© 2017 IJSRSET | Volume 3 | Issue 1 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

A Novel Approach to Optimize Area of Fused Floating Point Three Term Adder

C. Yamunarani, M. Indu

Department of Electrical and Communication Engineering, SNS College of Technology, Tamilnadu, India

ABSTRACT

This paper presents an area efficient architecture for fused floating point addition using three terms. The first step of fused floating point addition is exponent comparison and significand alignment which occupies a major proportion of area in the overall architecture. Reduction in area is achieved by replacing exponent processing , significand alignment block and mantissa addition block of the existing fused floating point three term adder architecture with blocks having reduced area and comparable speed .The blocks proposed utilizes less hardware compared to the existing blocks without any compromise in the performance. The performance measures are evaluated using a specific tool and reduction in area is observed from the proposed work.

Keywords: Area Efficient , Exponent Compare, Significand Alignment, Ling Adder.

I. INTRODUCTION

The most preferred standard for floating point arithmetic is IEEE-754 standard [1]. The need for uniform treatment of real numbers and efficient approximation of real numbers lead to the evolution of IEEE-754 standard and has been adopted universally by almost all computer manufacturers. It specifies interchange, arithmetic formats and methods for binary and decimal floating-point arithmetic in computer programming environments .Floating point arithmetic involves single, double and quadruple precision, where precision refers to number of significant digits it takes to represent a number. Computing machines are organized with multi-core computing system, to provide high floating point computational support. Graphic and Multimedia applications require intensive single precision operations in parallel. Hardware implementations such as PCs based on Intel *87 chips support only single ,double and double extended, and most other hardware implementations support only single and double precision. Floating point DSPs have offered faster and easier manipulation that outweighs the importance of floating point units. Addition of floating point numbers is a basic requirement for DSP applications involving large dynamic range of data operands.

There are several works related to fused floating point arithmetic such as fused multiply-add unit [2] ,fused add-subtract unit [3,4] and fused dot-product unit [5].Fused floating point units performs multiple operations in a single unit. Among the operations ,floating point addition is frequently used yet complex operation .Floating point arithmetic includes processes such as exponent processing and significand alignment, normalization, addition and rounding. These operations require increased area, power consumption and latency for these reasons we tend to prefer optimizations. In case of floating point two term addition, there exists several optimized architectures and very few works related to floating point three term addition [6, 7, 8].

II. METHODS AND MATERIAL

1. Existing Floating Point Three Term Adders

Discrete design for three term addition employed series of two term additions, that lead to the loss of accuracy and takes twice the area, latency and power of two term adder blocks. To overcome that drawback fused floating point units [6,7,8] came into existence, these units shares a common logic to perform two additions at once with improved accuracy since rounding performed once.

Traditional fused floating point three term adder [7,8] takes three operands and perform two additions at once. There are chances for optimizations with the traditional architecture as proposed in [6]. The optimizations proposed in fused floating point three term adder [6] includes 1)New exponent compare and significand alignment scheme to compute the maximum exponent and shifts the significands according to the exponent differences , 2)Dual-reduction to avoid the need for complementation after the significand addition, 3)Early normalization to reduce the adder size while maintaining precision ,4)Three input LZA in parallel with significand addition to prevent delay overhead and 5)Compound addition and rounding as in Fig 1.

