DESIGN AND IMPLEMENTATION OF EFFICIENT REVERSIBLE MULTIPLIER USING VEDIC MATHEMATICS TOOL

A

Thesis

Submitted in partial fulfilment of the requirements

for the award of the degree of

MASTER OF TECHNOLOGY

in

VLSI DESIGN & EMBEDDED SYSTEMS

SUBMITTED BY

Ms. DIKSHA RUHELA

University Roll No: 2k13/VLSI/06

UNDER THE GUIDANCE OF

Dr. MALTI BANSAL

(Assistant Professor)

Department of Electronics & Communication Engineering

Delhi Technological University, Delhi

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

DELHI-42

2013-2015

DELHI TECHNOLOGICAL UNIVERSITY

Department of Electronics & Communication Engineering

CERTIFICATE

This is to certify that the dissertation entitled "**Design and Implementation of Efficient Reversible Multiplier using Vedic Mathematics Tool**" is a bonafide work of **Diksha Ruhela** (University Roll No. 2K13/VLSI/06), a student of Delhi Technological University. This project was carried out under my direct supervision and guidance and forms a part of the Master of Technology Course in **Electronics and Communication Engineering** with specialization in "**VLSI Design and Embedded Systems**" at Delhi Technological University, Delhi.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University/ Institute for the award of any Degree or Diploma.

Date:

DR. Malti Bansal

(Project Guide)
Assistant Professor
Department Of Electronics &
Communication Engineering
Delhi Technological University

Department of Electronics and Communication Engineering

Delhi Technological University

(Formerly Delhi College of Engineering)

Bawana Road, Delhi- 110042

CANDIDATE'S DECLARATION

I, **Diksha Ruhela**, Roll No. **2k13/VLSI/06**, student of **M.Tech (VLSI Design and Embedded Systems)**, hereby declare that the dissertation entitled "**Design and Implementation of Efficient Reversible Multiplier using Vedic Mathematics Tool**", under the supervision of **Dr. Malti Bansal**, Assistant Professor, Electronics and Communication Engineering Department, Delhi Technological University, in partial fulfilment of the requirements for the award of the degree of Master of Technology in VLSI Design and Embedded Systems, has not been submitted elsewhere for the award of any other degree or diploma.

I hereby solemnly and sincerely affirm that all the particulars stated above by me are true and correct to the best of my knowledge and belief.

Place: **Delhi** **Diksha Ruhela**

Date: **2k13/VLSI/06**

ACKNOWLEDGEMENT

My deepest respect and appreciation goes to my advisor, **Dr. Malti Bansal**, **Assistant Professor**, Department of Electronics & Communication Engineering, Delhi Technological University, for her guidance, support and encouragement provided in my expedition towards the Master's Degree in Technology. I will forever be grateful for her endless advice, incredible patience, generosity and friendship. Without her help and guidance, this dissertation would have never been possible.

I am also grateful to **Prof. Prem R. Chadha, (Head of Department)** Department of Electronics and Communication, Delhi Technological University, for his support.

I am grateful to my father, mother and brother for their enduring love, immense moral support and encouragement throughout my life and especially during this project.

DIKSHA RUHELA

2k13/VLSI/06
M.Tech
(VLSI DESIGN
& EMBEDDED SYSTEMS)

ABSTRACT

Multiplier is one of the important block in almost all the arithmetic logic units. These multipliers are mostly used in the fields of the Digital Signal Processing (DSP), Fast Fourier Transform, convolution, filtering and microprocessor applications. A system's performance is generally determined by the performance of the multiplier, because the multiplier is generally the slowest element in the system. Furthermore, it is generally the most area consuming. Hence, optimizing the speed and area of the multiplier is a major design issue. Since multiplier is the main component and hence a high speed and area efficient multiplier can be achieve by using Vedic mathematics. In this work we have implemented the Vedic multiplier using Chinese Abacus Adder with and without using Reversible logic gates.

Reversible logic is one of the promising fields for future low power design technologies. Since one of the requirements of all DSP processors and other embedded devices is to minimize power dissipation multipliers with high speed and lower dissipations are critical.

This work is devoted to the design of a high speed Vedic multiplier using reversible logic gates. For arithmetic multiplication, various Vedic multiplication techniques like *Urdhva Tiryakbhyam, Nikhilam* and *Anurupye* have been thoroughly discussed. It has been found that *Urdhva Tiryakbhyam* Sutra is the most efficient Sutra (Algorithm), giving minimum delay for multiplication of all types of numbers, either small or large.

Further, the Verilog HDL coding of *Urdhva Tiryakbhyam* Sutra for 32x32 bits and 64x64 bits multiplication and their FPGA implementation by Xilinx Synthesis Tool on Spartan 3E kit have been done. The synthesis results show that the computation time for calculating the product of 4x4 multiplication is less as compared with other conventional multipliers.

REFEREED PUBLICATIONS

[1] Dr. Malti Bansal, **Diksha Ruhela**, "High Speed & Area Efficient Vedic Multiplier using Adiabatic Logic", Journal of Basic and Applied Engineering Research Volume 1, Number 11; October-December 2014 pp. 14-17.

[2] **Diksha Ruhela** & Dr. Malti Bansal, "Vedic Multiplier with Chinese Abacus Adder Design using Reversible Logic Gates", International Conference on VLSI, Communication and Network (VCAN-2015), Alwar-301030, ISBN:978-93-84869-55-7, April – 2015 pp. 9-12.

[3] **Diksha Ruhela** & Dr. Malti Bansal, "Adiabatic Vedic Multiplier Design Using Chinese Abacus Approach", International Journal of Advanced Research in Computer and Communication Engineering, ISSN (Online) 2278-1021 ISSN (Print) 2319-5940 Vol. 4, Issue 4, April 2015.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADSP	:	Advanced Digital Signal Processing
ASIC	:	Application-Specific Integrated Circuit
ATE	:	Automatic Test Equipment
ATPG	:	Automatic Test Pattern Generator
BIST	:	Built In Self-Test
CIAF	:	Computation Intensive Arithmetic Functions
CLB	:	Combinational Logic Blocks
CPLD	:	Complex Programmable Logic Device
CUT	:	Circuit Under Test
DFT	:	Design for Test
DFT	:	Discrete Fourier Transforms
DSP	:	Digital Signal Processing
FFT	:	Fast Fourier Transforms
FPGA	:	Field Programming Gate Array
IC	:	Integrated Circuits
IFFT	:	Inverse Fast Fourier Transforms
IOB	:	Input Output Blocks
IOP	:	Input Output Pins
ISE	:	Integrated Software Environment
JTAG	:	Joint Test Action Group
LFSR	:	Linear Feedback Shift Register
MAC	:	Multiply and Accumulate
NCD	:	Native Circuit Description
NGC	:	Native Generic Circuit
NGD	:	Native Generic Database
ORA	:	Output Response Analyser
PAR	:	Place And Route

PLA	:	Programmable Logic Arrays
PRPG	:	Pseudo Random Pattern generator
ROM	:	Read Only Memory
RPG	:	Random Pattern Generation
RTL	:	Register Transfer Level
SOPC	:	System-On - A - Programmable-Chip
SR	:	Signature Registers
TPG	:	Test Pattern Generator
UCF	:	User Constraints File
UT	:	Urdhva Triyakbhyam
VM	:	Vedic Mathematics

CHAPTER-1

INTRODUCTION

1.1 MOTIVATION

Multipliers are one of the most important functional block in digital filters such as Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) Filters. These filters are used in a wide variety of Digital Signal Processing (DSP) applications such as A/D converters, signal modulators, audio signal processing, multimedia, and process control just to name a few.

Digital multipliers are the one of the repeated used essential block in any digital circuit design. They are having high speed of operation, most reliable and energy efficient components that are mainly used to utilized for implementing any operation. With the increase in demand of various DSP application led to increases the demand of high speed processing. One of the vital function of arithmetic operations in such applications is multiplication and the implementation of high speed multiplier circuit has been a subject of interest over decades. Arithmetic operations having higher throughput are essential to achieve the desired performance in many real-time signal and image processing applications [2]. This work presents different multiplier architectures comparison with the proposed Vedic multiplier in area, power and speed prospect. Multiplication-based operations includes Multiply and Accumulate (MAC) operation and inner product are one of the frequently used Computation- Intensive Arithmetic Functions (CIAF) currently implemented and designed for many Digital Signal Processing (DSP) applications that includes convolution, Fast Fourier Transform (FFT), audio signal processing , signal modulator, filtering and in microprocessors in its arithmetic and logic unit and many others[1]. Currently, multiplication time is still the dominant factor in determining the instruction cycle time of a DSP chip. Most of the execution time in DSP algorithms is dominated in multiplication process only, so there is a need of high speed along with less power dissipation and area efficient multiplier. Reducing the time delay and power consumption are one of the very essential requirements for many applications [2, 3].

Multiplier based on Vedic Mathematic using Chinese Abacus Adder is one of the fast and low power consumption multiplier**.** Minimizing power consumption and area for digital systems involves optimization at all levels of the design. This optimization process includes the technology used for implementing the digital circuits or digital logic gates, the architecture view for implementing the circuits, the circuit style and circuit topologies, and at the highest level of implementation stage, the algorithms that are being implemented for its design of operation. Multiplier is also available in different type depending upon the arrangement of the components, thus it provides the availability to the designer to choose the particular type of multiplier architecture based on the application. Initially multiplication operation was implemented generally with a sequence of addition, subtraction and shift operations. Many algorithm proposals have been found in literature to perform multiplication operation, each offering different advantages and having trade off in terms of speed of operation, circuit design complexity, area usage and power consumption.

In many DSP operations, the multiplier plays a vital role in determining the speed of operation. It mainly lies in the critical delay path and ultimately determines the performance of the algorithm. For any general purpose processor includes DSP the speed of multiplication operation is of great importance. For many computing system the multiplier is a fairly large block and therefore minimizing the circuitry involve in its designing is also an area of research. The amount of circuitry which is involved in designing is directly proportional to the square of its resolution i.e., a multiplier of size 'n' bits has n^2 gates. For performing the multiplication algorithms in DSP applications latency and throughput are the two major concerns area seen from delay perspective. Latency is defined as the amount of real delay in computing a function, a measure of how long the inputs to a device remains stable, till the final result available at outputs. Throughput is defined as the measuring of the count of multiplications performed in a given period of time. It is found that multiplier is not only a high delay block but also having a

source of power dissipation. That's why it is also one of the major aim of designer to design the block having minimum power consumption. Delay optimization technique can be used to reduce the delay associated with the block.

In all the digital signal processors (DSPs), digital multiplier is one of among core component and the speed of the DSP at high extend is mainly determined by the computation speed of its multipliers. There are many algorithm design for doing the multiplication in past decades, the most commonly used multiplication algorithm in the digital hardware are booth multiplication algorithm and array multiplication algorithm. Booth multiplication is one of the important multiplication algorithm used in digital hardware. In this algorithm large booth arrays are required for doing the high speed multiplication and exponential operations which in turns to require large number of partial sum and partial carry registers. For doing the multiplication of any two *n*-bit operands using a radix-4 booth multiplier requires approximately $n / (2m)$ clock cycles to generate the least significant half of the final product, where *m* is the number of Booth recorder adder stages. Thus, it associates a large propagation delay. Whereas in case of Array algorithm the computation time taken by the multiplier is comparatively less because the partial products which are generated are calculated independently in parallel. In the array multiplier the delay is basically associated with the time taken by the signals to propagate through the gates that form the multiplication array.

In this thesis, to design the high speed & area efficient digital multiplier architecture, *Urdhva Tiryakbhyam Sutra* along with the Chinese abacus approach is used. This architecture is seem to be very similar to one of the popularly designed array multiplier architecture. The effectiveness of this Sutra is to reduce the NXN multiplier structure into some efficient low order radix multiplier structures. The proposed multiplication algorithm is illustrated to show its computational efficiency by taking an example of reducing a 4X4-bit multiplication to a single 2X2-bit multiplication operation [4]. This proposed work presents a systematic design

for fast and area efficient digit multiplier for multiplying binary number system based on Vedic mathematics using Chinese abacus adder. The Multiplier Architecture is based on one of the famous Vertical and Crosswise algorithm of ancient Indian Vedic Mathematics [5].

1.2 OBJECTIVE

The objective of this work is to design a high speed, area efficient and less power dissipated multiplier which can be used in any DSP as well as other processors application. This work deals with the study, design and implementation of Vedic multiplier using Chinese abacus adder. In this work, study of Vedic multiplication, has been explored. Architecture of Vedic multiplier based on speed, area and power dissipation specification is designed here. Hardware Implementation of this multiplier has been done on Spartan 3E Board.

1.3 TOOLS USED

Simulation Software: Modelsim6.1e has been used for simulation. ISE9.2i (Integrated system environment) has been used for synthesis and verification.

Hardware used: Xilinx Spartan3E (Family), XC3S500 / XC3S1600 (Device), FG320 / FG484 (Package), -5 (Speed Grade) FPGA devices.

CHAPTER-2
VEDIC MULTIPLICATION ALGORITHMS

2.1 HISTORY OF VEDIC MATHEMATIC

Sri Bharati Krsna Tirthaji (1884-1960) rediscovered the maths of the ancient system between the period of 1911 and 1918, known as “Vedic Mathematics”, from the Sanskrit text known as the Vedas. At the initial period of the twentieth century, when Europe was showing a great interest in Sanskrit text, Bharati Krsna tells us about some scholars ridiculed certain texts which were headed 'Ganita Sutras'- which means mathematics. At that time they failed in finding the mathematics during the translation and dismissed the texts as rubbish. Bharati Krsna, who was himself a scholar of Sanskrit, Mathematics, History and Philosophy, studied these texts thoroughly and after lengthy investigation he was able to reconstruct the mathematics of the Vedas. At his research he found that all of mathematics is based on sixteen Sutras or word-formulae.

All the several modern mathematical terms including arithmetic, geometry (plane, co-ordinate), trigonometry, quadratic equations, factorization and even calculus has been covered in that. For example, 'Vertically and Crosswise` is one of these Sutras. These formulae have a great logic for directing one mind in natural way, this formulae actually describe the way mind naturally works. It is a very appropriate method for studying the maths and generating one interest in subject.

The interesting feature of the Vedic system is its coherence. The whole system is unified, systematic and interrelated with each other instead of using the hotch-potch of unrelated techniques. And these are all easily understood. This unifying quality make it very attractive and satisfying, it not only makes the mathematics easy but also make it enjoyable and encourages innovation.

In the Vedic system, Vedic method is often used for solving the large sum or 'difficult' problems. These innovative and striking methods are just make a part of a complete system of mathematics which is far more systematic, unified and interrelated than the modern 'system'. Vedic Mathematics manifests the coherent and unified structure of mathematics and the methods are complementary, direct and easy.

Because of the beauty of Vedic mathematic which claimed to work on the natural principles on which the human minds works helps to reduce the cumbersome-looking calculations come across in conventional mathematics to a very easy simple one. It is one of the beauty of Vedic mathematics. This is one of the very interesting and attractive field and presents some effective algorithms which can be used to applied in various branches of engineering such as computing and digital signal processing [8, 9].

Vedic Mathematics composed the part of Jyotish Shastra which is itself a parts of Vedangas. The word “Vedic” is originated from the word “Veda” which means the store-house of all knowledge. Vedic mathematics is generally based on 16 Sutras (or aphorisms) dealing with various branches of mathematics like arithmetic, algebra, geometry, etc. These Sutras along with their brief meanings are enlisted below alphabetically.

1) (*Anurupye*) Shunyamanyat – If one is in ratio, the other is zero.

2) Chalana-Kalanabyham – Differences and Similarities.

3) Ekadhikina Purvena – By one more than the previous One.

4) Ekanyunena Purvena – By one less than the previous one.

5) Gunakasamuchyah – The factors of the sum is equal to the sum of the factors.

6) Gunitasamuchyah – The product of the sum is equal to the sum of the product.

7) *Nikhilam* Navatashcaramam Dashatah – All from 9 and last from 10.

8) Paraavartya Yojayet – Transpose and adjust.

9) Puranapuranabyham – By the completion or non-completion.

10) Sankalana- vyavakalanabhyam – By addition and by subtraction.

11) Shesanyankena Charamena – The remainders by the last digit.

12) Shunyam Saamyasamuccaye – When the sum is the same that sum is zero.

13) Sopaantyadvayamantyam – The ultimate and twice the penultimate.

14) Urdhva-tiryakbhyam – Vertically and crosswise.

15) Vyashtisamanstih – Part and Whole.

16) Yaavadunam – Whatever the extent of its deficiency.

As mentioned earlier, all the above mentioned Sutras were rediscovered from ancient Vedic texts in the early period of last century. Many Sub-sutras were also discovered at the same time, which are not discussed here. These methods and ideas which have been discovered during that period of time can be directly applied to system of mathematical tools like trigonometry, plain and spherical geometry, conics, calculus (both differential and integral), and applied mathematics of various kinds.

The multiplier architecture based upon its architecture is generally classified into three categories. First is the serial multiplier which mainly emphasizes on hardware and minimum amount of chip area consumption. Second is parallel multiplier (array and tree) which is mainly used to carries out high speed mathematical operations. But the drawback of parallel multiplier is the consumption of relatively larger chip area. Third is the combination of serial- parallel

multiplier which serves as a good trade-off between the time consuming serial multiplier and the area consuming parallel multiplier.

2.2 ALGORITHMS OF VEDIC MATHEMATICS

2.2.1 VEDIC MULTIPLICATION

The proposed Vedic multiplier presented in this work is design using the Vedic multiplication formulae (Sutras). Traditionally in the Vedic mathematics the Sutras have been used for the multiplication of numbers belongs to the decimal number system. In this work, we apply the same ideas of multiplication for multiplying the numbers belongs to the binary number system to make the proposed algorithm compatible with the digital hardware. Vedic multiplication is based on some algorithms; some are discussed below:

2.2.1.1 *URDHVA TIRYAKBHYAM* SUTRA

Urdhva – tiryagbhyam (UT) is the general formula applicable to all cases of multiplication and also in the division of a large number by another large number. It means "Vertically and cross wise." The concept behind of this sutra is the generation of all partial products can be done with the concurrent addition of these partial products. The operation of parallelism in generation of partial products and their concurrent addition is obtained using *Urdhava Triyakbhyam* explained in fig 2.1. The algorithm can be generalized for order of radix. Due to parallel operation of the generation of the partial products and their sums, the multiplier is independent of the clock frequency of the processor. Thus, the multiplier will require the less amount of time comparative to conventional method to calculate the product and hence is independent of the clock frequency. The net advantage is that it reduces the frequent interference of microprocessors to operate at increasingly high clock frequencies. While a higher clock frequency generally results in increased processing power, its disadvantage is that

it also increases power dissipation which results in higher device operating temperatures. These problems of higher power dissipation can easily be circumvent by adopting this Vedic multiplier in place of other conventional multiplier by microprocessors designer in order to avoid catastrophic device failures. Due to regular structure, on increasing the input and output data bus widths the processing power of the multiplier can be easily varied or increased and also it can also be easily fabricated in a silicon chip. The Multiplier has the merit that on increasing the number of bits, area and gate delay increases very slowly as compared to other multipliers. Therefore it is area, time and power efficient. It is showing that this architecture is quite efficient in terms of silicon area and speed [10, 4].

1) Multiplication of two decimal numbers- 325*738

To illustrate this multiplication scheme, let us consider two decimal numbers (325 * 738). Diagram for the multiplication of these number is shown in Fig.2.2. It has been shown below that the digits on the both sides of the line are multiplied and added with the carry obtained from the previous step. This generates calculation results into one of the bits of the final result and a carry. This generated carry of the previous stage is added in the next step of the calculation and hence the process goes on till all the digits get multiplied. If more than one line are there in one step, all the results are added to the previously generated carry. In each step, least significant bit generated acts as the result bit of that product and all other bits act as carry for the next step. Initially, the carry is taken to be zero. To make this methodology more clear, an alternate illustration by taking an example is given with the help of line diagrams in figure 2.2 where the dots represent bit “0” or “1”[4].

