Real Time Implementation of Sobel Edge Detector on FPGA

Dissertation submitted towards the partial fulfillment for the Degree of

Master of Technology

in

VLSI Design

Submitted By

Dhruv Singhal

Roll No. 601461009

Under the supervision of

Dr. Vinay Kumar

Assistant Professor, ECED

Department of Electronics & Communication Engineering

Thapar University, Patiala-147004

2016

CERTIFICATE

I hereby declare that the work which is being presented in the thesis entitled, **"Real Time Implementation of Sobel Based Edge Detector on FPGA"** in partial fulfilment of the requirements for the award of degree of Master of Technology in VLSI Design at Electronics and Communication Department of Thapar University, Patiala is an authentic record of my study carried out under the supervision of **Dr. Vinay Kumar**, Assistant Professor, ECED.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Date: 14-07-2016

Dhruv Singhal

Place: Patiala

Roll No.601461009

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Date: 14-07-2016

Place: Patiala

Dr. Vinay Kumar
Assistant Professor, ECED
Thapar University, Patiala

Counter signed by:

Dr. Sanjay Sharma
Professor & Head, ECED
Thapar University, Patiala

Dr. S. S. Bhatia
Dean of Academy Affairs
Thapar University, Patiala

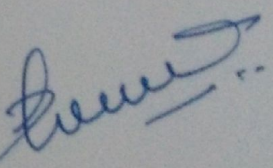
ACKNOWLEDGEMENT

It is my proud privilege to acknowledge and extend my gratitude to several people who helped me directly or indirectly in completion of this report. I express my heart full indebtedness and owe a deep sense of gratitude to my teacher and my faculty guide **Dr. Vinay Kumar, Assitant Professor, Electronics and Communication Engineering Department,** for his sincere guidance and support with encouragement to go ahead.

I am also thankful to **Dr. Sanjay Sharma, Professor and Head, Electronics and Communication Engineering Department**, for providing us with the adequate infrastructure for carrying out the work. I am also thankful to **Dr. Amit Kohli, P.G. Coordinator, Electronics and Communication Engineering Department**, for the motivation and inspiration that triggered me for the work.

My greatest thanks to all who wished me success especially my parents. I would like to thank my friends who were always there to help me.

The study has indeed helped me to explore knowledge and avenues related to my topic and I am sure it will help me in my future.



(Dhruv Singhal)

ABSTRACT

Image processing is a computationally intensive operation and typically done in software using CPU processing power. However, even with the advances in computing technology today, software-based image processing requires expensive and powerful CPUs to perform real-time image processing. This is where a low cost FPGA based image processing solution becomes useful. It eliminates the need for powerful CPUs and at the same time can perform real-time processing relatively easily. This work presents the implementation of such an image processing solution in hardware, using a FPGA at its core.

The high level goal is to detect edges of an image and the algorithm used for this purpose is Sobel edge detection. Edge detection is the fundamental operation in image analysis. It has to be done for any high end application, such as security cameras. There are various image detection techniques available. These include- Canny, Krisch, Lapla1 and Lapla2. As compared to other techniques Sobel has been chosen for the research work, over the other techniques, because of its simplicity and moderate average risk (AVR) to signal to noise (SNR) ratio.

TABLE OF CONTENTS

LIST OF ACRONYMS

CPU	Central Processing Unit
FPGA	Field Programmable Gate Array
AVR	Average Risk
SNR	Signal to Noise Ratio
HDL	Hardware Description Language
DIP	Digital Image Processing
PC	Personal Computer
ASIC	Application Specific Integrated Circuit
VLSI	Very Large Scale Integration
IC	Integrated Circuit
VGA	Video Graphic Array
RAM	Random Access Memory
CLB	Configurable Logic Block
OTP	One Time Programmable
ALU	Arithmetic Logic Unit
LUT	Look Up Table
FF	Flip-Flop
BRAM	Block RAM
DCM	Digital Clock Multiplexer
JTAG	Joint Test Action Group
DSP	Digital Signal Processor
hsync	Horizontal Sync
vsync	Vertical Sync

LIST OF FIGURES

LIST OF TABLES

CHAPTER

1 INTRODUCTION

Recent development of Hardware Description Languages (HDLs) and their easy availability, makes them capable of not only describing the circuits logically but also checking the performance and functionality of designs using development tools and test environments. As the name suggests, HDLs are hardware descriptive languages and the main advantage of using them is the possibility of an immediate hardware implementation [1].

Other advantages offered by HDLs are hardware portability and the re-programmability of hardware. HDLs work on binary data, and therefore the input and output data need to be manipulated to match the binary format permitted by the simulators. However, the main challenge is the replication of image processing algorithms into a HDL [1][2].

Digital Image Processing (DIP) proliferated with the advancement of cheaper computer and application specific hardware. Today DIP finds its application in areas like arts, space exploration, multimedia services, medicine, authentication, surveillance and automated industry inspection [1]. Areas of application of DIP are very diverse and to develop a basic idea of its applications, classification of images according to their source are required. Few examples are acoustic, electromagnetic energy spectrum (e.g., X-ray), ultrasonic and synthetic images produced by computer. Different types of processes are used in the above mentioned applications, including object detection and image quality enhancement [1].

With the development in the field of DIP, quality and size of images are increasing day by day. Demand of proper resources and hardware for image processing applications increases drastically with the advancement of digital devices capability. Sometimes general purpose PCs and CPUs fail to avail these resources and are not very efficient in terms of speed, time and power distribution. Very Large Scale Integrated (VLSI) technology has brought another alternative of

hardware implementation. The use of application specific hardware and HDL allow designers to implement image processing algorithms in very efficient manner and ensures a smaller time to market interval [2]. This is where the Application Specific Integrated Circuit (ASICs) and Field Programmable Gate Arrays (FPGAs), comes into picture. They offer greater efficiency when compared with software implementation. There are mainly two options available for hardware implementation- ASIC and FPGA. Both are reconfigurable and support pipelining and parallelism.

The use of reconfigurable hardware to implement algorithms for image processing minimizes the time to market cost. It also makes possible rapid prototyping with simplified debugging and verification stages. Therefore, the reconfigurable devices seem to be the ideal choice for implementation of image processing algorithms [3].

1.1 MOTIVATION

The motivation to study and develop a better real-time design is to fulfil the need of high throughput rate and resources required in complex operations by DIP. To fulfil this need, parallel processing in the form of custom hardware or multiprocessing in the form of software task become indispensable. For a particular application, hardware solutions are always preferable for their lower system cost. In present work, hardware solution to perform Sobel based edge detection algorithm has been proposed. The purpose of designing such a design is to allow detection of edges in images, as fast as possible, with the resources currently available.

1.2 KEY CONTRIBUTIONS

With the successful completion of this work the following has been achieved:

1. Design and simulation of an algorithm for approximate fixed point model for Sobel based edge detector.
2. Display of a binary image on VGA monitor using BRAM available on FPGA.

1.3 ORGANIZATION OF THE THESIS

The organization of this thesis is as follows:

Chapter 1: This chapter discussed the major advantages of using HDL and FPGA in the research work presented in this report.

Chapter 2: This chapter gives an introduction to FPGA and the advancements in this field over time. It also justifies the selection of the kit used for this work over the other available options.

Chapter 3: This chapter introduces Sobel edge detection technique and its applications.

Chapter 4: This chapter presents an introduction to VGA monitors and their timing constraints.

Chapter 5: This chapter gives a brief review of the related work that has been done in this area.

Chapter 6: This chapter discusses the methodology adopted for research work and its flow.

Chapter 7: It contains the simulations and results of the work. It also encloses the screenshot of the simulation result and the resultant edge detected image.

Chapter 8: In this chapter, the summary of the dissertation and the future scope of this research work have been presented.

CHAPTER 2

INTRODUCTION TO FPGA

As the name suggests, FPGAs are a collection of reprogrammable gate arrays. FPGAs overcame the limitations of gate array ASICs and eventually reduce their demand. FPGAs are developed and marketed, mainly with the following two intentions: (a) for ASICs prototyping and (b) to achieve more time to market by installing FPGAs in systems first and later replacing them at the earliest [4].

FPGAs offer a fast programming option, as they can be easily programmed with the help of computers in a matter of minutes. ASICs, on the other hand, take months to fabricate. Since designers have got their hands on FPGA, they have noticed that FPGAs can be a low cost alternative to ASICs [4][5]. This is because they offer comparable speed, less power consumption and less time to market. All of these at low cost. It is because of these factors that FPGAs began shipping with the systems without any intention of getting replaced by ASICs. But still FPGAs are no match to ASICs in high volume applications for example high-end IP core routers. In these cases FPGAs are used to prototype ASICs to complete their verification and validation as soon as possible [5].

ICs with a matrix of Configurable Logic Blocks (CLBs) and programmable interconnects are used to make FPGAs as shown in Figure 2.1. FPGAs offer post-manufacturing re-programmability according to desired application or functionality requirements. It is this particular feature that sets FPGAs distinctly apart from the ASICs. ASICs, unlike FPGAs, are customized to perform specific design tasks. FPGAs also come in one-time programmable (OTP) version, however the SRAM based FPGAs are the ones that dominate. The SRAM based variant can be reprogrammed with the evolution of the design.

2.1. FPGA ARCHITECTURE

There are different FPGAs that are manufactured by different vendors but the basic architecture is same for all. This architecture is shown in Figure 2.1. Clearly, according to the Figure 2.1 FPGA architecture includes configurable logic blocks or CLBs and configurable input/output (I/O) blocks. These blocks are connected using programmable interconnects [5]. In addition to these, every block requires a clock signal to drive its operation and for which it requires additional clock circuit. Other logic blocks such as ALUs, memory, and decoders can also be a part of this architecture.

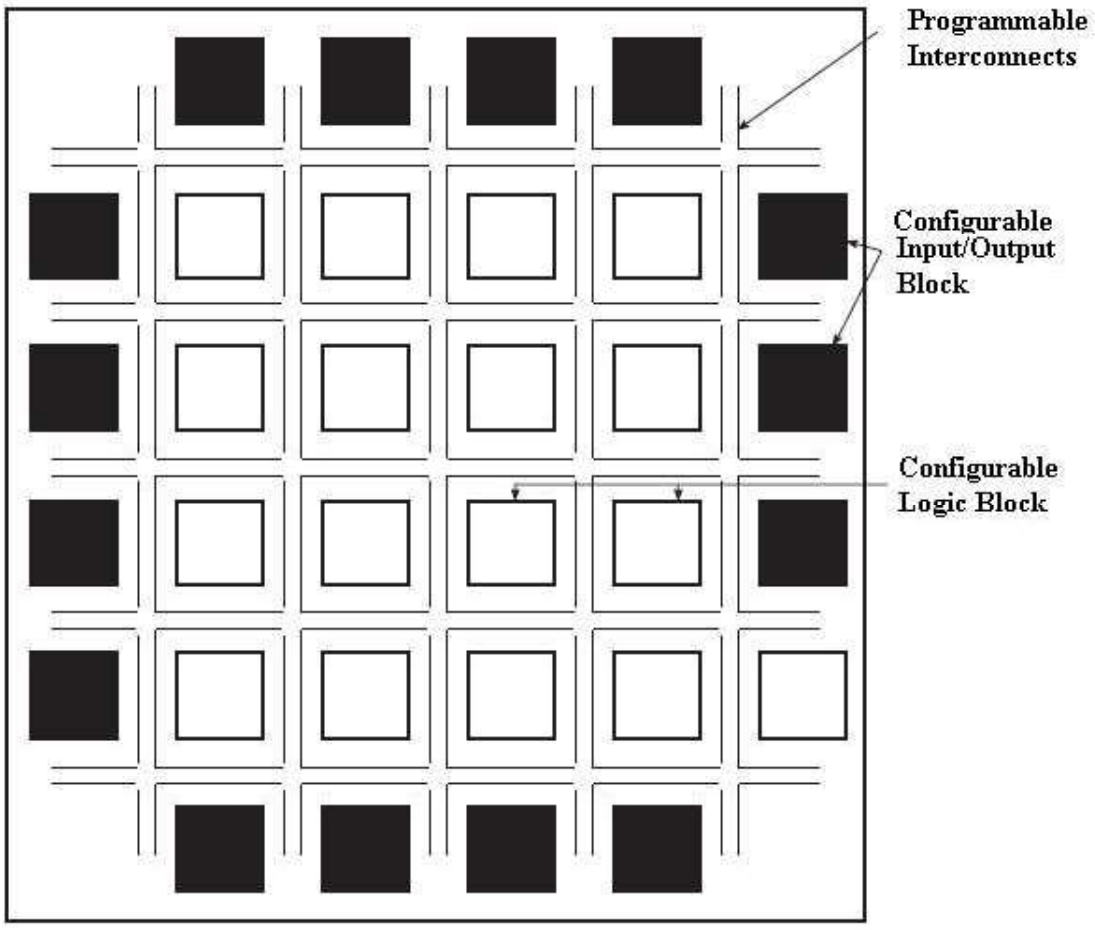


Figure 2.1: Generic FPGA architecture [5]

2.1.1 Configurable Logic Blocks (CLBs)

The CLBs are the logic containing blocks in the FPGA. In conventional market all FPGA vendors make use of large-grain architecture. Such an architecture uses CLBs, depicted in Figure 2.2, which contain logic that is enough to result in a small state machine. According to the Figure 2.2, the block uses

Look Up Tables (LUTs) to create arbitrary combinatorial logic functions [5]. The other elements include flip-flops used for clocked storage elements, and multiplexers for routing the logic within the block. MUX decide what logic will appear at output port. If Select is low we get the LUT logic as output whereas if Select is high we get logic from flip-flop. In Figure 2.2 A, B, C and D are four inputs to a 4-input LUT where Rst and Clk are reset and clock signal, respectively.

