A BSD ADDER REPRESENTATION USING FFT BUTTERFLY ARCHITECHTURE

MD.Hameed Pasha[1], *J.Radhika*[2]
[1.] *Associate Professor, Jayamukhi Institute of Technogical Sciences, Narsampet, India*
[2.] *Department of ECE, Jayamukhi Institute of Technological Sciences, Narsampet, India*

Abstract: Fast Fourier transform (FFT) coprocessor, having a significant impact on the performance of communication systems, has been a hot topic of research for many years. The FFT function consists of consecutive multiply add operations over complex numbers, dubbed as butterfly units. Applying floating-point (FP) arithmetic to FFT architectures, specifically butterfly units, has become more popular recently. It offloads compute-intensive tasks from general-purpose processors by dismissing FP concerns (e.g., scaling and overflow/underflow). However, the major downside of FP butterfly is its slowness in comparison with its fixed-point counterpart. This reveals the incentive to develop a high-speed FP butterfly architecture to mitigate FP slowness. This brief proposes a fast FP butterfly unit using a devised FP fused-dot-product-add (FDPA) unit, to compute AB±CD±E, based on binary-signed-digit (BSD) representation. The FP three-operand BSD adder and the FP BSD constant multiplier are the constituents of the proposed FDPA unit. A carry-limited BSD adder is proposed and used in the three-operand adder and the parallel BSD multiplier so as to improve the speed of the FDPA unit. Moreover, modified Booth encoding is used to accelerate the BSD multiplier. The synthesis results show that the proposed FP butterfly architecture is much faster than previous counterparts but at the cost of more area.

Index Terms – Carry Select Adder (CSA) representation, butterfly unit, Fast Fourier Trans-form (FFT), Floating-Point (FP), three-operand addition.

I. INTRODUCTION

Fast Fourier transform (FFT) circuitry consists of several consecutive multipliers and adders over complex numbers; hence an appropriate number representation must be chosen wisely. Most of the FFT architectures have been using fixed-point arithmetic, until recently that FFTs based on floating-point (FP) operations grow The main advantage of FP over fixed-point arithmetic is the wide dynamic range it introduces; but at the expense of higher cost. Moreover, use of IEEE-754-2008 standard for FP arithmetic allows for an FFT coprocessor in collaboration with general purpose processors. This offloads compute-intensive tasks from the processors and leads to higher performance. The main drawback of the FP operations is their slowness in comparison with the fixed-point counterparts. A way to speed up the FP arithmetic is to merge several operations in a single FP unit, and hence save delay, area, and power consumption. Using redundant number systems is another well-known way of overcoming FP slowness, where there is no word-wide carry propagation within the intermediate operations

FP activities, their main drawback is slower when compared to fixed-point units. FP arithmetic as a way to speed up the integration of several operations in a single FP unit, and hence save the delay, area, and power consumption. Using redundant number systems, FP overcoming slowing the propagation of the carry over the intermediate term, there is no known way to work.

II.LITERATURE SURVEY

The Sequential Fast Fourier Transform:

The Discrete Fourier Transform is an operation performed on a series of elements to convert the underlying domain (e.g., time) to frequency, or vise- versa. The result has many useful applications and is one of the most widely used algorithms of the 20th and 21st centuries. The typical DFT operation performed on N elements x1; x2:::xN is denied as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}}$$

As a result of the input range of each element of the array X is an additive that requires the cooperation of every element of x. Thus, an N- DFT operation element in the input array O (N2) is. In the case of a large array of applications, compute the DFT of this difficult time. Therefore, DFT-time-complexity of the algorithm, it is desirable to reduce. Cooley and Tukey Fast Fourier algorithm introduced in 1965, to transform, to significantly reduce the complexity of computing discrete Fourier transform and conquer approach is the partition. FFT method, we have no input in the range of a DFT (NlogN) to be able to count. FFT for a very brief introduction to the original paper, but its effect is evident as soon as the numbered of turning around. Several documents shortly after being published in more detail and more FFT algorithm generalize. However, for many years after its introduction, the development of some of the technology, "Cooley-Tukey algorithm" was held, and the theoretical aspects of the re- search FFT algorithm, or FFT analysis focused on the details of the proposals.

