TEST Engineering & Management

January - February 2020
ISSN: 0193 - 4120 Page No. 9830 - 9839

Design and Performance Analysis of Single Precision Floating Point Multiplier Using Parallel Prefix Adders

Dr. R.Senthil Ganesh[1], Dr. S.A. Sivakumar[2], Dr.R.Naveen[3], Dr. B.Maruthi Shankar [4]
[1]Associate Professor-ECE, P.S.R. Engineering College, Sivakasi, Tamilnadu, India.
[2]Professor & Head-ECE, Dr.K.V.Subba Reddy College of Engineering for Women, Kurnool, AP, India.
[3]Professor & Principal, Dr.K.V.Subba Reddy College of Engineering for Women, Kurnool, AP, India.
[4]Associate Professor-ECE, Sri Krishna College of Engineering and Technology, Coimbatore, Tamilnadu, India.
[1]senthilganeshgg@gmail.com, [2]drsasivakumar@gmail.com, [3]drnaveenraman@gmail.com, [4]maruthishankar@gmail.com,

Article Info
Volume 82
Page Number: 9830 - 9839
Publication Issue:
January-February 2020

Article History
Article Received*: 18 May 2019*
Revised*: 14 July 2019*
Accepted*: 22 December 2019*
Publication*: 14 February 2020*

Abstract
This paper presents single precision floating point (32-bit) multiplier design using Parallel Prefix algorithm and Radix-4 Booth algorithm. Parallel prefix adders such as Kogge-Stone and Han-Carlson adders are implemented by parallel prefix algorithm is used to perform the partial product addition in the multiplication operation and this adder is also used in the exponent addition in the multiplier design. Radix-4 Booth algorithm is used to reduce the multiplier bits so that the number of partial products generation can be reduced significantly. The simulation results of single precision floating point multiplier designed using Kogge-Stone and Han-Carlson adder implementations are compared. The multiplier is designed using Tanner EDA 13.0 tool in 130nm CMOS technology.

Keywords; *Parallel Prefix Algorithm, Radix-4Booth Algorithm, Single Precision Floating Point Multiplier, Adders*

I. INTRODUCTION

Arithmetic operations used in DSP algorithms and other applications can be implemented with many different techniques. These different techniques differ with respect to their implementation cost such as space, time and power. The data involved can be either bit-serial or bit-parallel are processed. Generally, a number can have an integer and a fractional part. This type of number systems are classified into two groups such as fixed-point and floating point number system. The arithmetic operations performed on these type of number systems are called as fixed-point and floating point arithmetic. In fractional fixed point arithmetic, the numbers may be considered in the range $1 < x < 1$. Hence, the product will be in the same range. The word length is reduced by keeping the left and dropping the right hand part. This would result in a small error in the quantized product. Conversely, a floating point arithmetic is described by a signed mantissa and a signed exponent, ex. $\pm m\ 2^{\pm exp}$. The magnitude of the mantissa is usually normalized to the interval [0.5, 1]. Floating point arithmetic is used in general purpose computers to handle values that vary over a large range. In most DSP algorithms, however, the necessary signal range can be made small by appropriate scaling of the signal levels [5].

Three implementation styles for addition and multiplication are frequently encountered in many DSP and other applications such as bit-serial, bit parallel and digit serial. Bit-serial systems process 1-bit of the input sample every clock cycle. These systems can be synthesized using integer linear programming based scheduling approach. Bit-parallel systems process one whole word of the input sample each clock cycle and ideal for high speed applications. Digit-serial systems process multiple number of bits every clock cycle and are best suited for applications requiring moderate sample rate, where area and power consumptions are critical. Bit-serial arithmetic is used for the implementation of data-flow algorithms of medium complexity and low to medium data rate,

January - February 2020
ISSN: 0193 - 4120 Page No. 9830 - 9839

whereas Bit-parallel arithmetic operators are used for the implementation of data-flow algorithms of low complexity and high data rate [5].

Multiplication involves two basic operations are the generation of partial products and their accumulation. Consequently there are two ways to speed up the multiplication process such as reduce the number of partial products or accelerate their accumulation of the partial products. To reduce the number of partial products, a straight forward approach is to examine 2 or more bits of the multiplier at a time. The reduction in the number of partial products can reduce the latency of the multiplication operation. However, this requires the generation of multiples 1A, 2A, 3A, etc., where A is the multiplicand [5]. To accelerate the partial products addition, high speed adders such as tree adders, etc., are used. Parallel Prefix Adders (PPA) are suitable for efficient and performance oriented designs. PPA are also known as logarithmic adders because the delay of these adders are lesser of the order of [log N] for an N-bit adder. These adders are less complex, regular structures as well as faster in operations and also it takes lesser number of steps to carry calculation. The various types of PPA are available such as Kogge-Stone, Brent-Kung, Knowles, Sklansky, Han-Carlson [1]. In this paper, we have presented the single precision floating point multiplier implementation using Booth algorithm which reduces the number of partial products and PPAs for faster addition process. The results between the implementation of Kogge-Stone and Han-Carlson adders are compared. The single precision floating point representation follows the standard Institute of Electrical and Electronics Engineers (IEEE) format.

