A DISTRIBUTED CANNY EDGE DETECTION AND ITS FPGA IMPLEMENTATION

by

S.PRASATH

Reg No: 1020106013

of

KUMARAGURU COLLEGE OF TECHNOLOGY,
COIMBATORE - 641 049.
(An Autonomous Institution affiliated to Anna University of Technology, Coimbatore)

A PROJECT REPORT

Submitted to the

FACULTY OF ELECTRONICS AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements for the award of the degree

of

MASTER OF ENGINEERING

in

APPLIED ELECTRONICS

APRIL 2012

BONAFIDE CERTIFICATE

Certified that this project report titled **"A DISTRIBUTED CANNY EDGE DETECTION AND ITS FPGA IMPLEMENTATION"** is the bonafide work of **Mr.S.PRASATH (Reg No: 1020106013)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report of dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

MS.S.UMAMAHESWARI — **PROJECT GUIDE**

DR. RAJESWARI MARIAPPAN — **HEAD OF THE DEPARTMENT**

The candidate with Register No. 1020106013 was examined by us in Project Viva-Voce examination held on _ _ _ _ _ _

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

First I would like to express my praise and gratitude to the Lord, who has showered his grace and blessing enabling me to complete this project in an excellent manner. He has made all things beautiful in his time.

I express my sincere thanks to our beloved Director **Dr.J.Shanmugam**, Kumaraguru College of Technology, I thank for his kind support and for providing necessary facilities to carry out the work.

I express my sincere thanks to our beloved Principal **Dr.S.Ramachandran**, Kumaraguru College of Technology, who encouraged me in each and every steps of the project work.

I would like to express my sincere thanks and deep sense of gratitude to our HOD, **Dr.Rajeswari Mariappan**, Department of Electronics and Communication Engineering, for her valuable suggestions and encouragement which paved way for the successful completion of the project work. I also thank her for her kind support and for providing necessary facilities to carry out the work.

In particular, I wish to thank and everlasting gratitude to the project coordinator **Ms.R.Hemalatha, M.E.,** Assistant Professor**,** Department of Electronics and Communication Engineering for her expert counseling and guidance to make this project to a great deal of success.

I am greatly privileged to express my deep sense of gratitude to my guide **Ms.S.Umamaheswari, M.E.,(Ph.d),** Associate Professor, Department of Electronics and Communication Engineering, Kumaraguru College of Technology throughout the course of this project work and I wish to convey my deep sense of gratitude to all the teaching and non-teaching of ECE Department for their help and cooperation.

Finally, I thank my parents and my family members for giving me the moral support and abundant blessings in all of my activities and my dear friends who helped me to endure my difficult times with their unfailing support and warm wishes.

ABSTRACT

Edge detection is one of the key stages in image processing and object recognition. Canny edge detector treats edge detection as a signal processing problem to design an optimal edge detector and has been widely used for edge detection. The objective of the project is to present a distributed Canny edge detection algorithm which results in significantly reduced memory requirements, decreased latency with no loss in edge detection performance as compared to the conventional Canny algorithm. The new algorithm uses a low-complexity 8-bin non-uniform gradient magnitude histogram to compute block-based hysteresis thresholds that are used by the Canny edge detector. FPGA-based hardware architecture of our proposed algorithm is presented in this project and the architecture is synthesized on the Xilinx FPGA family. Experimental results are also given to show the efficiency of the proposed method.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

FPGA	Field Programmable Gate Array
VLSI	Very Large Scale Integration
VHDL	Very High Speed Hardware Design Language
ASIC	Application Specific Integrated Circuits
VHSIC	Very High Speed Integrated Circuit
RAM	Random Access Memory
MATLAB	Matrix Laboratory
BRAM	Block Select RAM
CLB	Configurable Logic Block
NMS	Non-Maximum Suppression
FIFO	First In First Out
PU	Processing unit
CE	Compute Engine
FIR	Finite Impulse Response

CHAPTER 1
INTRODUCTION

1.1. DIGITAL IMAGE PROCESSING

Digital image processing is an ever expanding and dynamic area with applications reaching out into our everyday life such as in medicine, space exploration, surveillance, authentication, automated industry inspection and in many more areas. Modem image processing applications demonstrate an increasing demand for computational power and memory space. This stems from the fact that image and video resolutions have multiplied in the past few years, especially after the introduction of high definition video and high resolution digital cameras. Therefore there is a need for image processing implementations that can perform demanding computations on substantial amounts of data, with high throughput, and often need to meet real-time requirements.

1.2 PROJECT GOAL

The ultimate goal of this project is to detect the edges of the image in a better manner. This project uses block based method for detecting even smaller edges in definite manner. Also the FPGA implementation of the proposed algorithm is presented which is necessary for real time applications.

1.3 SOFTWARE USED

- MATLAB 2009a
- Modelsim
- Xilinx ISE 8.1i

CHAPTER 2
LITERATURE REVIEW

2.1 EDGE DETECTION

Image edge is a fundamental feature of image, which contains abundant internal information, such as direction, step characteristics, shape and etc, so it has been widely used in image segmentation, image categorization, image registration, and pattern recognition. The edge mostly exists between object and object, object and background, area and area. Edge detection is considered as an important research of this domain and a huge amount of researches has been conducted, typical the first order differential operator such as Roberts operator, Prewitt operator, and Sobel operator and second order differential operator such as Laplace operator and LOG operator.

Edge detection serves as a pre-processing step for many image processing algorithms such as image enhancement, image segmentation, tracking and image/video coding. Typically, edge detection algorithms are implemented using software. With advances in Very Large Scale Integration (VLSI) technology, their hardware implementation has become an attractive alternative, especially for real-time applications.

2.2 PLATFORMS USED FOR DSP DESIGN

There are several different choices a designer has when implementing a DSP system of any sort. Hardware, of course, offers much greater speed than a software implementation, but one must consider the increase in development time inherent in creating a hardware design. Most software designers are familiar with C, but in order to develop a hardware system, one must either learn a hardware design language such as VHDL or Verilog, or use a software-to-hardware conversion scheme, such as Streams-C, which converts C code to VHDL, or MATCH , which converts MATLAB code to VHDL.

While the goals of such conversion schemes are admirable, they are currently in development and surely not suited to high speed applications such as video processing. Ptolemy is a system that allows modelling, design, and simulation of embedded systems.

Ptolemy provides software synthesis from models. While this type of system may be a dominant design platform in the future, it is still under much development, meaning that it may not be a viable design choice for some time. A discussion on the various viable options for DSP system design is found below.

2.2.1 PC DIGITAL SIGNAL PROCESSING PROGRAMS

Signal processing programs used on a PC allow for rapid development of algorithms, as well as equally rapid debug and test capabilities. It is common for many hardware designers to use some sort of PC programming environment to implement a design to verify functionality prior to a lengthy hardware design.

MATLAB is such an environment. Although it was created for manipulating matrices in general, it is well suited to some image processing applications. MATLAB treats an image as a matrix, allowing a designer to develop optimized matrix operations implementing an algorithm. However, if the eventual goal is a hardware device, the algorithms are instead often written to operate similarly to the proposed hardware system, which results in an even slower algorithm. Systems such as ID and its graphical component ENVI are more specifically geared to image processing applications, and include many pre-written algorithms commonly used to process images.

However, even specialized image processing programs running on PCs cannot adequately process large amounts of high-resolution streaming data, since PC processors are made to be for general use. Further optimization must take place on a hardware device.

2.2.2 APPLICATION SPECIFIC INTEGRATED CIRCUITS

Application Specific Integrated Circuits (ASICs) represent a technology in which engineers create a fixed hardware design using a variety of tools. Once a design has been programmed onto an ASIC, it cannot be changed. Since these chips represent true, custom hardware, highly optimized, parallel algorithms are possible. However, except in high-volume commercial applications, ASICs are often considered too costly for many designs. In addition, if an error exists in the hardware design and is not discovered before product shipment, it cannot be corrected without a very costly product recall.

