ANALYSIS, VERIFICATION AND FPGA IMPLEMENTATION OF VEDIC MULTIPLIER WITH BIST CAPABILITY

A thesis report submitted in the partial fulfillment of the

requirement for the award of the degree of

Master of Technology

in

VLSI Design & CAD

Submitted by

Vinay Kumar

Roll No.: 60761026

Under the Guidance Of

Mr. Arun Kumar Chatterjee

Lecturer, ECED

Department of Electronics and Communication Engineering

THAPAR UNIVERSITY

(Formerly Known as Thapar Institute of Engineering and Technology)

PATIALA-147004, INDIA

June – 2009

DECLARATION

I hereby declare that the work which is being presented in the thesis entitled, **"Analysis, Verification and FPGA Implementation of Vedic Multiplier with BIST Capability"** in partial fulfillment of the requirement for the award of degree of M.Tech (VLSI Design & CAD) at electronics and Communication Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Mr. Arun Kumar Chatterjee,** Lecturer, ECED.

The matter presented in this thesis has not been submitted in any other University/Institute for the award of my degree.

Date:_____ **Vinay Kumar**

Roll.No. 60761026

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Mr. Arun Kumar Chatterjee
Lecturer
ECED, Thapar University
Patiala -147004

Countersigned by:

Dr. A. K. Chatterjee
Professor & Head
Electronics and Communication Engg. Department
Thapar University,
Patiala - 147004

Dr. R. K. Sharma
Dean Academic Affairs
Thapar University
Patiala – 147004

ACKNOWLEDGEMENT

To discover, analyze and to present something new is to venture on an untrodden path towards and unexplored destination is an arduous adventure unless one gets a true torchbearer to show the way. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by various people. Words are often too less to reveals one's deep regards. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this thesis. I acknowledge with gratitude and humility my indebtedness to **Mr. Arun Kumar Chatterjee, Lecturer, ECED,** Thapar University, Patiala, under whose guidance I had the privilege to complete this thesis. I wish to express my deep gratitude towards her for providing individual guidance and support throughout the thesis work.

I convey my sincere thanks to **Dr.A.K.Chatterjee, Professor & Head of Electronics & Communication Department,** Thapar University, Patiala for his encouragement and cooperation.

I express my heartfelt gratitude towards **Mrs. Alpana Agarwal, Assistant Professor & PG coordinator, ECED,** Thapar University, Patiala, for their valuable guidance, encouragement, inspiration and the enthusiasm with which she solved my difficulties in VLSI Design & CAD Lab.

I would also like to thank all staff members and my co-students who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of my thesis.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the Almighty who bestowed self-confidence, ability and strength in me to complete this work for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

.

Vinay Kumar

ABSTRACT

This thesis work is devoted for the design of a high speed Vedic multiplier, its implementation on reconfigurable hardware and Built in Self Testing (BIST) of the implemented multiplier. Interfacing of FPGA with a PS2 KEYBOARD has also been done.

For arithmetic multiplication various Vedic multiplication techniques like Urdhva tiryakbhyam, Nikhilam and Anurupye has been thoroughly discussed. It has been found that Urdhva tiryakbhyam Sutra is 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 input of Vedic multiplier has been given by a PS2 KEYBOARD and output has been displayed on LCD of Spartan 3E kit. The synthesis results show that the computation time for calculating the product of 32x32 bits is 7.784 ns, while for the product of 64x64 bits is 10.241 ns.

Finally, the implemented design has been tested by using Built in Self Test, which shows that this Vedic multiplier is completely fault free.

TABLE OF CONTENTS

DECLERATION.....i
ACKNOWLEDGEMENT ii
ABSTRACT.....iii
TABLEOFCONTENTS.....iv
LISTOFFIGURES.....viii
LISTOFTABLES.....x
TERMINOLOGY.....xi

CHAPTER 1.....1 - 4

INTRODUCTION.....1

1.1 Objective.....3

1.2 Thesis Organization.....4

1.3 Tools Used.....4

CHAPTER 2.....5 - 16

VEDIC MULTIPLICATION ALGORITHMS.....5

2.1 History of Vedic mathematics.....5

2.2 Algorithms of Vedic mathematics.....6

2.2.1 Vedic multiplication.....6

2.2.1.1 Urdhva Tiryakbhyam Sutra.....7

2.2.1.2 Nikhilam Sutra.....12

2.2.2 Square algorithm.....13

2.2.3 Cube algorithm.....14

2.2 Performance.....15

2.3.1 Power……...15

2.3.2 Speed……...……..…...15

2.3.3 Area……...……...16

CHAPTER 3…….…….17 - 26

DESIGN AND SOFTWARE SIMULATION…….……...17

3.1 Block design of 64x64 bits Vedic multiplier…….……...17

3.2 Implementation of 2x2 bits multiplier…….…….18

3.3 Implementation of 4x4 bits Vedic multiplier……..…....19

3.4 Implementation of 8x8 bits Vedic multiplier……...21

3.5 Implementation of 16x 16 bits multiplier…….……22

3.6 Implementation of 32x32 bits Vedic multiplier…….……..23

3.7 Design implementation of 64x64 bits Vedic multiplier…….…..25

CHAPTER 4…….…….27 - 43

IMPLEMENTATION AND TESTING OF MULTIPLIER……....27

4.1 FPGA Implementation…….……27

4.1.1 Overview of FPGA……..……..27

4.1.2 Design Entry……..……29

4.1.3 Behavioral Simulation……..…….29

4.1.4 Design Synthesis……..….30

4.1.5 Design Implementation……..……...30

4.2 Implement design in FPGA…….…….32

4.3 Download design to the spartan-3e kit…….……32

4.4 Programming the FPGA…….……..32

4.4.1 LCD Interfacing Programming in FPGA……..…33

4.4.2 Keyboard Interfacing Programming in FPGA…..…33

4.5 BIST – an Overview…….……35

4.5.1 Basic BIST Architecture…….……..35

4.5.2 BIST Process……..……...36

4.5.3 BIST Implementation…….37

4.6 Designs for testability……...38

4.6.1 Pseudo Random Pattern Generator……..39

4.6.2 BIST Response Compaction…….……....40

4.6.3 Output Response Analyzer……..40

4.7 Implementation of Vedic multiplier with BIST……..41

4.7.1 BIST implementation flow……..……..42

4.7.2 Block diagram and RTL view of Vedic

Multiplier with BIST…….42

CHAPTER 5…….……..44 - 56

RESULTS AND CONCLUSION……..……44

5.1 Results …….……44

5.1.1 Simulation results of 64x64 bits Vedic

Multiplier……....44

5.1.2 Outputs of 32x32 bits display on LCD screen

of FPGA…….……47

5.1.3 BIST Results…….……49

5.1.3.1 LFSR…….…….49

5.1.3.2 Input Stimulus……51

5.1.3.3 Comparator…….53

5.2 Conclusion…….……...55

5.3 Future work….....56

REFERENCES…….……57

LIST OF FIGURES

Figure No.	Title of Figure	Page No.
Figure 2.1:	Multiplication of two decimal numbers by Urdhva Tiryakbhyam	8
Figure 2.2:	Line diagram for multiplication of two 4 - bit numbers	10
Figure 2.3:	Hardware architecture of the Urdhva tiryakbhyam multiplier	11
Figure 2.4:	Multiplication Using Nikhilam Sutra	12
Figure 3.1:	Block diagram of 64x64 Vedic Multiplier	17
Figure 3.2:	Block diagram of 2x2 Multiplier	18
Figure 3.3:	RTLView of 2x2 Bits Multiplier by ModelSim	19
Figure 3.4:	Block diagram of 4x4 Bit Vedic Multiplier	19
Figure 3.5:	Algorithm of 4x4 bit Vedic Multiplier	20
Figure 3.6:	RTL View of 4x4 Bit Vedic Multiplier by ModelSim	21
Figure 3.7:	8X8 Bits decomposed Vedic Multiplier	22
Figure 3.8:	16x16 Bits decomposed Vedic Multiplier	22
Figure 3.9:	32X32 Bits proposed Vedic Multiplier	23
Figure 3.10:	Block diagram of 32X32 Bit Vedic Multiplier	24
Figure 3.11:	RTL View of 32X32 bits Vedic Multiplier	24
Figure 3.12:	Block diagram of 64x64 bit Vedic Multiplier by ISE	25
Figure 3.13:	Proposed architecture of 64x64 bits Vedic Multiplier	25

Figure 3.14: RTL View of 64X64 bits Vedic Multiplier.....26

Figure 4.1: FPGA Design Flow.....29

Figure 4.2: (a) Design Flow during FPGA Implementation (b) Procedure Followed for Implementation…….31

Figure 4.3: Pin diagram of LCD Interfacing of 64X64Vedic Multiplier.....33

Figure 4.4: Pin diagram of Keyboard interfacing of 64X64 Vedic Multiplier.....34

Figure 4.5: Block Diagram of BIST Architecture.....35

Figure 4.6: Basic BIST Architecture.....36

Figure 4.7: BIST Hierarchy.....37

Figure 4.8: BIST architecture.....38

Figure 4.9: Basic testing flow.....39

Figure 4.10: Linear Feedback Shift Register.....40

Figure 4.11: Implementation flow of BIST…….41

Figure 4.12: Implementation of vedic multiplier using BIST.....42

Figure 4.13: RTL view of Vedic multiplier with BIST by ISE Tool.....43

Figure 5.1: Simulation result of 64x64 bits Vedic multiplier by Model Sim.....44

Figure 5.2: LCD display (output of 32X32) on Spartan 3E FPGA kit.....47

Figure 5.3: Output of 32X32 bits multiplier shown on LCD of SPARTAN kit…....48

Figure 5.4: Waveform generation (TPG) by LFSR.....49

Figure 5.5: Wave form generation for input stimulus.....51

Figure 5.6: Simulation result of (final BIST capability) comparator.....53

LIST OF TABLES

Table No.	Title of Table	Page No.
Table 4.1:	Summary of FPGA features	27
Table 4.2:	Device (XC3S500) Utilization Summary for 64x64 bits multiplier	28
Table 4.3:	Device (XC3S1600) Utilization Summary for 64x64 bits multiplier	28
Table 5.1:	Synthesis results of multipliers	46
Table 5.2:	Synthesis results of LCD display of multiplier (32X32)	49
Table 5.3:	Synthesis results of LFSR	50
Table 5.4:	Synthesis results of input stimulus	52
Table 5.5:	Synthesis results of (final BIST capability) comparator	54
Table 5.6:	Comparison of Multipliers	55

TERMINOLOGY

DSP	Digital signal Processing
ADSP	Advanced digital signal processing
XST	Xilinx synthesis technology
FPGA	Field programming gate array
ATE	Automatic test equipment
BIST	Built in self test
CUT	Circuit under test
ATPG	Automatic test pattern generator
TPG	Test pattern generator
DFT	Design for test
DFT	Discrete Fourier transforms
MAC	Multiply and Accumulate
FFT	Fast Fourier transforms
IFFT	Inverse Fast Fourier transforms
CIAF	Computation Intensive arithmetic functions
IC	Integrated Circuits
ROM	Read only memory
LFSR	Linear feedback shift register
PRPG	Pseudo random pattern generator
ORA	Output response analyzer

SR	Signature registers
PLA	Programmable logic Arrays
NGC	Native generic circuit
NGD	Native generic database
NCD	Native circuit description
UCF	User constraints file
CLB	Combinational logic blocks
IOB	Input output blocks
PAR	Place and Route
ISE	Integrated software environment
IOP	Input output pins
CPLD	Complex programmable logic device
RTL	Register transfer level
JTAG	Joint test action group
RAM	Random access memory
FDR	Fix data register
SOPC	System-on –a-Programmable-chip
ASIC	Application-specific integrated circuit
RPG	Random pattern generation

CHAPTER 1

INTRODUCTION

Multiplication is an important fundamental function in arithmetic operations. Multiplication-based operations such as Multiply and Accumulate(MAC) and inner product are among some of the frequently used Computation- Intensive Arithmetic Functions(CIAF) currently implemented in many Digital Signal Processing (DSP) applications such as convolution, Fast Fourier Transform(FFT), filtering and in microprocessors in its arithmetic and logic unit [1]. Since multiplication dominates the execution time of most DSP algorithms, so there is a need of high speed multiplier. Currently, multiplication time is still the dominant factor in determining the instruction cycle time of a DSP chip.

The demand for high speed processing has been increasing as a result of expanding computer and signal processing applications. Higher throughput arithmetic operations are important to achieve the desired performance in many real-time signal and image processing applications [2]. One of the key arithmetic operations in such applications is multiplication and the development of fast multiplier circuit has been a subject of interest over decades. Reducing the time delay and power consumption are very essential requirements for many applications [2, 3]**.** This work presents different multiplier architectures. Multiplier based on Vedic Mathematics is one of the fast and low power multiplier**.**

Minimizing power consumption for digital systems involves optimization at all levels of the design. This optimization includes the technology used to implement the digital circuits, the circuit style and topology, the architecture for implementing the circuits and at the highest level the algorithms that are being implemented. Digital multipliers are the most commonly used components in any digital circuit design. They are fast, reliable and efficient components that are utilized to implement any operation. Depending upon the arrangement of the components, there are different types of multipliers available. Particular multiplier architecture is chosen based on the application.

In many DSP algorithms, the multiplier lies in the critical delay path and ultimately determines the performance of algorithm. The speed of multiplication operation is of great importance in DSP as well as in general processor. In the past multiplication was implemented generally with a sequence of addition, subtraction and

shift operations. There have been many algorithms proposals in literature to perform multiplication, each offering different advantages and having tradeoff in terms of speed, circuit complexity, area and power consumption.

The multiplier is a fairly large block of a computing system. The amount of circuitry involved is directly proportional to the square of its resolution i.e. A multiplier of size n bits has n^2 gates. For multiplication algorithms performed in DSP applications latency and throughput are the two major concerns from delay perspective. Latency is the real delay of computing a function, a measure of how long the inputs to a device are stable is the final result available on outputs. Throughput is the measure of how many multiplications can be performed in a given period of time; multiplier is not only a high delay block but also a major source of power dissipation. That's why if one also aims to minimize power consumption, it is of great interest to reduce the delay by using various delay optimizations.

Digital multipliers are the core components of all the digital signal processors (DSPs) and the speed of the DSP is largely determined by the speed of its multipliers. Two most common multiplication algorithms followed in the digital hardware are array multiplication algorithm and Booth multiplication algorithm. The computation time taken by the array multiplier is comparatively less because the partial products are calculated independently in parallel. The delay associated with the array multiplier is the time taken by the signals to propagate through the gates that form the multiplication array. Booth multiplication is another important multiplication algorithm. Large booth arrays are required for high speed multiplication and exponential operations which in turn require large partial sum and partial carry registers. Multiplication of two *n*-bit operands using a radix-4 booth recording 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, a large propagation delay is associated with this case. Due to the importance of digital multipliers in DSP, it has always been an active area of research and a number of interesting multiplication algorithms have been reported in the literature [4].

In this thesis work, Urdhva tiryakbhyam Sutra is first applied to the binary number system and is used to develop digital multiplier architecture. This is shown to be very similar to the popular array multiplier architecture. This Sutra also shows the effectiveness of to reduce the NXN multiplier structure into an efficient 4X4 multiplier

structures. Nikhilam Sutra is then discussed and is shown to be much more efficient in the multiplication of large numbers as it reduces the multiplication of two large numbers to that of two smaller ones. The proposed multiplication algorithm is then 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 work presents a systematic design methodology for fast and area efficient digit multiplier based on Vedic mathematics .The Multiplier Architecture is based on the Vertical and Crosswise algorithm of ancient Indian Vedic Mathematics [5].

This work presents methodology to detect and locate faults using Built in Self Test (BIST) and other various Designs for Testability (DFT) methods. It also introduces Automatic Test Pattern Generation for maximum fault coverage. The task of determining whether the chip is working properly or not is very tedious. However, if chip is not properly fabricated, they can cause system failure and result in heavy loss in economy. System failure results difficulty in debugging. The debugging cost grows exponentially as we move from chip to board level and then toward the system level. As the number of transistor integrated in the chip increases, task to test the functionality of chip become more and more difficult. To overcome these design issues, Design for Testability has become more important.

Automatic Test Pattern Generation is a process of generating patterns to test a circuit which is described strictly with a logic level net list. They can generate circuit test vectors to detect faults in the circuit. They can find redundant or unnecessary circuit logic and they can prove whether one circuit implementation matches the circuit implementation [6, 7].

1.1 OBJECTIVE

The main objective of this thesis is to design and implementation of a fast multiplier with self testing, which can be used in any processor application. This thesis deals with the study, design and implementation of Vedic multiplier with BIST technique. In this thesis work, study of Vedic multiplication, square and cube algorithms has been explored. Architecture of Vedic multiplier based on speed specification is designed here. Hardware Implementation of this multiplier has been done on Spartan 3E Board. Then, finally BIST technique is used for testing of this multiplier.

1.2 THESIS ORGANIZATION

The basic concept of multiplications, architectures and functionality of Vedic multiplier has been discussed in **chapter 2.**

Chapter 3 describes the design and software implementation of different modules of the Vedic multiplier. These modules are coded using simulation software ISE (Integrated Software Environment) of Xilinx and ModelSim.

In **chapter 4** hardware implementation of Vedic multiplier on FPGA using Xilinx tool has been discussed. Also design flow of BIST is explained.

Results obtained from realization of Vedic multiplier on FPGA kit, in terms of speed, area and power has been shown in **chapter 5.** Also results of Built In Self Test has been observed. A conclusion has been made by these results and future scope of the thesis work has been discussed.

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 MATHEMATICS:-

Vedic mathematics is part of four Vedas (books of wisdom). It is part of Sthapatya- Veda (book on civil engineering and architecture), which is an upa-veda (supplement) of Atharva Veda. It covers explanation of several modern mathematical terms including arithmetic, geometry (plane, co-ordinate), trigonometry, quadratic equations, factorization and even calculus.

His Holiness Jagadguru Shankaracharya Bharati Krishna Teerthaji Maharaja (1884-1960) comprised all this work together and gave its mathematical explanation while discussing it for various applications. Swahiji constructed 16 sutras (formulae) and 16 Upa sutras (sub formulae) after extensive research in Atharva Veda. Obviously these formulae are not to be found in present text of Atharva Veda because these formulae were constructed by Swamiji himself. Vedic mathematics is not only a mathematical wonder but also it is logical. That's why VM has such a degree of eminence which cannot be disapproved. Due these phenomenal characteristic, VM has already crossed the boundaries of India and has become a leading topic of research abroad. VM deals with several basic as well as complex mathematical operations. Especially, methods of basic arithmetic are extremely simple and powerful [4, 9].

The word 'Vedic' is derived from the word 'veda' which means the store-house of all knowledge. Vedic mathematics is mainly 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 noncompletion.

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.