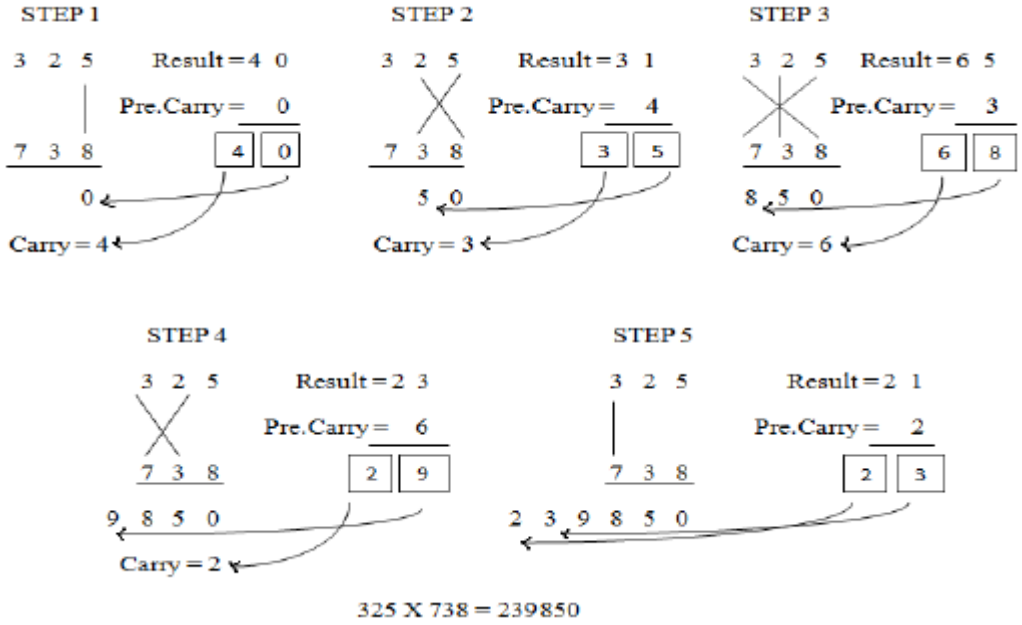


Figure 2.1: Multiplication of two three digits decimal numbers by *Urdhva Tiryakbhyam*.

2) Algorithm for the multiplication of two 4 x 4 bit Binary number Using *Urdhva Tiryakbhyam* (Vertically and crosswise) [10]

CP = Cross Product (Vertically and Crosswise)

X3 X2 X1 X0 Multiplicand

Y3 Y2 Y1 Y0 Multiplier

H G F E D C B A

P7 P6 P5 P4 P3 P2 P1 P0 Product

PARALLEL COMPUTATION METHODOLOGY

1. CP X0 = X0 * Y0 = A
   Y0
2. CP X1 X0 = X1 * Y0+X0 * Y1 = B
   Y1 Y0
3. CP X2 X1 X0 = X2 * Y0 +X0 * Y2 +X1 * Y1 = C
   Y2 Y1 Y0
4. CP X3 X2 X1 X0 = X3 * Y0 +X0 * Y3+X2 * Y1 +X1 * Y2 = D
   Y3 Y2 Y1 Y0
5. CP X3 X2 X1 = X3 * Y1+X1 * Y3+X2 * Y2 = E
   Y3 Y2 Y1
6. CP X3 X2 = X3 * Y2+X2 * Y3 = F
   Y3 Y2
7 CP X3 = X3 * Y3 = G
   Y3

3) Algorithm for the multiplication of two 8 X 8 Bit Binary number Using *Urdhva Triyakbhyam* (Vertically and crosswise) [11]

```
A =      A7A6A5A4      A3A2A1A0
          X1              X0
B =      B7B6B5B4      B3B2B1B0
         Y1              Y0
               X1  X0
             * Y1  Y0
-------------------------------------------------------
               F E D C
```

CP = X0 * Y0 = C

CP = X1 * Y0 + X0 * Y1 = D

CP = X1 * Y1 = E

Where CP = Cross Product.

Note: Each Multiplication operation is an embedded parallel 4x4 Multiply module.

To elucidate the multiplication algorithm, let us consider the example of two binary numbers a3a2a1a0 and b3b2b1b0. This 4 digit multiplication terms would results in more than 4 bits, we express it as ..r4r3r2r1r0. Line diagram for the multiplication of two 4-bit numbers is shown in Fig. 2.2 which is nothing but the one to one mapping of the Fig.2.1 in binary system. For easily understanding, each bit is represented by a circle. Least significant bit r0 is simply obtained by multiplying the least significant bits of the four bit multiplicand and the multiplier. The process is followed according to the steps shown in Fig. 2.1.

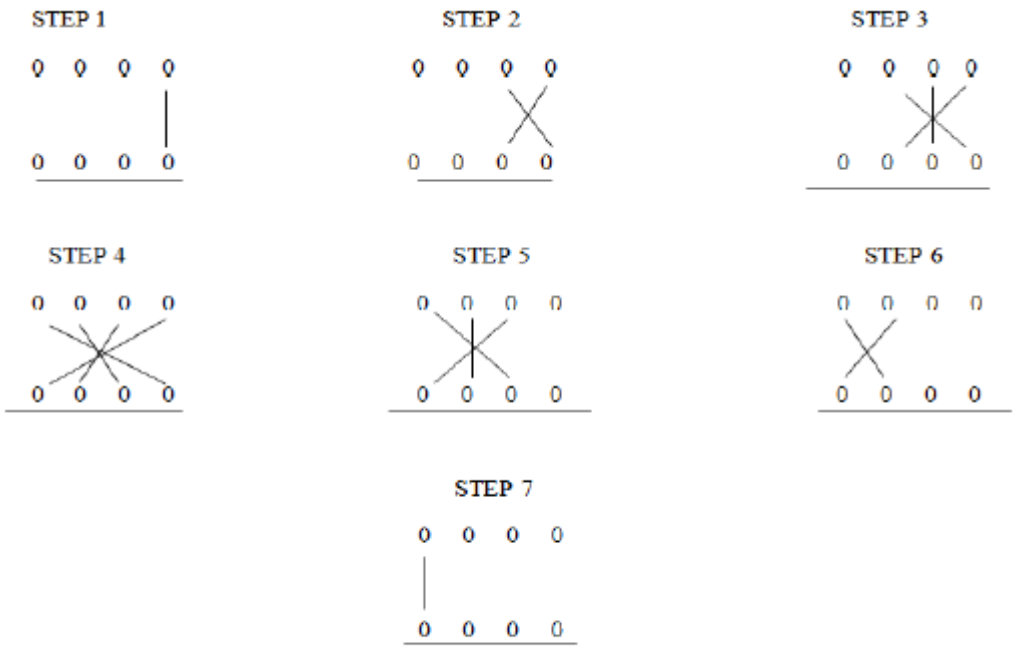


Figure 2.2: Line diagram for multiplication of two 4 - bit numbers.

Firstly, least significant bits of the multiplicand are multiplied with the least significant bit of the multiplier which results in the least significant bit of the product (vertical). Then, the next higher bit of the multiplier is multiply by LSB of the same multiplicand and added with the crosswise product of LSB and next higher bit of the multiplier and multiplicand respectively. The sum which generated from this computation is added into second bit of the product and the carry is generated is added with the output of next stage sum which is obtained by the crosswise and vertical multiplication and addition of three bits of the two numbers from least

significant position. In last, all the four bits are processed likewise with crosswise multiplication and addition to give the final sum and carry. The resultant sum is the result of the product of corresponding bit of the multiplicand and multiplier and the carry which is generated is added to the next stage of multiplication and addition of three bits except the LSB. The same operation processed with the MSBs until the multiplication of the two MSBs of the multiplicand and multiplier each results into the MSB of the product. For example, if in some intermediate step, we get 100, then 0 will act as result bit (referred as r_n) and 10 as the carry (referred as c_n). It should be clearly noted that cn may be a multi-bit number.

Thus we get the following expressions:

$$r0=a0b0; \quad (1)$$

$$c1r1=a1b0+a0b1; \quad (2)$$

$$c2r2=c1+a2b0+a1b1 + a0b2; \quad (3)$$

$$c3r3=c2+a3b0+a2b1 + a1b2 + a0b3; \quad (4)$$

$$c4r4=c3+a3b1+a2b2 + a1b3; \quad (5)$$

$$c5r5=c4+a3b2+a2b3; \quad (6)$$

$$c6r6=c5+a3b3 \quad (7)$$

with c6r6r5r4r3r2r1r0 being results into the final multiplication product. Hence, this is the general mathematical formula applicable to all cases of multiplication.

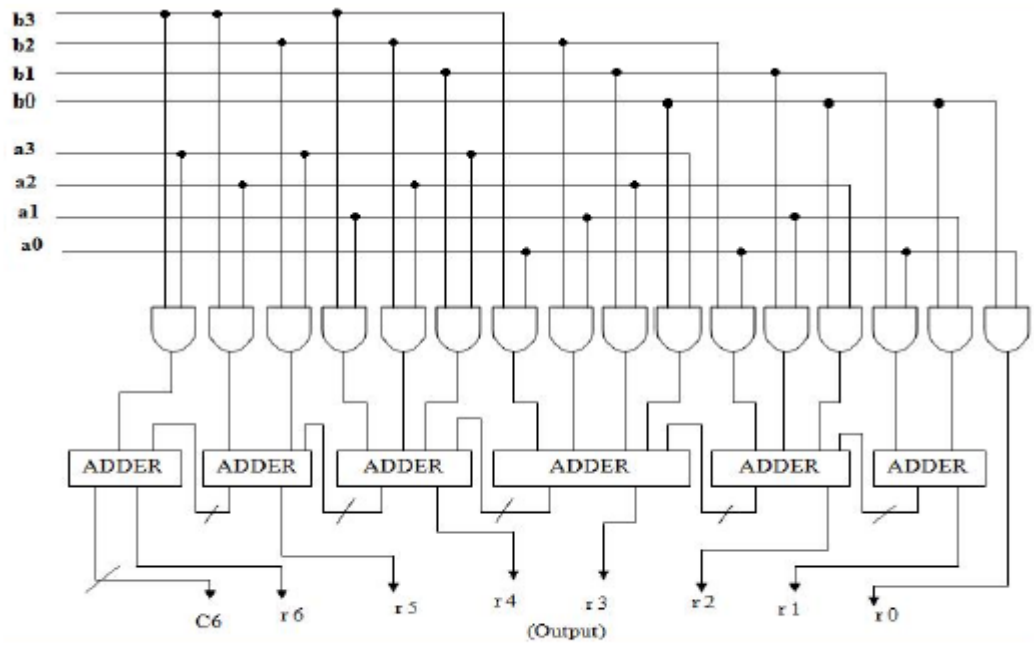


Figure 2.3: Hardware architecture of the Urdhva tiryakbhyam multiplier [4]

The hardware realization of a binary number of 4-bit multiplier is shown in figure2.3. It has been seem that this hardware design shown above is very similar to that of the famous array multiplier where an array of adders is required to arrive at the final product. All the partial products in this algorithm are calculated in parallel and the delay associated is mainly the time consuming process mainly by the carry propagating in each steps through the adders which form the multiplication array. Clearly, it is seem that it is not an efficient or suitable algorithm for doing the multiplication of large numbers as a lot of propagation delay is involved in each consecutive cases and hence led to slow processing speed. To deal with this problem, we now discuss *Nikhilam* Sutra which produce an efficient method of multiplying two large numbers in less propagating carry steps.

2.2.1.2 *NIKHILAM* SUTRA

Nikhilam Sutra literally means "all from 9 and last from 10". This sutras is basically more efficient when the numbers involved for multiplication contain large bits. Since it mechanism

involves in finding out the compliment of the large number from its nearest base to perform the multiplication operation on it. Its basic fundamental is larger is the original number, lesser the complexity of the multiplication it involves. We first illustrate this Sutra by considering the multiplication of two decimal numbers, let us considered the example (96 * 93) whose base is 100. Basically the base is chosen according to the given number which is nearest to and greater than both the given two numbers which has been multiplied.

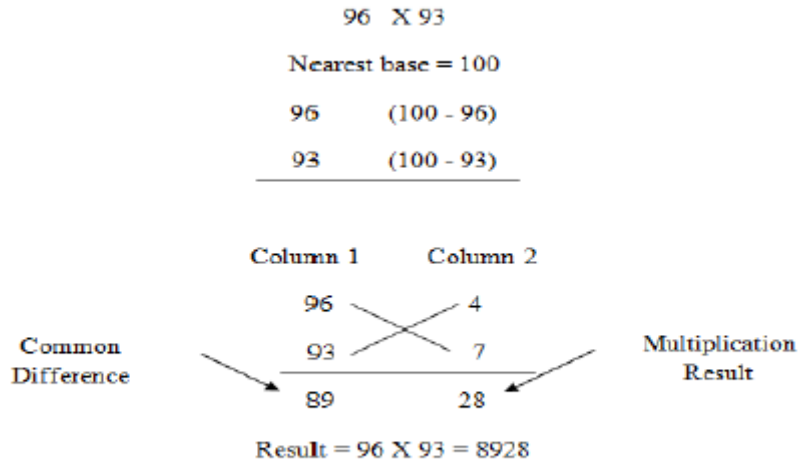


Figure 2.4: Multiplication Using Nikhilam Sutra [4]

The right hand side (RHS) of the given product can be obtained by simply multiplying the numbers of the Column 2 (7*4 = 28) as shown above. The left hand side (LHS) of the product of the given number can be found by cross subtracting the second number of Column 2 from the first number of Column 1 or vice versa, i.e., 96 - 7 = 89 or 93 - 4 = 89 as shown above by line illustration. The final result is obtained by concatenating RHS and LHS (Answer = 8928) [4].

2.3 PERFORMANCE

2.3.1 POWER:

In the study we have found that the Vedic Multiplier requires lesser number of gates comparative to other algorithm for given NxN bits Multiplier so its power dissipation is very small as compared to other multiplier architectures studied so far. Because of the less switching operation of its architecture comparative to other architectures for same operation.

2.3.2 SPEED:

Vedic multiplier algorithm results in one of the fastest algorithm comparative to the other algorithm of multiplier like array multiplier and Booth multiplier. As the number of bits increases from NxN bits to N*2xN*2 bits, the timing delay is greatly reduced comparatively to the other multiplier. Vedic multiplier has the greatest advantage as compared to other multipliers on various parameter like gate delays and regularity of structures. In the comparative study of multiplication for 16 x 16 bit number in Vedic, Booth and array multiplier it is found that the delay in Vedic multiplier for 16 x 16 bit number is 32 ns while the delay in Booth and Array multiplier are 37 ns and 43 ns respectively [12]. Thus, this analysis illustrate that Vedic multiplier shows the highest speed among other conventional multipliers.

2.3.3 AREA:

In the comparative study and on analysis it is found that the area needed for Vedic square multiplier is comparatively less as compared to other multiplier architectures i.e., the number of devices used in Vedic square multiplier are 259 while Booth and Array Multiplier, are 592 and 495 respectively; for 16 x 16 bit number when implemented on Spartan FPGA [12].

Thus, the analysis shows that the algorithm used for Vedic square multiplier is consuming less area and the high speed of the reviewed architectures. The Vedic square and cube architecture proved to exhibit improved efficiency in terms of speed and area compared to Booth and Array Multiplier. Due to its parallel and regular structure, this architecture can be easily be realized

on silicon and can work at high speed without increasing the clock frequency. It has the advantage that as the number of bits increases, the gate delay and area increase very slowly as compared to the square and cube architectures of other multiplier architecture. Speed improvements are gained by parallelizing the generation of partial products with their concurrent summations. It is demonstrated that this design is quite efficient in terms of silicon area/speed. Such a design should enable substantial savings of resources in the FPGA when used for image/video processing applications.

CHAPTER-3
DESIGN AND SOFTWARE SIMULATION

The implementation technique and designing tool used for the designing of the Vedic Multiplier is based on a novel technique of digital multiplication which is quite different from the conventional method of multiplication like add and shift, where smaller blocks are used to design the bigger one. The Vedic Multiplier is designed in Verilog HDL, as its give effective utilization of structural method of modelling. The individual block is implemented using Verilog hardware description language. The functionality of each block is verified using simulation software, ModelSim and ISE.

3.1 DESIGN OF 64X64 BITS VEDIC MULTIPLIER

This block represents the implementation of 64x64 bit Vedic Multiplier.

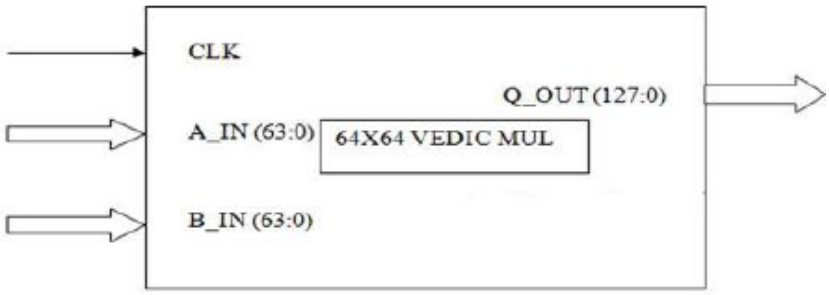


Figure 3.1: Block diagram of 64x64 Vedic Multiplier

There are four ports namely, data input (A_IN), data input (B_IN), clock (CLK), data output (Q_OUT); all signals are active high.

The representation has four ports:

1) A_IN (63:0): It is the first input of 64 bit to the Vedic Multiplier.

2) B_IN (63:0): It is the second input of 64 bit of the Vedic Multiplier.

3) CLK: It is clock input.

4) Q_OUT (127:0): It is the output register of 128 bit Vedic Multiplier.

This code was designed using synchronous resets, use in FPGAs. Both numerical accuracy and performance of the Vedic Multiplier versions of this code have been verified in a Xilinx Spartan 3E XC3s500 / 3E XC3s1600 at 50 MHZ and also by ModelSim6.1e.

3.2 IMPLEMENTATION OF VEDIC MULTIPLIER OF 2X2 BITS

It is clear that this basic building blocks of multiplier consist of one bit multipliers and one bit adders. One bit multiplication can be performed through two input AND gate and for addition, full adder can be utilized. The 2 x 2 bit multiplier is shown in figure 3.2.

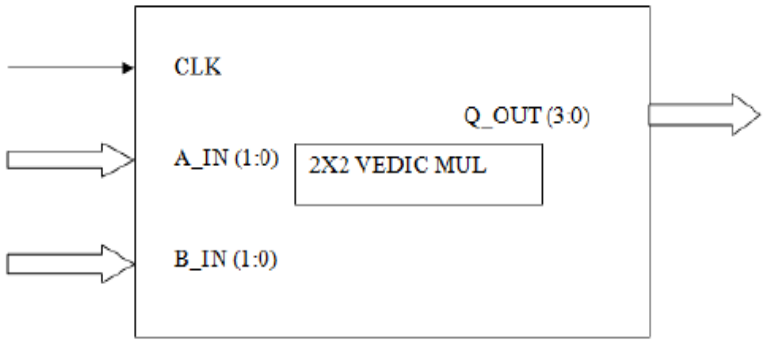


Figure 3.2: Block diagram of 2x2 Multiplier.

