BEHAVIORAL LEVEL SIMULATION OF A VEDIC MULTIPLIER FOR AN ALU UNIT

A PROJECT REPORT

Submitted by

KIRUTHIKA. R	**Reg. No.:13BEC075**
MAHESWARI. V. S	**Reg. No.:13BEC085**
MALATHI.S	**Reg. No.:13BEC086**
MERLIN SAHAYA AJITHA.A	**Reg. No.:13BEC302**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

ANNA UNIVERSITY: CHENNAI

APRIL 2017

BONAFIDE CERTIFICATE

Certified that this project report **"BEHAVIORAL LEVEL SIMULATION OF A VEDIC MULTIPLIER FOR AN ALU UNIT" is the bonafide work of KIRUTHIKA. R [Reg. No: 13BEC075], MAHESWARI. V. S [13BEC085], MALATHI. S [13BEC086] and MERLIN SAHAYA AJITHA. A [13BEC302]** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other project or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Mr.R.KARTHIKUMAR M.E.

PROJECT SUPERVISOR

Department of ECE

Kumaraguru College of Technology

Coimbatore – 641049

SIGNATURE

Dr.K.MALARVIZHI M.E., Ph.D

PROFESSOR AND HEAD

Department of ECE

Kumaraguru College of Technology

Coimbatore – 641049

The candidates with Register No: 13BEC075, 13BEC085, 13BEC086 and 13BEC302 are examined by us in the project viva-voce examination held on

_____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

The demand for enhancing the ability of processors to handle complex and challenging processes has resulted in the integration of a number of cores of processors into a single chip. The speed of ALU depends on the multiplier unit. In algorithmic and structural levels, numerous multiplication techniques have been developed to enhance the multiplier efficiency which concentrates in the reduction of the partial products and their addition methods but the multiplication principle remains the same in all of the cases. Vedic Mathematics is an ancient mathematics system which has a unique set of calculation techniques based on the 16 Sutras. Employment of these techniques in the coprocessor computation algorithms will reduce the complexity, execution time, area, power etc. Our work is to prove that **Nikhilam Navatashcaramam Dashatah** Vedic method is one of the efficient ways for multiplication. In this method, the multiplication is done with the help of addition, or logic and basic multiplication. This method is implemented in the software XILINX using the Verilog code.

ACKNOWLEDGEMENT

We express our sincere thanks to the Management of Kumaraguru College of Technology and Joint Correspondent **Shri.ShankarVanavarayar** for the kind support and for providing necessary facilities to carry out the project work.

We would like to express our sincere thanks to our beloved Principal **Dr.R.S.Kumar, Ph.D.,** Kumaraguru College of Technology, who encouraged us in each and every step of the project.

We would like to thank **Dr.K.Malarvizhi, M.E., Ph.D.,** Head of the Department, Electronics and Communication Engineering, for her kind support and for providing necessary facilities to carry out the project work.

We wish to thank with everlasting gratitude to our Project Coordinator **Dr.A.Vasuki, M.E., Ph.D.,** Department of Electronics and Communication Engineering for her consistent support throughout the course of this project work.

We are greatly privileged to express our deep sense of gratitude and heartfelt thanks to our Project Guide **Mr.R.Karthikumar, M.E.,** Department of Electronics and Communication Engineering for his expert counseling and guidance to make this project to a great deal of success and also we wish to convey our regards to all teaching and non-teaching staff of ECE Department for their help and cooperation.

Finally, we thank our parents and our family members for giving us the moral support and abundant blessings in all of our activities and our dear friends who helped us to endure our difficult times with their unfailing support and warm wishes.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABBREVIATION	FULL FORM OF ABBREVIATION	PAGE NO.
ALU	Arithmetic – Logic Unit	3
LHS	Left Hand Side	11
RHS	Right Hand Side	11
MUX	Multiplexer	11
Col	Column	11
BCD	Binary Converted to Decimal	12
FPGA	Field Programmable Gate Array	26

CHAPTER 1

INTRODUCTION

The process of adding a number or an integer to itself for a specified number of times is Multiplication. In other words, it is known as “repeated addition”. When the addition is repeated for a small number of times for instance 4+4+4=12, where addition is done twice with the number ‘4’, multiplication is not that of use but for a large number of times like 1000 times addition then it is a problem. In such cases, multiplication is very useful and it is a simple method. Due to the concept of multiplication, many real-time problems are solved. Nowadays, due to the development of technology, processors are playing a vital role in solving the real-time problems. One of the most vital functions in many processor applications is Multiplication. The unit which performs Multiplication is known as Multiplier. Various structures and logic have been proposed for implementation of multipliers.

1.1 VEDIC MATHEMATICS

The name given to the ancient system of Indian Mathematics is Vedic Mathematics. It was rediscovered from the Vedas by Sri Bharati Krsna Tirthaji (1884-1960) between 1911 and 1918. According to all his research, vedic mathematics is based on sixteen Sutras. The sixteen sutras are as follows:

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 - Factors of the sum is equal to the sum of factors.
6. Gunitasamuchyah - The product of sum is equal to 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 same then 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.

The above formulae describe the way the mind naturally works. Coherence is the most striking feature of the Vedic system and it is easily understandable. Thus, mathematics is made easy, enjoyable and it encourages innovation.

Therefore, the 'difficult' problems or huge sums can be solved immediately by the Vedic method. The above methods are just a part of a complete system of Mathematics which is more systematic than the modern 'system'. Thus, the coherent structure of mathematics is provided by Vedic Mathematics.

Vedic Mathematics is used in showing its application in fast calculations like multiplication, division, squaring, cubing and some other. Many unique solutions are provided by mathematics in several instances where trial and error method is available.

A highly efficient approach is provided by Vedic Mathematics to mathematics by covering a wide range from elementary multiplication to relatively advanced topics. But the scheme of Vedic is not simply a collection of rapid methods but it is a unified approach. The properties of numbers in every practical application are extensively exploited by Vedic Mathematics.

Various arithmetic modules, using these Vedic techniques, can be designed and integrated into a Vedic ALU, compatible for Co-Processors.

1.2 ARITHMETIC - LOGIC UNIT

A combinational digital electronic circuit which performs arithmetic and bitwise operations on integer binary numbers is called as ALU (Arithmetic – Logic Unit). Since, it is a combinational logic circuit, its outputs in response to input changes will change asynchronously.

Normally stable signals are applied to all of the ALU inputs and when the "propagation delay" has passed for the signals to be propagated through the ALU circuitry, the result of the ALU operation appears at the ALU outputs. The external circuitry is connected to the ALU. It allows sufficient time for the signals to propagate through the ALU before sampling the result.

In general, ALU is controlled by an external circuitry which apply signals to the inputs. The external circuitry employs sequential logic in order to control the ALU operation. Sequential logic is a clock signal of a sufficiently low frequency to ensure that there is enough time for the ALU outputs to be settled under worst-case conditions.

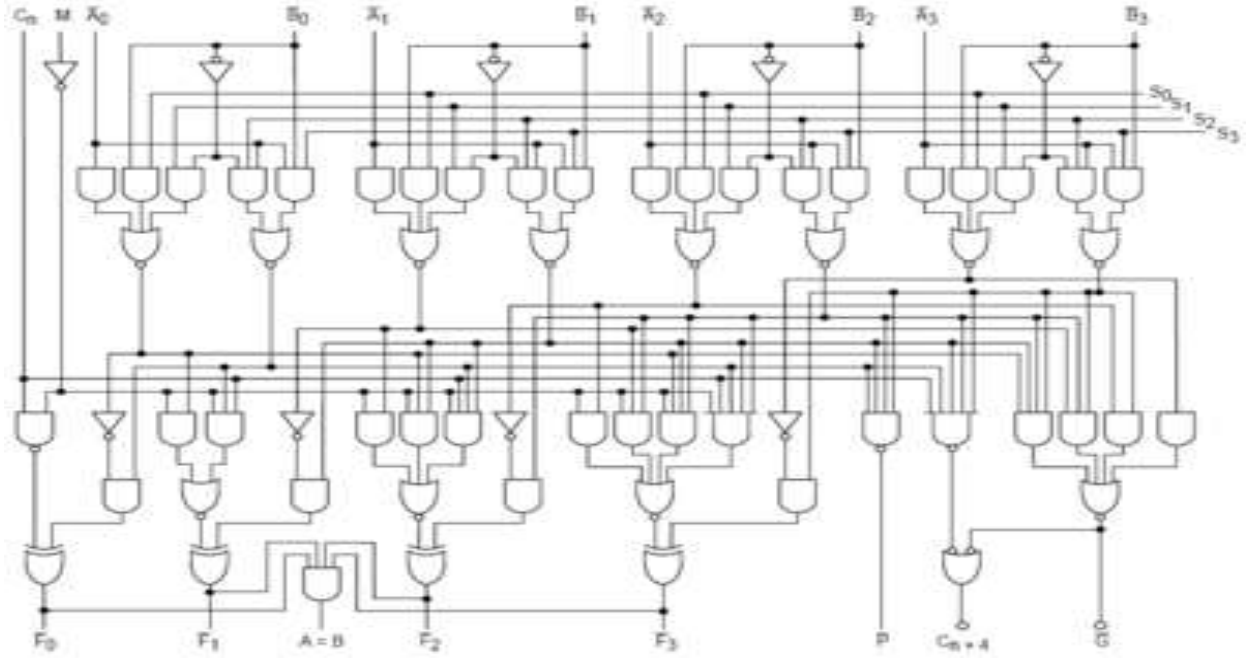


Figure 1.2.1: A simple four – bit ALU

The speed of ALU depends on the multiplier. Hence many techniques for multiplication are proposed such that the speed of the ALU can be improvised.

As we know, Vedic Mathematics is more systematic and efficient than the general multiplication concept, Vedic Mathematics is implemented in the multiplier. Hence, the multiplier is known as the Vedic Multiplier.

CHAPTER 2

LITERATURE SURVEY

2.1 ARRAY MULTIPLIERS

An Array multiplier has very common regular structure. Array multipliers are implemented by directly mapping the manual multiplication into the hardware. An n bit Array multiplier has n x n array of AND gates in order to generate partial products, n x (n-1) full adders and n half adders. A full adder sums each partial product bit with the sum from the previous adder and a carry from the less significant previous adder. The length of the multiplier is denoted by the number of rows in an array multiplier and width of multiplicand is denoted by width of each row.

The design steps for a 16x16 bit array multiplier which is simulated using Xilinx 8.1 tool is explained below.