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

The beauty of Vedic mathematics lies in the fact that it reduces the otherwise cumbersome-looking calculations in conventional mathematics to a very simple one. This is so because the Vedic formulae are claimed to be based on the natural principles on which the human mind works. This is a very interesting field and presents some effective algorithms which can be applied to various branches of engineering such as computing and digital signal processing [8, 9].

The multiplier architecture can be generally classified into three categories. First is the serial multiplier which emphasizes on hardware and minimum amount of chip area. Second is parallel multiplier (array and tree) which carries out high speed mathematical operations. But the drawback is the relatively larger chip area consumption. Third is serial- parallel multiplier which serves as a good trade-off between the times consuming serial multiplier and the area consuming parallel multipliers.

2.2 ALGORITHMS OF VEDIC MATHEMATICS:-

2.2.1 VEDIC MULTIPLICATION

The proposed Vedic multiplier is based on the Vedic multiplication formulae (Sutras). These Sutras have been traditionally used for the multiplication of two numbers in the decimal number system. In this work, we apply the same ideas to the binary

number system to make the proposed algorithm compatible with the digital hardware. Vedic multiplication based on some algorithms, some are discussed below:

2.2.1.1 Urdhva Tiryakbhyam sutra

The multiplier is based on an algorithm Urdhva Tiryakbhyam (Vertical & Crosswise) of ancient Indian Vedic Mathematics. Urdhva Tiryakbhyam Sutra is a general multiplication formula applicable to all cases of multiplication. It literally means "Vertically and crosswise". It is based on a novel concept through which the generation of all partial products can be done with the concurrent addition of these partial products. The parallelism in generation of partial products and their summation is obtained using Urdhava Triyakbhyam explained in fig 2.1. The algorithm can be generalized for n x n bit number. Since the partial products and their sums are calculated in parallel, the multiplier is independent of the clock frequency of the processor. Thus the multiplier will require the same amount of time to calculate the product and hence is independent of the clock frequency. The net advantage is that it reduces the need 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. By adopting the Vedic multiplier, microprocessors designers can easily circumvent these problems to avoid catastrophic device failures. The processing power of multiplier can easily be increased by increasing the input and output data bus widths since it has a quite a regular structure. Due to its regular structure, it can be easily layout in a silicon chip. The Multiplier has the advantage that as the number of bits increases, gate delay and area increases very slowly as compared to other multipliers. Therefore it is time, space and power efficient. It is demonstrated that this architecture is quite efficient in terms of silicon area/speed [10, 4].

1) Multiplication of two decimal numbers- 325*738

To illustrate this multiplication scheme, let us consider the multiplication of two decimal numbers (325 * 738). Line diagram for the multiplication is shown in Fig.2.2. The digits on the both sides of the line are multiplied and added with the carry from the previous step. This generates one of the bits of the result and a carry. This carry is added in the next step and hence the process goes on. If more than one line are there in one step, all the results are added to the previous carry. In each step, least significant bit acts as the

result bit and all other bits act as carry for the next step. Initially the carry is taken to be zero. To make the methodology more clear, an alternate illustration is given with the help of line diagrams in figure 2.2 where the dots represent bit '0' or '1'[4].

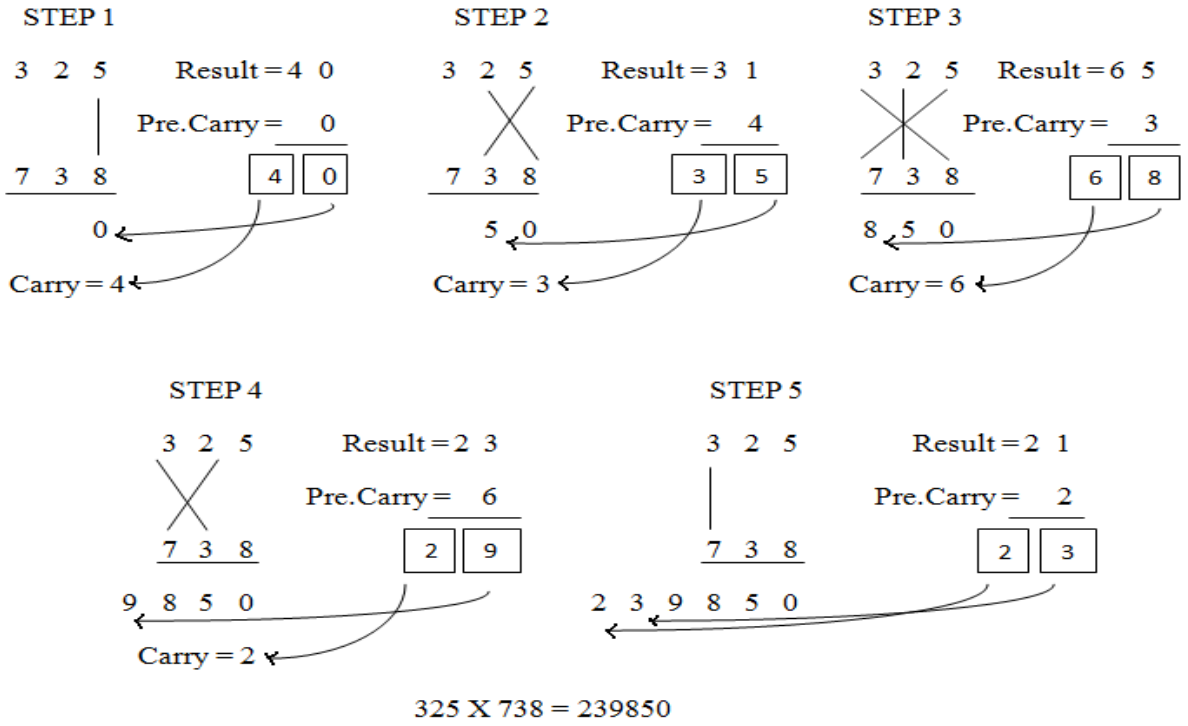


Figure 2.1: Multiplication of two decimal numbers by Urdhva Tiryakbhyam.

2) Algorithm for 4 x 4 bit Vedic multiplier Using Urdhva Tiryakbhyam (Vertically and crosswise) for two Binary numbers [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 8 X 8 Bit Multiplication Using Urdhva Triyakbhyam (Vertically and crosswise) for two Binary numbers [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 illustrate the multiplication algorithm, let us consider the multiplication of two binary numbers a3a2a1a0 and b3b2b1b0. As the result of this multiplication would be more than 4 bits, we express it as... r3r2r1r0. Line diagram for multiplication of two 4-bit numbers is shown in Fig. 2.2 which is nothing but the mapping of the Fig.2.1 in binary system. For the simplicity, each bit is represented by a circle. Least significant bit r0 is obtained by multiplying the least significant bits of the multiplicand and the multiplier. The process is followed according to the steps shown in Fig. 2.1.

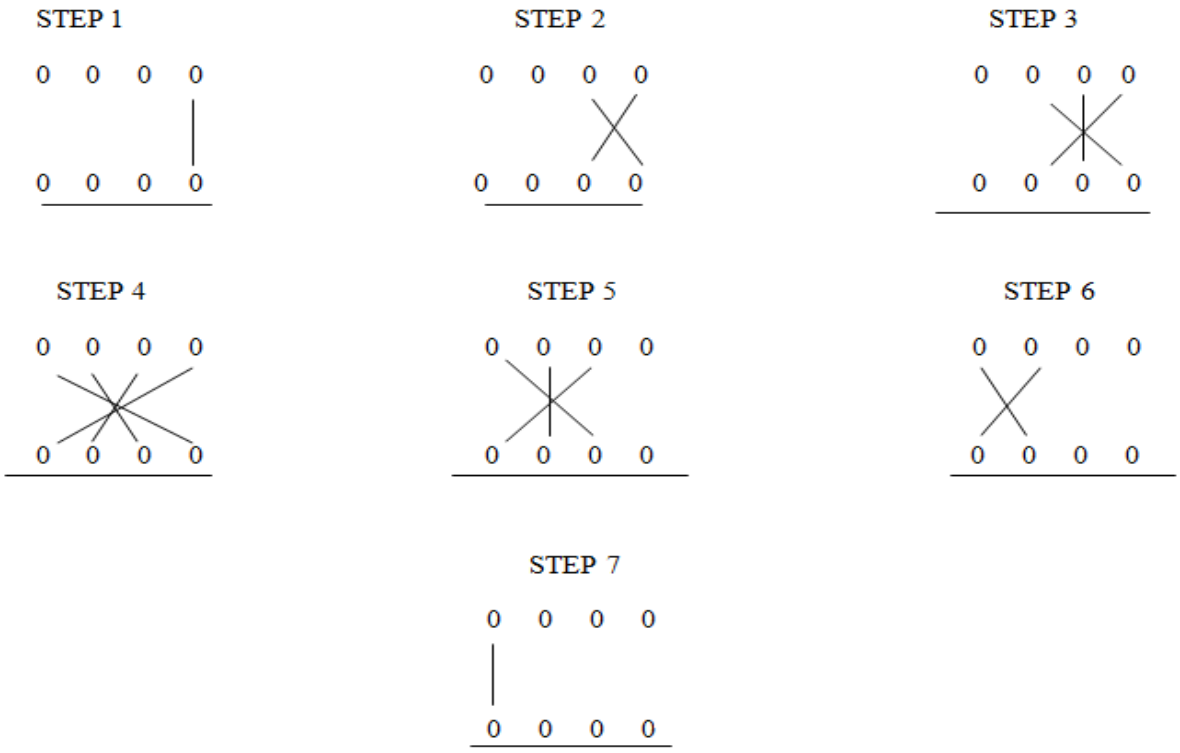


Figure 2.2: Line diagram for multiplication of two 4 - bit numbers.

Firstly, least significant bits are multiplied which gives the least significant bit of the product (vertical). Then, the LSB of the multiplicand is multiplied with the next higher bit of the multiplier and added with the product of LSB of multiplier and next higher bit of the multiplicand (crosswise). The sum gives second bit of the product and the carry is added in the output of next stage sum obtained by the crosswise and vertical multiplication and addition of three bits of the two numbers from least significant position. Next, all the four bits are processed with crosswise multiplication and addition to give the sum and carry. The sum is the corresponding bit of the product and the carry is

again added to the next stage multiplication and addition of three bits except the LSB. The same operation continues until the multiplication of the two MSBs to give the MSB of the product. For example, if in some intermediate step, we get 110, then 0 will act as result bit (referred as rn) and 11 as the carry (referred as cn). It should be clearly noted that cn may be a multi-bit number.

Thus we get the following expressions:

$r0=a0b0;$ (1)

$c1r1=a1b0+a0b1;$ (2)

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

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

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

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

$c6r6=c5+a3b3$ (7)

With c6r6r5r4r3r2r1r0 being the final product. Hence this is the general mathematical formula applicable to all cases of multiplication.

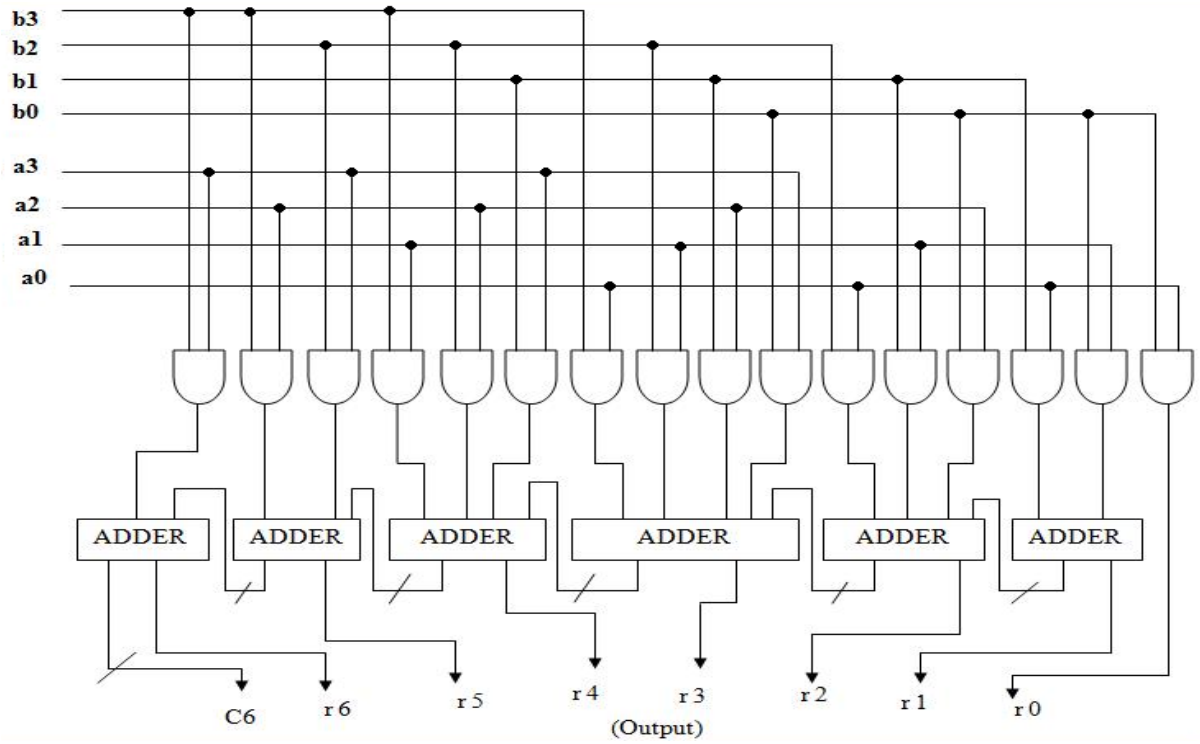


Figure 2.3: Hardware architecture of the Urdhva tiryakbhyam multiplier [4]

The hardware realization of a 4-bit multiplier is shown in figure2.3.

This hardware design 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 are calculated in parallel and the delay associated is mainly the time taken by the carry to propagate through the adders which form the multiplication array. Clearly, this is not an efficient algorithm for the multiplication of large numbers as a lot of propagation delay is involved in such cases. To deal with this problem, we now discuss Nikhilam Sutra which presents an efficient method of multiplying two large numbers [4].

2.2.1.2 Nikhilam Sutra

Nikhilam Sutra literally means "all from 9 and last from 10". Although it is applicable to all cases of multiplication, it is more efficient when the numbers involved are large. Since it finds out the compliment of the large number from its nearest base to perform the multiplication operation on it, larger is the original number, lesser the complexity of the multiplication. We first illustrate this Sutra by considering the multiplication of two decimal numbers (96 * 93) where the chosen base is 100 which is nearest to and greater than both these two numbers.

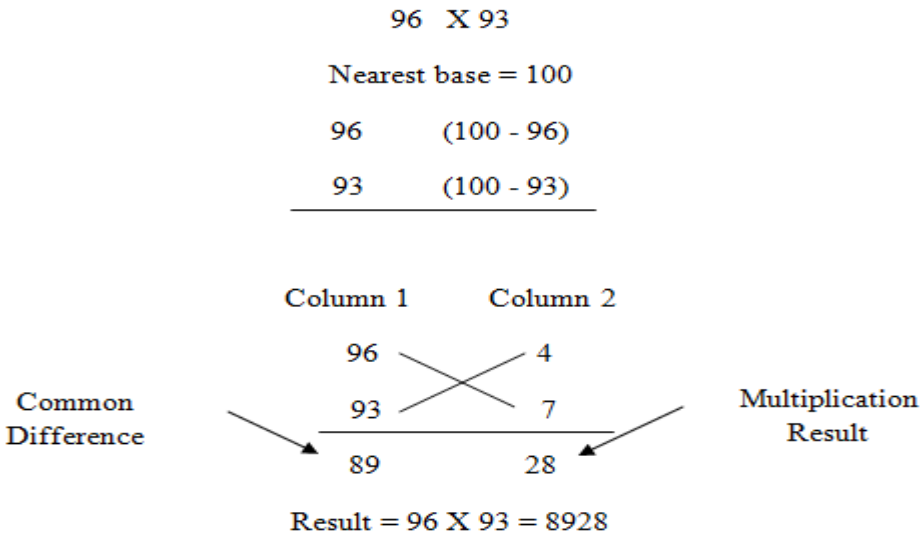


Figure 2.4: Multiplication Using Nikhilam Sutra [4]

The right hand side (RHS) of the product can be obtained by simply multiplying the numbers of the Column 2 (7*4 = 28). The left hand side (LHS) of the product 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. The final result is obtained by concatenating RHS and LHS (Answer = 8928) [4].

2.2.2 SQUARE ALGORITHM

In order to calculate the square of a number, we have utilized "Duplex" D property of Urdhva Triyakbhyam. In the Duplex, we take twice the product of the outermost pair and then add twice the product of the next outermost pair and so on till no pairs are left. When there are odd numbers of bits in the original sequence, there is one bit left by itself in the middle and this enters as its square. Thus for 987654321,

D= 2 * (9 * 1) + 2 * (8 * 2) + 2 * (7 * 3) + 2 * (6 * 4) + 5 * 5 = 165.
Further, the Duplex can be explained as follows

For a 1 bit number D is its square.

For a 2 bit number D is twice their product

For a 3 bit number D is twice the product of the outer pair + square of the middle bit.

For a 4 bit number D is twice the product of the outer pair + twice the product of the inner pair.

The algorithm is explained for 4 x 4 bit number. The Vedic square has all the advantages as it is quite faster and smaller than the array, Booth and Vedic multiplier [10, 13].

a) Algorithm for 4 x 4 bit Square Using Urdhva Tiryakbhyam D - Duplex [10, 13]

	X3	X2	X1	X0	Multiplicand
	X3	X2	X1	X0	Multiplier

H	G	F	E	D	C	B	A	

P7	P6	P5	P4	P3	P2	P1	P0	Product

PARALLEL COMPUTATION

1. D = X 0 * X0 = A
2. D = 2 * X1 * X0 = B

3. D = 2 * X2 * X0 + X1 * X1 = C

4. D = 2 * X3 * X0 + 2 * X2 * X1 = D

5 D = 2 * X3 * X1 + X 2 * X2 = E

6. D = 2 * X3 * X2 = F

7. D = X3 * X3 = G

2.2.3 CUBE ALGORITHM

The cube of the number is based on the Anurupye Sutra of Vedic Mathematics which states "If you start with the cube of the first digit and take the next three numbers (in the top row) in a Geometrical Proportion (in the ratio of the original digits themselves) then you will find that the 4th figure (on the right end) is just the cube of the second digit".

If a and b are two digits then according to Anurupye Sutra,

a^3 a^2b ab^2 b^3

$2a^2b$ $2ab^2$

$a^3 + 3a^2b + 3ab^2 + b^3 = (a + b)^3$

This sutra has been utilized in this work to find the cube of a number. The number M of N bits having its cube to be calculated is divided in two partitions of N/2 bits, say a and b, and then the Anurupye Sutra is applied to find the cube of the number. In the above algebraic explanation of the Anurupye Sutra, we have seen that a^3 and b^3 are to be calculated in the final computation of $(a+b)^3$.

The intermediate a^3 and b^3 can be calculated by recursively applying Anurupye sutra. A few illustrations of working of Anurupye sutra are given below [10].

