IMPLEMENTATION OF A REAL TIME HOUGH TRANSFORM USING FPGA TECHNOLOGY

A THESIS SUBMITTED TO
THE UNIVERSITY OF KENT
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE SUBJECT OF ELECTRONIC ENGINEERING

By
Evangelos Alexiadis
December 2009

To my family and friends

Acknowledgments

First and foremost I would like to express my deep and sincere gratitude to my supervisor, Peter Lee. His wide knowledge and his logical way of thinking have been of great value for me. His understanding, encouraging and personal guidance have provided a good basis for the present thesis. I could not have asked for a better supervisor and without his help I could not have finished my PhD successfully.

I would like to thank my friends in the Embedded Systems laboratory, both old and new, who have been able to keep me smiling even during the most difficult times. Special thanks to my colleague, Dr Michael Wisdom. You have been such a good laugh and an excellent office-neighbour and thank you for not minding my endless interruptions on your desk asking for help and advice.

To the people in the Department of Electronics at the University of Kent; and especially to Dr. Konstantinos Sirlantzis; thank you for the opportunities and the help you have given me over the years. I am privileged to have studied and worked with you all. To the IT people and especially to Julian Lucas, as well as the support staff Helen Winder, Denyse Menne, Ariella Bowman, Catherine Butler and Nina Lozanska; thank you all for being so amazing and for your help through out these years.

I would especially like to thank the Mitzalis family, for their support and belief in me all these years. You treated me as part of your family and I will never forget this. You are wonderful people and I feel honoured to have meeting you. God bless you.

A big thank you to all my friends for being there for me through the tough and the good times; Alexi, Louiza, Vangeli, Alexandre, Giorgo, Dimitra, Stathi,

Dimitri, Sissy, Vicky, Kostantia, Tzeni, Theodore, Frossina and Antoni, and my wonderful landlords, Mr. Alan & Mrs. Marje; thank you for keeping me strong and for putting up with me. I am blessed to have you all.

To my dear friends Popara and Antonis; you have been a constant source of laughter, inspiration and support. I love you, and I will always be there for you.

Finally I wish to express my deepest thanks to the members of my family; each of you has a special place in my heart. I would like to thank my parents, Alexandro and Eleni, and my brother Vasili, for their love, understanding, patience, endless support, and never failing to have faith in me. I love you so much, and I would not have made it this far without you.

Last but not least, many thanks to my Pinaki mou, for her understanding, endless patience, help during my writing and encouragement when it was most required. She made me a happy person and gave me the extra strength, motivation and love necessary to get things done. I love you more (possible) and thank you for being part of my life.

Evangelos Alexiadis

Canterbury, December 2009

Abstract

This thesis is concerned with the modelling, design and implementation of efficient architectures for performing the Hough Transform (HT) on mega-pixel resolution real-time images using Field Programmable Gate Array (FPGA) technology. Although the HT has been around for many years and a number of algorithms have been developed it still remains a significant bottleneck in many image processing applications.

Even though, the basic idea of the HT is to locate curves in an image that can be parameterized: e.g. straight lines, polynomials or circles, in a suitable parameter space, the research presented in this thesis will focus only on location of straight lines on binary images. The HT algorithm uses an accumulator array (accumulator bins) to detect the existence of a straight line on an image. As the image needs to be binarized, a novel generic synchronization circuit for windowing operations was designed to perform edge detection. An edge detection method of special interest, the canny method, is used and the design and implementation of it in hardware is achieved in this thesis.

As each image pixel can be implemented independently, parallel processing can be performed. However, the main disadvantage of the HT is the large storage and computational requirements. This thesis presents new and state-of-the-art hardware implementations for the minimization of the computational cost, using the Hybrid-Logarithmic Number System (Hybrid-LNS) for calculating the HT for fixed bit-width architectures. It is shown that using the Hybrid-LNS the computational cost is minimized, while the precision of the HT algorithm is maintained.

Advances in FPGA technology now make it possible to implement functions as the HT in reconfigurable fabrics. Methods for storing large arrays on FPGA's are presented, where data from a 1024 x 1024 pixel camera at a rate of up to 25 frames per second are processed.

Publications Arising From This Work

1 P. Lee and E. Alexiadis. "An implementation of a multiplierless Hough transform on an FPGA platform using hybrid-log arithmetic ". in *SPIE Conference on Real-Time Image Processing 2008*, California, USA, 2008

TABLE OF CONTENTS

1 Introduction.....1

1.1 Motivation, aim and objectives.....1

1.2 Hough Transform.....1

1.3 Logarithms and Hough Transform.....2

1.4 Edge Detection and Hough Transform.....3

1.5 FPGA and Hough Transform.....4

1.6 Thesis Organization.....6

1.7 Research Contributions.....9

2 Hough Transform Literature Review.....10

2.1 Introduction.....10

2.2 The Hough Transform Method.....10

2.3 Implementing the Hough Transform.....12

2.4 Advantages and Disadvantages of the Hough Transform.....14

2.5 Early Development of the Hough Transform.....15

2.6 Hough Transform Methods.....16

2.6.1 Generalized Hough Transform.....16

2.6.2 Fast Hough Transform.....17

2.6.3 Adaptive Hough Transform.....19

2.6.4 Fast Adaptive Hough Transform.....19

2.6.5 The Binary Hough Transform.....20

2.6.6 The Dynamic Combinatorial Hough Transform.....21

2.6.7 Connective Hough Transform.....21

2.6.8 The Hierarchical Hough Transform....22

2.6.9 The Probabilistic Hough Transform....23

2.6.10 The Weighted Hough Transform....24

2.6.11 Multiresolution Hough Transform....24

2.6.12 Randomized Hough Transform....25

2.7 Applications of the Hough Transform....26

2.8 Parallel Processing Architectures....33

2.8.1 SIMD Implementation....36

2.8.2 MIMD Implementation....37

2.8.3 Dedicated Systems....38

2.9 Conclusion....41

3 Edge Detection & Digital Logarithms Literature Review....42

3.1 Introduction....42

3.2 Introduction to Edge Detection....42

3.3 Edge Detection Methods....45

3.3.1 Sobel Method....46

3.3.2 Canny Method....48

3.3.2.1 Smoothing....49

3.3.2.2 Gradient Calculation....50

3.3.2.3 Magnitude and Phase....51

3.3.2.4 Non-Maximum Suppression....51

3.3.2.5 Threshold....51

3.4 Introduction to Digital Logarithms....52

3.4.1 Digital Logarithms Methods....52

3.5 Conclusion.....61

3.6 Hardware Architecture.....59

3.7 Results Using the Canny Algorithm.....59

3.8 A Synchronizing Circuit.....70

3.9 Conclusion.....72

4 Implementation of the Canny Edge Detection Method.....63

4.1 Introduction.....63

4.2 Hardware Implementation.....63

4.2.1 Moving Window Operator.....63

4.3 Canny Hardware Implementation.....66

4.3.1 Image Smoothing.....66

4.3.2 Horizontal and Vertical Gradient Calculation.....67

4.3.3 Directional Non-Maximum Suppression.....68

4.3.4 Threshold and Thinning.....71

4.4 Hardware Architecture.....72

4.5 Results Using the Canny Algorithm.....72

4.5.1 Software Version of Canny Algorithm Using Floating Point Arithmetic.....74

4.5.2 Software Version of Canny Algorithm Using Fixed Point Arithmetic.....75

4.5.3 Hardware Version of Canny Algorithm Using floating Point Arithmetic.....77

4.5.4 Hardware Version of Canny Algorithm Using Fixed Point Arithmetic.....79

4.6 A Synchronizing Circuit....83

4.7 Conclusion....85

5 Overview of FPGA Technology....86

5.1 Introduction....86

5.1.1 Digital Signal Processors....86

5.1.2 ASICs....88

5.2 Introduction to FPGA's....88

5.3 Summary of Modern FPGAs....91

5.4 Number Representation on FPGAs....94

5.5 Xilinx's Xtreme DSP Block....97

5.6 Conclusion....99

6 Hybrid Logarithmic Number System....102

6.1 Introduction....102

6.2 Logarithmic Converter Design....106

6.2.1 Design Considerations....106

6.2.2 Logarithmic Converter....107

6.2.3 Logarithmic Multiplication Implementation....113

6.3 Conclusion....114

7 Hybrid-LNS & Hough Transform....115

7.1 Introduction....115

7.2 The Linear Hough Transform....115

7.2.1 Proposed Linear Implementation Using MATLAB....117

7.3 The Logarithmic Hough Transform....124

7.3.1 Proposed Logarithmic Implementation Using MATLAB....126

7.4 Hardware Implementation.....132

7.5 Conclusion.....133

8 Design of a LUT Based Accumulator Cell.....135

8.1 Introduction.....135

8.2 Parametric Description of a Straight Line.....135

8.3 Accumulator Cell.....136

8.4 An Alternative Accumulator Cell.....137

8.5 Simulation Results.....138

8.6 The Complete System.....140

8.7 Conclusion.....143

9 Summary & Conclusion.....144

9.1 Summary.....142

9.2 Future Work.....147

R References.....R1

A Appendix A-Test images and HT parameter space output......A1

B Appendix B- Series of tests with different fixed points.....B1

LIST OF FIGURES

Figure 1.5-1 – Block Diagram of the System.....5
Figure 2.2-1 – Left Figure: xy Plane, Right Figure: Parameter Space.....11
Figure 2.2-2 – Parametric Description of a Straight Line.....12
Figure 2.3-1 – Subdivision of the mc-Plane Into Cells.....13
Figure 2.6-1 – Shapes Detected with the GHT. a) Simple Shape b) Composite Shape.....17
Figure 2.6-2 – Example of a Quadtree Search Used in the FHT.....18
Figure 2.7-1 – Vehicle License Plate Identification Using the HT.....30
Figure 2.7-2 – Experimental results of lane boundary detection. The detected lane boundaries are overlaid on the input gray scale images.....31
Figure 2.7-3 – Simple s–topes: a) Monosphere, b) Bisphere, c) Thrisphere, d) Tetrasphere.....33
Figure 2.8-1 – SIMD Architecture Block Diagram.....34
Figure 2.8-2 – MIMD Architecture Block Diagram.....35
Figure 2.8-3 – A Pipelined Architecture for Real Time Measurements.....39
Figure 2.8-4 – The Elementary CORDIC Arithmetic Unit.....41
Figure 3.2-1 – Example of Edge Detection. (a) Image on a Grey Background. (b) Edge Enhanced Image Showing Only the Outlines of the Objects Using the Canny Method.....43
Figure 3.2-2 – Step Edges. (a) The Change in Level Occurs Exactly at Pixel 10. (b) The Same Level Change as Before, but Over 4 Pixels Centred at Pixel 10. This is a *Ramp* Edge. (c) Same Level Change but Over 10 Pixels, Centred at 10. (d) A Smaller Change Over 10 Pixels. The Insert Shows the Way the Image Would Appear, and the Dotted Line Shows Where the Image was Sliced to Give the Illustrated Cross-Section.....45
Figure 3.3-1 – Sobel Output.....47
Figure 3.3-2 – Schematic of Canny Edge Detection.....49
Figure 3.3-3 – Prewitt Kernels.....51
Figure 3.4-1 - Mitchell's Approximation.....53
Figure 3.4-2 – Two-Part Logarithm Approximation.....55

Figure 3.4-3 – A 4-bit Leading One Detector.....57
Figure 3.4-4 – A 16-bit Leading One Detector.....57
Figure 3.4-5 - Floating-Point / Logarithm Converter.....61
Figure 4.2-1– Example of the Window Operator in a 5x5 Image.....65
Figure 4.2-2 – Architecture of a 3x3 Window.....66
Figure 4.3-1 – Filter Coefficients a) 1-D Across Rows b) 1-D Across Columns c) 2-D Filter.....67
Figure 4.3-2 – Gradient Orientation.....69
Figure 4.3-3 – Pixel Interpolation.....70
Figure 4.4-1 – Canny Algorithm Block Diagram.....73
Figure 4.5-1 – Software Implementation Using Floating Point Arithmetic....74-75
Figure 4.5-2 – Software Implementation Using 8-bits Fixed Point Arithmetic...76
Figure 4.5-3 – Difference Between Floating Point and Fixed Point Arithmetic..77
Figure 4.5-4 – Hardware Implementation Using Floating Point Arithmetic..77-78
Figure 4.5-5 – Difference Between Software Version Using Floating Point and Hardware.....79
Figure 4.5-6– Hardware Implementation Using 8-bit Fixed Point Arithmetic....80
Figure 4.5-7– Difference Between Hardware Version Using Fixed Point and Software Version Using Floating Point Arithmetic.....81
Figure 4.6-1– The Synchronization Circuit Block Diagram.....84
Figure 5.1-1 – Block Diagram of an ADSP-21xxx Core.....87
Figure 5.2-1– FPGA Architecture.....89
Figure 5.2-2 – FPGA Programmable Logic Cell.....90
Figure 5.3-1 – Altera Logic Element (LE).....92
Figure 5.3-2 – Xilinx Virtex Slice.....93
Figure 5.3-3 – Altera DSP Block Diagram.....94
Figure 5.4-1 – Floating-Point Representation.....95
Figure 5.4–2 – Fixed-Point Representation.....96
Figure 5.5-1 – A DSP48 Tile Consisting of Two DSP48 Slices.....99
Figure 6.1-1 – Graph of Log(X) for $0 \leq X \leq 10$.....103
Figure 6.1-2 – Non Linear Functions.....105
Figure 6.2-1 - Linear to Log Converter Block Diagram.....108
Figure 6.2-2 - Log to Linear Converter Block Diagram.....110
Figure 6.2-3 - LUT Shapes.....112

Figure 6.2-4 - Logic for Multiplication in the Logarithmic Domain.....113
Figure 7.2-1- Basic HT Calculation Element.....116
Figure 7.2-2 - Test Image at Rotated Angles of 60 and 45 Degrees.....118
Figure 7.2-3 - Hough Transform Parameter Space Output Graphs on 1024 x 1024 Binarised Images Using Floating Point Arithmetic Precision.....119
Figure 7.2-4 - Hough Transform Parameter Space Output Graphs on 1024 x 1024 Binarised Images Using Linear 8-bit Arithmetic Precision.....120
Figure 7.2-5 - Hough Transform Parameter Space Output Graphs on 1024 x 1024 Binarised Images Using Linear 12-bit Arithmetic Precision.....121
Figure 7.2-6 - Hough Transform Parameter Space Difference Graphs on 1024 x 1024 Binarised Images Between Floating Point and Linear 8-bit Arithmetic Precision.....122-123
Figure 7.2-7- Hough Transform Parameter Space Difference Graphs on 1024 x 1024 Binarised Images Between Floating Point and Linear 12-bit Arithmetic Precision.....123-124
Figure 7.3-1- Hybrid-LNS HT Element.....125
Figure 7.3-2 - Hough Transform Parameter Space Output Graphs on 1024 x 1024 Binarised Images Using Floating Point Arithmetic Precision.....126-127
Figure 7.3-3- Hough Transform Parameter Space Output Graphs on 1024 x 1024 Binarised Images Using Hybrid-Log 8-bit Arithmetic Precision.....127-128
Figure 7.3-4: Hough Transform Parameter Space Output graphs on 1024 x 1024 Binarised Images Using Hybrid-Log 12-bit Arithmetic Precision.....128-129
Figure 7.3-5- Hough Transform parameter Space Difference Graphs on 1024 x 1024 Binarised Images Between Floating Point and Hybrid-Log 8-bit Arithmetic Precision.....130
Figure 7.3-6- Hough Transform Parameter Space Difference Graphs on 1024 x 1024 Binarised Images Between Floating Point and Hybrid-Log 12-bit Arithmetic Precision.....131
Figure 8.3-1- Basic Accumulator Cell Block Diagram.....137
Figure 8.4-1- Alternative Accumulator Cell Block Diagram.....138
Figure 8.6-1- Stages of the HT Implementation.....140

LIST OF TABLES

Table 1.5-1 – Resolution Vs Frames Per Second.....6
Table 2.8-1 – The CORDIC Arithmetic Function.....40
Table 4.5-1– Logic Calculations for the Canny Edge Detector.....82
Table 5.2-1– Summary of Four Commercial FPGA.....90
Table 5.3-1 - Key Features for the Latest FPGAs.....92
Table 6.1-1- Logarithm Conversion Rules.....103
Table 6.2-1 – Example Linear to Logarithm Converter LUT.....108
Table 6.2-2 – Example Logarithm to Linear Converter LUT.....111
Table 6.2-3 – MSE results for conversion to and from log / linear for different size / shaped LUTs.....112
Table 7.2-1- Correlation Between Number of Hough Elements with Operations per Second.....117
Table 7.4-1- Implementation Statistics of HT Elements.....133
Table 8.6-1 - The Complete System.....142

LIST OF ABBREVIATIONS

HT	Hough Transform
FPGA	Field Programmable Gate Array
LNS	Logarithmic Number System
Hybrid-LNS	Hybrid Logarithmic Number System
GHT	General Hough Transform
ASIC	Application-Specific Integrated Circuit
SHT	Standard Hough Transform
PSF	Parameter Space Function
LUT	Look Up Table
GHT	Generalized Hough Transform
FHT	Fast Hough Transform
AHT	Adaptive Hough Transform
FAHT	Fast Adaptive Hough Transform
BHT	Binary Hough Transform
DCHT	Dynamic Combinatorial Hough Transform
DGHT	Dynamic Generalized Hough Transform
CHT	Connective Hough Transform
HHT	Hierarchical Hough Transform
PHT	Probabilistic Hough Transform
WHT	Weighted Hough Transform
MHT	Multiresolution Hough Transform
RHT	Randomized Hough Transform
VLSI	Very Large Scale Integration
BHT	Bounded Hough Transform
SIMD	Single Instruction Multiple Data
MIMD	Multiple Instruction Multiple Data
PE	Processing Element
CAM	Content Addressable Memory
CORDIC	Co-Ordinate Rotation Digital Computer
FIFO	First In First Out
VHDL	Very high speed integrated circuit Hardware Description Language
SR	Shift Register
CLB	Configurable Logic Block
DSP	Digital Signal Processors
DCT	Discrete Cosine Transform
LZD	Leading Zero Detector

CHAPTER ONE

INTRODUCTION

1.1 Motivations, aims and objectives

The research presented in this thesis focuses on the efficient and fast implementation of the Hough Transform (HT) [1] on Field Programmable Gate Array (FPGA) using the logarithmic number system (LNS) and more specifically using the Hybrid-LNS [2],[3]. The HT has historically been the standard method used for the detection of straight lines and edges in binary images. Since its invention by Hough in 1962 [1] the transform has been extended to the General Hough Transform (GHT) [4] enabling the detection of more generic shapes such as curves, circles and parabolas and making it an extremely useful tool in many image processing applications. However, in this thesis only the location of straight lines on binary images will be examined. The HT is known for its computational intensive requirements and the implementation of it, even with today's FPGA, is problematic due to the limited resources in terms of memory, computational resources and speed. This is the reason for looking at alternatives architectures for implementing the HT using multiplierless techniques, such as the Hybrid-LNS. The aim of this research is to implement the HT with as less as possible resources in hardware using FPGA's. For achieving this the Hybrid-LNS will be used, the Canny edge detection method will implemented in hardware with the use of an synchronization circuit, and a LUT based accumulator cell will be designed for minimizing the memory requirements of the HT output.

1.2 Hough Transform

The HT uses the concept of point-line duality to locate lines in an image. A point P in an image can be defined using a pair of coordinate (x, y) or in terms of a set

of lines passing through it. When considering a set of collinear points P_i and generating a set of possible lines that pass through each point it becomes clear that there is just one line that is common to all of the sets. It is therefore possible to find the line containing all the points P_i by removing all lines without multiple "hits". Although the transform can be used in higher dimensions the main use is in two dimensions.

The HT is an important image processing operation, where its methods offer robustness against noise but there are some problems involved like high computing cost and extreme memory requirements [5]. Also, the transforms can be influenced by errors of discretization/quantization of continuous space. Because of these problems the performance of the standard HT needs to be improved. There are numerous, in-depth, and varying publications on the HT available in the literature. More precise HT using variable filters and transforms determining efficient sampling intervals have been used as well, in order to examine the quantization errors. Even though the HT has been around for many years and a number of algorithms have been developed it still remains a significant bottleneck in many image processing applications (see Chapter 2) [6].

1.3 Logarithms and Hough Transform

The main advantage of the LNS is that multiplication and division in the linear domain is simplified to addition and subtraction in the log domain respectively, as it can be seen from equations (1.3:1) and (1.3:2).

$$\log_b(A \times B) = \log_b(A) + \log_b(B) \quad \textbf{(1.3:1)}$$

$$\log_b(A \div B) = \log_b(A) - \log_b(B) \quad \textbf{(1.3:2)}$$

On the other hand, the LNS is unable to represent all numbers directly and additional information is required to represent the sign of the number and the special case of x = 0.

Implementing addition and subtraction in the logarithmic domain [7] is much more complicated as can be seen from (1.3:3) and (1.3:4).

$$\log_b(A+B) = i + \log_b(1+b^{j-i}) \quad \textbf{(1.3:3)}$$

$$\log_b(A-B) = i + \log_b(1-b^{j-i}) \quad \textbf{(1.3:4)}$$

Where: $i = \log|A|$ and $j = \log|B|$

Those are not straight forward equations, as the non linear functions (1.3:5) and (1.3:6) must be evaluated, where are normally implemented using a Look Up Table (LUT).

$$F(r = j-i) = \log_b(1+b^r) \quad \textbf{(1.3:5)}$$

$$F(r = j-i) = \log_b(1-b^r) \quad \textbf{(1.3:6)}$$

An alternative solution to this problem is the use of the hybrid-LNS where multiplication is performed in the log domain and addition performed in linear domain. This removes the necessity for implementing the non-linear function and LUT methods are used to translate between the log and linear domains. The size of the LUT grows exponentially with the number of bits of resolution and becomes prohibitively large when more than 16 bits of accuracy is required. For image processing, where the image data is limited to 8-10 bits of resolution the LUT requirements are acceptable and both LNS and Hybrid-LNS arithmetic are suitable for processing gray scale images.

The obtained results (see Chapter 7) indicate that implementing the Hough transform using the Hybrid-LNS, an efficient, low-power and low cost solutions can be achieved compared to the equivalent linear implementation using custom ASIC or FPGA technology.

1.4 Edge Detection and Hough Transform

To extract line segments or any geometric structure from a gray scale or colour image using the HT, the image need to be transformed to a binary one using thresholding or any other edge detection method. The binarized images are the edge maps of the original grey level or colour image.

Edges can be detected by applying a high pass frequency filter in the Fourier domain, or by convolving the image with an appropriate kernel in the spatial domain. In practice, edge detection is performed in the spatial domain, because it is computationally less expensive and often yields better results. Since edges correspond to strong illumination gradients, the derivatives of the image are used for calculating the edges.

Canny edge detection [8] is considered to be the ideal edge detection algorithm compared with others, (Sobel, Prewitt) because it produces very sharp and thin edges. As it is a method of special interest, it will extensively be described and implemented in hardware for the scope of this research.

The Canny edge detection uses a multi-stage algorithm to detect a wide range of edges in images. It first smoothes the image to eliminate the noise and then finds the image gradient to highlight regions with high spatial derivatives. The algorithm then tracks along these regions and suppresses any pixel that is not at the maximum (nonmaximum suppression). The gradient array is now further reduced by a process called hysteresis. Hysteresis is used to track along the remaining pixels that have not been suppressed. It uses two thresholds where if the magnitude is below the first threshold, it is set to zero (made a nonedge). If the magnitude is above the high threshold, it is made an edge. Finally, if the magnitude is between the 2 thresholds, then it is set to zero unless there is a path from this pixel to a pixel with a gradient above the high threshold.

1.5 FPGA and Hough Transform

In the decade and a half since the introduction of the first commercial FPGA, these devices have grown in complexity from a few hundred to millions of gates of programmable logic. Once used primarily as "glue logic," FPGAs today are key system-level components packed with features such as on-chip memory, clock management capability and programmable support for high performance I/O signaling standards. FPGAs allow equipment makers to significantly reduce their time to market. And because they are manufactured on the most advanced

semiconductor process technologies available, FPGAs offer levels of design flexibility, performance and logic density that make them a viable and cost effective alternative to traditional fixed-logic ASICs. More important, FPGAs can be reprogrammed even after an end system has been deployed at a customer's site. As a result, FPGAs technology is opening up a new area of equipment design that allows for hardware upgrades over a network. This promises to reduce equipment maintenance costs, extend the life cycle of products and create new sources of revenue for manufacturers by allowing them to add new features and capabilities remotely to installed products [9]. Using FPGAs technology fast re-configurability, either partially or totally, can be achieved and a designer can meet the required performance with a minimal amount of resources [10] [174].

As mentioned in section 1.1, in order to compute and implement the HT on an FPGA it is quite complex in terms of memory, computational resources and speed [5]. Minimization of possible multiplication and look-up table's utilization using the LNS is the solution to the problem.

Instead of using a sequential algorithm it will be important to explore methods and techniques that exploit the inherent parallelism of the logic blocks available on an FPGA. The research will compare and control existing algorithms and their performance when implemented on an FPGA and explore new methods for improving this performance. The aim is to provide full 'real-time' operation with mega-pixel size digital images up to 25 frames per second using Xilinx Virtex4™ architecture. A block diagram of the system is showing in Figure 1.5-1.

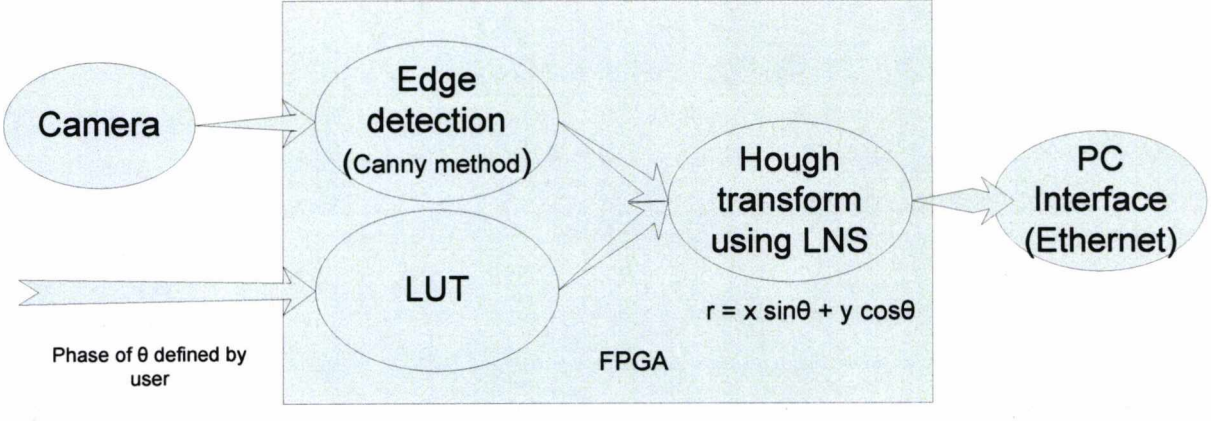


Figure 1.5-1: Block Diagram of the System

The camera being used for capturing the test images is a Pulnix 1Mpixel camera operating at 25 frames per second. The interface to the camera is based on the CameraLink signals where the control signals are PVAL (Pixel Valid), HVAL (Line Valid) and FVAL (Frame Valid). These signals make it possible to determine the size of the active image and the valid pixels.

As it can be seen from Table 1.5-1 [11], there is a large range of camera manufactures, operating at higher resolutions (up to 10 Mpixels). However, this increase in resolution often results in a reduction of frames per second.

Table 1.5-1: Resolution Vs Frames Per Second [11]

Manufacturer	Resolution (Mpixels)	Frames per second
Omnivision	9	8.9
"	8	10
"	5	7.5 & 15
"	3	15
"	2	15 & 30
"	1	30
CIS Americas	5	15
Pulnix	2	15
PixelLink	2	20 & 15
"	3	12
"	6.6	5
Kodak	10.3	15
"	4	13

1.6 Thesis Organization

Chapter 2 introduces the HT and provides relevant background information about it, as well as the reasons why this transform is so widely accepted in the image processing world. A mathematical approach to the transform is given to understand its operation, as well as the basic theory and underlying principles are

presented. In order to support the research, the deficiencies in other transforms and representations in this field are outlined. As the HT has been proved a valuable tool in large range of computer vision problems in fields such as industrial automation, robotics, biomedicine and satellite observation of Earth, several applications of the HT are presented. As one of the main characteristics of the HT is the independent simple calculation of every feature in an image, parallelism can be successfully achieved. For that reason, the different configuration types for parallel processing are described in detail, and the between differences are introduced.

Binarization of the image is required to successfully extract line segments from an image using the HT, by any edge detection method. There are many ways to perform edge detection and there are several edge detection methods. The Canny edge detection algorithm is known to many as the optimal edge detector. In Chapter 3, an extensive description of the Canny method, as well as, all the stages for the implementation involved is presented. In the same chapter, an introduction to digital logarithms and the usefulness in hardware-based arithmetic, along with the history of the hardware implementation of logarithms is presented in the form of a literature review, to demonstrate the possibilities and use of logarithms for this purpose. This also shows the most modern techniques and implementations in this area.

In Chapter 4, a detailed hardware implementation of the Canny edge detection method is presented. A software and a hardware version of the method has been generated, where a comparison between the two versions is presented, using floating point and fixed point arithmetic. Also, a novel flexible LUT based synchronizing circuit for 2-D imaging filters of variable dimensions is introduced.

As hardware implementations are the main focus for this thesis, the FPGA is introduced and the Xilinx Virtex4™ FPGA is described in Chapter 5. A comparison of alternative technologies that may also be used to implement the transform and image compression system is described. Also, the most advanced arithmetic resources available on the FPGA are detailed.

Chapter 6 introduces the Hybrid Logarithmic Number System (Hybrid-LNS) and the implementations in hardware. Details are given along with a worked example, for clarification, of the implementation of logarithmic arithmetic appropriate for the implementation of the HT proposed in the subsequent chapter.

Chapter 7 presents a new investigation into the effect of using logarithmic arithmetic on the Standard Hough Transform (SHT) algorithm for hardware implementation using Matlab® simulation. The methods investigated in Chapter 6 will be applied in this chapter. It has been shown that the HT can be implemented on technology as the FPGA's [12], [13], with the use of multipliers. This chapter describes how, by using logarithmic arithmetic, the need for multipliers is eliminated, while precision of the algorithm is maintained. Finally, the implementation of the SHT on FPGA using Hybrid-LNS arithmetic is presented, as well as, a comparison with the linear SHT is made. The relative simplicity of the structures presented, indicates that it is feasible to implement multiple elements operating in parallel using just the basic CLB elements available on a typical FPGA fabric and leaving the DSP slices and Block RAM free for other functions in the image processing chain. Depending on the overall throughput, results shows that it is possible to process data from a 1024 x 1024 pixel camera at a rate of up to 25 frames per second.

A LUT based accumulator cell that can be used as part of the HT architecture for storing the calculated Parameter Space Function (PSF), derived from a binary image is presented in Chapter 8. The accumulator cell uses the distributed memory elements available on a Xilinx FPGA fabric to store the intermediate results prior to passing them on for post processing and feature extraction. Parallel implementation can be achieved where significantly large images (megapixel) are being processed. In the same chapter, the complete system is presented in terms of hardware requirements, and a comparison between the different stages is taking place.

A review of all the research presented in this thesis, along with a summary of findings is presented in Chapter 9 as a conclusion. Directions for further research are also suggested.

1.7 Research Contributions

The five main areas of this research are presented in:

- Chapter 4 for the implementation of the Canny edge detection method on hardware.
- Chapter 4 for the design of a novel generic synchronization circuit for windowing operations.
- Chapter 7 for the implementation of the HT using Hybrid-LNS.
- Chapter 8 for presenting methods for storing large arrays on FPGA's.
- Chapter 8 for the combined edge detection and HT on an FPGA.

Chapter 4 describes the Canny edge detection method and the hardware implementation of it is presented. As a moving window operator forms the basic implementation of the Canny algorithm, and a number of pre-calculated steps are required a novel synchronization circuit architecture was designed. The circuit uses LUT resources available on FPGA devices as variable length shift registers. The synchronization circuit architecture is one of the main contributions of this thesis.

Another contribution can be seen in Chapter 7. Using the most advanced technology available on the latest FPGA devices, Chapter 7 describes a novel implementation of the HT where the Hybrid-LNS method is used. The implementation was presented at the SPIE conference in 2008 [14].The proposed HT using Hybrid-LNS is designed to use less logic, operate at higher speeds, while precision of the algorithm is maintained.

As memory requirement is one of the main drawbacks of the HT algorithm, Chapter 8 presents a novel LUT based accumulator cell architecture for storing the calculated PSF derived from a binary image. The accumulator cell uses the distributed memory elements available on a Xilinx FPGA fabric to store the intermediate results of the HT prior to passing them on for processing and feature extraction. This is the last, but not least, main contribution gained out of this thesis.

CHAPTER TWO

HOUGH TRANSFORM LITERATURE REVIEW

2.1 Introduction

This chapter provides an overview of the literature and relevant background information about the HT, and the reasons why this transform is so widely accepted and used, thus, providing the basis for this thesis. However, there are numerous, in-depth, and varying publications on the HT available in the literature. Therefore, it is necessary to categorise these publications in an appropriate manner. Hence, a mathematical approach to the transform will be introduced in this chapter, in order to aid the understanding of its basic operation. Then, the basic theory and underlying principles, as well as the deficiencies in other transforms and representations in this field, will be outlined, in order to support the research introduced and described in subsequent chapters of this thesis. The research review presented in this chapter will follow a chronological order.

2.2 The Hough Transform Method

The HT was originally proposed by Hough in 1962 [1] and has become a standard tool in image analysis that allows recognition of global patterns in an image space, by recognizing local patterns (ideally a point) in a transformed parameter space. The basic idea of this technique is to locate curves in an image that can be parameterized: e.g. straight lines, polynomials or circles, in a suitable parameter space. Its main use is in two-dimensional spaces to find straight lines, centers of circles with a fixed radius, or parabolas, although the transform can be used in other dimensions. The research presented in this chapter will focus only on location of straight lines on binary images.

Consider the equation of the straight line in slope-intercept form shown in (2.2:1) for different values of a and b and a point (x_i, y_i).

$$y_i = ax_i + b \qquad \textbf{(2.2:1)}$$

Infinitely many lines pass through this point and all satisfy the equations of the straight line. Solving the straight line equation as (2.2:2) and considering the parameter space yields the equation of a single line for a fixed pair (x_i, y_i).

$$b = -ax_i + y_i \qquad \textbf{(2.2:2)}$$

A second point (x_j, y_j) also has a line in parameter space associated with it, and this line intercepts the line associated with (x_i, y_i) at (m', c'), where m' is the slope and c' is the intercept of the line containing both (x_i, y_i) and (x_j, y_j) in the xy-plane. Generally, all points on this line have lines in parameter space that intercept at (m', c') as Figure 2.2-1 shows below [16].

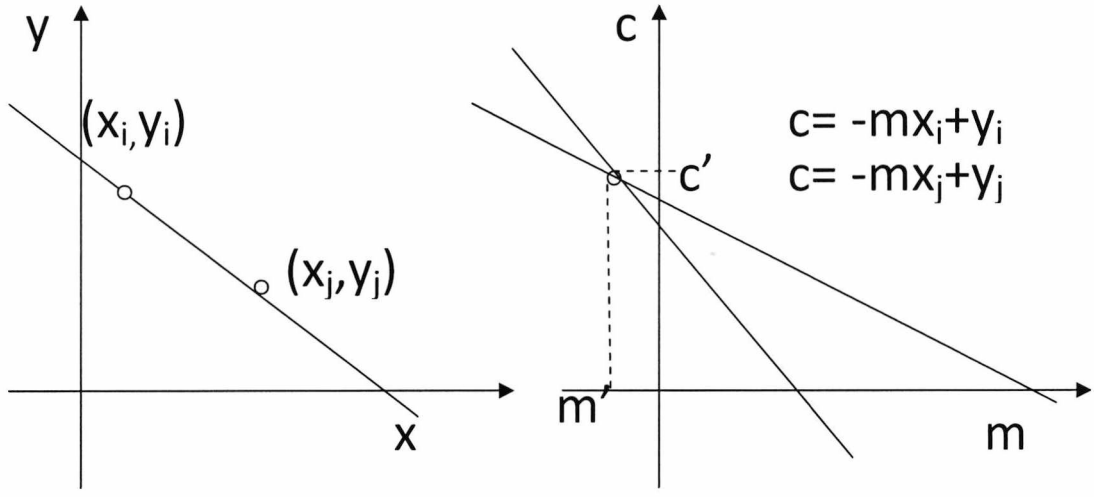


Figure 2.2-1: Left Figure: xy Plane, Right Figure: Parameter Space

A practical difficulty with this approach is that the slope of the meaning line approaches infinity as the line approaches the vertical direction. One way to overcome this difficulty was suggested by Duda and Hart [6] where they used the normal representation of a line which is shown in (2.2:3) where x and y are the co-

ordinates of a point on the line, p is the perpendicular distance of the line from the centre of the image and θ is the angle between the perpendicular to the line and the x-axis [17]. (Figure 2.2-2) The mapping of (*x, y*) points into the 2D (ρ, θ) parameter space is achieved by sampling the θ axis and then calculating, using equation (2.2:3), the corresponding ρ value. As θ is varied a sinusoidal curve is generated whose amplitude and phase are determined by the image points (*x, y*) [18].

$$x \cos(\theta) + y \sin(\theta) = \rho \qquad \textbf{(2.2:3)}$$

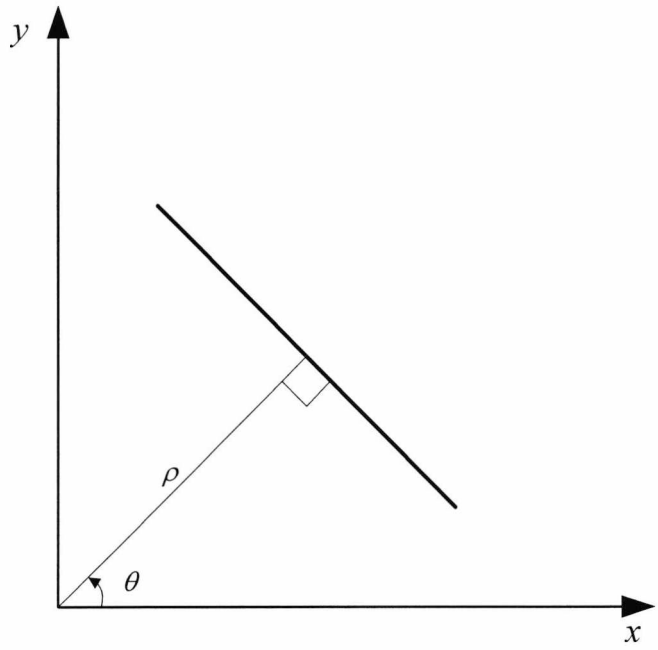


Figure 2.2-2: Parametric Description of a Straight Line

2.3 Implementing the Hough Transform

The HT algorithm uses an accumulator array (accumulator bins) to detect the existence of a line. The dimension of the accumulator is equal to the number of unknown parameters of the HT problem. For example, the linear HT problem has two unknown parameters: m and c. The two dimensions of the accumulator array would correspond to quantized values for m and c. For each pixel and its neighborhood, the Hough transform algorithm determines if there is enough evidence of an edge at that pixel. If so, it will calculate the parameters of that line, and then look for the accumulator's bin that the parameters fall into, and increase the

value of that bin. By finding the bins with the highest values (peaks), typically by looking for local maxima in the accumulator space, the most likely lines can be extracted, and their (approximate) geometric definitions to be read. The size of the high value bins is a function of several factors such as the number of points composing the line, the number of other points in the image and the choice of parameter bin size. The size of parameter bins is usually chosen to correspond to the required precision of parameter space [16].

For the performance of the HT algorithm, the following steps required.

i) Define $\{m_{min}, m_{max}\}$ and $\{c_{min}, c_{max}\}$ as the expected ranges of m and c respectively as it is shown in Figure 2.3-1. The cell at coordinates (i,j), with accumulator value A(i,j), corresponds to the square associated with parameter space coordinates (m_i, c_j) [16].

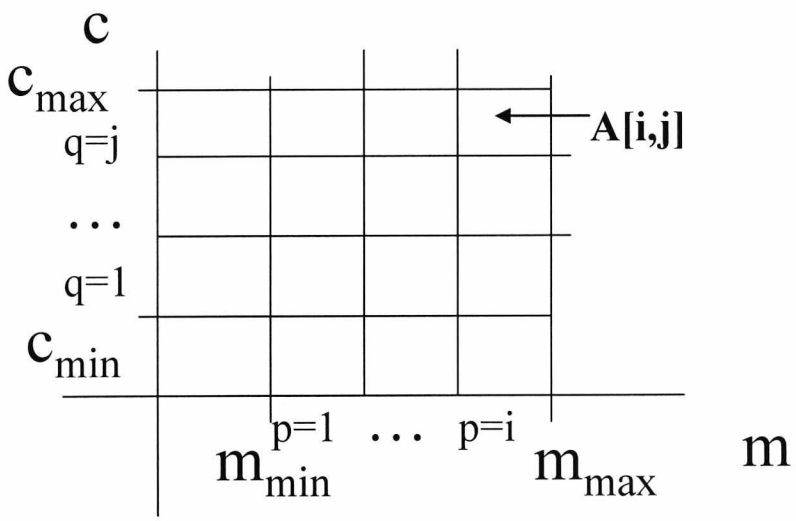


Figure 2.3-1: Subdivision of the mc-Plane Into Cells

ii) Set A $[i, j] = 0$ for all i, j to initialise [16].

iii) For each point (x', y') of interest in (x, y) space

Set m_p = each subdivision in range $\{m_{min}, m_{max}\}$ and solve for the corresponding c_q using the equation $c_q = -x'm_p + y'$ to get corresponding c-value. The resulting c_q are then rounded off to the nearest allowed value in the m-axis.

Then A [*p*, *q*] <= A [*p*, *q*] + 1

Repeat for all points of interest.

iv) After completion, a value of M in A [*i*, *j*] corresponds to M points in the (x, y) plane lying on the line $y = m_i x + c_j$ [16].

The accuracy of collinearity required for the points is established by the number of subdivisions in the (m, c) plane.

2.4 Advantages and Disadvantages of the Hough Transform

There are several positive aspects that render the HT an important image processing tool. First, parallel processing can be achieved as each image pixel can be implemented independently. More than one processing unit can be used and as a result the HT algorithm can be useful for real-time applications. Second, pixels lying on one line need not all be adjacent. This can be very useful when trying to detect lines with short breaks in them due to quantization noise, or when objects are partially occluded. Occlusion is a serious problem for most other detection techniques such as convolution or noise filters, but the HT overcomes this problem by using the size of a parameter peak, which is directly proportional to the number of matching boundary and template points. Third, the HT offers robustness against noise produced by poor image segmentation, or from the boundaries of shapes other than those searched for in an image. Finally, the HT can process several straight lines at the same time, in the same image, as each line produces a distinct peak in the accumulator array [5].

However, the HT can give misleading results when objects happen to be aligned by chance. This clearly can become a disadvantage, where the detected lines are infinite lines described by their (*m*,*c*) values, rather than finite lines with defined end points. However, the main disadvantage is the large storage and computational requirements. The computational attractiveness of the HT arises from subdividing the parameter space into accumulator cells as it is shown in Figure 2.3-1 [16]. In order to accurately represent the result into a continuous parameter space, the number of the accumulator cells, in both dimensions *m* and *c*, must be large. This

means that the HT, in terms of computing storage, requires m x c bins. Moreover, a large θ means a large number of evaluations of equation (2.2:3) to accumulate the transform. These two problems have had a combined impact on the production of more efficient implementations of the basic HT idea.

2.5 Early Development of the Hough Transform

The HT was first introduced by Paul Hough in 1962 [1], as a method and means for recognizing complex patterns rather than shapes in images, even though Rosenfeld [17] was the researcher that noted its potential advantages as an image processing algorithm and made the HT obvious to the image processing world. Hough's invention was based on studying complex patterns formed by particle tracks in pictures through a viewing field. More specifically, studying particle tracks in pictures taken through a bubble chamber. The slope-intercept parametric representation of a line was used, but with the main drawback of generating an unbounded parameter space. To bypass the problem, Duda and Hart [6] suggested that a straight line can be usefully parameterized by the length, ρ, and the orientation, θ, of the vector to the line from the image origin. Using the (ρ, θ) parameters mean, that image's points map into sinusoidal curves in a two-parameter space. This procedure can yield unsatisfactory results when pictures contain random noise that cannot be removed. Cohen and Toussaint [20] modified the Duda-Hart procedure, by studding the distribution of background counts for random noise points in finite size images, and this compensates for noise whether the distribution is known or not. O'Gorman and Clowes [21], using the gradient direction in the HT, managed to recover straight lines in digitized pictures containing polyhedral, and Van Veen and Groen [22] investigated the influence of the discretization effects on the HT, in both the image and parameter spaces.

In 1981 Deans [23] pointed out that the HT is simply a special case of the Radon transform, which has been known since 1917, and, therefore, accompanied by a large amount of theoretical and mathematical literature associated with it. The Radon transform has been studied in relation to computer-aided tomography and on a two-Euclidean plane is defined as shown in equation 2.5:1.

$$R(\rho,\theta) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x,y)\delta(p - x\cos(\theta) - y\sin(\theta))dxdy \qquad \textbf{(2.5:1)}$$

where δ is the Dirac delta function. The delta function term forces integration of *f (x, y)* along the equation of the straight line (2.2:3). For shapes other than straight lines the δ function argument can be replaced by a function which forces integration of the image along contours appropriate to the shape [5].

Subsequently, several methods of HT were introduced in order to improve the computational efficiency and practicability for use in real time image analysis tasks as well as speeding up the process as much as possible [170]. Nevertheless those methods are described below, it is worth mentioning that the purpose of this research was to prove that fine processing can be achieved all at once, by using different arithmetic structures rather than multiresolution or *coarse to fine* techniques. Nowadays, even though all the calculations required for implementing the different methods of the HT are possible to achieve with the processing power available, by using the logarithmic approach (a detailed discussion will be presented later on in this thesis), power consumption and logic resources can be reduced significantly.

2.6 Hough Transform Methods

2.6.1 Generalized Hough Transform

In 1981, Ballard [4] developed a HT which does not decompose the image into its component features, for example, straight lines, but rather extract the shape in its entirety. He called it the Generalized Hough transform (GHT). It could efficiently find arbitrary shapes for any orientation or any scale, and that was achieved with the use of directional edge information. Few graphic examples of the information used by the GHT are shown in Figure 2.6-1. Lines indicates gradient directions Each boundary point was stored as a vector relative to some reference point; that being the distance, r, and the direction, θ, of the line connecting the boundary point and the reference point. Entries in a list are indexed by the local edge direction at the

boundary point. The resulting list is called an R-table. As the image points are only compared with a subset of the R-table entries, indexing by edge direction decreases the computational expense of the method. Consequently, complex shapes can be decomposed into simpler shapes and it is possible to have a shape representation which can accommodate this type of structural description [5], [24].

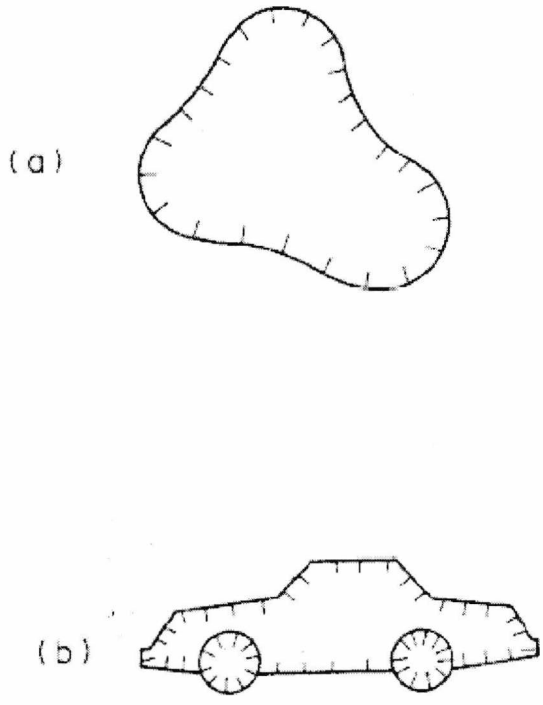


Figure 2.6-1: Shapes Detected with the GHT. a) Simple Shape b) Composite Shape. [4]

2.6.2 Fast Hough Transform

Several authors have realized that the memory requirements and computational load of the HT can be reduced using a intelligent iterative *coarse to fine* accumulator technique. Such a technique involves examination of the accumulator array at various scales and an attempt is made to evaluate it in detail only in those areas having a high density of counts. The fast Hough transform (FHT) of Li *et al.* [25] uses this technique. The FHT uses a multidimensional quadtree which maps the image points into hyperplanes. It recursively divides the parameter space into a nested hierarchy of hypercubes from low to high resolution and perform the HT only on the hypercubes with votes exceeding a selected threshold. Figure 2.6-2 shows an example of a quadtree where the entire parameter space is considered first, then the upper right sub-quadrant and so on. Far from growing exponentially, the search

converges rapidly on the accumulation point. The decision on whether a hypercube receives a vote from the hyperplane depends on whether the hyperplane intersects the hypercube. The advantage of using hyperplanes as opposed to an array of accumulators is that the intersection between planes and parameter cells can be efficiently computed using an incremental test, where only additions and shifts are required to implement the test and as a result the computational cost scales linearly with the dimensionality of the parameter space. On the other hand, using the incremental intersection testing method, each quadrant must store the distance of every image feature from its center. If there are many image features this can represent a large overhead. In addition, the rigid quadtree decomposition of the parameter space means that lines generating peaks which cross the boundaries of quadrants may be missed by the processing [24],[26].

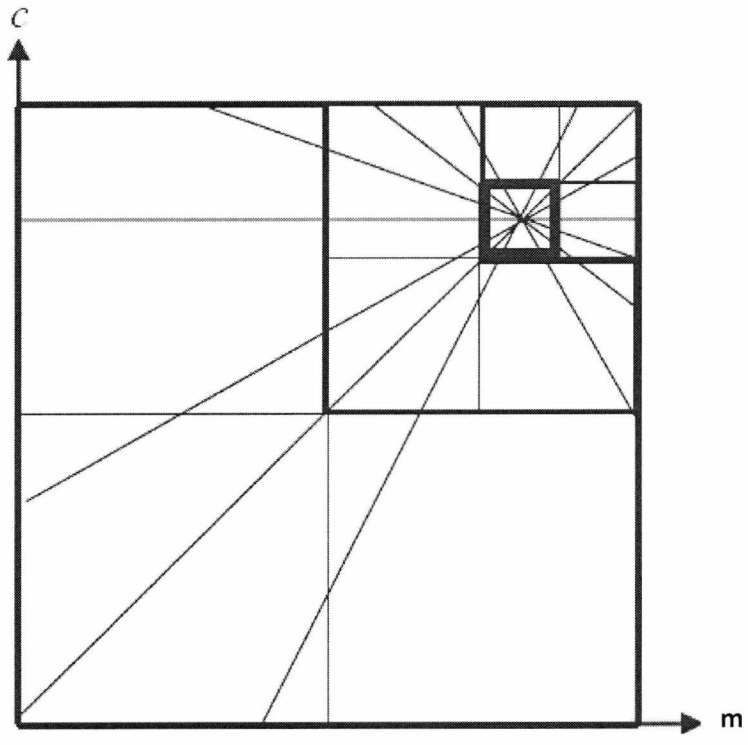


Figure 2.6-2: Example of a Quadtree Search Used in the FHT

2.6.3 Adaptive Hough Transform

A more flexible variation of the *coarse to fine* technique was used by Illingworth and Kittler [27] in their implementation called the Adaptive Hough Transform (AHT). The AHT avoids the problems of calculating the HT using a large number of cells, by implementing a multiresolution peak finding search of the parameter space, using a small fixed sized accumulator. A 9 x 9 accumulator initially covers the full range of parameters. After accumulation, candidate peaks are identified by thresholding. The parameter limits covered by the accumulator array are then adjusted to focus on the most prominent candidate peaks. Iteration of the cycle of the HT accumulation, peak analysis and parameter limit redefinition should lead to rapid and accurate determination of parameter peak locations. The performance of the AHT was tested using simple images containing digital line and circle segments. However, serious interpretation problems have been found when the method is applied to complex images. In particular, the method can not identify lines reliably, unless the number of lines is small and they are long with respect to the size of the image. Additions to the basic AHT have been made for successful implementations. Cao *et al.* [28] have modified the AHT to include a "labeling" technique and have successfully implemented the method in parallel. Onda *et al.* [29] have implemented a modified AHT algorithm in which the range of parameters calculated is constrained by the gradient of each pixel, resulting in a sharper local peak. Finally, Berger and Khosla [30] have implemented a modified AHT in combination with a weighted least squares algorithm where a weight is computed for each data point based on the residual error of the previous estimate.

2.6.4 Fast Adaptive Hough Transform

A very similar approach of the AHT was implemented by Haule and Malowany, the Fast Adaptive Hough Transform (FAHT) [31]. Here again, the parameter space is investigated at several resolutions, but is only evaluated in fine detail in those regions where high densities of image points occur. The differentiation from the AHT is due to the degree of freedom allowed in the redefinition of parameter limits, i.e. the degree of flexibility in the placement and choice of window shape which

defines the range of parameter under study. In the FAHT, a complex strategy to analyze the parameter space is employed at each resolution, and this has a result that more appropriate parameter limits for subsequent processing can be defined. The range of parameters can increase, decrease, or remain the same and translation of the parameter limits by non integral multiples of the cell distance is permitted. Changes in resolution can be made independently in each of the parameter dimensions every time the parameter limits are redefined. In summary the advantage of the FAHT is that the size, shape and positioning of the parameter windows is very flexible and is determined by the data itself rather than being imposed as a result of the choice of data structure and initial parameter limits [24].

