FFT REPRESENTATION USING FLOATING-POINT BUTTERFLY ARCHITECTURE

[1]Mr.S. BHARATH KUMAR, [2] Mr .P. VENKATESWARLU, [3]Dr. Mr. ARVIND KUNDU
[1]PG Student, [2]Assistant professor, [3]Associate Professor
[1]Department of ECE, [1]SCIENT INSTITUTE OF TECHNOLOGY, HYDERABAD

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. 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. In this we can implement the FFT butterfly using FP butterfly unit due to this we can save the loss of data compared to the normal FFT algorithm. This is also a one of application of FP butterfly arithmetic. The synthesis results show that the proposed FP butterfly architecture is much faster than previous counterparts are but at the cost of more area.

System Configuration:-

In the hardware part a normal computer where Xilinx ISE 14.3 software can be easily operated is required, i.e., with a minimum system configuration

Key words: FFT, FDPA, BSD

I.INTRODUCTION

Joseph Fourier in 1811 in a paper presented to the French Academy of Sciences. As soon as the paper is published, the Fourier effect and un- benownst, yet many problem domains, which would have a lasting impact on a range. Fourier presented by the magazine in its original domain (eg, time) equally distributed samples to the frequency domain as a method of transforming a set _nite Fourier transform concept was introduced. Apply the mathematical concept of a set of complex numbers and Fourier in_nite the transition does not involve integration. This is a continuing type of Fourier transform has many applications in physics and engineering, but it is the form of a discrete Fourier transform of the computer systems that can easily [53] can be implemented. The discrete Fourier transform (DFT) and its applications in computer science in 1965 (FFT) to transform the James Cooley and John Tukey [18] led to the fast Fourier. Divide and conquer using the method, FFT O (N2) O (Nlog (N)) also reduces the computational complexity of the DFT. When such signi_cant computational complexity, FFT Fourier transform many di_cult problems were a convenient solution. Turning around the areas of digital signal pro cessing and the number of FFT knots, which in the 20th century, [22] is considered to be one of the most inuential algorithms.

This is invaluable tools in the implementation of high performance systems, combining the re programmability advantage of general Purpose processors with the speed and parallel processing. At some point, require general purpose arithmetic processing units which are not standard components of FPGA devices. More recently, the increasing size of FPGA devices allowed researchers too efficiently Implement operators in the 32-bit single Precision format .Single precision format, the most basic format of the ANSI/IEEE 754-1985 binary floating-point arithmetic standard. Double precision and quad precision described more bit operation so at same time we perform 32 and 64 bit of operation of arithmetic unit. Floating point includes single precision, double precision and quad precision floating point format representation and provide implementation technique of various arithmetic calculation. Normalization and alignment are useful for operation and floating point number should be normalized before any calculation.

Floating point arithmetic implementation described various arithmetic operations like addition, subtraction, multiplication, division. In science computation this circuit is useful. A general purpose arithmetic unit require for all operations. In this paper single precision floating point arithmetic addition and subtraction is described. This paper includes single precision floating point format representation and provides implementation technique of addition and subtraction arithmetic calculations. Low precision custom format is useful for reduce the associated circuit costs and increasing their speed. I consider the implementation of 32 bit addition and subtraction unit for floating point arithmetic.

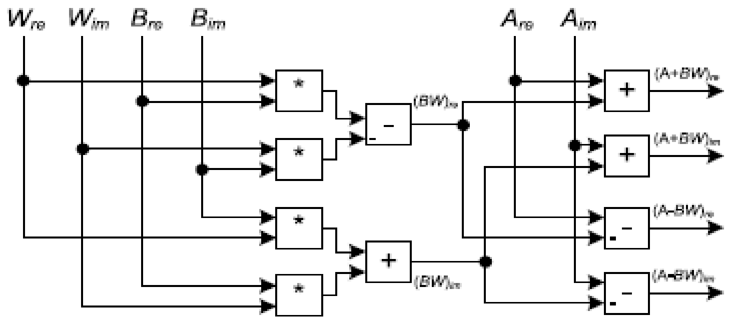


Fig1: FFT butterfly architecture with expanded complex numbers.

The conversion, from non-redundant, to a redundant format is a carry-free operation, however, the reverse conversion requires carry propagation. This makes redundant representation more useful where many consecutive arithmetic operations are performed prior to the final result.

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.

1) All the significands 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 significands.

3) Design of FP three-operand adders for operands with BSD significands.

