ASIAN JOURNAL OF PHARMACEUTICAL AND CLINICAL RESEARCH

INNOVARE ACADEMIC SCIENCES
Knowledge to Innovation

Special Issue (April)

Online - 2455-3891
Print - 0974-2441

Full Proceeding Paper

EFFICIENT FLOATING POINT FAST FOURIER TRANSFORM BUTTERFLY ARCHITECTURE USING BINARY SIGNED DIGIT MULTIPLIER AND ADDERS

SHIVANI ACHARYA, AUGUSTA SOPHY BEULET P

Department of CSE, School of Electronics, VIT, Chennai, Tamil Nadu, India. Email:asha.s@vit.ac.in

Received: 23 January 2017, Revised and Accepted: 03 March 2017

ABSTRACT

Fast Fourier transform (FFT) is one of the most important tools in digital signal processing as well as communication system because transforming time domain to S-plane is very convenient using FFT. As FFT uses various techniques to convert a signal from time domain to S-domain and inverse, out of which butterfly technique is the one on which paper is focused on. Butterfly technique uses additions and multiplications of operands to get the required output. Floating point (FP) is used as operands due to their flexibility. As the computations involving FP has less speed, we have used binary signed digit (BSD). BSD will take the less time for addition and subtraction. Three bit BSD adder and FP adder together will make a fused dot product add (FDPA) unit. In FDPA, unit addition and subtraction will be one group and multiplication will be one group and then their respective results will be fused. Modified booth encoding and decoding algorithm are used here to make the complex multiplication with ease.

Keywords: Binary signed digit, Floating point, Fused dot product add, Fast Fourier transform, Redundant number system.

 DOI: http://dx.doi.org/10.22159/ajpcr.2017.v10s1.19568

INTRODUCTION

Fast Fourier transform (FFT) is used to convert time domain signal into S-plane with high speed and efficiency. FFT using butterfly method is one of the easiest methods contains successive additions, subtraction, and multiplication for complex numbers. Hence, this method can be used as a tool to compute arithmetic in efficient way and in less time. According to Cooley and Tukey [4], while calculating the FFT using butterfly method, we can distribute the samples in even and odd. The main advantage of using FFT is that it have symmetry and periodicity property, which is very important aspect from computer arithmetic. This architecture makes use of floating point (FP) numbers as FP has dynamic range and also the memory space which they use is comparatively low. According to IEEE standard 754-2008 [3], it is used for FP arithmetic operations. The FP along with FFT butterfly.

Architecture contributes as FFT coprocessors. As the name suggest coprocessors, which mean they can be used along with the main coprocessors to reduce the workload on main processors. However, the main problem is that FP has its slowness in operations to reduce this drawback redundant number system (RNS) can be used. The main features of RNS are less delay, area, power consumption [2]. RNS can be defined as a system a number system, defined by radix r and digit set [α,β], is redundant if and only if β-α+1> r. This RNS used for reducing the delay by reducing the operations on carry propagation. For consecutive addition and subtraction, we can make use of RNS. In this paper, we have used RNS in the range of {−1,0,1}. Hence, it is names as binary signed digit (BSD) representation. Carry propagation is known as the main decelerator in digital arithmetic operations. Fused dot product adds (FDPA) unit consists of FP three-operand BSD adder and FP BSD constant multiplier.

The proposed methodology for these techniques used in this paper is as follows:

1. To convert significands of FP into BSD and carry limited adder is to be designed
2. FP constant multipliers for operands with BSD significands
3. FP three-operand adders for operands with BSD significands and Design FDPA for operands with BSD significands.

METHODS

Considering the signal having N samples, it has to find the FFT of signal. Then, according to Swartzerlandera and Saleh [1], N samples can be divided into even and odd, i.e., into N/2-input signals. This methodology is called as butterfly unit. From Fig. 1 A and B are the two samples where A complex number consists of one real and one imaginary component such that $A=A_{re}+A_{im}$ and $B=B_{re}+B_{im}$ and in calculating FFT the important is twiddled factor, which is defined as W^{nk}_{N}, where W can be represented as complex number $W=W_{re}+W_{im}$. According to Fig. 1 B is multiplied with W, on multiplying with B with W,

