

12-17-2010

## Implementation of Separable & Steerable Gaussian Smoothers on an FPGA

Arjun Joginipelly  
*University of New Orleans*

Follow this and additional works at: <https://scholarworks.uno.edu/td>

---

### Recommended Citation

Joginipelly, Arjun, "Implementation of Separable & Steerable Gaussian Smoothers on an FPGA" (2010).  
*University of New Orleans Theses and Dissertations*. 98.  
<https://scholarworks.uno.edu/td/98>

This Thesis-Restricted is protected by copyright and/or related rights. It has been brought to you by ScholarWorks@UNO with permission from the rights-holder(s). You are free to use this Thesis-Restricted in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis-Restricted has been accepted for inclusion in University of New Orleans Theses and Dissertations by an authorized administrator of ScholarWorks@UNO. For more information, please contact [scholarworks@uno.edu](mailto:scholarworks@uno.edu).

# Implementation of Separable & Steerable Gaussian Smoothers on an FPGA

A Thesis

Submitted to the Graduate Faculty of the  
University of New Orleans  
in partial fulfillment of the  
requirements for the degree of

Master of Science  
in  
Engineering  
Electrical

by

Arjun Kumar Joginipelly  
B.S, JNTU University, 2007

December 2010

## **Dedication**

I dedicate this work to the people who have had a profound effect on my life. To my parents, Joginipelly Raj Gopal Rao & J.Prabha Rani; to my uncles, P.Govind Rao and P.Mohan Rao; to my sister and brother, M.Sri Laxmi & J.Anil Kumar and last, but certainly not least my grandparents Potlapally Bapu Rao and P.Bharatamma. The blessings and love you all gave were always with me and encouraged me in stepping forward in life.

## **Acknowledgements**

I would like to thank Dr. Dimitrios Charalampidis, my advisor, for his help, suggestions and guidance throughout the course of my thesis research. I appreciate his direction, supervision of my work and his patience especially for reading, rereading and editing my thesis which helped me in progressing in the right path.

I acknowledge my friend Mr. Rajesh Chary for his support throughout my studies without which it would have been impossible for me to get through my Master's degree. His patience, assistance and insight served as invaluable assets in both my personal and academic life.

I would also like to thank to Dr. Vesselin Jilkov and Dr. George Ioup for their willingness to serve as members in my thesis committee.

Most importantly, I would like to thank my parents for teaching me study habits and the value of education. I am indebted to them for their encouragement in my childhood years to constantly strive for a successful career.

And last, but certainly not least, I would like to express my warmest regards and gratitude to my grandfather for inspiring me from an early age to commit myself to helping others.

Finally I would like thank my friends and colleagues for being eager and prompt enough to help me when I needed them.

## **Glossary of Abbreviations**

FPGA – Field Programmable Gate Arrays

PLD – Programmable Logic Device

ASIC – Application Specific Integrated Circuit

FSM – Finite State Machine

DSP – Digital Signal Processor

VHDL – Very High speed integrated Description Language

ISE – Integrated Software Environment

DSF – Directional Smoothing Filter

LB – Logic Block

LUT – Look up Table

BRAM – Block RAM

IP – Intellectual Property

RAM – Random Access Memory

# Table of Contents

<b>List of Tables .....</b>	<b>vii</b>
<b>List of Figures.....</b>	<b>viii</b>
<b>Abstract.....</b>	<b>ix</b>
<b>Chapter 1 .....</b>	<b>1</b>
1.1 Introduction.....	1
1.2 Research Objectives.....	2
1.3 Scope of Thesis .....	3
1.4 Organization of Thesis .....	3
<b>Chapter 2 .....</b>	<b>4</b>
2.1 Field Programmable Gate Array (FPGA) .....	4
2.2 Xilinx VirtexII Pro FPGA Platform.....	6
<b>Chapter 3 .....</b>	<b>9</b>
3.1 Design Language .....	9
3.1.1 Verilog Hardware Design Language .....	11
3.1.2 VHSIC Hardware Design Language (VHDL) .....	12
3.2 Software Tools .....	12
3.3 Other languages and Tools .....	13
<b>Chapter 4 .....</b>	<b>14</b>
4.1 Convolution.....	14
4.2 Gaussian Mask.....	16
4.3 Steerable & Separable Gaussian Smoothing Filters .....	18

<b>Chapter 5 .....</b>	<b>21</b>
5.1 Hardware Implementation .....	21
5.2 Proposed Design Methodology.....	22
5.2.1 General 2D Convolution Method.....	24
5.2.2 Separable Convolution Method 1 using multiple BRAMs .....	28
5.2.3 Separable Convolution Method 2 using FIFO .....	33
5.2.4 Comparisons of Convolution Methods .....	38
5.2.5 Extension of Separable Convolution Method 2 using FIFO.....	39
5.3 Proposed Steerable Concept Implementation.....	43
<b>Chapter 6 .....</b>	<b>47</b>
6.1 Summary and Conclusions .....	47
6.2 Limitations .....	48
6.3 Future Work .....	48
<b>Bibliography .....</b>	<b>49</b>
<b>Appendix .....</b>	<b>53</b>
<b>Vita .....</b>	<b>80</b>

## List of Tables

Table 5.1: A 7×7 Test Image

Table 5.1: A 3×3 Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

Table 5.3: Device Utilization Summary of Two Dimensional Convolution Method

Table 5.4: Horizontal Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

Table 5.5: Vertical Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

Table 5.6: Device Utilization Summary of Separable Convolution Method 1

Table 5.7: Horizontal Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

Table 5.8: Vertical Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

Table 5.9: Device Utilization Summary of Separable Convolution Method 2

Table 5.10: Comparison of Convolution Methods

Table 5.11: Horizontal Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

Table 5.12: Vertical Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

Table 5.13: Device Utilization Summary of Separable Convolution Method 2 extended to 48×48  
image size and Gaussian mask of 9×1 and 1×9

Table 5.14: Horizontal Gaussian Mask with Mean = 0,  $\sigma_x = 3$ ,  $\sigma_y = 5$ ,  $N = 0.001$

Table 5.15: Vertical Gaussian Mask with Mean = 0,  $\sigma_x = 3$ ,  $\sigma_y = 5$ ,  $N = 0.001$

Table 5.16: Device Utilization summary of steerable implementation on a virtex4 board



## List of Figures

Figure 2.1: Architecture of a generic FPGA .....	4
Figure 2.2: Architecture of Logic Block with one 4-input LUT .....	5
Figure 2.3: Block Diagram of XUP VirtexII Pro FPGA Board .....	7
Figure 2.4: Picture of XUP VirtexII Pro FPGA Board .....	8
Figure 3.1: Hardware Design Flow .....	10
Figure 3.2: Software Design Flow .....	11
Figure 4.1: 2 D Convolution Operation .....	15
Figure 4.1: 2 D Gaussian Mask .....	17
Figure 5.1: Block Diagram of Two Dimensional Convolution Method .....	24
Figure 5.2: Schematic Diagram of Two Dimensional Convolution Method .....	25
Figure 5.3: Simulation Results of Two Dimensional Convolution Method .....	26
Figure 5.4: Block Diagram of Separable Convolution Method 1 .....	29
Figure 5.5: Schematic Diagram of Separable Convolution Method 1 .....	30
Figure 5.6: Simulation Results of Separable Convolution Method 1 .....	31
Figure 5.7: Block Diagram of Separable Convolution Method 2 .....	34
Figure 5.8: Schematic Diagram of Separable Convolution Method 2 .....	35
Figure 5.9: Simulation Results of Separable Convolution Method 2 .....	36
Figure 5.10: Schematic Diagram of Separable Convolution Method 2 extended to a 48×48 image and a Gaussian masks of 9×1 and 1×9 .....	40
Figure 5.11: Simulation Results of Separable Convolution Method 2 extended to a 48×48 image and a Gaussian masks of 9×1 and 1×9 .....	41
Figure 5.12: Block Diagram of Steerable Implementation .....	44
Figure 5.13: Schematic Diagram of Steerable Implementation .....	45

## **Abstract**

Smoothing filters have been extensively used for noise removal and image restoration. Directional filters are widely used in computer vision and image processing tasks such as motion analysis, edge detection, line parameter estimation and texture analysis. It is practically impossible to tune the filters to all possible positions and orientations in real time due to huge computation requirement. The efficient way is to design a few basis filters, and express the output of a directional filter as a weighted sum of the basis filter outputs. Directional filters having these properties are called “Steerable Filters”. This thesis work emphasis is on the implementation of proposed computationally efficient separable and steerable Gaussian smoothers on a Xilinx VirtexII Pro FPGA platform. FPGAs are Field Programmable Gate Arrays which consist of a collection of logic blocks including lookup tables, flip flops and some amount of Random Access Memory. All blocks are wired together using an array of interconnects. The proposed technique [2] is implemented on a FPGA hardware taking the advantage of parallelism and pipelining.

### **Keywords**

Field Programmable Gate Arrays (FPGAs), Parallel Image Processing, Directional Smoothing Filters, Steerable Filters, Gaussian Mask, Separable Convolution.

# CHAPTER 1

## Introduction

### 1.1 Introduction

Current developments of computer systems tend to reduce the size of the hardware. This is a conclusion drawn from Moore's law [1]. The hardware specifications and capabilities of a small laptop ten years ago are comparable to today's mobile devices, such as the iPhone 3GS. As a result, embedded computer systems are also becoming increasingly pervasive. For instance, today's cars include embedded systems to monitor a wide range of multi-media features such as audio, video, voice control, and navigation [22]. Another area where embedded systems play an important role is digital image processing with applications such as automated surveillance systems [23], traffic light controller systems [24]. In earlier times, those systems were mostly built with Application Specific Integrated Circuits (ASICs) which are not reprogrammable (or reconfigurable). A malfunction in one ASIC often results in a complete replacement of the faulty component. The ASICs lack of flexibility to be reprogrammed is promoting their counterpart, namely the FPGA (Field Programmable Gate Array) chips.

Recently, FPGA technology has become a viable target for the implementation of algorithms in image processing applications [18], [19]. FPGAs generally consist of a logic block based system, which usually includes lookup tables, flip-flops and some amount of Random Access Memory (RAM), all wired together using an array of interconnects. All of the logic in an FPGA can be reconfigured with a different design as often as the designer likes. This type of architecture allows a large variety of logic designs dependent on the processor's resources.

Today, FPGAs can be developed to implement parallel design methodologies, which is not possible in dedicated DSP designs. ASICs were traditionally preferred over FPGAs because of their speed, lower power consumption, and higher functionality. However, the improvements on FPGA technology in recent years have almost closed this gap. ASIC design methods can also be used for FPGA design, facilitating gate level implementations, thereby decreasing development time and time-to-market. However, engineers usually use a hardware language, which allows for a design methodology similar to software design. Maintenance can be performed when an error is found in the implemented design, since the FPGA fabric can always be reconfigured. This software view of hardware design allows for a lower overall support requirements, lower cost, and design abstraction.

The key advantages of FPGAs when compared to DSP implementations include performance, integration and customization using parallel and pipeline design techniques. Due to the support of parallelism, FPGAs may be able to achieve huge gains in performance compared to DSP implementations.

## **1.2 Research Objectives**

The main objective of this thesis is to develop an efficient architecture for directional Gaussian smoothers simulated in VHDL and prototyped on device technology of XILINX VirtexII-Pro FPGA platform. Implementation on the target device takes the advantage of parallelism of FPGA and ensures high throughput.

### **1.3 Scope of Thesis**

The main contribution in this thesis is the design and implementation of directional Gaussian smoothers [2] on FPGA. Firstly, derivations are presented to show that Gaussian filters are separable. Secondly, in [13], it was shown that these filters can also be made approximately steerable. The inferred equations are also derived and presented here for completeness. The functionality of directional (or steerable) Gaussian smoothers is examined using Matlab simulations. Then, a VHDL model is developed for a test image of  $7 \times 7$  and a Gaussian mask of  $3 \times 3$ . Based on the simulation results and logic utilization, we implemented the convolution operation similar to the techniques presented in [15], [17]. Furthermore, additional techniques were implemented to improve logic utilization and processing speed for performing convolution. All the hardware architectural models are prototyped on XC2VP30FFG896, a device technology of Xilinx VirtexII-Pro FPGA platform. For all methods implemented on the target device, comparisons are made using logic utilization (in terms of number of flip-flops and slice count) and number of clock cycles per pixel.

### **1.4 Organization of Thesis**

Chapter 2 describes FPGAs in detail and an overview of XILINX VirtexII Pro Development Board. Chapter 3 describes the language used and the software tool used for programming FPGAs. Chapter 4 describes the concepts of convolution, Gaussian filters, and the steerability concept for Gaussian smoothers. Chapter 5 describes the hardware implementation and design methodology. Chapter 6 includes conclusions, limitations, and future work.

## CHAPTER 2

### FPGA and Xilinx VirtexII Pro Board

#### 2.1 Field Programmable Gate Array (FPGA)

An FPGA is a chip that allows the user to control and reprogram the functionality of its logic circuits. All FPGAs consist of three major components, namely Logic Blocks (LB), I/O Blocks, and Programmable Routing or Interconnect as shown in figure 2.1 [3].

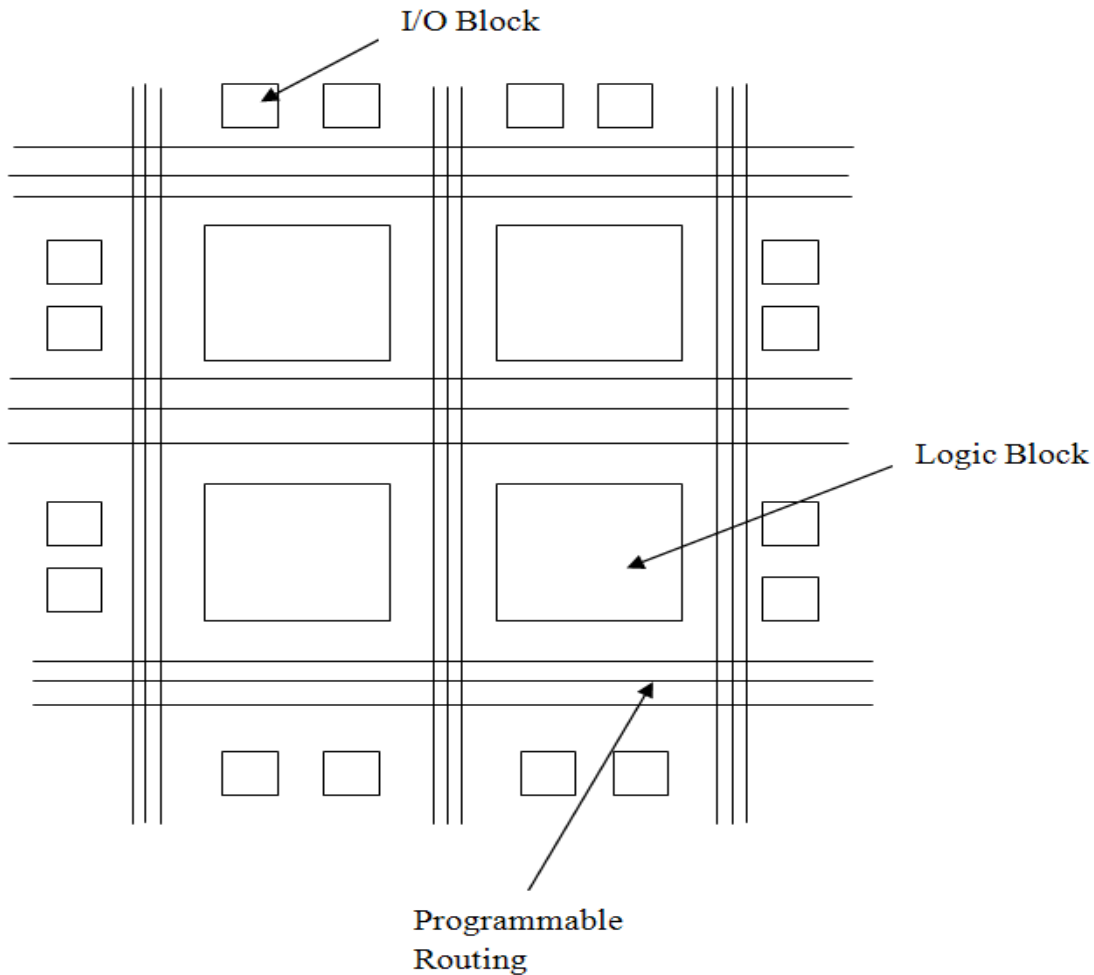


Figure 2.1: Architecture of a generic FPGA [3]

In order to implement a circuit on an FPGA, each LB is programmed to perform a small part of the logic and each I/O block is programmed to act as input or output, as required by the circuit. The programmable routing is also configured to make all necessary connections between LBs and from LBs to I/O blocks.

The processing power of an FPGA is directly proportional to the processing capabilities of its LBs and the total number of LBs available in the array. Currently, most of the commercial FPGAs use LBs that contain one or more Look-up Tables (LUTs), typically a 4-input LUT. A 4-input LUT can implement any binary function of 4 logic inputs. The architecture of a simple LB containing one 4-input LUT and one flip-flop for storage is shown in figure 2.2 [3].

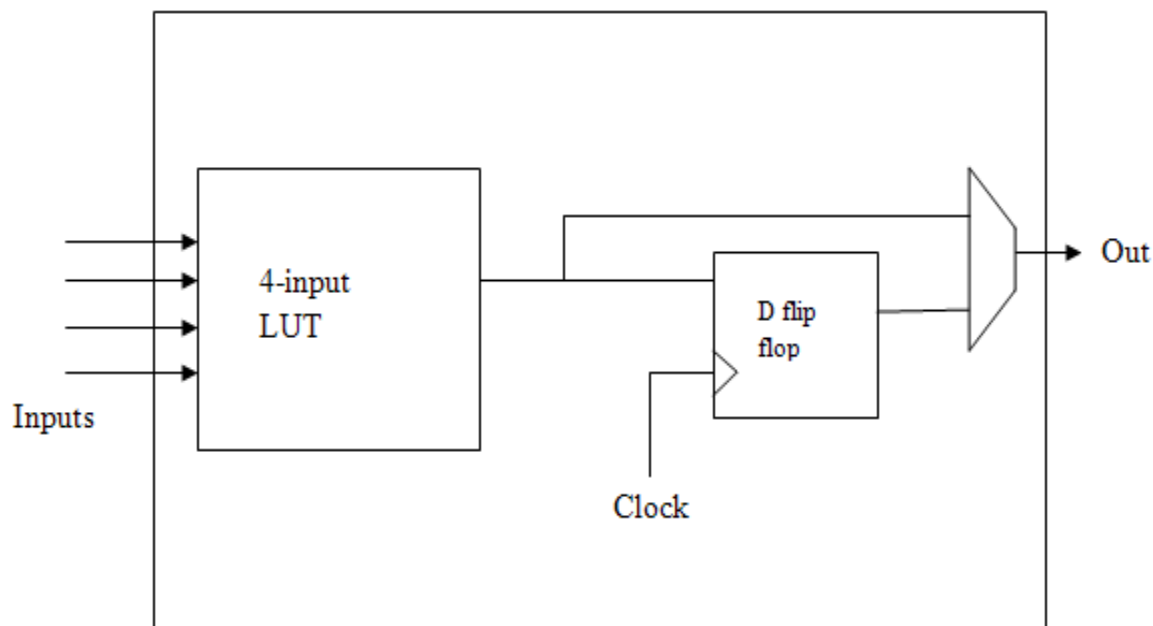


Figure 2.2: Architecture of Logic Block with one 4-input LUT [3]

Modern FPGAs also contain blocks of on-chip memory as well. For example, the target FPGA device XU2VP30 used in this thesis work contains 136 blocks of 4Kbits of RAM, 13696 slices, 27392 LUTs, 136 18×18 embedded multipliers and 556 bonded IOB's. An overview and detailed explanation of target device used is presented in next subheading.

## **2.2 Xilinx VirtexII Pro FPGA Platform**

The XU2VP30-FFG896 is a Xilinx manufactured Virtex-2 Evaluation Board with an advanced hardware platform that consists of high performance VirtexII Pro Platform FPGA [9], surrounded by peripheral components that can be used to create a complex system. Main features of the platform are the following:

- Virtex®-II Pro FPGA with PowerPC® 405 cores
- Maximum 2 GB of Double Data Rate (DDR) SDRAM
- Compact Flash connector
- Embedded Platform Cable USB configuration port
- Programmable Configuration PROM
- On-board 10/100 Ethernet PHY device
- RS-232 DB9 serial port
- Two PS-2 serial ports
- Four LEDs connected to Virtex-II Pro I/O pins
- Four switches connected to Virtex-II Pro I/O pins
- Five push buttons connected to Virtex-II Pro I/O pins
- Six expansion connectors joined to 80 Virtex-II Pro I/O pins
- High-speed expansion connector joined to 40 Virtex-II Pro I/O pins



- AC-97 audio CODEC with audio amplifier and speaker/headphone output
- Microphone and line level audio input
- On-board XSGA output, up to 1200 x 1600 at 70 Hz refresh
- Three Serial ATA ports, two Host ports and one Target port
- Off-board expansion MGT link, with user-supplied clock
- 100 MHz system clock, 75 MHz SATA clock
- Provision for user-supplied clock
- On-board power supplies
- Power-on reset circuitry
- PowerPC 405 reset circuitry

The block diagram of the board is shown in figure 2.3.