The Multiplier circuit is based on add and shift algorithm. The multiplication of the multiplicand with one multiplier bit is done to generate each partial product bit. Then the partial products are shifted based on their bit orders and then added. The addition is performed with normal carry propagate adder. n-1 adders are required where n is the multiplier length.

Array multipliers are very slow as their critical path is very long. The main advantage is the regular structure which leads to easy layout and design.

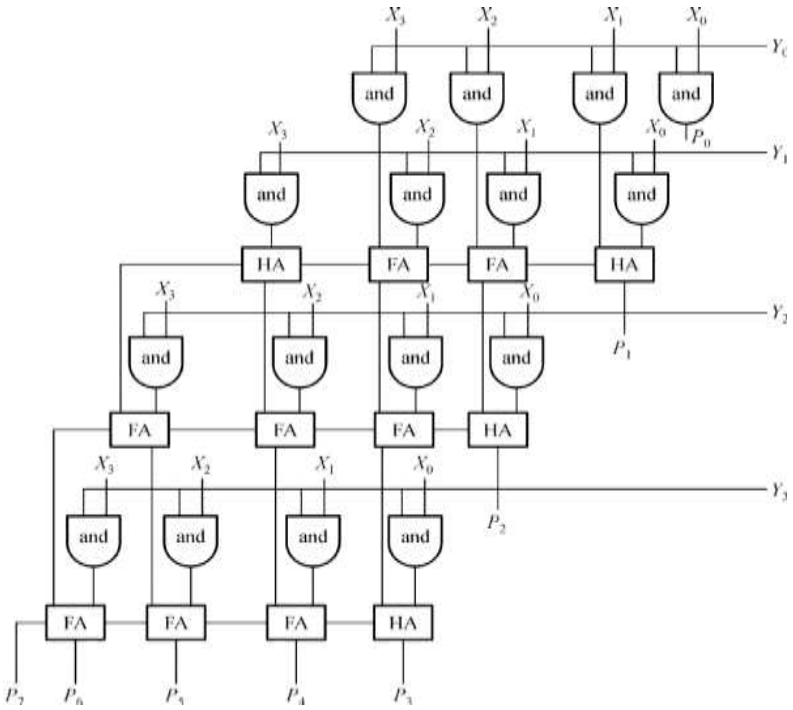


Figure 2.1.1 4 x 4 Array Multiplier

X,Y – operands of 4 bit Z – 8 bit product of X and Y FA – full adder

2.2 CARRY SAVE MULTIPLIER

The simplest of all multipliers is the carry save array multiplier. The principle idea behind the carry save adders is it sums the partial products together to reduce them. Since most of the reduction computes the lower half of the product, the final carry propagate adder only needs to add the upper half of the product. For the Carry save array multiplier implementation, each adder is modified such that it can perform partial product generation and an addition. Thus, in the Carry save multiplier partial product generation is done by utilizing AND gates and uses an array of carry save adders to perform reduction. Thus, the partial products has nm number of AND gates, m half adders and ((n-1) . (m-1))-1 = n .m – n – m full adders. The final row of (n-1) adders is a ripple carry adder carry propagate adder.

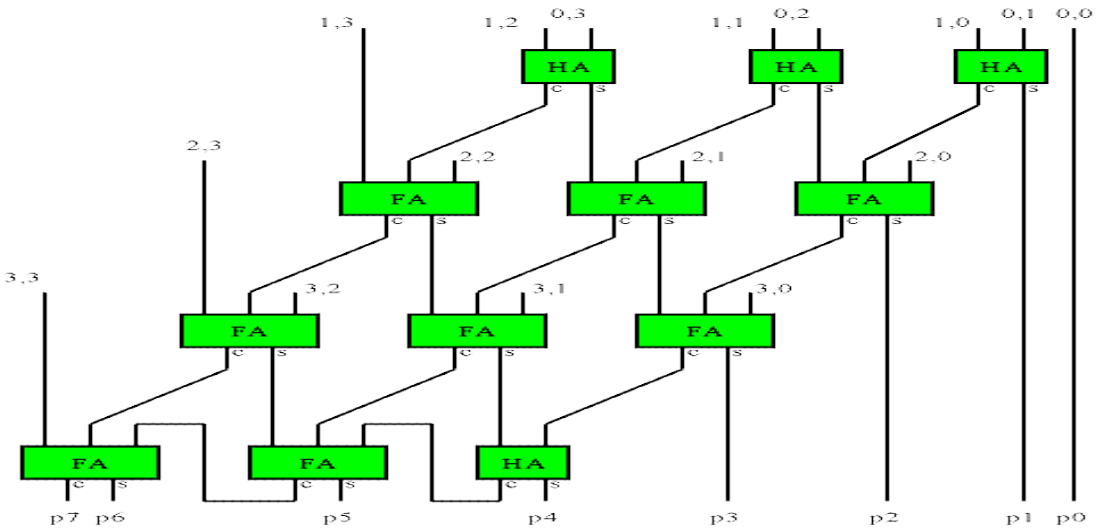


Figure 2.2.1 : 4x4 Carry Save Multiplier

X,Y – operands of 4 bit P – 8 bit product of X and Y FAB – full adder

This Carry save multiplier was first proposed by Braun Edward Louis in 1963. It is restricted to perform the addition of two unsigned numbers. Thus, also known as a non-additive multiplier since it does not add an additional operand to the result of the multiplication. For an m-bit multiplicand and an n-bit multiplier, it is an m-bit by n-bit digital logic system which produces a (m + n) product.

It has the following advantages:

- The Carry save multiplier has a linear delay directly proportional to the length of the input word.
- The number of gates required to implement the Carry save array multiplier is directly proportional to the product n x m or n^2 for m = n.
- It has regular area which enhances the reproducibility of the design.

2.3 URDHVA TIRYAGBHYAM SUTRA

The Urdhva Tiryakbhyam Sutra is a vertical and crosswise multiplication algorithm. The principle is explained as follows

2.3.1 MULTIPLICATION USING URDHVA TIRYAGBHYAM SUTRA

Suppose we have to multiply (ax+b) by (cx+d). The product is $acx^2 + x(ad+bc) + bd$. This can be obtained as follows:

Step 1: The coefficient of x^2 is obtained by the vertical multiplication of a and c

Step 2: The coefficient of x is obtained by the crosswise multiplication of a and d and of b and c and the addition of the two products

Step 3: The independent term is arrived at by vertical multiplication of the absolute terms b and d.

2.3.2 EXAMPLE OF A 2X2 URDHVA MULTPLIER

Let the operands be a and b.

a_1 a_0 b_1 b_0

1 0 1 1

Let the product be p and carry be c.

a_1 a_0 1 0

x x

b_1 b_0 1 1

p_3 p_2 p_1 p_0 0110

p_0: $a_0 * b_0$ c_0 - carry

p_1: $a_0 * b_1 + a_1 * b_0 + c_0$ c_1 – carry

p_2: $a_1 * b_1 + c_1$ c_2 – carry

p_3: c_2

2.3.3 EXAMPLE OF A 4X4 URDHVA MULTIPLIER

Let the operands be a and b.

a_3 a_2 a_1 a_0 b_3 b_2 b_1 b_0

1 1 0 1 1 0 1 0

Let the product be p and carry be c

$a_3\ a_2\ a_1\ a_0$ 1 1 0 1

x x

$b_3\ b_2\ b_1\ b_0$ 1 0 1 0

$p_7\ p_6\ p_5\ p_4\ p_3\ p_2\ p_1\ p_0$ 10000010

p_0 : $a_0 * b_0$ c_0 - carry

p_1 : $a_0 * b_1 + a_1 * b_0 + c_0$ c_1 - carry

p_2 : $a_0 * b_2 + a_1 * b_1 + a_2 * b_0 + c_1$ c_2 - carry

p_3 : $a_0 * b_3 + a_1 * b_2 + a_2 * b_1 + a_3 * b_0 + c_2$ c_3 - carry

p_4 : $a_1 * b_3 + a_2 * b_2 + a_3 * b_1 + c_3$ c_4 - carry

p_5 : $a_2 * b_3 + a_3 * b_2 + c_4$ c_5 - carry

p_6 : $a_3 * b_3 + c_5$ c_6 - carry

p_7 : c_6

2.3.4 EXAMPLE OF A 8X8 URDHVA MULTIPLIER

Let the operands be a and b.

$a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0$ $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$

0 1 1 0 0 0 1 1 0 1 1 0 0 0 1 1

Let the product be p and carry be c

$a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0$ 0 1 1 0 0 0 1 1

x $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$ x 0 1 1 0 0 0 1 1

$p_7\ p_6\ p_5\ p_4\ p_3\ p_2\ p_1\ p_0\ p_7\ p_6\ p_5\ p_4\ p_3\ p_2\ p_1\ p_0$ 0010011001001001

p_0 : $a_0 * b_0$ c_0 - carry

p_1 : $a_0 * b_1 + a_1 * b_0 + c_0$ c_1 - carry

p_2 : $a_0 * b_2 + a_1 * b_1 + a_2 * b_0 + c_1$ c_2 - carry

p_3 : $a_0 * b_3 + a_1 * b_2 + a_2 * b_1 + a_3 * b_0 + c_2$ c_3 - carry

p_4 : $a_0 * b_4 + a_1 * b_3 + a_2 * b_2 + a_3 * b_1 + a_4 * b_0 + c_3$ c_4 - carry

p_5 : $a_0 * b_5 + a_1 * b_4 + a_2 * b_3 + a_3 * b_2 + a_4 * b_1 + a_5 * b_0 + c_4$ c_5 - carry

p_6 : $a_0 * b_6 + a_1 * b_5 + a_2 * b_4 + a_3 * b_3 + a_4 * b_2 + a_5 * b_1 + a_6 * b_0 + c_5$

c_6 - carry

p_7 : $a_0 * b_7 + a_1 * b_6 + a_2 * b_5 + a_3 * b_4 + a_4 * b_3 + a_5 * b_2 + a_1 * b_7 + a_7 * b_0$

$+ c_6$ c_7 -carry

p_8 : $a_1 * b_7 + a_2 * b_6 + a_3 * b_5 + a_4 * b_4 + a_5 * b_3 + a_6 * b_2 + a_7 * b_1 + c_7$

c_8 - carry

p_9 : $a_2 * b_7 + a_3 * b_6 + a_4 * b_5 + a_5 * b_4 + a_6 * b_3 + a_7 * b_2 + c_8$ c_9 - carry

p_{10}: $a_3 * b_7 + a_4 * b_6 + a_5 * b_5 + a_6 * b_4 + a_7 * b_3 + c_9$ c_{10}- carry

p_{11}: $a_4 * b_7 + a_5 * b_6 + a_6 * b_5 + a_7 * b_4 + c_{10}$ c_{11}- carry

p_{12}: $a_5 * b_7 + a_6 * b_6 + a_7 * b_5 + c_{11}$ c_{12}- carry

p_{13}: $a_6 * b_7 + a_7 * b_6 + c_{12}$ c_{13}- carry

p_{14}: $a_7 * b_7 + c_{13}$ c_{14}- carry

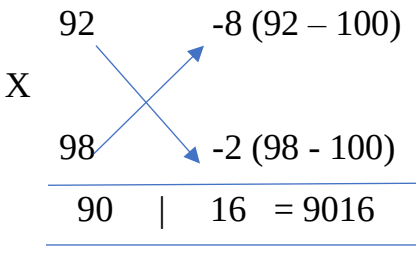
p_{15}: c_{14}

2.4 NIKHILAM NAVATASHCARAMAM DASHATAH SUTRA

Nikhilam Sutra means “all from 9 and last from 10”. Even though it is valid to all the cases of multiplication, it is more efficient when the numbers involved are large. Since it executes with the compliment of the large number from its adjacent base to perform the multiplication operation on it, complexity of the multiplication is lesser.

