A Novel IEEE-754 Floating-Point Butterfly Architecture based on Multi Operand Adders

Thota Swetha, S. Srinivas

***Abstract:** FFT (Fast Fourier Transform) is one of most efficient algorithm widely used in communication systems.FFT function consists of Butterfly units with multiply add operations over complex numbers. A floating point is applied to FFT design, mainly to butterfly units. The concentrated tasks are calculated from general purpose processor by order FP concerns. The significant drawback of FP butterfly is its slowness in its examination with its fixed point. In proposed FP butterfly uses a fused dot product add(FDPA) unit to calculate butterfly unit, depending on binary signed digit(BSD).A BSD adder is introduced and utilized as a part of the three operand adder and parallel BSD multiplier, in order to enhance the speed of the FDPA unit. A modified booth encoding is utilized to accelerate BSD multiplier. The results shows that proposed FP butterfly design is considerably speedier than past butterfly design.*

***Index Terms:** Binary-Signed Digit (BSD) Representation, Butterfly Unit, Complex Number System, Fast Fourier Transform (FFT), Floating-Point (FP), Redundant Number System, Three-Operand Addition.*

I. INTRODUCTION

Fast Fourier Transform (FFT) is utilized as a part of numerous applications like range analyzers, multidimensional change, ghastly music, OFDM based remote broadband communication system and numerous signal handling applications. Discrete Fourier Transform is an intense instrument to perform recurrence examination of a signal.

The primary inconvenience being the quantity of calculations required to compute the DFT coefficients. Fast Fourier Transform is a proficient calculation in calculation the DFT of a signal.

The FFT calculation decreases the calculation time and in addition calculation complexity nature when contrasted with coordinate calculation of DFT.

Forms of FFT algorithms

Decimation in Time (DIT): In Decimation in time algorithm, the input sequence is divided into subsequences, i.e., even and ordinary sequences.

These subsequences are once more divided in possible method.

The DFTs of those sequences are calculated and then ultimately combined.

In this algorithm, sequence is cut up into smaller subsequences in time domain.

Figure 1 exhibit how the Decimation in Time algorithm works.

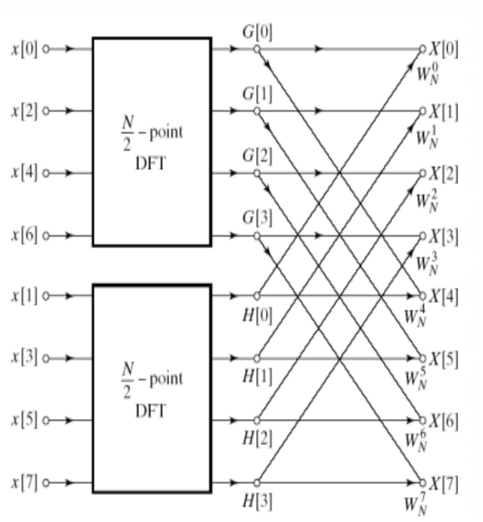


Fig 1: Division in N/2-point DFTs

Decimation in Frequency (DIF): In decimation in Frequency calculation, rather than partitioning the arrangement in time, Frequency samples of DFT are disintegrated into littler groupings. In this calculation recurrence tests are isolated into to even and odd examples. This decay is proceeded till we get the 2-point DFTs. In Decimation in Frequency calculation, symmetry of decay is turned around from Decimation in time, i.e., disintegration continues from left to right. Figure 6 demonstrates the last stream chart of 8-point DFT utilizing Decimation in Frequency FFT calculation.

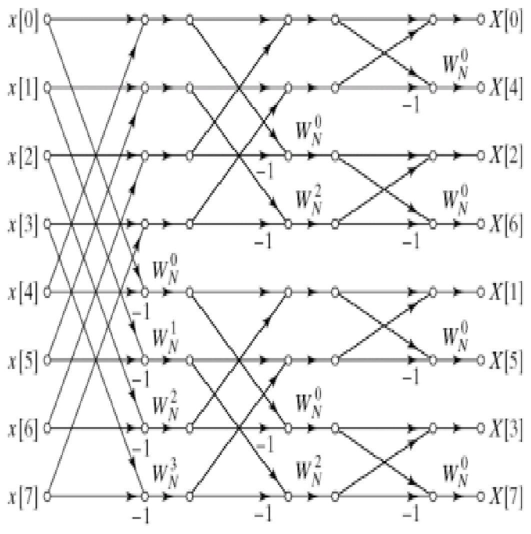


Fig. 2: Point DFT using decimation in frequency Fixed point Representation

Manuscript received January 25, 2019.