4) Design of FP fused-dot-product-add (FDPA) units for operands with BSD significands.

II.Floating point arithmetic

Floating point numbers are one possible way of representing real numbers in binary format; the IEEE 754 [1] standard presents two different floating point formats, Binary interchange format and Decimal interchange format. Multiplying floating point numbers is a critical requirement for DSP applications involving large dynamic range. This paper focuses only on single precision normalized binary interchange format. Fig. 1 shows the IEEE 754 single precision binary format representation; it consists of a one bit sign (S), an eight bit exponent (E), and a twenty three bit fraction (M or Mantissa). An extra bit is added to the fraction to form what is called the significand1. If the exponent is greater than 0 and smaller than 255, and there is 1 in the MSB of the significand then the number is said to be a normalized number; in this case the real number is represented by (1)

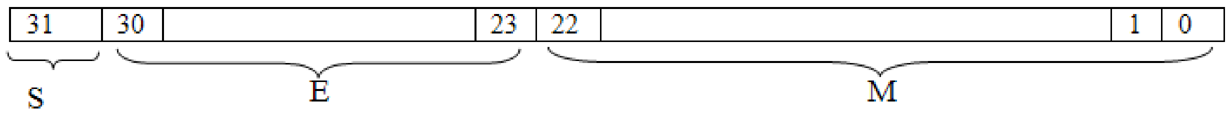


Figure 2. IEEE single precision floating point format

Z = (-1S) * 2 (E - *Bias*) * (1.M)__

Where M = m22 2-1 + m21 2-2 + m20 2-3+…+ m1 2-22+ m0 2-23;

Bias = 127.

Multiplying two numbers in floating point format is done by 1- adding the exponent of the two numbers then subtracting the bias from their result, 2- multiplying the significand of the two numbers, and 3- calculating the sign by XOR ing the sign of the two numbers. In order to represent the multiplication result as a normalized number there should be 1 in the MSB of the result (leading one). Floating-point implementation on FPGAs has been the interest of many researchers. In, an IEEE 754 single precision pipelined floating point multiplier was implemented on multiple FPGAs. In , a custom 16/18 bit three stage pipelined floating point multiplier that doesn't support rounding modes was implemented. In, a single precision floating point multiplier that doesn't support rounding modes was implemented using a digit-serial.

UNDERFLOW/OVERFLOW DETECTION:

Overflow/underflow means that the result's exponent is too large/small to be represented in the exponent field. The exponent of the result must be 8 bits in size, and must be between 1 and 254 otherwise the value is not a normalized one.

An overflow may occur while adding the two exponents or during normalization. Overflow due to exponent addition may be compensated during subtraction of the bias; resulting in a normal output value (normal operation). An underflow may occur while subtracting the bias to form the intermediate exponent. If the intermediate exponent < 0 then it's an underflow that can never be compensated; if the intermediate exponent = 0 then it's an underflow that may be compensated during normalization by adding 1 to it. When an overflow occurs an overflow flag signal goes high and the result turns to ±Infinity (sign determined according to the sign of the floating point multiplier inputs). When an underflow occurs an underflow flag signal goes high and the result turns to ±Zero (sign determined according to the sign of the floating point multiplier inputs). De normalized numbers are signaled to Zero with the appropriate sign calculated from the inputs and an underflow flag is raised. Assume that E1 and E2 are the exponents of the two numbers A and B respectively;

the result's exponent is calculated

Eresult = E1 + E2 - 127

E1 and E2 can have the values from 1 to 254; resulting in

Eresult having values from -125 (2-127) to 381 (508-127); but for normalized numbers, Eresult can only have the values from 1 to 254.

2.1Addition/Subtraction

If two numbers x and y which described floating point numbers and we perform calculation sum or difference *of* two numbers. Firstly, we check for zero of two values and then require next step calculate the difference of the two exponents, so need operation of alignment. Align the significand *Ex–Ey =0* and put exponent *ER* is result of the two exponents. Now add or subtract the two significands *Ex* and *Ey*, according to the effective operation. Normalize the result *SR*, adjusting *ER* as appropriate. Step by step we can perform addition and subtraction of two floating point numbers. In the first step, the floating point operands *x* and *y* are unpacked and checks for zero, infinity. If we can assume that neither operand is infinity.

