# HIGH SPEED LOW POWER MULTIPLIERS BASED ON REVERSIBLE LOGIC METHODS

**A Project Report submitted in Fulfillment**

**of the Requirements for the Award of the Degree of BACHELOR OF TECHNOLOGY**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

Submitted by

E.Pravallika - 20X41A0473

B.Sandeep - 20X41A0465

G.Praveen - 20X41A0474

J.Aravind - 20X41A0479

T.Karthik - 20X41A04B1

Under the Guidance of

Ms. N. Kalavathi Assistant Professor



**Department of Electronics and Communication Engineering**

**S.R.K INSTITUTE OF TECHNOLOGY**

**Enikepadu, Vijayawada-521108 Andhra Pradesh**

**May 2024**

# CERTIFICATE

This is to certify that the project report entitled “**HIGH SPEED LOW POWER MULTIPLIERS BASED ON REVERSIBLE LOGIC METHODS**” is a bonafide record of work carried out by **E. Pravallika (20X4A0473), B. Sandeep (20X41A0465), G. Praveen (20X41A0474), J. Aravind (20X41A0479), T. Karthik (20X41A0B1).** under my guidance and supervision in partial fulfillment of the requirements for award of the degree of Bachelor of Technology in ***Electronics and Communication Engineering*** *of* ***Jawaharlal Nehru Technological University, Kakinada.***

#### N. Kalavathi Dr. S. Sri Gowri

(PROJECT GUIDE) (HEAD OF THE DEPARTMENT)

# ACKNOWLEDGEMENT

First, we sincerely salute our esteemed institution **SRK Institute of Technology** for giving this golden opportunity for fulfilling our warm dreams of becoming engineers.

We hereby express our sincere gratitude to our principal, **Dr. M. Ekambaram Naidu,** who has rendered his constant encouragement and valuable suggestions making our goals successful.

We are also thankful to our Head of Department **Dr. S. Sri Gowri**, for her constant encouragement and invaluable support for the project.

We are glad to express our gratitude to **Ms. N. Kalavathi,** Assistant professor, our guide, for her guidance and cooperation in completing this project successfully.

We are sincerely thankful to all the faculty members of Electronics and Communication Department for their co-operation, encouragement.

We thank everyone who has rendered help directly or indirectly in the completion of this project successfully.

#### Project Members:

* E. Pravallika - 20X41A0473
* B. Sandeep - 20X41A0465
* G. Praveen - 20X41A0474
* J. Aravind - 20X41A0479
* T. Karthik - 20X41A04B1

CONTENT

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE NO** |
| List of Figures |  | i |
| Abstract |  | ii |
| **Chapter 1:** | **INTRODUCTION** | 1 to 4 |
| **Chapter 2:** | **LITERATURE REVIEW** | 5 to 8 |
| **Chapter 3:** | **PROPOSED METHOD** | 9 to 15 |
| 3.1 | Reversible Logic gates | 10 |
| 3.2 | Implementation of Reversible gate | 10 to 13 |
| 3.3 | Implementation Wallace Tree multiplier and GCD Processor | 13 to 15 |
| **Chapter 4:** | **SOFTWARE SPECIFICATIONS** | 16 to 33 |
| **4.1** | History of Verilog | 18 |
| **4.2** | Introduction | 18 |
| **4.3** | Design Styles | 18 |
| 4.3.1 | Bottom-up Design | 18 |
| 4.3.2 | Top-up Design | 18 |
| 4.3.3 | Features of Verilog HDL | 18 |
| **4.4** | **VLSI Design Flow** | 18 |
| **4.5** | **System Specifications** | 19 |
| 4.5.1 | Architecture Design | 19 |
| 4.5.2 | Behavioral or Functional Design | 19 |
| 4.5.3 | Logic Design | 20 |
| 4.5.4 | Circuit Design | 20 |
| 4.5.5 | Physical Design | 20 |
| 4.5.6 | Layout Verification | 20 |
| 4.5.7 | Fabrication and Testing | 20 |
| **4.6** | **Module** | 20 |
| 4.6.1 | Instances | 21 |
| 4.6.2 | Ports | 21 |
| 4.6.3 | Identifiers | 21 |
| 4.6.4 | Data Types | 22 |
| 4.6.5 | Register Data Types | 22 |
| **4.7** | **Modeling Concepts** | 22 |
| 4.7.1 | Abstraction Levels | 22 |
| 4.7.2 | Behavioral or Algorithmic Level | 22 |
| 4.7.3 | Register-Transfer Level | 23 |
| 4.7.4 | Gate Level | 23 |
| **4.8** | **Operators** | 23 |
| 4.8.1 | Arithmetic Operators | 23 |
| 4.8.2 | Relational Operators | 24 |
| 4.8.3 | Bit-Wise Operators | 24 |
| 4.8.4 | Logical Operators | 25 |
| 4.8.5 | Reduction Operators | 25 |
| 4.8.6 | Shift Operators | 26 |
| 4.8.7 | Concatenation Operators | 26 |
| 4.8.8 | Operator Precedence | 27 |
| 4.8.9 | Switch Level | 27 |
| **4.9** | **Getting Started with Tool** | 27 |
| 4.9.1 | Introduction | 27 |
| 4.9.2 | Creating a New Project | 28 |
| 4.9.3 | Opening Designs | 29 |
| 4.9.4 | Configuring Project Settings | 31 |
| 4.9.5 | Synthesis and Implementation of the Design | 31 |
| **4.10** | **Xilinx Vivado Simulation Procedure** | 32 |
| **Chapter 5:** | **RESULTS** | **35** |
| **Chapter 6:** | **ADVANTAGES AND APPICATIONS** | **39** |
| **6.1** | **Advantages** | **40** |
| **6.2** | **Applications** | **40** |
| **Chapter 7:** | **CONCLUSION** | **41** |

**REFERENCES** 43

# LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **NAME** | **PAGE NO** |
| Figure 1 | Reversal gate | 10 |
| Figure 2 | Feynman gate | 11 |
| Figure 3 | Toffoli gate | 11 |
| Figure 4 | Peres gate | 11 |
| Figure 5 | Fredkin gate | 12 |
| Figure 6 | HNG gate | 12 |
| Figure 7 | 4\*4 reversible Multiplier circuit using HNG gates and peres gate | 12 |
| Figure 8 | Reversible adder circuit using HNG gate | 13 |
| Figure 9 | Reversible Adder | 13 |
| Figure 10 | Irreversible Adder | 13 |
| Figure 11 | Flow diagram of Wallace tree diagram | 14 |
| Figure 12 | Dot diagram of 16-Bit Wallace tree Multipiler | 14 |
| Figure 13 | Block diagram of GCD processor | 15 |
| Figure 14 | DFT design flow | 19 |
| Figure 15 | Relational operators | 24 |
| Figure 16 | Bit-wise operators | 24 |
| Figure 17 | Logical operators | 25 |
| Figure 18 | Reduction operators | 26 |
| Figure 19 | Swift operators | 26 |
| Figure 20 | Operator precedence | 27 |
| Figure 21 | New Project Wizard | 28 |
| Figure 22 | Implemantation Design | 29 |
| Figure 23 | Settings Dialog Box | 31 |
| Figure 24  Figure 25  Figure 26  Figure 27  Figure 28  Figure 29  Figure 30  Figure 31  Figure 32  Figure 33  Figure 34  Figure 35 | Schematic Window  ⅰ  Project Summary  RTL Schematic Wallace tree Multiplier  Simulation of Wallace tree Multiplier  Area of Wallace tree Multipler  Time Delay of Wallace tree Multipler  Power of Wallace tree Multipler  RTL Schematic of Vedic Multiplier  Simulation of Vedic Multiplier  Area of Vedic Multiplier  Time Delay of Vedic Multiplier  Power of Vedic Multiplier | 32  33  35  35  35  36  36  37  37  38  38  38 |

ⅱ

# ABSTRACT

The aim of this research is on multipliers with higher speed and a lower-complexity. Reversible logic has recently found extensive application in multiple areas including multipliers, low-power CMOS circuits, optical data processing, and quantum circuits. Here, reversible logic is used to create efficient multipliers that are constructed with Vedic mathematics and Wallace tree architectures. A particularly nice thing about these proposed multipliers is that they all utilize known logical principles, such as the reversible Half Adder, the reversible Full Adder, the Dual Key Gate, RG Gate. As a means of guaranteeing better performance, multiple multipliers are distinguished according to reversible logic circuits. Simulated and synthesized using the Xilinx Vivado , the results of the proposed design are investigated.

**Keywords:** Full Adder, Half adder, Reversible Logic, Vedic mathematics, Xilinx vivado

**CHAPTER 1**

**INTRODUCTION**

Reversible logic gates are digital logic gates that operate in a way that enables the recovery of both input and output from the gates. Unlike traditional logic gates, which are non-reversible, reversible logic gates have the property that the input can be uniquely determined from the output, and vice versa.

In reversible computing, it is essential to conserve energy and minimize heat generation. Reversible logic gates offer a way to achieve this goal, as they can perform computations without dissipating energy as heat. This property has made reversible logic gates a crucial component in the development of low-power electronic circuits, quantum computing, and other areas of advanced computing.

In reversible logic gates, each gate has an equal number of input and output bits, and the number of input bits is equal to the number of output bits. Also, the reversible gates satisfy a fundamental property known as the reversibility principle, which states that the mapping from input to output is bijective, i.e., it is a one-to-one and onto function.

Some commonly used reversible logic gates include Toffoli gate, Fredkin gate, and Peres gate, among others. These gates can be combined to implement more complex reversible circuits that can perform a wide range of logical operations.

Reversible computing was started when the basis of thermodynamics of information processing was shown that conventional irreversible circuits unavoidably generate heat because of losses of information during the computation. The different physical phenomena can be exploited to construct reversible circuits avoiding the energy losses. One of the most attractive architecture requirements is to build energy lossless, small and fast quantum computers. Most of the gates used in digital design are not reversible for example NAND, OR and EXOR gates.

A Reversible circuit/gate can generate unique output vector from each input vector, and vice versa, i.e., there is a one-to-one correspondence between the input and output vectors. Thus, the number of outputs in a reversible gate or circuit has the same as the number of inputs, and commonly used traditional NOT gate is the only reversible gate. Each Reversible gate has a cost associated with it called Quantum cost. The Quantum cost of a Reversible gate is the number of 2\*2 Reversible gates or Quantum logic gates required in designing. One of the most important features of a Reversible gate is its garbage output i.e., every input of the gate which is not used as input to other gate or as a primary output is called garbage output.

In digital design energy loss is considered as an important performance parameter. Part of the energy dissipation is related to non-ideality of switches and materials. Higher levels of integration and new fabrication processes have dramatically reduced the heat loss over the last decades. The power dissipation in a circuit can be reduced by the use of Reversible logic. Landauer’s principle states that irreversible computations generate heat of K\*Tln2 for every bit of information lost, where K is Boltzmann’s constant and T the absolute temperature at which the computation performed. Bennett showed that if a computation is carried out in Reversible logic zero energy dissipation is possible, as the amount of energy dissipated in a system is directly related to the number of bits erased during computation. The design that does not result in information loss is irreversible.

