# Design and Analysis of High Performance Multipliers Using VHDL

**V.Karuppasamy**
Assistant Professor
Department of Electronics and Communication Engineering
S.Veerasamy Chettiar College of Engineering & Technology Puliangudi-627 855, Tamilnadu

*Abstract*— Today every digital circuit has to face the power consumption issues. The design of an efficient integrated circuit in terms of power, area and speed simultaneously, has become a very challenging problem. Power dissipation is recognized as a critical parameter in modern VLSI design field. Multiplication is a binary mathematical operation of scaling one number by another. A fast and energy efficient multiplier is always needed in electronics industry especially DSP, image processing and arithmetic units in microprocessors. In this work high performance multiplier architecture is proposed, by the use of shift-and add and booth multiplier approaches. Low power consumption can be achieved through some modifications in the conventional multiplier architectures. By the use of these multiplier approaches, power consumption can be achieved through the reduction of partial product generation. The proposed multiplier architectures will reduce the number stages in multiplication and also reduces the propagation delay in digital circuits. The system will be designed by using VHDL (Very High speed integrated circuit Hardware Descriptive Language).

***Key words:*** VHDL, DSP, VLSI

## I. INTRODUCTION

### A. *Multiplication:*

Multiplication is an important fundamental function in arithmetic operations. Multiplication-based operations such as Multiply and Accumulate (MAC) and inner product are among some of the frequently used Computation-Intensive Arithmetic Functions (CIAF) currently implemented in many Digital Signal Processing (DSP) applications such as convolution,Fast Fourier Transform(FFT),Filtering (FIR,IIR) and in microprocessors in its arithmetic and logic unit.

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

### B. *Basics of Multiplier:*

Multiplication is a mathematical operation that at its simplest is an abbreviated process of adding an integer to itself a specified number of times.A number (multiplicand) is added to itself a number of times as specified by another number (multiplier) to form a result (product).The multiplicand is then multiplied by each digit of the multiplier beginning with the rightmost, Least Significant Digit (LSD). Intermediate results (partial products) are place done a top the other, offset by one digit to align digits of the same weight.

The final product is determined by summation of all the partial-products. Here, we assume that MSB represent the sign of the digit. The operation of multiplications rather simple in digital electronics. We can check initial stage also that whether the product will be positive or negative or after getting the whole result, MSB of the results tells the sign of the product.

This algorithm uses addition and shift left operation to calculate the product of two numbers. Based upon the above procedure, we can deduce an algorithm for find any kind of multiplication.

### C. *Multiplicand:*

The Multiplicand block is composed of 8 D Flip-Flop blocks, which store the A byte for processing during the complete multiplication cycle. The register is loaded with the LOAD-cmd signal from the controller. The basic design for the multiplicand block is that of an 8-bit register.The top level multiplicand and module generates an 8-bit register from individual 1-bit D Flip-Flops.The individual D Flip-Flops have been designed both structurally and behaviorally and the synthesized results are compared.

The detailed diagram of the Multiplicand module the byte is loaded into the register only when the LOAD-cmd is received from the controller and the register is cleared when a global reset is applied.

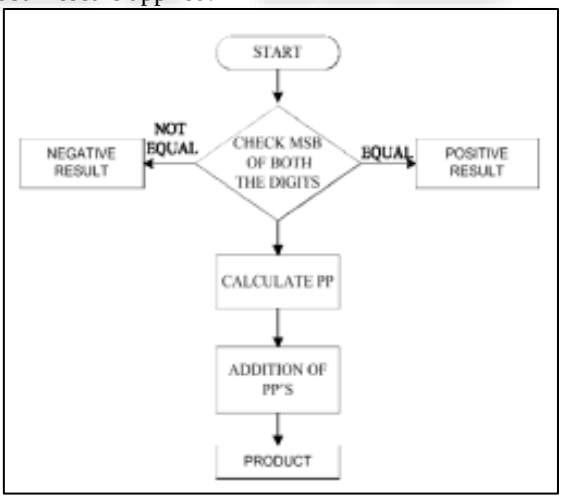


Fig. 1: Structure of basic multiplier

### D. *Binary multiplication:*

In the binary number system the digits, called bits, are limited to the set [0,1]. The result of multiplying any binary number by a single binary bit is either 0, or the original number. This makes forming the intermediate partial-products simple and efficient. Summing these partial-products is the time consuming task for binary multipliers. One logical approach is to form the partial-products one at a time and sum them as they are generated. Often implemented by software on

 

processors that do not have a hardware multiplier, this technique works fine, but is slow because at least one machine cycle is required to sum each additional partial-product. For applications where this approach does not provide enough performance, multipliers can be implemented directly in hardware.

