IMAGE PROCESSING ALGORITHMS ON FPGA

By

MOHAMED NASIR BIN MOHAMED SHUKOR

FINAL PROJECT REPORT

Submitted to the Electrical & Electronics Engineering Programme
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Universiti Teknologi Petronas
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

IMAGE PROCESSING ALGORITHMS ON FPGA

by

Mohamed Nasir Bin Mohamed Shukor

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Approved:

Mr. Lo Hai Hiung,
Project Supervisor

Lo Hai Hiung
Lecturer
Electrical & Electronic Engineering
Universiti Teknologi PETRONAS
Bandar Seri Iskandar, 31750 Tronoh
Perak Darul Ridzuan, Malaysia

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

June 2007

CERTIFICATION OF APPROVAL

IMAGE PROCESSING ALGORITHMS ON FPGA

by

Mohamed Nasir Bin Mohamed Shukor

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Approved:

Mr. Lo Hai Hiung
Project Supervisor

Lo Hai Hiung
Lecturer
Electrical & Electronic Engineering
Universiti Teknologi PETRONAS
Bandar Seri Iskandar, 31750 Tronoh
Perak Darul Ridzuan, Malaysia

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

June 2007

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

Mohamed Nasir Bin Mohamed Shukor

ABSTRACT

The objective of this project is to construct a real time hardware image processing system which based on Field Programmable Gate Array (FPGA). The chosen image processing algorithms are implemented on two systems which are Color Filtering System and Edge Detection System. This project utilizes Altera DE2 development board empowered by Cyclone II FGPA pair with 1.3 Mega pixel CMOS camera from Terasik Technologies. Verilog HDL is chosen as the hardware programming language for these systems and its compiled using Quartus II program. In order to verify the functionality of the hardware systems, Matlab 7.1 (as an engineering tool) is use to simulate the expected output for both system.

ACKNOWLEDGEMENTS

Special thanks to my beloved parents for their enduring and loyal support during the course of this Final Year Project (FYP).

I wish a big appreciation for my Supervisors, Mr. Lo Hai Hiung and Mr. Patrick Sebastian for the invaluable support and guidance which directly influence the success of this project.

Last but not least, I wish thank you for all who involved either directly or indirectly helping me in completing the project.

TABLE OF CONTENTS

LIST OF TABLES ix

LIST OF FIGURES x

CHAPTER 1 INTRODUCTION 1

1.1 Background Study 1

1.2 Problem Statement 2

1.2.1 Problem Identification 2

1.2.2 Significance of the Project 2

1.3 Objective and Scope of Study 3

1.3.1 Software Design 3

1.3.2 Hardware Design 3

CHAPTER 2 LITERATURE REVIEW/ THEORY 4

2.1 Digital Grayscale Image 4

2.2 Digital Color Image 5

2.3 Color Filtering Process 6

2.4 Edge Detection Method 7

2.4.1 Edge Detection Using Intensity Differential Method 8

2.4.2 Edge Detection Using Sobel Method 9

CHAPTER 3 METHODOLOGY/PROJECT WORK 10

3.1 Procedures Identification 10

3.2 Tools Required 11

3.2.1 DE2 Board (FPGA Development Board) and TRDB_DC2 Camera 11

3.2.2 Software Required 11

CHAPTER 4 RESULT AND DISCUSSION 12

4.1 Color Filtering System via Matlab 12

4.2 Color Filtering System via Verilog 16

4.2.1 Pixel Filtering Process 16

4.2.2 Color Filtering System 18

4.2.3 Pixel Counting Process 19

4.3 Edge Detection Process via Matlab 20

4.4 Edge Detection Process via Verilog 27

CHAPTER 5 CONCLUSION AND RECOMMENDATION 32
REFERENCES 33
APPENDICES 34
Appendix A Sobel EDGE DETECTION METHOD 35
Appendix B intensity different edge detection method 37
Appendix C Edge detection via verilog 39
Appendix D Color filtering system via matlab 45
Appendix E Color filtering system via verilog 46

LIST OF TABLES

Table 1 : 3 x 3 Convolution Masking 9
Table 2 : Working Procedures 10
Table 3 : Three Dimension array matrix 12
Table 4 : Matrix Size for Saved Data 13
Table 5 : Matrix representation of each layer 13
Table 6 : Flowchart on determined the horizontal edge 22
Table 7 : Flowchart on determined the vertical edge 24
Table 8 : Flowchart on determined the combined edge 26
Table 9 : Flowchart of system operation 31
Table 10 : Advantage and Disadvantage for Intensity Different Edge Detection Method 31

LIST OF FIGURES

Figure 1 : Discreet Image Depth [1] 4
Figure 2 : RGB Color Model [1] 5
Figure 3 : Block RGB Color Model 6
Figure 4 : Edge Detection Using Intensity Differential Method 8
Figure 5 : Sobel Filtering Method 9
Figure 6 : TRBD_DC2 and DE2 Development Board 11
Figure 7 : Grayscale representation of each layer 13
Figure 8 : Input Image and its Corresponding Output 15
Figure 9 : Single Band Pass Filte 17
Figure 10 : Double Bands Pass Filters 17
Figure 11 : Comparison for Single and Double Band Pass Filter 18
Figure 12 : Snapshot of Color Filtering System 18
Figure 13 : 7-SEG Display on Altera DE2 board 19
Figure 14 : The Grayscale input 20
Figure 15 : The BW image (after conversion) 20
Figure 16 : Horizontal Edge Detection 21
Figure 17 : Vertical Edge Detection 23
Figure 18 : Output of combined image from vertical and horizontal edge 25
Figure 19 : Output of Edge Detection via Verilog Program 27
Figure 20 : example of image conversion 28
Figure 21 : The blue box as indicator for 50 x 50 matrix 29
Figure 22 : Example of horizontal edge detection 30

CHAPTER 1
INTRODUCTION

1.1 Background Study

Recently, Field Programmable Gate Array (FPGA) technology has become an alternative for the implementation of software algorithms. The unique structure of the FPGA has allowed the technology to be used in many applications from video surveillance to medical imaging applications [1].

Field Programmable Gate Array (FPGA) is a large-scale integrated circuit that can be re-programmed. The term "field programmable" refers to ability of changing the operation of the device. While "gate array" refers to the basic internal architecture that makes re-programming is possible [11].

1.2 Problem Statement

1.2.1 Problem Identification

This project based on basic color image processing system and image edge detection system. Matlab program is utilized to study and develop image processing algorithms. Then equivalent hardware algorithms are developed in order for implementation on FPGA.

1.2.2 Significance of the Project

Mapping of image processing algorithms on FPGA will give direction of actually implementing software algorithms on FPGA. From the viewpoint of cost and flexibility, FPGA technology is becoming an alternative in design and development of computer intensive system. It is flexible in a sense of customization capability and faster development time.

1.3 Objective and Scope of Study

The main objective of this research project is to develop a real time video processing hardware system based on FPGA. The scope of study for this project is divided into two, software design and hardware design.

1.3.1 Software Design

Matlab is used to implement and simulate the image processing algorithms. First, the algorithms will be tested on an image (instead of video) to verify the algorithms processing capabilities

1.3.2 Hardware Design

Verilog HDL is used as the design file and compiled using Quartus II. Verilog HDL is based on Matlab algorithms and able to give equivalent output as the Matlab program.

CHAPTER 2

LITERATURE REVIEW/ THEORY

2.1 Digital Grayscale Image

From a grayscale image, it can be sample and re-presented as a matrix representation of m x n. A pixel at (x,y) on the matrix is a byte of data from the lowest 0000 0000 = 0(which known as black) to highest 1111 1111 = 255(which known as white). Its means, a pixel for of grayscale image used image depth of 256 levels from 0 to 255[1].

