VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Machhe, Belgaum-590018, Karnataka.

Project Report on

An Efficient Radix-3 Multiplierless 2D Convolution Filter for Visual Search Applications

Submitted in partial fulfillment of the requirement for the award of the degree of

MASTER OF TECHNOLOGY
in
VLSI DESIGN AND EMBEDDED SYSTEMS
By

HARSHITHA A
USN: 1BG17LVS02

Under the Guidance of
Dr. Yasha Jyothi M. Shirur
Professor,
Dept. of ECE, BNMIT, Bengaluru

Vidyaya Amrutham Ashnuthe

BNM Institute of Technology

Approved by AICTE, Affiliated to VTU, Accredited as Grade A Institution by NAAC.
All UG branches – CSE, ECE, EEE, ISE & Mech.E Accredited by NBA for academic years 2018-19 to 2020-21 & valid upto 30.06.2021
Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage, Bengaluru- 560070, INDIA
Ph: 91-80- 26711780/81/82   Email: principal@bnmit.in, www. bnmit.org

Department of Electronics & Communication Engineering
2018 – 2019

BNM Institute of Technology

Approved by AICTE, Affiliated to VTU, Accredited as Grade A Institution by NAAC.
All UG branches – CSE, ECE, EEE, ISE & Mech.E Accredited by NBA for academic years 2018-19 to 2020-21 & valid upto 30.06.2021
Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage, Bengaluru- 560070, INDIA
Ph: 91-80- 26711780/81/82 Email: principal@bnmit.in, www. bnmit.org

Department of Electronics & Communication Engineering

Vidyaya Amrutham Ashnuthe

CERTIFICATE

This is to certify that the project work entitled **"An Efficient Radix-3 Multiplierless 2D Convolution Filter for Visual Search Applications"** has been successfully carried out by **Ms. HARSHITHA A** bearing the **USN**: **1BG17LVS02**, a bonafide student of BNM Institute of Technology, Bengaluru in partial fulfillment for the **Master of Technology in VLSI Design and Embedded Systems** of the **Visvesvaraya Technological University**, **Belagavi** during the year **2018-2019**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the project report deposited at the department library. The project report has been approved as it satisfies the academic requirements in respect to project work prescribed for the said Degree.

Dr. Yasha Jyothi M. Shirur	**Dr. P. A. Vijaya**	**Dr. Krishnamurthy G. N.**
Professor, Dept. of ECE	Professor and HOD	Principal
BNMIT, Bengaluru	Dept. of ECE, BNMIT, Bengaluru	BNMIT, Bengaluru

EXTERNAL VIVA VOCE

Name of the Examiners	**Signature with Date**
1. ____________________________	_______________________
2. ____________________________	_______________________

DECLARATION

I, the undersigned solemnly declare that the evaluation of Project report work entitled **"An Efficient Radix-3 Multiplierless 2D Convolution Filter for Visual Search Applications"** is based on my work carried out during the course study under the supervision of **Dr. Yasha Jyothi M. Shirur**, Professor, Department of ECE, B.N.M. Institute of Technology, Bengaluru.

I assert that the statements made and conclusions drawn are an outcome of the project work. I further declare that, to the best of my knowledge and belief, this report does not contain any work which has been submitted for the award of the degree or any other degree in this university or any other university.

Harshitha A

[1BG17LVS02]

ACKNOWLEDGEMENT

On successful completion of this project, I would like to place on record my sincere thanks and gratitude to the concerned people, whose suggestions and words of encouragement has been valuable.

I would like to express my heartfelt gratitude to **B.N.M. Institute of Technology**, for giving me the opportunity to pursue Degree in Master of Technology. I take this opportunity to thank **Prof. T. J. Rama Murthy**, Director, BNMIT, **Prof. Eishwar N. Mannay,** Dean, BNMIT and **Dr. Krishnamurthy. G. N.,** Principal, BNMIT for their support and encouragement to pursue this project.

I would like to thank **Dr. P. A. Vijaya,** Professor and Head, Dept. of Electronics and Communication Engineering, for her support and encouragement.

I would like to thank my guide and PG co-coordinator **Dr. Yasha Jyothi M. Shirur,** Professor, Dept. of Electronics and Communication Engineering**,** who has been the source of inspiration throughout my project work and has provided me guidance and valuable suggestions at every stage of my project work.

Finally, I am grateful to all the teaching and non-teaching staff of Department of Electronics and Communication for their help in the successful completion of my project. Last but not the least I would like to extend sincere gratitude to my parents and all my friends who were a constant support throughout my project work.

Harshitha A
[1BG17LVS02]

An Efficient Radix-3 Multiplierless 2D Convolution Filter for Visual Search Applications

Harshitha A
MTech (IV Sem)
Dept. of Electronics and Communication
BNMIT, Bangalore, INDIA
harshithaanand94@gmail.com

Dr. Yasha Jyothi M Shirur
Professor
Dept. of Electronics and Communication
BNMIT, Bangalore, INDIA
yashamallik@gmail.com

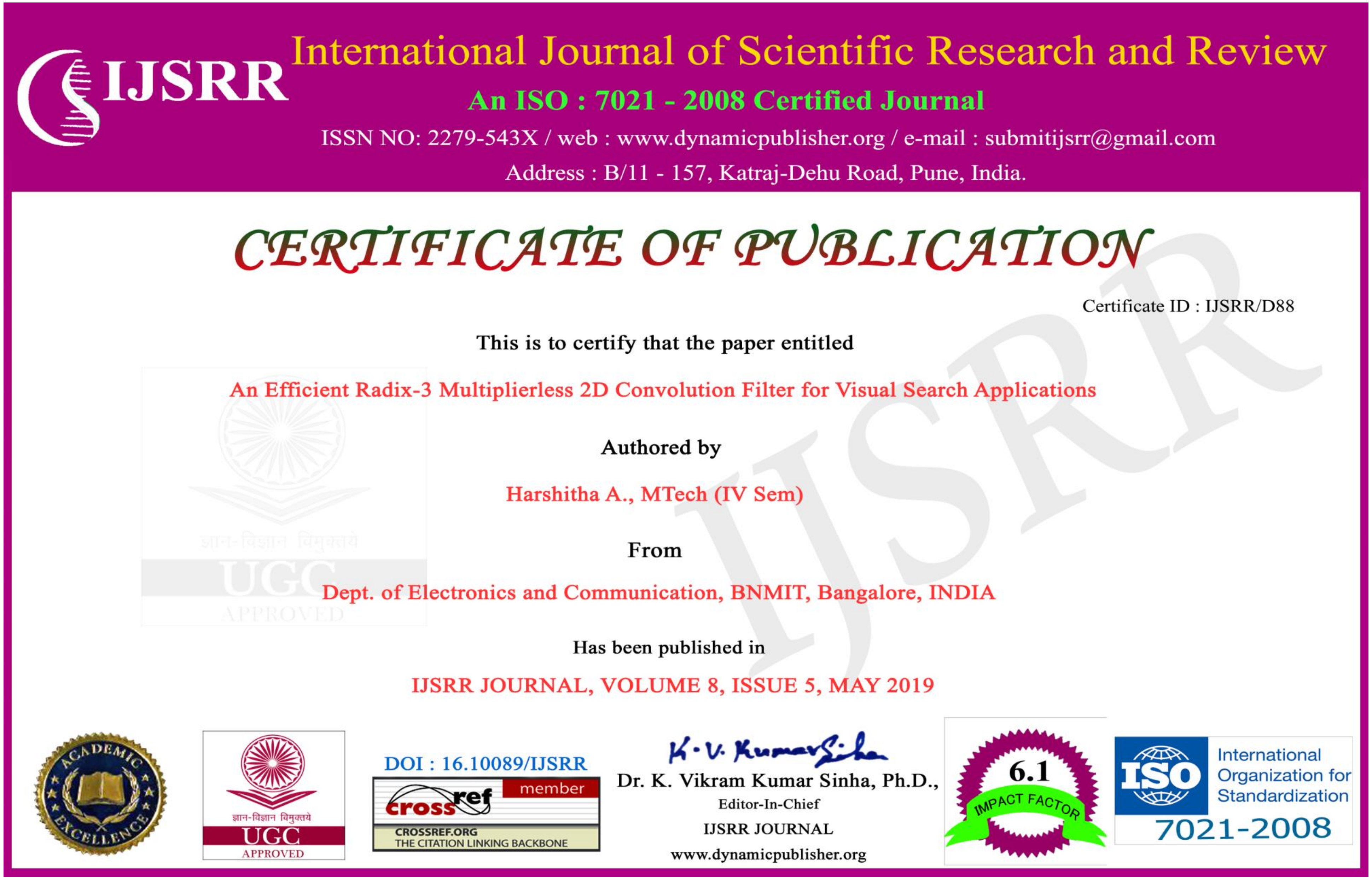
Abstract **– In real time Visual Search(VS) applications which involves object recognition and segmentation using image processing, a filtering stage is mandatory. Usually, in all filtering applications convolution is performed between the image tile and a kernel. A two-dimensional convolution circuit is presented based on the Bachet's weight decomposition theorem which basically exploits a new Radix-3 partitioning method of integer numbers. The proposed method eliminates the use of multipliers by -** ***chain of simplified single precision floating point (FP) adders*** **designed based on IEEE 754 FP signed addition algorithm. In order to increase the resolution and accuracy, the floating point representation is used. The proposed design provides more efficient implementation of floating point convolution architecture when a fixed set of coefficients is used over a range of input values. The design is coded in verilog and simulated using Isim from Xilinx ISE. The design is targeted on Virtex 7 FPGA Board. This paper mainly focuses at reducing the LUT count and power consumption by implementing proposed algorithm for multiplying two floating point numbers. The** ***area/LUT*** **count is significantly reduced by** ***48.54%*** **and power consumption by** ***20.17%*** **in comparison with [1]. However there's a tradeoff in delay.**

Index Terms— ***Bachet's weight decomposition, Convolution, Multipliers, Radix-3 partitioning.***

I. INTRODUCTION

In Visual Search applications, a 2D convolution filter is used to perform filtering as the object to be processed is always two dimensional. In such applications convolution is performed between the input image and a kernel. In these applications, Software implementations of the algorithm is limited in terms of maximum frequency that can be achieved to process a single image, thus making them difficult to achieve real-time performance, while the problem is even more worse for images having higher resolutions unless and until strong simplifications of the algorithms is done. Due to these reasons, Hardware implementation of filtering stages has become more and more frequent, but it must be noted that the effort to achieve optimized designs is not justified by the real-time requirements only. In fact, the increasing demand for handheld devices and Internet-of-Things(IoT) applications is making path for new designs, capable of achieving better Area-Power-Delay (ADP) trade-offs. Considering all these issues, during the last years, several hardware designs have been proposed aiming to obtain optimized filtering architectures. Hardware complexity of the design is the major problem for the applications aiming at high speed, lower area. Such complexity, certainly deteriorates the performance, due to the failure in the allocation of a large number of arithmetic operators resulting in consequent slackening of the overall circuit. The survey related to these problems shows that the above issue is usually managed by either adapting full/partial serialization of the filters and folding techniques, or by superseding the inherent complexity of fused Multiply-Adders and Multiply- ACcumulators (MAC). The former way usually results in significant reduction in filter performance. Hence, the latter approach is opted as it is the most accurate way to achieve a good Power, Performances, Area (PPA) trade-off. In such cases, several authors have preferred to replace the multiplier circuitry by adders and shifters in according to the coding of the operands, Canonical Signed Digit (CSD) and Modified Booth (MB). The effective simplification of filtering circuits is, when one of the operands is reduced to a bounded set of pre-calculated values, as in the case of pre-defined filter kernels. For the cases like these, in order to partition multiplications in simpler shifts and additions the Distributed Arithmetic (DA) method can be successfully employed. Usage of Distributed Arithmetic can be advantageous over MB and CSD by using memories to store pre-calculated partial sums, whose number can be decreased by incorporating Multiple Constant Multiplication (MCM) technique. Conversely, performance of DA is the result of trade-off between its natural bit serial operation and the parallelism by which the partial sums are calculated, leading to the excessive increase in the number of mapped physical resources.

This paper presents new radix-3 partitioning scheme based on Bachet's weight decomposition theorem



IJSRR International Journal of Scientific Research and Review

An ISO : 7021 - 2008 Certified Journal

ISSN NO: 2279-543X / web : www.dynamicpublisher.org / e-mail : submitijsrr@gmail.com

Address : B/11 - 157, Katraj-Dehu Road, Pune, India.

CERTIFICATE OF PUBLICATION

Certificate ID : IJSRR/D88

This is to certify that the paper entitled

An Efficient Radix-3 Multiplierless 2D Convolution Filter for Visual Search Applications

Authored by

Harshitha A., MTech (IV Sem)

From

Dept. of Electronics and Communication, BNMIT, Bangalore, INDIA

Has been published in

IJSRR JOURNAL, VOLUME 8, ISSUE 5, MAY 2019

ACADEMIC EXCELLENCE

ज्ञान-विज्ञान विमुक्तये UGC APPROVED

DOI : 16.10089/IJSRR

crossref member CROSSREF.ORG THE CITATION LINKING BACKBONE

[illegible]

Dr. K. Vikram Kumar Sinha, Ph.D.,
Editor-In-Chief
IJSRR JOURNAL
www.dynamicpublisher.org

6.1 IMPACT FACTOR

ISO International Organization for Standardization 7021-2008

IJSRR

ABSTRACT

Digital Signal Processing is one of the most sought-after processing techniques in various applications like audio, video, speech recognition, image processing *etc*. Among the several Digital Image Processing applications, Visual Search (VS) applications are trending in this new age digital era. Visual Search is basically visual interpretation of the data/ information that is being surfed in the search engine. Visual Search applications involving object recognition and segmentation require a filtering stage. Filtering is the process of convolution between the image tile and a kernel. Thus, convolution forms the core of filtering operation.

A two-dimensional convolution circuit is presented based on the Bachet's weight decomposition theorem which basically exploits a new Radix-3 partitioning method of integer numbers. This method replaces multipliers by - *chain of simplified single precision floating point (FP) adders,* thus eliminating shifters and recoding circuitry that is common in most of the other existing multiplication structures. The adders are designed based on IEEE-754 FP signed addition algorithm. In order to increase the resolution and accuracy, the floating-point representation is used over fixed point. The proposed design provides more efficient implementation of floating-point convolution architecture when a fixed set of coefficients is used over a range of input values.

The design is coded in verilog and simulated using Isim from Xilinx ISE. The design is synthesized using Cadence Encounter RTL Compiler targeted to 45nm slow_vdd1v0_basicCells.lib technology library to generate area, power and timing reports. The proposed 2D convolution architecture consumes an area of 30005 std cells, power of 246445.108 nW and timing of 74.584 ns.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

DSP	Digital Signal Processing
DIP	Digital image processing
VS	Visual Search
2D	Two Dimension
IoT	Internet-of-Things
ADP	Area-Power-Delay
MAC	Multiply-ACcumulators
PPA	Power, Performances, Area
CSD	Canonical Signed Digit
MB	Modified Booth
DA	Discrete Arithmetic
MCM	Multiple Constant Multiplication
FP32	32-bit/ Single Precision Floating Point
HW	Hardware
SW	Software
DPD	Densely Packed Decimal
EOP	Effective Operation
BCD	Binary Coded Decimal
IEEE	Institute of Electrical and Electronics Engineers

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

Chapter Outline:

This chapter gives an overview of various Digital Signal Processing (DSP) Applications, Motivation, Problem Statement, Objective and Organization of Report.

1.1 Introduction to DSP Applications

Digital Signal Processing is a processing technique where a signal is analyzed and modified in order to enhance its performance and efficiency. To achieve or produce higher quality signals several computational and numerical algorithms are applied to both analog and digital signals. DSP basically is used for error detection, filtering and compression of analog signals in transit. These operations are performed by executing processing algorithms that are DSP specific through a Digital Signal Processor or other device that can function in a similar manner. DSP after converting an analog signal into digital signal applies appropriate processing algorithms and techniques on signals. For instance, DSP targets on reduction of distortion and noise when audio signals are considered. Among many DSP applications, some of the popularly known are processing/compression of video/audio signals, digital image processing, recognition of speech signals, defence, space and biomedical applications.

Digital image processing (DIP) is the process of performing image processing on digital images using various available computer algorithms in order to improve the resolution accuracy. As a sub-field of DSP, DIP is advantageous as it allows application of huge range of algorithms to the input data thus aiding in reduction of noise build-up and distortion in signal during processing. DIP can be modeled in a similar fashion to the multidimensional systems as images are naturally two dimensional.

Digital Image Processing involves several processing stages such as acquisition, preprocessing, segmentation, feature extraction, recognition and interpretation which is depicted in Figure 1.1.

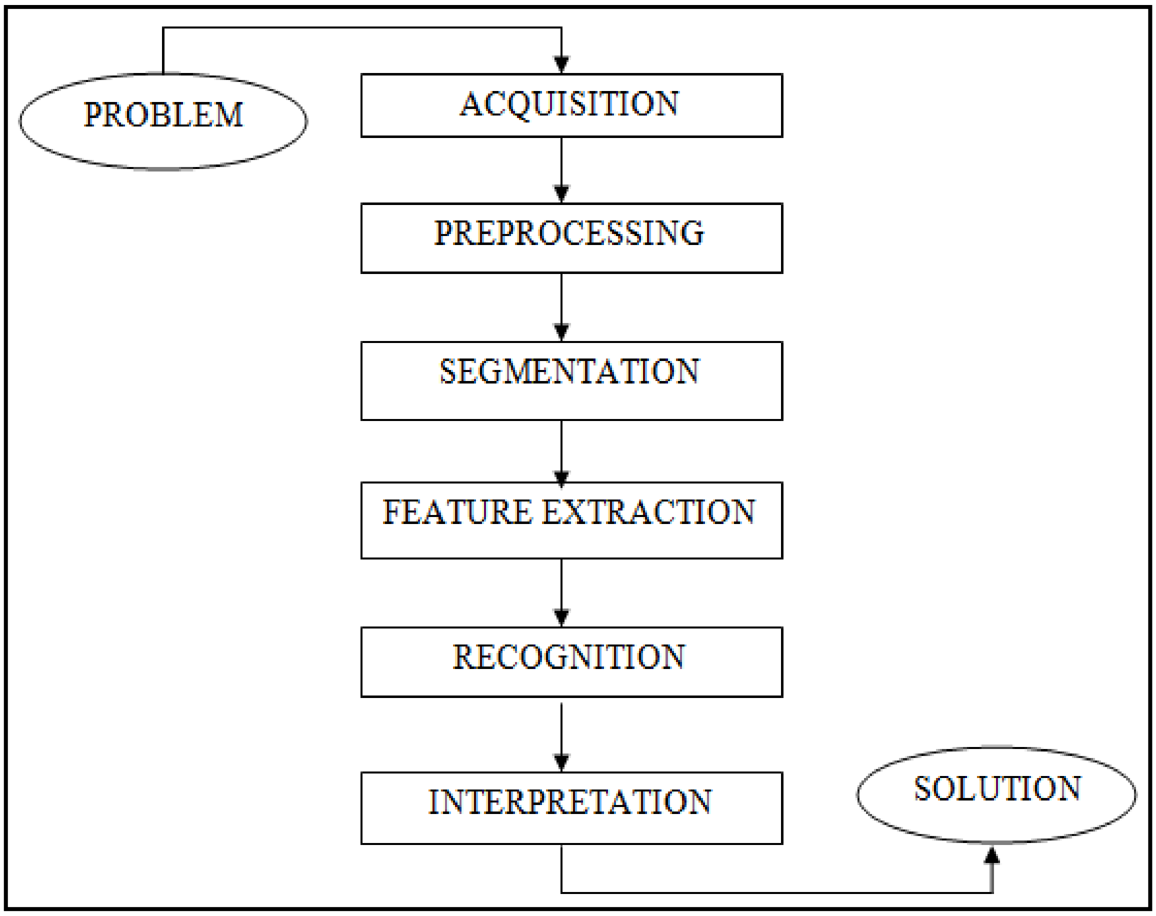


Figure 1.1: Digital Image Processing (DIP) Stages

DIP system can be divided into following steps:

- **Acquisition:** Responsible for converting a real scene captured by digital camera or other device into a digital image.
- **Preprocessing:** Some processing techniques for digital image correction are applied, so that the image can be better processed in the following steps.
- **Segmentation:** Segmentation seeks to split an image into regions or objects of interest. This process is guided by object features such as color, texture, and shape.
- **Feature Extraction:** In feature extraction step database is generated with characteristics of detected objects in segmentation step, such as size, area, and shape.
- **Recognition and Interpretation:** Finally, in recognition and interpretation step, objects detected in the segmentation stage, together with their characteristics, are classified and interpreted according to the desired solution.

Visual Search (VS) is one of the image processing applications where a filtering stage is always mandatory. Hence, a two- dimension (2D) convolution filter is used to perform filtering as the object to be processed is always two dimensional where, convolution is performed between the input image and a kernel. In these applications, Software implementations of the algorithm is limited in terms of maximum frequency that can be achieved to process a single image, thus making them difficult to achieve real-time performance, while the problem is even more worse for images having higher resolutions unless and until strong simplifications of the algorithms is done. Due to these reasons, Hardware implementation of filtering stages has become more and more frequent, but it must be noted that the effort to achieve optimized designs is not justified by the real-time requirements only. In fact, the increasing demand for handheld devices and Internet-of-Things (IoT) applications is making path for new designs, capable of achieving better Area-Power-Delay (ADP) trade-offs.

In order to address the issues discussed above, during the last few years, several hardware designs have been proposed aiming to obtain optimized filtering architectures. Hardware complexity of the design is the major problem for the applications aiming at high speed and lower area. Such complexity certainly deteriorates the performance, due to the failure in the allocation of a large number of arithmetic operators resulting in consequent slackening of the overall circuit.

The survey related to these problems shows that the above issue is usually managed by either adapting full/partial serialization of the filters and folding techniques, or by superseding the inherent complexity of fused Multiply-Adders and Multiply-Accumulators (MAC). The former way usually results in significant reduction in filter performance. Hence, the latter approach is opted as it is the most accurate way to achieve a good Power, Performances, Area (PPA) trade-off. In such cases, several authors have preferred to replace the multiplier circuitry by adders and shifters in accordance to the coding of the operands, Canonical Signed Digit (CSD) and Modified Booth (MB). The effective simplification of filtering circuits is, when one of the operands is reduced to a bounded set of pre-calculated values, as in the case of pre-defined filter kernels. For the cases like these, in order to partition multiplications in simpler shifts and additions the Discrete Arithmetic (DA) method can be successfully employed. Usage of Discrete Arithmetic can be advantageous over MB and CSD by using memories to store pre-calculated partial sums, whose number can be decreased by incorporating Multiple Constant Multiplication (MCM) technique. Conversely, performance of DA is the result of trade-off between its natural bit serial operation and the parallelism by which the partial sums are calculated, leading to the excessive increase in the number of mapped physical resources.

In this work new Radix-3 partitioning scheme based on Bachet's weight decomposition theorem with the purpose to improve performances of general purpose multipliers in the specific contexts of MCM is presented. The proposed architecture of the convolution circuit employs a multiplier to perform multiplication of 32-bit floating point (FP32) kernel coefficients with integer inputs (input pixels) defined over a range of integer values compatible with multimedia applications. This procedure eliminates the need for shifters and auxiliary circuitry which is common with most of the multiplier schemes by substituting multiplier with FP adders without compromising on the accuracy of the result. The proposed work will be best suitable for filtering applications involving multiple constant multiplications.

1.2 Motivation

In all VS applications, 2D filtering is a mandatory process. The rising demand for multimedia applications of higher quality and the increase in the functionalities required for VS has paved way for the logarithmic increase in devoted Hardware (HW) and Software (SW) systems' computational complexity. Added to this, is the rapid increment in the usage of the applications that mostly involve VS in the portable/handheld devices. All these result in the irreconcilable constraints such as mapped physical resources count and elaboration velocity. There are a number of HW and SW solutions to address these problems. However, due to the greater number of FP MAC computations which require larger memory resources performance of the design is hindered when implemented using a SW. In order to compensate this inadequacy, several authors have proposed numerous HW solutions. Conversely, to overcome these constraints, hardware implementation of the design needs to be simplified to a greater extent that results in alternative convolution architectures that are less complex. One such hardware architecture is the proposed work which focuses on making the design suitable for complex and large computations in VS applications. The proposed architecture will be best suitable where MAC operations and multiple constant multiplications are required.

1.3 Problem Statement

Increasing demand for higher quality multimedia contents has resulted in extreme research activity to improve the filter architecture. In applications such as video and image elaboration which aim at reducing the area with increasing computational complexity, HW complexity has become the prominent concern. This causes deterioration in the process of allocation of the resources for larger number of arithmetic operators which may further hinder the overall circuit performance. Thus with the purpose of improving the hardware architecture in terms of mapped physical resources, power and elaboration speed for VS applications, 2D convolution filter architecture has been proposed.

1.4 Objective

The important goals of this project:

- To design efficient 2D convolution filter to specifically improve VS applications.
- To explore radix-3 partitioning technique originated from Bachet's weight decomposition theory, where a multiplier is replaced by FP adders arranged in pipeline stages.
- The proposed work also finds its usage in all MAC applications.

1.5 Application Areas

- Electronic commerce (e-commerce) – online marketing and shopping of products.
- Bio-medical – various scan tests like ultrasound, X- ray, eco-cardiogram *etc*.
- Aerial/Space – remote sensing of the objects like interplanetary images *etc*.
- Industrial – recognition and sorting of products.
- Defence – Radar and sonar.
- Communication – Audio and video signal processing.
- Entertainment and Infotainment – Photoshop, pinterest.

1.6 Report Organization

The organization of the project report is as follows:

- **Chapter 1:** Focuses on Introduction to DSP Applications, Motivation, Problem Statement, Objective, Applications and Report Organization.
- **Chapter 2:** Describes about the research and analysis on several existing adder structures and convolution module architectures employing different implementation techniques.
- **Chapter 3:** Gives an insight to convolution module design specifications and the methodology adopted to design the proposed work.
- **Chapter 4:** Shows the simulation results of the basic building blocks of the proposed convolution architecture.
- **Chapter 5:** Discusses verification results and synthesis results of the integrated top level modules.
- **Chapter 6:** Concludes the design work with its future scope.

CHAPTER 2

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

Chapter Outline:

Literature review is a research on the existing work that basically involves an analysis about progress in the various theoretical and methodological techniques. This chapter mainly speaks about the various available convolution architectures, adder implementation methodologies and several existing multiplication techniques. It also concentrates on modules/blocks used to accomplish these techniques and their disadvantages in terms of area required, power consumed, delay produced, hardware resources needed, etc.

2.1 A Review on Convolution Architecture

Title: "VLSI Implementation of an Adaptive Edge-Enhanced Image Scalar for Real-Time Multimedia Applications".

Author: Shih-Lun Chen.

Description: This paper presents the architecture design of the simplified bilinear interpolation structure depicted in Figure 2.1, where the path delay is reduced as the design employs seven stages of pipelined architecture and multipliers are pipelined in two stages. From the register bank, input values say $P_{(m,\ n)}$, $P_{(m+1,\ n)}$, $P_{(m,\ n+1)}$, $P_{(m+1,\ n+1)}$ and from sharp filter the values say $P\_{(m,n)}$, $P\_{(m+1,\ n)}$, $P\_{(m,\ n+1)}$, $P\_{(m+1,\ n+1)}$ are obtained respectively. Here the edge detector produces the multiplexer control signal say M_ctr. For bilinear interpolation, the M_ctr signal acts as a select signal to select the data from sharp filter and from register bank, the source values. In this work hardware sharing technique is adopted. Thus employing 3 registers can reduce single calculation cycle of the function $\mathrm{dy} \times \ (P_{i,j+1} - P_{(i,j)}) + P_{(i,j)}$ which in turn reduces the need to use a multiplier and 2 adders. Here, FP calculations are replaced with integer calculations by using approximate technique. Hence a simple bilinear interpolation circuit can be designed only by using integer adders that are cost effective, multiplier circuitry and shifters instead of using FP adders.

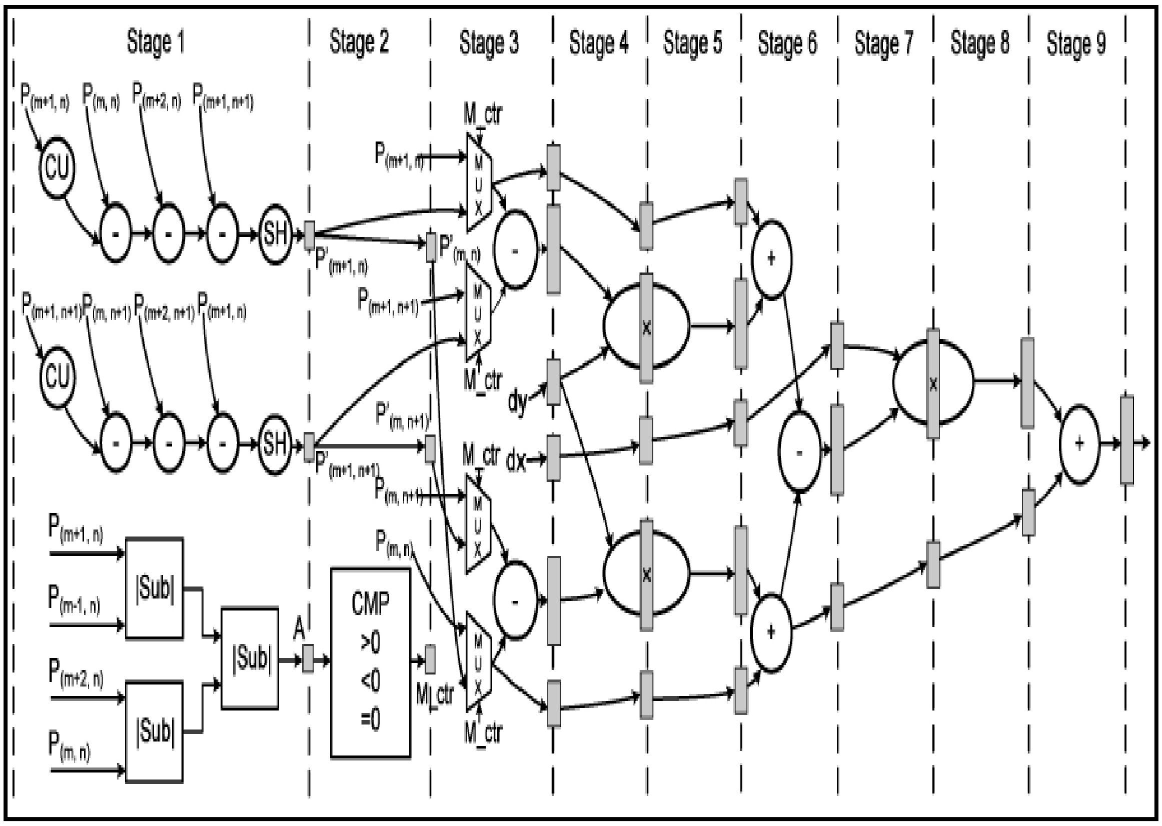


Figure 2.1: Pipelined Structure of Image Scalar Design [2]

Title: **“**Design and FPGA Implementation of 2D Gaussian Surround Function with Reduced On-Chip Memory Utilization”.

Authors: M. C Hanumantharaju, M. Ravishankar, D. R Rameshbabu.