$(15)^3 =$ 1 5 25 125

10 50

---

3 3 7 5

2.3 PERFORMANCE

2.3.1 POWER:

Vedic Multiplier has less number of gates required for given 8x8 bits Multiplier so its power dissipation is very small as compared to other multiplier architecture. Vedic Multiplier has less switching activity as compared to other architecture for same operation.

The improvements are explained in the following steps 1-5.

1. In parallel computation B of (a) square algo, the term X1*Y0+X0*Y1 of (2) multiplier algo, that is, two 4x4 multiply operations are reduced to 2* X1*X0, that is, one 4x4 multiply operation (and multiply by 2 requires only shift operation).
2. In parallel computation C of (a) square algo, the term X2 * Y0+X0*Y2 + X1 * Y1 of (2) multiplier algo, that is, three 4x4 multiply operations are reduced to 2* X2 * X0+X1 * X1, that is, only two 4x4 bit operations.
3. In parallel computation D of (a) square algo, the term X3 * Y0+X0* Y3+ X2 * Y1 +X1 * Y2 of (2) multiplier algo, that is, 4 multiply operations of 4x4 bit are reduced to 2 * X3 * X0+2 * X2 *X1, that is, only two multiply operations.
4. In parallel computation E of (a) square algo, the term X3 * Y1 +X1 * Y3+X2 * Y2 of (2) multiplier algo, that is, three 4x4 multiply operations are reduced to 2 * X3 * X1 + X2 * Y2, that is, only two multiply operations.
5. In parallel computation F of (a) square algo, the term X3*Y2+X2*Y3 of (2) multiplier algo, that is, two 4x4 bit multiply operations are reduced to 2 *X3 * X2, that is, only one multiply operation [10].

2.3.2 SPEED:

Vedic multiplier is faster than array multiplier and Booth multiplier. As the number of bits increases from 8x8 bits to 16x16 bits, the timing delay is greatly reduced for Vedic multiplier as compared to other multipliers. Vedic multiplier has the greatest advantage as compared to other multipliers over gate delays and regularity of structures. 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 multiplier shows the highest speed among conventional multipliers. It has this advantage than others to prefer a best multiplier.

2.3.3 AREA:

The area needed for Vedic square multiplier is very small as compared to other multiplier architectures i.e. the number of devices used in Vedic square multiplier are 259 while Booth and Array Multiplier is 592 and 495 respectively for 16 x 16 bit number when implemented on Spartan FPGA [12].

Thus the result shows that the Vedic square multiplier is smallest and the fastest 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 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 designing of 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 BLOCK DESIGN OF 64X64 BITS VEDIC MULTIPLIER

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

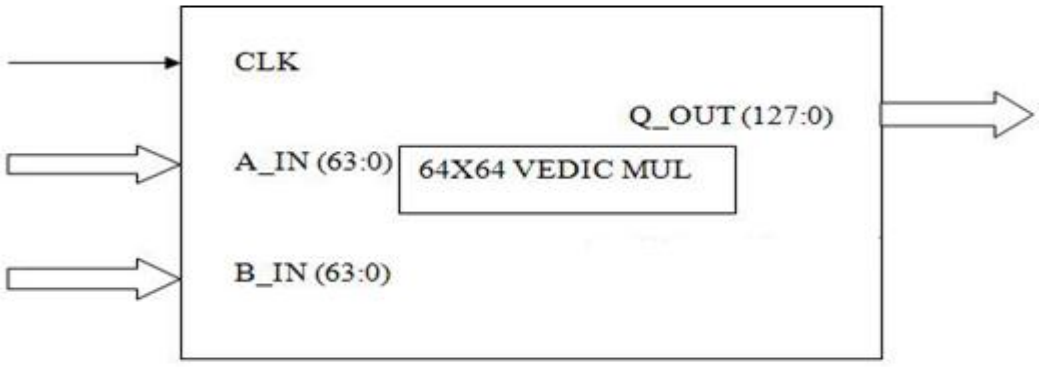


Figure 3.1: Block diagram of 64x64 Vedic Multiplier

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 Vedic Multiplier.
2) B_IN (63:0) : It is the second input of Vedic Multiplier.
3) CLK : It is the input clock.
4) Q_OUT (127:0) : It is the output register of Vedic Multiplier.

This code was designed using synchronous resets, for use in FPGA's. 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 2X2 BITS VEDIC MULTIPLIER

It is clear that the basic building blocks of this multiplier are one bit multipliers and 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.

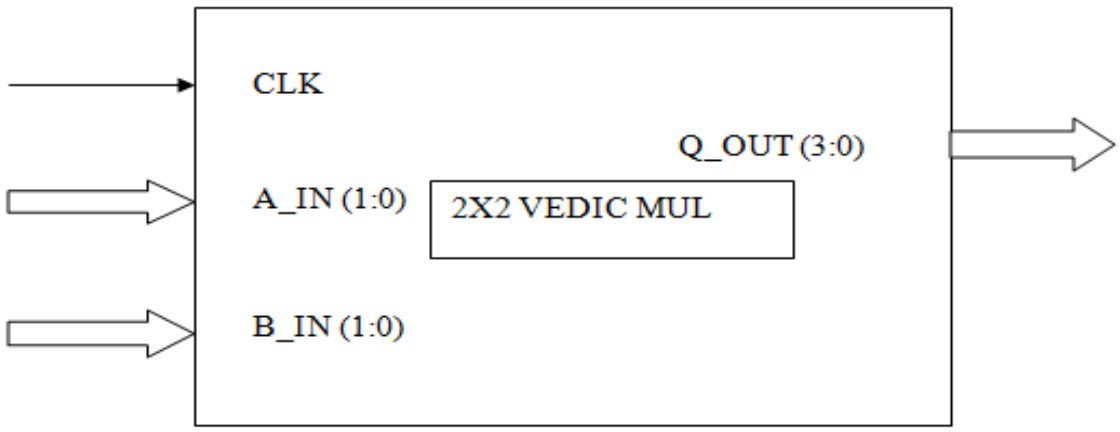


Figure 3.2: Block diagram of 2x2 Multiplier.

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

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

In Vedic method, Q0 is vertical product of bit A0 and B0, Q1 is addition of crosswise bit multiplication i.e. A1 & B0 and A0 and B1, and Q2 is again vertical product of bits A1 and B1 with the carry generated, if any, from the previous addition during Q1. Q3 output

is nothing but carry generated during Q2 calculation. This module is known as 2x2 multiplier block [5].

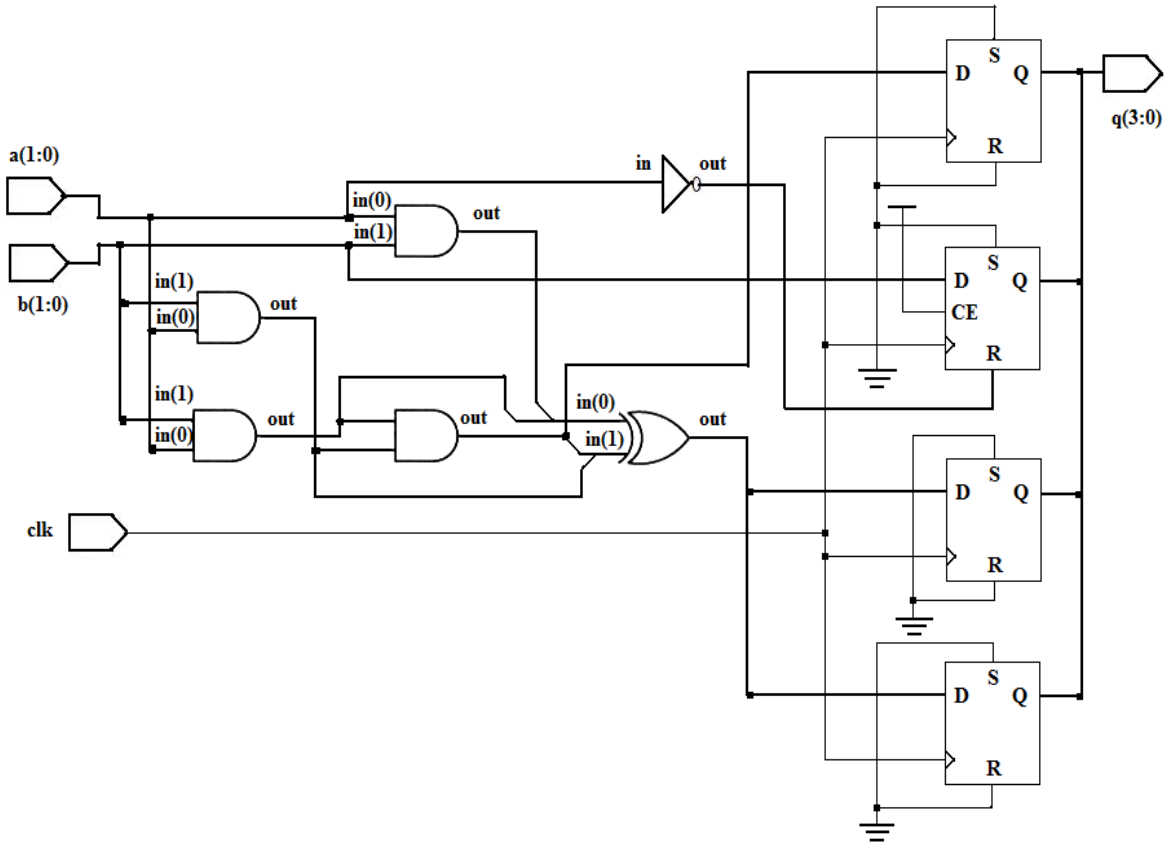


Figure 3.3: RTLView of 2x2 Bits Multiplier by ModelSim

3.3 IMPLEMENTATION OF 4X4 BITS VEDIC MULTIPLIER

For higher no. of bits in input, little modification is required. Divide the no. of bit

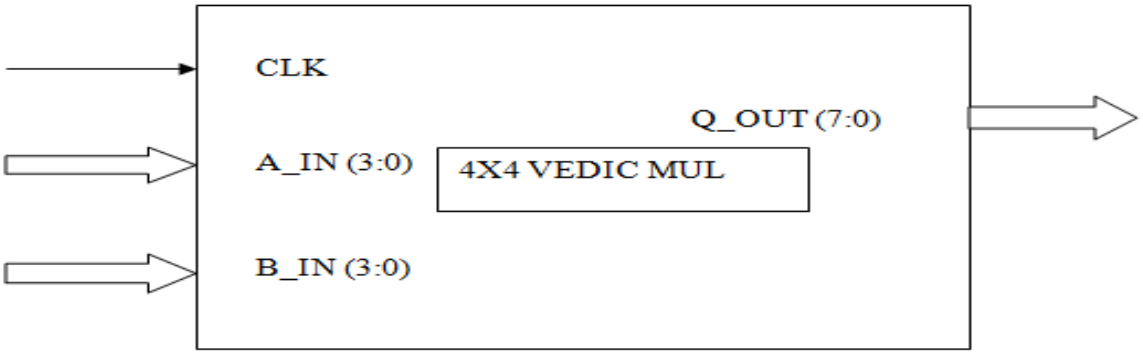


Figure 3.4: Block diagram of 4x4 Bit Vedic Multiplier

in the inputs equally in two parts.

Let's analyze 4x4 multiplications, say A3A2A1A0 and B3B2B1B0. Following are the output line for the multiplication result, Q7Q6Q5Q4Q3Q2Q1Q0. Block diagram of 4x4 Vedic Multiplier is given in fig 3.4.

Let's divide A and B into two parts, say A3 A2 & A1 A0 for A and B3B2 & B1B0 for B. Using the fundamental of Vedic multiplication, taking two bit at a time and using 2 bit multiplier block,

A3A2 A1A0

X B3B2 B1B0

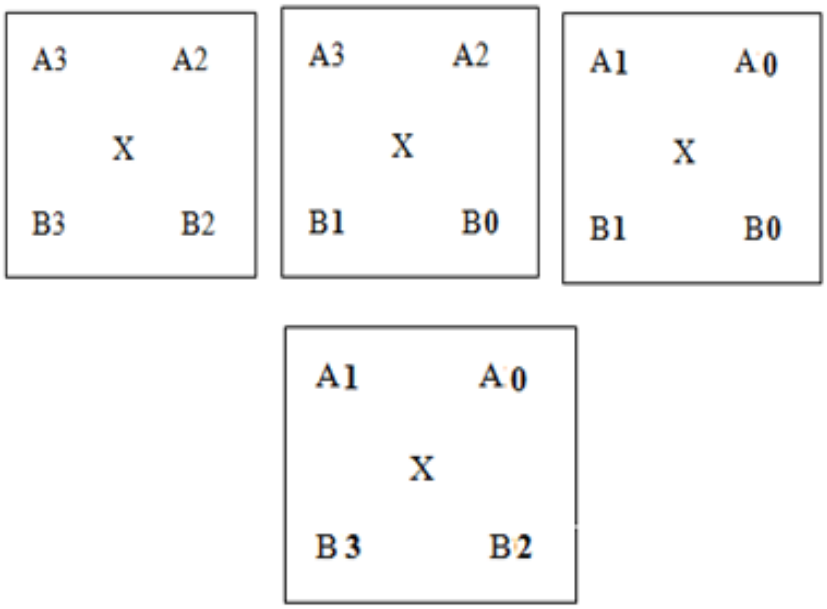


Figure 3.5: Algorithm of 4x4 bit Vedic Multiplier [5].

Each block as shown above is 2x2 bits multiplier. First 2x2 multiplier inputs are A1 A0 and B1 B0.The last block is 2x2 multiplier 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, Q7Q6Q5Q4Q3Q2Q1Q0 [5].

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

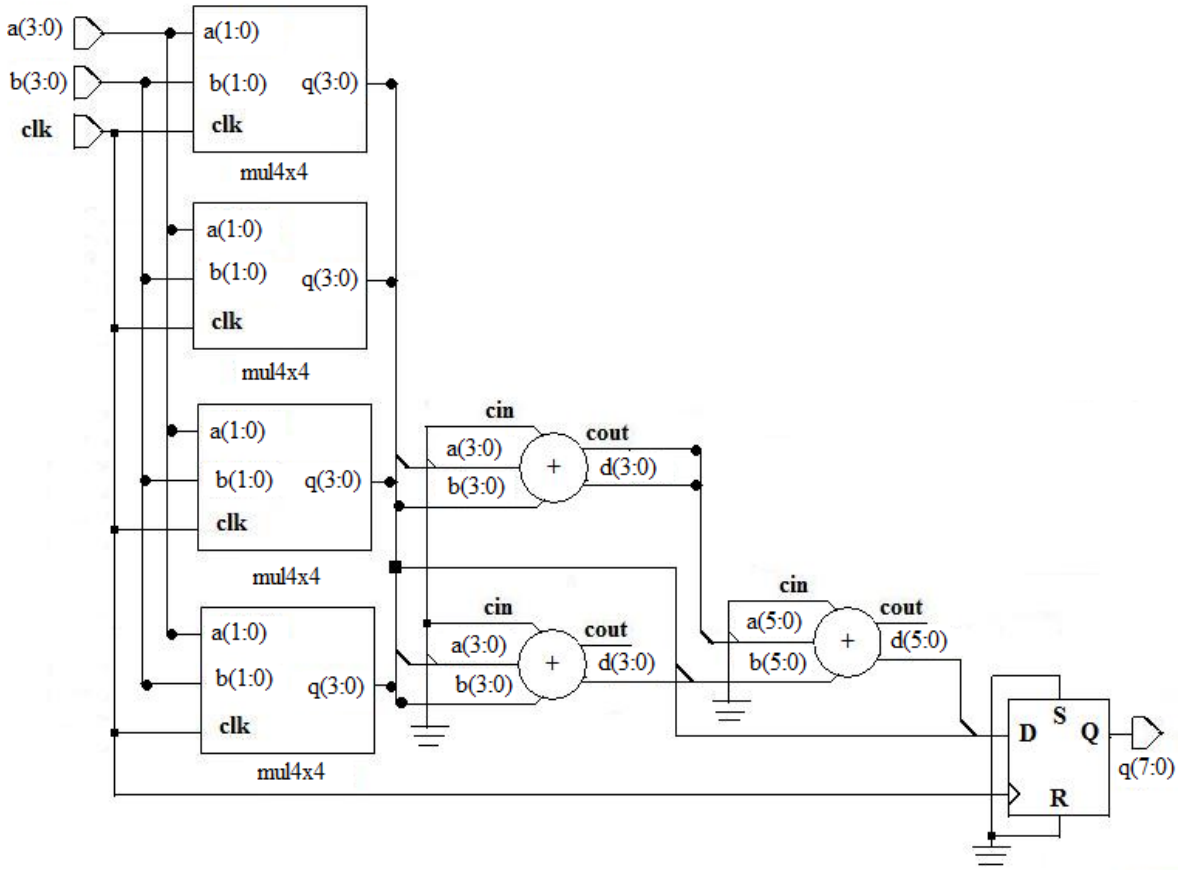


Figure 3.6: RTL View of 4x4 Bit Vedic Multiplier by ModelSim

3.4 IMPLEMENTATION OF 8X8 BITS VEDIC MULTIPLIER

The 8x 8 bit multiplier is structured using 4X4 bit blocks as shown in figure 3.7. In this figure the 8 bit multiplicand A can be decomposed into pair of 4 bits AH-AL. Similarly multiplicand B can be decomposed into BH-BL. The 16 bit product can be written as:

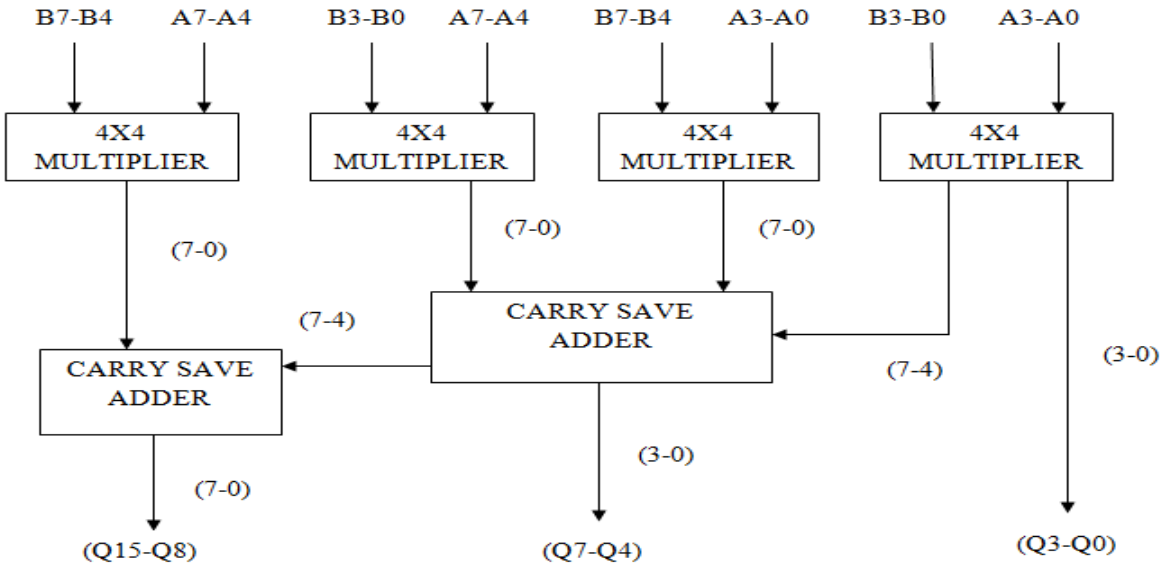
P= A x B= (AH-AL) x (BH-BL)

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

The outputs of 4X4 bit multipliers are added accordingly to obtain the final product. Thus, in the final stage two adders are also required [12].