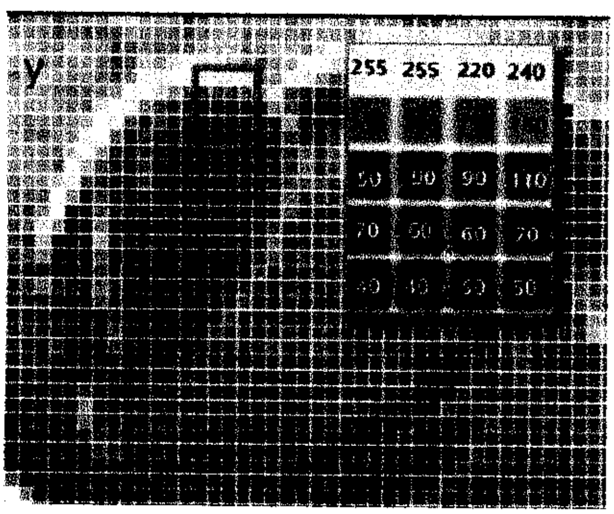


Figure 1 : Discreet Image Depth [1]

2.2 Digital Color Image

Digital color image is described using RGB color model. This color model consists of three channels: red, blue and green. Each channel is obtained by sampling the lightness in the specific spectrum (same operation as digital grayscale). A quantization level of 256 for each color will enables representation of 16 777 216 unique color, where {255,255,255} represent white and {0,0,0} represent black[1].

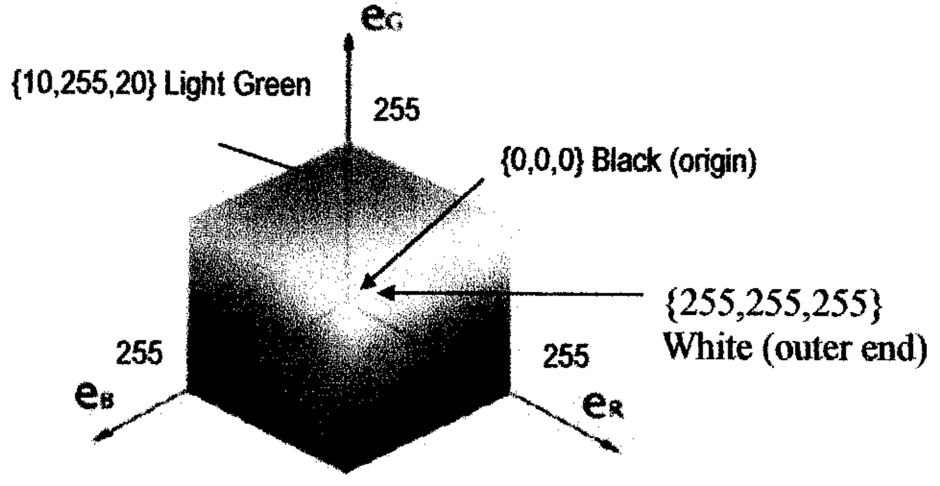
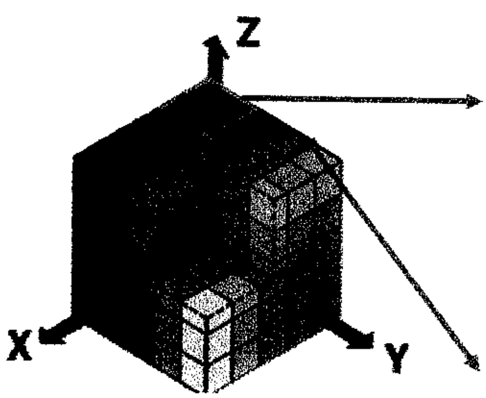


Figure 2 : RGB Color Model [1]

2.3 Color Filtering Process

Color filtering system for this project will selectively reduce the amount of color by allowing only "blue" color to pass through the filter. This idea can be explained by block RGB color model. When a pixel is determined and the color of the pixel is within the "blue" block, then this pixel will retain its color. Otherwise, the other color will be filter out. "Blue" color is determined using the standard from Kevin J. Walsh [13].



	Red	Green	Blue	Sample
Blue1	0	0	139	
Blue2	0	0	255	
Royalblue1	65	105	225	
Royalblue2	72	118	255	
Royalblue3	58	95	205	
Royalblue4	39	64	139	

	Red	Green	Blue	Sample
DeepSkyBlue1	0	191	255	
DeepSkyBlue2	0	178	238	
DeepSkyBlue3	0	154	205	
DodgerBlue1	30	144	255	
DodgerBlue2	28	134	238	
DodgerBlue3	24	116	205	

Figure 3 : Block RGB Color Model

2.4 Edge Detection Method

One of the popular image processing operations is the edge detection. The goal of edge detection is to mark the point of the digital image which the luminous intensity is changed drastically. By introducing edge detection, the processing image data will reduce significantly where the only left is the relevant data.

In the real time processing system, the edge detection operation is made by comparing two input data from an image rather than one. Its means, the first and second input pixel will be compared at the same time. In this type of operation, the mathematical operation such 2-D convolution is performed in order to get the output from the input data (image) [12].

2.4.1 Edge Detection Using Intensity Differential Method

This method used pixel comparing technique. First a grayscale image will be converted to a Black and White image by introducing a threshold value. Then from Black and White image, the matrix representation of it will undergo the second stage processing. Each pixel will be scan from rows to rows, columns to columns and at the same time, the value of each pixel is compared will its previous pixel. Any value change of each pixel (from "1=white" to "0=black" or vice versa) will be marked and finally create an edge representation for the image.

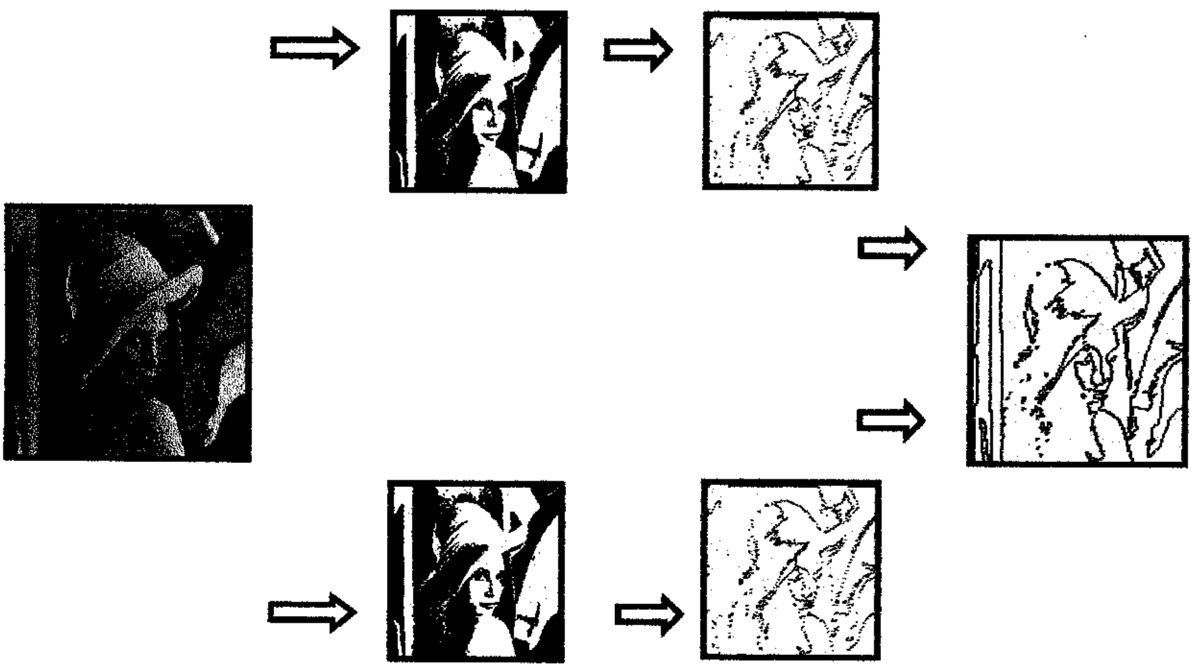


Figure 4 : Edge Detection Using Intensity Differential Method

2.4.2 Edge Detection Using Sobel Method

Sobel method [2] consists of 2-D convolution of an image. This convolution is divided to two sections, Vertical Convolution and Horizontal Convolution. Both convolutions will imply different 3x3 convolution mask. Finally, by using mathematical calculation both convolution output are merged and resulting an edge representation of original image.