Description: This work presents a multiplier design which has three levels of pipelining and uses parallel circuits of higher degree. Here multiplier is used to perform multiplication between two positive numbers say, n1 and n2 which are 8bit wide each. For Gaussian functions like smoothing, blurring that is of bigger size, a 16×16 multiplier can be incorporated in the design. The product from such a multiplier is also 16-bit data. The complete structure of the multiplier with three pipeline stages where each stage comprises of registers internally is shown in Figure 2.2. This work aims at improving the computation speed of the multiplier. Gaussian Control, exponential of Gaussian function, ROM, Adder and Multiplier are the functional modules. In applications such as smoothing, filtering, edge detection and enhancement of image, two dimensional Gaussian function finds its place.

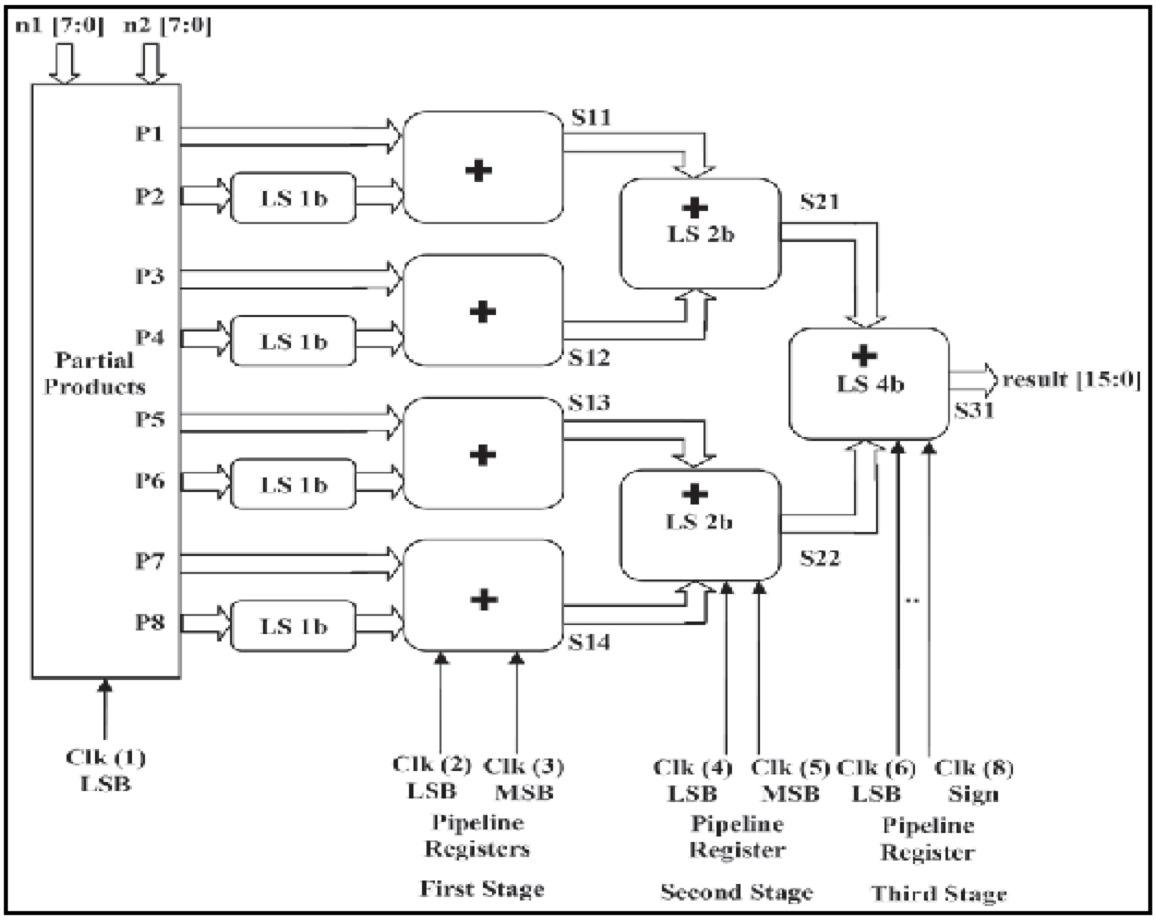


Figure 2.2: Multiplier Structure with 3 Pipeline Stages [3]

Title: “Implementation of Fixed Point 2D Gaussian Filter for Image Processing Based on FPGA”

Authors: Frank Cabello, Julio Leon, Yuso Iano, Rangel Arthur

Description: In this paper Multiply and Accumulate (MAC) architecture that is suitable for HW implementation is presented. Convolution is performed by acquiring from the actual image, [3 ×3] sub-images and each sub-image is convoluted with the kernel of size [3 ×3]. Every element in the sub-image is multiplied with the corresponding kernel values. After multiplication, these products are added together to obtain the single pixel value as the final result. Multiplication and Accumulation process is illustrated in Figure 2.3 where buffers arranged in rows hold the sub-images values that belong to the neighboring [3×3] window. Further the pixel values held in the row buffers are moved to shift registers. In order to improve computation speed, N-1 row buffers are needed if the convolution is performed on kernel matrix of size $[N \times N]$. The procedure used in this work is same as that of simultaneous convolution in FIR filters.

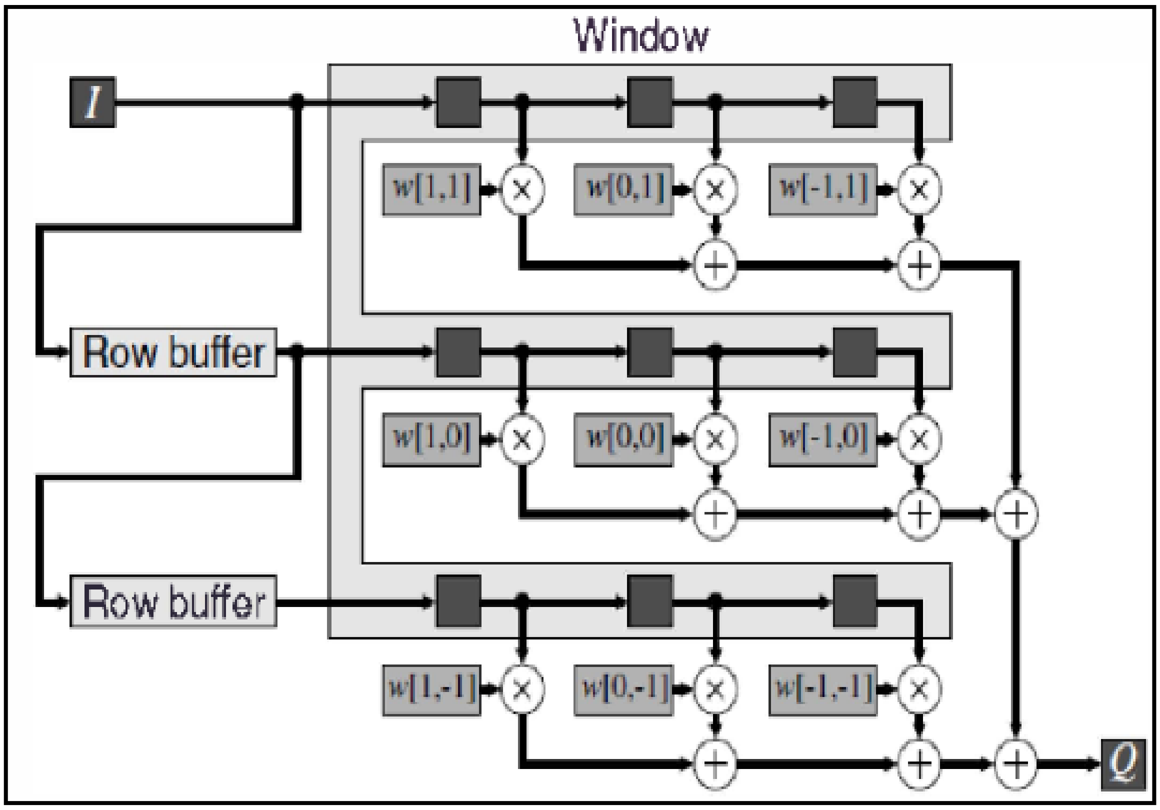


Figure 2.3: Convolution Architecture [4]

Title: “FPGA implementation of filtered image using 2D Gaussian filter”.

Authors: Leila kabbai, Anissa Sghaier, Ali Douik and Mohsen Machhout.

Description: Here, 2D Gaussian Filter structure is designed for hardware implementation. The core of the convolution is multiplication, hence three variants of multiplication is presented in this work. Among the 3, first one is a conventional method. Second one uses FPGA and DSP to enhance the FPGA resource’s scalability, thus trying to increase the computation speed. Final (third) one uses multiplier macros for greater precision and effective utilization of resources of FPGA. In this work, comparison of image quality is drawn between the HW (using VHDL) and SW (using MATLAB) implementations through PSNR (Peak Signal-Noise Ratio). Synchronous image filter block diagram is depicted in Figure 2.4. The design consists of three main modules namely a Control, Convolution and three B_RAM modules (one for input image matrix, second for Gaussian mask while third one for filtered output image). The address generation to B_RAM1 and B_RAM2 and data transfer from one B_RAM to the appropriate convolution module to perform MAC operation and finally storing the result in B_RAM3 is all managed by a Control module. Therefore synchronization between the B_RAM modules is necessary. B_RAM1 and B_RAM2 store Gaussian mask in .coe file and data test image in .coe file format generated using MATLAB software respectively. Control unit reads the data in B_RAM1 and B_RAM2 and after the necessary operations are performed, the filtered image data is stored in B_RAM3.

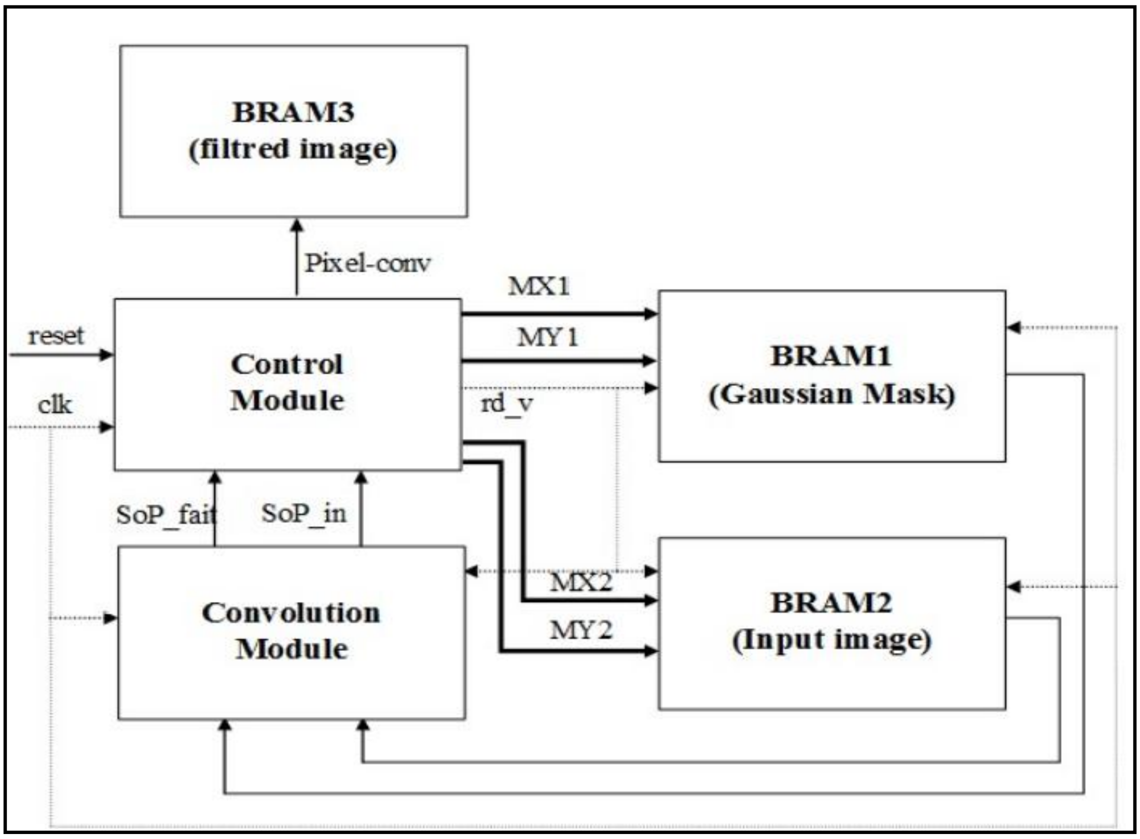


Figure 2.4: Synchronous Architecture of Image Filter [5]

2.2 A Glance on Adder Implementation

Title: “Implementation of Optimized Floating Point Adder on FPGA”

Authors: Deepak Mishra, Vipul Agarwal

Description: This paper presents the architecture of FP64 adder of IEEE754-2008 standard depicted in Figure 2.5 which can be further applied to 128-bit format. Here, the adder module design is done by encoding data in Densely Packed Decimal (DPD) number format for Discrete Floating Point calculations. The proposed design in this work targets on reducing consumption of power by incorporating Bypass adder. The decoder block in the design is used to extract sign (A_S, B_S), exponent (A_Exp, B_Exp) and mantissa (A_Mant, B_Mant) of operands A and B respectively. Effective Operation (EOP) unit performs logical xor on sign field of operand A and operand B. The design employs two paths say, Channel_A and Channel_B as the data paths for bigger and smaller values.

Mantissa adjustment is achieved by performing logical right shift using Barrel shifter. Now the appropriate operation is performed by a Binary Coded Decimal (BCD) adder. The Rounding block rounds off the mantissa value and Mantissa, Exponent and Sign fields are represented in IEEE754 – 2008 double precision floating point format.

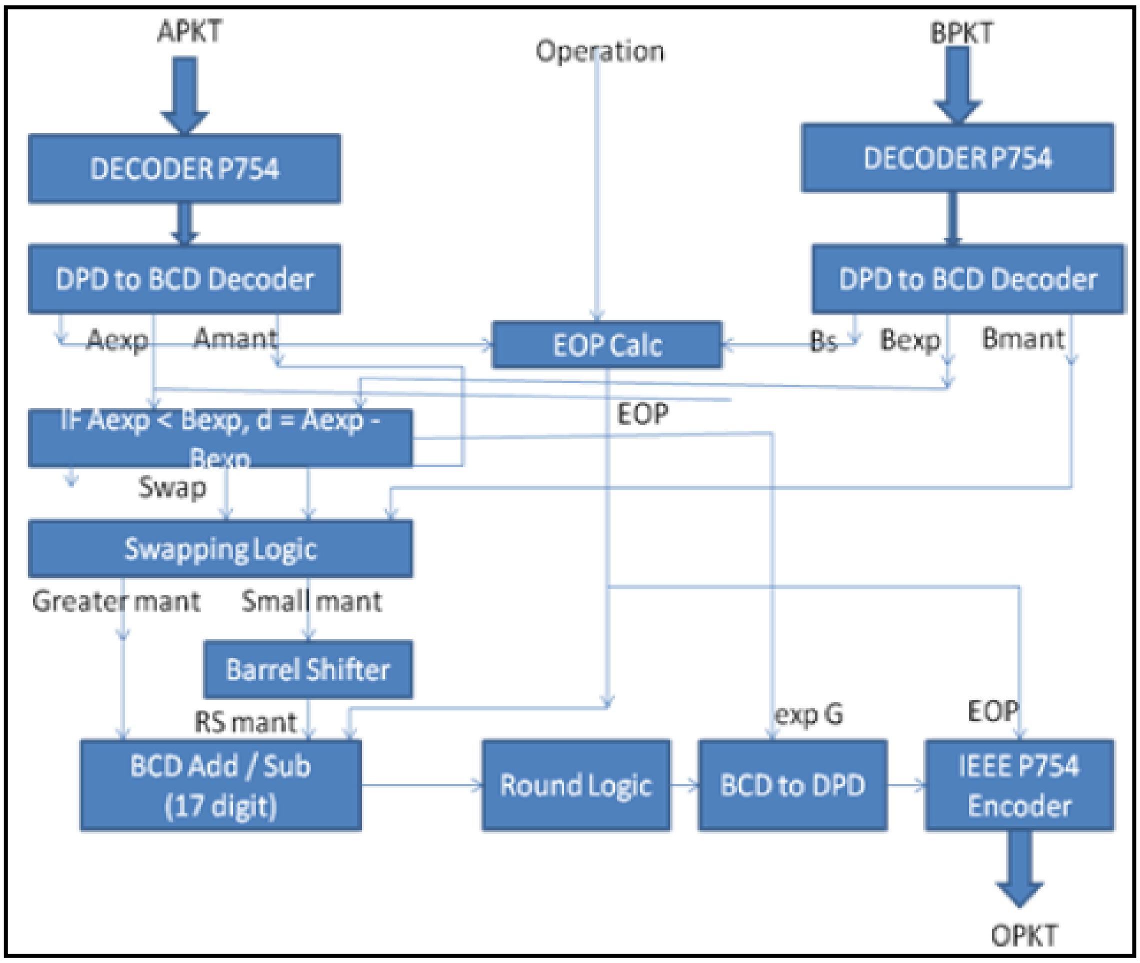


Figure 2.5: Floating Point Adder Architecture [6]

Title: "Area Efficient Floating-Point Adder and Multiplier with IEEE-754 Compatible Semantics"

Authors: Andreas Ehliar

Description: This work presents FP adder which is an open source and radix16 multiplier based on 36bit custom number. The main disadvantage of this format is that the rounding process is more complex when compared to radix2 format. The FP adder is a pipelined architecture with registers arranged in 5 pipeline stages. The various stages involved in the process of addition of FP numbers in IEEE standard is depicted in Figure 2.6. Here, in stage one comparison is drawn between the operands and in stage 2 swapping of operands is done if necessary. Then the operands are added and if needed, normalization of the result is done and exponent value is adjusted accordingly. To obtain the result and represent it in FP32 IEEE754 format, the intermediate result after the above mentioned process is post processed and if required rounding of the result is done to achieve accuracy. Again architecture of the multiplier is partitioned into three parts that has six registers in pipeline. In the first stage, mantissa multiplication is performed by an integer multiplier, while the second stage does normalization and post-processing and rounding of the number to its closest value is done in the third stage and finally the result is represented in FP32 IEEE754 standard.

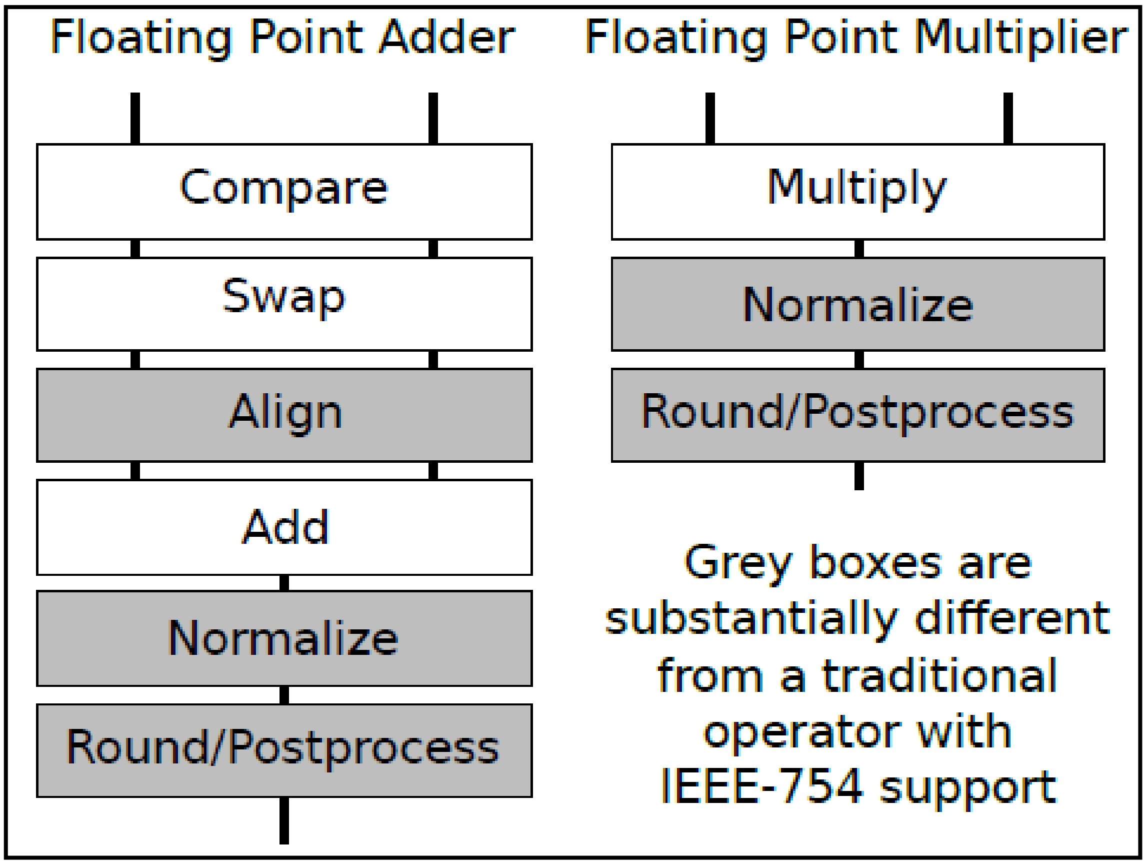

Floating Point Adder
Compare
Swap
Align
Add
Normalize
Round/Postprocess
Floating Point Multiplier
Multiply
Normalize
Round/Postprocess
Grey boxes are substantially different from a traditional operator with IEEE-754 support

Figure 2.6: Outline of High Radix FP Addition/Multiplication [7]

2.3 Survey on Multiplication Techniques

Title: “Low Power 4*4 Canonical Signed Digit Multiplier using 90nm Technology”.

Authors: Saloni, Dr. Neelam Rup Prakash

Description: This research work focuses on developing an algorithm for Multiplication using Canonical Signed Digit (CSD) format. Performance and area comparison is drawn between the array multiplier and Canonical Signed Digit multiplier. In constant multipliers, constant multiplication is the process of addition/ subtraction of terms in partial products that correspond to non-zero bit position. To represent the result in CSD format, least number of addition or subtraction iterations is required as multiplier encoded with CSD has minimum non-zero bits. Partial products are generated using the CSD truth table for multiplier. From the aid of truth table, appropriate multiplier structure for each bit combination is designed as seen in Figure 2.7.

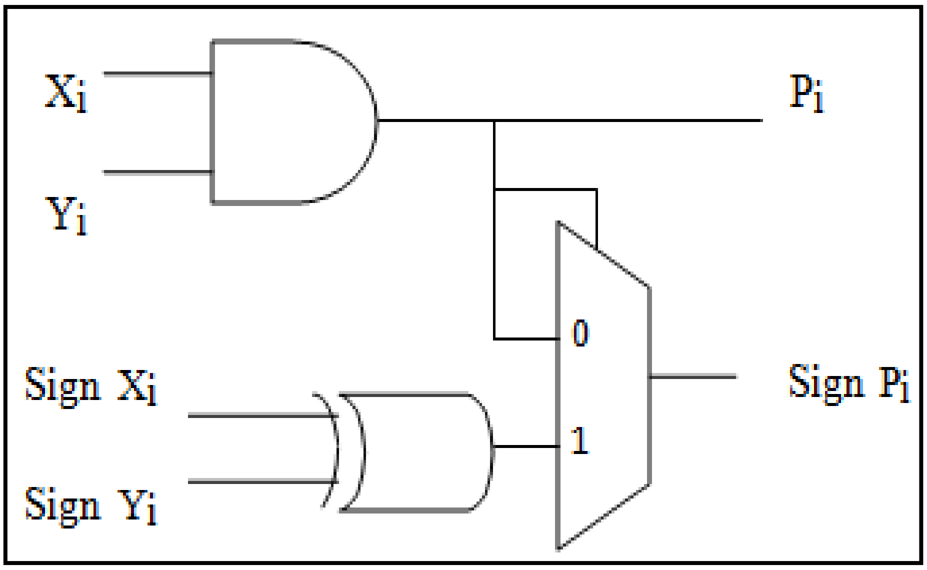


Figure 2.7: CSD Multiplier Structure [8]

Title: "Design and Implementation of Booth Multiplier and Modified Booth Multiplier".

Authors: Sakthivel. B, K. Maheshwari, J. Manojprabakar, S. Nandhini, A. Saravanapriya.

Description: This work presents the implementation technique of 8bit Modified Booth (MB) multiplier and the same is compared with the conventional 4bit Booth multiplier. MB multiplier that works on MB algorithm can accept signed operands as inputs and perform both addition/subtraction accordingly. No particularly different operation is performed to treat signed numbers. This paper focuses on the technique to implement parallel MAC with the least possible delay. Such MACs find place in DSP, video or graphics applications. In order to increase the multiplication speed, high speed adders are used to design MB multiplier hardware structure which basically consists of four modules namely, a complement generator, Booth Encoder, Partial product calculation unit and finally carry look ahead adder as seen in Figure 2.8.

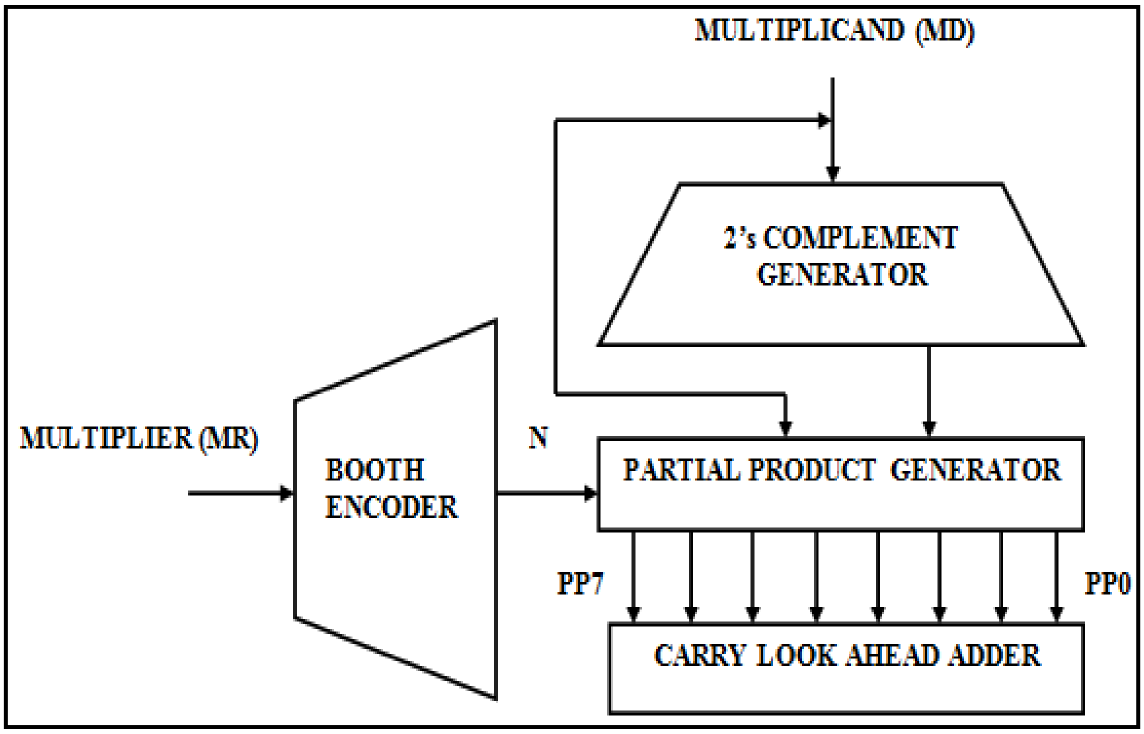


Figure 2.8: Architecture of Booth Multiplier [9]

Title: "Design and Implementation of Floating Point Multiplier for Better Timing Performance".

Authors: B. Sreenivasa Ganesh, J.E.N. Abhilash, G. Rajesh Kumar.

Description: This paper presents algorithm for multiplication on FP numbers and multiplier schematic for the same is shown in Figure 2.9.

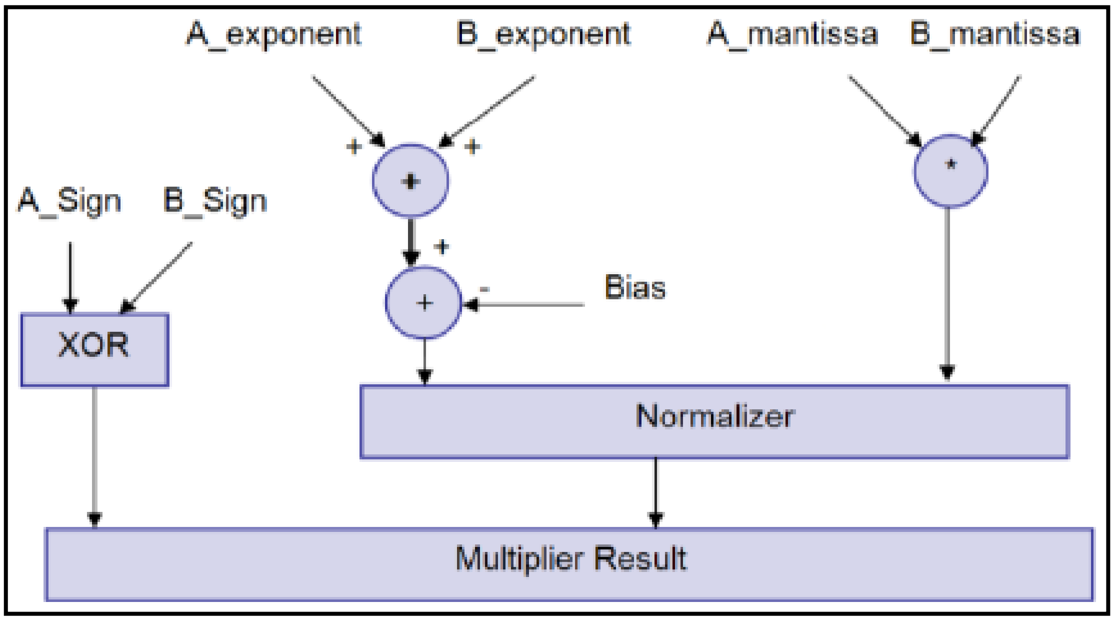


Figure 2.9: Block Diagram of FP Multiplier [10]

The FP multiplication algorithm to multiply two FP numbers of IEEE754 standard is as follows.

- Significand multiplication (mantissa with hidden bit) *i.e.* (1.M1×1.M2).
- Exponent addition *i.e.* (E_1 + E_2 - Exponent bias).
- Sign bit calculation using logical exclusive-or operation *i.e.* (S_1 *xor* S_2).
- Normalization *i.e.* adjusting significand such that MSB is 1.
- Rounding off of the product to the nearest possible value.
- Check if there is any overflow or underflow in the result.

Figure 2.9 depicts FP multiplier block diagram. The Mantissa Calculation Unit is a 24 bit multiplier as the input numbers are in single precision IEEE754 FP format. The Exponent Calculation Unit is an 8bit adder performing addition of two 8bit exponents. The result so obtained is subtracted by the exponent bias value (127). The multiplied significand result is normalized and exponent is adjusted accordingly in the normalization unit. Sign calculation unit performs the ex-or to decide the sign of the resultant. Final FP multiplier result is the concatenation of sign bit, exponent and mantissa.

2.4 Highlights of Existing Work

Enormous research has been done in this field, thus coming up with new MAC architectures with different number formats. Several other papers referred in order to accomplish the proposed design are listed in the Table 2.1. Below table highlights some of the important features adopted in literature work related to the proposed filtering architecture.

Table 2.1: Summary of Papers Related to the Proposed Work

Authors	**Title**	**Year of Publication**	**Inference**
G. D. Licciardo, C. Cappetta, L. Di Benedetto, and M. Vigliar	Multiplierless Stream Processor for 2D Filtering in Visual Search Applications.	2018	2D convolution filtering module structure is presented here.