$$BW_{re}=B_{re}*W_{re}-B_{im}*W_{im} \quad (1)$$

$$BW_{im}=B_{re}*W_{im}+B_{im}*W_{re} \quad (2)$$

From equation 1 and 2, it shows that for getting BW as a output, dot product and successive addition and subtraction. And then, this product is going to get added or subtracted with A. For this operation, FP is used and to reduce its slowness BSD conversion is used. The every significand of FP point is converted into BSD having a range from {−1,0,1}. Extending the concept of FP addition and multiplication the one fusion is created of the outputs at each stage. The FDPA units provide two outputs (dubbed – and +). The output computes $B_{re}W_{im}+B_{im}W_{re}-A_{im}$ and $B_{re}W_{im}+B_{im}W_{re}+A_{im}$ while the + output is the result of an FDPA consists of two main components are:

A. Redundant FP multiplier
B. Three operand FP BSD.

Redundant FP multiplier

In this circuit, parallel multipliers are used. Since this parallel multipliers are used for fast multiplication, parallel multipliers can be implemented using Modified Booth multipliers [10] Modified Booth multiplier encodes multipliers and then given as an input to multiplicand and then given to Wallace tree and carry look ahead adder. The two major steps of parallel multipliers are:

Partial product generation (PPG)

In the Modified Booth encoder, the multiplier is encoded using algorithm, considering the binary stream of bits, as the 1 encounters

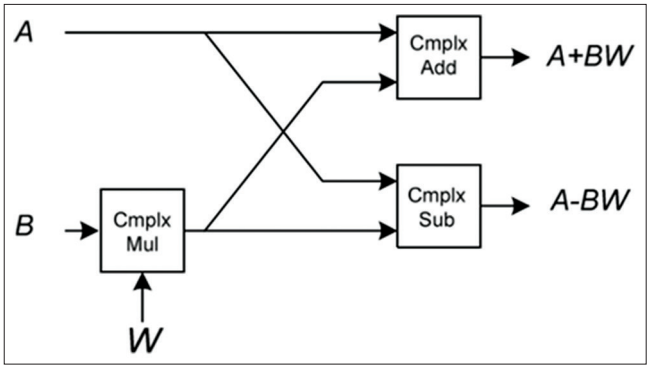

Fig. 1: Fast Fourier transform butterfly unit

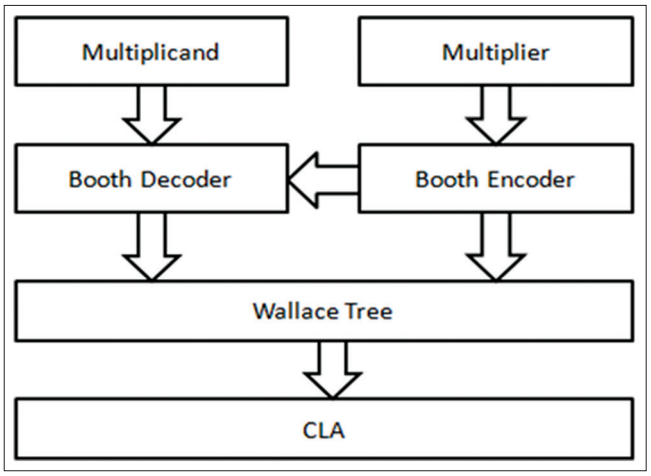

Fig. 2: Modified Booth algorithm

replacing it with −1 and after that if 0 comes in way replacing it with +1 And then making the pairs of two bits of multipliers and replacing bits.

Modified Booth algorithm uses n/2 clocks instead of n clocks for n bit multiplication which reduces the speed of operation. The Modified Booth encoding (MBE) is actually a radix-4 multiplication i.e., conversion from [0,3] to [−2,2]. The significands B_{re}, B_{im}, W_{re}, W_{im} are in BSD format. Multiplication of significands has three steps namely PPG, partial product reduction (PPR) and final addition [2]. Out of which PPG and PPR are important and time consuming, however the proposed multiplier keeps the product in redundant format so no need of carry propagating adder. The PPG, in a 2's complement representation of the multiplicand and multiplier, consists of arrays of AND operation such that each bit of the multiplier is ANDed to the whole bits of the multiplicand. This is not the case if the operands are represented in redundant format and/or Booth encoding. For example, if the multiplier is represented in the MBE, the PPG looks like the circuitry shown in Fig. 3. PPG of a redundant multiplier is even more complicated, since the cardinality of the multiplier's digit-set is more than the radix. Generating the multiples of the multiplicand is easy (shift and negation) for ±2x and ±1x; however, ±3x and ±5x (if exists) involve an addition. 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_{re}, W_{re}). The multiplications over significands can be computed via a series of shifters and adders. Most probably the barrel shifter is used for shifting. With the intention of reducing the number of adders [2], the significand of W is stored in MBE [7] in which every binary position takes a value of [−1,0,1] where there is at least one 0 in two adjacent positions. Therefore, n/2 add/sub is sufficient to compute an n-by-n multiplication.

