Scientific Journal of Impact Factor (SJIF): 4.72

e-ISSN (O): 2348-4470
p-ISSN (P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 4, Issue 11, November -2017

Design of High Speed, Low Power and Area Efficient 32-Bit Floating Point Multiplier

R Kalaimathi[1], R Senthil Ganesh[2]

[1]*M.E VLSI Design, Info Institute of Engineering, Tamilnadu*
[2]*Assistant Professor, Electronics and Communication Engineering, Info Institute of Engineering, Tamilnadu*

Abstract — *Digital arithmetic operations are the most important in the design of Digital Signal Processing (DSP) and Application Specific Integrated Circuit (ASIC) systems. Addition and Multiplication are the most basic arithmetic operations. Floating point (FP) multiplication is widely used in large set of scientific and signal processing computation. The speed of floating point arithmetic unit is very crucial performance parameter which impinges the operation of the system. A fast and accurate operation of a digital system is greatly influenced by the performance of different adders and multipliers using different methodology. Emerging trends in low power made attention towards the search of low power, high speed and area efficient adders and multipliers architecture. In this project, single precision (32-bit) floating point multiplier is designed. The main focus of this design is to reduce area, power and path delay to enhance the speed of the multiplication and addition operation. For floating point multiplier, Booth recoded multiplier is used where it reduces the number of partial products and thereby increase the speed of multiplication of the mantissa bits. For partial products addition, Carry Look-Ahead adder (CLA) and Kogge-Stone Adder (KSA) is implemented separately. The area, delay, power, speed, Power Delay Product (PDP) and Energy Delay Product (EDP) of two different implementation results are compared. Xilinx and ModelSim tool is used for simulation and VHDL coding for the design.*

Keywords- *Radix-4 Booth Multiplier, Carry Look-Ahead Adder, Kogge-Stone Adder, Single Precision Floating Point, Area, Delay, Power*

I. INTRODUCTION

Digital arithmetic operations are very important in the design of DSP and ASIC systems. Floating point computation has been widely used in graphics, digital signal processing, image processing and other applications. Floating point multiplication is a critical module in many applications especially for graphic processing unit, image recognition, navigation system in radar for identification, tracking & detection and 3D model. This type of complex application requires more time to process the data. Therefore, the high speed multiplication unit for floating point numbers is highly recommended to speed up multiplication process [4].

Multiplier output data size is twice than the input data size and therefore it consumes large chip area density, high complexity and time consuming process. Many different types of design models have been introduced to perform only for multiplication. Every design models have different multiplier and adder algorithms which perform the same operation but with different performance in terms of calculating speed and resource consumption. As the technology is improving, many researchers are trying to develop efficient multiplier designs which can offer high speed or low power or low area or the combination of all these three in single multiplier [4].

In this paper, radix-4 MBE algorithm is implemented for multiplication operation. This multiplier offers the advantage of both Booth algorithms. Booth algorithm is specially used for signed multiplication. It also reduces the number of partial products. The resource consumption of radix-4 MBE algorithm is less compared to the Wallace tree multiplier. In the existing system, CLA is used for partial product addition. The proposed system uses the same radix-4 MBE for partial product generation and Parallel Prefix Adders (PPA) such as KSA is used for partial product addition. PPA offers highly efficient solution to the binary addition and suitable for VLSI implementations.

II. IEEE FLOATING POINT REPRESENTATION

In the early stage, fixed point representation was the easiest method to convert the real numbers to binary because fixed-point representation adheres to the same basic arithmetic principles as integers. Fixed point representation has limited range of values and exceeding the limit can cause data overflow. The floating point number is nothing but the radix point (decimal point) can be placed anywhere relative to the significant digits of the number. In general, floating point representations are slower and less accurate than fixed point representations. It has better precision and support a much wider range of values. The size of the floating point representation that can be stored is either 32-bit (single precision) or 64-bit (double precision) defined by IEEE 754 standard [5].

