Architectures and Algorithms for Image and Video Processing using FPGA-based Platform

THESIS

Submitted in partial fulfillment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY
by

Jai Gopal Pandey

Under the Supervision of

Dr. Chandra Shekhar
CSIR-CEERI, Pilani

Co-supervision of
Dr. Abhijit Karmakar
CSIR-CEERI, Pilani

Co-supervision of
Prof. S. Gurunarayanan
BITS, Pilani

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING
BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
PILANI – 333031, INDIA

May, 2014

CERTIFICATE

This is to certify that the thesis entitled "**Architectures and Algorithms for Image and Video Processing using FPGA-based Platform"** being submitted by Mr. Jai Gopal Pandey, ID No. 2008PHXF018P to the Department of Electrical and Electronics Engineering, Birla Institute of Technology & Science (BITS), Pilani for the award of Doctor of Philosophy embodies original work done by him under our supervision.

Signature of the Supervisor

_____

Dr. Chandra Shekhar
Director, CSIR-CEERI, Pilani
Rajasthan-333031, India

Date:

Signature of the Co-supervisor

_____

Dr. Abhijit Karmakar
Principal Scientist
IC Design Group, CSIR-CEERI, Pilani
Rajasthan-333031, India
Date:

Signature of the Co-supervisor

_____

Dr. S. Gurunarayanan
Professor, Department of Electrical and Electronics Engineering
Dean, Work Integrated Learning Programmes Division
BITS Pilani, Pilani Campus
Rajasthan-333031, India
Date:

ABSTRACT

Driven by rapid technological advances and ever-increasing demand for new applications, system complexities have grown at almost an exponential rate. In this scenario, the traditional system design methods have rapidly become incapable of providing solutions that meet system requirements as, neither purely software-based nor purely hardware-based systems are able to meet the various expectations from the system solution. Modern-day complex systems inevitably necessitate inclusion of heterogeneous software and hardware components in the system. In particular, the rapid growth of image and video processing applications has created increasing demands for high-performance configurable hardware architectures and algorithms for building requisite electronic systems. To achieve high performance goals and fulfill the conflicting design needs of low power and easy system upgradeability, integrated hardware/software system are required.

Modern field-programmable gate arrays (FPGAs) have evolved to offers an embedded processor and many hard intellectual property (IP) components, required to create integrated hardware/software systems. The presence of logic blocks in FPGAs provides hardware configurability, whereas the embedded processor supports the programmability and the necessary control. With the advent of platform-based design methodology and associated integrated design tools, it is possible to utilize a variety of soft/hard IPs along with various off-the-shelf components available on the platform to build system solutions that meet required expectations.

In this work, architectures and algorithms have been proposed and developed for frequently used image and video processing applications. These architectures have been realized in the Xilinx Virtex-5 FX FPGA device available on the ML-507 platform. Apart from having sufficient logic blocks on which hardware is implemented this FPGA device also has an

embedded PowerPC 440 processor with requisite system software, to implement the application software around it. To start with, we have used the platform for developing an embedded architecture for real-time video capture, acquisition and its display for the standard 640×480 VGA resolution video through the FPGA fabric and for its frame-by-frame buffering in the external double-data rate synchronous dynamic random-access memory (DDR2 SDRAM).

We have then attempted to realize various complex arithmetic functions that are frequently required in image and video processing applications such as division, square root, inverse square root etc., through the use of logarithmic number system. For this purpose, a simple integer datapath has been created for processing 32-bit unsigned fixed-point numbers. Architectures for the binary logarithm and antilogarithm units are proposed that compute their approximate values within the specified range. These units have been utilized to realize the hardware architectures for various image processing functions that have been proposed in the thesis.

A novel hardware architecture for global image thresholding operation has been proposed that results in a resource-efficient FPGA implementation of the computation of between class variance (BCV) computation for realizing the Otsu's image thresholding algorithm. The compute-intensive BCV requires the computation of normalized cumulative histogram and normalized cumulative intensity area. The proposed architecture is logic resource efficient and has the ability to process large datasets by performing time-critical functions using available BRAMs and DSP slices.

We have next proposed an improved label-equivalence based connected component labeling algorithm that works on the binary images obtained from the image thresholding unit and identifies an object on a video frame. The proposed algorithm improves upon the

Stefano-Bulgarelli (SB) algorithm by modifying its equivalence handling procedure, and removes the partial merging problem associated with the SB algorithm. The improved algorithm is implemented on the embedded PowerPC processor of ML-507 platform. Results demonstrate that the improved algorithm handles equivalences efficiently and gives an accurate count of connected components.

Finally, all the hardware building blocks and algorithms described so far are utilized for an embedded implementation of a representative video processing application, e.g., object tracking based on kernel-based mean shift (KBMS) algorithm. The required application-specific architectural building blocks have been proposed for its embedded realization on Xilinx ML-507 platform. To understand issues related to the embedded realization of the KBMS algorithm, a MATLAB/C implementation is created. Subsequently, hardware architectures have been proposed for the time-critical parts, namely, the computations of weighted local histogram, kernel-smoothed local histogram (KSLH), Bhattacharyya coefficient based local similarity measure, center of gravity and the new mean shift location.

The embedded design also utilizes the soft IPs, which include, joint test action group (JTAG) controller, Block RAM (BRAM) controller, multi-port memory controller (MPMC), processor local bus (PLB), inter-integrated circuit (I2C) controller and the UART controller. The hard IPs utilized include the PowerPC 440 processor, BRAMs, digital clock manager (DCM) and DSP48E slices. The frame buffer part of the design is created in the available off-chip DDR2 SDRAM memory, which is controlled through the MPMC.

Embedded PowerPC processor has been used to configure and control various off-the-shelf system peripherals available on the platform along with the running of the application program. The application software, written in C language, runs on top of a standalone software platform and uses the application programmer interface (API) provided by the

software platform. In order to develop the required hardware and software in an integrated manner, the Xilinx embedded development kit (EDK) design tool has been used. To analyze the design in real-time, Xilinx ChipScope Pro integrated logic analyzer has been utilized. Xilinx XPower Analyzer tool has been used, for computing power consumption associated with different architectural modules.

In summary, the thesis explores and presents some of the concepts of emerging embedded system design techniques. It does so by way of identifying, building and integrating all the necessary hardware and software components for a real-time video processing application, namely object tracking (utilizing the kernel-based mean shift algorithm). The thesis also illustrates the use of platform-based design to achieve an efficiently configured hardware-software system solution that can meet the conflicting demands of high performance, low power and quick turnaround times for system development.

DEDICATION

This thesis is dedicated to all of my family members and friends.

ACKNOWLEDGEMENT

First, I would like to convey my deepest appreciation and thanks to my supervisor and mentor Dr. Chandra Shekhar, Director, Council of Scientific and Industrial Research (CSIR)- Central Electronics Engineering Research Institute (CEERI), Pilani, Rajasthan, India, for providing me the opportunity to work for the new grounds in architectural and algorithm exploration for image and video processing applications. His broad vision and multidisciplinary knowledge has enhanced the quality of this research work. I am very thankful to him for providing me his precious time out of his very active schedule.

My deepest gratitude is to my co-supervisor and adviser from CSIR-CEERI, Pilani, Dr. Abhijit Karmakar, Principal Scientist, IC Design Group, for his constant guidance, support and encouragement to carry out the directed work. His ideas and regular discussions have really helped me for the development of the thesis.

I would also like to thank my advisor and co-supervisor from Birla Institute of Technology and Science (BITS), Pilani, Dr. S. Gurunarayanan, Professor, Department of Electrical and Electronics (EEE) and Dean, Work Integrated Learning Programmes Division, BITS Pilani for his constant support, guidance and encouragement that have helped me a lot for carrying out this research work. I am grateful to him for devoting his valuable time to monitor the progress of the work and providing the required research directions.

My sincere gratitude also goes to the members of Doctoral Advisory Committee (DAC), Dr. Anu Gupta and Dr. Abhijit R. Asati from the department for going through the thesis and giving valuable inputs that have improved the thesis. I also take this opportunity to thank the members of Departmental Research Committee (DRC), Dept. of EEE and the Academic Research Division (ARD) for their valuable advises and for providing the required support.

I would also like to extend my thank to Sh. Raj Singh, Chief Scientist, IC Design Group, CSIR-CEERI, Pilani, for providing the necessary resources through the Department of Electronics and Information Technology (DeitY)/Ministry of Communications & Information Technology (MCIT), Government of India sponsored projects.

Many appreciations are also due to my colleagues, Mr. Sudhir Kumar, Dr. A. S. Mandal, Dr. S. C. Bose, Mr. Amit Kumar Mishra, Mr. A. K. Saini, Dr. Ravi Saini, Mr. Sanjay Singh, Mr. Sanjeev Kumar, Mr. Rajul Bansal and many others. Their constant support and encouragement always helped me to carry out the work.

I also wish to thank my family, especially, to my grandfather, father and mother. Their vision and inspiration motivated me to carry out this dissertation. Their blessings and encouragement contributed in many ways in my life. I am also very thankful to my brothers and sisters who regularly supported me for the work. My sincere thanks also go to my wife Richa and our son Jayesh. Their understanding, patience and valuable support have helped me at every stage of the thesis work.

Jai Gopal Pandey

_____

Place: Pilani

Date:

TABLE OF CONTENTS

Abstract ii
Dedication vi
Acknowledgement vii
Table of Contents ix
List of Figures xv
List of Tables xxi
List of Symbols xxiii
List of Abbreviations xxv

CHAPTER 1 Introduction 1
1.1 Background and Context 1
1.2 Motivation for the Work 6
1.3 Platform-Based Design Approach 9
1.3.1 FPGA-Based Embedded Vision Platforms 14
1.3.2 Platform Design Tools 16
1.4 Scope and Objectives 17
1.5 Proposed Hardware/Software Modules for an Embedded Video Processing Application 22
1.5.1 Platform Configuration 23
1.5.2 Video Acquisition 24
1.5.3 Arithmetic Datapath 25
1.5.4 Logarithm and Antilogarithm Units 25
1.5.5 Hardware Architecture of Global Thresholding 26
1.5.6 PowerPC Realization of Connected Component Analysis 26
1.5.7 Embedded Realization of Kernel-based Mean Shift Algorithm 27
1.6 Major Contributions and Organization of the Thesis 29
CHAPTER 2 Real-Time Video Streaming, Acquisition and Display using FPGA-based Platform 34

2.1 Introduction 34
2.2 Xilinx ML-507 Platform Configuration 38
2.2.1 Configuration of VGA Input Video Codec 39
2.2.2 Display Controller Configuration 42
2.3 Embedded Video Streaming Module 45
2.3.1 Hardware Components of the Video Streaming Module 48
2.4 Embedded Video Acquisition and Real-Time Display 52
2.4.1 The System Architecture 53
2.4.2 System Validation 60
2.5 Results 60
2.6 Conclusion 63
CHAPTER 3 Hardware Realizations of Logarithm and Antilogarithm Functions 64
3.1 Introduction 64
3.2 Approximation Methods for Computing Binary Logarithm and Antilogarithm 67
3.2.1 Binary Logarithmic Approximation Method 68
3.2.2 Approximation Method for Antilogarithm Computation 69
3.3 Fixed-Point Number Formats for the Proposed Architectures 70
3.4 Binary Logarithmic Approximation Circuit and the Proposed Architecture 71
3.4.1 The Proposed Architecture 72
3.4.2 Leading-One Finder (LOF) Circuit 74
3.4.3 The Barrel Shifter (BSHFT) Unit 79
3.4.4 Fractional Part Approximation (FPA) Unit for Logarithm Computation 81
3.4.5 Error Analysis of Logarithmic Approximation 82
3.5 FPGA Implementation of Binary Logarithm Unit 84
3.6 Binary Antilogarithm Approximation Unit and its Proposed Architecture 86
3.6.1 Architectural Building Blocks 87

3.6.2 Error Analysis of the Binary Antilogarithm Approximation 92

3.7 FPGA Implementation Results of the Binary Antilogarithm Unit 93

3.8 Conclusion 94

CHAPTER 4 Architecture and Hardware Realization of an Image Thresholding Algorithm 96

4.1 Introduction 96

4.2 RGB to Gray Conversion 98

4.3 Otsu's Automatic Threshold Selection Method 99

4.4 Hardware Implementation Issues Related to Otsu's Algorithm 102

4.5 The Proposed Architecture for Otsu's Algorithm 105

4.5.1 Fixed-Point Number Format 108

4.5.2 Normalized Cumulative Histogram (NCH) Computation 108

4.5.3 Normalized Cumulative Intensity Area (NCIA) and Total Mean Computation 113

4.5.4 Binary Logarithmic Between Class Variance (LOGBCV) Computation Unit 114

4.5.5 MAX Circuit 117

4.6 Results and Discussion 117

4.7 Thresholding Unit as an IP Core and the Required System-Level Arrangement 118

4.8 Conclusion 122

CHAPTER 5 Connected Component Labeling Algorithm and its PowerPC Implementation 123

5.1 Introduction 123

5.2 Two-scan Connected Component Label-Equivalence Process 126

5.2.1 Basic Terminology 126

5.2.2 Pixel-based Conventional Two-Scan Label-Equivalence Algorithms 128

5.3 The Stefano-Bulgarelli's Algorithm 129

5.4 Improved SB Algorithm 133

5.5 Comparative Analysis of the Improved SB Algorithm 136
5.5.1 Results for Specialized Artificial Binary Test Patterns 137
5.5.2 Results for Standard Images 140
5.6 Embedded PowerPC Implementation of the Improved SB Algorithm 143
5.7 Conclusion 146
CHAPTER 6 Embedded Implementation of Kernel-based Mean Shift Object Tracking Algorithm 147
6.1 Introduction 147
6.2 Kernel-based Mean Shift (KBMS) Object Tracking 152
6.2.1 Mean Shift Clustering 153
6.2.2 Target Representation 154
6.2.3 Target Model 155
6.2.4 Target Candidates 156
6.2.5 Kernel Profile 157
6.2.6 Bhattacharyya Coefficeint based Distance Metric 158
6.2.7 Distance Minimization and the Mean Shift Weight 159
6.3 The KBMS Tracking Algorithm Flow 160
6.4 MATLAB/C Implementation of the KBMS Tracking Algorithm 161
6.5 Embedded Implementation of the KBMS Tracking Algorithm 163
6.6 Kernel-Smoothed Local Histogram Computation 167
6.6.1 Color-space Quantization (m-Bins) and Color Histogram 168
6.6.2 Kernel Weight Computation 169
6.6.3 Normalization unit 171
6.6.4 Weighted Local Histogram Computation 172
6.7 Bhattacharyya Coefficient Computation 174
6.8 Mean Shift Weight Computational Unit 176

6.9 New Mean Shift Location Computation 178
6.10 Integration of Architectural Building Blocks 181
6.11 The System Control 183
6.12 Results and Discussions 183
6.12.1 FPGA Device Utilization for the KSLH Module 184
6.12.2 FPGA Device Utilization for the Bhattacharyya Coefficient Computation 184
6.12.3 FPGA Device Utilization for the Mean Shift Weight Computation 186
6.12.4 FPGA Device Utilization for the New Mean Shift Location Computation Unit 187
6.12.5 FPGA Device Utilization for the KBMS Unit 188
6.13 The Complete System View for implementation of KBMS Algorithm 190
6.14 Conclusion 191
CHAPTER 7 Conclusions 193
7.1 Summary of Achievements 193
7.2 Future Scope of Work 198
References 199
List of Publications 211
Brief Biography of the Candidate 213
Brief Biography of the Supervisors 214
APPENDIX A An Overview of the FPGA-based Platform A-1
A.1 Xilinx ML-507 FPGA Platform A-1
A.2 Field-Programmable Gate Array (FPGA) Device A-3
A.2.1 Configuration Logic Block (CLB) A-5
A.2.2 Slice Description A-6
A.2.3 Interconnect A-9
A.2.4 Select I/O A-9
A.2.5 Special-Purpose Function Blocks A-10

A.3 FPGA Configuration Options A-12
A.3.1 JTAG (Xilinx Download Cable and System ACE Controller) Configuration A-13
A.3.2 Platform Flash PROM Configuration A-14
A.3.3 Linear Flash Memory Configuration A-14
A.3.4 SPI Flash Memory Configuration A-14
A.4 PowerPC 440 Embedded Processor A-14
A.4.1 Crossbar and its Interfaces A-16
A.4.2 PLB Interface A-16
A.4.3 PLB Interconnection Techniques A-17
A.5 Memory Controller Interface (MCI) A-19
A.6 Other Embedded Processor Blocks A-20
A.7 Controllers A-20
A.7.1 Auxiliary Processing Unit (APU) A-20
A.7.2 DMA Controller A-20
APPENDIX B Xilinx ML-507 Platform Configuration for Embedded Vsion Application. B-1
B.1 Introduction B-1
B.2 Pan-Tilt-Zoom (PTZ) Video Camera B-1
B.3 PAL to VGA Converter B-2
B.4 Bus Protocols B-3
B.4.1 VGA Protocol B-3
B.4.2 Inter-Integrated Circuit (I2C) Bus Protocol B-5
B.5 Platform Set-up for the Embedded Vision Applications B-6
B.5.1 Programming the IDT Clock Generator B-6
B.5.2 VGA Input Video Codec B-9
B.5.3 Chrontel CH7301C Display Controller B-14

LIST OF FIGURES

Fig. 1.1: Classification of different resources in a typical image processing system. 5

Fig. 1.2: A general-purpose FPGA-based platform. 10

Fig. 1.3: The Xilinx ML-507 platform. 13

Fig. 1.4: Block diagram for realizing the object tracking algorithm. 21

Fig. 1.5: A generic image and video processing system. 23

Fig. 1.6: Complete system arrangement for realizing an object tracking algorithm. 28

Fig. 2.1: AD9980 with an FPGA 40

Fig. 2.2: Configuration of control registers of the video input video codec using I2C API (XIic_DynSend). 41

Fig. 2.3: CH7301C interface with the FPGA device. 43

Fig. 2.4: The Xilinx ML-507 FPGA platform as an embedded vision platform. 46

Fig. 2.5: Hardware blocks for the real-time video streaming. 46

Fig. 2.6: Block-diagram of the embedded video streaming module in Xilinx EDK. 47

Fig. 2.7: A frame of size captured video from the video camera. 48

Fig. 2.8: Development platform set-up for embedded video acquisition. 53

Fig. 2.9: System architecture for video acquisition. 54

Fig. 2.10: A video frame in the DDR2 SDRAM. 55

Fig. 2.11: System assembly view of the video acquisition design in Xilinx EDK. 56

Fig. 2.12: EDK block diagram view of the video acquisition design. 57

Fig. 2.13: EDK graphical view of the video acquisition design. 58

Fig. 2.14: System arrangement to validate the real-time video acquisition. 60

Fig. 2.15: Timing details of the video acquisition design obtained from Xilinx ChipScope Pro analyzer. 61

Fig. 2.16: Complete system set-up for embedded video acquisition. 61

Fig. 2.17: Total device utilized in the video acquisition design. 62

Fig. 2.18: FPGA slice utilization of each module for the embedded realization of video acquisition. 63

Fig. 3.1: Time consumption of arithmetic operations in a 3D graphics processor. Adapted from [47]. 64
Fig. 3.2: A simple arithmetic approach for realizing complex arithmetic functions. 65
Fig. 3.3: Fixed-point number format for the binary logarithm computation. 70
Fig. 3.4: Fixed-point number format for the binary antilogarithm computation. 70
Fig. 3.5: Binary logarithmic computation scheme. 72
Fig. 3.6: Proposed architecture for the binary logarithmic computation. 73
Fig. 3.7: Serial evaluation of the leading-one bit. 74
Fig. 3.8: Parallel/ serial evaluation of the leading-one bit. 75
Fig. 3.9: 4-bit leading-one finder (LOF4). 75
Fig. 3.10: Detailed circuit of a 16-bit leading-one finder (LOF16). 76
Fig. 3.11: Block diagram of the LOF16. 77
Fig. 3.12: Barrel shifter circuit (BSHFT) used in the binary logarithm computation unit. 79
Fig. 3.13: Fractional part approximation (FPA) unit for the binary logarithm computation. 81
Fig. 3.14: (a) Computed logarithms for 16.16 fixed-point numbers (b) associated percentage error in computation. 83
Fig. 3.15: Computed logarithms for the fractional numbers (b) associated percentage error in the computation. 83
Fig. 3.16: FPGA-based technology schematic for the proposed architecture of the binary logarithm computation unit. 84
Fig. 3.17: Power analysis of logarithm computation architecture. 85
Fig. 3.18: Block diagram of the binary antilogarithm computational unit. 87
Fig. 3.19: Fractional part approximation (FPA) unit for binary antilogarithm computation. 89
Fig. 3.20: Barrel shifter (BSHFT) unit for the binary antilogarithm computation. 90
Fig. 3.21: Percentage computational error (a) for positive input binary numbers (b) for the negative input binary numbers. 92
Fig. 3.22: FPGA-based technology schematic for the implemented binary antilogarithm computational unit. 93

Fig. 4.1: Block diagram for computing optimum threshold value using Otsu's algorithm. 102

Fig. 4.2: Direct implementation of Otsu's algorithm in hardware. 103

Fig. 4.3: Detailed structure of the proposed architecture for computing Otsu's algorithm. 105

Fig. 4.4: Block diagram of the proposed architecture for computing Otsu's algorithm. 107

Fig. 4.5: 32-bit fixed-point number format. 108

Fig. 4.6: BRAM read-write mode (a) read-first mode (b) write-first mode. 109

Fig. 4.7: Normalized cumulative histogram (NCH) computation block. 110

Fig. 4.8: Normalized cumulative histogram (NCH) computation timing diagram. 111

Fig. 4.9: ModelSim capture of normalized cumulative histogram (NCH). 112

Fig. 4.10: Normalized cumulative intensity area (NCIA) total mean computational block. 113

Fig. 4.11: LOGBCV computation. 114

Fig. 4.12: Logarithmic conversion unit with leading-one finder and fractional part approximation units. 115

Fig. 4.13: 16-bit Leading-one finder (LOF16). 116

Fig. 4.14: The fractional part approximation (FPA) unit of binary logarithm computation. 116

Fig. 4.15: Device utilization summary for the implementation of the thresholding architecture in the FPGA. 118

Fig. 4.16: System arrangement with the threshold computational unit. 119

Fig. 4.17: The native port interface (NPI) protocol. 120

Fig. 5.1: Pixel connectivity (a) 4-connectivity (b) 8-connectivity. 128

Fig. 5.2: Processing of equivalences in the first scan. 129

Fig. 5.3: C-Code for two-scan Stefano-Bulgarelli's (SB) algorithm. 130

Fig. 5.4: Two-scan labeling in the SB algorithm (a) artificial binary pattern (b) provisional labeling. 131

Fig. 5.5: Two-scan labeling algorithm results for (a) SB Algorithm (b) The improved SB algorithm. 133

Fig. 5.6: C-code for the improved SB algorithm. 134

Fig. 5.7: Number of connected components (#CC) and equivalence class for different artificial binary test patterns in Stefano-Bulgarelli's (SB) and in improved SB algorithm (a) First artificial binary test pattern (b) #CC identified by SB algorithm (c) #CC identified by improved SB algorithm (d) Second artificial binary test pattern (e) #CC identified by SB algorithm (f) #CC identified by improved SB algorithm (g) Third artificial binary test pattern (h) #CC identified by SB algorithm (i) #CC identified by improved SB algorithm.....138

Fig. 5.8: Number of connected components (#CC) and equivalence class for different artificial binary patterns in Stefano-Bulgarelli's (SB) and in the improved SB algorithm (a) Fourth artificial binary test pattern (b) #CC identified by SB algorithm (c) #CC identified by improved SB algorithm (d) Fifth artificial binary test pattern (e) #CC identified by SB algorithm (f) #CC identified by improved SB algorithm (g) Sixth artificial binary test pattern (h) #CC identified by SB algorithm (i) #CC identified by improved SB algorithm.....139

Fig. 5.9: PowerPC running the connected component analysis in an embedded environment.....143

Fig. 5.10: Generated linker script for the connected component analysis algorithm.....144

Fig. 5.11: Board support package settings for the connected component analysis algorithm.....145

Fig. 5.12: Execution of connected component analysis program on PowerPC 440 processor. 145

Fig. 6.1: Classification of object tracking methods.....148

Fig. 6.2: Pictorial representation of mean shift clustering. 154

Fig. 6.3: Epanechnikov kernel profile.....157

Fig. 6.4: C implementation of the KBMS algorithm. (a) Frame No.=12 (b) Frame No.=25 (c) Frame No.=32 (d) Frame No.=38 (e) Frame No.=42 (f) Frame No.=50 (g) Frame No.=55 (h) Frame No.=57.....162

Fig. 6.5: Embedded system arrangement for the mean shift object tracking. 164

Fig. 6.6: Complete hardware/software arrangement for realizing the object tracking algorithm. 165

Fig. 6.7: Complete hardware architectural units for KBMS algorithm.....166

Fig. 6.8: Architecture for computing kernel-smoothed local histogram. 167

Fig. 6.9: RGB color-space quantization into *m*-bins (a) R=0-15, G=0-255, B=0-255 (b) R=16-31, G=0-255, B=0-255 (c) R=240-255, G=0-255, B=0-255. 169

Fig. 6.10: A pictorial view of the kernel weights for the Epanechnikov kernel profile. 170
Fig. 6.11: Architecture for computing kernel weights. 171
Fig. 6.12: Weighted local histogram computation timing diagram. 173
Fig. 6.13: Architecture for computing the kernel-smoothed local histogram of an image. 173
Fig. 6.14: Architecture for computing the Bhattacharyya coefficient. 175
Fig. 6.15: Architecture for computing mean shift weights. 177
Fig. 6.16: Architecture for the new mean shift location computation. 180
Fig. 6.17: Integration of architectural building blocks for realizing the KBMS algorithm. 182
Fig. 6.18: FPGA technology schematic of Bhattacharyya coefficient computational unit. 185
Fig. 6.19: Synthesized view of the mean shift weight computation module. 187
Fig. 6.20: Synthesized view of the new mean shift location computation. 188
Fig. 6.21: Synthesized view of the complete KBMS unit. 189
Fig. 6.22: KBMS core in a system environment. 191
Fig. A.1: Xilinx ML-507 Platform. (a) front view (b) rear view. A-2
Fig. A.2: FPGA block structure (reproduced from embedded processor block in Virtex-5 FPGAs). A-4
Fig. A.3: Arrangement of slices within the CLB. A-5
Fig. A.4: Arrangement of SLICEL. A-7
Fig. A.5: Arrangement of SLICEM. A-8
Fig. A.6: Arrangement of IOB. A-9
Fig. A.7: Modern FPGA device. A-10
Fig. A.8: FPGA configurations. A-13
Fig. A.9: Block diagram of an embedded PowerPC 440 processor (reproduced from Xilinx UG200). A-15
Fig. A.10: Embedded processor block in Virtex-5 FPGAs (reproduced from Xilinx UG200). A-16
Fig. A.11: Simple processor-centric shared bus design (reproduced from Xilinx UG200). A-18
Fig. A.12: Simple processor-centric design using memory controller based main memory (reproduced from Xilinx UG200). A-19

Fig. B.1: Sony PTZ camera (a) front view (b) rear view (reproduced from Sony EVI-D70 PTZ camera). B-2

Fig. B.2: V2V Pro PAL to VGA converter (reproduced from MyGica V2V Pro). B-3

Fig. B.3: A 640×480 VGA resolution frame. B-4

Fig. B.4: I2C bus protocol. B-5

Fig. B.5: (a) IDT programmable clock structure (b) IDT programmable clock register settings. B-7

Fig. B.6: IDT5V9885 JTAG connector B-8

Fig. B.7: SVF output in Xilinx *iMPACT* B-9

Fig. B.8: AD9980 functional block diagram (reproduced from AD9980). B-10

Fig. B.9: CH7301C functional block diagram (reproduced from CH7301 DVI transmitter). B-15

LIST OF TABLES

Table 1.1: FPGA-based platforms 15

Table 2.1: VGA Timings for Resolution Video 39

Table 2.2: I/O Connection of VGA Input Video Codec with FPGA 40

Table 2.3: Control Registers of AD9980 41

Table 2.4: FPGA Interface Pins of AD9980 42

Table 2.5: Control Registers Value of Chrontel CH7301C Device 43

Table 2.6: CH7301C Chrontel Device Signals 44

Table 2.7: CH7301C Interface with the FPGA 44

Table 2.8: VGA_IN I/O Signals 49

Table 2.9: DE_GEN I/O Signals 50

Table 2.10: VGA_OUT I/O Signals 51

Table 3.1: Complex Arithmetic Operations using Logarithmic Number System 66

Table 3.2: Leading-One Finder (LOF16) Encoder 78

Table 3.3: Truth Table for Realizing the Barrel Shifter 80

Table 3.4: ROM Contents for the Binary Logarithm Computation 82

Table 3.5: FPGA Device Utilization for the Binary Logarithm Computation 85

Table 3.6: ROM Contents for the Antilogarithmic Computation 88

Table 3.7: BSHFT Data Routing Operation for the Binary Antilogarithm Computation 91

Table 3.8: FPGA Device Utilization for the Binary Antilogarithmic Computation 94

Table 4.1: FPGA Device Utilization for the Proposed Architecture for Threshold Computation 117

Table 4.2: Native Port Interface (NPI) Signals 121

Table 5.1: Processing of Equivalence Classes as in the SB Algorithm. 132

Table 5.2: Processing of Equivalence Classes in the Improved SB Algorithm. 136

Table 5.3: Comparison Between Different Labels Assigned and the Number of Connected Components (#CC) Detected for Artificial Binary Test Patterns. 137

Table 5.4: Comparison between the Numbers of Connected Components (#CC) Identified by the SB Algorithm and by the Improved SB Algorithm for Standard Images. 141

Table 5.5: Comparison Between Numbers of Conflicts Handled (#CH) by the SB Algorithm and by the Improved SB Algorithm with Standard Images. 142

Table 6.1: FPGA Device Utilization for Implementing the Proposed Architecture for Computing Kernel-Smoothed Local Histogram of an Image. 184

Table 6.2: FPGA Device Utilization for Implementing the Proposed architecture for Bhattacharyya Coefficient. 185

Table 6.3: FPGA Device Utilization for Implementing the Mean Shift Weight Computational Architecture. 186

Table 6.4: FPGA Device Utilization for Implementing the Proposed Architecture for New Mean Shift Location Computation. 187

Table 6.5: FPGA Device Utilization of Implementing the Complete KBMS Algorithm. 188

Table 6.6: FPGA Device Utilization Summary for Implementing Various Units of KBMS Algorithm. 190

Table B.1: VCO Range and Charge Pump and Current Settings B-11

LIST OF SYMBOLS

Symbol	Description
$\omega_0(k)$	Probability of background pixels class occurrences
$\omega_1(k)$	Probability of foreground pixels class occurrences
$\mu_0(k)$	Class mean of background pixels
$\mu_1(k)$	Class mean of foreground pixels
μ_T	Total mean-level
$\sigma_0^2(k)$	Class variance of background pixels
$\sigma_1^2(k)$	Class variance of foreground pixels
$\sigma_w^2(k)$	Within-class variance
$\sigma_B^2(k)$	Between-class variance for at threshold (k)
k^*	Optimal threshold value in Otsu's algorithm
$\omega(k)$	Zeroth-order cumulative moment
$\mu(k)$	First-order cumulative moment
$\sigma_B^2(k^*)$	Between-class variance at optimum threshold (k^*)
CC_n	Connected component (n^{th})
$C[n]$	Class of n^{th} label
δ	Kronecker delta function
$\{\boldsymbol{x}_i^*\}_{i=1\ldots n}$	Normalized pixel locations in the region
$k(x)$	Epanechnikov kernel profile
kw	Epanechnikov kernel weight
d_i	Distance of the pixel from center of the target
$\hat{\mathbf{q}}$	Target model
$\hat{\mathbf{p}}(\mathbf{y})$	Target candidate
$d(\mathbf{y})$	Bhattacharyya distance

ρ	Bhattacharya coefficient
w_i	Mean Shift weights
$\hat{\mathbf{y}}_0$	Current mean shift location
$\hat{\mathbf{y}}_1$	New mean shift location

LIST OF ABBREVIATIONS

API : Application Programmer Interface

APU : Auxiliary Processing Unit

BBD : Block-Based Design

BC : Bhattacharyya Coefficient

BCV : Between-Class Variance

BSB : Base System Builder

BSHFT : Barrel-Shifter

BSP : Board-Support Package

CC : Connected-Component

CCA : Connected-Component Analysis

CH : Conflict Handling

CLB : Configurable Logic Block

CODEC : Coder-Decoder

CoG : Center of Gravity

CoM : Center of Mass

CLK : Clock

CMOS : Complementary Metal-Oxide Semiconductor

CSQ : Color Space Quantization

D2M : Data2MEM

DC : Display Controller

DCM : Digital Clock Manager

DCR : Device Control Registers

DDR2 : Double Data Rate-2

DIP : Dual in-Line

DIV : Division

DMA : Direct Memory Access

DSP : Digital Signal Processing

DVI : Digital Visiual Interface

EDA : Electronic Design Automation

EDK : Embedded Development Kit

EOS : Embedded Operating System

EMIF : External Memory Interface

ESL : Electronic System Level

FCB : Fabric Coprocessor Bus

FPA : Fractional Part Approximation

FPGA : Field Programmable Gate Array

FPU : Floating Point Unit

HDL : Hardware Description Language

HIPR : Hypermedia Image Processing Reference

HSYNC : Horizontal Sync

HW : Hardware

I2C : Inter-Integrated Circuit

IC : Integrated Circuit

IDE : Integrated Design Environment

IDT : Integrated Device Technology

IEEE : Institute of Electrical and Electronics Engineers

IP : Intellectual Property

I/O : Input/Output

ISS : Instruction-Set Simulator

JTAG : Joint Test Action Group

KBMS : Kernel Based Mean Shift

KSLH : Kernel-Smoothed Local Histogram

KWC : Kernel Weight Computation

LNS : Logarithmic Number System

LOGBCV : Binary Logarithmic Between-Class Variance

LOF : Leading-One Finder

LUT : Look-Up Table

MCI : Memory Controller Interface

MMU : Memory Management Unit

MPLB : Master PLB

MPMC : Multi-Port Memory Controller

MSP : MicroBlaze Soft Processor

MSV : Mean-Shift Vector

MSWC : Mean Shift Weight Computation

MUX : Multiplexer

NCH : Normalized Cumulative Histogram

NCIA : Normalized Cumulative Intensity Area

NMSLC : New Mean Shift Location Computation

NPI : Native Port Interface

OS : Operating System

PAL : Phase Alternating Line

PBD : Platform-Based Design

PDF : Probability Density Function

PIP : Processing IP

PLB : Processor Local Bus

PO : Platform Object

PPC440 : PowerPC440 Processor

PTZ : Pan-Tilt-Zoom

PWR : Powering

RAM : Random Access Memory

RICP : Reciprocal

ROACH : Reconfigurable Open Architecture Computing Hardware

ROI : Region-Of-Interest

ROM : Read-Only Memory

RSQR : Reciprocal Square Root

RTL : Register-Transfer Level

RTOS : Real-Time Operating System

SB : Stefano-Bulgarelli

SCL : Serial Data Line

SDA : Serial Data Line

SDK : Software Development Kit

SDRAM : Synchronous Dynamic RAM

SIDBA : Standard Image Database

SODIMM : Small Outline Dual In-line Memory Module

SPLB : Slave PLB

SQR : Square

SQRT : Square Root

SW : Software

TDD : Timing-Driven Design

TLB : Translation Look-aside Buffer

UART : Universal Asynchronous Receiver-Transmitter

USC-SIPI : University of Southern California Signal & Image Processing Institute

VBS : Video Burst Sync

VC : Virtual Component

VDEC : Video Decoder

VFBC : Video Frame Buffer Controller

VGA : Video Graphics Array

VHDL : Very high speed integrated circuit Hardware Description Language

VLSI : Very Large Scale Integration

VSYNC : Vertical Sync

WLH : Weighted Local Histogram

WS : Weighted Sum

XPS : Xilinx Platform Studio

CHAPTER 1

INTRODUCTION

1.1 Background and Context

Image and video processing is used in a wide variety of applications such as video conferencing, video broadcasting and motion estimation [**1**], military aerial and satellite surveillance [**2**,**3**], biometric recognition [**4**], object tracking [**5**] and medical imaging applications [**6**,**7**]. The goal of real-time image and video processing system is to process the captured video, extract specific information and take appropriate action as needed [**8**]. The general structure of any image and video processing system consists of data acquisition, computation, communication, storage and display elements [**9**]. The data acquisition part performs the image data capture process. The captured data needs intermediate storage elements for its later processing as per the requirements. The computation unit does the required processing and communicates with all the other units via dedicated communication channels.

With the increasing popularity of multimedia, image processing and other vision-based applications, there is always a demand for high-resolution data processing elements. High-resolution standard image and video have large data, which needs processing within stipulated frame of time. With the increasing processing requirements of large data, the system complexity has also increased. Even though the recent computers are getting faster and faster, invariably, there is an emerging demand for even faster data processing mechanism. Modern image and video processing applications demand more specialized processing than is normally available in computers.

With high computational complexity of the modern algorithms, current software-based image and video processing systems are unable to meet the performance requirement in real-time. They do not achieve the required high performance that is required while working with available video frame rates. The effectiveness of real-time processing is primarily based on the idea of completing the required tasks in the time available between successive input frames of the incoming video, also known as sampling rate of frames or frame rate. Further, the image and video processing systems have ever-increasing demand for higher performance, lower power requirement and flexibility. These systems also need to process and manage a large amount of data within the constraint of real-time performance [**2**]. Therefore, quite often, dedicated embedded systems along with their architectures are required to be designed. Architectures for these image and video processing algorithms need to manage a large amount of data within real-time constraints; and parallelism is a fundamental requirement for most of these systems. Thus, their design as embedded system continues to be a challenging problem.

The dominant approaches that are used to implement complex image and video processing algorithms are using digital signal processors (DSP), application specific integrated circuits (ASICs), application-specific instruction-set processors (ASIPs), and field programmable gate arrays (FPGAs) [**10**,**11**,**12**]. The DSPs are high-performance programmable processors specifically designed for signal processing applications. They are extremely flexible, low power, and cost efficient, but lack hardware acceleration capabilities for leading image and video processing algorithms. ASICs provide very high performance, small silicon area and low power consumption but do not have the flexibility to adapt to new algorithms. ASIP is a promising design approach that offers an intermediary solution between ASIC and programmable processors. However, for ASIPs, commercially established system-level design tools are still under development. Compared to ASICs, FPGAs provide both

reasonably good performance and adaptability to many different algorithms for applications. In addition, in the case of FPGAs, the non-recurring engineering (NRE) cost and the design time are not as high as for ASICs [**13**,**14**]. The amount of resources in present-day FPGAs is quite high and can practically handle many processing operations without any difficulty. FPGAs have shorter development time, lot more computation power, reasonably good hardware fabric that adds to parallel processing capabilities which are very suitable for implementing various image and video processing algorithms as against the fixed architecture devices such as DSPs, ASIPs and ASICs [**15**,**16**,**17**].

Modern FPGAs embed many predefined and pre-fabricated IP components, such as digital signal processing (DSP) elements, embedded memories along with plenty of logic resources in a single chip. FPGAs are computationally even more powerful with the presence of embedded processors. In addition, the required system buses are embedded in the FPGA fabric so that entire systems-on-chip (SoC) can be implemented on these platforms [**18**]. In fact, the use of these embedded processors could easily represent the best solution when devising the optimal design for an embedded system as they involve consideration of constraints of performance, NRE cost, area, power consumption etc. from a dual hardware and software perspective.

A typical image and video processing application can be considered as an embedded system, which consists of multiple heterogeneous resources such as, processor, peripherals, dedicated logic blocks, memories, and software [**11**]. A general classification of different hardware and software components and the various resources are shown in Fig. 1.1. As apparent from the figure, the available hardware resources can broadly be classified as memory resources, functional resources and interface resources. The functional resources are used to process vast amount of data. They implement arithmetic or logic functions and can be

grouped into three main subclasses: primitive resources, intellectual properties (IPs) and application specific resources.

The primitive resources are general-purpose sub-circuits that are designed once and often used. The IPs could be functional IPs with domain-specific features or it could be a controller's IP. Fully characterized IPs in terms of area and performance can be stored in the design library from where we can reuse them as per the need.

The application-specific resources are the subsystems designed for the application-specific needs. The interface resources support data transfer and include different types of busses, whereas, the different types of memory resources are used to store data. The software resources include device drivers, real-time operating system (RTOS), application-program interface (API) and network communications, which are managed by the processor [**8**].

FPGA provides an excellent platform for implementing an embedded system with the required resources as above. The application-specific hardware architecture of the system in an FPGA can be a design choice to the user and has proved to be very effective. In the image and video processing applications, FPGA platform based design methodology can be effectively employed for rapidly achieving the goal from conception to a successful system model [**19**]. The wide range of requisite peripherals and the high performance FPGA device available on the platform provide adequate support to build new architectures and realize a complete system over the platform. In addition, the embedded processor present in the FPGA device makes it more versatile. The FPGA-based platform has an important bearing on the architectural choices and algorithm development for the implementation and verification of emerging heterogeneous real-time image and video processing systems.

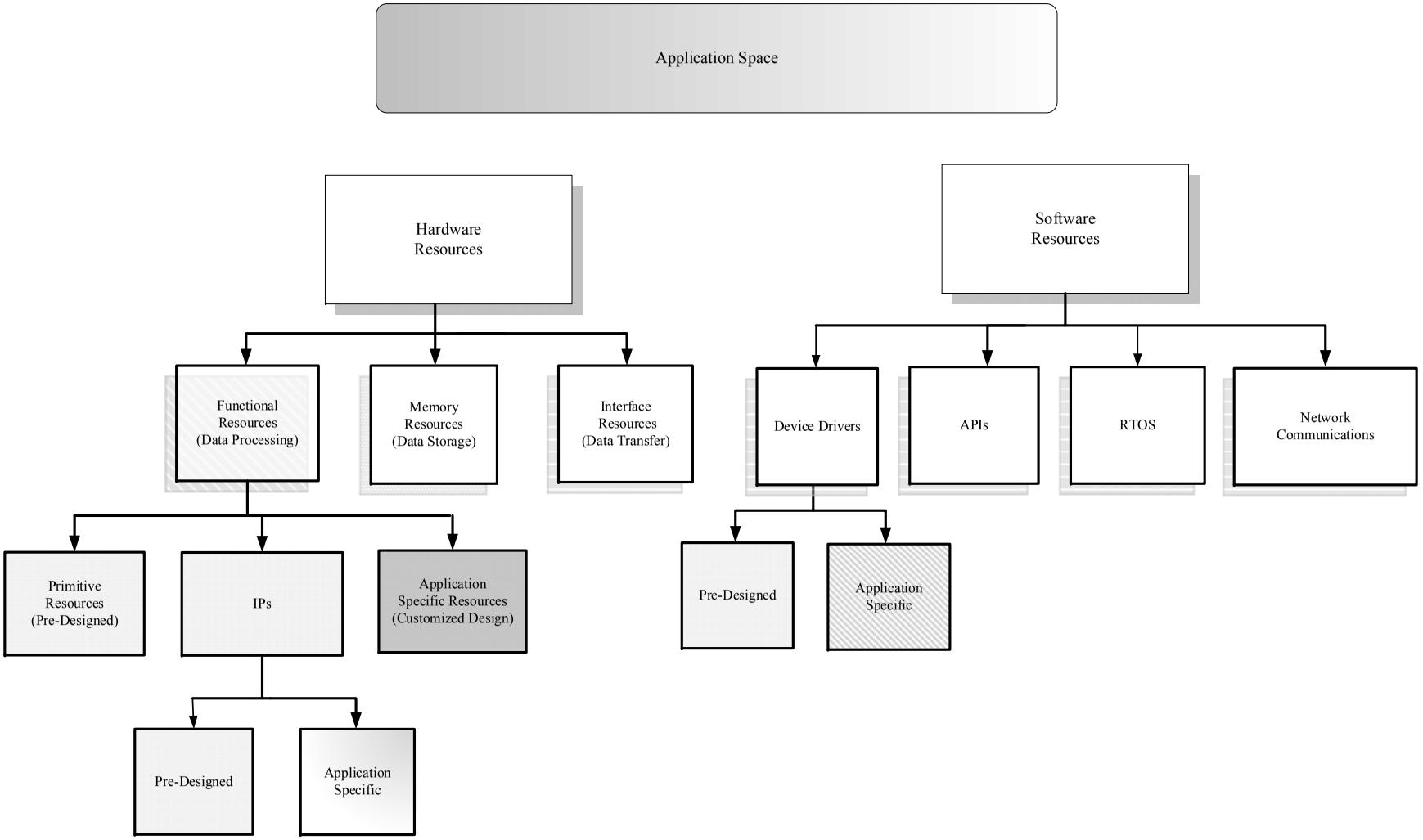


Fig. 1.1: Classification of different resources in a typical image processing system.

With the advent of modern FPGA-based platform and its heterogeneous resource offerings, the platform based design of image and video processing systems are getting more popular [**20**,**21**]. Apart from the other architectural developmental benefits, one of the main advantages of using the platform-based design and its associated integrated design environment is its wide range offerings of generic IP elements. These generic IP elements include processor IP core, interface/bus/bridge IP core, peripheral IP core, communication IP core, infrastructure IP core, memory controller IP core and debug IP core [**20**,**22**]. The notable FPGA platforms that support the IP core based embedded platform-centric design approach are offered by Xilinx [**23**], Altera [**24**] and Celoxica [**25**].

In this thesis, the IP based design approach is followed and various hardware architectural blocks have been designed that are required for image and video processing applications. The thesis deals with some of the important aspects of hardware-software partitioning and development of relevant architectures and algorithms. The work in the thesis uses Xilinx ML-507 FPGA platform that contains a Virtex-5 FXT FPGA device. The details of the Xilinx ML-platform and Virtex-5 device are illustrated in Appendix-A.

1.2 Motivation for the Work

Image and video processing systems and their associated algorithms can be implemented in software, hardware or in combination of both. The software implementation takes less development time but offers flexibility for any future change in the functionality of the tasks. However, the processing time of the software implementation is rather high. On the contrary, the hardware implementations can exploit the inherent parallelism of the tasks, and usually result in faster processing. Nonetheless, the hardware implementations are fixed and do not provide the necessary flexibility for prompt changes in the behavior of the systems. In addition, the development time for the hardware implementation is high as compared to

software implementation. Thus, the traditional design methodologies of providing complete hardware or software solutions are fast becoming infeasible. It is apparent that a mixed hardware-software realization would provide a better solution, which judiciously leverages the flexibility offered by the software and performance gain offered by the hardware.

In a hardware/software based design approach the optimal design choice can be obtained for handling conflicting design requirements, such as flexibility, power, resources, design time and cost [**10**]. Depending upon the specifications of data processing algorithm, the computational task can be sequential, concurrent or mixed. A processor available with the processing unit can easily manage the sequential part of the algorithm. However, for the computation of complex and other concurrent operations there are essential requirements of designing custom computing engines. These computing engines are made synchronous with the processor and used for complex image data processing. Thus, the best design choice leads to hardware/software mixed implementation.

To meet performance goals of various image and video processing systems, including the real-time constraints, a systemic arrangement of general purpose and some application-tailored hardware and software components is required. Both the development of hardware blocks with new features and the reuse of existing IP components are essential. Furthermore, design complexities are progressively rising with an increasing number of hardware and software components that have to work together in unison. The fast evolving specification of image and video processing system needs a configurable and flexible system architecture and associated components, so that, the system can also support new features.

In order to develop the required hardware and software components in an integrated fashion, the platform-based design approach offers the best possible features of both hardware and software [**19**,**26**]. Using the platform-based design approach, flexible system

architectures and their derivatives for any reasonably complex image and video processing algorithm can be rapidly developed [**20**]. The platform-based design approach has been hugely popular on field-programmable gate array (FPGA) devices. Some of the popular FPGA-based platforms are listed in Table 1.1 and elaborated in Section 1.3.1. The presence of processor and configurable blocks in the FPGA makes both the hardware and software components programmable. The configurability of hardware and software components makes the platform-based design a superior implementation choice for the image and video processing system.

As discussed in the *Background and Context* section, for embedded realization of image and video processing systems, apart from the standard heterogeneous components, we also need many hardware and software modules for the chosen application. Even though the general-purpose components are available in the form of standard image and video processing IP suite, the need for development of application-specific blocks based on hardware software partitioning cannot be undermined. This is in view of achieving the performance goal for a particular application. We have also utilized FPGA based platform for proposing new architectures and algorithms for frequently used components required for image and video processing applications.

In the proposed work, we have developed area-efficient new architectures and algorithms for some of the frequently used architectural and generic components in the image and video processing area. The image and video processing systems use numerous components, which are widely used across many applications. Some of the most commonly used modules include image/video read, image/video acquisition, video display, image conversion, histogram computation, similarity measure computation, global image thresholding, connected component analysis, smoothing function computation, center of gravity (COG) computation and some specialized arithmetic building blocks. These vital architectural building blocks are

designed to provide the foundation for building image and video processing systems, in IP based design environment. In addition, the proposed modules are also used to realize a standard object tracking algorithm [**27**]. The following section describes the platform-based design approach and the associated design tools, along with some of the popular image and video processing FPGA platforms.

1.3 Platform-Based Design Approach

Modern-day complex systems consist of heterogeneous hardware and software components. The hardware and software components provide the complete system's functionality. The absolute requirement of the system design aims to provide system functionality with adequate performance level while reducing its design time.

With the advent of new FPGA devices, it is possible to have a software programmable processor and the hardware accelerating engines in the same FPGA device. The logic blocks within the FPGA can be interconnected through the programming of interconnects to design desired hardware with embedded processor for general-purpose applications. The innovative development of FPGAs whose configuration could be reprogrammed unlimited number of times, thus providing the designer the option of developing reconfigurable architectures.

In application-driven architectural design context, the term platform is defined as a collection of subsystems and required interfaces that form a common arrangement of functional units from which a system and its derivatives can be efficiently developed and shaped [**28**,**29**,**30**]. Platform is an abstraction of a group of varied micro-architectures, which are programmable, and occasionally, run-time configurable in nature. It offers a universal architectural component that can support a variety of applications as well as the future derivatives of a given application space. Apart from having vital architectural building blocks, it also provides for the trade-offs among a set of essential architectural constraints,

such as, power, performance, area, design time, and cost [**28**,**29**,**30**]. In Fig. 1.2, such a platform-based design approach is depicted schematically. Here, the platform can be used by utilizing the available integrated design environment (IDE), which manages various IPs and their integration along with the configuration of available peripherals as per the specific application needs.

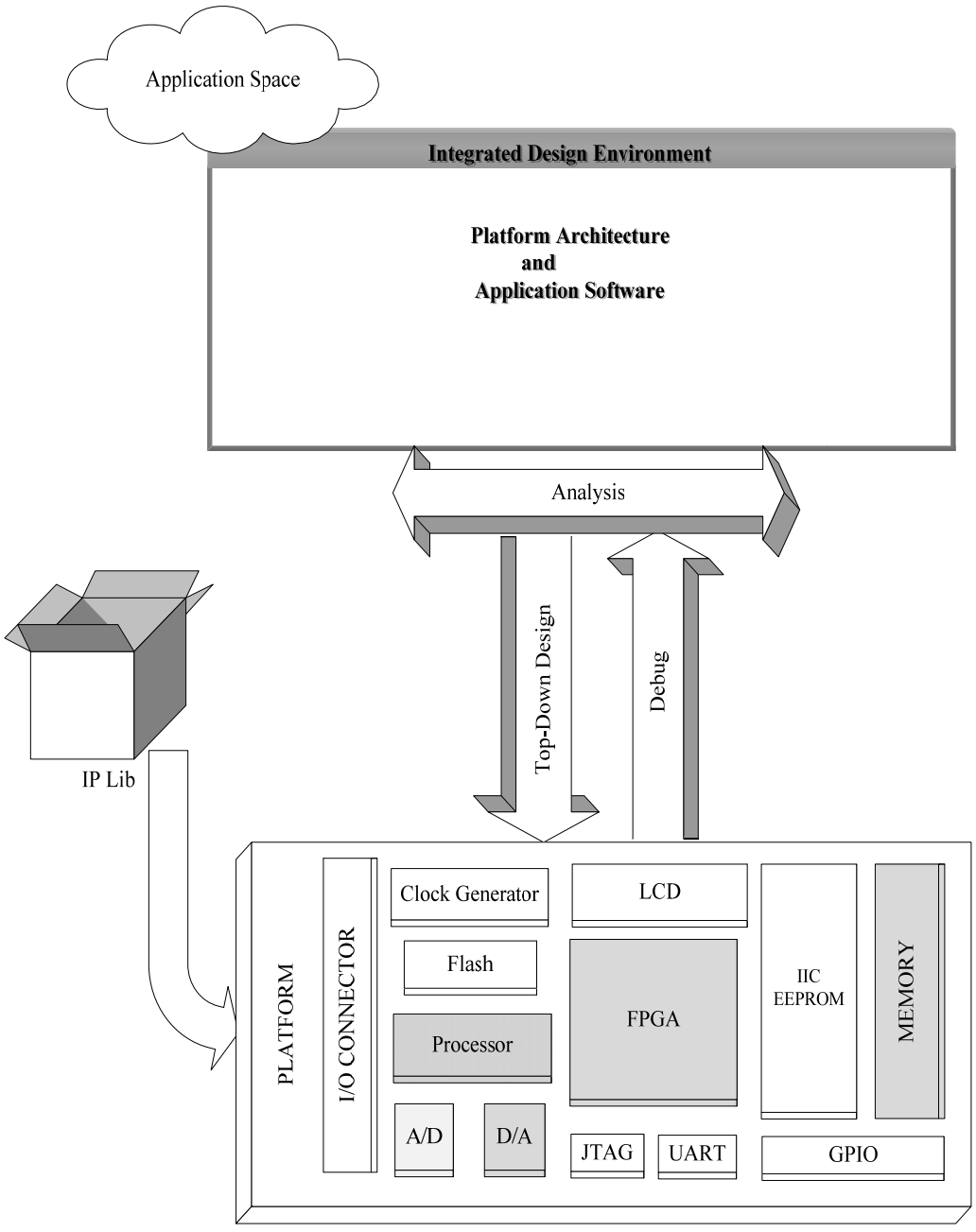


Fig. 1.2: A general-purpose FPGA-based platform.

The platform-based design approach is an amalgamation of several design approaches, which emphasizes systematic reuse for developing multipart products based upon the platform-compatible hardware and software. Every element of the platform can be selected and used through the customization of an appropriate set of design parameters through standard bus and application programmer interface (API) offered by the platform. There are various integrated design environments available, which offer complete support for the development of platform architecture and associated application software [**26**,**31**]. Thus, the platform-based design leverages the performance of most efficient derivative of an architecture and the flexibility offered by the programmability of the processor. The support of custom design hardware and reuse of IPs and other functional components makes the platform-based design approach more favorable for architecture exploration of complex digital system [**28**,**29**,**30**].

In an embedded FPGA-based platform, the software programmability comes from the availability of processor and hardware programmability comes from the presence of reconfigurable blocks of FPGA [**28**]. One such recent FPGA device is the Xilinx Virtex-5 FX family, which offers PowerPC 440 hard processor embedded in the FPGA fabric [**32**]. The combination of processor and run-time reconfigurable logic makes the FPGA-based platform very suitable for providing sufficient balance between the demands of application space and the architectural space. With embedded processor inside the FPGA, we can make trade-offs between hardware and software to maximize the performance. To use Xilinx FPGA-based platform, extensive peripherals and soft IP libraries are available [**33**].

The Xilinx ML-507 platform is shown in Fig. 1.3 that contains the PowerPC 440 processor in the Virtex-5 FPGA device and the other required platform peripherals. In the Xilinx design environment, the processor IP core can be a soft IP core like MicroBlaze processor or it can be a hard IP core such as PowerPC processor [**34**]. The interface IP core supports the

processor local bus (PLB), fast simplex link bus and the PowerPC device control register bus. In the peripheral IP core category, there are many general peripheral cores available including inter-integrated circuit (I2C) controller, TFT controller, watchdog timer, and interrupt controller. The communication IP provides universal asynchronous receiver transmitter (UART) controller and the Ethernet controller through which the platform can communicate with the host system or communicate in a network-based environment.

The external memory controller communicates through the memory control interface (MCI) bus that connects the external memory, such as the DDR2 memory with the PowerPC processor. The other important controllers are the joint test action group (JTAG) controller for PowerPC processor, processor reset controller, bus splitter and clock generator. Similarly, the memory controller IP supports multi-port memory controller (MPMC), Block RAM (BRAM) interface controller, and direct memory access (DMA) controller. Finally, the debug IP provides ChipScope Pro integrated logic analyzer (ILA), ChipScope Pro bus analyzer and ChipScope Pro virtual I/O [**35**].

Apart from the wide range of standard IP support, the application specific custom IPs developed by individual users can also be ported in the IP library using the ML-507 platform. The standard IPs are configurable and parameterizable in nature [**36**]. Depending upon the application needs these standard IPs can be configured and integrated with the custom IP. A high abstraction electronic design automation (EDA) tool manages the amalgamation of different varieties of IP [**37**]. The requirement of IP suite is highly dependent upon the selected application domain.

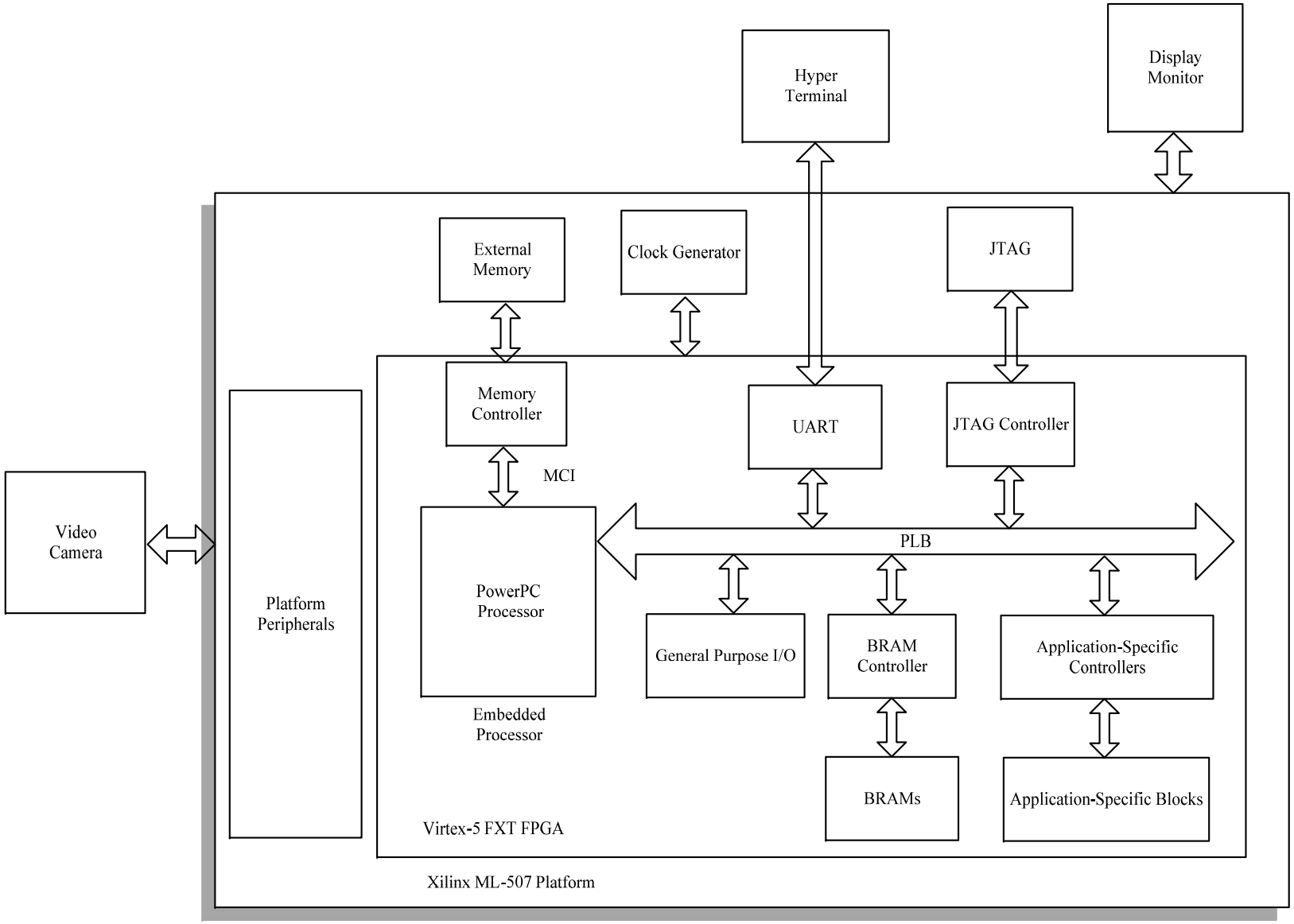


Fig. 1.3: The Xilinx ML-507 platform.

1.3.1 FPGA-Based Embedded Vision Platforms

There are many popular embedded vision platforms on FPGAs. The Xilinx Celoxica RC1000 XCV2000E FPGA-based platform is used to perform image pre-processing functions for embedded vision applications [**16**]. A general-purpose, multitasking, and reconfigurable platform is presented in [**21**]. Based on the Xilinx Virtex-II FPGA, a system level architecture is proposed and developed, which integrates embedded processor, memory control and interface technologies.

The system includes different functional modules, such as edge detection, zoom-in and zoom-out functions, which provide the flexibility of using the system as a general video processing platform according to different application requirements. Table 1.1 shows some of the related embedded platforms for image and video processing applications.

An FPGA-based embedded platform for real-time image acquisition and processing is presented in [**38**]. It contains a Texas Instrument's TMS320C6416T digital signal processor and Altera's FPGA EP3C25F324. The digital image data is first transferred into FPGA fabrics. After pre-processing, the data is transferred into DSP6416 by the interface of first in, first out (FIFO) in the FPGA and DSP6416 external memory interface (EMIF). Further, the image data is processed in DSP by real-time algorithms. Bravo et al. [**20**] have used Xilinx Virtex-4 xc4vfx12 FPGA-based platform, which contains an embedded PowerPC405 microprocessor. In this work, architecture for image acquisition and processing using a complementary metal oxide semiconductor (CMOS) sensor is presented. The sensor is interfaced with the FPGA platform for the smart camera application.

A reconfigurable open architecture computing hardware (ROACH) is a standalone FPGA processing board [**39**]. The main part of ROACH is a Xilinx Virtex-5 FPGA (either lx110t for logic-intensive applications, or sx95t for DSP-slice-intensive applications) device. A separate PowerPC runs Linux and it is used to control the platform [**39**]. Similarly, ROACH2 is a

Virtex-6 sx475t FPGA- based (xc6vsx475t device) platform. Here, an embedded PowerPC 440EPx stand-alone processor controls the required functions [**40**].

Table 1.1: FPGA-based platforms

S. No.	Work	Year	Platform
1	The platform of image acquisition and processing system based on DSP and FPGA Y. Lei, Z. Gang et al [**38**].	2008	Altera FPGA EP3C25F324 + TI TMS320C6416T DSP
2	A general-purpose FPGA-based reconfigurable platform for video and image processing J. Li, H. He et al. [**21**]	2009	Xilinx Virtex-II FPGA
3	Efficient smart CMOS camera based on FPGAs oriented to embedded image processing I. Bravo, J. Balinas et al. [**20**]	2011	Xilinx Virtex-4 FX FPGA (XC4VFX12)
4	A high-performance FPGA platform for adaptive optics real-time control Heng Zhang , Zoran Ljusic , et al. [**41**]	2012	Kermode Xilinx Virtex-6 SX475 FPGAs
5	An FPGA-based platform for accelerated offline spike sorting Sarah Gibson, Jack W. Judy and Dejan Markovic [**42**]	2013	Berkeley's CASPER- ROACH (Xilinx Virtex-5 XC5VLX110T/XC5VSX95T FPGA)
6	Berkeley's CASPER- ROACH (Reconfigurable Open Architecture Computing Hardware) a standalone FPGA processing board https://casper.berkeley.edu/wiki/ROACH [**39**]	2013	Xilinx Virtex-5 XC5VLX110T or Virtex-5 XC5VSX95T FPGA
7	Berkeley's CASPER-ROACH-2 https://casper.berkeley.edu/wiki/ROACH2 [**40**]	2013	Xilinx Virtex-6 SX475T FPGA (XC6VSX475T-1FFG1759C)

For the basic image and video processing, video starter kits can also be used. There are a numbers of video starter kits available [**43**,**44**,**45**]. However, these kits are expensive and do not contain top-of-the-line FPGA devices. The mounted FPGA on these platforms has limited resources, which imposes constraints for implementing any reasonably complex video processing algorithms on these kits.

Apart from the above issues, the mounted camera on the above mentioned kits is fixed and has very low resolution. For many real-time applications like video surveillance, tracking etc. there is a need for interfacing a higher resolution camera or a pan-tilt-zoom (PTZ) camera and other custom interfacing peripherals with a high performance processor. Thus, to implement a complex image and video processing algorithm there is requirement of a high-end device based FPGA platform, which can perform such applications competitively. The uses of the various tools for working with platform-based design are explained below.

1.3.2 Platform Design Tools

An embedded system is an amalgamation of hardware and software entities, which are managed by the hardware, software and the configuration tools. The platform peripherals are configured by using their high-level functions provided by the platform, which are available in the form of APIs. Similarly, the custom APIs can be developed for the application-specific user IPs.

Xilinx provides embedded development kit (EDK), design tool to manage the hardware and software components of the system [**37**]. It is an integrated design and development environment for designing embedded processing systems. This pre-configured kit includes Xilinx platform studio (XPS) and the software development kit (SDK), as well as all the documentation and IPs that are required for designing Xilinx platform FPGAs, such as Virtex-5 FXT FPGA with embedded PowerPC 440 hard processor cores and/or MicroBlaze soft processor cores [**46**]. Some of the platform design tools and their uses are explained below,

- XPS tool suite including graphical integrated design environment (IDE) and command-line support for developing hardware platforms for embedded applications.

- The Base System Builder (BSB) wizard enables creation of a working embedded system with the desired FPGA platform such as Xilinx ML-507 [**33**].
- SDK is the software-centric design environment based on the Eclipse IDE. It includes the GNU C/C++ compiler and debugger, Xilinx Microprocessor Debug (XMD) target server, Data2MEM (D2M) utility for bit stream loading and updating [**46**].
- Real-time operating system (RTOS) and embedded OS (EOS) provide design support and board support package (BSP) generation for numerous third party suppliers in the Xilinx environment [**46**].
- IP catalog that includes a wide variety of processing and peripheral cores such as processing IP (PIP) and flexible MicroBlaze soft processor (MSP) core for customizing the embedded system [**46**].

1.4 Scope and Objectives

In this thesis, we have proposed various hardware architectural modules along with their requisite software integration for embedded realization of a video processing application. The hardware/software implementation choices and development of hardware architectures needed are the main motivations of this thesis. The Xilinx ML-507 FPGA-based platform, tools and its associated design tools and methodologies support the required path to meet the various goals of the thesis. The embedded PowerPC processor, available on the Virtex-5 FXT FPGA device on the selected platform, fulfills the specific needs of hardware/software implementation. The main objectives of this thesis are as given below.

The first objective of the thesis is the development of the required configuration of Xilinx ML-507 platform on which the integrated hardware-software solutions are proposed for various embedded image and video processing applications. This requires the configuration

of the FPGA-platform peripherals using APIs and other required hardware building blocks. This configuration is necessary for accessing of the image pixels by the FPGA and for testing various architectural blocks designed subsequently. The required video acquisition unit is developed on the configured platform that uses some of the standard IP components and peripherals available on the platform. The real-time video acquisition module buffers 640×480 VGA resolution video frame available at 60 frames per second.

The second objective of the thesis is to propose and develop various architectural building blocks, that are mostly generic in nature and which can widely be used in many practical image and video processing systems. The developed intellectual property (IP) cores of the architectures can be used in any IP-based design environment and can be utilized to design a practical image and video processing system.

In the proposed architectures, most of the operations are performed using the 32-bit fixed-point format. The complex arithmetic operations are realized through a fixed-point binary logarithmic and antilogarithmic unit. Architectures based on the logarithmic number system (LNS) have the advantages of minimizing logic resources and the processing of large datasets, by realizing time-critical processes in the available BRAM and DSP slices available on the FPGA device and show effective use of resources for the required throughput and speed goal. The logarithm and antilogarithm units are utilized for various requisite complex operations such as square root, powering, inverse square root and division operations and provide the backbone of the many architectural blocks developed in the thesis.

For developing resource-efficient and high performance architectural building blocks for the compute-intensive modules, the fixed-point number system has been used in contrast to the floating-point number system [**47**,**48**]. The primary reason for this is that the fixed-point arithmetic uses simple integer datapath and can be easily realized in the small FPGA fabric,

thus, consuming fewer resources. The optimized FPGA macro elements available in the FPGA device can be customized and used as per the specific needs. Apart from this, the fixed-point arithmetic also offers higher clock rates, which are required to implement real-time image and video processing system.

The third objective of the thesis is to design a hardware architecture for the global image thresholding unit that works on the gray scale pixels and provides the corresponding binary image. The gray pixels are obtained from the RGB color pixels and the required hardware architecture for RGB to gray conversion is designed. The direct implementation of the chosen thresholding algorithm, i.e., the Otsu's algorithm [**49**] requires numerous computation intensive resources such as iterative squaring, complex multipliers, and dividers with fractional value accuracy [**50**,**51**]. Thus, we present a resource-efficient architecture for the design of Otsu's image thresholding algorithm for implementing in the Virtex-5 device available in ML-507 board. The between-class variance computation in Otsu's algorithm requires the hardware blocks for computing normalized cumulative histogram, mean and cumulative moments in single-cycle read-modify-write operations, that are implemented in the thesis. To simplify the thresholding operation, hardware architectures for the computation of normalized cumulative histogram (NCH), normalized cumulative intensity area (NCIA) and logarithm units are proposed that find usages in many image and video processing applications.

The fourth objective is the study and improvement of connected component analysis (CCA) algorithm. The CCA algorithm works on the binary image obtained from the image thresholding unit and segments out the object by region labeling. The popular raster-scan based CCA labeling algorithm proposed by Stefano and Bulgarelli (SB) is taken up for our study. An improved version of the algorithm is proposed that improves the equivalence

handling of the SB algorithm. As the proposed algorithm is rich in control and decision loops, it is implemented in the embedded PowerPC processor.

The fifth and final objective is to demonstrate a video processing application that utilizes the various blocks proposed and designed as above. Object tracking is selected as this application; the block diagram of the flow of processing is given in Fig. 1.5. After video acquisition, the processing is done in two stages, namely target object identification and object tracking. The identification of an object in a particular frame is done by first converting the color pixels to grayscale, and applying an image thresholding algorithm to segment out the foreground pixels from its background. The binary image obtained from output of the image thresholding unit is used by the connected component labeling algorithm to identify and segment the object from the background for the tracking application. Subsequently, the coordinates of the target, thus obtained from the object identification block, are input to the video tracking algorithm. The chosen video tracking algorithm is based on kernel-based mean shift approach (KBMS).

The KBMS algorithm is based on the concept of the mean shift clustering [**5**]. After, color space quantization, the histogram works on the local image statistics for target modeling and target candidate modeling. For smoothing of the probability density functions (pdf) histogram, a kernel weight computation is needed. The proposed architectures constitute the kernel smoothed local histogram block (KSLH) for modeling the target object. Further, a hardware architecture for similarity measure computation has been proposed, which computes similarities between two discrete histogram pdf-s. In the KBMS algorithm, this module plays the role for finding the distance between object's next position, with respect to its previous position [**27**].

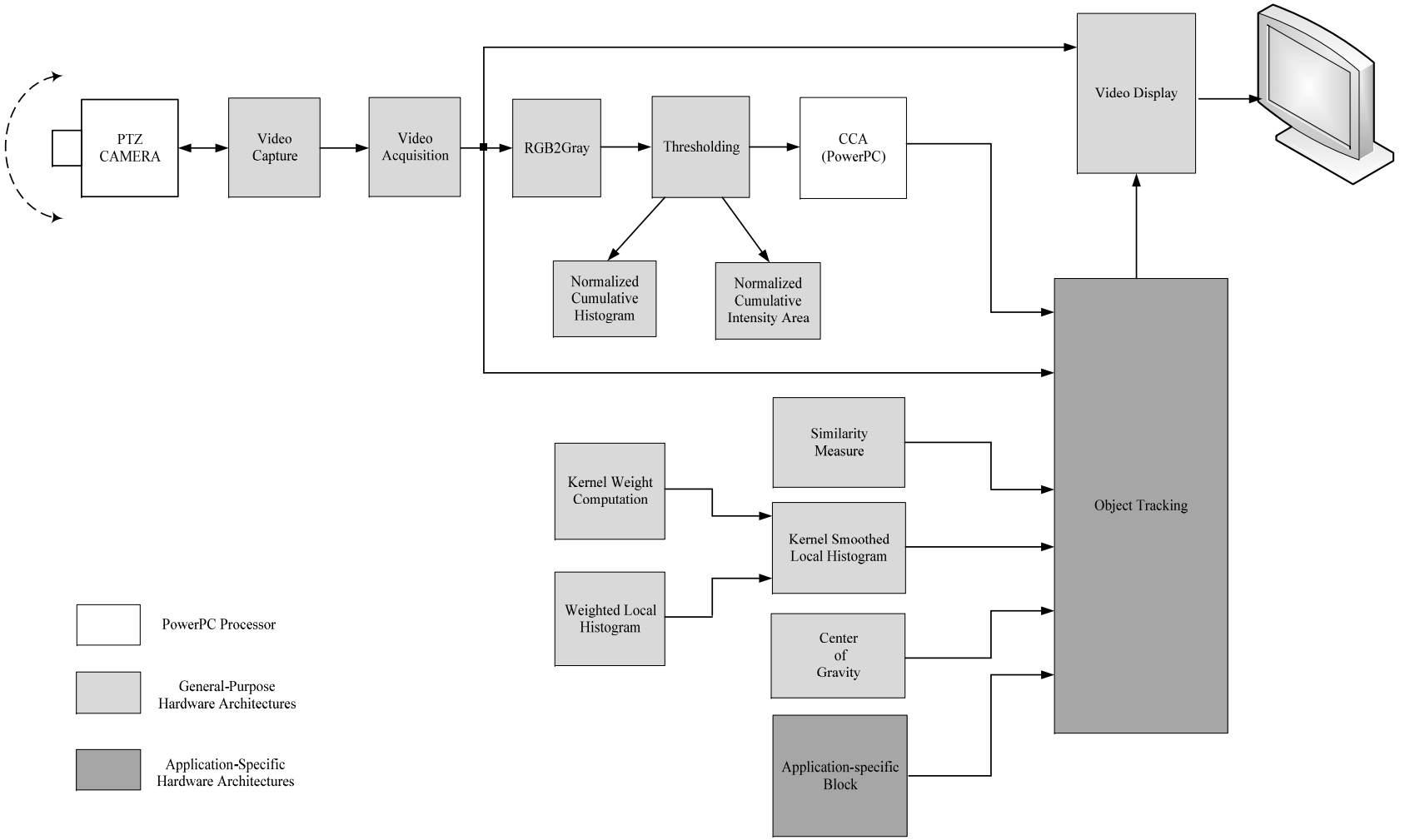


Fig. 1.4: Block diagram for realizing the object tracking algorithm.

The new location of the object is obtained by computing the center of gravity (COG) in this mean shift based tracking [**27**]. To realize the KBMS algorithm, the various building blocks as discussed above are shown in the Fig. 1.6. In the IP centric system environment, all the developed hardware and software blocks are used as per the specific bus interface. The specific integration of various modules for realization of the kernel-based object tracking algorithm is carried out. The Xilinx embedded development kit (EDK) design tool integrates the communication of the required IPs with the embedded PowerPC processor, which runs the application program and the configuration software. Xilinx XPower Analyzer tool has been used to compute power consumption associated with various architectural modules [**52**].

1.5 Proposed Hardware/Software Modules for an Embedded Video Processing Application

In this section, we describe the hardware architectural blocks and the algorithms that have been proposed in the thesis for implementation on Xilinx FPGA platform. The proposed blocks are needed for many image and video processing applications. Using the embedded approach and utilizing the hardware and software blocks, we have also implemented a reference video processing application, namely, object tracking based on the platform-based design methodology on the FPGA. A generic video processing system is shown in Fig. 1.4. Here, the video camera captures the real-time video within its region-of-interest (ROI). The video acquisition unit acquires the frame of images from the camera. The data processing unit does the computation necessary to realize the specific application algorithm. This unit and a display unit along with the requisite communication interface control, communicate and display the processed results. The feedback from the data processing unit to video camera is used for controlling the camera movement based on the processing and application requirement. The following subsections describe the proposed hardware blocks and software components along with the configurations necessary to implement the targeted application.

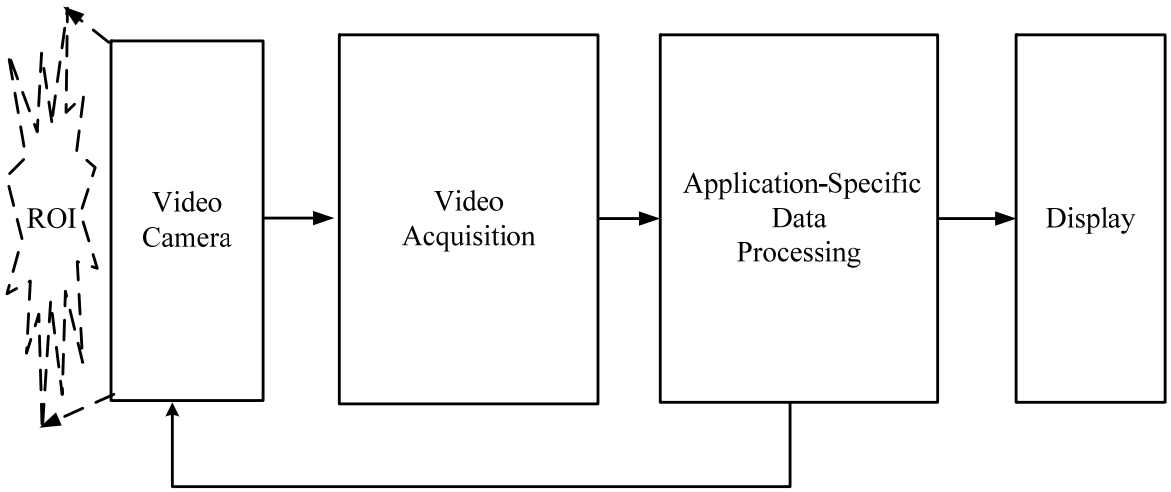


Fig. 1.5: A generic image and video processing system.

1.5.1 Platform Configuration

The configuration of the ML-507 FPGA-based platform is of foremost importance for realizing any real-time video processing application using platform based design. Subsequent to the appropriate configuration, the platform becomes ready for embedded video processing realization. This is achieved by using various off-the-shelf peripherals available on the platform and generic IPs available in its associated integrated design tools. The configuration is validated by capturing the real-time video and passing it out to the displaying unit for preliminary testing of the implementation framework. The configuration requires I2C and VGA bus protocols, which are controlled and managed by the embedded processor. The connectivity of the VGA video source is supported by the video input video codec (VDEC) chip that is available on the platform. The VDEC is configured by programming the various control registers in it through the PowerPC processor. Similarly, the display portion of the system uses display controller chip, which provides the facility to connect a video graphics array (VGA) or digital visiual interface (DVI) monitor through it.