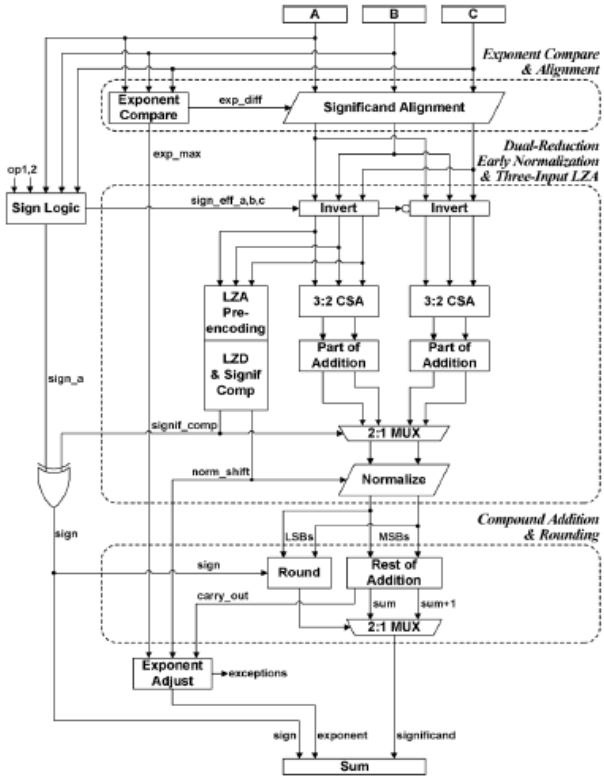

Figure 1. Existing fused floating point three term adder

Pipelining was employed to achieve high throughput and reduced delay. Existing and proposed architecture involves three pipeline stages to produce output at each cycle with the stages organized as

First Stage : Unpack-Exponent compare-significand alignment, Second stage : Invert-LZA / LZD-Normalization, Third stage: Significand addition-Round select.

III. RESULTS AND DISCUSSION

Proposed Area Efficient Fused Floating Point Addition

The exponent compare and significand alignment block of the existing work occupies major proportion of area in the overall architecture so it is modified with an area efficient block [7] and different comparison logic is proposed for the computation of maximum exponent. The Kogge stone adder for mantissa addition is replaced with Ling adder [16,17] since ling adder was found to have reduced computation and comparable speed. Pipelining is employed in the proposed work to sustain the advantage of high throughput, reduced delay and to obtain output at each clock cycle.

1) **Exponent compare and significand alignment**

Two numbers with unequal exponents can't be added, the significands need to be aligned according to exponent differences and this is performed by exponent compare and significand alignment block. Finding the maximum exponent is the first step of floating point addition. The second step in floating point arithmetic is to shift the normalized significand corresponding to the smaller exponent right by the exponent difference obtained by subtracting the smaller exponent from the maximum exponent .In general, Exponent difference is given by

$$\Delta E = E1 - E2$$

The smaller significand is shifted by the difference amount ΔE i.e., dividing the significand by $2^{\Delta E}$.Implicitly this is compensated by adding ΔE to the smaller exponent, equalizing it with larger exponent. Once the exponents are made equal, their significands can be directly added. The right shift denormalizes the smaller significand if the exponent difference ΔE is greater than zero. The information corresponding to data loss of significand should be provided for proper rounding operation that is why sticky logic is performed during significand alignment process.

Traditional methods finds maximum exponent using exponent differences. It involves subtraction, complementation and significand shift resulting in increased latency .whereas in exponent comparison method shown in fig. 2 six subtractions are performed in parallel between all the combination of two exponents. Need for complementation is eliminated by selecting a

positive difference from each pair. Difference computation and significand shift are overlapped since difference LSBs are produced first.

