DESIGN AND IMPLEMENTATION OF EFFICIENT MATRIX MULTIPLICATION USING VARIOUS ARCHITECTURE

A DISSERTATION REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTSFOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

VLSI DESIGN & EMBEDDED SYSTEM

SUBMITTED BY:

SHIVAM KUMAR

2K21/VLS/17

UNDER THE SUPERVISION OF

Dr. J.Panda
PROFESSOR

ELECTRONICS & COMMUNICATION ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

MAY 2023

M. Tech (VLSI Design and Embedded System) **Shivam Kumar** **2023**

ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, Shivam, Roll No. 2K21/VLS/17 student of M. Tech (VLSI Design & Embedded system), hereby declare that the project Dissertation titled **"Design and Implementation of efficient Matrix Multiplication using various architecture"** which is submitted by me to the Department of Electronics and Communication Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology is unique and has not been copied without proper citation. This work has never before been used to give a degree, diploma associateship, fellowship, or other equivalent title or recognition.

Place: Delhi   Shivam Kumar

Date: May 29, 2023

ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled **“Design and Implementation of efficient Matrix Multiplication using various architecture”** which is submitted by **Shivam Kumar , 2K21/VLS/17**, to the Department of Electronics & Communication Engineering, Delhi Technological University, Delhi in partial fulfillment of the prerequisite for the award of the degree of Master of Technology, is a documentation of the student's projectwork completed under my supervision. To the best of my knowledge, this work hasnever been submitted in part or in full for any degree or diploma at this university or anywhere else.

Place: Delhi

Date**:** May 29, 2023,

Dr. J.Panda

SUPERVISOR

Professor

Department of ECE, DTU

ACKNOWLEDGMENT

First and foremost, I would like to express my heartfelt gratitude to my mentor **Dr. J.Panda** for helping me on each step of this research, due to his continuous help and support I completed this research. His patience, motivation, ambition, and wide knowledge aided me throughout the research and dissertation writing process. Without his guidance, I could not have imagined completing this research. His suggestions and the way of managing the work were the most important key points, which made my path clear. He helped me use my full potential because of that I tried my best to get the maximum for this project. I am extremely grateful to him for guiding me throughout the project. I am thankful to Electronics Department and faculty for helping me, and thankful to myfriends who helped me directly and indirectly in my research.

Shivam Kumar

2K21/VLS/17

MTech: VLSI Design and Embedded System

(2021-2023)

ABSTRACT

The semiconductor industry plays a crucial role in the design and manufacture of integrated circuits (ICs) used in a wide range of electronic devices. VLSI technology allows for the integration of millions of transistors onto a single chip, enabling the creation of highly complex and powerful devices such as computers, smart phones, and other electronic devices. The VLSI industry is a key driver of innovation in the electronics industry and has played a major role in the development of new technologies and the proliferation of electronic devices in our daily lives. Consequently, area, speed, and power play a critical role in any circuit design .A circuit must be created to meet the present trend's requirements with minimal space and minimal time limitations.

Matrix multiplication is of significant importance in various fields and applications. Matrix multiplication plays a fundamental role in linear algebra, solving system of linear equations, data analysis and machine learning, computer graphics and computer vision, network theory and graph algorithm, etc. This thesis gives a thorough investigation into how the Wallace tree multiplier, Vedic multiplier, and parallel prefix adders might be combined to enhance matrix multiplication performance. These techniques contribute to achieving significant speed improvements, reduced and optimized resource utilization. The findings of this study add to understanding of digital circuit design by offering suggestions for choosing and incorporating effective multiplication methods for matrix operations. The thesis provides helpful advice to researchers and designers of digital circuits by explaining the trade-offs, benefits, and drawbacks of the integrated architecture.

Firstly, Ripple Carry adders, Kogge Stone adders, and Han Carlson adders have been designed and analyzed. After that, the Wallace tree multiplier and Vedic multiplier are designed using these adders. By combining both multiplier and adder, matrix multiplication designs, analyses the performance data, and interprets the results obtained from the experiments. Using the ISE Design Suite tools in Verilog, all circuits are created and simulations are run. The XC6SLX150T are the devices used for synthesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

S.No.	Abbreviation	Full Name
1.	VLSI	Very Large-Scale Integration
2.	RCA	Ripple Carry Adder
3.	KSA	Kogge Stone
4.	HCA	Han Carlson Adder
5.	WTM	Wallace Tree Multiplier
6.	VM	Vedic Multiplier
7.	DSP	Digital Signal Processing
8.	PPA	Parallel Prefix Adder
9.	UTM	Urdhva Tiryakbhyam multiplier

CHAPTER 1

INTRODUCTION

Multipliers are essential arithmetic units in the VLSI (Very Large Scale Integration) industry, particularly in digital signal processing (DSP), communication systems, image processing, and various other applications. They are responsible for performing multiplication operations, which are fundamental to numerous computational tasks. The VLSI industry continually works on developing advanced multiplier architectures to enhance speed, power efficiency, and area utilization. Emerging technologies such as approximate computing and approximate multipliers contribute to achieving a balance between accuracy and computational efficiency. Overall, multipliers are indispensable components in the VLSI industry, enabling efficient multiplication operations in various applications and driving ongoing research and development.

In many computer applications, matrix multiplication is an essential operation, especially in the areas of linear algebra, image processing, machine learning, and scientific computing. The need for effective and optimized matrix multiplication algorithms grows as the size and complexity of matrices continue to increase. There is a growing need for effective and optimized methods for matrix operations as the need for high-performance computing keeps rising. In this work, to create a comprehensive framework for high-performance matrix multiplication and multiplication operations in digital circuit designs, investigate the integration of parallel prefix adders alongside the Wallace tree multiplier and Vedic multiplier.

The Wallace tree multiplier and Vedic multiplier are most effective multiplication. The Wallace tree multiplier reduces the number of partial products and offers faster multiplication speeds by utilizing a combination of carry-save adders and carry-propagate adders. Similarly, the Vedic multiplier leverages simple arithmetic operations, inspired by ancient Indian mathematics principles, to achieve high-speed multiplication with fewer logic gates compared to conventional multipliers.

We suggest using parallel prefix adders into the architecture of matrix multiplication and the multiplier. Carry propagation is parallelized via parallel prefix adders, also known as carry-look-ahead adders, which significantly increase speed. Parallel prefix adders reduce the critical route time by efficiently allocating and computing the carries in parallel, which improves overall performance.

For the purpose of creating an optimized framework for matrix multiplication and multiplication operations in digital circuit designs, this thesis investigates the synergistic integration of the Wallace tree multiplier, Vedic multiplier, and parallel prefix adders. Utilizing each component's advantages will increase throughput, decrease latency, and enhance power efficiency.

Partitioning the input matrices, using the implemented multipliers to create partial products, and implementing parallel prefix adders for effective accumulation and carry propagation are all part of the suggested methodology. We hope to significantly increase matrix multiplication and multiplication operations performance by parallelizing the computation and reducing the critical path delay. All the design and simulation are performed in Verilog using ISE Design Suite tool. The selected board is XC6SLX150T.

1.1 Motivation

A variety of computer activities, from scientific simulations to machine learning algorithms, can be significantly sped up by using efficient matrix multiplication and multiplication processes. Traditional multiplication methods may become a bottleneck when data amount and complexity rise, reducing system performance as a whole. Therefore, there is a strong motivation to explore novel approaches that can enhance the efficiency and speed of matrix multiplication operations in digital circuit designs.

The Wallace tree multiplier and Vedic multiplier have already shown potential for accelerating multiplication and lowering the incidence of partial products. Additional methods must be investigated, though, in order to improve the performance of these multipliers and enable quicker matrix multiplication. Due to the parallelization of the carry propagation mechanism, parallel prefix adders have seen notable speed gains.

The necessity to create an optimized framework that incorporates the benefits of the Wallace tree multiplier, Vedic multiplier, and parallel prefix adders served as the driving force for this work. The aim of this work is to increase throughput, decrease latency, and reduce area in matrix multiplication and multiplication operations by utilizing the parallel computing capabilities of the adders coupled with effective multiplication strategies. The outcome of this research has the potential to benefit various domains that heavily rely on matrix operations, providing faster and more energy-efficient solutions for processing large-scale matrices leading to advancements in various domains. Furthermore, the insights gained from this research can contribute to the ongoing development of efficient arithmetic units and computing architectures, paving the way for future advancements

1.2 Objective

New questions, ideas, and understandings can arise as a result of an analysis of the available data. Exploring uncharted territory is the main objective of research in order to find new opportunities.

Primary objectives of this research project are as follows:

- To design and implement an efficient matrix multiplication algorithm using the Wallace tree multiplier, Vedic multiplier, and parallel prefix adders in digital circuit designs.
- To understand the working of multipliers and which ones perform the best, a comparison is done between Wallace tree multiplier and Vedic multiplier. To enhance the performance of multiplier, parallel prefix adder (Kogge Stone and Han Carlson adder) are used.
- To validate the correctness and accuracy of the matrix multiplication algorithm by comparing the results with established mathematical solutions.
- To contribute to the body of knowledge in the field of digital circuit design by investigating the synergistic integration of the Wallace tree multiplier, Vedic multiplier, and parallel prefix adders for effective matrix multiplication operations

By accomplishing these objectives, this effort gives a thorough understanding of the functionality, effectiveness, and applicability of the integrated framework. By enabling more effective and optimized matrix operations across a range of computing domains, the research's results and insights will boost digital circuit design methodologies.

1.3 Thesis Organization

This thesis is divided into six chapters. The thesis is organized as follows: Chapter 2 provides the literature review and the technology gap. Chapter 3 is a comprehensive review of the Ripple Carry adder, the Kogge Stone adder, and the Han Carlson adder. Chapter 4 examines the comparison of the Wallace tree multiplier and the Vedic multiplier using different adders. Chapter 5 depicts the design of matrix multiplication and its analysis on the basis of performance and area. The conclusion and future scope are presented in Chapter 6.

- CHAPTER 1- Provides a basic overview of matrix multiplication, multiplier and parallel prefix adder operation. In this chapter discussion areas are thesis's objective, motivation, methodology, and thesis organization.
- CHAPTER 2- This chapter provides a detailed overview of earlier research on parallel prefix adders and multiplication. According to published research, Vedic and Wallace tree multipliers are faster multipliers than traditional ones. Additionally, Han Carlson and Kogge Stone PPAs are quicker than Ripple carry adder.
- CHAPTER 3- This chapter covers the details of the implementation of Adder used in this work. The design and performance evaluation of the Kogge Stone, Ripple Carry, and Han Carlson adders are covered in this chapter.
- CHAPTER 4- This chapter covers the description of the Wallace tree and the Vedic multiplier. Both multipliers are designed using different adders (discussed in the previous chapter) and compared for their performance.
- CHAPTER 5- This chapter presents the methodology employed in this research, including the design and integration of the matrix multiplication algorithm. This chapter describes the implementation of matrix multiplication using multipliers (discussed in chapter 4) and comparison between area and delay.
- CHAPTER 6- This chapter contains the important outcomes of every chapter along with the future work.

In order to explore the directions for our current work, a comprehensive list of references is supplied at the end of the thesis.

CHAPTER 2

LITERATURE REVIEW

Literature review is essential when conducting research. It helps to establish a solid understanding of the topic and acknowledge the contributions of other scholars in the field. It also helps to identify contradictions, research gaps, conflicting findings, and unresolved issues that exist within the current body of knowledge. This information guides the researcher in formulating research questions and designing studies that address these gaps and contribute to the advancement of knowledge in the field.
This chapter is categorized into two sections: (1) previous reported work, and (2) technical gaps.

2.1 PREVIOUS REPORTED WORK:

Y. d. Ykuntam, K. Pavani and K. Saladi [1], proposed new and innovative architecture for the Wallace tree multiplier, which employs parallel prefix adders (PPAs) to perform the final addition step of partial products. They put forward five distinct structures for the Wallace tree multiplier, each incorporating a different type of adder.Detailed analysis of the proposed designs in terms of area and delay compared to traditional multiplier designs is completed.

S. Lad and V. S. Bendre [2], present an in-depth analysis of several Vedic sutras and their application in multiplier design. They discuss the design methodology, implementation details, and optimization techniques used to achieve high-performance multipliers. The performance metrics considered include area utilization, power consumption, and speed. The designs for the 16-bit sutras are implemented using Verilog language and evaluated. Comparative analysis is conducted with existing research work to assess the performance of the proposed sutras for better speed, area and power.

M. Kivi Sona and V. Somasundaram [3], propose a multiplier using Vedic mathematics based on Nikhilam architecture to improve the speed of operation and compares different architectures, including the existing Wallace Tree and Vedic mathematics based on Urdhva Tiriyagbhyam, with the proposed design to evaluate metrics such as area utilization, power consumption, and speed

T. Gupta and J. B. Sharma [4], The authors propose a Han-Carlson adder-based Vedic multiplier architecture that utilizes the Urdhva-Tiryakbhyam sutra from Vedic multiplication and the Han-Carlson adder, known for its high-speed performance. The authors implement a 64x64 bit Vedic multiplier using VHDL and compare the implementation results with conventional Vedic multipliers employing different adders. The results show that the proposed architecture offers improved delay, reduced hardware utilization (LUTs), and lower complexity

N. Kumar M., R. S. Adithyaa, B. Kumar D. and T. Pavithra [5], proposes a 16x16-bit Wallace Tree Multiplier using the Kogge Stone Adder and modified approximate Full Adder for improved performance. Performance analysis considers factors such as speed, area utilization, and power consumption.

A. Raju and S. K. Sa [6], The research focuses on using the Kogge Stone Adder (KSA) as a key component in multiplier architectures. The paper investigates different multiplier designs based on the KSA, exploring their performance characteristics and comparing them to conventional multiplier architectures. The author proposes multipliers utilizing the KSA, which exhibit improved performance in terms of speed and efficiency compared to conventional multiplier architectures.

P. Gulati, H. Yadav and M. K. Taleja [7], present a comparison of the three adders based on several performance metrics. The results obtained from the simulation and analysis provide insights into the performance trade-offs and advantages of using each adder. The paper contributes to the understanding of the performance characteristics of different adders in Vedic multiplier designs, which can help guide the selection and implementation of appropriate adders for Vedic multipliers in various applications.

S. Dubey and G. Verma [8], presents a comparison of different 4-bit adders in terms of power consumption, device utilization, and delay. The focus is on assessing the performance of the adders and identifying the adder that provides the best overall results.The Han Carlson Adder (HCA) is the best choice among the 4-bit adders evaluated for power consumption, device utilization, and delay. It operates based on parallel prefix computation and requires less area than the RCA, making it an efficient and effective solution for achieving high-quality results with reduced resource utilization.

R. Shanmuganathan and K. Brindhadevi [9], This study compares three different types of multiplier architectures using VHDL code and low power design methods. The Wallace tree multiplier is chosen for its high-speed operation. The Baugh Wooley multiplier is used for signed multiplication with reduced delay. The array multiplier is considered for its impact on delay, with less significance on area. This study contributes to the understanding of various multiplier architectures and their suitability for different application scenarios.

R. Anjana, B. Abishna, M. S. Harshitha, E. Abhishek, V. Ravichandra and M. S. Suma [10], present a novel approach to designing a Vedic multiplier by incorporating a Kogge-Stone adder. The results demonstrate that the Vedic multiplier with the Kogge-Stone adder outperforms the conventional multipliers on the metrices of speed and area, contributing to faster computation and improved performance. The experimental analysis reveals that the proposed Vedic multiplier achieves a balance between speed and area efficiency, making it suitable for various applications, including digital signal processing.

A. Sundhar, S. D. Tharshini, G. Priyanka, S. Ragul and C. Saranya [11], proposes a 16x16 multiplier architecture uses a 15-4 compressor and Kogge-Stone adder to enhance its performance. Results indicate that the proposed multiplier architecture using the 15-4 compressor and Kogge-Stone adder outperforms the multiplier architecture with a parallel adder in terms of speed. The Kogge-Stone adder contributes to faster computation, making the proposed architecture more efficient. These findings suggest potential applications in fields such as video and image processing.

M. N. Chandrashekara and S. Rohith [12], presents a promising approach to designing efficient multipliers using Vedic mathematics and the Modified Carry Save Adder. The objective is to develop a multiplier that can compute the product of two 8-bit binary numbers with high speed and efficiency. The results demonstrate that the Vedic multiplier utilizing the Urdhva Tiryagbhyam sutra with MCSA outperforms other multipliers in terms of speed

CHAPTER 3

CHARACTERISTIC COMPARISON FOR DIFFERENT ADDERS

The adder is an essential part of digital circuits and is used for a variety of tasks, from simple computations to complicated ones, including basic arithmetic. It is crucial in many areas, including computer architecture, signal processing, cryptography, , because it enables effective data manipulation, processing, and arithmetic operations.

The performance of the multiplier depends on the speed and efficiency of the adder used in the design. Faster and more efficient adders can significantly improve the performance of the multiplier and reduce the overall delay in the computation. Therefore, the design and implementation of the adder are crucial to achieving high-speed and high-performance multipliers. A parallel prefix adder (PPA) is a type of adder circuit that is used to perform fast and efficient addition of binary numbers.. Careful consideration and selection of the appropriate adder architecture are crucial for achieving high-performance multipliers.

This chapter is organized in five sections:

- The basic adder Ripple Carry adder, which is defined in section 3.1.
- A discussion of the parallel prefix adder and its kinds is covered in Section 3.2.
- The implementation and analysis of Kogge Stone adder and Han Carlson adder are discussed in Section 3.3 and 3.4 respectively.
- Section 3.5 of the chapter discusses comparisons and the results

3.1 Ripple Carry Adder

A ripple carry adder is a fundamental adder to add binary digits. The carry bit ripples or propagates through the stages of the adder, hence the name "ripple carry" adder. A chain of numerous full adders joined together forms the ripple carry adder. A carry-in bit, the matching bits from the numbers being added, two input bits, and the whole adder are all required. The carry from each full adder is fed into the carry-in for the next full adder in the chain as shown in Fig.3.1

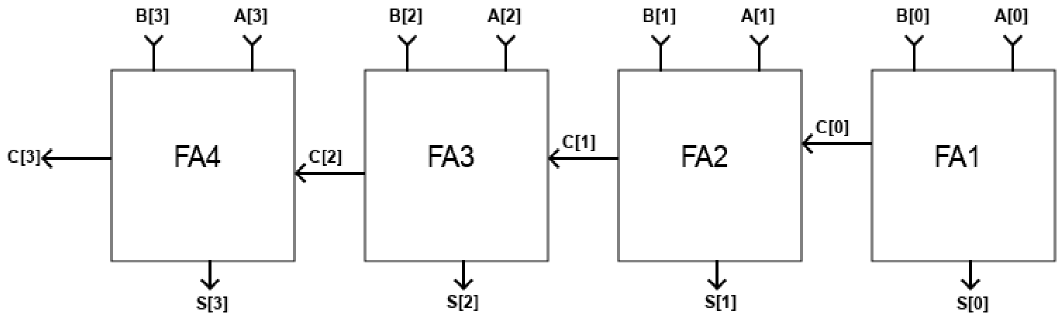


Fig. 3.1 Block diagram of Ripple Carry adder

3.1 Parallel Prefix Adders

Parallel Prefix Adders (PPA) are fast adders that are derived from Carry Look Ahead Adders. In this paper, PPA is used because it can perform parallel addition which means the partial additions can be computed parallel resulting in a significant reduction in the addition time [6]. This is in contrast with the traditional Ripple Carry Adder which performs Sequential addition and the consequent stage has to wait for the previous stage to complete.

Each stage in the parallel prefix adder performs parallel operations on different subsets of the input bits. The final carry-out is then calculated by prefixing the carry-out bits from each stage. The parallel prefix adder works by performing parallel computations within each group to calculate the carries. These computed carries are then passed to the next group, where they are combined with the corresponding inputs to compute the next level of carries. This process continues until the final carry-out bits are obtained.

Parallel Prefix Adder typically consists of 6 stages:

- Input preparation: The binary numbers to be performed operation are spilt into groups of bits. Each groups have a subset of input number.
- Perform parallel computation: To calculate the carries inside each group, parallel computations are used. Depending on the particular parallel prefix adder architecture, several methods, such as AND gates, XOR gates, and other logic circuits, can be used to do the carry calculation.
- Carry Propagation: To calculate the carries inside each group, parallel computations are used. Depending on the particular parallel prefix adder architecture, several methods, such as AND gates, OR gates, and other logic circuits, can be used to do the carry calculation.

- Carry Combination: The final carry-out bits are created by prefixing the carry bits from each group. This entails creating a precise pattern, frequently like a tree, connecting the carry outputs of each group to the inputs of the higher-order groups.
- Generate Sum bits: The input bits and the carry bits are combined in a straightforward XOR procedure to determine the sum bits. For each bit position, this can be done in parallel.
- Final Carry Out: The most significant carry-out piece of the addition operation is represented by the carry-out from the highest-order group.

The two main parallel prefix adders are explored in this chapter. Kogge Stone adder and Han Carlson adder are most efficient and faster adders in parallel prefix adder.

3.3 Kogge Stone Adder

The Kogge-Stone adder is a type of parallel prefix adder that efficiently computes the carry bits in parallel and combines them in a prefix manner to perform fast addition of binary numbers. In contrast to ripple carry adders, the Kogge-Stone adder uses parallelism and effective carry propagation to produce quicker addition. By parallelizing carry computations and reducing the ripple carry effect, it decreases the critical path delay.

Kogge Stone adder can broadly split into three stages. The first stage is the processing stage, in which the propagate and generate signals are calculated using each bit of the pair-bit signal.

$$P_i = A_i \text{ xor } B_i$$

$$G_i = A_i \text{ and } B_i$$

The second step generates carries by utilizing each bit individually and in parallel. This is utilized for carry generation and carries propagation in the intermediate stage logically and is referred to as the carry generation stage.

$$CP_{i:j} = P_{i:k+1} \text{ and } P_{k:j}$$

$$CGi:j = G_{i:k+1} \text{ or } (P_{i:k+1} \text{ and } G_{k:j})$$

In last step, sum and carry are calculated and this step is known as post processing stage.

$$Si = P_i \text{ xor } Ci\text{-}1$$

$$C_{i-1} = (P_i \text{ and } C_i) \text{ or } G_i$$

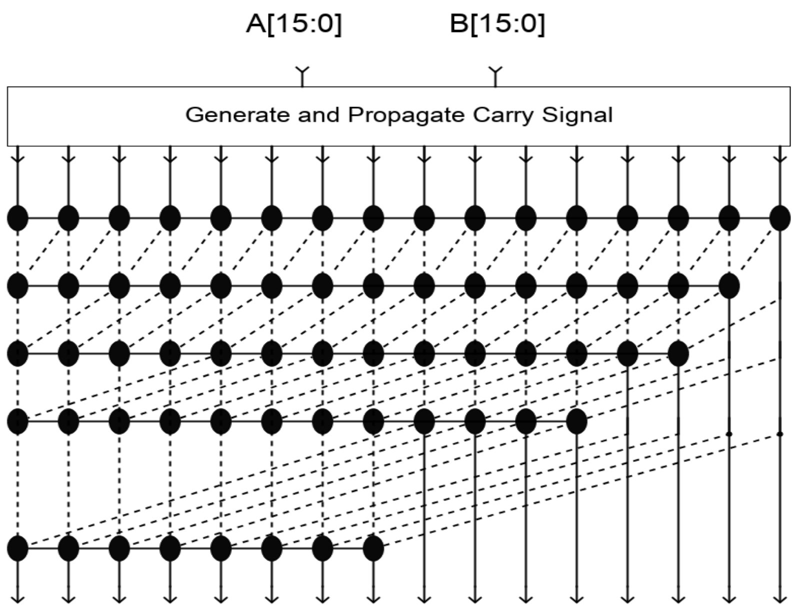


Fig. 3.2 Graphical Representation of 16-bit Kogge Stone adder

3.4 Han Carlson Adder

Han Carlson adder is similar to Kogge Stone adder. The adder has the benefit of using fewer cells and being shorter. To calculate the odd numbered prefixes, the Han-Carlson adder starts with a Brent-Kung stage, moves through Kogge-Stone stages, and finishes with another Brent-Kung stage. As a result, the complexity is reduced at the cost of an extra stage merging the carry. Due to fewer cells, the speed is fastest in all parallel prefix adder. The graphical representation of the Han Carlson adder is shown in Fig.3.3.

