Global Journal of Researches in Engineering: J General Engineering
Volume 14 Issue 6 Version 1.0 Year 2014
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 2249-4596 & Print ISSN: 0975-5861

Energy Efficient FMA for Embedded Multimedia Application

By Mandala Rakesh Raj & Ms S. Sujana
Vardhaman College of Engineering, India

Abstract- This article presents energy efficient fused multiplyadd for multimedia applications. Low cost, low power and high performance factors diddle the design of many microprocessors directed to the low-power figuring market. The floating point unit occupies a significant percentage of the silicon area in a microprocessor due to its wide data bandwidth and the area occupied by the multiply array. The fused floating-point multiply-add unit is utilitarian for digital signal processing (DSP) applications such as fast Fourier transform (FFT) and discrete cosine transform (DCT). The proposed designs are implemented for single precision and synthesized with a 45-nm standard cell library. To improve the performance of the fused floating point multiply-add unit, we are supervening upon leading zero anticipation with the novel leading zero detection, as the novel leading one detection algorithm allowing us to significantly reduce the anticipation failure rates.

Keywords: floating point, binary128, fused multiply add, simd, implementation, computer arithmetic.

GJRE-J Classification : FOR Code: 090607

Strictly as per the compliance and regulations of:

Energy Efficient FMA for Embedded Multimedia Application

Mandala Rakesh Raj [α] & Ms S. Sujana [σ]

Abstract- This article presents energy efficient fused multiply-add for multimedia applications. Low cost, low power and high performance factors diddle the design of many microprocessors directed to the low-power figuring market. The floating point unit occupies a significant percentage of the silicon area in a microprocessor due to its wide data bandwidth and the area occupied by the multiply array. The fused floating-point multiply-add unit is utilitarian for digital signal processing (DSP) applications such as fast Fourier transform (FFT) and discrete cosine transform (DCT). The proposed designs are implemented for single precision and synthesized with a 45-nm standard cell library. To improve the performance of the fused floating point multiply-add unit, we are supervening upon leading zero anticipation with the novel leading zero detection, as the novel leading one detection algorithm allowing us to significantly reduce the anticipation failure rates.

Keywords: floating point, binary128, fused multiply add, simd, implementation, computer arithmetic.

I. Introduction

This paper presents energy efficient fused multiply-add unit for multimedia applications. In this, floating point can be implemented by using different precisions. As we have SP (Single Precision), DP (Double Precision), QP (Quadruple Precision). IEEE Binary 32 which is pertained as single precision and binary64 referred to as double precision. Floating-point operations have both gained widespread popularity in versatile multimedia and scientific applications, resulting in modern processors patronize both the precisions. Due to accumulation errors in computations they are becoming deficient for today large-scale applications. This precision problem can be overcome by one promising approach that is by using the binary 128 which is referred to as quadruple precision or QP format. The accuracy and numerical stability of many applications can be improved by introducing this format and is already specified in the new IEEE-754-2008 standard. Another approach to which we have to improve the performance is using the fused multiply add (FMA) operation which yields one rounding error for two operations [3]. The first FMA is introduced in the year 1990 by IBM RS/6000 [6], [7]. After FMA is implemented by several companies like HP, MIPS, ARM and Intel. Many algorithms are developed on floating-point fused multiply add unit to decrease its latency [2], [4]. As we can say it is a key feature of the floating-point unit because it greatly increases the floating-point performance and accuracy since rounding is performed only once for the result.

A Field Programmable Gate Array, FPGA provides a versatile and inexpensive way to implement and test VLSI designs. It is mostly used in low volume applications that cannot afford silicon fabrication or designs which require frequent changes or upgrades.

In FPGA's, the bottleneck for designing efficient floating-point units has mostly been area with advancement in FPGA architecture [3], there is a significant increase in FPGA densities so latency has been the main focus of attention in order to improve performance.

The main contribution and objective of our work is to implement the architecture which is proposed by Lang/Bruguera but with little change to facilitate the implementation.

In reminder of this paper is organized as follows. Proposed FMA unit section 2 backgrounds, section 3 proposed methods and section 4 describes its general architecture. Section 5 provides the evaluation results and Section 6 concludes this paper.

