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

Floating-point butterfly architecture based on redundant number system and Fused-Dot-Product-Add unit

[1]Ch.Jaya Prakash, [2]Kondamuri Raji, [3]Dr PHS Tejo murthy

[1]Assistant Professor, [2]PG scholar, [3]Professor
[1,2,3]Department of ECE, Sir C.R.Reddy college of engineering, Andhra University, Andhrapradesh, India.

Abstract —- *The Fast Fourier transforms (FFT) plays a major role in ruling the performance of many communication systems. The butterfly unit is the major building block of FFT architectures which mainly consists of addition and multiplication operations over complex numbers. The number representation plays an important role in enhancing the speed of any digital system. The strings of digits can be represented using floating-point (FP) arithmetic or fixed-point to design butterfly unit. The FP arithmetic provides a wide dynamic range by reducing the design constraints like scaling and overflow/underflow, but it has low performance issues in terms of speed when compared to fixed-point representation. The fused-dot-product-add (FDPA) unit based on binary sign digit representation (BSD) is used to increase the speed of FP butterfly architecture; it is used to compute AB±CD±E. The FDPA unit consists of FP BSD multiplier and FP BSD three-operand-adder units that use a BSD carry-limit adder. FP three-operand adder using comparator is proposed to reduce the area and delay of the FDPA unit and a new BSD adder is further proposed to increase the speed of the existing architectures. The simulations of the architectures in this paper are done by using Xilinx software for efficient results.*

Keywords-: *butterfly unit, fixed-point, floating-point, binary sign digit representation, redundant number system.*

I. INTRODUCTION

The high speed processors are required to enhance the speed of the communication systems. The FFT co-processor is used in real-time multimedia services. It mainly consists of add and multiply operations over complex number. The format used to store the information can be either fixed point representation or floating point representation. The fixed point representation is most widely used in the design of the FFT architectures because of its high speed compared to FP; but in applications, such as banking and finance where precision is an important factor we use FP FFT architectures. The FP arithmetic helps the designer freeing from design concerns like scaling and overflow/underflow while providing wide dynamic range. The IEEE-758 format [2] specifies the FP arithmetic in computer programming environments. The floating-point number is represented as
$(-1)^{\text{sign}}$ x significand x $(\text{base})^{\text{base}}$

To increase the speed, reduce area and power we need to fuse several operations into single floating point unit. In this paper we use fused dot product add (FDPA) unit, to compute AB±CD±E using redundant representation. The redundant number system is used to eliminate intermediate carry propagation, so that many arithmetic operations are performed prior to the final result. The conversion from redundant to non-redundant requires carry propagation. The proposed FDPA unit computes AB±CD±E. The techniques that are used to improve the performance of the proposed design are

1) Design of Redundant FP BSD multiplier.
2) Design of Redundant FP BSD three-operand adder.
3) Design of FDPA unit.

II. FFT architectures

The equation for N-point FFT is given below

$$X[n] = \sum_{k=0}^{N-1} x(k) .W_N^{nk} \quad \ldots\ldots\ldots\ldots(1)$$

Where n = 0, 1, 2 …N-1, x (k) is input, W_N is the complex twiddle factor; where $W_N=e^{-2\Pi i/N}$. To simplify the hardware realization the equation 1 is decomposed into even and odd indices, this decomposition further leads to butterfly unit. The butterfly unit is the major building block of the FFT co-processor, which consists of mainly complex adder/subtractor and complex multiplier.

The conventional approach of DIT butterfly architecture with expanded complex numbers is shown in below figure 1. It consists of floating-point multiplication followed by floating-point addition and subtraction which leads to fused-multiply-add. The butterfly function cannot be directly implemented by this method even though it saves time, power and area. So, a dot-product unit which saves more time, area and power than fused-multiply-add unit is implemented**.**

The fused-dot-product-unit (FDPA) is implemented by combining a floating-point multiplication with a floating point three-operand addition. The following figure 2 depicts the FDPA unit with three-operand adder.