Thota Swetha, PG Scholar, SR Engineering College, Ananthasagar, Hasanparty, Warangal. Telangana, India

S. Srinivas, Associate Professor, SR Engineering College, Ananthasagar, Hasanparty, Warangal, Telangana, India. (e-mail:srinivas_s@srecwarangal.ac.in)

The sign field, integer field, and fractional fields are 3 parts in fixed point representation of a number. A 32-bit number is stored in the format is hold 1 bit for sign,15 bits for integer part and 16 bits for fractional part. A exceeded 32-bit number representation would need to be stored estimatedly. On a computer, 0 is used to represent + and 1 is used to represent.

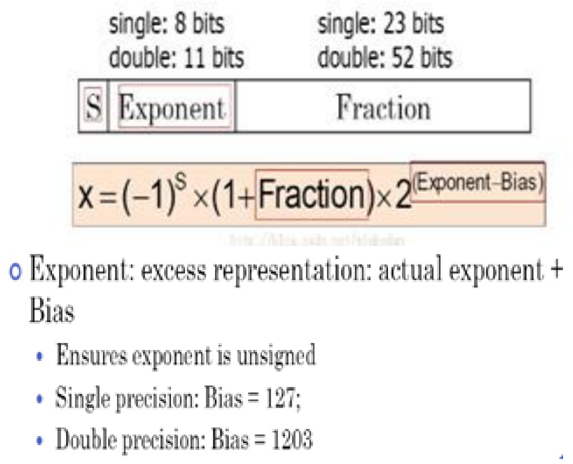
Example. The 32-bit string 1 | 000000000101011 | 1010000000000000

represents (−101011.101)2 = −43.625.

The fixed point notation, although not without virtues, is usually inadequate for numerical analysis as it does not allow enough numbers and accuracy.

Floating Point Representation

FLOATING-POINT arithmetic provides a wide dynamic range, freeing special purpose processor designers from the scaling and overflow/underflow concerns that arise with fixed-point arithmetic. Use of the IEEE-754 standard 32-bit floating-point format also facilitates using the fast Fourier transform (FFT) processors as coprocessors in collaboration with general purpose processors.



single: 8 bits
double: 11 bits

single: 23 bits
double: 52 bits

S	Exponent	Fraction

$$x=(-1)^S\times(1+\text{Fraction})\times 2^{(\text{Exponent}-\text{Bias})}$$

- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single precision: Bias = 127;
  - Double precision: Bias = 1203

AFP FFT butterfly architecture, as shown in Fig. 3, is made of FP multipliers followed by FP adders/subtractors to compute, for instance, AB + CD + E.

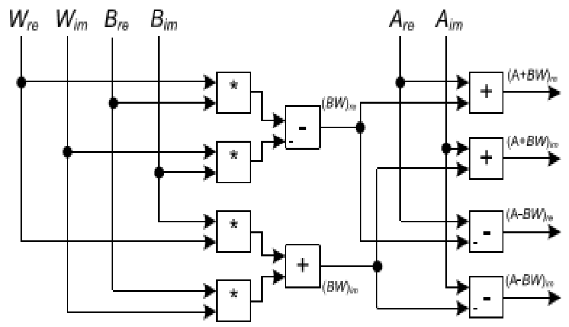


Fig. 3: FFT butterfly architecture with expanded complex numbers.

Merging the operations can be done in two ways: 1. Merging the multiplication with the next addition. This leads to a dot product unit, to compute, e.g. (AB + CD), followed by an addition (requires a four-operand adder). 2. Merging the multiplication with the next two additions.

This leads to a fused-dot-product-add unit(FDPA) (requires a three-operand adder). The multiplication in the dot-product unit is made of three phases: (i) partial product generation (PPG), (ii) partial product reduction (PPR) and (iii) the final addition. The dot-product unit actually merges phase with the next addition. The fused-dot-product-add unit is designed based on a three-operand adder which obtains the product (e.g. AB) from the FP multiplier and adds it with the other two operands (e.g. CD and E).