II. Background

In this paper, the floating-point fused multiply-add operation A x B + C is implemented for the IEEE floating- point format. In this format, a floating-point number X represents the value $X = (-1)^s \times f \times \beta^{e-p}$, Where s, f, β, e, and p are integers with s being the sign bit; f the normalized mantissa; β the radix, 2 for binary; e the biased exponent; and p the biased number.

The fused multiply-add unit gets the input operands A, B, and C with values $A = (-1)^{sa}.f_a.2^{ea}$, $B = (-1)^{sb}.f_b.2^{eb}$ and $C = (-1)^{sc}.f_c.2^{ec}$ and performs the fused multiply- add operation:

$$A \times B + C = rnd((-1)^{sa \oplus sb}.f_a f_b.2^{ea+eb} + (-1)^{sc}.f_c.2^{ec})$$

Where the computed fused multiply- add result is rounded and normalized. The FMA architecture proposed before implemented in several floating-point units of general-purpose processors is shown in Fig. 1. The steps in this implementation are:

Author α: M.Tech (DECS), Vardhaman College of Engineering, Vavilalapally, Karimnagar, Telangana.
e-mail: Mrakeshraj89@gmail.com
Author σ: Associate Professor, Vardhaman College of Engineering, Vavilalapally, Karimnagar, Telangana. e-mail: sujanasm@gmail.com

a) Multiplication and alignment shift

- Acquiring an intermediate carry-save product by multiplication of B and C.
- Inorder to reduce the latency, the bit inversion and alignment of the significand of A is done in parallel with the multiplication [2]. The bit inversion provides the one's implement of A for an effective subtraction.
- The shift amount of the alignment depends on the value of d = E_a – (E_b + E_c), where E_a, E_b, E_c are the exponents of the A, B, and C operands, respectively.
- When d ≥ 0 (i.e. E_a > (E_b + E_c)), in a conventional alignment, B x C would have to be aligned with a right shift of d bits.

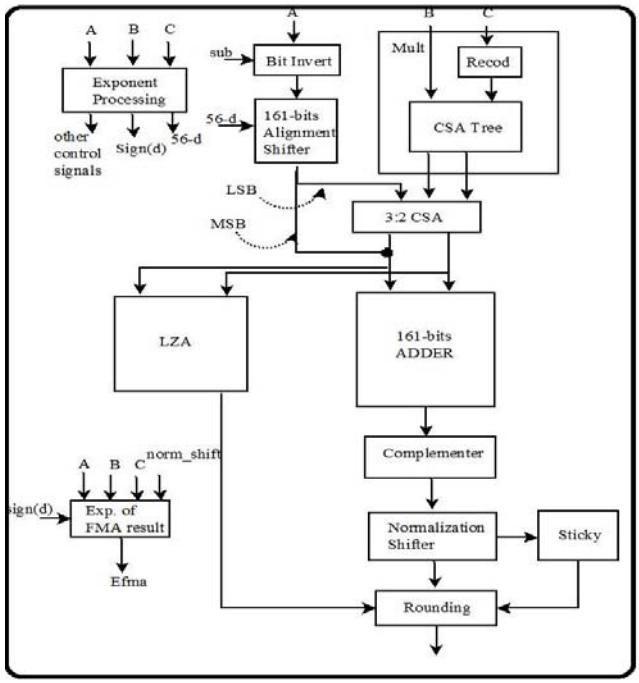

Fig. 1 : Basic architecture of FMA unit

- Instead, shift the addend A to the left to perform the alignment parallel with the multiplication. For double precision format the maximum left alignment shift would be 56 bits. When d ≥ 56, $B \times C$ is placed to the right of the least significant bit of A; in this case, B x C affects only the calculation of the sticky bit. Maximum left shift is obtained by observing that the guard (position 53) and the round (position 54) bits are 0 when the result signific and corresponding to the addend. Consequently two additional positions are included, resulting in the shift of 56 positions. When d < 0, the addend A would have to be aligned with a right shift of d bits. In this case the maximum alignment shift would be 105 bits for double precision formats.
- For shift amount larger than 105, d < -105, the operand A is placed to the right of the least-significant bit of $B \times C$, affecting only the calculation of the sticky bit.
- To avoid bidirectional shifter the alignment is implemented as a right shift by placing the addend A to the left of the most significant bit of the product B x C by 56 bits. Two extra bits are placed between the addend A and the product $B \times C$ to allow correct rounding when A is not shifted. For ≥ 0 with this implementation, A is right shifted (56- d) bits; then, the shift amount is shift amount = max {0, 56 – d}.
- For d < 0, A is right shifted 56-d bits, then shift amount = min {161, 56 – d}. By combining both cases, the shift amount is in the range [0:161], requiring a 161-bit right shifter. Moreover, the shift amount is computed as shift amount = 56-d.