The two main categories of binary multiplication include signed and unsigned numbers.

### *E. Multiplication process:*

The simplest multiplication operation is to directly calculate the product of two numbers by hand. This procedure can be divided into three steps: partial product generation, partial product reduction and the final addition. To further specify the operation process, let us calculate the product of two two's complement numbers, for example, 1101two (−3ten) and 0101two (5ten), when computing the product by hand.

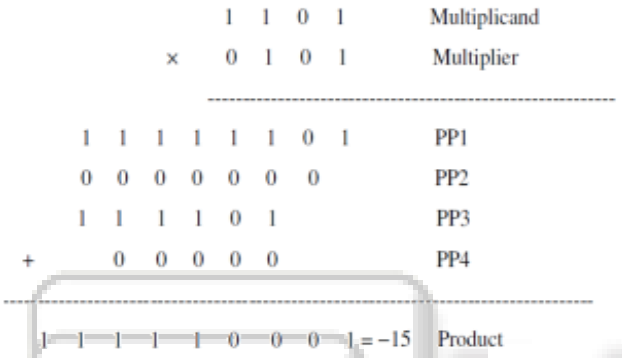


Fig. 2: Multiplication calculations by hand

The two rows before the product are called sum and carry bits. The operation of this method is to take one of the multiplier bits at a time from right to left, multiplying the multiplicand by the single bit of the multiplier and shifting the intermediate product one position to the left of the earlier intermediate products. All the bits of the partial products in each column are added to obtain two bits: sum and carry.

Finally, the sum and carry bits in each column have to be summed. Similarly, for the multiplication of an n-bit multiplicand and an m-bit multiplier, a product with n + m bits long and m partial products can be generated.

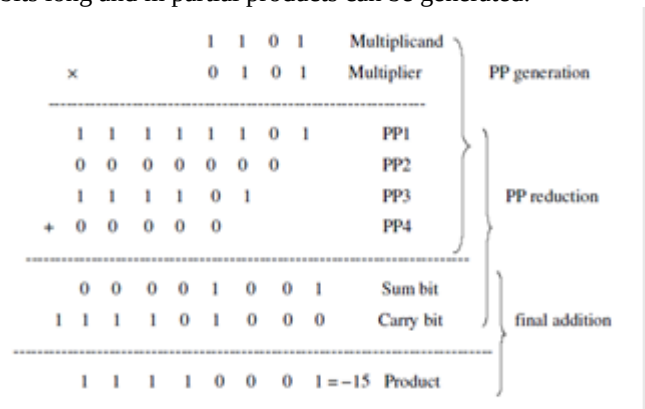


Fig. 3: Multiplication 0peration in hardware

### *F. VHDL Introduction:*

VHDL (Very High Speed IC Hardware description Language) is one of the standard hardware description language used to design digital systems. VHDL can be used to design the lowest level (gate level) of a digital system to the highest level (VLSI module). VHDL though being a rigid language with a standard set of rules allows the designer to use different methods of design giving different perspectives to the digital system. Other than VHDL there are many hardware description languages available in the market for the digital designers such as Verilog, ABEL, PALASM, CUPL, and etc but VHDL and Verilog are the most widely used HDLs. The major difference between hardware description programming languages and others is the integration of time. Timing specifications are used to incorporate propagation delays present in the system.

## II. SHIFT AND ADD MULTIPLIER

### *A. General Requirements:*

The requirement is to design a multiplier based on the shift and add method. The multiplier shall accept as inputs a multiplier and multiplicand as well as a Start signal. The multiplier shall then calculate the result using the shift and add method and provide the result along with a Stop signal. The design shall be coded in VHDL and simulated for proper functionality and timing.

