt to scan the last column every clock cycle of *f/2* will result in CP generating more coefficients than that can be handled by RP. The dataflow from RP side is as follow. Suppose, at clock cycle *n* the first two output coefficients of the CP L0,m and L1,m of the last column m are loaded into *Rtl0* and *Rtl1*, respectively. In the next clock cycle, cycle *n+1*, Rtl0 is transferred to *Rt0* of RP, while data in path L and *H* generated by CP during the cycle are not loaded into Rtl0 and Rtl1, since they are invalid coefficients. In cycle *n+2*, coefficients L2,m and L3,m generated by CP are loaded into *Rtl0* and *Rtl1*, respectively, while content of *Rtl1* is transferred to RP latch *Rt0* through muxr. This process is repeated until the run completes.

The control signal values for signals *Eth, Etl,* and *sr* that could be issued by a control unit are derived in Table 6.1 starting from clock cycle *n* where the first two output coefficients generated by CP are loaded into *Rtl0* and *Rtl1*. However, note that signal *Eth* can be eliminated, since it alternates between don't-care and 1. In addition, since the first value of signal *sr* is a don't-care and the rest of the signal values are same as that of signal *Etl,* then signal *sr* and *Etl* can be combined into one signal *sr*.

Table 6.1 Control signal values for *Eth, Etl, and sr*

CK *f/2*	*Eth*	*Etl*	*sr*
N	X	1	X
n+1	1	0	0
n+2	X	1	1
n+3	1	0	0
n+4	X	1	1

6.5 Processors' architecture development

6.5.1 Inverse 5/3 processor's architecture development

To complete the architecture for 2-D IDWT, the last phase is to design the row and column processors' datapath architectures for 5/3 and 9/7 algorithms separately that can be incorporated into CP and RP of the external architecture shown in Figure 6.4.1 (a). First, the datapath architecture for 5/3 will developed followed 9/7 in the next section.

Based on the algorithm (6.1) and the DDGs shown in Figure 6.2.1, the inverse 5/3 processor datapath architecture shown in Figure 6.5.1 is obtained. The multiplexers labeled *muxe0, muxe*1, and *muxe2* implement the symmetric extension algorithm incorporated into the DDGs. This 3-stage pipelined processor is formed by mapping the two lifting steps of the inverse 5/3 algorithm into two pipeline stages. Steps 1 and 2 are mapped into stages 1 and 3 in Figure 6.5.1, respectively. Then, stages 1 and 3 are connected through stage 2 to form a 3-stage pipelined processor. Stage 2 is necessary because stage3, which implements step 2, requires two successive low coefficients from stage 1 to perform an operation. When the first coefficient generated by stage 1 is in *Rt0* of stage 3, the second coefficient will be in Rt0 of stage 2 and will be applied to stage 3 through the path labeled X(2n+2), the Forward path. The nodes circled with even number in the DDGs, which represent step 1 of the algorithm, are all computed in stage 1 in the order indicated in the DDGs. Similarly, nodes circled with odd number, which represent step2, are computed in stage 3 in the order specified in the DDGs.

In the following the operations of the extension multiplexers are explained. First, according to DDGs for 5/3, in the calculation of the first low coefficient *X0*, the second input *Y1* must be allowed in stage 1 to pass through the two multiplexers, labeled *muxe0* and *muxe1* to the adder. Second, in the calculation of the last coefficient, for example, X8 in the DDG for odd length signals, the input coefficient *Y7*, which will be in *Rt1* of stage 2, must be allowed to pass through both *muxe0* and *muxe1* to the adder. On the other hand, during the normal computations, which take place between the first and last calculations, the current input coefficient in *Rt1* of stage 1 and the previous coefficient in *Rt1* of stage 2 are allowed to pass through muxe0 and *muxe1*, respectively, to the adder. However, note that in even length signals, according to the DDG in Figure 6.2.1 (*b*), the last high and low coefficients calculations take place as normal calculations. As for the extension multiplexer

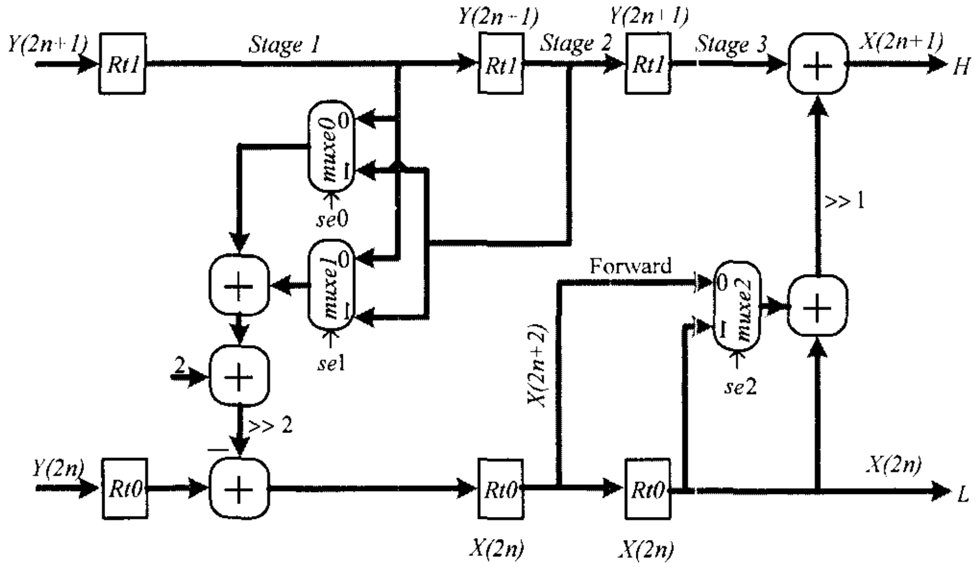


Figure 6.5.1 Inverse 5/3 processor datapath architecture with symmetric extension

labeled *muxe2* in stage 3, its normal function is to pass in all cases the forward signal, *X(2n+2)*, to the adder in stage 3, except in the even length signals and in the calculation of the last coefficients, multiplexer *muxe2* passes the coefficient stored in

Rt0 of stage 3 to the adder instead of the one in the Forward path. Table 6.2 shows the control signal values that are required to be issued by the control unit order for the extension multiplexers to perform the required functions.

Table 6.2 Extension's control signals

	se0	*se1*	*se2*
First	0	0	0
Normal	0	1	0
Last	1	1	0

a) Odd length signals

	se0	*Se1*	*se2*
First	0	0	0
Normal	0	1	0
Last	0	1	1

b) Even length signals

6.5.2 Inverse 9/7 processor's datapath architecture

Based on the 9/7 algorithm 6.2 and its DDGs shown in Figure 6.2.2, the inverse 9/7 processor datapath architecture is shown in Figure 6.5.2. This processor architecture is formed by mapping steps 3, 4, 5, and 6 of the algorithm into stages 2, 4, 5, and 7, respectively, while steps 1 and 2 are mapped into stage 1 to allow the two steps to perform in parallel. This architecture also can be thought formed by connecting two 5/3 processors at stage 4.

The multiplexers in stages 2, 4, 5, and 7 implement the symmetric extension algorithm that is part of the DDGs shown in Figure 6.2.2. Table 6.2 also provides appropriate control signal values that must be issued by the control unit to the 9/7 extension multiplexers so that they can perform their required functions. These extension multiplexers functions exactly the same way as that of the 5/3 described earlier.

6.5.3 Combined inverse 9/7 and 5/3 processors architecture

The 9/7 processor architecture shown in Figure 6.5.2 can be modified as shown in Figure 6.5.3 to give the combined processor architecture for both 9/7 and 5/3. The 5/3 processor is incorporated into the 9/7 processor by modifying stages 1, 2, and 4, while the remaining stages remain the same. The control signal labeled $lossy/\overline{lossless}$ enables the architecture to be selected either to perform 9/7 or 5/3 algorithms. Thus, if signal $lossy/\overline{lossless}$ is 1, the architecture reconstructs the image using 9/7 algorithm, otherwise, it reconstructs the image using 5/3 algorithm. The combined architecture could be a very useful and efficient in situations where the decoder in one site is required to perform either lossless or lossy image reconstruction. In addition, the advantage of the combined architecture is that a great saving in silicon area can be achieved.

6.5.4 Modified row and column processors for 5/3 and 9/7 external architecture

The 5/3 and 9/7 processors datapath architectures shown in Figures 6.5.1 and 6.5.2 were developed assuming the processors scan coefficients from external memory row-by-row or column-by-column. The CPs for 5/3 and 9/7 external architecture do, according to the scan methods shown in Figures 6.3.1 and 6.3.2, scan the external memory column-by-column. However, since the CPs for both 5/3 and 9/7 are required to rotate between executing coefficients of subbands LL and LH with that of HL and HH in an interleave fashion, the processor datapath architectures for 5/3 and 9/7 shown in Figures 6.5.1 and 6.5.2 should be modified as shown in Figures 6.5.4 and 6.5.5, respectively, in order to allow interleaving in execution. The 5/3 processor shown in Figure 6.5.1 is modified by adding one stage between stages 2 and 3, since it interleaves two column in execution, to obtain a 4-stage CP shown in Figure 6.5.4 that fit into 5/3 external architecture.

On the other hand, the 7-stage 9/7 processor datapath architecture shown in Figure 6.5.2 is modified by adding 3 stages between stages 3 and 4 and stages 6 and 7 each, since it is required to interleave 4 columns in the first run, to obtain a 13-stage CP shown in Figure 6.5.5 for 9/7 external architecture. Figure 6.5.5 show only the first seven stages, since the remaining 6 stages are identical to stages 2 to 7. Tables B.18 and B.19 (a) show the dataflow of the 5/3 and the 9/7 architectures, respectively, which illustrate how interleave execution takes place.

In Figure 6.5.5, the control signal, *s* of the two multiplexers labeled *mux* is set 1 in the first run to allow interleaving of 4 columns, whereas in all other runs it is set 0 to allow interleaving of 2 columns as required by scan method shown in Figure 6.3.2, which is identical to 5/3 scan method shown in Figure 6.3.1 in all runs except the first run. This also implies that reference to Figure 6.5.3, Figure 6.5.5 can be easily

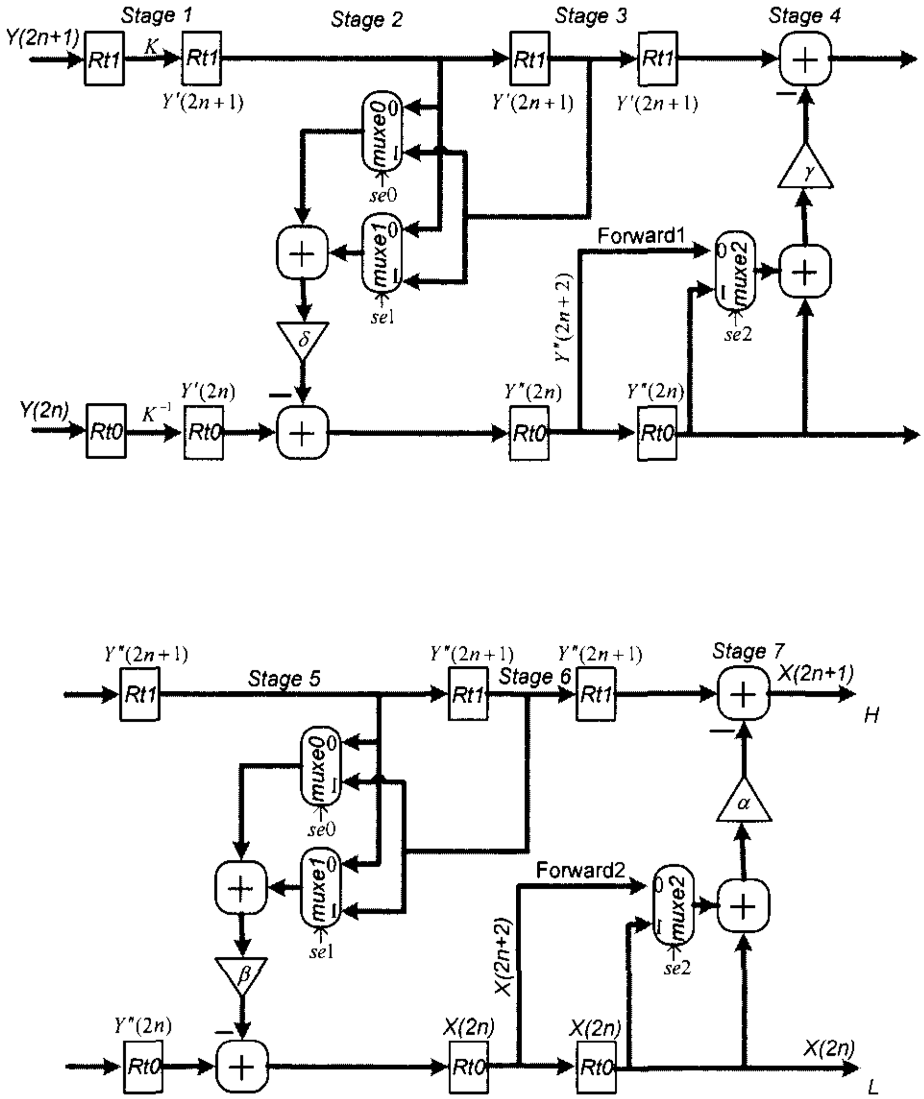


Figure 6.5.2 Inverse 9/7 processor datapath architecture with symmetric extension

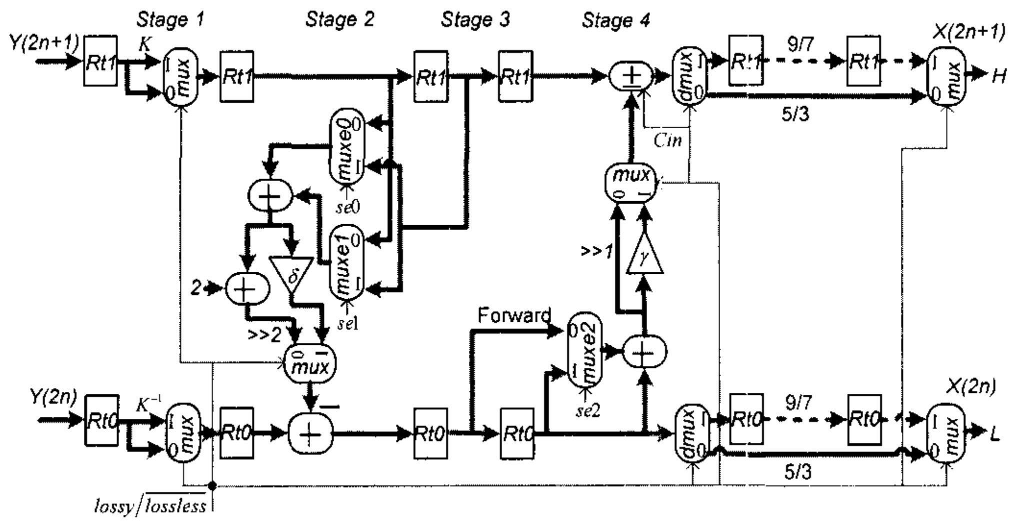


Figure 6.5.3 Combined Inverse 9/7 and 5/3 processor datapath architecture

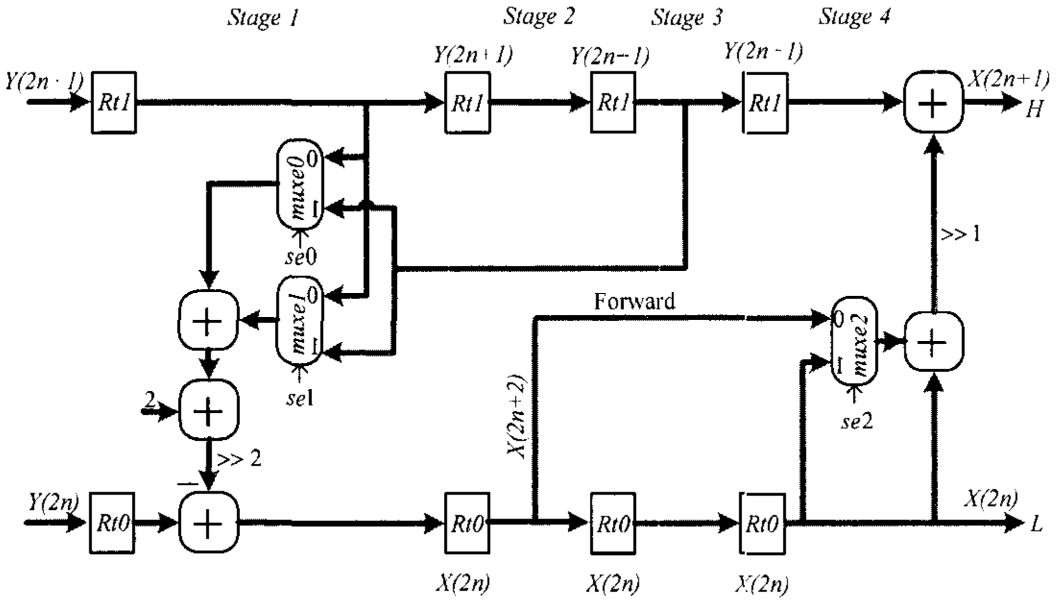


Figure 6.5.4 Modified inverse 5/3 CP datapath architecture with symmetric extension

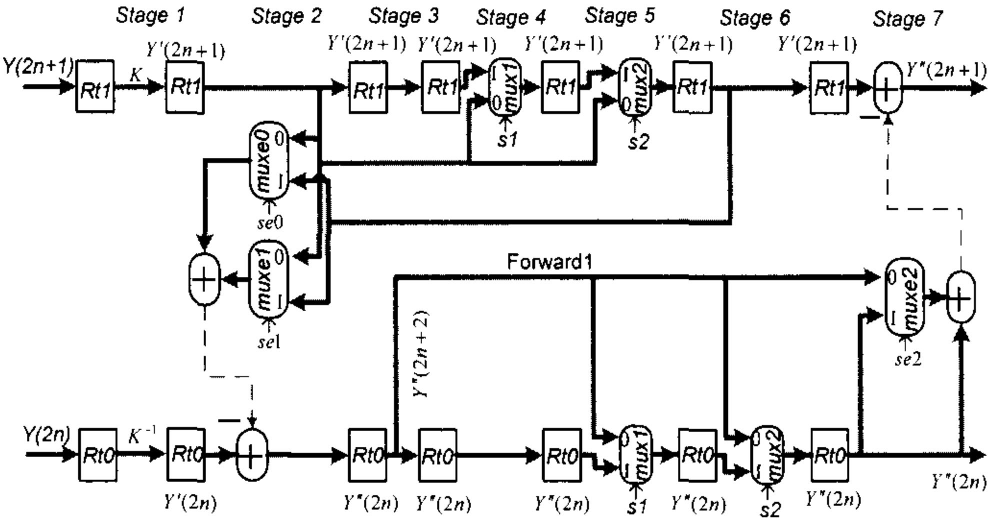


Figure 6.5.5 Modified CP for 9/7 and combined 5/3 and 9/7 datapath architecture

modified as a CP for combined 5/3 and 9/7 external architecture shown in Figure 6.4.1. Thus, when signals $lossy/\overline{lossless}$ of Figure 6.5.3 and *s* both are zero the architecture performs 5/3; otherwise, it performs 9/7.

On the other hand, the RP in the proposed external architecture scans coefficients of the high (H) and low (L) decompositions generated by CP according to scan methods shown in Figure 6.3.3 and 6.3.4 for 5/3 and 9/7, respectively. Thus, this would require modifying the 5/3 and the 9/7 processor datapath architectures shown in Figures 6.5.1 and 6.5.2, respectively, as follows. Looking at the input conditions of the 5/3 and the 9/7 in the DDGs and the scan methods shown in Figures 6.3.3 and 6.3.4 one can immediately recognize that all input coefficients occupying odd columns in Figures 6.3.3 and 6.3.4 in each run need to be stored in a temporary line buffer (TLB) of size *N*, since they are required in next run's computations. Therefore, a TLB should be added in both Figures 6.5.1 and 6.5.2.

Furthermore, according to the 5/3 DDGs, applying the scan method shown in Figure 6.3.3 would require addition of another TLB of size *N* in order to store low coefficients of a run calculated in stage 1 of Figure 6.5.1, since they are required in high coefficients that would be calculated in stage 3 in the next run. When these changes are incorporated into Figure 6.5.1, the 4-stage RP shown in Figure 6.5.6, is obtained for 5/3 external architecture. Table B.18 shows the dataflow of the 5/3 architecture. In this dataflow table, the first location of TLB1, for example, contains coefficient Y0(1) and the second location contains Y1(1) followed by Y2(1) in the third location and so on. In the first run, TLBs are only written, whereas starting from the second run, the TLBs are read and written in the same clock cycle. For instance, in the second run at cycle 30, Table B.18 shows that the first location of TLB1 is read into Rt2 of stage 2 and a new coefficient labeled Y0(3) is written into it.

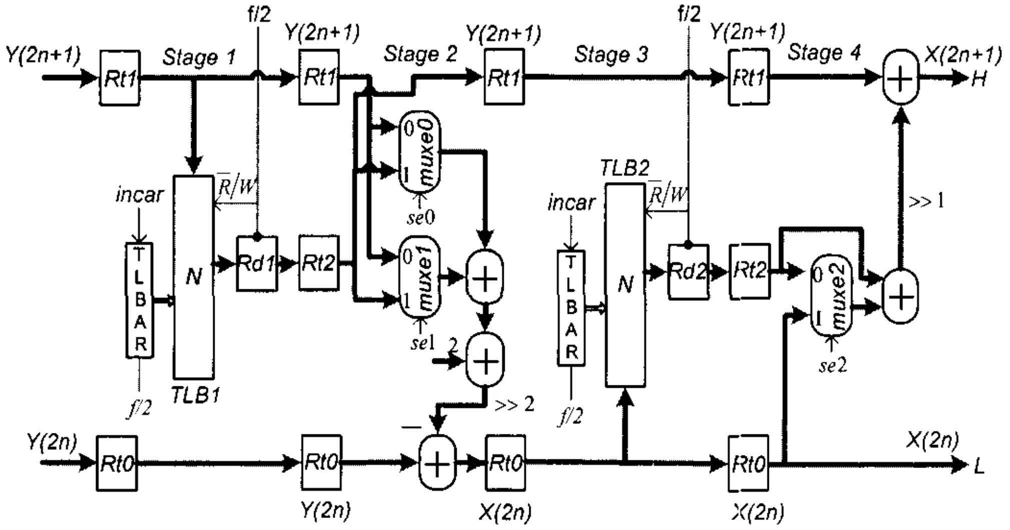


Figure 6.5.6 Modified inverse 5/3 RP datapath architecture with symmetric extension

On the other hand, according to the 9/7 DDGs, applying the scan method shown in Figure 6.3.4 would require addition of three TLBs each of size N in the datapath architecture shown in Figure 6.5.2. The first TLB is needed because all coefficients calculated in stage 2 of Fig. 6.5.2, in a run, are required in stage 4 in the next run. The second TLB is needed for storing N coefficients calculated in stage 4 in a run, which are required in the calculations that take place in stage 5 in the next run. The third TLB is necessary to keep N coefficients calculated in stage 5 in a run, which are required in stage 7 calculation in the next run. When these changes are incorporated into Figure 6.5.2, the 9-stage RP shown in Figure 6.5.7, is obtained for 9/7 external architecture.

ETLB : enable TLB incar : increment AR clar : clear AR

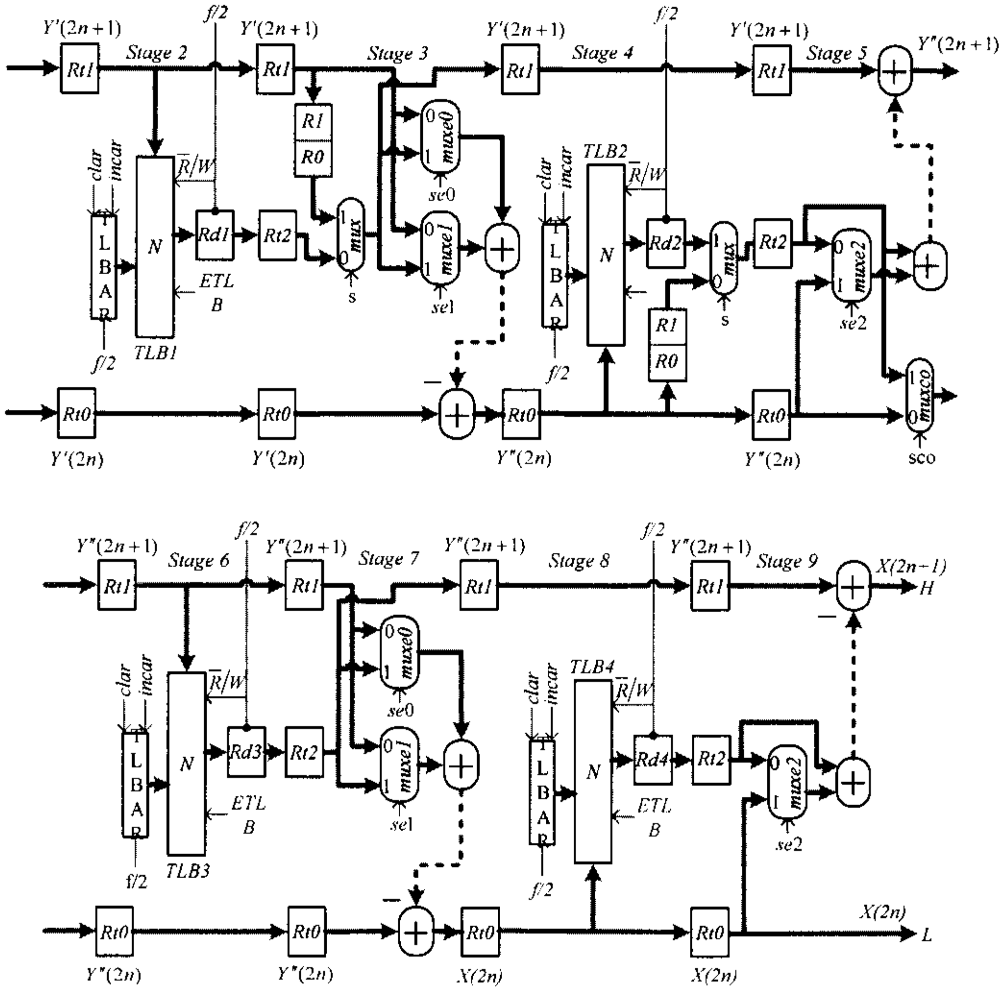


Figure 6.5.7 Modified RP for 9/7 and combined 5/3 and 9/7 datapath architecture

The registers labeled R0 and R1 in stage 3 of Figure 6.5.7 are added because the scan method for 9/7 illustrated in Figure 6.3.4 requires in the first run, for example, storing the second input coefficient of both rows 0 and 1 in Figure 6.3.4, labeled H0,0 and H1,0, since these two coefficients are required in the second operation of rows 0 and 1, respectively. Whereas, registers R0 and R1 in stage 4 are added to store in the first run, the first two coefficients computed in stage 3 for each two rows using the first two input coefficients of each row, since they are required in the two successive computations that take place in stage 5. Note that the control signal *s* of the two multiplexers, labeled *mux* in stages 3 and 4 of Figure 6.5.7 is set 1 in the first run to pass coefficients stored in R0 and R1 and 0 in all other runs to pass coefficients stored in TLB1 and TLB2.

The details of the 9/7 architecture dataflow from RP side is given in Table B.19 (b). This table shows that in the first run each two outputs are followed by two empty cycles. To see why this occurs can be determined by looking at column 5 (stage 5) in the dataflow Table B.19 (b), which shows that in clock cycle 22 and 23 no data are passed to stage 5 from 4. Similarly, in clock cycles 26 and 27, and so on. This mainly is a consequence of the scan method adopted in the first run, which forces stage 5 to wait each time on two successive coefficients calculated in stage 3 before it can proceed. However, in all subsequent runs, the 9/7 architecture would yield a pair of output every clock cycle.

It is very important to note that when the RP executes its last set of input coefficients, according to 9/7 DDGs for odd and even signals shown in Figure 6.2.2 it will not yield all required output coefficients as expected by the last run. For example, in the DDGs for odd length signals shown in Figure 6.2.2 (a), when the last input coefficient labeled Y8 is applied to RP it will yield output coefficient X5 and X6. To get the last remaining two coefficients X7 and X8, the RP must execute another run, which will be the last run in order to compute the remaining two output coefficients. Similarly, when the last two input coefficients labeled Y6 and Y7 in the DDG for even length signals shown in Fig. 6.2.2 (b) are applied to 9/7 RP it will yield output coefficients X3 and X4. To obtain the remaining output coefficients X5, X6, and X7, two more runs should be executed by RP according to the DDG. The first run will yield X5 and X6, whereas the last run will yield X7. The details of the computations that take place during each of these runs can be determined by examining the specific area of the DDGs.

Control signals of a pipelined processor such as the signals of the pipeline 9/7 RP shown in Figure 6.5.7 can be issued every clock cycle by a control unit. The control signal values issued in each clock cycle are transferred to the first stage of the pipeline and are loaded into the control signal latches (CSTs) that are similar to the pipeline latches, to carry these signal values from stage-to-stage. When a stage where a signal (or signals) is used is reached, the signal value carried by its CST is applied, while the remaining signals are carried to the next stage. For example, in Table 6.3 starting from cycle 14, the control signal values for signals *incar*, *clar*, *ETLB*, *se0*, etc. for 4

Table 6.3 Control signal values for 9/7 RP

CK	*incar*	*clar*	*ETLB*	*S*	*se0*	*se1*	*se2*
14	0	0	0	1	0	0	0
15	0	1	0	1	0	0	0
16	1	0	1	1	0	1	0
17	1	0	1	1	0	1	0

cycles are derived. In cycle 14, the control signal values listed at cycle 14 in Table 6.3 would be loaded by the control unit into CSTs of the first pipeline stage. Similarly, in cycle 15, the control signal values listed at cycle 15 in the table would be transferred to CSTs of the first stage, while the control signal values issued in cycle 14 would be transferred to CSTs of the next stage and so on.

In addition, observe that if registers R0 and R1 in stages 3 and 4 are eliminated, the RP for 9/7 from stages 2 to 5 and from 6 to 9 are similar in structure to the 4-stage 5/3 RP shown in Figure 6.5.6. This implies that the RP for 9/7 can be easily modified to work as a RP for the combined 5/3 and 9/7 external architecture.

In the combined architecture, signal *s* of the two multiplexers, labeled *mux* in stages 3 and 4 of Fig. 6.5.7 is set 0 if the architecture is to perform 5/3; otherwise, it is set 1 in the first run and 0 in all other run if the architecture is to perform 9/7. Moreover, the multiplexer labeled *muxco* in stage 5 is only needed in the combined 5/3 and 9/7 architecture, otherwise, it can be eliminated and Rt2 output can be connected directly to the input of the Rt0 of the next stage. Thus, in the combined architecture signal *sco* of *muxco* is set 0 if the architecture have to perform 5/3, otherwise, it is set 1 if the architecture have to perform 9/7.

Note that the TLBs in Figures 6.5.6 and 6.5.7 are required to be read and written in the same clock. Therefore, signal $\overline{R}/W$ is connected to clock *f/2* so that the TLB can be read in the first half cycle and written in the second half cycle. The register labeled TLBAR (TLB address register) generates addresses for TLB. Initially, TLBAR is cleared to zero to point at the first location. Then to address the next location, after each read and write, register TLBAR is incremented by one.

6.6 Performance Evaluation

Suppose t_m and t_p are the critical path delays of the external memory and the non-pipelined processor architecture, respectively. I is the number of input coefficients scanned from external memory for each operation. $I = 2$ for both inverse 5/3 and 9/7. Then the scan clock period τ and hence the scan frequency f of the proposed architecture can be determined by the following algorithm.

Statement4

```
case 1 : If  $t_m \geq t_p/k$  then
                    $\tau = t_m$
case 2 : Else   if  $t_p/I \cdot k \geq t_m$  then
                    $\tau = t_p/I \cdot k$
                else    $\tau = t_m$
```

In the algorithm above either case 1 or case 2 can be true. Case 2 implies the availability of a very high speed scan that can scan the two pixels required for an operation during the specified time limit given by t_p/k. If that is the case-the architecture shown in Figure 6.4.1 with it processor pipelined-the hardware utilization is 100% and the architecture is complete. Now, suppose τ_1 and τ_2 denote the scan clock periods of the architecture before and after pipelining, respectively. Then

$$\tau_1 = t_p/I. \tag{6.1}$$

And from *statement4*, case2

$$\tau_2 = t_p/I \cdot k = I \cdot \tau_1/I \cdot k = \tau_1/k. \tag{6.2}$$

The speedup factor S is then given by

$$S = \tau_1/\tau_2 = \tau_1/(\tau_1/k) = k \tag{6.3}$$

The efficiency E of k-stage pipeline is defined as

$$E = S/k = k/k = 1 \tag{6.4}$$

Thus, the architecture with pipelined processors is k times faster than the architecture with non-pipelined processors with efficiency 1.

On the other hand, case 1 implies low scanning frequency. That means the time required to scan the two pixels for an operation will take at least $2t_p/k$ seconds or two clock cycles, where t_p/k is the stage critical path delay of the pipelined processor. In that case, the proposed architecture would not only be slow but would be under utilized half of the time, since every 2 clock cycles would yield one output. To remedy this problem, the proposed architecture can be allowed to read from external memory the required 2 coefficients for an operation in parallel every clock cycle instead of one coefficient at a time, if the frequency of the pipelined architecture and the external memory scan frequency are made equal. This would require two buses instead of one to scan the external memory in the parallel scan architecture.

If the clock period τ_3 for both external memory and the pipelined architecture are made equal to t_p/k, then the speedup factor S of the pipelined parallel scan architecture as compared with the non-pipelined architecture is given by

$$S = \frac{\tau_{non}}{\tau_3} = \frac{t_p}{t_p/k} = k \tag{6.5}$$

The efficiency $E = S/k = 1$

That is the parallel scan architecture is k times faster than nonpipelined architecture with efficiency 1.

On the other hand, to compare the power consumption of the pipelined parallel and sequential scan architectures consider the following. First, since both pipelined parallel and sequential scan architectures operate with frequency k/t_p and are equal in capacitance, therefore, they consume the same power. Second, the external memory power consumption in the pipelined parallel scan architecture, $P_m(pipe)_{par}$ and that in the pipelined sequential scan architecture, $P_m(pipe)_{seq}$ can be determined as follow. If the power consumption of VLSI architecture can be estimated as

$$P = C_{total} \cdot V_o^2 \cdot f \tag{6.6}$$

where C_{total} denotes the total capacitance of the architecture, V_o is the supply voltage, and f is the clock frequency, then

$$P_m(pipe)_{par} = I \cdot C^m_{total} \cdot V_o^2 \cdot f_3 = I \cdot C^m_{total} \cdot V_o^2 \cdot k/t_p \tag{6.7}$$

$$P_m(pipe)_{seq} = C^m_{total} \cdot V_o^2 \cdot f_2 = C^m_{total} \cdot V_o^2 \cdot 1/\tau_2 = I \cdot C^m_{total} \cdot V_o^2 \cdot k/t_p \tag{6.8}$$

C^m_{total} is the total capacitance of the external memory.

Based on the above evaluations, it can be concluded that both pipelined parallel and sequential scan architectures achieve the same performance in terms of speedup, efficiency and they consume the same power.

6.7 Parallel Architecture Development

In order to best meet real-time applications 2-D DWT with demanding requirements, in this section, parallelism will be explored. The single pipelined architecture developed in the previous sections will be extended to 2- and 4-parallel pipelined architectures to achieve speedup factors of 2 and 4, respectively. First, the 2-parallel pipelined architecture for 5/3 and 9/7 will be developed followed by the 4-parallel pipelined architecture.

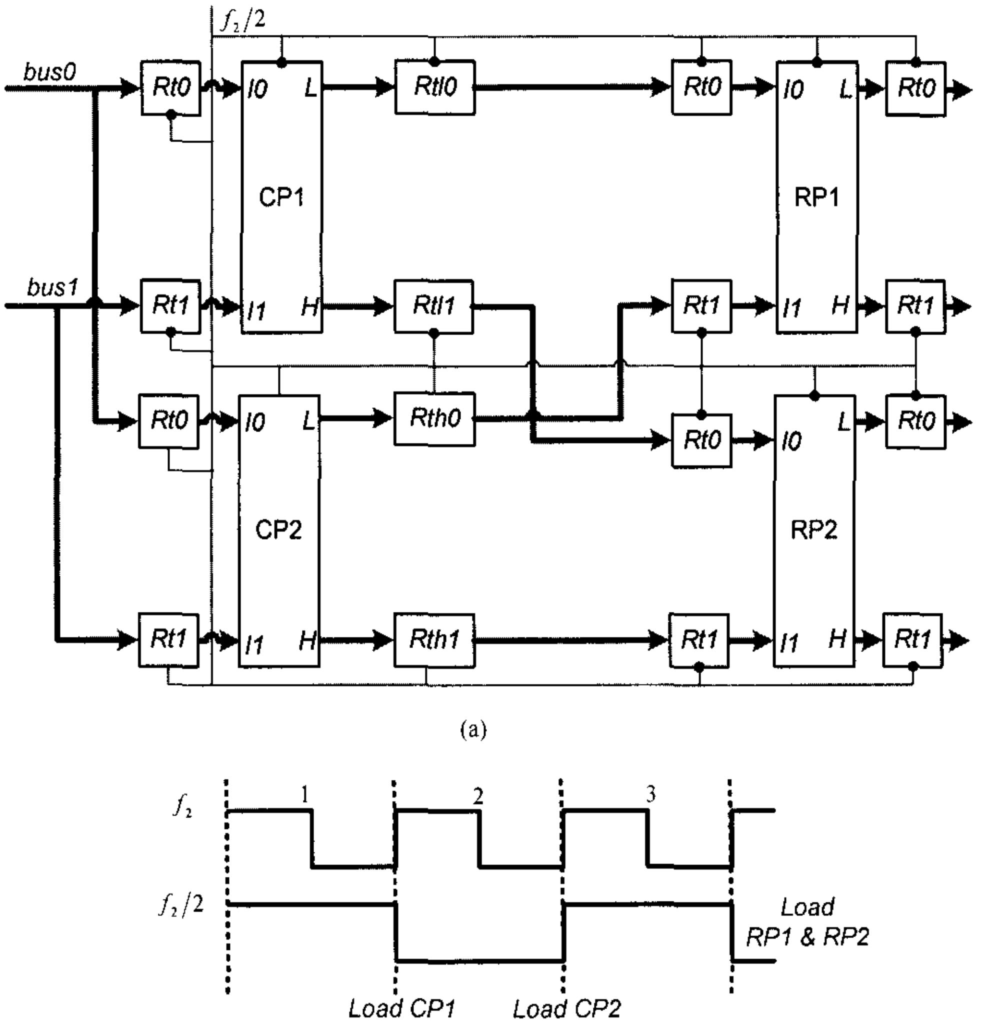
6.7.1 Proposed 2-parallel external architecture

Based on the scan methods and the DDGs for 5/3 and 9/7, the 2-parallel external architecture shown in Fig. 6.7.1 (a) is proposed for 5/3 and 9/7 and combined 5/3 and 9/7 for 2-D IDWT. The architecture consists of two k-stage pipelined column-processors labeled CP1 and CP2 and two k-stage pipelined row-processors labeled RP1 and RP2. The waveforms of the two clocks f_2 and $f_2/2$ that are used in the architecture are shown in Fig. 6.7.1 (b). The clock frequency f_2 is determined from *statement3* as

$$f_2 = 2k/t_p \tag{6.9}$$

The architecture scans the external memory with frequency f_2 and it operates with frequency $f_2/2$. Each clock cycle two new coefficients are scanned from external memory through the two buses labeled *bus0* and *bus1*. The two new coefficients are loaded into CP1 or CP2 latches *Rt0* and *Rt1* every time clock $f_2/2$ makes a negative or a positive transition, respectively. On the other hand, both RP1 and RP2 latches *Rt0* and *Rt1* load simultaneously new data from CP1 and CP2 output latches each time clock $f_2/2$ makes a negative transition.

The dataflow for 5/3 2-parallel architecture is shown in Table B.20, where CPs and RPs are assumed to be 4-stage pipelined processors. This 5/3 dataflow table is



(a)

(b)

Figure 6.7.1 (a) Proposed 2-parallel pipelined external architecture for 5/3 and 9/7 and combined 5/3 and 9/7 for 2-D IDWT (b) Waveforms of the clocks

derived based on the 9/7 scan methods shown in Figs. 6.3.2 and 6.3.4 instead of 5/3 scan method shown in Figs. 6.3.1. The reason is to show that 9/7 scan methods can be used for 5/3 as well. In addition, a unified scan method for both 9/7 and 5/3 make their control algorithms identical, which is advantageous especially in combined 5/3 and 9/7 architecture. The dataflow for 9/7 2-parallel architecture is similar, in all runs, to the 5/3 dataflow except in the first run, where RP1 and RP2 of the 9/7 architecture each would generate one output coefficient every other clock cycle, reference to clock $f_2/2$. The reason is that the first 4 coefficients of each row processed in the first run by either RP1 or RP2 of the 9/7 would require, according to the DDGs, two successive low coefficients from the first level of the DDGs labeled $Y''(2n)$ in order to carry out node 1 computations in the second level labeled $Y''(2n+1)$. In Table B.20, the output coefficients in *Rt0* of both RP1 and RP2 at cycles 19, 23, and 27 and so on represent the output coefficients of the 9/7 in the first run.

The strategy adopted for scheduling memory columns for CP1 and CP2 of the 5/3 and 9/7 2-parallel architectures, which are scanned according to the scan method shown in Figure 6.3.2, is as follow. In the first run, both 5/3 and 9/7 2-parallel architectures are scheduled for executing 4 columns of memory, two from each (A) and (B) of Figure 6.3.2 . The first two columns of Fig. 6.3.2 (A) are executed in an interleaved fashion by CP1, while the first two columns of Fig. 6.3.2 (B) are executed by CP2 also in an interleaved fashion as shown in the dataflow Table B.20. In all subsequent runs, 2 columns are scheduled for execution at a time. Each time one column from (A) of Fig. 6.3.2 will be scheduled for execution by CP1, while another from (B) will be scheduled for CP2. However, if number of columns in (A) and (B) of Fig. 6.3.2 is not equal, then the last run will consist of only one column of (A). In that case, schedule the last column in CP1 only, but its output coefficients will be executed by both RP1 and RP2. The reason is that if the last column is scheduled for execution by both CP1 and CP2, they will yield more coefficients than that can be handled by both RP1 and RP2.

On the other hand, scheduling RP1 and RP2 of 5/3 and 9/7 2-parallel architectures occurs according to scan method shown in Fig. 6.3.4. In this scheduling strategy, all rows of even and odd numbers in Fig. 6.3.4 will be scheduled for execution by RP1 and RP2, respectively. In the first run, 4 coefficients from each 2 consecutive rows will be scheduled for RP1 and RP2, whereas in all subsequent runs, two coefficients of each 2 consecutive rows will be scheduled for RP1 and RP2, as shown in Figure 6.3.4. However, if the number of columns in Figure 6.3.4 is odd, that occurs when number of columns in (A) and (B) of Fig. 6.3.2 is not equal, then the last run would require scheduling one coefficient of each 2 successive rows to RP1 and RP2.

In general, all coefficients belong to columns of even numbers in Fig. 6.3.4 will be generated by CP1 and all coefficients belong to columns of odd numbers will be generated by CP2. For example, in run 1, first, CP1 will generate two coefficients labeled L0,0 and L1,0 that belong to locations 0,0 and 1,0 in Fig. 6.3.4, while CP2 will generate coefficient H0,0 and H1,0 that belong to locations 0,1 and 1,1. Then coefficients in locations 0,0 and 0,1 are executed by RP1, while coefficients of locations 1,0 and 1,1 are executed by RP2. Second, CP1 will generate two coefficients for locations 0,2 and 1,2, while CP2 will generate two coefficients for locations 0,3 and 1,3. Then coefficients in locations 0,2 and 0,3 are executed by RP1, while coefficients in locations 1,2 and 1,3 are executed by RP2. The same process is repeated in the next two rows and so on.

In the second run, first, CP1 generates coefficients for locations 0,4 and 1,4, whereas CP2 generates coefficients for locations 0,5 and 1,5 in Fig. 6.3.4. Then coefficients in locations 0,4 and 0,5 are executed by RP1, while coefficients in locations 1,4 and 1,5 are executed by RP2. This process is repeated until the run completes. However, in the even that the last run processes only one column of (A), CP1 would generate first coefficients of locations 0,m and 1,m where m refers to the last column. Then coefficients of location 0,m is passed to RP1, while coefficient of location 1,m is passed to RP2. In the second time, CP1 would generate coefficients of locations 2,m and 3,m. Then 2,m is passed to RP1 and 3,m to RP2 and so on.

6.7.2 Modified CPs and RPs for 5/3 and 9/7 2-parallel external architecture

Each CP of the 2-parallel external architecture is required to execute two columns in an interleave fashion in the first run and one column in all other runs. Therefore, Fig. 6.5.1 should be modified as shown in Fig. 6.7.2 by adding one more stage between stages 2 and 3 for 5/3 2- parallel external architecture to allow interleaving of two columns as described in the dataflow Table B.20. Through the two multiplexers labeled *mux* the processor controls between executing 2 columns and one column. Thus, in the first run, the two multiplexers' control signal labeled s is set 1 to allow interleaving in execution and 0 in all other runs. The modified 9-stage CP for 9/7 2-parallel external architecture can be obtained by cascading two copies of Figure 6.7.2.