II. IEEE REPRESENTATION

An IEEE standard for Floating-Point arithmetic (IEEE 754) is established in 1985. This standard addresses many problems that found in the diverse floating point implementations that made them difficult to use portably and reliably. Most of the hardware floating point arithmetic units uses the standard IEEE 754 format. This format is a set of representations of symbols and numerical values. The IEEE format comprises of [6],

- Finite numbers, which may be either base 2 (binary) or base 10 (decimal). The numerical value of a finite number is represented in the equation (1). In equation (1),'s' is a sign value (zero or one), 'c' is a significand value (or mantissa), 'q' is an exponent value, b is a base (or radix).

$$(-1)^s \times c \times b^q \quad (1)$$

- Two infinities are $+\infty$ and $-\infty$.
- Two types of NaN are quiet NaN (qNaN) and signaling NaN (sNaN). The sign of a NaN has no meaning, but it may be predictable in some circumstances.

The IEEE 754 format for single precision floating point number of in binary representation is given in the equation (2) and the decimal representation is given in the equation (3). Figure 1 shows the single precision floating point number representation in IEEE 754 format. Sign bit represents the sign of the number and the sign of the mantissa also. Exponent value is either an 8-bit unsigned integer (0 to 255) or an 8-bit signed integer (-128 to 127). If the unsigned integer format is used, the exponent value used in the arithmetic is the exponent shifted by a bias value 127 which means for the IEEE 754 binary representation of 32 bits, an exponent value of 127 is actual zero. The exponent value +127 (i.e. all 0s) and +128 (i.e. all 1s) are used to indicate special numbers. The binary numbers to the right of decimal point is 23 bits which is referred as mantissa bits and the value '1' is referred as an implicit leading bit, unless the exponent is stored with all zeros[6].

The representation of IEEE 754 in binary format is [6],

$$(-1)^{b31} \text{x } 2^{(b30b29....b23)-127} \text{ x } (1.b_{22}b_{21}...b_0) \quad (2)$$

The representation of IEEE 754 in decimal format is [10],

$$(-1)^{sign} \text{ x } 2^{(e-127)} \text{ x } (1.\sum_{i=1}^{23}(b_{23-i}\, 2^{-i}) \quad) \quad (3)$$

January - February 2020
ISSN: 0193 - 4120 Page No. 9830 - 9839

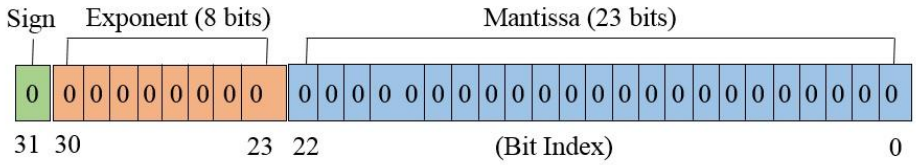


Figure 1. Single Precision Floating Point IEEE Format (32 bits)

III. BOOTH MULTIPLIER

Booth's algorithm has the advantage of reducing the number of partial products and therefore it is widely used in the design of hardware or software multipliers. It can be used for both signed-magnitude numbers and 2's complement numbers with no need for a correction term. In the modified Booth recoding algorithm, the signed digit set {-2, -1, 0, +1, +2} is used and therefore it is termed as 5-level Booth recoding [7]. In Booth recoding algorithm, first multiplier bits are recoded into signed digit representation, then the partial products are generated using the recoded multiplier digits and finally it is accumulated. Booth recoded multiplier consists of three parts, the recoding circuitry for multiplier bits, the partial product generation and accumulation. The multiplier bits are represented as X_{i+1}, X_i are recoded into signed digit Z_i with X_{i-1} serving as a reference bit. For an example, 8-bit number can be recoded are shown in the Figure 2 [5].

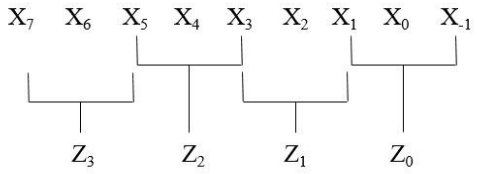


Figure 2. Example Bit Pair Recoding of Modified Booth Algorithm

Where X_{-1}= 0 is appended after LSB of the multiplier bits as a reference bit. The modified Booth recoding algorithm to generate $Z_i$from X_{i+1}, X_i and X_{i-1} is given in the Table 1. The value of $Z_i$is calculated using the general equation (4) [5].

$$Z_i = X_i + X_{i-1} - 2X_{i+1} \quad (4)$$

Booth Multiplier can be used in three distinct modes such as radix-2, radix-4,radix-8. Radix-4 Booth's algorithm is most widely because the number of partial products generation is reduced to n/2 where 'n' is the number of multiplier bits. The Booth algorithm is as follows [7],

- Append zero with the LSB.
- If number of multiplier bits are even, then append the MSB with two zero or elseappend with one zero.
- Divide the multiplier bits into overlapping group of 3-bits.
- Determine the partial products scale factor from the recoding table.
- Compute the multiplicand with the determined scale factor and thus results the partial product.

The value of Z_i(i.e. digit sets)can also be represented in the form of binary values and it is determined from the logic equations (5), (6) and (7) respectively. The Booth multiplier circuit is designed based on these binary value representation instead of digit sets representation. The binary values are represented as M, Y and Y2and the corresponding values are given in the Table 1 [10].

$$M = X_{i+1} \quad (5)$$

$$Y = X_i \oplus X_{i-1} \quad (6)$$

$$Y2 = (X_{i+1} \bullet X_i \bullet X_{i-1}) + (X_{i+1} \bullet X_i \bullet \overline{X_{i-1}}) \quad (7)$$

Table 1. Radix-4 MBE Recoding for A x B

Bits of Multiplier B			Z_i	M	Y	Y2	Partial Product
X_{i+1}	X_i	X_{i-1}					
0	0	0	0	0	0	0	A * 0
0	0	1	+1	0	1	0	A * 1