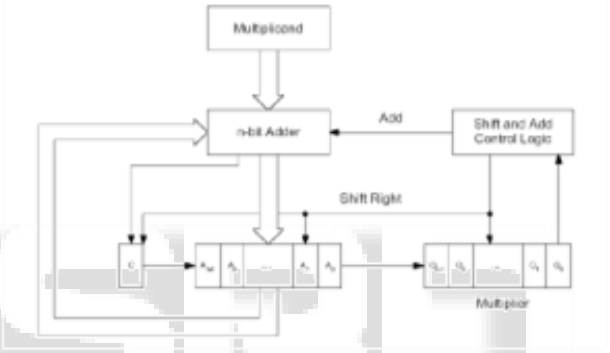


Fig. 4: Add/Shift Multiplier Block Diagram

### *B. Design Specifications:*

The design was implemented using a mixture of both structural design and RTL level design. In the following sections each of the modules are described in greater detail and associated diagrams, simulation outputs and timing are provided.

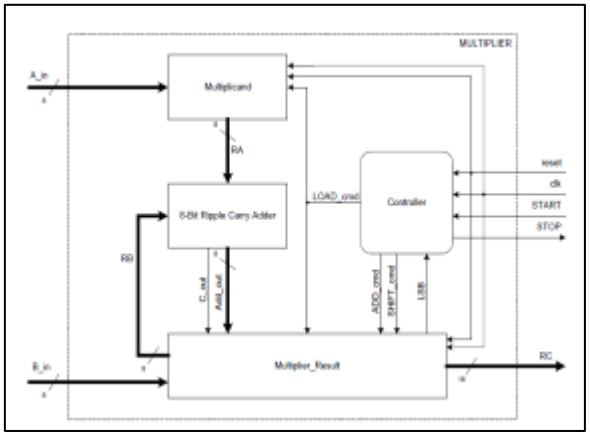


Fig. 5: Multiplier Design Block Diagram

### *C. Controller Design:*

The Controller is the control unit of the multiplier. It receives a START signal and consequently commands all other modules until the result is obtained and it outputs a STOP signal. The design was implemented as a finite state machine with states and transition logic The Start signal transitions the

 

state machine out of the idle state and into the initialize state whereby it commands the multiplicand and multiplier to be loaded into registers. Once loaded, the state machine goes through a series of test and shift, or test, add and shift operations depending on the status of the LSB bit. Upon reaching the maximum count for the multiplication cycle, the state machine goes back to the idle state and outputs a Stop signal.


Fig. 6: Controller Diagram

*D. Multiplicand Design:*

The Multiplicand block is composed of 8 D Flip-Flop blocks, which store the .A. byte for processing during the complete multiplication cycle. The register is loaded with the LOAD_cmd signal from the Controller. The basic design for the Multiplicand block is that of an 8-bit register. The top-level multiplicand module generates an 8-bit register from individual 1-bit D Flip-Flops. The individual D flip-flops have been designed both structurally and behaviorally and the synthesized results are compared. The byte is loaded into the register only when the LOAD_cmd is received from the Controller and the register is cleared when a global reset is applied.

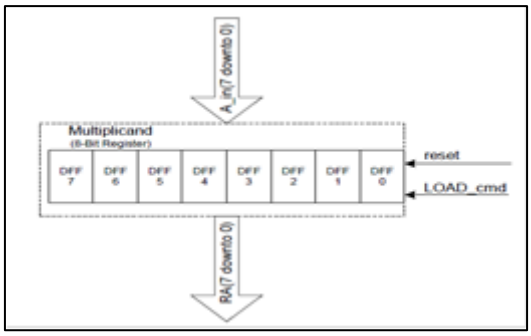


Fig. 6: Multiplicand Diagram

*E. Multiplier Design:*

The Multiplier/Result block stores the multiplier as well as the accumulated output of the adder. It allows the register to be logically shifted right and provides one of the Adder's inputs. The Multiplier/Result block consists of a 17-bit shift register and a multiplexer in order to provide this functionality. The design of the Multiplier/Result module was performed at the RTL level.The input to the module is the multiplier, B_in, which is loaded into bits 0 to 7 of the register when the LOAD_cmd is asserted.LSB is fed back to the controller to determine the next state, while RB is fed into the adder in order to be summed with the multiplicand. RC is the final multiplication result and is considered valid only when the controller asserts the STOP signal. If the Multiplier_Result receives a SHIFT_cmd without a prior ADD_cmd, the register will be shifted logically to the right. If the ADD_cmd precedes the SHIFT_cmd, bits 1 to 7 of the register will be placed into positions 0 to 6 while the 9 inputs bits from the adder will be placed into positions 7 to 15 of the register. This is essentially equivalent to storing the adder results and then shifting the entire register.