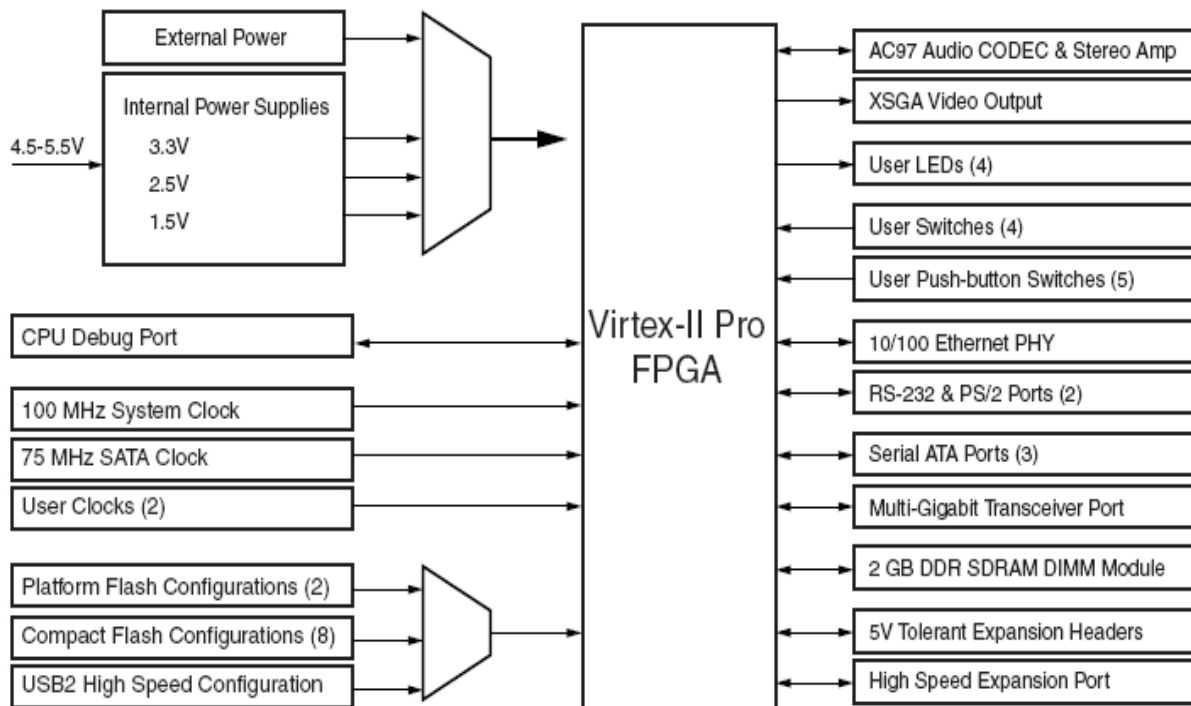


Figure 2.3: Block Diagram of XUP VirtexII Pro FPGA Board [8]

The picture of the board can be seen in figure 2.4 below.

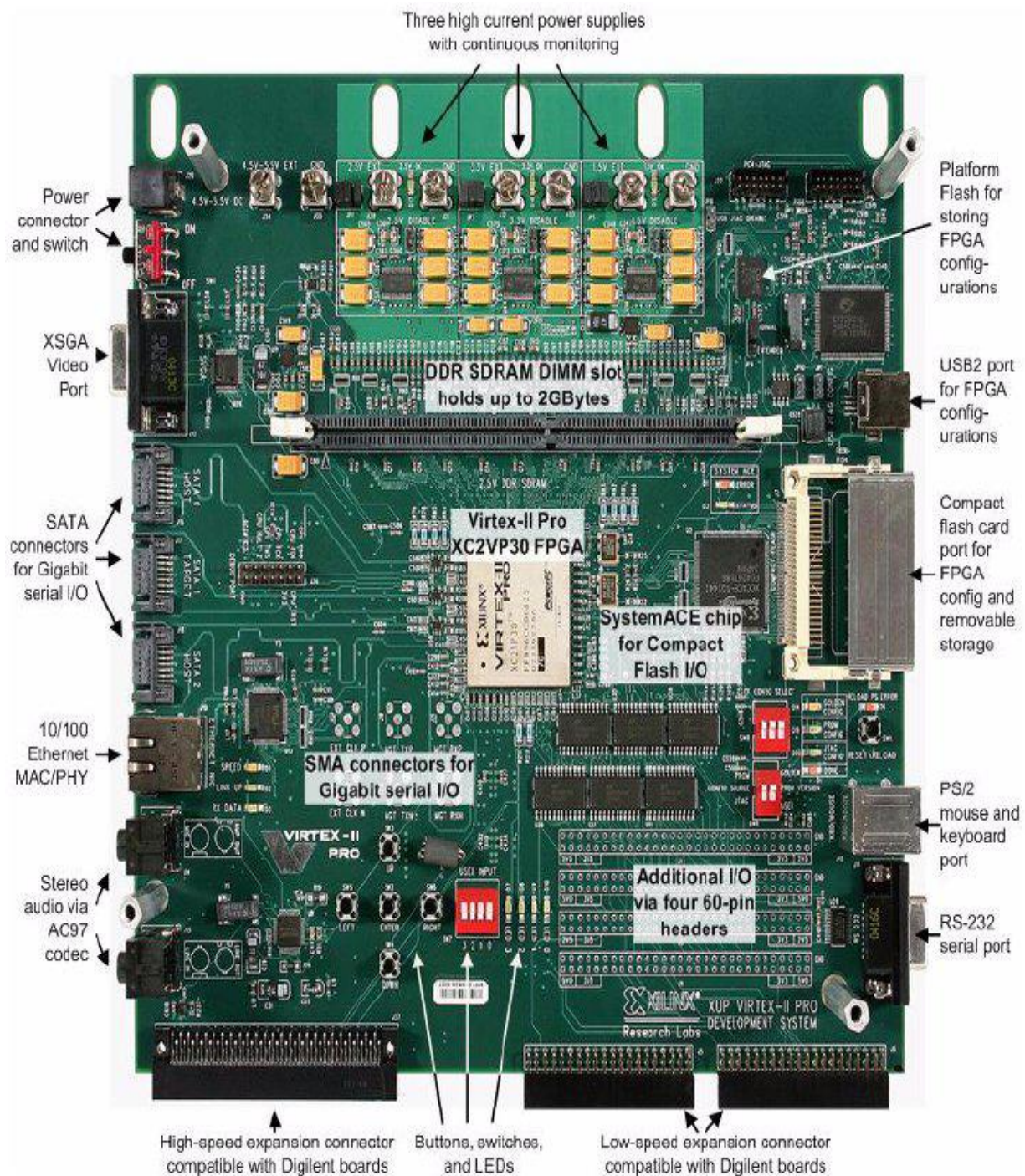


Figure 2.4: Picture of XUP VirtexII Pro Board [8]

## CHAPTER 3

### Design Language & Software Tools

#### 3.1 Design Language

There are several differences between the traditional software design flow and the established Verilog/VHDL design flow for FPGAs. After designing the circuit, there is a multistage process to go through before the design can be used in an FPGA. The first stage is synthesis, which takes HDL code and translates it into a netlist. A netlist is a textual description of a circuit diagram or schematic. Next, simulation may be used to verify that the design specified in the netlist functions correctly. Once verified, the netlist is converted into binary format. More specifically, the components and connections that the netlist defines are mapped to CLBs (map), and the design is placed and routed to fit onto the target FPGA (place and route). A second simulation (post, place and route simulation) is performed to help establish how well the design has been placed and routed. Finally, a “\*.bit” file is generated to load the design onto the FPGA. A “\*.bit” file is a configuration file that is used to program all of the resources within the FPGA. Using tools such as Xilinx Chipscope is then possible to verify and debug the design while it is running on the FPGA. In hardware, it is very important to establish that a design is functionality correct prior to implementation as a broken design could take a day or more to place and route and could potentially cause damage to system components. Figures 3.1 and 3.2 [6] illustrate the differences between software and hardware design flows.

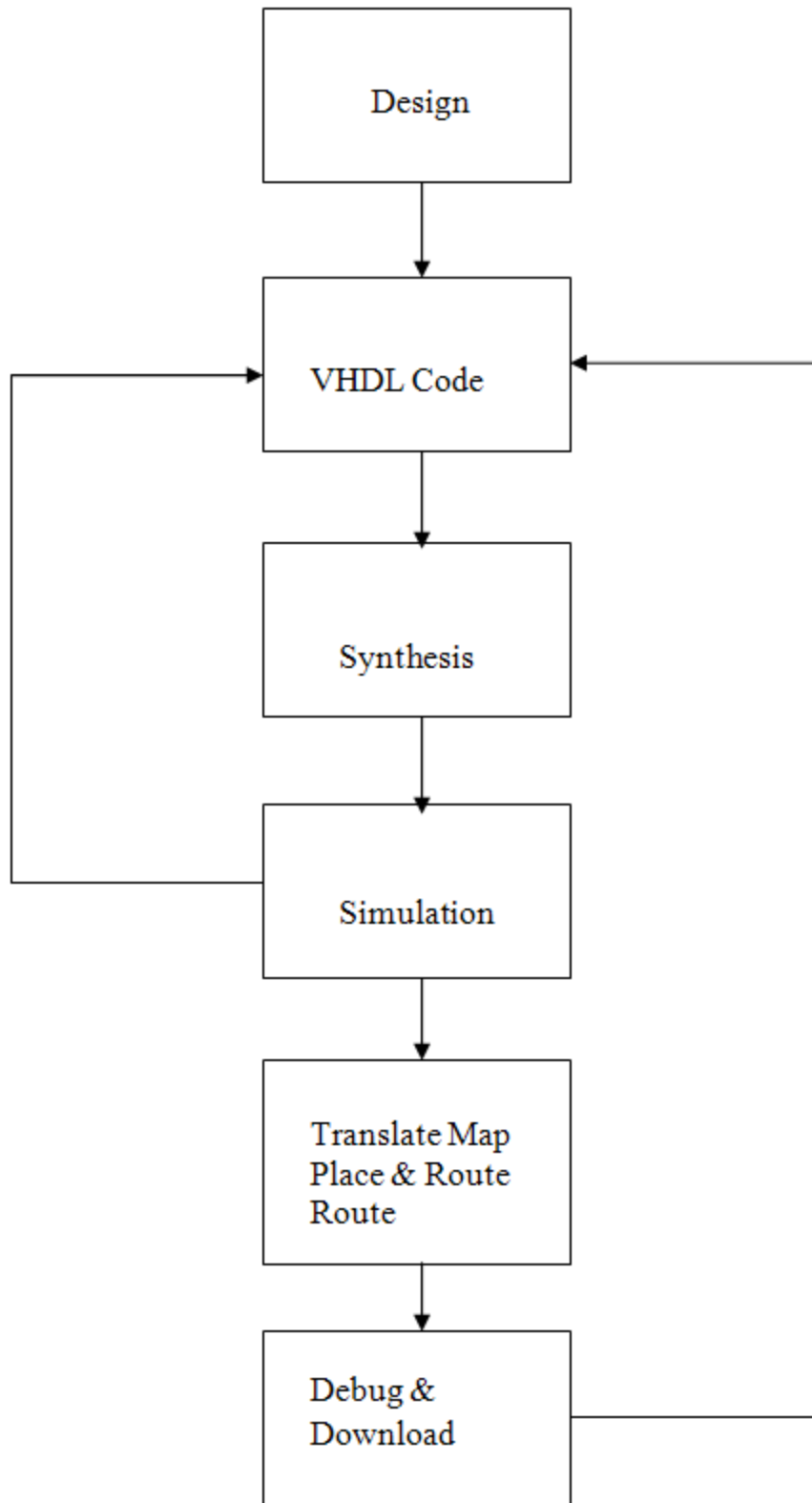


Figure 3.1: Hardware Design Flow [6]

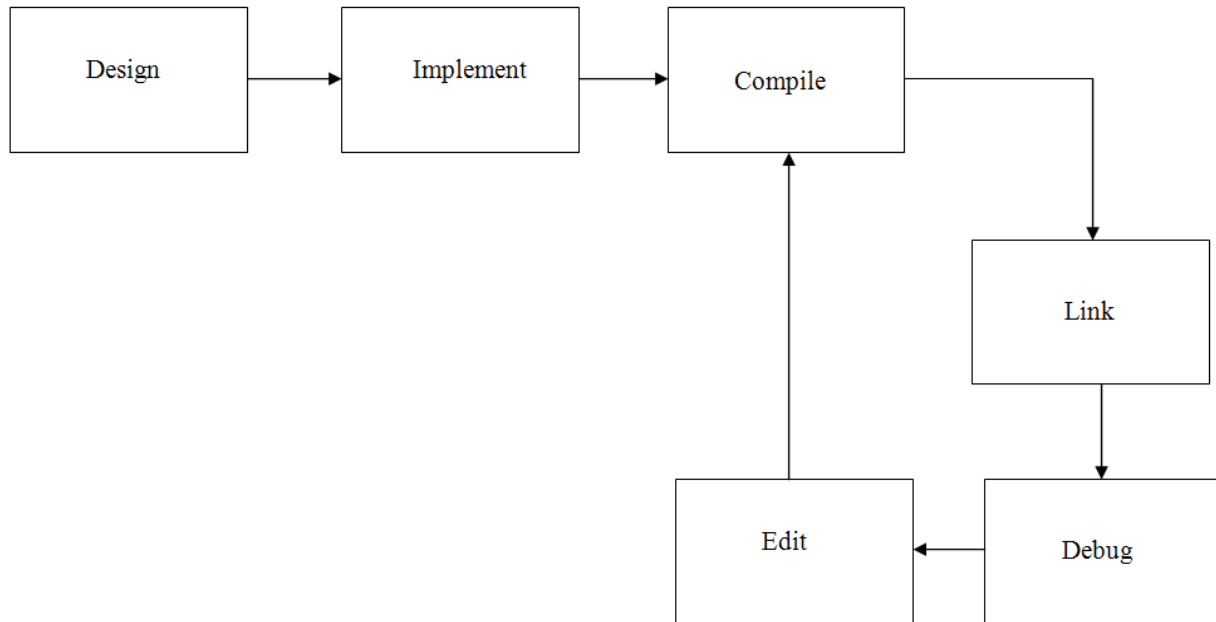


Figure 3.2: Software Design Flow

The following subsections discuss the two common high level hardware design languages (HDLs) in which FPGA algorithms are designed.

### 3.1.1 Verilog Hardware Design Language

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

### **3.1.2 VHSIC Hardware Design Language (VHDL)**

In Recent years, VHSIC (Very High Speed Integrated Circuit) Hardware Design Language (VHDL) has become an open IEEE standard [11]; it is supported by a large variety of design tools and is quite interchangeable between different vendors' tools. The first version of VHDL, IEEE 1076-87, appeared in 1987 and has since undergone an update in 1993, appropriately titled IEEE 1076-93. It is high level language similar to the computer programming language Ada, which is intended to support the design, verification, synthesis and testing of hardware designs.

It is very straightward to simulate simple logic designs such as D flip-flop. However it is surprisingly difficult to implement it in hardware as we have to take into account of I/O issues, access to resources external to FPGA such as memory, push-buttons, DIP switches and etc. If you want to retrieve a value from main memory and use it on FPGA then you need to instantiate a memory controller [31].

## **3.2 Software Tools**

Xilinx is one of the leading largest producers of Xilinx boards and tools which provide fully functional VHDL and Verilog development environment with full range of editing, synthesis, simulation and implementation tools. The Xilinx tools are relatively user friendly and tools required for our basic design are free to download. In my thesis I have used ISE 10.1.03 [7] and ISIM is used as simulation tool. Matlab 9.1 version is used for verification of obtained results, to check the functionality of concepts such as convolution, separability, steerability and to create “.coe” file [31] which is used to load any data into BRAM of FPGA board. Details about “.coe”file are explained in chapter 5.

### 3.3 Other languages and tools

A list of other available languages and tools are given below:-

- SystemC - Open SystemC Initiative (OSCI) - <http://www.systemc.org/>
- Catapult C - Mentor Graphics - [http://www.mentor.com/products/c-based\\_design/](http://www.mentor.com/products/c-based_design/)
- Impulse C - Impulse Accelerated Technologies - <http://www.impulsec.com/>
- Carte - SRC Computers - <http://www.srccomp.com/CarteProgEnv.htm>
- Streams C - Los Alamos National Laboratory - <http://www.streams-c.lanl.gov/>
- AccelChip - MATLAB DSP Synthesis - <http://www.accelchip.com/>
- Starbridge - VIVA - <http://www.starbridgesystems.com/>
- NAPA-C - National Semiconductor - <http://portal.acm.org/citation.cfm?id=795813>
- SA-C - Colorado State University - <http://www.cs.colostate.edu/cameron/compiler.html>
- CoreFire - Annapolis Micro Systems - <http://www.annapmicro.com/>
- Trident compiler - Los Alamos National Laboratory - <http://trident.sourceforge.net/>
- Reconfigurable Computing Toolbox - DSPlogic - [www.dsplogic.com/home/products](http://www.dsplogic.com/home/products)

Details of a number of these FPGA programming tools can be found on the University of Florida's High Performance Computing and simulation Research Centre web pages [http://docs.hcs.ufl.edu/xd1/app\\_mappers](http://docs.hcs.ufl.edu/xd1/app_mappers).

## CHAPTER 4

### Convolution & Steerable Gaussian Smoothing Filters

#### 4.1 Convolution

Convolution is a common image processing operation that filters an image by calculating the sum of products between the input image and a smaller image like array called the “convolution kernel or convolution filter”. A convolution operation can achieve blurring, sharpening, noise reduction, edge detection and other useful imaging operations depending on the selection of values in the convolution kernel.

Mathematically, a two dimensional convolution on image can be represented by the following equation.

$$h(m,n) = \sum_{i=0}^{height-1} \sum_{j=0}^{width-1} g(i,j) f(m-i,n-j) \dots\dots\dots(1)$$

where  $f$  is the input image,  $g$  is the filter and  $h$  is the output image

In the above equation, the function  $f$  represents the input image and  $g$  represents the convolution kernel. The double summation is based on the width and height of the convolution kernel. A convolution operation is computed by aligning the center of the convolution kernel with the pixel at the same position in the input image. Multiplying the values of input image pixels with the pixels covered by the convolution kernel and then summing the results provide the value of the particular pixel in the output image.



For instance, a two dimensional convolution using a  $3 \times 3$  input image and  $3 \times 3$  kernel would look like as follows:

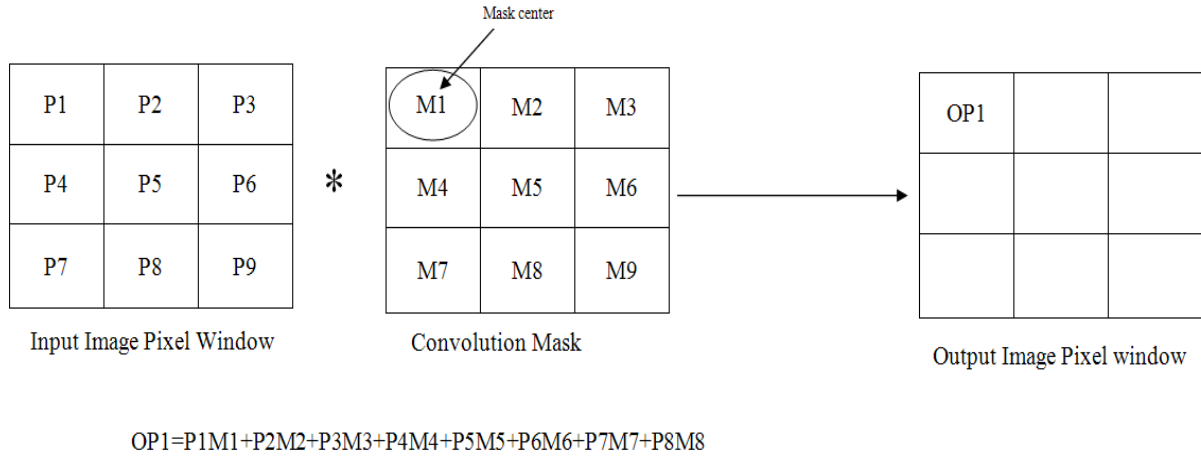


Figure 4.1: 2D Convolution Operation

In order to calculate an output pixel for a given mask of size  $m \times n$ ,  $mn$  multiplications and  $mn-1$  additions are required. The Gaussian mask and one of its important properties, namely separability, are presented with more details in the following section.

## 4.2 Gaussian Mask

The Gaussian distribution in 1D has the following form:

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

In 2D, a circularly symmetric Gaussian has the form

$$g(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \dots\dots\dots (3)$$

where  $g$  is the gaussian kernel weight at the location with coordinates  $x$  and  $y$ . The  $\sigma$  parameter is the standard deviation of the Gaussian distribution which determines the sharpness or smoothness of the Gaussian function. The term  $\frac{1}{2\pi\sigma^2}$  is normalization constant.

The idea of Gaussian convolution is to use this 2D envelope as a point spread function. The degree of smoothing is determined by the standard deviation  $\sigma$  of the Gaussian. Since the image is stored as a collection of discrete pixels, a discrete approximation to the Gaussian function is required to perform the convolution. The Gaussian mask weights fall off to almost zeros at the mask edges. A general 2D Gaussian is shown below:-

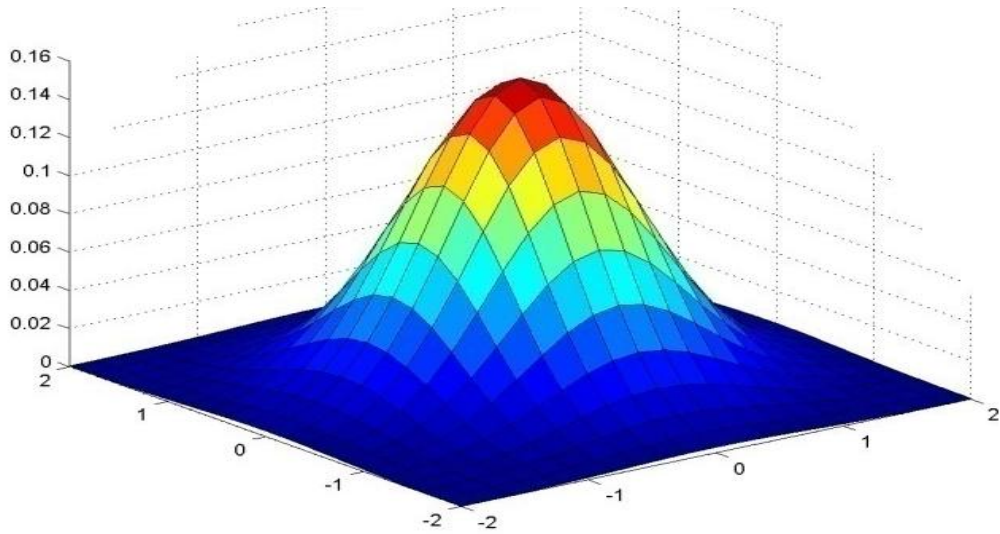


Figure 4.2: A 2D Gaussian Mask

The greatest advantage of the Gaussian filters of equation (3) is that they are separable. In particular, the product of two 1D Gaussian functions gives a higher dimensional Gaussian function and this can be represented mathematically as follows:-

$$g(x, y) = g(x)g(y) \dots\dots\dots(4)$$

An important application of separability is that convolution with a 2D Gaussian kernel can be replaced by a cascade of 1D Gaussian kernels, making the whole convolution process much more efficient with fewer number of multiplications. Therefore convolution using separable filter is performed in two steps. The input or original image is convolved with a filter of size  $N \times 1$ , while the result is convolved with a filter of size  $1 \times N$ . Hence in this case of separable convolution, a total of  $2N$  multiplications and  $2N-2$  additions are required which is significantly less compared to the non-separable case, particularly for large-scale filters.