TEST Engineering & Management

January - February 2020
ISSN: 0193 - 4120 Page No. 9830 - 9839

0	1	0	+1	0	1	0	A * 1
0	1	1	+2	0	0	1	A * 2
1	0	0	-2	1	0	1	A * -2
1	0	1	-1	1	1	0	A * -1
1	1	0	-1	1	1	0	A * -1
1	1	1	-0	1	0	0	A * 0

IV. PARALLEL PREFIX ADDERS

For given 'N' inputs $\{X_N, ….., X_1\}$, Parallel prefix circuit computes N outputs $\{Y_N, …,.Y_1\}$using an arbitrary associative operator 'o' and explained as [4],

$Y_1 = X_1$, $Y_2 = X_2 o X_1$, $Y_3 = X_3 o X_2 o X_1$, ……, $Y_N = X_N o X_{N-1} o …. o X_2 o X_1$ (8)

The common prefix computations include addition, incrementation, priority encoding, etc. Most prefix computations pre-compute intermediate variables $\{Z_{N:N}, …., Z_{1:1}\}$ from the inputs. The prefix network combines these intermediate variables to form the prefixes $\{Z_{N:1}, …, Z_{1:1}\}$. The outputs are post-computed from the inputs and prefixes [4]. For example, adders take inputs $\{A_N, …, A_1\}$, $\{B_N, …, B_1\}$ and C_{in} produces the sum output $\{S_N, …, S_1\}$ using intermediate generate (G) and propagate (G) prefix signals [4]. The parallel prefix addition follows the three stages [8],

- Pre-processing Stage:

$$G_{[i:i]} = A_i \bullet B_i, \qquad G_{[0:0]} = C_{in} \tag{9}$$

$$P_{[i:i]} = A_i \oplus B_i, \quad P_{[0:0]} = 0 \tag{10}$$

- Prefix-computation:

$$G_{[i:j]} = \begin{cases} G_{[i:i]} & \text{if } i = j \\ G_{[i:k]} + P_{[i:k]}\, G_{[k-1:j]} & \text{otherwise} \end{cases} \tag{11}$$

$$P_{[i:j]} = \begin{cases} P_{[i:i]} & \text{if } i = j \\ P_{[i:k]}\, P_{[k-1:j]} & \text{otherwise} \end{cases} \tag{12}$$

- Post-processing:

$$C_i = G_{[i:0]}$$

$$S_i = P_i \oplus G_{[i-1:0]} \tag{13}$$

There are different ways to perform prefix computation such as serial-prefix structures like ripple carry adder have a latency of O(N) where 'N' is the number of bits. Single level carry look ahead structure reduce the latency by a constant factor. Parallel prefix structure uses a tree structure which reduces the latency to O(log N). The classic parallel prefix structures are Kogge-Stone, Brent-Kung and Sklansky. The hybrid structures are Han-Carlson (Kogge-Stone and Brent-Kung), Knowles (Kogge-Stone and Sklansky) and Ladner-Fischer (Sklansky and Brent-Kung). An ideal prefix structure have $\log_2 N$ stages, a fanout never exceeding 2 at each stage and no more than one horizontal track of wire at each stage. The classic architecture of prefix structure deviates from the ideal architecture with $2\log_2 N$ stages, fan-out of [N/2 +1] and N/2 horizontal tracks [4].

TEST Engineering & Management

January - February 2020
ISSN: 0193 - 4120 Page No. 9830 - 9839

The conditional sum addition logic for prefix addition proposed by Sklansky (1960) offers a minimum depth prefix network at the cost of increased fan-out for certain computational nodes. In 1973, Kogge and Stone proposed an algorithm that has both low fan-out and optimal depth but the circuit design is complex with the large number of interconnects. The algorithm proposed by Brent and Kung (1982) uses less computational nodes but possess maximal depth which accounts for increased latency. In 1980 Ladner and Fischer proposed a method that reduces the maximum fan-out for computational nodes in the critical path with slightly higher depth compared to Sklansky structure. Han and Carlson (1987) proposed the structure that combines both Kogge-Stone and Brent-Kung topology offers trade-off between logic-depth, interconnect count and the number of computational nodes. Knowles (2001) presented a class of logarithmic adders with minimum depth by allowing the fan-out to grow [9].

The prefix operator has two essential properties namely associative property and idempotent property which allows for greater parallelism. The associative property is explained in the equation (14) and idempotent property is explained in the equation (15) [1].

$$(G, P)_{[h:j]} o\ (G, P)_{[j:k]} = (G, P)_{[h:i]} o\ (G, P)_{[i:k]} \quad (14)$$

$$(G, P)_{[h:j]} o\ (G, P)_{[i:k]} = (G, P)_{[h:k]} \quad (15)$$

where $h > i \geq j \geq k$.

In this paper, valency-2 prefix computation is used which means that it uses 2-input associative operators. The parallel prefix structures are distinguished by the arrangement of prefix cells. For multiplier design, Han-Carlson Adder (HCA) and Kogge-Stone Adder (KSA) prefix structures are used and the results are compared. In Figure 5, the upper box performs the pre-processing and lower box performs the post-processing operation which is discussed in the equation (9), (10) and (13). The middle stage of the prefix network comprises of black cells, grey cells and buffers. The black cell performs the full prefix operations as discussed in the equation (11) and (12) whereas the grey cell performs the prefix operation of equation (11) only. The buffers are used to reduce the loading effect on the critical path. The logic circuit of the black and grey cells are shown in the Figure 3 and 4 respectively.In Figure 3 and 4, 'k-1' is represented as letter 'l'.