Let's take two inputs, each of 2 bits; say A1A0 and B1B0 and the output is of four bits, say Q3Q2Q1Q0. As per basic method of multiplication, result is obtained after getting partial product of each bits of multiplicands and doing addition simultaneously of each partial product.

```
                A1   A0
              X B1   B0
            ----------------
                A1B0  A0B0
          A1B1  A0B1
       ---------------------------
   Q3   Q2    Q1     Q0
```

As per the Vedic mathematic algorithm, Q0 is the vertical product of bit A0 and B0, Q1 is addition of crosswise bit multiplication i.e., A1 & B0 and A0 and B1 respectively, and Q2 is again vertical product of bits A1 and B1 with the carry generated, if any, from the previous

stage of the addition during Q1. Q3 output is nothing but carry generated during Q2 while calculation. This module is known as 2x2 multiplier block [5].

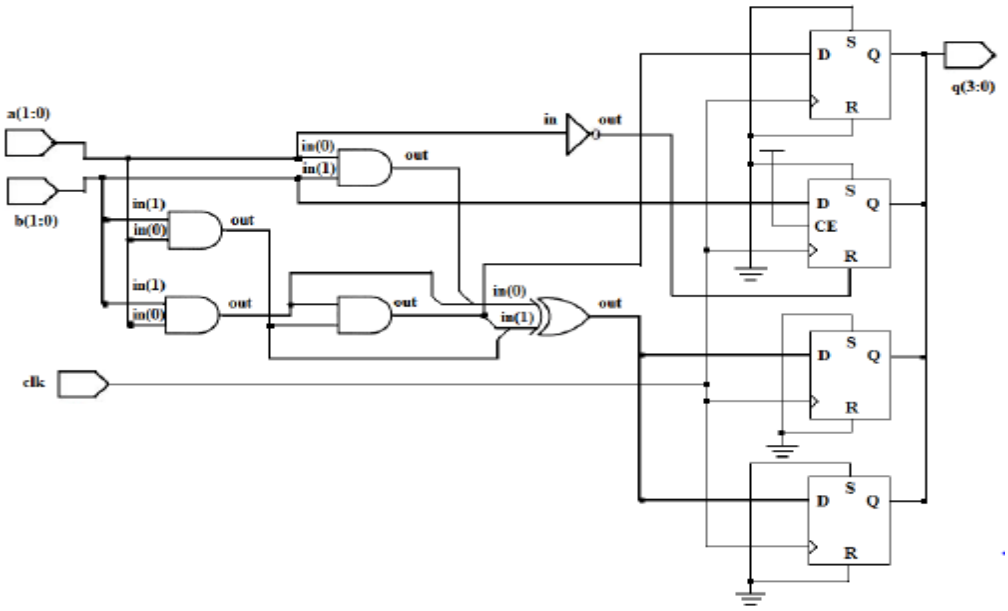


Figure 3.3: RTLView of 2x2 Bits Multiplier by ModelSim

3.3 IMPLEMENTATION OF VEDIC MULTIPLIER OF 4X4 BITS

For doing the multiplication of the higher no. of bits in the input, some modification is required.

For example:- Divide the no. of bits apply in the inputs in to two equal parts.

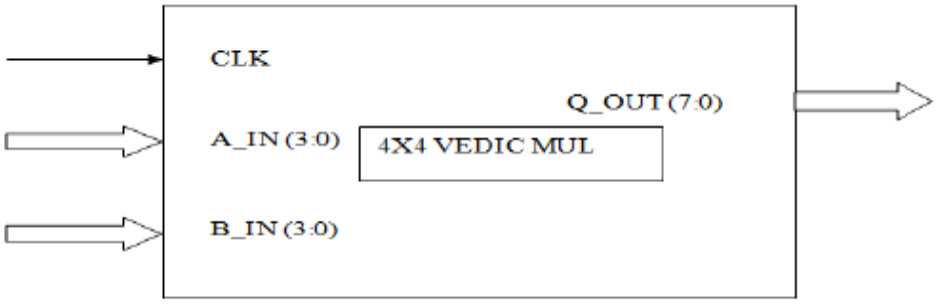


Figure 3.4: Block diagram of 4x4 Bit Vedic Multiplier

For analyse 4x4 multiplications, we takes A3A2A1A0 and B3B2B1B0 and the multiplication result of the given four bits number be Q7Q6Q5Q4Q3Q2Q1Q0. Block diagram of 4x4 Vedic Multiplier is shown in fig 3.4.

Let four bits number is divided into two small parts says A and B, A3 A2 & A1 A0 for A and B3B2 & B1B0 for B. Using the fundamentals of Vedic multiplication algorithms, we takes two bits at a time by using 2 bit multiplier block

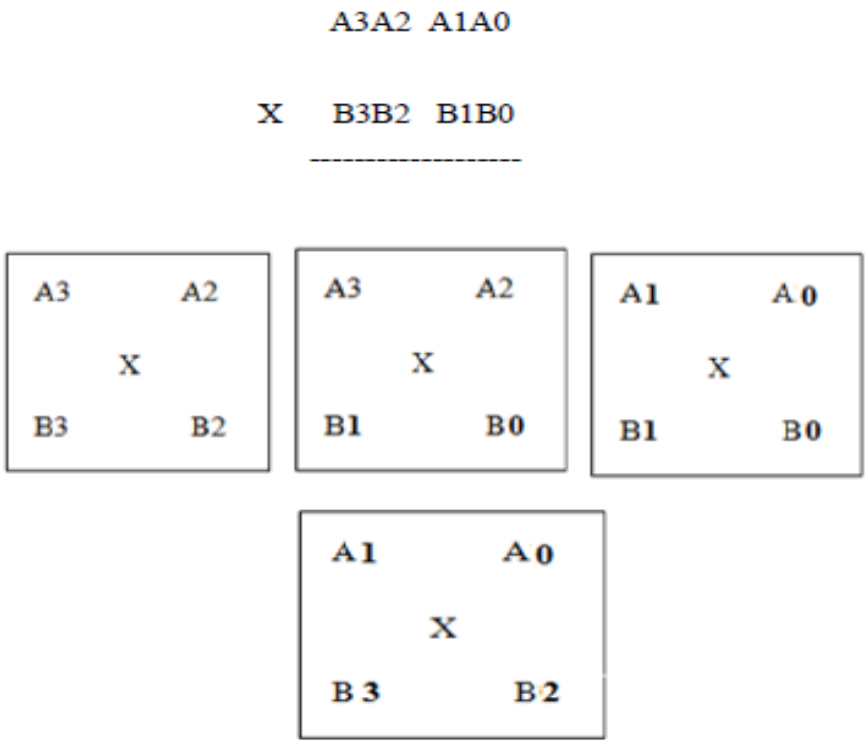


Figure 3.5: Algorithm of 4x4 bit Vedic Multiplier [5].

Each block of 2x2 bits multiplier is shown above. The inputs of the first 2x2 multiplier are A1 A0 and B1 B0 and for the last block of 2x2 multiplier is fed with inputs A3 A2 and B3 B2. The middle one shows two, 2x2 bits multiplier with inputs A3A2 & B1B0 and A1A0 & B3B2. So, the final result of multiplication, which is of 8 bit is results into Q7Q6Q5Q4Q3Q2Q1Q0 [5].

The 4x 4 bit multiplier is structured using 2X2 bit blocks as shown in figure 3.6.

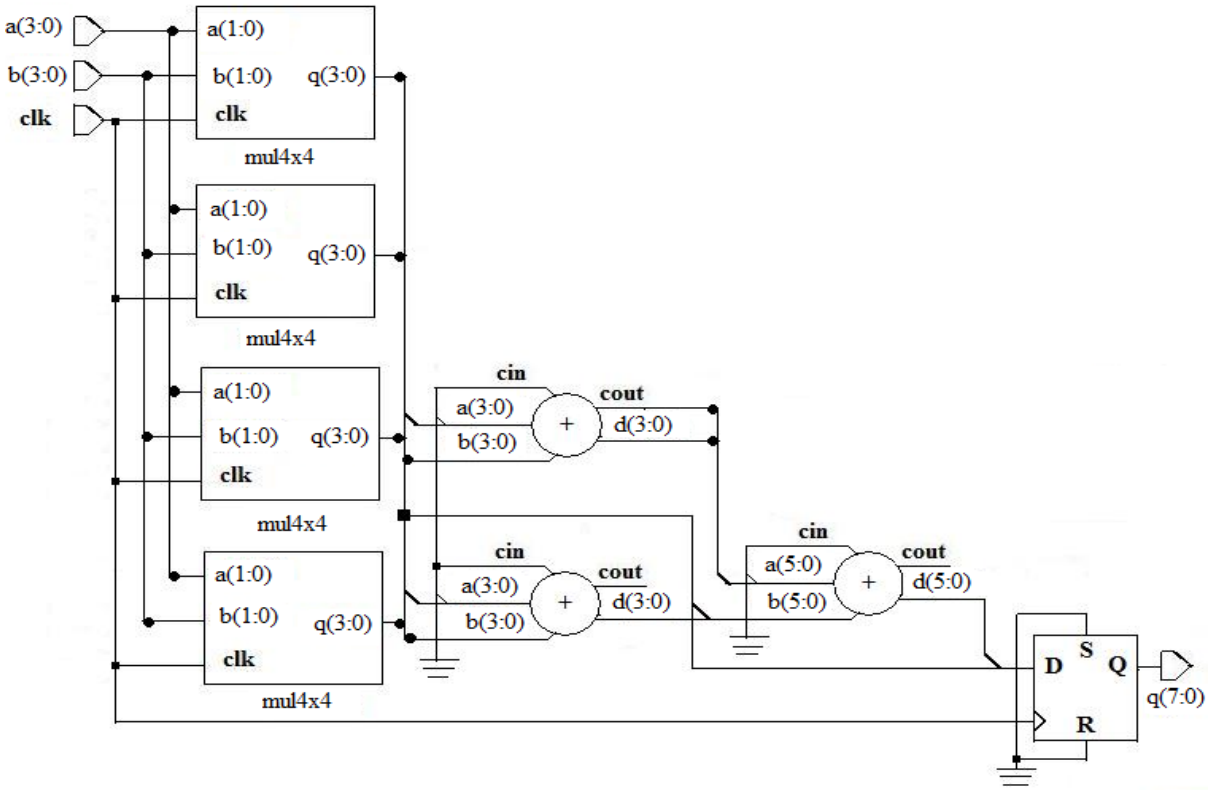


Figure 3.6: RTL View of 4x4 Bit Vedic Multiplier by ModelSim

3.4 IMPLEMENTATION OF VEDIC MULTIPLIER OF 8X8 BITS

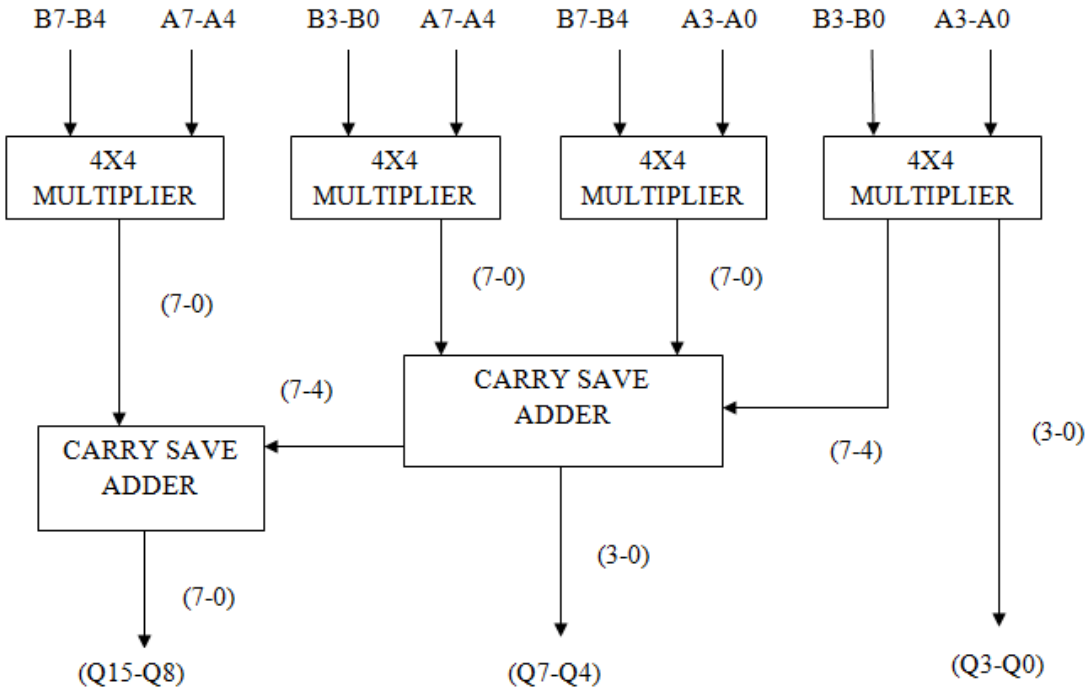
The implementation of 8x8 bit Vedic multiplier is structured using 4X4 bit multiplier blocks as shown in figure 3.7. As shown in this figure, the 8 bit multiplicand let say A can be decomposed into pair of 4 bits let it be AH-AL. Similarly, multiplicand say B can be decomposed into BH-BL. The resultant 16 bit product can be written as:

P= A x B= (AH-AL) x (BH-BL)

= AH x BH+AH x BL + AL x BH+ AL x BL

The outputs of each partial 4X4 bit multipliers are added accordingly to obtain the final product of 8x8 bit multiplier. Thus, in the final stage, two adders are also required for the computation process [12]. Now the basic building block of 8x8 bits Vedic multiplier is decomposed into

4x4 bits multiplier which can be implemented in its structural model. For bigger multiplier implementation like 8x8 bits multiplier, the 4x4 bits multiplier units have been used as components which are already implemented in ModelSim6.1e or Xilinx ISE9.2i library.

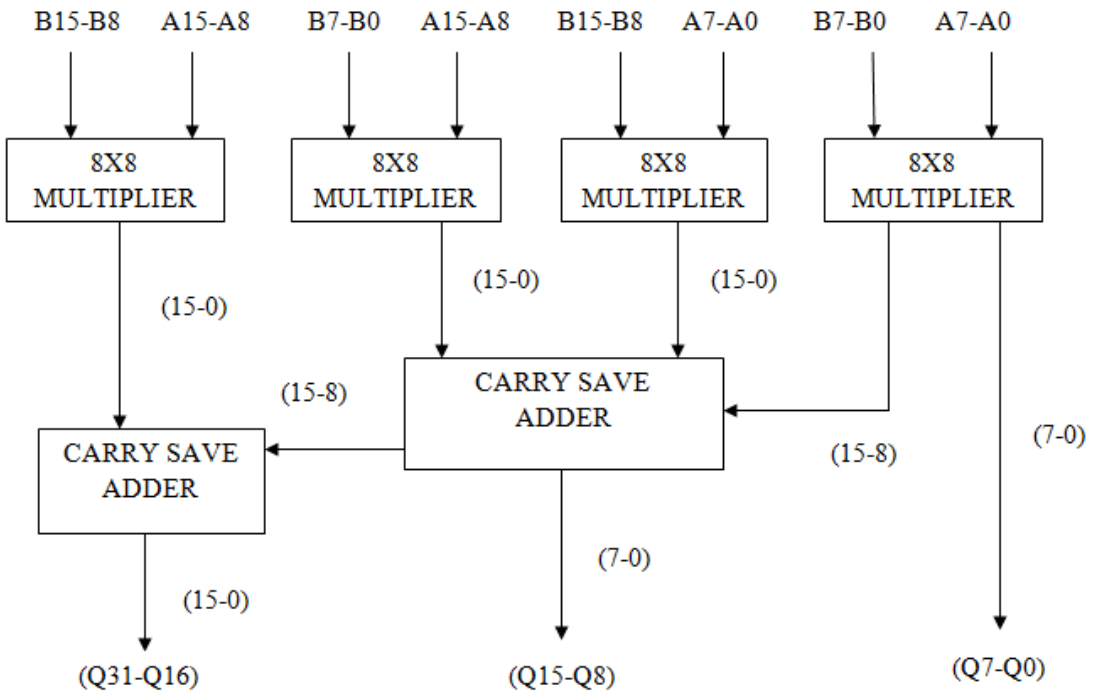


RESULT = (Q15- Q8) & (Q7- Q4) & (Q3-Q0)

Figure 3.7: 8X8 Bits decomposed Vedic Multiplier [12].

3.5 IMPLEMENTATION OF VEDIC MULTIPLIER OF 16X 16 BITS

The implementation of 16X16 bit multiplier structured using partial 8X8 bits blocks is shown in Figure 3.8. In this Figure 3.8 the 16 bit multiplicand A can be decomposed into pair of 8 bits AH-AL as done in case of 8x8 bit multiplier. Similarly multiplicand B can be decomposed into BH-BL. The outputs of 8X8 bit multipliers are added accordingly to obtain the 32 bits final product. Thus, in the final stage for doing the summation two adders are required [12]. Similarly, we have extended in a similar manner for input bits 32, 64.



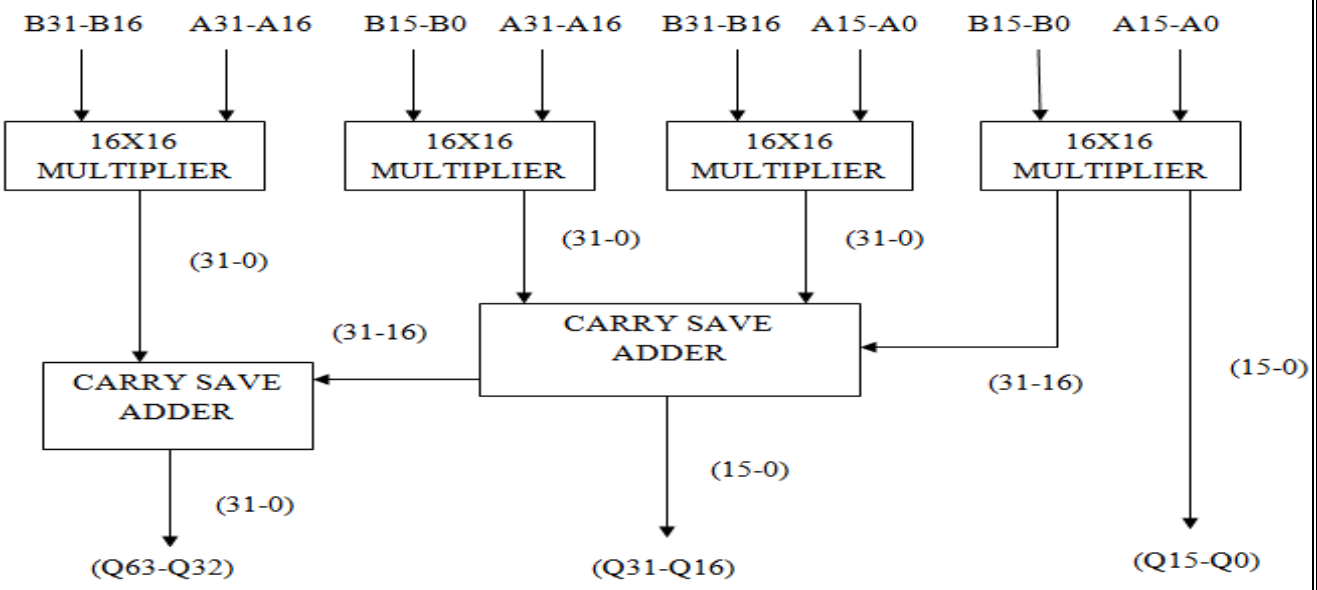
RESULT = (Q31- Q16) & (Q15- Q8) & (Q7- Q0)

Figure 3.8: Block diagram of 16x16 Bits decomposed Vedic Multiplier [12].

