Global Journals LaTeX JournalKaleidoscope™
Artificial Intelligence formulated this projection for compatibility purposes from the original article published at Global Journals. However, this technology is currently in beta. *Therefore, kindly ignore odd layouts, missed formulae, text, tables, or figures.*

Energy Efficient FMA for Embedded Multimedia Application

Mandala Rakesh Raj[1] and Ms S. Sujana[2]

[1] VARDHAMAN COLLEGE OF ENGINEERING

Received: 6 December 2013 Accepted: 2 January 2014 Published: 15 January 2014

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.

Index terms— floating point, binary128, fused multiply add, simd, implementation, computer arithmetic.

1 Introduction

his paper presents energy efficient fused multiplyadd 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.

2 II.

3 Background

In this paper, the floating-point fused multiplyadd operation A x B + C is implemented for the IEEE floating-point format. In this format, a floating-point number X represents the valuep e s f X ? × × ? = ?) 1 (

, Where s , f , ? ,e ,and p are integers with s being the sign bit; f the normalized mantissa; ? 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 valuesea a sa f A 2 . .) 1 (? = , eb b sb f B 2 . .) 1 (? = and ec c sc f C 2 . .) 1 (? =

and performs the fused multiply-add operation:) 2 . .) 1 (2 . .) 1 ((ec c sc eb ea b a sb sa f f f rnd C B A ? + ? = + × + ?

Where the computed fused multiply-add result is rounded and normalized. The FMA architecture proposed before implemented in several floating-point ? 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. 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 mostsignificant 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. 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.

4 III.

5 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.

6 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.

7 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 leastsignificant 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 mostsignificant 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.

8 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 Year 2014 J 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.0), . . ( ) . . (1 1 1 1 1 1 > + ? + × = + + + + + + i g g z z t g z z g t f i i i i i i i i i i i 1 0 0 .t t f = Where i i i b a t ? = i i i b a g . = i i i b a z . =

The LZD unit assumes n bits as input and produces log 2 n bits of the leading one position Pattern Position Valid 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:bias xe xs f X ? × × × ? = 2 1) 1 (

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:0 , 2 0) 1 (1 ? × = × × × ? = ? f i X f X e bias xs

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.

9 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 carrypropagate 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 ± . Rounding to nearest (even) is the standard's default

mode; rounding toward zero is helpful in many DSP applications; and rounding to ± is used in interval arithmetic, which affords bounds to be specified on the accuracy of a number.

10 J e XIV Issue VI Version

I V. Simulations Results

11 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.

12 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). [1]

2014

Figure 1: T © 2014

[1] © 2014 Global Journals Inc. (US)