On the other hand, RP1 and RP2 of the proposed 2-parallel architecture for 5/3 and 9/7 are required to scan coefficients of H and L decompositions generated by CP1 and CP2 according to the scan method shown in Fig. 6.3.4. In this scan method, all rows of even numbers are executed by RP1 and all rows of odds numbers

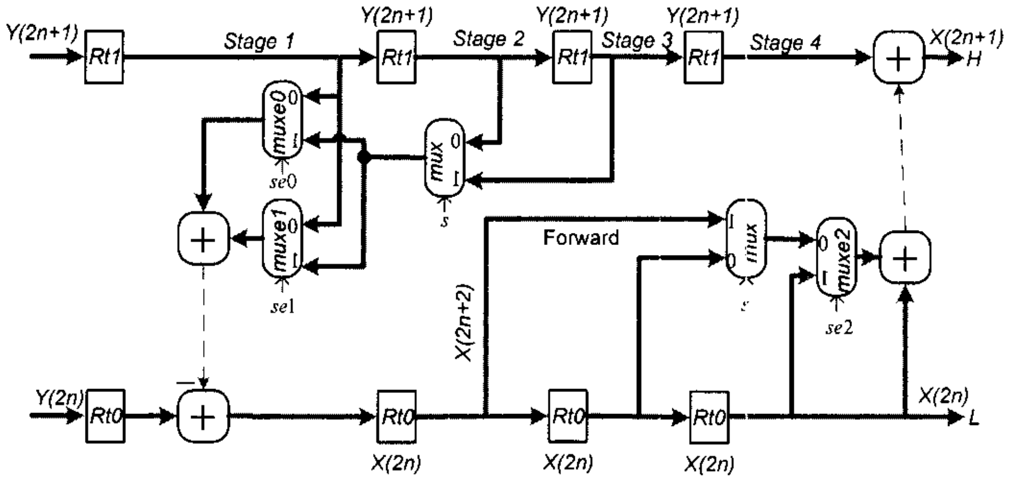


Figure 6.7.2 Modified inverse 5/3 CP for 2-parallel external architecture

are executed by RP2. That is, while RP1 is executing row0 coefficients, RP2 will be executing row1 coefficients and so on. In addition, looking at the DDGs for 5/3 and 9/7 one might immediately observe that applying the scan methods shown in Fig. 6.3.4 would require inclusion of temporary line buffers (TLBs) in RP1 and RP2 of the proposed 2-parallel external architecture as follows. In the first run, the fourth input coefficient of each row in the DDGs and the output coefficients labeled X(2) in the 5/3 DDGs and that labeled Y"(2), Y"(1), and X(0) in the 9/7 DDGs, generated by considering 4 inputs coefficients in each row, should be stored in TLBs, since they are required in the next run's computations. Similarly, in the second run, the sixth input coefficient of each row and the output coefficients labeled X(4) in the 5/3 DDGs and that labeled Y"(4), Y"(3), and X(2) in the 9/7 DDGs generated by considering 2 inputs coefficients in each row, should be stored in TLBs. Accordingly, 5/3 would require addition of 2 TLBs each of size N, whereas 9/7 would require addition of 4 TLBs each of size N. However, since 2-parallel architecture consists of two RPs, each 5/3 RP will has 2 TLBs each of size N/2 and each 9/7 RP will has 4 TLBs each of size N/2 as shown in Fig. 6.7.3. Figure 6.7.3 (a) represents the 5/3 modified RP, while both (a) and (b) represent the 9/7 modified RP for 2- parallel architecture.

To have more insight into the two RPs operations, the dataflow for 5/3 RP1 is given in Table 6.4 for first and second runs. Note that stage 1 input coefficients in Table 6.4 are exactly the same input coefficients of RP1 in Table B.20. In the first run, TLBs are only written, but in the second run and in all subsequent runs, TLBs are

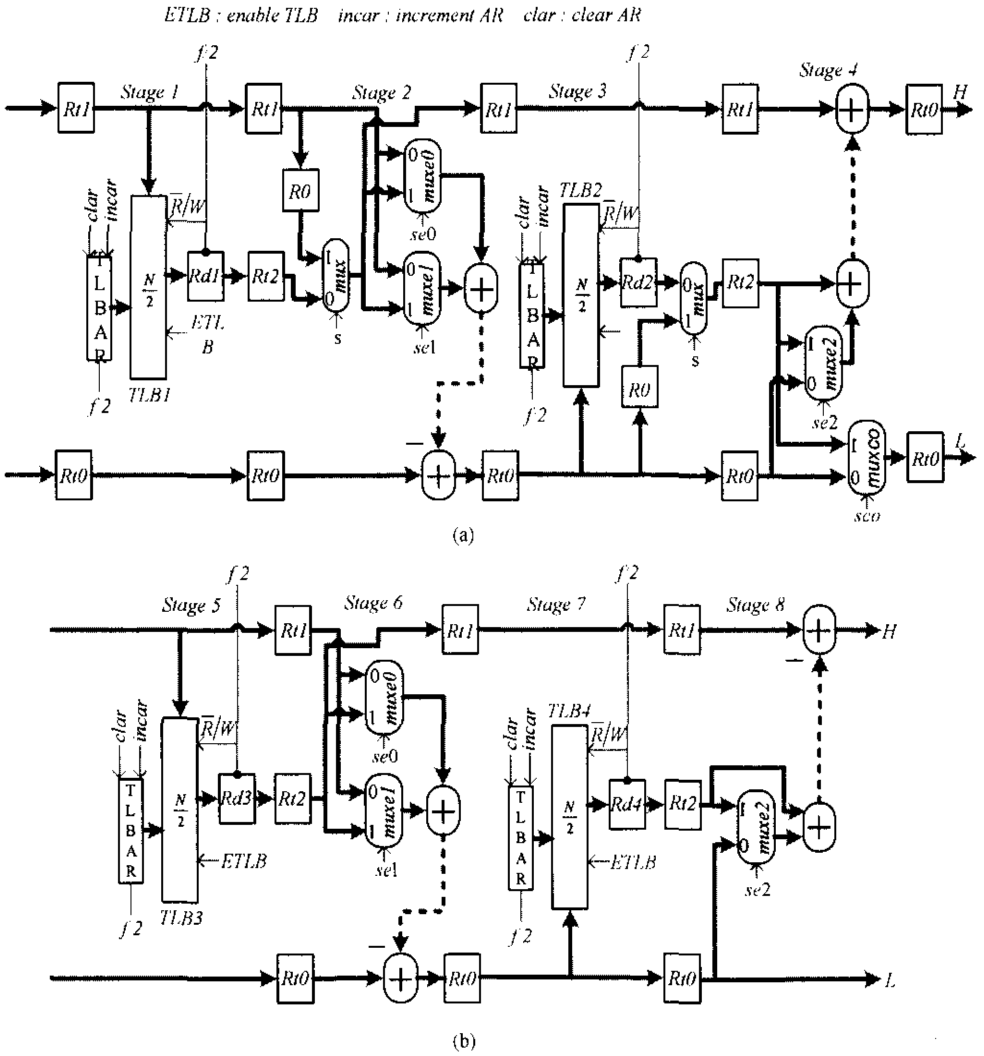


Figure 6.7.3 Modified RP for 2-parallel architecture (a) 5/3 (a, b) 9/7

read in the first half cycle and written in the second half cycle. In the cycle 15, Table 6.4 shows that coefficients H0,1 is stored in the first location of TLB1, while coefficient H2,1 is stored in the second location in cycle 19 and so on. Run 2 starts at cycle 27. In cycle 28, the first location of TLB1, which contains coefficients H0,1 is read during the first half cycle and is loaded into *Rd1* by the positive transition of the cycle, whereas coefficient H0,2 is written into the same location in the second half cycle. Then, the negative transition of clock cycle 10 transfers contents of *Rd1* to *Rt2* in stage 2.

In Figure 6.7.3, the control signal, *s*, of the two multiplexers' labeled mux is set 1 during run 1 to pass *R0* of both stages 2 and 3, whereas in all other runs, it is set 0 to

Table 6.4 Dataflow of the 5/3 RP1

	CK	RP1 input latches STAGE 1			STAGE 2				STAGE 3				STAGE 4			RP1output latches	
	f_2	Rt0	Rt1	TLB1	Rt0	Rt2	Rt1	R0	Rt0	Rt1	R0	TLB2	Rt0	Rt1	Rt2	Rt0	Rt1
RUN 1	11	L0,0	H0,0													-----	-----
	13	L0,1	H0,1		L0,0	----	H0,0									-----	-----
	15	L2,0	H2,0	H0,1	L0,1	----	H0,1	H0,0	X0,0	---	----					-----	-----
	17	L2,1	H2,1		L2,0	----	H2,0	-----	X0,2	H0,0	X0,0		X0,0	----	----	-----	-----
	19	L4,0	H4,0	H2,1	L2,1	----	H2,1	H2,0	X2,0	-----	-----	X0,2	X0,2	H0,0	X0,0	X0,0	-----
	21	L4,1	H4,1		L4,0	----	H4,0	-----	X2,2	H2,0	X2,0		X2,0	-----	-----	X0,2	X0,1
	23	L6,0	H6,0	H4,1	L4,1	----	H4,1	H4,0	X4,0	-----	-----	X2,2	X2,2	H2,0	X0,2	X2,0	-----
	25	L6,1	H6,1		L6,0	----	H6,0	-----	X4,2	H4,0	X4,0		X4,0	-----	------	X2,2	X2,1
RUN 2	27	L0,2	H0,2	H6,1	L6,1	----	H6,1	H6,0	X6,0	-----	-----	X4,2	X4,2	H4,0	X4,0	X4,0	-----
	29	L2,2	H2,2	H0,2	L0,2	H0,1	H0,2	-----	X6,2	H6,0	X6,0		X6,0	-----	------	X4,2	X4,1
	31	L4,2	H4,2	H2,2	L2,2	H2,1	H2,2	-----	X0,4	H0,1	-----	X6,2	X6,2	H6,0	X6,0	X6,0	-----
	33	L6,2	H6,2	H4,2	L4,2	H4,1	H4,2	----	X2,4	H2,1	-----	X0,4	X0,4	H0,1	X0,2	X6,2	X6,1
	35	----	-----	H6,2	L6,2	H6,1	H6,2	-----	X4,4	H4,1	-----	X2,4	X2,4	H2,1	X2,2	X0,4	X0,3
	37	----	-----		-----	-----	------	-----	X6,4	H6,1	-----	X4,4	X4,4	H4,1	X4,2	X2,4	X2,3
	39	----	-----		-----	-----	------	----	------	-----	-----	X6,4	X6,4	H6,1	X6,2	X4,4	X4,3
	41	----	-----		-----	-----	------	----	------	-----	-----		----	-----	------	X6,4	X6,3

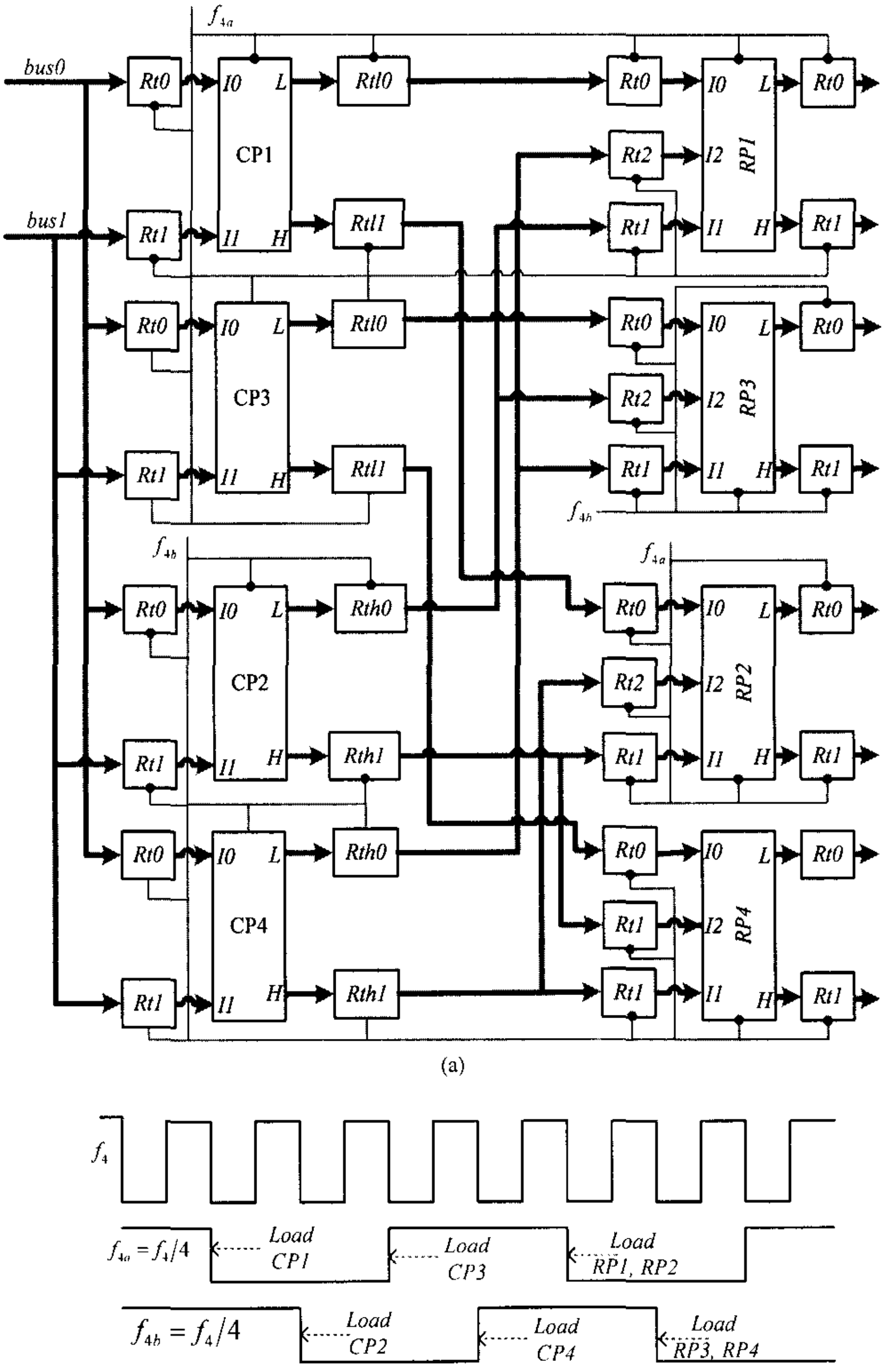
pass coefficients read from TLB1 and TLB2.

6.8 Proposed 4-parallel external architecture

To further increase speed of computations twice as that of the 2-parallel architecture, the 2-parallel architecture is extended to 4-parallel architecture as shown in Fig. 6.8.1 (a). This architecture is valid for 5/3, 9/7, and combined 5/3 and 9/7. It consists of 4 *k*-stage pipelined CPs and 4 *k*-stage pipelined RPs. The waveforms of the 3 clocks f_4, f_{4a}, and f_{4b} used in the architecture are shown in Fig. 6.8.1 (b). The frequency of clock f_4 is determined from *statement3* as

$$f_4 = 4k/t_p \tag{6.10}$$

The architecture scans the external memory with frequency f_4 and it operates with frequency f_{4a} and f_{4b}. Every time clock f_{4a} makes a negative transition CP1 loads into its input latches Rt0 and Rt1 two new coefficients scanned from external memory through the buses labeled *bus0* and *bus1*, whereas CP3 loads every time clock f_{4a} makes a positive transition. CP2 and CP4 load every time clock f_{4b} makes a negative and a positive transition, respectively. On the other hand, both RP1 and RP2 load simultaneously new data into their input latches *Rt0* and *Rt1* each time clock f_{4a} makes a negative transition, whereas RP3 and RP4 loads each time clock f_{4b} makes a negative transition.



(a)

(b)

Figure 6.8.1 (a) Proposed 2-D IDWT 4-parallel pipelined external architecture for 5/3 and 9/7 and combined 5/3 and 9/7 (b) Waveforms of the clocks

The dataflow for 4-parallel 5/3 external architecture is given in Table B.21, where CPs and RPs are assumed to be 3- and 4-stage pipelined processors, respectively. The dataflow table for 4-parallel 9/7 external architecture is similar in all runs to the 5/3 dataflow except in the first run, where RPs of the 9/7 architecture, specifically RP3 and RP4 generate a pattern of output coefficients different from that of the 5/3. RP3 and RP4 of the 9/7 architecture generate every clock cycle, reference to clock f_{4b}, two output coefficients as follows. Suppose, at cycle number n the first two coefficients X(0,0) and X(1,0) generated by RP3 and RP4, respectively, are loaded into output latch Rt0 of both processors. Then, in cycle $n+1$, RP3 and RP4 generate coefficients X(2,0) and X(3,0) followed by coefficients X(4,0) and X(5,0) in cycle $n+1$ and so on. Note that these output coefficients are the coefficients generated by both RP1 and RP2 in Table B.21.

The strategy used for scheduling memory columns for CPs of the 5/3 and 9/7 4-parallel architecture, which resemble the one adopted for 2-parallel architecture, is as follow. In the first run, both 5/3 and 9/7 4-parallel architecture will be scheduled to execute 4 columns of memory, two from (A) and the other two from (B), both of Fig. 6.3.2. Each CP will be assigned to execute one column of memory coefficients as illustrated in the first run of the dataflow shown in Table B.21, whereas in all subsequent runs, 2 columns at a time will be scheduled for execution by the 4 CPs. One column from Fig. 6.3.2 (A) will be assigned to both CP1 and CP3, while the other from Fig. 6.3.2 (B) will be assigned to both CP2 and CP4 as shown in the second run of Table B.21. However, if number of columns in (A) and (B) of Fig. 6.3.2 is not equal, then the last run will consist of only one column of (A). In that case, schedule the last column's coefficients in both CP1 and CP3 as shown in the third run of Table B.21, since an attempt to execute the last column using 4 CPs would result in more output coefficients been generated than that can be handled by the 4 RPs.

On the other hand, scheduling rows coefficients for RPs, which take place according to scan method shown in Fig. 6.3.4, can be understood by examining the dataflow shown in Table B.21. In cycle 17 and 18, the first two rows coefficients are scheduled for RPs as shown in Table B.21, while CPs generate coefficients of the next two rows, row2 and row3. Table B.21 shows that the first 4 coefficients of row 0 are scheduled for execution by RP1 and RP3, while, the first 4 coefficients of row 1 are scheduled for RP2 and RP4. In addition, note that all coefficients generated by CP4, which belong to column 3 in Fig. 6.3.4, are required in the second run's computations, according to the DDGs. Therefore, this would require inclusion of a TLB of size *N/4* in each of the 4 RPs to store these coefficients. The second run, however, requires these coefficients to be stored in the 4 TLBs as follows. Coefficients H0,1 and H1,1 generated by CP4 in cycle 16 should be stored in the first location of TLB of RP1 and RP2, respectively. These two coefficients would be passed to their respective TLB through the input latches of RP1 and RP2 labeled Rt2, as shown in cycle 17 of Table B.21. Whereas, coefficients H2,1 and H3,1 generated by CP4 at cycle 20 should be stored in the first location of TLB of RP3 and RP4, respectively. These two coefficients are passed to their respective TLB through the input latches of RP3 and RP4 labeled Rt1, as shown in cycle 22 of Table B.21. Similarly, coefficients H4,1 and H5,1 generated by CP4 at cycle 24 should be stored in the second location of TLB of RP1 and RP2, respectively, and so on. These TLBs are labeled TLB1 in Fig. 6.8.1 (a).

6.8.1 Column and row processors for 5/3 and 9/7 4-parallel external architecture

The 5/3 and the 9/7 processors datapath architectures shown in Figs. 6.5.1 and 6.5.2 were developed assuming the processors scan external memory either row by row or column by column. However, CPs and RPs of the 4-parallel architecture are required to scan external memory according to scan methods shown in Figs. 6.3.2 and 6.3.4, respectively. The 4-parallel architecture, in addition, introduces the requirement for communications among the processors in order to accomplish their task. Therefore, the processors datapath architectures shown in Figs. 6.5.1 and 6.5.2 should be modified according to the scan methods and the communications requirements so that they fit into the 4-parallel's processors. Thus, in the following, the modified 4 CPs will be developed first followed by the 4 RPs.

6.8.2 Modified CPs for 4-parallel architecture

The 4 CPs of the 4-parallel architecture each is required in the first run to execute one column at a time. That means the first run requires no modifications of the 5/3 and 9/7 datapath architectures shown in Figs. 6.5.1 and 6.5.2. However, in all subsequent runs, each two processors (CP1 and CP3 or CP2 and CP4) are assigned to execute one column together, which requires interactions between the two processors to accomplish the required task. Therefore, both CPs 1 and 3, similarly, CPs 2 and 4 should be modified as shown in Fig. 6.8.2 to allow communications. The two processors communicate or interact through the paths (buses) labeled *P1, P2, P3,* and *P4.* Fig. 6.8.2 shows modified 5/3 CPs 1 and 3 which is identical to CPs 2 and 4. Fig. 6.8.2 also represents the first 3 stages of 9/7 CPs 1 and 3 (and 9/7 CPs 2 and 4) and the remaining stages are identical to stages 1 to 3. Note that since the first 3 stages of 5/3 and 9/3 are similar in structure, the 5/3 processor can be easily incorporated into 9/7 processor to obtain the combined 5/3 and 9/7 processor for 4-parallel architecture.

The control signal, *s* of the 4 multiplexers, labeled *mux* is set 0 in the first run to

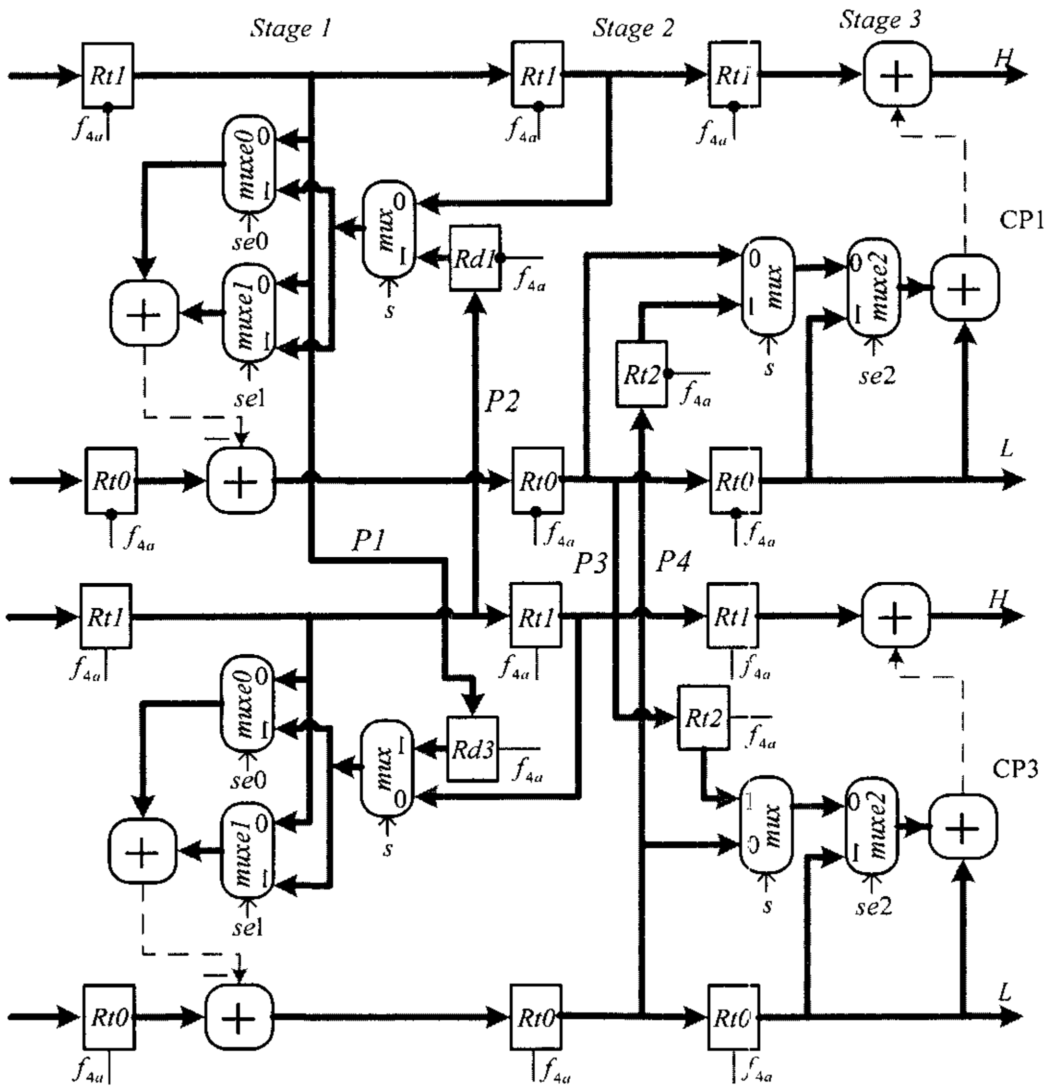
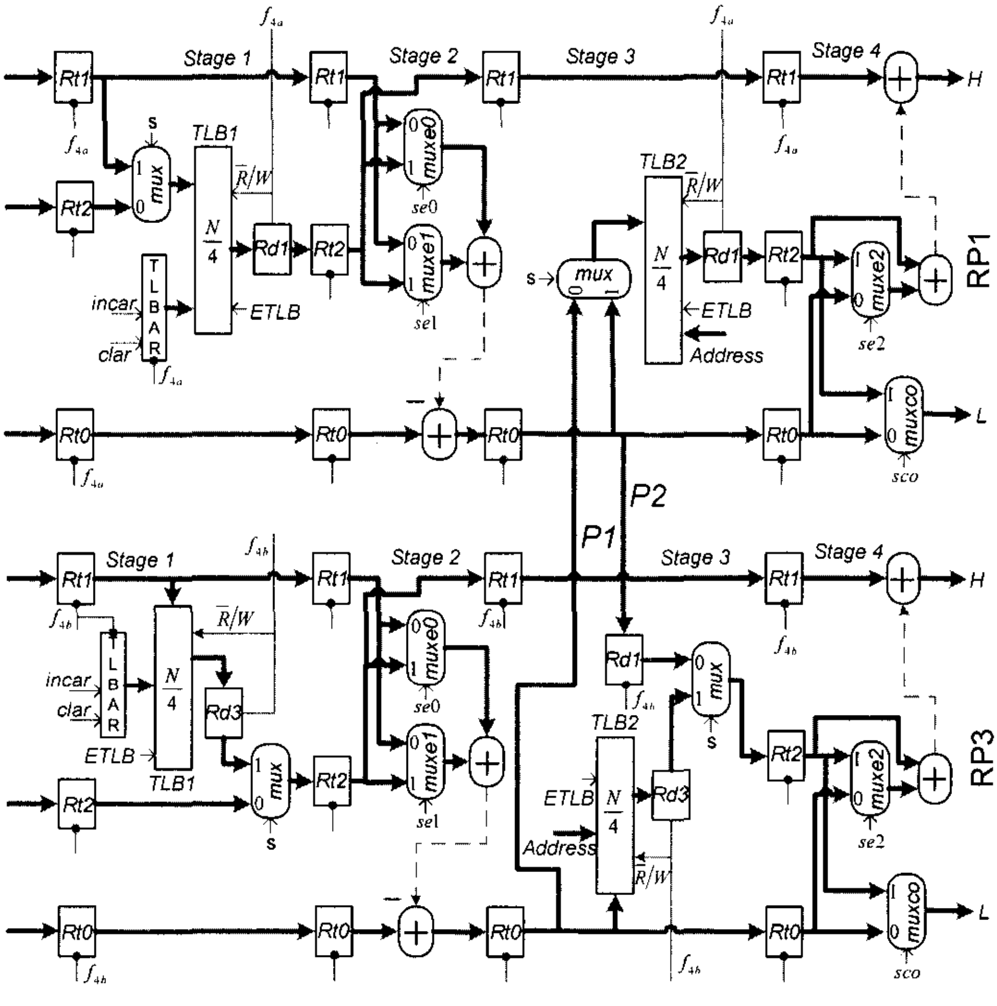


Figure 6.8.2 Modified 5/3 CPs 1 & 3 for 4-parallel architecture

allow each processor to execute one column and 1 in all other runs to allow execution of one column by two processors.

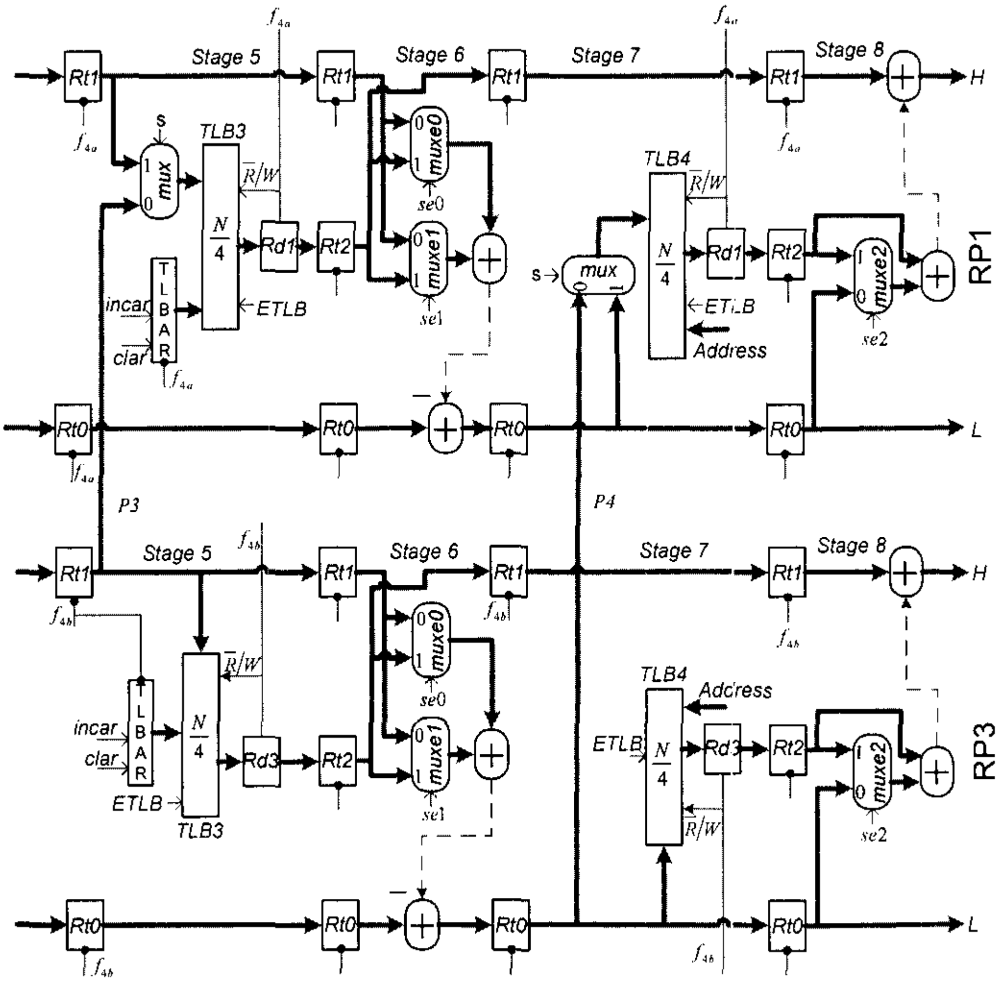
6.8.3 Modified RPs for 4-parallell architecture

In section 6.7.2, it has been pointed out the reasons for including TLBs in the two RPs of the 2-parallel architecture. For the same reasons, it is also necessary to include TLBs in the 4 RPs of the 4-parallel architecture, as shown in Figures 6.8.3 (a) and (a,b) for 5/3 and 9/7, respectively. The processor datapath for both RP1 and RP3, which is also identical to the processor datapath of both RP2 and RP4, are drawn together in Figs.6.8.3 (a) and (a,b) for 5/3 and 9/7, respectively, since in the first run,



(a)

Figure 6.8.3 (a) Modified 5/3 RPs 1 and 3 for 4-parallel external architecture



(b)

Figure 6.8.3 (a, b) Modified 9/7 RPs 1 and 3 for 4-parallel external architecture

both processors are required to execute together the first 4 coefficients of each row. Which implies interactions between the two processors during the computations and that take place through the paths (buses) labeled *P1, P2, P3,* and *P4.* However, in all subsequent runs, according to the scan method shown in Fig. 6.3.4, each RP will be scheduled to execute each time two coefficients of a row as shown in cycles 37 and 38 of Table B.21. The advantage of this organization is that the total size of the TLBs does not increase from that of the single pipelined architecture, when it is extended to 2- and 4- parallel architecture.

In the first run, all TLBs in Fig. 6.8.3 will be written only, whereas, in all other runs, the same location of a TLB will be read in the first half cycle and written in the second half cycle with respect to clock f_{4a} or f_{4b}.

The control signal, *s* of the six multiplexers, labeled *mux* in Fig. 6.8.3, is set 0 in the first run to allow in the RP1, coefficient coming through path 0 of each multiplexer to be stored in its respective TLB, whereas in the RP3, it allows contents of Rt2 and Rd1 in stages 1 and 3, respectively, to be passed to the next stage. In all subsequent runs, *s* is set 1 to pass coefficients read from TLBs to next stage.

Note that during run 2 all RPs execute independently with no interactions among them. In addition, in the first run, if the first coefficient generated by stage 2 of RP3 is stored in TLB2 of RP1, then the second coefficient should be stored in TLB2 of RP3 and so on. Similarly, TLB1, TLB3, and TLB4 of both RP1 and RP3 are handled. Furthermore, during the whole period of run 1, the control signals of the three extension multiplexers labeled *muxe0, muxe1,* and *muxe2* in RP1 should be set 0, according to Table 6.2, whereas those in RP3 should be set normal as shown in the second line of Table 6.2, since RP3 will execute normal computations during the period. However, in the second run and in all subsequent runs except the last run, the extension multiplexers control signals in all RPs are set normal. Moreover, the multiplexers labeled *muxco* in stage 4 is only needed in the combined 5/3 and 9/7 architecture, otherwise, it can be eliminated and Rt2 output can be connected directly to Rt0 input of the next stage in case of 9/7, whereas in 5/3, Rt0 is connected directly to output latch Rt0. In the combined architecture, signal *sco* of *muxco* is set 0 if the architecture is to perform 5/3; otherwise, it is set 1 if the architecture is to perform 9/7.

6.9 performance evaluation

In order to evaluate performance of the two proposed parallel pipelined architectures in terms of speedup and throughput as compared with single pipelined architecture consider the following. Assume subbands HH, HL, LH, and LL of each level are equal in size. The dataflow for single pipelined architecture shown in Table B.18 shows that $\rho_1 = 20$ clock cycles are needed to yield the first output. Then, the total number of output coefficients in the first run of the J$^{\text{th}}$ level reconstruction can be estimated with the help of Table B.18 as

$$N/2^{J-1} \tag{6.11}$$

and the total number of cycles in run1 is given by

$$2N/2^{J-1} \tag{6.12}$$

The total time, $T1$, required to yield n pairs of output coefficients for the Jth level reconstruction by single pipelined architecture can be estimated as

$$T1 = \left(\rho_1 + 2N/2^{J-1} + 2\left(n - 1/2\,N/2^{J-1}\right)\right)\tau_1 = \left(\rho_1 + N/2^{J-1} + 2n\right)t_p/2k \tag{6.13}$$

On the other hand, the dataflow Table B.20 for the 2-parallel pipelined architecture shows that $\rho_2 = 19$ clock cycles are needed to yield the first 2 output coefficients. Then, the total numbers of paired output coefficients in the first run of the Jth level reconstruction can be estimated as

$$3/2\,N/2^{J-1}. \tag{6.14}$$

The total number of 2-paired output coefficients is given by

$$3/4\,N/2^{J-1} \tag{6.15}$$

and the total number of cycles in run 1 is

$$2\,N/2^{J-1} \tag{6.16}$$

Note that the total number of paired output coefficients of the first run in each level of reconstruction starting from the first level can be written as

$$3/2\,N, 3/2\,N/2, 3/2\,N/4, \ldots\ldots\ldots\ldots, 3/2\,N/2^{J-1} \tag{6.17}$$

where the last term is Eq (6.14).

The total time, *T2*, required to yield n pairs of output coefficients for the Jth level reconstruction of an *NxM* image on the 2-parallel architecture can be estimated as

$$T2 = \left(\rho_2 + 2N/2^{J-1} + 2(n/2 - 3/4\,N/2^{J-1})\right)\tau_2 \tag{6.18}$$

$$T2 = \left(\rho_2 + 1/2\,N/2^{J-1} + n\right)t_p/2k \tag{6.19}$$

The term $2\left(n/2 - 3/4\,N/2^{J-1}\right)$ in (6.18) represents the total number of cycles of run 2 and all subsequent runs.

The speedup factor, *S2,* is then given by

$$S2 = \frac{T1}{T2} = \frac{\left(\rho_1 + N/2^{J-1} + 2n\right)t_p/2k}{\left(\rho_2 + 1/2\,N/2^{J-1} + n\right)t_p/2k} \tag{6.20}$$

For large n, the above equation reduces to

$$S2 = \frac{2(1/2\,N/2^{J-1} + n)}{(1/2\,N/2^{J-1} + n)} \cong 2 \tag{6.21}$$

That means the 2-parallel architecture is 2 times faster than the single pipelined architecture.

Similarly, the dataflow Table B.21 for the 4-parallel pipelined architecture shows that $\rho_4 = 33$ clock cycles are needed to yield the first two output coefficients. In addition, with the help of the dataflow table of the 4-paralell architecture it can be estimated that both RP1 and RP2, in the first run of each level reconstruction, yield $(N/2^{J-1})/2$ pairs of output coefficients, while both RP3 and RP4 yield $N/2^{J-1}$ pairs of output coefficients, a total of $3/2\,N/2^{J-1}$ pairs of output coefficients. The total number of cycles in run 1 is then given by

$$4(N/2^{J-1})/2 \tag{6.22}$$

Thus, the total time, *T4*, required to yield n pairs of output coefficients for the Jth level reconstruction of an *NxM* image on the 4-parallel architecture can be estimated as

$$T4 = \left(\rho_4 + 2N/2^{J-1} + 2\left(n - 3/2\,N/2^{J-1}\right)/2\right)\tau_4 \tag{6.23}$$

$$T4 = \left(\rho_4 + 2N/2^{J-1} + \left(n - 3/2\,N/2^{J-1}\right)\right)t_p/4k \tag{6.24}$$

$$T4 = \left(\rho_4 + 1/2\,N/2^{J-1} + n\right)t_p/4k \tag{6.25}$$

The term $(n - 3/2\,N/2^{J-1})$ represents the total cycles of run 2 and all subsequent runs.

The speedup factor, *S4,* is then given by

$$S4 = \frac{T1}{T4} = \frac{\left(\rho_1 + N/2^{J-1} + 2n\right)t_p/2k}{\left(\rho_4 + 1/2\,N/2^{J-1} + n\right)t_p/4k} \tag{6.26}$$

For large n it reduces to

$$S4 = \frac{4(1/2\,N/2^{J-1} + n)}{(1/2\,N/2^{J-1} + n)} = 4 \tag{6.27}$$

Thus, the 4-parallel architecture is 4 times faster than the single pipelined architecture.

The throughput, *H,* which can be defined as number of output coefficients generated per unit time, can be written for each architecture as

$$H(\sin gle) = n/(\rho_1 + N/2^{J-1} + 2n)t_p/2k \qquad (6.28)$$

The maximum throughput, H^{max}, occurs when n is very large ($n \to \infty$), thus,

$$H^{\max}(\sin gle) = H(\sin gle)_{n\to\infty} \cong nkf_p/(1/2\, N/2^{J-1} + n) \qquad (6.29)$$

$$H(2 - parallel) = n/(\rho_2 + 1/2\, N/2^{J-1} + n)t_p/2k \qquad (6.30)$$

$$H^{\max}(2 - parallel) = H(2 - parallel)_{n\to\infty} \cong 2knf_p/(1/2\, N/2^{J-1} + n) \qquad (6.31)$$

$$H(4 - parallel) = n/(\rho_4 + 1/2\, N/2^{J-1} + n)t_p/4k \qquad (6.32)$$

$$H^{\max}(4 - parallel) = H(4 - parallel)_{n\to\infty} \cong 4knf_p/(1/2\, N/2^{J-1} + n) \qquad (6.33)$$

Thus, the throughputs of the 2-parallel and the 4-parallel pipelined architectures have increased by factors of 2 and 4, respectively, as compared with the single pipelined architecture.

6.10 Conclusions

In this chapter, to show the effectiveness of the approach adopted for developing forward architectures in chapters 3 and 4, the architectures for 2-dimensional inverse discrete wavelet transform (2-D IDWT) for 5/3 and 9/7 were developed. First, a high-speed single pipelined inverse architecture including its column-processor (RP) and row-processor (CP) were developed. Then, the single pipelined architecture is extended to 2-parallel and 4-parallel to achieve speedup factors of 2 and 4, respectively, according to the evaluation given in section 6.9. The advantage of the single pipelined architecture developed here is that it only requires a total temporary line buffer (TLBs) of sizes 2N and 4N for 5/3 and 9/7, respectively, and the TLB requirement does not increase when it extended to parallel architecture. The interleaving technique is utilized to speedup the computations by allowing the two processors to work in parallel earlier during the computations and to reduce TLB requirement between CP and RP to a few registers.

CHAPTER 7

EXPERIMENTAL RESULTS

7.1 Performance analysis

In chapter 3, two scan methods were developed for 9/7 algorithm. The first scan method shown in Figure 3.5.1 can be used for both 9/7 and 5/3 algorithms. Architecture developed based on this scan method will not yield any output coefficients in the first run. However, starting from the second run its dataflow is same as that of the 5/3 dataflow shown in Table B.6. On the other hand, the 9/7 architecture developed based on the second scan method shown in Figure 3.5.3 will yield output coefficients starting from the first run, as illustrated in the dataflow shown in Table B.2(a). This might give the impression that the second scan method performs better than the first scan method. To show that both scan methods achieve the same performance in terms of the total number of cycles and throughput, consider the following. From the RP and the CP of the 9/7 shown in Figures 3.8.8(a) and 3.8.4(a), respectively, which are based on the scan method shown in Figure 3.5.1, it can be shown that $(\rho_1 + N)$ cycles are needed to yield the first pair of output coefficients. The remaining (n-1) pairs of output coefficients, which will be produced according to Table B.6, would require (n-1) cycles. Thus, the total time *T1* required to yield n pairs of output coefficients for j-level decomposition of an NxM image is given by

$$T1 = (\rho_1 + N + (n-1))\tau_1 \tag{7.1}$$

where $\tau_1 = t_p/k$ is the clock period. The throughput H is given by

$$H = n/(\rho_1 + N + (n-1))\tau_1 \tag{7.2}$$

The maximum throughput, $H^{\max}$ occurs when n is very large ($n \to \infty$), thus,

$$H^{\max} = H_{n\to\infty} = nkf_p/N + n \tag{7.3}$$

On the other hand, Table B.2 of the architecture based on the scan method shown in Figure 3.5.3, indicates $\rho_2 = 23$ cycles are needed to yield the first pair of output coefficients. In addition, the total number of paired output coefficients and the total number of cycles in the first run are N and $2N\text{-}2$, respectively. Thus, the total time *T2* required to yield n pairs of output coefficients for j-level decomposition is estimated as

$$(\rho_2 + (2N-2) + (n-N))\tau_2 \qquad (7.4)$$

where $\tau_2 = t_p/k$. The throughput H is given by

$$H = n/(\rho_2 + 2N + (n-N))\tau_2 \qquad (7.5)$$

$$H^{\max} = H_{n\to\infty} = nkf_p/N + n \qquad (7.6)$$

Similar analysis also can be carried out for intermediate architectures based on the scan methods shown in Figures 3.7.1 (a) and (b). Equations 7.1, 7.3, 7.4, and 7.6, show that the architectures developed based on both scan methods give the same performance in terms of the number of clock cycles and throughput, if $\tau_1 = \tau_2$. However, the hardware and the control complexities of the architecture based on the second scan method, as indicated in Figures 3.8.4(b) and 3.8.8(b) are more complex than the one based on the first scan method shown in Figures 3.8.4(a) and 3.8.8(a). The situation becomes even more complex and worse when the architecture is extended to parallel. Furthermore, the implementation results in Figurers C.3.3 and C.4.3 show the speed advantage of the first scan method. Figure C.3.3 shows that the first scan method architecture operates with frequency 147.95 MHz, while Figure C.4.3 shows the second scan method architecture operates with frequency 136.04 MHz. For these reasons, therefore, the first scan method is adopted for all parallel architectures developed in chapter 4.

7.2 Performance evaluations and comparisons

This section evaluates and compares architectures developed in this research with most recent architectures in the literatures. The architectures are evaluated in terms of hardware complexity, hardware utilization, computing complexity, and control complexity. Hardware complexity is measured by the number of multipliers, the number of adders, the total size of the line buffer, and the complexity of the control circuits [40]. Computing complexity for 2-D DWT is estimated by the number of clock cycles required to scan an NxM image for j levels of decomposition.

Table 7.1 shows the performance comparison results. The line-based architecture presented in [1] requires a line buffer of size 5.5*N* implemented in two-port RAM. Besides, its critical path delay is large, *4Tm + 8Ta.* Whereas the proposed architectures use single-port RAMs of sizes 3*N* and *4N* for overlapped and nonoverlapped architectures, respectively.