3.6 IMPLEMENTATION OF VEDIC MULTIPLIER OF 32X32 BITS

In this the 32 bits multiplicand say A is decomposed into pairs of 16 bits AH-AL. Similarly, 32 bits multiplicand B can be decomposed into BH-BL. Block diagram of the architecture of 32x32 bits Vedic Multiplier is shown in figure 3.9.

The outputs of 16X16 bit multipliers are added accordingly to obtain the 64 bits final product. Thus, in the final stage, two adders are required for computation.



RESULT = (Q63-Q32) & (Q31-Q16) & (Q15-Q0)

Figure 3.9: 32X32 bits proposed Vedic Multiplier.

The 32X32 bit multiplier is structured using small 16X16 bit blocks. The block diagram of 32 x32 bit multiplier is shown below.

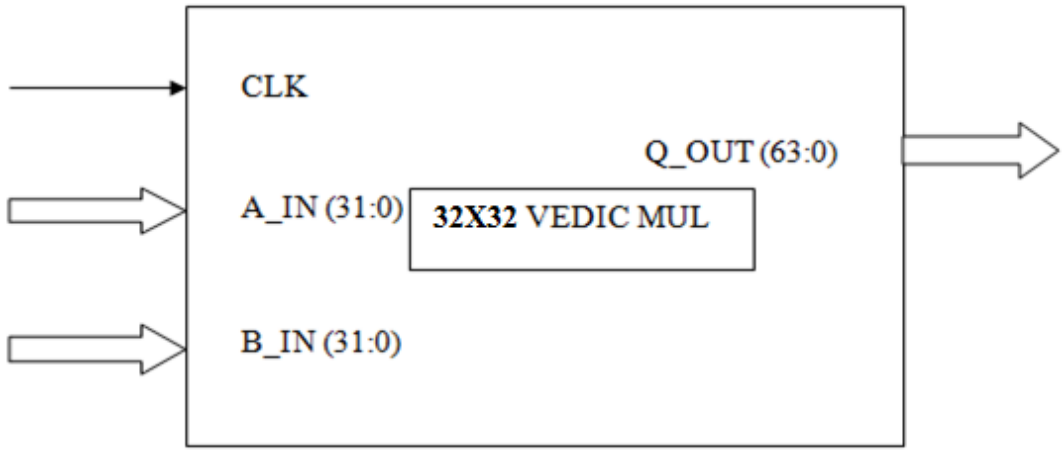


Figure 3.10: Block diagram of 32X32 Bit Vedic Multiplier

The implemented RTL view of 32x32 bits Vedic Multiplier by using 16x16 blocks with the help of ModelSim6.1e Tool is given below:

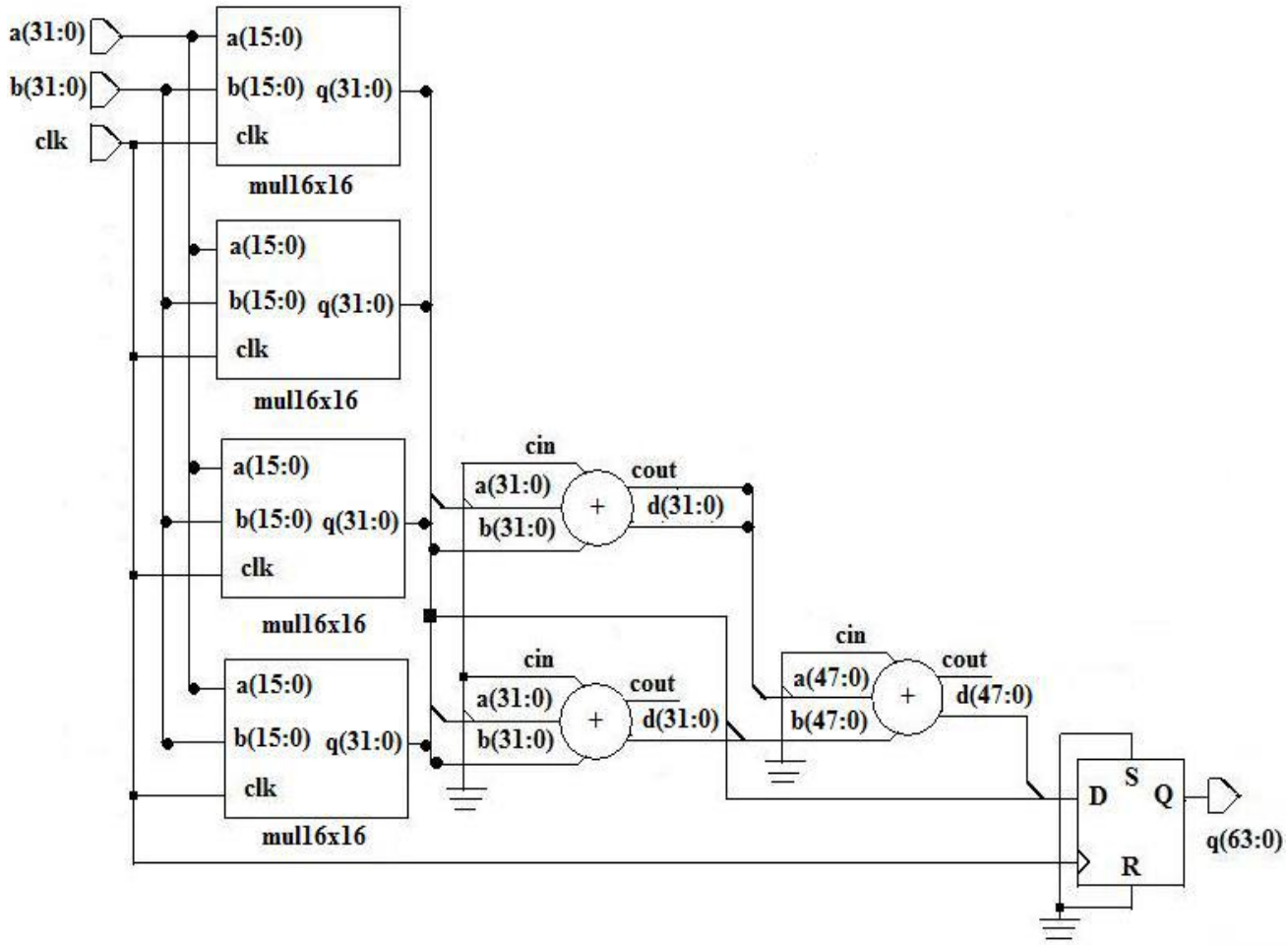


Figure 3.11: RTL View of 32X32 bits Vedic Multiplier

CHAPTER-4
REVERSIBLE LOGIC GATES

Energy loss is one of the important part of consideration in designing the digital circuit. The technological non ideality of switches and materials is one of the part which cause the energy loss in the system. And it is seems that the other cause of the problem arises from Landauer's principle for which there is no solution. Landauer's Principle [3] states that logical computations that are irreversible necessarily generate k*T*ln (2) joules of heat energy, where k is the Boltzmann's Constant (k=1.38xlO-23 J/K), T is the absolute temperature at which the computation is performed. The heat generated in this although be very small and can be negligible , But as we know Moore's Law predicts the loss of information with the exponential growth of heat generated, which will be a very much noticeable amount of heat loss in next decade. Also, the second law of thermodynamics states that, any process that is reversible in nature will not change its entropy. If we see from the thermo dynamical grounds, the erasure of one bit of information from the mechanical degrees of a system must be accompanied by the thermalization of an amount of k*T*ln (2) joules of energy. The information entropy H can be calculated for any probability distribution. Similarly the thermodynamic entropy S refers to thermodynamic probabilities specifically.

Where the gain in entropy always represent nothing more but the loss of information. In simple language we can say the design that is free from the information loss is called reversible. It naturally takes care of heat generated due to information loss. Bennett [4] states that by replacing the network with reversible logic gates zero energy dissipation would be possible. Thus, in future circuit design technologies reversibility enters and become the essential property.

In [6], the multiplier is designed using two units; one is the partial product generation unit constructed using Fredkin gates and other is the summing unit constructed using 4x4 TSG gates. [8] presented a fault tolerant reversible 4x4 multiplier circuit. For construction of this

circuit parity, preserving FRG and MIG gates were used. Multiplier circuit was designed in two parts. In second part of circuit instead of using half adders and full adders MIG gates were used. [7] has proposed a design of reversible multiplier which makes use of Peres gate for generation of partial products as compared to [10], which uses Fredkin gates. For the construction of adders the HNG gate was devised. [15] proposes low quantum cost realization of reversible multipliers which mainly uses Peres full adder gate (PF AG) for its design. It also uses Peres gates for the generation of partial products.

4.1 REVERSIBLE LOGIC

Reversible logic is refers to a promising computing design paradigm which presents a method for constructing computers that produce no heat dissipation. Reversible computing emerged as a result of the application of quantum mechanics principles towards the development of a universal computing machine that produce no heat dissipation. Specifically, the fundamentals of reversible computing are based on the relationship between entropy, heat transfer between molecules in a system, the probability of a quantum particle occupying a particular state at any given time, and the quantum electrodynamics between electrons when they are in close proximity. The basic principle of reversible computing is to obtained the device with an identical number of input and output lines which will produce a computing environment where the electrodynamics of the system allow for prediction of all future states based on known past states, and the system reaches every possible state, resulting in no heat dissipation**.**

A reversible logic gate is an N-input N-output logic device that provides one to one mapping between the input and the output. It not only helps us to determine the outputs states from the inputs states but also helps us to uniquely recover the inputs from the outputs. Garbage outputs are those which do not contribute to the reversible logic realization of the design. Quantum cost refers to the cost of the circuit in terms of the cost of a primitive gate. Gate count is the

number of reversible gates used to realize the function. Gate level refers to the number of levels which are required to realize the given logic functions.

The following are the important design constraints for reversible logic circuits:

I. Reversible logic gates do not allow fan-outs.

2. Reversible logic circuits should have minimum quantum cost.

3. The design can be optimized so as to produce minimum number of garbage outputs.

4. The reversible logic circuits must use minimum number of constant inputs.

5. The reversible logic circuits must use a minimum logic depth or gate levels.

The basic reversible logic gates encountered during the design are listed below:

I. Feynman Gate [5]:

It is a 2x2 reversible logic gate. It is also known as Controlled Not (CNOT) Gate. It has quantum cost one and is generally used for Fan Out purposes.

2. Peres Gate [17]:

It is a 3x3 reversible logic gate. It has quantum cost four. It is used to realize various Boolean functions such as AND, XOR.

3. Fredkin Gate [16]:

It is a 3x3 reversible logic gate. It has quantum cost five. It can be used to implement a Multiplexer.

4. HNG Gate[7]:

It is a 4x4 reversible logic gate. It has quantum cost six. It is used for designing ripple carry adders. It can produce both sum and carry in a single gate thus minimizing the garbage and gate counts.

5. NFT Gate[5]

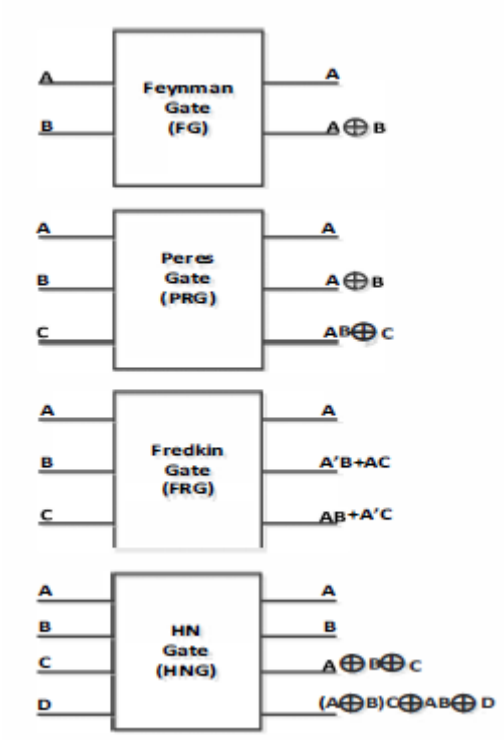
The 3*3 New Fault Tolerant gate (NFT) with quantum cost of 5 and has worst case delay of 3 it has better correction capability. The output states map to the inputs in this manner.

A = A XOR B, B = AC' XOR B'C, and

C = AC' XORB C,

6. F2G Gate[5]

The 3*3 Feynman double gate gate with quantum cost of 2 has worst case delay of 3 it has better correction capability.



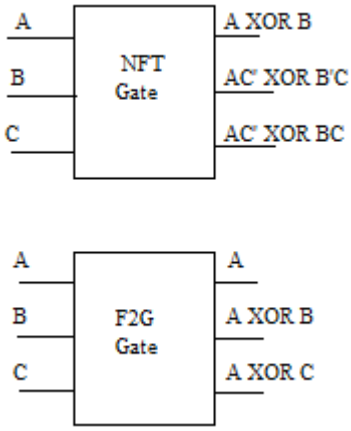


Figure4.1: Reversible Logic Gates[1][2][3][4][5]

CHAPTER-5
CHINESE ABACUS ADDER

5.1 INTRODUCTION

The Chinese abacus is one of the very popular and efficient technique used to perform various arithmetic operations. It is one of the mostly used techniques in China for doing the mathematical calculation in schools, colleges and other small commercial enterprises. Since last centuries, it is adopted in many parts of the world. The speed of use is the main feature of the Chinese abacus which make it competitor with other techniques. A well-trained operator is often capable of competing with electronic pocket calculators. The time required for inputting data manually is comparable to the electronic approach, and the generation of the result in the Chinese abacus is so straightforward that the total computation time is extremely less.

5.2 OPERATION PRINCIPLE

The design of Chinese abacus is consists of a set of unity elements, each elements representing the various decade of decimal number. Each element is made up of unity weight five beads and two beads having a weight of 5. The representation of this shown in Fig.5. 1(a) represents the number seven. The coding rule is used in this is thermometric; thus, in order to represent a number below five, then in the main part of units the same number of beads will be raised. For numbers above five, one bead with having the weight 5 will be lowered. In such a way, a decimal number comprised in the range from 0 to 15 can be represented using a basic element. The the use of two beads with weight 5 is one of the key feature of Chinese abacus adder. This allows the operator to minimize the transmission of rests. A fast implementation of elementary arithmetic functions such as addition and subtraction can be performed using the thermometric code.

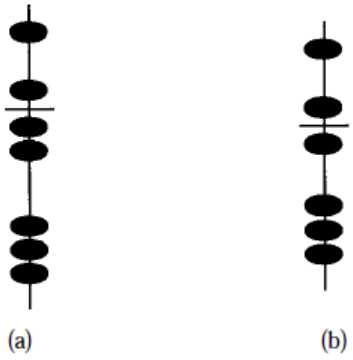


Fig. 5.1. Chinese-abacus coding represents (a) a decimal number and (b) an octal number.

In this work, each basic column element of this abacus adder has three lower beads with a weight of one and three higher beads with a weight of four. The basic element is able to represent decimal numbers in the range from 0 to 15 as depicted in figure 5.1(a). In this methodology, the 4-bit adder only contributes two carry ripple transmissions. One is internal carry from lower beads to higher beads; the other is external carry from this column element to next column element.

The function of this abacus adder is divided into three parts as depicted in figure 5.2. The first part is B/A (binary to abacus) module. The second part is P/A (parallel addition) module. The third part is T/B (Thermometric to Binary) transformation module. The three modules are discussed in the following sections.

As shown in figure 5.2 the function of radix-4 abacus adder is completed in four phases. In first phase, binary input is converted into abacus output known as B/A(binary to abacus) module. In second phase, parallel additon takes place know as P/A(paralle addition module). In the last third phase, thermometric input is converted into binary output know as T/B(thermometric to binary) transformation module.

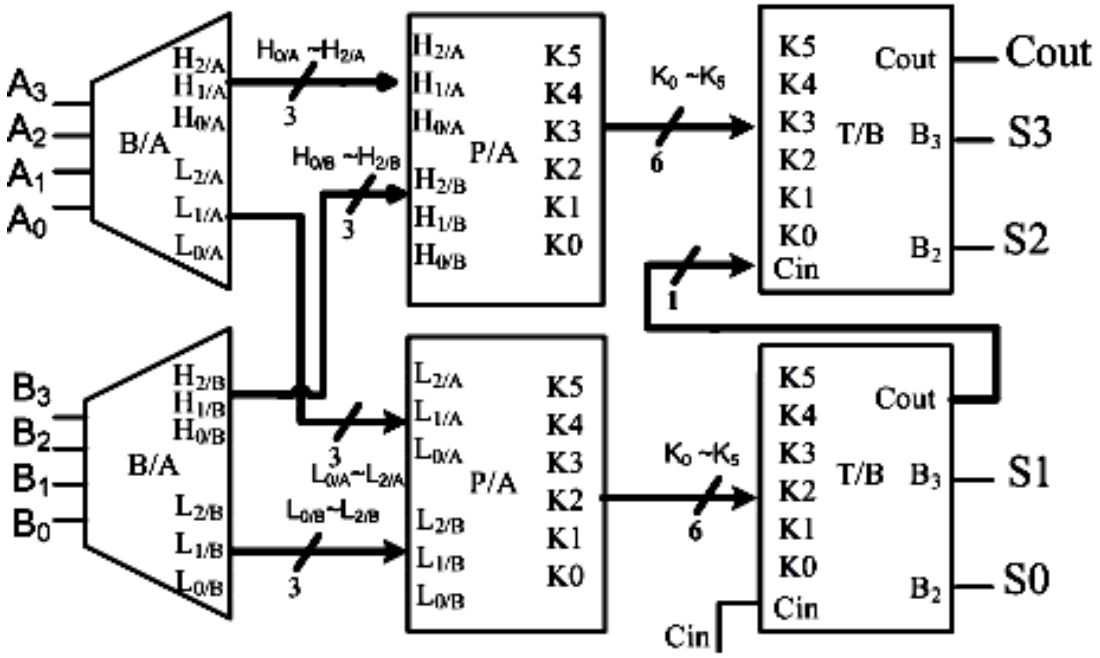


Figure5.2: Block diagram of radix-4 abacus adder.

5.2.1 B/A module:

This module converts a 4-bit binary number $(b3b2b1b0)_2$ into an abacus representation $(H2H1H0|L2L1L0)_{abacus}$. (H2H1H0) represent three higher beads each having the weight of four. Similarly, (L2L1L0) represent three lower beads each having unity weight. (H2H1H0) and (L2L1L0) are calculated by the following equations:

$$H_2 = b_3b_2,\ H_1 = b_3,\ H_0 = b_3 + b_2,$$
$$L_2 = b_1b_0,\ L_1 = b_1,\ L_0 = b_1 + b_0,$$

where $0 < H2 < H1 < H0 < 1$ and $0 < L2 < L1 < L0 < 1$.

For example, $(0111)_2 = (001|111)_{abacus} = (0+0+1)*4 + (1+1+1)*1 = 7$.

5.2.2 P/A module:

This block is used to count the lower and higher beads (L2L1L0), (H2H1H0) of the abacus numbers, respectively. The sum of (H2AH1AH0A) and (H2BH1BH0B), the upper part P/A

shown in figure 2, will then be represented as the thermometric transformation (K5K4K3K2K1K0), where 0 < Ki < Kj < 1 for i > j.