A set of reversible gates are needed to design reversible circuit. Several such gates are proposed over the past decades. Arithmetic circuits such as Adders, Sub tractors, Multipliers and Dividers are the essential blocks of a Computing system. Dedicated Adder/Sub tractor circuits are required in a number of Digital Signal Processing applications. Several designs for binary Adders and Sub tractors are investigated based on Reversible logic. Minimization of the number of Reversible gates, Quantum cost and garbage inputs/outputs are the focus of research in Reversible logic.

Pawel Kerntopf explained multipurpose Reversible gates and example of efficient binary multipurpose reversible gates. Thapliyal and Ranganathan proposed the design of Reversible Binary Sub tractor using TR Gate. The particular function like Binary Subtraction is implemented using TR gate effectively by reducing number of Reversible gates, Garbage outputs and Quantum Cost. Thapliyal and Ranganathan presented a design of Reversible latches viz., D Latch, JK latch, T latch and SR latch that are optimized in terms of quantum cost, delay and garbage outputs. Lihui Ni et al., described general approach to construct the Reversible full adder and can be extended to a variety of Reversible full-adders with only two Reversible gates. Irina Hashmi and Hafiz Hasan Babu designed an efficient reversible barrel shifter which is capable of left shift/rotate used for high speed ALU applications.

Robert Wille et al., explored two techniques from irreversible equivalence checking applied in the reversible circuit domain. (i) Decision diagram Technique equivalence checking for quantum circuits and (ii) Boolean satisfiability checking for garbage input/outputs. Noor Muhammed Nayeem et al.,presented designs of Reversible shift registers such as serial-in serial-out, serial-in parallel-out, parallel-in serialout, parallel-in parallel-out and universal shift registers. Majid Mohammadi, Mohammad Eshghi et al., proposed a synthesis method to realize a Reversible Binary Coded Decimal adder/subtractor circuit. Genetic algorithms and don’t care concepts used to design and optimize all parts of a Binary Coded Decimal adder circuit in terms of number of garbage inputs/outputs and quantum cost.

Majid Mohammadi and Mohammad Eshghi explained about the behavioral description and synthesis of quantum gates. To synthesize reversible logic circuits, V and V+ gates are shown in the truth table form and shown that bigger circuits with a greater number of gates can be synthesized. Rekha James et al., proposed an implementation of Binary Coded Decimal adder in Reversible logic, which is basis of ALU for reversible CPU.

VLSI implementations using one type of building block can decrease system design and manufacturing cost. Himanshu Thapliyal and Vinod presented the Transistor realization of a new 4\*4 Reversible TSG gate. The gate alone operates as a Reversible full adder. The Transistor realizations of 1-bit Reversible full adder, ripple carry adder and carry skip adder are also discussed. Himanshu Thapliyal and Srinivas proposed a 3x3 Reversible TKS gate with two of its outputs working as 2:1 multiplexer.

The gate used to design a Reversible half adder and further used to design multiplexer based Reversible full adder. The multiplexer based full adder is further used to design Reversible 4x4 Array and modified Baugh Woolley multipliers Yvan Van Rentergem and Alexis De Vos presented four designs for Reversible full-adder circuits and the implementation of these logic circuits into electronic circuitry based on C-MOS technology and pass-transistor design. The chip containing three different Reversible full adders are discussed. Mozammel Khan proposed realizations of ternary half and full-adder circuits using generalized ternary gates. Mozammel Khan discussed quantum realization of ternary Toffoli gate which requires fewer gates than the existing literature.

Abhinav Agrawal and Niraj Jha presented 6 the first practical synthesis algorithm and tool for Reversible functions with a large number of inputs. It uses positive-polarity Reed-Muller decomposition at each stage to synthesize the function as a network. In digital design energy loss is considered as an important performance parameter. Part of the energy dissipation is related to non-ideality of switches and materials. Higher levels of integration and new fabrication processes have dramatically reduced the heat loss lover the last decades. The power dissipation in a circuit can be reduced by the use of reversible logic.

According to Landaulet’s principle states that irreversible computations generate heat of K\*Tln2 for every bit of information lost, where K is Boltzmann’s constant and T the absolute temperature at which the computation performed. Bennett showed that if a computation is carried out in reversible logic zero energy dissipation is possible, as the amount of energy dissipated in a system is directly related to the number of bits erased during computation. The design that does not results in information loss is irreversible. A set of reversible gates are needed to design reversible circuit. Several such gates are proposed over the past decades

**CHAPTER 2**

**LITERATURE REVIEW**

*[1]. C. H. Bennett, "Notes on the history of reversible computation," in IBM Journal of Research and Development, vol. 32, no. 1, pp. 16-23, Jan. 1988.*

We review the history of the thermodynamics of information processing, beginning with the paradox of Maxwell's demon; continuing through the efforts of Szilard, Brillouin, and others to demonstrate a thermodynamic cost of information acquisition; the discovery by Landauer of the thermodynamic cost of information destruction; the development of the theory of and classical models for reversible computation; and ending with a brief survey of recent work on quantum reversible computation.

*[2]. H. Thapliyal and N. Ranganathan, "Design of Efficient Reversible Binary Subtractors Based on a New Reversible Gate," 2009 IEEE Computer Society Annual Symposium on VLSI, Tampa, FL, USA, 2009, pp. 229-234.*

Reversible logic has extensive applications in quantum computing, low power VLSI design, quantum dot cellular automata and optical computing. While several researchers have investigated the design of reversible logic elements, there is not much work reported on reversible binary subtractors. In this paper, we propose the design of a new reversible gate called TR gate. Further, we investigate the design of reversible binary subtractors based on the proposed TR gate. The proposed TR gate is better for designing reversible binary subtractor compared to such gates discussed in literature in terms of quantum cost, garbage outputs and complexity of gates.

*[3] H. Thapliyal and N. Ranganathan, "Design of Reversible Latches Optimized for Quantum Cost, Delay and Garbage Outputs," 2010 23rd International Conference on VLSI Design, Bangalore, India, 2010, pp. 235-240.*

Reversible logic has extensive applications in emerging nanotechnologies, such as quantum computing, optical computing, ultra low power VLSI and quantum dot cellular automata. In the existing literature, designs of reversible sequential circuits are presented that are optimized for the number of reversible gates and the garbage outputs. The optimization of the number of reversible gates is not sufficient since each reversible gate is of different computational complexity, and thus will have a different quantum cost and delay. While the computational complexity of a reversible gate can be measured by its quantum cost, the delay of a reversible gate is another parameter that can be optimized during the design of a reversible sequential circuit. In this work, we present novel designs of reversible latches that are optimized in terms of quantum cost, delay and the garbage outputs. The optimized designs of reversible latches presented in this work are the D Latch, JK latch, T latch and SR latch.

*[4] L. Ni, Z. Guan and W. Zhu, "A General Method of Constructing the Reversible Full-Adder," 2010 Third International Symposium on Intelligent Information Technology and Security Informatics, Jian, China, 2010, pp. 109-113.*

The reversible gates, attracting people's attention increasingly, have been widely used in low-power CMOS design, optical computing and quantum computing. In many existing literatures, only the methods of constructing certain specific reversible full-adders were presented, while we proposed a general approach to construct the reversible full-adder. According to the approach, we can realize a variety of reversible full-adders flexibly with only two reversible gates and two garbage outputs, which have improvements in the gate count and garbage count and can reduce the cost of network.

*[5] N. M. Nayeem, M. A. Hossain, L. Jamal and H. M. H. Babu, "Efficient Design of Shift Registers Using Reversible Logic," 2009 International Conference on Signal Processing Systems, Singapore, 2009, pp. 474-478.*

Reversible shift registers are required to construct reversible memory circuits. This paper presents novel designs of reversible shift registers such as serial-in serial-out (SISO), serial-in parallel-out (SIPO), parallel-in serial-out (PISO), parallel-in parallel-out (PIPO) and universal shift registers. In order to show the efficiency, lower bounds of the proposed designs are shown in terms of number of gates required, garbage outputs produced and quantum cost needed. As far as it is known, this is the first attempt to apply reversible logic to implement shift registers (except SISO). Appropriate theorems and lemmas are presented to clarify the proposed designs. The contribution of this paper will engender a new thread of research in the field of reversible sequential circuits.

*[6] R. K. James, T. K. Shahana, K. P. Jacob and S. Sasi, "A New Look at Reversible Logic Implementation of Decimal Adder," 2007 International Symposium on System-on-Chip, Tampere, Finland, 2007, pp. 1-4.*

Reversibility plays a fundamental role when computations with minimal energy dissipation are considered. In recent years, reversible logic has emerged as one of the most important approaches for power optimization with its application in low power CMOS, quantum computing and nanotechnology. This research proposes a new implementation of Binary Coded Decimal (BCD) adder in reversible logic. The design reduces the number of gates and garbage outputs compared to the existing BCD adder reversible logic implementations. So, this design gives rise to an implementation with a reduced area and delay.

*[7] H. Thapliyal and A. P. Vinod, "Transistor Realization of Reversible TSG Gate and Reversible Adder Architectures," APCCAS 2006 - 2006 IEEE Asia Pacific Conference on Circuits and Systems, Singapore, 2006, pp. 418-421.*

Reversible logic is emerging as a promising technology with applications in design of low power arithmetic and data path units for digital signal processing (DSP), quantum computing, nanotechnology, and optical computing. In this paper, the transistor realization of a new 4\*4 reversible gate called "TSG" gate is presented. The proposed TSG gate has the ability to operate as a reversible full adder i.e., reversible full adder is implemented using a single gate. The transistor realizations of 1-bit reversible full adder, reversible ripple carry adder and reversible carry skip adder are also presented. In order to have the reduced transistor overhead of reversible carry skip adder, its modified design has been proposed. We also demonstrate a method to minimize the overhead in transistor implementation of reversible arithmetic units. The transistor implementation of reversible arithmetic circuits presented in this paper finds extensive applications in computationally intensive DSP tasks.