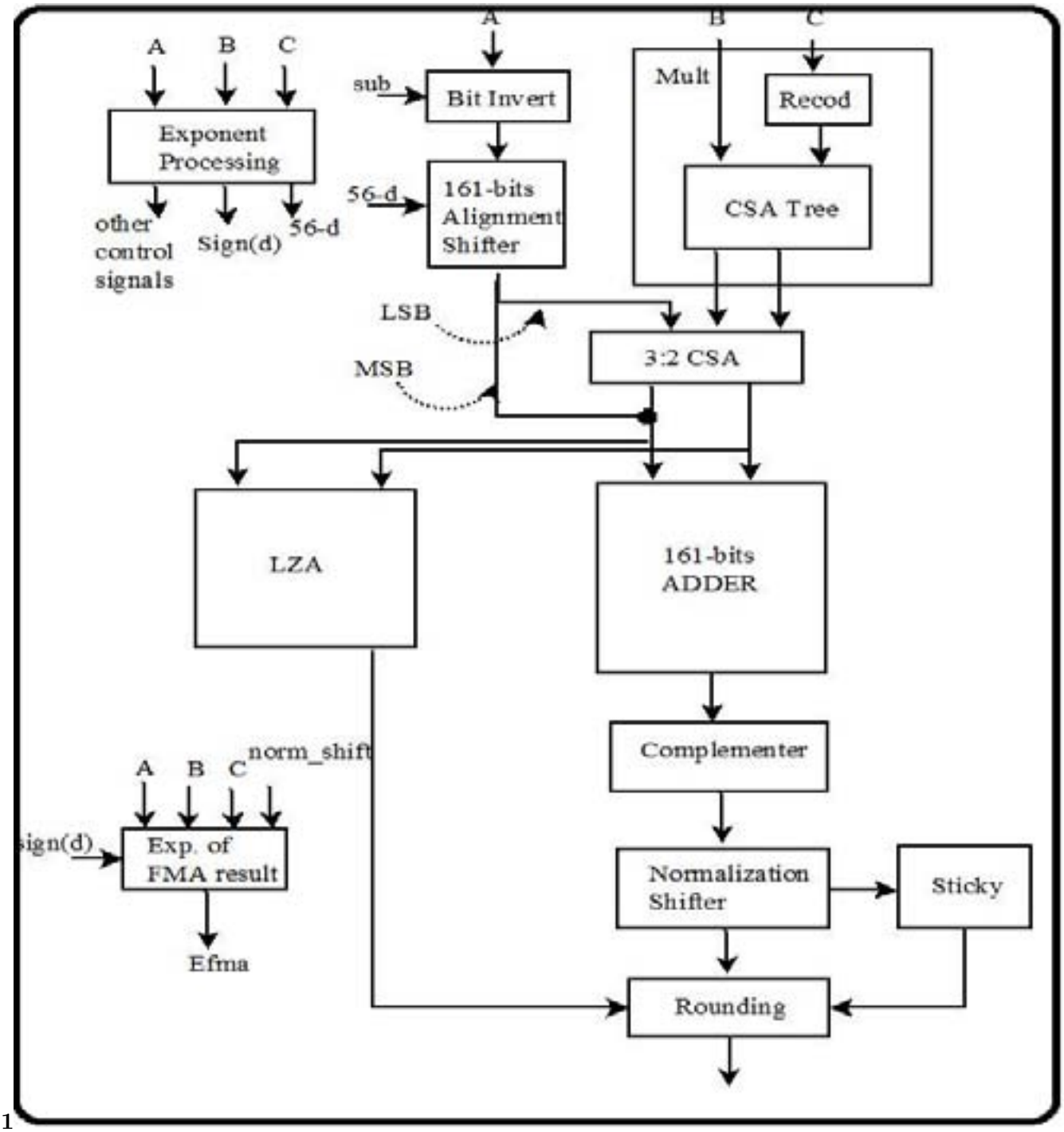


Figure 2: Fig. 1 :