The function of P/A module is modelled in the following equations:

$$f_1 = \overline{H}_{2A} H_{1A}, \quad (3)$$

$$f_2 = \overline{H}_{1A} H_{0A}, \quad (4)$$

$$K_0 = (\overline{H}_{0A})(H_{0B}) + (f_2)(1) + (f_1)(1) + (H_{2A})\,(1), \quad (5)$$

$$K_1 = (\overline{H}_{0A})(H_{1B}) + (f_2)(H_{0B}) + (f_1)(1) + (H_{2A})\,(1), \quad (6)$$

$$K_2 = (\overline{H}_{0A})(H_{2B}) + (f_2)(H_{1B}) + (f_1)(H_{0B}) + (H_{2A})\,(1), \quad (7)$$

$$K_3 = (\overline{H}_{0A})(0) + (f_2)(H_{2B}) + (f_1)(H_{1B}) + (H_{2A})\,(H_{0B}), \quad (8)$$

$$K_4 = (\overline{H}_{0A})(0) + (f_2)(0) + (f_1)(H_{2B}) + (H_{2A})\,(H_{1B}), \quad (9)$$

$$K_5 = (\overline{H'}_{0A})(0) + (f_2)(0) + (f_1)(0) + (H_{2A})\,(H_{2B}). \quad (10)$$

This module has the functionality same as that of multiplexer. The addend A is used as a signal selector to modify the configuration of augend B resulting in the thermometric sum as shown in figure 3. The addend A consist of four order each for higher beads (H2AH1AH0A) or lower beads (L2AL1AL0A), i.e., 000, 001, 011, and 111. The sum of (L2AL1AL0A) and (L2BL1BL0B), the lower part P/A shown in figure 5.2, it is also representing the thermometric transformation as per the following equations:

$$f_1 = \overline{L}_{2A} L_{1A}, \quad (11)$$

$$f_2 = \overline{L}_{1A} L_{0A}, \quad (12)$$

$$K_0 = (\overline{L}_{0A})(L_{0B}) + (f_2)(1) + (f_1)(1) + (L_{2A})\,(1), \quad (13)$$

$$K_1 = (\overline{L}_{0A})(L_{1B}) + (f_2)(L_{0B}) + (f_1)(1) + (L_{2A})\,(1), \quad (14)$$

$$K_2 = (\overline{L}_{0A})(L_{2B}) + (f_2)(L_{1B}) + (f_1)(L_{0B}) + (L_{2A})\,(1), \quad (15)$$

$$K_3 = (\overline{L}_{0A})(0) + (f_2)(L_{2B}) + (f_1)(L_{1B}) + (L_{2A})\,(L_{0B}), \quad (16)$$

$$K_4 = (\overline{L}_{0A})(0) + (f_2)(0) + (f_1)(L_{2B}) + (L_{2A})\,(L_{1B}), \quad (17)$$

$$K_5 = (\overline{L}_{0A})(0) + (f_2)(0) + (f_1)(0) + (L_{2A})\,(L_{2B}). \quad (18)$$

For example, higher beads from addend A (H2AH1AH0A) =111 and higher beads from augend B (H2BH1BH0B) = 011 will activate the segment. The thermometric sum (K5K4K3K2K1K0) will be (011111). The P/A module as depicted in figure 5.2 are derived from the equations (3)–(18). This P/A module is used to count all beads simultaneously. Whereas Gang's abacus adder count each bead one after one [7].

5.2.3 T/B module:

This module is used to transforms the thermometric sum to binary numbers. Outputs S1 (or S3), S0 (or S2) and Cout are determined by the following equations

$$S_0(\text{or } S_2) = \overline{K}_0 C_{in} + \overline{K}_1 K_0 \overline{C}_{in} + \overline{K}_2 K_1 C_{in} + \overline{K}_3 K_2 \overline{C}_{in} + \overline{K}_4 K_3 C_{in} + \overline{K}_5 K_4 \overline{C}_{in} + K_5 C_{in}, \quad (19)$$

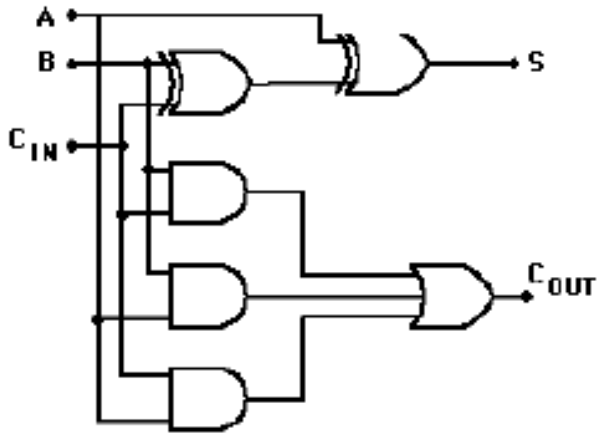
$$S_1 \text{ (or } S_3) = K_5 + K_4 C_{in} + \overline{K}_2 K_0 C_{in} + \overline{K}_3 K_1 \overline{C}_{in}, \quad (20)$$

$$C_{out} = K_3 + K_2 C_{in} . \quad (21)$$

CHAPTER-6
ADDERS

Arithmetic operations such as addition, subtraction, multiplication and division are widely used and play a very vital role in various digital systems operation such as digital signal processor (DSP) architecture, microprocessor and microcontroller and data process unit.

Adders are one of the essential logic circuits designed to perform high speed arithmetic operations and are one of the basic block in designing of any digital systems because of their extensive use in several basic operations such as subtraction, multiplication and division. In many computers and other kinds of processors, adders are used not only in the arithmetic logic unit, but also in other parts of the processing operation of processor, where they are used to calculate addresses, table indices, and other same kind of operations. The very basic arithmetic operation of the adder is the addition of two binary digits, i.e., bits. A combinational circuit that adds two bits whose the schematic diagram is shown below is called a half adder. The block diagram and truth table of the full adder is shown in Fig. 6.1. It is used to add three bits, where the third bit is generated from a previous addition operation i.e., carry coming from lower order bits after addition [1]-[6].



A	B	Cin	Cout	S
0	0	0	0	O
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figure 6.1: Block diagram and truth table of full adder

Addition is one of the fundamental arithmetic operation that is used in many VLSI design systems like DSP architecture, microprocessor, microcontroller and data process unit. For

enhancing fast computation process this VLSI system requires fast addition which impacts the overall performance of digital system. Adder is used for doing this addition operation. Adder structure is classified in various ways depending upon their execution procedure such as serial or parallel way. Many studies has been carried out in designing of high-speed, low-area, or low-power adders [7]-[10]. Some of these types of adders are Ripple Carry Adder (RCA), Carry-Look ahead Adder (CLA), Carry Save Adder (CSA), Carry Select Adder, Carry-Bypass Adder or Carry Skip Adder (CSK) etc. discussed in brief [11-22].

6.1. Ripple Carry Adder (RCA)

RCA is simply an array of full adder connected in series so that the carry must propagate through every full adder before the addition is complete. Each full adder inputs a Cin, which is the Cout of previous adder. This kind of adder is called as a ripple carry adder, since each carry bit ripples to the next full adder. RCA is always preferred in terms of power and area when it appears to be fast enough for its intended purpose. RCAs requires the least amount of hardware comparative of all other adders, but it is the slowest because of the ripple of carry in each full adder [11]-[12].

6.2. Carry Look ahead Adder (CLA)

Two come over the drawback of RCA, CLA has been designed. A fast method of adding numbers is called carry-look ahead. The computation method of this adder is fast because it does not require the carry signal to propagate stage by stage, causing a bottleneck. Instead it uses additional logic to expedite the propagation and generation of carry information, allowing fast addition at the expense of more hardware requirement [15]-[18].

6.3. Carry Save Adder (CSA)

CSA is a kind of adder with low propagation delay, it adds three input numbers to an output pair of numbers instead of adding two input numbers to a single sum output. Where its two outputs are then summed by a traditional carry-look ahead or ripple carry adder, we receive the sum of all three inputs. In particular, the propagation delay of a CSA is not affected by the width of vectors being added. Each full adder's output S is connected to corresponding output bit of one output, and it's output Cout is connected to the next higher output bit of the second output; the lowest bit of the second output is fed directly from the carry-save's Cin input [23].

6.4. Carry Select Adder

In the Carry Select Adder the n-bit adder is divided into "k" ripple-carry adders of n/k bits each and except the lowest order part; all these adder blocks are simulated. The simplest n-bit carry select adder is built using three n/2 bit ripple carry adders. The first adder is utilized to compute the lower half of the n-bit sum, while the other two compute the higher half: one based on the assumption that the input carry is zero, the other based on the assumption that it is one. This way the computation of the higher half can start immediately without the wait for the lower half to complete. When the lower half of the sum is computed and the carry input for the next stage is available, the correct half of the sum is selected by a multiplexer. Because of the simulation technique, the required area and power consumption of this adder particularly doubles with respect to RCA [23].

6.5. Carry-Bypass Adder or Carry Skip Adder (CSK)

CSK is very simple but creative adder with a minimum number of additional logic. The n-bit adder is divided into "k" ripple-carry adder blocks. Each adder block has a group propagate signal meaning that when this signal is 1, an incoming carry cannot be absorbed and will propagate through the adder block as an alternative by skipping the adder segment via the skip logic [14].

CHAPERT -7

ARCHITECHTURE OF PROPOSED WORK

7.1 ARCHITECTURE OF REVERSIBLE *URDHVA TIRYAKBHAYAM* MULTIPLIER

The 4x4 Vedic multiplier is design using *Urdhva Tiryakbhayam* sutra along with the Chinese abacus methodology for doing the addition of the partial generated products. It may simply use three 4-bit abacus adders, shown in figure 7.3. The all component of multiplier has been designed using reversible logic gates. The digital logic implementation of the 2X2 *Urdhva Tiryakbhayam* multiplier using the conventional logic gates [11] is as shown in figure 7.1. The expressions for the four output bits are given below figure 7.1. The block diagram of reversible logic gate implementation of 2x2 UT multiplier is as shown in figure 7.4. This design does not consider the fanouts. The circuit requires a total of six reversible logic gates out of which five are Peres gates and remaining one is the Feynman Gate. The quantum cost of the 2X2 *Urdhva Tiryakbhayam* Multiplier is enumerated to be 21. The number of garbage outputs is 9 and number of constant inputs is 4.

The high speed, area efficient and less power dissipated Reversible 4X4 *Urdhva Tiryakbhayam* Multiplier emanates from the 2X2 multiplier. The block diagram of the proposed high speed, area efficient and less power dissipated 4X4 Vedic Multiplier is shown in the figure 7.4. The 4x4 Vedic Multiplier consists of four 2X2 multipliers, each of which processes four bits as an inputs; two bits from the multiplicand and two bits from the corresponding multiplier. The lower two bits of the output of the first 2X2 multiplier are entrapped as the lowest two bits of the final result of multiplication. Two zeros are concatenated with the upper two bits and fed as an input to the four bit Chinese abacus adder. The other four input bits for the Chinese abacus adder are obtained from the second 2X2 multiplier. Likewise, the outputs of the third and the terminal 2X2 multipliers are given as inputs to the second four bit Chinese abacus adder. The outputs of these four bit Chinese abacus adders are in turn 5 bits each, which need to be summed

up. This is done by a five bit Chinese abacus adder which generates a six bit output. These six bits form the upper bits of the final result.

The Chinese abacus adder is consummated (realized) using the NFT and F2G Gate. Single NFT Full Adder (SNFA**)** is a Fault Tolerant full adder circuit which consists of one New Fault Tolerant (NFT) gate and three Feynman Double (F2G) gates where the quantum cost is 11 and the total number of garbage outputs is 3. This design also does not take into consideration the fanouts of the gates. For this design the quantum cost is calculated to be 162, the number of garbage outputs will be 62, the total number of gates used will be 36 and the number of constant inputs will be 29.

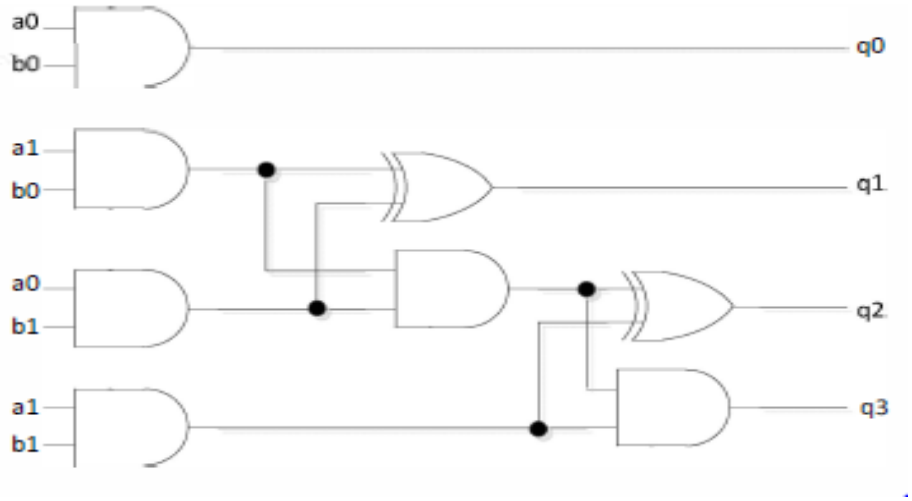


Figure 7.1: conventional logic implementation of 2x2 UT multiplier [19]

$$q0 = a0.b0,$$

$$ql = (a1.b0) \text{ xor } (a0.bl),$$

$$q2 = (a0.al.b0.bl) \text{ xor } (al.bl),$$

$$q3 = a0.al.b0.bl,$$

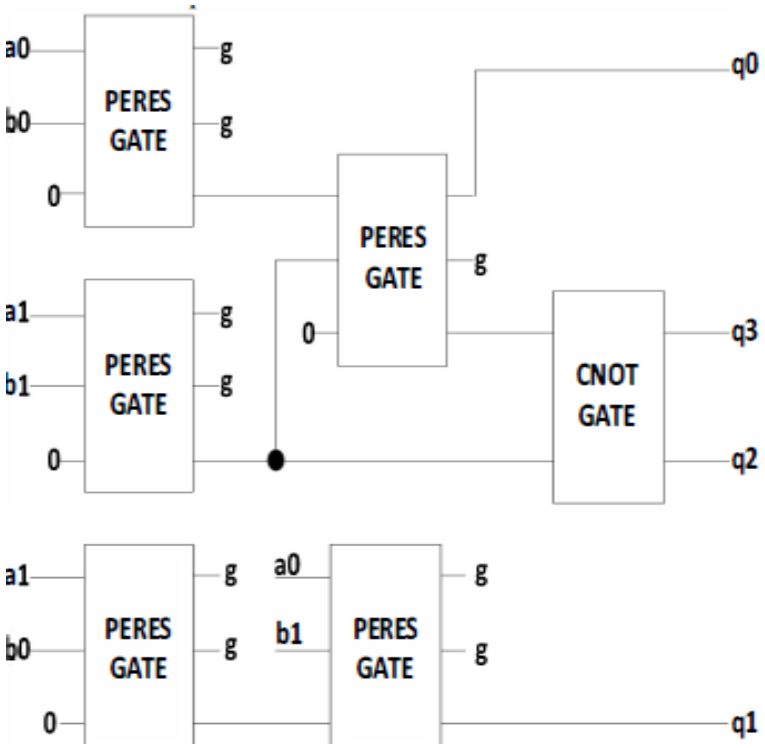


Figure 7.2: Reversible Implementation of 2x2 UT Multiplier

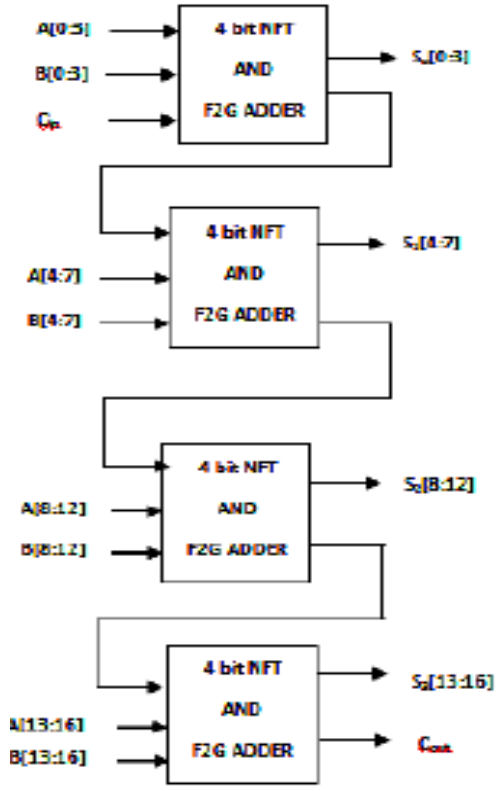


Figure 7.3: Reversible logic gate implementation of Chinese Abacus Adder. [5]

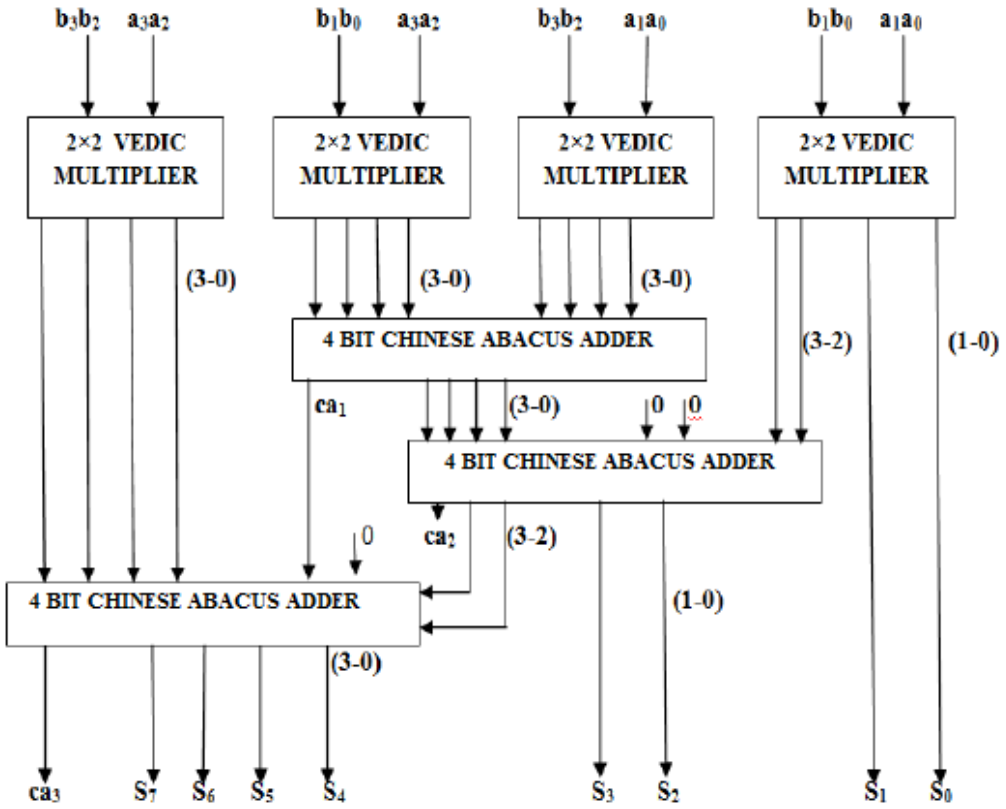


Figure 7.4: Block diagram of proposed 4x4 UT Multiplier using Chinese Abacus Adder[34].

CHAPTER-8
RESULTS AND CONCLUSION

In this work we have implemented and analyse the Vedic multiplier in two ways and compared the results. In first work, we have compared the ripple carry adder, carry look ahead adder and Chinese Abacus adder; in the second work we design the Vedic multiplier using Chinese abacus adder with and without using reversible logic gates and compares the results.