The multiplier unit used in this project performs multiplication of single precision floating point data. The standard IEEE 754 floating point format consists of a 32-bit and 64-bit vector split into three sections as shown in Figure

@IJAERD-2017, All rights Reserved

1 and Figure 2. The IEEE standards have been extraordinarily successful in ensuring a level of portability for computer arithmetic across a vast array of implementations and disparate architectures.

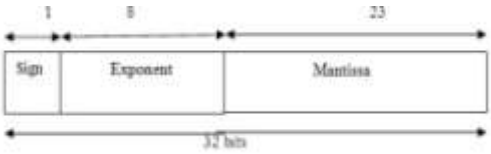

Figure 1. Single precision format

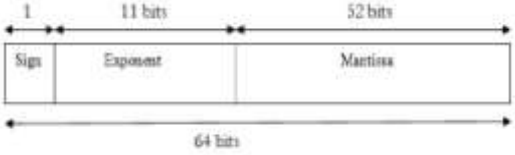

Figure 2. Double precision format

As illustrated in the Figure 1, IEEE binary single precision floating point format has 1 sign bit (S), an 8 exponent bits (E) and a 23 mantissa bits (M). The IEEE format requires normalization, since it uses radix-2, it is known a prior that the first bit of the mantissa is a 1, which means that it can be implied. The implied bit gives IEEE format an extra bit of mantissa and is called as significand1. If the exponent is greater than 0 and smaller than +127, and there is 1 in the MSB of the significand then the number is said to be a normalized number. Table 1 represents the bit range of the single and double precision floating point. The normalized floating point numbers have the form expressed in Equation (1) where $M = n_{22}\,2^{-1} + n_{21}\,2^{-2} + n_{20}\,2^{-3} + \ldots + n_1\,2^{-22} + n_0\,2^{-23}$ [5],

$$Z = (-1^{S}) * 2^{(E-Bias)} * (1.M) \quad (1)$$

Table 1. Bit range of single and double precision floating point

	Sign	Exponent	Emax	Emin	Mantissa	Bias
Single Precision	1 [31]	8 [30-23]	+127	-126	23 [22-0]	127
Double Precision	1 [63]	11 [62-52]	+1023	-1022	52 [51-0]	1023

2.1 Floating Point Multiplier Algorithm

The multiplication of floating point numbers X and Y is explained with the help of flow chart shown in the Figure 3. The floating point multiplication procedure is [6],

- Represent input numbers in binary IEEE 754 standard.
- Compute the sign of the input by taking the XOR of X and Y sign bit. If the XOR value is 1 then the sign is positive otherwise it is negative sign.
- Multiply the mantissa part and normalize the result. The output of the multiplication is 48 bits, so truncate it to 24 bits which is the allowed number of mantissa part.
- Add the exponent of X and Y and subtracting bias (127_{10}) from it. If the sum of exponent exceeds 127, it will result to overflow, then set exponent to 128 and mantissa to 0 gives positive or negative infinity. If the sum of exponent is more negative than 126 results overflow.