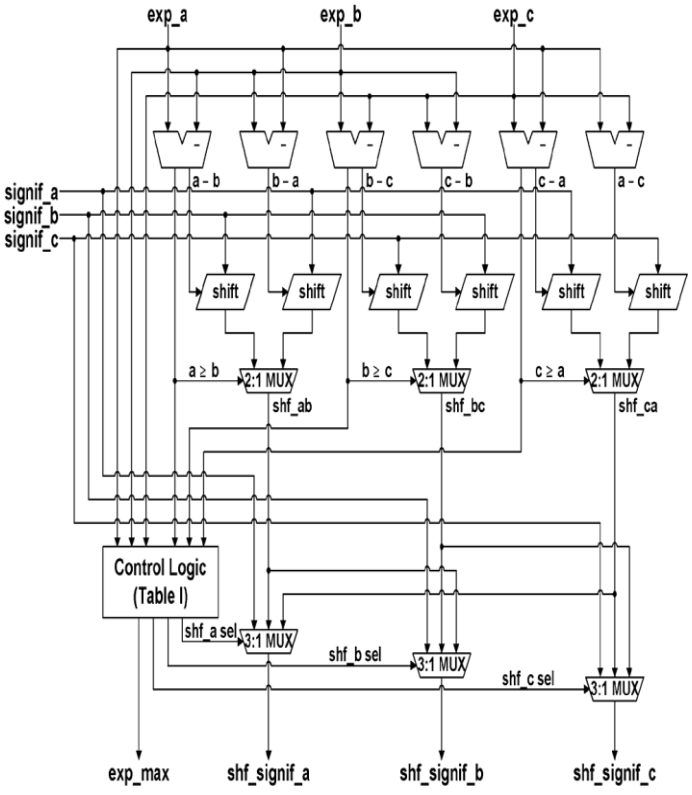

Figure 2. Existing Exponent comparision and significand alignment block

This employs another comparator and a multiplexer and finally the maximum out of the three exponents are obtained at the output. Subtraction between the maximum exponent and the other two exponent gives the exponent difference and this becomes the shift amount for significand alignment. Since computation of maximum exponent occurs at the earlier stage prior to difference computation, there is no way for the difference to go negative so need for complementation as well as selection of absolute value of difference is eliminated.But in case of existing logic, prediction of maximum exponent takes place only at the last stage so one may not know whether the difference will be positive or negative so we use 6 subtractors (two for each combination) and also we need a multiplexer to select the positive difference value for alignment. In addition to that, the existing work compares three pairs of exponents whereas in proposed work only two comparisons are done. All these modifications lead to efficient reduction in area and significance is better observed with large number of inputs.

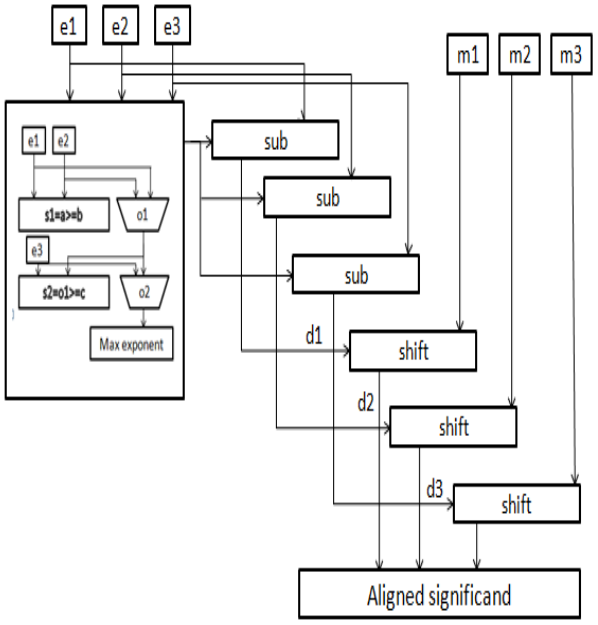

Figure 3. Proposed Exponent comparision and significand alignment block

Sticky logic is performed during this process to determine guard (G), round(R) and sticky(S) bits. G and R are the first two bits under LSB, S is set if one of the over shifted bits are 1.The bit width of aligned significand is 2f+6 bit wide to guarantee significand precision. The largest exponent and aligned significands are found using control logic based on exponent comparison and is done for each combination of exponent. With increasing number of inputs the proposed block for exponent comparison and significand alignment achieves significant reduction in area and the extension of the architecture is also very simple compared to extension of existing work.

This is followed by part of mantissa addition[2]. The mantissa addition is performed with the reduced significands using ling adder. Three input leading zero anticipation[10] is performed in parallel with mantissa addition.[2,9,11,12,13]. In normal cases the result of addition is usually normalized with LZD placed after the adder block. To hide the delay of reduction block, three input LZA is employed it has 1) Pre encoding indicator vectors and 2) LZD tree to compute the shift amount.