2.6.5 The Binary Hough Transform

The Binary Hough Transform, (BHT) [32], [33] is a technique for straight line detection in binary images based on a modified slope/intercept parameterization of the straight line. To implement this method the dimensions of both the image and the transform space should be integer's power of two. This implies that the calculations required for the parameter determination can be executed using only adders and delay-elements without any multipliers (multiplications are performed by binary shifts) and the sampled slope and intercept can be represented in full precision for fixed point arithmetic, where integer arithmetic is used without rounding errors. As this is the most efficient way for a computer to execute calculations, the BHT gives a significant saving in computational time and hardware resources. In addition, the BHT can be effectively implemented in high-throughput systolic array architectures, as showed by Costa and Sandler [34], and it has been experimentally verified that better accuracy for determination of parameters and effectiveness than the standard HT with normal parameterization can be achieved. The main drawback of the technique is that it requires four two-dimensional accumulator arrays for straight line detection [24], limiting the hardware resources. Another such algorithm, which generates the HT by using only the incremental addition operations, and without the calculations of trigonometric functions and multiplications, was presented by Koshimizu and Numada [35] and is called the fast incremental Hough transform (FIHT).

2.6.6 The Dynamic Combinatorial Hough Transform

An interesting variation of the HT, the Dynamic Combinatorial Hough Transform (DCHT) was proposed by Leavers *et al* [36]. It uses information present in the location of the feature points in an image and as a result only one dimensional accumulation of evidence required to determine the parameters associated with a given shape. The co-ordinates of the edge points in an image are listed in the order of their appearance. A point of the list which is randomly selected is fixed and paired with all other points from the list. These points are then accumulated in a θ histogram. If n of the points in the list are co-linear with the first point, it results in a peak of value n at the θ value of this line in the θ histogram. After the peak detection, the value of ρ from equation (2.2:3) may be calculated using the (x, y) coordinates of the first point. All the n co-linear points are removed from the list, and the process is repeated. In each of the following passes the first point in the list is combined with all the other points, and a new θ histogram is generated. The algorithm continues until all the edge points contributing to straight line segments have been removed from the list. The DCHT is a significant improvement of the SHT, as it is computational less intensive algorithm and much more efficient in memory utilization. It uses a 1-dimentional accumulator instead of a 2-dimentional accumulator as in other HT methods [37]. Leavers [38] later on generalized the DCHT to the Dynamic Generalized Hough Transform (DGHT) where only a fraction of the coordinates of the edge points is accumulated, and these points are chosen probabilistically. In addition, the DGHT incorporates a mechanism for selecting the first point. Points around the candidate first points are summed first in a horizontal and then in a vertical direction. If either sum exceeds a threshold then the point is accepted.

2.6.7 Connective Hough Transform

A common problem of the HT is that it can give misleading results when objects happen to be aligned by chance. This is known as the connectivity problem [6]. It arises because the accumulator counts only the number of points that share the same

parameters. These points may not be connected with each other. Thus the position of a best-fit line can be distorted by the presence of unrelated figure points in another part of the image [6]. A novel method called the connectivity Hough Transform (CHT) [37], aims to solve this problem. The implementation uses the dynamic combinatorial Hough transform (DCHT) method of calculation and accumulation. Due to the nature of the accumulation, a unique line segment is detected for each point with no redundancy. Using the CHT, all feature points as they are detected, are tested for connectivity with respect to neighbouring points. Only two 1-dimentional accumulators are used which minimize the memory complexity of the method and since disconnected points are not accumulated, the method achieves a significant speed-up compared with the SHT. For a 128 x 128 binary image, it was found that 40 seconds required by using the SHT, where only 16 seconds required using the CHT [37].

2.6.8 The Hierarchical Hough Transform

Two main constraints have to be taken into consideration in order to obtain a useful line finding scheme, when complex images or inaccurate feature measurements are under test, collinearity and proximity. The question is how strong the proximity constraint should be. In the SHT, the proximity can be chosen arbitrarily in the post processing step. An alternative method for imposing proximity constraints, which is based on the use of a hierarchical structure, is the Hierarchical Hough Transform (HHT) [39]. It is based on applying HT type algorithm at all levels in a pyramid structure. At the bottom level of the pyramid short line segments are detected by applying the HT algorithm in small sub-images, i.e. 16 x 16 pixels. This is a robust mechanism as the accumulator array requires a small and easily interpreted structure. The advantage of such a structure is that the range of possible ρ values is directly proportional to image size and as a result in a small sub-image the number of ρ bins is small. Also, the intrinsic accuracy with which angles can be estimated in a small sampled image window is poor and therefore it is reasonable to divide the full θ range into only a few bins. After low level line segments have been found, the method proceeds bottom up from this description by combining line segments within local neighborhoods into longer lines. The grouping process of the line segments is

also based on a HT algorithm and involves a relatively sparse accumulator array which can be represented as a linked list rather than an array data structure. Line segments which are found to have common pixels with the neighborhood pixels move up the hierarchy and take part in grouping at higher levels while lines with no common pixels at a particular level terminate [26], [39].

2.6.9 The Probabilistic Hough Transform

Two are the main stages where the SHT can be implemented. The first is an incrementation stage, where the accumulators corresponding to cells that the sinusoid of equation (2.2:3) intercepts are incremented. The computational time of this stage depends on the number of edge points in an image. The second stage is an exhaustive search for maxima in the accumulator array, where the computation time depends on the size of the accumulator array. As the number of cells in the accumulator array is much smaller compared with the number of edge points in an image, the incrementation stage usually dominates the execution time of the SHT algorithm. Kiryati *et al.* [40] proposed a new algorithm, the Probabilistic Hough Transform (PHT), in which the image data are randomly sampled and only the sampled subset of image points is transformed. As the number of edge points in the incrementation process is much smaller, a significant computational saving is achieved. The key to successful application of the PHT is the dependence of the algorithm's performance on the fraction of the data that is used. The selected subset of image points is accumulated as in the SHT. Theoretical analysis and experimental results showed that even in the presence of distracting features, significant noise and errors in the coordinates of the image points, large computational savings are succeeded using the PHT. The algorithm deal with binary edge data and do not depend on computationally expensive global preprocessing of the image. A threshold effect exists concerning the number of false peaks found as a function of sample size. As the complexity of the image increases, the sample size also correspondingly increases [24].

2.6.10 The Weighted Hough Transform

Once the HT has been accumulated the pattern of counts in the accumulator array has to be analyzed to estimate the presence and location of local peaks. The most common method is to determine a threshold. Any accumulator cell with more counts than the threshold indicates a possible image segment. The threshold is chosen either using prior knowledge or it can be automatically selected by analyzing the distribution of counts in the array. However, analyzing the accumulator array is not always easy and may present difficulties. The reason for that relies to the fact that at any scale of discretization of the accumulator array, a high count in a particular cell may be generated because of several insignificant peaks rather than a single significant one, or a true peak may be split between several accumulator cells and not detected. The weighted Hough transform (WHT) of Ibrahim *et al.* [41] identifies peaks by using the image directly without the use of any threshold, and weighting the contributions to the transform space made by each pixel according to its value in that image. The greater the weight of a pixel, the stronger it's effect on line detection. For example, if the weight of a pixel is zero, it is equivalent that this pixel does not exist. That means there is no need for calculation of that pixel and this has a result of minimizing the overall computational cost [42].

2.6.11 Multiresolution Hough Transform

Another efficient implementation of the SHT has been proposed from Atiquzzaman [43], the Multiresolution Hough transform (MHT), where significant reduction of the computing time is achieved. Although the MHT is based on a *coarse-to-fine* iterative search, it has a few significant differences from the other HT methods. Firstly, multiresolution images and accumulator arrays are used in the iterations. A set of reduced-resolution images is generated from the original image. The HT is first applied to the smallest image using a very small accumulator array. Subsequent iterations use images and accumulator arrays of increasing sizes. Secondly, a logarithmic parameter - range reduction method that is suitable for the transform has been proposed by the author. Using this method a faster convergence and better stability is achieved. Finally, consideration of the discretization errors when

accumulating the parameter space has led to the use of a simple peak detection algorithm. Since the MHT uses such an algorithm, the computation time is significantly lower compared with other algorithms, if the time for peak detection is also taken into account. The MHT algorithm can not only be used for detection of straight lines but also can be generalized for patterns with any number of parameters [43].

2.6.12 Randomized Hough Transform

In 1990, a novel HT- like approach was proposed named the Randomized Hough Transform, (RHT) from Xu *et al* [44] for detecting curves from a binary image. The basic idea of the RHT algorithm lies in the fact that each parameter space point can be expressed with two points from the original binary image. If those points happen to be on the same curve, then the corresponding parameter space location is the same as the location of the maximum corresponding to the curve in the SHT [45]. In the SHT, a single pixel in the original image is mapped into a curve in the parameter space, whereas in the RHT, a pair of pixels is mapped to a single cell in the parameter space. This is the main difference between the SHT and the RHT algorithms. Assuming P is the set of edge points in an original binary image, (x, y) be the coordinates in the original image and (a, b) the two parameters of the curves. In the RHT, point pairs $(p_i (x, y), (p_j (x, y)$ are picked randomly from the set P and corresponding accumulator cells are incremented in the (a, b) space. For each chosen point pair in the original picture, only one parameter cell is accumulated [45]. As P stores any parameter point mapped from pixels of the image space, the RHT can implicitly observe the whole extent of an infinite parameter space. In addition, P stores the real value parameter points without discretization, and this has a result that RHT has an inherently high resolution. Another advantage of the RHT is that due to the frequent resetting of set P, the storage of the RHT is always kept quite small. Finally, in the SHT, as a pixel is transformed into a curve and all the cells lying on that curve are accumulated, the computing speed is constrained by the size of the accumulator array. In contrast, for the RHT, at each step, only one parameter point of P is updated where the computational cost is significantly decreased [24], [44]. An improved performance of the RHT is detailed in [46].

2.7 Applications of the Hough Transform

Computer vision is the branch of artificial intelligence that focuses on providing systems with the functions typical of human vision. To date, important applications have been generated by computer vision in fields such as industrial automation, robotics, biomedicine, and satellite observation of Earth. Today, more and more manufactures are using computer vision technology to improve their productivity and reduce costs. It integrates optical components with computerized control systems to achieve greater productivity from existing automated manufacturing equipment. The HT has proved a valuable tool in a large range of computer vision problems.

Examples of the use of the straight line HT are numerous. Inigo *et al.* [47] used the straight line HT to identify edges of roads and tracks in order to guide mobile robots. Dyer [48] used the HT to inspect the scaling accuracy of needle-type instruments. The method does not require the position, orientation, or size of the instrument to be known a *priori* and can be implemented in high speed hardware. Huang *et al.* [49] used the HT to detect seismic patterns in seismograms. The travel-time curves of the direct and the refracted waves are straight lines on a seismogram, whereas the travel-time pattern of the reflected waves is a hyperbola. Because of the cluttered and fragmented nature of the seismic data, HT is a particularly attractive method to be used.

Kushnir *et al.* [50] have suggested a method of the HT for the automated recognition of Hebrew characters. Almost all of the characters in the Hebrew alphabet consist of linear strokes. Peaks detected in the (ρ, θ) parameter space of a HT line detector were used as features in a pattern recognition system. The method was tested on a sample set of eighteen print-simulated alphabets and a recognition rate of 99.6% was achieved. Lin and Dubes [51] experimented with a straight line HT method, for counting ridges in an automated fingerprint analysis system. In such a system, the image had to preprocess with a predefined threshold and a small image window in which ridges appear as straight lines was selected. Each ridge produces a peak in the (ρ, θ) parameter space, where the counting of the peaks was relevant to the number

of ridges on the fingerprint. Determination of the threshold was difficult, undesirable lines due to noise were counted as ridges and the overall computational cost was high.

The linear HT has been used by Shibata and Frei [52], to detect and recognize targets in infrared imagery in real time. Edge finding operators were used to find the outline of the targets and to extract edge gradients and orientations of them, which then were mapped into a parameter space. The four boundaries of the target produced four sharp peaks in the parameter space. Cowart *et al.* [53], used frame-to-frame difference images in order to reject clutter and enhance the detection of moving targets. The tracks of those nonmaneuvering (i.e. constant velocity) targets appear as line segments that can be detected using the HT. Skingley and Rye [54], used the HT to detect faint lines in synthetic aperture radar (SAR) images. In their work, different post processing techniques have been applied and various problems have been addressed such as the detection of peaks and troughs in the transform space, the detection of line and points and the removal of false alarms.

The probability of detecting a line, in terms of the relative length and intensity has also been calculated. Shu *et al.* [55] have used a modified HT to detect edge lines in scanning electron microscope (SEM) images of VLSI (very large scale integration) resist patterns. Modifications to the basic line detecting HT were made in order to preferentially detect connected edge segments. This was achieved by constructing a bit array for each accumulator cell, where the incrementation of each cell was related to the status of the bit array. Also, a threshold was used which was adaptive to the orientation and location of the line segment under consideration.

A new formulation of the HT technique by Nixon, [56] aid to detect linear brightness variation areas within a picture. Once these areas are defined, they can consequently be suppressed, thus resulting in image restoration. Another application of the linear HT was used by Thomson and Sokolowska [57], to analyse cleavage cracks in minerals. The cracks in a crystal image are isolated and thresholded to create a binary image. The HT is used to detect the presence of alignments in these data.

The generalized HT has been applied and extended in many practical applications for the determination of motion parameters from a sequence of images. An approach to the segmentation of dynamic scenes containing textured objects moving against a textured background was presented by Jayaramamurthy and Jain [58]. Firstly, active regions in a frame which contain moving objects were found. Then, a HT technique used to determine the motion parameters associated with each active region. Experiments illustrated the efficiency of the approach for moving textured objects even in the presence of occlusion. Another use of the generalized HT for patterns of motion in a displacement field image produced by the time difference of two images was given by Adiv [59]. One of the major drawbacks in the methodology, the other being the computational expense, is that small objects will only produce small peaks in a parameter space. As a solution, Adiv proposed the use of coarse parameter resolution for the detection of large objects, and partitioning the image into smaller images, for the detection of smaller objects.

A method proposed by Kalviainen [60] has been introduced to calculate 2-D motion in a sequence of time-varying images. His method is based on the RHT and called motion detection using the RHT (MDRHT). The main idea is to randomly select a point pair of two consecutive images and compute the translation with them taking advantage the benefits of RHT [61]. Radford [62] used a 3-D parameter space to map motion parameters for translation and rotation. The (ρ, θ) parameters used in straight line detection are the two of those parameters, where the third is a length l, which measures the motion of an image feature between successive frames. By using this formulation, the focus of expansion of translational motion, as well as the centre of rotation for angular motion was achieved. Segmentation of a scene can be achieved by grouping points with similar motion parameters. Silberberg *et al.* [63] also used the GHT for recognizing 3-D objects in an image by matching a structural model of the object with information extracted from the image. A recognition model was constructed in such a way that can be identified and distinguished from one another based on the visibility and ease of detection of images of the model entities. Then, this recognition model is used by the GHT procedure in order to identify likely instances of the object model in an image, which would finally be verified by a top-down analysis. Henderson and Fai [64] also used the GHT for 3-D object detection

from a laser range-finding system. The success of the application was based on the correct choice of distinctive feature points.

The difference compared to the Silberberg approach is that Henderson first detects planar segments in the data and then matches them with the structural model. Kasif *et al.* [65] attempted a solution to the subgraph isomorphism problem, which involves finding if a given graph is an induced subgraph of a larger graph, by using the GHT. In their work, matching subgraphs derived from geographical maps was achieved even when map image was incomplete due to occlusion, low boundary contrast or other factors. Their method is suitable for parallel processing implementation on a network of simple processing elements.

Mirmehdi *et al.* [66] addressed the problem of real time label inspection using the HT in rectangular and oval labels as examples. The HT is applied in different images of product labels where peaks are detected. After peak detection, a list of all the straight lines in the image is obtained and is further examined against a known model label to determine the presence of faulty labels. General faults with labels are shifting from normal position, sticking at a tilted angle, tearing, folding or unreadable print. The HT was implemented in parallel where a high speed process can be achieved.

Kamat and Ganesan [67] used the HT to identify vehicle license plates from image frames for vehicle tracking purposes (Figure 2.7-1). The test image first passes through an edge operator stage (a modified Sobel operator). Next, the edge detected image is thresholded and the implemented look-up table based HT is applied to the thresholded image. Peaks are detected and are passed through the final stage, the line position extraction stage, where the detection of the license plate is achieved.

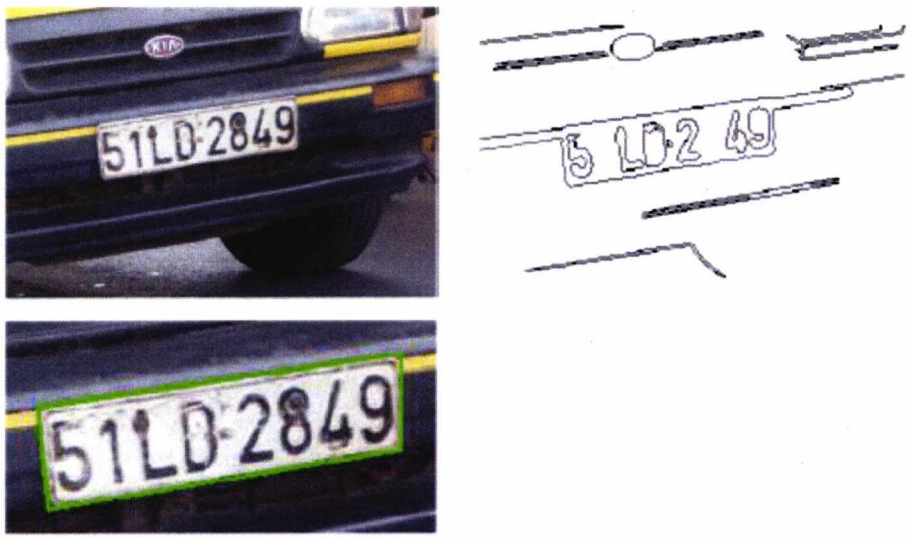


Figure 2.7-1: Vehicle License Plate Identification Using the HT [67]

He *et al.* [68] also used the HT algorithm to detect a car number plate skew without any object segmentation process. Leaving the robustness of the skew detection process unaffected, He proposed and successfully adapted to the HT different speed up approaches, such as the coarse/fine process, the sparse data process and the partial edge image process. Results showed that the speed up factor can be increased up to 20 depending on the complexity level of the car image.

The multiresolution HT, MHT, has been applied by Yu and Jain [69], for lane boundary detection. Lane detection is the problem of locating road lane boundaries without a *priori* knowledge of the road geometry. A lane boundary location can be very helpful in several applications such as intelligent vehicles, highway maintenance with intelligent cruise control, cambered power steering and automatic navigation. In order to minimize the computational cost and increase the accuracy of the lane detection, a MHT was used where the parameter space is separated into subspaces. The required parameters are estimated separately using a multiresolution strategy. Experimental results show that the method is very accurate, as it can be seen from Figure 2.7-2, on lane images in various situations including different lane marking conditions (single or double, solid or broken) and road environments (paved or unpaved, shadows or poor illumination).

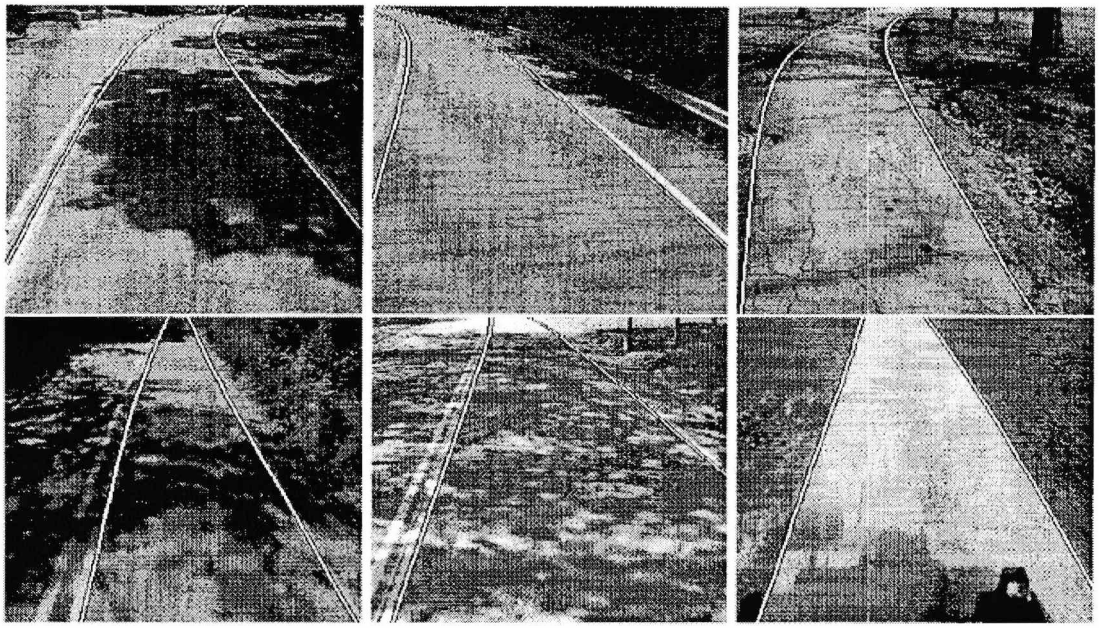


Figure 2.7-2: Experimental results of lane boundary detection. The detected lane boundaries are overlaid on the input gray scale images [69].

Muñiz *et al.* [70] have used the HT to decode linear barcodes. Even though linear barcodes are one of the oldest technologies related to the Automated Identification and Data Capture (AIDC), they are still made up of bars and spaces. For most applications a manually operated laser scanner is used, but in many others where the volume of information is too high, an automated approach has to be implemented. Muñiz applied the HT in sub-images which only contains a barcode, plotted the HT accumulator matrix generated from each image and finally analyzed it in order to decode the represented barcode. The method has been tested to read Code 39 and EAN-13 barcodes and has been compared with a commercial system (21351 barcodes out of 23038 were successfully read compared with a commercial system which managed to read only 11519). Sun and Willett [58] used the HT to detect long and weak chirp or linear frequency modulated (LFM) signals. Those kind of signals are important in many areas, such as radar, sonar and seismic. The chirp signals are weak (below -20dB), very long in order that they can be detected (e.g. 10 hours duration) and in addition to Gaussian noise, powerful tones are masking them. Due to the computational load and the ability to detect the chirp signals, HT is the most suitable compared with other methods tested by the authors. It showed that the HT can detect a 10 hour signal below -50dB in real-time provided its overall frequency change exceeds 1 Hz.

A customized approach, based on the GHT, was achieved by Tezmol *et al.* [72] for a robust segmentation technique capable of finding the location and orientation of the cervical vertebrae in x-ray images. As the GHT can give promising results regardless of noise and occlusions as well as variations in orientation and scale of the target image, is the most suitable method for this application. Chang *et al.* [73] tried to overcome the problem of tracking moving objects in a video stream by using the optical flow technique. The optical flow technique finds the velocity vectors at each pixel in the entire video. It requires complex computations and is sensitive to noise. For those reason a new method based on the HT and on voting accumulation was proposed by the authors to improve the accuracy and reducing the computation time of the technique. Experimental results shows that by using the new method the accuracy of finding the optical flow vectors has been improved as well as the computation time required for the extraction of moving objects information (from 843.66ns to 151.52ns for an 256 x 256 image).

Another method for tracking objects in a sequence of sparse range images has been introduced by Greenspan *et al.* [74]. They used a Bounded HT (BHT), which is a variation of the general HT. It exploits the coherence across image frames that result from the relationship between known bounds on the object's velocity and the sensor frame rate. The method has been implemented and tested on a variety of objects using both simulated and real data. Experiment results shows that the BHT can work with any shape of object and it functions quite well in presence of sparse data. Rosito and Schramm [75] have used a windowed HT to detect rectangular structures in images. The image is scanned and a sliding window is used to compute the HT. Peaks are extracted from the image under test and a rectangular is detected when four extracted peaks satisfy certain geometric conditions. The method can be used for both synthetic and natural images.

Rovira-Mas *et al.* [76] have used the HT in order to detect crop rows from forward-view images captured from a moving automated tractor. A five step HT approach was used for increasing the processing speed as well as the quality of the image analysis. Real-time field navigation of the agricultural machinery following crop

rows at normal operating speed could be achieved. Pre- and pro- processing of the image of the tractor's forward view was mandatory for the effectiveness of the application. Another real-time image processing algorithm based on run length encoding (RLE) for a vision-based controller of a Humanoid Robot system was introduced by Messom *et al.* [77]. The RLE algorithm can not only identify objects in an image, but it can provide information about their size and position. Using the HT, recognition of landmarks can be made which helps the robot localization. Other applications of the HT include an efficient lane–detection algorithm by Tsai *et al.* [78] as well as a technique for collision avoidance between two spherically extended polytopes (s-topes) [68] (Figure 2.7-3), which is a common object model in a robotic system.

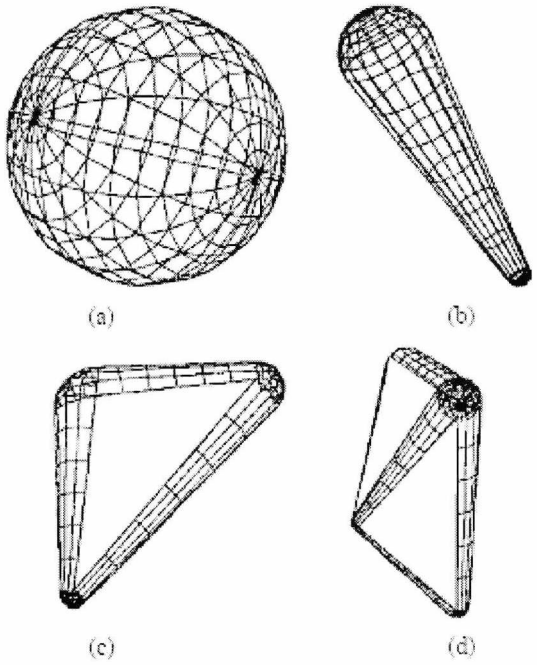


Figure 2.7-3: Simple s–topes: a) Monosphere, b) Bisphere, c) Thrisphere, d) Tetrasphere [68]

2.8 Parallel Processing Architectures of the Hough Transform

Parallel processing allows the implementation of many independent operations simultaneously. The number of operations depends on the number of processors in a given system as well as its architecture. The HT has been implemented almost in every existing parallel system and several solutions have been fabricated to improve speed and reduce the computational cost of the algorithm [171]. In this section, a

review of existing research will be presented for different implementations of the HT in real-time hardware, as well as specialized parallel architectures where the HT has been applied. As one of the main characteristics of the HT is the independent simple calculation of every feature in an image, parallelism can be successfully achieved.

There are different configuration types for parallel processing. The single instruction, multiple data (SIMD) type consists of n processing elements (PE), in which each simultaneously perform the same operation on n independent pieces of data [24]. Most often each PE has an arithmetic logic unit, several registers and a few kilobits of RAM. The SIMD block diagram is shown in Figure 2.8-1 [80].

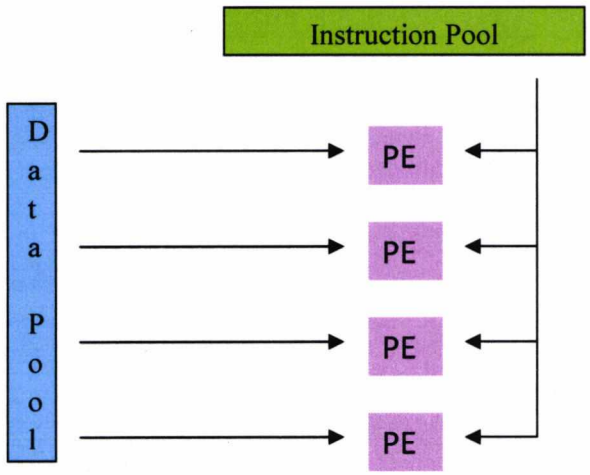


Figure 2.8-1: SIMD Architecture Block Diagram [80]

Another type of parallel processing is the multiple instruction multiple data (MIMD) in which processors simultaneously perform different sets of instructions on different data. In comparison to SIMD type, the MIMD consists of a smaller number of faster processors [24]. The MIMD block diagram is shown in Figure 2.8-2 [81].

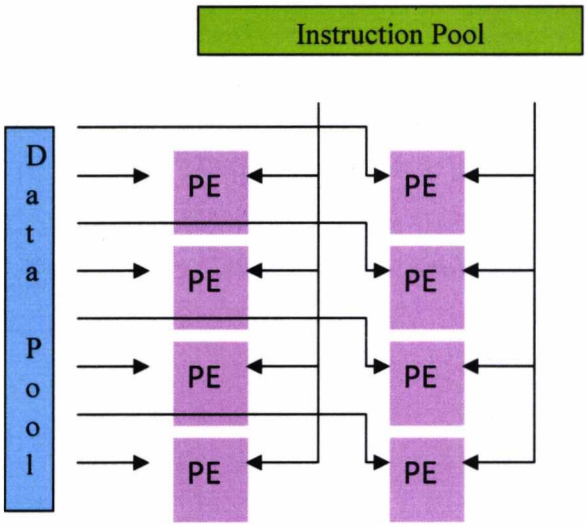


Figure 2.8-2: MIMD Architecture Block Diagram [81]

There are various ways in which the memory can be distributed. One way is for all *n* processors to share the same memory, or each processor having a unique memory address space connected to a network with interprocessor links. In the first case, the main problem is memory access contention, whereas in the second case, network congestion can be as bad as or worse than congestion in access to a shared memory [24]. There is also a combination of the previous architectures; namely the SIMD/MIMD configuration.

There are two different configuration types of HT implementation; one where the HT is mapped onto existing general purpose parallel systems, and one where the construction and design is based on a special-purposes dedicated system. The implementation on a general parallel system can be analyzed using a parallelization strategy and the embedding of the phases of the algorithm on the hardware available. The implementation of a dedicated system, on the other hand, is based on the short execution time and low cost [82]. Different SIMD and MIMD implementations which have been developed as well as some dedicated systems will be presented in the following section.

2.8.1 SIMD Implementation

As we have seen in the previous section, the HT requires a number of tasks, each consisting of a number of operations applied on a number of data. For the above reasons, there is a need for systems that can efficiently support the parallel modality proposed by the SIMD architectures. SIMD architectures can be categorized according to the way the processing elements are configured. There are several possible topologies, such as linear array, mesh (fixed topology, augmented, reconfigurable), tree, pyramid and hypercube. In general, many simple PEs can be used [82].

A linear array will have n PEs with a linear interconnecting path, in which each processor can communicate with its neighbours to the right and to the left. A controller sends instructions to each PE. Fisher and Highnam [83] have implemented the HT in a scan line array processor (SLAP), where Li *et al.* [84] developed two HT algorithms for straight line detection on SIMD architecture. In the first one, the image features are assigned to each PE, where the coordinates of the parameter cell are addressed simultaneously to every PE by a central controller. If the hypersurface generated by the image feature intercepts the cell, the PE sends a vote back to the controller. All the votes on the PEs can be summed by the central controller and stored for later analysis. In the second algorithm, each PE is assigned to a volume of the parameter space and broadcasts the image features. The method selection relies on three factors; the number of available PEs in the system, the number of image features, and the number of parameter cells. Another parallel architecture was developed by Alnuweiri and Kumar [85], which is a combination of an orthogonally accessed memory and a linear array structure [24].

Mesh-connected arrays of processors were first proposed nearly 30 years ago for parallel image processing. In a mesh-connected SIMD configuration, each processor is connected to its nearest neighbour, both up and down, as well as left and right. In such as system the processors are arranged in a square lattice, where each PE is labelled according to its position in the lattice, and can communicate with its four neighbours [24]. The regular structure and the simple interconnection topology make

the mesh well suited for VLSI implementation. However, mesh-connected arrays tend to be quite slow when transferring data operation required over long distances. As a solution to this problem, a variety of bus systems has been proposed. The static nature of these systems does not allow for modification of the processors' communications patterns while the algorithm is executed. Rosenfeld *et al.* [86] as well as Kannari and Chuang [87] have studied various mappings of the HT onto such arrays.

In order to overcome the problem caused by the static nature of the bus systems mentioned above, researchers have developed alternative bus systems, whose configuration can change under program control to suit communication needs (reconfigurable). Reconfigurable mesh parallel processing systems can be found on [88], [89], [90], [91], [92], [93], [94] and [95]. Other topologies where the HT has been implemented are the tree [96], the pyramid [97], [98], [99], [100], and the hypercube [101], [102], [103] [172] [176].

2.8.2 MIMD Implementation

Parallel implementation of the HT on MIMD systems can be categorized according to the memory organization, which means the way that the image and the transform spaces are distributed among the PEs. There are three possible configurations [104]:

a) The global image memory and distributed transform memory: all PEs have access to all feature points in the image and each PE computes the HT in different segments of the transform space.
b) The distributed image memory and global transform memory: each PE has access to a different segment of feature points in the image, and computes the HT for the entire transform space.
c) The distributed image memory and distributed transform memory: each PE has access to a different segment of feature points in the image, and computes the HT for a different segment of the transform space.

The first configuration causes a serious memory contention. This is caused during the accumulation stage of the HT, where the shared parameter space undergoes an updating process. In order to avoid that, all the PEs must have access to the same entry in image memory simultaneously. In the second configuration, memory contention is still present but allows more alternatives of access to image memory space. Finally, the third configuration involves quite a large overhead in global shifting and summation of data from all parts of the mesh, and as a result, extensive data transmission among the processors. The problem can be eliminated using a MIMD parallel processor to enable the PEs to access different image memory segments concurrently. Memory contention can still occur, though is less likely than in the other two configurations.

2.8.3 Dedicated Systems

In this section several dedicated systems which target real-time execution of the HT will be presented. The implementation for a dedicated system is based on the short execution time and low cost. Such systems can be categorized in the following way; 1) systolic structure systems, 2) pipeline systems, and 3) other systems.
HT systems with a systolic architecture usually rely on image preprocessing to obtain a stream of incoming feature points. The first systolic structures for straight line detection based on the HT were introduced by Chuang and Li [105] and Kung and Webb [106] in 1985. Later on, more systolic implementations were designed [32], [107], and [108].

Another real-time architecture where the HT has been implemented by several authors is the pipeline architecture. The implementations grouped in this family differ from the systolic ones in the set-up of the PEs. Figure 2.8-3 shows a pipelined architecture example. Such systems can be found on [109], [110], and [111]. Other pipeline systems are developed to fit in Application Specific Integrated Circuit (ASIC) [112] and FPGA implementations. FPGAs have become a competitive alternative for high performance digital signal processing applications. Using FPGAs, faster and lower cost-designs can be achieved. The HT has been implemented in FPGAs by Tagzout *et al.* [12] and Cucchiara *et al.* [13]. As the HT

implementation in this thesis is based on FPGAs, a more detailed description about FPGAs will follow in consequent chapters.

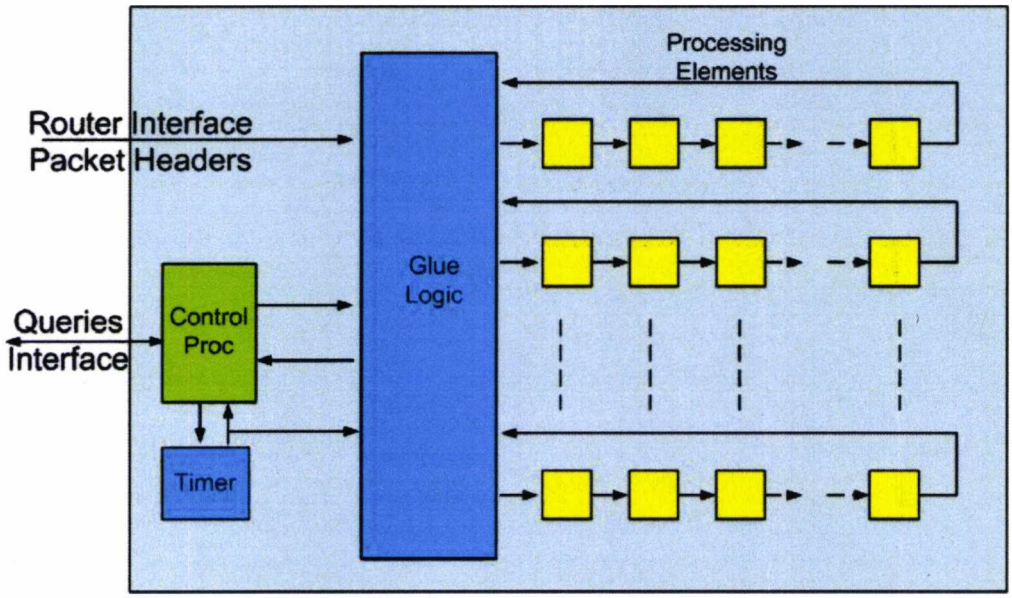


Figure 2.8-3: A Pipelined Architecture for Real Time Measurements [109]

Other specific HT implementations use approaches that can not be classified in any specific way. Such an approach is the Content Addressable Memory (CAM) – based HT for straight line detection [113]. The voting process and the peak extraction, which compose the HT, are directly executed by CAM. The CAM has sufficient parallelism for practical applications and has a double role. It acts as a SIMD type PE array that performs highly parallel processing, and also as a memory for a two-dimensional HT. Voting is executed in every scanning line, and not in every pixel, as other methods have utilized. The CAM based HT implementation can not only achieve straight line extractions, but more complex curves and their end-points as well [114], [115].

Another system for implementing the HT uses the Co-Ordinate Rotation Digital Computer (CORDIC) unit as a basic PE. The CORDIC was developed by Volder [116] to solve trigonometric problems that arise in navigation applications. It was unified by Walther [117] and it is an iterative procedure to compute magnitude and phase or the rotation of a vector in circular, linear and hyperbolic co-ordinate systems, described by the parameter *m* shown in Table 2.8-1.

Table 2.8-1: The CORDIC Arithmetic Function

	m = 1	m = 0	m = −1
Rotation $z \rightarrow 0$	$x' = x \cos z + y \sin z$ $y' = -x \sin z + y \cos z$	$x' = x$ $y' = y - zx$	$x' = x \cos hz - y \sin hz$ $y' = -x \sin hz + y \cos hz$
Vectoring $y \rightarrow 0$	$x' = \sqrt{(x^2+y^2)}$ $z' = z - \tan^{-1}(y/x)$	$x' = x$ $z' = z-(y/x)$	$X' = \sqrt{(x^2 - y^2)}$ $Z' = z - \tanh^{-1}(y/x)$

Since the CORDIC algorithms require only primitive operations, such as shifts and additions, it can be readily used to evaluate trigonometric functions. Figure 2.8-4 shows the CORDIC arithmetic unit. CORDIC is generally faster than other approaches when a hardware multiplier is unavailable, or when the number of gates required to implement the functions it supports should be minimized [175]. On the other hand, when a hardware multiplier is available, lookup-table methods are generally faster than CORDIC. Majumdar [118] and Maharatha *et al.* [117], have implemented the HT using the CORDIC unit and they concluded that it can be a good candidate for low-power, high performance real-time HT computation.

Rhodes *et al.* [119] have used the technique of Restructurable VLSI (RVLSI) to implement a HT processor. It is based on a wafer-scale integration technology containing many add-and-multiply cells. A large area of wafers is filled with standard logic cells and the wafers are tested after fabrication to determine which cells are functional. All the working cells are connected by fusing or breaking links of a metal matrix. The RVLSI approach to wafer-scale integration utilizes a laser to "restructure" the silicon circuitry after processing is complete. Using this technique manages to implement the HT at frame rates. There are several other dedicated systems where the HT is implemented and can be found on [120], [121], [122], [123], [124], and [125] [173].

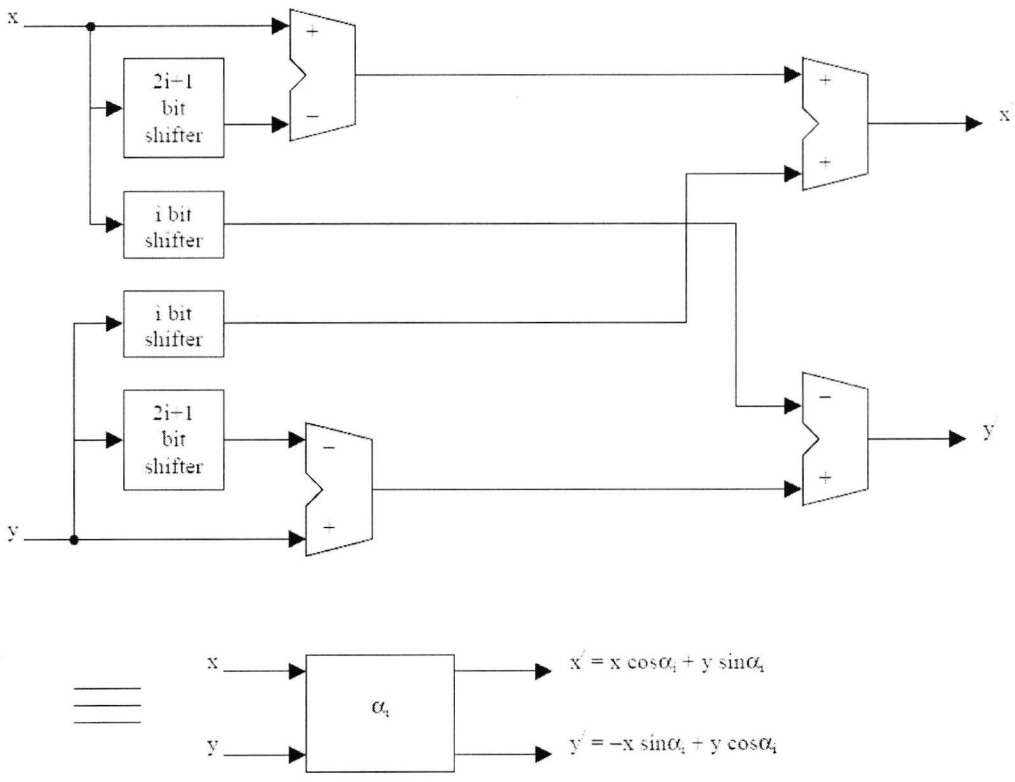


Figure 2.8-4: The Elementary CORDIC Arithmetic Unit [117]

2.9 Conclusion

A detailed presentation of the HT has been discussed in this chapter, as well as, several different methods, applications and architectures were presented. The common factor of all those methods is the calculation of the accumulator bins, which is the key factor of the HT algorithm. It is worth mentioning that the conclusions and solutions which were obtained in this research can be successfully applied to every single method and architecture presented above. Detailed application to each method will not take place in this thesis, but the basic processing step which is the calculation of the SHT, as well as, an efficient implementation of the HT on FPGA's by using logarithmic arithmetic will be presented and discussed in subsequently chapters.

CHAPTER THREE

EDGE DETECTION & DIGITAL LOGARITHMS LITERATURE REVIEW

3.1 Introduction

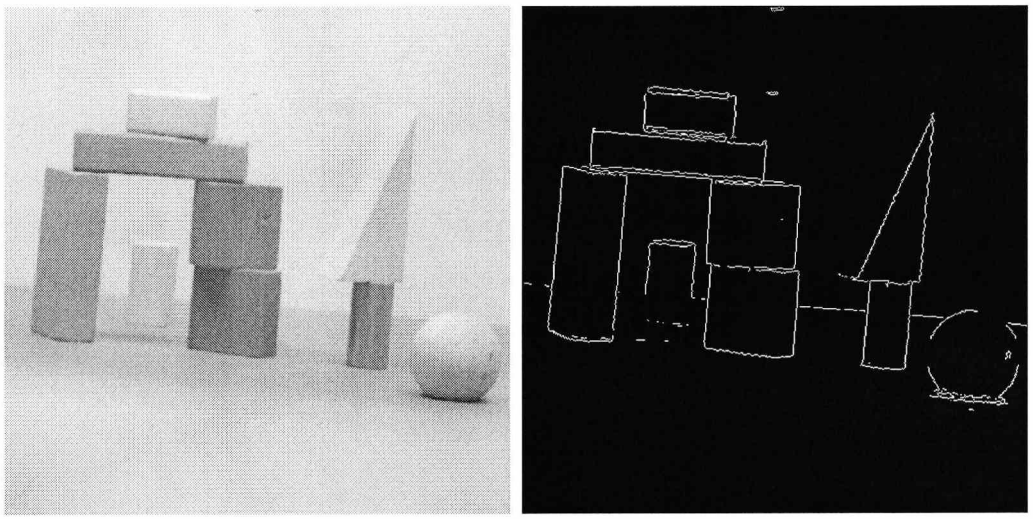
This chapter provides an overview of the edge detection, as well as the digital logarithms. In the next sections a brief description of some traditional edge detection methods will be presented and compared, where a method of special interest, the Canny method, will be extensively described and all the different stages of the Canny edge detection method will be outlined. An introduction to digital logarithms will follow. Over the past forty years, several digital logarithmic conversion methods and their applications on hardware have been examined in the literature and they will be presented in detailed in section 3.4. As it can be seen, many improvements have been attempted since the earliest digital logarithmic conversion algorithm, where some of the methods are not as applicable to hardware implementation as others.

3.2 Introduction to Edge Detection

Edge detection is one of the most commonly used operations in image analysis, and there are probably more algorithms in the literature for enhancing and detecting edges than any other single subject. The reasons for this are that edges are places in the image with strong intensity contrast, and form the outline of an object. An edge is the boundary between an object and the background, and indicates the boundary between overlapping objects. Edge detection is extremely

useful in image segmentation, when division of the image into areas corresponding to different objects is required. If the edges in an image can be identified accurately, all of the objects can be located and basic properties such as area, perimeter, and shape can be measured. Representation of an image by its edges has a further advantage that the amount of data needed to store the image is reduced significantly, while the most of the image information is retained. Since computer vision involves the identification and classification of objects in an image, edge detection is an essential tool. [16]

A straightforward example of edge detection is illustrated in Figure 3.2-1 [126]. The edge enhanced version of the same image (b) has lines outlining the objects. Note that there is no way to tell which parts of the image are background and which are object; only the boundaries between the regions are identified.



(a) (b)

Figure 3.2-1: Example of Edge Detection. (a) Image on a Grey Background. (b) Edge Enhanced Image Showing Only the Outlines of the Objects Using the Canny Method.

Edge detection is part of a process called segmentation - the identification of regions within an image. The regions that may be objects in Figure 3.2-1 have been isolated, and further processing may determine what kind of object each region represents. While in this example edge detection is merely a step in the segmentation process, it is sometimes all that is needed, especially when the objects in an image are lines.

There are a number of possible definitions of an edge, each being applicable in various specific circumstances. One of the most common and most general definitions is the ideal step edge, illustrated in Figure 3.2-2a. In this one-dimensional example, the edge is simply a change in grey level occurring at one specific location. The greater the change in level the easier is to detect the edge, but in the ideal case any level change can be seen quite easily. The first complication occurs because of digitization. It is unlikely that the image will be sampled in such a way that all of the edges happen to correspond exactly with a pixel boundary. Indeed, the change in level may extend across some number of pixels (Figure 3.2-2b-d). The actual position of the edge is considered to be the centre of the ramp connecting the low grey level to the high one.

The second complication is the ubiquitous problem of noise. Due to a great many factors such as light intensity, type of camera and lens, motion, temperature, atmospheric effects, dust, and others, it is very unlikely that two pixels that correspond to precisely the same grey level in the scene will have the same level in the image. Noise is a random effect, and it can be characterized only statistically. The result of noise on the image is to produce a random variation in level from pixel to pixel, and so the smooth lines and ramps of the ideal edges are never encountered in real images [16].

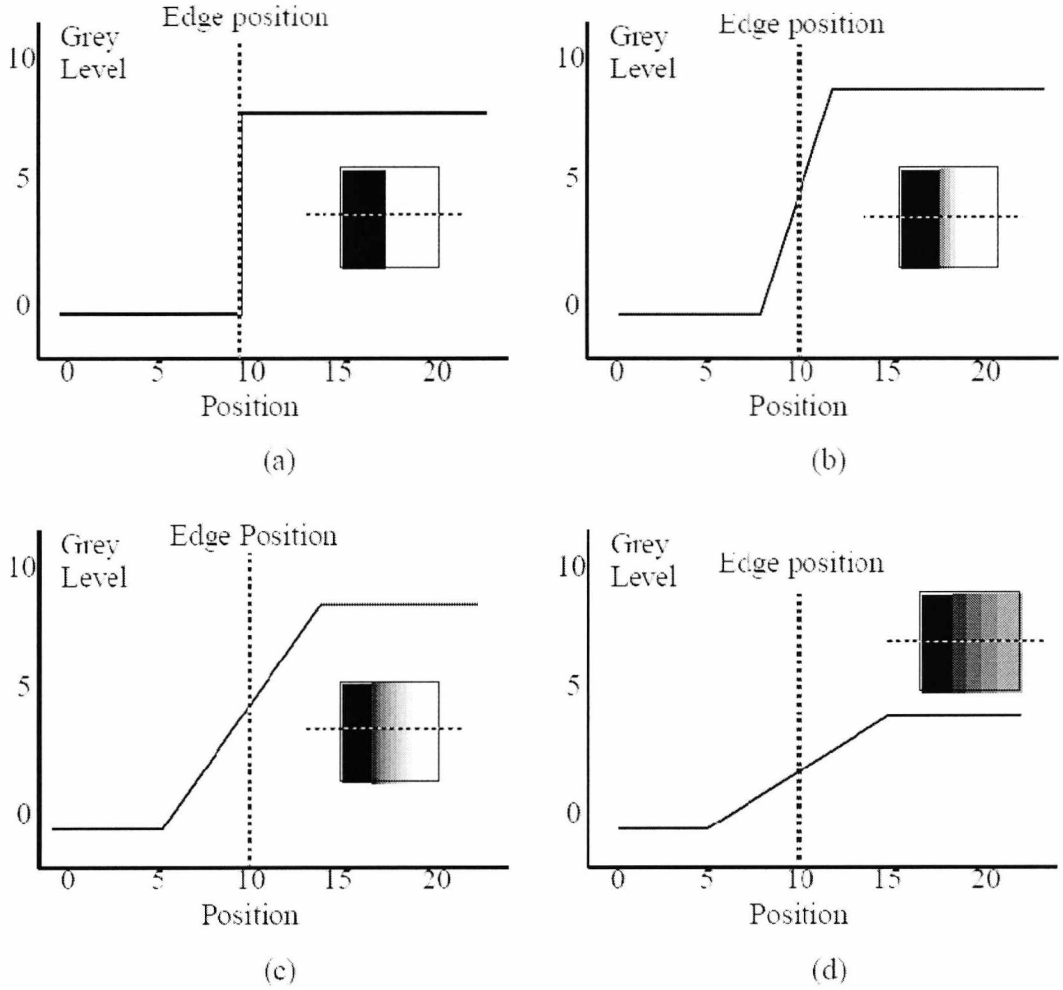


Figure 3.2-2: Step edges. (a) The Change in Level Occurs Exactly at Pixel 10. (b) The Same Level Change as Before, but Over 4 Pixels Centred at Pixel 10. This is a *Ramp* Edge. (c) Same Level Change but Over 10 Pixels, Centred at 10. (d) A Smaller Change Over 10 Pixels. The Insert Shows the Way the Image Would Appear, and the Dotted Line Shows Where the Image was Sliced to Give the Illustrated Cross-Section [130].

3.3 Edge Detection Methods

Edges can be detected by applying a high pass frequency filter in the Fourier domain, or by convolving the image with an appropriate kernel in the spatial domain. In practice, edge detection is performed in the spatial domain, because it is computationally less expensive and often yields better results. Since edges correspond to strong illumination gradients, the derivatives of the image are used for calculating the edges [130]. In the next section, the Sobel method will be outlined as performs a 2-D spatial gradient measurement on images, uses a pair of 3 x 3 convolution masks and is incredibly sensitive to noise in pictures.

3.3.1 Sobel Method

Edges are detected in a discrete two-dimensional image through first and second order derivatives, or gradient. The gradient is determined through a monadic process – each pixel (cell) within an image (vector) is examined with a local mask (kernel), that permits the application of a function with that pixel and its NxN neighbours, dependent on the mask size. Typical edge detection operations can apply two of these masks on the local 3x3 neighbourhood, which are in turn used for convolution operations. In the case of the Sobel operation [16], [127], two masks are used for each pixel considering x and y coordinates:

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \qquad \textbf{(3.3:1)}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \qquad \textbf{(3.3:2)}$$

where A is the source image. Similar techniques such as the Prewitt [128] or the Robert [128] edge detector use the same approach below but with different masks. The objective here is to use the values of G_x and G_y to produce a gradient magnitude, which can be provided by:

$$|G| = \sqrt{Gx^2 + Gy^2} \qquad \textbf{(3.3:3)}$$

Giving G as the gradient. For faster computation, the following is also acceptable:

$$|G| = |Gx| + |Gy| \qquad \textbf{(3.3:4)}$$

The angle of orientation of the edge (relative to the pixel grid) giving rise to the spatial gradient is given by:

$$\theta = \arctan(Gy / Gx) \qquad \textbf{(3.3:5)}$$

The values of Gx and Gy are derived from the two masks applied to the image, using (3.3:1) for x and (3.3:2) for y. The two components of the gradient are conveniently computed and added in a single pass over the input image using the pseudo-convolution operator shown below:

$$\begin{bmatrix} P_1 & P_2 & P_3 \\ P_4 & P_5 & P_6 \\ P_7 & P_8 & P_9 \end{bmatrix}$$

Using this kernel, the approximate magnitude is given by equation (3.2:6).

$$|G| = \sqrt{\begin{matrix} |(P_1 + 2P_2 + P_3) - (P_7 + 2P_8 + P_9)|^2 \\ + \\ |(P_3 + 2P_6 + P_9) - (P_1 + 2P_4 + P_7)|^2 \end{matrix}} \qquad \textbf{(3.3:6)}$$

This gives the gradient for horizontal and vertical texture, combined, giving the complete texture for an image. Members of G can also be vetted by a threshold in order to look for edges of specific intensity. This coupled with the 3x3 neighbourhood of the Sobel operation makes it very sensitive to single points in the image, and noise. Figure 3.3-1 shows the output of image under test, when the Sobel edge detection method applied to it.

