Design of High-Speed Parallel Multiplier

A Project report submitted in partial fulfillment
of the requirements for the degree of B. Tech in Electrical Engineering

By

Sanmoy Mallick (11701619037)
Saheb Mallick (11701619030)
Sambrita Biswas (11701619008)

Under the supervision of

Mr. Deepam Gangopadhyay
Assistant Professor of the Department of Electrical Engineering

Department of Electrical Engineering
RCC INSTITUTE OF INFORMATION TECHNOLOGY
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA – 700015, WEST BENGAL
Maulana Abul Kalam Azad University of Technology (MAKAUT)
© 2022

Acknowledgement

It is my great fortune that I have got the opportunity to carry out this project work under the supervision of **Mr. Deepam Gangopadhyay**, Assistant Professor in the Department of Electrical Engineering, RCC Institute of Information Technology (RCCIIT), Canal South Road, Beliaghata, Kolkata-700015, affiliated to Maulana Abul Kalam Azad University of Technology (MAKAUT), West Bengal, India. I express my sincere thanks and deepest sense of gratitude to my guide for his constant support, unparalleled guidance, and limitless encouragement.

I wish to convey my gratitude to **Prof. (Dr) Shilpi Bhattacharya**, HOD, Department of Electrical Engineering, RCCIIT, and to the authority of RCCIIT for providing all kinds of infrastructural facilities for the research work.

I would also like to convey my gratitude to all the faculty members and staff of the Department of Electrical Engineering, RCCIIT for their wholehearted cooperation to make this work turn into reality.

_____

Sanmoy Mallick (11701619037)

_____

Saheb Mallick (11701619030)

_____

Sambrita Biswas (11701619008)

Name & Signature of Students

Place: _____

Date: _____

CERTIFICATE

To whom it may concern

This is to certify that the project work entitled **Design of High-Speed Parallel Multiplier** is the bona fide work carried out by **Sanmoy Mallick (11701619037), Saheb Mallick (11701619030), Sambrita Biswas (11701619008)**, students of B.Tech in the Dept. of Electrical Engineering, RCC Institute of Information Technology (RCCIIT), Canal South Road, Beliaghata, Kolkata-700015, affiliated to Maulana Abul Kalam Azad University of Technology (MAKAUT), West Bengal, India, during the academic year 2021-22, in partial fulfillment of the requirements for the degree of Bachelor of Technology in Electrical Engineering and this project has not submitted previously for the award of any other degree, diploma, and fellowship.

_____

Signature of the Guide
Name:
Designation

_____

Signature of the HOD, EE
Name:
Designation

_____

Signature of the External Examiner
Name:
Designation:

Abstract

This project is aimed at designing a high-speed parallel multiplier that can multiply 2 equal-sized binary digit arrays in a more speed-efficient manner. Conventional multipliers work at a much lower speed due to the increased delay of incorporated adders inside them. The main workflow of the multiplier is the same as the conventional method which is adding the multiplicand and multiplier and then right-shifting the entire sum on the basis of LSB bits but the adder that is incorporated in the multiplier is developed in a different way in terms of its speed. The development of the adder in terms of its speed increases the overall speed of the multiplier which we actually aim at in this project.

This project has been planned to be having the most suitable architecture for speed efficient parallel multiplier. XILINX ISE SUIT 14.7 has been used as the programming and developing environment and VHDL is the language used for designing the aimed simulative model. Three aspects structural, behavioral, and data flow have been used while developing the design of the multiplier. At first, the Register Level Transfer schemas have been designed through VHDL coding, and then testbench modules have been created for testing the previously created RTLs. Basic digital gate logic has been incorporated from scratch in some parts of the design, especially in the design of the adder. The entire project has been designed via a hierarchical structure of designing which means the entire schema has been designed from the basic building blocks such as logic gates and other combinational components.

A total of 4 major broad head modules have been first designed to uphold the entire multiplier design. They are the Shift register, Square Root Carry Select Adder, Accumulator, and a Central Controller. After designing these 4 major components from the basic scratch level, they have been interconnected in a proposed circuit layout. This entire connection and synchronization uphold the multiplier which has been later tested for its speed and output. Major analysis has been done by comparing the designed multiplier and previously designed multipliers which incorporate the basic level adders such as RCA, CSLA, CLA, etc. This comparison provides us with useful data which states that the multiplier designed with SQRT CSLA has a much lesser delay time or increased speed than the previously designed multipliers with basic adders. In actuality, we are multiplying two N-bit binary numbers and getting a product of 2N. While doing this execution we are strongly giving stress to the delay reduction of the multiplier which means an increase in speed.

Contents

(1) Introduction

The multiplier is the logical device of great concern in a specific processor that is incorporated into a computer's internal system, in any computing system, the task of the processor is very crucial. The task of multiplication is the most time-consuming task of a processor thus enhancing the performance of the multiplier leads to better performance of the processor, especially in the field of digital signal processing, data processing ASIC, and digital image processing. Many application systems based on DSP require extremely fast processing of a huge amount of digital data. Some specific DSP processes like convolution need fast and effective digital multiplication. The multiplier is also an important element in the microprocessor. The demand for fast processors is increasing for high-speed data processing. Since the multiplier requires the longest delay among the basic operational blocks in the digital system its performance must be optimized effectively in terms of its speed or delay. Many high-performance algorithms and architectures have been proposed to accelerate multiplication. The speed of multiplication can be increased by reducing the number of partial products. Various multiplication algorithms such as Vedic, Booth, Modified Booth, Braun, and Baugh-Wooley have been proposed. Any multiplier can be divided into three stages: The partial products generation stage these are generated by AND operation, the partial products addition stage can be carried by different adders, and the final addition stage, this addition stage is the most crucial step of the entire multiplier. The delay of the adder incorporated decides the speed of the entire multiplier in which it is incorporated, for this reason, the main optimization is to be done in the adder. The adder must be designed in such a way that it always gives out its sum in a shorter delay time or at a much higher speed. The algorithm on which the adder is working is one of the most crucial parts. The speed of the adders will be also compared with each other which in turn compares the entire multiplier's speed.

(2) Building Blocks of the Multiplier

(2.1) 16-Bit Shift Register Design

A shift register is a type of sequential logic circuit that is used for storing and transferring digital data. It consists of a chain of flip-flops or latches connected together, allowing data to be shifted from one stage to the next. The basic operation of a shift register involves shifting the contents of each stage to the next stage in a controlled manner. This shifting can occur either in a serial or parallel fashion, depending on the type of shift register used. In a serial shift register, the data is shifted bit-by-bit through the register in a single file. The data is usually entered or extracted at one end of the register, while the other end serves as an input or output. The most common types of serial shift registers are the Serial-in, Serial-out (SISO) register, Serial-in, Parallel-out (SIPO) register, Parallel-in, Serial-out (PISO) register, and Parallel-in, Parallel-out (PIPO) register. Parallel Shift Register: In a parallel shift register, all the bits are shifted simultaneously. This means that data can be entered or extracted in parallel at multiple stages of the register. The most common types of parallel shift registers are the Parallel-in, Serial-out (PISO) register and Parallel-in, Parallel-out (PIPO) register. Applications of shift registers include data storage, data manipulation, serial-to-parallel or parallel-to-serial conversion, data delay, and synchronization. They are commonly used in various digital systems, such as computers, communication systems, data processing, and control systems. It's worth mentioning that there are different variations and configurations of shift registers, such as the universal shift register, bidirectional shift register, ring counter, and Johnson counter. Each configuration has its own specific characteristics and applications. Overall, shift registers are essential building blocks in digital electronics and play a crucial role in various applications involving data manipulation and transfer. A design of a 16-bit Shift Register has been made and implemented. D Flip Flops serves as the basic scratch element of the register. A D flip-flop, also known as a Data flip-flop or Delay flip-flop, is a type of sequential logic circuit. It stores and transfers data based on a clock signal. The D flip-flop has a single input called the "D" input and two outputs: the "Q" output and the complemented output, "$\overline{Q}$". Data Input (D): The D input represents the data to be stored in the flip-flop. It can be either a logic high (1) or a logic low (0). Clock Input (CLK): The CLK input is the clock signal that controls the operation of the flip-flop. The flip-flop captures the data input (D) and stores it when a positive or negative transition of the clock signal occurs, depending on the specific type of flip-flop. Output (Q): The Q output represents the stored data. When the clock signal transitions occur, the flip-flop updates the Q output to the value of the D input. The flip flops have been connected in Parallel in Parallel out

(PIPO) mode. A total set of 16 flip flops have been used to store a 16-bit multiplicand. Each separate clock and reset pin of those flip flops have been short-circuited and connected to the external Reset and Clock pin of the shift register.