2.2.3 DEDICATED DIGITAL SIGNAL PROCESSORS

Digital Signal Processors (DSPs) such as those available from Texas Instruments are a class of hardware devices that fall somewhere between an ASIC and a PC in terms of performance and design complexity. They can be programmed with either assembly code or the C programming language, which is one of the platform's distinct advantages. Hardware design knowledge is still required, but the learning curve is significantly lower than some other design choices, since many engineers have knowledge of C prior to exposure to DSP systems. However, algorithms designed for a DSP cannot be highly parallel without using multiple DSPs. Algorithm performance is certainly higher than on a PC, but in some cases, ASIC or FPGA systems are the only choice for a design. Still, DSPs are a very common and efficient method of processing real-time data.

One area where DSPs are particularly useful is the design of floating point systems. On ASICs and FPGAs, floating-point operations are rather difficult to implement. For the scope of this project, this is not an issue because all images consist of only integer data.

Recent advances in DSP technology have resulted in very high-speed algorithm implementations. While the advantages of ASICs and FPGAs are still applicable, this new generation of DSPs has made some engineers reconsider FPGA development. Still, as new DSPs arrive to the market, so do new FPGAs, and it is expected that the two architectures will have similarly increasing performance for each new generation of processors.

2.2.4 FIELD PROGRAMMABLE GATE ARRAYS

Field Programmable Gate Arrays (FPGAs) represent reconfigurable computing technology, which is in some ways ideally suited for video processing. Reconfigurable computers are processors which can be programmed with a design, and then reprogrammed (or reconfigured) with virtually limitless designs as the designer's needs change.

FPGAs generally consist of a system of logic blocks (usually look up tables and flip-flops) and some amount of Random Access Memory (RAM), all wired together using a vast array of interconnects. All of the logic in an FPGA can be rewired, or reconfigured, with a different design as often as the designer likes. This type of architecture allows a large variety

of logic designs dependent on the processor's resources), which can be interchanged for a new design as soon as the device can be reprogrammed.

Today, FPGAs can be developed to implement parallel design methodology, which is not possible in dedicated DSP designs. ASIC design methods can be used for FPGA design, allowing the designer to implement designs at gate level. However, usually engineers use a hardware language such as VHDL or Verilog, which allows for a design methodology similar to software design. This software view of hardware design allows for a lower overall support cost and design abstraction.

2.3 FPGA DESIGN OPTIONS

In order to create an FPGA design, a designer has several options for algorithm implementation. While gate-level design can result in optimized designs, the learning curve is considered prohibitory for most engineers, and the knowledge is not portable across FPGA architectures. The following text discusses several high-level hardware design languages (HDLs) in which FPGA algorithms may be designed

2.3.1 VERILOG HDL

Originally intended as a simulation language, Verilog HDL represents a formerly proprietary hardware design language. Currently Verilog can be used for synthesis of hardware designs and is supported in a wide variety of software tools. It is similar to the other HDLs, but its adoption rate is decreasing in favour of the more open standard of VHDL. Still, many designers favour Verilog over VHDL for hardware design, and some design departments use only Verilog. Therefore, as a hardware designer, it is important to at least be aware of Verilog

2.3 .2 ALTERA HARDWARE DESIGN LANGUAGE

Altera Hardware Design Language (AHDL) is proprietary, and is only supported in Altera-specific development tools. This may be seen as a drawback, but since AHDL is proprietary, its use can also result in more efficient hardware design, when code portability is not an issue. In typical design environments, different FPGA architectures are used for

different designs, meaning that time spent learning AHDL may be wasted if a Xilinx FPGA is later chosen.

2.3.3 VHSIC HARDWARE DESIGN LANGUAGE

In recent years, VHSIC (Very High Speed Integrated Circuit) Hardware Design Language (VHDL) has become a sort of industry standard for high-level hardware design. Since it is an open IEEE standard, it is supported by a large variety of design tools and is quite interchangeable (when used generically) between different vendors' tools. It also supports inclusion of technology-specific modules for most efficient synthesis to FPGAs.

The first version of VHDL, IEEE 1076-87, appeared in 1987 and has since undergone an update in 1993, appropriately titled IEEE 1076-93. It is a high-level language similar to the computer programming language Ada, which is intended to support the design, verification, synthesis and testing of hardware designs.

2.4 DESIGN APPROACH

Prior to any hardware design, the author chose to create software versions of the algorithms in MATLAB. Using MATLAB procedural routines to operate on images represented as matrix data, these software algorithms were designed to resemble the hardware algorithms as closely as possible. While a hardware system and a matrix-manipulating software program are fundamentally different, they can produce identical results, provided that care is taken in development. This approach was taken because it speeds understanding of the algorithm design. In addition, this approach facilitates comparison of the software and synthesized hardware algorithm outputs, allowing detailed error calculations.

This project was targeted for FPGA systems for two reasons. One, the author had some previous experience in FPGA implementations of video processing algorithms. Two, FPGAs represent a new direction for DSP systems, and there is much original work to be done in terms of optimized algorithms for this type of system.

VHDL was chosen as a target design language because of familiarity and its wide-ranging support, both in terms of software development tools and vendor support. Today,

more engineers are learning VHDL than Verilog, which is another compelling reason for its use in this project.

The design flow for this project is represented in Figure 1.5.1. This shows the interaction between the VHDL design environment and the FPGA-specific tools. In the first state, a design is created in VHDL. Next, the code's syntax is verified and the design is synthesized, or compiled, into a library. The design is next simulated to check its functionality. Stimulating the signals in the design and viewing the output waveforms in the VHDL simulator allows the designer to determine proper functionality of the design.

Next, the design is processed with vendor-specific place-and-route tools and mapped onto a specific FPGA in software. This allows the engineer to view a floor plan and hierarchical view of the design, which can help verifying a proper mapping procedure. Next, the design is verified for proper functionality once again. This step is important because it assures that the design is correct in its translation from VHDL to gate level. If this is found to be correct, the design can then be programmed onto the specified FPGA

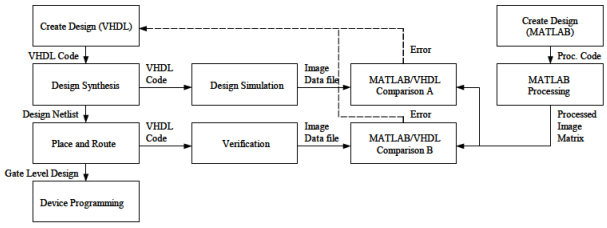

Figure 2.1: Hardware Design Flow

2.5 FIELD PROGRAMMABLE GATE ARRAYS (FPGAS)

A Field Programmable Gate Array (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. In its simplest form an FPGA consists of an array of uncommitted elements that can be programmed or interconnected according to a user's specification. The ability to reprogram these devices over and over again of the flexibility of interconnection resources makes FPGAs an ideal device for implementing & testing ASIC prototypes. The Figure 2.1 portrays the architecture of a conceptual FPGA.

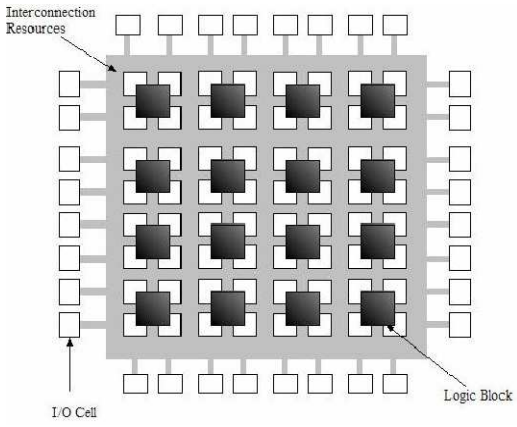

Figure 2.2: FPGA Architecture

The most important components in an FPGA are configurable logic blocks (CLB's), input-output blocks and programmable switches. The architecture has a two dimensional array of CLBs that are by general interconnection resources. These CLBs can be as simple as 2-input NAND gates or it can have a complex structure such as multiplexers or look-up tables.