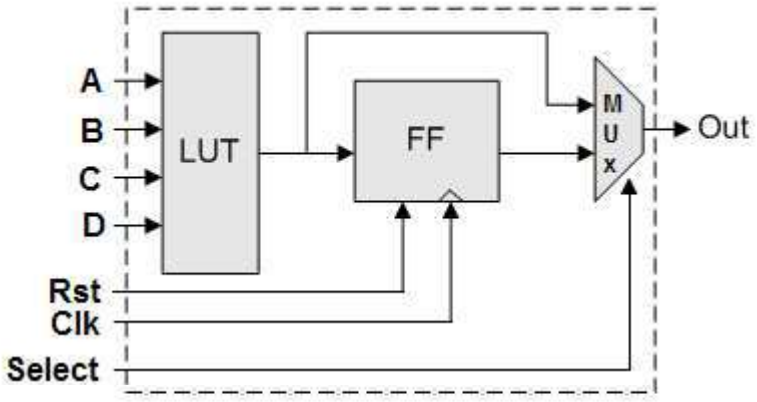


Figure 2.2: FPGA configuration logic block

2.1.2 Configurable I/O Blocks

Another important block in the FPGA architecture is a configurable I/O block, depicted in Figure 2.3. It is used to input the signals onto the chip, and also for sending out the output off the chip. To offer this functionality it makes use of an input and an output buffer. These buffers make use of three-state and also open collector output controls [6]. Generally discrete resistors, connected externally to the chip, are avoided by making use of pull up resistors on the outputs, or sometimes even pull down resistors, for the termination of signals and buses.

The output allows the programmability of their polarity, they can be programmed to be active high or active low. Programmability of the output slew rate according to fast or slow rise and fall times is also possible. To ensure that the setup time requirements of the externally connected devices are met, flip-flops are placed at the outputs. These ensure that the clocked signals are feed directly to the pins without encountering significant delay. Flip-flops at the input are required to reduce the delay on the signal before it reaches the flip-flop. This reduces the hold time requirement of the FPGA [5][6].

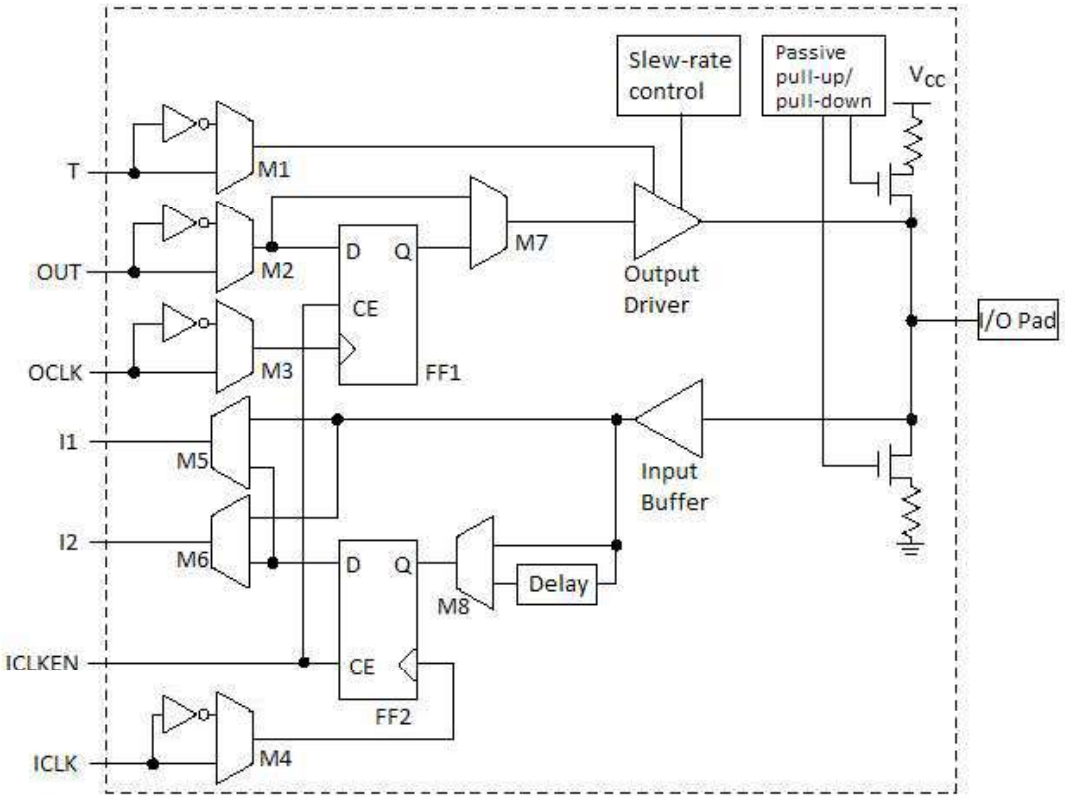


Figure 2.3: FPGA configurable I/O block

2.1.3 Programmable Interconnect

Figure 2.4 represents the hierarchy of interconnect resources. The critical CLBs, which are placed physically far away from each other should not induce much delay. And because of that, long lines are used for making the connections. Long lines are specially designed to have low impedance and as a result low propagation time. The same lines can also function as buses within the chip. For the CLBs that happen to be located physically close to each other short lines are used for connecting the CLBs. Transistors are utilized as a switch for turning on or off the connections between different lines. For making specific and flexible combinations of connections between the long and short interconnect lines, programmable switch matrices are used in the FPGA. A bus is created by connecting numerous CLBs to a long line using three state buffers [6].

Global clock lines are long lines, there fast operation is used for drawing a connection between the clock buffers and every clocked element in all the CLBs. This allows the clock signal distribution throughout the FPGA while maintaining a

minimal skew between the clock signals that arrive at the different flip-flops in the chip [6].

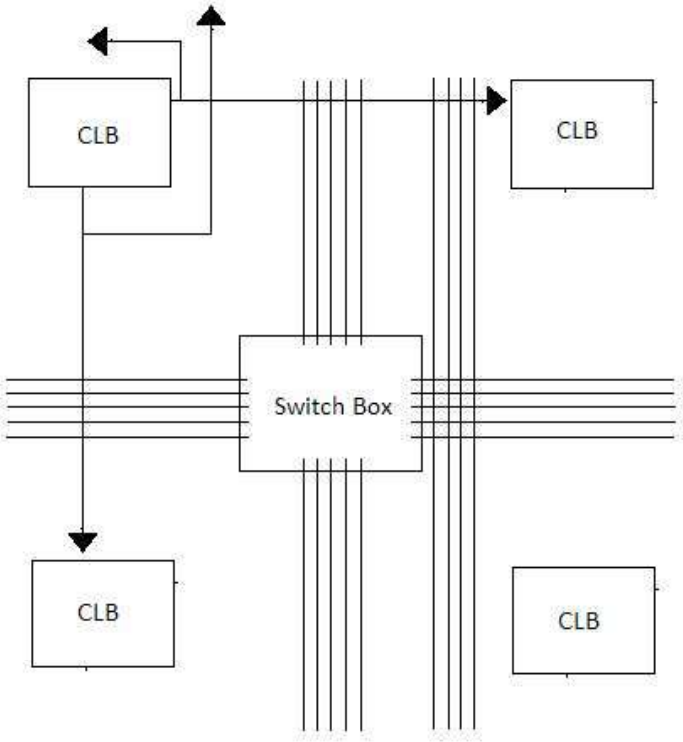


Figure 2.4: FPGA programmable interconnect

2.1.4 Clock Drivers

Clock drivers are special high drive clock buffers that are used by special I/O blocks distributed throughout the FPGA. The clock drivers connect to the clock input pads and are responsible for driving the clock signals onto the global clock lines. The clock lines are designed to offer low skew times and fast propagation time. It is not possible to guarantee absolute skew and delay anywhere other than the global clock lines on a FPGA, this makes the synchronous design of a FPGA necessary [6].

2.2 VARIOUS FPGA FAMILIES

In the market there are mainly two vendors in the field of FPGA, one is ALTERA and other is XILINX. Generally these FPGAs are categorised into families on the basis of their features. The vast variety of choices in every cost segment enables a designer to be spared of on-board constraints, such as memory

and processing speed. It also allows the designer to have a low cost solution based on their design [7][8].

Following are some examples of the ALTERA and XILINX family FPGAs:

1. Altera major families/series are [7]:
   1.1 Stratix
   1.2 Arria
   1.3 Cyclone
   1.4 Max
2. Xilinx major families/series are [8]:
   2.1 Artix
   2.2 Kintex
   2.3 Virtex
   2.4 Spartan

Below are the Altera MAX10 and Xilinx Spartan 3e product table and their features, Table 2.1 mention the features of Altera MAX10 product family whereas Xilinx Spartan 3e product features and mentioned in Table 2.2.

Table 2.1: Altera MAX10 Product Table [7]

Product Line	10M02	10M04	10M08	10M16	10M25	10M40	10M50
LEs (K)	2	4	8	16	25	40	50
Block memory (kb)	108	189	378	549	675	1260	1638
User flash memory (kb)	12	16-156	32-172	32-296	32-400	64-736	64-736
18x18 multipliers	16	20	24	45	55	125	144
PLLs	1, 2	1, 2	1, 2	1 ,4	1, 4	1, 4	1, 4
Internal configuration	Single	Dual	Dual	Dual	Dual	Dual	Dual
ADAC	-	1	1	1	2	2	2
Temperature sensor	-	1	1	1	1	1	1
External memory	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 2.2: Xilinx Spartan 3e Product Table [8]

Product Line	XC3S100E	XC3S250E	XC3S500E	XC3S1200E	XC3S1600E
System gates	100K	250K	500K	1200K	1600K
Slices	960	2448	4656	8672	14752
Logic cells	2160	5508	10476	19512	33192
CLB flip-flops	1920	4896	9312	17344	29504
Distributed RAM	15	38	73	136	231
Block RAM (kb)	72	216	360	504	648
Digital Clock	2	4	4	8	8
Single Ended I/Os	108	172	232	304	376
Differential I/O Pairs	40	68	92	124	156

Based on description given in the Table 2.1-2.2 appropriate hardware can be selected with proper resources. Below is the list of the resources we want for the proposed work:-

1. Memory:
   We are taking a 90x90 image as input so the total requirement of memory comes out to be:
   90 x 90 x 8(bits per pixel) = 64800 bits
2. Clock Generator Circuit:
   We need a 25 MHz clock to refresh every pixel of the image. Why 25 MHz? This part of calculation is explained in Chapter 4.
3. VGA Connector:
   As we are going to display the resultant image on a VGA monitor we need a hardware which supports a VGA port. According to the Table 2.1 and 2.2, all the mentioned Max10 and Spartan 3E products support VGA port on-board.
4. Comparable Number of Flip-Flops:
   As our Verilog implementation deals with a large number of arrays, many for loops are present in the design managing 8100 pixels. So it is necessary to take considerable amount of flip-flops into account.

2.3 FINAL HARDWARE SELECTION

The Xilinx XC3S500E Spartan 3e FPGA Starter Kit selected as the hardware for proposed work. It provides a low-cost development and evaluation platform for FPGA design [9].

Figure 2.5 shows Spartan 3e Starter Kit board, which includes following components and features:

- 200,000-gate Xilinx Spartan 3e XC3S500 FPGA in a 256-ball thin Ball Grid Array package (XC3S500FT256)
    - 4,320 logic cell equivalents
    - Twelve 18K-bit block RAMs (216K bits)
    - Twelve 18x18 hardware multipliers
    - Four Digital Clock Managers (DCMs)
    - Up to 173 user-defined I/O signals
- 2Mbit Xilinx XCF02S Platform Flash, in-system programmable configuration PROM
    - 1Mbit non-volatile data or application code storage available after FPGA configuration
    - Jumper options allow FPGA application to read PROM data or FPGA configuration from other sources
- 1M-byte of Fast Asynchronous SRAM
    - Two 256Kx16 ISSI IS61LV25616AL-10T 10 ns SRAMs
    - Configurable memory architecture
        - Single 256Kx32 SRAM array, ideal for MicroBlaze code images
        - Two independent 256Kx16 SRAM arrays
    - Individual chip select per device
    - Individual byte enables
- 3-bit, 8-color VGA display port
- 9-pin RS-232 Serial Port
    - DB9 9-pin female connector (DCE connector)
    - RS-232 transceiver/level translator

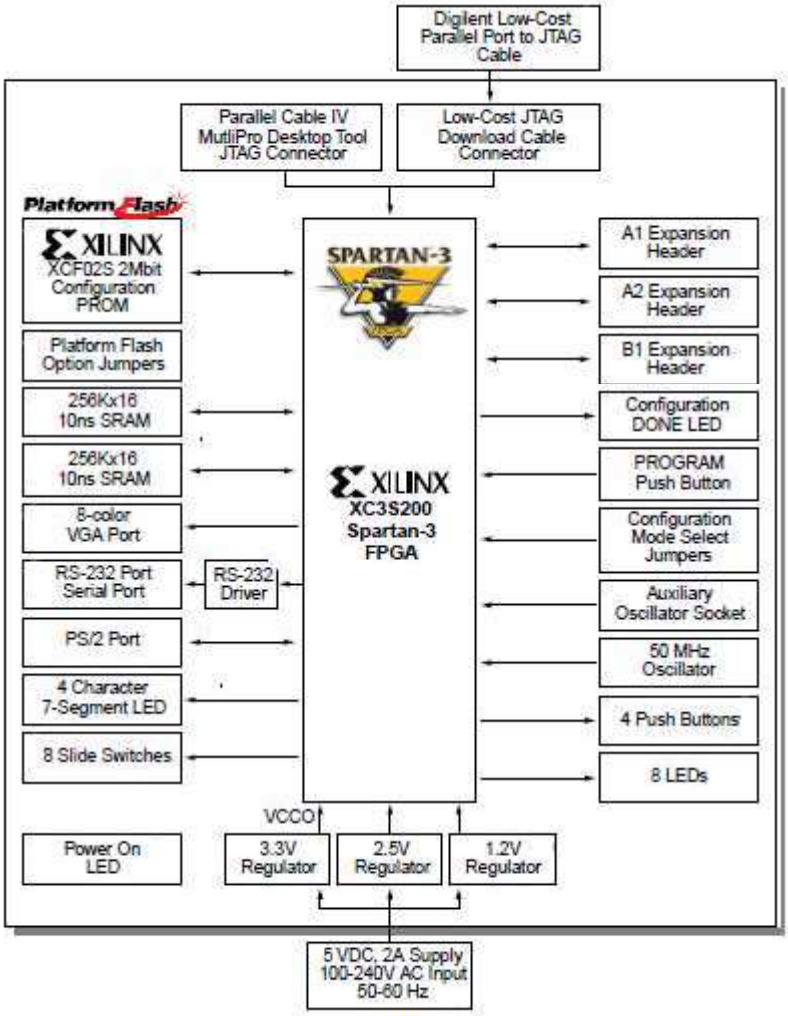


Figure 2.5: Xilinx Spartan 3e starter kit board block diagram [9]

CHAPTER

3 INTRODUCTION TO SOBEL EDGE DETECTION METHOD

Digital image processing deals with the manipulation of digital images by means of computer algorithm. It is a subcategory of signals and systems which mainly focuses on images. DIP focuses on the development and implementation of image processing algorithms [10]. The basic flow is shown in Figure 3.1.