*[8] H. Thapliyal and M. B. Srinivas, "Novel design and reversible logic synthesis of multiplexer based full adder and multipliers," 48th Midwest Symposium on Circuits and Systems, 2005., Covington, KY, USA, 2005, pp. 1593-1596 Vol. 2.*

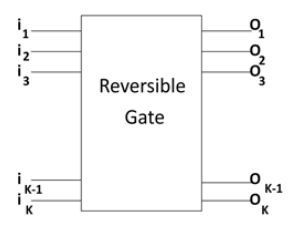
#### Quantum arithmetic must be built from reversible logic components. This is the driving force for the proposed novel 3/spl times/3 reversible gate termed TKS gate having two of its outputs working as 2:1 multiplexer. The proposed TKS gate is used to design a reversible half adder and is further used to design multiplexer based reversible full adder. The multiplexer based full adder is further used to design reversible 4/spl times/4 Array and modified Baugh Wooley multipliers. A novel 4/spl times/4 multiplier architecture with reversible logic is also proposed in which the partial products can be generated in parallel and their additions are reduced to logarithmic steps. In the proposed multiplier, all the operations are decomposed into levels, thereby significantly reducing the power consumption through a control circuitry which will switch off those levels which are not active. Thus, this work provides the initial threshold to building of complex systems which can execute more complicated operations. The reversible circuits designed and proposed in this paper form the basis for an ALU of a primitive quantum.

**CHAPTER 3**

**PROPOSED METHOD**

## Reversible Logic gate

A reversible logic gate is an n-input n-output logic device with one-to-one mapping. This helps to determine the outputs from the inputs and also the inputs can be uniquely recovered from the outputs. Also, in the synthesis of reversible circuits direct fan-Out is not allowed as one–to-many concept is not reversible. However, fan-out in reversible circuits is achieved using additional gates. A reversible circuit should be designed using minimum number of reversible logic gates. From the point of view of reversible circuit design, there are many parameters for determining the complexity and performance of circuits.



**Fig.1**: Reversible Gate

A gate with k inputs and k outputs is called k\*k gate. The gate/circuit that does not loose information is called reversible. The input vector and output vector of a reversible gate is as shown in equations (1) & (2).

Input Vector Ik = (I1, I2, I3, …, Ik-1, Ik) . . . . . .. (1)

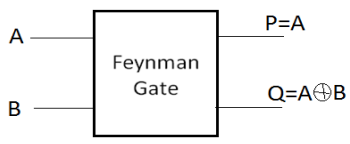
\Output Vector Ok = (O1, O2, O3, …, Ok-1, Ok) . . . . . .. (2)

### Implementation of Reversible gate

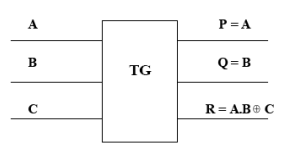
Reversible computation is related to other emerging technologies such as quantum computation optical computing and nanotechnologies that use a similar or slightly extended set of gates First implementations and fabrications of reversible logic in CMOS technology have also been accomplished. These exploit that reversible logic is particularly suitable when it comes to reuse of signal energy (in contrast to static CMOS logic that sinks the signal energy with each gate), and, when using adiabatic switching to switch transistors in a more energy efficient way.

In fact, reversible circuits have shown that such implementations have the potential to reduce energy consumption by a factor. A drawback of these implementations comes from another law related to transistors, namely that the energy consumption is directly related to the execution frequency. If one performs many computations every second, the energy consumption per computation rises. Performing fewer computations lowers the energy consumption per computation. Of course, this implies that not all applications are necessarily suited for implementation using reversible circuits. However, many embedded devices do not need to perform billions of computations every second. In the rest of this section will focus on how to implement reversible gates in CMOS. First, we brief review some basics of CMOS transistor implementation as used in this work, and afterward we explain how this is used in an implementation of reversible gates.

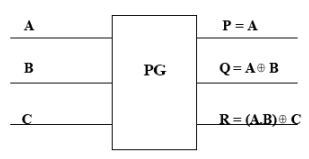
The following are various types of reversible gates



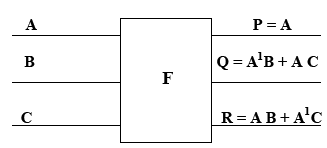
**Fig.2**: Feynman Gate



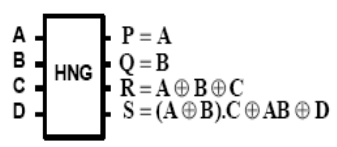
**Fig.3**: Toffoli Gate



**Fig.4**: Peres Gate

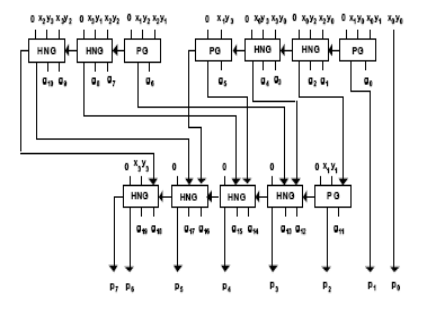


**Fig.5**: Fredkin Gate



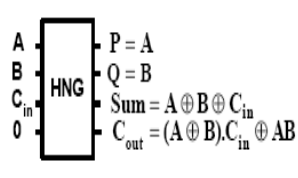
**Fig.6**: HNG Gate

Below shown figure 7 depicts a 4x4 reversible multiplier in which we have used only HNG and Peres Gate.

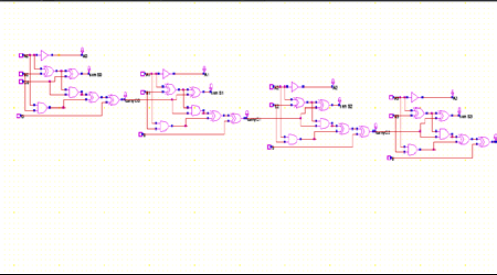


**Fig.7**: 4x4 reversible multiplier circuit using HNG gates and Peres gates

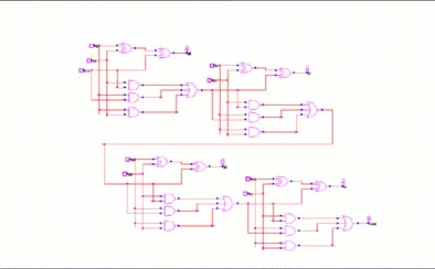
The basic cell for such a multiplier is a full adder (FA) accepting three bits. We use HNG gates as reversible full adder which is depicted in Fig. 8. The proposed reversible multiplier circuit uses eight reversible HNG full adders. In addition, it needs four reversible half adders. It is possible to use HNG gate as half adder, but we use Peres gate as reversible half adder because it has less hardware complexity and quantum cost compared to the HNG gate.



**Fig.8**: Reversible adder circuit using HNG gates



**Fig.9**: Reversible Adder



**Fig.10**: Irreversible Adder

## Implementation of Wallace tree Multiplier and GCD processor

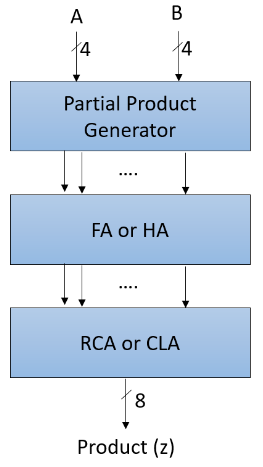
## Wallace tree Multiplier

The Wallace tree multiplier is a hardware implementation of a digital multiplier, used to perform fast multiplication of two binary numbers. It was first proposed by C.S. Wallace in 1964 and has since become a popular technique for implementing high-speed multipliers in digital circuits.

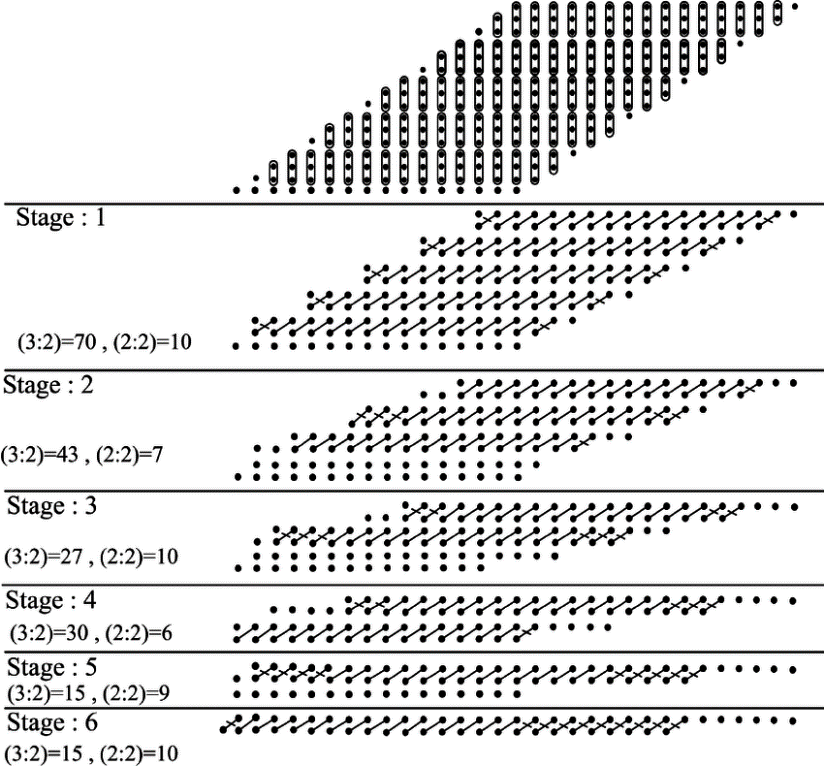
The basic idea behind the Wallace tree multiplier is to split the two numbers being multiplied into smaller, partial products, which are then combined using a series of binary adders. The partial products are generated using a combination of AND gates and half adders, and the final product is obtained by summing

The advantage of the Wallace tree multiplier over other multiplier architectures is that it reduces the number of partial products that need to be added together, which reduces the overall complexity of the circuit and improves its speed. However, the implementation of a Wallace tree multiplier can be more complex than other multiplier architectures, such as the Booth multiplier or array multiplier.

Overall, the Wallace tree multiplier is a powerful and efficient technique for implementing high-speed multipliers in digital circuits, and is commonly used in applications such as signal processing, digital filtering, and image processing.



**Fig.11**: Flow Diagram of Wallace Tree Multiplie

****

**Fig.12** : Dot Diagram of 16-bit Wallace Tree Multiplier

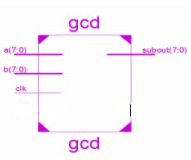
* 1. **GCD Processor**

A GCD (Greatest Common Divisor) processor is a type of digital circuit or hardware module that is designed to perform the calculation of the greatest common divisor of two integers. The GCD is the largest positive integer that divides both input integers without leaving a remainder.

GCD processors are commonly used in a variety of applications, such as error-correcting codes, cryptography, and signal processing. They can be implemented using different algorithms, including the Euclidean algorithm and the binary GCD algorithm.

The basic operation of a GCD processor involves taking two input integers and recursively computing their GCD until a solution is found. The output of the GCD processor is the final GCD value. Depending on the specific implementation, GCD processors can operate on fixed-point or floating-point numbers, and can be optimized for speed, area, or power consumption.

GCD processors can be implemented in software or hardware. In software, GCD algorithms can be coded using a programming language and run on a general-purpose processor. In hardware, GCD processors can be designed using dedicated digital circuits, such as field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs), for higher performance and lower power consumption.