The relation between the two operands *x* and *y* is determined based on the relation between *Ex* and *Ey* and by comparing the significands *sx* and *sy*, which is required if *Ex*=*Ey*. Swapping *Sx* and *Sy* is equivalent to swapping *x* and *y* and making an adjustment to the sign *sr*. this swapping requires only multiplexers. In the next step, alignment of two significand used. The significand alignment shift is performed and the effective operation is carried out. This step useful for equal the significand numbers.

Alignment achieved by shifting either smaller number to right. Now we can say that increasing it exponent .if alignment has been completed then we go for next step normalization. Alignment is useful for next step of normalization. Without this step result not be calculated. Normalization is last step for floating point arithmetic operation. *Sr* is normalized by after the alignment and shifter, which can perform by right shifts. If *sr* is normalized, then it shows result of two floating point number of addition and subtraction.

Addition and subtraction of two floating point number described in following block diagram. Step by step we perform calculation of two floating point number addition and subtraction. Firstly if we taken two floating point number for addition and subtraction then in starting both number will be unpack.

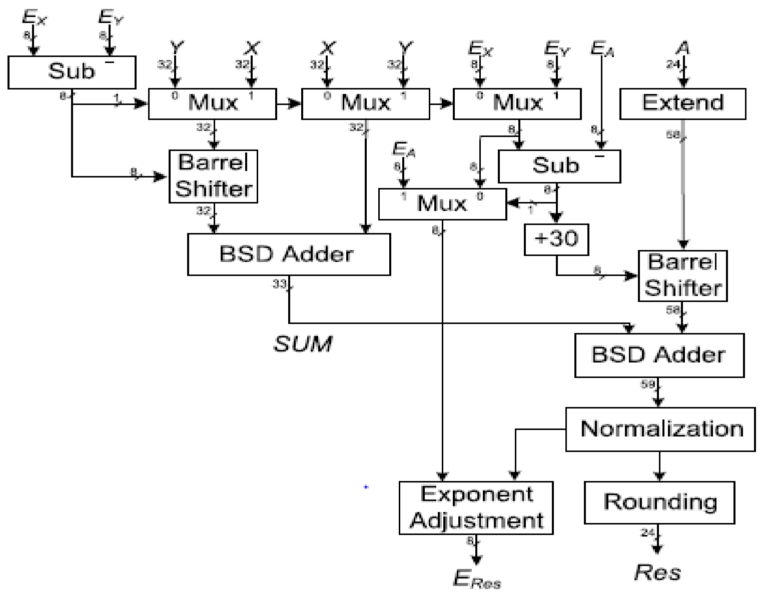


Fig3: Proposed FP three-operand addition.

After that we identify sign, exponent and significand of both numbers. This process depends on single precision and double precision and quad precision format. According to this format we taken sign, exponent and significand bit. The IEEE standard specifies that format. Normalization process will be completed after specify IEEE standard. In this step normalize the number means move binary point so that it is one bit from left. Shifting the mantissa left by one bit decrease the exponent by one or shifting the mantissa right by one bit increases the exponent by one. Before subtracting, compare magnitude and change sign bit if order of operands is changed .Alignment can be done after that process and finally we get result of addition and subtraction of two floating point numbers.

2.2 DFT&FFT

Fast Fourier Transform (FFT) is one of the most efficient ways to implement Discrete Fourier Transform (DFT) due to its reduced usage of arithmetic units. DFT is one of those primary tools that are used for the frequency analysis of discrete time signals and to represent a discrete time sequence in frequency domain using its spectrum samples. The analysis (forward) and synthesis (inverse) equations of an N point FFT are given below.

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{kn}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k]\, W_N^{-kn}$$

Where,$W_N{}^{kn} = e^{-j2\pi}{}_{/N}{}^{kn}$ As evident from equation (2.1.1), the basis of both synthesis and analysis equations remains same thus increasing the scope of the architecture to both analyze and synthesize. Due to increased employability of FFT in modern electronic systems, higher radix FFTs such as radix – 4, radix – 8, radix –2^k, split radix, etc. are designed for improved timing and reduced hardware. The basic difference of the mentioned methods lies in the structure of their butterfly units.

2.2.1The Discrete Fourier Transform (DFT):

Transforms such as discrete fourier transform (DFT) are a major block in many communication systems like OFDM, etc. DFT is also considered as one of the major tools to perform frequency analysis of discrete time signals. A discrete time sequence can be represented by samples of its spectrum in the frequency domain, using DFT.