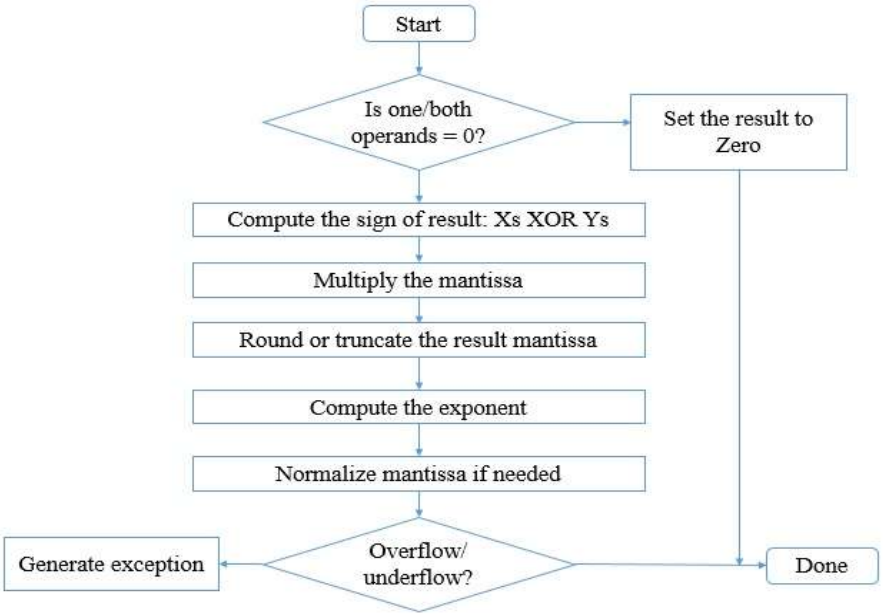

Figure 3. Floating point multiplication flowchart

III. BOOTH REPRESENTATION

Booth's algorithm is widely use to implement software and hardware multipliers because it reduces the number of partial product generation which means it reduces the multiplier bits. It can be used for signed magnitude numbers and

@IJAERD-2017, All rights Reserved

2's complement numbers by eliminating the correction term. A modified algorithm was proposed by Booth Mac Sorley in which a three bits is scanned. The Booth Mac Sorley algorithm, is termed as Modified Booth algorithm or Booth algorithm and can be generalized to any radix format. This technique reduces the number of partial products by one half irrespective of inputs. The recoding is performed within two steps such as encoding and selection. The encoding step scans the three bits of the multiplier and define the operation to be carried out on the multiplicand. The three bits scanning is termed as radix-4 representation [2]. For example, a 3-bit recoding would require the following set of digits to be multiplied by the multiplicand: 0, +1, -1, +2, (-2). Booth multiplication preserves the sign of the result and also allows for smaller, faster multiplication [2]. Booth Multiplier can be used in three distinct modes such as radix-2, radix-4, radix-8 etc. Of this, Radix-4 Booth's algorithm is most widely because of number of partial products is reduced to n/2.

The following methods are followed while Booth's multiplication process [2]:

- Right Shift Circulant (RSC) – This method shifts the binary string to the one bit position right. For example: 101101, after right shift circulant the result is 110110.
- Right Shift Arithmetic (RSA) – In this method, the two binary numbers are added together and shift the result to the right 1 bit position. Example: 0101 + 0011 = 1000, now shift all bits right and put the MSB bit of the result at the beginning of the new string: 11000.

3.1 Modified Booth Multiplier (Radix-4)

Using radix-4 Modified Booth Encoding (MBE) multiplier, number of partial products can be reduced by half. MBE generates (N/2) partial products, where N is the number of bits. By means using overlapping technique, it compares three bits at a time [10]. For single precision floating point multiplication two 24-bit (1 – hidden bit, 23 – mantissa bits) numbers are to be multiplied. In this case, 24 partial products will get generated if the normal multiplication method is followed. But by using MBE, number of partial products generated can be reduced to 12. Radix-4 recoding, the most common modified booth's recoding scheme is used with the digit set {-2, -1, 0, 1, 2} and shown in the Table 2. In radix-4, grouping starts from the LSB, and the first block uses only two bits of the multiplier and assumes a reference bit 0 for the third bit [10] and shown in Figure 4.

Figure 4. Example for bit pairing as per radix-4 booth recoding

Table 2. Radix-4 MBE recoding for A * B

Bits of Multiplier B			Encoding operation on Multiplicand A	Partial Product
y_{i+1}	y_i	y_{i-1}		
0	0	0	0	M * 0
0	0	1	+B	M * 1
0	1	0	+B	M * 1
0	1	1	+2B	M * 2
1	0	0	-2B	M * -2
1	0	1	-B	M * -1
1	1	0	-B	M * -1
1	1	1	0	M * 0

IV. EXISTING SYSTEM

The existing system is implemented by Radix-4 MBE and CLA for mantissa multiplication process. Radix-4 MBE algorithm is used for partial product generation. MBE algorithm reduces the number of partial products. CLA is used to perform the addition of partial products. It increases the speed of the multiplication process due to reduction in the multiplier bits.