Now the basic building block of 8x8 bits Vedic multiplier is 4x4 bits multiplier which implemented in its structural model. For bigger multiplier implementation like 8x8 bits multiplier the 4x4 bits multiplier units has been used as components which are

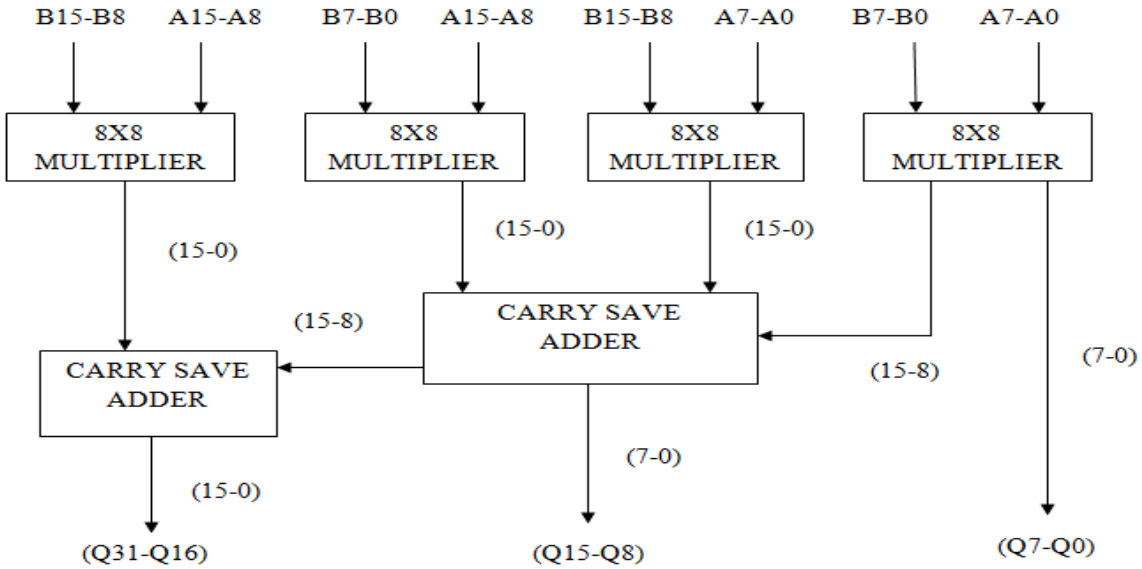
implemented already in ModelSIm6.1e or Xilinx ISE9.2i library. The structural modelling of any design shows fastest design.



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

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

3.5 IMPLEMENTATION OF 16X 16 BITS VEDIC MULTIPLIER



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

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

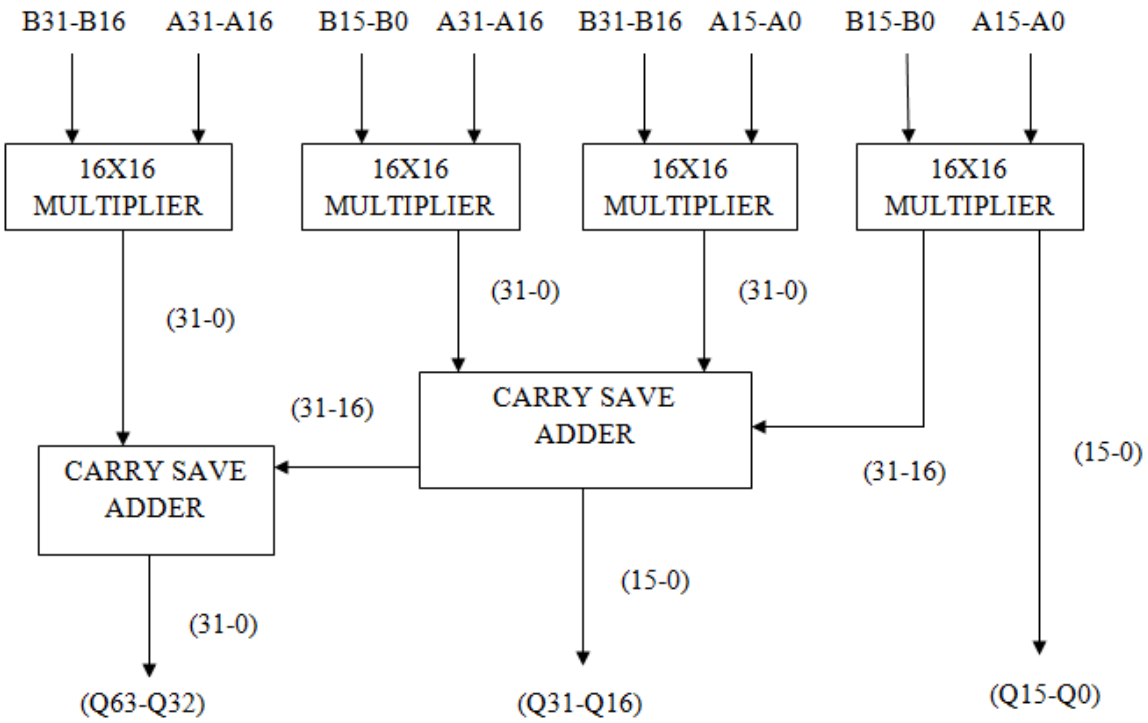
The 16X16 bit multiplier structured using 8X8 bits blocks as shown in Figure 3.8. In this Figure 3.8 the 16 bit multiplicand A can be decomposed into pair of 8 bits AH-AL. Similarly multiplicand B can be decomposed into BH-BL.

The outputs of 8X8 bit multipliers are added accordingly to obtain the 32 bits final product. Thus, in the final stage two adders are also required [12].

Similarly, we have extended same for input bits 32, 64.

3.6 IMPLEMENTATION OF 32X32 BITS VEDIC MULTIPLIER

The 32 bits multiplicand A is decomposed into pair of 16 bits AH-AL. Similarly multiplicand B can be decomposed into BH-BL. Architecture of 32x32 bits Vedic Multiplier is shown as:



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

Figure 3.9: 32X32 Bits proposed Vedic Multiplier.

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

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

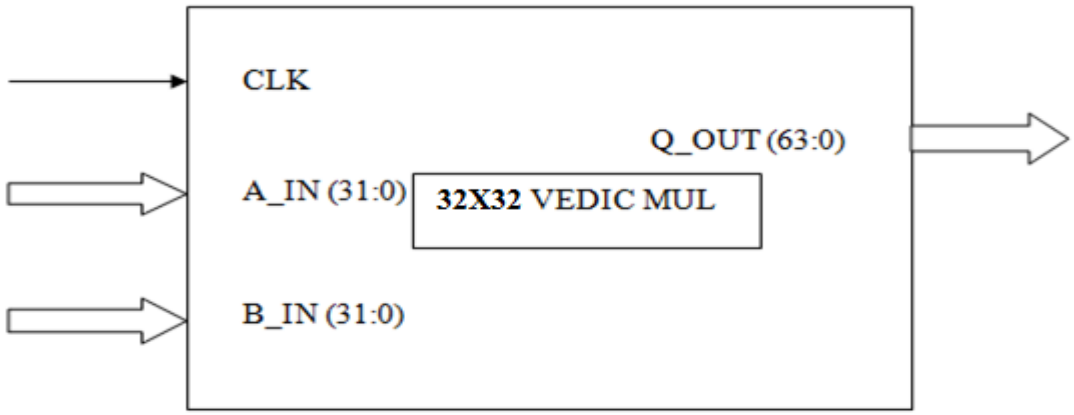


Figure 3.10: Block diagram of 32X32 Bit Vedic Multiplier

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

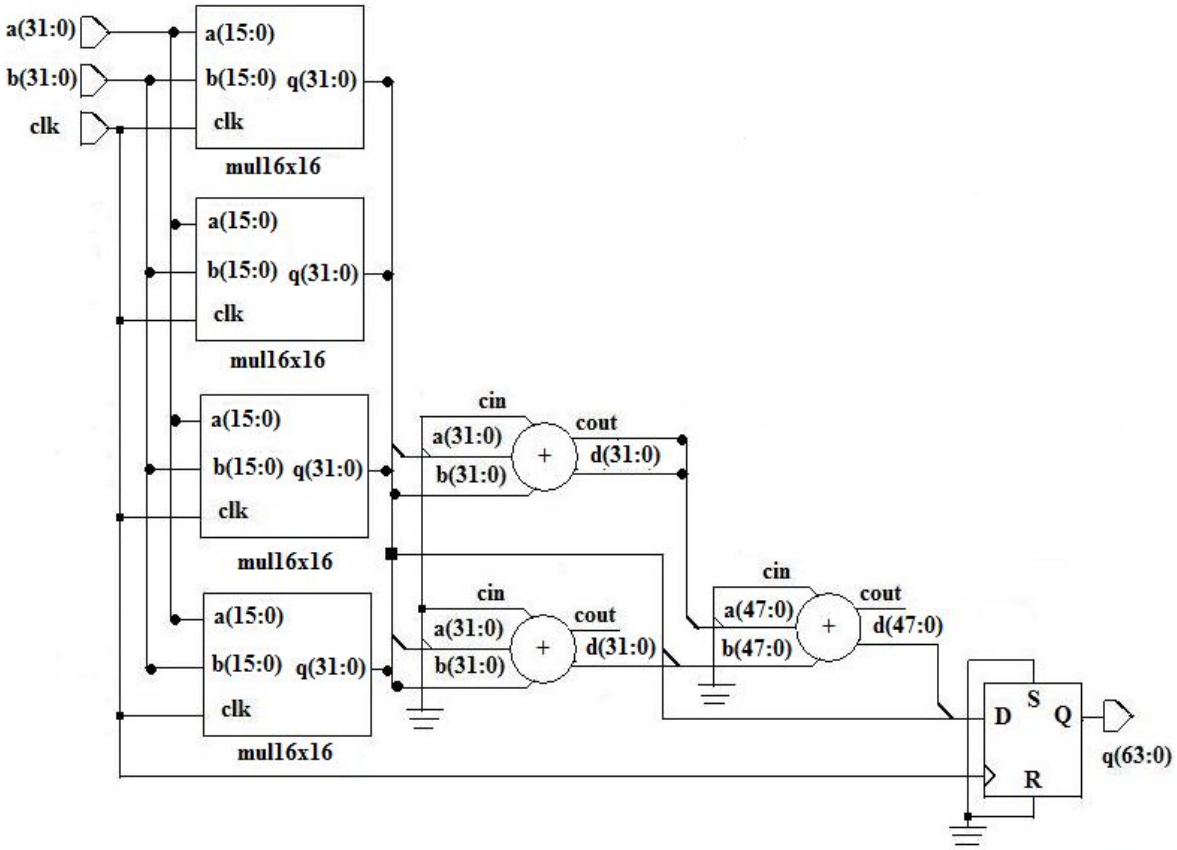


Figure 3.11: RTL View of 32X32 bits Vedic Multiplier.

3.7 DESIGN IMPLEMENTATION OF 64X64 BITS VEDIC MULTIPLIER

The 64X64 bit multiplier is structured using 32X32 bit blocks. The block diagram of 64 x64 bit multiplier by ISE Tool is shown below.

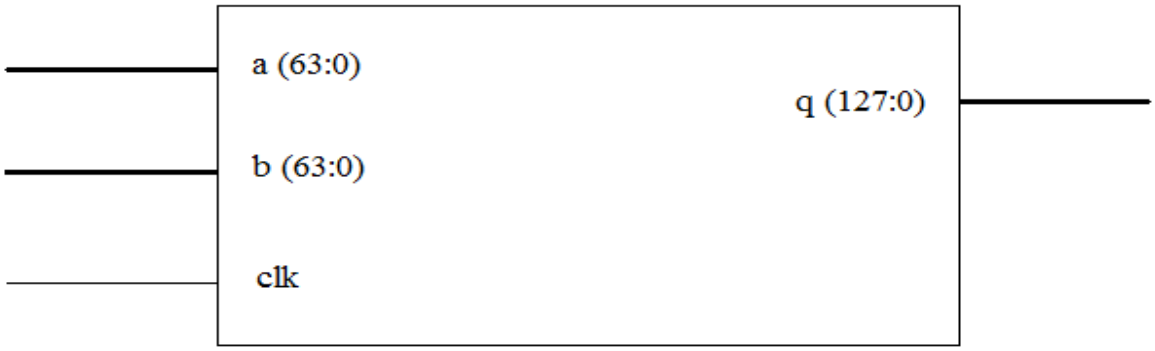
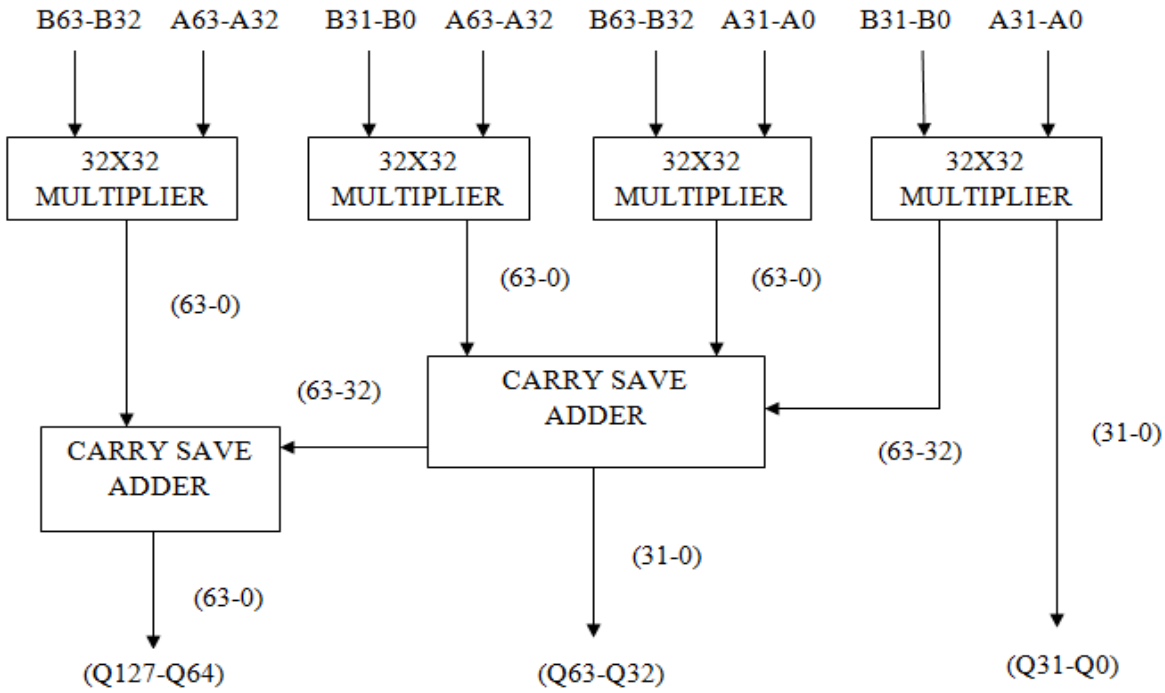


Figure 3.12: Block diagram of 64x64 bit Vedic Multiplier by ISE.

The 64 bits multiplicand A is decomposed into pair of 32 bits AH-AL. Similarly multiplicand B can be decomposed into BH-BL. Architecture of 64x64 bits Vedic Multiplier is shown as:



RESULT = (Q127-Q64) & (Q63-Q32) & (Q31-Q0)

Figure 3.13: Proposed architecture of 64x64 bits Vedic Multiplier.

The outputs of 32X32 bit multipliers are added accordingly to obtain the 128 bits final product. Thus, in the final stage two adders are also required.

The implemented RTL View of 64x64 bits Vedic Multiplier by using 32x32 blocks with the help of ModelSim Tool 6.1e is given below:

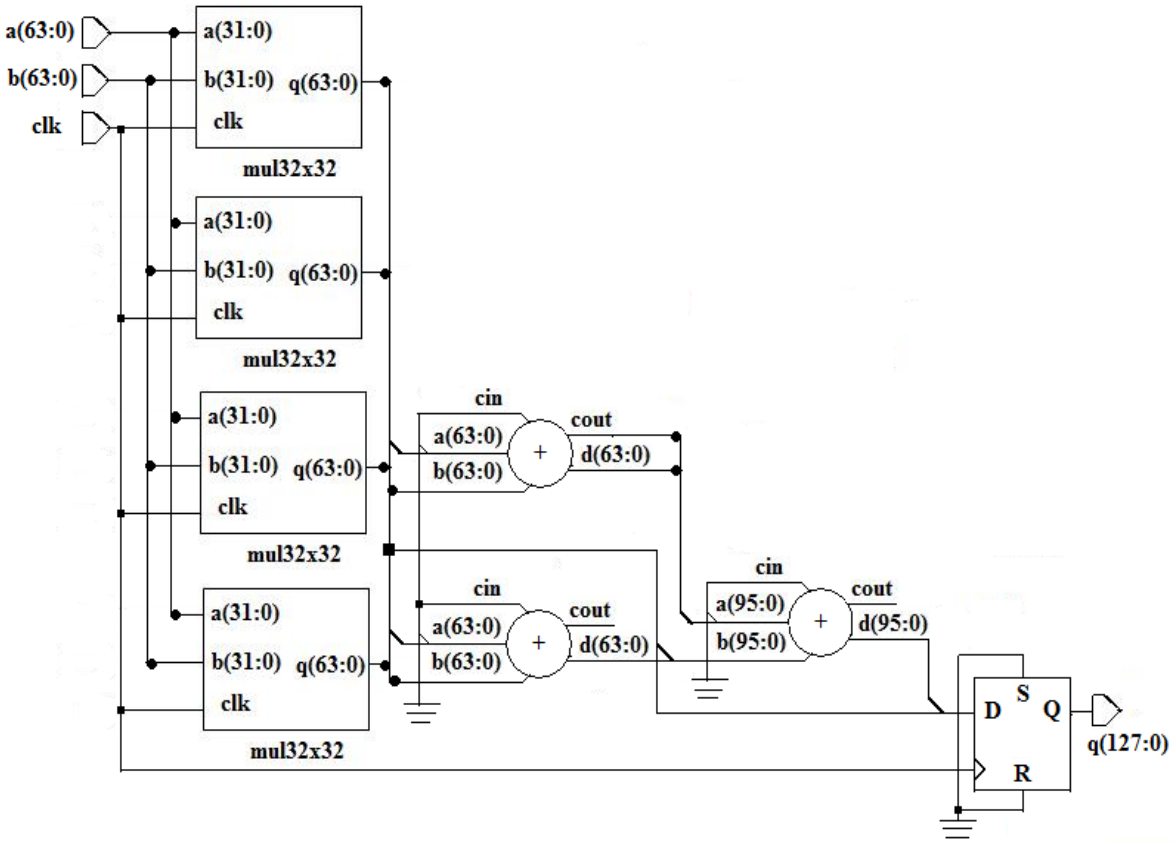


Figure 3.14: RTL View of 64X64 bits Vedic Multiplier.