Flipping structure [2] introduces a new method to shorten the critical path of the lifting-based architecture to one multiplier delay but requires a line buffer of size *11N* [43]. In [21], a modified view of the flipping structure, which shortens the critical path delay to one multiplier and reduces the size of the line buffer required to 4*N*, is presented. In fact, [2, 21] have only introduced a method not an architecture, which aims at shorting the critical path delay of lifting- based to one multiplier delay. However, this issue becomes less important after the fact that scale factors and coefficients of the 9/7 filter can be implemented in hardware using only two adders as illustrated in [23]. The proposed overlapped and nonoverlapped architectures require a total line buffer of size *3N* and 4N, respectively. However, note that by adding a line buffer of size N in the nonoverlapped architecture, the power consumption has been

Table 7.1 Comparisons of several1-level (9/7) 2-D DWT architectures

Architecture	Multi	Adders	Line buffer	Computing Time	Critical Path
Generic RAM-based [1]	10	16	5.5 N	$2(1-4^{-j})NM$	4Tm +8Ta
Flipping [2]	10	16	11N	$2(1-4^{-j})NM$	Tm
Chao [60]	6	8	5.5N	$2(1-4^{-j})NM$	Tm
PLSA [21]	12	16	4N	N/A	Tm
Bing [43]	6	8	5.5N	$2(1-4^{-j})NM$	Tm
Lan [29]	12	12	6N	$2(1-4^{-j})NM$	Tm
Jain [61]	9	16	10N	$2(1-4^{-j})NM$	Tm+Ta
Cheng [22](2-parallel)	18	32	5.5N	$(1-4^{-j})NM$	N/A
FIDF [62](2-parallel)	24	32	5N	$(1-4^{-j})NM$	Tm+2Ta
Proposed (overlapped)	10	16	3N	$2(1-4^{-j})NM$	Tm+2Ta
Proposed (nonoverlapped)	10	16	4N	$2(1-4^{-j})NM$	Tm+2Ta
Proposed (2-parallel)	18	32	3N	$(1-4^{-j})NM$	Tm+2Ta
Proposed (4-parallel)	36	64	3N	$1/2(1-4^{-j})NM$	Tm+2Ta
Prop. (2-parallel intermediate)	18	32	3N	$(1-4^{-j})NM$	Tm+2Ta
Prop. (3-parallel intermediate)	28	48	3N	$2/3(1-4^{-j})NM$	Tm+2Ta
Prop. (single pipelined inverse)	10	16	4N	$2(1-4^{-j})NM$	Tm+2Ta
Proposed (2-parallel inverse)	18	32	4N	$(1-4^{-j})NM$	Tm+2Ta
Proposed (4-parallel inverse)	36	64	4N	$1/2(1-4^{-j})NM$	Tm+2Ta

reduced to minimum. Thus, the nonoverlapped architecture could be a very efficient alternative in applications where power consumption is a serious concern.

In [43, 60], by reordering the lifting-based DWT of the 9/7 filter, the critical path of the pipelined architectures have been reduced to one multiplier delay but requires a total line buffer of size 5.5*N*. However, [43] requires two row processors and [60] requires 4 processing elements (PEs), two in each horizontal and vertical processors, to perform prediction lifting and update lifting. In addition, both [43, 60] require the use of real multipliers with long delay that cannot be implemented by using arithmetic shift method [23]. The architecture proposed in [29] achieves a critical path of one multiplier delay using very large number of pipeline registers. In addition, it requires a total line buffer of size 6*N*. In the efficient pipelined architecture [61], a critical path delay of Tm+Ta is achieved through optimized data flow graph but requires a total line buffer of size 10*N*.

On the other hand, the architectures proposed in [22, 62], like the proposed 2-parallel architectures, achieve a speedup factor of 2. However, [62], the deeply parallel architecture requires a total line buffer of size 5*N*, whereas [22] requires a total line buffer of size 5.5*N*. The advantage of the parallel architectures developed in this research is that the total line buffer does not increase from that of the proposed single pipeline architectures when the degree of parallelism is increased. In addition, the architectures proposed in this research are real architectures, which compared with architectures listed in Table 7.1 are accurate and complete.

7.3 Experimental results and comparisons

To further verify that the architectures developed here are accurate, efficient and practically can be implemented, we have chosen for FPGA implementation five architectures, which are representative of the other architectures: the 5/3 forward overlapped scan architecture shown in Figure 3.6.1, the inverse 5/3 architecture shown in Figure 6.4.1, two 9/7 forward overlapped architectures, one is based on the scan method shown in Figure 3.5.1 and the other is based on the scan method shown in Figure 3.5.3, and the 5/3 2-parallel architecture shown in Figure 4.2.1. First, the Verilog HDL descriptions for the five architectures are developed and then implemented on Altera FPGA with 16-bit word length for internal datapath. The

Verilog HDL program codes for the five architectures are named as module "decorrelate_processor" for forward 5/3 architecture, module "reconst_processsor" for inverse 5/3 architecture, module "decrrelation2_processor9_7" for the first 9/7 architecture based on the scan method of Figure 3.5.1, module "decorelation_processor9_7" for the second 9/7 architecture based on the scan method of Figure 3.5.3, and module "two_parallel_DWT" for the 5/3 2-parallel architecture. The Verilog descriptions of the five architectures are compiled and synthesized on Altera FPGA Stratix II device EP2515F484C3 using Quartus II CAD software. This software provides automatic mapping of designs written in Verilog into Field Programmable Gate Arrays (FPGAs).

The compilation and the synthesis reports for module "decorrelate_processor" are shown in Figures C.1.1, C.1.2, and C.1.3, whereas, the compilation reports for module "reconst_processor" are shown in Figures C.2.1, C.2.2, and C.2.3. The forward 9/7 compilation reports for module " decrrelation2_processor9_7" are shown in Figures C.3.1, C.3.2 and C.3.3, while that of module "decorelation_processor9_7" are shown in Figures C.4.1, C.4.2, and C.4.3. The 2-parallel architecture compilation reports for module "two_parallel_DWT" are shown in Figures C.5.1, C.5.2, and C.5.3.

The compilation report in Figure C.1.1 shows that the design uses 93 pins, a total of 438 logic cells, and a total of 434 registers, whereas, the compilation report shown in Figure C.1.2 indicates that the total power dissipation of the design is 500.46 mW. On the other hand, the Compilation Report-Timing Analyzer Summary shown in Figure C.1.3 lists four parameters. The first parameter t_{su} indicates the worse-case setup time required is 3.195 ns and it is from Ed3 to REd3. This parameter means that signal Ed3 must have a stable value at least 3.195 ns before each active edge of the clock. The second parameter t_{co} indicates the worse-case clock-to-output delay is 6.301 ns from register L_data_out[8] to pin L_data_out[8]. In other words, it indicates the time elapsed from an active edge of the clock at the clock pin until an output signal is produced at an output pin [65]. The third parameter in the Timing Analyzer Summary is t_h, which give the worse-case hold time, and it is 1.831 ns for the path from pin data_in0[0] to register Rt0_1[10]. Hence, the signal at pin data_in0[0] must maintain a stable value for at least 1.831 ns after each active edge of the clock. The last parameter in the list gives the maximum frequency, which is often called F_{max}, at which the synthesized circuit can operate is185.74 MHz. This is a useful indicator of performance. The maximum frequency is determined by the path with longest propagation delay, often called the critical path, between any two registers (flip-flops) in the circuit.

Figure C.1.3, shows that the maximum operating frequency F_{max} of the module is determined by the TLB operations where the path with longest delay occurs. This is expected since the overlapped architecture requires both read and write operations in the TLB to take place in the same clock cycle. However, since the intermediate architecture for 5/3 shown, in Fig 3.7.2, does not require such constraint on its TLB, therefore, the intermediate architecture would operate with higher frequency. Furthermore, the synthesis results shown in Figures C.1.3, C.2.3, C.3.3, and C.4.3, which show the maximum frequencies of the four implemented architectures, imply that the parallel forms of these architectures will also operate with the same frequencies. In fact, the 5/3 2-parallel architecture operating with frequency of 186.01 MHz, which is the parallel form of the single 5/3 pipelined architecture operating with frequency of 185.74 MHz, verifies that the 2-parallel architecture is 2 times faster than the single pipelined architecture. This result is also in agreement with the theoretical evaluation given in section 4.2.4.

To compare the implementation results of our architectures with other implementations in the literature, Table 7.2 is provided which summarizes the experimental results of several implemented architectures. This table shows that the 5/3 implementations in [3, 24] with 8-bit word length operate with frequencies of 110 MHz and 129.93 MHz, respectively, whereas, the proposed 5/3 forward and inverse with 16-bit word length operate with maximum frequencies of 185.74 MHz and 188.32 MHz, respectively. In addition, the implementation in [3] requires a large number of FPGA logic cells and registers. On the other hand, the 5/3 2-parallel architecture in [62], which is implemented on the same FPGA device, operates with a frequency of 145.54 MHz, whereas, the proposed 5/3 2-parallel architecture operates with frequency of 186.01 MHz.

The last 3 implementations in Table 7.2 are 9/7 architectures. Comparing the two 9/7 architectures, in term of speed, with the architectures proposed in [30, 40], shows

Table 7.2 Experimental results and comparisons

Architectures	Type	Logical cells	Regs	Max frequency	Power dissipation	Word length
Zewail [24]	5/3	473	149	112.93 MHz	N/A	8-bit
FIDF[62] 2-parallel	5/3	1316	466	145.54 MHz	N/A	16-bit
Gregory[3]	5/3	1741	2542	110 MHz	N/A	8-bit
PLSA[21]	9/7	416	192	152.39 MHz	N/A	16-bit
Xiong[40]	9/7	2992	N/A	50 MHz	393.62 mw	16-bit
Sandro[30]	9/7	1002	N/A	105 MHz	N/A	8-bit
Proposed forward	5/3	438	434	185.74 MHz	500.46 mw	16-bit
Proposed inverse	5/3	446	457	188.32 MHz	465.39 mw	16-bit
Proposed 2-parallel forward	5/3	872	697	186.01 MHz	580.98 mw	16-bit
Proposed first	9/7	2036	858	147.95 MHz	673.37 mw	16-bit
Proposed second	9/7	2529	1049	136.04 MHz	739.36 mw	16-bt

that the 9/7 architectures implemented in this work operate with higher frequencies. In addition, the implementation in [40] requires more logic cells and the operating frequency is very slow, 50 MHz. The implementation in [21], operates with a frequency of 152.39 MHz, which is slightly higher than the first proposed 9/7 implementation, which operate with a maximum frequency of 147.95 MHz. However, [21] introduced only a method, not architecture, for reducing the critical path delay to one multiplier and had implemented only one processor for 1-D DWT, while 2-D DWT architectures usually consist of two processors.

The final stage of the implementation is the timing simulation. To verify that both forward and inverse 5/3 architectures, the 5/3 2-parallel architecture, and both 9/7 architectures perform their intended logical functions accurately in the worst case timing of the target device; we have applied test input patterns and have simulated the implemented architectures' hardware modules. Figures 7.3.1, 7.3.2, 7.3.3, 7.3.4, and 7.3.5 show the simulation waveform results for the five implemented architectures. The forward 5/3 module "decorrelate_processor" is simulated by applying a 2-dimensional array of size 6x5 containing random numbers. This 6x5 image is scheduled according to the scan method shown in Figure 3.5.1, which requires 3 pixels to be fed into the circuit every clock cycle. The 3 pixels are indicated as data_in0, data_in1, and data_in2 in Figure 7.3.1. In cycle number 2 of Figure 7.3.1, the first 3 pixels 22, 143, and 65 of the first row are applied to the hardware module. In cycle 3, the first 3 pixels 62, 5, and 222 of the second row are applied to the hardware module. In cycle 7, the last 3 pixels 64, 121, and 34 of the last row are applied to complete the first run. The second run begins at cycle 8, where pixels 65, 192, and 115 are applied, and ends at cycle 13 with pixels 34, 143, and 32. The last run begins at cycle 14 and ends at cycle 19. Note that pixels of the last column are applied to the circuit one pixel at a time as shown in Figure 7.3.1, which is in accordance with the scan method.

The first two outputs of run1 simulation, which are shown under the labels L_data_out and H_data_out in Figure 7.3.1, appear at cycle 12 with output coefficients 21 and -103. These two coefficients belong to the first locations in subbands LL and LH, respectively. The second two output coefficients -3 and -207 belong to the first locations in subbands HL and HH, respectively. The hardware module alternates between generating output coefficients for subbands LL and LH and subbands HL and HH until the run ends. The first run ends by the positive transitions of clock cycle 18 with output coefficients 2 and 33. The positive transition of cycle 18 marks the ending of run1 and the beginning of run 2 with coefficients 131 and 29. These two output coefficients belong to the first location of the second column in each subbands LL and LH, respectively. The positive transition of clock cycle 24 marks the ending and the beginning of run 2 and the last run, respectively. The last run generates only output coefficients for subbands LL and LH. The simulation results in Figure 7.3.1 show that the hardware module for "decorretate_processor" precisely performs its function and according to Table B.6.

The signal between data_in2 and L_data_out in Figure 7.3.1 are control signals for RP and CP of Figures 3.8.7 and 3.8.3, respectively. The control signals sre0, sre1, and sre2 are control signals for RP's extension multiplexers, whereas, signals sce0, sce1, and sce2 are the control signals for CP's extension multiplexers. Signals incar and rst_TLBAR control the operation of the TLBAR (TLB address register), while signal ETLB is used for enabling TLB for read and write operations. The control signals Ed2, Ed3, and Ed4 control the operations of the registers and multiplexers that exist between the RP and the CP in Figure 3.6.1 and are set in Figure 7.3.1 according to Table 3.2.

In order to validate the inverse architecture, the output coefficients generated by module "decorrelate_processor" are fed into the inverse hardware module "reconst_processor" as shown in Figure 7.3.2. The coefficients are scheduled

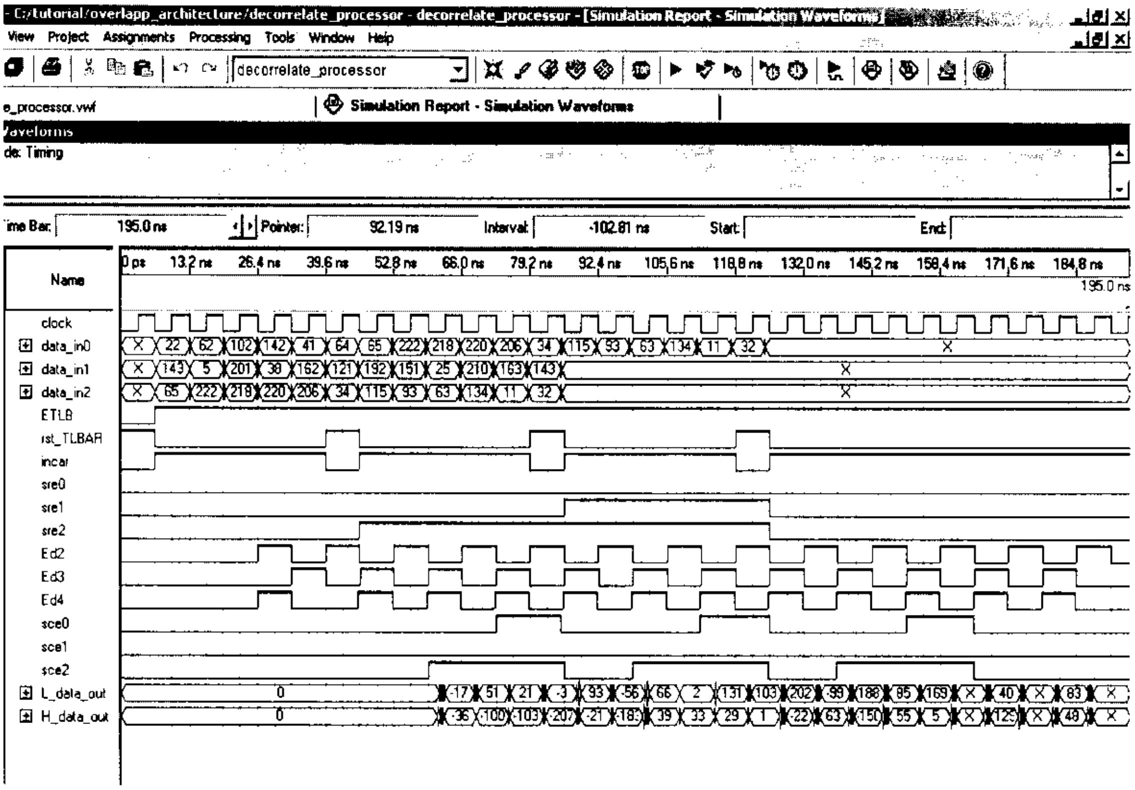


Fig. 7.3.1 Simulation Waveforms for forward 5/3 module "decorrelate_processor".

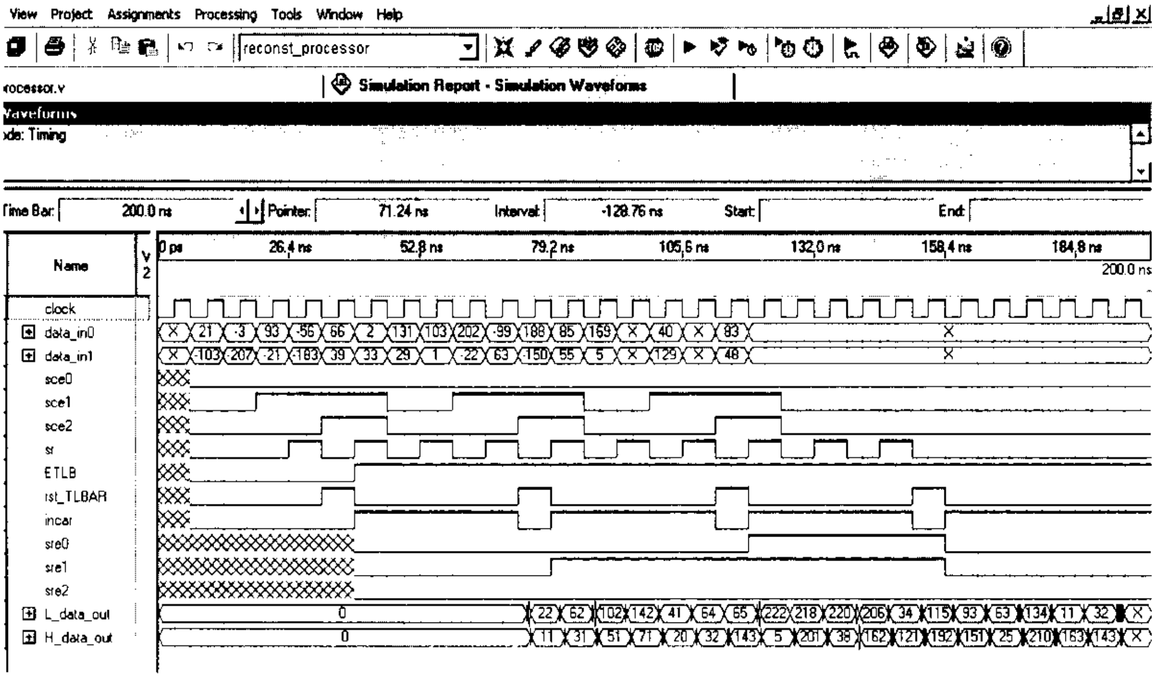
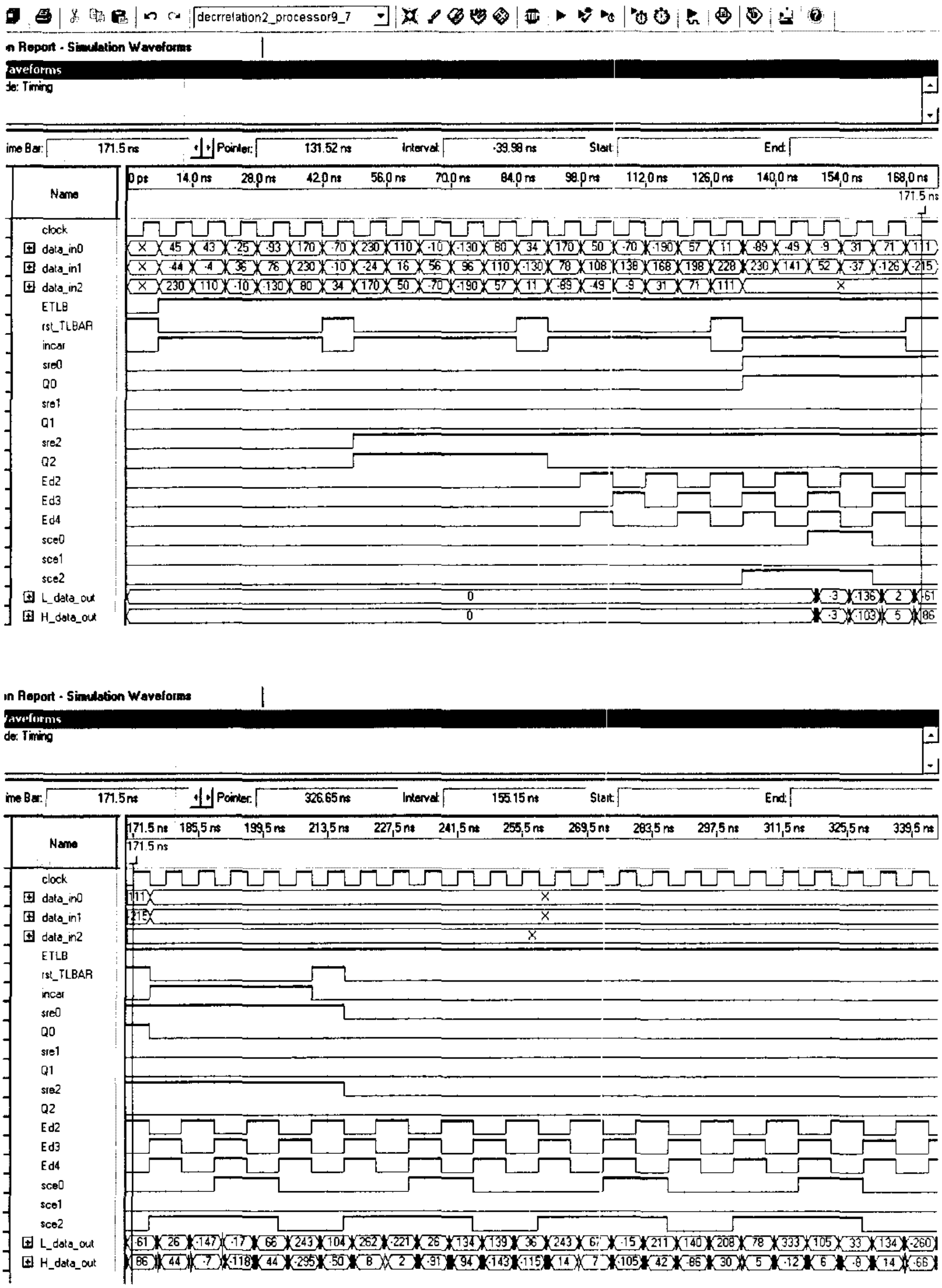


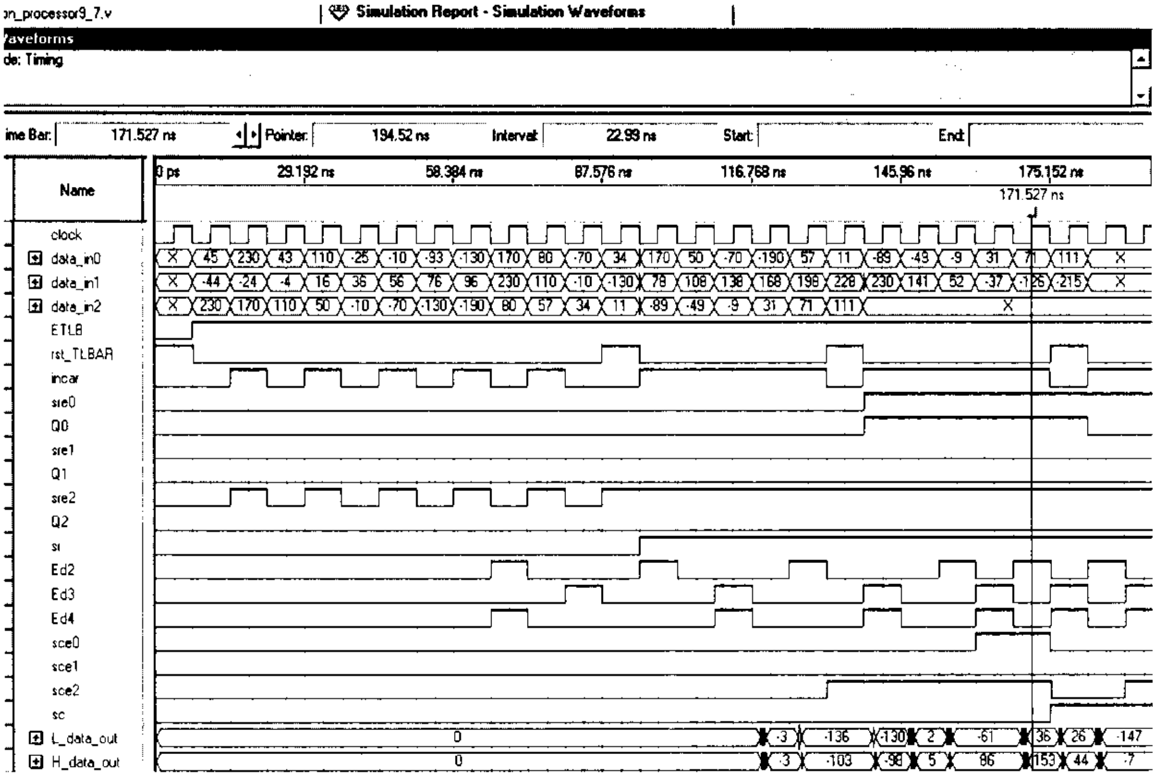
Figure 7.3.2 Simulation Waveforms for inverse 5/3 module "reconst_processor"



Time Bar: 171.5 ns Pointer: 131.52 ns Interval: -39.98 ns Start: End:

Time Bar: 171.5 ns Pointer: 326.65 ns Interval: 155.15 ns Start: End:

Fig. 7.3.3 Simulation Waveforms for first 9/7 module "decrrelation2_processor"

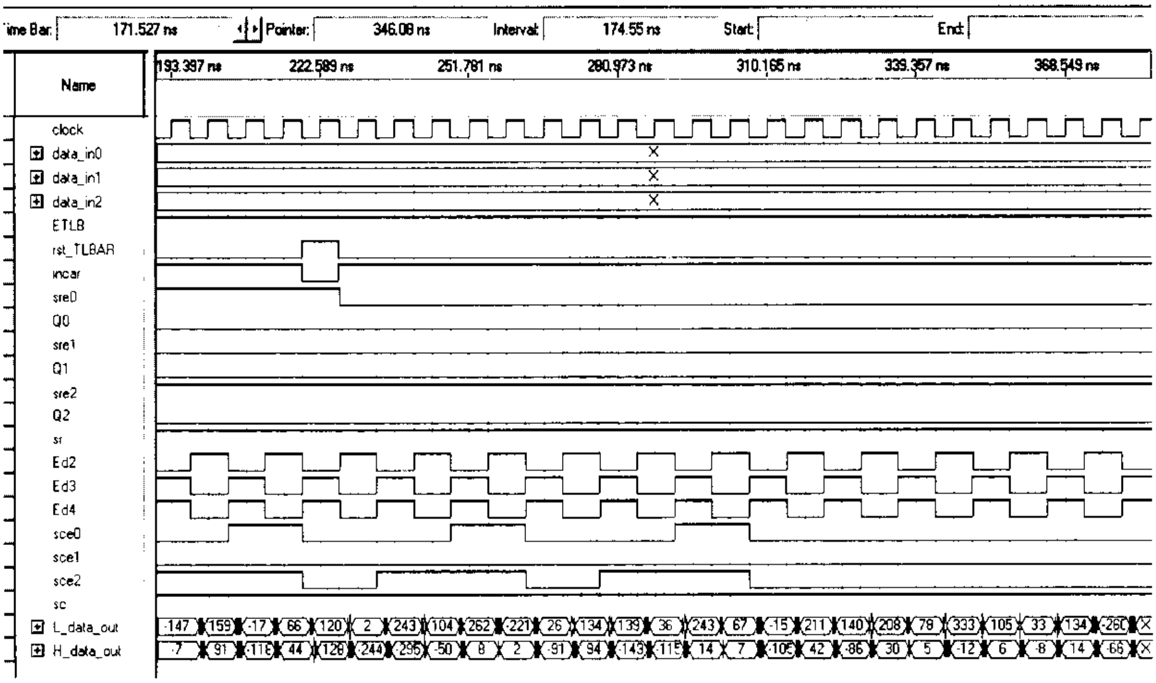


on_processor9_7.v | Simulation Report - Simulation Waveforms

Waveforms

de: Timing

ime Bar: 171.527 ns Pointer: 194.52 ns Interval: 22.99 ns Start: End:



ime Bar: 171.527 ns Pointer: 346.08 ns Interval: 174.55 ns Start: End:

Fig 7.3.4 Simulation Report – Simulation Waveforms for second 9/7 module “decorelation_processor”

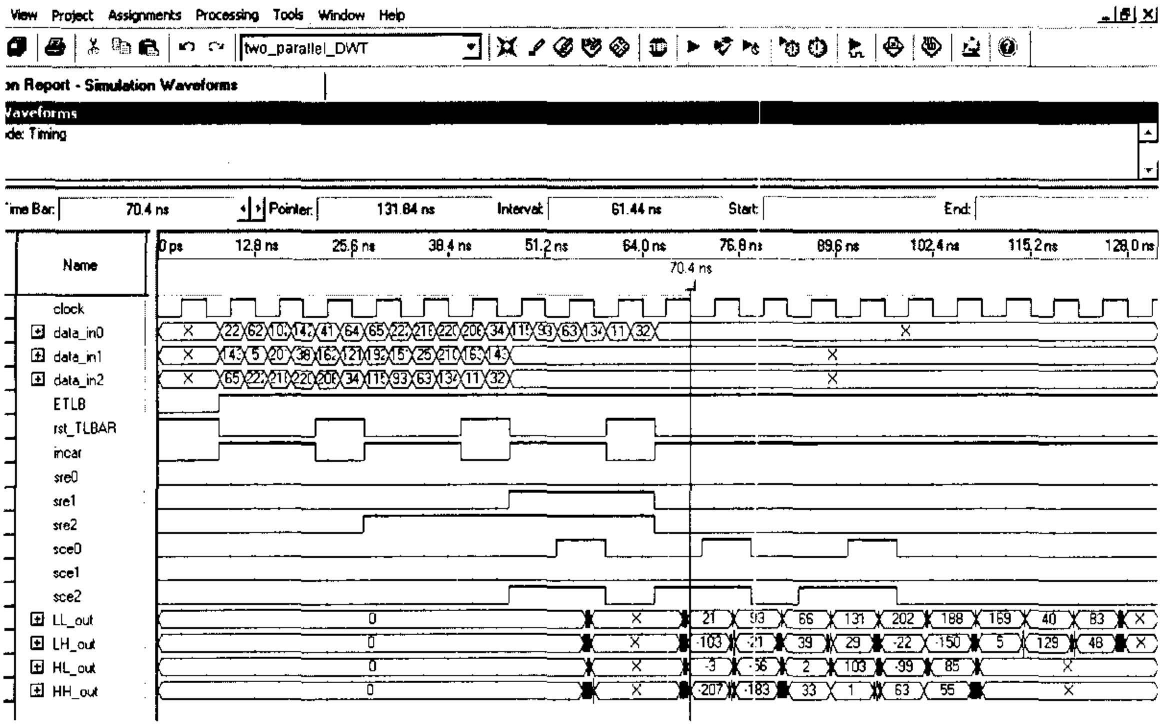


Fig 7.3.5 Simulation Report – Simulation Waveforms for the 5/5 2-parallel's module "decorelation_processor"

according to the scan method shown in Figure 6.3.1. The output of the simulation in Figure 7.3.2 indicates that the hardware module "reconst_processor" accurately reconstructs the original image pixels. In Figure 7.3.2, the first six outputs of run1 under L_data_out are valid output pixels, while the first output of H_data_out are not, according to the Table B.18. The first six outputs of L_data_out represent pixels of the first column in the 6x5 image. The second and the last runs each yield two columns to complete the 5 columns of the 6x5 image. The CP and the RP of the inverse external architecture implement the datapath architectures shown in Figures 6.5.4 and 6.5.6, respectively. The control signal sr of the external architecture is set in Figure 7.3.2 according to Table 6.1.

The hardware modules for both 9/7 forward pipelined overlapped architectures are tested by applying an image of size 6x8. This image is scanned into the first 9/7 hardware module "decrrelation2_processor" according to the scan method shown in Figure 3.5.1 and the results of the simulation are shown in Figure 7.3.3. This module does not yield any output coefficients in the first run, but starting from the second run it generates output patterns that are similar to the 5/3 forward overlapped architecture. It yields its first pair of output coefficients 26, and 44 at clock cycle 25, as shown in Figure 7.3.3. The positive transition of clock cycle 31 marks the ending of the second run, with output coefficients 104 and -50, and the beginning of the third run with output coefficients 262 and 8. This hardware module implements the RP and the CP datapath shown in Figures 3.8.8 (a) and 3.8.4 (a), respectively. The control signals sre0, Q0, sre1, Q1, sre2, and Q2, which are issued according to Table B.5, are control signal for RP's extension multiplexers. The control signals Ed2, Ed3, and Ed4 in Figure 7.3.3 are set according to Table B.2 (c).

On other hand, in the second 9/7 hardware module "decorelation_processor9_7", the image is scanned into the module according to the scan method shown in Figure 3.5.3. The simulation results are shown in Figure 7.3.4. The difference between this module and the first 9/7 module is that this module generates output coefficients starting from the first run and according to Table B.2. In Figure 7.3.4, its first pair of output coefficients 26 and 44 appears at cycle 25. The positive transition of clock cycle 35 marks the ending of the first run, with output coefficients 104 and -50, and the beginning of the second run with output coefficients 262 and 8. The simulation results shown in Figures 7.3.3 and 7.3.4 for both 9/7 module verify that both hardware modules perform their logical functions accurately in the worse case timing simulation. This hardware module implements the RP and CP datapath architectures shown in Figures 3.8.8 (b) and 3.8.4 (b), respectively. A table similar to Table B.2 (c), which contains control signal values, was derived from Table B.2 (b) for signals Ed2, Ed3, and Ed4 and then was used in Figure 7.3.4 for setting these signals.

The 5/3 2-parallel hardware module "two_parallel_DWT" is simulated by applying an image of size 6x5 which is identical to the one applied to the single pipelined architecture's module "decorrelate_processor". The image pixels are scanned into the hardware module according to the scan method shown in Figure 3.5.1. The simulation results are shown in Figure 7.3.5. In this figure, the first 4 output coefficients 21, -103, -3, and -207 appear at cycle 11. The positive transition of clock cycle 14 marks the ending of the first run with output coefficients 66, 39, 2, and 33 and the beginning of the second run with output coefficients 131, 29, 103, and 1. Cycle 17 marks the ending of the second run with output coefficients 188, 150, 85, and 55 and the beginning of the last run with output coefficients 169 and 5. In the last run, only CP2 generates output coefficients for subbands LL and LH. The 5/3 2-parallel's simulation results shown in Figure 7.3.5 are identical to the 5/3 single pipelined architecture's simulation results shown in Figure 7.3.1 and that verifies that the 2-parallel architecture performs its intended computations correctly as required. The 2-parallel hardware module implements the RP and the CP datapath architectures shown in Figure 4.2.2 and 3.8.1, respectively. In Figure 7.3.5, RP1 input latches are loaded with 3 pixels every time the clock makes a negative transition, whereas, RP2 input latches are loaded on the positive transition of the clock.

The six papers listed in Table 7.2, which had implemented their architectures on FPGA, had only provided synthesis results such as shown in Table 7.2 without any simulation waveforms results. Simulation results such as shown in Figs 7.3.1, 7.3.2, 7.3.3, 7.3.4, and 7.3.5 serve as prove the implemented architectures perform their functions correctly under the worse case timing of the target FPGA device.

7.4 Conclusions

In this chapter, 5 selective architectures, which are representative of the other architectures developed in this work, are implemented and synthesized on Altera FPGA. The compilation results of the implementation and comparisons are summarized in Table 7.2. the comparison results given in Table 7.1 and 7.2 including simulation results shown in Figs 7.3.1, 7.3.2, 7.3.3, 7.3.4, and 7.3.5 verify that the architectures implemented in this work not only are accurate and fast but are efficient in terms of power dissipation and hardware complexity. In addition, the synthesis results of the 2-parallel architecture shown in Fig C.5.3 confirm that the 2-parallel pipelined architecture is 2 times faster than the single pipelined architecture. Furthermore, the compilation results given in Figs C.3.1, C.3.2, and C.3.3 for the first 9/7 architecture and compilation results shown in Figs C.4.1, C.4.2, and C.4.3 for the second 9/7 architecture show that the first 9/7 architecture performs better than the second 9/7 architecture in terms of speed, power consumption, and hardware complexity.

CHAPTER 8

CONCLUSIONS AND RECOMMENDATIONS

8.1 Conclusions

In this research, two highly efficient and novel architectures for 2-D DWT are proposed that meet the high speed, low power, and memory requirements for real-time applications. The most noticeable accomplishment is the elimination of the internal memories, between row and column processors, which dominates the hardware cost. In the proposed pipelined architecture based on the nonoverlapped scan method, the power consumption due to the external frame memory access is reduced to minimum and it could be a very efficient alternative in applications where the power consumption is a serious issue.

In the development of the architectures, two cases were identified based on the scanning frequencies; case1, low scan frequency and case2, high scan frequency. In case1, the optimal performances of the pipelined architectures in terms of speed, efficiency, and hardware utilization are achieved by scanning 3 pixels in parallel each cycle. This requires slight modifications of the architectures developed in the first part that scan the external memory pixel-by-pixel. In case2, the optimal performances of the architectures are immediately obtained by pipelining the processors with no further modifications of the architectures developed in the first part.

Furthermore, the critical path delay of the proposed pipelined architectures can be reduced to four adders delays when multiplications operations in the 9/7 processors are implemented by adders only. The advantage of the approach adopted in the development of the two proposed architectures is that it can be used in developing architecture for any 2-D DWT algorithm and it is certain to yield very efficient architectures in terms of hardware complexity, speedup, and power consumption with manageable control complexity.

Based on the generalization of the overlapped scan method, the intermediate architecture is developed, which aims at reducing the power consumption of the overlapped areas without using the expensive line buffer to somewhat between the two extreme architectures, the overlapped and nonoverlapped. Compared with the power consumption of scanning the external memory for the architecture based on the first scan method, the intermediate architecture decreases the power by 22% with no lost in speed. While the intermediate architecture with the second dataflow decreases the power consumption of scanning the external memory by 48%. However, the second dataflow increases the total execution time by 16.7% over the architecture based on the first scan method and the intermediate architecture using the first dataflow. In addition, since the reduction in the power consumption is achieved without using a line buffer, the intermediate architecture occupies less silicon area. Therefore, intermediate architecture could be a very efficient alternative for high-speed, low cost, and low power applications such as mobile video phone.

To further improve performance in terms of speed and throughput to best meet real-time applications of 2-D DWT with demanding requirements, parallel architectures were developed. The single pipelined overlapped architecture is extended to 2-parallel, 3-parallel, and 4-parallel architectures to achieve speedup factors of 2, 3, and 4, respectively, according to the evaluation given in section 4.2.4. The scheme adopted in the development of the 4-parallel architecture optimizes the performance, in term of number of clock cycles requires for j levels of decomposition, as compared with the alternative scheme which increases the execution time by $M/2^{j-1}$ cycles for each level of decomposition, when case 2 occurs. Similarly, the single pipeline intermediate architecture is extended to 2-parallel and 3-parallel architectures. According to the evaluation given in section 4.3.5, the 2-parallel and 3-parallel intermediate architectures achieve speedup factors of 2 and 3, respectively. The intermediate parallel architecture reduces the power consumption of the external memory by a factor of 7/9 as compared with the overlapped parallel architecture, Eq(4.57).

The advantage of the proposed parallel architectures developed in this research is that the total temporary line buffer (TLB) does not increase from that of the proposed single pipelined architectures, when degree of parallelism is increased. In addition, the comparison results show that single and parallel architectures developed in this research compared with most recent architectures in the literature require only a total TLB of size *N* in the 5/3 processor datapath and *3N* in the 9/7, while other architectures listed in Table 10 require more TLBs, which are very expensive memory components. In addition, the control architecture that detects occurrence of the last run and the 6 cases of the intermediate architectures is also designed. Furthermore, to reduce control designs effort, several tables giving the control signal values for several control signals are provided.

This research has also addressed in details one of the important issues that have been overlooked so far, that is, the 2-D DWT memory architectures and management and has proposed two novel VLSI memory architectures, the LL-RAM and subband memory, which are based on the first scan method. The LL-RAM and subband memory were designed such that DWT unit performs effectively both read and write operation in the LL-RAM and write only into suband memory while compression unit reads subband memory. How the two memory architectures can be modified for higher scan method is also illustrated. The banking technique is used to further improve and form more efficient memory architectures in terms of speed and power consumption. The bank-based architecture can be thought formed by dividing the module-based RAM architecture, which can be considered as one big bank, into several smaller independent banks. Inside the smaller banks reads and writes are performed as in the big bank but faster and more efficiently. The advantage of the two proposed memory architectures is that they can be easily incorporated into single or parallel 2-D DWT processor architectures.

To show that the architectures developed in this research are simple to control, the control algorithms for 4-parallel architecture including the LL-RAM and the subband memory were developed. To ease the control development, the overall system control is divided into several smaller units. Then, the algorithmic state machine (ASM) for each unit is developed. The control algorithms developed here can be used to derive the hardware of the control.

Furthermore, based on data dependency graphs (DDGs) and scan methods specifically developed for inverse 5/3 and 9/7, the external architectures for single and parallel 5/3, 9/7, and combined 5/3 and 9/7 were developed. First, a high-speed single pipelined inverse architecture including its column-processor (RP) and row-processor (CP) were developed. Then, the single pipelined architecture is extended to 2-parallel and 4-parallel to achieve speedup factors of 2 and 4, respectively. The advantage of the single pipelined architecture developed here is that it only requires a total temporary line buffer (TLBs) of sizes 2N and 4N for 5/3 and 9/7, respectively, and the TLB requirement does not increase when it extended to parallel architecture. The combined architecture is very useful and efficient in situations where a decoder in one site is required to perform either lossless 5/3 or lossy 9/7 image reconstruction. In addition, the advantage of the combined architecture is that a considerable saving in silicon area can be achieved. The proposed architectures besides precisely implementing the two algorithms, their control complexity is simple. Specifically the external architecture's control signals of the single pipelined inverse architecture shown in Figure 6.4.1 were reduced to only one control signal. The interleave technique used by CP for combing subbands not only speeds up the computations by allowing RP to work in parallel with CP as early as possible, but reduces internal memory requirement between CP and RP to a few registers.

The processor datapath architectures were first developed assuming the external memory is scanned either row-by-row or column-by-column. However, since the external architectures developed in this work scan the external memory differently, the processors datapath for single and parallel architectures are modified in order to fit into the external architectures' processors.

The symmetric extension algorithm is incorporated in the data dependency graphs (DDGs) to handle the boundary problem and then implemented by all architectures developed in this work. Symmetric extension is a necessary treatment to prevent distortion from appearing at the image boundaries.

The scan method adopted, for development of architectures, not only reduces the internal memory between RPs and CPs to a few registers, but also reduces the internal memory or number of TLBs in the RP to minimum. In addition, it allows CPs to work in parallel with RPs earlier during the computation, which leads in reducing the latency to a few cycles.

The approach or the strategy adopted in the development of the proposed single and parallel architectures can be used in architecture development for any 2-D DWT algorithm and it is certain to yield very efficient architectures in terms of hardware complexity, speedup, and power consumption with manageable control complexity.

The simulation results of the five architectures implemented and synthesized on Altera FPGA verify that the architectures developed in this work not only are accurate and fast but are efficient in terms of power dissipation and hardware complexity. In addition, the synthesis results of the 2-parallel architecture shown in Figure C.5.3 confirm that the 2-parallel pipelined architecture is 2 times faster than the single pipelined architecture. Furthermore, the compilation results given in Figures C.3.1, C.3.2, and C.3.3 for the first 9/7 architecture and compilation results shown in Figures C.4.1, C.4.2, and C.4.3 for the second 9/7 architecture show that the first 9/7 architecture performs better than the second 9/7 architecture in terms of speed, power consumption, and hardware complexity.

The Verilog version used in Altera FPGA Quartus II does not have the capability of supporting simulation using real images. This limitation has forced my to use images of sizes 6x5 and 6x8 containing random numbers in the final simulation. Another limitation is that the architectures developed in this work are designed to process the whole image as one tile. JPEG2000 allows (optionally) an image to be divided into a number of smaller non-overlapping rectangular blocks known as "tiles" and then each tile is processed independently by DWT unit. This mechanism is a useful to use for computing 2-D DWT of a large image independent of its size with the use of the smaller intermediate memory (LL-RAM) to store "LL" coefficients for next level of decomposition. Thus a control algorithm is needed to divide a large image into tiles and then passes each tile to the DWT unit for processing.

8.2 Recommendations

The possible future work would be to extend the approach and the techniques acquired from this research to develop architectures for any 2-D DWT algorithms including development of VLSI architectures for signal and image processing algorithms. This work also could be extended to develop architectures for 3-dimensional images where computational requirements are very intensive with complex and large memory requirements. Furthermore, the concept and techniques developed in this work also can aid in the development of VLSI architectures for Turbo decoder. Turbo code is one of the most attractive error correction codes and it is an essential component in digital communication and data storage systems

Another possibility would be to extend this work to develop architecture for compression part of the system, which uses EBCOT (Embedded Block Code with Optimized Truncation), to independently code each subband coefficients. EBCOT contains Tier 1 and Tier 2. Tier 1 is implemented in hardware, whereas, Tier 2 is implemented in software. The insight gained from this work would aid the designer to develop compression architecture that can be integrated into the 2-D DWT architecture.

Moreover, this research includes many in-depth and optimized designs and therefore, can be available reference for graduate students and researchers pursuing in-depth study in this field.

REFERENCES

[1] Chao-Tsung, Po-Chih, and Liang-Gee, "Generic RAM-Based architecture for 2-D Discrete wavelet transform with line-based method", IEEE trans on circuits an Systems for video technology, vol. 15, No.7, July 2005, PP. 910 – 920.

[2] C.-T. Huang, P.-C, Tseng, and L.-G Chen, "Flipping structure: an efficient VLSI Architecture for lifting-based discrete wavelet transform," IEEE Trans. Signal Processing, vol. 52, No. 4, April 2004, PP. 1080 - 1089.