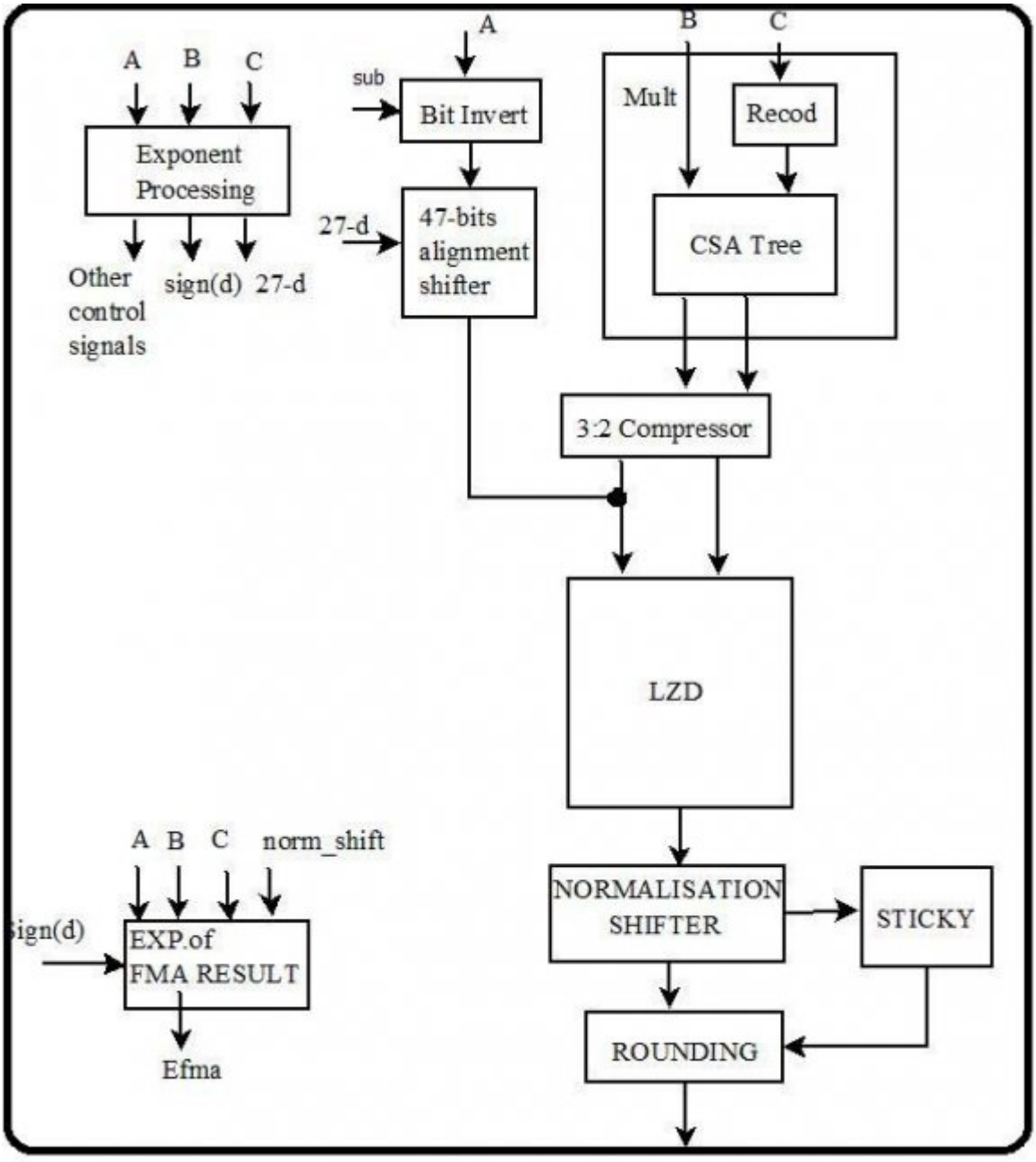
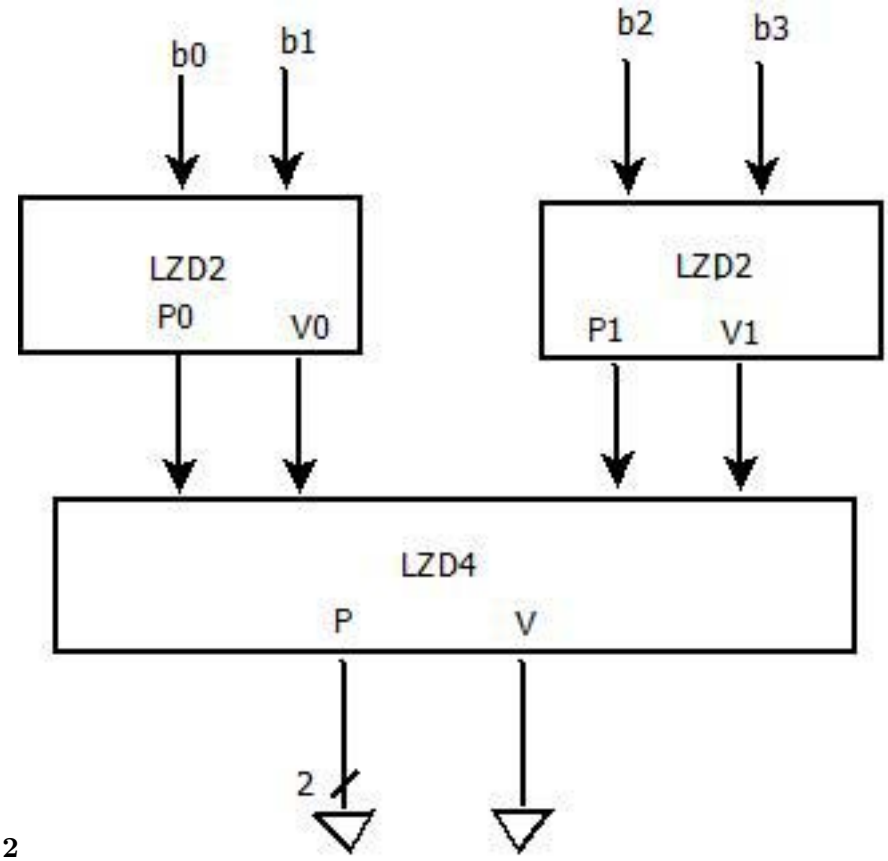

A
B
C
sub
Bit Invert
Exponent Processing
27-d
47-bits alignment shifter
Other control signals
sign(d)
27-d
Mult
Recod
CSA Tree
3:2 Compressor
LZD
A
B
C
norm_shift
Sign(d)
EXP.of FMA RESULT
Efma
NORMALISATION SHIFTER
STICKY
ROUNDING