Authors	Title	Year of Publication	Inference
G. D. Licciardo, C. Cappetta, L. Di Benedetto, and M. Vigliar	Weighted partitioning for fast multiplier-less multiple constant convolution circuit	2017	Employs radix3 partitioning method based on Bachet's weight decomposition theory. Shifters and recoding circuitry is completely replaced by adders.
G. D. Licciardo, C. Cappetta, Antonio D'Arienzo, and Alfredo Rubino	Stream Processor for Real-Time Inverse Tone Mapping of Full-HD Images	2015	Edge preserving bilateral filter architecture and its operation principle. The design avoids DRAM, instead uses acquiring devices coupled with sensors.
Sahdev D. Kanjariya, Rutarth Patel	Architecture and Design of Generic IEEE-754 Based Floating Point Adder, Subtractor and Multiplier	2015	Floating point adder, subtractor and multiplier design along with some exceptional cases is presented in this work. Virtex-4 FPGA implementation.
Ghassem Jaberipur, B. Parhami, Saeid Gorgin	Redundant-Digit Floating-Point Addition Scheme Based on a Stored Rounding Value	2010	Radix16 redundant representations, floating point addition/subtraction algorithm and hardware architectures.
Bipul C. Paul, Shinobu Fujita and Masaki Okajima	ROM-Based Logic (RBL) Design: A Low-Power 16 Bit Multiplier	2009	ROM based 16×16 multiplier for low-power applications.

Limitations of the existing work are listed below.

- Increased area with the increasing design complexity.
- High power consumption.
- Low elaboration speed.
- Increase in the requirement of hardware resources for higher resolution image processing applications.

CHAPTER 3

DESIGN SPECIFICATIONS AND METHODOLOGY

CHAPTER 3

DESIGN SPECIFICATIONS AND METHODOLOGY

Chapter Outline:

This chapter gives a detailed description of the convolution architecture taking into consideration each and every block of the design. It gives a brief outlook of the functional specifications, software requirements of the design and also the mathematical background of the design.

3.1 Design Specifications

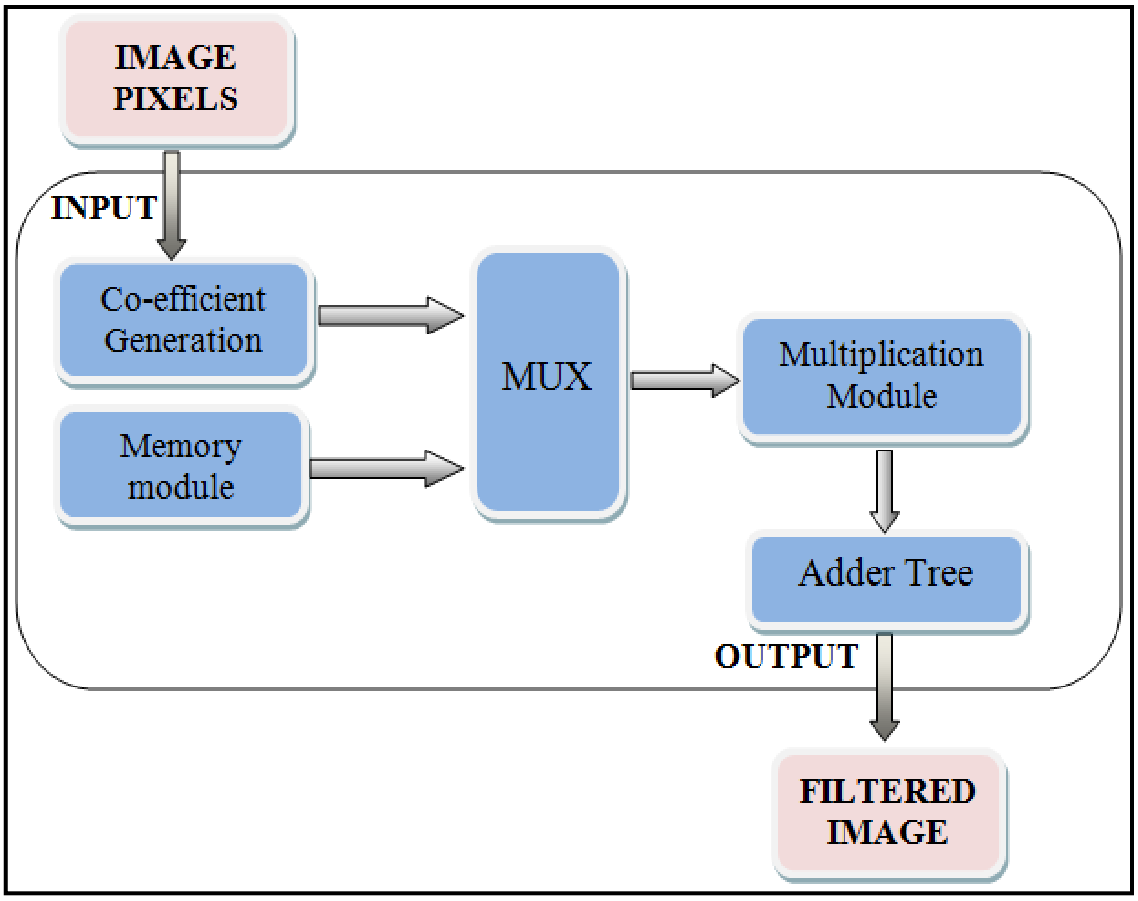


Figure 3.1: Block Diagram of 2D Convolution Module

The block diagram of two-dimensional convolution module is shown in Figure 3.1. Coefficient generation module, Memory module, Multiplexer, Multiplier and Adders are units used in the proposed work to design 2D convolution module. The input is chosen as (3×3) image with each pixel value 8 bit wide. The convolution operation is performed on input image and the kernel coefficients with kernel size=3. To obtain filtered image of size (3×3) as the output nine cycles of convolution operations need to be performed.

Below are some of the functional specifications of the proposed design.

- Input is an 8-bit unsigned integer data, specifically image pixel in image processing applications.
- Output is a single precision (32-bit) floating point number of IEEE754 standard.
- The design requires six 2:1 MUXs, one of the inputs being from LUT and the other from Coefficient-ROM (CROM). The output is fed to the adders.
- Adders perform addition on IEEE754 single precision FP numbers. Inputs to and outputs from FP adders are 32-bit data.
- The adder stages depth is given by $\lfloor log_2(n+1) \rfloor$ = $\lfloor log_2(8+1) \rfloor$ =3.
- Each convolution operation requires nine MAC operations thus nine multipliers and eight adders are needed.

3.2 Software Requirement

- Language: Verilog HDL.
- Tools Used:
  - ➢ Xilinx ISE to simulate the design and Xilinx Synthesis Technology (XST) to synthesize and generate RTL schematics.
  - ➢ Cadence Encounter RTL Compiler to generate synthesis reports for area, power and timing.

3.3 Mathematical Background

Over several decades there’s an issue as to how all the numbers can be represented using the minimum number of integers such that the numbers over a certain range can be expressed as a combination of them. Because several applications discussed earlier in the introduction require filtering apparatus and the complexity of filtering circuit increases with increasing range of image pixels, it’s high time to find a solution to reduce the complexity of the filtering circuit. The proposed work presents a technique to eliminate the need to employ multipliers in convolution circuit by replacing them with a chain of adders.

There are several papers discussing as to how to represent numbers using a fixed set of integers. Having defined the set $W_r = \{3^0, 3^1, 3^2, \dots , 3^{m-1}, R\}$ with $R = r - (3^0 + 3^1 + 3^2 + \dots + 3^{m-1})$ each value ranging from 0 to r can be obtained by superimposing the terms in W_r that are multiplied with a coefficient λ_j where $\lambda_j \in \{ -1, 0, 1\}$. Generally, *decomposition* of an unsigned integer r is defined as a structured sequence of unsigned integer values such that $r = \beta_0 + \beta_1 + \dots + \beta_m$ *with* $\beta_0 < \beta_1 < \dots < \beta_m$ and as parts of the partition set $\{\beta_0, \dots, \beta_{m-1}, \beta_m\}$, it can be stated that

1. Each integer value in the range $0 \le p \le r$ can be described as

$$p = \sum_{j=0}^{m} \lambda_j \beta_j \qquad \dots\dots \quad (3.1)$$

2. An alternative decomposition of r satisfying (3.1) with *lesser parts* than m+1 does not exist.

Here each partition in r consists of exactly $m + 1 = \lfloor log_3 2r \rfloor + 1$ parts.

Table 3.1: Bachet's Weight Partition

INPUT	**PARTITION**					
	3^0	3^1	3^2	3^3	…	β_m
0	**0**	**0**	**0**	**0**	…	**0**
1	**+1**	**0**	**0**	**0**	…	**0**
2	**-1**	**+1**	**0**	**0**	…	**0**
3	**0**	**+1**	**0**	**0**	…	**0**
4	**+1**	**+1**	**0**	**0**	…	**0**
5	**-1**	**-1**	**+1**	**0**	…	**0**
⋮	⋮	⋮	⋮	⋮	⋮	⋮
P	λ_0	λ_1	λ_2	λ_3	……	λ_m
⋮	⋮	⋮	⋮	⋮	⋮	⋮
R	**+1**	**+1**	**+1**	**+1**	**+1**	**+1**

Table 3.1 shows how each positive integer can be represented in a radix-3 format. For instance: an 8 bit input takes {0, 1, 3, 9, 27, 81, 134} as its parts. The input 45 can be represented as a combination of coefficients and parts as 45= (0)1+ (0)3 + (-1)9 + (-1)27 + (+1)81 + (0)134 with the coefficients being {0, 0, -1, -1, +1, 0} as per Table 3.1.

The same partitioning method can be adopted to perform MAC operations between the vector of coefficients A_i and input vector x_i.

$$y = \sum_{i=0}^{i-1} A_i x_i \qquad \text{… …} \; (3.2)$$

Combining (1) and (2), we get,

$$y = \sum_{i=0}^{i-1} \sum_{j=0}^{m} A_i \lambda_{ij} \beta_j \qquad \text{… …} \; (3.3)$$

3.4 Architecture Design of Convolution Module

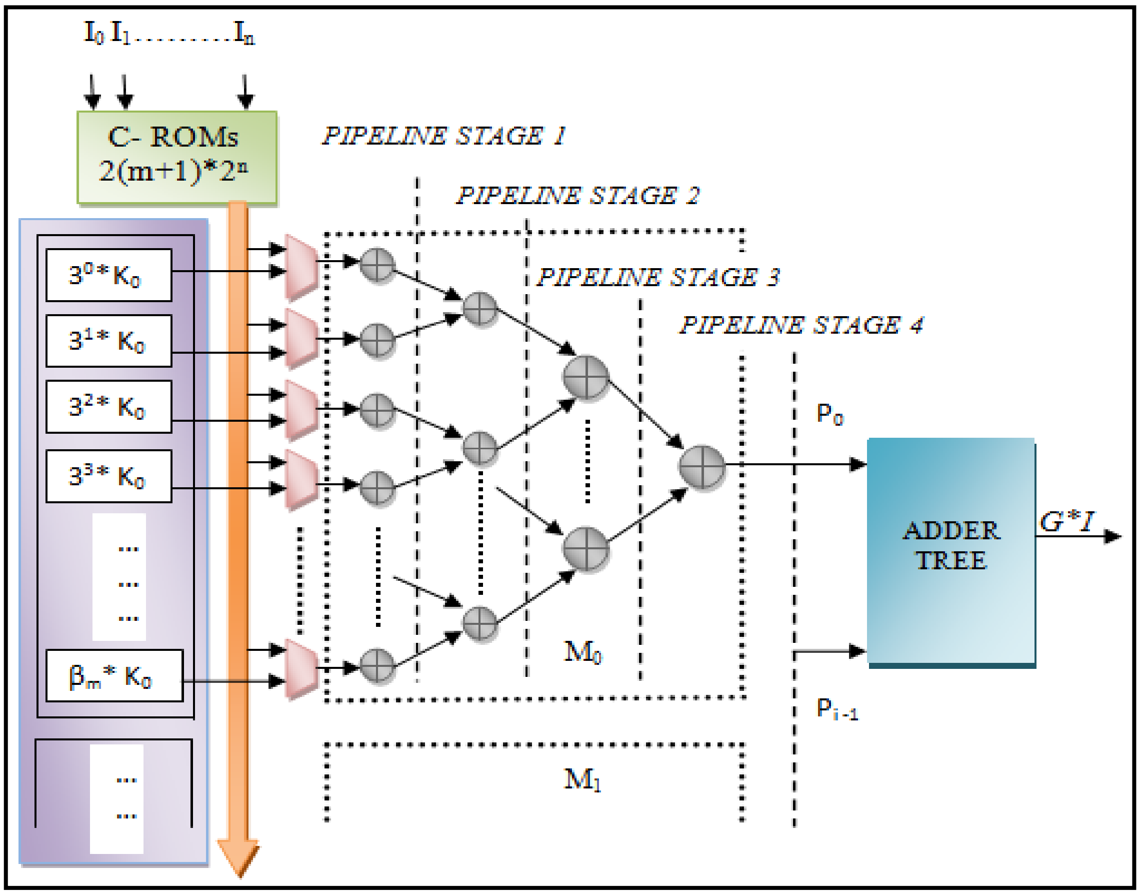


Figure 3.2: Multiplier and Accumulator (Convolution) Architecture

The hardware architecture of the proposed design can be implemented as shown in the Figure 3.2. This scheme performs the convolution of single precision floating point kernel vector, *G* with the vector of unsigned integer input *I* of *n* bits. To perform convolution, *G* is to be defined and each value in the input vector *I* is decomposed into *m+1* parts as per Table 3.1. Each value in *G* defined as [G_0 G_1.....G_{h-1}] is premultiplied with radix-3 weights defined as $\{3^0, 3^1, \ldots, \beta_m\}$ and stored in the LUT. These premultiplied values are stored in IEEE 754 single precision FP standard. Here $2(m+1)\times 2^n$ bit ROM stores the coefficients $\lambda_j \in$ {-1, 0, 1} as {11, 00, 01}(each sign coded with 2bits) which is used to select the sign of the input values through the multiplexer bank. The CROM output is fed to MUX which acts as a select signal to choose inputs to the adders in the first pipeline stage of multipliers. Further each multiplier is replaced by adders as represented within the dashed box in Figure 3.2 whose tree depth is $\lfloor log_2(n+1)\rfloor$. Unlike [12], adders can perform *addition of signed integer values* and hence values taking λ_j as -1 need not have to be stored in the two's complement form which further discards requirement of additional circuitry to perform the conversion of resulting data to obtain actual results. The adders are designed as per the IEEE 754 FP32 signed addition algorithm which can take all possible combinations {+ +, + -, - +, - -} of operands including zero. Also no separate method is used to code the exponents of values in G making the design more clear and simple.

Though this convolution architecture is compatible with all kinds of kernel we have chosen to implement a 2D Gaussian filter with Gaussian kernel defined as

$$G(p,q) = \frac{1}{2\pi\sigma^2} e^{-\frac{(p^2+q^2)}{2\sigma^2}} \qquad \ldots\ldots (3.4)$$

working with Unit-8 inputs. The input range is chosen to be *r* = 255 with the number of parts being $m+1 = \lfloor log_3 2r\rfloor + 1 = 6$ i.e. {1, 3, 9, 27, 81, 134} as $\beta_m \equiv R = 255 - (3^0 + 3^1 + 3^2 + 3^3 + 3^4) = 134$. The kernel size is chosen to be K = 3 as it is the lowest dimension that is allowed for Visual Search applications and $\sigma = 1$. Here, input is chosen as (3×3) image with each pixel value 8 bit wide whose(3×3) image matrix is as follows.

$$I = \begin{bmatrix} 130 & 135 & 140 \\ 117 & 123 & 130 \\ 105 & 113 & 121 \end{bmatrix} \qquad \ldots\ldots (3.5)$$

The(3×3) Gaussian Blur kernel matrix with $\sigma = 1$ is as follows:

$$G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad \dots\dots\ (3.6)$$

The convolution result thus obtained is the filtered output which is represented in the matrix form as follows.

$$G * I = \begin{bmatrix} 72 & 99 & 76 \\ 90 & 124 & 96 \\ 63 & 88 & 69 \end{bmatrix} \qquad \dots\dots\ (3.7)$$

3.4.1 Block Level Description

This section gives in detail, the description of every block used in the design with the inputs and outputs along with the data flow between the various blocks. Following are the several blocks used to accomplish the design.

- **Description of LUT/GK_ROM:**

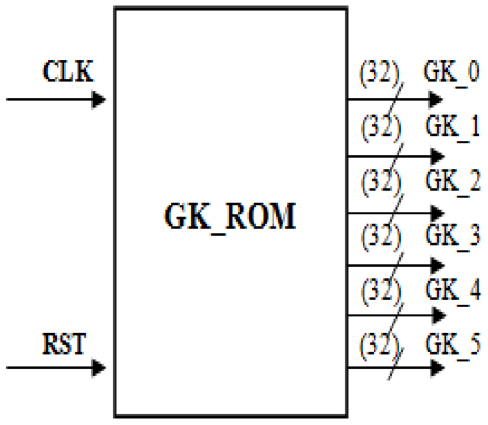


Figure 3.3: Schematic of GK_ROM

GK_ROM/ LUT block is used to store the premultiplied values of one of the operands with the radix3 weights. In case of image processing applications with 8bit input image pixels, kernel coefficients are premultiplied with six parts of radix3 weights and stored in the LUT (as per Table 3.1) in IEEE754 FP32 standard.

The schematic and signal description of GK_ROM is shown in Figure 3.3 and Table 3.2 respectively.

Table 3.2: Signal Description of GK_ROM Block

NAME	WIDTH	DIRECTION	DESCRIPTION
CLK	1	IN	Triggers the event during positive edge of the clock.
RST	1	IN	RST=1, resets the entire system, thus no operations occur.. RST=0, sets the system, thus normal operation is carried out.
GK_0 – GK_5	32	OUT	Premultiplied Kernel coefficients with base3 weights in IEEE754 FP32 standard.

Timing diagram of GK_ROM is shown in Figure 3.4. Here, one of the kernel coefficients say 4/16=0.25 from the eq^n 3.6 is chosen and is premultiplied by the base3 weights divided into six parts as $\{(3^0 \times 0.25), (3^1 \times 0.25), (3^2 \times 0.25), (3^3 \times 0.25), (3^4 \times 0.25), (3^5 \times 0.25)\}$ and the resulting values are stored in the GK_ROM in FP32 IEEE754 standard. During rising edge of the clock and when reset is low these values appear as the output. Hence reset is considered to be active low in the entire design. When reset is low the output will be 0.

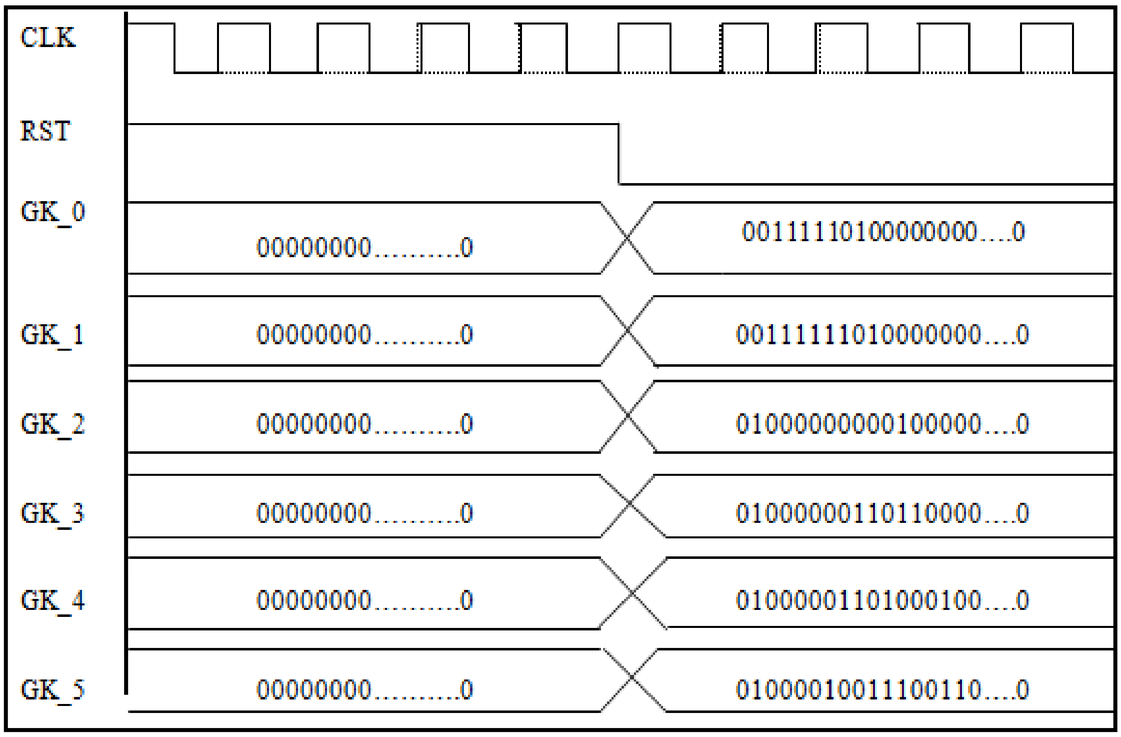


Figure 3.4: Timing Diagram of GK_ROM

- **Description of C_ROM (Coefficient ROM):**

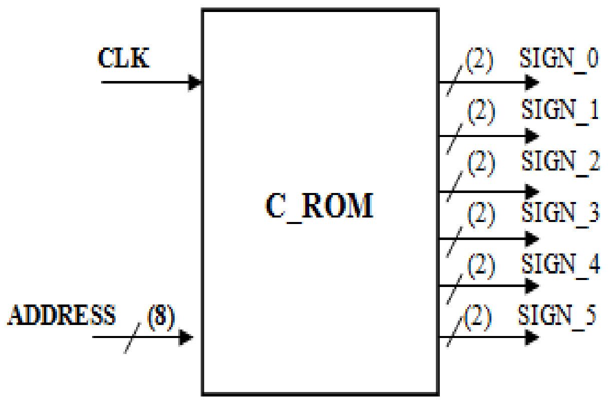


Figure 3.5: Schematic of C_ROM

C_ROM stores the ternary sign values i.e. (-1, 0, +1) as 2bit binary equivalents (11, 00, 01) respectively. When the input is fed to C_ROM block as 8bit address the corresponding sign values in accordance with their equivalent base3 weights as per Table 3.1 appear at the output as seen in Figure 3.5.

Signal description of the C_ROM block as per the schematic in Figure 3.5 is shown in Table 3.3.

Table 3.3: Signal Description of C_ROM Block

NAME	WIDTH	DIRECTION	DESCRIPTION
ADDRESS	8	IN	Binary equivalent of the 8-bit unsigned integer input (image pixel).
CLK	1	IN	Triggers the event during positive edge of the clock.
SIGN_0-SIGN_5	2	OUT	2-bit sign values stored in C_ROM for the given input as per the Table 3.1.

In Figure 3.6 8bit image pixel is provided as the input address. The first input is 13 (represented as 8bit binary equivalent in Figure 5.6) whose corresponding sign values as per Table 3.1 are {+1, +1, +1, 0, 0, 0}. These signs appear as 2bit binary values at the output where 0 is represented as 00, +1 as 01 and -1 as 11 respectively. Second input 117 is an image pixel chosen from *eq*^{n} 3.5 whose sign values are {0, 0, +1, +1, +1, 0} and output is represented in its 2bit binary equivalent form. Variation of the output is as per the clock's rising edge and reset is active low.

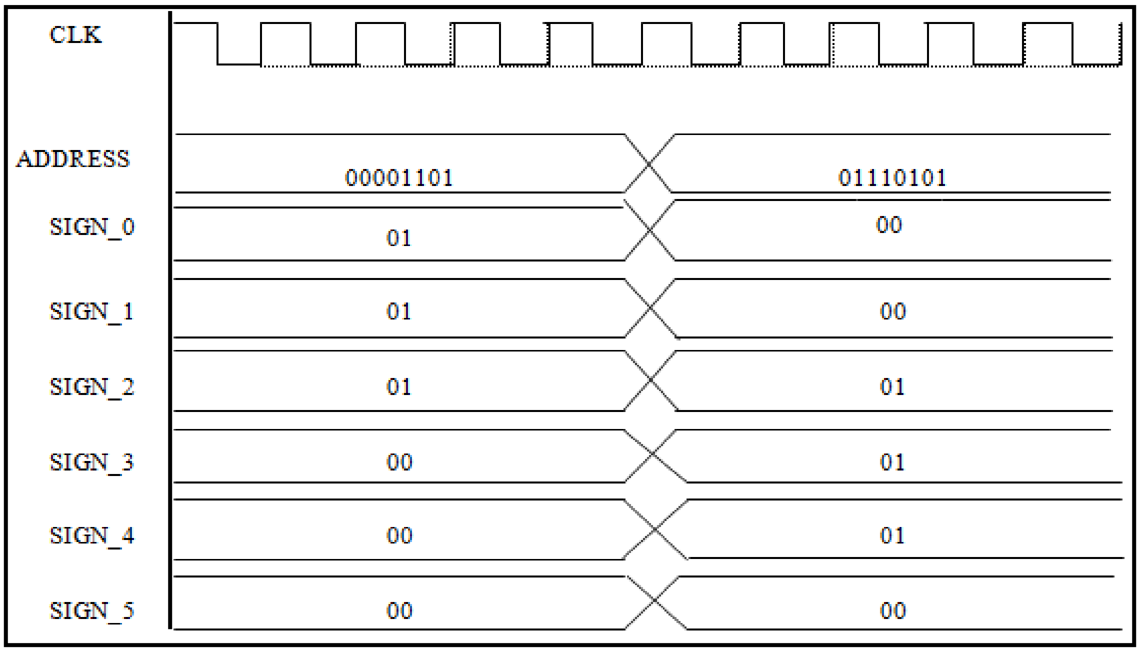


Figure 3.6: Timing Diagram of C_ROM

- **Functional Description of Multiplexer (MUX):**

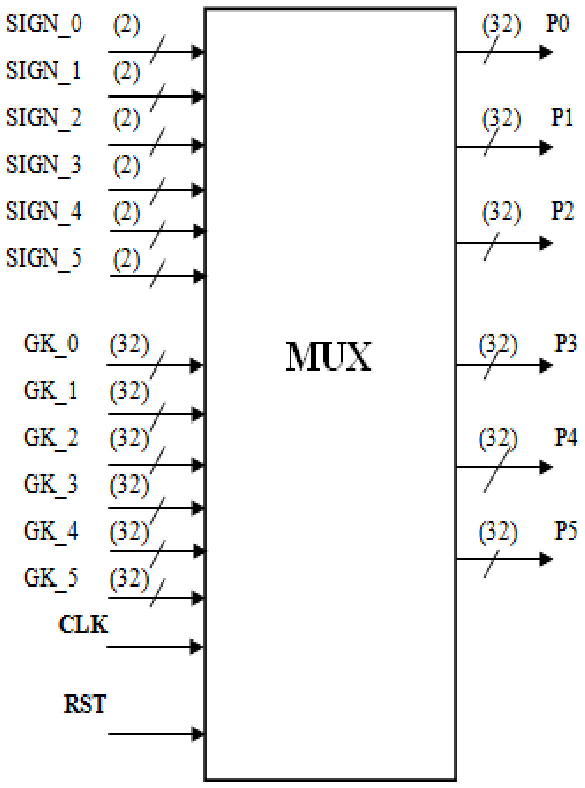


Figure 3.7: Schematic of Multiplexer

Multiplexer, based on the input pixels, chooses the appropriate premultiplied kernel coefficient set stored in GK_ROM module in IEEE 754 single precision FP standard with the respective sign coefficients set for that particular input pixel stored in C_ROM module. Thus one of the inputs to the MUX is from GK_ROM while the other is from C_ROM. The output from the multiplexer are sign values adjoined with the premultiplied coefficients respectively to form the single precision FP equivalents according the Bachet's weight decomposition seen in Table 3.1.

Schematic and signal description of the Multiplexer is shown in Figure 3.7 and Table 3.4 respectively.

Table 3.4: Signal Description of Multiplexer Block

NAME	WIDTH	DIRECTION	DESCRIPTION
CLK	1	IN	Triggers during positive edge.
RST	1	IN	When reset *i.e.* RST=1 no operation, when set *i.e.* RST=0 normal operation
GK_0 – GK_5	32	IN	Output from the GK_ROM which are premultiplied Kernel coefficients.
SIGN_0 – SIGN_5	2	IN	Output from C_ROM. Equivalent sign values to the input provided.
P0 – P5	32	OUT	Kernel coefficients adjoined with sign values as per Table 3.1.

Based on the sign values (SIGN_0–SIGN_5) the premultiplied data stored in GK_ROM (GK_0-GK_5) vary and appear at the ouput as (P0-P5). If the sign value is 00 the output is 0. If the sign value is 01 then the ouput is a positive number and the MSB in the 32bit ouput becomes 0. If the sign value is 11 then the output is a negative number and MSB in the 32bit output becomes 1. The same operation can be observed in the Figure 3.8.

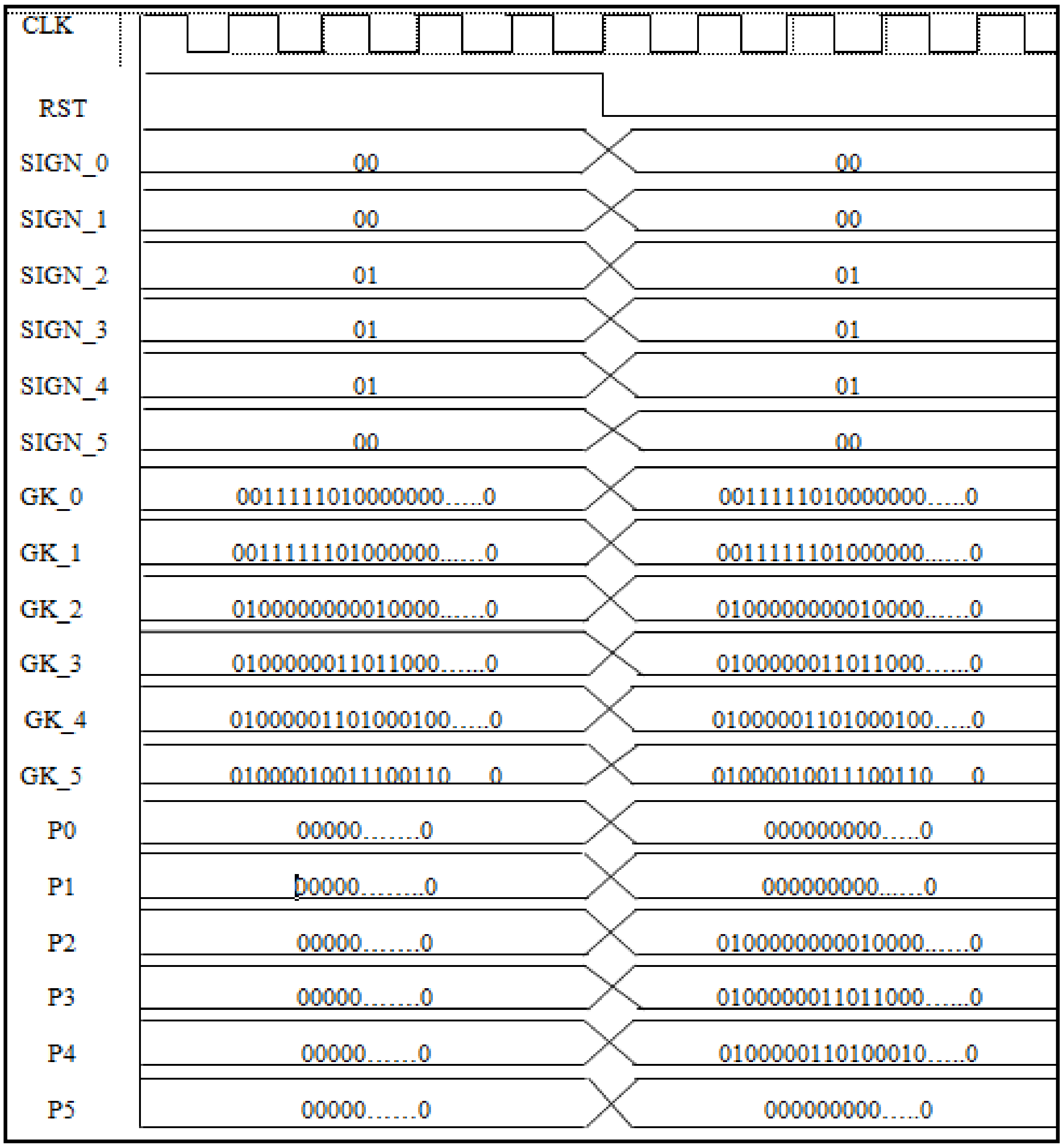


Figure 3.8: Timing Diagram of Multiplexer

- **Functional Description of FP32 ADDER:**

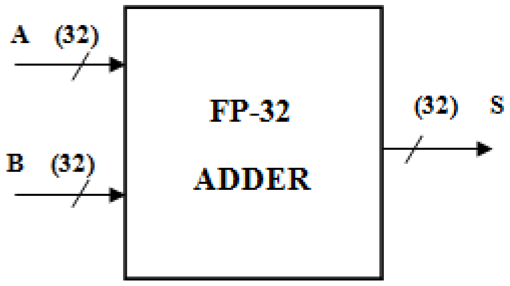


Figure 3.9: Schematic of FP32 ADDER

Inputs to the fisrt pipelined stage of adders is from MUX and the output from the first adder stage is fed to the second stage of adders and so on. Both inputs to and outputs from the adder are in FP32 IEEE754 standard. These adders are designed by combining both addition and subtraction floating point algorithms as the inputs to adders are signed values.