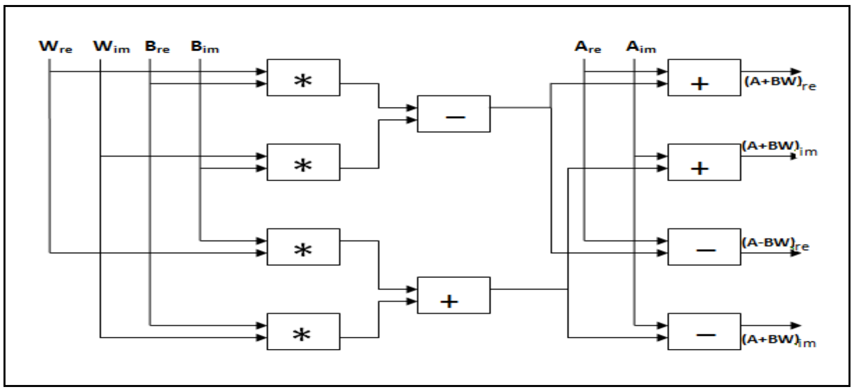


Figure 1. Conventional approach of DIT butterfly architecture with expanded complex numbers

The representation of floating-point operands i.e. exponent and significand has a great impact on the performance on the FDPA unit. The exponents are represented in twos complement after subtracting the bias according to the bias determined by the IEEE format, while the significands of A_{re}, A_{im}, B_{re} and B_{im} are represented in binary sign digit(BSD) format [6]. In BSD representation every bit takes the value of {-1,01} , where each BSD bit is represented by two bits posibit(x_i) and negabit(X_i).The modified booth encoding is used to store the significands of W, where there is at least one 0 in two adjacent positions so that only multiples of ±B and ±2B are produced.

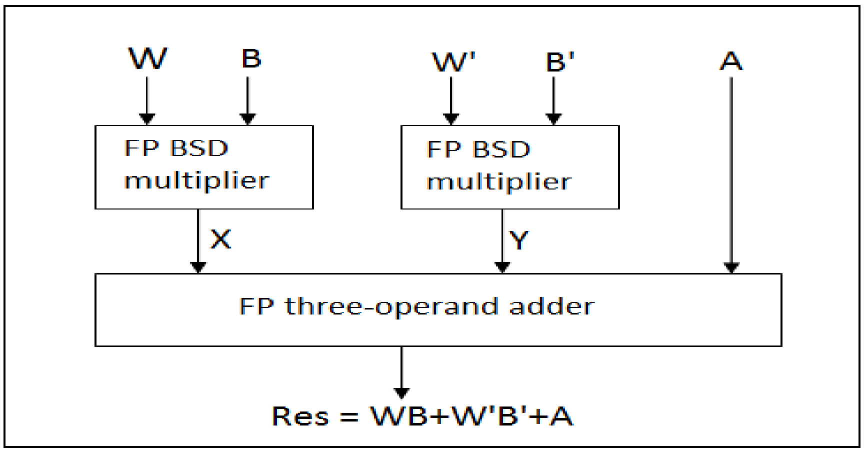


Figure 2. FDPA unit with three-operand adder

2.1. Redundant floating-point BSD multiplier

Multiplications in any arithmetic operations consumes more time and power, so design of low power multiplier unit is still popular in VLSI design. Many algorithms have been proposed to design efficient multipliers, in this paper we are going to design redundant floating-point multiplier based on binary sign digit representation. This design follows mainly two steps

1) Partial product generation and
2) Partial product reduction

2.1.1 Partial Product Generation (PPG)

Generation of the partial product is more complicated for the redundant multiplier. Partial products are generated by multiplying the multiplier by each bit of the multiplicand. The inputs (A, B, W) of the redundant multiplier are represented in different format, so the PPG step for redundant multiplier will be completely different form general methods.

The significand of W is stored in modified booth ranging {1, 0, 1}, where there is at least one 0 in two adjacent positions so that only multiples of ±B and ±2B are produced. This reduces the number of adders required to [n/2] which leads to simpler partial product generation to compute n-by-n multiplication. The table for generation of partial product is given below table 1 and equivalent circuit diagram is given in below figure 3. One partial product is generated per two binary positions of multiplicand W, each binary position of W consists of two bits (Wi^-Wi^+) to represent {-1, 0, 1}. The multiplicand B is represented in BSD, 2B is generated by 1-bit left shift over B and –B (-2B) is generated by inverting B (2B), each partial product consists of (n+1) digits.