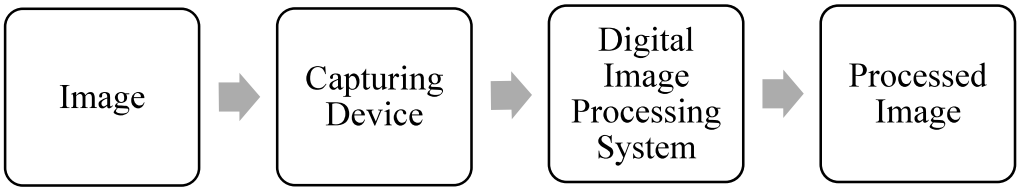


Figure 3.1: Basic image processing flow

Digital image processing has very vast applications in almost every field. Now a day it is not just limited to manipulating the image brightness and contrast, rather there is much more. DIP finds its application in the medical field, remote sensing, transmission & encoding and machine/robot vision [10].

3.1 WHAT IS AN IMAGE?

Capturing an image from a camera is a physical process. An array of sensors is used to capture image. When sunlight falls on an object it gets reflected, this reflected intensity is sensed by the sensor, and a voltage signal corresponding to it is generated. Then this analog voltage signal is converted to a digital value by means of quantization and sampling, in order to get a digital image.

An image is nothing but a 2-dimensional array, I(x,y), where x represents the horizontal coordinates and y represents the vertical coordinates. These coordinates contain the digital value known as pixel of an image. Pixel is the smallest entity of an image. It corresponds to a value ranging from 0 to 255 (integer). These pixel values are used in order to process an image.

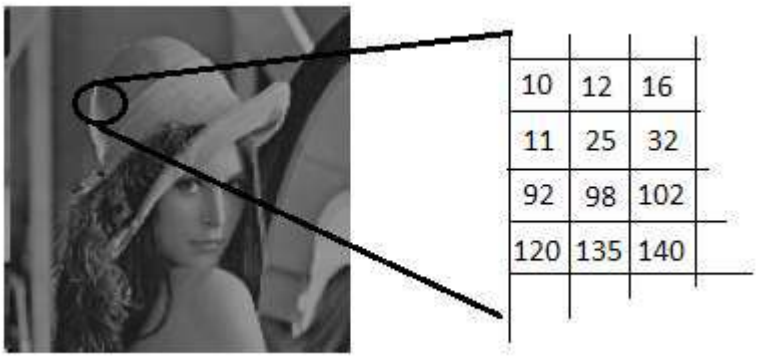


Figure 3.2: Pixel values corresponds to encircled image

3.2 EDGE DETECTION

Edges are very important features in any image. They comprise the main details of an image and found wherever there is a sudden change or discontinuity in an image. They are the prime source of information about the shape of any object.

The objective of detecting edges in an image is to capture important events and changes in the properties of the object. Generally, edges in an image are likely to give information about [11]:

- Discontinuities in depth
- Discontinuities in surface orientation
- Changes in material properties
- Variations in scene illumination

When the image is passed through any edge detector, the result leads to connected curves indicating edges or boundaries of object. These edges correspond to surface markings as well as surface orientations. It is to be noted that the resultant image has very less data to be processed. It filters out less relevant data, while keeping important details of an image safe. Therefore after edge detection is done, the image processing are simplified as information content is reduced in comparison to the original image. However, it is not always possible to obtain such ideal edges from real life images of moderate complexity [12].

Edge detection is one of the fundamental steps in image processing, image analysis, image pattern recognition, and computer vision techniques. There are several edge detection methods i.e. Prewitt, Sobel, Roberts and Canny operator. Commonly used

method for detecting edges is to apply derivative operators on images. Derivative based approaches can be categorized into two groups, namely first and second order derivative methods. First order derivative based techniques depend on computing the gradient in several directions and combining the result of each gradient. The value of the gradient magnitude and orientation is estimated using two differentiation masks which are explained in Section 3.3 [13].

In present work Sobel edge detection method is considered. Because of the simplicity and common usage, this method is preferred over others mentioned in this report.

3.3 SOBEL OPERATOR

Various methods are available for edge detection. The majority of methods can be categorized into these two groups [14]:

1. Gradient: In this method, edges are detected by looking for the minimum and maximum of the first derivative of image; for e.g. Prewitt, Roberts, Sobel operator. In such methods, the edges are very sharp for the detected features. This has been shown in Figure 3.3.
2. Laplacian: To find edges in this method, zero crossings are searched in the second derivative; for e.g. Laplacian of Gaussian, Marr-Hildreth etc. An edge has one dimensional shape of a ramp and calculating the derivative of the image can highlight its location, as shown in Figure 3.4.

Input Image

Output Image

Figure 3.3: The gradient method

Input Image Output Image

Figure 3.4: The laplacian method

The Sobel operator is a member of the gradient category. It is a discrete differentiation operator which computes an approximation of the gradient of the image intensity function [14]. Consider an image f(x, y) as shown in Figure 3.5. The *x*-coordinate is defined here as increasing in the "down"-direction, and the *y*-coordinate is defined as increasing in the "right"-direction. Different operators are evaluated in Equation (i) and (ii) for pixel (i, j).

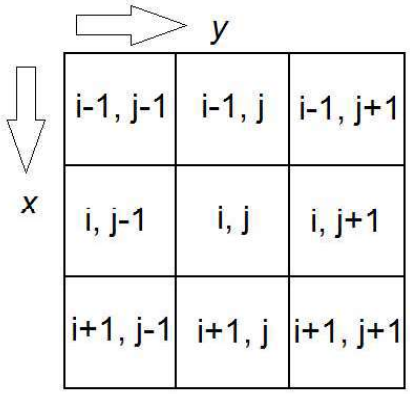


Figure 3.5: Image f(x, y)

$$\Delta x = f(i, j+1) - f(i+1, j) \quad \text{(i)}$$

$$\Delta y = f(i, j+1) - f(i, j) \quad \text{(ii)}$$

where, i and j are pixel coordinates of image f(x,y). Equation (i) and (ii) correspond to convolving the image with the following masks mentioned in Equation (iii).

$$\Delta x = \begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}, \quad \Delta y = \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix} \quad \text{(iii)}$$

We can also use 3x3 mask. The advantage of using a larger mask size is the reduction in the errors due to the effects of noise. But we can increase mask size to an extent as with increase in mask size accuracy gets reduce. An advantage of using a mask of odd size is that the operators are centred and provide an estimate that is based on a centre pixel (i, j). One important edge operator of this type is the Sobel edge operator. The Sobel edge operator masks are given in Equation (iv) and (v).

$$\text{Horizontal Mask } \Delta x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{(iv)}$$

$$\text{Vertical Mask } \Delta y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & 2 & 1 \end{bmatrix} \quad \text{(v)}$$

Let's say 'I' is our source image, and G_x and G_y are two images which contain the horizontal and vertical derivative approximations. The computations are illustrated in Equation (vi).

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & 2 & 1 \end{bmatrix} * I \quad \text{(vi)}$$

where, * represents the 2- dimensional convolution operation.

At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \quad \text{(vii)}$$

Its approximated formula shown in Equation (viii)[14].

$$|G| = |G_x| + |G_y| \quad \text{(viii)}$$

3.4 HDL IMPLEMENTATION OF SOBEL OPERATOR - ALGORITHM MAPPING

The N×N window operation is shown in the Figure 3.5. In present work window operation is Sobel operation in which 3×3 mask is convolved with the input image.

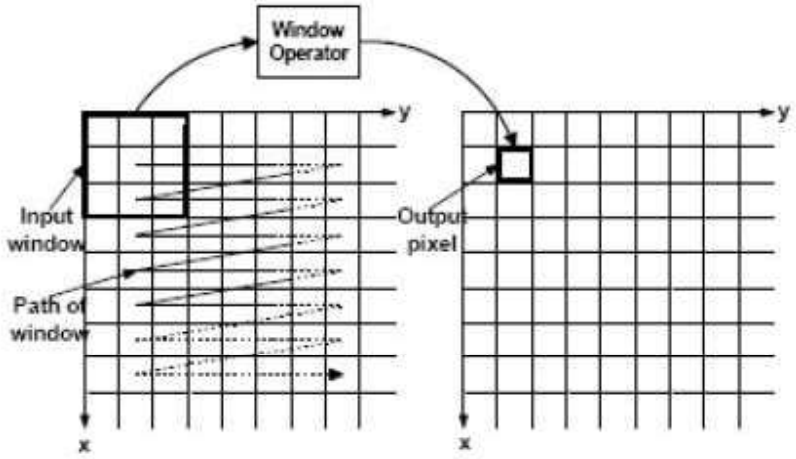


Figure 3.6: Mapping the window operation

In Verilog we cannot slide window over an image because of memory constraint which restrict us in obtaining all these pixels in each clock cycle. To resolve that, we store the entire input image into a frame buffer and to produce output image we access the neighbourhood pixels and applied the operator as needed as shown in Figure 3.7. Input data from the previous N-1 rows can be cached using a shift register for when the window is scanned along subsequent lines [16].

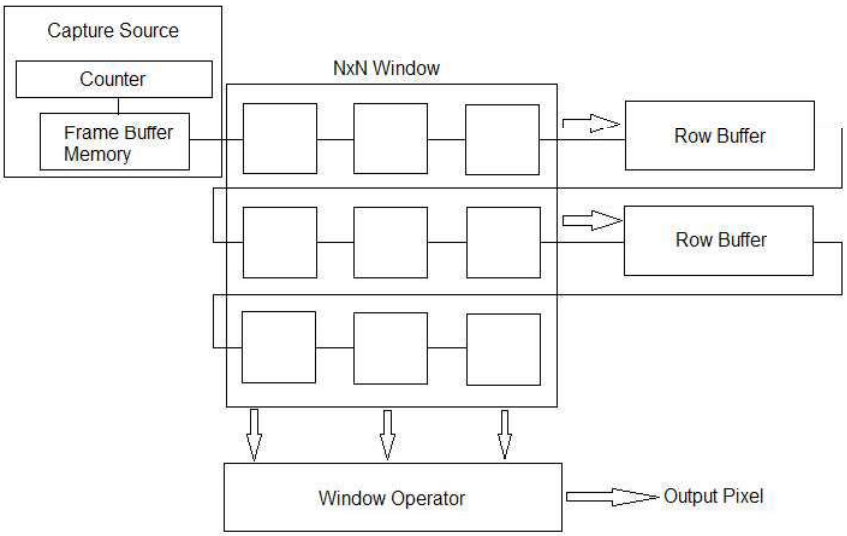


Figure 3.7: Window operation using buffers

To understand this consider an image I(x, y) of size 90x90 having pixels value ranging from 0 to 255. This image is converted into a column matrix and saved as a txt file. Then this file is read in our code and pixel values are stored in 2D array or frame buffer to say.

Now to apply window operation on stored image (in present work it is convolution with G_x and G_y which are explained in Equation (vi)) pixel values of image are feed to variables p0, p1, p2, p3, p4, p5, p6, p7 and p8 using row buffers. In every clock cycle values of these variables get new pixel values from frame buffer and after convolving these values with mask it get saved in output matrix as shown in Figure 3.8.

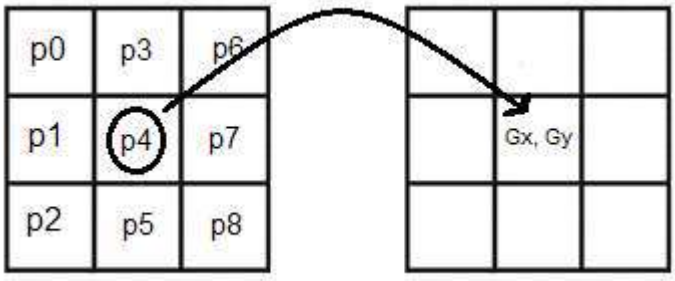


Figure 3.8: Mapped window operation

$$G_x(p4) = (p2-p0) + 2(p5-p3) + (p8-p6) \quad \text{(ix)}$$

$$G_y(p4) = (p0-p6) + 2(p1-p7) + (p2-p8) \quad \text{(x)}$$

We applied two *for* loops to get the new pixel values from frame buffer and to calculate gradient values as illustrated in pseudocode below:

```
Line1: for (i=1; i<91; i=i+1)
Line 2:    for (j=1; j<91; j=j+1)
Line5:              Gx = ((p[i+1][j-1]-p[i-1][j-1])+((p[i+1][j]-p[i-1][j])<<1)+(p[i+1][j+1]-p[i-1][j+1]));
Line6:              Gy = ((p[i-1][j-1]-p[i-1][j+1])+((p[i][j-1]-p[i][j+1])<<1)+(p[i+1][j-1]-p[i+1][j+1]));
Line7:              abs_gx = (Gx[10]?(~Gx+1):Gx);
Line8:              abs_gy = (Gy[10]?(~Gy+1):Gy);
Line9:              sum = (abs_gx + abs_gy);
Line10:  out[addrposition] = (|sum[10:8]?8'hff:sum[7:0]);
Line11:    end
Line12: end
```

Above pseudocode consists of two *for* loops, *Line1* and *Line2*, required to access every pixel of source image. In *Line3* and *Line4* values of horizontal and vertical gradient G_x and G_y are evaluated. Now to calculate the gradient value G magnitude of G_x and G_y is required (refer Equation (viii)). After getting absolute values, sum of these will give 11 bit binary number because signed extension are considered while designing hardware. After limiting the values to 255, final output pixel is saved in out[addrposition] array. Basic hardware design of pseudocode is shown in Figure 3.9.