### 4.3 Steerable & Separable Gaussian Smoothing Filters

Directional or orientation filters are widely used in computer vision and image processing, such as motion analysis, edge detection and texture analysis. In general, the shifts, edges and lines can be characterized by a set of parameters including position, orientation, width or size. In order to obtain the response of a filter at any arbitrary position and orientation it is very important to tune the filters to all possible positions and orientations in real time. However, huge computations are required in this way. The efficient way is to design a family of filters so that any filter in this family can be represented by few basis filters. Therefore, the output of a filter can be expressed as a weighted sum of basis filter outputs. Such filters are called “steerable filters”.

Steerability implies that the output  $O_\theta(x, y)$  of a filtering operation using a filter oriented at an angle  $\theta$  can be computed as the linear combination of a finite set of  $M$  outputs  $\{ O_{\theta_0}(x, y), O_{\theta_1}(x, y), \dots, O_{\theta_{M-1}}(x, y) \}$  obtained by applying the same filter oriented at directions  $\theta_0, \theta_1, \dots, \theta_{M-1}$ , respectively. A 2D separable and steerable filter can be written as:

$$g_\theta(x, y) = \sum_{r=-R}^R g_{iso}(x - r\cos(\theta), y - r\sin(\theta))g^{1D}(r) \dots\dots\dots(5)$$

where it was assumed that the size of  $g^{1D}(r)$  is equal to  $2R+1$ .

The filter described in (5) can be applied to an image  $I(x, y)$  in two steps. In the first step, the filter  $g_{iso}(x, y)$  is applied to the image.

$$I_{iso}(x, y) = I(x, y) * g_{iso}(x, y) \dots\dots\dots(6)$$

In the second step, the following operation is applied to the image  $I_{\text{iso}}(x, y)$ .

$$I_{\theta}(x, y) = \sum_{r=-R}^R I_{\text{iso}}(x - r\cos(\theta), y - r\sin(\theta)) g^{1D}(r) \dots\dots\dots (7)$$

The operation described in (6) and (7) is equivalent to the operation where the input image  $I(x, y)$  is filtered by a Gaussian directional smoothing filter (DSF) oriented at direction  $\theta$ . The function  $g_{\text{iso}}(x, y)$  describes a separable filter and can thus be implemented in an efficient manner. More specifically,  $g_{\text{iso}}(x, y)$  can be expressed as  $g_{\text{iso}}(x, y) = g_x(x)g_y(y)$  where

$$g_x(x) = \frac{1}{2\pi\sigma_x^2} e^{-x^2/(2\sigma_x^2)} \quad \text{and,} \quad g_y(y) = e^{-y^2/(2\sigma_y^2)}.$$

Hence,  $g_{\text{iso}}(x, y)$  can be applied to  $I(x, y)$  by first filtering  $I(x, y)$  in a horizontal manner using  $g_x(x)$  and then by filtering the result in vertical manner using  $g_y(y)$ . Equation (7) describes a linear combination of shifted versions of the image  $I_{\text{iso}}(x - r\cos(\theta), y - r\sin(\theta))$ , which depend on the filtering direction  $\theta$ . The coefficients of the linear combination are equal to the values of  $g^{1D}(r)$ . Image  $I_{\text{iso}}(x - r\cos(\theta), y - r\sin(\theta))$  can be represented as the convolution between the input image  $I(x, y)$  and the filter  $g_{\text{iso}}(x - r\cos(\theta), y - r\sin(\theta))$ .

Thus, the proposed implementation is steerable in the sense that the final output  $I_{\theta}(x, y)$  can be expressed as a linear combination of the filtering operation outputs  $I_{\text{iso}}(x - r\cos(\theta), y - r\sin(\theta))$  of a set of  $2R+1$  fundamental filters  $g_{\text{iso}}(x - r\cos(\theta), y - r\sin(\theta))$ , parameterized by  $r$ , applied on the input image  $I(x, y)$ .

The isotropic filter  $g_{\text{iso}}(x, y)$  is low pass and almost 100% of the energy of the filter is included within the frequency band  $[-3/\sigma_x, 3/\sigma_x]$ . Therefore, the output  $g_{\text{iso}}(x, y)$  obtained by the filtering the input image  $I(x, y)$  with  $g_{\text{iso}}(x, y)$  is band limited within the frequency range  $(-\pi, \pi]$  in any direction  $\theta$ . Thus, equation (7) can be modified without introducing significant aliasing.

$$I_{\theta}(x, y) = \sum_{k=-[R/D]}^{[R/D]} I_{\text{iso}}(x - kD\cos(\theta), y - kD\sin(\theta)) g^{1D}(kD) \dots\dots\dots(8)$$

$$\text{where } g^{1D}(r) = g(kD) = \frac{1}{\sqrt{2\pi(\sigma_y^2 - \sigma_x^2)}} e^{\frac{-(kD)^2}{2(\sigma_y^2 - \sigma_x^2)}} \dots\dots\dots(9)$$

$$D = \frac{\pi\sigma_x}{3}, \text{ is a down sampling factor } \dots\dots\dots(10)$$

$[R/D]$  equals to the integer part of  $[R/D]$ . Since the range of unique frequencies in discrete signals is  $(-\pi, \pi]$ ,  $D$  can be as large as the largest integer not greater than  $\pi\sigma_x/3$ , so that aliasing does not occur. The goal of introducing a down sampling factor is to further reduce the computational complexity of the filtering operation.

## CHAPTER 5

### Hardware Implementation & Design Methodology

#### 5.1 Hardware Implementation

This chapter explains in detail the reconfigurable hardware implementations of image processing algorithms discussed in chapter 4, on a Xilinx VirtexII-Pro FPGA platform. The algorithms implemented are:

- General two dimensional convolution method
- Separable convolution method 1 (using multiple BRAMs)
- Separable convolution method 2 (using FIFO)
- Steerable method

Convolution is one of the basic and common operations on images. It uses a sliding window operator as discussed in section 4.2 of chapter 4. Based on the convolution operation, the weighted sum of the input pixels within the window, considering that the window is centered at pixel  $(x, y)$  is equal to the output at location  $(x, y)$ . The weights are the values of the filter assigned to every pixel of the window.

Convolution requires a significant amount of computational power. In order to calculate an output pixel for a given mask of size  $m \times n$ ,  $mn$  multiplications and  $mn-1$  additions are required. Therefore, in order to perform a two dimensional convolution on a  $256 \times 256$  gray scale image and  $3 \times 3$  mask a total of 589,824 multiplications and 65,535 additions are required.

A single multiplication requires significant hardware resources and produces long delays. In order to improve the performance of the convolution operation, it is necessary to reduce the number of multiplications. Different techniques of performing multiplication on hardware are explained in [20], [21]. Hence in the approach presented in this thesis, the algorithms are developed by paying special attention to reducing the number of multiplications, thereby decreasing the number of hardware resources while maintaining a satisfactory throughput in terms of clock cycles.

## **5.2 Proposed Design Methodology**

The main goal of this thesis is to implement steerable filtering techniques on FPGA efficiently. The task is divided into steps which facilitate the building of the basic blocks. As described in section 4.3 of chapter 4, the particular steerable filtering technique requires that the image is first smoothed. This is achieved by convolving the original image with a Gaussian mask. This convolution component is possibly the most important building block. Optimizing and pipelining at this stage improves the implementation efficiency.

First, a small test image of  $7 \times 7$  and a Gaussian mask of  $3 \times 3$  were chosen for performing the convolution operation. The two dimensional convolution operation was implemented using three different approaches which are listed below:-

- 1) General two dimensional convolution Method
- 2) Separable convolution method 1 (using multiple BRAMs)
- 3) Separable convolution method 2 (using FIFO)



A detailed explanation of each method, their performances and the associated logic utilization along with algorithmic state diagrams are presented in the following subsections. For all methods explained below, a test  $7 \times 7$  image and a  $3 \times 3$  Gaussian mask derived using equation (3) with mean = 0 and standard deviation = 1 and normalizing factor  $N = 0.0016$  are considered. Each test image pixel is represented using 16 bits and each mask value is also represented using 16 bits. A  $7 \times 7$  test image and a  $3 \times 3$  Gaussian mask are shown below:-

1	2	3	4	5	0	0
6	7	8	9	10	0	0
11	12	13	14	15	0	0
16	17	18	19	20	0	0
21	22	23	24	25	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Table 5.1: A  $7 \times 7$  Test Image

37	61	8
61	100	14
8	14	2

Table 5.2: A  $3 \times 3$  Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

### 5.2.1 General two dimensional convolution method

In this method, BRAM is used to store a  $7 \times 7$  test image using .coe file [31] which is generated with Matlab. The Matlab program used for generating .coe file is available in the appendix. An image controller is designed as a Finite State Machine (FSM) using VHDL to access the stored image in the BRAM. VHDL code for image read/write controller is available in the appendix. The obtained image pixels and mask pixels are controlled using pixel and mask controller blocks. A multiplier is designed using the Intellectual Property (IP) core [32]. The inputs to the multiplier are obtained from the pixel and mask controller blocks. The multiplier block generates an output which is represented using  $2n-1$  bits. The multiplier inputs are represented using  $n$  bits. In this thesis work  $n$  was set equal to 16. The multiplier outputs are then given to an adder which provides a 34 bit output. The adder output is the two dimensional convolution result between the  $7 \times 7$  test image and the  $3 \times 3$  Gaussian mask. The block diagram representation of two dimensional convolution is shown below:-

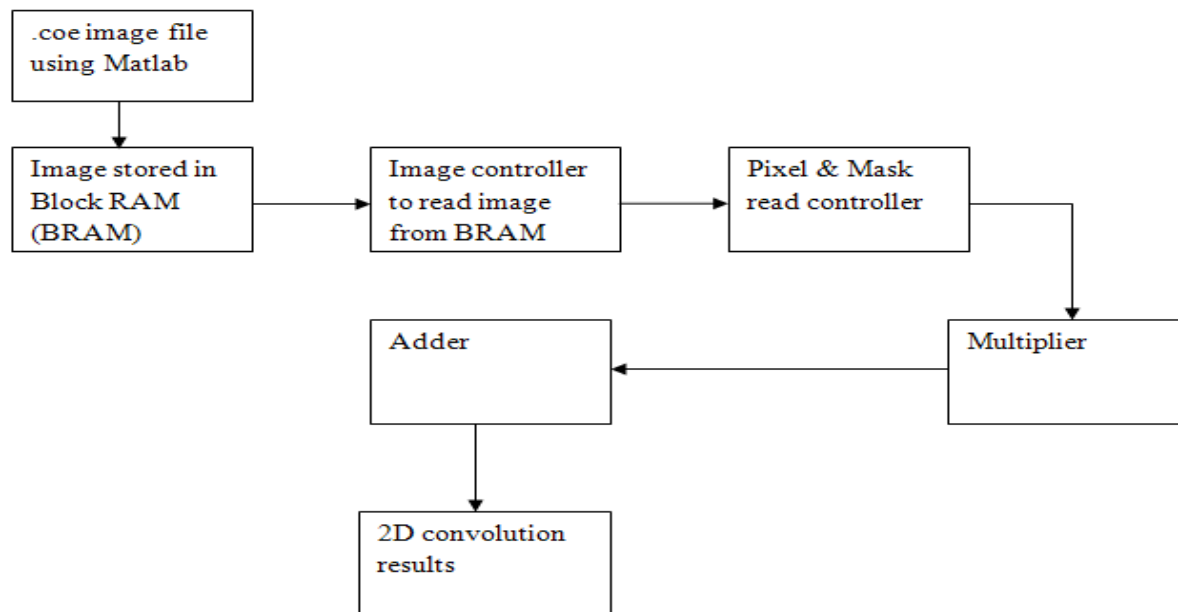


Figure 5.1: Block Diagram of Two Dimensional Convolution Method

In the schematic diagram below, the block named as “topmodule” is the image and mask controller. The module that stores the image in BRAM, and the image controller which reads the image from BRAM are embedded in the topmodule block. The outputs of topmodule are connected to 9 multipliers. The outputs of the 9 multipliers are finally connected to a 32 bit adder named as “adder\_32”. A 34 bit result obtained from adder\_32 is the two dimensional convolution between the  $7 \times 7$  test image and the  $3 \times 3$  Gaussian mask. A complete schematic diagram of general two dimensional convolution method is shown below:-

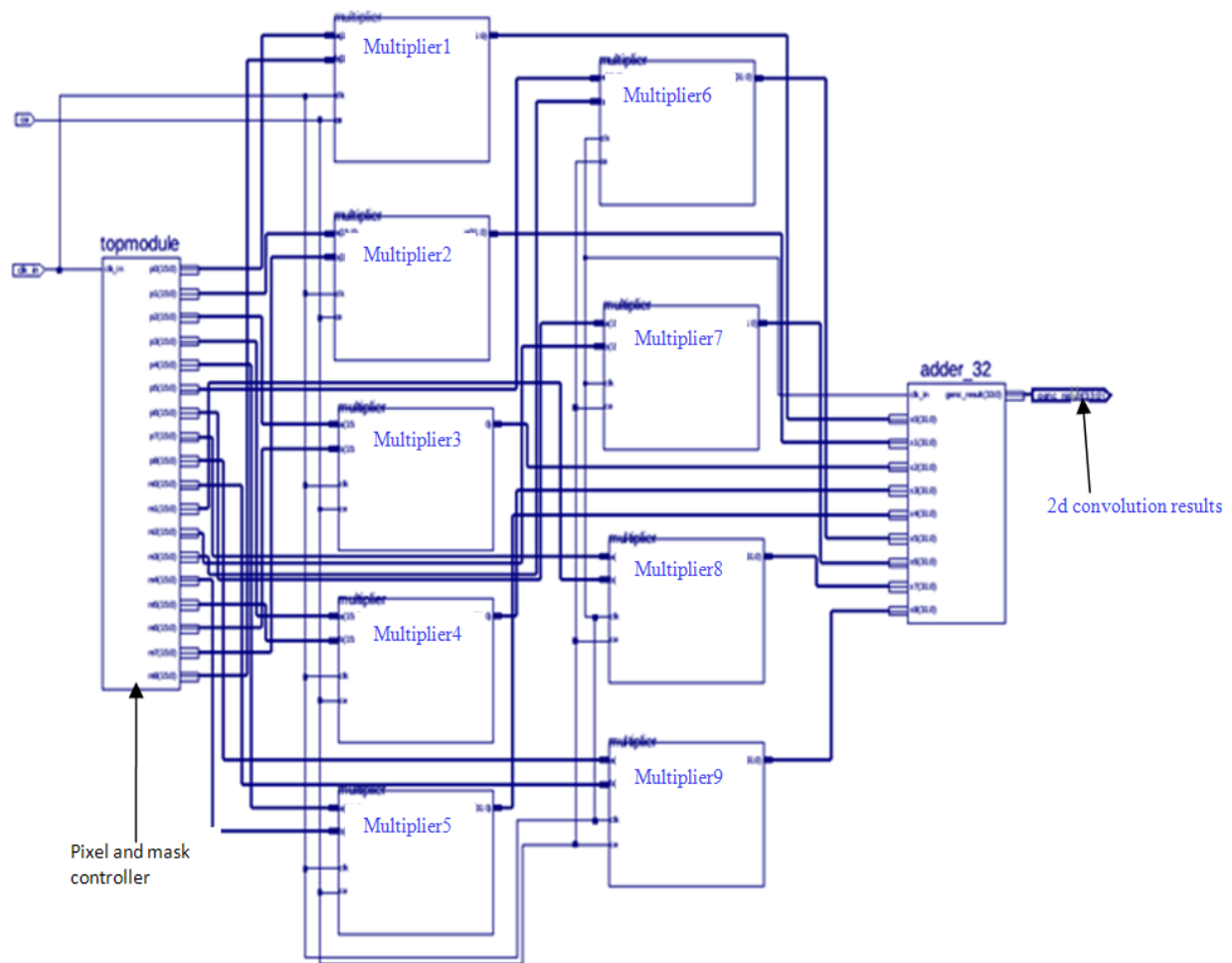


Figure 5.2: Schematic Diagram of Two Dimensional Convolution Method

The schematic design is simulated using Xilinx ISIM simulator for verification purpose. In the simulation diagram below, the reader may observe at the annotations, the image pixel controller outputs and the mask controller outputs, the multiplier outputs, and finally, the two dimensional convolution results. The simulation results are verified with Matlab and are provided in the appendix. The simulation results of two dimensional convolution are shown below:-



Figure 5.3: Simulation Results of Two Dimensional Convolution Method

Finally, the overall design is simulated using Xilinx XST Synthesizer to obtain the logic or hardware resource utilization on the target device. The design summary of the two dimensional convolution method is shown below:-


Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	414	13696	3%
Number of Slice Flip Flops	596	27392	2%
Number of 4 input LUTs	306	27392	1%
Number of bonded IOBs	36	556	6%
Number of BRAMs	1	136	0%
Number of MULT18X18s	9	136	6%
Number of GCLKs	2	16	12%

Table 5.3: Device Utilization Summary of Two Dimensional Convolution Method

In the simulation results, it can be observed that the total number of clock cycles required to complete a two dimensional convolution between a  $7 \times 7$  test image and a  $3 \times 3$  Gaussian mask is equal to 148. Hence the two dimensional convolution performance for the direct method is approximately 3 clocks per pixel.

### 5.2.2 Separable convolution method 1(using multiple BRAMs)

As discussed in section 4.2 of chapter 4, a Gaussian mask is separable. The separable Gaussian mask is derived using equations [3] and [4] with mean equal to zero,  $\sigma$  equal to zero and normalizing factor  $N = 0.0016$  are shown below:-

61	100	14
----	-----	----

Table 5.4: Horizontal Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

61
100
14

Table 5.5: Vertical Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

Similar to the regular convolution approach presented in section 5.2.1, BRAM is used to store a  $7 \times 7$  test image using .coe file [31] and an image controller is designed to access the stored image in the BRAM. The obtained image pixels and mask pixels are controlled using pixel and mask controller block. A multiplier is designed using the IP core [32]. The inputs to the multiplier are obtained from the pixel and mask controller blocks. The multiplier block generates an output which is represented using  $2n-1$ bits. The multiplier inputs are represented using  $n$  bits. In this thesis work  $n$  was set equal to 16. The multiplier outputs are then given to an adder which provides a 34 bit output. The adder output is the vertical (intermediate) convolution result between the  $7 \times 7$  test image and the  $3 \times 1$  vertical Gaussian mask.

A write and read controller is designed as a FSM using VHDL for writing the vertical (or intermediate) convolution result into the BRAM, and a read controller to read the intermediate convolution results. At pixel and mask controller block, the vertical Gaussian mask pixels (34 bits) and the vertical convolution result pixels (34 bits) are accessed and given to multiplier and adder block. The 70 bit output obtained is the final result of separable convolution using method 1 between the  $7 \times 7$  test image and the  $1 \times 3$  horizontal Gaussian mask. The block diagram representation of separable convolution method 1 (using multiple BRAMs) is shown below:-

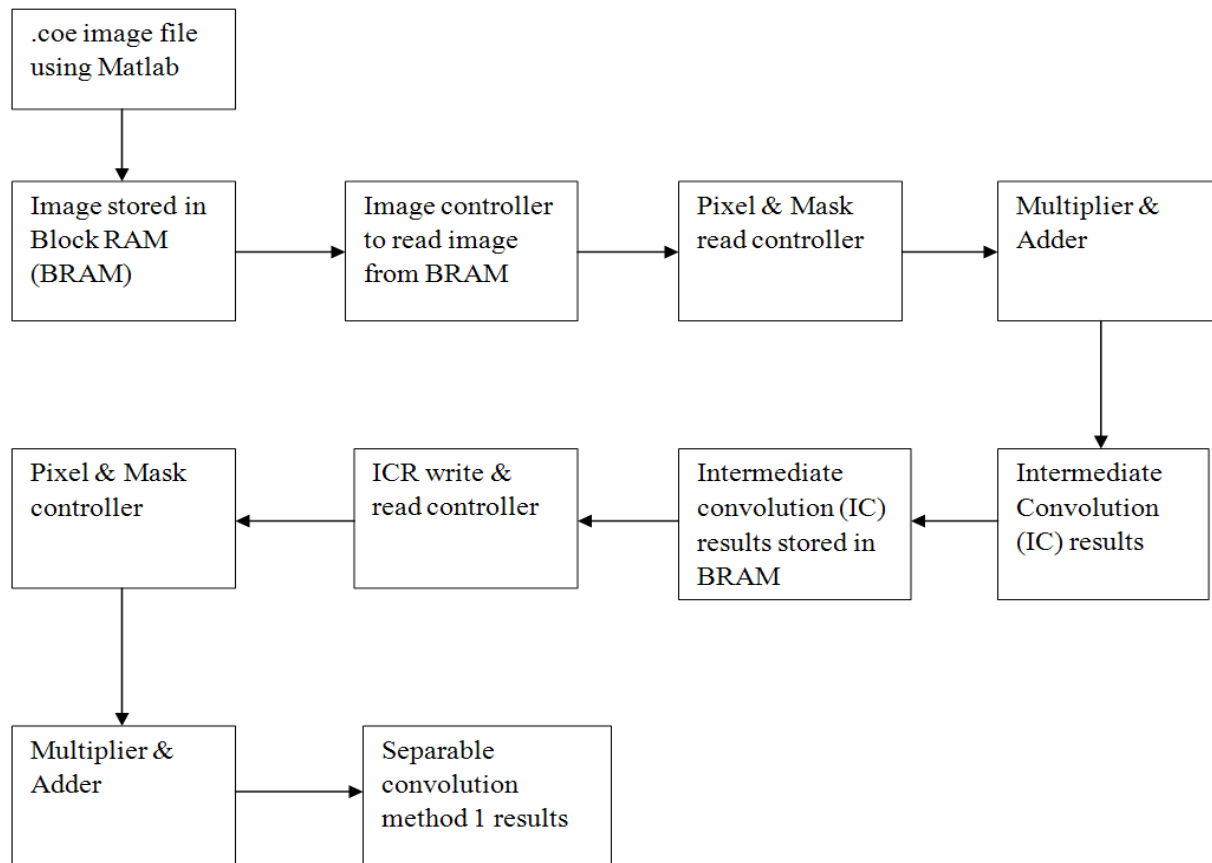


Figure 5.4: Block Diagram of Separable Convolution Method 1 (using multiple BRAMs)