2.4.1 MULTIPLICATION USING NIKHILAM SUTRA

Let the operands be 92 and 98 and it’s base is 100.



The RHS of the product can be attained by just multiplying the numbers of the col 2 (-8 * -2 = 16). The LHS of the product can be found by cross subtracting the succeeding number of col 2 from the initial number of col 1 i.e., 92 - 2 = 90 or 98 - 8 = 90. Thus, the result is attained by the concatenation of RHS and LHS (Answer = 9016).

2.4.2 BARREL SHIFTER

An important part of ALU of a processor is Barrel Shifter. It is used to perform shift right, shift left, and rotational operations. It is designed using Mux trees to use in repetitive form so that minimum power is consumed by the barrel shifter. Here multiplexer is used as the basic building block for a barrel shifter. Therefore, the required number of multiplexer for any barrel shifter is calculated using following formula.

Number of Multiplexers = (No. of Bits) * logR2R (No. of Bits)

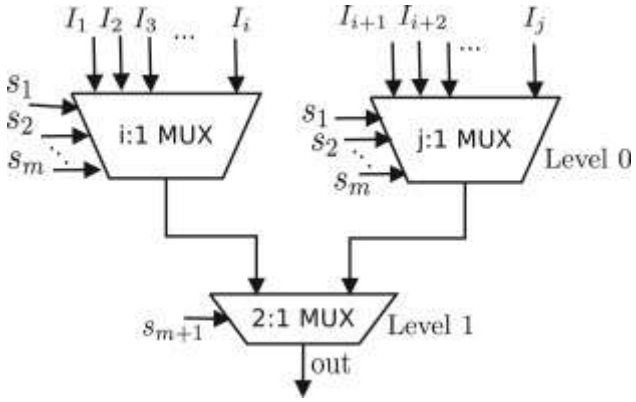


Figure 2.4.1 Block Diagram of Barrel shifter using Mux trees

CHAPTER 3

PROPOSED TECHNIQUE

From the previous chapter, it is known that the Nikhilam Sutra is used in multiplication in order to reduce the complexity of multiplier. Thus, the Nikhilam Sutra is implemented using barrel shifter in order to obtain the product in binary number system.

3.1 PROPOSED NIKHILAM SUTRA

In binary number system, generally, for a given n bit the total number of combinations is calculated using the following formula

Number of combinations = 2^n , n – number of bits.

Whereas, in decimal number system, the values of n bit ranges from **0 to (2^n) – 1.**

According to the Nikhilam Sutra, the multiplication is performed using the operands and their adjacent base which is selected such that it is the nearest power of 10. In detail, in decimal number system, when we consider the numbers from 0 to 9, they all are single digit number and they are nearest to 10 which is 10^1 or we can also say that there is only one zero in 10. When we consider the numbers from 10 to 50, they all are two-digit number and they are nearest to 10 as well as 100 (10^2) where both are powers of 10. But when we take in terms of number of zeros in 10 and 100, there is only one zero in 10 and two zeros in 100. Thus, the adjacent base is selected in such way that number of zeros in the power of 10 and number of maximum digits among the operands are equal.

The multiplication principle is same as the one explained in the previous chapter (chapter 2) and the final result is displayed in binary format during the simulation using barrel shifter.

In the proposed method, the Nikhilam Sutra is implemented in the multiplier same as mentioned in the previous chapter (chapter 2) but the product is obtained in the BCD format instead of binary number system.

3.1.1 ALGORITHM FOR A 4X4 NIKHILAM MULTIPLIER

In this multiplier, the inputs are of 4 bits. Therefore, here **n=4.** By substituting the value of n in the above mentioned formula and range we get,

Number of combinations = 2^4 = 16.

Range of values:

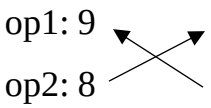
Decimal number system : **0 to 15**

Binary number system : **0000 to 1111**

As per the explanation given above, even though the numbers from 10 to 15 are of 4 bits, their adjacent base will be 100 which is a 8 bit in binary number system. Hence, the numbers from 0 to 9 only be the eligible operands for the 4x4 Nikhilam Multiplier in terms of both Nikhilam Sutra and 4 bit binary numbers.

For example,

Let the operands be op1 and op2 and base is 10. Let x and y be the difference between op1 and base and op2 and base i.e., x = op1 – base, y = op2 – base.



op1: 9 -1 :x (9-10 = -1)

op2: 8 -2 :y (8-10 = -2)

7 | 2 = 72 (final answer)

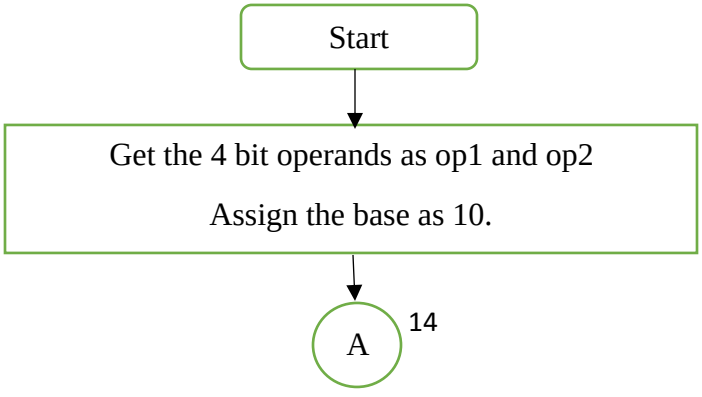
In the above example, after finding the value of x and y, the product of x and y is the ones place of the final product as 10 has only one zero (-1 * -2 = 2). If the product is a two-digit number then the tens place of that product is considered as carry and its ones place value is assigned to ones place of the final product and if the product is a single digit number then consider carry as zero and assign the single digit as ones place of the final product. For the tens place of final product, sum of the OR logic of op1+y and op2+x and carry (9-2 OR 8-1 = 7) is assigned. Thus final product is 72 (9*8= 72). Thus, the algorithm for 4x4 Nikhilam Multiplier is as follows

1. Start the program.
2. Get the two operands.
3. Assign base as 10 and store the value of two's complement of base in a temporary variable.
4. Add the temporary variable with operands to get x and y value.
5. If any one of the operands is either 10 or zero, then assign the temporary product as zero otherwise assign the product of x and y as temporary product.
6. Find the OR logic between the operands, x and y.
7. Convert the temporary product in binary format to BCD format as the code is in Verilog.
8. Assign the ones place value of temporary product to ones place value of final product.
9. Add the OR logic answer with tens place value of temporary product and the result is assigned to tens place value of final product. Thus, the final result is in BCD format.
10. Stop the program.

On comparing the delays for Nikhilam using barrel shifter multiplier (refer chapter 2) and Nikhilam in BCD format, it is observed that latter is more efficient than the former. The delay comparison table for 4x4 multipliers is provided in the chapter 5.

The algorithm of the proposed method for 4x4 Nikhilam Multiplier is shown in the form of flowchart given below.