Cooley-Tukey Algorithm

Cooley-Tukey algorithm, as introduced, accounting (both additions and multiplications) to calculate the discrete Fourier transform to reduce the need to use a divide and conquer approach. The Cooley-Tukey algorithm, outlined here, radix-2 D (decimation-in-time), and for a simple FFT algorithm serves as our base.

The basic steps of the algorithm

- Decimate - two (i.e. source 2) of DFT to create small, even or odd set the split of the original input.
- Multiply - Multiply each element of the source of the unity of the coalition (called twiddle factors).
- Butterfly - (see picture 1), the other with a small element of the corresponding add each element of each of the DFT.

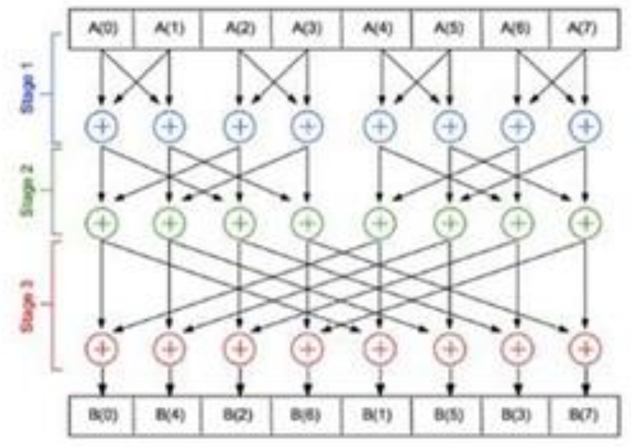


Fig.1. Recursive process of radix 2 DIT-FFT

Existing systems FFT

An FFT processor chip was originally designed. OFDM FFT processor is the focal point of both the transmitter and receiver. FFT high performance, combined with low energy consumption, such as high-throughput approach to the implementation of an ASIC is computationally demanding operation.

Architecture

The FFT and IFFT has the property that, if FFT

(Re(xi)+ jIm(xi)) = Re(Xi)+ jIm(Xi) and
IFFT(Re(Xi)+ jIm(Xi)) = Re(xi)+ jIm(xi)

Where xi and Xi are N words long sequences of complex valued, samples and sub-carriers respectively, then 1/N * FFT (Im(Xi)+ jRe(Xi)) = Im(xi)+ jRe(xi). Therefore, it is necessary not only to discuss the implementation of the FFT equalizer.

To calculate the inverse transform, the real and imaginary part of the input and output are changed. N 1 / n scaling a power of two, so that the right binary word log2 (N) bits in the same switch. Even simpler, binary point left log2 (N) bits is shifted to remember. If ever, did not show up to change a bit, depending on how it is used, the output from IFFT, is required.

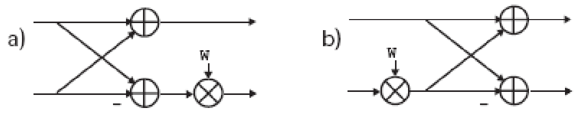


Figure 2: **A radix-2 DIF butterfly (a) and a radix-2 DIT butterfly (b), where W is the twiddle factor.**

FFT algorithm is the basic building block can be realized with a butterfly operation. Frequency (DIF) time, (d) and the decimation of the butterfly in the death of two types of operations, the two are shown in Figure 2. DIF is the difference between the before and after the addition or subtraction and multiplication of featured twiddle factor is in place. FFT based on the division to conquer and due to the input range is N R source, known as the point length N = RP, so, and p positive integer is the most effective. An N-point FFT, to count the butterflies are connected to the p stages.

The map is an example of a hardware-point radix- made N = 16 -2 DIF FFT is shown in Figure 2(a). The input data, x (N), the output of data occurs in a random order, however, X (N) to return to observe it. Reversed the order of the data generated to re-order bit is known and described. Figure 2 (b) FFT bit reversed order, which will result in the reshaping of the input and output, so it is possible to form the natural order. Figure 2(a), moving vertically, parallel data paths and reconfigure the product to the natural order, the way to control cross connections while, and think. As well as all the arrows in Figure 2 (b), if one turned around from the FFT, DIF FFT is performed instead.