Most logic blocks also contain some type of flip- flop, to aid in the implementation of sequential circuits. The interconnect consists of segments of wire, where the segments may be of various lengths. These interconnects are made up programmable switches that serve to connect the CLBs to the wire segments, or one wire segment to another. The wire segments

along with programmable switches are together wined as routing architecture. Similar to the logic blocks, these switches can be designed in many ways.

Some FPGAs offer a large number of simple connections between blocks where as others provide fewer, but complex routes. The programmable switches can be constructed in several ways including: pan-transistors controlled by static RAM cells, antifuses, EPROM transistors and EEPROM transistors. Commercially FPGAs have been classified into four-major categories based on their interconnection. The interconnection can be symmetrical array, row based, hierarchical, or sea of gates. Table 2.1 shows commercially available FPGA's.

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

Table 2.1 Summary of Four Commercial FPGAs

2. 6 BLOCK SELECT RAM (BRAMS)

Block Select RAM (BRAMs) are dedicated blocks of memory that can store large amounts of data. Each memory block is four CLBs high and is organized into memory columns stretching the entire height of the chip. There is one such memory column between every twelve CLB columns. The block Select RAM also includes dedicated routing to provide an efficient interface with both CLBs and other block Select RAM Each Block Select RAM is a fully synchronous dual-ported and can store 4096 bits.

Each port has independent control signals so that the two ports can be configured independently. The width of each addressable location can vary from 1 to 16 bits. For example, if each location is 16-bits wide, then there will be 256 such locations within one Block Select RAM memory. The dual-port Block Select RAM is used our implementation to store image data.

2.7 OPTIMIZATION TECHNIQUES

Each of the logic gates in the circuit has delay associated with it as the inputs propagates through the outputs. Optimization is the main part while modelling hardware to reduce the propagation delay and to exploit parallelism and pipelining. The following techniques are followed in this work for hardware implementation of the image processing algorithms.

2.7.1 PARALLELISM

Exploiting the potential parallelism of the program and then run different non-conflicting operations at the same clock cycle to acquire speed up. On FPGA's by designing specific hardware many operations can be run in parallel, significant speed up can be obtain. This is the main reason why application on FPGA can sometimes run faster than the software version even though the FPGA hardware run at much slower clock speed.

2.7.2 PIPELINING

Pipelining is an implementation technique whereby multiple tasks are overlapped in execution. Ideally next task is started after every clock cycle. When the pipeline is full, the throughput will be a task per cycle in regardless of how many cycles it takes for the task to finish.

CHAPTER 3
EXISTING METHOD

A lot of edge detection algorithms, such as Robert detector, Prewitt detector, Kirsch detector, Gauss-Laplace detector and Canny detector have been proposed. Among these algorithms, Canny algorithm has been used widely in the field of image processing because of its good performance.

3.1 CANNY EDGE DETECTION ALGORITHM

The Canny edge detector is predominantly used in many real-world applications due to its ability to extract significant edges with good detection and good localization performance. Unfortunately, the Canny edge detection algorithm contains extensive pre-processing and post-processing steps and is more computationally complex than other edge detection algorithms, such as Roberts, Prewitt and Sobel algorithms. Furthermore, it performs hysteresis thresholding which requires computing high and low thresholds based on the entire image statistics. This places heavy requirements on memory and results in large latency hindering real-time implementation of the Canny edge detection algorithm.

The Canny edge detection algorithm is considered a "standard method" and is used by many researchers, because it provides very sharp and thin edges. The Canny operator works in a multi-stage process. Canny edge detection uses linear filtering with a Gaussian kernel to smooth noise and then computes the edge strength and direction for each pixel in the smoothed image. This is done by differentiating the image in two orthogonal directions and computing the gradient magnitude as the root sum of 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 done on the thinned edge magnitude image using hysteresis. In hysteresis, two edge strength thresholds are used. All candidate edge pixel 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 low threshold that can be connected to any pixel above the high threshold through a chain are labelled as edge pixels

Some approaches have been proposed for real-time edge detection. Alzahrani and Chen present an absolute different mask (ADM) edge detection algorithm and its pipelined VLSI architecture for real-time application. But the edge detector in offers a trade-off between precision, cost and speed, and its capability to detect edges is not as good as the Canny algorithm.

There is another set of work on Deriche filters that have been derived using Canny's criteria. For instance, it was stated in that a network with four transputers takes 6s to detect edges in a 256 × 256 image using the Canny- Deriche algorithm, far from the requirement for real-time applications.

The approach of operates on two rows of pixels at a time. This reduces the memory requirement at the expense of a decrease in the throughput. Furthermore, it is known that the original Canny edge detection algorithm needs two adaptive image-dependent high and low thresholds to remove false edges. However, the algorithm in just fixes high and low thresholds in order to overcome the dependency between the blocks, which results in decreased edge detection performance.

3.1.1 PRINCIPLE OF CANNY ALGORITHM

The canny algorithm consists of three criterion of the edge detection algorithm.

1) **The criterion of SNR**

The larger of the SNR, the higher quality of the detection edge. The SNR is defined as follow:

$$SNR = \frac{\left|\int_{-W}^{+W} G(-x)h(x)dx\right|}{\sigma\sqrt{\int_{-W}^{+W} h^2(x)dx}} \qquad (3.1)$$

Where, the $G(x)$ represents the edge function, $h(x)$ represents the impulse response of the filter of width is W. σ represents the mean square deviation of the Gaussian noise.

2) The criterion of positioning accuracy

The positioning accuracy of the edge is defined as follow:

$$L = \frac{\left|\int_{-W}^{+W} G'(-x)h'(-x)dx\right|}{\sigma\sqrt{\int_{-W}^{+W} h'^2(x)dx}} \quad (3.2)$$

Where the $G'(x)$ and $h'(x)$ respectively is the derivative of the $G(x)$ and $h(x)$, the larger of the positioning accuracy, the result is better.

3) The criterion of the singleness edge response

To ensure the edge only have one response, the average distance ($D(f')$) of the zero-crossing point of the derivative of the impulse response of the edge detection algorithm. The $D(f')$ should meet the follow formula:

$$D(f') = \pi\left\{\frac{\int_{-\infty}^{+\infty} h'^2(x)dx}{\int_{-W}^{+W} h'^2(x)dx}\right\}^{\frac{1}{2}} \quad (3.3)$$

Where, $h''(x)$ is the second derivative of $h(x)$.

3.2 ALGORITHM FLOW

Canny developed an approach to derive an optimal edge detector based on three criteria related to the detection performance. The model was based on a step edge corrupted by additive white Gaussian noise. A block diagram of the Canny edge detection algorithm is shown in Fig. 3.1.

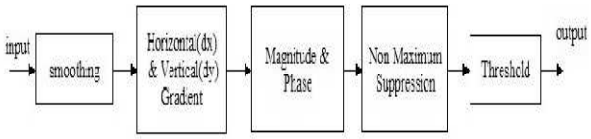

Figure 3.1**:** Schematic of canny edge detection

The original Canny algorithm consists of the following steps:

1. Smoothing the input image by Gaussian mask. The output smoothed image is denoted as I(x, y).
2. Calculating the horizontal gradient Gx(x, y) and vertical gradient Gy(x, y) at each pixel location by convolving the image I(x, y) with partial derivatives of a 2D Gaussian function.
3. Computing the gradient magnitude G(x, y) and direction θG(x, y) at each pixel location.
4. Applying non-maximum suppression (NMS) to thin edges.
5. Computing the hysteresis high and low thresholds based on the histogram of the magnitudes of the gradients of the entire image.
6. Performing hysteresis thresholding to determine the edge map.