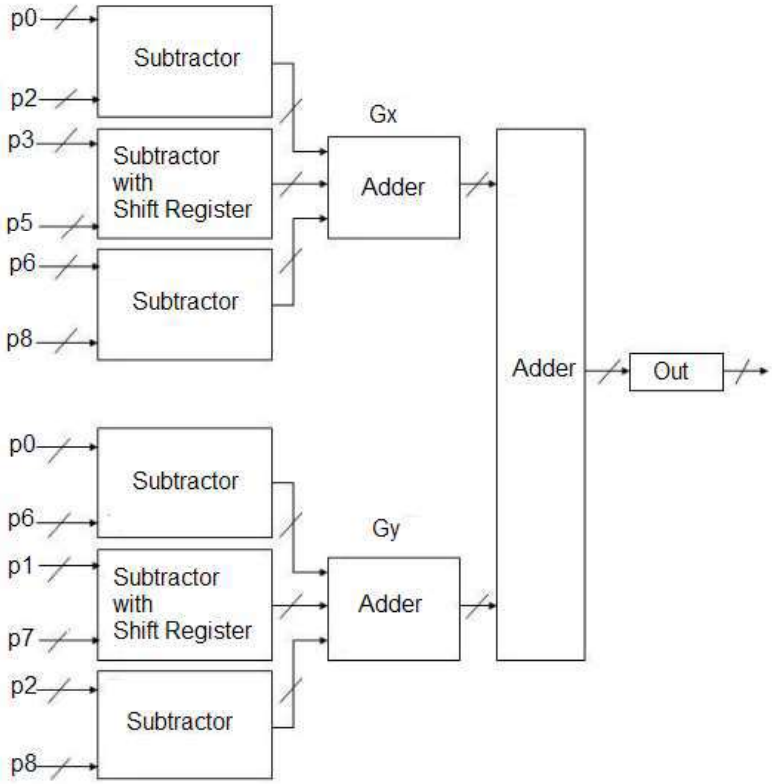


Figure 3.9: Hardware design for sobel operator code

CHAPTER

4 INTRODUCTION TO VGA CONTROLLER

Video Graphic Array, popularly known by its short form VGA. It is display standard developed by IBM and introduced in 1987. VGA provides 640x480 resolution color display with a refresh rate of 60 Hz and 16 colors displayed at a time [17].

A VGA display basically has three color signals *red*, *green* and *blue* where, VGA image is controlled by two signals – horizontal sync (*hsync*) and vertical sync (*vsync*). The *hsync* signifies the first and last of a pixel line with a negative pulse in each case as shown in Figure 4.1. All the pixels are sent in a 25.17 µs time window in a 31.77 µs space between the *hsync* and *vsync* pulses. Image is dark where the blank spaces are defined i.e. no data has been sent [17].

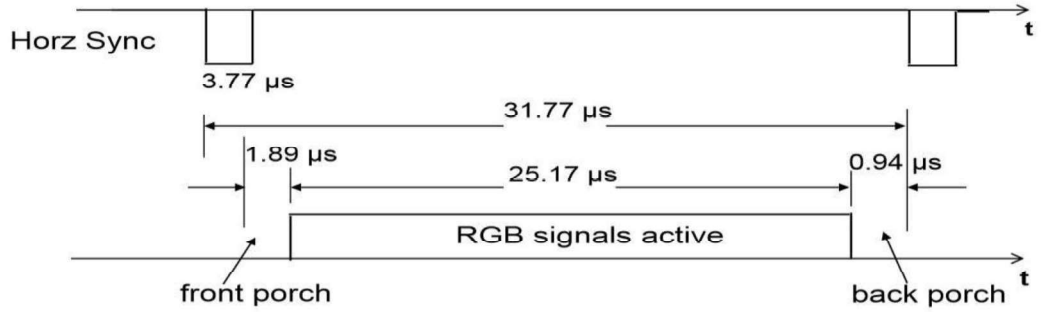


Figure 4.1: Horizontal sync timing diagram

The *vsync* is as same as to the *hsync* except, here the negative pulse signifies the first and last of each frame as a whole and the complete frame is displayed in a 15.36 ms time window in the space between pulses, i.e. 16.784 ms [17].

Figure 4.2 depicts the basic horizontal and vertical display timings. A basic idea of VGA display operation is shown with respect to the VGA control timings.

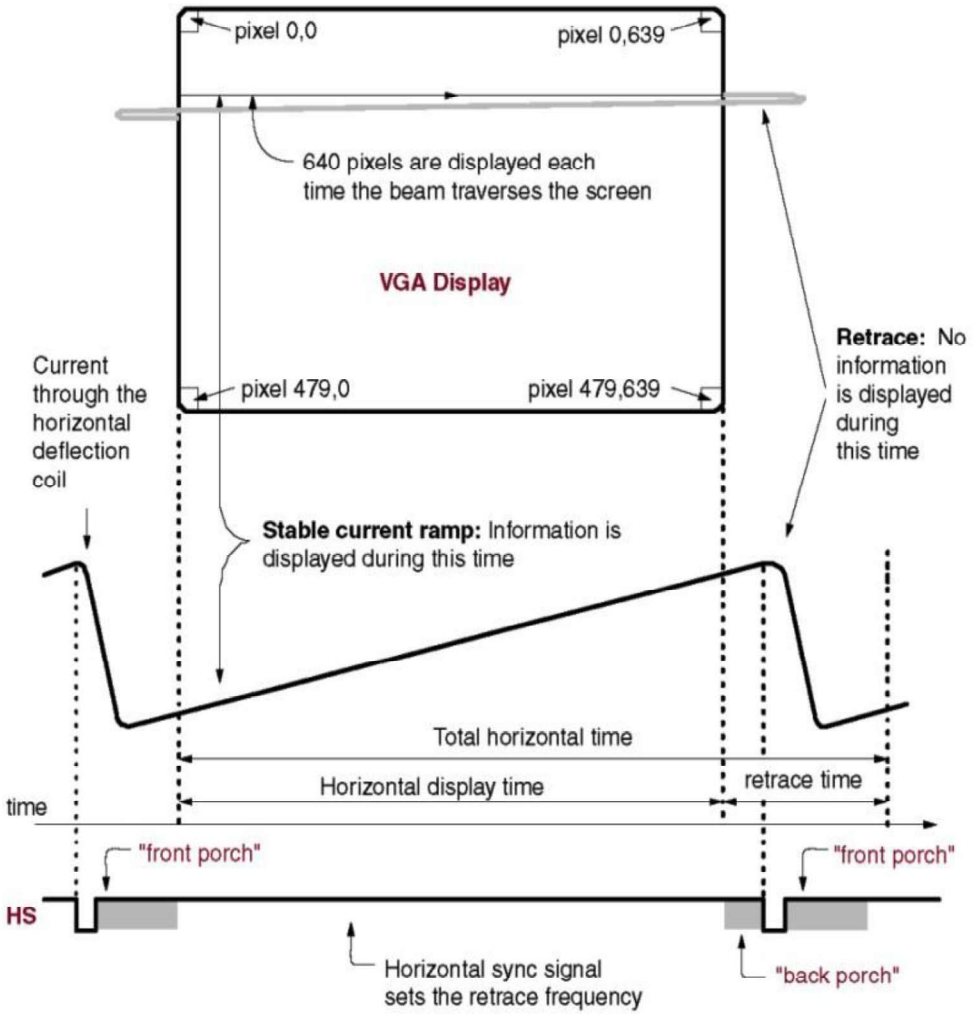


Figure 4.2: Display timing diagram [17]

4.1 BASIC PIXEL TIMINGS

If there is a time window of 25.17 µs to handle all of the pixels, then some calculations has to be done so that FPGA can display correct data in available time. For example, for a 640x480 VGA display, 640 pixels have to be sent to the monitor in just 25.17 µs. A simple calculation shows that for each pixel we need 25.17 µs/640 = 39.328 ns of time period. If our clock frequency is 50 MHz on the FPGA, then that gives a minimum clock period of 20 ns, which can be achieved using a relatively standard FPGA. The typical value of refresh rate used in present monitors is fixed at 60 Hz and to achieve this refresh rate on a 640x480 pixel screen, calculation is done as follows [17]:

Pixel Rate = (Total Horizontal Pixels * Total Vertical Lines *Number of screens/second)

= 800 * 525 * 60 = 25 MHz

Table 4.1: 640x480 Mode VGA Timing [9]

Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_s	Sync pulse	16.784 ms	416800	521	32 us	800
T_{disp}	Display Time	15.36 ms	384000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1600	2	3.84 us	96
T_{fp}	Front porch	320 us	8000	10	640 ns	16
T_{bp}	Back porch	928 us	23200	29	1.92 us	48

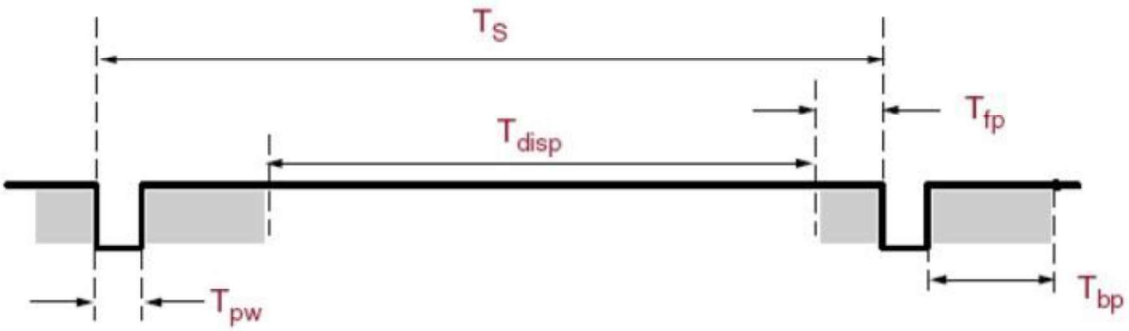


Figure 4.3: VGA control diagram

4.2 HDL IMPLEMENTATION OF THE VGA CONTROLEER

Implementation steps are as follows:

1. Special mod-800 counter and a decoding circuit are used to generate sync signal.
2. Counting was started from the beginning of the display region so that the output of the counter can be used as the horizontal (x-axis) coordinate. Thus giving us the *hsync* signal.
3. The same can be done to generate a vertical (y-axis) coordinate by using a counter and a decoding circuit. But instead of mod-800 we use mod-525 counter here. And this constitutes our *vsync* signal.

4. *hdisplay* signal is used to indicate whether the current horizontal is in the displayable region or not. It is asserted when the count is smaller than 640.
5. *vdsiplay* signal is used to indicate whether the current vertical coordinate is in the displayable region or not. It is asserted when the line count is less than 480.
6. 50 MHz clock is used in the operation which is available on board of FPGA.
7. As shown in previous calculations pixel clock is just half of the system clock i.e. 25 MHz. A mod-2 counter is used to generate pixel clock.

4.3 UNDERSTANDING THE INTERFACING DETAILS WITH FPGA

hsync, *vsync*, *red*, *green* and *blue* are the five active signal for VGA port present on the FPGA Spartan 3e kit. VGA signal is analog in nature so we need a digital to analog convertor but, we are working on Spartan 3e starter kit which uses only one bit per color, so we do not require any digital to analog convertor. We have three color signals which are *red*, *green* and *blue*, so we can have eight different color combinations as shown in Table 4.2. For that we need to give proper binary input combinations to VGA port [9].

Table 4.2: Three Bit VGA Combination [9]

Red (R)	Green (G)	Blue (B)	Resulting Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

CHAPTER

5 LITERATURE REVIEW

Chou C. *et al.*, 1993[18] implemented a digital filter algorithms based on FPGAs. In their paper they have explained, why a general purpose DSP implementations often lack the performance, and ASIC limitations. Their examples of IIR and FIR filter implementations illustrate that the FPGA approach is both flexible and provides performance comparable or superior to traditional approaches.

Chan S.C. *et al.*, 1994[19] proposes a flexible programmable image processing system, PIPS using FPGA. For the integration of FPGA and DSP to handle bit-level arithmetic operations found in image processing applications. The versatility of the system is shown and explained for the implementation of a 1D median filter.

A. DeHon, 2000[20] performs various tests on several applications and finds that FPGAs are between 10 and 1000 times faster than available Pentiums for image processing. He concluded that FPGAs are well suited for real-time image processing applications where there is the opportunity to exploit data parallelism.

Crookes D. *et al.*, 2000[21] presents the design of an FPGA based image processing coprocessor (IPC) along with its high level programming environment. The coprocessor instruction set is based on a core level containing the operations of image algebra. Architectures for user-defined compound operations can be added to the system. Possibly the most significant aspect of their work is that it opens the way for image processing application developers to exploit the high performance capability of a direct hardware solution, while programming in an application oriented model.

Gribbon K. T. and Bailey D. G., 2004 [22] presents a novel FPGA implementation of a barrel distortion correction algorithm with a focus on reducing hardware complexity. They explains the difficulties which are present while conversion from a software algorithm to one that runs in hardware. These include the inability to do offline

processing, data bandwidth constraints and the need to minimise logic gate count. The use of a LUT with interpolation can reduce the complexity of the hardware design without significant loss of precision compared to calculating values at run-time.

D.T. Saegusa *et al.*, 2008[23] compares the performance of FPGAs with those processors using three applications in image processing; two-dimensional filters, stereo-vision and k-means clustering, and make it clear how fast is an FPGA in image processing, and how many hardware resources are required to achieve the performance.

Praveen Vanaparthy *et al.*, 2013[24] presents real time hardware image enhancement techniques using FPGA. They focus on implementation of image enhancement algorithms like brightness control, contrast stretching, negative transformation, thresholding, filtering techniques on FPGA that have become a competitive alternative for high performance digital signal processing applications.