2.3 Radix – 4 FFT algorithms:

When the number of data points N in the DFT is a power of 4 (i.e; $N = 4^v$) , and it is more efficient to employ a radix -4 algorithm.

Now describe the radix -4 decimation – in – time FFT algorithm, now we selecting L= 4 and M= N/4 in the divide – and – conquer approach describe in section (2.2). now for this region the choice of L and M. we have l , p = 0,1,2,3; m , q= 0,1,2,….N/4 – 1;n = 4m + l; and k= (N/4)p + q. the decimate the N point input sequence into four sub-sequences, x(4n), x(4n +1) ,x(4n + 2) and x(4n + 3), n= 0,1,……N/4.

The decimation – in time procedure can be can be repeated recursively v times. Hence the FFT algorithm consist of v stage , where each stage contain N/4 butterflies. The algorihm is

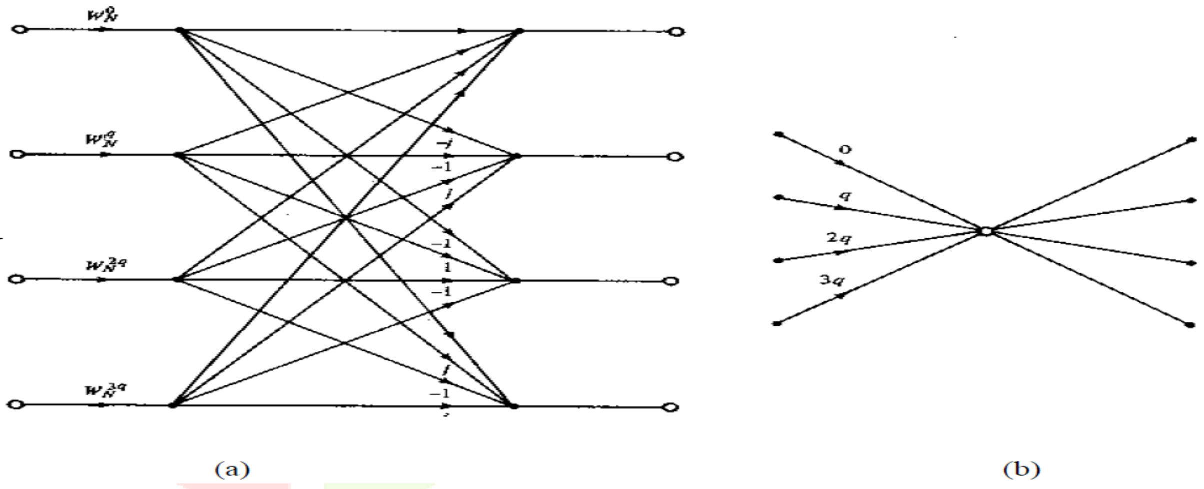


Figure4: Basic butterfly computation in a radix – 4 FFT algorithm

Note that X(p,q) = X(4p + q) , q = 0,1,2,3. The N – point DFT is decimated into four N/4 point DFT and we have a decimation – in – frequency algorithm. The computation equation define the basic radix – 4 butterfly for the decimation – in – frequency algorithm.