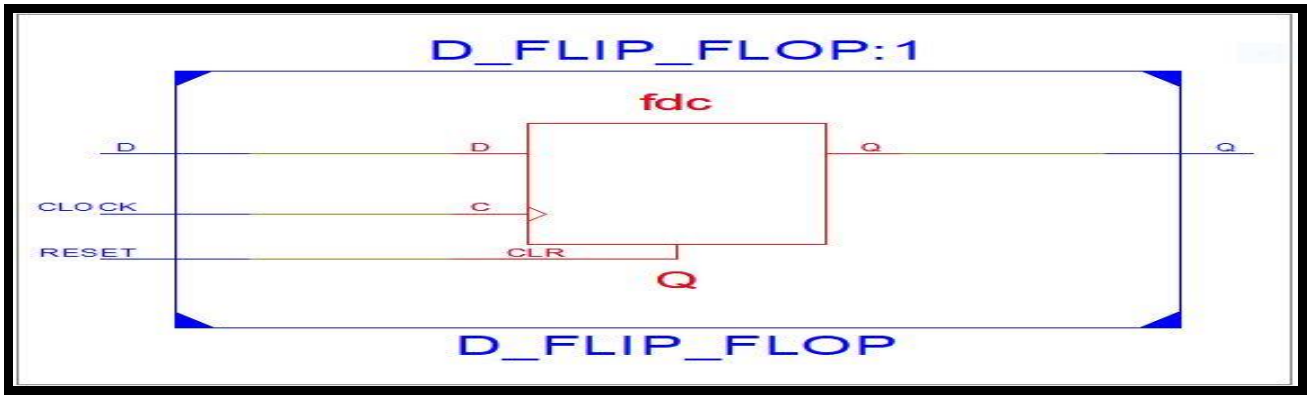

Fig 1- D-Flip-Flop RTL Schema

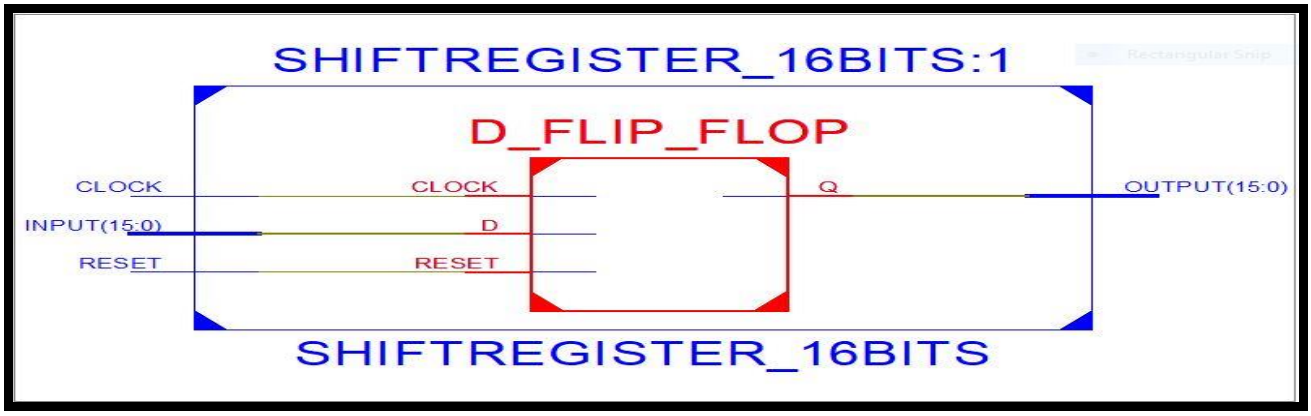

Fig 2 – 16 Bit Shift Register RTL Schema

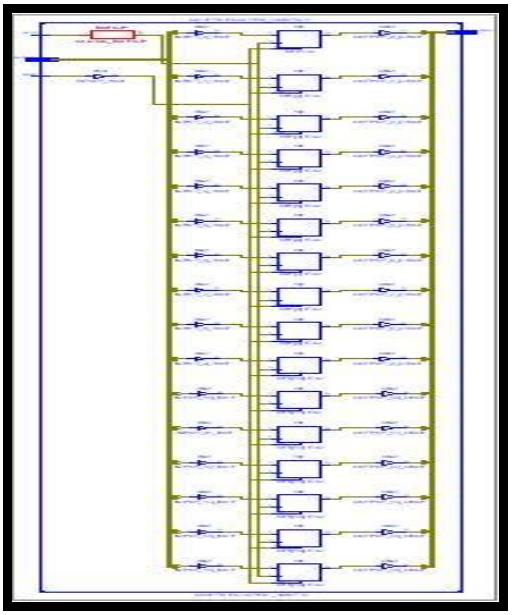

Fig 3 - 16 Bit Shift Register Technology Schema

(2.2) 32-Bit Right Shifting Accumulator Design

In digital circuits, an accumulator is a register or storage component that is used to accumulate or sum up a series of values. It is commonly used in arithmetic and counting operations. The primary purpose of an accumulator is to store and update the cumulative result of a sequence of data. It typically consists of a register that holds the current accumulated value and a set of inputs for receiving new data to be added to the accumulator. Accumulators are often used in applications such as, Arithmetic Operations: Accumulators are frequently employed in arithmetic circuits, such as adders or multipliers, to accumulate partial results during the computation of larger operations. For example, in a multiplication operation, the accumulator can hold the intermediate results as the multiplication progresses. Counters: In digital counters, an accumulator can be used to store and increment a count value. Each time a count pulse is received, the accumulator value is incremented, allowing the counter to keep track of the number of events or cycles that have occurred. Signal Processing: Accumulators are utilized in various signal processing applications, such as digital filters or Fast Fourier Transform (FFT) algorithms. They help in accumulating and processing the sampled input data to generate desired output signals. Data Compression: In some compression algorithms, an accumulator is used to accumulate data and generate compressed representations. For instance, in delta encoding, the accumulator stores the previously encoded value, and the difference between the new data and the accumulator is encoded and transmitted. Accumulators can be implemented using various digital logic elements, such as registers, adders, and multiplexers. The specific design and configuration depend on the application requirements and the desired functionality. Overall, accumulators are vital components in digital circuits for accumulating, processing, and storing data during various operations, making them an essential part of many digital systems. The accumulator is a right-shifting storage register that acts like a temporary memory element. After the completion of a single clock cycle, the result stored in the accumulator is shifted to its right by 1 bit. The multiplier input is stored in the LSB part. The accumulator works on the initialization of three commands that are ADD COMMAND, LOAD COMMAND, and SHIFT COMMAND. These command pins are active and high in nature. The primary function of this accumulator is to accumulate or compound up a series of values. It receives input data and compounds it to the current accumulated value. The adder simulates the accumulator. The adder sums up and sends in the values to the accumulator. Gradual right shifting takes place in the accumulator. The extreme LSB bit of the accumulator decides whether the shifting will occur or not. These bits again come depending on the adder which sends the sum of the multiplier and multiplicand.

If the start signal is 0 the state is IDLE and the output is stopped, and vice versa. When start = 1 output is load _cmd =1. when Load _cmd =1 the state assigned is TEST. During the TEST state, the controller checks the input of LSB, if LSB = 0 then the test assigned in SHIFT and output shift command is 1. If shift_ cmd =1 then one single clock cycle is counted (count =count+1) If LSB = 1 then the state assigned is ADD and output add_cmd = 1, then the state assigned is a shift and one clock cycle is completed. If 16 clock cycles are counted by the controller after repeating the above states then the state assigned is IDLE and the STOP command is generated (STOP=1).

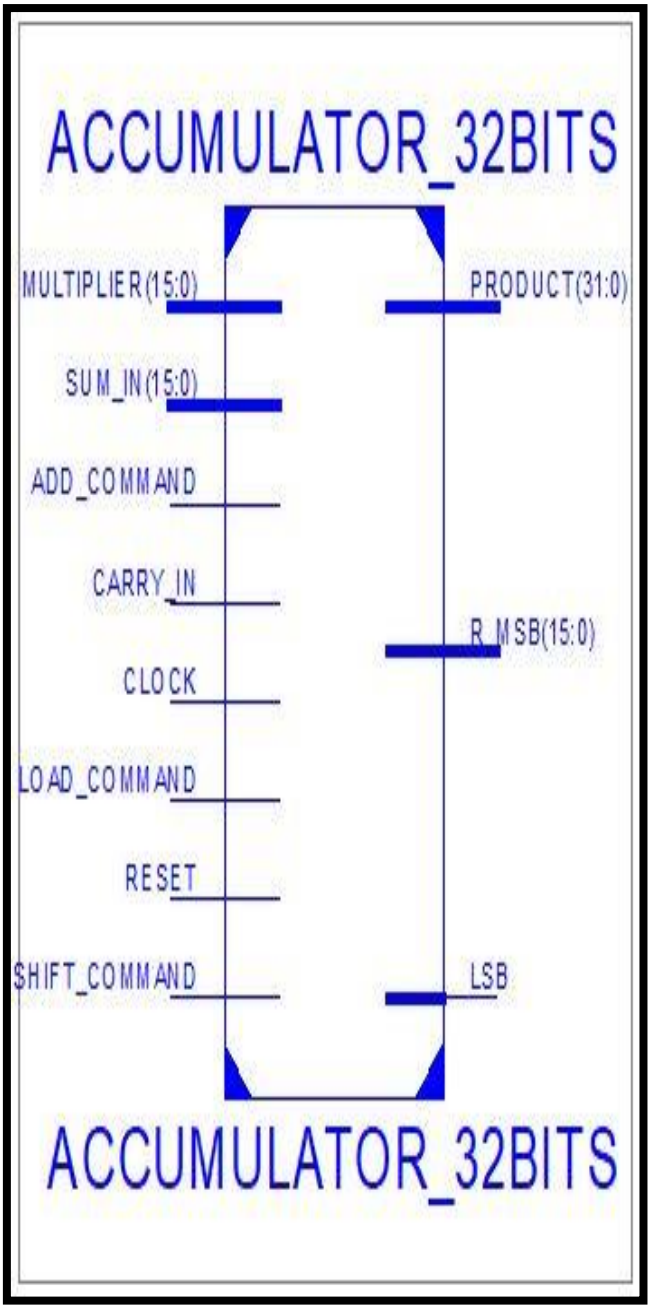

Fig 4 – 32 Bit Accumulator RTL Schema

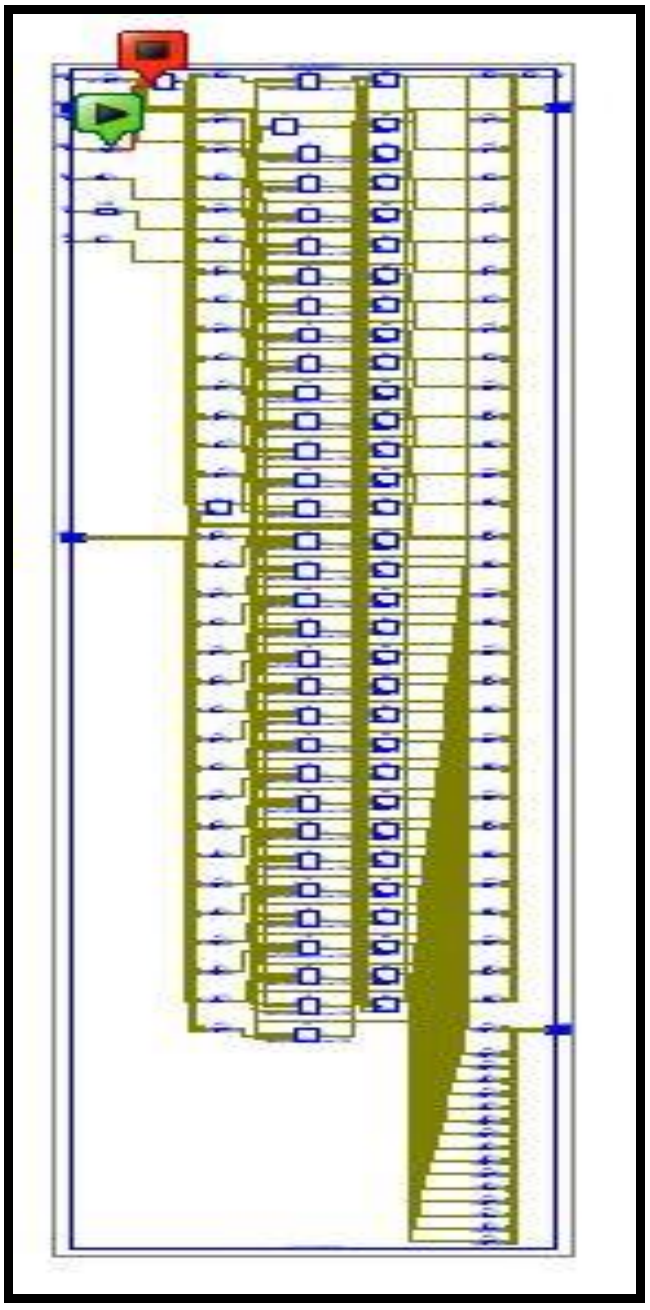

Fig 5 – 32 Bit Accumulator Technology Schema

(2.3) 16-Bit Square Root Carry Select Adder Design (SQRT CSLA)