Mohammad I. *et al.*, 2013[25] describes an efficient FPGA based hardware design for different image processing, enhancement, and filtering algorithms. They explained the advantage of a reprogrammable device for continuous changes and optimizations on the hardware.

CHAPTER

6 METHODOLOGY

In this chapter the basic flow of present work has been explained step by step. We have adopted a divide and conquer policy to achieve the goal of image-processing using FPGA. The work has been organized into two phases. The first phase consists of FPGA implementation of basic building blocks, like VGA controller for display and memory controller for BRAM based video memory. The second phase consists of development of the image processing algorithm. In this phase, we take care of HDL implementation of Sobel edge detection algorithm.

We need an image source for the rest of the project development. Hence, we decided to use the memory that comes with FPGA-Block RAM (BRAM) as the image source. Specifically, the BRAM available in Xilinx Spartan3e board is used for this purpose. The advantage is that we can initialize the Block RAM with an image using Xilinx ISE software.

6. HARDWARE AND SOFTWARE

The Spartan3e FPGA Development board from Digilent used for this project is shown in Figure 6.1. ModelSim is used for RTL simulation. Xilinx ISE is used for synthesis. The impact tool that comes with Xilinx ISE is used for downloading the bit stream into the Spartan 3e FPGA.

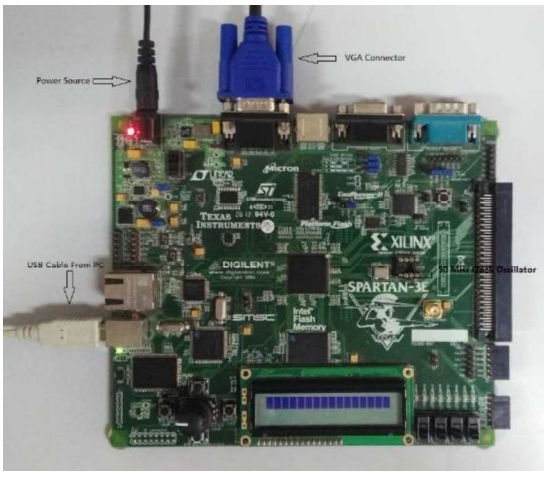


Figure 6.1: Spartan 3e FPGA board

6.2 DEATILED FLOW CHART

Figure 6.2 shown below depicts the step by step flow of proposed work.

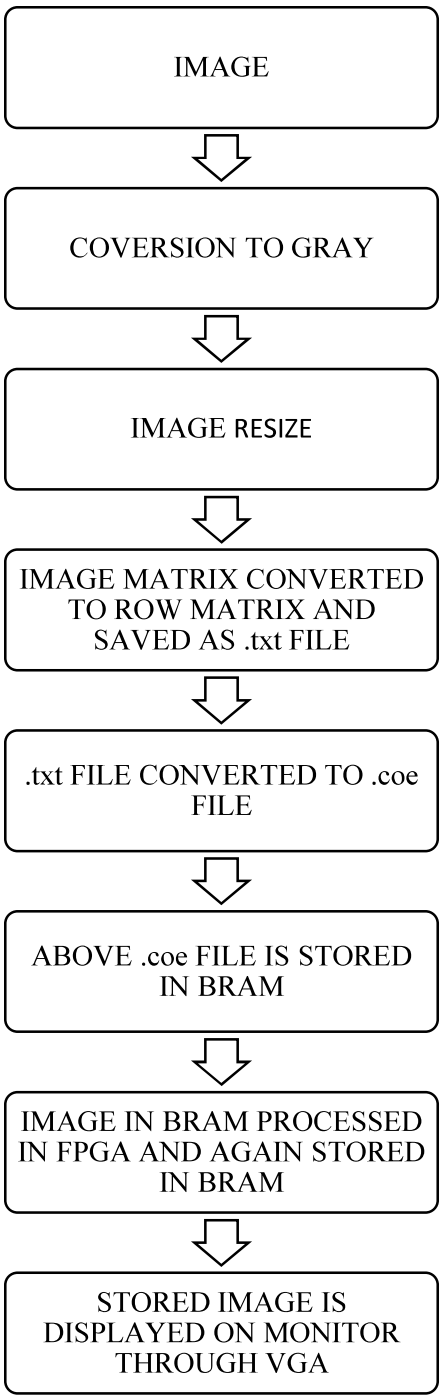


Figure 6.2: Basic flow diagram

The above flow chart can be segmented into three main groups as shown in Figure 6.3:

1. Pre-Processing
2. Processing Image Pixels on FPGA
3. Display image

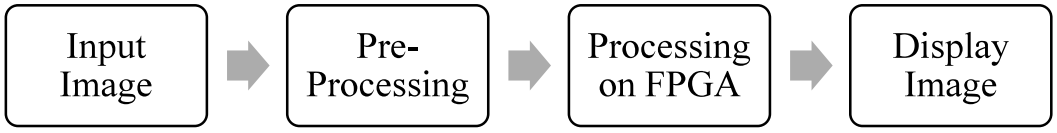


Figure 6.3: Categorization of the flow chart

Step1: Pre-Processing

Consider an RGB image of size 256x256x3, but due to limited memory available on FPGA board it has to be resized to 90x90 gray image by means of any software (we use MATLAB). After resizing image matrix (size 90x90) is converted into a column matrix of 8100x1 and corresponding pixels are saved on disk as a .txt file. Then this .txt file converted to a .coe file which will be used when assigning these pixel values to BRAM of FPGA board, created with the help of Xilinx IP Core Generator.

Step2: Processing Image Pixels on FPGA

It is not advisable to use an integrated image system with the FPGA, rather more efficient way is to store the image in Random Access Memory (RAM) and retrieve it frame by frame. To implement this we use BRAM available on FPGA. Image pixels which are stored in BRAM of FPGA board are read for further processing. These pixels are then sent to basic image enhancement algorithm code using Verilog and programmed on FPGA board. Final processed pixels are again stored in other BRAM.

Step3: Display Image

The implementation of this step requires at most care while writing down interface code. To display an image on VGA monitor we have to write BRAM-to-VGA interface code. While writing VGA interface code we have to take care of clock timings and pixel refresh rate very carefully.

6.3 COE FILE GENERATION

Following are the steps to generate COE file:

1. Take an image.
2. Resize it.
3. Save the image as column matrix (8100x1) containing pixel values in the txt format to the disk.
4. .txt file extension changed to .coe to create a COE file.

6.4 BITMAP GENERATION LOGIC USING BRAM

The pixel generation code generates the bits which have to be sent to the RGB signal of VGA port and needs to be displayed on the monitor. For the large number of bits which has to map, all the pixels to the memory should be present on board for fast and economical execution. In the bit-mapped scheme, all the pixels or data, is saved or mapped to BRAM beforehand, with the help of Xilinx IP Core Generator. A BRAM module is created using Xilinx IP Core Generator. Then COE file we have created is mapped on to the BRAM.

6.4.1 BRAM Creation

BRAM are different memory modules in FPGA devices separated from regular logic cells. Thus, there is no need of any memory controller. We have two options here with BRAM it can be single ported or dual ported. Each BRAM consists of 18K by 1 of memory array. The Spartan 3XCS500 device has 12 block RAMs, totaling 216K bits.

Following are the steps to create BRAM:

1. Select design then select Create New Source.
2. Select IP (CORE Generator & Architecture Wizard) from the list.
3. Mention any file name you wish to use. In present case it is “mybram”.
4. Then click on Next as shown in Figure 6.4.

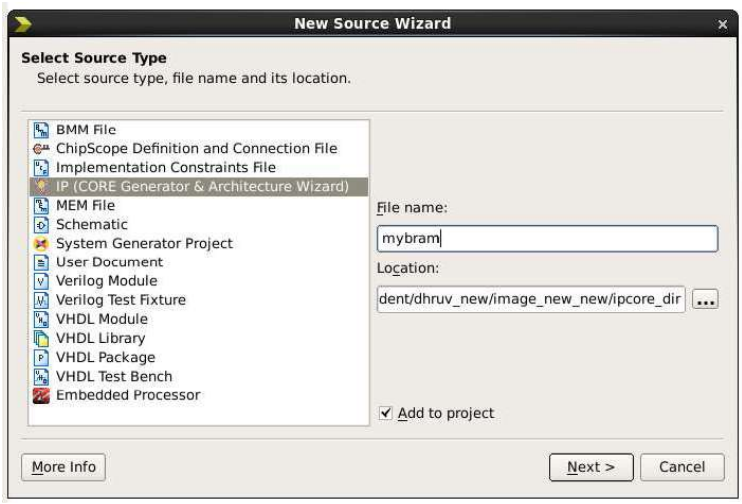


Figure 6.4: New source wizard – select source type

1. After selecting next, a new window will open as shown below.
2. Go to Memory Interface Generator folder.
3. Then go to RAMs & ROMs folder.
4. Then select Block Memory Generator.
5. Then click Next as shown in Figure 6.5.

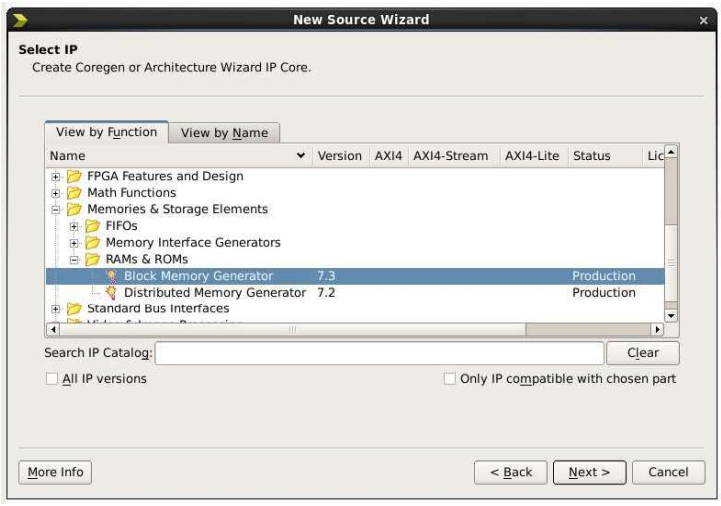


Figure 6.5: New source wizard – select IP

6. In the next window we have to select width and depth. These are 8 and 8100, respectively. This is because, there is 8 bit data (rgb), one for each color signal, and 90x90=8100 pixels have to be stored.
7. Click next after it as shown in Figure 6.6.

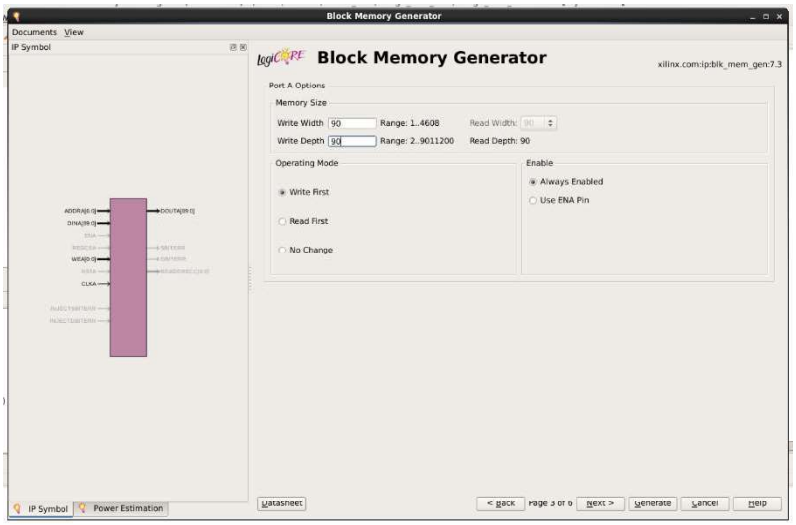


Figure 6.6: Block memory generator wizard- I

8. In the next window, load the COE file which has to be mapped on the BRAM.
9. Click next after it as shown in Figure 6.7.

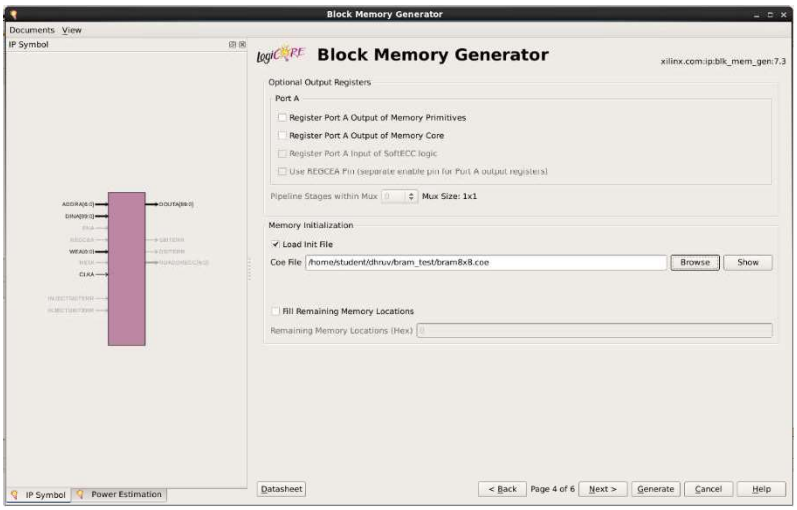


Figure 6.7: Block memory generator wizard- II

Select Generate after it to create a BRAM block. This process will take some time. After the process completes a source is added to your design hierarchy, see Figure 6.8.