**Fig. 13** Block Diagram of GCD Processor

**CHAPTER 4**

**SOFTWARE SPECIFICATIONS**

## 4.1 HISTORY OF VERILOG

Verilog was started initially as a proprietary hardware modeling language by Gateway Design Automation Inc. around 1984. It is rumored that the original language was designed by taking features from the most popular HDL language of the time, called HiLo, as well as from traditional computer languages such as C. At that time, Verilog was not standardized and the language modified itself in almost all the revisions that came out within 1984 to 1990.

Verilog simulator was first used beginning in 1985 and was extended substantially through 1987. The implementation was the Verilog simulator sold by Gateway. The first major extension was Verilog-XL, which added a few features and implemented the infamous "XL algorithm" which was a very efficient method for doing gate-level simulation.

The time was late 1990. Cadence Design System, whose primary product at that time included thin film process simulator, decided to acquire Gateway Automation System. Along with other Gateway products, Cadence now became the owner of the Verilog language, and continued to market Verilog as both a language and a simulator.

At the same time, Synopsys was marketing the top-down design methodology, using Verilog. This was a powerful combination. In 1990, Cadence recognized that if Verilog remained a closed language, the pressures of standardization would eventually cause the industry to shift to VHDL. Consequently, Cadence organized the Open Verilog International (OVI), and in 1991 gave it the documentation for the Verilog Hardware Description Language. This was the event which "opened" the language.

## INTRODUCTION

* HDL is an abbreviation of Hardware Description Language. Any digital system can be represented in a REGISTER TRANSFER LEVEL (RTL) and HDLs are used to describe this RTL.
* Verilog is one such HDL and it is a general-purpose language –easy to learn and use. Its syntax is similar to C.
* The idea is to specify how the data flows between registers and how the design processes the data.
* To define RTL, hierarchical design concepts play a very significant role. Hierarchical design methodology facilitates the digital design flow with several levels of abstraction.
* Verilog HDL can utilize these levels of abstraction to produce a simplified and efficient representation of the RTL description of any digital design.
* For example, an HDL might describe the layout of the wires, resistors and transistors on an

in terms of logical gates and flip flops in a digital system, i.e., the gate level. Verilog supports all of these levels.

## DESIGN STYLES:

Any hardware description language like Verilog can be design in two ways one is bottom-up design and other one is top-down design.

#### Bottom-Up Design:

The traditional method of electronic design is bottom-up (designing from transistors and moving to a higher level of gates and, finally, the system). But with the increase in design complexity traditional bottom-up designs have to give way to new structural, hierarchical design methods.

#### Top-Down Design:

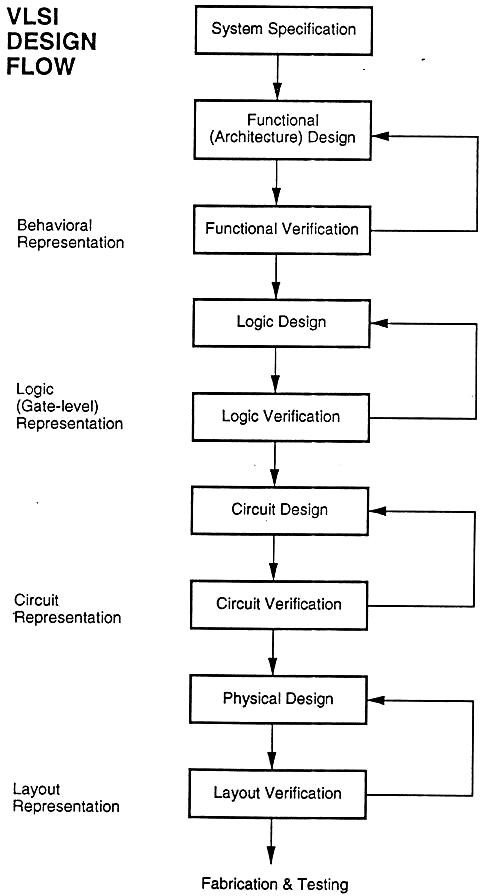
For HDL representation it is convenient and efficient to adapt this design-style. A real top-down design allows early testing, fabrication technology independence, a structured system design and offers many other advantages. But it is very difficult to follow a pure top-down design. Due to this fact most designs are mix of both the methods, implementing some key elements of both design styles.

#### Features of Verilog HDL

* + - 1. Verilog is case sensitive.
      2. Ability to mix different levels of abstract freely.
      3. One language for all aspects of design, testing, and verification.
      4. In Verilog, Keywords are defined in lower case.
      5. In Verilog, Most of the syntax is adopted from "C" language.
      6. Verilog can be used to model a digital circuit at Algorithm, RTL, Gate and Switch level.
      7. There is no concept of package in Verilog.
      8. It also supports advanced simulation features like TEXTIO, PLI, and UDPs.

## VLSI DESIGN FLOW

The VLSI design cycle starts with a formal specification of a VLSI chip, follows a series of steps, and eventually produces a packaged chip.



### System Specification:

**Fig 14:** DFT Design Flow

The first step of any design process is to lay down the specifications of the system. System specification is a high level representation of the system. The factors to be considered in this process include: performance, functionality, and physical dimensions like size of the chip.

The specification of a system is a compromise between market requirements, technology and economic viability. The end results are specifications for the size, speed, power, and functionality of the VLSI system.

#### Architectural Design

The basic architecture of the system is designed in this step. This includes, such decisions as RISC (Reduced Instruction Set Computer) versus CISC (Complex Instruction Set Computer), number of ALUs, Floating Point units, number and structure of pipelines, and size of caches among others. The outcome of architectural design is a Micro-Architectural Specification (MAS).

#### Behavioral or Functional Design:

In this step, main functional units of the system are identified. This also identifies the interconnect requirements between the units. The area, power, and other parameters of each unit are estimated.

Modules. The key idea is to specify behavior, in terms of input, output and timing of each unit, without specifying its internal structure.

The outcome of functional design is usually a timing diagram or other relationships between units.

#### Logic Design:

In this step the control flow, word widths, register allocation, arithmetic operations, and logic operations of the design that represent the functional design are derived and tested.

This description is called Register Transfer Level (RTL) description. RTL is expressed in a Hardware Description Language (HDL), such as VHDL or Verilog. This description can be used in simulation and verification

#### Circuit Design:

The purpose of circuit design is to develop a circuit representation based on the logic design. The Boolean expressions are converted into a circuit representation by taking into consideration the speed and power requirements of the original design. Circuit Simulation is used to verify the correctness and timing of each component

The circuit design is usually expressed in a detailed circuit diagram. This diagram shows the circuit elements (cells, macros, gates, transistors) and interconnection between these elements. This representation is also called a netlist. And each stage verification of logic is done.

#### Physical design:

In this step the circuit representation (or netlist) is converted into a geometric representation. As stated earlier, this geometric representation of a circuit is called a layout.

Layout is created by converting each logic component (cells, macros, gates, transistors) into a geometric representation (specific shapes in multiple layers), which perform the intended logic function of the corresponding component. Connections between different components are also expressed as geometric patterns typically lines in multiple layers.

#### Layout verification:

Physical design can be completely or partially automated and layout can be generated directly from netlist by Layout Synthesis tools. Layout synthesis tools, while fast, do have an area and performance penalty, which limit their use to some designs. These are verified.

#### Fabrication and Testing:

Silicon crystals are grown and sliced to produce wafers. The wafer is fabricated and diced into individual chips in a fabrication facility. Each chip is then packaged and tested to ensure that it meets all the design specifications and that it functions properly.

## MODULE:

A module is the basic building block in Verilog. It can be an element or a collection of low level design blocks. Typically, elements are grouped into modules to provide common functionality used in places of the design through its port interfaces, but hides the internal implementation.

#### Syntax:

module<module name> (<module\_port\_list>);

…..

<module internals> //contents of the module

….

Endmodule

* + 1. **Instances**

A module provides a template from where one can create objects. When a module is invoked Verilog creates a unique object from the template, each having its own name, variables, parameters and I/O interfaces. These are known as *instances*.

#### Ports:

* + - 1. Ports allow communication between a module and its environment.
      2. All but the top-level modules in a hierarchy have ports.
      3. Ports can be associated by order or by name.

You declare ports to be input, output or inout. The port declaration syntax is: *Input* [range\_val:range\_var] list\_of\_identifiers; *output*[range\_val:range\_var] list\_of\_identifiers;

*inout*[range\_val:range\_var] list\_of\_identifiers;

#### Identifiers

* + - 1. Identifiers are user-defined words for variables, function names, module names, and instance names. Identifiers can be composed of letters, digits, and the underscore character.
      2. The first character of an identifier cannot be a number. Identifiers can be any length.
      3. Identifiers are case-sensitive, and all characters are significant.

An identifier that contains special characters, begins with numbers, or has the same name as a keyword can be specified as an escaped identifier*.* An escaped identifier starts with the backslash character(\) followed by a sequence of characters, followed by white space.

#### Keywords:

* + - 1. Verilog uses keywords to interpret an input file.
      2. You cannot use these words as user variable names unless you use an escaped identifier.
      3. Keywords are reserved identifiers, which are used to define language constructs.
      4. Some of the keywords are always, case, assign, begin, case, end and end case etc.

#### Data Types:

Verilog Language has two primary data types:

* + - 1. ***Nets*** - represents structural connections between components.
      2. ***Registers*** - represent variables used to store data. Every signal has a data type associated with it. Data types are:
      3. ***Explicitly declared*** with a declaration in the Verilog code.
      4. ***Implicitly declared*** with no declaration but used to connect structural building blocks in the code. Implicit declarations are always net type "wire" and only one bit wide.

#### Register Data Types

* + - 1. Registers store the last value assigned to them until another assignment statement changes their value.
      2. Registers represent data storage constructs.
      3. Register arrays are called memories.
      4. Register data types are used as variables in procedural blocks.
      5. A register data type is required if a signal is assigned a value within a procedural block
      6. Procedural blocks begin with keyword initial and always.

The data types that are used in register are register, integer, time and real.

## MODELING CONCEPTS:

#### Abstraction Levels:

* + - 1. Behavioral level
      2. Register-Transfer Level
      3. Gate Level
      4. Switch level

#### Behavioral or algorithmic Level

* + - 1. This level describes a system by concurrent algorithms (Behavioral).
      2. Each algorithm itself is sequential meaning that it consists of a set of instructions that are executed one after the other.
      3. The blocks used in this level are ‘initial’, ‘always’ ,‘functions’ and ‘tasks’ blocks
      4. The intricacies of the system are not elaborated at this stage and only the functional description of the individual blocks is prescribed.
      5. In this way the whole logic synthesis gets highly simplified and at the same time more efficient.

#### Register-Transfer Level:

* + - 1. Designs using the Register-Transfer Level specify the characteristics of a circuit by operations and the transfer of data between the registers.
      2. An explicit clock is used. RTL design contains exact timing possibility, operations are scheduled to occur at certain times.
      3. Modern definition of a RTL code is "Any code that is synthesizable is called RTL code".