The Square Root Carry Select Adder (SQRT-CSLA) is a type of adder that is specifically designed to perform square root computations. It is an extension of the Carry Select Adder (CSLA), which is a fast parallel adder used for additional operations. The SQRT-CSLA is capable of performing square root operations by utilizing multiple parallel carriers select adders and additional logic. It takes advantage of the fact that the square root operation involves computing the sum of multiple square terms. By breaking down the square root computation into smaller segments and performing them in parallel, the SQRT-CSLA achieves faster execution compared to traditional sequential square root algorithms. The SQRT-CSLA operates in several stages: Partitioning: The input operands are partitioned into smaller segments, typically by splitting them into groups of bits. Each group represents a separate segment of the square root computation. Square Computation: In each segment, the square of the input values is computed. This can be done using conventional multiplication techniques or specialized square calculation circuits. Carry Select Adder: Multiple carry select adders are used to perform additional operations on the squared segments. Each carries a select adder that handles a specific range of bits within the squared segments. Selection: The output of each carry select adder is combined using selection logic to determine the final result. The selection logic chooses the appropriate result from each segment based on the intermediate carry bits generated during the addition. The advantage of the SQRT-CSLA is that it enables parallel computation of square roots, reducing the overall computation time. However, it requires additional hardware resources and complex selection logic compared to regular CSLA or square root algorithms. The square root carry select adder is constructed by equalizing the delay through two carry chains and the block multiplexer signal from the previous stage. It is also called a non-linear carry select adder. This is an extension of linear CSLA which improves the delay time greatly. By using SQRT CSLA, the delay can be verified, as the time waiting for the carry bit is used to calculate an extra input bit in each stage. This modified design will reduce delay as compared with regular SQRT CSLA. Based on this modification 16 Bit SQRT CSLA architecture and simulation are developed and incorporated into the proposed SQRT CSLA-based multiplier. This adder works on the principle of adding the inputs and selecting the carry propagation on the basis of 2X1 MUX selection. Full adders and mux are the basic scratch-level building block of this added. A full adder is a combinational logic circuit that performs the addition of three input bits: two binary digits (A and B) and a carry-in (Cin) bit. It produces a sum (S) output and a carry-out (Cout) output. The full adder takes into account both the current inputs and the carry generated from a previous stage. 2 Half adder port mapping forms a Full adder

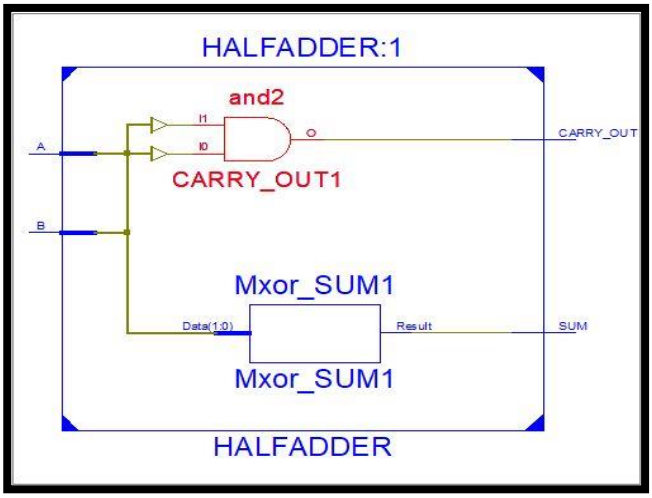

Fig-6 Half Adder RTL Schema

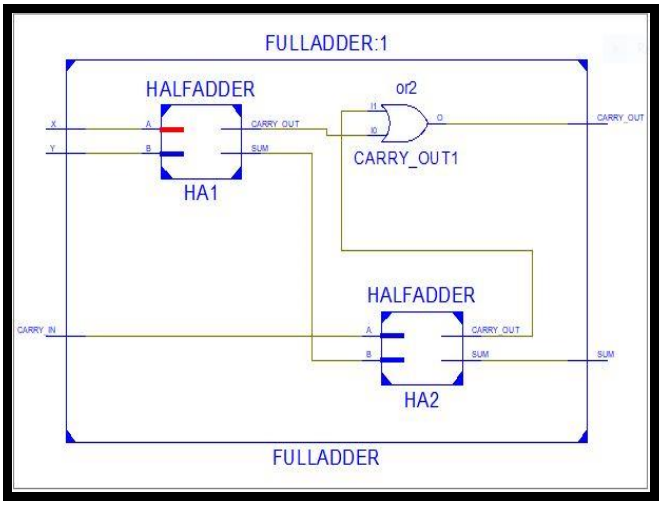

Fig-7 Full Adder RTL Schema

Full adders have been interconnected to form 2,3,4,5 bits ripple carry adders.

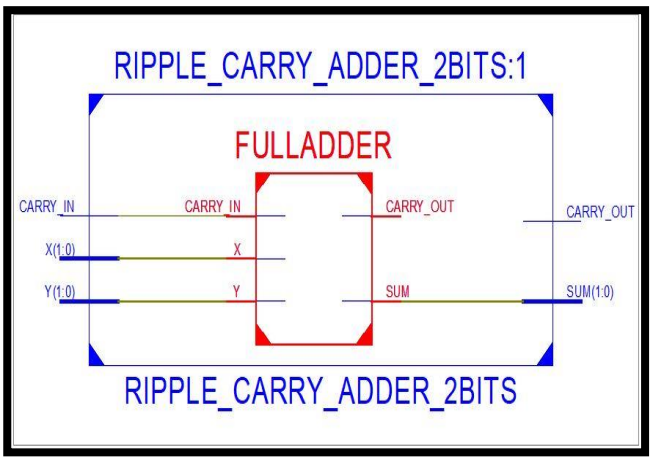

Fig-8 2 Bit RCA RTL Schema

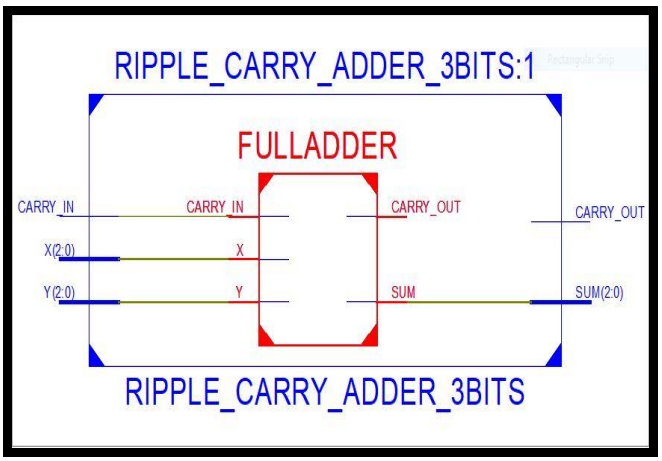

Fig-9 3 Bit RCA RTL Schema

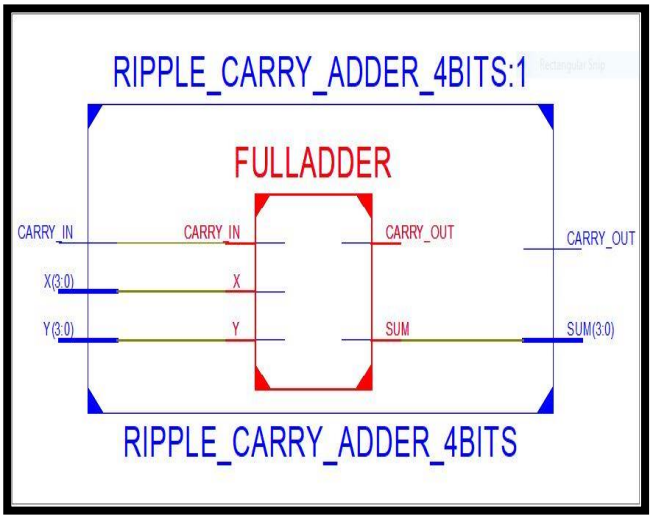

Fig-10 4 Bit RCA RTL Schema

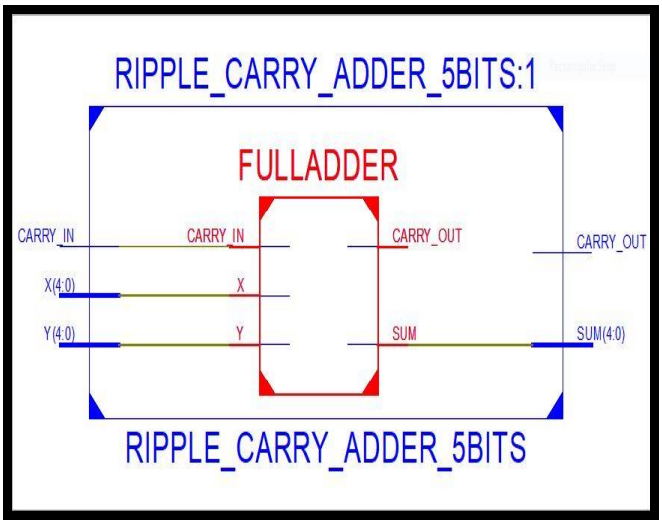

Fig-11 5 Bit RCA RTL Schema

2X1 MUX are designed for multiplexing these RCAs. A 2 x 1 multiplexer (MUX) is a digital logic circuit that has two data inputs, one select, input, and one output. The select input determines which data input is passed through to the output. The term "2 x 1" refers to the number of data inputs and the number of output lines. Here MUX has been designed by applying simple if else logic in VHDL Codes.