Table 1. Generation of ith partial product

$W^-_{i+1}\ W^+_{i+1}$	$W^-_i\ W^+_i$	$\|W^-_{i+1}\ W^+_{i+1}\ W^-_i\ W^+_i\|$	PP_i
0 0	0 0	0	0
0 0	0 1	1	B
0 0	1 1	-1	-B
0 1	0 0	2	2 x B
1 1	0 0	-2	-2 x B

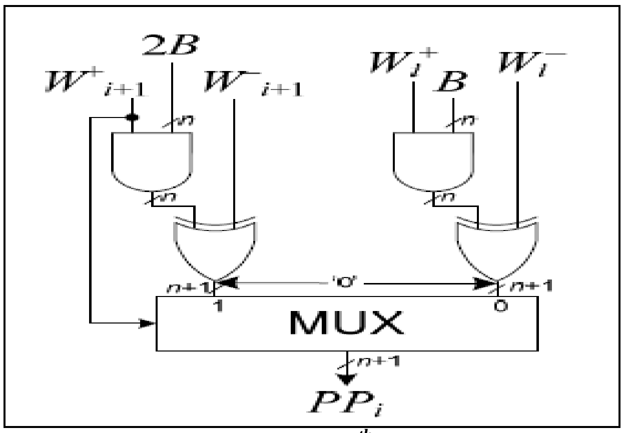


Figure 3. Generation of ith partial product

2.1.2 Partial Product Reduction (PPR):

The partial products generated in the PPG step are reduced using carry limit adders and the final product is produced. The BSD adder shown in figure 4 is used for the carry limit addition. The BSD number system is one of the optimized redundant number system, which is defined by radix 2 and digit set ranging {-1, 0, 1}. It is used for high speed calculations since it eliminates intermediate carry propagation in arithmetic operations. The PosNeg BSD adder for the carry-limited addition is shown below figure 4, the negabit is a bit in range {-1, 0} and is represented by capital letters, the posibit is a bit in range {0,1} and is represented in small letters.

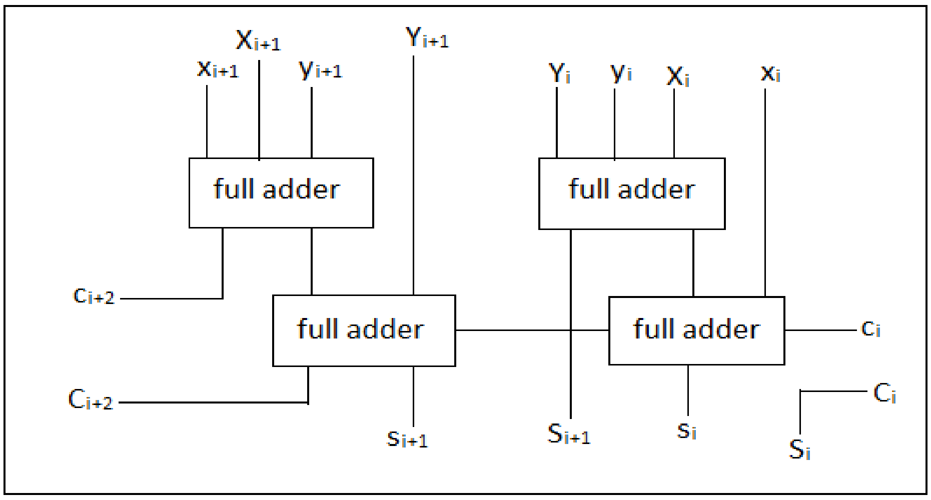


Figure 4. PosNeg BSD adder (2-digit)

For an n-by-n multiplier, the final product produced may be more than 2n, in this paper the significand of A and B which are represented in BSD are of 24 bits, and W(represented in modified booth) is of 25 bits. So, the output produced is of 51 bit. The BSD carry limit adders are used for the reduction of the partial production; here the reduction is done in four levels using 12 BSD adders. The redundant product of the BSD multiplier is given below after discarding bits in position 0 to 18. The final product is given to the three-operand adder, the overall block diagram of floating-point BSD multiplier is given below figure 5.