3.2.1 SMOOTHING

In the first stage the 5x5 Gaussian convolution mask is used for smoothing. The effect of Gaussian convolution is to blur an image. The degree of smoothing is determined by the standard deviation of the Gaussian.

The Gaussian distribution in 1-D has the form:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}} \quad (3.4)$$

Where σ is the standard deviation of the distribution.

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

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \quad (3.5)$$

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 effectively zero more than about three standard deviations from the mean, and so convolution kernel is truncated. The convolution kernel of standard deviation(s) 1.4 is used for smoothing in this thesis as shown in Figure 3.2.

5x5 Gaussian Smooth Filter σ= 1.4

$\frac{1}{115}$

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2

Figure 3.2: 5x5 Gaussian convolution mask

The effect of Gaussian convolution is to blur an image. The degree of smoothing is determined by the standard deviation of the Gaussian.

3.2.2 GRADIENT CALCULATION

After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. Most edge detection methods work on the assumption that an edge occurs where there is a discontinuity in the intensity function or a very steep intensity gradient in the image as shown in Figure 3.3.

Most edge-detecting operators can be thought of as gradient-calculators. Because the gradient is a continuous-function concept and images are discrete functions, we have to approximate it. Since derivatives are linear and shift- invariant, gradient calculation is most often done using convolution. Numerous kernels have been proposed for finding edges, some of the kernels are: Roberts Kernel, Kirsch Compass Kernel, Prewitt Kernel, Sobel Kernel, and many others.

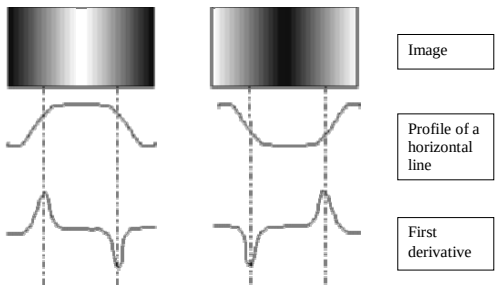

Figure 3.3: Gradient of image

The Prewitt kernels are based on the simple idea of the central difference between rows for horizontal gradient and difference between columns for vertical gradient

$$\frac{\partial I}{\partial x} \approx \frac{I(x+1, y) - I(x-1, y)}{2} \quad (3.6)$$

and

$$\frac{\partial I}{\partial y} \approx \frac{I(x, y+1) - I(x, y-1)}{2} \quad (3.7)$$

The following convolution masks are derived from equations.

0	0	0
-1	0	1
0	0	0

Fig 3.4 Horizontal Convolution

0	-1	0
0	0	0
0	1	0

Fig 3.5 Vertical Convolution

These convolutions are used for calculating the horizontal and vertical gradients.

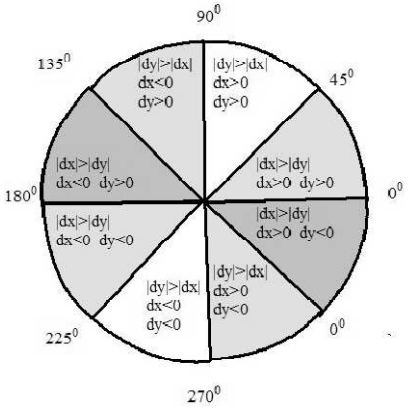

Figure 3.6 Gradient Orientation

3.2.4 NON-MAXIMUM SUPPRESSION

With the magnitude and direction obtained from previous stage one can apply the thresholding operation in the gradient-based method and end up with ridges of edge pixel. To get rid of ridges, 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 is called thinning process.

Once the direction of the gradient is known, the values of the pixels found in the neighbourhood 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 neighbours, along the direction of the gradient. For example, if the approximate direction of the gradient is between 00 and 450, the magnitude of the gradient at *Px,y* is compared with the magnitude of the gradient at adjacent points as shown in Figure 3.7.

3.2.3 MAGNITUDE AND PHASE

Convolution of the image with horizontal and vertical gradients produces horizontal gradient (dx) and vertical gradient (dy) respectively.

The absolute gradient magnitude (|G|) is calculated by the mean square root of the horizontal (dx) and vertical (dy) gradients.

That is,

$$|G| = \sqrt{(dx\text{^}2 + dy\text{^}2)} \quad (3.8)$$

To reduce the computational cost of magnitude, it is often approximated with absolute sum of the horizontal and vertical gradients

$$(|G| = |dx| + |dy|) \quad (3.9)$$

The direction of the gradient (ø) is calculated by arctangent of the vertical gradient to the horizontal gradient

$$\theta = \arctan(dy / dx) \quad (3.10)$$

Since arctangent is a very complex function and also requires floating point numbers, it is very difficult 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 Px,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 3.6.

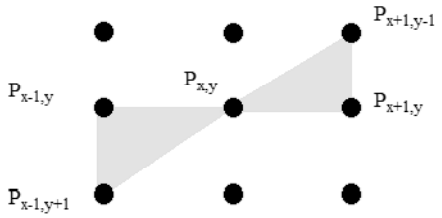

Figure 3.7 Pixel Interpolation

Where

$P_{x,y} = |dx_{x,y}| + |dy_{x,y}|$

The values of the gradient at the point Pa and Pb are defined as follows:

$$P_a = \frac{P_{x+1,y-1} + P_{x+1,y}}{2}, where\ P_{x+1,y-1} = |dx_{x+1,y-1}| + |dy_{x+1,y-1}|\ and\ P_{x+1,y} = |dx_{x+1,y}| + |dy_{x+1,y}| \quad (3.11)$$

$$P_b = \frac{P_{x-1,y+1} + P_{x,y+1}}{2}, where\ P_{x-1,y+1} = |dx_{x-1,y+1}| + |dy_{x-1,y+1}|\ and\ P_{x,y+1} = |dx_{x,y+1}| + |dy_{x,y+1}| \quad (3.12)$$

The centre pixel *Px,y* is considered as an edge , if *x, y a p > p* and *x, y b p > p* . If neither condition is not satisfied then the centre pixel is eliminated.

3.2.5 THRESHOLD

The output image of non-maximum suppression stage may consist of broken edge contours, single edge points which contribute to noise. This can be eliminated by thresholding with *hysteresis*. Two thresholds are considered for hysteresis, one high threshold other low threshold.

If any edge response is above a high threshold, those pixels constitute definite edge output of the detector for a particular scale. Individual weak responses usually correspond to noise, but if these points are connected to any of the pixels with high threshold, they are more likely to be actual edges in the image. Such connected pixels are treated as edge pixels if their response is above a low threshold.

To get thin edges two thresholds (high threshold (TH) and low threshold (TL)) are used. If the gradient of the edge pixel is above the TH, it is considered as an edge pixel. If the gradient of the edge pixel is below TL then it is unconditionally set to zero. If the gradient is between these two, then it is set to zero unless there is a path from this pixel to a pixel with a gradient above TH ; the path must be entirely through pixels with gradients of at least TL.

3.3 RESULT OF EXSITING ALGORITHM

Fig 3.8 Original Lena Image Fig 3.9 Canny Edge Detection Image

CHAPTER 4
DISTRIBUTED CANNY EDGE DETECTION ALGORITHM

4.1 ALGORITHM DESCRIPTION

The Canny edge detection algorithm operates on the whole image and has a latency that is proportional to the size of the image. While performing the original Canny algorithm at the block-level would speed up the operations, it would result in loss of significant edges in high-detailed regions and excessive edges in texture regions. Natural images consist of a mix of smooth regions, texture regions and high-detailed regions and such a mix of regions may not be available locally in every block of the entire image. We proposed a distributed Canny edge detection algorithm, which removes the inherent dependency between the various blocks so that the image can be divided into blocks and each block can be processed in parallel.