4.1 Carry Look-Ahead Adder

In CLA, carry signals are calculated in advance based on the input signals and therefore the reduction in the carry propagation time. To understand the working of CLA, manipulate the Boolean expression dealing with full adder. The Propagate P and Generate G in a full adder is given in the Equations (2) and (3). Here, both propagate and generate signals depends only on the input bits and therefore it will be valid only after one gate delay. The sum and carryout signals are given in the Equations (4) and (5) and these equations show that a carry signal will be generated if both bits A_i and B_i are 1 and if either A_i or B_i is 1, carry-in C_i is 1 [1]. The general expression of the carry signal C_{i+1} is given in the Equation (6).

@IJAERD-2017, All rights Reserved

$$P_i = A_i \oplus B_i \quad (2)$$

$$G_i = A_i \text{ AND } B_i \quad (3)$$

$$S_i = P_i \oplus C_{i-1} \quad (4)$$

$$C_{i+1} = G_i + (P_i \text{ AND } C_i) \quad (5)$$

$$C_{i+1} = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + \ldots\ldots P_iP_{i-1}\ldots.P_2P_1G_0 + P_iP_{i-1} \ldots.P_1P_0C_0 \quad (6)$$

Using the Equation (6), carryout C_0 - C_4 bits for the 4-bit adder can be computed. C_2, C_3 and C_4 does not depend on its previous carry-in. Therefore, carry bit C_4 need not to wait for the carry bit C_3 to propagate. The carry bit C_4 reach steady state as soon as C_0 is computed. CLA adder's structure can be divided into three parts such as propagate/generate generator, sum generator and carry generator shown in Figure 5. In practice, because of loading capacitance, it is not possible to use the CLA to realize the constant delay for the larger-bit adders and hence therefore, larger delay and power consumption. With increase in the bit size, CLA structure is more complex and results increase in area and power consumption. Speed also will drop with increase in bit size [1].

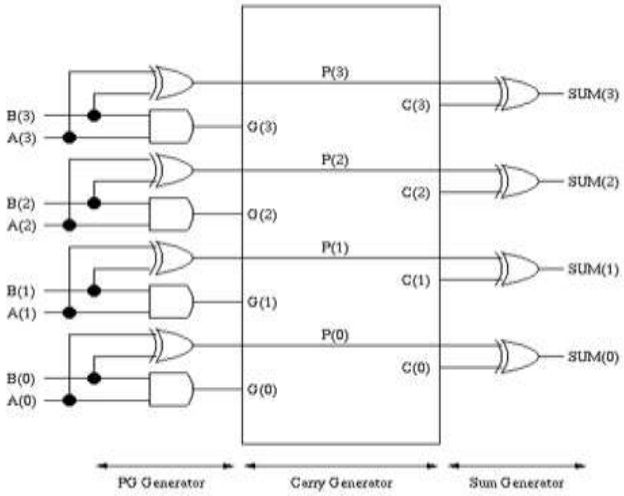

Figure 5. Block diagram of 4-bit CLA

4.2 Design Methodology

The block diagram of MBE with CLA multiplier architecture is shown in the Figure 6. In this, Booth encoder component accepts the multiplier bits as input. Booth encoder component are implemented with Radix-4 MBE algorithm to reduce the number of multiplier bits. The output of the booth encoder is the reduced multiplier bits and it is fed as input into the Partial Product (PP) generator component.

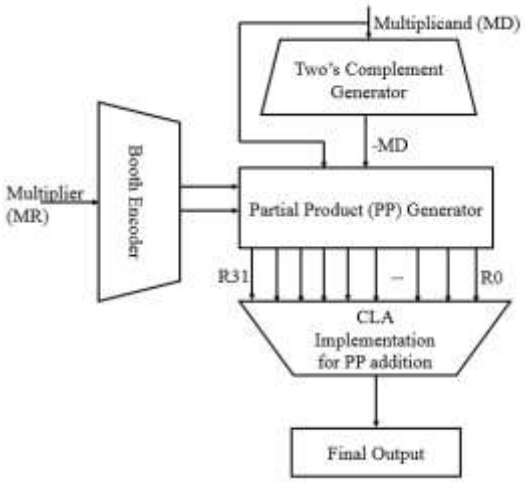

Figure 6. Block diagram of MBE with CLA multiplier

The PP generator component generates the partial product for each multiplier bits with the multiplicand bits. If the multiplicand is negative number it is converted to 2's complement form using Two's complement generator before it is given as input to the PP generator. Once PPs are generated, PPs are added using CLA implementation in the PP addition component. In this design, PPs are added one by one.