In the schematic diagram below, the block named “topmodule2” is the intermediate convolution results write and read controller. The modules that store the image in BRAM and the image controller which reads the image from BRAM are embedded in topmodule2 block. The outputs of topmodule2 are connected to customfifo2. Customfifo2 access the required vertical convolution results and horizontal mask pixels, which are then connected to the three multipliers. The multiplier outputs are connected to an adder which provides a 70 bit output. The adder output is the separable convolution between the  $7 \times 7$  test image and the separable Gaussian masks  $3 \times 1$  &  $1 \times 3$ . A complete schematic diagram of separable convolution method 1 (using multiple BRAMs) is shown below:-

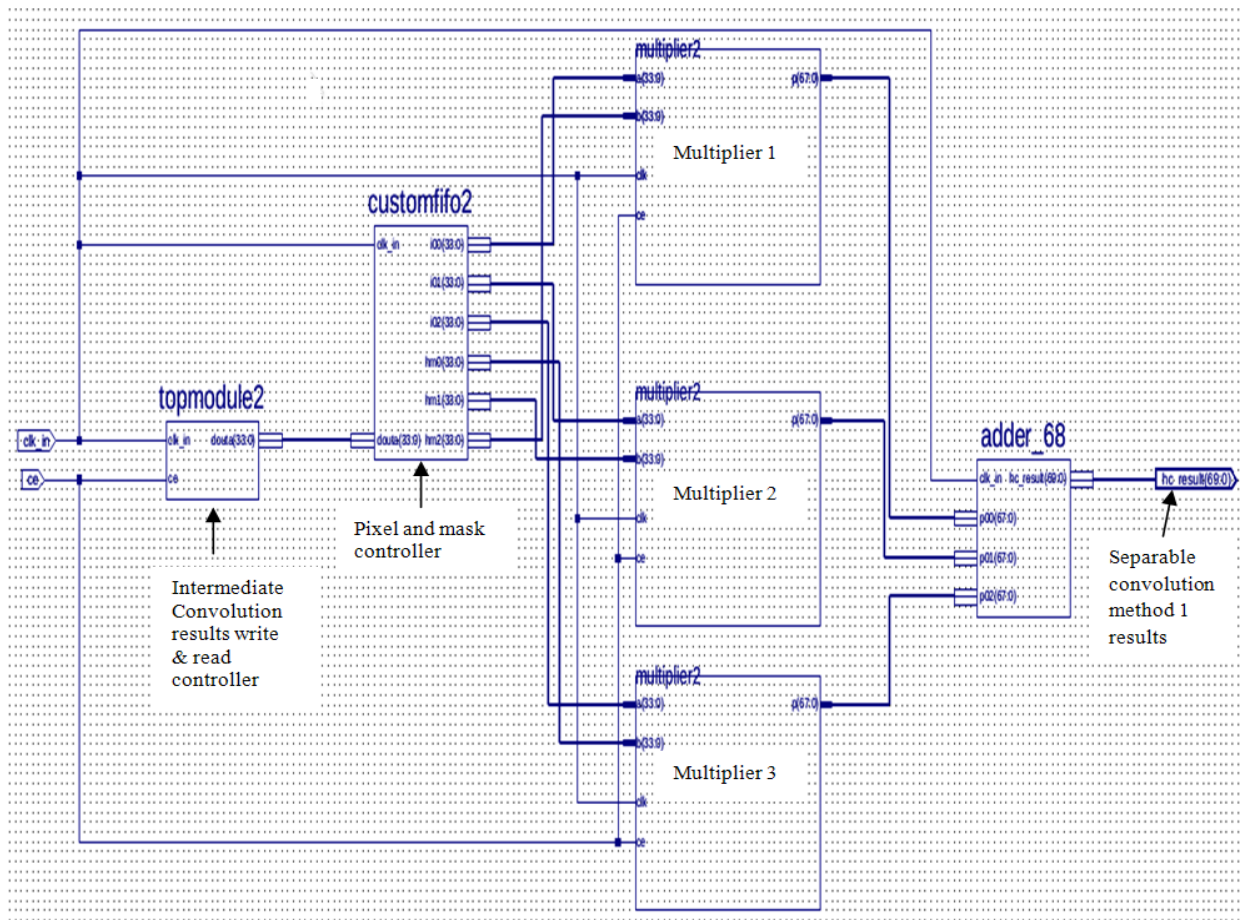


Figure 5.5: Schematic Diagram of Separable Convolution Method 1 (using multiple BRAMs)



The schematic design is simulated using Xilinx ISIM simulator for verification purpose. In the simulation diagram below, the reader may observe at the annotations, the image pixel controller outputs and mask controller outputs, the multiplier outputs and finally the separable convolution method 1 results. The simulation results are compared with Matlab, and are provided in the appendix. The simulation results of separable convolution method 1 are shown below:-

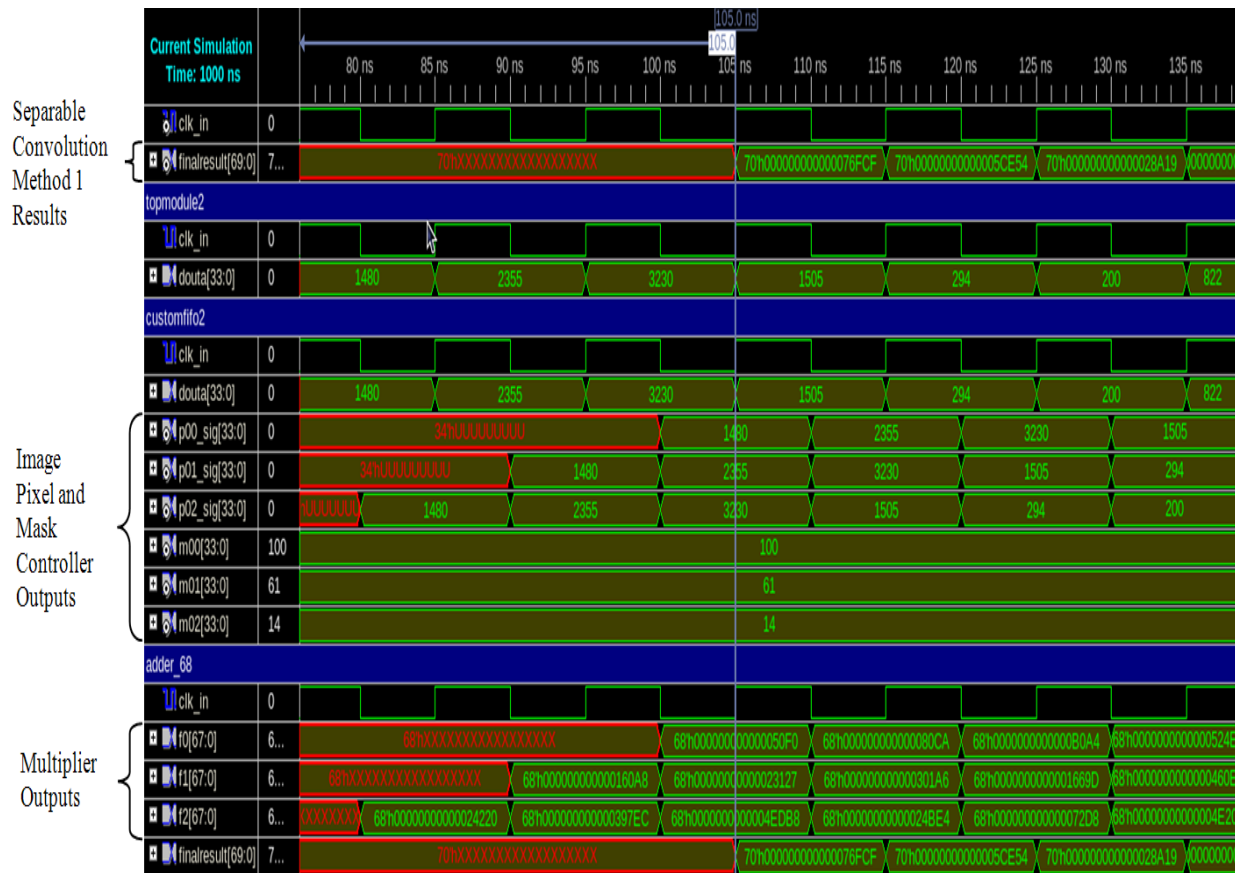


Figure 5.6: Simulation Results of Separable Convolution Method 1 (using multiple BRAMs)

Finally, the overall design is simulated using Xilinx XST Synthesizer to obtain the logic or hardware resource utilization on the target device. The design summary of the separable convolution method 1 is shown below:-


Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	553	27,392	2%	
Number of 4 input LUTs	711	27,392	2%	
<b>Logic Distribution</b>				
Number of occupied Slices	579	13,696	4%	
Number of Slices containing only related logic	579	579	100%	
Number of Slices containing unrelated logic	0	579	0%	
<b>Total Number of 4 input LUTs</b>	<b>729</b>	<b>27,392</b>	<b>2%</b>	
Number of bonded <a href="#">IOBs</a>	72	556	12%	
Number of RAMB16s	2	136	1%	
Number of MULT18X18s	15	136	11%	
Number of BUFGMUXs	2	16	12%	

Table 5.6: Device Utilization Summary of Separable Convolution Method 1

In the simulation results, it can be observed that the total number of clock cycles required for completing the separable convolution between a  $7 \times 7$  test image and a  $3 \times 1$  &  $1 \times 3$  Gaussian masks using the method of multiple BRAMs is equal to 108. Hence the separable convolution method1 (using multiple BRAMs) is approximately 2 clocks per pixel.

### 5.2.3 Separable convolution method 2 (using FIFO)

As discussed in section 4.2 of chapter 4, a Gaussian Mask is separable. Separable Gaussian mask is derived using equations [3] and [4] with mean equal to zero,  $\sigma$  equal to 1 and normalizing factor  $N = 0.0016$  are shown below:-

61	100	14
----	-----	----

Table 5.7: Horizontal Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

61
100
14

Table 5.8: Vertical Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

Similar to the regular convolution approach presented in section 5.2.1, BRAM is used to store a  $7 \times 7$  test image using .coe file [31] and an image controller is designed to access the stored image in the BRAM. The obtained image pixels and mask pixels are controlled using the pixel and mask controller blocks. A multiplier is designed using IP core [32]. The inputs to multiplier are obtained from the pixel and mask controller blocks. The multiplier block generates an output which is represented using  $2n-1$ bits. The multiplier inputs are represented using  $n$  bits. In this thesis work  $n$  was set equal to 16. The multiplier outputs are then given to an adder which provides a 34 bit output. The adder output is the vertical (intermediate) convolution result between the  $7 \times 7$  test image and the  $3 \times 1$  vertical Gaussian mask. Instead of writing the vertical convolution results into BRAM we save few rows and columns of vertical convolution result (i.e.

7 rows and 3 columns) in a 2D array vector. Parallelism is implemented, which yields in obtaining final convolution result in parallel with the vertical or intermediate convolution result. A read controller is designed to read the intermediate results saved in 2 dimensional arrays. At pixel and mask controller block, vertical Gaussian mask pixels (34 bits) and vertical convolution result pixels (34 bits) are accessed and given to multiplier and adder block. The 70 bit output obtained is the final result of separable convolution using method 2 between the  $7 \times 7$  test image and the  $1 \times 3$  horizontal Gaussian mask. The block diagram representation of separable convolution method 2 (using FIFO) is shown below:-

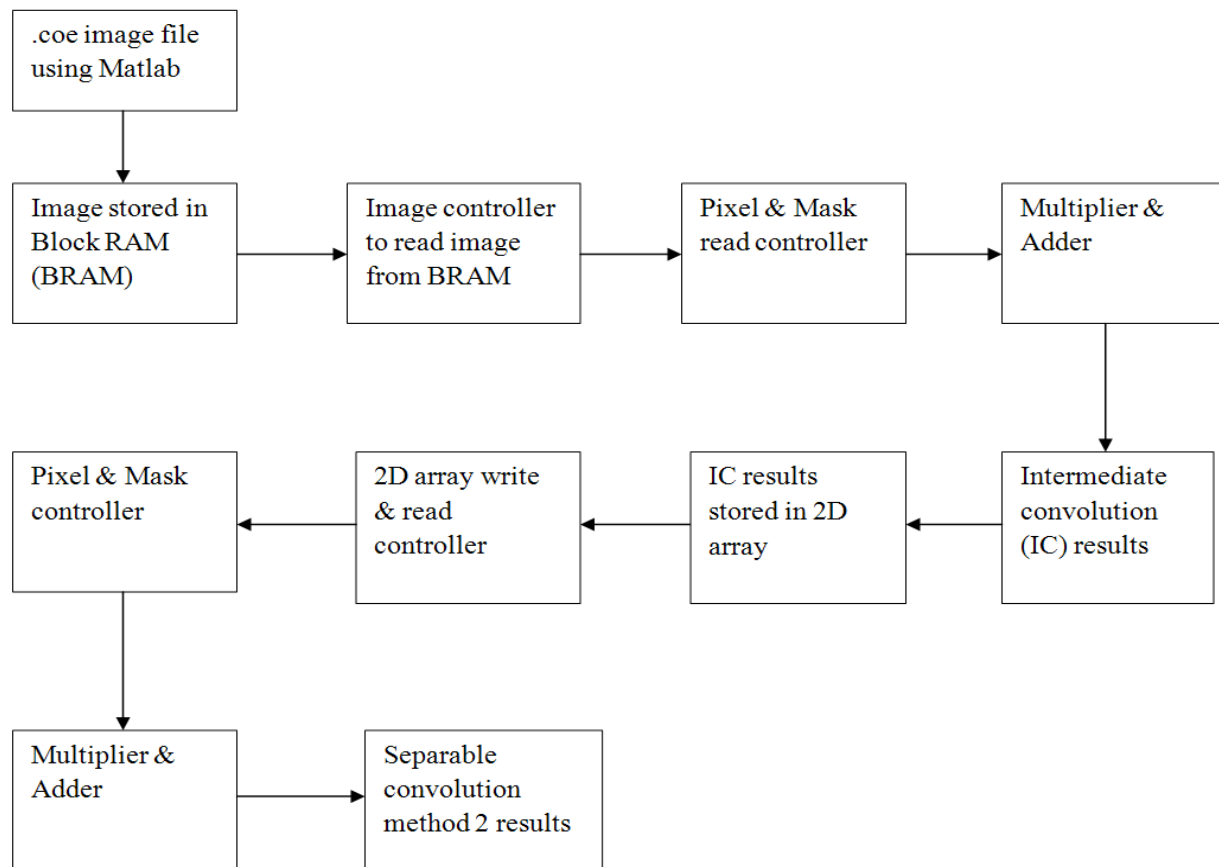


Figure 5.7: Block Diagram of Separable Convolution Method 2 (using FIFO)

In the schematic diagram below, the block named with “topmodule1” is the “intermediate convolution results write and read controller”. The modules that store the image in BRAM and the image controller which reads the image from BRAM are embedded in topmodule2 block. The outputs of topmodule1 are connected to the seperable2\_controller. Seperable2\_controller access required vertical convolution results and horizontal mask pixels which are then connected to 3 multipliers. The multiplier outputs are connected to an adder which provides a 70 bit output. The adder output is the separable convolution between the  $7 \times 7$  test image and the separable Gaussian masks  $3 \times 1$  &  $1 \times 3$ . A complete schematic diagram of separable convolution method 2 (using FIFO) is shown below:-

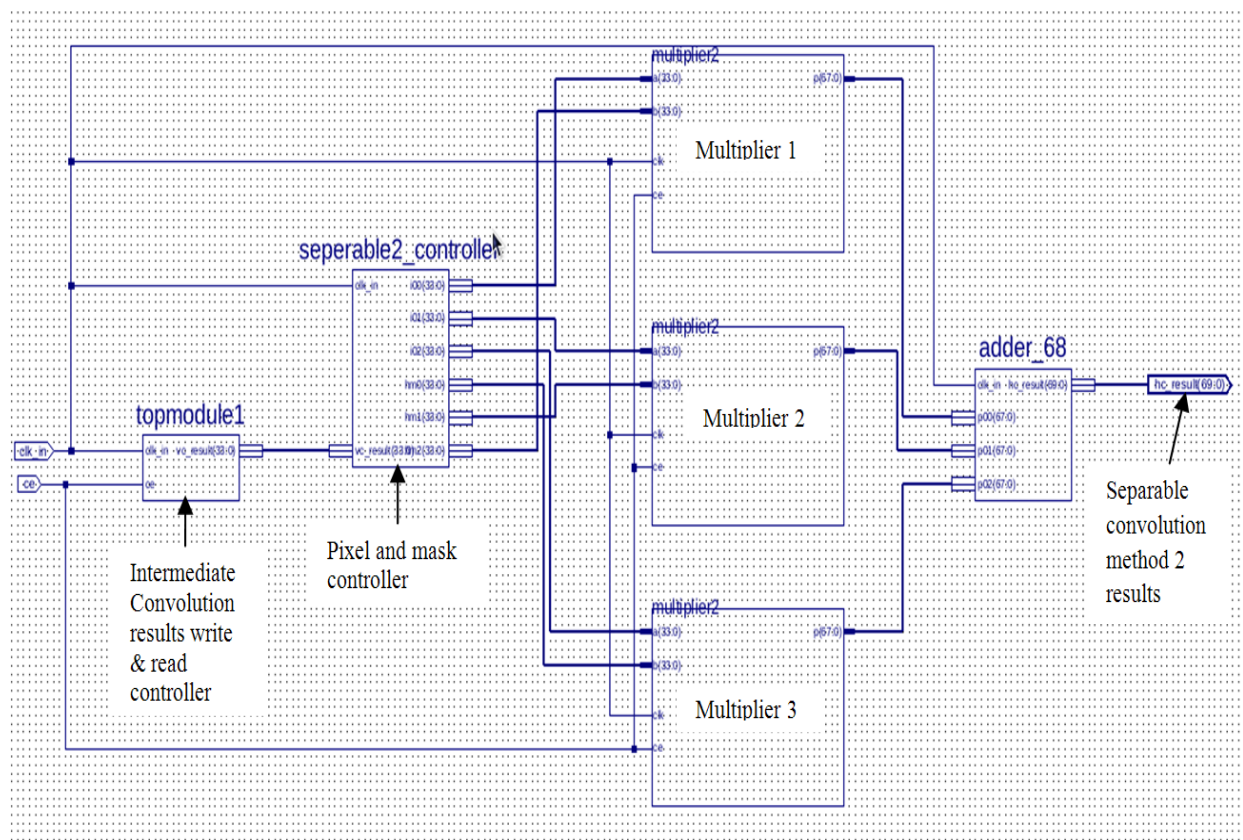


Figure 5.8: Schematic Diagram of Separable Convolution Method 2 (using FIFO)

The schematic design is simulated using Xilinx ISIM simulator for verification purpose. In the simulation diagram below, the reader may observe at the annotations, the image pixel controller outputs and mask controller outputs, the multiplier outputs and finally the separable convolution method 2 results. The above obtained simulation results are verified with Matlab and are provided in the appendix. The simulation results of separable convolution method 2 are shown below:-

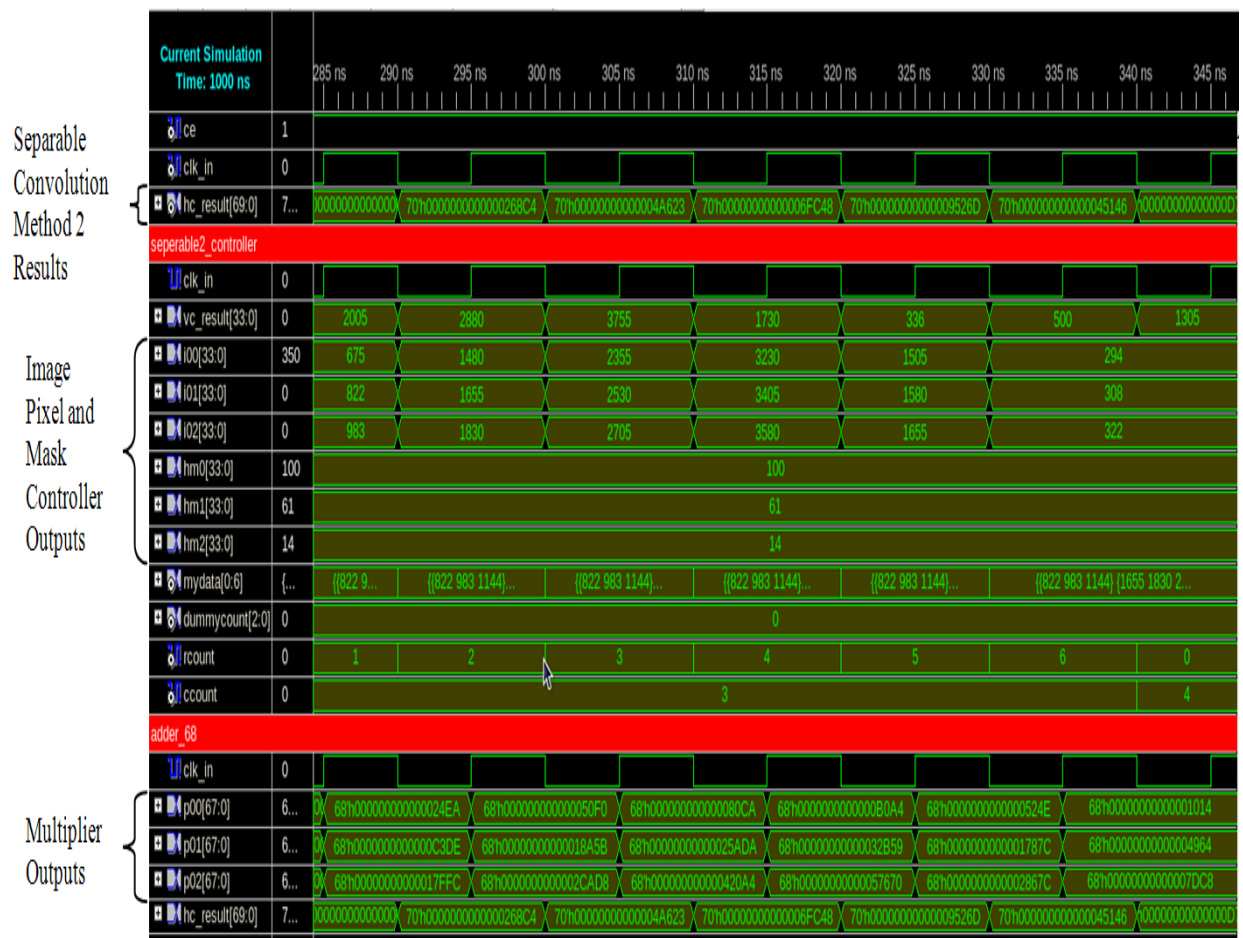


Figure 5.9: Simulation Results of Separable Convolution Method 2 (using FIFO)

Finally, the overall design is simulated using Xilinx XST Synthesizer to obtain the logic or hardware resource utilization on the target device. The design summary of separable convolution method 2 is shown below:-


Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	1389	13696	10%
Number of Slice Flip Flops	1073	27392	3%
Number of 4 input LUTs	2379	27392	8%
Number of bonded IOBs	72	556	12%
Number of BRAMs	1	136	0%
Number of MULT18X18s	15	136	11%
Number of GCLKs	1	16	6%

Table 5.9: Device Utilization Summary of Separable Convolution Method 2

In the simulation results, it can be observed that the total number of clock cycles required for completing the separable convolution between a  $7 \times 7$  test image and a  $3 \times 1$  &  $1 \times 3$  Gaussian masks using the method of FIFO is equal to 62. Hence the separable convolution method 2 (using multiple BRAMs) is approximately 1 clock per pixel.