Schematic and signal description of single precision floating point adder are shown in Figure 3.9 and Table 3.5 respectively.

Table 3.5: Signal Description of FP32 Adder.

NAME	WIDTH	DIRECTION	DESCRIPTION
A	32	IN	Operand A in IEEE754 single precision floating point standard.
B	32	IN	Operand B in IEEE754 single precision floating point standard.
S	32	OUT	Sum after performing addition on signed numbers.

In Figure 3.10 for the sake of simplicity, values are represented in hexadecimal instead of representing in FP32 IEEE754 standard. The first set of inputs are A= 7.36, B= -1.19 and the corresponding sum is S=6.17. The second set of inputs are A= 0.99, B= 6.27 and the corresponding sum is S= 7.26.

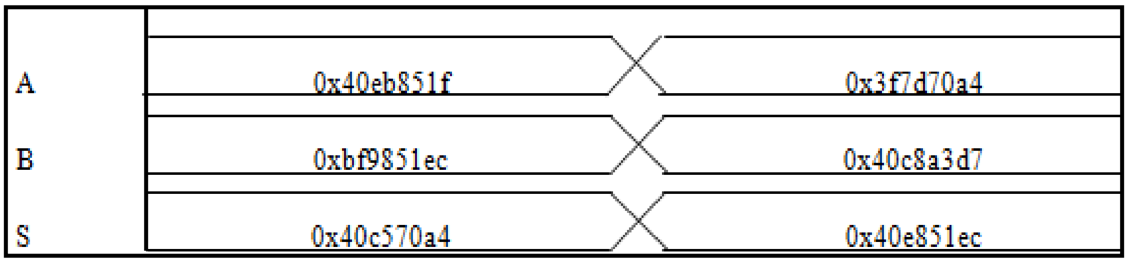


Figure 3.10: Timing Diagram of FP32 Adder

- **Description of FP32 Multiplier:**

One of the inputs to multiplier is an image pixel (I) while the other is the premultiplied kernel coefficient stored in GK_ROM. The output from the final pipeline stage of adders is the multiplier output. Both input to and output from multiplier are in IEEE754 FP32 standard.

Schematic and signal description of single precision floating point multiplier are shown in Figure 3.11 and Table 3.6 respectively.

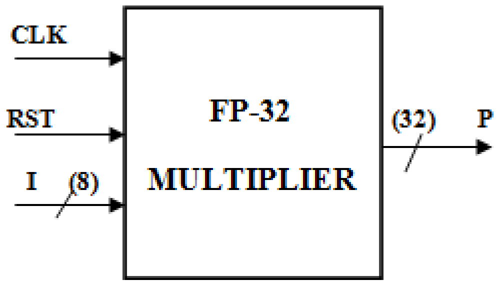


Figure 3.11: Schematic of FP32 Multiplier

Table 3.6: Signal Description of FP32 Multiplier.

NAME	WIDTH	DIRECTION	DESCRIPTION
CLK	1	IN	Triggers during positive edge of the clock signal.
RST	1	IN	When reset i.e. RST=1, no operation takes place. When set i.e. RST=0, normal operation takes place.
I	8	IN	8bit unsigned integer input (image pixel).
P	32	OUT	32bit product from the multiplier which is fed to the adder tree.

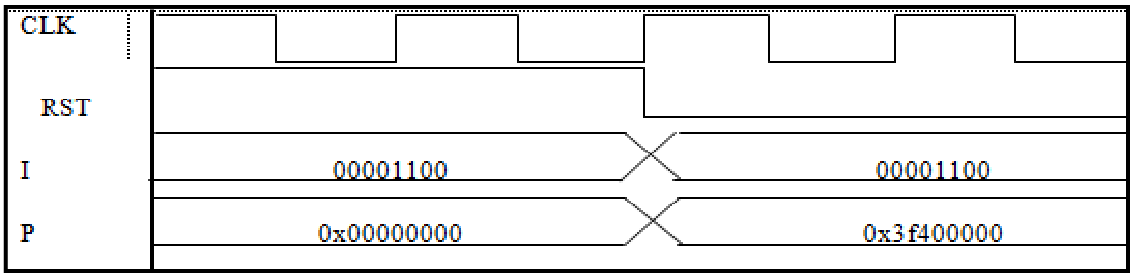


Figure 3.12: Timing Diagram of FP32 Multiplier

One of the inputs to the multiplier is I= 12 while the other is 0.0625 *i.e.* $(3^0 \times \frac{1}{16})$, the value stored in GK_ROM. The output is the product P= 0.75 as seen in Figure 3.12.

- **Block Description of Convolution Unit (one pixel):**

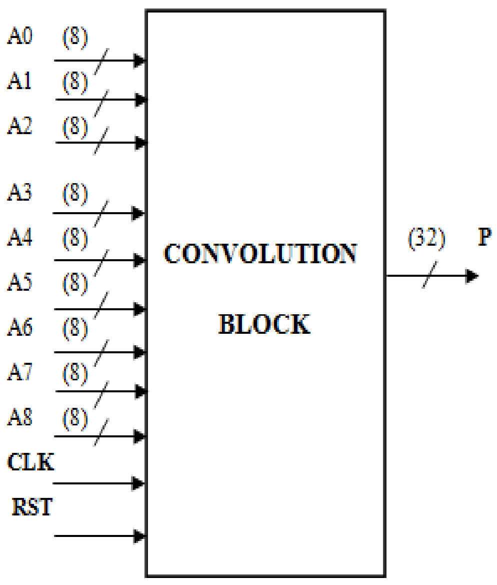


Figure 3.13: Schematic of Convolution Unit (one pixel)

Convolution primarily can be described as a simple Multiplication and Accumulation (MAC) operation. The inputs to multipliers are an image matrix and a kernel matrix. Each multiplier performs multiplication of an image pixel and a kernel coefficient and output from these multipliers are added together to obtain the convoluted output values. The input to convolution module is 8bit unsigned integer and output from this module is in IEEE754 FP32 format.

The schematic and signal description of the convolution module is depicted in Figure 3.13 and Table 3.7 respectively.

Table 3.7: Signal description of Convolution module

NAME	WIDTH	DIRECTION	DESCRIPTION
CLK	1	IN	Triggers during positive edge of the clock signal.
RST	1	IN	When reset i.e. RST=1, no operation takes place. When set i.e. RST=0, normal operation takes place.
A0 - A8	8	IN	8bit unsigned integer inputs (image pixels).
P	32	OUT	32bit convoluted output values.

The convolution operation is performed between Image matrix (A0-A8) and Kernel matrix defined in the eq^n 3.5 and eq^n 3.6 respectively. One cycle of convolution operation is performed on the centre image pixel and result P for the same is shown in Figure 3.14.

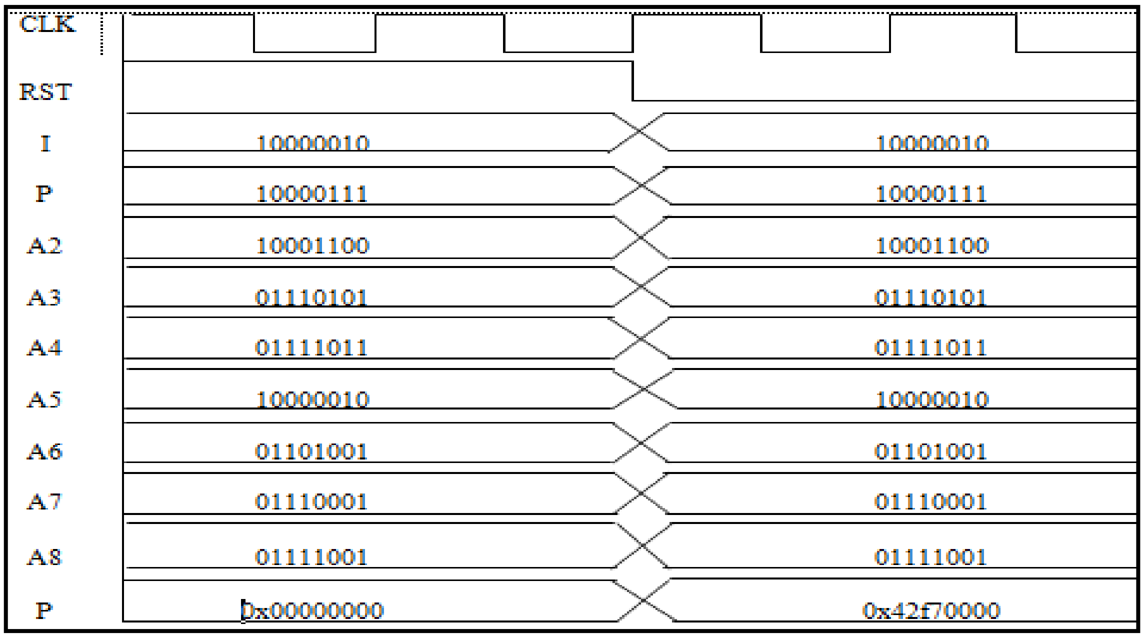


Figure 3.14: Timing Diagram of Convolution Unit (one pixel convolution)

- **Block Description 2D Convolution Filter:** The schematic and signal description of convolution filter is shown in Figure 3.15 and Table 3.8 respectively.

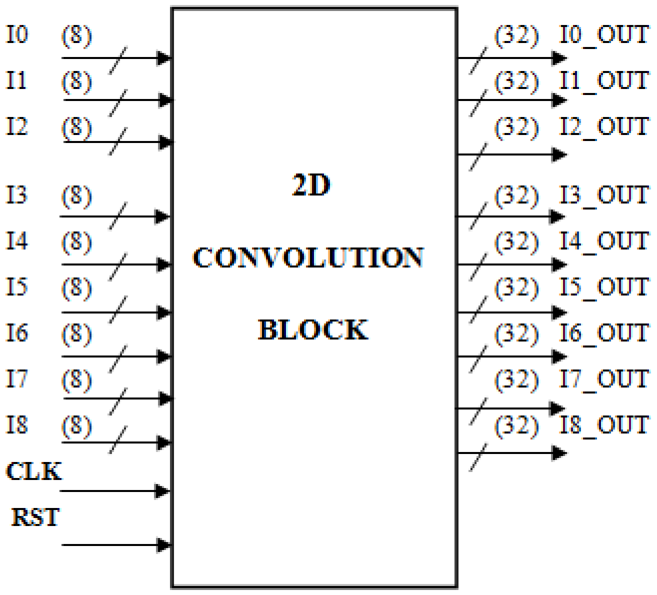


Figure 3.15: RTL Schematic of 2D Convolution Filter

Convolution operation is performed in several cycles to obtain the filtered output for a (3×3) image and the filtered output as shown in the *eq*^{n} 3.7 is represented in Figure 3.16

Table 3.8: Signal description of 2D Convolution module

NAME	WIDTH	DIRECTION	DESCRIPTION
CLK	1	IN	Triggers during positive edge of the clock signal.
RST	1	IN	When reset i.e. RST=1, no operation takes place. When set i.e. RST=0, normal operation takes place.
I0 - I8	8	IN	8bit unsigned integer inputs (image pixels).
I0_OUT - I8_OUT	32	OUT	32bit convoluted/filtered output values.

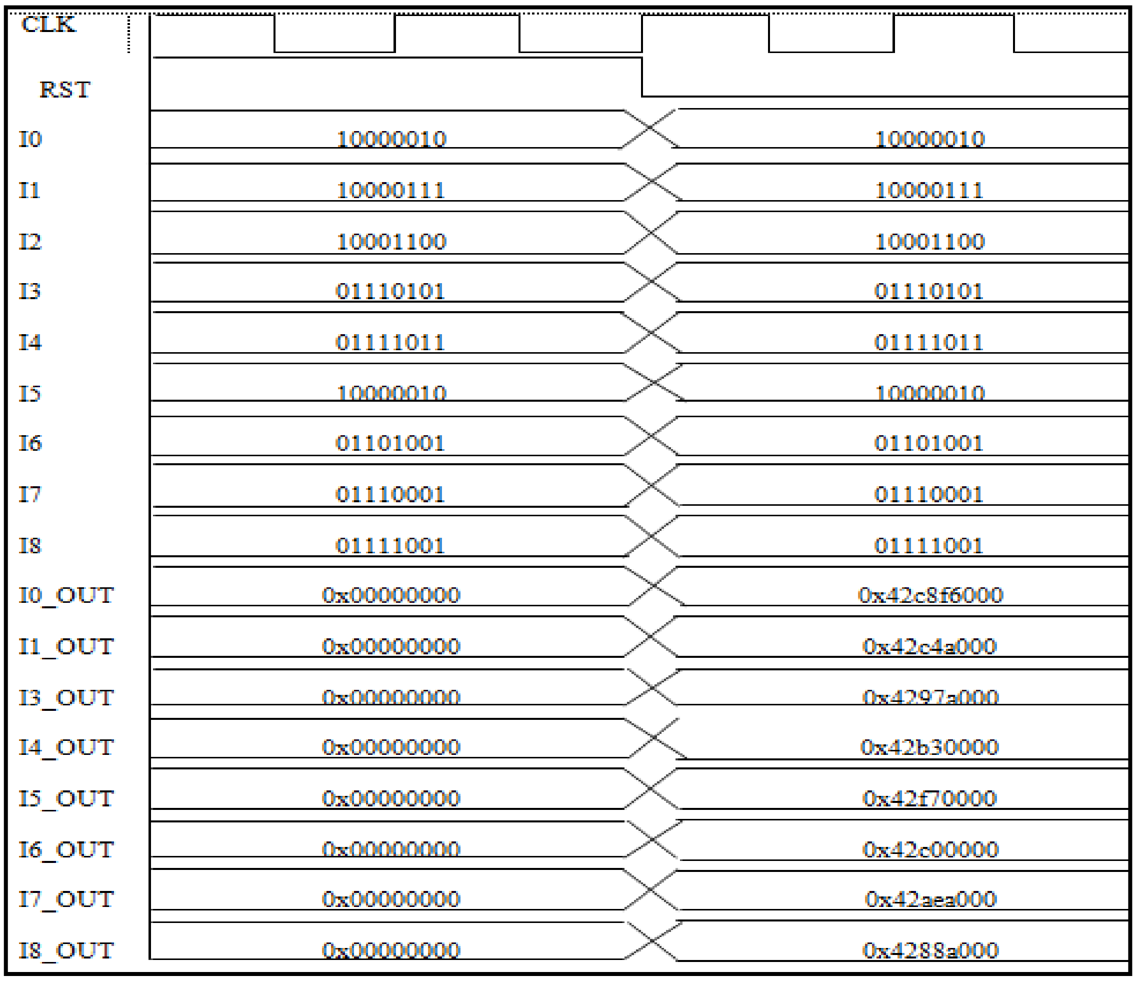


Figure 3.16: Timing Diagram of 2D Convolution Filter

3.4 IEEE754 FP Number System

The IEEE FP Arithmetic (IEEE754) standard is a universal technical standard for arithmetic operations on FP numbers that was set up and enforced by Institute of Electrical and Electronics Engineers in the year 1985. This standard overcomes various problems encountered during FP implementations like portability and reliability. Hence most designs that involve FP arithmetic are of IEEE754 standard.

The following rules are addressed in this standard.

- Arithmetic formats: Data in decimal, binary and FP format include finite numbers (subnormal and signed numbers), infinite numbers and not a number data (NaNs).
- Interchange formats: These are the formats like decimal and binary in which the FP data can be communicated in the most effective and efficient form.
- Rounding rules: One of the important properties that need to be satisfied during FP conversion process.
- Operations: Trigonometric functions, arithmetic and other operations in arithmetic number system.
- Exception Handling: In case of any exceptional conditions (like overflow or underflow, divide by zero, *etc*) indication flag is raised.

There are several FP interchange formats available as shown in the table below.

Table 3.9: FP Interchange Formats

Interchange Format	**Precision**	**Base**	**Exponent bits**	**Significand bits**	**Exponent Bias**
Binary16	Half precision	2	5	11	15
Binary32	Single	2	8	24	127
Binary64	Double	2	11	53	1023
Binary128	Quadruple	2	15	113	16383
Binary256	Octuple	2	19	237	262143
Decimal32	-	10	7.58	7	101
Decimal64	-	10	9.58	16	398
Decimal128	-	10	13.58	34	6176

The data encoded using IEEE754 FP format has three main specifications namely, base *b* – binary or decimal, Precision *p* and exponent *e* ranging from e_{min} to e_{max} where $e_{min} = 1 - e_{max}$.

Any floating point format comprises a sign bit, exponent and significand. Sign indicates the sign of a given number, Exponent gives the exponent value, and Significand is one hidden bit along with mantissa.

3.4.1 Single Precision Floating Point Format

In our design we use single precision floating point number format where a FP number is represented by 32 bits. A floating point number of single precision is composed of three fields as seen in Figure 3.17.

- Sign bit S: The sign field indicates a given FP number is positive if the value of S is 0 and negative if S is 1. It is 1 bit wide.
- Biased exponent (E = e + bias): This gives us a biased exponent value *i.e.* 127. It is 8 bit wide.
- Mantissa M: The fractional part of the number without considering a hidden bit. It is 23 bit wide together constituting 32 bits.

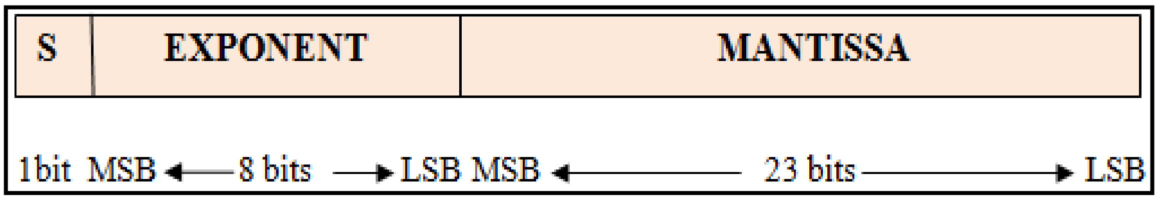


Figure 3.17: IEEE Single Precision Floating Point Format

The numbers in FP32 format are classified as follows:

- Normalized numbers: The bias is $2^7–1 = 128 − 1 = 127$, the range of the exponent is [−126:127], while its binary value is in the range [1:254]. The numbers that satisfy these conditions belong to the class of normalized numbers.
- Denormalized numbers: The exponent is -126, while its binary value is 0. Such numbers belong to this category.

- Infinities and NaN: These special representations have a binary value of $2^8 - 1 =$ 256-1=255 for the exponent (all ones).

3.5.2 Binary to Single Precision FP Conversion

- Given a number, consider the absolute value of a number for conversion from binary to single precision FP format.

-3.75 = |-3.75| = 3.75

- Convert the number into binary format.
- Hence first consider the fractional part and perform the conversion as shown below.

$$0.75 \times 2 = 1.5 \quad 1$$

$$0.5 \times = 1 \quad 1$$

Therefore, $(0.75)_{10}$ =$(11)_2$.

- Now consider the decimal part for conversion. Represent the decimal part in the binary format *i.e.* $(3)_{10} = (11)_2$.
- After conversion the number takes the form $(3.75)_{10} = (11.11)_2 = (11.11) \times 2^0$.
- Normalize the number in binary format as follows.

$$(3.75)_{10} = (1.111) \times 2^1.$$

- Here, to represent the number in FP format following considerations are made. Sign=1, Mantissa (unadjusted) = (1.111), Exponent (unadjusted) = 1.
- Exponent = Exponent (unadjusted) + Bias Exponent = 1+ $(127)_{10} = (128)_{10}$.
- Exponent= $(10000000)_2$.
- Mantissa=11100000000000000000000.
- Sign=1 (as the number is negative).
- Finally, single precision FP format of 3.75=1 10000000 11100000000000000000000.

3.5.3 Floating Point Signed Addition Algorithm

Two operands say, N1 and N2 are required to compute their successive addition or subtraction. For that E1 and E2 are the exponents, M1 and M2 are the mantissas and Sign1 and Sign2 are the signs of the operands N1 and N2 respectively. Mantissa together with the hidden bit is called significand which will be of 24bits. The significand of N1 and N2 is given by S1 and S2 respectively. The resulting number's sign bit is represented by Sign, exponent by E, mantissa by M and significand by S. The flow of the algorithm is as follows**.**

Compare the operands N1 and N2. If Sign1 and Sign2 are same then perform addition else perform subtraction.

Case 1: Sign1 and Sign2 are same and hence addition operation needs to be performed.

- Compare the absolute value (only exponent and significand) of N1 and N2.

Case 1.1: If M1 and M2, E1 and E2 are equal.

- Retain M1 as the resulting mantissa M and Sign1 as the resulting sign Sign. Add E1 by 1 *i.e.* E1+1 which will be the resultant exponent E. Table 3.10 below shows some of the operands of this type.

Table 3.10: Operands N1 and N2 are equal.

INPUTS	**SUM**

A= -1.73= 0 01111111 10111010111000010100100
B= -1.73= 0 01111111 10111010111000010100100
	SUM= -3.46= 0 10000000 10111010111000010100100

A= 3.44= 0 10000000 10111000010100011110110
B= 3.44= 0 10000000 10111000010100011110110
	SUM= 6.88 = 0 10000001 10111000010100011110110

Case 1.2: If absolute value of N1 > N2 *i.e.* |N1|>|N2|

- Make the resultant E = E1 tentatively, as the absolute value of N1 > N2.
- Find the Ediff = E1 - E2.
- Right shift S2 by Ediff times.
- Add significand of N1 with that of N2.
- Check if the 24th bit *i.e.* hidden bit of resultant significand S is 1.
- If it is 1, then the significand S is in the normalized form.
- Now consider the 23 bits excluding the hidden bit from significand S and this will be the final mantissa M and no changes need to be made to the tentative exponent E which will be the final exponent.

- If it is not 1, left shift the leading data (*i.e.* 0s) until the hidden bit of significand S becomes 1.
- Keep the count of 0s *i.e.* number of times the data was left shifted and subtract that count from the tentative exponent E. Now the resulting value will be final exponent E.
- Now consider the 23 bits excluding the hidden bit from significand S and this will be the final mantissa M.
- Sign will be the sign of operand N1. Table 3.11 shows the operands that belong to this category.

Table 3.11: |N1| > |N2| with Same Signs.

INPUTS	**SUM**

A = 2 = 0 10000000 00000000000000000000000
B = 1 = 0 01111111 00000000000000000000000
	SUM= 3 = 0 10000000 10000000000000000000000

A= -9.98 = 1 10000010 00111111010111000010100
B= -0.06 = 1 01111010 11101011100001010001111
	SUM = -10.04 = 1 10000010 01000001010001111010111

Case 1.3: If absolute value of N2 > N1.

- Make the resultant E = E2 tentatively, as the absolute value of N2 > N1.
- Find the Ediff = E2 - E1.
- Right shift S1 by Ediff times.
- Add significand of N2 with that of N1.
- Check if the 24th bit *i.e.* hidden bit of resultant significand S is 1.
- If it is 1, then the significand S is in the normalized form.
- Now consider the 23 bits excluding the hidden bit from significand S and this will be the final mantissa M and no changes need to be made to the tentative exponent E which will be the final exponent.
- If it is not 1, left shift the leading data (*i.e.* 0s) until the hidden bit of significand S becomes 1.
- Keep the count of 0s *i.e.* number of times the data was left shifted and subtract that count from the tentative exponent E. Now the resulting value will be final exponent E.

- Now consider the 23 bits excluding the hidden bit from significand S and this will be the final mantissa M.
- Sign will be the sign of operand N2. Table 3.12 shows the operands that belong to this category.

Table 3.12: |N2| > |N1| with Same Signs.

INPUTS	**SUM**

A = 1.2 = 00111111100110011001100110011010
B =1.92= 00111111111101011100001010001111
	SUM = 3.12 = 0 10000000 10001111010111000010100

A = 9 = 0 10000010 00100000000000000000000
B= 20 = 0 10000011 01000000000000000000000
	SUM = 29 = 0 10000011 11010000000000000000000

Case 2: Sign1 and Sign2 are different and hence subtraction operation needs to be performed.

- Compare the absolute value (only exponent and significand) of N1 and N2.

Case 2.1: If M1 and M2, E1 and E2 are equal.

- Make the entire result directly as 0. Table 3.13 below shows some of the operands of this type.

Table 3.13: Operands N1 and N2 are equal with opposite signs.

INPUTS	**SUM**

A= -1.73 = 1 01111111 10111010111000010100100
B= +1.73= 0 01111111 10111010111000010100100
	SUM= 0 = 0 00000000 00000000000000000000000

A= +3.44= 0 10000000 10111000010100011110110
B= -3.44= 1 10000000 10111000010100011110110
	SUM= 0 = 0 00000000 00000000000000000000000

Case 2.2: If absolute value of N1 > N2.

- Make the resultant E = E1 tentatively, as the absolute value of N1 > N2.
- Find the Ediff = E1 - E2.
- Right shift S2 by Ediff times.

- Subtract absolute value of N2 from N1.
- Check if the 24th bit *i.e.* hidden bit of resultant significand S is 1.
- If it is 1, then the significand S is in the normalized form.
- Now consider the 23 bits excluding the hidden bit from significand S and this will be the final mantissa M and no changes need to be made to the tentative exponent E which will be the final exponent.
- If it is not 1, left shift the leading data (*i.e.* 0s) until the hidden bit of significand S becomes 1.
- Keep the count of 0s *i.e.* number of times the data was left shifted and add that count to the tentative exponent E. Now the resulting value will be final exponent E.
- Now consider the 23 bits excluding the hidden bit from significand S and this will be the final mantissa M.
- Sign will be the sign of operand N1. Table 3.14 shows the operands that belong to this category.

Table 3.14: |N1| > |N2| with Different Signs.

INPUTS	**SUM**

A = 2.96= 0 10000000 01111010111000010100100
B= -1.23= 1 01111111 00111010111000010100100
	SUM = 1.73 = 0 01111111 10111010111000010100100

A= -5.46= 1 10000001 01011101011100001010010
B= 2.14 = 0 10000000 00101011100001010001111
	SUM = -3.12= 1 10000000 10001111010111000010100

Case 2.3: If Absolute Value of N2 > N1 with Different Signs

- Make the resultant E = E2 tentatively, as the absolute value of N2 > N1.
- Find the Ediff = E2 - E1.
- Right shift S1 by Ediff times.
- Subtract significand of N1 from that of N2.
- Check if the 24th bit *i.e.* hidden bit of resultant significand S is 1.
- If it is 1, then the significand S is in the normalized form.

- Now consider the 23 bits excluding the hidden bit from significand S and this will be the final mantissa M and no changes need to be made to the tentative exponent E which will be the final exponent.
- If it is not 1, left shift the leading data (*i.e.* 0s) until the hidden bit of significand S becomes 1.
- Keep the count of 0s *i.e.* number of times the data was left shifted and add that count to the tentative exponent E. Now the resulting value will be final exponent E.
- Now consider the 23 bits excluding the hidden bit from significand S and this will be the final mantissa M.
- Sign will be the sign of operand N2. Table 3.15 shows the operands that belong to this category.

Table 3.15: |N2| > |N1| with Different Signs.

INPUTS	**SUM**

A= 4.46 = 0 10000001 00011101011100001010010
B= -6.38= 1 10000001 10011000010100011110110
	SUM = -1.92 = 1 01111111 11101011100001010001111

A= -2.94=1 10000000 01111000010100011110110
B= 3.06 =0 10000000 10000111101011100001010
	SUM = 0.12 = 0 01111011 11101011100001010001111

CHAPTER 4

BLOCK LEVEL DESIGN VERIFICATION

CHAPTER 4

BLOCK LEVEL DESIGN VERIFICATION

Chapter Outline:

The proposed 2D convolution filter hardware architecture is designed in verilog HDL and simulated in Isim simulator from Xilinx and synthesis reports are generated using Cadence Encounter RTL Compiler targeted to 45nm slow_vdd1v0_basicCells.lib technology library (Discussed in Appendix-A). In this chapter, RTL schematics and simulation waveforms generated to verify the functionality of the various basic building blocks constituting the design is discussed.

4.1 Simulation Results of GKROM

Top level and detailed RTL schematics of GKROM are shown in Figure 4.1 and Figure 4.2 respectively.

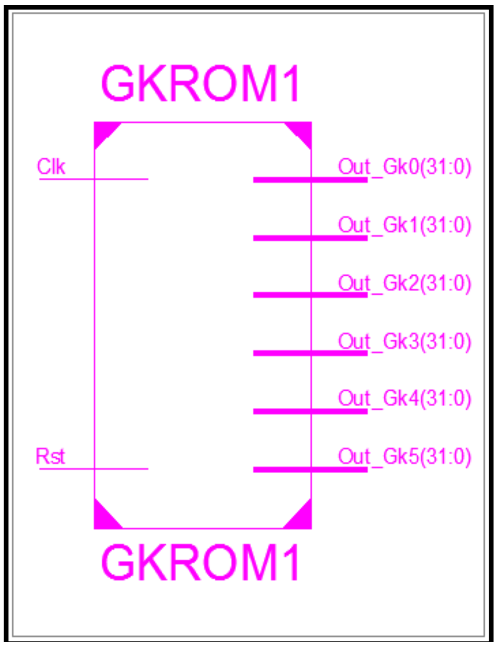


Figure 4.1: Top Level RTL Schematic of GKROM

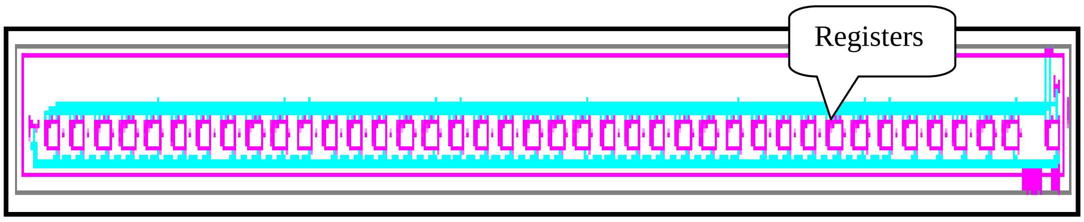


Figure 4.2: RTL Schematic of GKROM

Stimuli: Clk, Rst.