#### Gate Level:

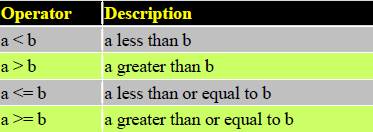
* + - 1. Within the logic level the characteristics of a system are described by logical links and their timing properties.
      2. All signals are discrete signals. They can only have definite logical values (`0', `1', `X', `Z`). The usable operations are predefined logic primitives (AND, OR, NOT etc gates).
      3. It must be indicated here that using the gate level modeling may not be a good idea in logic design.
      4. Gate level code is generated by tools like synthesis tools in the form of netlists which are used for gate level simulation and for backend.

## OPERATORS

Verilog provided many different operators types. Operators can be,

* Arithmetic Operators
* Relational Operators
* Bit-wise Operators
* Logical Operators
* Reduction Operators
* Shift Operators
* Concatenation Operator
* Conditional Operator
  + 1. **Arithmetic Operators**
       1. These perform arithmetic operations. The + and - can be used as either unary (-z) or binary (x-y) operators.
       2. Binary: +, -, \*, /, % (the modulus operator)
       3. Unary: +, - (This is used to specify the sign)
       4. Integer division truncates any fractional part
       5. If any operand bit value is the unknown value x, then the entire result value is x
       6. Register data types are used as unsigned values (Negative numbers are stored in two's complement form).
    2. **Relational Operators**

Relational operators compare two operands and return a single bit 1or 0. These operators synthesize into comparators. Wire and reg variables are positive Thus (-3’b001) = = 3’b111 and (-3d001)>3d1 10, however for integers -1<>



**Fig 15** : Relational Operators

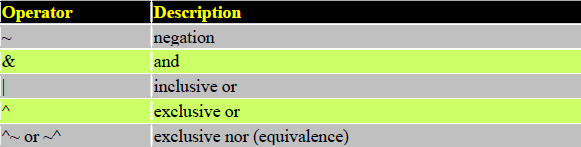
The result is a scalar value

* + - 1. 0 if the relation is false (a is bigger than b)
      2. 1 if the relation is true ( a is smaller than b)
      3. x if any of the operands has unknown x bits (if a or b contains X)

**Note:** If any operand is x or z, then the result of that test is treated as false (0)

* + 1. **Bit-wise Operators:**

Bitwise operators perform a bit wise operation on two operands. This take each bit in one operand and perform the operation with the corresponding bit in the other operand. If one operand is shorter than the other, it will be extended on the left side with zeroes to match the length of the longer operand.



**Fig 16** : Bit-Wise Operators Computations include unknown bits, in the following way:

-> ~x = x

-> 0&x = 0

-> 1&x = x&x = x

-> 1|x = 1

-> 0|x = x|x = x

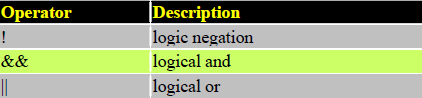
-> 0^x = 1^x = x^x = x

-> 0^~x = 1^~x = x^~x = x

When operands are of unequal bit length, the shorter operand is zero-filled in the most significant bit positions.

* + 1. **Logical Operators**

Logical operators return a single bit 1 or 0. They are the same as bit-wise operators only for single bit operands. They can work on expressions, integers or groups of bits, and treat all values that are nonzero as “1”. Logical operators are typically used in conditional (if ... else) statements since they work with expressions.

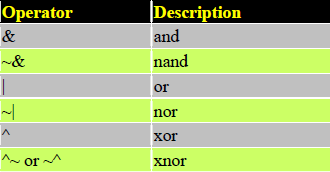


**Fig 17**: Logical Operators Expressions connected by && and || are evaluated from left to right Evaluation stops as soon as the result is known

The result is a scalar value:

* + - 1. 0 if the relation is false
      2. 1 if the relation is true
      3. x if any of the operands has x (unknown) bits
    1. **Reduction Operators**

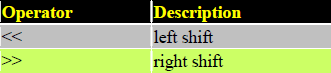
Reduction operators operate on all the bits of an operand vector and return a single-bit value. These are the unary (one argument) form of the bit-wise operators.



**Fig 18**: Reduction Operators

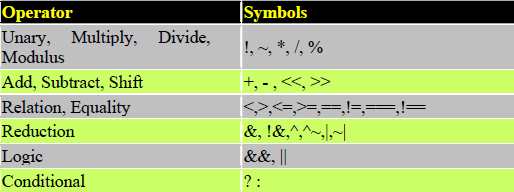
* + - 1. Reduction operators are unary.
      2. They perform a bit-wise operation on a single operand to produce a single bit result.
      3. Reduction unary NAND and NOR operators operate as AND and OR respectively, but with their outputs negated.
    1. **Shift Operators**

Shift operators shift the first operand by the number of bits specified by the second operand. Vacated positions are filled with zeros for both left and right shifts (There is no sign extension).



**Fig 19**: Shift Operators

* + - 1. The left operand is shifted by the number of bit positions given by the right operand.
      2. The vacated bit positions are filled with zeroes.
    1. **Concatenation Operator**
       1. The concatenation operator combines two or more operands to form a larger vector.
       2. Concatenations are expressed using the brace characters { and }, with commas separating the expressions within.
       3. ->Example: + {a, b[3:0], c, 4'b1001} // if a and c are 8-bit numbers, the results has 24 bits
       4. Un-sized constant numbers are not allowed in concatenations
    2. **Operator Precedence**



**Fig 20** : Operator Precedence

#### Switch Level:

This is the lowest level of abstraction. A module can be implemented in terms of switches, storage nodes and interconnection between them. However, as has been mentioned earlier, one can mix and match all the levels of abstraction in a design. RTL is frequently used for Verilog description that is a combination of behavioral and dataflow while being acceptable for synthesis.

### Getting Started with Tool

Frist we need to download and install Xilinx and ModelSim. These tools both have free student versions. Please accomplish Appendix B, C, and D in that order before continuing with this tutorial. Additionally if you wish to purchase your own Spartan3 board, you can do so at Digilent’s Website. Digilent offers academic pricing. Please note that you must download and install Digilent Adept software. The software contains the drivers for the board that you need and also provides the interface to program the board.

#### Introduction

Xilinx Tools is a suite of software tools used for the design of digital circuits implemented using Xilinx Field Programmable Gate Array (FPGA) or Complex Programmable Logic Device (CPLD). The design procedure consists of (a) design entry, (b) synthesis and implementation of the design, (c) functional simulation and (d) testing and verification. Digital designs can be entered in various ways using the above CAD tools: using a schematic entry tool, using a hardware description language (HDL) – Verilog or VHDL or a combination of both. In this lab we will only use the design flow that involves the use of Verilog HDL.

The CAD tools enable you to design combinational and sequential circuits starting with Verilog HDL design specifications. The steps of this design procedure are listed below:

1. Create Verilog design input file(s) using template driven editor.
2. Compile and implement the Verilog design file(s).
3. Create the test-vectors and simulate the design (functional simulation) without using a PLD (FPGA or CPLD).
4. Assign input/output pins to implement the design on a target device.
5. Download bitstream to an FPGA or CPLD device.
6. Test design on FPGA/CPLD device

A Verilog input file in the Xilinx software environment consists of the following segments:

***Header:*** module name, list of input and output ports.

***Declarations:*** input and output ports, registers and wires.

***Logic Descriptions:*** equations, state machines and logic functions.

***End:*** endmodule

All your designs for this lab must be specified in the above Verilog input format. Note that the *state diagram*

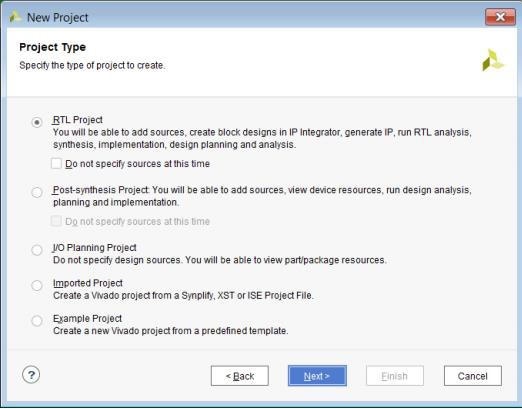
segment does not exist for combinational logic designs.

#### Programmable Logic Device: FPGA

In this lab digital designs will be implemented in the Basys2 board which has a Xilinx Spartan3E – XC3S250E FPGA with CP132 package. This FPGA part belongs to the Spartan family of FPGAs. These devices come in a variety of packages. We will be using devices that are packaged in 132 pin package with the following part number: XC3S250E-CP132.

#### Creating a New Project

Creating Projects You can use the New Project wizard to easily create different types of projects in the Vivado IDE. To open the New Project wizard, select File > New Project. This wizard enables you to specify a project location and name and create the types of projects shown in below figure



**Fig 21** : New Project Wizard—Project Type Page

**Project Name**: Write the name of your new project which is user defined.

**Project Location**: The directory where you want to store the new project in the specified location in one of your drive. In above window they are stored in location c drive which is not correct, the location of software and code should not be same location and Clicking on NEXT.

For each of the properties given below, click on the ‘**value**’ area and select from the list of values that appear.

* + - 1. **Device Family**: Family of the FPGA/CPLD used. In this laboratory we will be using the Spartan3E FPGA’s.
      2. **Device**: The number of the actual device. For this lab you may enter **XC3S250E** (this can be found on the attached prototyping board)
      3. **Package**: The type of package with the number of pins. The Spartan FPGA used in this lab is packaged in CP132 package.
      4. **Speed Grade**: The Speed grade is “-4”.
      5. **Synthesis Tool**: **XST** [VHDL/Verilog]
      6. **Simulator:** The tool used to simulate and verify the functionality of the design. Then click on **NEXT**

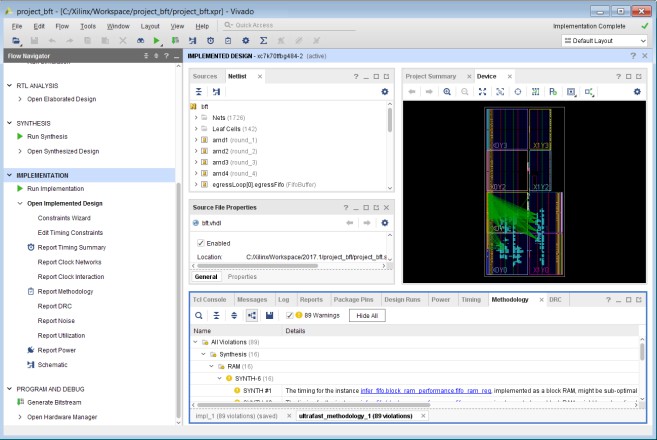
to save the entries.