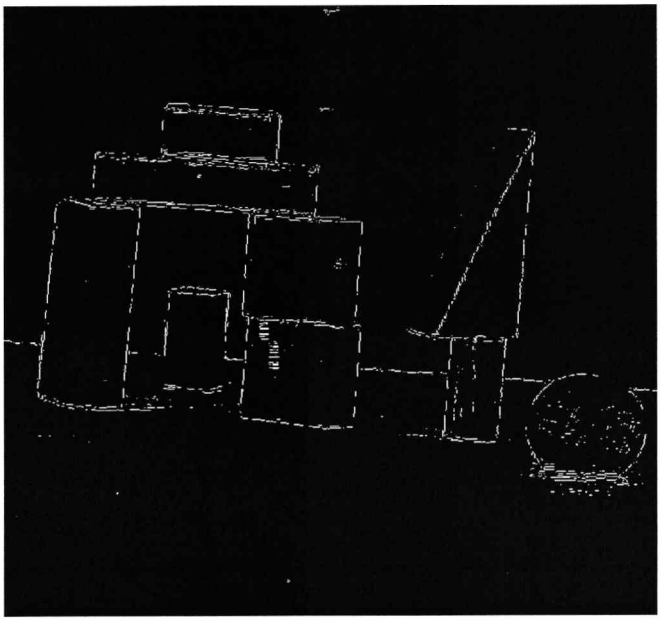


Figure 3.3-1: Sobel Output

3.3.2 Canny Method

The Canny edge detection algorithm [8] is used by many researchers, because it produces sharp and thin edges. It extends the Sobel method by incorporating additional steps before and after the derivation of gradient from the masks. It works in a multi-stage process. Canny edge detection uses linear filtering with a Gaussian kernel [129] to smooth noise and then computes the edge strength and direction for each pixel in the smoothed image. This is achieved by differentiating the image into two orthogonal directions and computing the gradient magnitude as the root sum of the squares of the derivatives. The gradient direction is computed using the arctangent of the ratio of the derivatives. Candidate edge pixels are identified as the pixels that survive a thinning process, called non-maximal suppression. In this process, the edge strength of each candidate edge pixel is set to zero if its edge strength is not larger than the edge strength of the two adjacent pixels in the gradient direction. Thresholding is then taking place on the thinned edge magnitude image using hysteresis. In hysteresis, two edge strength thresholds are used. All candidate edge pixels values below the lower threshold are labelled as non-edges and the pixels values above the high threshold are considered as definite edges. All pixels above the low threshold that can be connected to any pixel above the high threshold through a chain are labelled as edge pixels. The schematic of the Canny edge detection is shown in Figure 3.3-2 [130], [131], and [132].

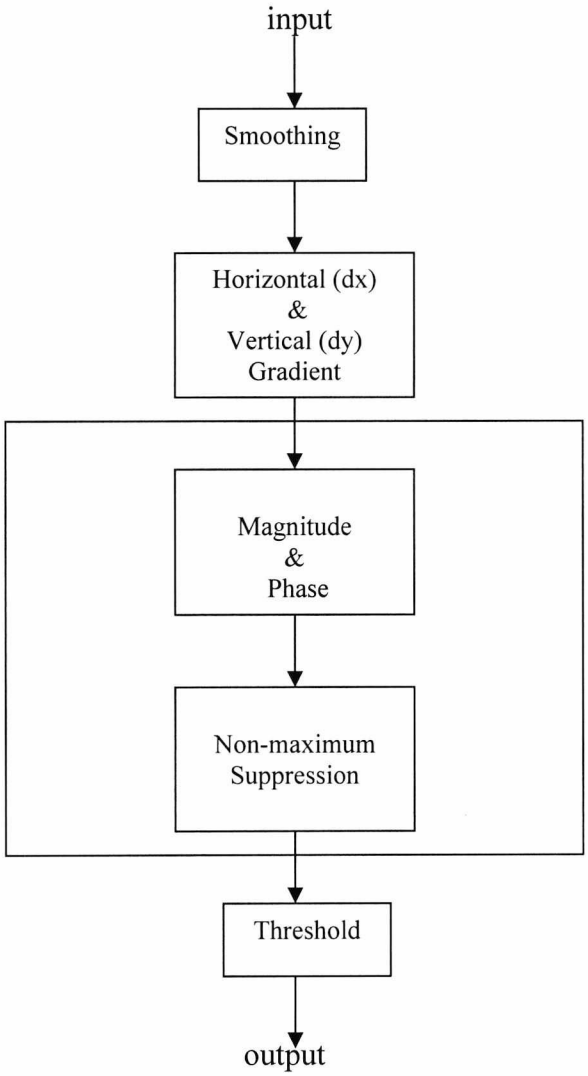


Figure 3.3-2: Schematic of Canny Edge Detection

3.3.2.1 Smoothing

The Gaussian distribution in 1-D has the form:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}}$$

where σ is the standard deviation of the distribution.

In 2-D, a circularly symmetric Gaussian has the form:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

The idea of Gaussian convolution is to use this 2-D distribution as a point spread function, and this is achieved by convolution, since the image is stored as a collection of discrete pixels. A discrete approximation to the Gaussian function is required to perform the convolution. In theory, the Gaussian distribution is non-zero everywhere, which would require an infinitely large convolution kernel, but in practice it is affectively zero more than about three standard deviations from the mean, and so the convolution kernel is truncated. The convolution kernels which have been used for smoothing in this thesis are shown in section 3.5-1 later on in this chapter. The effect of Gaussian convolution is to blur the image, where the degree of smoothing is determined by the standard deviation of the Gaussian.

3.3.2.2 Gradient Calculation

The next step after smoothing the image and eliminating the noise is to find the edge strength by taking the gradient of the image. Most of the edge detection methods assume that an edge occurs where there is discontinuity in the intensity function or a steep intensity gradient in the image as shown in Figure 3.2-2.

Most edge-detecting operators can be thought as gradient-calculators. As the gradient is a continuous-function concept and images are discrete functions, approximation of the gradient is required. Since derivatives are linear and shift-invariant, gradient calculation is most often done using convolution. Several kernels have been proposed for finding edges such as: Robert kernel, Prewitt kernel and Sobel kernel [131].

Due to the Prewitt kernels (Figure 3.3-3) usage of the central difference between rows for horizontal gradient and central difference between columns for vertical gradient, they were chosen as the appropriate kernels in this thesis for calculating the horizontal and vertical gradients.

Horizontal Convolution

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

Vertical Convolution

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

Figure 3.3-3: Prewitt Kernels [131]

3.3.2.3 Magnitude and Phase

Convolution of the image with horizontal and vertical gradients produces horizontal gradient (dx) and vertical gradient (dy) respectively. As shown previously, in equation 3.2:3, the absolute gradient magnitude |G| is calculated by the mean square root of the horizontal (dx) and vertical (dy) gradients. In order to reduce the computational cost of magnitude, it is often approximated with the absolute sum of the horizontal and vertical gradients (equation 3.2:4). The direction of the gradient is calculated by arctangent of the vertical gradient to the horizontal (equation 3.3:5).

3.3.2.4 Non-Maximum Suppression

As the magnitude and the direction have been obtained from the previous stage, a threshold operation can be applied in the gradient-based method, in a result of finding the ridges of edge pixels. Then, the edge strength of each candidate edge pixel is set to zero if its edge strength is not larger than the edge strength of the two adjacent pixels in the gradient direction. This operation is called thinning.

3.3.2.5 Threshold

The final step of the Canny process is the thresholding by hysteresis. With thresholding, a further elimination of broken edge contours, or single edge points which contribute to noise can be achieved. Such contours or pixels can be contained in the output image of the non-maximum suppression. Two thresholds

are required for hysteresis, one high threshold and one low. If the gradient of the edge pixel is above the high threshold (T_H), it is considered as an edge pixel. If the gradient of the edge pixel is below the low threshold (T_L), then it is set to zero. If the gradient is between these two, then is set to zero unless there is a path from this pixel to a pixel with a gradient above T_H. Pixels with gradients of at least T_L are a prerequisite for that path.

3.4 Introduction to Digital Logarithms

The use of logarithms for arithmetic, and their applications in hardware, has been extensively examined in the academic literature over the past forty years. The methods proposed in the early papers of this research were limited by the technology. The pioneer of the logarithmic multiplication and division use in hardware was Mitchell in 1962 [147]. Even though technology has developed drastically throughout this period, even the more recent publications described here refer back to Mitchell's [147] influential paper. As technology progresses, more advanced solutions based on the principles of Mitchell [147] and other more recent publications, have become possible. The following sections will outline, and critically evaluate key published work from the decades leading up to the present methods. The development of the techniques used for hardware implementations of logarithms throughout the decades will also be reviewed.

3.4.1 Digital Logarithms Methods

Previously to Mitchell's paper [147], on-chip memory resources restricted the implementations on hardware, and did not allow for logarithms to be stored in Look-Up-Tables (LUTs). Mitchell, effectively recognised the task of performing multiplication and division, where shift and add functions were utilized. Therefore, logarithms were utilized to simplify these functions to faster and smaller add and subtract ones, as described in the previous section. It was, nevertheless, also recognised that the calculation of logarithms may require more time and resources (to reduce errors) than multiplication and division. For this reason, his paper proposed a method of approximating logarithms, where the

advantages of its simplicity would counteract the disadvantages of the errors. In addition, a method for reducing errors caused by approximation was also proposed.

The method of approximation proposed takes n-1 as the characteristic of the logarithm where there are n bits in the integral part of the linear number X, and the mantissa is approximated to S-1, where S is obtained by shifting the binary value X so that it lies between 1 and 2. As shown Figure 3.4-1, this is a straight line approximation of the $\log_2(S)$. Therefore, due to the limited size of memory resources available, the lack of LUTs made the aforementioned method an attractive solution.

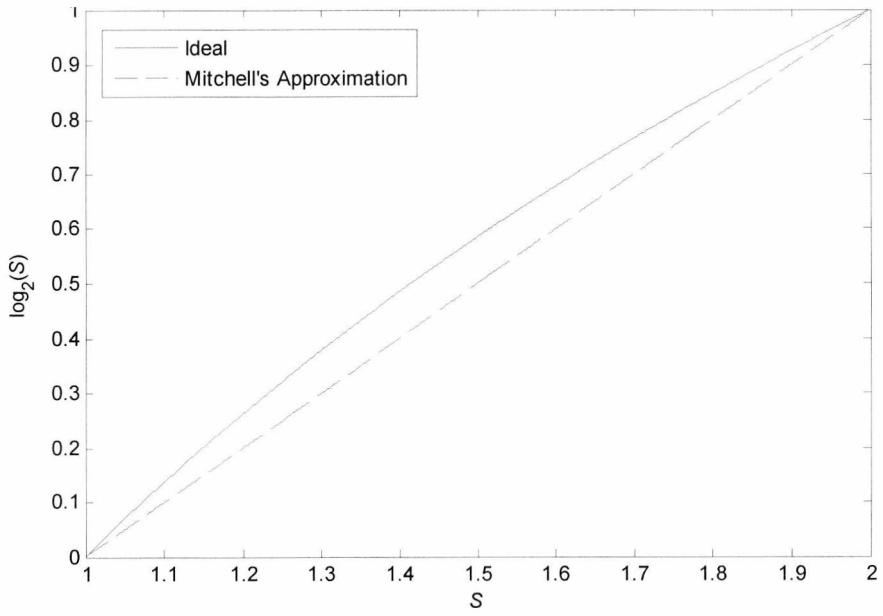


Figure 3.4-1: Mitchell's Approximation [147]

The errors were analysed for multiplication and division, as well as for the logarithms. For the error in the logarithm, the absolute maximum digital error was 0.08639, or 13.67%. The maximum possible error in multiplication was -11.1%, and the maximum possible error with division was 12.5%.

A method for reducing the multiply error was, therefore, proposed. By adding error correction operations, the multiply error can be reduced to a maximum of 2.8%. This can be further reduced by adding more operations. However, the

author notes that the additional complexity may make the logarithmic multiplication impractical compared to the shift-and-add method.

A similar solution for division, however, was not found, although one alternative possible solution was suggested; the use of an LUT to store correction factors for various intervals of S, but the comparison and memory look-up may render this solution slower than the conventional shift-and-add division.

Looking at other, more current papers, it becomes obvious that Mitchell paved a very influential way for further development of logarithm conversions and arithmetic on hardware. The solutions proposed were very practical at the time due to the high speed and low resource requirements. Nevertheless, there was still much room for improvement with regard to the accuracy and use of technology, as it became more readily available.

Combet *et al*, [148] based on Mitchell [147], proposes a method of trading an increase in hardware and decrease in speed for higher accuracy, improving on Mitchell's error by a factor of 6. Their proposed solution segments the Mitchell approximation to numerous straight lines (R`) in order to reduce the error. The segments intersect the curve at various points, interpolated with straight lines between them. The authors further suggested that, by adding more segments, the accuracy will increase, but at the cost of hardware and speed performance.
The following expressions were suggested from trial and error, as a four point approximation of the logarithmic curve. Again, the integer is found in the same manner as previously described for Mitchell's algorithm.

$$R`(1+S) = S + \frac{5}{16}S \qquad \text{For } 0 \le S < \frac{1}{4}$$

$$R`(1+S) = S + \frac{5}{64}S \qquad \text{For } \frac{1}{4} \le S < \frac{1}{2}$$

$$R`(1+S) = S + \frac{1}{8}\bar{S} + \frac{3}{128} \qquad \text{For } \frac{1}{2} \le S < \frac{3}{4}$$

$$R`(1+S) = S + \frac{1}{4}\bar{S} \qquad \text{For } \frac{3}{4} \le S < 1$$

By using rational denominators, these approximations allow for efficient hardware implementation, although at the expense of accuracy. Their system uses counting and shifting as before, but also binary decision-making to add to the complexity. The hardware implementation was also detailed in their paper. This was improved and expanded by [149], [150].

Dean, [151] proposed a solution similar to that of the aforementioned authors, the plot of which is show in Figure 3.4-2. In this case, two lines are used to approximate the logarithmic curve, and they are intersecting at S = 1.5 and y = $\log_2(S)$ = 0.625. This consequently reduces the error compared with [148] by using closer approximations. The equations for the lines are given by $y = \frac{5}{4}S - \frac{5}{4}$ and $y = \frac{3}{4}S - \frac{1}{2}$. Again these are all based on rational numbers, and can, therefore, easily be implemented on binary logic-based hardware.

To further reduce the error, a three line approximation is also proposed with intersections at S = 1.25 and S = 1.75, or S = 1.3125 and S = 0.875 to simplify the logic. This reduces the error to less than 1%.

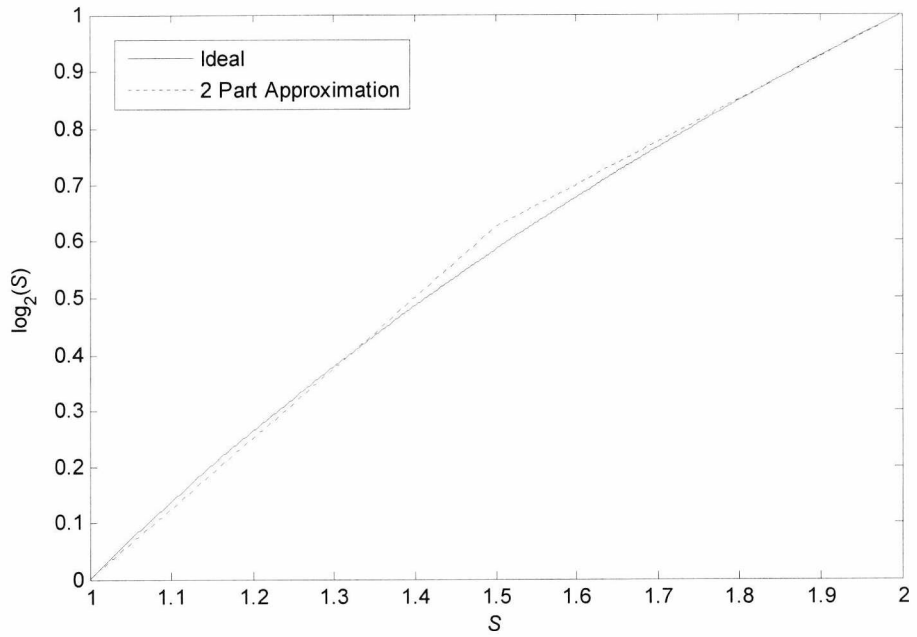


Figure 3.4-2: Two-Part Logarithm Approximation [151]

A new method, not based on Mitchell's algorithm, was proposed for evaluating the logarithm (base 2) of a binary number by Philo [152]. This method utilizes an iterative algorithm, where the number is squared and divided by two, as shown in (3.4:1).

$$S_{i+1} = \begin{cases} S_i^2 \quad if\ S_i^2 < 2 & (b_i = 0) \\ 0.5S_i^2\ if\ S_i^2 \geq 2 & (b_i = 1) \end{cases} \qquad \textbf{(3.4:1)}$$

Where $\log_2 X_1 = 0.b_1b_2b_3... = b_1 \log_2(2^{1/2}) + b_2 \log_2(2^{1/4}) + b_3 \log_2(2^{1/8}) + ...$

The number to be converted is shifted to give the integer of the logarithm. Therefore, the resulting fraction S where $1 \leq S < 2$, is converted.

This procedure generates results with about 17 bits of accuracy. Nevertheless, the successive evaluation of the square of the number negates any possible benefits of using this method for multiplication or squaring, as this is involved in the method itself. The same solution was also proposed at the same time by K. J. Dean [153], [154]. Other notable papers using logarithmic arithmetic are [155], [156].

Abed and Siferd, [157] presented two implementations for logarithm conversions. Both implementations are based on Mitchell's algorithms [147], but instead use only combinational logic and can produce the results in a single clock cycle. Although initially implemented for CMOS VLSI, the structures and Leading One Detector (LOD) are of interest to this thesis.

Previously suggested LODs [147], [148], [158]], although unable to be implemented at high speed, do provide reasonable results for word lengths above eight bits. The purpose of the LOD suggested in their paper [157] was to overcome this caveat, and be able to handle both small and large word sizes with speed and efficiency. Previous implementations used either shifting and counting, or bit-by-bit serial evaluation. Therefore, Abed and Siferd (2000) suggested a cross between previously unexplored parallel evaluation, and serial evaluation. The small LOD as shown in 3.4-3 is used as a building block for larger LODs as shown in 3.4-4.

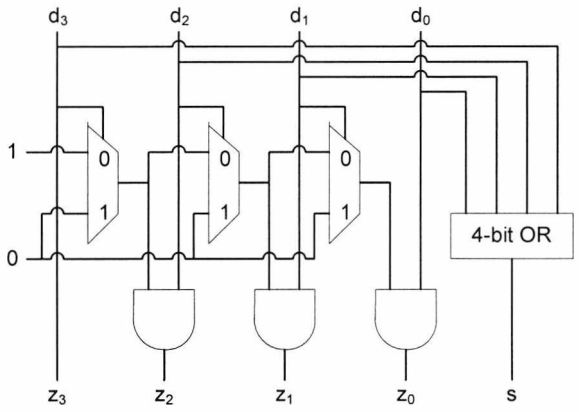


Figure 3.4-3: A 4-bit Leading One Detector [157]

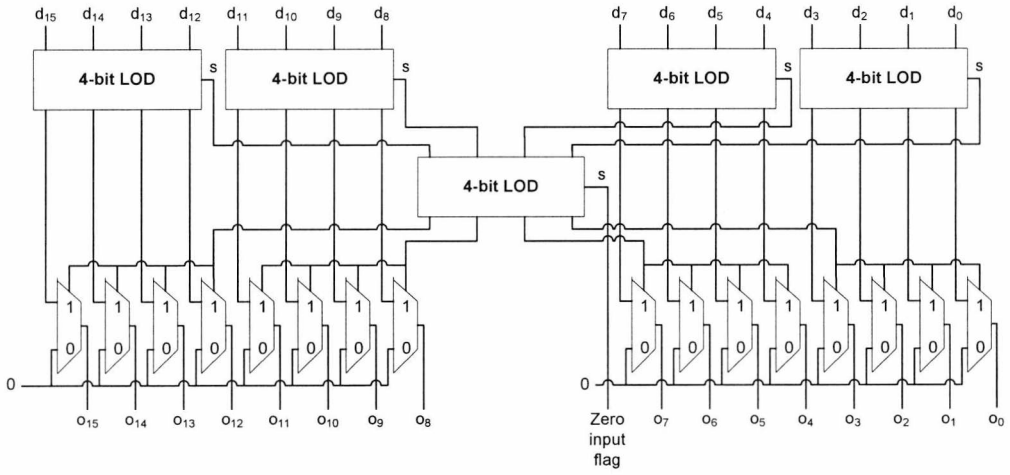


Figure 3.4-4: A 16-bit Leading One Detector [157]

The LOD uses simple logic in a repetitive and modular structure, thus making the LOD particularly suitable for VLSI implementation. The large LOD may be preceded or followed by D Flip-Flops. The output will be the same length as the original input, but only the most significant bit will be displayed on the output. An n by $\log_2(n)$ ROM look-up-table can be, then, used to convert the output to a binary number representing the position of the most significant one.

In concurrence with Mitchell's algorithm, the characteristic of the logarithm will consist of the result of the ROM, while the mantissa will consist of the shifted original binary value. Although the errors were not analysed in this case, by implementing Mitchell's algorithm, the authors suggested that this will only be

appropriate where errors are acceptable. Nevertheless, the positive aspect of the LOD is that it requires small hardware and makes the conversion fast and efficient.

As on-chip hardware resources have become larger, faster and more copious in recent FPGA devices, this paper presents a new, efficient and rational conversion method. Although Lee [3] shows an implementation of the Discrete Cosine Transform (DCT) and its inverse (IDCT), his Hybrid-Logarithm method clearly indicates how such transforms can be made with efficient hardware and conversion techniques.

In this proposed solution, an LUT is used for the conversion of the fractional parts of the fixed-point and logarithmic binary numbers for conversion. An LUT for linear to logarithm conversion, combined with barrel shifters and simple logic, can make full logarithmic conversion a possibility. Due to most images being encoded with few bits (8 or 10) the LUT is able maintain its small size. As stated in [159], the use of binary logarithm or hybrid-logarithm arithmetic at this level of precision can be an effective alternative to normal binary arithmetic, thus requiring less hardware and less power. The technique of logarithmic conversion used in Lee (2005) will be described and presented in detail in Chapter 6 of this thesis.

By applying hybrid-logarithms to the DCT and IDCT, results indicate that the reduction in resulting image quality compared to the fixed-point implementation was minimal, despite the simplified hardware. To illustrate the differences in images produced, Lee (2005) multiplied the difference in pixel values by a factor of 10 for 4-bit fractional implementation, and by 20 for the 6-bit implementation. As the author notes, there was no qualitative difference in the resultant images for 8 bits.

Lee's paper, clearly demonstrates the state-of-art implementation of logarithmic conversion and filtering for low bit-width, in image processing systems using available FPGA technology. For the 8-bit implementation, 2K bits were required for the LUT which can easily be implemented on block RAM on FPGAs, and

128 bits for the 4-bit version requiring a modicum of distributed RAM resources [10].

The above algorithms may be classified by their relative simplicity of implementation and the limited accuracy they achieve (up to 12 bits). However, the achievable accuracy is sufficient for a number of important applications, including the implementation of the HT using LSN. For that reason the examples which are presented in Chapter 6, section 6.3.2 are based in such algorithms. In addition, there are algorithms that achieve a significant improvement in accuracy (>12 bits), but require significantly more memory to store the approximation coefficients and, in most cases, also a multiplier. A method for implementing this linear interpolation is introduced by Fang-Shi *et al* [2]. There are also more accurate methods available, such as the polynomial approximations where three multipliers are required, however, these methods far exceed the needs for the implementation of the HT and will not be the focus of this thesis.

Fang-Shi *et al.* outlined the design and hardware implementation of a hybrid-logarithmic number system (LNS) arithmetic processor. This processor can perform multiplication, division, square and square roots. All input and output values, as well as addition and multiplication arithmetic are performed in the standard 32-bit IEEE floating-point number system. As demonstrated in previous sections of this chapter, [5.1] the said arithmetic operations are simple to implement with the LNS. Addition and subtraction, however, are far more complicated.

Interesting new architectures were devised and described in order to assist with a need for large word length linear and logarithmic conversions. It has been noted in the literature that converters purely based on ROM look-up-tables (LUTs) for conversion [138, 139] have limited word-length due to restricted silicon area. Furthermore, shift-and-add-based conversion algorithms [148] result in large errors. For these reasons, these algorithms are incompatible with the standard 32-bit floating-point format.

The floating point representation is $x = (-1)^s (1 + 0.M) 2^{E-B}$. Based on this equation, the logarithm of a floating point number can be represented by (3.4:2), where again M is the mantissa, E is the exponent, B is the bias (-127 for single precision). The sign bit is not taken into account, as only the magnitude is converted.

$$x` = E - B + \log_2 (1 + y) \qquad \textbf{(3.4:2)}$$

For simplicity of notation, $y = 0.M$. It can be seen from (3.4:2) that the biased exponent does not change. As $\log_2 (1+y)$ can be approximated to y, the mantissa can also does not change. This approximation will be exact only when $y = 0$ or 1; anywhere between, and there will be errors that can cause a significant impact in arithmetic operations. Hence, an error correction value (E_y) is added to y to improve the accuracy. These error correction values are stored in 2048 locations in an LUT. This LUT is addressed from the most significant 11 bits of y (y_1), and returns the correction value to be added. The correction gives an accurate approximation of $\log_2 (1+y)$ at these 2048 points, but between them, significant errors still occur.

The approximation can be further enhanced by linearly interpolating the 2048 values. The LUT also gives the correction value difference (ΔE_y), which is multiplied by the least significant 12 bits of y (y_2), to implement this interpolation. This together gives an accurate conversion according to the equation shown in (3.4:3). The architecture is shown in Figure 3.4-5(a) for the floating-point to LNS converter.

$$\log_2 (1 + y) \approx y + E_y \pm \Delta E_y \times y_2 \qquad \textbf{(3.4:3)}$$

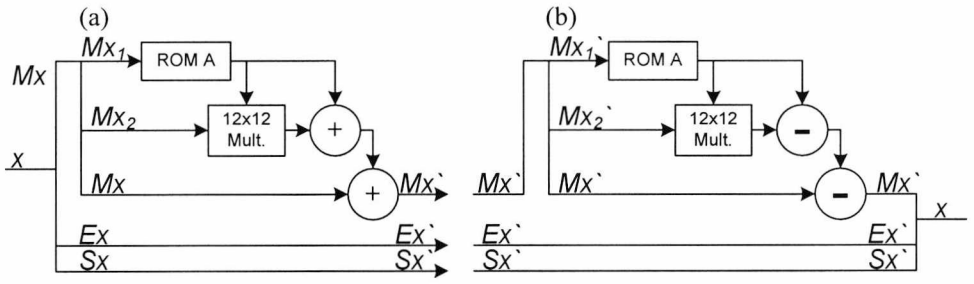


Figure 3.4-5: Floating-Point / Logarithm Converter [2]

As seen in 3.4-5(b), the LNS to floating-point converter is implemented the same way. This generates a simple and accurate conversion between floating-point and logarithms. The arithmetic processor described above, gave performance ratios of 6.4:1 for division, 8:1 for square root, and 2:1 for the FFT calculation over floating point equivalents. This is useful for arithmetic using wide bit-widths (number of bits used to represent a value), but it would be ineffective to use this for multiplication (addition in the logarithmic domain) for bit-widths of less than 12 bits, as a 12x12 bit multiplier is used in each conversion. The aforementioned shows interesting error correction and logarithmic conversion techniques, particularly useful to floating point. It may be possible for aspects of the error correction to be applied to fixed-point binary representation if need be. With fixed-point binary, a conversion of the integer to the characteristic and normalisation (*S*) will still be necessary [10].

3.5 Conclusion

In this chapter, an introduction to edge detection and a comparison between traditional edge detection methods was presented. The Canny edge detection method was extensively outlined. Canny extends the Sobel method by incorporating additional steps before and after the derivation of gradient from the masks. As it works in a multi-stage process, all the different stages which are involved for the implementation of it were presented.

.

This chapter has also outlined influential past literature on logarithmic arithmetic hardware implementation. Many improvements have been attempted since the

earliest breakthrough algorithm, as well as entirely new methods of logarithmic arithmetic and hardware implementation. Some of these methods, although interesting and informative, are not as applicable to hardware implementation as others. The Hybrid-LNS method by Lee [3] is the state-of-art implementation of logarithmic conversion, and it is the method that it will be used for implementing the HT on FPGA's. Even that Hybrid-LNS has a limited accuracy (up to 12 bits), it is sufficient enough for a number of applications, including the implementation of the HT.

CHAPTER FOUR

IMPLEMENTATION OF THE CANNY EDGE DETECTION METHOD

4.1 Introduction

This chapter provides a detailed hardware implementation of the Canny edge detection, and a novel moving window operator, which forms the basic implementation of the Canny algorithm. A comparison between software and a hardware version of the Canny method, using either floating point arithmetic or fixed point arithmetic will take place. The results will be presented in sections 4.5 and a description of the synchronization circuit in section 4.6. A summary and conclusion is presented in section 4.7.

4.2 Hardware Implementation

Each stage of the Canny algorithm will be examined separately, and the overall performance of the algorithm will be discussed. The total amount of the hardware recourses occupied will be shown, as well as, a flexible LUT based synchronising circuit for 2-D imaging filters of variable dimensions.

4.2.1 Moving Window Operator

The moving window operator usually processes one pixel of the image at a time, changing its value by some function of a local region of pixels (covered by the window). The operator moves over the image to process all the pixels in the

image. A 3x3 moving window is used for the Gaussian smoothing filter operation and an example of the window operator is shown in Figure 4.2-1 for a 5x5 image. For the pipelined implementation of image processing algorithms all the pixels in the moving window operator must be accessed at the same time for every clock. In order to access all the pixels in a moving window system, a design was devised that took advantage of certain features of FPGAs. The First In First Out (FIFO) buffers are used to create the effect of moving an entire window of pixels through the memory for every clock cycle. A FIFO consists of a block of memory and shift registers that manages the traffic of data to and from the FIFO. The data are sent through the camera straight away, where this allows a throughput of one pixel per clock cycle.

For a 3x3 moving window two FIFO buffers are used. The size of the FIFO buffer is given as W-3, where W is the width of the image. To access all the values of the window for every clock cycle the two FIFO buffers must be full. Figure 4.2-2 shows the architecture of the 3x3 moving window. For every clock cycle, a pixel is read from the camera and placed into the bottom left corner location of the window.

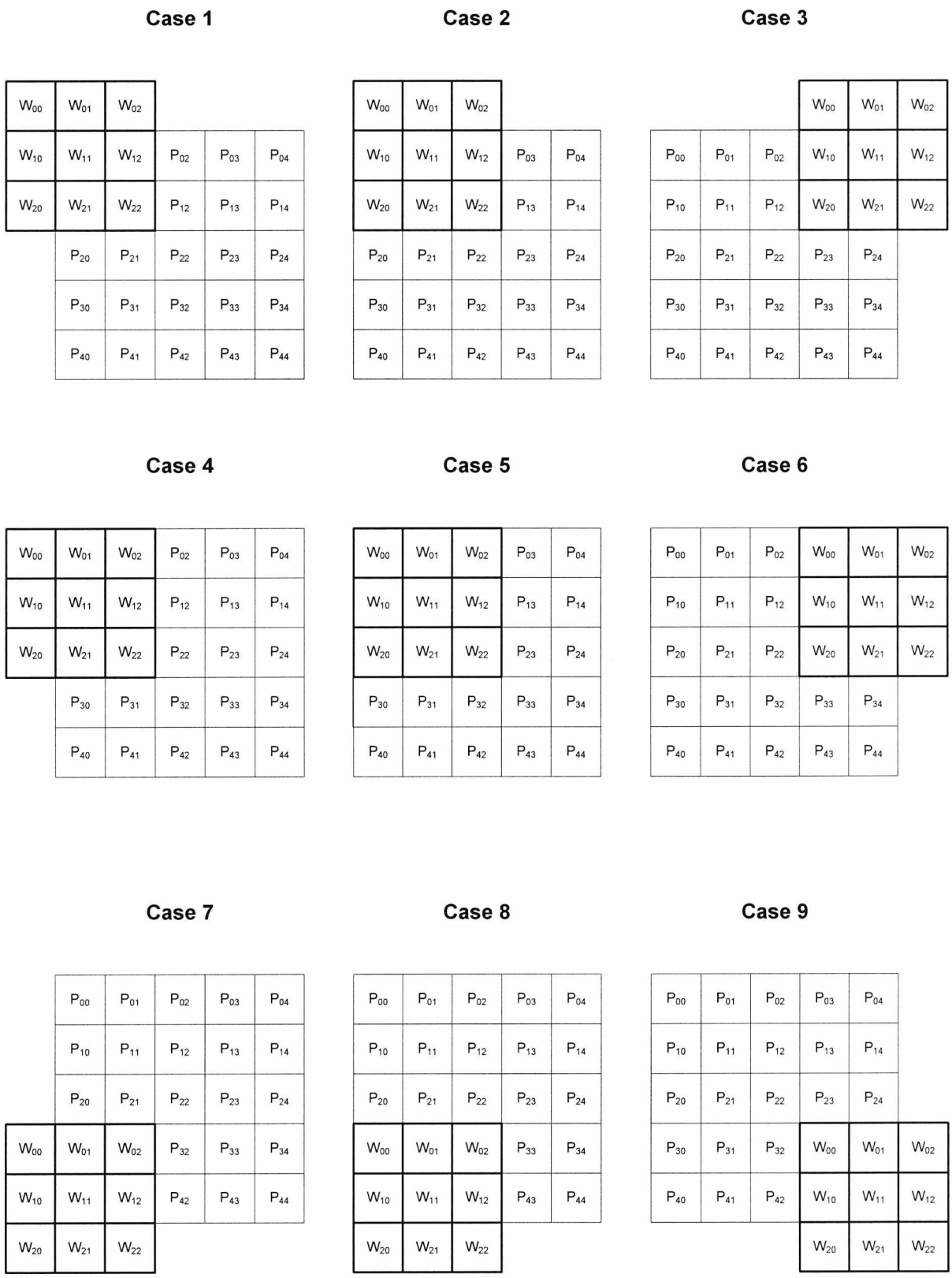


Figure 4.2-1: Example of the Window Operator in a 5x5 Image

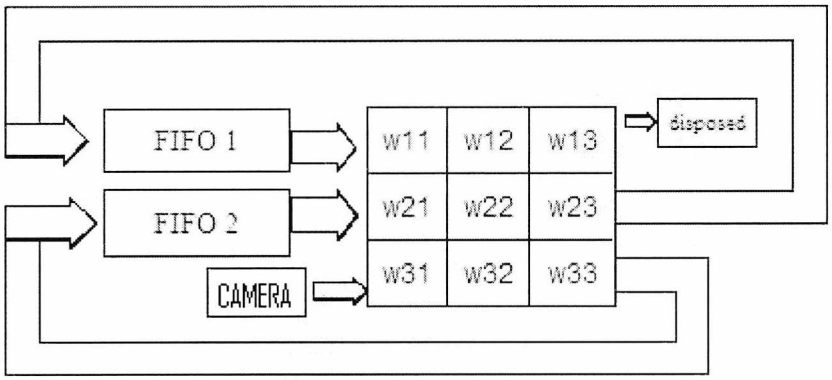


Figure 4.2-2: Architecture of a 3x3 Window [130]

The contents of the window are shifted to the right, with the rightmost member being added to the tail of the FIFO. The top right pixel is displayed after the computation on the pixels is completed, since it is not used in future computation.

4.3 Canny Hardware Implementation

The Canny edge detection operation consists of four main stages, as it can be seen in Figure 3.3-2 in Chapter 3 Section 3.3.2. First is the image smoothing, followed by the horizontal and vertical gradient calculation stage. Then, the directional non-maximum suppression stage occurs, where the threshold and thinning stage completes the operation.

4.3.1 Image Smoothing

The Canny edge detector first requires convolution with 2-D Gaussian, and then with the derivative of a Gaussian. Since Gaussian filter is separable, for smoothing, we can use two 1-D convolutions in order to achieve the effect of convolving with 2-D Gaussian. In addition, saving a number of multipliers is achieved. The convolution is operating along rows and then columns.

The one of the two 1-D filters consists of a 3x1 kernel, where the second consists of a 1x3 kernel respectively. The 2-D filter required for the derivative of the

Gaussian consists of a 3x3 kernel. The coefficient values of the filters are shown in Figure 4.3-1 and are defined using an 8-bit fixed point resolution.

22	163	22

a

22
163
22

b

6	0	-6
44	0	-44
6	0	-6

c

Figure 4.3-1: Filter Coefficients a) 1-D Across Rows b) 1-D Across Columns c) 2-D Filter

As a 3x3 moving window operator is used, two FIFO buffers are employed to access all the pixels in the 3x3 window at the same time. Since the design is pipelined, the Gaussian smoothing starts once the two FIFO's buffers are full. That is, the output is produced after a latency of twice the width of the image plus two (2*width +2) cycles. The output of this stage is given as input to the horizontal and vertical gradient calculation stage.

4.3.2 Horizontal and Vertical Gradient Calculation

This stage calculates the vertical and horizontal gradients using the 3x3 Prewitt convolution kernels shown in Figure 3.3-3 in Chapter 3 Section 3.3.2.2. An 8-bit pixel in row order of the image produced during every clock cycle in the image smoothing stage is used as the input in this stage. Since 3x3 convolution kernels are used to calculate the gradients, neighboring eight pixels are required to calculate the gradient of the center pixel and the output pixel produced in previous stage is a pixel in row order. In order to access eight neighboring pixels in a single clock cycle, two FIFO buffers are employed to store the output pixels of the previous stage. The gradient calculation introduces negative numbers. In hardware programming languages, negative numbers can be handled easily by using signed data types. Signed data means that a negative number is interpreted as the 2's complement of number. In this design, an extra bit is used for signed numbers as compared to unsigned 8 bit numbers i.e. 9 bits are used to represent a gradient output instead of 8. Two gradient values are calculated for each pixel, one for vertical and other for horizontal. The 9 bits of vertical gradient and the 9

bits of the horizontal gradient are concatenated to produce 18 bits. Since the whole design is pipelined, an 18 bit number is generated during every clock cycle, which forms the input to the next stage (see Figure 4.4-1 on page 73).

4.3.3 Directional Non-Maximum Suppression

The aim of maximum suppression is to determine the position of local maxima in the image as a first step in finding edges in the image. The local maxima is found by measuring changes in the gradient at each pixel in the image or region of interest.

The values of each component of the gradient obtained from the previous stage are used to get the magnitude and direction. The direction of the gradient is calculated mathematically as the arctangent of the vertical gradient component over the horizontal gradient component [130].

$$\text{direction} = \arctan\left(\frac{dy}{dx}\right) \qquad \textbf{(4.3:1)}$$

.

Since arctangent is a very complex function and also requires floating point numbers, it is inefficient to implement such functions on FPGA. Instead, the value and sign of the components of the gradient is analyzed to calculate the direction of the gradient. If the current pixel is $P_{x,y}$ and the values of the derivatives at that pixel are *dx* and *dy*, the direction of the gradient at *P* can be approximated to one of the sectors shown in the Figure 4.3-2.

Once the direction of the gradient is known, the values of the pixels found in the neighborhood of the pixel under analysis are interpolated. The pixel that has no local maximum gradient magnitude is eliminated. The comparison is made between the actual pixel and its neighbors, along the direction of the gradient, and for that reason a moving 3x3 window is used across the image.

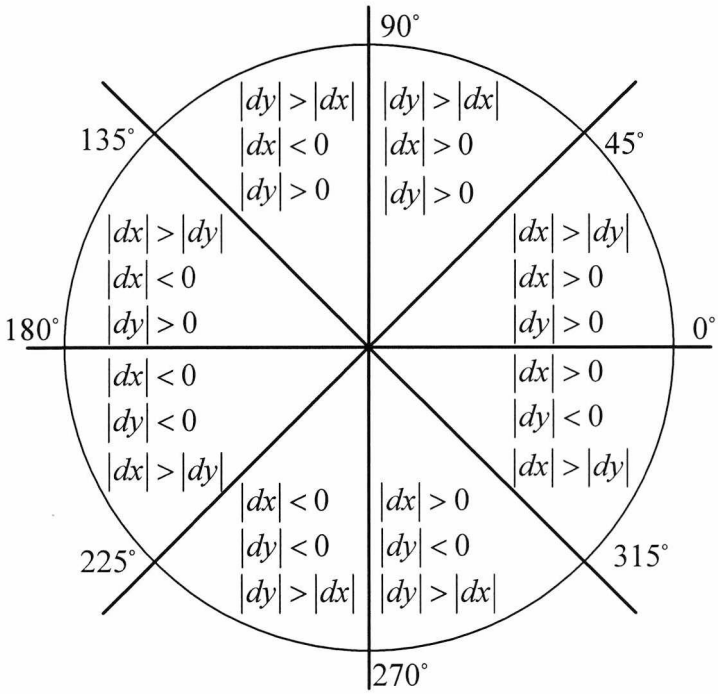


Figure 4.3-2: Gradient Orientation

To remove the possibility of spurious edges being detected the border pixels in the image are not included in the calculation. The purpose of this stage of the edge detection algorithm is to determine pixels that represent the maximum gradient in a localised part of the image. The calculation is based on the work described in [155]. This uses a method to determine the gradient in a 3x3 image window as shown in Figure 4.3-3.

The Matlab® implementation for the magnitude of each pixel in this calculation is defined by

$$\text{mag} = \sqrt{x_{i,j}^2 + y_{i,j}^2} \qquad \textbf{(4.3:2)}$$

This is a complex function to evaluate in hardware, requiring both a squaring and a square-root circuit. Instead a low complexity alternative is used.

$$\text{mag} = \frac{|x_{ij}| + |y_{ij}|}{2} \qquad \textbf{(4.3:3)}$$

So, if the gradient of the image passing through the regions P_a and P_b, the gradient at $P_{x,y}$ is compared with the magnitude of the gradient at adjacent points where $P_{x,y} = |dx_{x,y}| + |dy_{x,y}|$.

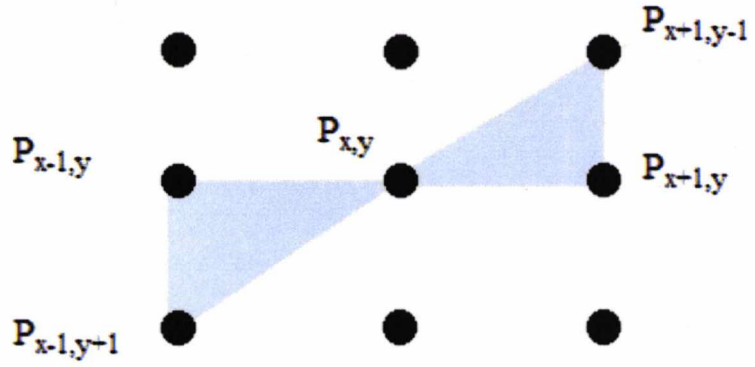


Figure 4.3-3: Pixel Interpolation

The values of the gradient at the point P_a and P_b are defined as follows:

$$P_a = \frac{P_{x+1,y-1} + P_{x+1,y}}{2} \quad \text{where}$$

$$P_{x+1,y-1} = |dx_{x+1,y-1}| + |dy_{x+1,y-1}| \text{ and } P_{x+1,y} = |dx_{x+1,y}| + |dy_{x+1,y}|$$

$$P_b = \frac{P_{x-1,y+1} + P_{x-1,y}}{2} \quad \text{where}$$

$$P_{x-1,y+1} = |dx_{x-1,y+1}| + |dy_{x-1,y+1}| \text{ and } P_{x-1,y} = |dx_{x-1,y}| + |dy_{x-1,y}|$$

The algorithm defines an edge when *both* $P_{x,y} > P_a$ and $P_{x,y} > P_b$ are satisfied. Otherwise the pixel is eliminated (set to zero) before it is passed on to the next stage of the algorithm.

The output produced in the previous stage is an 18 bit number, first nine bits are horizontal gradient and other nine bits are vertical gradient. In order to access all the pixels in the 3x3 window at the same time two eighteen bit FIFO buffers of width of the image minus three array size are employed. To calculate the phase and magnitude at every pixel the horizontal and vertical gradient values derived

from the eighteen bit number are used. The output produced in this stage is given as input to the threshold stage.

4.3.4 Threshold and Thinning

The output image of the non-maximum suppression stage may consist of broken edge contours or single edge points which contribute to noise. This can be eliminated by thresholding. In this algorithm the thresholds are calculated in the previous frame using a histogram or by an external register.To get thin edges two thresholds are used, $Threshold_H$ and $Threshold_L$.

The thresholding algorithm [130] is used to separate the pixels in the image into 3 regions before being passed to a simple thinning algorithm. The first region is the Off or zero pixels (encoded as 00) and that occur when the gradient of the pixel is less than $Threshold_L$ The second region is the On or one pixels (encoded 11) and occur when the gradient of the pixel is greater than $Threshold_H$. Finally the third region is the weak pixels (encoded 01) and it is occur when $Threshold_L < |P_{x,y}| < Threshold_H$. In this case it is set to zero unless there is a path from this pixel to a pixel with a gradient above $Threshold_H$. The path must be entirely through pixels with gradients of at least $Threshold_L$.

To get the connected path from the weak edge pixel, a 3x3 window operator is used. If the center pixel is a weak pixel and any of the neighbors is an On or one pixel, then the weak pixel is considered as On or one pixel.

After the completion of the thresholding process a thinning algorithm [130] is used, which is based on a simple morphological algorithm. It is aims at turning the 2-bit image into a simple 1-bit (binary) image which is then used as the input into the Hough transform itself.

4.4 Hardware Architecture

After the description of the individual stages of the Canny algorithm, the connection of each of the Canny elements in hardware was implemented. Figure 4.4-1 shows the pre-processing chain used for performing edge detection using the Canny algorithm. The performance of the algorithm is strongly dependent on the complexity with which these pre-processing elements are implemented and the size of the windows used.

4.5 Results Using the Canny Algorithm

Simulation of the results for the proposed implementation had been made on Matlab®. Four different implementations were designed for software and hardware routines. Floating point arithmetic, as well as fixed point arithmetic using 8-bits of precision was used and comparisons between the implementations were made. Each original image is 1024 x 1024 pixels, 8 bits per pixel and greyscale. The following section shows the results obtained, after implementing the canny algorithm in software, as well as in hardware.

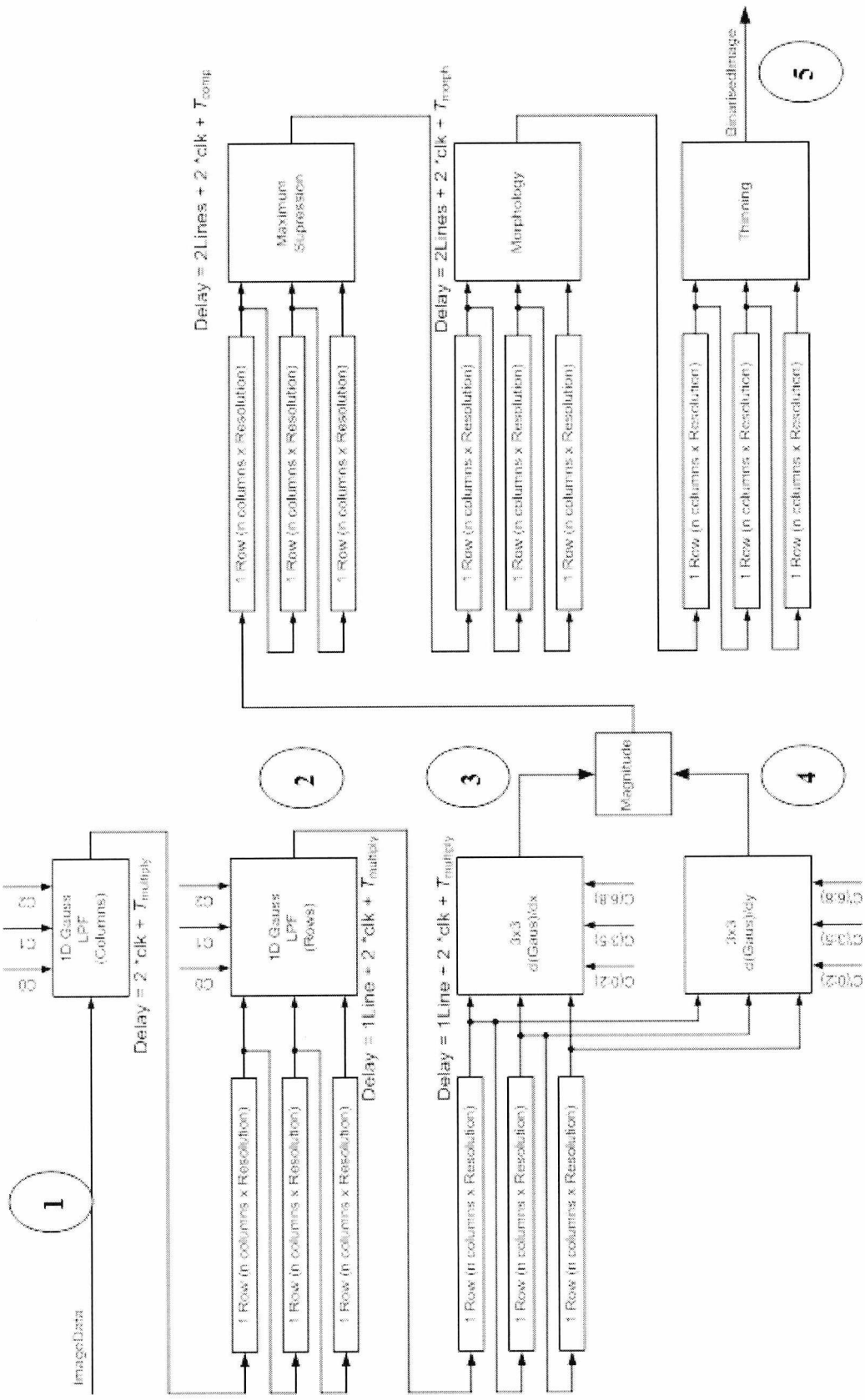


Figure 4.4-1: Canny Algorithm Block Diagram

4.5.1 Software Version of Canny Algorithm Using Floating Point Arithmetic

The first set of results can be seen in Figure 4.5-1, and were obtained using Matlab® with floating point arithmetic. Each image shows the output obtained from the different stages numbered 1 to 5 in the block diagram in Figure 4.4-1.

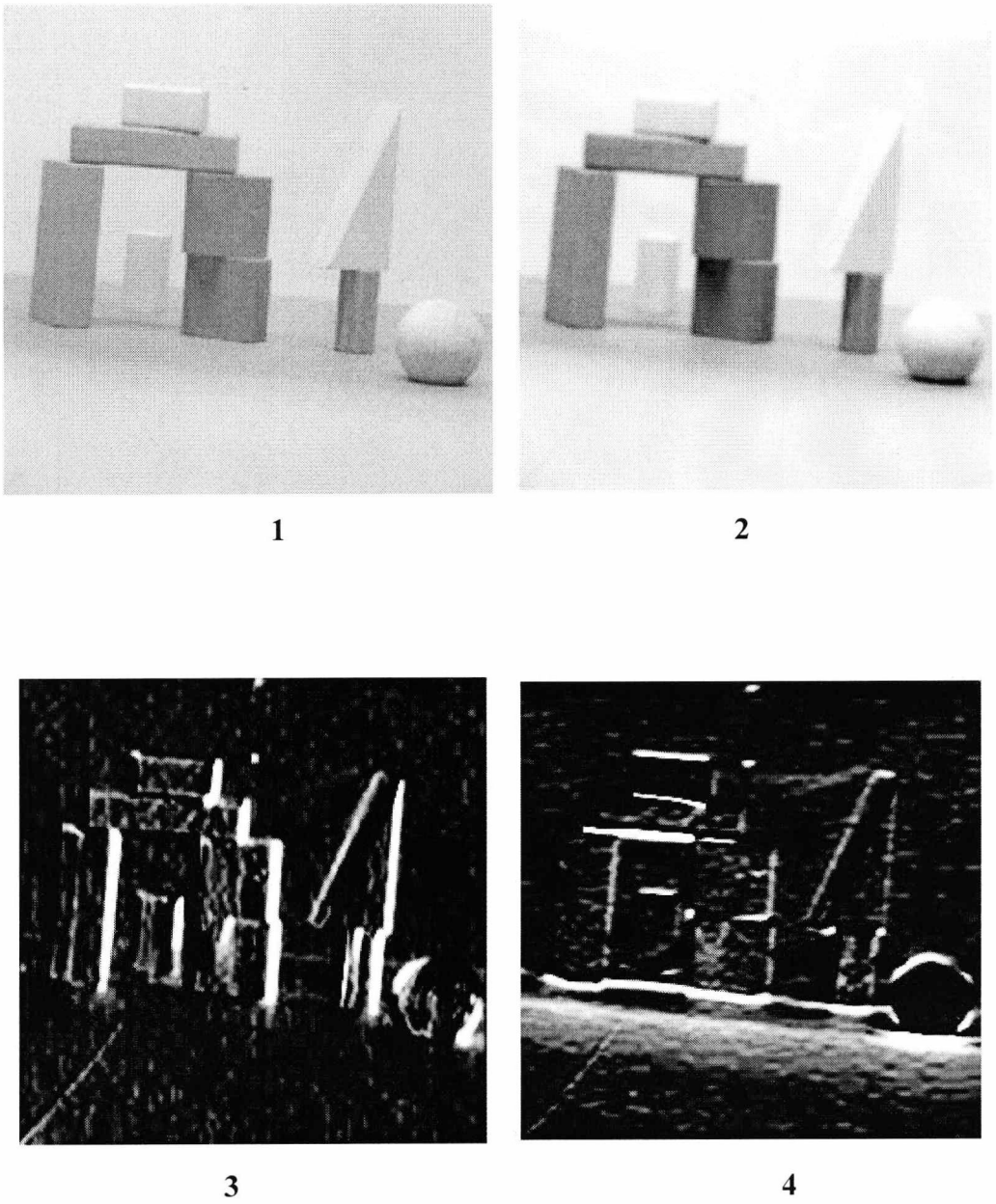


Figure 4.5-1: Software Implementation Using Floating Point Arithmetic

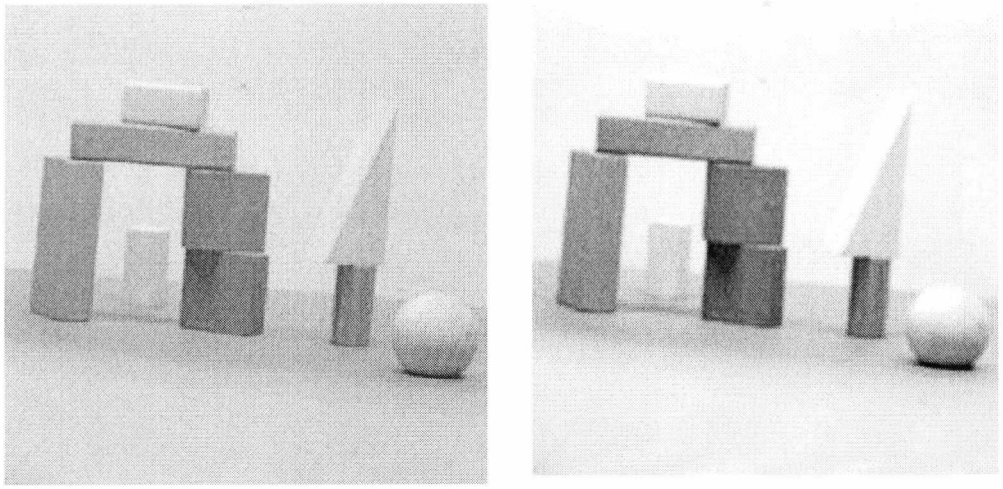
5

Figure 4.5-1: Software Implementation Using Floating Point Arithmetic

As it can be seen, the binarized output image (Figure4.4-1:5) shows the sharp and thin edges produced by using the canny edge detection algorithm, and also verify why the canny edge detection algorithm is quite successful among others.