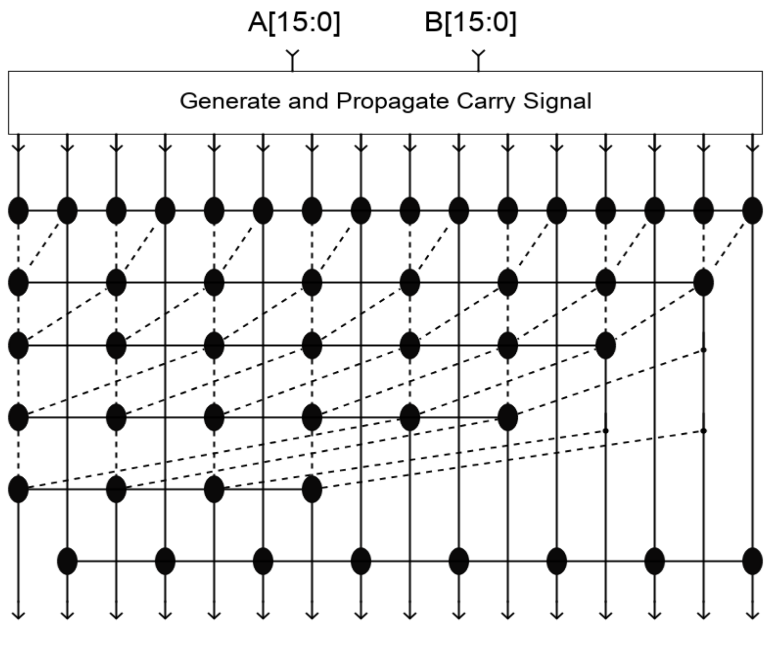
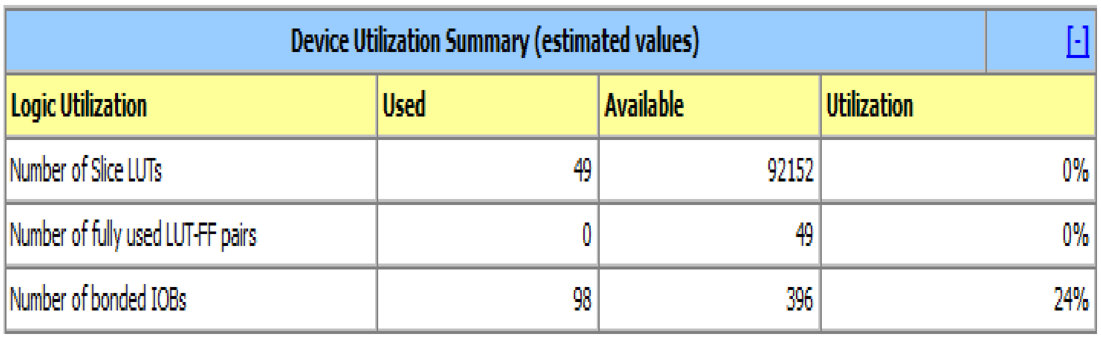


Fig. 3.3 Graphical Representation of 16-bit Han Carlson adder

3.5 Simulation and Synthesis Results

In this section a comparative analysis of 32-bits Ripple Carry Adder, Kogge Stone Adder and Han Carlson Adder are performed. The factors that are considered for comparison are area and delay. Through this detailed analysis, we aim to provide insights into the performance characteristics of different adders and assist in making informed design choices for multiplier implementations.

3.5.1 Ripple Carry Adder



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	49	92152	0%
Number of fully used LUT-FF pairs	0	49	0%
Number of bonded IOBs	98	396	24%

Fig 3.4 : Device utilization of Ripple Carry Adder

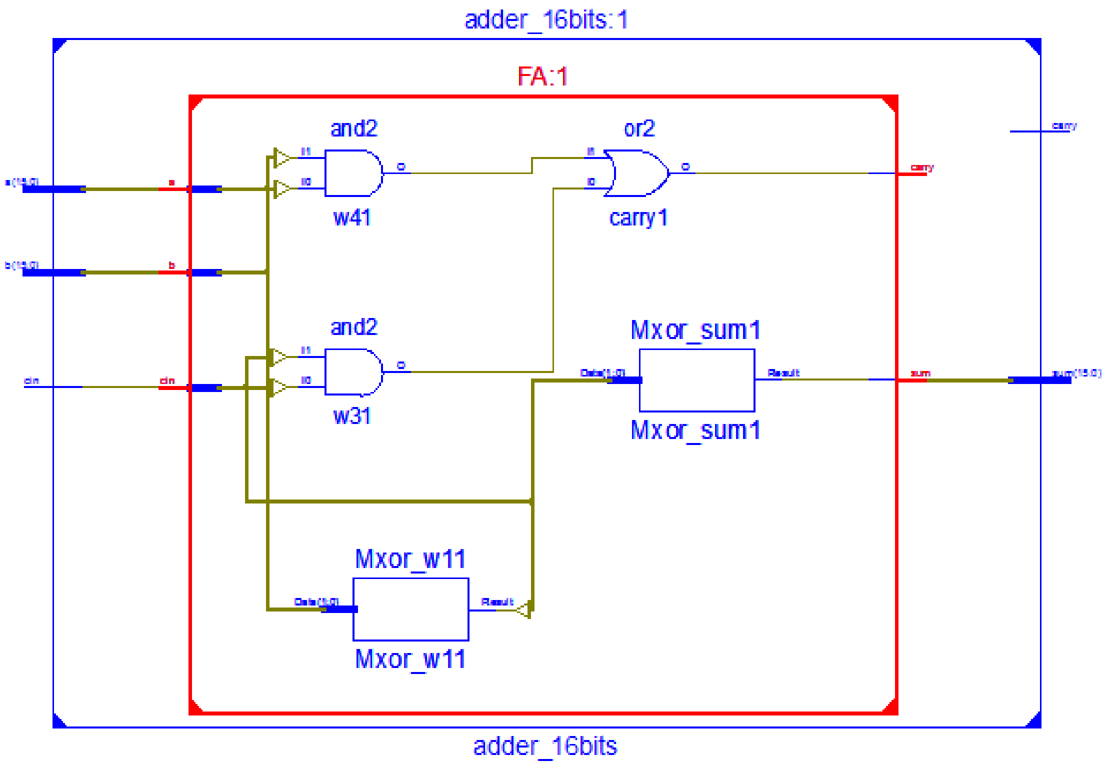
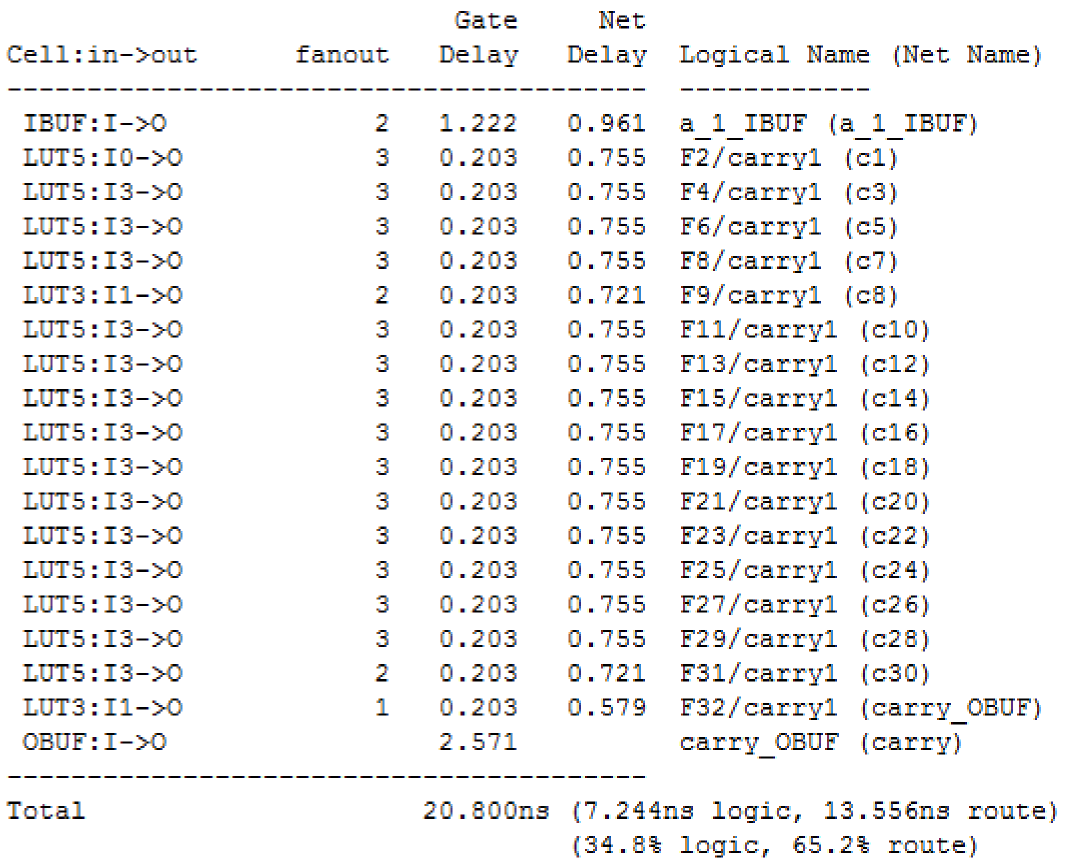


Fig 3.5 : Schematic of Ripple Carry Adder



```
                                   Gate     Net
Cell:in->out        fanout   Delay   Delay  Logical Name (Net Name)
-------------------------------------------  ------------
 IBUF:I->O               2   1.222   0.961  a_1_IBUF (a_1_IBUF)
 LUT5:I0->O              3   0.203   0.755  F2/carry1 (c1)
 LUT5:I3->O              3   0.203   0.755  F4/carry1 (c3)
 LUT5:I3->O              3   0.203   0.755  F6/carry1 (c5)
 LUT5:I3->O              3   0.203   0.755  F8/carry1 (c7)
 LUT3:I1->O              2   0.203   0.721  F9/carry1 (c8)
 LUT5:I3->O              3   0.203   0.755  F11/carry1 (c10)
 LUT5:I3->O              3   0.203   0.755  F13/carry1 (c12)
 LUT5:I3->O              3   0.203   0.755  F15/carry1 (c14)
 LUT5:I3->O              3   0.203   0.755  F17/carry1 (c16)
 LUT5:I3->O              3   0.203   0.755  F19/carry1 (c18)
 LUT5:I3->O              3   0.203   0.755  F21/carry1 (c20)
 LUT5:I3->O              3   0.203   0.755  F23/carry1 (c22)
 LUT5:I3->O              3   0.203   0.755  F25/carry1 (c24)
 LUT5:I3->O              3   0.203   0.755  F27/carry1 (c26)
 LUT5:I3->O              3   0.203   0.755  F29/carry1 (c28)
 LUT5:I3->O              2   0.203   0.721  F31/carry1 (c30)
 LUT3:I1->O              1   0.203   0.579  F32/carry1 (carry_OBUF)
 OBUF:I->O                   2.571          carry_OBUF (carry)
-------------------------------------------
Total                      20.800ns (7.244ns logic, 13.556ns route)
                                    (34.8% logic, 65.2% route)
```

Fig 3.6 : Delay Report of Ripple Carry Adder

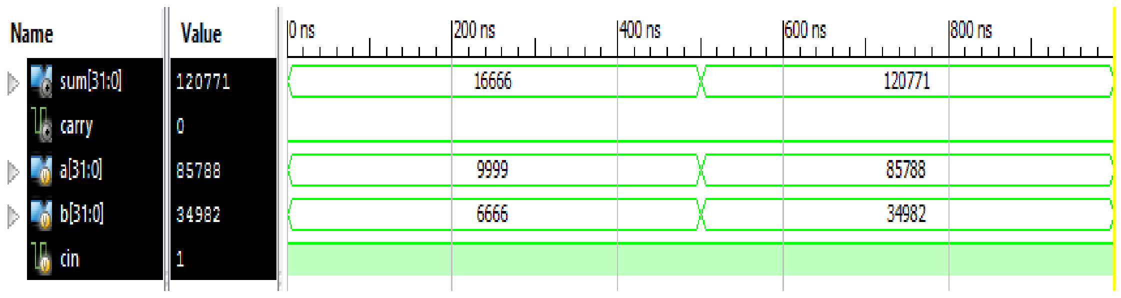
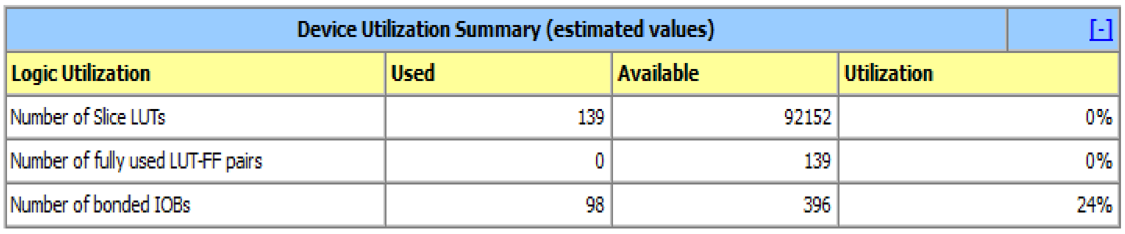


Fig 3.7: Simulation Result of Ripple Carry Adder

3.5.2 Kogge Stone Adder



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	139	92152	0%
Number of fully used LUT-FF pairs	0	139	0%
Number of bonded IOBs	98	396	24%

Fig 3.8: Synthesis Result of Kogge Stone Adder

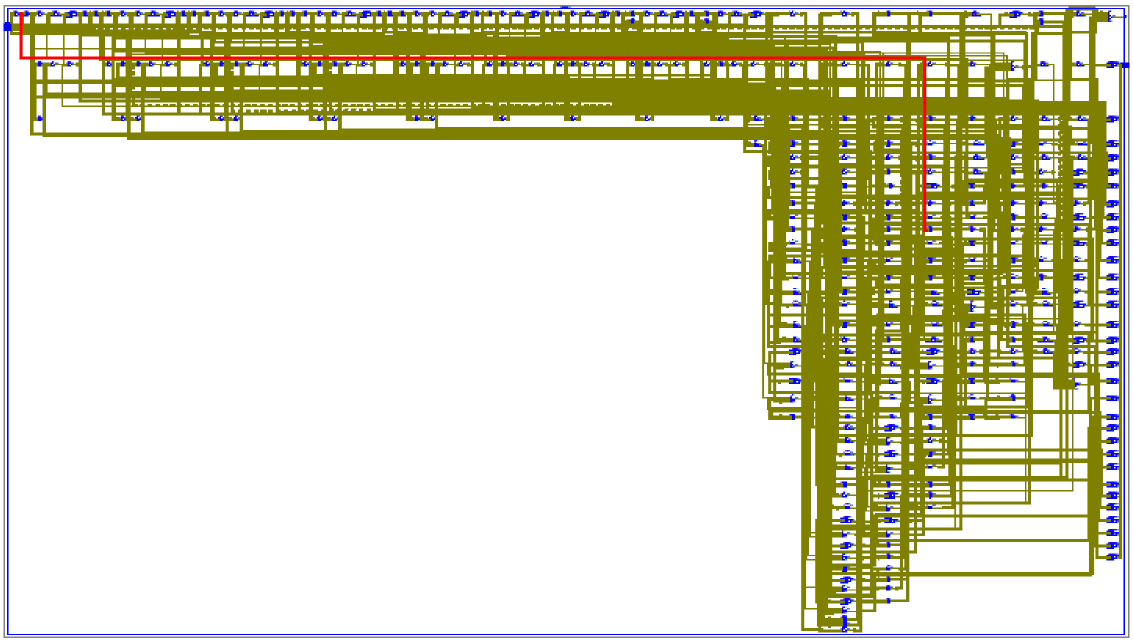


Fig 3.9: : Schematic of Kogge Stone Adder

```
                                    Gate     Net
Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
 IBUF:I->O             5   1.222   0.819  a_2_IBUF (a_2_IBUF)
 LUT2:I0->O            2   0.203   0.981  Mxor_p2_xo<0>1 (p2)
 LUT6:I0->O            4   0.203   0.912  c2 (c2)
 LUT5:I2->O            3   0.205   0.879  c6 (c6)
 LUT6:I3->O            2   0.205   0.721  c141 (c141)
 LUT4:I2->O            2   0.203   0.845  c142 (c14)
 LUT6:I3->O            2   0.205   0.721  c221 (c221)
 LUT4:I2->O            1   0.203   0.808  c222 (c22)
 LUT6:I3->O            1   0.205   0.580  c303_SW0 (N14)
 LUT6:I5->O            1   0.205   0.580  c303 (c30)
 LUT3:I2->O            1   0.205   0.579  Mxor_sum<31>_xo<0>1 (sum_31_OBUF)
 OBUF:I->O                 2.571          sum_31_OBUF (sum<31>)
----------------------------------------
Total                     14.260ns (5.835ns logic, 8.425ns route)
                                   (40.9% logic, 59.1% route)
```

Fig 3.10: Delay Report of Kogge Stone Adder

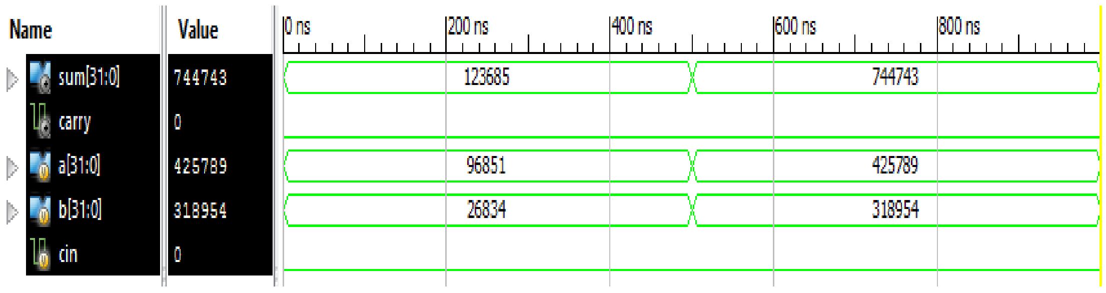
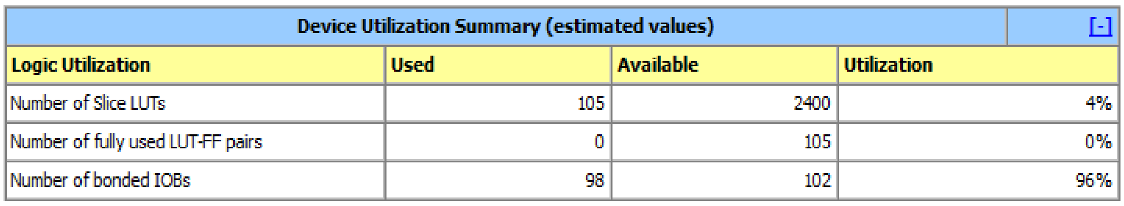


Fig 3.11: : Simulation Result of Kogge Stone Adder

3.5.3 Han Carlson Adder



Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	105	2400	4%
Number of fully used LUT-FF pairs	0	105	0%
Number of bonded IOBs	98	102	96%

Fig 3.12 : Device utilization of Han Carlson Adder

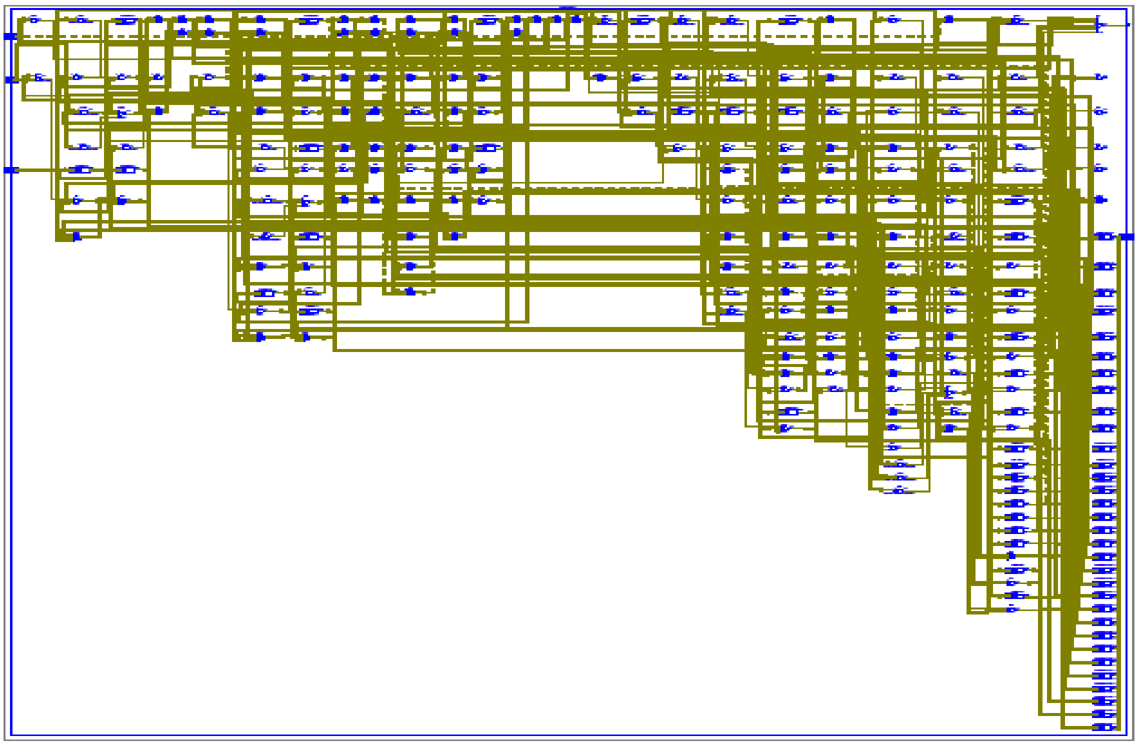


Fig 3.13 : Schematic of Han Carlson Adder

```
                                Gate     Net
Cell:in->out          fanout   Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
 IBUF:I->O                 3   1.222   0.898  a_5_IBUF (a_5_IBUF)
 LUT4:I0->O                3   0.203   0.995  p4_51 (p4_5)
 LUT5:I0->O                3   0.203   0.879  p4_71 (p4_7)
 LUT3:I0->O                1   0.205   0.924  g0_73 (g0_7)
 LUT6:I1->O                3   0.203   0.879  c111 (c11)
 LUT5:I2->O                3   0.205   0.879  c191 (c19)
 LUT6:I3->O                2   0.205   0.845  c271 (c27)
 LUT3:I0->O                1   0.205   0.579  Mxor_sum<28>_xo<0>1 (sum_28_OBUF)
 OBUF:I->O                     2.571          sum_28_OBUF (sum<28>)
----------------------------------------
Total                         12.101ns (5.222ns logic, 6.879ns route)
                                       (43.2% logic, 56.8% route)
```

Fig 3.14 : Delay Report of Han Carlson Adder

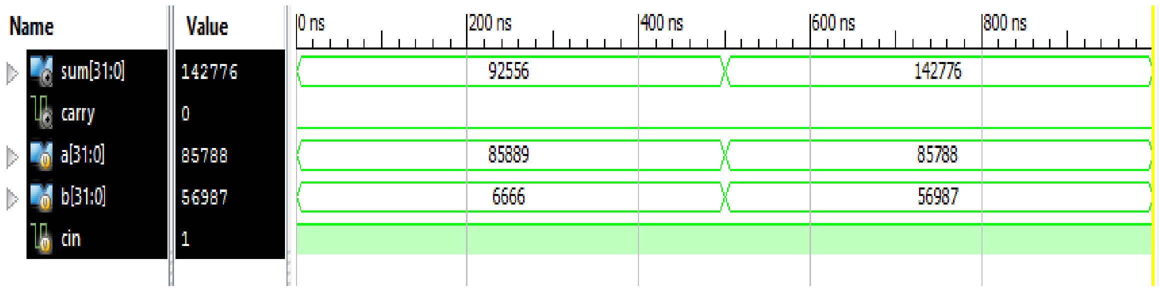


Fig 3.15: Simulation Result of Han Carlson Adder

Table 3.1 shows Comparative Analysis of Delay and Area of Adders

Adders	Delay(ns)	Area(LUT)
Ripple Carry Adder	20.8	49
Kogge Stone Adder	14.26	139
Han Carlson Adder	12.101	105

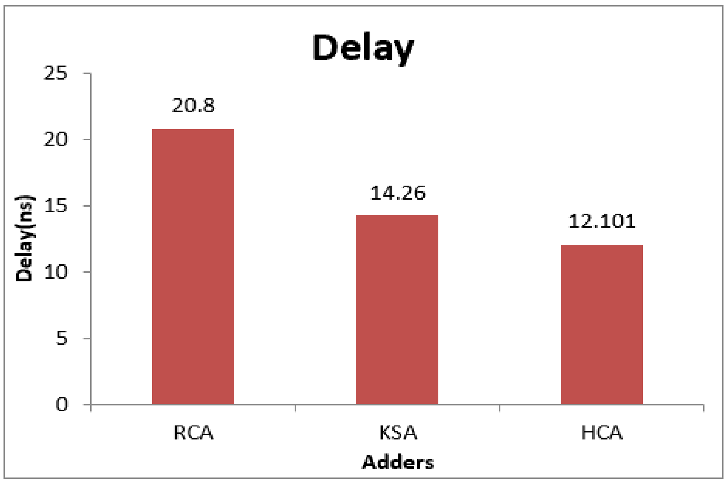


Fig 3.16. Delay comparison graph for Adders

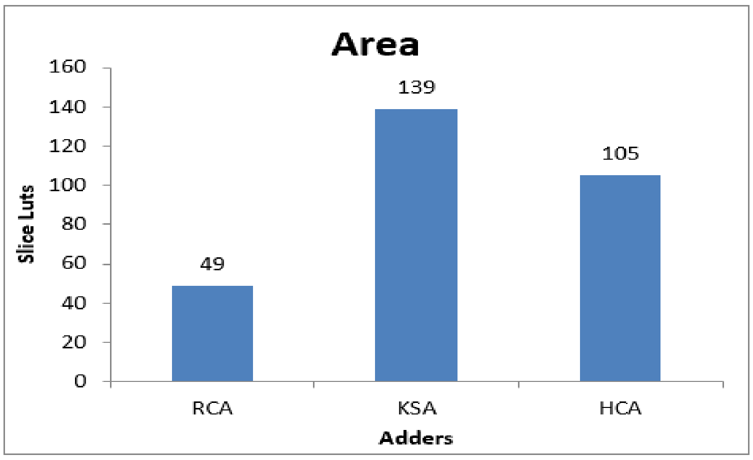


Fig 3.17. Area comparison graph for Adders

3.6 Important Outcomes:

In this chapter, the design and analyses of several adders are carried out and compared to determine the optimal option for the best performance.