First work:

Comparison of the 4 bit ripple carry adder and Carry Look Ahead Adder with the radix-4 Chinese abacus adder is done and it is found that Chinese abacus adder is much faster than ripple carry adder and carry look ahead adder. The RTL view and simulation results of both the adders are shown below:

- Ripple carry Adder:

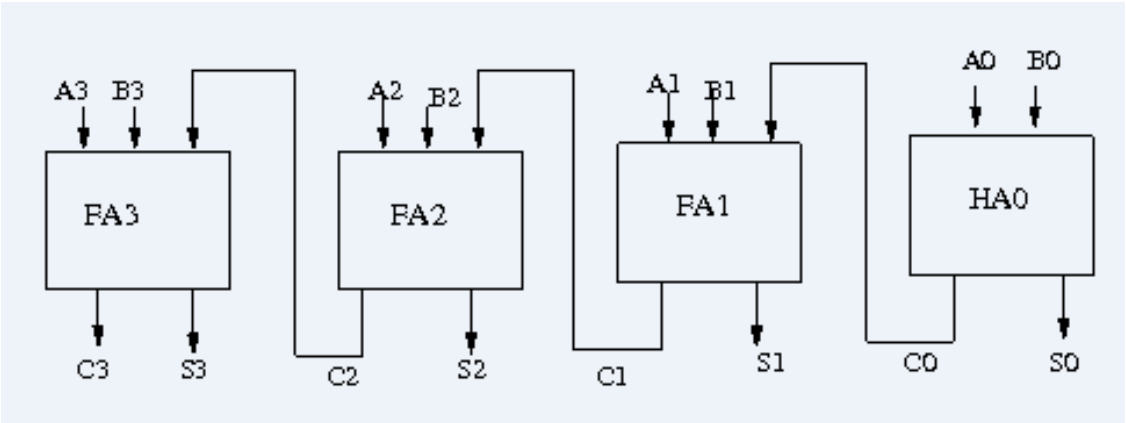


Figure 8.1: Block diagram of 4 bit ripple carry adder

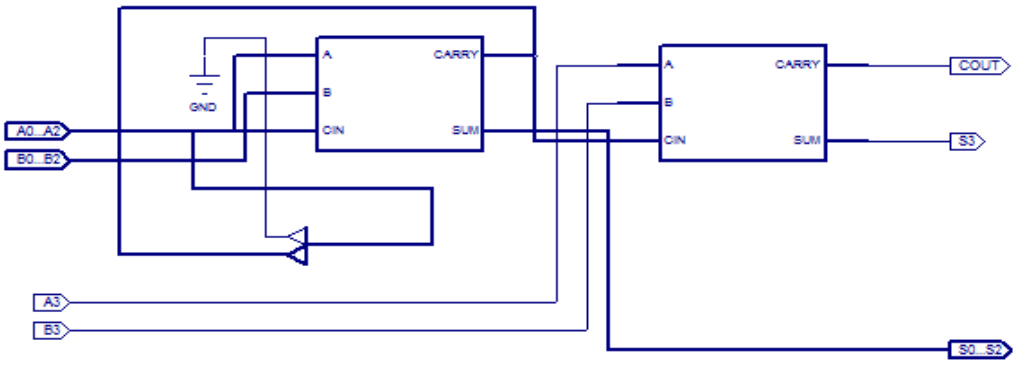


Figure 8.2: RTL view of 4 bit ripple carry adder

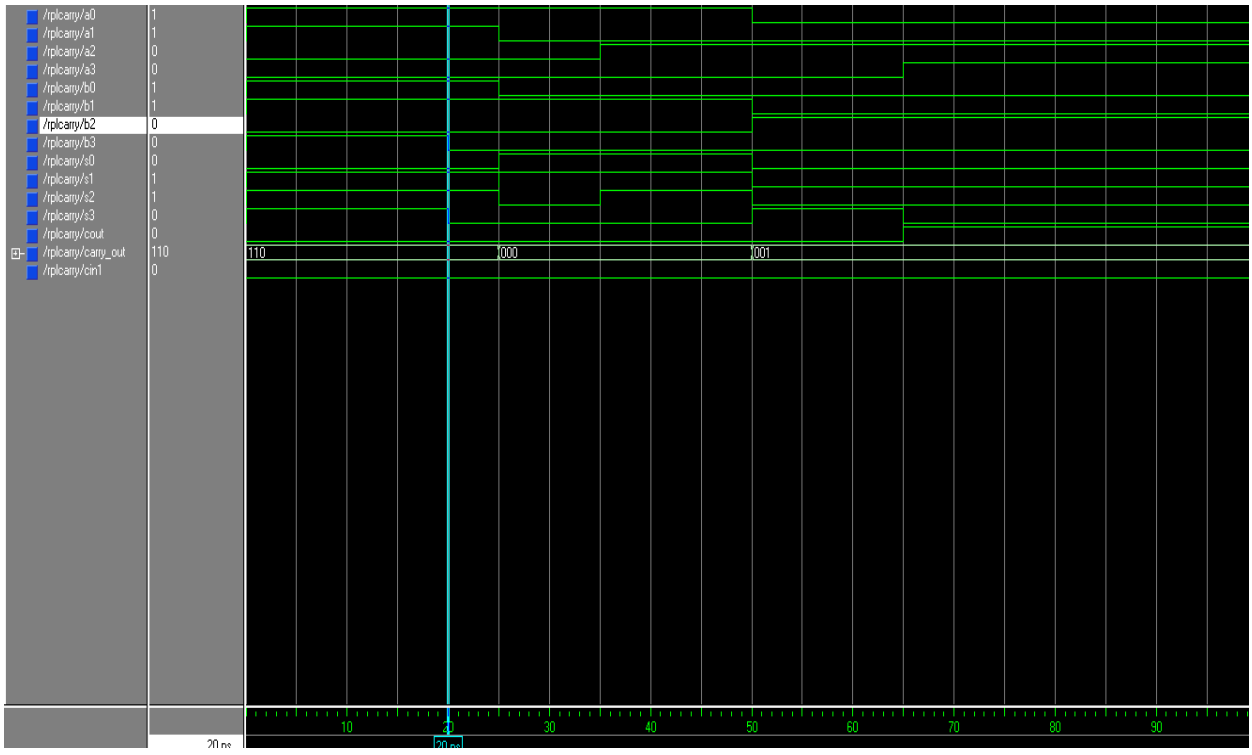


Figure 8.3: Simulation result of 4 bit ripple carry adder

- Carry Look Ahead Adder

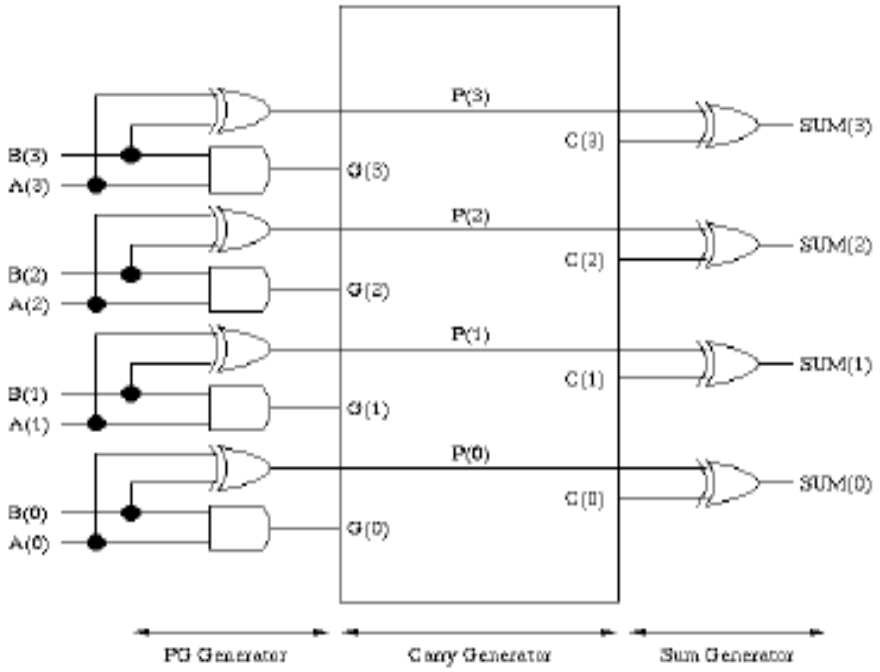


Figure 8.4: Block diagram of 4 bit Carry Look Ahead carry Adder

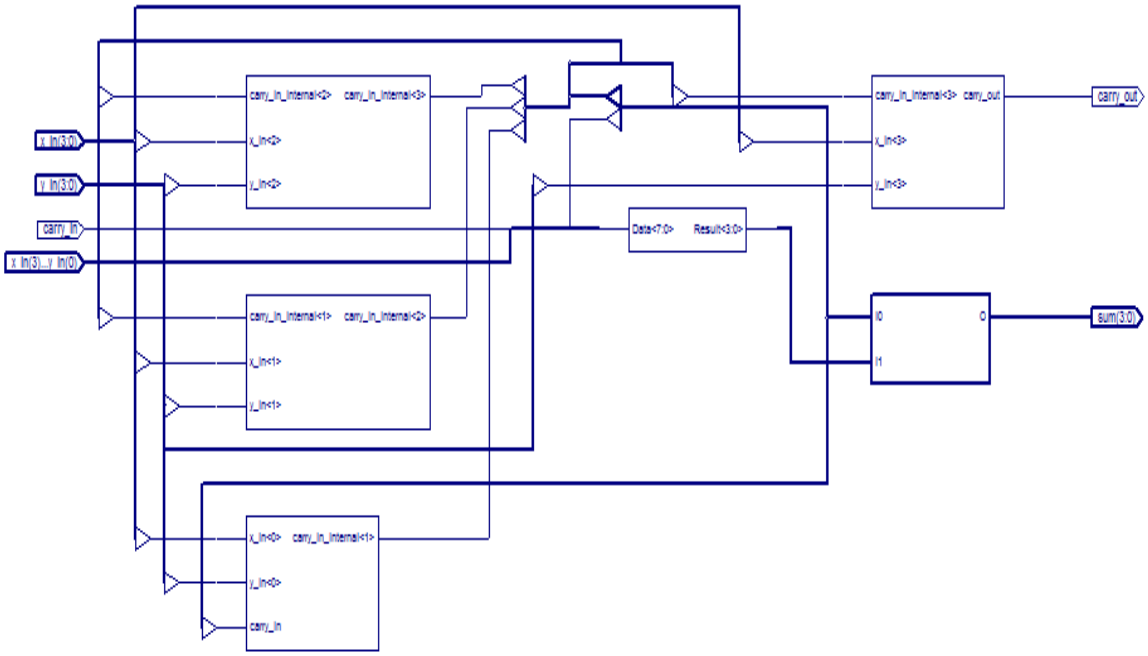


Figure 8.5: RTL view of 4 bit Carry look ahead adder

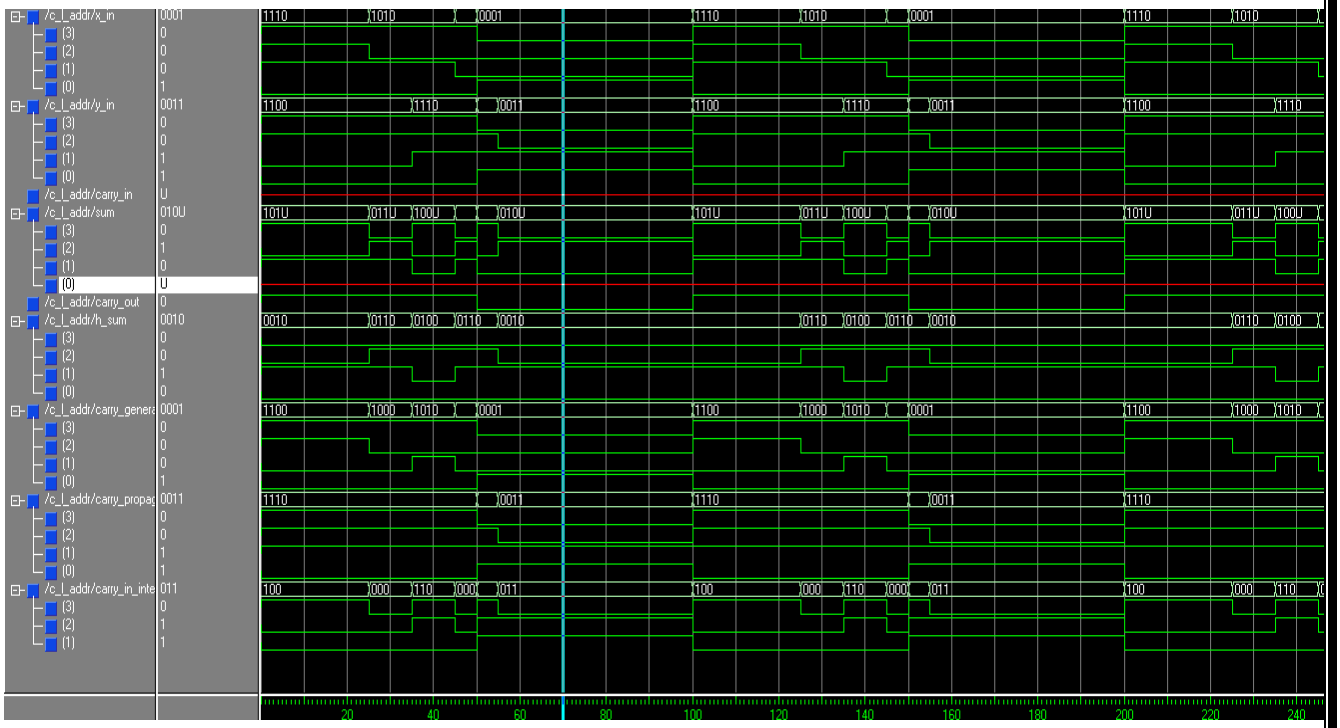


Figure 8.6: Simulation result of 4 bit carry look ahead adder.

- Chinese Abacus Adder

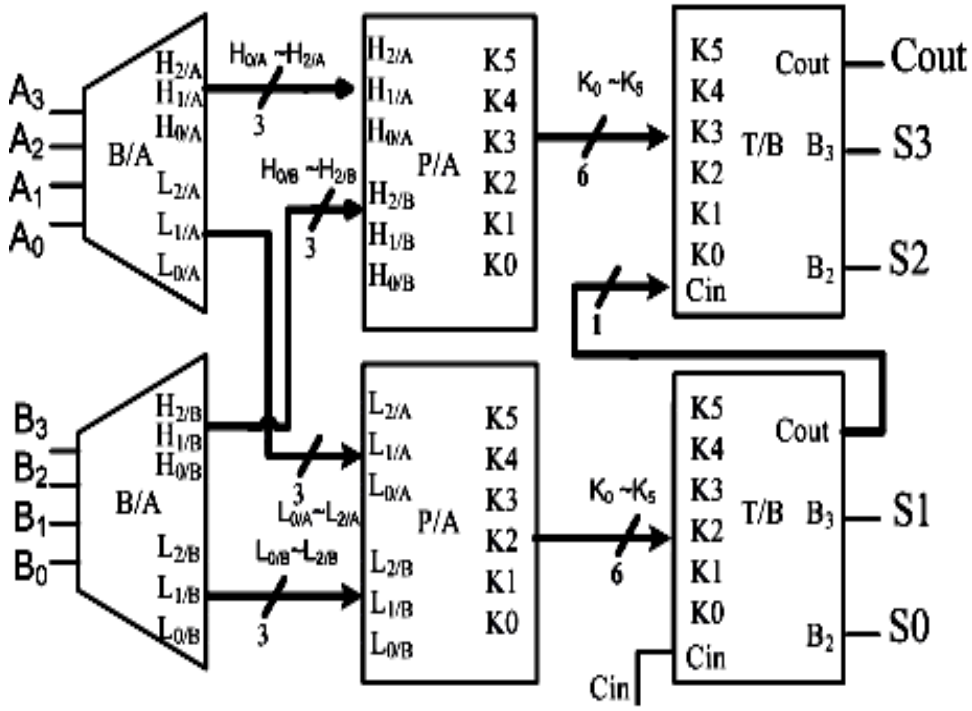


Figure8.7: Block diagram of radix-4 abacus adder.

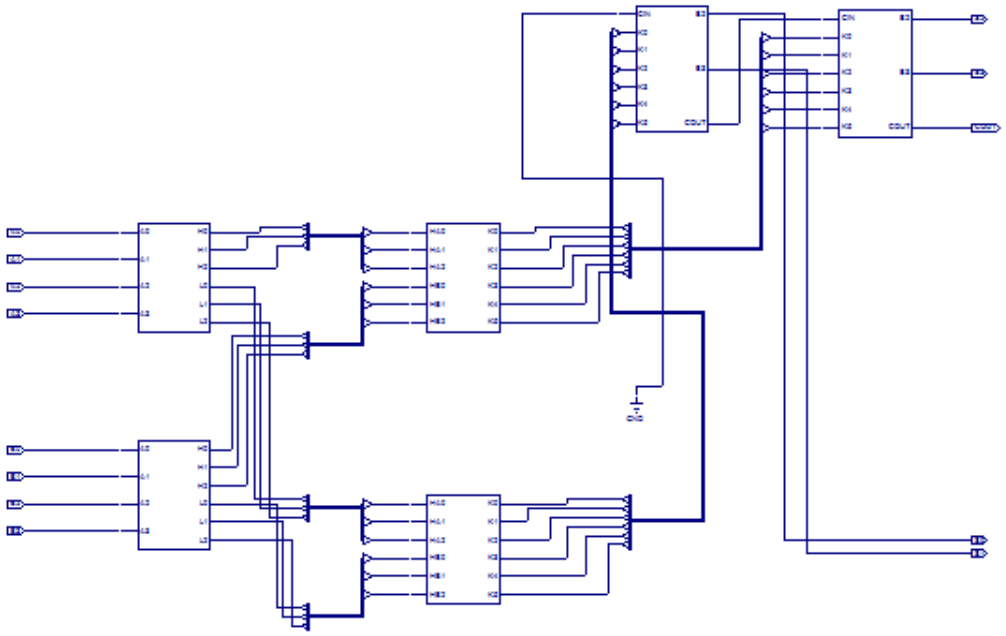


Figure 8.8: RTL view of Chinese Abacus Adder

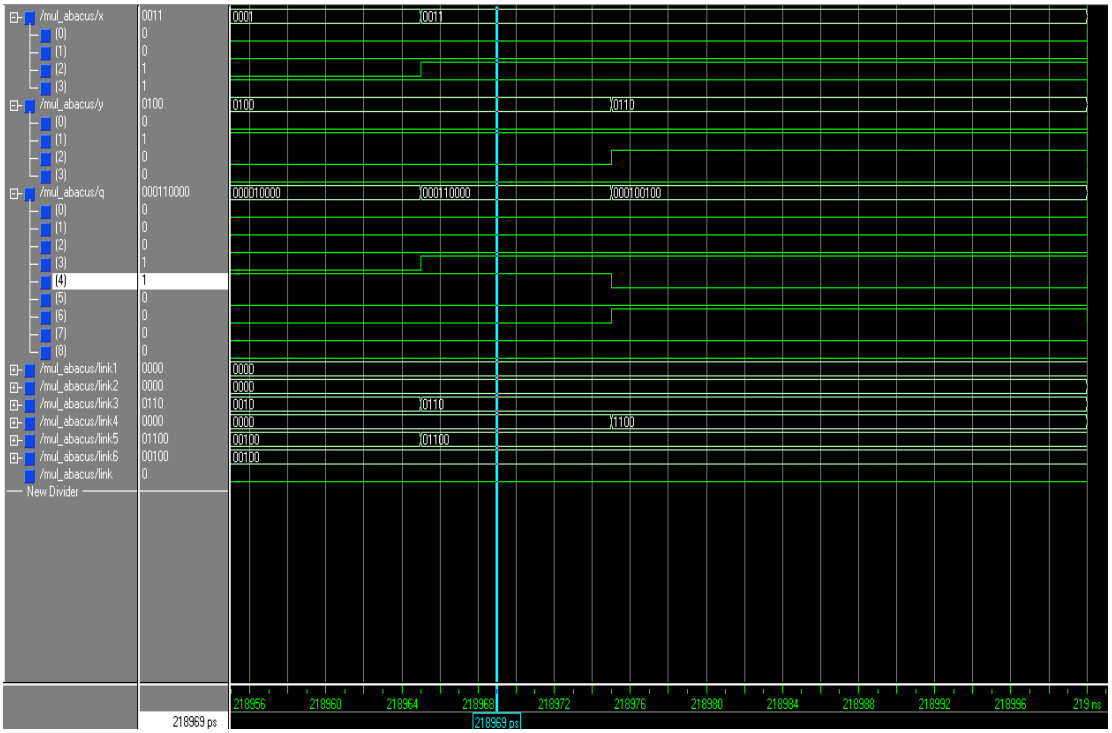


Figure 8.9: Simulation result of Chinese Abacus Adder

TABLE 8.1 COMPARISON OF ADDERS

Methods	Ripple Carry Adder	Carry Look Ahead Adder	Abacus Adder
Number of slices	4 out of 768 (4%)	5 out of 768 (3%)	6 out of 768 (3%)
Number of 4 input LUTs	7 out of 1536 (4%)	8 out of 1536 (3%)	10 out of 1536 (3%)
Number of bonded IOBs	13 out of 97 (17%)	14 out of 97 (17%)	13 out of 97 (17%)
Net propagation delay	10.923 ns	11.88 ns	9.678 ns
Frequency	91.54 MHz	84.17 MHz	103.32 MHz

Result:

Simulation results shows that although Chinese abacus adder takes larger area as compared to other ripple carry adder and carry look ahead adder, but results in 11% less propagation delay & 18.53% less propagation delay compares to ripple carry adder and carry look ahead adder, respectively.