The control registers of the display controller peripheral are also programmed through I2C bus protocol by sending data from the PowerPC embedded processor. Some simple hardware

modules are realized in the FPGA fabric to facilitate the streaming of real-time video for standard 640×480 VGA input and its subsequent display for validating the configuration. The design and configuration of the platform is of generic nature and is extensible which can be managed as per the application needs. The developed video/image read and display unit facilitates the development of a wide range of image and video streaming applications. Some of the video streaming application are, video streaming system [**53**], video streaming over wireless network [**54**], traffic management [**55**] and wireless network quality management system [**56**].

1.5.2 Video Acquisition

After configuring the platform, the VGA input video coder-decoder (codec) provides the image frame to the FPGA fabrics. To perform the requisite image processing operations on the image pixels, the image or frame(s) of video is/are required to reside in a memory. An image or video acquisition unit fulfills the need of buffering the large set of image pixels in the memory. The Xilinx ML-507 platform offers a 256 MB DDR2 SDRAM memory, in which a large number of frames can be stored. In the proposed unit of image and video acquisition, the DDR2 SDRAM memory is used to store a real-time video captured by the camera. In the embedded architecture, as proposed, the ML-507 platform is utilized to realize an image and video acquisition unit for buffering a standard 640×480 pixel frame at 60 frames per second. A video-to-frame converter hardware module is realized in the FPGA fabric that converts the video into frames and sends them to the memory. Similarly, frame to video conversion is realized using a frame to video conversion hardware module in the FPGA. To control and access the DDR2 SDRAM memory, a multi-port memory controller (MPMC) is utilized. The MPMC offers a video frame buffer controller (VFBC) protocol which supports the frame buffering operation. The architecture uses one port of the MPMC, which is dedicated for buffering the image frame and another port is utilized to retrieve back

the frame from the memory. Here, both the ports work as per the VFBC protocol. To control and manage the data buffering operation, the embedded PowerPC processor communicates with the memory controller interface (MCI) bus. This communication is supported by the third port of the MPMC, which independently works as per the MCI bus protocol. The architectural arrangement uses a few generic IP elements that are offered by the integrated design tool.

1.5.3 Arithmetic Datapath

The video processing blocks need high-performance arithmetic datapath with reasonably good arithmetic precision. Integer arithmetic based computational operations provide resource-efficient, high performance datapath, but they lack the arithmetic precision needed. To achieve the required precision floating-point number system could be a good choice but its realization is resource-intensive that slows down the datapath as compared to the integer-based datapath. A fixed-point number system offers the area and speed advantage of integer datapath with reasonably good precision. The fixed-point number system based datapath can work at higher clock frequencies and provides the ease of implementation of an integer-based datapath. In addition to this, most of the integer arithmetic based off-the-shelf hard IP components offered by the FPGA device can also be efficiently utilized by the fixed-point arithmetic based datapath. Thus, to fulfill the high-performance computational needs of image processing, most of the compute-intensive operations are realized by utilizing the fixed-point arithmetic number system. The hardware architectures are thus proposed with fixed-point arithmetic.

1.5.4 Logarithm and Antilogarithm Units

It is well known that complex computations, such as the computation of square root and division, can be achieved by using the logarithmic number system (LNS). The LNS architectures require simple arithmetic operations, such as only addition/subtraction and

shifting. The proposed hardware architectures of logarithm and antilogarithm units use the fixed-point arithmetic. These units have been utilized for the computation of between-class-variance needed for the global thresholding operation and for designing the hardware architecture for other application-specific blocks.

1.5.5 Hardware Architecture of Global Thresholding

The thresholding unit works on the gray image data and computes the optimum threshold value for the required binary conversion of gray level image. The image thresholding unit finds a wide range of applications. Some of the popular applications are noise reduction for human action recognition [**57**], automated visual inspection of defects [**58**], change detection [**59**], real-time segmentation of images with complex backgrounds [**60**], text detection in natural images [**61**], optical character recognition and image extraction [**62**,**63**], adaptive progressive thresholding [**6**], and personal verification [**4**]. To achieve the real-time computational efficiency of the global image thresholding process, the hardware implementation of the thresholding algorithm is necessary [**50**,**51**,**64**]. The direct hardware implementation of the global image thresholding algorithm as proposed by Otsu [**49**] boils down to the computation of between-class variance (BCV). The BCV architecture is broken down in hardware blocks for the computation of normalized cumulative histogram, mean and cumulative moments, using single-cycle read-modify-write operations. The hardware unit also requires many computation intensive resources such as iterative squaring, complex multipliers, and dividers with fractional value accuracy [**50**,**51**]. In our work, a resource-efficient architecture for the design of Otsu's image thresholding algorithm and its implementation in the Virtex-5 device available in ML-507 platform is presented.

1.5.6 PowerPC Realization of Connected Component Analysis

The connected component analysis (CCA) algorithm is taken up next for its implementation and utilization in the embedded design framework. The CCA algorithm, segments out objects

of interest from the background pixels by means of connected component or region labeling [**8**,**9**]. The connected component analysis is used in a variety of applications, which includes, finding individual letters in complex color images [**65**], automatic feature extraction from scanned topographic maps [**66**], reading text in scene images [**67**], face recognition [**68**], fingerprint identification [**69**], automated inspection [**70**], automatic writer identification [**71**,**72**], computer-aided diagnosis [**73**], video and signal based surveillance, barcode recognition [**74**], medical image analysis [**7**] and object recognition and tracking [**75**]. One of the most widely used CCA algorithms is proposed by Stefano and Bulgarelli (SB) [**76**]. In our proposed work, the equivalence handling of SB algorithm is improved upon so that number of conflicts is less and precise results are obtained. The improved algorithm is abundant in control and simple decision loops. Thus, the CCA algorithm has been given a software implementation and runs on the embedded PowerPC processor [**34**].

1.5.7 Embedded Realization of Kernel-based Mean Shift Algorithm

In our work, we have chosen the application of real-time object tracking that utilizes the proposed architectures and algorithms, and gives an embedded realization of it using platform-based design methodology. Object tracking is defined as the problem of estimating the trajectory of an object in the image plane as it moves around a scene. The object tracking algorithm finds a wide use in the image and video processing applications including those for augmented reality [**77**], automated vehicle tracking [**78**], target localization in unmanned air vehicles [**79**], face tracking [**80**], identity verification [**81**] and many more [**5**,**8**,**82**]. The object tracking algorithm that we have selected for embedded implementation is the KBMS algorithm [**27**]. Researchers have reported that a hardware or hardware/software implementation is necessary for the KBMS algorithm to achieve effective real-time computational efficiency [**83**,**84**].

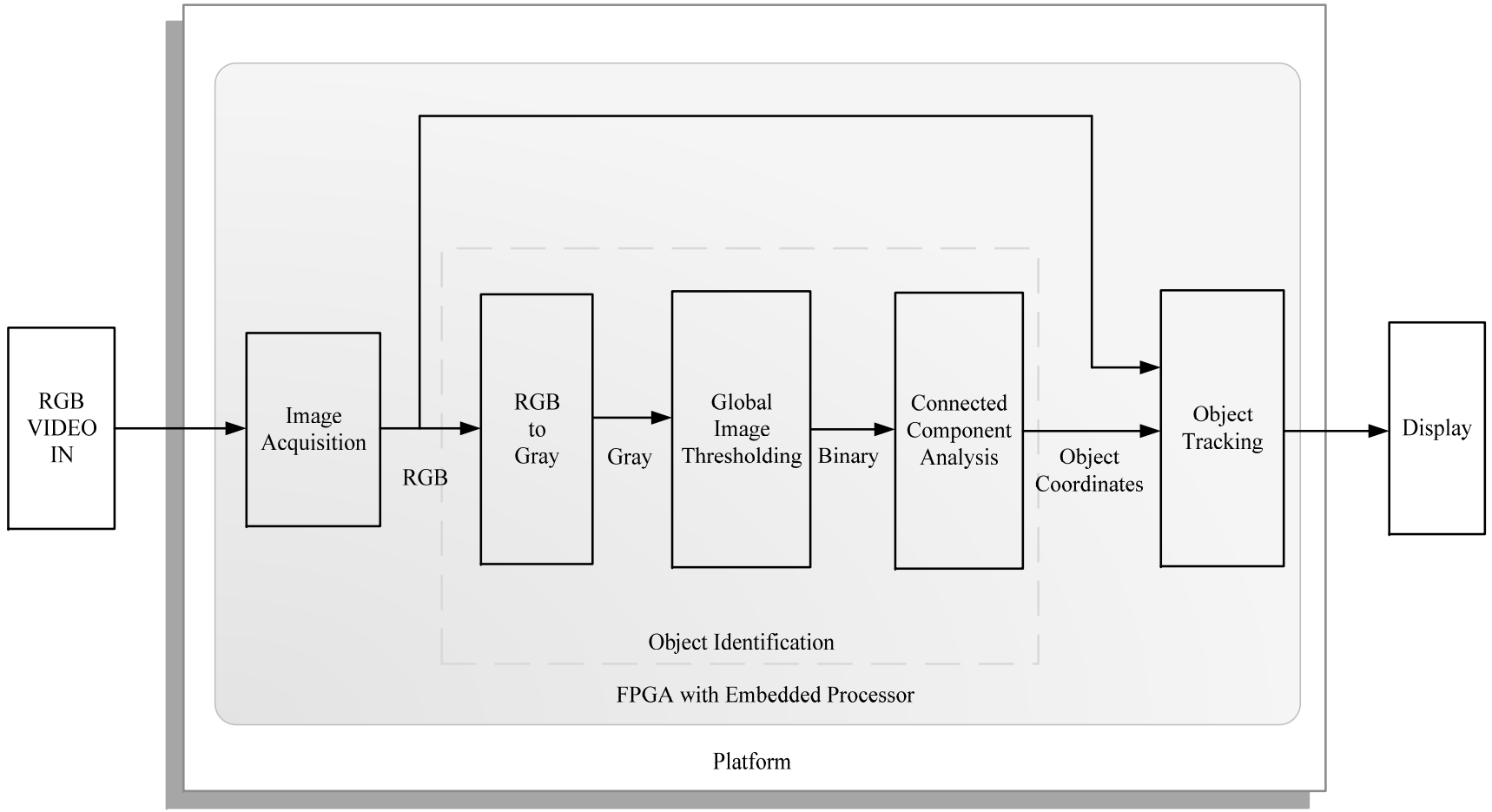


Fig. 1.6: Complete system arrangement for realizing an object tracking algorithm.

The KBMS algorithm utilizes most of the proposed hardware architectural block along with the CCA algorithm. The block diagram of the flow of processing is given in Fig. 1.5. After video acquisition, the processing is done in two stages, namely target object identification and object tracking. The identification of an object in a particular frame is done by first converting the color pixels to grayscale, and applying an image thresholding algorithm to segment out the foreground pixels from their background. The binary image obtained from the output of the image thresholding unit is used by connected component labeling algorithm to identify and segment the object from the background for the tracking application. Subsequently, the coordinates of the target, thus obtained from object identification block, are input to the video tracking algorithm that is based on kernel-based mean shift approach.

The datapath uses the fixed-point arithmetic, which offers reasonably good performance with reduced hardware consumption. Apart from utilizing the hardware architectural blocks as designed, the software tasks such as simpler data movement and control operations required in the KBMS algorithm are handled by the embedded processor available in Xilinx ML-507. The PowerPC processor manages the control steps of the KBMS algorithm along with running the CCA algorithm.

1.6 Major Contributions and Organization of the Thesis

In this thesis, we have proposed various hardware architectural modules along with their requisite software integration for embedded realization of video processing application. The approach followed in our work is based on the platform-based design methodology on Xilinx ML-507 FPGA platform. The reference application chosen for utilizing various hardware architectural blocks is KBMS object tracking algorithm.The work carried out in the thesis makes some important contributions. The specific contributions of the thesis are listed below:

- An embedded architecture has been proposed for capturing 640×480 resolution real-time video and buffering it in the DDR2 SDRAM and implemented in Virtex-5 FX FPGA of ML-507 platform. This work has been accomplished after the required configuration of the Xilinx ML-507 FPGA-based platform. This approach can further be utilized for several video streaming applications and for applications requiring DDR2 SDRAM frame buffering.
- Resource-efficient hardware architectures for logarithm and antilogarithm computing units have been proposed. The proposed architectures are implemented in the Virtex-5 FPGA. Using logarithmic number system (LNS), these architectures are utilized for realizing complex arithmetic functions, as required.
- The global image thresholding architecture proposed by Otsu [**49**] has been implemented using the proposed architectural blocks namely, NCH, NCIA and between-class variance (BCV).
- An efficient and improved two-scan equivalence-based connected component labeling algorithm has been proposed drawing on the work of on Stefano and Bulgarelli [**76**] and implemented in the PowerPC embedded processor of Xilinx ML-507.
- Hardware/software partitioning and an embedded implementation of the KBMS object tracking algorithm has been implemented on using the ML-507 platform.
- Hardware architecture for various modules have been proposed, which are shown Fig. 1.6. The proposed architectures include computation of KSLH, kernel weight computation, weighted local histogram (WLH) computation, the similarity measure computation, center of gravity computation (COG) and some application-specific hardware modules. These architectural blocks are implemented in the Virtex-5 FXT device.

The different units of the architectures and algorithms for image and video processing application are organized in individual chapters. The related literature review has been covered in the respective chapters. Each chapter is dedicated to addressing the of specific image/video processing need. The second chapter illustrates the hardware/software approach for supporting the platform for various image and video processing applications. The various computational building blocks used in the thesis are based on the logarithm and antilogarithm components, which are covered in chapter three. Chapter four illustrates the thresholding unit need for the connected component labeling algorithm that is discussed in chapter five. Chapter six of the thesis covers the implementation approach for the kernel-based mean shift object tracking algorithm. The detailed organization of each chapter is follows:

Chapter 2 covers the details of the hardware/software based extensible embedded architecture for the real-time video capture/ acquisition, streaming, and its display. The architecture is based on the shared bus topology, which is controlled by the embedded PowerPC processor. The real-time VGA resolution for this work is 640×480 and the video frame rate is 60 fps. The hardware architectures for the video capture, video display and some standard IP components are synthesized in the available Xilinx Virtex-5 xc5vfx70t FPGA device. The execution of software is monitored and controlled on the hyper-terminal managed by the UART interface provided on the platform. The video camera and the display monitor are interfaced through the configuration of video input video codec and display controller peripherals available on the platform. The work described in this chapter provides the foundation for building the required architectural blocks that are needed for realizing image and video processing applications.

Chapter 3 illustrates FPGA-based architectures for computing different complex arithmetic functions such as division, square root and powering. To simplify the computational overhead a very simple datapath is created. The concept of the fixed-point arithmetic is utilized to

propose architectures for the binary logarithmic and antilogarithmic units using logarithmic number system (LNS). This chapter also describes the details of each architectural building block and their FPGA realization in the Virtex-5 FXT device. The fixed-point elements used in the architectural units use the FPGA native hard IP components.

Chapter 4 proposes an area-efficient architecture for realizing an automatic image thresholding algorithm. The selected algorithm is the Otsu's global automatic image thresholding algorithm. As shown in Fig. 1.6, the proposed architecture uses various building blocks such as normalized cumulative histogram, normalized cumulative intensity area for computing the between-class variance. The proposed architecture also utilizes the logarithmic computational unit developed in Chapter 3. Chapter 4 also discusses the system-level arrangement of the image thresholding computational block as soft IP along with its communications with other IPs and different kinds of buses.

Chapter 5 proposes an improved version of one of the widely used Stefano-Bulgarelli (SB) algorithm on connected component analysis. In our work, the equivalence handling mechanism of the SB algorithm is improved to achieve complete merger for all the possible cases. The improved algorithm is tested using a variety of test patterns and standard images and compared with the SB algorithm. The results demonstrate that the improved algorithm is simple, manages equivalences efficiently, and gives accurate count of connected components. The algorithm runs on the embedded PowerPC 440 processor available in the Xilinx Virtex-5 xc5vfx70t device.

Chapter 6 proposes hardware software embedded implementation of KBMS tracking algorithm. To analyze the various implementation needs, the KBMS object tracking algorithm is realized in MATLAB and C. After analyzing the software implementation the hardware/software implementation is proposed. The image acquisition block described in the

Chapter 2 works as a frame buffer for the object tracking. The work of Chapter 3 is utilized to propose and implement various architectural modules needed by the object tracking algorithm. The work of Chapter 4 together with that of Chapter 5 provides the segmented object for its subsequent tracking. This chapter proposes architectures for KSLH computation, kernel weighted histogram (KWH) computation, similarity measurement, center of gravity (COG), cumulative histogram computation, and some application specific blocks required in the kernel-based mean shift object tracking algorithm

Chapter 7 summarizes the thesis and provides its conclusion. The chapter also discusses the future scope of work pertaining to the thesis and each of its chapters.

CHAPTER 2

REAL-TIME VIDEO STREAMING, ACQUISITION AND DISPLAY USING FPGA-BASED PLATFORM

2.1 Introduction

A typical image or video processing system invariably consists of an acquisition block and an application-specific data processing unit. The block diagram of a generic video processing system is shown in Fig. 1.1. It consists of a video camera that captures the real-time video within its region-of-interest (ROI), a data processing unit and a display unit along with the requisite communication interfaces to control, communicate and display the processed results. Quite often, prior to data processing unit, a video acquisition module for frame buffering application is required. Some applications which do not need image memory storage and for which video streaming is good enough include, video streaming over wireless network [**54**], traffic management [**55**], wireless network quality management in different network conditions [**56**]. In [**85**] an FPGA-based license plate recognition system has been designed, which processes streaming video data. Nevertheless, a majority of image and video processing systems require an intermediate image buffer [**8**,**9**,**86**].

For the rudimentary image acquisition and processing, video starter kits can also be used. There are a numbers of video starter kits are available [**43**,**44**,**45**] and are elaborated in Section 1.3 of Chapter 1. The acquisition of image and video and their processing using these starter kits is straightforward. However, these kits are costlier and do not contain top-of-the-line FPGA devices. The mounted FPGAs on these platforms have limited resources, which impose constraints on implementing any reasonably complex video processing algorithms on

these kits. Apart from the above issues, the mounted camera on the above-mentioned kits is fixed and has very low resolution.

For many real-time applications like video surveillance, tracking etc. there is a need for interfacing a higher resolution camera or a pan-tilt-zoom (PTZ) camera and other custom peripherals with a high performance processor. Thus, to implement a complex image and video processing algorithm there is requirement of a high-end device based FPGA platform, which can perform such applications competitively. Therefore, for video acquisition and display, we have selected Xilinx ML-507 platform [**33**], which has Virtex-5 FX FPGA device [**87**] and the necessary peripherals needed for image and video processing applications. The Virtex-5 FX series FPGAs are optimized for embedded processing and memory-intensive applications with high-speed serial connectivity. Hey have a high-performance embedded PowerPC 440 processor [**34**] which can be used to implement area-efficient embedded systems. To handle high-resolution video, the platform requires a high-end camera, to be interfaced with the platform. For this interfacing, the platform requires some specific configurations.

As discussed above, in the image and video processing system the image acquisition block plays a vital role of capturing the incoming video [**2**,**88**,**89**] and thus, determines the overall performance of the system [**20**,**90**,**91**,**92**,**93**]. In [**17**] a frame grabber has been used to capture and exchange video data with the hardware co-processor. In this approach, a LabVIEW-based graphical development environment is used to control the frame grabber. In another approach to capture still frames from an analog video camera, an FPGA platform based image acquisition module is used [**94**]. In this method, a frame grabber card, which is a daughter card to the FPGA board, is interfaced through an I/O port of the FPGA platform. An embedded platform for image acquisition and processing has been developed by [**91**] which

uses a combination of DSP and FPGA devices. In their prototype, a Texas Instrument's digital signal processor (TMS320C6416T) and an Altera FPGA (EP3C25F324) are used. In [**92**] a real-time image acquisition and VGA display system has been realized on DE2 development board. This work is based on Cyclone II series FPGA (EP2C35F672C6) available on Altera DE2 development and education board as the core control device. Along with the platform resources, a Terasic CMOS image sensor (TRDB-D5M) has been used for the hardware configuration. Some of the real-time implementation issues in embedded image processing using FPGA-based architectures are presented in [**93**]. In this work, an Altera Stratix FPGA (EPISF1020C7) device has been used to implement a smart camera platform. In [**20**] an architecture is presented for image acquisition and processing using a CMOS sensor for the smart camera system. In their design a Xilinx Virtex-4 FX (XC4VFX12) FPGA based platform has been used along with embedded PowerPC 405 microprocessor.

In this chapter, we present our work on the module for video streaming, acquisition and display on a VGA monitor using Xilinx ML-507 FPGA-based platform. The module provides an essential common architectural block for realizing most of the practical image and video processing applications. The ML-507 platform provides a video input video codec peripheral, which supports the capturing of a real-time video. Similarly, to display-out the results, there is a display controller chip available on the platform. The details of video codec and display controller peripheral chips are covered in Appendix-B. To achieve the goals of acquiring and then displaying out the video, the platform requires appropriate configurations of these peripheral chips. The VGA input video codec peripheral is programmed through inter-integrated circuit (I2C) bus protocol so that the input video can be accessed by the hardware blocks realized in the FPGA fabrics. To interface a VGA/DVI display monitor, the control registers of the display controller peripheral are programmed through I2C bus. The

interfacing of a camera and a display unit to the platform set it up completely for the video streaming applications.

This chapter additionally focuses on the architectural arrangement for real-time image acquisition and its display. Here, real-time video in RGB analog format is captured from an analog PTZ camera. The captured video is converted into frames and buffered in the DDR2 SDARM memory. This requires a multi-port-memory controller (MPMC). The stored frames can be retrieved and converted back into the standard VGA resolution of 640×480 and displayed on the VGA monitor. This module is an essential predecessor to any image and video processing application. In the design, we stream the video frames on an individual basis, buffer the frames in the external DDR2 SDRAM memory and display the buffered frames through the hardware cores in FPGA fabric on VGA monitor in real-time. The embedded PowerPC 440 processor, available on the Xilinx Virtex-5 FX FPGA device is used to configure the platform peripherals.

This module is essential for developing any complete real-time video processing system, which grabs image or video, processes it and shows the result on display. The module can be utilized in a wide range of applications such as, image barcode recognition [**74**], change detection [**95**], edge detection [**96**], face recognition [**97**] and object tracking [**5**,**82**] application described in Chapter 6. The architectural arrangement for image acquisition and display module presented in this chapter is also utilized for validating various hardware/software embedded video processing architectural units researched in this thesis such as units for image thresholding described in Chapter 4 and the unit for connected component analysis described in Chapter 5.

The organization of rest of the chapter is as follows: in Section 2.2, we describe the Xilinx ML-507 FPGA platform set up and its required configuration for making the platform

suitable for image and video processing applications. In Section 2.3, we describe a real-time embedded video streaming module and its hardware/software components. Section 2.4 covers an FPGA-based embedded architecture for acquisition of real-time video and its implementation in the Xilinx Virtex-5 FPGA device. Section 2.5 presents the results and Section 2.6 concludes the chapter.

2.2 Xilinx ML-507 Platform Configuration

The Xilinx ML-507 platform [**33**] has a Virtex-5 FX FPGA device [**87**] and the necessary peripherals needed for an image and video processing system. The Virtex-5 FX series FPGAs are optimized for embedded processing and memory-intensive applications with high-speed serial connectivity. They have a high-performance embedded PowerPC 440 processor [**34**] which can be used to implement area-efficient embedded systems. The dedicated memory interface port of the processor enables it to simultaneously access both the memory bus and Processor Local Bus (PLB) to maximize the throughput [**98**]. The Virtex-5 xc5vfx70t FPGA device has one PowerPC440 (PPC440) processor surrounded by the FPGA fabric [**33**], the details of which are given in Appendix-A. Appendix-A also illustrates about the embedded PPC440 processor. To use the ML-507 platform for the embedded image and video processing applications, the platform requires interfacing of an analog camera, a PAL to VGA converter, and a VGA monitor.

We have done the required platform configuration for the Xilinx ML-507 using embedded PowerPC processor. The PowerPC processor uses preconfigured I2C bus controller and the processor local bus (PLB) bus controller. The processor configures the control registers of the video input video codec. For accessing the Virtex-5 FXT FPGA embedded PowerPC processor a design has been created in Xilinx embedded development kit (EDK) using joint

test action group (JTAG), I2C, PLB and UART controller soft IPs. Similarly, the processor programs the control registers of the display controller chip.

Table 2.1: VGA Timings for Resolution Video

No. of pixels	Active	Front Porch (FP)	Sync	Back Porch (BP)	Total
Horizontal	640	20	96	44	800
Vertical	480	13	2	30	525

The details of the necessary components for performing the configuration are given in Appendix-B. Appendix-B also covers the VGA and the I2C bus protocols, which are used for the configuration of the VGA input video codec and the display controller peripheral available on the FPGA platform. In addition, Appendix-B also focuses on the programming of IDT clock generator, which is used for the generation of custom clock frequency [**99**]. In Table 2.1, the timing details of the VGA protocol for 640×480 video resolution at 60 frames per second are shown. The configuration process for the interfacing of VGA input video codec and the display controller peripheral chips on the Xilinx ML-507 FPGA platform is given below.

2.2.1 Configuration of VGA Input Video Codec

The ML-507 platform contains a VGA input video codec connector that supports connectivity to an external VGA source. The circuit-level arrangement of interfacing the VGA input video codec with the FPGA pin is shown in Fig. 2.1. Table 2.2 shows the I/O connections for the VGA input video codec. The addresses of AD9980 control registers and the configuration values are given in Table 2.3. The control registers of AD9980 is configured by sending data as a master on the I2C bus by writing application software in the 'C' programming language. This application software runs on top of a standalone software platform and uses the API provided by standalone software platform.

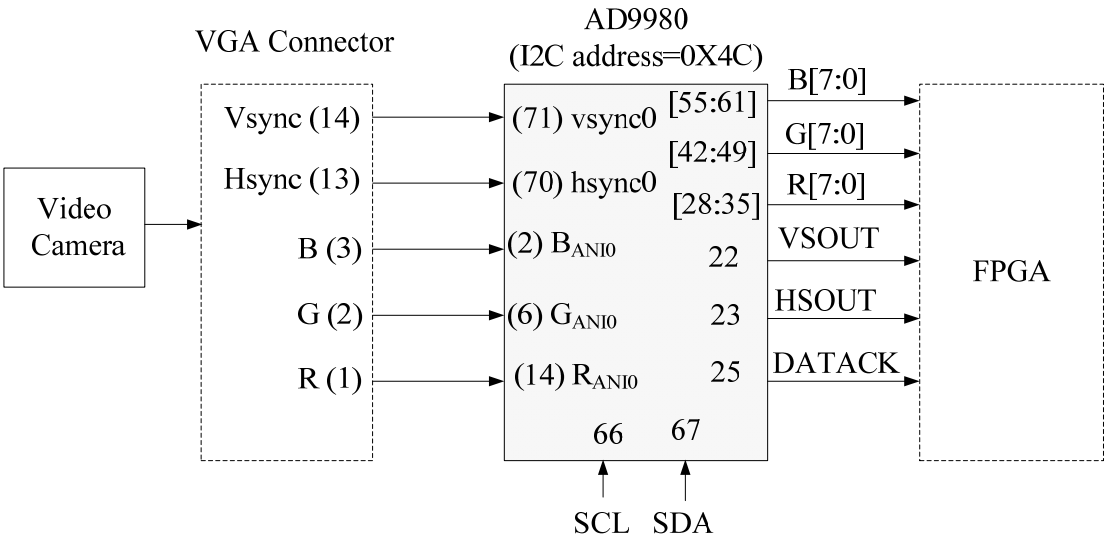


Fig. 2.1: AD9980 with an FPGA

Table 2.2: I/O Connection of VGA Input Video Codec with FPGA

Pin Type	Pin	Function	Pin No.
Inputs	R_{AIN0}	Channel 0 Analog Input for converter R	14
	G_{AIN0}	Channel 0 Analog Input for converter G	6
	B_{AIN0}	Channel 0 Analog Input for converter B	2
	HSYNC0	Horizontal Sync input for channel 0	70
	VSYNC0	Vertical Sync input for channel 0	71
Outputs	RED[7:0]	Outputs of converter R, bit-7 is the MSB	28-35
	GREEN[7:0]	Outputs of converter G, bit-7 is the MSB	42-49
	BLUE[7:0]	Outputs of converter B, bit-7 is the MSB	55-61
	DATACK	Data output clock	25
	HSOUT	Hsync output clock (Phase-aligned with DATACK)	23
	VSOUT	Vsync output clock	22
Control	SDA	Serial port data I/O	66
	SCL	Serial Port Data Clock	67

AD9980 is configured for 640×480 at 60 frames per second (fps) video resolution through programming of its internal registers. The details of each register are given in [**100**]. An I2C controller is used to write and read the control registers of the AD9980.

Table 2.3: Control Registers of AD9980

Address (Hex)	Value (Hex)	Address (Hex)	Value (Hex)	Address (Hex)	Value (Hex)
01	32	0A	00	14	18
02	00	0B	02	15	0A
03	48	0C	00	18	00
04	80	0D	02	19	04
05	40	0E	00	1A	1A
06	00	0F	02	1B	3B
07	40	10	00	1C	FF
08	00	12	18	2D	E8
09	40	13	60	2E	E0

An example of the configuration of control registers using XIic_DynSend function is shown in Fig. 2.2.

```
send_count=XIic_DynSend(BaseAddress,Address,*BufferPtr,2, Option);
if(send_count != 2 )
{
xil_printf("Error writing to address %02x\r\n", Address);
break;
}
```

Fig. 2.2: Configuration of control registers of the video input video codec using I2C API (XIic_DynSend).

where,

`BaseAddress` : Base address of the I2C Device

`Address` : 7 bit I2C address of the device to send the specified data

`BufferPtr` : Points to the data to be sent

`ByteCount` : Number of bytes to be sent

Table 2.4: FPGA Interface Pins of AD9980

Net Name	FPGA Pin	Net Name	FPGA Pin	Net Name	FPGA Pin	Net Name	FPGA Pin
R[0]	AG5	G[0]	Y8	B[0]	AC4	CLAMP	AH7
R[1]	AF5	G[1]	Y9	B[1]	AC5	COAST	AG7
R[2]	W7	G[2]	AD4	B[2]	AB6	EVEN_B	W6
R[3]	V7	G[3]	AD5	B[3]	AB7	VSOUT	Y6
R[4]	AH5	G[4]	AA6	B[4]	AA5	HSOUT	AE7
R[5]	AG6	G[5]	Y7	B[5]	AB5	SOGOUT	AF6
R[6]	Y11	G[6]	AD6	B[6]	AC7	DATACK	AH18
R[7]	W11	G[7]	AE6	B[7]	AD7	-	-

The configuration of the control registers of AD9980 is accomplished by using the XIic_DynSend function of I2C. The macro function, XIic_DynSend, sends the 7-bit address during both read and write operations. It sends data using polled I/O and blocks until the data has been sent. It takes care of the details to format the address correctly. This macro is designed to be called internally to the drivers for dynamic controller functionality. The FPGA pins for interfacing the Analog Devices AD9980 video decoder (VDEC) device [**100**] is shown in Table 2.4.

2.2.2 Display Controller Configuration

A DVI/VGA monitor can be interfaced with the Xilinx ML-507 platform by using a DVI connector present on the ML-507 platform [**101**]. The DVI connector uses Chrontel CH7301C DVI transmitter/display controller device [**101**]. To facilitate the display controller for accessing the FPGA pins, the circuit level arrangement is shown in Fig. 2.3. The FPGA device provides the digital graphics input signals to the CH7301C display controller device, which are subsequently encoded and transmitted to the DVI/VGA monitor. The CH7301C device accepts data over one 12-bit wide variable voltage data port, which supports different data formats including RGB and YCrCb. The CH7301C device is controlled through I2C bus.

The control registers of the device, which are programmed through the I2C bus, are shown in Table 2.5.

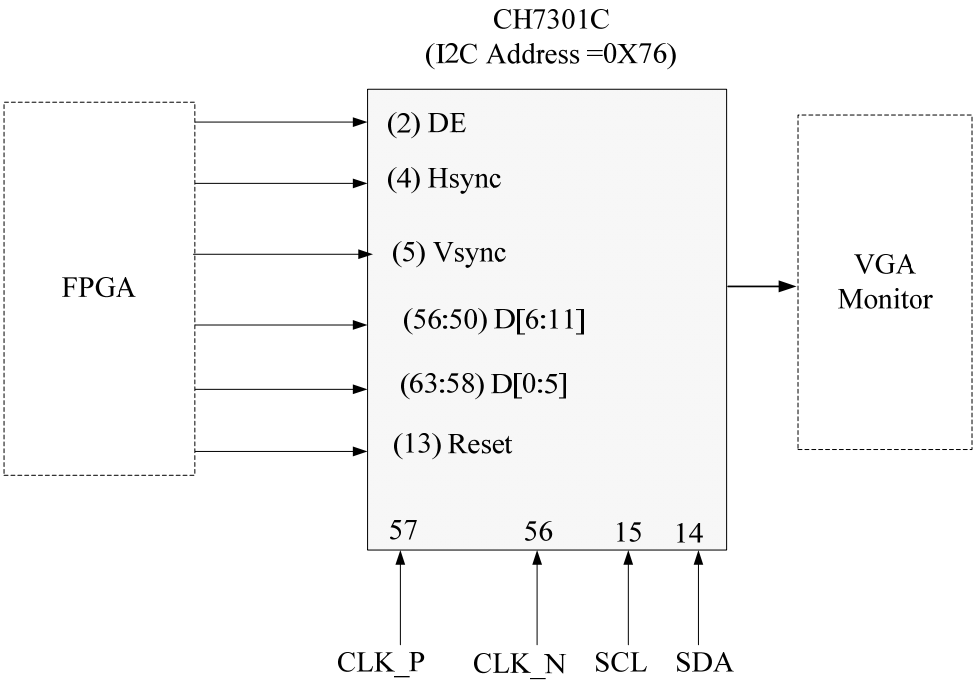


Fig. 2.3: CH7301C interface with the FPGA device.

Table 2.5: Control Registers Value of Chrontel CH7301C Device

Address(Hex)	21	2D	2E	33	34	36	49
Value (Hex)	09	E8	E0	08	16	60	C0

The signals, which are used with the FPGA device, are explained in the Table 2.6. Similar to the VDEC register configurations, the control registers of CH7301C are configured through IIC XIic_DynSend function. After configuring the IDT clock generator (as detailed in Appendix-B), the VGA input video codec and the display controller devices, the Xilinx ML-507 FPGA platform is all set for the image and video processing applications. To interface the display controller chip with the FPGA the pin configurations are shown in Table 2.7.

Table 2.6: CH7301C Chrontel Device Signals

Signal	Description
Data Enable (DE)	This pin accepts data enable signal, which is high when active video data is input to the device, and low all other times.
Horizontal Sync(H)	This I/O pin receives/sends out horizontal sync input from/output to the graphics controller.
Vertical Sync (V)	This I/O pin receives/sends out vertical sync input from/output to the graphics controller.
RESET	When this pin is low, the device is held in the power-on reset condition, otherwise reset is controlled through the serial port register.
SPD	This pin functions as the serial data pin of the serial port interface.
SPC	This pin functions as the clock pin of the serial port interface.
D[11:0]	These pins accept the twelve data inputs from a digital video port of a graphics controller.

Next section illustrates an extensible hardware-software video streaming module. This serves as a module in the general framework for all vision-based applications leveraging the features of reconfigurable platforms, which are necessary for vision systems like camera sensors and standard display ports. After configuring the platform peripherals, the platform is ready to capture the real-time video.

Table 2.7: CH7301C Interface with the FPGA

Net	FPGA Pin	Net	FPGA Pin
D[0]	AB8	D[9]	AB10
D[1]	AC8	D[10]	AP14
D[2]	AN12	D[11]	AN14
D[3]	AP12	CLK_P	AL11
D[4]	AA9	CLK_N	AL10
D[5]	AA8	HSYNC	AM12
D[6]	AM13	VSYNC	AM11
D[7]	AN13	DE	AE8
D[8]	AA10	RESET_B	AK6

2.3 Embedded Video Streaming Module

Embedded system architecture for video streaming application is developed around the Xilinx ML-507 FPGA platform. This optimal system integration makes use of the embedded PowerPC440 processor, integration of intellectual property (IPs) blocks along with the design of custom hardware realized in the FPGA fabric. The design in the FPGA is capable of internal image and video acquisition without the aid of an external frame grabber or any software running on an external computer. The design can work on a stream of image data coming into the FPGA from an external camera sensor. The incoming image data is organized into frames by the design internally and sent for the required processing and then on to the display unit. The system offers the requisite flexibility to design and implement embedded image and video processing applications. Here, the PTZ camera acts as a pure input sensor for the vision based application. The design decouples the processing from the image sensor to the FPGA and in that sense extends the functionality of the camera. The aim is to develop the core components of this design that are implemented in the FPGA and are part of the general underlying infrastructure of all vision-based systems and letting the applications build themselves naturally over these components.

The existence of an embedded processor in the FPGA provides the system with the flexibility to choose which parts of an image processing algorithm are to be implemented on the software (PowerPC 440) and rest in the hardware as custom design logic blocks in the FPGA fabrics. This flexible hardware/software system facilitates the development of a vision-based system. The FPGA has sufficient computational power and proves to be a suitable platform for developing complex applications over the lightweight acquisition, storage and display components built inside it. In this design PowerPC 440, embedded processor is used for the interfacing of FPGA-based custom modules and IPs along with the configuration of platform peripherals. Fig. 2.4 shows the block diagram of the design.

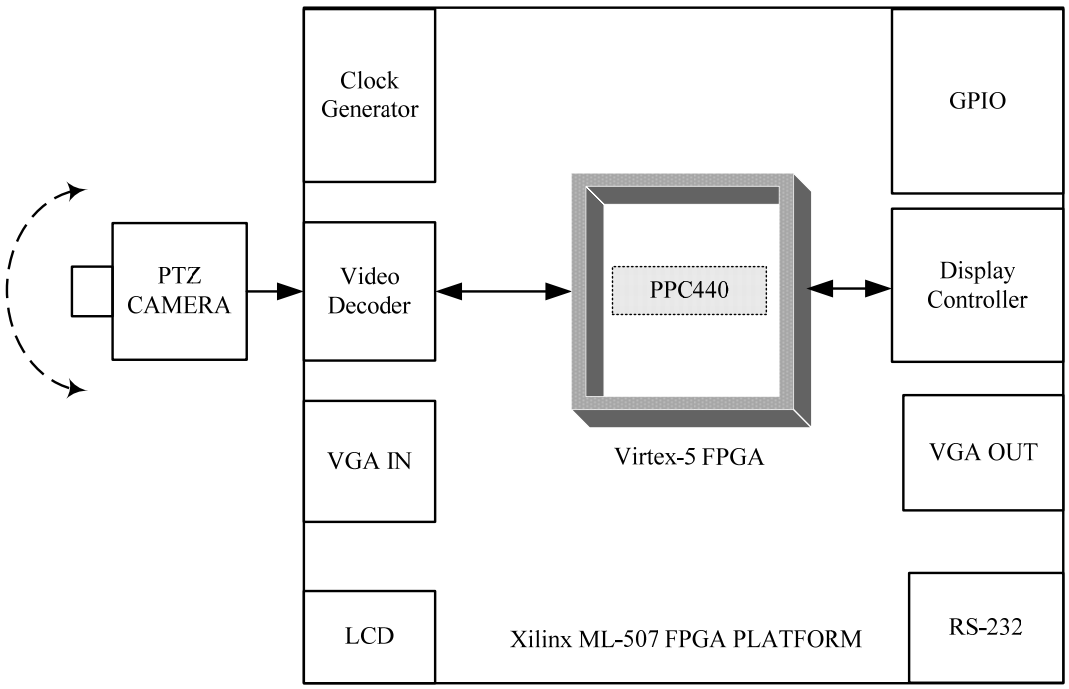


Fig. 2.4: The Xilinx ML-507 FPGA platform as an embedded vision platform.

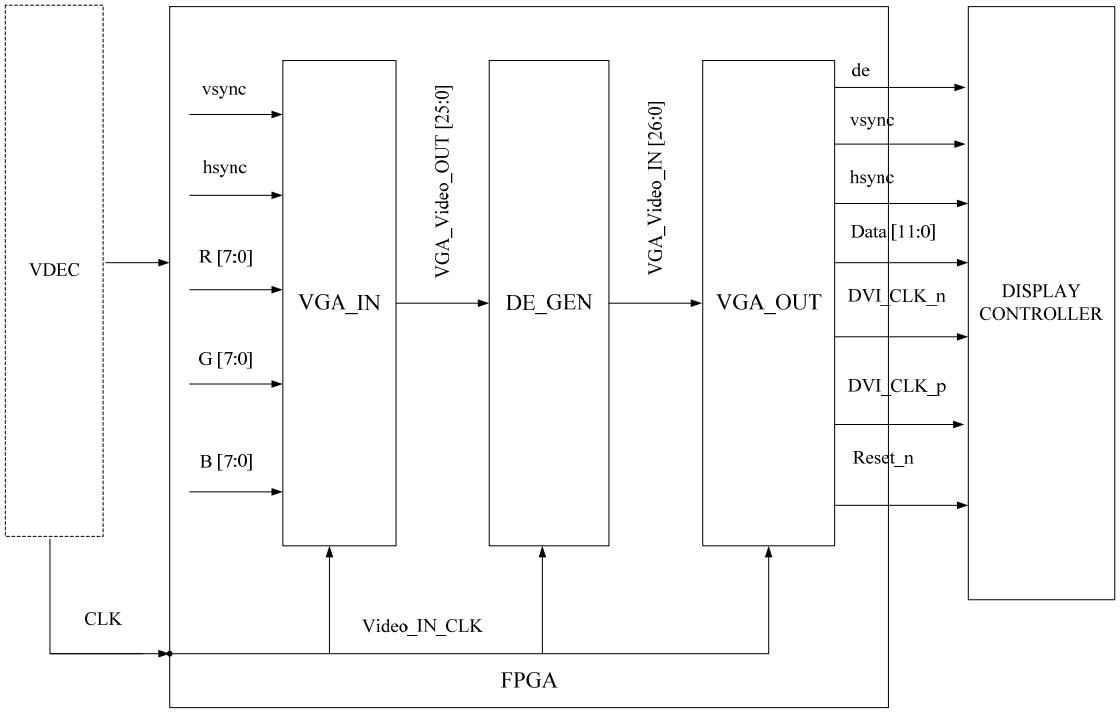


Fig. 2.5: Hardware blocks for the real-time video streaming.

The software environment of the system consists of application software and device drivers. The hardware part of the system includes the configurable logic blocks in the FPGA fabric. This integration of software and hardware provides the complete system functionality. The system requires interfacing of PTZ analog camera with the FPGA platform. For this interfacing VGA IN port is used. The video decoder chip registers are configured using the I2C bus according to the resolution and frame rate of the incoming video. This is achieved by using the I2C bus controller's low-level device driver functions.

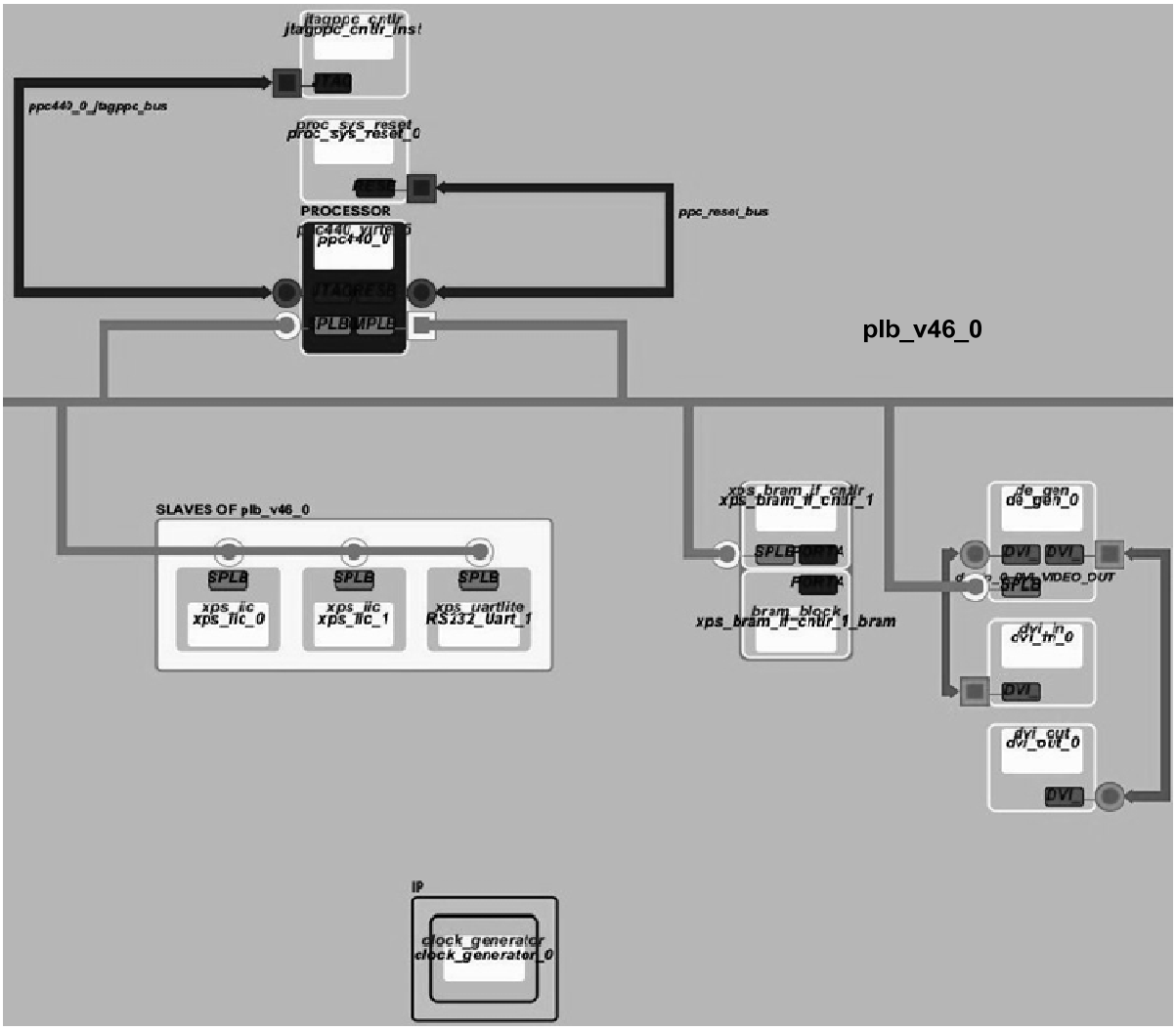


Fig. 2.6: Block-diagram of the embedded video streaming module in Xilinx EDK.

To interface a VGA monitor, DVI OUT port is used after configuring video display controller chip registers through the I2C bus. The required FPGA platform configurations for the above and other vision-based applications are explained in Appendix-B. In next section the details of the hardware components of the video streaming module are explained.

2.3.1 Hardware Components of the Video Streaming Module

A design is implemented in the FPGA logic which facilitates the streaming of video from camera to the monitor through the FPGA logic in real-time. For implementation of the design, Xilinx provided IPs, namely, digital clock manager (DCM), PLB, XPS I2C interface together with some of the IPs from Xilinx Spartan-3A DSP video starter kit [**45**] IPs are utilized along with the PowerPC 440 embedded processor. Hardware blocks used in the implemented system are shown in Fig. 2.5. Block diagram view of the design in the Xilinx EDK tool is shown in Fig. 2.6. A snapshot of the streaming video is shown in Fig. 2.7. Details of each module are described in the following subsections.

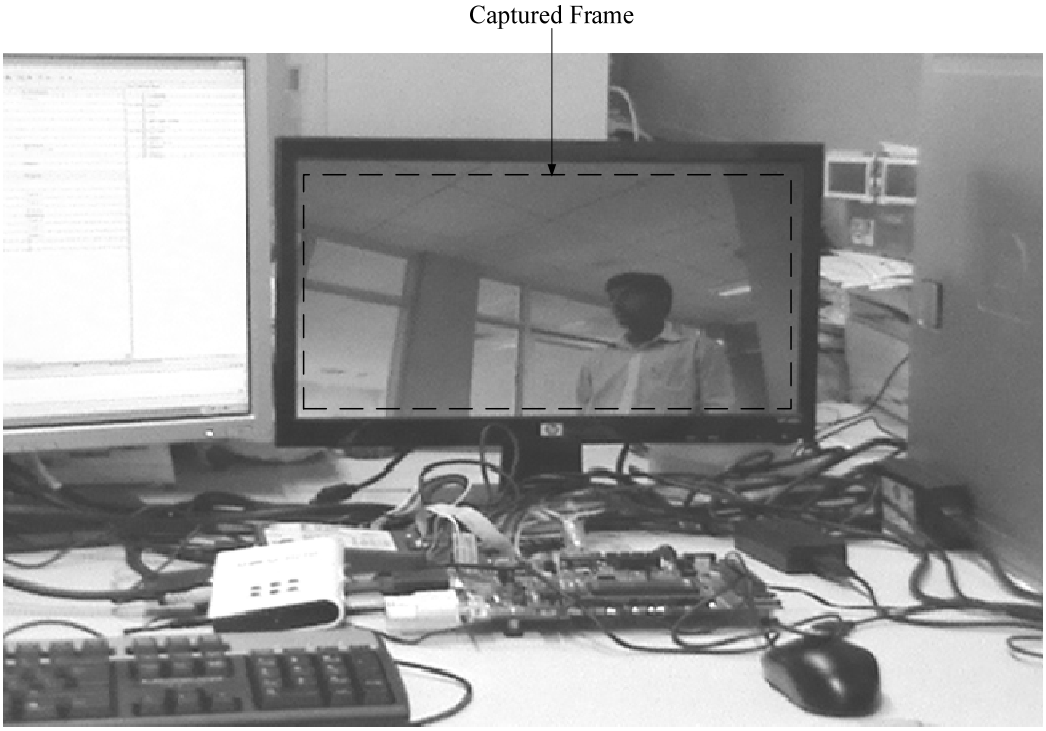


Fig. 2.7: A frame of size captured video from the video camera.

Table 2.8: VGA_IN I/O Signals

Signal Name	Direction	Interface	Description
vsync	I	To the FPGA pin	Vertical Sync input
hsync	I	To the FPGA pin	Horizontal Sync Input
red(7:0)	I	To the FPGA pin	Red input
green(7:0)	I	To the FPGA pin	Green input
blue(7:0)	I	To the FPGA pin	Blue input
clk	I	To the FPGA pin	Clock input(pixel rate)
ce	I	To the FPGA pin	Clock enable(Active High)
de_o	O	VGA_VIDEO_OUT	Data Enable output(Active video)
hsync_o	O	VGA_VIDEO_OUT	Vertical Sync output
vsync_o	O	VGA_VIDEO_OUT	Horizontal Sync output
red_o (7:0)	O	VGA_VIDEO_OUT	Red output
green_o (7:0)	O	VGA_VIDEO_OUT	Green output
blue_o (7:0)	O	VGA_VIDEO_OUT	Blue output

2.3.1.1 VGA_IN

VGA_IN peripheral core provides a connection to the AD9980 video decoder chip. This peripheral core brings in the input signals from the video input video codec chip, registers the signals, and groups the video signals into a unified bus that interconnects to other IPs for processing. Along similar lines, a bus interface called VGA_VIDEO_OUT is defined for the VGA_IN peripheral core outputs. The details of VGA_IN I/O signals are given in Table 2.8.

2.3.1.2 DE_GEN

DE_GEN peripheral core provides the ability to generate a data enable (de) signal for analog video streams. The data enable signal marks the beginning of the active video that needs to be written to the external memory. The DE_GEN core achieves this by analyzing the input hsync and vsync signals combined with the front porch and back porch clock cycles based on the VGA protocol. The PowerPC processor communicates the porch values to DE_GEN core over the PLB interface based on the video resolution. The vertical back porch value contains

the number of clock cycles that lie between the active edge of hsync and the first active video pixel. Thus, it includes the vertical back porch, the hsync pulse width, and the border preceding the first active video pixel.

The vertical front porch value contains the number of clock cycles that lie between the last active video pixel and the active edge of hsync. This includes the vertical front porch and the border following the last active video pixel. The horizontal back porch value contains the number of the lines of data that lie between the active edge of vsync and the first line of active video. This includes the horizontal back porch, the vsync pulse width, and the border preceding the first line of active video. The horizontal front porch value includes the number of lines of data that lie between the last line of active video and the active edge of vsync, which indicates start of a new frame. This includes the horizontal front porch as well as the border following the last line of active video. Bus interfaces called VGA_VIDEO_IN and VGA_VIDEO_OUT are defined for the de_gen peripheral core. The details of the DE_GEN I/O signals are given in Table 2.9.

Table 2.9: DE_GEN I/O Signals

Signal Name Bus	**Direction**	**Interface**	**Description**
Vsync	I	-	Vertical Sync input
Hsync	I	-	Horizontal Sync Input
red(7:0)	I	-	Red input
green(7:0)	I	-	Green input
blue(7:0)	I	-	Blue input
clk	I	-	Clock input(pixel rate)
ce	I	-	Clock enable(Active High)
de_o	O	VGA_VIDEO_OUT	Data Enable output(Active video)
hsync_o	O	VGA_VIDEO_OUT	Vertical Sync output
vsync_o	O	VGA_VIDEO_OUT	Horizontal Sync output
red_o (7:0)	O	VGA_VIDEO_OUT	Red output
green_o (7:0)	O	VGA_VIDEO_OUT	Green output
blue_o (7:0)	O	VGA_VIDEO_OUT	Blue output

2.3.1.3 VGA_OUT

VGA_OUT peripheral core provides a connection to the CH7301C DVI transmitter device. This peripheral core takes in the VGA_VIDEO_IN bus (that is driven by the VGA_VIDEO_OUT port of the DE_GEN core) and formats the video data to the format required by the display controller device. Details of the VGA_OUT I/O signals are given in Table 2.10.

The CH7301 is capable of driving either the DVI displays or the analog VGA displays. For analog displays, the de signal is required. The ML-507 platform has a DVI port as the output port, so the digital video interface signals are generated by the dvi_out core. A DVI-to-VGA converter is used externally in case of analog displays. In the next section, we describe the embedded video acquisition and display module.

Table 2.10: VGA_OUT I/O Signals

Signal Name	Direction	Interface	Description
de_i	I	-	data enable input
vsync_i	I	-	Vertical Sync input
hsync_i	I	-	Horizontal Sync Input
red_i(7:0)	I	-	Red input
green_i(7:0)	I	-	Green input
blue_i(7:0)	I	-	Blue input
clk	I	-	Clock input(pixel rate)
ce	I	-	Clock Enable(Active High)
reset	I	-	Reset(Active High)
de	O	To the FPGA pin	Data Enable output
hsync	O	To the FPGA pin	Vertical Sync output
vsync	O	To the FPGA pin	Horizontal Sync output
VGA_data(11:0)	O	To the FPGA pin	Data Output
VGA_clk_p	O	To the FPGA pin	VGA Clock(Positive Phase)
VGA_clk_n	O	To the FPGA pin	VGA Clock(Negative Phase)
reset_n	O	To the FPGA pin	VGA Reset(Active Low)

2.4 Embedded Video Acquisition and Real-Time Display

As discussed in the Section 2.1, for extracting the meaning from streaming video, a selective image and video acquisition module is necessary. The video frame that needs to be processed requires a frame buffer memory. The frame buffer is a memory that is used to hold video frames, which require further processing. The acquisition module stores the required frames in the frame buffer as per the needs of the particular application. The image and video acquisition module provides data from the extracted video frames to the application-specific data processing unit, which, in turn, processes the data as per the required application algorithm and provides necessary control signals to the camera to capture the subsequent frames of interest.

The amount of memory needed to retain the frames depends primarily on the video resolution and per pixel color depth. Following formula provides the amount of video memory needed for particular video resolution with known per pixel color depth.

$$\text{Video memory} = \text{X-resolution} \times \text{Y-resolution} \times \text{Number of bits per pixel} \quad (2.1)$$

In standard VGA video resolution of 640×480 pixels, each pixel is represented by 32-bits. Thus, one video frame requires around 2 MB of memory. Therefore, to make a frame buffer there is a requirement of large memory space. The available Block RAMs, which can store up to 36 K bits of data in the Xilinx Virtex-5 FPGA, do not suffice for this purpose. Apart from the memory size limitation, these memory elements are utilized for other fast logic realizations in the design. Therefore, in our design a 256 MB DDR2 SDRAM memory available on the Xilinx ML-507 platform is used for the frame buffer application. Our design routes the frame of video from camera to the monitor through the DDR2 SDRAM memory and the FPGA logic in real-time.

The top-level system architecture is shown in Fig. 2.8. It consists of a Xilinx ML-507 FPGA platform, a Sony PTZ camera [**102**], a PAL to VGA converter [**103**], and a VGA monitor for displaying the output video. It uses the video input video codec (VDEC), and the display controller (DC) peripherals of the Xilinx ML-507 platform.

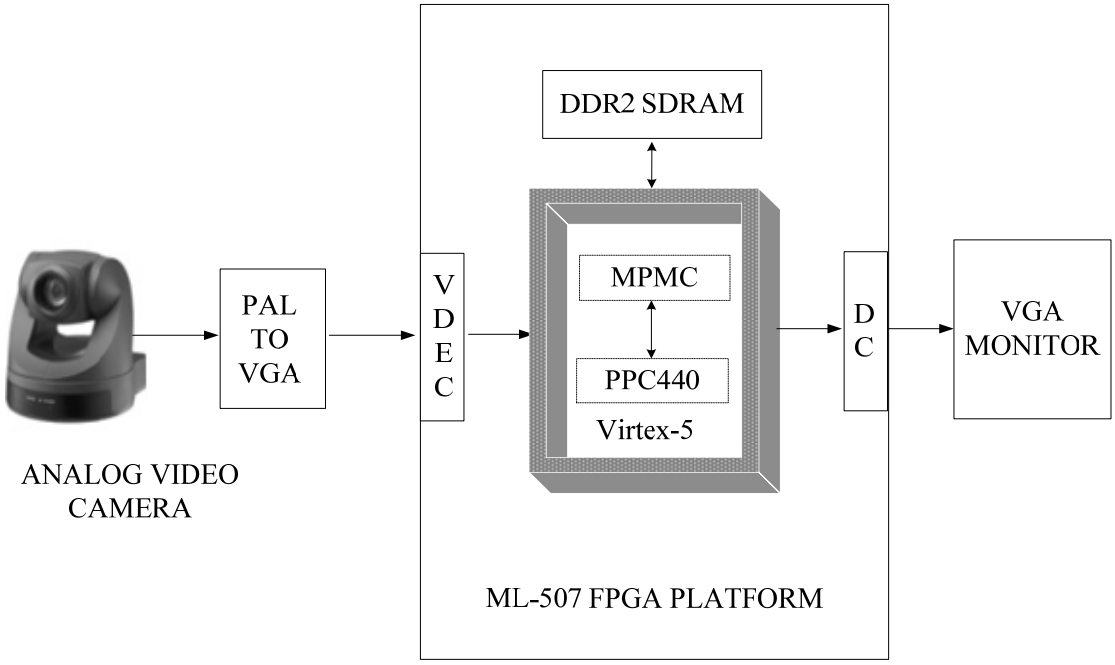


Fig. 2.8: Development platform set-up for embedded video acquisition.

2.4.1 The System Architecture

The system architecture of video acquisition, storage and display system is shown in Fig. 2.9. The software environment of the system consists of application software and device drivers. The hardware part of the system includes the configurable logic blocks in FPGA. This integration of software and hardware provides the complete system functionality.

In this design PowerPC 440 embedded processor is used for the interfacing of FPGA-based custom modules and IPs along with the configuration of platform peripherals. The video input video codec chip registers and the video DVI transmitter chip registers are configured by using the I2C bus controller's low-level device driver functions for the resolution of

640×480 @60 frames/sec. Programmable clock generator is used to provide the custom clock for the VGA display. Subsequently, the design is implemented in the FPGA logic which facilitates the streaming of video from camera to the monitor through the FPGA logic in real-time. Details of the hardware building blocks and the IPs used are discussed below.

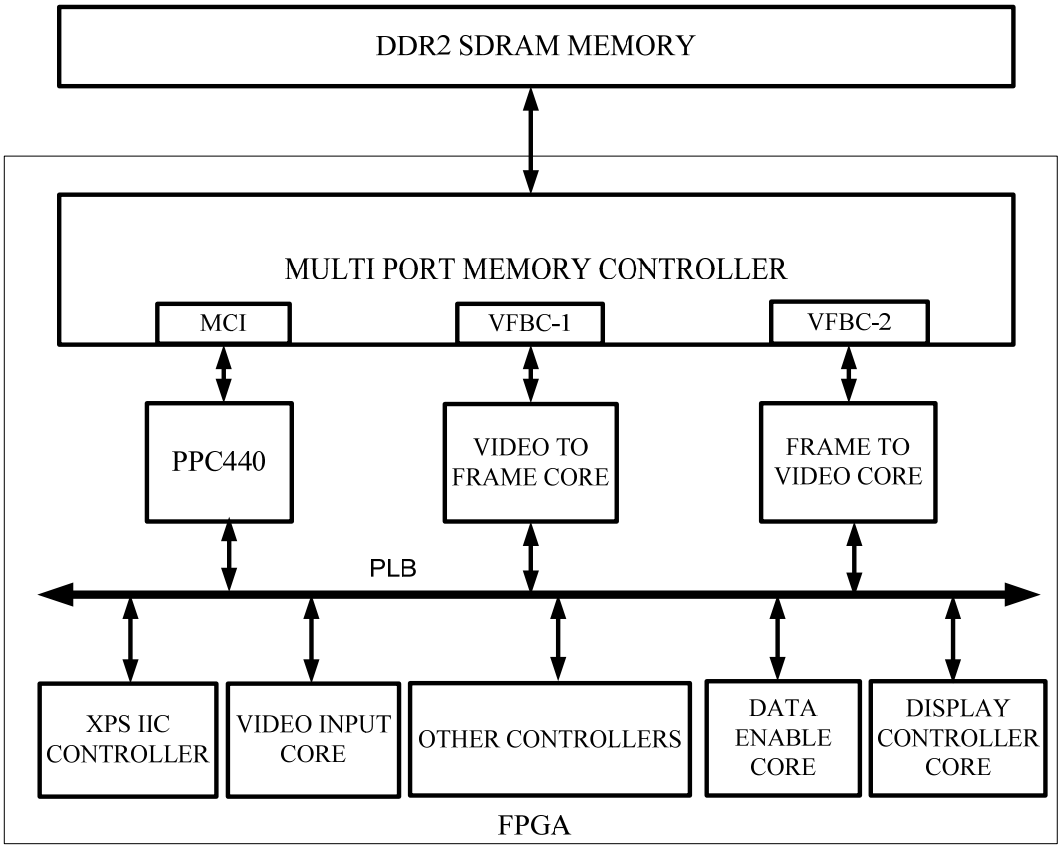


Fig. 2.9: System architecture for video acquisition.

To interface a DVI monitor, DVI OUT port of the ML-507 platform has been used after configuring the on-board video display controller chip registers [**101**] through the *I2C* bus. The application software is written in C language and it runs on the Xilinx-provided *standalone* software platform [**104**]. Further, it uses the developed APIs as needed and also utilize the required ones from among those provided by the software platform. In this embedded architecture, peripherals like video input video codec (VDEC), display controller (DC), and some of the Xilinx provided IPs, such as, multi-port-memory controller (MPMC)

[**105**], digital clock manager (DCM) [**106**], Xilinx Platform Studio (XPS) I2C controller, along with some of the Xilinx Spartan-3A DSP video starter kit [**45**] IPs are used. For bussing this variety of IPs, the architecture uses two bus protocols. The 128-bit processor local bus (PLB) protocol [**98**] provides the infrastructure for connecting an optional number of PLB masters and slaves into an overall PLB system. The second bus is the memory controller interface (MCI) which provides an interface between PowerPC 440 microprocessor and a soft memory controller implemented in the FPGA logic [**34**].

The arrangement of a video frame in the DDR2 memory is shown in Fig. 2.10. Here, each color R, G and B requires 8-bit memory storage. The last byte remains zero. One pixel representation, this requires 32-bit storage. The row and column is defined as,

$$(r,c) = (\text{Row No.} \times \text{No. of Column} + \text{Col. No.}) \times 4 \tag{2.2}$$

where, r= Row number of pixel in the memory

c= Column number of the pixel in the memory

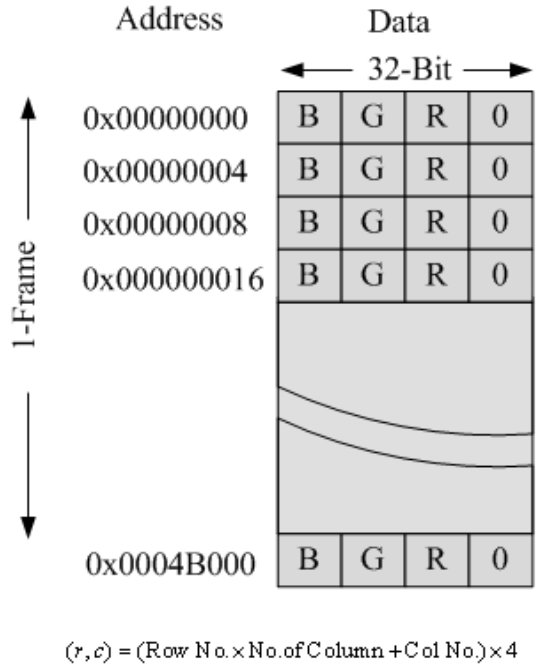


Fig. 2.10: A video frame in the DDR2 SDRAM.

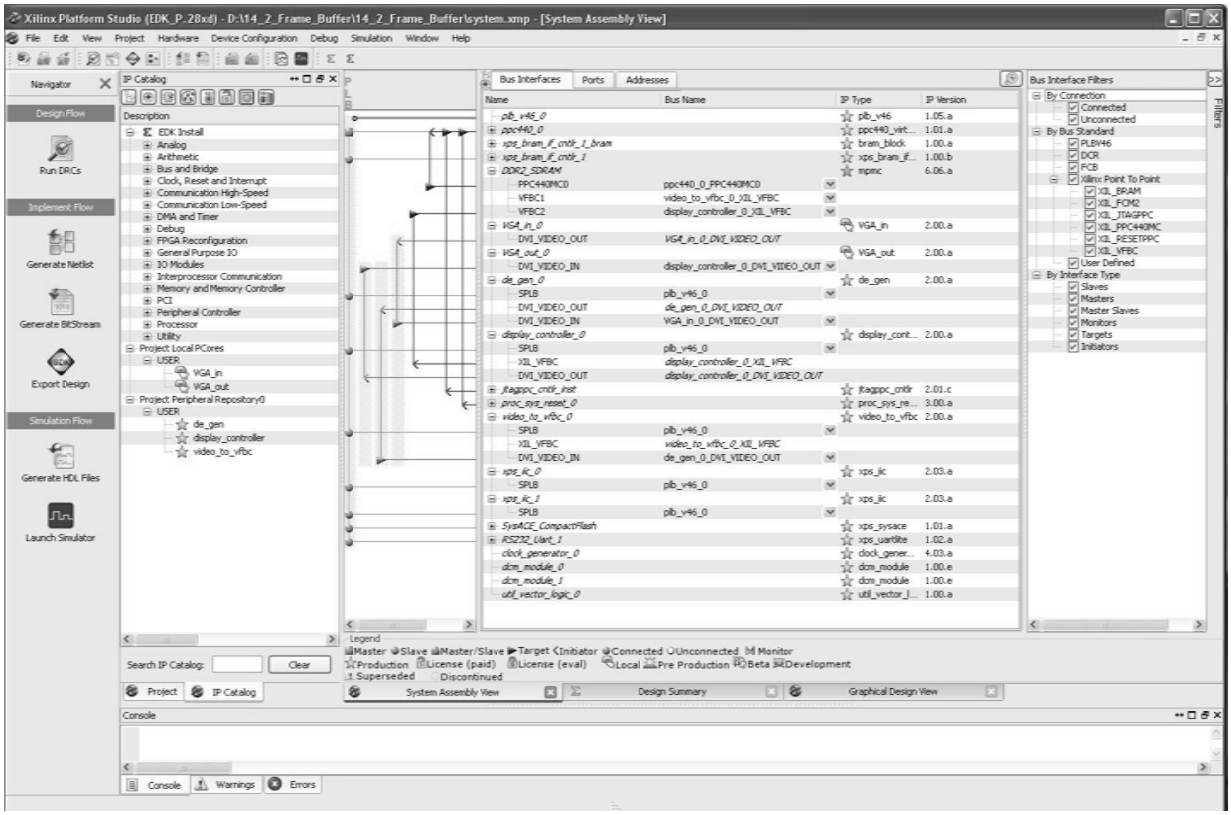


Fig. 2.11: System assembly view of the video acquisition design in Xilinx EDK.

All the required interconnections among the architectural blocks are made in the Xilinx embedded development kit (EDK) design environment. A snapshot of the EDK system assembly view of the design is shown in Fig. 2.11. Here, the PowerPC embedded processor is selected to control the peripherals and IPs. The design uses PLB, BRAM, UART, I2C and DCM controllers. Custom design IPs include VGA_IN and VGA_OUT. To control the DDR2 SDRAM memory, the design uses MPMC controller with its three active ports namely MCI, VFBC-1 and VFBC-2. By using all the above IPs and the PLB bus, the system block diagram using the EDK design tool is shown in Fig. 2.12. This figure shows the integration of various modules for achieving the real-time frame acquisition in DDR2 SDRAM memory, which is interfaced through the MPMC controller. The design uses two instances of the I2C controller, which controls the video read and video display processes. The UART controller displays the PowerPC execution sequences in a hyper terminal on the host PC.

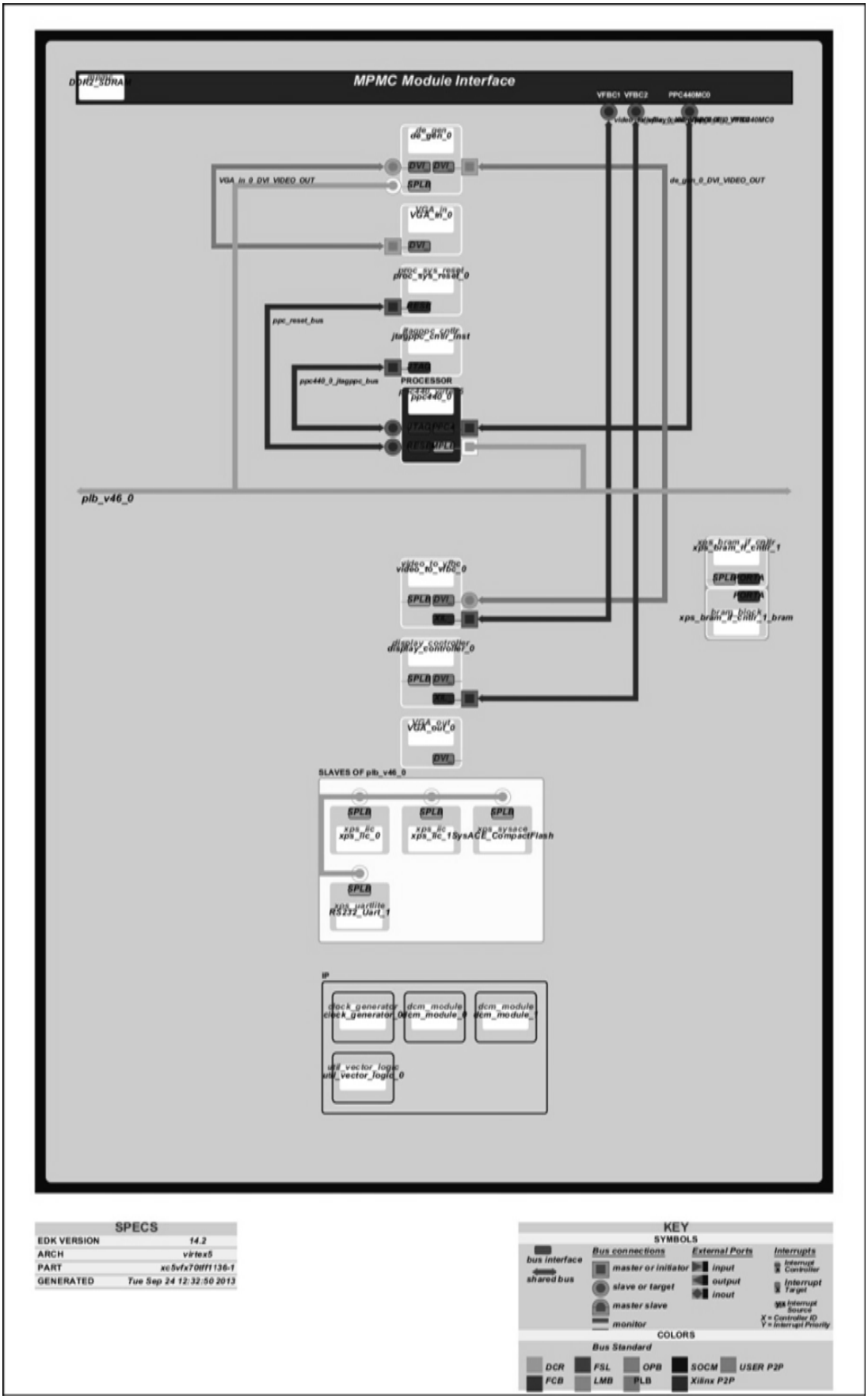


Fig. 2.12: EDK block diagram view of the video acquisition design.

The graphical view of the design is shown in Fig. 2.13. As apparent, apart VGA-IN, DE_GEN and VGA_OUT hardware cores, the architecture utilizes a few more hardware cores like MPMC, video to frame, frame to video core and display controller cores. Further, details of each core are provided in the following sub-sections.

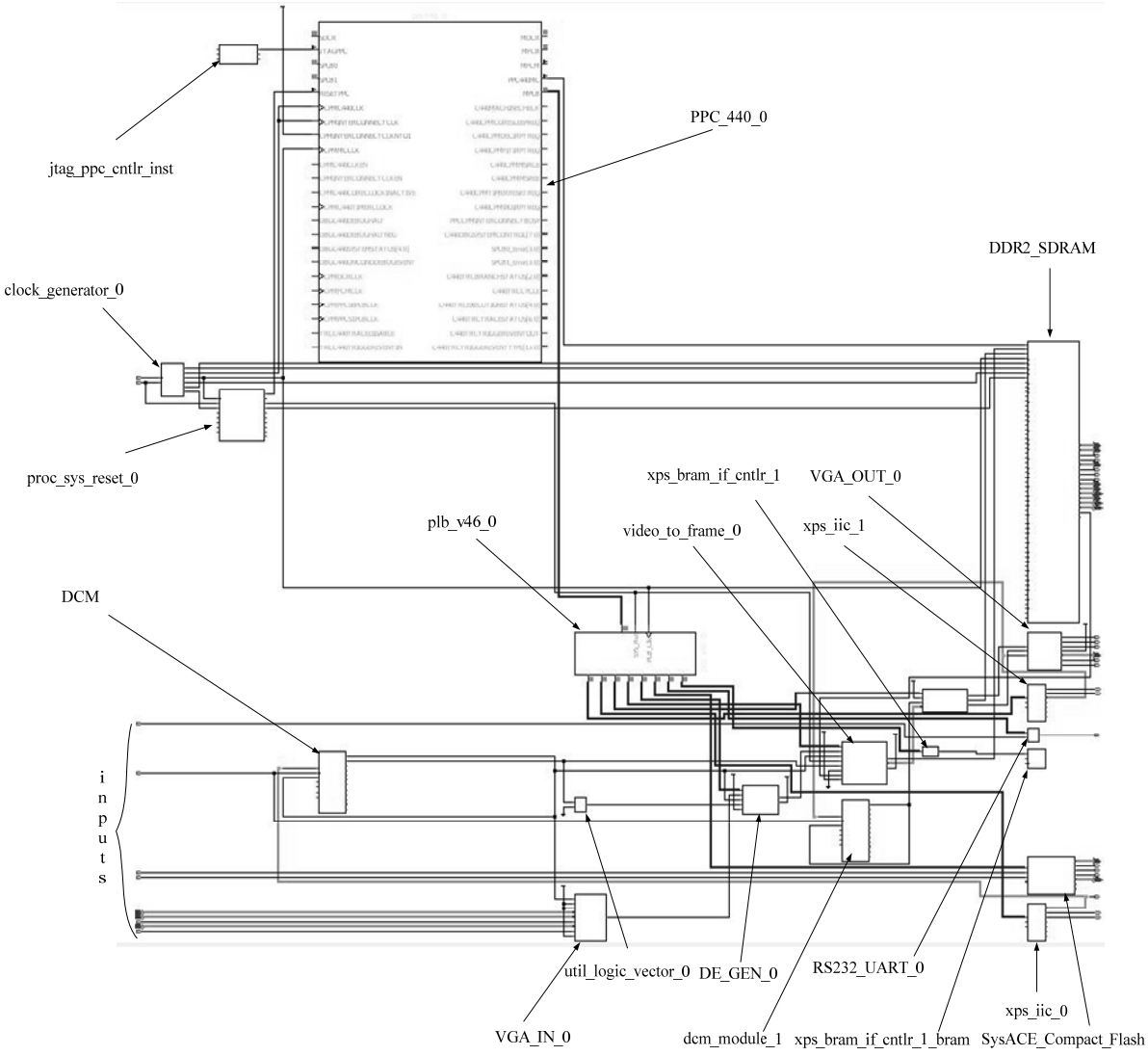


Fig. 2.13: EDK graphical view of the video acquisition design.

2.4.1.1 Multi-port Memory Controller (MPMC)

The MPMC is a parameterizable memory controller that supports DDR2 SDRAM [**105**]. MPMC provides access to memory for one to eight ports. It has been used for interfacing to DDR2 SDRAM. Video frame buffer controller (VFBC) is a special interface for video frame

data and is an essential part of the multi-port memory controller. It is used in video applications where hardware control of two-dimensional (2D) data is needed to achieve real-time operation. The VFBC allows a user-defined Intellectual Property (IP) to read and write data in 2D sets regardless of the size or the organization of external memory transactions. It has separate asynchronous first-in first-out (FIFO) interfaces for write data input, command input and read data output.

2.4.1.2 Video to Frame Core

The video to frame peripheral core controls the storing of video frames into frame buffers. It writes video data to the VFBC interface on the MPMC memory controller. The video to frame core is connected with DDR2 SDRAM via the multi-port memory controller and it works in synchrony with the VGA_IN and DE_GEN cores.

2.4.1.3 Frame to Video Core

The frame to video peripheral core reads video frames out of memory. It provides pixel clock of 25.175 MHZ to the display controller peripheral core to display digital video resolution of 640×480 video on the DVI/VGA monitor. The frame to video core retrieves the active video data from the DDR2 SDRAM memory via VFBC interface of the MPMC controller. The fetched data is used by the display controller unit.

2.4.1.4 Display Controller Core

The display controller peripheral core provides a connection to the CH7301C DVI transmitter device. This peripheral core accepts the external clock signal generated by the IDT clock generator along with the output data of frame to video peripheral core and formats the active video data to the format required by the DVI transmitter device. The ML-507 platform has a DVI port as the output port. A DVI-to-VGA converter is used for the display on a VGA monitor.

2.4.2 System Validation

The system arrangement to validate the real-time video acquisition, storage and display system is shown in Fig. 2.14. In this arrangement, the ML-507 platform is connected with host computer running EDK, via a platform cable USB II and a RS-232 serial cable. The RS-232 connection is made to observe execution of the C program running on PPC440 processor on the hyper-terminal. A VGA monitor is connected to the platform through the DVI OUT port by using DVI to VGA converter.