II. EXISTING SYSTEM

The discrete and fused units are made to show utility of Fused DP and Fused AS units for FFT execution, FFT butterfly units. First, radix-2 decimation in frequency FFT butterfly was designed. It is shown in Fig. 2. All lines carry complex pairs of 32-bit IEEE-754 numbers and all operations are unpredictable. In fig3 the complex add, subtract, and multiply operations shown. These operations includes complex add or subtract with to real adders and complex multiplier performed by four real multipliers and real adders. The six real adders and four real multipliers are used for complete butterfly as shown on the figure. In this and the following figure, for the IEEE-754 single-precision data all lines are 32-bits wide. Alternatively, as shown in Fig.4, the two fused add-subtract units (marked as FAS in the figure) are used to perform complex add and subtract and the two fused dot product units (marked as FDP) are used to perform complex multiplication. These results are acquired from having completed a total format for the butterfly unit. Thus, the sum of the parts (i.e., four multipliers and six adders for the discrete version or two Fused DP and two Fused AS units for the fused version) are smaller than the areas. The divergence is because of the clock conveyance and I/O circuits. Similarly, the delays are less than the sum of the delays of the parts. This is because the conditions that produce the worst-case delay for one part are different from the conditions that produce the worst-case delay for other parts.

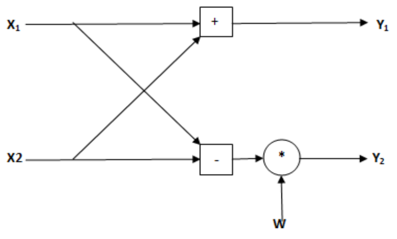


Fig. 4: Radix-2 DIF FFT butterfly

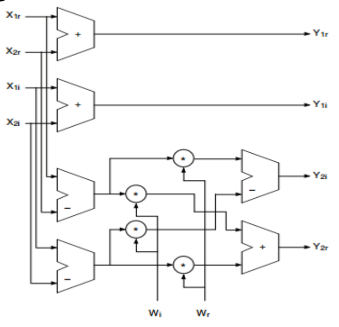


Fig. 5: Discrete implementation of the radix-2 DIF FFT butterfly

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

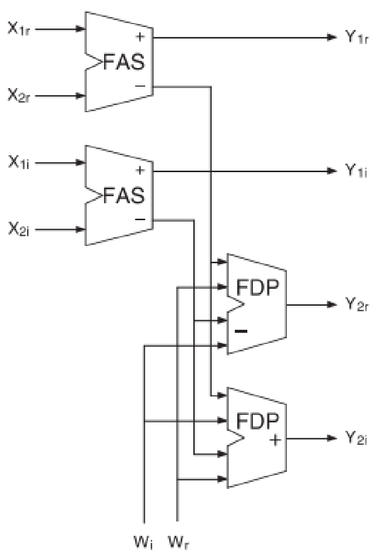


Fig. 6: Fused implementation of the radix-2 DIF FFT butterfly

III. PROPOSED SYSTEM

The FFT could be implemented in hardware based on an efficient algorithm in which the *N*-input FFT computation is simplified to the computation of two (*N*/2)-input FFT. The butterfly unit is called as decomposition leads to 2-input FFT block. The proposed butterfly unit consists a complex fused-multiply– add with FP operands. In Fig. 1 the required modules shows by expanding the complex numbers.

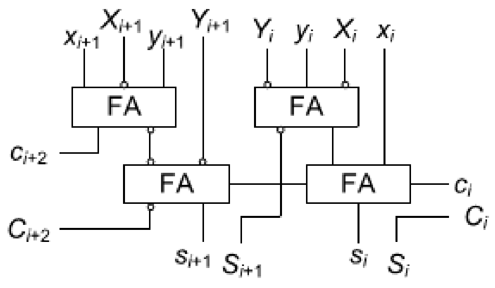


Fig. 6: BSD adder (two-digit slice).

According to Fig. 3, the constituent operations for butterfly unit are a dot-product (e.g., *BreWim* + *Bim Wre)* followed by an addition/subtraction which leads to the proposed FDPA operation (e.g., *BreWim* + *Bim Wre* + *Aim)*. The below discussion is about Implementation details of FDPA, over FP operands.

Table 1: Generation of a PP

$W^-_{i+1}W^+_{i+1}$	$W^-_iW^+_i$	$W^-_{i+1}W^+_{i+1}\,W^-_i\,W^+_i$	PP_i
0 0	0 0	0	0
0 0	0 0	1	B
0 0	1 1	-1	-B
0 1	0 0	2	2xB
1 1	0 0	-2	-2xB

The significands of Are, Aim, Bre, and Bim are represented in BSD when all inputs are represented in two's complement (after subtracting the bias).Within this representation every binary position takes values of {−1, 0, 1} represented by one negative-weighted bit (negabit) and one positive-weighted bit (posibit). The carry-limited addition circuitry for BSD numbers is shown in Fig. 5, where capital (small) letters symbolizes negabits (posibits).

The critical path delay of this adder consists of three full-adders. The FDPA is introduced with a redundant FP multiplier took after by a redundant FP three-operand adder.