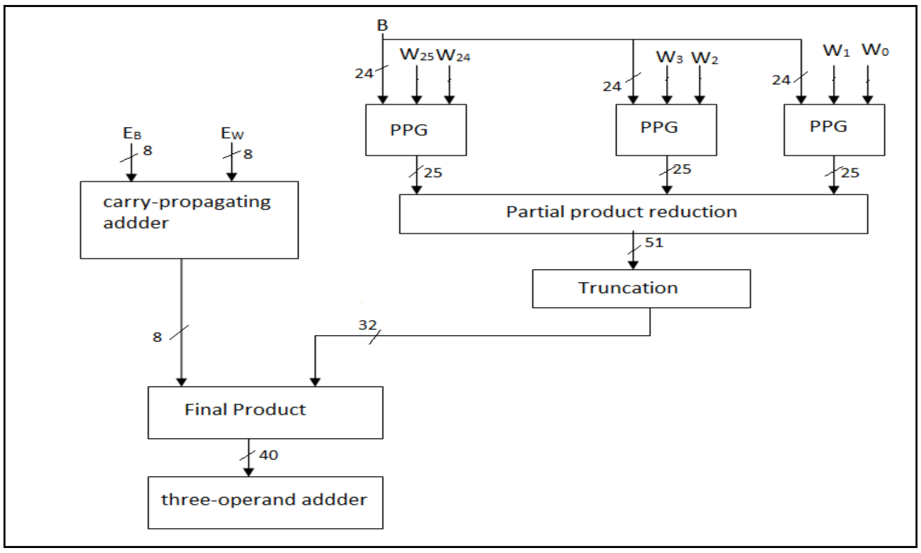


Figure 5. Floating-point BSD multiplier

2.2. FP BSD three-operand adder

A three-operand adder can be designed by using two 2-operand adders, but it leads to more power consumption so we design the Floating-point three operand BSD adder as shown in figure 6. The two inputs(X, Y) to the 3-operand FP adder are from redundant FP multiplier and the third input is A with a 24-digit significand. The building blocks of the three-operand floating point adder are given below

A binary subtractor is used to determine the larger exponent between X and Y i.e. $\Delta = (E_X - E_Y)$.

The multiplexer is used to select the significand with the smaller exponent and sends to a barrel shifter is which is used to shift the significand with the smaller exponent Δ bits to the right.

The output from the barrel shifter is sent to BSD adder along with the significand with larger exponent, finally the SUM output is produced.

The third-operand A is initially shifted 30 digits to left to reduce the critical path delay of three-operand adder. Then the value of $\Delta A = E_{big} - E_A + 30$ is computed, and the result is passed over barrel shifter. The barrel shifter then right shifts the operand A to ΔA positions.

The SUM output and the second barrel shifter output are added using BSD adder. The 59-bit output produced from the BSD adder is normalized using normalization block.

The normalization block mainly deals with calculation of number of leading zeroes using leading zero detection block (LZD)[7].

The exponent adjustment block subtracts the amount of left-shit bits from the largest exponent. Based on the round and guard bits the round value is determined and it is added to the digit in rounding position.