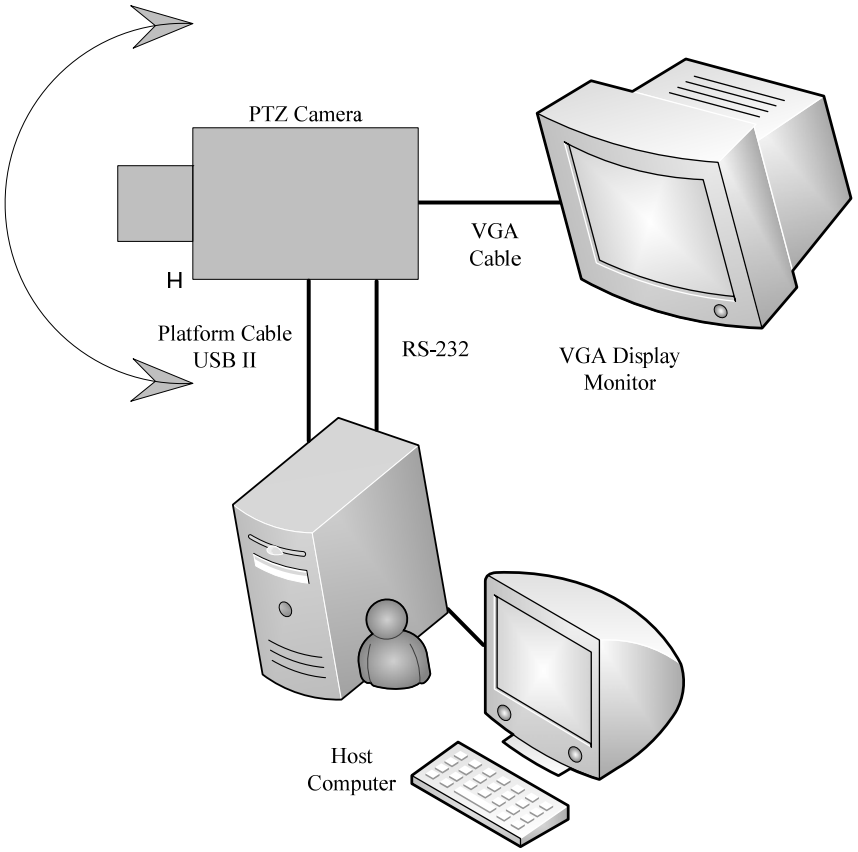


Fig. 2.14: System arrangement to validate the real-time video acquisition.

2.5 Results

The real-time video is captured from the PTZ camera, which is interfaced via a PAL to VGA converter with the Xilinx ML-507 platform. The VGA timing details of the design as obtained from Xilinx ChipScope Pro analyzer for 640×480 @60 fps video resolution is shown in Fig. 2.15.

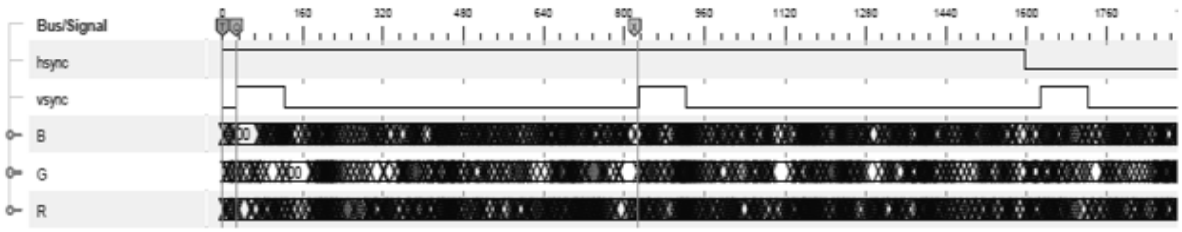


Fig. 2.15: Timing details of the video acquisition design obtained from Xilinx ChipScope Pro analyzer.

The Fig. 2.15 shows the timing details of vsync, hsync, R, G and B signals. The captured video is converted into a set of frames by using customized logic in FPGA fabric. The stored frames are converted back into VGA format by using customized FPGA logic, which are displayed on a VGA monitor using Xilinx ML-507 platform. In the timing diagram, the vsync signal shows the control of the VGA monitor to start displaying a new image or a new frame of a video. The hsync signal controls the monitor to refresh another row of 640 pixels. The video signal redraws the entire screen 60 times per second.

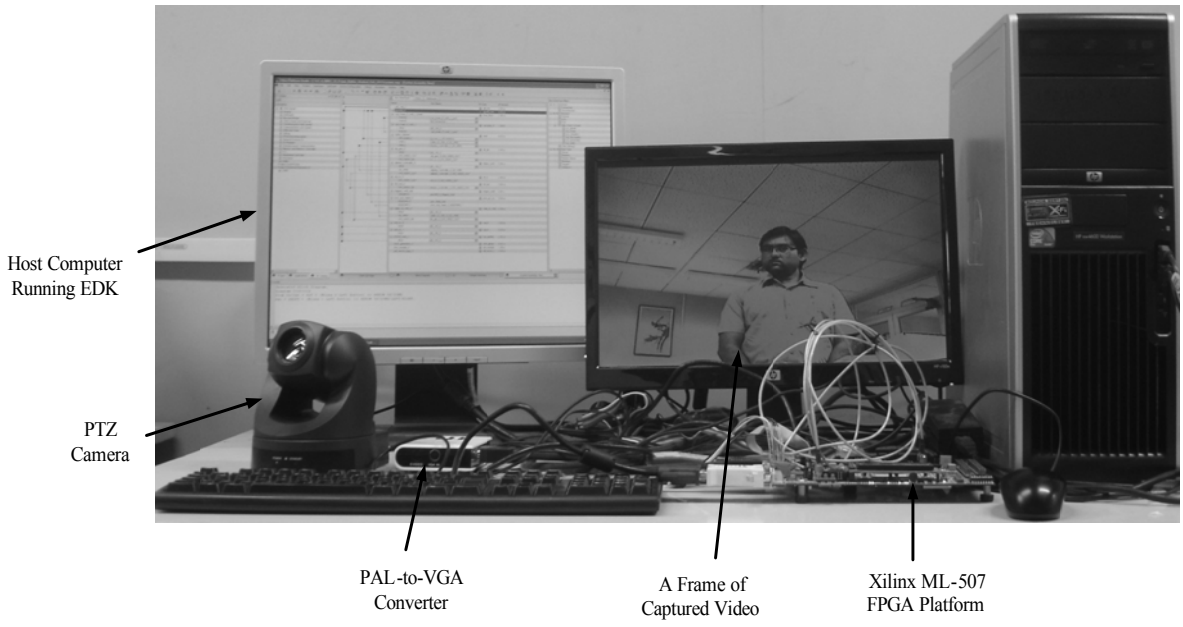


Fig. 2.16: Complete system set-up for embedded video acquisition.

Real-time video in RGB analog format is captured from the PTZ camera. The captured video is converted into the frames and buffered into DDR2 SDARM memory using MPMC. The stored frames are converted into VGA resolution of 640×480 and displayed on the VGA monitor. The architecture uses Xilinx ML-507 FPGA platform. A captured video frame with the complete set-up of the design is shown in Fig. 2.16.

The total device utilization summary of the design is shown in Fig. 2.17. As apparent, the design uses mainly, the available PowerPC processor, 20 % of the slice registers, 17 % of the slice LUTs and 19 % BRAMs.

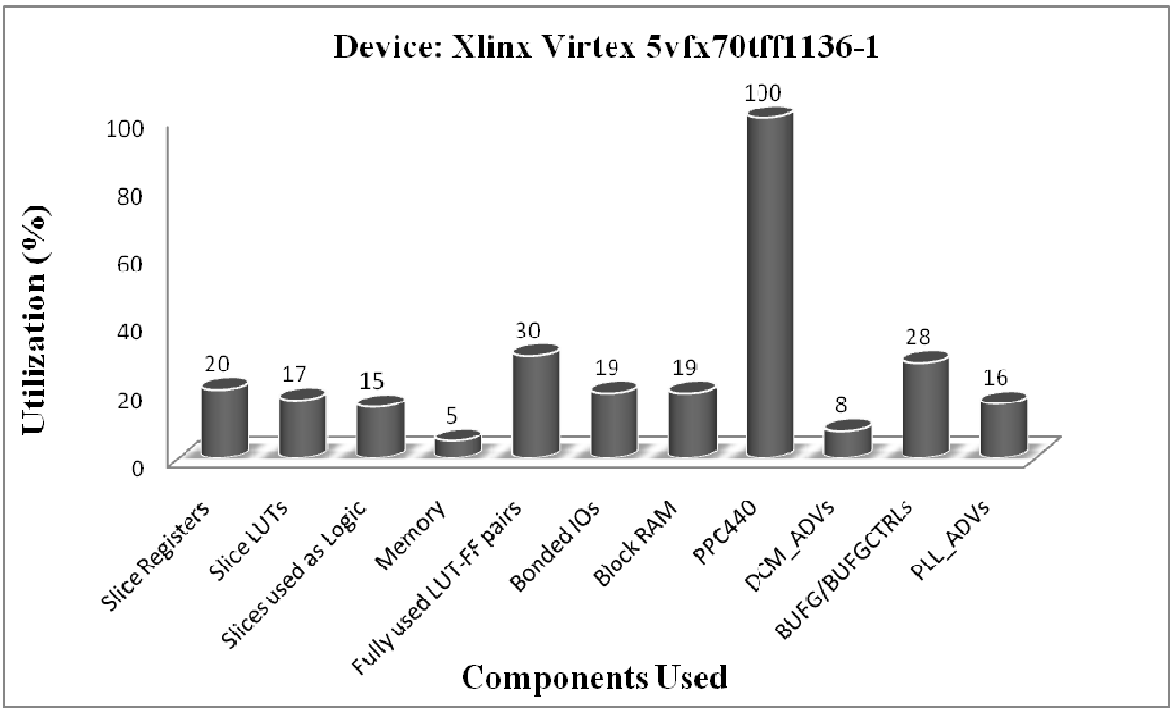


Fig. 2.17: Total device utilized in the video acquisition design.

Fig. 2.18 shows the slice utilization of each module in the FPGA fabric. From the total device utilization summary, it is evident that, apart from the PowerPC 440 processor, the total FPGA resources utilized are only to extent of eighteen percent (18%). The unutilized FPGA resources are sufficient for implementing many practical real-time video processing applications.

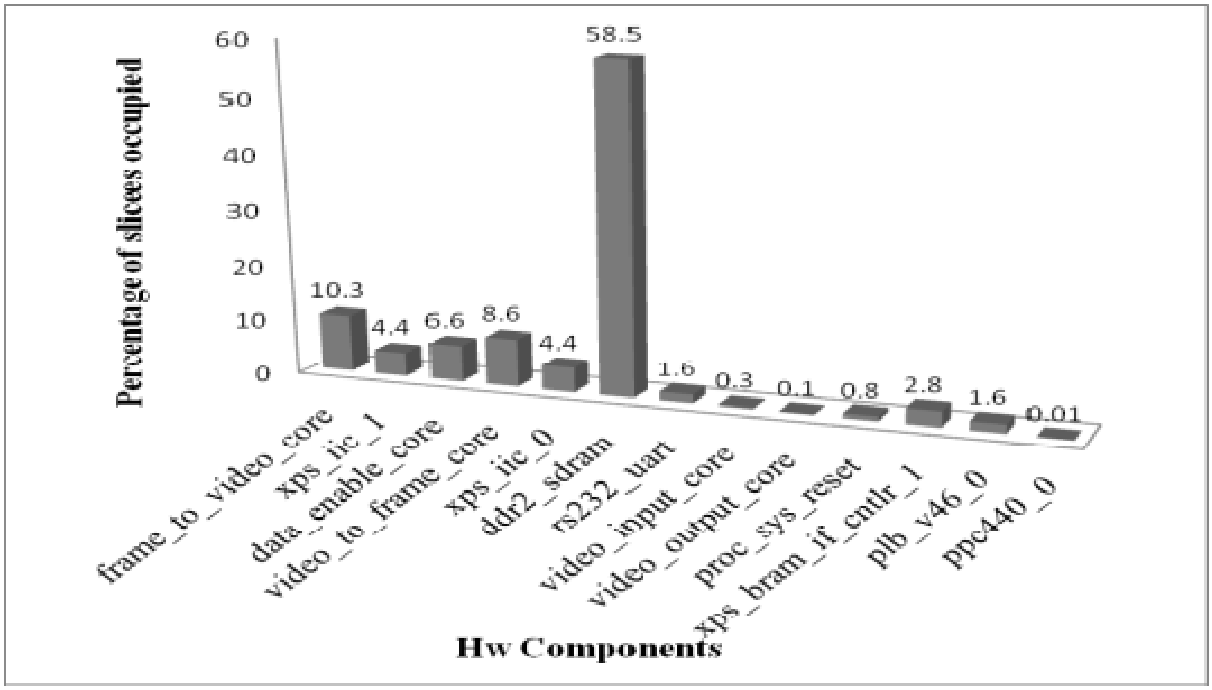


Fig. 2.18: FPGA slice utilization of each module for the embedded realization of video acquisition.

2.6 Conclusion

We have demonstrated a platform-based design approach for an image/video streaming application. In the developed extensible hardware-software video streaming module, we stream the frames on an individual basis through the FPGA fabric using custom designed hardware IPs in real-time. It enables the camera to be used in a variety of real-time applications.

We have also demonstrated an embedded design for video acquisition and display, which is a predecessor to any image and video processing application. In this design, we stream the video frames on an individual basis, buffer the frames in the external DDR2 SDRAM memory and display the stored frames through the hardware cores in FPGA fabric on a VGA monitor in real-time. The embedded PowerPC 440 processor, available on the Xilinx Virtex-5 FX FPGA device, is used to configure the platform peripherals for both the above designs.

CHAPTER 3

HARDWARE REALIZATIONS OF LOGARITHM AND ANTILOGARITHM FUNCTIONS

3.1 Introduction

Real-time data processing applications such as image and video processing, multimedia, digital signal processing (DSP) and 3D graphics require area-efficient complex arithmetic elements like divider, square-root, exponential, powering and inverse square-root circuits [**48**,**107**]. These computing elements should be fast, area-efficient and low power. The straight hardware implementation of these complex circuit elements is slow, power hungry and takes large silicon area. A representative graph, which shows the time consumption of arithmetic operations in 3D graphics processor, is shown in Fig. 3.1 [**47**]. As apparent, the division (DIV) operation takes around eighty-one percent (81 %) of the total computation time.

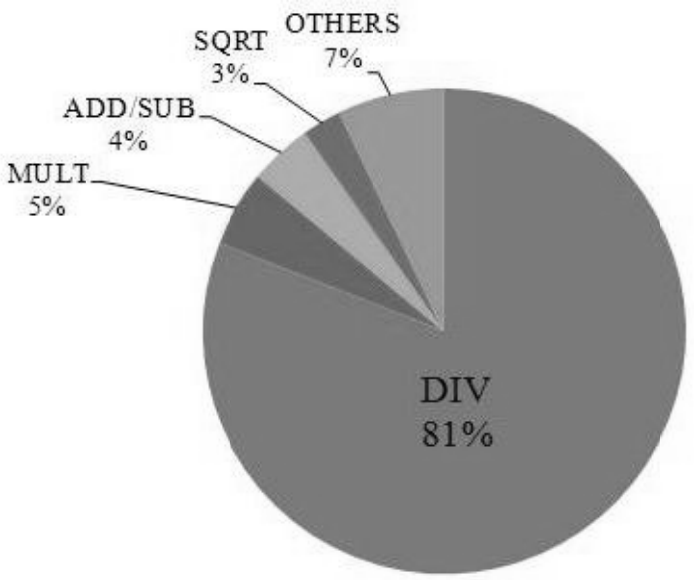


Fig. 3.1: Time consumption of arithmetic operations in a 3D graphics processor. Adapted from [47].

The complex functions, such as division, square root, exponential, power and inverse square-root can be easily realized through the logarithmic and antilogarithmic computational circuits [**48**]. The logarithmic number system (LNS) simplifies complex arithmetic operations into simple arithmetic operations such as, addition/subtraction and shifting operations. The mechanism of this simpler arithmetic approach is shown in Fig. 3.2.

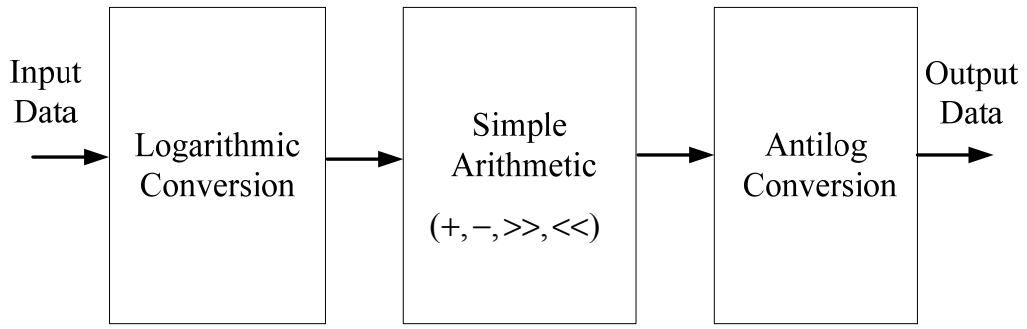


Fig. 3.2: A simple arithmetic approach for realizing complex arithmetic functions.

The mathematical expressions for the realization of complex arithmetic functions are shown in Table 3.1. This simplicity and improvement in design metrics are obtained at the cost of conversion overheads from integer to logarithmic and vice versa, yet the overhead is much smaller and it is bearable for realizing most of the practical embedded systems [**48**,**108**,**109**].

Thus, the hardware realization of logarithm and antilogarithm functions is of paramount importance, not to mention their usefulness in implementing other important complex arithmetic operations [**48**,**108**]. For hardware implementation, field programmable gate array (FPGAs) is one of the most promising candidates where many predefined and pre-fabricated components, such as dedicated adder, multiplier, embedded memories and embedded processors are available along with plenty of logic resources within a single FPGA device [**34**,**87**]. The FPGA macro elements can be utilized for the basic hardware building blocks, like RAM, adder, multiplier [**87**]. Usually, the elements of the FPGA are available for non-

floating point data path, as the need of floating point data type is very specific and consumes large amount of logic resources. These pre-fabricated elements can be used by incorporating fixed-point data type [**47**,**110**,**111**] which leads to a low-cost, fast and energy efficient circuit implementation. A datapath incorporating the fixed-point arithmetic unit can be implemented using minimal FPGA resources. Thus, we can make compact, fast and power saving hardware architecture using minimal logic resources.

Table 3.1: Complex Arithmetic Operations using Logarithmic Number System

Operation	**Representation**	**Normal Arithmetic**	**Binary Logarithmic Arithmetic**
Division	DIV	x / y	$\log_2 x - \log_2 y$
Reciprocal	RICP	$1/x$	$-\log_2 x$
Square root	SQRT	$\sqrt{x}$	$\log_2 x \gg 2$
Reciprocal square root	RSQR	$1/\sqrt{x}$	$-\log_2 x \gg 2$
Square	SQR	x^2	$\log_2 x \ll 2$
Powering	PWR	x^y	$y.\log_2 x$

In this chapter, two architectures are proposed for the hardware realization of logarithmic and antilogarithmic functions, which are subsequently realized in the FPGA. The architectures are based on piecewise approximation methods for binary logarithm and antilogarithm functions. The fixed-point number system is employed for implementing these architectures. The architecture of logarithm computation is capable of finding approximate logarithm of an integer number, integer with fractional number and only fractional number. The architecture uses the same set of circuit elements for all computations.

The architecture for antilogarithm computation, works for both positive and negative binary numbers. In the proposed architecture, a unique barrel-shifter is designed which shifts the input data to the left or right by the given count. To validate the approximation efficiency,

error analysis with thousands of uniformly distributed numbers is performed. The proposed architectures are then implemented in the Xilinx Virtex-5 xc5vfx70t FPGA device.

The logarithm unit designed in this chapter is utilized for the implementation of image thresholding algorithm discussed in Chapter 4. In addition, computational blocks such as square root and divider required for kernel-smoothed local histogram computation discussed in Chapter 6 are realized in hardware utilizing these logarithmic and antilogarithmic blocks using the concept of LNS. Further, the Bhattacharya coefficient computation, center of gravity computation and computation of mean shift based object tracking algorithm discussed in Chapter 6 primarily rely on the hardware blocks developed in this chapter. These realizations are explained in subsequent chapters of this thesis.

The rest of this chapter is structured as follows: Section 3.2 presents the piecewise linear approximation methods for computing binary logarithm and antilogarithm. The formats of fixed-point number system are explained in Section 3.3. Section 3.4 presents the proposed architecture of the binary logarithmic approximation unit along with all its constituent architectural building blocks. This section also covers the error analysis performed. Section 3.5 provides the details of the FPGA implementation of the proposed binary logarithmic unit. Section 3.6 presents the proposed architecture of the binary antilogarithmic approximation unit along with all the constituent architectural building blocks. This section also presents the error analysis results of the proposed binary antilogarithmic approximation unit. The FPGA implementation results of the proposed architecture are illustrated in Section 3.7. Finally, Section 3.8 concludes this chapter.

3.2 Approximation Methods for Computing Binary Logarithm and Antilogarithm

To compute binary logarithm and antilogarithm the popular computational methods used are as follows: The first method is the straight-line approximation method as suggested by

Mitchell [**111**]. The second method is comprised of the piecewise linear approximation methods, given in [**48**,**108**,**109**,**112**,**113**] and the approximation method followed by error correcting method [**109**]. The piecewise linear approximation method is suitable for an area-efficient implementation. In the proposed work, for the computation of logarithm and antilogarithm of a binary number the piecewise linear approximation method is used, which is explained below.

3.2.1 Binary Logarithmic Approximation Method

Let B be a binary number in the range $2^J \le B < 2^{K+1}$, $j = (-1,-2,-3,...,J)$, $k_l = (0,1,2,3,...,K)$ and $k_l \ge j$. Here, B can be expressed as: $B = b_{k_l} \cdots b_4 b_3 b_2 b_1 b_0 \cdot b_{-1} b_{-2} b_{-3} b_{-4} b_{-5} \cdots b_J$. The number B can be further written as:

$$B = \sum_{i=J}^{K} 2^i b_i \tag{3.1}$$

where b_i = '0' or '1'. Let b_{k_l} be the most significant leading-one bit, i.e., $b_{k_l} = '1'$. Now the number B can be written as:

$$B = 2^{k_l} \left(1 + \sum_{i=J}^{k_l - 1} 2^{i-k_l} b_i \right) \tag{3.2}$$

Let,

$$f_l = \sum_{i=J}^{k_l - 1} 2^{i-k_l} b_i \tag{3.3}$$

Since $k_l \ge j$, f_l will be in the range $0 \le f_l < 1$. Therefore, the number B becomes, $B = 2^{k_l}(1 + f_l)$. Now, by taking the binary logarithm of this equation we can get, $\log_2 B = k_l + \log_2(1 + f_l)$. Thus, the characteristic part (integer) of $\log_2 B$ is simply k_l and the mantissa part (fractional) is the term $\log_2(1 + f_l)$.To obtain logarithm of a fractional

number we used a shifting method. Let x be a fractional binary number and we have to calculate it's logarithm. When the number x is left-shifted by n bits (n=16) it becomes $(x << n) = x.2^n$. Let the shifted value is represented as x', so, $\log_2 x' = \log_2 x - n$.

3.2.2 Approximation Method for Antilogarithm Computation

Let X be a binary number in the range $2^{-16} \le X < 2^4$ (1.4.16 fixed-point format). Let k_a represents the integer (characteristic) part with most significant bit as the sign bit and f_a represents the fractional part (mantissa) of the fixed-point binary number X. The value $X[20] = 0$ represents that the input binary number is a positive number and if $X[20] = 1$ the input number is a negative number. Based on the fixed-point number format, the computation of antilogarithmic value is given in (3.4):

$$\text{Antilog}(X) = 2^X = 2^{k_a}.2^{f_a} \tag{3.4}$$

Depending on the sign bit, the k_a and f_a values of (3.4) are modified. Here, in piecewise linear approximation the fractional data (f_a) is approximated in the range of $0 \le f_a < 1$. When the data is negative it goes outside the above range, we simply subtract the fractional part from '1', and the integer part is decremented by '1'. By this, the same approximation is also used for the negative binary numbers. The modified values of k_a and f_a can be incorporated to obtain the antilogarithm and (3.4) can be written as,

$$\text{Antilog}(X) = 2^X = \begin{cases} 2^{k_a}.2^{f_a} & \text{sign bit} = 0 \\ 2^{k_a - 1}.2^{1 - f_a} & \text{sign bit} = 1 \end{cases} \tag{3.5}$$

Based on the piece wise linear approximation method, the fixed-point datapath is used for the computation of binary logarithm and antilogarithm computation. The formats of the fixed-point number are given below.

3.3 Fixed-Point Number Formats for the Proposed Architectures

The fixed-point number system can be used in place of a floating-point number system [**47**,**48**]. The hardware architecture for fixed-point arithmetic is much simpler as compared to that for floating-point arithmetic, as fixed-point arithmetic uses only integer datapath. Therefore, the fixed-point unit requires less area and hence it consumes less power [**47**]. Further, the hardware architecture of the fixed-point arithmetic can be easily implemented in a small FPGA fabric. Along with this, we can also use the available optimized FPGA macro elements, which are customized for the desired arithmetic operations at higher clock frequencies. For the implementation of the binary logarithmic architecture, a 16.16 fixed-point format is used which is shown in Fig. 3.3.

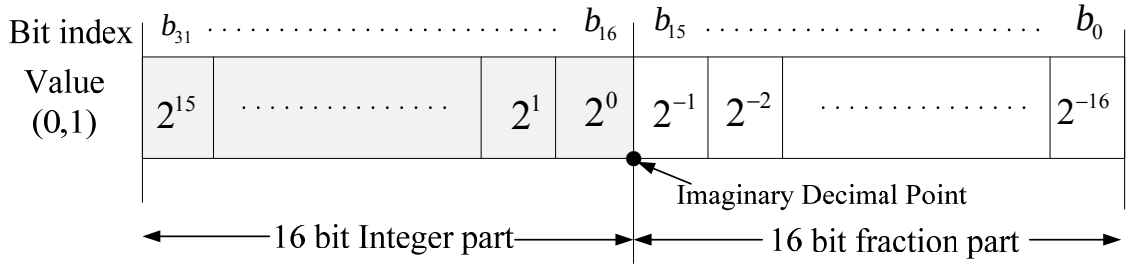


Fig. 3.3: Fixed-point number format for the binary logarithm computation.

Similarly, for the implementation of datapath for the proposed antilogarithm architecture, a 1.4.16 fixed-point format is used, which is shown in Fig. 3.4.

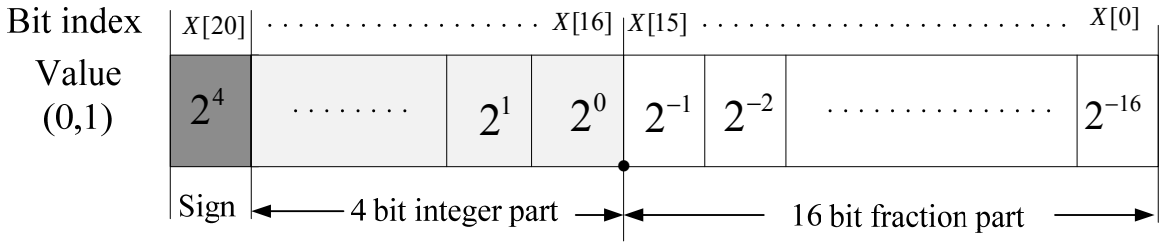


Fig. 3.4: Fixed-point number format for the binary antilogarithm computation.

3.4 Binary Logarithmic Approximation Circuit and the Proposed Architecture

Mitchell introduced a binary logarithmic conversion algorithm [**111**]. It is demonstrated that by using binary logarithmic operation, the multiplication and division can be modified in the form of a simple binary addition (or subtraction) operation. The other mathematical operations such as squaring, powering, reciprocal etc. can also be derived by incorporating the binary logarithmic and antilogarithmic units [**48**] which is given below.

The straight-line approximation of binary logarithmic as proposed by [**111**] requires an error-correcting stage. To improve the functional accuracy level of Mitchell's algorithm some VLSI architectures have been proposed [**48**,**108**,**109**,**112**]. In most of these approaches, the logarithmic curve is divided into a number of different regions and the piecewise straight line approximates each region.

A two-region approximation is presented in [**112**]. A four-region linear approximation with look-up table (LUT) based residual error correction stage that compensates for the piecewise interpolation error is presented in [**88**]. In [**108**,**114**] the two, three and six regions are considered. A CMOS VLSI implementation of a 16-bit logarithmic converter is proposed in [**114**]. A CMOS VLSI implementation of 32-bit binary-to-binary logarithm converter is presented in [**88**]. A region approximation scheme for binary logarithmic conversion is presented in [**108**]. It illustrates a CMOS VLSI implementation of a logarithmic computation circuit. All the above methods use straight-line segments to approximate the precise logarithmic curve such that the values of constant and slopes in each region of the intervals become multiple of powers-of-two integers, so that the hardware cost of the interpolation is minimal. The truncated fractional part is used to correct the approximation error.

3.4.1 The Proposed Architecture

As explained in the approximation approach (Section 3.2.1), the characteristic part (k_l) can be easily generated by incorporating a leading-one finder (LOF) block whereas the fractional part approximation (FPA) unit obtains the mantissa part. The block diagram of the binary logarithmic computation scheme is shown in Fig. 3.5.

Here, the LOF block represents a 16-bit leading-one finder circuit, which receives 16-bit input and provides a 4-bit encoded output containing the position of leading-one bit in the 4-bit binary number format. The bits after the leading-one position are applied to the FPA unit, which provides the approximated fractional part of the input number. The outputs of LOF and FPA units are combined which provides the binary logarithmic of the input number. One extra bit (*S*) represents the sign of the result.

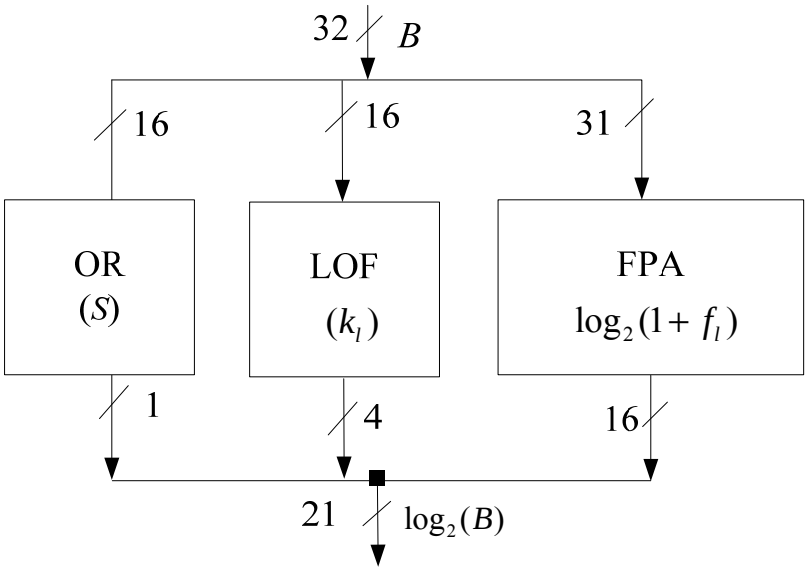


Fig. 3.5: Binary logarithmic computation scheme.

Based on the above concept, we propose an area-efficient architecture of a binary logarithmic approximation unit. The proposed architecture utilizes fixed-point data format and is capable of finding its binary logarithm in the range $(2^{-n} \le N \le 2^n - 1)$ with n =16. Along

with the FPGA fabric, the datapath of the proposed architecture uses FPGA off-the-shelf component such as, adder and multiplier (DSP48E). The proposed architecture is able to find out the binary logarithm of a 16-bit integer number, 16-bit fractional number or a 16.16-bit fixed-point number. The error analysis is performed for both the cases, with only fractional number in the range of $0 \leq f_l < 1$ as well as with the fixed-point number. The implementation results show the presented architecture is simple and area-efficient i.e. it consumes very few FPGA slices. The error analysis up to the five places of decimal depicts that the proposed architecture has 0.05 % error with 16.16 fixed-point numbers and 0.34 % with fractional number (f_l). This error is minimal and it is bearable for a practical embedded system.

In the proposed architecture, the eight-region piecewise linear approximation is used as in [**48**]. The approximation coefficients are stored in the eight locations of an 18-bit ROM. The top-level view of the proposed architecture is shown in Fig. 3.6.

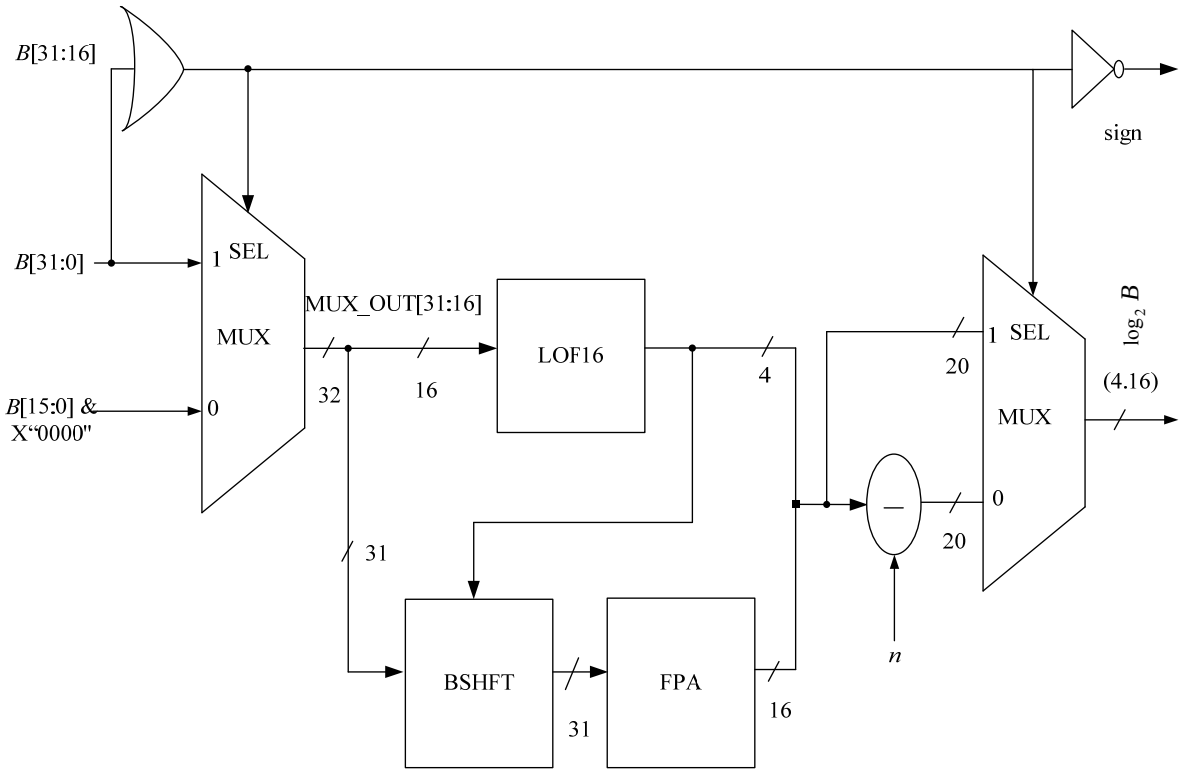


Fig. 3.6: Proposed architecture for the binary logarithmic computation.

Here, the OR gate (five 4-bit OR gate network makes a 16-bit OR gate) receives sixteen bits of the input *B*[31:16]. As discussed in the fixed-point format representation, the upper sixteen bits of the input contain the integer part and the lower sixteen bits contain the fractional part. If the input number has an integer part, the OR gate output will be '1' otherwise the OR gate output will be '0'. The output of the OR gate is provided to a 32-bit, 2-to-1 multiplexer select line which selects either the input word or the fractional part with appended zeros. Based on the OR gate output the input multiplexer routes the selected data to the leading-one finder (LOF16) and to the barrel shifter (BSHFT) circuit [**108**,**114**]. The LOF16 circuit receives the upper 16-bits of the multiplexer output, which are examined for leading-one. The internal detail of the LOF16, BSHFT and FPA blocks are discussed below.

3.4.2 Leading-One Finder (LOF) Circuit

The leading-one finder (LOF) is a 16-bit circuit. Usually a normal leading-one finder searches for the leading-one serially from MSB to LSB, which is a slow process as shown in Fig. 3.7.

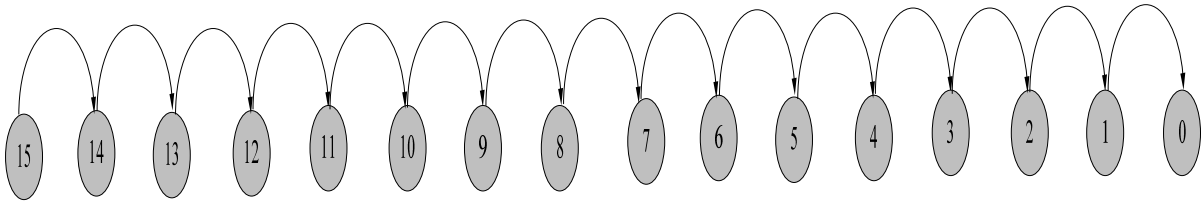


Fig. 3.7: Serial evaluation of the leading-one bit.

We can make a parallel/serial combination of leading-one finders to make a fast leading-one finder circuit as shown in Fig. 3.8. Here the 16-bit data is organized into four groups, each group having 4-bits. The 4-bits of a group are evaluated serially using a serial 4-bit LOF circuit [**108**], and all groups work concurrently. The 4-bit LOF circuit utilizes six 2-to-1 multiplexers and evaluates the inputs from MSB to LSB serially. As shown in Fig. 3.9, the 4-

bit output of the circuit provides the information about the leading-one bit and its corresponding position. To realize a 16-bit leading-one finder circuit (LOF16), the 4-bit LOF circuits are organized in to two stages so that concurrent evaluation of four LOF4 groups could take place. The circuit organization for the LOF16 is shown in Fig. 3.10.

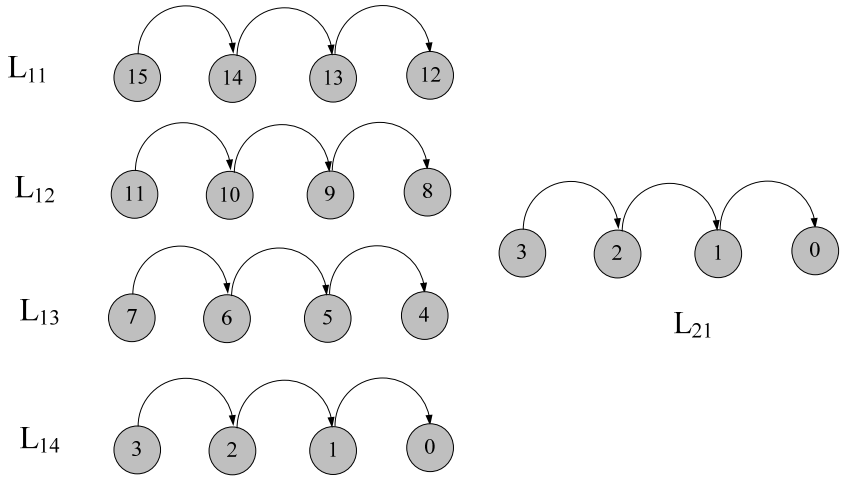


Fig. 3.8: Parallel/ serial evaluation of the leading-one bit.

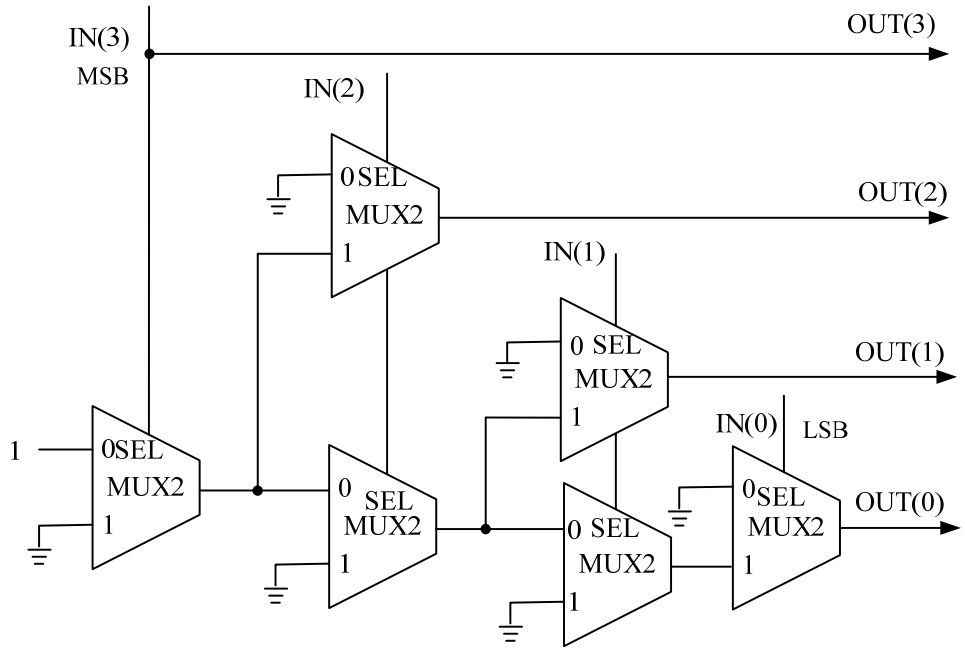


Fig. 3.9: 4-bit leading-one finder (LOF4).

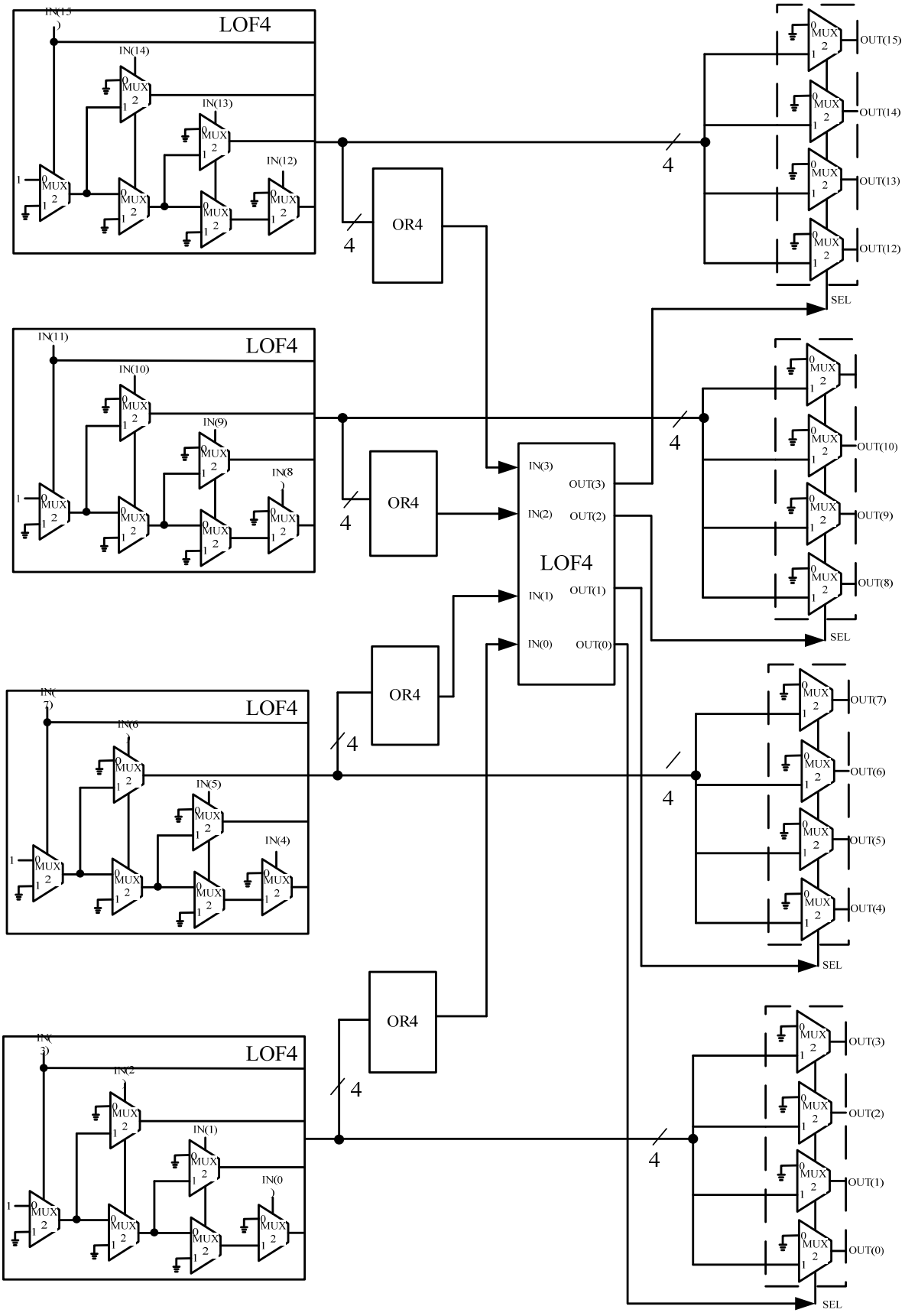


Fig. 3.10: Detailed circuit of a 16-bit leading-one finder (LOF16).

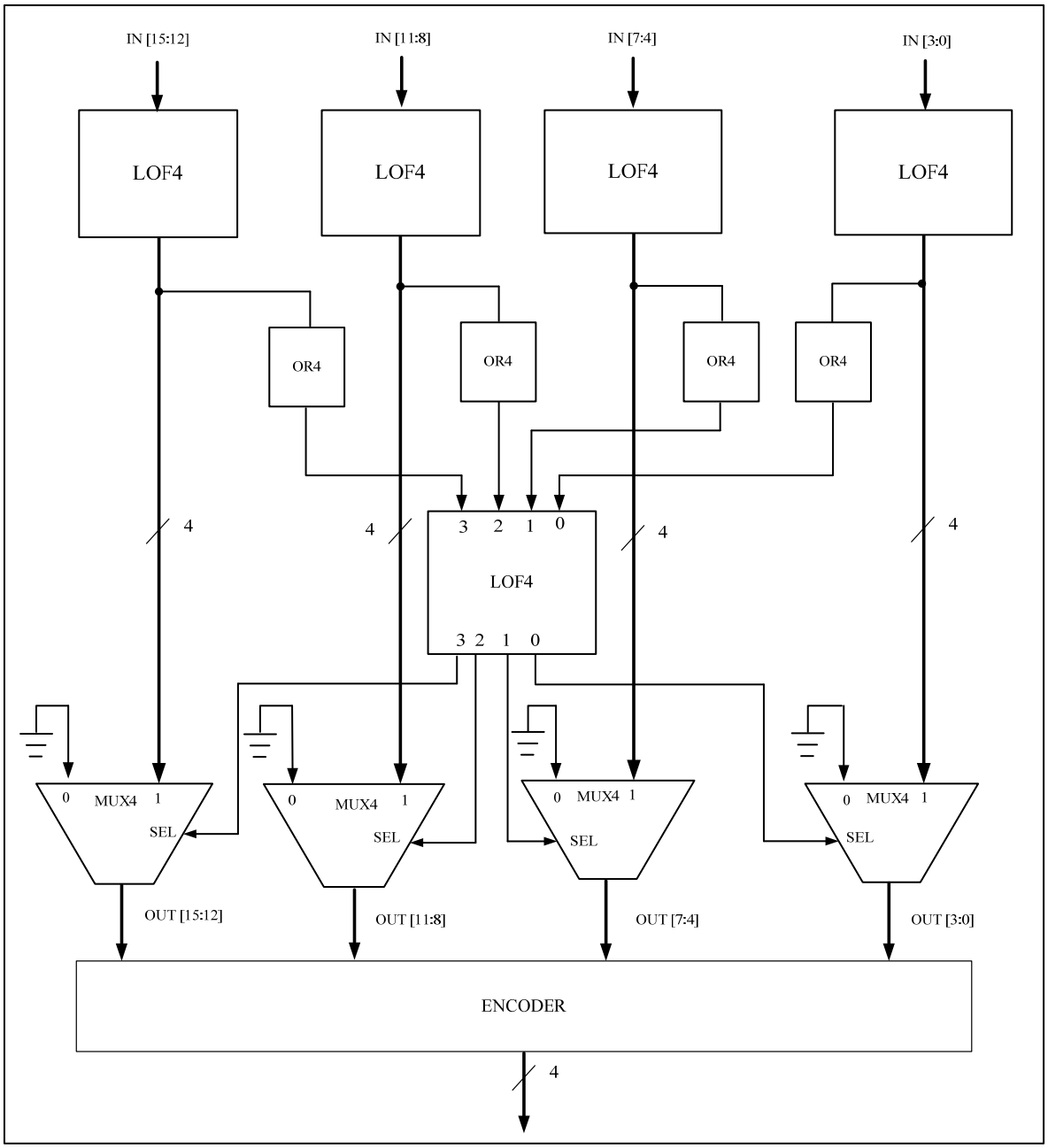


Fig. 3.11: Block diagram of the LOF16.

The simplified block diagram of the 16-bit LOF circuit (LOF16) is shown in the Fig. 3.11. In the (LOF16), four 4-bit LOF circuits are organized in the first stage, which receives 16-bit input and provides four 4-bit output groups. The output of each LOF4 circuit is provided to a 4-bit OR gate (OR4). The outputs of each OR4 gates are provided to the second stage of the LOF4 circuit. The second stage LOF4 circuit selects the first stage LOF4 circuit, which carries the leading-one. The four outputs of the second stage LOF circuit are fed to the select

lines of four 4-bit multiplexers. The first input of each multiplexer receives 4-bits from the outputs of the first stage LOF circuits and the second input is connected to logic '0'. The four 4-bit outputs of the multiplexers are fed to a binary encoder circuit which encodes these 16-bits into a 4-bit binary equivalent.

The 4-bits of the encoder output provide the position of the leading-one bit in the input. The truth table of the encoder is shown in Table 3.2. Here, 'X' can be logic '0' or logic '1'. The computed leading-one bit carries the information about the characteristic part of the binary logarithm. To compute the fractional value of the binary logarithmic, the bits following the leading-one bit, are passed on to a barrel shifter (BSHFT) circuit for further processing as discussed below.

Table 3.2: Leading-One Finder (LOF16) Encoder

Address	**Encoder Out (*S*)**
0000000000000000	0000
0000000000000001	0000
000000000000001X	0001
00000000000001XX	0010
0000000000001XXX	0011
000000000001XXXX	0100
00000000001XXXXX	0101
0000000001XXXXXX	0110
000000001XXXXXXX	0111
00000001XXXXXXXX	1000
0000001XXXXXXXXX	1001
000001XXXXXXXXXX	1010
00001XXXXXXXXXXX	1011
0001XXXXXXXXXXXX	1100
001XXXXXXXXXXXXX	1101
01XXXXXXXXXXXXXX	1110
1XXXXXXXXXXXXXXX	1111

3.4.3 The Barrel Shifter (BSHFT) Unit

After obtaining the leading-one bit, we evaluate the information about the characteristic part of the binary logarithm. In order to compute the fractional value of the binary logarithmic, the lower order bits following the leading-one bit are provided to a binary barrel shifter (BSHFT) circuit. The BSHFT circuit is composed of two 31-bit, 8-to-1 multiplexers and one 31-bit, 2-to-1 multiplexer as shown in Fig. 3.12.

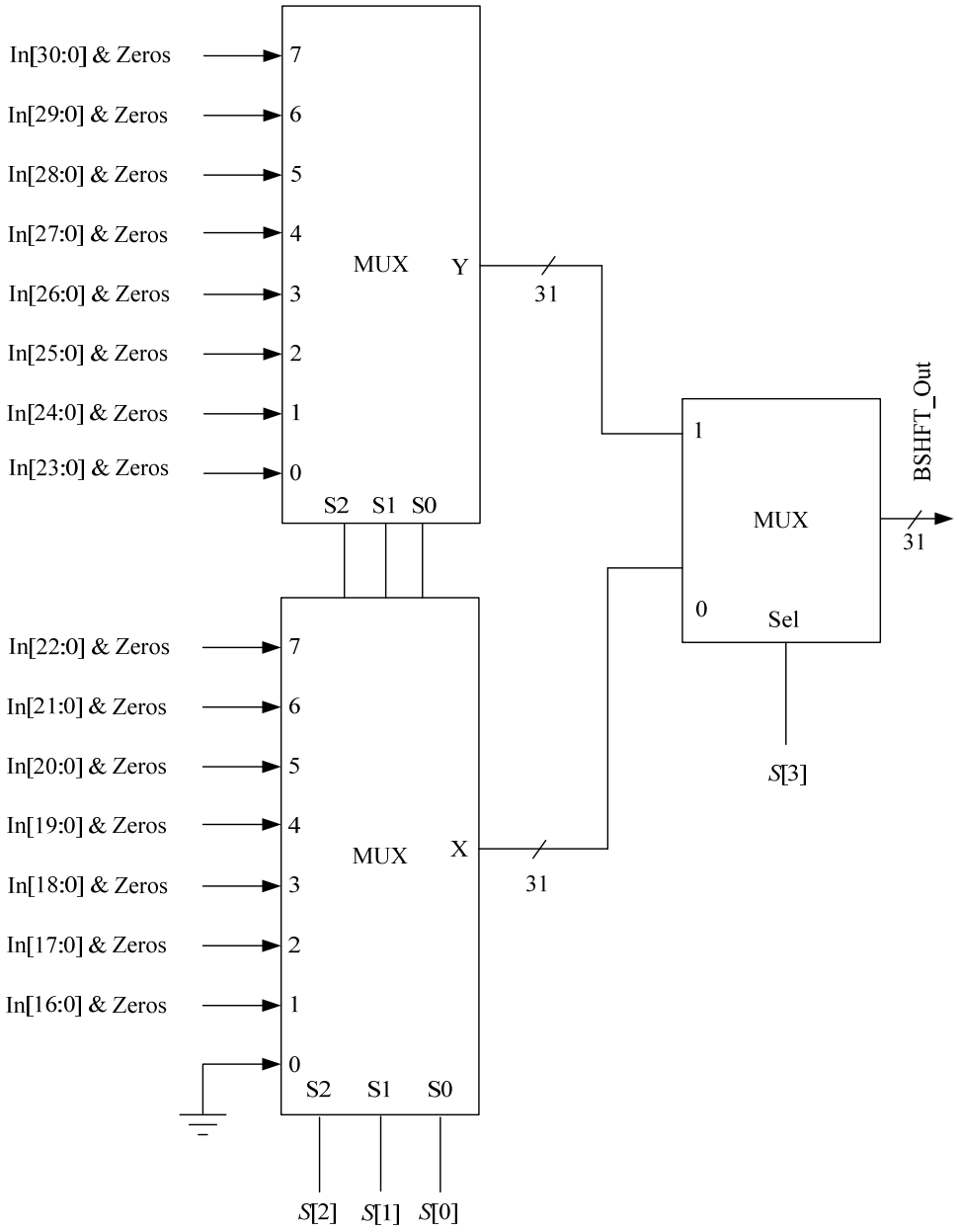


Fig. 3.12: Barrel shifter circuit (BSHFT) used in the binary logarithm computation unit.

It provides selection bits for the fractional-part approximation (FPA) circuit for computing fractional value of the binary logarithm. The selection process for required bit shifting in the BSHFT circuit is shown in Table 3.3. As explained in the Section 3.4.2 and shown in Fig. 3.11, the LOF16 circuit provides 4-bit output which is represented as *S*[3:0]. Bit *S*[3] is utilized to select the right most multiplexer of BSHFT circuit, which is shown in Fig. 3.12. In the circuit *S* [2:0] are used to select sixteen different input combinations through two 8-bits multiplexers. Here, bits *S* [2:0] are provided to the select lines of two multiplexers. Depending upon the bit value of *S* [3], any one of the multiplexer is selected. The selected multiplexer routes its input data to the output. The selection criteria is given in the Table 3.3. The output of BSHFT circuit is provided to a FPA unit, which is explained below.

Table 3.3: Truth Table for Realizing the Barrel Shifter

S	**Z**
0000	X “00000000”
0001	input(16 downto 0) & “00000000000000”
0010	input(17 downto 0) & “0000000000000”
0011	input(18 downto 0) & “000000000000”
0100	input(19 downto 0) & “00000000000”
0101	input(20 downto 0) & “0000000000”
0110	input(21 downto 0) & “000000000”
0111	input(22 downto 0) & “00000000”
1000	input(23 downto 0) & “0000000”
1001	input(24 downto 0) & “000000”
1010	input(25 downto 0) & “00000”
1011	input(26 downto 0) & “0000”
1100	input(27 downto 0) & “000”
1101	input(28 downto 0) & “00”
1110	input(29 downto 0) & ‘0’
1111	input(30 downto 0)

3.4.4 Fractional Part Approximation (FPA) Unit for Logarithm Computation

The architecture of a 31-bit fractional part approximation (FPA) unit is shown in Fig. 3.13. In the circuit shown in the Fig. 3.13, an 18×8 bit size ROM is used to store the approximated coefficients obtained from [**48**]. First 3-bits of the BSHFT output (f_l) are used to address the ROM. The contents of the ROM are shown in Table 3.4. The first eight-bits from the MSB side of the ROM content are multiplied with the output of the BSHFT unit (f_l). For this multiplication, an FPGA hard IP multiplier (DSP48E) is used.

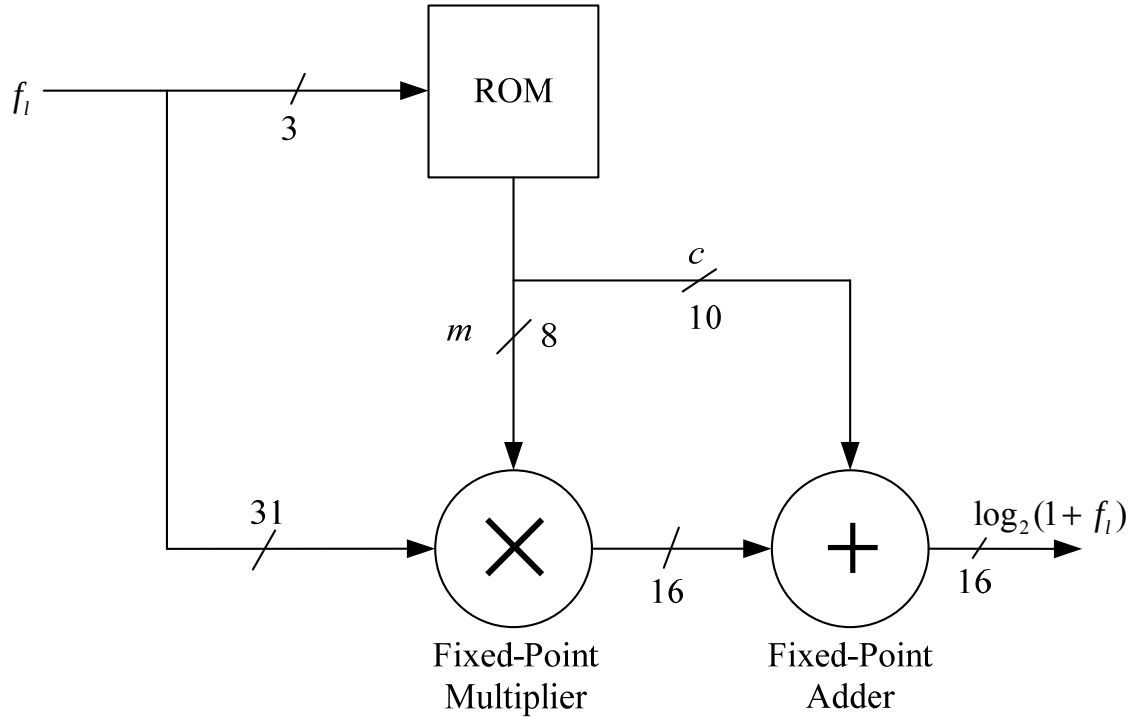


Fig. 3.13: Fractional part approximation (FPA) unit for the binary logarithm computation.

The output of the multiplier and the rest ten bits of the ROM are added by a fixed-point adder. The output of the adder provides the approximated mantissa (value of fractional part (f_l) of the binary logarithmic of the numbers).

Table 3.4: ROM Contents for the Binary Logarithm Computation

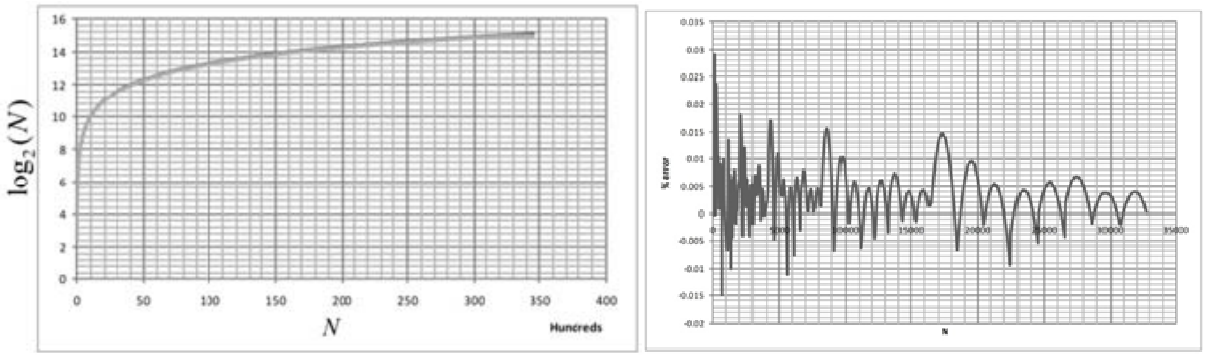
Address	Content
000	101011110000000000
001	100110110000010100
010	100011100000101110
011	100000010001010100
100	011101110001111011
101	011011100010100111
110	011001100011010111
111	010111110100001000

As discussed and shown in the Fig. 3.5, the generated characteristic and mantissa parts are combined which gives the binary logarithm of any 16.16 bit fixed-point binary number, in the 1.4.16 fixed-point format. Here, the first bit represents the sign of the output, the next four bits represent the characteristic part and the remaining 16-bits show the fractional value of the output. The functionality of the proposed architecture is validated by performing the required error analysis. The next section illustrates the details of error analysis performed using uniform random numbers.

3.4.5 Error Analysis of Logarithmic Approximation

To perform error analysis for the design, multiple sets of uniform random numbers (N) are generated. The range of N is $(0 \le N \le 2^n - 1)$. These random numbers are converted into a 32-bit (16.16) fixed-point data format. The converted random numbers are applied to the implemented design through a VHDL test-bench input file [**115**]. The output of the test-bench is converted into its corresponding real data type. The converted data, which consists of the computed binary logarithm result, is written into a binary file. The computed data are compared with the standard binary logarithm outputs up to five places of decimal digits. The graph of the computed logarithm is shown in Fig. 3.14(a). The percentage error between the

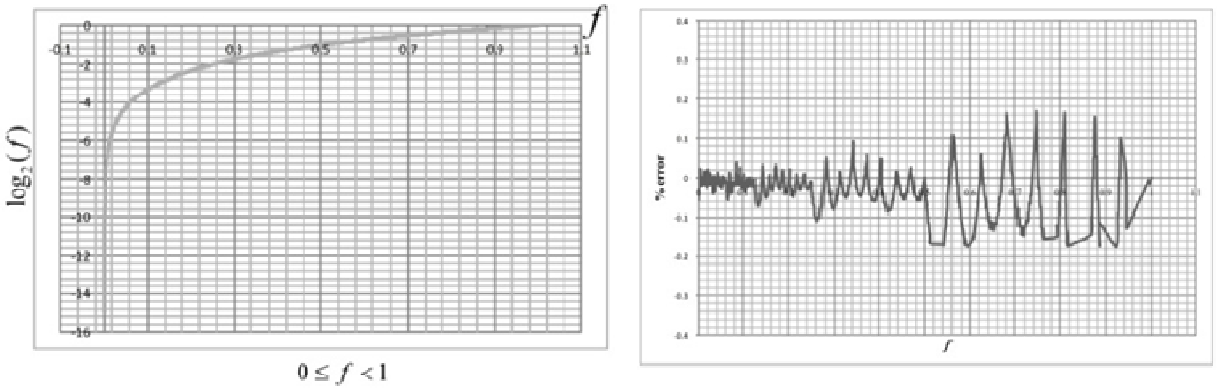
standard logarithmic output and the computed outputs are plotted in a graph, which is shown in Fig. 3.14(b).



(a) (b)

Fig. 3.14: (a) Computed logarithms for 16.16 fixed-point numbers (b) associated percentage error in computation.

The computational error of the implemented logarithmic computation circuit vis-à-vis the standard logarithmic values is less than 0.05 % over the entire range. Along similar lines, the computed outputs of the circuit for various random fractional values are plotted Fig. 3.15 (a). The fractional input numbers lie in the range $0 \le f_l < 1$. The percentage of computational error in the computed output is shown in Fig. 3.15 (b), maximum percentage error being 0.34 % .



(a) (b)

Fig. 3.15: Computed logarithms for the fractional numbers (b) associated percentage error in the computation.

The above error analysis shows that the proposed circuit has minimal errors in both the cases. The associated error is acceptably small for most of the practical image processing applications requiring embedded real-time solutions.

3.5 FPGA Implementation of Binary Logarithm Unit

The proposed architecture is implemented in Xilinx Virtex-5 xc5vfx70t FPGA device. The technology schematic of the implemented design as obtained from the Xilinx ISE tool is shown in Fig. 3.16.

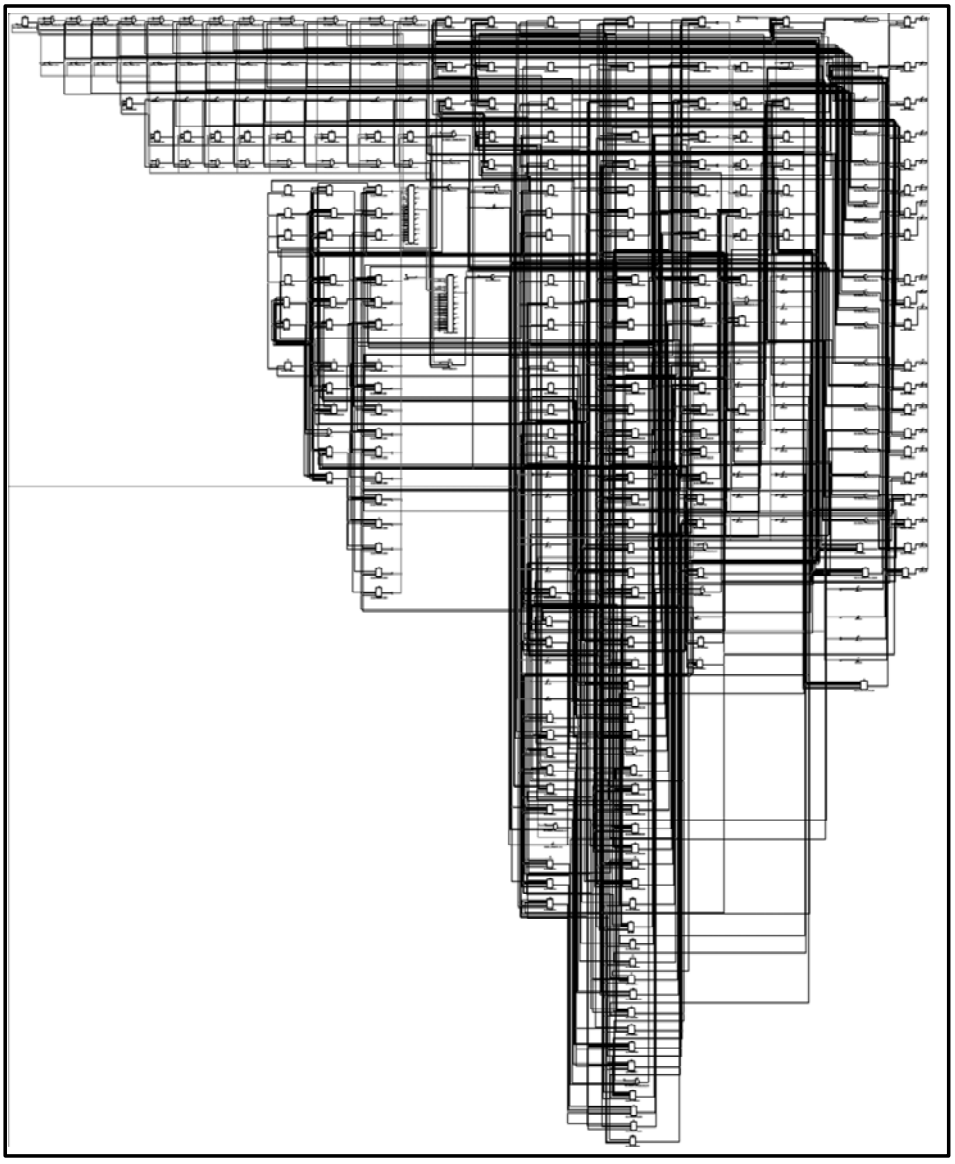


Fig. 3.16: FPGA-based technology schematic for the proposed architecture of the binary logarithm computation unit.

It is evident from the schematic, the implemented design consumes only a few FPGA slices and other logic resources. The FPGA device resource utilization summary for the design is given in Table 3.5. It is evident from the Table 3.5, that the proposed architecture utilizes only 209 LUTs out of available 44800 LUTs, which represent around 0.47 % utilization. The IOB utilization is 8.3 % (53 out of 640). Similarly, out of the 128 available DSP48E slices, the proposed architecture uses only 02 slices, which represents around 1.6 % utilization.

Table 3.5: FPGA Device Utilization for the Binary Logarithm Computation

Device Elements	**Utilization**
LUTs	209 /44800 (0.47 %)
External IOBs	53/640 (8.3 %)
DSP48Es	2/128 (1.6 %)

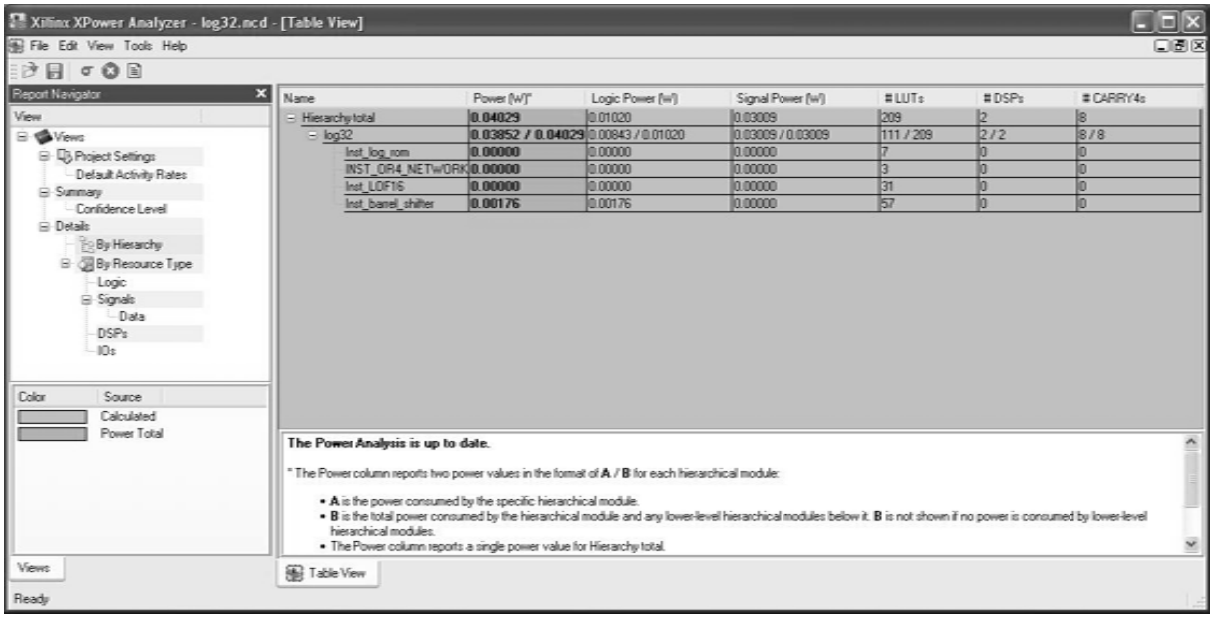


Fig. 3.17: Power analysis of logarithm computation architecture.

The architecture consumes 40.29 mW of total power, computed using Xilinx XPower analyzer [**52**], as shown in Fig. 3.17. Next section illustrates details of the binary antilogarithm approximation unit and its implementation in the FPGA device.

3.6 Binary Antilogarithm Approximation Unit and its Proposed Architecture

An antilogarithm approximation process without any hardware implementation is presented in [**111**,**116**]. As discussed in [**108**,**109**,**114**] the hardware implementation of antilogarithm converter is not very common in the literature. An antilogarithm converter architecture using CMOS is proposed in [**114**]. In this implementation, the circuit accepts 4-bit binary word to control a 16-bit logical shifter. The circuit is primarily designed for the positive binary numbers and the computational error analysis is not covered. The same concept is used to design a 32-bit antilogarithm converter [**108**,**109**]. Here, a 5-bit word of the characteristic part is used to control a 32-bit logarithmic shifter [**108**]. The upper 12-bits of the mantissa are provided to an arithmetic correcting circuit, which is based on 2, 6, and 7 region-correcting algorithms [**108**]. In another approach, a lookup table (LUT) and interpolation-based method is used to find the antilogarithm and has been implemented in the Xilinx xc2vp30 FPGA [**117**]. The implementation also focuses on the positive binary numbers. A piecewise linear approximation method for the positive and negative input numbers is discussed in [**109**]. In an implementation of the method, the integer and the fractional parts are computed separately which are then utilized by a barrel-shifter [**48**].

We propose a new architecture for the binary antilogarithm computation, which accepts both positive and negative input numbers. A curve-fitting method for the eight-regions of piecewise linear approximation of the fractional part is used to obtain the approximation coefficients. The computed approximation coefficients are stored in a small ROM, which are used by a fractional part approximation (FPA) unit. The integer part controls a unique barrel-shifter (BSHFT) which shifts the FPA output data. Depending upon the polarity of the input binary number the shifter shifts the input number to either left or right. The datapath of the

proposed architecture utilizes fixed-point arithmetic. The architecture is implemented in a Xilinx Virtex-5 xc5vfx70t FPGA device.

The implemented architecture utilizes available off-the-shelf FPGA components like multiplier and adder along with some of the FPGA slices. The device utilization shows that the proposed architecture utilizes minimal FPGA resources. The computational error analysis using thousands of uniform random numbers is performed and it is established that the proposed architecture provides antilog computation with acceptably small error values.

3.6.1 Architectural Building Blocks

As discussed in Section 3.2.2, the approximate antilogarithm value of an input binary number (X) is calculated by (3.4). The top-level block diagram of the proposed antilogarithm architecture is shown in Fig. 3.18. Here, a fractional part approximation (FPA) unit finds the value 2^{f_a} as required in (3.4). After finding the fractional part, a barrel-shifter (BSHFT) is used to left or right shift the computed FPA output value by the number of bits corresponding to the characteristic part as per (3.4).

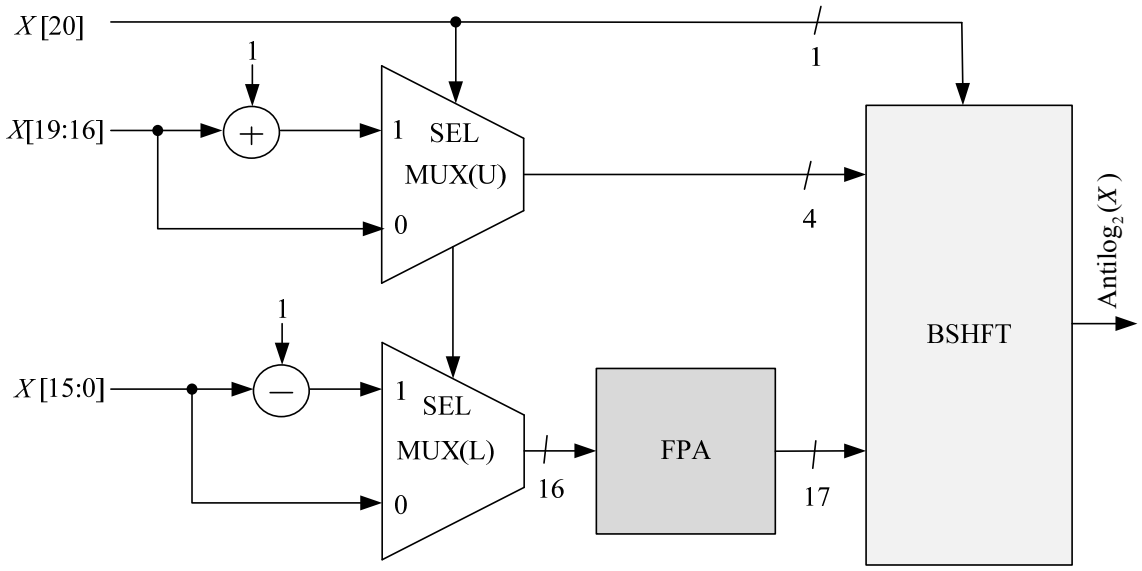


Fig. 3.18: Block diagram of the binary antilogarithm computational unit.

Depending upon the sign bit of the input number *X*[20], the input values for the FPA unit are selected by the two multiplexers (MUX (U) and MUX (L). The upper MUX (U) selects the values of k_a (4-bit) of the integer part of *X* [19:16], as per (3.5). The output of the MUX (U) is provided to the BSHFT unit which is used for the required shifting of the fractional part approximate value computed by the FPA unit. In a similar way, depending upon the sign bit, the lower MUX(L) selects the 16-bit fractional input number (5). The output of this multiplexer is also provided to the BSHFT unit. The details of the FPA and BSHFT units are given below:

3.6.1.1 The Fractional Part Approximation (FPA) Unit

In the proposed architecture the eight-region piecewise approximations is used to find the fractional part 2^{f_a} [**48**,**116**]. The fractional part can be approximately represented as:

$$2^{f_a} = m_i . f_a + c_i \tag{3.6}$$

where $0 \le i \le 7$ and it represents the eight piecewise linear regions. The calculated approximation coefficients (m_i and c_i) are stored in the eight locations of a 19-bit ROM, which is implemented in the FPGA fabric. The contents of the ROM are given in Table 3.6.

Table 3.6: ROM Contents for the Antilogarithmic Computation

ROM Address	Values
000	0101110010000000000
001	0110000001111111110
010	0110111101111100011
011	0111100101111000100
100	1000001001110100000
101	1000111101101100000
110	1001101101100011000
111	1010100101010110111

The first 8-bits from the MSB of the 19-bit ROM content are used to store the values of m_i and the rest 11-bits retain the values of c_i. To calculate the binary approximation of 2^{f_a} as per (3.6), the first three bits of the fractional binary number address the ROM. The ROM provides the above approximation coefficients. To compute (3.6), a fixed-point binary multiplier is used for the multiplication of m_i with the 16-bit fractional input number (f_a). The multiplier output is fed to a fixed-point adder, which adds c_i to it. The complete circuit arrangement of the FPA unit is shown in Fig. 3.19.