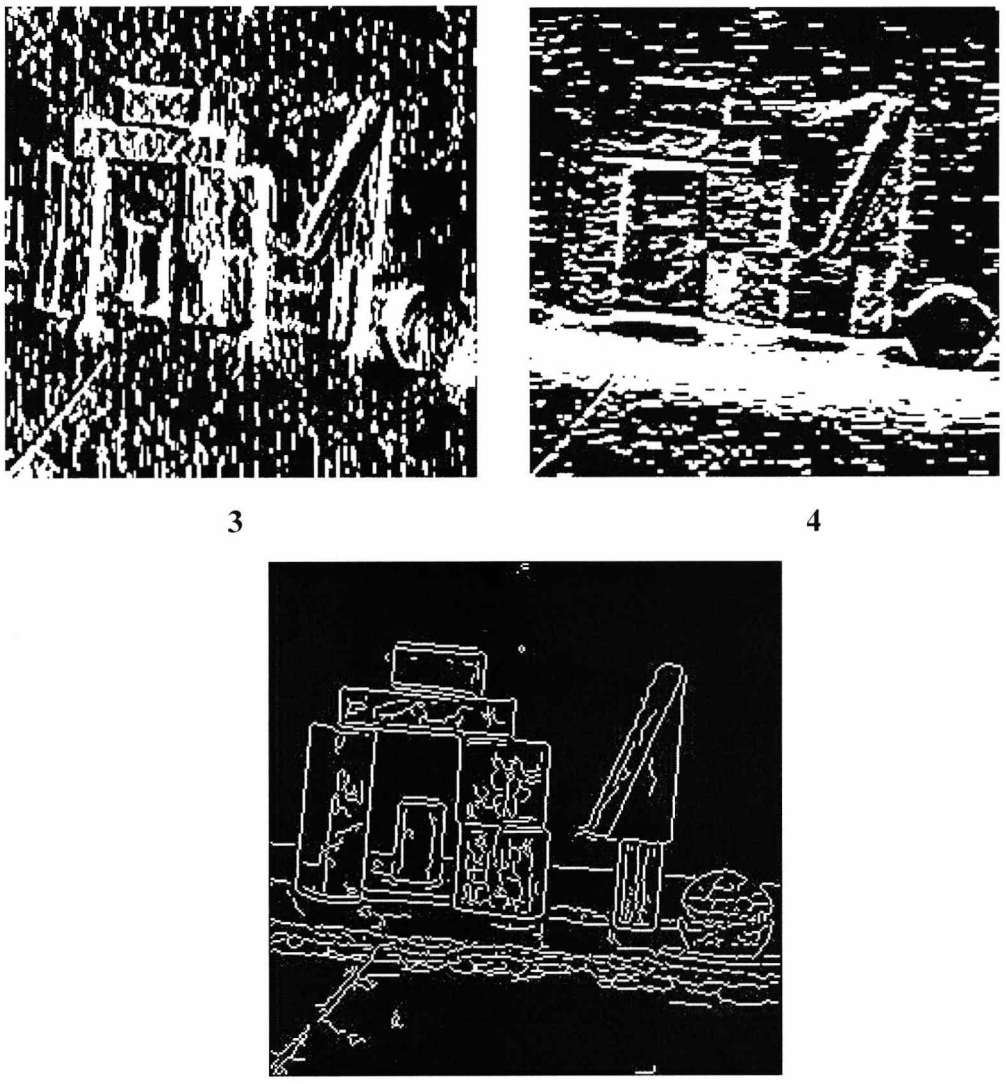
4.5.2 Software Version of Canny Algorithm Using Fixed Point Arithmetic

Figure 4.5-2 shows the results obtained using Matlab® with 8-bits fixed point arithmetic. Once again, each image shows the output obtained from the different stages numbered 1 to 5 in the block diagram in Figure 4.4-1.



1 2

Figure 4.5-2: Software Implementation Using 8-bits Fixed Point Arithmetic



3 4

5

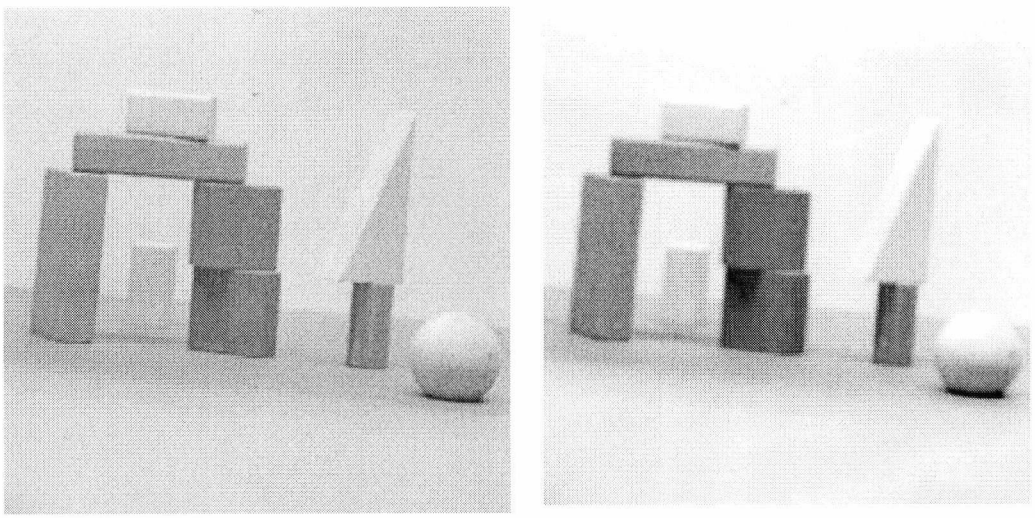
Figure 4.5-2: Software Implementation Using 8-bits Fixed Point Arithmetic

Even with fixed point arithmetic, it can be seen that the output of the image under test is very similar to the one using floating point. The difference between floating point and fixed point arithmetic was also investigated and the result is shown in Figure 4.5-3. In terms of pixel difference it was found that the two versions differ by 53152 pixels (≈5.1%), out of 1048576 total pixels (1024 x 1024).

Figure 4.5-3: Difference Between Floating Point and Fixed Point Arithmetic

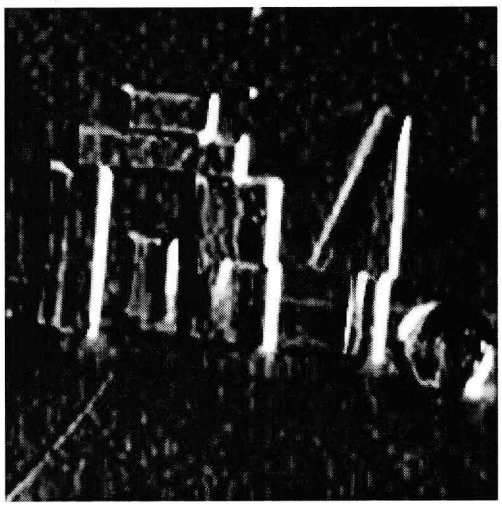
4.5.3 Hardware Version of Canny Algorithm Using Floating Point Arithmetic

A hardware version of the Canny edge detection was designed and tested, using floating point arithmetic and the results are shown in Figure 4.5-4. The magnitude calculation, as well as, the implementation of the directional non-maximum suppression is the main differences between the software versions. Each image shows the output obtained from the different stages numbered 1 to 5 in the block diagram in Figure 4.4-1.



1 **2**

Figure 4.5-4: Hardware Implementation Using Floating Point Arithmetic



3

4

5

Figure 4.5-4: Hardware Implementation Using Floating Point Arithmetic

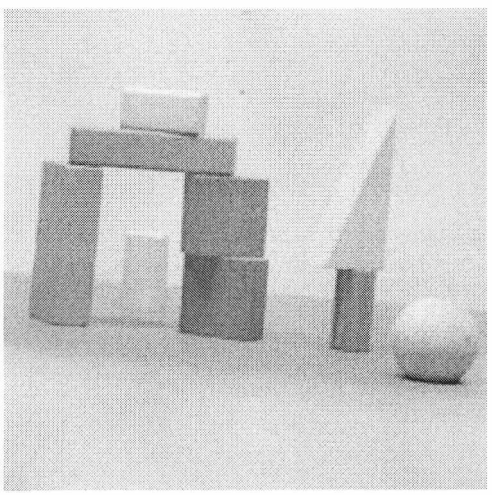
A sharp and thin edge output was produced (Figure 4.5-4:5) using the hardware version proving the success of the edge detection design. An investigation was made between the software version using floating point and the hardware, and the results are shown in Figure 4.5-5.

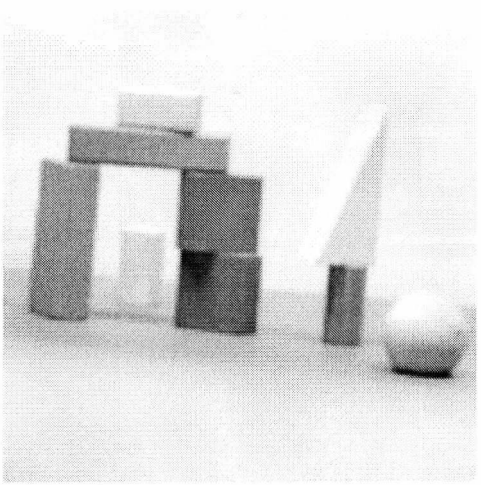
Figure 4.5-5: Difference Between Software Version Using Floating Point and Hardware.

In terms of pixel difference it was found that the two versions differ by 23856 pixels (≈2.28%), out of 1048576 total pixels (1024 x 1024).

4.5.4 Hardware Version of Canny Algorithm Using Fixed Point Arithmetic

The final set of results was obtained using the hardware version of the canny algorithm using 8-bits fixed point arithmetic and can be seen in Figure 4.5-6. Each image shows the output obtained from the different stages numbered 1 to 5 in the block diagram in Figure 4.5-1.





1 **2**

Figure 4.5-6: Hardware Implementation Using 8-bit Fixed Point Arithmetic

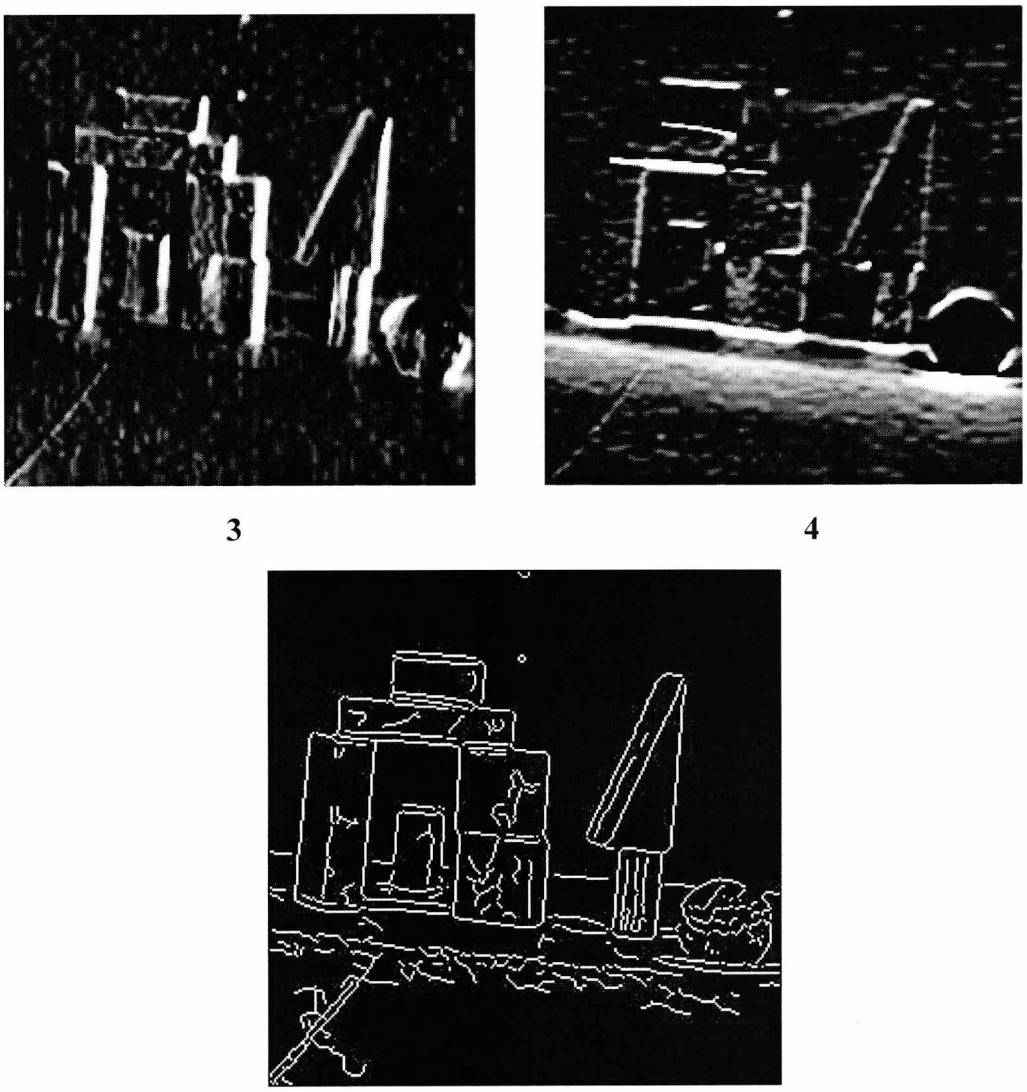


Figure 4.5-6: Hardware Implementation Using 8-bit Fixed Point Arithmetic

Even with fixed point arithmetic, it can be seen that a satisfactory output was obtained. The difference between the hardware fixed point and the software floating point arithmetic was also investigated and the result is shown in Figure 4.5-7. In terms of pixel difference it was found that the two versions differ by 24742 pixels (≈2.36%), out of 1048576 total pixels (1024 x 1024).

Figure 4.5-7: Difference Between Hardware Version Using Fixed Point and Software Version Using Floating Point Arithmetic.

After the simulation of the hardware version using Modelsim, where 8-bit fixed point arithmetic was used, for the Canny edge detection algorithm, the number of logic requirements was calculated and the results are shown in Table 4.5-1.

As the table shows, for a 1024 x 1024 image, 1204 CLB's and 30 18x18 multipliers required. Assuming that the implementation is on a Virtex-4 series FPGA, where the maximum number of slices is 89088 (Chapter 5), the implementation of the canny edge detection occupies only $\approx 5.41\%$.

Table 4.5-1: Logic Calculations for the Canny Edge Detector

	Gauss X	Gauss Y	DGauss X	DGaussY	Max Suppression	Threshold	Thin	Total
Row Memory	0	1024 x 2 x 8	1024 x 2 x 8	N/A	1024 x 3 x 8	N/A	1024 x 3 x 2	1024 x 62/16 = 3968 Slices
Multipliers	3	9	9	9	N/A	N/A	N/A	30 18 x 18 Mults
Adders	2 X 17	4 x 17 + 2 x 18 + 1 x 19	4 x 17 + 2 x 18 + 1 x 19	4 x 17 + 2 x 18 + 2 x 18 + 1 x 19	8	N/A	N/A	447 = 447 Slices
Comparators	0	0	0	0	8 x 8	2 x 8	1 x 8	88 = 88 Slices
Logic slices	10	15	12	12	N/A	4	2	55 Slices
Sync Circuits	0	1	1	N/A	1	0	1	4 x 12 = 48 Slices
FF's	2 x 8 2 x 1	2 x 18 2 x 1 2 x 1 8 x 2	9 x 8 2 x 8 4 x 1	9 x 8 2 x 8 4 x 1	9 x 8 8 x 8 8	2 x 2	9 x 2	424 = 212 Slices
Total Logic								4818 Slices/ 1204 CLB's

4.6 A Synchronizing Circuit [133]

Extraction of edges in an image using the canny algorithm prior to the HT require a number of pre-processing steps to remove noise from the captured image or to accentuate features in the image prior to the image extraction, measurement or object recognition algorithms.

For many real-time applications this pre-processing is performed by convolving the image with simple 2D windowing functions having dimensions of 3x3, 5x5, 7x7 or even 9x9 pixels. This type of processing requires row memories to store the data prior to processing it with the windowing function. As a consequence there is a latency (usually of a number of rows) before the output of the windowing function is ready to be used as the input to the next windowing function which, in turn, has its own latency. The increasing processing power and memory available on modern FPGAs is now making it possible to use larger windows and this leads to an increase in the latency between each stage of the pre-processing cascade. The filtering operations being performed by the windowing function itself also incur an additional latency. This is strongly dependent on the filter architecture chosen and the implementation technology used. This combined latency of the windowing function and the signal processing time itself (also called the pipeline) will vary depending on the requirements of the application and the length of the cascade being implemented.

Although it is a simple problem to calculate the delay through each part of the cascade it can be time consuming at the design stage to test and verify that the correct delays have been added to the logic controlling the timing of each element in the cascade. One solution is to use a global timing controller with decoders used for each stage of the processing chain. Although simple in principle, this approach becomes increasingly complex with the length and number of pre-processing elements. It also results in a significant routing overhead as the number of control signals required increase. An alternative, proposed here, is to build a matching Shift Register (SR) for each control signal

which compensates for the delay in each processing element. The use of simple and efficient SR elements implemented using the RAM based LUTs in FPGA fabrics to build these delays, were designed for the canny edge detection.

The circuit in Figure 4.6-1 shows the basic architecture of the synchronising element used for each circuit in the cascade shown in Figure 3.6-1. A separate variable length SR is used to delay the controlling signal by the required number of lines and pixels dependent upon the size of the window and the number of clocks required to perform the filtering operation on one pixel element. The SRs are built using a simple SRL16 structure [134] which configures the Look-up Table (LUT) elements available in a logic slice to be configured as a variable length shift register. For Xilinx devices this usually incurs a minimum delay of 2 clock cycles. (Smaller delays will require a simple FF to replace LUT based SR). Once the delay through each processing element has been determined the address bits for the pixel delays (AP) and for the line delays (AL) can be hardwired to constant logical values.

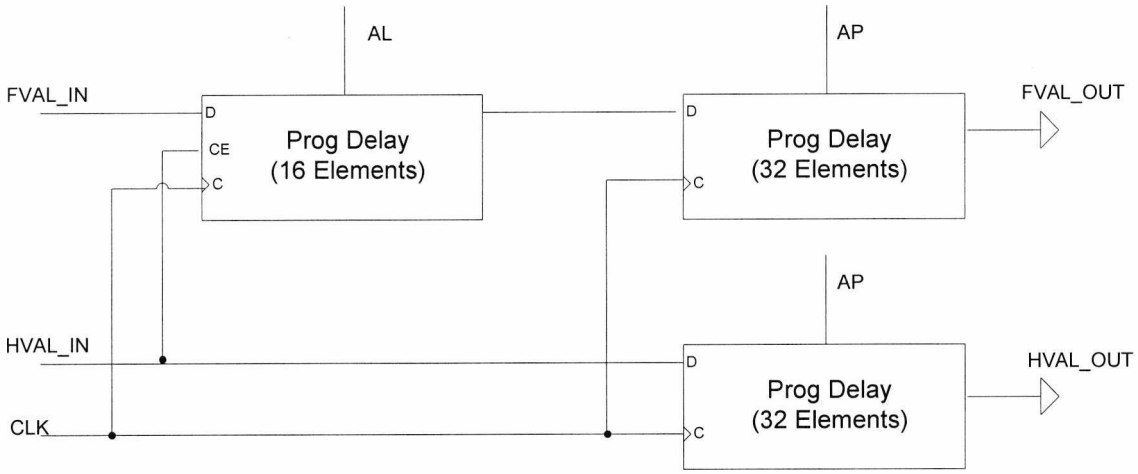


Figure 4.6-1: The Synchronization Circuit Block Diagram

The synchronisation cell described was modelled using Modelsim PE 6.4 implemented on a Virtex-4 architecture using Xilinx ISE 9.2. Synthesis results show that each synchronising element requires just 3 CLBs. This represents a small fraction of the array logic available on most FPGA fabrics. For the Canny Edge Detection algorithm the synchronising circuit used just 12 CLBs out of a total of 22272 available.

4.7 Conclusion

In this chapter, a hardware implementation of the Canny edge detection method was successfully presented, using the latest FPGA technology. A synchronization circuit for windowing operations was implemented, as a number of pre-processing steps are required prior to the image extraction. With the use of the synchronization circuit the Canny edge detection method was implemented in software and hardware using both 8-bit fixed point arithmetic and floating point arithmetic.

For the software version, the difference between floating point and fixed point arithmetic was also investigated, and in terms of pixel difference it was found that they differ by 53152 pixels (≈5.1%), out of 1048576 total pixels (1024 x 1024).

For the hardware version, the difference between floating point and fixed point arithmetic was investigated as well, and in terms f pixel difference is was found that they differ by 24742 pixels (≈2.36%), out of 1048576 total pixels (1024 x 1024).

Simulations results show that only a small fraction (≈ 5.23 %.) of the total available hardware resources in a Virtex-4 device required for the Canny edge detection implementation.

CHAPTER FIVE

OVERVIEW OF FPGA TECHNOLOGY

5.1 Introduction

The aim of this chapter is to provide background information on digital implementation technologies, in order to aid understanding of the implementations proposed in later chapters of this thesis. There are many types of technology available today to enable the implementation of logic-based systems. The following section presents basic background information on some of the most common devices such as the digital signal processors (DSP) and ASICs for comparison with the FPGAs, for understanding the reasons using the FPGA in subsequent chapters. In section 3, the alternatives in modern FPGAs will be presented. Section 4 describes common value representation formats implemented on FPGA technology, and section 5 provides a description of the dedicated arithmetic function blocks available in the modern Xilinx FPGAs. Finally, section 6 provides a summary of the current chapter.

5.1.1 Digital Signal Processors

Digital Signal Processor (DSP) ICs combine high speed and computationally intensive applications on a single device. They are a class of hardware devices that fall somewhere between an ASIC (discussed later in this chapter) and a PC in terms of their performance and the design complexity. They are not used for control logic as much as they are used for digital signal processing. However, they are an important device for comparison with the FPGA for the purposes of the research presented in this thesis.

A DSP device is capable of operating at high frequencies (1.2G Hz) and performing a vast number of instructions (e.g. 9600 per second). However, such high-performance devices can be costly. There are, nevertheless, more modest devices that can operate at lower frequencies and cost much less. C-programming language as well as Matlab, has become the main design tools for DSPs, where the main manufactures are Texas Instruments and Analog Devices. The architecture of an ADSP-21xxx core can be seen in Figure 5.1-1, where it contains three independent computational units: the ALU, the multiplier/accumulator (MAC), and the shifter.

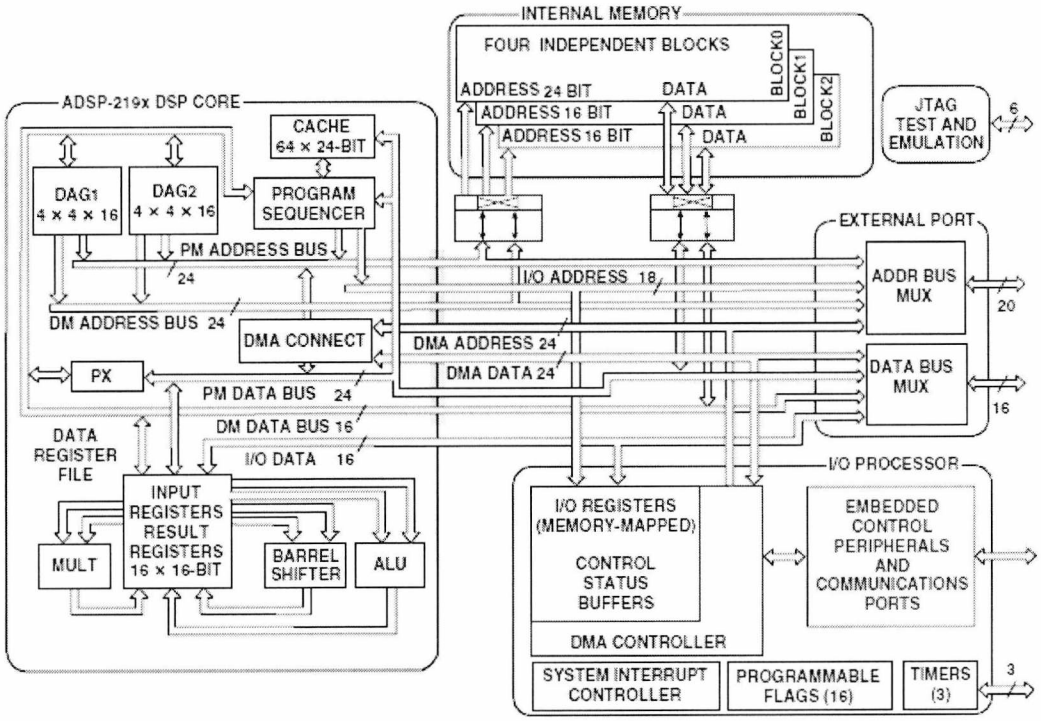


Figure 5.1-1: Block Diagram of an ADSP-21xxx Core.

DSP's are available as both floating point and fixed point architectures. However, there is a significant speed, as well as, price difference between them and for that reason fixed-point representation mostly used. These sequentially executed instructions are compiled from a high level language, and are stored in an internal program memory. This allows for in-circuit reprogramming. General input / output pins are often available.

5.1.2 ASICs

Application Specific Integrated Circuits (ASICs) are custom-built chips designed to perform a specific task. An important feature of these chips is that once the application has been designed, and the IC built, the functionality cannot be changed without redesigning and remanufacturing the chip. The ASIC is not general purpose, and the logic cells are not configurable. Low level logic is used for the design of the device, which will therefore be directly translated on to the device substrate. Embedded processor cores are used for the design of some ASICs.

The custom-built design of the ASICs enables its optimisation for high speed and low power, far more than could be achieved with general purpose programmable logic devices like DSPs, microcontrollers and FPGA's. The design of the ASICs dictates the exact speed and power required [10].

Nevertheless, ASICs have very expensive fabrication process, and, as a result, FPGAs are often used for prototyping before a final chip is designed and manufactured. This allows for removing of errors and mistakes, without re-fabrication costs. Like FPGAs, the hardware can be configured allowing for parallel concurrent logic operations, rather than instructions sequentially executed. Therefore, the devices can operate in the same way, using compatible designs. FPGAs are generally more appropriate than ASICs for implementing a system where reconfiguration is necessary or proprietary systems where low volumes are manufactured.

5.2 Introduction to FPGA's

The FPGA, since its invention in 1984 by Xilinx's Ross Freeman, has become the optimal hardware for the implementation of large and complicated logic systems. An FPGA, as name suggests, is a programmable device in which the final logic structure can be directly configured by the end user for a variety of applications. Figure 5.2-1 shows the architecture of a conceptual FPGA.

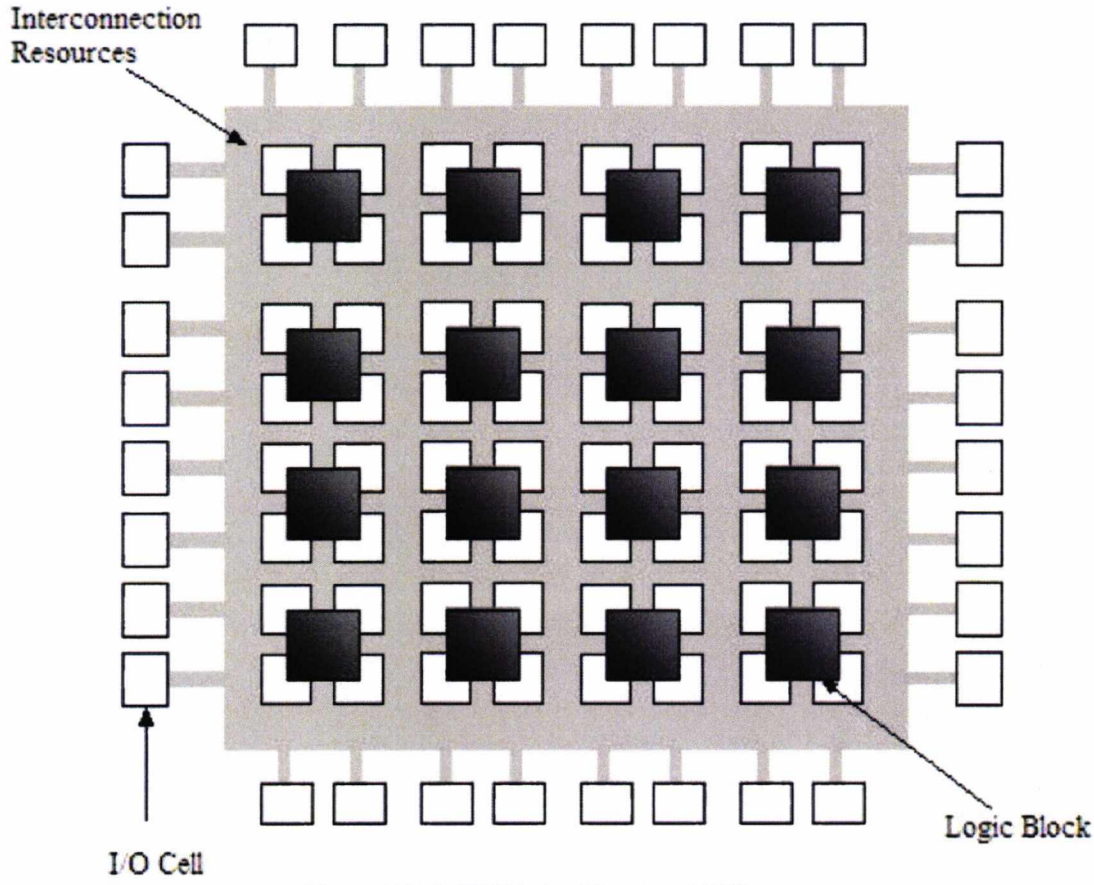


Figure 5.2-1: FPGA Architecture [130]

The most important components in an FPGA are the configurable logic blocks (CLB's), input-output blocks and programmable switches. The architecture is built from a 2-D array of CLB's that are connected by general interconnection resources. These CLB's can be as simple as 2-input NAND gates or it can have a complex structure such as multiplexers or look-up tables (LUT's). Most logic blocks also contain some type of flip-flop, to enable the implementation of sequential circuits.

Figure 5.2-2 demonstrates a simplified representation of a typical FPGA logic cell. Previous configurable technology such as PLAs or PLDs, utilised sum-of-product based logic implementation; whereas FPGA implements a RAM-based LUT, configured during power-up to give a logic output based on the four inputs which make up the LUT address.

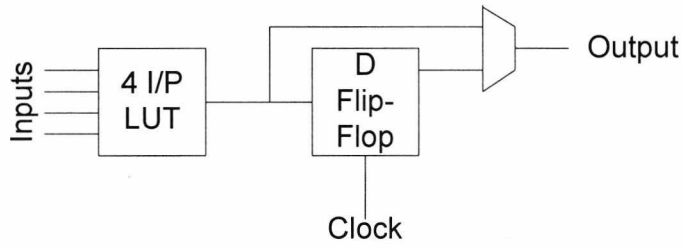


Figure 5.2-2: FPGA Programmable Logic Cell

The FPGA's principle relies on programmable interconnections between the logic cells. The FPGA achieves enormous flexibility in its design due to the programmable routing channels between the logic cells. The use of separate logic cells enables the hardware to perform different tasks at the same time, allowing logic decisions to be made simultaneously, resulting in fast implementations with predictable timing.

Commercially FPGA's have been classified into four-major categories based on their interconnection. The interconnection can be symmetrical array, row-based or hierarchical. Table 5.2-1 shows the commercially available FPGA's.

Table 5.2-1: Summary of Four Commercial FPGA.

Company	Architecture	Logic Block Type	Programming Technology
Actel	Row-Based	Multiplexer-Based	Anti-fuse
Altera	Hierarchical-PLD	PLD Block	EPROM
QuickLogic	Symmetrical Array	Mutltiplexer-Based	Anti-fuse
Xilinx	Symmetrical Array	Look-up Table	Static RAM

Modern FPGA's can contain millions of logic gates, as well as, hardware blocks dedicated to specific high-speed functions. These include high performance input and output technology, system clock managers, dedicated blocks of memory, Ethernet MAC hardware and special arithmetic units (which will be discussed in more detail later in this chapter), and even microprocessors [9]. Each of the aforementioned functions can be configured to fit particular requirements. For instance, the blocks of RAM can be configured as FIFOs,

CAMs and ROMs with single or true dual port connectivity, with a number of possible output registers, aspect ratios and other settings to choose from.

Certain functions are readily available on dedicated hardware, but there are also functions that have been developed by second or third parties that can be bought separately. These include high speed LAN (Local Area Network), Camera Link interfaces, and many signal processing functions. These Intellectual Properties (IPs), once bought, can be implemented as sub-modules within the VHDL design.

This range of functions and versatility makes FPGA, not only useful for complex sub-systems, but also applicable for the implementation of complete systems. The technology on FPGAs has expanded so rapidly, that , now, whole computer control systems can be implemented on a single FPGA, such as the Commodore Amiga 500 in a project called MiniMig [161] (now commercially available), or a fully functional PC running Linux [135], [136]. It must be mentioned, however, that some additional devices will still be required for power, analogue sub-systems, configuration devices etc. Nevertheless, with the several hundred pins available on-chip, all logic, networking and digital input / output functions can be implemented on the single FPGA.

5.3 Summary of Modern FPGAs

Hardware implementation of complex digital systems finds a flexible medium in the form of FPGAs. The two main market-leading manufactures of FPGAs are Xilinx and Altera, providing many families available on the market, offering different levels of complexity, both in size and on-chip resources.

Table 5.3-1 lists the two main manufacturers' latest FPGAs key features [141], [168], and [169]. These devices represent the high-end (and high-cost) FPGA families with many additional high-performance dedicated hardware built in. It must be noted that both manufacturers offer less-costly FPGAs, without the additional features, such as high-speed dedicated resources (Spartan from Xilinx

and Cyclone from Altera). During the period of writing up this thesis, Xilinx have introduced the Virtex-6 FPGA, and Altera has announced the Stratix III FPGA family.

Table 5.3-1 - Key Features for the Latest FPGAs

Manufacturer	Series	Logic Gates	RAM Blocks	DSP Blocks
	Virtex-6	11,640 - 118,560	9504Kb - 38,304Kb	288 - 2,016
Xilinx	Virtex-5	4,800 – 51840	936Kb - 18,576Kb	24 - 1,056
	Virtex-4	5,472 - 89,088	648Kb - 9,936Kb	32 - 512
Altera	Stratix II	15,600 - 179,400	419Kb - 9,383Kb	12 - 96
	Stratix I	10,570 - 41,250	920Kb - 3,424Kb	6 – 14

It should be noted here, that Altera's logic is implemented in logic elements (LEs) (Figure 5.3-1), whereas Xilinx's logic is implemented in slices (Figure 5.3-2). Although no direct comparison is possible, it can be said, as a rough estimation, that there are two Altera LEs to one Xilinx slice [10].

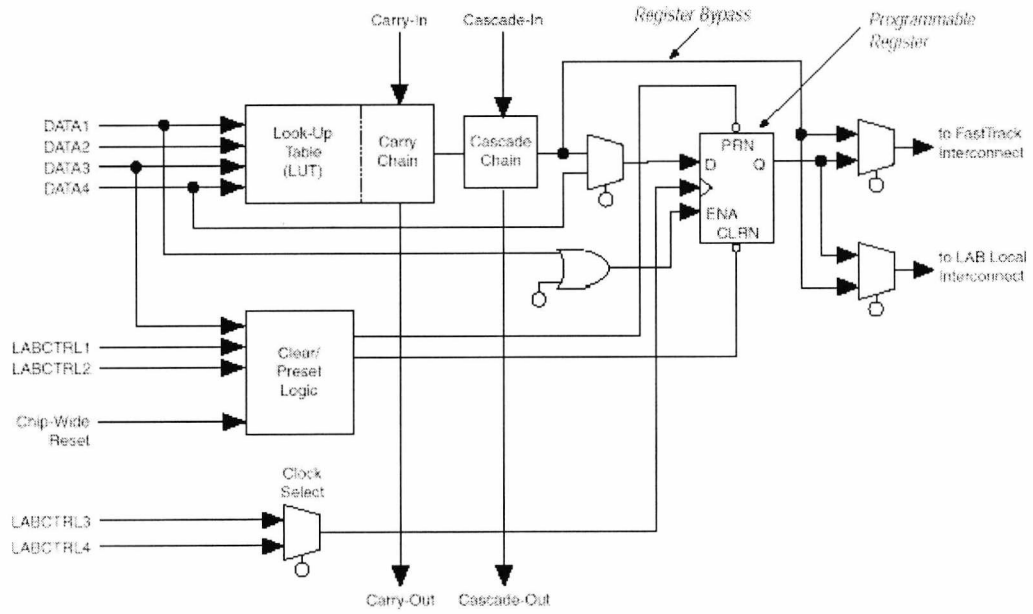


Figure 5.3-1: Altera Logic Element (LE) [10]

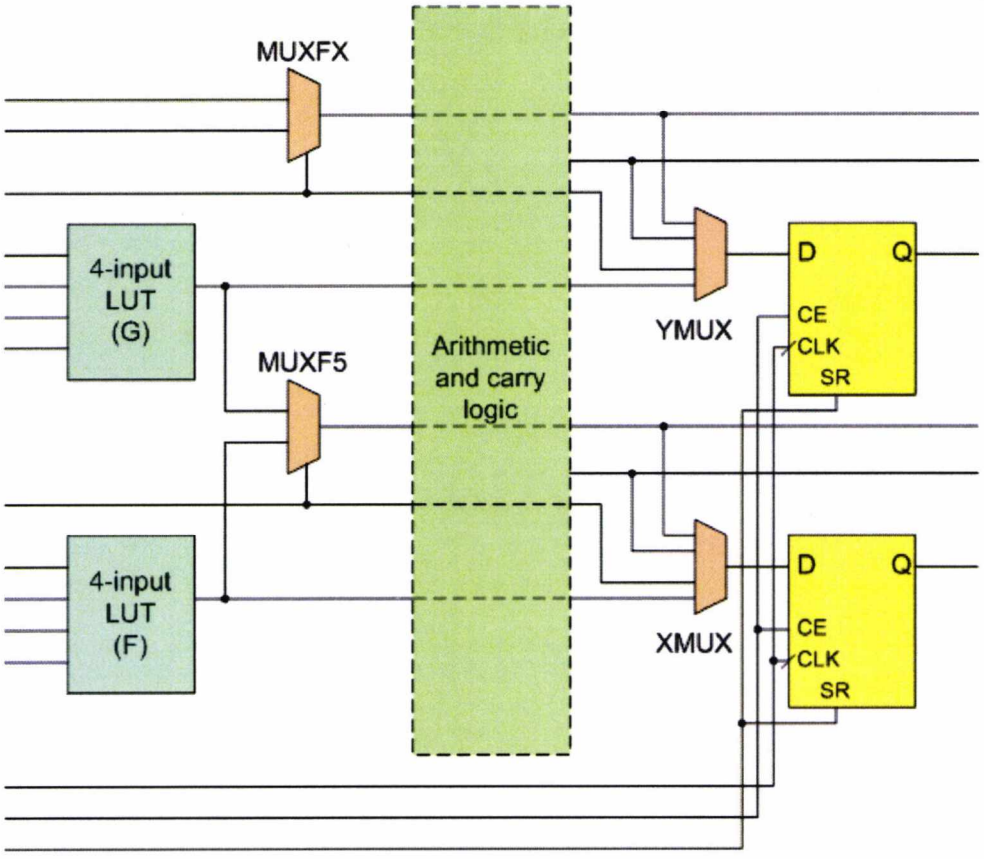


Figure 5.3-2: Xilinx Virtex Slice. [10]

The DSP-dedicated hardware implemented in the Virtex 4 devices (as described in section 5.5) contains an 18 x 18-bit multiplier and a 48-bit adder, whereas the Virtex 5 devices [142], contains a 25 x 18-bit multiplier and a 48-bit adder. The operating mode can be set dynamically, with no configuration necessary and set during design. The DSP hardware on Altera's Stratix devices [137] contains four 18-bit multipliers, followed by three adders / accumulators (Figure 5.3-3), although for the Stratix I FPGAs, various parts of the DSP hardware cannot be individually or dynamically configured.

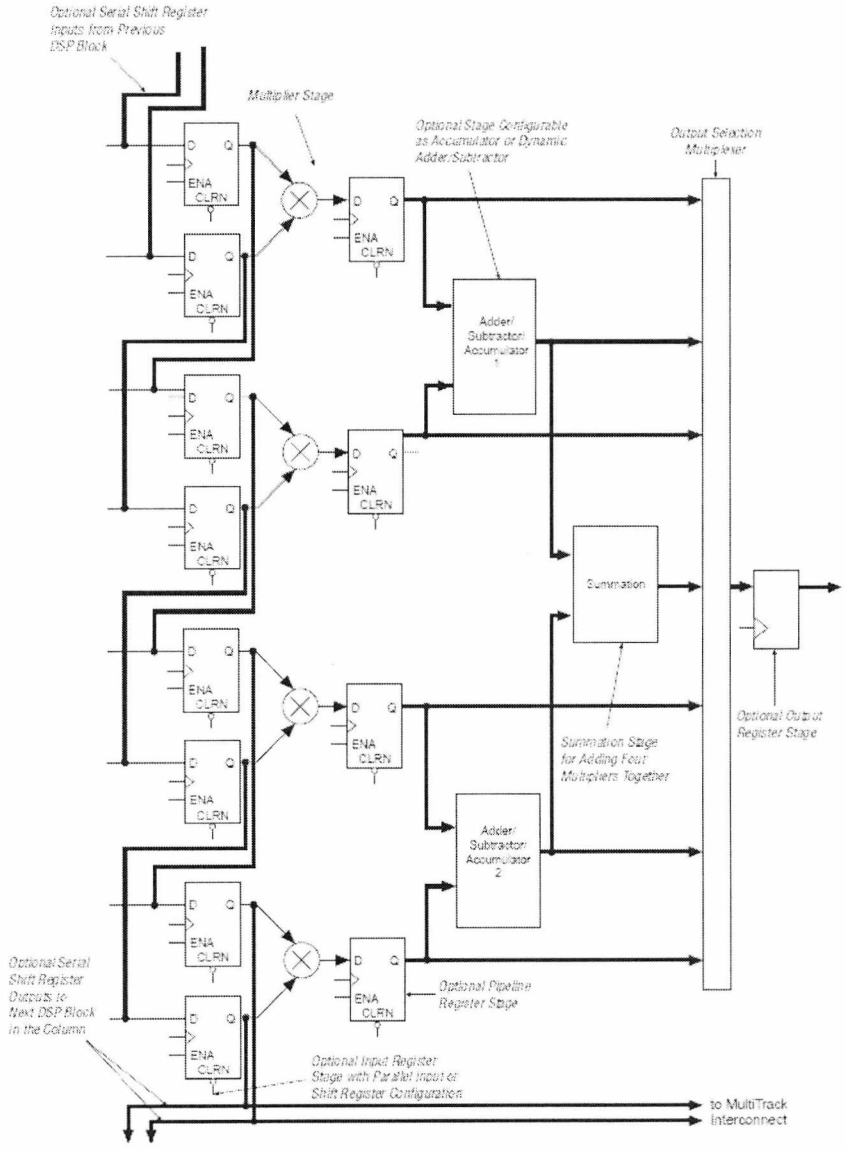


Figure 5.3-3: Altera DSP Block Diagram [137]

5.4 Number Representation on FPGAs

Traditionally, arithmetic calculations have been the most inefficient tasks to be implemented on FPGA designs. These will need to be carefully designed at a low level for optimisation. Subsequent chapters will present the many techniques developed to simplify arithmetic calculations.

The arithmetic implemented in hardware uses two main formats of binary numbers. The first format is floating-point, where the value is represented by a mantissa (fraction) and exponent as shown in Figure 5.4-1. The IEEE-754-1985 standard [138] defines two types of floating-point representation; these are single

precision, which uses 32 bits, and double precision, which uses 64 bits, as illustrated in the figure below.

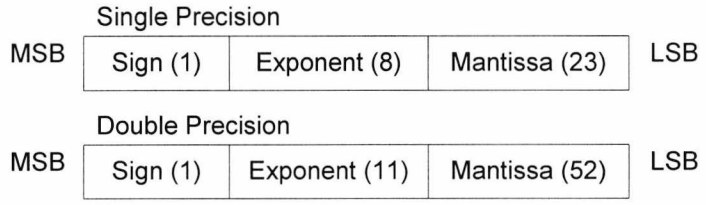


Figure 5.4-1: Floating-Point Representation

The order shown above ranges from the most significant bit (MSB) on the left, to the least significant bit (LSB) on the right. To allow for positive and negative values, the sign bit (S) is required. However, the exponent (E) can separately be positive or negative. To allow for this, the exponent is biased by -127 for single precision, and -1023 for double precision (B). This represents the number as shown in (5.4:1), where M is the mantissa.

$$x = (-1)^{s}(1+0.M)2^{E-B} \qquad \textbf{(5.4:1)}$$

This enables the representation of very large ranges of values with very good precision, particularly fractions, as these are explicitly represented. However, common arithmetic is difficult to implement due to the possibility of differing exponents. For FPGAs, this representation would be difficult to implement due to the requirement of 32 or 64 bits to represent values, and because of the non-trivial arithmetic [139], [140]. In most real-world applications, this range of representation is not commonly required.

More rational representations include unsigned binary, two's complement and sign-magnitude. Since these do not include any information on an exponent, in their current form, they can only represent integers. However, the least significant bit (LSB) need not represent 2^0. As long as the weights of the corresponding bits are the same and constant, the result will always be valid, although scaled by a power of two that is dependent on the scales of the input values and the arithmetic operation.

To represent fractions, the LSB represents a power of two that is less than zero. This will, however, decrease the range of values that can be represented. This is the principle of representing fractions in fixed-point binary. As long as the position of the virtual binary point is known and kept consistent for each node, all arithmetic will be valid.

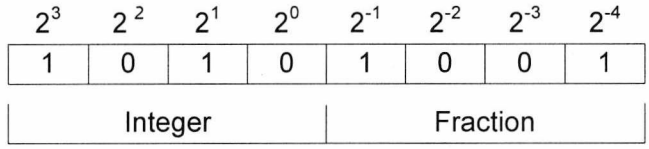


Figure 5.4-2: Fixed-Point Representation

In the example shown in Figure 5.4-2, the unsigned integer value represented if the LSB was weighted 2^0 would be 169. However, with the weighting shown, the value 10.5625 is represented. Although the scale is not shown, as long as all values in this node are known to be scaled down by 2^4, values from 0 to 15.9375 with steps of 0.0625 can be represented. This example shows the fixed-point representation with a format of (8,-4) which represents the total number of bits and the power of two weighting of the least significant bit respectively. The same principle holds for fixed-point variations of two's complement and sign-magnitude representations.

Instead of 255 being the highest number, the weight of most significant bit is also scaled down by a factor of 2^4 in this example. Therefore, when deciding the fixed-point format, the highest possible value must be able to be represented. All other bits left over may be used to represent fractions where a set number of bits are defined. Therefore the format is arbitrary and can be set depending on the maximum value and precision required. With a fixed format, arithmetic is easily implemented. For example, division by a power of two only requires a virtual shift of the binary point, and the binary value need not be changed (or rounded). For this reason, it is the preferred format for high-speed, low-level logic-based implementations including implementations on the FPGA.

So far the two main formats of binary numbers have been discussed. There is also a third format of binary numbers. This is the logarithmic number system [3]

which relies somewhere between the floating point and the fixed-point arithmetic. Logarithmic arithmetic will be used in this thesis for implementing the HT on an FPGA, and a more detailed description of it will follow in a subsequent chapter.

There are two main arguments why the logarithmic arithmetic is going to be used. Floating point arithmetic has a huge dynamic range and is relatively complex to implement on FPGA. Both the implementation of multipliers and adders in hardware are complex and require complicated logic structures. As it has been proved in Chapter 7, floating point arithmetic is far more accurate than is it actually needed for implemented the HT and such a large dynamic range is not required.

On the other hand, the fixed point arithmetic can be used to implement the HT, where the dynamic range can be controlled by the user, but a number of multipliers are required for equation 2.2:3 in Chapter 2, Section 2.2. One of the fundamental properties of logarithms is that multiplication in linear domain can be replaced by addition in the logarithmic domain (equation 1.3:1 in Chapter 1 Section 1.3). Therefore, by using logarithmic arithmetic, multipliers became adders. Even though there are lots of multipliers on FPGA's, there are still a limited number of them. Also, another reason for not using multipliers is that the size of them is significantly larger than it is actually needed. Logarithmic arithmetic may be is not that accurate, but it is relatively close to the fixed point arithmetic, and the conversion algorithms from logarithm to linear domain and visa versa are quite simple, allowing an efficient hardware implementation as shown in Chapter 6.

5.5 Xilinx's Xtreme DSP Blocks

In the past years, the techniques that Xilinx have been using to implement multipliers on FPGAs were the distributed arithmetic, and the constant coefficient multiplier, which was built in the LUTs. Following those methods, the 18 x 18 bits multiplier was developed (Spartan III), where the latest families from Xilinx (Virtex 4, Virtex 5 and Virtex 6) include high-speed dedicated

arithmetic blocks called DSP48s. Virtex 4 devices contain typically between 32 and 192 of these dedicated slices, although the highest specification Virtex 4, designed towards high-speed DSP (Digital Signal Processing) functions, contains 512 [10].

By appropriately configuring each DSP48 block, several operating modes, including multiply, add, multiply add and multiply accumulate, can be achieved. As these modes suggest, each DSP block contains a high-speed multiplier followed by a high-speed adder. The multiplier is 2's complement 18x18 bit, and allows a 36-bit result to be passed to the 48-bit 3-input adder. Figure 5.5-1 presents a detailed diagram of the DSP48 tile, consisting of two DSP48 slices, with full details available in the Xilinx Virtex 4 Xtreme DSP user guide [141]. The DSP48 blocks used in the Virtex 4 FPGA exist as pairs, and share the 48-bit adder input, as seen on the above diagram. Furthermore, between one of the 18-bit multiplier inputs, and the output which feeds to the next adder, there are high-speed dedicated interlinks, resulting in linked DSP48 blocks that form a high-speed DSP chain.

These DSP blocks can perform operations at up to 500 MHz by using the Digital Clock Manager (DCM) blocks, which are embedded to the FPGA. As DSP blocks use dedicated high-speed logic, generic programmable logic could be utilised for implementing a design. Nevertheless, the aforementioned are still there, whether used or not. However, depending on the requirements of the application, they can be connected to and used by the programmable logic. This allows for the implementation of high-speed arithmetic, without the use of programmable logic, and can therefore save many logic slices.