3.1.2 FLOWCHART FOR A 4X4 NIKHILAM MULTIPLIER



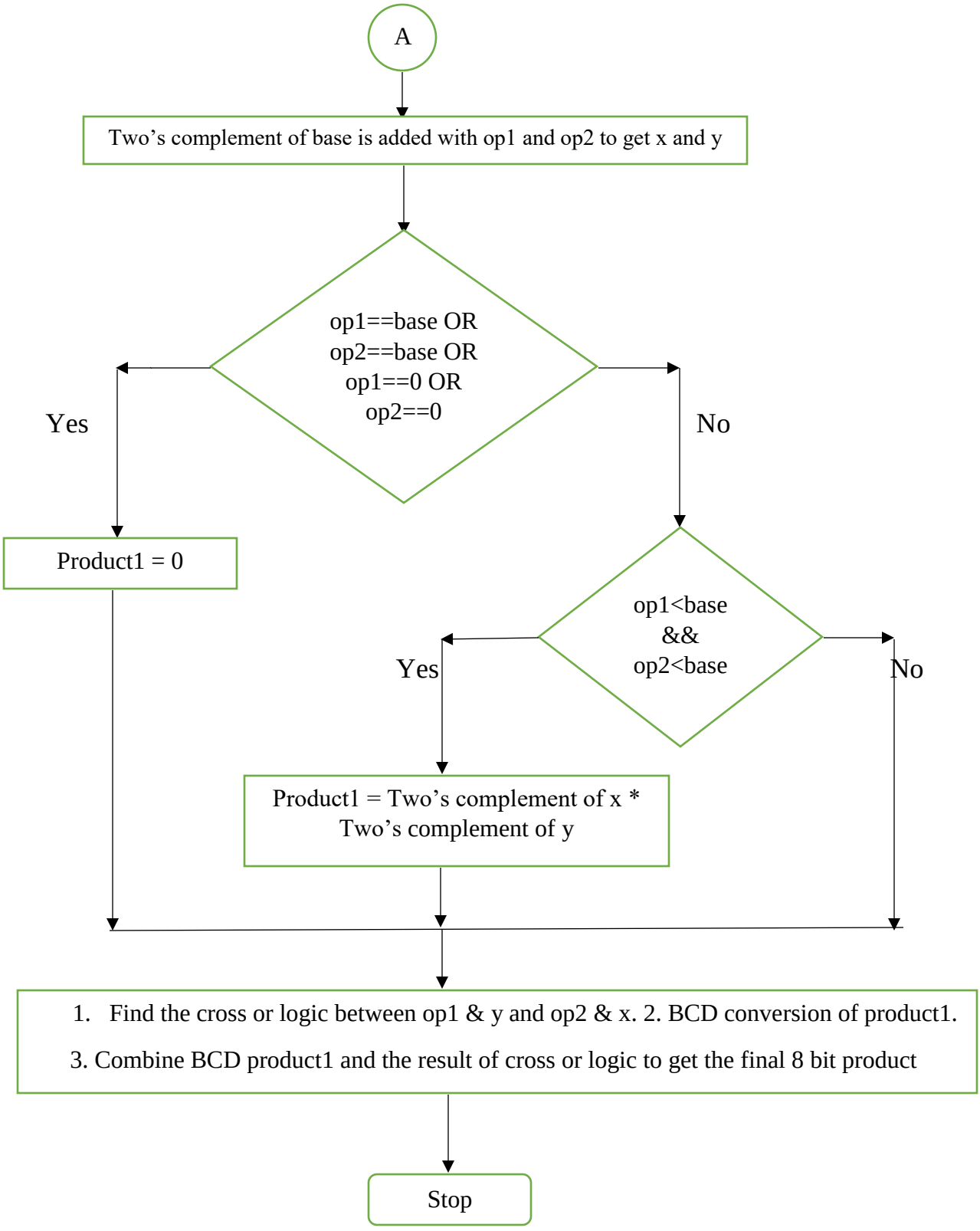


Figure 3.1.1 :Flowchart for a 4x4 Nikhilam Multiplier

3.1.3 ALGORITHM FOR A 8X8 NIKHILAM MULTIPLIER

In this multiplier, the inputs are of 8 bits. Therefore, here **n=8.** By substituting the value of n in the above mentioned formula and range we get,

Number of combinations = 2^8 = 256.

Range of values:

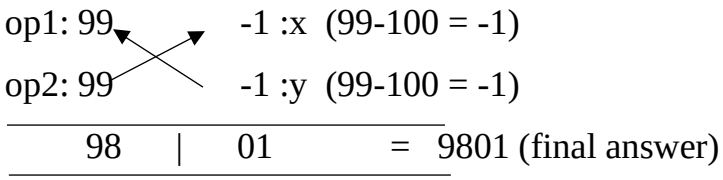
Decimal number system : **0 to 255**

Binary number system : **00000000 to 11111111**

As per the explanation given above, even though the numbers from 100 to 255 are of 8 bits, their adjacent base will be 1000 which is a 16 bit in binary number system. Hence, the numbers from 0 to 99 only be the eligible operands for the 8x8 Nikhilam Multiplier in terms of both Nikhilam Sutra and 8 bit binary numbers.

For example,

Let the operands be op1 and op2 and base is 100. Let x and y be the difference between op1 and base and op2 and base i.e., x = op1 – base, y = op2 – base.



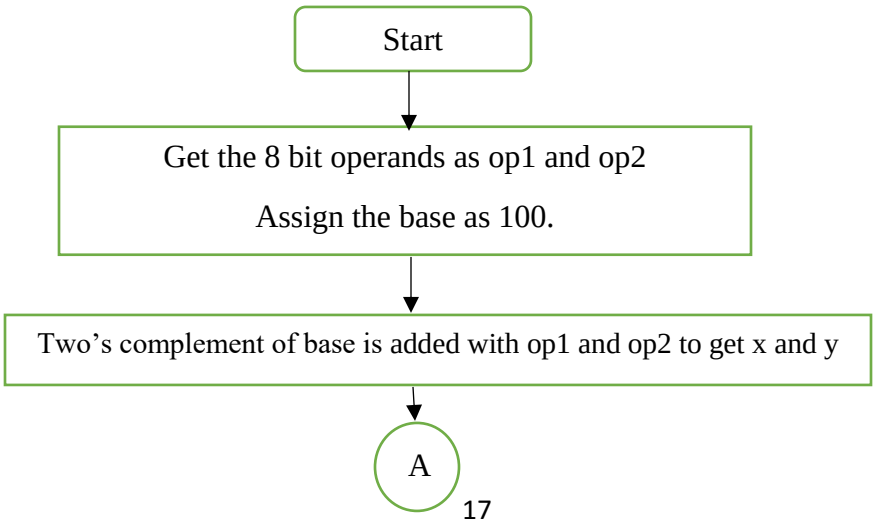
In the above example, after finding the value of x and y, the product of x and y is the tens and ones place of the final product as 100 has two zeros (-1 * -1 = 1). If the product is a three-digit number then the hundreds place of that product is considered as carry and its tens and ones place values are assigned to tens and ones place of the final product and if the product is a two-digit number then consider carry as zero and assign the two-digit as tens and ones place of the final product. For the thousands and hundreds place of final product, sum of the OR logic of op1+y and op2+x and carry is assigned. (99-1 OR 99-1 = 98). Thus the final product is 9801 (99 * 99 = 9801).

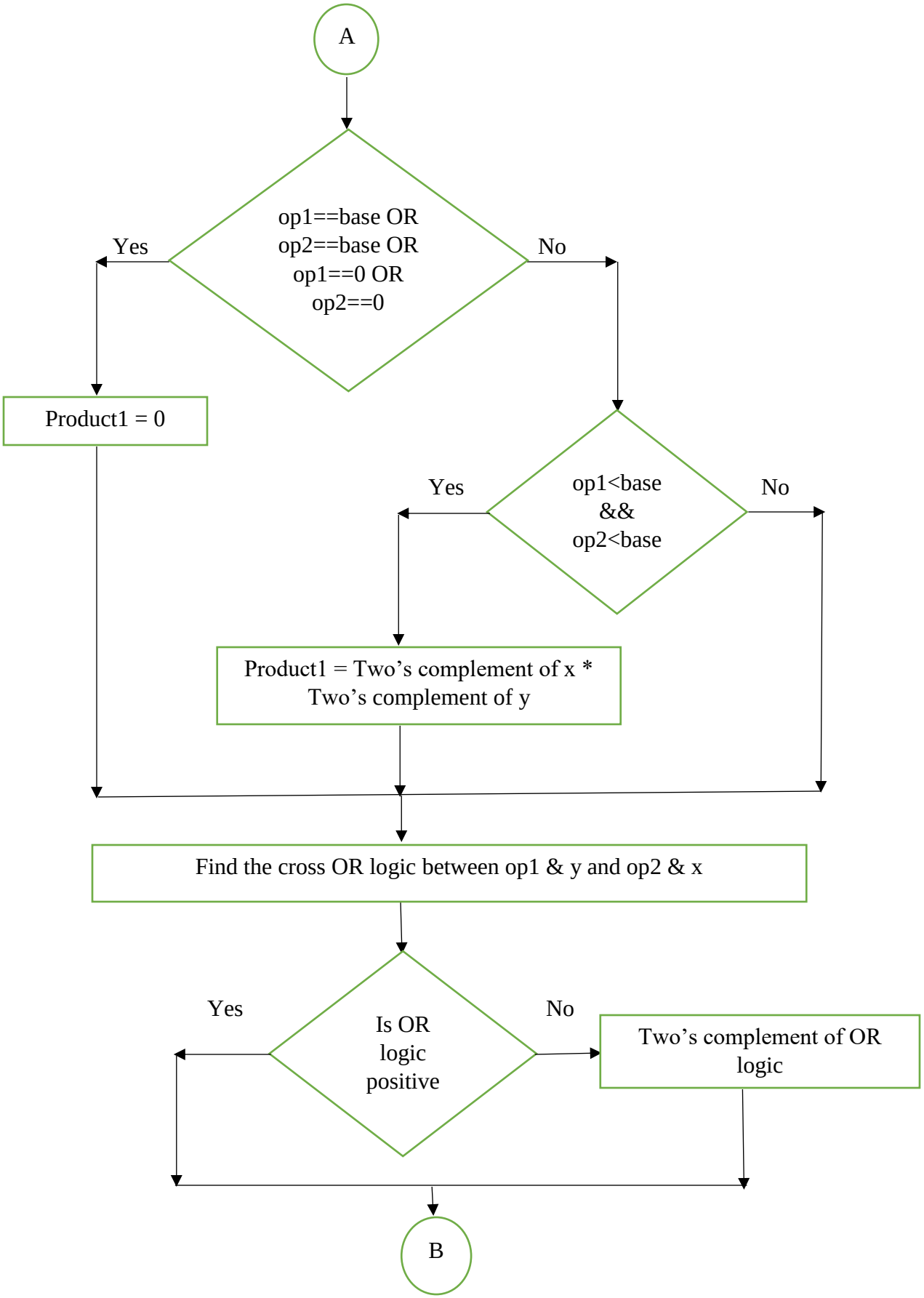
Thus, the algorithm for 4x4 Nikhilam Multiplier is as follows

1. Start the program.
2. Get the two operands.
3. Assign base as 100 and store the value of two's complement of base in a temporary variable.
4. Add the temporary variable with operands to get x and y value.
5. If any one of the operands is either 100 or zero, then assign the temporary product as zero otherwise assign the product of x and y as temporary product.
6. Find the OR logic between the operands, x and y and convert into BCD.
7. Convert the temporary product in binary format to BCD format as the code is in Verilog.
8. Assign the tens and ones place value of temporary product to tens and ones place value of final product.
9. Add the BCD OR logic answer with thousands and hundreds place value of temporary product and the result is assigned to thousands and hundreds place value of final product. Thus, the final result is in BCD format.
10. Stop the program.

The algorithm of the proposed method for 8x8 Nikhilam Multiplier is shown in the form of flowchart given below.