- The Ripple Carry, Kogge Stone and Han Carlson adder are design and simulated.
- Delay and Area have been calculated and plotted.
- Performing addition using Han Carlson adder can reduce delay up to 40%.
- In terms of area, the ripple carry adder utilized space better than the parallel prefix adder

CHAPTER 4

DESIGN AND ANALYSIS OF MULTIPLIER USING DIFFERENT PARALLEL PREFIX ADDER

Multiplication operation is an essential component in a processing unit that is widely employed in numerous disciplines, including engineering, physics, computer science, and many others. Numerous multiplication operations are required by many algorithms. The effectiveness and speed of these operations directly affect the system's overall performance.

Improving the performance of a multiplier can bring several benefits, such as:

- Faster Calculation: The time needed to perform multiplication operations can be decreased by optimizing a multiplier, which could result in quicker calculation times.
- Reduced power consumption: Reduced power consumption from faster computing is also beneficial for battery-operated gadgets and low-power systems.
- Increased accuracy: In many applications, like digital signal processing and image processing, higher levels of accuracy are required, and high-performance multipliers are able to perform multiplication operations with these levels of accuracy.
- Improved system throughput: The throughput of the entire system can be boosted by optimizing the performance of a multiplier, which is crucial in high-performance computing systems.

In this chapter, we will discuss different multipliers and how to improve their performance by using a faster adder.

The following is the arrangement of this chapter:

- Efficient Multiplier to be designed and the implementation of Wallace Tree and Vedic Multiplier is described in Section 4.1.
- In section 4.2, simulated and synthesized results are presented,
- Performance comparison of the implemented Multipliers in terms of Delay and Area discussed in section 4.3.

4.1 Multiplier

Wallace tree multiplier and the Vedic Urdhva Tiryakbhyam Multiplier (UTM) are two fastest multipliers used in digital circuit. The Wallace tree multiplier is based on the Wallace tree algorithm, a quick multiplication method that minimizes the number of partial products needed to compute the final result. The Vedic UTM, on the other hand, is based on prehistoric Indian mathematics systems that use particular formulas and procedures to execute multiplication. It utilizes various sutras and sub-sutras to simplify the process by breaking it into smaller steps.

For and efficient multiplication,it is important to increase the performance of the adders involved. A series of additions are required for multiplication, and the pace at which these additions are completed directly affects the total time required for multiplication. The speed of the adder becomes significant when multiplying huge numbers or carrying out multiplication operations in high-performance computing systems. In this chapter, all the adders that were discussed in the previous chapter are used to improve the performance of multipliers. By using the faster adder in the multiplier, we can improve throughput, reduce execution time, enhance computational efficiency, and impact complex operations.

4.1.1 Wallace Tree Mutliplier

The Wallace tree multiplier is based on the concept of"tree reduction," which involves decreasing the number of partial products that must be combined to create the final product. This algorithm allows for efficient carry propagation in parallel processing. It reduces the overall area footprint of the circuit by minimizing the number of logic gates needed. It continues to be widely utilized in a range of applications where efficient and fast multiplication operations are essential.

The steps involved in a Wallace tree multiplier's operation are as follows:

- Partial products generation: A series of AND gates is used to multiply the binary digits of the multiplicand by the binary digits of the multiplier. The resulting partial products are arranged in columns.
- Reduction of partial products: To create a set of sums and carries, the partial products are reduced using a succession of half-adders and full-adders
- Final Addition: A succession of full adders is used to condense the Wallace tree into a single total.

Overall, utilizing digital circuits, the Wallace tree multiplier is a very effective approach to multiplying two binary values. The graphical representation of 8-bits Wallace tree multiplier shows in Fig.4.1

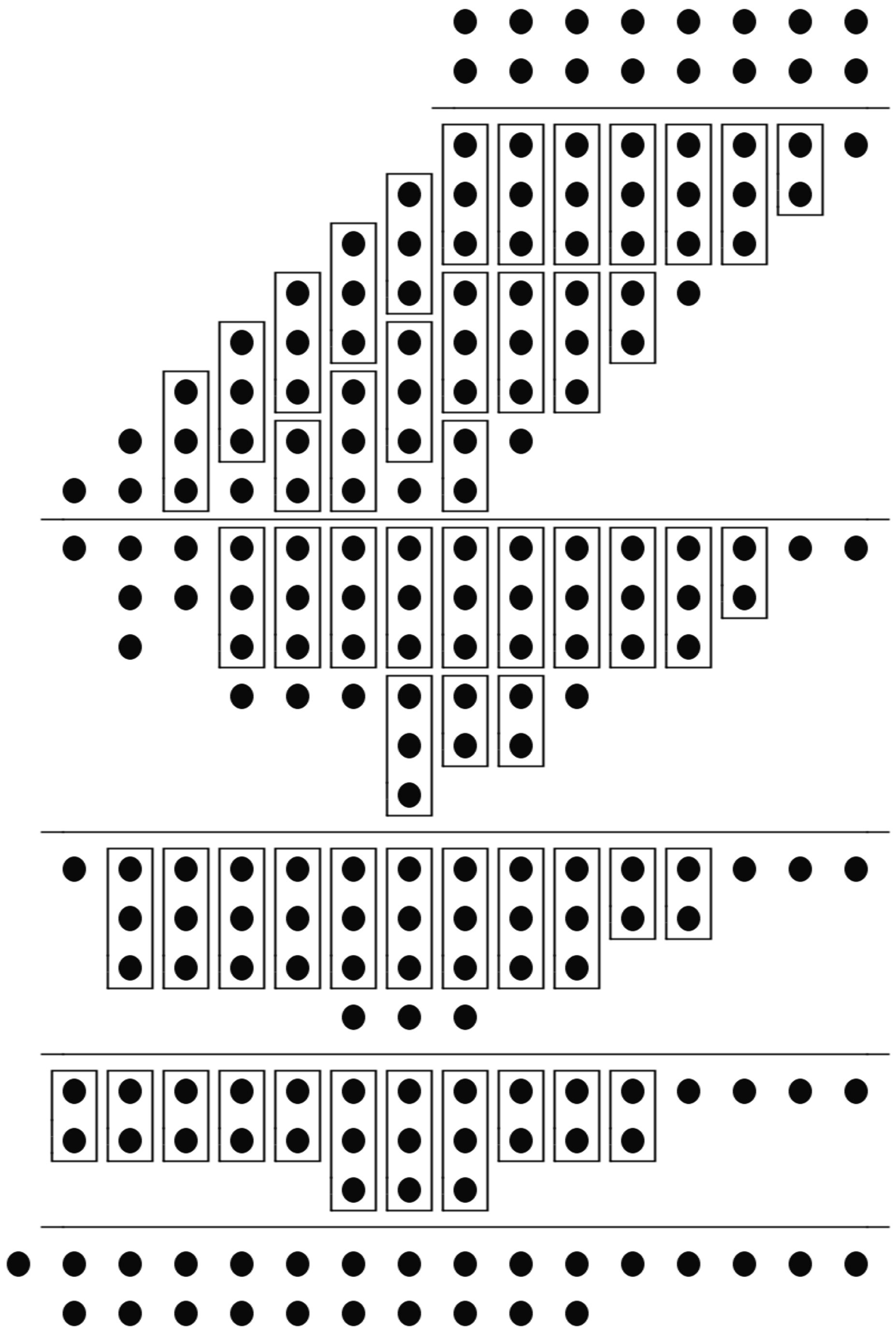


Fig. 4.1 Graphical Representation Of 8X8 bit Wallace Tree Multiplier

The Wallace Tree Multiplier is constructed by generating partial products where both the input numbers are multiplied by passing through AND gates. Further reduction of the partial products is carried out by adding them using half adders and full adders. This reduction is carried out till the last two rows are received. To achieve a faster addition, the last two rows are added using Kogge Stone adder and Han Carlson Adder (discussed in previous chapter). Fig.4.2 depicts the flow diagram for generating the Wallace Tree Multiplier.

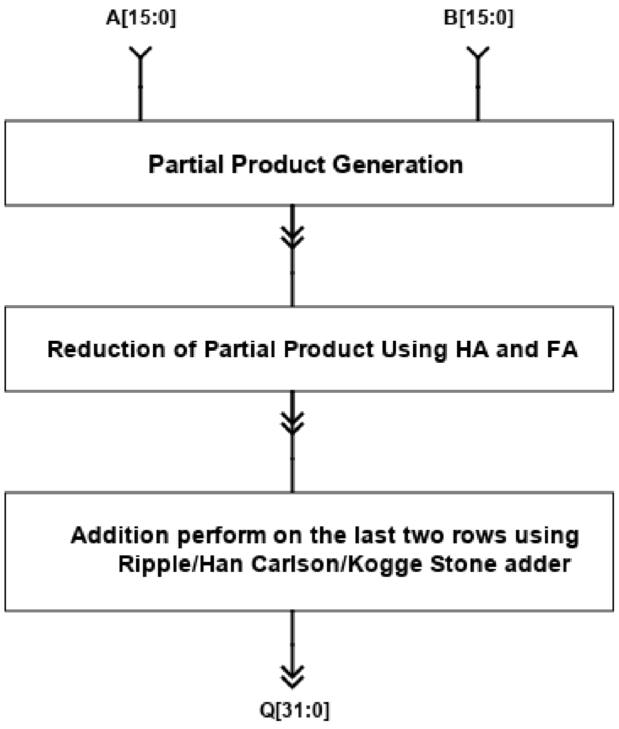

A[15:0]
B[15:0]
Partial Product Generation
Reduction of Partial Product Using HA and FA
Addition perform on the last two rows using Ripple/Han Carlson/Kogge Stone adder
Q[31:0]

Fig 4.2. Flow diagram of Wallace Tree Multiplier

The variation in performance of the Wallace Tree multiplier due to changes in the adder will be discussed in Section 4.3.

4.1.2 Urdhva-Tiryakbhyam Vedic Multiplier

The Vedic multiplier is a sort of multiplier derived from the Vedic mathematics of ancient India. It offers a different way to carry out multiplication operations using particular formulas and methods. Comparing the Vedic multiplier to conventional multiplication algorithms, there are a number of benefits, including simplicity, speed, and efficiency. To make multiplication easier, it applies a variety of sutras (formulas) and sub-sutras (corollaries) from Vedic mathematics. It is one of the most used and fastest sutras in Vedic mathematics. According to the sutra, there is a quick and easy method for multiplying two-digit numbers that entails multiplying the vertical and crosswise components and adding the results.

The basic steps of the Vedic multiplier algorithm for multiplication are as follows:

- Depending on place values, divide the multiplicand and multiplier into their appropriate halves.
- Perform cross-multiplication between the corresponding parts of the multiplicand and the multiplier. In order to do this, numbers must be multiplied both vertically and diagonally, and the partial products must be arranged in a particular way.
- Vertical additions are used to add up the cross-multiplication step's partial products.
- Any necessary carries from each vertical addition are carried forward.
- The final output is created by combining the results.

Given below is an example to illustrate multiplication:

Multiply A1A0 by B1B0

- Multiply the rightmost vertical terms A0 and B0. Write the output A0B0 on the rightmost part of the answer
- Multiply the leftmost vertical terms A1 and B1. Write the output A1B1 on the leftmost part of the answer
- Then multiply the cross-wise terms i.e.A1B0 and A0B1. Add the two results (A1B0+A0B1). If the resultant is a 2digit, then the leftmost digit of this term will be carried forward to the left
- The final result is as follows: (A1B1) +(A1B0+A0B1) +(A0B0)

The gate level implementation of 2x2 Vedic Multiplier as shown in Fig.4.3

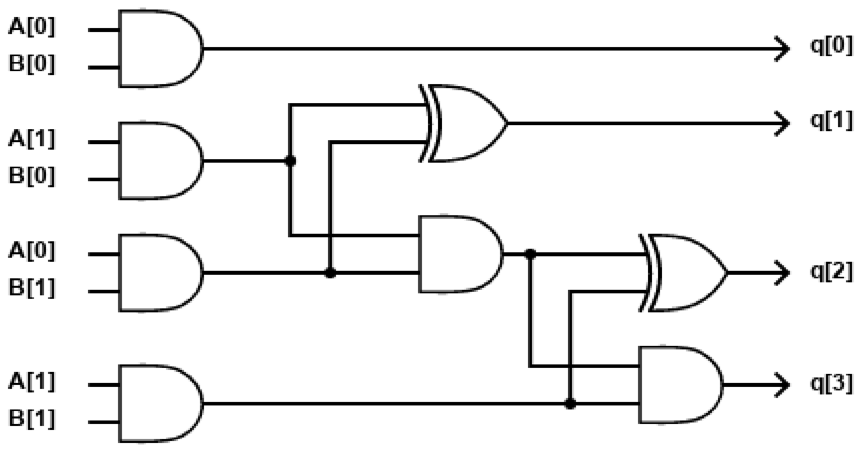


Fig.4.3: 2x2 Gate level netlist of Vedic Multiplier

In order to construct an N-bit Vedic Multiplier, the input values A and B are split into two equal parts. These parts are referred to as AH, AL, BH, and BL, representing the Most Significant Bits (MSB) and Least Significant Bits (LSB) respectively. This division allows the creation of an N x N size Vedic multiplier, as depicted in Figure 4.4. are utilized as inputs for four Vedic Multipliers, each of size N/2. These outputs are then fed into four additional Vedic Multipliers, this time with a size of N/4. The N/4 size Vedic Multipliers are further split into four Vedic Multipliers of size N/8, following a similar pattern [2]. This continues until the splitting reaches a block size of 2x2, as depicted in the block diagram shown in the figure. Finally, the outputs of the four N/2 x N/2 multipliers are inputted into an adder with (N + N/2) bits.

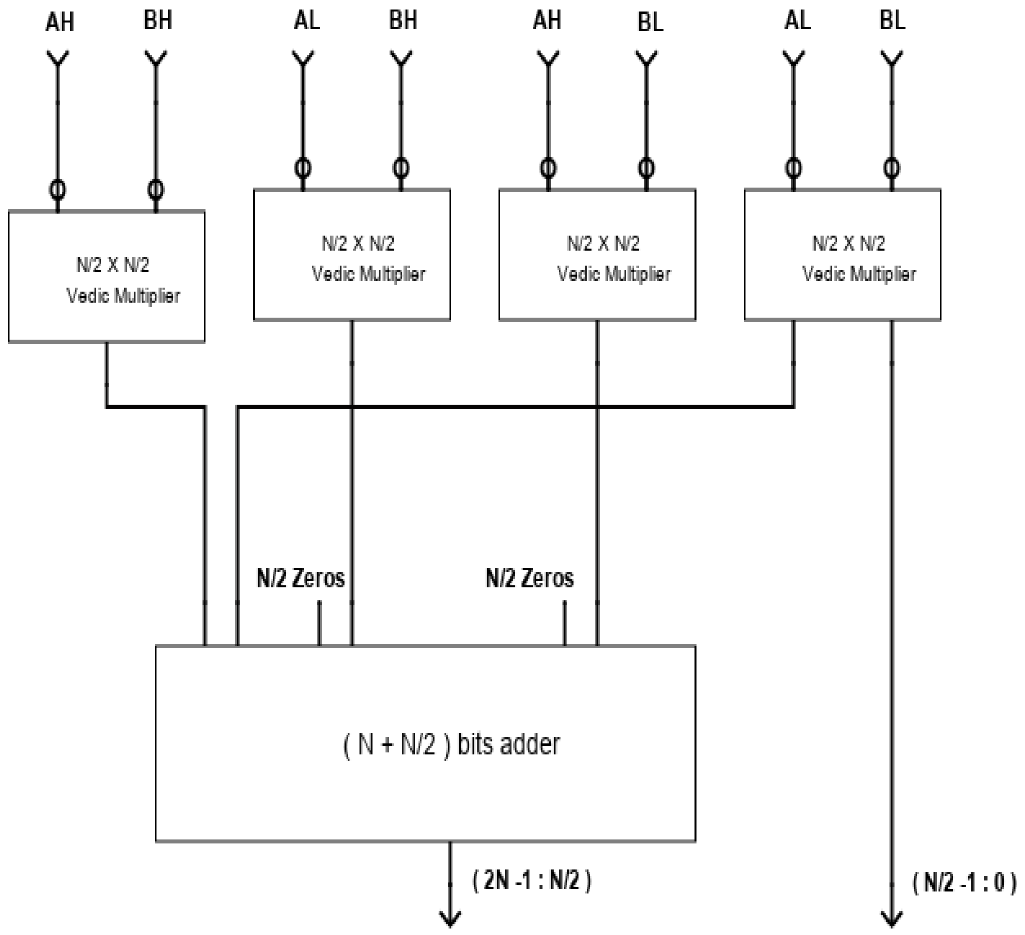
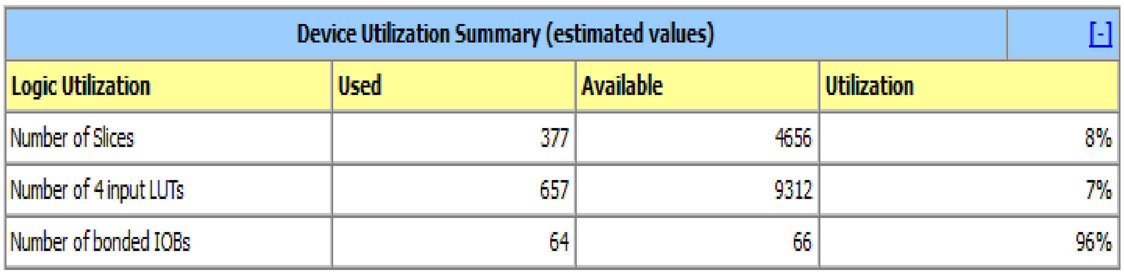

AH
BH
AL
BH
AH
BL
AL
BL
N/2 X N/2 Vedic Multiplier
N/2 X N/2 Vedic Multiplier
N/2 X N/2 Vedic Multiplier
N/2 X N/2 Vedic Multiplier
N/2 Zeros
N/2 Zeros
(N + N/2) bits adder
(2N -1 : N/2)
(N/2 -1 : 0)

Fig.4.4 Block diagram of NXN bit Vedic Multiplier

Performance of Vedic multiplier depends upon the types of adder used in the circuit. The variation in performance of the Vedic Multiplier due to changes in the adder will be discussed in next section.

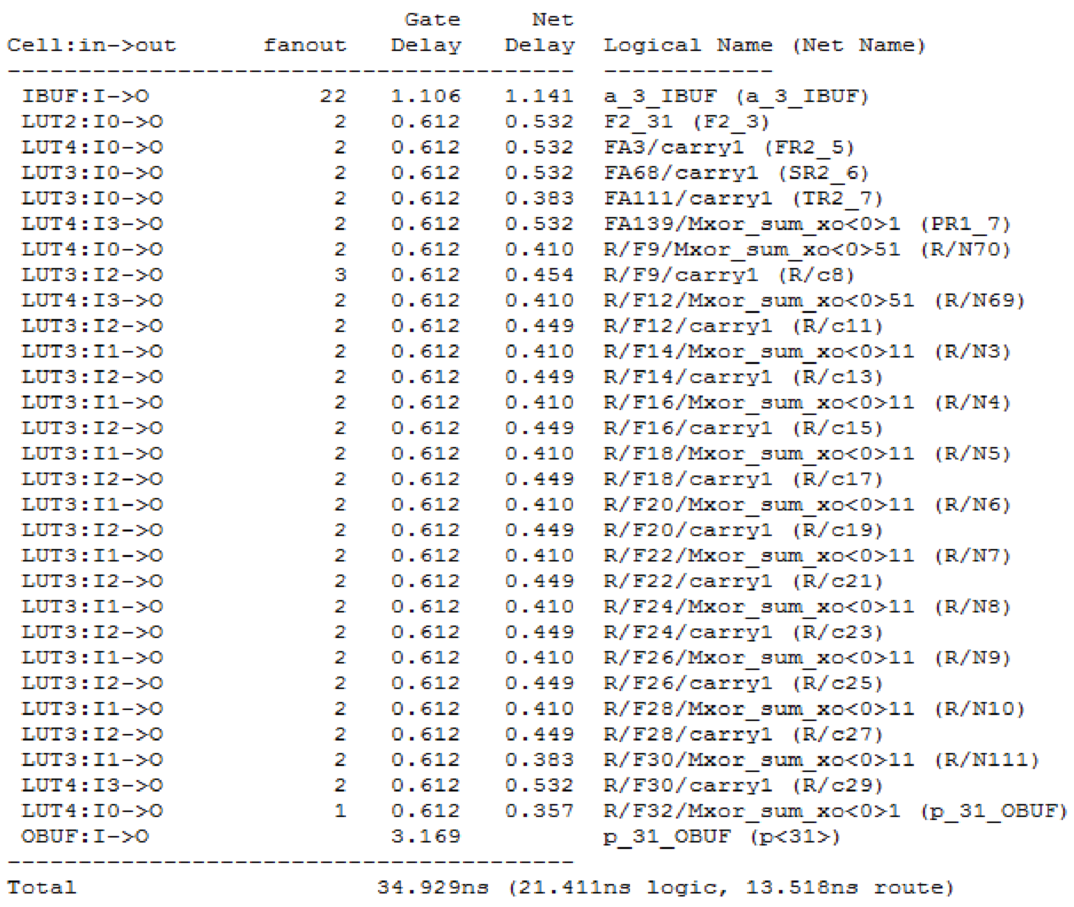
4.2 Simulation and Synthesis Results

4.2.1 Wallace Tree Multiplier using Ripple Carry Adder



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	377	4656	8%
Number of 4 input LUTs	657	9312	7%
Number of bonded IOBs	64	66	96%