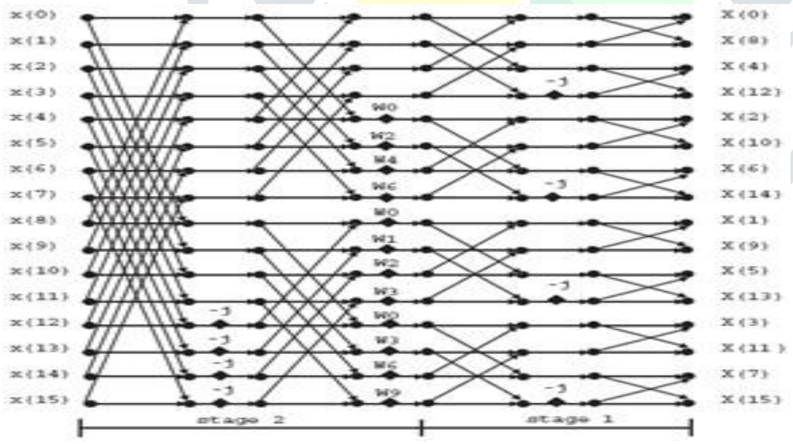


Figure 2.1: A hardware mapped N = 16-point radix-22 DIF FFT algorithm

III .PROPOSED BUTTERFLY ARCHITECTURE

This brief proposes a butterfly architecture using redundant FP arithmetic, which is useful for FP FFT coprocessors and contributes to digital signal processing applications. Although there are other works on the use of redundant FP number systems, they are not optimized for butterfly architecture in which both redundant FP multiplier and adder are required. The novelties and techniques used in the proposed design include the following

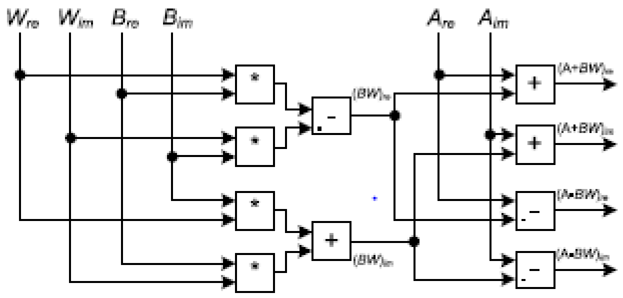
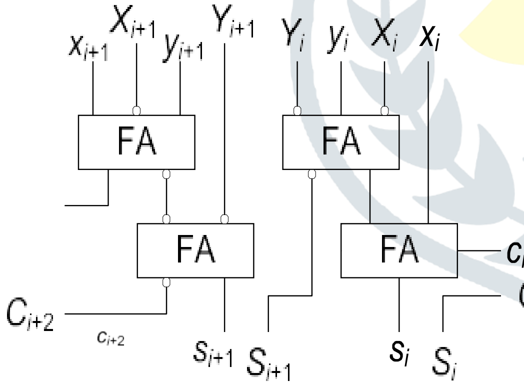


Fig.3.1 .FFT butterfly architecture with expanded complex numbers

1) All the significant are represented in binary signed- digit (BSD) format and the Corresponding carry-limited adder is designed.
2) Design of FP constant multipliers for operands with BSD significant.
3) Design of FP three-operand adders for operands with BSD significant.
4) Design of FP fused-dot-product-add (FDPA) units (i.e., AB ± CD ± E*)* for operands with BSD significant.

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. Continuing this decomposition leads to 2-input FFT block, also known as butterfly unit. The proposed butterfly unit is actually a complex fused-multiply–add with FP operands. Expanding the complex numbers, Fig.3.1 shows the required modules. According to Fig.3.1, the constituent operations for butterfly unit are a dot-product (e.g., *Bre Wim + Bim Wre)* followed by an addition/subtraction which leads to the proposed FDPA operation (e.g., *Bre Wim* + *Bim Wre* + *Aim).* Implementation details of FDPA, over FP operands.