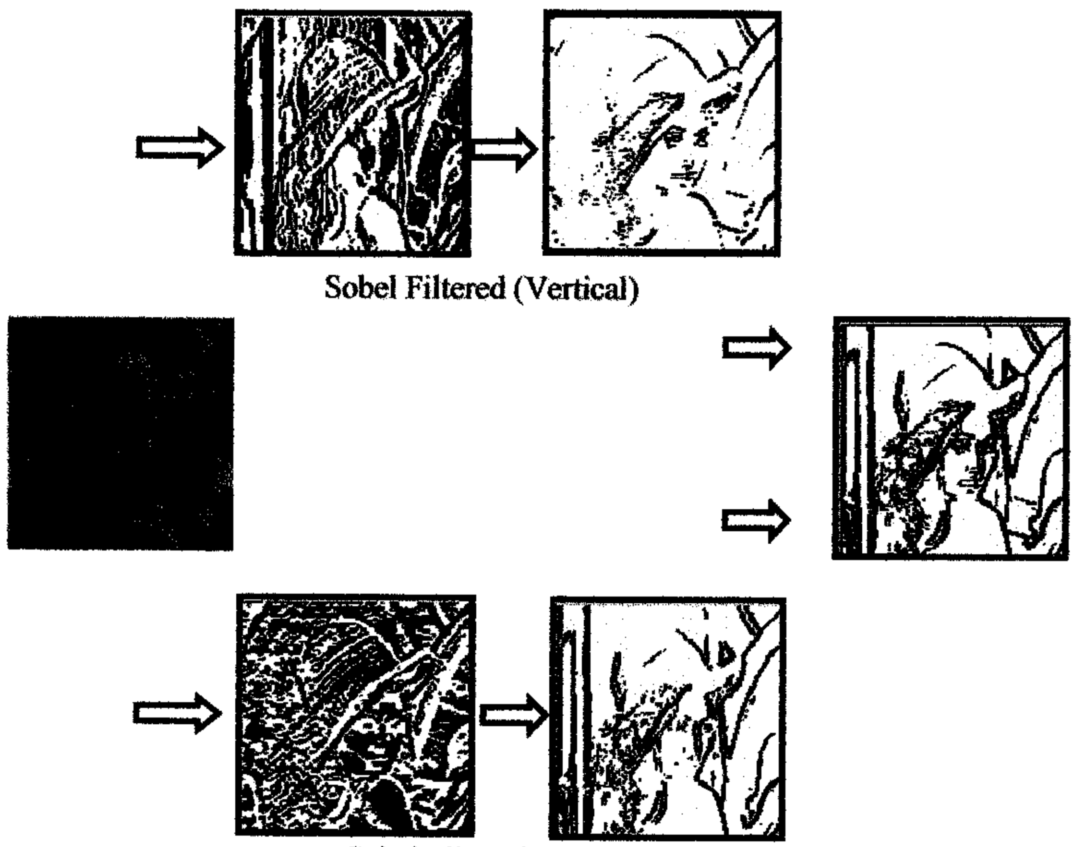


Figure 5 : Sobel Filtering Method

Table 1 : 3 x 3 Convolution Masking

Vertical Mask

+1	+2	+1
0	0	0
-1	-2	-1

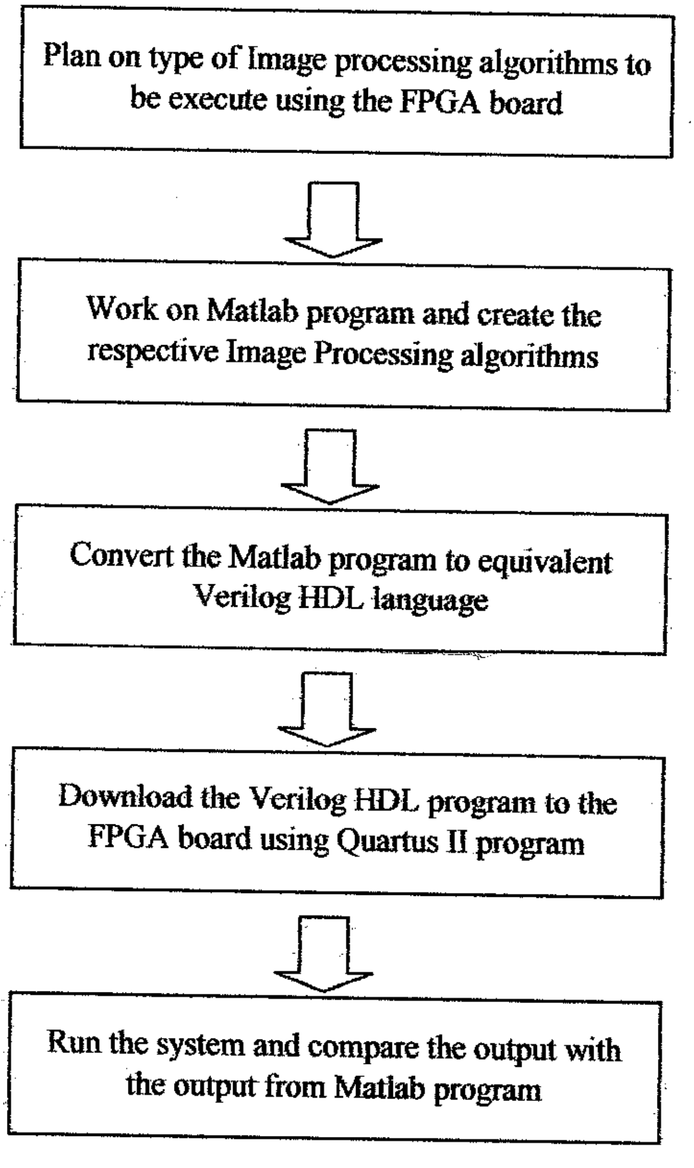
Horizontal Mask

-1	0	+1
-2	0	+2
-1	0	+1

CHAPTER 3
METHODOLOGY/PROJECT WORK

3.1 Procedures Identification

Table 2 : Working Procedures



3.2 Tools Required

Several engineering tools have been utilized in working out with the project.

3.2.1 DE2 Board (FPGA Development Board) and TRDB_DC2 Camera

Altera DE2 development board will be use while TRDB_DC2 is a 1.3 Mega Pixel digital camera development kit that can be attached at the expansion slot of DE2 board.

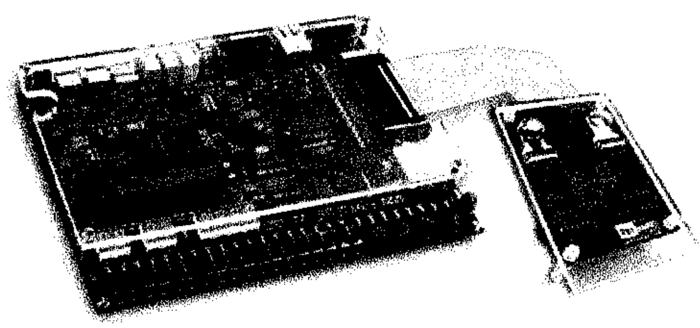


Figure 6 : TRBD_DC2 and DE2 Development Board

3.2.2 Software Required

Software required for completing this project:

1. Matlab
2. Quartus II program

CHAPTER 4

RESULT AND DISCUSSION

4.1 Color Filtering System via Matlab

1. First an input image to be processed is saved in the work directory on the Matlab folder.
2. Read the image data using "imread" instruction.

RGB – imread('Test2.jpeg')

Now, Matlab recognize the image of a three dimensional matrix (m x n x 3).

Table 3 : Three Dimension array matrix

Name	Size	Bytes Class
RGB	281x371x3	312753 uint8 array

3. Segregated a single three dimensional matrix into three of two dimensional matrix (m x n).

RED = RGB (: , : , 1);

GREEN = RGB (: , : , 2);

BLUE = RGB (: , : , 3);

Then each layer of the matrix is assigned to its respective name.