### 5.2.4 Comparisons of convolution methods

For comparisons, we used a  $7 \times 7$  test image and  $3 \times 3$  Gaussian mask with both pixels represented using 16 bits and the target device is XU2VP30-FFG896(-7). A comparison table is presented below to explain which method is more feasible for applying steerability.

Methods ( $7 \times 7$ image and $3 \times 3$ Gaussian mask)	Slices [13696]	Slice flip flops [27392]	4 input LUT's [27392]	Bonded IOBs [556]	BRA Ms [136]	Multipli ers [136]	Clock s per pixel
Two Dimensional convolution	414 (2%)	596(2%)	306(1%)	36(6%)	1(0%)	9(6%)	~3
Separable Convolution Method 1	579(4%)	553(2%)	729(2%)	72(12%)	2(1%)	15(11%)	~2
Separable Convolution Method 2	1389(10)	1073(3%)	2379(8%)	72(12%)	1(0%)	15(11%)	~1

Table 5.10: Comparison of Convolution Methods

From the above comparisons, two dimensional convolution method is preferable as it uses few resources with satisfactory performance. Practically, implementation might not be possible if we go for larger mask sizes as it requires 3 clocks per pixel and more number of multipliers. Hence the separable convolution method 2 is most favorable in terms of performance for larger mask sizes, as it requires only 1 clock per pixel and less number of multipliers when compared with two dimensional convolution.



### 5.2.5 Extension of separable convolution method 2 (using FIFO)

The chosen convolution method i.e. separable convolution method 2 (using FIFO) is extended for a larger image of  $48 \times 48$  and a separable Gaussian masks of  $1 \times 9$  and  $9 \times 1$ . A Matlab program was used for generating  $48 \times 48$  gray scale image is provided in the appendix. Image pixels are represented using 8 bits. Separable Gaussian masks of  $1 \times 9$  and  $9 \times 1$  derived using equation [3] & [4] with mean equal to zero,  $\sigma$  equal to 1 and normalizing factor  $N = 0.0016$  are shown below:-

100	61	14	1	0	0	0	0	0
-----	----	----	---	---	---	---	---	---

Table 5.11: Horizontal Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

100
61
14
1
0
0
0
0
0

Table 5.12: Vertical Gaussian Mask with Mean = 0,  $\sigma = 1$  and  $N = 0.0016$

A complete schematic diagram of separable convolution method 2 extended to a larger image of  $48 \times 48$  and separable Gaussian masks of  $9 \times 1$  and  $1 \times 9$  is shown below:-

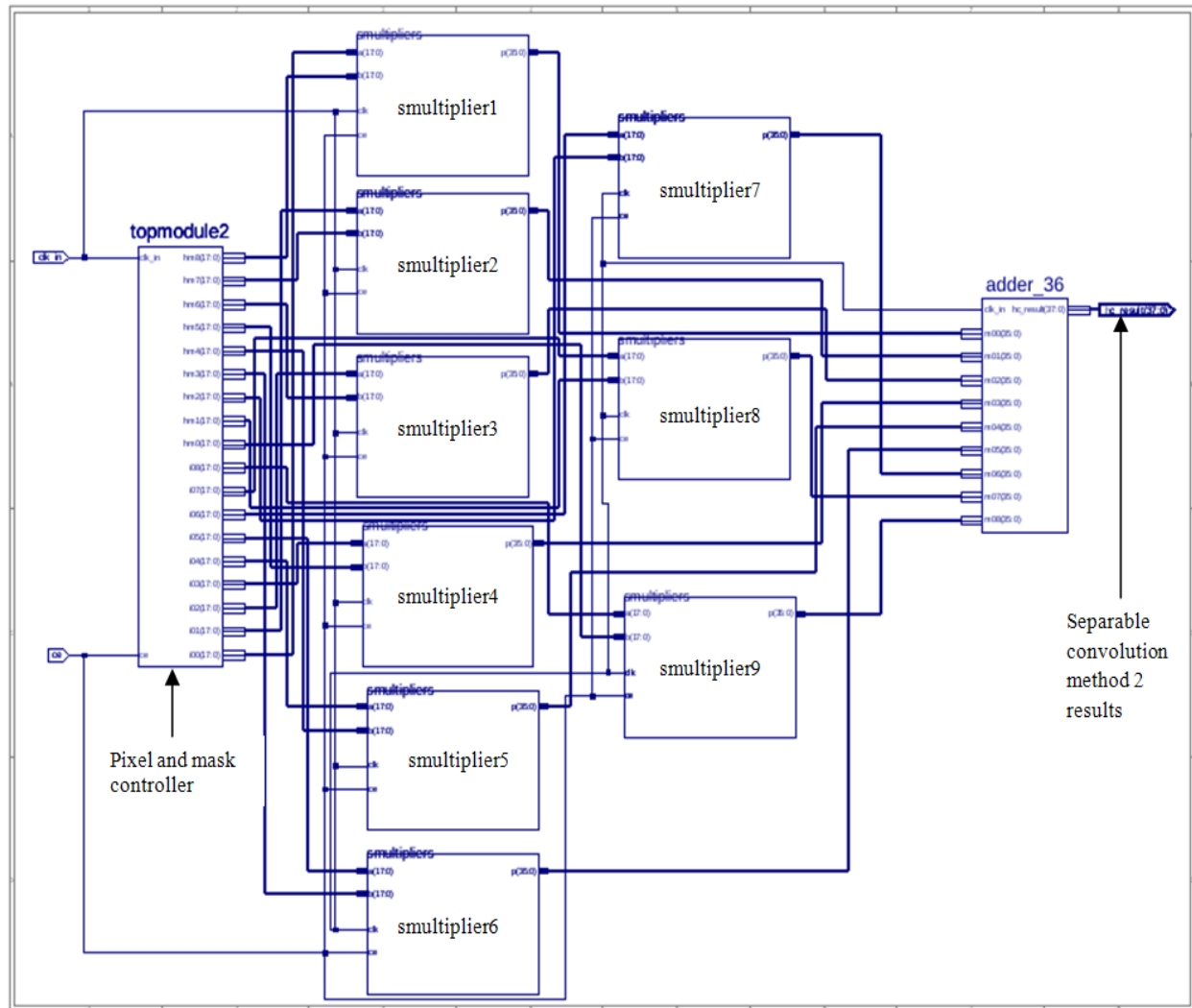


Figure 5.10: Schematic Diagram of Separable Convolution Method 2 extended to a  $48 \times 48$  image and Gaussian masks of  $9 \times 1$  and  $1 \times 9$

The schematic design is simulated using Xilinx ISIM simulator for verification. In the simulation diagram below, the reader may observe at the annotations, the image pixel controller outputs and mask controller outputs, the multiplier outputs and finally the separable convolution method 2 results. The above obtained simulation results are verified with Matlab. The simulation results of separable convolution method 2 extended to a larger image of 48×48 and separable Gaussian masks of 9×1 and 1×9 is shown below:-

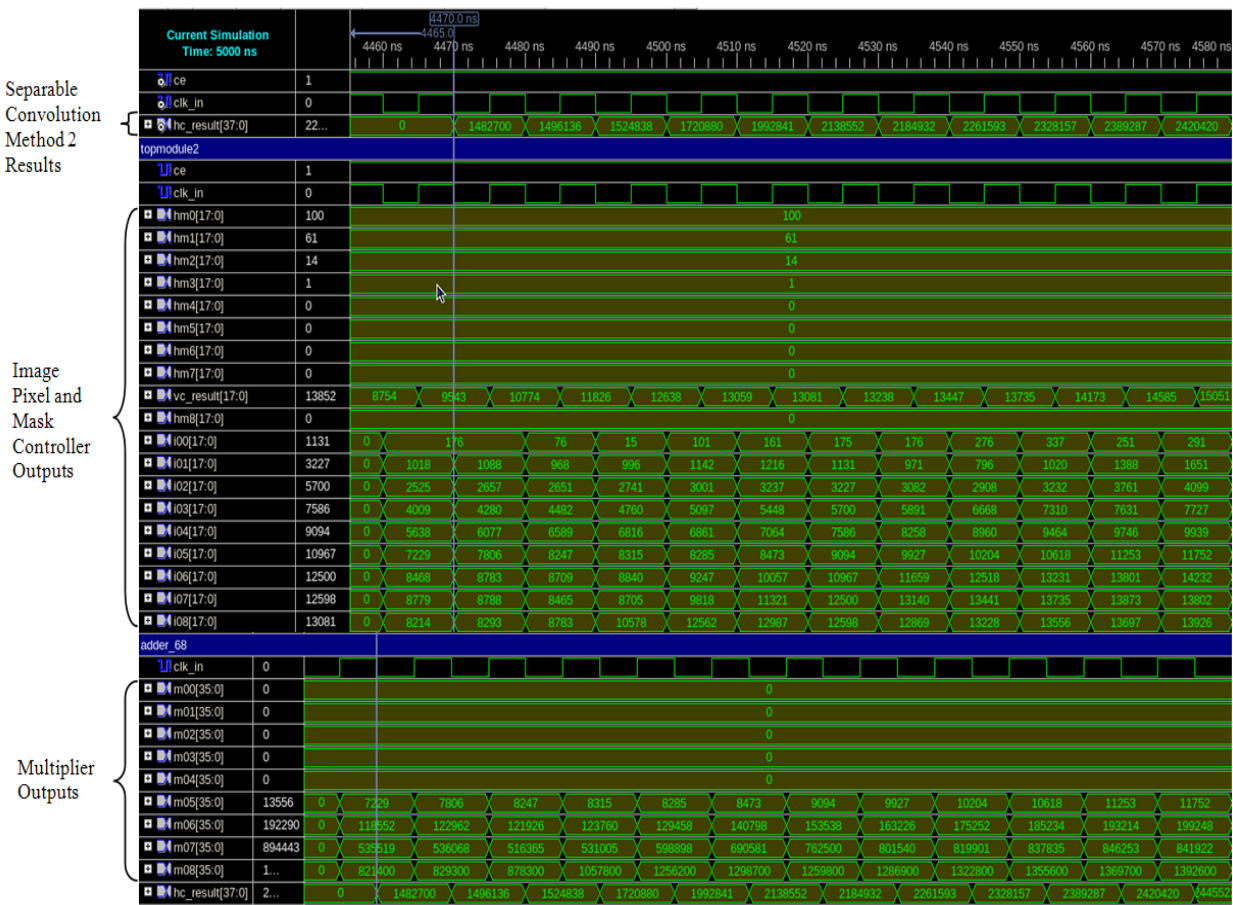


Figure 5.11: Simulation results of Separable Convolution Method 2 extended to a 48×48 image and Gaussian masks of 9×1 and 1×9

The design summary of separable convolution method 2 extended to a  $48 \times 48$  image size and Gaussian mask of  $9 \times 1$  and  $1 \times 9$  are shown below:-

Logic Utilization	Used	Available	Utilization
Number of Slices	10813	13696	78%
Number of Slice Flip Flops	7198	27392	26%
Number of 4 input LUTs	18906	27392	69%
Number of bonded IOBs	40	556	7%
Number of BRAMs	2	136	1%
Number of MULT18X18s	45	136	33%
Number of GCLKs	1	16	6%

Table 5.13: Device Utilization Summary of Separable Convolution Method 2 extended to  $48 \times 48$  image size and Gaussian mask of  $9 \times 1$  and  $1 \times 9$

In the simulation results, it can be observed that the total number of clock cycles required for completing the separable convolution between a  $48 \times 48$  test image and a  $9 \times 1$  &  $1 \times 9$  Gaussian masks using the method of FIFO is equal to 2130. Hence the separable convolution method 2 (using FIFO) is approximately 1 clock per pixel.

After ensuring successful working of separable convolution method 2(using FIFO), concept of steerability is applied on the above obtained final results which is explained in next section.

### 6.3 Proposed Steerable Concept Implementation

Steerability is applied on the obtained results of separable convolution (method 2) between the 48×48 image and the Gaussian masks of 9×1 and 1×9. For applying steerability, a steerable Gaussian mask is derived using equation (9) and decimation factor using the equation (10).

The derived steerable Gaussian masks of 7×1 and 1×7 for mean = 0,  $\sigma_x = 3$ ,  $\sigma_y = 5$ , Normalizing factor  $N = 0.001$  and decimation factor  $D = 3$  are shown below:-

8	33	76	100	76	33	8
---	----	----	-----	----	----	---

Table 5.14: Horizontal Gaussian Mask with Mean = 0,  $\sigma_x = 3$ ,  $\sigma_y = 5$ ,  $N = 0.001$

8
33
76
100
76
33
8

Table 5.15: Vertical Gaussian Mask with Mean = 0,  $\sigma_x = 3$ ,  $\sigma_y = 5$ ,  $N = 0.001$

In the steerable concept, we access pixels in different directions depending on the decimation factor. Then, the pixels are multiplied with the weights of Gaussian mask and finally given to adder to obtain final steerable results in a particular direction. For example, the decimation factor used here is  $D = 3$ . The pixels are accessed in different directions such as horizontal, vertical, diagonal and etc which are at a distance of 3 from each other and this is continued till the end of the image. The final results obtained in each particular direction are our required steerable results. The block diagram representation of implementing steerable concept on the results of separable convolution method 2 in horizontal and vertical direction is shown below:-

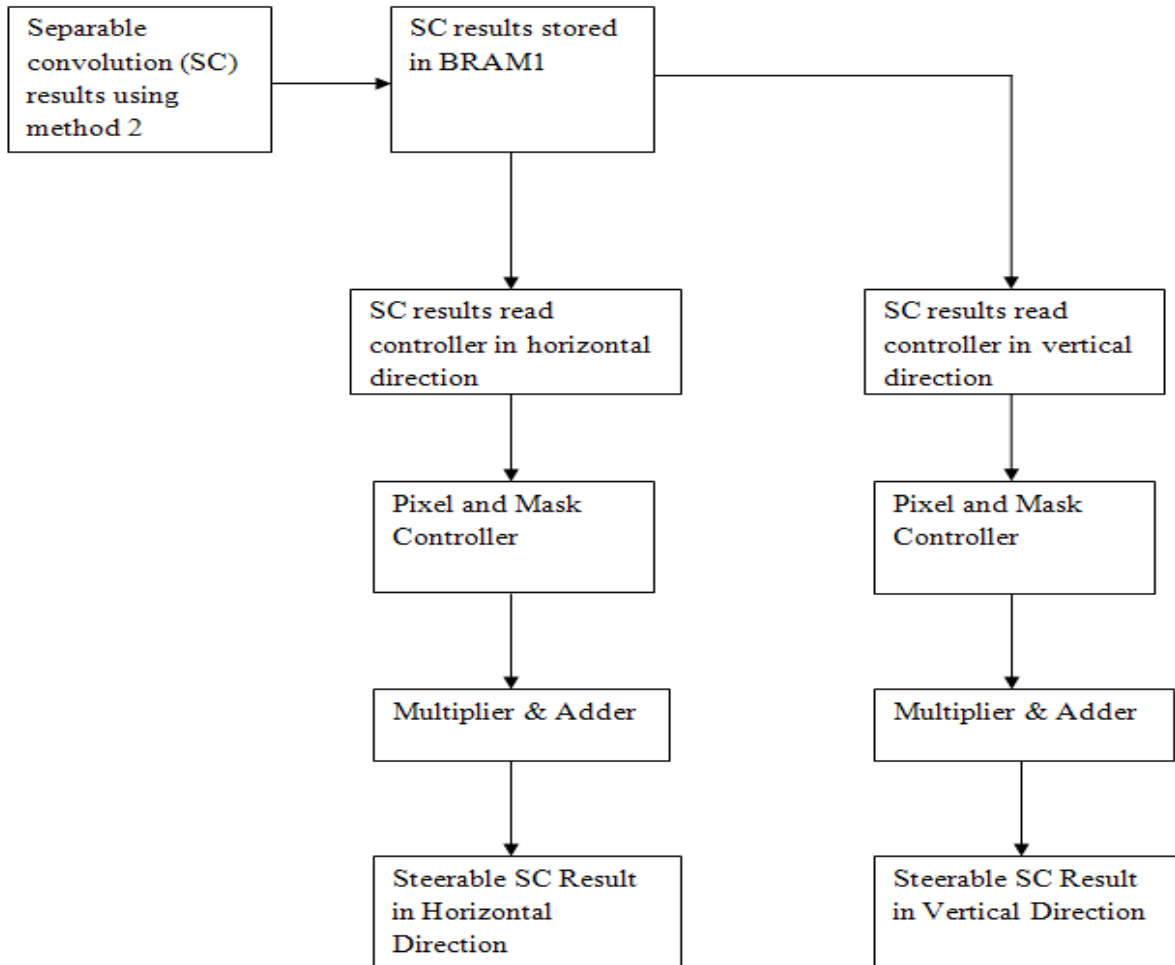


Figure 5.12: Block Diagram of Steerable Implementation

A complete schematic diagram of steerable implementation on  $48 \times 48$  image using Gaussian mask of  $7 \times 1$  and  $1 \times 7$  in horizontal, vertical and diagonal directions is shown below:-

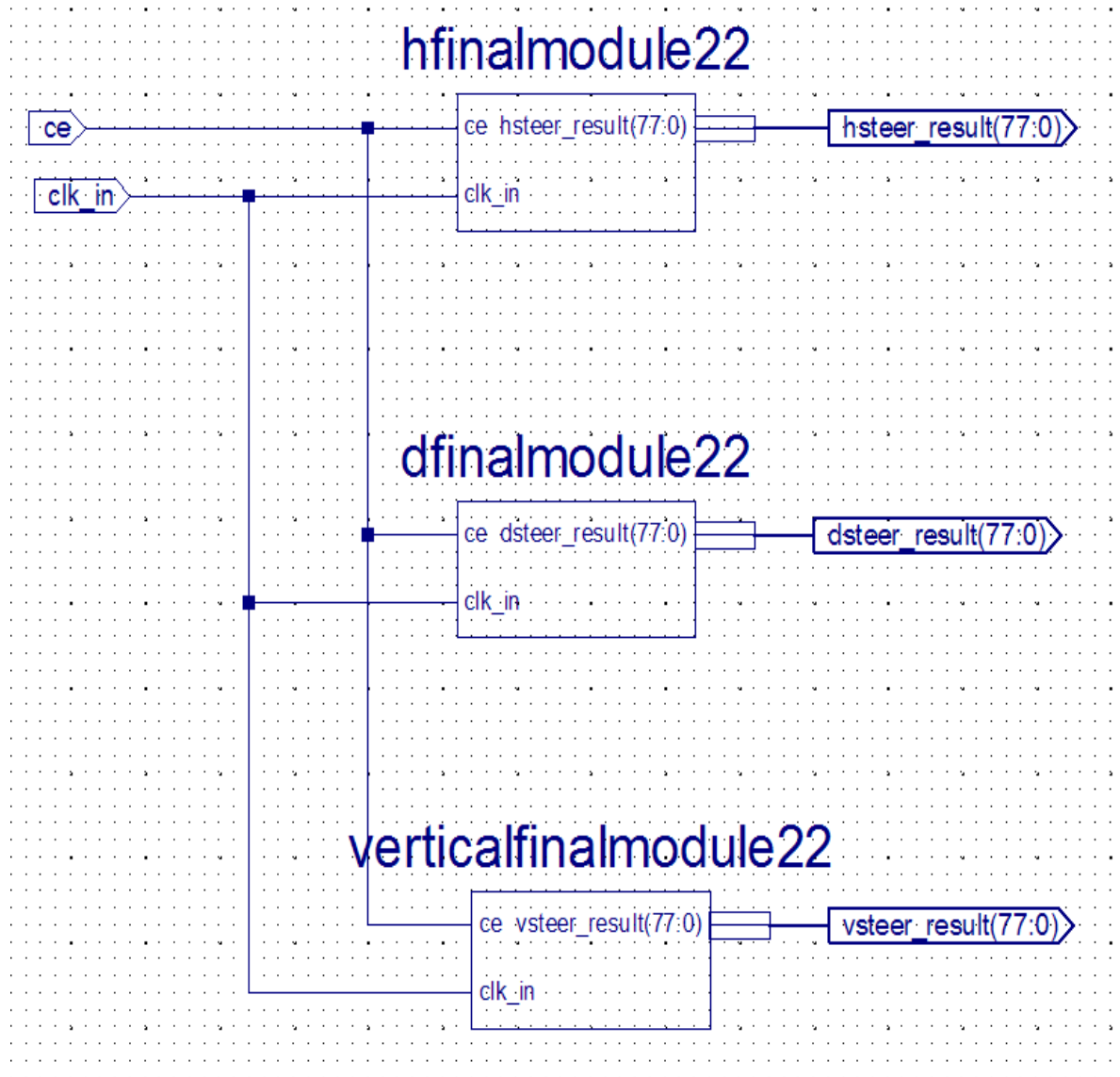


Figure 5.13: Schematic Diagram of Steerable Implementation

The design summary of steerable implementation on 48×48 image using Gaussian mask of 7×1 and 1×7 in horizontal, vertical and diagonal directions is obtained for a Xilinx Virtex4 shown below:-

Device Utilization Summary (estimated values)			<a href="#">[-]</a>
Logic Utilization	Used	Available	Utilization
Number of Slices	36358	49152	73%
Number of Slice Flip Flops	24868	98304	25%
Number of 4 input LUTs	68079	98304	69%
Number of bonded IOBs	236	768	30%
Number of FIFO16/RAMB16s	24	240	10%
Number of GCLKs	1	32	3%

Table 5.16: Device Utilization summary of steerable implementation on a virtex4 board.