#### Opening Designs:

Use the Flow Navigator or Flow menu to select the following commands:

* + - 1. Open Elaborated Design
      2. Open Synthesized Design
      3. Open Implemented Design

The Flow > Open Implemented Design command populates the Vivado IDE as shown in below figure.



**Fig 22** : Implemented Design

All project files such as schematics, netlists, Verilog files, VHDL files, etc., will be stored in a subdirectory with the project name.

In order to open an existing project in Xilinx Tools, select **File->Open Project** to show the list of projects on the machine. Choose the project you want and click **OK**.

If creating a new source file, click on the NEW SOURCE.

Creating a Verilog HDL input file for a combinational logic design:

In this lab we will enter a design using a structural or RTL description using the Verilog HDL. You can create a Verilog HDL input file (**.v** file) using the HDL Editor available in the Xilinx Vivado Tools (or any text editor).

In the previous window, click on the NEW SOURCE

(Note: “**Add to project**” option is selected by default. If you do not select it then you will have to add the new source file to the project manually.)

Select **Verilog Module** and in the “File Name:” area, enter the name of the Verilog source file you are going to create. Also make sure that the option **Add to project** is selected so that the source need not be added to the project again. Then click on **Next** to accept the entries.

In the **Port Name** column, enter the names of all input and output pins and specify the **Direction** accordingly. A Vector/Bus can be defined by entering appropriate bit numbers in the **MSB/LSB** columns. Then click on **Next>**to get a window showing all the new source information above window. If any changes are to be made, just click on **<Back** to go back and make changes. If everything is acceptable, click on **Finish > Next > Next > Finish** to continue.

Once you click on **Finish**, the source file will be displayed in the sources window in the **Project Navigator.** If a source has to be removed, just right click on the source file in the **Sources in Project** window in the **Project Navigator** and select **remove** in that. Then select **Project -> Delete Implementation Data** from the Project Navigator menu bar to remove any related files.

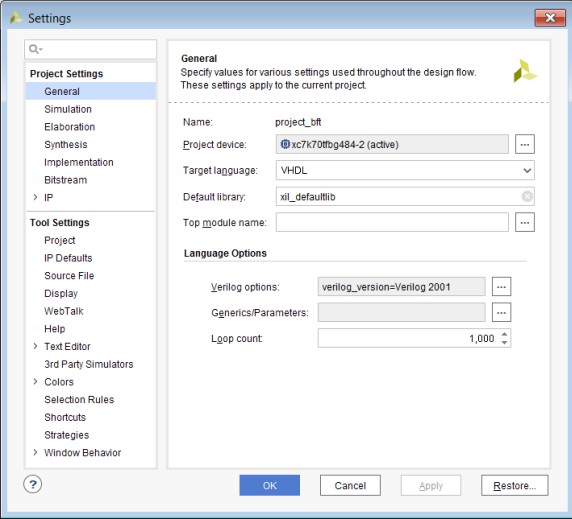
#### Editing the Verilog source file

The source file will now be displayed in the **Project Navigator** window (Figure 8). The source file window can be used as a text editor to make any necessary changes to the source file. All the input/output pins will be displayed. Save your Verilog program periodically by selecting the **File->Save** from the menu. You can also edit Verilog programs in any text editor and add them to the project directory using “Add Copy Source”.

Here in the above window we will write the Verilog programming code for specified design and algorithm in the window.

After writing the programming code we will go for the synthesis report.

#### Configuring Project Settings

You can configure the Project Settings in the Settings dialog box to meet your design needs. These settings include general settings, related to the top module definition and language options, as well as simulation, elaboration, synthesis, implementation, bitstream, and IP settings.

**Fig 23** : Settings Dialog Box—Project Settings General Category To open the Settings dialog box, use any of the following methods:

* In the Flow Navigator Project Manager section, click Settings.
* Select Tools > Settings.
* In the main toolbar, click the Settings toolbar button.
* In the Project Summary, click the Edit link next to Settings.
  + 1. ***Synthesis and Implementation of the Design:***

The design has to be synthesized and implemented before it can be checked for correctness, by running functional simulation or downloaded onto the prototyping board. With the top-level Verilog file opened (can be done by double-clicking that file) in the HDL editor window in the right half of the Project Navigator, and the view of the project being in the **Module view** , the **implement design** option can be seen in the **process view**. **Design entry utilities** and **Generate Programming File** options can also be seen in the process view.

To synthesize the design, double click on the **Synthesize Design** option in the **Processes window**.

To implement the design, double click the **Implement design** option in the **Processes window**. It will go through steps like **Translate, Map and Place & Route**. If any of these steps could not be done or done with errors, it will place a **X** mark in front of that, otherwise a tick mark will be placed after each of them to indicate the successful completion

After synthesis right click on synthesis and click view text report in order to generate the report of our design.

## XILINX VIVADO SIMULATION PROCEDURE:

After completion of synthesis we will go simulation in order to verify the functionality of the implemented design.

Click on **Run Simulation** and set the module that is need to Run

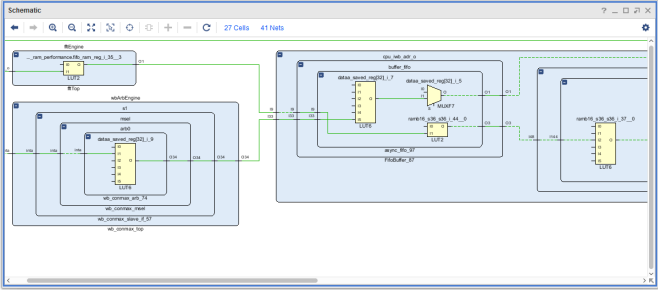
Next **double click on Run Behavioral Simulation** to check the errors. If no errors are found then double click on simulate behavioral model to get the output waveforms.

After clicking on **simulate behavioral model**, the simulation widow will appear pass the input values by making force constant and if it is clock by making force clock. Mention the simulation period and run for certain time and results will appear as shown in following window. Verify the results to the given input values.

#### Using the Schematic Window:

You can generate a Schematic window for any level of the logical or physical hierarchy. You can select a logic element in an open window, such as a primitive or net in the Netlist window, and use the Schematic command in the popup menu to create a Schematic window for the selected object.

An elaborated design always opens with a Schematic window of the top-level of the design, as shown in below figure.



**Fig 24**: Schematic window

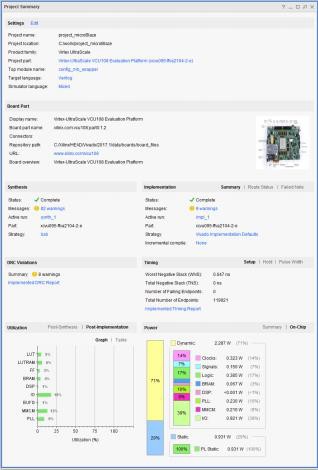
#### Project Summary

The Vivado IDE includes an interactive Project Summary (Figure 3-11) that updates dynamically as design commands are run and the design progresses through the design flow. It provides project and design information, such as the project part, board, and state of synthesis and implementation.

It also provides links to detailed information, such as links to the Messages and Reports windows as well as the Settings dialog box.

As synthesis and implementation complete, DRC violations, timing values, utilization percentages, and power estimates are also populated. To open the Project Summary, do either of the following:

* Select Window > Project Summary.
* Click the Project Summary toolbar button.