Table 4 : Matrix Size for Saved Data

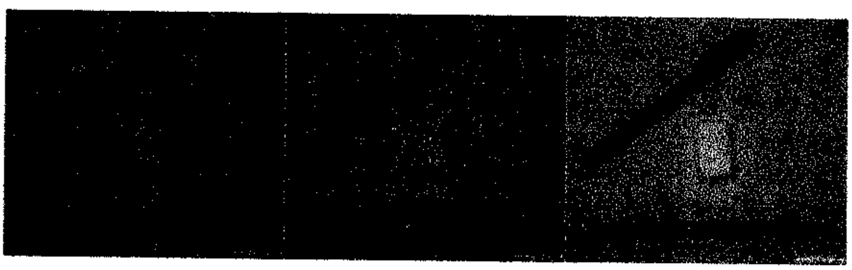
Name	Size	Bytes Class
BLUE	281x371	104251 uint8 array
GREEN	281x371	104251 uint8 array
RED	281x371	104251 uint8 array
RGB	281x371x3	312753 uint8 array

4. Show back the three layers as a grayscale image.

Figure ,imshow(RED);

Figure ,imshow(GREEN);

Figure ,imshow(BLUE);



RED GREEN BLUE

Figure 7 : Grayscale representation of each layer

Table 5 : Matrix representation of each layer

Columns 1 through 4	Columns 1 through 4	Columns 1 through 4
202 203 204 205	202 203 204 205	202 203 204 205
205 206 206 205	202 203 204 205	205 206 206 205
79 79 79 77	84 83 83 82	78 78 79 79
84 83 83 82	78 79 80 81	78 78 79 79
78 78 79 79	84 83 83 82	78 79 80 81
78 79 80 81	78 79 80 81	78 79 80 81

5. **Each pixel value from three m x n matrix will be compare with the RGB color model. If the value fall within the blue color region, the pixel value will be maintain. If else, value of 255 will be replace to the actual pixel value.**

```
for i = 1:m
    for j = 1:n
            if RED(i,j) < 110 && GREEN(i,j) <110 && BLUE(i,j) > 80 ;
            RED(i,j) = RED(i,j) ;GREEN(i,j)=GREEN(i,j) ;BLUE(i,j) =BLUE(i,j);
            else
            RED(i,j) = 255 ; GREEN(i,j) = 255 ; BLUE(i,j) = 255 ;
            end
    end
end
```

%% This loop will read each pixel value from three resource data at one time (RED, GREEN and BLUE matrixes). Then a filter is introduced in order to determine which color that the pixel fall within. If it is within the "blue region" pixel, then the values pass through. Else, the pixel value change to white {255,255,255}.

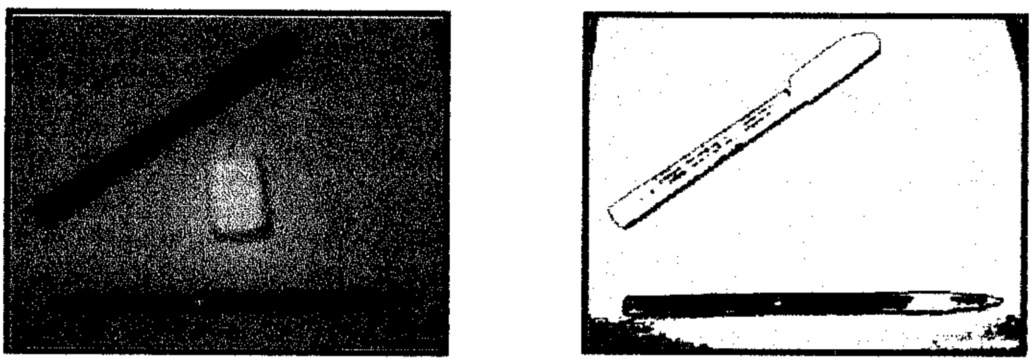
6. All the processed data from RED, GREEN and BLUE m x n matrix is combined to create a new three dimension matrix. Blue color pixel is maintained while other color is change to white.

Y(:,:,1)=RED;

Y(:,:,2)=GREEN;

Y(:,:,3)=BLUE;

figure,imshow(Y);



Input Image Output Image

Figure 8 : Input Image and its Corresponding Output

4.2 Color Filtering System via Verilog

4.2.1 Pixel Filtering Process

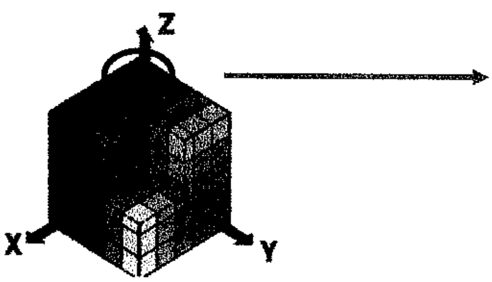
For filtering process, conditional operator was used. The conditional operator (? :) acts like a software if-then-else instruction that select between two operation. The pass band introduced will imply of two types of "blue" color, the "bright blue" and "less bright blue".

mCCD_R, mCCD_G and mCCD_B are the arrays of data captured by CMOS camera from 1.3 Mega Pixel Terasic CMOS Camera Module. These arrays supply 10 bits of image data for each pixel which create a 30 bit RGB for each color pixel representation. The processing method for normal 24 bit RGB compare with 30 bit RGB from CMOS camera is just the same, only the threshold value need to be converted for 30 bits RGB.

Single Band Pass Filter

This filter will only consider one type of "blue" color either "bright blue" or "less bright blue".

```
assign oRed = ( mCCD_R[9:0] <= 10'h121 && mCCD_G[10:1] <= 10'h1D9 &&
mCCD_B[9:0] >= 10'h22E )
? mCCD_R[9:0] : mCCD_R[9:0] ; // Single band pass filter is introduced
```



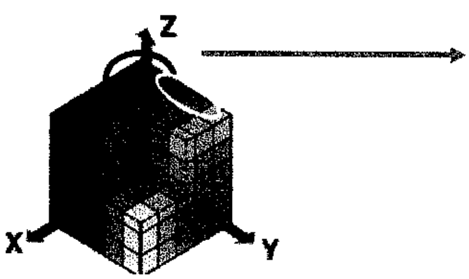
	Red	Green	Blue	Sample
Blue1	0	0	139	
Blue2	0	0	255	
Royalblue1	65	105	225	
Royalblue2	72	118	255	
Royalblue3	58	95	205	
Royalblue4	39	64	139	

Figure 9 : Single Band Pass Filte

Double Bands Pass Filters

This filter will consider both types of "blue" color, "bright blue" or "less bright blue".

```
assign oRed = ((mCCD_R[9:0]<=10'h121 && mCCD_G[10:1]<=10'h1D9 &&
mCCD_B[9:0]>=10'h22E) || (mCCD_R[9:0]<=10'h078 &&
mCCD_G[10:1]<=10'h2FE && mCCD_B[9:0]>=10'h336))
? mCCD_R[9:0] : mCCD_R[9:0] ; // Two pass band filter is introduced
```



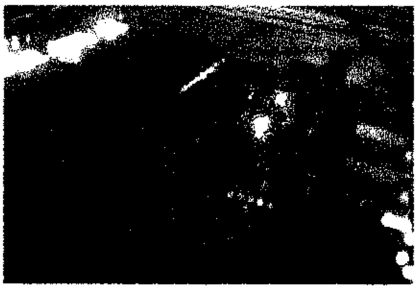
	Red	Green	Blue	Sample
Blue1	0	0	139	
Blue2	0	0	255	
Royalblue1	65	105	225	
Royalblue2	72	118	255	
Royalblue3	58	95	205	
Royalblue4	39	64	139	

	Red	Green	Blue	Sample
DeepSkyBlue1	0	191	255	
DeepSkyBlue2	0	178	238	
DeepSkyBlue3	0	154	205	
DodgerBlue1	30	144	255	
DodgerBlue2	28	134	238	
DodgerBlue3	24	116	205	

Figure 10 : Double Bands Pass Filters

Output Comparison for Single and Double band filtering method

Double Band Pass



Single Band Pass

Figure 11 : Comparison for Single and Double Band Pass Filter

4.2.2 Color Filtering System

For the actual system, picture in picture mode is selected. This mode enables user to see the input and output image at the same screen on the same time.