CHAPTER 4

IMPLEMENTATION AND TESTING OF MULTIPLIER

4.1 FPGA IMPLEMENTATION

The FPGA that is used for the implementation of the circuit is the Xilinx Spartan 3E (Family), XC3S500 / XC3S1600 (Device), FG320 / FG484 (Package), -5 (Speed Grade). The working environment/tool for the design is Xilinx ISE 9.2i.

Device	XC3S500	XC3S1600
Technology	90nm	90nm
I/O Standards	232	376
No. of slices	4656	14752
Slice flip flops	9312	29504
4 input LUT's	9312	29504
Bonded IOBs	232	376
No. of GCLK's	24	24

Device family	spartan 3E
Target device	XC3S500
Package	FG320
Speed Grade	-5

Device family	spartan 3E
Target device	XC3S1600
Package	FG484
Speed Grade	-5

Table 4.1: Summary of FPGA features

4.1.1 Overview of FPGA

As the FPGA architecture evolves and its complexity increases, CAD software has become more mature as well. Today, most FPGA vendors provide a fairly complete set of design tools that allows automatic synthesis and compilation from design specifications in hardware specification languages, such as Verilog or VHDL, all the way down to a bit stream to program FPGA chips. A typical FPGA design flow includes the steps and components shown in Fig. 4.1[14].

Design constraints typically include the expected operating frequencies of different clocks, the delay bounds of the signal path delays from input pads to output pads (I/O delay), from the input pads to registers (setup time), and from registers to output pads (clock-to-output delay). In some cases, delays between some specific pairs of registers may be constrained.

Logic Utilizatio	Used	Available	Utilization	Note(s)
Number of Slices:	6966	4656	149%	Resourse Overuse
Number of 4 input LUTs:	10609	9312	113%	Resource Overuse
Number of bonded IOBs	257	232	110%	Resourse Overuse

Table 4.2: Device (XC3S500) Utilization Summary for 64x64 bits multiplier

Logic Utilizatio	Used	Available	Utilization	Note(s)
Number of Slices:	6966	14752	47%	
Number of 4 input LUTs:	10609	29504	35%	
Number of bonded IOBs	257	376	68%	

Table 4.3: Device (XC3S1600) Utilization Summary for 64x64 bits multiplier

The second design input component is the choice of FPGA device. Each FPGA vendor typically provides a wide range of FPGA devices, with different performance, cost, and power tradeoffs. The designer may start with a small (low capacity) device with a nominal speed-grade. But, if synthesis effort fails to map the design into the target device, the designer has to upgrade to a high-capacity device. Similarly, if the synthesis result fails to meet the operating frequency, he has to upgrade to a device with higher speed-grade. In both the cases, the cost of the FPGA device will increase by 50% or even

by 100%. Thus better synthesis tools are required, since their quality directly impacts the performance and cost of FPGA designs [15, 16].

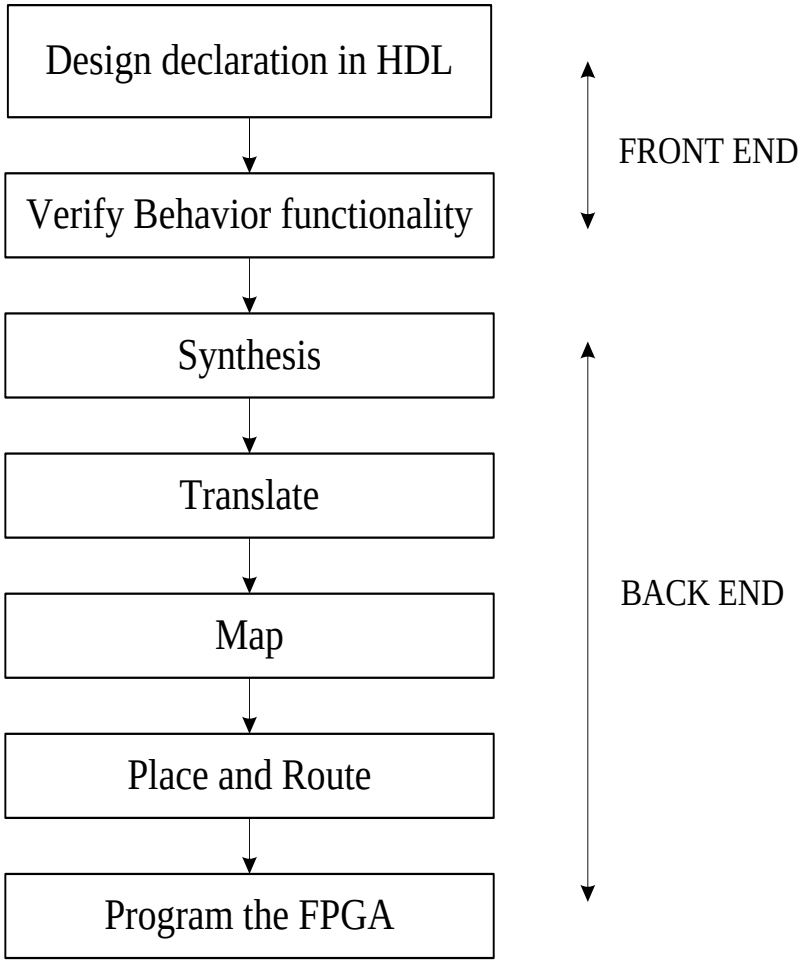


Figure 4.1: FPGA Design Flow

4.1.2 Design Entry

The basic architecture of the system is designed in this step which is coded in a Hardware description Language like Verilog or VHDL. A design is described in Verilog using the concept of a design module.

4.1.3 Behavioral Simulation

After the design phase, the code is verified using a simulation software i.e. Modelsim6.1e for different inputs to generate outputs and if it verifies then we will proceed further otherwise modifications and necessary corrections will be done in the HDL code. This is called as the behavioral simulation.

4.1.4 Design Synthesis

After the correct simulations results, the design is then synthesized. During synthesis, the Xilinx ISE tool does the following operations:

1. HDL Compilation: The tool compiles all the sub-modules in the main module if any and then checks the syntax of the code written for the design.

2. Design Hierarchy Analysis: Analysis the hierarchy of the design.

3. HDL Synthesis: The process which translates VHDL or Verilog code into a device net list format. i.e. a complete circuit with logical elements such as Multiplexer, Adder/Subtractors, Counters, Registers, Flip flops, Latches, Comparators, XORs, Tristate Buffers, Decoders, etc.

4. Advanced HDL Synthesis: Low Level synthesis: The blocks synthesized in the HDL synthesis and the Advanced HDL synthesis are further defined in terms of the low level blocks such as buffers, lookup tables. It also optimizes the logic entities in the design by eliminating the redundant logic, if any. The tool then generates a 'netlist' file (NGC file) and then optimizes it.

4.1.5 Design Implementation

The design implementation process consists of the following sub processes:

1. Translation: The Translate process merges all the input netlists and design constraints informations and outputs a Xilinx NGD (Native Generic Database) file.

2. Mapping: The Map process is run after the Translate process is complete. Mapping maps the logical design described in the NGD file to the components/primitives (slices/CLBs) present on the target device. The Map process creates an NCD file.

3. Place and Route: The place and route (PAR) process is run after the design has been mapped. PAR uses the NCD file created by the Map process to place and route the design on the target FPGA design.

4. Bitstream Generation: The collection of binary data used to program reconfigurable logic device is most commonly referred to as a "bit stream," although this is somewhat

misleading because the data are no more bit oriented than that of an instruction set processor and there is generally no “streaming”.

5. Functional Simulation: Post-Translate (functional) simulation can be performed prior to mapping of the design. This simulation process allows the user to verify that your design has been synthesized correctly and any differences due to the lower level of abstraction can be identified.

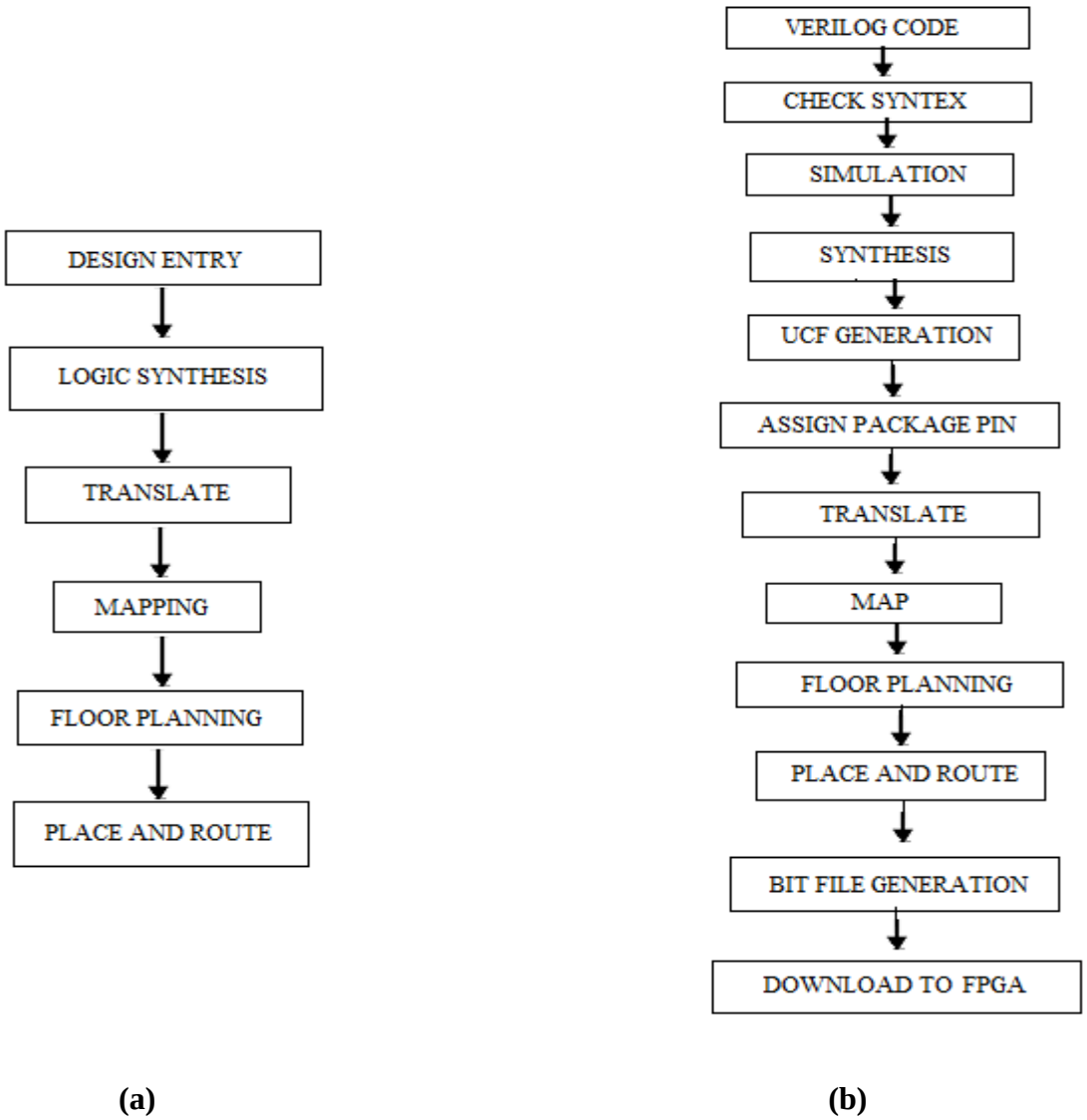


Figure 4.2: (a) Design Flow during FPGA Implementation (b) Procedure Followed for Implementation

6. Static timing analysis: Three types of static timing analysis be performed that are:

(a) Post-fit Static timing analysis: The Analyze Post-Fit Static timing process opens the timing Analyzer window, which lets you interactively select timing paths in your design for tracing the timing results.
(b) Post-Map Static Timing Analysis: You can analyze the timing results of the Map process. Post-Map timing reports can be very useful in evaluating timing performance (logic delay + route delay).

(c) Post Place and Route Static Timing Analysis: Post-PAR timing reports incorporate all delays to provide a comprehensive timing summary. If a placed and routed design has met all of your timing constraints, then you can proceed by creating configuration data and downloading a device.

7. Timing Simulation: After your design has been placed and routed, a timing simulation netlist can be created. This simulation process allows you to see how your design will behave in the circuit.

4.2 IMPLEMENT DESIGN IN FPGA

The Spartan-3E Starter Kit board highlights the unique features of the Spartan-3E FPGA family and provides a convenient development board for embedded processing applications. The board implements the design and verifies that it meets the timing constraints after check syntax and synthesis.

4.3 DOWNLOAD DESIGN TO THE SPARTAN-3E KIT

This is the last step in the design verification process. This section provides simple Instructions for downloading the design to the Spartan-3 Starter Kit demo board.

4.4 PROGRAMMING THE FPGA

A programming file is generated by running the Generate Programming File process. This process can be run after the FPGA design has been completely routed. The Generate Programming File process runs BitGen, the Xilinx bitstream generation program, to produce a bitstream (.BIT) or (.ISC) file for Xilinx device configuration. The FPGA device is then configured with the .bit file using the JTAG boundary scan method. After

the Spartan device is configured for the intended design, then its working is verified by applying different inputs.

4.4.1 LCD Interfacing Programming in FPGA

The figure 4.3 represents the implementation of LCD Interfacing of 64x64 bits Vedic multiplier.

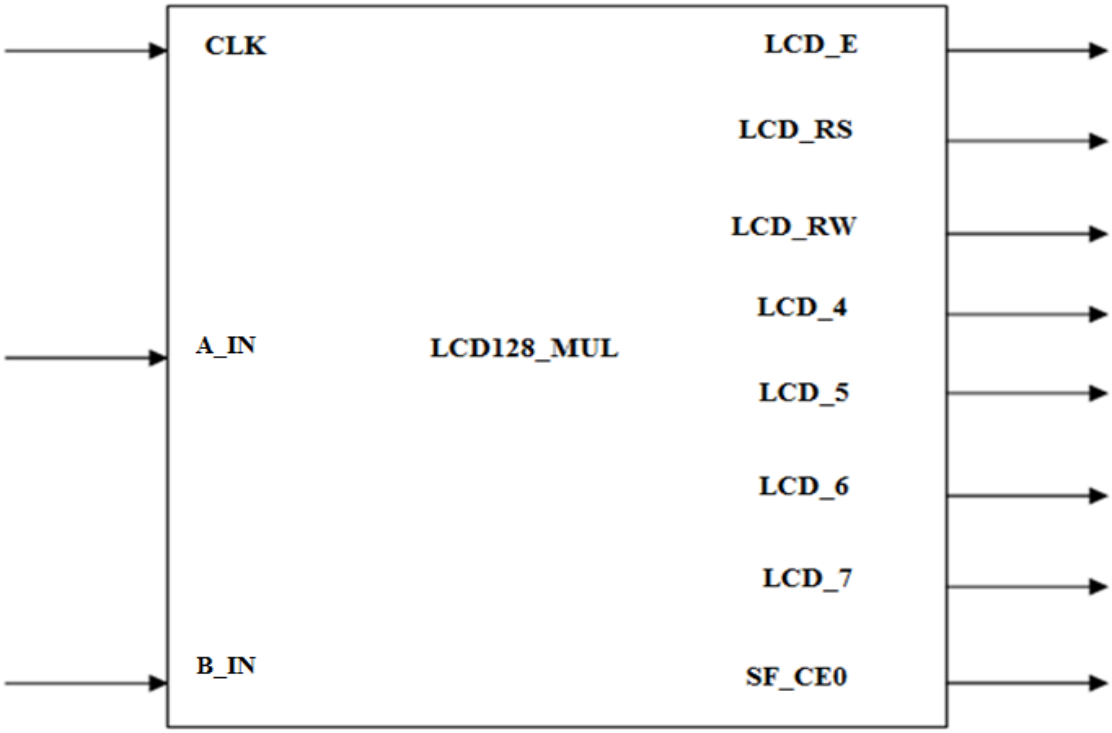


Figure 4.3: Pin diagram of LCD Interfacing of 64X64Vedic Multiplier.

There are eleven ports namely, input clock (CLK), (A_IN, B_IN) ports of 64x64 bit Vedic multiplier, these are valid input values, (LCD_E) Read/Write Enable Pulse ‘0’ for Disabled ‘1’for Read/Write operation enabled, FPGA pin number is M18. (LCD_RS) Register Select ‘0’ for Instruction register during write operations, Busy Flash during read operations, ‘1’ for Data for read or write operations, FPGA pin number is L18. (LCD_RW) Read/Write Control ‘0’ for WRITE, LCD accepts data '1’ for READ, LCD Presents data, FPGA pin number is L17. (LCD_4) Data bit DB4, FPGA pin number is R15. (LCD_5) Data bit DB5 FPGA pin number is R16. (LCD_6) Data bit DB6,FPGA pin number is P17.(LCD_7) Data bit DB7 FPGA pin number is M15.(SF_CE0) When the

Strata Flash memory is disabled (SF_CE0 = High), then the FPGA application has full read/write access to the LCD.

4.4.2 Keyboard Interfacing Programming in FPGA

The figure 4.4 represents the implementation of Keyboard interfacing of 64x64 Vedic Multiplier.