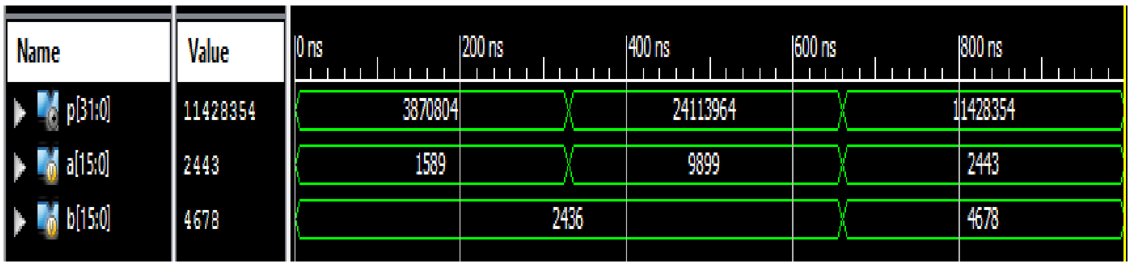
Fig.4.5. Device utilization of Wallace Tree multiplier using RCA



```
                                 Gate     Net
Cell:in->out          fanout    Delay    Delay   Logical Name (Net Name)
----------------------------------------  ------------
 IBUF:I->O              22     1.106    1.141  a_3_IBUF (a_3_IBUF)
 LUT2:I0->O              2     0.612    0.532  F2_31 (F2_3)
 LUT4:I0->O              2     0.612    0.532  FA3/carry1 (FR2_5)
 LUT3:I0->O              2     0.612    0.532  FA68/carry1 (SR2_6)
 LUT3:I0->O              2     0.612    0.383  FA111/carry1 (TR2_7)
 LUT4:I3->O              2     0.612    0.532  FA139/Mxor_sum_xo<0>1 (PR1_7)
 LUT4:I0->O              2     0.612    0.410  R/F9/Mxor_sum_xo<0>51 (R/N70)
 LUT3:I2->O              3     0.612    0.454  R/F9/carry1 (R/c8)
 LUT4:I3->O              2     0.612    0.410  R/F12/Mxor_sum_xo<0>51 (R/N69)
 LUT3:I2->O              2     0.612    0.449  R/F12/carry1 (R/c11)
 LUT3:I1->O              2     0.612    0.410  R/F14/Mxor_sum_xo<0>11 (R/N3)
 LUT3:I2->O              2     0.612    0.449  R/F14/carry1 (R/c13)
 LUT3:I1->O              2     0.612    0.410  R/F16/Mxor_sum_xo<0>11 (R/N4)
 LUT3:I2->O              2     0.612    0.449  R/F16/carry1 (R/c15)
 LUT3:I1->O              2     0.612    0.410  R/F18/Mxor_sum_xo<0>11 (R/N5)
 LUT3:I2->O              2     0.612    0.449  R/F18/carry1 (R/c17)
 LUT3:I1->O              2     0.612    0.410  R/F20/Mxor_sum_xo<0>11 (R/N6)
 LUT3:I2->O              2     0.612    0.449  R/F20/carry1 (R/c19)
 LUT3:I1->O              2     0.612    0.410  R/F22/Mxor_sum_xo<0>11 (R/N7)
 LUT3:I2->O              2     0.612    0.449  R/F22/carry1 (R/c21)
 LUT3:I1->O              2     0.612    0.410  R/F24/Mxor_sum_xo<0>11 (R/N8)
 LUT3:I2->O              2     0.612    0.449  R/F24/carry1 (R/c23)
 LUT3:I1->O              2     0.612    0.410  R/F26/Mxor_sum_xo<0>11 (R/N9)
 LUT3:I2->O              2     0.612    0.449  R/F26/carry1 (R/c25)
 LUT3:I1->O              2     0.612    0.410  R/F28/Mxor_sum_xo<0>11 (R/N10)
 LUT3:I2->O              2     0.612    0.449  R/F28/carry1 (R/c27)
 LUT3:I1->O              2     0.612    0.383  R/F30/Mxor_sum_xo<0>11 (R/N111)
 LUT4:I3->O              2     0.612    0.532  R/F30/carry1 (R/c29)
 LUT4:I0->O              1     0.612    0.357  R/F32/Mxor_sum_xo<0>1 (p_31_OBUF)
 OBUF:I->O                     3.169           p_31_OBUF (p<31>)
----------------------------------------
Total                        34.929ns (21.411ns logic, 13.518ns route)
```

Fig.4.6. Delay Report of Wallace Tree Multiplier using RCA

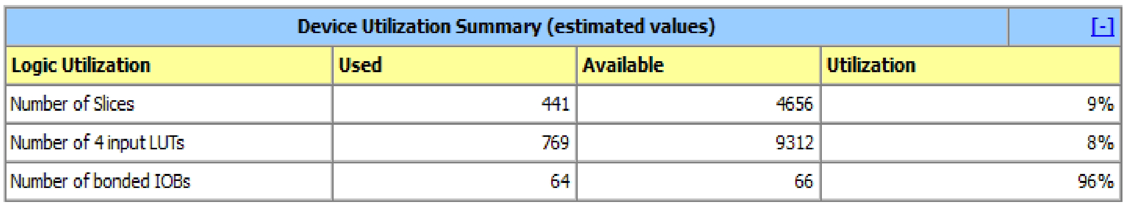
Fig.4.7 Schematic of Wallace Tree Multiplier using RCA



Name	Value	0 ns	200 ns	400 ns	600 ns	800 ns
p[31:0]	11428354	3870804		24113964		11428354
a[15:0]	2443	1589		9899		2443
b[15:0]	4678	2436				4678

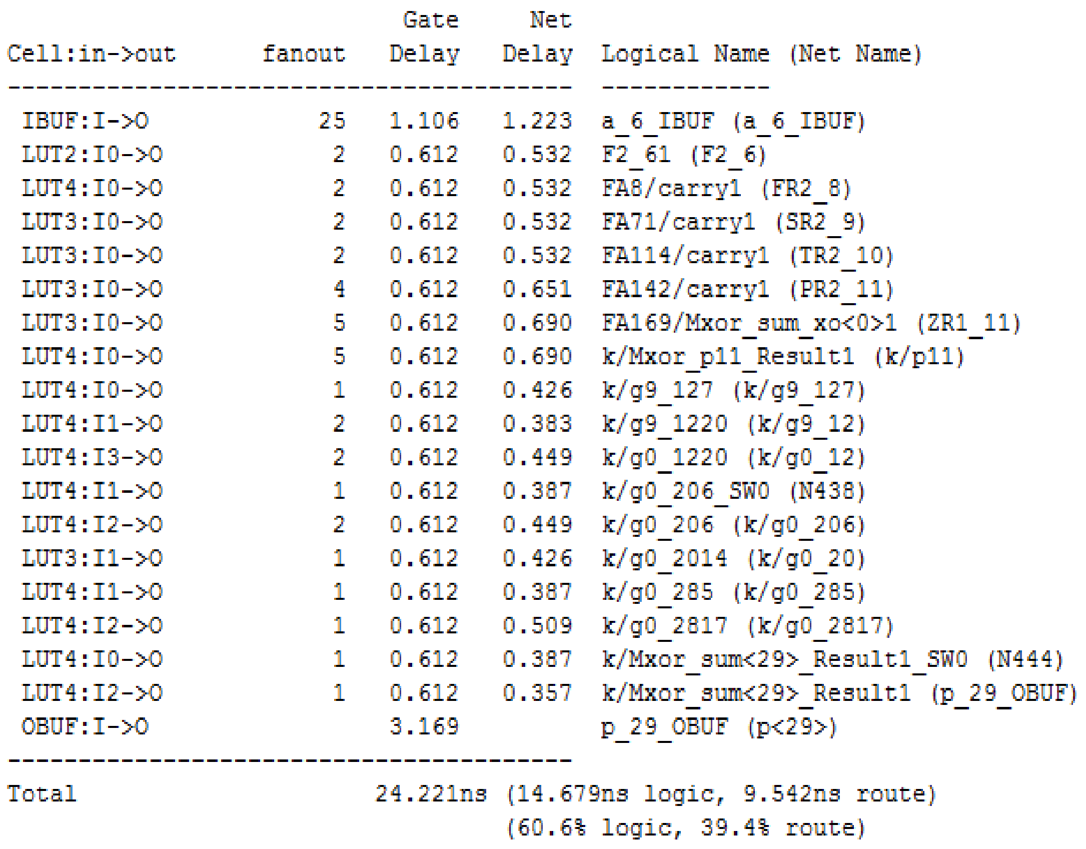
Fig.4.8. Simulation Result of Wallace Tree Multiplier using RCA

4.2.2 Wallace Tree Multiplier using Kogge Stone Adder



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	441	4656	9%
Number of 4 input LUTs	769	9312	8%
Number of bonded IOBs	64	66	96%

Fig.4.9. Device Utilization of Wallace Tree Multiplier using KSA



```
                                Gate     Net
Cell:in->out      fanout   Delay    Delay  Logical Name (Net Name)
----------------------------------------  ------------
 IBUF:I->O            25   1.106    1.223  a_6_IBUF (a_6_IBUF)
 LUT2:I0->O            2   0.612    0.532  F2_61 (F2_6)
 LUT4:I0->O            2   0.612    0.532  FA8/carry1 (FR2_8)
 LUT3:I0->O            2   0.612    0.532  FA71/carry1 (SR2_9)
 LUT3:I0->O            2   0.612    0.532  FA114/carry1 (TR2_10)
 LUT3:I0->O            4   0.612    0.651  FA142/carry1 (PR2_11)
 LUT3:I0->O            5   0.612    0.690  FA169/Mxor_sum_xo<0>1 (ZR1_11)
 LUT4:I0->O            5   0.612    0.690  k/Mxor_p11_Result1 (k/p11)
 LUT4:I0->O            1   0.612    0.426  k/g9_127 (k/g9_127)
 LUT4:I1->O            2   0.612    0.383  k/g9_1220 (k/g9_12)
 LUT4:I3->O            2   0.612    0.449  k/g0_1220 (k/g0_12)
 LUT4:I1->O            1   0.612    0.387  k/g0_206_SW0 (N438)
 LUT4:I2->O            2   0.612    0.449  k/g0_206 (k/g0_206)
 LUT3:I1->O            1   0.612    0.426  k/g0_2014 (k/g0_20)
 LUT4:I1->O            1   0.612    0.387  k/g0_285 (k/g0_285)
 LUT4:I2->O            1   0.612    0.509  k/g0_2817 (k/g0_2817)
 LUT4:I0->O            1   0.612    0.387  k/Mxor_sum<29>_Result1_SW0 (N444)
 LUT4:I2->O            1   0.612    0.357  k/Mxor_sum<29>_Result1 (p_29_OBUF)
 OBUF:I->O                 3.169           p_29_OBUF (p<29>)
----------------------------------------
Total                    24.221ns (14.679ns logic, 9.542ns route)
                                  (60.6% logic, 39.4% route)
```

Fig.4.10. Delay Report of Wallace Tree Multiplier using KSA

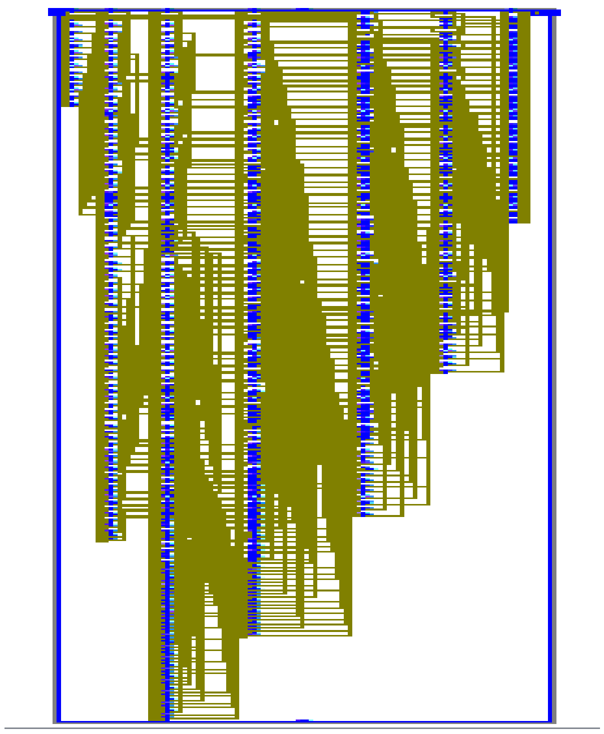


Fig.4.11. Schematic of Wallace Tree Multiplier Using KSA

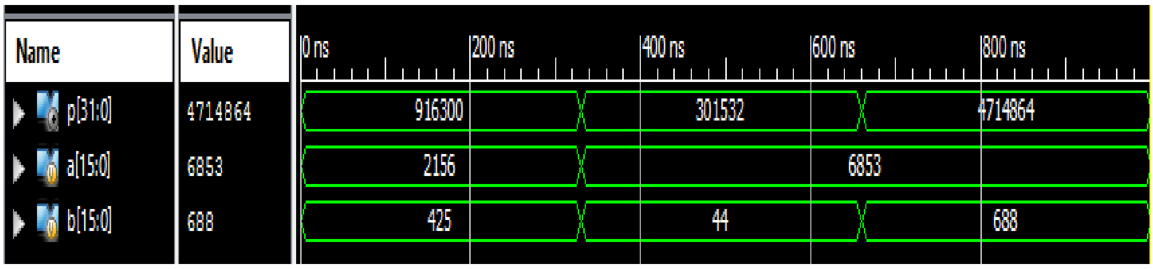
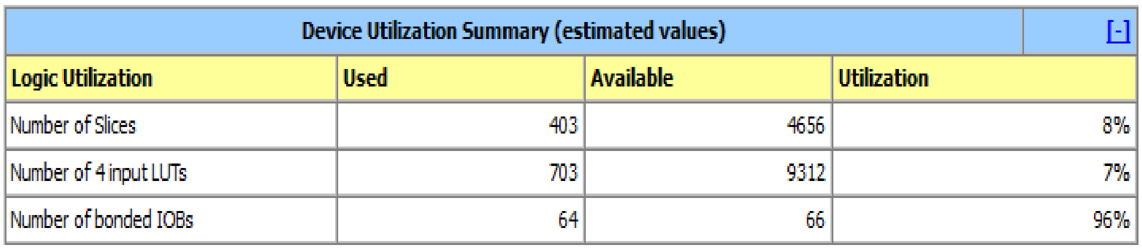


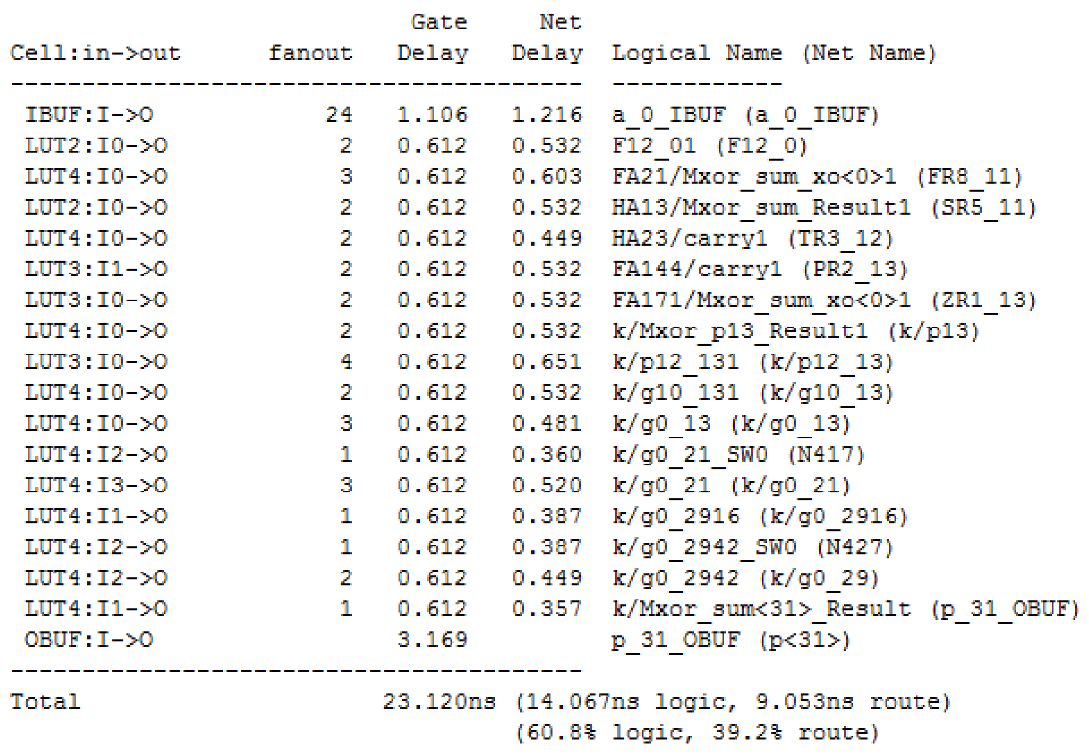
Fig.4.12. Simulation of Wallace Tree Multiplier Using KSA

4.2.3 Wallace Tree Multiplier using Ripple Han Carlson Adder



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	403	4656	8%
Number of 4 input LUTs	703	9312	7%
Number of bonded IOBs	64	66	96%

Fig.4.13 Device utilization of Wallace Tree multiplier using Han Carlson



```
                            Gate     Net
Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
 IBUF:I->O            24   1.106   1.216  a_0_IBUF (a_0_IBUF)
 LUT2:I0->O            2   0.612   0.532  F12_01 (F12_0)
 LUT4:I0->O            3   0.612   0.603  FA21/Mxor_sum_xo<0>1 (FR8_11)
 LUT2:I0->O            2   0.612   0.532  HA13/Mxor_sum_Result1 (SR5_11)
 LUT4:I0->O            2   0.612   0.449  HA23/carry1 (TR3_12)
 LUT3:I1->O            2   0.612   0.532  FA144/carry1 (PR2_13)
 LUT3:I0->O            2   0.612   0.532  FA171/Mxor_sum_xo<0>1 (ZR1_13)
 LUT4:I0->O            2   0.612   0.532  k/Mxor_p13_Result1 (k/p13)
 LUT3:I0->O            4   0.612   0.651  k/p12_131 (k/p12_13)
 LUT4:I0->O            2   0.612   0.532  k/g10_131 (k/g10_13)
 LUT4:I0->O            3   0.612   0.481  k/g0_13 (k/g0_13)
 LUT4:I2->O            1   0.612   0.360  k/g0_21_SW0 (N417)
 LUT4:I3->O            3   0.612   0.520  k/g0_21 (k/g0_21)
 LUT4:I1->O            1   0.612   0.387  k/g0_2916 (k/g0_2916)
 LUT4:I2->O            1   0.612   0.387  k/g0_2942_SW0 (N427)
 LUT4:I2->O            2   0.612   0.449  k/g0_2942 (k/g0_29)
 LUT4:I1->O            1   0.612   0.357  k/Mxor_sum<31>_Result (p_31_OBUF)
 OBUF:I->O                 3.169          p_31_OBUF (p<31>)
----------------------------------------
Total                     23.120ns (14.067ns logic, 9.053ns route)
                                   (60.8% logic, 39.2% route)
```

Fig.4.14. Delay Report of Wallace Tree multiplier using Wallace Tree HCA

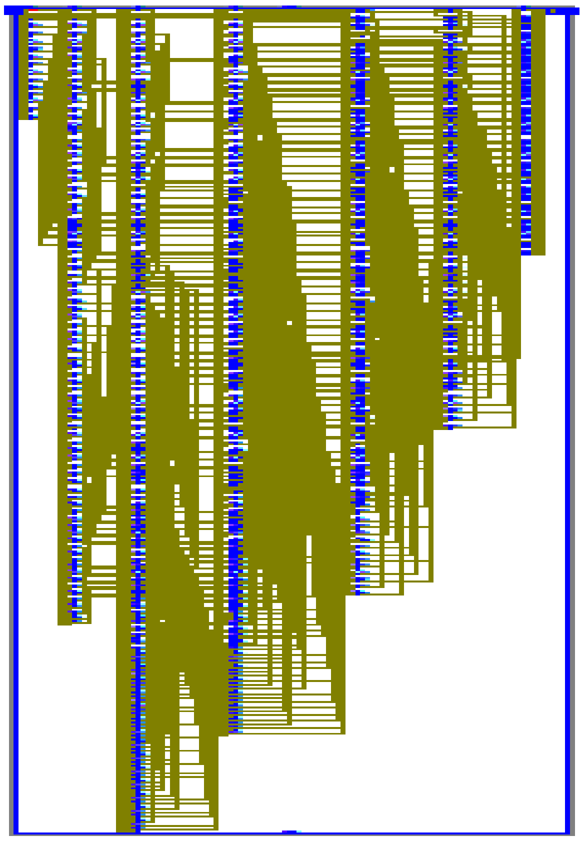
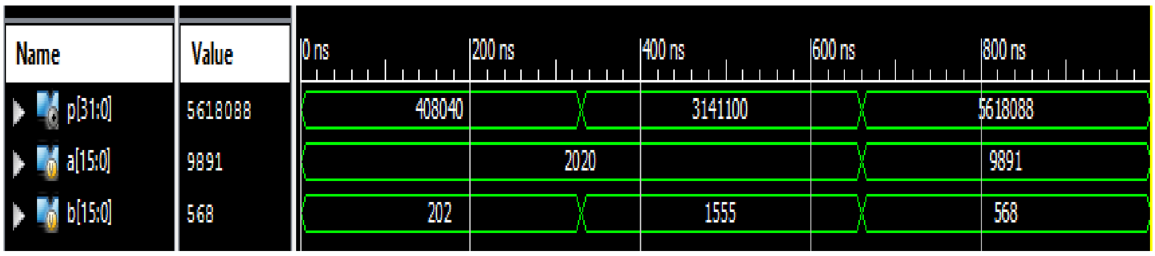


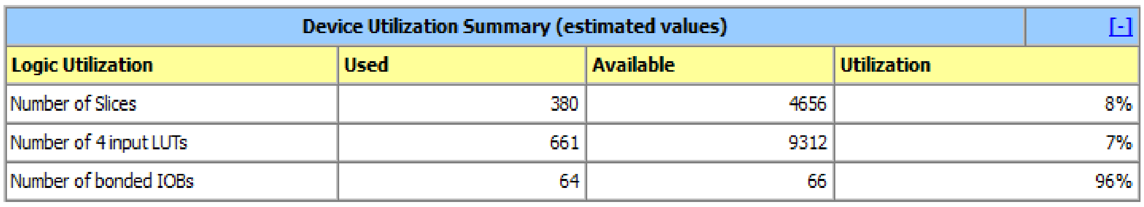
Fig.4.15. Schematic of Wallace Tree Multiplier using HCA



Name	Value	0 ns	200 ns	400 ns	600 ns	800 ns
p[31:0]	5618088	408040		3141100		5618088
a[15:0]	9891	2020				9891
b[15:0]	568	202		1555		568

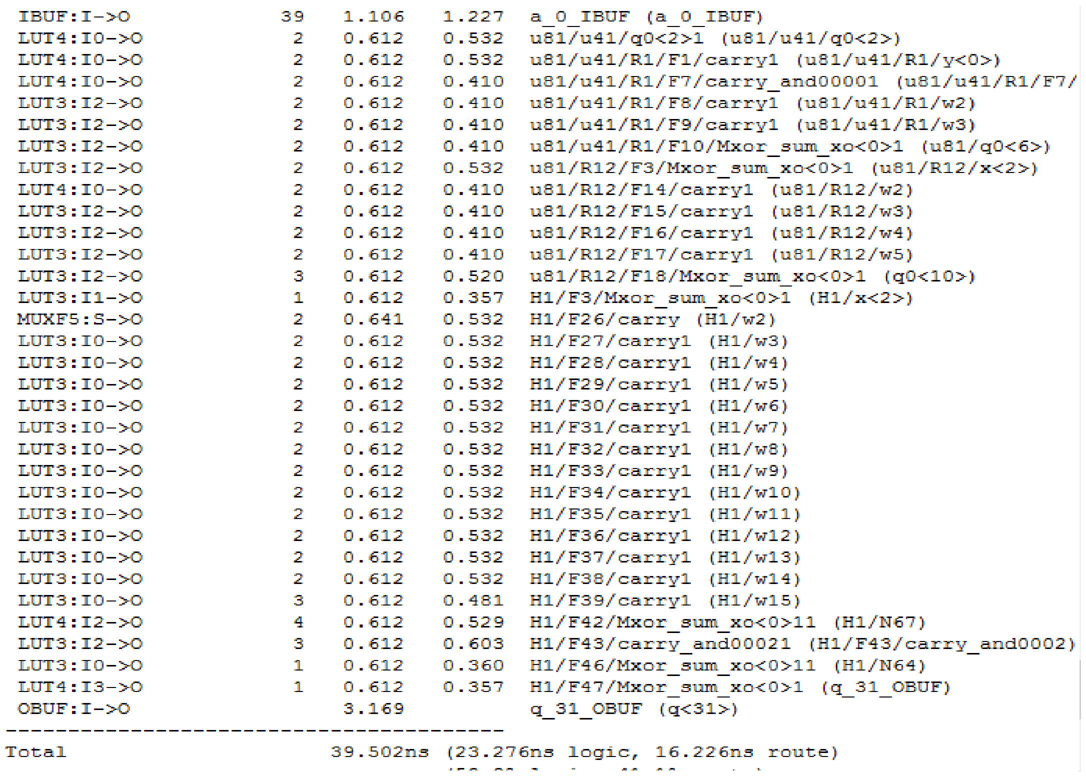
Fig.4.16. Simulation of Wallace Tree Multiplier using HCA