Response: Out_GK0 – Out_GK5

As seen in Figure 4.3 Marker 1 indicates Rst=1 and when reset is high the output is 0 as indicated by Marker 2. When reset Rst=0 indicated by Marker 3 the values stored in GKROM reflect in the output as indicated by Marker 4.

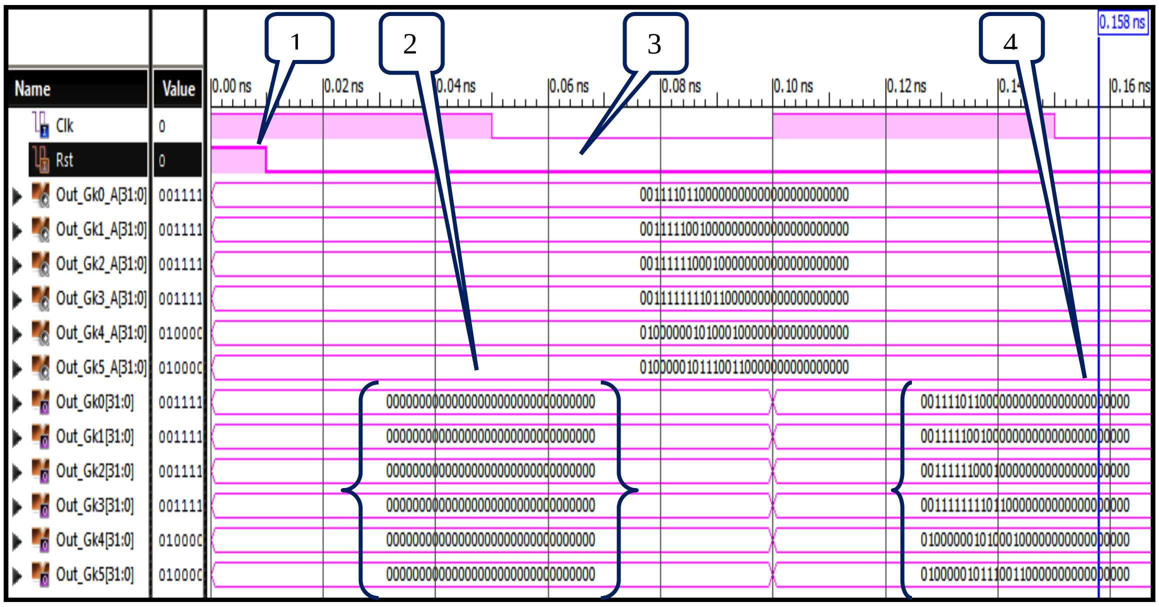

1
2
3
4
0.158 ns
Name
Value
0.00 ns
0.02 ns
0.04 ns
0.06 ns
0.08 ns
0.10 ns
0.12 ns
0.14
0.16 ns
Clk 0
Rst 0
Out_Gk0_A[31:0] 001111 00111101100000000000000000000000
Out_Gk1_A[31:0] 001111 00111110010000000000000000000000
Out_Gk2_A[31:0] 001111 00111111000100000000000000000000
Out_Gk3_A[31:0] 001111 00111111110110000000000000000000
Out_Gk4_A[31:0] 010000 01000000101000100000000000000000
Out_Gk5_A[31:0] 010000 01000001011100110000000000000000
Out_Gk0[31:0] 001111 00000000000000000000000000000000 00111101100000000000000000000000
Out_Gk1[31:0] 001111 00000000000000000000000000000000 00111110010000000000000000000000
Out_Gk2[31:0] 001111 00000000000000000000000000000000 00111111000100000000000000000000
Out_Gk3[31:0] 001111 00000000000000000000000000000000 00111111110110000000000000000000
Out_Gk4[31:0] 010000 00000000000000000000000000000000 01000000101000100000000000000000
Out_Gk5[31:0] 010000 00000000000000000000000000000000 01000001011100110000000000000000

Figure 4.3: Simulation Waveform of GKROM

4.2 Simulation Results of CROM

Top level and detailed RTL schematics of CROM are shown in Figure 4.4 and Figure 4.5 respectively.

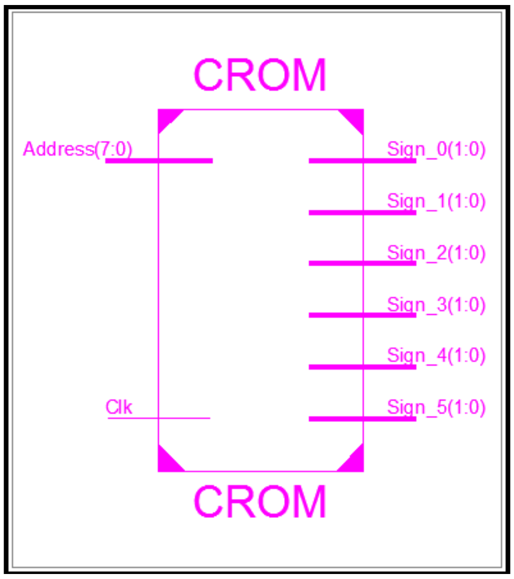


Figure 4.4: Top Level RTL Schematic of CROM

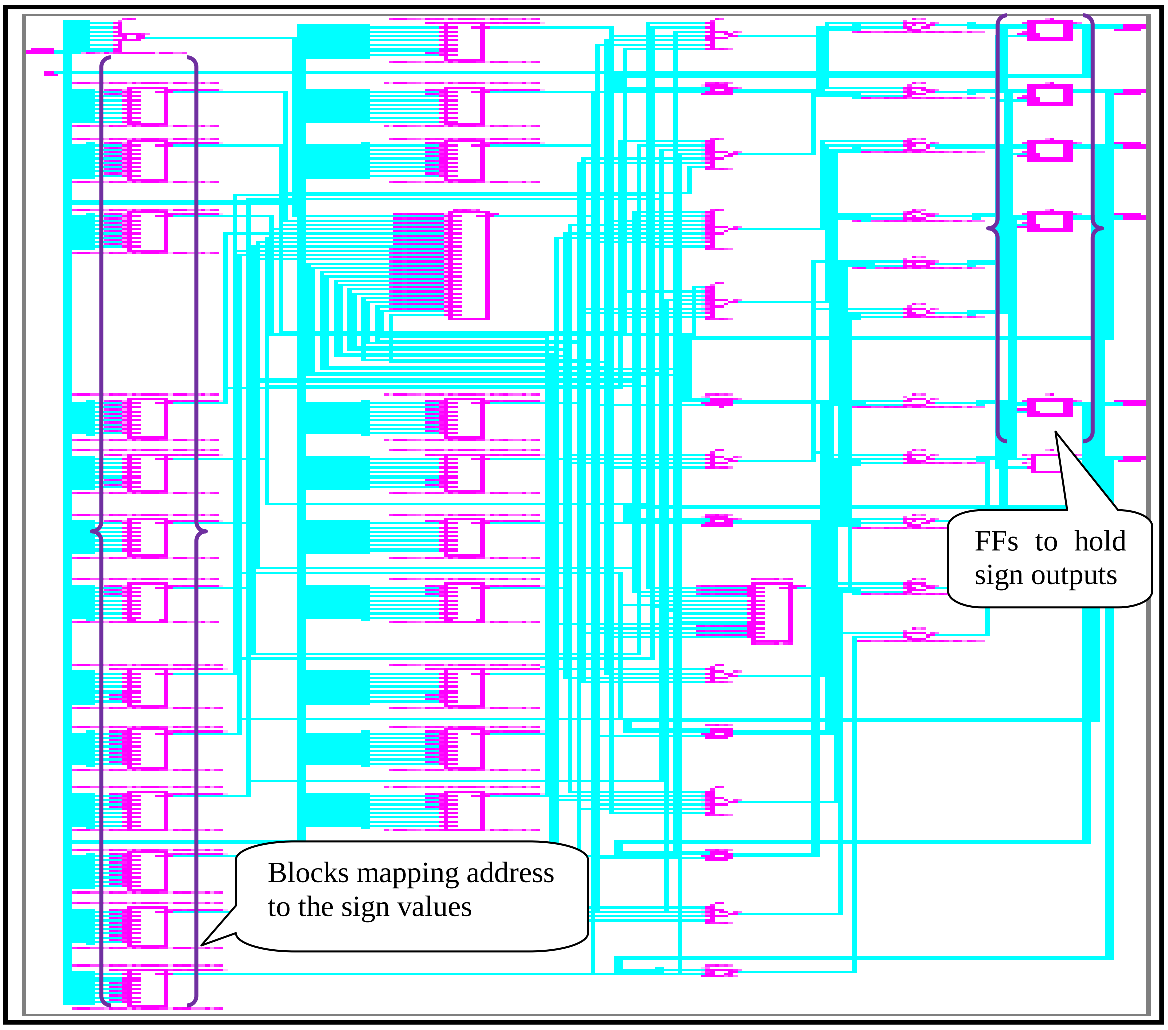


Figure 4.5: RTL Schematic of CROM

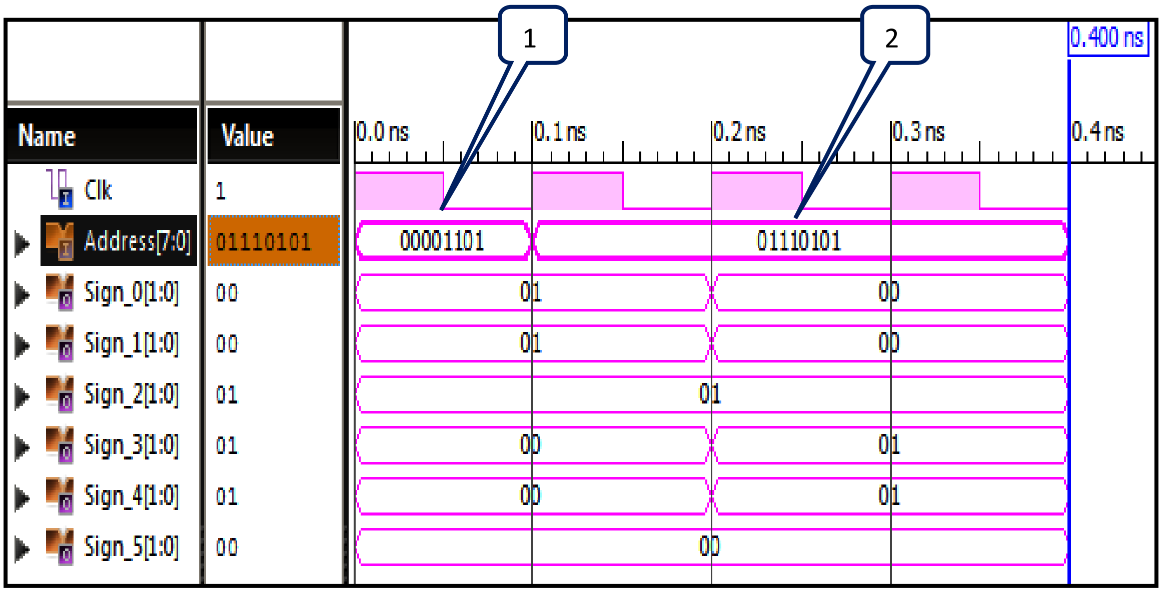


Figure 4.6: Simulation Waveform of CROM

Stimuli: Address, Clk

Response: Sign_0 - Sign_5

In Figure 4.6 Marker 1 indicates the first input address and Marker 2 indicates the second input address. With the rising clock edge the corresponding sign values, each 2bit wide appear as the output.

4.3 Verification Results of Multiplexer

Top level and detailed RTL schematics of Multiplexer are shown in Figure 4.7 and Figure 4.9 respectively.

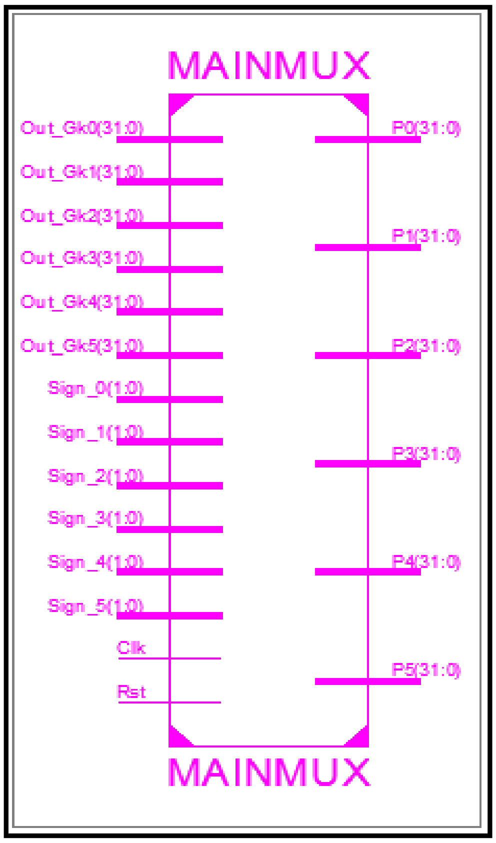
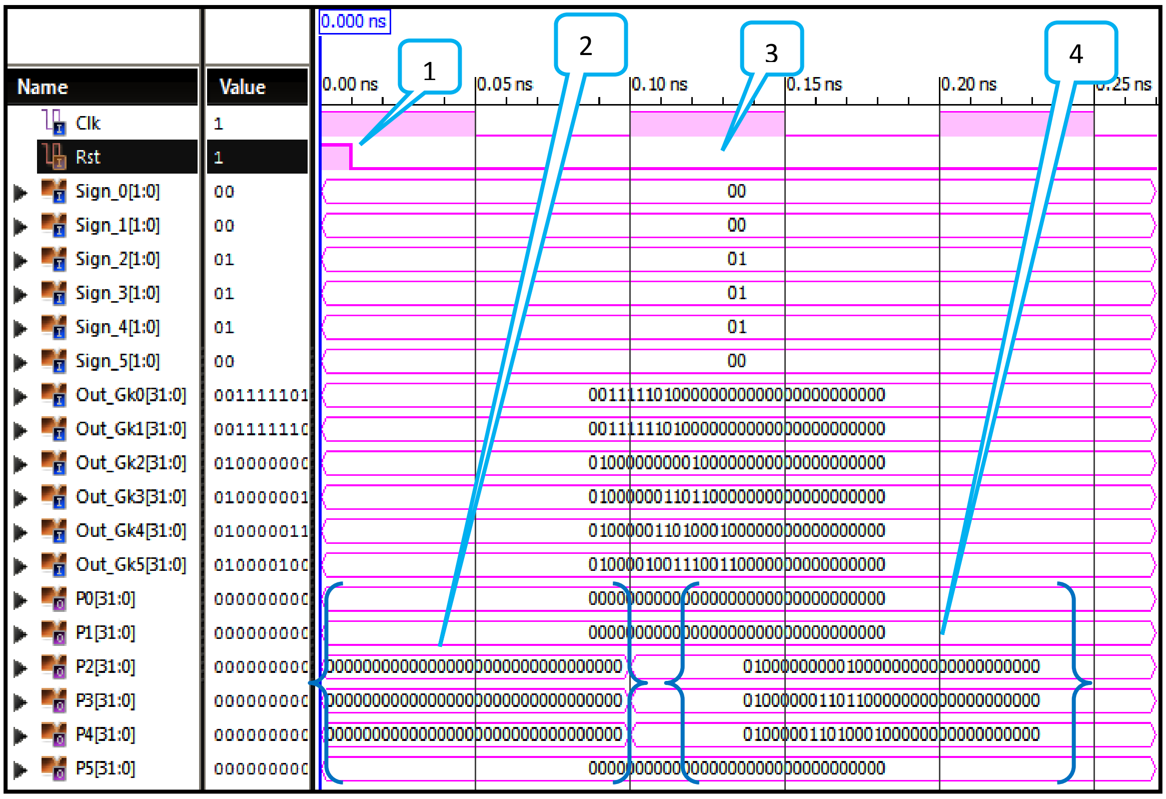


Figure 4.7: Top Level RTL Schematic of Multiplexer

Stimuli: Clk, Rst, Sign_0- Sign_5, Out_GK0- Out_GK5

Response: P0- P5

In Figure 4.8 Marker 1 indicates that the Rst is 0 and the corresponding output is indicated by Marker 2 where the output values are 0. Marker 3 indicates that the Rst is and during rising edge of the clock the change in the output values according to the input can be observed as indicated by Marker 4.



```
Name            Value       
Clk             1
Rst             1
Sign_0[1:0]     00          00
Sign_1[1:0]     00          00
Sign_2[1:0]     01          01
Sign_3[1:0]     01          01
Sign_4[1:0]     01          01
Sign_5[1:0]     00          00
Out_Gk0[31:0]   001111101   00111110100000000000000000000000
Out_Gk1[31:0]   001111110   00111111010000000000000000000000
Out_Gk2[31:0]   010000000   01000000001000000000000000000000
Out_Gk3[31:0]   010000001   01000000110110000000000000000000
Out_Gk4[31:0]   010000011   01000001101000100000000000000000
Out_Gk5[31:0]   010000100   01000010011100110000000000000000
P0[31:0]        000000000   00000000000000000000000000000000
P1[31:0]        000000000   00000000000000000000000000000000
P2[31:0]        000000000   00000000000000000000000000000000   01000000001000000000000000000000
P3[31:0]        000000000   00000000000000000000000000000000   01000000110110000000000000000000
P4[31:0]        000000000   00000000000000000000000000000000   01000001101000100000000000000000
P5[31:0]        000000000   00000000000000000000000000000000
```

Figure 4.8: Functionality Waveform of Multiplexer

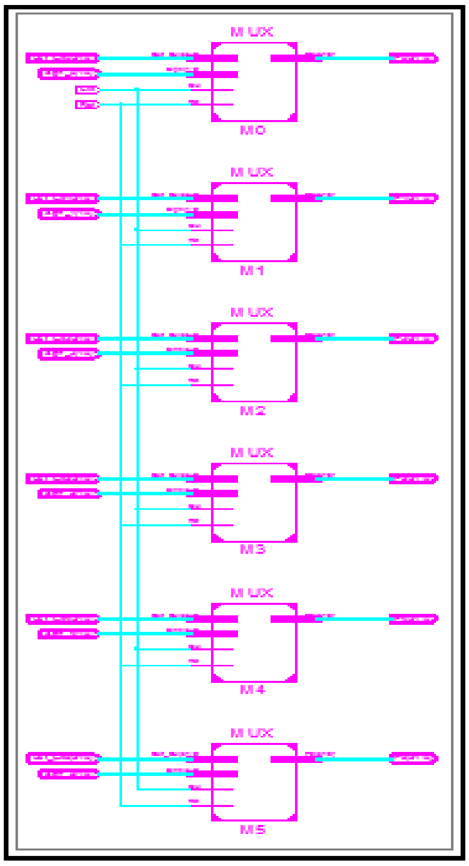


Figure 4.9: RTL Schematic of Multiplexer

4.4 Verification Results of FP32 Adder

Top level and detailed RTL schematics of Floating Point Adder are shown in Figure 4.10 and Figure 4.11 respectively.

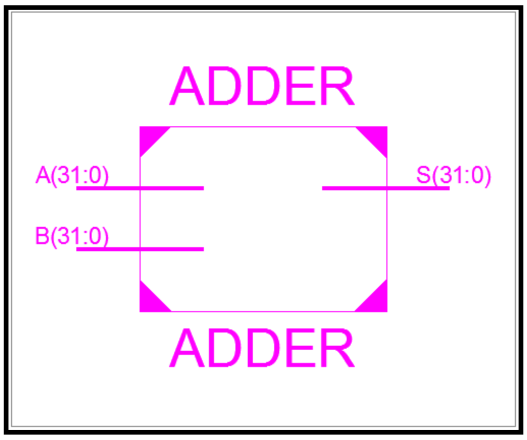


Figure 4.10: Top Level RTL Schematic of FP32 Adder

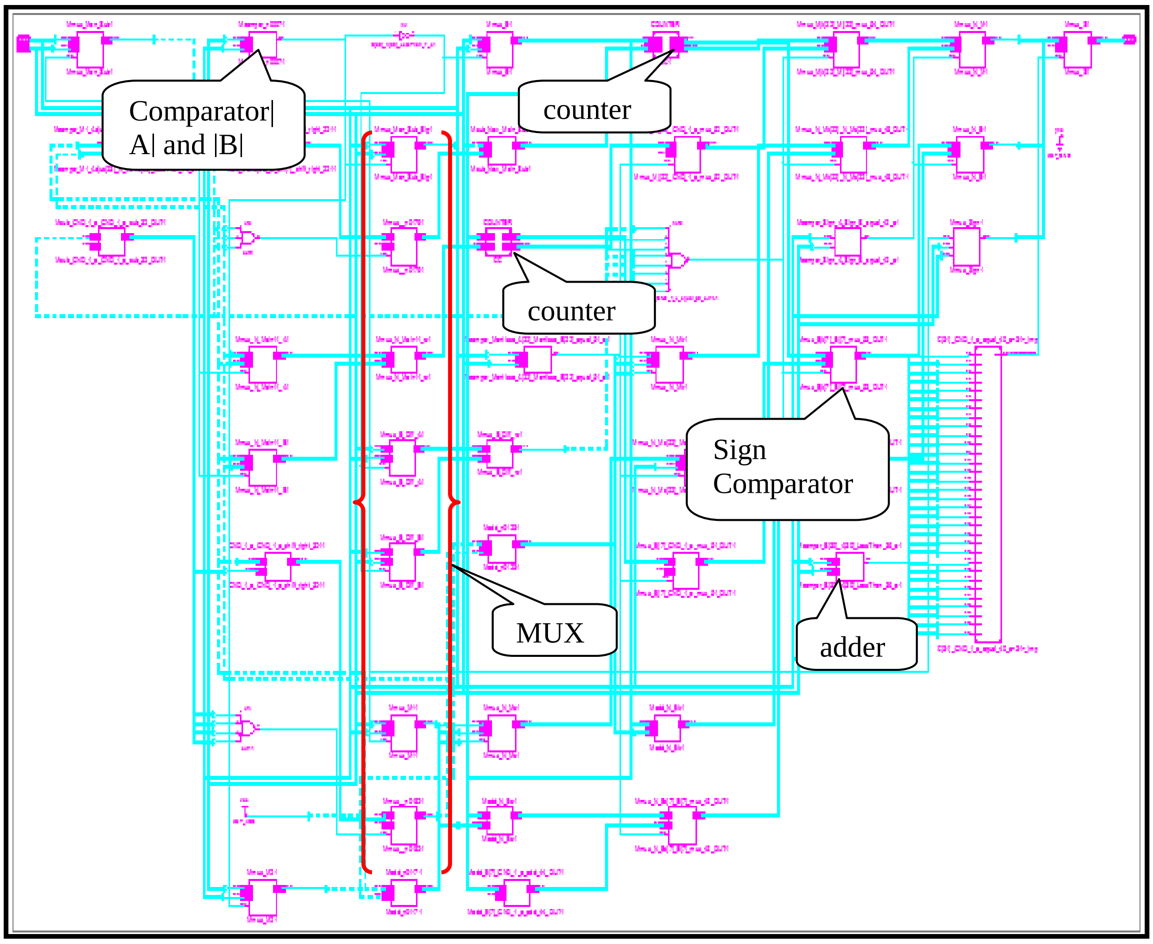


Figure 4.11: RTL Schematic of FP32 Adder

Stimuli: Operands A, B.

Response: Sum S.

As seen in Figure 4.12 Marker 1 indicates first set of inputs A= -1.73, B= -1.73 and the corresponding output is S= -3.46. Marker 2 indicates second set of inputs A= 3.44, B= 3.44 and the corresponding output is S= 6.88.

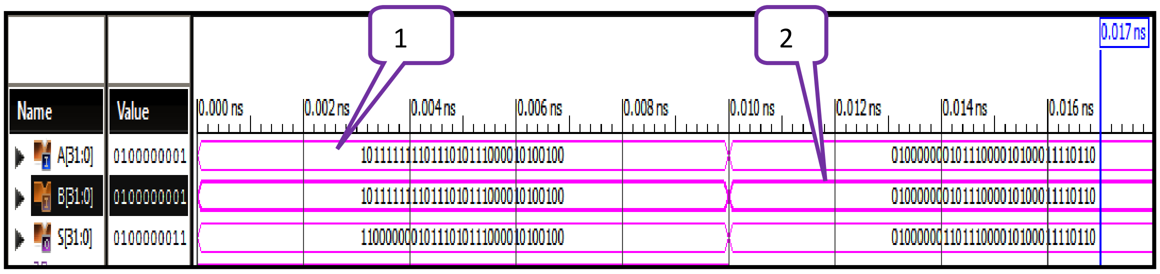


Figure 4.12: FP32 Addition Simulation Waveform for A=B

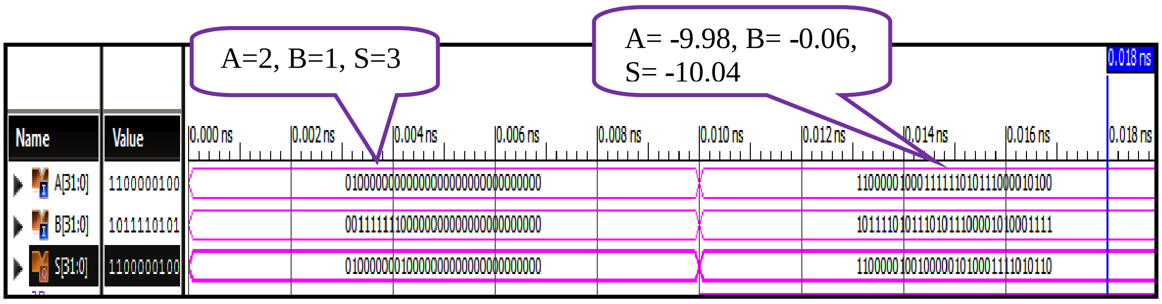


Figure 4.13: FP32 Addition Simulation Waveform for |A| > |B| with Same Signs

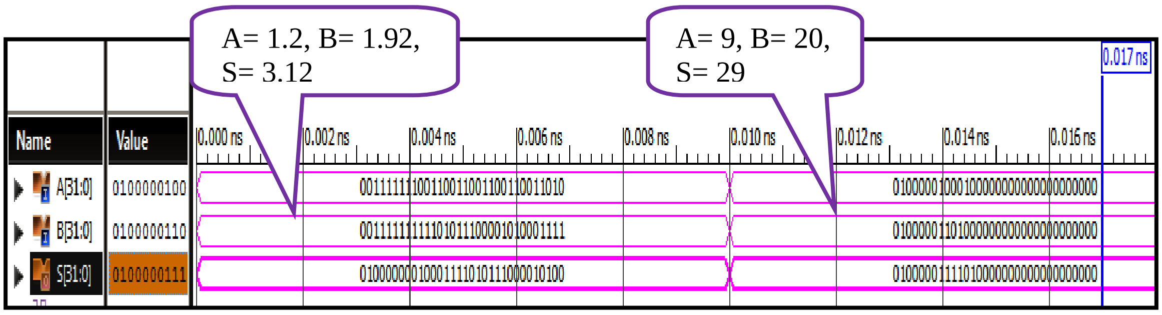


Figure 4.14: FP32 Addition Simulation Waveform for |A| < |B| with Same Signs

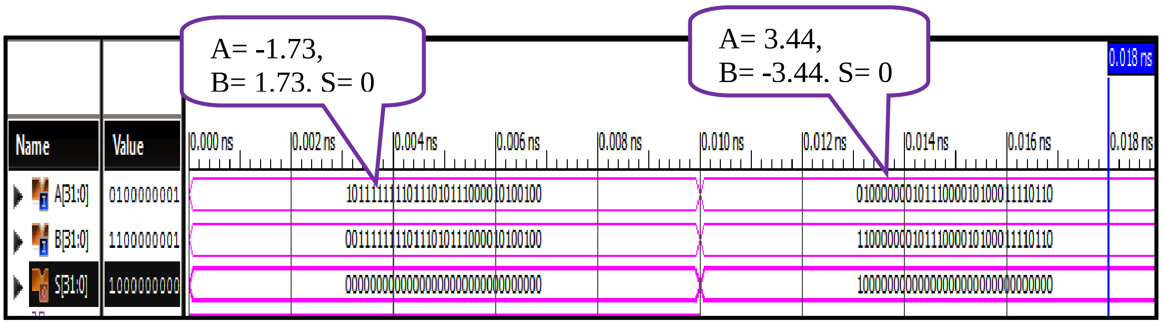
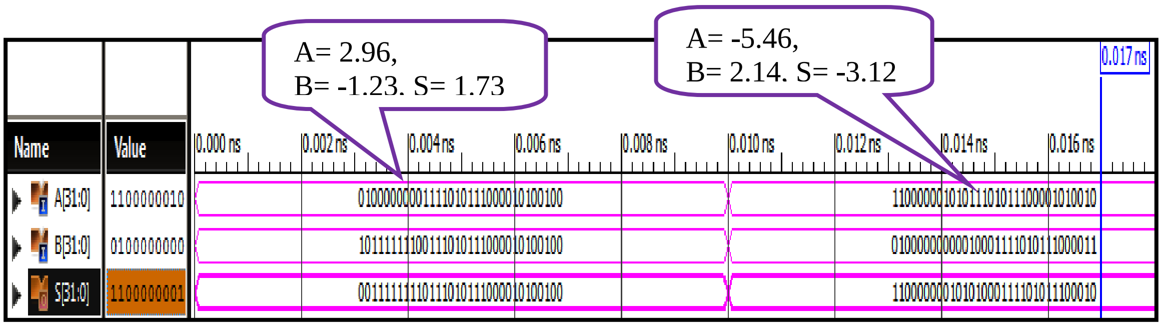
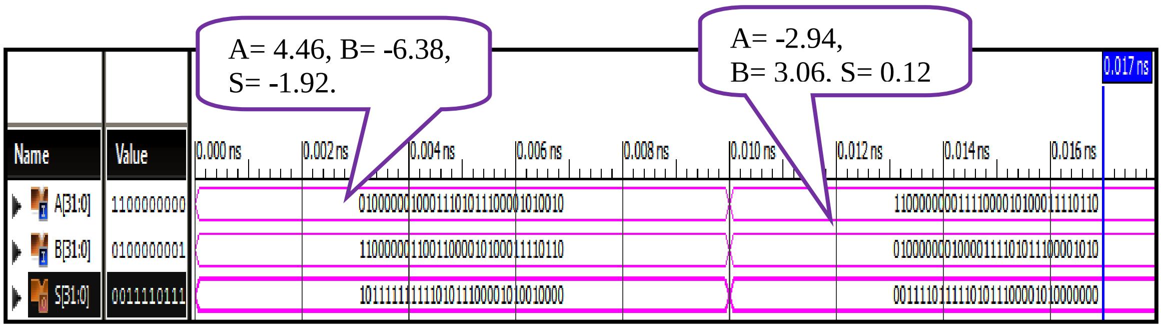


Figure 4.15: FP32 Addition Simulation Waveform for |A|=|B| with Different Signs



```
A[31:0]  1100000010  01000000011110101110000101001000   11000001010111010111000010100100
B[31:0]  0100000000  10111111100111010111000010100100   01000000000100011110101110000110
S[31:0]  1100000001  00111111110111010111000010100100   11000000101010001111010111000100
```

Figure 4.16: FP32 Addition Simulation Waveform for |A|>|B| with Different Signs



```
A[31:0]  1100000000  01000001000111010111000010100100   11000000011110000101000111101100
B[31:0]  0100000001  11000001100110000101000111101100   01000000100001111010111000010100
S[31:0]  0011110111  10111111111010111000010100100000   00111101111010111000010100000000
```

Figure 4.17: FP32 Addition Simulation Waveform for |A|<|B| with Different Signs

CHAPTER 5

RESULTS AND DISCUSSION

CHAPTER 5

RESULTS AND DISCUSSIONS

Chapter Outline:

The proposed 2D convolution filter hardware architecture is designed in verilog HDL and simulated in Isim simulator from Xilinx and synthesis reports are generated using Cadence Encounter RTL Compiler targeted to 45nm slow_vdd1v0_basicCells.lib technology library. In this chapter, we present RTL schematics, simulation results and synthesis reports of integrated top level modules of the design.