b) The multiplier produce 106-bit sum and carry vectors that are reduced together with the aligned A using 3:2 CSA, because the product has only 106 bits. The 55 most-significant bits will be sign extension bits, for theses 55 most significant bits, we use two multiplexers, one to select between A and inverted A as a sum vector and the second one to select between zeros and A as a carry vector by Xor-ing sign extension bits then the outputs of the two multiplexers are concatenated at the output of the CSA to obtain the 161-bit sum and carry words.

c) The next step is the addition of the carry and sum words and the determination of the shift amount for normalization.

- The carry and sum words, obtained at the output of the CSA, are added in a 161-bit one's complement adder (with end around carry adjustment for effective subtraction). As the result can be negative, a complementer is required at the output of the adder.
- In parallel with the significands addition, the normalization shift is determined. The LZA (Leading Zero Anticipator) [12] produces the amount of the shift directly from the operands.

d) Once the result of the addition is obtained, the normalization shift can be performed since the shift amount has been already determined. A normalization shift is required to place the most-significant bit of the result at bit 0; as a consequence, normalization is performed to compensate for the cancellation produced in subtraction as well as to compensate for the way the alignment is performed.

e) The last step is the rounding of the result

With this scheme, the delay of the FMA operation is determined by the sum of the delays of the

following components: multiplier, 3-2 CSA, 161-bit adder plus complementer, normalization, and rounding. On the other hand, the main hardware components are: multiplier, alignment shifter, 3:2 CSA, LZA, 161-bit adder, normalization shifter, and rounder.

III. Proposed Architecture

We now describe the proposed FMA architecture. Since the unit is quite complex, we present this description in a single step. In this section, we give an overview of the scheme, with just enough detail to make it understandable and believable. Here we use Fig. 2 to illustrate the description.

The objective of the proposed FMA architecture is to reduce the overall delay, and Power. Since, in floating-point addition and multiplication, one of the approaches to reduce latency has been to combine addition with rounding [5], [10], [12], [13], we follow the same approach. For this approach, in floating-point addition and multiplication, the order of normalization and rounding is interchanged. This seems impractical to do for FMA because, before the normalization, the rounding position is not known. The solution we explore is to perform the normalization before the addition.

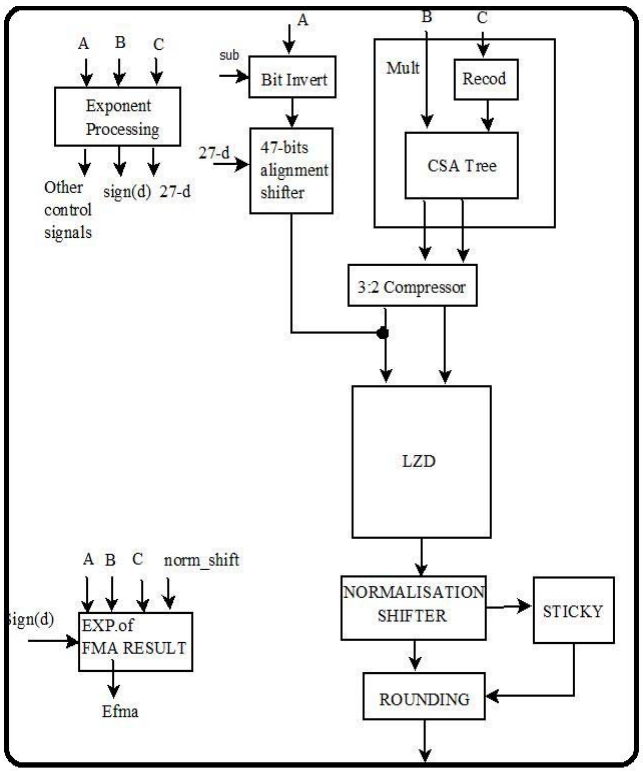

Fig. 2 : Block diagram of the proposed FMA Architecture

To improve the performance of the fused floating point multiply-add unit, we are supervening upon leading zero anticipation with the novel leading zero detection, as the novel leading one detection algorithm allowing us to significantly reduce the anticipation failure rates. The proposed leading digit is worked using tree structure, where inputs of n bits are divided into n/2 pairs of bits.