Proposed Redundant Floating-Point Multiplier

The partial product generation (PPG) and PP reduction (PPR) are two main stages in proposed multiplier, like manner other parallel multipliers. However, contrary to the conventional multipliers, our multiplier keeps the product in redundant format and hence there is no need for the final carry-propagating adder. The exponents of the input operands are taken care of in the same way as is done in the conventional FP multipliers; however, normalization and rounding are left to be done in the next block of the butterfly architecture (i.e., three-operand adder).

1) Partial Product Generation: The PPG step of the proposed multiplier is completely different from that of the conventional one because of the representation of the input operands (B, W, B, W). Moreover, given that Wre and Wim are constants [5], the multiplications in Fig. 3 (over significands) can be computed through a series of shifters and adders. With the intention of reducing the number of adders, we store the significand of W in modified Booth encoding. Given the modified Booth representation of Wre and Wim , one PP, selected from multiplicand B, is generated per two binary positions of the multiplier W, as shown in Table I. Fig. 7 shows the required circuitry for the generation of PPi based on Table I where each PP consists of (n + 1) digits (i.e., binary positions).

2) Partial Product *Reduction:* The major constituent of the PPR step is the proposed carry-limited addition over the operands represented in BSD format. This carry-limited addition circuitry is shown in Fig. 6 (two-digit slice).

Since each PP (PP*i)* is (*n* + 1)-digit (*n*, . . . , 0) which is either *B* (*n* − 1, . . . , 0) or 2*B* (*n*, . . . , 1), the length of the final product may be more than 2*n*.

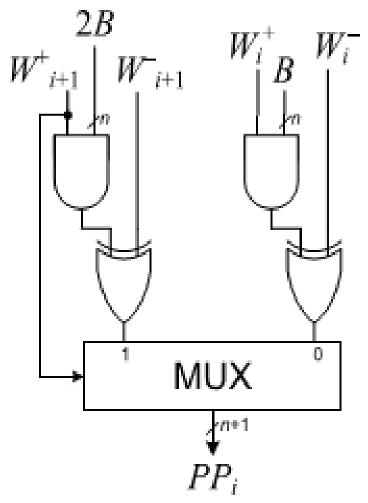


Fig. 7: Generation of the ith PP

*Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication*

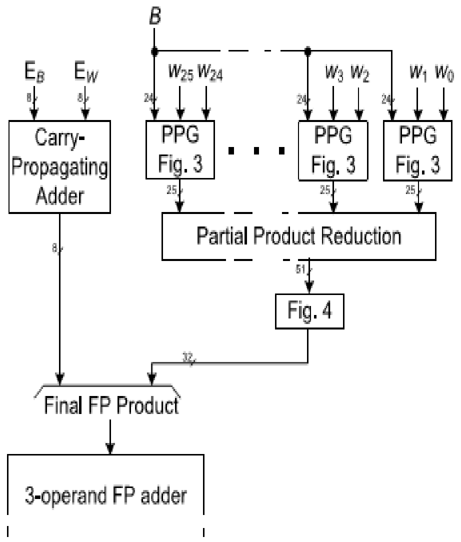


Fig. 8: Proposed redundant FP multiplier

Assuming that the sign-embedded significands of inputs *A* and *B* (24 bits) are represented in BSD; while that of *W* is represented in modified Booth encoding (25 bits), the last PP has 24-(binary position) width (instead of 25), given that the most significant bit of *W* is always 1.The reduction of the PPs is done in four levels using 12 BSD adders. Given that *B* is in ±[1, 2) and *W* in [1, 2), the final product is in ±[1, 4) and would fit into 48 binary position (47…0). Consequently, positions 45 down to 0 are fractions. Similar to standard binary representation, Guard (G) and Round (R) positions are sufficient for correct rounding. Therefore, only 23 + 2 fractional binary positions of the final product are required to guarantee the final error <2−23. Selecting 25 binary positions out of 46 fractional positions of the final product dismisses positions 0 to 20. However, the next step addition may produce carries to G and R positions. Nevertheless, because of the carry-limited BSD addition, contrary to standard binary addition, only positions 20 and 19 may produce such carries. In overall, positions 0 to 18 of the final product are not useful and hence a simpler PPR tree is possible. Fig. 7 shows the required digits passed to the three-operand adder.