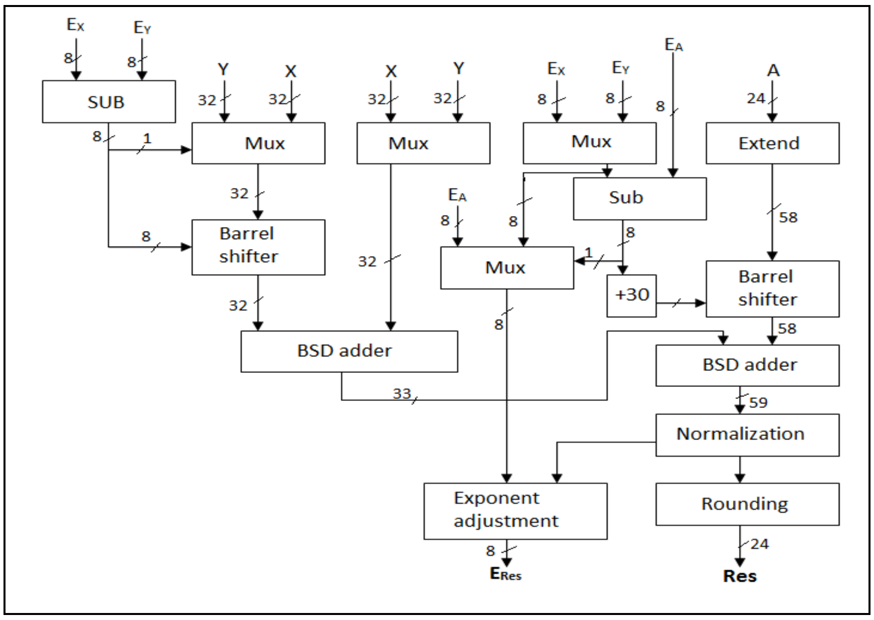


Figure 6. Floating-point three operand BSD adder

III. Proposed three-operand-adder using comparator

The floating-point three-operand BSD adder shown in figure 6 which contains four multiplexers is replaced by a single comparator in the proposed design. This makes the design easy by reducing the area and delay. The comparator is used to find the larger exponent from the three exponents. A new BSD adder is also proposed, this BSD adder can be used when BSD adder contains some inputs as zero. The proposed BSD adder is faster than the previous BSD adder and used in the proposed FP three-operand-adder. The block diagram of the BSD adder is shown in figure 7.