5.1 Verification of Integrated Modules

5.1.1 Verification of FP32 Multiplier

Detailed and Top level RTL schematics of FP32 Multiplier are shown in Figure 5.1 and Figure 5.2 respectively.

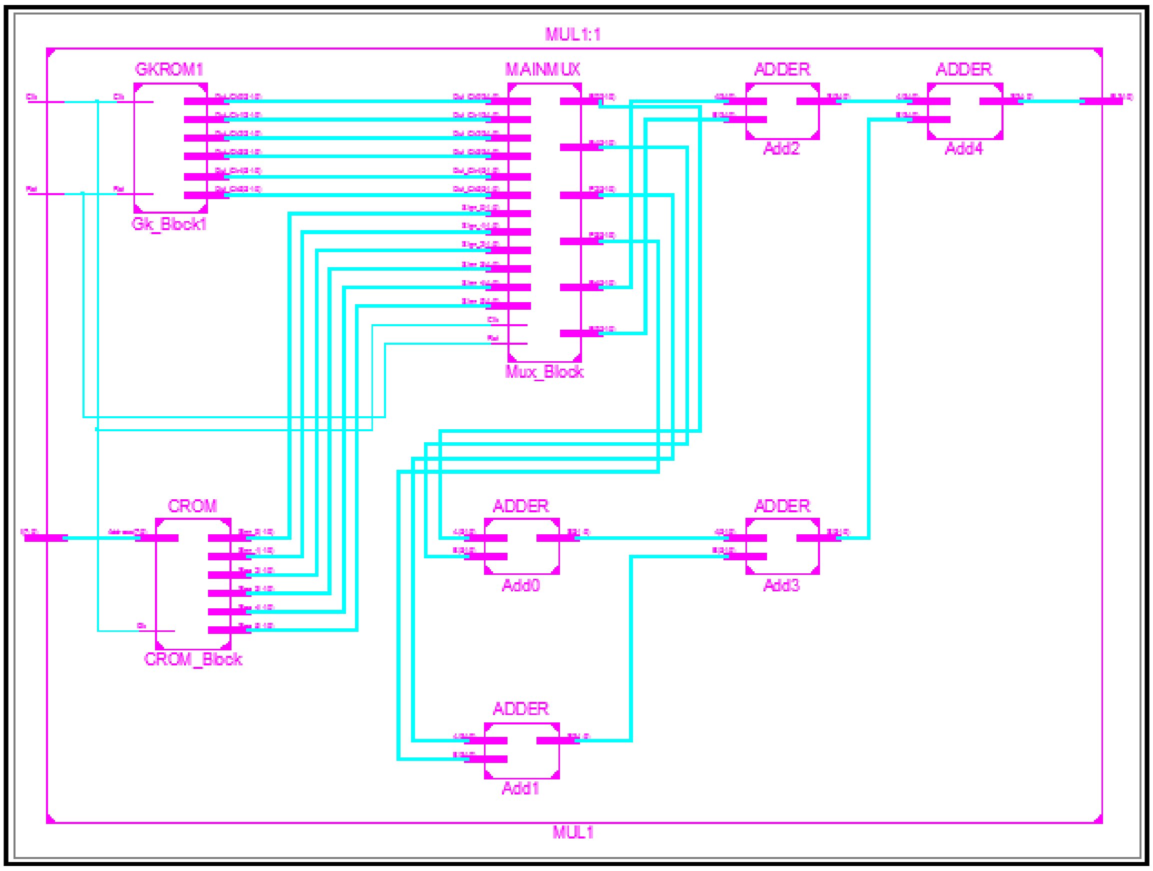


Figure 5.1: RTL Schematic of FP32 Multiplier

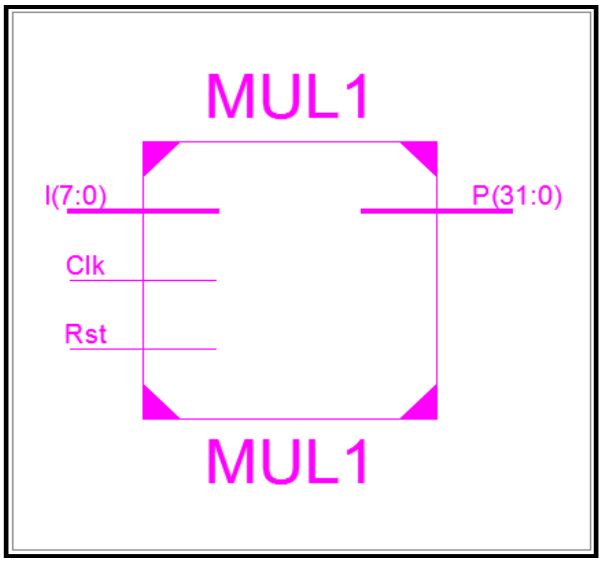


Figure 5.2: Top Level RTL Schematic of FP32 Multiplier

Stimuli: Clk, Rst, I.

Response: P.

One of the inputs to the multiplier is I=12 as seen in Figure 5.3 while the other is 0.0625, the one stored in GKROM. Callout 1 indicates reset is high and the corresponding product P is 0 as shown by Callout 2. Callout 3- Reset is low. Callout 4- When reset is low multiplication result P= 0.75 appears as the output.

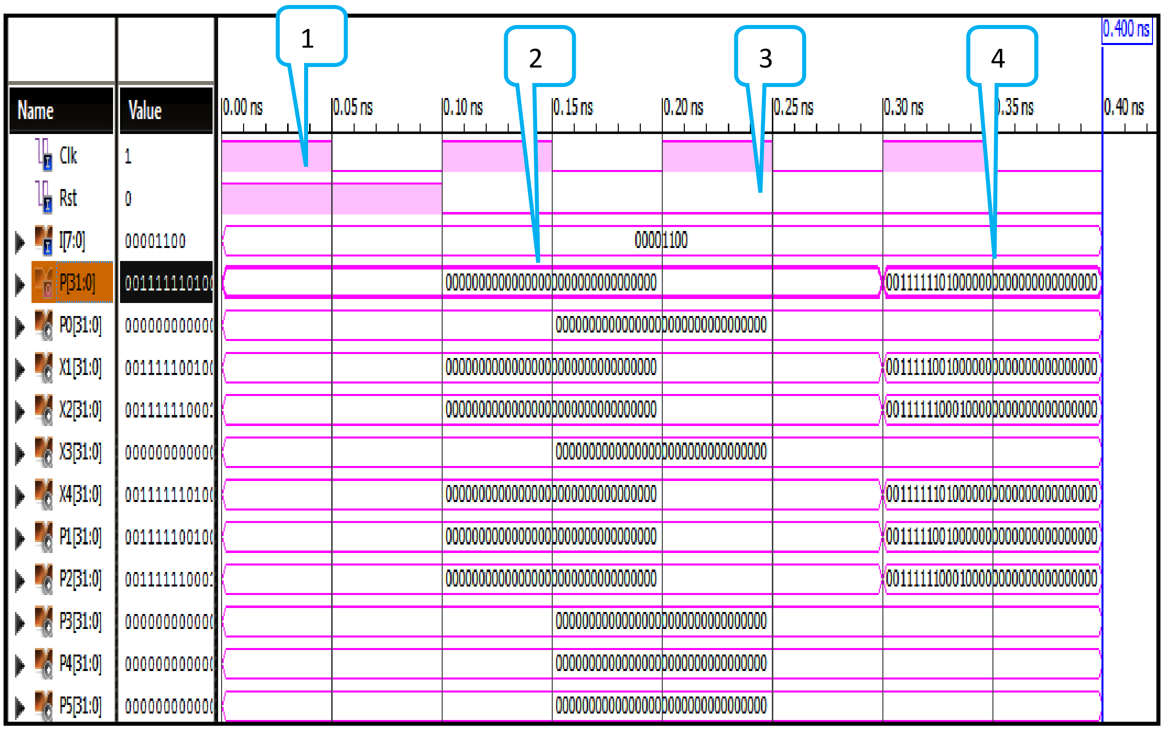


Figure 5.3: FP32 Multiplication Functionality Waveform

5.1.2 Verification of Convolution Unit

Top level and detailed RTL schematics of Convolution module are shown in Figure 5.4 and Figure 5.5 respectively.

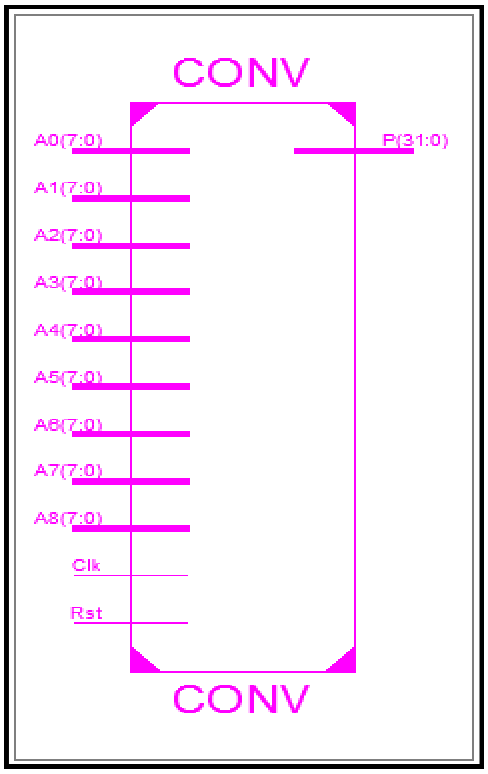


Figure 5.4: Top Level RTL Schematic of Convolution Module

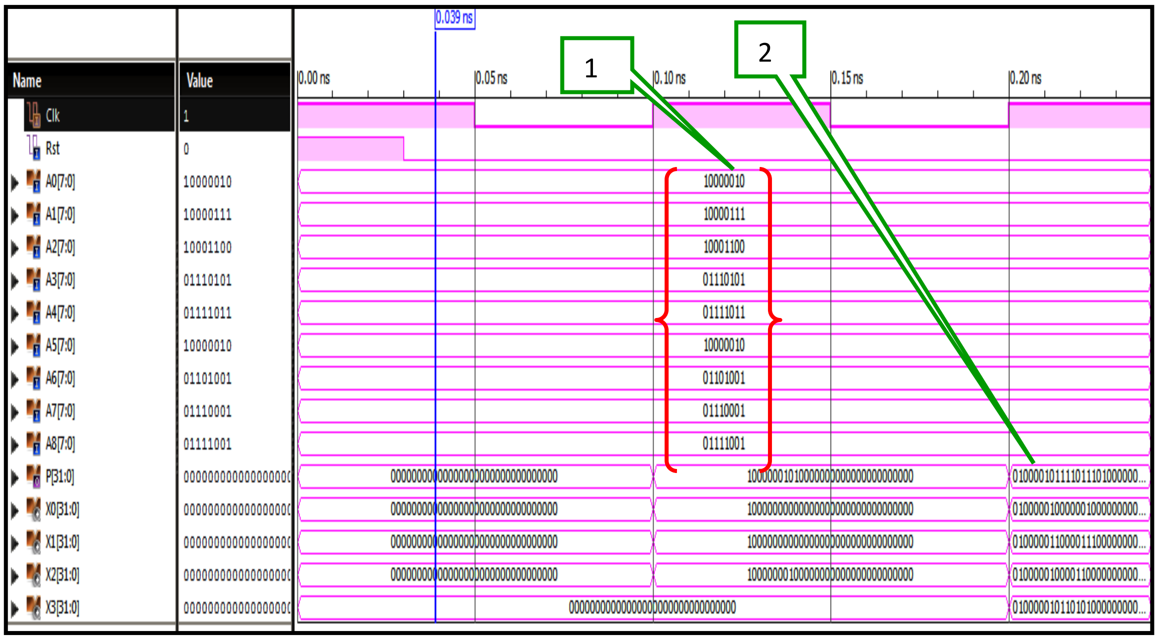


Figure 5.5: Convolution Functionality Waveform for One Pixel

Stimuli: Clk, Rst, A0-A8.

Response: P.

During rising edge of the clock and when reset is high changes in the output can be observed as seen in Figure 5.5. Callout 1 is used to highlight the set of input values provided as per the eq^n 3.5. These inputs with the kernel values (eq^n 3.6) undergo convolution operation and the result is a convoluted pixel value indicated by Callout 2.

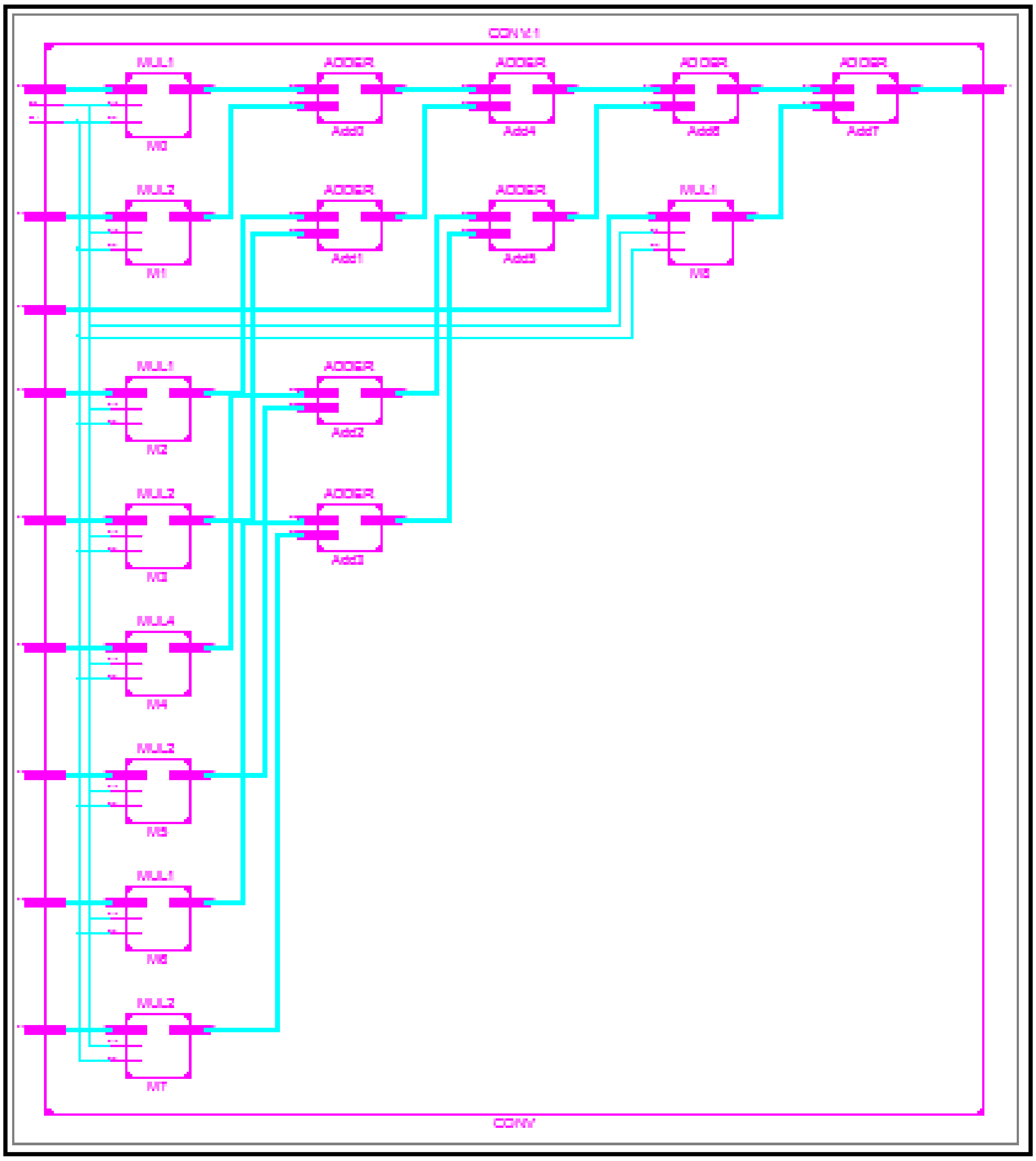
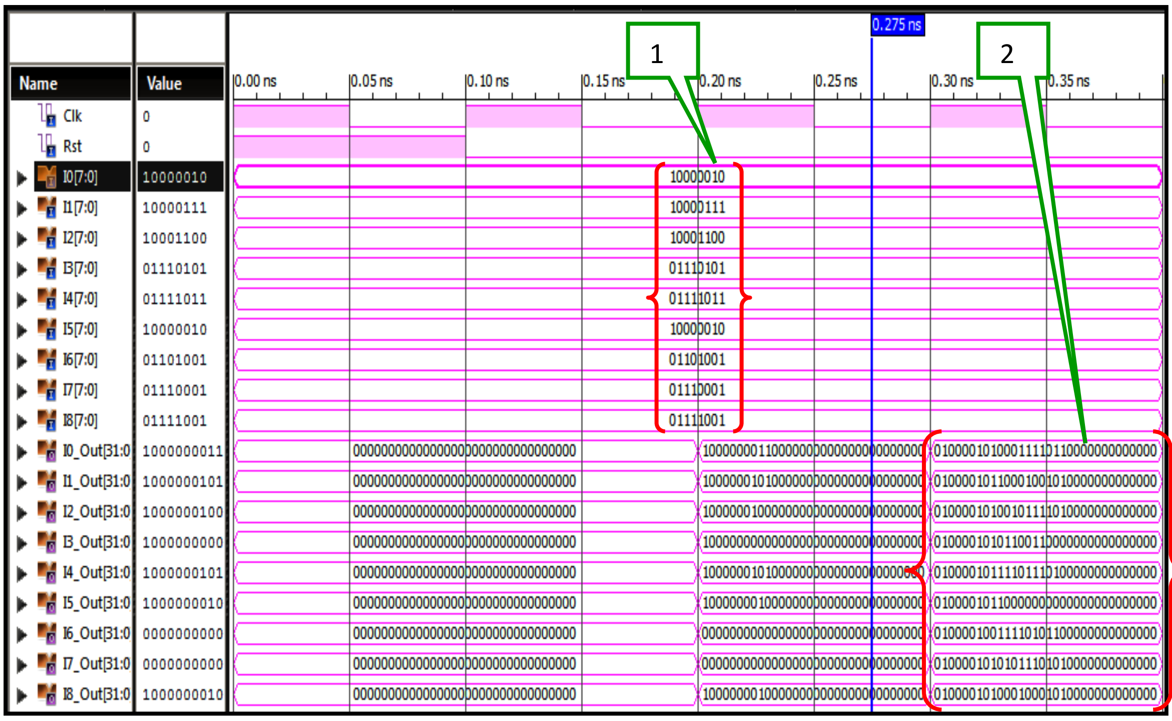


Figure 5.6: RTL Schematic of Convolution Architecture



Name	Value
Clk	0
Rst	0
I0[7:0]	10000010
I1[7:0]	10000111
I2[7:0]	10001100
I3[7:0]	01110101
I4[7:0]	01111011
I5[7:0]	10000010
I6[7:0]	01101001
I7[7:0]	01110001
I8[7:0]	01111001
I0_Out[31:0	1000000011
I1_Out[31:0	1000000101
I2_Out[31:0	1000000100
I3_Out[31:0	1000000000
I4_Out[31:0	1000000101
I5_Out[31:0	1000000010
I6_Out[31:0	0000000000
I7_Out[31:0	0000000000
I8_Out[31:0	1000000010

Figure 5.7: 2D Convolution Filter Functionality Waveform

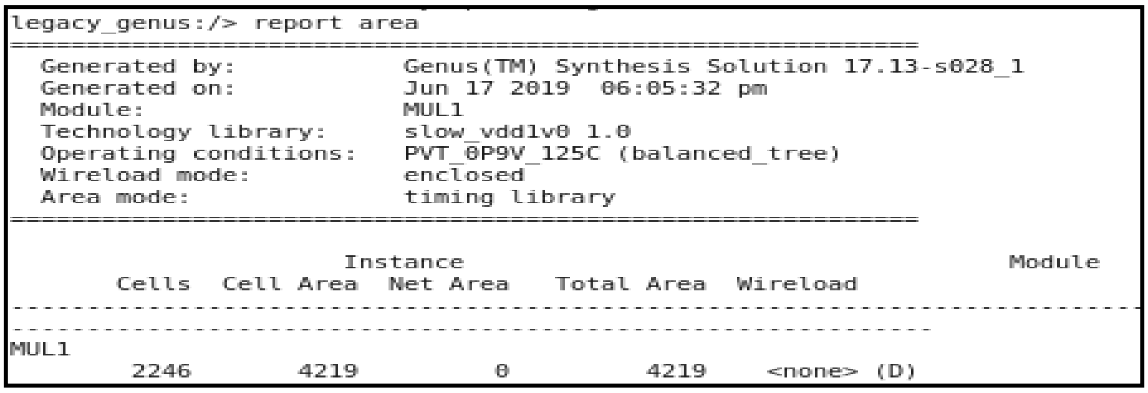
Stimuli: Clk, Rst, I0-I8.

Response: I0_OUT - I8_OUT.

Similar convolution operation is performed between the input values as pr the eq^n 3.5 highlighted by Callout 1 and the kernel values (eq^n 3.6). All the nine filtered pixel values are generated as highlighted by Callout 2.

5.2 Synthesis Report Generated from Cadence Encounter RTL Compiler Tool

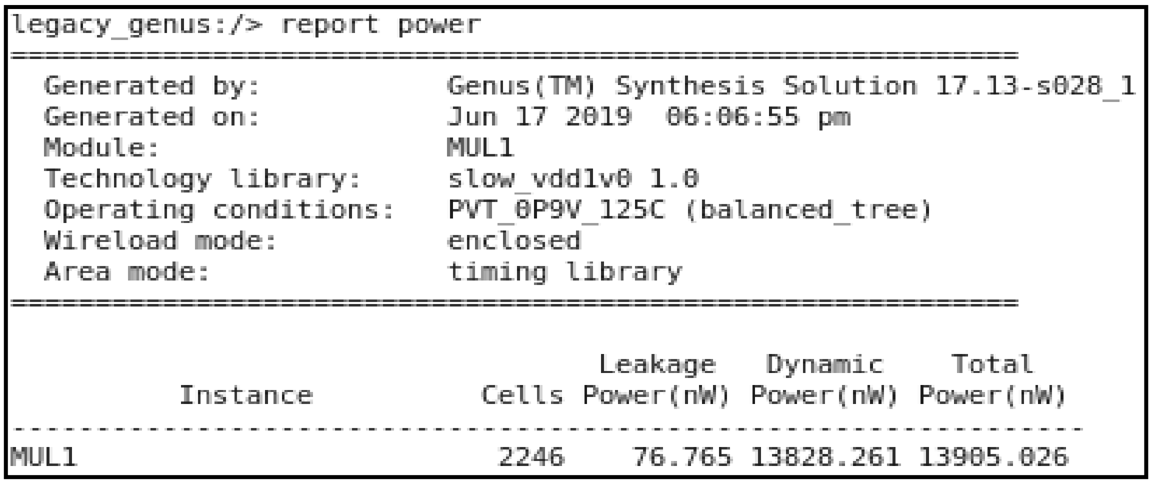
5.2.1 FP32 Multiplier



```
legacy_genus:/> report area
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 17 2019  06:05:32 pm
  Module:                 MUL1
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

                     Instance                                Module
       Cells  Cell Area  Net Area   Total Area  Wireload
---------------------------------------------------------------------------
-------------------------------------------------------------
MUL1
        2246       4219         0         4219     <none> (D)
```

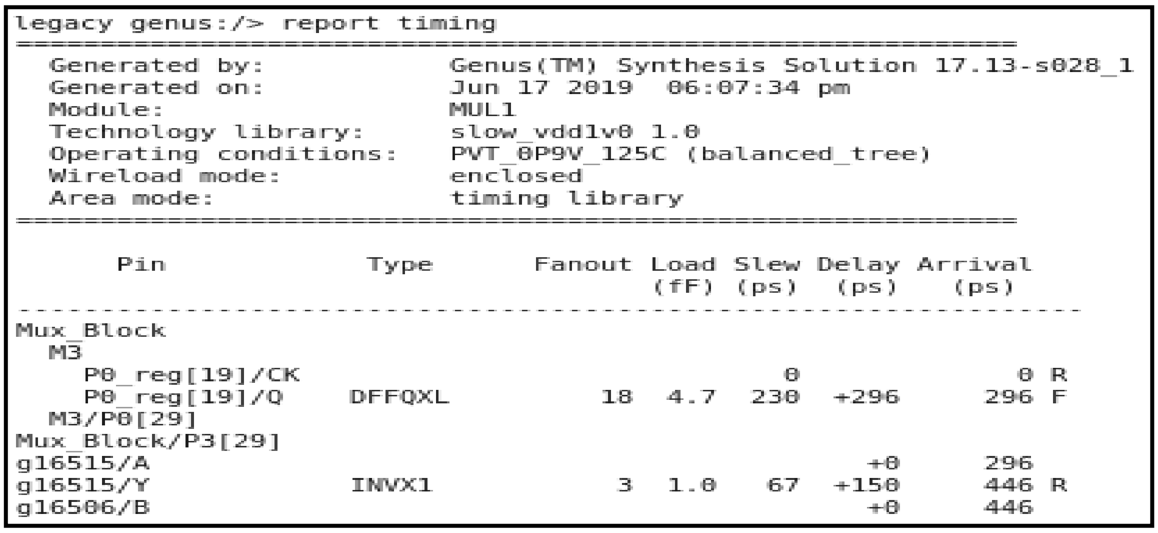
Figure 5.8: Area Report of Multiplier



```
legacy_genus:/> report power
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 17 2019  06:06:55 pm
  Module:                 MUL1
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

                                Leakage   Dynamic    Total
         Instance         Cells Power(nW) Power(nW) Power(nW)
---------------------------------------------------------------
MUL1                       2246    76.765 13828.261 13905.026
```

Figure 5.9: Power Report of Multiplier

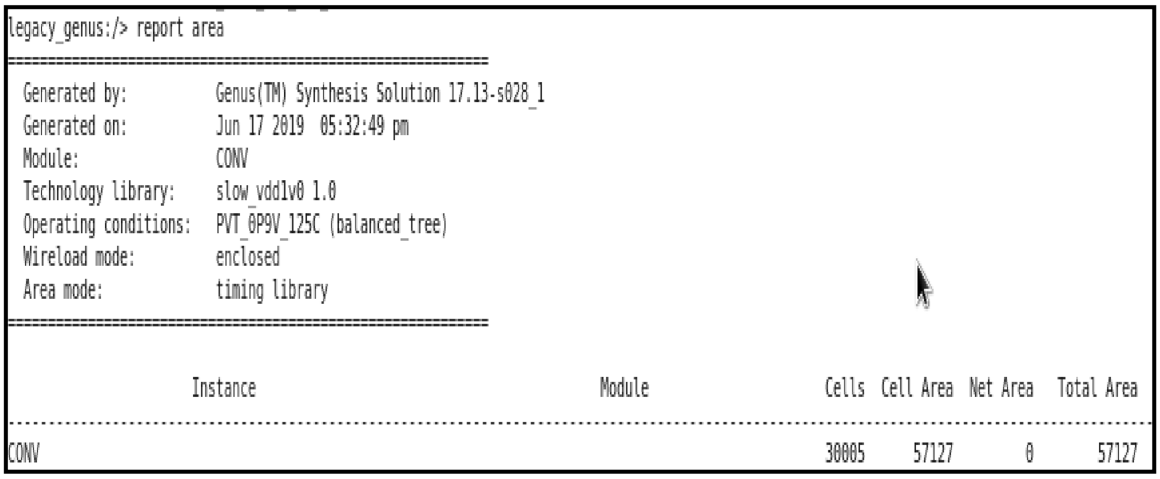


```
legacy genus:/> report timing
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 17 2019  06:07:34 pm
  Module:                 MUL1
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

      Pin            Type      Fanout Load Slew Delay Arrival
                                      (fF) (ps)  (ps)   (ps)
----------------------------------------------------------------
Mux_Block
  M3
    P0_reg[19]/CK                              0            0 R
    P0_reg[19]/Q    DFFQXL         18  4.7   230  +296     296 F
  M3/P0[29]
Mux_Block/P3[29]
g16515/A                                            +0     296
g16515/Y            INVX1           3  1.0    67  +150     446 R
g16506/B                                            +0     446
```

Figure 5.10: Timing Report of Multiplier

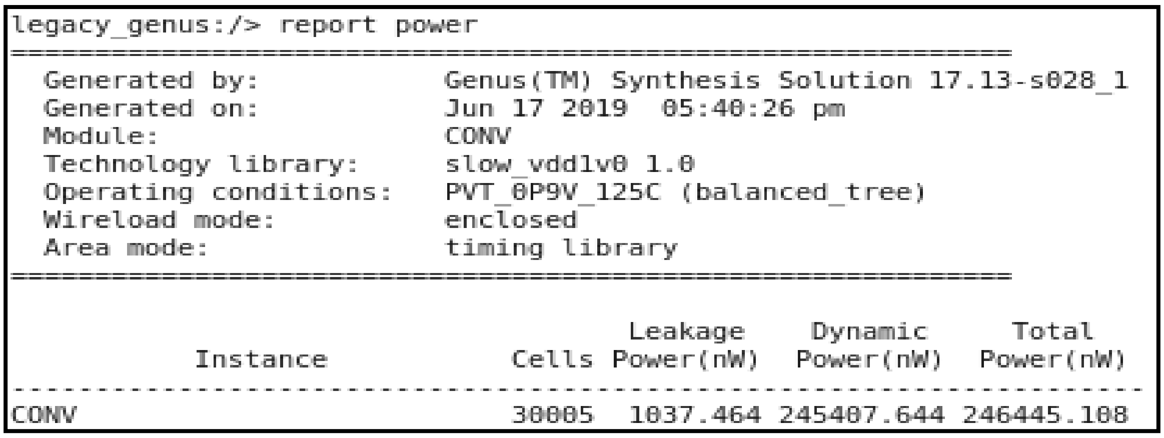
5.2.2 Convolution Module



```
legacy_genus:/> report area
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 17 2019  05:32:49 pm
  Module:                 CONV
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

            Instance                    Module          Cells  Cell Area  Net Area  Total Area
-----------------------------------------------------------------------------------------------
CONV                                                    30005      57127         0       57127
```

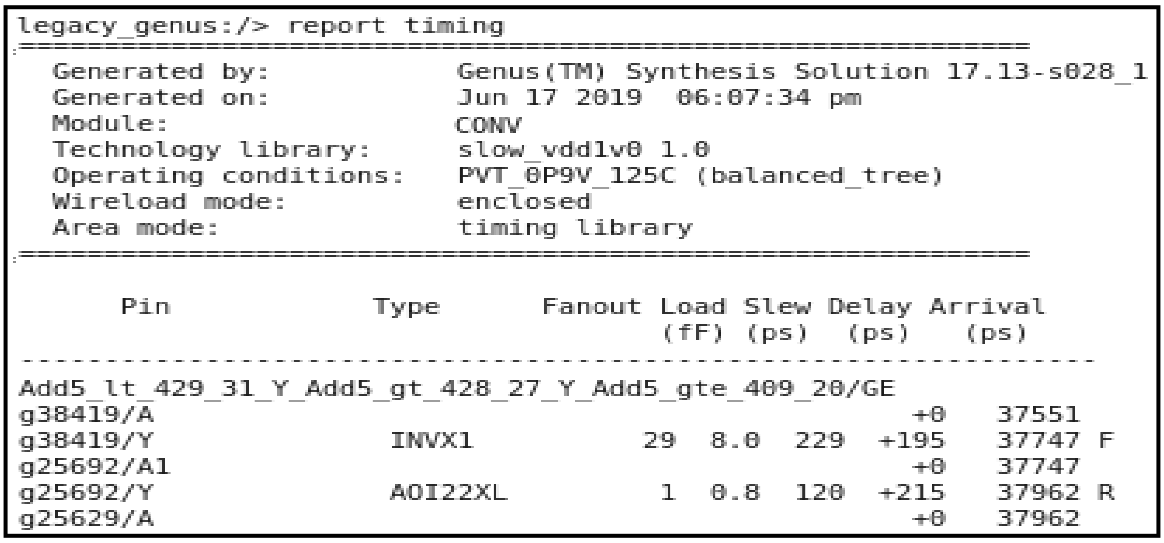
Figure 5.11: Area Report of Convolution Module



```
legacy_genus:/> report power
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 17 2019  05:40:26 pm
  Module:                 CONV
  Technology library:     slow_vddlv0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

                                  Leakage    Dynamic     Total
         Instance          Cells Power(nW)  Power(nW)  Power(nW)
--------------------------------------------------------------------
CONV                       30005  1037.464 245407.644 246445.108
```

Figure 5.12: Power Report of Convolution Module



```
legacy_genus:/> report timing
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 17 2019  06:07:34 pm
  Module:                 CONV
  Technology library:     slow_vddlv0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

      Pin            Type      Fanout Load Slew Delay Arrival
                                      (fF) (ps)  (ps)    (ps)
----------------------------------------------------------------
Add5_lt_429_31_Y_Add5_gt_428_27_Y_Add5_gte_409_20/GE
g38419/A                                          +0   37551
g38419/Y             INVX1          29  8.0  229  +195   37747 F
g25692/A1                                         +0   37747
g25692/Y             AOI22XL         1  0.8  120  +215   37962 R
g25629/A                                          +0   37962
```

Figure 5.13: Timing Report of Convolution Module

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

This work basically presents a convolution circuit employing a novel base3 partitioning technique extracted from Bachet's weight decomposition theorem that allows substitution of multipliers, encoders and supplementary circuitry used to perform convolution operation, typically employed in filters by a chain of adders arranged as pipeline stages and LUTs to store the premultiplied kernel coefficients. Here, adders can perform *addition of signed integer values* and hence values taking $\boldsymbol{\lambda_i}$ as -1 need not have to be stored in the two's complement form which further discards the requirement of additional circuitry to perform conversion of resulting data to obtain actual results. This also reduces the count of LUTs as we can exclude storing two's complemented data, thereby reducing the area. The proposed design is the best suitable for applications based on multiple constant multiplication technique.