3.1.4 FLOWCHART FOR A 8X8 NIKHILAM MULTIPLIER




A
op1==base OR
op2==base OR
op1==0 OR
op2==0
Yes
No
Product1 = 0
op1<base
&&
op2<base
Yes
No
Product1 = Two's complement of x *
Two's complement of y
Find the cross OR logic between op1 & y and op2 & x
Is OR
logic
positive
Yes
No
Two's complement of OR
logic
B

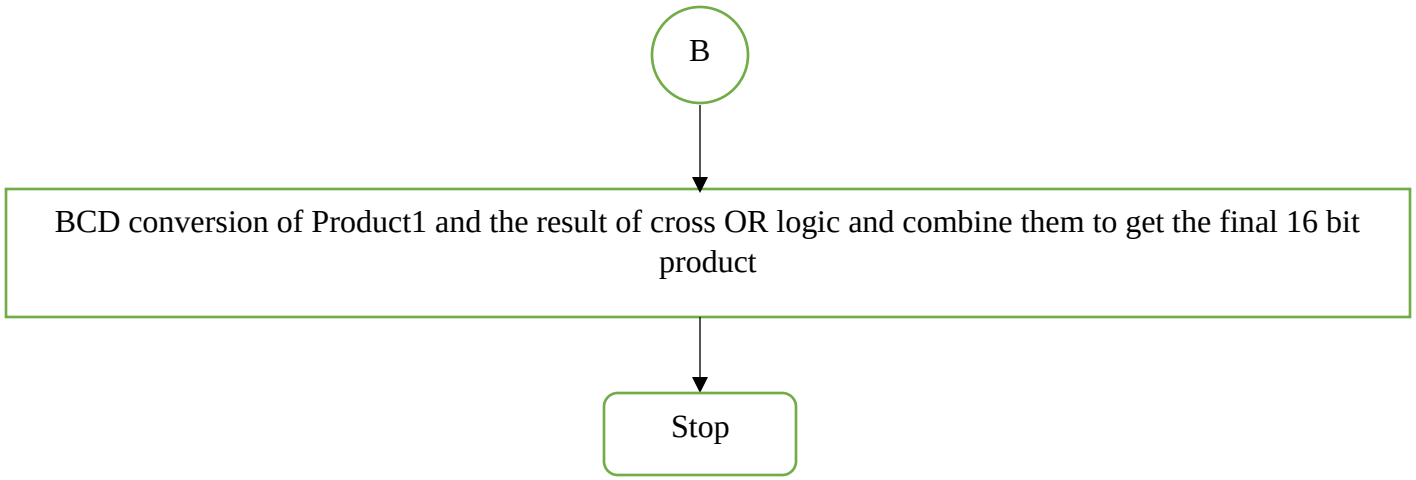


Figure 3.1.2 :Flowchart for a 8x8 Nikhilam Multiplier

CHAPTER 4

SOFTWARE DESCRIPTION

4.1. XILINX ISE

Xilinx ISE is a software tool developed by Xilinx Company. It provides the design environment for FPGA products. It is primarily used for the synthesis and analysis of HDL designs. It enables the developer to synthesis their designs and perform timing analysis and examine RTL diagrams and finally configure the target device with the programmer. Project navigator is used as the primary user interface for ISE. It includes the design hierarchy, a source code editor, transcript and processes. The design hierarchy consists of design files. The processes hierarchy describes the operations performed by the ISE on the current module. The window denotes the errors that arise with each function. The transcript window provides status of currently running operations and informs the engineers on design issues. The Xilinx runs on RHEL, SLED, FreeBSD, Microsoft windows. Its size is 6.1 gigabytes.

System-level testing can be performed with ISIM simulator. The test programs must also be written in HDL languages. These programs include simulated input signal waveforms. The ISIM performs the various type of simulations, such as logical verification; behavioral verification; post place and route simulation.

The advantage of the Xilinx is that during synthesis the Xilinx's patented algorithms allow designs to run up to 30% faster than the other programs and also allows greater logic density which reduces project time and costs.

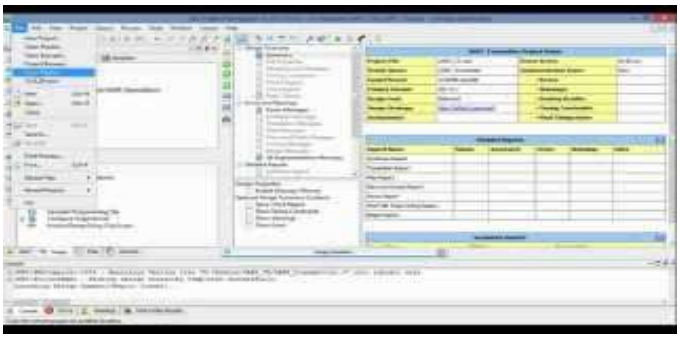


Figure 4.1.1 :Opening page of Xilinx software along with Design Summary

4.1.1 PROCEDURE TO CREATE A NEW PROJECT WITH SOURCE FILE

In the Xilinx Software, we can write a program only after creating a project under which the program will be executed. So, it is a necessity to create a project in order to execute a program in Xilinx. The procedure to create a new project is as follows

1. Click on the Xilinx icon "**ISE Design Suite 14.7**" in the Desktop.

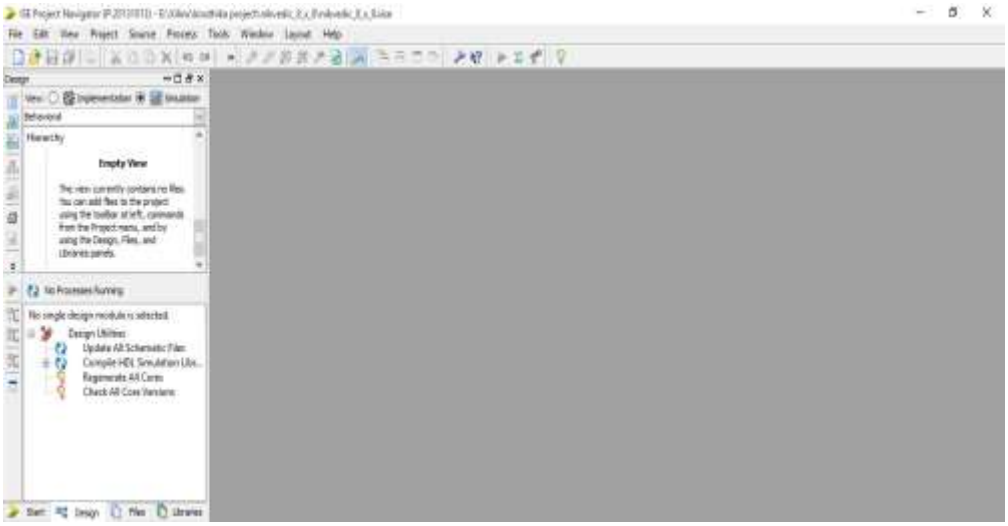


Figure 4.1.2 :Opening page of Xilinx Software without any project displayed in the window

2. After the Xilinx window opens, select **File -> New Project**. A dialog box appears. In that dialog box select the path where the project is to be saved and click **Next.**

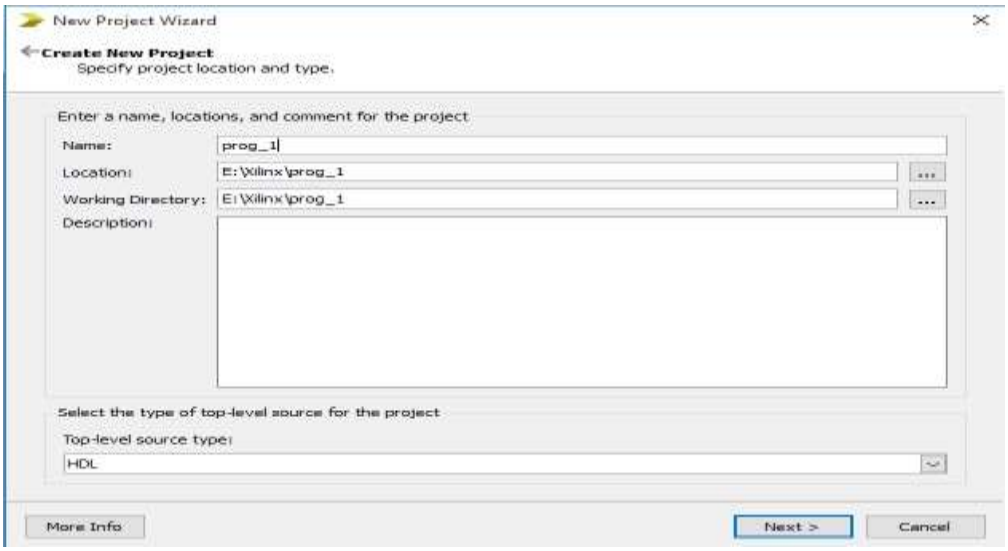


Figure 4.1.3 :Dialog Box to Create a new project.

3. Now, a dialog box will appear where the specifications of the project must be mentioned such that the written program is compatible to be executed in the hardware kit and it is executed successfully. The below picture shows the properties of the project which is to be executed in Spartan 6 FPGA kit