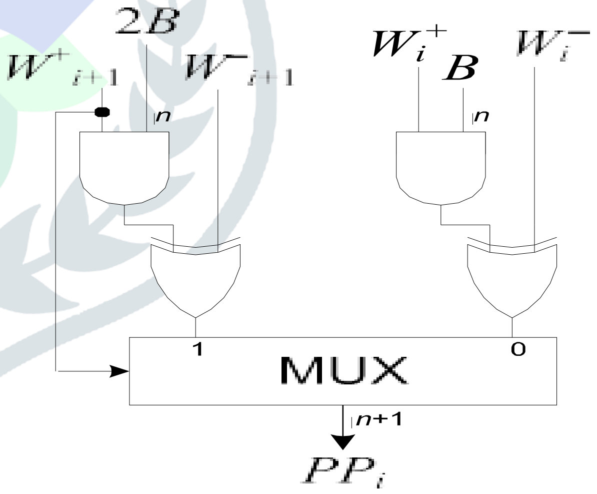


Figure.3.2. a. BSD adder (two-digit slice). Figure. 3.2 .b. Generation of the *i*th PP.

The exponents of all the inputs are assumed and represented in two's complement (after subtracting the bias), while the significant of *Are*, *Aim*, *Bre*, and *Bim* are represented in BSD. 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).

Proposed Redundant Floating-Point Multiplier

The proposed multiplier, likewise other parallel multipliers, consists of two major steps, namely, partial product generation (PPG) and PP reduction (PPR). 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^r, W^r).Moreover, given that W_{re} and W_{im} are constants, the multiplications in Fig.5 (over significant) can be computed through a series of shifters and adders.

Fig. 4. Digits to three-operand adder.

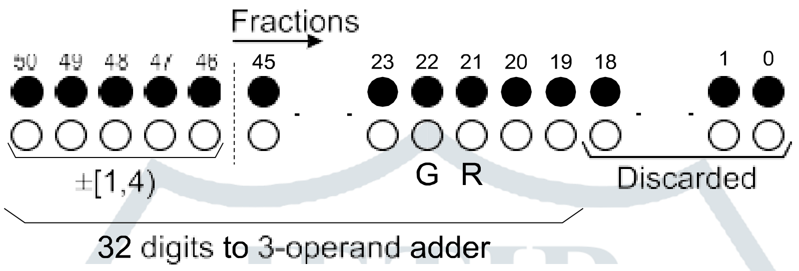
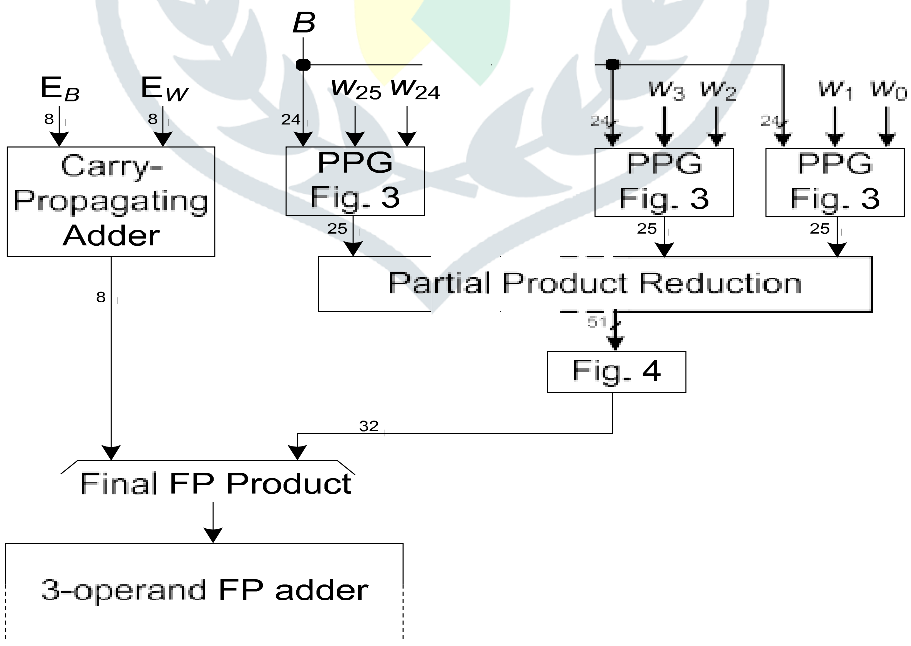


TABLE I

GENERATION OF A PP

$W_{i+1}^- W_{i+1}^+$	$W_i^- W_i^+$	$\|W_{i+1}^- W_{i+1}^+ W_i^- W_i^+\|$	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	$2 \times B$
1 1	0 0	-2	$-2 \times B$

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 $(n,\ldots,0)$ which is either B $(n-1,\ldots,0)$ or $2B$ $(n,\ldots,1)$, the



length of the final product may be more than $2n$.