@IJAERD-2017, All rights Reserved

V. PROPOSED SYSTEM

The proposed system is implemented by Radix-4 MBE algorithm and KSA for mantissa multiplication. Radix-4 MBE follows the same design methodology as used in the existing system. KSA adder is used to perform the addition of partial products. It increases the speed of the multiplication process compared to CLA implementation. Also, it consumes less power and occupied less area than the CLA implementation.

5.1 Parallel Prefix Adder

Parallel prefix adders (PPA) also known as carry-tree adders, pre-compute propagate and generate signals. PPA can be divided into three main parts, namely the pre-processing, carry graph and post-processing. The pre-processing part will generate the propagate (p) and generate (g) bits. It is a parallel form of generating the carry bit and therefore, it performs addition operation faster. The various types of PPA are different from each other based on their carry graph and number of levels [9]. The different types of PPAs are Kogge-Stone adder, Brent-Kung adder, Knowles adder and so on. Of these, KSA adder performs faster than other PPA.

KSA is a parallel prefix form of CLA and its block diagram is shown in the Figure 7. There are three processing steps to perform operation such as pre-processing stage, carry look-ahead network stage and post-processing stage. In pre-processing stage, generate and propagate signals corresponding to each pair of input bits are generated using Equation (2) and (3). In carry generation network, uses group propagate and generate as intermediate signals which are given by the logic Equations (7) and (8). In post-processing stage, it computes the sum bits using the logic Equation (9) [9].

$$P_{i:j} = P_{i:k+1} \text{ AND } P_{k:j} \quad (7)$$

$$G_{i:j} = G_{i:k+1} \text{ OR } (P_{i:k+1} \text{ AND } G_{k:j}) \quad (8)$$

$$S_i = P_i \oplus C_{i-1} \quad (9)$$

The better performance of KSA is because of its minimum logic depth and bounded fan-out. It has a lower fan-out for each stage, which increases performance. The main advantage is it has bypassing nature. Main goal of using this adder is to reduce carry propagation so that it becomes high speed, reduce delay, time consumption will be get reduced. Delay of KSA can also be decreased by only rerouting the wires. The speed can be increased by eliminating redundant black cells which results is reduction in area of the adder [9]. Figure 8 shows the block diagram of 8-bit KSA.