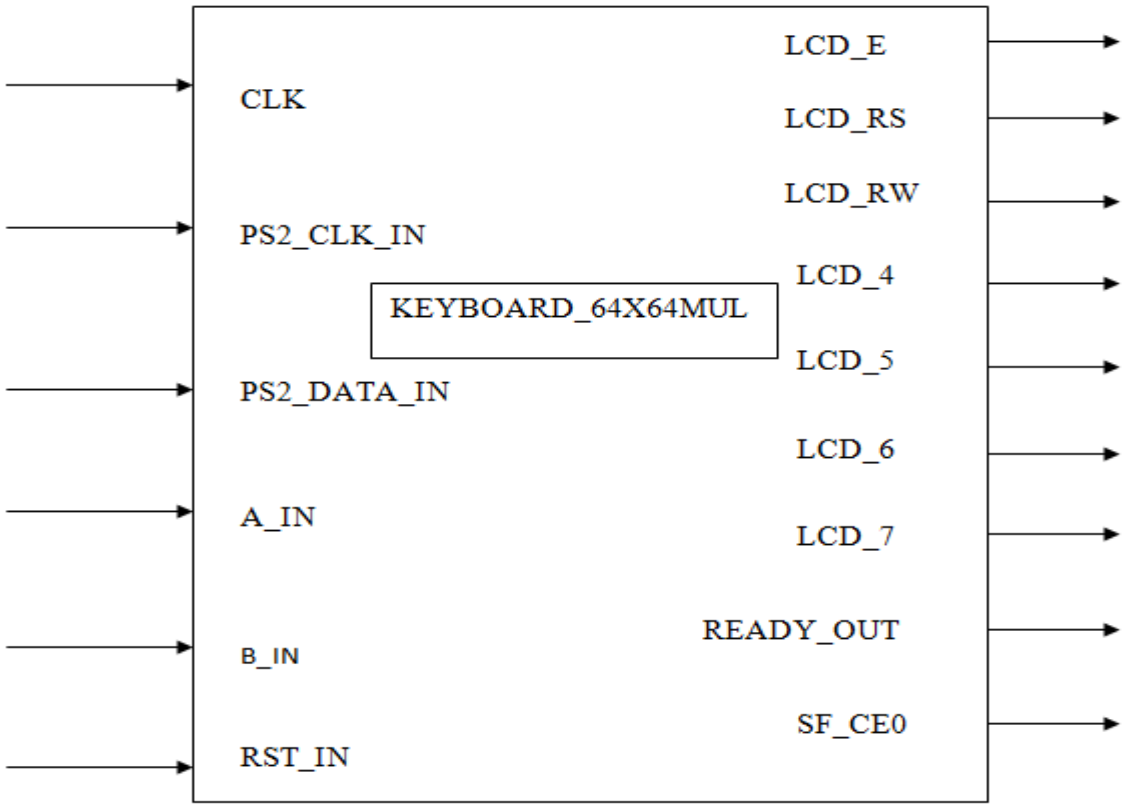


Figure 4.4: Pin diagram of Keyboard interfacing of 64X64 Vedic Multiplier

There are fifteen ports, some of the ports used in keyboard interfacing these are all LCD interfacing ports rest of the ports are discussed as follows:

The representation of the rest four ports:

(1) PS2_CLK_IN: It is the input wire of PS/2 Clock line. FPGA pin number is G14.

(2) PS2_DATA_IN: It is the input wire of PS/2 Data line, FPGA pin number is G13.

(3) RST_IN: It is the input wire of Asynchronous reset.

(4) READY_OUT: It is the output register, this output high when module is output ready.

4.5 BIST – AN OVERVIEW

A digital system is tested and diagnosed during its lifetime on numerous occasions. It is very critical to have quick and very high fault coverage testing. One common and widely used in semiconductor industry for IC chip testing is to ensure this is to specify test as one of the system functions and thus becomes self-test. A system designed without an integrated test strategy which covering all levels from the entire system to components is being described as chip-wise and system-foolish. A proper designed Built-In-Self-Test (BIST) is able to offset the cost of added test hardware while at the same time ensuring the reliability, testability and reduce maintenance cost [6, 17].

The basic idea of BIST, in its most simple form, is to design a circuit so that the circuit can test itself and determine whether it is “good” or “bad” (fault-free or faulty, respectively). This typically requires additional circuitry whose functionality must be capable of generating test patterns as well as providing a mechanism to determine if the output responses of the circuit under test (CUT) to the test patterns correspond to that of a fault-free circuit [6].

4.5.1 Basic BIST Architecture

A representative architecture of the BIST circuitry as it might be incorporated into the CUT is illustrated in the block diagram of Figure 4.5.

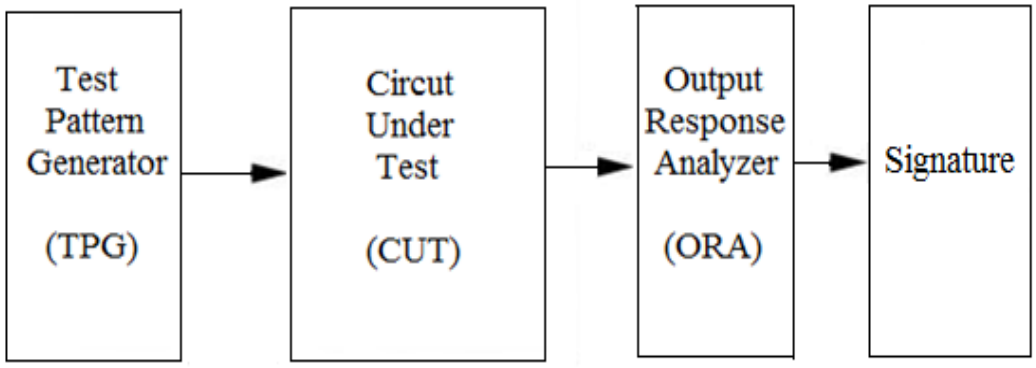


Figure 4.5: Block Diagram of BIST Architecture [7].

This BIST architecture includes two essential functions as well as two additional functions that are necessary to facilitate execution of the self-testing feature while in the system.

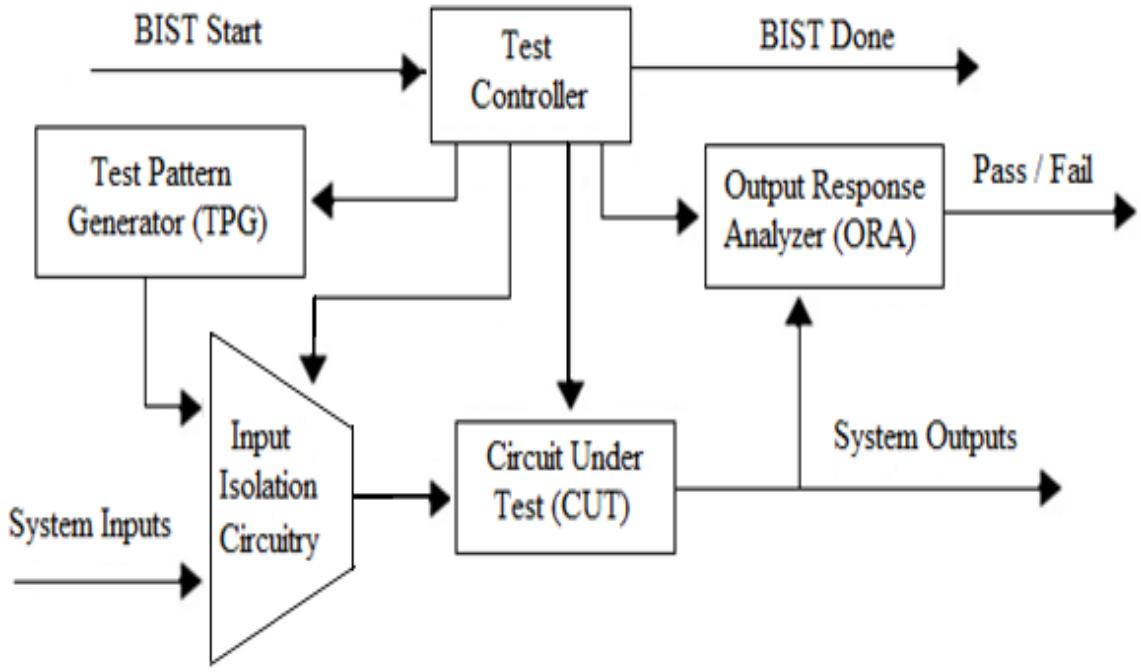


Figure 4.6: Basic BIST Architecture [6].

The two essential functions include the test pattern generator (TPG) and output response analyzer (ORA). While the TPG produces a sequence of patterns for testing the CUT, the ORA compacts the output responses of the CUT into some type of Pass/Fail indication.

The other two functions needed for system-level use of the BIST include the test controller (or BIST controller) and the input isolation circuitry. Aside from the normal system I/O pins, the incorporation of BIST may also require additional I/O pins for activating the any BIST sequence (the BIST Start control signal), reporting the results of the BIST (the Pass/Fail indication), and an optional indication (BIST Done) that the BIST sequence is complete and that the BIST results are valid and can be read to determine the fault-free/faulty status of the CUT [6, 7].

4.5.2 BIST Process

Figure 4.7 shows the BIST system hierarchy for the 3 level of packaging which is the system level, board level and chip level. The system consists of several PCBs (or

boards). Each of the PCB has multiple chips. The system Test Controller can activate self-test simultaneously on all PCBs. Each Test Controller on each PCB board can activate self-test on all the chips on the board.

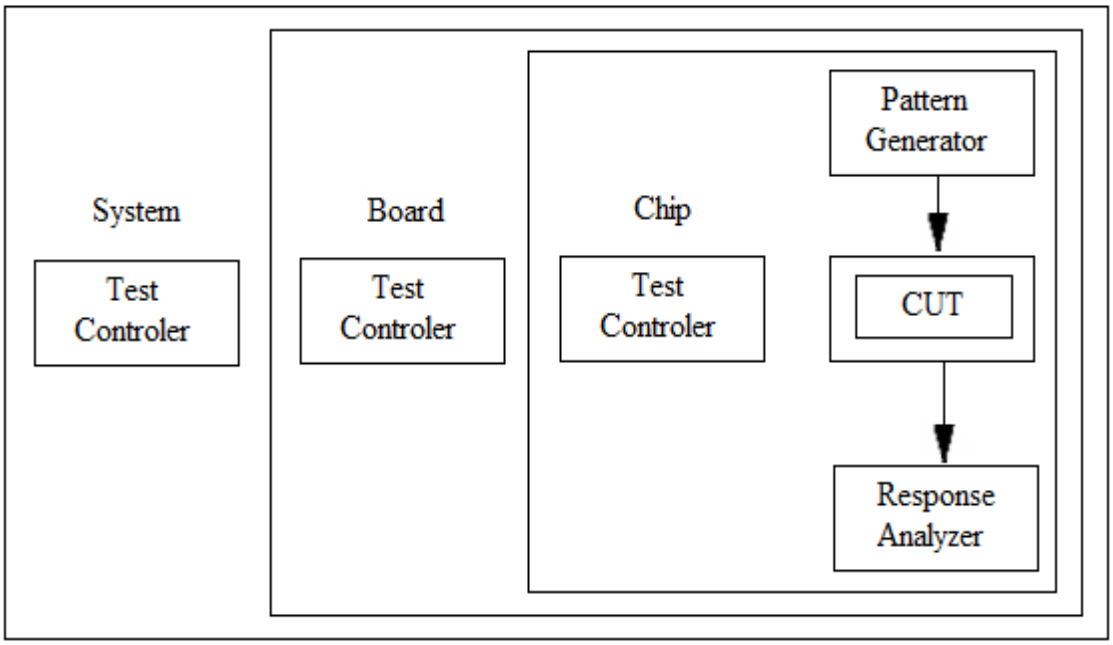


Figure 4.7: BIST Hierarchy [17].

The chip Test Controller runs the self-test on the chip and transmits the result out to the board Test Controller. The board Test Controller accumulates test results from all chips on the PCB and sends the results to the system Test Controller. The system Test Controller uses all of these results to determine if the chips and board are faulty.

4.5.3 BIST Implementation

Figure 4.8 shows the BIST hardware architecture in more detail. In this project, the BIST module in the IOP is developed based on the architecture in Figure 4.6. Basically, a design with embedded BIST architecture consists of a test controller, hardware pattern generator, input multiplexer, circuit under test (CUT) which in this project is the IOP and output response compactor.

Optionally, a design with BIST capability may includes also the comparator and Read-Only-Memory (ROM). As shown in Figure 4.6, the test controller is used to control

the test pattern and test generation during BIST mode. Hardware pattern generator functions to generate the input pattern to the CUT.

Normally, the pattern generator generates exhaustive input test patterns to the CUT to ensure the high fault coverage. For example, a CUT with 10 inputs will required 1024 test patterns. Primary Inputs are the input for CUT during the non BIST mode or in other word, functional mode. Input multiplexer is used to select correct inputs for the CUT for different mode.

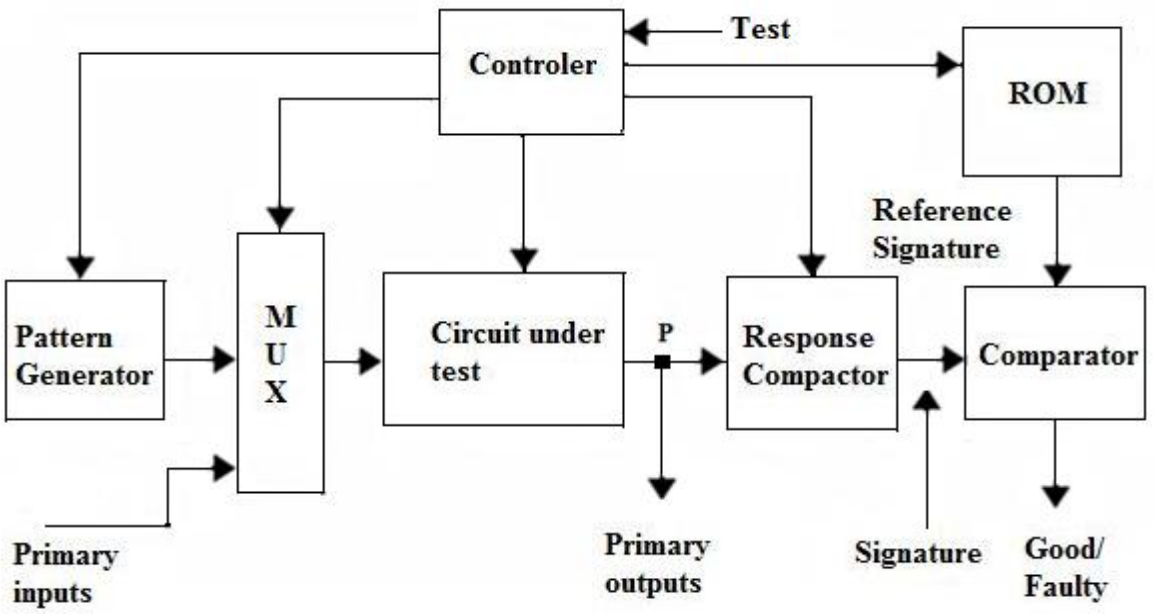


Figure 4.8: BIST architecture [6, 17].

During BIST mode, it selects input from the hardware pattern generator while during functional mode, selects primary inputs. Output response compactor acts as compactor to reduce the number of circuit responses to manageable size that can be used as the signature and stored on the ROM.

4.6 DESIGNS FOR TESTABILITY

The mechanics of testing, as illustrated in Figure 4.9, are similar at all levels of testing, including design verification. A set of input stimuli is applied to a circuit and the output response of that circuit is compared to the known good output response, or

expected response, to determine if the circuit is "good" or "faulty". There are various testing techniques of a circuit. Basic testing flow is shown below.

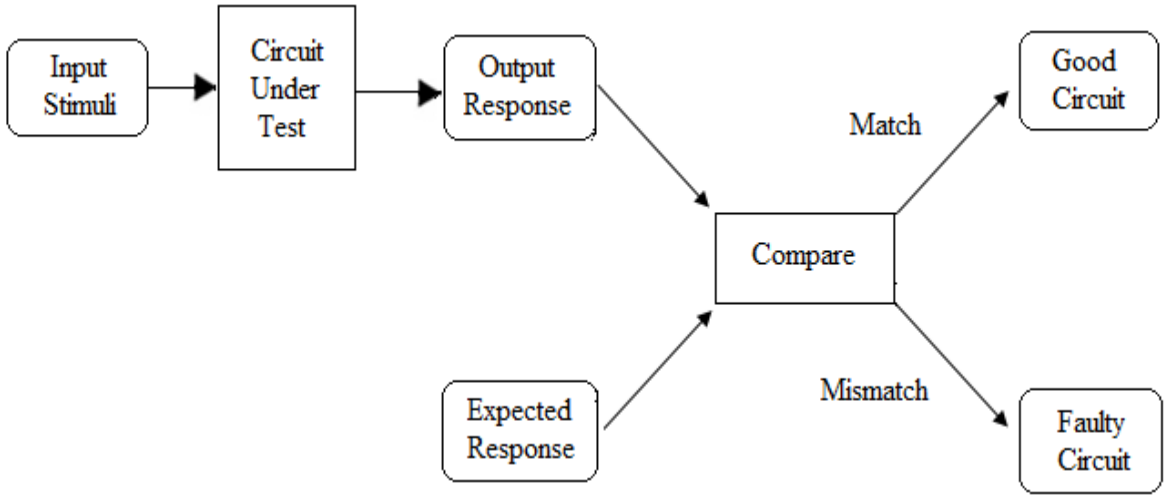


Figure 4.9: Basic testing flow.

BIST include:

1. Pseudo random pattern generator(PRPG)
2. BIST Response Compaction
3. Output response analyzer(ORA)

4.6.1 Pseudo Random Pattern Generator

To perform test on any circuit, a Pseudo random pattern generator use mostly linear feedback shift register (LFSR) to generate input test vectors.

Standard LFSR

The standard LFSR method has been used as the test pattern generator for the BIST. A LFSR is a shift register where the input is a linear function of two or more bits (taps) as shown in Figure 4.10. It consists of D flip-flops and linear exclusive-OR (XOR) gates. It is considered an external exclusive-OR LFSR as the feedback network of the XOR gates feeds externally from X0 to Xn-1. One of the two main parts of an LFSR is the shift register. A shift register is used to shift its contents into adjacent positions within the register or, in the case of the position on the end, output of the register. The position

on the other end is left empty unless some new content is shifted into the register. The bit positions selected for use in the feedback function are called "taps". The list of the taps is known as the "tap sequence".

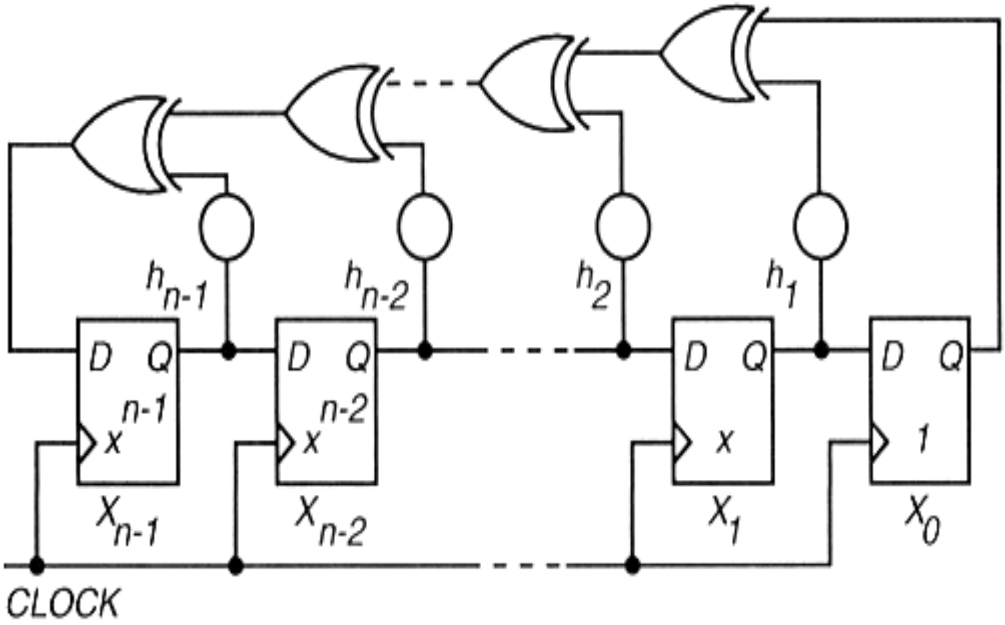


Figure 4.10: Linear Feedback Shift Register [17].