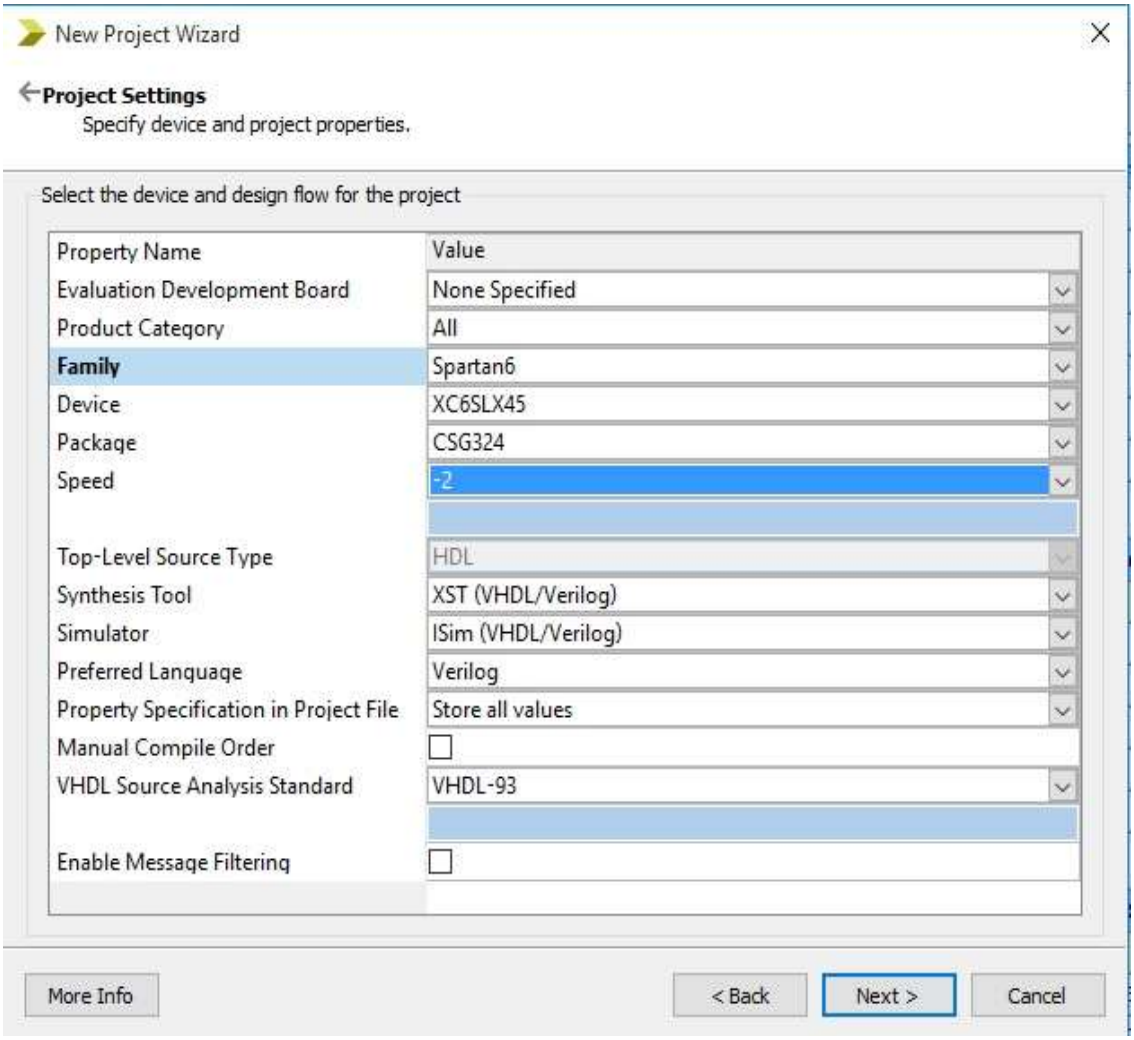


Figure 4.1.4 : Dialog Box to specify device and project properties.

After selecting the above properties, click **Next**. A dialog box will appear which shows the selected properties of project and then click Finish. Thus, a project will be created under the specified name "prog_1".

4. To create a new source module, select **Project -> New Source**. A dialog box appears as follows

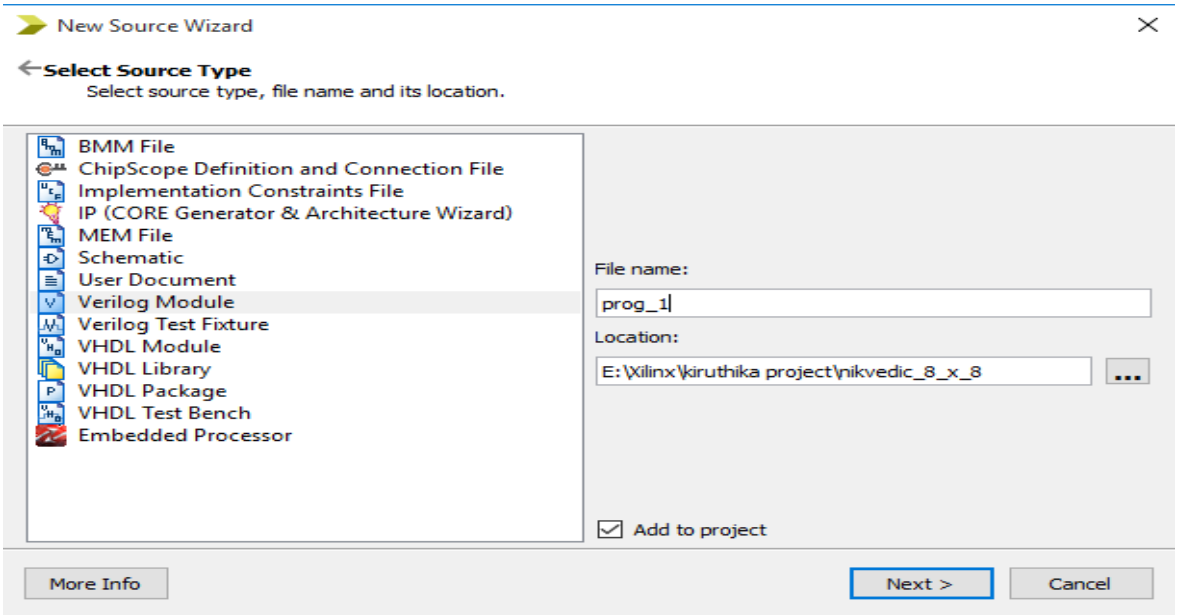


Figure 4.1.5 :Dialog Box for selecting a source file.

Once the box appears, select **Verilog Module** and give a name to the source file. Then click **Next.**

5. Specify the ports for the source file in the upcoming dialog box which is shown below

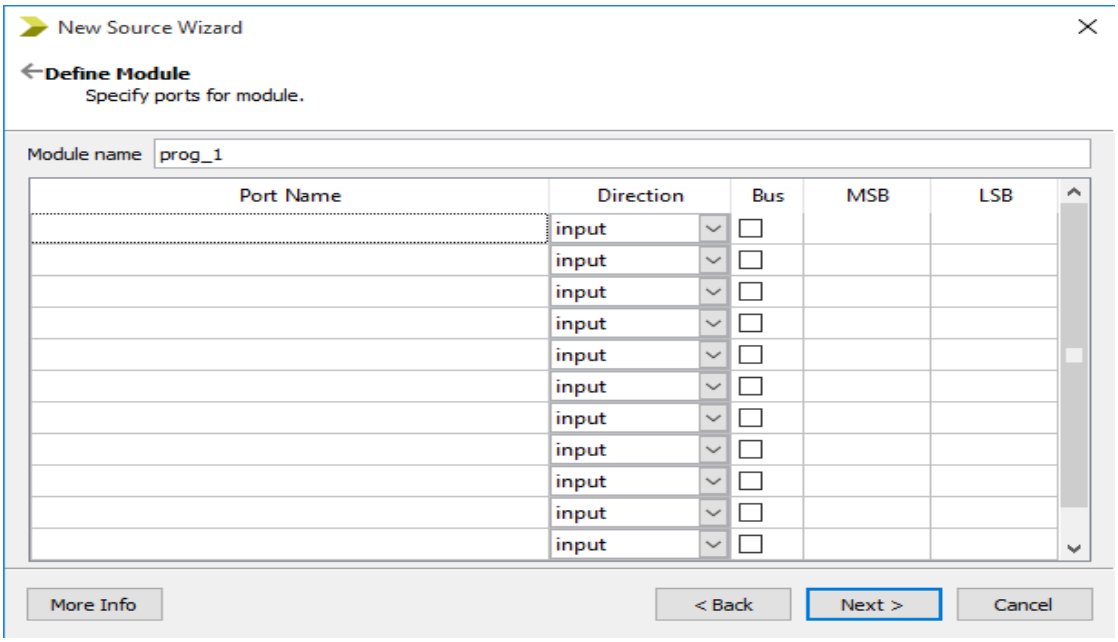


Figure 4.1.6 :Dialog Box to specify the ports for the source file

After the specifications, click **Next** then click **Finish** in the next appearing dialog box. Thus, a new source module is created under the project prog_1.

HARDWARE DESCRIPTION

4.2. SPARTAN 6 FPGA

The most cost-optimized FPGAs are Spartan 6. It offers connectivity features such as high logic-to-pin ratios, small form-factor packaging and has a diverse number of supported input and output protocols are Spartan 6. More than 40 input and output standards are used for the simplified system design. It is built on 45nm technology. The devices are ideally suited for a range of advanced bridging applications mainly found in automotive infotainment, consumer, and industrial automation. It provides the increased system performance up to 8 low power 3.2 GB/s serial transceivers. It also offers power reduction upto 1.2v core voltage.

Figure 4.2.1 : Spartan 6 FPGA Kit

4.2.1 PROCEDURE TO FLASH THE PROGRAM INTO THE FPGA KIT

After writing the program for the proposed Nikhilam Method and simulating it, the following steps are followed in order to implement the program into the FPGA Kit.

1. Synthesize the written Verilog code under Implementation View.

2. After synthesizing the code, select the **Design** tab.
3. In the Design tab, select **User Constraints -> Create Timing Constraints**.
    a. A file with extention "**program_name.ucf**" will be created.
    b. Open the ".ucf" file with text editor.
    c. Assign the inputs and outputs along with the locations of dip switches, push buttons and leds and save the file.
4. Double click on **Implement Design**.
5. After successful implementation, right click **Generate Programming File**.
    a. Select **Process Properties.**
    b. Enable the checkbox of **Create binary file configuration**.

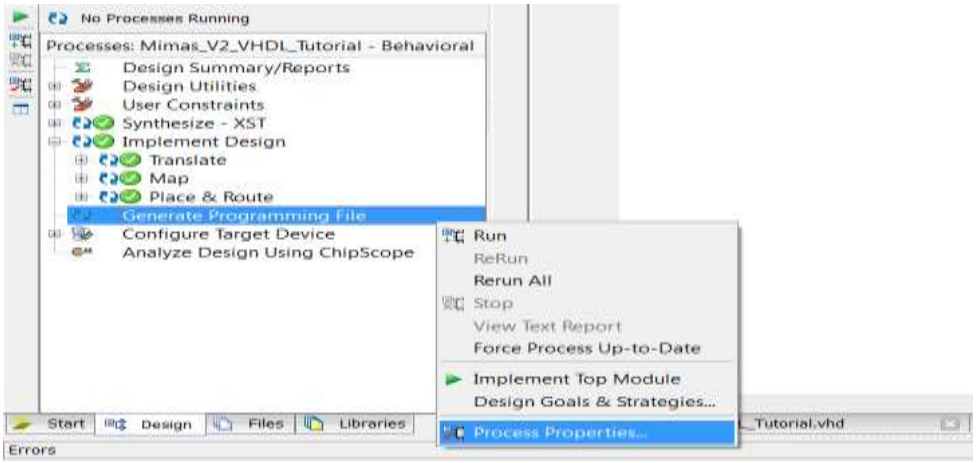


Figure 4.2.2: View of the operations present in the Design tab under Implementation View

6. Double click **Generate Programming File**.
7. After the successful generation of the binary file, connect the Spartan 6 FPGA kit to the system using a USB cable.
8. Double click on the **waxwingflashconfig** icon available in the desktop.
9. On the appearing dialog box, the connected FPGA kit must be detected.