## III. BOOTH MULTIPLIER

Conventional array multipliers, like the Braun multiplier and Baugh Woolley multiplier achieve comparatively good performance but they require large area of silicon, unlike the add shift algorithms, which require less hardware and exhibit poorer performance. The Booth multiplier makes use of Booth encoding algorithm in order to reduce the number of partial products by considering two bits of the multiplier at a time, thereby achieving a speed advantage over other multiplier architectures.

This algorithm is valid for both signed and unsigned numbers. It accepts the number in 2's complement form, based on radix-2 computation. It can handle signed binary multiplication by using 2's complement representation. This increases the complexity of how signs of the operands get stored in auxiliary circuits.

*A. Non-Booth Recoding:*

Using the non-Booth encoding method for partial product generation, the multiplier bits are examined sequentially starting from LSB to MSB. If the multiplier bit is one, the partial product is simply the multiplicand. Otherwise, the partial product is zero.

Each new partial product is shifted one bit position to the left. Each partial product can be produced by just using a row of two-input AND gates. The number of partial products generated equals the size of the multiplier bit

*B. Booth Algorithm:*

*1) Booth Recoding:*

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. The algorithm was invented by Andrew Donald Booth in 1951 while doing research on crystallography at Birkbeck College in Bloomsbury, London.

Booth used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed. Booth's algorithm is of interest in the study of computer architecture. This method can be used to multiply two two's complement number without the sign bit extension. Booth recoding is a method of reducing the number of partial products to be summed. Booth observed that when strings of '1' bits occur in the multiplicand the number of partial products can be reduced by using subtraction.

The operation of Booth recoding consists of two major steps: the first one is to take one bit of the multiplier, and then to decide whether to add the multiplicand according to the current and previous bits of the multiplier. This encoding scheme is serial, which means that the different value of the 2-bits (current and previous bits) corresponds to the different operations.

 

| $X_i$ | $X_{i-1}$ | Operations | Comments | $Y_i$ |
|---|---|---|---|---|
| 0 | 0 | Shift only | String of zeros | 0 |
| 1 | 1 | Shift only | String of zeros | 0 |
| 1 | 0 | Subtract and Shift | Beginning of string of ones | 1 |
| 0 | 1 | Add and Shift | End of string of ones | 1 |

Table 1: Recoding in Booth Algorithm

The serial recoding scheme is usually applied in serial multipliers. The advantage of this method is that the partial product circuit is simple and easy to implement. Therefore, this scheme is suitable for the implementation of small multipliers. The drawback is that the method is not able to efficiently handle the sign extension and it generates a number of partial products as many as the number of bits of the multiplier, which results in many adders needed so that the area and power consumption increase. This method is not applicable for large multipliers. An example of booth multiplier which multiply the following:

Multiplicand (a) = (-3)

Multiplier (b) = (-5)

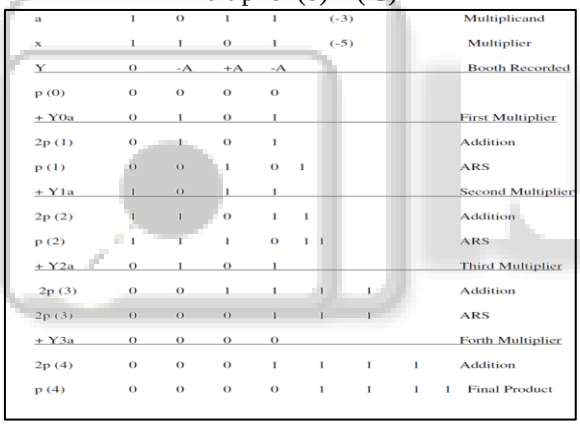


Fig. 7: An example of booth multiplier

## IV. WALLACE TREE MULTIPLIER

Wallace tree multiplier consists of three step process, in the first step, the bit product terms are formed after the multiplication of the bits of multiplicand and multiplier, in second step, the bit product matrix is reduced to lower number of rows using half and full adders, this process continues till the last addition remains, in the final step, final addition is done using adders to obtain the result [1].