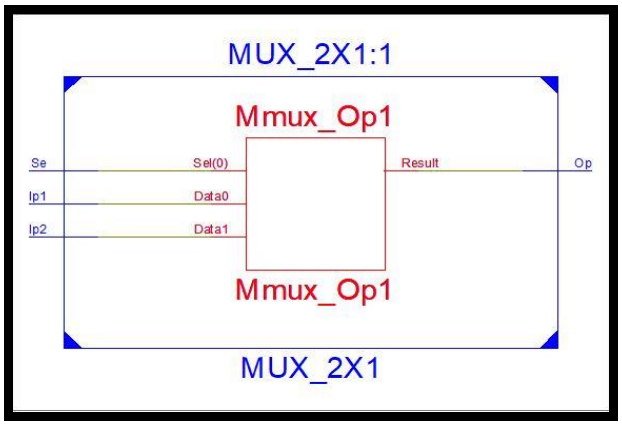

Fig-12 2X1 MUX RTL Schema

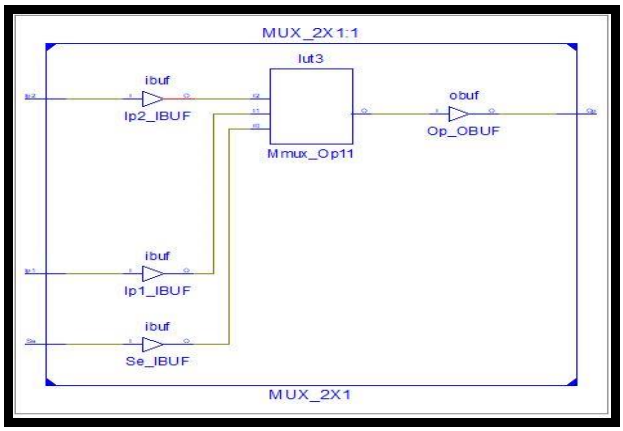

Fig-13 2X1 MUX Technology Schema

Ripple carries adder has been multiplexed with each other to form 2,3,4,5 bits carry select adder (CSLA).

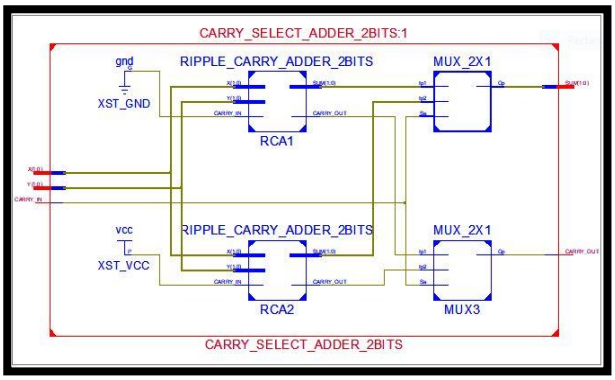

Fig-14 2 Bit CSLA RTL Schema

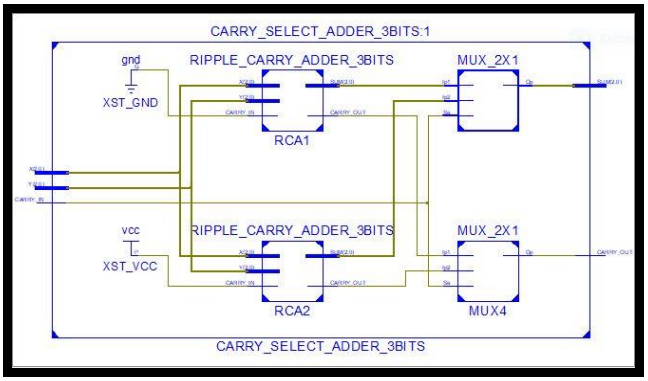

Fig-15 3 Bit CSLA RTL Schema

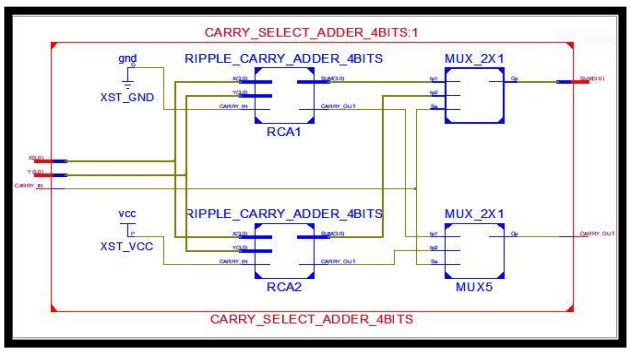

Fig-16 4 Bit CSLA RTL Schema

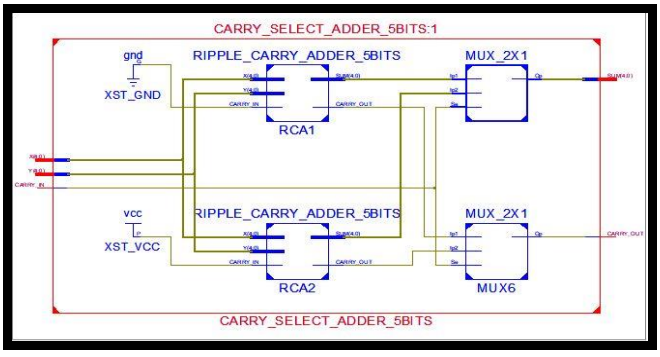

Fig-17 5 Bit CSLA RTL Schema

This carry select adder has been interconnected bit-wise via carry propagations. The bitwise inputs have been given separately to the carry select adder and the carry propagation is fed from the previous stage to the next stage CSLA. The adder's speed completely depends on the carrier selected by the multiplexer. The multiplexer increases the speed of the adder by selecting required carry bits and discarding the unwanted ones. The size of the adder is 16 bits and it is the main component on which the speed of the entire multiplier depends. The delay time taken by the SQRT CSLA is equal to the delay that is taken by the 2x1 MUX each MUX takes A delay of 1.044 nano Sec so 4 mux will take 4*1.044 nano Sec which is equal to 4.176 nano Sec. This delay decides the entire multipliers' speed or delay time. The speed of the multiplier is inversely proportional to the delay of the adder.

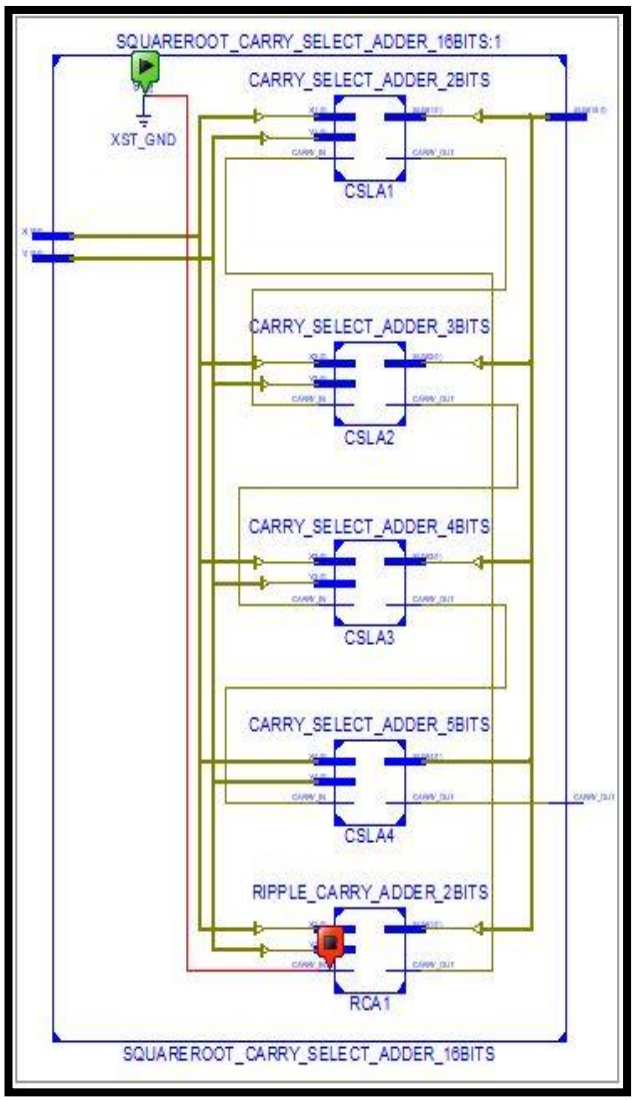

Fig-18 16 Bit SQRT CSLA RTL Schema

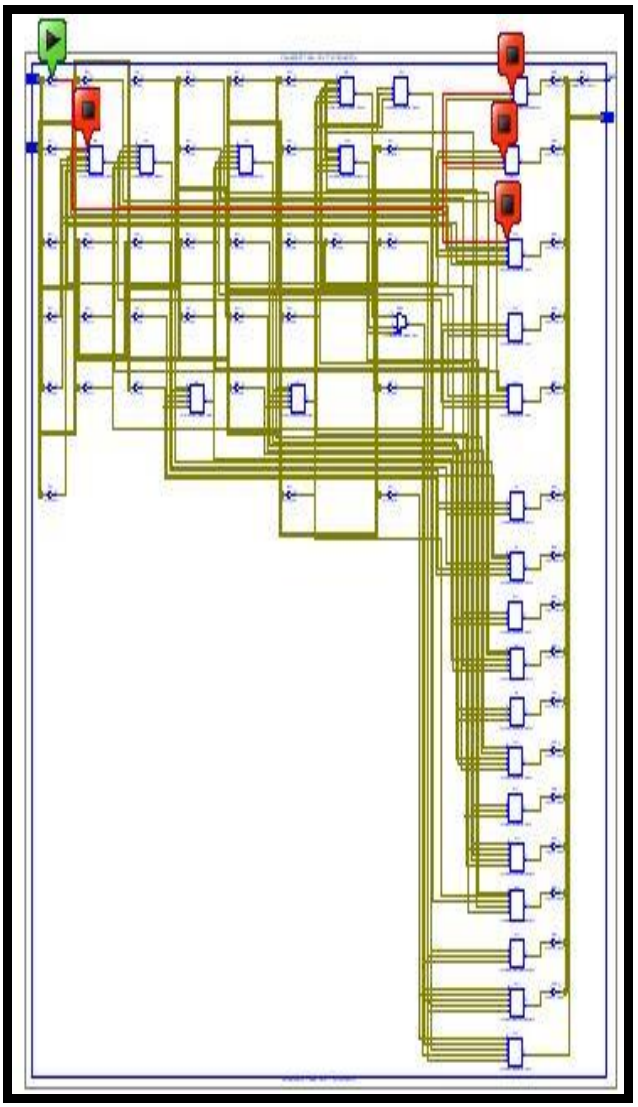

Fig-19 16 Bit SQRT CSLA Technology Schema

(2.4) Control Unit Design

The control unit is a FSM-based model FSM stands for Finite State Machine. It is a mathematical model used to represent and analyze systems with discrete states and transitions between those states. FSMs are widely used in computer science, electrical engineering, and other fields for modeling and designing systems with a finite number of states. In an FSM, the system's behavior is divided into a set of states, and transitions between states are triggered by certain events or conditions. Each state represents a specific configuration or condition of the system, and the transitions define how the system moves from one state to another in response to inputs or events. The basic components of an FSM are as follows: States: The distinct configurations or conditions that a system can be in. Each state represents a specific mode or behavior of the system. Transitions: The rules or conditions that determine how the system moves from one state to another. Transitions are triggered by events or inputs and specify the next state to transition to. Events: The inputs, signals, or events that can trigger a transition between states. These events can be internal (generated within the system) or external (coming from the environment). Actions: The actions or operations associated with a state or a transition. Actions can be performed when entering or exiting a state or during a transition. FSMs are particularly useful for modeling systems that have a finite number of well-defined states and where the system's behavior can be represented as a sequence of transitions between those states. They are widely used in various applications, including software development, protocol design, digital circuit design, and control systems. There are different types of FSMs, such as Mealy machines and Moore machines, which differ in terms of how they associate outputs with states and transitions. Overall, FSMs provide a clear and structured way to represent and analyze the behavior of systems with discrete states, making them an important tool in various areas of computer science and engineering.