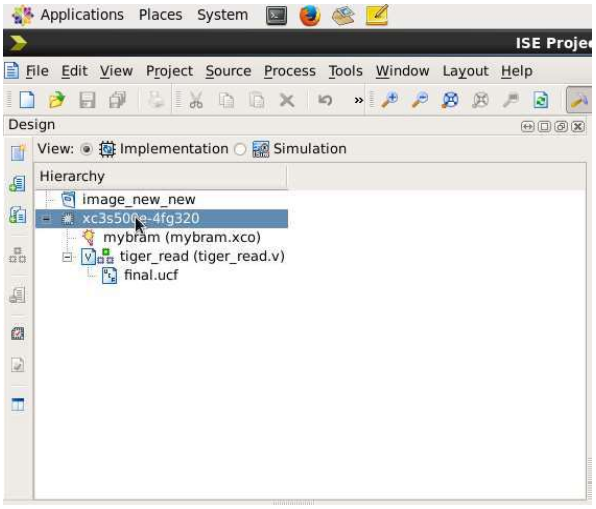


Figure 6.8: Design hierarchy- I

6.5 PROGRAMMING FPGA

Next step is to program our hardware which is Spartan 3e FPGA board. We need to check whether our designed code is synthesizable or not. To check Verilog code synthesizability it need to pass certain steps like place and route check, mapping check etc. Successful simulation doesn't guarantee that your code is synthesizable. Step by step procedure is explained below to check code for synthesizability and program FPGA board.

Steps for programming a FPGA.

1. Creat a new project.
2. Creat a verilog module and User Constraint File (UCF).
3. Generat post synthesis simulation model and implemented design
4. Generat Programming file.
5. Confirgur FPGA using IMPACT.
6. Program the FPGA.

1. Create a new Project:

Open the Xilinx ISE 14.7 setup and choose create a new project. Then select type of evaluation board. Our project details are shown in below Figure 6.9.

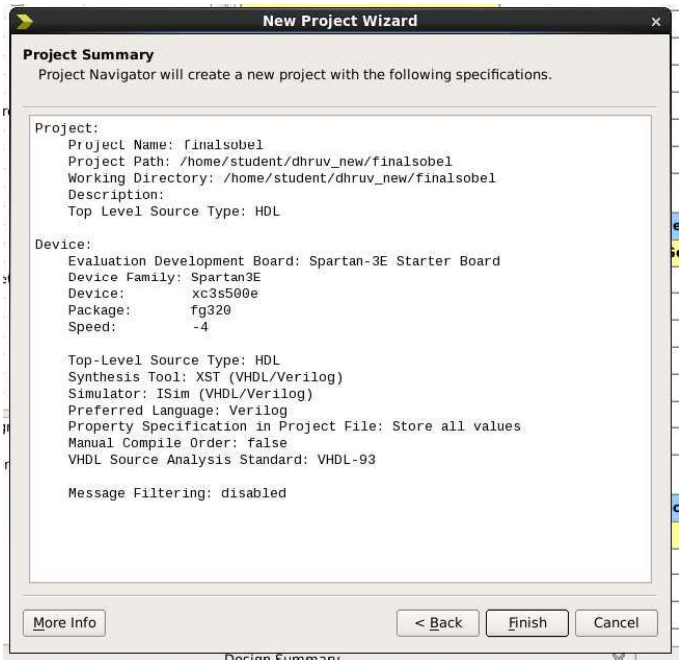


Figure 6.9: Project details

2. Create a verilog module:

Click on the icon showed in the below figure to create a new module, then select a verilog module from the list and name it accordingally. In present case it is finalsobel. Write down your whole code in the finalsobel.v file and then click on check syntax as shown in Figure 6.10. If all are correct a check sign will be visible near it.

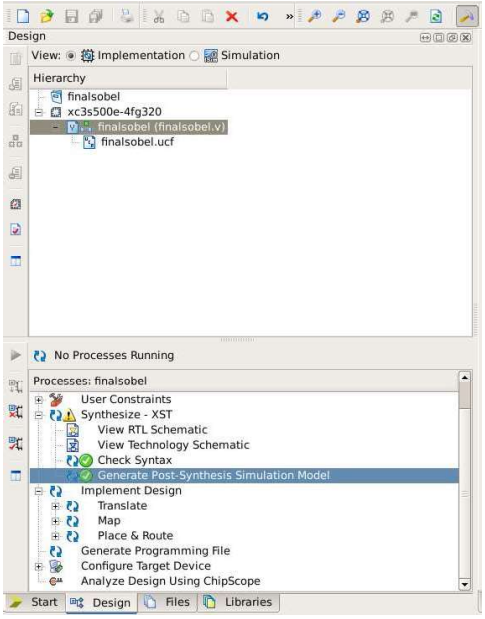


Figure 6.10: Design heirarchy- II

3. Create a UCF file.

To Program a FPGA you need an UCF in which we have to mention which of our input and output is associated to which PIN of FPGA board. In present work UCF is just like as shown below:-

```
##USER CONSTRAINTS
NET "rst" LOC = N17;
NET "clk" LOC = C9;
NET "hsync" LOC = F15;
NET "vsync" LOC = F14;
NET "r" LOC = H14;
NET "g" LOC = H15;
NET "b" LOC = G15;
```

In which *rst*, and *clk* are the inputs which is mapped to N17 and C9 FPGA pin, and *hsync*, *vsync*, *r*, *g*, *b* are the outputs which are mapped to the pins as mentioned above.

N17 is a switch having two position LOW and HIGH. When LOW, it will give '0' to the rst and '1' when high. Where as C9 is the location of available 50 MHz clock which is used in our code to genertate 25 Mhz clock required for pixel refreshing. Function of rest nets and their associated locations are explained in the Chapter 4.

4. Generating post synthesis simulation model and implementing design:

This is very important step before programing FPGA, when we run this option it check all the resource which your design is going to use. If you use any operator or function in your code which is not synthesisable, it will give you an error. It also give you an eror when you exhaust available resources i.e. memory and LUT's. A check mark will appear when your design passes all checks as shown in Figure 6.11.

After post synthesis model is ready, design should pass translation, mapping and placement routing test. If there is any error while running these three test design will

fail synthesis test. If there are no error are Xilinx Synthesis Tool (XST) will show a synthesis report with “Synthesis Passed” message.

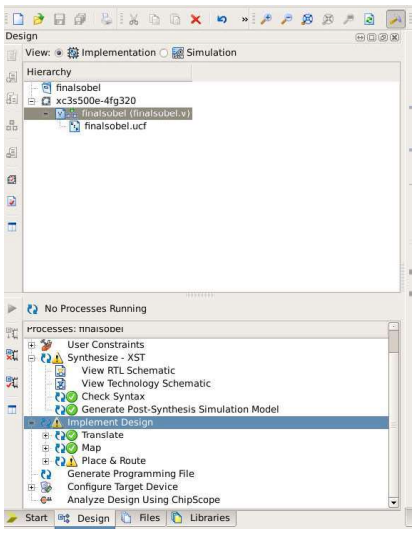


Figure 6.11: Generate post-synthesis simulation model

5. Generating programming file:

Verilog code is not used to program FPGA. It is only used to generate bit file. A bit file contains all constants, unrolled loops and data which is generated by the XST. This bit file is programmed into FPGA. Click on the option generate programming file from the list shown in Figure 6.12, and when check comes you will notice there will be a bit file created and saved to your working diretory.

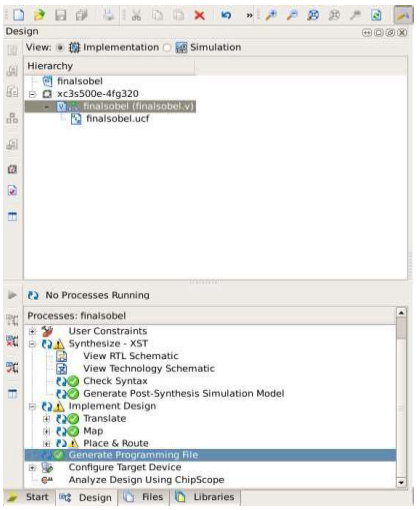


Figure 6.12: Genrate programming file

6. Confirguring FPGA using IMPACT Confirguration mode:

As soon as bit file is created you can program FPGA. Click on Confirgure target device and ISE IMACT configuring window will get open as shown in Figure 6.13. First you need to do select boundary scan so that your device is detected by the software. Make sure power supply is ON while doing any further process.

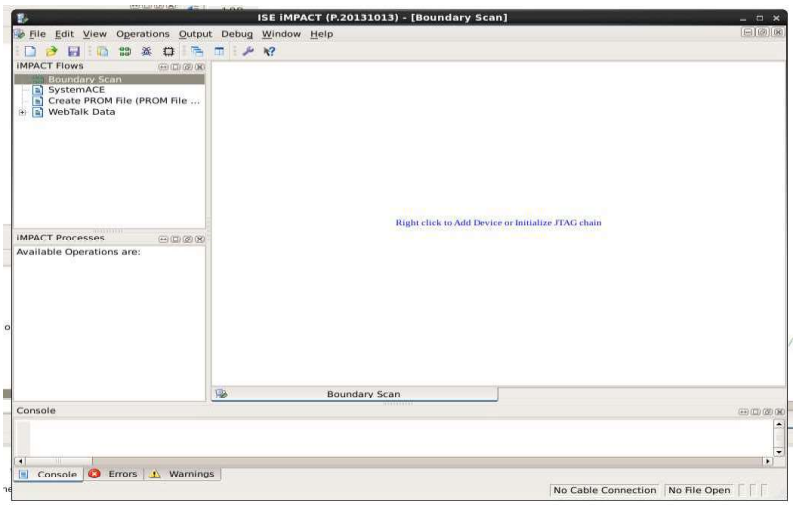


Figure 6.13: ISE impact boundary scan

After your device is detected by the ISE IMPACT, right click on the workspace and select Initialise JTAG chain. It will show Identify succeded once identification is done. As shown in below Figure 6.14.

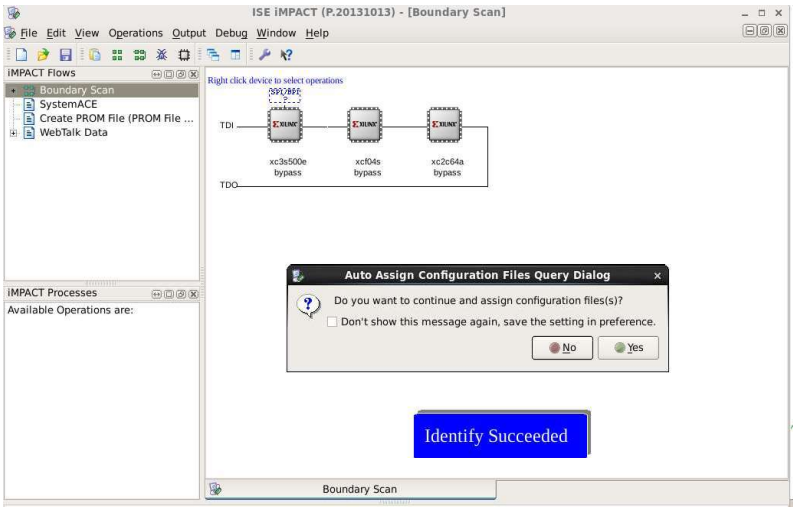


Figure 6.14: JTAG chain initialization

Once JTAG chain is initialized start assigning configuration files. Figure 6.15 shows the selection of bit file from the directory, in present case it is finalsobel.bit.

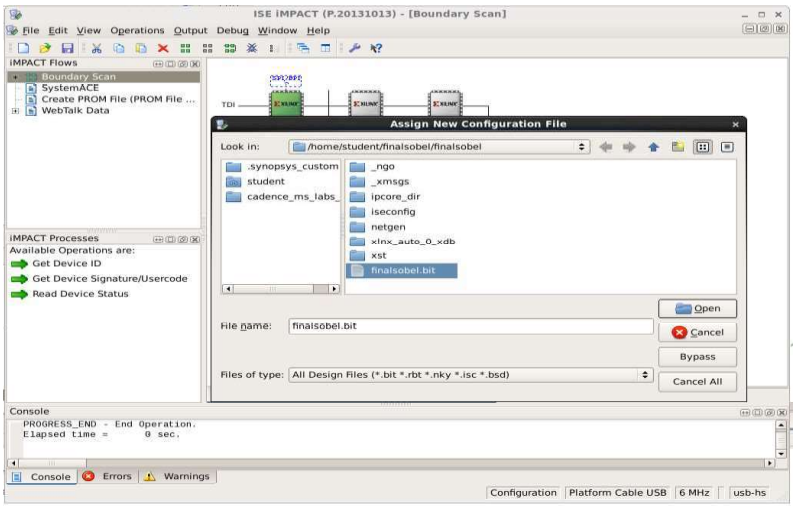


Figure 6.15: Selection of bit file from disk

Once bit file is assigned last step is to program FPGA. Double click on the Program option from the left list, it will show Program Succeeded once it is complete as shown in Figure 6.16.

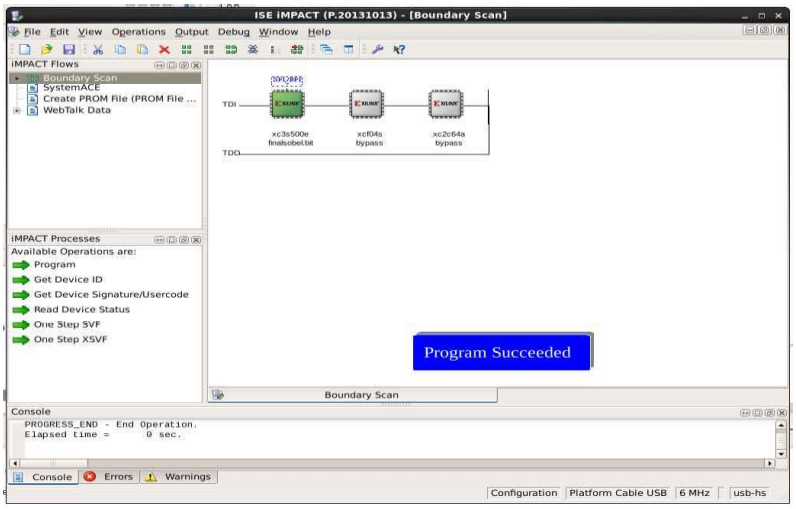


Figure 6.16: FPGA programming successful

CHAPTER

7 SIMULATIONS AND RESULT

Sobel edge detection algorithm is implemented on Spartan XC3S500 Kit by writing Verilog code. The Verilog code is compiled, synthesized and programmed to Xilinx Spartan 3e Starter Kit using Xilinx ISE Design Suite. The used image is 90 x 90 pixels with 256 gray levels. The utilization summary of FPGA hardware resources is provided in Table 7.1. This FPGA based Sobel edge detection design operate at 25 MHz clock frequency.