4.2.4 Vedic Multiplier using Ripple Carry Adder



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	380	4656	8%
Number of 4 input LUTs	661	9312	7%
Number of bonded IOBs	64	66	96%

Fig.4.17. Device Utilization of Vedic Multiplier using RCA



```
IBUF:I->O             39   1.106   1.227  a_0_IBUF (a_0_IBUF)
LUT4:I0->O             2   0.612   0.532  u81/u41/q0<2>1 (u81/u41/q0<2>)
LUT4:I0->O             2   0.612   0.532  u81/u41/R1/F1/carry1 (u81/u41/R1/y<0>)
LUT4:I0->O             2   0.612   0.410  u81/u41/R1/F7/carry_and00001 (u81/u41/R1/F7/
LUT3:I2->O             2   0.612   0.410  u81/u41/R1/F8/carry1 (u81/u41/R1/w2)
LUT3:I2->O             2   0.612   0.410  u81/u41/R1/F9/carry1 (u81/u41/R1/w3)
LUT3:I2->O             2   0.612   0.410  u81/u41/R1/F10/Mxor_sum_xo<0>1 (u81/q0<6>)
LUT3:I2->O             2   0.612   0.532  u81/R12/F3/Mxor_sum_xo<0>1 (u81/R12/x<2>)
LUT4:I0->O             2   0.612   0.410  u81/R12/F14/carry1 (u81/R12/w2)
LUT3:I2->O             2   0.612   0.410  u81/R12/F15/carry1 (u81/R12/w3)
LUT3:I2->O             2   0.612   0.410  u81/R12/F16/carry1 (u81/R12/w4)
LUT3:I2->O             2   0.612   0.410  u81/R12/F17/carry1 (u81/R12/w5)
LUT3:I2->O             3   0.612   0.520  u81/R12/F18/Mxor_sum_xo<0>1 (q0<10>)
LUT3:I1->O             1   0.612   0.357  H1/F3/Mxor_sum_xo<0>1 (H1/x<2>)
MUXF5:S->O             2   0.641   0.532  H1/F26/carry (H1/w2)
LUT3:I0->O             2   0.612   0.532  H1/F27/carry1 (H1/w3)
LUT3:I0->O             2   0.612   0.532  H1/F28/carry1 (H1/w4)
LUT3:I0->O             2   0.612   0.532  H1/F29/carry1 (H1/w5)
LUT3:I0->O             2   0.612   0.532  H1/F30/carry1 (H1/w6)
LUT3:I0->O             2   0.612   0.532  H1/F31/carry1 (H1/w7)
LUT3:I0->O             2   0.612   0.532  H1/F32/carry1 (H1/w8)
LUT3:I0->O             2   0.612   0.532  H1/F33/carry1 (H1/w9)
LUT3:I0->O             2   0.612   0.532  H1/F34/carry1 (H1/w10)
LUT3:I0->O             2   0.612   0.532  H1/F35/carry1 (H1/w11)
LUT3:I0->O             2   0.612   0.532  H1/F36/carry1 (H1/w12)
LUT3:I0->O             2   0.612   0.532  H1/F37/carry1 (H1/w13)
LUT3:I0->O             2   0.612   0.532  H1/F38/carry1 (H1/w14)
LUT3:I0->O             3   0.612   0.481  H1/F39/carry1 (H1/w15)
LUT4:I2->O             4   0.612   0.529  H1/F42/Mxor_sum_xo<0>11 (H1/N67)
LUT3:I2->O             3   0.612   0.603  H1/F43/carry_and00021 (H1/F43/carry_and0002)
LUT3:I0->O             1   0.612   0.360  H1/F46/Mxor_sum_xo<0>11 (H1/N64)
LUT4:I3->O             1   0.612   0.357  H1/F47/Mxor_sum_xo<0>1 (q_31_OBUF)
 OBUF:I->O                 3.169          q_31_OBUF (q<31>)
 ----------------------------------------
Total                    39.502ns (23.276ns logic, 16.226ns route)
```

Fig.4.18. Delay Report of Vedic Multiplier using RCA

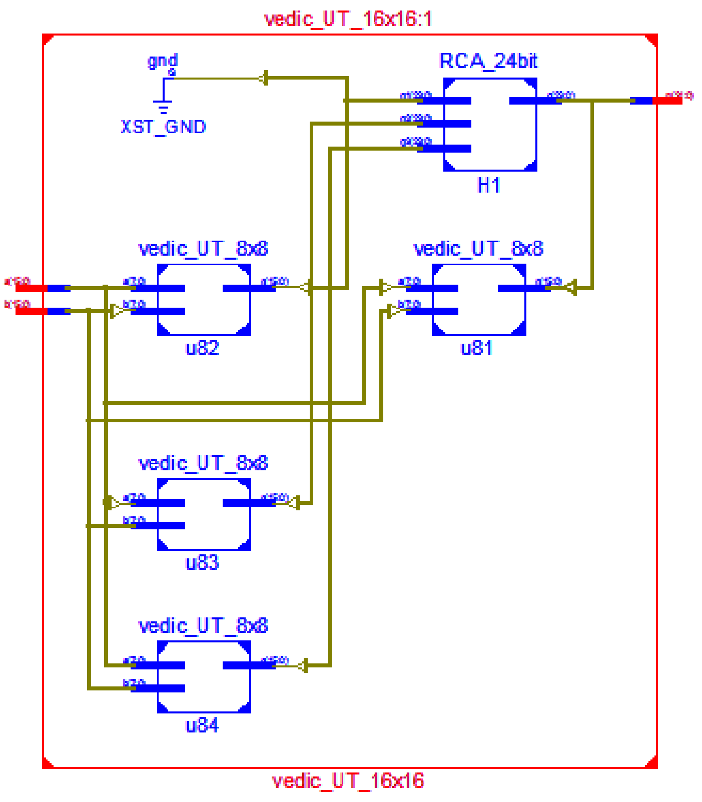


Fig.4.19. Schematic of Vedic Multiplier using RCA

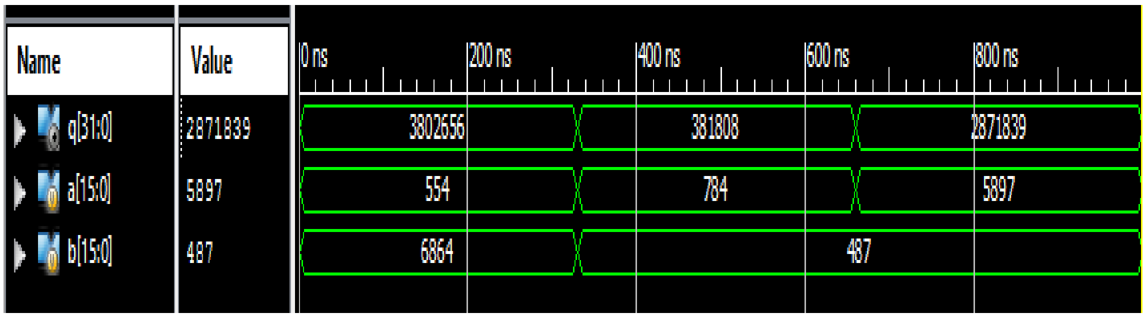
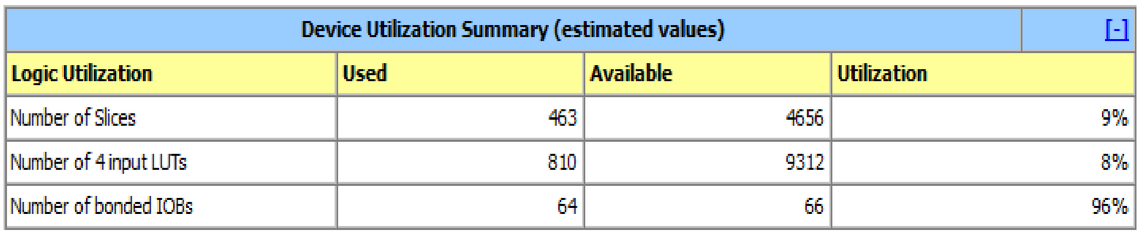


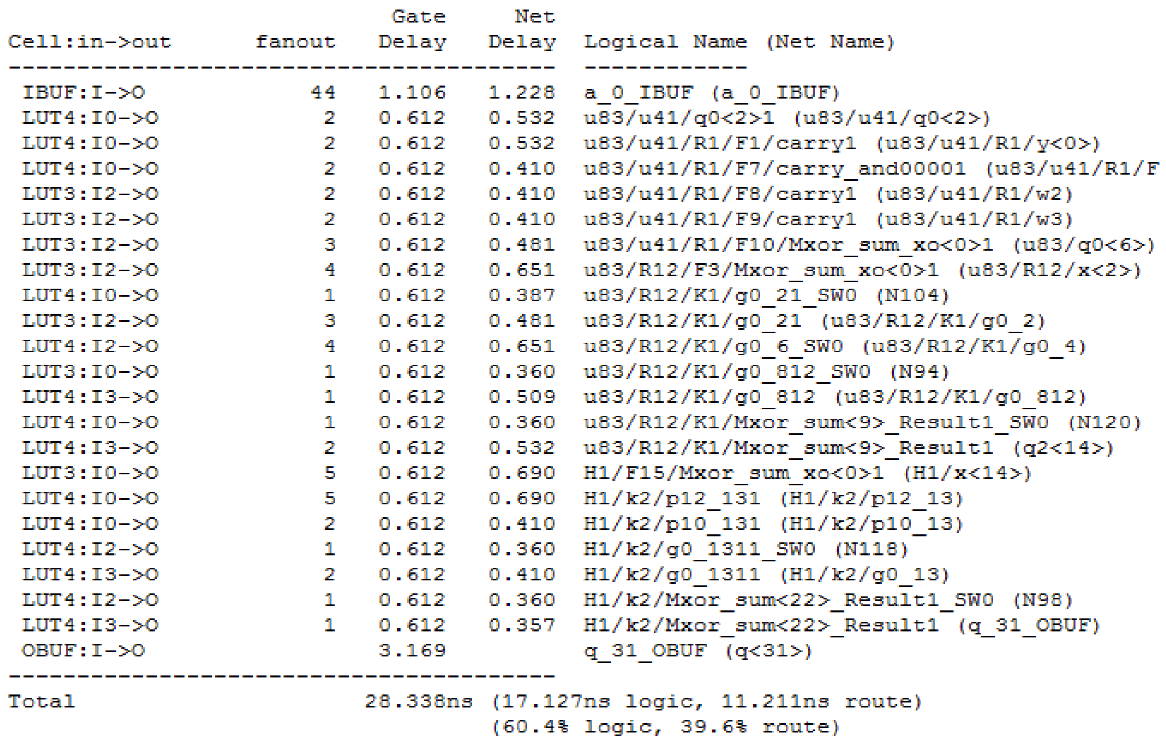
Fig.4.20. Simulation Result of Vedic Multiplier using RCA

4.2.5 Vedic Multiplier using Kogge Stone Adder



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	463	4656	9%
Number of 4 input LUTs	810	9312	8%
Number of bonded IOBs	64	66	96%

Fig.4.21. Device Utilization of Vedic Multiplier using KSA



```
                                   Gate     Net
Cell:in->out       fanout   Delay    Delay   Logical Name (Net Name)
----------------------------------------    ------------
 IBUF:I->O            44    1.106    1.228  a_0_IBUF (a_0_IBUF)
 LUT4:I0->O            2    0.612    0.532  u83/u41/q0<2>1 (u83/u41/q0<2>)
 LUT4:I0->O            2    0.612    0.532  u83/u41/R1/F1/carry1 (u83/u41/R1/y<0>)
 LUT4:I0->O            2    0.612    0.410  u83/u41/R1/F7/carry_and00001 (u83/u41/R1/F
 LUT3:I2->O            2    0.612    0.410  u83/u41/R1/F8/carry1 (u83/u41/R1/w2)
 LUT3:I2->O            2    0.612    0.410  u83/u41/R1/F9/carry1 (u83/u41/R1/w3)
 LUT3:I2->O            3    0.612    0.481  u83/u41/R1/F10/Mxor_sum_xo<0>1 (u83/q0<6>)
 LUT3:I2->O            4    0.612    0.651  u83/R12/F3/Mxor_sum_xo<0>1 (u83/R12/x<2>)
 LUT4:I0->O            1    0.612    0.387  u83/R12/K1/g0_21_SW0 (N104)
 LUT3:I2->O            3    0.612    0.481  u83/R12/K1/g0_21 (u83/R12/K1/g0_2)
 LUT4:I2->O            4    0.612    0.651  u83/R12/K1/g0_6_SW0 (u83/R12/K1/g0_4)
 LUT3:I0->O            1    0.612    0.360  u83/R12/K1/g0_812_SW0 (N94)
 LUT4:I3->O            1    0.612    0.509  u83/R12/K1/g0_812 (u83/R12/K1/g0_812)
 LUT4:I0->O            1    0.612    0.360  u83/R12/K1/Mxor_sum<9>_Result1_SW0 (N120)
 LUT4:I3->O            2    0.612    0.532  u83/R12/K1/Mxor_sum<9>_Result1 (q2<14>)
 LUT3:I0->O            5    0.612    0.690  H1/F15/Mxor_sum_xo<0>1 (H1/x<14>)
 LUT4:I0->O            5    0.612    0.690  H1/k2/p12_131 (H1/k2/p12_13)
 LUT4:I0->O            2    0.612    0.410  H1/k2/p10_131 (H1/k2/p10_13)
 LUT4:I2->O            1    0.612    0.360  H1/k2/g0_1311_SW0 (N118)
 LUT4:I3->O            2    0.612    0.410  H1/k2/g0_1311 (H1/k2/g0_13)
 LUT4:I2->O            1    0.612    0.360  H1/k2/Mxor_sum<22>_Result1_SW0 (N98)
 LUT4:I3->O            1    0.612    0.357  H1/k2/Mxor_sum<22>_Result1 (q_31_OBUF)
 OBUF:I->O                  3.169           q_31_OBUF (q<31>)
----------------------------------------
Total                      28.338ns (17.127ns logic, 11.211ns route)
                                    (60.4% logic, 39.6% route)
```

Fig.4.22. Delay Report of Vedic Multiplier Using KSA

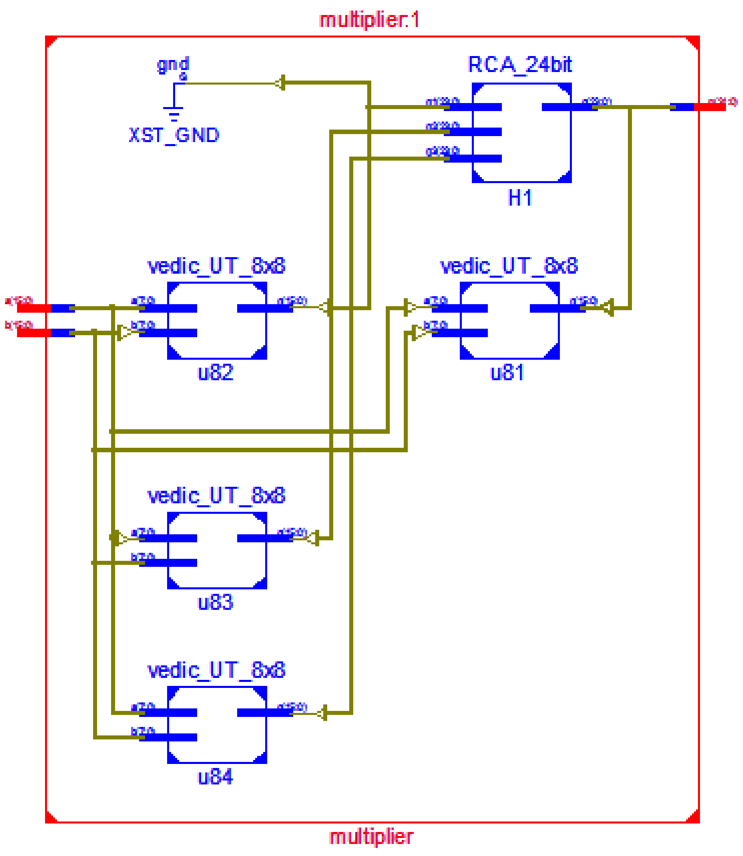
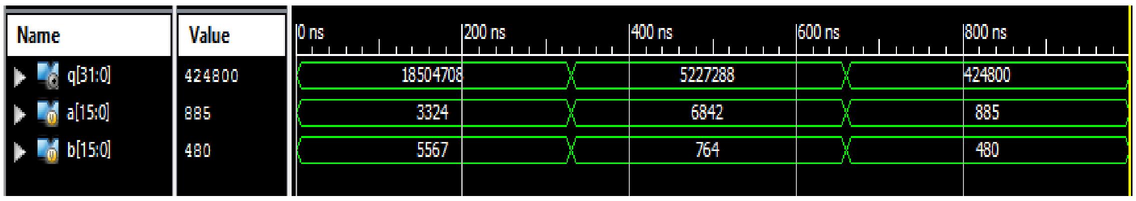


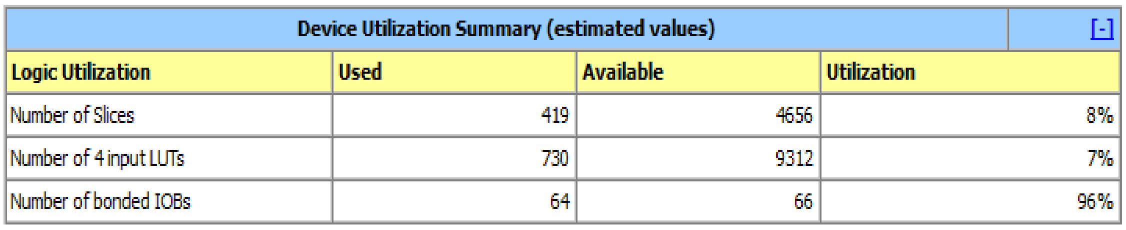
Fig.4.23. Schematic of Vedic Multiplier Using KSA



```
Name       Value    0 ns        200 ns       400 ns       600 ns       800 ns
q[31:0]    424800       18504708                  5227288                   424800
a[15:0]    885              3324                     6842                      885
b[15:0]    480              5567                      764                      480
```

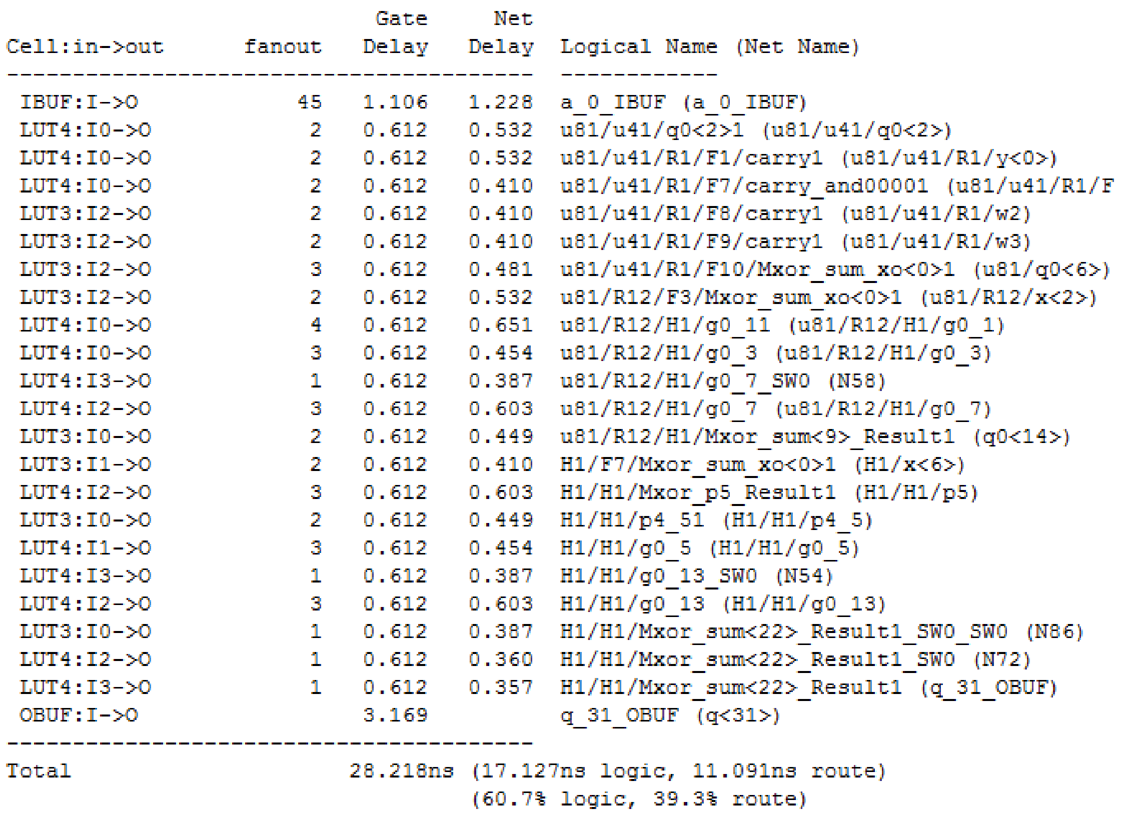
Fig.4.24. Simulation Result of Vedic Multiplier Using KSA

4.2.6 Vedic Multiplier using Han Carlson Adder



Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	419	4656	8%
Number of 4 input LUTs	730	9312	7%
Number of bonded IOBs	64	66	96%

Fig.4.25. Device Utilization of Vedic Multiplier using HCA



```
                                    Gate     Net
Cell:in->out           fanout     Delay    Delay   Logical Name (Net Name)
----------------------------------------   ------------
 IBUF:I->O                 45     1.106    1.228   a_0_IBUF (a_0_IBUF)
 LUT4:I0->O                 2     0.612    0.532   u81/u41/q0<2>1 (u81/u41/q0<2>)
 LUT4:I0->O                 2     0.612    0.532   u81/u41/R1/F1/carry1 (u81/u41/R1/y<0>)
 LUT4:I0->O                 2     0.612    0.410   u81/u41/R1/F7/carry_and00001 (u81/u41/R1/F
 LUT3:I2->O                 2     0.612    0.410   u81/u41/R1/F8/carry1 (u81/u41/R1/w2)
 LUT3:I2->O                 2     0.612    0.410   u81/u41/R1/F9/carry1 (u81/u41/R1/w3)
 LUT3:I2->O                 3     0.612    0.481   u81/u41/R1/F10/Mxor_sum_xo<0>1 (u81/q0<6>)
 LUT3:I2->O                 2     0.612    0.532   u81/R12/F3/Mxor_sum_xo<0>1 (u81/R12/x<2>)
 LUT4:I0->O                 4     0.612    0.651   u81/R12/H1/g0_11 (u81/R12/H1/g0_1)
 LUT4:I0->O                 3     0.612    0.454   u81/R12/H1/g0_3 (u81/R12/H1/g0_3)
 LUT4:I3->O                 1     0.612    0.387   u81/R12/H1/g0_7_SW0 (N58)
 LUT4:I2->O                 3     0.612    0.603   u81/R12/H1/g0_7 (u81/R12/H1/g0_7)
 LUT3:I0->O                 2     0.612    0.449   u81/R12/H1/Mxor_sum<9>_Result1 (q0<14>)
 LUT3:I1->O                 2     0.612    0.410   H1/F7/Mxor_sum_xo<0>1 (H1/x<6>)
 LUT4:I2->O                 3     0.612    0.603   H1/H1/Mxor_p5_Result1 (H1/H1/p5)
 LUT3:I0->O                 2     0.612    0.449   H1/H1/p4_51 (H1/H1/p4_5)
 LUT4:I1->O                 3     0.612    0.454   H1/H1/g0_5 (H1/H1/g0_5)
 LUT4:I3->O                 1     0.612    0.387   H1/H1/g0_13_SW0 (N54)
 LUT4:I2->O                 3     0.612    0.603   H1/H1/g0_13 (H1/H1/g0_13)
 LUT3:I0->O                 1     0.612    0.387   H1/H1/Mxor_sum<22>_Result1_SW0_SW0 (N86)
 LUT4:I2->O                 1     0.612    0.360   H1/H1/Mxor_sum<22>_Result1_SW0 (N72)
 LUT4:I3->O                 1     0.612    0.357   H1/H1/Mxor_sum<22>_Result1 (q_31_OBUF)
 OBUF:I->O                        3.169            q_31_OBUF (q<31>)
----------------------------------------
Total                            28.218ns (17.127ns logic, 11.091ns route)
                                          (60.7% logic, 39.3% route)
```

Fig.4.26. Delay Report of Vedic Multiplier using HCA

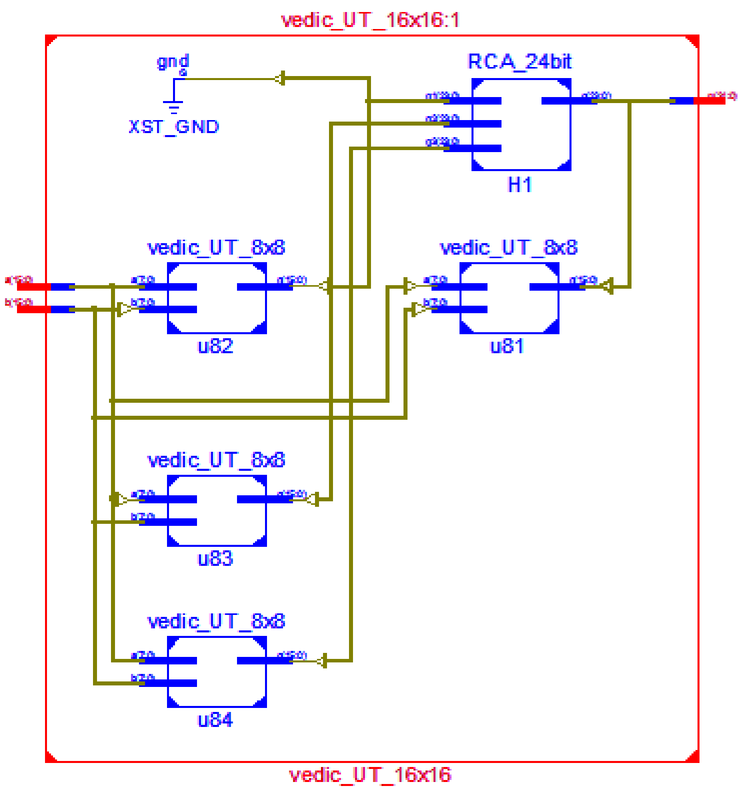


Fig.4.27. Schematic of Vedic multiplier using HCA

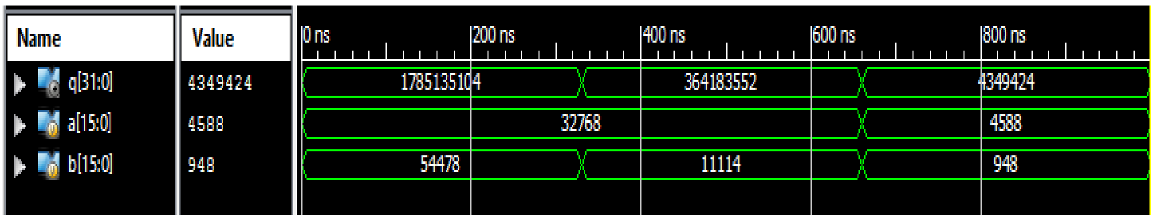


Fig.4.28. Simulation result of Vedic multiplier using HCA

Table 1 shows the comparative analysis of the performance of the Wallace Tree Multiplier and Vedic Multiplier designs using Ripple Carry Adder, Kogge Stone Adder, and Han Carlson Adder on the basis of Area (LUT) and Delay(ns).