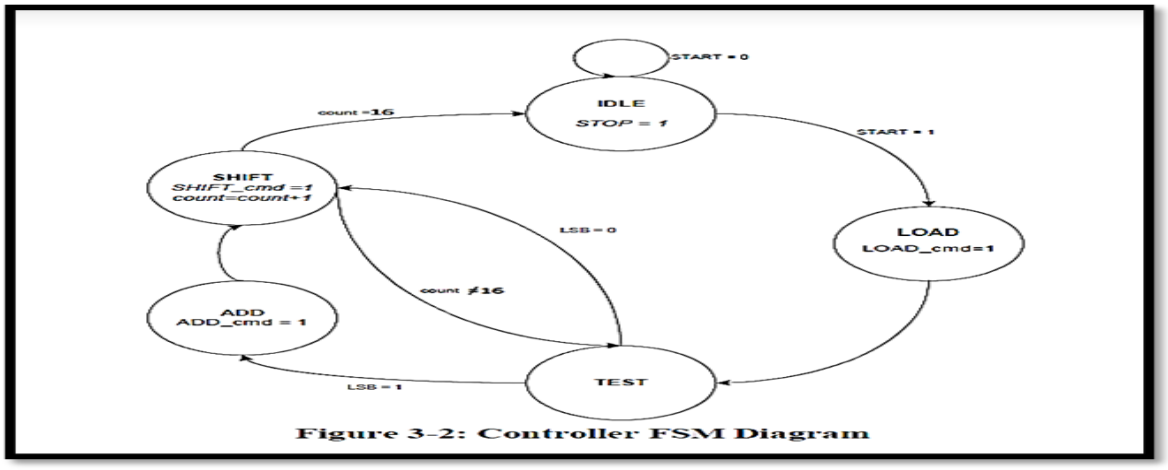

Fig-20 FSM Model Used In Designing The Control Unit

If the start signal is 0 the state is IDLE and the output is stopped, and vice versa. When start = 1 output is load_cmd =1. when Load _cmd =1 the state assigned is TEST. During the TEST state, the controller checks the input of LSB, if LSB = 0 then the test is assigned in SHIFT and the output shift command is 1. IF shift_cmd =1 then one single clock cycle is counted (count =count+1). IF LSB = 1 then the state assigned is ADD and output add_cmd = 1, then the state assigned is a shift and one clock cycle is completed. If 16 clock cycles are counted by the controller after repeating the above states then the state assigned is IDLE and the STOP command is generated (STOP=1).

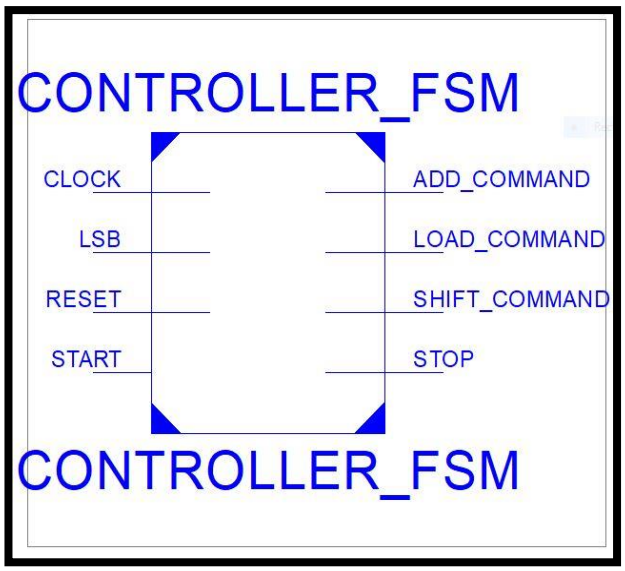

Fig-21 FSM Based Controller RTL Schema

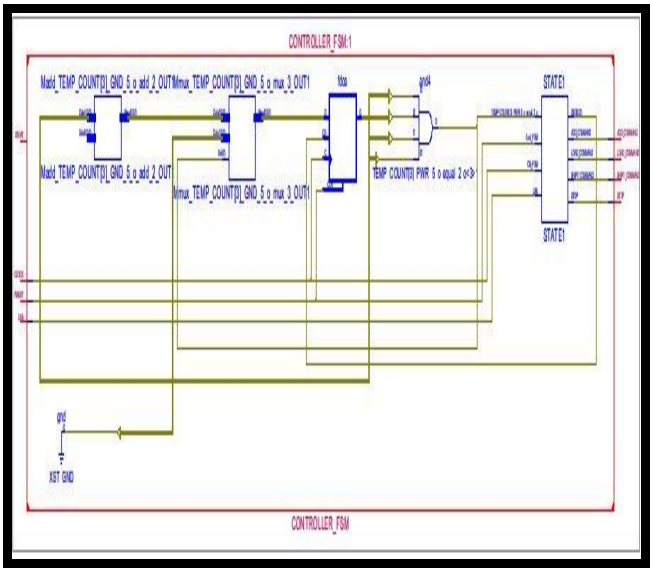

Fig-22 FSM Based Controller RTL Sub Block

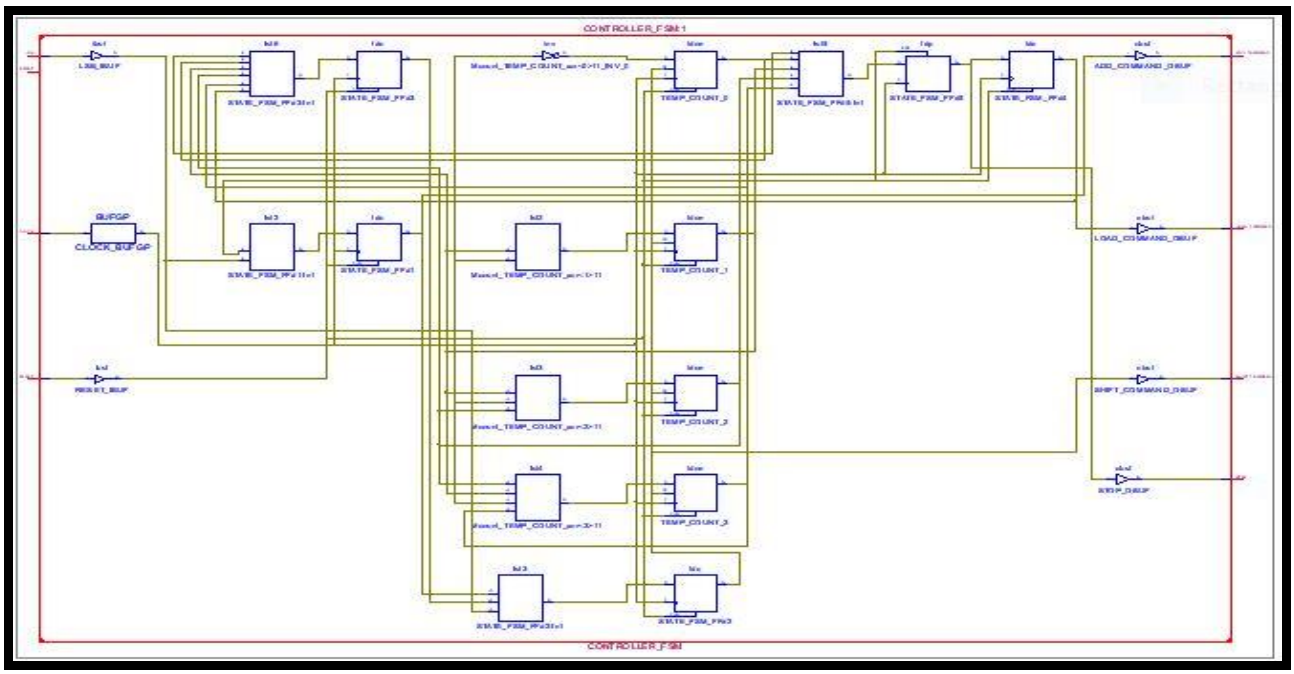

Fig-23 FSM Based Controller Technology Schema

(3) Interconnection of the Building Blocks to Form the Single Multiplier Entity

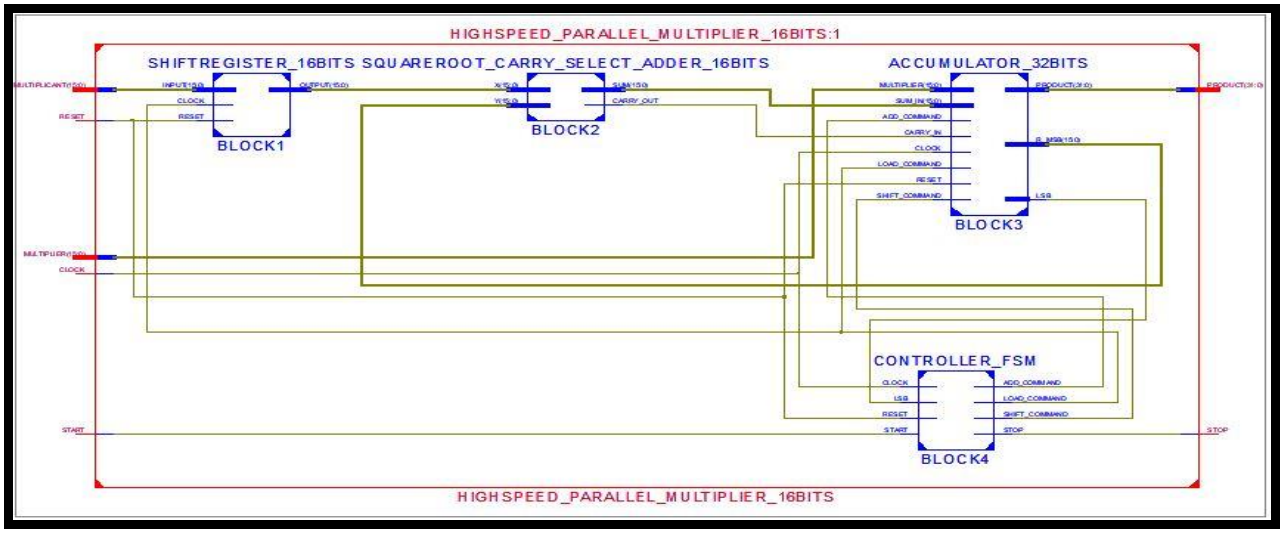

Fig- 24 Interconnection Between the Building Blocks Forming the Multiplier Entity

The 4 major building blocks that are formed previously are interconnected with each other according to the above-given circuit layout. All the blocks work together in sync which is made possible by introducing a single clock input of the entire entity which is shorted to the clock inputs of other components. A Reset pin is also introduced to rearrange the entire cyclic process of the multiplier and is similarly shorted with the reset pins of the other subblocks. 16 Bit each of multiplier and multiplicand is given as input with separate Start, Clock, and Reset inputs. The multiplier gives output in the form of 32 bits product and a Stop signal.