By convention, the output bit of an LFSR that is n bits long is the nth bit; the input bit of an LFSR is bit 1 [19, 23].

4.6.2 BIST Response Compaction

During BIST, for every test pattern that being generated, the CUT produces a set of output values. In order to ensure the chip is fault free, every output values from the CUT for each test pattern will need to compare with the correct output values obtained from the simulations. This is a tedious and time consuming process. Thus, it is necessary to reduce the enormous of circuit responses to a manageable size that can be either store in the chip or can easily compared with the golden response values [18, 23].

4.6.3 Output Response Analyzer

The on-chip storage of the entire test input vectors with their corresponding outputs are stored in RAM. After applying the input, Outputs are checked and compared with the

stored ones. If stored data is same as the output of the circuit then circuit is fault free otherwise it is faulty [21, 22].

In BIST, both the Test Pattern Generation (TPG) and Output Response Analyzer (ORA) are incorporated inside the chip. Assuming that all levels of hierarchy uses BIST, each element test itself and transmits the result to the higher level of hierarchy [6, 20].

4.7 IMPLEMENTATION OF VEDIC MULTIPLIER WITH BIST

4.7.1 BIST implementation flow: The implementation flow of BIST on Vedic multiplier is shown in fig 4.11.

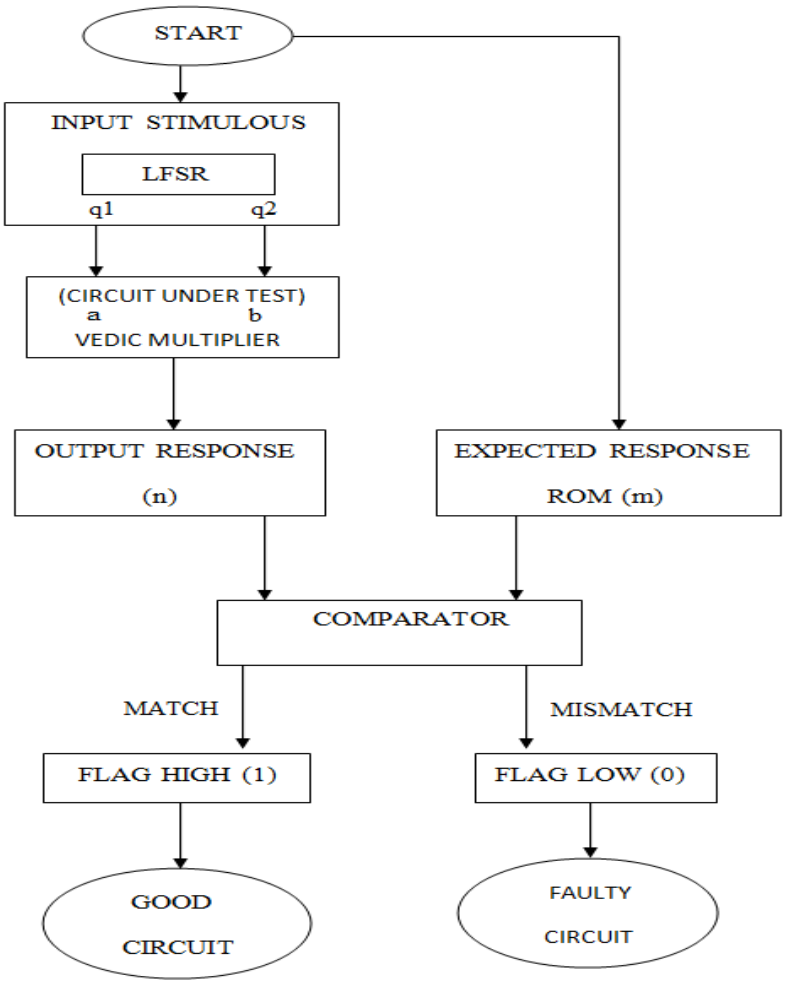


Figure 4.11: Implementation flow of BIST

4.7.2 Block diagram and RTL view of Vedic multiplier with BIST

This block represents the implementation of 64x64 bits Vedic Multiplier using BIST.

The representation has six ports:

1) CLK: It is the input clock.
2) A_IN (63:0): It is the first input value.

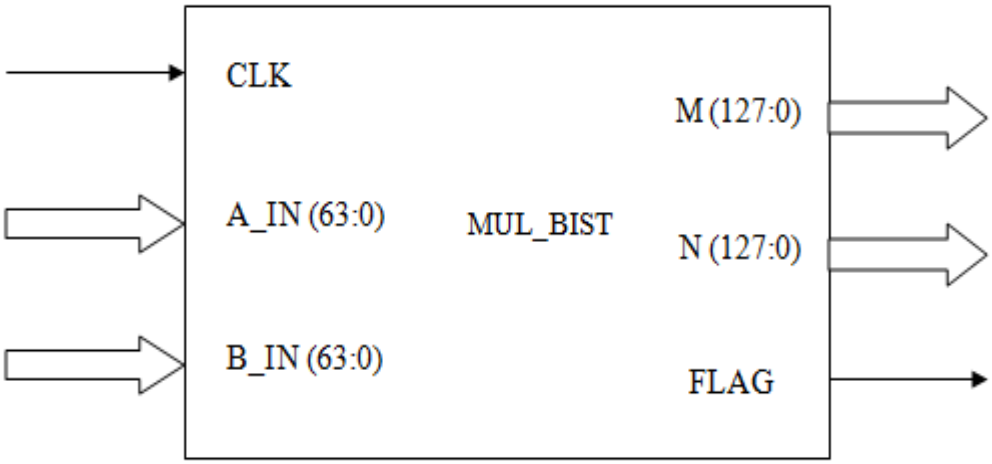


Figure 4.12: Implementation of vedic multiplier using BIST

3) B_IN (63:0): It is the second input of the 64x64 bits Vedic multiplier.
4) M (127:0): It is the output register of Multiplier in which expected result of multiplier is saved.
5) N (127:0): It is output of design which is compared with expected result.
6) Flag: It is the output register, this output will give high when output response equal to expected response otherwise flag will be zero.

Figure 4.13, shows the RTL schematic of Vedic multiplier in which two outputs q1 and q2 which are generated by LFSR are applied as inout wire to CUT (Multiplier) through Input Stimulus (which interface q1 and q2 to the input of CUT). Then output n of CUT compare with calculated result saved in register m. Both the results matched correctly and the flag (output register) remains high (1). Thus the designed circuit is fault free and BIST technique is proper and efficient for testing of multiplier.

Thus, a fault free circuit (Vedic multiplier) is implemented in this thesis work. After implementation on hardware it can be used in any processor to increase speed of computation.

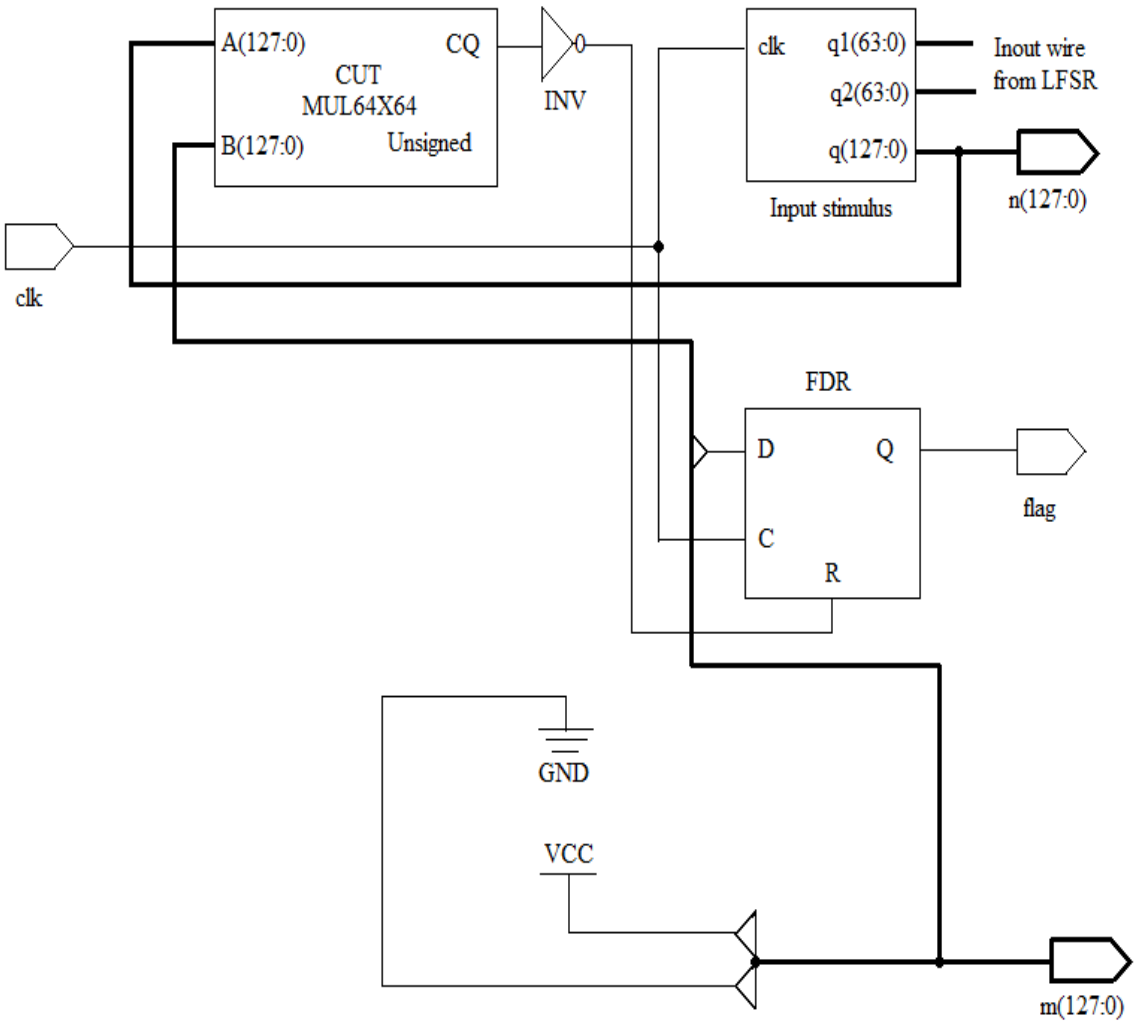


Figure 4.13: RTL view of Vedic multiplier with BIST by ISE Tool

CHAPTER 5

RESULTS AND CONCLUSION

5.1 RESULTS

5.1.1 SIMULATION RESULTS OF 64X64 BITS VEDIC MULTIPLIER

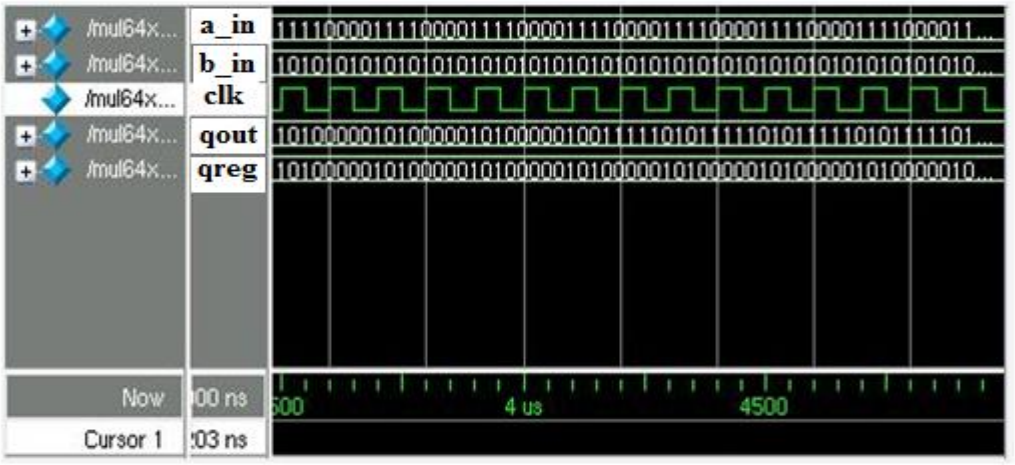


Figure 5.1: Simulation result of 64x64 bits Vedic multiplier by Model Sim

Description:-

a : Input data 64 – bit

b : Input data 64 – bit

clk : Input clock

q _ out : Output data 128 – bit

q _ reg : Output reg 128 – bit

a = 1111000011110000111100001111000011110000111100001111000011110000111100
0011110000

b = 1010101010101010101010101010101010101010101010101010101010101010
1010101010

q = 1010000010100000101000001010000010100000101000001010000001001111101011111010111110101111101011111010110000

Thus simulated result and calculated result match correctly.

1 Synthesis Results

Device utilization summary:

Selected Device: 3s500efg320-5

Number of Slices:	1712 out of 4656 36%
Number of Slice Flip Flops:	1808 out of 9312 19%
Number of 4 input LUTs:	2665 out of 9312 28%
Number used as logic:	2529
Number used as Shift registers:	136
Number of IOs:	129
Number of bonded IOBs:	129 out of 232 55%
Number of GCLKs:	1 out of 24 4%

Minimum period: 7.784ns (Maximum Frequency: 128.461MHz)

Total estimated power consumption: 81 (mW)

But for 64x64 bits multiplication this device shows a message: - This design is too large for given device and package. As

Note: More than 100% of Device resources are used.

So here for 64x64 bits multiplication device SPARTEN 3E (XC3S1600E-5-FG484) is used.

Device utilization summary:

Selected Device: 3s1600efg484-5	
Number of Slices:	6966 out of 14752 47%
Number of Slice Flip Flops:	7328 out of 29504 24%
Number of 4 input LUTs:	10609 out of 29504 35%
Number used as logic:	10049
Number used as Shift registers:	560
Number of IOs:	257
Number of bonded IOBs:	257 out of 376 68%

Number of GCLKs: 1 out of 24 4%

FPGA Device Package	Type of Multiplier	Area (in LUT's)	Delay (ns)	Speed (MHz)	Power (mW)	Note Use
SPARTAN XC3S500E FG320	VEDIC MUL 32X32	2665/9%	7.784	128.461	81	
	VEDIC MUL 64X64	10609/113%	10.241	97.647	-	Device overuse
SPARTAN XC3S1600E FG484	VEDIC MUL 32X32	2665/9%	1.689	128.461	132	
	VEDIC MUL 64X64	10609/35%	10.241	597.647	203	

Table 5.1: Synthesis results of multipliers.

2 Timing Results

Minimum period: 10.241ns (Maximum Frequency: 97.647MHz)

Minimum input arrival time before clock: 4.676ns

Maximum output required time after clock: 4.040ns

Maximum combinational path delay: No path found

3 Power Results

Total estimated power consumption: 203 (mW)

5.1.2 OUTPUTS OF 32X32 BITS DISPLAY ON LCD SCREEN OF FPGA

In burning our code of 32X32 Bits Vedic multiplier on FPGA kit (XC3S500E-5-FG320) we see final step that a REPORT "Program Succeeded" blink after step IMPACT [BOUNDARY SCAN].

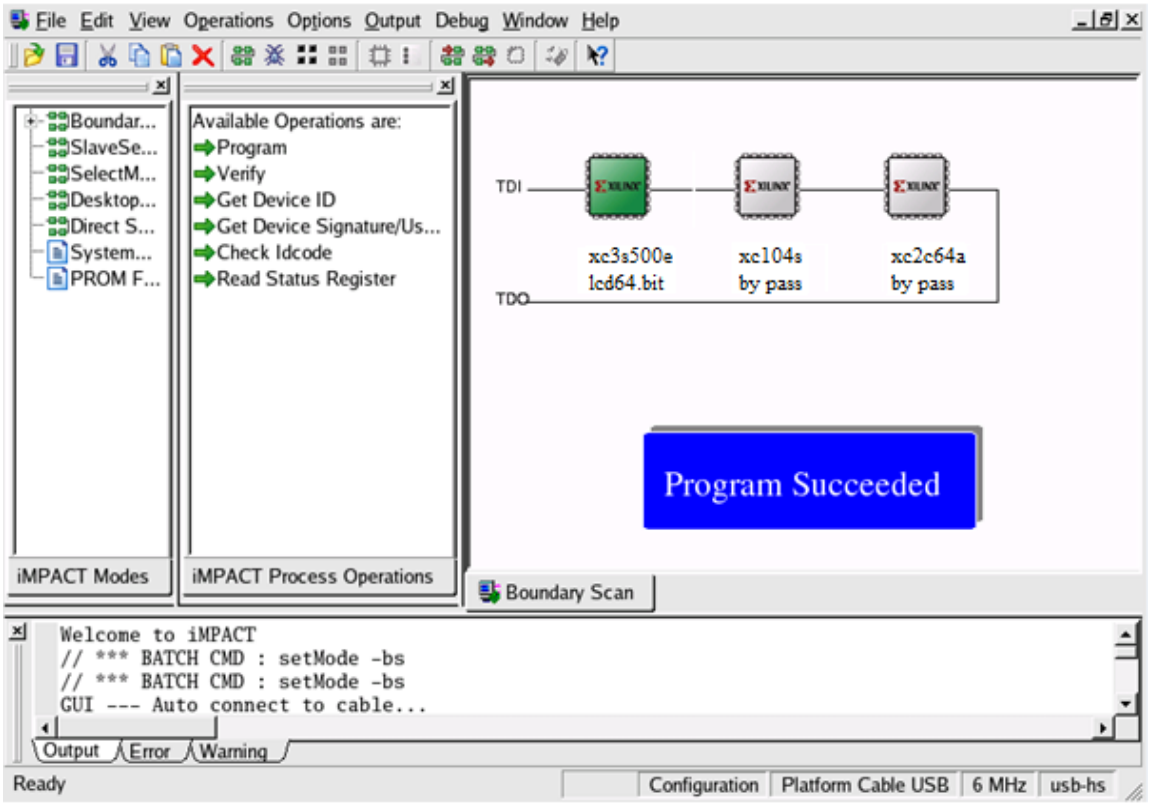


Figure 5.2: LCD display (output of 32X32) on Spartan 3E FPGA kit

1 Synthesis Results

Selected Device: 3s500efg320-5			
Number of Slices:	25 out of	4656	0%
Number of Slice Flip Flops:	36 out of	9312	0%
Number of 4 input LUTs:	48 out of	9312	0%
Number used as logic:	41		
Number used as Shift registers:	7		
Number of IOs:	9		
Number of bonded IOBs:	9 out of	232	3%
Number of GCLKs:	1 out of	24	4%

Output on LCD Screen = A0A0A09F5F5F5F60 (In hexa).