Figure 12 : Snapshot of Color Filtering System

4.2.3 Pixel Counting Process

A pixel counting process added to the system. This system purpose is to measure the dimension of the color by calculating the amount of pixel that represents it.

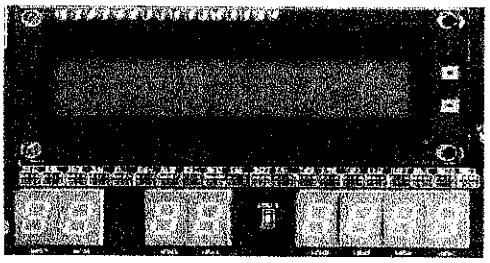


Figure 13 : 7-SEG Display on Altera DE2 board

```
begin
    if(!iRST)
    Frame_Cont    <=    0;
    else
    begin
        if( ({Pre_FVAL,iFVAL} == 2'b01) && mSTART )
        Frame_Cont    <=    0;
        else
        begin
            if(iLatch)  // iLatch is activated when a "Blue" pixel is detected for each frame
            Frame_Cont    <=    Frame_Cont+1;  // Frame_Cout is the amount of "Blue" pixel that detected within a frame
        end
    end
end
```

Frame_Cont are registered as [31:0] arrays which is capable to handle eight 7-SEG displays.

4.3 Edge Detection Process via Matlab

The edge detection by using intensity differential method is used as project reference output for the hardware system.

1. A grayscale image is read as the input data.

   GRAY=imread('K.bmp');

Figure 14 : The Grayscale input

2. Threshold value is introduced in order for convert the grayscale image to Black and While image.

```
[rows colms] = size(GRAY);
 for i = 1:rows
        for j = 1:colms
            if GRAY(i,j) > 100; %% introduced threshold
value for conversion to Black n White
                BW(i,j)=1;
            else
                BW(i,j)=0;
            end
        end
 end
```

Figure 15 : The BW image (after conversion)

3. **Intensity different for horizontal edge detection is determined.**

```
%% edge detection horizontal edge detection
hori=BW;
[rows colms] = size(BW);
for i = 1:rows
       for j = 2:colms
           if BW(i,j) == BW(i,j-1);
               hori(i,j)=1;
           else
               hori(i,j)=0;
           end
       end
end
```

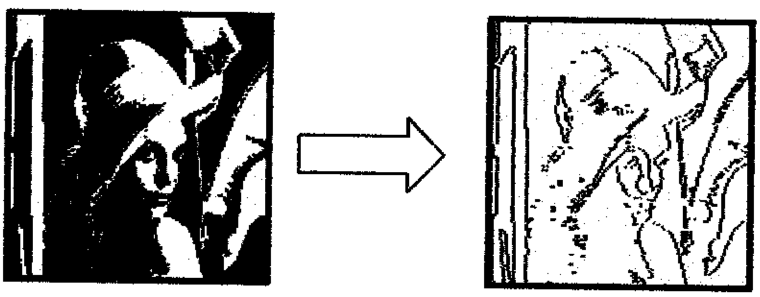
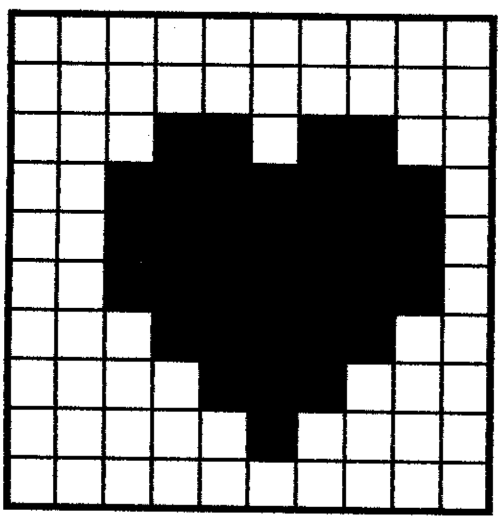


Figure 16 : Horizontal Edge Detection

How to determine horizontal edge

This horizontal edge detection method scanned the matrix from left to right, rows by rows. Any state change (from "1" to "0" or vice versa) will be mark which representing the corresponding image edge.

Table 6 : Flowchart on determined the horizontal edge



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	1	0	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	1	0	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

4. Intensity different for vertical edge detection is determined.

```
%% edge detection vertical edge detection
verti=BW;
[rows colms] = size(BW);
for i = 2:rows
       for j = 1:colms
            if BW(i,j) == BW(i-1,j);
                 verti(i,j)=1;
            else
                 verti(i,j)=0;
            end
       end
end
```

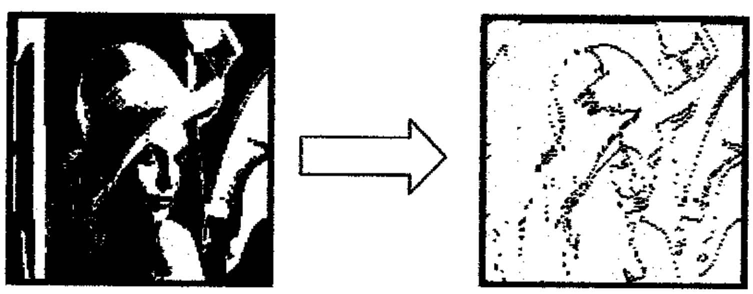
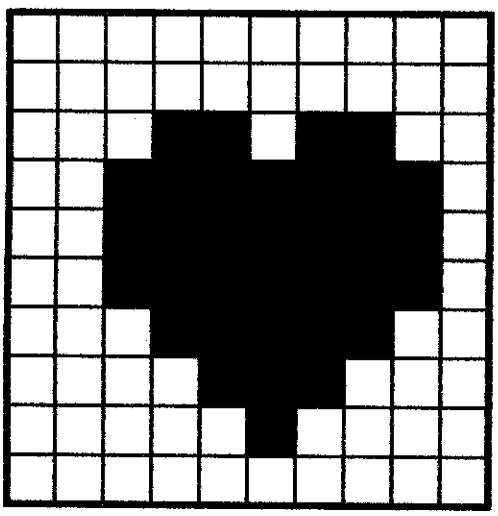


Figure 17 : Vertical Edge Detection

How to determine vertical edge

This vertical edge detection method scanned the matrix from top to bottom, column by column. Any state change (from "1" to "0" or vice versa) will be mark which representing the corresponding image edge.

Table 7 : Flowchart on determined the vertical edge



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	1	0	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	1	0	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

5. Edge data from horizontal and vertical edge is combined to construct the full image of edge detection system.

```
%% edge detection combined edge detection
combi=BW;
[rows colms] = size(BW);
for i = 1:rows
       for j = 1:colms
            if ( verti(i,j)==0 | hori(i,j)==0);
                 combi(i,j)=0;
            else
                 combi(i,j)=1;
            end
       end
end
```

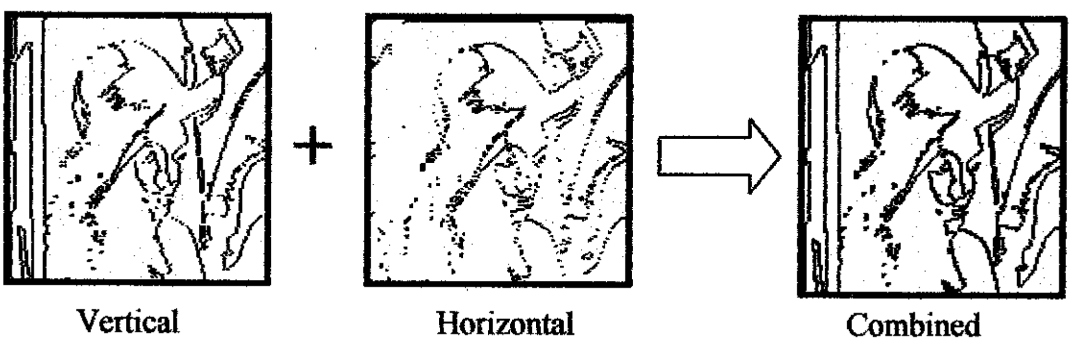


Figure 18 : Output of combined image from vertical and horizontal edge

How to determine combined edge

This combined edge detection method added the vertical edge matrix with horizontal edge matrix. This operation will create a full output of edge detection for the input image.