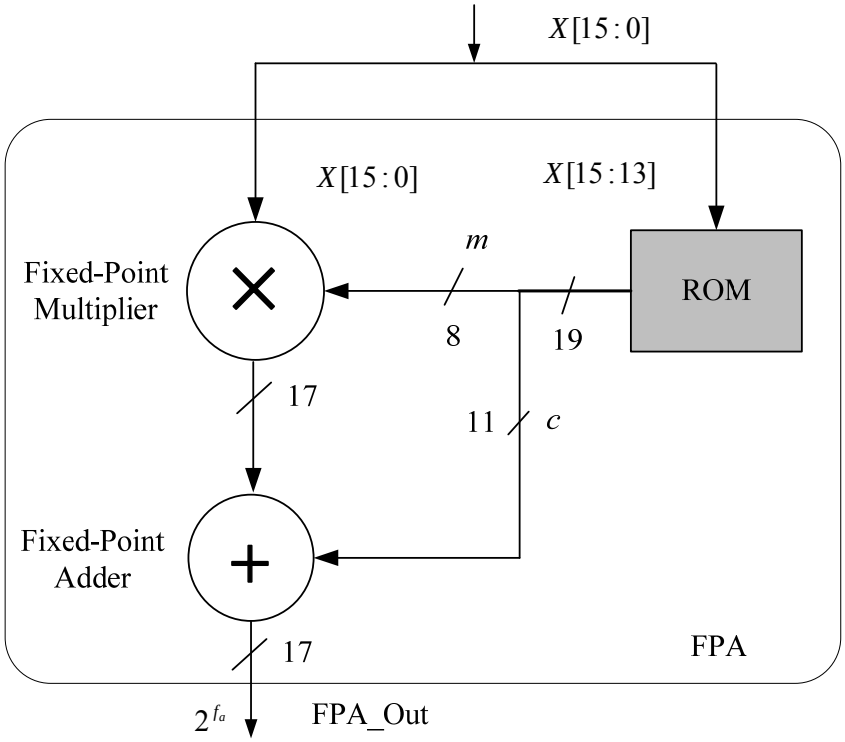


Fig. 3.19: Fractional part approximation (FPA) unit for binary antilogarithm computation.

3.6.1.2 Barrel Shifter (BSHFT) Unit for the Binary Antilogarithm Computation

A barrel shifter unit is used to shift the computed value of the 'FPA_Out'. The shifted data (BSHFT_32) is output of the fractional part approximation unit (3.4) shown in Fig. 3.20. As discussed in the Section 3.6.1, depending upon the sign bit of *X*, the shift can be to the right or left. When input number is positive, the FPA output value is left-shifted by k_a bits.

Whereas, when the number is negative, the FPA output value is right-shifted by the k_a bits. Depending upon the sign bit the appropriate input data (3.5) is selected which is shown in Fig. 3.18. The four bits of the integer part of the input number (X) controls the BSHFT data routing operation. The details of shift operation are given in Table 3.7.

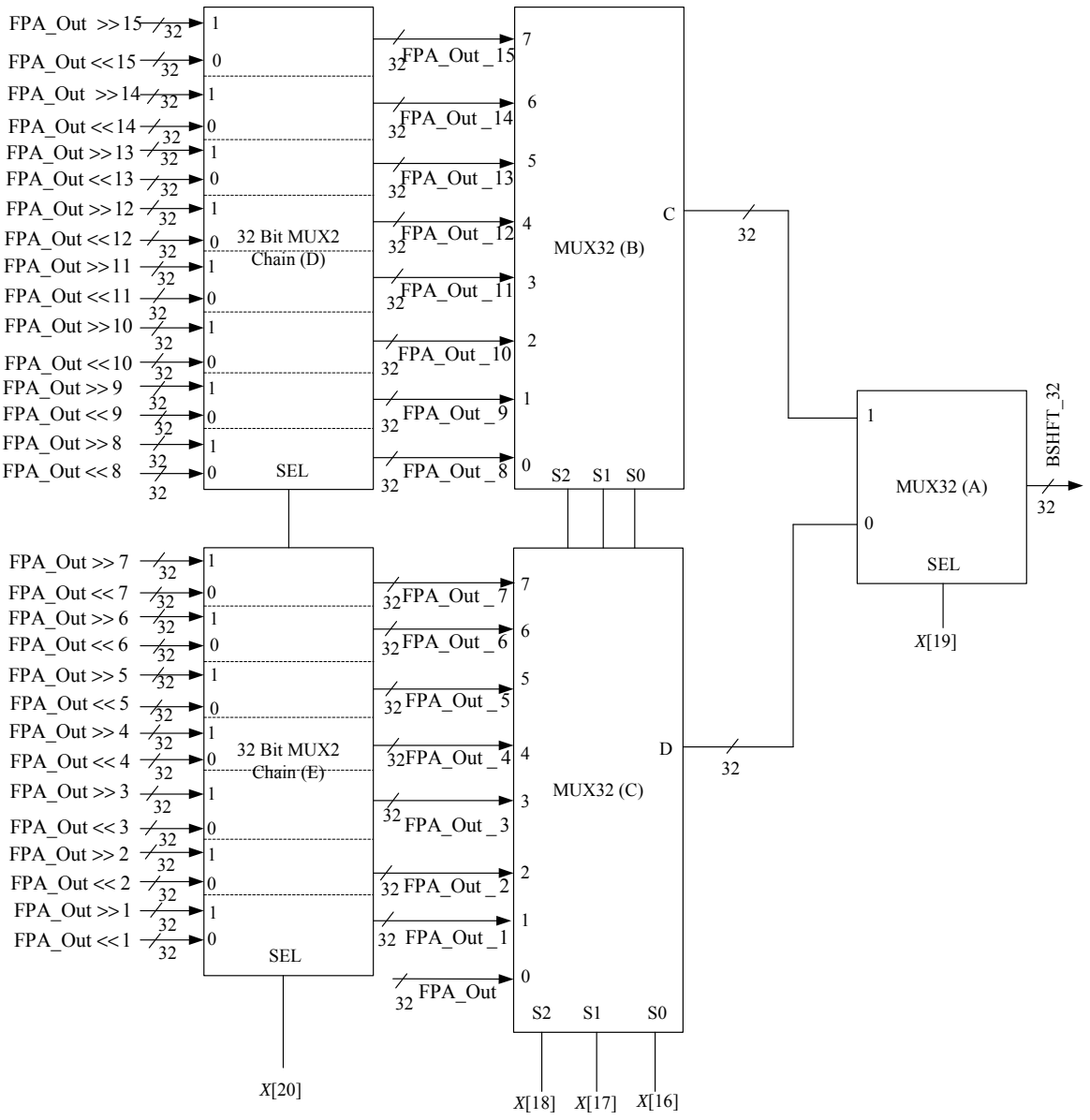


Fig. 3.20: Barrel shifter (BSHFT) unit for the binary antilogarithm computation.

The BHSFT unit is composed of five 32-bit multiplexers (MUX32) arranged as per the diagram shown in Fig. 3.20. As given in Table 3.7, the $X[19]$ bit is used to control a two-

channel 32-bit multiplexer (MUX32A). The MUX32A takes the two groups of 32-bit data, which comes from two multiplexer (MUX32B and MUX32C). These two multiplexers are eight-channel 32-bit multiplexers that are controlled by bit $X[18:16]$. When bit $X[19]='0'$, the MUX32 (A) routes the data of the MUX32(C) to the output, and when $X[19]='1'$ it passes the MUX32 (B) data to the output.

Table 3.7: BSHFT Data Routing Operation for the Binary Antilogarithm Computation

MUX Select Lines		**32-bit MUX2 (A) Input Values**	
$X[19]$	$X[18:16]$	$X[20]=1$	$X[20]=0$
0	000	FPA_Out	FPA_Out
0	001	FPA_Out $\gg 1$	FPA_Out $<< 1$
0	010	FPA_Out $\gg 2$	FPA_Out $<< 2$
0	011	FPA_Out $\gg 3$	FPA_Out $<< 3$
0	100	FPA_Out $\gg 4$	FPA_Out $<< 4$
0	101	FPA_Out $\gg 5$	FPA_Out $<< 5$
0	110	FPA_Out $\gg 6$	FPA_Out $<< 6$
0	111	FPA_Out $\gg 7$	FPA_Out $<< 7$
1	000	FPA_Out $\gg 8$	FPA_Out $<< 8$
1	001	FPA_Out $\gg 9$	FPA_Out $<< 9$
1	010	FPA_Out $\gg 10$	FPA_Out $<< 10$
1	011	FPA_Out $\gg 11$	FPA_Out $<< 11$
1	100	FPA_Out $\gg 12$	FPA_Out $<< 12$
1	101	FPA_Out $\gg 13$	FPA_Out $<< 13$
1	110	FPA_Out $\gg 14$	FPA_Out $<< 14$
1	111	FPA_Out $\gg 15$	FPA_Out $<< 15$

MUX32 (B) and MUX32(C) take their input data from a chain of two-channel 32-bit MUXs (D, E). These multiplexers receive the seventeen-bit data from 'FPA_Out'. While shifting, the required amount of zeros is appended to the left or right of the input data (FPA_Out) to make a 32-bit data width for all the multiplexers. Here, depending upon the

sign-bit polarity, the two-channels of the MUX-chain operate. The multiplexer chain routes the left shifted data for positive input data and the right shifted input data for the case of negative input data.

3.6.2 Error Analysis of the Binary Antilogarithm Approximation

To perform the error analysis multiple sets of uniformly distributed random numbers (N) in the range of $2^{-N} \leq N \leq 2^{N} - 1$ are generated. These random numbers are changed into a 21-bit (1.4.16) fixed-point data format. These inputs are applied to the implemented design through a VHDL test-bench input file. The output of the test-bench is converted into the real data type and written into a binary file. The converted data are compared with the standard binary antilogarithm outputs up to five places of decimal digits. The percentage error between the standard antilogarithm output and the output obtained from the proposed architecture are plotted in a graph, which is shown in Fig. 3.21. The maximum percentage of computational error is 0.16 %, which is acceptable for most image processing applications.

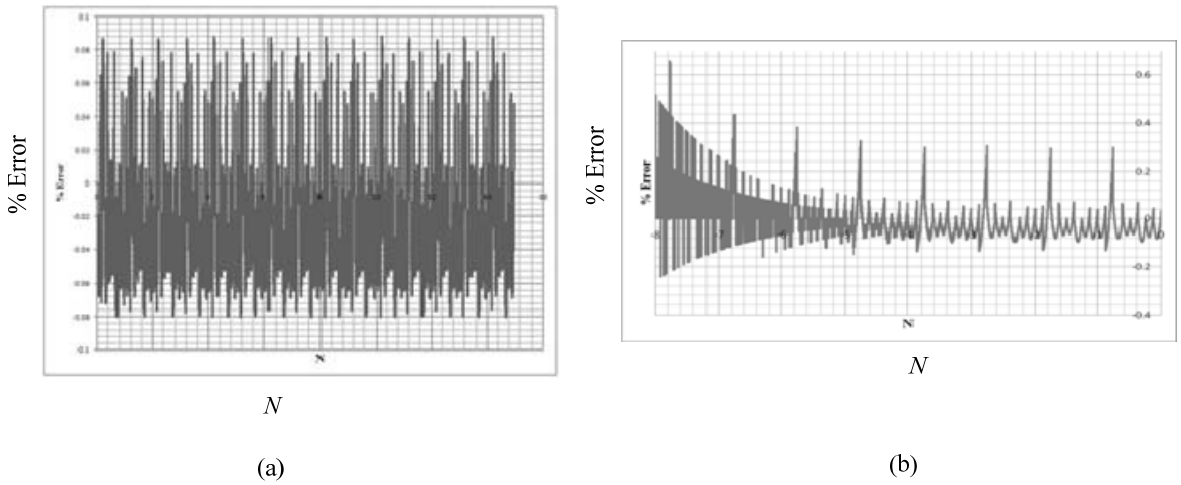


Fig. 3.21: Percentage computational error (a) for positive input binary numbers (b) for the negative input binary numbers.

3.7 FPGA Implementation Results of the Binary Antilogarithm Unit

The proposed architecture for the antilogarithm computation is implemented in the Xilinx Virtex-5 xc5vfx70t FPGA device. The FPGA technology schematic for the implemented design is shown in Fig. 3.22. As it is evident from the technology schematic, the implemented design utilizes a few FPGA resources.

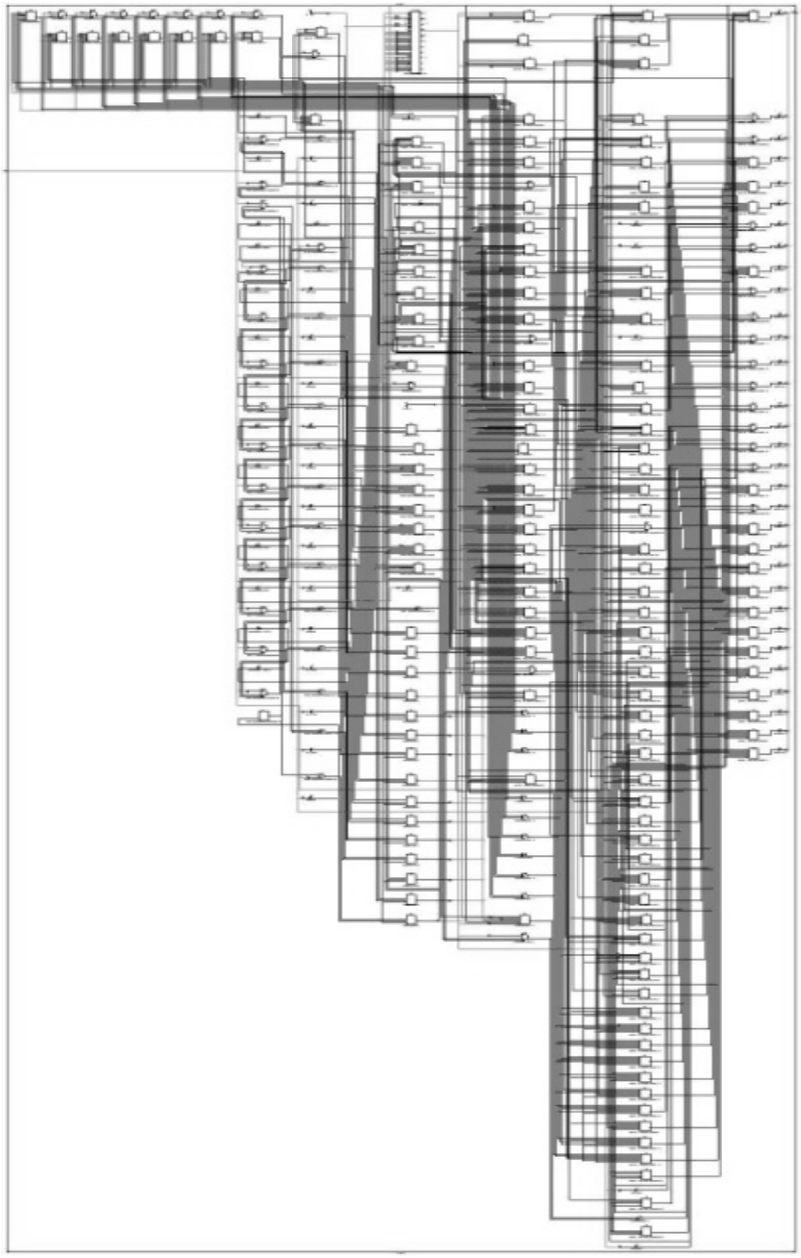


Fig. 3.22: FPGA-based technology schematic for the implemented binary antilogarithm computational unit.

The FPGA device resource utilization summary of the design is given in Table 3.8. It can be observed from Table 3.8, that the proposed architecture requires only 0.37 % of the FPGA LUTs. Along with this, the antilogarithm computation block requires 0.78% (1 out of 148) of the DSP48E slice available with the Virtex-5 FPGA device. The proposed architecture, utilizes simple arithmetic circuits that use a fixed-point datapath, which leads to reduction of the number of input/output blocks (IOBs). The implemented architecture utilizes only 8.28 % IOBs. The total power consumption of the proposed architecture is found to be 21 mW.

Table 3.8: FPGA Device Utilization for the Binary Antilogarithmic Computation

Elements	**Proposed Architecture**
Slice LUTs	163 /44800 (0.37 %)
External IOBs	53/640 (8.28 %)
DSP48Es	1/128 (0.78 %)

3.8 Conclusion

Hardware architectures for binary logarithm and antilogarithm approximation circuits are proposed in this chapter. The proposed architectures are suitable for embedded image and video processing applications. The proposed architectures are based on fixed-point data type and are implemented in Xilinx Virtex-5 xc5vfx70t FPGA device.

The hard macro cores like the adder and the multiplier available in FPGA device are utilized for the computation of the mantissa part of the binary logarithm. A leading-one finder circuit obtains the characteristic portion of the binary logarithm. The FPGA device utilization shows that the proposed architecture utilizes minimal FPGA resources. The power consumption of the proposed architecture for logarithm computation as computed using XPower analyzer is 40.29 mW. The error analysis of the implemented architecture is performed with thousands of uniform random numbers. The error analysis shows that the proposed architecture provides adequate levels of accuracy. Maximum error is percentage of

0.05 % with 16.16 fixed-point numbers and 0.34 % with fractional numbers in the range $0 \leq f_l < 1$.

For the design of the antilogarithm unit, the characteristic portion of the binary number is used to shift the computed mantissa part with the help of a barrel-shifter. The barrel-shifter uses a few multiplexers to route the logical shifted value of the mantissa part. The output of the barrel-shifter is the approximate value of the binary antilogarithm. The FPGA device utilization data shows that the proposed architecture uses minimal FPGA resources and it consumes 21 mW power. The error analysis of the implemented architecture is performed with thousands of uniformly distributed random numbers. The error analysis shows that the proposed architecture provides adequate level of accuracy. The percentages of computational errors are found to lie in the range of $\pm 0.08\%$ for positive binary numbers and -0.2 % to $+0.6$ % for negative binary numbers.

The real-time realization of complex arithmetic functions such as square root function, the raised to the power function, and the division function on fixed-point numbers required in Chapter 4 and 6, have been made possible through the transformation and realization of the computations in the logarithmic domain and then back into the fixed-point number system using the logarithm approximation and antilogarithm approximation unit described in this chapter.

CHAPTER 4

ARCHITECTURE AND HARDWARE REALIZATION OF AN IMAGE THRESHOLDING ALGORITHM

4.1 Introduction

In various image and video processing applications, it is necessary to extract the gray levels of object pixels, which are significantly different from the object's background [**3**,**8**,**118**]. The image thresholding is defined as an operation, by which a gray-level image is converted into its corresponding binary image. The thresholding operation is used to extract an object from its background such that each pixel is either classified as an object pixel (white) or a background pixel (black) [**119**]. The image/video acquisition module developed in Chapter 2 provides 640×480 pixel RGB image. A RGB-to-gray conversion module converts the RGB image into its corresponding gray-scale form. The thresholding unit provides an optimum threshold value by which the gray scale image is converted into a binary image. The obtained binary image is used by the connected component labeling algorithm, which is described in Chapter 5.

In an image, the thresholding operation can be performed globally or locally. In the global or fixed thresholding process, the threshold value is constant throughout the image, whereas, in the local or variable thresholding, multiple threshold values of the same image can exist. Many image and video processing applications need image thresholding unit [**119**], which include, text detection in natural images [**61**], adaptive progressive thresholding [**6**], noise reduction for human action recognition [**57**], real-time segmentation of images with complex backgrounds [**60**], personal verification [**4**], optical character recognition and image extraction [**62**,**63**], automatic target recognition [**120**].

In [**119**], the thresholding methods are categorized into six broad groups namely, histogram shape-based methods, clustering-based methods, entropy-based methods, object attribute-based methods, spatial methods and local methods.

A clustering-based nonparametric, unsupervised method of automatic threshold selection for image segmentation in gray-level images was presented by Otsu [**49**]. Otsu's method is a very popular thresholding technique, which is applied to a wide variety of applications such as: text detection in natural images [**61**], adaptive progressive thresholding [**6**], noise reduction for human action recognition [**57**], real-time segmentation of images with complex background [**60**], personal verification [**4**], optical character recognition and image extraction [**62**,**63**].

These applications require real-time computational efficiency of the image thresholding process. To achieve this, hardware implementation of the thresholding algorithm is necessary [**50**,**51**,**64**]. A direct implementation of Otsu's algorithm in hardware requires many computation intensive resources such as iterative squaring, complex multipliers, and dividers with fractional value accuracy [**50**,**51**]. A VLSI architecture for the segmentation of endoscopic images using Otsu's approach has been proposed in [**50**]. A field-programmable gate array (FPGA) based architecture for the between-class variance (BCV) computation of Otsu's algorithm has been presented in [**51**] for Xilinx Virtex xcv800 hq240-4 FPGA device where a 256×256 image data is stored in four 16 K RAM chips. Along similar lines, an architecture for the BCV, which employs Altera's divider and multiplier megacores, is presented in [**64**].

This chapter presents a resource-efficient architecture for the design of Otsu's thresholding algorithm and its implementation in the FPGA device. The proposed architecture is implemented for a 640×480 size of input image that is captured by a real-time high-

resolution analog camera and buffered in a DDR2 SDRAM memory. The computation of between-class variance in Otsu's algorithm requires the evaluation of a normalized cumulative histogram, mean and cumulative moments, which need single-cycle read-modify-write operations. These operations are achieved by incorporating the FPGA slices, dual-port Block RAM memories and DSP slices with DDR2 SDRAM as a frame buffer. The datapath of the architecture is fixed-point arithmetic based and it does not require any divider. The proposed design is implemented in the Xilinx Virtex-5 xc5vfx70tffg1136-1 FPGA device, available on the Xilinx ML-507 platform [**33**]. In order to develop the required hardware and software in an integrated manner, the Xilinx Embedded Development Kit (EDK) design tool is used [**46**]. The proposed architecture is utilized for the connected component analysis algorithm, which is covered in Chapter 5.

The rest of the chapter is organized as follows. In Section 4.2, the RGB-to-gray conversion process is described. Section 4.3 discusses the Otsu's automatic threshold selection method. In Section 4.4 the hardware implementation issues of Otsu's algorithm are covered. Section 4.5 is used to describe the proposed architecture for FPGA implementation of Otsu's global automatic image thresholding algorithm. This section also covers the details of each building blocks of the proposed architecture. Section 4.6 shows the implementation results. The proposed architecture can also be utilized as a core, Section 4.7 covers the details of system arrangement with thresholding unit used as a core. Finally, Section 4.8 concludes the chapter.

4.2 RGB to Gray Conversion

Image thresholding algorithm works on the gray scale pixels. The gray pixels are obtained from the RGB color pixels. The captured RGB pixels (each 8-bit) can be converted into the 8-bit gray level format by the following expression [**121**,**122**].

$$\text{Grayscale} = R \times 0.2989 + G \times 0.5870 + B \times 0.1140 \quad (4.1)$$

The above expression uses the weighted sum of R, G and B. Since division by powers of two (through shifting) is hardware friendly, we have used division friendly approximations for coefficient values of (4.1). The following equation is used to convert the RGB image, into reasonably acceptable results in the form of the gray level image.

$$\text{Grayscale} = R \times 0.25 + G \times 0.5 + B \times 0.125 \tag{4.2}$$

To obtain a gray-level image from RGB data, the above expression (4.2) uses only shifting and addition operations. The components in expression (4.2) consist of 2-bit right shifted Red (R) pixel, 1-bit right-shifted Green (G) pixel and 3-bit right-shifted Blue (B) pixel. The shifted R, G and B pixels are accumulated, which provides the corresponding gray-scale image. The converted gray-level image with 8-bit gray values $(0 \ldots 255)$ are buffered in the DDR2 SDRAM memory. The RGB2Gray unit uses embedded PowerPC 440 processor and the Xilinx video frame buffer controller (VFBC) available with its multi-port memory controller (MPMC) IP [**105**]. The read-write process uses a 32-bit native port interface (NPI) protocol, which is synchronous with the MPMC controller. The details of the NPI protocol are explained in Section 4.7. The converted gray-level image is used in the automatic thresholding unit. In the next section, the details of automatic threshold selection method given by Otsu are explained.

4.3 Otsu's Automatic Threshold Selection Method

Otsu presented a clustering-based global thresholding method, which is based on the shape properties of the gray-level histogram [**49**]. The algorithm is summarized in the following. Let n_i represents the number of pixels with gray level i, L be the number of gray levels $[1, 2 \ldots L]$ in the image and N be the total number of pixels in the image i.e. $N = n_0 + n_1 + \ldots n_L$. The probability distribution or the normalized histogram of the gray level image is defined as,

$$p_i = \frac{n_i}{N},\ p_i \geq 0,\quad \sum_{i=0}^{L} p_i = 1 \tag{4.3}$$

If we divide the pixels into two classes C_0 and C_1 corresponding to background (0) and foreground (1) pixels by threshold at level *k*, then the probabilities of class occurrence are:

$$\omega_0(k) = Pr(C_0) = \sum_{i=0}^{k} p_i \tag{4.4a}$$

$$\omega_1(k) = \Pr(C_1) = \sum_{i=k+1}^{L} p_i \tag{4.4b}$$

The class means are given by,

$$\mu_0(k) = \sum_{i=0}^{k} \frac{i.p_i}{\omega_0(k)} \tag{4.5a}$$

$$\mu_1(k) = \sum_{i=k+1}^{L} \frac{i.p_i}{\omega_1(k)} \tag{4.5b}$$

The total mean-level of the original image is,

$$\mu_T = \sum_{i=0}^{L} i.p_i \tag{4.6}$$

For any value of *k*

$$\omega_0\mu_0 + \omega_1\mu_1 = \mu_T, \tag{4.7}$$

where, $\omega_0 + \omega_1 = 1$.

The individual class variances corresponding to the background and foreground are,

$$\sigma_0^2(k) = \sum_{i=0}^{k} [i - \mu_0(k)]^2 \frac{p_i}{\omega_0(k)} \tag{4.8a}$$

$$\sigma_1^2(k) = \sum_{i=k+1}^{L} [i - \mu_1(k)]^2 \frac{p_i}{\omega_1(k)} \tag{4.8b}$$

Now, the within-class variance (WCV) is defined as,

$$\sigma_w^2(k) = \omega_0(k)\sigma_0^2(k) + \omega_1(k)\sigma_1^2(k) \tag{4.9}$$

and the between-class variance (BCV) is given as,

$$\sigma_B^2(k) = \omega_0(k)\omega_1(k)(\mu_1 - \mu_0)^2 \tag{4.10}$$

We can express the total variance as,

$$\sigma_T{}^2 = \sigma_w^2(k) + \omega_0(k)[1 - \omega_0(k)][\mu_0(k) - \mu_1(k)]^2 \tag{4.11}$$

In (4.11) the first term is WCV $(\sigma_w^2(k))$ and the second term is BCV $(\sigma_B^2(k))$. It is noted that within-class variance is based on the second-order statistics (class variance), while the between-class variance is based on the first-order statistics (class mean). The total variance is constant and independent of *k*. So, minimizing the within-class variance is the same as maximizing the BCV $(\sigma_B^2(k))$.Thus, the gray level for which the BCV is maximum is chosen as the most suitable threshold value (k^*), which can be expressed as,

$$k^* = \arg \max_{0 \le k \le L} \sigma_B^2(k) \tag{4.12}$$

The conceptual diagram for the computation of optimum threshold value using Otsu's algorithm is shown in Fig. 4.1.

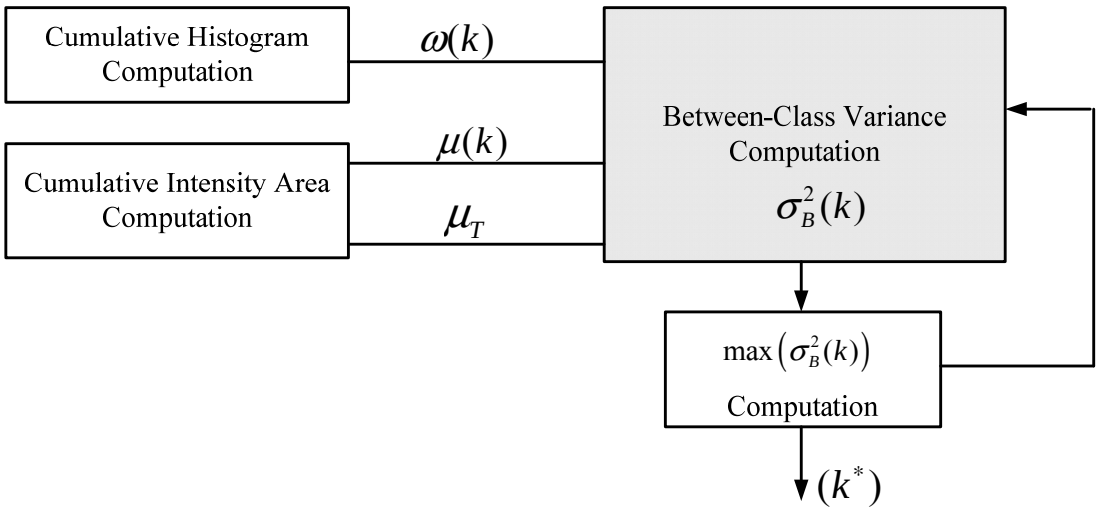


Fig. 4.1: Block diagram for computing optimum threshold value using Otsu's algorithm.

As apparent to compute the BCV, there is requirement of the cumulative histogram $\omega(k)$, and cumulative area $\mu(k)$ computation. The total mean-level of the image can be computed through $\mu(k)$ computing block. The optimal threshold (k^*) is obtained through a sequential search for the maximum of $\sigma_B^2(k)$ for $0 \le k \le L$, shown in the Fig. 4.1.

4.4 Hardware Implementation Issues Related to Otsu's Algorithm

As we know that the optimal threshold (k^*) is obtained through a sequential evaluation for the maximum of $\sigma_B^2(k)$ for $0 \le k \le L$. Now, by using (4.4) and (4.5), we can write (4.10) as,

$$\sigma_B{}^2 = \omega_0(k)\left[1-\omega_0(k)\right]\left[\frac{\mu_T-\mu_k}{1-\omega_0(k)}-\frac{\mu_k}{\omega_0(k)}\right]^2 \tag{4.13}$$

The direct hardware implementation of the BCV computation (4.13) is shown in Fig. 4.2. After computing the cumulative histogram (CH) and the cumulative intensity area (CIA), the computed values are stored in to two RAMs. We observe that a direct implementation of (4.13) requires a large number of compute-intensive complex operations such as, two divisions, one squaring and three multiplications.

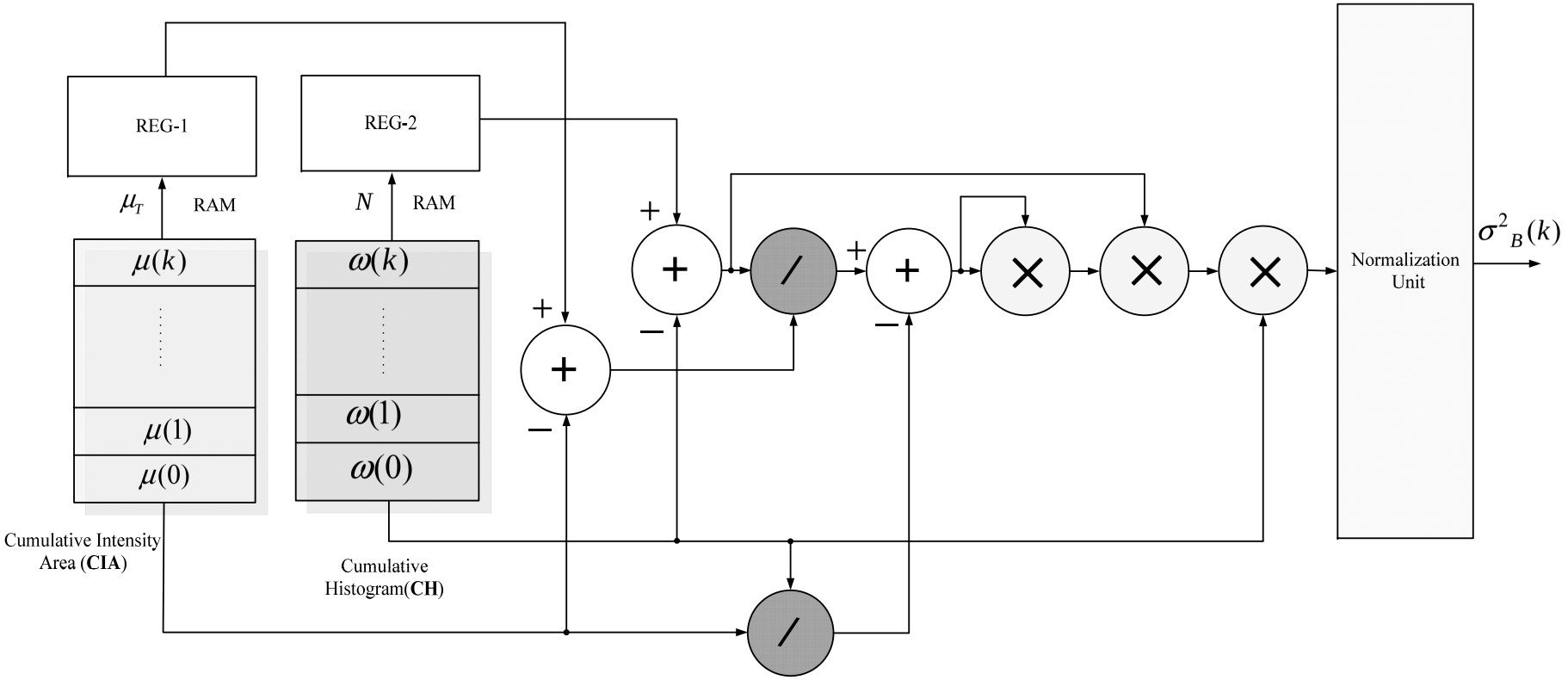


Fig. 4.2: Direct implementation of Otsu's algorithm in hardware.

Apart from this, to normalize the computed data (4.3) it also requires a separate normalization unit, which ultimately turns into a division process [**51**]. Since Otsu's method needs maximization of BCV of the foreground and background pixels of the image so that an optimum threshold (k^*) can be established. The between-class variance (4.12) can also be further written as given as:

$$\sigma_B^2(k) = \frac{[\mu_T.\omega(k) - \mu(k)]^2}{\omega(k).[1-\omega(k)]} \tag{4.14}$$

where, the zeroth-order cumulative moment is,

$$\omega(k) = \sum_{i=0}^{k} p_i \tag{4.15}$$

and the first-order cumulative moment is given by,

$$\mu(k) = \sum_{i=0}^{k} i.p_i \tag{4.16}$$

and the total mean value can be derived as,

$$\mu_T = \sum_{i=0}^{L} i.p_i = \mu(L) \tag{4.17}$$

As evident from the expression of between-class variance (4.14), the computation of $\sigma_B^2(k^*)$ requires the computation of terms $\omega(k)$ in (4.3) and (4.15), $\mu(k)$ in (4.3) and (4.16). The BCV equation (4.14) can be converted into simple addition and subtraction operations by taking the logarithm of both sides of (4.14) as,

$$\log_2\sigma_B^2(k) = 2\log_2[\mu_T.\omega(k) - \mu(k)] - \log_2\omega(k) - \log_2[1-\omega(k)] \tag{4.18}$$

and the optimum threshold can be obtained as,

$$k^* = \arg\max_{0 \le k \le L} \log_2 \sigma_B^2(k) \tag{4.19}$$

Thus, we can get the optimum value of threshold (k^*) by a sequential search for the maximum of $\log_2 \sigma_B^2(k)$ in the range of k.

4.5 The Proposed Architecture for Otsu's Algorithm

The details of the proposed architecture for computing Otsu's algorithm are shown in Fig. 4.3.

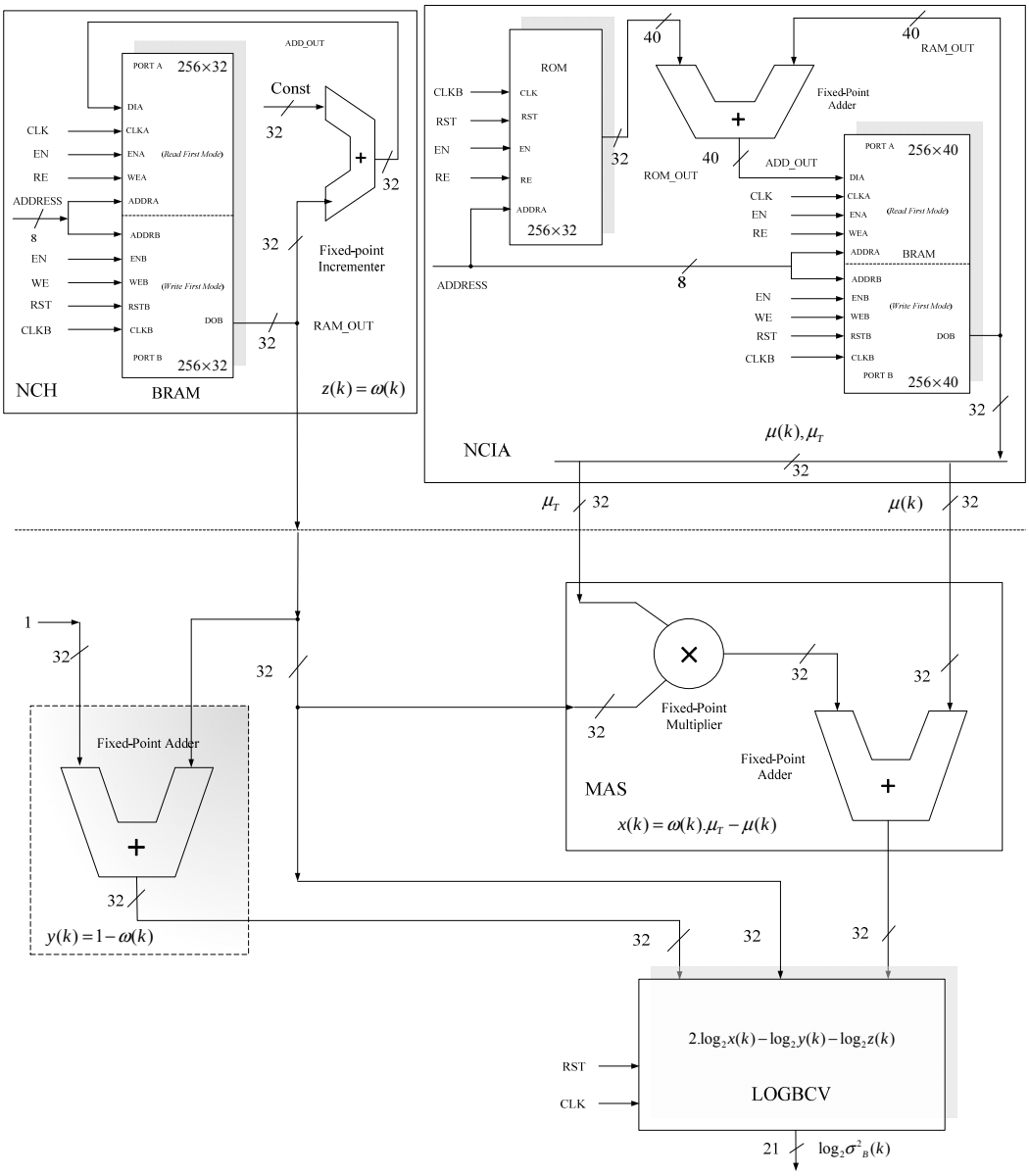


Fig. 4.3: Detailed structure of the proposed architecture for computing Otsu's algorithm.

This architecture for Otsu's thresholding algorithm is based on the concept of logarithmic number system (LNS) as explained in the previous section. It utilizes the realization of logarithmic function as presented in Chapter 3. The proposed architecture is realized for the 640×480 pixel input image that is captured by a real-time high-resolution analog camera and buffered in a DDR2 SDRAM memory.

The computation of between-class variance in Otsu's algorithm requires the evaluation of a normalized cumulative histogram and the mean and cumulative moments, which is achieved through single-cycle read-modify-write operations. These operations are achieved by incorporating in the datapath, the FPGA slices, dual-port Block RAM memories and the DSP slices along with the DDR2 SDRAM as a frame buffer. The datapath architecture is fixed-point arithmetic based and it does not require any divider or normalization unit.

The required normalization in normalized cumulative histogram (NCH) computation is obtained through adding the normalization constant (which is the reciprocal weight of the total number of pixels) with the computed cumulative histogram. In a similar fashion, the need of normalization is also taken care of in the computation of normalized cumulative intensity area (NCIA). This following subsection presents the various architectural building blocks for the implementation of Otsu's algorithmic the Virtex-5 FPGA device.

The simplified block diagram of the proposed architecture is shown in Fig. 4.4. Here, NCH and NCIA blocks hold the computed zeroth-order cumulative moment (4.15) and the first-order cumulative moment (4.16). Both NCH and NCIA blocks are realized using FPGA BRAMS. We obtained the result of computations of (4.15) and (4.16) for each intensity level, *k*, using single-cycle read-modify-write operation without the need of any normalizing divider unit of equation (4.3).

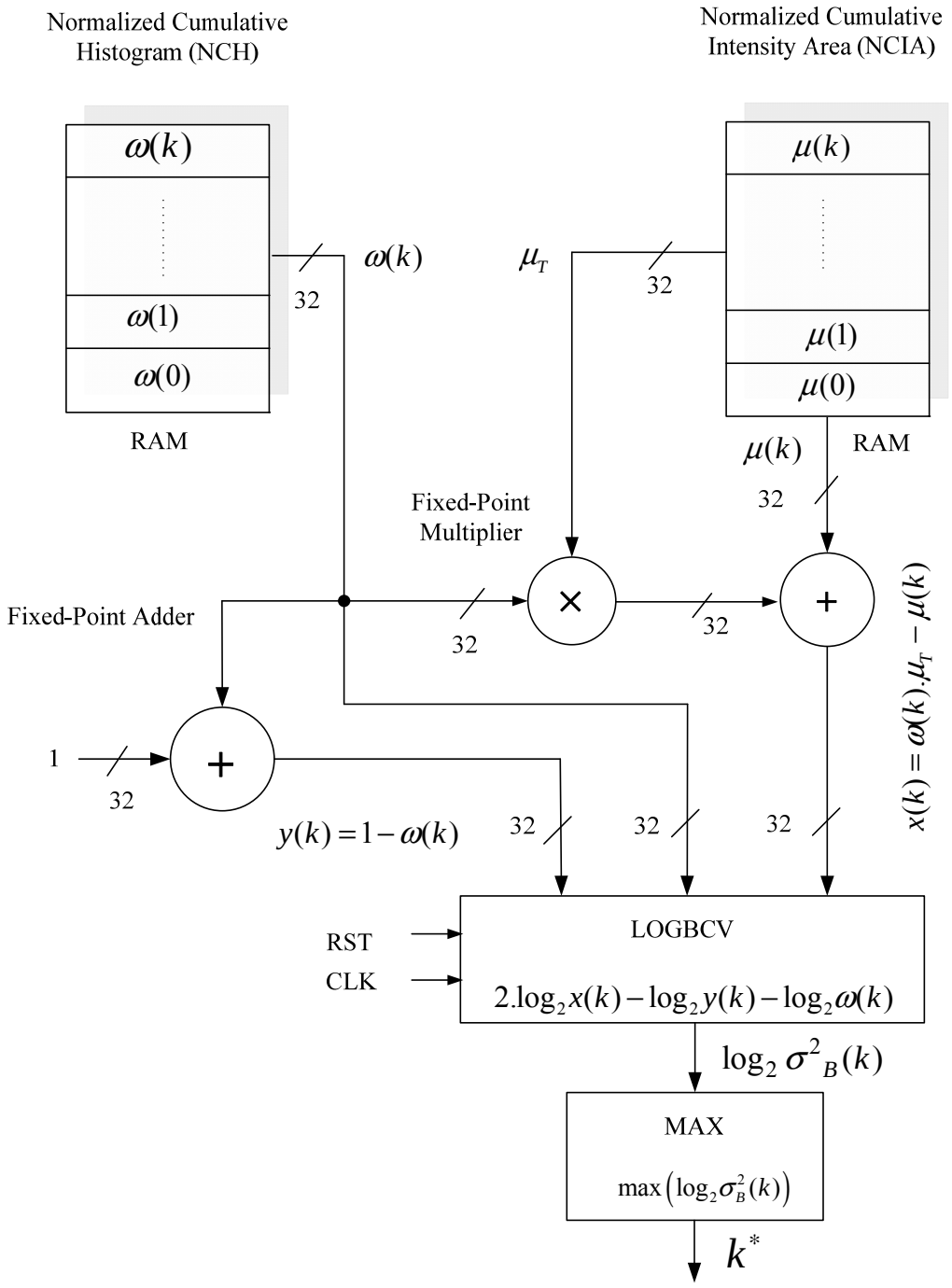


Fig. 4.4: Block diagram of the proposed architecture for computing Otsu's algorithm.

The divider to compute the BCV (4.14) is replaced by incorporating a binary logarithmic computation circuit, as direct division operation is complex, area-inefficient and slow. The modules of the architecture are implemented using fixed-point number format as explained below.

4.5.1 Fixed-Point Number Format

To use the optimized FPGA macro elements available with the FPGA device we have used fixed-point arithmetic. In the proposed architecture, most of the operations are performed in a 32-bit (16.16) unsigned fixed-point number format. Fig. 4.5 shows the format of the fixed-point number, which is same as that used in the computation of logarithm of a binary number as discussed in the Section 3.3 of Chapter 3.

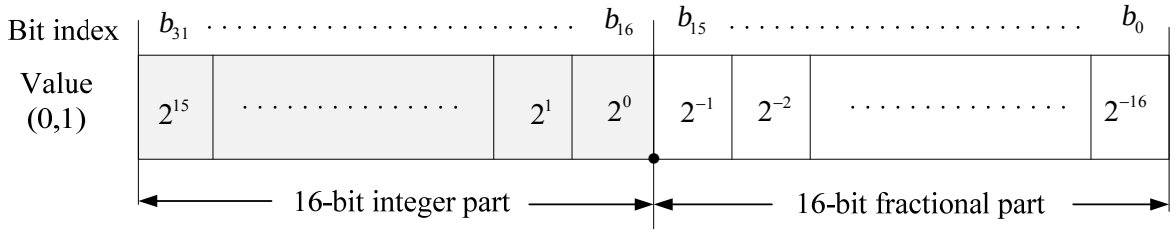


Fig. 4.5: 32-bit fixed-point number format.

4.5.2 Normalized Cumulative Histogram (NCH) Computation

Normalized cumulative histogram computation requires hardware acceleration to satisfy the high-speed needs for real-time thresholding operation. To compute a histogram in a single-cycle per pixel manner, a read-modify-write operation is needed. We can achieve a read-modify-write operation per clock cycle by incorporating a dual port BRAM memory. The single-cycle read-modify-write operation can be achieved by operating one port of the dual-port BRAM in the read-first mode and other port as a write-first mode as shown in Fig. 4.6(a) and Fig. 4.6(b) respectively.

Each memory cycle can be either a read or a write, so we need to divide each pixel clock cycle into two sub-cycles: a read cycle for getting the current value, and a write cycle for updating the memory content [**123**]. This is achieved by operating the dual ported BRAM on both the edges of the video clock. The circuit arrangement for the NCH computation is shown in Fig. 4.7. With active high enable (ENA) and write enable (WE) signals, the port A

of the BRAM operates on the rising edge of the video clock, which is applied at port CLKA in the read-first mode. Similarly, with the active high enable (ENB) and write enable (WEB) signals along with active low reset (RSTB) signal, the port B operates on the falling edge of the video clock, which is applied at CLKB port.

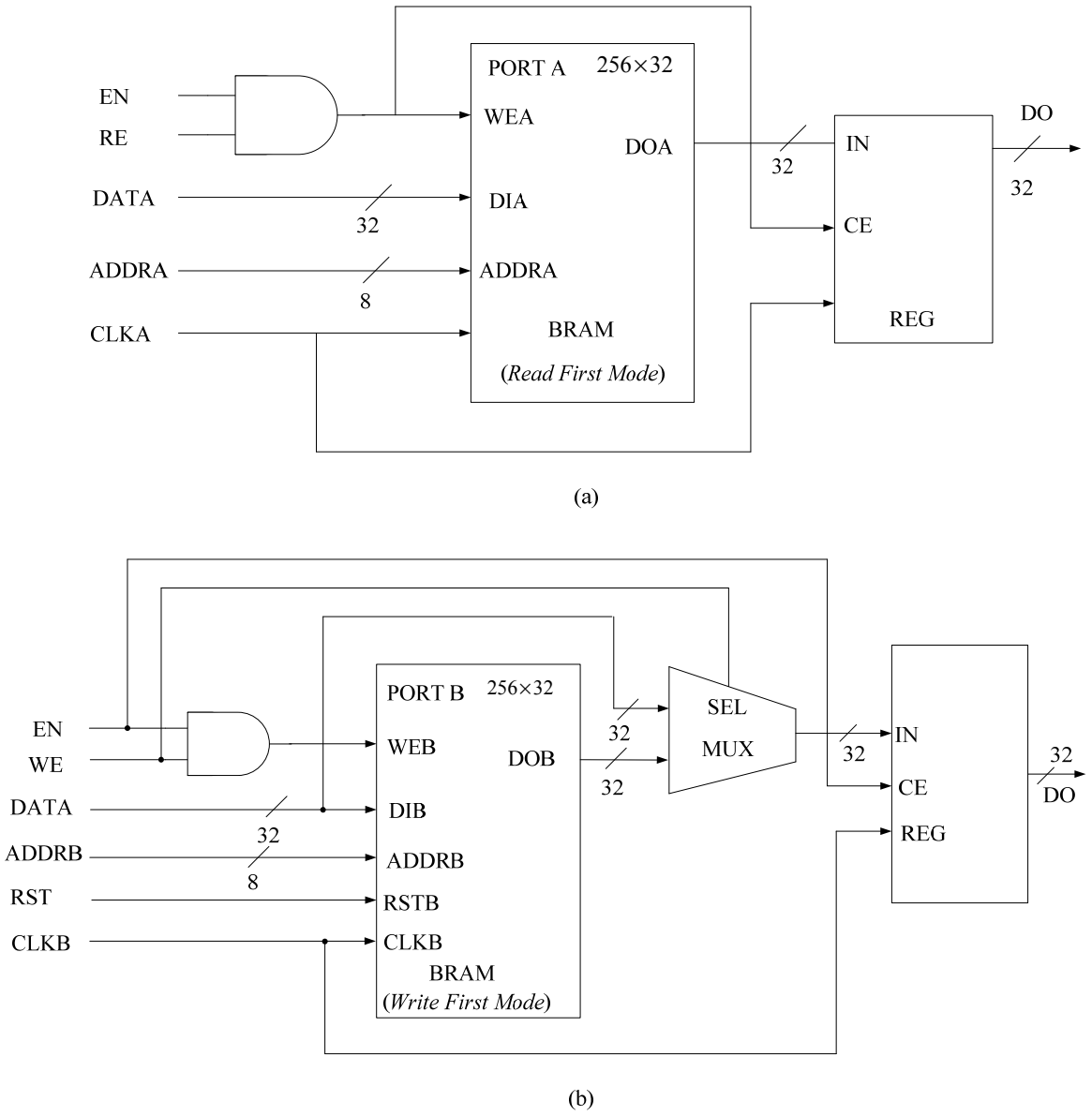


Fig. 4.6: BRAM read-write mode (a) read-first mode (b) write-first mode.

We can get the normalized cumulative histogram in the same clock cycle in which the read-modify write operation is being performed. For this, the reciprocal weight of the total number of pixels, i.e., $1/N$ is calculated. The content of the memory locations addressed by

each newly arrived pixel is incremented by the computed constant value ($1/N$). After completion of read-modify-write cycle, the BRAM memory locations hold the normalized cumulative histogram.

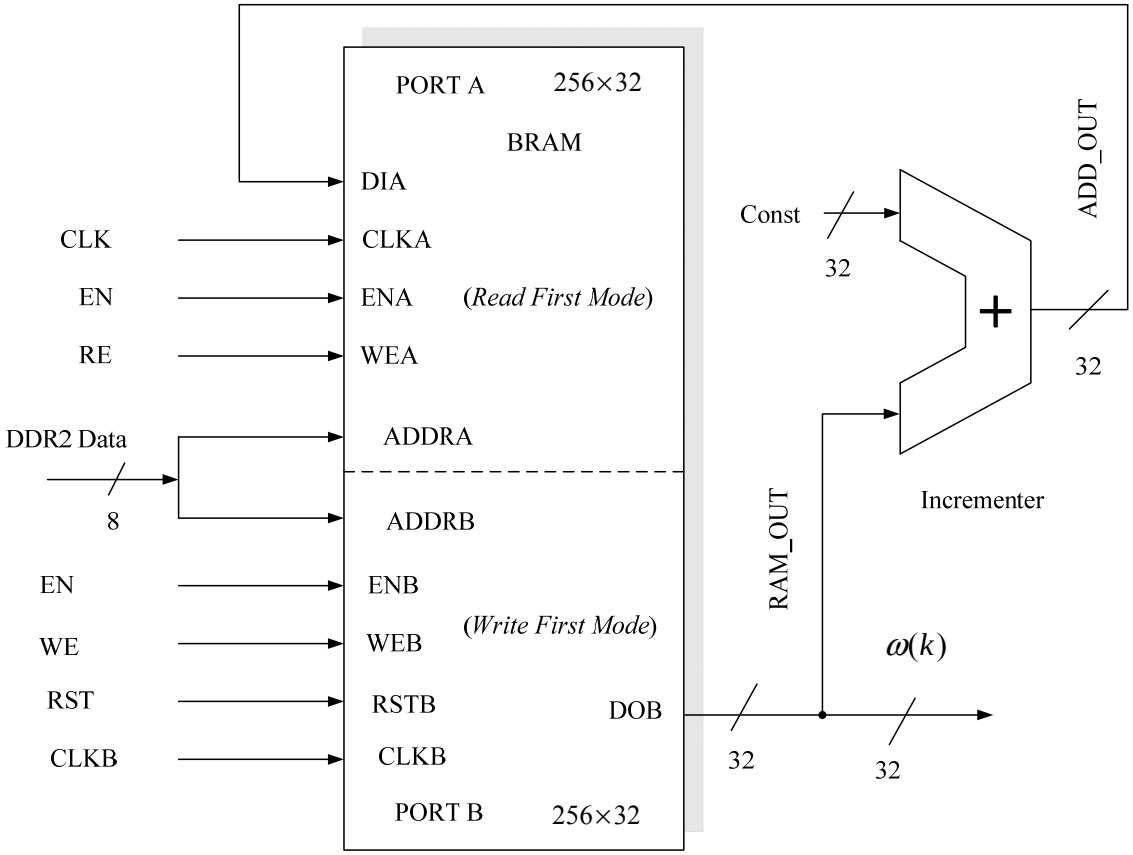


Fig. 4.7: Normalized cumulative histogram (NCH) computation block.

An example of this process is shown in the Fig. 4.8. Here for each arrival of a new pixel the respective value in NCH BRAM is incremented by $1/N$ and written back to the same memory location. By taking reciprocal value of the total pixel counts (i.e., $1/N$=1/307200=0.0000032552) we have obtained a fractional value, which can be easily represented in 32-bit unsigned fixed-point format as 0.0000369D (Hex). Here, we have used all the 32 bits for the internal datapath in 0.32 fixed-point format. Based on the data size of this constant value we have selected a 256×32 bit size dual port BRAM memory. The computed normalized cumulative histogram for all the data pixels is available in the BRAM

memory locations in 32-bit unsigned fixed-point format (16.16) as per the format shown in Fig. 4.5.

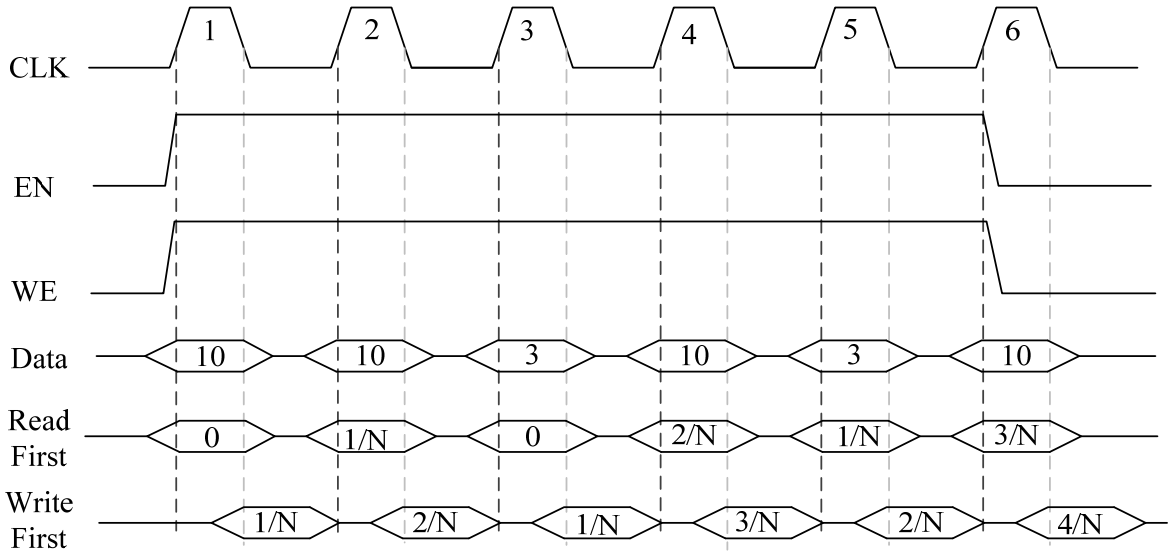


Fig. 4.8: Normalized cumulative histogram (NCH) computation timing diagram.

The ModelSim [**124**] snapshot of the normalized cumulative histogram (NCH) computation block is shown in Fig. 4.9. Here, it is shown that the BRAM which stores the NCH values are incremented with each arrival of its input data on its input port.

With the active enable (*en*) and write-enable (WE), the BRAM works at each edge of the clock (clk) and it increments its address locations at each arrival of input by 1/*N*. In the timing diagram shown in Fig. 4.9, the RAM memory locations 0, 8, 253, 254 and 255 are shown.

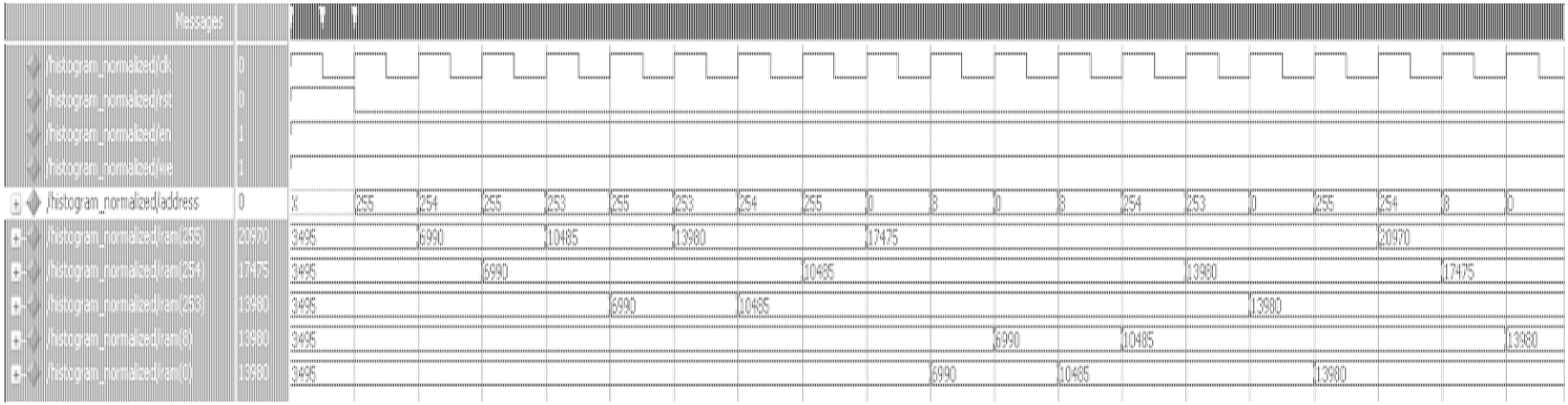


Fig. 4.9: ModelSim capture of normalized cumulative histogram (NCH).

4.5.3 Normalized Cumulative Intensity Area (NCIA) and Total Mean Computation

To compute the first-order normalized cumulative intensity area, $\mu(k)$ (4.16), we pre-calculate the constant terms, i/N, in the range of $0 \le i \le 255$. These values are stored in 256 locations of a ROM, starting from 0 to 255, in the 32-bit unsigned fixed-point format. Similar to the NCH computation, the circuit for the first-order cumulative moment and mean computation also utilizes dual-port BRAMS. The circuit arrangement for the $\mu(k)$ computation is shown in Fig. 4.10.

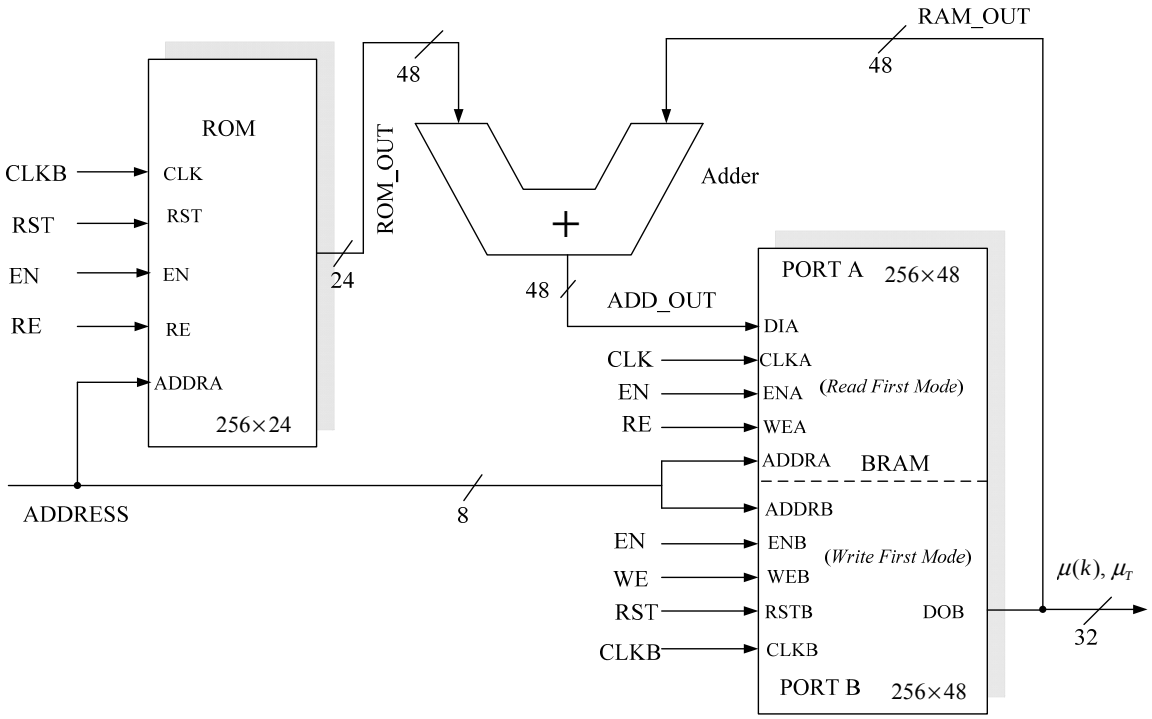


Fig. 4.10: Normalized cumulative intensity area (NCIA) total mean computational block.

In this circuit arrangement, a read-modify-write operation per clock cycle is obtained by incorporating a dual port BRAM memory, similar to the NCH computation. The data for which the cumulative moment, $\mu(k)$ and mean (μ_T), are to be computed, address the dual port BRAM at its address bus. Here, the maximum value of μ_T can consist of 16-bit integer value; whereas the minimum value of $\mu(k)$ can be represented in 32-bit fractional value. Therefore,

we have selected a 48-bit internal data width of the RAM as well as that of fixed-point adder circuits. The outputs of the circuit is in 32-bit (16.16) unsigned fixed-point format. The representation has $1.53\times e-5$ level fractional value accuracy.

4.5.4 Binary Logarithmic Between Class Variance (LOGBCV) Computation Unit

As explained in Section 4.4, the between-class variance (4.14) is calculated by taking the binary logarithm of $\sigma_B^2(k)$ (4.18), which is shown in Fig. 4.11. For the computation of binary logarithm of binary number, the architecture developed in Chapter 3 is used. In the present context, it is summarized below.

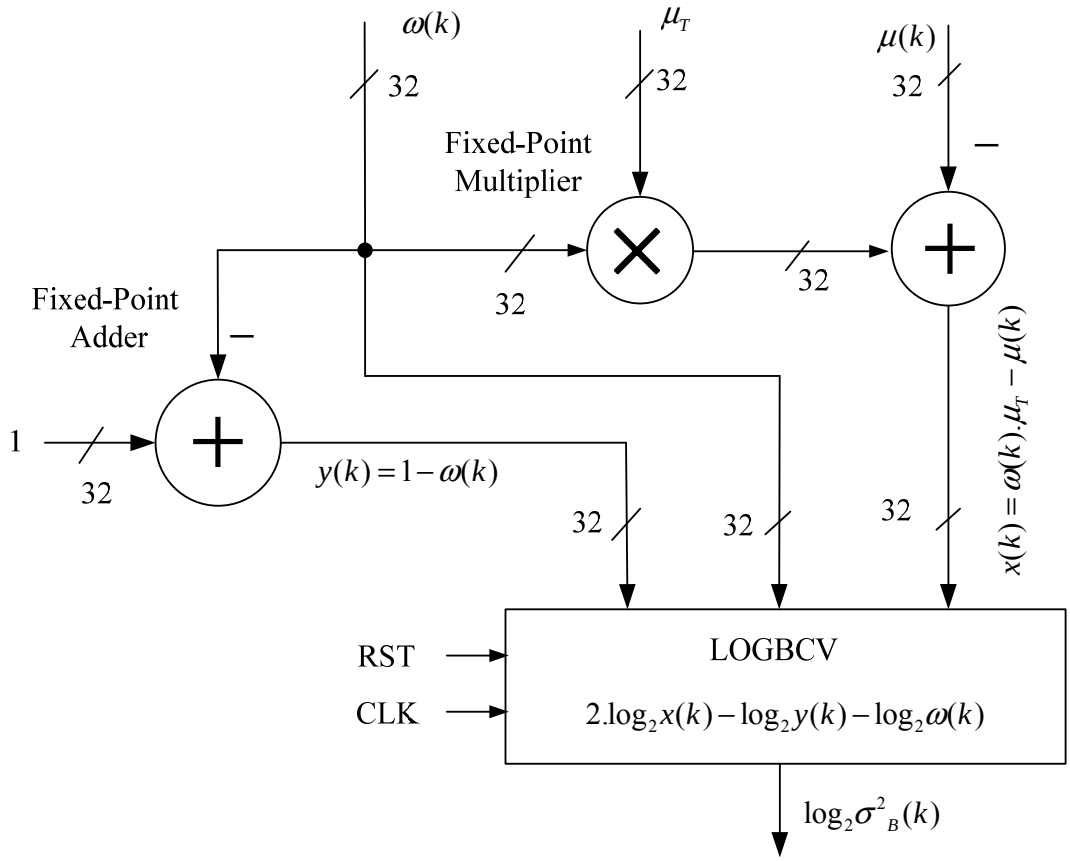


Fig. 4.11: LOGBCV computation.

To compute the logarithm of the binary number we have to compute the characteristic and mantissa parts separately as shown in Fig. 4.12. The characteristic part of the logarithmic

value (k) is obtained by incorporating a leading-one finder (LOF) circuit, whereas, the fractional part approximation unit provides the fractional part of the logarithm. The details of these units are discussed in the Chapter 3 (Section 3.4.2).

The 16-bit leading-one finder (LOF16) circuit is designed by arranging four 4-bit leading-one finder (LOF4) circuits in first stage and one 4-bit LOF in second stage. The circuit also uses four 4-bit OR gates (OR4) and four 4-bit multiplexers (MUX4). The 16-bits from the MUX4 outputs are provided to a binary encoder, which provides 4-bit binary equivalent of the leading-one bit position in the input binary sequence. The circuit arrangement of LOF16 is shown in Fig. 4.13.

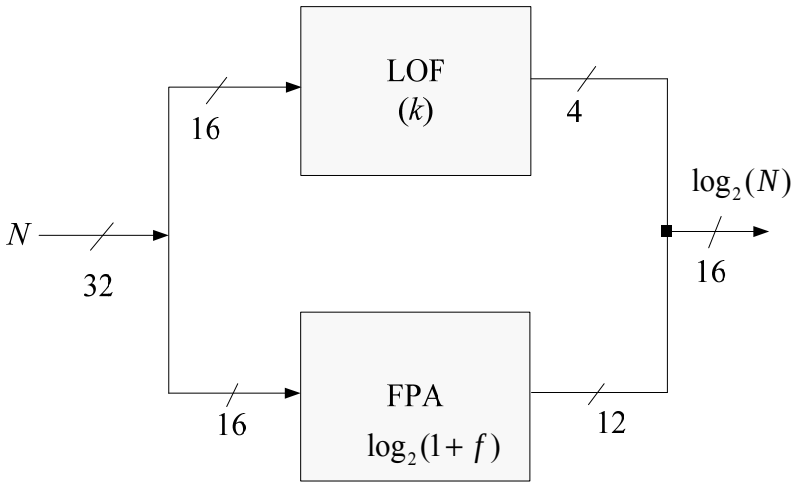


Fig. 4.12: Logarithmic conversion unit with leading-one finder and fractional part approximation units.

The LOF16 output controls a barrel-shifter (BSHFT), which sends the required bits to the fractional point approximation (FPA) unit, which is covered in the Chapter 3 (Section 3.4.4) and re-shown in Fig. 4.14. Here, in our implementation of the Otsu's thresholding algorithm, the anti-logarithm conversion circuit is not required as we are interested in finding out at which grey-level the logarithm value attains its maximal value.

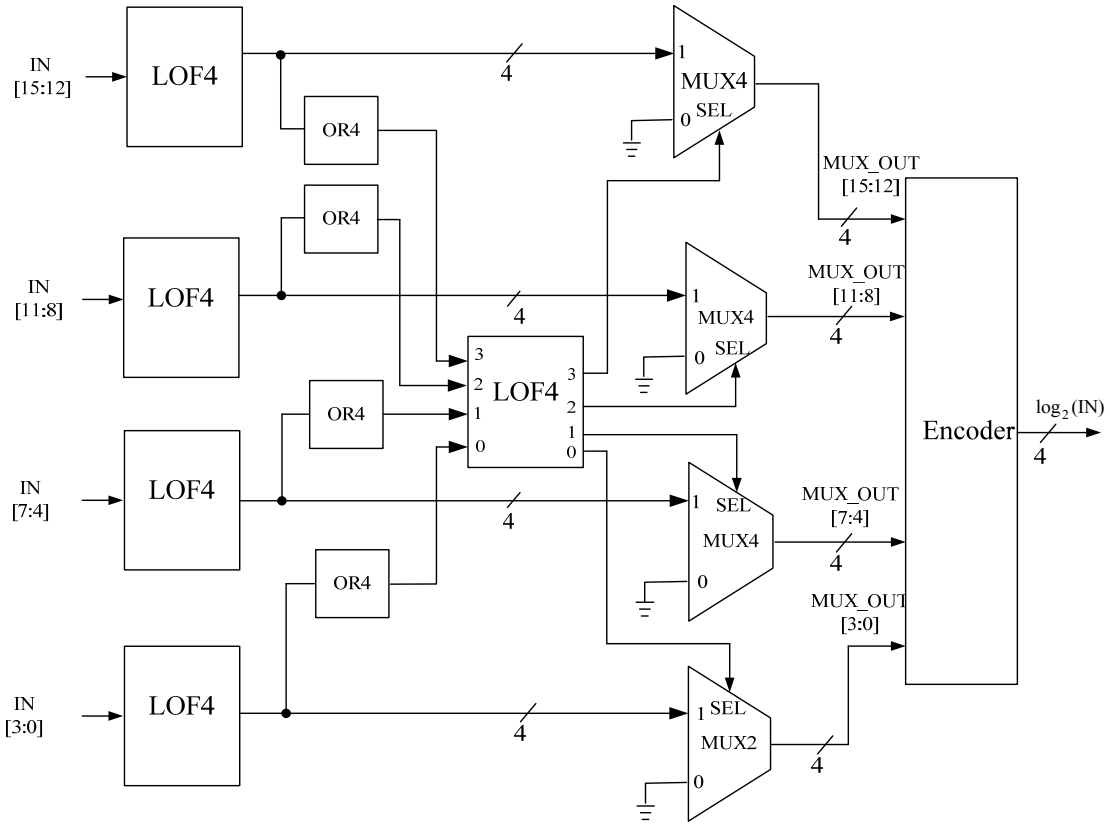


Fig. 4.13: 16-bit Leading-one finder (LOF16).

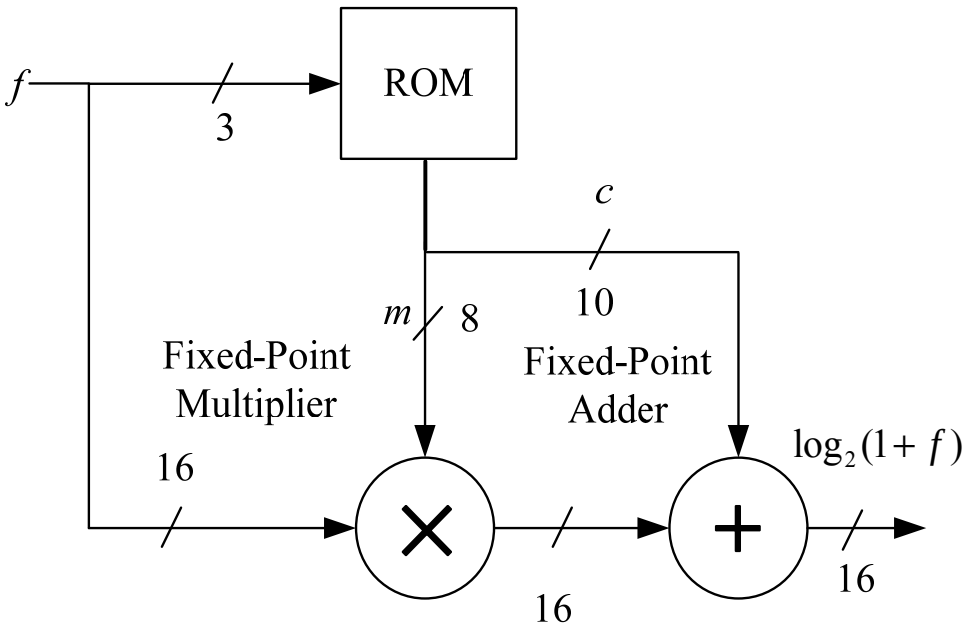


Fig. 4.14: The fractional part approximation (FPA) unit of binary logarithm computation.

4.5.5 MAX Circuit

The MAX circuit receives input from the LOGBCV unit and searches for the appropriate value of threshold (k^*) for which $\log_2 \sigma_B^2(k)$ obtains its maximum value. This is shown in Fig. 4.4. This block has been designed using a 16-bit comparator circuit.

4.6 Results and Discussion

The proposed architecture for Otsu's thresholding algorithm has been implemented by us in VHDL and synthesized using Xilinx ISE 14.2 targeted for the Xilinx Virtex-5 xc5vfx70t ffg1136-1 FPGA device. The device utilization summary for the main resource elements is shown in Fig. 4.17. In order to compare the implementation results of the proposed architecture we have selected the architecture proposed in [**50**], and [**51**]. The hardware architecture for computation of between-class variance as proposed by [**50**] and that by [**51**] had been implemented by [**51**] on Xilinx Virtex xcv800 FPGA. Table 4.1 shows the comparative results.

Table 4.1: FPGA Device Utilization for the Proposed Architecture for Threshold Computation

Elements	**Architecture [**50**]**	**Architecture [**51**]**	**Proposed Architecture**
Image Size	256×256	256×256	640×480
Image Buffer	RAM	RAM	DDR2
Area (Slices)	622/9408 (6.6%)	109/9408 (1.2%)	168/11200 (1.5%)
External IOBs	113/166 (68.1%)	49/166 (29.5%)	33/640 (5.2%)

It can be observed that the proposed architecture requires only 1.5 % of the FPGA slices for the computation of between-class variance (4.14). Along with this, to compute the cumulative mean (4.15) and moments (4.16) we are using 2.7% (4 out of 148) of the Block RAMs and 3.9% (5 out of 128) of DSP48E slices available with the Virtex-5 FPGA. The total power consumtion of the proposed thresholding architecture is 15 mW.

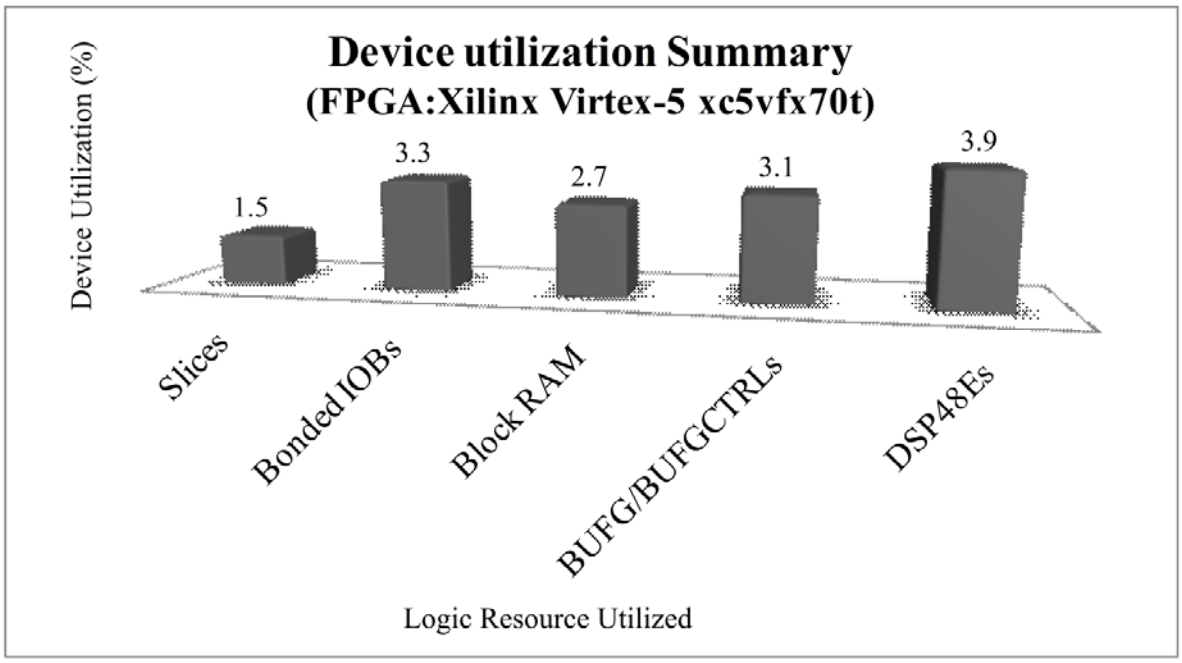


Fig. 4.15: Device utilization summary for the implementation of the thresholding architecture in the FPGA.

In addition, the number of IOBs (input/output blocks) used is also reduced. This is because, in the proposed architecture, the threshold value can be readily obtained by logarithmic approximation of between-class variance, which requires a simpler arithmetic circuit with fewer bit representations using fixed-point arithmetic. In the implementation, we have used a standard VGA resolution image of size of 640×480 pixels which is stored in the off-chip DDR2 SDRAM, whereas in the implementations of [**50**,**51**] the image has been kept on FPGA-based RAM resources.