Table 4.1 shows Comparitive Analysis Of Wallace Tree Multiplier And Vedic Multiplier Using Different Adders

Multiplier	Delay(ns)	Area(LUTs)
Wallace Tree Using Ripple Carry Adder	34.929	657
Wallace Tree Using Kogge Stone Adder	24.221	769
Wallace Tree Using Han Carlson Adder	23.12	703
Vedic Using Ripple Carry Adder	39.502	661
Vedic Using Kogge Stone Adder	28.338	810
Vedic Using Han Carlson Adder	28.128	730

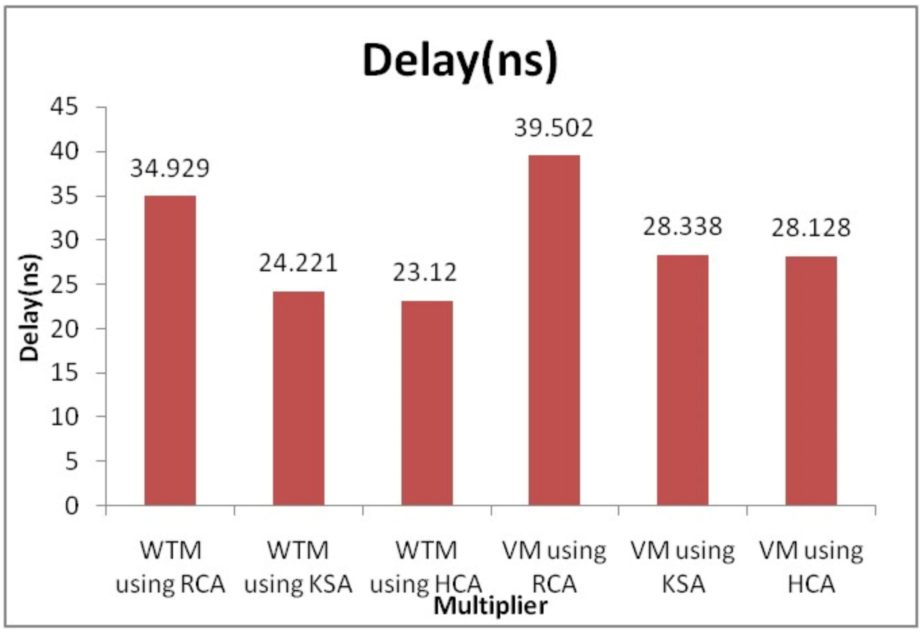


Fig.4.29 Delay comparison graph for multipliers

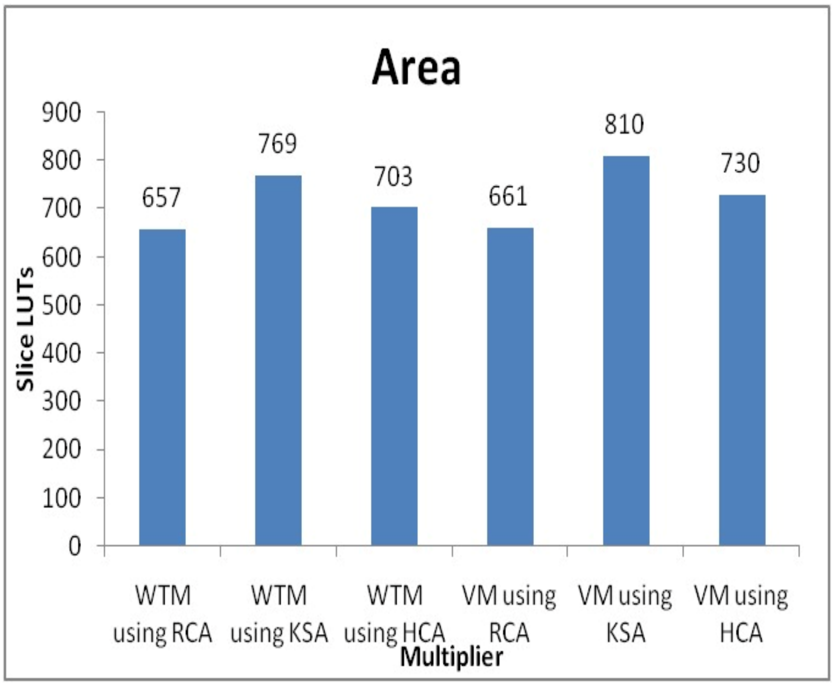


Fig.4.30 . Area comparison graph for multipliers

4.3 IMPORTANT OUTCOMES

In this chapter, the design and analyses of multipliers are carried out and compared to determine the optimal option for the best performance.

- The Wallace tree multiplier and Vedic multiplier are design and simulated using Ripple Carry adder, Kogge Stone adder and Han Carlson adder..
- Delay and Area have been calculated and plotted
- Speeding up the circuit by using the Wallace tree multiplier and the Han Carlson adder.
- The Vedic multiplier with ripple carry adder used less space but was the slowest in terms of speed.

CHAPTER 5

DESIGN AND COMPARISON OF MATRIX MULTIPLICATION USING WALLACE TREE AND VEDIC MULTIPLIER WITH PARALLEL PREFIX ADDER

Matrix multiplication is a basic operation in many mathematical concepts and methods, including the solution of linear equation systems, computer graphics transformations, and calculations involving neural networks. In many applications, the effectiveness of matrix multiplication is a crucial factor to take into account, especially when dealing with huge matrices or when matrix multiplication needs to be done repeatedly. Multiplication of matrices can be computationally demanding, particularly for large matrices. The amount of time needed to complete calculations can be considerably decreased by optimizing matrix multiplication techniques. In real-time applications like computer graphics, simulations, and scientific computations, where faster execution can result in more responsive and effective systems, this is especially crucial. Matrix multiplication algorithm performance optimization can help minimize memory accesses and lower the memory footprint, resulting in more effective memory usage.

Matrix multiplication performance can vary greatly depending on the matrix multiplication method used, with effects on things like computational complexity, memory access patterns, parallelizability, and cache utilization. The effectiveness of matrix multiplication can also be impacted by the adder that is selected in a matrix multiplier. The critical path delay of the matrix multiplier, which affects the total latency of the multiplication operation, can be influenced by the adder choice. The critical route length can be shortened with faster adders with less latency, which will speed up multiplication. Lower latency can be achieved with adder designs that have reduced carry propagation delays and optimized logic. In this chapter, matrix multiplication is performed using the Wallace Tree Multiplier and Vedic Multiplier. For the addition purpose, Ripple Carry Adder, Kogge Stone Adder, and Han Carlson Adder are used and compared for their performance.

This chapter is organized in following sub-sections:

- The fundamental of matrix multiplication concept discuss in Section 5.1.
- The Simulation and Synthesis Resultss are found in section 5.2.
- The important outcomes of the improved sense amplifier are discussed in Section 5.3.

The basic process of matrix multiplication in linear algebra entails multiplying two matrices to produce a new matrix.

5.1 Fundamental of matrix multiplication

The "left" matrix and the "right" matrix are the two matrices needed to multiply two matrices. The size of the left matrix, denoted as A, is characterized by its dimensions of m x n, where m represents the number of rows and n represents the number of columns. The right matrix, referred to as B, has dimensions of n x p, where p represents the number of columns. The resulting matrix, denoted as C, will have dimensions of m x p, where m represents the number of rows from matrix A and p represents the number of columns from matrix B.The dot product of the i-th row of matrix A and the j-th column of matrix B is used to calculate each element C[i][j] in the resulting matrix.

The i-th row of matrix A and the j-th column of matrix B's dot product are formed by multiplying and adding the respective elements. Element A[i][k] from the i-th row of matrix A is multiplied by element B[k][j] from the j-th column of matrix B in this operation, where k is a number between 1 and n. The last element of matrix C[i][j] is obtained by adding the generated products.

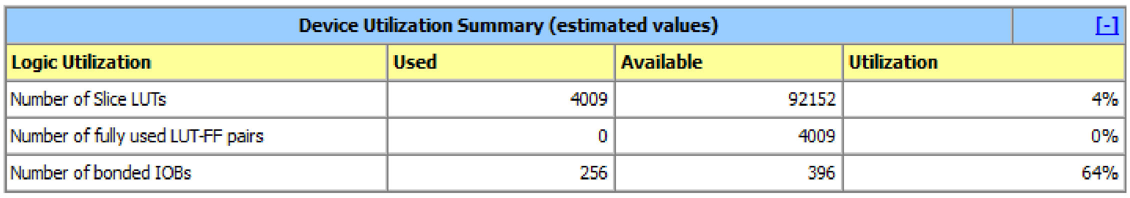
The mathematical formula for the element C[i][j] is:

$$C[i][j] = A[i][1] * B[1][j] + A[i][2]* B[2][j] +\ldots + A[i][n]*B[n][j]$$

The resultant matrix C, which represents the multiplication of matrices A and B, is created by repeating this calculation for each member.

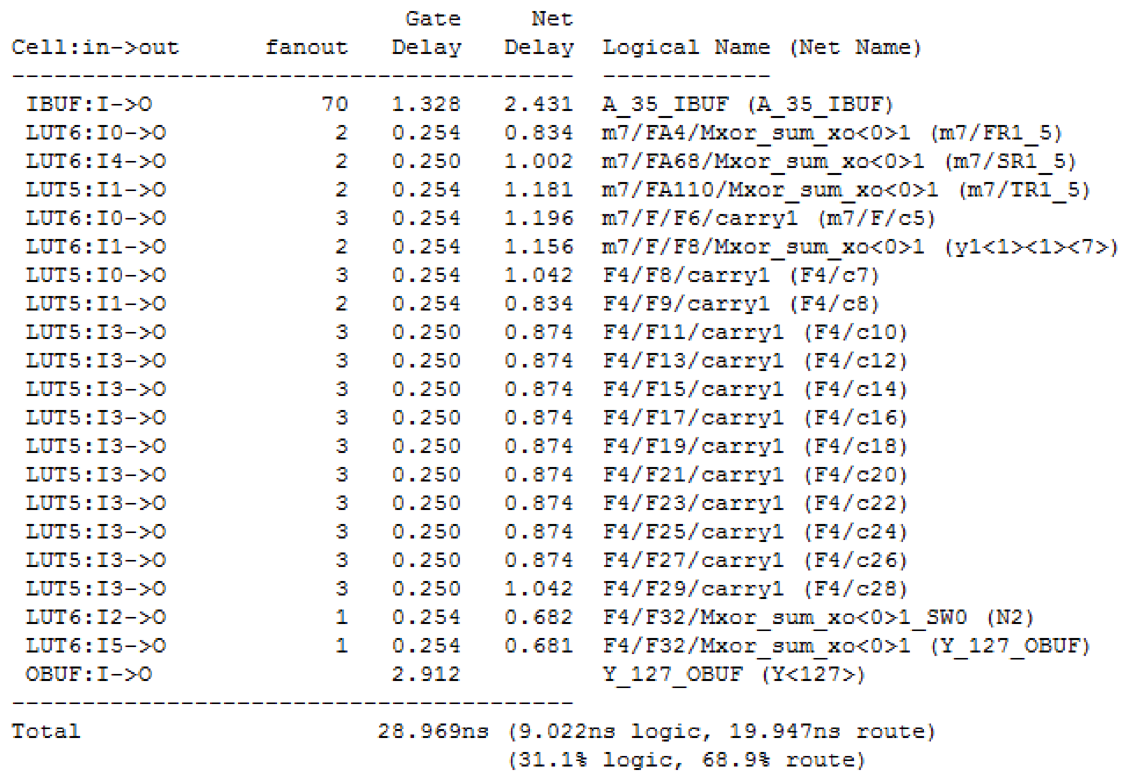
5.2 Simulated and Synthesis Results

5.2.1 Matrix multiplication using Wallace Tree Multiplier and RCA



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	4009	92152	4%
Number of fully used LUT-FF pairs	0	4009	0%
Number of bonded IOBs	256	396	64%

Fig 5.1. Device utilization of Matrix Multiplication using Wallace Tree Multiplier and RCA



```
                                  Gate     Net
Cell:in->out       fanout    Delay    Delay  Logical Name (Net Name)
---------------------------------------  ------------
 IBUF:I->O             70   1.328    2.431  A_35_IBUF (A_35_IBUF)
 LUT6:I0->O             2   0.254    0.834  m7/FA4/Mxor_sum_xo<0>1 (m7/FR1_5)
 LUT6:I4->O             2   0.250    1.002  m7/FA68/Mxor_sum_xo<0>1 (m7/SR1_5)
 LUT5:I1->O             2   0.254    1.181  m7/FA110/Mxor_sum_xo<0>1 (m7/TR1_5)
 LUT6:I0->O             3   0.254    1.196  m7/F/F6/carry1 (m7/F/c5)
 LUT6:I1->O             2   0.254    1.156  m7/F/F8/Mxor_sum_xo<0>1 (y1<1><1><7>)
 LUT5:I0->O             3   0.254    1.042  F4/F8/carry1 (F4/c7)
 LUT5:I1->O             2   0.254    0.834  F4/F9/carry1 (F4/c8)
 LUT5:I3->O             3   0.250    0.874  F4/F11/carry1 (F4/c10)
 LUT5:I3->O             3   0.250    0.874  F4/F13/carry1 (F4/c12)
 LUT5:I3->O             3   0.250    0.874  F4/F15/carry1 (F4/c14)
 LUT5:I3->O             3   0.250    0.874  F4/F17/carry1 (F4/c16)
 LUT5:I3->O             3   0.250    0.874  F4/F19/carry1 (F4/c18)
 LUT5:I3->O             3   0.250    0.874  F4/F21/carry1 (F4/c20)
 LUT5:I3->O             3   0.250    0.874  F4/F23/carry1 (F4/c22)
 LUT5:I3->O             3   0.250    0.874  F4/F25/carry1 (F4/c24)
 LUT5:I3->O             3   0.250    0.874  F4/F27/carry1 (F4/c26)
 LUT5:I3->O             3   0.250    1.042  F4/F29/carry1 (F4/c28)
 LUT6:I2->O             1   0.254    0.682  F4/F32/Mxor_sum_xo<0>1_SW0 (N2)
 LUT6:I5->O             1   0.254    0.681  F4/F32/Mxor_sum_xo<0>1 (Y_127_OBUF)
 OBUF:I->O                  2.912           Y_127_OBUF (Y<127>)
---------------------------------------
Total                     28.969ns (9.022ns logic, 19.947ns route)
                                   (31.1% logic, 68.9% route)
```

Fig 5.2. Delay Report of Matrix Multiplication using Wallace Tree Multiplier and RCA

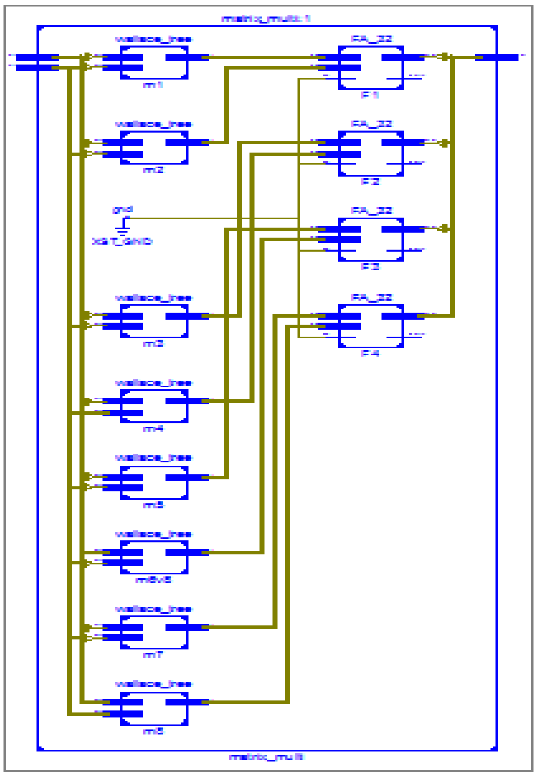


Fig 5.3. Schematic of multiplication using Wallace Tree Multiplier and RCA

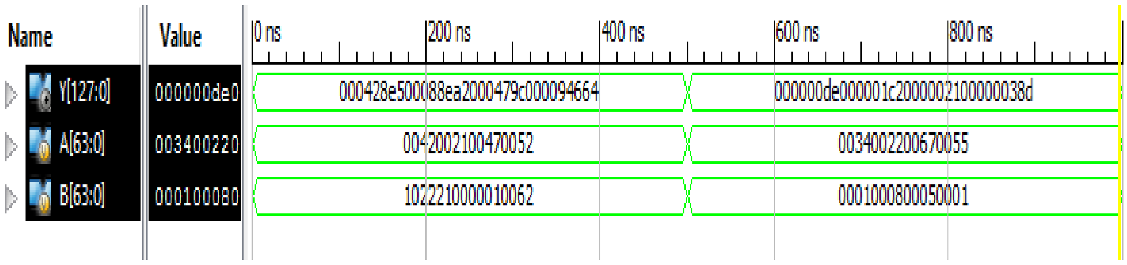
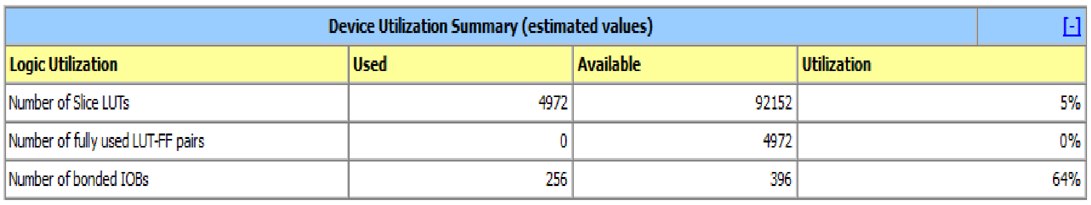


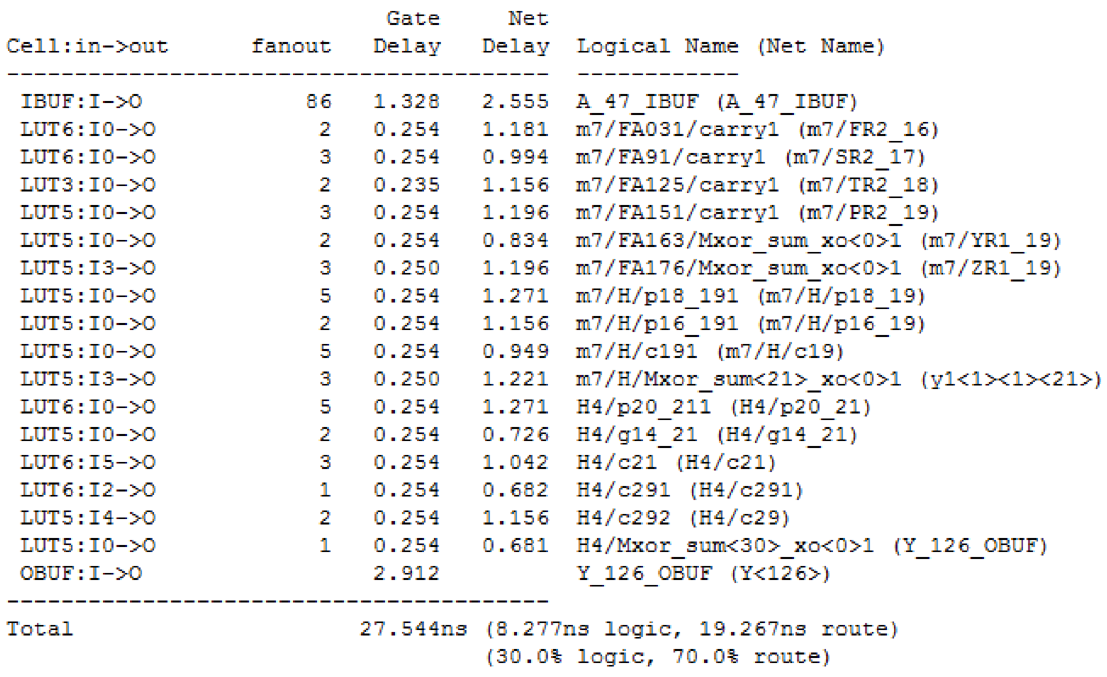
Fig 5.4. Simulation of multiplication using Wallace Tree Multiplier and RCA

5.2.2 Matrix Multiplication using Wallace Tree Multiplier and KSA



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	4972	92152	5%
Number of fully used LUT-FF pairs	0	4972	0%
Number of bonded IOBs	256	396	64%