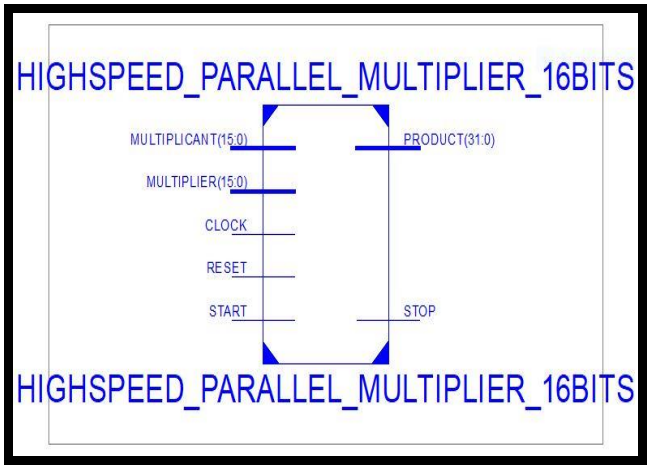

Fig-25 Multiplier's Top Level RTL Schema

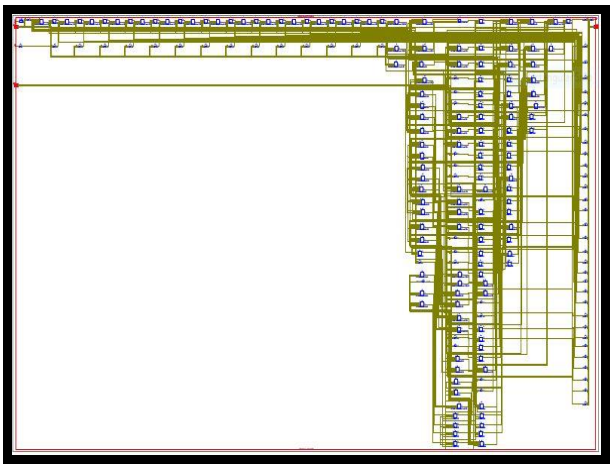

Fig-26 Multiplier's Technology Schema

(4) Working Algorithm of the Multiplier

The right shifting algorithm is commonly used in parallel multipliers to efficiently perform multiplication. Here is a general outline of the right shifting algorithm for parallel multipliers: Partition the multiplier and multiplicand: Divide the multiplier and multiplicand into smaller partitions or slices. Each partition contains a subset of the bits from the original numbers. Initialize the partial products: For each partition of the multiplier, generate partial products by multiplying it with the entire multiplicand. These partial products represent the intermediate results of the multiplication. Right-shift and accumulate: Starting from the least significant partition, right-shift each partial product by a specific number of positions corresponding to the partition's position. Accumulate the right-shifted partial products, summing them together to get the final product. Adjust the accumulated partial products: Since right-shifting can result in lost bits, adjust the accumulated partial products to account for any lost bits. This adjustment is typically done by adding additional terms or carry bits. Combine the adjusted partial products: Finally, combine the adjusted partial products to obtain the final product of the multiplication. It's important to note that the exact details and implementation of the right shifting algorithm can vary depending on the specific design and requirements of the parallel multiplier. Additionally, there are different variations and optimizations that can be applied to improve performance, such as using carry-save adders or parallel prefix adders for accumulation. The given figure shows the algorithm that is used by the multiplier to compute its product. It defines the right-shifting algorithm. It shows how a 4-bit right-shifting multiplication takes place. In this project, we propose a 16-bit multiplication so it will follow the same algorithm but will take 16 complete cycles instead of 4 cycles. The given number of bits decides the number of clock cycles the entire system will go through while computing its product. In brief, we can say that "Number of bits = Number of clock cycles needed". This algorithm controls the entire system of the multiplier.

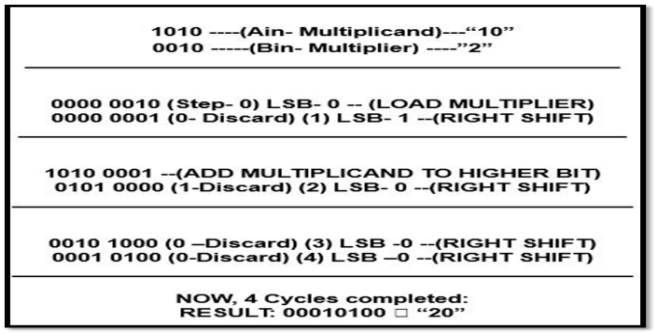

Fig-27 Right Shifting Algorithm

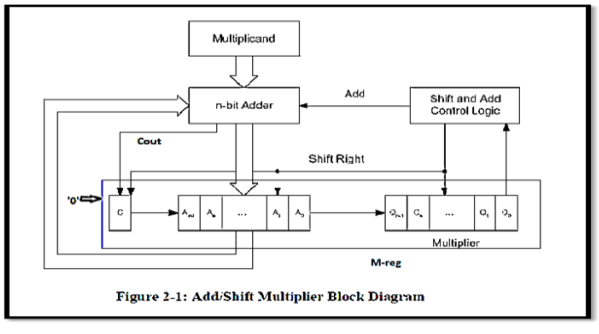

Fig-28 Algorithm Wise Working of The Multiplier

(5) Output Analysis of the Multiplier

The below-given figure is the test bench simulation outcome of the entire multiplier entity. The test bench simulations were done and the actual delay time of the designed multiplier was obtained. At 475.000 nano Sec instance the output is gained, after completing 16 clock cycles, The delay time taken is 345.000 nano Sec. The stop signal is set to '1', and the start signal is set low '0'.

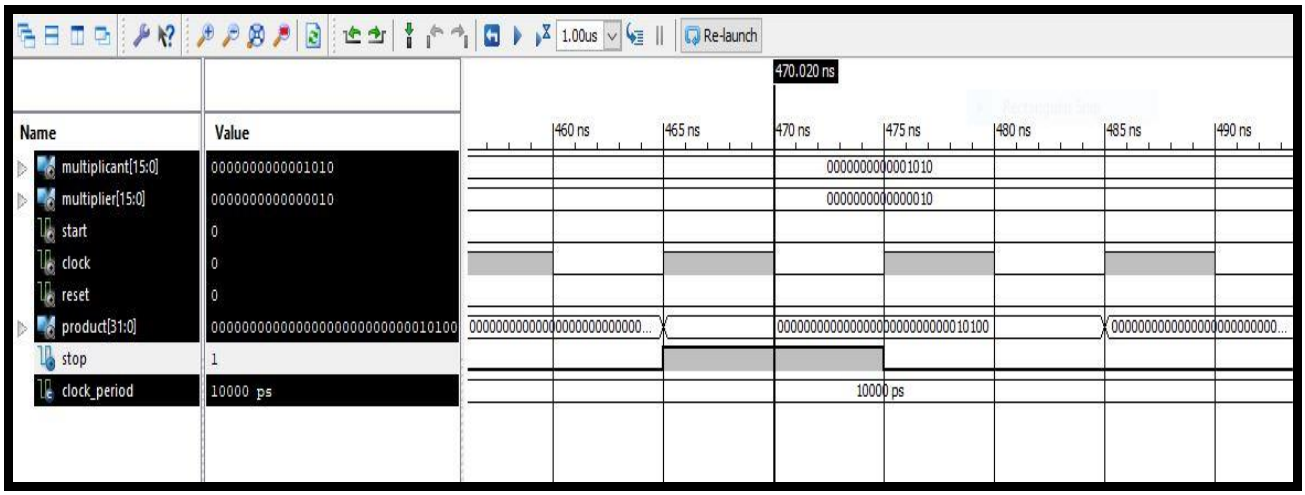

Fig-29 Test Bench Output Simulation Analysis of The Multiplier Entity

The speed of the entire multiplier solely depends on the delay time taken by the adder incorporated inside it. In this case, we have incorporated a Square Root Carry Select Adder, so the speed of this adder decides the speed of the multiplier entity. Or we can also say that delay of the adder decides the delay time taken or speed of the multiplier entity. The delay of the adder is decided by the mux delays incorporated inside it.

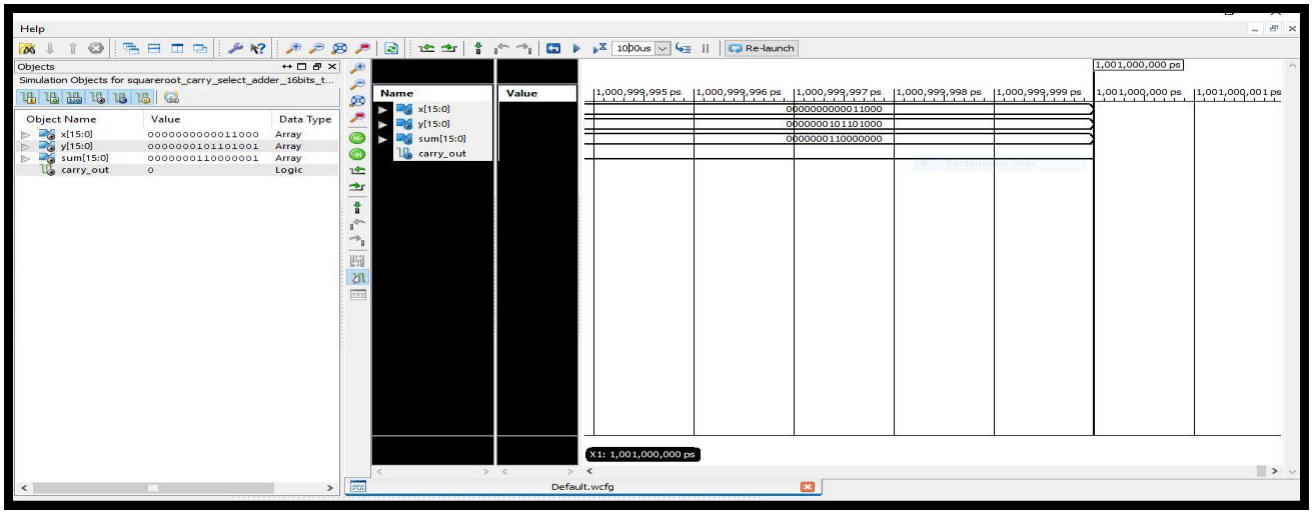

Fig-30 Test Bench Output Simulation Analysis of The SQRT CSLA Entity

(6) Speed Analysis of Different Multipliers

As discussed earlier the speed of the multiplier entity depends upon the delay taken by the adder incorporated inside it so the adder delay solely decides the speed of the multiplier, that means for comparison of multiplier speed we can easily compare only the delay time taken by the incorporated adder. By our statement, it is also clear that the speed of the multiplier is inversely proportional to the delay of the adder. The more the delay of the adder lesser will be the speed of the multiplier and vice versa. The equation can be written as follows.