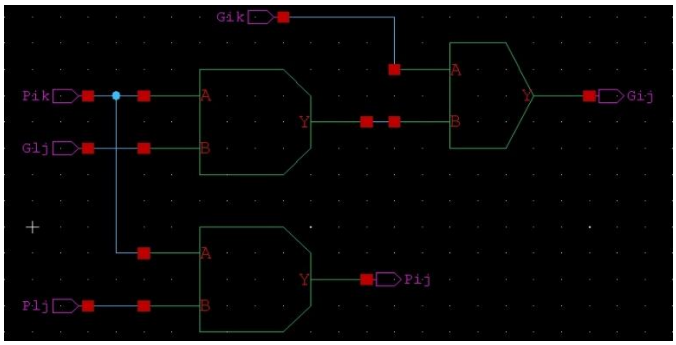


Figure 3. Logic Circuit of Black Cell

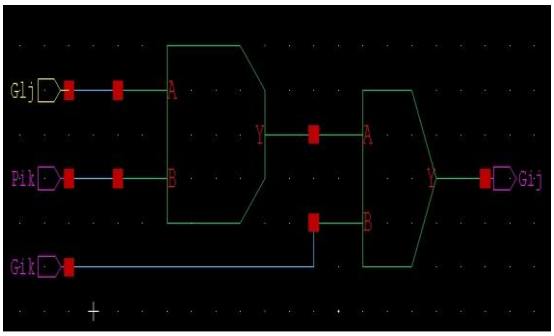


Figure 4. Logic Circuit of Grey Cell

A. KOGGE-STONE ADDER

Figure 5 shows the prefix structure of 8-bit and 24-bit KSA. The 8-bit KSA[8] is used for exponent addition and 24-bit KSA [3] is used for partial products addition in the multiplier design. This prefix structure is suitable for high speed applications, but with the cost of area and power. The delay of this structure is $\log_2(N)$ and the computational nodes are $[N (\log_2 N) - N + 1]$[8]. This structure resolves the fan-out problem by recursive doubling algorithm. To limit the lateral fan-out, idempotency property is used. But it increases the number of lateral wires at each stage. The reason is, there is a massive overlap between the prefix sub-terms being pre-computed.

January - February 2020
ISSN: 0193 - 4120 Page No. 9830 - 9839

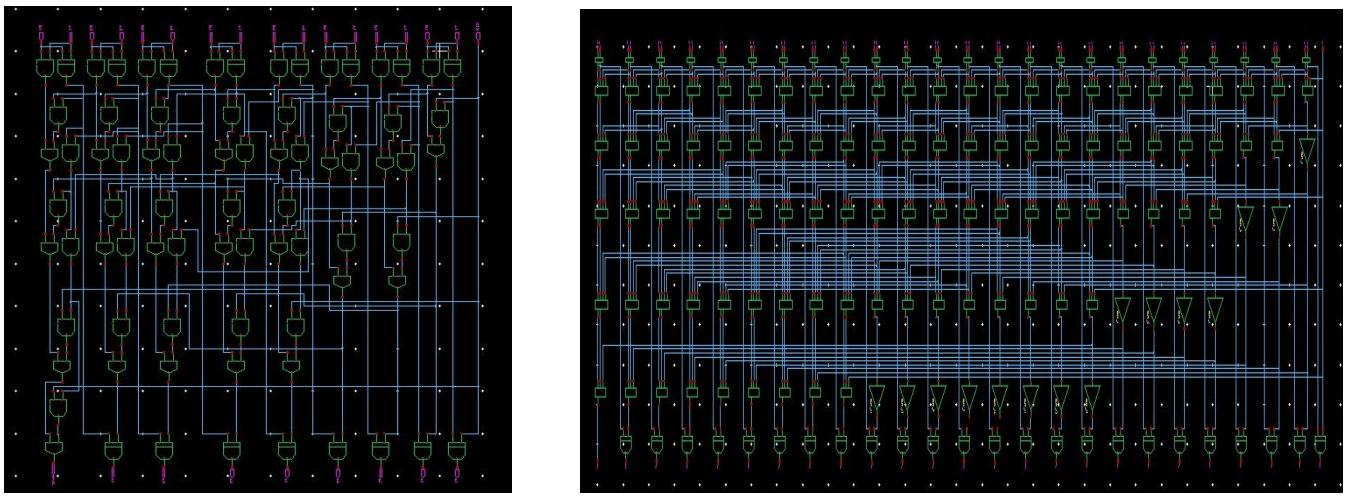


Figure 5. Prefix Structure of 8-bit and 24-bit KSA

B. HAN-CARLSON ADDER

It is a hybrid design of Brent-Kung and Kogge-Stone. Figure 6 shows the prefix structure of 8-bit [8] and 24-bit HCA [2]. The first stage is the pre-processing stage, the second stage resembles Brent-Kung and the middle stages resemble kogge-Stone. It possess wires with shorter span than KSA. The black and grey cells are placed at the odd bit positions in the initial stages. In the final stage of prefix computation the grey cells are placed at the even bit positions. The delay of HCA is given by [$\log_2 N + 1$] and the computation hardware complexity is [(N/2) $\log_2 N$][8]. When compared to KSA, the hardware complexity is greatly reduced with the cost of an additional stage to its carry merge path. Similar to KSA, the 8-bit HCA is used in the exponent addition and 24-bit is used in the partial products addition. For example, say there are three rows of partial products which is 23-bit. First two rows of partial products are added using 24-bit HCA. From the result of the addition, last two LSB values are taken as final product value. The remaining 22-bits are added with the last row of the partial products using the same 24-bit HCA which gives final product appended with previously computed two LSBs.