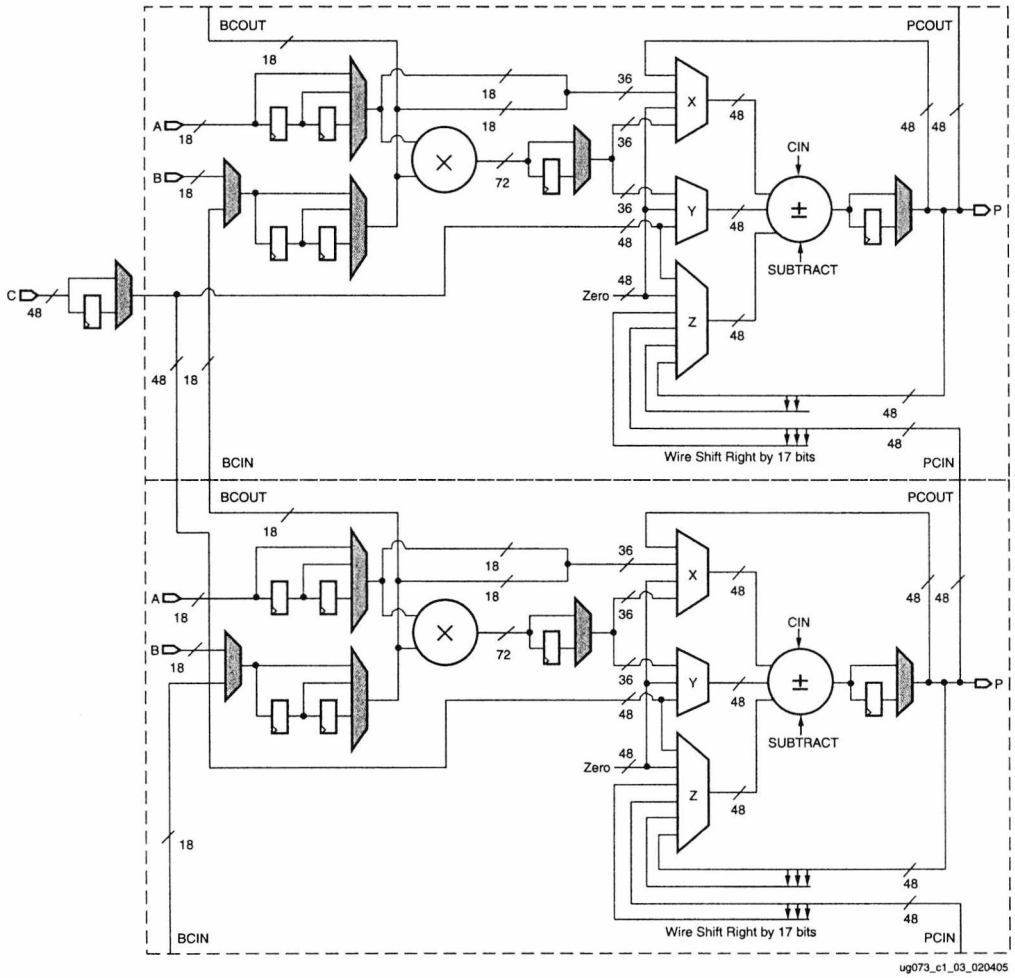


Figure 5.5-1: A DSP48 Tile Consisting of Two DSP48 Slices [141]

As has been shown in Chapter 7, Section 7.2.1, no more than 12 bits are required to perform the HT. The size of the DSP48 slices is 18 x 18 bits, which is far more accurate than is actually needed. As a result, the DSP48 slices are not going to be used for calculating the HT, and a number of them will be saved for other DSP requirements in the processing chain such as edge detection.

5.6 Conclusion

The FPGA, as demonstrated in this chapter, is a very versatile and powerful piece of hardware, capable of implementing large and complex logic-based designs. The FPGA is capable of implementing tens of thousands - up to millions - of logic elements on a single IC. In addition to this, many commonly used functions are still available, both as dedicated hardware within the IC, or as

subsystems that can be bought and combined with the existing design. This enables several components such as logic, memory, communication, clock management, standard interfaces and mathematical function, to be implemented as a "black box", to name but a few.

By describing the logic function, rather than giving a sequence of instructions, high-speed, concurrent operations can take place with predictable and optimisable timing and resource requirements. The optimisation component is what makes the FPGA different to other types of hardware that may also be used. Many alternative technologies that are able to implement similar applications execute a sequence of instructions, allowing for ease of development, at the expense of the ability to optimise. Each system or device described is useful for particular applications or circumstances.

Digital signal processor ICs are fit for use in high-speed, small-size and low-cost implementations of digital signal processing and arithmetic-based systems. These are not as applicable to logic-based systems, but the technology may be implemented within any other appropriate technology type.

ASICs are useful as a high-speed, low-power final product, but can become an expensive technology for development and prototyping as they are custom-built and not reconfigurable. It is common practice to design and develop an application on an FPGA, then port it to an ASIC once finalised and tested rigorously, in order to control for errors and mistakes.

Modern FPGAs now contain very fast and efficient dedicated DSP slices, capable of performing calculations without utilising logic cells that could, instead, be put to other uses. These DSP48 blocks are capable of operating at up to 500 MHz in ideal circumstances, and are often available, irrespective of whether used or not. Nevertheless, it has been shown that these units are designed to be configured for generic and common DSP architectures, and not the latest architectures developed for specific DSP functions

An FPGA platform is being used for testing all the different algorithms developed and described in this thesis. By using FPGA technology it can be shown that these algorithms are working in such a system and a future design (if required) of an ASIC it may be possible to implement.

The purpose of this chapter was to provide an overview of FPGA technology, its principles, and the reasons behind its usage. Since all hardware implementations proposed in chapters to follow are designed for the latest FPGA technology, the capabilities and mathematical implementations were therefore introduced in this chapter.

CHAPTER SIX

HYBRID LOGARITHMIC NUMBER SYSTEM

6.1 Introduction

John Napier [143] first introduced the logarithm in 1614 in theoretical mathematical fields of study, and its use has since then become increasingly widespread in many areas of research. For instance, logarithms can be utilized in many areas of science for the representation of different scales; namely the Richter scale in seismology, the decibel for acoustic and electric power measurement, entropy in thermodynamics and the pH in chemistry. Many responses such as the human interpretation of acoustic amplitude or the cent, minor second, major second, and octave for the relative pitch of notes in music, are also logarithmic [10].

In mathematical terms, the logarithm of a given number to a given base is the power or exponent to which the base must be raised in order to produce the given number as shown in equation 6.1:1 where b is the logarithm base. As such, the inverse of the logarithmic transform is the base to the power of the logarithm of the number. Typical values for the base are e (called the natural log with a value of 2.718), 10 (or the common log) and 2, which will be represented throughout this chapter. The logarithmic base must be positive, but not 0 or 1.

$$x = b^{\log_b(x)} \qquad \textbf{(6.1:1)}$$

As can be seen in Figure 6.1-1, for various logarithmic bases, with X ranging from 1 to 10, the range of the transformed Log(X) is much smaller than 10. However, below 1, the transformed range is much greater than 1 and for that

reason care must be taken with output precision when transforming from logarithmic to linear domain.

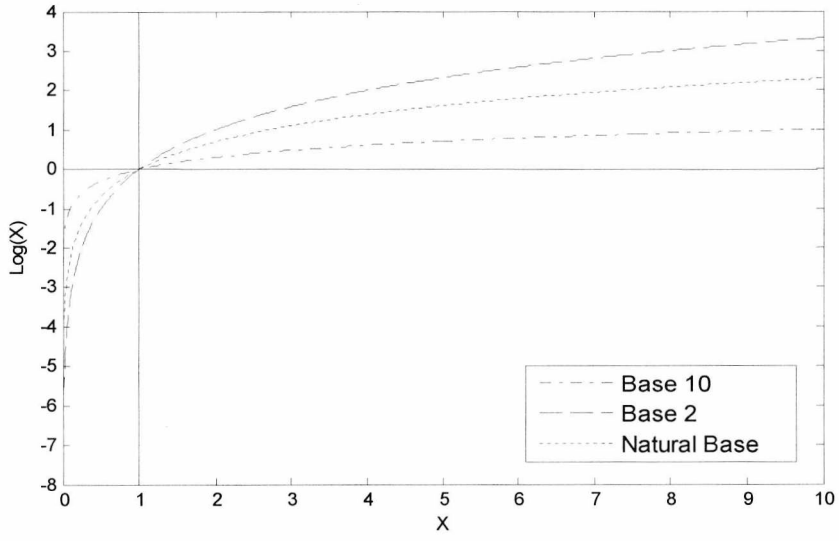


Figure 6.1-1: Graph of Log(X) for $0 \leq X \leq 10$

Table 6.1-1 below shows various values for X, and their logarithmic equivalents. All of these values are independent of the logarithmic base.

Table 6.1-1- Logarithm Conversion Rules

X	Log(X)
<0	NaN
0	-Inf
<1	Negative
1	0
Inf	Inf

The representation of large ranges of values with a smaller range of values is facilitated by the non-linear nature of logarithms. The logarithmic transform response will be greater at lower values, where variation is most significant, than at higher values, presuming that the same amount of change occurred. For example, working in base 10, the change from 100 to 50 (in linear domain) is 50 (reduction of 50%) and the change from 1000 to 950 is also 50, but this is only a reduction of 5%. Converting these linear values to logarithms, the change from 2 to 1.69 (in logarithmic domain) is 0.31 and the change from 3 to 2.977 is 0.023.

These fundamental properties may appear over-simplistic at first; however, their implications will be made clear later in this chapter. Moreover, in this chapter the main property of logarithms will be outlined, namely, their arithmetic properties.

It can be seen in the equations 1.3:1 and 1.3:2 in Chapter 1 Section 1.3 (where b is the logarithm base) that multiplication in the linear domain becomes addition in the logarithmic domain, and similarly, division becomes subtraction. For a simple example of multiplying 5 by 6 using 1.3:1, we can see that:

$$\log_{10}(5) + \log_{10}(6) = 0.69897 + 0.77815 = 1.47712$$

$$10^{1.47712} = 30$$

Therefore 5 and 6 have been multiplied without doing any multiplication. Division is similarly calculated. This relationship can have a large impact on hardware in terms of calculations. Multiplication in hardware is an important operation that may require shared logic or resources that can be in much demand, therefore, rendering it inefficient with respect to speed. Logarithms provide the solution with regards to the aforementioned problem; multiplication can be performed on two numbers without actually multiplying or using any hardware specifically associated with it. Addition can be used instead, where the numbers are converted to and from the logarithmic domain. The result, if required, can be transformed back to the linear domain.

This very simple logarithmic method works well for multiplication and division. Multiplication and addition are both commonly used for Digital Signal Processing (DSP) algorithm, especially for filter implementations. The method of implementing addition and subtraction in the logarithmic domain [7] is much more complicated as can be seen from (6.1:2) and (6.1:3).

$$\log_b(A+B) = i + \log_b(1+b^{j-i}) \quad \textbf{(6.1:2)}$$

$$\log_b(A-B) = i + \log_b(1-b^{j-i}) \quad \textbf{(6.1:3)}$$

Where: $i = \log|A|$ and $j = \log|B|$

Taking (6.1:2) as an example of adding 600 to 200 in logarithm base 2, we get:

$$\log_2(600+200) = 9.2288 + \log_2(1+2^{7.6439-9.2288}) = 9.6438$$

$$2^{9.6438} = 800$$

As seen above, this is not a straight forward equation, as the non linear functions (6.1:4) and (6.1:5) must be evaluated.

$$F(r=j-i) = \log_b(1+b^r) \quad \textbf{(6.1:4)}$$

$$F(r=j-i) = \log_b(1-b^r) \quad \textbf{(6.1:5)}$$

A graph of these functions in Figure 6.1-2 illustrates the singularity in $f_s(f)$ that makes linear approximation difficult. While $f_a(r)$ is well behaved near 0, $f_s(r) \rightarrow -\infty$ as $r \rightarrow 0$ [144], [145].

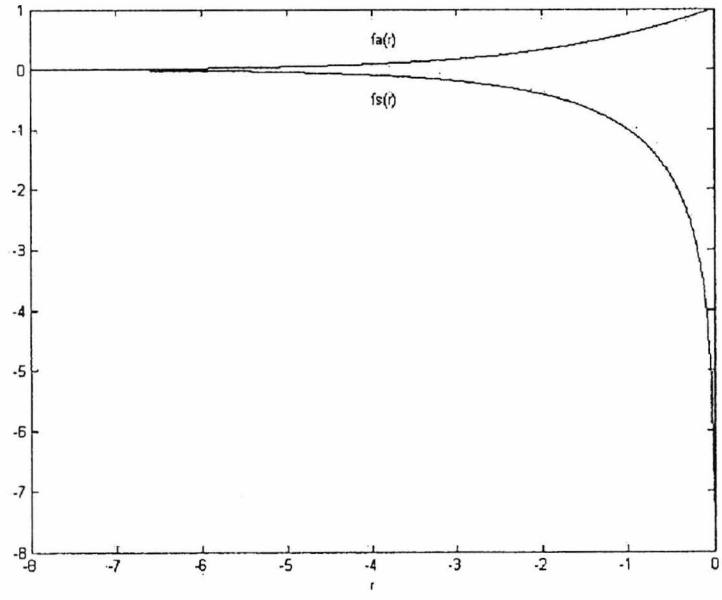


Figure 6.1-2: Non Linear Functions

It is commonly accepted [146] that the use of logarithmic arithmetic for hardware implementations, of signal processing in particular, is in need of further research and utilization. To the best of this author's knowledge at the time this chapter was written, a study of the impact of performing the HT on FPGA technology, whilst taking advantage of logarithms has not been reported in the literature. The

only report of the use of logarithms and FPGA technology can be found on [10]. This chapter will be dedicated to the investigation of the aforementioned impact.

The next section will describe how the HT can be optimally implemented on FPGA technology.

6.2 Logarithmic Converter Design

6.2.1 Design Considerations

In order to design the HT, certain considerations must be taken into account. For instance, the speed at which the device can operate is important. Such high speed can be achieved by using dedicated hardware rather than a software alternative. Hence, the hardware option is chosen for the aforementioned reason and it should be of high speed. At the same time, hardware space is finite, and shared resources on FPGAs may be required for other modules on the same device. As such, the logic resources must be minimised. This minimization will also result in a reduction of the power consumption of the device. This action will therefore make the device suitable for battery powered portable applications, or applications where wider processing is required in the same device.

For a solution to be a state-of-the-art, the proposed methods of the new solution must have advantages over current modern ones. For this reason, current and relevant solutions will be selected, described and compared in the next part of this thesis. As shown in the preceding literature review, there are many methods of converting between logarithmic and linear domains. A method that is deemed appropriate will be selected as an example for this thesis, although the structure proposed may be implemented using any appropriate conversion method. The test will take into account practical considerations for hardware implementation on FPGA technology.

6.2.2 Logarithmic Converter

Full logarithmic arithmetic is used, for all calculations, e.g. addition, subtraction, multiplication and division being computed with the logarithmic number system (LNS). However, in one of the previous papers, a more rational approach has been taken [3]. A system known as the Hybrid-Logarithmic Number System (Hybrid-LNS) uses very simple arithmetic steps, effectively and efficiently, using LNS to make multiplications, and linear numbers to make the addition calculations.

The "Leading Zero Detector" (LZD) [160], [161] produces three values using simple logic. A single bit flag, indicating the sign of X, is produced. Trivial logic is required for sign-magnitude numbers to output the sign bit of X. According to the rules previously outlined in Table 6.1-1, at the beginning of this chapter, negative numbers cannot be converted to logarithms. Nevertheless, negative numbers can be multiplied; hence, the converter must be able to convert them. For negative values of X, the absolute value is converted, and the flag indicates the occurrence.

Similarly, a zero value of X has no logarithmic equivalent, although multiplication by zero can be possible. Again, a value of zero in the LZD can be detected using simple logic. With a zero value of X (or a value too small to be represented by anything other than zero), no conversion occurs, and this is indicated by the 1-bit "Zero" flag.

The LZD also produces an integer representing the position of the most significant '1' (not including the sign bit) in the binary number X. This value also represents the integer part of the logarithm of X (for base 2). To simplify the logic, this value is biased positively so that the "Integer" value is never negative, as has been implemented in [3].

The "Barrel Shifter" shifts X to the right by the unbiased "Integer" number of places. Therefore the output of the "Barrel Shifter" (S) will have the equivalent decimal value $1 \leq S < 2$. Because the binary position weighted 2^0 will always have

the value of '1' (inferred bit), this bit can be ignored and only the bits representing the fraction of *S* will be converted in the look-up-table (LUT). The linear to logarithm converter is depicted in Figure 6.2-1 below.

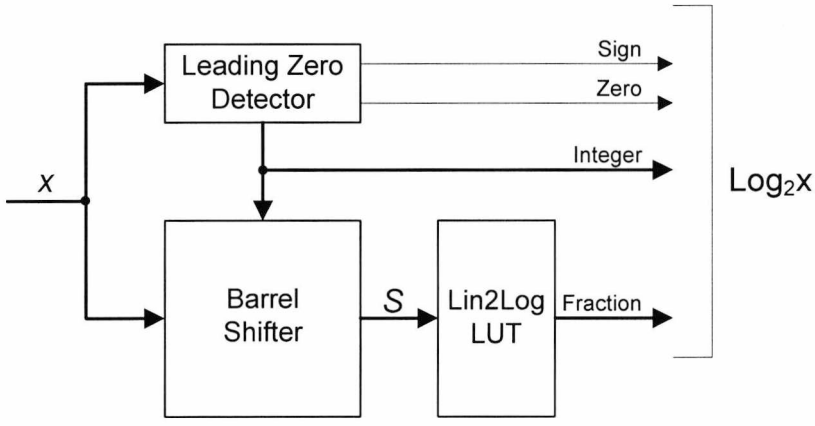


Figure 6.2-1: Linear to Log Converter Block Diagram [3]

The contents of the LUT are the rounded-to-fit, pre-calculated (offline) logarithmic equivalents to the input *S*. The proposed solution in this chapter uses a set relation between the bit-widths of the address and the contents, although they do not necessarily have to be related to each other. Therefore the address for the LUT is provided by *S*, and the fraction part of the logarithm result is provided by the contents of the LUT. As mentioned before, if the LUT is unfeasibly large for the hardware, further logic can be used to interpolate intermediate values in the LUT, something which for small LUTs would be unnecessary.

Table 6.2-1: Example Linear to Logarithm Converter LUT

Fraction (*A*)	LUT Address	$Log_2(A+1)$ (rounded to 8 bits)	Contents $Log_2(A)$ (2^8)
0	0	0	0
0.125	1	0.171875	44
0.25	2	0.3203125	82
0.375	3	0.4609375	118
0.5	4	0.5859375	150
0.625	5	0.69921875	179
0.75	6	0.80859375	207
0.875	7	0.90625	232

As an example illustrating this method, is given in Table 6.2-1 utilizing an LUT with three address bits, and a width of eight bits. The address represents fractions between 0 and 1 at equal intervals. The more bits in the LUT address and contents, the more precise and accurate the conversion will be.

To convert the linear value -164 to logarithm (base 2):

-164 = 110100100 in binary (sign magnitude)

This is non-zero, therefore ***ZERO*** = **'0'**, and is negative, therefore ***SIGN*** = **'1'**

The leading one is in the seventh bit position (not including the sign bit) with significance 2^7, therefore:

INTEGER = **7**

After scaling down by 2^7 and the removal of the leading '1', the binary number is 0.0100100, but with the fraction bits rounded to three bits gives 010 taking the fraction bits in isolation, which becomes the address of the LUT (address 2), giving the output 82 x 2^{-8} according to Table 6.2-1.

Therefore, ***FRACTION*** = **0.3203125**

Theoretically $\log_2(164) = 7.357552$. With this method using an 8 x 8bit LUT 7.3203125 is given.

As only eight fractions are given in this example, the LUT gives a close approximation. If more fractions were to be stored in the LUT, the result will be much closer.

The logarithm-to-linear converter is shown in Figure 6.2-2. This is effectively an inverse transform, a mirror of the linear-to-logarithm converter, and works much

in the same way. Two of the main components are identical, with the "Sign-Magnitude Converter" (SMC) using simpler logic than the LZD.

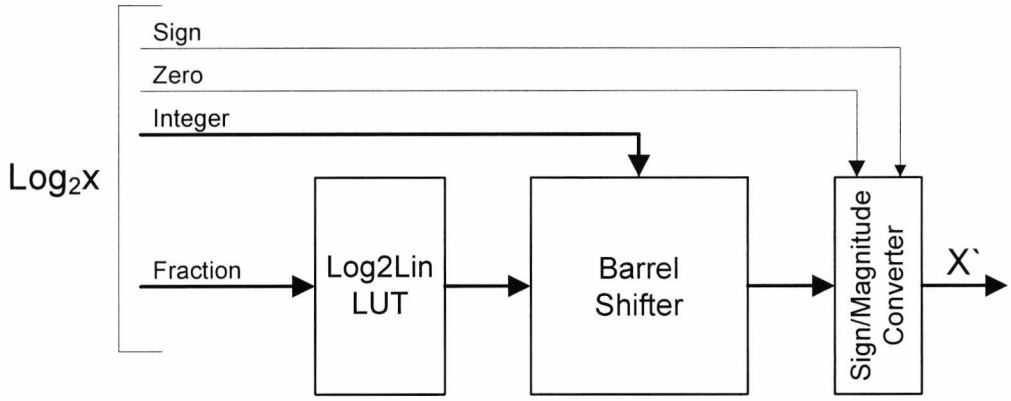


Figure 6.2-2: Log to Linear Converter Block Diagram [3]

Using the LUT, the logarithm fraction is firstly converted into the linear fraction. The logarithm-to-linear LUT follows the same rules as the linear-to-logarithm LUT. The length and width need not match or be related, but the larger these are, the more precision and accuracy will be achieved in the conversion. The fraction provides the address for the LUT ROM, while the contents contain the closest equivalent linear value. The leading '1' is not necessary to be stored in the contents of the LUT.

The Barrel Shifter will shift the output of the LUT with the leading '1' inserted "*Integer*" places to the left or right, depending on the biased value. For positive, non-zero linear values, this will give the correct converted value. However, if the linear value is zero (from the linear to logarithm converter output flag), the output X, will be set to zero. The Sign flag will also be added to the sign bit of X where sign magnitude representation is used (or used for negative conversion for 2's complement representation).

Taking the result of the linear-to-logarithm converter as an example for the logarithm-to-linear converter:

Fraction **= 0.3203125**

Integer **= 7**

***Zero* = 0**

***Sign* = 1**

A simple LUT for the logarithm-to-linear converter is shown in Table 6.2-2. Again this is an 8 x 8-bit LUT. It must be noted that in terms of accuracy and precision, this will produce large errors. However, it is an example for illustrational purposes, demonstrating the said operation.

Table 6.2-2: Example Logarithm to Linear Converter LUT

Fraction (A)	LUT Address	2^A-1 (rounded to 8 bits)	Contents 2^A-1(2^8)
0	0	0	0
0.125	1	0.08984370	23
0.25	2	0.18750000	48
0.375	3	0.29687500	76
0.5	4	0.41406250	106
0.625	5	0.54296875	139
0.75	6	0.68359375	175
0.875	7	0.83593750	214

The LUT input will be 01010010 (*fraction*), which rounded to three bits gives 011 and forms the address of 3 on the LUT.

Therefore the LUT output will be 76 scaled up by 2^8, or 0.296875.

By inserting back the inferred '1', this becomes 1.296875.

The Barrel Shifter will shift this *Integer* places to the left (in this case multiplying by 2^7), giving 166.

Due to the sign flag, the sign bit becomes significant in the SMC, giving the final value of:

***X`* = -166**

X = -164.

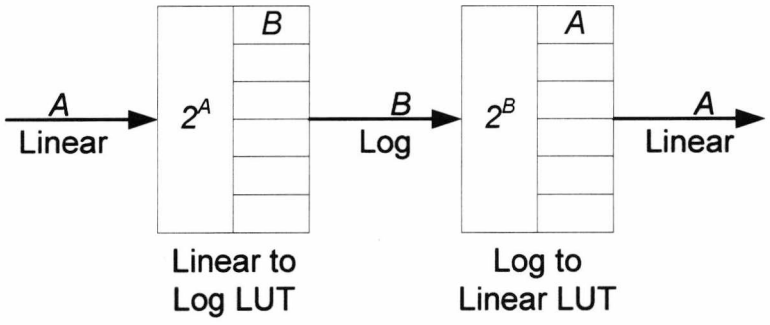


Figure 6.2-3: LUT Shapes

Although this uses converter of an 8 x 8-bit LUT, precision and accuracy will be affected by the size and aspect ratios of the LUTs. A test, based on the examples shown, was devised in order to demonstrate the above. Input values X ranging from 1 to 256, were converted to logarithms and back using LUTs as described above. Figure 6.2-3 exemplifies how the width of the linear-to-logarithm converter corresponds to the length of the logarithm-to-linear converter, and vice-versa. The LUT lengths and widths were varied from 2 to 8 bits, although the width and length of the linear-to-logarithm LUT kept the same as the logarithm-to-linear LUT as shown in Figure 6.2-3. The mean square error (MSE) as shown in (6.2:1) of these results is tabulated in
Table 6.2-3. In this case, $n = 256$.

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(X_j - X'_j)^2 \qquad \textbf{(6.2:1)}$$

Table 6.2-3 – MSE results for conversion to and from log / linear for different size / shaped LUTs

		Logarithmic bit-width (B)						
		2	3	4	5	6	7	8
Linear bit-width (A)	2	352.1	200.5	48.9	48.9	48.9	48.9	48.9
	3	391.3	166.6	51.6	12.3	12.3	12.3	12.3
	4	541.4	138.4	41.1	18.9	3.2	3.2	3.2
	5	490.8	129.0	40.1	12.5	4.9	0.9	0.9
	6	512.0	131.3	40.0	10.4	3.1	1.6	0.3
	7	480.4	124.4	33.8	8.6	2.1	0.6	0.2
	8	485.2	126.8	34.5	8.4	2.0	0.6	0.1

It is clear to see from Table 6.2-3 that without enough bits to represent the linear values (linear-to-logarithm LUT length and logarithm-to-linear LUT width, or A in Figure 6.2-3), the errors can see a limited reduction as the logarithm's bit-

above four or five. When the logarithmic bit-width (B) is A+2 or above, no additional advantage is given with respect to the errors.

6.2.3 Logarithmic Multiplication Implementation

Hybrid-LNS implement the equivalent of multiplying by using adders. However, these adders cannot take into account the Zero and Sign flags unassisted; extra elementary logic is required as shown in Figure 6.2-4. Multiplying two negative numbers gives a positive answer, as does multiplying two positive numbers, and a negative multiplied by a positive gives a negative. Therefore an XOR is used on the two sign flags for the numbers being multiplied. Similarly, if either value being multiplied is zero, the result will be zero, and therefore an OR gate will be used on these flags.

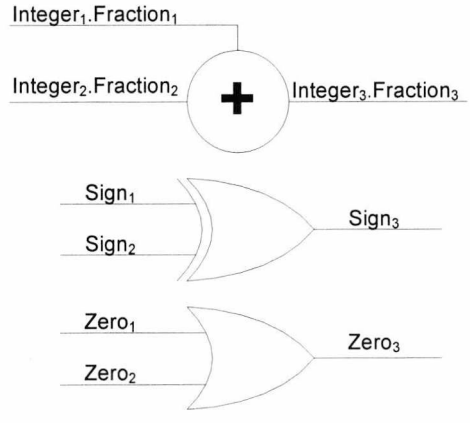


Figure 6.2-4: Logic for Multiplication in the Logarithmic Domain

As an example, in Figure 6.2-4 two linear numbers that are to be multiplied using Hybrid-LNS are converted using separate linear-to-logarithm converters as described above. Each gives four signals as shown in Figure 6.2-4. The subscripts 1 and 2 denote the signals from each linear-to-logarithm converter and subscript 3 denotes the input to the logarithm-to-linear converter. The integer and fractions from each linear-to-logarithm converter are combined before being summed together. The resulting four signals would then be used by the logarithm-to-linear converter to produce the result of the logarithmic addition.

6.3 Conclusion

The aim of this chapter was to provide background information on the underlying principles of logarithms for arithmetic, intending towards hardware implementation. From the basic principles of logarithms, it can be seen that multiplication, a process often difficult and slow to implement in hardware, can be effectively replaced with the addition of logarithms. This can also be used in a similar way for division (although most DSP applications only use multiplication), although addition and subtraction have been shown to be not so simplistic in their implementation with logarithms.

This part of this thesis also had a detailed look at the implementation of an efficient linear / logarithmic converter with the use of an example. Furthermore, optimisation of the implementation with respect to the look-up-table design was examined, particularly addressing the effect of different aspect ratios. From the results shown as part of this research, LUT sizes of address bits = width bits – 2 for the linear-to-logarithmic converter LUT (and opposite for the logarithmic-to-linear converter LUT) have been shown to give optimum results where any more address bits would give no further advantage in conversion quality. This is convenient for when the LUT width is said to be equivalent to the width of the linear representation, then the address bits of the LUT will be the same as the linear bit-width after removing the sign and inferred bits.

As clearly demonstrated in this chapter, the most suitable arithmetic operation is multiplication. The implementation of multiplication, using logarithms with respect to hardware implementation, has also been described. The background information presented in this chapter sets the foundations for the HT implementation using the Hybrid-LNS that will be outlined in the next chapter.

CHAPTER SEVEN

HYBRID-LNS & HOUGH TRANSFORM

7.1 Introduction

A detailed introduction to logarithms, as well as, conversion and arithmetic implementations has been discussed in the previous chapter. Multiplication can be effectively replaced with the addition of logarithms. Using the Hybrid-LNS, multiplication is performed in the log domain and addition performed in the linear domain. This chapter will apply the methods investigated in the previous chapter to the SHT. The effects that logarithmic arithmetic will have on the structures and quality of results of the HT have not yet been investigated in current literature, and it is this that this chapter will contribute.

It has been shown previously [12], [13] how the HT can be implemented on technology such as the FPGA. These solutions typically use multipliers on the FPGA fabric to calculate the HT. This chapter will describe how, by using logarithmic arithmetic, the need for multipliers is eliminated, while precision of the algorithm is maintained. Also, an implementation of the HT on FPGA using Hybrid-LNS arithmetic will be described in the following sections and a comparison with the linear HT will be made. The results will be presented in sections 6.3 and 6.4, with a summary and conclusion in section 6.5.

7.2 The Linear Hough Transform

The implementation of the HT, as it has been discussed in the Chapter 2, Section 2.2, is based on the equation 2.2:3, where the set of lines passing through each point in an image is represented by a set of sinusoidal curves in (ρ, θ) parameter space. Although, elegant in concept the HT is beset by a number of computational problems when applied to real-time image processing of large

images: the computation of all possible values of ρ and θ for each point in the image and the accumulation of the results for all possible points in (ρ, θ) parameter space. Evaluation of equation 2.2:3 for each pixel element in the image represents the fundamental signal processing operations required for implementing the HT. A direct form, as shown in Figure 7.2-1, requires the use of two multipliers and an adder. The contents of the LUT are the pre-calculated (offline) logarithmic equivalents for sine and cosine respectively, rounded to fit.

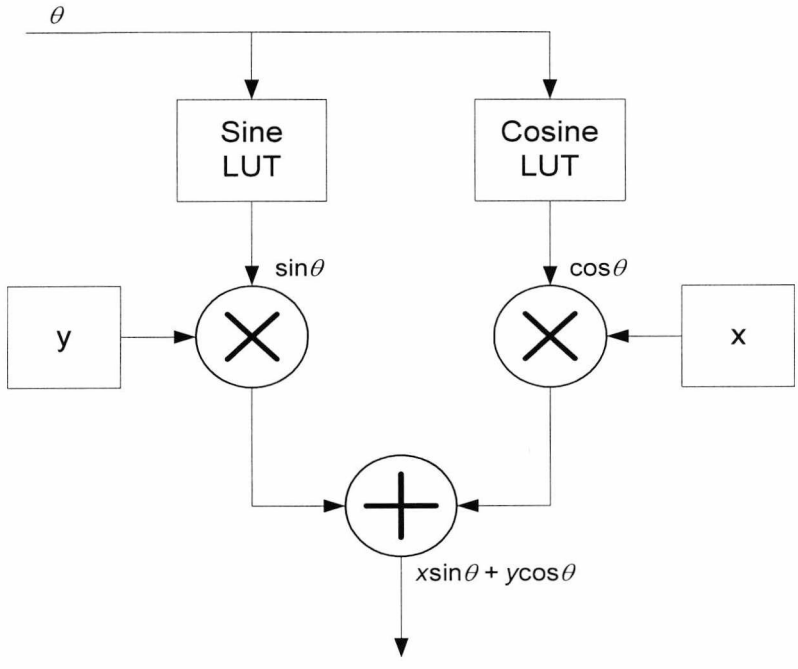


Figure 7.2-1: Basic HT Calculation Element

However, the choice for implementing the function shown in the above figure is not ideal, as multiple structures are needed to process image data at mega-pixel resolution images. Using a Pulnix AccuPIXEL monochrome camera with a resolution of 1024 x 1024 pixels at 25 frames per second the input frequency is 1024 x 1024 x 25 Hz ≈ 26MHz. For each pixel to be calculated for, say 180 angles, requires 1024 x 1024x 25 x 180 ≈ 4.7 x 10^9 operations per second. To be realistic 32 or 64 Hough elements working in parallel would be required to achieve these frame rates as it can be seen from Table 7.2-1.

Table 7.2-1: Correlation Between Number of Hough Elements with Operations per Second.

Hough elements	Operations per second
1	4.7×10^9
2	2.35×10^9
4	1.175×10^9
32	146.9×10^6
64	73.4×10^6

Furthermore the longest possible line in the image is $\sqrt{1024 \times 1024} = 1024$ bits. This requires each accumulator cell in the PSF to be at least 10 bits wide. If 180 angle increments are required, then according to Matlab® software routines there must be 2896 x 180 = 521280 accumulator bins. If each accumulator need to be 10 bits then this represents 325.8 Kbits of memory (each bin store 16 addresses). From the above example it can be seen that an alternative architecture, using the general FPGA fabric (the CLBs or Configurable Logic Blocks), could produce a more efficient solution. A potential advantage of this approach is that it would leave the DSP slices and Block RAM free for other functions in the image processing chain.

7.2.1 Proposed Linear Implementation Using Matlab®

Simulation of the results for the proposed implementation will be made on Matlab® with software routines defined in [162]. Standard test images [163] will be used to obtain the results, as have been used in other, similar papers, as well as, captured images using the Pulnix 1280 x 1024 monochrome camera. These are shown in Figure 7.2-2. Each original image is 1024 x 1024 pixels, 8 bits per pixel and greyscale. Each image is imported into Matlab® with the imread() function. A first step is to look the effects of using finite precession arithmetic for the HT on the test images. For a full set of results for these tests, please refer to Appendix A. A representative sample of two images is provided in this chapter for analysis. Figure 7.2-2 shows the test images captured by a Pulnix camera. It was aligned at different angles to the camera, 60 and 45 degrees respectively [14].

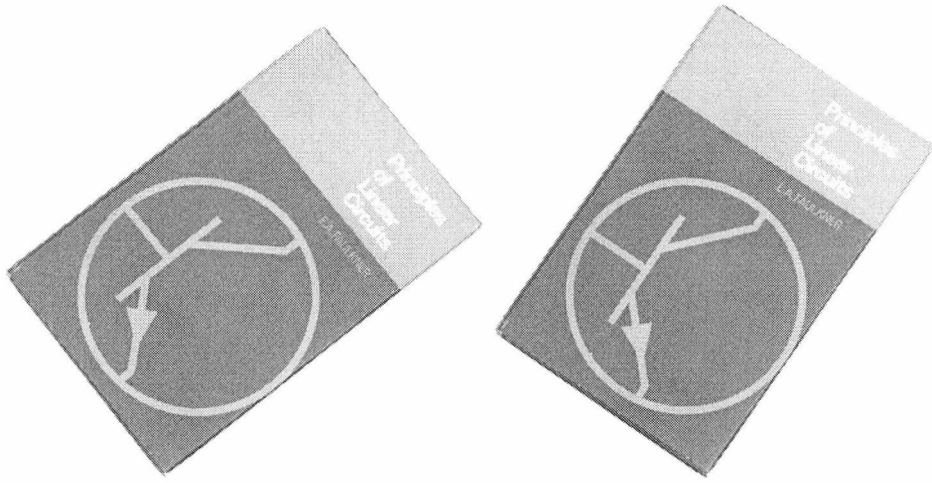
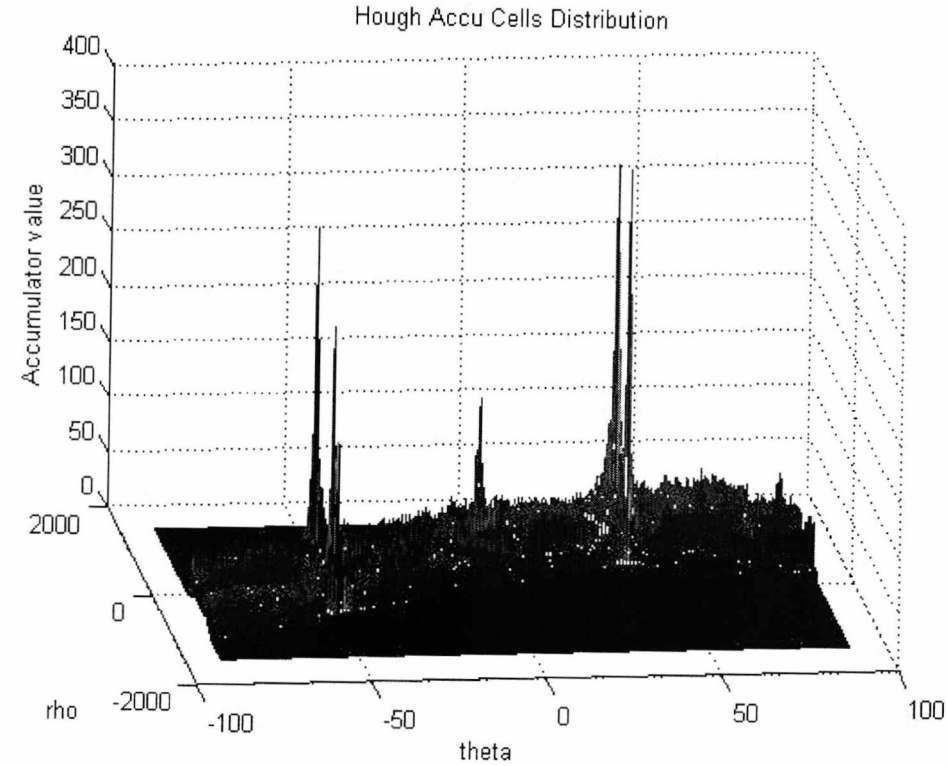
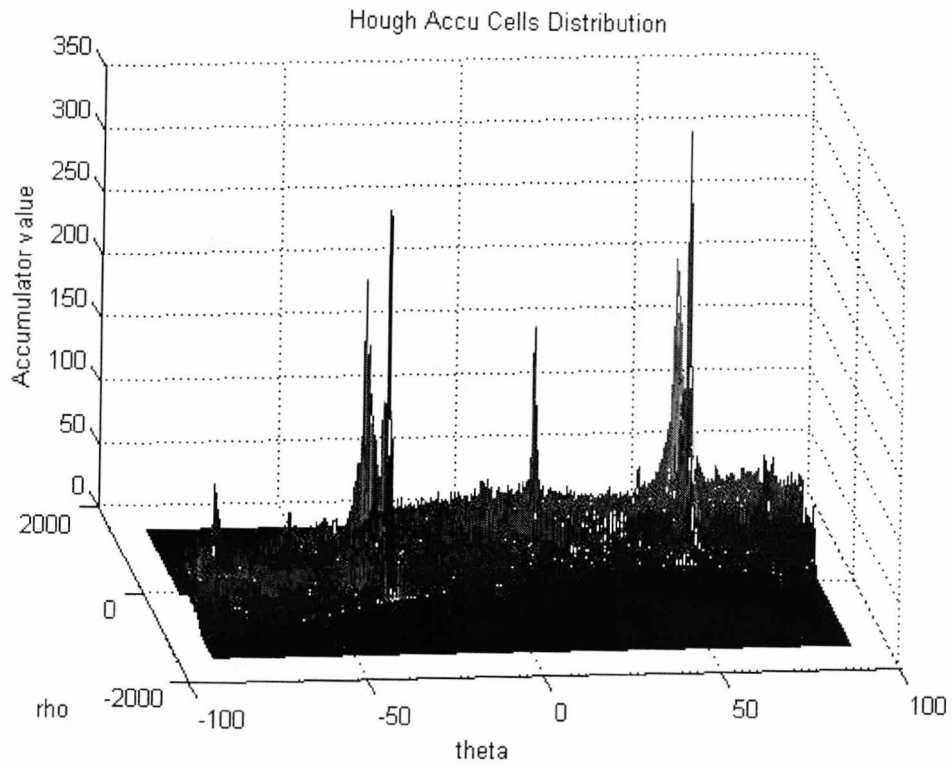


Figure 7.2-2: Test Image at Rotated Angles of 60 and 45 Degrees.

After a series of tests, which can be found in Appendix B, with different fixed points (4-bit, 8-bit, 10-bit, 12-bits, 16-bit and 20-bit), it was concluded that: below 8-bits of precision the results were unsatisfactory, as information of the images was lost and above 12-bits of precision the results were the same as the floating point ones. For this reason it was decided that for the entirely tests 8-bits and 12-bits of precision will be used. Also the effect of using the round function in comparison with the fix function was tested, and is shown that using the round function the results were closer to the optimum ones. The parameter space was calculated using Matlab® with floating point precision and again using 8-bit and 12-bit fixed point arithmetic for the values of the sine and cosine stored in the LUTs. Figures 7.2-3, 7.2-4 and 7.2-5 show the outputs of these calculations for 45^{o} and 60^{o}.

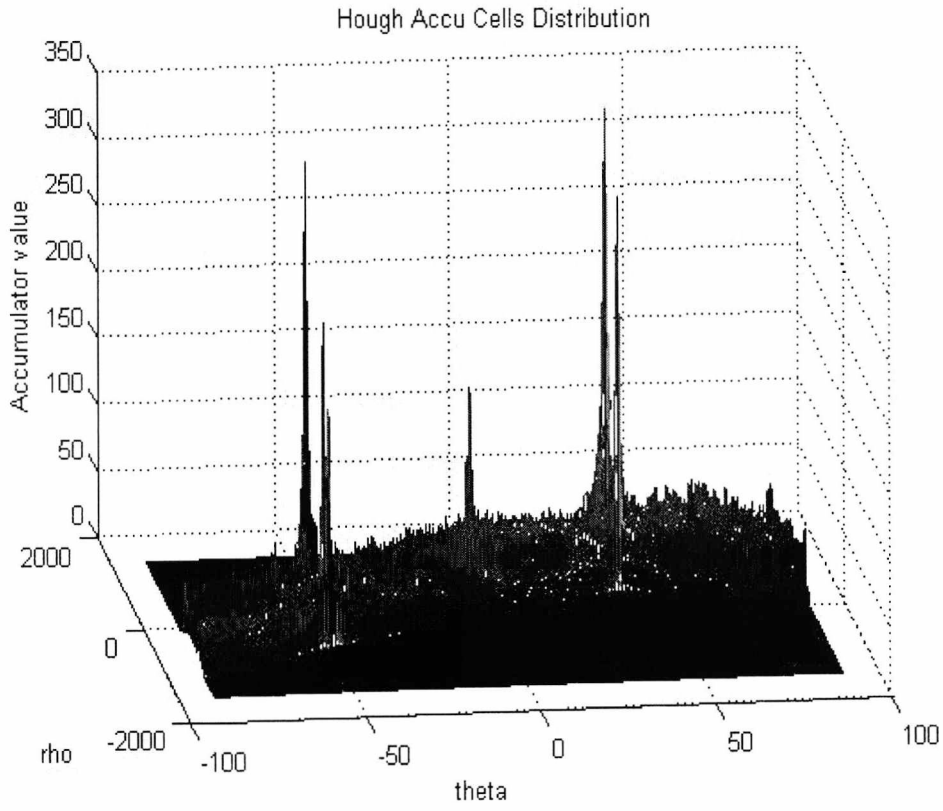


(a) Floating Point calculation at 60°

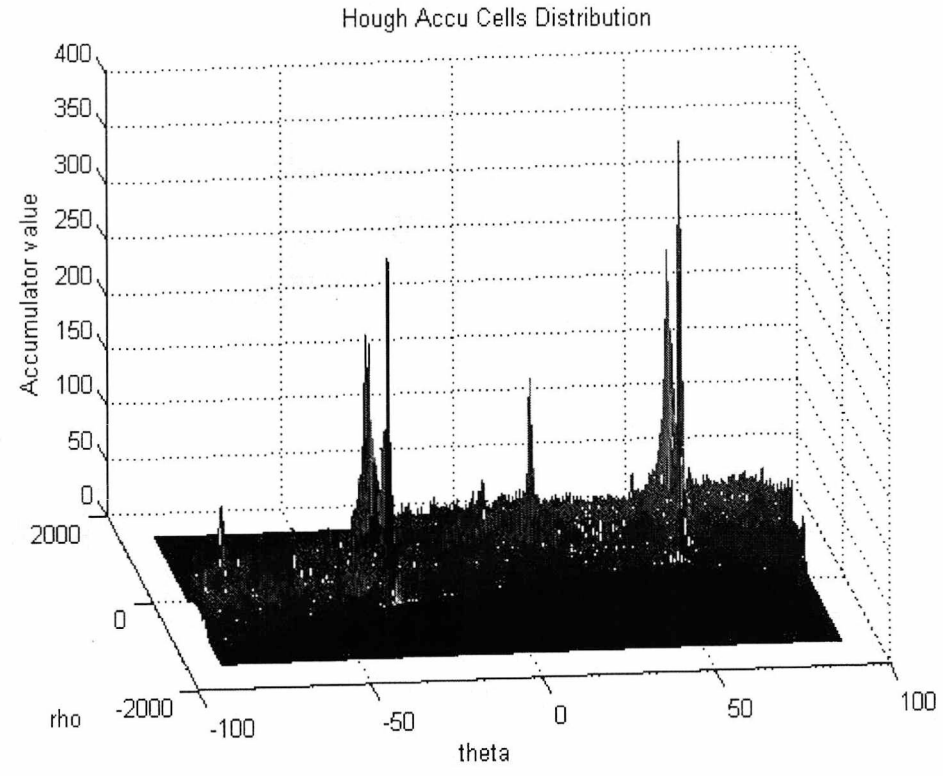


(b) Floating Point calculation at 45°

Figure 7.2-3: Hough Transform Parameter Space Output Graphs on 1024 x 1024 Binarised Images Using Floating Point Arithmetic Precision.

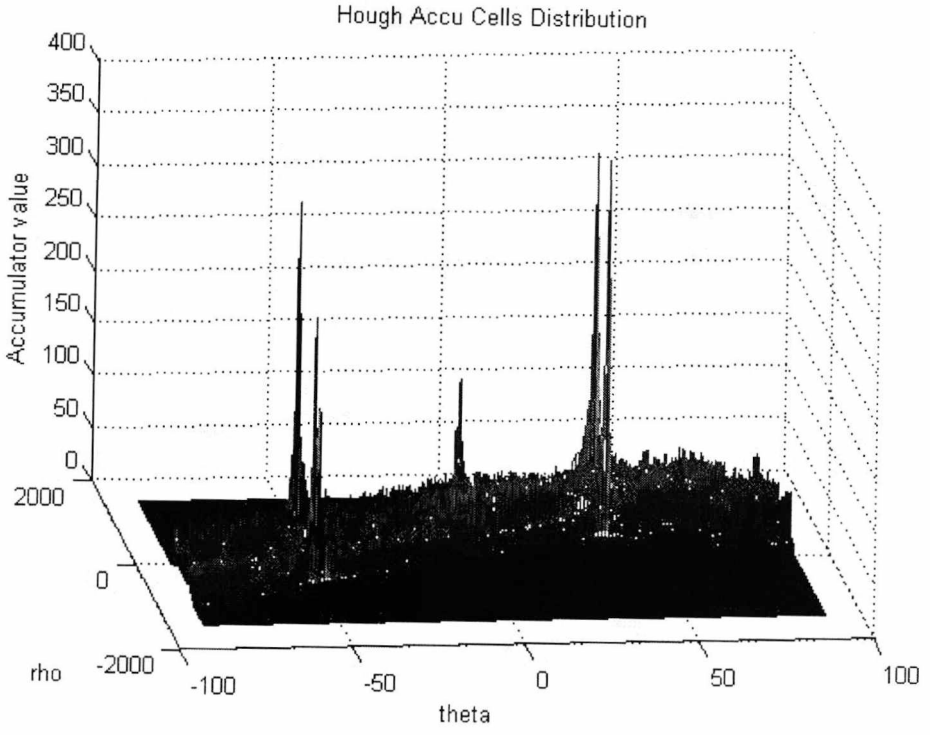


(a) Linear calculation 8 bits (rounding) at 60°

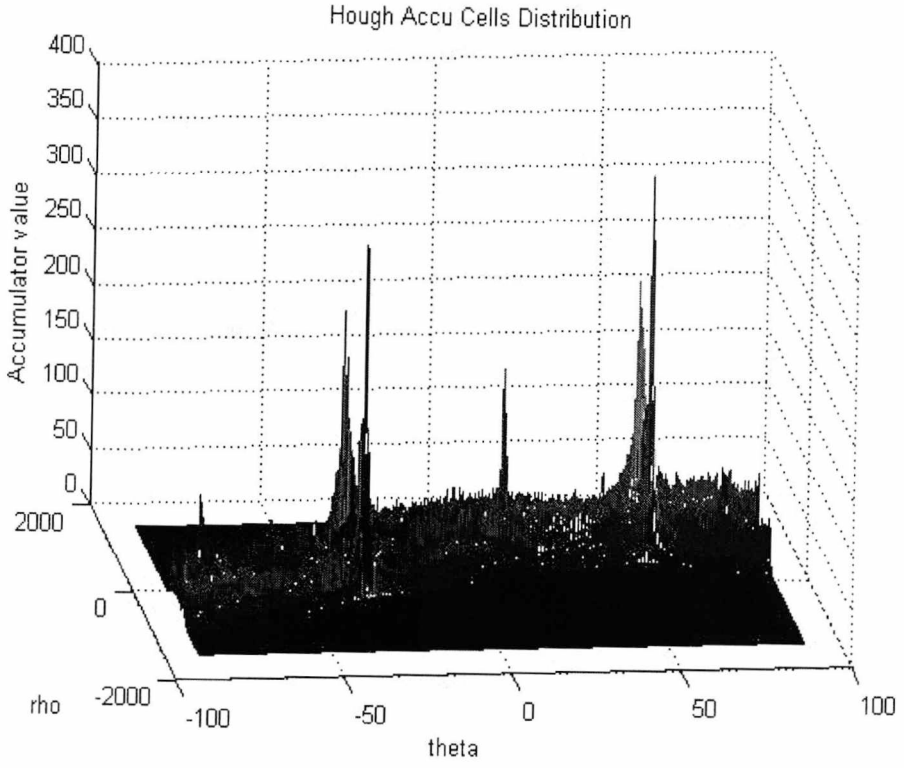


(b) Linear calculation 8 bits (rounding) at 45°

Figure 7.2-4: Hough Transform Parameter Space Output Graphs on 1024 x 1024 Binarised Images Using Linear 8-bit Arithmetic Precision.



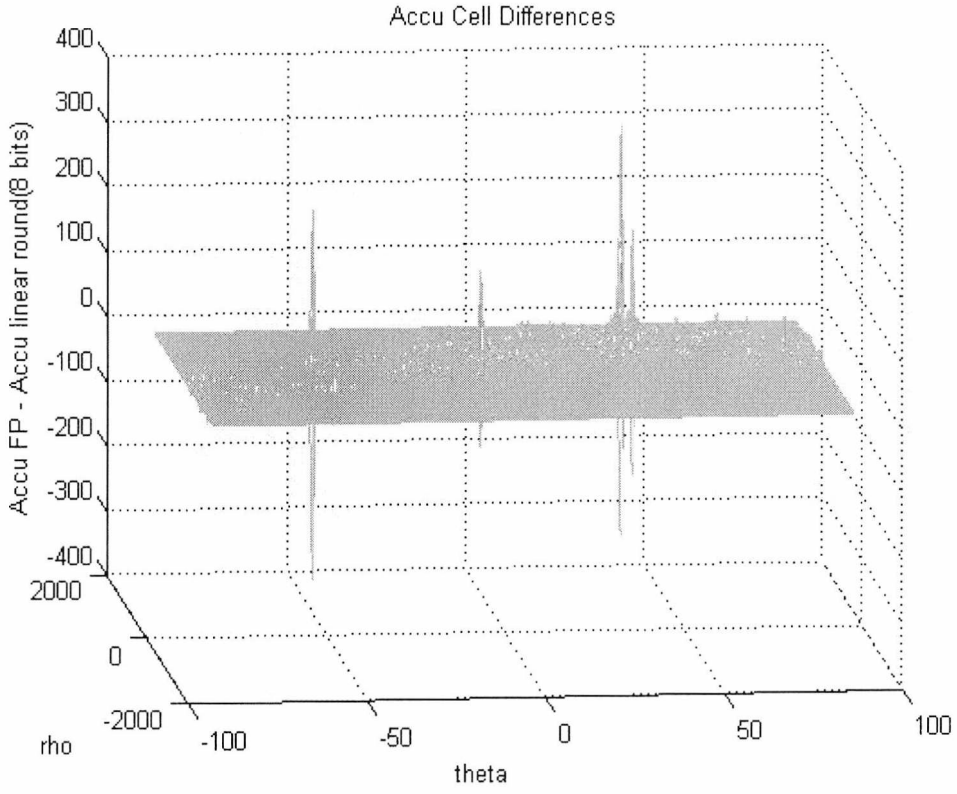
(a) Linear calculation 12 bits (rounding) at 60°



(b) Linear calculation 12 bits (rounding) at 45°

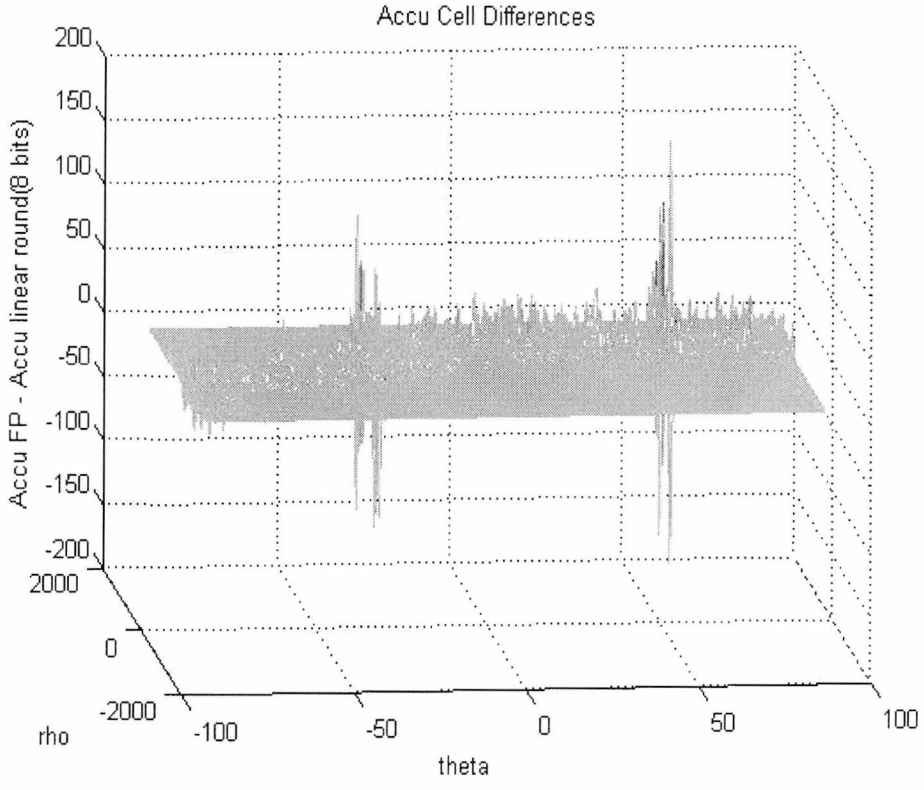
Figure 7.2-5: Hough Transform Parameter Space Output Graphs on 1024 x 1024 Binarised Images Using Linear 12-bit Arithmetic Precision.