Second work:

In this work, we have designed the Vedic multiplier with Chinese abacus adder with and without using reversible logic gates, and compared the results.

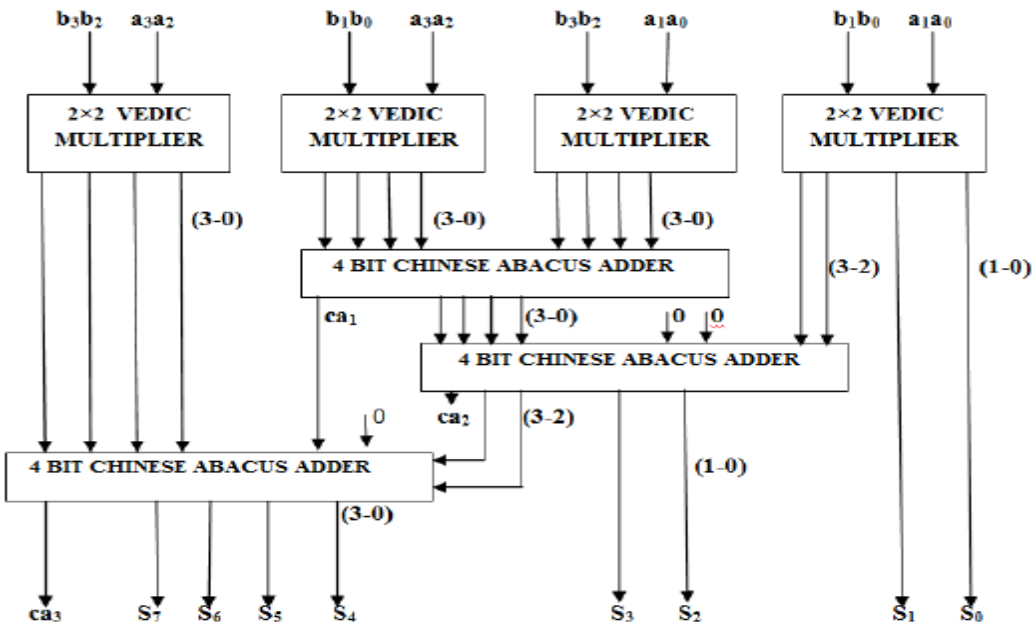


Figure 8.10: Block Diagram of 4x4 Vedic Multiplier Using Chinese Abacus Adder

- **Vedic multiplier without using revesible logic gates**

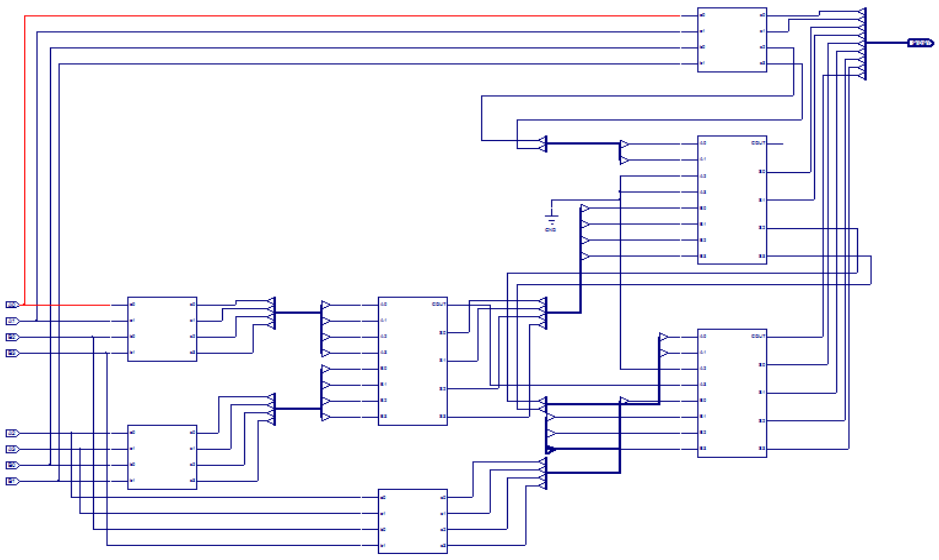


Figure 8.11: RTL view of 4x4 Vedic multiplier without using reversible logic gates.

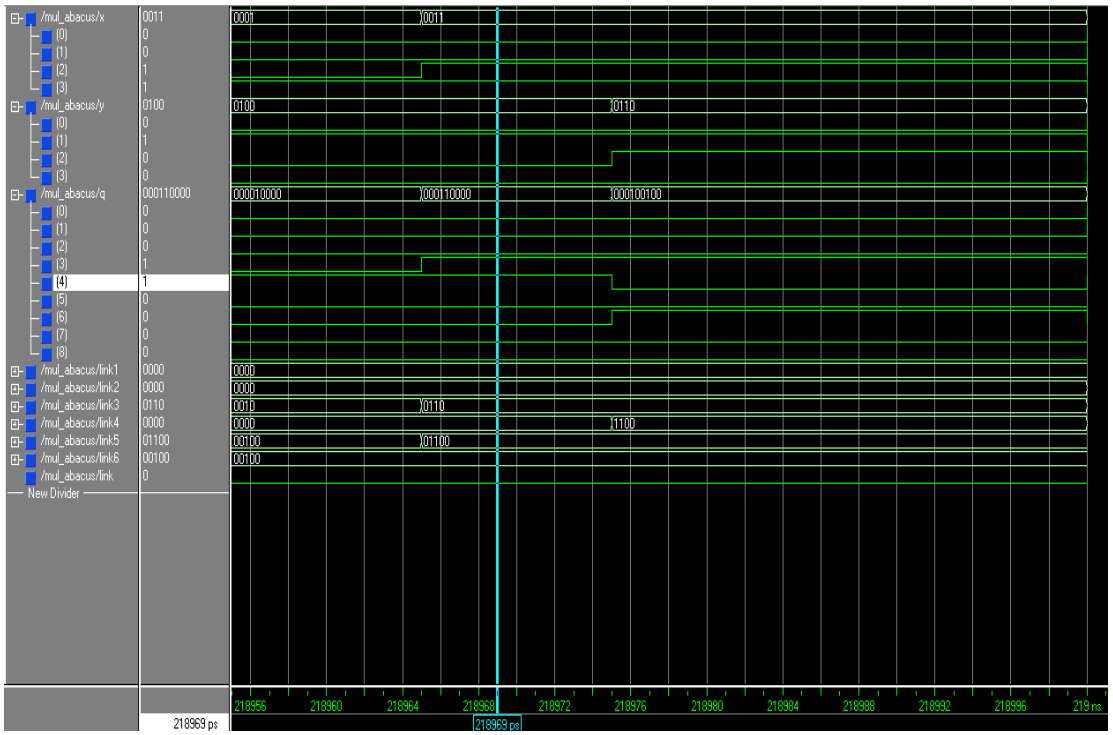


Figure 8.12: Simulation Result for 4x4 Vedic multiplier without using reversible logic gates.

- Vedic multiplier using reversible logic gates.

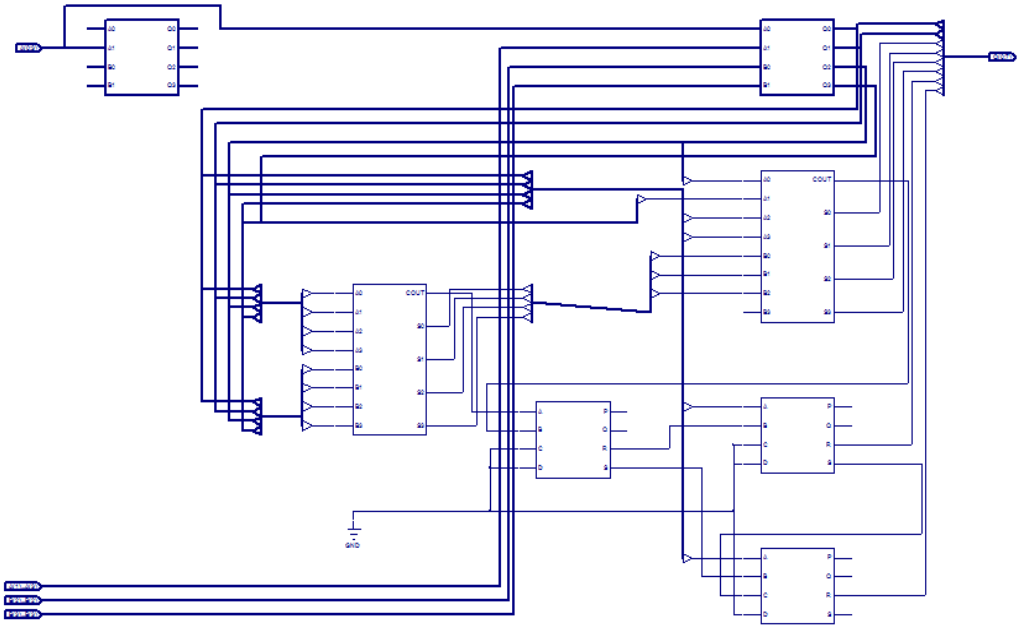


Figure 8.13: RTL veiw of 4x4 Vedic multiplier using reversible logic gates

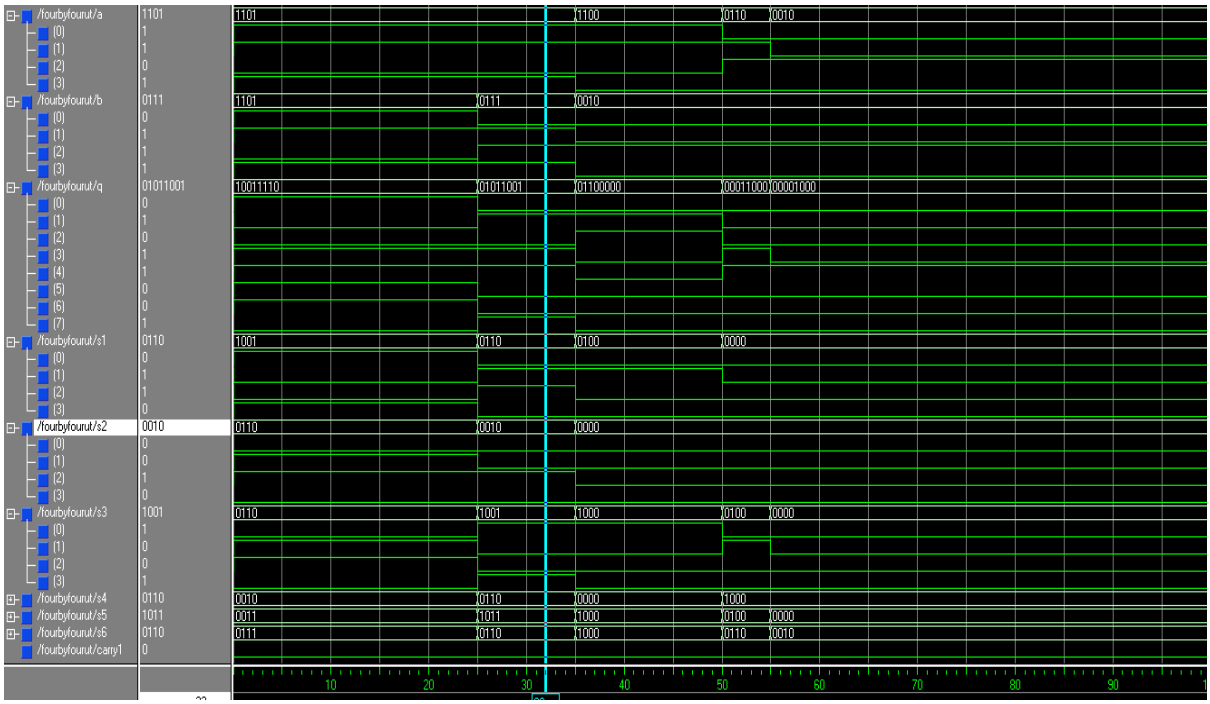


Figure 8.14: simulation result of 4x4 Vedic multiplier using reversible logic gates

TABLE 8.2: COMPARISON OF UT DESIGN WITH AND WITHOUT USING REVERSIBLE LOGIC GATES

Methods	**UT design using conventional method**	**UT design using reversible logic gates**
Number of slices	36 out of 768 (4%)	28 out of 768 (3%)
Number of 4 input LUTs	65 out of 1536 (4%)	49 out 1536 (3%)
Number of bonded IOBs	17 out of 97 (17%)	16 out of 97 (17%)
Net propagation delay	20.705 ns	15.830 ns
Frequency	48.297MHz	63.17MHz

8.1 RESULT:

Results show that 4x4 UT vedic multiplier designed in this work using reversible logic gates, consume less area and has high speed as compared to 4x4 UT vedic multiplier design without using reversible logic gates. Results shows that 4x4 UT vedic multiplier with using reversible logic gates is 23% faster and takes less area than UT design without reversible logic gates.

8.2 CONCLUSION & FUTURE SCOPE:

Results shows that Vedic multiplier design using UT sutra tailored with the Chinese abacus adder can be used in designing the application requiers fast computaion and on chip area is not in much concern. The use of Vedic multiplier approach results in a competitive technique as compared respect to conventional fast multiplier. The simulation results show that this approach of Vedic multiplier using Chinese Abacus adder is very efficient for low-power, high-speed applications. This architecture is also easy for pipeline implementation. Another advantage is that the methodology may reduce the number of ripple carries and partial product generation in many steps. It can be inferred that Vedic multiplier using Chinese Abacus adder is quite efficient as compared to the conventional multiplier, for multiplication operation.

REFERENCES

[1] R. Feynman, "Quantum Mechanical Computers," Optics News, Vol.1l, pp. 11-20, 1985.

[2] Peres, “Reversible logic and quantum computers”, Phys. Rev. A 32 (1985) 3266-3276.

[3] E. Fredkin and T. Toffoli,"Conservative Logic", Int'l 1 Theoretical Physics Vo121, pp.219-253, 1982.

[4] M. Shams, M. Haghparast and K. Navi, Novel reversible multiplier circuits in nanotechnology. World Appl. Sci. J., 3(5): 806-810, 2008.

[5] Shailja Shukla, Tarun Verma and Rita Jain, “Design of 16 Bit Carry Look Ahead Adder Using Reversible Logic”, International Journal of Electrical, Electronics and Computer Engineering 3(1): 83-89(2014).

[6] Franco Maloberti and Chen Gang, "The Chinese Abacus method can we use it for digital arithmetic", Proceedings of the 8th Great Lakes Symposium on VLSI, pp. 192 - 195 19-21 Feb. 1998.

[7] Franco Maloberti and Chen Gang, "Use of the Chinese Abacus method for digital arithmetic functions", Proceedings of the 1998 IEEE International Symposium on Circuits and Systems, Vol. 5, pp. 213 - 216, vol.5, 31 May - 3 June 1998.

[8] Franco Maloberti and Chen Gang, "Performing Arithmetic Functions with the Chinese Abacus Approach," IEEE Transactions on circuits and systems-II: Analog and Digital Signal processing, vol. 46, no. 12, pp. 1512- 1515, Dec. 1999.

[9] B.D. Andreev, E. Titlebaum, Friedman. E.G., "Tapered Transmission Gate Chains for Improved Carry Propagation,"MWSCAS-2002. The 45th Midwest Symposium on Circuits and Systems, Volume 3, pp. 449 - 452, Aug. 4-7, 2002.

[10] Behrooz Parhami “Computer Arithmetic-Algorithms and Hardware Designs”, Oxford University Press, Inc (2000).

[11] G Ganesh Kumar and V Charishma, “Design of high speed Vedic multiplier using Vedic mathematics techniques”, ltn'l J. of Scientific and Research Publications, Vol. 2 Issue 3, pp. 32-36 March 2012.

[12] Somayeh Babazadeh and Majid Haghparast, "Design of a Nanometric Fault Tolerant Reversible Multiplier Circuit" Journal of Basic and Applied Scientific Research, Vol 2 pp. 25-28 Mar 2012.

[13] Shu-Chung Yi, Kun-Tse Lee, Jin-Jia Chen, Chien-Hung Lin, Chuen-Ching Wang, “The new architecture of radix-4 Chinese abacus adder”, Proceedings of the 36th International Symposium on Multiple-Valued Logic (ISMVL’06) on 17 May 2006 IEEE Conference.

[14] Shu-Chung Yi, “Performing Arithmetic Functions with the Chinese Abacus Approach”, IEEE Transaction on circuits and systems-II on Mar-1, 2012.

[15] Rakshith Saligram and Rakshith T.R. "Design of Reversible Multipliers for linear filtering Applications in DSP", International Journal of VLSI Design and Communication systems, Dec-2012.

[16] Harpreet Singh Dhillon Abhijit Mitra, "A Digital Multiplier Architecture using Urdhava Tiryabhyam Sutra of Vedic Mathematics", International Conference on Design & Technology of Integrated Systems in Nanoscale Era, Dec-2008.

[17] Purushottam D. Chidgupkar and Mangesh T. Karad, “The Implementation of Vedic Algorithms in Digital Signal Processing”, Global J. of Eng. Educ., Vol.8, No.2 © 2004 UICEE Published in Australia.

[18] Himanshu Thapliyal and Hamid R. Arabnia, “A Time-Area- Power Efficient Multiplier and Square Architecture Based On Ancient Indian Vedic Mathematics”, International Conference on Low Power Design at Graduate Studies Research Center Athens, Georgia U.S.A. Jul 17, 2011.

[19] E. Abu-Shama, M. B. Maaz, M. A. Bayoumi, "A Fast and Low Power Multiplier Architecture", The Center for Advanced Computer Studies, Circuit and System, IEEE 39th Midwest Symposium on Vol (1), 18-21 Aug 1996

[20] Harpreet Singh Dhillon and Abhijit Mitra, "A Reduced- Bit Multiplication Algorithm for Digital Arithmetics", International Journal of Computational and Mathematical Sciences 2;2 © www.waset.org Spring 2008.