Due to limited number of resources on the target device Xilinx Virtex 2 pro, device utilization summary of steerable implementation is obtained for Xilinx Virtex 4 board which has more number of resources. From the obtained simulation results, the performance of implementing steerable concept on a 48×48 image and 1×7, 7×1 Gaussian mask in horizontal, vertical and diagonal directions is achieved in 2 clocks per pixel.



## CHAPTER 6

### Conclusions & Future Work

#### 6.1 Summary & Conclusions

The following summary and conclusions were drawn based on implementation and experimentation:-

1. Three different techniques of convolution are developed and an assessment of these methods is prepared by considering device resource utilization and performance in terms of clocks per pixel.
2. The second separable implementation presented in this thesis requires the smallest number of clock cycles per pixels compared to the other implementations.
3. The concept of steerability is applied in horizontal, vertical and diagonal directions on a  $48 \times 48$  smoothed image. The smoothed image is obtained by convolving the original image  $48 \times 48$  with  $1 \times 9$  &  $9 \times 1$  Gaussian masks. Three  $7 \times 1$  Gaussian masks were used for the steerable outputs, which are acquired by convolving original using Gaussian mask of  $7 \times 1$ . The steerable filtering technique is synthesized and its effectiveness is confirmed using simulation results.
4. Due to the limitations of target device, Xilinx Virtex 2 Pro board, and software issues or bugs related to ISE 10.3, the separable convolution using method 2 is put into operation on the target device for smaller  $1 \times 3$  and  $3 \times 1$  Gaussian masks and the input image of  $48 \times 48$ .

## **6.2 Limitations**

The following limitations are listed below:-

1. The target device VirtexII Pro is not supported for implementation by newer versions of Xilinx ISE Design Suite Software (11 and higher versions).
2. The previous version i.e. ISE 10.3 has software bugs which does not provide proper simulation and synthesis results when a large number of multipliers are used.

## **6.3 Future Work**

The following work has to be performed in order to improve the efficiency of hardware implementation of steerability concept.

1. An efficient way of accessing image pixels along different directions from block RAM in less number of clock cycles.
2. Exploring an efficient way of representing image and mask pixels in less number of bits.
3. Efficient methods of dropping off the unused most significant bits before and after the multiplier or adder stage thereby reducing resource utilization of multipliers.
4. Coding efficiently to improve parallelism and reducing longest path delays to improve pipelining.

## Bibliography

1. Moore's Law made real by Intel Innovations [www.intel.com/technology/mooreslaw/](http://www.intel.com/technology/mooreslaw/)
2. Dr. Dimitrios Charalampidis, "Efficient Directional Gaussian Smoothers", IEEE Geoscience and Remote Sensing, July 2009
3. D. Brown, R. Francis, J. Rose, Z. Vransic, Field-Programmable Gate Arrays, Kluwer Academic Publishers, May 1992.
4. Pong P. Chu, A text book on "FPGA prototyping by VHDL examples Xilinx Spartan 3 version". 2008
5. J. Webster, A document on "Programmable Logic Arrays", Wiley Encyclopedia of Electrical and Electronics Engineering, 1999
6. Richard Waln, Ian Bush, Martyn Guest, Miles Deegan, Igor Kozln and Christine Kitchen, A report on "An overview of FPGAs and FPGA programming; Initial experiences at Daresbury", Nov 2006.
7. Xilinx, "ISE 10.3 Design Suite and software manuals",  
<http://www.xilinx.com/itp/xilinx10/books/manuals.pdf>
8. Xilinx, "Xilinx University Program Virtex-II Pro Development System Hardware Reference Manual-UG069", July 2009
9. Xilinx, "Virtex-II Pro Platform FPGA User Guide-UG012", v4.2, Nov 2007

10. Daggu Venkateshwar Rao and Muthukumar Venkatesan , “Implementation and Evaluation of Image Processing Algorithms on Reconfigurable Architecture using C-based Hardware Descriptive Languages”, International Journal of Engineering and Applied Sciences, 2006
11. IEEE Std 1076, “ IEEE Standard VHDL Language Reference Manual”, Jan 2000
12. V. Lakshmanan, “A Separable Filter for Directional Smoothing”, IEEE Geoscience and Remote Sensing Letters, July 2004.
13. William T. Freeman and Edward H. Adelson, “The Design and Use of Steerable Filters”, IEEE Transaction of Pattern Analysis and Machine Intelligence, Sept 1991.
14. C. Chou, S. Mohana krishnan, J. Evans, “FPGA Implementation of Digital Filters”, Proc. ICSPAT, 1993
15. Hui Zhang, Mingxin Xia, and Guangshu Hu, “A Multiwindow Partial Buffering Scheme for FPGA Based 2-D Convolvers, IEEE Transaction on Circuits and Systems, Feb 2007.
16. Dr. Haled Benkrid, Mr. Samir Belkacemi, “Design and Implementation of a 2D Convolution Core for Video Applications on FPGAs”, International Workshop on Digital and Computational Video, Nov 2002.
17. Francisco Cardells-Tormo, Pep-Lluis Molinet,” Area Efficient 2-D Shift Variant Convolvers for FPGA Based Digital Image Processing”, IEEE Transaction on Circuits and Systems, Feb 2006.
18. Manish Kumar Birla, “FPGA Based Reconfigurable Platform for Complex Image Processing”, IEEE International Conference on Electro/information Technology, Nov 2006.

19. Bruce A. Draper, J. Ross Beveridge, A.p Willem Bhom, Charles Ross, Monica Chawathe, Jeffrey Hammes, A publication on “Accelerated Image Processing on FPGAs”, 2008.
20. G. Sutter, E. Todorovich, G.Bioul, M. Vazquez, J-P. Deschamps, “FPGA Implementation of BCD Multipliers”, International Conference on Reconfigurable Computing and FPGAs, 2009.
21. Lakshmanan, Masuri Othman and Mohamad Alauddin Mohd. Ali, “High Performance Parallel Multiplier using Wallace-Booth Algorithm”, Proceeding of ICSE, 2002.
22. Schoeters Jurgen, Van Winkel Jan, Goedeme Toon, Meel Jan, “In-Vehcile Movie Streaming using an Embedded System with MOST Interface”, Automative Electronics-3<sup>rd</sup> Institution of Engineering and Technology Conference, June 2007.
23. A. W. Azman, A. Bigdeli, Y. M. Mustafah, B. C. Lovell, “ Optimizing Resources of an FPGA-based Smart Camera Architecture”, Proceedings of Digital Image Computing Techniques and Applications, 2008.
24. WM EI-Medany, MR Hussain,” FPGA Based Advanced Real Traffic Light Controller System Design”, IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, Sept 2007.
25. Justin L. Tripp, Henning S. MOrtveit, Anders A. Hanson, Maya Gokhale, “ Metropolitan Road Traffic Simulation on FPGAs”, IEEE symposium on Field-Programmable Custom Computing Machines, 2005.
26. C. Berthaud, E. Bourennane, M. Paindavoine and C. Milan, “ Real Time Image Rotation on FPGA’s Board”, Proceedings of ICSP, 1998

27. Peter Mc. Curry, Fearghal Morgan, Liam Kilmartin,” Xilinx Implementation of Pixel Processor for Object Detection Applications”, Proceedings of Signals and Systems Conference, 2001.
28. Paul D. Fiore, Dane Kottke, Wojciech Krawiec, David Campagna, “ Efficient Feature Tracking with Application to Camera Motion Estimation”, IEEE , 2002
29. Kah-Howe Tan, Wen Fung Leong, Sameer Kadam, M.A Soderstrand and Louis G. Johnson, “ Public Domain Matlab Program to Generate Highly Optimized VHDL for FPGA Implementation”, IEEE, 2001.
30. K. R. Castleman, M. Schulze, Q, Wu, “Simplified Design of Steerable Pyramid Filters”, IEEE Proceedings, 1998.
31. Xilinx, “Block Memory Generator v 2.8 Logic Core”, Sept 2008.
32. Xilinx, “Logic Core IP Multiplier v 11.2”, Sept 2009.

# Appendix

## 2D Convolution Method VHDL Source Files

```
-----
-- filename: dualportram_image_controller.vhd
-- author: Arjun Joginipelly
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity dualportram_image_controller is
  Port ( clk_in : in  STD_LOGIC;
        douta : in  STD_LOGIC_VECTOR (15 downto 0);
        addra : out STD_LOGIC_VECTOR (5 downto 0);
        dina : out STD_LOGIC_VECTOR (15 downto 0);
        wea : out  STD_LOGIC;
        ena : out  STD_LOGIC;

        p0 : out std_logic_vector (15 downto 0);
        p1 : out std_logic_vector (15 downto 0);
        p2 : out std_logic_vector (15 downto 0);
        p3 : out std_logic_vector (15 downto 0);
        p4 : out std_logic_vector (15 downto 0);
        p5 : out std_logic_vector (15 downto 0);
        p6 : out std_logic_vector (15 downto 0);
        p7 : out std_logic_vector (15 downto 0);
        p8 : out std_logic_vector (15 downto 0);

        m0 : out std_logic_vector (15 downto 0);
        m1 : out std_logic_vector (15 downto 0);
        m2 : out std_logic_vector (15 downto 0);
        m3 : out std_logic_vector (15 downto 0);
        m4 : out std_logic_vector (15 downto 0);
        m5 : out std_logic_vector (15 downto 0);
        m6 : out std_logic_vector (15 downto 0);
        m7 : out std_logic_vector (15 downto 0);
        m8 : out std_logic_vector (15 downto 0);

        clk_out : out STD_LOGIC);
end dualportram_image_controller;

architecture Behavioral of dualportram_image_controller is
```

```

type state_reg_type is (initialstate,state1,state2,state3,state4,
state5,state6,state7,state8,state9,halt);
signal sreg:state_reg_type:=initialstate;

signal account:std_logic_vector(5 downto 0):="000000";
signal addra_sig:std_logic_vector(5 downto 0):="000000";
signal data_present:std_logic:='0';
signal p0_sig:std_logic_vector(15 downto 0):=(others=>'0');
signal p1_sig:std_logic_vector(15 downto 0):=(others=>'0');
signal p2_sig:std_logic_vector(15 downto 0):=(others=>'0');
signal p3_sig:std_logic_vector(15 downto 0):=(others=>'0');
signal p4_sig:std_logic_vector(15 downto 0):=(others=>'0');
signal p5_sig:std_logic_vector(15 downto 0):=(others=>'0');
signal p6_sig:std_logic_vector(15 downto 0):=(others=>'0');
signal p7_sig:std_logic_vector(15 downto 0):=(others=>'0');
signal p8_sig:std_logic_vector(15 downto 0):=(others=>'0');

begin
clk_out<=clk_in;
    process(clk_in)
    begin
        if(clk_in'event and clk_in<='0') then
            case sreg is
                when initialstate=> wea<='0';
                                                                    ena<='1';
                                                                    sreg<=state1;

                when state1=> data_present<='0';
                    p0_sig<=douta;
                                                                    addra_sig<=addra_sig+1;
                                                                    sreg<=state2;

                when state2=> p1_sig<=douta;
                                                                    addra_sig<=addra_sig+1;
                                                                    sreg<=state3;

                when state3=> p2_sig<=douta;
                                                                    addra_sig<=addra_sig+5;
                                                                    sreg<=state4;

                when state4=> p3_sig<=douta;
                                                                    addra_sig<=addra_sig+1;
                                                                    sreg<=state5;

                when state5=> p4_sig<=douta;
                                                                    addra_sig<=addra_sig+1;
                                                                    sreg<=state6;

                when state6=> p5_sig<=douta;
                                                                    addra_sig<=addra_sig+5;
                                                                    sreg<=state7;

                when state7=> p6_sig<=douta;

```



```

        addra_sig<=addra_sig+1;
        acount<=acount+1;
        sreg<=state8;

        when state8=> p7_sig<=douta;

        addra_sig<=addra_sig+1;
        sreg<=state9;

        when state9=> p8_sig<=douta;
            addra_sig<=acount;

        data_present<='1';
        if(acount=50) then
            sreg<=halt;
        else
            sreg<=state1;
        end if;

        when halt=> wea<='0';

        ena<='0';

    end case;
end if;
end process;

process(data_present)
begin
    if(data_present'event and data_present='1') then
        p0<=p0_sig;
        p1<=p1_sig;
        p2<=p2_sig;
        p3<=p3_sig;
        p4<=p4_sig;
        p5<=p5_sig;
        p6<=p6_sig;
        p7<=p7_sig;
        p8<=p8_sig;
    end if;
end process;

addra<=addra_sig;
m0<=x"0064";
m1<=x"003D";
m2<=x"000E";
m3<=x"003D";
m4<=x"0025";
m5<=x"0008";
m6<=x"000E";
m7<=x"0008";
m8<=x"0002";

end Behavioral;

```

-----  
-- filename: multiplier.vhd

-- author: Arjun Joginipelly  
-----

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synthesis translate_off
Library XilinxCoreLib;
-- synthesis translate_on
ENTITY multiplier IS
    port (
        clk: IN std_logic;
        a: IN std_logic_VECTOR(15 downto 0);
        b: IN std_logic_VECTOR(15 downto 0);
        ce: IN std_logic;
        p: OUT std_logic_VECTOR(31 downto 0));
END multiplier;
```

```
ARCHITECTURE multiplier_a OF multiplier IS
-- synthesis translate_off
component wrapped_multiplier
    port (
        clk: IN std_logic;
        a: IN std_logic_VECTOR(15 downto 0);
        b: IN std_logic_VECTOR(15 downto 0);
        ce: IN std_logic;
        p: OUT std_logic_VECTOR(31 downto 0));
end component;
```

```
-- Configuration specification
    for all : wrapped_multiplier use entity XilinxCoreLib.mult_gen_v10_1(behavioral)
        generic map(
            c_a_width => 16,
            c_b_type => 1,
            c_ce_overrides_sclr => 0,
            c_has_sclr => 0,
            c_round_pt => 0,
            c_model_type => 0,
            c_out_high => 31,
            c_verbosity => 0,
            c_mult_type => 1,
            c_ccm_imp => 0,
            c_latency => 1,
            c_has_ce => 1,
            c_has_zero_detect => 0,
            c_round_output => 0,
            c_optimize_goal => 1,
            c_xdevicefamily => "virtex2p",
            c_a_type => 1,
            c_out_low => 0,
            c_b_width => 16,
            c_b_value => "10000001");
```

```

-- synthesis translate_on
BEGIN
-- synthesis translate_off
U0 : wrapped_multiplier
    port map (
        clk => clk,
        a => a,
        b => b,
        ce => ce,
        p => p);
-- synthesis translate_on

END multiplier_a;

```

---

```

-- filename: adder.vhd

-- author: Arjun Joginipelly

```

---

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity adder_32 is
    Port ( clk_in : in  STD_LOGIC;
          x0 : in  STD_LOGIC_VECTOR (31 downto 0);
          x1 : in  STD_LOGIC_VECTOR (31 downto 0);
          x2 : in  STD_LOGIC_VECTOR (31 downto 0);
          x3 : in  STD_LOGIC_VECTOR (31 downto 0);
          x4 : in  STD_LOGIC_VECTOR (31 downto 0);
          x5 : in  STD_LOGIC_VECTOR (31 downto 0);
          x6 : in  STD_LOGIC_VECTOR (31 downto 0);
          x7 : in  STD_LOGIC_VECTOR (31 downto 0);
          x8 : in  STD_LOGIC_VECTOR (31 downto 0);
          genc_result : out  STD_LOGIC_VECTOR (33 downto 0));
end adder_32;

architecture Behavioral of adder_32 is

    signal x0_sig:std_logic_vector(33 downto 0);
    signal x1_sig:std_logic_vector(33 downto 0);
    signal x2_sig:std_logic_vector(33 downto 0);
    signal x3_sig:std_logic_vector(33 downto 0);
    signal x4_sig:std_logic_vector(33 downto 0);
    signal x5_sig:std_logic_vector(33 downto 0);
    signal x6_sig:std_logic_vector(33 downto 0);
    signal x7_sig:std_logic_vector(33 downto 0);

```

```

signal x8_sig:std_logic_vector(33 downto 0);

begin
    process(clk_in,x0,x1,x2,x3,x4,x5,x6,x7,x8)
    begin
        x0_sig<="00" & x0;
        x1_sig<="00" & x1;
        x2_sig<="00" & x2;
        x3_sig<="00" & x3;
        x4_sig<="00" & x4;
        x5_sig<="00" & x5;
        x6_sig<="00" & x6;
        x7_sig<="00" & x7;
        x8_sig<="00" & x8;
        genc_result<=x0_sig+x1_sig+x2_sig+x3_sig+x4_sig+x5_sig+x6_sig+x7_sig
                    +x8_sig;

        end process;
end Behavioral;

```

---

```
-- filename: resultmodule.vhd
```

```
-- author: Arjun Joginipelly
```

---

```

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;

```

```

entity resultmodule is
    port ( ce      : in  std_logic;
          clk_in   : in  std_logic;
          genc_result : out std_logic_vector (33 downto 0));
end resultmodule;

```

```

architecture BEHAVIORAL of resultmodule is
    signal XLXN_3   : std_logic_vector (15 downto 0);
    signal XLXN_6   : std_logic_vector (15 downto 0);
    signal XLXN_7   : std_logic_vector (15 downto 0);
    signal XLXN_8   : std_logic_vector (15 downto 0);
    signal XLXN_9   : std_logic_vector (15 downto 0);
    signal XLXN_10  : std_logic_vector (15 downto 0);
    signal XLXN_11  : std_logic_vector (15 downto 0);
    signal XLXN_12  : std_logic_vector (15 downto 0);
    signal XLXN_13  : std_logic_vector (15 downto 0);
    signal XLXN_14  : std_logic_vector (15 downto 0);
    signal XLXN_15  : std_logic_vector (15 downto 0);
    signal XLXN_16  : std_logic_vector (15 downto 0);
    signal XLXN_17  : std_logic_vector (15 downto 0);

```

```

signal XLXN_19 : std_logic_vector (15 downto 0);
signal XLXN_20 : std_logic_vector (15 downto 0);
signal XLXN_21 : std_logic_vector (15 downto 0);
signal XLXN_22 : std_logic_vector (15 downto 0);
signal XLXN_23 : std_logic_vector (15 downto 0);
signal XLXN_39 : std_logic_vector (31 downto 0);
signal XLXN_40 : std_logic_vector (31 downto 0);
signal XLXN_41 : std_logic_vector (31 downto 0);
signal XLXN_42 : std_logic_vector (31 downto 0);
signal XLXN_43 : std_logic_vector (31 downto 0);
signal XLXN_44 : std_logic_vector (31 downto 0);
signal XLXN_45 : std_logic_vector (31 downto 0);
signal XLXN_46 : std_logic_vector (31 downto 0);
signal XLXN_47 : std_logic_vector (31 downto 0);
    attribute box_type:string;
component topmodule
    port ( clk_in : in    std_logic;
          p0  : out  std_logic_vector (15 downto 0);
          p1  : out  std_logic_vector (15 downto 0);
          p2  : out  std_logic_vector (15 downto 0);
          p3  : out  std_logic_vector (15 downto 0);
          p4  : out  std_logic_vector (15 downto 0);
          p5  : out  std_logic_vector (15 downto 0);
          p6  : out  std_logic_vector (15 downto 0);
          p7  : out  std_logic_vector (15 downto 0);
          p8  : out  std_logic_vector (15 downto 0);
          m0  : out  std_logic_vector (15 downto 0);
          m1  : out  std_logic_vector (15 downto 0);
          m2  : out  std_logic_vector (15 downto 0);
          m3  : out  std_logic_vector (15 downto 0);
          m4  : out  std_logic_vector (15 downto 0);
          m5  : out  std_logic_vector (15 downto 0);
          m6  : out  std_logic_vector (15 downto 0);
          m7  : out  std_logic_vector (15 downto 0);
          m8  : out  std_logic_vector (15 downto 0));
end component;

component multiplier
    port ( a  : in  std_logic_vector (15 downto 0);
          b  : in  std_logic_vector (15 downto 0);
          clk : in  std_logic;
          ce  : in  std_logic;
          p   : out std_logic_vector (31 downto 0));
end component;

component adder_32
    port ( clk_in : in  std_logic;
          x0      : in  std_logic_vector (31 downto 0);
          x1      : in  std_logic_vector (31 downto 0);
          x2      : in  std_logic_vector (31 downto 0);
          x3      : in  std_logic_vector (31 downto 0);
          x4      : in  std_logic_vector (31 downto 0);
          x5      : in  std_logic_vector (31 downto 0);
          x6      : in  std_logic_vector (31 downto 0);
          x7      : in  std_logic_vector (31 downto 0);
          x8      : in  std_logic_vector (31 downto 0);

```

```

        genc_result : out std_logic_vector (33 downto 0));
end component;
    attribute box_type of multiplier : component is "black_box";

```

```
begin
```

```

XLXI_1 : topmodule
    port map (clk_in=>clk_in,
        m0(15 downto 0)=>XLXN_23(15 downto 0),
        m1(15 downto 0)=>XLXN_22(15 downto 0),
        m2(15 downto 0)=>XLXN_21(15 downto 0),
        m3(15 downto 0)=>XLXN_20(15 downto 0),
        m4(15 downto 0)=>XLXN_19(15 downto 0),
        m5(15 downto 0)=>XLXN_17(15 downto 0),
        m6(15 downto 0)=>XLXN_16(15 downto 0),
        m7(15 downto 0)=>XLXN_15(15 downto 0),
        m8(15 downto 0)=>XLXN_14(15 downto 0),
        p0(15 downto 0)=>XLXN_3(15 downto 0),
        p1(15 downto 0)=>XLXN_6(15 downto 0),
        p2(15 downto 0)=>XLXN_7(15 downto 0),
        p3(15 downto 0)=>XLXN_8(15 downto 0),
        p4(15 downto 0)=>XLXN_9(15 downto 0),
        p5(15 downto 0)=>XLXN_10(15 downto 0),
        p6(15 downto 0)=>XLXN_11(15 downto 0),
        p7(15 downto 0)=>XLXN_12(15 downto 0),
        p8(15 downto 0)=>XLXN_13(15 downto 0));

```

```

XLXI_2 : multiplier
    port map (a(15 downto 0)=>XLXN_3(15 downto 0),
        b(15 downto 0)=>XLXN_14(15 downto 0),
        ce=>ce,
        clk=>clk_in,
        p(31 downto 0)=>XLXN_39(31 downto 0));