Fig 5.5. Device Utilization of Matrix Multiplication using WTM and KSA



```
                                   Gate     Net
Cell:in->out          fanout   Delay   Delay   Logical Name (Net Name)
----------------------------------------   ------------
 IBUF:I->O               86   1.328   2.555  A_47_IBUF (A_47_IBUF)
 LUT6:I0->O               2   0.254   1.181  m7/FA031/carry1 (m7/FR2_16)
 LUT6:I0->O               3   0.254   0.994  m7/FA91/carry1 (m7/SR2_17)
 LUT3:I0->O               2   0.235   1.156  m7/FA125/carry1 (m7/TR2_18)
 LUT5:I0->O               3   0.254   1.196  m7/FA151/carry1 (m7/PR2_19)
 LUT5:I0->O               2   0.254   0.834  m7/FA163/Mxor_sum_xo<0>1 (m7/YR1_19)
 LUT5:I3->O               3   0.250   1.196  m7/FA176/Mxor_sum_xo<0>1 (m7/ZR1_19)
 LUT5:I0->O               5   0.254   1.271  m7/H/p18_191 (m7/H/p18_19)
 LUT5:I0->O               2   0.254   1.156  m7/H/p16_191 (m7/H/p16_19)
 LUT5:I0->O               5   0.254   0.949  m7/H/c191 (m7/H/c19)
 LUT5:I3->O               3   0.250   1.221  m7/H/Mxor_sum<21>_xo<0>1 (y1<1><1><21>)
 LUT6:I0->O               5   0.254   1.271  H4/p20_211 (H4/p20_21)
 LUT5:I0->O               2   0.254   0.726  H4/g14_21 (H4/g14_21)
 LUT6:I5->O               3   0.254   1.042  H4/c21 (H4/c21)
 LUT6:I2->O               1   0.254   0.682  H4/c291 (H4/c291)
 LUT5:I4->O               2   0.254   1.156  H4/c292 (H4/c29)
 LUT5:I0->O               1   0.254   0.681  H4/Mxor_sum<30>_xo<0>1 (Y_126_OBUF)
 OBUF:I->O                    2.912          Y_126_OBUF (Y<126>)
----------------------------------------
Total                       27.544ns (8.277ns logic, 19.267ns route)
                                     (30.0% logic, 70.0% route)
```

Fig 5.6. Delay Report of Matrix Multiplication using WTM and KSA

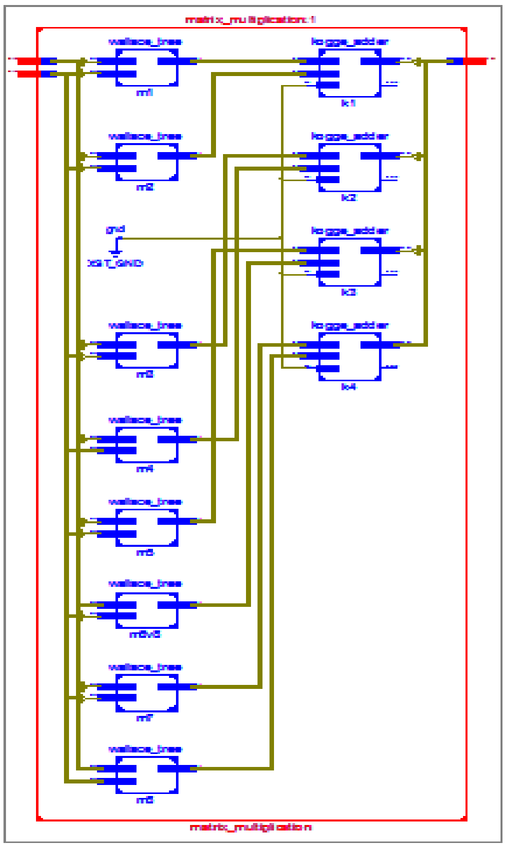
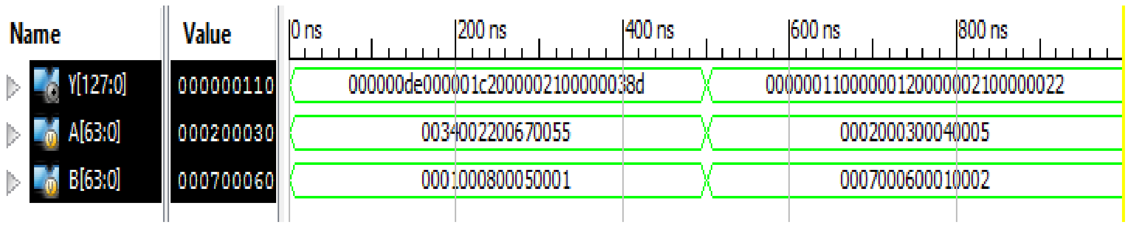


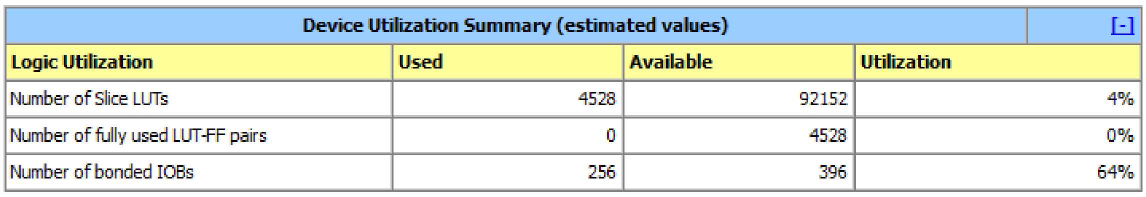
Fig 5.7. Schematic of Matrix Multiplication using WTM and KSA



```
Name        Value        |0 ns     |200 ns     |400 ns     |600 ns     |800 ns
Y[127:0]    000000110    000000de000001c2000002100000038d   00000011000000120000002100000022
A[63:0]     000200030    0034002200670055                   0002000300040005
B[63:0]     000700060    0001000800050001                   0007000600010002
```

Fig 5.8. Simulation result of Matrix Multiplication using WTM and KSA

5.2.3 Matrix Multiplication using Wallace Tree Multiplier and HCA



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	4528	92152	4%
Number of fully used LUT-FF pairs	0	4528	0%
Number of bonded IOBs	256	396	64%

Fig 5.9. Device Utilization of Matrix Multiplication using WTM and HCA

```
                                   Gate     Net
Cell:in->out          fanout    Delay    Delay   Logical Name (Net Name)
----------------------------------------  ------------
 IBUF:I->O              100   1.328    2.485  A_0_IBUF (A_0_IBUF)
 LUT4:I0->O               2   0.254    1.181  m3/H3/Mxor_sum_xo<0>1 (m3/FR5_7)
 LUT6:I0->O               4   0.254    1.080  m3/HA12/carry1 (m3/SR3_8)
 LUT5:I1->O               2   0.254    1.156  m3/FA113/carry1 (m3/TR2_9)
 LUT5:I0->O               2   0.254    1.156  m3/FA141/carry1 (m3/PR2_10)
 LUT6:I1->O               6   0.254    1.104  m3/FA168/Mxor_sum_xo<0>1 (m3/ZR1_10)
 LUT3:I0->O               3   0.235    1.221  m3/k/p10_111 (m3/k/p10_11)
 LUT6:I0->O               1   0.254    0.790  m3/k/c133_SW0 (N204)
 LUT6:I4->O               3   0.250    0.994  m3/k/c133 (m3/k/c13)
 LUT5:I2->O               2   0.235    0.954  m3/k/c171 (m3/k/c17)
 LUT6:I3->O               1   0.235    0.790  m3/k/c251 (m3/k/c251)
 LUT6:I4->O               6   0.250    1.152  m3/k/Mxor_sum<26>_xo<0>1 (y1<0><1><26>)
 LUT4:I0->O               4   0.254    1.234  k2/p25_261 (k2/p25_26)
 LUT6:I1->O               2   0.254    1.002  k2/c2621 (k2/c262)
 LUT6:I2->O               1   0.254    0.682  k2/c303_SW0 (N198)
 LUT6:I5->O               1   0.254    0.682  k2/c303 (k2/c30)
 LUT3:I2->O               1   0.254    0.681  k2/Mxor_sum<31>_xo<0>1 (Y_63_OBUF)
 OBUF:I->O                    2.912           Y_63_OBUF (Y<63>)
----------------------------------------
Total                      26.583ns (8.239ns logic, 18.344ns route)
                                    (31.0% logic, 69.0% route)
```

Fig 5.10. Delay Report of Matrix Multiplication using WTM and HCA

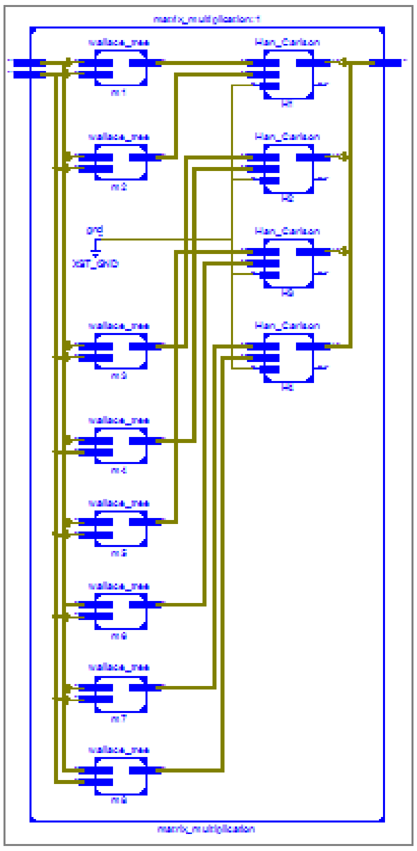


Fig 5.11. Schematic of Matrix Multiplication using WTM and HCA

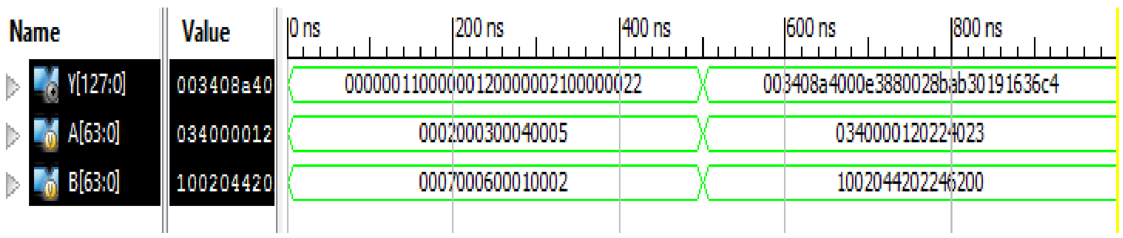
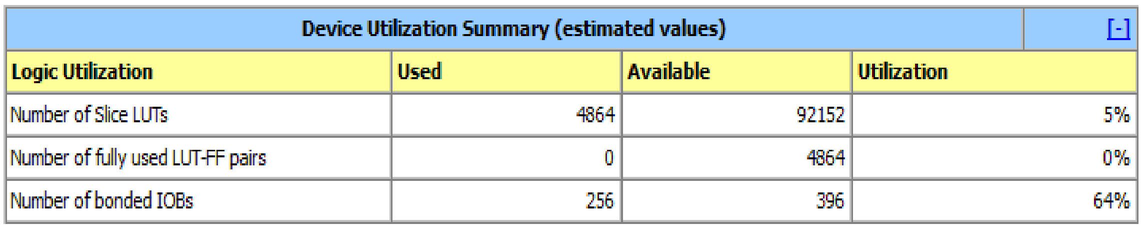


Fig 5.12. Simulation result of Matrix Multiplication using WTM and HCA

5.2.4 Matrix Multiplication using Vedic Multiplier and Ripple Carry Adder



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	4864	92152	5%
Number of fully used LUT-FF pairs	0	4864	0%
Number of bonded IOBs	256	396	64%

Fig 5.13. Device utilization of Matrix Multiplication using Vedic Multiplier and RCA

```
                                Gate     Net
Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
 IBUF:I->O           114   1.328   2.528  A_32_IBUF (A_32_IBUF)
 LUT4:I0->O            2   0.254   1.156  m5/u81/u41/u23/Mxor_q<1>_xo<0>1 (m5/u81/u41/q2<1>)
 LUT6:I1->O            2   0.254   1.156  m5/u81/u41/R1/F2/carry1 (m5/u81/u41/R1/y<1>)
 LUT6:I1->O            4   0.254   1.080  m5/u81/u41/R1/F8/carry1 (m5/u81/u41/R1/w2)
 LUT6:I2->O            3   0.254   0.994  m5/u81/u41/R1/F10/Mxor_sum_xo<0>1 (m5/u81/q0<6>)
 LUT3:I0->O            2   0.235   1.002  m5/u81/R12/F3/Mxor_sum_xo<0>1 (m5/u81/R12/x<2>)
 LUT6:I2->O            3   0.254   0.766  m5/u81/R12/F15/carry1 (m5/u81/R12/w3)
 LUT5:I4->O            3   0.254   0.766  m5/u81/R12/F17/carry1 (m5/u81/R12/w5)
 LUT5:I4->O            3   0.254   0.766  m5/u81/R12/F19/carry1 (m5/u81/R12/w7)
 LUT5:I4->O            2   0.254   1.156  m5/u81/R12/F20/Mxor_sum_xo<0>1 (m5/q0<12>)
 LUT6:I1->O            2   0.254   1.156  m5/H1/F5/Mxor_sum_xo<0>1 (m5/H1/x<4>)
 LUT5:I0->O            3   0.254   0.994  m5/H1/F28/carry1 (m5/H1/w4)
 LUT5:I2->O            2   0.235   1.156  m5/H1/F30/Mxor_sum_xo<0>1 (y1<1><0><14>)
 LUT5:I0->O            3   0.254   0.874  F3/F15/carry1 (F3/c14)
 LUT5:I3->O            3   0.250   0.874  F3/F17/carry1 (F3/c16)
 LUT5:I3->O            3   0.250   0.874  F3/F19/carry1 (F3/c18)
 LUT5:I3->O            3   0.250   0.874  F3/F21/carry1 (F3/c20)
 LUT5:I3->O            3   0.250   0.874  F3/F23/carry1 (F3/c22)
 LUT5:I3->O            3   0.250   0.874  F3/F25/carry1 (F3/c24)
 LUT5:I3->O            3   0.250   0.874  F3/F27/carry1 (F3/c26)
 LUT5:I3->O            3   0.250   0.874  F3/F29/carry1 (F3/c28)
 LUT6:I4->O            1   0.250   0.910  F3/F30/carry1 (F3/c29)
 LUT6:I3->O            1   0.235   0.681  F3/F32/Mxor_sum_xo<0>1 (Y_95_OBUF)
 OBUF:I->O                 2.912          Y_95_OBUF (Y<95>)
----------------------------------------
Total                     32.998ns (9.739ns logic, 23.259ns route)
                                   (29.5% logic, 70.5% route)
```

Fig 5.14. Delay Report of Matrix Multiplication using Vedic multiplier and RCA

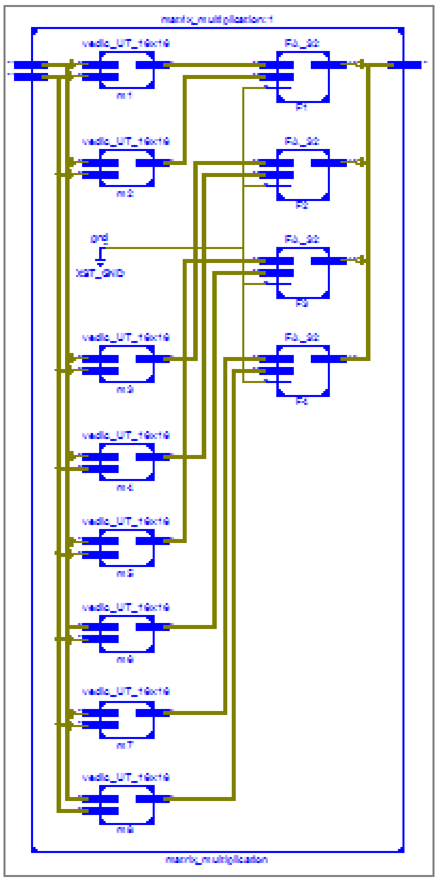
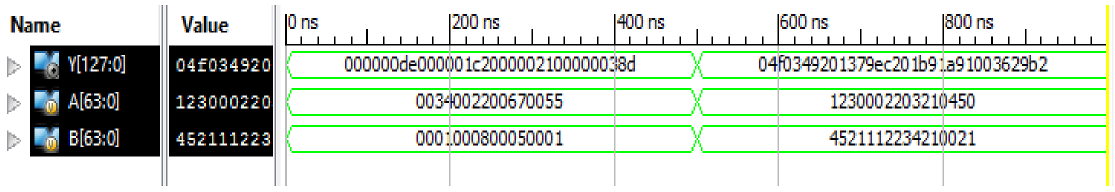


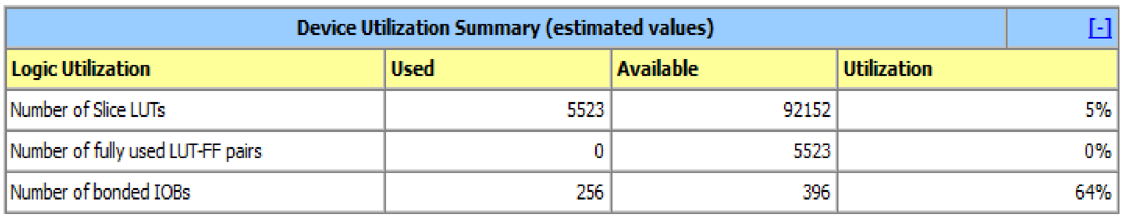
Fig 5.15. Schematic of Matrix Multiplication using Vedic multiplier and RCA



```
Name        Value       |0 ns        |200 ns       |400 ns       |600 ns       |800 ns
Y[127:0]    04f034920   000000de000001c20000021000000038d    04f0349201379ec201b91a91003629b2
A[63:0]     123000220   0034002200670055                     1230002203210450
B[63:0]     452111223   0001000800050001                     4521112234210021
```

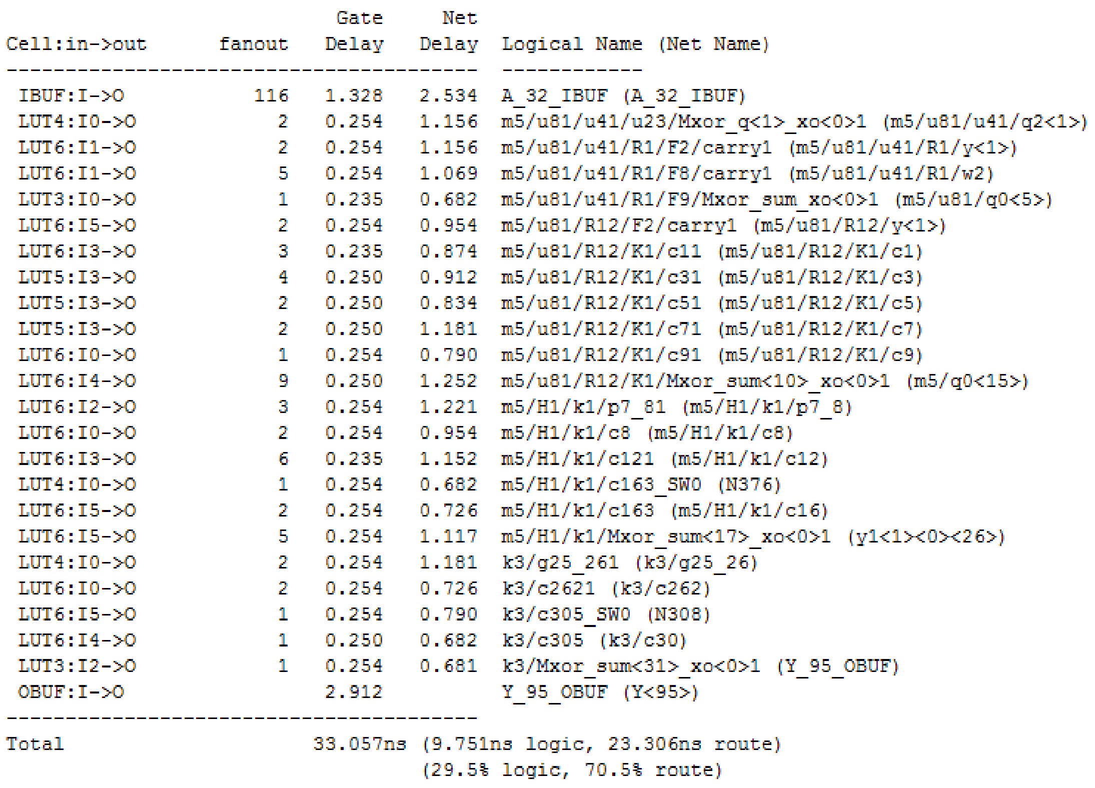
Fig 5.16. Simulation result of Matrix Multiplication using Vedic multiplier and RCA

5.2.5 Matrix Multiplication using Vedic Multiplier and KSA



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	5523	92152	5%
Number of fully used LUT-FF pairs	0	5523	0%
Number of bonded IOBs	256	396	64%

Fig 5.17. Device Utilization of Matrix Multiplication using Vedic multiplier and KSA



```
                                  Gate     Net
Cell:in->out        fanout    Delay    Delay   Logical Name (Net Name)
----------------------------------------   ------------
 IBUF:I->O            116    1.328    2.534  A_32_IBUF (A_32_IBUF)
 LUT4:I0->O             2    0.254    1.156  m5/u81/u41/u23/Mxor_q<1>_xo<0>1 (m5/u81/u41/q2<1>)
 LUT6:I1->O             2    0.254    1.156  m5/u81/u41/R1/F2/carry1 (m5/u81/u41/R1/y<1>)
 LUT6:I1->O             5    0.254    1.069  m5/u81/u41/R1/F8/carry1 (m5/u81/u41/R1/w2)
 LUT3:I0->O             1    0.235    0.682  m5/u81/u41/R1/F9/Mxor_sum_xo<0>1 (m5/u81/q0<5>)
 LUT6:I5->O             2    0.254    0.954  m5/u81/R12/F2/carry1 (m5/u81/R12/y<1>)
 LUT6:I3->O             3    0.235    0.874  m5/u81/R12/K1/c11 (m5/u81/R12/K1/c1)
 LUT5:I3->O             4    0.250    0.912  m5/u81/R12/K1/c31 (m5/u81/R12/K1/c3)
 LUT5:I3->O             2    0.250    0.834  m5/u81/R12/K1/c51 (m5/u81/R12/K1/c5)
 LUT5:I3->O             2    0.250    1.181  m5/u81/R12/K1/c71 (m5/u81/R12/K1/c7)
 LUT6:I0->O             1    0.254    0.790  m5/u81/R12/K1/c91 (m5/u81/R12/K1/c9)
 LUT6:I4->O             9    0.250    1.252  m5/u81/R12/K1/Mxor_sum<10>_xo<0>1 (m5/q0<15>)
 LUT6:I2->O             3    0.254    1.221  m5/H1/k1/p7_81 (m5/H1/k1/p7_8)
 LUT6:I0->O             2    0.254    0.954  m5/H1/k1/c8 (m5/H1/k1/c8)
 LUT6:I3->O             6    0.235    1.152  m5/H1/k1/c121 (m5/H1/k1/c12)
 LUT4:I0->O             1    0.254    0.682  m5/H1/k1/c163_SW0 (N376)
 LUT6:I5->O             2    0.254    0.726  m5/H1/k1/c163 (m5/H1/k1/c16)
 LUT6:I5->O             5    0.254    1.117  m5/H1/k1/Mxor_sum<17>_xo<0>1 (y1<1><0><26>)
 LUT4:I0->O             2    0.254    1.181  k3/g25_261 (k3/g25_26)
 LUT6:I0->O             2    0.254    0.726  k3/c2621 (k3/c262)
 LUT6:I5->O             1    0.254    0.790  k3/c305_SW0 (N308)
 LUT6:I4->O             1    0.250    0.682  k3/c305 (k3/c30)
 LUT3:I2->O             1    0.254    0.681  k3/Mxor_sum<31>_xo<0>1 (Y_95_OBUF)
 OBUF:I->O                   2.912           Y_95_OBUF (Y<95>)
----------------------------------------
Total                       33.057ns (9.751ns logic, 23.306ns route)
                                     (29.5% logic, 70.5% route)
```