[21] Shamim Akhter, "VHDL Implementation of Fast NXN Multiplier Based on Vedic Mathematics", IEEE Conference on Implementation of Low power Design, Jaypee Institute of Information Technology University, Noida, Nov-2013.

[22] Charles E. Stroud, "A Designer‟s Guide to Built-In Self-Test", University of North Carolina at Charlotte, ©2002 Kluwer Academic Publishers New York, Boston, Dordrecht, London, Moscow.

[23] Douglas Densmore, "Built-In-Self Test (BIST) Implementations An overview of design tradeoffs", University of Michigan EECS 579 – Digital Systems Testing by Professor John P. Hayes 12/7/01.

[24] Jagadguru Swami Sri Bharati Krishna Tirthji Maharaja,"Vedic Mathematics", Motilal Banarsidas, Varanasi, India, 1986.

[25] Himanshu Thapliyal, Saurabh Kotiyal and M. B Srinivas, "Design and Analysis of A Novel Parallel Square and Cube Architecture Based On Ancient Indian Vedic Mathematics", IEEE Conference of Centre for VLSI and Embedded System Technologies, International Institute of Information Technology, May 2005

[26] Himanshu Thapliyal and M.B Srinivas, "VLSI Implementation of RSA Encryption System Using Ancient Indian Vedic Mathematics", Conference of IEEE of VLSI and Embedded System Technologies, International Institute of Information Technology, May 2004.

[27] Himanshu Thapliyal and M.B Srinivas, "An Efficient Method of Elliptic Curve Encryption Using Ancient Indian Vedic Mathematics", Conference of IEEE, July 2005.

[28] Deming Chen, Jason Cong, and Peichan Pan, "FPGA Design Automation: A Survey", Foundations and Trends in Electronic Design Automation Volume 1 Issue 3, November 2006.

[29] Ken Chapman, "Initial Design for Spartan-3E Starter Kit (LCD Display Control)", Xilinx Ltd 16th February 2006.

[30] Goh Keng Hoo, "Verilog design of Input / Output Processor with Built-In-Self-Test", Conference of Applied Science in Universiti Teknologi Malaysia, April 2007.

[31] Michael L. Bushnell and Vishwani D. Agrawal, "Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits", Kluwer Academic Publishers, 2002.

[32] Dr. Malti Bansal, Diksha Ruhela, "High Speed & Area Efficient Vedic Multiplier using Adiabatic Logic", Journal of Basic and Applied Engineering Research Volume 1, Number 11; October-December 2014 pp. 14-17.

[33] Diksha Ruhela & Dr. Malti Bansal, "Vedic Multiplier with Chinese Abacus Adder Design using Reversible Logic Gates", International Conference on VLSI, Communication and Network (VCAN-2015), Alwar-301030, ISBN:978-93-84869-55-7, April – 2015 pp. 9-12.

[34] Diksha Ruhela & Dr. Malti Bansal, "Adiabatic Vedic Multiplier Design Using Chinese Abacus Approach", International Journal of Advanced Research in Computer and Communication Engineering, ISSN (Online) 2278-1021 ISSN (Print) 2319-5940 Vol. 4, Issue 4, April 2015

APPENDIX

APPENDIX A
VHDL CODE OF PROPOSED WORK

- **Ripple Carry Adder**

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

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity rplcarry is port ( A0,A1,A2,A3,B0,B1,B2,B3: IN BIT; S0,S1,S2,S3,COUT : OUT BIT);
end rplcarry;

architecture Behavioral of rplcarry is
  COMPONENT  threebitadder PORT( A, B, CIN :IN BIT; SUM, CARRY : OUT BIT);
    END COMPONENT;
                SIGNAL CARRY_OUT : BIT_VECTOR(1 TO 3);
                SIGNAL CIN1 : BIT :='0';
begin
        P1: threebitadder PORT MAP( A0, B0,CIN1,S0,CARRY_OUT(1));
        P2: threebitadder PORT MAP( A1, B1,CARRY_OUT(1),S1,CARRY_OUT(2));
        P3: threebitadder PORT MAP( A2, B2,CARRY_OUT(2),S2,CARRY_OUT(3));
        P4: threebitadder PORT MAP( A3, B3,CARRY_OUT(3),S3,COUT);

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity threebitadder is     PORT (A , B, CIN: IN BIT ; SUM, CARRY : OUT BIT);
end threebitadder;

architecture Behavioral of threebitadder is
BEGIN
        SUM<= ( (A XOR B) XOR CIN);
        CARRY<= ( A AND ( B XOR CIN) ) OR( B AND CIN);
end Behavioral;
```

- **Carry Look Ahead Adder**

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;


ENTITY c_l_addr IS PORT ( x_in : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
 y_in : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        carry_in : IN STD_LOGIC; sum : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
```

```
        carry_out : OUT STD_LOGIC );
        END c_l_addr;
        ARCHITECTURE behavioral OF c_l_addr IS
        SIGNAL h_sum : STD_LOGIC_VECTOR (3 DOWNTO 0);
        SIGNAL carry_generate : STD_LOGIC_VECTOR (3 DOWNTO 0);
        SIGNAL carry_propagate : STD_LOGIC_VECTOR (3 DOWNTO 0);
        SIGNAL carry_in_internal : STD_LOGIC_VECTOR(3 DOWNTO 1);
                BEGIN
                h_sum <= x_in XOR y_in;
                carry_generate <= x_in AND y_in;
                carry_propagate <= x_in OR y_in;
  PROCESS (carry_generate,carry_propagate,carry_in_internal)
  BEGIN
  carry_in_internal(1) <= carry_generate(0) OR (carry_propagate(0) AND carry_in);
  inst: FOR i IN 1 TO (3-1) LOOP
  carry_in_internal(i+1) <= carry_generate(i) OR (carry_propagate(i) AND
carry_in_internal(i));
  END LOOP;
   carry_out <= carry_generate(3) OR (carry_propagate(3) AND carry_in_internal(3));
    END PROCESS;
        sum(0) <= h_sum(0) XOR carry_in;
        sum(3 DOWNTO 1) <= h_sum(3 DOWNTO 1) XOR carry_in_internal(3 DOWNTO
1);
        END behavioral;
```

- **Chinese Abacus Adder**

```
entity ADDER is PORT (A0,A1,A2,A3,B0,B1,B2,B3:IN BIT; S0,S1,S2,S3,COUT : OUT
BIT);
end ADDER;
```

```
architecture Behavioral of ADDER is
  COMPONENT BinTOabc port(A0,A1,A2,A3:IN BIT; H2, H1, H0,L2, L1, L0: OUT BIT);
  end component;
  component PtoA PORT(HA0,HA1,HA2,HB0,HB1,HB2: IN BIT; K0,K1,K2,K3,K4,K5:
OUT BIT);
  end component;
  component  TtoB PORT(K0,K1,K2,K3,K4,K5,CIN:IN BIT; B2, B3,COUT: OUT BIT);
  end component;
  SIGNAL HA,HB,LA,LB : BIT_VECTOR(0 TO 2);
  SIGNAL KA,KB:BIT_VECTOR(0 TO 5);
  SIGNAL C1,CIN: BIT;
begin
   Z1: BinTOabc PORT MAP(A0,A1,A2,A3,HA(2),HA(1),HA(0),LA(2),LA(1),LA(0));
   Z2: BinTOabc PORT MAP(B0,B1,B2,B3,HB(2),HB(1),HB(0),LB(2),LB(1),LB(0));
   Z3:PtoA PORT MAP(HA(0),HA(1),HA(2),HB(0),HB(1),HB(2),KA(0),KA(1),KA(2),KA(3),
KA(4),KA(5));
   Z4: PtoA PORT MAP(LA(0),LA(1),LA(2),LB(0),LB(1),LB(2),KB(0),KB(1),KB(2),KB(3),
KB(4),KB(5));
   Z5: TtoB   PORT MAP(KA(0),KA(1),KA(2),KA(3),KA(4),KA(5), C1,S2,S3,COUT);
   Z6: TtoB   PORT MAP(KB(0),KB(1),KB(2),KB(3),KB(4),KB(5), CIN,S0,S1,C1);
end Behavioral;


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


--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
```

```
--use UNISIM.VComponents.all;

entity BinTOabc is port(A0,A1,A2,A3:IN BIT; H2, H1, H0,L2, L1, L0: OUT BIT);
end BinTOabc;

architecture Behavioral of BinTOabc is
begin
   H2<=A3 AND A2;
   H1<=A3;
   H0<=A3 OR A2;
   L2<=A1 AND A0;
   L1<=A1;
   L0<=A1 OR A0;
end Behavioral;

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

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity PtoA is PORT(HA0,HA1,HA2,HB0,HB1,HB2: IN BIT; K0,K1,K2,K3,K4,K5: OUT
BIT);
end PtoA;
architecture Behavioral of PtoA is
begin
 PROCESS      (HA0,HA1,HA2,HB0,HB1,HB2)
```

```
 variable F1,F2: BIT;
begin
 F1:=( not HA2) AND HA1;
 F2:=( NOT HA1) AND HA0;
 K0<=(( not HA0) AND HB0) OR (F2 AND '1') OR (F1 AND '1')OR (HA2 AND '1') ;
 K1<=(( not HA0) AND HB1) OR (F2 AND HB0) OR (F1 AND '1') OR (HA2 AND '1') ;
 K2<=(( not HA0) AND HB2) OR (F2 AND HB1) OR (F1 AND HB0) OR (HA2 AND '1') ;
 K3<=(( not HA0) AND '0') OR (F2 AND HB2) OR (F1 AND HB1) OR (HA2 AND HB0) ;
 K4<=(( not HA0) AND '0') OR (F2 AND '0') OR (F1 AND HB2) OR (HA2 AND HB1) ;
 K5<=(( not HA0) AND '0') OR (F2 AND '0') OR (F1 AND '0') OR (HA2 AND HB2) ;
  END PROCESS;
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity TtoB is     PORT(K0,K1,K2,K3,K4,K5,CIN:IN BIT; B2, B3,COUT: OUT BIT);
end TtoB;
architecture Behavioral of TtoB is
begin
   B2<=((NOT K0) AND CIN) OR(((NOT K1) AND K0) AND (NOT CIN)) OR
      (((NOT K2) AND K1) AND CIN)OR (((NOT K3) AND K2) AND (NOT CIN))
    OR (((NOT K4) AND K3) AND CIN) OR (((NOT K5) AND K4) AND (NOT CIN))
    OR (K5 AND CIN);
```

```
    B3<= K5 OR (K4 AND CIN) OR (((NOT K2) AND K0) AND CIN) OR (((NOT K3) AND
K1) AND (NOT CIN));

    COUT<= K3 OR (K2 AND CIN);

end Behavioral;
```

- **UT VEDIC MULTIPLIER DESIGN USING REVERSIBLE LOGIC GATE**

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mul_abacus is port( x , y  : in bit_vector(0 to 3); q : inout bit_vector(0 to 8));
end mul_abacus;
architecture Behavioral of mul_abacus is
--component ADDER PORT (A0,A1,A2,A3,B0,B1,B2,B3:IN BIT; S0,S1,S2,S3,COUT : OUT BIT);
--end component;
 component  rplcarry port( A0,A1,A2,A3,B0,B1,B2,B3: IN BIT; S0,S1,S2,S3,COUT : OUT BIT);
end component;
component mux_twobytwo port ( a0, a1, b0, b1 : in bit; s0, s1,s2,s3 : out bit);
end component;
  signal link1, link2, link3, link4 : bit_vector(0 to 3);
  signal link5, link6 : bit_vector(0 to 4);
  signal link : bit :='0';
begin
   m1 :  mux_twobytwo port map( x(0), x(1), y(0), y(1), q(0), q(1), link1(2), link1(3));
   m2 :  mux_twobytwo port map( x(0), x(1), y(2), y(3), link2(0), link2(1), link2(2), link2(3));
   m3 :  mux_twobytwo port map( x(2), x(3), y(0), y(1), link3(0), link3(1), link3(2), link3(3));
   m4 :  mux_twobytwo port map( x(2), x(3), y(2), y(3), link4(0), link4(1), link4(2), link4(3));
```

```
   a1 :  rplcarry  port map(link2(0), link2(1), link2(2), link2(3), link3(0), link3(1), link3(2),
                        link3(3),link5(0), link5(1), link5(2), link5(3), link5(4));
   a2 :  rplcarry    port map(link1(2), link1(3), link, link, link5(0), link5(1) , link5(2),
                        link5(3),q(2), q(3), link6(2), link6(3), link6(4) );
   a3 :  rplcarry     port map(link6(2), link6(3), link, link5(4), link4(0), link4(1), link4(2),
                        link4(3),q(4), q(5), q(6), q(7), q(8));
end Behavioral;

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

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux_twobytwo is port ( a0, a1, b0, b1 : in bit; s0, s1,s2,s3 : out bit);
end mux_twobytwo;
architecture Behavioral of mux_twobytwo is
begin
    s0 <=  a0 and b0;
    s1 <= ( a1 and b0) xor (a0 and b1);
    s2 <= (( a0 and a1) and (b0 and b1))xor (a1 and b1);
    s3 <= ((a0 and a1) and (b0 and b1));
end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity rplcarry is port ( A0,A1,A2,A3,B0,B1,B2,B3: IN BIT; S0,S1,S2,S3,COUT : OUT BIT);
end rplcarry;
architecture Behavioral of rplcarry is
  COMPONENT  threebitadder PORT( A, B, CIN :IN BIT; SUM, CARRY : OUT BIT);
    END COMPONENT;
                SIGNAL CARRY_OUT : BIT_VECTOR(1 TO 3);
                SIGNAL CIN1 : BIT :='0';
begin
        P1: threebitadder PORT MAP( A0, B0,CIN1,S0,CARRY_OUT(1));
        P2: threebitadder PORT MAP( A1, B1,CARRY_OUT(1),S1,CARRY_OUT(2));
        P3: threebitadder PORT MAP( A2, B2,CARRY_OUT(2),S2,CARRY_OUT(3));
        P4: threebitadder PORT MAP( A3, B3,CARRY_OUT(3),S3,COUT);
end Behavioral;

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity threebitadder is     PORT (A , B, CIN: IN BIT ; SUM, CARRY : OUT BIT);
end threebitadder;

architecture Behavioral of threebitadder is
BEGIN
        SUM<= ( (A XOR B) XOR CIN);
        CARRY<= ( A AND ( B XOR CIN) ) OR( B AND CIN);

end Behavioral;
```

- **UT VEDIC MULTIPLIER DESIGN USING CONVENTIONAL METHODE**

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity utmulti is port(A0,A1,A2,A3,B0,B1,B2,B3 : IN BIT; S : OUT BIT_VECTOR( 0 TO 8));
end utmulti;

architecture Behavioral of utmulti is
  COMPONENT  mux_twobytwo port  (a0, a1, b0, b1 : in bit; s0, s1,s2,s3 : out bit);
  end component;
  --component rplcarry  port ( A0,A1,A2,A3,B0,B1,B2,B3: IN BIT; S0,S1,S2,S3,COUT : OUT
BIT);
--end component;
component ADDER PORT (A0,A1,A2,A3,B0,B1,B2,B3:IN BIT; S0,S1,S2,S3,COUT : OUT
BIT);
END COMPONENT;
signal link2, link3, link4, link5 : bit_vector( 0 to 3);
signal link1, link6 : bit_vector(2 to 3);
signal link : bit :='0';
SIGNAL CARRY1, CARRY2 : BIT;
begin
  m1: mux_twobytwo port map (A0, A1, B0, B1, S(0), S(1),link1(2), link1(3));
  m2: mux_twobytwo port map (A0, A1, B2, B3, link2(0), link2(1), link2(2), link2(3));
```

```
  m3: mux_twobytwo port map (A2, A3, B0, B1, link3(0), link3(1), link3(2), link3(3));
  m4: mux_twobytwo port map (A2, A3, B2, B3, link4(0), link4(1), link4(2), link4(3));
  m5: ADDER     port map (link2(0), link2(1), link2(2), link2(3), link3(0), link3(1), link3(2),
link3(3),
                         link5(0), link5(1), link5(2), link5(3), CARRY1);
  m6: ADDER     port map (link1(2), link1(3), link, link, link5(0), link5(1), link5(2), link5(3),
                         S(2), S(3), link6(2), link6(3), CARRY2);
  m7: ADDER    port map (link6(2), link6(3), link, CARRY1, link4(0), link4(1), link4(2),
link4(3),
                         S(4), S(5),S(6),S(7),S(8));

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

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ADDER is PORT (A0,A1,A2,A3,B0,B1,B2,B3:IN BIT; S0,S1,S2,S3,COUT : OUT
BIT);
end ADDER;

architecture Behavioral of ADDER is
  COMPONENT BinTOabc port(A0,A1,A2,A3:IN BIT; H2, H1, H0,L2, L1, L0: OUT BIT);
  end component;
  component PtoA PORT(HA0,HA1,HA2,HB0,HB1,HB2: IN BIT; K0,K1,K2,K3,K4,K5:
OUT BIT);
  end component;
```

```
  component  TtoB PORT(K0,K1,K2,K3,K4,K5,CIN:IN BIT; B2, B3,COUT: OUT BIT);
  end component;
  SIGNAL HA,HB,LA,LB : BIT_VECTOR(0 TO 2);
  SIGNAL KA,KB:BIT_VECTOR(0 TO 5);
  SIGNAL C1,CIN: BIT;
begin
   Z1: BinTOabc PORT MAP (A0,A1,A2,A3,HA(2),HA(1),HA(0),LA(2),LA(1),LA(0));
   Z2: BinTOabc PORT MAP (B0,B1,B2,B3,HB(2),HB(1),HB(0),LB(2),LB(1),LB(0));
   Z3:PtoA PORT MAP
(HA(0),HA(1),HA(2),HB(0),HB(1),HB(2),KA(0),KA(1),KA(2),KA(3),
KA(4),KA(5));
   Z4:PtoA PORT MAP (LA(0),LA(1),LA(2),LB(0),LB(1),LB(2),KB(0),KB(1),KB(2),KB(3),
KB(4),KB(5));
   Z5: TtoB     PORT MAP(KA(0),KA(1),KA(2),KA(3),KA(4),KA(5), C1,S2,S3,COUT);
   Z6: TtoB     PORT MAP(KB(0),KB(1),KB(2),KB(3),KB(4),KB(5), CIN,S0,S1,C1);
end Behavioral;

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

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux_twobytwo is port ( a0, a1, b0, b1 : in bit; s0, s1,s2,s3 : out bit);
end mux_twobytwo;

architecture Behavioral of mux_twobytwo is
```

```
begin
    s0 <=  a0 and b0;
    s1 <= ( a1 and b0) xor (a0 and b1);
    s2 <= (( a0 and a1) and (b0 and b1))xor (a1 and b1);
    s3 <= ((a0 and a1) and (b0 and b1));
end Behavioral;

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

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity threebitadder is     PORT (A , B, CIN: IN BIT ; SUM, CARRY : OUT BIT);
end threebitadder;
architecture Behavioral of threebitadder is
BEGIN
            SUM<= ( (A XOR B) XOR CIN);
            CARRY<= ( A AND ( B XOR CIN) ) OR( B AND CIN);
end Behavioral;
```

APPENDIX B
CERTIFICATE IN CONFERENCE OF IJARCCE-2015
AND PUBLISHED PAPER

APPENDIX C

CERTIFICATE IN CONFERENCE OF VCAN-2015 AND PUBLISHED PAPER

APPENDIX D

CERTIFICATE IN CONFERENCE OF AEPCECE-2014

AND PUBLISHED PAPER