[3] Gregory Dillin, Benoit Georis, Jean-Didier Legant, and Olivier Cantineau,"Combined Line-based Architecture for the 5-3 and 9-7 Wavelet Transfprm of JPEG2000,"IEEE Trans. on circuits and Systems, vol. 13, No. 9, Sep. 2003, PP. 944 – 950..vol 54, No. 5, May 2006, PP. 1910 – 1916.

[4] Daubechies and Sweldens, "Factoring wavelet transform into lifting schemes," J. Fourier Analysis and Application, vol. 4, No. 3, 1998, PP. 245 – 267..

[5] Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions," in proc. SPIE, vol. 2569, 1995, PP. 68 - 79.

[6] Calderbank, I. Daubechies, Swelden, and Yeo, "Wavelet transforms that map integers," J. Applied and Computational Harmonic. Analysis, vol. 5, No. 3, Sept. 1998, PP.332 - 369.

[7] ISO/IEC, ISO/IE15444-1, information technology-JPEG2000 image coding system, 2000. Website : http://www.jpeg.org/CDs15444.html.

[8] David S. and Michael W. "JPEG 2000 image compression fundamentals, standards and practice," Kluwer Academic pulishers, 2002.

[9] Mu-Yu chiu, Kun-Bin Lee and Chein-Wei Jen,"Optimal data transfer and Buffering Schemes for JPEG2000 encoder," in proceeding IEEE workshop signal proc. Sytst 2003, PP. 177 - 182.

[10] K. K. Parhi and T. Nishitani, "VLSI architectures for discrete wavelet transforms," IEEE Trans. Very Large Scale integration (VLSI) System, June 1993, PP. 191 - 202.

[11] C. Chryatis and A. Orlega, "Line-based, reduced memory, wavelet image compression," IEEE Trans. Image Processing, vol. 9, No. 3, March 2000, PP. 378 – 389.

[12] M. Week and M. Bayoumi, "Discrete wavelet transform: Architectures, design and performance issues," J. VLSI signal processing & systems, vol. 35, No. 2, 2003, PP. 155 – 178.

[13] Kishore A., Chaitati Ch., and Tinku A. "A VLSI architecture for lifting-based forward and inverse wavelet transform," IEEE Trans. on signal processing, vol.50, No. 4, April 2002.

[14] W. Jiang and A. Ortega, "Lifting Factorization-based Discrete Wavelet Transform Architecture Desgin," IEEE Trans. on Circuits & Sys. For Video Technology, vol. 11, N. 5, May 2001, 651 – 657.

[15] Anil K. Jain, "Fundamentals of digital image processing," Prentice Hall 1989.

[16] K. K. Parhi, VLSI Digital Signal Processing System: Design and Implementation. New York: Wiley, 1999.

[17] F. Marino, "Efficient high-speed/ low-power pipelined architecture for the direct 2-D

discrete wavelet transform," IEEE Trans. Circuits Sys.II, Analog Digital signal processing vol. 47, No. 12, Dec 2000, PP. 1476 - 1491.

[18] Sanjit K. Mitra, "Digital signal processing, a computer–based approach," McGraw_Hill, 2001.

[19] Jaideva C. Goswami and Andrew K. Chan,"Fundamentals of wavelets, theory algorithm, and applications," New York, Wiley, 1999.

[20] Agostino Abbate, Casimer M. and Pankaj K. Das,"Wavelets and subbands, Fundamentals and applications," Birkauser, 2001.

[21] Cheng_Yi, Jim_Wen, and Jian Liu, "A note on "Flipping structure: an efficient VLSI architecture for lifting-based discrete wavelet transform"," IEEE trans. on signal processing, vol.54, No. 5, May 2006, PP. 1910 – 1916.

[22] Cheng-Yi Xiong, Jim-Wen Tian and Jian Liu,"Efficient parallel architecture for lifting-based two-dimensional discrete wavelet transform," IEEE Int. Workshop VLSI Design & Video Tech. China, May 2005, PP. 75 - 78.

[23] Qing-ming Yi and Sheng-Li Xie,"Arithmetic shift method suitable for VLSI implementati-on to CDF 9/7 discrete wavelet transform based on lifting scheme," Proceedings of the Fourth Int. Conf. on Machine Learning and Cybernetics, Guangzhou, August 2005, PP. 5241-5244.

[24] R. Zewail, P. Marshall, S. Kozicki, N. Ying, D. Elliott, and N. Durdle," A reconfigurable fully Scalable integer wavelet transform unit for JPEG200," CCECE/CCGEI Saskatoon, IEEE, May 2005, PP. 798 - 801.

[25] Zhi-Rong Gao and Cheng-Yi Xiong," An efficient Line-based architecture for 2-D discrete wavelet transform," proceeding of IEEE international conference on communications, circuits and systems, 2005, PP. 1322 - 1325.

[26] Chengyi Xiong, Jinwen Tian and Jian Lui,"A fast VLSI architecture for two-dimensional discrete wavelet transform based on lifting scheme," proceeding of IEEE 7th international conference on solid-state and integrated circuits technology, 2004, PP. 1661-1664.

[27] Srikar Movva and Srinivasan S.,"A novel architecture for lifting-based discrete wavelet transforms for JPEG2000 standard suitable for VLSI implementation," Proceedings of the 6th International Conf. on VLSI Design, 2003 IEEE, PP. 202 – 207

[28] K-C. B Tan and T. Arslan,"Shift-accumulator ALU centric JPEG2000 5/3 lifting based discrete wavelet transform architecture," proceeding of 2003 IEEE, PP. v161 – v164.

[29] Xuguang Lan, Nanning Zheng, & Yuehu Liu, "Low-Power and High-Speed VLSI Architecture For lifting-Based Forward and Inverse Wavelet Transform" IEEE transaction on consumer electronics. Vol. 51, issue 2, 2005, PP 379 385..

[30] Sandro V. Silva & Sergio Bampi, "Area and Throughput Trade_offs in the Design of Pipelined Discrete Wavelet Transform architectures," proceedings of the design automation and Test in Europe conference and Exhibtion. 2005 IEEE, PP 32 - 37.

[31] W. Swelden, "The lifting scheme: A custom-design construction of biorthogonal wavelets, " Applied and computational Harmonic Analysis, vol. 3, No. 15, 1996, PP.186 – 200.

[32] Zhong Guangjum, Cheng Lizhi & Chen Huowang, "A simple 9/7-tap wavelet filter based on lifting scheme," proceeding of IEEE international conference on image processing, vol. 2, 2001,

PP. 249 – 252.

[33] Chao-Tsung Huang, Po-Chih Tseng, and Liang-Gee Chen, "Analysis and VLSI Architecture for 1-D and 2-D Discrete Wavelet Transform," IEEE Trans. on signal processing, vol.53, No. 4, April 2005, PP.1575 – 1586..

[34] Jen-Shiun Chiang, Chih-Hsien Hsia, Hsin-Jung Chen, and Te-Jung Lo, "VLSI Architecture of Low Memory and High Speed 2-D lifting-Based Discrete Wavelet Transform for JPEG 2000 Applications," IEEE international symposium on circuits and systems, ISCAS 2005, Vol. 5, PP. 4554 – 4557.

[35] S. Barua, J. E. Carletta, K. A. Kotteri, A. E. Bell, "An efficient architecture for lifting-base two-dimensional discrete wavelet transforms", Integration, the VLSI journal, 2005 Elsevier, PP. 341 - 352.

[36] G. Dimitroulakos, M. D. Galanis, A. Milidonis, and C. E. Goutis, "A high-throughput and Memory efficient 2-D discrete wavelet transform hardware architecture for JPEG2000 Standard," IEEE international symposium on circuits and systems ISCAS 2005, Vol. 1, PP.472 – 475..

[37] Chengjun Zhang, Chunyan Wang, M. Omair Ahmad," A VLSI architecture for a High-speed computation of the 1-D discrete wavelet transform", IEEE international symposium on circuits and systems, ISCAS 2005, Vol. 2, PP. 1461 – 1464.

[38] K. A. Kotteri, S. Barua, A. E. Bell, and J. E. Carletta,"A comparison of hardware implementations of the Biorthogonal 9/7 DWT : convolution versus lifting", IEEE Trans. on Circuits & System, vol. 52, No.5, May 2005, PP. 256 - 260.

[39] Yuan-Long Jean, Kai-Jearg, Kai-Jyun Liang, Jiun-Hau Tu, Jain-Zhou Huang, and Pingshou Cheng, "An embeded wavelet image coding algorithm and its hardware implementation Based on Zero-block and Array (EZBA)," 48th Midwest symposium on circuits and systems, IEEE 2005, Vol. 2, PP. 1414 - 1417.

[40] C-Y. Xiong, J-W. Tian, J. Liu, "Efficient high-speed/low-power line- based architecture for 2-dimensional discrete wavelet transforms using lifting scheme," IEEE Trans. On Circuits & sys. for Video Tech.Vol.16, No. 2, February 2006, PP. 309-316.

[41] Michael Unser, & Thierry Blu, "Wavelet Theory Demystified", IEEE Trans. on Signal Processing, vol. 51, No. 2, February 2003, PP. 470 - 483.

[42] Chao-Tsung Huang, Po-Chih Tseng & Liang-Gee Chen, "VLSI architecture for forward discrete wavelet transform based on B-spline factorization", Journal of VLSI signal processing, 2005 Springer Science, PP. 343 - 353.

[43] B-F. Wu, C-F. Lin, "A high-Performance and Memory-Efficient Pipeline Architecture for the 5/3 and 9/7 Discrete Wavelet Transform of JPEG2000 Codec," IEEE Trans. on Circuits & Sys. for Video Technology, Vol. 15, No. 12, December 2005, PP. 1615 – 1628.

[44] Iain E. G. Richardson, "H. 264 and MPEG-4 video compression, video coding for next-generation multimedia," Wiley 2003.

[45] Maurizio Martina and Guido Masera, "Low-complexity, Efficient 9/7 Wavelet Filters Implementation," proceeding of 2005 IEEE.

[46] Gab Cheon Jung, Seong Mo Park, and Jung Hyoun Kim, "An Efficient VLSI Architecture for JPEG2000 Encoder," proceeding of 2005 IEEE, PP. 1203 – 1206.

[47] David B. H. Tay, "A class of lifting based integer wavelet transform," proceeding of IEEE international conference on image processing, vol. 1, 2001, PP. 602 - 605.

[48] Zhi-Rong Gao & Cheng-Yi Xiong, "Combining Parallel Lifting and Retiming Architecture for Discrete Wavelet Transform," IEEE Int. workshop VLSI design and video Technology, Suzhou, China, May 2005, PP. 175 - 178.

[49] Michael D. Adams, & Faouzi Kossentini, "Reversible Integer-to-Integer Wavelet Transforms for Image Compression: Performance Evaluation and Analysis," IEEE Transactions on Image Processing, vol. 9, No. 6, June 2000, PP. 1010 - 1024.

[50] P.-C. Tseng, C.-T. Huang, and L.-G Chen,"VLSI implementation of shape adaptive discrete wavelet transform," in Proc. SPIE Int. conf. Visual Communications and Image Processing, 2002, PP. 655 – 666.

[51] N. D. Zervas,G. P. Anagnostopoulos, V. Spiliotopoulos, Y. Andreopoulos, and C.E. Goutis, "Evalution of design alternatives for the 2-D discrete wavelet transform," IEEE Trans. Circuits Syst. Video Technology, vol. 11, No. 12, December 2001, PP. 1246 - 1262.

[52] H. Yamauchi et al., "Image processor capable of block-noise-free JPEG2000 compression with 30 frames / s for digital camera applications," in proceeding IEEE Int. Solid-State Circuits Conf., vol. 1, PP. 46 – 477, 2003.

[53] Jorg Ritter and Paul Molitor, " A pipelined architecture for partitioned DWT based lossy image compression using FPGA's," Monterey, CA, USA, ACM 2001, PP. 201 - 206.

[54] L. Liu, X. Wang, H. Meng, L. Zhang, Z. Wang, and H. Chen, " A VLSI architecture of spatial combinative lifting algorithm based 2-D DWT/IDWT," in proc. 2002 Asia-pacific conf. circuits and systems, vol. 2, 2002, PP. 299 – 304.

[55] B. F. Wu and C. F. lin, "A rescheduling and fast pipeline VLSI architecture for lifting-based discrete wavelet transforms," in proceeding IEEE ISCAS, May 2003, PP. 732 - 735

[56] H. Meng and Z. Wang, "Fast spatial combinative lifting algorithm of wavelet Transform using the 9/7 filter for image block compression," Electron. Lett., Vol. 36, No. 21, Oct. 2002, PP. 1766 – 1767.

[57] MPEG-4, ISO/IEC JTC1/SC29/WG11,FCD 14496, "coding of moving pictures and audio," May 1998.

[58] Kai Hwang, "Advaced Computer Architecture: Parallelism, Scalability, Programmabilty," McGraw-Hill 1993.

[59] Hongyu Liao, Mrinal Kr., and Btuce F. "Efficient architectures for 1-D and 2-D lifting-based wavelet transform," IEEE Trans. on signal processing, vol. 52, No. 5, May 2004.

[60] W. Chao, W. Zhilin, C. Peng, and L. Jie, "An efficient VLSI Architecture for lifting-based discrete wavelet transform," Mulltimedia and Epo, 2007 IEEE International conference, PP. 1575-1578.

[61] R. Jain and P. R. Panda,"An efficient pipelined VLSI architecture for Lifting-based 2D-discrete wavelet transform," ISCAS, 2007 IEEE, PP. 1377-1380.

[62] B-F. Li and Y. Dou, "FIDP A novel architecture for lifting-based 2D DWT in JPEG2000," MMM (2), lecture note in computer science, vol. 4352, Springer, 2007, PP. 373-382.

[63] Peng Cao, Xin Guo, Chao Wang, and Jie Li, "Efficient architecture for two-dimensional discrete Wavelet transform based lifting scheme," 7th International Conference on ASIC, ASICON'07, 2007 IEEE, PP. 225 – 228.

[64] Chengyi Xiong, Jinwen Tian, and Jian Liu, "Efficient architecture fot two-dimensional discrete Wavelet transform using lifting scheme," IEEE transactions on image processing, Vol. 16, No. 3, March 2007, PP. 607 – 614.

[65] Stephen Brown and Zvonko Vranesic, "Fundamentals of digital logic with verilog design," Second edition, Mc Graw-Hill, higher eductation, 2008.

[66] Chih-Hsien Hsia and Jen-shium Chiang, "New memory-efficient hardware architecture of 2-D dual-mode lifting-based discrete wavelet transform for JPEG2000," 11th IEEE Singapore International Conference on Communication Systems, 2008 IEEE, PP. 766 – 772.

[67] Jie Guo, Ke-yan Wang, Cheng-ke Wu, and Yun-song Li, "Efficient FPGA implementation of Modified DWT for JPEG2000," 9th International Conference on Solid and Integrated-circuit Technology, 2008 IEEE, PP. 2200 – 2303.

[68] Wei-Ming Li, Chih-Hsien Hsia, and Jen-Shiun Chiang, "Memory-efficient architecture of 2-D dual-Mode discrete wavelet transform using lifting scheme for motion-JPEG2000," IEEE International Symposium on circuits and systems, 2009, PP. 750 – 753.

[69] Pingping Yu, Suying Yao, and Jiangtao Xu, "An efficient architecture for 2-D lifting-based discrete wavelet transform," 4th IEEE Conference on Industrial Electronics and Applications, 2009, PP. 3667 – 3670.

[70] Xiaodong Xu and Yiqi Zhou, "Efficient FPGA implementation of 2-D DWT for 9/7 float wavelet filter," International Conference on Information Engineering and Computer Science, 2009 IEEE, PP. 1 – 4

[71] Chung-Fu Lin, Pei-kung, and Bing-Fei Wu, "An efficient pipeline architecture and memory bit-width Analysis for discrete wavelet transform of the 9/7 filter for JPEG2000," J Sign Process Syst, 2009 Springer, PP. 245 – 253.

APPENDIX A

SOFTWARE SIMULATION PROGRAM DEVELOPMENT

A.1 Introduction

It will be of a great benefit to start this research by developing a software simulation program that computes both forward and inverse 2-D DWT using lifting-based 5/3 algorithms. The forward operations decorrelate the original image to be amenable to compression, whereas the inverse operations reconstruct the original image from the decorrelated image. Developing a simulation program will give the hardware architecture designer available opportunity to learn in details the behavior of the algorithm and acquire a firm understanding, which in turn will enable him to develop more accurate architecture.

A.2 Forward and inverse lifting-based 5/3 algorithms and software development

Lifting-based forward and inverse 5/3 wavelet transform algorithms are defined by the JPEG2000 image compression standard as follow s [7, 27, 29].

5/3 forward algorithm

$$step1: Y(2j+1) = X(2j+1) - \left\lfloor \frac{X(2j) + X(2j+2)}{2} \right\rfloor$$

$$step2: Y(2j) = X(2j) + \left\lfloor \frac{Y(2j-1) + Y(2j+1) + 2}{4} \right\rfloor$$

5/3 inverse algorithm

$$step1: X(2j) = Y(2n) - \left\lfloor \frac{Y(2j-1) + Y(2j+1) + 2}{4} \right\rfloor$$

$$step2: X(2j+1) = Y(2j+1) + \left\lfloor \frac{X(2j) + X(2j+2)}{2} \right\rfloor$$

Based on the above two algorithms, the data dependency graphs (DDGs) for forward and inverse, shown in Figures 3.3.1 and 6.2.1, respectively, are derived. The symmetric extension algorithm recommended by JPEG2000 is also incorporated into the DDGs to handle boundary problems. Based on forward and inverse algorithms and the DDGs, the software simulation program listed below is developed.

****

*   This software is developed by Ibrahim Saeed Koko at   *

*   Universiti Teknologi PETRONAS (UTP)   *

****

FORWARD PROGRAM

```
% program fdwt
X1 = imread('cameraman.tif');  %read image and store it in
                               %     array X
X1 = rgb2gray(X1);             % Separates image from colors
X  = double(X1);               % convert pixels from grayscale
                               % numbers to signed numbers
[m,n] = size(X);

YH = horizontalf(X);            % first level decomposition
YL = horizontalfl(X,YH);
YHH1 = verticalf(YH);
YHL1 = verticalFL(YH,YHH1);
YLH1 = verticalf(YL);
YLL1 = verticalFL(YL,YLH1);


YH = []; YL =[];                 %free YH and YL
[m,n] = size(YLL1);

YH = horizontalf(YLL1);
YL = horizontalfl(YLL1,YH);      % second level decomposition
YHH2 = verticalf(YH);
YHL2 = verticalFL(YH,YHH2);
YLH2 = verticalf(YL);
YLL2 = verticalFL(YL,YLH2);

YH = []; YL = [];  YLL1 = [];
[m,n] = size(YLL2);
YH = horizontalf(YLL2);          % third level decomposition
YL = horizontalfl(YLL2,YH);
YHH3 = verticalf(YH);
YHL3 = verticalFL(YH,YHH3);
```

```
YLH3 = verticalf(YL);
YLL3 = verticalFL(YL,YLH3);

YLL2 = [];   YH = [];    YL = [];
```

```
function YH= horizontalf(z0)        %horizontal highpass decomposition
[m,n]= size(z0);
k = fix(n/2);
for i = 1:m
    for j = 1:k
        if (j < k) | (k ~= n/2)
            YH(i,j) = z0(i,2*j) - fix((z0(i,2*j-1)+z0(i,2*j+1))/2);
        else
            YH(i,j) = z0(i,2*j) - z0(i,2*j-1);
        end
    end
end
```

```
function YL= horizontalfl(z0,YH)     %horizontal lowpass decomposition
[m,n]= size(z0);
k = fix(n/2);
if k ~= n/2
    k = k + 1;
end

for i = 1:m
    for j = 1:k
        if j == 1
            YL(i,j) = z0(i,2*j-1) + fix(YH(i,j)/2);
        else if (fix(n/2) == n/2) | (j < k)
            YL(i,j) = z0(i,2*j-1) + fix((YH(i,j-1)+YH(i,j)+2)/4);
            else
                YL(i,j) = z0(i,2*j-1) + fix(YH(i,j-1)/2);
            end
        end
    end
end
```

```
function ZL= verticalFL(z1,ZH)        %vertical lowpass decomposition
[m,n] = size(z1);
k = fix(m/2);
if k ~= m/2
   k = k + 1;
end
for i = 1:n
    for j = 1:k
        if j == 1
            ZL(j,i) = z1(2*j-1,i)+fix(ZH(j,i)/2);
        else if(fix(m/2) == m/2) | (j < k)
            ZL(j,i) = z1(2*j-1,i)+fix((ZH(j-1,i)+ZH(j,i)+2)/4);
            else ZL(j,i) = z1(2*j-1,i) + fix(ZH(j-1,i)/2);
            end
        end
    end
end
```

```
function ZH = verticalf(z1)   %vertical highpass decomposition
[m,n] = size(z1);
k = fix(m/2);
for i = 1:n
    for j = 1:k
        if (j < k) | (k ~= m/2)
            ZH(j,i) = z1(2*j,i)-fix((z1(2*j-1,i)+z1(2*j+1,i))/2);
        else
            ZH(j,i) = z1(2*j,i)-z1(2*j-1,i);
        end
    end
end
```

```
% function f2dwt

fdwt;                           % call main program.

Y3 = [YHH3 YLH3;YHL3 YLL3];     % combine subbands to obtain
Y2 = [YHH2 YLH2; YHL2 Y3];      % decorrelated image
```

```
Y1 = [YHH1 YLH1; YHL1 Y2];             %decomposed image

Y = mat2gray(Y1);             %covert a data to a grayscale image
figure, imshow(Y);             % Display decorrelated image
title('Decorrelated image')

[m,n] = size(Y1);
y = 1:1:m;
x = 1:1:n;
[x,y] = meshgrid(x,y);
figure, mesh(x,y,Y1);
title('This figure shows the decomposed image pixels are
                 decorrelated')
figure, mesh(x,y,X);
title('This figure shows the original image pixels highly are
            Correlated')
```

INVERSE PROGRAM

```
%program idwt

fdwt;                          % activate fdwt to compute the fdwt.
YL = verticalR(YLH3,YLL3);     % first level reconstruction
YH = verticalR(YHH3,YHL3);
YLL2 = horizontalR(YH,YL);

YH = []; YL = [];

YL = verticalR(YLH2,YLL2);     % second level reconstruction
YH = verticalR(YHH2,YHL2);
YLL1 = horizontalR(YH,YL);

YL = [];   YH = [];

YL = verticalR(YLH1,YLL1);     % third level reconstruction
YH = verticalR(YHH1,YHL1);
xr1 = horizontalR(YH,YL);      % reconstructed image

xr = mat2gray(xr1); % convert matlab image to a grayscale image.
```

```
figure, imshow(xr)        % display the reconstructed image.
title('Reconstructed image')

figure, imshow(X1)                  % display the original mage.
title('Original image')

DIFF = difference(xr1,X)            %call function difference
```

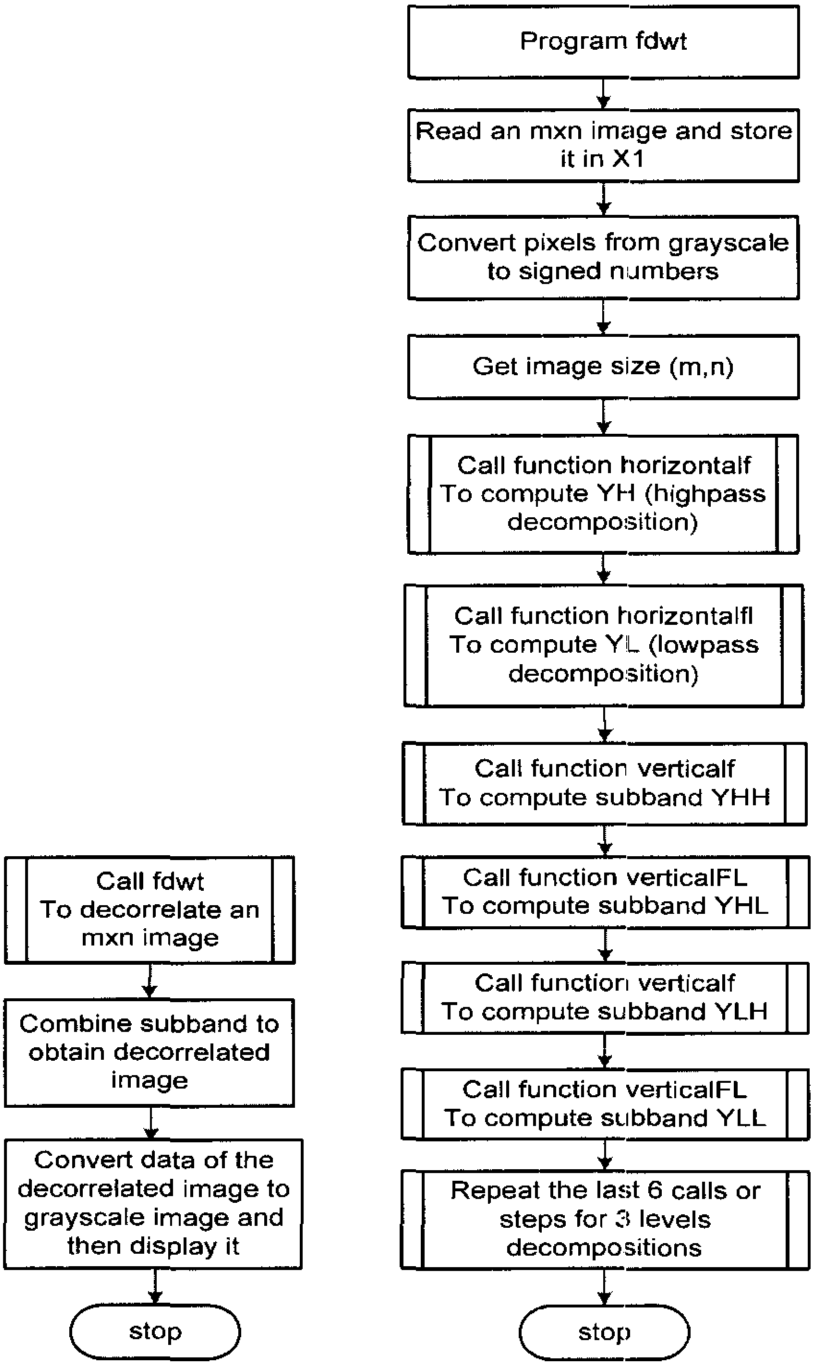
```
function Xrec = horizontalR(YH,YL)  %horizontal reconstruction
   [m,n1] = size(YL);
   [m,n] = size(YH); Xrec = zeros(m,n+n1);
   for i = 1:m
       for j = 1:n1         %horizontal lowpass reconstruction
           if j == 1
               Xrec(i,2*j-1) = YL(i,j) - fix(YH(i,j)/2);
           else if (n1 == n) | (j < n1)
               Xrec(i,2*j-1) = YL(i,j) - fix((YH(i,j-1)+YH(i,j)+2)/4);
               else Xrec(i,2*j-1) = YL(i,j) - fix(YH(i,j-1)/2);
               end
           end
       end
   end
   for i = 1:m
       for j = 1:n         % horizontal highpass reconstruction
           if (j < n) | (n1 ~= n)
               Xrec(i,2*j) = YH(i,j) + fix((Xrec(i,2*j-1) +
Xrec(i,2*j+1))/2);
           else
               Xrec(i,2*j) = YH(i,j) + Xrec(i,2*j-1);
           end
       end
   end
```

```
function YL = verticalR(YLH,YLL)     %vertical reconstruction.
[m,n] = size(YLH);
[m1,n] = size(YLL);
if m1 ~= m
YL = zeros(2*m+1,n);
else YL = zeros(2*m,n);
```

```
end
for i = 1:n                  % vertical lowpass reconstruction
    for j = 1:m1
        if j == 1
            YL(2*j-1,i) = YLL(j,i) - fix(YLH(j,i)/2);
        else if (m1 == m) | (j < m1)
            YL(2*j-1,i) = YLL(j,i) - fix((YLH(j-1,i)+YLH(j,i)+2)/4);
            else YL(2*j-1,i) = YLL(j,i) - fix(YLH(j-1,i)/2);
             end
        end
    end
end
for i = 1:n                 %vertical highpass reconstruction
    for j = 1:m
        if (j < m) | (m1 ~= m)
            YL(2*j,i) = YLH(j,i) + fix((YL(2*j-1,i) + YL(2*j+1,i))/2);
        else
            YL(2*j,i) = YLH(j,i) + YL(2*j-1,i);
        end
    end
end
```

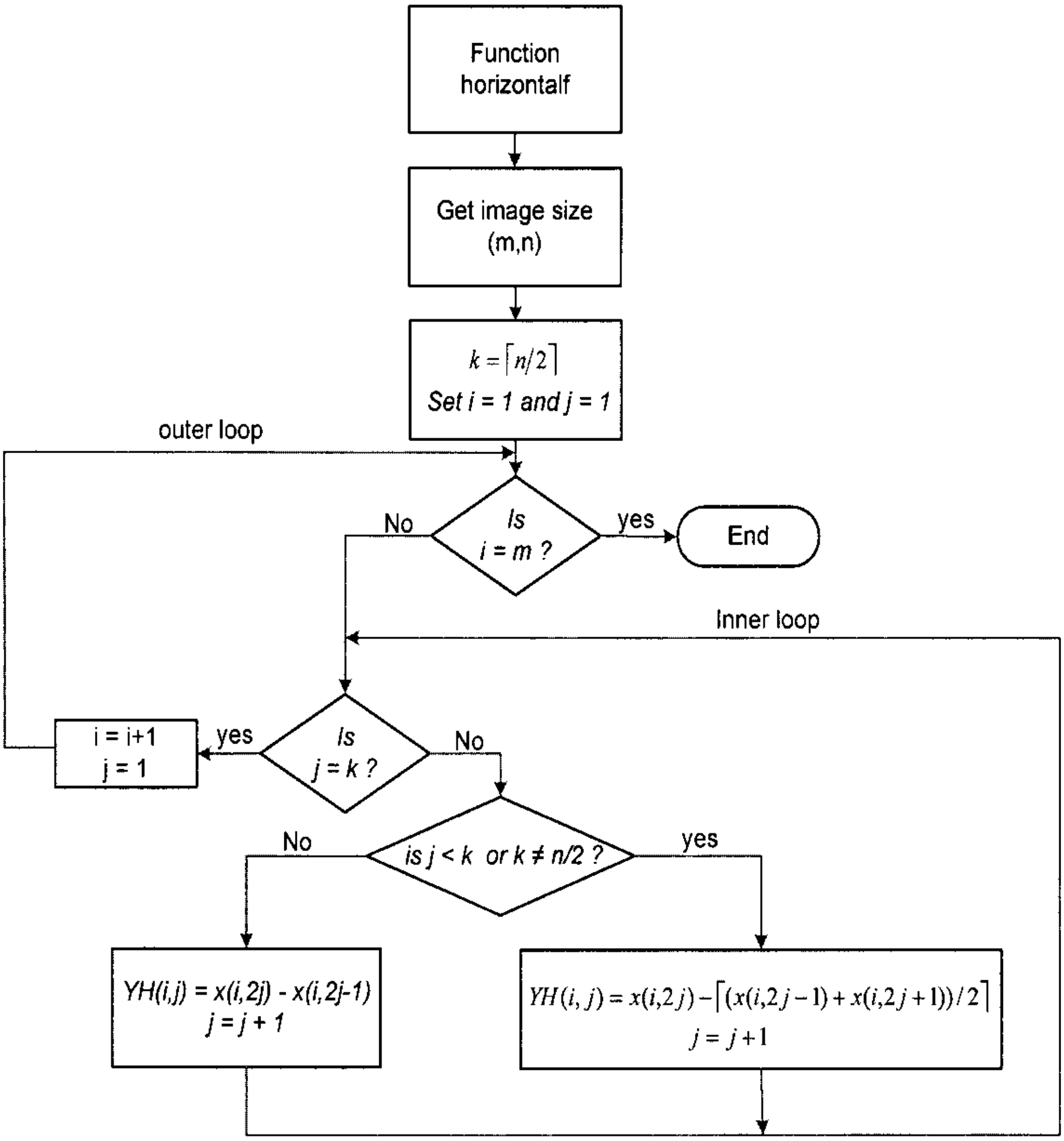
```
function diff = difference(x1,x2)% This function computes the
difference %between the original image and the reconstructed image.
[m,n] = size(x2);
z=0;
for i = 1 : m
    for j = 1 : n
        z = z + (x1(i,j) - x2(i,j));  % compute differences.
    end
end
if z = 0
    disp('the orginal and the reconstructed images are identical')
     disp('We have a perfect reconstruction')
end
else disp(‘the original and the reconstructed images are not
identical’)
```

The flowcharts for both forward and inverse 2-D DWT programs are shown in Figures A.3.1 and A.3.2, respectively. Note that in the forward program, the flowcharts for functions verticalFL and verticalf are similar to the flowcharts for functions horizontalfl and horizontalf, respectively. The only difference is that the vertical functions compute column-wise, whereas, the horizontal functions compute row-wise.


Call fdwt
To decorrelate an mxn image
Combine subband to obtain decorrelated image
Convert data of the decorrelated image to grayscale image and then display it
stop
Program fdwt
Read an mxn image and store it in X1
Convert pixels from grayscale to signed numbers
Get image size (m,n)
Call function horizontalf
To compute YH (highpass decomposition)
Call function horizontalfl
To compute YL (lowpass decomposition)
Call function verticalf
To compute subband YHH
Call function verticalFL
To compute subband YHL
Call function verticalf
To compute subband YLH
Call function verticalFL
To compute subband YLL
Repeat the last 6 calls or steps for 3 levels decompositions
stop

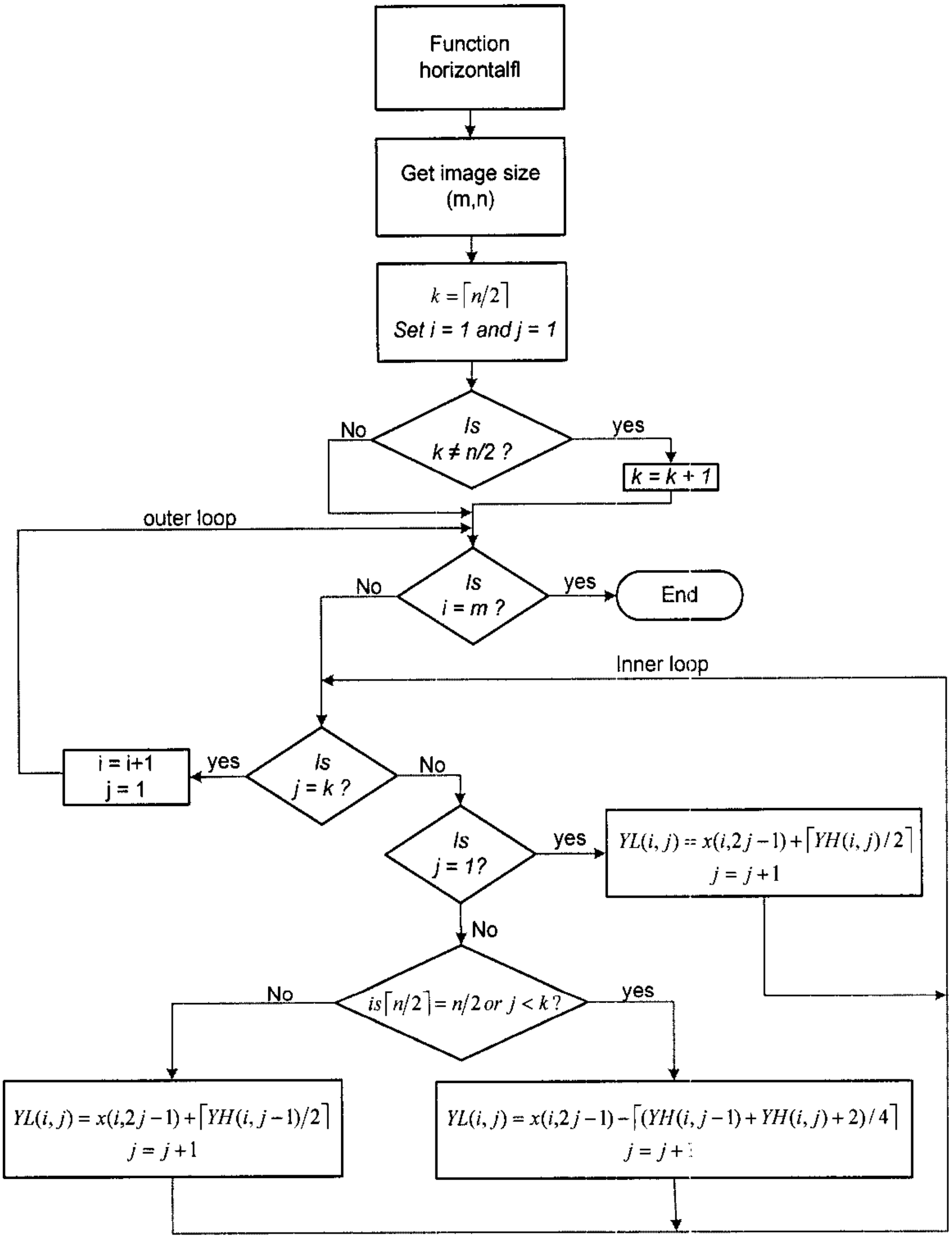
(a) (b)

Figure A.3.1 (a) Main program (b) Forward program.



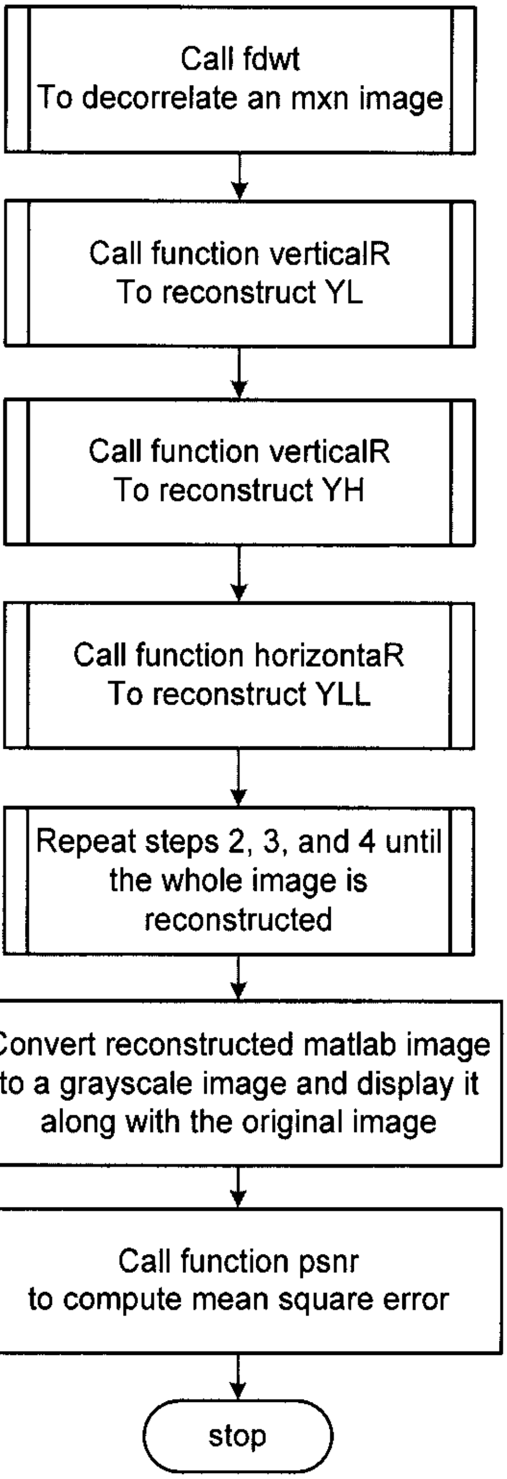
(c)

Figure A.3.1 (c) Horizontal highpass decomposition flowchart



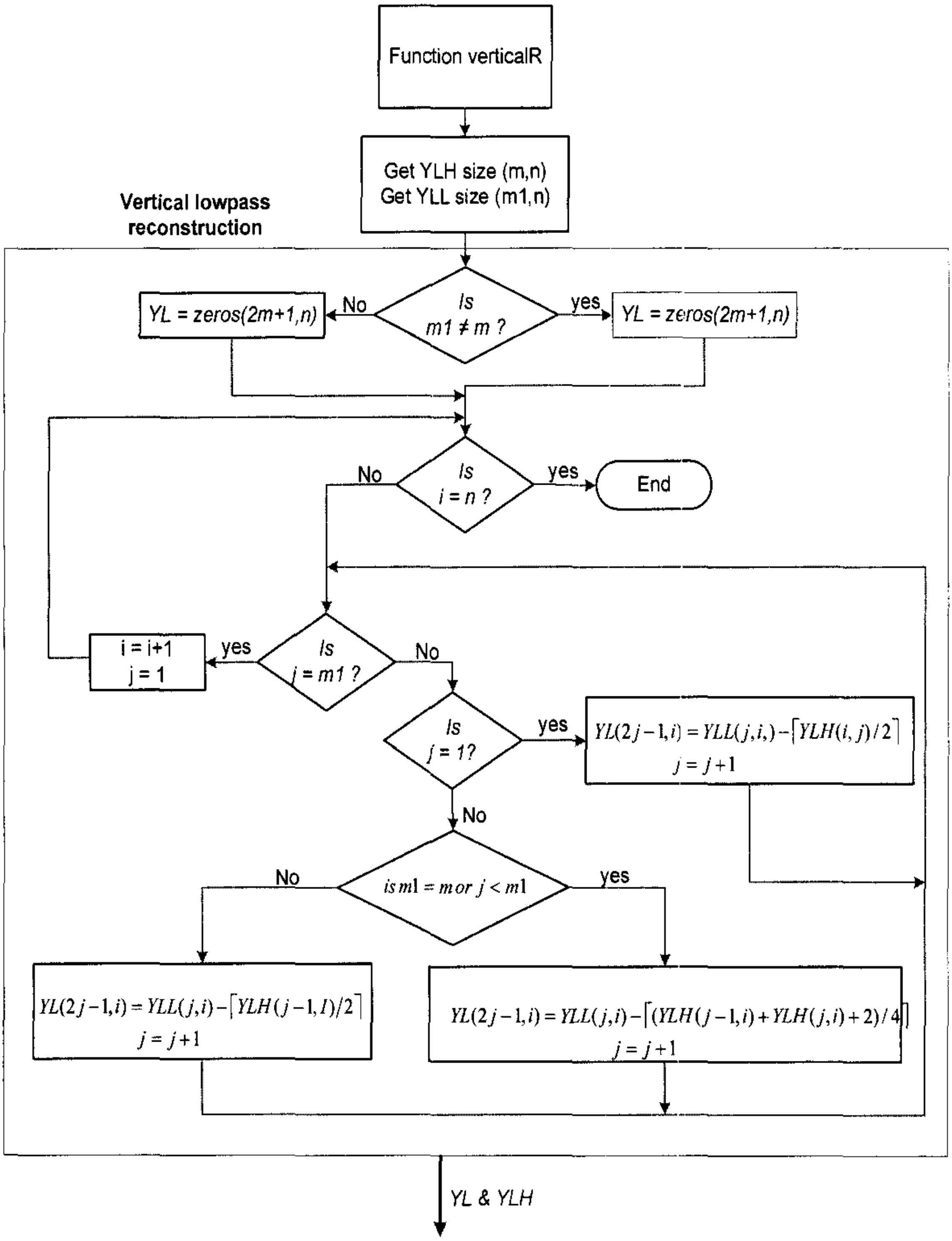
(d)

Figure A.3.1 (d) Horizontal lowpass decomposition flowchart.