Table 7.1: Device Utilization Summary

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total No. Of Slice Register	58	3840	1%
No. used as Flip-Flops	57		
No. used as Latches	390	3840	10%
Logic Distribution			
No. of Occupied Slices	284	1920	14%
No. of Slices Containing only Related Logic	284	284	100%
No. of Slices Containing only Unrelated Logic	0	284	0%
Total No. of 4 I/P LUTs	418	3890	10%
No. Used as Logic	393		
No. Used as route-thru	25		
No. of Bounded I/Os	7	173	4%
IOB Flip-Flops	2		
No. of DCLKs	2	8	25%
Total Equivalent Gate Count for Design	3412		
Additional JTAG Gate Count for IOBs	336		

We tested our design and present the results for different images. Figure 7.1 shows the work space setup.

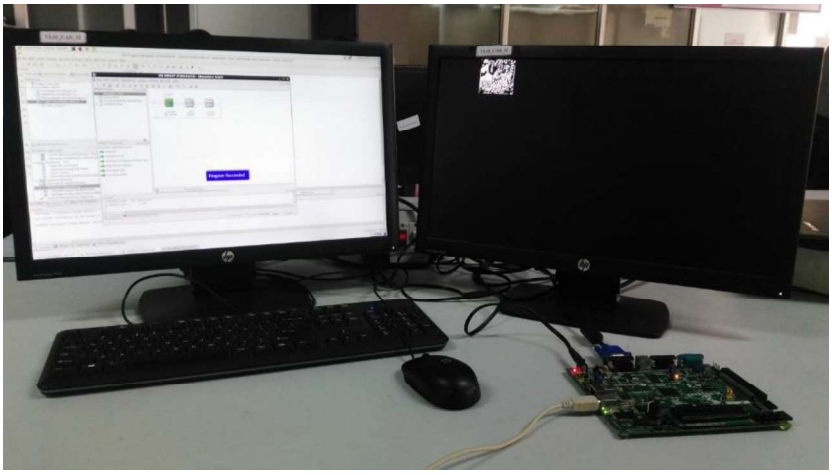


Figure 7.1: Set up image

Figure 7.2 shows the simulation result of 8x8 image used to test BRAM read write design.

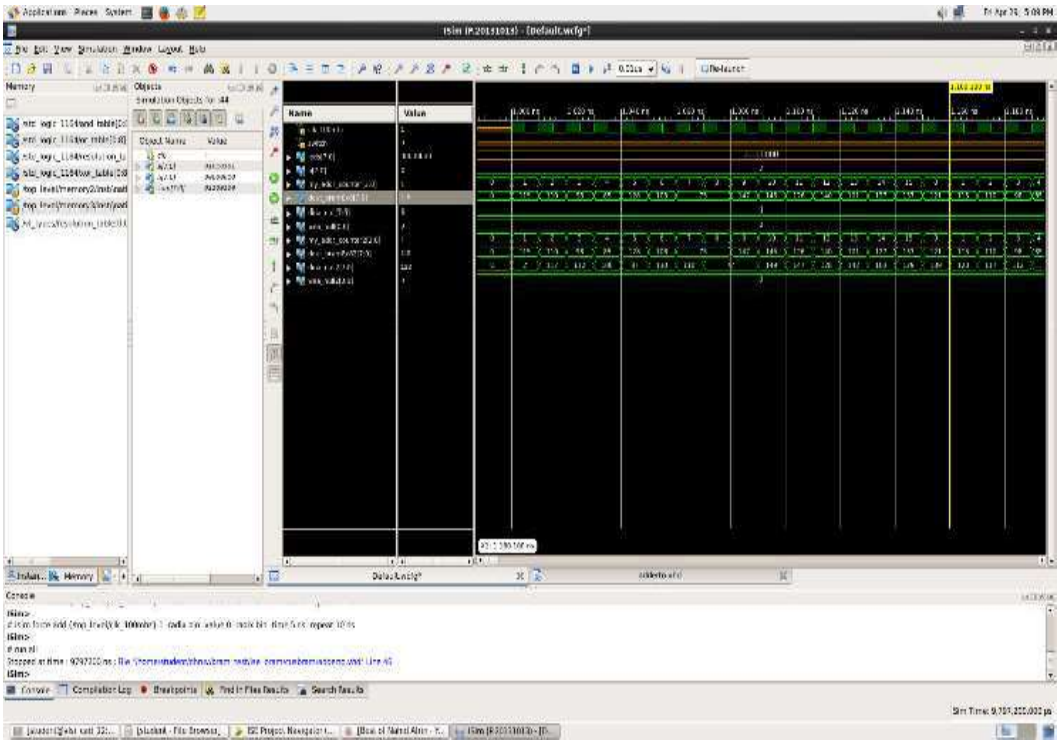


Figure 7.2: Simulation result for 8x8 image read from BRAM

In Figure 7.3, a 90 x 90 binary image is displayed on VGA monitor to test our BRAM to VGA interfacing design.

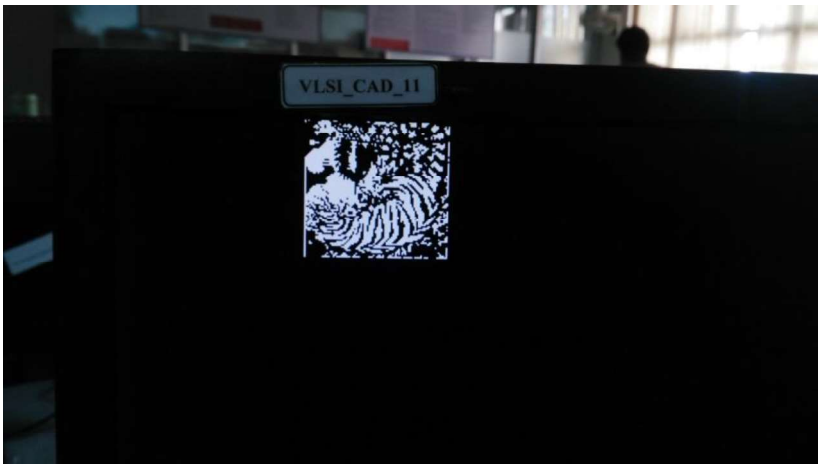


Figure 7.3: Binary image displayed on VGA monitor

Figure 7.4-7.7 shows the simulation result for Sobel operator. Where, Figure 7.4-7.5 display the input image pixels value and Figure 7.6-7.7 display the output edge detected image pixel values.

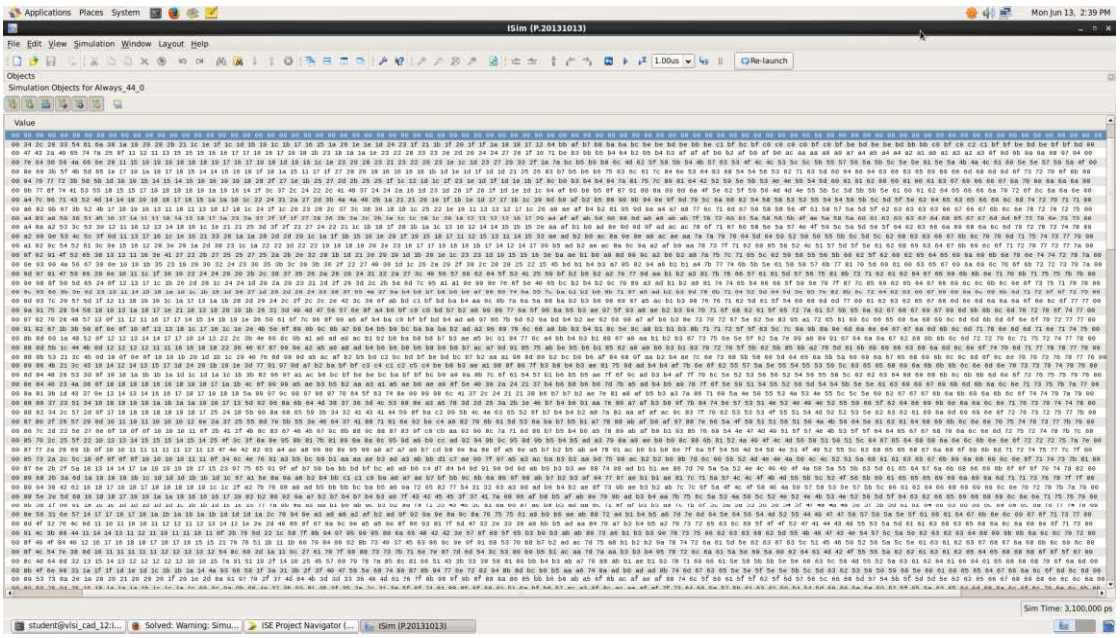


Figure 7.4: Simulation showing pixel values for input image-I

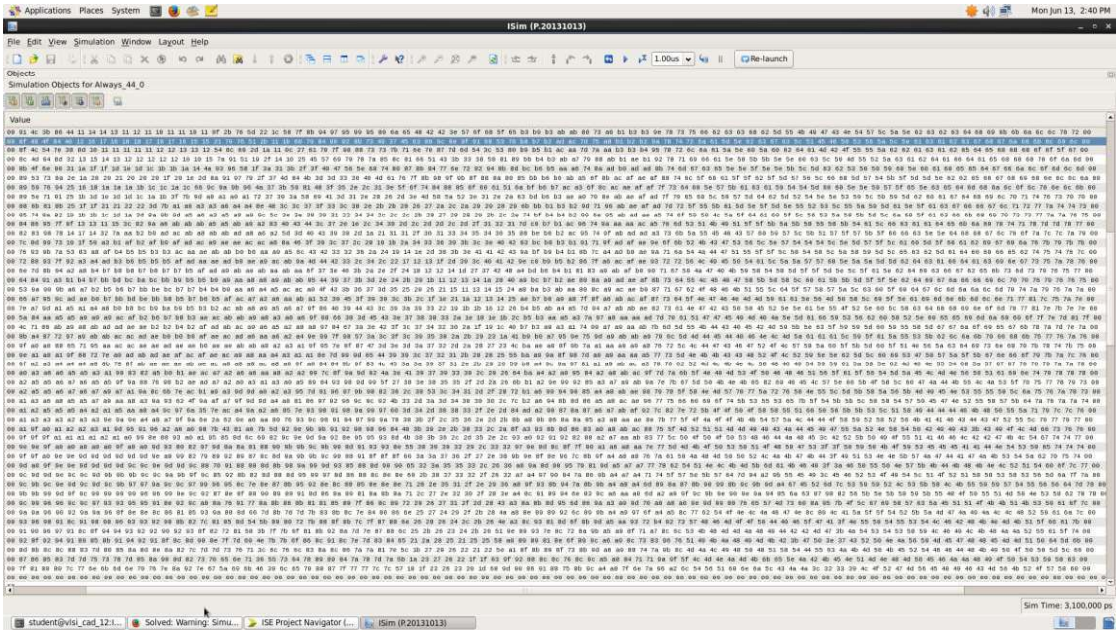


Figure 7.5: Simulation showing pixel values for input image-II

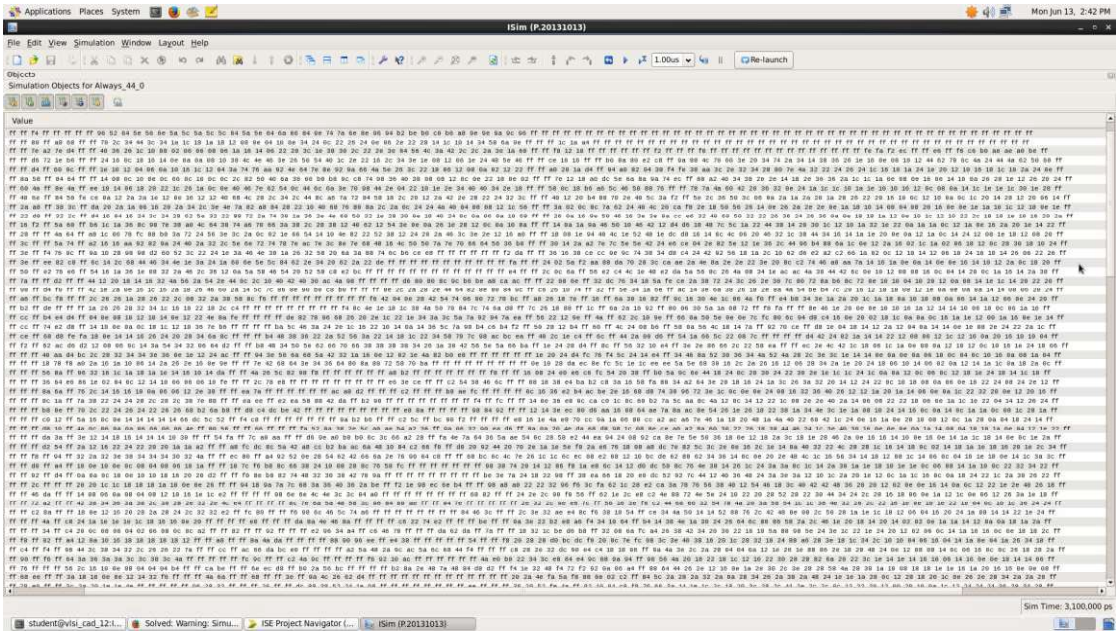


Figure 7.6: Simulation showing pixel values for output image-I

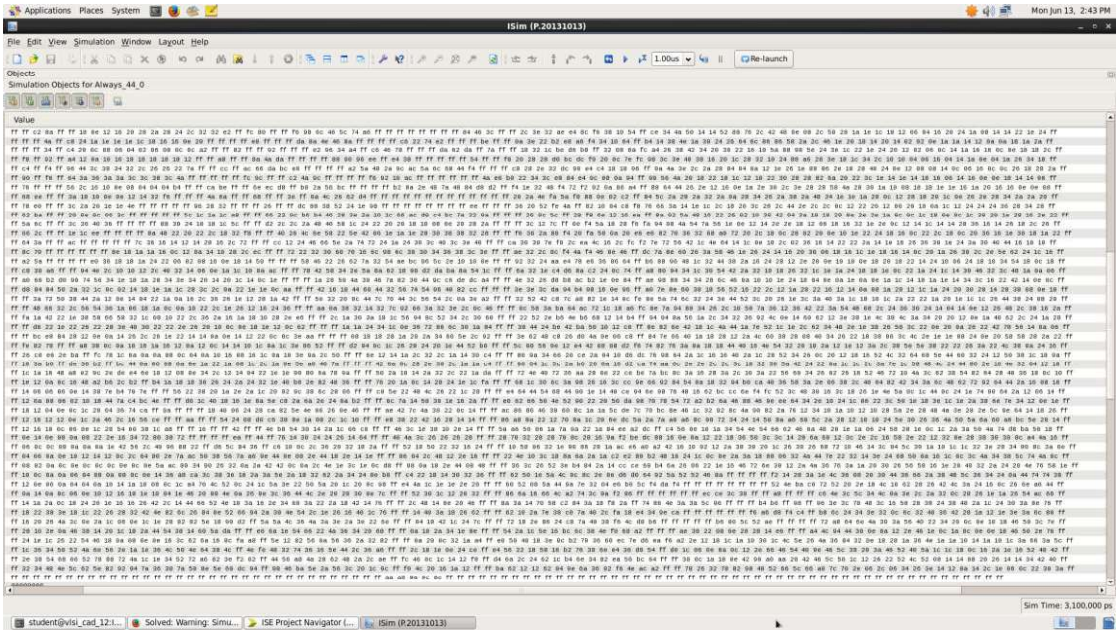


Figure 7.7: Simulation showing pixel values for output image-II

Finally we get a gradient image and to convert it to binary image we need to give a thershold as shown in Figure 7.8.