```

```

XLXI_3 : multiplier
    port map (a(15 downto 0)=>XLXN_6(15 downto 0),
        b(15 downto 0)=>XLXN_15(15 downto 0),
        ce=>ce,
        clk=>clk_in,
        p(31 downto 0)=>XLXN_40(31 downto 0));

```

```

XLXI_4 : multiplier
    port map (a(15 downto 0)=>XLXN_7(15 downto 0),
        b(15 downto 0)=>XLXN_16(15 downto 0),
        ce=>ce,
        clk=>clk_in,
        p(31 downto 0)=>XLXN_41(31 downto 0));

```

```

XLXI_5 : multiplier
    port map (a(15 downto 0)=>XLXN_8(15 downto 0),
        b(15 downto 0)=>XLXN_17(15 downto 0),
        ce=>ce,
        clk=>clk_in,
        p(31 downto 0)=>XLXN_42(31 downto 0));

```

```

XLXI_6 : multiplier
    port map (a(15 downto 0)=>XLXN_9(15 downto 0),

```

```

b(15 downto 0)=>XLXN_19(15 downto 0),
ce=>ce,
clk=>clk_in,
p(31 downto 0)=>XLXN_43(31 downto 0));

```

XLXI\_7 : multiplier

```

port map (a(15 downto 0)=>XLXN_10(15 downto 0),
b(15 downto 0)=>XLXN_20(15 downto 0),
ce=>ce,
clk=>clk_in,
p(31 downto 0)=>XLXN_44(31 downto 0));

```

XLXI\_8 : multiplier

```

port map (a(15 downto 0)=>XLXN_11(15 downto 0),
b(15 downto 0)=>XLXN_21(15 downto 0),
ce=>ce,
clk=>clk_in,
p(31 downto 0)=>XLXN_45(31 downto 0));

```

XLXI\_9 : multiplier

```

port map (a(15 downto 0)=>XLXN_12(15 downto 0),
b(15 downto 0)=>XLXN_22(15 downto 0),
ce=>ce,
clk=>clk_in,
p(31 downto 0)=>XLXN_46(31 downto 0));

```

XLXI\_10 : multiplier

```

port map (a(15 downto 0)=>XLXN_13(15 downto 0),
b(15 downto 0)=>XLXN_23(15 downto 0),
ce=>ce,
clk=>clk_in,
p(31 downto 0)=>XLXN_47(31 downto 0));

```

XLXI\_11 : adder\_32

```

port map (clk_in=>clk_in,
x0(31 downto 0)=>XLXN_39(31 downto 0),
x1(31 downto 0)=>XLXN_40(31 downto 0),
x2(31 downto 0)=>XLXN_41(31 downto 0),
x3(31 downto 0)=>XLXN_42(31 downto 0),
x4(31 downto 0)=>XLXN_43(31 downto 0),
x5(31 downto 0)=>XLXN_44(31 downto 0),
x6(31 downto 0)=>XLXN_45(31 downto 0),
x7(31 downto 0)=>XLXN_46(31 downto 0),
x8(31 downto 0)=>XLXN_47(31 downto 0),
genc_result(33 downto 0)=>genc_result(33 downto 0));

```

end BEHAVIORAL;

## Separable Convolution Method 1 VHDL Source Files

```
-----

-- filename: customfifo1.vhd

-- author: Arjun Joginipelly

-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity customfifo1 is
    Port ( clk_in : in  STD_LOGIC;
          dout : in  STD_LOGIC_VECTOR (15 downto 0);
          i0:out std_logic_vector(15 downto 0);
              i1:out std_logic_vector(15 downto 0);
              i2:out std_logic_vector(15 downto 0);
              vm0:out std_logic_vector (15 downto 0);
              vm1:out std_logic_vector (15 downto 0);
              vm2:out std_logic_vector (15 downto 0));

end customfifo1;

architecture Behavioral of customfifo1 is
    signal i0_sig:std_logic_vector(15 downto 0):="0000000000000000";
    signal i1_sig:std_logic_vector(15 downto 0):="0000000000000000";
    signal i2_sig:std_logic_vector(15 downto 0):="0000000000000000";

begin
    process(clk_in)
    begin
        if(clk_in'event and clk_in='0') then
            i0_sig<=i1_sig;
            i1_sig<=i2_sig;
            i2_sig<=dout;
        end if;
    end process;

    i0<=i0_sig;
    i1<=i1_sig;
    i2<=i2_sig;
    vm0<=x"0064";
    vm1<=x"003D";
    vm2<=x"000E";

end Behavioral;
```



-----  
-- filename: image\_controller.vhd

-- author: Arjun Joginipelly  
-----

entity image\_controller is

    Port ( clk\_in : in STD\_LOGIC;  
          douta : in STD\_LOGIC\_VECTOR (15 downto 0);  
          clk\_out : out STD\_LOGIC;  
          dina : out STD\_LOGIC\_VECTOR (15 downto 0);  
          wea : out STD\_LOGIC;  
          ena : out STD\_LOGIC;  
          addra : out STD\_LOGIC\_VECTOR (7 downto 0);  
          dout : out STD\_LOGIC\_VECTOR (15 downto 0));

end image\_controller;

architecture Behavioral of image\_controller is

    signal addra\_sig:std\_logic\_vector(7 downto 0):="00000000";  
    signal account:std\_logic\_vector(7 downto 0):="00000000";  
    signal rcount:std\_logic\_vector(7 downto 0):="00000000";  
    signal ccount:std\_logic\_vector(7 downto 0):="00000000";

    type state\_reg\_type is (initialstate,rd\_state,halt);

    signal sreg:state\_reg\_type:=initialstate;

begin

    clk\_out<=clk\_in;

    process(clk\_in)

    begin

        if(clk\_in'event and clk\_in<='0') then

            case sreg is

                when initialstate=>  wea<='0';

                                    ena<='1';

                                    ccount<="00000001";

                                    sreg<=rd\_state;

                when rd\_state=>  addra\_sig<=account;

                                    rcount<=rcount+1;

                                    account<=account+7;

                                    if(rcount=6) then

  rcount<="00000000";

  ccount<=ccount+1;

  account<=ccount;

  if(ccount=7) then

  sreg<=halt;

  else

  end if;

                                    else

  sreg<=rd\_state;

                                    end if;

    sreg<=rd\_state;

```

        when halt=> wea<='0';
            ena<='0';

        end case;
    end if;
end process;
addra<=addra_sig;
dout<=douta;

end Behavioral;

-----

-- filename: topmodule3.vhd
-- author: Arjun Joginipelly

-----

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;

entity topmodule3 is
    port ( clk_in    : in  std_logic;
          finalresult : out std_logic_vector (69 downto 0));
end topmodule3;

architecture BEHAVIORAL of topmodule3 is
    signal XLXN_3    : std_logic_vector (33 downto 0);
    signal XLXN_5    : std_logic_vector (67 downto 0);
    signal XLXN_6    : std_logic_vector (67 downto 0);
    signal XLXN_7    : std_logic_vector (67 downto 0);
    component topmodule2
        port ( clk_in : in  std_logic;
              douta  : out std_logic_vector (33 downto 0));
    end component;

    component customfifo2
        port ( clk_in : in  std_logic;
              douta  : in  std_logic_vector (33 downto 0);
              f0     : out std_logic_vector (67 downto 0);
              f1     : out std_logic_vector (67 downto 0);
              f2     : out std_logic_vector (67 downto 0));
    end component;

    component adder_68
        port ( clk_in    : in  std_logic;
              f0         : in  std_logic_vector (67 downto 0);
              f1         : in  std_logic_vector (67 downto 0);
              f2         : in  std_logic_vector (67 downto 0);

```

```

        finalresult : out std_logic_vector (69 downto 0));
end component;

begin
    XLXI_1 : topmodule2
        port map (clk_in=>clk_in,
            douta(33 downto 0)=>XLXN_3(33 downto 0));

    XLXI_2 : customfifo2
        port map (clk_in=>clk_in,
            douta(33 downto 0)=>XLXN_3(33 downto 0),
            f0(67 downto 0)=>XLXN_5(67 downto 0),
            f1(67 downto 0)=>XLXN_6(67 downto 0),
            f2(67 downto 0)=>XLXN_7(67 downto 0));

    XLXI_3 : adder_68
        port map (clk_in=>clk_in,
            f0(67 downto 0)=>XLXN_5(67 downto 0),
            f1(67 downto 0)=>XLXN_6(67 downto 0),
            f2(67 downto 0)=>XLXN_7(67 downto 0),
            finalresult(69 downto 0)=>finalresult(69 downto 0));

end BEHAVIORAL;

```

## Separable Convolution Method 2 VHDL Source Files

```

-----
-- filename: finalmodule.vhd
-- author: Arjun Joginipelly
-----

```

```

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;

entity finalmodule is
    port ( ce      : in  std_logic;
          clk_in   : in  std_logic;
          hc_result : out std_logic_vector (37 downto 0));
end finalmodule;

architecture BEHAVIORAL of finalmodule is
    signal XLXN_4  : std_logic_vector (17 downto 0);
    signal XLXN_5  : std_logic_vector (17 downto 0);
    signal XLXN_6  : std_logic_vector (35 downto 0);
    signal XLXN_7  : std_logic_vector (17 downto 0);
    signal XLXN_8  : std_logic_vector (17 downto 0);
    signal XLXN_9  : std_logic_vector (17 downto 0);
    signal XLXN_10 : std_logic_vector (17 downto 0);
    signal XLXN_11 : std_logic_vector (17 downto 0);

```

```

signal XLXN_12 : std_logic_vector (17 downto 0);
signal XLXN_13 : std_logic_vector (17 downto 0);
signal XLXN_14 : std_logic_vector (17 downto 0);
signal XLXN_15 : std_logic_vector (17 downto 0);
signal XLXN_16 : std_logic_vector (17 downto 0);
signal XLXN_17 : std_logic_vector (17 downto 0);
signal XLXN_18 : std_logic_vector (17 downto 0);
signal XLXN_19 : std_logic_vector (17 downto 0);
signal XLXN_20 : std_logic_vector (17 downto 0);
signal XLXN_21 : std_logic_vector (17 downto 0);
signal XLXN_22 : std_logic_vector (17 downto 0);
signal XLXN_23 : std_logic_vector (35 downto 0);
signal XLXN_24 : std_logic_vector (35 downto 0);
signal XLXN_25 : std_logic_vector (35 downto 0);
signal XLXN_26 : std_logic_vector (35 downto 0);
signal XLXN_27 : std_logic_vector (35 downto 0);
signal XLXN_28 : std_logic_vector (35 downto 0);
signal XLXN_29 : std_logic_vector (35 downto 0);
signal XLXN_30 : std_logic_vector (35 downto 0);

```

component topmodule2

```

port ( clk_in : in  std_logic;
      ce      : in  std_logic;
      hm8     : out std_logic_vector (17 downto 0);
      hm7     : out std_logic_vector (17 downto 0);
      hm6     : out std_logic_vector (17 downto 0);
      hm5     : out std_logic_vector (17 downto 0);
      hm4     : out std_logic_vector (17 downto 0);
      hm3     : out std_logic_vector (17 downto 0);
      hm2     : out std_logic_vector (17 downto 0);
      hm1     : out std_logic_vector (17 downto 0);
      hm0     : out std_logic_vector (17 downto 0);
      i08     : out std_logic_vector (17 downto 0);
      i07     : out std_logic_vector (17 downto 0);
      i06     : out std_logic_vector (17 downto 0);
      i05     : out std_logic_vector (17 downto 0);
      i04     : out std_logic_vector (17 downto 0);
      i03     : out std_logic_vector (17 downto 0);
      i02     : out std_logic_vector (17 downto 0);
      i01     : out std_logic_vector (17 downto 0);
      i00     : out std_logic_vector (17 downto 0));

```

end component;

component adder\_36

```

port ( clk_in  : in  std_logic;
      m00      : in  std_logic_vector (35 downto 0);
      m01      : in  std_logic_vector (35 downto 0);
      m02      : in  std_logic_vector (35 downto 0);
      m03      : in  std_logic_vector (35 downto 0);
      m04      : in  std_logic_vector (35 downto 0);
      m05      : in  std_logic_vector (35 downto 0);
      m06      : in  std_logic_vector (35 downto 0);
      m07      : in  std_logic_vector (35 downto 0);
      m08      : in  std_logic_vector (35 downto 0);
      hc_result : out std_logic_vector (37 downto 0));

```

end component;

```

component smultipliers
  port ( a : in  std_logic_vector (17 downto 0);
        b : in  std_logic_vector (17 downto 0);
        clk : in  std_logic;
        ce : in  std_logic;
        p : out std_logic_vector (35 downto 0));
end component;

begin
  XLXI_1 : topmodule2
    port map (ce=>ce,
              clk_in=>clk_in,
              hm0(17 downto 0)=>XLXN_22(17 downto 0),
              hm1(17 downto 0)=>XLXN_20(17 downto 0),
              hm2(17 downto 0)=>XLXN_18(17 downto 0),
              hm3(17 downto 0)=>XLXN_16(17 downto 0),
              hm4(17 downto 0)=>XLXN_14(17 downto 0),
              hm5(17 downto 0)=>XLXN_12(17 downto 0),
              hm6(17 downto 0)=>XLXN_8(17 downto 0),
              hm7(17 downto 0)=>XLXN_7(17 downto 0),
              hm8(17 downto 0)=>XLXN_5(17 downto 0),
              i00(17 downto 0)=>XLXN_4(17 downto 0),
              i01(17 downto 0)=>XLXN_9(17 downto 0),
              i02(17 downto 0)=>XLXN_10(17 downto 0),
              i03(17 downto 0)=>XLXN_11(17 downto 0),
              i04(17 downto 0)=>XLXN_13(17 downto 0),
              i05(17 downto 0)=>XLXN_15(17 downto 0),
              i06(17 downto 0)=>XLXN_17(17 downto 0),
              i07(17 downto 0)=>XLXN_19(17 downto 0),
              i08(17 downto 0)=>XLXN_21(17 downto 0));

  XLXI_2 : adder_36
    port map (clk_in=>clk_in,
              m00(35 downto 0)=>XLXN_6(35 downto 0),
              m01(35 downto 0)=>XLXN_23(35 downto 0),
              m02(35 downto 0)=>XLXN_24(35 downto 0),
              m03(35 downto 0)=>XLXN_25(35 downto 0),
              m04(35 downto 0)=>XLXN_26(35 downto 0),
              m05(35 downto 0)=>XLXN_27(35 downto 0),
              m06(35 downto 0)=>XLXN_28(35 downto 0),
              m07(35 downto 0)=>XLXN_29(35 downto 0),
              m08(35 downto 0)=>XLXN_30(35 downto 0),
              hc_result(37 downto 0)=>hc_result(37 downto 0));

  XLXI_3 : smultipliers
    port map (a(17 downto 0)=>XLXN_4(17 downto 0),
              b(17 downto 0)=>XLXN_5(17 downto 0),
              ce=>ce,
              clk=>clk_in,
              p(35 downto 0)=>XLXN_6(35 downto 0));

  XLXI_4 : smultipliers
    port map (a(17 downto 0)=>XLXN_9(17 downto 0),
              b(17 downto 0)=>XLXN_7(17 downto 0),
              ce=>ce,
              clk=>clk_in,

```

```

        p(35 downto 0)=>XLXN_23(35 downto 0));

    XLXI_5 : multipliers
    port map (a(17 downto 0)=>XLXN_10(17 downto 0),
        b(17 downto 0)=>XLXN_8(17 downto 0),
        ce=>ce,
        clk=>clk_in,
        p(35 downto 0)=>XLXN_24(35 downto 0));

    XLXI_6 : multipliers
    port map (a(17 downto 0)=>XLXN_11(17 downto 0),
        b(17 downto 0)=>XLXN_12(17 downto 0),
        ce=>ce,
        clk=>clk_in,
        p(35 downto 0)=>XLXN_25(35 downto 0));

    XLXI_7 : multipliers
    port map (a(17 downto 0)=>XLXN_13(17 downto 0),
        b(17 downto 0)=>XLXN_14(17 downto 0),
        ce=>ce,
        clk=>clk_in,
        p(35 downto 0)=>XLXN_26(35 downto 0));

    XLXI_8 : multipliers
    port map (a(17 downto 0)=>XLXN_15(17 downto 0),
        b(17 downto 0)=>XLXN_16(17 downto 0),
        ce=>ce,
        clk=>clk_in,
        p(35 downto 0)=>XLXN_27(35 downto 0));

    XLXI_9 : multipliers
    port map (a(17 downto 0)=>XLXN_17(17 downto 0),
        b(17 downto 0)=>XLXN_18(17 downto 0),
        ce=>ce,
        clk=>clk_in,
        p(35 downto 0)=>XLXN_28(35 downto 0));

    XLXI_10 : multipliers
    port map (a(17 downto 0)=>XLXN_19(17 downto 0),
        b(17 downto 0)=>XLXN_20(17 downto 0),
        ce=>ce,
        clk=>clk_in,
        p(35 downto 0)=>XLXN_29(35 downto 0));

    XLXI_11 : multipliers
    port map (a(17 downto 0)=>XLXN_21(17 downto 0),
        b(17 downto 0)=>XLXN_22(17 downto 0),
        ce=>ce,
        clk=>clk_in,
        p(35 downto 0)=>XLXN_30(35 downto 0));

end BEHAVIORAL;

```

## Steerable Implementation VHDL Source Files

```
-----
-- filename: hcr_wr_vert_rd_horz_controller.vhd
-- author: Arjun Joginipelly
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity hcr_wr_vert_rd_horz_controller is
  Port ( clk_in : in  STD_LOGIC;
        hc_result : in  STD_LOGIC_VECTOR (37 downto 0);
        addra : out  STD_LOGIC_VECTOR (11 downto 0);
        ena : out  STD_LOGIC;
        dina : out  STD_LOGIC_VECTOR (37 downto 0);
        clk_out : out  STD_LOGIC;
        wea : out  STD_LOGIC);
end hcr_wr_vert_rd_horz_controller;

architecture Behavioral of hcr_wr_vert_rd_horz_controller is

  type state_reg_type is (vwr_initialstate,vwr_dummystate,vwr_state,
    hrd_initialstate,hrd_state,halt);
  signal sreg:state_reg_type:=vwr_initialstate;

  signal dummycount:std_logic_vector(8 downto 0):="000000000";
  signal acount:std_logic_vector(11 downto 0):=(others=>'0');
  signal ccount:std_logic_vector(11 downto 0):=(others=>'0');
  signal rcount:std_logic_vector(11 downto 0):=(others=>'0');
  signal addra_sig:std_logic_vector(11 downto 0):=(others=>'0');

  --signal dacount:std_logic_vector(11 downto 0):=(others=>'0');

begin
  clk_out<=clk_in;
  process(clk_in)
  begin
    if(clk_in'event and clk_in='1') then
      case sreg is
```

```

when vwr_initialstate=> wea<='1';
    ena<='1';
    ccount<="000000000001";

sreg<=vwr_dummystate;

    when vwr_dummystate=> dummycount<=dummycount+1;

if(dummycount=445) then
    dummycount<="000000000";
    sreg<=vwr_state;
else
    sreg<=vwr_dummystate;
end if;

    when vwr_state=> addra_sig<=account;

    dina<=hc_result;
    rcount<=rcount+1;
    account<=account+48;
    if(rcount=47) then

rcount<="000000000000";
ccount<=ccount+1;
    account<=ccount;
    if(ccount=48)
then
    sreg<=hrd_initialstate;
else
    sreg<=vwr_state;
end if;
else
    sreg<=vwr_state;
end if;

    when hrd_initialstate=> wea<='0';

    ena<='1';
    account<="000000000000";
    addra_sig<="000000000000";
    ccount<="000000000000";
    rcount<="000000000001";
    sreg<=hrd_state;

    when hrd_state=> addra_sig<=account;

```



```

account<=account+3;
if(account=2304) then
    sreg<=halt;
else
    sreg<=hrd_state;
end if;

sreg<=hrd_state;

when halt=> wea<='0';
                                ena<='0';

                                end case;
                                end if;
                                end process;
addra<=addra_sig;

end Behavioral;

-----

-- filename: diagonal_rd_controller.vhd
-- author: Arjun Joginipelly

-----

entity diagonal_rd_controller is
    Port ( clk_in : in  STD_LOGIC;
          douta : in  STD_LOGIC_vector(7 downto 0);
          clk_out: out std_logic;
          wea: out std_logic;
          ena: out std_logic;
          addra: out std_logic_vector(7 downto 0);
          dina: out std_logic_vector( 7 downto 0);
          dout: out std_logic_vector ( 7 downto 0) );
end diagonal_rd_controller;

architecture Behavioral of diagonal_rd_controller is

type state_reg_type is (initialstate,diagonalrldstate,halt);
signal sreg:state_reg_type:=initialstate;

signal account:std_logic_vector( 7 downto 0):="00000000";
signal dacount:std_logic_vector(7 downto 0):="00000000";---dummycount;
signal addra_sig:std_logic_vector(7 downto 0):="00000000";

begin
clk_out<=clk_in;
    process(clk_in)

```

```

begin
    if(clk_in'event and clk_in='0') then

        case sreg is

            when initialstate=> wea<='0';
                                                                    ena<='1';

            account<="00000000";
                                                                    dacount<="00000000";

            sreg<=diagonalrdstate;

            when diagonalrdstate=> addra_sig<=dacount;

            dacount<=dacount+98;

            if(dacount>=1960) then

            if(account<39) then

                dacount<=account+1;

                account<=account+1;

            end if;

            if(account=39) then

                dacount<=account+9;

                account<=account+9;

                end if;

                if(account>39) then

                    dacount<=account+48;

                    account<=account+48;

                    end if;

                    if(account>1920) then

                        sreg<=halt;

                    else

                        sreg<=diagonalrdstate;

                    end if;

```

```

else

sreg<=diagonalrdstate;

end if;

when halt=> wea<='0';
ena<='0';

end case;

end if;

end process;
addra<=addra_sig;
dout<=douta;

end Behavioral;

```

## Matlab M Files

%project7 folder results verification with matlab

```

clc,clear all;
image=[1 2 3 4 5;6 7 8 9 10;11 12 13 14 15;16 17 18 19 20;21 22 23 24 25];
gaussianmask=[100 61 14; 61 37 8 ; 14 8 2];
gc=1/628;
c=conv2(image,gaussianmask,'full');% required result to verify with vhdl simulation