Wallace tree reduces the number of partial products to be added into 2 final intermediate results.The Wallace tree basically multiplies two unsigned integers,A Wallace tree is an efficient hardware implementation of a digital circuit that multiplies two integers, devised by an Australian Computer Scientist Chris in 1964.

The Wallace tree has three steps:

1) Partial Product Generation Stage
2) Partial Product Reduction Stage
3) Partial Product Addition Stage

### *A. Half adder:*

Half adder is a combinational arithmetic circuit that adds two numbers and produces a sum bit (S) and carry bit (C) as the output. If A and B are the input bits, then sum bit (S) is the X-OR of A and B and the carry bit (C) will be the AND of A and B. From this it is clear that a half adder circuit can be easily constructed using one X-OR gate and one AND gate.

| INPUT | | OUTPUT | |
|---|---|---|---|
| A | B | SUM | CARRY |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Table 2: Truth table of half adder

### *B. Full adder:*

A full adder adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds three one-bit numbers, often written as A,B, and Cin; A and B are the operands, and Cin is a bit carried in from the next less significant stage.

| INPUT | | | OUTPUT | |
|---|---|---|---|---|
| A | B | $C_{IN}$ | SUM | CARRY |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Table 3: Truth table of full adder

### *C. Ripple carry adder:*

A simple ripple carry adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the carry output from each full adder connected to the carry input of the next full adder in the chain significant bits.The delay increase with the increase in the number of bits to be added.

### *D. Circuit Implementation:*

#### *1) Circuit Description:*

We implement our design in Cadence 5.0 environment with NCSU Free PDK 45 nm technology.

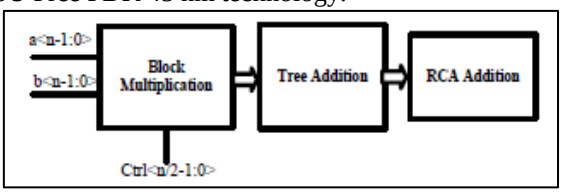


Fig. 8: Three Top level structure

Each n/k block is further divided into n/(2k) blocks until each block contains 2 bit inputs. Then the 2 bit multiplier will calculate the small partial products and accumulate recursively until it is the partial product of the n/k block.,The 32 bit multiplier divided into 4 blocks for each multiplicand will end up with 64 8-bit partial products. The lower left triangle colored in green indicates that these blocks could be potentially turned off to achieve power savings.

 

*2) Wallace Tree Accumulation:*

Tree addition is the process of compressing the partial products. Instead of having 64 1-bit partial products for conventional 8-bit multiplier, we just have 16 4-bit partial products.

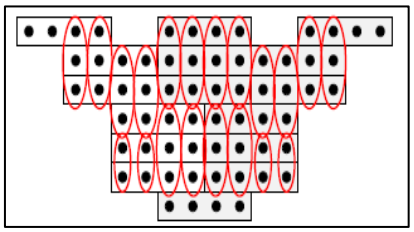

Fig. 9: Tree Addition

We follow the same way as Wallace tree does to compress the partial products. 3 to 2 compressors and 2 to 1 compressors are used.

*3) Ripple carry Addition:*

After several steps of block partial products compressing, there are just two levels of partial products left. We use RCA Addition block to add the two levels and get the final products. The block is simply a ripple carry adder. This adder can be replaced as prefix adder. Considering the power consumption, we choose the ripple carry adder. By utilizing the inversion property of ripple carry adder, we can achieve lower power consumption.

*8x8 Wallace Tree Multiplier*:

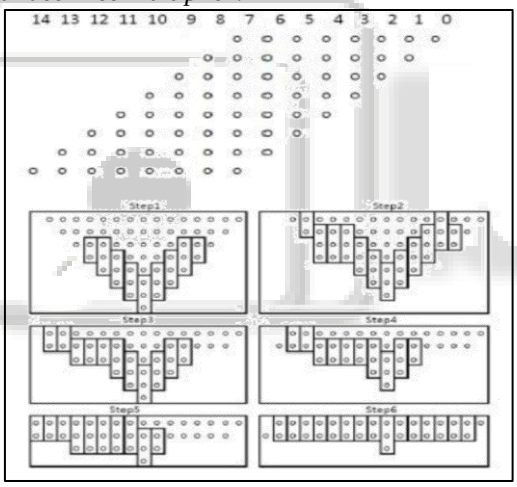


Fig. 10: Example of 8X8 multiplier

Shows the basic architecture and the steps involved in 8x8 wallace tree multiplier. The procedure of multiplication is same as that of 4x4 wallace multiplier. In the Wallace Tree method, the circuit is quite irregular although the speed of operation is high.