Call fdwt
To decorrelate an mxn image
Call function verticalR
To reconstruct YL
Call function verticalR
To reconstruct YH
Call function horizontaR
To reconstruct YLL
Repeat steps 2, 3, and 4 until the whole image is reconstructed
Convert reconstructed matlab image to a grayscale image and display it along with the original image
Call function psnr
to compute mean square error
stop

(a)

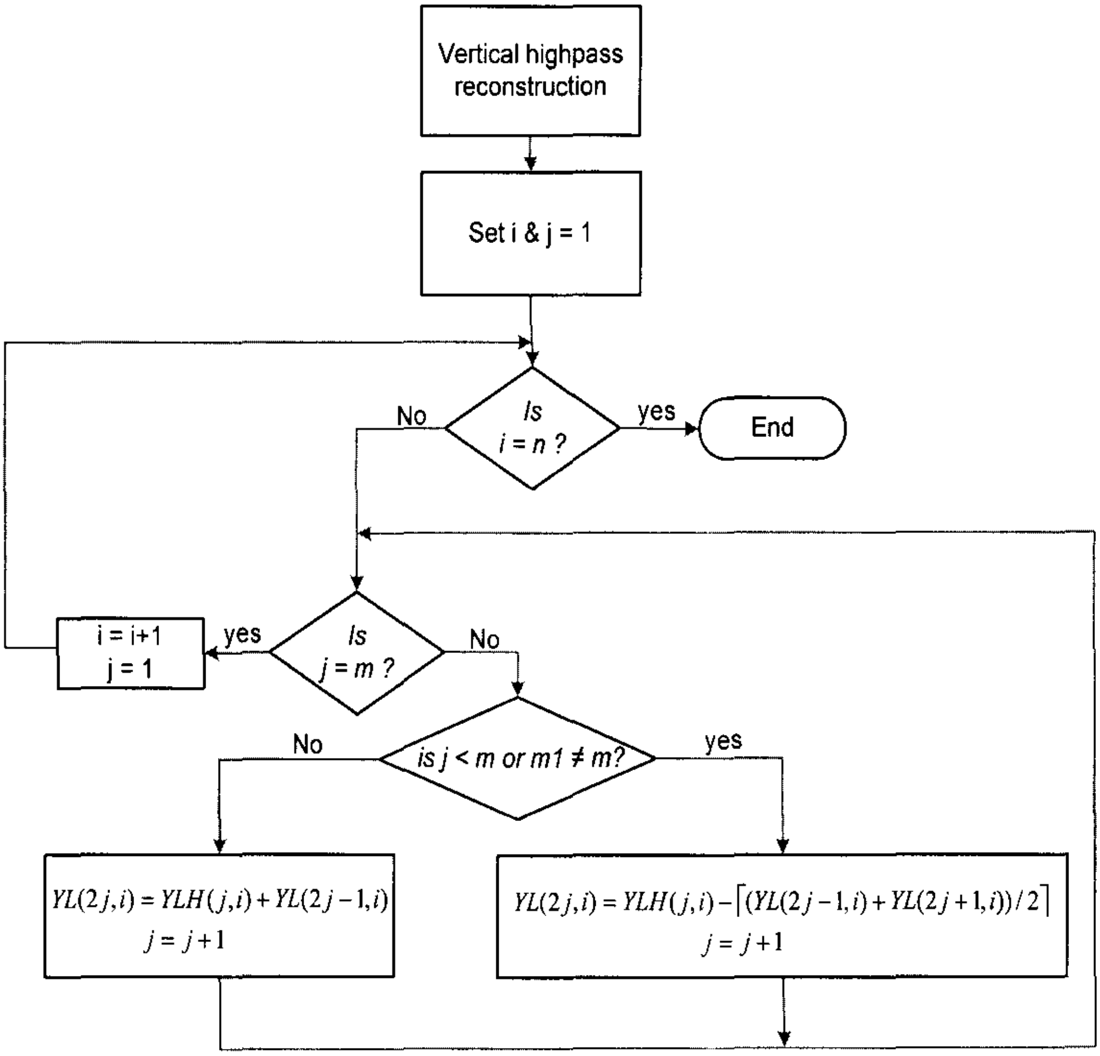
Figure A.3.2 (a) Inverse Program


Function verticalR
Get YLH size (m,n)
Get YLL size (m1,n)
Vertical lowpass reconstruction
YL = zeros(2m+1,n)
No
Is m1 ≠ m ?
yes
YL = zeros(2m+1,n)
No
Is i = n ?
yes
End
i = i+1
j = 1
yes
Is j = m1 ?
No
Is j = 1?
yes
$YL(2j-1,i) = YLL(j,i,) - \lceil YLH(i,j)/2 \rceil$
$j = j+1$
No
No
$is\, m1 = m\, or\, j < m1$
yes
$YL(2j-1,i) = YLL(j,i) - \lceil YLH(j-1,I)/2 \rceil$
$j = j+1$
$YL(2j-1,i) = YLL(j,i) - \lceil (YLH(j-1,i) + YLH(j,i) + 2)/4 \rceil$
$j = j+1$
YL & YLH

Continue to the next page

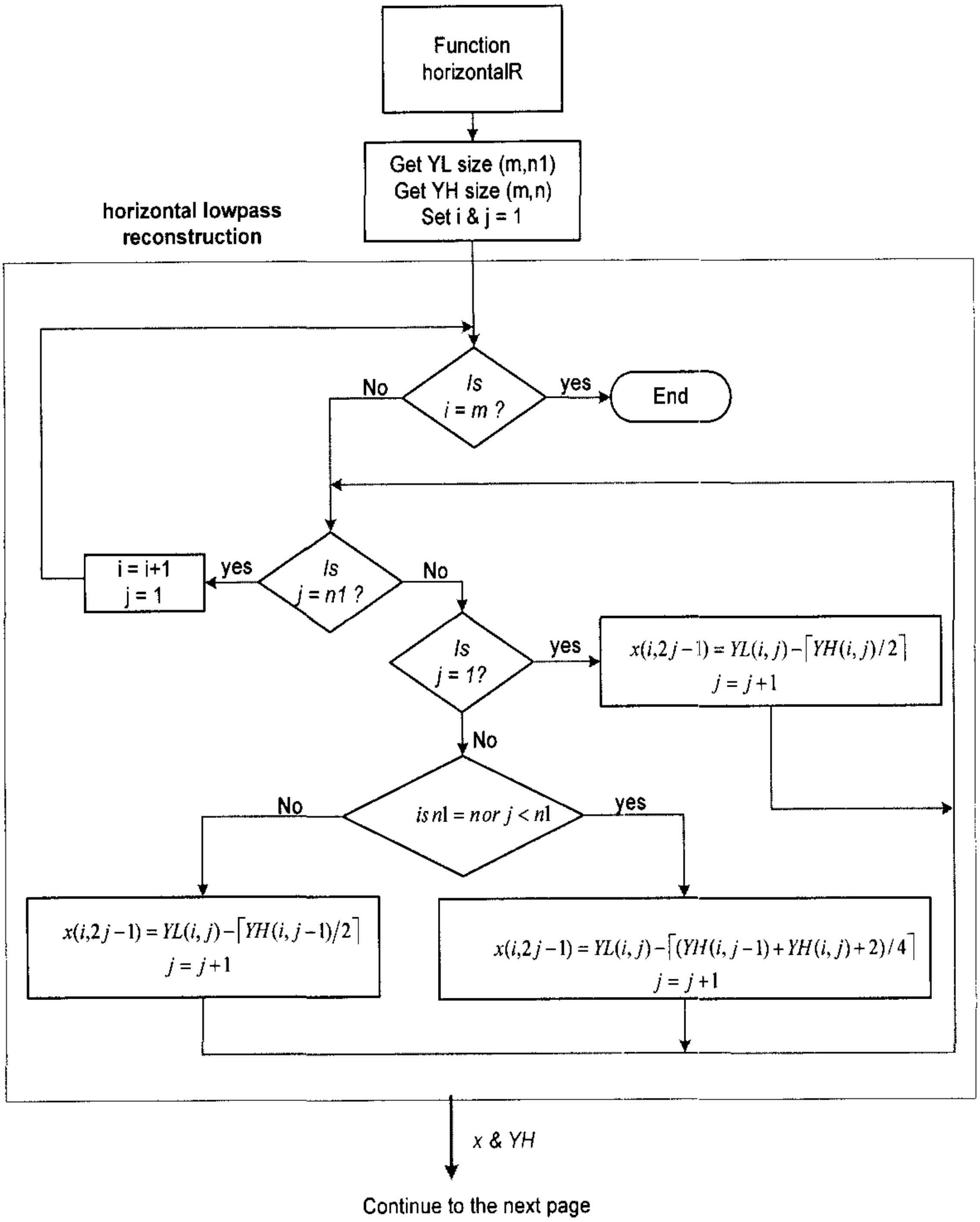
(b)

Figure A.3.2 (b) Vertical lowpass flowchart.


Vertical highpass reconstruction
Set i & j = 1
Is $i = n$?
No
yes
End
Is $j = m$?
yes
No
$i = i+1$
$j = 1$
is $j < m$ or $m1 \neq m$?
No
yes
$YL(2j,i) = YLH(j,i) + YL(2j-1,i)$
$j = j+1$
$YL(2j,i) = YLH(j,i) - \lceil (YL(2j-1,i) + YL(2j+1,i))/2 \rceil$
$j = j+1$

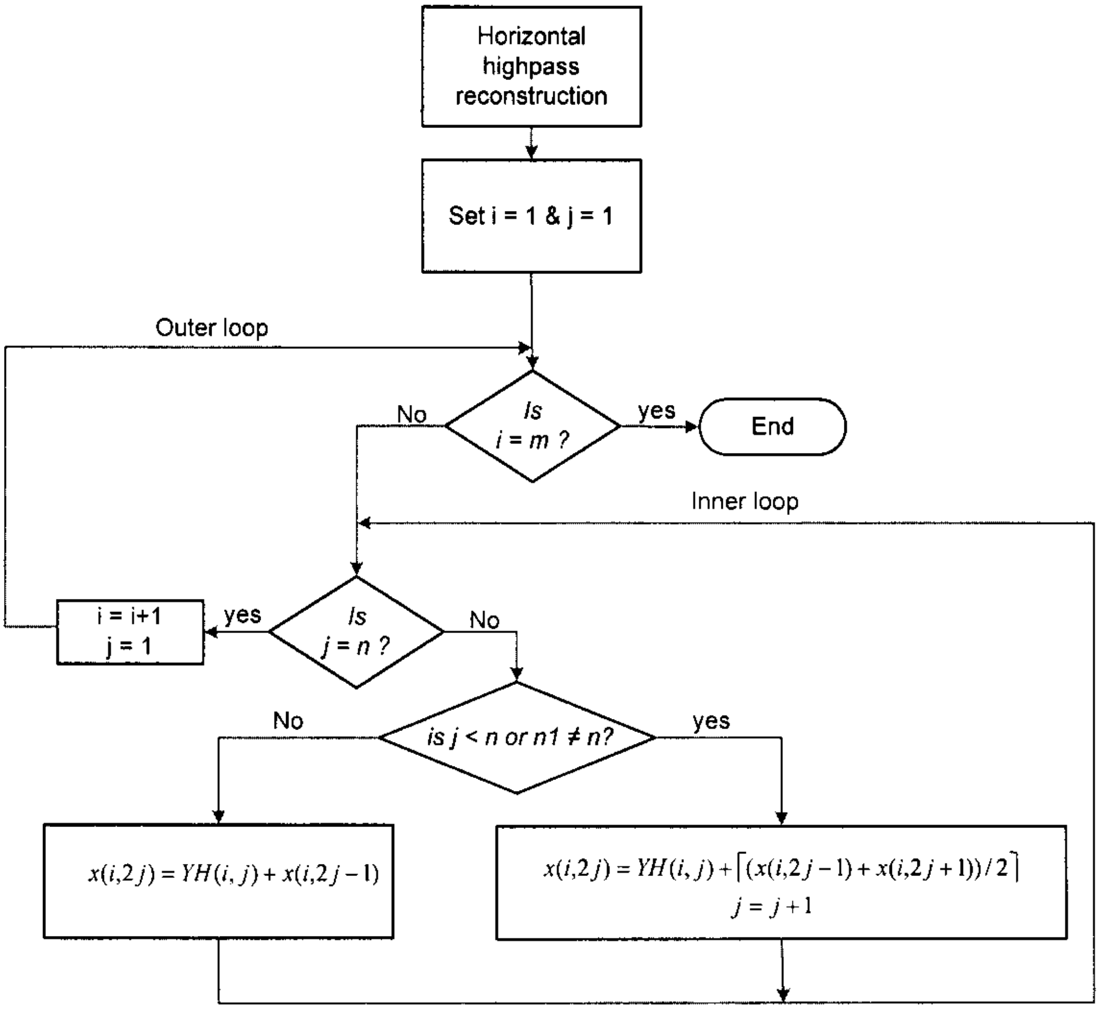
(c)

Figure A.3.2 (c) Vertical highpass reconstruction flowchart.



(d)

Figure A.3.2 (d) Horizontal lowpass flowchart.


Horizontal highpass reconstruction
Set i = 1 & j = 1
Outer loop
Is i = m ?
No
yes
End
Inner loop
i = i+1
j = 1
yes
Is j = n ?
No
No
is j < n or n1 ≠ n?
yes
x(i,2j) = YH(i,j) + x(i,2j − 1)
x(i,2j) = YH(i,j) + ⌈(x(i,2j − 1) + x(i,2j + 1))/2⌉
j = j + 1

(e)

Figure A.3.2 (e) Horizontal highpass reconstruction flowchart.

The forward program consists of 6 parts: *program fdwt*, reads in the original image to be decomposed and then calls appropriate functions to decompose (decorrelate) it, the *horizontalf* and *horizontalfL* functions compute DWT in the horizontal direction to yield the highpass (H) and the lowpass (L) decompositions, respectively, the *verticalfh* function computes DWT in the vertical direction to decompose H into subbands HH and HL, the *verticalfl* function computes DWT in the vertical direction to decompose L into subbands LH and LL. The last part of the forward program is *program f2dwt* (the main program). This program combines subbands of the decomposed image to form the decorrelated image. Then, it displays the decorrelated image and plots pixels of the original and decorrelated images to show correlation and decorrelation properties, respectively.

The forward program is activated by typing at the prompt "f2dwt", which activates the main program. The main program in turn calls *fdwt*. The four functions named *horizontalf*, *horizontalfl*, *verticalf*, and *verticalFL* are called from program *fdwt*, for example, in the first level decomposition as follows. First, the *horizontalf* function is called followed by the *horizontalfl* function to yield H and L decompositions, which are stored in YH and YL, respectively. Then, function *verticalf* is called with YH as a parameter to yield HH, which is stored in YHH1. Next function *verticalFL* is called with YH and YHH1 as parameters to yield subband HL which is stored in YHL1. Again, function *verticalf* is called, with YL as a parameter, to yield subband LH which is stored in YLH1. Then, function verticalFL is called with YL and YLH1 as parameters to yield subband LL which is stored in YLL1. This process is repeated in each decomposition level until the entire image is decomposed into the desire number of levels.

On the other hand, the inverse program consists of 4 parts (functions): *idwt*, *verticalR*, *horizontalR*, and *psnr*. The *verticalR* function reconstructs the original by combining in each level subbands LH and LL into L and subband HH and HL into H. Whereas, the function of *horizontalR*, in each level, is to combine H and L decompositions to form the next LL subband..

The *difference* function computes the difference between the original image (X1) and the reconstructed image (X2) using the following formula [8, 15].

$$z = \sum_{i=1}^{M} \sum_{j=1}^{N} (X1(i,j) - X2(i,j)) \tag{A.1}$$

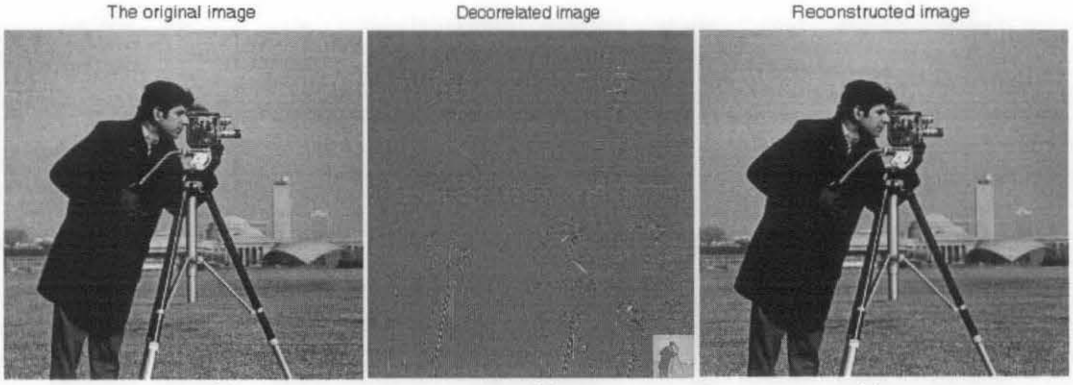
If the difference (z) is zero, the two images' pixels are identical; otherwise, the two images' pixels are not identical.

The function of the *idwt* is to reconstruct the original image by calling *verticalR* and *horizobntalR*. The inverse program is activated by typing at the prompt "idwt", which activates program *idwt*. Then, *idwt* calls *fdwt* to decorrelate the image. The reconstruction process for the first level begin by calling *verticalR* with YLH3 and YLL3 as parameters to yield L3 decomposition which is stored in YL. Again, function *verticalR* is called with YHH3 and YHL3 as parameters to yield H3 decomposition which is stored in YH. Then, function *horizontalR* is called with YH and YL as parameters to yield subband LL2 which is stored in YLL2. This completes the first level reconstruction. For each subsequence level reconstruction the above steps are repeated until the whole image is reconstructed. When this is done, *idwt* dislays both the original and reconstructed images and then call *psnr* to compute signal-to-noise ratio (SNR) between the original image and reconstructed image.

The following figures show simulation results of applying an original image to the software simulation program. When the image shown in Figure A.3.3 (a) was processed by the forward simulation program, the result was the image shown in Figure A.3.3 (b), which is the wavelet representation of the decorrelated image. Then the decorrelated image is applied to the inverse software program to yield the image shown in

Figure A.3.3 (c) which is a perfect reconstruction of the original image without distortion in the image boundaries.

On the other hand, the result of the simulation in Figures A.3.4 and A.3.5 show clearly the correlation and decorrelation properties, respectively. Figure A.6 shows the original image pixels are highly correlated, while Figure A.3.5 shows the image pixels, which are the result of applying FDWT to the original image pixel, are decorrelated.



(a) (b) (c)

Figure A.3.3 (a) The original image (b) Decorretated image (c) Reconstructed image

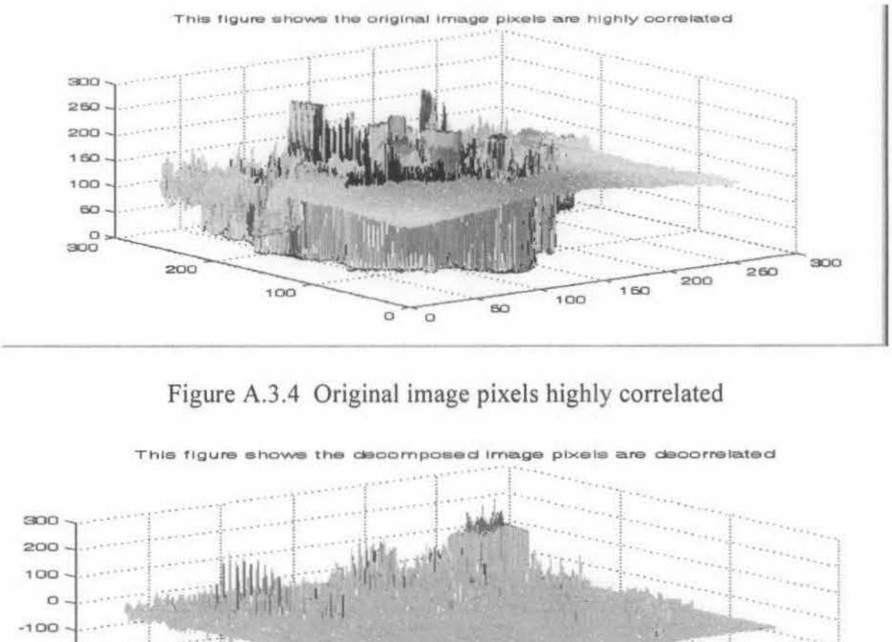


Figure A.3.4 Original image pixels highly correlated

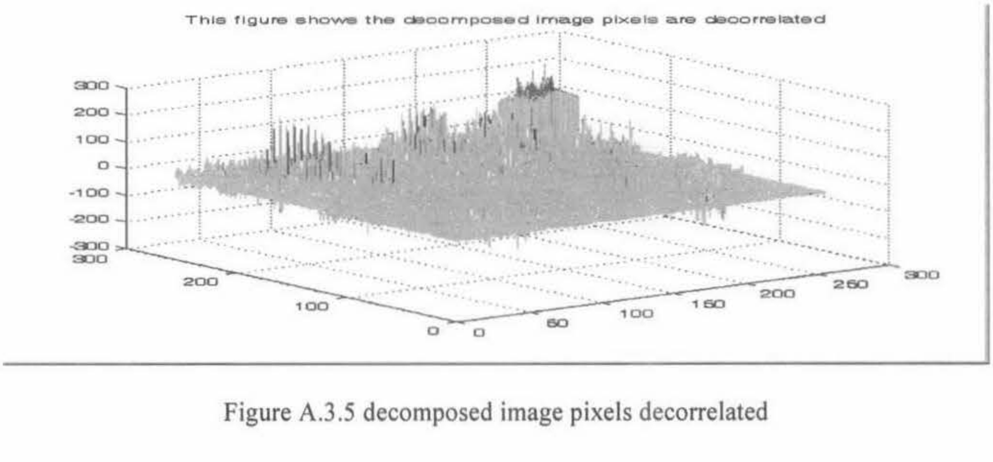


Figure A.3.5 decomposed image pixels decorrelated

APPENDIX B

DATAFLOW AND CONTROL SIGNALS TABLES

B.1 Dataflow tables of chapter 3

Table B.1 Dataflow for 5/3 overlapped and overlapped scan architectures

	Ck *f*	Rd0 Rd1 Rd LB	RP's input latches Rt0 Rt2 Rt1	RP's output latches Rd0 Rd3 Rd4 Rd6 Rd5	Cp's input latches Rt3 Rt4 Rt5	Cp's output atches Rt6 Rt7
Run 1	1	x0,0				
	2	x0,0 x0,1				
	3	---- --- x0,2	x0,0 x0,2 x0,1			
	4	x1,0 --- x0,2				
	5	x1,0 x1,1				
	6	--- ---- x1,2	x1,0 x1,2 x1,1	L0,0 --- H0,0 ---- -----		
	7	x2,0 ---- x1,2				
	8	x2,0 x2,1				
	9	---- --- x2,2	x2,0 x2,2 x2,1	L0,0 L1,0 H0,0 ---- H1,0		
	10	x3,0 --- x2,2				
	11	x3,0 x3,1				
	12	---- --- x3,2	x3,0 x3,2 x3,1	L2,0 ---- H0,0 H2,0 H1,0	L0,0 L2,0 L1,0	
	13	x4,0 --- x3,2				
	14	x4,0 x4,1				
	15	--- --- x4,2	x4,0 x4,2 x4,1	L2,0 L3,0 H2,0 ----- H3,0	H0,0 H2,0 H1,0	LH0,0 LL0,0
	16	x5,0 --- x4,2				
	17	x5,0 x5,1				
	18	--- --- x5,2	x5,0 x5,2 x5,1	L4,0 --- H2,0 H4,0 H3,0	L2,0 L4,0 L3,0	HH0,0 HL0,0
	19	x6,0 --- x5,2				
	20	x6,0 x6,1				
	21	--- --- x6,2	x6,0 x6,2 x6,1	L4,0 L5,0 H4,0 ----- H5,0	H2,0 H4,0 H3,0	LH1,0 LL1,0
	22	--- ---				
	23	--- ---				
	24	--- ---	---- ---- ----	L6,0 ---- H4,0 H6,0 H5,0	L4,0 L6,0 L5,0	HH1,0 HL1,0
Run 2	25	x0,2 ---				
	26	x0,2 x0,3				
	27	---- --- x0,4	x0,2 x0,4 x0,3	L6,0 ----- H6,0 ----- -----	H4,0 H6,0 H5,0	LH2,0 LL2,0
	28	x1,2 --- x0,4				
	29	x1,2 x1,3				
	30	---- ---- x1,4	x1,2 x1,4 x1,3	L0,1 ----- H6,0 H0,1 -----	L6,0 ----- -----	HH2,0 HL2,0
	31	x2,2 ---- x1,4				
	32	x2,2 x2,3				
	33	---- ---- x2,4	x2,2 x2,4 x2,3	L0,1 L1,1 H0,1 ----- H1,1	H6,0 ----- -----	------ LL3,0
	34	x3,2 ---- x2,4				
	35	x3,2 x3,3				
	36	---- ---- x3,4	x3,2 x3,4 x3,3	L2,1 ---- H0,1 H2,1 H1,1	L0,1 L2,1 L1,1	------ HL3,0
	37	x4,2 ---- x3,4				
	38	x4,2 x4,3				
	39	---- ---- x4,4	x4,2 x4,4 x4,3	L2,0 L3,1 H2,1 ----- H3,1	H0,1 H2,1 H1,1	LH0,1 LL0,1
	40	x5,2 ---- x4,4				
	41	x5,2 x5,3				
	42	---- ---- x5,4	x5,2 x5,4 x5,3	L4,1 ---- H2,1 H4,1 H3,1	L2,1 L4,1 L3,1	HH0,1 HL0,1
	43	x6,2 ---- x5,4				
	44	x6,2 x6,3				
	45	---- ---- x6,4	x6,2 x6,4 x6,3	L4,1 L5,1 H4,1 ----- H5,1	H2,1 H4,1 H3,1	LH1,1 LL1,1

Note that in Table B.1 at cycles 22, 23, and 24 the external memory is not scanned and no pixels are loaded into RP latches Rt0, Rt1, and Rt2 at cycle 24 where a transition from run 1 to run 2 is made. This is only required every time a transition from a run to the next is made when the column length N of an image is odd.

Dataflow tables for the second 9/7 pipelined overlapped architecture, developed based on the scan method shown in Figure 3.5.3, are shown in Tables B.2 (a) and (b), respectively. Note that when the column length N of an image is odd, after the second run, an empty cycle should be inserted whenever a transition is made from a run to the next as shown for example at cycle 22 in Table B.2 (b).

Control signal values for signals Ed2, Ed3, Ed4, Ed5, Ed6, S0, and S1 derived from Table B.2(a) are shown in Table B.2 (c). Note that number of control signals in Table B.2 (c) can be reduced to 3 signals by observing that signals Ed2, Ed6, S0 and S1 are equal and so are signals Ed3 and Ed5.

Table B.2 (a) Dataflow of the second 9/7 pipelined overlapped architecture for even N

	ck	RP's input latches Rt0 Rt2 Rt1	RP's output Latches Rd2 Rd3	 Rd4 Rd6 Rd5	CP's input Latches Rt3 Rt4 Rt5	CP's output Latches Rt6 Rt7
Run1	1	x0,0 x0,2 x0,1				
	2	x0,2 x0,4 x0,3				
	3	x1,0 x1,2 x1,1				
	4	x1,2 x1,4 x1,3				
	5	x2,0 x2,2 x2,1				
	6	x2,2 x2,4 x2,3				
	7	x3,0 x3,2 x3,1				
	8	x3,2 x3,4 x3,3				
	9	x4,0 x4,2 x4,1				
	10	x4,2 x4,4 x4,3	L0,0 ---	H0,0		
	11	x5,0 x5,2 x5,1	L0,0 ---	H0,0		
	12	x5,2 x5,4 x5,3	L0,0 L1,0	H0,0 ---- H1,0		
Run2	13	x0,4 x0,6 x0,5	L0,0 L1,0	H0,0 ---- H1,0		
	14	x1,4 x1,6 x1,5	L2,0 ---	H0,0 H2,0 H1,0	L0,0 L2,0 L1,0	
	15	x2,4 x2,6 x2,5	L2,0 ---	H0,0 H2,0 H1,0	H0,0 H2,0 H1,0	
	16	x3,4 x3,6 x3,5	L2,0 L3,0	H2,0 ---- H3,0	----- ----- -----	
	17	x4,4 x4,6 x4,5	L2,0 L3,0	H2,0 ---- H3,0	----- ----- -----	
	18	x5,4 x5,6 x5,5	L4,0 ---	H2,0 H4,0 H3,0	L2,0 L4,0 L3,0	
Run3	19	x0,6 x0,8 x0,7	L4,0 ---	H2,0 H4,0 H3,0	H2,0 H4,0 H3,0	
	20	x1,6 x1,8 x1,7	L4,0 L5,0	H4,0 ---- H5,0	----- ----- -----	
	21	x2,6 x2,8 x2,7	L4,0 L5,0	H4,0 ---- H5,0	----- ----- -----	
	22	x3,6 x3,8 x3,7	L0,1 ----	H4,0 H0,1 H5,0	L4,0 L4,0 L5,0	
	23	x4,6 x4,8 x4,7	L0,1 L1,1	H0,1 ---- H1,1	H4,0 H4,0 H5,0	LH0,0 LL0,0
	24	x5,6 x5,8 x5,7	L2,1 ----	H0,1 H2,1 H1,1	L0,1 L2,1 L1,1	HH0,0 HL0,0
	25		L2,1 L3,1	H2,1 ---- H3,1	H0,1 H2,1 H1,1	------- -------
	26		L4,1 ----	H2,1 H4,1 H3,1	L2,1 L4,1 L3,1	------- -------
	27		L4,1 L5,1	H4,1 ---- H5,1	H2,1 H4,1 H3,1	LH1,0 LL1,0
	28		L0,2 ----	H4,1 H0,2 H5,1	L4,1 L4,1 L5,1	HH1,0 HL1,0
	29		L0,2 L1,2	H0,2 ---- H3,1	H4,1 H4,1 H5,1	------- -------
	30		L2,2 ----	H0,2 H2,2 H1,2	L0,2 L2,2 L1,2	------- -------
	31		L2,2 L3,2	H2,2 ---- H3,2	H0,2 H2,2 H1,2	LH2,0 LL2,0
	32		L4,2 ----	H2,2 H4,2 H3,2	L2,2 L4,2 L3,2	HH2,0 HL2,0
	33		L4,2 L5,2	H4,2 ---- H5,2	H2,2 H4,2 H3,2	LH0,1 LL0,1
	34				L4,2 L4,2 L5,2	HH0,1 HL0,1
	35				H4,2 H4,2 H5,2	LH1,1 LL1,1
	36					HH1,1 HL1,1
	37					LH2,1 LL2,1
	38					HH2,1 HL2,1

Table B.2 (b) Dataflow of the second 9/7 pipelined overlapped architecture for odd N

	ck	RP's input latches Rt0 Rt2 Rt1	RP's output Latches Rd2 Rd3	 Rd4 Rd6 Rd5	CP's input Latches Rt3 Rt4 Rt5	CP's output Latches Rt6 Rt7
Run 1	9	x4,0 x4,2 x4,1				
	10	x4,2 x4,4 x4,3	L0,0 ---	H0,0		
	11	x5,0 x5,2 x5,1	L0,0 ---	H0,0		
	12	x5,2 x5,4 x5,3	L0,0 L1,0	H0,0 ---- H1,0		
	13	x6,0 x6,2 x6,1	L0,0 L1,0	H0,0 ---- H1,0		
	14	x6,2 x6,4 x6,3	L2,0 ---	H0,0 H2,0 H1,0	L0,0 L2,0 L1,0	
Run 2	15	x0,4 x0,6 x0,5	L2,0 ---	H0,0 H2,0 H1,0	H0,0 H2,0 H1,0	
	16	x1,4 x1,6 x1,5	L2,0 L3,0	H2,0 ---- H3,0	----- ----- -----	
	17	x2,4 x2,6 x2,5	L2,0 L3,0	H2,0 ---- H3,0	----- ----- -----	
	18	x3,4 x3,6 x3,5	L4,0 ---	H2,0 H4,0 H3,0	L2,0 L4,0 L3,0	
	19	x4,4 x4,6 x4,5	L4,0 ---	H2,0 H4,0 H3,0	H2,0 H4,0 H3,0	
	20	x5,4 x5,6 x5,5	L4,0 L5,0	H4,0 ---- H5,0	----- ----- -----	
	21	x6,4 x6,6 x6,5	L4,0 L5,0	H4,0 ---- H5,0	----- ----- -----	
	22	----- ---- -----	L6,0 ----	H4,0 H6,0 H5,0	L4,0 L6,0 L5,0	
Run 3	23	x0,6 x0,8 x0,7	L6,0 ----	H6,0 ---- -----	H4,0 H6,0 H5,0	LH0,0 LL0,0
	24	x1,6 x1,8 x1,7	L0,1 ----	H6,0 H0,1 ----	L6,0 ----- -----	HH0,0 HL0,0
	25	x2,6 x2,8 x2,7	L0,1 L1,1	H0,1 ---- H1,1	H6,0 ----- -----	------- -------
	26	x3,6 x3,8 x3,7	L2,1 ----	H0,1 H2,1 H1,1	L0,1 L2,1 L1,1	------- -------
	27	x4,6 x4,8 x4,7	L2,1 L3,1	H2,1 ---- H3,1	H0,1 H2,1 H1,1	LH1,0 LL1,0
	28	x5,6 x5,8 x5,7	L4,1 ----	H2,1 H4,1 H3,1	L2,1 L4,1 L3,1	HH1,0 HL1,0
	29	x6,6 x6,8 x6,7	L4,1 L5,1	H4,1 ---- H5,1	H2,1 H4,1 H3,1	------- -------
	30		L6,1 ----	H4,1 H6,1 H5,1	L4,1 L6,1 L5,1	------- -------
	31		L6,1 ----	H6,1 ---- -----	H4,1 H6,1 H5,1	LH2,0 LL2,0
	32		L0,2 ----	H6,1 H0,2 ----	L6,1 ----- -----	HH2,0 HL2,0
	33		L0,2 L1,2	H0,2 ---- H1,2	H6,1 ----- -----	----- LL3,0
	34		L2,2 ----	H0,2 H2,2 H1,2	L0,2 L2,2 L1,2	----- HL3,0
	35		L2,2 L3,2	H2,2 ---- H3,2	H0,2 H2,2 H1,2	LH0,1 LL0,1
	36		L4,2 ----	H2,2 H4,2 H3,2	L2,2 L4,2 L3,2	HH0,1 HL0,1
	37		L4,2 L5,2	H4,2 ---- H5,2	H2,2 H4,2 H3,2	LH1,1 LL1,1
	38		L6,2 ----	H4,2 H6,2 H5,2	L4,2 L6,2 L5,2	HH1,1 HL1,1
	39		L6,2 ----	H6,2 ---- -----	H4,2 H6,2 H5,2	LH2,1 LL2,1
	40			H6,2 ---- -----	L6,2 ---- -----	HH2,1 HL2,1
	41				H6,2 ---- -----	------ LL3,1
						------ HL3,1
						LH0,3 LL0,3
						HH0,3 HL0,3

Table B.2 (c) Control signal values

	clock	Ed2	Ed3	Ed4	Ed5	Ed6	S0	S1
Run 1	10	1	X	1	X	X	1	X
	11	0	X	0	X	X	X	X
	12	0	1	0	1	X	X	X
	13	0	0	0	0	X	X	X
	14	1	X	0	0	1	X	1
	15	0	X	0	0	0	X	0
	16	0	1	1	1	X	0	X
	17	0	0	0	0	X	X	X
	18	1	X	0	0	1	X	1
	19	0	X	0	0	0	X	0
	20	0	1	1	1	X	0	X
	21	0	0	0	0	X	X	X
Run 2	22	1	X	0	0	1	X	1
	23	0	1	1	1	X	0	0
	24	1	X	0	0	1	X	1
	25	0	1	1	1	X	0	0
	26	1	X	0	0	1	X	1
	27	0	1	1	1	X	0	0
	28	1	X	0	0	1	X	1
	29	0	1	1	1	X	0	0
	30	1	X	0	0	1	X	1

Table B.3 Dataflow of the intermediate architecture

						YH									YL										
						SR0			SR2			SR1			SR0			SR1							
Clk	Rd0	Rd1	Rt0	Rt2	Rt1	R2	R1	R0	R2	R1	R0	R2	R1	R0	R2	R1	R0	R2	R1	R0	Rt3	Rt4	Rt5	Rt6	Rt7
1	x0,0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	x0,0	x0,1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	x0,2	-	x0,0	x0,2	x0,1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	x0,2	x0,3				-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	x0,2	x0,3				-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	x0,4	-	x0,2	x0,4	x0,3	h0,0	-	-	-	-	-	-	-	-	L0,0	-	-	-	-	-	-	-	-	-	-
7	x0,4	x0,5					-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-
8	x0,4	x0,5					-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-
9	-	-	x0,4	x0,6	x0,5	h0,1	h0,0	-	-	-	-	-	-	-	L0,1	L0,0	-	-	-	-	-	-	-	-	-
10	x1,0	-						-	-	-	-	-	-	-			-	-	-	-	-	-	-	-	-
11	x1,0	x1,1						-	-	-	-	-	-	-			-	-	-	-	-	-	-	-	-
12	x1,2	-	x1,0	x1,2	x1,1	h0,2	h0,1	h0,0	-	-	-	-	-	-	L0,2	L0,1	L0,0	-	-	-	-	-	-	-	-
13	x1,2	x1,3							-	-	-	-	-	-				-	-	-	-	-	-	-	-
14	x1,2	x1,3							-	-	-	-	-	-				-	-	-	-	-	-	-	-
15	x1,4	-	x1,2	x1,4	x1,3	h0,2	h0,1	h0,0	-	-	-	h1,0	-	-	L0,2	L0,1	L0,0	L1,0	-	-	-	-	-	-	-
16	x1,4	x1,5							-	-	-		-	-					-	-	-	-	-	-	-
17	x1,4	x1,5							-	-	-		-	-					-	-	-	-	-	-	-
18	-	-	x1,4	x1,6	x1,5	h0,2	h0,1	h0,0	-	-	-	h1,1	h1,0	-	L0,2	L0,1	L0,0	L1,1	L1,0	-	-	-	-	-	-
19	x2,0	-							-	-	-			-						-	-	-	-	-	-
20	x2,0	x2,1							-	-	-			-						-	-	-	-	-	-
21	x2,2	-	x2,0	x2,2	x2,1	h0,2	h0,1	h0,0	-	-	-	h1,2	h1,1	h1,0	L0,2	L0,1	L0,0	L1,2	L1,1	L1,0	-	-	-	-	-
22	x2,2	x2,3							-	-	-										-	-	-	-	-
23	x2,2	x2,3							-	-	-										-	-	-	-	-
24	x2,4	-	x2,2	x2,4	x2,3	h0,2	h0,1	h0,0	h2,0	-	-	h1,2	h1,1	h1,0	L2,0	L0,2	L0,1	-	L1,2	L1,1	L0,0	L2,0	L1,0	-	-
25	x2,4	x2,5								-	-													-	-
26	x2,4	x2,5								-	-													-	-
27	-	-	x2,4	x2,6	x2,5	h0,2	h0,1	h0,0	h2,1	h2,0	-	h1,2	h1,1	h1,0	L2,1	L2,0	L0,2	-	-	L1,2	L0,1	L2,1	L1,1	Lh0,0	LL0,0
28	x3,0	-									-														
29	x3,0	x3,1									-														
30	x3,2	-	x3,0	x3,2	x3,1	h0,2	h0,1	h0,0	h2,2	h2,1	h2,0	h1,2	h1,1	h1,0	L2,2	L2,1	L2,0	-	-	-	L0,2	L2,2	L1,2	Lh0,1	LL0,1
31	x3,2	x3,3																							
32	x3,2	x3,3																							
33	x3,4	-	x3,2	x3,4	x3,3	h2,0	h0,2	h0,1	-	h2,2	h2,1	h3,0	h1,2	h1,1	L2,2	L2,1	L2,0	L3,0	-	-	h0,0	h2,0	h1,0	Lh0,2	LL0,2
34	x3,4	x3,5																							
35	x3,4	x3,5																							
36	-	-	x3,4	x3,6	x3,5	h2,1	h2,0	h0,2	-	-	h2,2	h3,1	h3,0	h1,2	L2,2	L2,1	L2,0	L3,1	L3,0	-	h0,1	h2,1	h1,1	hh0,0	hL0,0
37	x4,0	-																							
38	x4,0	x4,1																							
39	x4,2	-	x4,0	x4,2	x4,1	h2,2	h2,1	h2,0	-	-	-	h3,2	h3,1	h3,0	L2,2	L2,1	L2,0	L3,2	L3,1	L3,0	h0,2	h2,2	h1,2	hh0,1	hL0,1
40	x4,2	x4,3																							
41	x4,2	x4,3																							
42	x4,4	-	x4,2	x4,4	x4,3	h2,2	h2,1	h2,0	h4,0	-	-	h3,2	h3,1	h3,0	L4,0	L2,2	L2,1	-	L3,2	L3,1	L2,0	L4,0	L3,0	hh0,2	hL0,2
43	x4,4	x4,5																							
44	x4,4	x4,5																							
45	-	-	x4,4	x4,6	x4,5	h2,2	h2,1	h2,0	h4,1	h4,0	-	h3,2	h3,1	h3,0	L4,1	L4,0	L2,2	-	-	L3,2	L2,1	L4,1	L3,1	Lh1,0	LL1,0

It is important to keep in mind that each time a transition from a run to the next is made, when the column length of an image is odd, the external memory is not scanned for *i* consecutive clock cycles reference to the processor clock, where *i* = 1, 2 , 3,... denotes first, second, third scan methods, and so on. The reason is that during this period the CP would be processing the last coefficient in *i* columns of each H and L decomposition that were under consideration in the previous run as required by the DDG for odd length signals. No such situation arises when the column length of an image is even.

Table B.4 Second dataflow for intermediate architecture

						YH									YL										
						SR0			SR2			SR1			SR0			SR1							
Clk	Rd0	Rd1	Rt0	Rt2	Rt1	R2	R1	R0	R2	R1	R0	R2	R1	R0	R2	R1	R0	R2	R1	R0	Rt3	Rt4	Rt5	Rt6	Rt7
1	x0,0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	x0,0	x0,1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	x0,2	-	x0,0	x0,2	x0,1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	x0,2	x0,3				-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	x0,4	-	x0,2	x0,4	x0,3	h0,0	-	-	-	-	-	-	-	-	L0,0	-	-	-	-	-	-	-	-	-	-
6	x0,4	x0,5					-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-
7	-	-	x0,4	x0,6	x0,5	h0,1	h0,0	-	-	-	-	-	-	-	L0,1	L0.0	-	-	-	-	-	-	-	-	-
8	x1,0	-						-	-	-	-	-	-	-			-	-	-	-	-	-	-	-	-
9	x1,0	x1,1						-	-	-	-	-	-	-			-	-	-	-	-	-	-	-	-
10	x1,2	-	x1,0	x1,2	x1,1	h0,2	h0,1	h0,0	-	-	-	-	-	-	L0,2	L0,1	L0,0	-	-	-	-	-	-	-	-
11	x1,2	x1,3							-	-	-	-	-	-				-	-	-	-	-	-	-	-
12	x1,4	-	x1,2	x1,4	x1,3	h0,2	h0,1	h0,0	-	-	-	h1,0	-	-	L0,2	L0,1	L0,0	L1,0	-	-	-	-	-	-	-
13	x1,4	x1,5							-	-	-		-	-					-	-	-	-	-	-	-
14	-	-	x1,4	x1,6	x1,5	h0,2	h0,1	h0,0	-	-	-	h1,1	h1,0	-	L0,2	L0,1	L0,0	L1,1	L1,0	-	-	-	-	-	-
15	x2,0	-							-	-	-			-						-	-	-	-	-	-
16	x2,0	x2,1							-	-	-			-						-	-	-	-	-	-
17	x2,2	-	x2,0	x2,2	x2,1	h0,2	h0,1	h0,0	-	-	-	h1,2	h1,1	h1,0	L0,2	L0,1	L0,0	L1,2	L1,1	L1,0	-	-	-	-	-
18	x2,2	x2,3							-	-	-										-	-	-	-	-
19	x2,4	-	x2,2	x2,4	x2,3	h0,2	h0,1	h0,0	h2,0	-	-	h1,2	h1,1	h1,0	L2,0	L0,2	L0,1	-	L1,2	L1,1	L0,0	L2,0	L1,0	-	-
20	x2,4	x2,5								-	-													-	-
21	-	-	x2,4	x2,6	x2,5	h0,2	h0,1	h0,0	h2,1	h2,0	-	h1,2	h1,1	h1,0	L2,1	L2,0	L0,2	-	-	L1,2	L0,1	L2,1	L1,1	Lh0,0	LL0,0
22	x3,0	-									-														
23	x3,0	x3,1									-														
24	x3,2	-	x3,0	x3,2	x3,1	h0,2	h0,1	h0,0	h2,2	h2,1	h2,0	h1,2	h1,1	h1,0	L2,2	L2,1	L2,0	-	-	-	L0,2	L2,2	L1,2	Lh0,1	LL0,1
25	x3,2	x3,3																							
26	x3,4	-	x3,2	x3,4	x3,3	h2,0	h0,2	h0,1	-	h2,2	h2,1	h3,0	h1,2	h1,1	L2,2	L2,1	L2,0	L3,0	-	-	h0,0	h2,0	h1,0	Lh0,2	LL0,2
27	x3,4	x3,5																							
28	-	-	x3,4	x3,6	x3,5	h2,1	h2,0	h0,2	-	-	h2,2	h3,1	h3,0	h1,2	L2,2	L2,1	L2,0	L3,	L3,0	-	h0,1	h2,1	h1,1	hh0,0	hL0,0
29	x4,0	-																							
30	x4,0	x4,1																							
31	x4,2	-	x4,0	x4,2	x4,1	h2,2	h2,1	h2,0	-	-	-	h3,2	h3,1	h3,0	L2,2	L2,1	L2,0	L;3.2	L3,1	L3,0	h0,2	h2,2	h1,2	hh0,1	hL0,1
32	x4,2	x4,3																							
33	x4,4	-	x4,2	x4,4	x4,3	h2,2	h2,1	h2,0	h4,0	-	-	h3,2	h3,1	h3,0	L4,0	L2,2	L2,1	-	L3,2	L3,1	L2,0	L4,0	L3,0	hh0,2	hL0,2
34	x4,4	x4,5																							
35	-	-	x4,4	x4,6	x4,5	h2,2	h2,1	h2,0	h4,1	h4,0	-	h3,2	h3,1	h3,0	L4,1	L4,0	L2,2	-	-	L3,2	L2,1	L4,1	L3,1	Lh1,0	LL1,0

Control signals such as sre0, sre1, sre2, and incar etc., are issued by the control unit and are loaded, in every clock cycle, into the first stage of the RP. Then, these signals are carried from stage-to-stage. When a stage where a signal is used is reached that signal is applied and the reset are carried on to the next stage until the last stage is reached. However, in the 9/7, applying the scan methods such as shown in Figures 3.5.1, 3.5.3, and 3.8.1 would require these signals values of the RP to change as they move from stage-to-stage, especially in the last and extra runs. Tables B.5 (a) and (b) and the circuit shown in Figure B.1.1, which operate according to Table B.5, are provided in order to be applied as described in section 3.8.3.

- Signal sre1 takes on the signal values of Table B.5 (a), when the row length of an image is odd. In the case of even length both sre1 and Q1 are set 0 in all runs.

- Table B.5 (a) is used for signal sre0 for both odd and even length.

- Table B.5 (b) is applied only in the architecture developed based on the scan method of Figure 3.5.1. For the architecture based on the scan method of Figure 3.5.3, signal sre2 is set to alternate between 0 and 1, while Q2 is set 0, in the first run. In all subsequent runs, sre2 and Q2 are set 1 and 0, respectively, as shown in the third row of Table B.5 (b).

Table B.5 (a) control signal values

sre0 sre1	Q0 Q1		
0	0	0	Run 1 to the run before last.
0	0	0	
0	0	0	
0	0	0	
1	1	0	Last run
1	0	1	Extra run

Table B.5 (b) control signal values for sre2

sre2	Q2	
0	X	Run 1
1	1	Run 2
1	0	Run 3 to extra run.