The BSD representation of multiplicand B, the value of –B (–2B) is generated through a simple NOT over all bits of B (2B). 2B is generated via a 1-bit left shift over B. Note that each partial product consists of (n+1) digits (i.e., binary positions), each of which has a negabit and a posibit.

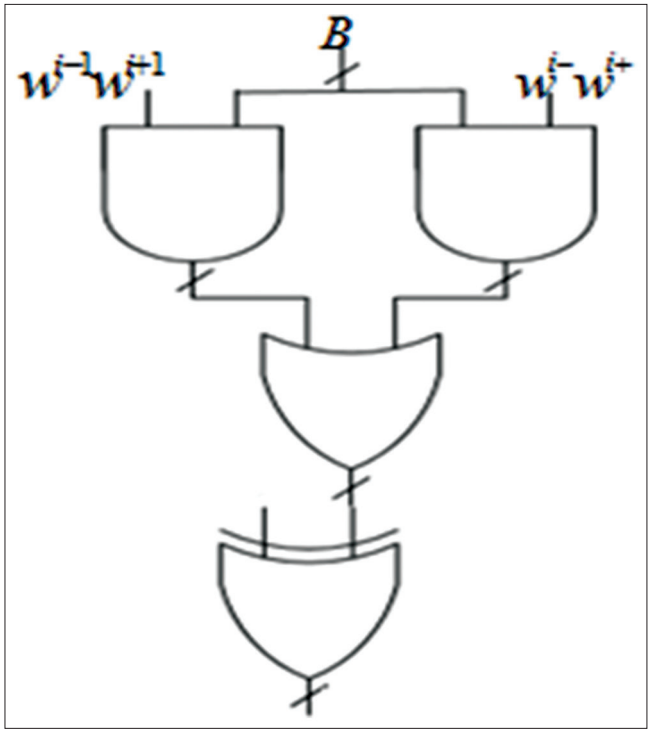

Fig. 3: Partial product generation in Modified Booth encoding

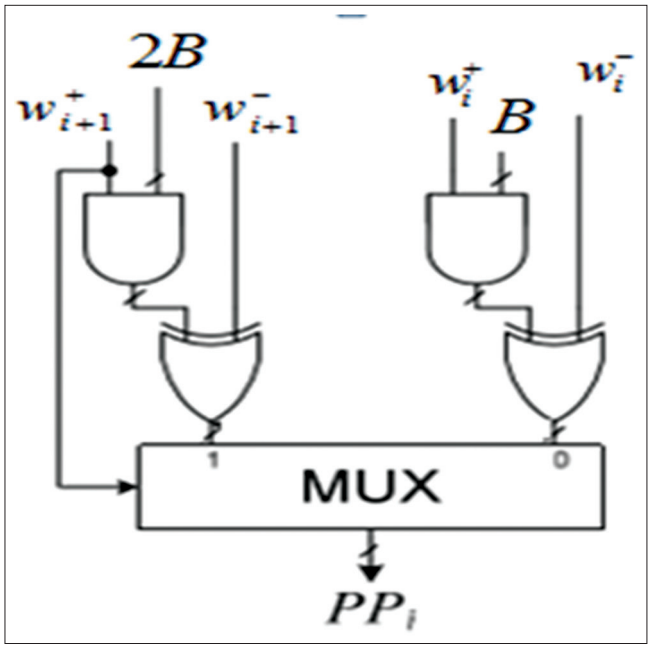

Fig. 4: Circuitry for partial product generation according to Table 1

Table 1: Generation of i^{th} partial product

$W^-_{i+1}W^+_{i-1}$	$W^-_iW^+_i$	$W^-_{i+1}W^+_{i-1}W^-_iW^+_i$	PP_i
00	00	0	0
00	01	1	B
00	11	−1	−B
01	00	2	2*B
11	00	−2	−2*B