## V. SIMULATION RESULTS

*A. Shift and Add Multiplier output:*

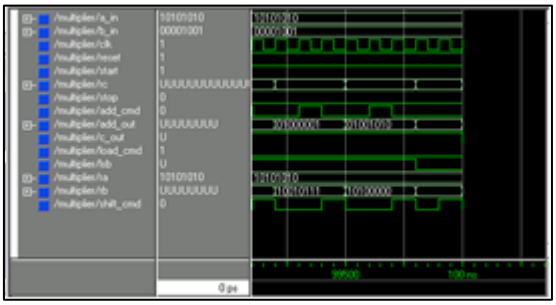

Fig. 11: Shift and Add Multiplier output

*1) Controller output:*

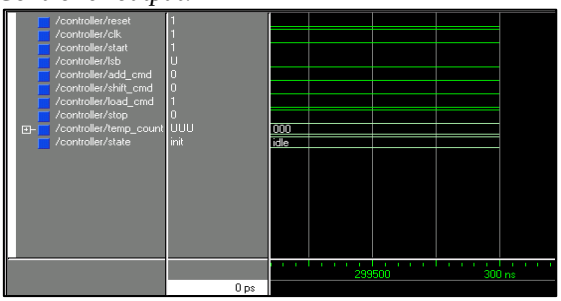


Fig. 12: Controller output

*2) Multiplicand output:*

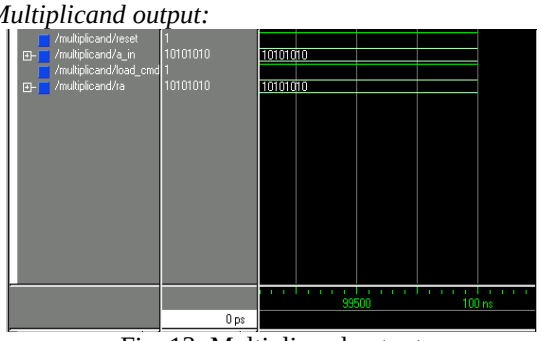


Fig. 13: Multiplicand output

*3) Multiplier output:*

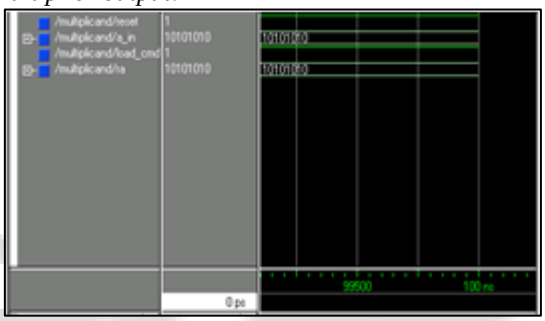

Fig. 14: Multiplier output

*B. Booth Multiplier output:*

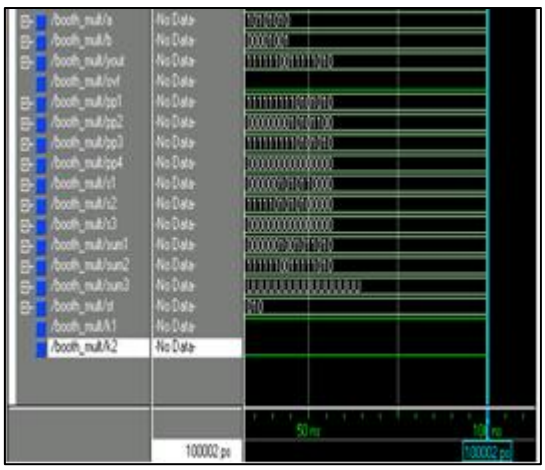

Fig. 15: Booth Multiplier output

*1) Booth Encoder Output:*

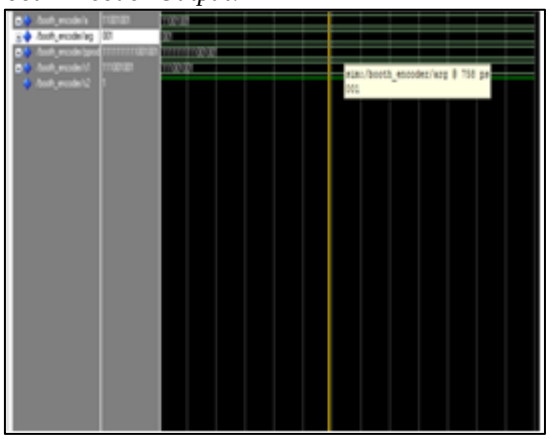

Fig. 16: Booth Encoder output

*C. Wallace Tree Multiplier output:*

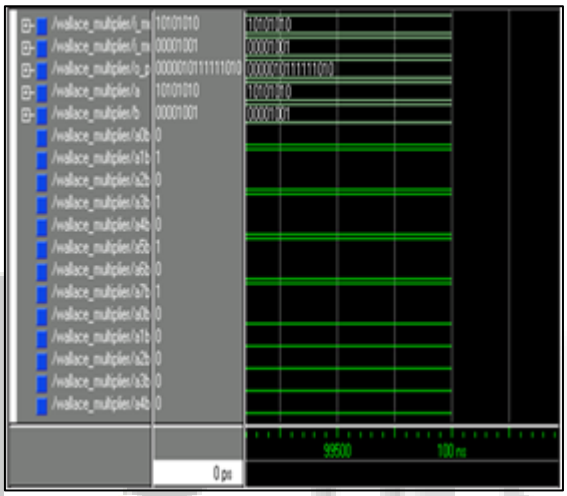

Fig. 17: Wallace Tree multiplier

*D. Comparison of Shift and Add, Booth and Wallace Tree Multipliers:*

| | Shift and Add Multiplier | Booth Multiplier | Wallace Tree Multiplier |
|---|---|---|---|
| Generic Clock Buffer | 8 | 4 | 4 |
| Utilization Ratio | 100 | 100 | 100 |
| LUT | 58 | 245 | 127 |
| Minimum Period | 5.883ns | No Path Found | No Path Found |
| Delay | 5.883ns | 39.293ns | 24.028ns |
| Fan out | 500 | 100 | 100 |
| Total Memory Usage | 70112Kbytes | 60624Kbytes | 58576Kbytes |

Table 4: Comparison Table of Shift and Add,Booth and Wallace Tree multipliers