The input image is divided into m×m overlapping blocks. The adjacent blocks overlap by (L − 1)/2 pixels for a L × L gradient mask. However, for each block, only edges in the central n × n (where n = m + L − 1) non-overlapping region are included in the final edge map. Steps 1 to 4 and Step 6 of the distributed Canny algorithm are the same as in the original Canny algorithm except that these are now applied at the block level.

Step 5, which is the hysteresis high and low thresholds calculation, is modified to enable parallel processing. In [4], a parallel hysteresis thresholding algorithm was proposed based on the observation that a pixel with a gradient magnitude of 2, 4 and 6 corresponds to blurred edges, psycho visually significant edges and very sharp edges, respectively.

In order to compute the high and low hysteresis thresholds, very finely and uniformly quantized 64-bin gradient magnitude histograms are computed over overlapped blocks. If the 64-bin uniform discrete histogram is used for the high threshold calculation, this entails performing 64 multiplications and 64*Np comparisons, where Np is the total number of pixels in an image. Therefore, it is necessary to find a good way to reduce the complexity of the histogram computation.

It was observed that the largest peak in the gradient magnitude histograms after NMS of the Gaussian smoothed natural images occurs near the origin and corresponds to low-frequency content, while edge pixels form a series of smaller peaks where each peak corresponds to a class of edges having similar gradient magnitudes.

Consequently, the high threshold should be selected between the largest peak and the second largest edge peak. A sample gradient magnitude histogram is shown in Fig. 4.2 for the 512×512 Lena image (Fig.4.1). Based on the above observation, we propose a non-uniform quantizer to discretize the gradient magnitude histogram. Specifically, the quantizer needs to have more quantization levels in the region between the largest peak A and the second largest peak B and few quantization levels in other parts. Fig.4. 3 shows the schematic diagram of the designed quantizer. Accordingly, n reconstruction levels can be computed as follows:

$$R1 = (\min + \max)/2 \quad (4.1)$$

$$Ri+1 = (\min + Ri)/2, \quad i = 2, ..., n \quad (4.2)$$

Where min and max represent, respectively, the minimum and maximum values of the gradient magnitude after NMS, and Ri is the reconstruction level.

The proposed distributed thresholds selection algorithm is shown in below. Let Gt be the set of pixels with gradient magnitudes greater than a threshold t, and let NGt for t = 2, 4, 6, be the number of corresponding gradient elements in the set Gt. Using NGt, an intermediate classification threshold C is calculated to indicate whether the considered block is high-detailed, moderately edged, blurred or textured. Consequently, the set Gt = Gt=c can be selected for computing the high and low thresholds.

The high threshold is calculated based on the histogram of the set Gc such that 20% of the total pixels of the block would be identified as strong edges. The lower threshold is the 40% percentage of the higher threshold as in the original Canny algorithm. We compared the high threshold value that is calculated using the proposed distributed algorithm based on an 8-bin non-uniform gradient magnitude histogram with the value obtained when using a 16-bin non-uniform gradient magnitude histogram.

These two high thresholds have similar values. Therefore, we use the 8-bin non-uniform gradient magnitude histogram in our implementation.

Original Image

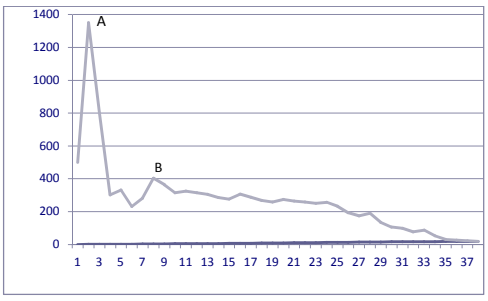

Fig 4.1 Histogram of the gradient magnitude after non-maximal suppression of the Lena image.

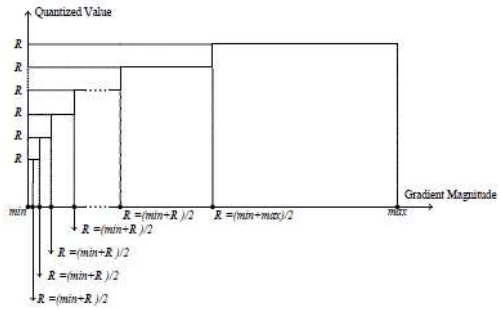

Fig 4.2 Reconstruction values and quantization levels; min and max represent, respectively, the minimum and maximum values of the gradient magnitude after NMS

4.2 THRESHOLD SELECTION SCHEME

Pseudo-code for the proposed distributed threshold selection scheme is given as

Let Gt: set of pixels with gradient magnitude greater than a threshold t.

N_{G4}: number of elements in the set Gt

Step 1: Determine G*t* for *t* = 2, 4, and 6 and N_{Gt} for *t* = 2, 4.

Step 2: If (N_{G4} > 0.25*Total_block_pels)

C = 6 /* *High-detailed* */

Else if ((N_{G4} > 0.05*Total_block_pels)

C = 4 /* *moderately-edged* */

Else if ((N_{G2} < 0.25*Total_block_pels)

C= 2 /**blurred* */

Else

Exit; /* *textured* */

Step 3: Compute the normalized histogram of G*t*=*C* and the corresponding cumulative distribution function *F*(G*t*=*C*).

Step 4: Compute High threshold as *F*(High threshold) = 0.8

Step 5: Compute Low threshold = 0.4*High threshold

CHAPTER 5
HARDWARE IMPLEMENTATION

In this section, we describe the hardware implementation of our proposed distributed Canny edge detection algorithm on the Xilinx Spartan 2E FPGA (XC5VSX240T).

First the implementation of the moving window operator which form the basis of the proposed algorithm is explained

5.1 MOVING WINDOW OPERATOR

The algorithms implemented in this work use the moving window operator. The moving window operator usually process 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. In this section a 3x3 moving window used for the median filtering, morphological and edge detection algorithms and a 5x5 moving window used in Gaussian smoothing filter operation are explained.

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 a controller that manages the traffic of data to and from the FIFO. The FIFO's are implemented using circular buffers constructed from multiport block RAM with an index keeping track of the front item in the buffer. The availability of multi-port block RAM in the Xilinx Vertex-E FPGA helps in achieving the read and write operations of the RAM in the same clock cycle. This allows a throughput of one pixel per clock cycle. The same effect can be achieved using double-width RAMs implemented in lookup tables on the FPGA. However, the use of block RAMs is more efficient and has less associated logic for reading and writing.

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 5.1 shows the architecture of the 3x3 moving window. For every clock cycle, a pixel is read from the RAM and placed into the bottom left corner location of the window.

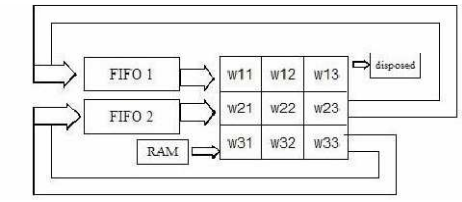

Fig 5.1 Architecture of 3x3 moving window

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 disposed after the computation on the pixels is completed, since it is not used in future computation.

Similarly for a 5x5 window operation four FIFO buffers are used, Each FIFO size is W-5, where W is width of the image. To access the values in the moving window in every clock cycle the four FIFO buffers must be full. Figure 5.2 shows the architecture of 5x5 moving window. For every clock cycle, a pixel read from the RAM is placed into the bottom left corner location of the window. 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 disposed after the computation of the pixels is computed.