Figures 7.2-3(a), and 7.2-3(b) shows the accumulator distribution of the linear Hough output when floating point arithmetic was used on the test images. The high peaks which correspond to the higher value of the accumulator cells can also be seen. Figures 7.2-4(a), and 7.2-4(b) shows the accumulator distribution when 8-bits of precision was used. It can be seen that even with such low precision, the same peaks have been identified with small variation on the peak values. Finally with the last set of figures, 7.2-5(a), and 7.2-5(b), where 12-bits of precision are used, the output results are almost identical with the floating point ones. Another method for supporting the above statements is to take the difference between the floating point arithmetic and the two fixed point solutions.



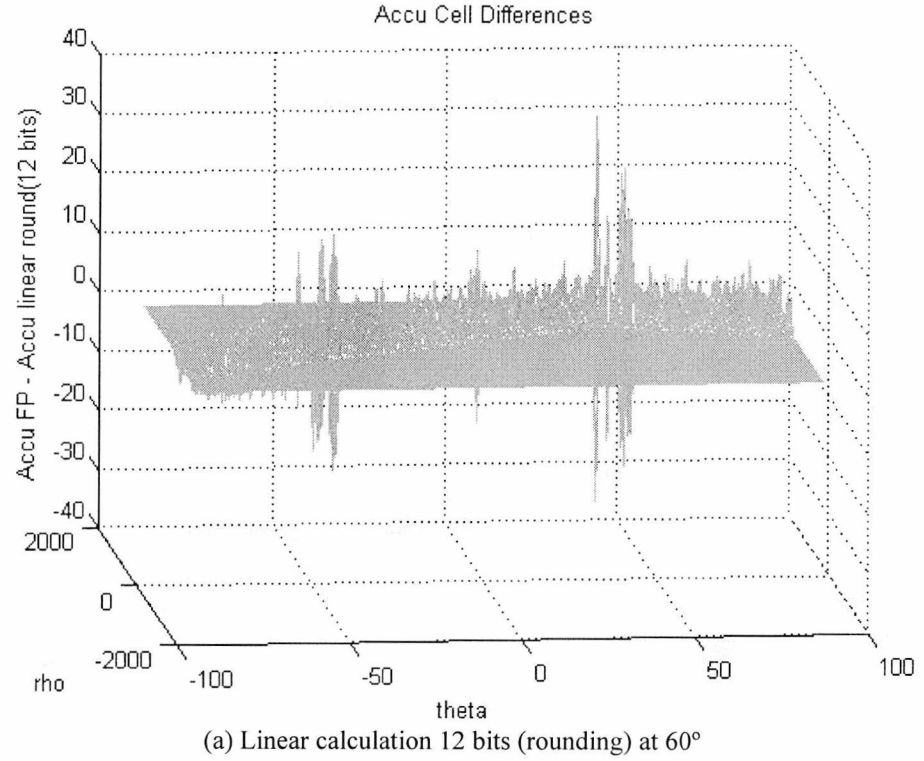
(a) Linear calculation 8 bits (rounding) at 60°

Figure 7.2-6: Hough Transform Parameter Space Difference Graphs on 1024 x 1024 Binarised Images Between Floating Point and Linear 8-bit Arithmetic Precision.



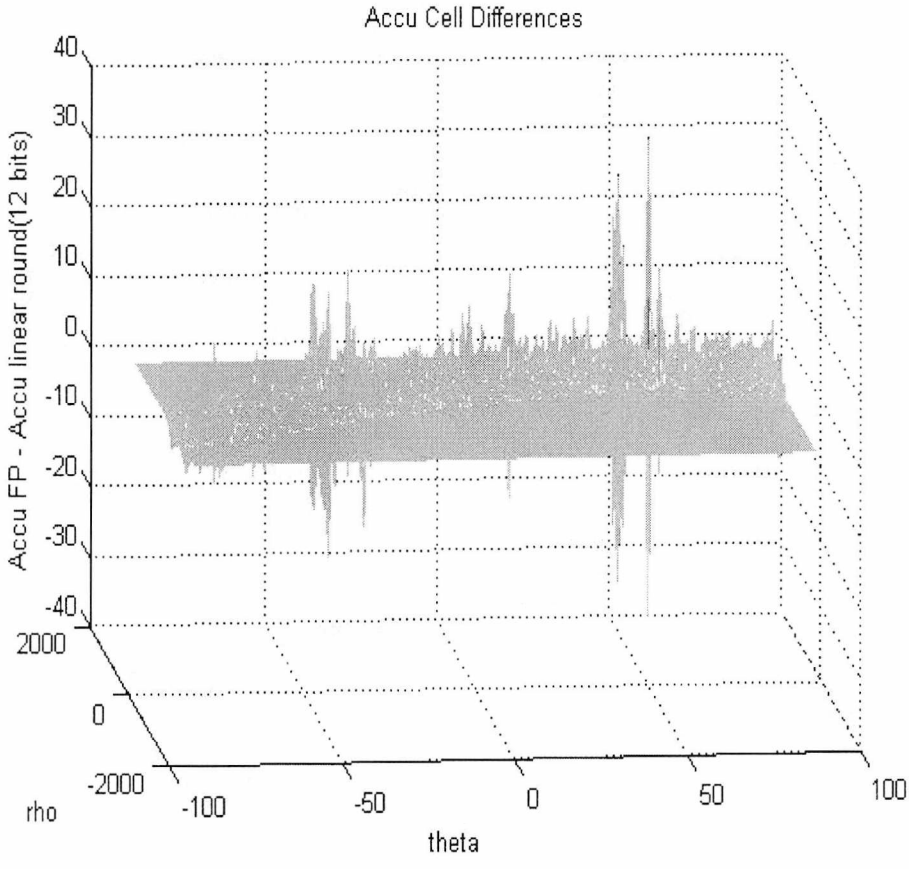
(b) Linear calculation 8 bits (rounding) at 45°

Figure 7.2-6: Hough Transform Parameter Space Difference Graphs on 1024 x 1024 Binarised Images Between Floating Point and Linear 8-bit Arithmetic Precision.



(a) Linear calculation 12 bits (rounding) at 60°

Figure 7.2-7: Hough Transform Parameter Space Difference Graphs on 1024 x 1024 Binarised Images Between Floating Point and Linear 12-bit Arithmetic Precision.



(b) Linear calculation 12 bits (rounding) at 45°

Figure 7.2-7: Hough Transform Parameter Space Difference Graphs on 1024 x 1024 Binarised Images Between Floating Point and Linear 12-bit Arithmetic Precision.

Figure 7.2-6(a) and 7.2-6(b) shows the difference maps between the Hough output where the floating point arithmetic, and the 8-bits of precision was used for the 60 and 45 degrees test images respectively. For the 60 degree image the highest peak difference is between ± 280, where for the 45 degrees image is ± 125. Those values as it can be seen from figures 7.2-7(a) and 7.2-7(b) are dropped significantly to ± 28 and ± 29 respectively, when 12-bits of precision were used. As previously, the maps show that even at relatively low precision the peaks are similar to those produced when using floating point arithmetic.

7.3 The Logarithmic Hough Transform

An alternative structure, which may result in a better utilization of FPGA resources, is based on the use of the log arithmetic instead of normal binary

arithmetic. Hybrid-LNS or hybrid-log arithmetic is an alternative solution where multiplication is performed in the log domain and addition performed in linear domain. The contents of the LUT are the pre-calculated (offline) logarithmic equivalents for sine and cosine respectively, rounded to fit. However, in the logarithmic HT method additional LUTs are used to translate between the log and linear domains [14].

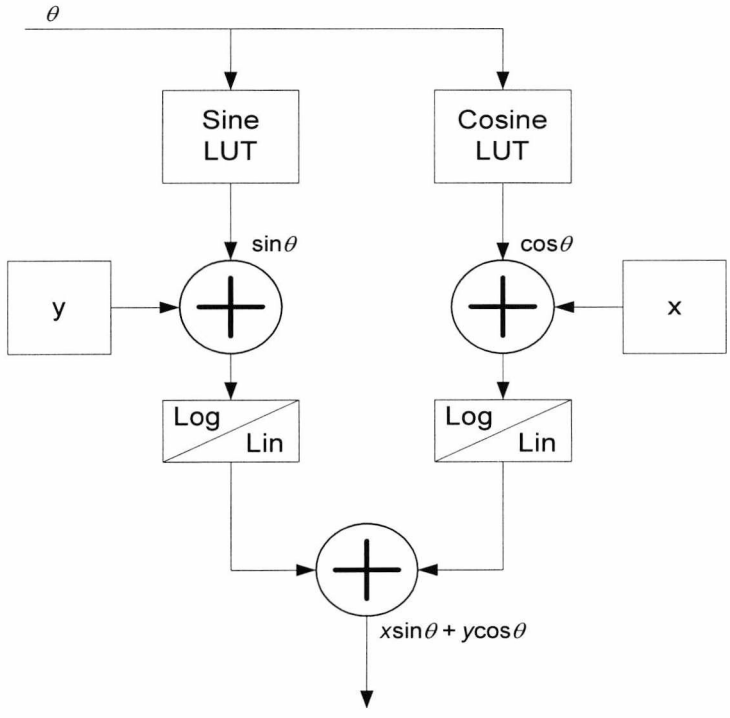
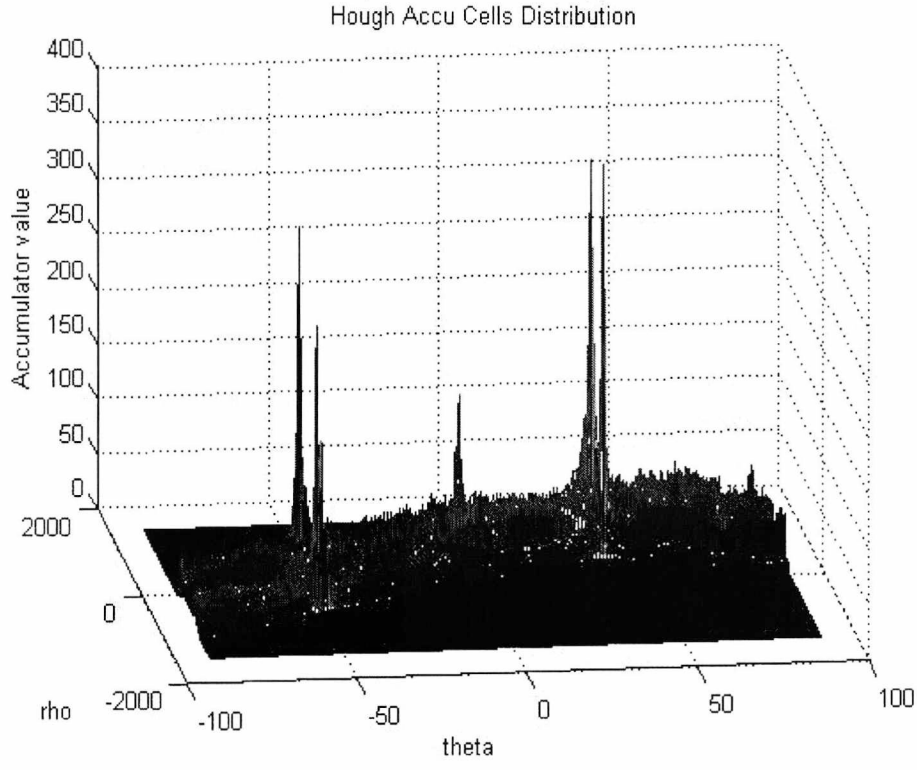


Figure 7.3-1: Hybrid-LNS HT Element

Figure 7.3-1 shows the proposed architecture of the hybrid-log HT calculation unit. It can be seen that multiplication has been replaced by addition. As with the original linear solution the sine and cosine components are stored in LUTs. The difference is that it is the logarithm of the sine and cosine components that are stored. These have been calculated offline so there is no need for a converter. The basic architecture of the log-to-linear converter is show in figure 6.2-2 in Chapter 6 Section 6.2.2. To ensure that the logarithm can represent a number arbitrarily close to zero the data can be scaled by a multiplication factor of 2^n where n is an integer. The pixel location as defined by x and y needs a Lin2Log conversion, but this will be common to all HT units and hence has not been included in this diagram.

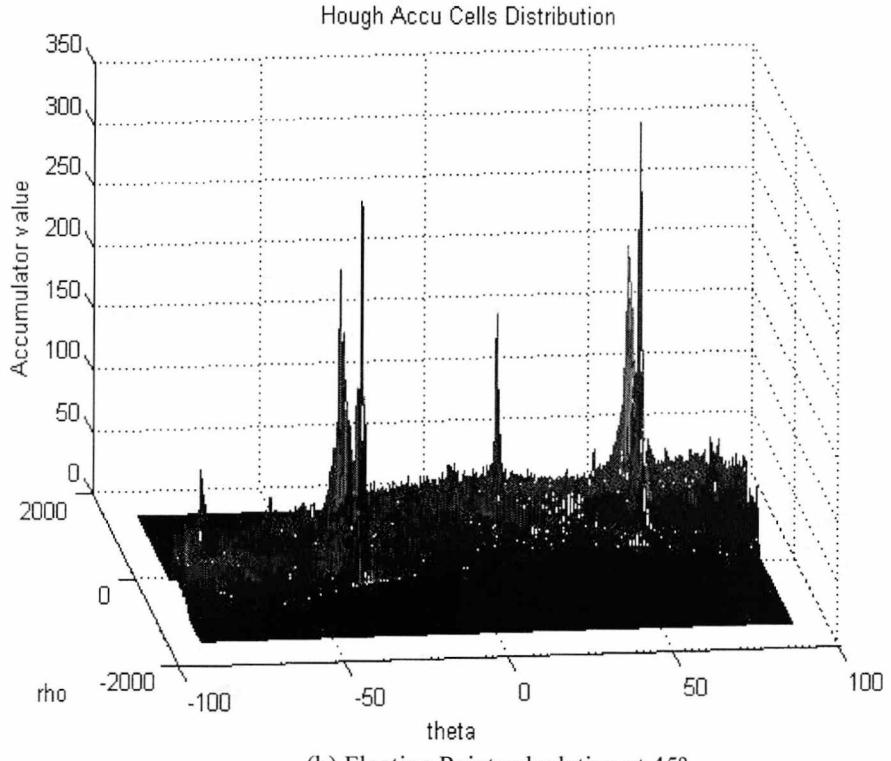
7.3.1 Proposed Logarithmic Implementation Using Matlab®

Matlab® experiments using hybrid-log architecture for the HT processing unit have been performed on the test images shown in Figure 7.2-2 with software routines defined in [162]. The parameter space was calculated using Matlab® for floating point precision and again using 8-bit and 12-bit fixed point arithmetic for the values of the sine and cosine stored in the LUTs, as well as, for the values in the logarithm-to-linear converters. The HT maps for floating point, 8 and 12 bits of fractional precision are shown in Figures 7.3-2, 7.3-3, and 7.3-4.



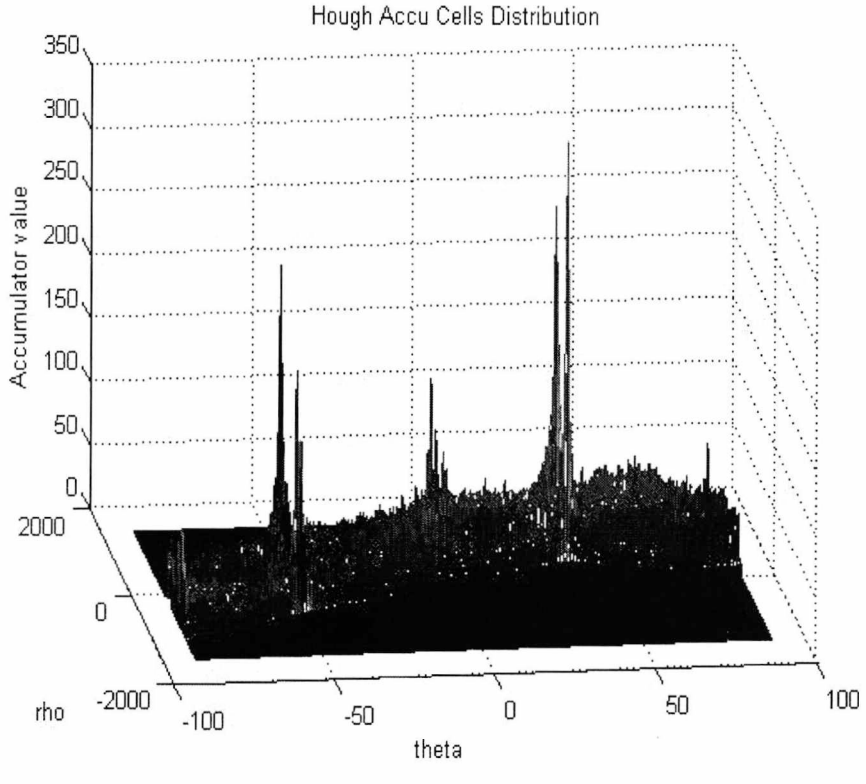
(a) Floating Point calculation at 60°

Figure 7.3-2: Hough Transform Parameter Space Output Graphs on 1024 x 1024 Binarised Images Using Floating Point Arithmetic Precision.



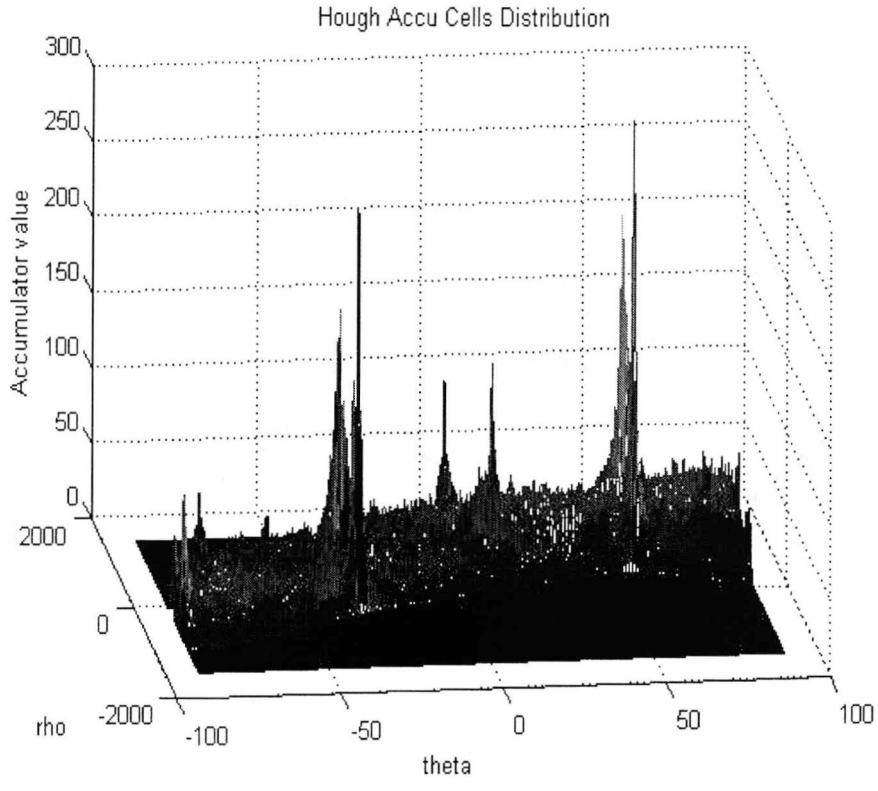
(b) Floating Point calculation at 45°

Figure 7.3-2: Hough Transform Parameter Space Output Graphs on 1024 x 1024 Binarised Images Using Floating Point Arithmetic Precision.



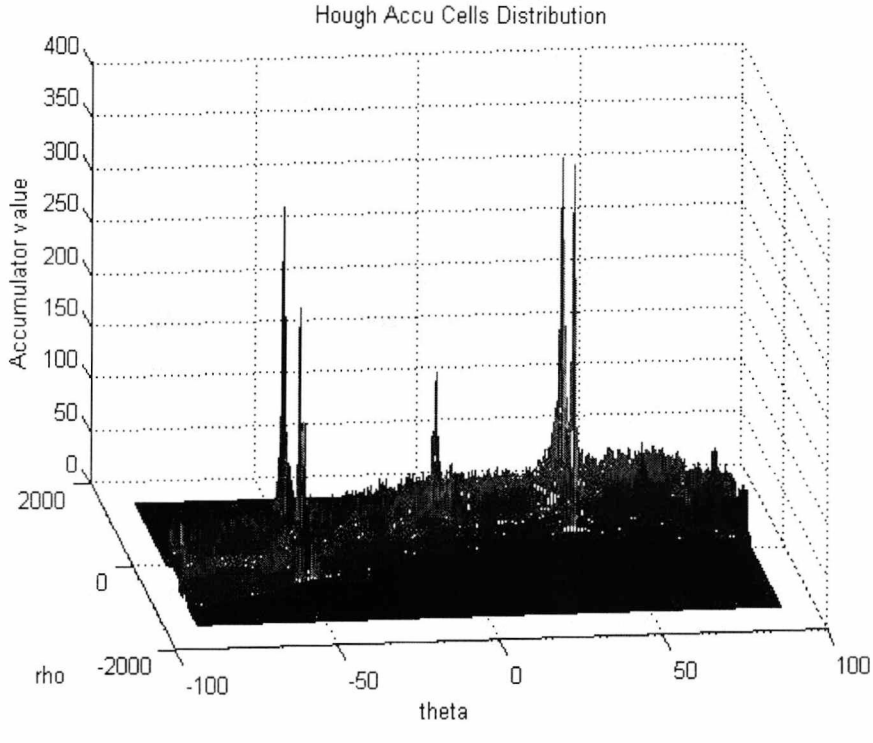
(a) Hybrid-log calculation 8 bits (rounding) at 60°

Figure 7.3-3: Hough Transform Parameter Space Output Graphs on 1024 x 1024 Binarised Images Using Hybrid-Log 8-bit Arithmetic Precision.



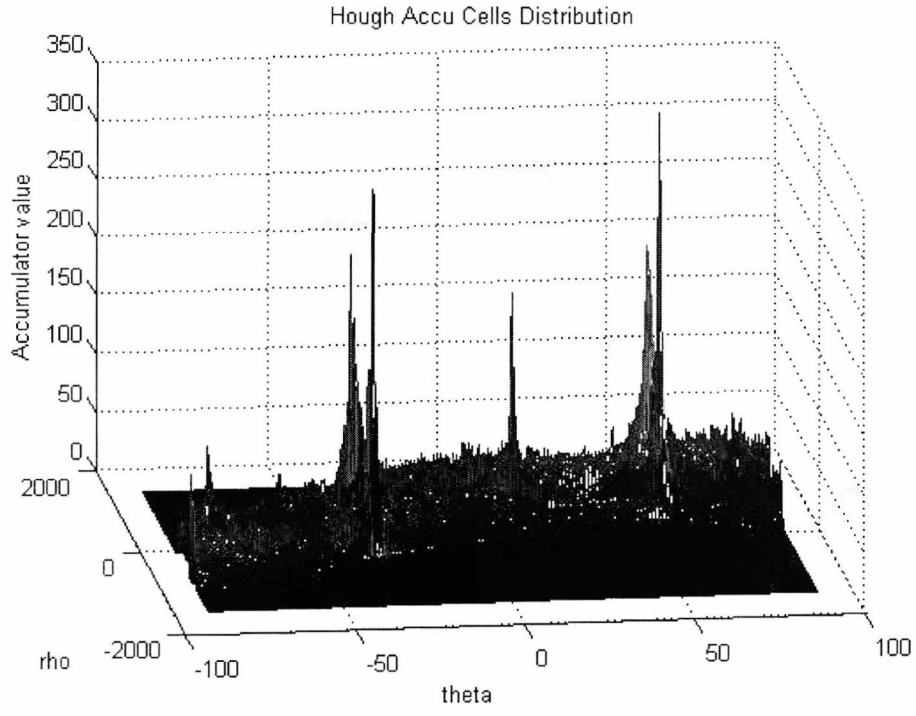
(b) Hybrid-log calculation 8 bits (rounding) at 45°

Figure 7.3-3: Hough Transform Parameter Space Output Graphs on 1024 x 1024 Binarised Images Using Hybrid-Log 8-bit Arithmetic Precision.



(a) Hybrid-log calculation 12 bits (rounding) at 60°

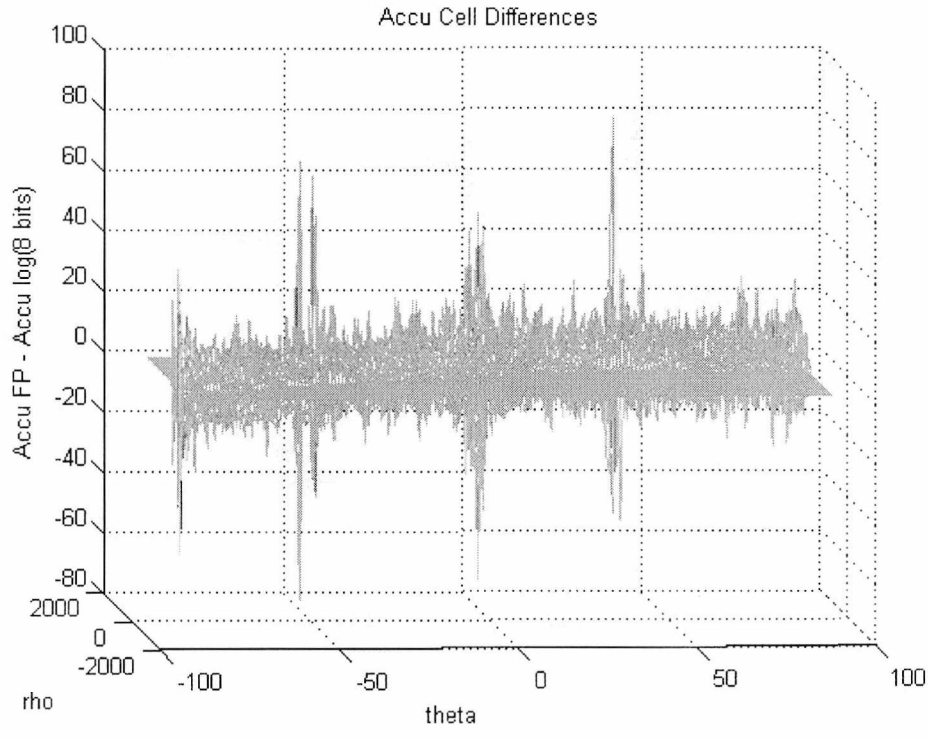
Figure 7.3-4: Hough Transform Parameter Space Output graphs on 1024 x 1024 Binarised Images Using Hybrid-Log 12-bit Arithmetic Precision.



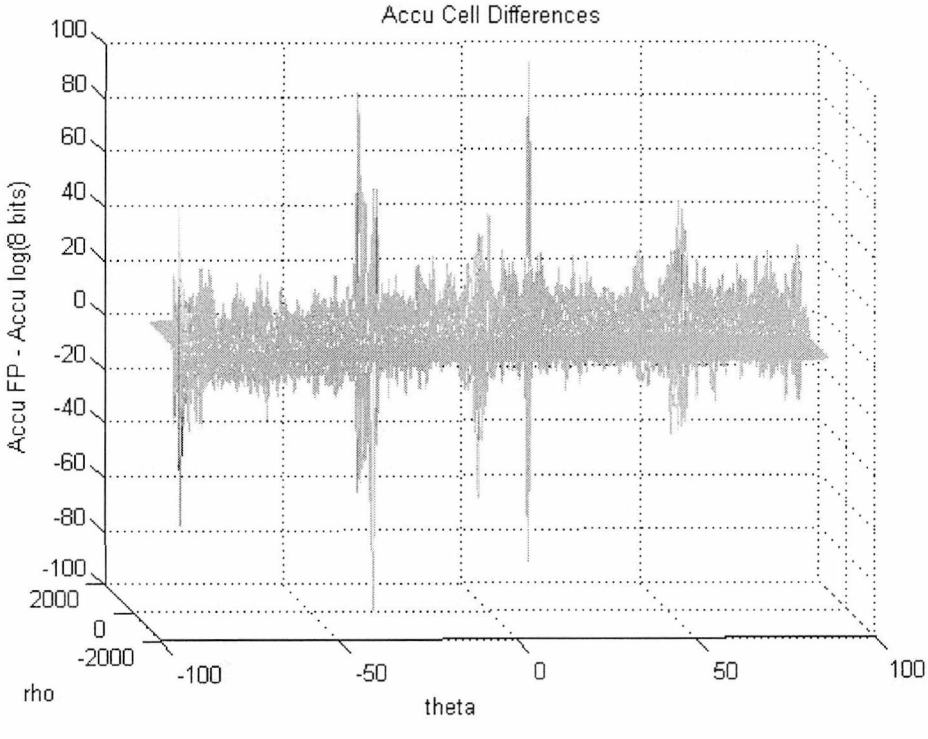
(b) Hybrid-log, 12 bits (rounding) at 45°

Figure 7.3-4: Hough Transform Parameter Space Output graphs on 1024 x 1024 Binarised Images Using Hybrid-Log 12-bit Arithmetic Precision.

Figure 7.3-2(a), and 7.3-2(b) shows the logarithmic Hough output using floating point arithmetic for both images. The high peaks which correspond to the higher value of the accumulator cells can also be seen. By comparing the results from figure 7.3-2 with the results in figure 7.3-3 it can be seen that, even when 8- bits of arithmetic precision is used, the same peaks are identified with a small variation on the accumulator values. More accurate results compared with those in figure 7.3-2 can be seen in figure 7.3-4, where 12-bits of arithmetic precision are used. Once more, the difference between the floating point arithmetic and the two fixed point solutions was taken and the results can be seen in figures 7.3-5 and 7.3-6.

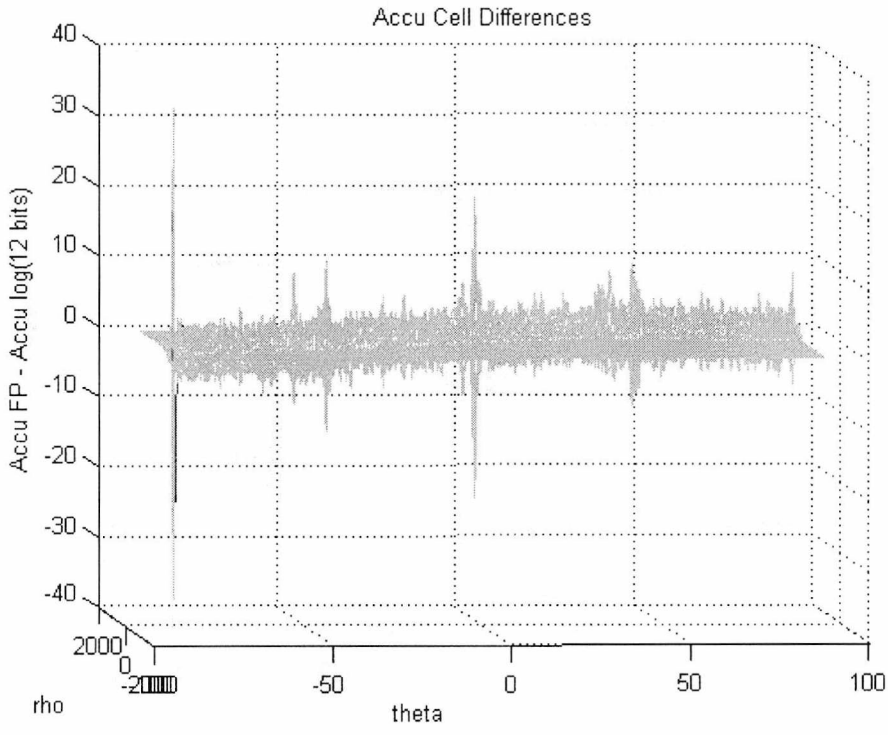


a) Hybrid-log calculation 8 bits (rounding) at 60°

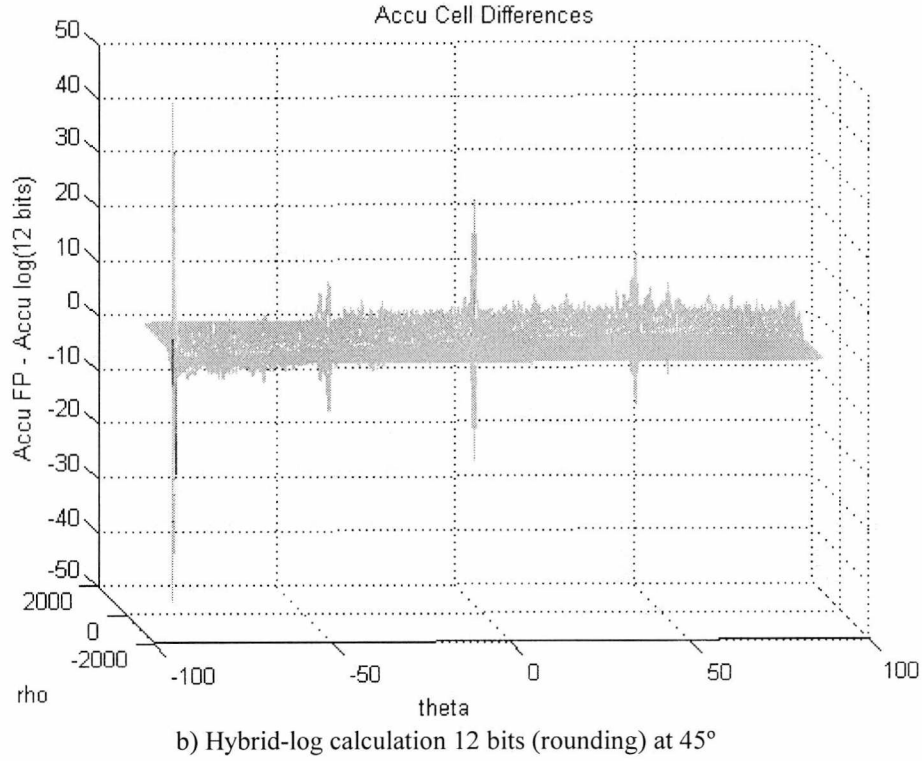


b) Hybrid-log calculation 8 bits (rounding) at 45°

Figure 7.3-5: Hough Transform parameter Space Difference Graphs on 1024 x 1024 Binarised Images Between Floating Point and Hybrid-Log 8-bit Arithmetic Precision.



a) Hybrid-log calculation 12 bits (rounding) at 60°



b) Hybrid-log calculation 12 bits (rounding) at 45°

Figure 7.3-6: Hough Transform Parameter Space Difference Graphs on 1024 x 1024 Binarised Images Between Floating Point and Hybrid-Log 12-bit Arithmetic Precision.

Figure 7.3-5(a) and 7.3-5(b) shows the difference maps between the Hough output where the floating point arithmetic, and the 8-bits of precision was used for the 60 and 45 degrees test images respectively. For the 60 degree image the highest peak difference is between ± 80, where for the 45 degrees image is ± 90. Figures 7.3-6(a), and 7.3-6(b) show that the values has been dropped significantly to ± 18 and ± 25 respectively, when 12-bits of precision were used (The error of the log plots at -90 degrees as shown in these plots is spurious). Comparison between figures 7.2-6, 7.2-7 and 7.3-5, 7.3-6, shows that the hybrid-log architecture outperforms the linear equivalent at similar levels of fractional precision, as it generates fewer errors than the equivalent linear one.

7.4 Hardware Implementation

Matlab® results have shown that hybrid-log architecture outperforms the linear one. The next step was the implementation of both architectures in hardware for the calculation of the required hardware resources. For the hybrid-log implementation, apart from the LUTs where the pre-calculated (offline) logarithmic equivalents for sine and cosine are stored, additional LUTs, required for the conversion between the logarithmic and the linear domain.

The size of the LUTs needed to convert between the log and linear domains grow exponentially with the number of bits of resolution and becomes prohibitively large when more than 16 bits of accuracy are required [164]. At 16 bits the overall size of the LUTs has been reduced by using solutions based on piecewise linear, Taylor or polynomial approximation [165]. This reduces the size of the LUT at the cost of additional multiplier(s) of small dimension. At resolutions of 12 bits and below there are a wide range of solutions with reduced memory requirements which are completely multiplierless [166], [167]. For image processing, where the image data is limited to 8-10 bits of resolution, the LUT requirements are acceptable and both LNS and Hybrid-LNS arithmetic have been shown to be suitable low-complexity and low-power alternatives to fixed-point processing of gray scale images. In the case of the HT an equal number of

additions and multiplications are required so, in principle at least, both LNS and hybrid log appear possible alternatives to using fixed point arithmetic.

The major elements used in both the linear and hybrid log solutions shown in Figures 7.2-1 and 7.3-1, were generated using the Xilinx CoreGenerator® toolset with maximum pipelining. Both solutions have been implemented and synthesized using Xilinx ISE 9.2 and the results are shown in Table 7.4-1 [14].

Table 7.4-1: Implementation Statistics of HT Elements

	Number of CLBs / slices			
Implementations	8 addresses per LUT 23 LUTs in total	16 addresses per LUT 12 LUTs in total	32 addresses per LUT 6 LUTs in total	64 addresses per LUT 3 LUTs in total
Linear 8-bits	257 / 1028	134 / 536	67 / 268	34 / 136
Linear 12-bits	**564 / 2256**	**294 / 1176**	**147 / 588**	**74 / 296**
Log 8-bits	**315 / 1260**	**164 / 656**	**82 / 328**	**41 / 164**
Log 12-bits	602 / 2408	314 / 1256	157 / 628	79 / 316

The number of CLBs as well as the number of respectively slices (one CLB = four slices) required for all the above implementations can be seen in Table 7.4-1, where the memory elements are available in multiples of 8 (3-bits), 16 (4-bits), 32 (5-bits) and 64 (6-bits) respectively. As the performance of the hybrid-log with 8 bits of fractional precision is equivalent to the linear solution with 12-bit resolution it can be seen that there is an advantage to be obtained by using the hybrid-log solution. For real-time implementation with the megapixel camera 64 HT elements running at approximately 74 MHz are necessary to achieve a performance of 25 frames per second.

7.5 Conclusion

This chapter has shown that it is possible to use multiplierless architectures based on hybrid-log arithmetic to implement the main processing elements necessary to perform the HT. It has been found that below 8-bits of precision the results were

unsatisfactory, as information of the images was lost and above 12-bits of precision the results were the same as the floating point ones.

A linear and a logarithmic implementation using Matlab® with 8-bits, 12-bits and floating point arithmetic was presented. The high peaks which correspond to the higher value of the accumulator cells was found and compared. Even with 8-bits of resolution the same peaks have been identified compared to the floating ones, with small variation on the peak values. The difference between the floating point and the two fixed point solutions was taken for the two test images and it shows for the linear implementation and the 60 degree image that the highest peak difference is between ± 280, where for the 45 degrees image is ± 125. Those values are dropped significantly to ± 28 and ± 29 respectively, when 12-bits of precision were used. For the logarithmic implementation it can be seen that for the 60 degree image the highest peak difference is between ± 80, where for the 45 degrees image is ± 90. Those values have been dropped significantly to ± 18 and ± 25 respectively, when 12-bits of precision were used.

Matlab® results have shown that hybrid-log architecture outperforms the linear one. The implementation of both architectures in hardware for the calculation of the required hardware resources was presented and it shows that the performance of the hybrid-log with 8 bits of fractional precision is equivalent to the linear solution with 12-bit resolution. Consequently, there is an advantage to be obtained by using the hybrid-log solution.

The relative simplicity of these structures means that it is feasible to implement multiple elements operating in parallel using just the basic CLB elements available on a typical FPGA fabric and leaving the DSP slices and Block RAM free for other functions in the image processing chain. Depending on the overall throughput it is possible to process data from a 1024 x 1024 pixel camera at a rate of up to 25 frames per second. However, as indicated earlier in the thesis this is only the first problem associated with implementing the HT on an FPGA fabric. Another limitation is the size of the memory array needed, which will be described in the next chapter.

CHAPTER EIGHT

DESIGN OF A LUT BASED ACCUMULATOR CELL

8.1 Introduction

This chapter presents a simple and compact architecture for building the accumulator cells needed in a HT processor to store "hits" when binarised image data, defined by their x and y co-ordinates, are translated into a Parameter Space Function (PSF) prior to extracting geometric primitives from the image. Also, the complete system is presented in terms of hardware requirements. The accumulator cells are implemented using Look-Up Table (LUT) resources available on most modern FPGA fabrics and are configured to allow high-speed real-time processing of the image. The modular architecture can easily be expanded to accommodate images with different resolutions. It is particularly suited to architectures such as the Xilinx Virtex 4 and Virtex 5 which has increased the capacity of the LUT cells available in the fabric as well as increasing the number of them that can be found on a single device making this architecture a suitable candidate for performing the HT in real-time on mega-pixel images.

8.2 Parametric Description of a Straight Line

As already it has been shown in Chapter 2, Section 2.2, the original method used to calculate the lines in the HT represented every possible line using the slope-intercept equation as shown in 2.2:1.

However, this approach is impractical due to the unlimited ranges of a and b when used to describe vertical or near-vertical lines. This approach has therefore been replaced by the normal or (ρ,θ) form as shown in equation 2.2:3 and

Figure 2.2-2 in Chapter 2. Now the set of lines passing through each point P_i in an image is represented by a set of sine curves in (ρ,θ) parameter space.

Although elegant in concept the HT is beset by a number of practical problems when applied to real-time image processing. Two of the main problems are the computational overhead incurred by the necessity to calculate all possible values of ρ and θ for each non-zero point in a binary image and the memory required to store the results of accumulating all possible points in (ρ,θ) parameter space before post processing them to find the most significant lines in the image.[5] Both of these problems become increasingly acute when dealing with increasing image resolutions now available with mega-pixel cameras operating at increasing frame rates (> 100 frames per second). Although the use of high speed parallel processing elements has helped with the first problem, the problem of the parameter space memory remains a significant one, especially when the signal processing is being performed on an FPGA which even at current levels of integration is limited for such applications.

8.3 Accumulator Cell

Each processing element (PE) is used to calculate ρ using a disjoint subset of θ in parallel. The resulting 2-tuple in parameter space (ρ,θ) represents a unique address in (ρ,θ) parameter space. Hence the accumulation process can also proceed in parallel using a unique accumulation element (AE) associated with each PE.

In Figure 8.3-1, a basic block diagram of the accumulator cell can be seen. The size of the accumulator is determined by the maximum length of a line that can be found in the image of interest. Using the Pulnix AccuPixel camera the resolution is 1024 x 1024 pixels and the longest line in the image is

$$\sqrt{1024^2 \times 1024^2} = 1448\,pixels \qquad \textbf{(8.3:1)}$$

For such an image an accumulator cell would require 11 bits of precision to ensure that overflow cannot occur during accumulation.

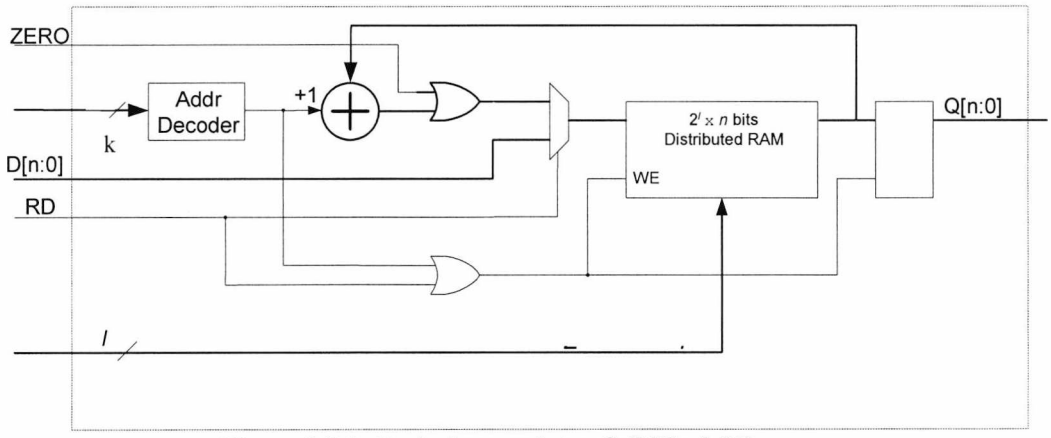


Figure 8.3-1: Basic Accumulator Cell Block Diagram

Each accumulator block is a subset of θ and ρ and it is addressed using n bits whereby k bits are used to decode the accumulator block and l bits are used to address the accumulator cell within this block. For a Xilinx Spartan3 or Virtex4 device this l is 4 bits because the size of the LUT in a slice is 16x1 bits.[141] For Xilinx Virtex5 [142] (and the new Virtex6) devices, the size of the LUT has been expanded to 64x1, making this architecture more efficient.

Each time a valid address to the accumulator cell is detected the incrementer is set to +1 and the accumulator addressed is incremented by one bit. Once the entire image has been scanned the data in the accumulator cells need to be passed on to a post processor which is used to extract any geometric features present in the original image. Because of the architecture in the FPGA there are several different methods for doing this, such as reading the data one block at a time through a decoder. Each method has an impact on the amount of time available before the next frame of the image can be processed.

8.4 An Alternative Accumulator Cell

A disadvantage of the cell shown in Figure 8.3-1 is that the contents of all the accumulators need to be read before the next frame of calculations can begin. If this is done sequentially this takes a significant amount of time which impacts on

the overall throughput of the algorithm. The cell shown in Figure 8.4-1 shows an alternative architecture which exploits the architectural attributes of the LUT structures implemented on Xilinx devices which can also be configured as variable length shift-registers. Each cell is read sequentially and stored in a separate SR. This is done in each block, in parallel. Once the data is stored in the SR is can be downloaded and processed while the next frame is being calculated.

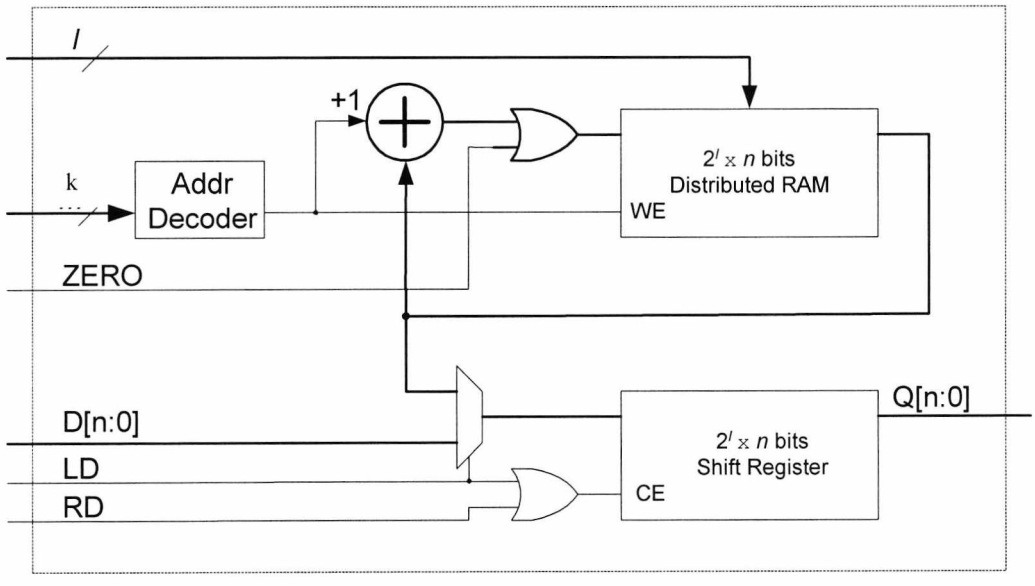


Figure 8.4-1: Alternative Accumulator Cell Block Diagram

8.5 Simulation Results

The accumulator cells described above was modelled using Modelsim implemented on a Xilinx Virtex 4 architecture using Xilinx ISE 9.2. For the simulation the address space was 16 bits. The simple architecture shown in Figure 8.3-1 used 16 slices or 4 CLBs and synthesis results indicate that it can operate at a maximum frequency of 77 MHz. The architecture in Figure 8.4-1 used 25 slices or 7 CLBs and operated at a maximum clock frequency of 77 MHz as well.

For the architecture in Figure 8.4-1 and according to Matlab® software routines defined in [148], the following number for the accumulator cells required for 180 degrees with a step of one degree per pixel (drho) in a 1024x1024 image:

For rho axis:

the maximum length of the image is: $D = \sqrt{1024^2 + 1024^2} = 1448$

the quantization steps are : $Q = \frac{D}{drho} = \frac{1448}{1} = 1448$

therefore, the number of steps where the rho axis will be divided is:

$$nrho = 2 * Q = 2896$$

For theta axis:

With the assumption of one degree per pixel, the theta axis will be divided in 180 steps.

This has a result of a total

$$2896 * 180 = 521280 \; cells$$

As each accumulator cell requires to be at least 11 bits wide and the depth of each cell is 16 bits, in terms of memory usage the amount of memory required is:

$$\frac{521280}{16} * 11 = 358.38 \; Kbits \; or \; 44.77 \; Kbytes$$

In terms of number of CLBs required, if each accumulator cell requires 25 slices and stores 16 addresses then:

$$\frac{521280}{16} * 25 = 814500 \; slices \; or \; 203625 \; CLBs$$

The above calculations were made with the assumptions that a Virtex-4 FPGA slice is used, which is based on 4-input LUT's (16 bits). The Virtex-5 and Virtex-6 FPGA slices are based on a 6-input LUT's (64 bits), where less memory (81.45 Kbits or 10.18 Kbytes) and hardware resources are required (203625 slices or 50907 CLBs).

8.6 The Complete System

After presenting the LUT based accumulator cell, which is the last individual stage of the overall system in this thesis, it is worth looking at all the stages described so far in terms of occupancy of hardware resources in an FPGA device, and more specifically in a Virtex-4 FPGA.

As it can be seen from Figure 8.6-1, the overall process of the HT implementation with the hardware requirements necessity can be categorized in three stages.

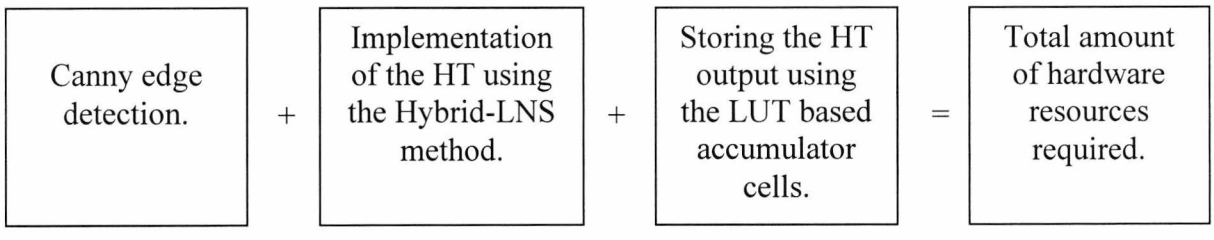


Figure 8.6-1: Stages of the HT Implementation

A summary of all the stages and the number of hardware requirements is shown in Table 8.6-1. When implementing the Canny edge detection on a Virtex-4 device, only 5.23% of the device is used. Similarly, the implementation of the HT using the Hybrid-LNS method can require from 0.18% (using 3 LUT with 8-bits of precision) up to 2.7% (using 23 LUT with 12-bits of precision) of the device. Finally, as it can be seen from the calculations above, implementing the accumulator cells is impossible in a Virtex-4 device, Virtex-5 devise or even in a Virtex-6 using only the distributed RAM. The required slices for the accumulator cell implementation are 814500 for a Virtex-4 and 203625 for a Virtex-5 device and Virtex-6 devise, where the maximum number of slices in a Virtex-4, Virtex-5 and Virtex6 are 89088, 51840 and 118560 respectively [141], [168], [169].