4.7 Thresholding Unit as an IP Core and the Required System-Level Arrangement

The system-level arrangement of the image thresholding computational block as a hardware IP along with its communications with other IPs and buses is shown in Fig. 4.16. The proposed architecture uses two bus protocols for communication with the processor. The first one is a 128-bit processor local bus (PLB) protocol, which provides the infrastructure for connecting a PLB master and slave into an overall PLB system. The second bus is the

memory controller interface (MCI) bus which provides an interface between the PowerPC 440 microprocessor and a soft memory controller implemented in FPGA logic. In order to develop the required hardware and software in an integrated manner, Xilinx Embedded Development Kit (EDK) design tool has been used.

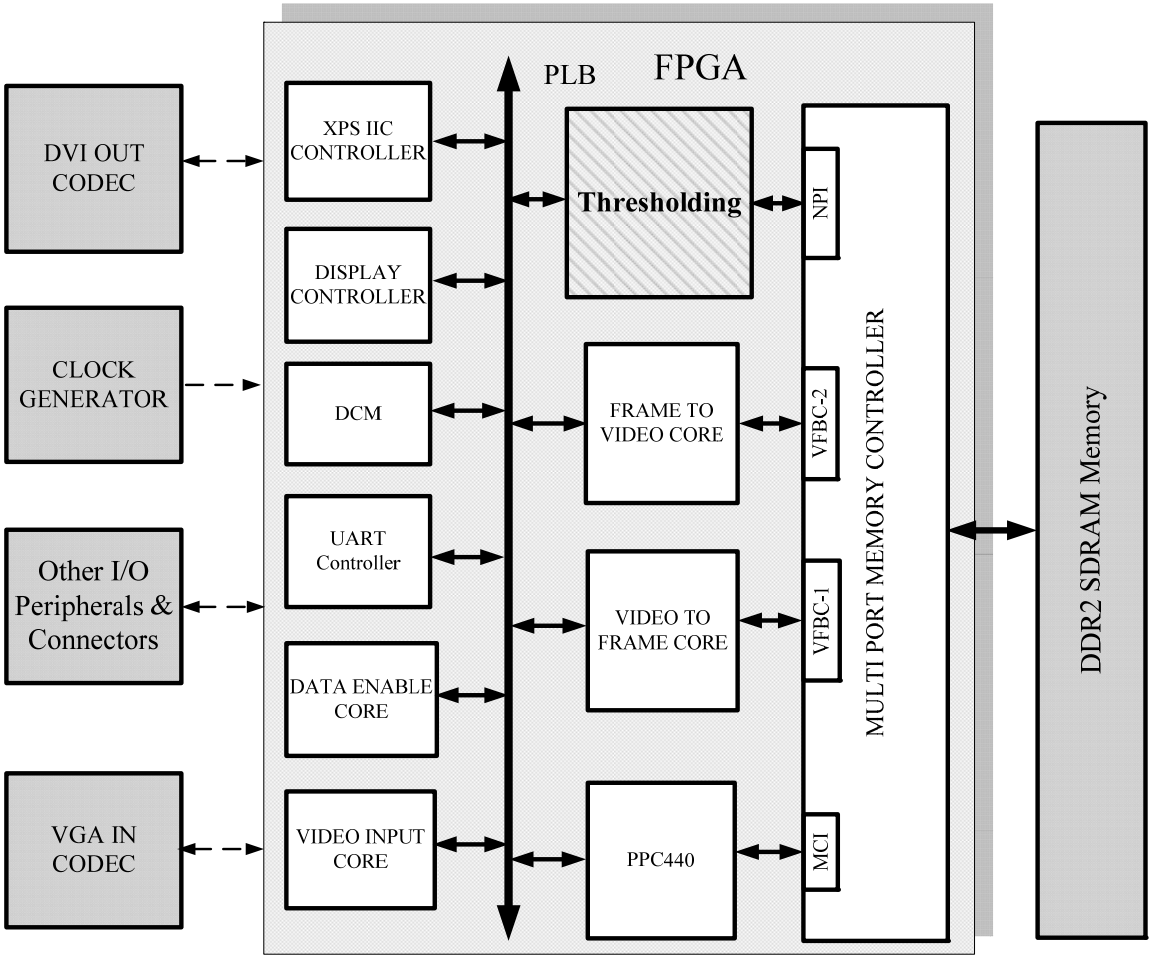


Fig. 4.16: System arrangement with the threshold computational unit.

Real-time analog video is captured from the camera with a resolution of 640×480 pixels at 60 fps. The captured data is converted into 8-bit gray level format and stored in the DDR2 SDRAM memory using embedded PowerPC 440 processor and the Xilinx video frame buffer controller (VFBC) available with its multi-port memory controller (MPMC) IP [**105**]. The system arrangement for image acquisition uses the peripherals available on the Xilinx

ML-507 platform along with some of the IP elements in a similar manner as described in Chapter 2.

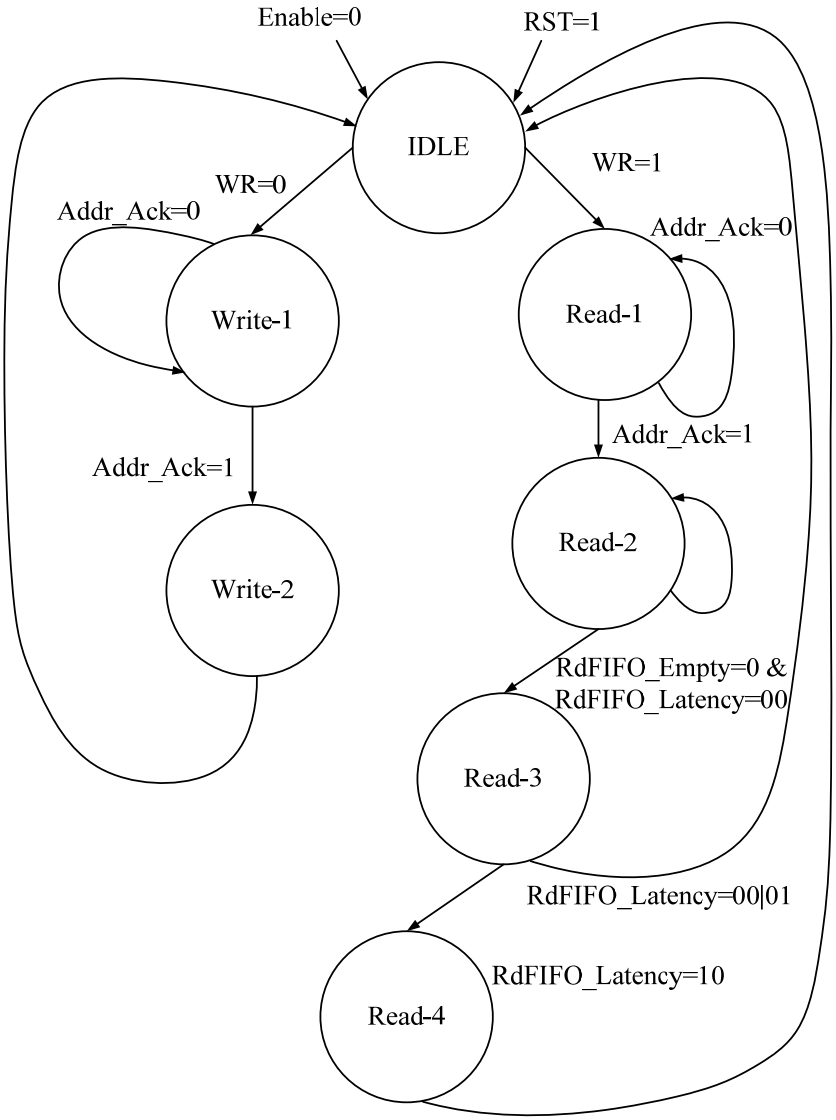


Fig. 4.17: The native port interface (NPI) protocol.

The platform contains a VGA input video codec connector that supports connectivity to an external VGA source. It utilizes an Analog Devices AD9980 video decoder device, which is programmed to generate a video clock of 25.175 MHz for the thresholding unit and other required blocks that are controlled by inter-integrated circuit (I2C) general-purpose input-output registers [**100**]. For this, the control registers of AD9980 is configured by sending data

as a master on the I2C bus controller's low-level device driver functions [**100**]. The generated video clock is routed through the digital clock manager (DCM) to the various internal design modules.

Table 4.2: Native Port Interface (NPI) Signals

Signal	Description
Addr	Indicates the starting address of a particular request.
AddrReq	Indicates that NPI is ready for MPMC to arbitrate an address request.
RNW	Read/Not Write; 0 = Write request; 1 = Read request.
Size	0x00 = Word transfers (32-bit NPI).
RdModWr	Read/ modify/write.
InitDone	Initialization is complete and FIFOs are available for use.
AddrAck	Indicates that MPMC has begun arbitration for address request.
WrFIFO_Data	Data to be pushed into MPMC write FIFOs.
WrFIFO_BE	Indicates which bytes of *WrFIFO_Data* to write.
WrFIFO_Push	Indicates push *WrFIFO_Data* into write FIFOs .
RdFIFO_Data	Data to be popped out of MPMC read FIFOs.
RdFIFO_Pop	Indicates that read FIFO fetch the next value of *RdFIFO_Data.* (Must be asserted for one cycle of MPMC clock.)
RdFIFO_Empty	When 0, it indicates that enough data is in the read FIFOs to assert.
RdFIFO_Latency	Indicates the number of cycles from the time *RdFIFO_Pop* is asserted and/or RdFIFO_Empty is de asserted until *RdFIFO_Data* and *RdFIFO_RdWdAddr* are valid.
	0 =*RdFIFO_Data* and *RdWdAddr* are valid in the same cycle as the assertion of *RdFIFO_Pop*.
	1=*RdFIFO_Data* and *RdFIFO_RdWdAddr* are valid in the cycle following the assertion of *RdFIFO_Pop*.
	2=*RdFIFO_Data* and *RdFIFO_RdWdAddr* are valid two cycles following the assertion of *RdFIFO_Pop*.

The application software, written in 'C' language, runs on top of a standalone software platform and controls all the hardware blocks and platform peripherals through PowerPC processor. We have utilized the application programmer interface (API) offered by the

software platform. The thresholding block communicates with the DDR2 SDRAM memory through a 32-bit native port interface (NPI) which is synchronized with the MPMC controller. NPI protocol is shown in Fig. 4.17. The signals used in NPI protocol are shown in Table 4.2.

4.8 Conclusion

An FPGA-based architecture for the computation of Otsu's normalized cumulative mean, moment and between-class variance is presented. The proposed architecture is implemented on Xilinx Virtex-5 xc5vfx70tffg1136-1 FPGA device available with the Xilinx ML-507 FPGA platform. The system operates at standard VGA clock frequency of 25.175 MHz, for the frame size of 640×480 pixels at 60 frames per second. To save the system resources, we have created a very simple and efficient datapath, which does not contain any complex hardware building blocks. In the proposed architecture, most of the operations are performed on the 32-bit unsigned fixed-point numbers, requiring only a single-cycle per operation. The architecture has the advantages of minimizing logic resources and the processing of large datasets, by conducting time critical processes on BRAMS and DSP slices. The total device utilization summary shows that, the total FPGA resources utilized are around only fourteen percent (14%). The remaining FPGA resources are sufficient for implementing many practical real-time image and video processing applications. The power consumption of the proposed architecture for threshold computation is 15 mW. In order to manage the required hardware IPs and configuration software in an integrated manner, Xilinx Embedded Development Kit (EDK) design tool has been used.

CHAPTER 5

CONNECTED COMPONENT LABELING ALGORITHM AND ITS POWERPC IMPLEMENTATION

5.1 Introduction

In binary image analysis, objects are usually extracted by means of connected components or region labeling operation [**3**,**8**,**89**]. The connected components are defined as regions of adjacent foreground pixels that have the same input label (value). The output of connected component analysis operation is a labeled image in which distinct objects have unique labels, which distinguishes them for the system (processor) recognition. The binary image obtained from Chapter 4 is used by connected component labeling algorithm to segment the object for tracking application, which has been described in detail in Chapter 6. In this chapter, we propose an improved label-equivalence based two-scan connected component labeling algorithm which improves upon an existing algorithm [**76**] and implement the same in the embedded PowerPC processor available in the Xilinx Virtex-5 FPGA device.

Labeling connected components in a binary image is one of the most essential operations in the field of image processing, pattern recognition and computer vision [**125**,**126**,**127**,**9**]. Once objects are individually labeled, they can be separately processed, modified or used for further image processing applications. The connected component analysis can be used in a variety of applications, such as, finding individual letters in a scanned document, object recognition and its tracking [**7**,**128**,**129**,**68**], face recognition, fingerprint identification, automated inspection, computer-aided diagnosis [**130**,**131**,**132**], video and signal based surveillance, barcode recognition, and medical image analysis, [**74**,**1**].

Raster-scan and label-equivalence resolving based algorithms are one of the most popular categories of labeling algorithms. These algorithms can be either pixel-based [**76**], [**133**,**134**,**135**,**136**] or run-based [**137**,**138**,**139**,**140**,**141**,**142**]. The pixel-based method resolves label equivalences between pixels whereas the run-based ones resolve label equivalences between the block of consecutive object pixels, i.e., runs. Other important labeling algorithms such as, searching and label propagation algorithms [**143**,**144**,**145**,**146**,**147**] and the contour tracing algorithms [**148**,**149**], process an image in an irregular manner. In hierarchical tree-based algorithms, the provisional label to a pixel is assigned as per its surrounding neighbors. The searching of neighbors is based on a decision tree structure [**150**,**151**,**152**,**153**,**154**] which can be an exhaustive search. Some of the parallel algorithms are specifically developed for the parallel machines, which are based on divide-and-conquer approach [**155**,**156**,**157**]. These algorithms are unsuitable for applications which use simple computer architectures.

In a digital system, images are generally scanned in a raster fashion. Therefore, in order to label the connected component pixels, most of the algorithms rely on raster-scan and label-equivalence resolving method [**133**,**134**,**135**,**136**]. This simple method is sequential in nature and is widely used digital image processing, [**134**,**143**]. Moreover, the raster-scan algorithms are quite suitable for pipeline processing [**130**].

In the raster-scan and label-equivalence based algorithms there are various methods to handle the equivalences associated with the foreground pixels, which are discussed in [**76**] and [**144**,**145**]. Among these methods, the class-based label-equivalence resolving approach as proposed in [**144**,**145**] and expanded by Stefano and Bulgarelli (SB) in [**76**] is found to be very efficient. In this two-scan based approach, a class is defined to be a simple one-dimensional array which can be as large as the maximum number of provisional labels. With a class identifier associated with each label, equivalences are processed during the first scan

by merging equivalence classes as they arrive. Subsequently, in the second scan the connected component labeling is performed [**76**].

In the SB algorithm, the main advantage of checking for new equivalence in the class domain is to exploit the transitive property for class merging. Whenever two classes are merged, the class identifier of the survivor equivalent class modifies the class identifier of the deleted equivalent class. In various cases, the SB algorithm fails to merge *all* the members of the deleted class as implied by the transitive property. These cases occur during first scan when an expanding label set of a connected component runs across a new equivalence, which involves a label (from the growing label set) other than the maximum of expanding label set. Partial merging occurs in such cases, in which a few of the labels from deleted class move to the survivor class, while the others are left behind due to an improper equivalence handling mechanism.

In this chapter, we propose an improved label-equivalence based two-scan connected component labeling algorithm, which improves upon the SB algorithm and eliminates the partial merging problem. This is achieved by modifying the equivalence handling loop of SB algorithm such that full merger of equivalences is accomplished. Some of the random binary test patterns and standard gray scale images [**158**] which are converted to binary using Otsu's method of thresholding [**49**] are used to test the improved SB algorithm in 4-connectivity case and its performance is compared with that of the SB algorithm. The results show that our improved SB algorithm handles the equivalent class conflicts efficiently, has lower conflicts and gives the correct number of connected components.

The raster-scan based connected component labeling algorithm, such as the SB and its improved version, are sequential in nature and do not need any computational resource other than those required for decision processing. The algorithm essentially works on the selection

decisions and looping, which can be easily handled by a processor. Based on the above reasoning, we have chosen to implement the improved SB algorithm, proposed in the chapter in the embedded PowerPC 440 processor available in the Xilinx Virtex-5 FPGA of ML-507 [**34**].

The rest of this chapter is organized as follows. The basics of connected component labeling process and related algorithms are discussed in Section 5.2. Section 5.3 introduces the improved SB algorithm. Section 5.4 provides details of the improved SB algorithm. A comparative analysis with the SB algorithm is given in Section 5.5. This section also discusses the results obtained with artificial binary test patterns and with standard gray scale images. Section 5.6 gives the PowerPC implementation results and finally, Section 5.7 concludes the chapter.

5.2 Two-scan Connected Component Label-Equivalence Process

The basic terminology of the two-scan connected component label-equivalence algorithm is discussed in Subsection 5.2.1. Subsection 5.2.2 covers the conventional pixel-based two-scan label-equivalence algorithms. The outline of Stefano-Bulgarelli's algorithm is given in Subsection 5.2.3.

5.2.1 Basic Terminology

Let *I* be a binary image with '1' representing the foreground pixels and '0' representing the background pixels. A pixel value at position (x, y) is represented by $P(x, y)$. The definition of connected component depends on the pixel's surroundings. Two pixels, $P(x, y)$ and $Q(x, y)$, are connected if there exists a path of pixels $(P_0, P_1, ...P_m)$ such that $P_0 = P$, $P_m = Q$. The other pixels in the path are known as the surrounding (neighbor) pixels such that P_i is a neighbor of P_{i-1} for $1 \le i \le m$. A connected component may be 4-connected or 8-

connected. In a 4-connected situation, any pixel with coordinate (x, y) has at least one element in that connected component having coordinates in 4-neighbor set as:

$$\{(x, y-1), (x, y+1), (x-1, y), (x+1, y)\} \tag{5.1}$$

similarly the 8-connected component has at least one element having coordinates in 8-neighbor set as:

$$\{(x-1, y-1), (x-1, y), (x-1, y+1), (x, y-1), (x, y+1), (x+1, y-1), (x+1, y), (x+1, y+1)\} \tag{5.2}$$

Labeling of connected component is an operation where groups of connected pixels (connected component) of a binary image are classified as different objects with unique labels. Let $n (n \in N)$ represent the index of a connected component in the image and CC_n represent the individual connected components.

In the labeling process, we assign a unique label to each connected component. The resultant labeled image consists of various connected components in which a unique label is assigned to pixels belonging to the same connected component and different labels are assigned to distinct components. The labeled image can be represented as:

$$I(x, y) = \begin{cases} B & \text{if } P(x, y) = B \\ n & \text{if } P(x, y) \in CC_n \end{cases} \tag{5.3}$$

Thus, an input binary image is transformed into a frame in which the foreground pixels are modified into labels which indentify the connected components. In a raster scan, the labeling of a pixel $\{x, y\}$ is done with the help of its neighboring pixels, which have already been run across by the raster scan. In 4-connectivity case, such neighbors are $\{p, q\}$ as shown in the Fig. 5.1 (a), where p is a pixel at $\{(x-1), y\}$ and q is a pixel at $\{x, (y-1)\}$ of (5.1). Similarly, in

the case of 8-connectivity, the already labeled pixels are $\{p, q, r, s\}$.In this case the pixels p, q, r and s are located at positions $\{(x-1),(y-1)\}$, $\{(x-1), y\}$, $\{(x-1),(y+1)\}$ and $\{x,(y-1)\}$ respectively, as given in (5.2) and shown in Fig. 5.1 (b). The following sub-section discusses some of the pixel-based conventional two-scan label-equivalence algorithms.

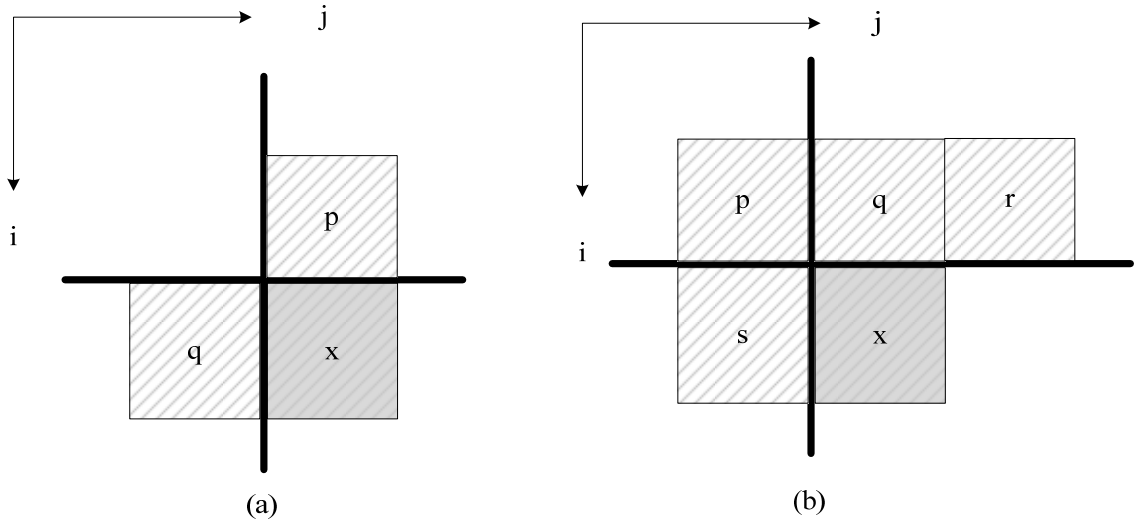


Fig. 5.1: Pixel connectivity (a) 4-connectivity (b) 8-connectivity.

5.2.2 Pixel-based Conventional Two-Scan Label-Equivalence Algorithms

Pixel-based conventional raster-scan and label-equivalence algorithms scan an image in the raster fashion. During the first raster-scan, a provisional label is assigned to the foreground pixels [**76**], [**134**,**139**], [**144**,**145**]. After the first scan, a connected component may consist of many provisional labels. Therefore, a second scan is required to assign a unique label to each component [**76**,**139**]. After completion of second scan, the new image is segregated into various connected components, each marked distinctly by a unique label.

The classical two scan labeling algorithm [**139**] processes label equivalences after completion of the first scan. To improve efficiency of the labeling process, a class array can be used [**76**,**144**,**145**]. Class-based equivalence resolving algorithms for connected component labeling are presented in [**144**,**145**]. To reduce the complexity of the earlier

algorithms and to expedite the labeling process, an improved connected component labeling algorithm was presented by Stefano-Bulgarelli (SB) [**76**]. In this algorithm, equivalences are processed during the first scan itself (rather than processing the equivalences in the second scan). The following section discusses the SB algorithm in detail.

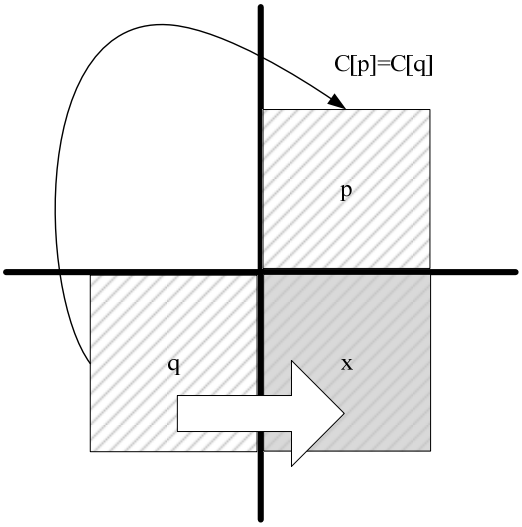


Fig. 5.2: Processing of equivalences in the first scan.

5.3 The Stefano-Bulgarelli's Algorithm

This algorithm processes the equivalences in the first pass to determine the equivalent classes associated with the labels. The equivalence handling mechanism uses a class for this purpose, which is a one-dimensional array and can be as large as maximum number of labels. A class exists for every provisional label assigned. In the first raster scan, the pixel under scan is labeled with the help of its neighboring pixels and the conflict between the labels of neighboring pixel is resolved instantly as shown in Fig. 5.2. Merging of equivalence classes as soon as a new equivalence is found, improves the efficiency of the labeling process. It happens because the equivalence check is carried out in the class domain rather than in the label domain [**76**].

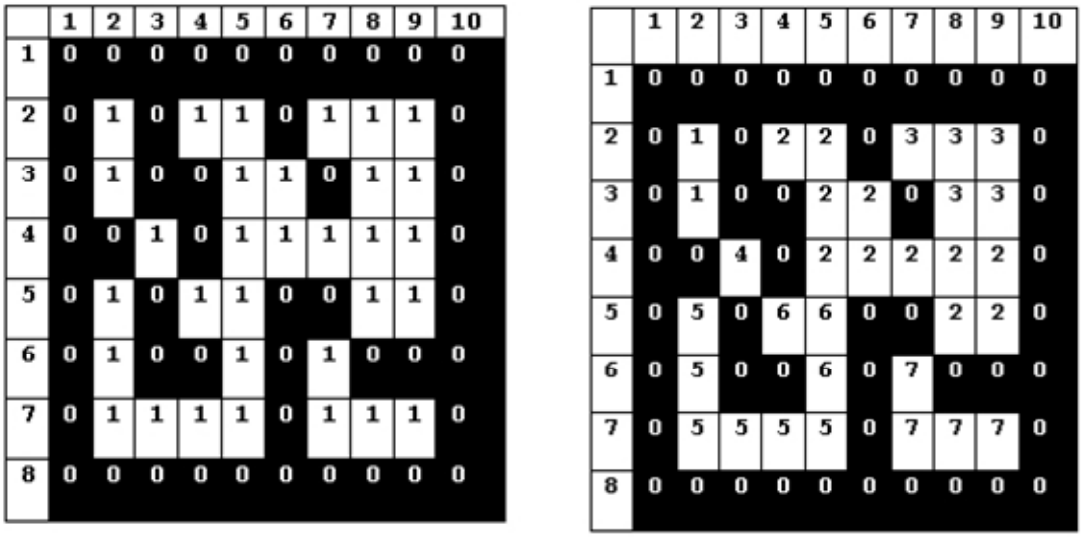
```
// C has been initialized as C[i]=i;
//First SCAN
for (i=1;i<ROWS-1;i++)
for (j=1;j<COLUMNS-1;j++){
if(I[i][j]==1){
lp=I[i-1][j];
lq=I[i][j-1];
if (lp==0 && lq==0){
NewLabel++;
lx=NewLabel;}
else if ((C[lp]!=C[lq])&&(lp!=0)&&(lq!=0)){
for (k=0;k<=NewLabel;k++)
if (C[k]==C[lp])C[k]=C[lq];
lx=lq;}
else if (lq!=0)lx=lq;
else if (lp!=0)lx=lp;
I[i][j]=lx;}}
//Second Scan
for (i=0;i<7;i++)
for (j=0;j<10;j++)
if (I[i][j]!=0)I[i][j]=C[I[i][j]];}
```

Fig. 5.3: C-Code for two-scan Stefano-Bulgarelli's (SB) algorithm.

After completion of first scan, the class array holds the updated class identifiers associated with corresponding provisional labels. A second scan is run over the image by replacing temporary labels with the class identifier of its equivalence class. Fig. 5.3 shows C code of the algorithm in the case of 4-connectivity [**76**].

To validate the efficiency of the above algorithm, we have taken a number of artificial test patterns and standard binary images which will be explained in later sections. Out of

various binary test patterns, a simple test pattern, as shown in Fig. 5.4(a), is used to discuss this algorithm in detail.



	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	1	0	1	1	0	1	1	1	0
3	0	1	0	0	1	1	0	1	1	0
4	0	0	1	0	1	1	1	1	1	0
5	0	1	0	1	1	0	0	1	1	0
6	0	1	0	0	1	0	1	0	0	0
7	0	1	1	1	1	0	1	1	1	0
8	0	0	0	0	0	0	0	0	0	0

(a)

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	1	0	2	2	0	3	3	3	0
3	0	1	0	0	2	2	0	3	3	0
4	0	0	4	0	2	2	2	2	2	0
5	0	5	0	6	6	0	0	2	2	0
6	0	5	0	0	6	0	7	0	0	0
7	0	5	5	5	5	0	7	7	7	0
8	0	0	0	0	0	0	0	0	0	0

(b)

Fig. 5.4: Two-scan labeling in the SB algorithm (a) artificial binary pattern (b) provisional labeling.

The provisional labeling of the test pattern, after the first pass, is shown Fig. 5.4(b). In Fig 5.4(b), when a pixel at pixel position (4, 8) had been scanned, a conflict occurred between label 3 and label 2 (refer the label mask shown in Fig. 5.1(a) for the 4-connectivity case). As per SB algorithm, the two labels were held equivalent and their corresponding classes were merged (as in Fig. 5.2). The class of label 2 (i.e., $C[2] = 2$) was transferred to the class of label 3, therefore, $C[3] = 2$. Similar equivalences occur at pixels (5, 5) and (7, 5). The processing of equivalences associated with the labels of Fig. 5.4 (b) is shown in Table 5.1.

In various cases, the SB algorithm fails to merge all the members of the deleted class as implied by the transitive property. Such failure occurs because of partial merging as explained here. Let us consider a case in which an equivalence (or conflict) of labels l_j and l_k occurs, where $l_k > l_j$, and the class of l_k i.e., $C[l_k]$ is replaced by class $C[l_j]$ in the class array.

Suppose a new conflict occurs between labels l_i and l_j where $C[l_i]$ is the survivor class of the two. In such cases, all the members of the equivalent class $C[l_j]$ should be merged with members of equivalent class $C[l_i]$ and therefore, $C[l_j]$ is replaced with $C[l_i]$ in the class array. However, all members of the equivalent class $C[l_j]$ are not merged due to a problem in equivalence handling mechanism of SB algorithm. As depicted in Fig. 5.3, the inner *for loop*, used for maintaining equivalences, merges the label l_x from $C[l_j]$ to $C[l_i]$, *iff* $C[l_x] = C[l_j]$ and $l_x < l_j$ (and not in case when $l_x > l_j$). It happens because the loop changes the class of label l_j from $C[l_j]$ to $C[l_i]$, at the count of l_j and therefore, label l_k $(l_k > l_j)$ now belong to a different class from label l_j, and hence not merged to equivalence class $C[l_i]$. Due to this reason, in case of various geometric patterns and images, the algorithm presented in [**76**] fails to connect the components correctly.

Table 5.1: Processing of Equivalence Classes as in the SB Algorithm.

Position	NewLabel	l_p	l_q	l_x	$C[0]$	$C[1]$	$C[2]$	$C[3]$	$C[4]$	$C[5]$	$C[6]$	$C[7]$
(1,1)	0	0	0	0	0	1	2	3	4	5	6	7
(4,8)	4	3	2	2	0	1	2	**2**	4	5	6	7
(5,5)	6	2	6	6	0	1	**6**	2	4	5	6	7
(7,5)	7	6	5	5	0	1	**5**	2	4	5	**5**	7

The SB algorithm is applied to the test pattern in Fig. 5.4(a). The partial merging problem in SB algorithm is shown in Fig. 5.5(a). The result of SB algorithm shows five numbers of connected components, while the correct count of connected components is four as shown in Fig. 5.5 (b) and discussed in the next section. As a result of this problem, a single connected component labeled '5' is split into two components labeled as '5' and '2', which is not the

case. To remove the limitations of SB algorithm [**76**], we propose an improved and efficient algorithm for equivalence handling as discussed in the next section.

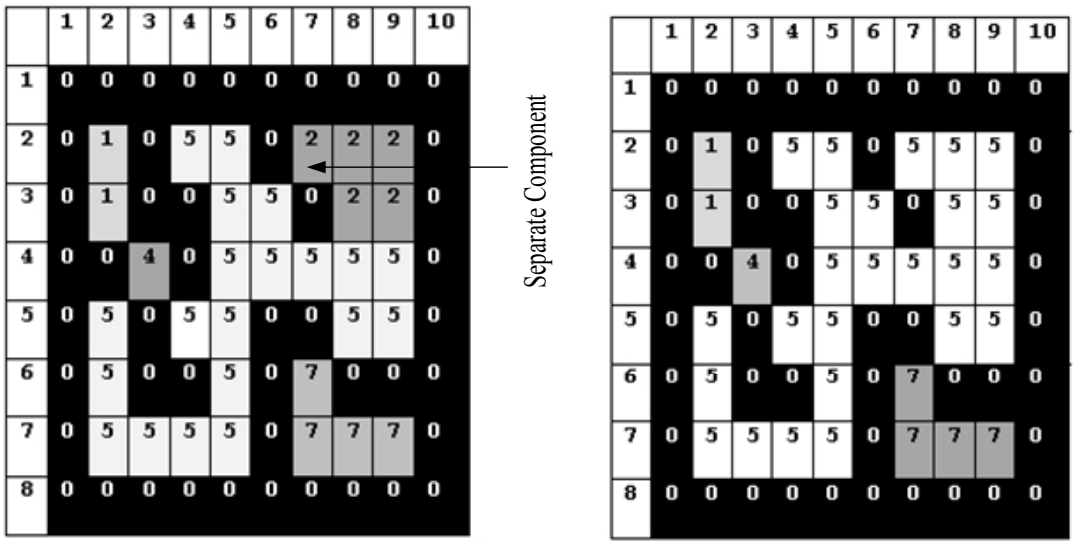


Fig. 5.5: Two-scan labeling algorithm results for (a) SB Algorithm (b) The improved SB algorithm.

5.4 Improved SB Algorithm

The algorithm being proposed improves the SB algorithm and eliminates partial merging problem. It is achieved by modifying the equivalence handling loop of SB algorithm to realize complete merging. To resolve the equivalences, we have used the notion of class identifier in this algorithm which is similar to [**76**], [**144**,**145**]. Along with the class identifier, we have also selected the 4-connectivity case so that we can compare the improved SB algorithm with [**76**].

Complete merging necessitates previous equivalences must also be considered for modification along with current modification. In case of class merger in the current scan, all the labels associated with current deleted class move to current survivor class. An important thing to consider is that this current deleted class must have been a survivor class when it has

encountered equivalence in a previous scan. It implies that the members of a current deleted class also include labels that it has acquired from all the previous conflicts. It is, therefore, necessary that all the labels of the current deleted class (including the one acquired from all the previous conflicts) must move to current survivor class. Complete merging is achieved when all labels from deleted class move to the survivor class.

```
// C has been initialized as C[i]=I;
//First SCAN
for (i=1;i<7;i++)
for (j=1;j<10;j++){
if(I[i][j]==1){
lp=I[i-1][j];
lq=I[i][j-1];
if (lp==0 && lq==0){
NewLabel++;
lx=NewLabel;
else if ((C[lp]!=C[lq])&&(lp!=0)&&(lq!=0))
for (k=0;k<=NewLabel;k++){
if (C[k]==C[lp]){
if (k!=lp)C[k]=C[lq];}
if (k==NewLabel)C[lp]=C[lq];
lx=lq;}
else if (lq!=0)lx=lq;
else if (lp!=0)lx=lp;
I[i][j]=lx;}}
//Second Scan
for (i=0;i<7;i++)
for (j=0;j<10;j++)
if (I[i][j]!=0)
[i][j]=C[I[i][j]];}
```

Fig. 5.6: C-code for the improved SB algorithm.

The C code for the improved SB algorithm is given in Fig. 5.6 and it is discussed here for thecase of example given in subsection 2.3. In this algorithm the inner *for loop*, used for maintaining equivalences is modified to handle all kinds of conflicts. It merges label l_x from $C[l_j]$ to $C[l_i]$, *iff* $C[l_x] = C[l_j]$ irrespective of whether $l_x < l_j$ or $l_x > l_j$. Only after merging all such l_x, the class of label l_j is changed from $C[l_j]$ to $C[l_i]$ and thus the problem of partial merging is avoided.

The various different cases are covered with the help of previous artificial binary test pattern which is given in Fig. 5.4(a). The various conflicts associated with the two labels of Fig. 5.4(b) are shown in Table 5.2. When a pixel at pixel position (4, 8) is scanned, a conflict occurs between label '3' and label '2'. In this case, the two labels are held equivalent and their corresponding classes are merged. That is, the class of label '2' (*C* [2] =2) is transferred to the class of label '3', so *C* [3] =2. Now when the pixel position (5, 5) is scanned, a conflict occurs between label '6' and label '2'. The two labels are again held equivalent and therefore, *C* [2] must get the class of *C* [6]. As *C* [3] had the class of *C* [2], it must also be updated with the class of *C*[6]. Since label '3' is greater than label '2', this would have posed a partial merging problem in SB's case. However, in the improved SB algorithm both *C* [2] and *C* [3] get the class of *C* [6].

The processing of equivalence classes in the improved SB algorithm is shown in Table 5.2. In the context of the labeled image shown in Fig. 5.4(b), class *C* [3] is modified with the class of *C* [5] which is shown in Table 5.2. When the first scan is over, each provisional label is changed to their corresponding class representative as is shown in Fig. 5.5(b). In the next section, we demonstrate the experimental results obtained with various artificial test patterns and standard images.

Table 5.2: Processing of Equivalence Classes in the Improved SB Algorithm.

Position	NewLabel	l_p	l_q	l_x	$C[0]$	$C[1]$	$C[2]$	$C[3]$	$C[4]$	$C[5]$	$C[6]$	$C[7]$
(1,1)	0	0	0	0	0	1	2	3	4	5	6	7
(4,8)	4	3	2	2	0	1	2	**2**	4	5	6	7
(5,5)	6	2	6	6	0	1	**6**	**6**	4	5	**6**	7
(7,5)	7	6	5	5	0	1	**5**	**5**	4	5	**5**	7

5.5 Comparative Analysis of the Improved SB Algorithm

To demonstrate efficiency of the improved SB algorithm, we have generated several specialized artificial binary patterns. Apart from the artificial binary patterns, various gray-level images are obtained from the standard image database (SIDBA) developed by the University of Tokyo [**158**]. Some of the images have also been selected from the USC-SIPI image database of University of southern California [**159**] and from Gonzalez and Woods [**127**]. Such selection ensures that images differ in aerial, medical, artificial, natural and textural properties so that the performance of our algorithm for varied applications can be tested.

The artificial images contain specialized patterns (for example, spiral-like, checkerboard-like, honeycomb-like). True gray scale images of size $(150\times150, 256\times256, 300\times300, \textit{and}\ 512\times512)$ are selected for the test without any preprocessing. These gray-level images are transformed into binary images by using Otsu's unsupervised automatic threshold selection method [**49**].

These specialized artificial binary patterns and the standard images are applied to the Stefano-Bulgarelli (SB) algorithm [**76**] and to the improved SB algorithm, where, both algorithms are implemented in the C language. The results with specialized artificial binary patterns and that with standard images are separately discussed below.

5.5.1 Results for Specialized Artificial Binary Test Patterns

Out of the several binary test patterns, six test patterns are selected for algorithm verification. These patterns are shown in Fig. 5.7 and Fig. 5.8. The six cases associated with each artificial binary pattern are shown in Table 5.3. The results in each case are discussed below.

Table 5.3: Comparison Between Different Labels Assigned and the Number of Connected Components (#CC) Detected for Artificial Binary Test Patterns.

	SB Algorithm [76]		**Improved SB Algorithm**	
Case	**Labels**	**#CC**	**Labels**	**#CC**
1	5, 6	2	6	1
2	1, 3, 5	3	5	1
3	1, 3	2	3	1
4	3, 4, 5	3	5	1
5	1, 5, 7	3	7	1
6	1, 2, 4	3	4	1

Case 1: In Fig. 5.7(a), the SB algorithm identifies two connected components (labels 5 and 6 in Fig. 5.7(b)) while the improved SB algorithm identifies only one connected component (labels 6 in Fig. 5.7 (c)).

Case 2: In Fig. 5.7(d), the SB algorithm identifies three connected components (labels 1, 3 and 5 in Fig. 5.7(e)) while the improved SB algorithm identifies only one connected component (label 5 in Fig. 5.7 (f)).

Case 3: In Fig. 5.7(g), the SB algorithm identifies two connected components (labels 1 and 3 in Fig. 5.7(h)) while the improved SB algorithm identifies only one connected component (label 3 in Fig. 5.7(i)).

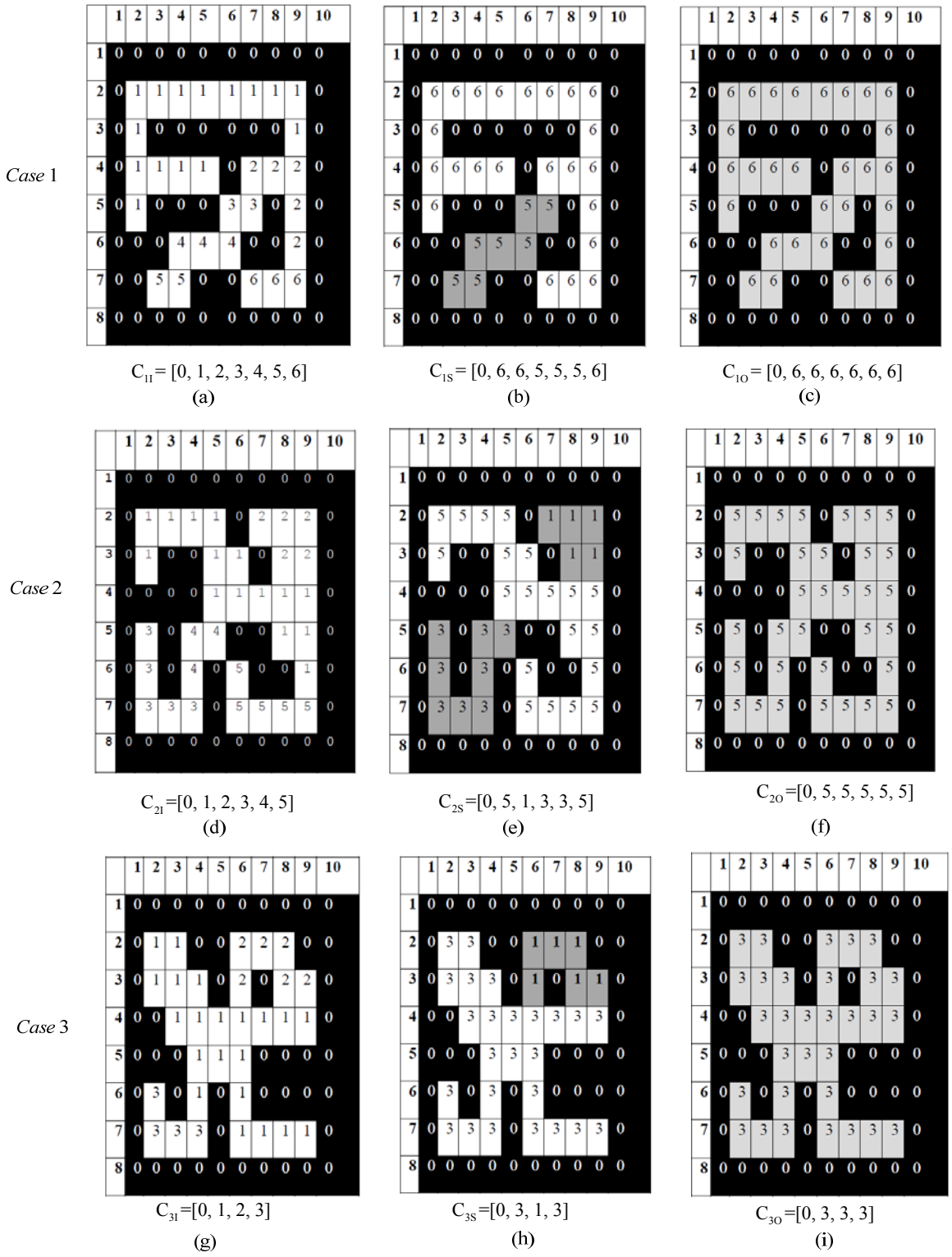


Fig. 5.7: Number of connected components (#CC) and equivalence class for different artificial binary test patterns in Stefano-Bulgarelli's (SB) and in improved SB algorithm (a) First artificial binary test pattern (b) #CC identified by SB algorithm (c) #CC identified by improved SB algorithm (d) Second artificial binary test pattern (e) #CC identified by SB algorithm (f) #CC identified by improved SB algorithm (g) Third artificial binary test pattern (h) #CC identified by SB algorithm (i) #CC identified by improved SB algorithm.

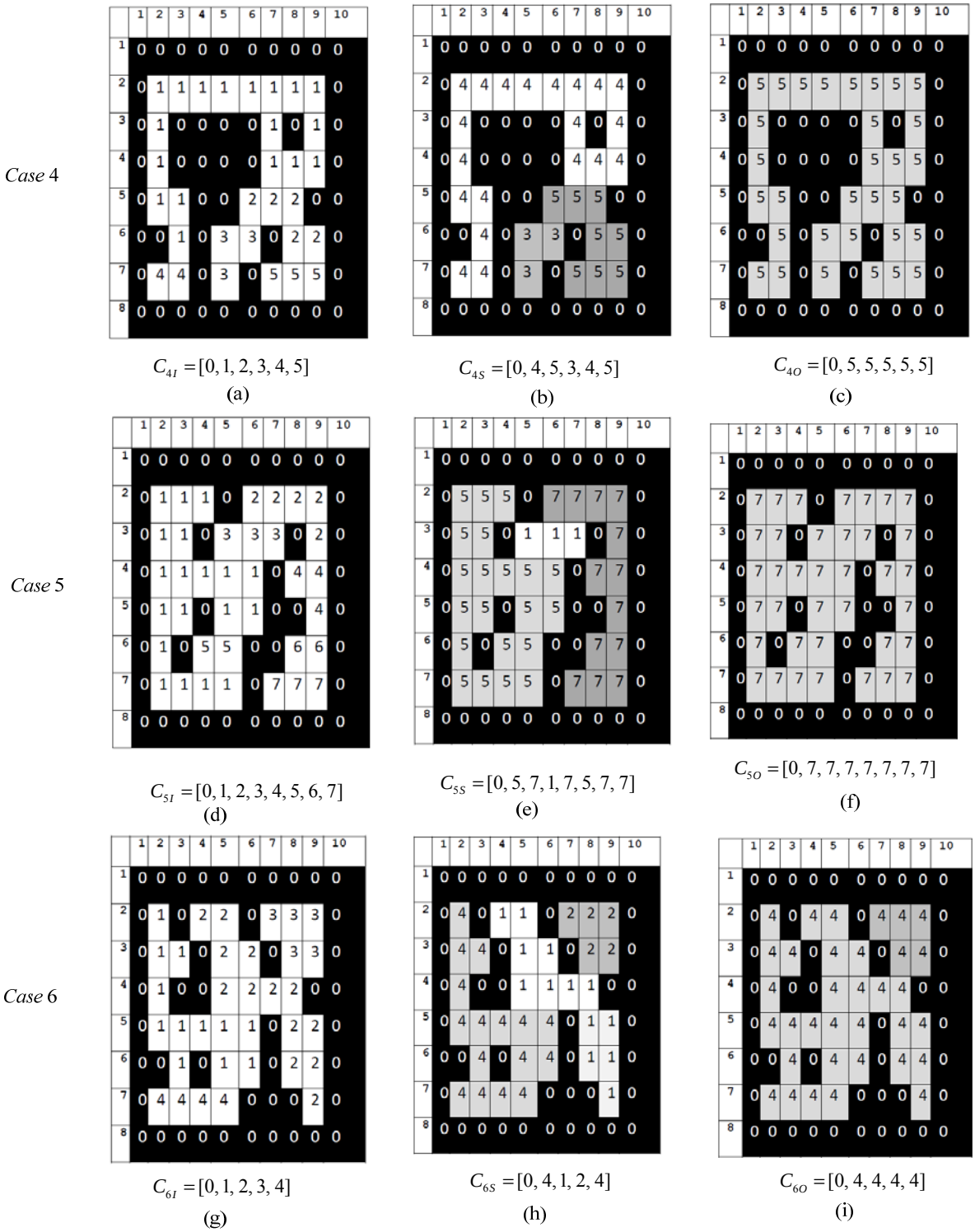


Fig. 5.8: Number of connected components (#CC) and equivalence class for different artificial binary patterns in Stefano-Bulgarelli's (SB) and in the improved SB algorithm (a) Fourth artificial binary test pattern (b) #CC identified by SB algorithm (c) #CC identified by improved SB algorithm (d) Fifth artificial binary test pattern (e) #CC identified by SB algorithm (f) #CC identified by improved SB algorithm (g) Sixth artificial binary test pattern (h) #CC identified by SB algorithm (i) #CC identified by improved SB algorithm.

Case 4: In Fig. 5.8(a), the SB algorithm identifies three connected components (labels 3, 4 and 5 in Fig. 5.8(b)) while the improved SB algorithm identifies only one connected component (label 5 in Fig. 5.8(c)).

Case 5: In Fig. 5.8(d) the SB algorithm identifies three connected components (labels 1, 5 and 7 in Fig. 5.8(e)) while the improved SB algorithm identifies only one connected component (label 7 in Fig. 5.8(f)).

Case 6: In Fig. 5.8(g) the SB algorithm identifies three connected components (labels 1, 2 and 4 in Fig. 5.8(h)) while the improved SB algorithm identifies only one connected component (label 4 in Fig. 5.8(i)).

5.5.2 Results for Standard Images

The number of connected components identified by the improved SB algorithm is also compared with the SB algorithm using several standard images. The selected images for performance evaluation are: 5873_1g, 5882_1g, 5888_1g, 5888_1r, beans and CHEST_X-RAY each of size 150×150 from the standard database of SIDBA [**158**]. The next set of images used for comparison are: Fig0106(c)(cygnusloop-gamma), Fig0107(e)(cygnusloop-Xray), Fig0118(b)(crabpulsar-xray), Fig0118(c)(crabpulsar-optical), Fig0118(d)(crabpulsar-infrared) and Fig0222(a)(face) of size 300×300 are selected from the database of Gonzalez and Woods [**127**]. In addition, we have selected the images: 5.1.10, 5.1.13, 5.1.09, 6.2.09, 5.1.12, 6.2.11 each having size of 256×256and boat.512, 7.1.10, numbers.512, 7.1.02, 7.1.07 and elaine.512 of size 512×512 from USC-SIPI image database of the University of Southern California [**159**].

The description of the comparison is summarized below. The number of connected components (*#CC*) detected by the SB algorithm and by the improved SB algorithm is listed in Table 5.4.

Table 5.4: Comparison between the Numbers of Connected Components (#CC) Identified by the SB Algorithm and by the Improved SB Algorithm for Standard Images.

Source	Image Size	Image	#CC (SB)	#CC (Improved SB)
SIDBA [**158**]	150×150	5873_1g.jpg	173	128
		5882_1g.jpg	213	168
		5888_1g.jpg	409	364
		5888_1r.jpg	459	408
		beans.jpg	88	56
		CHEST_X-RAY.jpg	367	313
USC-SIPI [**159**]	256×256	5.1.10.tiff	622	517
		5.1.13.tiff	15	4
		5.1.09.tiff	741	646
		6.2.09.tiff	337	231
		5.1.12.tiff	129	109
		6.2.11.tiff	295	202
Gonzalez and Woods [**127**]	300×300	Fig0106(c)(cygnusloop-gamma).tiff	597	563
		Fig0107(e)(cygnusloop-Xray).tiff	2757	2698
		Fig0118(b)(crabpulsar-xray).tiff	6844	6686
		Fig0118(c)(crabpulsar-optical).tiff	245	227
		Fig0118(d)(crabpulsar-infrared).tiff	27	14
		Fig0222(a)(face).tif	148	110
USC-SIPI [**159**]	512×512	boat.512.tiff	392	251
		7.1.10.tiff	1092	763
		numbers.512.tiff	2171	1908
		7.1.02.tiff	106	69
		7.1.07.tiff	2580	1855
		elaine.512.tiff	2826	2581

In Table 5.5, the conflicts handled (#CH) by these two algorithms are tabulated. It can be observed from the above table that in comparison to the SB algorithm, the improved SB

algorithm has lesser number of conflicts. It provides accurate number of conflicts as the equivalent classes are already resolved. This results in lesser and accurate number of connected components as expected and observed from Table 5.5.

Table 5.5: Comparison Between Numbers of Conflicts Handled (#CH) by the SB Algorithm and by the Improved SB Algorithm with Standard Images.

Source	**Image Size**	**Image**	**#CH (SB)**	**#CH (Improved SB)**
SIDBA [**158**]	150×150	5873_1g.jpg	259	241
		5882_1g.jpg	274	251
		5888_1g.jpg	395	388
		5888_1r.jpg	386	376
		beans.jpg	396	390
		CHEST_X-RAY.jpg	375	343
USC-SIPI [**159**]	256×256	5.1.10.tiff	905	882
		5.1.13.tiff	97	82
		5.1.09.tiff	596	528
		6.2.09.tiff	1120	1006
		5.1.12.tiff	211	207
		6.2.11.tiff	1098	982
Gonzalez and Woods [**127**]	300×300	Fig0106(c)(cygnusloop-gamma).tif	333	313
		Fig0107(e)(cygnusloop-Xray).tif	553	537
		Fig0118(b)(crabpulsar-xray).tif	1837	1624
		Fig0118(c)(crabpulsar-optical).tif	232	226
		Fig0118(d)(crabpulsar-infrared).tif	152	146
		Fig0222(a)(face).tif	413	401
USC-SIPI [**159**]	512×512	boat.512.tiff	1146	1045
		7.1.10.tiff	2435	2114
		numbers.512.tiff	2545	2268
		7.1.02.tiff	447	431
		7.1.07.tiff	4636	3861
		elaine.512.tiff	1845	1745

5.6 Embedded PowerPC Implementation of the Improved SB Algorithm

The system arrangement has been made for running the connected component analysis algorithm on the PowerPC processor for the embedded environment is shown in Fig. 5.9. The required hardware and software configurations are developed in Xilinx Platform Studio (XPS). Subsequently, the bit stream and the Block RAM memory map (BMM) files are exported to Xilinx Software Development Kit (SDK) for the required software configuration.

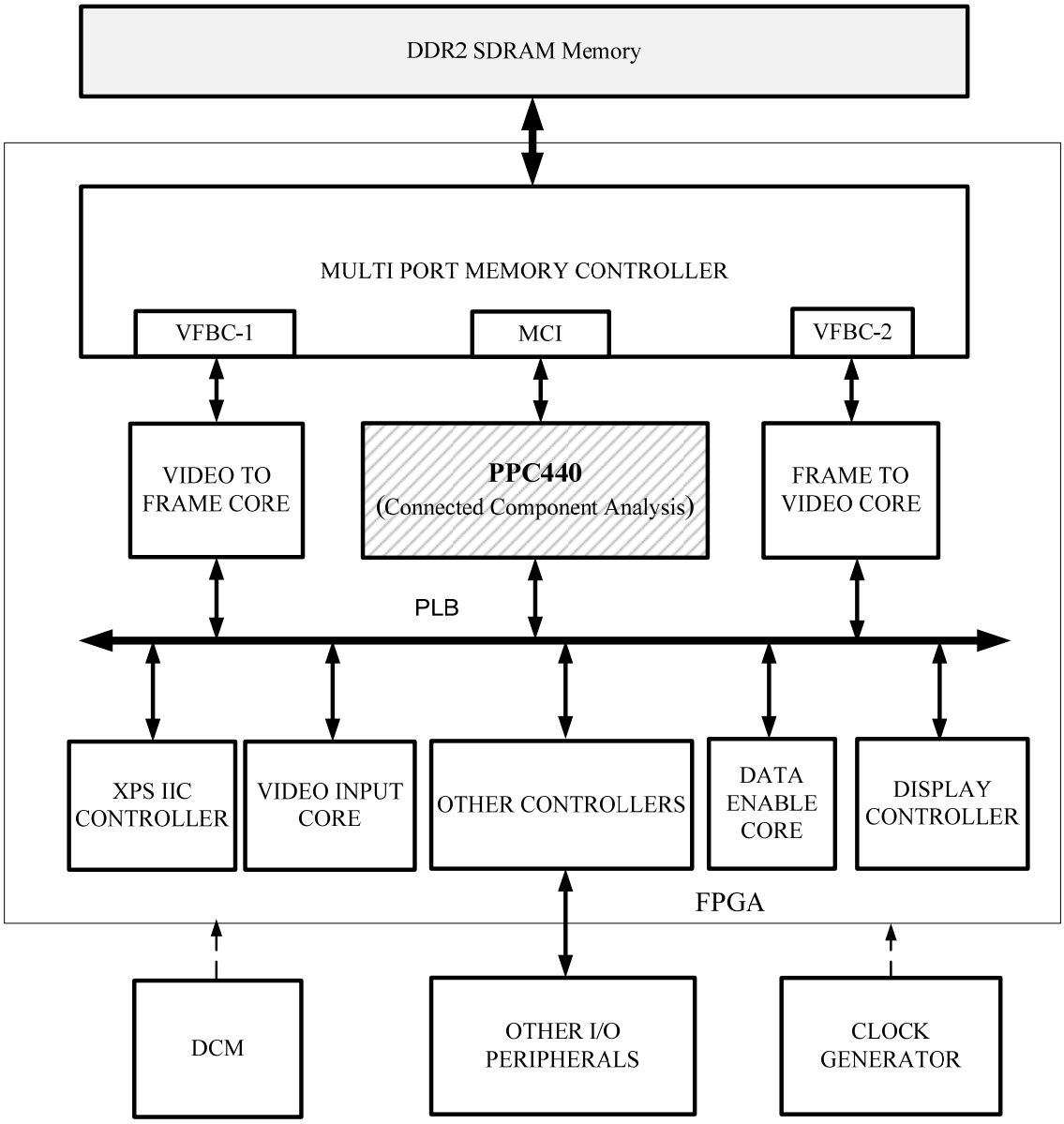


Fig. 5.9: PowerPC running the connected component analysis in an embedded environment.

The C program for connected component labeling algorithm is developed and kept either in DDR2 memory or in Block RAMs (BRAMs). We have chosen the BRAM option for faster processing and execution. A BRAM size of 16 KB suffices for executing the program. In the SDK environment, the linker script generator produces the executable and linking format file (ELF). The generated linker script is shown in Fig. 5.10.

The generated ELF, BMM and the bit stream files are used to configure the Xilinx Virtex-5 xc5vfx70t FPGA device and the embedded PowerPC 440 processor. The processor uses a standalone operating system (version 3.06a), which is a low-level software-layer, that provides access to the basic processor features which can be selected by configuring the board support package shown in Fig. 5.11.

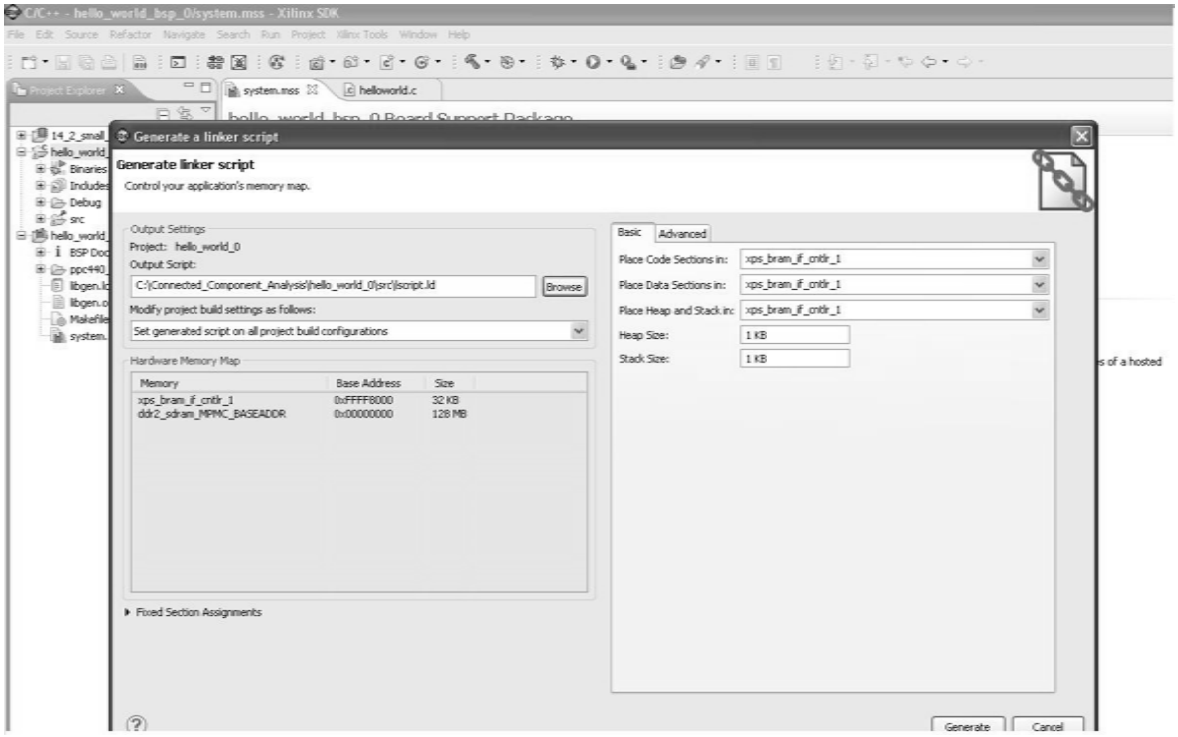


Fig. 5.10: Generated linker script for the connected component analysis algorithm.

We have used the powerpc-eabi-gcc GNU GCC compiler to compile the program. The hyper-terminal snapshot of the execution of connected component analysis program in PowerPC 440 processor is shown in Fig. 5.12. Here, the PowerPC processor runs at 125

MHz for the 640×480 sized image. The processing time required for executing the programme is found to be 0.1 ms.

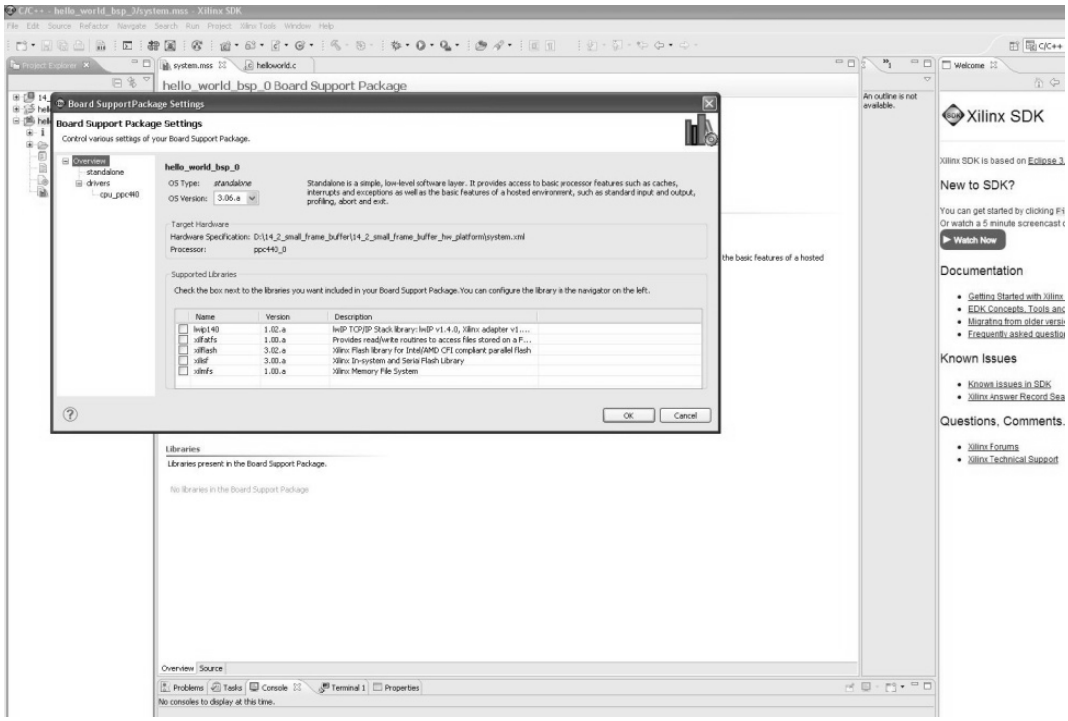


Fig. 5.11: Board support package settings for the connected component analysis algorithm.

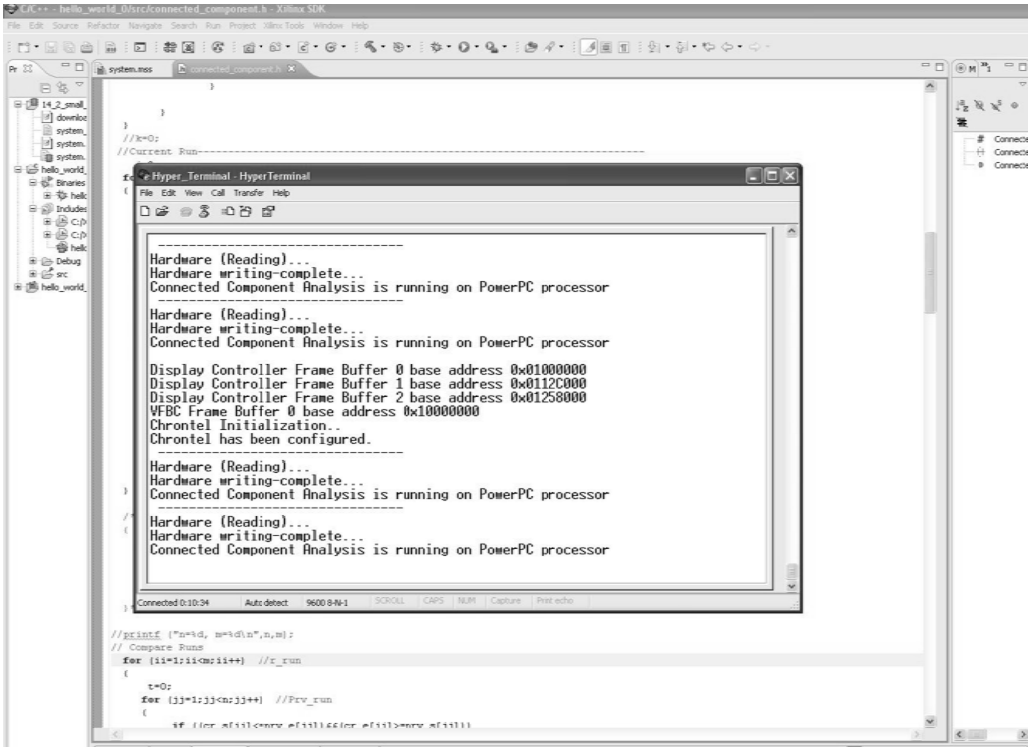


Fig. 5.12: Execution of connected component analysis program on PowerPC 440 processor.

5.7 Conclusion

An improved label-equivalence based connected component labeling algorithms has been presented. The algorithm resolves any label-equivalences in the first scan itself, as soon as they are found. The label-equivalence process is independent from the different temporary labels assigned. The presented algorithm improves the SB algorithm by modifying the equivalence handling procedure, which removes the partial merging problem. Thus, the algorithm eliminates the component disintegration in cases when expanding component runs across a new equivalence, which involves a label other than the maximum of expanding label set. This makes the improved SB algorithm efficient and provides correct count of connected components.

To show the experimental results, we have presented C-code for the 4-connectivity case. However, the improved SB algorithm is independent of *n*-connectivity and works well in the case of 8-connectivity too. The presented algorithm is tested using a variety of artificial test patterns and random standard images. The results demonstrate that the improved algorithm, is simple, manages equivalences efficiently, suffers a lesser number of label conflicts and gives correct count of connected components. The algorithm is simple in principle and easy to implement in C/MATLAB. The C implementation of the improved connected component algorithm runs efficiently on PowerPC 440 processor available as an embedded processor in the Xilinx Virtex-5 xc5vfx70t FPGA device. The PowerPC implementation of connected component analysis is primarily developed for implementing embedded systems for automated object tracking application. The connected component analysis algorithm can also be used as module for other image and video processing applications. The systemic arrangement for running the connected component software module on embedded PowerPC processor along with other image read and display modules is shown in Fig. 5.9.

CHAPTER 6

EMBEDDED IMPLEMENTATION OF KERNEL-BASED MEAN SHIFT OBJECT TRACKING ALGORITHM

6.1 Introduction

Object tracking can be defined as the problem of estimating the trajectory of an object in the image plane as it moves around a scene. Object tracking has a wide range of image and video processing applications such as automated vehicle tracking [**78**], target localization in unmanned air vehicles [**79**], augmented reality [**160**], face tracking [**80**], identity verification [**161**] and many more [**82**,**8**,**5**]. A large number of object tracking methods exist, which primarily depend upon the object attributes such as, representation, features, motion, appearance, shape and the environment in which the tracking is performed. The classification of different types of object tracking approaches is shown in Fig. 6.1.

As shown in the figure, the object tracking method mainly fall in three groups, namely point tracking, kernel tracking and silhouette tracking [**82**]. The point tracking methods are mainly suitable for very small objects. These tracking methods can be deterministic or statistical. In kernel tracking method, the kernel refers to the object shape and appearance [**27**]. For example, the kernel can be rectangular template or an elliptical shape with an associated histogram. Objects are tracked by computing the motion of the kernel in consecutive frames. The motion computation involves identifying the associated parametric transformations such as translation, rotation and affine. The kernel-based method is further divided into two categories one is based on multi-view approach and the other one is template based. The two sub-categories of multi-view are view subspace and classifier based. In silhouette tracking, the silhouette represents the object, which is a region inside the contour of

the object [**162**]. Here, the contour representation defines the boundary of the object [**163**]. The two broad sub-categories of silhouette tracking methods are shape matching and contour tracking. The contour tracking can be state-space based or direct minimization based, which is further categorized in variational and heuristic forms [**82**].

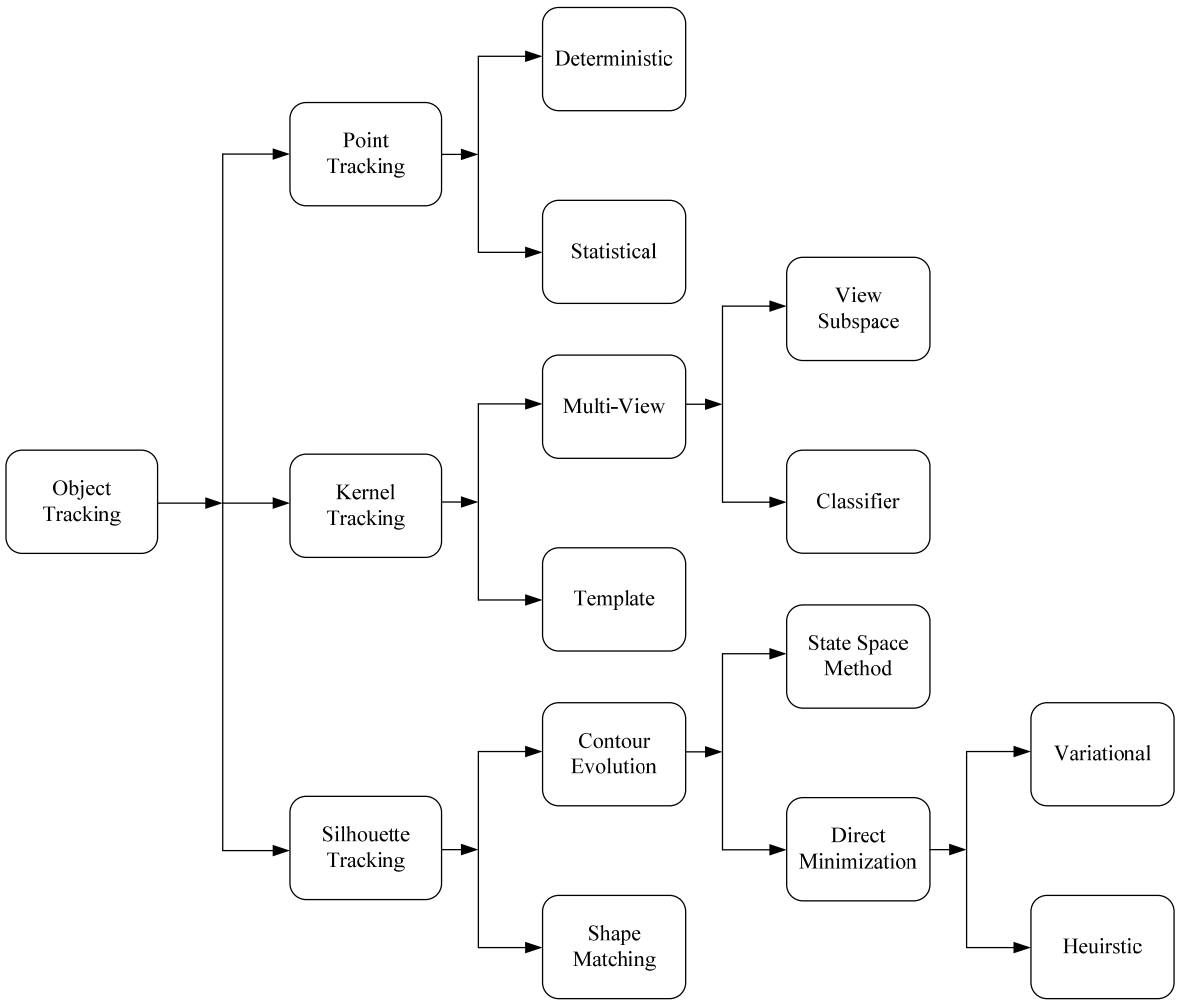


Fig. 6.1: Classification of object tracking methods.

Based on the object attributes, the object tracking operation can, further, be divided into a series of steps, such as object representation, feature selection, and object detection [**27**,**82**]. An object can be represented by its shape and appearance. Commonly used object-shape representations include, point or a set of points, primitive geometric shapes, object silhouette and contour, and articulated shape models [**82**]. Similarly, commonly used appearance

features of the objects are probability densities of object appearance, templates, active appearance models, and multi-view appearance models. Feature selection is the second most vital part of object tracking [**164**]. Some of the widely used visual features are color, edge, optical flow, and texture. Undoubtedly, most important aspect of object tracking is object detection [**82**]. The object detection is required in every frame or when the object first appears in the scene. Some of the commonly used object detection methods are point detectors, segmentation, background modeling, and supervised classifiers [**27**,**82**]. Depending upon the object shape and its tracking environment, several effective techniques such as particle filter [**165**,**166**], optical flow [**167**,**168**], continuously adaptive mean shift (camshaft) [**169**,**170**], and mean shift [**5**,**27**] are widely used in the image and video processing applications.

In real-time system implementations, only a small percentage of total system resources should generally be utilized for the tracking part, so that the rest can be used for other compute-intensive application-specific tasks. Therefore, it is desirable to keep the computational complexity of the tracker minimal. In this context, mean shift algorithm is a popular algorithm for real-time object tracking. It needs the definition of a similarity function to measure the distance between histograms of the target object and the target candidates. The Bhattacharyya distance is a popular distance measure, which can be used for the measurement of distances between two probability distributions (histograms). In the kernel-based object-tracking algorithm, the similarity between the target candidates and the target model is measured with the help of Bhattacharyya distance [**27**,**171**].

The implementation of kernel-based mean shift algorithm using a soft processor is reported in [**83**,**84**]. In this approach, the Xilinx MicroBlaze soft processor is used to run the algorithm. The soft processor along with the other logic resources is synthesized in the Xilinx xc3s500e FPGA device. The implementation uses the Xilinx Spartan-3E FPGA platform

[**83**]. The FPGA device utilization summary shows that with a 320×256 size image frame, almost 84% of the total available slices, 70% of total available BRAMs and 50% of the arithmetic computational blocks get utilized. A different approach, which mainly uses FPGA BRAMs to implement mean shift filter is given in [**172**]. It uses the Xilinx xc4vlx160 FPGA device mounted on the Celoxica RC2000-4 FPGA platform. In the implementation, the image is stored in the FPGA Block RAMs. It is reported in [**172**] that to implement a filter size of 31×31, the circuit needs 376 BRAMs, which is larger than the available 288 BRAMs in the device [**172**]. A similar kind of implementation for a mean shift based image segmentation application is given in [**173**].

For our work, we have selected an FPGA based platform, namely the Xilinx ML-507, for efficient real-time implementation of object tracking. With the availability of an embedded processor in the FPGA device we can completely do away with the uses of the soft processor as utilized by [**83**] and [**84**]. As discussed in [**83**], the soft processor itself takes 60% of the total available FPGA slices which is a substantial portion of the FPGA resources. Similarly, the need of capturing the image in BRAMs as proposed in [**172**,**173**] can be completely done away with by considering other memory resources, such as DDR2. In spite of the availability of large BRAM resources, a real-time image can be buffered in the available off-chip memory (DDR2). The available BRAM resources can then be utilized for other application-related operations requiring time-critical computations with large throughput.

In our work, the consecutive video frames with a resolution of 640×480 are first buffered in the DDR2 SDRAM memory, which uses the frame acquisition module described in Chapter 2. Subsequently, an embedded implementation of kernel-based mean shift (KBMS) algorithm is done in the Xilinx Virtex-5 FPGA device available on the Xilinx ML-507 platform. The available embedded PowerPC processor provides the necessary controls and

manages the various peripherals and IPs, as required. The time consuming tasks such as, computation of complex arithmetic functions and performing frequent iterative operations are accelerated by hardware realizations. The KBMS algorithm requires the design of various building blocks including those required for the computation of kernel-weight, kernel-smoothed local histogram, computation of distance measure (using Bhattacharyya coefficient), mean shift weight and new location for the mean shift. This chapter proposes efficient architectures for the above building blocks, which are used to implement the KBMS algorithm.

In this chapter, an embedded design approach for implementing the KBMS algorithm is presented. Before implementing the algorithm through different hardware and software blocks, the KBMS algorithm is realized in the MATLAB programming language. It uses the MATLAB in-built functions for image/video read, image/video display along with some of the available arithmetic functions. Further, in order to explore algorithmic level transformations and tradeoffs necessary for mapping the algorithm on to hardware, a C implementation is developed. In order to get improved speed, analysis of time critical functions is performed and suitable data types are selected for the intended performance gain.

The compute-intensive and time-consuming operations are identified for hardware realizations where as simple data movement and control operations are marked for handling by the processor. The embedded implementation of the KBMS algorithm is done on the Xilinx ML-507, a Virtex-5 FX FPGA based platform. The embedded PowerPC 440 processor available in the FPGA device is used for implementing the software tasks, and the hardware blocks are realized with the FPGA using the FPGA fabric, the BRAMs, and the DSP slices. The datapath uses the fixed-point arithmetic, which offers reasonably good performance with reduced hardware consumption. Furthermore, to simplify the complex arithmetic function into simple addition/subtraction and shift operations, the concepts of logarithmic number

system (LNS) is utilized, as presented in Chapter 3. The implementation also uses the image acquisition module described in the Chapter 2. The frame acquisition module provides a 640×480 resolution video frame containing the object.

Following paragraph describes the organization of the rest of this chapter. The kernel-based mean shift algorithm and its related constituent units are discussed in Section 6.2. In Section 6.3, the kernel-based mean shift (KBMS) algorithm flow is described. Section 6.4 presents the MATLAB/C implementation results. In Section 6.5 the embedded implementation of the kernel-based mean shift algorithm is illustrated along with its constituent building blocks. An architecture for kernel-smoothed local histogram has been proposed in the Section 6.6, which is used in target and the candidate modeling. Section 6.7 proposes an architecture for computing the Bhattacharyya coefficient. In Section 6.8, we give an architecture for computing the mean shift weights. Section 6.9 gives architecture for new mean shift location computational unit. Integration of various architectural modules is described in Section 6.10. The overall control mechanism is described in Section 6.11. FPGA implementation results are provided in Section 6.12. In Section 6.13, a complete system view of the design is shown and Section 6.14 concludes the chapter.