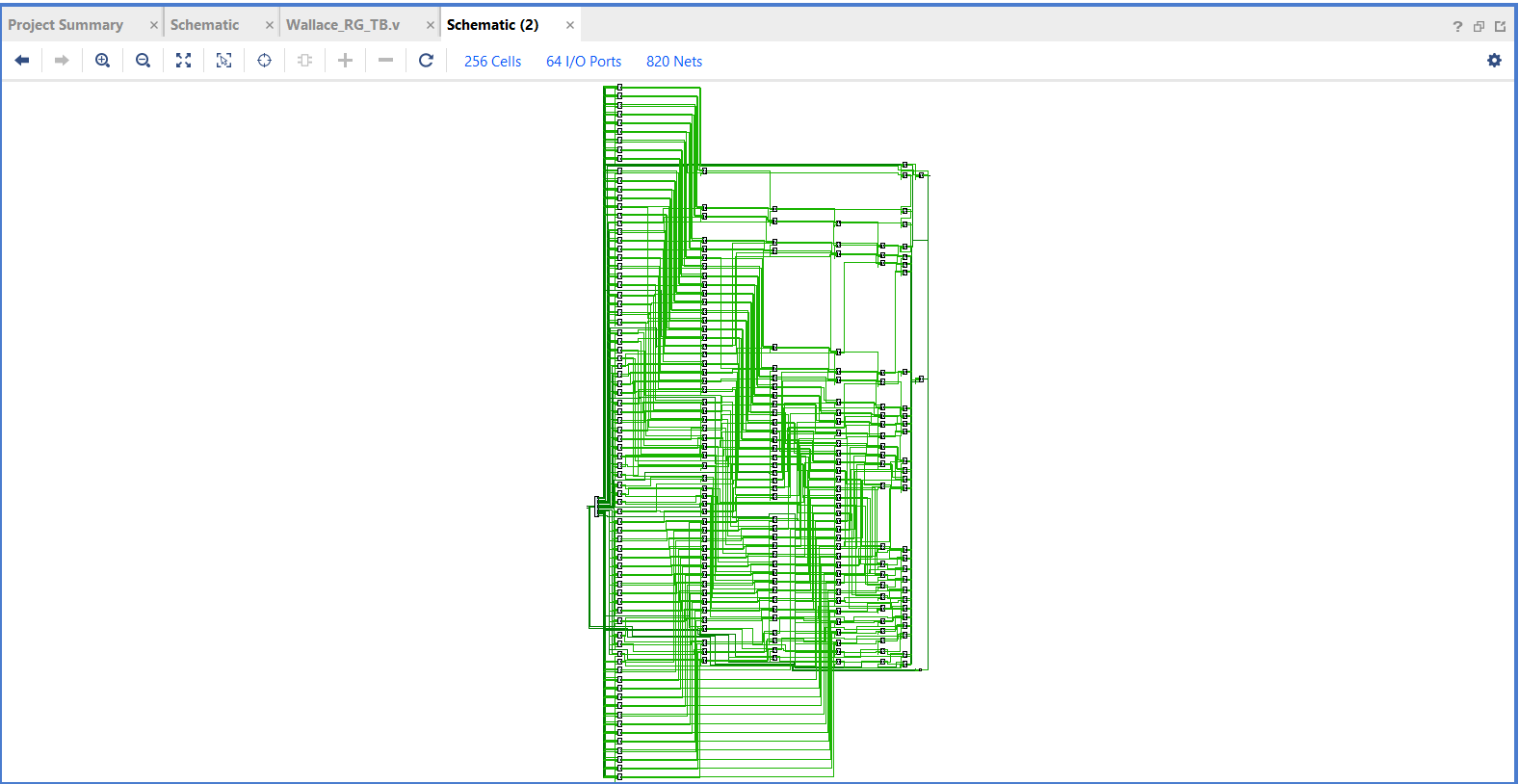


**Fig 25**: Project Summary

**CHAPTER 5**

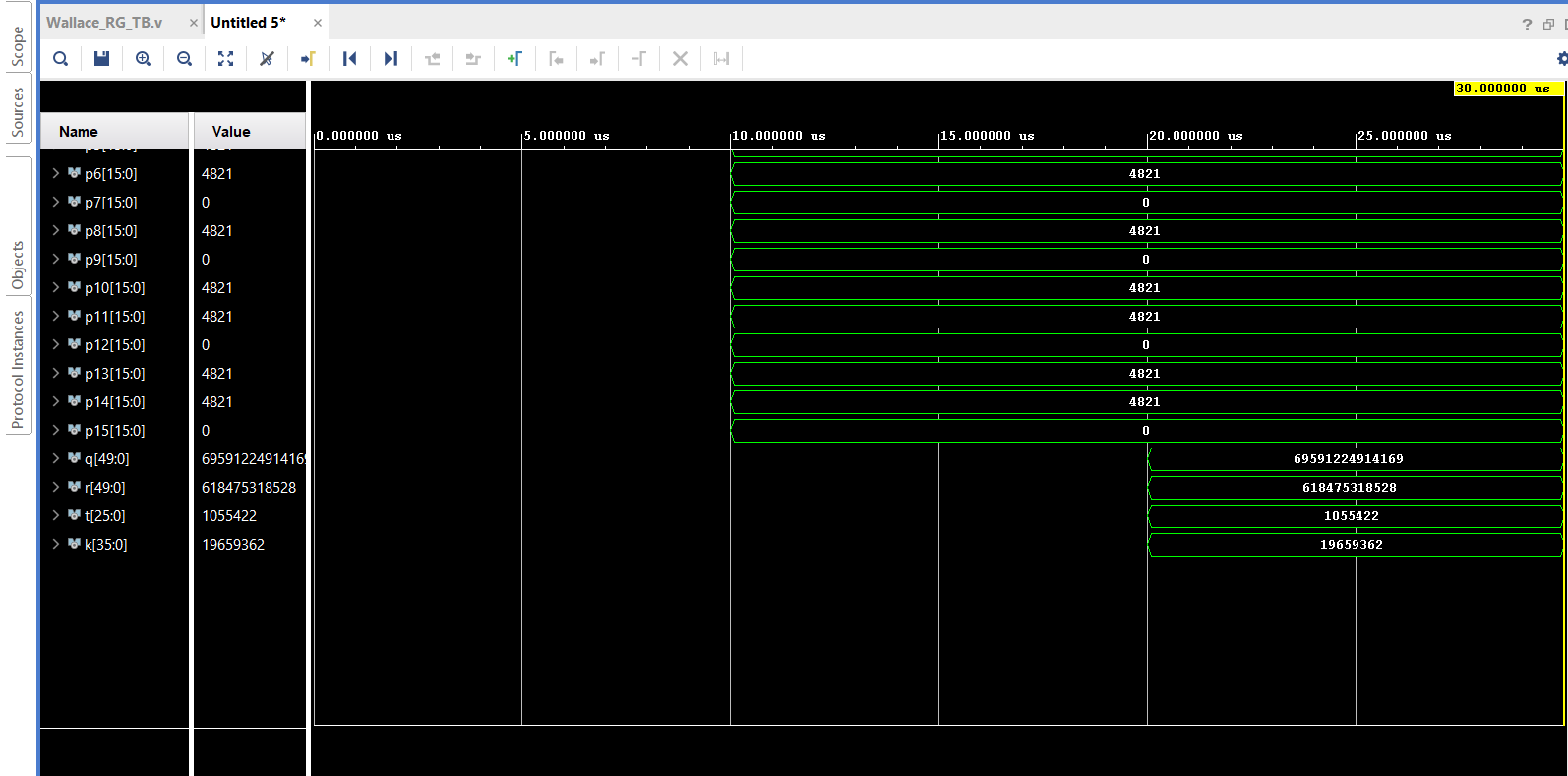
**RESULTS**

### RTL schematic of Wallace Tree Multiplier :



**Fig 26** : RTL Schematic of Wallace tree Multiplier

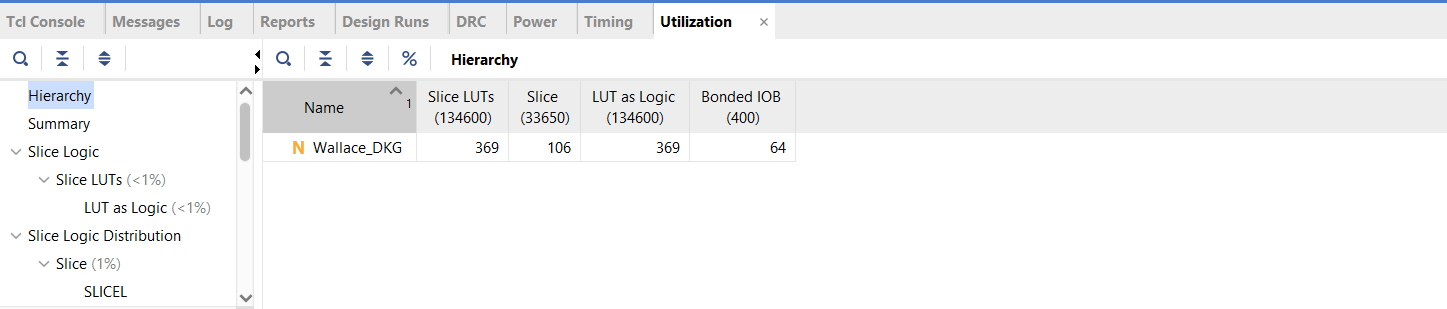
**Simulation of Wallace tree Multiplier :**



**Fig 27**: Simulation Results of Wallace tree

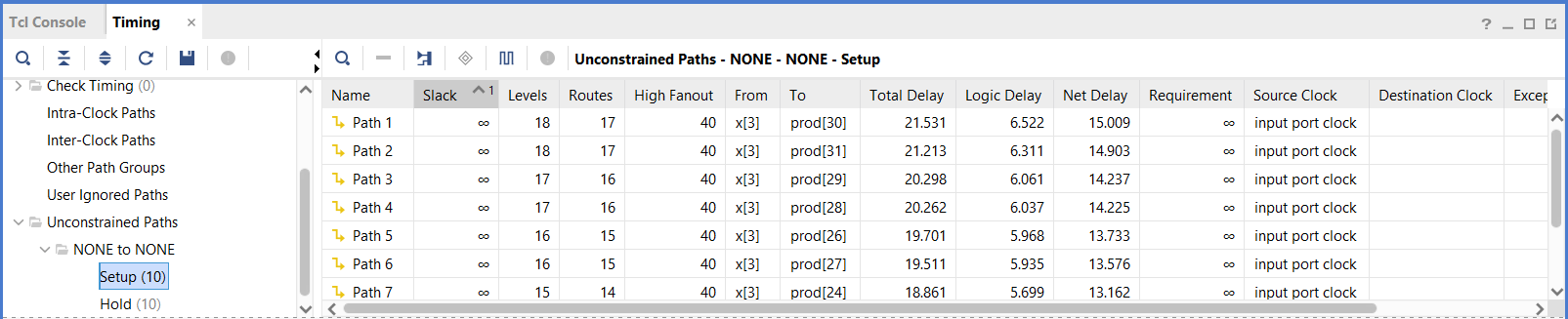
**Interpretation:** Fig 27 shows that simulation of Wallace tree multiplier where the 32-bit plain text and key are given as inputs to encryption block and the 32-bit cipher text is obtained as output. The encrypted output and the 32-bit key is given as inputs to the decryption block and the original text obtained as output.

### Area :



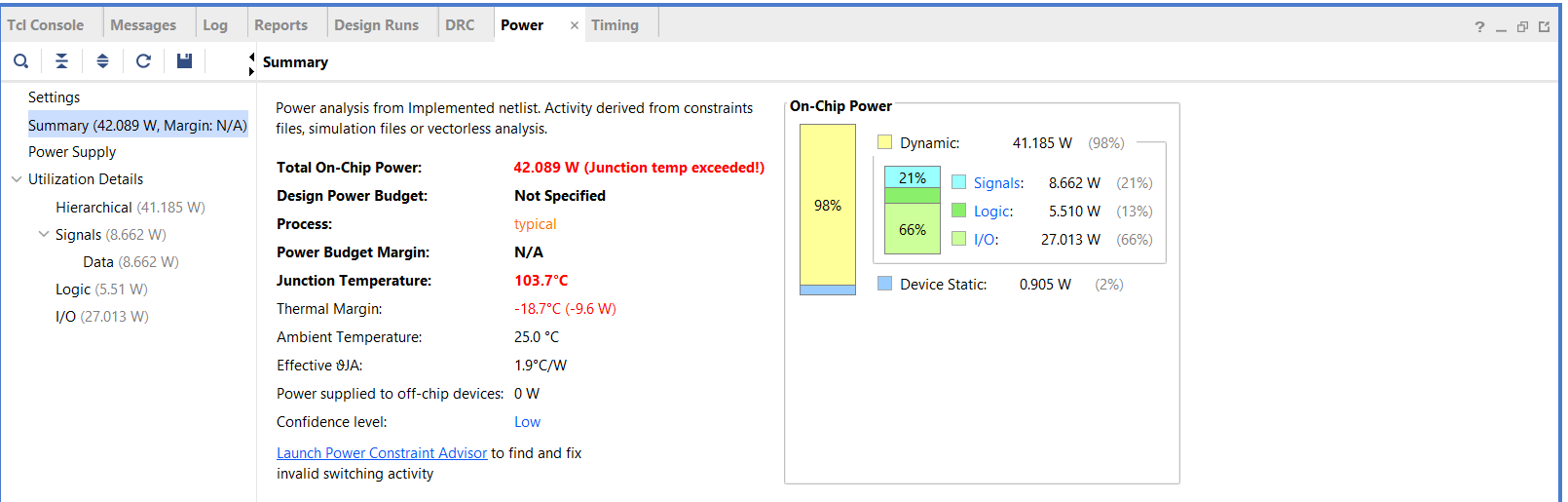
**Fig 28**: Area of Wallace tree Multiplier