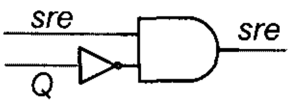


Figure B.1.1 circuit

Table B.6 5/3 Dataflow for overlapped and nonoverlapped parallel scan architecture

Clk	Rt0	Rt2	Rt1	Rd2	Rd3	Rd4	Rd6	Rd5	Rt3	Rt4	Rt5	Rt6	Rt7
1	x0,0	x0,2	x0,1	-	-	-	-	-	-	-	-	-	-
2	x1,0	x1,2	x1,1	-	-	-	-	-	-	-	-	-	-
3	x2,0	x2,2	x2,1	-	-	-	-	-	-	-	-	-	-
4	x3,0	x3,2	x3,1	-	-	-	-	-	-	-	-	-	-
5	x4,0	x4,2	x4,1	L0,0	-	H0,0	-	-	-	-	-	-	-
6	x5,0	x5,2	x5,1	L0,0	L1,0	H0,0	-	H1,0	-	-	-	-	-
7	x6,0	x6,2	x6,1	L2,0	-	H0,0	H2,0	H1,0	L0,0	L2,0	L1,0	-	-
8	x7,0	x7,2	x7,1	L2,0	L3,0	H2,0	-	H3,0	H0,0	H2,0	H1,0	-	-
9	x8,0	x8,2	x8,1	L4,0	-	H2,0	H4,0	H3,0	L2,0	L4,0	L3,0	-	-
10	x9,0	x9,2	x9,1	L4,0	L5,0	H4,0	-	H5,0	H2,0	H4,0	H3,0	LH0,0	LL0,0
11	x10,0	x10,2	x10,1	L6,0	-	H4,0	H6,0	H5,0	L4,0	L6,0	L5,0	HH0,0	HL0,0
12	x11,0	x11,2	x11,1	L6,0	L7,0	H6,0	-	H7,0	H4,0	H6,0	H5,0	LH1,0	LL1,0
13	x12,0	x12,2	x12,1	L8,0	-	H6,0	H8,0	H7,0	L6,0	L8,0	L7,0	HH1,0	HL1,0
14	x13,0	x13,2	x13,1	L8,0	L9,0	H8,0	-	H9,0	H6,0	H8,0	H7,0	LH2,0	LL2,0
15	x14,0	x14,2	x14,1	L10,0	-	H8,0	H10,0	H9,0	L8,0	L10,0	L9,0	HH2,0	HL2,0
16	x15,0	x15,2	x15,1	L10,0	L11,0	H10,0	-	H11,0	H8,0	H10,0	H9,0	LH3,0	LL3,0

Table B.7 5/3 Dataflow for intermediate parallel scan architecture

					YL						YH													
					SR0			SR1			SR0			SR2			SR1							
Clk	Rd	Rt0	Rt2	Rt1	R2	R1	R0	R2	R1	R0	R2	R1	R0	R2	R1	R0	R2	R1	R0	Rt3	Rt4	Rt5	Rt6	Rt7
1	x0,2	x0,0	x0,2	x0,1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	x0,4	x0,2	x0,4	x0,3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	x0,6	x0,4	x0,6	x0,5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	x1,2	x1,0	x1,2	x1,1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	x1,4	x1,2	x1,4	x1,3	L0,0	-	-	-	-	-	H0,0	-	-	-	-	-	-	-	-	-	-	-	-	-
6	x1,6	x1,4	x1,6	x1,5	L0,1	L0,0	-	-	-	-	H0,1	H0,0	-	-	-	-	-	-	-	-	-	-	-	-
7	x2,2	x2,0	x2,2	x2,1	L0,2	L0,1	L0,0	-	-	-	H0,2	H0,1	H0,0	-	-	-	-	-	-	-	-	-	-	-
8	x2,4	x2,2	x2,4	x2,3	L0,2	L0,1	L0,0	L1,0	-	-	H0,2	H0,1	H0,0	-	-	-	H1,0	-	-	-	-	-	-	-
9	x2,6	x2,4	x2,6	x2,5	L0,2	L0,1	L0,0	L1,1	L1,0	-	H0,2	H0,1	H0,0	-	-	-	H1,1	H1,0	-	-	-	-	-	-
10	x3,2	x3,0	x3,2	x3,1	L0,2	L0,1	L0,0	L1,2	L1,1	L1,0	H0,2	H0,1	H0,0	-	-	-	H1,2	H1,1	H1,0	-	-	-	-	-
11	x3,4	x3,2	x3,4	x3,3	L2,0	L0,2	L0,1	-	L1,2	L1,1	H0,2	H0,1	H0,0	H2,0	-	-	H1,2	H1,1	H1,0	L0,0	L2,0	L1,0	-	-
12	x3,6	x3,4	x3,6	x3,5	L2,1	L2,0	L0,2	-	-	L1,2	H0,2	H0,1	H0,0	H2,1	H2,0	-	H1,2	H1,1	H1,0	L0,1	L2,1	L1,1	-	-
13	x4,2	x4,0	x4,2	x4,1	L2,2	L2,1	L2,0	-	-	-	H0,2	H0,1	H0,0	H2,2	H2,1	H2,0	H1,2	H1,1	H1,0	L0,2	L2,2	L1,2	-	-
14	x4,4	x4,2	x4,4	x4,3	L2,2	L2,1	L2,0	L3,0	-	-	H2,0	H0,2	H0,1	-	H2,2	H2,1	H3,0	H1,2	H1,1	H0,0	H2,0	H1,0	LH0,0	LL0,0
15	x4,6	x4,4	x4,6	x4,5	L2,2	L2,1	L2,0	L3,1	L3,0	-	H2,1	H2,0	H0,2	-	-	H2,2	H3,1	H3,0	H1,2	H0,1	H2,1	H1,1	LH0,1	LL0,1
16	x5,2	x5,0	x5,2	x5,1	L2,2	L2,1	L2,0	L3,2	L3,1	L3,0	H2,2	H2,1	H2,0	-	-	-	H3,2	H3,1	H3,0	H0,2	H2,2	H1,2	LH0,2	LL0,2
17	x5,4	x5,2	x5,4	x5,3	L4,0	L2,2	L2,1	-	L3,2	L3,1	H2,2	H2,1	H2,0	H4,0	-	-	H3,2	H3,1	H3,0	L2,0	L4,0	L3,0	HH0,0	HL0,0
18	x5,6	x5,4	x5,6	x5,5	L4,1	L4,0	L2,2	-	-	L3,2	H2,2	H2,1	H2,0	H4,1	H4,0	-	H3,2	H3,1	H3,0	L2,1	L4,1	L3,1	HH0,1	HL0,1
19	x6,2	x6,0	x6,2	x6,1	L4,2	L4,1	L4,0	-	-	-	H2,2	H2,1	H2,0	H4,2	H4,1	H4,0	H3,2	H3,1	H3,0	L2,2	L4,2	L3,2	HH0,1	HL0,1
20	x6,4	x6,2	x6,4	x6,3	L4,2	L4,1	L4,0	L5,0	-	-	H4,0	H2,2	H2,1	-	H4,2	H4,1	H5,0	H3,2	H3,1	H2,0	H4,0	H3,0	LH1,0	LL1,0
21	x6,6	x6,4	x6,6	x6,5	L4,2	L4,1	L4,0	L5,1	L5,0	-	H4,1	H4,0	H2,2	-	-	H4,2	H5,1	H5,0	H3,2	H2,1	H4,1	H3,1	LH1,1	LL1,1

B.2 Dataflow tables of chapter 4

Table B.8 Dataflow for 2-parallel architecture

CK	RP	RP1 & RP2			Rth	Rtl	CP1 input latches			CP2 input latches			CP1 & CP2 OUTPUTS			
		Rt0	Rt2	Rt1			Rt0	Rt2	Rt1	Rt0	Rt2	Rt1	Rt0	Rt1	Rt0	Rt1
1	1	x0,0	x0,2	x0,1												
2	2	x1,0	x1,2	x1,1												
3	1	x2,0	x2,2	x2,1												
4	2	x3,0	x3,2	x3,1												
5	1	x4,0	x4,2	x4,1												
6	2	x5,0	x5,2	x5,1												
7	1	x6,0	x6,2	x6,1	H0,0	L0,0										
8	2	x7,0	x7,2	x7,1	H1,0	L1,0										
9	1	x8,0	x8,2	x8,1	H2,0	L2,0	H0,0	H2,0	H1,0	L0,0	L2,0	L1,0				
10	2	x9,0	x9,2	x9,1	H3,0	L3,0	---------------------			-----------------------						
11	1	x10,0	x10,2	x10,1	H4,0	L4,0	H2,0	H4,0	H3,0	L2,0	L4,0	L3,0				
12	2	x11,0	x11,2	x11,1	H5,0	L5,0	--------------------			-----------------------						
13	1	x12,0	x12,2	x12,1	H6,0	L6,0	H4,0	H6,0	H5,0	L4,0	L6,0	L5,0				
14	2	x13,0	x13,2	x13,1	H7,0	L7,0	--------------------			-----------------------						
15	1	x14,0	x14,2	x14,1	H8,0	L8,0	H6,0	H8,0	H7,0	L6,0	L8,0	L7,0	HH0,0	HL0,0	LH0,0	LL0,0
16	2	x15,0	x15,2	x15,1	H9,0	L9,0	---------------------			---------------------			----------------------------------			
17	1	x16,0	x16,2	x16,1	H10,0	L10,0	H8,0	H10,0	H9,0	L8,0	L10,0	L9,0	HH1,0	HL1,0	LH1,0	LL1,0
18	2	x17,0	x17,2	x17,1	H11,0	L11,0	---------------------			---------------------			----------------------------------			
19	1	x18,0	x18,2	x18,1	H12,0	L12,0	H10,0	H12,0	H11,0	L10,0	L12,0	L11,0	HH2,0	HL2,0	LH2,0	LL2,0

Table B.9 dataflow of the 3-parallel architecture

Ck	RP	RP's input latches			RP's output latches			(CP1 & CP3) / CP2 input latches					
f_3		Rt0	Rt2	Rt1	Rth	Rtl	Rtl3b	Rt0	Rt2	Rt1	Rt0	Rt2	Rt1
1	1	x 0,0	x 0,2	x 0,1									
2	2	x 1,0	x1,2	x 1,1									
3	3	x 2,0	x 2,2	x 2,1									
4	1	x 3,0	x 3,2	x 3,1									
5	2	x 4,0	x 4,2	x 4,1									
6	3	x 5,0	x 5,2	x 5,1									
7	1	x 6,0	x 6,2	x 6,1									
8	2	x 7,0	x 7,2	x 7,1									
9	3	x 8,0	x 8,2	x 8,1									
10	1	x 9,0	x 9,2	x 9,1	H0,0	L0,0							
11	2	x 10,0	x 10,2	x 10,1	H1,0	L1,0							
12	3	x 11,0	x 11,2	x 11,1	H2,0	L2,0							
13	1	x 12,0	x 12,2	x 12,1	H3,0	L3,0		H0,0	H2,0	H1,0	L0,0	L2,0	L1,0
14	2	x 13,0	x 13,2	x 13,1	H4,0	L4,0		----------------------			H2,0	H4,0	H3,0
15	3	x 14,0	x 14,2	x 14,1	H5,0	L2,0	L5,0	--					
16	1	x 15,0	x 15,2	x 15,1	H6,0	L6,0	--------	H4,0	H60	H5,0	L2,0	L4,0	L3,0
17	2	x 16,0	x 16,2	x 16,1	H7,0	L7,0	--------	----------------------			L4,0	L6,0	L5,0
18	3	x 17,0	x 17,2	x 17,1	H8,0	L8,0	--------	--					
19	1	x 18,0	x 18,2	x 18,1	H9,0	L9,0	--------	H6,0	H8,0	H7,0	L6,0	L8,0	L7,0
20	2	x 19,0	x 19,2	x 19,1	H10,0	L10,0	-------	-----------------------			H8,0	H10,0	H9,0
21	3	x 20,0	x 20,2	x 20,1	H11,0	L8,0	L11,0	--					
22	1	x 21,0	x 21,2	x 21,1	H12,0	L12,0	-------	H10.0	H12,0	H11,0	L8,0	L10,0	L9,0
23	2	x 22,0	x 22,2	x 22,1	H13,0	L13,0	-------	------------------------			L10,0	L12,0	L11,0
24	3	x 23,0	x 23,2	x 23,1	H14,0	L14,0	-------	--					
25	1	x 24,0	x 24,2	x 24,1	H15,0	L15,0	-------	H12.0	H14,0	H13.0	L12,0	L14,0	L13,0
26	2	x 25,0	x 25,2	x 25,1	H16,0	L16,0	-------	-------------------------			H14,0	H16,0	H15,0
27	3	x 26,0	x 26,2	x 26,1	H17,0	L14,0	L17,0	--					
28	1	x 27,0	x 27,2	x 27,1	H18,0	L18,0	-------	H16.0	H18,0	H17,0	L14,0	L16,0	L15,0
29	2	x 28,0	x 28,2	x 28,1	H19,0	L19,0	-------	-------------------------			L16,0	L18,0	L17,0

CK	CP1 & CP3 output latches				CP2 output latches	
	Rth	Rtl	Rth	Rtl	Rth	Rtl
22	HH0,0	HL0,0	LH0,0	LL0,0		
23	----------------------------------				HH1,0	HL1,0
24	----------------------------------				------------------------	
25	HH2,0	HL2,0	LH1,0	LL1,0	------------------------	
26	----------------------------------				LH2,0	LL2,0
27	----------------------------------				------------------------	
28	HH3,0	HL3,0	LH3,0	LL3,0	------------------------	
29	----------------------------------				HH4,0	HL4,0

Table B.10 5/3 4-parallel architecture's dataflow

ck	RP	RP's input latches			RP's output latches		CP1 & CP3 input latches					
		Rt0	Rt2	Rt1	Rth	Rtl	Rt0	Rt2	Rt1	Rt0	Rt2	Rt1
1	1	x 0,0	x 0,2	x 0,1								
2	2	x 1,0	x1,2	x 1,1								
3	3	x 2,0	x 2,2	x 2,1								
4	4	x 3,0	x 3,2	x 3,1								
5	1	x 4,0	x 4,2	x 4,1								
6	2	x 5,0	x 5,2	x 5,1								
7	3	x 6,0	x 6,2	x 6,1								
8	4	x 7,0	x 7,2	x 7,1								
9	1	x 8,0	x 8,2	x 8,1								
10	2	x 9,0	x 9,2	x 9,1								
11	3	x 10,0	x 10,2	x 10,1								
12	4	x 11,0	x 11,2	x 11,1								
13	1	x 12,0	x 12,2	x 12,1	H0,0	L0,0						
14	2	x 13,0	x 13,2	x 13,1	H1,0	L1,0						
15	3	x 14,0	x 14,2	x 14,1	H2,0	L2,0	H0,0	H2,0	H1,0	L0,0	L2,0	L1,0
16	4	x 15,0	x 15,2	x 15,1	H3,0	L3,0	--					
17	1	x 16,0	x 16,2	x 16,1	H4,0	L4,0	--					
18	2	x 17,0	x 17,2	x 17,1	H5,0	L5,0	--					
19	3	x 18,0	x 18,2	x 18,1	H6,0	L6,0	H4,0	H6,0	H5,0	L4,0	L6,0	L5,0
20	4	x 19,0	x 19,2	x 19,1	H7,0	L7,0	--					
21	1	x 20,0	x 20,2	x 20,1	H8,0	L8,0	--					
22	2	x 21,0	x 21,2	x 21,1	H9,0	L9,0	--					
23	3	x 22,0	x 22,2	x 22,1	H10,0	L10,0	H8,0	H10,0	H9,0	L8,0	L10,0	L9,0
24	4	x 23,0	x 23,2	x 23,1	H11,0	L11,0	--					
25	1	x 24,0	x 24,2	x 24,1	H12,0	L12,0	--					
26	2	x 25,0	x 25,2	x 25,1	H13,0	L13,0	--					
27	3	x 26,0	x 26,2	x 26,1	H14,0	L14,0	H12,0	H14,0	H13,0	L12,0	L14,0	L13,0
28	4	x 27,0	x 27,2	x 27,1	H15,0	L15,0	--					
29	1	x 28,0	x 28,2	x 28,1	H16,0	L16,0	--					

CK	CP2 &CP4 input latches						CP1 & CP3 output latches				CP2 & CP4 output latches			
	Rt0	Rt2	Rt1	Rt0	Rt2	Rt1	Rth1	Rtl1	Rth3	Rtl3	Rth2	Rtl2	Rth4	Rtl4
17	H2,0	H4,0	H3,0	L2,0	L4,0	L3,0								
18	--													
19	--													
20	--													
21	H6,0	H8,0	H7,0	L6,0	L8,0	L7,0								
22	--													
23	--													
24	--													
25	H10,0	H12,0	H11,0	L10,0	L12,0	L11,0								
26	--													
27	--						HH0,0	HL0,0	LH0,0	LL0,0	----------------------------------			
28	--						----------------------------------				----------------------------------			
29	H14,0	H16,0	H15,0	L14,0	L16,0	L15,0	----------------------------------				HH1,0	HL1,0	LH1,0	LL1,0

Table B.11 4-parallel's TLBs read and write dataflow for case 2

		RP1							
		Stage 2							Stage 3
	Ck f_4	Rt0 Rt1	Sa12	1a	1b	BIR1	TLB1	BOR1	Rt2 Rt0 Rt1
Run1	5	x0,0 H0,0	1	0	0	-------			---- ----- -----
	7	x0,0 H0,0	1	0	0	-------			------ ----- ------
	9	x4,0 H4,0	1	0	0	H2,0			------ x0,0 H0,0
	11	x4,0 H4,0	1	0	0	H2,0			------ x0,0 H0,0
	13	x8,0 H8,0	1	1	0	H6,0	H2,0		------ x4,0 H4,0
	15	x8,0 H8,0	1	1	0	H6,0	H2,0		------ x4,0 H4,0
	17	x12,0 H12,0	1	2	0	H10,0	H2,0 H6,0		------ x8,0 H8,0
	19	x12,0 H12,0	1	2	0	H10,0	H2,0 H6,0		------ x8,0 H8,0
	21	x16,0 H16,0	1	3	0	H14,0	H2,0 H6,0 H10,0		------ x12,0 H12,0
	23	x16,0 H16,0	1	3	0	H14,0	H2,0 H6,0 H10,0		------ x12,0 H12,0
Run2	25	x2,2 H2,1	1	0	0	H0,1	H2,0 H6,0 H10,0 H14,0		------ x16,0 H16,0
	27	x2,2 H2,1	1	0	0	H0,1	H2,0 H6,0 H10,0 H14,0	H2,0	------ x16,0 H16,0
	29	x6,2 H6,1	1	1	0	H4,1	H0,1 H6,0 H10,0 H14,0	H2,0	H2,2 x2,2 H2,1
	31	x6,2 H6,1	1	1	0	H4,1	H0,1 H6,0 H10,0 H14,0	H6,0	H2,2 x2,2 H2,1
	33	x10,2 H10,1	1	2	0	H8,1	H0,1 H4,1 H10,0 H14,0	H6,0	H6,0 x6,2 H6,1
	35	x10,2 H10,1	1	2	0	H8,1	H0,1 H4,1 H10,0 H14,0	H10,0	H6,0 x6,2 H6,1
	37	x14,2 H14,1	1	3	0	H12,1	H0,1 H4,1 H8,0 H14,0	H10,0	H10,0 x10,2 H10,1
	39	x14,2 H14,1	1	3	0	H12,1	H0,1 H4,1 H8,1 H14,0	H14,0	H10,0 x10,2 H10,1
Run3	41	x0,4 H0,2	0	4	0	H16,1	H0,1 H4,1 H8,1 H12,1	H14,0	H14,0 x14,2 H14,1
	43	x0,4 H0,2	0	4	0	H16,1	H0,1 H4,1 H8,1 H12,1	H0,1	H14,0 x14,2 H14,1
	45	x4,4 H4,2	0	0	1	H2,2	----- H4,1 H8,1 H12,1 H16,1	H0,1	H0,1 x0,4 H0,2
	47	x4,4 H4,2	0	0	1	H2,2	----- H4,1 H8,1 H12,1 H16,1	H4,1	H0,1 x0,4 H0,2
	49	x8,4 H8,2	0	1	2	H6,2	H2,2 ----- H8,1 H12,1 H16,1	H4,1	H4,1 x4,4 H4,2

		RP3							
		Stage 2							Stage 3
	Ck f_4	Rt0 Rt1	Sa34	3a	3b	BIR3	TLB3	BOR3	Rt2 Rt0 Rt1
Run1	7	x2,0 H2,0	1	0	0	H0,0			------ ------ ------
	9	x2,0 H2,0	1	0	0	H0,0			------ ------ ------
	11	x6,0 H6,0	1	1	0	H4,0	H0,0		----- x2,0 H2,0
	13	x6,0 H6,0	1	1	0	H4,0	H0,0		----- x2,0 H2,0
	15	x10,0 H10,0	1	2	0	H8,0	H0,0 H4,0		----- x6,0 H6,0
	17	x10,0 H10,0	1	2	0	H8,0	H0,0 H4,0		----- x6,0 H6,0
	19	x14,0 H14,0	1	3	0	H12,0	H0,0 H4,0 H8,0		----- x10,0 H10,0
	21	x14,0 H14,0	1	3	0	H12,0	H0,0 H4,0 H8,0		----- x10,0 H10,0
Run2	23	x0,2 H0,1	0	4	0	H16,0	H0,0 H4,0 H8,0 H12,0		----- x14,0 H14,0
	25	x0,2 H0,1	0	4	0	H16,0	H0,0 H4,0 H8,0 H12,0	H0,0	----- x14,0 H14,0
	27	x4,2 H4,1	0	0	1	H2,1	----- H4,0 H8,0 H12,0 H16,0	H0,0	H0,0 x0,2 H0,1
	29	x4,2 H4,1	0	0	1	H2,1	----- H4,0 H8,0 H12,0 H16,0	H4,0	H0,0 x0,2 H0,1
	31	x8,2 H8,1	0	1	2	H6,1	H2,1 ----- H8,0 H12,0 H16,0	H4,0	H4,0 x4,2 H4,1
	33	x8,2 H8,1	0	1	2	H6,1	H2,1 ----- H8,0 H12,0 H16,0	H8,0	H4,0 x4,2 H4,1
	35	x12,2 H12,1	0	2	3	H10,1	H2,1 H6,1 ------ H12,0 H16,0	H8,0	H8,0 x8,2 H8,1
	37	x12,2 H12,1	0	2	3	H10,1	H2,1 H6,1 ------ H12,0 H16,0	H12,0	H8,0 x8,2 H8,1
	39	x16,2 H16,1	0	3	4	H14,1	H2,1 H6,1 H10,1 ------ H16,0	H12,0	H12,0 x12,2 H12,1
	41	x16,2 H16,1	0	3	4	H14,1	H2,1 H6,1 H10,1 ------ H16,0	H16,0	H12,0 x12,2 H12,1
Run3	43	x2,4 H2,2	1	0	0	H0,2	H2,1 H6,1 H10,1 H14,1 ------	H16,0	H16,0 x16,2 H16,1
	45	x2,4 H2,2	1	0	0	H0,2	H2,1 H6,1 H10,1 H14,1 ------	H2,1	H16,0 x16,2 H16,1
	47	x6,4 H6,2	1	1	0	H4,2	H0,2 H6,1 H10,1 H14,1 ------	H2,1	H2,1 x2,4 H2,2
	49	x6,4 H6,2	1	1	0	H4,2	H0,2 H6,1 H10,1 H14,1 ------	H2,1	H2,1 x2,4 H2,2

1a : TLBAR1a, 1b : TLBAR1b, 3a : TLBAR3a, 3b : TLBAR3b

Table B.12 Dataflow for 2-parallel intermediate architecture (k=3)

Ck	RP	Rd0	RP's input latches			RdH	SRH0			SRH1			RdL	SRL0			SRL1		
			Rt0	Rt2	Rt1		R2	R1	R0	R2	R1	R0		R2	R1	R0	R2	R1	R0
1	1	x 0,2	x 0,0	x 0,2	x 0,1														
2	2	x 0,4	x 0,2	x 0,4	x 0,3														
3	1	x 0,6	x 0,4	x 0,6	x 0,5														
4	2	x 1,2	x 1,0	x 1,2	x 1,1														
5	1	x 1,4	x 1,2	x 1,4	x 1,3														
6	2	x 1,6	x 1,4	x 1,6	x 1,5														
7	1	x 2,2	x 2,0	x 2,2	x 2,1		H0,0	-----	-----					L0,0	-----	-----			
8	2	x 2,4	x 2,2	x 2,4	x 2,3		H0,1	H0,0	-----					L0,1	L0,0	-----			
9	1	x 2,6	x 2,4	x 2,6	x 2,5		H0,2	H0,1	H0,0					L0,2	L0,1	L0,0			
10	2	x 3,2	x 3,0	x 3,2	x 3,1					H1,0	-----	-----					L1,0	-----	-----
11	1	x 3,4	x 3,2	x 3,4	x 3,3					H1,1	H1,0	-----					L1,1	L1,0	-----
12	2	x 3,6	x 3,4	x 3,6	x 3,5		H0,2	H0,1	H0,0	H1,2	H1,1	H1,0		L0,2	L0,1	L0,0	L1,2	L1,1	L1,0
13	1	x 4,2	x 4,0	x 4,2	x 4,1		H2,0	H0,2	H0,1	-----	H1,2	H1,1		L2,0	L0,2	L0,1	------	L1,2	L1,1
14	2	x 4,4	x 4,2	x 4,4	x 4,3	H2,1	H2,0	H0,2	H0,1	-----	H1,2	H1,1	L2,1	L2,0	L0,2	L0,1	------	L1,2	L1,1
15	1	x 4,6	x 4,4	x 4,6	x 4,5	H2,2	H2,1	H2,0	H 0,2	-----	H1,2	H1,1	L2,2	L2,1	L2,0	L0,2	------	L1,2	L1,1
16	2	x 5,2	x 5,0	x 5,2	x 5,1	H2,2	H2,1	H2,0	H 0,2	H3,0	-----	H1,2	L2,2	L2,1	L2,0	L0,2	L3,0	------	L1,2
17	1	x 5,4	x 5,2	x 5,4	x 5,3	------	H2,2	H2,1	H 2,0	H3,1	H3,0	------	------	L2,2	L2,1	L2,0	L3,1	L3,0	-----
18	2	x 5,6	x 5,4	x 5,6	x 5,5	------	H2,2	H2,1	H 2,0	H3,2	H3,1	H3,0	------	L2,2	L2,1	L2,0	L3,2	L3,1	L3,0
19	1	x 6,2	x 6,0	x 6,2	x 6,1	------	H4,0	H2,2	H2,1	------	H3,2	H3,1	------	L4,0	L2,2	L2,1	------	L3,2	L3,1
20	2	x 6,4	x 6,2	x 6,4	x 6,3	H4,1	H4,0	H2,2	H2,1	------	H3,2	H3,1	L4,1	L4,0	L2,2	L2,1	------	L3,2	L3,1
21	1	x 6,6	x 6,4	x 6,6	x 6,5	H4,2	H4,1	H4,0	H2,2	------	H3,2	H3,1	L4,2	L4,1	L4,0	L2,2	------	L3,2	L3,1
22	2	x 7,2	x 7,0	x 7,2	x 7,1	H4,2	H4,1	H4,0	H2,2	H5,0	------	H3,2	L4,2	L4,1	L4,0	L2,2	L5,0	-----	L3,2
23	1	x 7,4	x 7,2	x 7,4	x 7,3	------	H4,2	H4,1	H4,0	H5,1	H5,0	------	------	L4,2	L4,1	L4,0	L5,1	L5,0	-----
24	2	x 7,6	x 7,4	x 7,6	x 7,5	-----	H4,2	H4,1	H4,0	H5,2	H5,1	H5,0	------	L4,2	L4,1	L4,0	L5,2	L5,1	L5,0
25	1	x 8,2	x 8,0	x 8,2	x 8,1	------	H6,0	H4,2	H4,1	------	H5,2	H5,1	------	L6,0	L4,2	L4,1	-----	L5,2	L5,1

ck	RP	CP1 &CP2 input latches						CP1 & CP2 output latches			
		Rt0	Rt2	Rt1	Rt0	Rt2	Rt1	Rt1	Rt0	Rt1	Rt0
13	1	H0,0	H2,0	H1,0	L0,0	L2,0	L1,0				
14	2	--									
15	1	H0,1	H2,1	H1,1	L0,1	L2,1	L1,1				
16	2	--									
17	1	H0,2	H2,2	H1,2	L0,2	L2,2	L1,2				
18	2	--									
19	1	H2,0	H4,0	H3,0	L2,0	L4,0	L3,0	HH0,0	HL0,0	LH0,0	LL0,0
20	2	--						----------------------------------			
21	1	H2,1	H4,1	H3,1	L2,1	L4,1	L3,1	HH0,1	HL0,1	LH0,1	LL0,1
22	2	--						----------------------------------			
23	1	H2,2	H4,2	H3,2	L2,2	L4,2	L3,2	HH0,2	HL0,2	LH0,2	LL0,2
24	2	--						----------------------------------			
25	1	H4,0	H6,0	H5,0	L4,0	L6,0	L5,0	HH1,0	HL1,0	LH1,0	LL1,0

Table B.13 Dataflow of the last run for cases 4 and 3 when N is even

Ck	RP	Rd0	RP's input latches	RdH	SRH0	SRH1	RdL	SRL0	SRL1
			Rt0 Rt2 Rt1		R2 R1 R0	R2 R1 R0		R2 R1 R0	R2 R1 R0
25	1	------	--------------------	------	H6,0 H4,2 H4,1	------ H5,2 H5,1	-----	L6,0 L4,2 L4,1	----- L5,2 L5,1
26	2	x 0,8	X 0,6 x 0,8 x 0,7	H6,1	H6,0 H4,2 H4,1	------ H5,2 H5,1	L6,1	L6,0 L4,2 L4,1	----- L5,2 L5,1
27	1	------	X 0,8 x 0,8 x 0,9	H6,2	H6,1 H6,0 H4,2	------ H5,2 H5,1	L6,2	L6,1 L6,0 L4,2	----- L5,2 L5,1
28	2	x 1,8	x 1,6 x 1,8 x 1,7	H6,2	H6,1 H6,0 H4,2	H7,0 ------ H5,2	L6,2	L6,1 L6,0 L4,2	L7,0 ------ L5,2
29	1	------	x 1,8 x 1,8 x 1,9	-----	H6,2 H6,1 H6,0	H7,1 H7,0 ------	------	L6,2 L6,1 L6,0	L7,1 L7,0 ------
30	2	x 2,8	X2,6 x 2,8 x 2,7	------	H6,2 H6,1 H6,0	H7,2 H7,1 H7,0	------	L6,2 L6,1 L6,0	L7,2 L7,1 L7,0
31	1	------	x 2,8 x 2,8 x 2,9	------	----- H6,2 H6,1	------ H7,2 H7,1	------	------ L6,2 L6,1	------ L7,2 L7,1
32	2	x 3,8	x 3,6 x 3,8 x 3,7	H0,3	----- H6,2 H6,1	------ H7,2 H7,1	L0,3	------ L6,2 L6,1	------ L7,2 L7,1
33	1	------	x 3,8 x 3,8 x 3,9	H0,4	H0,3 ----- H6,2	------ H7,2 H7,1	L0,4	L0,3 ------ L6,2	------ L7,2 L7,1
34	2	x 4,8	x 4,6 x 4,8 x 4,7	H0,4	H0,3 ----- H6,2	H1,3 ------ H7,2	L0,4	L0,3 ----- L6,2	L1,3 ------ L7,2
35	1	------	x 4,8 x 4,8 x 4,9	------	H0,4 H0,3 -----	H1,4 H1,3 ------	------	L0,4 L0,3 -----	L1,4 L1,3 -----
36	2	x 5,8	x 5,6 x 5,8 x 5,7	H2,3	----- H0,4 H0,3	------ H1,4 H1,3	L2,3	----- L0,4 L0,3	------ L1,4 L1,3
37	1	------	x 5,8 x 5,8 x 5,9	H2,4	H2,3 ----- H0,4	------ H1,4 H1,3	L2,4	L2,3 ----- L0,4	------ L1,4 L1,3
38	2	x 6,8	x 6,6 x 6,8 x 6,7	H2,4	H2,3 ----- H0,4	H3,3 ------ H1,4	L2,4	L2,3 ----- L0,4	L3,3 ------ L1,4
39	1	------	x 6,8 x 6,8 x 6,9	------	H2,4 H2,3 -----	H3,4 H3,3 ------	------	L2,4 L2,3 -----	L3,4 L3,3 -----
40	2	x 7,8	x 7,6 x 7,8 x 7,7	H4,3	------ H2,4 H2,3	----- H3,4 H3,3	L4,3	------ L2,4 L2,3	----- L3,4 L3,3
41	1	------	x 7,8 x 7,8 x 7,9	H4,4	H4,3 ------ H2,4	----- H3,4 H3,3	L4,4	L4,3 ------ L2,4	----- L3,4 L3,3
42	2	------	--------------------	H4,4	H4,3 ------ H2,4	H5,3 ----- H3,4	L4,4	L4,3 ------ L2,4	L5,3 ----- L3,4
43	1	------	--------------------	------	H4,4 H4,3 ------	H5,4 H5,3 -----	------	L4,4 L4,3 ------	L5,4 L5,3 -----
44	2	------	--------------------	H6,3	----- H4,4 H4,3	------ H5,4 H5,3	L6,3	----- L4,4 L4,3	------ L5,4 L5,3
45	1	------	--------------------	H6,4	H6,3 ----- H4,4	------ H5,4 H5,3	L6,4	L6,3 ----- L4,4	------ L5,4 L5,3
46	2	------	--------------------	H6,4	H6,3 ----- H4,4	H7,3 ------ H5,4	L6,4	L6,3 ----- L4,4	L7,3 ------ L5,4
47	1	------	--------------------	------	H6,4 H6,3 -----	H7,4 H7,3 ------	------	L6,4 L6,3 -----	L7,4 L7,3 -----
48	2	------	--------------------	------	----- H6,4 H6,3	----- H7,4 H7,3	------	----- L6,4 L6,3	----- L7,4 L7,3
49	1	------	--------------------	------	----- ------ H6,4	----- H7,4 H7,3	------	----- ------ L6,4	----- L7,4 L7,3
50	2	------	--------------------	------	----- ------ H6,4	----- ------ H7,4	------	----- ------ L6,4	----- ------ L7,4

Ck	RP	CP1 &CP2 input latches	CP1 & CP2 output latches
		Rt0 Rt2 Rt1 Rt0 Rt2 Rt1	Rt Rt Rt Rt
29	1	H4,2 H6,2 H5,2 L4,2 L6,2 L5,2	HH1,2 HL1,2 LH1,2 LL1,2
30	2	--	------------------------------------
31	1	H6,0 H6,0 H7,0 L6,0 L6,0 L7,0	HH2,0 HL2,0 LH2,0 LL2,0
32	2	--	------------------------------------
33	1	H6,1 H6,1 H7,1 L6,1 L6,1 L7,1	HH2,1 HL2,1 LH2,1 LL2,1
34	2	--	------------------------------------
35	1	H6,2 H6,2 H7,2 L6,2 L6,2 L7,2	HH2,2 HL2,2 LH2,2 LL2,2
36	2	--	------------------------------------
37	1	H0,3 H2,3 H1,3 L0,3 L2,3 L1,3	HH3,0 HL3,0 LH3,0 LL3,0
38	2	--	------------------------------------
39	1	H0,4 H2,4 H1,4 L0,4 L2,4 L1,4	HH3,1 HL3,1 LH3,1 LL3,1
40	2	--	------------------------------------
41	1	H2,3 H4,3 H3,3 L2,3 L4,3 L3,3	HH3,2 HL3,2 LH3,2 LL3,2
42	2	--	------------------------------------
43	1	H2,4 H4,4 H3,4 L2,4 L4,4 L3,4	HH0,3 HL0,3 LH0,3 LL0,3
44	2	--	------------------------------------
45	1	H4,3 H6,3 H5,3 L4,3 L6,3 L5,3	HH0,4 HL0,4 LH0,4 LL0,4
46	2	--	------------------------------------
47	1	H4,4 H6,4 H5,4 L4,4 L6,4 L5,4	HH1,3 HL1,3 LH1,3 LL1,3
48	2	--	------------------------------------
49	1	H6,3 H6,3 H7,3 L6,3 L6,3 L7,3	HH1,4 HL1,4 LH1,4 LL1,4
50	2	--	------------------------------------
51	1	H6,4 H6,4 H7,4 L6,4 L6,4 L7,4	HH2,3 HL2,3 LH2,3 LL2,3

Table B.14 Dataflow of the last run for cases 4 and 3 when N is odd

Ck	RP	Rd0	RP's input latches	RdH	SRH0	SRH1	RdL	SRL0	SRL1
			Rt0 Rt2 Rt1		R2 R1 R0	R2 R1 R0		R2 R1 R0	R2 R1 R0
22	2	-----	-------------------	H4,2	H4,1 H4,0 H2,2	H5,0 ------ H3,2	L4,2	L4,1 L4,0 L2,2	L5,0 ---- L3,2
23	1	-----	-------------------	-----	H4,2 H4,1 H4,0	H5,1 H5,0 -----	-----	L4,2 L4,1 L4,0	L5,1 L5,0 ----
24	2	-----	-------------------	-----	H4,2 H4,1 H4,0	H5,2 H5,1 H5,0	-----	L4,2 L4,1 L4,0	L5,2 L5,1 L5,0
25	1	-----	-------------------	-----	H6,0 H4,2 H4,1	----- H5,2 H5,1	-----	L6,0 L4,2 L4,1	----- L5,2 L5,1
26	2	x0,8	x 0,6 x 0,8 x0,7	H6,1	H6,0 H4,2 H4,1	----- H5,2 H5,1	L6,1	L6,0 L4,2 L4,1	----- L5,2 L5,1
27	1	-----	x 0,8 ------ -----	H6,2	H6,1 H6,0 H4,2	----- H5,2 H5,1	L6,2	L6,1 L6,0 L4,2	----- L5,2 L5,1
28	2	x1,8	x 1,6 x 1,8 x1,7	H6,2	H6,1 H6,0 H4,2	----- ------ H5,2	L6,2	L6,1 L6,0 L4,2	----- ----- L5,2
29	1	-----	x 1,8 ------ -----	------	H6,2 H6,1 H6,0	-------------------	-----	L6,2 L6,1 L6,0	-------------------
30	2	x2,8	X2,6 x 2,8 x2,7	------	H6,2 H6,1 H6,0	-------------------	-----	L6,2 L6,1 L6,0	-------------------
31	1	-----	x 2,8 ------ -----	------	------ H6,2 H6,1	-------------------	-----	------L6,2 L6,1	-------------------
32	2	x3,8	x 3,6 x 3,8 x3,7	H0,3	------ H6,2 H6,1	-------------------	L0,3	------L6,2 L6,1	-------------------
33	1	-----	x 3,8 ------ -----	------	H0,3 ------ H6,2	-------------------	L0,4	L0,3 ------L6,2	-------------------
34	2	x4,8	x 4,6 x 4,8 x4,7	------	H0,3 ------ H6,2	H1,3 ------ ------	L0,4	L0,3 ------L6,2	L1,3 ----- -----
35	1	-----	x 4,8 ------ -----	------	------ H0,3 -----	------ H1,3 ----	-----	L0,4 L0,3 ----	L1,4 L1,3 -----
36	2	x5,8	x 5,6 x 5,8 x5,7	H2,3	------ ----- H0,3	------ ----- H1,3	L2,3	----- L0,4 L0,3	----- L1,4 L1,3
37	1	-----	x 5,8 ------ -----	------	H2,3 ------ -----	------ ---- H1,3	L2,4	L2,3 ---- L0,4	----- L1,4 L1,3
38	2	x6,8	x 6,6 x 6,8 x6,7	------	H2,3 ------ -----	H3,3 ------ -----	L2,4	L2,3 ---- L0,4	L3,3 ----- L1,4
39	1	-----	x 6,8 ------ -----	------	------ H2,3 -----	------ H3,3 -----	-----	L2,4 L2,3 ----	L3,4 L3,3 -----
40	2	-----	-------------------	H4,3	------ ------ H2,3	------ ----- H3,3	L4,3	------L2,4 L2,3	----- L3,4 L3,3
41	1	-----	-------------------	------	H4,3 ------- ----	------ ----- H3,3	L4,4	L4,3 ---- L2,4	----- L3,4 L3,3
42	2	-----	-------------------	------	H4,3 ------- ----	H5,3 ------ -----	L4,4	L4,3 ---- L2,4	L5,3 ---- L3,4

Ck	RP	CP1 &CP2 input latches	CP1 & CP2 output latches
		Rt0 Rt2 Rt1 Rt0 Rt2 Rt1	Rt Rt Rt Rt
22	2	---	-------------------------------------
23	1	H2,2 H4,2 H3,2 L2,2 L4,2 L3,2	HH0,2 HL0,2 LH0,2 LL0,2
24	2	---	-------------------------------------
25	1	H4,0 H6,0 H5,0 L4,0 L6,0 L5,0	HH1,0 HL1,0 LH1,0 LL1,0
26	2	---	-------------------------------------
27	1	H4,1 H6,1 H5,1 L4,1 L6,1 L5,1	HH1,1 HL1,1 LH1,1 LL1,1
28	2	---	-------------------------------------
29	1	H4,2 H6,2 H5,2 L4,2 L6,2 L5,2	HH1,2 HL1,2 LH1,2 LL1,2
30	2	---	-------------------------------------
31	1	H6,0 ------ ------ L6,0 ------ ------	HH2,0 HL2,0 LH2,0 LL2,0
32	2	---	-------------------------------------
33	1	H6,1 ------ ------ L6,1 ----- ------	HH2,1 HL2,1 LH2,1 LL2,1
34	2	---	-------------------------------------
35	1	H6,2 ------ ------ L6,2 ----- ------	HH2,2 HL2,2 LH2,2 LL2,2
36	2	---	-------------------------------------
37	1	H0,3 H2,3 H1,3 L0,3 L2,3 L1,3	------- HL3,0 ------- LL3,0
38	2	---	-------------------------------------
39	1	---------------------- L0,4 L2,4 L1,4	------- HL3,1 ------- LL3,1
40	2	---	-------------------------------------
41	1	H2,3 H4,3 H3,3 L2,3 L4,3 L3,3	-------- HL3,2 -------- LL3,2
42	2	---	-------------------------------------

Table B.15 Dataflow of the last run for cases 2 and 1 when N is even

Ck	RP	Rd0	RP's input latches			RdH	SRH0			SRH1			RdL	SRL0			SRL1		
			Rt0	Rt2	Rt1		R2	R1	R0	R2	R1	R0		R2	R1	R0	R2	R1	R0
25	1	------	--------------------			------	H6,0	H4,2	H4,1	------	H5,2	H5,1	-----	L6,0	L4,2	L4,1	-----	L5,2	L5,1
26	2	------	--------------------			H6,1	H6,0	H4,2	H4,1	------	H5,2	H5,1	L6,1	L6,0	L4,2	L4,1	-----	L5,2	L5,1
27	1	------	x 0,6	x 0,6	x 0,7	H6,2	H6,1	H6,0	H4,2	------	H5,2	H5,1	L6,2	L6,1	L6,0	L4,2	-----	L5,2	L5,1
28	2	------	x 1,6	x 1,6	x 1,7	H6,2	H6,1	H6,0	H4,2	H7,0	------	H5,2	L6,2	L6,1	L6,0	L4,2	L7,0	------	L5,2
29	1	------	X2,6	x 2,6	x 2,7	-----	H6,2	H6,1	H6,0	H7,1	H7,0	------	------	L6,2	L6,1	L6,0	L7,1	L7,0	------
30	2	------	x 3,6	x 3,6	x 3,7	------	H6,2	H6,1	H6,0	H7,2	H7,1	H7,0	------	L6,2	L6,1	L6,0	L7,2	L7,1	L7,0
31	1	------	x 4,6	x 4,6	x 4,7	------	-----	H6,2	H6,1	------	H7,2	H7,1	------	------	L6,2	L6,1	------	L7,2	L7,1
32	2	------	x 5,6	x 5,6	x 5,7	------	-----	H6,2	H6,1	------	H7,2	H7,1	------	------	L6,2	L6,1	------	L7,2	L7,1
33	1	------	x 6,6	x 6,6	x 6,7	H0,3	-----	H6,2	H6,1	------	H7,2	H7,1	L0,3	------	L6,2	L6,1	------	L7,2	L7,1
34	2	------	x 7,6	x 7,6	x 7,7	------	H0,3	-----	H6,2	H1,3	------	H7,2	------	L0,3	-----	L6,2	L1,3	------	L7,2
35	1	------	--------------------			H2,3	------	H0,3	-----	-----	H1,3	------	L2,3	-----	L0,3	-----	------	L1,3	-----
36	2	------	--------------------			H2,3	H2,3	------	H0,3	H3,3	------	H1,3	L2,3	L2,3	-----	L0,3	L3,3	-----	L1,3
37	1	------	--------------------			H4,3	-----	H2,3	------	------	H3,3	------	L4,3	------	L2,3	-----	------	L3,3	-----
38	2	------	--------------------			H4,3	H4,3	-----	H2,3	H5,3	------	H3,3	L4,3	L4,3	-----	L2,3	L5,3	------	L3,3
39	1	------	--------------------			H6,3	------	H4,3	-----	------	H5,3	------	L6,3	------	L4,3	-----	------	L5,3	-----
40	2	------	--------------------			H6,3	H6,3	------	H4,3	H7,3	------	H5,3	L6,3	L6,3	------	L4,3	L7,3	------	L5,3
41	1	------	--------------------			------	------	H6,3	------	------	H7,3	------	------	------	L6,3	------	-----	L7,3	------
42	2	------	--------------------			------	------	------	H6,3	------	------	H7,3	------	------	------	L6,3	-----	------	L7,3
43	1	------	--------------------			------	--------------------			--------------------			------	--------------------			------------------		
44	2	------	--------------------			------	--------------------			--------------------			------	--------------------			------------------		
45	1	------	--------------------			------	--------------------			--------------------			------	--------------------			------------------		
46	2	------	--------------------			------	--------------------			--------------------			------	--------------------			------------------		
47	1	------	--------------------			------	--------------------			--------------------			------	--------------------			------------------		
48	2	------	--------------------			------	--------------------			--------------------			------	--------------------			------------------		
49	1	------	--------------------			------	--------------------			--------------------			------	--------------------			------------------		

ck	RP	CP1 &CP2 input latches						CP1 & CP2 output latches			
		Rt0	Rt2	Rt1	Rt0	Rt2	Rt1	Rt	Rt	Rt	Rt
28	2	--						----------------------------------			
29	1	H4,2	H6,2	H5,2	L4,2	L6,2	L5,2	HH1,2	HL1,2	LH1,2	LL1,2
30	2	--						----------------------------------			
31	1	H6,0	H6,0	H7,0	L6,0	L6,0	L7,0	HH2,0	HL2,0	LH2,0	LL2,0
32	2	--						----------------------------------			
33	1	H6,1	H6,1	H7,1	L6,1	L6,1	L7,1	HH2,1	HL2,1	LH2,1	LL2,1
34	2	--						----------------------------------			
35	1	H6,2	H6,2	H7,2	L6,2	L6,2	L7,2	HH2,2	HL2,2	LH2,2	LL2,2
36	2	--						----------------------------------			
37	1	H0,3	H2,3	H1,3	L0,3	L2,3	L1,3	HH3,0	HL3,0	LH3,0	LL3,0
38	2	--						----------------------------------			
39	1	H2,3	H4,3	H3,3	L2,3	L4,3	L3,3	HH3,1	HL3,1	LH3,1	LL3,1
40	2	--						----------------------------------			
41	1	H4,3	H6,3	H5,3	L4,3	L6,3	L5,3	HH3,2	HL3,2	LH3,2	LL3,2
42	2	--						----------------------------------			
43	1	H6,3	H6,3	H7,3	L6,3	L6,3	L7,3	HH0,3	HL0,3	LH0,3	LL0,3
44	2	--						----------------------------------			
45	1	--						HH1,3	HL1,3	LH1,3	LL1,3
46	2	--						----------------------------------			
47	1	--						HH2,3	HL2,3	LH2,3	LL2,3
48	2	--						----------------------------------			
49	1	--						HH3,3	HL3,3	LH3,3	LL3,3