Fig. 5. Proposed redundant FP multiplier.

Assuming that the sign-embedded significant 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 i±[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. 4 shows the required digits passed to the three-operand adder.
Fig.5 shows the proposed redundant FP multiplier.

Proposed Redundant Floating-Point Three-Operand Adder

The straightforward approach to perform a three-operand FP addition is to concatenate two FP adders which leads to high 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. 3.2). Moreover, sign logic is eliminated.

The bigger exponent between E_X and E_Y (called E_{Big}) is deter- mined using a binary subtractor ($6 = E_X - E_Y$); and the significand of the operand with smaller exponent (X or Y) is shifted "6"-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 $6_A = E_{\text{Big}} - E_A + 30$ is computed which shows the amount of right shifts required to be performed on extended (with the initial position of 30 digits shifted to left). Fig. 6 shows the alignment implemented in the proposed three-operand FP adder. Next, a BSD adder adds the aligned third significand (58-digit) to SUM (33-digit) generated from the first BSD adder.

Since the input operands have different number of digits, this adder is a simplified 58-digit BSD adder. 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.

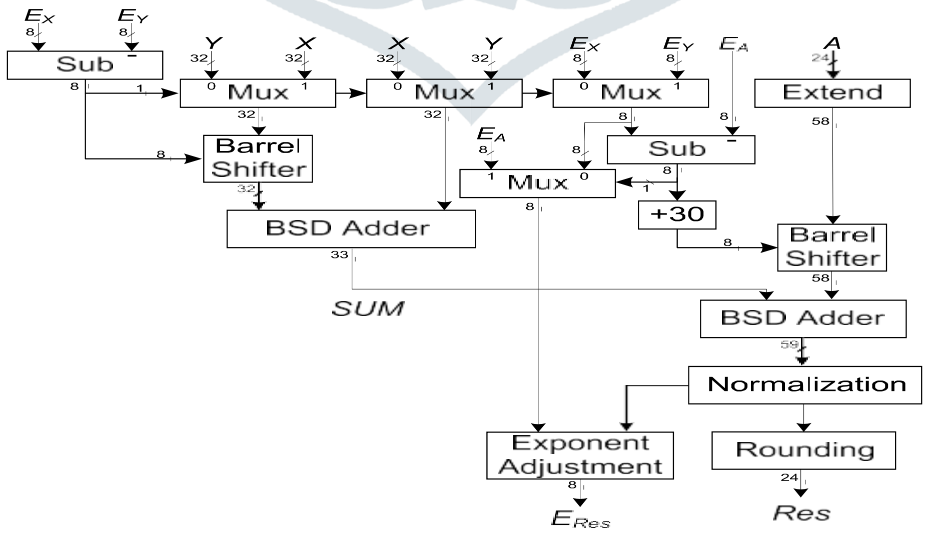
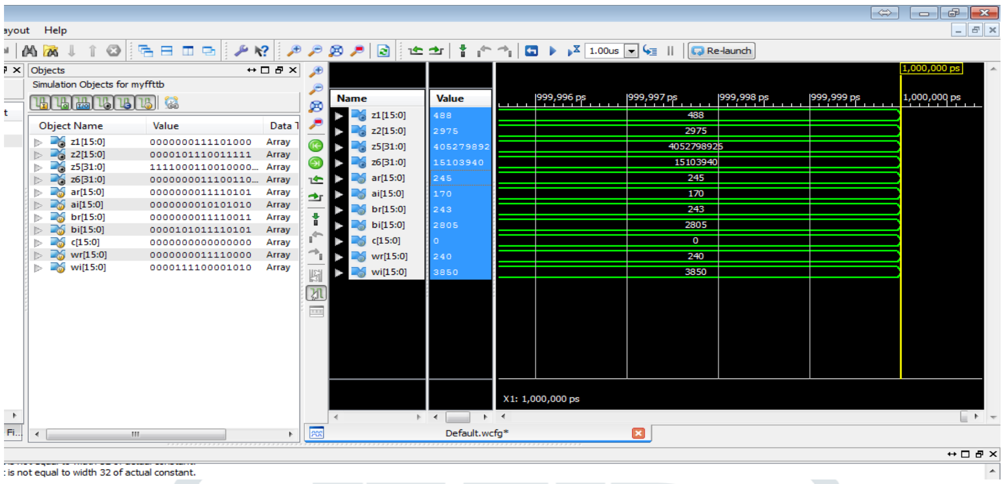