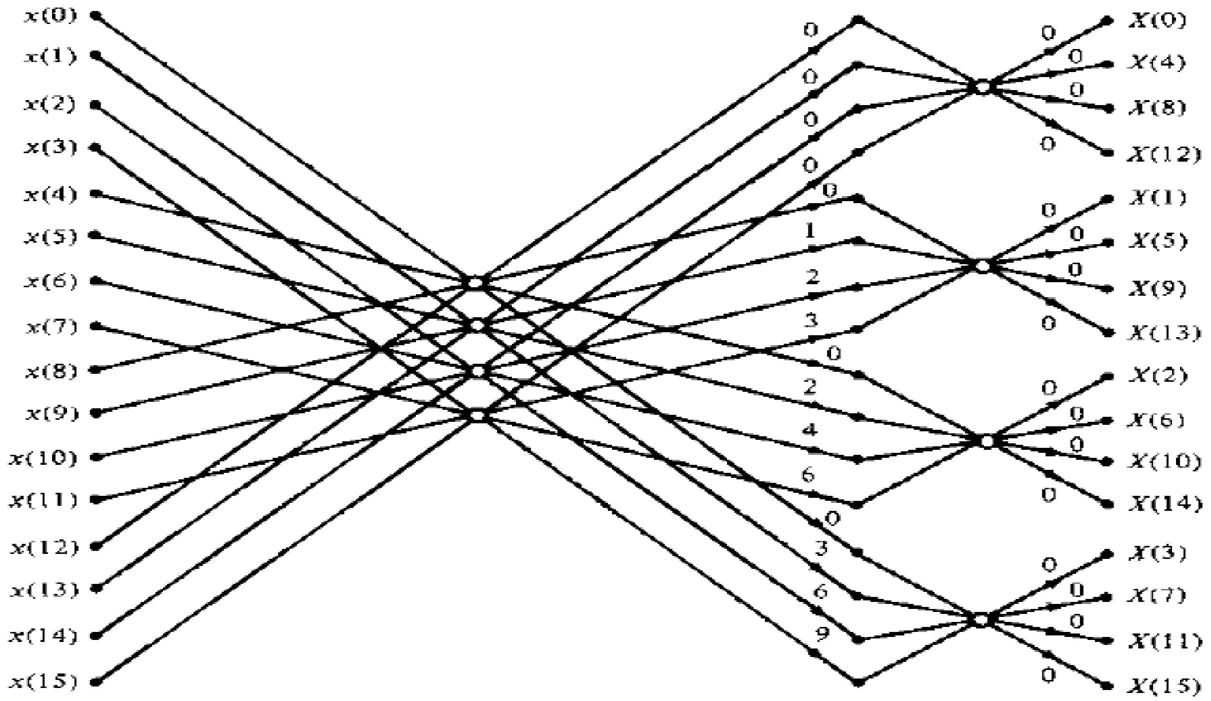


Figure5: Sixteen- point, radix – 4 decimation – in – frequency algorithm with input in normal order and output in digit – reversed order.

A 16 point radix – 4 decimation – in – frequency FFT algorithm shown in figure (2.2). the inputs is in normal order but the outputs comes in digit reversed order. It has exactly same as computational complexity as the decimation – in frequency algorithm.

III.PROPOSED MODEL

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).

3.1 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. (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*,

3.2 ***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. (two-digit slice).

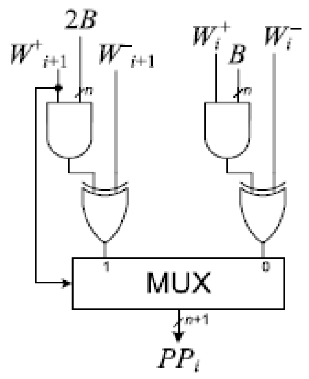


Fig6: Generation of PP

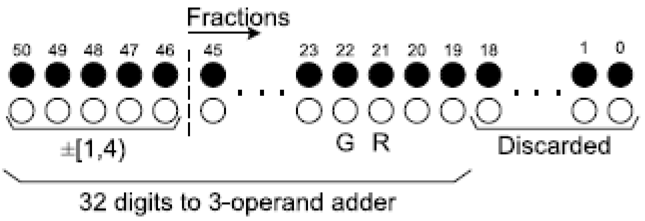


Fig7: Digits to three-operand adder.

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.