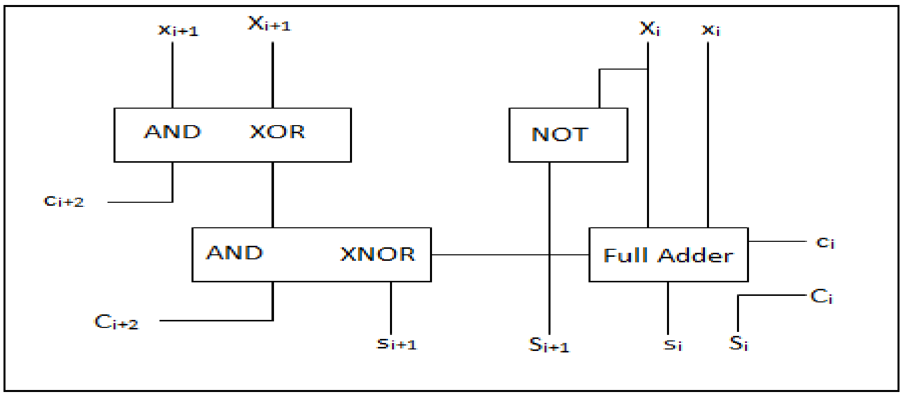


Figure 7. BSD adder with some inputs assigned to zero

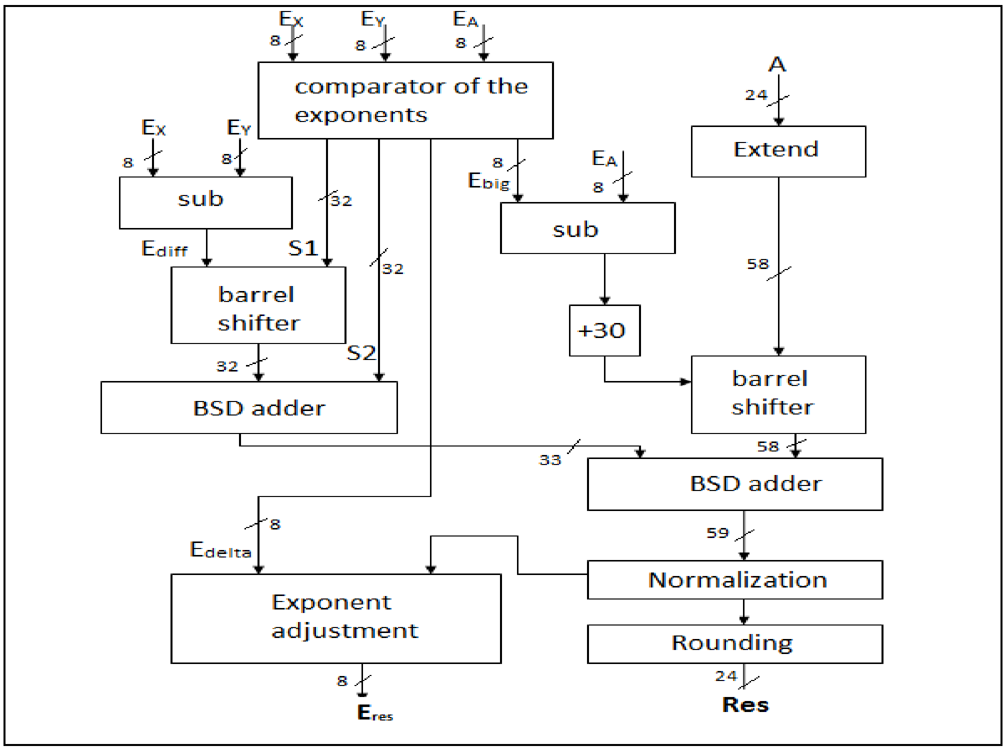
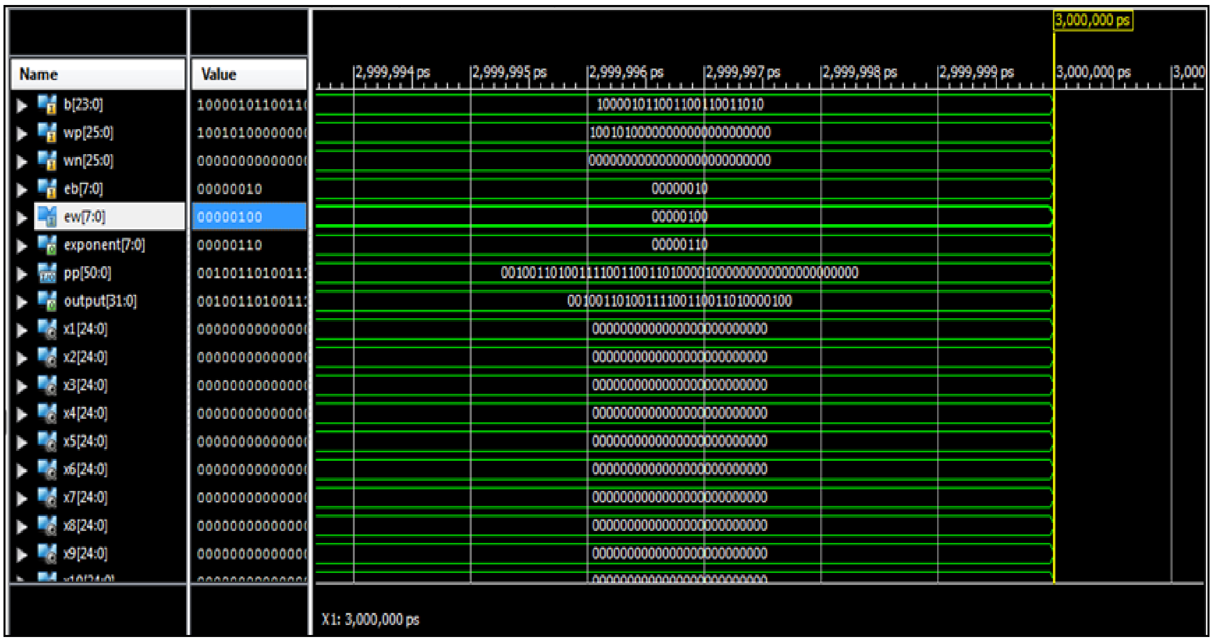


Figure 8. Proposed FP three-operand adder using comparator

The block diagram for the proposed three-operand adder using the comparator is shown in figure 8. The output S1 of the comparator represents the significand of exponent with smallest value, S2 represents the significand of exponent with largest value among X and Y respectively.