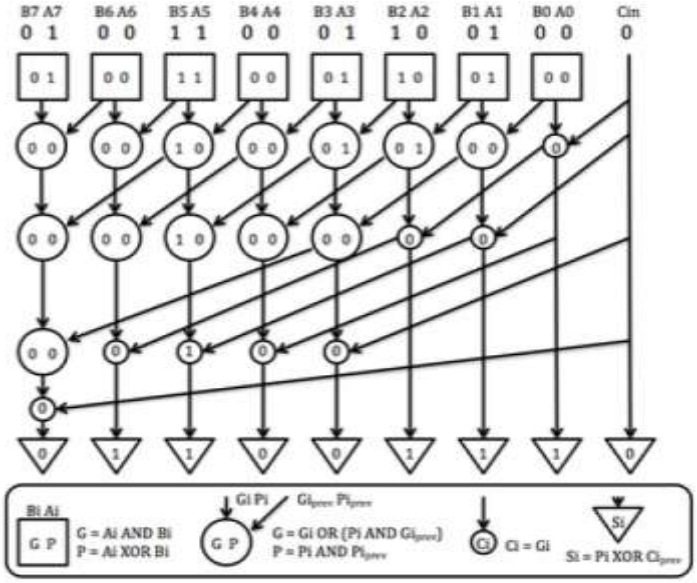

Figure 7. Block diagram of 8-bit KSA

5.2 Design Methodology

The overview of multiplication of X and Y using MBE and KSA implementation is shown in the Figure 8. X and Y inputs are 32-bit FP values. If the input is decimal number it should be converted into binary representation number and then start the process. If X or Y or both the values are 0, then an exception or an error will be generated. If Y is negative, it should be 2's complemented before it starts multiplication process. The mantissa bits (23 bits) of X and Y is multiplied in the multiplier where MBE and KSA are implemented for PP generation and PP addition. Once the mantissa part is computed, sign part is computed by taking the XOR of X and Y sign input. XOR gate is implemented using AND and OR gates. The last part need to compute is the exponent. The exponent part of the X and Y input must be biased after addition. The exponent part is implemented using Full adder circuits to add the exponents of X and Y inputs.

@IJAERD-2017, All rights Reserved

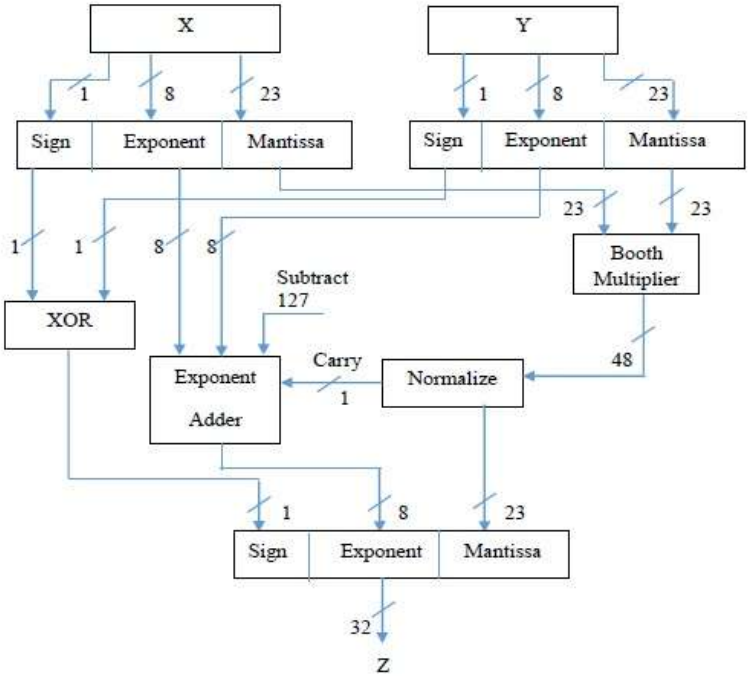

Figure 8. Overview of floating point multiplier using MBE and KSA

The block diagram of MBE with KSA multiplier architecture is shown in the Figure 9. In this, Booth encoder component accepts the multiplier bits as input. Booth encoder component are implemented with Radix-4 MBE algorithm to reduce the number of multiplier bits. The output of the booth encoder is the reduced multiplier bits and it is fed as input into the PP generator component. The PP generator component generates the partial product for each multiplier bits with the multiplicand bits. If the multiplicand is negative number it is converted to 2's complement form using Two's complement generator before it is given as input to the PP generator. Once PPs are generated, PPs are added using KSA implementation in the PP addition component. In KSA implementation, the PP addition are added parallel instead of step by step process. Due to parallel operation of KSA, computations are performed faster than the other adder methods such as CLA and so on.