[$S \propto 1/D$ (S = Speed of the multiplier entity, D = Delay time taken by the adder entity)]

ADDERS	DELAYS (Nano-Sec)
16-Bit Square Root Carry Select Adder (SQRT CSLA)	6.274
16-Bit Carry Look Ahead Adder (CLA)	7.031
16-Bit Carry Select Adder (CSLA)	7.328
16-Bit Ripple Carry Adder (RCA)	16.784

Table 1 – Delay Comparison of Different Adders

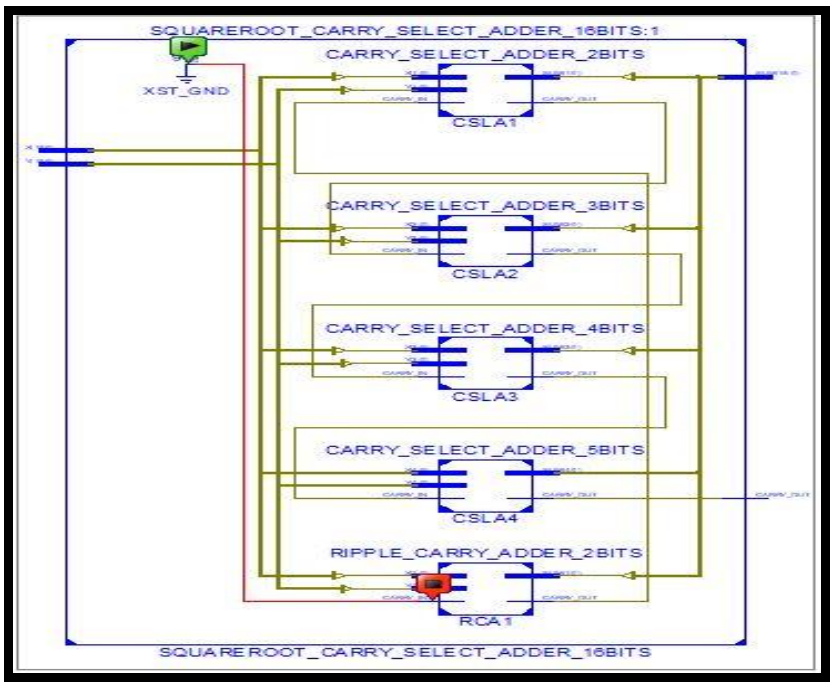

Fig-31 SQRT CSLA RTL Schema

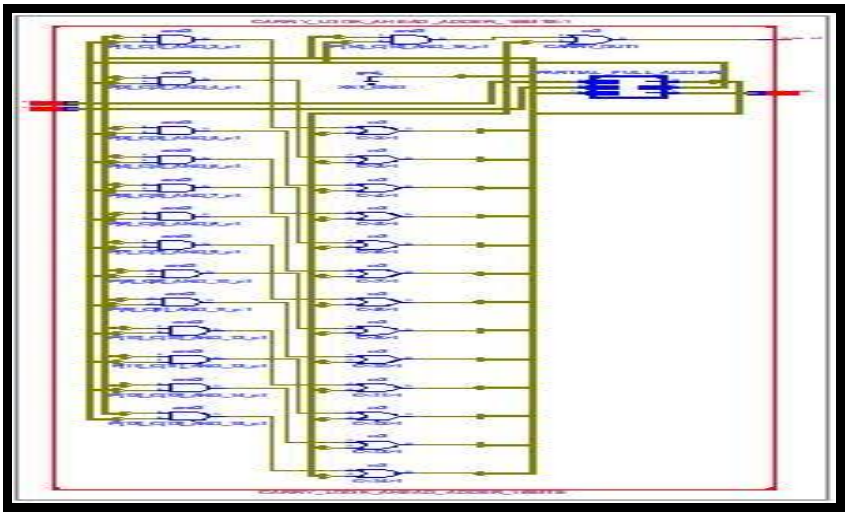

Fig-32 CLA RTL Schema

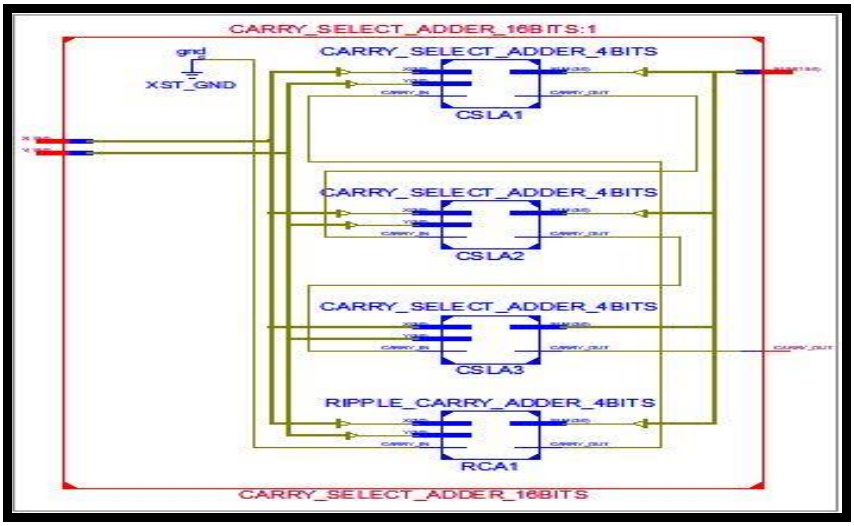

Fig-33 CSLA RTL Schema

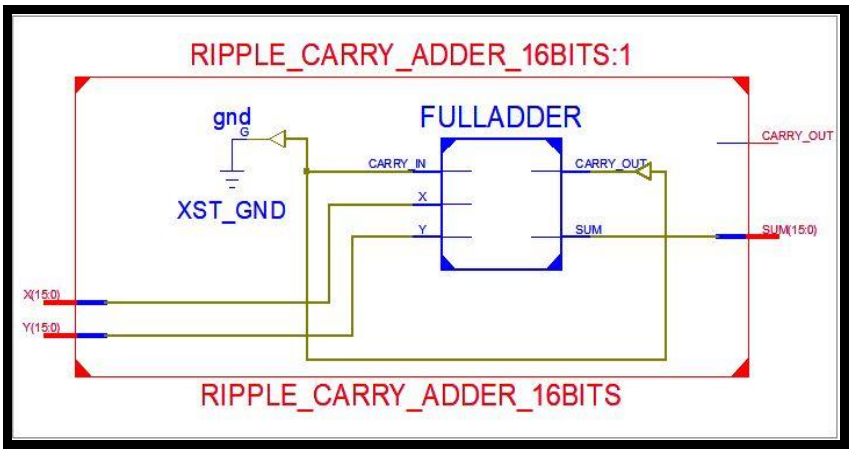

Fig-34 RCA RTL Schema

So according to the comparison, we can state the following hierarchy level on the basis of delays, speed

Delay of SQRT CSLA < Delay of CSLA < Delay of RCA

Speed of SQRT CSLA > Speed of CSLA > Speed of RCA

This means the multiplier which will have SQRT CSLA incorporated in it, will be having the greatest speed, and the multiplier which will have RCA incorporated in it, will be having the lowest speed. We have got the following data after comparing the respective multipliers' delays.

MULTIPLIERS	DELAYS (Nano-Sec)
16 Bit SQRT CSLA Based Multiplier	345.000
16 Bit SQRT CLA Based Multiplier	345.020
16 Bit CSLA Based Multiplier	345.040
16 Bit RCA Based Multiplier	345.060

Table 2 – Delay comparison of Multipliers

(7) Software IDEs & Programming Language Used

We have used XILINX ISE suit 14.7 as our IDE. Xilinx ISE (Integrated Software Environment) Suite 14.7 is a software toolchain provided by Xilinx, a leading manufacturer of Field-Programmable Gate Arrays (FPGAs) and Programmable Logic Devices (PLDs). ISE Suite 14.7 is one of the versions of Xilinx ISE and was released in 2013. Xilinx ISE Suite 14.7 includes a comprehensive set of design tools for FPGA and CPLD development. Some of its features like Design Entry: It provides various options for designing digital circuits, including schematic-based entry, Hardware Description Languages (HDL) like VHDL and Verilog, and graphical design tools like Xilinx Platform Studio (XPS).Synthesis: The software includes Xilinx Synthesis Technology (XST), which converts the RTL (Register Transfer Level) description into a gate-level representation. Simulation: ISE Suite 14.7 offers a simulator called ISim, which allows users to simulate and verify their designs before implementation. It supports both VHDL and Verilog simulations. IP Cores: Xilinx ISE Suite 14.7 includes a range of pre-designed Intellectual Property (IP) cores, such as memory controllers, communication interfaces, and DSP blocks, which can be integrated into the user's designs, these features helped in executing the project.

We have used VHDL as the programming and designing language. VHDL (VHSIC Hardware Description Language) is a hardware description language used in digital circuit design and simulation. It stands for "VHSIC" (Very High-Speed Integrated Circuits) as it was initially developed in the 1980s by the U.S. Department of Défense for designing complex integrated circuits. VHDL is widely used in the field of digital design, especially for the development of digital systems using programmable logic devices like Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs). It allows designers to describe the behavior and structure of digital circuits at various levels of abstraction, enabling the design and simulation of complex systems. Some key features and uses of VHDL include Behavioral Modeling: VHDL allows designers to describe the behavior of a digital circuit using concurrent and sequential processes. Behavioral models specify how the circuit should operate and react to different inputs. Structural Modeling:

VHDL supports structural modeling, which enables the description of circuits in terms of interconnected components and their interconnections. This allows for modular and hierarchical design, facilitating reuse and design optimization. Data Types and Operators: VHDL provides a range of data types, including standard logic types, integer types,

real types, and arrays. It also supports various operators for performing arithmetic, logical, and bitwise operations on these data types. Simulation: VHDL is commonly used for the simulation of digital circuits before their physical implementation. Simulation allows designers to verify the correctness and functionality of their designs and perform various test scenarios. Synthesis: VHDL is also used for synthesis, which is the process of converting a high-level design description into a gate-level representation that can be implemented on an FPGA or ASIC device. Synthesis tools analyze the VHDL code and generate a netlist of logic gates and flip-flops. Design Reuse: VHDL supports the concept of libraries and packages, allowing for the organization and reuse of commonly used design entities, functions, and procedures. This promotes modularity and efficiency in design. Verification and Testbenches:

VHDL provides constructs for creating test benches, which are sets of stimuli and expected outputs used to verify the behavior of a design during simulation. Testbenches enable designers to thoroughly test their circuits and validate their functionality. VHDL is a standardized language (IEEE Standard 1076) and is widely supported by electronic design automation (EDA) tools from various vendors, making it a popular choice for digital design and verification in both industry and academia.