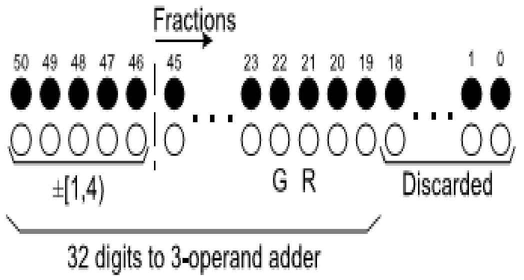


Fig. 9: Digits to three-operand adder

B. Proposed Redundant Floating-Point Three-Operand Adder

The two FP adders are connected to a three –operand FP addition is a direct way in order to save latency, power and area consumption. A better way is to use fused three-operand FP adders. In the proposed three-operand FP adder, a new alignment block is implemented and CSA–CPA are replaced by the BSD adders (Fig. 5). Moreover, sign logic is eliminated. The bigger exponent between *E X* and *EY* (called *E* Big*)* is determined using a binary subtractor (Δ = *E X − EY)*; and the significand of the operand with smaller exponent (*X* or *Y)* is shifted Δ–bit to the right. Next, a BSD adder computes the addition result (SUM = *X* + *Y)*, using the aligned *X* and *Y.*

Adding third operand (i.e., SUM+ *A)* requires another alignment. This second alignment is done in a different way so as to reduce the critical path delay of the three-operand adder. First, the value of *A* = *E* Big − *E A* + 30 is computed which shows the amount of right shifts required to be performed on extended *A* (with the initial position of 30 digits shifted to left). Fig. 10 shows the alignments implemented in the proposed three-operand FP adder. Next, the aligned third significand (58-digit) is added by a BSD adder to SUM (33-digit) generated from the first BSD adder. Since in 58-digit BSD adder input operands have different number of digits.

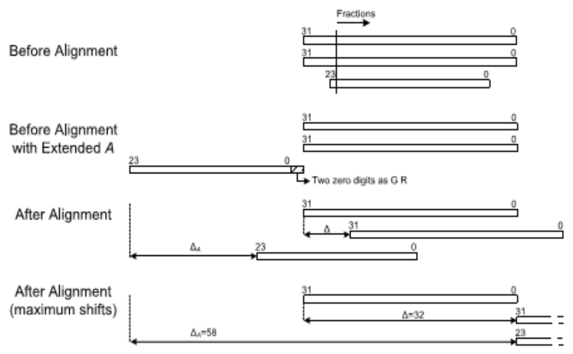


Fig. 10: Proposed three-operand alignment scheme

Next steps are normalization and rounding, which are done using conventional methods for BSD representation. It should be noted that the leading zero detection (LZD) block could be replaced by a four-input leading-zero-anticipation for speed up but at the cost of more area consumption. The other modification would replace our single path architecture with the dual path to sacrifice area for speed.

The appeared Fig.11 is introduces a FP three-operand adder with new alignment and addition blocks. In addition, because of the sign-embedded presentation of the significands (i.e., BSD), a sign logic is not required.

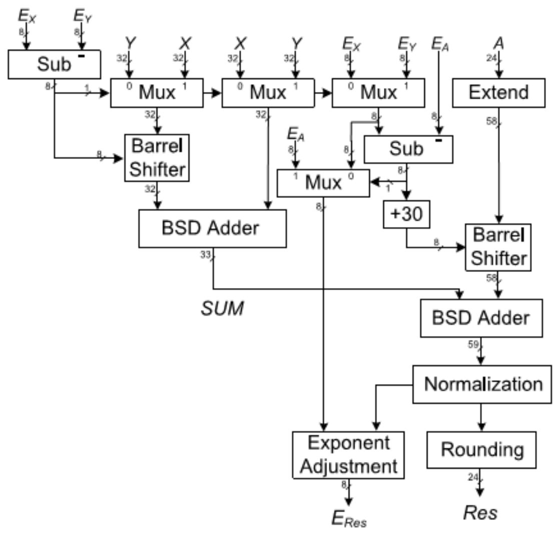


Fig. 11: Proposed FP three-operand addition.

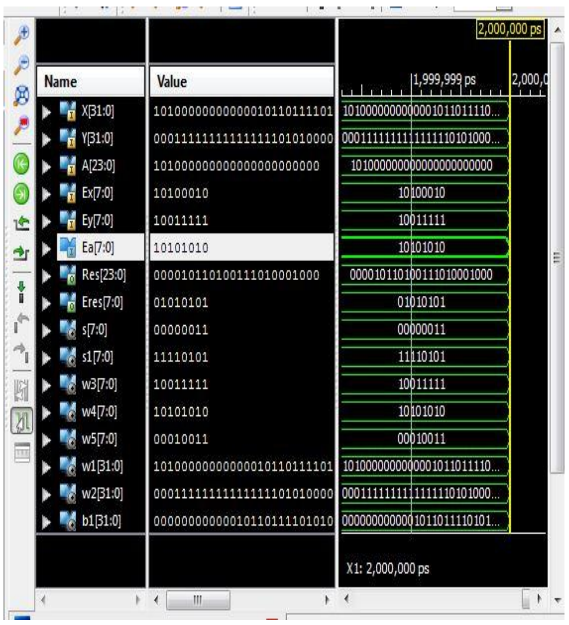
IV. RESULTS

The Verilog HDL Modules have successfully simulate, verified and synthesized using Xilinxise13.2.