For each pair of bits a two bit count is generated and another counter is triggered to calculate the depth of the tree. For example a four digit can be paired into two pairs and a counter is used to find the one/ zero in pair *i and i+1* and the second counter is used to find the value of required bit in which pair and at which level. This method is continued of $log_2 (n)$ levels. To boost this tree structure we use a structure method which speed's up by a 4 bit or even 8bit to reduce the hierarchy of the tree structure.

IV. Detailed Description of some Modules of the Architecture

a) 3:2 CSA

The multiplier produce 106-bit sum and carry vector that are reduced together with the aligned A using 3:2 CSA. Although the output of the multiplier must be positive number because we multiply two positive numbers (sign and magnitude representation), one of the two output vectors of the multiplier may be negative because of using booth algorithm which use negative sets {-1,-2} which convert a positive number with sign and magnitude representation to a negative number with two's complement representation. The addition of sum and carry vectors must be a positive number but one of them, not both, may be negative.

Instead of using 161-bit CSA, only the 106 least-significant bits of the aligned A are needed as input to the 3:2 CSA, because the product of sum and carry vectors has only 106 bits and The 55 most-significant bits will be sign extension bits which have two cases {0, 0} if both sum and carry vectors are positive or {0, 1} if one of them is negative. For the 55 most-significant bits, we use two multiplexers, one to select between A and inverted A as a sum vector and the second one to select between zeros and A as a carry vector by Xor-ing sign extension bits then the outputs of the two multiplexers are concatenated at the output of the CSA to obtain the 161-bit sum and carry words.

b) Leading zero Anticipator

The leading zero anticipator (LZA) has two main parts: the encoding of the leading-one position i.e. detection module and the correction of this position i.e. correction module. The detection module are divided into two parts the first one is called LZA and it determines the number of leading zeros i.e. the position of the leading one. By producing a string of zeros and ones where the position of the most significant 1 is the position of the leading one. The second part, called leading zero detectors (LZD), counts the number of zero digits from the left-most position until the first nonzero digit is reached i.e. leading one position. Since the detection is done from most significant bit to least

significant bit regardless of the carry that may come from the least significant bit, the detection of leading one position may be off by one bit. The LZA logic takes two input strings and uses a set of logical equations given below.

After LZA logic LZD is used to drive the normalization shifter [11] by encoding the position of leading one to its weighted binary representation.

$$f_i = t_{i+1} \times (g_i.z_{i+1} + z_i.\bar{g}_{i+1}) \oplus \bar{t}_{i+1}(z_i.\bar{z}_{i+1} + g_i.\bar{g}_{i+1}), i > 0$$

$$f_0 = \bar{t}_0.t_1$$

Where $t_i = a_i \oplus b_i$

$$g_i = a_i.b_i$$

$$z_i = \bar{a}_i.\bar{b}_i$$

The LZD unit assumes n bits as input and produces $\log_2$ n bits of the leading one position

Pattern	Position	Valid
1x	0	Yes
01	1	Yes
00	X	No

Table (I) shows the truth table of 2-bits LZD. By using two 2-bits LZD's we can get 4-bit LZD is shown in Figure (a). Following the same concept we can get LZD with higher number of output using hierarchical structure.

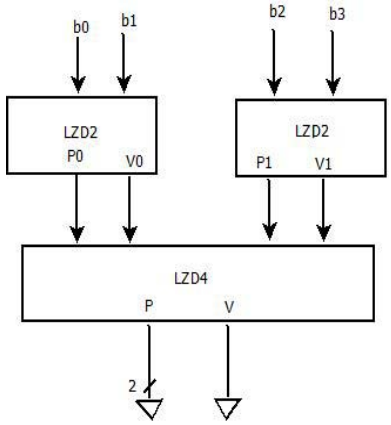

Fig. 3 : Using 2-input LZD

c) Normalization

The IEEE 754 binary floating-point standard defines a set of normalized numbers and four sets of special numbers. Of the four types of special numbers, three do not require computation for arithmetic operations. These include Not-a-Numbers (NaN), infinities, and zeros. De-normalized numbers, also known as subnormal or denormals are the fourth type of special number and do require computation.