Fig 5.18. Delay Report of Matrix Multiplication using Vedic multiplier and KSA

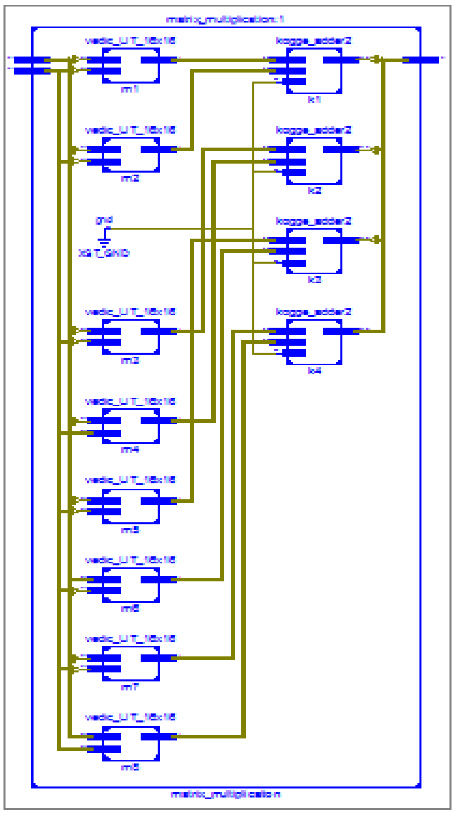


Fig 5.19. Schematic of Matrix Multiplication using Vedic multiplier and KSA

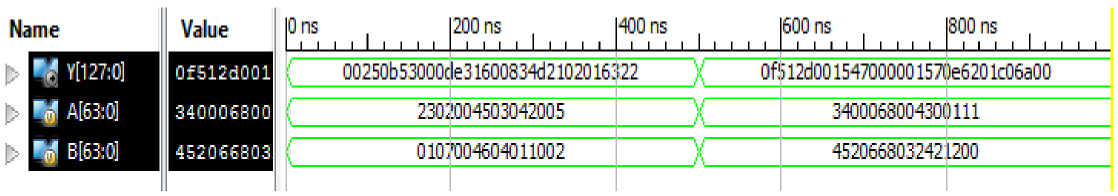
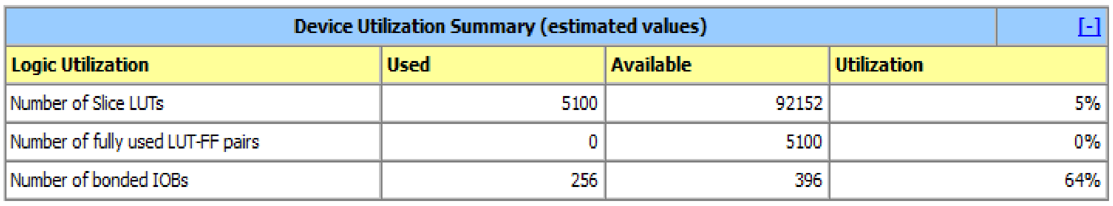


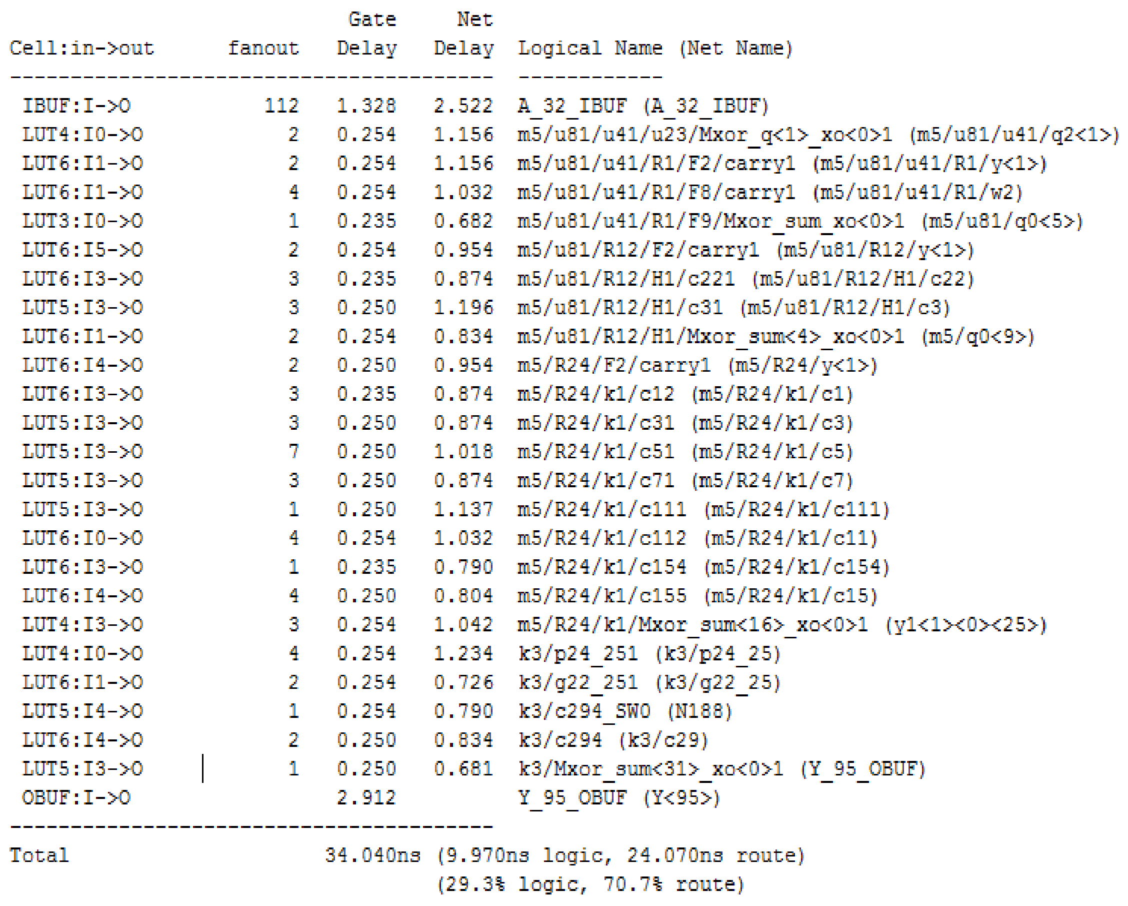
Fig 5.20. Simulation result of Matrix Multiplication using Vedic multiplier and KSA

5.2.6 Matrix Multiplication using Vedic multiplier and HCA



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	5100	92152	5%
Number of fully used LUT-FF pairs	0	5100	0%
Number of bonded IOBs	256	396	64%

Fig 5.21. Device Utilization of Matrix Multiplication using Vedic multiplier and HCA



```
                                Gate     Net
Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
 IBUF:I->O           112   1.328   2.522  A_32_IBUF (A_32_IBUF)
 LUT4:I0->O            2   0.254   1.156  m5/u81/u41/u23/Mxor_q<1>_xo<0>1 (m5/u81/u41/q2<1>)
 LUT6:I1->O            2   0.254   1.156  m5/u81/u41/R1/F2/carry1 (m5/u81/u41/R1/y<1>)
 LUT6:I1->O            4   0.254   1.032  m5/u81/u41/R1/F8/carry1 (m5/u81/u41/R1/w2)
 LUT3:I0->O            1   0.235   0.682  m5/u81/u41/R1/F9/Mxor_sum_xo<0>1 (m5/u81/q0<5>)
 LUT6:I5->O            2   0.254   0.954  m5/u81/R12/F2/carry1 (m5/u81/R12/y<1>)
 LUT6:I3->O            3   0.235   0.874  m5/u81/R12/H1/c221 (m5/u81/R12/H1/c22)
 LUT5:I3->O            3   0.250   1.196  m5/u81/R12/H1/c31 (m5/u81/R12/H1/c3)
 LUT6:I1->O            2   0.254   0.834  m5/u81/R12/H1/Mxor_sum<4>_xo<0>1 (m5/q0<9>)
 LUT6:I4->O            2   0.250   0.954  m5/R24/F2/carry1 (m5/R24/y<1>)
 LUT6:I3->O            3   0.235   0.874  m5/R24/k1/c12 (m5/R24/k1/c1)
 LUT5:I3->O            3   0.250   0.874  m5/R24/k1/c31 (m5/R24/k1/c3)
 LUT5:I3->O            7   0.250   1.018  m5/R24/k1/c51 (m5/R24/k1/c5)
 LUT5:I3->O            3   0.250   0.874  m5/R24/k1/c71 (m5/R24/k1/c7)
 LUT5:I3->O            1   0.250   1.137  m5/R24/k1/c111 (m5/R24/k1/c111)
 LUT6:I0->O            4   0.254   1.032  m5/R24/k1/c112 (m5/R24/k1/c11)
 LUT6:I3->O            1   0.235   0.790  m5/R24/k1/c154 (m5/R24/k1/c154)
 LUT6:I4->O            4   0.250   0.804  m5/R24/k1/c155 (m5/R24/k1/c15)
 LUT4:I3->O            3   0.254   1.042  m5/R24/k1/Mxor_sum<16>_xo<0>1 (y1<1><0><25>)
 LUT4:I0->O            4   0.254   1.234  k3/p24_251 (k3/p24_25)
 LUT6:I1->O            2   0.254   0.726  k3/g22_251 (k3/g22_25)
 LUT5:I4->O            1   0.254   0.790  k3/c294_SW0 (N188)
 LUT6:I4->O            2   0.250   0.834  k3/c294 (k3/c29)
 LUT5:I3->O      |     1   0.250   0.681  k3/Mxor_sum<31>_xo<0>1 (Y_95_OBUF)
 OBUF:I->O                 2.912          Y_95_OBUF (Y<95>)
----------------------------------------
Total                    34.040ns (9.970ns logic, 24.070ns route)
                                  (29.3% logic, 70.7% route)
```

Fig 5.22. Delay Report of Matrix Multiplication using Vedic multiplier and HCA

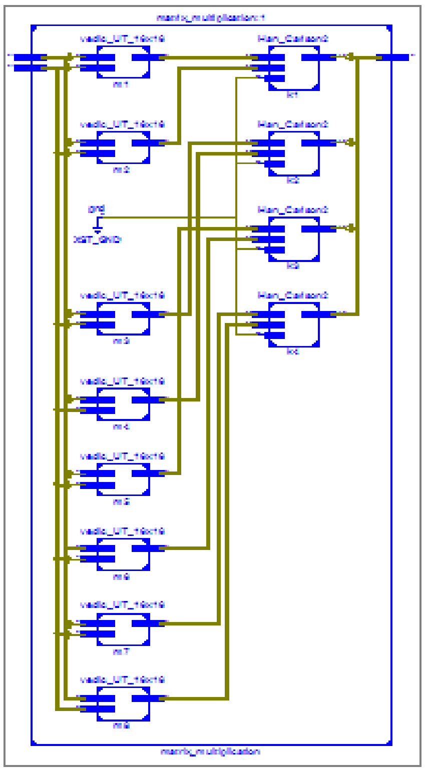
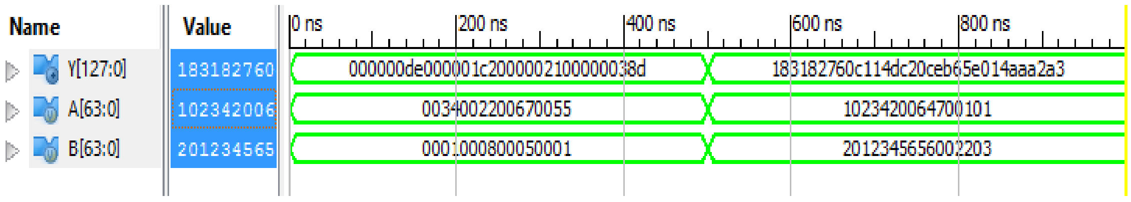


Fig 5.23. Schematic of Matrix Multiplication using Vedic multiplier and HCA



```
Name        Value        0 ns        200 ns        400 ns        600 ns        800 ns
Y[127:0]    183182760    000000de000001c200000210000038d    183182760c114dc20ceb65e014aaa2a3
A[63:0]     102342006    0034002200670055                   1023420064700101
B[63:0]     201234565    0001000800050001                   2012345656002203
```

Fig 5.24. Simulation result of Matrix Multiplication using Vedic multiplier and HCA

Table 5.1 shows Comparative Analysis Of Matrix Multiplication using Different   Multipliers

Matrix Multiplier Method	Delay(ns)	Area(LUTs)
Wallace Tree Multiplier and RCA	28.969	4009
Wallace Tree Multiplier and KSA	27.544	4972
Wallace Tree Multiplier and HCA	26.583	4528
Vedic Multiplier and RCA	32.998	4864
Vedic Multiplier and KSA	33.057	5523
Vedic Multiplier and HCA	34.04	5100

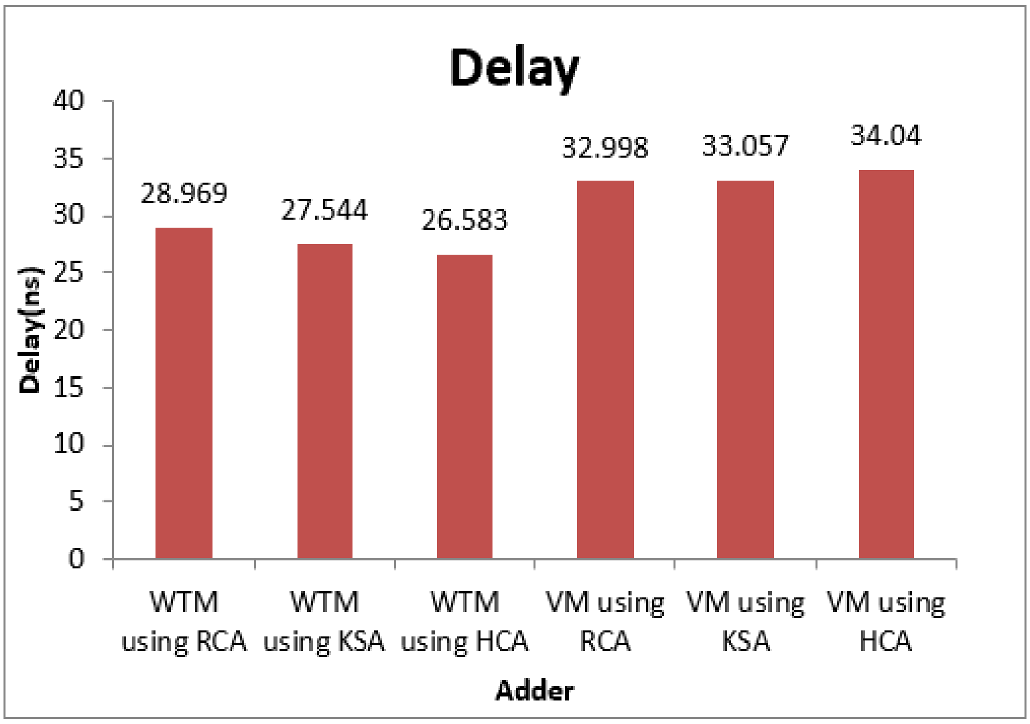


Fig 5.25 Delay comparison graph for Matrix Multiplication

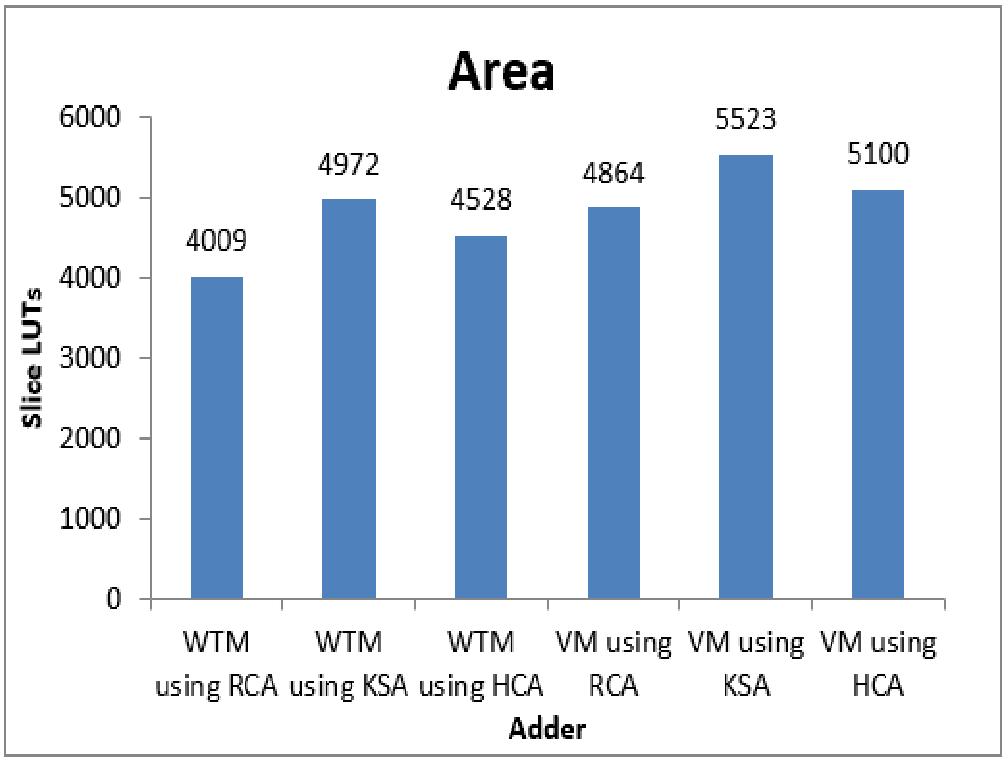


Fig 5.26. Area comparison graph for Matrix Multiplication

5.3 Important Outcomes:

The thesis provides an integrated framework for matrix multiplication that integrates the Wallace tree multiplier, Vedic multiplier, and parallel prefix adders. Comparing the framework to conventional multiplication methods, it shows enhanced performance and efficiency.

- It demonstrates that the Wallace tree multiplier outperforms the Vedic multiplier in terms of speed and area utilization.
- The quickest matrix multiplication method uses a Wallace tree multiplier and a Han Carlson adder.
- On comparing their area requirements, the most efficient multiplier is the Wallace tree multiplier combined with a Ripple carry adder.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

This thesis has presented a comprehensive investigation into the optimization of matrix multiplication through the utilization of the Wallace tree multiplier, Vedic multiplier, and parallel prefix adders. This efficiency in improving the performance of matrix multiplication operations has been revealed by the performance evaluation and analysis, which has been quite helpful.

The best superior 32 bits adder is first designed. The Han Carlson adder provides higher performance in terms of speed, according to examination and comparison with the Kogge-Stone adder and ripple carry adder. The Han Carlson adder combines the advantages of the Han-Carlson parallel prefix structure and the carry-save adder, resulting in a reduction of critical path delay and efficient carry propagation. The Han Carlson adder reduces the delay by around 40% compared to the Ripple Carry adder. In terms of area, due to their added circuitry for parallel processing and carry propagation, the HAN-Carlson adder and Kogge-Stone adder often demand more hardware resources than the ripple carry adder.

The performance of 16x16 multipliers is then designed and compared after that. Vedic and Wallace tree multipliers are successfully designed with Han Carlson, Kogge Stone, and Ripple Carry adders, and their performance is compared. According to the results, the Wallace tree multiplier and Han-Carlson adder performed faster than the other configuration. Faster multiplication operations were made possible by the Wallace tree multiplier's parallel architecture, and overall performance was improved because the Han-Carlson adder considerably decreased the propagation delay of carry bits.

Matrix Multiplication is implemented using the designed multipliers and adders. By leveraging the strengths of the Wallace tree multiplier, Vedic multiplier, and parallel prefix adders, the integrated framework offers a powerful approach for efficient matrix multiplication, paving the way for advancements in various computational domains. The performance evaluation demonstrates that the combination of the Wallace tree multiplier and the Han-Carlson adder achieves the fastest matrix multiplication results compared to other techniques.

6.2 FUTURE SCOPE

The future scope of this thesis includes a number of prospective directions for additional study and advancement. While the thesis has focused on the performance comparison of matrix multiplication with the Wallace tree multiplier and the Vedic multiplier using different adders, future studies can investigate other advanced multiplier designs.

The following points can be used to extend and modify the SA. Improved circuit design can further reduce delay.

- The thesis has highlighted optimization techniques such as memory access optimization, and pipelining to improve the performance of matrix multiplication. To further optimize the calculation process and lower the computational complexity, future studies can go deeper into algorithmic improvements, such as using Strassen's algorithm or the Coppersmith-Winograd algorithm for matrix multiplication.
- Matrix Multiplication is frequently utilized in many different fields, and each field could have particular needs and limitations. Research in the future can concentrate on selecting the multiplier and optimizing the optimization approaches for certain applications.

These areas of study will aid in the development and improvement of matrix multiplication methods as well as their adaptability to various computational fields.

REFERENCES

[1] Y. d. Ykuntam, K. Pavani and K. Saladi, ”Design and analysis of High speed wallace tree multiplier using parallel prefix adders for VLSI circuit designs,” 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2020, pp. 1-6.

[2] T. Gupta and J. B. Sharma, “Han–Carlson adder based high-speed Vedic multiplier for complex multiplication,” Microsystem Technologies, vol. 24, no. 9, 2018,pp. 3901–3906.

[3] A. Raju and S. K. Sa, ”Design and performance analysis of multipliers using Kogge Stone Adder,” 2017 3rd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), Tumkur, India, 2017, pp. 94-99.

[4] P. Gulati, H. Yadav and M. K. Taleja, ”Implementation of an efficient multiplier using the vedic multiplication algorithm,” 2016 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, India, 2016, pp. 1440-1443.

[5] S. Lad and V. S. Bendre, ”Design and Comparison of Multiplier using Vedic Sutras,” 2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA), Pune, India, 2019, pp. 1-5.

[6] S. Dubey and G. Verma, ”Analysis of Basic Adder with Parallel Prefix Adder,” 2020 First IEEE International Conference on Measurement, In- strumentation, Control and Automation (ICMICA), Kurukshetra, India, 2020, pp. 1-6.

[7] M. Kivi Sona and V. Somasundaram, “Vedic multiplier implementation in VLSI,” Mater. Today, vol. 24,2020, pp. 2219–2230.

[8] R. Shanmuganathan and K. Brindhadevi, “Comparative analysis of various types of multipliers for effective low power,” Microelectron. Eng., vol. 214, 2019,pp. 28–37.

[9] R. Anjana, B. Abishna, M. S. Harshitha, E. Abhishek, V. Ravichandra and M. S Suma, ”Implementation of vedic multiplier using Kogge-stone adder,” 2014 International Conference on Embedded Systems (ICES), Coimbatore, India, 2014, pp. 28-31.

[10] A. Sundhar, S. D. Tharshini, G. Priyanka, S. Ragul and C. Saranya, ”Performance Analysis of Wallace Tree Multiplier with Kogge Stone Adder using 15-4 Compressor,” 2019 International Conference on Com- munication and Signal Processing (ICCSP), Chennai, India, 2019, pp. 0903-0907.

[11] M. N. Chandrashekara and S. Rohith, ”Design of 8 Bit Vedic Multiplier Using Urdhva

Tiryagbhyam Sutra With Modified Carry Save Adder,” 2019 4th International Conference on Recent Trends on Electronics, Information, Communication Technology (RTEICT), Bangalore, India, 2019, pp. 116-120.

[12] N. Kumar M., R. S. Adithyaa, B. Kumar D. and T. Pavithra, ”Design Analysis of Wallace Tree based Multiplier using Approximate Full Adder and Kogge Stone Adder,” 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coim- batore, India, 2020, pp. 612-616.

[13] U. Penchalaiah and S. K. VG, "Design of High-Speed and Energy-Efficient Parallel Prefix Kogge Stone Adder," 2018 IEEE International Conference on System, Computation, Automation and Networking (ICSCA), Pondicherry, India, 2018, pp. 1-7, doi: 10.1109/ICSCAN.2018.8541143.

[14] P. Mehta and D. Gawali, "Conventional versus Vedic Mathematical Method for Hardware Implementation of a Multiplier," 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies, Bangalore, India, 2009, pp. 640-642, doi: 10.1109/ACT.2009.162.

[15] D. R. Gandhi and N. N. Shah, "Comparative analysis for hardware circuit architecture of Wallace tree multiplier," 2013 International Conference on Intelligent Systems and Signal Processing (ISSP), Vallabh Vidyanagar, India, 2013, pp. 1-6, doi: 10.1109/ISSP.2013.6526864.

[16] S. K. Yezerla and B. Rajendra Naik, "Design and estimation of delay, power and area for Parallel prefix adders," 2014 Recent Advances in Engineering and Computational Sciences (RAECS), Chandigarh, India, 2014, pp. 1-6, doi: 10.1109/RAECS.2014.6799654.

[17] J. Liu, Y. Zhu, H. Zhu, C. -K. Cheng and J. Lillis, "Optimum Prefix Adders in a Comprehensive Area, Timing and Power Design Space," 2007 Asia and South Pacific Design Automation Conference, Yokohama, Japan, 2007, pp. 609-615, doi: 10.1109/ASPDAC.2007.358053.

[18] C. N. Shilpa, K. D. Shinde and H. V. Nithin, "Design, Implementation and Comparative Analysis of Kogge Stone Adder Using CMOS and GDI Design: A VLSI Based Approach," 2016 8th International Conference on Computational Intelligence and Communication Networks (CICN), Tehri, India, 2016, pp. 570-574, doi: 10.1109/CICN.2016.117.