6.2 Kernel-based Mean Shift (KBMS) Object Tracking

An object tracker typically consists of two components, which are combined together depending on the tracking needs of specific application. The first component, namely, target representation and localization deals with the changes in the appearance of the target, and is a bottom-up process. Filtering and data association is the other component, which is mostly a top-down process and deals with the dynamics of the tracked object, learning the scene priors and evaluation of the different hypotheses. The formulation of filtering and data association process is through the state-space approach for modeling discrete-time dynamic systems [**27**].

The tracking application developed in this chapter relies on the target representation and localization and is based on kernel-based mean shift approach. The basic premise of this tracking method is, that only a small change takes place in the location and appearance of the target in two consecutive frames. Thus, the localization can be achieved by maximizing a likelihood type function. First, the target is spatially masked with an isotropic kernel and a smooth similarity function is defined next. Similarity between the target model and the target candidates in the next frame is measured using a similarity metric. Thus, the target localization problem is reduced to finding the maximum of the similarity metric. This method of target representation and localization can be integrated with various motion filters and data association techniques. The mean shift clustering is described below which is followed by the various steps of the kernel-based mean shift (KBMS) object tracking algorithm.

6.2.1 Mean Shift Clustering

Mean shift is a non-parametric density estimation technique used for various low-level vision tasks [**5**,**171**]. The mean shift clustering algorithm starts with the initialization of a large number of hypothesized cluster centers randomly chosen from the large data set [**174**]. Each cluster center is moved to the center of gravity (COG) lying inside within its region-of-interest (ROI). The vector that is defined by the old and the new cluster centers is called the mean shift vector (MSV). The MSV is computed iteratively until the cluster centers do not change their positions. A pictorial representation of mean shift clustering is shown in Fig. 6.2. Here, a set of random data has been shown. In the first cluster the COG is at location *x*. The algorithm checks for the new clusters. In the new cluster, the COG is at location *y*. As apparent from the figure, the center of gravity is shifted from the location *x* to *y*. The MSV is the distance between *x* and *y* vectors. Similarly, when the center of gravity is shifted at location *z*, the MSV moves ahead.

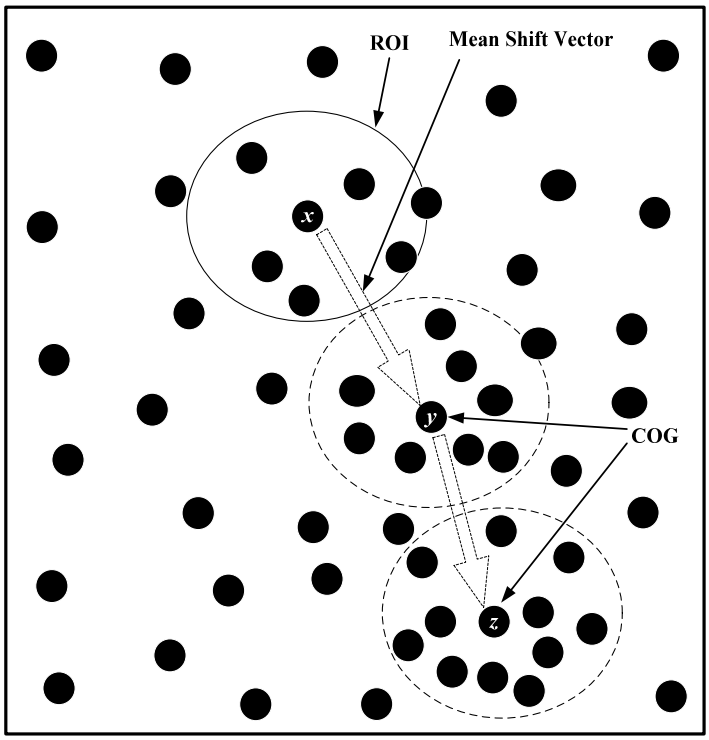


Fig. 6.2: Pictorial representation of mean shift clustering.

Kernel-based object tracking utilizes the principle of mean shift clustering approach [**5**]. In this approach, color is used as the visual feature to from an appearance model [**5**]. To satisfy the low computational cost requirement imposed by real-time processing, the *m*-bin discrete density histogram as suggested in [**5**] is used. Probabilistic distribution for the target in the target frame is compared with the probabilistic distribution of the target to be tracked (also known as candidate model or candidate target) in the consecutive frames. The flow of the KBMS algorithm [**5**], is explained below.

6.2.2 Target Representation

Color as the feature space is selected to characterize the target. The reference target model is represented by its probability density function (pdf), q in R, G, B color. The target model is considered as centered at the spatial location zero. In the subsequent frames, a target candidate is defined at location $\mathbf{y}$, and is characterized by the pdf, $p(\mathbf{y})$. Both pdf-s are estimated from the image data [**5**,**27**]. The target model is defined by

$$\hat{\mathbf{q}}=\{\hat{q}_u\}_{u=1\ldots m} \tag{6.1}$$

where, $\sum_{u=1}^{m}\hat{q}_u=1$.

Similarly, the target candidate is defined as

$$\hat{\mathbf{p}}(\mathbf{y})=\{\hat{p}_u(\mathbf{y})\}_{u=1\ldots m} \tag{6.2}$$

where, $\sum_{u=1}^{m}\hat{p}_u=1$.

Here, the histograms defined in (6.1) and (6.2) are the non-parametric density estimates of the target model and the target candidates in the *m*-bin reduced color feature space. The similarity function between the histogram pdf-s, $\hat{\mathbf{p}}(\mathbf{y})$ and $\hat{\mathbf{q}}$ is denoted by,

$$\hat{\rho}(\mathbf{y})\equiv\rho[\hat{\mathbf{p}}(\mathbf{y}),\hat{\mathbf{q}}] \tag{6.3}$$

This function plays the role of a likelihood function and its local maximum in the image points to the presence of the object in the second frame having a representation similar to $\hat{\mathbf{q}}$ (6.1) defined in the first frame. The similarity function is regularized by masking the objects with an isotropic kernel in the spatial domains [**27**]. When the kernel weights (kw), carrying continuous spatial information, are used in defining the feature space representations, $\hat{\rho}(\mathbf{y})$ becomes a smooth function in **y**.

6.2.3 Target Model

A target is represented by an elliptical/circular region in the image. To remove the influence of different target dimensions, all targets are first normalized to a unit circle [**27**]. Let $\{\mathbf{x}_i^*\}_{i=1\ldots n}$ be the normalized pixel locations in the region defined as the target model. The

region is centered at zero. An isotropic kernel, with convex and monotonic decreasing kernel profile, $k(\mathbf{x})$ assigns smaller weights to pixels (at location, $\mathbf{x}$) farther from the center, given by, $k(\mathbf{x}) = k\left(\|\mathbf{x}\|^2\right)$. By using these weights, robustness of the density estimation increases.

The function $b: R^2 \rightarrow \{1,2,3,\ldots,m\}$ associates with the pixel at location $\mathbf{x}_i^*$ the index $b(\mathbf{x}_i^*)$ of its bin in the quantized feature space. The probability of feature $u = 1,2,3,\ldots,m$ in the target model is then computed as:

$$\hat{q}_u = C_1 \sum_{i=1}^{n} k\left(\|\mathbf{x}_i^*\|^2\right) \delta\left[b(\mathbf{x}_i^*) - u\right] \tag{6.4}$$

where δ is the Kronecker delta function. The normalization factor C_1 is derived by imposing the condition $\sum_{u=1}^{m} \hat{q}_u = 1$, since the summation of delta functions for $u = 1,2,3,\ldots,m$ is equal to one. We thus obtain the value of C_1 as:

$$C_1 = \frac{1}{\sum_{i=1}^{n} k\left(\|\mathbf{x}_i^*\|^2\right)} \tag{6.5}$$

6.2.4 Target Candidates

Let the normalized pixel locations of the target candidate, centered at **y** in the current frame be $\{\mathbf{x}_i\}_{i=1..n}$. The probability of feature $u = 1,2,3,\ldots,m$ in the target candidate is given in [**27**] as:

$$\hat{p}_u(\mathbf{y}) = C_2 \sum_{i=1}^{n} k\left(\|\mathbf{y} - \mathbf{x}_i\|^2\right) \delta\left[b(\mathbf{x}_i) - u\right] \tag{6.6}$$

where, $C_2 = \dfrac{1}{\sum_{i=1}^{n} k\left(\|\mathbf{y} - \mathbf{x}_i\|^2\right)}$ (6.7)

is the normalization constant for the pdf of the target candidate. Note that, it is assumed that the size of the target does not change with time, and no adaptation has been incorporated to tackle this in our implementation. This eventually assumes that the target candidate is contained in the same number of pixels in the subsequent target frames. The similarity function in (6.3) inherits the properties of the kernel profile, $k(\mathbf{x})$ as both the target model and target candidate models contain $k(\mathbf{x})$. Thus, a differentiable kernel profile yields a differentiable similarity function and efficient gradient-based optimizations can be employed for finding the maximum of the similarity function, ρ .

6.2.5 Kernel Profile

A convex and monotonic decreasing Epanechnikov kernel profile [**171**] is used for assigning a smaller weight to the locations that are farther from the center of the target and higher weight to the nearby locations. A representative picture of the Epanechnikov kernel is shown in Fig. 6.3.

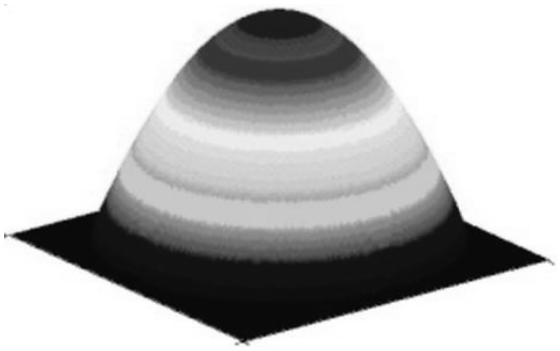


Fig. 6.3: Epanechnikov kernel profile.

The profile is a nonnegative, non-increasing and piecewise continuous function. The Epanechnikov kernel for a scalar *x* is defined as,

$$k(x) = \begin{cases} 1-x & 0 \le x \le 1 \\ 0 & x > 1 \end{cases} \tag{6.8}$$

The above profile can be extended to multiple dimensions and the exact form of the Epanechnikov kernel profile at d-dimensional coordinate **x** is given by [**27**,**171**],

$$k(\mathbf{x}) = \begin{cases} \frac{1}{2} c_d^{-1} (d+2)\left(1 - \|\mathbf{x}\|^2\right) & \text{if } \|\mathbf{x}\| \le 1 \\ 0 & \text{otherwise} \end{cases} \tag{6.9}$$

where, c_d is the volume of the unit *d*-dimensional sphere.

6.2.6 Bhattacharyya Coefficeint based Distance Metric

Bhattacharyya distance is used for the measurement of distances between two smoothed histograms [**175**]. It solves many image/video processing and pattern recognition problems and find numerous image/video processing applications which includes classification, clustering [**171**], distributed frequency comparisons [**176**], and image retrieval [**177**]. We can collect data sets at different times or under different conditions and then by using the similarity measure, the distributional testing can be used to determine whether they are identical or not.

The Bhattacharyya distance between the distributions (6.1) and (6.2) is defined as,

$$d(\mathbf{y}) = \sqrt{1 - \rho\left[\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}\right]} \tag{6.10}$$

where,

$$\rho(\mathbf{y}) \equiv \rho[\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}] = \sum_{u=1}^{m} \sqrt{\hat{p}_u(\mathbf{y}) \hat{q}_u} \tag{6.11}$$

The expression in (6.11) is known as the sample estimate of the Bhattacharyya coefficient and it increases with the decrease of distance between the two histograms.

6.2.7 Distance Minimization and the Mean Shift Weight

Minimization of Bhattacharyya distance (6.10) necessitates the maximization of Bhattacharyya coefficient (6.11) [**27**]. Identification of the new target position in the current frame begins at position $\hat{\mathbf{y}}_0$ of the target in the previous frame. It requires computing $\{\hat{p}_u(\hat{\mathbf{y}}_0)\}_{u=1..m}$ of the target candidate at location $\hat{\mathbf{y}}_0$ in the current frame. As illustrated in [**27**], it is assumed that the target candidate $\{\hat{p}_u(\mathbf{y})\}_{u=1..m}$ does not change abruptly from the initial $\{\hat{p}_u(\hat{\mathbf{y}}_0)\}_{u=1..m}$, which is often a valid assumption between successive frames. With this assumption, using a Taylor expansion around $\{\hat{p}_u(\hat{\mathbf{y}}_0)\}_{u=1..m}$, the linear approximation of the Bhattacharyya coefficient (6.11) can be approximately obtained as [**27**],

$$\rho[\hat{\mathbf{p}}(\mathbf{y}),\hat{\mathbf{q}}] \approx \frac{1}{2}\sum_{u=1}^{m}\sqrt{\hat{p}_u(\hat{\mathbf{y}}_0)\hat{q}_u} + \frac{C_2}{2}\sum_{i=1}^{n} w_i k\left(\left\|\mathbf{y}-\mathbf{x}_i\right\|^2\right) \tag{6.12}$$

where, $w_i's$ are the weights, defined as,

$$w_i = \sum_{u=1}^{m}\sqrt{\frac{\hat{q}_u}{\hat{p}_u(\hat{\mathbf{y}}_0)}}\delta\left[b(\mathbf{x}_i)-u\right] \tag{6.13}$$

As is evident from (6.12), to minimize the Bhattacharyya distance (6.10), the second term in (6.12) has to be maximized, which is dependent on **y**. This term represents the density estimate computed with the kernel profile $k(\mathbf{x})$ (6.9) at **y** in the present frame. Here, w_i (6.13) weights the data. By using the mean shift procedure [**5**], the kernel is recursively moved from the present location, $\hat{\mathbf{y}}_\mathbf{0}$ to the new location $\hat{\mathbf{y}}_1$. The expression for $\hat{\mathbf{y}}_1$ scan be obtained as per the following relation,

$$\hat{\mathbf{y}}_1 = \frac{\sum_{i=1}^{n}\mathbf{x}_i w_i}{\sum_{i=1}^{n} w_i} \tag{6.14}$$

Expression (6.14) is a form of center of gravity (COG) computation. The center of gravity computation is a widely used operations in the area of image and video processing [**9**]. The concepts of COG is used, in many image and video processing applications. Some of the work that uses COG concept includes, the accurate object localization in gray level images [**178**], feature-based image registration [**179**], augmented reality conferencing system [**180**], and system for landing unmanned aerial vehicle [**181**,**182**].

The center of gravity (COG) computation for the mean shift new location [**27**] is constituted of two parts. The first part computes the center of gravity of the *x*-coordinate, which is the average of the *x*-coordinates of all the pixels. The second part computes the COG of the *y*-coordinate, which is the average of the *y*-coordinates of all the pixels in the image. To arrive at (6.14), the Epanechnikov kernel profile is used that is described in Section 6.2.5 [**5**]. The complete kernel-based mean shift target localization algorithm is explained in the following section.

6.3 The KBMS Tracking Algorithm Flow

The kernel-based object tracking algorithm whose primary goal is to maximize the Bhattacharyya coefficient $\rho[\hat{\mathbf{p}}(\mathbf{y}),\hat{\mathbf{q}}]$ is summarized below [**27**].

The target model $\{\hat{q}_u\}_{u=1\ldots m}$ and its location $\hat{\mathbf{y}}_0$ in the previous frame are known a priori.

Step 1: Initialize the location of the target in the current frame with $\hat{\mathbf{y}}_0$, compute $\{\hat{p}_u(\hat{\mathbf{y}}_0)\}_{u=1\ldots m}$ and evaluate,

$\rho[\hat{\mathbf{p}}(\mathbf{y}_0),\hat{\mathbf{q}}] = \sum_{u=1}^{m}\sqrt{\hat{p}_u(\hat{\mathbf{y}}_0)\hat{q}_u}$ as in (6.11)

Step 2: Derive the mean shift weights $\{w\}_{i=1\ldots n}$ as in (6.13).

Step 3: Find the new location of the target candidate using mean shift iteration (6.14).

Step 4: Compute $\{\hat{p}_u(\hat{\mathbf{y}}_1)\}_{u=1\ldots m}$ and evaluate

$\rho[\hat{\mathbf{p}}(\mathbf{y}_1),\hat{\mathbf{q}}]=\sum_{u=1}^{m}\sqrt{\hat{p}_u(\mathbf{y}_1)\hat{q}_u}$ as in (6.11).

Step 5: While $\rho[\hat{\mathbf{p}}(\hat{\mathbf{y}}_1),\hat{\mathbf{q}}]<\rho[\hat{\mathbf{p}}(\hat{\mathbf{y}}_0),\hat{\mathbf{q}}]$

Do $\hat{\mathbf{y}}_1\leftarrow\frac{1}{2}(\hat{\mathbf{y}}_0+\hat{\mathbf{y}}_1)$,

Evaluate $\rho[\hat{\mathbf{p}}(\hat{\mathbf{y}}_1),\hat{\mathbf{q}}]$

Step 6: If $\|\hat{\mathbf{y}}_1-\hat{\mathbf{y}}_0\|<\varepsilon$ Stop.

Otherwise, set $\hat{\mathbf{y}}_0\leftarrow\hat{\mathbf{y}}_1$ and go to *Step* 2.

To analyze the algorithm in detail, the KBMS algorithm is implemented in MATLAB and in C language, which is described below.

6.4 MATLAB/C Implementation of the KBMS Tracking Algorithm

The KBMS algorithm, as elaborated in Section 6.3 is implemented in MATLAB for understanding the steps in the algorithm and the convergence issues. The MATLAB code is verified with several stored video files. To identify the embedded implementation issues and to formulate the quantitative analysis, the KBMS algorithm is subsequently implemented in C language. All the functions in C language implementation are custom-made except the video read and display. To capture and display the video, the C implementation utilizes the standard OpenCV video read and write functions [**183**].

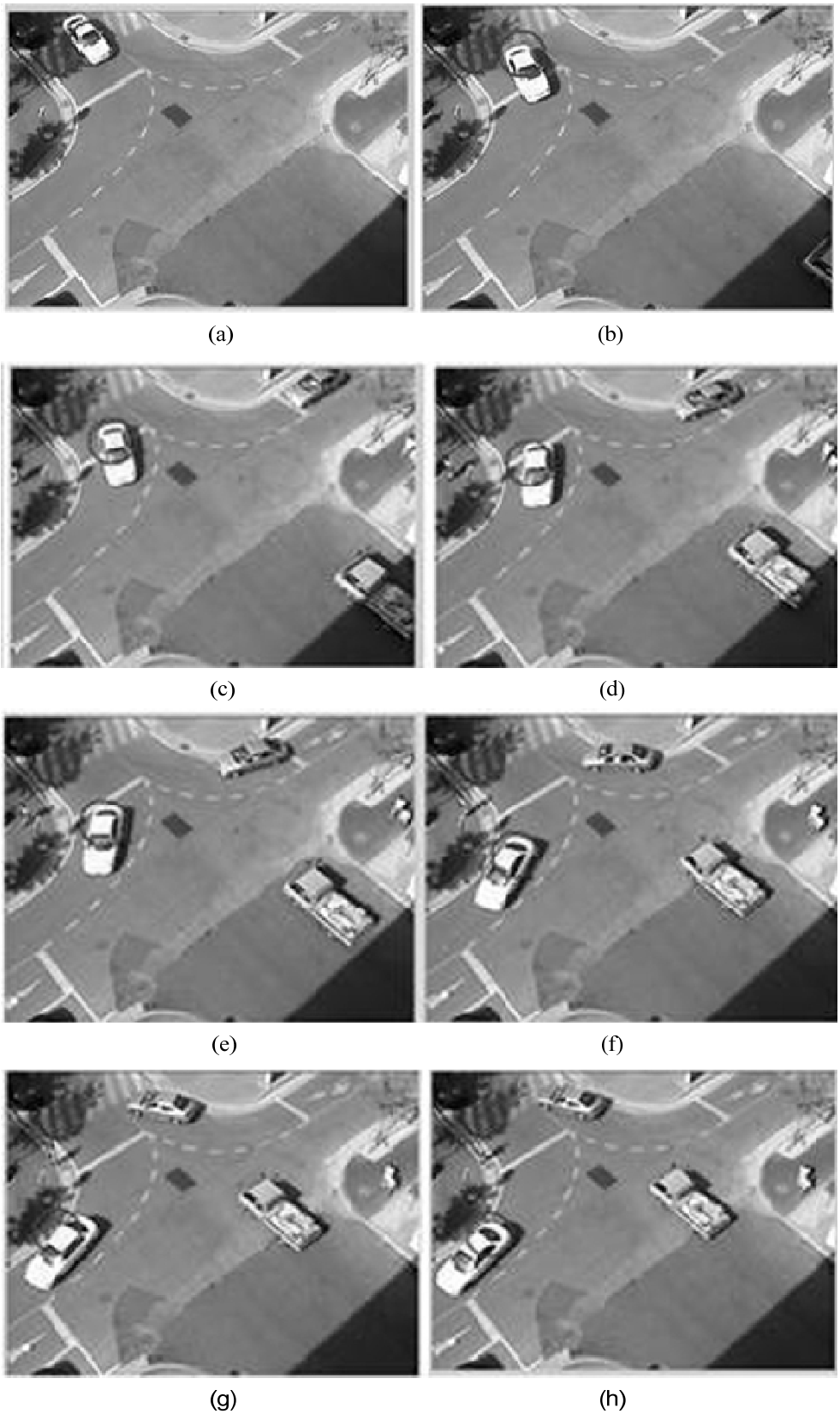


Fig. 6.4: C implementation of the KBMS algorithm. (a) Frame No.=12 (b) Frame No.=25 (c) Frame No.=32 (d) Frame No.=38 (e) Frame No.=42 (f) Frame No.=50 (g) Frame No.=55 (h) Frame No.=57.

While integrating the C code with OpenCV functions we have selectively integrated the video capture and display functions. Later, in the embedded implementation of the complete system, the developed video acquisition and display unit described in Chapter 2 replaces the OpenCV functions.

The C code is compiled using the GCC complier [**184**] in Linux environment and it runs on a normal computer. Fig. 6.4 shows the results of tracking of a car at various frame numbers (Frame Nos. 25, 32, 38, 42, 50, 55 and 57) where the circular target coordinates are specified in Frame No. 12. To determine the time-consuming portion of the program, the code is profiled using the GNU gprof profiler [**185**]. To view the graphical representation of the different functions call, the Valgrind and Callgrind open source software are used [**186**].

In the C implementation, it is observed that the function which compute kernel-smoothed local histogram and the Bhattacharyya coefficient, consume most of the computation time. To achieve better performance these two functions are therefore identified for hardware implementation. In the next section, the embedded implementation of the KBMS tracking algorithm is presented.

6.5 Embedded Implementation of the KBMS Tracking Algorithm

In this section the embedded implementation approach for kernel-based object tracking algorithm is and various building blocks are utilized. The embedded system arrangement for realizing the KBMS algorithm is shown in Fig. 6.5. As depicted in the figure, a 640×480 resolution video frame is captured from an analog camera and is buffered in the DDR2 SDRAM memory available on the Xilinx ML-507 FPGA platform, by the video acquisition unit as described in Chapter 2. The embedded PowerPC processor, available with the Xilinx Virtex-5 FX FPGA, accesses the stored frame. Application software running on the embedded PowerPC processor controls the frame acquisition process.

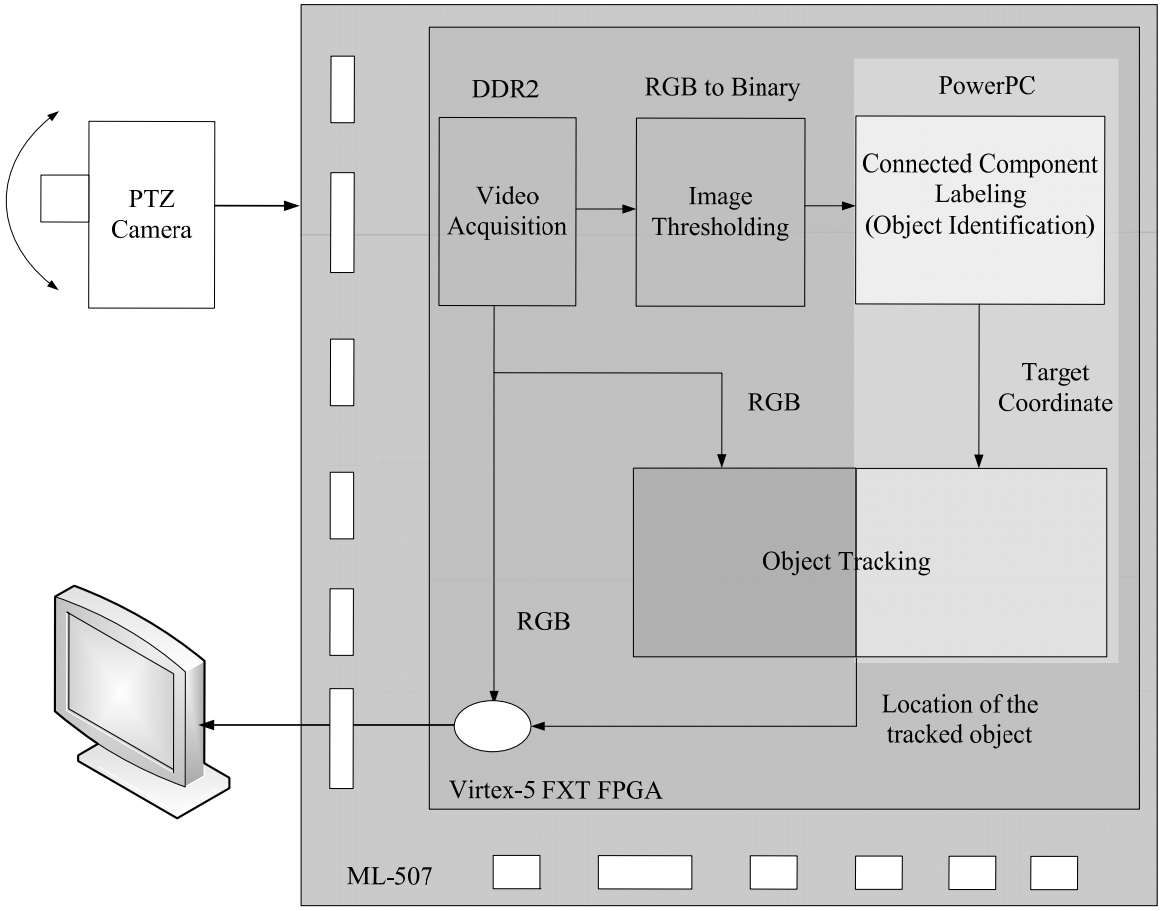


Fig. 6.5: Embedded system arrangement for the mean shift object tracking.

The complete hardware and software arrangement is shown in Fig. 6.6. To design the complex arithmetic elements, the logarithmic and antilogarithmic blocks illustrated in Chapter 3 are used. The image thresholding blocks of Chapter 4 is used by the connected component labeling algorithm to segment out and identify the object, which is covered in Chapter 5. The hardware portion of the object tracking algorithm includes color space quantization, candidate/target modeling, kernel-weight computation, computation of Bhattacharyya coefficient, mean shift weight computation and the computation of new mean shift location. The various hardware architectural units for realizing the KBMS algorithm are shown in Fig. 6.7.

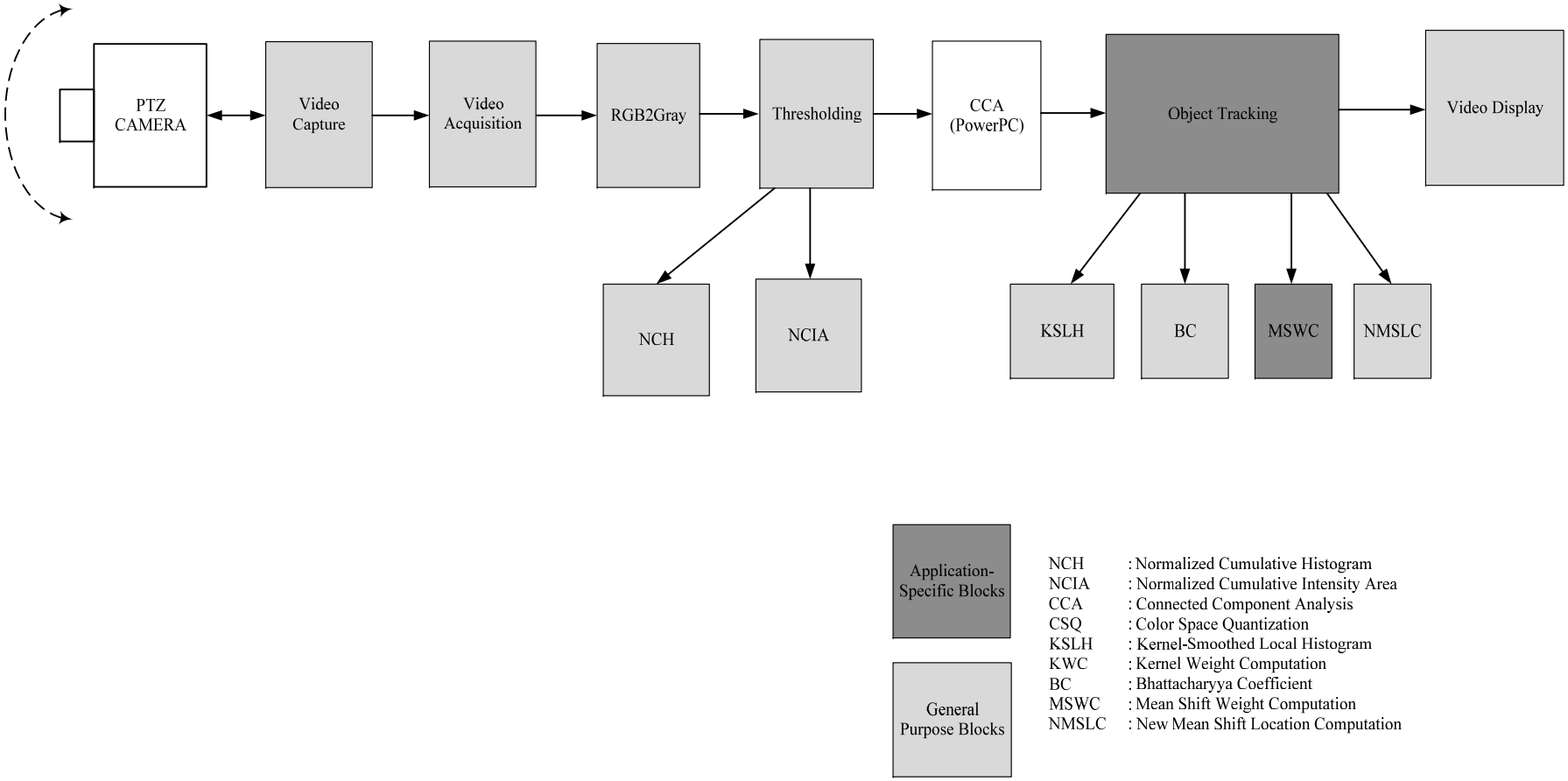


Fig. 6.6: Complete hardware/software arrangement for realizing the object tracking algorithm.

As shown in above figure, the color space quantization is simply a data selection process, which is easily realized in the hardware. The candidate and target modeling needs weighted local histogram (WLH) computation, which is smoothed by the kernel profile. Thus, the candidate and the target modeling hardware need kernel-smoothed local histogram (KSLH) computation. The KSLH unit utilizes the kernel-weight and weighted local histogram computation. The mean shift weight computational unit is a form of center of gravity (COG) computation.

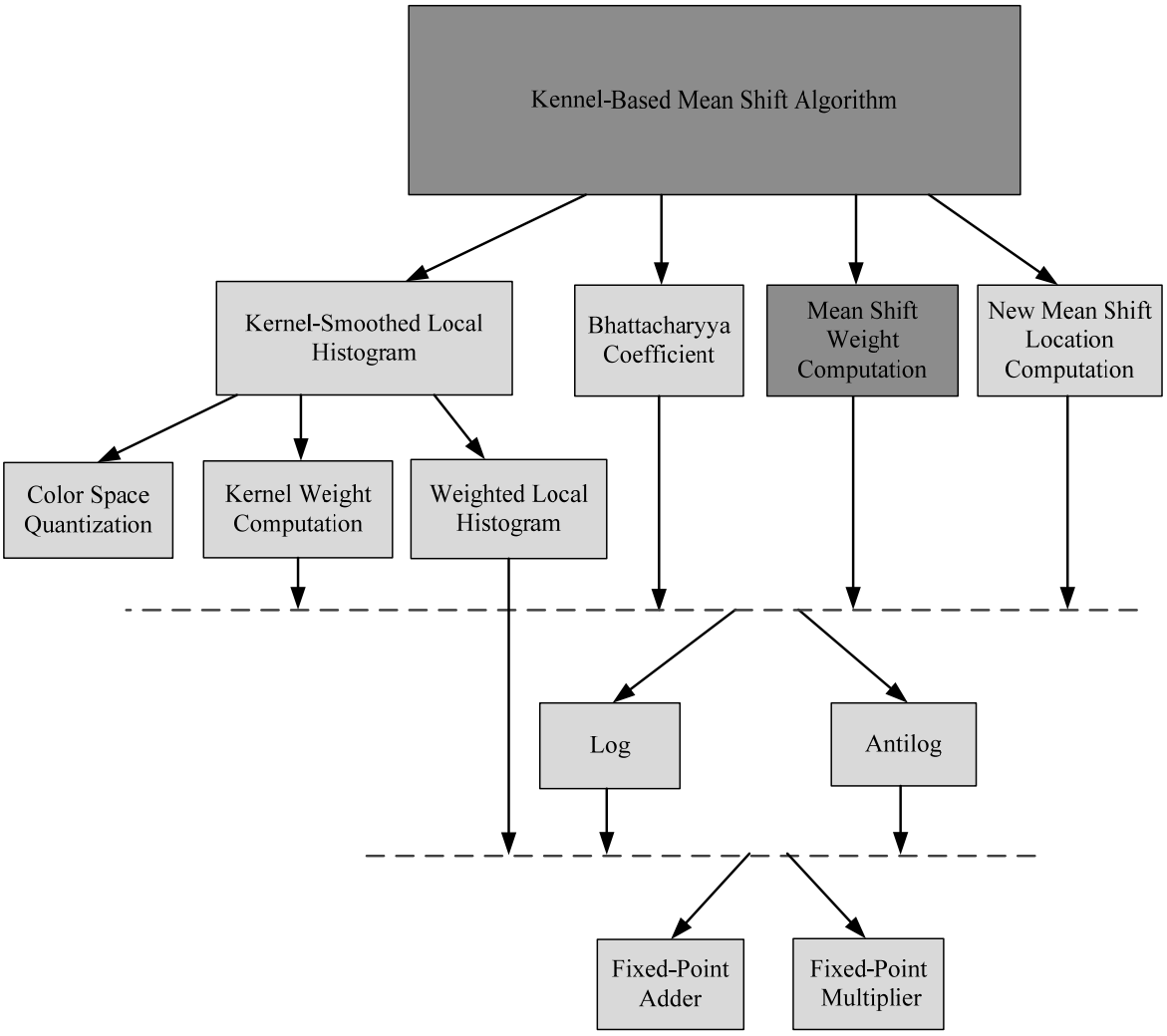


Fig. 6.7: Complete hardware architectural units for KBMS algorithm.

The binary logarithm and antilogarithm units illustrated in the Chapter 3 compute the kernel-weight. The weighted histogram is computed by using BRAMs with fixed-point

multiplier and fixed-point adder. The concept of LNS is also used to compute the Bhattacharyya coefficient, the mean shift weights and to find out the new mean shift location. Following sections describe the detail of the above architectural units.

6.6 Kernel-Smoothed Local Histogram Computation

As illustrated in *Step* 1 of the KBMS algorithmic flow covered in the Section 6.3, the target model (6.4) and the target candidate (6.6) require kernel-smoothed local histogram (KSLH) computation. The basic building blocks of KSLH computation and their architectural arrangement is shown in Fig. 6.8.

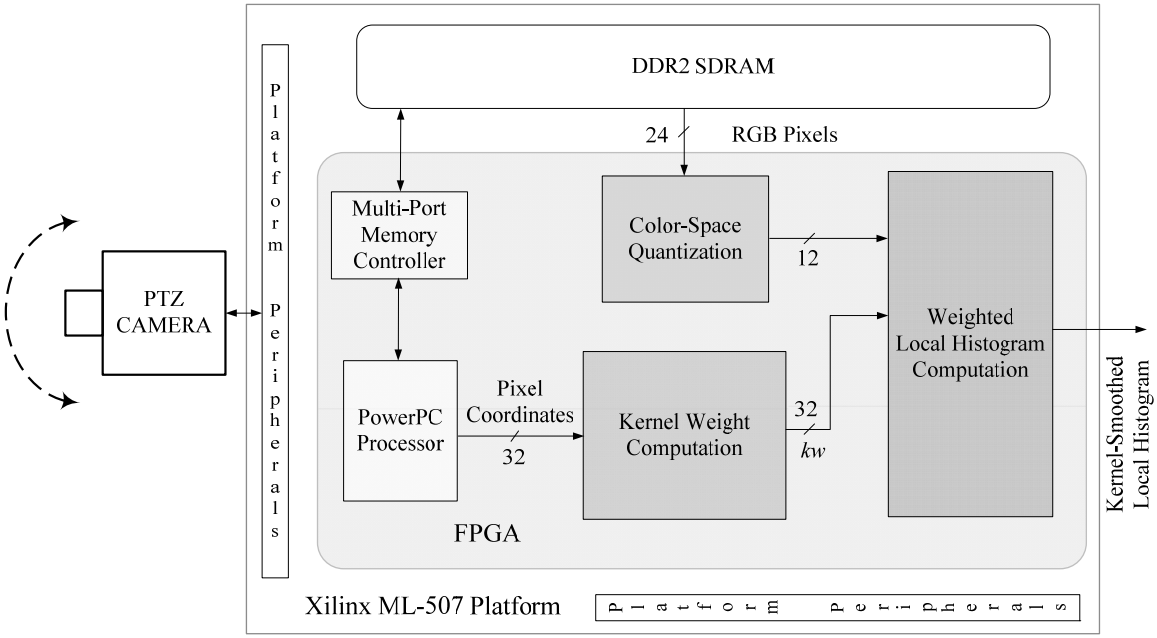


Fig. 6.8: Architecture for computing kernel-smoothed local histogram.

The proposed implementation requires single-cycle read-modify-write operations, which is achieved by operating one port of the dual-port BRAM in the read-first mode and other port as a write-first mode. Each pixel clock cycle is divided into two sub-cycles: a read cycle for getting the current value, and a write cycle for updating the memory content. This is achieved by operating the dual port BRAMS on both the edges of the video clock. After completion of all the read-modify-write cycles, the BRAM memory locations hold the kernel smoothed local

histogram of the image. The proposed architecture effectively utilizes various off-the-shelf FPGA macro elements and platform peripherals for the required throughput. The KSLH in the RGB color space is obtained by normalizing the pixel coordinates of the region of interest (ROI) to an unit circle before applying the weighting with the kernel profile which is defined for a unit circle [**27**]. The KSLH unit requires three main sub-units namely, color-space quantization, kernel-weight computation and weighted local histogram computation units. These architectural units are described below.

6.6.1 Color-space Quantization (*m*-Bins) and Color Histogram

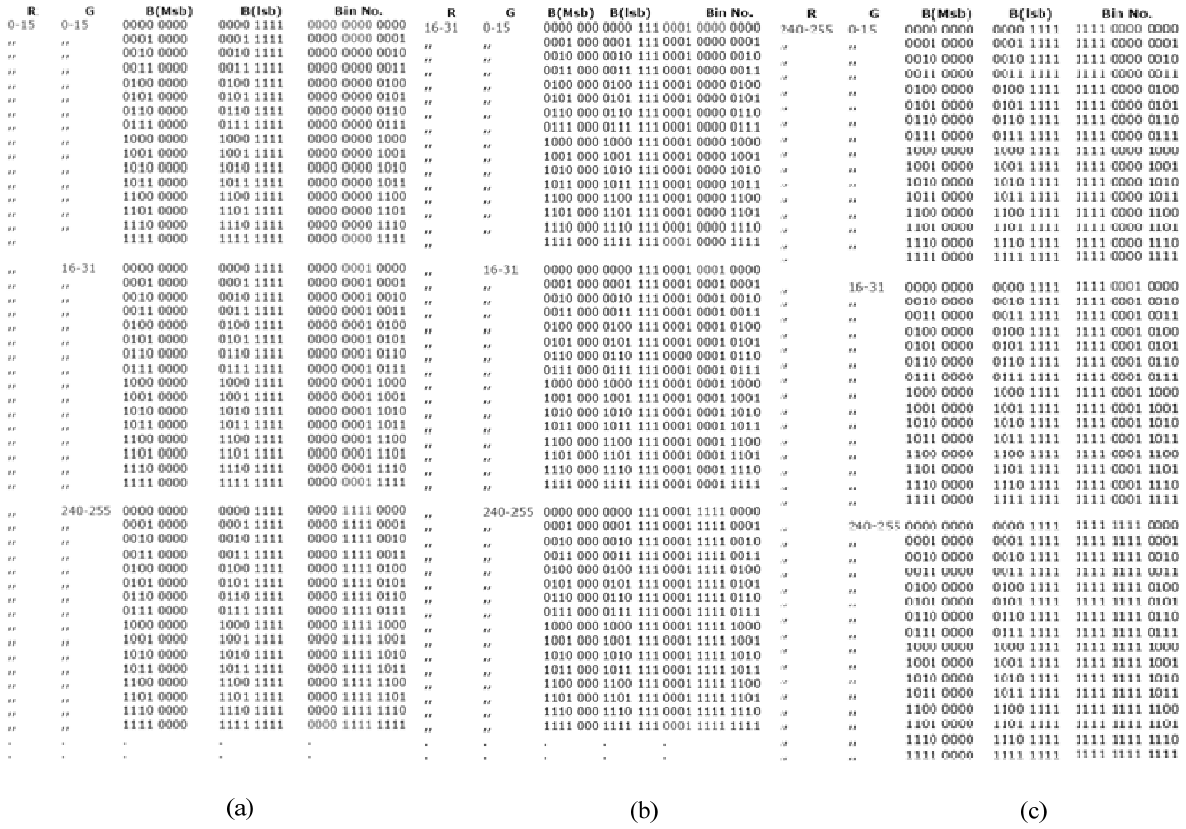
Full color space of size $256\times256\times256$ is quantized into $16\times16\times16$ color-space, as shown in Fig. 6.9. In Fig. 6.9 (a) the values of R varies from 0 to 15 for all the possible values of G and B values. Fig. 6.9 (b) shows the case where the value of R varies from 16 to 31 for each possible value of G and B. Similarly, in the case shown in Fig. 6.9 (c), the value of R varies from 240 to 255 for the full range of G and B.

The full R, G, and B space contains 4096 color bins, which can be addressed by concatenating the upper 4-bits of each of R, G and B. Thus, the 4096 locations are addressed by 12-bits, which can be written as:

$$Bin\ address = \text{R [7:4] \& G[7:4] \& B[7:4]} \tag{6.15}$$

The probability density function of color *u* is represented by the use of *m*-bin histogram and the R, G, B feature space is quantized into $16\times16\times16$ bins. Each bin corresponds to a range of pixel value as range of bin-0 is $(0\ldots15)$, bin-1 is $(16..31)$ and so on. The value of *m* is 4095 colors and range of color is $(0\ldots4095(m))$. By the use of above identity, the probability of color *u* is derived for the target model. While deriving the probability of each color in the target space, first three-color component (R, G, B) is checked as per (6.15) to find out the corresponding color bin it belongs to and further we put that pixel into the corresponding

location in the histogram. This process quantizes the full color space into the reduced color space.



(a) (b) (c)

Fig. 6.9: RGB color-space quantization into *m*-bins (a) R=0-15, G=0-255, B=0-255 (b) R=16-31, G=0-255, B=0-255 (c) R=240-255, G=0-255, B=0-255.

6.6.2 Kernel Weight Computation

Weights are required to smoothen the kernel function; it gives faster target localization in the successive frames because it increases the robustness of the estimation of color histogram, as surrounding pixels of the target center are less reliable owing to being frequently affected by the occlusion or background [**27**]. Here, the radius of kernel profile is taken equal to '1' and by assuming that the pixel coordinates of the target are normalized. Weights (kw) are derived by using the kernel function in normalized coordinates. After finding the distance (d_i) of the pixel coordinates from the center the weights are computed using the kernel function. Distance of the pixel from the center of the target is,

$$d_i = \sqrt{x_i^{\,2} + y_i^{\,2}} \tag{6.16}$$

$$kw = \begin{cases} 0 & \text{if } (d_i)^2 \geq 1 \\ \frac{2}{\pi}(1 - d_i) & \text{otherwise} \end{cases} \tag{6.17}$$

where d_i, is the distance of the pixel from center of the target and *kw* is derived from the Epanechnikov kernel profile. A pictorial view of the kernel weights (kw) for the Epanechnicov profile is shown in Fig. 6.10. The detailed architecture of the kernel-weight computation block is shown in Fig. 6.11.

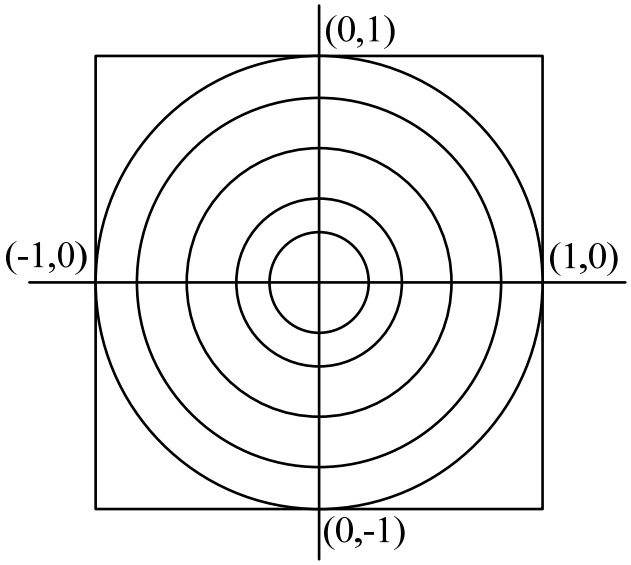


Fig. 6.10: A pictorial view of the kernel weights for the Epanechnikov kernel profile.

Here, the computed d_i is applied to both the inputs of the 32-bit fixed-point multiplier (in 16.16 format), which provides d_i^2. The square rooting in (6.16) is performed by computing binary logarithm, one-bit right shift and the antilogarithm. The subsequent component in the architecture is used to compute the kernel weight (kw) based on the condition of (6.17). The upper 16-bits of the multiplier, i.e., the integer part is applied to an OR network which is a 16-bit OR gate made from four 4-bit OR gates.

The output of the OR network is used as the select line for a multiplexer. Based on the OR network output, the multiplexer routes the appropriate data as per (6.17). To remove the divider for the computation of (6.17), the term, $(2/\pi)$, which is a fractional constant value is pre-computed and multiplied with $(1-d_i)$. The first input of the multiplexer comes from the output of a subtractor and a multiplier unit, which computes $(2/\pi)\times(1-d_i)$. The second input of the multiplexer is kept at logic zero level.

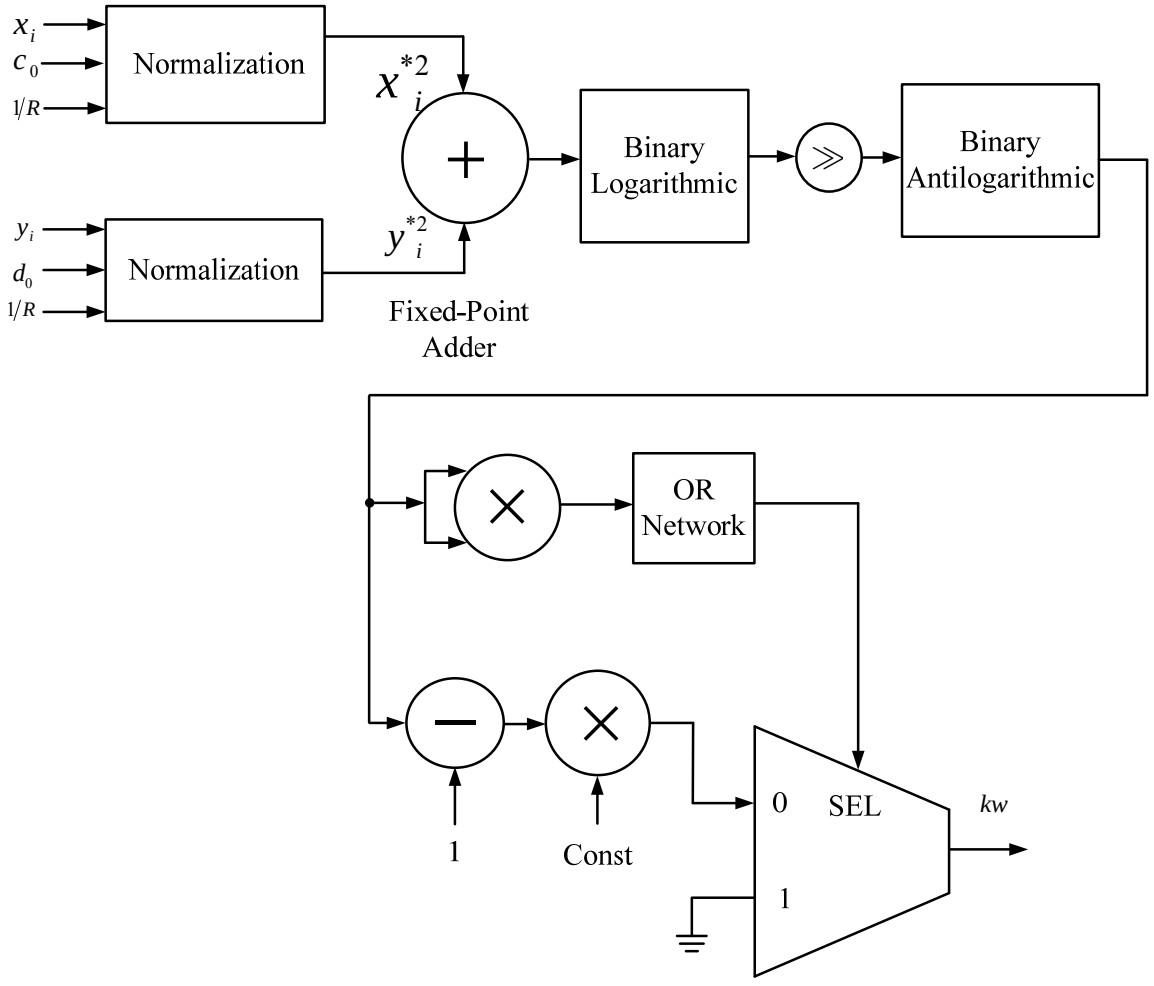


Fig. 6.11: Architecture for computing kernel weights.

6.6.3 Normalization unit

The normalization unit accepts the center coordinates (c_0, d_0) and the radius of the circle (R) to normalize all the local pixels and obtain $\mathbf{x}_i^* = (x_i^*, y_i^*)$. Normalization is required to eliminate the influence of different target dimensions as the object can have irregular shape. This is achieved by first normalizing the pixel coordinates of the target space to a unit circle. Further, independent rescaling of the row and column dimensions of the target space is done. The equations used for the rescaling, are as follows:

$$x_i^* = \frac{1}{R}(x_i - c_0) \tag{6.18}$$

$$y_i^* = \frac{1}{R}(y_i - d_0) \tag{6.19}$$

where, x_i and y_i are the row and column pixel coordinates from the circular (in general, this could be an ellipsoidal region) target region, respectively. Further, x_i^* and y_i^* are the normalized values of the x-cordinate and y-coordinate. Here, the enter coordinates of the circle is denoted by (c_0, d_0).

6.6.4 Weighted Local Histogram Computation

The weighted local histogram (6.4) is computed by incorporating a BRAM along with an incrementer. For this, each pixel clock cycle is divided into two sub-cycles: a read cycle for getting the current value, and a write cycle for updating the memory contents, as shown in Fig. 6.12. The content of the memory locations addressed by each newly arrived pixel is incremented by the computed kernel weight value (kw) based on the pixel location. After completion of the read-modify-write cycle, the BRAM memory locations hold the KSLH, $\hat{\mathbf{q}}$ of the image.

Proposed architecture for KSLH computation is shown in Fig. 6.13. Note that, in Fig. 6.13, the newly arrived color count is scaled with the corresponding kernel weight value (kw) (derived according to their distance from the center of ROI) before accumulation with the existing count.

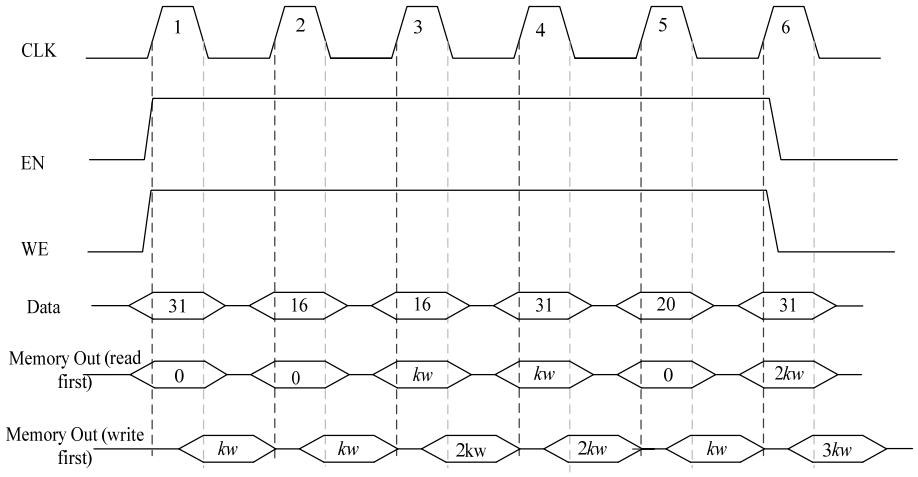


Fig. 6.12: Weighted local histogram computation timing diagram.

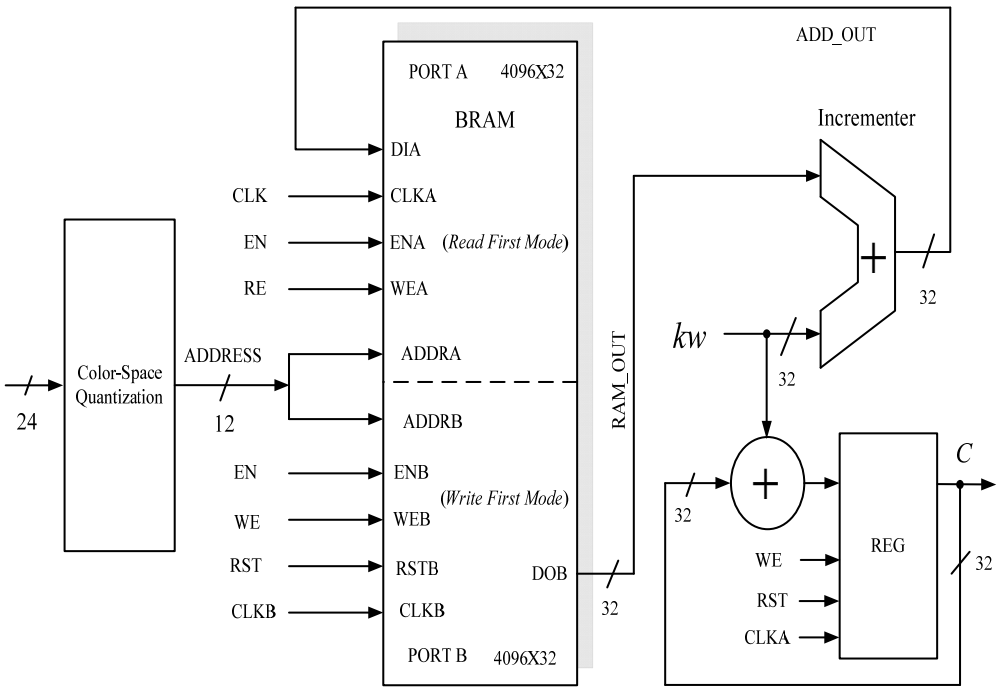


Fig. 6.13: Architecture for computing the kernel-smoothed local histogram of an image.

6.7 Bhattacharyya Coefficient Computation

As described in Section 6.3, after computing the target model $\hat{\mathbf{q}}$ (6.1) and the candidate model $\hat{\mathbf{p}}(\mathbf{y})$ (6.2), the KBMS algorithm needs Bhattacharyya coefficient in *Step* 1. The concept of LNS-based implementation is used to construct the datapath of the Bhattacharyya coefficient (6.11) computation unit. To compute the Bhattacharyya coefficient (6.11), can be expressed as,

$$\rho(\mathbf{y}) = \sum_{u=1}^{m} \sqrt{\frac{\hat{p}_u(\mathbf{y})}{C_2} \frac{\hat{q}_u}{C_1}} \tag{6.20}$$

where, C_1 is a constant and it is the accumulated value of (6.2) and C_2 is the accumulated value of (6.1). The values of C_1 and C_2 are computed as per (6.5) and (6.7) respectively. The terms in (6.20), for $u = 1, 2, 3, ..., m$ can be written in logarithmic domain as,

$$\log \sqrt{\frac{\hat{p}_u(\mathbf{y})}{C_2} \frac{\hat{q}_u}{C_1}} = \frac{1}{2}\left[\log\left(\hat{p}_u(\mathbf{y})\right) + \log\left(\hat{q}_u\right) - \log\left(C_1\right) - \log\left(C_2\right)\right] \tag{6.21}$$

So we can write (6.20) as,

$$\rho(\mathbf{y}) = \sum_{u=1}^{m} \text{Antilog}\left[\tfrac{1}{2}\left[\log\left(\hat{p}_u(\mathbf{y})\right) + \log\left(\hat{q}_u\right) - \log\left(C_1\right) - \log\left(C_2\right)\right]\right] \tag{6.22}$$

It is evident from (6.22), that, after computing the cumulative histograms (6.1) and (6.2) we require LNS-based datapath. To construct a LNS-based architecture and the associated system architectural building blocks, binary logarithmic units, right shifter and one binary antilogarithmic unit is needed. Based on these architectural units, the proposed architecture for computing the Bhattacharyya coefficient is shown in Fig. 6.14. In the Fig. 6.14, the BRAM-1 stores the kernel-smoothed local histogram of the target, which is defined in (6.1).

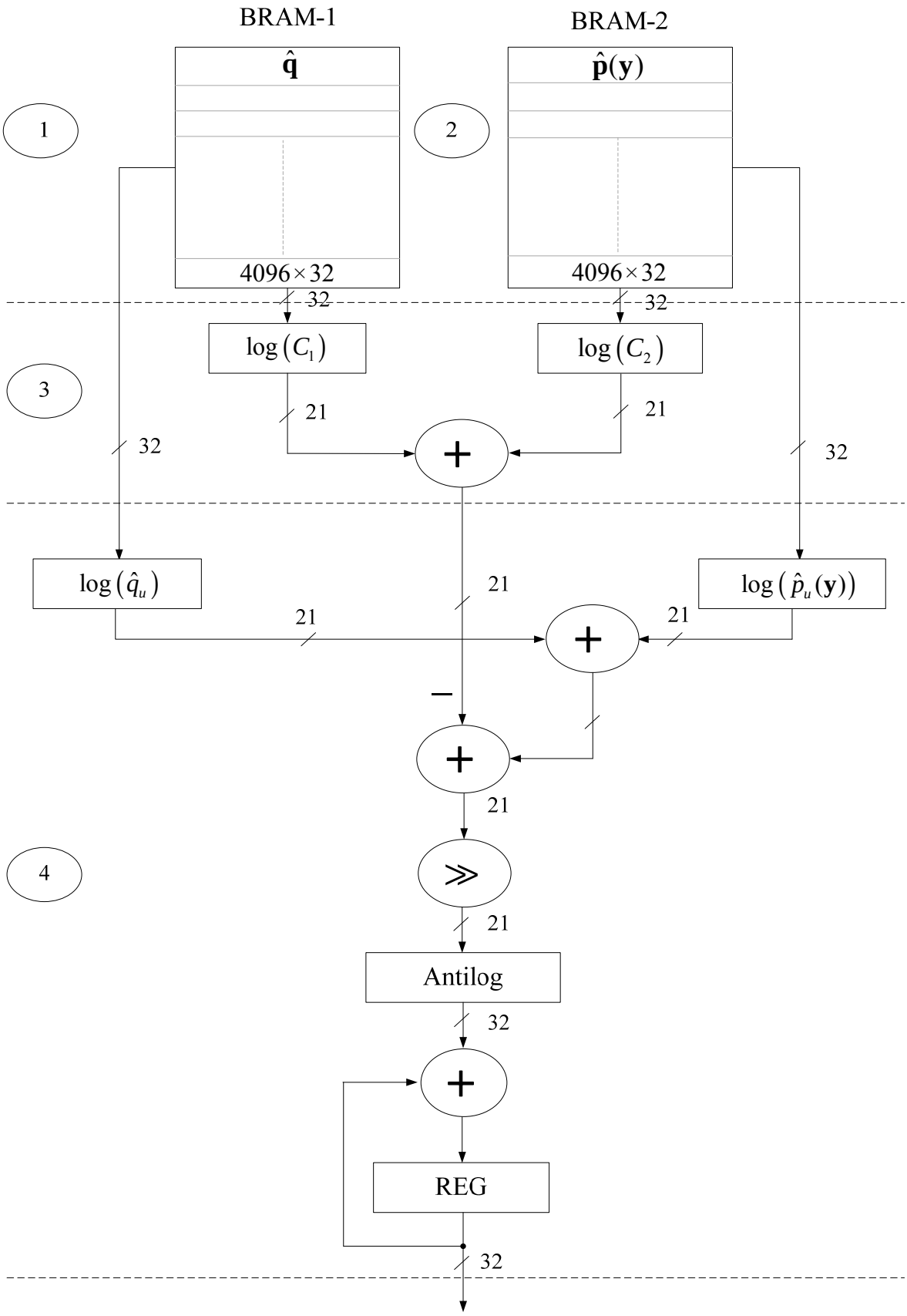


Fig. 6.14: Architecture for computing the Bhattacharyya coefficient.

Similarly, the BRAM-2 retains the kernel-smoothed local histogram of the candidate (6.3).The architecture shown in Fig. 6.14 works in five different stages, which are as follows:

Stage 0: In *Stage* '0', the BRAM-1, BRAM-2 and the last register is initialized to zeros.

Stage 1: In this stage, we compute the kernel-smoothed local histogram (6.1) using Block RAM memory (BRAM-1). The kernel-smoothed histogram as required in (6.1) and (6.2) is computed by the method described in Section 6.6 and shown in the Fig. 6.14. The computed value is in 32-bit fixed-point format.

Stage 2: In *Stage* '2', the kernel-smoothed local histogram of (6.6) is computed. The detailed computational process is illustrated in Section 6.6. The computed values are in 32-bit fixed-point and retained in the BRAM-2.

Stage 3: In *Stage* '3', the value of $\log(C_1)$ and $\log(C_2)$ is computed. The computed values are in 21-bit fixed-point format, which are added by a 21-bit fixed-point adder.

Stage 4: *Stage* '4', provides the computed Bhattacharyya coefficient value.

6.8 Mean Shift Weight Computational Unit

The KBMS algorithm requires mean shift weights in *Step* 2. As discussed and explained in Section 6.2.7, the mean shift weight can be computed by (6.13). Similar to Section 6.7, the concept of the LNS is used for computing (6.13), which is expressed as,

$$w_i = \sum_{u=1}^{m} \sqrt{\frac{\hat{q}_u / C_1}{\hat{p}_u(\hat{\mathbf{y}}_0)/C_2}}\,\delta[b(\mathbf{x}_i) - u] \tag{6.23}$$

where, C_1 represents the accumulated value of (6.2) and C_2 is the accumulated value of (6.1). The value of C_1 is computed as per (6.5) and the C_2 is computed by (6.7). Expression (6.23), can be written as,

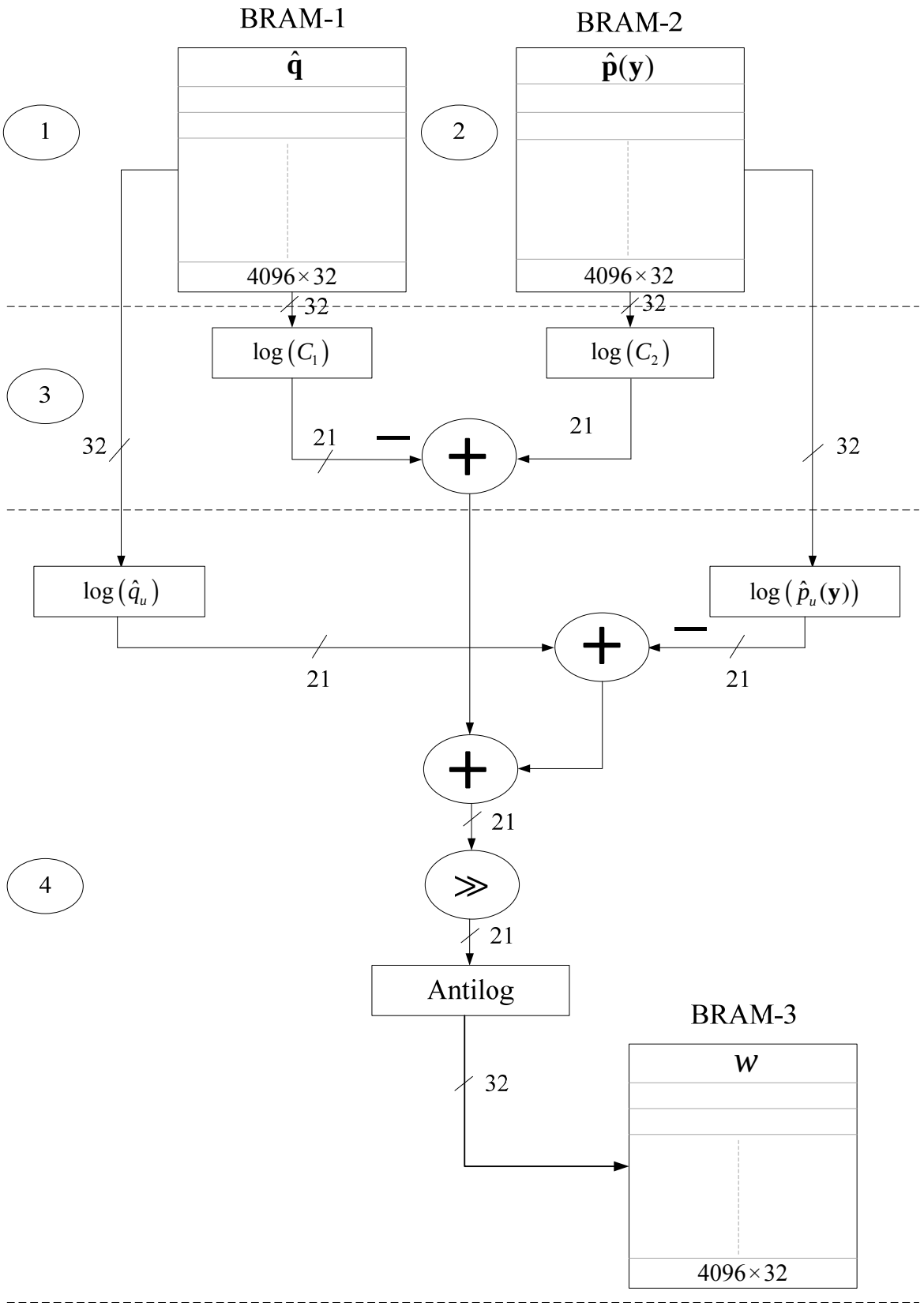


Fig. 6.15: Architecture for computing mean shift weights.

$$w_i = \sum_{u=1}^{m} \text{Antilog}\left[\tfrac{1}{2}\left[\log\left(\hat{q}_u\right) - \log\left(\hat{p}_u(\hat{\mathbf{y}}_0)\right) + \log\left(C_2\right) - \log\left(C_1\right) \right] \right] \delta\left[b(\mathbf{x}_i) - u\right] \quad (6.24)$$

After computing the kernel-smoothed local histograms (6.1) and (6.2), we require an LNS-based datapath to obtain w_i (6.24). To construct an LNS-based architecture and the required architectural building blocks, four binary logarithmic units, one right shifter and one binary antilogarithmic unit are required. By utilizing these architectural units, the proposed architecture for mean shift weight computation is shown in Fig. 6.15. The proposed architecture works in five different stages, which are discussed below:

Stage 0: In *Stage* '0', all the BRAMs are initialized to zero.

Stage 1: In this stage, we compute the kernel-smoothed local histogram (6.1) using BRAM-1 (this is the same computation using BRAM-1, as described in Section 6.7).

Stage 2: In *Stage* '2', the histogram (6.3) is computed. The computed histogram values are retained in the BRAM-2(this is the same BRAM-2 as described in the Section 6.7).

Stage 3: In *Stage* '3', the subtraction of two logarithmic values is performed.

Stage 4: *Stage* '4', provides the computed mean shift weights, which reside in BRAM-3.

6.9 New Mean Shift Location Computation

As explained in the above section the mean shift weights are computed and stored in BRAM-3. The 4096 locations of BRAM-3 contain 32-bit values. Similar to the Bhattacharyya coefficient and the mean shift location computations the LNS based approach is used to design the architecture for computing the new mean shift location. It uses three binary logarithmic units, two binary antilogarithmic units, four fixed-point adder/subtractor and two fixed-point multipliers. As explained in Section 6.2.7, the new location of the mean

shift is computed by (6.14), which is a simple weighted average. The *x*-coordinate of (6.14), which is represented by $\hat{\mathbf{y}}_{1(x-coor)}$ can be written as,

$$\hat{\mathbf{y}}_{1(x-coor)} = \text{Antilog}\left[\log\left(\sum_{i=1}^{n} \mathbf{x}_{i(x-coor)} w_i \right) - \log\left(\sum_{i=1}^{n} w_i \right) \right] \tag{6.25}$$

and similarly, the *y*-coordinate of (6.24) is expressed as,

$$\hat{\mathbf{y}}_{1(y-coor)} = \text{Antilog}\left[\log\left(\sum_{i=1}^{n} \mathbf{x}_{i(y-coor)} w_i \right) - \log\left(\sum_{i=1}^{n} w_i \right) \right] \tag{6.26}$$

Based on expressions (6.25) and (6.26), an architecture for computing the new mean shift location is proposed. The details of the architecture for computing the new mean shift location is shown in Fig. 6.16.

The proposed architecture works in two stages, which are explained below. First, each BRAM memory location is accessed for obtaining the new mean shift location coordinates. There are three concurrent computations in the iterative computing process. The objective of the first computation is to find out the summation of all the mean shift weights. This is obtained by accessing each locations of the BRAM and accumulating its value in a register, which results in a total mean shift weight $w_t = \sum_{i=0}^{4095} w_i$.

In concurrence, the second computation uses each BRAM memory location and multiplies it with the *x*-coordinate, $(x_i)_x$ of the normalized pixel coordinate, $(\mathbf{x}_i)$. The *x*-coordinate of the weighted sum (WS) computation provides as $\text{WS}_x = \sum_{i=0}^{4095} w_i \times (x_i)_x$. Similarly, in parallel, the y coordinates $(x_i)_y$ of $\mathbf{x}_i$ are multiplied with the total mean shift weight, w_t to provide $\text{WS}_y = \sum_{i=0}^{4095} w_i \times (x_i)_y$. After getting the values of w_t , WS_x and the WS_y the binary logarithmic circuits are used in the second stage of the computation.

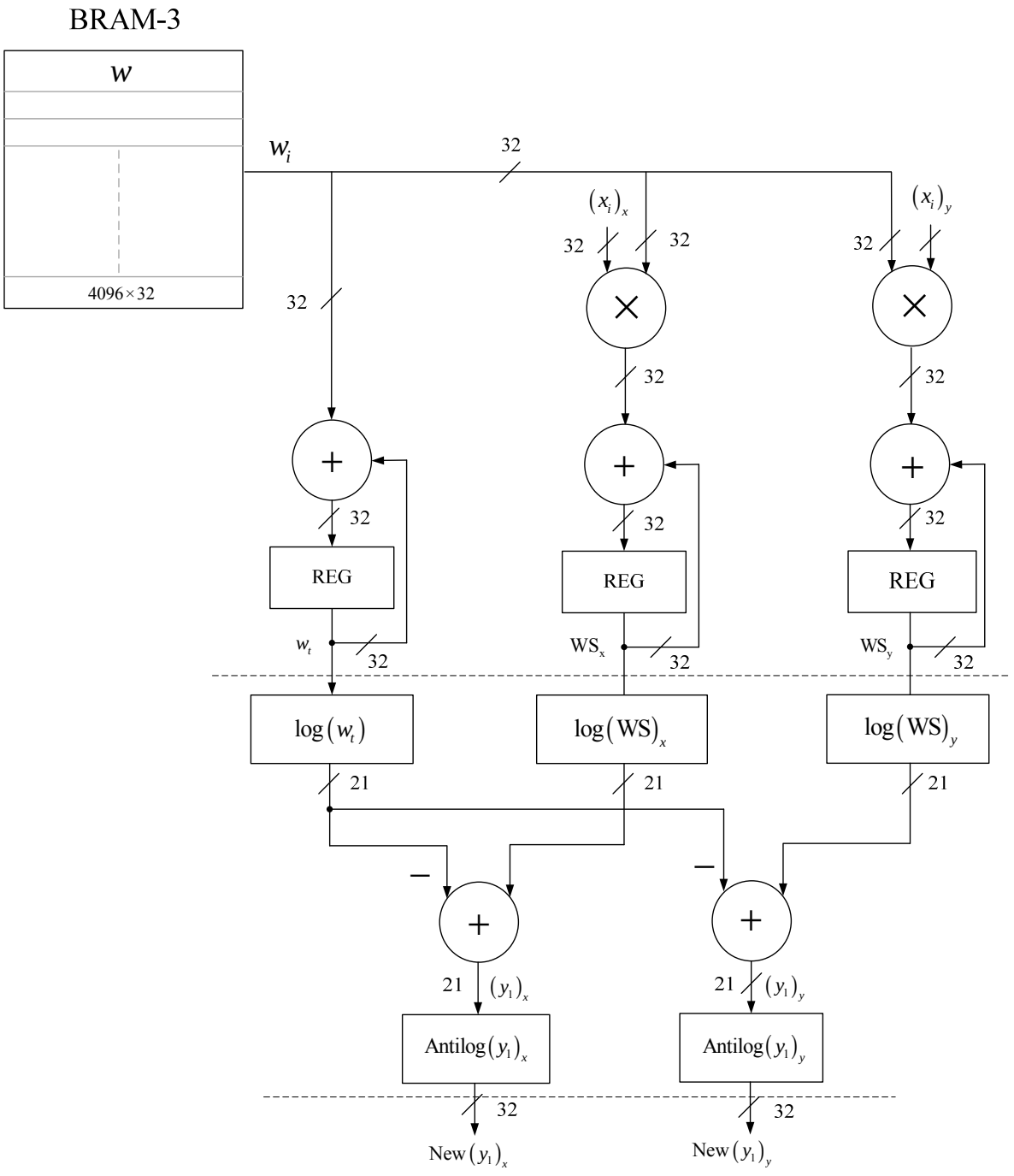


Fig. 6.16: Architecture for the new mean shift location computation.

As shown in Fig. 6.16, the leftmost logarithmic circuit provides the logarithmic equivalent of the denominator and the remaining two logarithmic blocks provide the numerator terms of (6.14) in the logarithmic form. With the LNS based approach, two fixed-point subtractors and

two antilogarithmic units compute (6.14). The computed new mean shift locations are represented as $\hat{\mathbf{y}}_{1(x-coor)}$ and $\hat{\mathbf{y}}_{1(y-coor)}$.

6.10 Integration of Architectural Building Blocks

While all the independent architectural units as general-purpose hardware components have been described in the foregoing sections, their integration for realizing the KBMS algorithm is described in this section. The full circuit level organization integrating the KSLH units, the Bhattacharyya coefficient computational unit, the mean shift weight computational unit and the new mean shift location computing unit is shown in Fig. 6.17. Here, the Block RAM-1 (BRAM-1) retains the kernel-smoothed local histogram of the target ($\hat{\mathbf{q}}$), and it is represented as KSLH-1 in the figure. Similarly, the Block RAM-2 (BRAM-2) holds the kernel-smoothed local histogram of the candidate ($\hat{\mathbf{p}}(\mathrm{y})$) shown as KSLH-2 in Fig. 6.17.

The first section of the circuit computes the Bhattacharyya coefficient, which uses the KSLH-1 and the KSLH-2 units, four logarithmic units, four arithmetical blocks for addition, one shifter for right shifting, one antilogarithmic unit and a register. In a concurrent, manner the middle block of the circuit computes the mean shift weight by using the KSLH-1 and KSLH-2 units. The computed mean shift weights are simultaneously stored in BRAM-3.

The weights are used by the last block of Fig. 6.17, which computes the new mean shift location. The architecture shown in Fig. 6.17 uses the shares the hardware resources across its units. The hardware resources which are shared among the Bhattacharyya coefficient computation unit and the mean shift weight computation unit includes, the logarithmic blocks, $\log(C_1)$, $\log(C_1)$, $\log(\hat{q}_u)$ and the $\log(p_u(y))$. Similarly, the $\log(w_t)$ hardware block is shared within the new mean shift location computation unit.

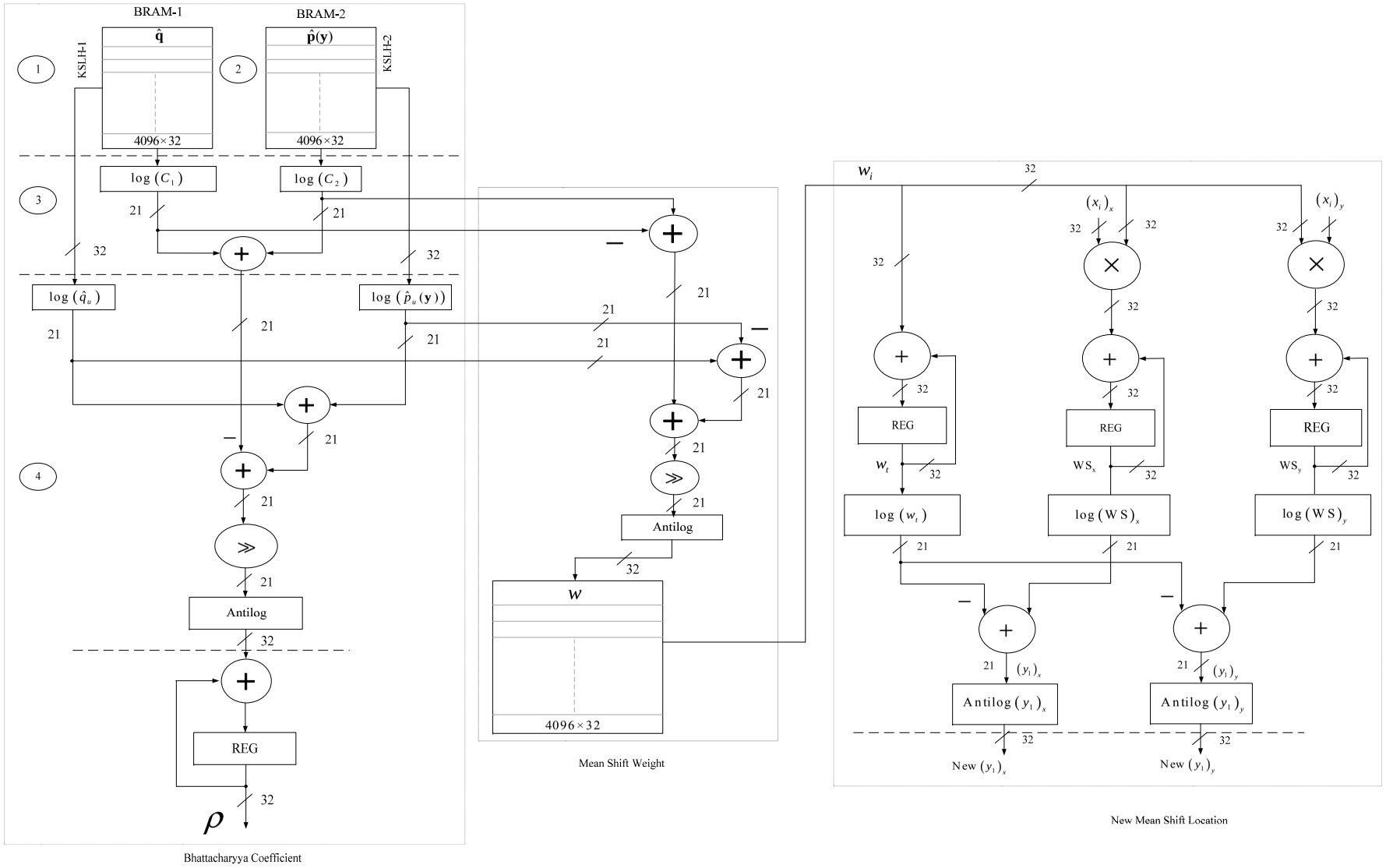


Fig. 6.17: Integration of architectural building blocks for realizing the KBMS algorithm.