```

%project8 folder results verification with matlab

```

clc,clear all;
image=[1 2 3 4 5 0 0;6 7 8 9 10 0 0;11 12 13 14 15 0 0;16 17 18 19 20 0 0;21
22 23 24 25 0 0;0 0 0 0 0 0 0;0 0 0 0 0 0 0];
vgmask = [100; 61; 14];
constant=1/628;
vc=conv2(image,vgmask,'full'); %%%%%%%%% required vertical convolution result
to verify with vhdl simulation

image1=[1480,1655,1830,2005,2180,0,0;2355,2530,2705,2880,3055,0,0;3230,3405,3
580,3755,3930,0,0;1505,1580,1655,1730,1805,0,0;294,308,322,336,350,0,0];
hgmask=[100 61 14];
hc=conv2(vc,hgmask,'full'); %%% required horizontal convolution result to
verify with vhdl simulation

hc1=conv2(image1,hgmask,'full');
resultinhex=dec2hex(hc1',70);
resultinbin=dec2bin(hc1',70);

```

```
%In this project you generate a coe file for an image of 40*40 inorder to
%load in to Block RAM of FPGA. Also final results are obtained for
verification
```

```
clear all;
clc;
I1 = imread('8.jpg');
I1 = rgb2gray(I1);
I2 = zeros(40,8);
I3 = zeros(8,48);
hc = horzcat(I1,I2);
vc = vertcat(hc,I3);
final = vc;

[rows,coloumns]= size(final);

fp = fopen('project12file.coe.txt','w');
for i=1:rows
    for j=1:coloumns
        fprintf(fp,'%s, ',dec2hex(final(i,j),8));
    end
end
fclose(fp);

vgmask=[100;61;14;1;0;0;0;0;0];
hgmask=[100 61 14 1 0 0 0 0 0];

vcr=conv2(final,vgmask,'full');
hcr=conv2(vcr,hgmask,'full');

hcinhex=dec2hex(hcr,38);
% steerable convolution results in horizontal,vertical,diagonal and
% reversediagonal directions.

hdmask=[8 33 76 100 76 33 8];
vdmask=[8;33;76;100;76;33;8];

hsteer_result=conv2(hcr,hdmask,'full');
vsteer_result=conv2(hcr,vdmask,'full');
dsteer_result=conv2(hcr,vdmask,'full');
rdsteer_result=conv2(hcr,vdmask,'full');
```

## “.coe” File Format

```
memory_initialization_radix=16;
memory_initialization_vector=00000002, 00000001, 00000001, 00000001, 00000001, 00000001, 00000001,
00000001, 00000001, 00000001, 00000001, 00000001, 00000001, 00000002, 00000002, 00000001, 00000001,
00000002, 00000003, 00000003, 00000002, 00000002, 00000002, 00000001, 00000001, 00000001, 00000001,
00000001, 00000000, 00000000, 00000001, 00000001, 00000001, 00000002, 00000000, 00000000, 00000000,
00000000, 00000001, 00000002, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,
00000000, 00000001, 00000001, 00000001, 00000001, 00000002, 00000004, 00000003, 00000003, 00000004,
00000005, 00000004, 00000004, 00000003, 00000003, 00000004, 00000005, 00000003, 00000003, 00000003,
00000003, 00000003, 00000003, 00000002, 00000002, 00000005, 00000004, 00000004, 00000003, 00000002,
00000002, 00000002, 00000002, 00000003, 00000005, 00000002, 00000000, 00000003, 00000001, 00000001,
00000001, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000001,
00000001, 00000005, 00000009, 0000000C, 0000000D, 0000000D, 0000000D, 0000000F, 0000000E, 0000000E,
0000000E, 0000000D, 0000000B, 0000000D, 0000000F, 0000000F, 0000000F, 0000000E, 0000000E, 0000000D,
0000000D, 0000000D, 0000000D, 0000000C, 0000000C, 0000000C, 0000000D, 0000000D, 0000000D,
0000000D, 0000000D, 0000000A, 0000000C, 0000000A, 00000008, 0000000A, 00000006, 00000001, 00000001,
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000001, 00000004,
0000000A, 00000011, 00000015, 00000017, 00000018, 00000017, 00000017, 00000018, 0000001A, 0000001D,
0000001D, 0000001C, 0000001B, 0000001C, 0000001B, 0000001B, 0000001B, 0000001B, 0000001B, 0000001B,
0000001A, 00000019, 00000017, 00000017, 00000018, 00000019, 00000019, 00000019, 00000019, 00000018,
00000013, 00000016, 00000013, 00000012, 00000013, 0000000C, 00000003, 00000001, 00000000, 00000000,
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000001, 00000004, 0000000B, 00000013,
00000019, 0000001C, 0000001D, 0000001D, 00000021, 00000023, 00000027, 0000002C, 00000030, 00000030,
0000002B, 00000026, 00000021, 00000022, 00000024, 00000025, 00000025, 00000024, 00000022, 00000021,
00000022, 00000021, 00000021, 00000020, 0000001F, 0000001D, 0000001C, 0000001C, 0000001C, 0000001D,
0000001A, 00000018, 00000019, 00000011, 00000005, 00000001, 00000000, 00000000, 00000000, 00000000,
00000000, 00000000, 00000000, 00000000, 00000001, 00000004, 0000000B, 00000014, 0000001B, 00000020,
00000023, 00000023, 0000002C, 00000030, 00000034, 00000039, 00000043, 00000048, 00000040, 00000032,
0000002D, 0000002E, 00000030, 00000031, 00000031, 0000002F, 0000002D, 0000002C, 00000027, 00000027,
00000026, 00000025, 00000025, 00000025, 00000025, 00000025, 00000022, 00000022, 0000001D, 0000001B,
0000001B, 00000012, 00000005, 00000001, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,
00000000, 00000000, 00000001, 00000005, 0000000D, 00000016, 0000001E, 00000025, 0000002A, 0000002C,
00000031, 00000036, 00000038, 0000003E, 0000004F, 0000005C, 00000052, 0000003E, 0000003B, 0000003B,
0000003A, 0000003A, 00000039, 00000038, 00000037, 00000036, 00000030, 0000002F, 0000002D, 0000002B,
0000002A, 0000002B, 0000002B, 0000002C, 00000028, 00000026, 0000001F, 0000001C, 0000001C, 00000014,
00000005, 00000002, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,
00000001, 00000004, 0000000C, 00000015, 0000001F, 00000027, 0000002D, 00000030, 0000002C, 00000032,
00000034, 0000003A, 00000050, 00000064, 0000005B, 00000043, 0000003F, 0000003E, 0000003B, 00000039,
00000038, 00000037, 00000037, 00000037, 0000003B, 00000039, 00000034, 0000002F, 0000002B, 00000028,
00000027, 00000027, 0000002C, 00000029, 00000021, 0000001D, 0000001E, 00000015, 00000008, 00000002,
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000001, 00000007,
00000010, 00000018, 00000021, 0000002B, 00000033, 00000034, 00000030, 00000031, 00000035, 0000003F,
0000004D, 00000056, 00000052, 00000048, 00000041, 0000003A, 00000039, 0000003C, 0000003B, 0000003B,
0000003B, 00000039, 00000039, 0000003B, 0000003E, 0000003C, 00000035, 0000002E, 0000002D, 00000031,
0000002C, 0000002F, 0000002A, 00000022, 0000001E, 00000014, 00000006, 00000002, 00000000, 00000000,
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000001, 00000006, 0000000F, 00000019,
00000024, 0000002E, 00000032, 00000031, 0000002F, 0000003A, 00000045, 00000049, 00000049, 0000004A,
0000004B, 0000004B, 0000004A, 00000044, 00000040, 0000003F, 00000040, 0000003F, 0000003C, 0000003A,
00000040, 0000003B, 00000039, 0000003C, 00000041, 00000041, 0000003A, 00000033, 00000030, 00000033,
0000002E, 00000025, 00000020, 00000015, 00000008, 00000002, 00000000, 00000000, 00000000, 00000000,
00000000, 00000000, 00000000, 00000000, 00000000, 00000005, 0000000F, 0000001A, 00000027, 00000030,
00000031, 0000002F, 00000034, 00000041, 0000004C, 0000004B, 00000044, 00000042, 00000047, 0000004D,
00000050, 0000004E, 00000045, 00000041, 00000044, 00000044, 00000042, 00000043, 0000003F, 0000003F,
0000003E, 0000003E, 00000040, 00000043, 00000042, 0000003E, 00000033, 00000037, 00000034, 0000002C,
```

00000024, 00000016, 00000007, 00000002, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,  
00000000, 00000000, 00000000, 00000006, 00000010, 0000001C, 00000027, 0000002F, 00000033, 00000033,  
00000043, 00000046, 00000048, 00000047, 00000046, 00000047, 0000004C, 0000004F, 00000057, 0000005A,  
00000052, 0000004A, 0000004B, 0000004B, 0000004A, 0000004C, 00000044, 00000046, 00000046, 00000041,  
0000003E, 00000040, 00000046, 0000004B, 00000037, 0000003B, 00000038, 00000030, 00000028, 00000017,  
00000007, 00000002, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,  
00000001, 00000007, 00000012, 0000001E, 00000027, 0000002F, 00000036, 0000003C, 0000004D, 0000004A,  
00000049, 00000049, 0000004C, 0000004E, 0000004F, 00000050, 0000005C, 00000062, 00000063, 0000005D,  
00000059, 00000058, 00000054, 00000050, 00000050, 0000004C, 00000048, 00000046, 00000047, 00000049,  
0000004D, 00000051, 0000004E, 0000004C, 00000041, 00000034, 00000028, 00000017, 00000008, 00000004,  
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000001, 00000007,  
00000013, 00000020, 00000029, 00000031, 0000003C, 00000045, 00000049, 0000004B, 0000004E, 00000050,  
00000050, 00000050, 00000053, 00000058, 0000005C, 00000065, 00000071, 00000074, 00000072, 00000076,  
00000074, 00000067, 0000005B, 00000054, 0000004F, 0000004D, 0000004B, 00000049, 0000004F, 00000058,  
00000073, 00000069, 00000052, 0000003B, 00000029, 00000015, 00000006, 00000003, 00000000, 00000000,  
00000000, 00000000, 00000000, 00000000, 00000000, 00000001, 00000006, 00000012, 00000021,  
0000002D, 00000036, 00000041, 0000004A, 00000046, 0000004A, 00000050, 00000056, 00000058, 0000005C,  
00000067, 00000072, 00000077, 00000078, 00000089, 00000096, 0000009B, 000000AD, 000000B2, 0000009F,  
0000007F, 0000006C, 0000005B, 00000055, 0000004F, 00000049, 00000052, 00000061, 0000007A, 0000006C,  
00000052, 0000003C, 0000002A, 00000015, 00000006, 00000002, 00000000, 00000000, 00000000, 00000000,  
00000000, 00000000, 00000000, 00000000, 00000001, 00000005, 00000011, 00000022, 00000031, 0000003B,  
00000044, 0000004C, 0000004B, 0000004C, 00000050, 00000058, 00000062, 0000006F, 00000081, 00000090,  
000000A3, 0000009B, 000000A9, 000000B7, 000000C0, 000000DD, 000000EB, 000000D6, 000000B2,  
00000089, 00000063, 0000005A, 0000005C, 0000005A, 0000005C, 00000065, 0000005F, 00000054, 00000041,  
00000033, 00000029, 00000017, 00000007, 00000002, 00000000, 00000000, 00000000, 00000000, 00000000,  
00000000, 00000000, 00000000, 00000002, 00000004, 00000010, 00000029, 00000035, 0000003A, 0000004A,  
0000004D, 0000004C, 0000004D, 00000053, 00000060, 0000006C, 0000007A, 0000009D, 000000C4, 000000DF,  
000000DF, 000000B7, 000000A6, 000000BF, 000000E0, 000000F4, 000000F0, 000000D5, 000000AE, 0000007C,  
00000062, 00000064, 00000069, 00000060, 00000051, 0000004D, 0000004E, 00000043, 0000002D, 00000024,  
0000001B, 00000007, 00000003, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,  
00000000, 00000002, 00000007, 00000014, 0000002B, 00000037, 0000003E, 0000004D, 0000004F, 0000004E,  
0000004F, 00000054, 00000061, 00000072, 0000008C, 000000BB, 000000E8, 000000F2, 000000EF, 000000A2,  
0000007B, 000000A7, 000000E4, 000000E8, 000000DA, 000000C4, 000000AC, 0000008C, 00000076, 0000006C,  
00000065, 00000059, 0000004D, 0000004B, 0000004D, 00000044, 0000002E, 00000025, 0000001B, 00000007,  
00000004, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000001,  
00000009, 00000017, 0000002C, 00000038, 00000042, 00000050, 0000004F, 0000004E, 00000052, 00000058,  
00000066, 0000007A, 00000098, 000000CA, 000000EE, 000000F4, 000000D7, 00000079, 0000005E, 0000004D,  
00000080, 000000C0, 000000C5, 000000B5, 000000AA, 0000009A, 00000085, 0000006F, 0000005D, 00000051,  
0000004B, 00000049, 0000004D, 00000046, 00000030, 00000026, 0000001B, 00000007, 00000004, 00000000,  
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000002, 0000000A, 00000018,  
0000002C, 00000039, 00000044, 00000052, 0000004E, 00000050, 00000054, 00000059, 00000066, 0000007B,  
0000009A, 000000C8, 000000EB, 000000EB, 000000B1, 00000058, 00000062, 0000004A, 00000039, 00000082,  
000000AE, 000000BD, 000000B1, 0000009B, 00000081, 00000069, 00000057, 0000004F, 0000004E, 00000049,  
0000004D, 00000047, 00000031, 00000027, 0000001B, 00000007, 00000004, 00000000, 00000000, 00000000,  
00000000, 00000000, 00000000, 00000000, 00000000, 00000002, 0000000A, 00000019, 0000002E, 0000003B,  
00000046, 00000054, 00000050, 00000054, 00000057, 0000005A, 00000065, 0000007B, 00000098, 000000BE,  
000000DF, 000000C9, 00000083, 0000006C, 0000006D, 00000054, 0000003F, 00000074, 0000009E, 000000DA,  
000000BE, 00000098, 00000079, 00000065, 00000059, 00000051, 0000004F, 00000049, 0000004E, 00000048,  
00000032, 00000027, 0000001B, 00000006, 00000003, 00000000, 00000000, 00000000, 00000000, 00000000,  
00000000, 00000000, 00000000, 00000003, 0000000B, 00000019, 00000030, 0000003D, 00000047, 00000055,  
00000053, 00000051, 00000059, 00000060, 0000006C, 0000007F, 00000093, 000000AA, 000000C0, 000000AE,  
000000CF, 00000077, 0000004A, 0000005D, 00000052, 000000B8, 000000E4, 000000ED, 000000CB, 00000098,  
00000076, 00000066, 0000005C, 00000053, 0000004C, 0000004A, 0000004E, 00000047, 00000031, 00000027,  
0000001B, 00000006, 00000002, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,  
00000000, 00000003, 0000000A, 00000017, 00000030, 0000003E, 00000045, 00000054, 00000055, 00000052,  
0000005C, 00000063, 0000006C, 0000007A, 00000089, 00000099, 000000AA, 000000C1, 000000D5, 000000D3,

00000099, 0000009D, 000000C2, 000000DF, 000000F4, 000000F3, 000000CE, 00000096, 00000073, 00000064,  
0000005B, 00000051, 0000004B, 0000004A, 0000004D, 00000045, 0000002F, 00000026, 0000001C, 00000006,  
00000002, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000003,  
00000008, 00000014, 0000002F, 0000003D, 00000043, 00000052, 00000055, 0000005A, 00000060, 00000060,  
00000061, 0000006C, 00000080, 0000009A, 000000B1, 000000DA, 000000F1, 000000EB, 000000DB, 000000C1,  
000000C5, 000000EF, 000000F3, 000000F0, 000000C9, 00000092, 0000006E, 0000005E, 00000055, 0000004F,  
0000004D, 0000004A, 0000004C, 00000043, 0000002E, 00000026, 0000001C, 00000006, 00000002, 00000000,  
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000002, 00000007, 00000017,  
0000002E, 0000003D, 00000044, 0000004D, 00000058, 00000056, 0000005B, 0000005C, 0000005D, 00000072,  
00000097, 000000B6, 000000C3, 000000E7, 000000F4, 000000F9, 000000F0, 000000D8, 000000C3, 000000CD,  
000000E6, 000000D4, 000000A9, 0000007D, 00000068, 0000005E, 00000058, 0000004F, 00000046, 0000004C,  
00000046, 00000040, 0000002E, 00000022, 0000001A, 00000008, 00000002, 00000000, 00000000, 00000000,  
00000000, 00000000, 00000000, 00000000, 00000000, 00000002, 00000008, 00000018, 00000031, 00000041,  
00000047, 0000004F, 00000057, 00000051, 00000055, 00000059, 00000064, 00000081, 000000A8, 000000C4,  
000000CD, 000000E7, 000000EF, 000000F1, 000000E1, 000000CC, 000000BC, 000000B6, 000000B6,  
0000009D, 00000085, 00000070, 00000063, 00000055, 0000004C, 0000004C, 0000004E, 0000004C, 00000043,  
0000003C, 0000002D, 00000022, 00000019, 00000007, 00000002, 00000000, 00000000, 00000000, 00000000,  
00000000, 00000000, 00000000, 00000000, 00000001, 00000006, 00000017, 00000031, 00000043, 0000004C,  
00000052, 00000058, 0000004D, 0000004F, 00000053, 00000061, 00000084, 000000AC, 000000C4, 000000CA,  
000000D7, 000000DB, 000000D4, 000000C1, 000000B3, 000000AE, 0000009D, 00000083, 00000076, 00000068,  
00000060, 0000005D, 00000055, 0000004D, 0000004B, 0000004A, 0000004C, 00000040, 00000039, 0000002D,  
00000023, 00000018, 00000006, 00000001, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,  
00000000, 00000000, 00000001, 00000006, 00000014, 0000002A, 0000003C, 00000044, 00000048, 0000004C,  
00000053, 00000055, 00000056, 00000061, 0000007E, 000000A2, 000000BA, 000000C2, 000000C7, 000000C2,  
000000B8, 000000A5, 0000009C, 000000A2, 00000098, 0000007C, 0000006B, 0000005E, 00000058, 00000059,  
00000056, 00000051, 0000004D, 00000048, 00000049, 0000003C, 00000037, 0000002F, 00000025, 00000017,  
00000005, 00000001, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,  
00000003, 00000005, 0000000F, 00000020, 0000002E, 00000034, 00000036, 00000038, 0000004C, 00000052,  
00000058, 0000005E, 0000006F, 00000088, 000000A0, 000000AA, 000000A8, 0000009D, 00000092, 00000082,  
0000007A, 00000086, 0000008C, 00000082, 00000064, 0000005A, 00000055, 00000053, 0000004E, 0000004D,  
00000050, 00000050, 0000003F, 00000035, 00000034, 0000002E, 00000025, 00000017, 00000005, 00000002,  
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000002, 00000004,  
0000000D, 0000001B, 00000028, 0000002F, 00000032, 00000034, 0000003F, 0000004B, 00000055, 00000058,  
0000005D, 00000069, 00000076, 0000007E, 0000007A, 00000070, 00000066, 0000005D, 00000059, 00000064,  
00000077, 00000080, 0000006B, 0000005B, 0000004F, 0000004D, 0000004E, 00000050, 0000004F, 00000049,  
00000034, 0000002F, 00000031, 0000002B, 00000021, 00000016, 00000006, 00000003, 00000000, 00000000,  
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000001, 00000004, 0000000E, 0000001D,  
0000002A, 00000032, 00000035, 00000037, 0000003B, 00000047, 00000053, 00000056, 00000056, 00000059,  
0000005B, 0000005C, 00000058, 00000056, 00000052, 00000050, 00000052, 00000057, 00000063, 00000073,  
00000074, 00000062, 00000054, 00000052, 00000054, 00000053, 0000004A, 0000003C, 00000032, 00000031,  
00000033, 00000028, 0000001D, 00000014, 00000006, 00000002, 00000000, 00000000, 00000000, 00000000,  
00000000, 00000000, 00000000, 00000000, 00000001, 00000006, 00000011, 0000001E, 00000029, 0000002F,  
00000030, 00000031, 0000002F, 00000039, 00000044, 0000004A, 0000004D, 0000004F, 0000004E, 0000004B,  
00000049, 0000004D, 0000004A, 0000004B, 00000051, 0000004E, 0000004E, 00000058, 0000006D, 00000067,  
00000063, 0000005E, 00000054, 0000004C, 00000045, 0000003D, 00000036, 00000037, 00000038, 00000028,  
0000001A, 00000013, 00000005, 00000003, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,  
00000000, 00000000, 00000001, 00000007, 00000010, 0000001B, 00000024, 0000002A, 0000002E, 00000030,  
0000002F, 0000002F, 00000031, 00000038, 00000041, 00000047, 0000004A, 0000004A, 00000045, 00000045,  
00000045, 00000044, 00000043, 00000046, 0000004C, 00000050, 0000005E, 00000080, 0000008B, 0000006C,  
00000048, 00000041, 0000003F, 00000035, 00000038, 00000037, 00000030, 00000024, 00000019, 0000000F,  
00000006, 00000002, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,  
00000002, 00000006, 00000010, 0000001A, 00000023, 00000028, 0000002C, 0000002E, 00000030, 0000002F,  
0000002F, 00000032, 00000037, 0000003C, 0000003F, 0000003F, 00000046, 00000048, 0000004A, 0000004A,  
0000004A, 0000004B, 0000004F, 00000053, 0000005C, 0000007E, 0000008B, 00000072, 00000051, 00000046,  
00000042, 0000003A, 00000036, 00000034, 0000002E, 00000024, 0000001A, 00000012, 00000008, 00000002,  
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000002, 00000005,

[illegible]



[illegible]

## **Vita**

Arjun Kumar Joginipelly was born on April 4<sup>th</sup>, 1985, in Karimnagar, AP, India. The author completed his bachelor's degree in Electronics and Communication Engineering from Jawaharlal Nehru Technological University, Hyderabad, India in 2007 with distinction. He finished his Masters in Electrical Engineering from the University of New Orleans in December 2010 with a cumulative GPA of 3.8. He is continuing his studies at University of New Orleans to pursue PhD in Engineering and Applied Sciences. The author is working with his advisor Dr. Dimitrios Charalampidis as a Research Assistant. His research interests are in Digital Signal and Image Processing, Embedded Systems, FPGA and its Prototyping.