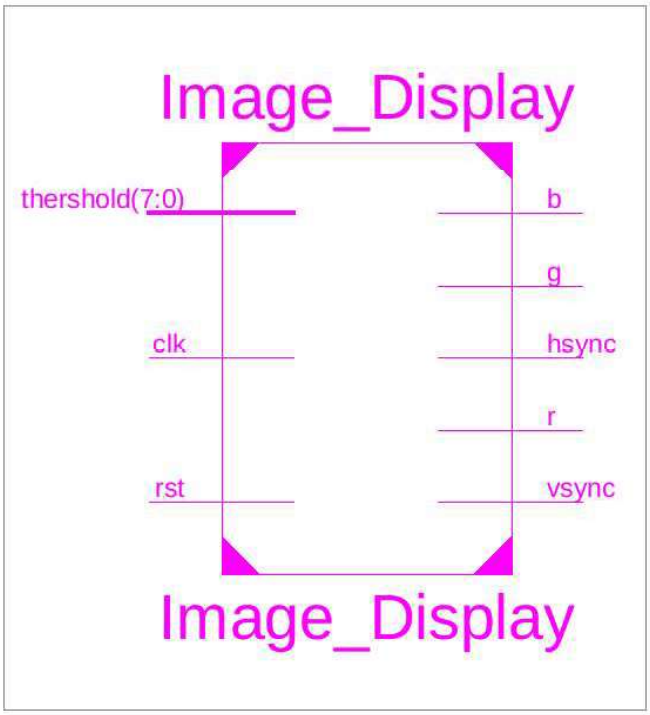


Figure 7.8: Top level RTL of design used to display binary image on VGA

(a)

(b)

Figure 7.9: (a) Input image and (b) Output image on VGA monitor

Figure 7.9(a) shows the 90 x 90 input image and output edge detected image which displayed on VGA monitor is shown in Figure 7.9(b).

We test our design for more images and compare it with the MATLAB implementation of Sobel edge detection algorithm. Figure 7.10 depicts the comparison result.

(a)

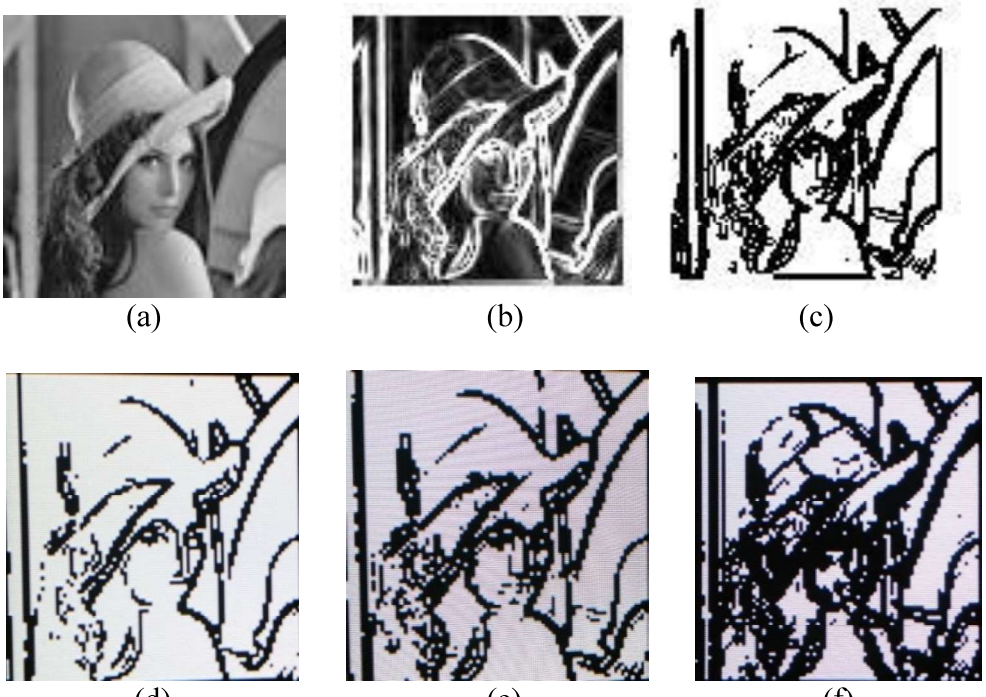
(b)

(c)

(d)

Figure 7.10: (a) Input image, (b) Sobel gradient image from MATLAB, (c) Sobel edge detected image by MATLAB and (d) Sobel edge detected image on VGA monitor

We can say, we are getting very fine edges as compared to MATLAB implementation. We also test our design for different threshold values, result are shown in Fig 7.11.



(a) (b) (c)

(d) (e) (f)

Figure 7.11: (a) Input image, (b) Sobel gradient image, (c) Sobel edge detected image, Sobel edge detected image on VGA monitor with thershold (d) 150, (e) 100 and (f) 50

CHAPTER

8 SUMMARY AND FUTURE SCOPE

In proposed work we have successfully implemented Sobel based edge detection technique on FPGA, which includes writing Verilog code for Sobel operator and BRAM to VGA interface. Hardware design for approximate method based Sobel edge operator is done. We used buffers i.e. frame buffer and row buffer to overcome the limitation of memory bandwidth while window operation.

The reusable nature of the Verilog design allow for its code to be reused in different designs easily. For example in present design we implemented 3x3 window architecture which can be reused in any algorithm that uses a pixel window method to compute its output. With few modifications in mask and a new edge detection technique can be implemented. One can easily instantiate Verilog code as component and port map statements into another Verilog design. Because of this, the applications for the code created can be used in many different image processing algorithms.

Another scope in this work can be operation on bigger image. We used 90x90 image because of limited memory availability on FPGA Spartan Series. One can use higher series of FPGA like Kintex and Artix. They have large amount of memory onboard which enable designer to operate on large images.

Despite these possible improvements, this thesis is considered to be a success. The knowledge and experience gained from completing this project will certainly be helpful in future designs.

REFERENCES

[1] Iuliana Chiuchisan, Marius Cerlinca, Alin-Dan Potorac and Adrian Graur, "Image enhancement methods approach using verilog hardware description language," *11th International Conference on Development And Application Systems*, Romania, 2012.

[2] P.Kalyan Chakaravathi, "FPGA based architecture for real time edge detection," *Proceedings of 2015 Global Conference on Communication Technologies*, 2015.

[3] C. T. Johnston, K. T. Gribbon and D. G. Bailey, "Implementing image processing algorithms on FPGAs," *Institute of Information Sciences & Technology, Massey University Private Bag 11-222, Palmerston North, New Zealand*, 2005.

[4] All about FPGAs, http://www.eetimes.com/document.asp?doc_id=1274496

[5] Xilinx Inc., http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm

[6] National Instruments, http://www.ni.com/fpga/

[7] Altera MAX10 FPGA Family Overview: https://www.altera.com/products/fpga/max-series/max-10/overview.html

[8] Xilinx DS312 Spartan-3e Family Overview: http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[9] Xilinx UG130 Spartan-3e Starter Kit User Guide: http://www.xilinx.com/support/documentation/boards_and_kits/ug130.pdf

[10] Dhanabal R, Bharathi V and S.Kartika, "Digital image processing using sobel edge detection algorithm in FPGA," *Journal of Theoretical and Applied Information Technology*, Vol. 58 No.1, e-ISSN: 1817-3195, p-ISSN: 1992-8645, 2013.

[11] D. Ziou and S. Tabbone, "Edge detection techniques - an overview," *International Journal of Pattern Recognition and Image Analysis*, Vol.8, pp. 537–559, 1998.

[12] Raman Maini and Dr. Himanshu Aggarwal, “Study and comparison of various image edge detection techniques,” *International Journal of Image Processing,* Vol.3, 2000.

[13] Girish N.Chaple, R.D.Daruwala and Manoj S.Gofane, “Comparision of robert, prewitt, sobel operator based edge detection methods for real time uses on FPGA,” *International Conference on Technologies for Sustainable Development*, 2015.

[14] O.R. Vincent and O. Folorunso, “A descriptive algorithm for sobel image edge detection,” *Proceedings of Informing Science & IT Education Conference (InSITE),* 2009.

[15] K. Babu Ravi Teja, Abhilash S. Warrier, Akshay S. Belvadi and Dhiraj R. Gawhane, “Design and implementation of neighbourhood processing operations on FPGA using verilog HDL,” *IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)*, Vol. 4, Issue 1, pp 75-80, e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197, 2014.

[16] Guangda Su, Jiongxin Liu, Yan Shang, Boya Chen and Shi Chen, "Theory and application of image neighborhood parallel processing," *Image Processing (ICIP), 2009 16th IEEE International Conference,* Vol.2, pp.2313, 2316, 2009.

[17] Design Recipes for FPGAs – A Simple VGA Interface, http://www.eetimes.com/document.asp?doc_id=1274613

[18] Chou C., Mohanakrishnan S. and Evans J., “FPGA implementation of digital filters,” *Proceedings of International Conference on Space Planes and Aircraft Technologies*, 1993.

[19] Chan S.C., Ngai H.O., and Ho K.L., "A programmable image processing system using FPGA," *Circuits and Systems, 1994 IEEE International Symposium*, Vol.2, pp.125, 128, 1994.

[20] A. DeHon, "The density advantage of reconfigurable computing," *IEEE Computer,* Vol. 33, pp. 41-49, 2000.

[21] Crookes D., Benkrid K., Bouridane A., Alotaibi K., and Benkrid A., “Design and implementation of a high level programming environment for FPGA-based image processing,” *IEEE Proceedings-Vision Image and Signal Processing*, Vol. 147, pp. 377-384, 2000.

[22] Gribbon K. T. and Bailey D. G., "A novel approach to real-time bilinear interpolation," *Second IEEE International Workshop on Electronic Design, Test and Applications, Perth, Australia*, pp. 126, 2004.

[23] D. T.Saegusa, T.Maruyama and Y.Yamaguchi, "How fast is an FPGA in image processing?," *Institute of Electronics, Information and Communication Engineers Technical Report*, Vol.108, pp.83–88, 2008.

[24] Praveen Vanaparthy, Sahitya.G, Krishna Sree and Dr.C.D.Naidu, "FPGA implementation of image enhancement algorithms for biomedical image processing," *International journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, Vol. 2, e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765, 2013.

[25] Mohammad I., AlAli, Khaldoon, M. Mhaidat, and Inad A. Aljarrah, "Implementing image processing algorithms in FPGA hardware," *IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, Vol. 32, pp. 1-5, p-ISSN: 2305-4799, 2013.

[26] D. Ziou and S. Tabbone, "Edge detection techniques - an overview," *International Journal of Pattern Recognition and Image Analysis*, Vol. 8, pp. 537–559, 1998.

[27] Najjar W. A., Böhm W., Draper B. A., Hammes J., Rinker, R. Beveridge, J. R. Chawathe M. and Ross, C., "High-level language abstraction for reconfigurable computing," *IEEE Computer*, Vol. 36, pp. 63-69, 2003.

[28] Chi-Jeng Chang, Pei-Yung Hsiao and Zen-Yi Huang, "Integrated operation of image capturing and processing in FPGA," *IJCSNS International Journal of Computer Science and Network Security*, Vol.6 No.1A, pp 173-179, 2008.

[29] Offen, R. J., "VLSI Image Processing," *London: Collins*, 1985.

601461009

by Dhruv Singhal

FILE	601461009_DHRUV.PDF (6.77M)		
TIME SUBMITTED	13-JUL-2016 01:02PM	WORD COUNT	8080
SUBMISSION ID	689379810	CHARACTER COUNT	38902

601461009

ORIGINALITY REPORT

23%	19%	16%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.eetimes.com Internet Source	2%
2	www.safe-tech.ca Internet Source	1%
3	eem.anadolu.edu.tr Internet Source	1%
4	www.memoireonline.com Internet Source	1%
5	Crookes, D., K. Benkrid, A. Bouridane, K. Alotaibi, and A. Benkrid. "Design and implementation of a high level programming environment for FPGA-based image processing", IEE Proceedings - Vision Image and Signal Processing, 2000. Publication	1%
6	www.celoxica.co.jp Internet Source	1%
7	www.altera.com Internet Source	1%
8	Chu. "VGA Controller I: Graphic", FPGA	1%