The computed value of Bhattacharyya coefficient (ρ), mean shift weights (w_i), and new mean shift locations $\hat{\mathbf{y}}_{1(x-coor)}$ and $\hat{\mathbf{y}}_{1(y-coor)}$ are utilized by the KBMS algorithm explained in the Section 6.3. The Bhattacharyya coefficient (ρ) is utilized in *Step* 1, the mean shift weights (w_i) are used in *Step* 2, whereas, the new mean shift locations $\hat{\mathbf{y}}_{1(x-coor)}$ and $\hat{\mathbf{y}}_{1(y-coor)}$ are utilized in *Step* 3 of the KBMS algorithmic flow.

6.11 The System Control

The application software, written in 'C' programming language, runs on top of a Xilinx standalone software platform. The application program controls all the hardware blocks and platform peripherals by using the application programmer interface (API) offered by the software platform along with some of the basic functions developed for individual hardware blocks. The core communicates with the DDR2 SDRAM memory through a 32-bit native port interface (NPI) which is synchronous with the MPMC controller [**105**].

The embedded PowerPC processor, available in the Xilinx Virtex-5 xc5vfx70t FPGA device, is used to control the above architectural units. The PowerPC embedded processor uses the general-purpose registers of the I2C controller for the required control. The application program runs in the Xilinx SDK environment and it controls the complete system.

6.12 Results and Discussions

The proposed architecture has been implemented using the Very-high speed integrated circuit Hardware Description Language (VHDL) and synthesized with Xilinx ISE 14.2 for the Virtex-5 xc5vfx70tffg1136-1 FPGA device available on the Xilinx ML-507 platform. The FPGA device utilization summary for various modules is described below:

6.12.1 FPGA Device Utilization for the KSLH Module

The FPGA device utilization summary for implementing the kernel-smoothed local histogram (KSLH) computation is shown in Table 6.1. As shown in the table, the proposed architecture need around 1% of the FPGA slices. The architecture utilizes 2.7% (4 out of 148) of Block RAMs and 10.16% (13 out of 128) of DSP48E slices of Virtex-5 FX FPGA device. The computed power of the KSLH unit is 45.6 mW.

Table 6.1: FPGA Device Utilization for Implementing the Proposed Architecture for Computing Kernel-Smoothed Local Histogram of an Image.

Elements	Device Utilization	Utilization (%)
Slice LUTs	441 /44800	0.98
External IOBs	113/640	17.66
BRAMs	4/148	2.70
DSP48Es	13/128	10.16

In the proposed architecture, the complex arithmetic operations are converted into simple arithmetic operations by using binary logarithmic and antilogarithmic circuits using fixed-point datapath. The architecture uses standard 640×480 VGA resolution image. The image is captured from a high-resolution camera and subsequently buffered in the off-chip DDR2 SDRAM memory.

6.12.2 FPGA Device Utilization for the Bhattacharyya Coefficient Computation

The Bhattacharyya coefficient (BC) computation needs around 5% of the FPGA slices. Table 6.2, shows the device utilization summary of the proposed BC architecture. The architecture utilizes 5.4% (8 out of 148) of the Block RAM and 27.34% (35 out of 128) of DSP48E slices available in the Virtex-5 xc5vfx70t FPGA device.

The block-level architectural view of the proposed architecture for computing the Bhattacharyya coefficient is shown in the Fig. 6.18. It utilizes the two instances of the KSLH

unit, four instances of the binary logarithmic unit and one binary antilogarithmic unit. The power consumption of the Bhattacharyya coefficient architecture is 52.1 mW.

Table 6.2: FPGA Device Utilization for Implementing the Proposed architecture for Bhattacharyya Coefficient.

Elements	**Device Utilization**	**Utilization (%)**
Slice Registers	81/44800	0.18
Slice LUTs	2020 /44800	4.51
Bonded IOBs	131/640	20.47
BRAMs/FIFOs	8/148	5.4
BUFG/BUFGCTRLs	1/32	3.1
DSP48Es	35/128	27.34

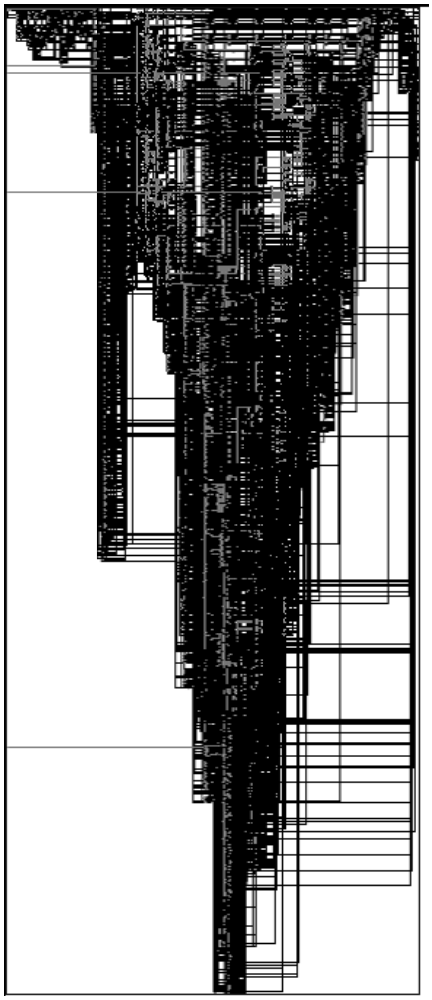


Fig. 6.18: FPGA technology schematic of Bhattacharyya coefficient computational unit.

Fig. 6.18 shows the graphical representation of the post-synthesis (optimized and mapped) netlist containing Xilinx primitives elements, which includes, look-up-tables (LUTs),digital clock manager (DCM), I/O buffers, and flip-flops. The ISE schematic viewer is use to visualize the properties of all the elements.

6.12.3 FPGA Device Utilization for the Mean Shift Weight Computation

The mean shift weight computational unit uses the same set of architectural components as required in the Bhattacharyya coefficient unit with the exception of an additional BRAM. As shown in Fig. 6.15 it uses four logarithmic units, one shifter and one binary antilogarithmic unit with three BRAMs. The FPGA device utilization summary for the mean shift weight computational unit is shown in Table 6.3.

Table 6.3: FPGA Device Utilization for Implementing the Mean Shift Weight Computational Architecture.

Elements	Device Utilization	Utilization (%)
Slice Registers	49/44800	0.10
Slice LUTs	1998 /44800	4.46
Bonded IOBs	145/640	22.66
BRAMs/FIFOs	12/148	8.10
BUFG/BUFGCTRLs	1/32	3.1
DSP48Es	35/128	27.34

After optimization and technology-targeting phase of the synthesis process, a schematic representation of the synthesized design is shown in Fig. 6.19. This schematic shows a representation of the design in terms of logic elements optimized to the target Xilinx Virtex-5 xc5vfx70t FPGA device. It contains LUTs, carry logic, I/O buffers, and other technology-specific components. The schematic shows a technology-level representation of the developed HDL. The computed power of the proposed architecture has been found to be 69.2 mW.

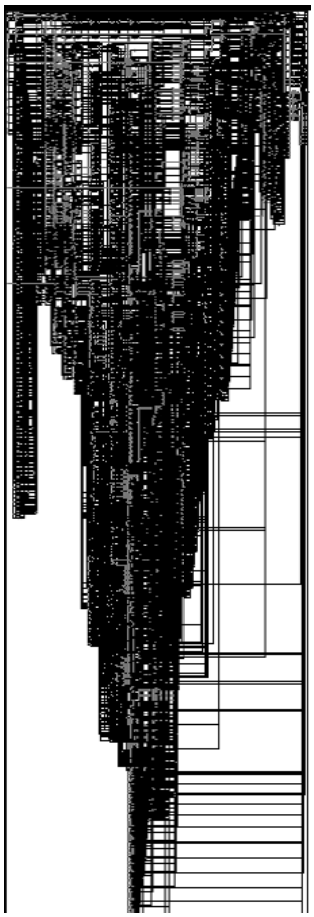


Fig. 6.19: Synthesized view of the mean shift weight computation module.

6.12.4 FPGA Device Utilization for the New Mean Shift Location Computation Unit

Computation of the new mean shift location is performed as shown in Fig. 6.16, using logarithmic and antilogarithmic units. The FPGA device utilization for it is shown in Table 6.4.

Table 6.4: FPGA Device Utilization for Implementing the Proposed Architecture for New Mean Shift Location Computation.

Elements	Device Utilization	Utilization (%)
Slice Registers	64/44800	0.14
Slice LUTs	1139 /44800	2.54
Bonded IOBs	145/640	22.67
BRAMs/FIFOs	4/148	6.75
BUFG/BUFGCTRLs	1/32	3.13
DSP48Es	12/128	9.36

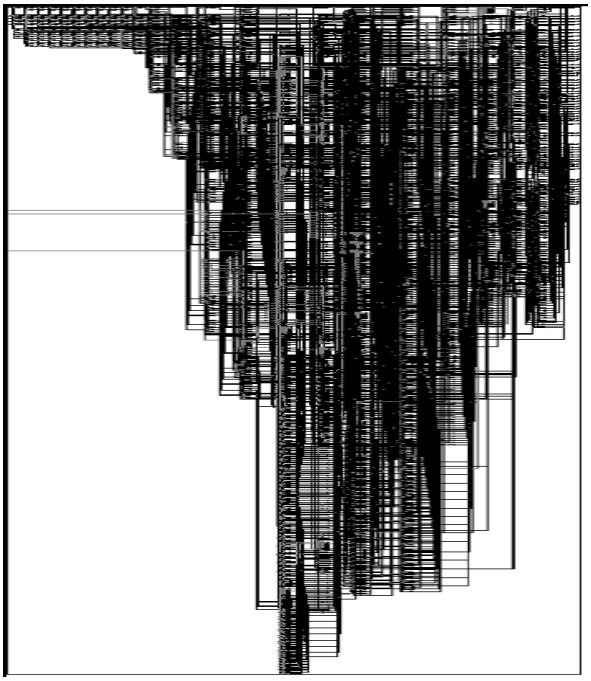


Fig. 6.20: Synthesized view of the new mean shift location computation.

The computed power of the design is found to be 45.7 mW. All the above architectural building blocks are used to realize the KBMS algorithm. The integrated design to realize the KBMS algorithm has been discussed in Section 6.10. In the following subsection the FPGA device utilization of the integration is covered.

6.12.5 FPGA Device Utilization for the KBMS Unit

The integrated module shown in Fig. 6.17 has been synthesized in the FPGA device. The device utilization for the complete KBMS unit is presented in Table 6.5.

Table 6.5: FPGA Device Utilization of Implementing the Complete KBMS Algorithm.

Elements	Device Utilization	Utilization (%)
Slice Registers	144/44800	0.32
Slice LUTs	3470 /44800	7.75
Bonded IOBs	273/640	42.66
BRAMs/FIFOs	12/148	8.11
BUFG/BUFGCTRLs	1/32	3.13
DSP48Es	46/128	35.94

As shown in Table 6.5, the KBMS unit uses 7.75 % FPGA slices, 8.11 % BRAMs, 3.13 % BUFG and 35.94 % DSP48E i.e. DSP slices. The synthesized view of the complete KBMS unit is shown in Fig. 6.21.

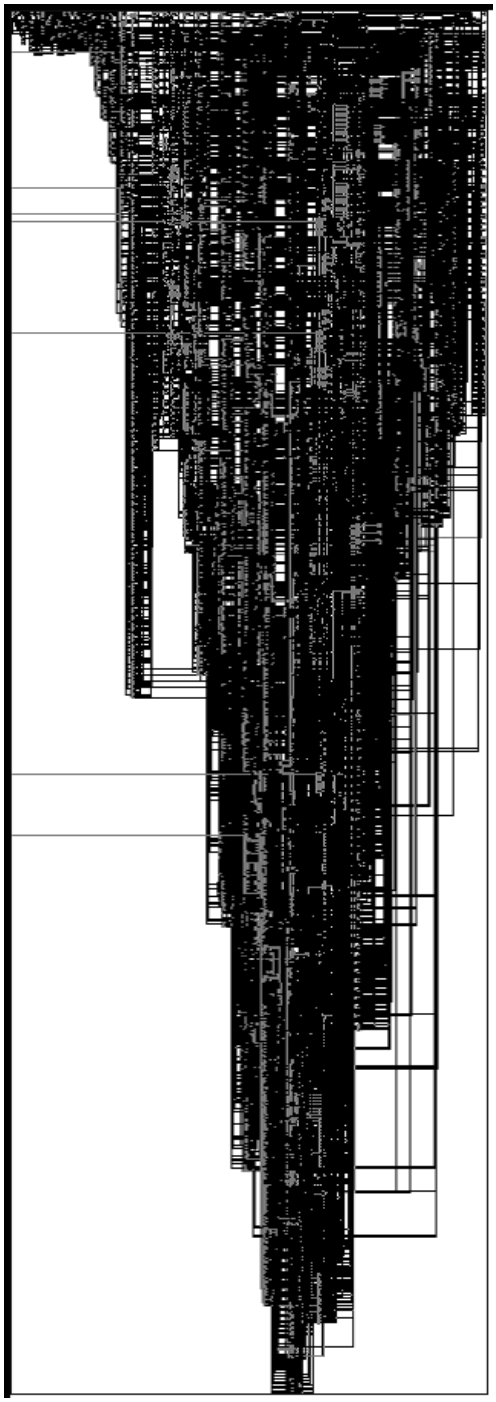


Fig. 6.21: Synthesized view of the complete KBMS unit.

In Fig. 6.21, the Xilinx synthesis tool (XST) infers components, such as, carry logic, BRAMs, shift registers, LUTs, clock buffers, multiplexers, arithmetic functions (DSP48E), which are associated with the Xilinx Virtex-5 xc5vfx70t FPGA device.

Table 6.6: FPGA Device Utilization Summary for Implementing Various Units of KBMS Algorithm.

Independent Architectures	**Slices** (11200)	**BRAMs** (148)	**DSP48Es** (128)	**Bonded IOBs** (640)
Image Acquisition	2240	28	0	121
Binary logarithm	53	0	2	53
Binary antilogarithm	41	0	1	53
Image thresholding	168	4	5	33
KBMS algorithm	868	12	46	273

Table 6.6 summarizes the FPGA device utilization for all the units. The complete system view with the KBMS core is discussed below.

6.13 The Complete System View for implementation of KBMS Algorithm

The proposed architecture can be used as an intellectual property (IP) core in an embedded system environment. The placement of the KBMS core along with its interfaces with other IPs and buses is shown in Fig. 6.21. For communication with the embedded PowerPC 440 (PPC440) processor, the proposed system architecture utilizes processor local bus (PLB) and memory controller interface (MCI) bus protocols. The MCI provides an interface between PPC440 processor and a soft multi-port memory controller (MPMC) implemented in the FPGA fabric [**34**]. The frame acquisition uses the PPC440 processor and the Xilinx video frame buffer controller (VFBC) available with the MPMC IP. The AD9980 video decoder chip is programmed through inter-integrated circuit (I2C) bus, which generates 25.175 MHz video clock. All the various architectural units utilize the generated video clock, which is managed by the digital clock manager (DCM).

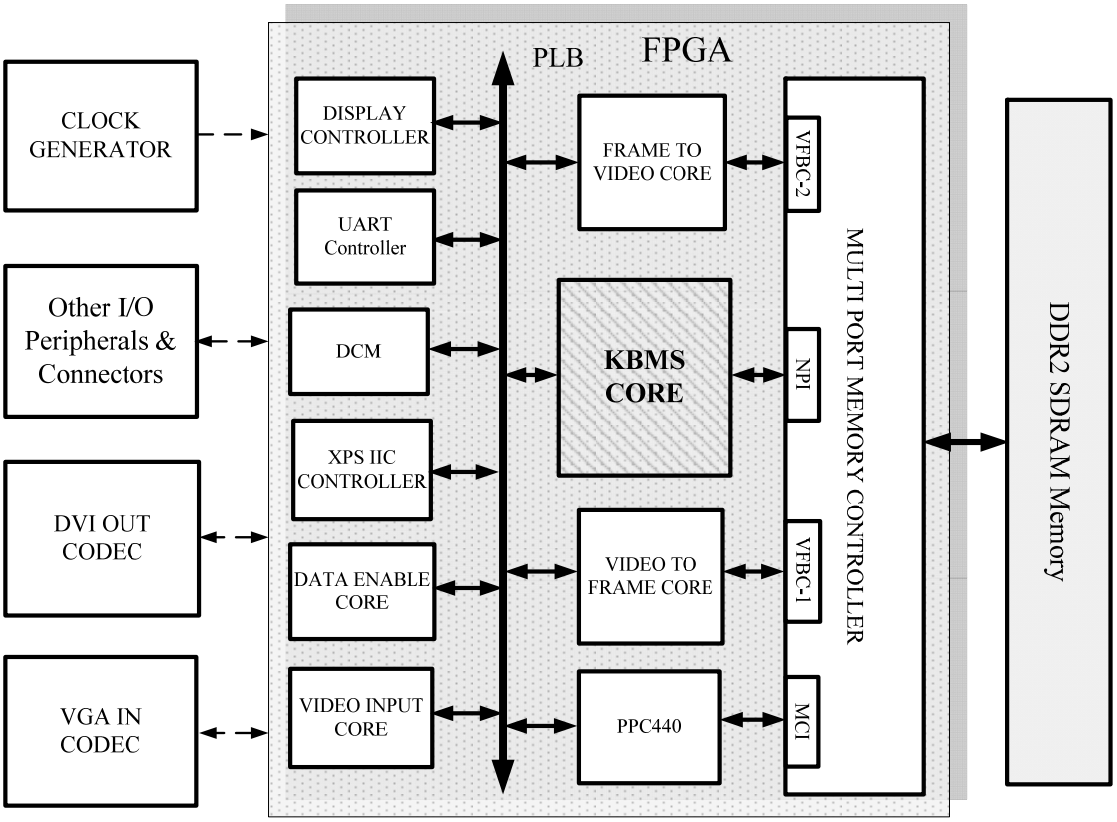


Fig. 6.22: KBMS core in a system environment.

6.14 Conclusion

In this chapter, architectures for the kernel-smoothed local histogram (KSLH) computation, Bhattacharyya coefficient computation, mean shift weight computation and new mean shift location computation have been proposed. The proposed architectures have been utilized to implement the kernel-based mean shift (KBMS) object tracking algorithm. The presented architecture uses dual-port BRAMS with single cycle read-modify-write operation to compute the kernel-smoothed local histogram for an image. Here, an embedded PowerPC processor controls the frame acquisition part of the architecture, which uses DDR2 SDRAM memory, video decoder and display controller chips, available on the Xilinx ML-507 Virtex-5 FX FPGA device based platform. Xilinx embedded development kit (EDK) design tool is used to integrate the required IPs with the embedded PowerPC processor, which runs application program and the configuration software.

In the proposed architectures, most of the operations are performed in the 32-bit fixed-point format. The complex arithmetic operations are realized through fixed-point binary logarithmic and antilogarithmic units. The architecture has the advantages of minimizing logic resources, and processing of large datasets in real-time, by realizing time-critical processes through the available BRAMS and DSP slices. The design results in an effective use of FPGA resources for the required throughput and speed goal. The work presents and demonstrates an effective design approach for realizing high-performance embedded hardware-software based systems.

CHAPTER 7

CONCLUSIONS

7.1 Summary of Achievements

In this thesis, a set of hardware architectural modules have been presented for resource-efficient embedded realization of image and video processing applications. These architectures have been designed using platform-based design methodology that allows exploration and development of new and emerging image and video processing systems. The hardware architectures are realized in the Virtex 5 FPGA device available on the ML-507 platform. The designed modules can be utilized as intellectual property (IP) cores for rapid development of systems.

We have presented a real-time image and video acquisition and display module that is required across a wide range of image/video processing applications. Next, to efficiently realize complex arithmetic functions such as square root, division, and raise-to-the-power function by using logarithm number system (LNS), architectures for binary logarithm and antilogarithm have been presented. Many image/video processing applications require an efficient hardware architecture for image thresholding. We have presented a resource-efficient FPGA-based architecture for global image thresholding. Also in various image/video applications, it is necessary to find connected components present in binary images. We have presented an improved label-equivalence based two-scan connected component algorithm along with its implementation on the embedded PowerPC processor. The presented algorithm improves upon the Stefano-Bulgarelli (SB) algorithm by modifying the equivalence handling procedure of SB algorithm for efficient identification of connected components. The improved connected component algorithm has been used for obtaing target

coordinates, which are required for embedded implementation of kernel-based mean shift (KBMS) object tracking algorithm.

Finally, all the above described hardware and software modules alongwith some additionally required hardware blocks have been utilized for the embedded FPGA implementation of the KBMS object tracking application. The additionally required hardware architectural blocks for implementation of the KBMS algorithm are blocks for similarity measure computation, center of gravity computation, mean shift weight computation and new mean shift location computation. FPGA-based architectures for these blocks have been proposed and implemented on the Virtex-5 FX series device available on Xilinx ML-507 platform. The developed architectures have the advantages of reduced logic resources and processing of large datasets by realizing time-critical processes in the available BRAMS and DSP slices. The Xilinx embedded development kit (EDK) design tool has been used to manage the integration of various architectures and algorithms presented in this thesis.

Register-transfer-level (RTL) modeling of all the architectural building blocks has been done in VHDL language. The datapath of the architecture has been optimized by using the concepts of functional unit sharing and operator merging. In the designed datapath, outputs of the different functional units share the common destinations at different times. Therefore, in the datapath, several signals are merged into a bus. This design strategy leads to the minimization of substantial amount of FPGA resources. In the same way, registers with non-overlapping access times are merged to share the register input and output ports. The modular structure of the developed datapath also supports pipelining for higher throughput.

The work starts with configuration of the Xilinx FPGA-based platform and the required peripherals for image and video processing applications. The embedded PowerPC processor available in the FPGA device is used to configure the VGA input video codec and the display

controller on-platform peripherals. The control registers of these peripherals are programmed through inter-integrated circuit (I2C) bus using low-level device driver functions and their application programming interfaces (API). The design is implemented in the Virtex-5 FPGA fabrics, which facilitates real-time video streaming on a VGA monitor. Subsequently, an FPGA-based embedded architecture is implemented in the Xilinx ML-507 platform for the frame acquisition application. The architecture presented allows buffering of 640×480 resolution video frames in the DDR2 SDRAM memory. This embedded design is utilized by different applications for further processing of captured video frames.

To compute the complex arithmetic functions required for image/video processing a simple integer datapath is created. The designed datapath uses 32-bit unsigned fixed-point numbers and utilizes the concepts of logarithmic number system. Architectures for the binary logarithm and antilogarithm units are proposed for finding their approximate values within the specified range. To find the characteristic part of the logarithm of a binary numbers, a novel leading-one finder circuit has been proposed. The fractional part approximation unit proposed, computes the mantissa part of the binary logarithm. The same circuit arrangement has been used to compute the binary logarithm of integer and fractional numbers. The proposed architecture for logarithm computation utilizes only 209 LUTs out of available 44800 LUTs, which represent around 0.47 % utilization. Similarly, out of the 128 available DSP48E slices, the proposed architecture uses only 02 slices, which represents around 1.6 % utilization. In antilogarithm computation, the characteristic portion of the binary number is used to shift the computed mantissa part with the help of a barrel-shifter. The barrel-shifter of the proposed architecture of antilogarithm unit uses a few multiplexers to route the logically shifted value of the mantissa part. The circuit arrangement for computing binary antilogarithm also uses same set of circuit elements to compute binary antilogarithm of positive and negative numbers. The proposed architecture for binary antilogarithm

computation requires only 0.37 % of the FPGA LUTs, 0.78% of the DSP48E slice available with the Virtex-5 FPGA device.

Error analysis has been performed on the implemented architectures using thousands of uniformly distributed random numbers. It shows that the maximum error is percentage of 0.05 % with 16.16 fixed-point numbers and 0.34 % with fractional numbers for binary logarithm computation. In binary antilogarithm computation the percentages of computational errors are found to lie in the range of $\pm 0.08\%$ for positive binary numbers and -0.2% to $+0.6\%$ for negative binary numbers. The associated percentage computational errors are relatively small percentage band, which is acceptable for a wide range of image and video processing applications. The developed logarithmic and antilogarithmic units are utilized for the purpose of hardware architecting of various compute-intensive blocks presented in the thesis.

A novel hardware architecture for global image thresholding operation has been presented next. Thresholding operation is performed on gray-level images, so that optimal value of threshold could be obtained for binary conversion of images. An efficient global automatic image thresholding algorithm, proposed by Otsu, is taken for hardware implementation. The compute-intensive between class variance computation (BCV) is required for implementing Otsu's algorithm. In the presented work, an area-efficient FPGA-based architecture for the computation of BCV is proposed. It requires computing normalized cumulative histogram (NCH) and normalized cumulative intensity area (NCIA). These modules are developed by incorporating the embedded components available in the FPGA, which include digital clock manager (DCM), BRAMs, and DSP slices. The proposed architecture requires only 1.5 % of the FPGA slices for the computation of between-class variance, 2.7% of the Block RAMs have been used to compute the cumulative mean and moments and we are using total 3.9% of

available DSP48E slices. The proposed architecture has the advantages of minimizing logic resources and the ability to process large datasets by conducting time-critical functions on available BRAMs and DSP slices. The FPGA device utilization of the design shows that the proposed architecture utilizes a small number of FPGA BRAMs, DSP slices and LUTs.

The binary image obtained from the image thresholding unit is utilized by the connected component analysis algorithm. In our work, we have proposed an improved label-equivalence based connected component analysis algorithm. The proposed algorithm improves on the Stefano-Bulgarelli (SB) algorithm by modifying its equivalence handling procedure, and removes the partial merging problem associated with the SB algorithm. It searches for the label-equivalence and as soon as it is found, the algorithm resolves the label-equivalences in the first scan itself. The label-equivalence process is independent from the different temporary labels assigned. The improved algorithm is implemented on the embedded PowerPC processor of the ML-507 platform. The results demonstrate that the improved algorithm handles equivalences efficiently and gives accurate count of connected components.

An embedded architecture for object tracking application is considered next, which utilizes the developed architectural building blocks and algorithms. Additionally required application-specific architectural building blocks are developed for this purpose. The kernel-based mean shift (KBMS) algorithm is taken for the embedded realization of the object tracking application. To perform analysis on the KBMS algorithm its MATLAB/C implementation is developed. Computation of kernel-smoothed local histogram, center of gravity, Bhattacharyya coefficient based local similarity measure and the mean shift weight are found to be the main time-critical parts of the KBMS algorithm. FPGA-based hardware architecture blocks for implementing the computations are proposed and presented in detail. The embedded PowerPC processor has been used to run the software components as well as

to configure and control various on-platform system peripherals used. The power consumption associated with different architectural modules is obtained by using Xilinx XPower Analyzer tool.

7.2 Future Scope of Work

Rapid growth of image and video processing systems has raised increasing demand for system functionality and diversity. Hardware architectures and algorithms presented in this thesis can be part of the architectural development for any practical image and video processing system using FPGA-based platform. The approach followed can easily be transferred on to future FPGA-based platforms and their associated embedded processors leading to design gains in terms of programmable systems integration, increased system performance and overall cost reduction.

Today and in foreseeable future application-specific system designing will demand integration of various heterogeneous components. Intellectual property (IP) based design and implementation approach as presented in this thesis can support the development of application-specific complex image and video processing systems and their derivatives. With the presented design approach, development of complex practical system architectures and their prototypes is feasible in minimal amounts of time. The developed hardware/software building blocks along with standard IPs can also be leveraged for the development of highest performance-lowest power solutions for applications that target mass markets.

Finally, designing of various derivatives of the developed architectures and their integration is possible for any processor of choice. We have so far considered only a single processor, which is embedded in the FPGA device, however, with the availability of multiple or multi-core processors in various upcoming platforms, the hardware/software units can be efficiently exploited for designing future embedded image/video processing systems.

REFERENCES

[1] N. J. Li, C. F. Chuang, Y. T. Wei, W. J. Wang, and H. C. Chen, "A video surveillance system for people detection and number estimation," in *Proceedings of IEEE Int'l Conf. on Fuzzy theory and it's applications (iFUZZY)*, Taichung, Taiwan, 2012, pp. 249-253.

[2] N. Kehtarnavaz and M. Gamadia, *Real-time Image and Video Processing:From Research to Reality*. Morgan and Claypool Publication, 2006, DOI:10.2200/S00021ED1V01Y200604IVM005.

[3] J. R. Parker, *Algorithms for Image Processing and Computer Vision*, 2nd ed. Wiley Publishing Inc., 2011.

[4] A. Kumar, D. C. Wong, H. C. Shen, and A. K. Jain, "Personal verification using palmprint and hand geometry biometric," in *Audio-and Video-Based Biometric Person Authentication*. Springer Berlin Heidelberg, 2003, pp. 668-678.

[5] D. Comaniciu, V. Ramesh, and P. Meer, "Real-time tracking of non-rigid objects using mean shift," in *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, vol. 2, Hilton Head Island, SC, 2000, pp. 142-149.

[6] H. Tian, T. Srikanthan, and K. V. Asari, "Automatic segmentation algorithm for the extraction of lumen region and boundary from endoscopic images," *Medical and Biological Engineering and Computing*, vol. 39, no. 1, pp. 8-14, 2001, DOI:10.1007/BF02345260.

[7] C. Y. Chang, Y. F. Lei, C. H. Tseng, and S. R. Shih, "Thyroid segmentation and volume estimation in ultrasound images," *IEEE Trans. on Biomedical Engineering*, vol. 57, no. 6, pp. 1348-1357, 2010.

[8] R. Szeliski, *Computer Vision Algorithms and Applications*. Springer-Verlag, 2011.

[9] T. Acharya and A. K. Ray, *Image Processing Principles and Applications*. Wiley Inter-science, 2005.

[10] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design Modeling, Synthesis and Verification*. New York: Springer Publication, 2010.

[11] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design:The Hardware/Software Interface*. San Francisco, California:: Morgan Kaumann Publication, 2010.

[12] W. Wolf, *High Performance Embedded Computing, Architectures, Applications and Methodologies*. San Francisco, California: Morgan Kaufmann Publishers, 2007.

[13] C. M. Maxfield, *The Design Warrior's Guide to FPGAs*. Amsterdam: Elsevier Publication, 2004.

[14] P. R. Wilson, *Design Recipes for FPGAs*. Amsterdam: Elsevier Publication, 2007.

[15] R. Pellizzoni and M. Caccamo, "Real-time management of hardware and software tasks for FPGA-based embedded systems," *IEEE Trans. on Computers*, vol. 56, no. 12, pp. 1666-1680, 2007, DOI:10.1109/TC.2007.70763.

[16] S. McBader and L. P., "An FPGA implementation of a flexible, parallel image processing architecture suitable for embedded vision systems," in *Proceedings of the IEEE Int'l Parallel and Distributed Processing Symp.*, Nice, France, April, 2003.

[17] J. A. Kalomiros and J. Lygouras, "Design and evaluation of a hardware/software FPGA-based system for fast image processing," *Microprocessors and Microsystems*, vol. 32, no. 2, pp. 95-106, 2008.

[18] J. Cong, et al., "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Trans.on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473-491, Apr. 2011.

[19] V. K. Madisetti and C. Arpikanondt, *A Platform-Centric Approach to System-on-Chip (SoC) Design*. Springer Science, 2005.

[20] I. Bravo, et al., "Efficient smart CMOS camera based on FPGAs oriented to embedded image processing," *Sensors*, vol. 11, no. 3, pp. 2282-2303, 2011, DOI:10.3390/s110302282.

[21] J. Li, H. He, H. Man, and S. Desai, "A general-purpose FPGA-based reconfigurable platform for video and image processing," *Advances in Neural Networks*, vol. 5553, pp. 299-309, 2009.

[22] Xilinx. (2012) Embedded processing peripheral IP cores. [Online]. http://www.xilinx.com/ise/embedded/edk_ip.htm

[23] Xilinx. Xilinx FPGAs. [Online]. http://www.xilinx.com/fpga/index.htm

[24] Altera. (2012, Jan.) Altera FPGAs. [Online]. http://www.altera.com/

[25] Celoxica. (2013, Mar.) Celoxica | Ultra-low latency and accelerated computing solutions. [Online]. http://www.celoxica.com/

[26] D. Densmore and R. Passerone, "A Platform-Based Taxonomy for ESL Design," *IEEE Design & Test of Computers*, vol. 23, no. 5, pp. 359-374, May 2006, DOI:10.1109/MDT.2006.112.

[27] D. Comaniciu, S, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564-577, May 2003, DOI:10.1109/TPAMI.2003.1195991.

[28] L. P. Carloni, F. D. Bernardinis, C. Pinello, A. L. Sangiovanni-Vincentelli, and M. Sgroi, "Platform-based design for embedded systems," in *The Embedded Systems Handbook*, R. Zurawski, Ed. Boca Raton, USA: CRC Press, 2005, pp. 1-26.

[29] F. Vahid and T. Givargis, "Platform tuning for embedded systems design," *IEEE Trans. on Computer*, vol. 34, no. 2, pp. 112-114, 2001, DOI:10.1109/2.901171.

[30] A. S. Vincentelli and G. Martin, "Platform based design and software design methodology for embedded systems," *IEEE Design and Test of computers*, vol. 18, no. 6, pp. 23-33, 2001.

[31] B. Bailey and G. Martin, *ESL models and their application*. New York: Springer Publication, 2010.

[32] Xilinx. (2010) Xilinx UG190 Virtex-5 FPGA User Guide. [Online]. www.xilinx.com/support/documentation/user_guides/ug190.pdf

[33] Xilinx. Virtex-5 FXT FPGA ML507 evaluation platform. [Online]. http://www.xilinx.com/ml507

[34] Xilinx. (2011) Embedded processor block in Virtex-5 FPGAs. [Online]. http://www.xilinx.com/support/documentation/user_guides/ug200.pdf

[35] Xilinx. (2011, Apr.) ChipScope Pro and the Serial I/O Toolkit. [Online]. http://www.xilinx.com/tools/cspro.htm

[36] D. Amos, A. Lesea, and R. R., *FPGA-based Prototyping Methodology Manual*. USA: Synopsys, 2011.

[37] Xilinx. (2011) EDK concepts, tools and training. [Online]. http://www.xilinx.com/support/documentation/dt_edk_edk12-4.htm

[38] Y. Lei, et al., "The platform of image acquisition and processing system based on DSP and FPGA," in *Int'l Conf. on Smart Manufacturing Application*, KINTEX, Gyeonggi-do, Korea, 2008, pp. 470-473.

[39] CASPER. (2013) Reconfigurable open architecture computing hardware. [Online]. https://casper.berkeley.edu/wiki/ROACH

[40] CASPER. (2013) ROACH2. [Online]. https://casper.berkeley.edu/wiki/ROACH2

[41] H. Zhang, et al., "A high-performance FPGA platform for adaptive optics real-time control," in *Proceedings of SPIE 8447, Adaptive Optics Systems III*, Amsterdam, Netherlands, 2012, pp. 84472E-84472E.

[42] G. Sarah, J. W. Judy, and M. D., "An FPGA-based platform for accelerated offline spike sorting," *Journal of Neuroscience Methods*, vol. 215, no. 1, pp. 1-11, Apr. 2013.

[43] Xilinx. (2011) Avnet Spartan-6 FPGA industrial video processing kit. [Online]. http://www.xilinx.com/products/boards-and-kits/AES-S6IVK-LX150T-G.htm

[44] Xilinx. (2010) Virtex-4 Video Starter Kit. [Online]. http://www.xilinx.com/products/devkits/HW-V4SX35-VIDEO-SK-US.htm

[45] Xilinx. (2010) XtremeDSP Video Starter Kit-Spartan-3A DSP edition. [Online]. http://www.xilinx.com/products/boards-and-kits/DO-S3ADSP-VIDEO-SK-UNI-G.htm

[46] Xilinx. (2010) Platform Studio and the Embedded Development Kit (EDK). [Online]. http://www.xilinx.com/tools/platform.htm

[47] J. H. Sohn, R. Woo, and H. J. Yoo, "A programmable vertex shader with fixed-point SIMD datapath for low power wireless applications," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conf. on Graphics Hardware*, Sarajevo, Bosnia-Herzegovina, 2004, pp. 107-114.

[48] H. Kim, B. G. Nam, J. H. Sohn, J. H. Woo, and H. J. Yoo, "A 231-MHz, 2.18-mW 32-bit logarithmic arithmetic unit for fixed-point 3-D graphics system," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 11, pp. 2373-2381, 2006, DOI:10.1109/JSSC.2006.882887 .

[49] N. Otsu, "A threshold selection method for gray-level histograms," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 9, no. 1, pp. 62-66, Jan. 1997, DOI: 10.1109/TSMC.1979.4310076 .

[50] K. V. Asari, T. Srikanthan, S. Kumar, and D. Radhakrishnan, "A pipelined architecture for image segmentation by adaptive progressive thresholding," *Microprocessors and Microsystems*, vol. 23, no. 8, pp. 493-499, Dec. 1999.

[51] H. Tian, S. K. Lam, and T. Srikanthan, "Implementing Otsu's thresholding process using area-time efficient logarithmic approximation unit," in *Proceedings of the 2003 Int'l Symp. on Circuits and Systems(ISCAS'03)*, vol. 4, Bangkok, 2003, pp. IV-21-IV-24.

[52] Xilinx. (2013, Dec.) XPower Analyzer. [Online]. http://www.xilinx.com/products/design_tools/logic_design/verification/xpower_an.htm

[53] C. E. Luna, L. P. Kondi, and A. K. Katsaggelos, "Maximizing user utility in video streaming applications," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13, no. 2, pp. 141-148, Feb. 2003, DOI:10.1109/TCSVT.2002.808439.

[54] S. Khan, Y. Peng, E. Steinbach, M. Sgroi, and W. Kellerer, "Application-driven cross-layer optimization for video streaming over wireless networks," *IEEE Communications Magazine*, vol. 44, no. 1, pp. 122-130, Jan. 2006, DOI:10.1109/MCOM.2006.1580942.

[55] M. Esteve, C. E. Palau, M. N.J., and B. Molina, "A video streaming application for urban traffic management," *Journal of Network and Computer Applications*, vol. 30, no. 2, pp. 479-498, Apr. 2007.

[56] K. Piamrat, C. Viho, J. Bonnin, and A. Ksentini, "Quality of experience measurements for video streaming over wireless networks," in *Proceedings of IEEE 6th Int'l Conf. on Information Technology: New Generations (ITNG '09)*, Las Vegas, NV, 27-29 April 2009, pp. 1184-1189.

[57] S. Arseneau and J. R. Cooperstock, "Real-time image segmentation for action recognition," in *Proceedings of IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing, 1999*, Victoria, BC, Canada, 1999, pp. 86-89.

[58] H. F. Ng, "Automatic thresholding for defect detection," *Pattern Recognition Letters*, vol. 27, no. 14, pp. 1644-1649, Oct. 2006.

[59] P. L. Rosin, "Thresholding for change detection," in *Proceedings of IEEE 6th Int'l Conf. on Computer Vision*, Bombay, India, 4-7 Jan. 1998, pp. 274-279.

[60] M. J. Seow and K. V. Asari, "A parallel VLSI architecture for real-time segmentation of images with complex background environment," in *Proceedings of 10th NASA Symp. on VLSI Design*, Albuquerque, New Mexico, USA, 2002, pp. 1031-1036.

[61] B. Epshtein, E. Ofek, and Y. Wexler, "Detecting text in natural scenes with stroke width transform," in *Proceedings of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, CA, 2010, pp. 2963-2970.

[62] T. Abak, U. Baris, and B. Sankur, "The performance of thresholding algorithms for optical character recognition," in *Proceedings of the Fourth Int'l Conf. on Document Analysis and Recognition (ICDAR'97)*, vol. 2, Ulm, 1997, pp. 697-700.

[63] M. Kamel and A. Zhao, "Extraction of binary character/graphics images from grayscale document images," *CVGIP: Graphical Models and Image Processing*, vol. 55, no. 3, pp. 203-217, May 1993.

[64] W. Jianlai, Y. Chunling, Z. Min, and W. Changhui, "Implementation of Otsu's thresholding process based on FPGA," in *Proceedings of 4th IEEE Conf. on Industrial Electronics and Applications (ICIEA 2009)*, Xi'an, 2009, pp. 479-483.

[65] Y. Zhong, K. Karu, and A. K. Jain, "Locating text in complex color images," in *Proceedings of the 3rd IEEE Int'l Conf. on Document Analysis and Recognition*, vol. 1, Montreal, Que, 14-16 Aug. 1995, pp. 146-149.

[66] A. Pezeshk and R. L. Tutwiler, "Automatic feature extraction and text recognition from scanned topographic maps," *IEEE Trans. on Geoscience and Remote Sensing*, vol. 12, no. 49, pp. 5047-5063, Dec. 2011, DOI:10.1109/TGRS.2011.2157697.

[67] A. Shahab, F. Shafait, and A. Dengel, "ICDAR 2011 Robust Reading Competition Challenge 2: Reading Text in Scene Images," in *Procedding of IEEE Int'l Conf. on Document Analysis and Recognition (ICDAR)*, Beijing, China, 18-21 Sept. 2011, pp. 1491-1496.

[68] R. I. Hammoud, B. R. Abidi, and M. A. Abidi, *Face Biometrics for Personal Identification*. Springer, 2007.

[69] M. Tico and P. Kuosmanen, "An algorithm for fingerprint image postprocessing," in *34th Asilomar Conf. on Signals, Systems and Computers*, vol. 2, Pacific Grove, CA, USA, 29 Oct.-01 Nov. 2000, pp. 1735-1739.

[70] K. K. Sreenivasan, M. Srinath, and A. Khotanzad, "Automated vision system for inspection of IC pads and bonds," *IEEE Trans. on Components, Hybrids, and Manufacturing Technology*, vol. 16, no. 3, pp.

333-338, May 1993, DOI:10.1109/33.232061.

[71] L. Schomaker, M. Bulacu, and K. Franke, "Automatic writer identification using fragmented connected-component contours," in *Ninth Int'l Workshop on Frontiers in Handwriting Recognition(IWFHR-9 2004.)*, Tokyo, Japan, 26-29 Oct. 2004, pp. 185-190.

[72] L. Schomaker and M. Bulacu, "Automatic writer identification using connected-component contours and edge-based features of uppercase Western script," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 787-798, Jun. 2004, DOI:10.1109/TPAMI.2004.18.

[73] H. Yoshida and J. Nappi, "Three-dimensional computer-aided diagnosis scheme for detection of colonic polyps," *IEEE Trans. on Medical Imaging*, vol. 20, no. 12, pp. 1261-1274, Dec. 2001, DOI:10.1109/42.974921.

[74] D. T. Lin, M. C. Lin, and K. Y. Huang, "Real-time automatic recognition of omnidirectional multiple barcodes and DSP implementation," *Machine Vision and Applications*, vol. 22, no. 2, pp. 409-419, 2011.

[75] H. Hedberg, F. Kristensen, and V. Owall, "Implementation of a labeling algorithm based on contour tracing with feature extraction," in *Proceedings of 2007 IEEE Int'l Symp. on Circuits and Systems (ISCAS 2007)*, New Orleans, USA, 27-30 May 2007, pp. 1101-1104.

[76] L. D. Stefano and A. Bulgarelli, "A simple and efficient connected components labeling algorithm," in *Proceedings 1999 Int'l Conf. on Image Analysis and Processing*, Venice, 1999, pp. 322-327.

[77] A. I. Comport, E. Marchand, M. Pressigout, and F. Chaumette, "Real-time markerless tracking for augmented reality: the virtual visual servoing framework," *IEEE Trans. on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 615-628, Jul. 2006, DOI:10.1109/TVCG.2006.78.

[78] F. E. Bunn, "Automated vehicle tracking and service provision system," U.S. Patent Patent 6,240,365, May 29, 2001.

[79] M. Quigley, M. A. Goodrich, S. Griffiths, and A. Eldredge, "Target acquisition, localization, and surveillance using a fixed-wing mini-UAV and gimbaled camera," in *Proceedings of the 2005 IEEE Int'l Conf. on Robotics and Automation(ICRA)*, Barcelona, Spain, 18-22 April, 2005, pp. 2600-2605.

[80] K. Schwerdt and J. L. Crowley, "Robust face tracking using color," in *Proceedings of Fourth IEEE Int'l Conf. on Automatic Face and Gesture Recognition*, Grenoble, 2000, pp. 90-95.

[81] F. Yang and M. Paindavoine, "Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification," *IEEE Trans. on Neural Networks*, vol. 14, no. 5, pp. 1162-1175, Sep. 2003, DOI:10.1109/TNN.2003.816035.

[82] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *Journal of ACM Computing Surveys (CSUR)*, vol. 38, no. 4, pp. 1-45, 2006, DOI:10.1145/1177352.1177355.

[83] U. Ali, M. B. Malik, and K. Munawar, "FPGA/soft-processor based real-time object tracking system," in *2010 VI Southern Programmable Logic Conf. (SPL)*, Sao Carlos, 1-3 April 2009, pp. 33-37.

[84] U. Ali and M. B. Malik, "Hardware/software co-design of a real-time kernel based tracking system," *Journal of Systems Architecture; Special Issue on HW/SW Co-Design: Tools and Applications*, vol. 66, no. 8, pp. 317-326, Aug. 2010.

[85] N. Bellas, S. M. Chai, M. Dwyer, and D. Linzmeier, "FPGA implementation of a license plate recognition SoC using automatically generated streaming accelerators," in *IEEE 20th Int'l Parallel and Distributed Processing Symp. (IPDPS 2006)*, Rhodes Island, 25-29 April 2006, pp. 1-8.

[86] B. Jahne, *Practical Handbook on Image Processing for Scientific and Technical Applications*, 2nd ed.

London: CRC Press, 2004.

[87] Xilinx. (2010) Xilinx Virtex-5 FXT FPGAs. [Online]. http://www.xilinx.com/products/virtex5/fxt.htm

[88] R. Gutierrez and J. Valls, "Low cost hardware implementation of logarithm approximation," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 12, pp. 2326-2330, 2011, DOI: 10.1109/TVLSI.2010.2081387 .

[89] A. L. Bovik, Ed., *Handbook of Image Video Processing*. San Diego, USA: Academic Press, 2000.

[90] F. Morgan, T. Bennett, A. Shearer, and M. Redfern, "An FPGA-based time resolved data acquisition system for astronomical and other applications," in *Proceedings of the Irish Signals and Systems Conf.*, UCD, 2000, pp. 336-341.

[91] Y. Lei, et al., "The platform of image acquisition and processing system based on DSP and FPGA," in *Int'l Conf. on Smart Manufacturing Application, 2008. ICSMA*, Gyeonggi-do, Korea, 2008, pp. 470-473.

[92] H. Hou, W. Zhang, H. D., and Z. T., "Design and realization of real-time image acquisition and display system based on FPGA," in *Mechanical Engineering and Technology*. Berlin Heidelberg: Springer, 2012, pp. 565-573.

[93] F. Dias, F. Berry, J. Serot, and F. V. Marmoiton, "Hardware, design and implementation issues on a FPGA-based smart camera," in *Proceedings of the ACM/IEEE Int'l Conf. on Distributed Smart Cameras*, Vienna, 2007, pp. 20-26.

[94] M. Leeser, S. Miller, and H. Yu, "Smart camera based on reconfigurable hardware enables diverse real-time applications," in *Proceedings of the 12th Annual IEEE Symp. on Field-programmable Custom Computing Machines (FCCM'04)*, 2004, pp. 147-155.

[95] R. J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam, "Image change detection algorithms: a systematic survey," *IEEE Trans. on Image Processing*, vol. 14, no. 3, pp. 294-307, Mar. 2005, DOI:10.1109/TIP.2004.838698.

[96] A. Elmabrouk and A. Aggoun, "Edge detection using local histogram analysis," *Electronics Letters*, vol. 34, no. 12, pp. 1216-1217, Jun. 1998, DOI:10.1049/el:19980851.

[97] T. Theocharides, G. Link, N. Vijaykrishnan, M. J. Irwin, and W. Wolf, "Embedded hardware face detection," in *Proceedings of IEEE 17th Int'l Conf. on VLSI Design*, Mumbai, India, 2004, pp. 133-138.

[98] Xilinx. (2011) LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a). [Online]. http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf.

[99] IDT. (2010) Programmable Clock Generator. Programmable_Clok_V2_593.zip.

[100] A. Devices. (2010) AD9980 high-performance 8-bit display interface. [Online]. http://www.analog.com/en/audiovideoproducts/analoghdmidviinterfaces/ad9980/products/product.html

[101] Chrontel, "CH7301 DVI transmitter," http://www.chrontel.com/products/7301.htm.

[102] Sony, "Sony EVI-D70 PTZ camera," http://www.pro.sony.eu/biz/lang/en/eu/product/ptzcams/evi-d70p/overview.

[103] MyGica. V2V Pro. [Online]. http://www.mygica.com/old/pa/v2vpro.asp

[104] Xilinx. Embedded systems tools reference guide. [Online]. http://www.xilinx.com/tools/sdk.htm

[105] Xilinx. (2010) Multi-port memory controller (DDR/DDR2/SDRAM). [Online].

http://www.xilinx.com/products/ipcenter/mpmc.htm

[106] Xilinx. (2010) Digital clock manager (DCM) module. [Online]. http://www.xilinx.com/support/documentation/ip_documentation/dcm_module.pdf

[107] J. A. Pineiro, "Algorithm and architecture for logarithm, exponential, and powering computation," *IEEE Trans. on Computers*, vol. 53, no. 9, pp. 1085-1096, Sep. 2004.

[108] K. H. Abed and R. E. Siferd, "CMOS VLSI implementation of a low-power logarithmic converter," *IEEE Trans. on Computers*, vol. 52, no. 11, pp. 1421-1433, 2003, DOI:10.1109/TC.2003.1244940.

[109] K. H. Abed and R. E. Siferd, "VLSI implementation of a low-power antilogarithmic converter," *IEEE Trans. on Computers*, vol. 52, no. 9, pp. 1221-1228, 2003, DOI:10.1109/TC.2003.1228517 .

[110] J. V. L. Low, C. C. Jong, J. V. S. Low, T. F. Tay, and C. H. Chang, "A fast and compact circuit for integer square root computation based on Mitchell logarithmic method," in *Proceedings of 2012 IEEE Int'l Symp. on Circuits and Systems (ISCAS)*, Seoul, Korea (South), 2012, pp. 1235-1238.

[111] J. N. Mitchell, "Computer multiplication and division using binary logarithm," *IRE Trans. Computer*, vol. EC-11, pp. 512-517, 1962.

[112] T. B. Juang, S. H. Chen, and H. J. Cheng, "A lower error and ROM-free logarithmic converter for digital signal processing applications," *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol. 56, no. 12, pp. 931-935, 2009, DOI: 10.1109/TCSII.2009.2035270 .

[113] D. De Caro, N. Petra, and A. G. M. Strollo, "Efficient logarithmic converters for digital signal processing applications," *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol. 58, no. 10, pp. 667-671, Oct. 2011.

[114] K. H. Abed and R. E. Siferd, "CMOS VLSI implementation of 16-Bit logarithm and anti-logarithm converters," in *Proceedings of 43rd IEEE Midwest Symp. on Circuits and Systems*, vol. 2, Lansing, MI, USA, 2000, pp. 776-779.

[115] B. Cohen, *VHDL Coding Styles and Methodologies*, 2nd ed. Kluwer Academic Pub., 1999.

[116] M. Combet, H. Zonneveld, and L. Verbeek, "Computation of the base two logarithm of binary numbers," *IEEE Trans. on Electronic Computers*, vol. EC-14, no. 6, pp. 863-867, Dec. 1965, DOI:10.1109/PGEC.1965.264080.

[117] S. Paul, N. Jayakumar, and S. P. Khatri, "A fast hardware approach for approximate, efficient logarithm and antilogarithm computations," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 2, pp. 269-277, Feb. 2009, DOI:10.1109/TVLSI.2008.2003481.

[118] L. Guan, S. Y. Kung, and J. Larsen, *Multimedia Image and Video Processing*. New York: CRC Press, 2001.

[119] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *Journal of Electronic Imaging*, vol. 13, no. 1, pp. 146-168, Jan. 2004.

[120] B. Bhanu, "Automatic target recognition: state of the art survey," *IEEE Trans. on Aerospace and Electronic Systems*, vol. AES-22, no. 4, pp. 364-379, Jul. 1986.

[121] S. G. Wu, et al., "A leaf recognition algorithm for plant classification using probabilistic neural network," in *2007 IEEE Int'l Symp. on Signal Processing and Information Technology*, Giza, Egypt, 15-18 Dec. 2007, pp. 11-16.

[122] G. Cheng, J. C. Huang, C. Zhu, Z. Liu, and L. Cheng, "Perceptual image quality assessment using a geometric structural distortion model," in *Proceedings of 17th IEEE Int'l Conf. on Image Processing*

(ICIP), Hong Kong, 26-29 Sep. 2010, pp. 325-328.

[123] E. Garcia, "Implementing a histogram for image processing applications," *Xcell Journal Online*, vol. 38, pp. 40-46, 2000.

[124] ModelSim. (2011, Jan.) ModelSim. [Online]. http://www.mentor.com/products/fpga/model

[125] W. K. Pratt, *Digital Image Processing*, 3rd ed. John Wiley & Sons Inc., 2001.

[126] A. K. Jain, *Fundamentals of Digital Image Processing*. Pearson Prentice Hall, 2007.

[127] R. C. Gonzalez and R. C. Woods. (2010) Images from digital image processing. [Online]. http://www.imageprocessingplace.com/DIP-3E/dip3e_book_images_downloads.htm

[128] J. J. Hull, et al., "Triggering applications based on a captured text in a mixed media environment," U.S. Patent 7 672(543), Mar. 02, 2010.

[129] J. C. Wu, J. W. Hsieh, and W. S. Chen, "Morphology-based text line extraction," *Machine Vision and Applications*, vol. 19, no. 3, pp. 195-207, 2008.

[130] X. D. Yang, "Design of fast connected components hardware," in *Proceedings of IEEE Computer Society Conf. on Computer Vision and Pattern Recognition Proceedings (CVPR'88)*, Ann Arbor, MI, USA, 1988, pp. 937-944.

[131] M. Michele, F. Tombari, D. Brunelli, L. D. Stefano, and L. Benini, "Multimodal abandoned/removed object detection for low power video surveillance systems," in *Proceedings of IEEE 6th Int'l Conf. on advanced video and signal based surveillance (AVSS'09)*, Genova, Italy, 2009, pp. 188-193.

[132] C. Wolfe, T. C. Graham, and J. A. Pape, "Seeing through the fog: an algorithm for fast and accurate touch detection in optical tabletop surfaces," in *ACM Int'l Conf. on Interactive Tabletops and Surfaces*, Saarbrücken, Germany, 2010, pp. 73-82.

[133] R. M. Haralick, "Some neighborhood operators," in *Real-Time Parallel Computing*, M. Onoe, K. Preston, and A. Rosenfeld, Eds. Springer US, 1981, pp. 11-35, DOI:10.1007/978-1-4684-3893-2_2.

[134] K. Suzuki, I. Horiba, and N. Sugie, "Linear-time connected-component labeling based on sequential local operations," *Computer Vision and Image Understanding*, vol. 89, no. 1, pp. 1-23, 2003.

[135] D. G. Bailey and C. T. Johnston, "Single pass connected components analysis," in *Proceedings of Image and Vision Computing*, vol. 10, New Zealand, Dec. 2007, pp. 282-287.

[136] A. AbuBaker, R. Qahwaji, S. Ipson, and M. Saleh, "One scan connected component labeling technique," in *Proceedings of IEEE Int'l Conf. on Signal Processing and Communications (ICSPC 2007)*, Dubai, 2007, pp. 1283-1286.

[137] K. Appiah, A. Hunter, P. Dickinson, and J. Owens, "A run-length based connected component algorithm for FPGA implementation," in *IEEE Int'l Conf. on ICECE Technology (FPT 2008)*, Taipei, Taiwan, 2008, pp. 177-184.

[138] L. He, Y. Chao, and K. Suzuki, "A run-based two-scan labeling algorithm," *IEEE Trans. on Image Processing*, vol. 17, no. 5, pp. 749-756, 2008.

[139] L. He, Y. Chao, K. Suzuki, and K. Wu, "Fast connected-component labeling," *Pattern Recognition*, vol. 42, no. 9, pp. 1977-1987, 2009.

[140] L. He, Y. Chao, K. Suzuki, and H. Itoh, "A run-based one-scan labeling algorithm," in *Image Analysis and Recognition*, M. Kamel and A. Campilho, Eds. Springer Berlin Heidelberg, 2009, pp. 93-102,

10.1007/978-3-642-02611-9_10.

[141] L. He, Y. Chao, and K. Suzuki, "An efficient first-scan method for label-equivalence-based labeling algorithms," *Pattern Recognition Letters*, vol. 31, no. 1, pp. 28-35, 2010.

[142] L. He, Y. Chao, and K. Suzuki, "Two efficient label-equivalence-based connected-component labeling algorithms for 3-D Binary Images," *IEEE Trans. on Image Processing*, vol. 20, no. 8, pp. 2122-2134, 2011.

[143] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, 2nd ed. San Diego: Elsevier, 1982, vol. 1.

[144] H. Samet and M. Tamminen, "An improved approach to connected component labeling of images," in *Proceedings of Int'l Conf. on Computer Vision And Pattern Recognition(CVPR)*, 1986, pp. 312-318.

[145] M. Dillencourt, H. Samet, and M. Tamminen, "A general approach to connected component labeling for arbitrary image representations," *Journal of the ACM*, vol. 39, no. 2, pp. 253-280, Apr. 1992.

[146] Y. Shima, T. Murakami, M. Koga, H. Yashiro, and H. Fujisawa, "A high-speed algorithm for propagation-type labeling based on block sorting of runs in binary images," in *IEEE 10th Int'l Conf. on Pattern Recognition Proceedings*, vol. 1, Atlantic City, NJ, USA, 1990, pp. 655-658.

[147] Q. Hu, G. Qian, and W. L. Nowinski, "Fast connected-component labeling in three-dimensional binary images based on iterative recursion," *Computer vision and image understanding*, vol. 99, no. 3, pp. 414-434, 2005.

[148] J. K. Clemens, "Optical character recognition for reading machine applications," Ph.D. dissertation, Massachusetts Inst. of Technology, Cambridge, 1965.

[149] F. Chang, C. J. Chen, and C. J. Lu, "A linear-time component-labeling algorithm using contour tracing technique," *Computer Vision and Image Understanding*, vol. 93, no. 2, pp. 206-220, 2004.

[150] C. L. Jackins and S. L. Tanimoto, "Octrees and their use in representing 3D objects," *Computer Graph. Image Process*, vol. 14, no. 3, pp. 249-270, Nov. 1980.

[151] H. Samet and M. Tamminen, "Efficient component labeling of images of arbitrary dimension represented by linear bintrees," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 10, no. 4, pp. 579-586, 1988.

[152] K. Wu, E. Otoo, and A. Shoshani, "Optimizing connected component labeling algorithms," in *Proceedings of SPIE, Medical Imaging 2005: Image Processing*, vol. 5747, San Diego, CA, 2005, pp. 1965-1976.

[153] K. Wu, E. Otoo, and K. Suzuki, "Optimizing two-pass connected-component labeling algorithms," *Pattern Analysis and Applications*, vol. 12, no. 2, pp. 117-135, 2009.

[154] C. Grana, D. Borghesani, and R. Cucchiara, "Optimized block-based connected components labeling with decision trees," *IEEE Trans. on Image Processing*, vol. 19, no. 6, pp. 1596-1609, 2010.

[155] L. W. Tucker, "Labeling connected components on a massively parallel tree machine," in *Proceedings of IEEE Conf. Computer Vision and Pattern Recognition*, Miami, FL, 1986, pp. 124-129.

[156] M. Manohar and H. K. Ramapriyan, "Connected component labeling of binary images on a mesh connected massively parallel processor," *Computer Vision, Graphics, and Image Processing*, vol. 45, no. 2, pp. 133-149, 1989.

[157] H. M. Alnuweiri and V. K. Prasanna, "Parallel architectures and algorithms for image component labeling," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 14, no. 10, pp. 1014-1034,

1992.

[158] SIDBA. (2012, Jan.) Standard image database SIDBA. [Online]. http://vision.kuee.kyoto-u.ac.jp/IUE/IMAGE_DATABASE/STD_IMAGES/

[159] USC-SIPI. (2012) The USC-SIPI image of the University of Southern California. [Online]. http://sipi.usc.edu/database/database

[160] A. I. Comport, E. Marchand, M. Pressigout, and F. Chaumette, "Real-time markerless tracking for augmented reality: the virtual visual servoing framework," *IEEE Trans. on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 615-628, Jul. 2006, DOI:10.1109/TVCG.2006.78.

[161] F. Yang and M. Paindavoine, "Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification," *IEEE Trans. on Neural Networks*, vol. 14, no. 5, Sept. 2003, DOI:10.1109/TNN.2003.816035.

[162] K. M. Cheung, S. Baker, and T. Kanade, "Shape-from-silhouette across time part II: applications to human modeling and markerless motion tracking," *Int'l Journal of Computer Vision*, vol. 63, no. 3, pp. 225-245, Jul. 2005.

[163] A. Yilmaz, X. Li, and M. Shah, "Contour-based object tracking with occlusion handling in video acquired using mobile cameras," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1531-1536, Nov..

[164] J. Shi and C. Tomasi, "Good features to track," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '94)*, Seattle, WA, 21-23 Jun 1994, pp. 593-600.

[165] K. Nummiaroa, E. K. Meier, and L. V. Gool, "An adaptive color-based particle filter," *Image and Vision Computing*, vol. 21, no. 1, pp. 99-110, Jan. 2003.

[166] C. Yang, R. Duraiswami, and L. Davis, "Fast multiple object tracking via a hierarchical particle filter," in *Proceedings of IEEE 10th Int'l Conf. on Computer Vision (ICCV 2005)*, vol. 1, Beijing, 17-21 Oct. 2005, pp. 212-219.

[167] I. Mikic, S. Krucinski, and J. D. Thomas, "Segmentation and tracking in echocardiographic sequences: active contours guided by optical flow estimates," *IEEE Trans. on Medical Imaging*, vol. 17, no. 2, pp. 274-284, Apr. 1998, DOI:10.1109/42.700739.

[168] M. J. Black and D. A. Jepson, "EigenTracking: robust matching and tracking of articulated objects using a view-based representation," *Int'l Journal of Computer Vision*, vol. 26, no. 1, pp. 63-84, Jan. 1998.

[169] J. G. Allen, R. Y. D. X., and J. S. J., "Object tracking using CamShift algorithm and multiple quantized feature spaces," in *Proceedings of the Pan-Sydney Area Workshop on Visual Information Processing*, Australia, 2004, pp. 3-7.

[170] Z. Wang, X. Yang, Y. Xu, and S. Yu, "CamShift guided particle filter for visual tracking," *Pattern Recognition Letters*, vol. 30, no. 4, pp. 407-413, Mar. 2009.

[171] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603-619, May 2002, DOI:10.1109/34.1000236.

[172] D. B. K. Trieu and T. Maruyama, "An implementation of the mean shift filter on FPGA," in *IEEE Int'l Conf. on Field Programmable Logic and Applications (FPL)*, Chania, Greece, 5-7 Sept. 2011, pp. 219-224.

[173] D. B. K. Trieu and T. Maruyama, "Real-time color image segmentation based on mean shift algorithm using an FPGA," *Journal of Real-Time Image Processing*, pp. 1-12, Jan. 2013, DOI:10.1007/s11554-

012-0319-9.

[174] B. Gorry, Z. Chen, K. Hammond, A. Wallace, and G. Michaelson, "Using mean-shift tracking algorithms for real-time tracking of moving images on an autonomous vehicle testbed platform," *Int'l Journal of Computer Science & Engineering*, pp. 165-170, 2007.

[175] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd ed. Academic Press, 1990.

[176] F. J. Aherne, N. A. Thacker, and P. I. Rockett, "The Bhattacharyya metric as an absolute similarity measure for frequency coded data," *Kybernetika*, vol. 34, no. 4, pp. 363-368, 1998.

[177] S. H. Cha, "Comprehensive survey on distance/similarity measures between probability density functions," *Int'l Journal of Mathematical Models and Methods in Applied Sciences*, vol. 4, no. 1, pp. 300-307, 2007.

[178] H. C. van Assen, M. Egmont-Petersen, and J. H. C. Reiber, "Accurate object localization in gray level images using the center of gravity measure: accuracy versus precision," *IEEE Trans. on Image Processing*, vol. 11, no. 12, pp. 1379-1384, Dec. 2002, DOI:10.1109/TIP.2002.806250.

[179] X. Dai and S. Khorram, "A feature-based image registration algorithm using improved chain-code representation combined with invariant moments," *IEEE Trans. on Geoscience and Remote Sensing*, vol. 37, no. 5, pp. 2351-2362, Sep. 1999, DOI:10.1109/36.789634.

[180] H. Kato and M. Billinghurst, "Marker tracking and HMD calibration for a video-based augmented reality conferencing system," in *Proceedings of 2nd IEEE and ACM Int'l Workshop on Augmented Reality (IWAR '99)*, San Francisco, CA, 20-21 Oct. 1999, pp. 85-94.

[181] C. S. Sharp, O. Shakernia, and S. S. Sastry, "A vision system for landing an unmanned aerial vehicle," in *Proceedings of IEEE Int'l Conf. on Robotics and Automation (ICRA)*, vol. 2, Seoul, South Korea, 21-26 May 2001, pp. 1720-1727.

[182] S. Saripalli, J. F. Montgomery, and G. Sukhatme, "Vision-based autonomous landing of an unmanned aerial vehicle," in *Proceedings of IEEE Int'l Conf. on Robotics and Automation (ICRA '02)*, vol. 4, Washington, DC, 11-15 May 2002, pp. 2799-2804.

[183] G. Bradski and A. Kaehler, *Learning OpenCV Computer Vision with the OpenCv Library*, 1st ed. USA: O'Reilly, 2008.

[184] GNU. (2010, Apr.) GCC, the GNU Compiler Collection. [Online]. http://gcc.gnu.org/

[185] J. Fenlason. (2010, Jun.) GNU gprof. [Online]. http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html

[186] G. Valgrind. (2011) Valgrind . [Online]. http://valgrind.org/

[187] Xilinx, "UG347 ML505/ML506/ML507 Evaluation Platform," User Guide.

[188] G. Hawkes, "I2C video peripheral loader," Xilinx Application note: microBlaze and multimedia development board XAPP293(0.5), Dec. 2001.

[189] Xilinx. (2010) XPS IIC bus interface. [Online]. http://www.xilinx.com/products/intellectual-property/xps_iic.htm

[190] Micron, "Micron DDR2 SDRAM SODIMM," http://www.micron.com/products/ProductDetails.html?product=produpro/dram_modules/sodimm/MT4HTF3264HY-667.

[191] IDT. (2010, Jan.) 3.3V EEPROM Programmable Clock generator. [Online]. http://www.idt.com/products/clocks-timing/clock-generators-synthesizers-and-zero-delay-

buffers/general-purpose-clock-generators-synthesizers-and-zero-delay-buffers/5v9885t-33v-eeprom-programmable-clock-generator

[192] R. Sass and A. G. Schmidt, *Embedded Systems Design with Platform FPGAs Principles and Practices*. Morgan Kaufmann Publishers, 2010.

LIST OF PUBLICATIONS

Conference Publications:

(i) J. G. Pandey, A. Karmakar, A. K. Mishra, C. Shekhar, and S. Gurunarayanan, "Implemention of an improved connected component labeling algorithm using FPGA-based platform," *To be presented in IEEE Int'l Conf. on Signal Processing and Communications* (*SPCOM 2014*), IISc-Bangalore, India, 22-25 Jul. 2014.

(ii) J. G. Pandey, A. Karmakar, C. Shekhar, and S. Gurunarayanan, "An FPGA-based architecture for kernel-smoothed local histogram computation," *To be presented in IEEE Int'l Symposium on Circuits and Systems (ISCAS-2014)*, Melbourne, Australia, 01-05 Jun., 2014.

(iii) J. G. Pandey, A. Karmakar, C. Shekhar, and S. Gurunarayanan, "A novel architecture for FPGA implementation of Otsu's global automatic image thresholding algorithm," in *Proceedings of IEEE 27th Int'l Conf. on VLSI Design and 13th Int'l Conf. on Embedded Systems* (*VLSI Design 2014*), Mumbai, India, 5-9 Jan. 2014, pp. 300-305.

(iv) J. G. Pandey, A. Karmakar, C. Shekhar, and S. Gurunarayanan, "An FPGA-based novel architecture for the fixed-point binary antilogarithmic computation," in *Proceedings of IEEE Int'l Conf. on Electronic Systems, Signal Processing and Computing Technologies (ICESC)*, Nagpur, India, 09-11 Jan. 2014, pp. 23-28.

(v) J. G. Pandey, A. Karmakar, C. Shekhar, and S. Gurunarayanan, "An FPGA-based fixed-point architecture for binary logarithmic computation," in *Proceedings of 2nd IEEE Int'l Conf. in Image Information Processing (ICIIP-2013)*, Shimla, India, 09-12 Dec. 2013, pp. 383-388.

(vi) J. G. Pandey, A. Karmakar, and C. Shekhar, "Platform-based design approach for implementing real-time image and video processing applications," in *Proceedings of IEEE 4th Int'l Conf. on Electronics Computer Technology (ICECT)*, Kanyakumari, India, 6-8 Apr. 2012, pp. 488-490.

(vii) J. G. Pandey, A. Karmakar, and C. Shekhar, "An embedded architecture for implementation of a video acquisition module of a smart camera system," in *Proceedings of IEEE Int'l Conf. on Devices, Circuits and Systems (ICDCS)*, Coimbatore, India, 15-16 Mar. 2012, pp. 191-194.

(viii) J. G. Pandey, A. S. Mandal, S. Purushottam, and C. Shekhar, "Platform-based design approach for video processing in a smart camera system," in *Proceedings of 3rd IEEE Int'l Conf. on Computer Modeling and Simulation* (*ICCMS 2011*), Mumbai, India, 07-09 Jan. 2011, pp. 373-376.

Journal Publications:

(i) J. G. Pandey, S. Purushottam, A. Karmakar, and C. Shekhar, "*Platform-based extensible hardware-software video streaming module for a smart camera system*," *Int'l Journal of Modeling and Optimization*, vol. 2, no. 4, pp. 482-487, Aug. 2012, DOI:10.7763/IJMO.2012.V2.167.

(ii) J. G. Pandey, A. Karmakar, C. Shekhar, and S. Gurunarayanan, "Platform-based design approach for embedded vision applications," *Journal of Image and Graphics*, vol. 1, no. 1, pp. 1-6, Mar. 2013, DOI:10.12720/joig.1.1.1-6.

BRIEF BIOGRAPHY OF THE CANDIDATE

Jai Gopal Pandey was born in the city of Gorakhpur, India in 1979. He received Master of Science (Electronics) degree from D. D. U. Gorakhpur University, India in 2001 and Master of Technology degree in Electronics Design and Technology, with specialization in VLSI Design from U. P. Technical University, Lucknow, India, in 2003. He is with Council of Scientific and Industrial Research - Central Electronics Engineering Research Institute (CSIR-CEERI), Pilani, Rajasthan-333031, India since 2005. He is working as a Scientist with IC Design Group, CSIR-CEERI, Pilani, in the area of real-time image/video processing with FPGA platform. His research interest includes VLSI design and high-performance computer architecture for embedded system applications. Mr. Pandey is a member of IEEE, IACSIT, and life member of Semiconductor Society of India.

BRIEF BIOGRAPHY OF THE SUPERVISORS

Dr. Chandra Shekhar was born in India, in 1951. He received M.Sc. degree in Physics in 1971 and Ph.D. degree in 1975, from BITS, Pilani, India. He is with Council of Scientific and Industrial Research - Central Electronics Engineering Research Institute (CSIR-CEERI), Pilani, Rajasthan-333031, India since 1977. He is currently serving as the Director of CSIR-CEERI, Pilani, since 2003. His research interests include analog and mixed signal design, VLSI design and design methodologies, application specific processor design, CAD for VLSI, and Physics and modeling of MOS Devices. He has published several research papers in various reputed international/national journals and conferences. Dr. Shekhar is a Fellow of IETE and life member of Indian Physics Association, Semiconductor Society of India and Indo-French Technical Association.

Dr. Abhijit Karmakar was born in West Bengal, India, in 1971. He received the B.E. degree in electronics and telecommunication engineering in 1993 from Jadavpur University, India, the M.Tech. degree in electrical engineering from the Indian Institute of Technology (IIT), Madras, India, in 1995, and the Ph.D. degree in electrical engineering from IIT Delhi in 2007. He is with Council of Scientific and Industrial Research - Central Electronics Engineering Research Institute (CSIR-CEERI), Pilani, Rajasthan-333031, India since 1995. His research interests are in the areas of VLSI design, digital signal processing, auditory modeling, and speech quality evaluation. Dr. Karmakar is a member of IEEE, fellow of IETE and life member of Semiconductor Society of India.

Prof. S. Gurunarayanan was born in India. He received his M.Sc. degree in Physics from Alagappa University, Karaikudi, India in the year of 1987. He received his M. E. Degree in systems and information in 1990 and Ph.D. degree in engineering in the year 2000 from BITS, Pilani, India. He is with BITS, Pilani, India since 1987. There, he is working as a

professor since 2005. Dr. Gurunarayanan is currently serving as Dean, Work Integrated Learning Programmes Division, in BITS, Pilani, Rajasthan, India. Dr. Gurunarayanan's research interest includes VLSI design, digital design and computer organization, VLSI Architecture, embedded system design.

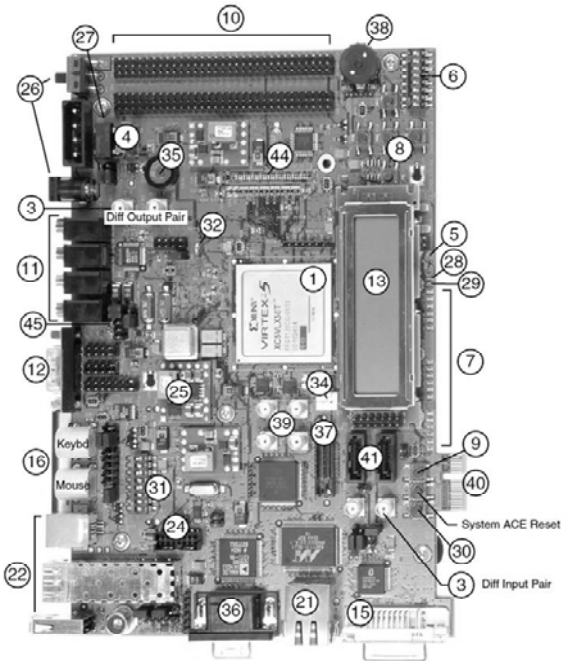
APPENDIX A

AN OVERVIEW OF THE FPGA-BASED PLATFORM

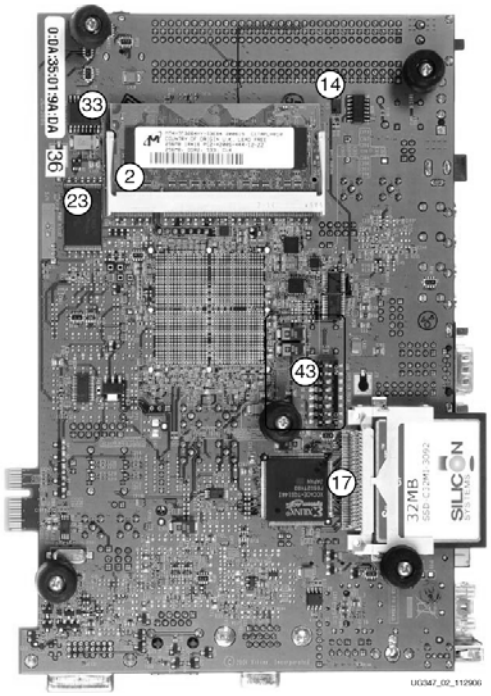
A.1 Xilinx ML-507 FPGA Platform

The Xilinx ML-507 platform [**187**] contains a Virtex-5 FPGA (XC5VFX70T) which has one PowerPC440 (PPC440) processor surrounded by the FPGA fabric [**33**]. The software tasks can be executed on the PPC440 processor, while the FPGA is used for hardware acceleration. Fig. A.1 shows the organization of different components of the platform [**33**]. Some of the important features of this platform are follows:

- PPC 440 embedded reduced instruction set computing (RISC) processor [**34**].
- Memory controller interface (MCI); provides an interface between PPC440 processor and a soft memory controller implemented in the FPGA logic [**34**].
- Processor local bus (PLB) is a 128-bit bus, which provides bus infrastructure for connecting an optional number of PLB masters and slaves into an overall PLB system [**98**].
- VGA input video codec connector; supports connectivity to an external VGA source [**100**].
- Inter-integrated circuit (I2C) bus support [**188**,**189**].
- 64-bit wide, 256 MB Micron MT4HTF3264HY-667 DDR2 small outline dual in-line memory module (SODIMM) [**190**]; acts as a video frame buffer for the image acquisition system.
- 11,200 configurable logic block (CLB) slices; provided for implementing combinational and sequential logic functions [**32**].



(a)



(b)

1. Virtex-5 FPGA.
2. 256 MB DDR2 SODIMM.
3. Differential clock I/O with SMA connectors.
4. Oscillators.
5. LCD brightness and contrast adjustment.
6. GPIO DIP switches (Active-High).
7. User and error LEDs (Active-High).
8. User pushbuttons (Active-High).
9. CPU reset button (Active-Low).
10. XGI expansion headers.
11. Stereo AC97 audio codec.
12. RS-232 serial port.
13. 16-character x 2-line LCD.
14. I2C bus with 8-Kb EEPROM.
15. DVI connector.
16. PS/2 mouse and keyboard ports.
17. System ACE and CompactFlash connector.
18. ZBT synchronous SRAM.
19. Linear flash chips.
20. Xilinx XC95144XL CPLD.
21. 10/100/1000 Tri-speed ethernet PHY.
22. USB controller with host and peripheral ports.
23. Xilinx XCF32P platform flash PROM.
24. JTAG configuration port.
25. Onboard power supplies.
26. AC adapter and input power switch/jack.
27. Power indicator LED.
28. DONE LED.
29. INIT LED.
30. Program switch.
31. Configuration address and mode DIP switches.
32. Encryption key battery.
33. SPI flash.
34. I2C fan controller and temp./voltage monitor.
35. Piezo.
36. VGA input video codec.
37. JTAG trace/debug.
38. Rotary encoder
39. Differential GTP/GTX I/O with SMA connectors.
40. PCI express interface.
41. Serial-ATA host connectors.
42. SFP connector.
43. GTP/GTX clocking circuitry.
44. Soft touch landing pad.
45. System monitor.