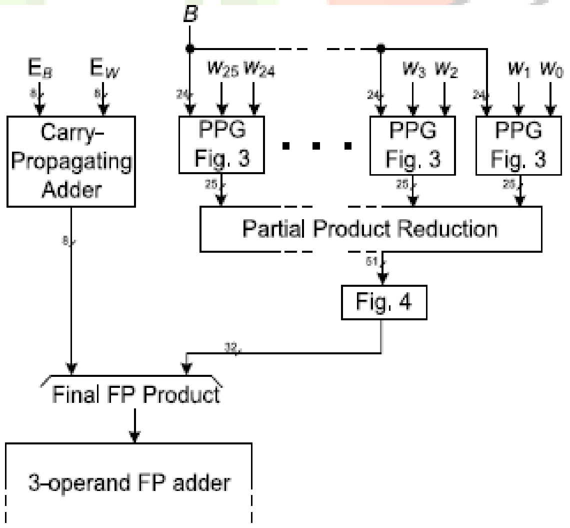


Fig8: Proposed redundant FP multiplier

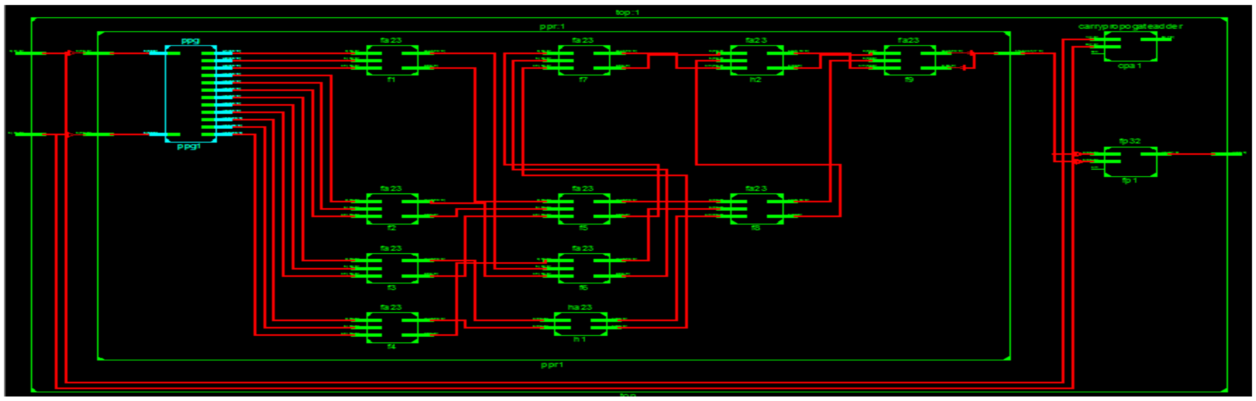


Fig9: RTL Schematic for Floating point butterfly representation

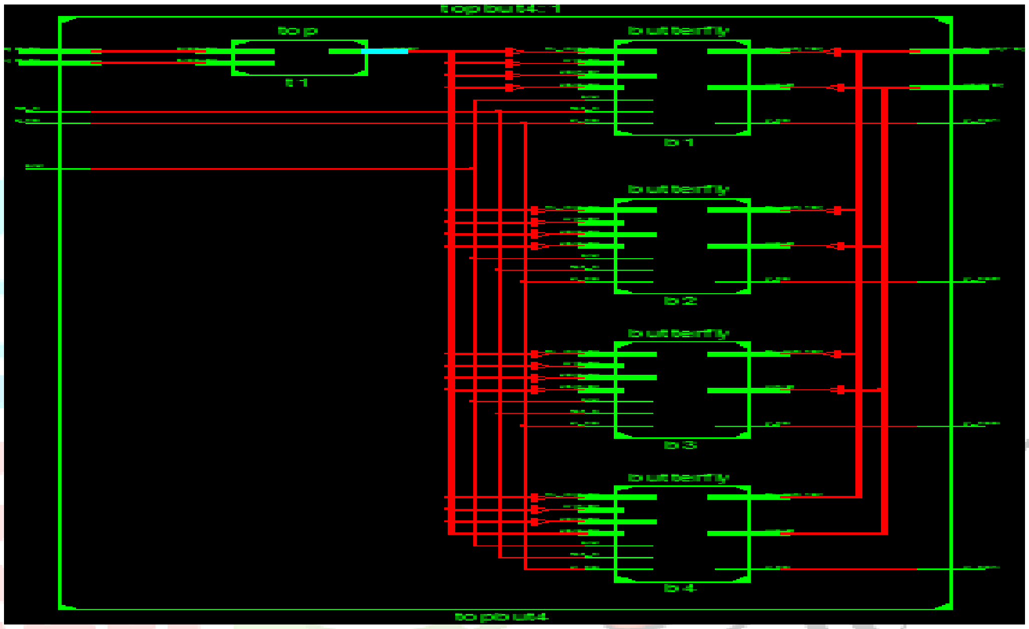
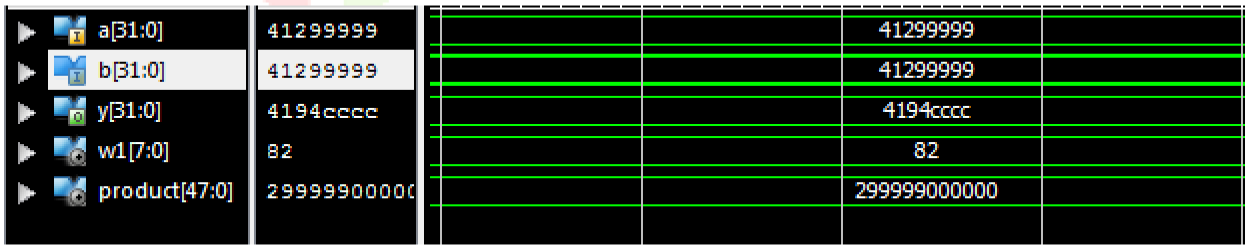