Normalized numbers can be described by the following:

$$X = (-1)^{xs} \times 1 \times f \times 2^{xe-bias}$$

Where X is the value of the normalized number, X_s is the sign bit, $X_{f\ is}$ the fractional part of the significand, Xe is the exponent, and *bias* is the bias of the format which corresponds to 127, 1,023, and 16,383, for single, double, and quad, respectively.

Denormalized numbers can be described by the following:

$$X = (-1)^{xs} \times 0 \times f \times 2^{1-bias}, X_e = i \times f \neq 0$$

The denormal format differs from a normal number in that there is no implied bit and the exponent is not equal to X_e – *bias*, but , instead, is forced up by 1 to E_{min} which is equal to -126, -1,022, and -16,382, depending on the format.

Using the results from the LZD, the result from the adder is shifted left to normalize the result. That means now the first bit is 1.

The normalize is mostly a large shifter. The shifter has more than one stage. The stages are organized from the coarsest to the finest. The last stage performs a shift by one or two due to correction signal. This should have a negligible effect on the delay of the last stage.

d) Rounding

The IEEE 754 floating-point standard has been widely adopted since its introduction in 1985. The standard requires that all arithmetic operations are rounded so as to maintain as high a degree of accuracy and uniformity across different computing platforms as possible. The rounding decision is taken by knowing also sticky and round bits. The sticky bit is calculated from the result by OR-ing all least significant bits after the round bit. Rounding operations were originally viewed as a final separate step in most arithmetic circuit implementations. This has been merged with the carry-propagate addition in floating-point adders by delaying normalization until after rounding. Four different rounding modes are laid down by the IEEE floating-point standard [8], [9]: rounding toward 0, rounding to nearest (even), and rounding to $\pm\infty$. Rounding to nearest (even) is the standard's default mode; rounding toward zero is helpful in many DSP applications; and rounding to $\pm\infty$ is used in interval arithmetic, which affords bounds to be specified on the accuracy of a number.

V. Simulations Results

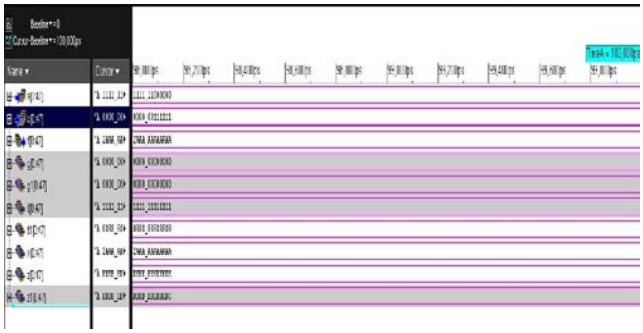

Fig. 4 : Simulation result of Leading Zero Anticipatior

Fig. 5 : Simulation result of Leading zero detector

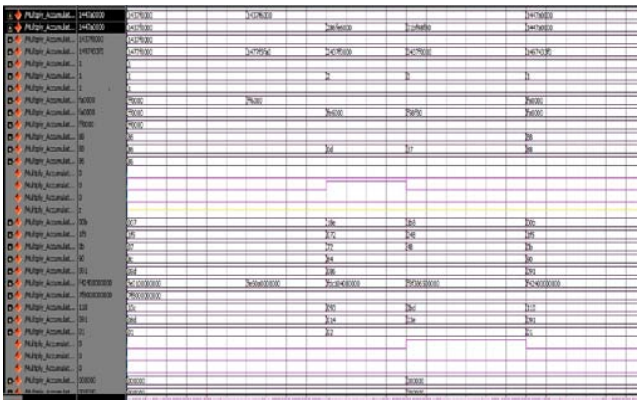

Fig. 6 : Top Module of Fused Multiply Add

a) Delay

The delay of both architectures is measured in the proposed architecture is less delay. Which is represented through the graph which is shown in the Fig the red line in the graph shows the actual FMA and the blue line represents the modified FMA.

b) Power

The power is the important aspect of the any architecture as the power decreases the power consumption of the entire processor decreases. In this project the power of the both proposed and the previous architecture are calculated using Cadence RC complier in different TSMC standard libraries. The proposed architecture is efficient in terms of power.