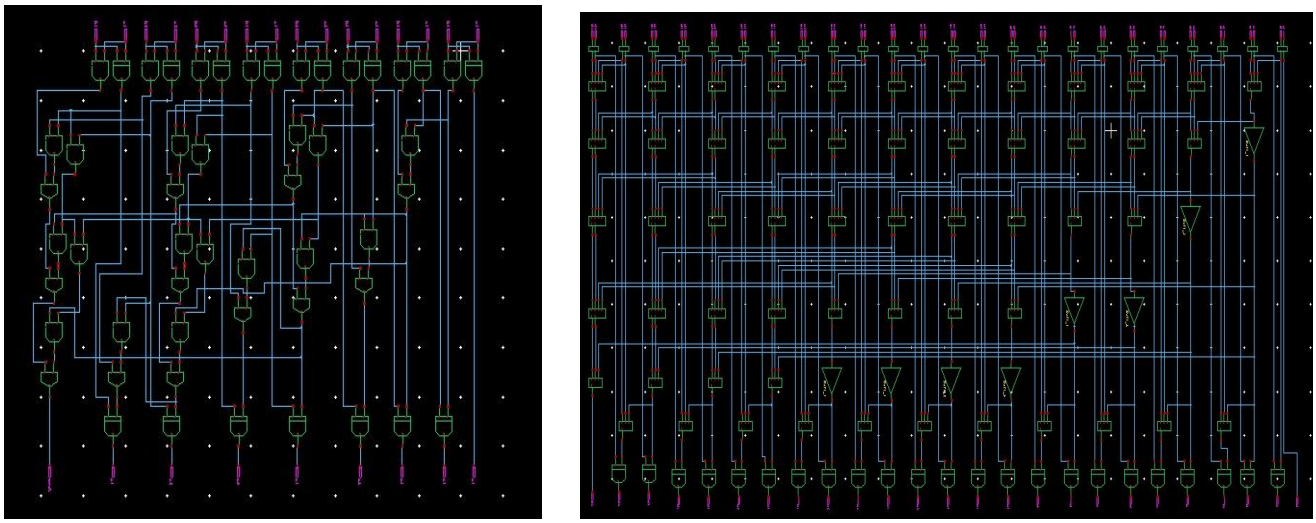


Figure 6. Prefix Structure of 8-bit and 24-bit HCA

V. DESIGN METHODOLOGY

Figure 7shows the overview of the single precision floating point multiplier using MBE and PPA.It consists of the components such as Input component where it receives two 32-bit single precision floating point format as inputs, Bias components where it computes the exponent part

January - February 2020
ISSN: 0193 - 4120 Page No. 9830 - 9839

of the input values, Booth multiplier component where the multiplicationof mantissa part takes place and the last one is the Normalize component where the normalization of the multiplication value takes place if it is not in normalized format. The sign value is computed by taking XOR of MSB of the given two inputs. The multiplication procedure is shown in the Figure 8 [10] and the block diagram of Booth multiplier part is shown in the Figure 9 [10].

The 23-bit booth multiplier component for 23-bit mantissa multiplication consists of 3-bit Radix-4 Booth encoder component to encode the multiplier bits, 23-bit partial product generator component and 24-bit PPA component for partial product addition. Booth Encoder logic circuit and 23-bit partial product generator logic circuits are shown in the Figure 10 and 12 respectively. The logic gate of 1-bit partial product generator circuit is derived from the logic equation (16) and shown in the Figure 11. In equation (16), X_j and X_{j-1} is consecutive multiplicand bits, Y, Y2 and M are encoded values of multiplier bits. In Figure 11, X_{-1}, X_0 represents consecutive bits of the multiplicand and M, Y, Y2 represents the encoded values of the multiplier bits. Half Adder (HA) is used to compute the Partial Product (PP) for the corresponding multiplicand bit with the respective encoded value. For the first partial product (LSB value) $C_{IN}$is considered as 'M' value and for the subsequent partial product generation, C_{IN} value is taken from the previous C_{out}. The 24-bit PPA component have two different implementations for HCA and KSA.

$$PP_{ij} = [(X_j \bullet Y \bullet Y2) + (\overline{X_{j-1}} \bullet Y \bullet \overline{Y2})] \oplus M \quad (16)$$