The absolute worse case scenario is considered for the above calculations, as a full frame is processed and there are accumulator cells for every pixel in that frame. If a full frame is required, then the internal Block RAM needs to be considered. The maximum block RAM in a Virtex-4 device is approximately 10Mbytes, where for a Virtex-5 device is approximately 18Mbytes and for a Virtex-6 devise is approximately 38Mbytes. [168] Otherwise, *coarse to fine*

techniques or time multiplexing techniques for reducing the number of accumulator cells are necessary. Such techniques, involve examination of the accumulator array at various scales and an attempt is made to evaluate it in detail only in those areas having a high density of counts. Some of such techniques have been described in literature review chapter (Chapter 2) and can be combined with the described implementation, as all of them are applicable, in order for the combined system to be implemented faster and with less computational cost.

	CLBs/Slices								Multipliers
Canny edge detection	1204 / 4818								30 18x18
HT with Hubrid-LNS	8 bits	12 bits	8 bits	12 bits	8 bits	12 bits	8 bits	12 bits	
8 addresses per LUT 23 LUTs in total	315 / 1260	602 / 2408							0
16 addresses per LUT 12 LUTs in total			164 / 656	314 / 1256					0
32 addresses per LUT 6 LUTs in total					82 / 328	157 / 628			0
64 addresses per LUT 3 LUTs in total							41 / 164	79 / 316	0
Accumulator cells	203625 / 814500								0
Total Logic CLBs / Slices	205144 / 820623	205431 / 821771	204993 / 820019	205143 / 820619	204911 / 819691	204986 / 819991	204870 / 819527	204908 / 819679	30 18x18

Table 8.6-1: The Complete System

8.7 Conclusion

This chapter has presented a new LUT based accumulator cell that can be used as part of the HT architecture, for storing the calculated Parameter Space Function (PSF) derived from a binary image. The accumulator cell uses the distributed memory elements available on a Xilinx FPGA fabric to store the intermediate results prior to passing them on for post-processing and feature extraction. This architecture enables a parallel implementation of the HT and PSF accumulators to be implemented, thereby speeding up the processing of the image or enabling significantly larger (megapixel) images to be processed in real-time. Additional local LUT resources can also be easily configured as temporary local memory, which can be used to reduce the inter-frame latency of the system.

Also, the complete system, which consists of three stages, was presented in terms of hardware requirements and a comparison between the stages was also detailed. The results show that the implementation of the Canny edge detection, as well as the HT using the Hybrid-LNS system, uses only a small percentage (approximately 8%) of a Virtex-4 FPGA device, in comparison to the implementation of the accumulator cells, where different techniques need to be considered. If a full frame requires processing, the internal Block RAM can be used for implementing the required accumulator cells.

CHAPTER NINE

SUMMARY & CONCLUSION

9.1 Summary

This thesis has presented techniques using Hybrid-LNS arithmetic for building the HT algorithm on FPGA technology. This work included the analysis of the HT using Hybrid-logarithmic arithmetic, the implementation of the Canny edge detection method for the binarization of the image, as well as the design of a compact architecture for building the accumulator cells, when binarized image data are translated into a Parameter Space Function (PSF), prior to extracting geometric primitives from the image.

Chapter 1 introduced, the general concept of the HT algorithm, the necessity of the edge detection prior to the HT, and the implementation of the HT using the Hybrid-LNS arithmetic. As stated in this chapter, the HT focuses on image processing applications, with goals set in the areas of speed, size (in terms of logic) and accuracy.

The HT was described in detailed terms in Chapter 2, starting from the basic principles of the algorithm, along with how it works, the advantages and disadvantages of it, as well as, the different applications and methods. The process of calculating the HT was also shown, along with the possible parallel processing architectures for implementing it on the target technology.

Edge detection of an image is necessary prior to the HT algorithm calculation. The comparison between edge detection methods, as well as, the detailed description of the Canny method was presented in Chapter 3. A literature review of the digital logarithms is also presented in this chapter, and it was shown, through a review of the relevant literature, how the implementation of logarithms

and logarithmic arithmetic has developed over the past fifty years into a method appropriate for the implementation of DSP functions.

As a number of pre-processing steps are required for removing noise or to accentuate features in the image prior to the image extraction, a novel generic, synchronizing circuit architecture for windowing operations was implemented in Chapter 4. The Canny edge detection method was implemented in hardware using the latest FPGA technology, where fixed point arithmetic it was compared with solutions built using floating point arithmetic. The synchronizing circuit architecture with the implementation of the Canny edge detection method formed the two of the contributions of the thesis. Simulation results showed that a small fraction of the array logic available on most FPGA fabrics is used for the implementation of the Canny edge detection on FPGA, allowing other functions in the image processing chain to be processed.

All designs proposed in this thesis are aimed towards implementation on FPGA technology, as they are very versatile and powerful piece of hardware, capable of implementing large and complex logic-based designs and in a very high speed. A more detailed description of the FPGAs is outlined in Chapter 5. Also, the most common devices such as the digital signal processors (DSP) and the ASICs are compared with the FPGAs, and the most advanced arithmetic technology for FPGAs were described in detail.

An alternative method of implementing arithmetic on hardware using logarithms was described in Chapter 6. The most appropriate number system for arithmetic implementation on hardware was analysed and fully described, and was found to be a cross between logarithmic and linear arithmetic. A description was given of how multiplication is most efficiently implemented with logarithms, and addition with the linear number system. This is known as hybrid-logarithms number system.

Chapter 7, therefore, proposed the implementation of the system already described in Chapter 6, specifically applied to the HT algorithm. The chapter showed the benefits that implementing arithmetic on hardware can bring to the

transform. Such a method has not been investigated in any published literature and it forms the main novel investigation presented in this thesis. How this can be implemented in the HT algorithm, was demonstrated in Chapter 7, along with the potential advantages it has to offer over purely linear-based arithmetic implementations. Due to the relative simplicity of the structure described in Chapter 6, it is feasible to implement multiple such elements operating in parallel. It was shown that, by using logarithmic arithmetic, the need of multipliers is eliminated, while precision of the HT is maintained. Depending on the overall throughput it is possible to process data from a 1024 x 1024 pixel camera at a rate of up to 25 frames per second.

Computational cost is a main drawback of the HT algorithm, but is not the sole concern. The size of the memory array needed to store the results of accumulating all possible points in the PSF, before post processing them to find the most significant lines in the image, is another main problem of the HT algorithm. Chapter 8 presented a novel LUT based accumulator cell that can be used as part of the HT architecture for storing the calculated PSF. The aforementioned PSF is derived from a binary image. Assuming the worst absolute scenario, where there is an accumulator cell for every pixel in the image, it was shown that it is not possible to implement the accumulator cells in one of the latest FPGAs devices.

In the same chapter, a comparison between the implementation of the Canny edge detection, in conjunction with the HT main processing block using logarithmic arithmetic and the LUT based accumulator cell was made. It was found that, the hardware implementation on an FPGA of the Canny method with the HT main processing block, require only a small percentage of the overall hardware recourses. However, when implementing the accumulator cells for a 1024 x 1024 image, different techniques need to be considered.

Overall, this thesis has made several contributions to the state of the art that can be summarized as follows:

- An investigation into the effects and possible benefits that the use of Hybrid-Logarithmic arithmetic has on the HT algorithm. This showed

how the need for multipliers could be eliminated, while the precision of the algorithm is maintained.

- An optimal implementation of the HT proposed using the most advanced arithmetic hardware available on modern FPGAs. Despite not implementing multipliers, the algorithm performance was found to be almost identical to the software implementation.
- A flexible design of a generic synchronization circuit proposed for windowing operations in 2D imaging filters of variable dimensions, enabling parallel processing implementation on FPGA fabrics.
- A successful implementation of the Canny edge detection method on hardware, by the use of a novel synchronization circuit for the windowing operations, using fixed point arithmetic.
- A design of a LUT based accumulator cell for high speed HT algorithms on FPGA fabrics for storing the PSF.
- A combined implementation between the Canny edge detection, the main HT processing block and the LUT based accumulator cell on FPGA fabrics.

9.2 Future Work

This thesis has provided evidence of a comprehensive investigation into the implementation of the HT algorithm on FPGA technology. A novel technique for efficient implementation of the algorithm using Hybrid-LNS has been shown, along with the implementation of the algorithm on the latest FPGA technology. No doubt, as FPGA technology incorporates faster, more accurate, and more versatile integrated arithmetic components, new possibilities and techniques will become possible.

The effects of using hybrid-logarithmic-based arithmetic on the HT algorithm were investigated, in addition to a successful hardware implementation of the Canny edge detection method. Although the design of a LUT based accumulator cell for storing the calculated PSF was made, further investigation is required for reducing the number of the accumulator cells, such as splitting the image in

smaller sections rather than processing the full frame. Another solution could be the use of *coarse to fine* or time multiplexing techniques described in the literature review chapter (Chapter 2), as all of the aforementioned methods are applicable to the proposed HT implementation. In the case that a full frame is required to be processed, the Block RAM available in the FPGA devices needs to be considered for the implementation of the accumulator cells.

Another possible line of future research that can potentially offer further valuable information, but was out of the scope of this thesis, is the implementation of the HT using hybrid-logarithmic arithmetic in more generic shapes rather that straight lines. These shapes could be curves, circles or parabolas.

As camera technology is incorporated faster and operates at higher resolutions (up to 10 Mpixels), this research can be scaled to work with those higher resolutions. Although more computations, as well as memory will be required, the principles used will be the same as the ones proposed in this thesis.

REFERENCES

[1] P.V.C Hough, "Method and Means for Recognizing Complex Patterns", *U.S Patent 3069654*, Dec. 18, 1962.

[2] F.-s. Lai and C.-F.E. Wu, "A Hybrid Number System Processor with Geometric and Complex Arithmetic Capabilities". **IEEE Transactions on Computers** 1991, **40**(8): pp. 952-962.

[3] P. Lee. "An Evaluation of a Hybrid Logarithmic Number System DCT / IDCT Algorithm". in *Proceedings of the IEEE ISCAS*. 2005. Kobe, Japan.

[4] D. H. Balard, "Generalizing the Hough transform to detect arbitrary shapes", *Pattern Recognition*. **13**(2), 1981, pp.111-122.

[5] J.Illingworth and J. Kittler, "A survey of the Hough transform", *Computer Vision Graphics Image processing*, **44**, 1988, pp.87-111.

[6] R.O.Duda and P.E.Hart, "Use of the Hough transform to detect lines and curves in pictures", *Comm. Assoc. Comput. Mach.* **15**(1), 1972, pp.11-15.

[7] N.G. Kingsbury and P.J.W. Rayner, "Digital Filtering Using Logarithmic Arithmetic". **Electronics Letters**, 1971. **7**(2): pp. 56-58.

[8] Canny, J., *A,* "Computational Approach to Edge Detection", **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Vol. **8**, No. 6, November, 1986. pp. 679-698.

[9] S. Trimberger. "Redefining the FPGA for the Next Generation". in *International Conference on Field Programmable Logic and Applications, 2007*. 2007. Amsterdam, Netherlands. pp. 4.

[10] M. Wisdom, "Optimal wavelet-based image compression on FPGA technology," doctoral dissertation, Department of Electronics, University of Kent, Kent, United Kingdom, 2009.

[11] Camera manufactures [Available from: http://www.aegis-elec.com/products/gigabit_ethernet_cameras.html].

[12] S. Tagzout, K. Achour, and O. Djekoune, "Hough transform algorithm for FPGA implementation", *Signal Processing*, vol. **81**, no. 6, 2001, pp.1295-1301.

[13] R. Cucchiara, G. Neri and M. Piccardi, "A real-time implementation of the Hough transform", **Journal of System Architectures**, vol. **45**, 1998, pp. 31-45.

[14] P. Lee, E. Alexiadis, "An implementation of a multiplierless Hough transform on an FPGA platform using hybrid-log arithmetic", *SPIE Conference on Real-Time Image Processing 2008*, California, USA, 2008.

[15] P. Lee, E. Alexiadis, " A flexible LUT based accumulator cell for high speed Hough transforms on FPGA fabrics", to be submitted to IET Letters, January 2010.

[16] Rafael C. Gonzalez, Richard E. Woods. Digital Image Processing, Prentice Hall, 2nd edition, 2002.

[17] Majumdar A. K." Design of an ASIC for straight line detection in an image". *13th international conference on VLSI design*. January 2000.

[18] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins. Digital Image processing using MATLAB, Prentice Hall, 2004.

[19] A.Rosenfeld, "Picture Processing by computer", *ACM Computing Surveys(CSUR)*, Vol **1**(3), New York, 1969, pp.147-176.

[20] M. Cohen and G. T. Toussaint, "On the detection of structures in noisy pictures", *Pattern Recognition* **9**, 1977, pp.95-98.

[21] F.O'Gorman and M. B. Clowes, "Finding picture edges through collinearity of feature points", *IEEE T-COMP,* **25**, 1976, pp.449 – 456.

[22] T. M. Van Veen and F. C. A. Groen, "Discretization errors in the Hough transform", *Pattern Recognition* **14**, 1981, pp.137 – 145.

[23] S.R.Deans, "Hough transform from the radon transform", *IEEE Trans.Pattern Anal. Mach. Intel.*, **PAMI-3**, 1981, pp.185-188.

[24] V. F. Leavers, "Survey: Which Hough transform?", *CVGIP: Image understanding*.**58**(2), 1993, pp.250-264.

[25] H. Li, M. A. Lavin, R. J. Le Master, "Fast Hough transform: A hierarchical approach", **CVGIP**. **36**, 1986, pp. 139-161.

[26] J. Princen, H. K. Yuen, J. Illingworth, J. Kittler, " A comparison of Hough transform methods", in *Proc. IEEE 3rd Int. Conf. on Image Processing and its Applications*, 1989, pp.73-77.

[27] J.Illingworth and J. Kittler, "The adaptive Hough transform, *IEEE Pattern Anal. Mach. Intell.***PAMI-9**(5), 1987, pp. 690-698.

[28] X. Cao, M. G. Rodd, F. Deravi, and Q. M. Wu, "Detection of multiple cycles based on adaptive Hough transform", *Eng, Appl, Artif. Intell.* **1**(2), 1988, pp. 97-101.

[29] K. Onda, K. Sasao, and Y. Aoki, "A method for detecting skewed symmetry axis of objects, *J.Inst. TV Eng.* Japan **43**(9), 1989, pp. 982-989.

[30] A. D. Berger and K. Khosla, "Using tactile data for real-time feedback", *Int. J. Rob. Res.***10** (2), 1991, pp.88-102.

[31] D. D. Haule and A. S. Malowany, "Object recognition using fast adaptive Hough transform", *IEEE Pasific Rim Conf. on Communications, Computers and Signal Processing*, Cat. No. 89CH 2691-4, 1989, pp.91-94.

[32] L. da F. Costa and M. B. Sandler, "A binary Hough transform and its efficient implementation in a systolic array architecture", **Pattern Recognition Letters**, vol. **10**, no. 5, 1989, pp.329-334.

[33] L. da F. Costa and M. B. Sandler, "A complete and efficient time system for line segment based on the binary Hough transform", In *Proc. Euromicro '90 Workshop on Real Time*, 1990, pp. 205-213.

[34] L. da F. Costa, P. Tzionas, and M. B. Sandler, "On the VLSI implementation of the binary Hough transform", *IEE Colloquium Digest 1990/95*, 1990.

[35] H. Koshimizu and M. Numada, "FIHT2 Algorithm: A fast incremental Hough transform", *IEICE Trans.*, Japan, **74** (10), pp. 3389-3393.

[36] V. F. Leavers, D. Ben-Tzvi, and M. B. Sandler, "A dynamic combinatorial Hough transform for straight lines and circles", *Proc. 5th Alvey Vision Conf. London* (UK), 1989, pp.163-168.

[37] S. Y. K. Yuen, T. S. L. Lam, and N. K. D. Leung, "Connective Hough transform", *Image & Vision Computing* vol.**11,** no.5, June 1993, pp. 295-301.

[38] V. F. Leavers, "Active intelligent vision using the dynamic generalized Hough transform", *Proc. 1st British Machine Vision Conf.*, 1990, pp. 49-54.

[39] J. Princen, J. Illingworth, and J. Kittler, "A hierarchical approach to line extraction based on the Hough transform", *Computer Vision Graphics & Image Processing.* **52**, 1990, pp. 57-77.

[40] N. Kiryati, Y. Eldar, and A. M. Bruckstein, " A probabilistic Hough transform", *Pattern Recognition*, vol. **24**, no. 4, 1991, pp.303-316.

[41] M. K. Ibrahim, E. C. L. Ngau, and M. F. Daemi, "Weighted Hough transform", *J. Soc. Photo-Opt. Instrum. Eng.*, **1607**, 1992, pp. 237-241.

[42] K. Sugawara, "Weighted Hough transform on a gridded image plane", *Proceedings of the 4th International Conference on Document Analysis and Recognition*, 1997, pp. 701-704.

[43] M. Atiquzzaman, "Multiresolution Hough transform – An efficient method of detecting patterns in images", *IEEE transactions on pattern analysis and machine intelligence*, vol. **14**, no. 11, 1992, pp.1090-1095.

[44] L. Xu, E.Oja, and P. Kultanen, "A new curve detection method: Randomized Hough transform (RHT)", *Pattern Recognition Letters*, vol. **11**, 1990, pp.331-338.

[45] L. Xu, "Randomized Hough transform (RHT): Basic mechanisms, algorithms, and computational complexities", *CVGIP: Image understanding*, vol. **57**, no.2, 1993, pp.131-154.

[46] A. R. Hare and M. B. Sandler, "Improved performance randomized Hough transform", *Electronic Letters.* **28** (18), 1992, 1678-1679.

[47] R. M. Inigo, E. S. McVey, B. J. Berger, and M. J. Wirtz, " Machine vision applied to vehicle guidance", *IEEE T-PAMI*, vol. **6**, 1984, pp.820-826.

[48] C. R. Dyer, " Gauge inspection using Hough transforms", *IEEE T-PAMI*, vol. **5**, no.6, 1983, pp.621-623.

[49] K. Y. Huang, K. S. Fu, T. H. Sheen, and S. W. Cheng, "Image processing of seismograms: (A) Hough transformation for the detection of seismic patterns; (B) thinning processing in the seismogram", *Pattern Recognition*, vol. **18**, no. 6, 1985, pp. 429-440.

[50] M. Kushnir, K. Abe, and K. Matsumoto, " An application of the Hough transform to the recognition of printed Hebrew characters", *Pattern Recognition,* vol. **16**, no. 2, 1983, pp.183-191.

[51] W. C. Lin and R. Dubes, "A review of ridge counting in dermatoglyphics", *Pattern Recognition*, vol. **16**, no.1, 1983, pp.1 -8.

[52] T. Shibata, W. Frei, "Hough transform for target detection in infrared imagery", *SPIE* **281**, 1981, pp.105-109.

[53] A. E. Cowart, W. E. Snyder, and W. H. Ruedger, " The detection of unresolved targets using the Hough transform", *CVGIP* **21**, 1983, pp.222-238.

[54] J. Skingley and A. J. Rye, "The Hough transform applied to SAR images for thin line detection", *Pattern Recognition Letters*, vol. **6**, 1987, pp.61-67.

[55] D. B. Shu, C. C. Li, J. F. Mancuso, and Y. N. Sun, "A line extraction method for automated SEM inspection of VLSI resist", *IEEE T-PAMI* **10**, no.1, 1988, pp.117-120.

[56] M. Nixon, "Application of the Hough transform to correct for linear variation of background illumination in images", *Pattern Recognition Letters*, vol. **3**, 1985, pp.191-194.

[57] R. C. Thomson and E. Sokolowska, "Mineral cleavage analysis via the Hough transform", in Proceedings, *British Pattern Recognition Association 4th International Conf. on Pattern Recognition*, 1988, pp.390-398.

[58] S. N. Jayaramamurthy, and R. Jain, "An approach to the segmentation of textured dynamic scenes", *CVGIP* **21**, 1983, pp.239-261.

[59] G. Adiv, "Recovering motion parameters in scenes containing multiple moving objects", in *IEEE CVPR Conf.,* Washington, 1983, pp.399-400.

[60] H. Kalviainen, "Motion detection using the randomised Hough transform: exploiting gradient information and detecting multiple moving objects", *IEE Proceedings of Vision, Image and Signal Processing*, vol. **143**, no. 6, 1996, pp.361-369.

[61] H. Kalviainen, "Applications of the Hough transform for image processing and analysis", *Pattern Recognition and Image Analysis,* vol.**13**, no. 2, 2003, pp.187-190.

[62] C. J. Radford, "Optical flow fields in Hough transform space", *Pattern Recognition Letters*, vol. **4**, 1986, pp.293-303.

[63] T. M. Silberberg, L. Davis, D. Harwood, "An iterative Hough procedure for three-dimensional object recognition", *Pattern Recognition*, vol. **17**, No. 6, 1984, pp.621-629.

[64] T. C. Henderson, W. S. Fai, "The 3-D Hough shape transform", *Pattern Recognition Letters*, vol. **2**, 1984, pp.235-238.

[65] S. Kasif, L. Kitchen, and A. Rosenfeld, "A Hough transform technique for subgraph isomorphism", *Pattern Recognition Letters*, vol. **2**, 1983, pp.83-88.

[66] M. Mirmehdi, G. A. W. West, G. R. Dowling, "Label inspection using the Hough transform on transputer network", *Microprocessors and Microsystems*, **15** (3), 1991, pp.167-173.

[67] V. Kamat, and S. Ganesan, "An efficient implementation of the Hough transform for detecting vehicle license plates using DSP'S", *Proceedings of Real-Time Technology and Applications*, 1995, pp.58-59.

[68] M. G. He, A. L. Harvey, and T. Vinay, "Hough transform in car number plate skew detection", *Symposium on Signal Processing and its Applications, ISSPA*, vol. **2,** 1996, pp. 593-596.

[69] B. Yu, and A. K. Jain, "Lane boundary detection using a multiresolution Hough transform", *International Conference on Image Processing, ICIP*, vol. **2**, 1997, pp.748-751.

[70] R. Muñiz, L. Junco, and A. Otero, "A robust software barcode reader using the Hough transform", *Proceedings of Information Intelligence and Systems*, 1999, pp.313-319.

[71] Y. Sun and P. Willett, "The Hough transform for long chirp detection", *Proceedings of Conference on Decision and Control*, vol. **1**, 2001, pp.958-963.

[72] A. Tezmol, H. Sari-Sarraf, S. Mitra, R. Long, A. Gururajan, "Customized Hough transform for robust segmentation of cervical vertebrae from X-ray images", *Proceedings of Image Analysis and Interpretation,* 2002, pp.224-228.

[73] M. Chang, I. Kim, and J. Park, "Optical flow measurement based on boolean edge detection and Hough transform", *International Journal of Control, Automation, and Systems*, vol. **1**, no. 1, 2003, pp.119-126.

[74] M. Greenspan, L. Shang, and P. Jasiobedzki, "Efficient tracking with the bounded Hough transform", *CVPR*, vol. **1**, 2004, pp.520-527.

[75] C. Rosito and R. Schramm, "Rectangle detection based on a windowed Hough transform", *Proceedings of Computer Graphics and Image Processing,* 2004, pp.113-120.

[76] F. Rovira-Mas, Q. Zhang, J. F. Reid, and J. D. Will, "Hough-transform-based vision algorithm for crop row detection of an automated agricultural vehicle", *Proceedings of the Institution of Mechanical Engineers*, Vol. **219,** Issue 8, 2005, pp.999-1010.

[77] C. H. Messom, G. S. Gupta, and S. N. Demidenko, "Hough transform run length encoding for real-time image processing", *IEEE transactions on Instrumentation and Measurement*, vol.**56**, no. 3, 2007, pp.962-967.

[78] Y. Tsai, L. Jian, P. Hsu, and B. Wang, "Implementation of autonomous vehicles with the Hough transform and fuzzy control", *SICE,* 2007, pp.2095-2101.

[79] E. Bernabeu and J. Tornero, "Hough transform for distance computation and collision avoidance", *IEEE transactions on Robotics and Automation,* vol.**18**, no. 3, 2002, pp.393-398.

[80] SIMD block diagram. (2009, Mar) [Available from: http://en.wikipedia.org/wiki/SIMD].

[81] MIMD block diagram. (2009, Mar) [Available from: http://en.wikipedia.org/wiki/MIMD].

[82] M. Feretti and M. G. Albanesi, "Architectures for the Hough transform: A survey", *IAPR Workshop on Machine Vision Applications*, 1996, pp.542-551.

[83] A. L. Fisher and P. T. Highnam, "Computing the Hough transform on a scan line array processor", *IEEE trans. on Pattern Analysis and Machine Intelligence*, vol. **11**, no.3, 1989, pp.262-265.

[84] Z. N. Li, F. Tong, and R. G. Laughlin, "Parallel algorithms for line detection on a 1xN array processor", *Proceedings of the IEEE Inter. Conf. on Robotics and Automation*, 1991, pp.2312-2318.

[85] H. M. Alnuweiri and V. K. P. Kumar, "Optimal image algorithms on an orthogonally-connected memory-based architecture", in *10th Int. Conf. on Pattern Recognition*, 1990.

[86] A. Rosenfeld, J. Ornelas. Jr, and Y. Hung, "Hough transform algorithms for mesh-connected SIMD parallel processors", *Computer Vision, and Image Processing,* **41**, 1988, pp.293-305.

[87] C. S. Kannan and H. Y. H. Chuang, "fast Hough transform on a mesh connected processor array", Information Processing Letters, vol.**33**, 1990, pp.243-248.

[88] S. Olariu, J. L. Schwing, and J. Zhang, "Computing the Hough transform on reconfigurable meshes", *Image and Vision Computing*, vol. **11**, no. 10, 1993, pp.623-628.

[89] T. W. Kao, S. J. Horng, and Y. L. Wang, "An O(1) time algorithms for computing histogram and Hough transform on a cross-bridge reconfigurable array of processors", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. **25**, no. 4, 1995, pp.681-687.

[90] T. W. Kao, S. J. Horng, Y. L. Wang, and K. L. Chung, "A constant time algorithm for computing Hough transform", *Pattern Recognition*, vol. **26**, no. 2, 1993, pp.277-285.

[91] Y. Pan, K. Li, and M. Hamdi, "An improved constant-time algorithm for computing the Radon and Hough transform on a reconfigurable mesh", *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, vol. **29**, no. 4, 1999, pp.417-421.

[92] K. P. Lam, "Implementation of the Hough transform on a fine grained SIMD distributed array processor", *Int. Symposium on Computer Architecture and Digital Signal Processing*, 1989.

[93] C. Guerra, and S. Hambrush, "Parallel algorithms for line detection on a mesh ", *Journal of parallel and distributed computing,* vol. **6**, no. 1, 1989, pp.1-19.

[94] K. L. Chung and H. Y. Lin, "Hough transform on reconfigurable meshes", *Computer Vision and Image Understanding*, vol.**61**, no. 2, 1995, pp.278-284.

[95] S. S. Lee, S. J. Horng, T. W. Kao, and H. R. Tsai, "Optimal computing Hough transform on a reconfigurable array of processors with wider bus networks", *Pattern Recognition*, vol. **29**, no.4, 1996, pp. 603-613.

[96] H. A. H. Ibrahim, J. R. Kender, and D. E. Shaw, "On the application of massively parallel SIMD tree machines to certain intermediate-level vision tasks", *Computer Vision, Graphics, and Image Processing*, vol. **36**, 1986, pp. 53-75.

[97] M. Atiquzzaman, "Pipelined implementation of the multiresolution Hough transform in a pyramid multiprocessor", *Pattern Recognition Letters*, vol. **15**, 1994, pp. 841-851.

[98] Z. N. Li and D. Zhang, "Fast line detection in a hybrid pyramid", *Pattern Recognition Letters*, vol. **14**, 1993, pp. 53-63.

[99] J. M. Jolion and A. Rosenfeld, " An O(log n)pyramid Hough transform", *Pattern Recognition Letters*, vol. **9**, 1989, pp. 343-349.

[100] G. Bongiovanni, C. Guerra, and S. Levialdi, "Computing the Hough transform on a pyramid architecture", *Machine Vision and Application*, vol. **3**, no. 2, 1990, pp.117-123.

[101] H. Y. H. Chuang and L. Chen, "An efficient Hough transform algorithm on SIMD hypercube", *Proc. of the Inter. Conf. on Parallel and Distributed Systems*", 1994, pp.236-241.

[102] J. J. Little, G. E. Blelloch, and T. A. Cass, "Algorithmic techniques for computer vision on a fine-grained parallel machine", IEEE *Trans. on Pattern Analysis and Machine Intelligence*, vol. **11**, no.3, 1989, pp.244-257.

[103] H. Li, " Fast Hough transform for multidimensional signal processing", *IEEE Inter. Conf. on Acoustics, Speech and Signal Processing*, vol.**11**, 1986, pp.2063-2066.

[104] D. B. Tzvi, A. A. Naqvi, and M. Sandler, "Efficient parallel implementation of the Hough transform on a distributed memory system", *Image and Vision Computing*, vol. **7**, no. 3, 1989, pp. 167-172.

[105] H. Y. Chuang and C. C. Li, "A systolic array for straight line detection by modified Hough transform", *Proc. of IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, 1985, pp.300-304.

[106] H. T. Kung and J. A. Webb, "Global operations on a systolic array machine", Proc. *IEEE Int. Conf. on Computer Design VLSI in Computers*, 1985, pp.165-171.

[107] H. F. Li, D. Pao, and R. Jayakumar, "Improvements and systolic implementations of the Hough transformation for straight line detection", *Pattern Recognition*, vol. **22**, no. 6, 1989, pp.697-706.

[108] A. Epstein, G. U. Paul, B. Vettermann, C. Boulin, and F. Klefenz, "A parallel systolic array ASIC for real-time execution of the Hough transform", *IEEE Trans. on Nuclear Science*, vol. **49**, no. 2, 2002, pp.339-346.

[109] K. Hanahara, T. Maruyama, and T. Uchiyama, "A real-time processor for the Hough transform", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. **10**, no. 1, 1988, pp.121-125.

[110] S. B. Shukla, V. Ramakrishnan, and D.P. Agrawai, "A pipelined architecture for on-line low-level vision", *Proceedings on EUROMICRO '90 workshop on Real Time*, 1990, pp.198-204.

[111] J. L. C. Sanz and E. B. Hinkle, "Computing Projections of digital images in image processing pipeline architectures", *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. **35**, no. 2, 1987, pp.198-207.

[112] M. F. X. B. Van Swaaij, F. V. M. Catthoor, and H. J. De Man, " Deriving ASIC architectures for the Hough transform", *Parallel Computing*, vol. **16**, 1990, pp.113-121.

[113] M. Nakanishi and T. Ogura, "A real-time CAM-based Hough transform algorithm and its performance evaluation", *Proceedings of the Inter. Conf. on Pattern Recognition*, vol. **2**, 1996, pp. 516-521.

[114] M. Mahmoud, M. Nakanishi and T. Ogura, "Hough transform Implementation on a reconfigurable highly parallel architecture", *Proceedings on Computer Architecture for Machine Perception*, 1997, pp. 186-194.

[115] M. Nakanishi and T. Ogura, "Real-time extraction using a highly parallel Hough transform board", P*roceedings on Inter. Conf. on Image Processing,* vol. **2**, 1997, pp. 582-585.

[116] J. Volder, "The CORDIC trigonometric computing technique", *IRE Transactions on Electronic Computers*, vol. **8**, no.3, 1950, pp.330-334.

[117] K. Maharatna and S. Banerjee, "A VLSI array architecture for Hough transform", *Pattern Recognition*, vol. **34**, 2001, pp.1503-1512.

[118] A. K. Majumdar, "Design of an ASIC for straight line detection in an image", *Inter. Conf. on VLSI Design*, 2000, pp. 128- 133.

[119 F. M. Rhodes, J. J. Dituri, G. H. Chapman, B. E. Emerson, A. M. Soares, and J. I. Raffel, "A monolithic Hough transform processor based on restructurable VLSI", *IEEE Trans. on Pattern Analysis and Machine Intelligence,* vol. **10**, no. 1, 1988, pp.106-110.

[120] K. Mayasandra, S. Salehi, W. Wang, and H. M. Ladak, "A distributed arithmetic hardware architecture for real-time Hough transform-based segmentation", *Electrical and Computer Engineering*, vol. **40**, no. 4, 2005, pp.1469-1472.

[121] A. Underhill, M. Atiquzzaman, and J. Ophel, "Performance of the Hough transform on a distributed memory", *Microprocessors and Microsystems*, vol. **22**, 1999, pp. 355-362.

[122] M. A. Fischler and O. Firschein, "Parallel guessing: A strategy for high-speed computation", *Pattern Recognition*, vol. **20**, no. 2, 1987, pp.257-263.

[123] K. Hanahara, T. Maruyama, and T. Uchiyama, "High-speed Hough transform processor and its applications to automatic inspection and measurement", *Proceedings on Inter. Conf. on Robotics and Automation*, vol. **3**, 1986, pp. 1954-1959.

[124] N. D. Francis, G. R. Nudd, T. J. Atherton, D. J Kerbyson, R. A. Packwood, and J. Vaudin, "Performance evaluation of the hierarchical Hough transform on an associative M-SIMD architecture", *Proceedings on Inter. Conf. on Pattern Recognition*, vol. **2**, 1990, pp. 509-511.

[125] D. B. Shu, J. G. Nash, M. M. Eshaghian, and K. Kim, "Straight-line detection on a gated-connection VLSI network", *Proceedings on Inter. Conf. on Pattern Recognition*, vol. **2**, no.2, 1990, pp. 456-461.

[126] Canny output (2009, Mar) [Available from: http://www.sci.utah.edu/~cscheid/spr05/imageprocessing/project4/imgs/canny_T10_aniso_arch.png]

[127] S. Chivapreecha and K. Dejhan, "Hardware implementation of Sobel-edge detection distributed arithmetic digital filter", *25th ACRS*, 2004, pp.284- 289.

[128] H. S. Neoh and A. Hazanchuk, "Adaptive edge detection for real-time video processing using FPGAs", *GSPx 2004 conference*, Altera, May 2005, No CF-EDG031505-1.0.

[129] E. Davies, Machine Vision: Theory, Algorithms and Practicalities, Academic Press, 1990, pp 42 - 44.

[130] M. Venkatesan and D. V. Rao, "Hardware acceleration of edge detection algorithm on FPGAs", *Celoxica Inc. research papers*, 2004.

[131] H. S. Neoh and A. Hazanchuk, "Adaptive edge detection for real-time video processing using FPGAs", GSPx 2004 conference, Altera, May 2005, No CF-EDG031505-1.0.

[132] F. Alzahrani and T. Chen, "A stand-alone ASIC for real-time edge detection", *Real-time Imaging*, vol.**3**, Issue.5, 1997, pp.363-378.

[133] P. Lee, E. Alexiadis, "A flexible LUT based synchronizing circuit for 2D imaging filters of variable dimensions implemented on FPGA fabrics", *to be submitted in IET Letters*, January 2010.

[134] SRL16 slice (2009, July) [Available from : http://www.xilinx.com/itp/xilinx5/data/docs/lib/lib0393_377.html]

[135] J. Williams. (2009, Feb) *Microblaze uClinux Project.* 2005; [Available from: http://www.itee.uq.edu.au/~jwilliams/mblaze-uclinux/index.html].

[136] B. Nelson. (2009, June) *The BYU Linux on FPGA Project.* 2005; [Available from: http://splish.ee.byu.edu/projects/LinuxFPGA/].

[137] Altera. (2009, June) *Stratix GX FPGA Family.* 2004; [Available from: http://www.altera.com/literature/ds/ds_sgx.pdf].

[138] *ANSI/IEEE Standard 754-1985, Standard for Binary Floating Point Arithmetic*

[139] N. Shirazi, A.Walters and P. Athanas. "Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines". in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*. 1995. pp. 155-162.

[140] K.S. Hemmert and K.D. Underwood. "An Analysis of the Double-Precision Floating-Point FFT on FPGAs". in *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. 2005. pp. 171 - 180.

[141] Xilinx. (2009, June) *XtreamDSP for Virtex-4 FPGAs User Guide.* 2006; [Available from: www.xilinx.com/bvdocs/userguides/ug073.pdf].

[142] A. Percey. (2009, June) *Advantages of the Virtex-5 FPGA 6-Input LUT Architecture*, 2007; [Available from: http://www.xilinx.com/support/documentation/white_papers/wp284.pdf]

[143] J. Napier, *Mirifici Logarithmorum Canonis Descriptio*. 1615.

[144] G. B. Balaji, K. Balaji, H. Sundararaman, A. Naveen, and K. R. Santha, "Memory reduction techniques for logarithmic number system", in *IEEE – ICSCN 2007*, 2007, pp.410-413.

[145] L. K. Yu and D. Lewis, "A 30-b integrated logarithmic number system processor", IEEE Journal of solid –state circuits, vol.**26**, no.10, 1991, pp.1433-1440.

[146] E.E. Swartzlander, D.V.S. Chandra, H.T. Nagle and S.A. Starks, "Sign Logarithm Arithmetic for FFT Implementation". *IEEE Transactions on Computers*, 1983. **32**(6): pp. 526-534.

[147] J.N. Mitchell, "Computer Multiplication and Division Using Binary Logarithms". *IRE Transactions on Electronic Computers*, 1962. **EC-11**: pp. 512-517.

[148] M. Combet, H. Van Zonneveld and L. Verbeek, "Computation of the Base Two Logarithm of Binary Numbers". *IEEE Transactions on Electronic Computers*, 1965. **EC-14**(6): pp. 863-867.

[149] K.H. Abed and R.E. Siferd, "CMOS VLSI Implementation of a Low-Power Logarithmic Converter". *IEEE Transactions on Computers*, 2003. **52**(11): pp. 1421-1433.

[150] K.H. Abed and R.E. Siferd, "VLSI Implementation of a Low-Power Antilogarithmic Converter". *IEEE Transactions on Computers*, 2003. **52**(9): pp. 1221-1228.

[151] K.J. Dean, "Binary Logarithms". *Electronic Engineering*, 1968. **40**: pp. 560-562.

[152] P.W. Philo, "An Algorithm to Evaluate the Logarithm of a Number to Base 2". *Radio Electronic Engineers*, 1969. **38**: pp. 49-50.

[153] K.J. Dean, "Cellular Logical Array for Obtaining the Square of a Binary Number". *Electronics Letters*, 1969. **5**(16): pp. 370-371.

[154] K.J. Dean, "A Fresh Approach to Logarithmic Computation". *Electronic Engineering*, 1969. **41**: pp. 488-490.

[155] E.E. Swartzlander, D.V.S. Chandra, H.T. Nagle and S.A. Starks, "Sign Logarithm Arithmetic for FFT Implementation". *IEEE Transactions on Computers*, 1983. **32**(6): pp. 526-534.

[156] N.G. Kingsbury and P.J.W. Rayner, "Digital Filtering Using Logarithmic Arithmetic". *Electronics Letters*, 1971. **7**(2): pp. 56-58.

[157] K.H. Abed and R.E. Siferd. "CMOS VLSI Implementation of 16-Bit Logarithm and Anti-Logarithm Converters". in *Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems*. 2000. pp. 776-779.

[158] S.L. SanGregory. "A Fast, Low-Power Logarithm Approximation with CMOS VLSI Implementation". in *IEEE Midwest Symposium on Circuits and Systems*. August 1999.

[159] T. Kurokawa, "Error Analysis of Recursive Digital Filters Implemented with Logarithmic Number Systems". *IEE Trans. on ASSP*, 1980. **28**(6): pp. 706-715.

[160] K.H. Abed and R.E. Siferd. "VLSI Implementations of Low-Power Leading-One Detector Circuits". in *IEEE SoutheastCon*. 2006. pp. 279-284.

[161] V.G. Oklobdzija, "Algorithmic Design of a Hierarchical and Modulator Leading Zero Detector Circuit". *Electronics Letters*, 1993. **29**(3): pp. 283-284.

[162] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, *Digital Image Processing using MATLAB*, Prentice Hall, 2004.

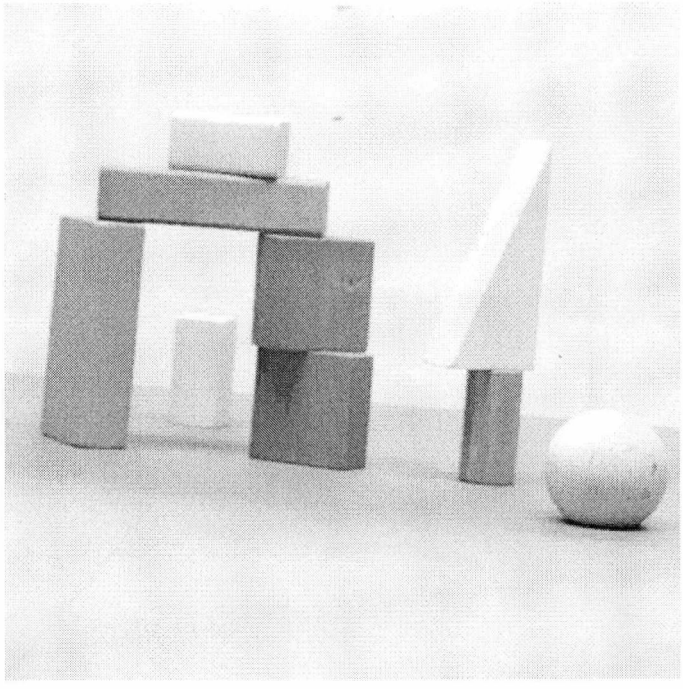
[163] U.o.S. California. (2009, Jan) *The USC-SIPI Image Database*. 2008; [Available from: http://sipi.usc.edu/database/].

[164] J.N. Coleman, E.I. Chester, C.I. Softley and J. Kadlec,"Arithmetic on the European logarithmic microprocessor," *IEEE Transactions on Computers,* vol **49** (100), pg 1152-1152, 2000.

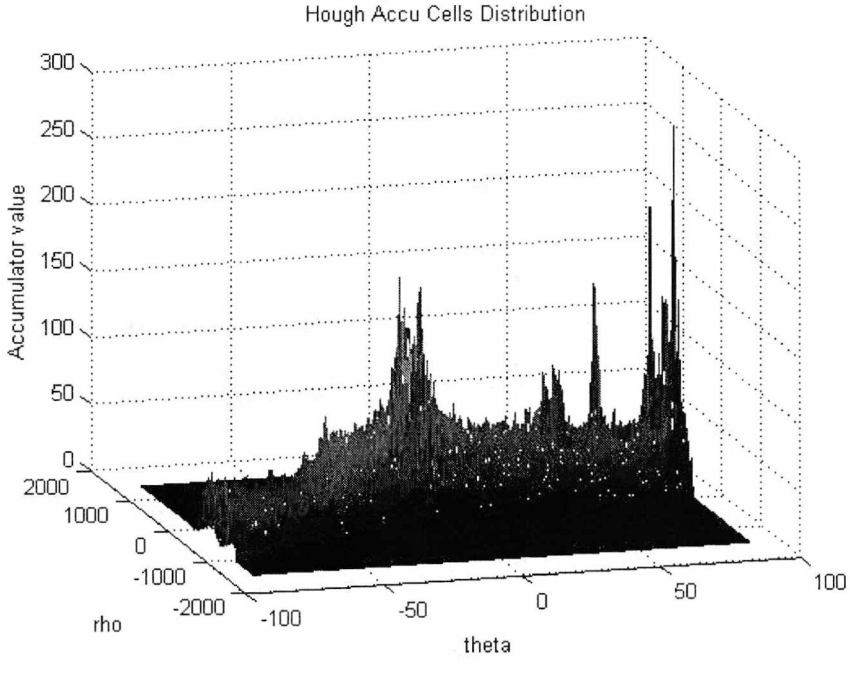
[165] H. Abed, R. E. Siferd, "CMOS VLSI Implementation of a Low-Power Antilogarithmic Converter," *IEEE Trans. on Computers*, Vol. 52, No. 9, pp1221-1228, Sep. 2003.

[166] S. L. SanGregory, R. E. Siferd, C. Brother, D. Gallagher, "A Fast, Low-Power, Logarithm Approximation with CMOS VLSI Implementation," *Proc. IEEE Midwest Symposium on Circuits and Systems*, Aug 1999.

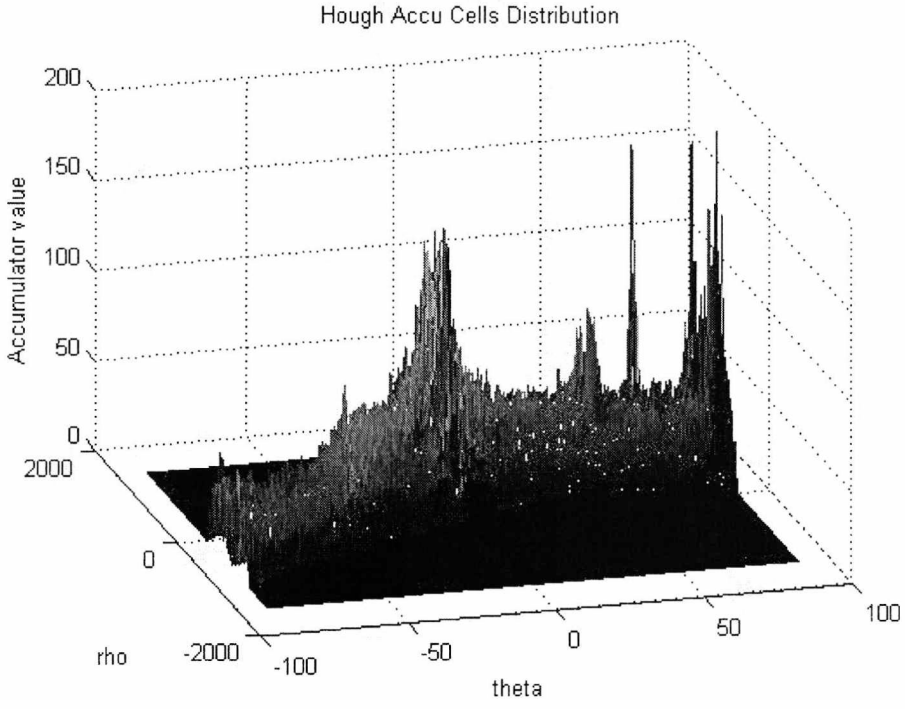
APPENDIX A:Test images and HT parameter space outputs



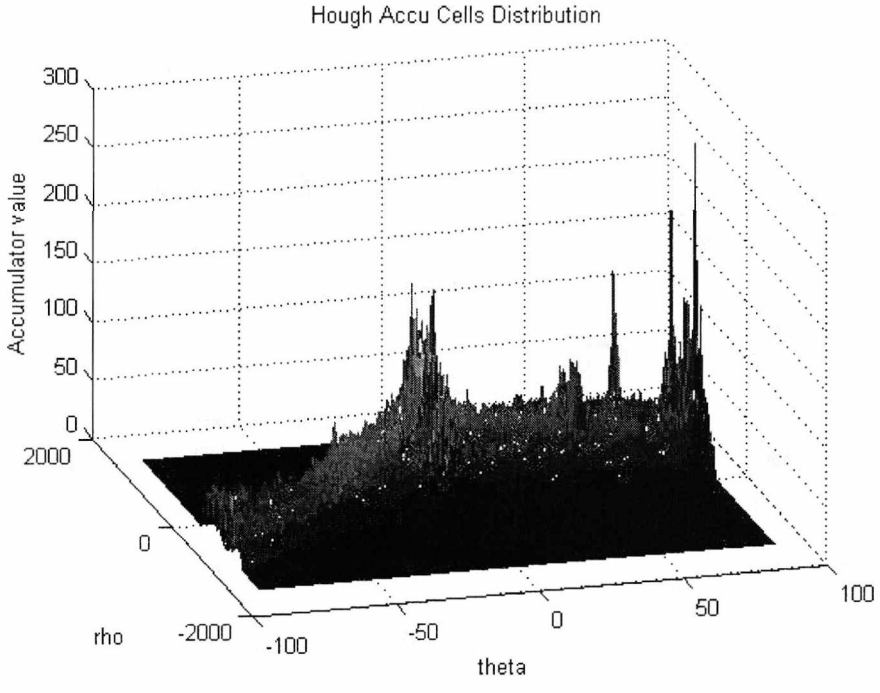
Test image 1024x1024 arch



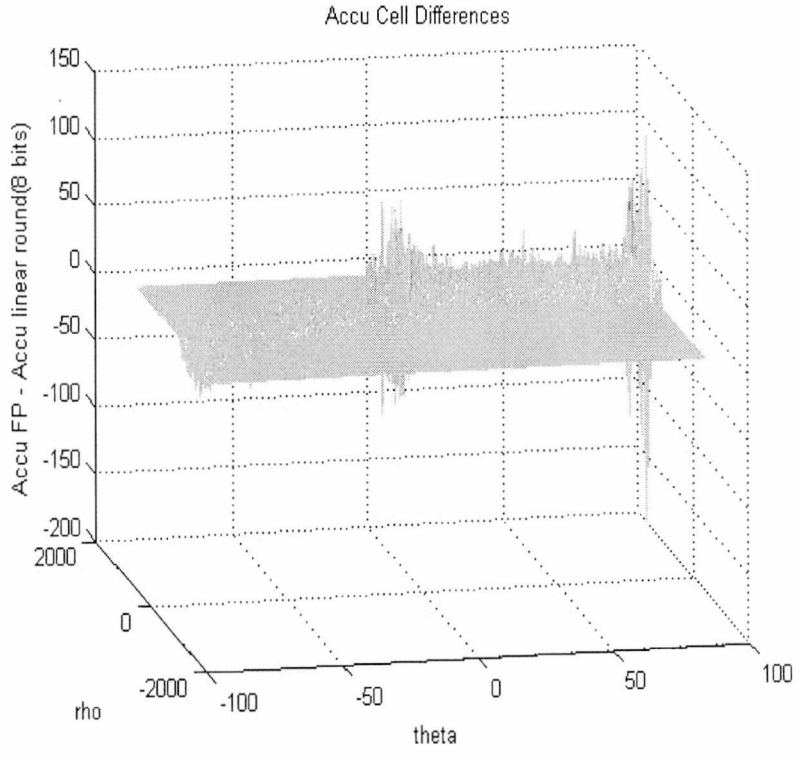
HT parameter space output using floating point arithmetic precision



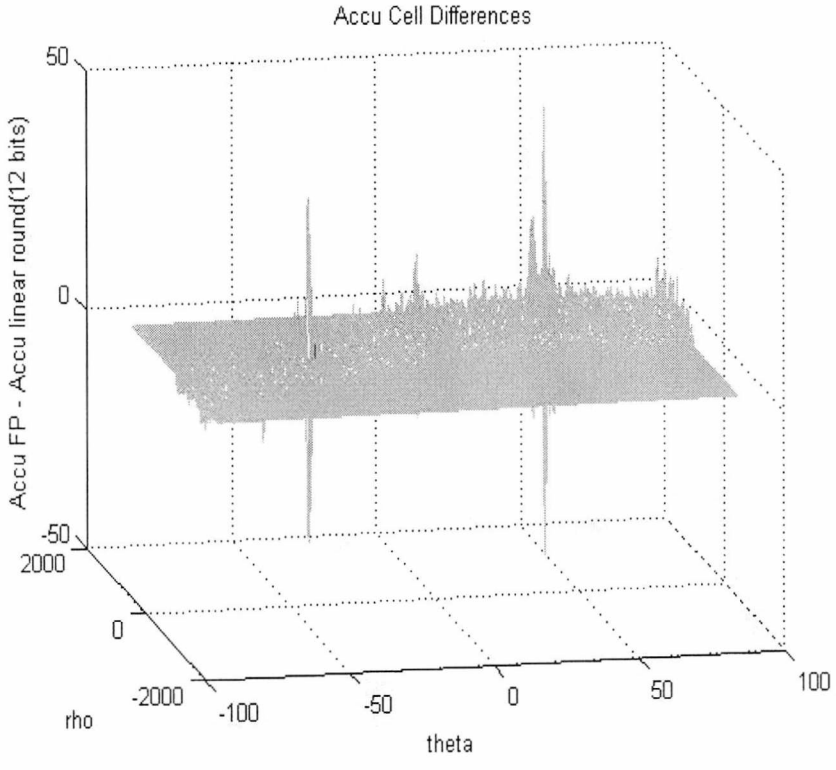
HT parameter space output using linear 8-bit arithmetic precision



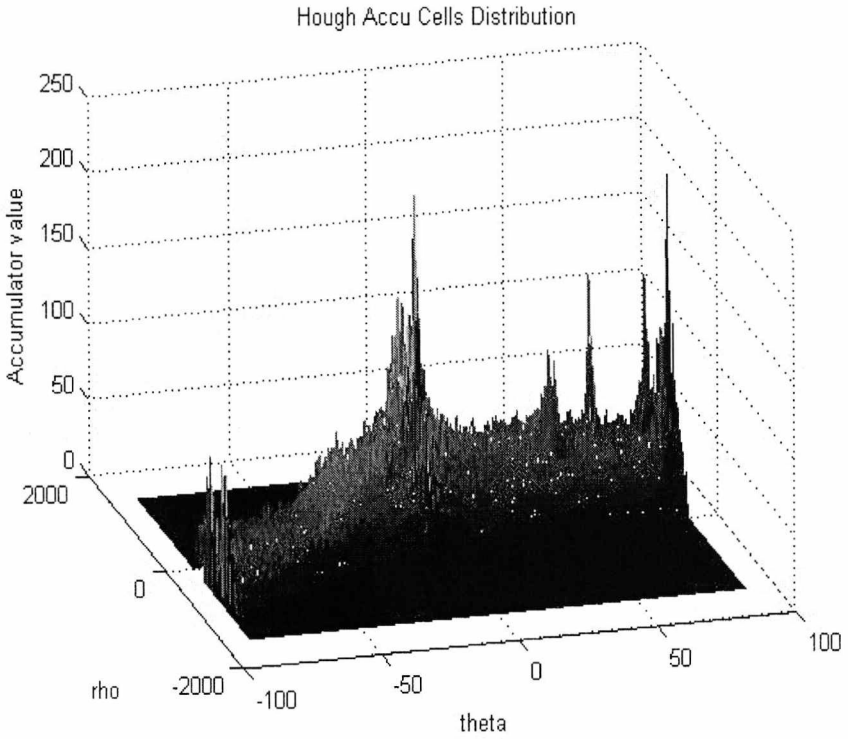
HT parameter space output using linear 12-bit arithmetic precision