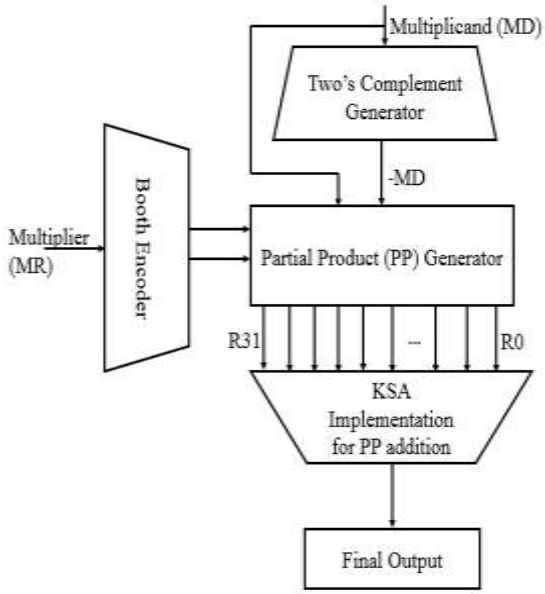

Figure 9. Block diagram of MBE with KSA multiplier

VI. SIMULATION RESULTS

VHDL coding is used for the design and simulation of this floating point multiplier [8]. Figure 10 shows the simulation result of the multiplication results of the two floating point numbers X = $(-18)_{10}$ and Y = 9.5_{10}. The binary values of X, Y and output values are,

@IJAERD-2017, All rights Reserved

X = 1 10000011 00100000000000000000000
Y = 0 10000010 00110000000000000000000
Output = 1 10000110 01010101100000000000000

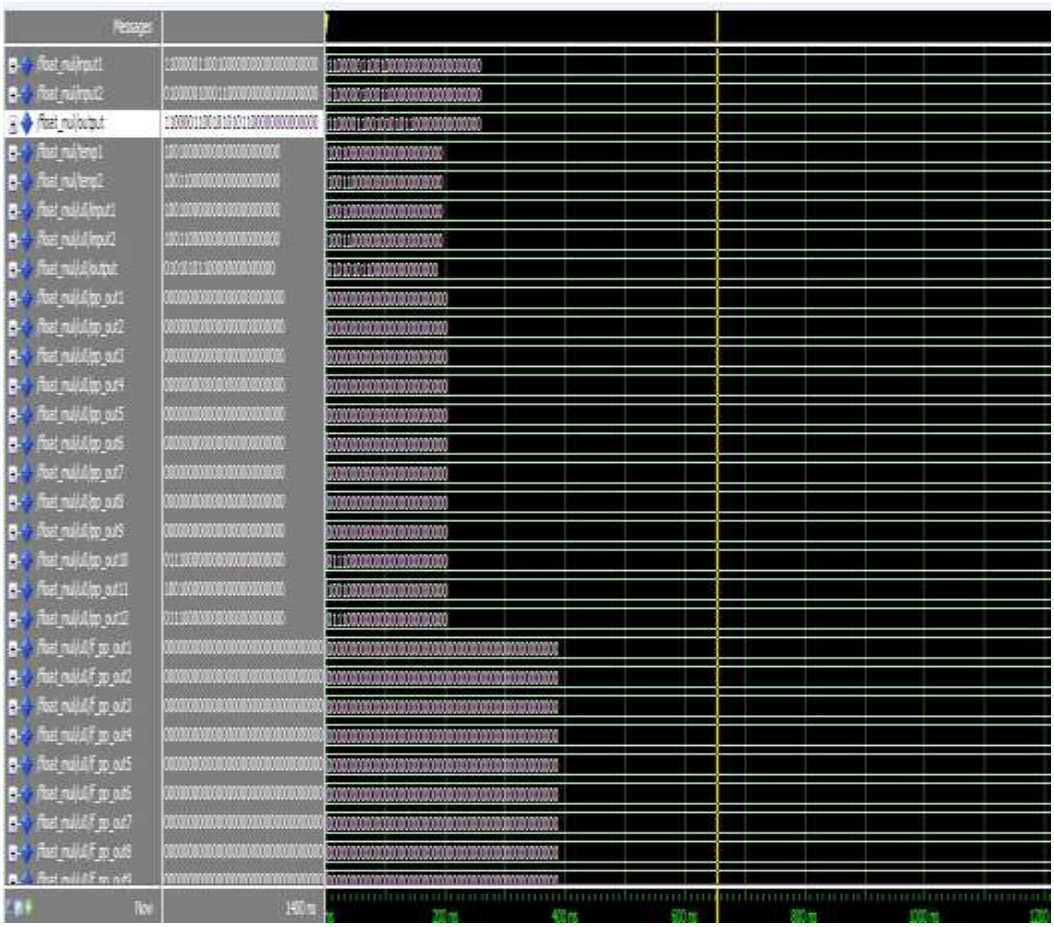

Figure 10. Simulation result of proposed multiplier

VII. COMPARISON RESULTS

Table 3 gives the comparison of the various parameters for 32-bit floating point multiplier with two different types of implementation. The area, delay and power is calculated using Xilinx FPGA Spartan 2E, FG676 – XC2S600E.