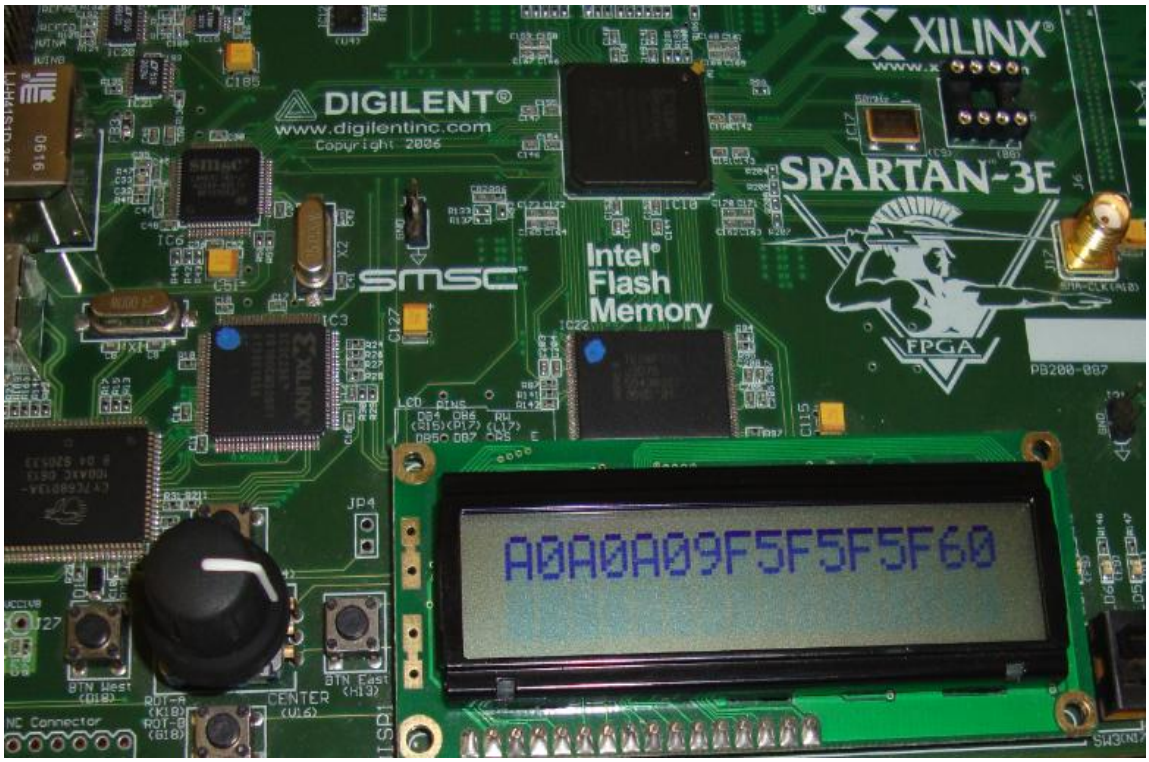


Figure 5.3: Output of 32X32 bits multiplier shown on LCD of SPARTAN kit

2 Timing Results

Speed Grade: -5

Minimum period: 4.191ns (Maximum Frequency: 238.635MHz)

Minimum input arrival time before clock: No path found

Maximum output required time after clock: 4.063ns

Maximum combinational path delay: No path found

3 Power Results

Total estimated power consumption: 151 (mW).

Here a table is given for LCD display results of 32x32 bits Vedic multiplier.

FPGA Device Package	Type of LCD DISPLAY	Area (in LUT's)/ Use	Delay (ns)	Speed (MHz)	Power (mW)
SPARTAN XC3S500E FG320	LCD DISPLAY 32X32	48/0%	4.191	238.635	151
SPARTAN XC3S1600E FG484	LCD DISPLAY 32X32	48/0%	4.191	238.635	202.62
	LCD DISPLAY 64X64	48/0%	4.191	238.635	203

Table 5.2: Synthesis results of LCD display of multiplier (32X32)

5.1.3 BIST RESULTS

5.1.3.1 LFSR

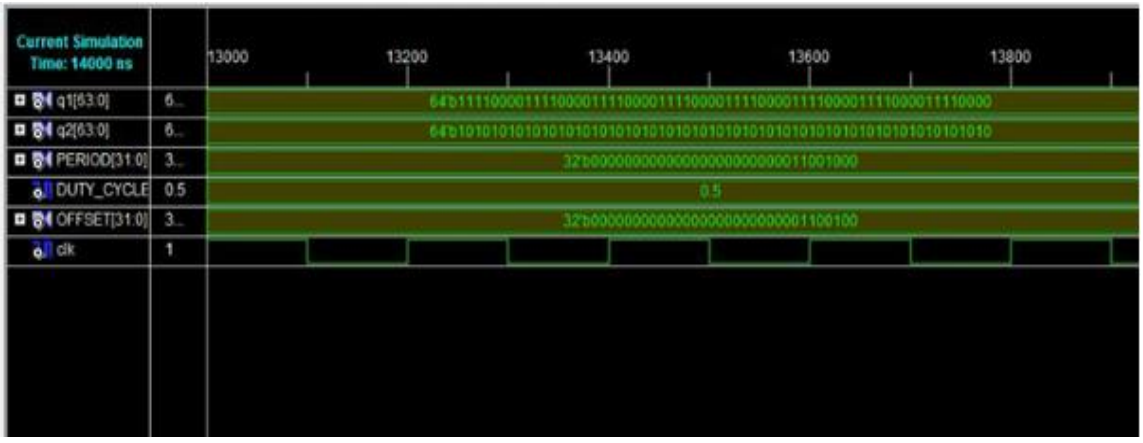


Figure 5.4: Waveform generation (TPG) by LFSR

To generate the TPG for multiplier LFSR is used. Here q1 and q2 are two input patterns Generated by LFSR.

q1 (63:0) =

1111000011110000111100001111000011110000111100001111000011110000

q2 (63:0) =

1010101010101010101010101010101010101010101010101010101010101010

1 Synthesis Results

Selected Device: 3s1600efg484-5

Number of Slices: 71 out of 14752 0%

Number of Slice Flip Flops: 122 out of 29504 0%

FPGA Device Package	Type of LFSR	Area (FLIP FLOP)	Delay (ns)	Speed (MHz)	Power (mW)
SPARTAN XC3S500E FG320	LFSR 32X32	58	1.689	592.084	80.98
	LFSR 64X64	122	1.689	592.084	151
SPARTAN XC3S1600E FG484	LFSR 32X32	58	1.689	592.084	132
	LFSR 64X64	122	1.689	592.084	202.62

Table 5.3: Synthesis results of LFSR

Number of IOs: 129

Number of bonded IOBs: 129 out of 376 34%

IOB Flip Flops: 2

Number of GCLKs: 1 out of 24 4%

2 Timing Results

Speed Grade: -5

Minimum period: 1.689ns (Maximum Frequency: 592.084MHz)

Minimum input arrival time before clock: No path found

Maximum output required time after clock: 4.063ns

Maximum combinational path delay: No path found

3 Power Results

Total estimated power consumption: 202.62 (mW).

5.1.3.2 INPUT STIMULUS

This is the top level module in which inputs q1, q2 and output q (127:0) is generated for the testing of circuit.

q1=1111000011110000111100001111000011110000111100001111000011110000

q2=1010101010101010101010101010101010101010101010101010101010101010

q=1010000010100000101000001010000010100000101000001010000010011111010111
1101011111010111110101111101011111010111110101111101100000

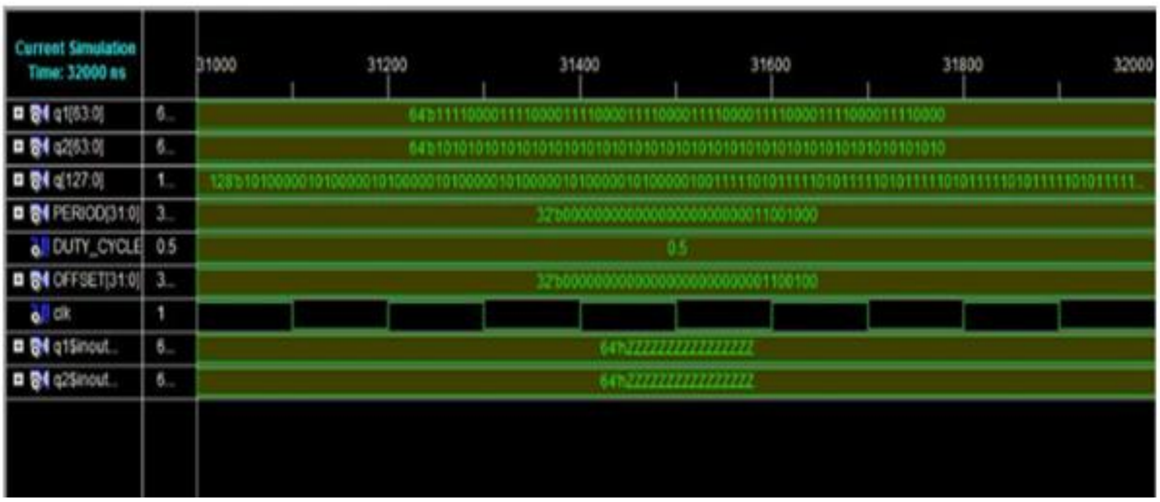


Figure 5.5: Wave form generation for input stimulus

1 Synthesis Results

Selected Device: 3s1600efg484-5

Number of Slices:	6141 out of 14752 41%
Number of Slice Flip Flops:	7114 out of 29504 24%
Number of 4 input LUTs:	10128 out of 29504 34%
Number used as logic:	9594
Number used as Shift registers:	534
Number of IOs:	257
Number of bonded IOBs:	257 out of 376 68%
IOB Flip Flops:	2
Number of GCLKs:	1 out of 24 4%

2 Timing Results

Speed Grade: -5

Minimum period:	10.209ns (Maximum Frequency: 97.949MHz)
Minimum input arrival time before clock:	No path found
Maximum output required time after clock:	4.780ns
Maximum combinational path delay:	No path found

3 Power Results

Total estimated power consumption: 203 (mW).

FPGA Device Package	Type of Input Stimulus	Area (in LUT's)	Delay (ns)	Speed (MHz)	Power (mW)
SPARTAN XC3S1600E FG484	Input Stimulus 32X32	2406/8%	7.784	128.461	202.72
	Input Stimulus 64X64	10128/34%	10.209	97.949	203

Table 5.4: Synthesis results of input stimulus

5.1.3.3 COMPARATOR

Comparator compares the output result n (127:0) with fixed reg. Result m (127:0). If both results match then flag becomes high (1) otherwise it becomes low (0). If results match then circuit is fault free. BIST is working properly.

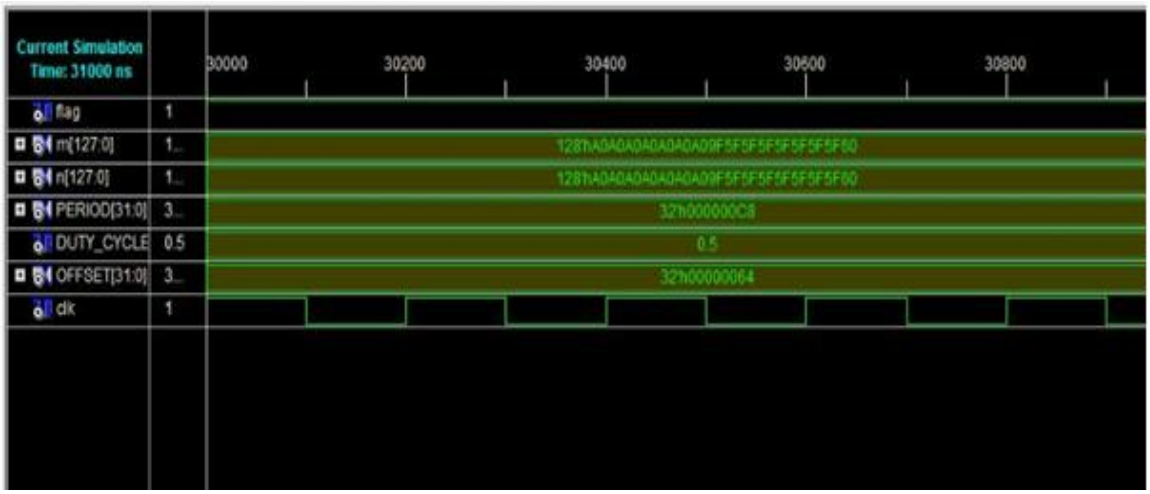


Figure 5.6: Simulation result of (final BIST capability) comparator

1 Synthesis Results

Selected Device: 3s1600efg484-5		
Number of Slices:	6158 out of 14752	41%
Number of Slice Flip Flops:	7115 out of 29504	24%
Number of 4 input LUTs:	10161 out of 29504	34%
Number used as logic:	9627	
Number used as Shift registers:	534	
Number of IOs:	258	
Number of bonded IOBs:	258 out of 376	68%
Number of GCLKs:	1 out of 24	4%

2 Timing Results

Speed Grade: -5	
Minimum period:	10.209ns (Maximum Frequency: 97.949MHz)
Minimum input arrival time before clock:	No path found
Maximum output required time after clock:	4.063ns

Maximum combinational path delay: No path found

3 Power Results

Total estimated power consumption: 203 (mW).

FPGA Device Package	Type of Comparator	Area (in LUT's)	Delay (ns)	Speed (MHz)	Power (mW)
SPARTAN XC3S1600E FG484	COMPARATOR 32X32	2423/8%	7.784	128.461	202.72
	COMPARATOR 64X64	10161/34%	10.209	97.949	203

Table 5.5: Synthesis results of (final BIST capability) comparator

From the given results we see that when we go to 32x32 bits multiplication to 64x64 bits multiplication (as hardware increase) the speed of multiplication not decrease in same ratio and not more power consumed.

5.2 CONCLUSION

1 The designs of 16x16 bits, 32x32 bits and 64x64 bits Vedic multiplier have been implemented on Spartan XC3S500-5-FG320 and XC3S1600-5-FG484 device. The computation delay for 16x16 bits Booth multiplier was 20.09 ns and for 16x16 bits Vedic multiplier was 6.960 ns. Also computation delays for 32x32 bits and 64x64 bits Vedic multiplier was obtained 7.784 ns and 10.241 ns respectively. It is therefore seen that the Vedic multipliers are much more faster than the conventional multipliers. The algorithms of Vedic mathematics are much more efficient than of conventional mathematics.

FPGA Device Package	Type of Multiplier	Area (in LUT's)	Delay (ns)	Speed (MHz)	Power (mW)
SPARTAN XC3S500E FG320	BOOTH MUL 16X16	618	20.09	49.76	40
	VEDIC MUL 16X16	641	6.960	143.699	81
	VEDIC MUL 32X32	2665	7.784	128.461	81
SPARTAN XC3S1600E FG484	VEDIC MUL 32X32	2665	7.784	128.461	132
	VEDIC MUL 64X64	10609	10.241	97.647	203

Table 5.6: Comparison of Multipliers

2 Urdhva tiryakbhyam, Nikhilam and Anurupye sutras are such algorithms which can reduce the delay, power and hardware requirements for multiplication of numbers.

3 FPGA implementation of this multiplier shows that hardware realization of the Vedic mathematics algorithms is easily possible.

4 Built in Self Tests of the implemented algorithms further proves that the design of Vedic multiplier circuit is fault free.

5.3 Future work: Vedic Mathematics, developed about 2500 years ago, gives us a clue of symmetric computation. Vedic mathematics deals with various topics of mathematics such as basic arithmetic, geometry, trigonometry, calculus etc. All these methods are very efficient as far as manual calculations are concerned.

If all those methods effectively implement hardware, it will reduce the computational speed drastically. Therefore, it could be possible to implement a complete ALU using all these methods using Vedic mathematics methods. Vedic mathematics is long been known but has not been implemented in the DSP and ADSP processors employing large number of multiplications in calculating the various transforms like FFTs and the IFFTs. By using these ancient Indian Vedic mathematics methods world can achieve new heights of performance and quality for the cutting edge technology devices.

REFERENCES

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

[2] Himanshu Thapliyal and Hamid R. Arabnia, "A Time-Area- Power Efficient Multiplier and Square Architecture Based On Ancient Indian Vedic Mathematics", Department of Computer Science, The University of Georgia, 415 Graduate Studies Research Center Athens, Georgia 30602-7404, U.S.A.

[3] E. Abu-Shama, M. B. Maaz, M. A. Bayoumi, "A Fast and Low Power Multiplier Architecture", The Center for Advanced Computer Studies, The University of Southwestern Louisiana Lafayette, LA 70504.

[4] 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.

[5] Shamim Akhter, "VHDL Implementation of Fast NXN Multiplier Based on Vedic Mathematics", Jaypee Institute of Information Technology University, Noida, 201307 UP, INDIA, 2007 IEEE.

[6] 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.

[7] 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.

[8] Shripad Kulkarni, "Discrete Fourier Transform (DFT) by using Vedic Mathematics", report, vedicmathsindia.blogspot.com, 2007.

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

[10] 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", Centre for VLSI and Embedded System Technologies, International Institute of Information Technology, Hyderabad, 500019, India, 2005 IEEE.

[11] Himanshu Thapliyal and M.B Srinivas, "VLSI Implementation of RSA Encryption System Using Ancient Indian Vedic Mathematics", Center for VLSI and Embedded System Technologies, International Institute of Information Technology Hyderabad-500019, India.

[12] Abhijeet Kumar, Dilip Kumar, Siddhi, "Hardware Implementation of 16*16 bit Multiplier and Square using Vedic Mathematics", Design Engineer, CDAC, Mohali.

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

[14] "Spartan-3E FPGA Starter Kit Board User Guide", UG230 (v1.1) June 20, 2008.

[15] 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.

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

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

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

[19] R. Bencivenga, T. J. Chakraborty and S. Davidson, "The Architecture of the Gentest Sequential Test Generator", in Proc. of the Custom Integrated Circuits Conference, pp. 17.1.1–17.1.4, May 1991.

[20] E. B. Eichelberger, E. Lindbloom, J. A. Waicukauski and T. W. Williams, "Structured Logic Testing", Englewood Cliffs, New Jersey, Prentice-Hall, 1991.

[21] M. Abramovici and C. Stroud, "BIST-Based Test and Diagnosis of FPGA Logic Blocks", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol.9, No.1, pp159-172, 2001.

[22] T. W. Williams and K. Parker, "Design for Testability – A Survey", IEEE Trans. On Computers, Vol. C-31, No. 1, pp. 2–15, January 1982.

[23] Alireza Sarvi, Dr. San Jose and Jenny Fan, "Automated BIST-based Diagnostic Solution for SOPC", ieeexplore.ieee.org/iel5/11202/36064/01708692.