Fig. A.1: Xilinx ML-507 Platform. (a) front view (b) rear view.

- Programmable system clock generator chip; available for generating a variety of non-integer clocks from 4.9 KHz to 500 MHz to the platform peripherals and FPGA [**191**,**99**].
- Digital clock manager (DCM); provides integer multiple of clocks to various peripherals [**106**].
- Multi-port memory-controller (MPMC); for external memory support [**105**].
- 5328 Kb Block RAMs (BRAMs); available as configurable internal RAM for the FPGA.
- RS-232 serial port; allows the FPGA to communicate serial data with another device or with PC.
- JTAG configuration port.
- DVI connector with display controller chip; to support an external DVI/VGA monitor [**101**].

A.2 Field-Programmable Gate Array (FPGA) Device

The field-programmable gate array (FPGA) is a semiconductor device. It is based on a matrix of configurable logic blocks (CLBs) connected through programmable interconnects. Instead of being restricted to any predetermined hardware function (as in ASIC, where the device is custom built for the particular design), an FPGA can be programmed to modify the product features and functions to reconfigure hardware for specific applications. This modification in the hardware can be done even after the product is installed in the field and that makes it a "field-programmable" device [**192**]. The FPGAs can be used to implement any logical function that an ASIC circuit (ASIC) can perform. Today's FPGAs can be partially or fully re-configurable to implement a desired logic function. The partial re-configuration feature allows a portion of the FPGA to be always running, while another portion of the same FPGA is being re-configured for the new set of logic functionality.

The previous generation FPGAs used I/Os with programmable logic and interconnects. The modern FPGAs consist of configurable embedded memory, high-speed transceivers, high-speed I/Os, logic blocks. FPGAs have evolved far beyond the basic capabilities present in their predecessors, and now incorporate pre-fabricated blocks of intellectual property (IP) of commonly used functionality such as Block RAM (single/dual port), clock management, and DSP (multiplier, adder, arithmetic-logic unit). Intellectual property (IP) blocks built into the FPGA fabric provide rich functions while lowering power and cost. Apart from this, the FPGA families are available which contain hard-embedded processor(s), transforming the devices into systems on a chip (SoC) [**23**].

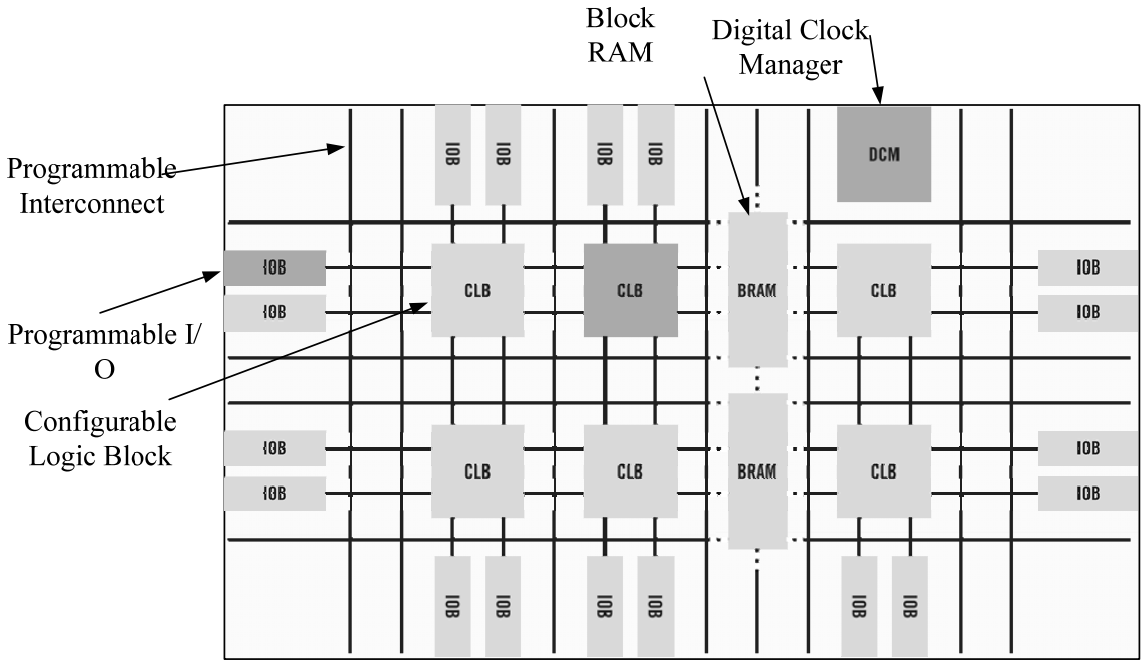


Fig. A.2: FPGA block structure (reproduced from embedded processor block in Virtex-5 FPGAs).

The block structure of an FPGA is shown in Fig. A.2 [**32**]. The main components of the FPGAs are: CLBs, BRAMs, programmable interconnects, programmable I/O, digital clock manager. As discussed earlier, the present day FPGAs also contain hard IP blocks like an

embedded processor and DSP slices (multiplier, adder). The details of each block are given below.

A.2.1 Configuration Logic Block (CLB)

The CLB is the basic logic unit of FPGAs. The number of CLBs and CLB features vary from device to device, but every CLB consists of a configurable switch matrix with 4 or 6 inputs, some selection circuitry (MUX, etc.), and flip-flops. The switch matrix is flexible and can be configured to implement combinatorial logic and shift register or RAM bits.

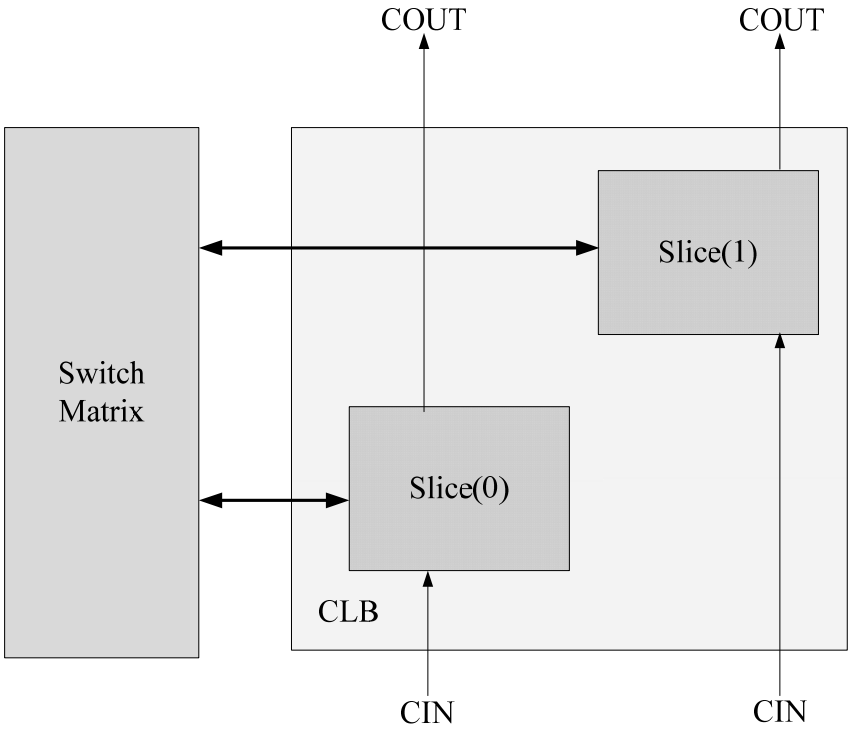


Fig. A.3: Arrangement of slices within the CLB.

User writes into the configuration memory, which defines function of the system. This includes the connectivity between the CLBs and the I/O cells, the logic to be implemented onto the CLBs, and the configuration of the I/O blocks. By changing data in the configuration memory, the function of the system can be changed. This change in data can be implemented at anytime during the FPGA operation (run-time configuration). In Xilinx Virtex-5 FPGA, a

CLB element contains a pair of independent slices, slice (0) and slice (1) as shown in Fig. A.3.

These slices are organized in a column and contain carry chains. The Xilinx tools assign slices as follows: X followed by a number identifies the position of each slice in a pair as well as the column position of the slice. The X number counts slices starting from the bottom in sequence 0, 1 (the first CLB column); 2, 3 (the second CLB column); etc. A Y followed by a number categorizes a row of slices. The number remains the same within a CLB, but increases in sequence from one CLB row to the next CLB row, starting from the bottom. The detail of the slice is given below.

A.2.2 Slice Description

Each slice includes four logic-function generators or look-up tables (six input LUTs), four storage elements, wide-function multiplexers, and carry logic. By these resources, the slice provides logic, arithmetic, and ROM functions. In addition to this, some slices support two extra functions: storing data using distributed RAM and shifting data with registers. Slices that support these additional functions are known as SLICEM; others are called SLICEL [**32**]. An arrangement of SLICEL is shown in Fig. A.4 and the SLICEM is shown in Fig. A.5 [**32**]

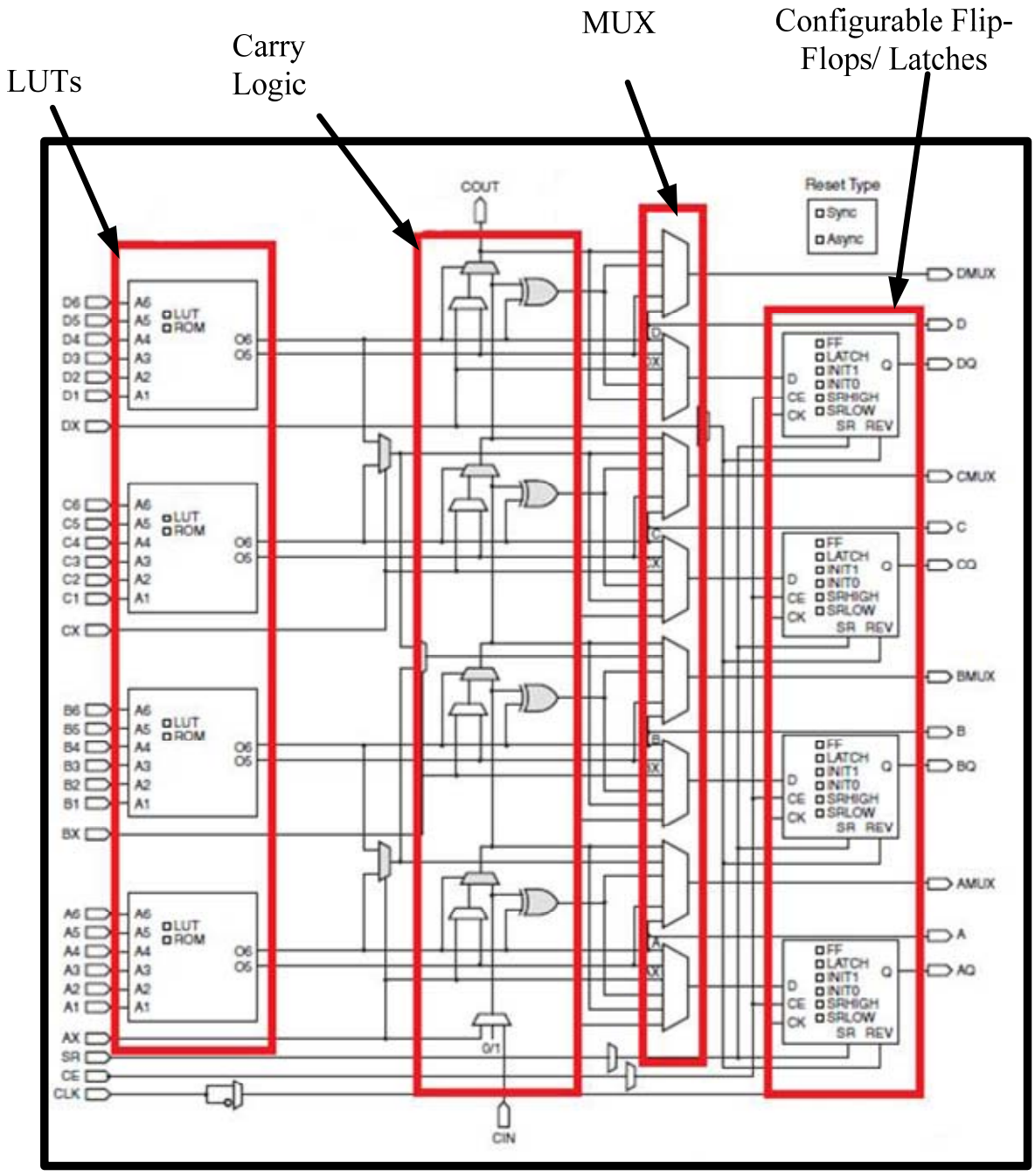


Fig. A.4: Arrangement of SLICEL.

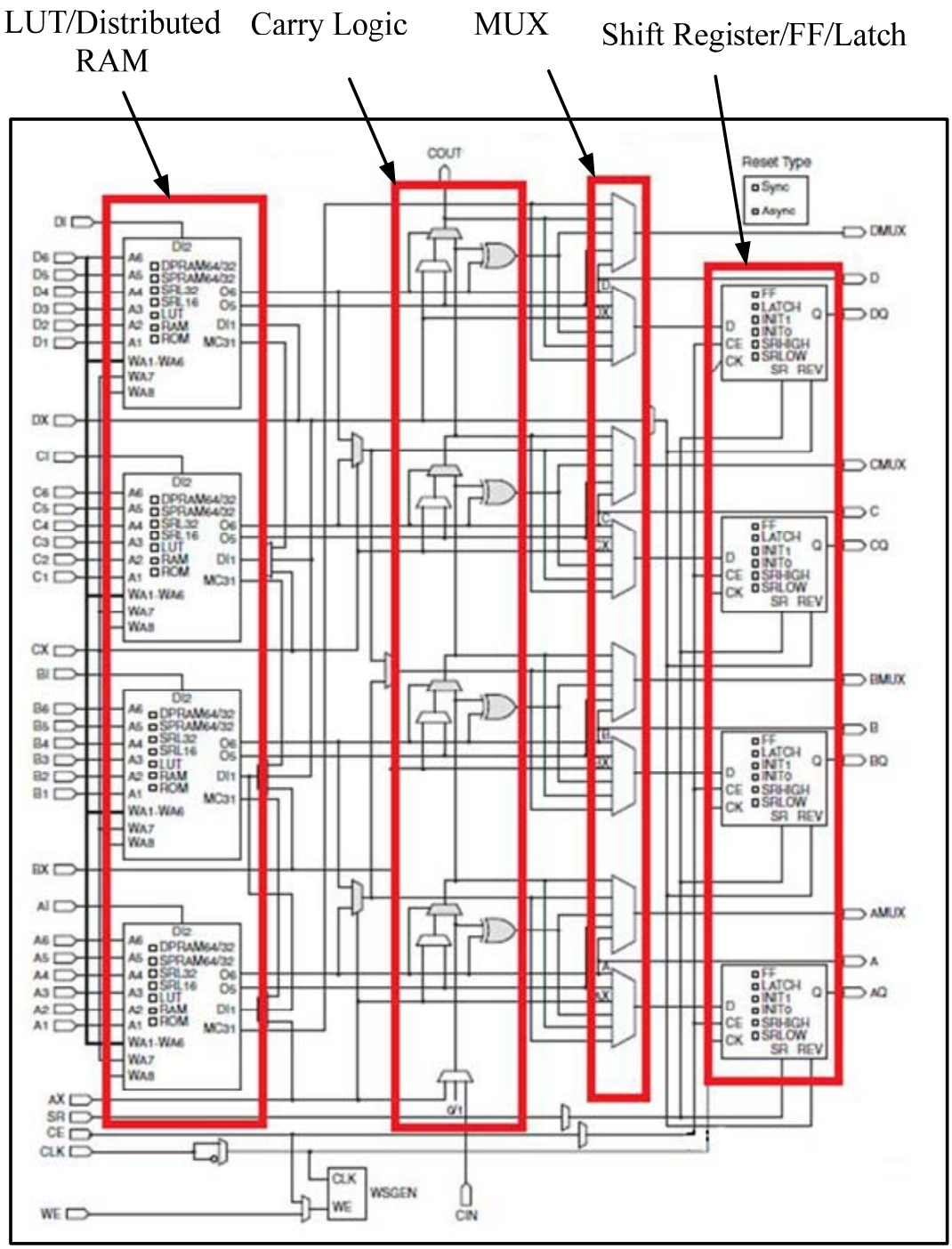


Fig. A.5: Arrangement of SLICEM.

A.2.3 Interconnect

While the CLB provides the logic capability, flexible interconnect routing routes the signals between CLBs and to/from I/Os. Routing comes in several ways, from that designed to interconnect between CLBs to fast horizontal and vertical long lines spanning the device, to global low-skew routing for clocking and other global signals [**32**].

A.2.4 Select I/O

The basic IOB and its connections to the internal logic and the device pad are shown in Fig. A.6 [**32**].

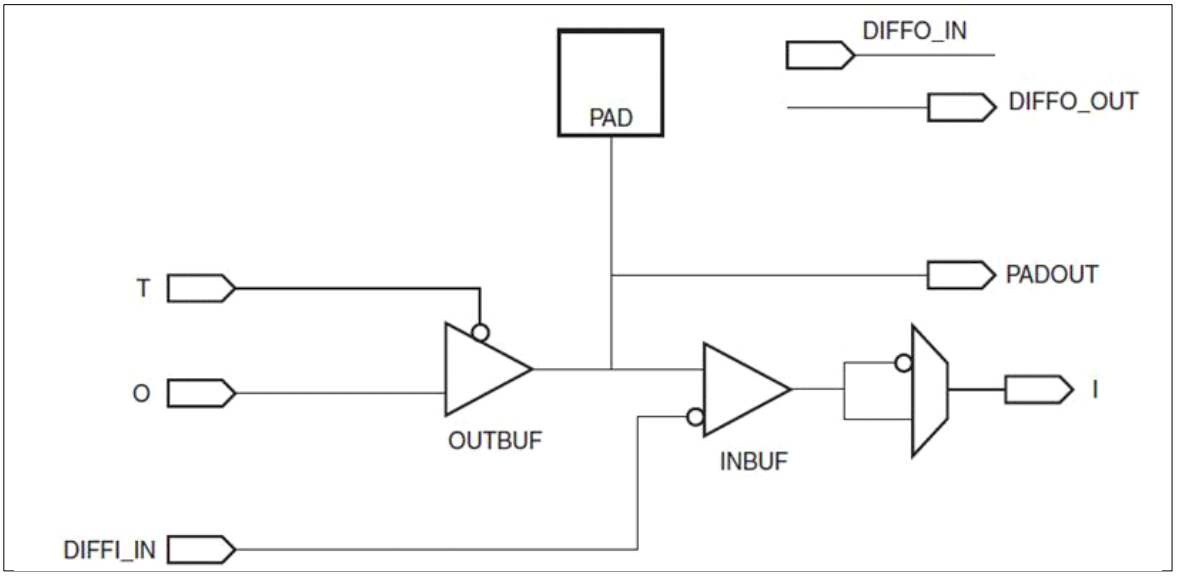


Fig. A.6: Arrangement of IOB.

FPGAs provide support for dozens of I/O standards thus providing the ideal interface bridge in the system. I/O in FPGAs is grouped in banks with each bank independently able to support different I/O standards [**32**]. Each IOB contains input, output, and 3-state SelectIO drivers. These drivers can be configured to various I/O standards. Differential I/O uses the two IOBs grouped together in one tile.

- Single-ended I/O standards (LVCMOS, LVTTL, HSTL, SSTL, GTL, PCI)

- Differential I/O standards (LVDS, HT, LVPECL, BLVDS, Differential HSTL and SSTL)
- Differential and VREF dependent inputs are powered by VCCAUX

Each Virtex-5 FPGA I/O tile contains two IOBs, and also two ILOGIC blocks and two OLOGIC blocks.

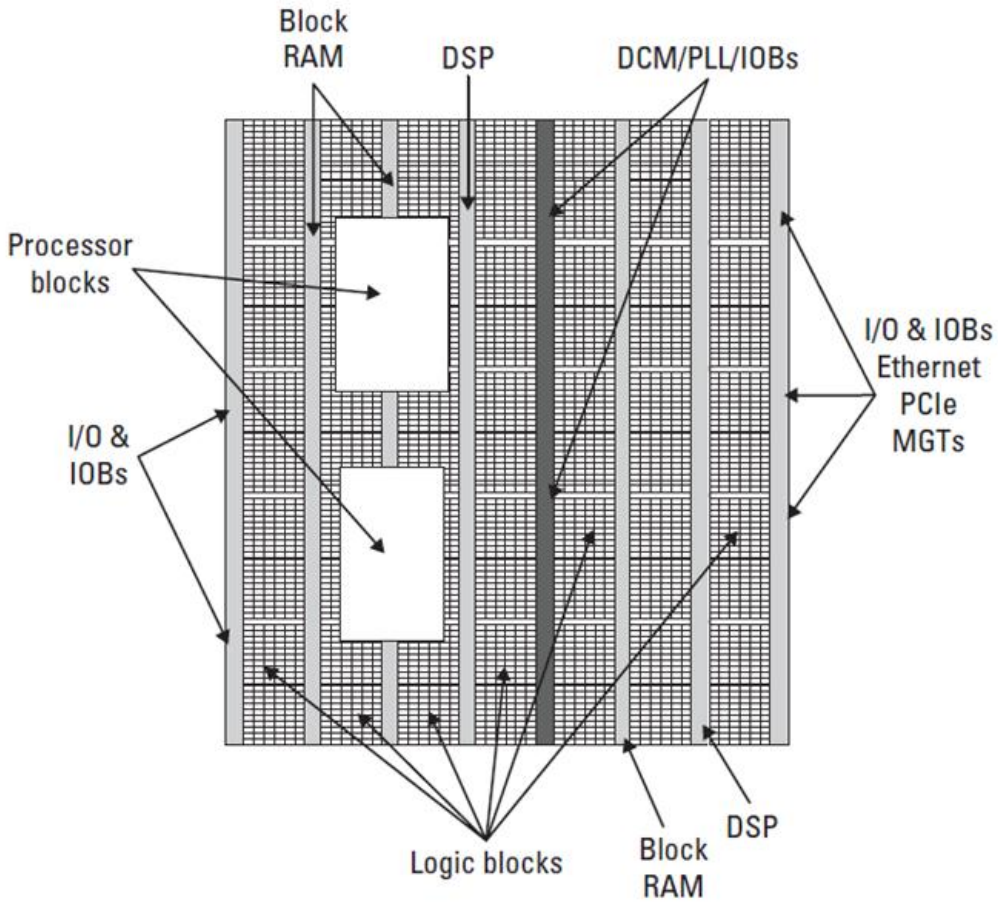


Fig. A.7: Modern FPGA device.

A.2.5 Special-Purpose Function Blocks

A large portion of the FPGA consists of logic blocks and routing logic to connect the programmable logic. Today's FPGA combine programmable logic with additional resources that are embedded into the fabric of the FPGA. The block diagram of a modern FPGA is shown in Fig. A.7 [**192**]. It shows the arrangement of special-purpose function blocks placed

throughout the FPGA. The logic blocks still occupy a majority of the FPGA fabric in order to support a variety of complex digital designs.

A.2.5.1 Digital Clock Manager (DCM)

Sometimes there are needs of different clock frequencies, as different logic cores can operate at different frequencies. A digital clock manager (DCM) allows different clock periods to be generated from a single reference clock. Digital clock management is provided by most FPGAs (all Xilinx FPGAs have this feature). The most advanced FPGAs from Xilinx offer both digital clock management and phase-looped locking that provide precision clock synthesis combined with jitter reduction and filtering [**192**].

A.2.5.2 Block RAM

Designers require the use of some amount of on-chip memory. Using logic cells it is possible to build variable-sized memory elements; however, as the amount of memory needed increases, these resources are quickly consumed. The solution is to provide a fixed amount of on-chip memory embedded into the FPGA fabric called Block RAM (BRAM). The amount of memory depends on the device.

In Virtex-5 FPGAs BRAM stores up to 36K bits of data and can be configured as either two independent 18 Kb RAMs, or one 36 Kb RAM. Each 36 Kb Block RAM can be configured as a 64K x 1 (when cascaded with an adjacent 36 Kb block RAM), 32K x 1, 16K x 2, 8K x 4, 4K x 9, 2K x 18, or 1K x 36 memory. Each 18 Kb block RAM can be configured as a 16K x 1, 8K x2 , 4K x 4, 2K x 9, or 1K x 18 memory [**32**].

A.2.5.3 Digital Signal Processing (DSP) Blocks

Complex designs may consist of either digital signal processing (DSP) or just some variety of multiplication, addition, and subtraction. It is possible to combine DSP blocks to perform larger operations, such as single and double precision floating point addition, subtraction,

multiplication, division, and square root. The number of DSP blocks is device dependent; however, they are typically located near the BRAMs, which is useful when implementing processing requiring input and/or output buffers [**23**], [**192**]. In Xilinx Virtex-5 FPGAs the DSP (DSP48E) slice resources contain a 25x18 two's complement multiplier and a 48-bit adder/subtacter/accumulator. Each DSP48E slice also contains extensive cascade capability to efficiently implement high-speed DSP algorithms [**32**].

A.2.5.4 Embedded Processor

One of the most important additions to the FPGA fabric is a processor (one or two processors) embedded within the FPGAs fabric, such as the FX series in Xilinx Virtex-5 FPGAs [**32**]. The availability of an embedded processor can simplify the design process significantly, while reducing resource usage and power consumption. The IBM PowerPC440 (PPC440) processor is the processor included in the Xilinx Virtex-5 FX FPGAs device [**32**].

A.3 FPGA Configuration Options

The Virtex-5 FPGA device on the ML-507 Platform can be configured by following ways [**33**].

- Xilinx download cable (JTAG)
- System ACE controller (JTAG)
- Two platform flash PROMs
- Serial peripheral interface (SPI) flash memory
- Linear flash memory

Following section provides an overviews of the possible means through which FPGAs can be configured.

A.3.1 JTAG (Xilinx Download Cable and System ACE Controller) Configuration

The JTAG port is used to configure the main devices of the board like FPGA, two Platform Flash PROMs, and CPLD. Fig. A.8 shows the JTAG chain of the board. The chain starts at the PC4 connector and goes through the Platform Flash PROMs, the CPLD, the System ACE controller, the FPGA, and an optional extension of the chain to the expansion card. Jumper J21 is used for the JTAG chain extension to the expansion card. The JTAG chain is utilized to program the Virtex-5 FPGA device and access the FPGA for hardware and software debug.

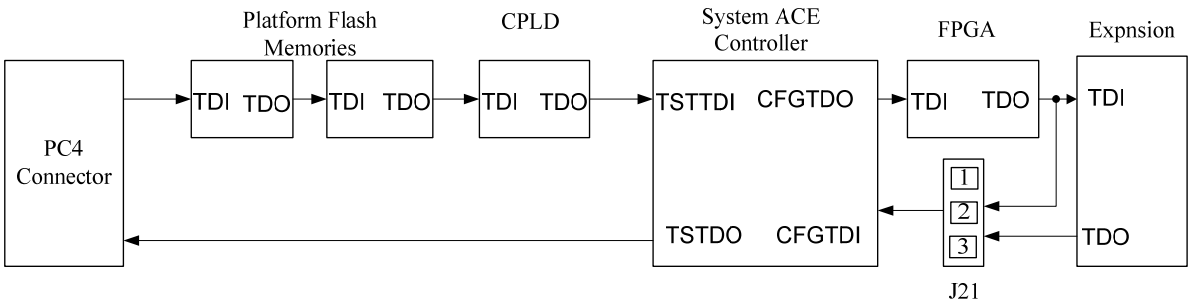


Fig. A.8: FPGA configurations.

The PC4 JTAG connection to the JTAG chain allows a host computer to download bit streams to the FPGA using the Xilinx iMPACT software tool. The PC4 JTAG connection also permits debug tools like the Xilinx ChipScope Pro analyzer or a software debugger to access the FPGA device.

The system ACE controller can also be utilized to program the FPGA through the JTAG port. The configuration information can be stored for the FPGA by using a compact flash card, which supports up to eight configuration images. The images can be selected by using the three configuration address dual in-line package (DIP) switches. The FPGA controls, the system ACE chip to reconfigure to any of the eight configuration images.

A.3.2 Platform Flash PROM Configuration

The FPGA device can also be programmed by utilizing a platform flash PROMs. A platform flash PROM can hold up to two configuration images (up to four with compression), which are selectable by the two least significant bits of the configuration address DIP switches. The board is wired so the platform flash PROM can download bitstreams in master serial, slave serial, master parallel, or slave parallel modes. The Xilinx iMPACT tool is used to program the platform flash PROM.

A.3.3 Linear Flash Memory Configuration

Data stored in the linear flash can also be used to program the FPGA (BPI mode). Up to four configuration images can be supported.

A.3.4 SPI Flash Memory Configuration

Data stored in SPI can be used to program the FPGA. The FPGA device is programmed upon power-up or whenever the Program button is pressed.

A.4 PowerPC 440 Embedded Processor

The Virtex 5 FX series FPGAs include one or two PowerPC 440 processors embedded within the FPGA fabric [**32**], [**34**] . The PowerPC 440 is a dual-issue, superscalar RISC processor with an operating frequency of up to 550 MHz. It contains seven-stage pipeline with out-of-order execution capabilities. Each comes with 32 KB, 64-way set associative level-1 instruction and data cache and a memory management unit (MMU) with a translation look-aside buffer (TLB) to support virtual memory. In addition to three separate 128-bit processor local bus (PLB) interfaces, the embedded processor provides interfaces for custom coprocessors and floating-point functions [**34**]. The block diagram of the PowerPC 440 embedded processor is shown in Fig. A.9.

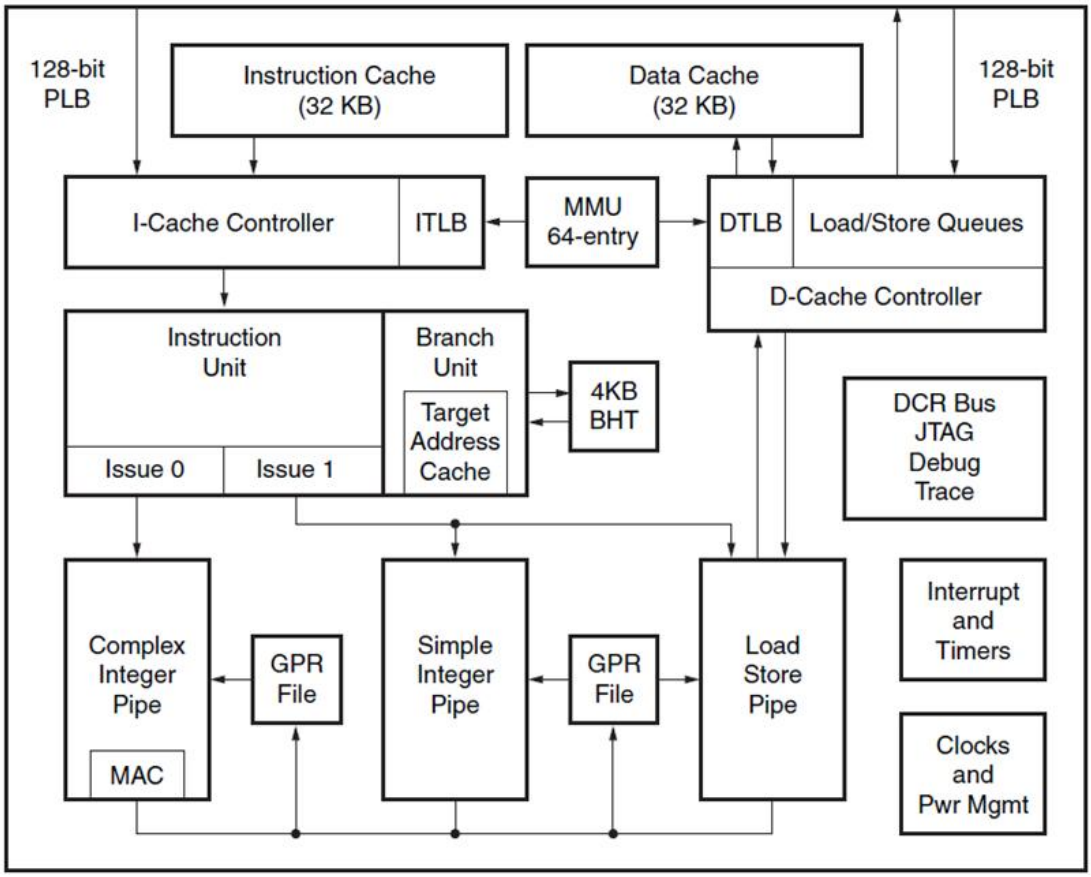


Fig. A.9: Block diagram of an embedded PowerPC 440 processor (reproduced from Xilinx UG200).

The main components of the embedded processor block in Virtex-5 FXT FPGAs are the processor, the crossbar and its interfaces, the auxiliary processing unit (APU) controller, and the control (clock and reset) module [**34**]. Fig. A.10 shows the embedded processor block and its components.

The processor has three PLB interfaces: one for instruction reads, one for data reads, and one for data writes. Typically, all three interfaces access a single large external memory. Peripheral access in PowerPC 440 systems is memory mapped, and the data PLB interfaces typically connect to various peripherals directly or via bridges. Peripherals can be implemented in hard IP elements (implemented in the FPGA fabrics) or soft logic, using the lookup tables (LUTs) and other primitive logic elements provided by the FPGA.

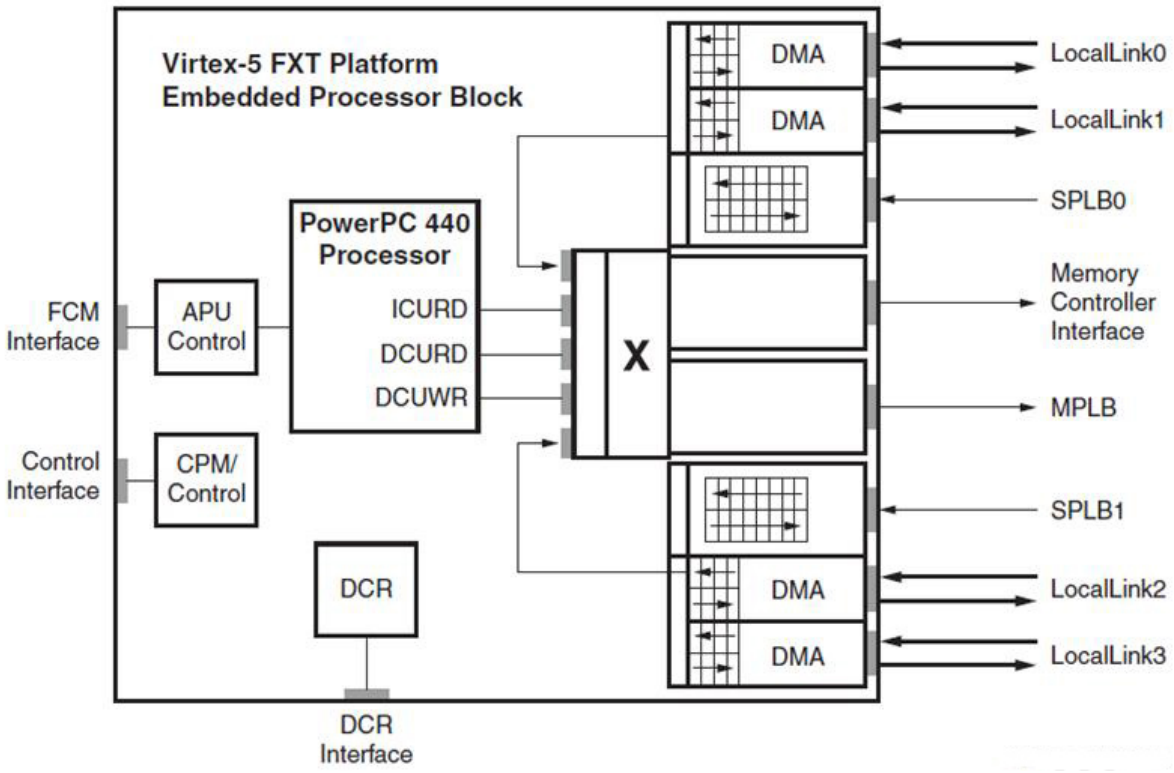


Fig. A.10: Embedded processor block in Virtex-5 FPGAs (reproduced from Xilinx UG200).

A.4.1 Crossbar and its Interfaces

The crossbar acts as a central arbitration and switching module that accepts master requests from up to five groups of master devices and redirects the transactions to one of two groups of slave devices. The crossbar also directs the responses from the slave devices back to the correct master devices. All data passing from any master device to any slave device within the embedded processor block in Virtex-5 FPGAs passes through the crossbar. The crossbar and its interfaces allow the processor with its three PLB interfaces, soft peripherals with PLB interfaces, and peripherals with LocalLink interfaces to share access to a high-performance memory controller.

A.4.2 PLB Interface

The PLB interface can be either mater PLB (MPLB) or slave PLB (SPLB). These interfaces are explained below.

A.4.2.1 MPLB Interface

The primary purpose of the crossbar MPLB interface is to provide access from the processor to PLB-based memory (if any) and non-memory peripherals. The MPLB also allows access from PLB-based masters outside the embedded processor block in Virtex-5 FXT FPGAs, connected via one of the SPLB interfaces, to PLB based memories and non-memory peripherals, which are also to be shared with the processor.

A.4.2.2 SPLB Interfaces

The primary purpose of two crossbar SPLB interfaces is to allow PLB-based masters outside the embedded processor block in Virtex-5 FPGAs to share access to the main memory on the crossbar memory controller interface (MCI). The crossbar is the primary means of establishing multi-ported access to the main memory in PowerPC 440 based systems. The SPLB interfaces also allow access to PLB-based memories and non-memory peripherals connected to the crossbar MPLB interface, which are also to be shared with the processor. A maximum of four masters can be connected to each SPLB interface.

A.4.3 PLB Interconnection Techniques

The crossbar in the embedded processor block provides a high-performance pathway to allow memory and other peripherals to be shared between the processor and other masters in the system. There are many ways that external masters, memories, and peripherals can be connected to the crossbar, which are explained below.

A.4.3.1 Simple Processor-Centric Shared Bus Design

In simple processor-centric shared bus design the “main memory” for the processor is attached to a memory controller on the PLB. The performance of this topology might be sufficient, particularly if there is no other masters in the system that need to share any of these memory or peripheral devices. Even so, access to any high-latency peripherals by the

data load/store unit might occasionally stall the processor's instruction fetch. Fig. A.11 shows the simple processor-centric shared bus design topology.

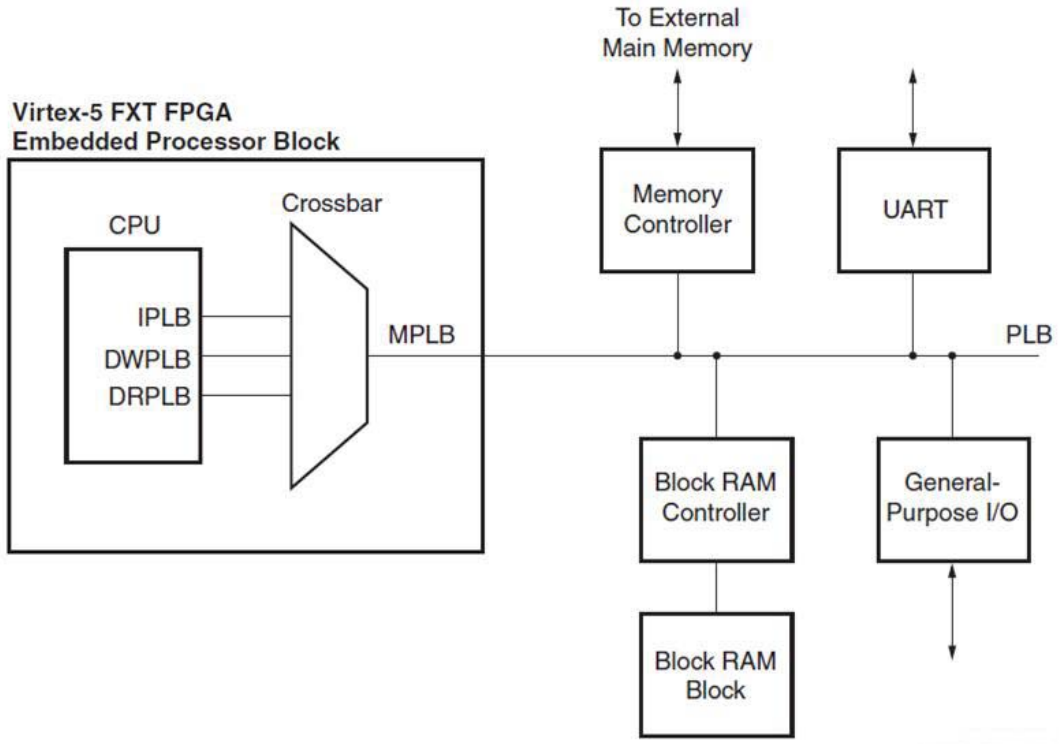


Fig. A.11: Simple processor-centric shared bus design (reproduced from Xilinx UG200).

A.4.3.2 Simple Processor-Centric Design Using Memory Controller Based Main Memory

In simple processor-centric design using memory controller based main memory topology, the PLB-based memory controller is replaced with one connected to the crossbar MCI, which is shown in Fig. A.12. Overall latency to memory is slightly improved due to the elimination of PLB arbitration cycles. Because the pathways to main memory and peripherals are now independent, peripheral access can no longer interfere with instruction fetch.

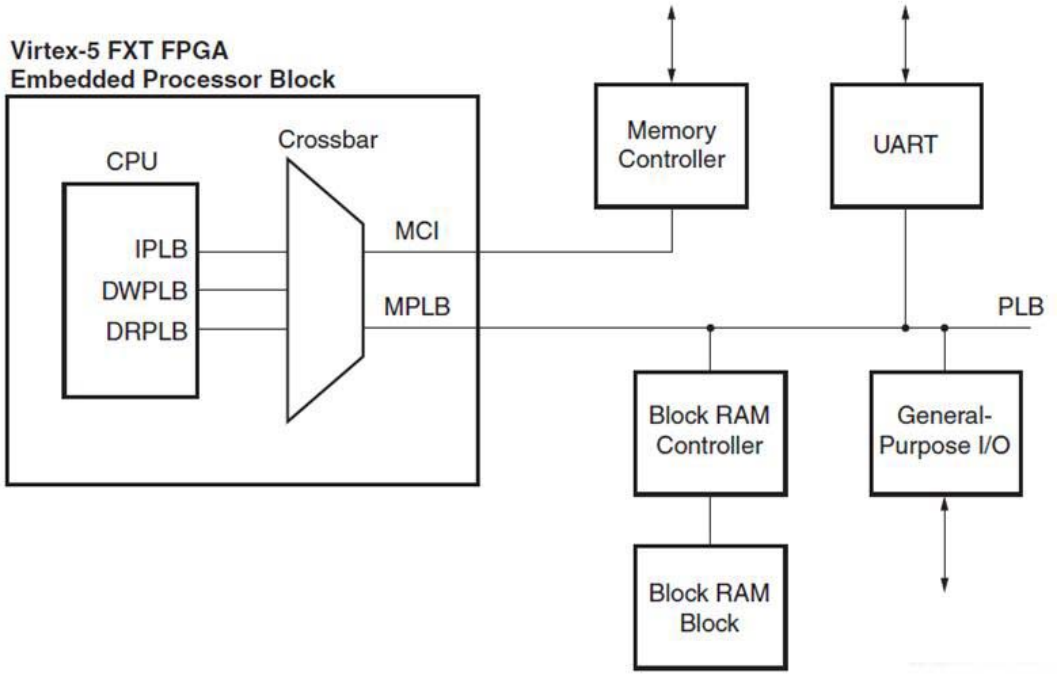


Fig. A.12: Simple processor-centric design using memory controller based main memory (reproduced from Xilinx UG200).

A.4.3.3 Other Topologies

Other PLB interconnection techniques include: main memory and peripherals shared between processor and external master, main memory shared between processor and DMA, external bridge with remote access to main memory and processor access to remote peripherals, external bridge with remote access to main memory and locally shared peripherals [**34**].

A.5 Memory Controller Interface (MCI)

The memory controller interface (MCI) block provides a bridge between the high-speed crossbar and a soft memory controller implemented in FPGA logic. The MCI provides a simple protocol that allows the soft memory controller to run much faster because it does not need to implement the more complex and more general PLB protocol [**34**].

A.6 Other Embedded Processor Blocks

Other important embedded processor blocks include: reset, cock, and power management interfaces, device control register bus, interrupt controller interface, JTAG interface, debug interface, and trace interface [**34**].

A.7 Controllers

The PPC440 supports auxiliary processing unit (APU) and DMA controller, which are explained below.

A.7.1 Auxiliary Processing Unit (APU)

The APU connects to the fabric coprocessor bus (FCB) to support custom instructions implemented in the FPGA fabric through APU programming [**34**]. For example, a double precision floating point unit (FPU) can be connected to the APU. Then, anytime the application needs to perform a floating-point computation, the processor will offload the computation to hardware where the computation can be performed faster than a software-emulated FPU [**192**] .

A.7.2 DMA Controller

The DMA controller consists of four independent DMA engines that provide high performance direct memory access for streaming data. Peripherals can directly transfer data to and from a memory controller connected to the processor block. Peripherals are connected to the DMA engines through the LocalLink interface. The DMA engines can be monitored and controlled through their device control registers (DCRs) [**34**].

APPENDIX B

XILINX ML-507 PLATFORM CONFIGURATION FOR EMBEDDED VSION APPLICATION

B.1 Introduction

Xilinx ML-507 platform contains a Virtex-5 FPGA (XC5VFX70T), which has one PowerPC440 (PPC440) processor surrounded by FPGA fabric [**33**]. To use the platform for the embedded image and video processing applications, the Xilinx ML-507 FPGA platform requires interfacing of a video camera, a PAL to VGA converter, and a VGA monitor. The detail of the development platform configuration along with its various components is described below.

B.2 Pan-Tilt-Zoom (PTZ) Video Camera

The pan-tilt-zoom (PTZ) cameras get their name because of their ability to pan (left and right), tilt (up and down), and zoom in and out of a picture plane. The PTZ cameras are able to enhance the image quality and increase the coverage area. It allows the user to have arbitrary viewing angle in a surveillance scene [**102**]. The PTZ cameras provide uniform resolution and are able to provide close observations of particular targets. These cameras are able to adopt a variety of roles such as following an object, zooming to acquire high resolution images, or imitating fixed view cameras, and, as a result, can support highly dynamic, reconfigurable task oriented surveillance.

The PTZ camera can adjust its orientation with respect to the region-of-interest (ROI) which is a very important functionality for any vision system. The selected camera for the development system is Sony EVI-D70P PTZ camera [**102**], which is shown in Fig. B.1(a) and Fig. B.1(b). The orientation of the camera can easily be controlled through RS-232 port on

the Xilinx ML-507 platform. The motivation behind the platform configuration is more towards developing an underlying infrastructure for acquisition, storage and display of video data and the design is independent of the resolution of the image data passed through. It works on all kinds of resolutions with or without pan-tilt or zoom need. The PTZ camera can be interfaced with VGA IN port of the ML-507 platform. The camera works with PAL signal system with composite video and S-video as the analog video outputs available having effective pixels of 752 (H) x 582 (V). The output of the camera is connected to the video IN port of the PAL to VGA converter, which is described below.

(a) (b)

Fig. B.1: Sony PTZ camera (a) front view (b) rear view (reproduced from Sony EVI-D70 PTZ camera).

B.3 PAL to VGA Converter

The composite video output of the PTZ camera is taken as the input for the video acquisition platform. The PAL standard of composite video is incompatible with ML-507 FPGA platform which uses RGB video signals. A PAL to VGA converter is used to convert the composite video format into VGA format. We have used V2V Pro video converter [**103**],

which is shown in Fig. B.2. This converter converts composite analog video into the corresponding RGB analog form. The output of the PAL to VGA converter is connected with the VGA IN connector of the ML-507 platform.

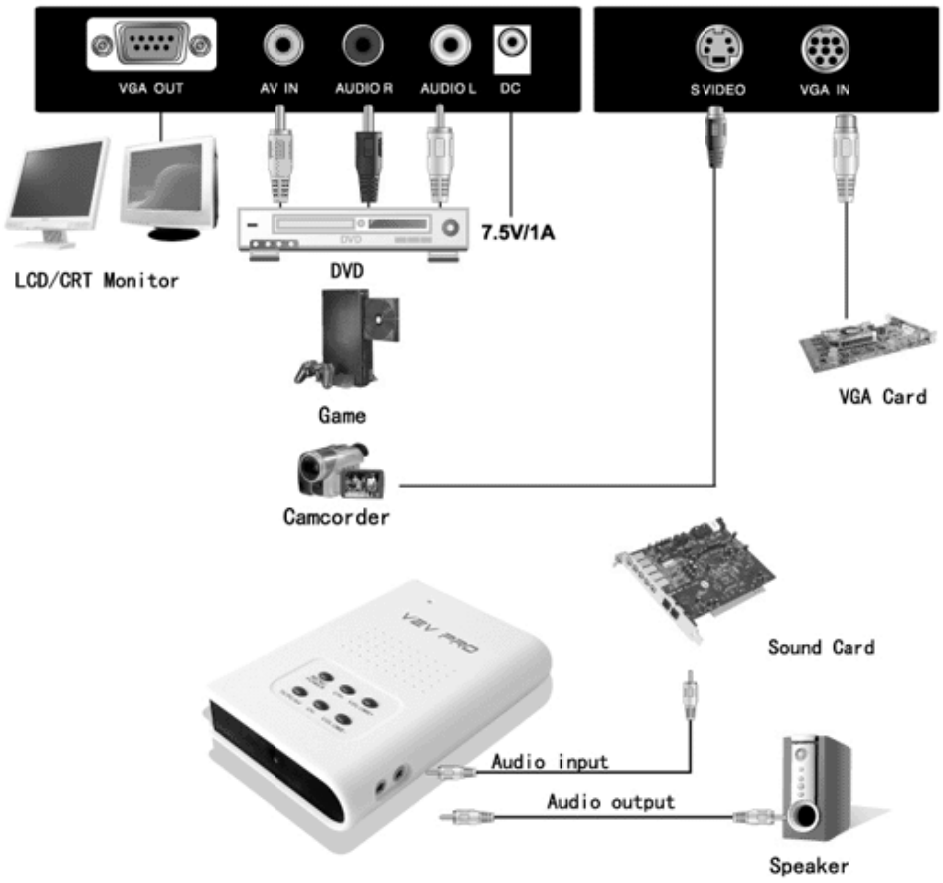


Fig. B.2: V2V Pro PAL to VGA converter (reproduced from MyGica V2V Pro).

B.4 Bus Protocols

To interface a PTZ camera and a VGA monitor with PPC440 processor for the system, the control registers of VGA input video CODEC and a DVI transmitter device are configured by I2C bus. These protocols are imperative for the proposed video acquisition system. This section describes the I2C and VGA protocol in detail.

B.4.1 VGA Protocol

A VGA video signal contains five types of active signals:

Horizontal sync (hsync) : digital signal, used for synchronization of the video.

Vertical sync (vsync) : digital signal, used for synchronization of the video.

Red (R) : analog signal, used to control the color.

Green (G) : analog signal, used to control the color.

Blue (B) : analog signal, used to control the color.

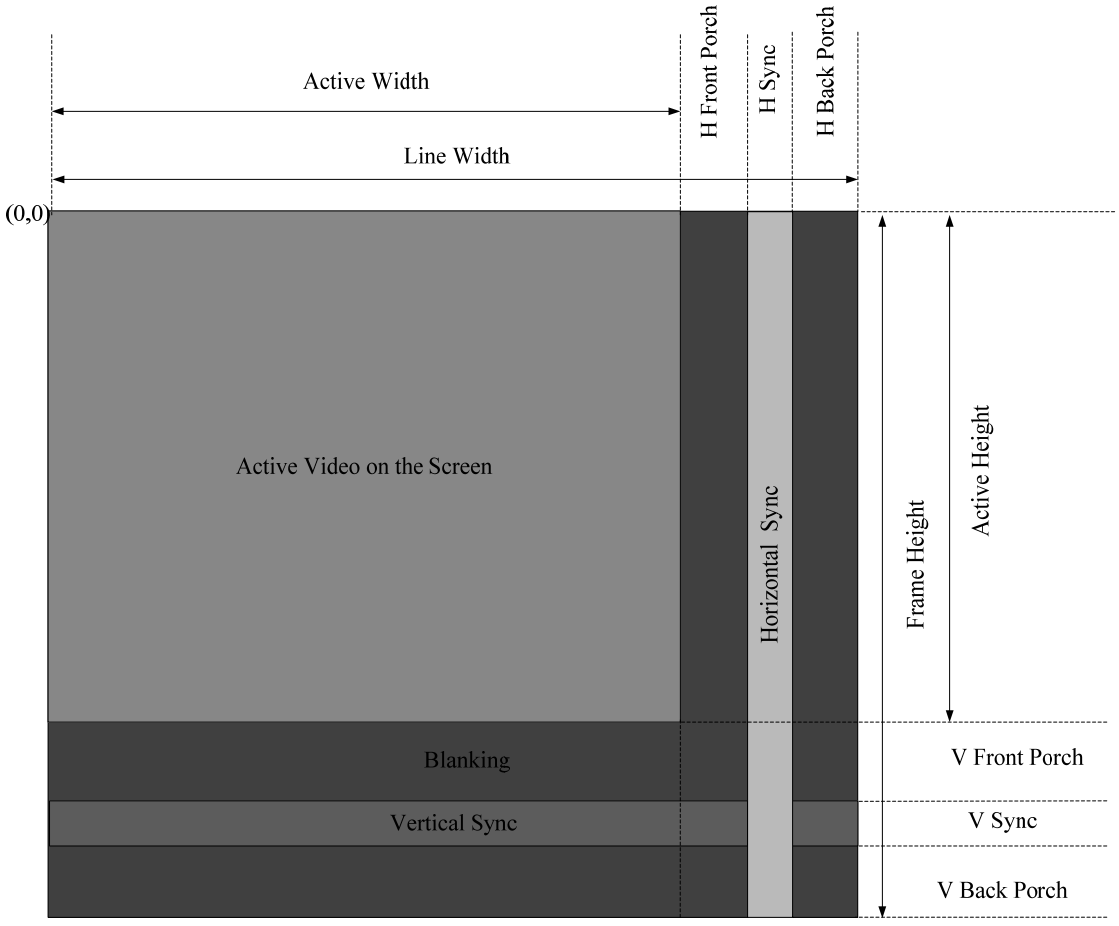


Fig. B.3: A 640×480 VGA resolution frame.

By changing the analog levels of RGB signals, all other colors are produced. The vsync signal controls the monitor to start displaying a new image or a new frame. The horizontal sync signal controls the monitor to refresh another row of 640 pixels. The video signal redraws the entire screen 60 times per second to provide for motion in the image and to

reduce flicker, this period is called the refresh rate. The duration in which the video data is being transmitted is called the active period. The remaining is the blanking portion. In the blanking interval, a sync pulse is generated. The sync pulse is followed by a back porch; which is used to decode the color information from composite signals. The front porch is a brief period inserted between the end of each transmitted line of picture and the leading edge of the next line sync pulse. The timing diagram of VGA signals is shown in Fig. B.3.

B.4.2 Inter-Integrated Circuit (I2C) Bus Protocol

Inter-integrated circuit (I2C) bus is used to form a system in which microprocessor controls one or more devices. An I2C bus consists of two wires named serial data (SDA) and serial clock (SCL), which carry information between the devices connected to the bus [**188**,**189**]. Both SDA and SCL transport bidirectional data between connected devices. By default, SDA and SCL are at logic-1. Therefore, when the bus is idle, both SDA and SCL are high. Each device on the bus has a unique address and can operate as either a transmitter or receiver.

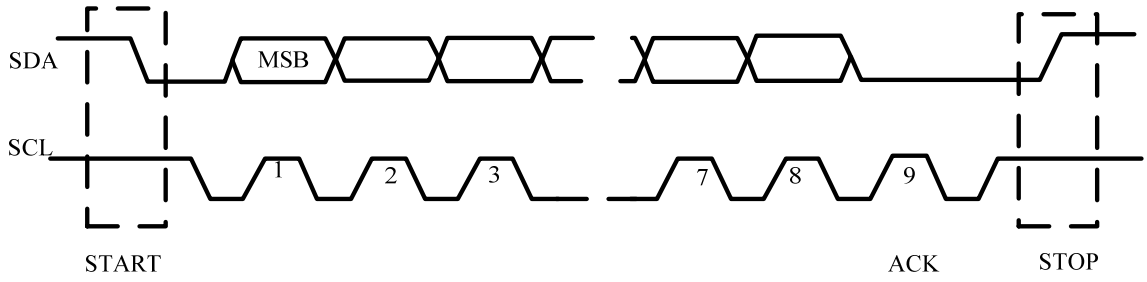


Fig. B.4: I2C bus protocol.

In addition, devices can also be configured as either master or slave. A master initiates a data transfer on the bus and generates the clock signals to permit the transfer. Any other addressed device is considered a slave. The interface is identical for master and slave devices. All devices have a unique address. They look at the address sent by the master, to decide whether the data is intended for them or not. The device generates acknowledge signal if it sees its address on the data bus. For interfacing with a device, the master sends transfer

START, followed by device address, data bytes and then transfer STOP. Data transfers on the I2C bus are initiated with a START condition, and are terminated with a STOP condition as shown in Fig. B.4.

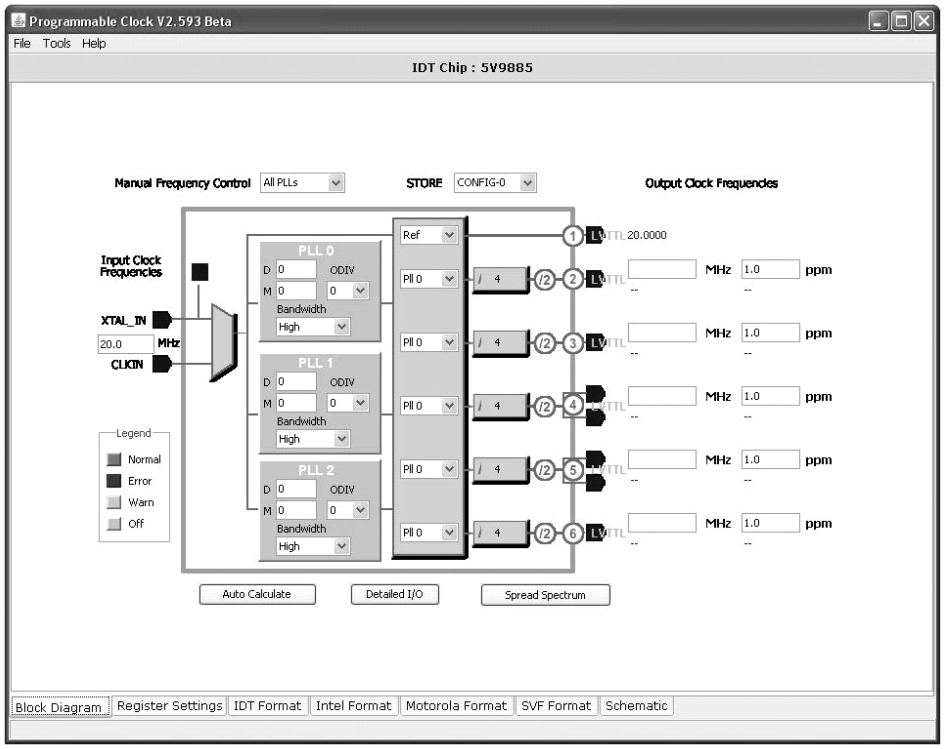
Normal data on the SDA line must be stable during the clock High period. The High or Low state of the data line can only change when SCL is Low. The START condition is a unique case and is defined by a High-to-Low transition on the SDA line while SCL is High. Likewise, the STOP condition is a unique case and is defined by a Low-to-High transition on the SDA line while SCL is High. The START and STOP data definitions ensure that the START and STOP conditions will never be confused as data [**188**].

B.5 Platform Set-up for the Embedded Vision Applications

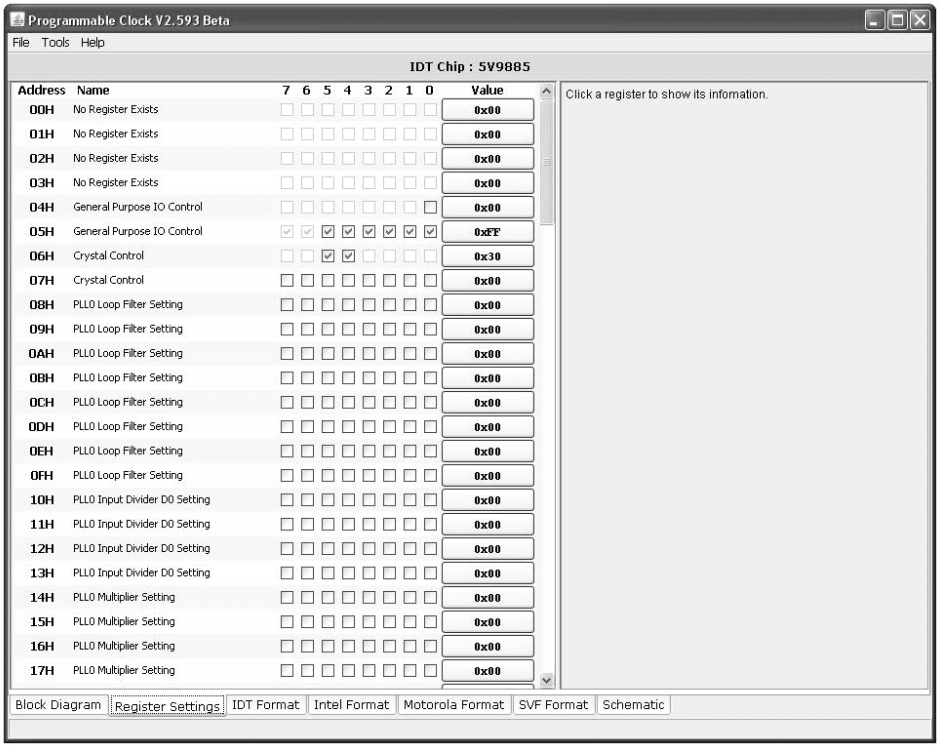
The Xilinx ML-507 platform is suited for embedded vision applications. This needs some specific platform tuning. For example to use the platform for 640x480 video resolution the control registers of video codec and the display controller are configured by specific values. Apart from this for displaying the video in particular resolution a specific clock frequency is needed. The specific clock frequency is generated by programming the IDT clock generator, which is explained below.

B.5.1 Programming the IDT Clock Generator

The ML-507 platform has an IDT5V9885 EEPROM programmable clock generator device. To generate custom clock frequency, the registers of the programmable clock generator are programmed. Programmable clock software provided by the IDT is used for the clock generator programming. The programmable interface of the IDT chip is shown in Fig. B.5(a). The various register settings of the IDT5V9885 are shown in Fig. B.5(b). To make the configured clock frequency available at the FPGA pin, the serial vector format (SVF) output of the software is downloaded to the platform, through the Xilinx *iMPACT* tool.



(a)



(b)

Fig. B.5: (a) IDT programmable clock structure (b) IDT programmable clock register settings.

To download the configuration bits, we have used Xilinx platform USB II download cable with flying leads connected in-between the platform and the *iMPACT* tool. An internal EEPROM allows saving and restoring the configuration of the device without having to reprogram it on power-up. The SVF file procedure is given below:

Connect a Xilinx download cable to the board using flying leads connected to jumper J3 as shown in Fig. 6.

Click Start→ iMPACT.

Click Boundary Scan.

Locate the SVF file (as generated by the IDT software) and click Open.

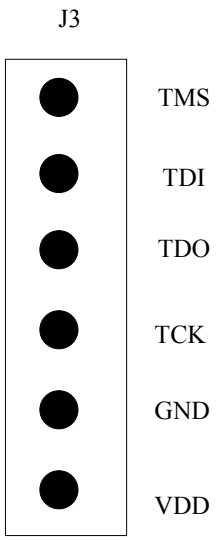


Fig. B.6: IDT5V9885 JTAG connector.

Right-click on the device and select Execute XSVF/SVF, shown in Fig. B.7.

To finish programming the chip, cycle the power by turning off the board power switch. After turning the board back on, verify that the clock frequencies are correct.

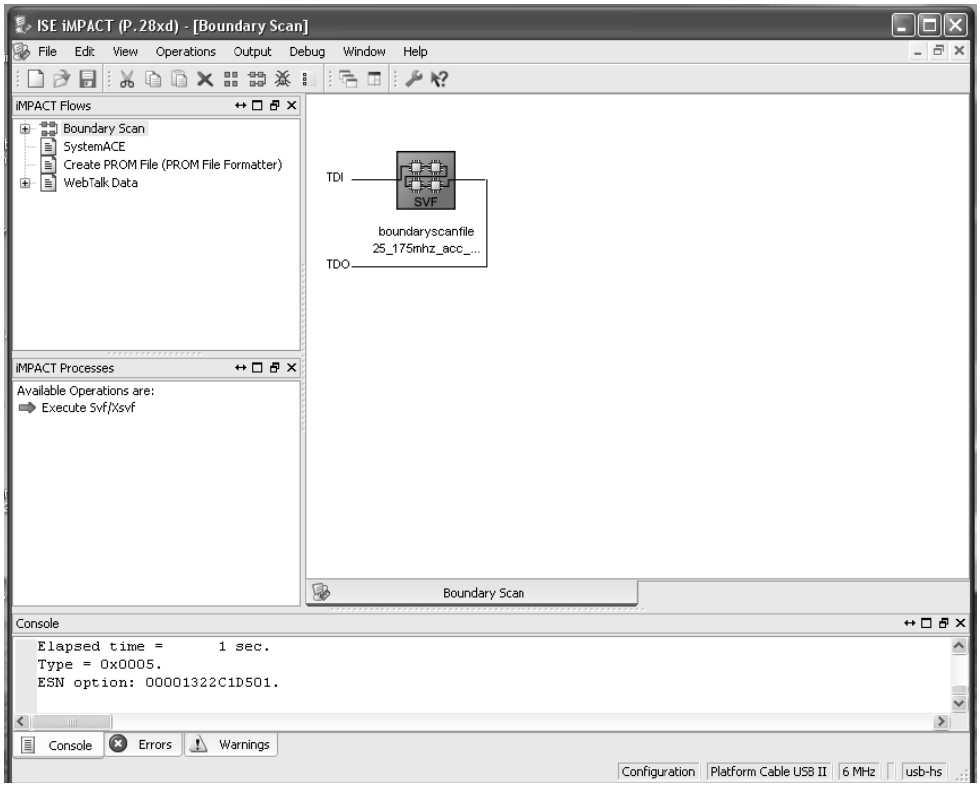


Fig. B.7: SVF output in Xilinx *iMPACT*.

B.5.2 VGA Input Video Codec

The FPGA platform supports connectivity to an external VGA source. The VGA input codec circuitry utilizes an Analog Devices AD9980 device. This is an 8-bit 95 MSPS interface optimized for capturing YPbPr video and RGB graphics signals. The AD9980 device is controlled by way of the Video I2C bus. The block diagram of the AD9980 high performance 8-bit display interface is shown in Fig. B.8.

The AD9980 is configured for the 640X480@60fps video resolution through programming of its internal registers. The details of each register are given in [**100**]. An I2C controller is used to write and read the control registers of the AD9980.

For example, the register address 0x01 is assigned to PLL Divide ratio MSBs register. This register is for bits [11:4] of the PLL divider. This register should be loaded first whenever a change is needed. The PLL derives a pixel clock from the incoming Hsync signal. The pixel

clock frequency is then divided by an integer value, such that the output is phase-locked to Hsync. This PLLDIV value determines the number of pixel times per line.

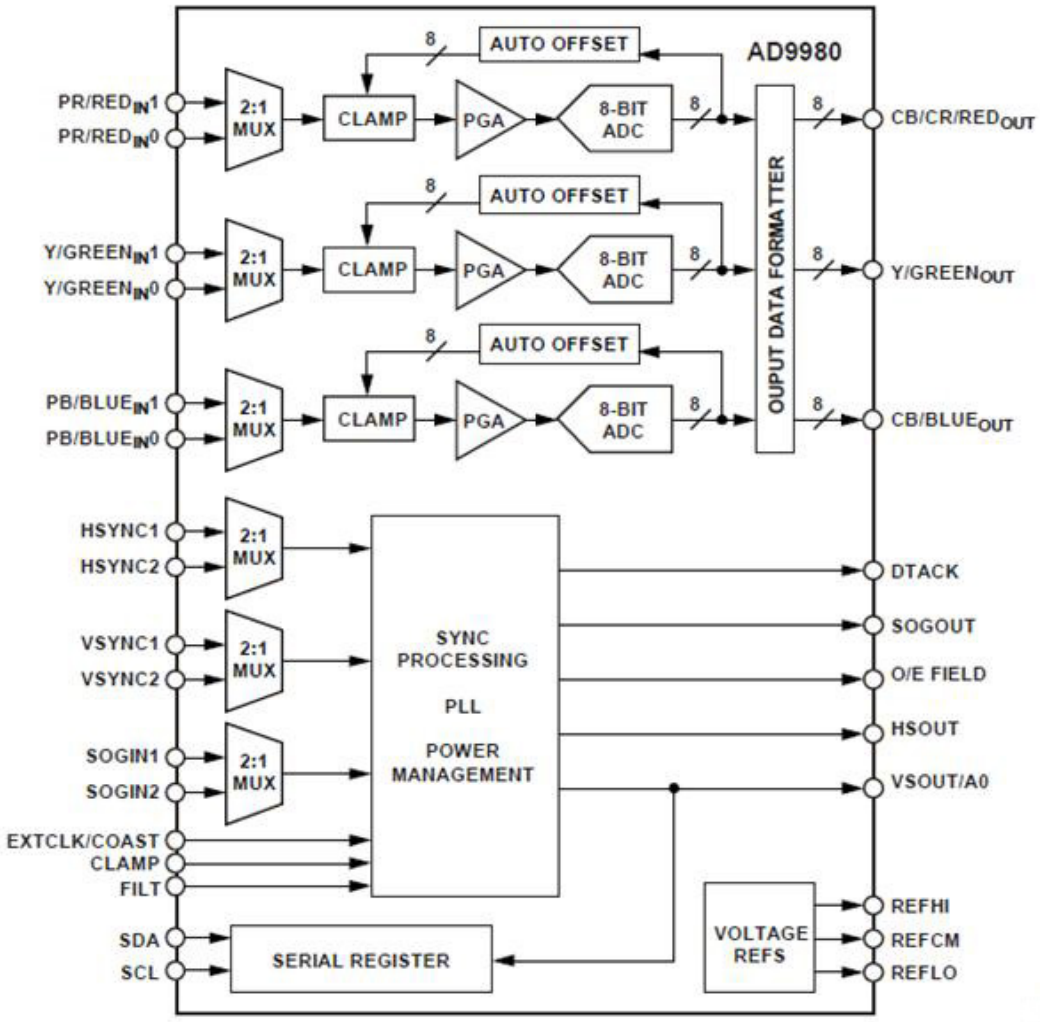


Fig. B.8: AD9980 functional block diagram (reproduced from AD9980).

The recommended VCO range and charge pump and current settings for the VGA standard display are as follows:

0x01: As for the 25.175 MHz clock the PLL divider must have the value of "800" or 0x320. Therefore, the register 0x01 is written with 0x32.

0x02: Similarly, the register 0x02 is used for the PLL divide ratio LSBs. As explained above this register is written with 0x00 data.

0x03: The register 0x03 is used for the clock generator control. The configuration of this register is as follows:

Bit [7:6]: These two bits establish the operating range of the clock generator. To configure the AD9980 chip for 25.175 MHz clock frequency the bits [7:6] of the register 0x03 must contain 01 data.

Table B.1: VCO Range and Charge Pump and Current Settings

Item	Settings
Standard	VGA
Resolution	640 × 480
Refresh rate	60
Horizontal Frequency	31.500
Pixel rate (MHZ)	25.175
PLL Divider	800
VCORNGE	01
Current (uA)	100

Bits [5:3] of this register are used to control the charge pump current. These three bits establish the current driving the loop filter in the clock generator. To set the clock frequency of 25.175 MHz the current must be set to 100. As given in the data sheet of AD9980, the bits [5:3] must be written with 001 to generate the above frequency. Bit 2 of register 0x03 determines the source of the pixel clock frequency. To use an internally generated clock bit 2 of this register must be zero. Logic 0 enables the internal PLL that generates the pixel clock from an externally provided Hsync.

- 0x04: ADC clock phase adjust
- 0x05: 7-bit Red channel gain control

- 0x06: Must be written 0x00 following a write of register 0x05 for the proper operation.
- 0x07: 7-bit Green gain control
- 0x08: Must be written 0x00 following a write of register 0x07 for the proper operation.
- 0x09: 7-bit Blue gain control
- 0x0A: Must be written 0x00 following a write of register 0x09 for the proper operation.
- 0x0B: 8-bit MSB of the Red channel offset control. It controls the brightness of each respective channel.
- 0x0C: Linked with 0x0B to form the 9-bit red offset that controls the brightness of the red-channel in auto-offset mode.
- 0x0D: 8-bit MSB of the Green channel offset control. It controls the brightness of each respective channel.
- 0x0E: Linked with 0x0D to form the 9-bit green offset that controls the brightness of the green-channel in auto-offset mode.
- 0x0F: 8-bit MSB of the Blue channel offset control. It controls the brightness of each respective channel.
- 0x10: Linked with 0x0F to form the 9-bit blue offset that controls the brightness of the blue-channel in auto-offset mode.
- 0x11: This register sets the threshold of the sync separator's digital comparator.
- 0x12: Hsync Control.
    - Bit [7]: 0, The chip determines the active Hsync source.
    - Bit [6]: 0, Hsync is from Hsyc input pin.
    - Bit [5]: 0, The chip selects the Hsync input polarity.

- Bit [4]: 1, Active high input Hsync.
- Bit [3]:1, Active high input Hsync output.
- Bit [2:0]: Reserved.

- 0x13: Sets the number of pixel clocks that Hsync out is active.
- 0x14: Vsync Control
  - Bit [7]: 0 The chip determines the active Vsync source.
  - Bit [6]: 0 Vsync is from Vsyc input pin.
  - Bit [5]: 0 The chip selects the Vsync input polarity.
  - Bit [4]: 1 Active high input Vsync.
  - Bit[3]:0 Active low input Vsync output.
  - Bit[2]: 0 The Vsync filter is disabled.
  - Bit [1]:Vsync output duration is unchanged.
  - Bit [0]: Reserved.
- 0x15: Sets the number of Hsync that Vsync out is active. (This is only used if 0x14, Bit 1 is set to 1).
- 0x16: The number of Hsync periods to Coast prior to Vsync.
- 0x17: The number of Hsync periods to Coast after Vsync.
- 0x18: Coast source
- 0x19: Clamp placement.
- 0x1A: Clamp duration.
- 0x1B: Clamp and offset
- 0x1C: Must be set to 0xFF for proper operation
- 0x1D: SOG control
- 0x1E:Power.
- 0x1F: Output select 1.

- Bit [7]: Reserved.
- Bit [6:5]:00, RGB Mode.
- Bit[4]: 1, Primary output is enabled.
- Bit[3]: 0, Secondary output is enabled.
- Bit[2:1]:10, Medium high output drive strength.
- Bit[0]: 0, Noninverted pixel clock.

- 0x20: Output select 2.
- 0x21: Must be set to default for proper operation.
- 0x22: Must be set to default for proper operation.
- 0x23: Sync filter window width.
- 0x24: Sync detect.
- 0x25: Sync polarity detect.
- 0x26: Hsync per Vsync MSBs.
- 0x27: Hsync per Vsync LSBs.
- 0x28: Must be written 0xBF for proper operation.
- 0x29: Must be written 0x02 for proper operation.
- 0x2A: Reserved.
- 0x2B: Reserved.
- 0x2C: Offset hold.
- 0x2D: Must be written 0xE8 for proper operation.
- 0x2E: Must be written 0xE0 for proper operation.

B.5.3 Chrontel CH7301C Display Controller

The functional block diagram of the CH7301C is shown in Fig. B.9 [**101**]. A DVI/VGA monitor can be interfaced with the ML-507 platform by using a DVI connector present on the ML-507 platform [**33**].

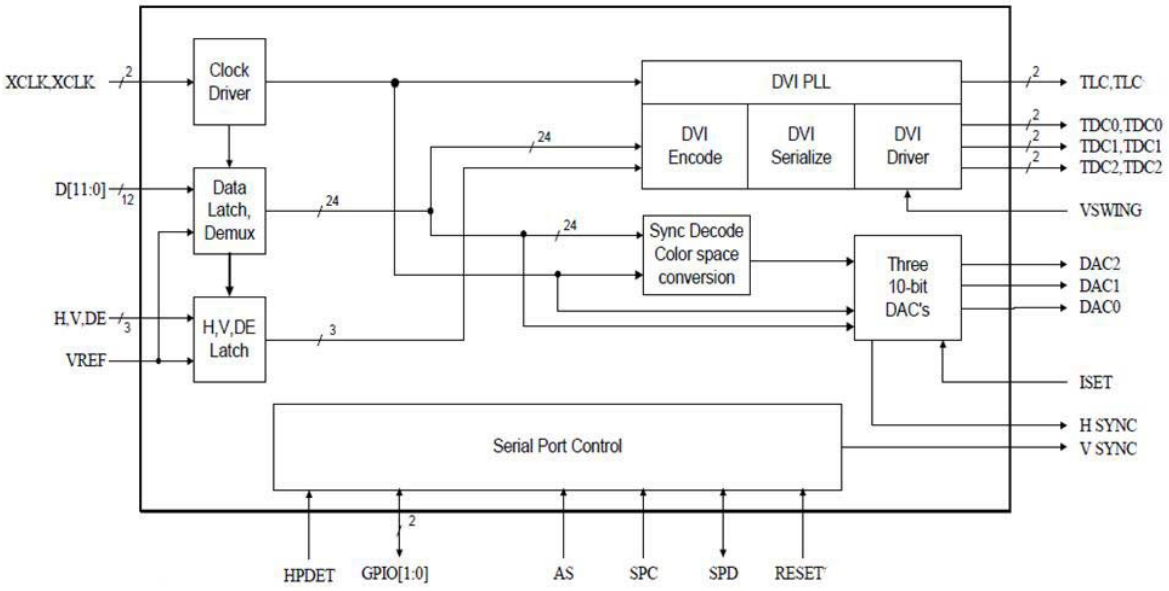


Fig. B.9: CH7301C functional block diagram (reproduced from CH7301 DVI transmitter).

The DVI connector uses Chrontel CH7301C DVI transmitter device or display controller device. It accepts a digital graphics input signal, and encodes and transmits data through the DVI connector. The device accepts data over one 12-bit wide variable voltage data port, which supports different data formats including RGB and YCrCb.