IV. Simulation results

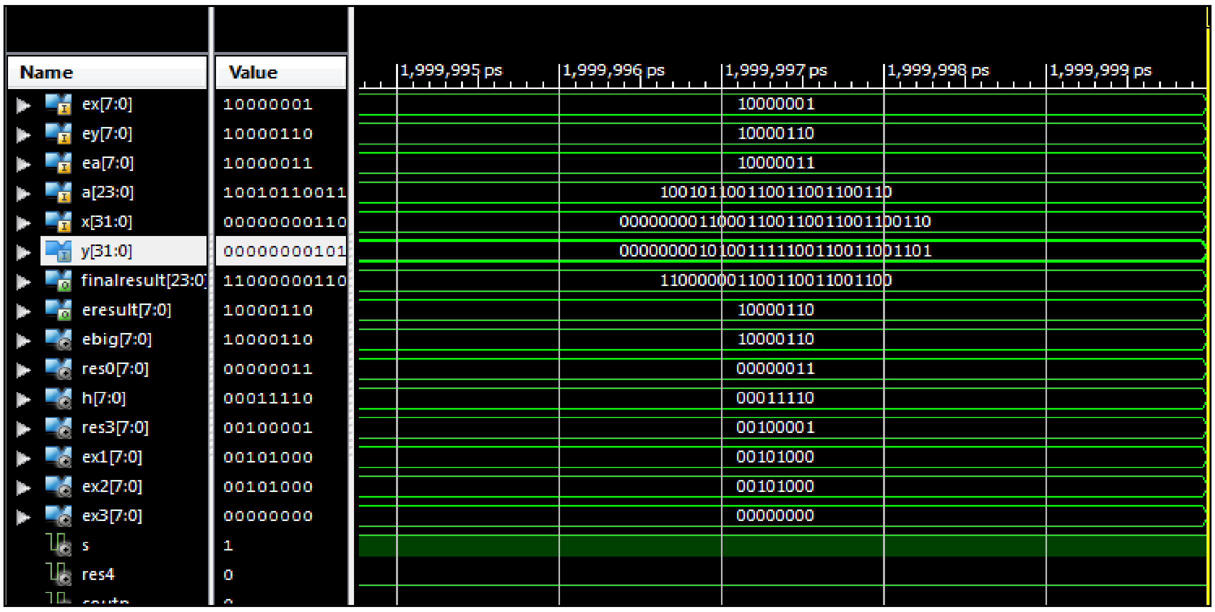
The simulation results of the FP BSD multiplier and FP three-operand BSD adder are shown in figure 9 and figure 10.



```
Name            Value             2,999,994 ps  2,999,995 ps  2,999,996 ps  2,999,997 ps  2,999,998 ps  2,999,999 ps  3,000,000 ps  3,000
b[23:0]         1000010110011     100001011001100110011010
wp[25:0]        1001010000000     10010100000000000000000000
wn[25:0]        0000000000000     00000000000000000000000000
eb[7:0]         00000010          00000010
ew[7:0]         00000100          00000100
exponent[7:0]   00000110          00000110
pp[50:0]        0010011010011     001001101001111001100110100001000000000000000000000
output[31:0]    0010011010011     00100110100111100110011010000100
x1[24:0]        0000000000000     0000000000000000000000000
x2[24:0]        0000000000000     0000000000000000000000000
x3[24:0]        0000000000000     0000000000000000000000000
x4[24:0]        0000000000000     0000000000000000000000000
x5[24:0]        0000000000000     0000000000000000000000000
x6[24:0]        0000000000000     0000000000000000000000000
x7[24:0]        0000000000000     0000000000000000000000000
x8[24:0]        0000000000000     0000000000000000000000000
x9[24:0]        0000000000000     0000000000000000000000000

X1: 3,000,000 ps
```