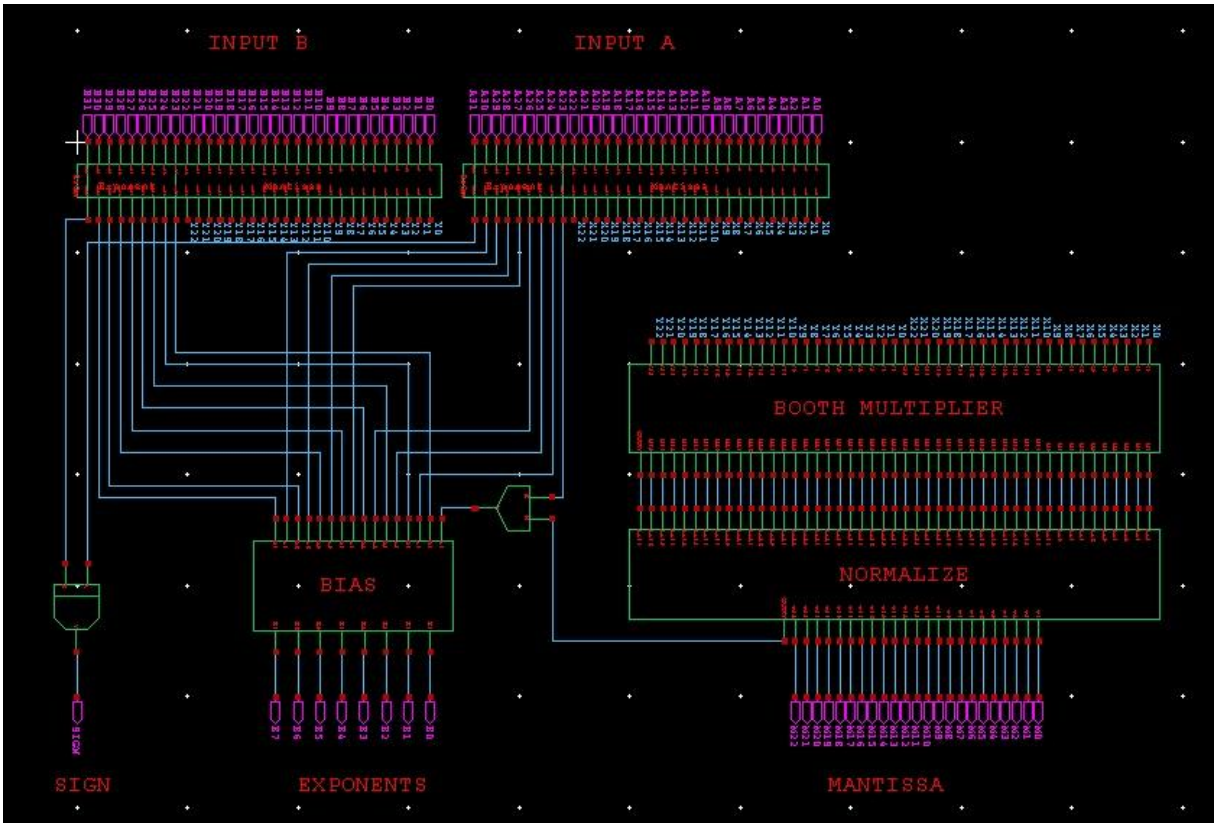


Figure 7. Design of Single Precision Floating Point Multiplier using MBE and PPA

January - February 2020
ISSN: 0193 - 4120 Page No. 9830 - 9839

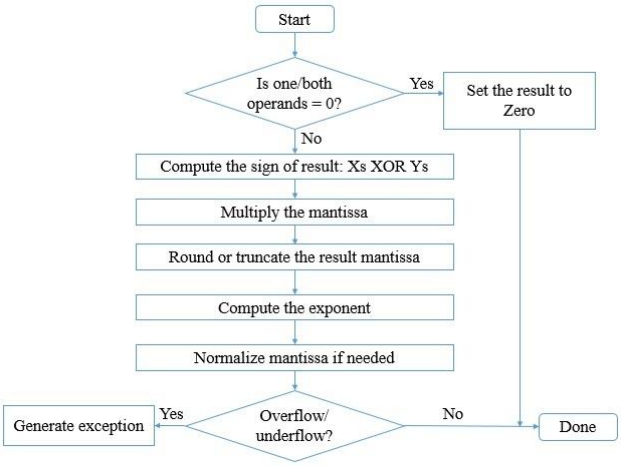


Figure 8. Mantissa Multiplication Flowchart

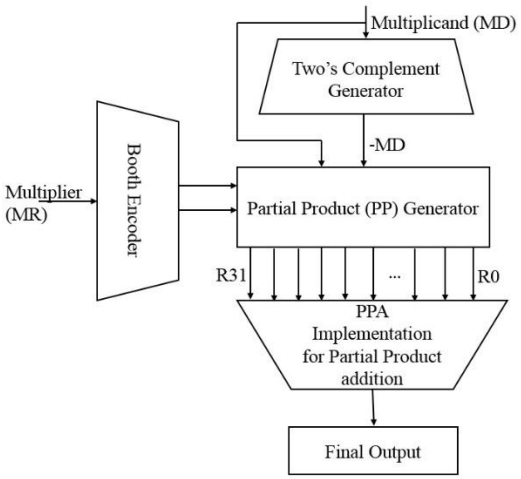


Figure 9. Block diagram of Booth Multiplier

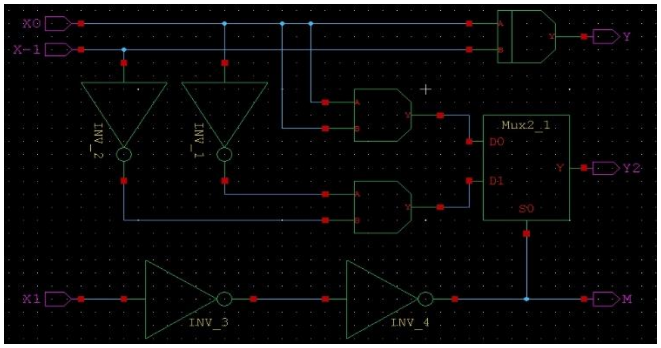


Figure 10. Booth Encoder Logic Circuit

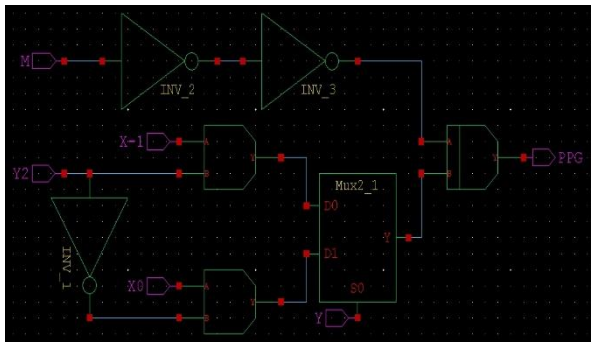


Figure 11. 1-bit Partial Product Generator

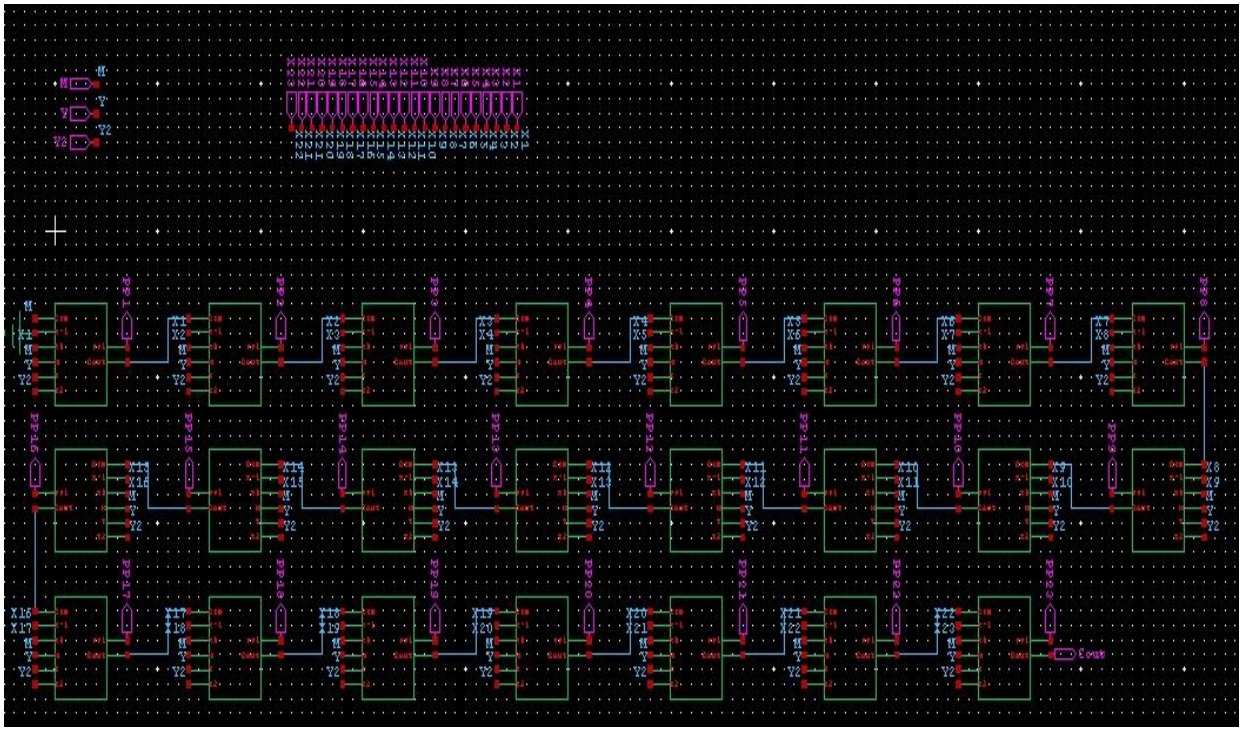


Figure 12. 23-bit Partial Product Generator Circuit

January - February 2020
ISSN: 0193 - 4120 Page No. 9830 - 9839

VI. SIMULATION RESULTS AND COMPARISON

Figure 13 shows the simulation result for A=(-18) and B=(9.5) and the corresponding binary values are A = $(11000001100100000000000000000000)_2$ and B = $(01000001000110000000000000000000)_2$. Figure 14 shows the simulation result for A=(-395.25) and B=(-200.567) and the binary representations are A = $(11000011110001011010000000000000)_2$ and B = $(11000011010010001001000100100111)_2$.

Table 2 shows the comparison of parameters between KSA and HCA implementation of multiplier design. The comparison chart for each parameter is shown in theFigure 15.

- PDP is calculated from power and delay, unit is 'fJ' where f denotes "femto" (10^{-15}). PDP = (Power * Delay) [8].
- EDP is calculated from PDP and delay, unit is 'yJs' where y denotes "yocto" (10^{-24}). EDP = (PDP * Delay) [8].

Table 2. Comparison of various parameters between KSA and HCA implementation

Parameters	MBE with KSA	MBE with HCA
Power (µW)	17.842	12.368
Delay (ns)	22.73	18.01
Transistor Count	38730	30886
PDP (fJ)	405.55	222.61
EDP (yJs)	9.218	2.751