Table B.16 Dataflow of the last run for cases 2 and 1 when N is odd

Ck	RP	Rd0	RP's input latches	RdH	SRH0	SRH1	RdL	SRL0	SRL1
			Rt0 Rt2 Rt1		R2 R1 R0	R2 R1 R0		R2 R1 R0	R2 R1 R0
22	2	------	---------------------	H4,2	H4,1 H4,0 H2,2	H5,0 ------ H3,2	L4,2	L4,1 L4,0 L2,2	L5,0 ----- L3,2
23	1	------	---------------------	------	H4,2 H4,1 H4,0	H5,1 H5,0 -------	------	L4,2 L4,1 L4,0	L5,1 L5,0 ----
24	2	------	---------------------	------	H4,2 H4,1 H4,0	H5,2 H5,1 H5,0	------	L4,2 L4,1 L4,0	L5,2 L5,1 L5,0
25	1	------	---------------------	------	H6,0 H4,2 H4,1	----- H5,2 H5,1	------	L6,0 L4,2 L4,1	----- L5,2 L5,1
26	2	------	---------------------	H6,1	H6,0 H4,2 H4,1	----- H5,2 H5,1	L6,1	L6,0 L4,2 L4,1	----- L5,2 L5,1
27	1	------	x 0,6 ------ -------	H6,2	H6,1 H6,0 H4,2	----- H5,2 H5,1	L6,2	L6,1 L6,0 L4,2	----- L5,2 L5,1
28	2	------	x 1,6 ------ -------	H6,2	H6,1 H6,0 H4,2	----- ------ H5,2	L6,2	L6,1 L6,0 L4,2	----- ----- L5,2
29	1	------	x 2,6 ------ -------	------	H6,2 H6,1 H6,0	--------------------	------	L6,2 L6,1 L6,0	------------------
30	2	------	x 3,6 ------ -------	------	H6,2 H6,1 H6,0	--------------------	------	L6,2 L6,1 L6,0	------------------
31	1	------	x 4,6 ------ -------	------	------ H6,2 H6,1	--------------------	------	------ L6,2 L6,1	------------------
32	2	------	x 5,6 ------ -------	------	------ H6,2 H6,1	--------------------	------	------ L6,2 L6,1	------------------
33	1	------	x 6,6 ------ -------	------	----- ------ H6,2	--------------------	L0,3	------ L6,2 L6,1	------------------
34	2	------	---------------------	------	------ ------ H6,2	--------------------	L0,3	L0,3 ------ L6,2	L1,3 ----- -----
35	1	------	---------------------	------	--------------------	--------------------	L2,3	----- L0,3 ------	------ L1,3 -----
36	2	------	---------------------	------	--------------------	--------------------	L2,3	L2,3 ------ L0,3	L3,3 ----- L1,3
37	1	------	---------------------	------	--------------------	--------------------	L4,3	------ L2,3 ------	------ L3,3 -----
38	2	------	---------------------	------	--------------------	--------------------	L4,3	L4,3 ------ L2,3	L5,3 ----- L3,3
39	1	------	---------------------	------	--------------------	--------------------	L6,3	------ L4,3 ------	------ L5,3 -----
40	2	------	---------------------	------	--------------------	--------------------	L6,3	L6,3 ------ L4,3	----- ------ L5,3
41	1	------	---------------------	------	--------------------	--------------------	------	------ L6,3 ------	------------------
42	2	------	---------------------	------	--------------------	--------------------	------	------ ------ L6,3	------------------
43	1	------	---------------------	------	--------------------	--------------------	------	--------------------	------------------

ck	RP	CP1 &CP2 input latches	CP1 & CP2 output latches
		Rt0 Rt2 Rt1 Rt0 Rt2 Rt1	Rt Rt Rt Rt
22	2	--	-----------------------------------
23	1	H2,2 H4,2 H3,2 L2,2 L4,2 L3,2	HH0,2 HL0,2 LH0,2 LL0,2
24	2	--	-----------------------------------
25	1	H4,0 H6,0 H5,0 L4,0 L6,0 L5,0	HH1,0 HL1,0 LH1,0 LL1,0
26	2	--	-----------------------------------
27	1	H4,1 H6,1 H5,1 L4,1 L6,1 L5,1	HH1,1 HL1,1 LH1,1 LL1,1
28	2	--	-----------------------------------
29	1	H4,2 H6,2 H5,2 L4,2 L6,2 L5,2	HH1,2 HL1,2 LH1,2 LL1,2
30	2	--	-----------------------------------
31	1	H6,0 ------ ------ L6,0 ------ ------	HH2,0 HL2,0 LH2,0 LL2,0
32	2	--	-----------------------------------
33	1	H6,1 ------ ------ L6,1 ----- ------	HH2,1 HL2,1 LH2,1 LL2,1
34	2	--	-----------------------------------
35	1	H6,2 ------ ------ L6,2 ----- ------	HH2,2 HL2,2 LH2,2 LL2,2
36	2	--	-----------------------------------
37	1	--------------------- L0,3 L2,3 L1,3	------- HL3,0 ------- LL3,0
38	2	--	-----------------------------------
39	1	--------------------- L2,3 L4,3 L3,3	------- HL3,1 ------- LL3,1
40	2	--	-----------------------------------
41	1	--------------------- L4,3 L6,3 L5,3	-------- HL3 2 -------- LL3,2
42	2	--	-----------------------------------
43	1	--------------------- L6,3 ------ ------	------- ------- LH0,3 LL0,3
44	2	--	-----------------------------------
45	1	--	------- ------- LH1,3 LL1,3
46	2	--	-----------------------------------
47	1	--	------- ------- LH2,3 LL2,3
48	2	--	-----------------------------------
49	1	--	------- ------- ------- LL3,3

Table B.17 Dataflow of the 3-parallel intermediate architecture

Ck	RP	Rd0	RP's input latches Rt0 Rt2 Rt1	RdH	SRH0 R2 R1 R0	SRH1 R2 R1 R0	RdL R1 R0	SRL0 R2 R1 R0	SRL1 R2 R1 R0
1	1	x0,2	x 0,0 x 0,2 x0,1						
2	2	x0,4	x 0,2 x 0,4 x0,3						
3	3	x0,6	x 0,4 x 0,6 x0,5						
4	1	x1,2	x 1,0 x 1,2 x1,1						
5	2	x1,4	x 1,2 x 1,4 x 1,3						
6	3	x1,6	x 1,4 x 1,6 x1,5						
7	1	x2,2	x 2,0 x 2,2 x2,1						
8	2	x2,4	x 2,2 x 2,4 x2,3						
9	3	x2,6	x 2,4 x 2,6 x2,5						
10	1	x3,2	x 3,0 x 3,2 x3,1		H0,0 ----- -----			L0,0 ----- -----	
11	2	x3,4	x 3,2 x 3,4 x3,3		H0,1 H0,0 -----			L0,1 L0,0 -----	
12	3	x3,6	x 3,4 x 3,6 x3,5		H0,2 H0,1 H0,0			L0,2 L0,1 L0,0	
13	1	x4,2	x 4,0 x 4,2 x4,1		H0,2 H0,1 H0,0	H1,0 ----- -----		L0,2 L0,1 L0,0	L1,0 ----- -----
14	2	x4,4	x 4,2 x 4,4 x4,3		H0,2 H0,1 H0,0	H1,1 H1,0 -----		L0,2 L0,1 L0,0	L1,1 L1,0 -----
15	3	x4,6	x 4,4 x 4,6 x4,5		H0,2 H0,1 H0,0	H1,2 H1,1 H1,0		L0,2 L0,1 L0,0	L1,2 L1,1 L1,0
16	1	x5,2	x 5,0 x 5,2 x5,1		H2,0 H0,2 H0,1	----- H1,2 H1,1		L2,0 L0,2 L0,1	------ L1,2 L1,1
17	2	x5,4	x 5,2 x 5,4 x5,3	-----	H2,1 H2,0 H 0,2	----- ------ H1,2	L2,1 ----	L2,0 L0,2 L0,1	------ L1,2 L1,1
18	3	x5,6	x 5,4 x 5,6 x5,5	H2,2	H2,1 H2,0 H 0,2	----- ------ H1,2	L2,2 L2,1	L2,0 L0,2 L0,1	------ L1,2 L1,1
19	1	x6,2	x 6,0 x 6,2 x6,1	-----	H2,2 H2,1H 2,0	H3,0 ----- ------	----- L2,2	L2,1 L2,0 L0,2	L3,0 ------ L1,2
20	2	x6,4	x 6,2 x 6,4 x6,3	-----	H2,2 H2,1H 2,0	H3,1 H3,0 ------	------ ------	L2,2 L2,1 L2,0	L3,1 L3,0 -----
21	3	x6,6	x 6,4 x 6,6 x6,5	-----	H2,2 H2,1H 2,0	H3,2 H3,1 H3,0	------ ------	L2,2 L2,1 L2,0	L3,2 L3,1 L3,0
22	1	x7,2	x 7,0 x 7,2 x7,1	-----	H4,0 H2,2 H2,1	------ H3,2 H3,1	------ ------	L4,0 L2,2 L2,1	------ L3,2 L3,1
23	2	x7,4	x 7,2 x 7,4 x7,3	-----	H4,1 H4,0 H2,2	------ ----- H3,2	L4,1 ------	L4,0 L2,2 L2,1	------ L3,2 L3,1
24	3	x7,6	x 7,4 x 7,6 x7,5	H4,2	H4,1 H4,0 H2,2	------ ------ H3,2	L4,2 L4,1	L4,0 L2,2 L2,1	------ L3,2 L3,1
25	1	x8,2	x 8,0 x 8,2 x8,1	-----	H4,2 H4,1 H4,0	H5,0 ------ ------	----- L4,2	L4,1 L4,0 L2,2	L5,0 ----- L3,2
26	2	x8,4	x 8,2 x 8,4 x8,3	-----	H4,2 H4,1 H4,0	H5,1 H5,0 ------	------ ------	L4,2 L4,1 L4,0	L5,1 L5,0 -----
27	3	x8,6	x 8,4 x 8,6 x8,5	-----	H4,2 H4,1 H4,0	H5,2 H5,1 H5,0	------ ------	L4,2 L4,1 L4,0	L5,2 L5,1 L5,0
28	1	x9,2	x 9,0 x 9,2 x9,1	-----	H6,0 H4,2 H4,1	------ H5,2 H5,1	------ ------	L6,0 L4,2 L4,1	----- L5,2 L5,1
29	2	x9,4	x 9,2 x 9,4 x9,3	-----	H6,1 H6,0 H4,2	------ ----- H5,2	L6,1 ------	L6,0 L4,2 L4,1	----- L5,2 L5,1
30	3	x9,6	x 9,4 x 9,6 x9,5	H6,2	H6,1 H6,0 H4,2	------ ------ H5,2	L6,2 L6,1	L6,0 L4,2 L4,1	----- L5,2 L5,1

ck	RP	CP1 & CP3 input latches Rt0 Rt2 Rt1 Rt0 Rt2 Rt1	CP2 input latches Rt0 Rt2 Rt1	CP1 & CP3 output latches Rth Rtl Rth Rtl	CP2 output latches Rth Rtl
16	1	H0,0 H2,0 H1,0 L0,0 L2,0 L1,0			
17	2	--	H0,1 H2,1 H1,1		
18	3	--	--------------------		
19	1	H0,2 H2,2 H1,2 L0,1 L2,1 L1,1	--------------------		
20	2	--	L0,2 L2,2 L1,2		
21	3	--	--------------------		
22	1	H2,0 H4,0 H3,0 L2,0 L4,0 L3,0	--------------------		
23	2	--	H2,1 H4,1 H3,1		
24	3	--	--------------------		
25	1	H2,2 H4,2 H3,2 L2,1 L4,1 L3,1	--------------------	HH0,0 HL0,0 LH0,0 LL0,0	
26	2	--	L2,2 L4,2 L3,2	------------------------------------	HH0,1 HL0,1
27	3	--	--------------------	------------------------------------	--------------------
28	1	H4,0 H6,0 H5,0 L4,0 L6,0 L5,0	--------------------	HH0,2 HL0,2 LH0,1 LL0,1	--------------------
29	2	--		------------------------------------	LH0,2 LL0,2
30	3	--	--------------------	------------------------------------	--------------------

B.3 Dataflow tables of chapter 5

In Table B.19 (a), the pipeline stages 4, 7, and 10 of Figure 6.5.5 have not included, since they are in the first run, which ends at cycle 20, only pass coefficients of the previous stage to the next, whereas in the second run, which begins at cycle 25, and in all subsequent runs, stages 4 and 10 are bypassed, as shown in Table B.19 (a). For instance, Rt0 and Rt1 of stage 2 are shown holding coefficients YL'2,0 and YL'2,1 in cycle 26, during which coefficient YL"2,0 is computed. Then in cycle 27 YL"2,0 is loaded into Rt0 of stage 3 while YL'2,1 is loaded into Rt1 of stage 5 through the multiplexer labeled *mux* bypassing stages 3 and 4. In cycle 28, YL'2,1 in Rt1 of stage 5 is loaded into Rt1 of stage 6, while YL"2,0 in Rt0 of stage 3 is transferred to Rt0 of stage 6 bypassing stages 4 and 5, where the two coefficients proceed together until stage 8.

Note that the first indexes in YL, YH, XL, and XH in Tables B.18 and B.19 (a) refer to column numbers in Figures 6.3.2 (A) and (B). While the second indexes refer to input numbers in each column in accordance with the convention followed in the DDGs. On the other hand, the first indexes of Y and X in Tables B.18 and B.19 (b) refer to input numbers in each row in accordance with the convention followed in the DDGs which is also indicated in the processors datapath architecture.

Table B.18 Dataflow of the 5/3 architecture

Ck	1	2	3	4	CP output latches	1	2	3	4	RP output latches
f	CP input latches					RP input latches				
	Rd0 Rt0 Rt1	Rt0 Rt1	Rt0 Rt1	Rt0 Rt1	Rtl0 Rtl1 Rth	Rt0 Rt1 TLB1	Rt0 Rt1 Rt2	Rt0 Rt1 TLB2	Rt0 Rt1 Rt2	Rt0 Rt1
1	LL0,0 ------- -------									
2	------- LL0,0 LH0,0									
3	HL0,0 LL0,0 LH0,0									
4	------- HL0,0 HH0,0	XL0(0) YL0(1)								
5	LL1,0 HL0,0 HH0,0	XL0(0) YL0(1)								
6	------- LL1,0 LH1,0	XH0(0) YH0(1)	XL0(0) YL0(1)							
7	HL1,0 LL1,0 LH1,0	XH0(0) YH0(1)	XL0(0) YL0(1)							
8	------- HL1,0 HH1,0	XL0(2) YL0(3)	XH0(0) YH0(1)	XL0(0) YL0(1)						
9	LL2,0 HL1,0 HH1,0	XL0(2) YL0(3)	XH0(0) YH0(1)	XL0(0) YL0(1)						
10	------- LL2,0 LH2,0	XH0(2) YH0(3)	XL0(2) YL0(3)	XH0(0) YH0(1)	L0,0 L1,0 -----					
11	HL2,0 LL2,0 LH2,0	XH0(2) YH0(3)	XL0(2) YL0(3)	XH0(0 YH0(1)	L0,0 L1,0 -----					
12	------ HL2,0 HH2,0	XL0(4) YL0(5)	XH0(2) YH0(3)	XL0(2) YL0(3)	----- L1,0 H1,0	L0,0 H0,0 ------				
13	LL3,0 HL2,0 HH2,0	XL0(4) YL0(5)	XH0(2) YH0(3)	XL0(2) YL0(3)	----- L1,0 H1,0	L0,0 H0,0 ------				
14	------- LL3,0 -------	XH0(4) YH0(5)	XL0(4) YL0(5)	XH0(2) YH0(3)	L2,0 L3,0 ------	L1,0 H1,0 Y0(1)	Y0(0) Y0(1) ----			
15	HL3,0 LL3,0 -------	XH0(4) YH0(5)	XL0(4) YL0(5)	XH0(2) YH0(3)	L2,0 L3,0 ------	L1,0 H1,0	Y0(0) Y0(1) ----			
16	------- HL3,0 ------	XL0(6) --------	XH0(4) YH0(5)	XL0(4) YL0(5)	----- L3,0 H3,0	L2,0 H2,0 Y1(1)	Y1(0) Y1(1) ----	X0(0) -----		
17	LL0,1 HL3,0 -------	XL0(6) --------	XH0(4) YH0(5)	XL0(4) YL0(5)	----- L3,0 H3,0	L2,0 H2,0	Y1(0) Y1(1) ----	X0(0) -----		
18	------- LL0,1 LH0,1	XH0(6) -------	XL0(6) --------	XH0(4) YH0(5)	L4,0 L5,0 ------	L3,0 H3,0 Y2(1)	Y2(0) Y2(1) ----	X1(0) ----- X0(0)	X0(0) ---- ----	
19	HL0,1 LL0,1 LH0,1	XH0(6) -------	XL0(6) --------	XH0(4) YH0(5)	L4,0 L5,0 ------	L3,0 H3,0	Y2(0) Y2(1) ----	X1(0) -----	X0(0) ---- ----	
20	------- HL0,1 HH0,1	XL1(0) YL1(1)	XH0(6) -------	XL0(6) --------	----- L5,0 H5,0	L4,0 H4,0 Y3(1)	Y3(0) Y3(1) ----	X2(0) ----- X1(0)	X1(0) ---- ----	X(0,0) -----
21	LL1,1 HL0,1 HH0,1	XL1(0) YL1(1)	XH0(6) -------	XL0(6) --------	----- L5,0 H5,0	L4,0 H4,0	Y3(0) Y3(1) ----	X2(0) -----	X1(0) ---- ----	X(0,0) -----
22	------- LL1,1 LH1,1	XH1(0) YH1(1)	XL1(0) YL1(1)	XH0(6) -------	L6,0 ------ ------	L5,0 H5,0 Y4(1)	Y4(0) Y4(1) ----	X3(0) ----- X2(0)	X2(0) ---- ----	X(1,0) -----
23	HL1,1 LL1,1 LH1,1	XH1(0) YH1(1)	XL1(0) YL1(1)	XH0(6) -------	L6,0 ------ ------	L5,0 H5,0	Y4(0) Y4(1) ----	X3(0) -----	X2(0) ---- ----	X(1,0) -----
24	------- HL1,1 HH1,1	XL1(2) YL1(3)	XH1(0) YH1(1)	XL1(0) YL1(1)	------ ------ ------	L6,0 H6,0 Y5(1)	Y5(0) Y5(1) ----	X4(0) ----- X3(0)	X3(0) ---- ----	X(2,0) -----
25	LL2,1 HL1,1 HH1,1	XL1(2) YL1(3)	XH1(0) YH1(1)	XL1(0) YL1(1)	------ ------ ------	L6,0 H6,0	Y5(0) Y5(1) ----	X4(0) -----	X3(0) ---- ----	X(2,0) -----
26	------- LL2,1 LH2,1	XH1(2) YH1(3)	XL1(2) YL1(3)	XH1(0) YH1(1)	L0,1 L1,1 ------	------ ----- Y6(1)	Y6(0) Y6(1) ----	X5(0) ----- X4(0)	X4(0) ---- ----	X(3,0) -----
27	HL2,1 LL2,1 LH2,1	XH1(2) YH1(3)	XL1(2) YL1(3)	XH1(0) YH1(1)	L0,1 L1,1 ------	------ -----	Y6(0) Y6(1) ----	X5(0) -----	X4(0) ---- ----	X(3,0) -----
28	------- HL2,1 HH2,1	XL1(4) YL1(5)	XH1(2) YH1(3)	XL1(2) YL1(3)	----- L1,1 H1,1	L0,1 H0,1	------ ----- ----	X6(0) ----- X5(0)	X5(0) ---- ----	X(4,0) -----
29	LL3,1 HL2,1 HH2,1	XL1(4) YL1(5)	XH1(2) YH1(3)	XL1(2) YL1(3)	----- L1,1 H1,1	L0,1 H0,1	------ ----- ----	X6(0) -----	X5(0) ---- ----	X(4,0) -----
30	------- LL3,1 -------	XH1(4) YH1(5)	XL1(4) YL1(5)	XH1(2) YH1(3)	L2,1 L3,1 ------	L1,1 H1,1 Y0(3)	Y0(2) Y0(3) Y0(1)	------- ----- X6(0)	X6(0) ---- ----	X(5,0) -----
31	HL3,1 LL3,1 -------	XH1(4) YH1(5)	XL1(4) YL1(5)	XH1(2) YH1(3)	L2,1 L3,1 ------	L1,1 H1,1	Y0(2) Y0(3) Y0(1)	------- -----	X6(0) ---- ----	X(5,0) -----
32	------- HL3,1 -------	XL1(6) -------	XH1(4) YH1(5)	XL1(4) YL1(5)	----- L3,1 H3,1	L2,1 H2,1 Y1(3)	Y1(2) Y1(3) Y1(1)	X0(2) Y0(1)	------ ---- ----	X(6,0) -----
33	------- HL3,1 -------	XL1(6) -------	XH1(4) YH1(5)	XL1(4) YL1(5)	----- L3,1 H3,1	L2,1 H2,1	Y1(2) Y1(3) Y1(1)	X0(2) Y0(1)	------ ---- ----	X(6,0) -----
34	------- ------- -------	XH1(6) -------	XL1(6) -------	XH1(4) YH1(5)	L4,1 L5,1 ------	L3,1 H3,1 Y2(3)	Y2(2) Y2(3) Y2(1)	X1(2) Y1(1) X0(2)	X0(2) Y0(1) X0(0)	------- -----
35	------- ------- -------	XH1(6) -------	XL1(6) -------	XH1(4) YH1(5)	L4,1 L5,1 ------	L3,1 H3,1	Y2(2) Y2(3) Y2(1)	X1(2) Y1(1)	X0(2) Y0(1) X0(0)	------- -----
36	------- ------- -------	--------- --------	XH1(6) -------	XL1(6) -------	----- L5,1 H5,1	L4,1 H4,1 Y3(3)	Y3(2) Y3(3) Y3(1)	X2(2) Y2(1) X1(2)	X1(2) Y1(1) X1(0)	X(0,2) X(0,1)
37	------- ------- -------	--------- --------	XH1(6) -------	XL1(6) -------	----- L5,1 H5,1	L4,1 H4,1	Y3(2) Y3(3) Y3(1)	X2(2) Y2(1)	X1(2) Y1(1) X1(0)	X(0,2) X(0,1)
38	------- ------- -------	--------- --------	--------- --------	XH1(6) -------	L6,1 ------ ------	L5,1 H5,1 Y4(3)	Y4(2) Y4(3) Y4(1)	X3(2) Y3(1) X2(2)	X2(2) Y2(1) X2(0)	X(1,2) X(1,1)
39	------- ------- -------	--------- --------	--------- --------	XH1(6) -------	L6,1 ------ ------	L5,1 H5,1	Y4(2) Y4(3) Y4(1)	X3(2) Y3(1)	X2(2) Y2(1) X2(0)	X(1,2) X(1,1)
40	------- ------- -------	--------- --------	--------- --------	--------- --------	------ ------ ------	L6,1 H6,1 Y5(3)	Y5(2) Y5(3) Y5(1)	X4(2) Y4(1) X3(2)	X3(2) Y3(1) X3(0)	X(2,2) X(2,1)
41	------- ------- -------	--------- --------	--------- --------	--------- --------	------ ------ ------	L6,1 H6,1	Y5(2) Y5(3) Y5(1)	X4(2) Y4(1)	X3(2) Y3(1) X3(0)	X(2,2) X(2,1)
42	------- ------- -------	--------- --------	--------- --------	--------- --------	------ ------ ------	----- ----- Y6(3)	Y6(2) Y6(3) Y6(1)	X5(2) Y4(1) X4(2)	X4(2) Y4(1) X4(0)	X(3,2) X(3,1)
43	------- ------- -------	--------- --------	--------- --------	--------- --------	------ ------ ------	----- -----	Y6(2) Y6(3) Y6(1)	X5(2) Y4(1)	X4(2) Y4(1) X4(0)	X(3,2) X(3,1)

Table B.19 (a) dataflow for 9/7 architecture from CP side

Ck *f*/2	1 CP input latches Rt0 Rt1	2 Rt0 Rt1	3 Rt0 Rt1	5 Rt0 Rt1	6 Rt0 Rt1	8 Rt0 Rt1	9 Rt0 Rt1	11 Rt0 Rt1	12 Rt0 Rt1	13 Rt0 Rt1	CP output latches Rtl0 Rtl1 Rth
1	LL0,0 LH0,0										
2	HL0,0 HH0,0	YL'0,0 YL'0,1									
3	LL0,1 LH0,1	YH'0,0 YH'0,1	YL"0,0 YL'0,1								
4	HL0,1 HH0,1	YL'1,0 YL'1,1	YH"0,0 YH'0,1								
5	LL1,0 LH1,0	YH'1,0 YH'1,1	YL"1,0 YL'1,1	YL"0,0 YL'0,1							
6	HL1,0 HH1,0	YL'0,2 YL'0,3	YH"1,0 YH'1,1	YH"0,0 YH'0,1	YL"0,0 YL'0,1						
7	LL1,1 LH1,1	YH'0,2 YH'0,3	YL"0,2 YL'0,3	YL"1,0 YL'1,1	YH"0,0 YH'0,1						
8	HL1,1 HH1,1	YL'1,2 YL'1,3	YH"0,2 YH'0,3	YH"1,0 YH'1,1	YL"1,0 YL'1,1	YL"0,0 YL"0,1					
9	LL2,0 LH2,0	YH'1,2 YH'1,3	YL"1,2 YL'1,3	YL"0,2 YL'0,3	YH"1,0 YH'1,1	YH"0,0 YH"0,1	XL0,0 YL"0,1				
10	HL2,0 HH2,0	YL'0,4 YL'0,5	YH"1,2 YH'1,3	YH"0,2 YH'0,3	YL"0,2 YL'0,3	YL"1,0 YL"1,1	XH0,0 YH"0,1				
11	LL2,1 LH2,1	YH'0,4 YH'0,5	YL"0,4 YL'0,5	YL"1,2 YL'1,3	YH"0,2 YH'0,3	YH"1,0 YH"1,1	XL1,0 YL"1,1	XL0,0 YL"0,1			
12	HL2,1 HH2,1	YL'1,4 YL'1,5	YH"0,4 YH'0,5	YH"1,2 YH'1,3	YL"1,2 YL'1,3	YL"0,2 YL"0,3	XH1,0 YH"1,1	XH0,0 YH"0,1	XL0,0 YL"0,1		
13	LL3,0 LH3,0	YH'1,4 YH'1,5	YL"1,4 YL'1,5	YL"0,4 YL'0,5	YH"1,2 YH'1,3	YH"0,2 YH"0,3	XL0,2 YL"0,3	XL1,0 YL"1,1	XH0,0 YH"0,1	XL0,0 YL"0,1	
14	HL3,0 HH3,0	YL'0,6 YL'0,7	YH"1,4 YH'1,5	YH"0,4 YH'0,5	YL"0,4 YL'0,5	YL"1,2 YL"1,3	XH0,2 YH"0,3	XH1,0 YH"1,1	XL1,0 YL"1,1	XH0,0 YH"0,1	L0,0 L1,0 ----
15	LL3,1 LH3,1	YH'0,6 YH'0,7	YL"0,6 YL'0,7	YL"1,4 YL'1,5	YH"0,4 YH'0,5	YH"1,2 YH"1,3	XL1,2 YL"1,3	XL0,2 YL"0,3	XH1,0 YH"1,1	XL1,0 YL"1,1	----- L1,0 H1,0
16	HL3,1 HH3,1	YL'1,6 YL'1,7	YH"0,6 YH'0,7	YH"1,4 YH'1,5	YL"1,4 YL'1,5	YL"0,4 YL"0,5	XH1,2 YH"1,3	XH0,2 YH"0,3	XL0,2 YL"0,3	XH1,0 YH"1,1	L0,1 L1,1 ----
17	LL4,0 ------	YH'1,6 YH'1,7	YL"1,6 YL'1,7	YL"0,6 YL'0,7	YH"1,4 YH'1,5	YH"0,4 YH"0,5	XL0,4 YL"0,5	XL1,2 YL"1,3	XH0,2 YH"0,3	XL0,2 YL"0,3	----- L1,1 H1,1
18	HL4,0 ------	YL'0,8 ------	YH"1,6 YH'1,7	YH"0,6 YH'0,7	YL"0,6 YL'0,7	YL"1,4 YL"1,5	XH0,4 YH"0,5	XH1,2 YH"1,3	XL1,2 YL"1,3	XH0,2 YH"0,3	L2,0 L3,0 ----
19	LL4,1 ------	YH'0,8 ------	YL"0,8 ------	YL"1,6 YL'1,7	YH"0,6 YH'0,7	YH"1,4 YH"1,5	XL1,4 YL"1,5	XL0,4 YL"0,5	XH1,2 YH"1,3	XL1,2 YL"1,3	----- L3,0 H3,0
20	HL4,1 ------	YL'1,8 ------	YH"0,8 ------	YH"1,6 YH'1,7	YL"1,6 YL'1,7	YL"0,6 YL"0,7	XH1,4 YH"1,5	XH0,4 YH"0,5	XL0,4 YL"0,5	XH1,2 YH"1,3	L2,1 L3,1 ----
21	------- ------	YH'1,8 ------	YL"1,8 ------	YL"0,8 -------	YH"1,6 YH'1,7	YH"0,6 YH"0,7	XL0,6 YL"0,7	XL1,4 YL"1,5	XH0,4 YH"0,5	XL0,4 YL"0,5	----- L3,1 H3,1
22	------- ------	---------- -------	YH"1,8 ------	YH"0,8 ------	YL"0,8 ------	YL"1,6 YL"1,7	XH0,6 YH"0,7	XH1,4 YH"1,5	XL1,4 YL"1,5	XH0,4 YH"0,5	L4,0 L5,0 ----
23	------- ------	---------- -------	---------- -------	YL"1,8 -------	YH"0,8 ------	YH"1,6 YH"1,7	XL1,6 YL"1,7	XL0,6 YL"0,7	XH1,4 YH"1,5	XL1,4 YL"1,5	----- L5,0 H5,0
24	------- ------	---------- -------	---------- -------	YH"1,8 ------	YL"1,8 -------	YL"0,8) -------	XH1,6 YH"1,7	XH0,6 YH"0,7	XL0,6 YL"0,7	XH1,4 YH"1,5	L4,1 L5,1 ----
25	LL0,2 LH0,2	---------- -------	---------- -------	-------- -------	YH"1,8 ------	YH"0,8 -------	XL0,8) -------	XL1,6 YL"1.7	XH0,6 YH"0,7	XL0,6 YL"0,7	----- L5,1 H5,1
26	HL0,2 HH0,2	YL'2,0 YL'2,1	---------- -------	-------- -------	---------- -------	YL"1,8 ------	XH0,8 ------	XH1,6 YH"1,7	XL1,6 YL"1,7	XH0,6 YH"0,7	L6,0 L7,0 ----
27	LL1,2 LH1,2	YH'2,0 YH'2,1	YL"2,0 ------	-------- YL'2,1	---------- -------	YH"1,8 -------	XL1,8 ------	XL0,8) -------	XH1,6 YH"1,7	XL1,6 YL"1,7	----- L7,0 H7,0
28	HL1,2 HH1,2	YL'2,2 YL'2,3	YH"2,0 ------	-------- YH'2,1	YL"2,0 YL'2,1	---------- -------	XH1,8 ------	XH0,8 ------	XL0,8 ------	XH1,6 YH"1,7	L6,1 L7,1 ----
29	LL2,2 LH2,2	YH'2,2 YH'2,3	YL"2,2 ------	-------- YL'2,3	YH"2,0 YH'2,1	---------- -------	-------- -------	XL1,8 ------	XH0,8 ------	XL0,8 ------	----- L7,1 H7,1
30	HL2,2 HH2,2	YL'2,4 YL'2,5	YH"2,2 ------	-------- YH'2,3	YL"2,2 YL'2,3	YL"2,0 YL"2,1	-------- -------	XH1,8 ------	XL1,8 -------	XH0,8 -------	L8,0 ---- -----
31	LL3,2 LH3,2	YH'2,4 YH'2,5	YL"2,4 ------	-------- YL'2,5	YH"2,2 YH'2,3	YH"2,0 YH"2,1	XL2,0 -------	------- YL"2,1	XH1,8 -------	XL1,8 -------	------ ----- -----
32	HL3,2 HH3,2	YL'2,6 YL'2,7	YH"2,4 ------	-------- YH'2,5	YL"2,4 YL'2,5	YL"2,2 YL"2,3	XH2,0 -------	------- YH"2,1	XL2,0 YL"2,1	XH1,8 -------	L8,1 ---- -----
33	LL4,2 -------	YH'2,6 YH'2,7	YL"2,6 ------	-------- YL'2,7	YH"2,4 YH'2,5	YH"2,2 YH"2,3	XL2,2 -------	------- YL"2,3	XH2,0 YH"2,1	XL2,0 YL"2,1	------ ----- -----
34	HL4,2 -------	YL'2,8 -------	YH"2,6 -------	-------- YH'2,7	YL"2,6 YL'2,7	YL"2,4 YL"2,5	XH2,2 -------	------- YH"2,3	XL2,2 YL"2,3	XH2,0 YH"2,1	L0,2 L1,2 -----
35	------- -------	YH'2,8 -------	YL"2,8 -------	-------- --------	YH"2,6 YH'2,7	YH"2,4 YH"2,5	XL2,4 -------	------- YL"2,5	XH2,2 YH"2,3	XL2,2 YL"2,3	----- L1,2 H1,2
36	------- -------	------- -------	YH"2,8 -------	-------- --------	YL"2,8 ------	YL"2,6 YL"2,7	XH2,4 -------	------- YH"2,5	XL2,4 YL"2,5	XH2,2 YH"2,3	L2,2 L3,2 -----
37	------- -------	------- -------	------- -------	------- -------	YH"2,8 -------	YH"2,6 YH"2,7	XL2,6 -------	------- YL"2,7	XH2,4 YH"2,5	XL2,4 YL"2,5	----- L3,2 H3,2
38	------- -------	------- -------	------- -------	------- -------	------- -------	YL"2,8 -------	XH2,6 -------	------- YH"2,7	XL2,6 YL"2,7	XH2,4 YH"2,5	L4,2 L5,2 -----
39	------- -------	------- -------	------- -------	------- -------	------- -------	YH"2,8 -------	XL2,8 -------	------- -------	XH2,6 YH"2,7	XL2,6 YL"2,7	----- L5,2 H5,2
40	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	XH2,8 -------	------- -------	XL2,8 -------	XH2,6 YH"2,7	L6,2 L7,2 -----
41	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	XH2,8 -------	XL2,8 -------	----- L7,2 H7,2
42	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	XH2,8 -------	L8,2 ----- -----
43	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	------- -------	----- ----- -----

Table B.19 (b) dataflow for 9/7 architecture from RP side

Ck *f*/2	1 RP input	2	3	4	5	6	7	8	9	RP out latches
	Rt0 Rt1	Rt0 Rt1 TLB1	Rt0 Rt1 Rt2 R1 R0	Rt0 Rt1 TLB2 R1 R0	Rt0 Rt1 Rt2	Rt0 Rt1 TLB2	Rt0 Rt1 Rt2	Rt0 Rt1 TLB4	Rt0 Rt1 Rt2	Rt0 Rt1
14	---- ----									
15	L0,0 H0,0									
16	L1,0 H1,0	Y0,0 Y0,1								
17	L0,1 H0,1	Ỹ1,0 Ỹ1,1	Y0,0 Y0,1 ---- ---- ----							
18	L1,1 H1,1	Ỹ0,2 Ỹ0,3	Ỹ1,0 Ỹ1,1 ---- Ỹ0,1 ----	Y̌0,0 ---- ---- ---- ----						
19	L2,0 H2,0	Ỹ1,2 Ỹ1,3 Ỹ0,3	Ỹ0,2 Ỹ0,3 ---- Ỹ1,1 Ỹ0,1	Y̌1,0 ---- ---- Y̌0,0 ----						
20	L3,0 H3,0	Ỹ2,0 Ỹ2,1 Ỹ1,3	Ỹ1,2 Ỹ1,3 ---- ---- Ỹ1,1	Y̌0,2 Y̌0,1 ---- Y̌1,0 Y̌0,0						
21	L2,1 H2,1	Ỹ3,0 Ỹ3,1	Ỹ2,0 Ỹ2,1 ---- ---- ----	Y̌1,2 Y̌1,1 Y̌0,2 ---- Y̌1,0	Y̌0,2 Y̌0,1 Y̌0,0					
22	L3,1 H3,1	Ỹ2,2 Ỹ2,3	Ỹ3,0 Ỹ3,1 ---- Ỹ2,1 ----	Y̌2,0 ---- Y̌1,2 ---- ----	Y̌1,2 Y̌1,1 Y̌1,0	Y̌0,0 Y̌0,1 ----				
23	L4,0 H4,0	Ỹ3,2 Ỹ3,3 Ỹ2,3	Ỹ2,2 Ỹ2,3 ---- Ỹ3,1 Ỹ2,1	Y̌1,0 ---- Y̌2,0 ----	---- ---- ----	Y̌1,0 Y̌1,1 Y̌0,1	Y̌0,0 Y̌0,1 ----			
24	L5,0 H5,0	Ỹ4,0 Ỹ4,1 Ỹ3,3	Ỹ3,2 Ỹ3,3 ---- ---- Ỹ3,1	Y̌3,2 Y̌2,1 Y̌3,0 Y̌2,0	---- ---- ----	---- ---- Y̌1,1	Y̌1,0 Y̌1,1 ----	x0,0 ---- ----		
25	L4,1 H4,1	Ỹ5,0 Ỹ5,1	Ỹ4,0 Ỹ4,1 ---- ---- ----	Y̌3,2 Y̌3,1 Y̌2,2 ---- Y̌3,0	Y̌2,2 Y̌2,1 Y̌2,0	---- ----	---- ---- ----	x1,0 ---- x0,0	x0,0 ---- ----	
26	L5,1 H5,1	Ỹ4,2 Ỹ4,3	Ỹ5,0 Ỹ5,1 ---- Ỹ4,1 ----	Y̌4,0 ---- Y̌3,2 ---- ----	Y̌3,2 Y̌3,1 Y̌3,0	Y̌2,0 Y̌2,1	---- ---- ----	---- ---- x1,0	x1,0 ---- ----	x0,0 ----
27	L6,0 H6,0	Ỹ5,2 Ỹ5,3 Ỹ4,3	Ỹ4,2 Ỹ4,3 ---- Ỹ5,1 Ỹ4,1	Y̌5,0 ---- Y̌4,0 ----	---- ---- ----	Y̌3,0 Y̌3,1 Y̌2,1	Y̌2,0 Y̌2,1 ----	---- ----	---- ---- ----	x1,0 ----
28	L7,0 H7,0	Ỹ6,0 Ỹ6,1 Ỹ5,3	Ỹ5,2 Ỹ5,3 ---- ---- Ỹ5,1	Y̌4,2 Y̌4,1 Y̌5,0 Y̌4,0	---- ---- ----	---- ---- Y̌3,1	Y̌3,0 Y̌3,1 ----	x2,0 ----	---- ---- ----	---- ----
29	L6,1 H6,1	Ỹ7,0 Ỹ7,1	Ỹ6,0 Ỹ6,1 ---- ---- ----	Y̌5,2 Y̌5,1 Y̌4,2 ---- Y̌5,0	Y̌4,2 Y̌4,1 Y̌4,0	---- ----	---- ---- ----	x3,0 ---- x2,0	x2,0 ---- ----	---- ----
30	L7,1 H7,1	Ỹ6,2 Ỹ6,3	Ỹ7,0 Ỹ7,1 ---- Ỹ6,1 ----	Y̌6,0 ---- Y̌5,2 ---- ----	Y̌5,2 Y̌5,1 Y̌5,0	Y̌4,0 Y̌4,1	---- ---- ----	---- ---- x3,0	x3,0 ---- ----	x2,0 ----
31	L8,0 H8,0	Ỹ7,2 Ỹ7,3 Ỹ6,3	Ỹ6,2 Ỹ6,3 ---- Ỹ7,1 Ỹ6,1	Y̌7,0 ---- Y̌6,0 ----	---- ---- ----	Y̌5,0 Y̌5,1 Y̌4,1	Y̌4,0 Y̌4,1 ----	---- ----	---- ---- ----	x3,0 ----
32	---- ----	Ỹ8,0 Ỹ8,3 Ỹ7,3	Ỹ7,2 Ỹ7,3 ---- ---- Ỹ7,1	Y̌6,2 Y̌6,1 Y̌7,0 Y̌6,0	---- ---- ----	---- ---- Y̌5,1	Y̌5,0 Y̌5,1 ----	x4,0 ----	---- ---- ----	---- ----
33	L8,1 H8,1	---- ----	Ỹ8,0 Ỹ8,1 ---- ---- ----	Y̌7,2 Y̌7,1 Y̌6,2 ---- Y̌7,0	Y̌6,2 Y̌6,1 Y̌6,0	---- ----	---- ---- ----	x5,0 ---- x4,0	x4,0 ---- ----	---- ----
34	---- ----	Ỹ8,2 Ỹ8,3	---- ---- ---- Ỹ8,1 ----	Y̌8,0 ---- Y̌7,2 ---- ----	Y̌7,2 Y̌7,1 Y̌7,0	Y̌6,0 Y̌6,1	---- ---- ----	---- ---- x5,0	x5,0 ---- ----	x4,0 ----
35	L0,2 H0,2	---- ---- Ỹ8,3	Ỹ8,2 Ỹ8,3 ---- ---- Ỹ8,1	---- ---- Y̌8,0 ----	---- ---- ----	Y̌7,0 Y̌7,1 Y̌6,1	Y̌6,0 Y̌6,1 ----	---- ----	---- ---- ----	x5,0 ----
36	L1,2 H1,2	Ỹ0,4 Ỹ0.5	---- ---- ---- ---- ----	Y̌8,2 Y̌8,1 ---- Y̌8,0	---- ---- ----	---- ---- Y̌7,1	Y̌7,0 Y̌7,1 ----	x6,0 ---- ----	---- ---- ----	---- ----
37	L2,2 H2,2	Ỹ1.4 Ỹ1,5 Ỹ0,5	Ỹ0,4 Ỹ0.5 Ỹ0,3 ---- ----	---- ---- Y̌8,2 ---- ----	Y̌8,2 Y̌8,1 Y̌8,0	---- ----	---- ---- ----	x7,0 ---- x6,0	x6,0 ---- ----	---- ----
38	L3,2 H3,2	Ỹ2,4 Ỹ2,5 Ỹ1,5	Ỹ1,4 Ỹ1.5 Ỹ1,3 ---- ----	Y̌0,4 Y̌0,3 ---- ---- ----	---- ---- ----	Y̌8,0 Y̌8,1	---- ---- ----	---- ---- x7,0	x7,0 ---- ----	x6,0 ----
39	L4,2 H4,2	Ỹ3,4 Ỹ3,5 Ỹ2,5	Ỹ2,4 Ỹ2.5 Ỹ2,3 ---- ----	Y̌1,4 Y̌1,3 Y̌0,4 ---- ----	Y̌0,4 Y̌0,3 Y̌0,2	---- ---- Y̌8,1	Y̌8,0 Y̌8,1 ----	---- ----	---- ---- ----	x7,0 ----
40	L5,2 H5,2	Ỹ4,4 Ỹ4,5 Ỹ3,5	Ỹ3,4 Ỹ3.5 Ỹ3,3 ---- ----	Y̌2,4 Y̌2.3 Y̌1,4 ---- ----	Y̌1,4 Y̌1,3 Y̌1,2	Y̌0,2 Y̌0,3	---- ---- ----	x8,0) ----	---- ---- ----	---- ----
41	L6,2 H6,2	Ỹ5.4 Ỹ5,5 Ỹ4,5	Ỹ4,4 Ỹ4.5 Ỹ4,3 ---- ----	Y̌3,4 Y̌3.3 Y̌2,4 ---- ----	Y̌2,4 Y̌2,3 Y̌2,2	Y̌1,2 Y̌1,3 Y̌0,3	Y̌0.2 Y̌0.3 Y̌0,1	---- ---- x8,0	x8,0 ---- ----	---- ----
42	L7,2 H7,2	Ỹ6.4 Ỹ6.5 Ỹ5.5	Ỹ5.4 Ỹ5.5 Ỹ5,3 ---- ----	Y̌4.4 Y̌4.3 Y̌3,4 ---- ----	Y̌3.4 Y̌3,3 Y̌3.2	Y̌2,2 Y̌2,3 Y̌1,3	Y̌1,2 Y̌1,3 Y̌1,1	x0,2 Y̌0.1	---- ---- ----	x8,0 ----
43	L8,2 H8,2	Ỹ7,4 Ỹ7,5 Ỹ6,5	Ỹ6.4 Ỹ6,5 Ỹ6,3 ---- ----	Y̌5,4 Y̌5,3 Y̌4,4 ---- ----	Y̌4.4 Y̌4,3 Y̌4,2	Y̌3,2 Y̌3,3 Y̌2,3	Y̌2,2 Y̌2,3 Y̌2,1	x1,2 Y̌1,1 x0,2	x0,2 Y̌0,1 x0,0	---- ----
44	---- ----	Ỹ8,4 Ỹ8,5 Ỹ7,5	Ỹ7,4 Ỹ7,5 Ỹ7,3 ---- ----	Y̌6,4 Y̌6.3 Y̌5,4 ---- ----	Y̌5.4 Y̌5,3 Y̌5,2	Y̌4,2 Y̌4,3 Y̌3,3	Y̌3,2 Y̌3,3 Y̌3.1	x2,2 Y̌2,1 x1,2	x1,2 Y̌1,1 x1,0	x0,2 x0,1
45	---- ----	---- ---- Ỹ8,5	Ỹ8.4 Ỹ8.5 Ỹ8,3 ---- ----	Y̌7.4 Y̌7.3 Y̌6.4 ---- ----	Y̌6.4 Y̌6,3 Y̌6,2	Y̌5,2 Y̌5,3 Y̌4,3	Y̌4,2 Y̌4,3 Y̌4,1	x3,2 Y̌3,1 x2,2	x2,2 Y̌2,1 x2,0	x1,2 x1,1