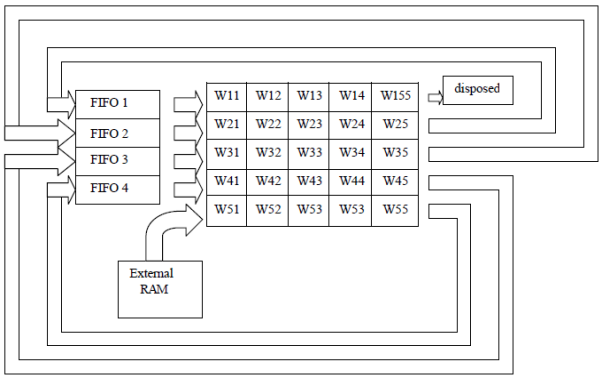

Figure 5.2: Architecture of 5x5 moving window

5.2. ARCHITECTURE OVERVIEW

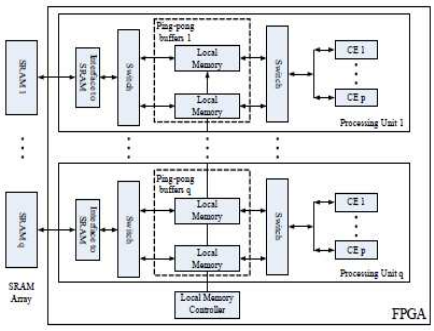

Fig 5.3 The architecture of the proposed distributed Canny algorithm

Depending on the available FPGA resources, the image needs to be partitioned into q sub-images and each sub-image is further divided into p m×m blocks. The proposed architecture, shown in Fig. 5.3, consists of q processing units in the FPGA and some Static RAMs (SRAM) organized into q memory banks to store the image data, where q equals to the image size divided by the SRAM size

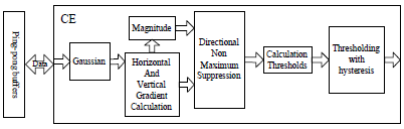

Fig. 5.4. Block diagram of the CE for the proposed distributed Canny edge detection

Each processing unit processes a sub-image and reads/writes data from/to the SRAM through ping-pong buffers, which are implemented with dual port Block RAMs (BRAM) on the FPGA. As shown in Fig. 5.4, each processing unit (PU) consists of p computing engines (CE), where each CE detects the edge map of an m×m block image. Thus, p · q blocks can be processed at the same time and the processing time for an N × N image is reduced, in the best case, by a factor of p · q.

The specific values of p and q depend on the processing time of each PE, the data loading time from the SRAM to the local memory and the interface between FPGA and SRAM, such as total pins on the FPGA, the data bus width, the address bus width and the maximum system clock of the SRAM. In our application, we choose p = 2 and q = 8. In the proposed architecture, each CE consists of the following 6 units, as shown in Fig. 5.4

1. Smoothening unit using Gaussian filter.
2. Vertical and horizontal gradient calculation unit.
3. Magnitude calculation unit.
4. Directional non-maximum suppression unit.
5. High and low threshold Calculation unit.
6. Thresholding with hysteresis unit.

Classically, the image smoothing is implemented by applying the Gaussian convolution on the entire image. Furthermore, the smoothened image is used as an input to calculate the gradient at every pixel, and these gradient values are used to calculate the phase and the magnitude for each pixel, which is followed by non-maximum suppression.

To get rid of ridges, 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 is called non-maximum suppression. Two threshold (High threshold & Low threshold) values are used get the connected edge pixels. This is called hysteresis.

On a general purpose computer the four stages of the canny edge detection are performed sequentially on the entire image, one stage followed by other stage. This approach on FPGA requires lot of hardware resources and design is slow. In order to efficiently use the hardware resources and increase the speed, hardware features like parallelism and pipelining are employed. Since output in each stage depend on the neighbouring pixels, a moving window operator discussed.

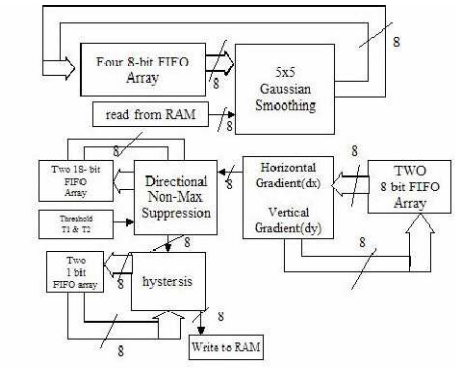

Fig 5.5 Pipelined Architecture

5.3. IMAGE SMOOTHENING

Smoothing of the image is achieved by 3x3 Gaussian convolution. 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 2 FIFO buffers are full. That is, the output is produced after a latency of twice width of image plus two (2*width +2) cycles. The output of this stage is given as input to the horizontal and vertical gradient calculation stage.

p^2	pq	p^2
pq	q^2	pq
p^2	pq	p^2

Fig 5.6 Mask for the low pass Gaussian filter

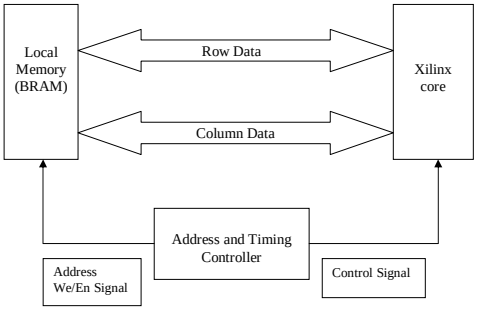

Fig 5.7 Pipelined Image Smoothening Unit.

The input image is smoothened using a 3×3 Gaussian mask, as shown in Fig. 5.6. The Gaussian filter (Fig. 5.6 is separable and, thus, the implementation of the 2-D convolution with the 3×3 Gaussian mask is achieved using row and column 1- D convolutions.

The proposed architecture for the smoothening unit is shown in Fig. 5.7 The main components of the architecture consist of a 1-D finite impulse filter (FIR) to process the data and the on-chip Block RAM (BRAM) to store the data. In our design, we adopt the Xilinx's pipelined FIR IP core, which provides a highly parameterizable, area-efficient, high-performance FIR filter utilizing the structure characteristics in the coefficient set, such as symmetry and conjungacy.

By exploiting the symmetry of the Gaussian filter, the architecture uses two multipliers to perform the 1-D convolution using a 3-tap filter. The address controller fetches the input image data from the local memory into the FIR core and, after the computation, it stores the results back in the BRAM.

5.4. GRADIENTS AND GRADIENT MAGNITUDE CALCULATION

This stage calculates the vertical and horizontal gradients using 3x3 convolution kernels shown in Figure 5.8. 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, neighbouring eight pixels are required to calculate the gradient of the centre pixel and the output pixel produced in previous stage is a pixel in row order. In order to access eight neighbouring pixels in a single clock cycle, two FIFO buffers are employed to store the output pixels of the previous stage.

This stage calculates the vertical and horizontal gradients using convolution kernels. The kernels vary in size from 3 × 3 to 9×9, depending on the sharpness of the image.

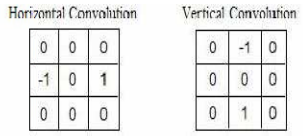

Figure 5.8 Gradient Convolution Kernels

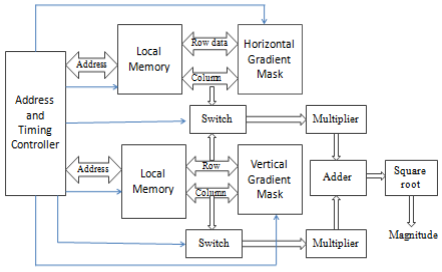

Fig 5.9 Gradient and Magnitude Calculation Unit.

The Xilinx FIR IP core, which can support up to 256 sets of coefficients with 2 to 1024 coefficients per set, is used to implement the kernels. The whole design is pipelined, and thus the output is generated every clock cycle. This is input to the magnitude calculation unit which computes, at each pixel location, the gradient magnitude from the pixel's horizontal and vertical gradients.