This work in future can be extended to design hardware architecture for two dimensional convolution filters which can accept both the positive and negative integers unlike the current design where the input values are restricted to be only positive, thus converting the design into general purpose from application specific.

REFERENCES

[1] *Sergio Silva Ribeiro, Adriano Ferrasa, Rosane Falate*, "Using Python with Simple-CV To Detect a Corn Kernel in Digital Image", Iberoamerican Journal of Applied Computing, V.4, N.2, Aug/2014.

[2] *Shih-Lun Chen*, "VLSI Implementation of an Adaptive Edge-Enhanced Image Scalar for Real-Time Multimedia Applications", IEEE Transactions on Circuits and Systems For Video Technology, Vol. 23, No. 9, September 2013.

[3] *M. C Hanumantharaju, M. Ravishankar, D. R Rameshbabu,* **"**Design and FPGA Implementation of 2D Gaussian Surround Function with Reduced On-Chip Memory Utilization", International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2013.

[4] *Frank Cabello, Julio Leon, Yuso Iano, Rangel Arthur*, "Implementation of Fixed Point 2D Gaussian Filter for Image Processing Based on FPGA", SPA, September 2015.

[5] *Leila kabbai, Anissa Sghaier, Ali Douik and Mohsen Machhout*, "FPGA implementation of filtered image using 2D Gaussian filter", International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 7, No. 7, 2016.

[6] *Deepak Mishra, Vipul Agarwal*, "Implementation of Optimized Floating Point Adder on FPGA", International Journal of Engineering Trends and Applications (IJETA) – Volume 2 Issue 3, May-June 2015.

[7] *Andreas Ehliar*, "Area Efficient Floating-Point Adder and Multiplier with IEEE-754 Compatible Semantics".

[8] *Saloni, Dr. Neelam Rup Prakash*, " Low Power 4*4 Canonical Signed Digit Multiplier using 90nm Technology", International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering (IJIREEICE) Vol. 5, Issue 6, June 2017.

[9] *Sakthivel B, K. Maheshwari, J. Manojprabakar, S. Nandhini, A. Saravanapriya*, "Implementation of Booth Multiplier and Modified Booth Multiplier", International Journal of Recent Trends in Engineering & Research (IJRTER), March 2017.

[10] *B. Sreenivasa Ganesh, J.E.N. Abhilash, G. Rajesh Kumar*, "Design and Implementation of Floating Point Multiplier for Better Timing Performance",

International Journal of Advanced Research in Computer Engineering & Technology(IJARCET) Volume 1, Issue 7, September 2012.

[11] *G. D. Licciardo, C. Cappetta, L. Di Benedetto, and M. Vigliar*, "Multiplier-less Stream Processor for 2D Filtering in Visual Search Applications", IEEE Trans. Circuits and Systems for Video Technology, 2018.

[12] *G. D. Licciardo, C. Cappetta, L. Di Benedetto, and M. Vigliar*, "Weighted partitioning for fast multiplier-less multiple constant convolution circuit," IEEE Trans. Circuits Syst. II, Express Briefs, 2017.

[13] *G. D. Licciardo, C. Cappetta, Antonio D'Arienzo, and Alfredo Rubino,* "Stream Processor for Real-Time Inverse Tone Mapping of Full-HD Images", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 23, No. 11, November 2015.

[14] *Sahdev D. Kanjariya, Rutarth Patel*, "Architecture and Design of Generic IEEE-754 Based Floating Point Adder, Subtractor and Multiplier", International Journal on Recent and Innovation Trends in Computing and Communication, Volume 3, Issue 5, May 2015.

[15] *Shahzad Khan, Mohd. Zahid Alam*, "Implementation and Simulation of IEEE 754 Single-Precision Floating Point Multiplier", International Journal of Engineering Science Invention, Volume 3 Issue 8, August 2014.

[16] *K. Mizuno et al.*, "A low-power real-time SIFT descriptor generation engine for full-HDTV video recognition," IEICE Trans. Electron., Vol. E94-C, No. 4, pp. 448–457, Apr. 2011.

[17] *Ghassem Jaberipur, B. Parhami, Saeid Gorgin, "*Redundant-Digit Floating-Point Addition Scheme Based on a Stored Rounding Value", IEEE Transactions on Computers, Vol. 59, No. 5, May 2010.

[18] *Bipul C. Paul, Shinobu Fujita and Masaki Okajima,* "ROM-Based Logic (RBL) Design: A Low-Power 16 Bit Multiplier*",* IEEE Journal of Solid-State Circuits, Vol. 44, No. 11, November 2009.

[19] *Gian Domenico Licciardo, Carmine Cappetta and Luigi Di Benedetto,* "Design of a Convolutional Two-Dimensional Filter in FPGA for Image Processing Applications", www.mdpi.com/journal/computers.

[20] *E. O'Shea*. (Oct. 2008). "Bachet's problem: As few weights to weigh them all."

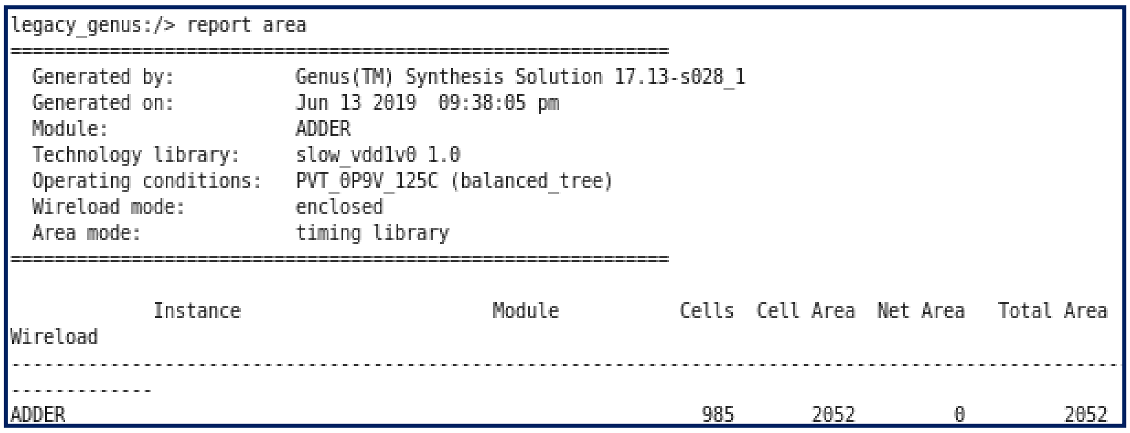
APPENDIX A

BLOCK LEVEL SYNTHESIS REPORT

APPENDIX A

BLOCK LEVEL SYNTHESIS REPORT

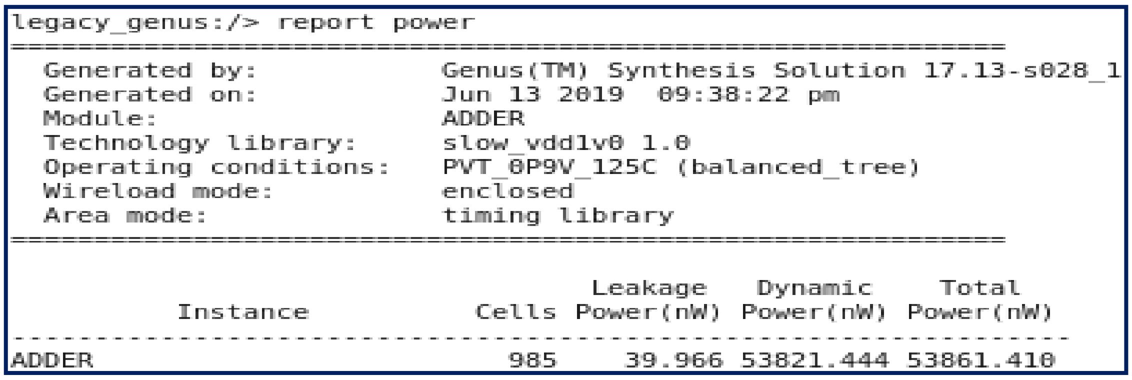
A.1 Synthesis Report of FP32 Adder



```
legacy_genus:/> report area
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 13 2019  09:38:05 pm
  Module:                 ADDER
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

          Instance                    Module          Cells  Cell Area  Net Area   Total Area
Wireload
--------------------------------------------------------------------------------------------------
-------------
ADDER                                                   985       2052         0         2052
```

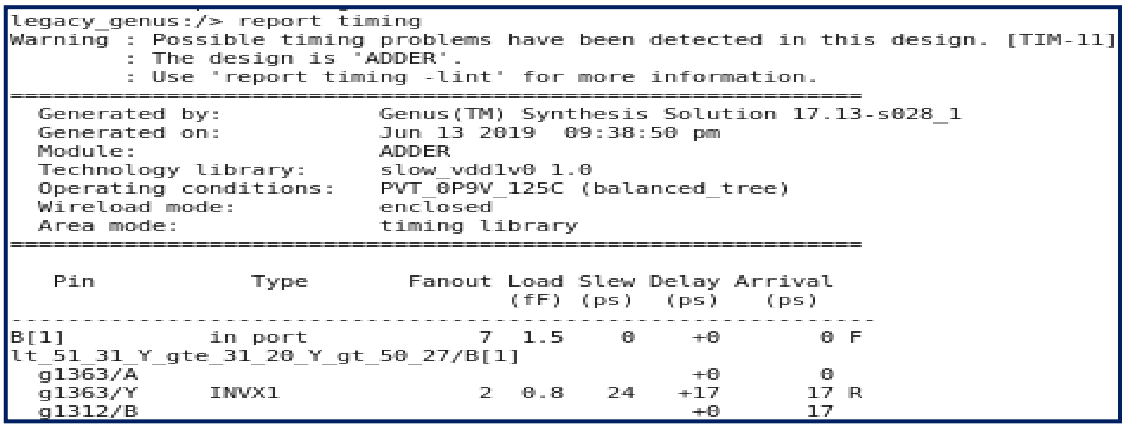
Figure A.1: Area Report of Adder



```
legacy_genus:/> report power
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 13 2019  09:38:22 pm
  Module:                 ADDER
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

                               Leakage   Dynamic    Total
        Instance         Cells Power(nW) Power(nW) Power(nW)
-------------------------------------------------------------
ADDER                      985    39.966 53821.444 53861.410
```

Figure A.2: Power Report of Adder

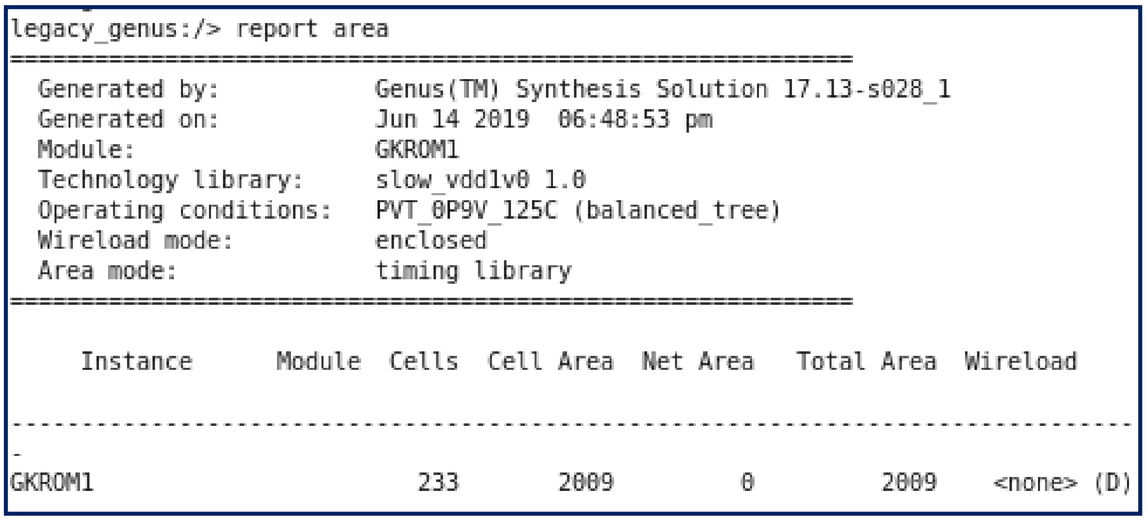


```
legacy_genus:/> report timing
Warning : Possible timing problems have been detected in this design. [TIM-11]
        : The design is 'ADDER'.
        : Use 'report timing -lint' for more information.
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 13 2019  09:38:50 pm
  Module:                 ADDER
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

   Pin            Type        Fanout Load Slew Delay Arrival
                                     (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------
B[1]           in port             7  1.5    0    +0       0 F
lt_51_31_Y_gte_31_20_Y_gt_50_27/B[1]
  g1363/A                                         +0       0
  g1363/Y      INVX1               2  0.8   24   +17      17 R
  g1312/B                                         +0      17
```

Figure A.3: Timing Report of Adder

A.2 Synthesis Report of GKROM

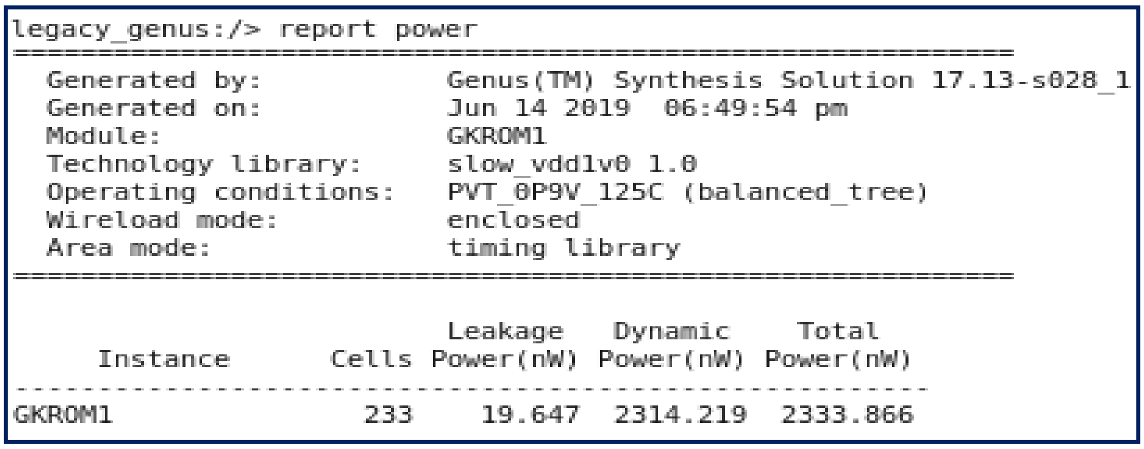


```
legacy_genus:/> report area
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 14 2019  06:48:53 pm
  Module:                 GKROM1
  Technology library:     slow_vddlv0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

     Instance      Module  Cells  Cell Area  Net Area   Total Area  Wireload

--------------------------------------------------------------------------------
-
GKROM1                      233       2009         0         2009    <none> (D)
```

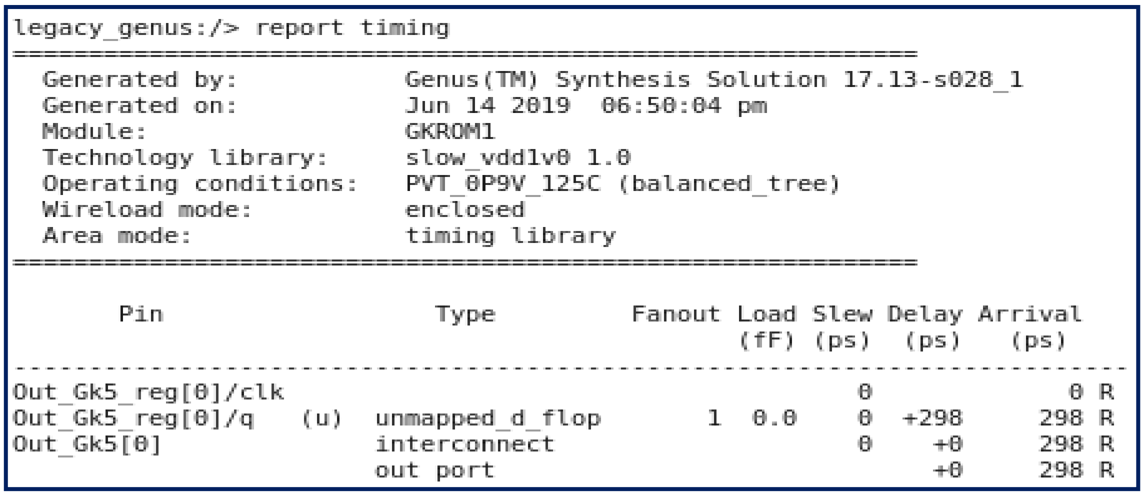
Figure A.4: Area Report of GKROM



```
legacy_genus:/> report power
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 14 2019  06:49:54 pm
  Module:                 GKROM1
  Technology library:     slow_vddlv0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

                            Leakage   Dynamic    Total
     Instance      Cells Power(nW) Power(nW) Power(nW)
-------------------------------------------------------
GKROM1               233    19.647  2314.219  2333.866
```

Figure A.5: Power Report of GKROM

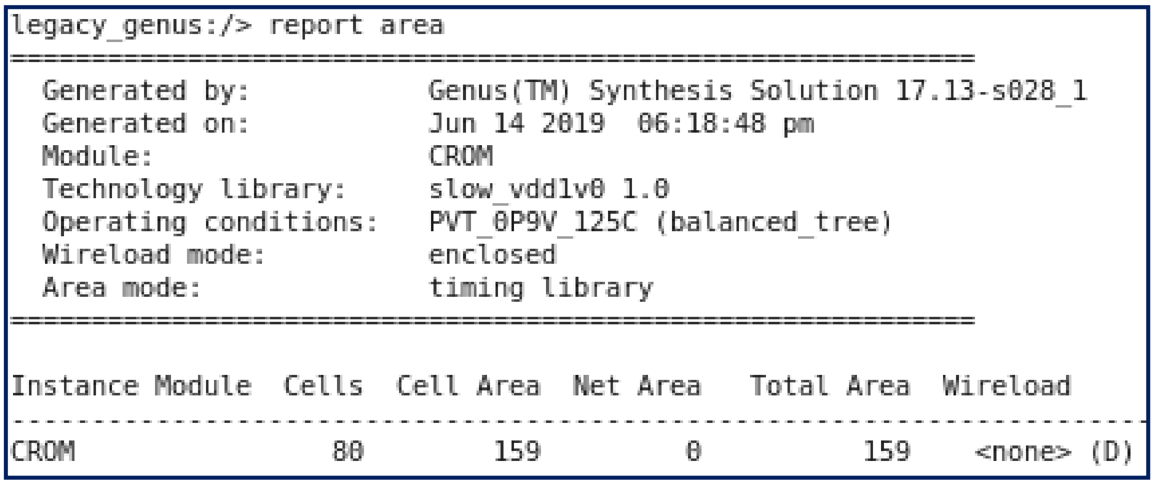


```
legacy_genus:/> report timing
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 14 2019  06:50:04 pm
  Module:                 GKROM1
  Technology library:     slow_vddlv0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

       Pin                    Type         Fanout Load Slew Delay Arrival
                                                  (fF) (ps)  (ps)    (ps)
---------------------------------------------------------------------------
Out_Gk5_reg[0]/clk                                        0            0 R
Out_Gk5_reg[0]/q    (u)  unmapped_d_flop       1  0.0     0  +298    298 R
Out_Gk5[0]               interconnect                     0    +0    298 R
                         out port                              +0    298 R
```

Figure A.6: Timing Report of GKROM

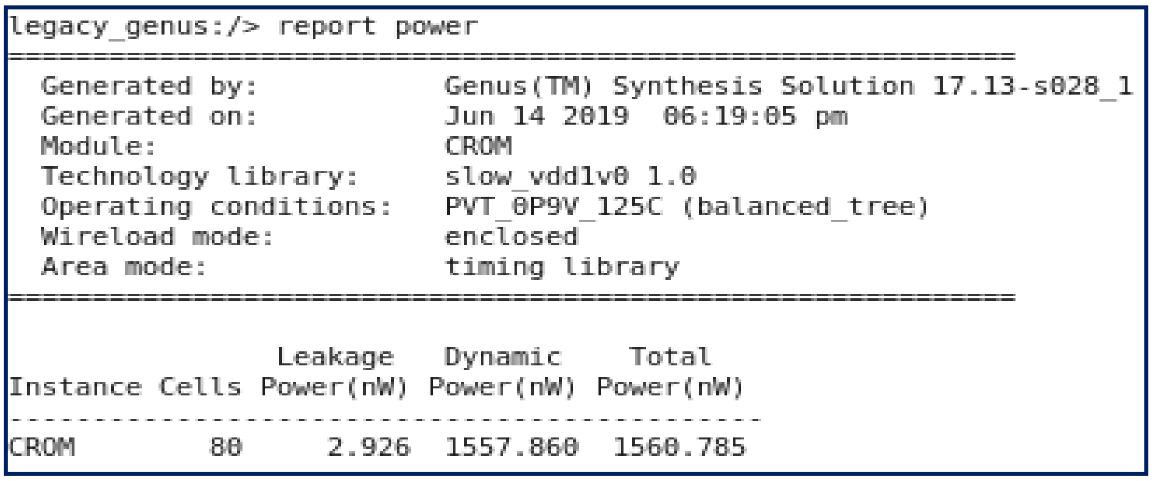
A.3 Synthesis Report of CROM



```
legacy_genus:/> report area
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 14 2019  06:18:48 pm
  Module:                 CROM
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

Instance Module  Cells  Cell Area  Net Area   Total Area  Wireload
--------------------------------------------------------------------
CROM                80        159         0          159    <none> (D)
```

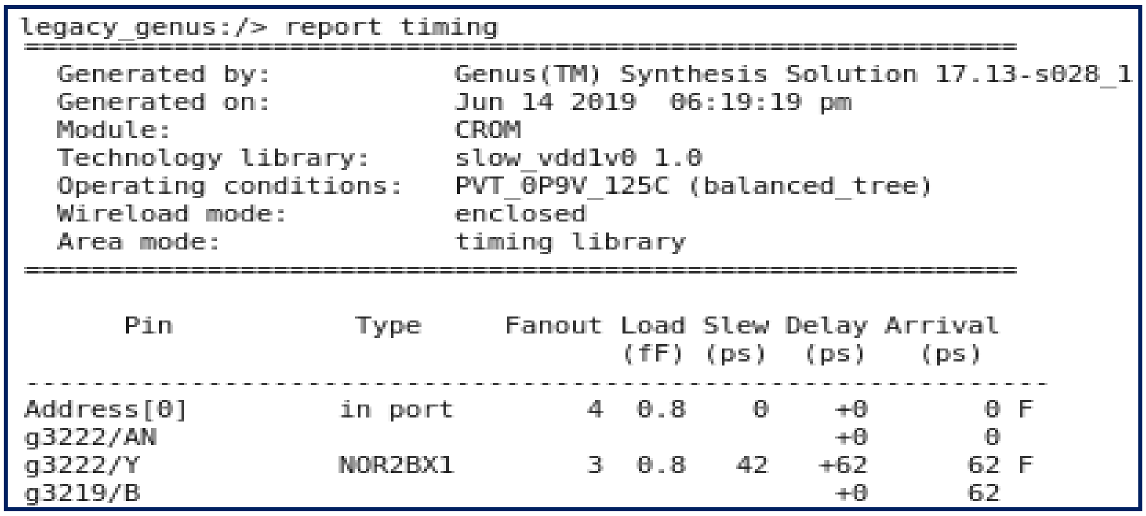
Figure A.7: Area Report of CROM



```
legacy_genus:/> report power
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 14 2019  06:19:05 pm
  Module:                 CROM
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

                 Leakage   Dynamic    Total
Instance Cells Power(nW) Power(nW) Power(nW)
---------------------------------------------
CROM        80     2.926  1557.860  1560.785
```

Figure A.8: Power Report of CROM

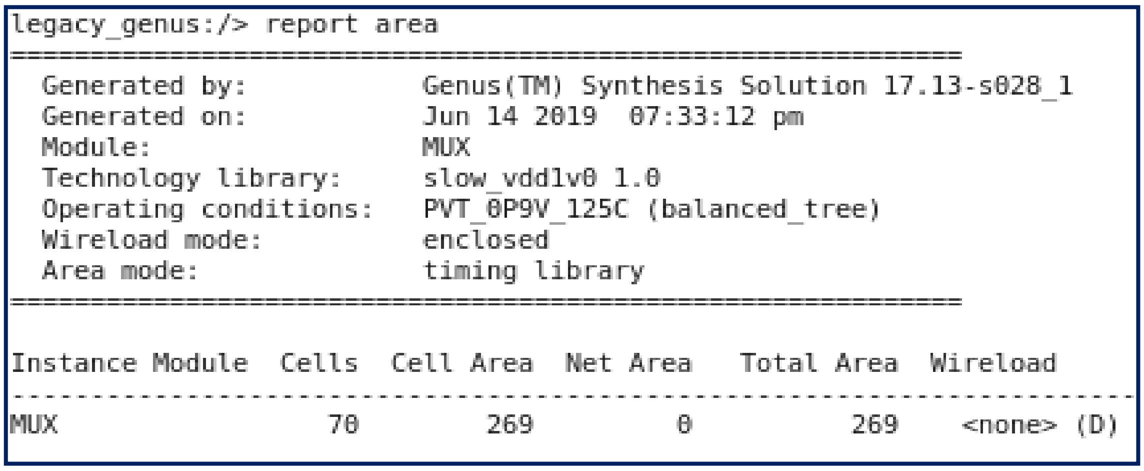


```
legacy_genus:/> report timing
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 14 2019  06:19:19 pm
  Module:                 CROM
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

       Pin            Type      Fanout Load Slew Delay Arrival
                                       (fF) (ps)  (ps)    (ps)
--------------------------------------------------------------
Address[0]         in port         4  0.8    0    +0       0 F
g3222/AN                                          +0       0
g3222/Y            NOR2BX1         3  0.8   42   +62      62 F
g3219/B                                           +0      62
```

Figure A.9: Timing Report of CROM

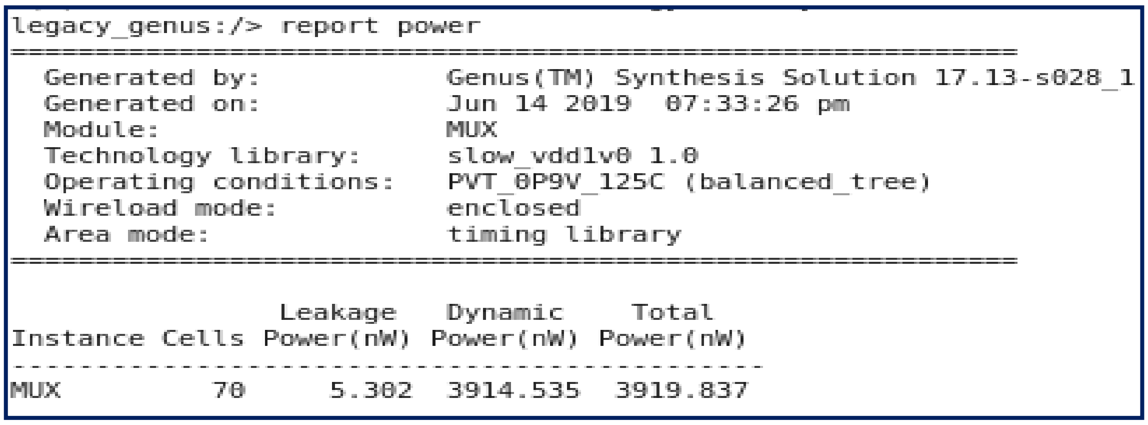
A.4 Multiplexer Synthesis Report



```
legacy_genus:/> report area
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 14 2019  07:33:12 pm
  Module:                 MUX
  Technology library:     slow_vddlv0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

Instance Module  Cells  Cell Area  Net Area   Total Area  Wireload
----------------------------------------------------------------------
MUX                 70        269         0          269    <none> (D)
```

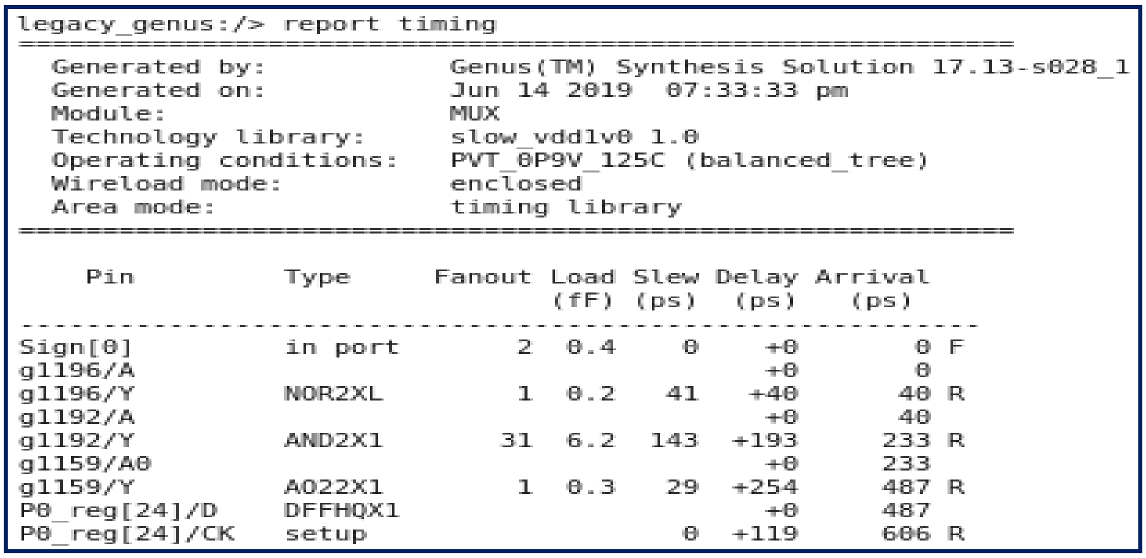
Figure A.10: Area Report of MUX



```
legacy_genus:/> report power
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 14 2019  07:33:26 pm
  Module:                 MUX
  Technology library:     slow_vddlv0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

                Leakage   Dynamic    Total
Instance Cells Power(nW) Power(nW) Power(nW)
------------------------------------------
MUX          70     5.302  3914.535  3919.837
```

Figure A.11: Power Report of MUX



```
legacy_genus:/> report timing
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.13-s028_1
  Generated on:           Jun 14 2019  07:33:33 pm
  Module:                 MUX
  Technology library:     slow_vddlv0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

    Pin          Type      Fanout Load Slew Delay Arrival
                                  (fF) (ps)  (ps)    (ps)
----------------------------------------------------------
Sign[0]         in port        2  0.4    0    +0       0 F
g1196/A                                       +0       0
g1196/Y         NOR2XL         1  0.2   41   +40      40 R
g1192/A                                       +0      40
g1192/Y         AND2X1        31  6.2  143  +193     233 R
g1159/A0                                      +0     233
g1159/Y         AO22X1         1  0.3   29  +254     487 R
P0_reg[24]/D    DFFHQX1                       +0     487
P0_reg[24]/CK   setup                    0  +119     606 R
```

Figure A.12: Timing Report of MUX

APPENDIX B

SYNTHESIS TOOL FLOW

APPENDIX B

SYNTHESIS TOOL FLOW

This section describes the steps involved in generating synthesis report using Cadence Encounter RTL Compiler targeted to 45nm slow_vdd1v0_basicCells.lib technology library as follows.