The pre_encoding vector is obtained by performing bit wise operation to generate W vector using four significands as follows,

W=A+B-C-D

The W vector is used to find propagate (t), generate(g) and kill(z) terms and F vector is computed as shown in equation below.This vector is then passed through LZD tree for leading zero count.In some cases there may be error in the anticipated zero count and correction logic to be incorporated as in [14,15,18] and to select the positive significand pair significand comparison bit is generated at this stage.

$$\mathbf{fi} = \mathbf{t_{i+1}}(\mathbf{g_i}\overline{\mathbf{z_{i-1}}} + \mathbf{z_i}\bar{\mathbf{g}}_{\mathbf{i-1}}) + \bar{\mathbf{t}}_{\mathbf{i+1}}(\mathbf{z_i}\overline{\mathbf{z_{i-1}}} + \mathbf{g_i}\overline{\mathbf{g_{i-1}}})$$

$$\mathbf{Signif\_comp} = \mathbf{z_n} + \mathbf{t_n z_{n-1}} + \mathbf{t_n t_{n-1} z_{n-2}} + \cdots + \mathbf{t_n t_{n-1} t_{n-2} \ldots z_0}$$

The significand comparison and sign_a are used for determination for final sign. Sign logic determines the sign of the result. Both positive and negative reduced significand pair are obtained using two reduction trees. Positive pair is selected based on the sign of the significand sum.

2) Mantissa Addition

The second modification is the adder with Ling adder for mantissa addition .The aligned significand after inversion and reduction will be passed to the mantissa addition block. Parallel prefix adder is employed for mantissa addition and few stages of addition is completed before normalization. The delay of an adder depends on how fast the carry reaches each bit position. Hence the major bottleneck in the design of binary addition is the carry chain which computes the carries. To reduce the delay and to improve the performance, the parallel prefix adders can be employed. The concept in parallel prefix adder is to compute a small group of intermediate prefixes and then find the large group of prefixes, until all the carry bits are computed. The three stages of a prefix adder includes

- Pre – Processing stage
- Prefix stage
- Post _ Processing stage

Carry equations of any conventional prefix adder and ling adder are shown below

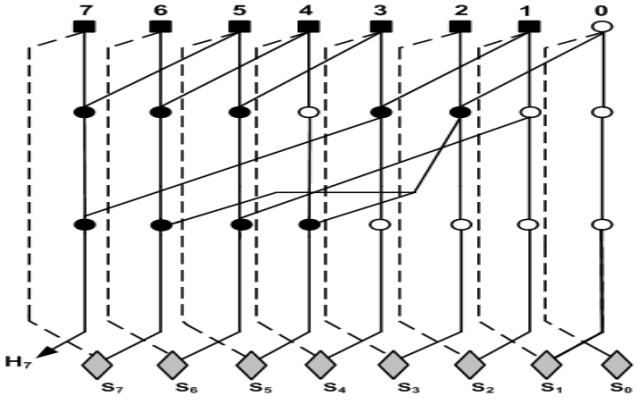

Figure 4. Ling adder for mantissa addition

C = gi + pi gi-1+pi pi-1 gi-2+……..pi pi-1 pi-2….g0
Hi = gi +gi-1+pi-1gi-2+…..+pi-1.pi-2…p1.g0

The ling adder shown in fig 4 can be extended for increasing inputs.

Rest of the addition and rounding is in the last stage of significand processing .Compound addition and rounding [10,19] are performed after normalization for higher and lower bits respectively. Compound addition determines higher bits including overflow bits and rounding determines three LSBs and round decision. A 3 bit adder determines three round up bits and L2, L1 and L0 to determine three LSBs of result, carry-out of this addition selects result between sum and sum+1 obtained from compound addition.

Exponent adjustment logic uses the maximum exponent from exponent comparison block, and is adjusted by subtracting shift amount and adding the carry-out(overflow bits) of significand addition. Exceptions specified in IEEE-754 standard such as overflow, underflow and inexact are found using this block.

IV. FUTURE SCOPE

There are chances for optimizations of delay in addition to area. Area optimization will be better observed when the architecture is improved for handling many inputs.