Canny suggests this is done by first applying a Gaussian blur to smooth out possible noise, and afterwards finding gradients using a mask operation, where the mask is calculated from the first derivative of the Gaussian in both directions. Since we have already implemented mask operations with 3x3 and 5x5 masks, both the Gaussian blur and its first

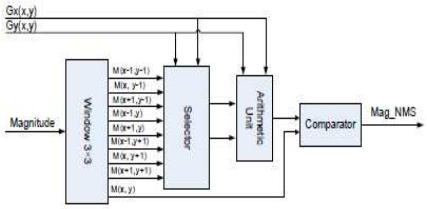

Fig 5.10 Directional Non Maximum Suppression Unit

The second phase is minimizing false edges - also known as non max suppression. This is done by only approving pixels, whose gradients are local extremes in the direction of the gradient. Furthermore a hysteresis threshold is performed. To test if the gradient is a local extreme along the direction of the gradient, one has to know this direction and the sizes of the gradients of both the actual pixel and also the ones along the gradient direction. The size can easily be found from equation 5.3.1 and so can the direction from equation 5.3.2, but instead of the ArcTan we use the comparison network from figure 5.11.

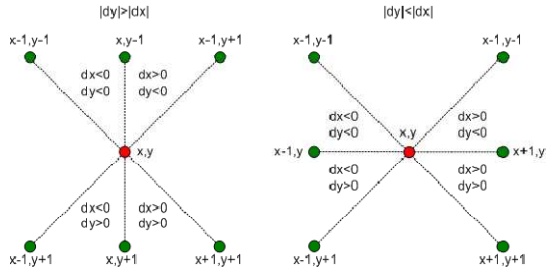

Figure 5.11. Determination of a gradient direction can be done using this comparison network.

derivative in both directions could easily be implemented using these. The equations for the 2D Gaussian filter and its derivatives can be seen from appendix F, together with discrete versions of 5x5 masks.

The size and direction of the resulting gradient can both be found from the gradients in the x- and y-direction. The ArcTan is often used by mathematicians when describing the direction, but in implementations it is very inefficient.

.

$$|G| = \sqrt{G_X^2 + G_Y^2} \qquad (5.3.1)$$

$$\theta = ArcTan\left(\frac{G_Y}{G_X}\right) \qquad (5.3.2)$$

The architecture of this unit is shown in Fig. 5.9. The gradient calculation architecture consists of two 1-D FIR models and the corresponding local memory. The filters for computing the horizontal and vertical gradient elements can process data in parallel. The magnitude computation consists of two multipliers and one square-root computation module, which are implemented by the Xilinx Math Function IP cores.

5.5 DIRECTIONAL NON MAXIMUM SUPPRESSION

Fig. 5.10 shows the architecture of the directional non-maximum suppression unit. In order to access all the pixels' gradient magnitudes in the 3 × 3 window at the same time, two FIFO buffers are employed. The horizontal gradient Gx and t vertical gradient Gy control the selector which delivers the gradient magnitude of neighbours along the direction of the gradient, into the arithmetic unit.

This arithmetic unit consists of one divider, two multipliers and one adder, which are implemented using the Xilinx Math Functions IP cores. The output of the arithmetic unit is compared with the gradient magnitude of the centre pixel, and the pixel that has no local maximum gradient magnitude is eliminated.

The comparisons are based on the values of the gradients in the x- and y-directions.

From figure 5.11 it is seen how the values of the gradients in the x- and y-direction for the centre pixel are used to determine an area between two neighbours. Since the direction of a gradient often is between two actual neighbours, we are interested in the interpolated gradient from those two. Knowing the size of the gradients for the two neighbours, it is possible to calculate the size of the interpolated gradient from the actual direction or position between neighbours. We do not have the actual direction, because we did not want to use ArcTan, but the position can be found from the relationship between the x- and y-gradients.

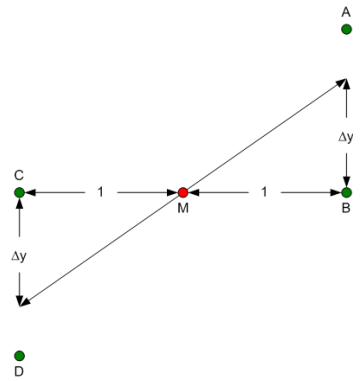

Figure 5.12: Drawing to help figure out the relationship between Δy and the gradients in the x- and y-directions.

Figure 5.12. helps to illustrate this. In this example we have chosen that |dy| < |dx|, dx > 0 and dy < 0. If we say that the distance from our centre-pixel (M) to the right-hand neighbour (B) is 1, then the distance (Δy) from (B) to the position we need can be calculated from (5.4.1). The distance from (B) to the upper-right neighbour (A) is also 1.

$$\Delta y = \frac{dy}{dx} \qquad (5.4.1)$$

When the position is known, the interpolated gradient size can be calculated from equation 5.4.2.

$$|G_{AB}| = y \cdot |G_B| + (1 - \Delta y) \cdot |G_A| \qquad (5.4.2)$$

Since the distance Δy is the same in both directions, equation 5.4.1 can also be used for finding the interpolated gradient size between neighbours (C) and (D). In the case where |dy| > |dx|, the same principle is applied, but the neighbours are different according to figure 4.28, and Δx should be used instead of Δy in equation 5.4.1 along with the new neighbours. Equation 5.4.3 shows the calculation of Δx.

$$\Delta x = \frac{dx}{dy} \qquad (5.4.3)$$

If the gradient size of the centre pixel is larger than the interpolated values on both sides along the direction, the pixel qualifies for the following hysteresis-threshold. The concept of hysteresis threshold is to compare the gradient-size to two different threshold-values. A low threshold that divides pixels in two groups - non-edges and possible edges. A high threshold that distinguishes between possible edges and positive edges. A result from this phase can be seen from picture 5.13.

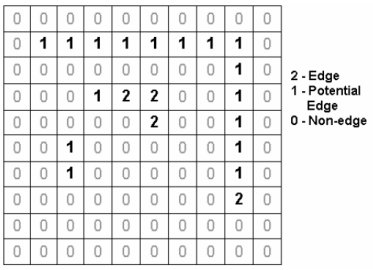

Figure 5.13. Result of the non max suppression phase

unit will compare this difference with 0. If the difference is 0, a stop signal will be generated to stop the accumulating of accumulator 2. The value of accumulator 2 is then the high threshold *Thh*, and the low threshold *Thl* will be set the value of the half of *Thh*. The pipelined design is shown in Fig. 5.14.

5.7 THRESHOLDING WITH HYSTERESIS

Since the output of the non maximum suppression unit contains some spurious edges, the method of thresholding with hysteresis is used. Two thresholds, high threshold ThH and low threshold ThL, which are obtained from the threshold calculation unit, are employed. Let f(x, y) be the image obtained from the non maximum suppression stage, f1(x, y) be the strong edge image and f2(x, y) be the weak edge image. Fig.5.15 illustrates the pipelined design of this stage.

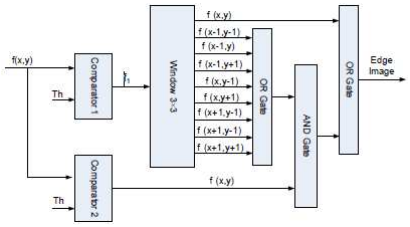

Fig 5.15 Pipelined architecture of the Thresholding Unit.

Instead of implementing each stage separately, the above mentioned pipelined design utilizes the resources efficiently on the FPGA and also increase the execution speed of the algorithms. Because the output is produced every clock cycle with an initial latency till the pipe is full.

5.6 CALCULATION OF THE HYSTERESIS THRESHOLDS

Since the low and high thresholds are calculated based on the gradient histogram, we need to compute the histogram of the image after it has undergone directional non-maximum suppression. An 8-step non-uniform quantizer is employed to obtain the discrete histogram for each processed block. The block-based hysteresis thresholds (high threshold ThH and low threshold ThL) are computed.