## VI. CONCLUSION

In this work, Booth multiplier, Shift and Add multiplier and Wallace tree multiplier were designed and the performance parameter such as Generic clock buffer, utilization ratio, LUT, delay, fan out, total memory usage were analyzed.

In the future we have to design a FIR filter and apply the Booth, Shift and Add and Wallace tree multipliers instead of the conventional multipliers in the filter. Finally compare and analyze the FIR filter and find the best multiplier for better power consumption and high speed of operation.

## REFERENCES

[1] Arunachalam.T and Kirubaveni.S "Analysis of high speed multipliers" International Conference on Communications and Signal Processing (ICCSP), 2013.
[2] Rao M.J. and Dubey.S. "A high speed and area efficient Booth recoded Wallace tree multiplier for fast arithmetic circuits "Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia), 2012.
[3] Rajput,R.P. ; Swamy, M.N.S." High Speed Modified Booth Encoder Multiplier for Signed and Unsigned Numbers" International Conference on Computer Modelling and Simulation (UKSim), 2012.
[4] M. Mottaghi-Dastjerdi, A. Afzali-Kusha, and M. Pedram, "BZ-FAD: A Low-Power Low-Area Multiplier Based on Shift-and-Add Architecture", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 17, No. 2, February 2009.
[5] Shiann-Rong Kuang, Member, IEEE, Jiun-Ping Wang, and Cang- Yuan Guo,"Modified Booth Multipliers With aRegular Partial Product Array", IEEE Transactions on Circuits and Systems: Express Briefs, Vol. 56, No. 5, May 2009.

 