Figure 4.2.3: Dialog Box of Waxwing Flash Config Tool showing the detection of Spartan 6 FPGA kit

10. If the kit is not detected, try the following steps :
    a. Click the **Scan for Devices** button or
    b. Connect the USB cable in other ports and reopen the window or
    c. Restart the system and repeat the steps 7, 8 and 9.

    If the kit is not detected even after following all the above steps, then must try in another system which has Xilinx ISE along with the waxwing flash software.

11. After detection of the kit, select **Load binary file -> "program_name.bin**" file from the location where the project is saved.
12. After loading the binary file, select **Program Flash**.

Figure 4.2.4: Dialog box of Waxwing Flash Config Tool after the program is flashed successfully.

13. After the successful flash of the program, press the reset button in the kit and set the inputs using dip switches and push buttons and the output is observed from the 8 leds.

CHAPTER 5

SIMULATION RESULTS

5.1 OUTPUT OF 4X4 ARRAY MULTIPLIER

The Array Multiplier is programmed using the following variables

- a – 4 bit operand 1 (array of 4)
- b – 4 bit operand 2 (array of 4)
- p – 8 bit product of operand 1 and operand 2 (array of 8)
- w – 40 bit wire value (temporary value with array of 40)

By forcing the values of operand 1 and operand 2, the value of product ‘p’ is observed in the simulator for a period of 1 second.

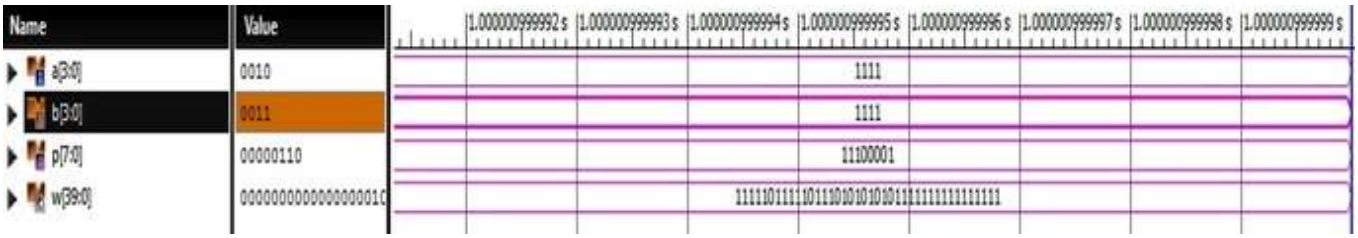


Figure 5.1: Simulation Result of a 4x4 Array Multiplier.

5.2. OUTPUT OF 4X4 CARRY – SAVE MULTIPLIER

The Carry – Save Multiplier is programmed using the following variables

- a – 4 bit operand 1 (array of 4)
- b – 4 bit operand 2 (array of 4)
- p – 8 bit product of operand 1 and operand 2 (array of 8)
- t – 23 bit temporary value (wire values with array of 23)
- pp – 16 bit partial product values (wire values with array of 16)

By forcing the values of operand 1 and operand 2, the value of product ‘p’ is observed in the simulator for a period of 1 second.

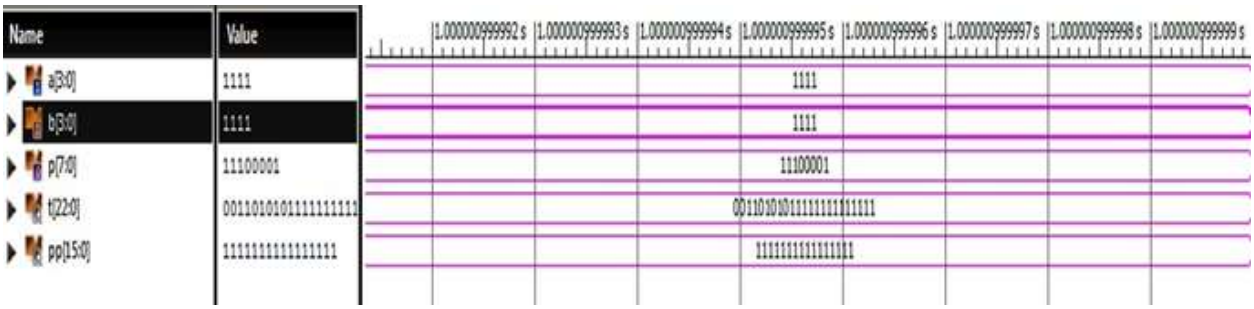


Figure 5.2: Simulation Result of a 4x4 Carry – Save Multiplier.

5.3. OUTPUT OF 2X2 URDHVA TIRYAGBHYAM VEDIC MULTIPLIER

The Urdhva Multiplier is programmed using the following variables

- a – 2 bit operand 1 (array of 2)
- b – 2 bit operand 2 (array of 2)
- c – 4 bit product of operand 1 and operand 2 (array of 4)
- temp – 4 bit temporary value(wire value with array of 4)

By forcing the values of operand 1 and operand 2, the value of product ‘c’ is observed in the simulator for a period of 1 second.

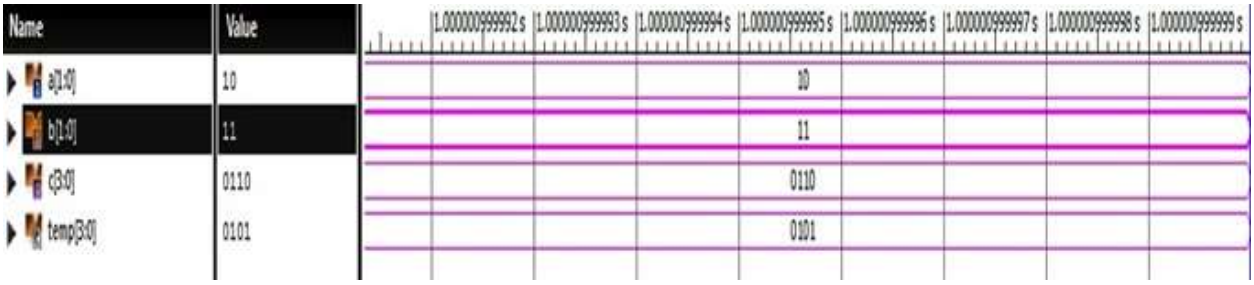


Figure 5.3: Simulation Result of a 2x2 Urdhva Vedic Multiplier.

5.4. OUTPUT OF 4X4 URDHVA TIRYAGBHYAM VEDIC MULTIPLIER

The Urdhva Multiplier is programmed using the following variables

- a – 4 bit operand 1 (array of 4)
- b – 4 bit operand 2 (array of 4)
- c – 8 bit product of operand 1 and operand 2 (array of 8)

By forcing the values of operand 1 and operand 2, the value of product ‘c’ is observed in the simulator for a period of 1 second.

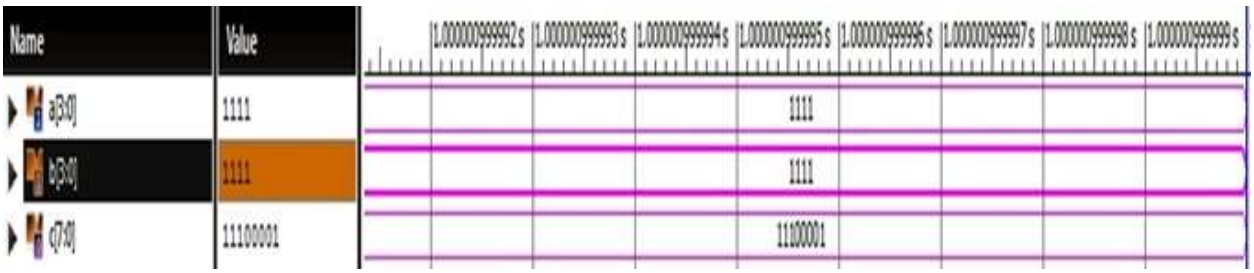


Figure 5.4: Simulation Result of a 4x4 Urdhva Vedic Multiplier.

5.5. OUTPUT OF 8X8 URDHVA TIRYAGBHYAM VEDIC MULTIPLIER

The Urdhva Multiplier is programmed using the following variables

- op1 – 8 bit operand 1 (array of 8)
- op2 – 8 bit operand 2 (array of 8)
- pdt – 16 bit product of operand 1 and operand 2 (array of 16)

By forcing the values of operand 1 and operand 2, the value of product ‘pdt’ is observed in the simulator for a period of 1 second. All the values are displayed in the format “Unsigned Decimal”.

Figure 5.5: Simulation Result of a 8x8 Urdhva Vedic Multiplier.

5.6. OUTPUT OF 4X4 NIKHILAM VEDIC MULTIPLIER USING BARREL SHIFTER

The Nikhilam Multiplier is programmed using the following variables

- A – 4 bit operand 1 (array of 4)
- B – 4 bit operand 2 (array of 4)
- prod – 8 bit product of operand 1 and operand 2 (array of 8)

By forcing the values of operand 1 and operand 2, the value of product 'prod' is observed in the simulator for a period of 1 second.

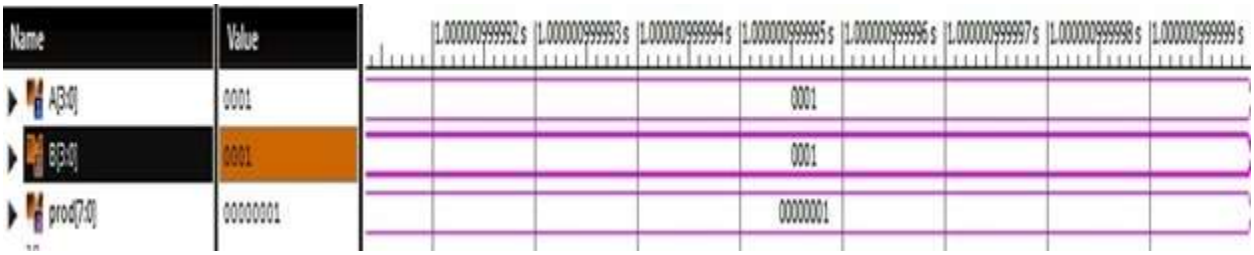


Figure 5.6: Simulation Result of a 4x4 Nikhilam Multiplier Using Barrel Shifter.