Fig. 6. Proposed FP three-operand addition

IV Simulation results



Simulation Objects for myffttb

Object Name	Value	Data T
z1[15:0]	0000000111101000	Array
z2[15:0]	0000101110011111	Array
z5[31:0]	1111000110010000...	Array
z6[31:0]	0000000011100110...	Array
ar[15:0]	0000000011110101	Array
ai[15:0]	0000000010101010	Array
br[15:0]	0000000011110011	Array
bi[15:0]	0000101011110101	Array
c[15:0]	0000000000000000	Array
wr[15:0]	0000000011110000	Array
wi[15:0]	0000111100001010	Array

Name	Value
z1[15:0]	488
z2[15:0]	2975
z5[31:0]	405279892
z6[31:0]	15103940
ar[15:0]	245
ai[15:0]	170
br[15:0]	243
bi[15:0]	2805
c[15:0]	0
wr[15:0]	240
wi[15:0]	3850

: is not equal to width 32 of actual constant.

Fig.7. simulation result

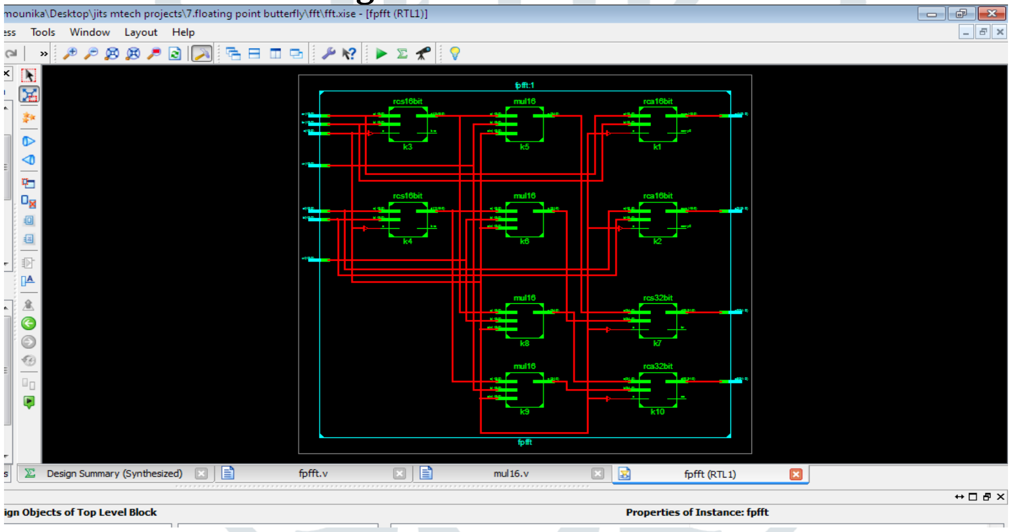

mounika\Desktop\jits mtech projects\7.floating point butterfly\fft\fft.xise - [fpfft (RTL1)]
Tools
Window
Layout
Help
fpfft:1
rca16bit
mul16
k3
k5
k1
k4
k6
k2
k8
k7
k9
k10
rca32bit
fpfft
Design Summary (Synthesized)
fpfft.v
mul16.v
fpfft (RTL1)
ign Objects of Top Level Block
Properties of Instance: fpfft