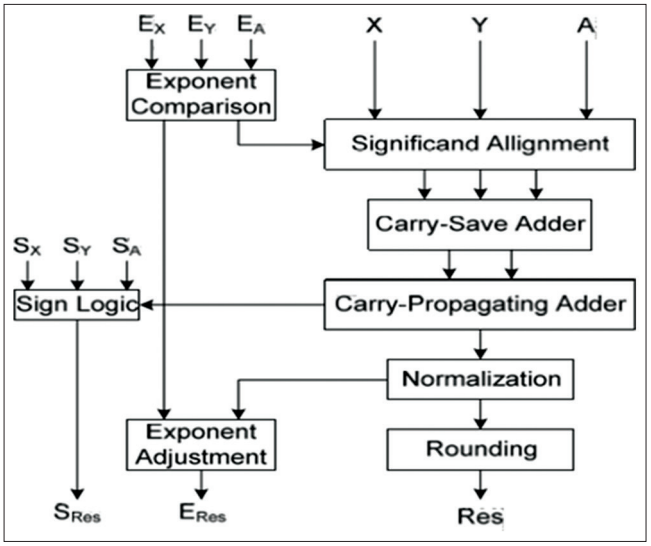

Fig. 5: Proposed floating point adder

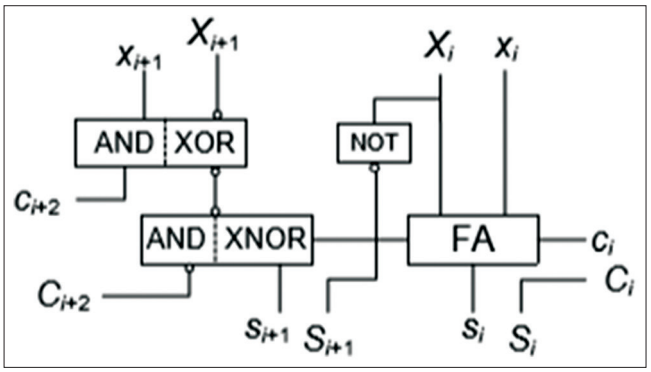

Fig. 6: Binary signed digit slice adder

PPR

The important task for the PPR contains carry-limited addition. In partial product, the addition of intermediate stage is done using carry look ahead adder in which at last carry is added into sum making less area to occupy. There are two approaches to reduce the partial products; (1) Reduction by rows and (2) reduction by columns. It takes three steps to reduce the eight operands while each step has the latency of one adder. The total number of adders used in this reduction method is seven. Reduction of p operands can be divided into two parts, each of which reduces p/2 operands and then add the outputs together [5].

FP adder

The addition of FP is carried using the following algorithm:

a. Exponent difference determines exponent difference Δ and the smaller exponent
b. Alignment shift: Shift right (Δ positions) the significands of the number having a smaller exponent. The largest exponent is the result's exponent
c. Addition/subtraction over significands: Determine and perform the actual operation; may need to swap the operands
d. Normalization shift: Shift right 1 bit, in case of addition overflow. Detect the number of leading zeros and shift left, in case of subtraction, such that there is a new hidden 1 to the left of the binary point
e. Rounding: Use extra bits (round, guard, and sticky) to round the result. This may lead to postnormalization
f. Exponent adjustment: Adjust the exponent to compensate for the shifts in (d) and (e). But in this case, the normalization and rounding are done using conventional methods for BSD representation.