Proposed system results:

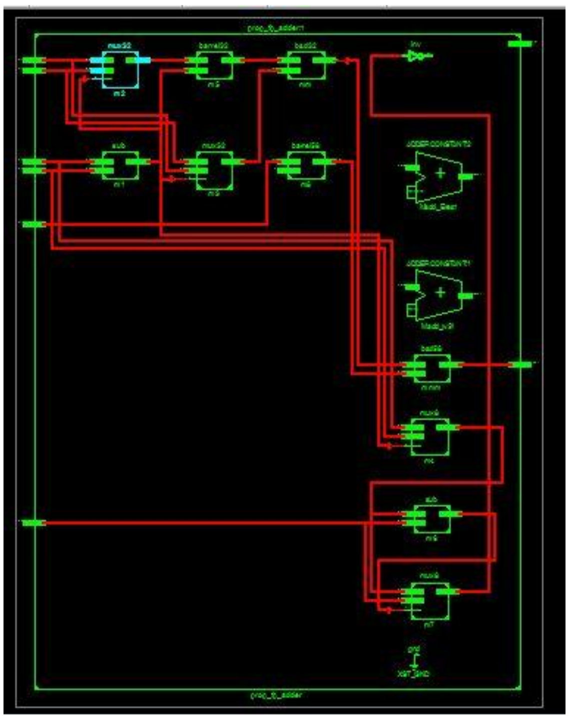
Fp adder results

Simulation results

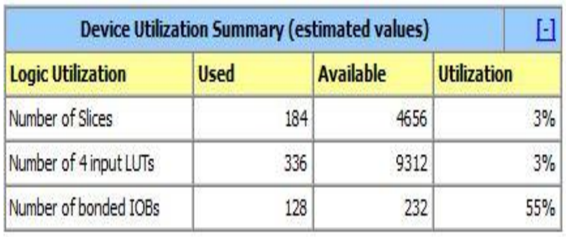


Name	Value
X[31:0]	10100000000000001011011101
Y[31:0]	000111111111111111101010000
A[23:0]	101000000000000000000000
Ex[7:0]	10100010
Ey[7:0]	10011111
Ea[7:0]	10101010
Res[23:0]	000010110100111010001000
Eres[7:0]	01010101
s[7:0]	00000011
s1[7:0]	11110101
w3[7:0]	10011111
w4[7:0]	10101010
w5[7:0]	00010011
w1[31:0]	10100000000000001011011101
w2[31:0]	000111111111111111101010000
b1[31:0]	00000000000010110111101010

RTL Schematic:

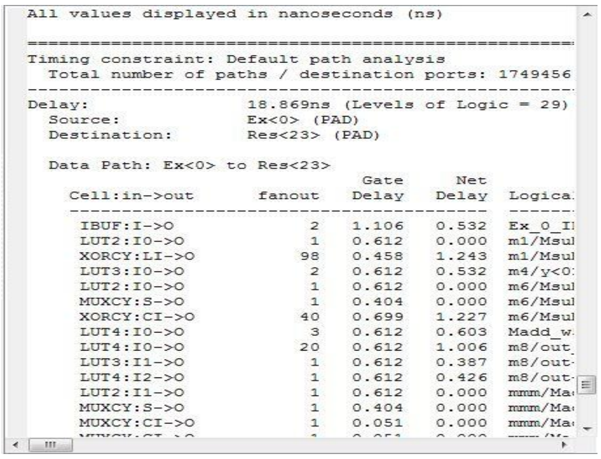


Design summary:



Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	184	4656	3%
Number of 4 input LUTs	336	9312	3%
Number of bonded IOBs	128	232	55%

Timing report:



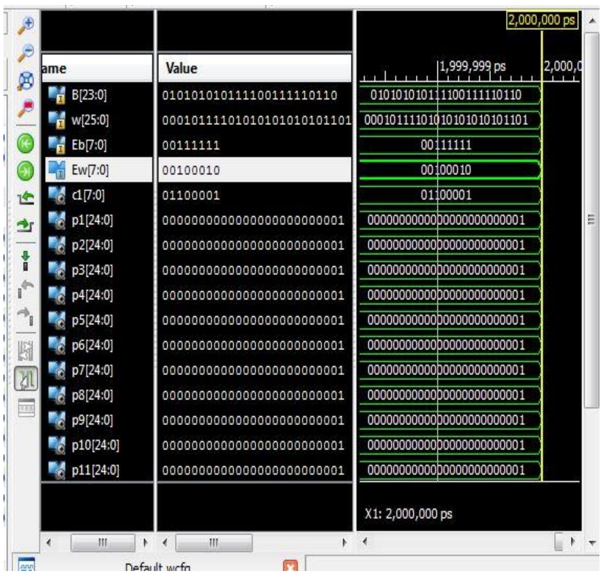
```
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 1749456
-------------------------------------------------------------------------
Delay:               18.869ns (Levels of Logic = 29)
  Source:            Ex<0> (PAD)
  Destination:       Res<23> (PAD)

  Data Path: Ex<0> to Res<23>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logica
    ----------------------------------------  ------
     IBUF:I->O             2   1.106   0.532  Ex_0_I
     LUT2:I0->O            1   0.612   0.000  m1/Msul
     XORCY:LI->O          98   0.458   1.243  m1/Msul
     LUT3:I0->O            2   0.612   0.532  m4/y<0
     LUT2:I0->O            1   0.612   0.000  m6/Msul
     MUXCY:S->O            1   0.404   0.000  m6/Msul
     XORCY:CI->O          40   0.699   1.227  m6/Msul
     LUT4:I0->O            3   0.612   0.603  Madd_w
     LUT4:I0->O           20   0.612   1.006  m8/out_
     LUT3:I1->O            1   0.612   0.387  m8/out-
     LUT4:I2->O            1   0.612   0.426  m8/out-
     LUT2:I1->O            1   0.612   0.000  mmm/Ma
     MUXCY:S->O            1   0.404   0.000  mmm/Ma
     MUXCY:CI->O           1   0.051   0.000  mmm/Ma
```

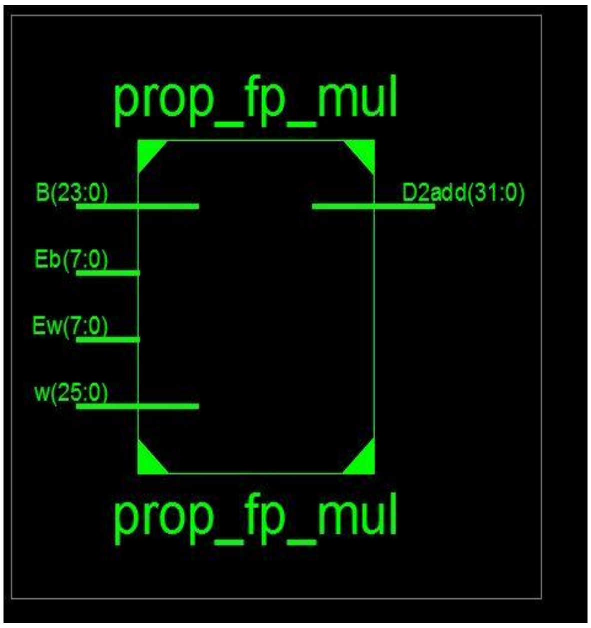
Fp mutiplier results:

Simulation results



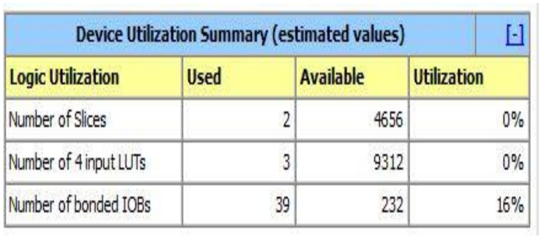
Name	Value
B[23:0]	010101010111100111110110
w[25:0]	00010111101010101010101101
Eb[7:0]	00111111
Ew[7:0]	00100010
c1[7:0]	01100001
p1[24:0]	0000000000000000000000001
p2[24:0]	0000000000000000000000001
p3[24:0]	0000000000000000000000001
p4[24:0]	0000000000000000000000001
p5[24:0]	0000000000000000000000001
p6[24:0]	0000000000000000000000001
p7[24:0]	0000000000000000000000001
p8[24:0]	0000000000000000000000001
p9[24:0]	0000000000000000000000001
p10[24:0]	0000000000000000000000001
p11[24:0]	0000000000000000000000001

RTL Schematic:



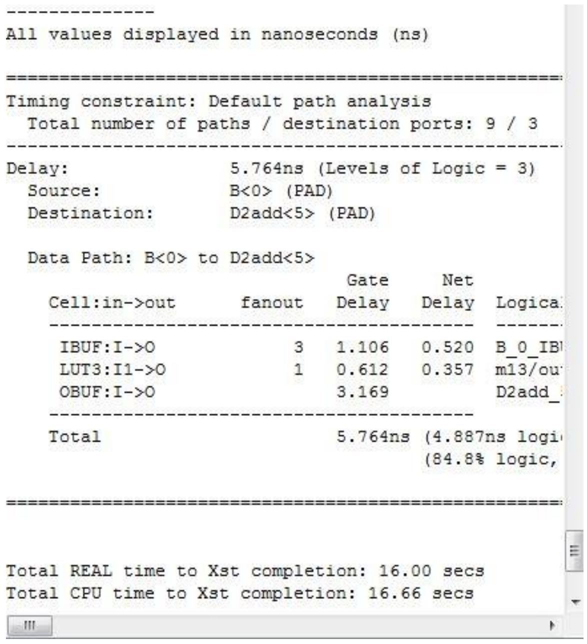
Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

Design summary:



Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	2	4656	0%
Number of 4 input LUTs	3	9312	0%
Number of bonded IOBs	39	232	16%

Timing report:



```
--------------
All values displayed in nanoseconds (ns)

=====================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 9 / 3
-----------------------------------------------------
Delay:               5.764ns (Levels of Logic = 3)
  Source:            B<0> (PAD)
  Destination:       D2add<5> (PAD)

  Data Path: B<0> to D2add<5>
                                 Gate     Net
    Cell:in->out      fanout    Delay   Delay  Logica
    ----------------------------------------  ------
      IBUF:I->O            3    1.106   0.520  B_0_IB
      LUT3:I1->O           1    0.612   0.357  m13/ou
      OBUF:I->O                 3.169          D2add_
    ----------------------------------------
    Total                       5.764ns (4.887ns logi
                                        (84.8% logic,

=====================================================


Total REAL time to Xst completion: 16.00 secs
Total CPU time to Xst completion: 16.66 secs
```

V. CONCLUSION

This paper introduces a rapid FP butterfly design, which is speedier than past works. The reason for this speed change is in two ways. One is By eliminating carry propagation with BSD representation and Second is new FDPA unit is introduced in brief. FDPA is combination of multipliers and adders used for FP butterfly, in this way higher speed is accomplished.

REFERENCES

1. IEEE Standard for Floating-Point Arithmetic, ANSI/IEEE Standard 754-2008, Aug. 2008.
2. R.K. Montoye, E. Hokenek, and S.L. Runyon, “Design of the IBM RISC System/6000 Floating-Point Execution Unit,” IBM J. Research and Development, vol. 34, pp. 59-70, 1990.
3. E. Hokenek, R.K. Montoye, and P.W. Cook, “Second-Generation RISC Floating Point with Multiply-Add Fused,” IEEE J. Solid-State Circuits, vol. 25, no. 5, pp. 1207-1213, Oct. 1990.
4. D. Takahashi, “A Radix-16 FFT Algorithm Suitable for Multiply-Add Instruction Based on Goedecker Method,” Proc. Int’l Conf. Multimedia and Expo, vol. 2, pp. II-845-II-848, July 2003.
5. J.H. McClellan and R.J. Purdy, “Applications of Digital Signal Processing to Radar,” Applications of Digital Signal Processing, A.V. Oppenheim, ed., pp. 239-329, Prentice-Hall, 1978.
6. B. Gold and T. Bially, “Parallelism in Fast Fourier Transform Hardware,” IEEE Trans. Audio and Electroacoustics, vol. AU-21, no. 1, pp. 5-16, Feb. 1973.
7. H.H. Saleh and E.E. Swartzlander, Jr., “A Floating-Point Fused Dot-Product Unit,” Proc. IEEE Int’l Conf. Computer Design (ICCD), pp. 427-431, 2008.
8. M.P. Farmwald, “On the Design of High-Performance Digital Arithmetic Units,” PhD thesis, Stanford Univ., 1981.
9. P.-M. Seidel and G. Even, “Delay-Optimized Implementation of IEEE Floating-Point Addition,” IEEE Trans. Computers, vol. 53, no. 2, pp. 97-113, Feb. 2004.
10. H. Saleh and E.E. Swartzlander, Jr., “A Floating-Point Fused Add-Subtract Unit,” Proc. IEEE Midwest Symp. Circuits and Systems (MWSCAS), pp. 519- 522, 2008.