Fig.8. Schematic of proposed method

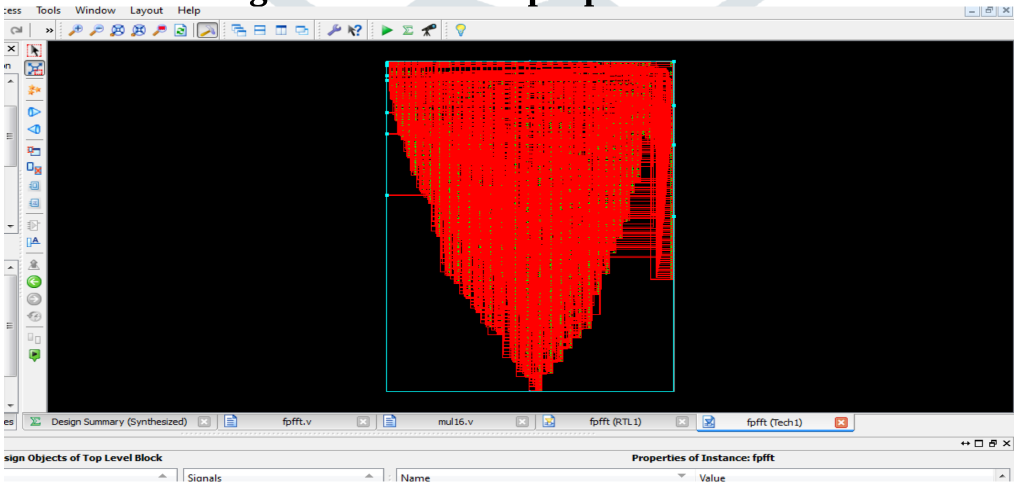

Tools
Window
Layout
Help
Design Summary (Synthesized)
fpfft.v
mul16.v
fpfft (RTL1)
fpfft (Tech1)
sign Objects of Top Level Block
Properties of Instance: fpfft
Signals
Name
Value

Fig 8.Technology Schematic of proposed method

V Conclusion

In this paper we have here in the processor FFT and FFT butterfly structure, reading, writing and execution addresses. But the high-risk area, which is faster than the previous works, we have a high-speed FP butterfly structure, proposed. The reason for this is the speed of development is twofold: it eliminates the carry propagation significands 1) BSD represented, and 2) the abbreviation of the proposed new FDPA unit. FP butterfly multiplications and additions need to be combined with the unit; Thus, high-speed additional LZD, normalize, remove, and can be achieved by rounding units. The next research FP adder three dual line method applied to the structure and other unnecessary FP. FP representations may be planning on using. Moreover, the design stage of the abolition of the use of improved techniques (ie, repeating LZD, normalize, and rounding) of the estimated costs, however, led to faster architectures.

References:

[1] E. E. Swartzlander, Jr., and H. H. Saleh, “FFT implementation withfused floating-point operations,” IEEE Trans. Comput., vol. 61, no. 2,pp. 284–288, Feb. 2012.
[2] J. Sohn and E. E. Swartzlander, Jr., “Improved architectures for afloating-point fused dot product unit,” in Proc. IEEE 21st Symp. Comput.Arithmetic, Apr. 2013, pp. 41–48.
[3] IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754-2008,Aug. 2008, pp. 1–58.
[4] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs,2nd ed. New York, NY, USA: Oxford Univ. Press, 2010.
[5] J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculationof complex Fourier series,” Math. Comput., vol. 19, no. 90, pp. 297–301,Apr. 1965.
[6] A. F. Tenca, “Multi-operand floating-point addition,” in Proc. 19th IEEESymp. Comput.
Arithmetic, Jun. 2009, pp. 161–168.
[7] Y. Tao, G. Deyuan, F. Xiaoya, and R. Xianglong, “Three-operand floating-point adder,” in Proc. 12th IEEE Int. Conf. Comput. Inf. Technol.,Oct. 2012, pp. 192–196.
[8] A. M. Nielsen, D. W. Matula, C. N. Lyu, and G. Even, “An IEEE compliant floating-point adder that conforms with the pipeline packet forwarding paradigm,” IEEE Trans. Comput., vol. 49, no. 1, pp. 33–47,Jan. 2000.
[9] P. Kornerup, “Correcting the normalization shift of redundant binary representations,” IEEE Trans. Comput., vol. 58, no. 10, pp. 1435–1439,Oct. 2009.
[10] 90 nm CMOS090 Design Platform, STMicroelectronics, Geneva, Switzerland, 2007.
[11] J. H. Min, S.-W. Kim, and E. E. Swartzlander, Jr., “A floating-point fused FFT butterfly arithmetic unit with merged multiple-constant multipliers,” in Proc. 45th Asilomar Conf. Signals, Syst. Comput., Nov