Fig. 7: Register transfer language Schematic Wallace tree

Fig. 8: Output of Wallace tree

In the proposed three-operand floating-point adder, a new alignment block is implemented and carry-save adders-carry propagate adder are replaced by the proposed BSD adders and sign logic is eliminated [4]. The exponent comparison and significand alignment of the proposed architecture are almost the same as that of the fused three-operand adder. The only difference is that the significand alignment block, in the proposed design, does not wait for the exponent comparison block to finish and part of significand alignment operation is overlapped with the exponent comparison [3]. Moreover, the addition part also overlaps with the significand alignment and exponent comparison. The only blocks that wait for other blocks to finish before they start their operations are Normalization, Rounding, and exponent adjustment. Therefore, having multiple blocks working partially in parallel makes the proposed three-operand floating-point adder faster than previous works [1]. 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 58-digit BSD adder in which some positions consist of the digit adder with $y_i Y_i y_{i+1} Y_{i+1}$ assigned to "0". The next steps are normalization and rounding which are done using conventional methods for BSD representation [23,24]. Normalization of redundant operands is more complicated than that of non-redundant operands. The first step to normalize a redundant represented number is to detect and eliminate the leading non-zero digits of no significance [24]. In other words, there is a need to eliminate non-zero digits whose total values are zero. The next step is to detect the number of leading zeros and then shift the operand to the left accordingly. The leading zero detector (LZD) can be implemented using a divide-and-conquer approach in which, first, a 2-bit LZD is designed and then larger LZD is built using the basic 2-bit LZD. D signals represent if any "1" is detected and P signals represent the position of the detected "1" [6]. Using the same approach, 4-bit LZD detectors can be used to build an 8-bit LZD and so on. At the end, the

value of P shows the number of leading zeros. This value is passed to the barrel shifter to perform left shifts on the operand.

EVALUATION

Xilinx tool is used for the simulation of the design. Wallace tree is used to implement the Modified Booth encoder. The Schematic of Wallace tree in register transfer language is given as:

CONCLUSION

We have used FP based butterfly architecture, in which we had firstly converted the FP into BSD to make addition quite simpler and time saving. Smaller area is required to design a Modified Booth algorithm. It will be more beneficial to use butterfly architecture so as reduce the area used by complete circuitry will be less and also the speed will be higher. Furthermore, dual-path FP architecture can be used to implement addition and subtraction of BSD. The critical path of the three-operand adder consists of: Two 8-bit carry-propagating subtractors (0.25 ns each), a MUX (0.07ns), a barrel shifter (0.29 ns), the final BSD adder (0.16 ns), normalization and rounding (0.75 ns), registers (0.22 ns).

REFERENCES

1. Swartzerlander EE Jr, Saleh HH. FFT implementation with fused floating point operations. IEEE Trans Comput 2012;61(2):284-8.
2. Sohn J, Swatzerlander EE Jr. Improved architectures for a floating point fused dot product unit. In: Proceeding IEEE, 21st Symposium on Computer Arithmetic. April; 2013. p. 41-8.
3. IEEE Standard for Floating Point Arithmatic, IEEE Standard, 754, 2008, August; 2008. p. 1-58.
4. Cooley JW, Tukey JW. An algorithm for machine calculation of complex Fourier Series. Math Comput 1965;19(90):297301.
5. Baba SK, Rajaramesh D. Design and implementation of advanced modified booth encoding multiplier. Int J Eng Sci Invent 2013;2(8):60-8. Available from: http://www.ijesi.org.
6. Schneider K, Willenbucher A. A New Algorithm for Carry Free Addition of BinarySigned-Digit Numbers, IEEE 22nd International Symposium on Field-Programmable Custom Computing Machines; 2014.
7. Min JH, Kim SW, Swatzerlander EE Jr. A floating point fused FFT butterfly arithmetic unit with mergerd multipliers. In: Proceeding 45th Asilomar Conference on Signals, Systems, and Computers. November; 2011. p. 520-4.
8. Tenca AF. Multi-operand floating point addition. In: Proceeding 19th Computer Arithmetic, June; 2009. p. 161-8.
9. Palnitkar S. Verilog HDL: A Guide to Digital Design and Synthesis, IEEE 1364-2001 Compliant. 2nd ed. Upper Saddle River, NJ: Pearson; 2003.
10. Prokis JG, Manolkis DG. Digital Signal Processing. 4th ed. Englewood Cliffs, NJ: Pearson Prentice Hall; 2007.
11. Kuang SR, Wang JP, Guo CY. Modified booth multipliers with a regular partial product array. IEEE Trans Circuits Syst II 2009;56(5):404-8.
12. Wang LR, Jou SJ, Lee CL. A Well-structured modified booth multiplier design. IEEE International Symposium on VLSI Design, Automation and Test. VLSI-DAT; 2008 Apr 23-25; Hsinchu. IEEE. p. 85-8.