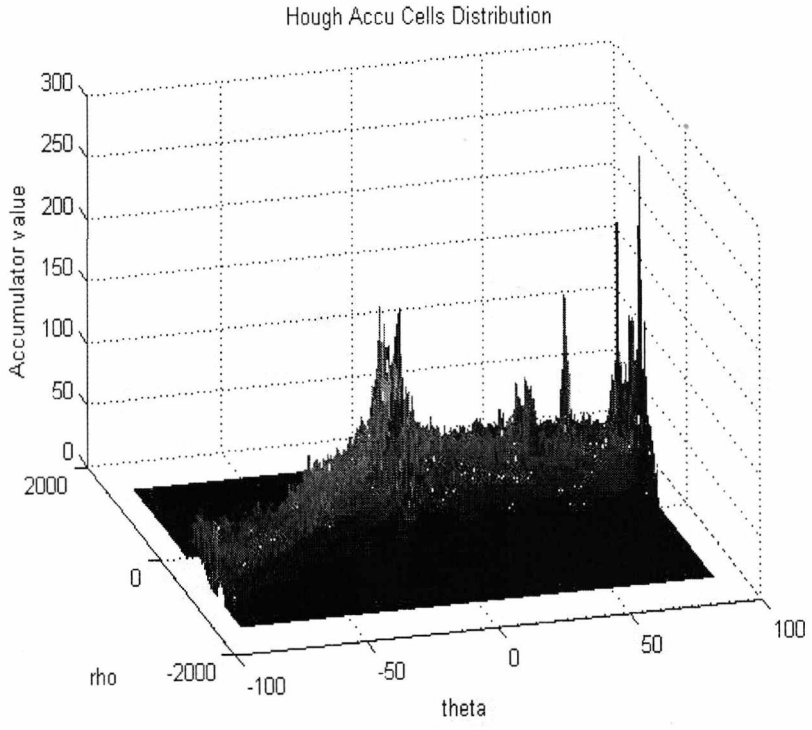
HT parameter space difference between FP and linear 8-bits arithmetic precision



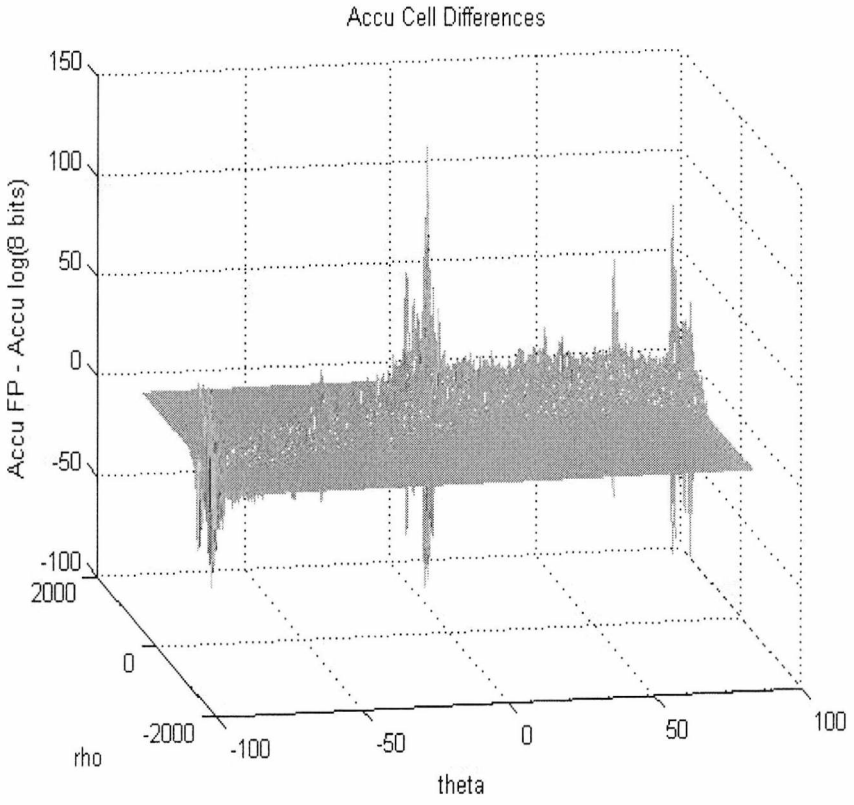
HT parameter space difference between FP and linear 12-bits arithmetic precision



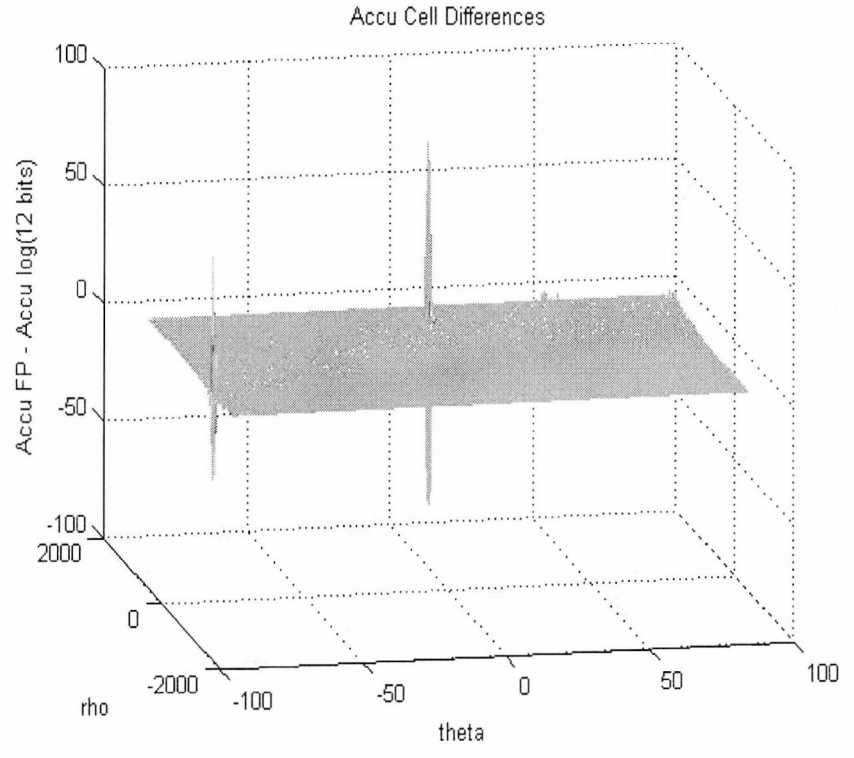
HT parameter space output using Hybrid-log 8-bit arithmetic precision



HT parameter space output using Hybrid-log 12-bit arithmetic precision

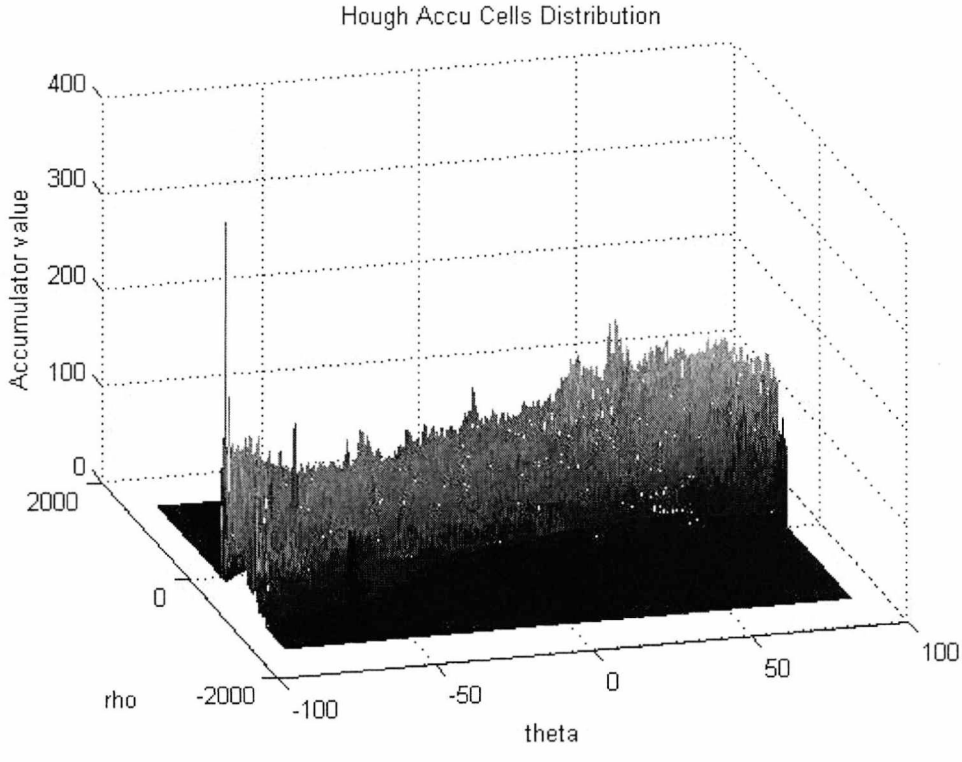


HT parameter space difference between FP and Hybrid-log 8-bits arithmetic precision

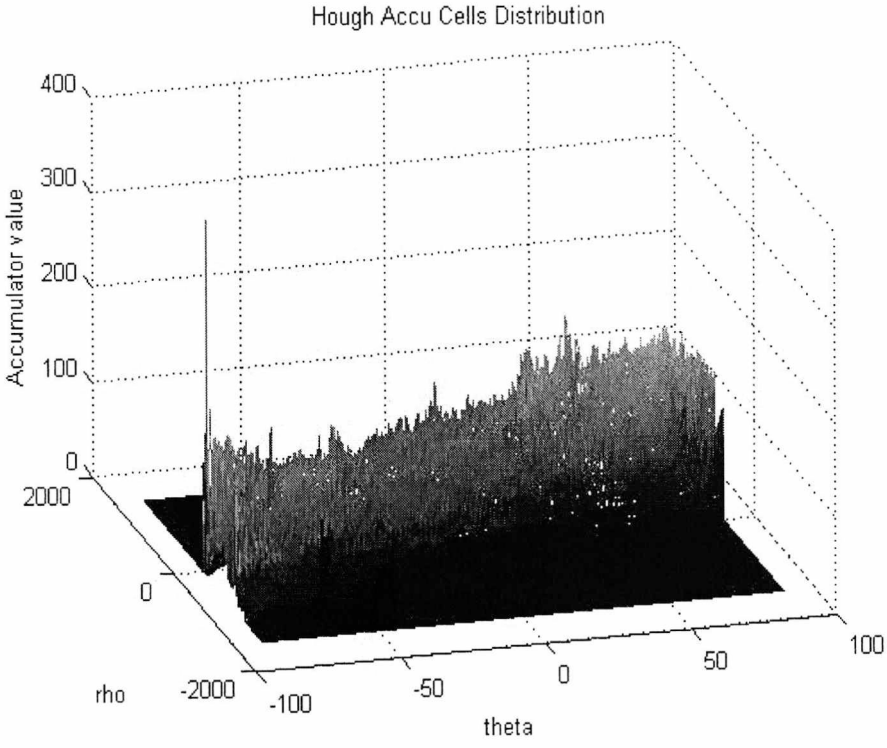


HT parameter space difference between FP and Hybrid-log 12-bits arithmetic precision

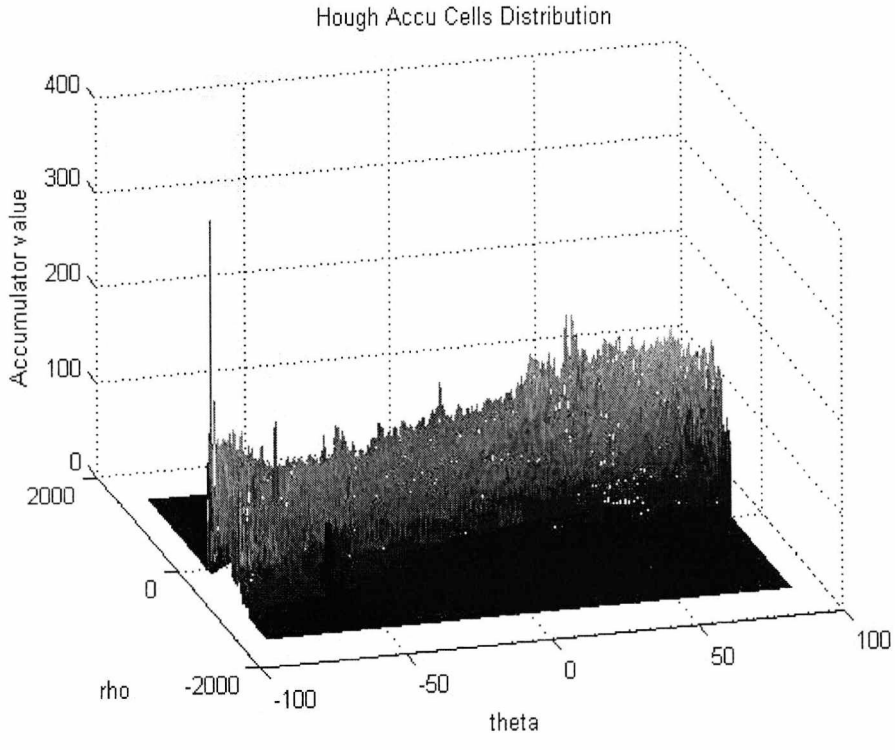
Test image 1024x1024 Lena



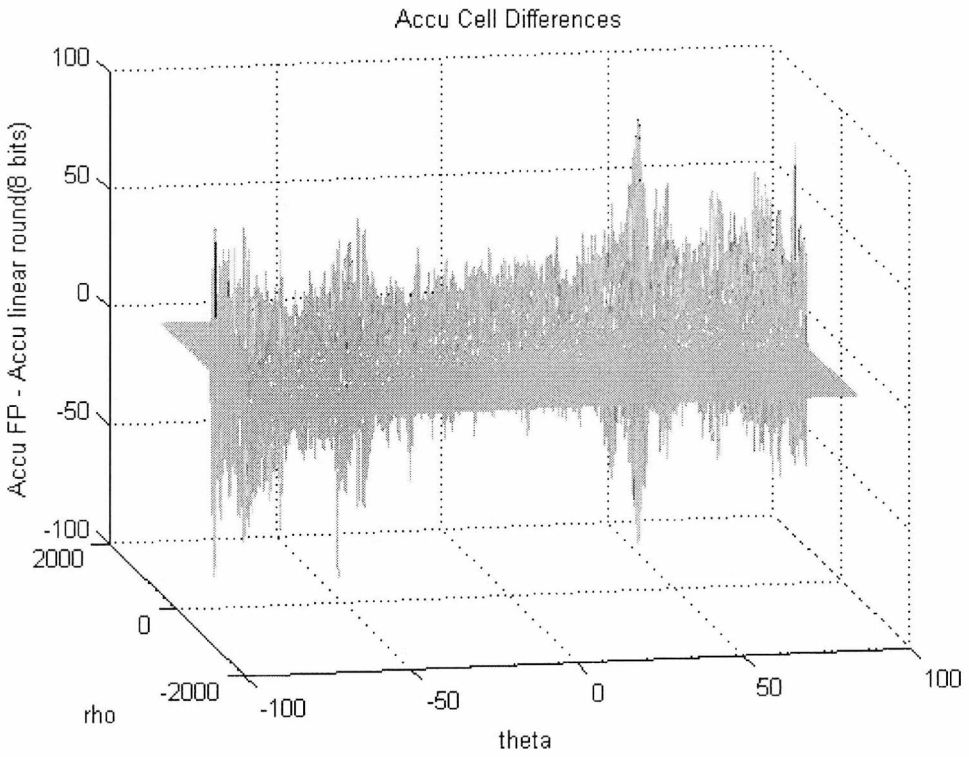
HT parameter space output using floating point arithmetic precision



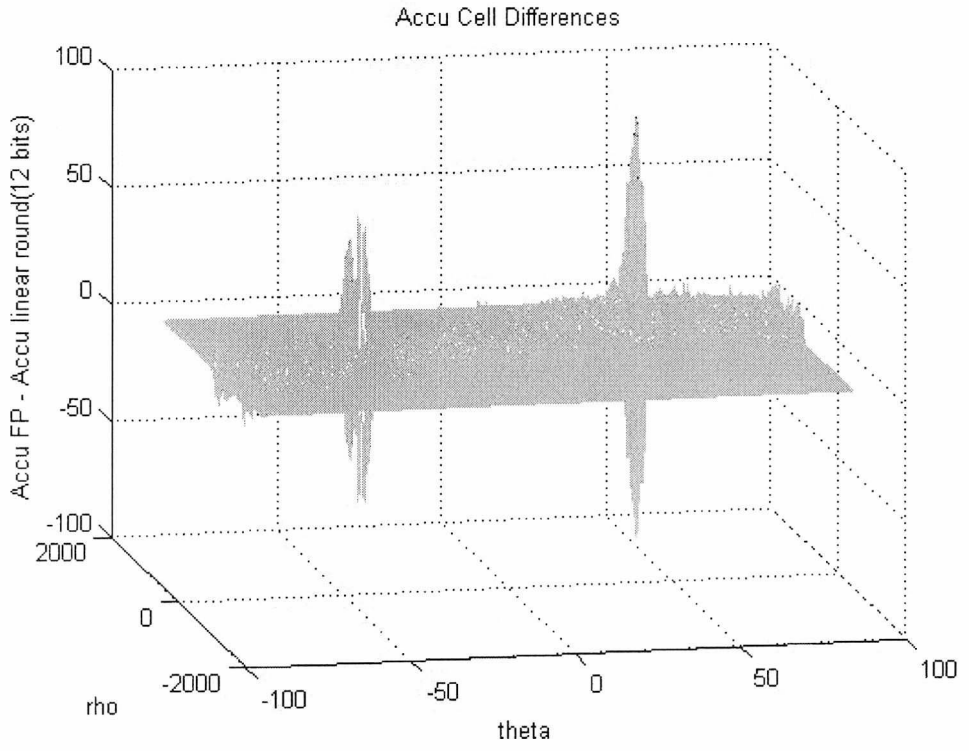
HT parameter space output using linear 8-bit arithmetic precision



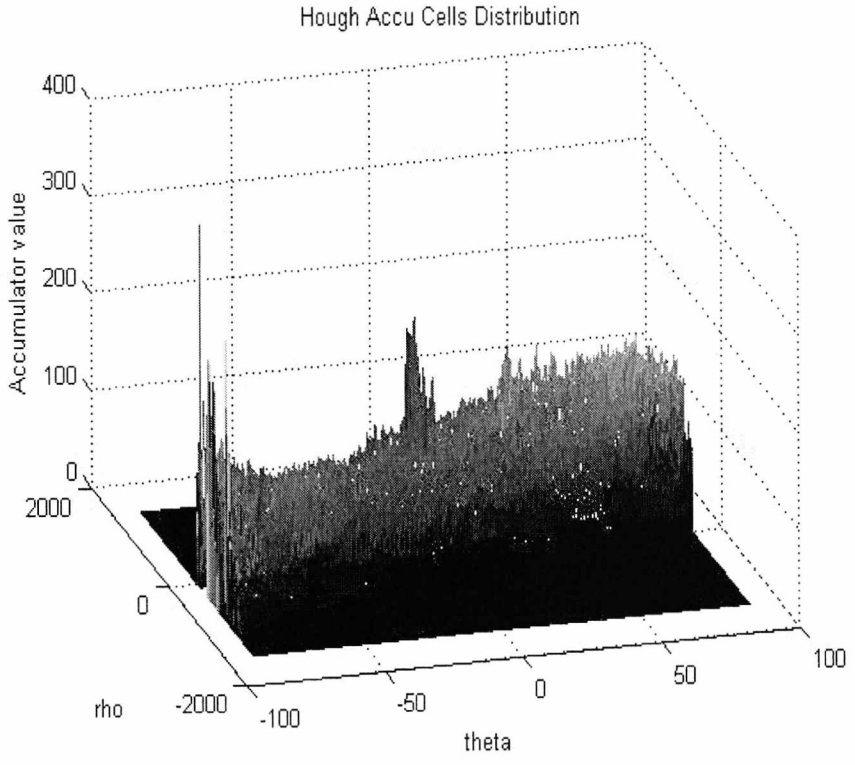
HT parameter space output using linear 12-bit arithmetic precision



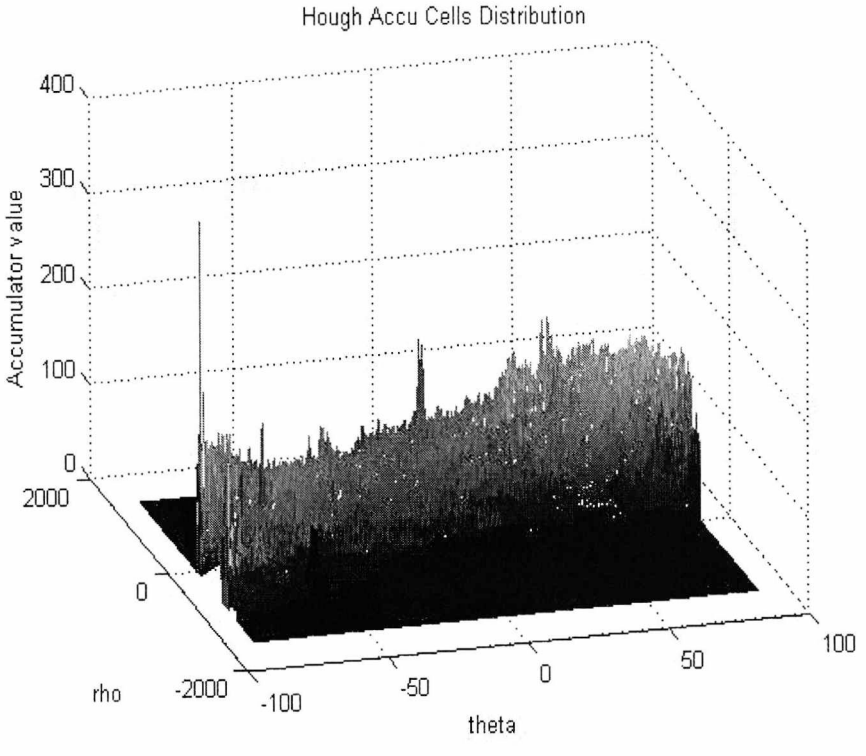
HT parameter space difference between FP and linear 8-bits arithmetic precision



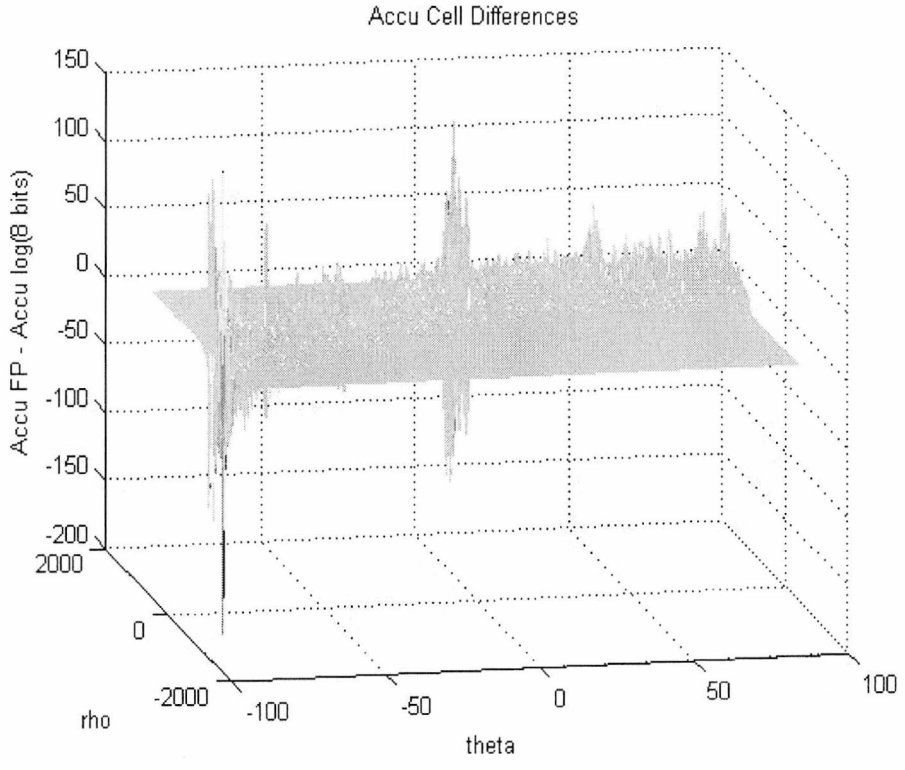
HT parameter space difference between FP and linear 12-bits arithmetic precision



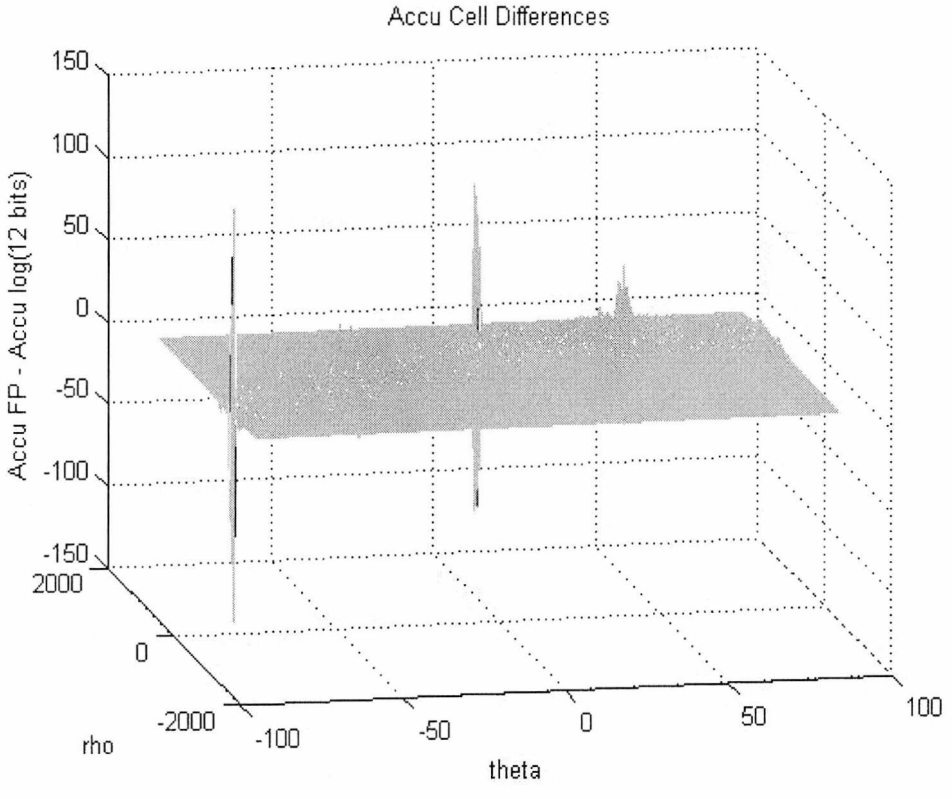
HT parameter space output using Hybrid-log 8-bit arithmetic precision



HT parameter space output using Hybrid-log 12-bit arithmetic precision

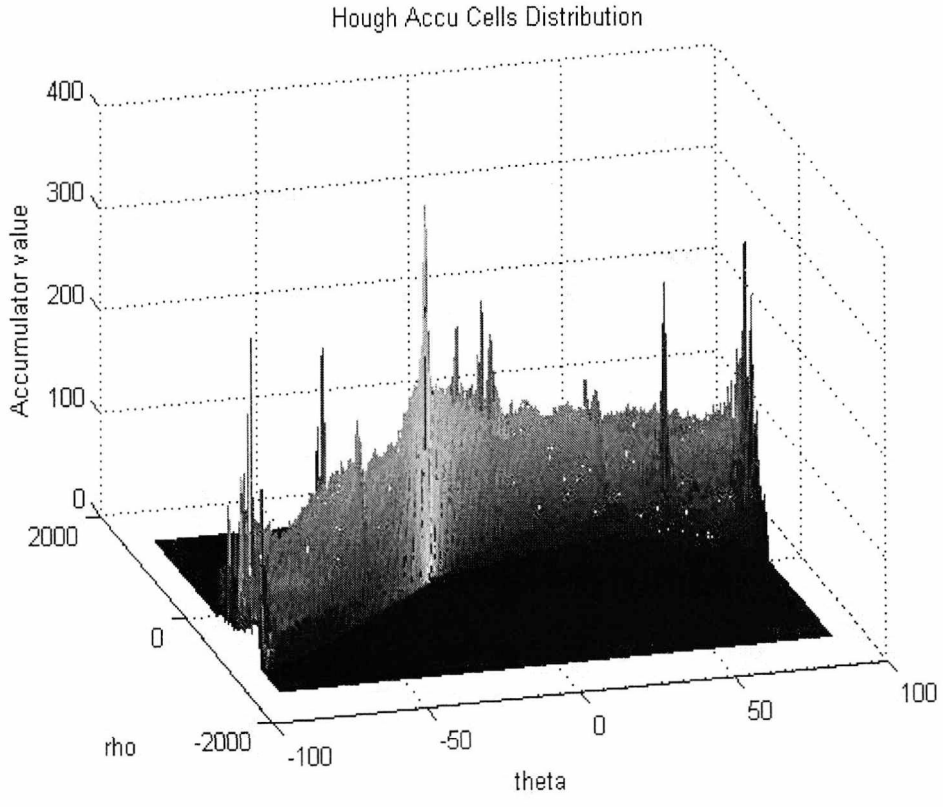


HT parameter space difference between FP and Hybrid-log 8-bits arithmetic precision

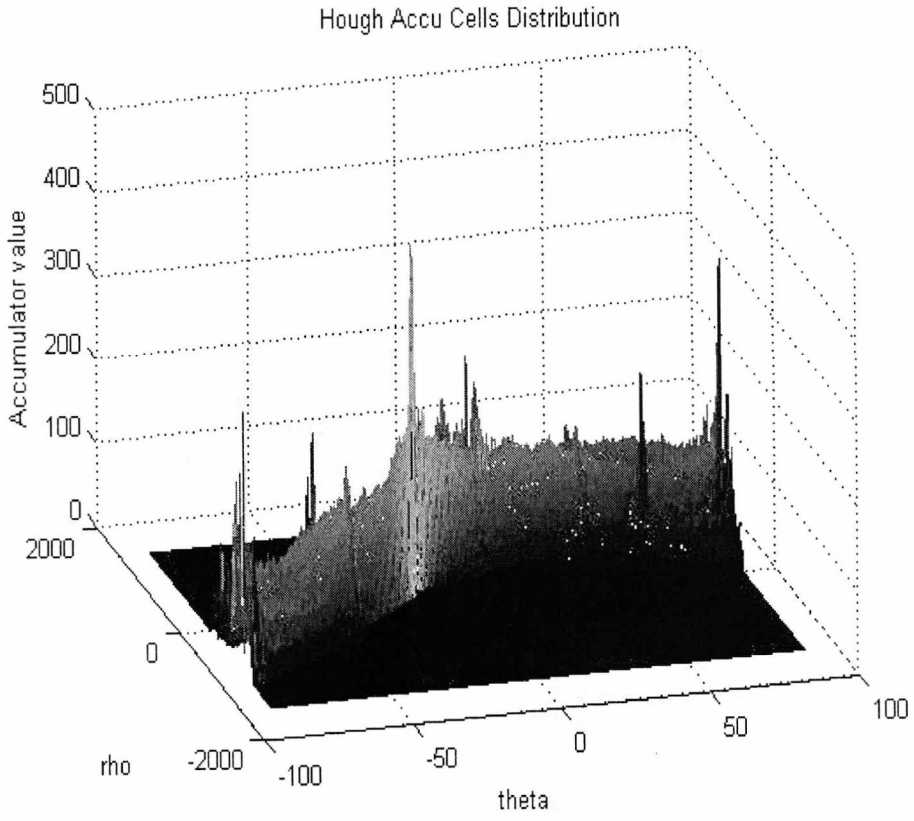


HT parameter space difference between FP and Hybrid-log 12-bits arithmetic precision

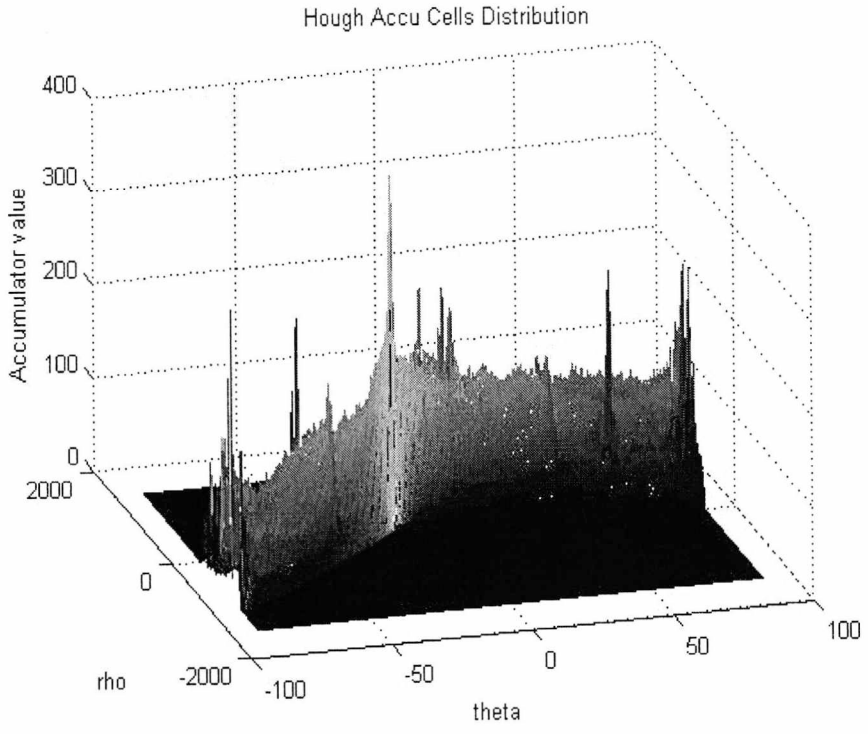
Test image 1024x1024 building



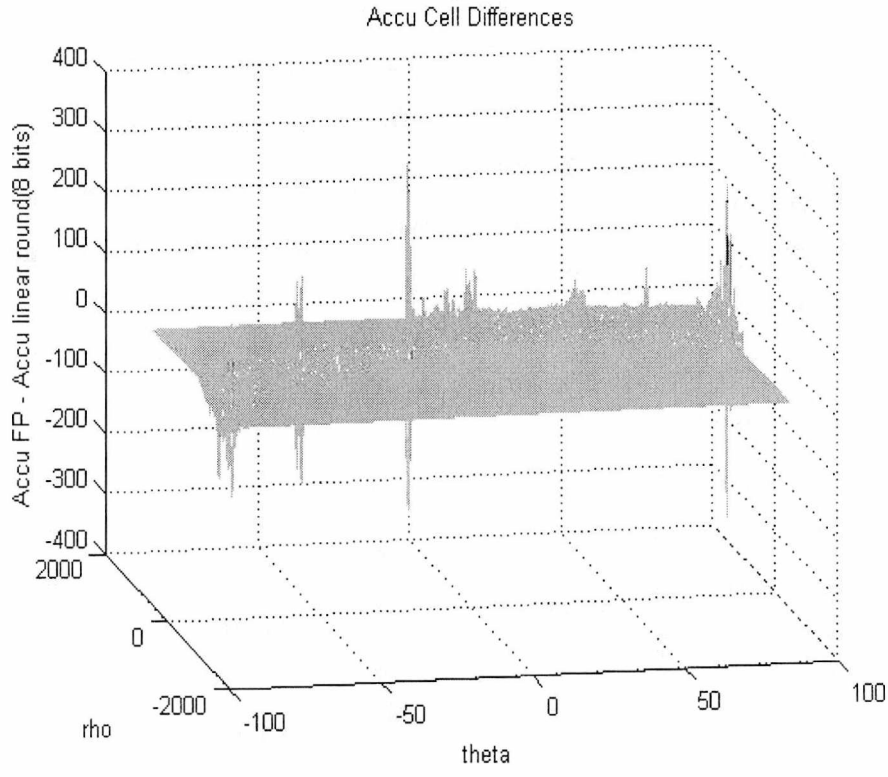
HT parameter space output using floating point arithmetic precision



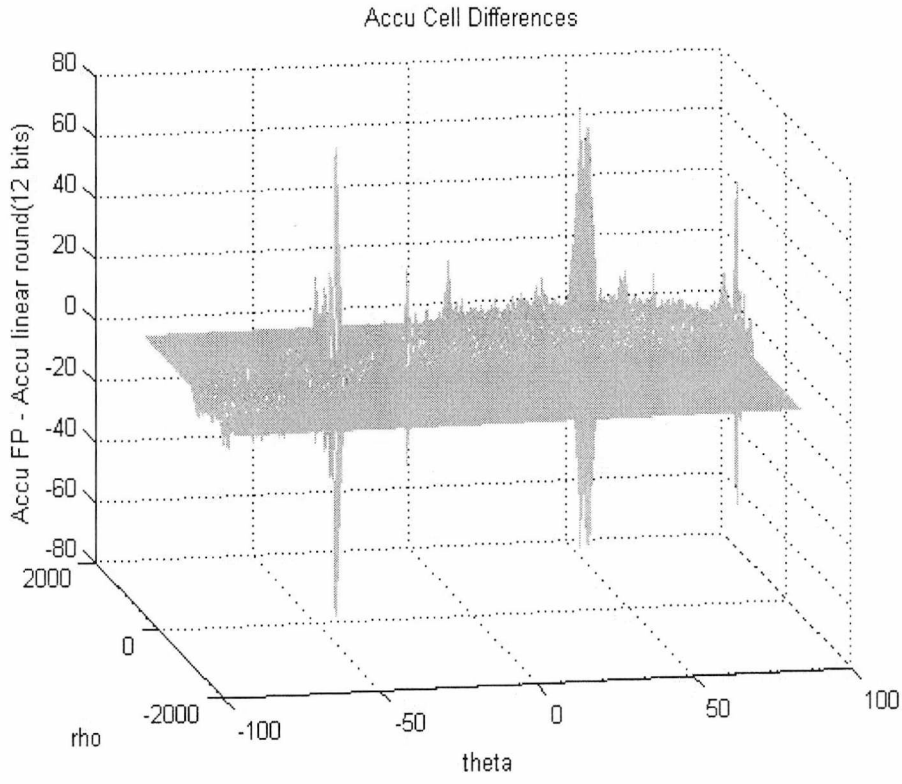
HT parameter space output using linear 8-bit arithmetic precision



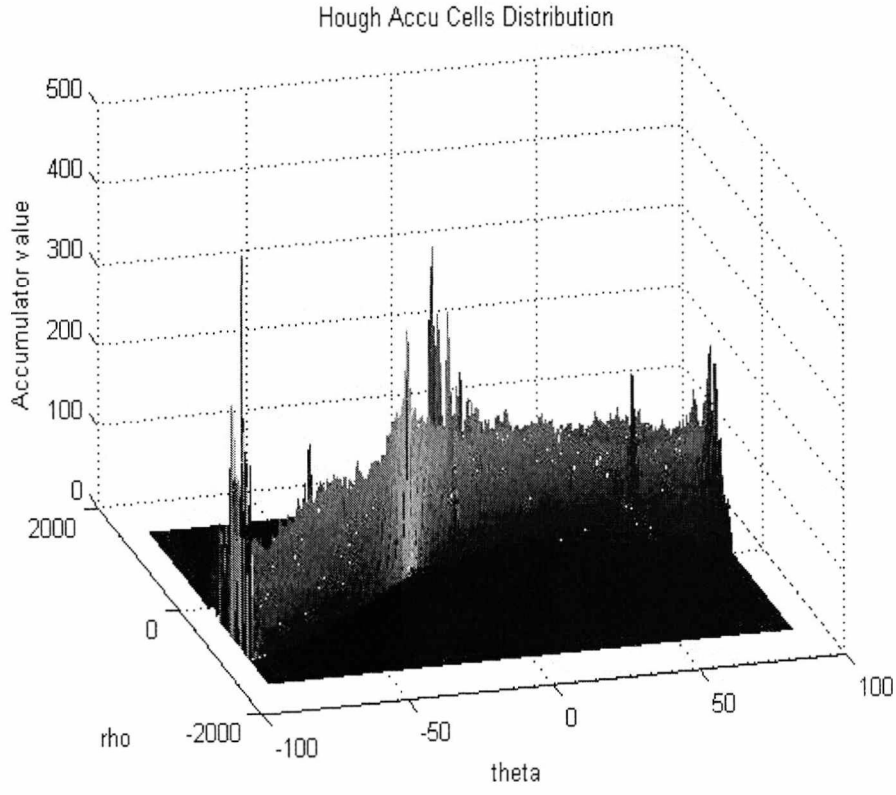
HT parameter space output using linear 12-bit arithmetic precision



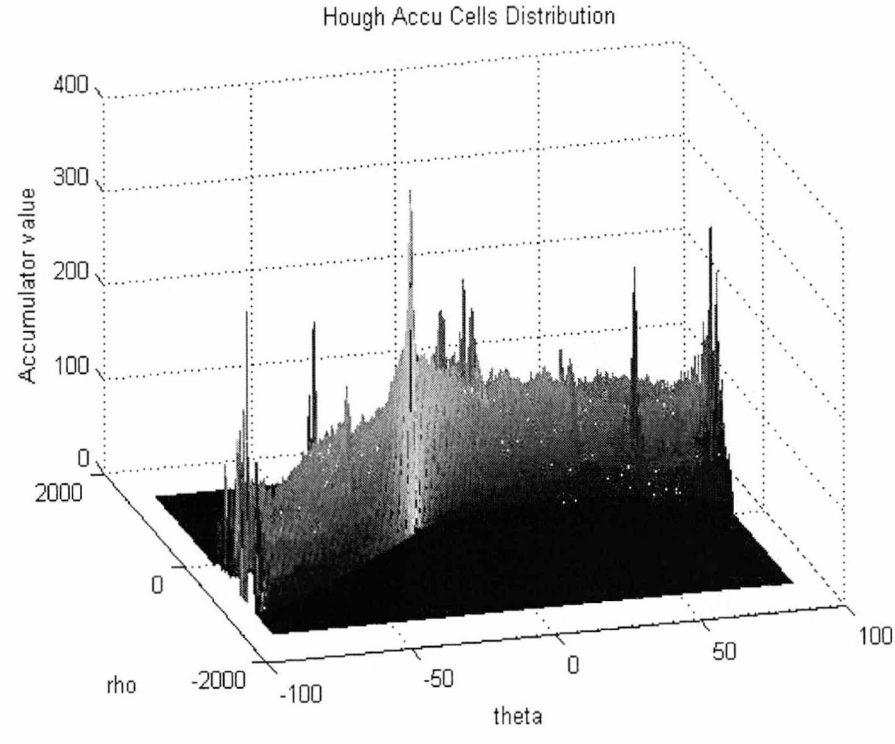
HT parameter space difference between FP and linear 8-bits arithmetic precision



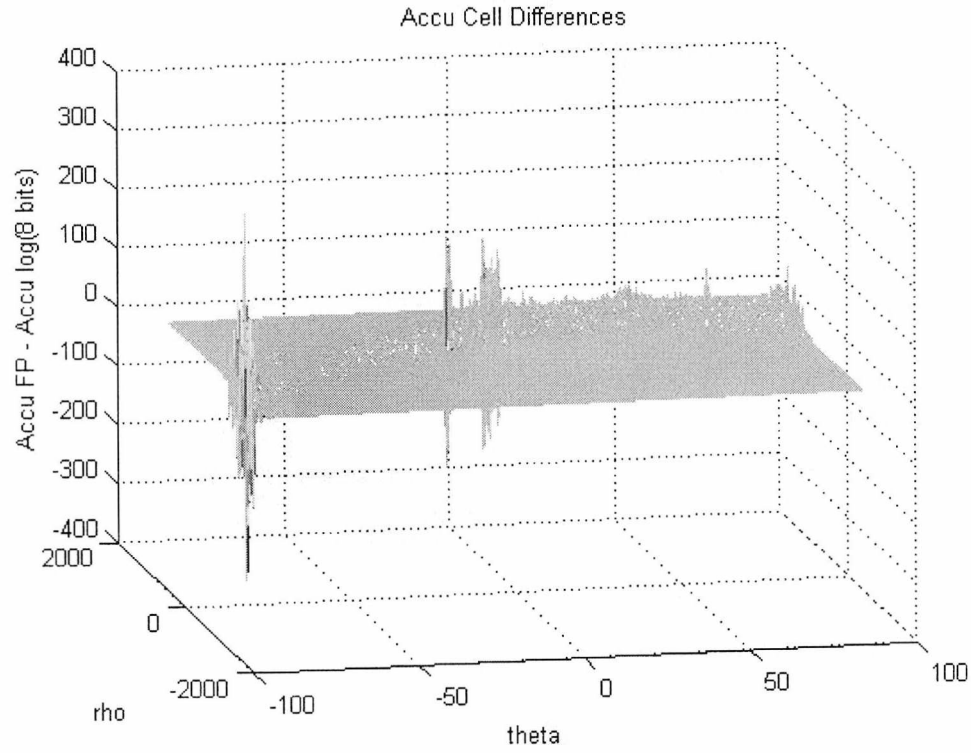
HT parameter space difference between FP and linear 12-bits arithmetic precision



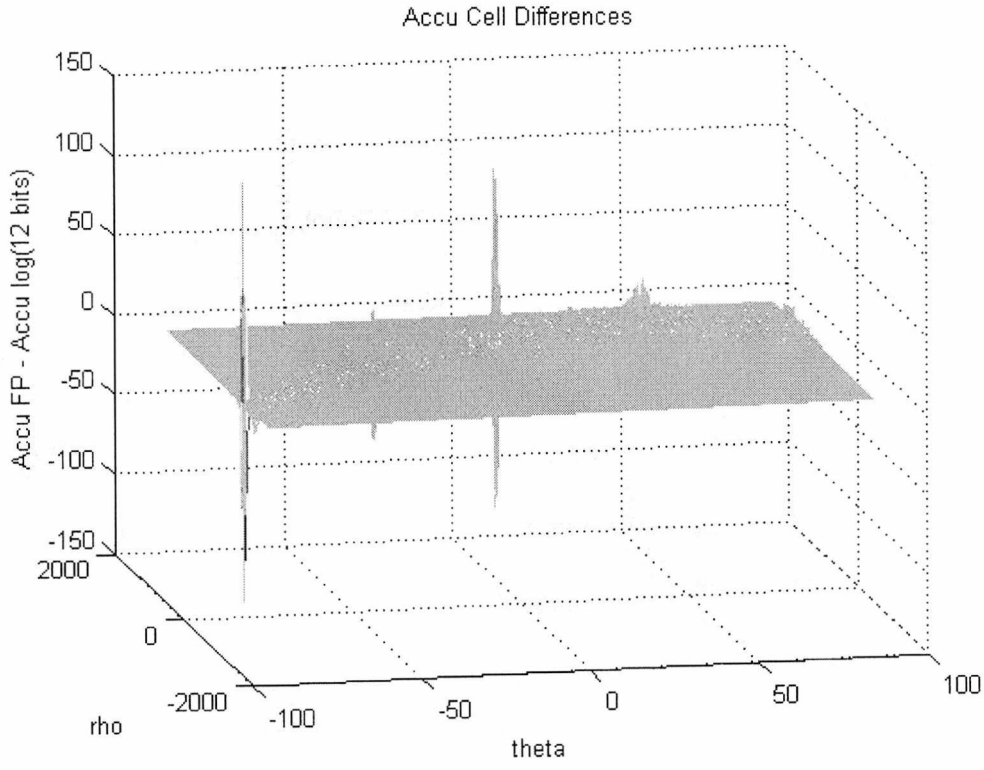
HT parameter space output using Hybrid-log 8-bit arithmetic precision



HT parameter space output using Hybrid-log 12-bit arithmetic precision



HT parameter space difference between FP and Hybrid-log 8-bits arithmetic precision



HT parameter space difference between FP and Hybrid-log 12-bits arithmetic precision

APPENDIX B:Series of tests with different fixed points

Matlab files	.mat file names	No of F.B	Peaks selected							
Hough	Gold_hough	----	284	40	59	53	396	51	68	41
Hough_linear_round										
X	Hough_linear_4	4	284	46	59	62	396	62	68	46
X	Hough_linear_8	8	284	41	59	52	396	54	68	41
Ok	Hough_linear_10	10	284	40	59	52	396	51	68	40
Ok	Hough_linear_12	12	284	41	59	53	396	53	68	40
Ok	Hough_linear_16	16	284	40	59	53	396	52	68	41
Ok	Hough_linear_20	20	284	40	59	53	396	51	68	41
Hough_linear_fix										
X	Hough_linear_fix4	4	284	45	59	57	396	64	68	45
Ok	Hough_linear_fix8	8	284	40	59	52	396	52	68	41
Ok	Hough_linear_fix10	10	284	41	59	53	396	53	68	41
Ok	Hough_linear_fix12	12	284	41	59	53	396	53	68	40
Ok	Hough_linear_fix16	16	284	41	59	53	396	52	68	41
Ok	Hough_linear_fix20	20	284	40	59	53	396	52	68	41
HoughTest_Log2										
X	log_Sc_th8xy1_FB4	4	1326	208	440	269	1364	167	163	134
X	log_Sc_th8xy1_FB5	5	672	113	220	134	1281	103	92	79
X	log_Sc_th8xy1_FB6	6	375	61	109	69	891	71	68	60
X	log_Sc_th8xy1_FB7	7	287	48	56	59	623	62	55	48
X	log_Sc_th8xy1_FB8	8	284	47	47	52	521	53	51	42
X	log_Sc_th8xy1_FB9	9	284	43	56	54	396	54	56	41
X	log_Sc_th8xy1_FB10	10	284	39	59	51	396	53	61	43
Ok	log_Sc_th8xy1_FB11	11	284	39	59	50	396	55	62	42
Ok	log_Sc_th8xy1_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th8xy1_FB20	20	284	40	59	53	396	52	68	41
Ok	log_Sc_th8xy1_FB25	25	284	40	59	53	396	51	68	41
X	log_Sc_th8xy2_FB4	4	1326	208	440	269	1364	167	163	134
X	log_Sc_th8xy4_FB4	4	1326	208	440	369	1364	167	163	134
X	log_Sc_th8xy6_FB4	4	1326	208	440	269	1364	167	163	134
X	log_Sc_th8xy8_FB4	4	1326	208	440	269	1364	167	163	134
X	log_Sc_th8xy10_FB4	4	1326	208	440	269	1364	167	163	134
X	log_Sc_th12xy2_FB4	4	1326	208	440	269	136	167	163	134

X	log_Sc_th16xy2_FB4	4	1326	208	440	269	4 1364	167	163	134
X	log_Sc_th12xy4_FB4	4	1326	208	440	269	1364	167	163	134
X	log_Sc_th16xy4_FB4	4	1326	208	440	269	1364	167	163	134
X	log_Sc_th12xy6_FB4	4	1326	208	440	269	1364	167	163	134
X	log_Sc_th16xy6_FB4	4	1326	208	440	269	1364	167	163	134
X	log_Sc_th12xy8_FB4	4	1326	208	440	269	1364	167	163	134
X	log_Sc_th16xy8_FB4	4	1326	208	440	269	1364	167	163	134

Matlab files	.mat file names	No of F.B	Peaks selected							
HoughTest_Log2										
X	log_Sc_th12xy10_FB4	4	1326	208	440	269	1364	167	163	134
X	log_Sc_th16xy10_FB4	4	1326	208	440	269	1364	167	163	134
X	log_Sc_th8xy2_FB8	8	284	47	47	52	521	53	51	42
X	log_Sc_th8xy4_FB8	8	284	47	47	52	521	53	51	42
X	log_Sc_th8xy8_FB8	8	284	47	47	52	521	53	51	42
X	log_Sc_th8xy10_FB8	8	284	47	47	52	521	53	51	42
X	log_Sc_th12xy2_FB8	8	284	47	47	52	521	53	51	42
X	log_Sc_th16xy2_FB8	8	284	47	47	52	521	53	51	42
X	log_Sc_th12xy4_FB8	8	284	47	47	52	521	53	51	42
X	log_Sc_th16xy4_FB8	8	284	47	47	52	521	53	51	42
X	log_Sc_th12xy6_FB8	8	284	47	47	52	521	53	51	42
X	log_Sc_th16xy6_FB8	8	284	47	47	52	521	53	51	42
X	log_Sc_th8xy2_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th8xy4_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th8xy6_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th8xy8_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th8xy10_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th12xy2_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th16xy2_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th12xy4_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th16xy4_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th12xy6_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th16xy6_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th12xy8_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th16xy8_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th12xy10_FB10	10	284	39	59	51	396	53	61	43
X	log_Sc_th16xy10_FB10	10	284	39	59	51	396	53	61	43

Ok	log_Sc_th8xy2_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th8xy4_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th8xy6_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th8xy8_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th8xy10_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th12xy2_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th16xy2_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th12xy4_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th16xy4_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th12xy6_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th16xy6_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th12xy8_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th16xy8_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th12xy10_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th16xy10_FB12	12	284	38	59	53	396	53	67	40
Ok	log_Sc_th12xy1_FB16	16	284	40	59	53	396	52	68	41
Matlab files	.mat file names	No of F.B	Peaks selected							
HoughTest_Log2										
Ok	log_Sc_th16xy1_FB16	16	284	40	59	53	396	52	68	41
Ok	log_Sc_th20xy1_FB16	16	284	40	59	53	396	52	68	41
Ok	log_Sc_th8xy2_FB16	16	284	40	59	53	396	52	68	41
Ok	log_Sc_th8xy4_FB16	16	284	40	59	53	396	52	68	41
Ok	log_Sc_th8xy8_FB16	16	284	40	59	53	396	52	68	41
Ok	log_Sc_th8xy12_FB16	16	284	40	59	53	396	52	68	41