Table II : Summary report of Area and Power

Methods Using- nm	Area (μm^2)	Power (mW)
Existing Method(90nm)	87,908	19
Proposed Method (45nm)	6,984	6.35

VI. Conclutions

Architecture for a floating-point Multiply-Add-Fused (FMA) unit that reduces the latency of the traditional FMA units has been proposed. The proposed FMA is based on the combination of the final addition and the rounding, by using proposed LZD. This novel leading one detection algorithm allowing us to significantly reduce the anticipation failure rates. We embedded the proposed technique in Fused floating point multiply and Accumulation unit and its silicon area and performance with other existing solution. This approach has been used previously to reduce the latency of the floating-point addition and multiplication. However, it can be used only if the normalization is performed after the rounding and this is not possible for the FMA operation because the rounding position is not known until the normalization has been performed. To overcome this difficulty, we propose that the normalization be carried out before the addition. This required a careful design of some other parts of the FMA, the leading-zeros-detector (LZD).

References Références Referencias

1. Yeh, H. “Fast method of floating point multiplication and accumulation”, US patent no.5867413, February 1999.
2. T. Lang and J.D. Bruguera, “Floating-Point Multiply-Add-Fused with Reduced Latency,” IEEE Trans. Computers, vol. 53, no. 8, pp. 988-1003, Aug. 2004.
3. C. Hinds, “An Enhanced Floating Point Coprocessor for Embedded Signal Processing and Graphics Applications,” Proc. 33rd Asilomar Conf. Signals, Systems, and Computers, pp. 147-151, 1999.
4. L. Chen and J. Cheng, “Architectural Design of a Fast Floating-Point Multiplication-Add Fused Unit Using Signed-Digit Addition,” Proc. Euromicro Symp. Digital Systems Design, p. 346, 2001.
5. G. Even and P.M. Seidel, “A Comparison of Three Rounding Algorithms for IEEE Floating-Point Multiplication,” IEEE. Trans. Computers, vol. 49, no. 7, pp. 638-650, July 2000.
6. Montoye, E., “Floating Point Unit for Calculating A=XY+Z Having Simultaneous Multiply and Add,” United States patent, no. 4969118, November 1990.
7. Montoye, R., Hokenek, E., and Runyon, S., “Second –Generation RISC Floating Point with Multiply-Add Fused,” IEEE journal of solid-state circuits, vol. 25,

no. 5, October 1990, Pages: 1207 –1990, pages 1207-1213.
8. S.F. Oberman, H. Al-Twaijry, and M.J. Flynn, "The SNAP Project: Design of Floating-Point Arithmetic Units," Proc. IEEE 13th Symp. Computer Arithmetic, pp. 156-165, 1997.
9. M.R. Santoro, G. Bewick, and M.A. Horowitz, "Rounding Algorithms for IEEE Multipliers," Proc. IEEE Ninth Symp. Computer Arithmetic, pp. 176-183, 1989.
10. P.M. Seidel and G. Even, "How Many Logic Levels Does Floating-Point Addition Require?" Proc. Int'l Conf. Computer Design (ICCD 98), pp. 142-149, 1998.
11. Schmookler, M., and Nowka, K., "Leading Zero Anticipation and Detection -- A Comparison of Methods," Proceedings of the 15th IEEE Symposium on Computer Arithmetic, Vail, Colorado, June 2001, Pages: 7 – 12.
12. Lang, T., and Bruguera, J., "Floating-Point Fused Multiply-Add with Reduced Latency," IEEE Transactions on Computers, vol. 53, no. 8, August 2004, Pages: 42 – 51.
13. Santoro, M., Bewick, G., and Horowitz, M.A., "Rounding Algorithms for IEEE Multipliers," Proceedings of the 9th IEEE Symposium on Computer Arithmetic, Santa Monica, California, USA, September 1989, Pages: 176 – 183.