- Copy the rclabs directory to *myname_digital.*
- Copy the verilog module design1.v to RTL directory in rclabs.
- Open the terminal and change the directory to */home/cadence/DATABASE/myname_digital*
- Invoke the tool for synthesis by typing the command *genus_legacy.ui*
- legacy_genus:/> read_hdl /home/cadence/*DATABASE/myname_digital*/rcfiles/rtl/design1.v
- legacy_genus:/> set_attribute library /home/cadence/*DATABASE/myname_digital*/rcfiles/lib/slow_vdd1v0_basicCells.lib
- legacy_genus:/> set_attribute lib_search_path /home/cadence/*DATABASE/myname_digital*/rcfiles/rtl/
- legacy_genus:/> elaborate design1
- legacy_genus:/> synthesize -to_mapped
- legacy_genus:/> report area
- legacy_genus:/> report power
- legacy_genus:/> report timing

APPENDIX C

PAPER PUBLICATION

An Efficient Radix-3 Multiplierless 2D Convolution Filter for Visual Search Applications

Harshitha A
MTech (IV Sem)
Dept. of Electronics and Communication
BNMIT, Bangalore, INDIA
harshithaanand94@gmail.com

Dr. Yasha Jyothi M Shirur
Professor
Dept. of Electronics and Communication
BNMIT, Bangalore, INDIA
yashamallik@gmail.com

***Abstract* – In real time Visual Search(VS) applications which involves object recognition and segmentation using image processing, a filtering stage is mandatory. Usually, in all filtering applications convolution is performed between the image tile and a kernel. A two-dimensional convolution circuit is presented based on the Bachet's weight decomposition theorem which basically exploits a new Radix-3 partitioning method of integer numbers. The proposed method eliminates the use of multipliers by - *chain of simplified single precision floating point (FP) adders* designed based on IEEE 754 FP signed addition algorithm. In order to increase the resolution and accuracy, the floating point representation is used. The proposed design provides more efficient implementation of floating point convolution architecture when a fixed set of coefficients is used over a range of input values. The design is coded in verilog and simulated using Isim from Xilinx ISE. The design is targeted on Virtex 7 FPGA Board. This paper mainly focuses at reducing the LUT count and power consumption by implementing proposed algorithm for multiplying two floating point numbers. The *area/LUT* count is significantly reduced by *48.54%* and power consumption by *20.17%* in comparison with [1]. However there's a tradeoff in delay.**

Index Terms— *Bachet's weight decomposition, Convolution, Multipliers, Radix-3 partitioning.*

I. INTRODUCTION

In Visual Search applications, a 2D convolution filter is used to perform filtering as the object to be processed is always two dimensional. In such applications convolution is performed between the input image and a kernel. In these applications, Software implementations of the algorithm is limited in terms of maximum frequency that can be achieved to process a single image, thus making them difficult to achieve real-time performance, while the problem is even more worse for images having higher resolutions unless and until strong simplifications of the algorithms is done. Due to these reasons, Hardware implementation of filtering stages has become more and more frequent, but it must be noted that the effort to achieve optimized designs is not justified by the real-time requirements only. In fact, the increasing demand for handheld devices and Internet-of-Things(IoT) applications is making path for new designs, capable of achieving better Area-Power-Delay (ADP) trade-offs. Considering all these issues, during the last years, several hardware designs have been proposed aiming to obtain optimized filtering architectures. Hardware complexity of the design is the major problem for the applications aiming at high speed, lower area. Such complexity, certainly deteriorates the performance, due to the failure in the allocation of a large number of arithmetic operators resulting in consequent slackening of the overall circuit. The survey related to these problems shows that the above issue is usually managed by either adapting full/partial serialization of the filters and folding techniques, or by superseding the inherent complexity of fused Multiply-Adders and Multiply- ACcumulators (MAC). The former way usually results in significant reduction in filter performance. Hence, the latter approach is opted as it is the most accurate way to achieve a good Power, Performances, Area (PPA) trade-off. In such cases, several authors have preferred to replace the multiplier circuitry by adders and shifters in according to the coding of the operands, Canonical Signed Digit (CSD) and Modified Booth (MB). The effective simplification of filtering circuits is, when one of the operands is reduced to a bounded set of pre-calculated values, as in the case of pre-defined filter kernels. For the cases like these, in order to partition multiplications in simpler shifts and additions the Distributed Arithmetic (DA) method can be successfully employed. Usage of Distributed Arithmetic can be advantageous over MB and CSD by using memories to store pre-calculated partial sums, whose number can be decreased by incorporating Multiple Constant Multiplication (MCM) technique. Conversely, performance of DA is the result of trade-off between its natural bit serial operation and the parallelism by which the partial sums are calculated, leading to the excessive increase in the number of mapped physical resources.

This paper presents new radix-3 partitioning scheme based on Bachet's weight decomposition theorem with the purpose to improve performances of general purpose multipliers in the specific contexts of MCM. Even though this method similar to the operation principle of DA, the proposed architecture for the convolution circuit is advantageous in terms of area, power to perform multiplication of 32-bit floating point (FP 32) kernel coefficients with integer inputs(input pixels) defined over a range of integer values compatible with multimedia applications . This procedure eliminates the need for shifters and auxiliary circuitry which is common with most of the multiplier schemes by substituting multiplier with FP adders without compromising on the accuracy of the result. This work is best suitable for filtering applications involving multiple constant multiplication.

II. MATHEMATICAL BACKGROUND

Over several decades there's an issue as to how all the numbers can be represented using the minimum number of integers such that the numbers over a certain range can be expressed as a combination of them. Because several applications discussed earlier in the introduction require filtering apparatus and the complexity of filtering circuit increases with increasing range of image pixels, it's high time to find a solution to reduce the complexity of the filtering circuit. In our paper we present a method to eliminate the use of multipliers in convolution circuit by replacing them with a chain of adders.

There are several papers about this argument one of which is discussed here. Having defined the set $W_r = \{3^0, 3^1, 3^2, \ldots, 3^{m-1}, R\}$ with $R = r - (3^0 + 3^1 + 3^2 + \ldots + 3^{m-1})$ each value in the range $[0{:}r]$ can be obtained by the superposition of the terms in W_r multiplied for a coefficient $C_j \in \{-1, 0, 1\}$. In general, *partition* of a positive integer r is defined as the ordered sequence of positive integers such that $r = \lambda_0 + \lambda_1 + \lambda_2 + \ldots + \lambda_m$ *with* $\lambda_0 < \lambda_1 < \lambda_2 < \ldots < \lambda_m$ and as parts of the partition set $\{\lambda_0, \ldots, \lambda_{m-1}, \lambda_m\}$, it can be stated that

1. Every integer $0 \le p \le r$ can be written as

$$p = \sum_{j=0}^{m} C_j \lambda_j \qquad (1)$$

II. There *does not exist* another partition of r satisfying (1) with *fewer parts* than m+1.

Here each partition of r consists of exactly $m + 1 = \lfloor log_3 2r \rfloor + 1$ parts.

Table I: **Bachet's Weight Partition**

INPUT	***PARTITION***					
	3^0	3^1	3^2	3^3	**…**	λ_m
0	**0**	**0**	**0**	**0**	**…**	**0**
1	**+1**	**0**	**0**	**0**	**…**	**0**
2	**-1**	**+1**	**0**	**0**	**…**	**0**
3	**0**	**+1**	**0**	**0**	**…**	**0**
4	**+1**	**+1**	**0**	**0**	**…**	**0**
5	**-1**	**-1**	**+1**	**0**	**…**	**0**
. .	. .	. .	. .	. .	. .	. .
p	C_0	C_1	C_2	C_3	**……**	C_m
. . .	. . .	. . .	. . .	. . .	. . .	. . .
r	**+1**	**+1**	**+1**	**+1**	**+1**	**+1**

Table I shows how each positive integer can be represented in a radix-3 format. For instance: an 8 bit input takes {0, 1, 3, 9, 27, 81, 134} as its parts. The input 45 can be represented as a combination of coefficients and parts as 45 = (0)1+ (0)3 + (-1)9 + (-1)27+(+1)81+(0)134 with the coefficients being {0, 0, -1, -1, +1, 0} as per Table I.

The same partitioning method can be adopted to perform MAC operations between the vector of coefficients A_i and input vector x_i.

$$y = \sum_{i=0}^{i-1} A_i x_i \qquad (2)$$

Combining (1) and (2), we get,

$$y = \sum_{i=0}^{i-1} \sum_{j=0}^{m} A_i C_{ij} \lambda_j \qquad (3)$$

III. PROPOSED ARCHITECTURE DESIGN

The hardware architecture of the proposed design can be implemented as shown in the Figure 1. This scheme performs the convolution of FP32 kernel vector, G and the unsigned integer input vector I of n

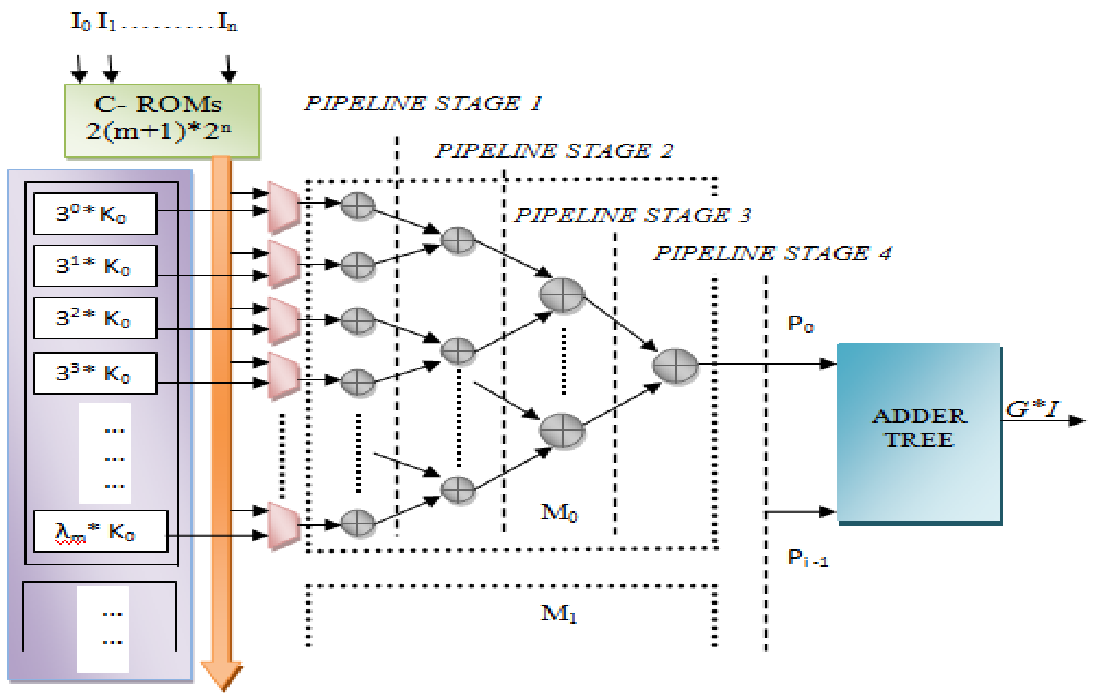


Figure 1. Proposed Multiplier Architecture

bit. To perform convolution, *G* is to be defined and each value in the input vector *I* is decomposed into *m+1* parts as per TABLE I. Each value in *G* defined as [k_0 k_1.....k_{h-1}] is premultiplied with radix-3 weights defined as {3^0, 3^1,…,λ_m} and stored in the LUT. These premultiplied values are stored in IEEE 754 standard. Here C-ROM stores the coefficients $C_j \in$ {-1, 0, 1} as {11, 00, 01}(each sign coded with 2bits) which is used to select the sign of the input values through the multiplexer bank. Further each multiplier is replaced by adders as represented within the dashed box in Figure 1 whose tree depth is $\lfloor log_3 2r \rfloor + 1$. Unlike [1], adders can perform *addition of signed integer values* and hence values taking C_j as -1 need not have to be stored in the two's complement form which further discards requirement of additional circuitry to perform the conversion of resulting data to obtain actual results. The adders are designed as per the IEEE 754 FP32 signed addition algorithm which can take all possible combinations {+ +, + -, - +, - -} of operands including zero. Also no separate method is used to code the exponents of values in G making the design more clear and simple.

Though this convolution architecture is compatible with all kinds of kernel we have chosen to implement a 2D Gaussian filter with Gaussian kernel defined as

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \qquad (4)$$

working with Unit-8 inputs. The input range is chosen to be r = 256 with the number of parts being $m+1 = \lfloor log_3 2r \rfloor + 1 = 6$ i.e. {1, 3, 9, 27, 81, 134} as $\lambda_m \equiv R = 255 - (3^0 + 3^1 + 3^2 + 3^3 + 3^4) = 134$. The kernel size is chosen to be K = 3 since it is the minimum allowable dimension for VS applications and $\sigma = 1$. The input is chosen as 5×5 image with each pixel value 8 bit wide.

IV. SIMULATION AND SYNTHESIS RESULTS

A 2D Gaussian convolution filter is a combination of multipliers built using FP32 adders. Hence single precision FP adder becomes the basic building block. These adders are instantiated in pipeline stages similar to the Figure 1 to behave as a multiplier. Several MAC operations are performed to realize a convolution operation and convolutions to obtain filtered output. Thus initially Figure 2 shows the simulation results of an adder that can perform addition on 32 bit FP values represented in IEEE 754 standard. *Callout 1* is used to indicate 1st cycle of addition operation performed on operands A= -7.36= 1 10000001 11010111000010100011111, B= 7.36= 0 10000001 11010111000010100011111 and the sum so obtained is S = 0 = 0 00000000 00000000000000000000000. *Callout 2* indicates 2nd cycle of addition operation where the operands are A= 7.36 = 0 10000001 11010111000010100011111, B= 7.36 = 0 10000001 11010111000010100011111

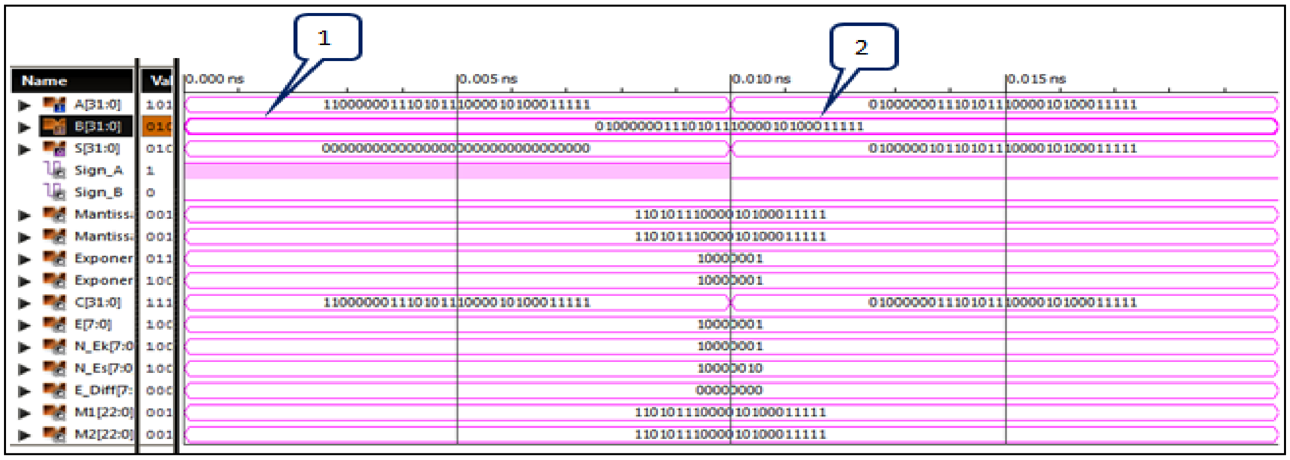


Figure 2. Simulation result of single precision floating point adder

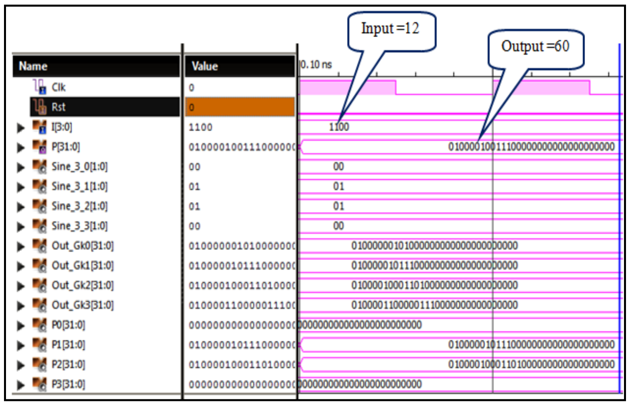


Figure 3. Simulation Results of single precision floating point Multiplier

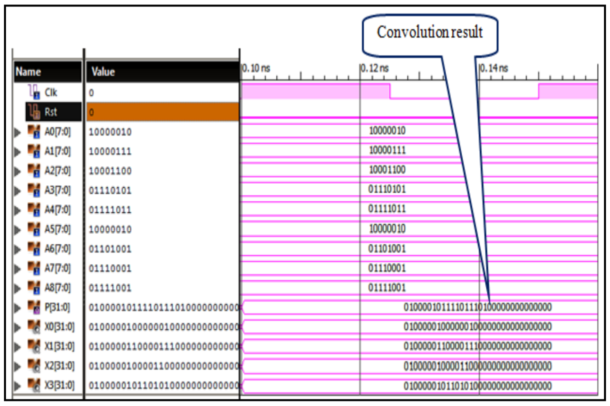


Figure 4. Simulation Results of Convolution Operation

and sum is S = 14.72 = 0 10000010 11010111000010100011111. As discussed earlier one of the operands is premultiplied with its ternary weights and stored in the LUT. Based on the input provided the sign coefficients are selected by the multiplexer bank and additions are performed as per Figure 1 thus resulting in multiplication. For the multiplication operation whose simulation results are shown Figure 3 the operand stored in LUT is 5 and the other operand provided as input is *I*=12 and the product P = 60. Convolution operation is performed between *G* calculated as per (4) and the input image *I*. Callout in Figure 4 indicates result obtained by performing convolution between *G* and *I*.

Table II represents the power, area, delay report of the proposed single precision FP multiplier in comparison with the existing multiplier design [1]. The proposed design is simulated in Xilinx ISE simulator and targeted on the latest Virtex 7 FPGA.

Power, area and delay reports are generated in the same platform. Area is reduced by 48.54% and power by 20.17% in comparison with [1]. In proposed design the LUT count is reduced thus showing reduction in area and power.

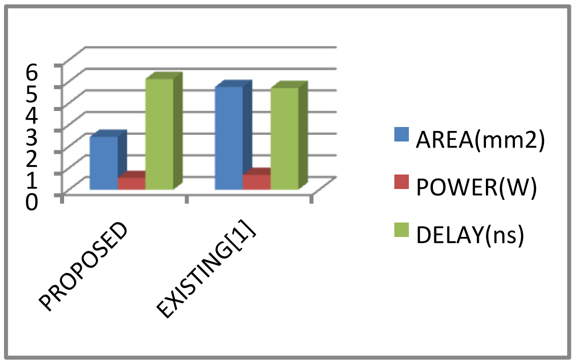


Figure 5. Power, Area, Delay comparison chart

Table II: **Synthesis of Proposed Multiplier targeted on recent FPGA**

	Proposed	*Existing*[1]
Platform	Virtex 7	Virtex
Area in LUTs(μm^2)	2,444	4750
Power (W)	0.546	0.684
Delay(ns)	5.12	4.7

V. CONCLUSION

In this paper we present a convolution circuit employing a new radix-3 partitioning method derived from Bachet's weight decomposition theorem, which allows substitution of multipliers, encoders and supplementary circuitry used to perform convolution operation, typically employed in filters by a chain of adders arranged as pipeline stages and LUTs to store the premultiplied kernel coefficients. Here unlike in [2], adders can perform *addition of signed integer values* and hence values taking C_i as -1 need not have to be stored in the two's complement form which further discards requirement of additional circuitry to perform conversion of resulting data to obtain actual results. This also reduces the count of LUTs as we can exclude storing two's complemented data, thereby reducing the area. The proposed design is the best fit for applications based on multiple constant multiplication technique.

REFERENCES

[1] G. D. Licciardo, C. Cappetta, L. Di Benedetto, and M. Vigliar, "Multiplier-less Stream Processor for 2D Filtering in Visual Search Applications", *IEEE Trans. Circuits and Systems for Video Technology*.

[2] G. D. Licciardo, C. Cappetta, L. Di Benedetto, and M. Vigliar, "Weighted partitioning for fast multiplier-less multiple constant convolution circuit," *IEEE Trans. Circuits Syst. II, Express Briefs*.

[3] E. O'Shea. (Oct. 2008). "Bachet's problem: As few weights to weigh them all."

[4] Gian Domenico Licciardo, Carmine Cappetta and Luigi Di Benedetto, "Design of a Convolutional Two-Dimensional Filter in FPGA for Image Processing Applications", www.mdpi.com/journal/computers

[5] B.Sreenivasa Ganesh, J.E.N.Abhilash, G. Rajesh Kumar, "Design and Implementation of Floating Point Multiplier for Better Timing Performance", *International Journal of Advanced Research in Computer Engineering & Technology(IJARCET) Volume 1, Issue 7, September 2012.*

[6] Shahzad Khan, Mohd. Zahid Alam, "Implementation and Simulation of IEEE 754 Single-Precision Floating Point Multiplier", *International Journal of Engineering Science Invention ISSN (Volume 3 Issue 8,August 2014 ,PP.11-16*

[7] Frank Cabello, Julio Leon, Yuso Iano, Rangel Arthur, "Implementation of Fixed Point 2D Gaussian Filter for Image Processing Based on FPGA", *SPA, September 2015.*

APPENDIX D

PLAGIARISM REPORT

1BG17LVS02

ORIGINALITY REPORT

12%	5%	7%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Gian Domenico Licciardo, Carmine Cappetta, Luigi Di Benedetto, Mario Vigliar. "Weighted Partitioning for Fast Multiplierless Multiple-Constant Convolution Circuit", IEEE Transactions on Circuits and Systems II: Express Briefs, 2017 Publication	2%
2	Submitted to Visvesvaraya Technological University Student Paper	2%
3	mdpi.com Internet Source	1%
4	rspublication.com Internet Source	1%
5	uec.repo.nii.ac.jp Internet Source	<1%
6	Submitted to University of Wollongong Student Paper	<1%
7	Bharat Garg, Shireesh Kumar Rai, Akash	

Puranik, GK Sharma. "ES-COINA: A novel energy scalable quality-aware color interpolation architecture", Microprocessors and Microsystems, 2019
Publication <1%

8 Submitted to University of Nottingham
Student Paper <1%

9 civilu.ce.utexas.edu
Internet Source <1%

10 repository.ntu.edu.sg
Internet Source <1%

11 G. D. Licciardo, C. Cappetta, L. Di Benedetto. "Design Criteria for Real-time Processing of HW Gabor Filters in Visual Search", 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018
Publication <1%

12 etheses.whiterose.ac.uk
Internet Source <1%

13 Gian Domenico Licciardo, Carmine Cappetta, Luigi Di Benedetto, Alfredo Rubino, Rosalba Liguori. "Multiplier-Less Stream Processor for 2D Filtering in Visual Search Applications", IEEE Transactions on Circuits and Systems for Video Technology, 2018
Publication <1%

14	www.ijetcse.com Internet Source	<1%
15	onlinelibrary.wiley.com Internet Source	<1%
16	etd.lib.metu.edu.tr Internet Source	<1%
17	dblp.dagstuhl.de Internet Source	<1%
18	kyutech.repo.nii.ac.jp Internet Source	<1%
19	ir.uiowa.edu Internet Source	<1%
20	ugspace.ug.edu.gh Internet Source	<1%
21	Lasith, K. K., and Anoop Thomas. "Efficient implementation of single precision floating point processor in FPGA", 2014 Annual International Conference on Emerging Research Areas Magnetics Machines and Drives (AICERA/iCMMD), 2014. Publication	<1%
22	Submitted to Defence Institute of Advanced Technology, Pune Student Paper	<1%

23	Submitted to Harrisburg University of Science and Technology Student Paper	<1%
24	Ghassem Jaberipur, Saeid Gorgin. "An improved maximally redundant signed digit adder", Computers & Electrical Engineering, 2010 Publication	<1%
25	polen.itu.edu.tr Internet Source	<1%
26	Yi Cai, Weiming Wang, Weifeng Qian, jia xing et al. "FPGA Investigation on Error-Flare Performance of a Concatenated Staircase and Hamming FEC Code for 400G Inter-Data Center Interconnect", Journal of Lightwave Technology, 2018 Publication	<1%
27	Werner-Michael Kulicke, Ulrich Wallbaum. "Determination of first and second normal stress differences in polymer solutions in steady shear flow and limitations caused by flow irregularities", Chemical Engineering Science, 1985 Publication	<1%
28	Xiang Li, Jin'an Xu, Wenbin Jiang, Qun Liu, Yajuan Lu. "A fixed-point decoding approach for statistical machine translation on mobile	<1%

	terminals", 2010 4th International Universal Communication Symposium, 2010 Publication	
29	"Advanced Computational and Communication Paradigms", Springer Science and Business Media LLC, 2018 Publication	<1%
30	www.cs.uakron.edu Internet Source	<1%
31	"Design of a Convolutional Two-Dimensional Filter in FPGA for Image Processing Applications", Computers, 2017 Publication	<1%
32	Amir, Israel, and Frank P. Higgins. "", Optics Illumination and Image Sensing for Machine Vision VI, 1992. Publication	<1%
33	Bipul C. Paul, Shinobu Fujita, Masaki Okajima. "ROM-Based Logic (RBL) Design: A Low-Power 16 Bit Multiplier", IEEE Journal of Solid-State Circuits, 2009 Publication	<1%
34	www.istp.org.in Internet Source	<1%
35	Mumin Sajad Shawl, Archika Singh, Nidhi Gaur, Shikha Bathla, Anupama Mehra.	<1%

"Implementation of Area and Power Efficient Components of a MAC Unit for DSP Processors", 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), 2018
Publication

36	Submitted to Informatics Education Limited Student Paper	<1%
37	Submitted to Universiti Teknologi Malaysia Student Paper	<1%
38	Submitted to Dublin City University Student Paper	<1%
39	Rao, Y. Srinivasa, M. Kamaraju, and D V S Ramanjaneyulu. "An FPGA implementation of high speed and area efficient double-precision floating point multiplier using Urdhva Tiryagbhyam technique", 2015 Conference on Power Control Communication and Computational Technologies for Sustainable Growth (PCCCTSG), 2015. Publication	<1%
40	audentiav.cluster002.ovh.net Internet Source	<1%
41	Submitted to Engineers Australia Student Paper	<1%

Leila kabbai, Anissa Sghaier, Ali Douik,

42	Mohsen Machhout. "FPGA implementation of filtered image using 2D Gaussian filter", International Journal of Advanced Computer Science and Applications, 2016 Publication	<1%
43	xxx.unizar.es Internet Source	<1%
44	epdf.tips Internet Source	<1%
45	Michael J. Schulte, Earl E. Swartzlander. "Программный интерфейс и конструкция аппаратуры для интервальной арифметики переменной разрядности", Reliable Computing, 1995 Publication	<1%
46	es.scribd.com Internet Source	<1%
47	Andreas Ehliar. "Area efficient floating-point adder and multiplier with IEEE-754 compatible semantics", 2014 International Conference on Field-Programmable Technology (FPT), 2014 Publication	<1%
48	www.ijettjournal.org Internet Source	<1%
49	"Artificial Intelligence and Evolutionary Computations in Engineering Systems",	<1%

Springer Science and Business Media LLC, 2016
Publication

50	Alex Goncalves Saraiva, Osamu Saotome, Roberto D'Amore, Elcio Shiguemori. "A Real Time Adaptive Template Matching Algorithm in UAV Navigation Using a SoC System", 2019 X Southern Conference on Programmable Logic (SPL), 2019 Publication	<1%
51	Signals and Communication Technology, 2014. Publication	<1%
52	www.ijireeice.com Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		