Table 8 : Flowchart on determined the combined edge

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	1	0	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Vertical

+

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	1	0	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Horizontal

↓↓

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	1	0	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Combined

4.4 Edge Detection Process via Verilog

The edge detection via Verilog is having several difficulties since the Cyclone II FPGA is a low performance FPGA. Furthermore within the hardware system, the input data is a video input instead of an image which required intense processing power.

For this system, only a 50 x 50 matrix from the overall image will be processed. This due to hardware low capability for processing large amount of input data at one time. The 50 x 50 matrix is marked by the blue box (as seen from figure 19).

Figure 19 : Output of Edge Detection via Verilog Program

These are the steps taken for edge detection via intensity different method:

1. The matrix data is capture onto the on board memory of the Cyclone II FPGA

```
reg    [3:0]  naser1pixels   [49:0][49:0];
// the matrix representation is capture for processing purpose

case(state)
                    grab:
                    begin

                    if ((ImXcoord >= Im2XcoordStart) &&
                           (ImXcoord <= Im2XcoordStart+49) &&
                           (ImYcoord >= Im2YcoordStart) &&
                           (ImYcoord <= Im2YcoordStart + 49))
                    begin
naser1pixels[ImXcoord-Im2XcoordStart][ImYcoord-Im2YcoordStart]              <-
(iSDRAM_IMAGE2[9:0]<- 10'b11001000) ? 0000 : 0001 ;
                    end
```

// (naser1pixels[ImXcoord-Im2XcoordStart][ImYcoord-Im2YcoordStart]) -- this algorithms will assign the matrix to the corresponding black("0") or white("1") value.

// (iSDRAM_IMAGE2[9:0]<= 10'b11001000) ? 0000 : 0001) - this algorithms will convert the grayscale image data to a BW image data

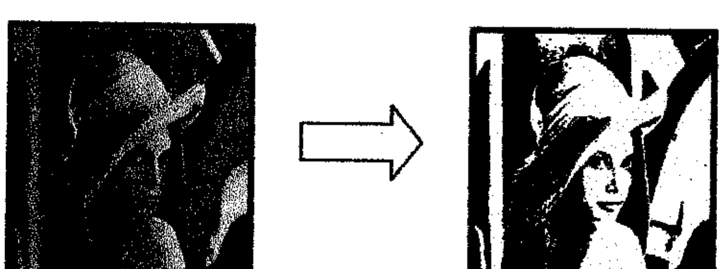


Figure 20 : example of image conversion

2. The 50 x 50 matrix is marked by a blue box for easy indication.

```
if((ImXcoord >= BestMatchXcoordStart-2)          &&
(ImXcoord <= BestMatchXcoordStart + 52)          &&
((ImYcoord == BestMatchYcoordStart-2) || (ImYcoord ==
BestMatchYcoordStart+52)))

rVGASRAM_DATA <= {iSDRAM_IMAGE1[9:0], 6'b001000};

else if((ImYcoord >= BestMatchYcoordStart-2)     &&
(ImYcoord <= BestMatchYcoordStart+52)            &&
((ImXcoord == BestMatchXcoordStart-2) || (ImXcoord ==
BestMatchXcoordStart+52)))

rVGASRAM DATA <= {iSDRAM IMAGE1[9:0], 6'b001000};
// this algorithms will mark the50 x 50 matrix and represented as the blue box on the
screen
// 6'b001000 = blue color
```

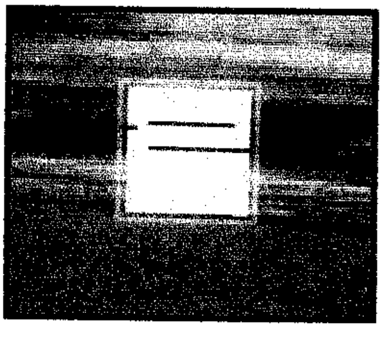


Figure 21 : The blue box as indicator for 50 x 50 matrix

3. Finally horizontal edge detection process is applied to the image.

```
else if ((ImXcoord >= Im2XcoordStart+1) &&
                                    (ImXcoord <= Im2XcoordStart+49) &&
                                    (ImYcoord >= Im2YcoordStart+1) &&
                                    (ImYcoord <= Im2YcoordStart+49))
                                    begin
rVGASRAM_DATA <=(naser1pixels[ImXcoord-Im2XcoordStart][ImYcoord-Im2YcoordStart] == naser1pixels[ImXcoord-(Im2XcoordStart+1)][ImYcoord-(Im2YcoordStart+1)]) ? 16'hffff : 16'h0000 ;
// this algorithms will run the horizontal edge detection system (refer result and discussion chapter)
```

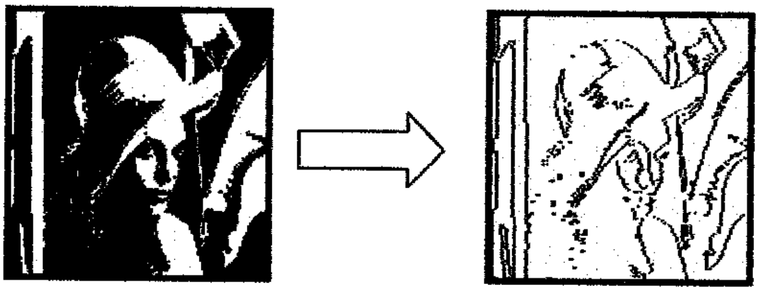


Figure 22 : Example of horizontal edge detection

Table 9 : Flowchart of system operation

Load the image data from SDRAM to the SRAM (the original image buffering system before the image being display)

↓

SRAM buffering system is introduced.

↓

Load and process the image data from the SRAM (using the on board memory of the FPGA)

↓

Re-write the processed image data onto the SRAM in order for the modified image to be shown at the screen

SRAM introduced within the system therefore not much modification being done to the original program. Modification to the original program might lead to system instability and hard to troubleshoot if any error is detected.

Table 10 : Advantage and Disadvantage for Intensity Different Edge Detection Method

Advantage	Disadvantage
Easy to implement	Not recognized as a standard method for edge detection process
Utilize simple mathematical calculation method which can be implemented using Cyclone II FPGA	Could produce error if the threshold value for the grayscale to BW image is not suitable. (no histogram analysis introduced within the system)

CHAPTER 5
CONCLUSION AND RECOMMENDATION

Color Filtering algorithms and Edge Detection algorithms are successfully implemented on Cyclone II FPGA.

In order to improve this project, few recommendations are proposed. First by introducing high end FPGA such Stratix II FPGA instead of the current Cyclone II FPGA. This offer more processing power so the higher level of image processing algorithms can be implemented on the system. Then introduce other type of camera instead of CMOS camera since by using CMOS camera, the ambient light can affect the camera efficiency.

REFERENCES

[1] Anthony Edward Nelson, thesis for Master of Science, "Implementation of Image Processing Algorithms on FPGA Hardware", 2000.

[2] Altera Corporation, "Edge Detection Reference Design" 2004

[3] Mau-Tsuen Yang, Rangachar Kasturi, Fellow, IEEE, and Anand Sivasubramaniam, Member, IEEE, "A Pipeline-Based Approach for Scheduling Video Processing Algorithms on NOW ", 2003.