Table B.20 Dataflow for 2-parallel inverse 5/3 architecture

	Ck f_2	CP	CP1 & CP2 input latches		CP1 output latches		CP2 output latches		RP1 input latches		RP2 input latches		Output latches of RP1		RP2	
			Rt0	Rt1	Rtl0	Rtl1	Rth0	Rth1	Rt0	Rt1	Rt0	Rt1	Rt0	Rt1	Rt0	Rt1
RUN 1	1	1	LL0,0	LH0,0												
	2	2	HL0,0	HH0,0												
	3	1	LL0,1	LH0,1												
	4	2	HL0,1	HH0,1												
	5	1	LL1,0	LH1,0												
	6	2	HL1,0	HH1,0												
	7	1	LL1,1	LH1,1												
	8	2	HL1,1	HH1,1												
	9	1	LL2,0	LH2,0	L0,0	L1,0										
	10	2	HL2,0	HH2,0			H0,0	H1,0								
	11	1	LL2,1	LH2,1	L0,1	L1,1			L0,0	H0,0	L1,0	H1,0				
	12	2	HL2,1	HH2,1			H0,1	H1,1								
	13	1	LL3,0	LH3,0	L2,0	L3,0			L0,1	H0,1	L1,1	H1,1				
	14	2	HL3,0	HH3,0			H2,0	H3,0								
	15	1	LL3,1	LH3,1	L2,1	L3,1			L2,0	H2,0	L3,0	H3,0				
	16	2	HL3,1	HH3,1			H2,1	H3,1								
RUN 2	17	1	LL0,2	LH0,2	L4,0	L5,0			L2,1	H2,1	L3,1	H3,1				
	18	2	HL0,2	HH0,2			H4,0	H5,0								
	19	1	LL1,2	LH1,2	L4,1	L5,1			L4,0	H4,0	L5,0	H5,0	X0,0	-----	X1,0	------
	20	2	HL1,2	HH1,2			H4,1	H5,1								
	21	1	LL2,2	LH2,2	L6,0	L7,0			L4,1	H4,1	L5,1	H5,1	X0,2	X0,1	X1,2	X1,1
	22	2	HL2,2	HH2,2			H6,0	H7,0								
	23	1	LL3,2	LH3,2	L6,1	L7,1			L6,0	H6,0	L7,0	H7,0	X2,0	-----	X3,0	-----
	24	2	HL3,2	HH3,2			H6,1	H7,1								
	25	1			L0,2	L1,2			L6,1	H6,1	L7,1	H7,1	X2,2	X2,1	X3,2	X3,1
	26	2					H0,2	H1,2								
	27	1			L2,2	L3,2			L0,2	H0,2	L1,2	H1,2	X4,0	------	X5,0	------
	28	2					H2,2	H3,2								
	29	1			L4,2	L5,2			L2,2	H2,2	L3,2	H3,2	X4,2	X4,1	X5,2	X5,1
	30	2					H4,2	H5,2								
	31	1			L6,2	L7,2			L4,2	H4,2	L5,2	H5,2	X6,0	-----	X7,0	-----
	32	2					H6,2	H7,2								
	33	1							L6,2	H6,2	L7,2	H7,2	X6,2	X6,1	X7,2	X7,1
	34	2														
	35	1											X0,4	X0,3	X1,4	X1,3
	36	2														
	37	1											X2,4	X2,3	X3,4	X3,3
	38	2														
	39	1											X4,4	X4,3	X5,4	X5,3

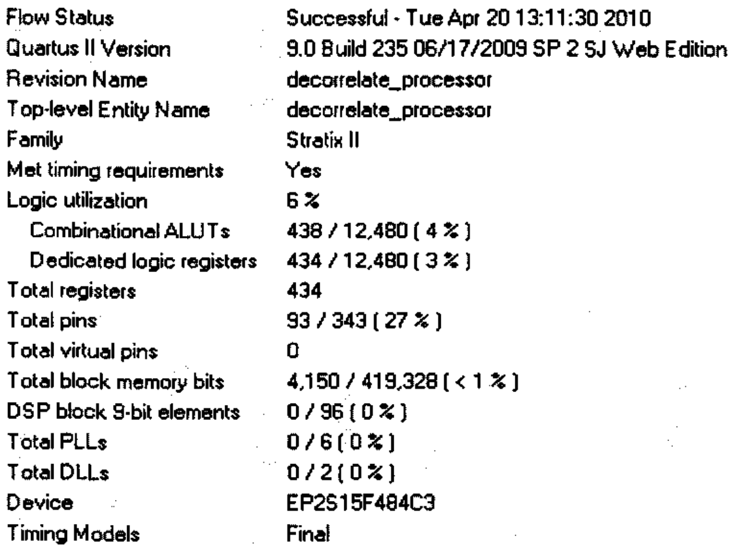
Table B.21 Dataflow for 4-parallel inverse 5/3 architecture

	CK f_s	CP	CPs input Latches Rt0 Rt1	CPs 1 &3 Out latches Rtl0 Rtl1	CPs 2 & 4 Out latches Rth0 Rth1	RPs 1 & 3 input latches RP Rt0 Rt1 Rt2	RPs 2 & 4 input latches RP Rt0 Rt1 Rt2	RPs 1 & 3 Out latches Rt0 Rt1	RPs 2 & 4 Out latches Rt0 Rt1
RUN 1	1	1	LL0,0 LH0,0						
	2	2	HL0,0 HH0,0						
	3	3	LL0,1 LH0,1						
	4	4	HL0,1 HH0,1						
	5	1	LL1,0 LH1,0						
	6	2	HL1,0 HH1,0						
	7	3	LL1,1 LH1,1						
	8	4	HL1,1 HH1,1						
	9	1	LL2,0 LH2,0						
	10	2	HL2,0 HH2,0						
	11	3	LL2,1 LH2,1						
	12	4	HL2,1 HH2,1						
	13	1	LL3,0 LH3,0	L0,0 L1,0					
	14	2	HL3,0 HH3,0		H0,0 H1,0				
	15	3	LL3,1 LH3,1	L0,1 L1,1					
	16	4	HL3,1 HH3,1		H0,1 H1,1				
	17	1	LL4,0 ------	L2,0 L3,0		1 L0,0 H0,0 H0,1	2 L1,0 H1,0 H1,1		
	18	2	HL4,0 ------		H2,0 H3,0	3 L0,1 H0,1 H0,0	4 L1,1 H1,1 H1,0		
	19	3	LL4,1 ------	L2,1 L3,1					
	20	4	HL4,1 ------		H2,1 H3,1				
RUN 2	21	1	LL0,2 LH0,2	L4,0 L5,0		1 L2,0 H2,0 -----	2 L3,0 H3,0 -----		
	22	2	HL0,2 HH0,2		H4,0 H5,0	3 L2,1 H2,1 H2,0	4 L3,1 H3,1 H3,0		
	23	3	LL1,2 LH1,2	L4,1 L5,1					
	24	4	HL1,2 HH1,2		H4,1 H5,1				
	25	1	LL2,2 LH2,2	L6,0 L7,0		1 L4,0 H4,0 H4,1	2 L5,0 H5,0 H5,1		
	26	2	HL2,2 HH2,2		H6,0 H7,0	3 L4,1 H4,1 H4,0	4 L5,1 H5,1 H5,0		
	27	3	LL3,2 LH3,2	L6,1 L7,1					
	28	4	HL3,2 HH3,2		H6,1 H7,1				
	29	1	LL4,2 ------	L8,0 -----		1 L6,0 H6,0 -----	2 L7,0 H7,0 -----		
	30	2	HL4,2 ------		H8,0 -----	3 L6,1 H6,1 H6,0	4 L7,1 H7,1 H7,0		
	31	3	------- -------	L8,1 -----					
	32	4	------- -------		H8,1 -----				
RUN 3	33	1	LL0,3 LH0,3	L0,2 L1,2		1 L8,0 H8,0 H8,1	2 ----- ---- ----	X0,0 ----	X1,0 -----
	34	2			H0,2 H1,2	3 L8,1 H8,1 H8,0	4 ----- ---- ----	X0,2 X0,1	X1,2 X1,1
	35	3	LL1,3 LH1,3	L2,2 L3,2					
	36	4			H2,2 H3,2				
	37	1	LL2,3 LH2,3	L4,2 L5,2		1 L0,2 H0,2 -----	2 L1,2 H1,2 -----	X2,0 ----	X3,0 -----
	38	2			H4,2 H5,2	3 L2,2 H2,2 -----	4 L3,2 H3,2 -----	X2,2 X2,1	X3,2 X3,1
	39	3	LL3,3 LH3,3	L6,2 L7,2					
	40	4			H6,2 H7,2				
	41	1	LL4,3 ------	L8,2 -----		1 L4,2 H4,2 -----	2 L5,2 H5,2 -----	X4,0 ----	X5,0 -----
	42	2			H8,2 ------	3 L6,2 H6,2 -----	4 L7,2 H7,2 -----	X4,2 X4,1	X5,2 X5,1
	43	3	------- -------	----- -----					
	44	4	------- -------		------ ------				
	45	1		L0,3 L1,3		1 L8,2 H8,2 -----	2 ----- ---- ----	X6,0 ----	X7,0 -----
	46	2			------ ------	3 ----- ----- -----	4 ----- ---- ----	X6,2 X6,1	X7,2 X7,1
	47	3		L2,3 L3,3					
	48	4			------ ------				
	49	1		L4,3 L5,3		1 L0,3 ------ -----	2 L1,3 ----- -----	X8,0 ----	----- -----
	50	2			------ ------	3 L2,3 ------ -----	4 L3,3 ----- -----	X8,2 X8,1	----- -----
	51	3		L6,3 L7,3					
	52	4			------ ------				
	53	1		L8,3 -----		1 L4,3 ------ -----	2 L5,3 ------ -----	X0,4 X0,3	X1,4 X1,3
	54	2			------ ------	3 L6,3 ------ -----	4 L7,3 ----- -----	X2,4 X2,3	X3,4 X3,3
	55	3		------ ------					
	56	4			----- ------				
	57	1				1 L8,3 ------ -----	2 ----- ---- ----	X4,4 X4,3	X5,4 X5,3
	58	2				3 ----- ----- -----	4 ----- ---- ----	X6,4 X6,3	X7,4 X7,3
	59	3							
	60	4							
	61	1						X8,4 X8,3	----- ------
	62	2						------ ------	----- ------

APPENDIX C

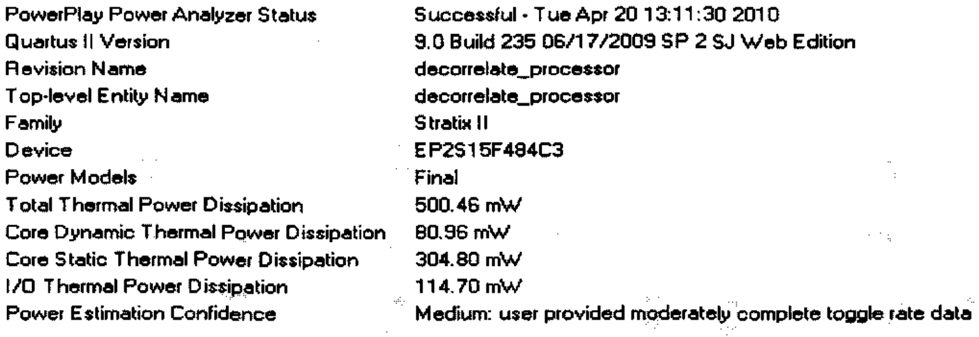
FPGA COMPILATION AND SYNTHESIS RESULTS

C.1 Compilation reports for forward 5/3 module "decorrelate_processor"



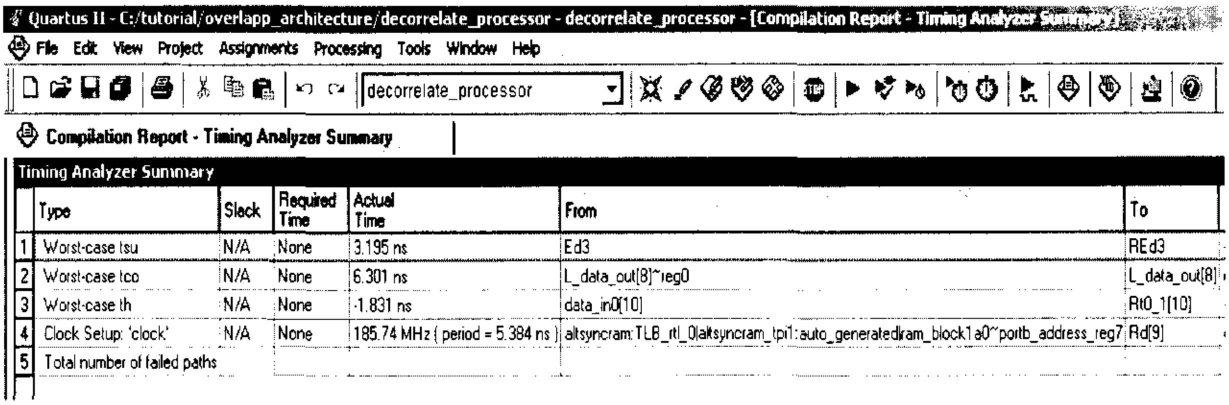
Flow Status	Successful - Tue Apr 20 13:11:30 2010
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	decorrelate_processor
Top-level Entity Name	decorrelate_processor
Family	Stratix II
Met timing requirements	Yes
Logic utilization	6 %
Combinational ALUTs	438 / 12,480 (4 %)
Dedicated logic registers	434 / 12,480 (3 %)
Total registers	434
Total pins	93 / 343 (27 %)
Total virtual pins	0
Total block memory bits	4,150 / 419,328 (< 1 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S15F484C3
Timing Models	Final

Figure C.1.1 Compilation Report – Flow Summary for forward 5/3 module "decorrelate_processor".



PowerPlay Power Analyzer Status	Successful - Tue Apr 20 13:11:30 2010
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	decorrelate_processor
Top-level Entity Name	decorrelate_processor
Family	Stratix II
Device	EP2S15F484C3
Power Models	Final
Total Thermal Power Dissipation	500.46 mW
Core Dynamic Thermal Power Dissipation	80.96 mW
Core Static Thermal Power Dissipation	304.80 mW
I/O Thermal Power Dissipation	114.70 mW
Power Estimation Confidence	Medium: user provided moderately complete toggle rate data

Figure C.1.2 Compilation Report – Power Analyzer summary for forward 5/3 module "decorrelate_processor".

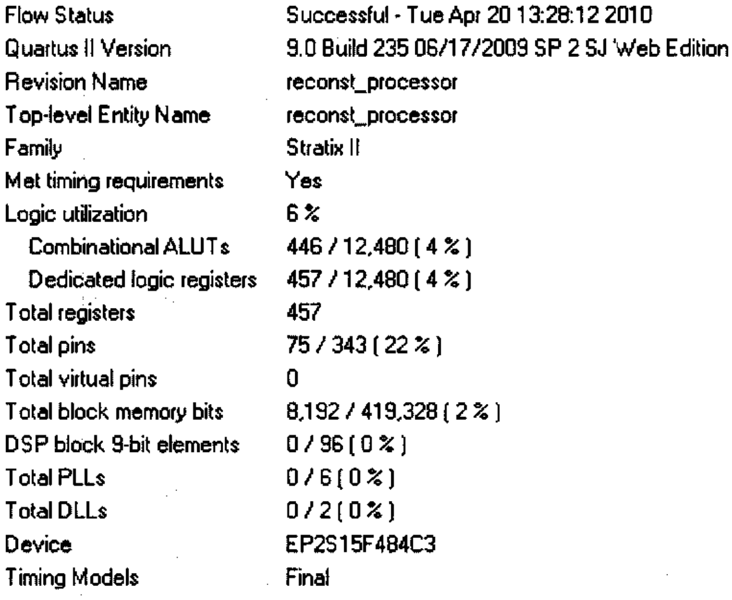

Quartus II - C:/tutorial/overlapp_architecture/decorrelate_processor - decorrelate_processor - [Compilation Report - Timing Analyzer Summary]
File Edit View Project Assignments Processing Tools Window Help
decorrelate_processor
Compilation Report - Timing Analyzer Summary

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To
1	Worst-case tsu	N/A	None	3.195 ns	Ed3	REd3
2	Worst-case tco	N/A	None	6.301 ns	L_data_out[8]~reg0	L_data_out[8]
3	Worst-case th	N/A	None	-1.831 ns	data_in0[10]	Rt0_1[10]
4	Clock Setup: 'clock'	N/A	None	185.74 MHz (period = 5.384 ns)	altsyncram:TLB_rtl_0|altsyncram_tpi1:auto_generated|ram_block1a0~portb_address_reg7	Rd[9]
5	Total number of failed paths					

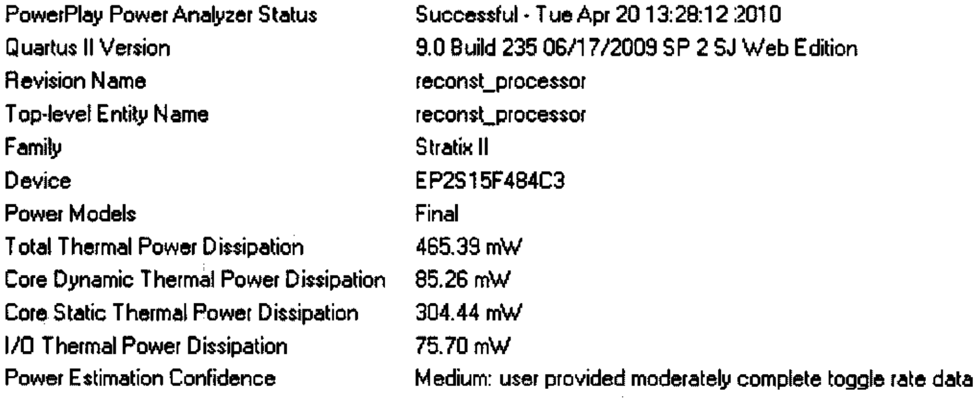
Figure C.1.3 Compilation Report – Timing Analyzer Summary for forward 5/3 module "decorrelate_processor"

C.2 Compilation reports for inverse 5/3 module "reconst_processor"



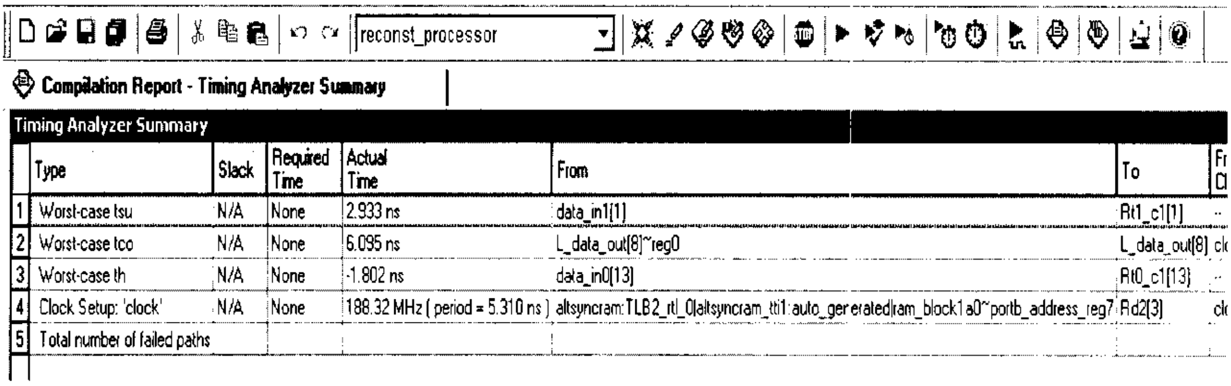
Flow Status	Successful - Tue Apr 20 13:28:12 2010
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	reconst_processor
Top-level Entity Name	reconst_processor
Family	Stratix II
Met timing requirements	Yes
Logic utilization	6 %
Combinational ALUTs	446 / 12,480 (4 %)
Dedicated logic registers	457 / 12,480 (4 %)
Total registers	457
Total pins	75 / 343 (22 %)
Total virtual pins	0
Total block memory bits	8,192 / 419,328 (2 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S15F484C3
Timing Models	Final

Figures C.2.1 Compilation Report – Flow Summary for inverse 5/3 module "reconst_processor"



PowerPlay Power Analyzer Status	Successful - Tue Apr 20 13:28:12 2010
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	reconst_processor
Top-level Entity Name	reconst_processor
Family	Stratix II
Device	EP2S15F484C3
Power Models	Final
Total Thermal Power Dissipation	465.39 mW
Core Dynamic Thermal Power Dissipation	85.26 mW
Core Static Thermal Power Dissipation	304.44 mW
I/O Thermal Power Dissipation	75.70 mW
Power Estimation Confidence	Medium: user provided moderately complete toggle rate data

Figure C.2.2 Compilation Report – Power Analyzer summary for inverse 5/3 module "reconst_processor".



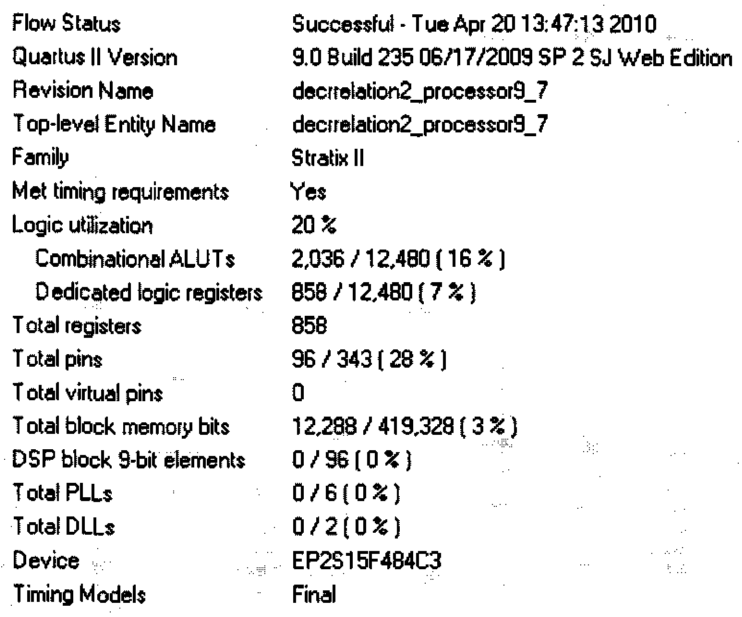
Compilation Report - Timing Analyzer Summary

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	Fr Cl
1	Worst-case tsu	N/A	None	2.933 ns	data_in1[1]	Rt1_c1[1]	--
2	Worst-case tco	N/A	None	6.095 ns	L_data_out[8]~reg0	L_data_out[8]	cl
3	Worst-case th	N/A	None	-1.802 ns	data_in0[13]	Rt0_c1[13]	--
4	Clock Setup: 'clock'	N/A	None	188.32 MHz (period = 5.310 ns)	altsyncram:TLB2_rtl_0|altsyncram_tti1:auto_generated|ram_block1a0~portb_address_reg7	Rd2[3]	cl
5	Total number of failed paths						

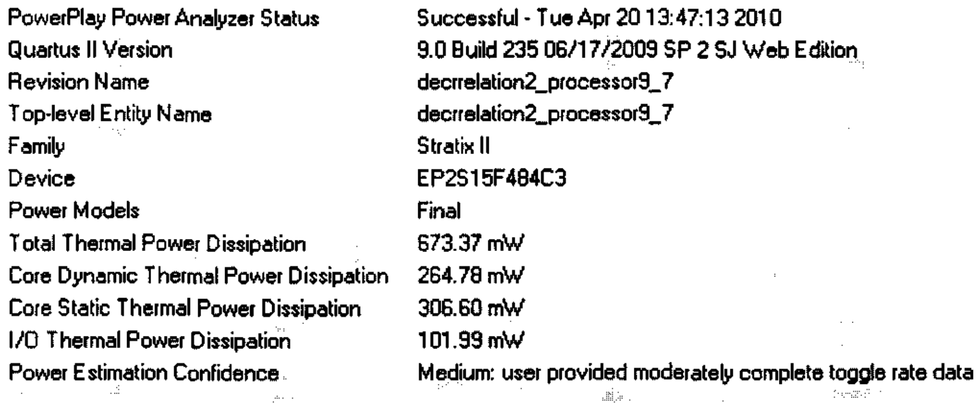
Fig. C.2.3 Compilation Report–Timing Analyzer Summary for inverse 5/3 module "reconst_processor"

C.3 Compilation reports for first forward 9/7 module "decorrelation2_processor9_7"



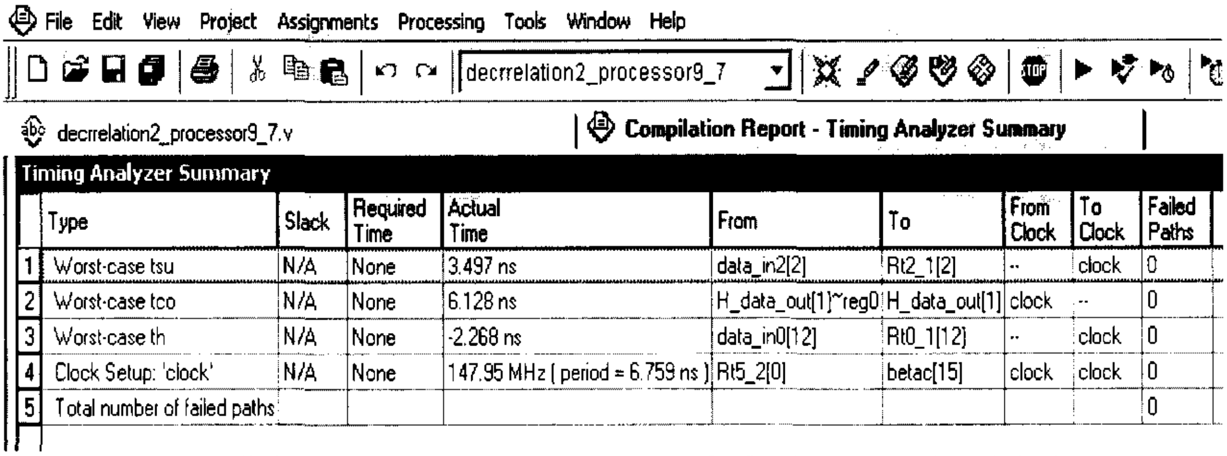
Flow Status	Successful - Tue Apr 20 13:47:13 2010
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	decrrelation2_processor9_7
Top-level Entity Name	decrrelation2_processor9_7
Family	Stratix II
Met timing requirements	Yes
Logic utilization	20 %
Combinational ALUTs	2,036 / 12,480 (16 %)
Dedicated logic registers	858 / 12,480 (7 %)
Total registers	858
Total pins	96 / 343 (28 %)
Total virtual pins	0
Total block memory bits	12,288 / 419,328 (3 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S15F484C3
Timing Models	Final

Fig. C.3.1Compilation Report – Flow Summary for first 9/7 module "decrrelation2_processor"



PowerPlay Power Analyzer Status	Successful - Tue Apr 20 13:47:13 2010
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	decrrelation2_processor9_7
Top-level Entity Name	decrrelation2_processor9_7
Family	Stratix II
Device	EP2S15F484C3
Power Models	Final
Total Thermal Power Dissipation	673.37 mW
Core Dynamic Thermal Power Dissipation	264.78 mW
Core Static Thermal Power Dissipation	306.60 mW
I/O Thermal Power Dissipation	101.99 mW
Power Estimation Confidence	Medium: user provided moderately complete toggle rate data

Figure C.3.2 Compilation Report – Power Analyzer summary for first 9/7 module "decrrelation2_processor".

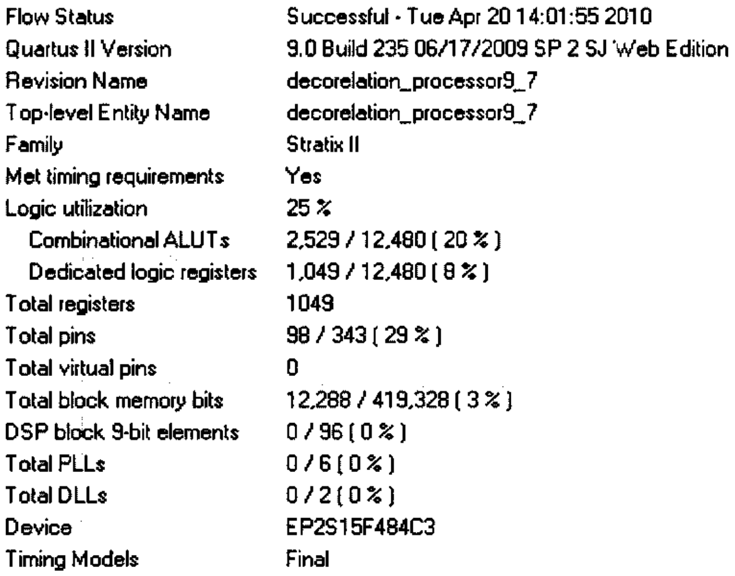

File Edit View Project Assignments Processing Tools Window Help
decrrelation2_processor9_7
decrrelation2_processor9_7.v
Compilation Report - Timing Analyzer Summary

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tsu	N/A	None	3.497 ns	data_in2[2]	Rt2_1[2]	--	clock	0
2	Worst-case tco	N/A	None	6.128 ns	H_data_out[1]~reg0	H_data_out[1]	clock	--	0
3	Worst-case th	N/A	None	-2.268 ns	data_in0[12]	Rt0_1[12]	--	clock	0
4	Clock Setup: 'clock'	N/A	None	147.95 MHz (period = 6.759 ns)	Rt5_2[0]	betac[15]	clock	clock	0
5	Total number of failed paths								0

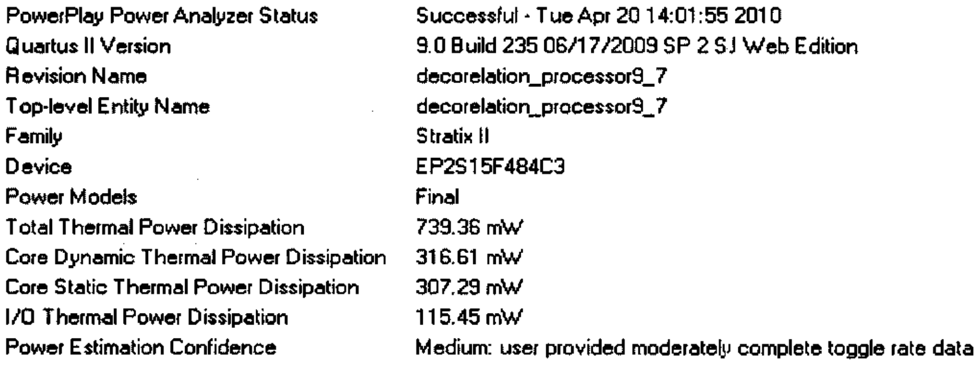
Figure C.3.3 Compilation Report – Timing Analyzer Summary for first 9/7 module "decrrelation2_processor"

C.4 Compilation reports for second forward 9/7 module "decorrelation_processor9_7"



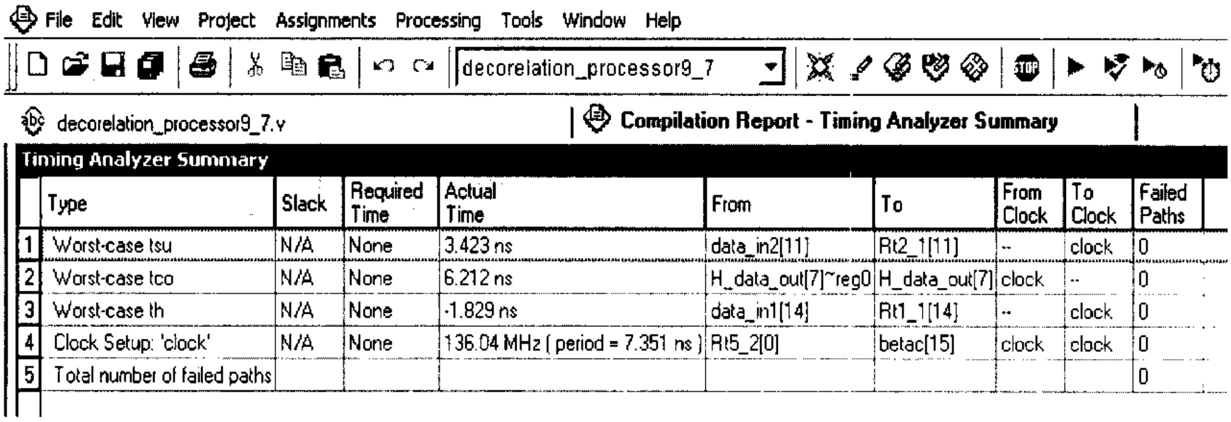
Flow Status	Successful - Tue Apr 20 14:01:55 2010
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	decorelation_processor9_7
Top-level Entity Name	decorelation_processor9_7
Family	Stratix II
Met timing requirements	Yes
Logic utilization	25 %
Combinational ALUTs	2,529 / 12,480 (20 %)
Dedicated logic registers	1,049 / 12,480 (8 %)
Total registers	1049
Total pins	98 / 343 (29 %)
Total virtual pins	0
Total block memory bits	12,288 / 419,328 (3 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S15F484C3
Timing Models	Final

Figure C.4.1 Compilation Report – Flow Summary for second 9/7 module "decorelation_processor"



PowerPlay Power Analyzer Status	Successful - Tue Apr 20 14:01:55 2010
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	decorelation_processor9_7
Top-level Entity Name	decorelation_processor9_7
Family	Stratix II
Device	EP2S15F484C3
Power Models	Final
Total Thermal Power Dissipation	739.36 mW
Core Dynamic Thermal Power Dissipation	316.61 mW
Core Static Thermal Power Dissipation	307.29 mW
I/O Thermal Power Dissipation	115.45 mW
Power Estimation Confidence	Medium: user provided moderately complete toggle rate data

Figure C.4.2 Compilation Report – Power Analyzer summary for second 9/7 module "decorelation_processor".



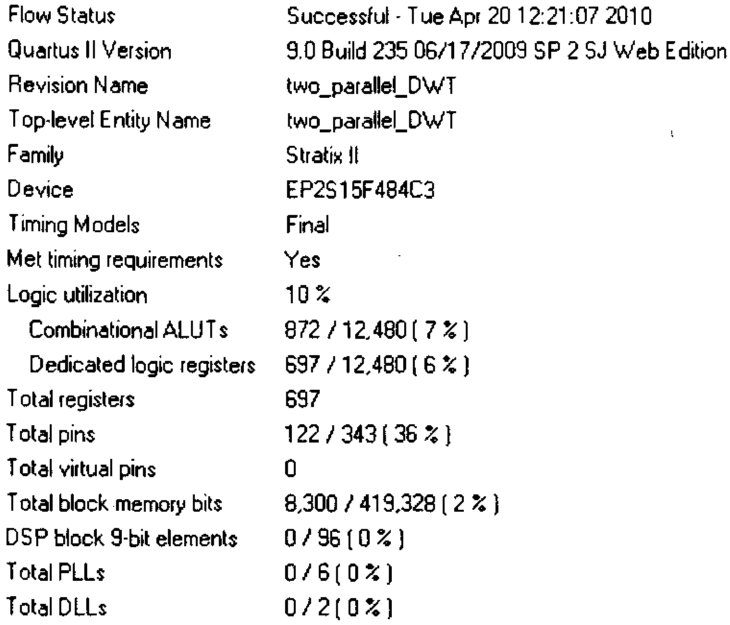
File Edit View Project Assignments Processing Tools Window Help
decorelation_processor9_7
decorelation_processor9_7.v
Compilation Report - Timing Analyzer Summary

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tsu	N/A	None	3.423 ns	data_in2[11]	Rt2_1[11]	--	clock	0
2	Worst-case tco	N/A	None	6.212 ns	H_data_out[7]~reg0	H_data_out[7]	clock	--	0
3	Worst-case th	N/A	None	-1.829 ns	data_in1[14]	Rt1_1[14]	--	clock	0
4	Clock Setup: 'clock'	N/A	None	136.04 MHz (period = 7.351 ns)	Rt5_2[0]	betac[15]	clock	clock	0
5	Total number of failed paths								0

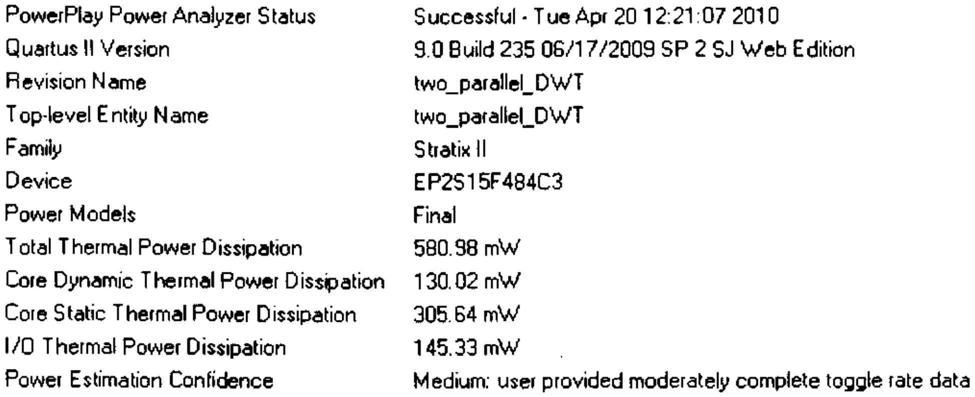
Fig. C.4.3 Compilation Report – Timing Analyzer Summary for second 9/7 module "decrrelation_processor"

C.5 Compilation reports for 5/3 2-parallel module “two_parallel_DWT7”



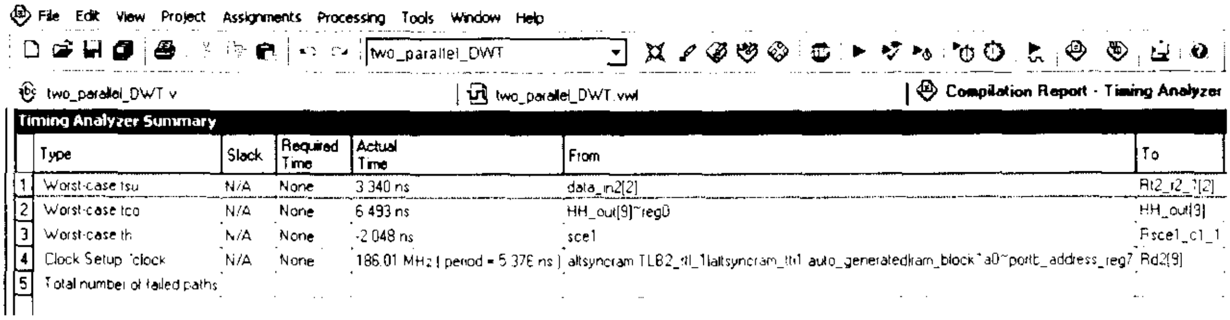
Flow Status	Successful - Tue Apr 20 12:21:07 2010
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	two_parallel_DWT
Top-level Entity Name	two_parallel_DWT
Family	Stratix II
Device	EP2S15F484C3
Timing Models	Final
Met timing requirements	Yes
Logic utilization	10 %
Combinational ALUTs	872 / 12,480 (7 %)
Dedicated logic registers	697 / 12,480 (6 %)
Total registers	697
Total pins	122 / 343 (36 %)
Total virtual pins	0
Total block memory bits	8,300 / 419,328 (2 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)

Figure C.5.1 Compilation Report – Flow Summary for 5/3 2-parallel module “two_parallel_DWT”



PowerPlay Power Analyzer Status	Successful - Tue Apr 20 12:21:07 2010
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	two_parallel_DWT
Top-level Entity Name	two_parallel_DWT
Family	Stratix II
Device	EP2S15F484C3
Power Models	Final
Total Thermal Power Dissipation	580.98 mW
Core Dynamic Thermal Power Dissipation	130.02 mW
Core Static Thermal Power Dissipation	305.64 mW
I/O Thermal Power Dissipation	145.33 mW
Power Estimation Confidence	Medium: user provided moderately complete toggle rate data

Figure C.5.2 Compilation Report – Power Analyzer summary for 5/3 2-parallel module “two_pararllel_DWT”.



File Edit View Project Assignments Processing Tools Window Help
two_parallel_DWT
two_parallel_DWT.v
two_parallel_DWT.vwf
Compilation Report - Timing Analyzer

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To
1	Worst-case tsu	N/A	None	3.340 ns	data_in2[2]	Rt2_r2_1[2]
2	Worst-case tco	N/A	None	6.493 ns	HH_out[9]~reg0	HH_out[9]
3	Worst-case th	N/A	None	-2.048 ns	sce1	Fsce1_c1_1
4	Clock Setup: 'clock'	N/A	None	186.01 MHz (period = 5.376 ns)	altsyncram:TLB2_rtl_1|altsyncram_tt1:auto_generated|ram_block1a0~portb_address_reg7	Rd2[9]
5	Total number of failed paths					

Fig. C.5.3 Compilation Report – Timing Analyzer Summary for 5/3 2-parallel module “decrrelation_processor”

APPENDIX D

PUBLICATIONS

Conference papers

[1] Ibrahim Saeed Koko and Herman Agustiawan, "Lifting-based VLSI architectures for 2- Dimensional discrete wavelet transform for Effective image compression," in: Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 Vol.1, Hong Kong, March 2008, PP. 339-347.

[2] Ibrahim Saeed Koko and Herman Agustiawan, "High-speed and power efficient lifting-based VLSI architectures for two-Dimensional discrete wavelet transform," proceedings of the IEEE Second Asia International Conference on Modeling and Simulation, AMS , May 2008, PP. 998-1005.

[3] Ibrahim Saeed Koko and Herman Agustiawan, "Pipelined lifting-based VLSI architecture for two-dimensional inverse discrete wavelet transform," proceedings of the IEEE International Conference on Computer and Electrical Engineering, ICCEE, December 2008, Phuket Island, Thailand, PP. 692-700.

[4] Ibrahim Saeed Koko and Herman Agustiawan, "Parallel Pipelined VLSI Architectures for Lifting-based Two-dimensional Forward Discrete Wavelet Transform," proceedings of the IEEE International Conference on signal acquisitions and processing, ICSAP, April 2009, Kuala Lumpur, Malaysia, PP. 18-25.

Journal papers

[5] Ibrahim Saeed and Herman Agustiawan, "Two-dimensional Discrete Wavelet Transform Memory Architectures," International Journal of Computer and Electrical Engineering, Vol. 1, No. 1, April 2009, PP 84-97.

[6] Ibrahim Saeed and Herman Agustiawan, "Parallel form of the Pipelined Lifting-based VLSI Architectures for Two-dimensional Discrete Wavelet Transform," International Journal of Computer Theory and Engineering, Vol. 1, No. 1, April 2009, PP 85-96.

[7] Ibrahim Saeed and Herman Agustiawan, "Parallel Form of the Pipelined Intermediate Architecture for 2-dimensional Discrete Wavelet Transform," IAENG International Journal of Computer Science, Vol. 36, issue 2, June 2009.

Book Chapter

[1] Ibrahim Saeed and Herman Agustiawan, "High Performance Parallel Pipelined Lifting-based VLSI Architectures for Two-Dimensional Inverse Discrete Wavelet Transform," book title: "VLSI", ISBN 978-3-902613-50-9, IN-TECH, Feb. 2010.