5.7. OUTPUT OF PROPOSED 4X4 NIKHILAM VEDIC MULTIPLIER

The Nikhilam Multiplier is programmed using the following variables

- op1 – 4 bit operand 1 (array of 4)
- op2 – 4 bit operand 2 (array of 4)
- x – 4 bit difference value of op1 and base (op1-base)
- y – 4 bit difference value of op2 and base (op2-base)
- base – nearest power of 10 (assigned as 10 for 4 bit input)
- tem3 – register value for base (temporary value with array of 4)
- pdt – 8 bit product of operand 1 and operand 2 (array of 8)
- pdt1 – 8 bit product of x and y (array of 8)

- temp1,temp2 – 4 bit register value for orlogic (temporary value with array of 4)
- orlog – 4 bit register value for or logic (final value with array of 4)
- T, O – BCD conversion of pdt1 (each is 4 bit)
- tem1, tem2 – register values for x and y (temporary value with array of 4)
- i – integer value for the loop (taken as 32 bit by default)

By forcing the values of operand 1 and operand 2, the value of product 'pdt' is observed in the simulator for a period of 1 second. The value of 'pdt' is displayed in the BCD format.

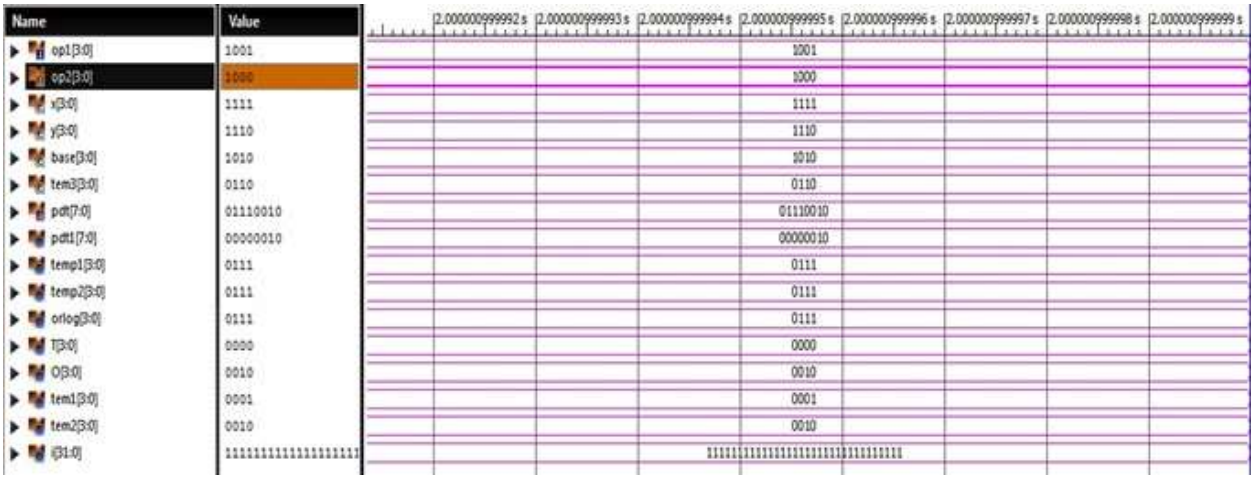


Figure 5.7: Simulation Result of a Proposed 4x4 Nikhilam Vedic Multiplier.

5.8. OUTPUT OF PROPOSED 8X8 NIKHILAM VEDIC MULTIPLIER

The Nikhilam Multiplier is programmed using the following variables

- op1 – 8 bit operand 1 (array of 8)
- op2 – 8 bit operand 2 (array of 8)
- x – 8 bit difference value of op1 and base (op1-base)
- y – 8 bit difference value of op2 and base (op2-base)
- base – nearest power of 10 (assigned as 100 for 8 bit input)
- temp – register value for base (temporary value with array of 8)

- pdt – 16 bit product of operand 1 and operand 2 (array of 16)
- pdt1 – 16 bit product of x and y (array of 16)
- temp1,temp2 – 8 bit register value for orlogic (temporary value with array of 8)
- orlog – 8 bit register value for or logic (fianl positive value with array of 4)
- temp3 – 8 bit register value for or logic (fianl negative value with array of 4)
- Th, T, O – BCD conversion of pdt1 (each is 4 bit)
- tem1, tem2 – register values for x and y (temporary value with array of 8)
- i – integer value for the loop (taken as 32 bit by default)

By forcing the values of operand 1 and operand 2, the value of product 'pdt' is observed in the simulator for a period of 1 second. The value of 'pdt' is displayed in the BCD format.

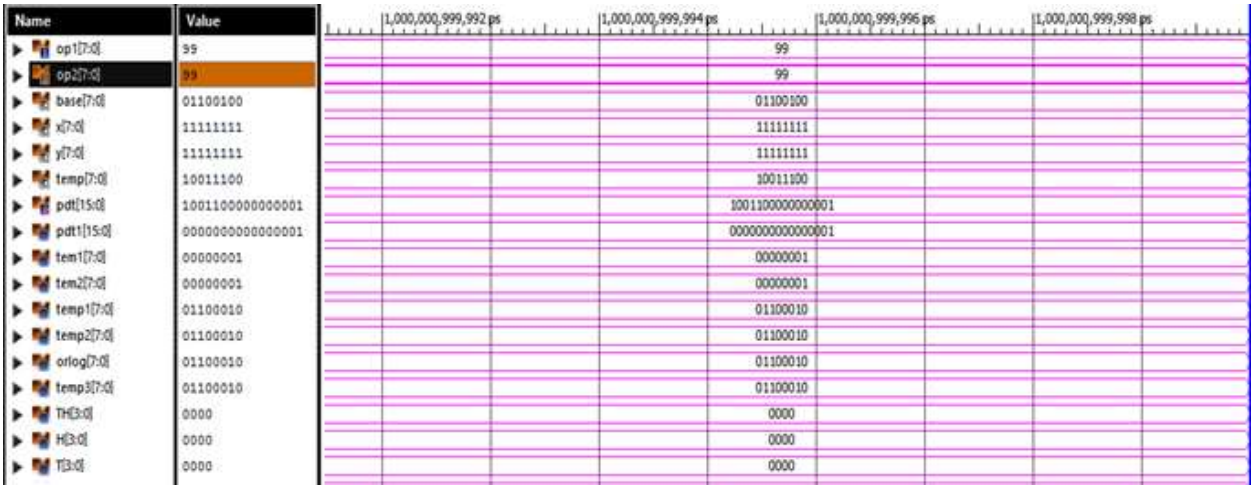


Figure 5.8: Simulation Result of a Proposed 8x8 Nikhilam Vedic Multiplier.

COMPARISON RESULTS

5.9. COMPARISON TABLE FOR 4 BIT

The terms Slices, 4 input LUTs, IOs and bonded IOBs refers to the area complexity and the terms Logic and Route refers to the time complexity of the program processed in the ISIM Simulator.

	URDHVA	**NIKHILAM**
Number of Slices	22 out of 4656	39 out of 4656
Number of 4 input LUTs	38 out of 9312	68 out of 9312
Number of IOs	16	16
Number of bonded IOBs	16 out of 158	16 out of 158
Time Taken for LOGIC	9.739 ns	8.652 ns
Time Taken for ROUTE	4.928 ns	4.126 ns
Total Time Taken	14.667 ns	12.778 ns

Table 5.1: Comparison Table for 4x4 Multipliers.

5.10. COMPARISON TABLE FOR 8 BIT

The terms Slices, 4 input LUTs, IOs and bonded IOBs refers to the area complexity and the terms Logic and Route refers to the time complexity of the program processed in the ISIM Simulator.

	URDHVA	NIKHILAM
Number of Slices	71 out of 27288	326 out of 27288
Number of LUT Flipflop-pairs used	71	326
Number of IOs	32	32
Number of bonded IOBs	32 out of 218	32 out of 218
Time Taken for LOGIC	11.310 ns	11.744 ns
Time Taken for ROUTE	31.716 ns	28.352 ns
Total Time Taken	43.026 ns	40.096 ns

Table 5.2: Comparison Table for 8x8 Multipliers.

5.11. DELAY COMPARISON TABLE FOR 4 BIT

In the below table, only the delay parameter is being compared among the three multipliers namely Urdhva, Nikhilam (using barrel shifter) and Proposed Nikhilam (using BCD format).

	URDHVA MULTIPLIER (BASIC)	NIKHILAM MULTIPLIER (USING BARREL SHIFTER)	PROPOSED NIHKILAM MULTIPLIER (USING BCD FORMAT)
DELAY (ns)	14.667	20.296	12.778

Table 5.3: Delay Comparison Table for 4x4 Multipliers.

CHAPTER 6

CONCLUSION

In the Urdhva Tiryagbyham method, the multiplication is done by cross and the results are added for the final product. In the Nikhilam method, the multiplication is done with the help of adders ,subtracters and bitwise or logic to obtain the final product. From the comparison tables results, it is observed that the proposed Nikhilam Multiplier is more efficient than the conventional Urdhva Multiplier though the area complexity of the proposed method is more. Therefore, our work has proved that the proposed vedic method "***Nikhilam Navatashcaramam Dashatah***" is one of the efficient ways for multiplication.

6.1 FUTURE WORK:

To implement the proposed Nikhilam method as a 16x16 and 32x32 multipliers along with the hardware implenmentation.

REFERENCES

1. Jagadguru Swami Sri Bharati Krsna Tirthji Maharaja," Vedic Mathematics", Motilal Banarsidas, Varanasi, India, 1986
2. Mrs. M. Ramalatha, Prof. D. Sridharan, "VLSI Based High Speed Karatsuba Multiplier for Cryptographic Applications Using Vedic Mathematics", IJSCI, 2007
3. M. Ramalatha, K. Deena Dayalan, P. Dharani, S. Deborah Priya, "High Speed Energy Efficient ALU Design using Vedic Multiplication Techniques", ACTEA 2009.
4. Thapliyal H. and Srinivas M.B. "High Speed Efficient N x N Bit Parallel Hierarchical Overlay Multiplier Architecture Based on Ancient Indian Vedic Mathematics", Transactions on Engineering, Computing and Technology, 2004, Vol.2.
5. Vikram Singh, Alok Dubey, Yogesh Khandagre, "Implementation of Vedic Math's Sutras and Barrel Shifter in Designing of Multiplier", IJISET, June 2015.