V. CONCLUSION

The proposed architecture is obtained by applying modification to significand alignment and mantissa addition of the existing work and the proposed architecture handles both single and double precision. In addition to this it can also handle exceptions .The

architecture utilizes pipelining stages to obtain reduced delay. The proposed architecture for three term addition has achieved reduction in area compared to its previous work. The performance metrices such as area, power and delay are evaluated using a tool for the existing and proposed work, these measures are tabulated for better understanding.

VI. REFERENCES

[1]. IEEE Standard for Floating-Point Arithmetic, ANSI/IEEE Standard 754-2008, IEEE, Inc., 2008.

[2]. T. Lang and J.D. Bruguera, "Floating-point fused multiply-add with reduced latency," IEEE Trans. Computers, vol. 53, pp. 988–1003, 2004.

[3]. H. H. Saleh and E. E. Swartzlander, Jr., "A floating-point fused add subtract unit," in Proc. 51st IEEE Midwest Symp. Circuits Syst., 2008,pp. 519–522.

[4]. J. Sohn and E. E. Swartzlander, Jr., "Improved architectures for a fused floating-point add-subtract unit," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 59, no. 10, pp. 2285–2291, Oct. 2012.

[5]. H. H. Saleh and E. E. Swartzlander, Jr., "A floating-point fused dot product unit," in Proc. IEEE Int. Conf. Computer Des., 2008, pp.427–431.

[6]. J. Sohn and E. E. Swartzlander, Jr., "A fused floating-point three-term adder," in IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 61, no. 10, pp. 2842–2850, Oct. 2014.

[7]. A. Tenca, "Multi-operand floating-point addition," in Proc. 21st Symp. Computer Arithmetic, 2009, pp. 161–168.

[8]. Y. Tao, G. Deyuan, F. Xiaoya, and R. Xianglong, "Three-operand floating-point adder," in Pro. 12th IEEE Int. Conf. Comput. Inf Technol., 2012, pp. 192–196.

[9]. Jongwook Sohn, "Improved architecture for fused floating-point arithmetic units," Ph.D. dissertation, University of Texas, Austin,2013.

[10]. 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.

[11]. M. S. Schmookler and K. J. Nowka, "Leading zero anticipation and detection-a comparison of methods," in Proc. 15th IEEE Symp. Computer Arithmetic, 2001, pp. 7–12.

[12]. V. G. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit comparison with logic synthesis," IEEE Trans. VLSI Syst., vol. 2, no. 1, pp. 124–128, Mar. 1994.

[13]. G. Dimitrakopoulos, K. Galanopoulos, C. Mavrokefalidis, and D.Nikolos , "Low- power leading zero counting and anticipation logic for high-speed floating point units," IEEE Trans. VLSI Syst., vol. 16, no. 7, pp. 837–850, Jul. 2008.

[14]. J. D. Bruguera and T. Lang, "Leading-one prediction with concurrent position correction," IEEE Trans. Computers, vol. 48, no. 10, pp.1083–1097, Oct. 1999.

[15]. R.Ji.Z.Ling,X.Zeng,B.Sui,L.Chen,J.Zhang,Y. Feng, and G. Luo, "Leading-one prediction with concurrent position correction," IEEE Trans. Comput., vol. 58, no. 12, pp. 1726–1727, Dec. 2009.

[16]. N.Poornima, V.S.Kanchana Bhaskaran, "Area efficient hybrid parallel prefix adders".

[17]. Giorgos Dimitrakopoulous,Dimitris Nikolos, "High-speed parallel prefix VLSI ling adders," IEEE Trans. Comput., vol. 54, no. 2, pp. 225–231, Feb. 2005.

[18]. P. Kornerup, "Correcting the normalization shift of redundant binary representations," IEEE Trans. Computers, vol. 58, pp. 1435–1439,2009.

[19]. G. Even and P.M. Seidel, "A comparison of three rounding algorithms for IEEE floating-point multiplication," IEEE Trans. Comput., vol. 49,pp. 638–650, 2000.