- Speed is calculated from delay where Speed = (1/Delay) [7].
- PDP is calculated from power and delay, unit is 'fJ' where f denotes 'femto'. PDP = (Power * Delay) [7].
- EDP is calculated from PDP and delay, unit is 'yJs' where y denotes 'yocto'. EDP = (PDP * Delay) [7].

Table 3. Comparison of the various parameters for 32-bit floating point multiplier with two different types of implementation

Parameters	MBE with CLA	MBE with KSA
Delay (ns)	25.564	18.02
Speed (GHz)	0.039	0.055
Area (Gate Count)	7440	7000
Power (µW)	0.320	0.230
PDP (fJ)	8.180	4.145
EDP (yJs)	209.11	74.70

VIII. CONCLUSION

In this paper, high speed, area efficient and nominal power consumption using Radix-4 Booth multiplier and KSA for 32-bit FP number was proposed. Measured results show that the delay is improved upto 30%, average power consumption is improved upto 28% and the area is reduced upto 26.33%. The result shows that the performance of the proposed multiplier results high speed, nominal power consumption and area requirement.

@IJAERD-2017, All rights Reserved

REFERENCES

[1] A. Alghamdi and F. Gebali. "Performance analysis of 64-bit carry lookahead adders using conventional and hierarchical structure styles", Pacific Rim Conference on Communications, Computers and Signal Processing. IEEE, pp. 4-5, 2015.
[2] A.D. Booth, "A Signed Binary Multiplication Technique", Qarterly J. Mechan. Appl. Math., Vol. IV, pp.100-150, 1951.
[3] J. Fadavi-Ardekani, *"M x N Booth Encoded Multiplier Generator Using optimized Wallace Trees"* IEEE trans. on VLSI Systems, Vol. 1, No.2, 1993.
[4] K. Hwang, "Computer Arithmetic: Principles, Architecture and Design", John Wiley and Sons, pp. 350–360, 1979.
[5] K. Paldurai, Dr. K. Hariharan, "FPGA Implementation of Delay Optimized Single Precision Floating point Multiplier", IEEE International Conference on Advanced Computing and Communication, pp. 5-8, 2015.
[6] R. Dhanabal, V. Bharathi, K. Shilpa, D.V. Sujana and S.K. Sahoo, "Design and Implementation of Low Power Floating Point Arithmetic Unit", International Journal of Applied Engineering Research, ISSN 0973-4562, vol. 9, no. 3, pp. 339-346, 2014.
[7] S.V. Saravanan, R. Senthil Ganesh, "Design of Low Power and Area Efficient Dynamic Comparator", International Journal of Printing, Packaging & Allied Sciences, vol.04, no.02, pp.1464-1470, 2016.
[8] Rajit Ram Singh, Asish Tiwari, Vinay Kumar Singh, Geetam S Tomar, "VHDL environment for floating point Arithmetic Logic Unit -ALU design and simulation", International Conference on Communication Systems and Network Technologies, pp. 4-9, 2011
[9] Zahi Moudallal, Ibrahim Issa, Moussa Mansour, Ali Chehab, and Ayman Kayssi. "A low-power methodology for configurable wide Kogge-Stone adders", Energy Aware Computing (ICEAC), IEEE, pp. 1-5, 2011.
[10] R. Kalaimathi, "A Survey on Area Efficient Low Power High Speed Multipliers", International Journal for Research in Applied Science & Engineering Technology, Vol.05, Issue XI, pp. 1-4, 2017.

@IJAERD-2017, All rights Reserved