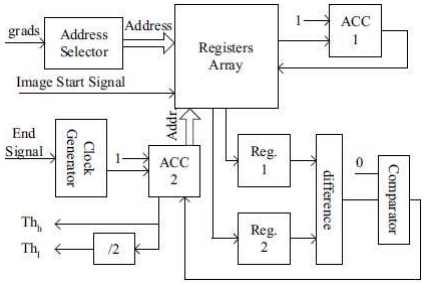

Fig 5.14. Design of threshold calculation

Since the threshold is calculated based on the gradient histogram, we need the histogram of the image after the NMS operating. Taking a 360x280x8bit gray image for instance, the magnitude of each gradient is always between 0 to 100, so we need a registers array containing 100 12bit-width registers to store all of the pixels with different gradient magnitude.

A pixel with certain gradient magnitude will arrive at every clock. Taking its magnitude as the address of the registers array, the content of this register is then accumulated by 1. When the end signal arrives, the clock generator circuit generates 100 clock cycles. During these cycles, the content of accumulator 2 which is accumulated by 1 in each clock is set as the address of the registers array, and then the difference circuit calculates the difference between the contents of this register and the next address register. A comparing

CHAPTER 6
RESULTS AND COMPARISION

6.1. MATLAB SIMULATION RESULTS

We conducted the following experiments to investigate the effectiveness of the new distributed Canny edge detector that is proposed in this paper. The detection effectiveness of the proposed distributed approach is analyzed by comparing the perceptual significance of its resulting map with the one produced by the original frame-based Canny edge detector. Fig.6.2 and 6.7 shows the edge maps that are obtained for the 512×512 Lena image using the original Canny edge detector and the proposed algorithm with a 3 × 3 gradient mask and block size of 64. The floating-point Matlab simulation results show that, compared with the traditional Canny edge detector and the algorithm proposed here can yield better edge detection results with few computations. Similar results were obtained for other tested images.

Fig 6.1 Original Lena image

Fig 6.2 Edge map conventional Canny edge detector

6.1.1 PROPOSED METHOD RESULTS

Fig 6.3 Derivative in X direction

Fig 6.4 Derivative in Y direction

Fig 6.5 Gradient

Fig 6.6 After Non-Maximal Suppression

Fig 6.7 Edge map of proposed method

6.2 FPGA SIMULATION RESULTS

The FPGA result is obtained using Modelsim. The output is synthesized using Xilinx ISE Simulator

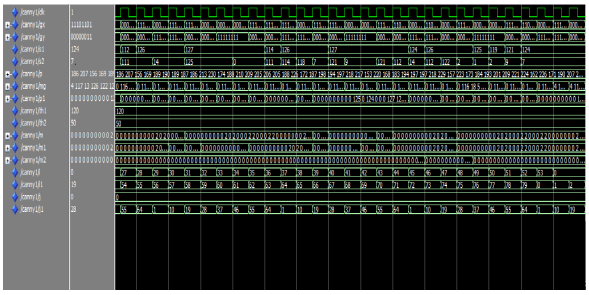

Fig 6.8 Modelsim simulation result

6.2.1 SYNTHESIS REPORT

Device used:

SPARTAN2E-XC2S400E-FG456

Timing Summary:

- Speed Grade: -7
- Minimum period: 22.804ns (Maximum Frequency: 43.78MHz)
- Maximum output required time after clock: 6.436ns

Design Summary:

Logic Utilization:

- Total Number Slice Registers : 742 out of 9600 7%
- Number used as Flip Flops : 728
- Number used as Latches : 14
- Number of 4 input LUTs : 4,083 out of 9600 42%

Logic Distribution:

- Number of occupied Slices : 2,499 out of 4800 52%
- Number of Slices containing only related logic : 2,516 out of 2,516 100%
- Number of Slices containing unrelated logic : 0 out of 2,516 0%

- Total Number 4 input LUTs : 4,185 out of 9600 43%
- Number of bonded IOBs : 81 out of 329 24%
- Number of GCLKs : 1 out of 4 25%
- Number of GCLKIOBs : 1 out of 4 25%
- Total equivalent gate count for design : 41,054
- Additional JTAG gate count for IOBs : 3,686

6.3 COMPARISION

Algorithm/Parameters	Conventional Canny Edge Detection Algorithm	Distributed Canny Edge Detection Algorithm
Minimum Period Required For Input Before Clock Cycle	30.458ns	22.804ns
Maximum Frequency	38.64MHz	43.78MHz
Number Of Occupied Slices	55%	52%
Total Number Of 4 Input LUTs	47%	43%

Table 6.1 Performance Comparison between Conventional and Proposed Method

CHAPTER 7
ADVANTAGES AND APPLICATIONS

ADVANTAGES

- Sharp edges can be detected
- Reduced memory requirements
- Reduced computational time
- Reduced loss in edge detection

APPLICATIONS

- Various image enhancement process
- Authentication purposes
- Robotic vision system
- Medical applications

CHAPTER 8
CONCLUSION

A novel distributed Canny edge detection algorithm that results in a significant speed up without sacrificing the edge detection performance has been presented. The Matlab simulation result shows that the proposed algorithm yields better edge detection result than the traditional canny edge detector. A novel non-uniform quantized histogram calculation method in order to reduce the computational cost of the hysteresis threshold selection is also presented. As a result, the computational cost of the proposed algorithm is very low compared to the original Canny edge detection algorithm. The algorithm has been mapped to onto a Xilinx Spartan-2E FPGA platform and tested using Modelsim.

REFERENCES

1. S. Varadarajan, C. Chakrabarti, L. J. Karam, and J. M. Bauza, "A distributed psycho-visually motivated Canny edge detector," *IEEE ICASSP, pp. 822 –825, Mar. 2010.*
2. Bing Wang, ShaoSheng Fan, "An improved CANNY edge detection algorithm", Second International Workshop on Computer Science and Engineering,pp. 497-500, 2009.
3. W. He and K. Yuan, "An improved Canny edge detector and its realization on FPGA," *WCICA, pp. 6561 –6564,* Jun. 2008.
4. Alirezae Rezaee. Extracting Edge of Images with Ant Colony, In Journal of Electrical Engineering, Vol. 59, 1, pp.57-59, 2008.
5. Pan Dafu, Wang Bo. An Improved Canny Algorithm. Proceedings of the 27th Chinese Control Conference 2008, 456-459.
6. Guo Xing, Wenya Qi, Ping Li, Ji'an Chen. Self-adapting edge detection of gray image [J]. Computer engineering and application. 2007, 43(5):63-66.
7. Guo Xing, Wenya Qi, Ping Li, Ji'an Chen. Self-adapting edge detection of gray image [J]. Computer engineering and application. 2007, 43(5):63-66.
8. Hocenski Zeljko, Vasilic Suzana and Hocenski Verica "Improved Canny Edge Detector in Ceramic Tiles Defect Detection," *IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference , pp.* 3328-3331, November 2006.
9. D. V. Rao and M. Venkatesan, "An efficient reconfigurable architecture and implementation of edge detection algorithm using Handle-C," *ITCC*, vol. 2, pp. 843 – 847, Apr. 2004.
10. H. J. Trussell: Comments on "Picture Thresholding Using an Iterative Selection Method", IEEE Transactions on System, Man, and Cybernetics, Vol. SMC-9, No. 5, May 1999.
11. J. Canny, "A computational approach to edge detection," *IEEE Trans. PAMI*, vol. 8, no. 6, pp. 679 –698, Nov. 1986.
12. Advanced Edge Detection Technique: Techniques in Computational Vision: http://www.cpsc.ucalgary.ca/Research/vision/501/edge detect.pdf.