Fig10: RTL Schematic for FFT using Floating point butterfly representation

IV.RESULTS



```
a[31:0]         41299999       41299999
b[31:0]         41299999       41299999
y[31:0]         4194cccc       4194cccc
w1[7:0]         82             82
product[47:0]   29999900000(   299999000000
```

Fig11: simulation Results for Floating point butterfly representation

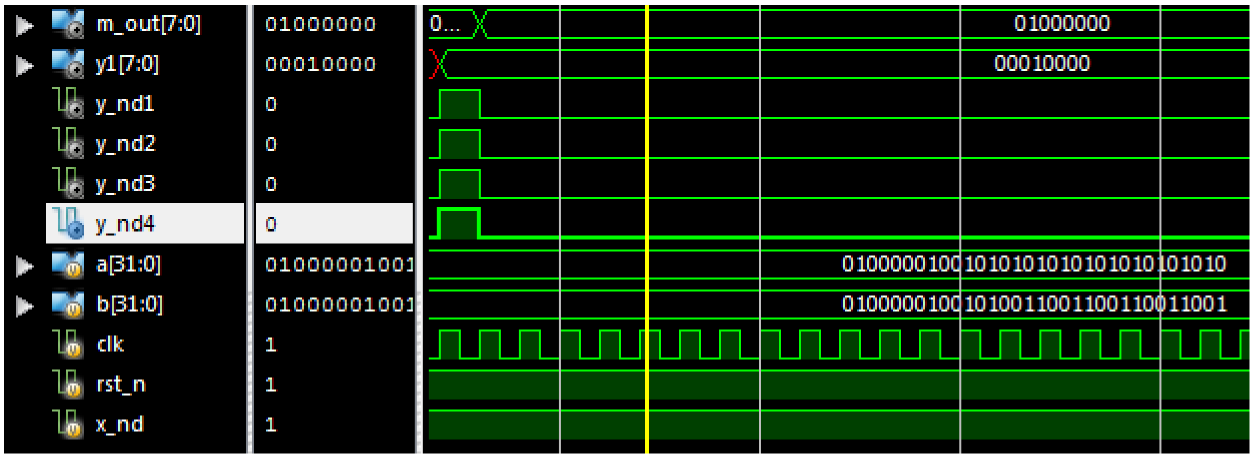


Fig12: simulation Results for FFT using Floating point butterfly representation

V.CONCLUSION

We proposed a high-speed FP butterfly architecture, which is faster than previous works but at the cost of higher area. The reason for this speed improvement is twofold: 1) BSD representation of the significands which eliminates carry-propagation and 2) the new FDPA unit proposed in this brief. This unit combines multiplications and additions required in FP butterfly; thus higher speed is achieved by eliminating extra LZD, normalization, and rounding units. Further research may be envisaged on applying dual-path FP architecture to the three-operand FP adder and using other redundant FP representations. Moreover, use of improved techniques in the termination phase of the design (i.e., redundant LZD, normalization, and rounding) would lead to faster architectures, though higher area costs are expected.

REFERENCES

[1] 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.
[2] 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.
[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 calculation of 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 IEEE Symp. 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 packetforwarding 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. 2011, pp. 520–524.

Author's Details:

Mr.S. BHARATH KUMAR, He received B.Tech degree in ECE Deparment from JNTUH. Presently He is pursuing M.TECH in BRANCH of VLSI&ES at SCIENT INSTITUTE OF TECHNOLOGY, IBRAHIMPATNAM.

Mr. P Venkateswarlu, He received B.Tech From JNTU Kakinada in Electronics and Communication Engineering. He did M.Tech from Andhra University in Communication System. He is working as Assistant Professor in SCIENT Institute of technology.His area of research in Microstrip Patch antenna and Data encoding techniques.

Dr. Arvind Kundu, he did B. Tech from H.P. University (SHIMLA) in Electronics & Communication. He did M. Tech from M.D. University (ROHTAK) in Electronics & Communication Engineering. He did Ph. D from Ranchi University and area of research is ADHOC Networks, EMBEDDED System, Cryptography, Message authentication Protocol, Image Processing, Routing protocol etc. He is working as HOD ECE Department at SCIENT INSTITUTE OF TECHNOLOGY, IBRAHIMPATNAM.