**Delay :**



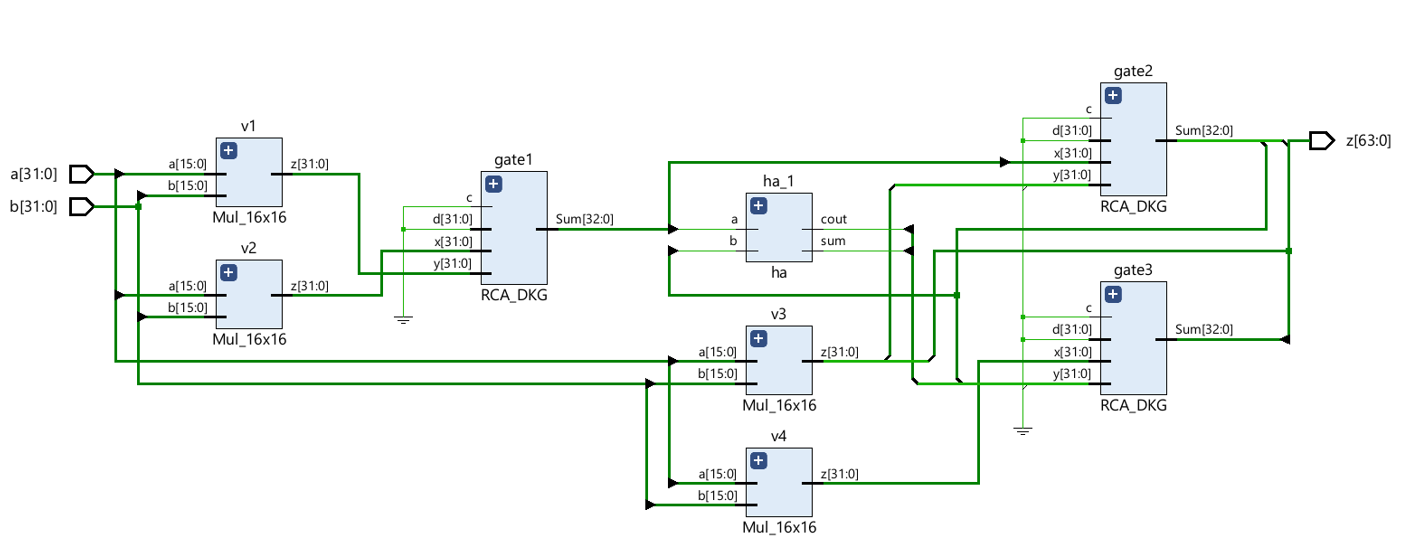
**Fig 29:** Time Delay of Wallace tree Multiplier

**Power :**



**Fig 30:**  Power utilization of Wallace tree Multiplier

### RTL schematic of Vedic Multiplier :

****

**Fig 31** :RTL Schematic of Vedic Multiplier

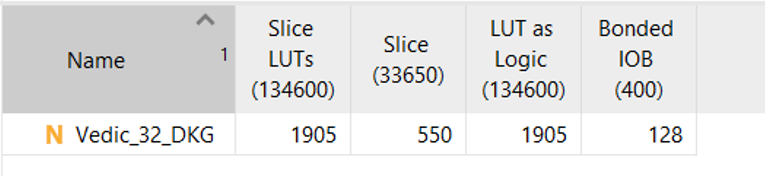
### Simulation results of Vedic Multiplier:

### 

**Fig 32** : Simulation results of Vedic Multiplier

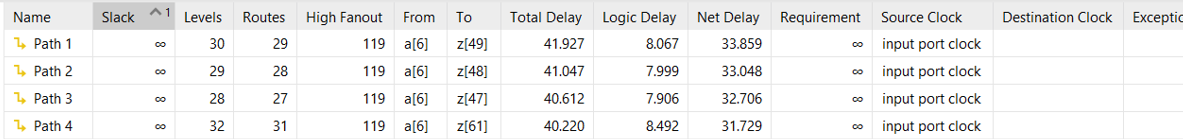
**Interpretation:** Fig 32 shows that simulation results of Vedic multiplier where the 64-bit plain text and key are given as inputs to encryption blocks and the 64-bit cipher text is obtained as output. The encrypted output under the 64-bit key is given as input to decryption block and the original plain texts is obtained as outputs.

### Area:

****

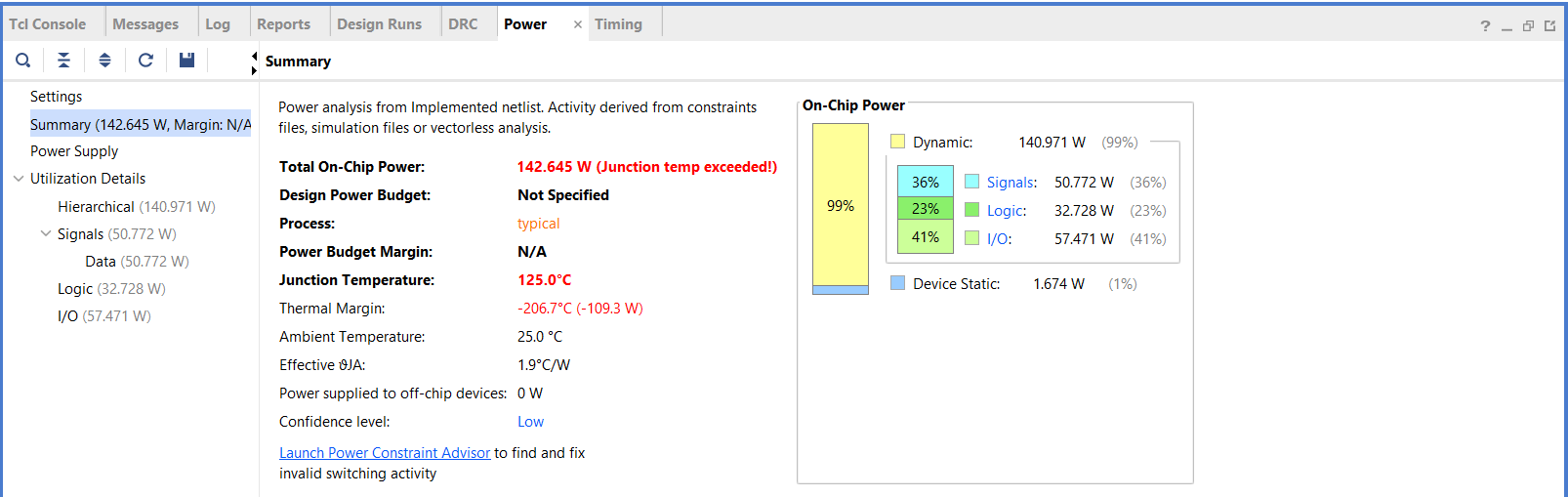
**Fig 33**: Area of Vedic Multiplier

**Delay :**



**Fig 34** : Delay of Vedic Multiplier

**Power :**



**Fig 35 :** Power utilization of Vedic Multiplier

**CHAPTER 6**

**ADVANTAGES & APPLICATIONS**

### Advantages

* Area and Hardware complexity of the design are optimized
* Speed of operation is increased

### Applications

#### Applications:

* Digital Circuits
* Processors
* Signal Processing Applications

**CHAPTER 8**

**CONCLUSION**

The focus of multiplier design has traditionally been delaying optimization, although this design goal has recently been supplemented by power consumption considerations. Our goal has been first to understand how power is dissipated in multipliers, and secondly to devise ways to reduce this power consumption. We have presented an investigation of multiplier power dissipation, along with some techniques which allow reductions in power consumption for this circuit. Given the importance of multipliers, it is likely that further research efforts will be directed at optimizing this block for delay and power efficiency. In this paper, we presented a novel 4x4 bit reversible multiplier circuit using HNG gates and Peres gates. Table 2 demonstrates that the proposed reversible multiplier circuit is better than the existing designs in terms of hardware complexity, number of gates, garbage outputs and constant inputs. Furthermore, the restrictions of reversible circuits were highly avoided. Our proposed reversible multiplier circuit can be applied to the design of complex systems in nanotechnology. All the proposed circuits are technology independent since quantum logic and optical logic implementations are not available.

# REFERENCES

1. C H Bennett, (1998) "Notes on the History of Reversible Computation", IBM Journal of Research and Development, vol. 32, pp. 16-23.
2. R Landauer, (1961) “Irreversibility and Heat Generation in the Computational Process”, IBM Journal of Research and Development, vol. 5, no. 3, pp. 183-191.
3. C H Bennett, (1973) “Logical Reversibility of Computation”, IBM Journal of Research and Development, vol. 17, no. 6, pp. 525-532.
4. T Toffoli, (1980) “Reversible Computing”, Technical Memo MIT/LCS/TM-151, MIT Lab for Computer Science.
5. H Thapliyal and N Ranganathan, (2009) “Design of Efficient Reversible Binary Sub tractors Based on a New Reversible Gate”, IEEE Proceedings of the Computer Society Annual Symposium on VLSI, pp. 229-234.
6. H Thapliyal and N Ranganathan, (2010) “Design of Reversible Latches Optimized for Quantum Cost, Delay and Garbage Outputs”, Proceedings of Twenty Third International Conference on VLSI Design, pp. 235-240.
7. Lihui Ni, Zhijin Guan, and Wenying Zhu, (2010) “A General Method of Constructing the Reversible FullAdder”, Third International Symposium on Intelligent Information Technology and Security Informatics, pp. 109- 113.
8. Irina Hashmi and Hafiz Md. Hasan Babu, (2010) “An Efficient Design of a Reversible Barrel Shifter”, Twenty Third International Conference on VLSI Design, pp. 93-98.
9. Robert Wille, Daniel Grobe , D Michael Miller, and Rolf Drechsler, (2009) “Equivalence Checking of Reversible Circuits”, Thirty Ninth International Symposium on Multiple-Valued Logic, pp. 324-330
10. Noor Muhammed Nayeem, Md. Adnan Hossain, Lafifa Jamal, and Hafiz Md. Hasan Babu, (2009) “Efficient Design of Shift Registers using Reversible Logic”, International Conference on Signal Processing Systems, pp. 474-478.
11. Majid Mohammadi, Mohammad Eshghi, Majid Haghparast and Abbas Bahrololoom, (2008) “Design and Optimization of Reversible BCD Adder/Sub tractor Circuit for Quantum and Nanotechnology Based Systems”, World Applied Sciences Journal, vol. 4, no. 6, pp. 787-792.
12. Majid Mohammadi and Mohammad Eshghi, (2008) “Behavioral description of V and V+ gates to Design Quantum Logic Circuits”, Fifth International MultiConference on Systems, Signals and Devices, pp. 1-6
13. Rekha K James, Shahana T K, K Poulose Jacob, and Sreela Sasi, (2007) “A New Look at Reversible Logic Implementation of Decimal Adder”, The International Symposium on System-On-Chip.
14. Himanshu Thapliyal and A P Vinod, (2006) “Transistor Realization of Reversible TSG Gate and Reversible Adder Architectures”, Proceedings of IEEE Asia Pacific Conference on Circuits and Systems, pp. 418- 421.
15. Himanshu Thapliyal and M B Srinivas, (2006) “Novel Design and Reversible Logic Synthesis of Multiplexer Based Full Adder and Multipliers”, Forty Eight Midwest Symposium on Circuits and Systems, vol. 2, pp. 1593 – 1596.