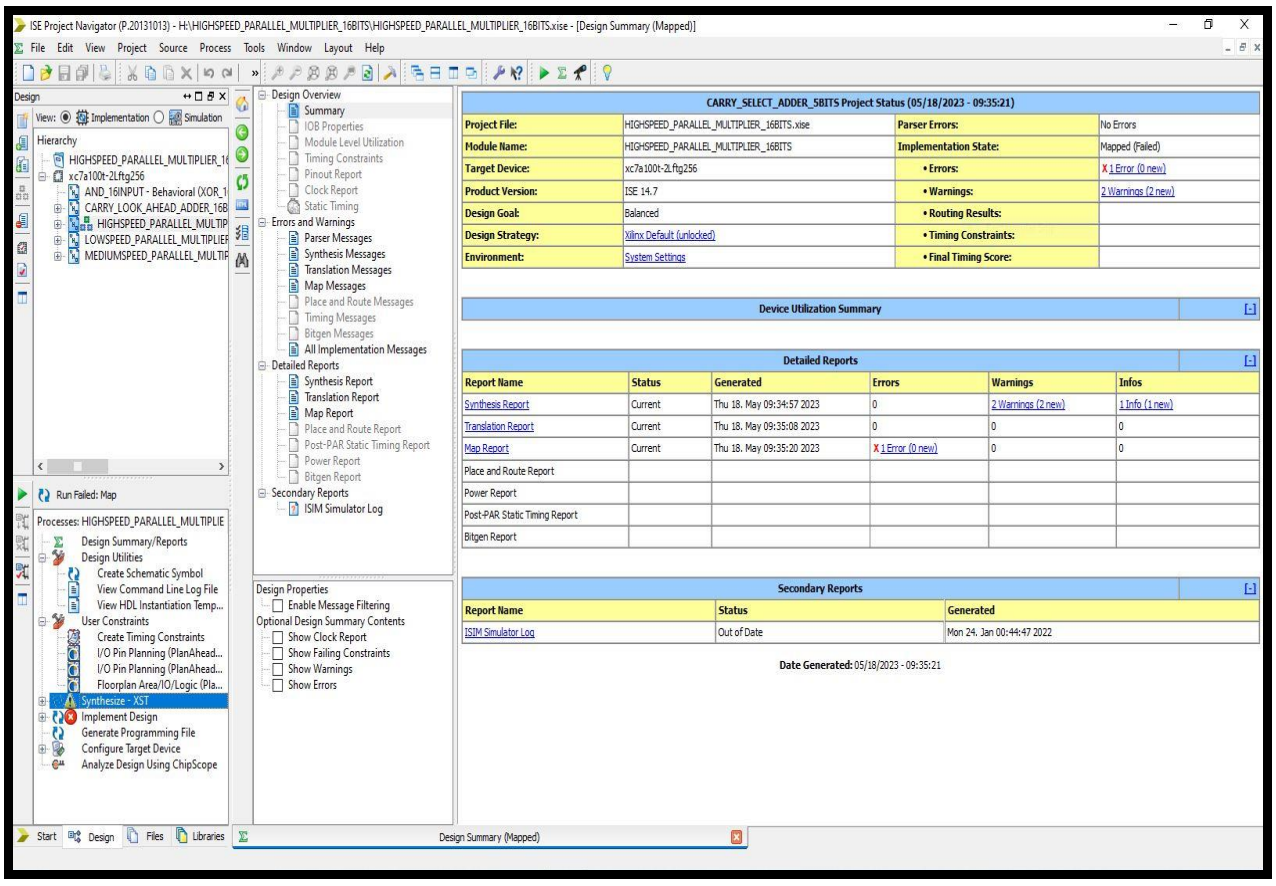

Fig-35 XILINX ISE Suite 14.7 Interface & Element Utilization Summary

(8) Scope for Further Development

High-speed parallel multipliers play a crucial role in various fields, including digital signal processing, cryptography, computer graphics, and scientific computing. As technology continues to advance, there are several potential future scopes for high-speed parallel multipliers: Emerging Computing Architectures: The development of new computing architectures, such as neuromorphic computing or quantum computing, may require innovative approaches to high-speed parallel multiplication. Adapting existing parallel multiplier designs or developing new algorithms specific to these architectures could be a future focus. Advanced Manufacturing Technologies: As manufacturing technologies advance, such as the use of nanoscale and 3D printing techniques, it opens up possibilities for creating intricate and high-performance multiplier architectures. The future scope lies in exploring new fabrication methods to build complex parallel multiplier structures that can achieve even higher speeds and lower power consumption. Optimized Algorithms: Research can focus on developing more efficient algorithms for parallel multiplication. This includes exploring novel techniques to reduce critical path delays, minimize power consumption, and optimize resource utilization. New algorithmic approaches could significantly improve the speed and efficiency of parallel multipliers. Hardware-Software Co-Design: Future scope lies in designing parallel multipliers in close collaboration with software and system-level considerations. Optimizing the entire hardware-software stack can lead to improved performance and efficiency. Exploring specialized instruction sets, compiler optimizations, and hardware accelerators tailored for parallel multiplication tasks can be areas of focus. Integration with AI and Machine Learning: With the increasing demand for AI and machine learning applications, integrating high-speed parallel multipliers with neural networks and deep learning algorithms can be a future direction. This includes exploring efficient multiplication techniques for matrix operations and tensor calculations, which are fundamental in many AI algorithms. Low-Power Design: Energy efficiency and power consumption are critical considerations in modern computing systems. Future scope lies in developing low-power designs for high-speed parallel multipliers, leveraging techniques such as voltage scaling, power gating, and dynamic power management. Exploring alternative arithmetic representations like approximate computing may also be beneficial. Hybrid Multiplication Techniques: Combining multiple multiplication techniques, such as parallel multipliers, sequential multipliers, and logarithmic number systems, can lead to higher performance and flexibility. Exploring hybrid multiplication approaches that leverage the strengths of different techniques can be a fruitful area of research. System-Level Integration: Future scope lies in

considering parallel multipliers as part of larger system designs. This includes exploring efficient interconnects, memory hierarchies, and data flow optimizations to maximize the overall system performance. Integration with other system components, such as processors, GPUs, or FPGAs, can also be an important aspect. Overall, the future scope for high-speed parallel multipliers lies in a multidisciplinary approach, combining algorithmic innovations, hardware design optimizations, and integration with emerging technologies. Continued research and development in these areas can lead to significant advancements in high-performance computing and enable a wide range of applications.

(9) Conclusion

High-speed multipliers are essential components in digital systems that perform multiplication operations efficiently. In conclusion, the choice of a high-speed multiplier depends on several factors and trade-offs, including performance, area utilization, power consumption, and implementation complexity. Ultimately, the choice of a high-speed multiplier depends on the specific requirements of the application, such as target clock frequency, area constraints, and power consumption limitations. Designers need to carefully evaluate these factors and select the most suitable multiplier architecture that strikes a balance between speed, area utilization, and complexity for their particular design scenario. The performance of multiplication in terms of speed and power is crucial for most Digital Signal Processing (DSP) applications. Many researchers have come up with various multipliers such as an array, Booth, carry save, Wallace tree, and modified Booth multipliers. However, for present-day applications, high-speed multipliers based on special fast adders are presently under focus due to their high speed and low power consumption. In this project, we propose a design of a 16-bit multiplier using a fast adder (square root carry select adder) to minimize the power-delay product of a multiplier intended for high-performance and low-power applications. The implementation result demonstrates that the proposed high-speed multiplier with SQRT CSLA significantly improves its delay.

(10) References

V. Vijayalakshmi, R. Seshadri, S. Ramakrishnan "Design and implementation of 32-bit unsigned multiplier using CLAA and CSLA". Published in 2013 International Conference on Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT)Date of Conference: 07-09 January 2013, Conference Location: Tiruvannamalai, India, Date Added to IEEE Xplore: 11 April 2013 ISBN Information Electronic ISBN:978-1-4673-5301-4, Print ISBN:978-1-4673-5300-7, CD:978-1-4673-5299-4, INSPEC Accession Number: 13431607, Publisher: IEEE, DOI: 10.1109/ICEVENT.2013.6496579

Anbumani V; Soviya S; Sneha S; Saran L "Speed and Power Efficient Vedic Multiplier using Adders with MUX", Published in 2021 Innovations in Power and Advanced Computing Technologies (i-PACT), Date of Conference: 27-29 November 2021, Conference Location: Kuala Lumpur, Malaysia, Date Added to IEEE Xplore: 08 February 2022, ISBN Information: Electronic ISBN:978-1-6654-2691-6, Print on Demand(PoD) ISBN:978-1-6654-2692-3, INSPEC Accession Number: 21720266, Publisher: IEEE, DOI: 10.1109/i-PACT52855.2021.9696992.

M Gopi; G B S R Naidu "128 Bit unsigned multiplier design and implementation using an efficient SQRT-CSLA", Published in 2015 13th International Conference on Electromagnetic Interference and Compatibility (INCEMIC), Date of Conference: 22-23 July 2015, Conference Location: Visakhapatnam, India, Date Added to IEEE Xplore: 05 October 2017, ISBN Information: Electronic ISBN:978-1-5090-5350-6, Print on Demand(PoD) ISBN:978-1-5090-5351-3, Electronic ISSN: 2573-3303, INSPEC Accession Number: 17239385, Publisher: IEEE, DOI: 10.1109/INCEMIC.2015.8055889

S. Manju; V. Sornagopal "An efficient SQRT architecture of Carry Select adder design by Common Boolean logic", Published in 2013 International Conference on Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT), Date of Conference: 07-09 January 2013, Conference Location: Tiruvannamalai, India, Date Added to IEEE Xplore: 11 April 2013, ISBN Information: INSPEC Accession, Number: 13431618, Publisher: IEEE, DOI: 10.1109/ICEVENT.2013.6496590

Sumod Abraham; Sukhmeet Kaur; Shivani Singh, "Study of various high-speed multipliers", Date of Conference: 08-10 January 2015, Date Added to IEEE Xplore: 24 August 2015, ISBN Information: Electronic ISBN:978-1-4799-6805-3, Print ISBN:978-1-4799-6804-6, CD:978-1-4799-6803-9, INSPEC Accession Number: 15385477, Publisher: IEEE, Conference Location: Coimbatore, India, DOI: 10.1109/ICCCI.2015.7218139