Figure 13. Simulation Result 1

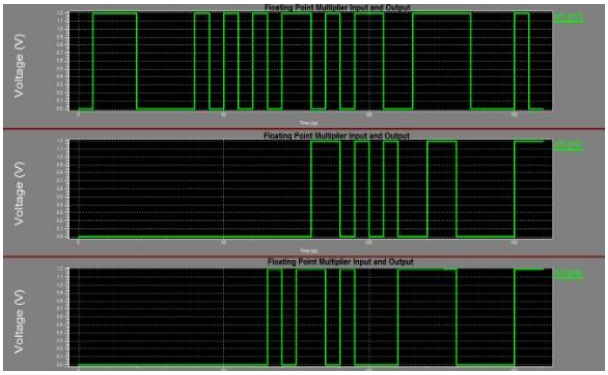
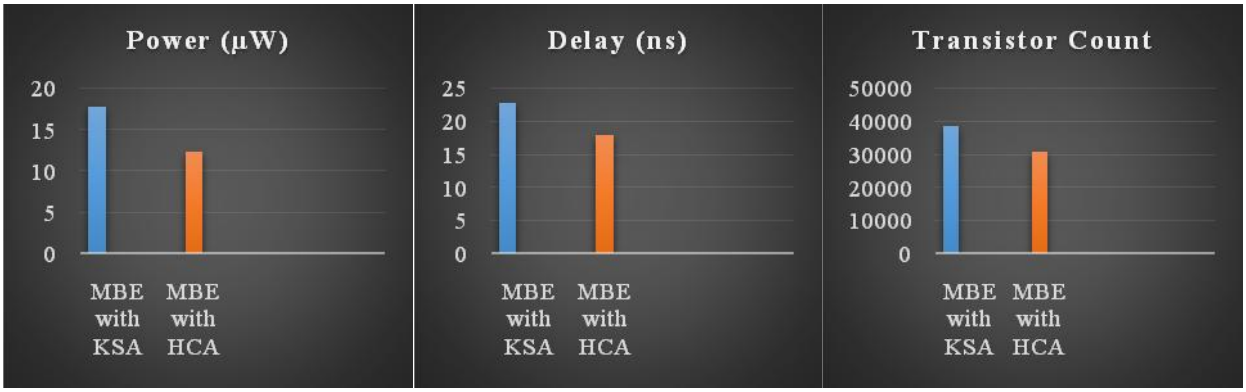


Figure 14. Simulation Result 2



January - February 2020
ISSN: 0193 - 4120 Page No. 9830 - 9839

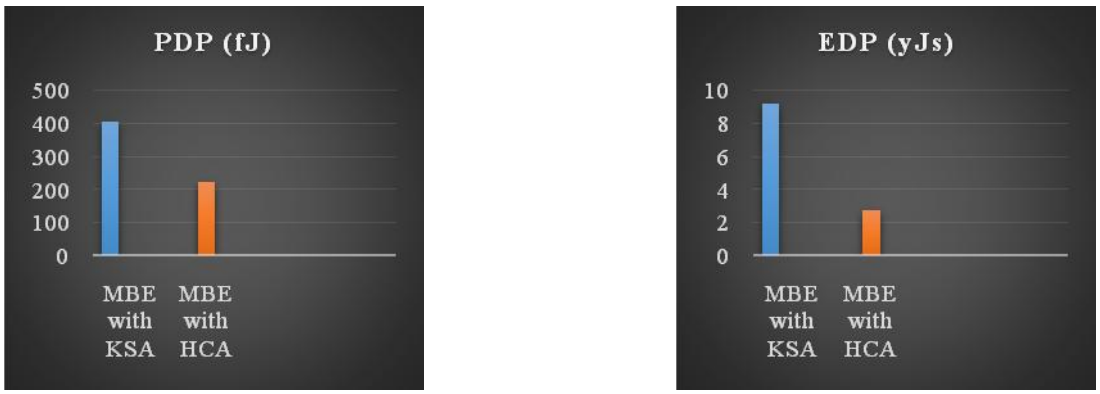


Figure 15. Comparison chart for various parameters

VII. CONCLUSION

In this paper, Single precision floating point multiplier using Booth and PPA are designed and the performance analysis between the KSA and HCA implementations are shown in the comparsion table. It is shown that, thedelay is improved upto 21%, average power consumption is improved upto 30% and the area is reduced upto 20% in HCA implementation than KSA implementation. The result shows that the performance of the proposed multiplier results high speed, nominal power consumption and area requirement.

REFERENCES

[1] A. Beaumont-Smith and C. Lim, "Parallel Prefix Adder Design",*Proc. I5Ih IEEE Symp. Comp. Arith.*, pp. 218-225, Jun 2001.

[2] T. Han and D. Carlson, "Fast Area-Efficient VLSI adders",*Proc. Symp. Comp. Arith.*, pp. 49-56, Sep. 1987.

[3] P. Kogge and H. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations",IEEE Trans. Computers,Vol. C- 22, no. *8*, pp. 786-793, Aug. 1973.

[4] D. Harris. "A Taxonomy of Parallel Prefix Networks". Thirty-Seventh Asilomar Conference on Signals, Systems and Computers, 2003. Vol. 2, pp. 2213 – 2217, Nov. 2003.

[5] A.D. Booth, "A Signed Binary Multiplication Technique", Qarterly J. Mechan. Appl. Math., Vol. IV, pp.100-150,1951.

[6] IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, 2008.

[7]R. Kalaimathi, R. Senthil Ganesh, "A Survey on Area Efficient Low Power High Speed Multipliers", International Journal for Research in Applied Science and Engineering Technology, Vol.5, Issue XI, pp. 2932-2937, Nov 2017.

[8] R. Senthil Ganesh, R. Kalaimathi, "Design and Analysis of Kogge-Stone and Han-Carlson Adders in 130nm CMOS Technology", International Journal of Research (IJR), Vol.5, pp. 1063-1068, Mar 2018.

[9] Y. Choi, "Parallel Prefix Adder Design with Matrix Representation", Proc. 17th IEEE Symposium on Computer Arithmetic, pp 90-98, Jun 2005.

[10] R. Kalaimathi, R. Senthil Ganesh, "Design of High Speed, Low Power and Area Efficient 32-bit Floating Point Multiplier", International Journal of Advanced Engineering and Research Development, Vol.4, Issue XI, pp. , Nov 2017.

.