[4] D.Crookes, K.Benkrid, A.Bouridane, K.Alotaibi and A.Benkrid "Design and implementation of a high level programming environment for FPGA-based image processing", 2000

[5] Koji Nakano Etsuko Takamichi, "An Image Retrieval System Using FPGAs",2000

[6] Marianne Sowa Resat, James C. Solinsky, H. Steven Wiley, Ken A. Perrine, Thomas A. Seem, " 3-D MULTISPECTRAL MONITORING OF LIVING-CELL SIGNALING USING CONFOCAL-IMAGING AND FPGA PROCESSING" ,2004

[7] Hyun Lim, Soon-Young Park and Seong-Jun Kang Wan-Hyun Cho, "FPGA Implementation of Image Watermarking Algorithm for a Digital Camera", 2003

[8] D Crookes, K Alotaibi, A Bouridane, P Donachy and A Benkrid, "An Environment for Generating FPGA Architectures for Image Algebra-based Algorithms",2000

[9] K Benkrid, D Crookes, A Bouridane, P Con and K Alotaibi, "A High Level Software Environment for FPGA Based Image Processing", 1999

[10] I.S. Uzun, A. Amira and A. Bouridane, "FPGA implementations of fast Fourier transforms for real-time signal and image processing",2005

[11] Micheal D. Ciletti, "Advanced Digital Design with the Verilog HDL", 2001

[12] *R. Williams Hunt Engineering,* " Using FPGAs for DSP Image Processing",*2006*

[13] Kevin J. Walsh," RGB to Color Name Mapping", 1996.

APPENDICES

APPENDIX A

SOBEL EDGE DETECTION METHOD

```
% Image Processing : Edge detection (Sobel)
% By Mohd Nasir Bin Mohd Shukor

input_img=imread('lena.bmp');
% Sobel filter horizontal
Hfilter = [-1 0 1;-2 0 2; -1 0 1];

% SObel filter vertical
Vfilter = [-1 -2 -1; 0 0 0; 1 2 1];

% image filtering
filteredImg1=filter2(Hfilter, input_img);
filteredImg2=filter2(Vfilter, input_img);

% edge detection
threshold=80;

% use horizontal gradient to detect edge
% learn how to use find to get the index of edge points
index=find(abs(filteredImg1)>threshold);
edge1=255*ones(1, prod(size(input_img)));
test=edge1;
edge1(index)=0;
edge1=reshape(edge1, size(input_img));

% use vertical gradient to detect edge
index2=find(abs(filteredImg2(:))>threshold);
edge2=255*ones(1, prod(size(input_img)));
edge2(index2)=0;
edge2=reshape(edge2, size(input_img));
```

```
[HORI VERT] = size(input_img);
for w = 1:HORI;
    for q = 1:VERT;
        if edge1(w,q) == 0 | edge2(w,q) == 0 ;
            edge3(w,q) = 0 ;
        else
            edge3(w,q) = 255 ;
        end
    end
end

% Compare results
figure;
subplot(3,2,1);
imshow(input_img);
title('Original Image');
subplot(3,2,2);
imshow(edge3);
title('Complete Edge Detection');
subplot(3,2,3);
imshow(filteredImg2);
title('Vertical Gradient');
subplot(3,2,4);
imshow(edge2);
title('Edge Detected Via Vertical Gradient');
subplot(3,2,5);
imshow(filteredImg1);
title('Honrizontal Gradient');
subplot(3,2,6);
imshow(edge1);
title('Edge Detected Via Honrizontal Gradient');
```

APPENDIX B

INTENSITY DIFFERENT EDGE DETECTION METHOD

```
GRAY=imread('K.bmp');
BW = GRAY;

[rows colms] = size(GRAY);
for i = 1:rows
       for j = 1:colms
           if GRAY(i,j) > 100; %% introduced threshold
value for conversion to Black n White
               BW(i,j)=1;
           else
               BW(i,j)=0;
           end
       end
end

%% edge detection horizontal edge detection
hori=BW;
[rows colms] = size(BW);
for i = 1:rows
       for j = 2:colms
           if BW(i,j) == BW(i,j-1);
               hori(i,j)=1;
           else
               hori(i,j)=0;
           end
       end
end

%% edge detection vertical edge detection
verti=BW;
[rows colms] = size(BW);
for i = 2:rows
       for j = 1:colms
           if BW(i,j) == BW(i-1,j);
               verti(i,j)=1;
           else
               verti(i,j)=0;
           end
       end
end
```

```
%% edge detection combined edge detection
combi=BW;
[rows colms] = size(BW);
for i = 1:rows
       for j = 1:colms
            if ( verti(i,j)==0 | hori(i,j)==0);
                combi(i,j)=0;
            else
                combi(i,j)=1;
            end
       end
end

   J = mat2gray(BW);
   figure(1);
   imshow(J);
   L = mat2gray(hori);
   figure(2);
   imshow(L);
   M = mat2gray(verti);
   figure(3);
   imshow(M);
   N = mat2gray(combi);
   figure(4);
   imshow(N);
```

APPENDIX C
EDGE DETECTION VIA VERILOG

```
module BlockSAD
(
        iReset,
        iClock,
        oProcessing,
        oLED,
        oSDRAM_READ_CLOCK,
        oSDRAM_READ_LOGIC,
        iSDRAM_IMAGE1,
        iSDRAM_IMAGE1_EMPTY,
        iSDRAM_IMAGE2,
        iCCDFRAMECOUNT,
        iCCDCLOCK,
        oVGASRAM_ADDR,
        oVGASRAM_DATA,
        iVGA_OK_TO_WRITE,
        iVGA_CTRL_CLK
);
        input                   iReset;
        input                   iClock;
        output                  oProcessing;
        output                  [17:0]  oLED;
        output                  oSDRAM_READ_CLOCK;
        output                  oSDRAM_READ_LOGIC;
        input           [15:0]  iSDRAM_IMAGE1;
        input                   iSDRAM_IMAGE1_EMPTY;
        input           [15:0]  iSDRAM_IMAGE2;
        input           [31:0]  iCCDFRAMECOUNT;
        input                   iCCDCLOCK;
        output          [17:0]  oVGASRAM_ADDR;
```

```
output      [15:0]  oVGASRAM_DATA;
input               iVGA_OK_TO_WRITE;
input               iVGA_CTRL_CLK;

reg         [9:0] Im2XcoordStart;
reg         [9:0] Im2YcoordStart;
reg         [9:0] BestMatchXcoordStart;
reg         [9:0] BestMatchYcoordStart;
reg         [9:0] ImXcoord;
reg         [9:0] ImYcoord;
reg         [17:0] rLED;
reg         [17:0] rVGASRAM_ADDR;
reg         [15:0] rVGASRAM_DATA;
reg         [3:0]   state;
reg         FrameGrabDone;
reg         ProcessingDone;
reg         [4:0]  SX;
reg         [4:0]  SY;
reg         doneproc;

reg     [3:0]   naser1pixels   [49:0][49:0]; // the matrix representation is capture for processing purpose

assign      oSDRAM_READ_LOGIC = iVGA_OK_TO_WRITE;
assign       oSDRAM_READ_CLOCK = iVGA_CTRL_CLK;
assign      oVGASRAM_ADDR = rVGASRAM_ADDR;
assign      oVGASRAM_DATA = rVGASRAM_DATA;
assign       oLED = rLED;
assign      oProcessing = 0;

parameter grab              = 4'd0;
parameter calcsumabs        = 4'd1;
parameter calcsumabs2       = 4'd2;
parameter done              = 4'd3;
```

```
always @ (posedge oSDRAM_READ_CLOCK)
begin
//the ram is a FIFO buffer, so we will need to scan and selectively add the correct
pixels.
        if (iReset)
        begin
            ImXcoord<=10'd0;
            ImYcoord<=10'd0;
            Im2XcoordStart<=10'd313;
            Im2YcoordStart<=10'd233;
            BestMatchXcoordStart<=10'd313;
            BestMatchYcoordStart<=10'd233;

            state<=grab;
            SX<=0;
            SY<=0;
            doneproc<=0;
        end

        else if (iVGA_OK_TO_WRITE) // modification of image only can be
done when data is synchronizing (buffering in SDRAM)
        begin

            rVGASRAM_ADDR <= { ImXcoord[9:1], ImYcoord[9:1]};

            if ((ImXcoord >= BestMatchXcoordStart-2)        &&
            (ImXcoord <= BestMatchXcoordStart+52)           &&
            ((ImYcoord == BestMatchYcoordStart-2) || (ImYcoord ==
BestMatchYcoordStart+52)))
        rVGASRAM_DATA <= {iSDRAM_IMAGE1[9:0], 6'b001000};
```

```
else if ((ImYcoord >= BestMatchYcoordStart-2) &&
(ImYcoord <= BestMatchYcoordStart+52) &&
((ImXcoord == BestMatchXcoordStart-2) || (ImXcoord ==
BestMatchXcoordStart+52)))
rVGASRAM_DATA <= {iSDRAM_IMAGE1[9:0], 6'b001000};
// this algorithms will mark the matrix which represented as the blue box on the screen

else if ((ImXcoord >= Im2XcoordStart+1) &&
(ImXcoord <= Im2XcoordStart+49) &&
(ImYcoord >= Im2YcoordStart+1) &&
(ImYcoord <= Im2YcoordStart+49))
begin
rVGASRAM_DATA <=(naser1pixels[ImXcoord-Im2XcoordStart][ImYcoord-Im2YcoordStart] == naser1pixels[ImXcoord-(Im2XcoordStart+1)][ImYcoord-(Im2YcoordStart+1)] ) ? 16'hffff : 16'h0000 ;
// this algorithms will run the horizontal edge detection system (refer result and discussion chapter)
end

else if (ImYcoord < 400)
rVGASRAM_DATA <= {iSDRAM_IMAGE2[9:0], 6'b000000};

else
rVGASRAM_DATA <= 16'h0;
```

```
case(state)
        grab:
        begin

        if((ImXcoord >= Im2XcoordStart) &&
                (ImXcoord <= Im2XcoordStart+49) &&
                (ImYcoord >= Im2YcoordStart) &&
                (ImYcoord <= Im2YcoordStart+49))
        begin
naser1pixels[ImXcoord-Im2XcoordStart][ImYcoord-Im2YcoordStart] <=
(iSDRAM_IMAGE2[9:0]<= 10'b11001000) ? 0000 : 0001 ;
        end
// this algorithms will convert the grayscale image data to a BW image data

        if (ImYcoord == 479) //done loading the registers up
        begin
                state<=done;
        end
        else
                state<=grab;
        end
        done:
        begin
                if (ImYcoord == 479) //done
                        state<=done; //wait to latch on
                else
                        state<=grab;
        end
endcase
```

```
                if (ImXcoord == 639)
                begin
                        ImXcoord <= 0;
                        if (ImYcoord == 479) //done
                        begin
                                ImYcoord<=0; //reached end of FIFO buffer
                        end
                        else
                        begin
                                ImYcoord <= ImYcoord + 1;
                        end
                end
                else
                        ImXcoord <= ImXcoord+1;
        end
    end
endmodule
```