Figure 9. Simulation result of Floating-point BSD multiplier



```
Name               Value          1,999,995 ps  1,999,996 ps  1,999,997 ps  1,999,998 ps  1,999,999 ps
ex[7:0]            10000001       10000001
ey[7:0]            10000110       10000110
ea[7:0]            10000011       10000011
a[23:0]            10010110011    100101100110011001100110
x[31:0]            00000000110    00000000110001100110011001100110
y[31:0]            00000000101    00000000101001111100110011001101
finalresult[23:0]  11000000110    110000001100110011001100
eresult[7:0]       10000110       10000110
ebig[7:0]          10000110       10000110
res0[7:0]          00000011       00000011
h[7:0]             00011110       00011110
res3[7:0]          00100001       00100001
ex1[7:0]           00101000       00101000
ex2[7:0]           00101000       00101000
ex3[7:0]           00000000       00000000
s                  1
res4               0
```

Figure 10. Simulation result of Floating-point three-operand BSD adder

V. Evolutions and comparisons

The evaluation results of the architectures proposed are presented and compared with previous architectures using Xilinx, Spartan 6 FPGA. The comparison of the proposed design with the existing one is shown in the table 2.

Table 2. Comparison of existing and proposed three-operand adders

	Existing three-operand adder	**Proposed three-operand adder**
Exponent comparison and significand alignment	2 x (8 bit sub), 4 x MUX, 2 x shifter, +30 block	2 x (8 bit sub), comparator, 2 x shifter, +30 block
Significand addition	2 x existing BSD adder	2 x proposed BSD adder

The comparison of the results of the existing and proposed architectures using Xilinx Spartan 6 FPGA is given below in table 3.

Table 3. Comparison of results of the existing and proposed architectures

	tech.	**delay (ns)**	**Area (μm^2)**
Existing three-operand adder	45 nm	3.24	57,122
Proposed three-operand adder	45 nm	2.75	51,450

VI. Conclusion

High speed butterfly architectures are designed using BSD representation and fused-dot-product. The use of redundant number system for the significands of the floating-point operands eliminates the intermediate carry-propagation. Modified booth encoding is used to speed up the floating-point multiplier. The fused-dot-product unit (FDPA) combines more floating-point operands; higher speed is achieved by eliminating extra leading-zero-detection (LZD), normalization and rounding units. A new BSD adder is proposed to further speed up BSD additions which in turn increases the speed of butterfly architecture. A comparator is designed and used in the design of three-operand adder; this comparator replaces four multiplexers thus reduces the delay and area in the proposed design. The results were compared using Spartan 6 FPGA 45nm technology. The delay is 15% less and the area is reduced to 51,450 μm^2 when compared to existing three-operand adder.

VII. REFERENCES

[1] Amir Kaivani and SeokbumKo, "Floating-Point Butterfly Architecture Based on Binary Signed-Digit Representation" ,Apr. 2015
[2] IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754-2008, Aug. 2008, pp. 1–58.
[3] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, 2nd Ed. New York, NY, USA: Oxford Univ. Press, 2010.
[4] E.E. Swartzlander, Jr., and H.H. Saleh, "FFT implementation with fused floating-point operations," IEEE Trans. Comput., vol. 61, no. 2, pp. 284–288, Feb. 2012.
[5] J.Sohn and E.E. Swartzlander, Jr., "Improved architectures for a floating-point fused dot product unit," in Proc. IEEE 21st Symp. Comput. Arithmetic, Apr. 2013, pp. 41–48.
[6] Somayeh Timarchi, Parham Ghayour, and Asadollah Shahbahrami, "A Novel High-Speed Low-Power Binary Signed-Digit Adder" Computer Architecture and Digital Systems (CADS) 2013 17th CSI International Symposium on, pp. 123-124, 2013.
[7] V. G. Oklobdzija IEEE Transactions on Very Large Scale Integration (VLSI) Systems," An Algorithmic and Novel Design of a Leading Zero Detector Circuit: Comparison with Logic Synthesis" Year: 1994, Volume: 2, Issue: 1 Pages: 124 – 128
[8] CH JAYAPRAKASH, T.PrudhviRaju, "Design of FIR Filter using Fifth approximation adder", Proc. International journal technology Research in engineering (IJTRE) Oct 2014, Vol.03,Isuue.02,ISSN 2347-4718.
[9] CH JAYAPRAKASH, "Design of low power scalable Digital Comparator using a parallel prefix tree", Proc. International conference on electrical, electronics & computer systems (ICEECs), vol.02, issue8, 9, 2014ISSN(online)-2347-2820.