APPENDIX D

COLOR FILTERING SYSTEM VIA MATLAB

```
RGB = imread('Test2.jpeg')

RED = RGB ( : , : , 1 );
GREEN = RGB ( : , : , 2 );
BLUE = RGB ( : , : , 3 );

Figure ,imshow( RED );
Figure ,imshow( GREEN );
Figure ,imshow( BLUE );

for i = 1:m
        for j = 1:n
                if RED(i,j) < 110 && GREEN(i,j) <110 && BLUE(i,j) > 80 ;
                RED(i,j) = RED(i,j) ;GREEN(i,j)=GREEN(i,j) ;BLUE(i,j) =BLUE(i,j);
                else
                RED(i,j) = 255 ; GREEN(i,j) = 255 ; BLUE(i,j) = 255 ;
                end
        end
end

Y(:,:,1)=RED;
Y(:,:,2)=GREEN;
Y(:,:,3)=BLUE;

figure,imshow(Y);
```

APPENDIX E

COLOR FILTERING SYSTEM VIA VERILOG

```
module RAW2RGB_4X(      oRed,
                        oGreen,
                        oBlue,
                        oDVAL,
                        oLatch,
                        iX_Cont,
                        iY_Cont,
                        iDATA,
                        iDVAL,
                        iCLK,
                        iRST    );
input   [10:0]  iX_Cont;
input   [10:0]  iY_Cont;
input   [9:0]   iDATA;
input           iDVAL;
input           iCLK;
input           iRST;
output  [9:0]   oRed;
output  [9:0]   oGreen;
output  [9:0]   oBlue;
output          oDVAL;
output          oLatch;
wire    [9:0]   mDATA_0;
wire    [9:0]   mDATA_1;
reg     [9:0]   mDATAd_0;
reg     [9:0]   mDATAd_1;
reg     [9:0]   mCCD_R;
reg     [10:0]  mCCD_G;
reg     [9:0]   mCCD_B;
reg             mDVAL;
```

```
assign oRed   = ((mCCD_R[9:0]<=10'h121  &&  mCCD_G[10:1]<=10'h1D9  &&
mCCD_B[9:0]>=10'h22E)          ||          (mCCD_R[9:0]<=10'h078          &&
mCCD_G[10:1]<=10'h2FE && mCCD_B[9:0]>=10'h336))
? mCCD_R[9:0] : mCCD_R[9:0] ;
assign oGreen =  ((mCCD_R[9:0]<=10'h121  &&  mCCD_G[10:1]<=10'h1D9  &&
mCCD_B[9:0]>=10'h22E)          ||          (mCCD_R[9:0]<=10'h078          &&
mCCD_G[10:1]<=10'h2FE && mCCD_B[9:0]>=10'h336))
? mCCD_G[10:1] : mCCD_R[9:0];
assign oBlue  = ((mCCD_R[9:0]<=10'h121  &&  mCCD_G[10:1]<=10'h1D9  &&
mCCD_B[9:0]>=10'h22E)          ||          (mCCD_R[9:0]<=10'h078          &&
mCCD_G[10:1]<=10'h2FE && mCCD_B[9:0]>=10'h336))
? mCCD_B[9:0] : mCCD_R[9:0];
// this algorithms will filter the color by introducing double pass band filter

assign oLatch =  ((mCCD_R[9:0]<=10'h121  &&  mCCD_G[10:1]<=10'h1D9  &&
mCCD_B[9:0]>=10'h22E)          ||          (mCCD_R[9:0]<=10'h078          &&
mCCD_G[10:1]<=10'h2FE && mCCD_B[9:0]>=10'h336)) ? 1 : 0 ;
// this algorithms will latch each time when it detect any blue color pixel in
conjunction with "color calculation system".

assign  oDVAL       =       mDVAL;
Line_Buffer   u0      (      .clken(iDVAL),
                                    .clock(iCLK),
                                    .shiftin(iDATA),
                                    .taps0x(mDATA_1),
                                    .taps1x(mDATA_0)   );
```

```
always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
        begin
                mCCD_R      <=      0;
                mCCD_G      <=      0;
                mCCD_B      <=      0;
                mDATAd_0<=          0;
                mDATAd_1<=          0;
                mDVAL       <=      0;
        end
        else
        begin
                mDATAd_0    <=      mDATA_0;
                mDATAd_1    <=      mDATA_1;
                mDVAL               <=      ( (|iY_Cont[1:0]) | (|iX_Cont[1:0]) )  ?
        1'b0    :       iDVAL;
                if({iY_Cont[0],iX_Cont[0]}==2'b01)
                begin
                        mCCD_R      <=      mDATA_0;
                        mCCD_G      <=      mDATAd_0+mDATA_1;
                        mCCD_B      <=      mDATAd_1;
                end
                else if({iY_Cont[0],iX_Cont[0]}==2'b00)
                begin
                        mCCD_R      <=      mDATAd_0;
                        mCCD_G      <=      mDATA_0+mDATAd_1;
                        mCCD_B      <=      mDATA_1;
                end
```

```
            else if({iY_Cont[0],iX_Cont[0]}==2'b11)
            begin
                mCCD_R    <=    mDATA_1;
                mCCD_G    <=    mDATA_0+mDATAd_1;
                mCCD_B    <=    mDATAd_0;
            end
            else if({iY_Cont[0],iX_Cont[0]}==2'b10)
            begin
                mCCD_R    <=    mDATAd_1;
                mCCD_G    <=    mDATAd_0+mDATA_1;
                mCCD_B    <=    mDATA_0;
            end
        end
    end

endmodule
```