Figure 3: Je



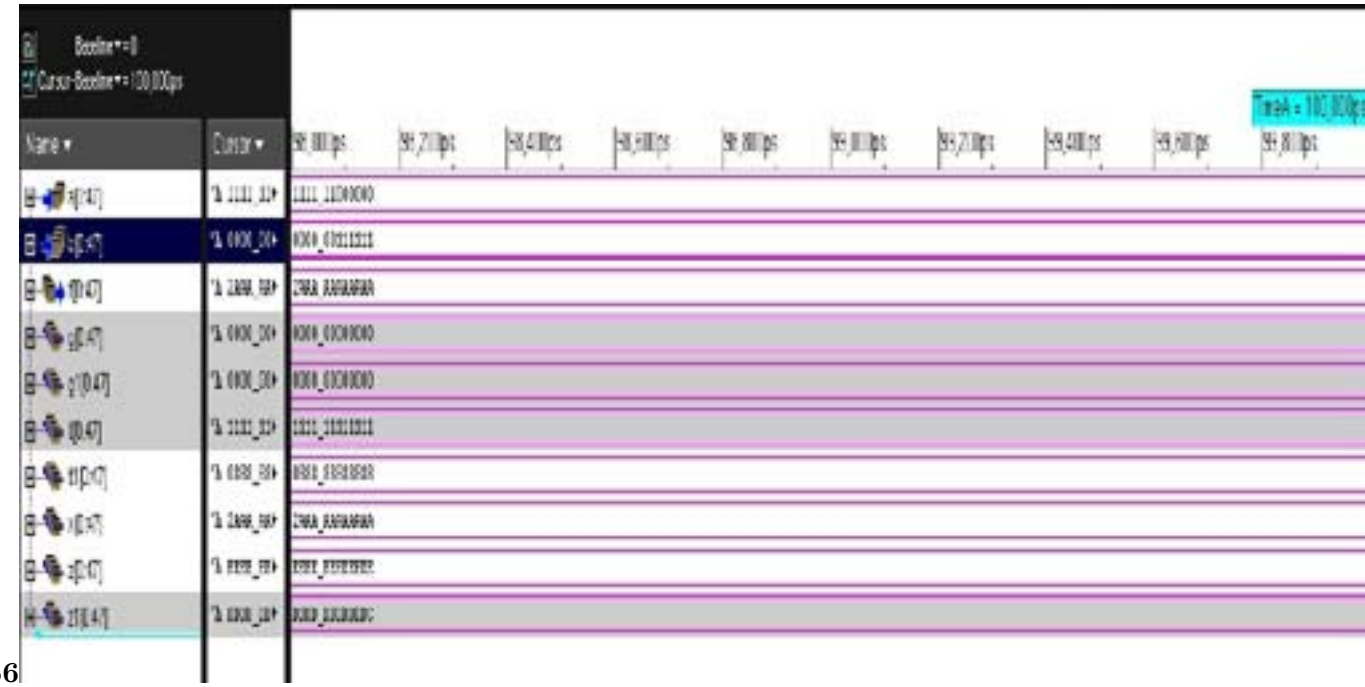
2

Figure 4: Fig. 2 :

Figure 5:

3

Figure 6: Fig. 3 :



456

Figure 7: Fig. 4 :Fig. 5 :Fig. 6 :

II

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

VI.

Figure 8: Table II :

[Computers (2004)] , *Computers* Aug. 2004. 53 (8) p. .

[Even and Seidel (2000)] 'A Comparison of Three Rounding Algorithms for IEEE Floating-Point Multiplication'. G Even , P M Seidel . *IEEE. Trans. Computers* July 2000. 49 (7) p. .

[Hinds ()] 'An Enhanced Floating Point Coprocessor for Embedded Signal Processing and Graphics Applications'. C Hinds . *Proc. 33rd Asilomar Conf. Signals, Systems, and Computers*, (33rd Asilomar Conf. Signals, Systems, and Computers) 1999. p. .

[Chen and Cheng ()] 'Architectural Design of a Fast Floating-Point Multiplication-Add Fused Unit Using Signed-Digit Addition'. L Chen , J Cheng . *Proc. Euromicro Symp. Digital Systems Design*, (Euromicro Symp. Digital Systems Design) 2001. p. 346.

[Yeh (1999)] *Fast method of floating point multiplication and accumulation*, H Yeh . February 1999.

[Montoye (1990)] *Floating Point Unit for Calculating A=XY+Z Having Simultaneous Multiply and Add*, E Montoye . November 1990. (United States patent)

[Lang and Bruguera (2004)] 'Floating-Point Fused Multiply-Add with Reduced Latency'. T Lang , J Bruguera . *IEEE Transactions on Computers* August 2004. 53 (8) p. .

[Lang and Bruguera] 'Floating-Point Multiply-Add-Fused with Reduced Latency'. T Lang , J D Bruguera . *IEEE Trans*

[Seidel and Even ()] 'How Many Logic Levels Does Floating-Point Addition Require?'. P M Seidel , G Even . *Proc. Int'l Conf. Computer Design (ICCD 98)*, (Int'l Conf. Computer Design (ICCD 98)) 1998. p. .

[Schmookler and Nowka (2001)] 'Leading Zero Anticipation and Detection –A Comparison of Methods'. M Schmookler , K Nowka . *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, (the 15th IEEE Symposium on Computer ArithmeticVail, Colorado) June 2001. p. .

[Santoro et al. ()] 'Rounding Algorithms for IEEE Multipliers'. M R Santoro , G Bewick , M A Horowitz . *Proc. IEEE Ninth Symp. Computer Arithmetic*, (IEEE Ninth Symp. Computer Arithmetic) 1989. p. .

[Santoro et al. (1989)] 'Rounding Algorithms for IEEE Multipliers'. M Santoro , G Bewick , M A Horowitz . *Proceedings of the 9th IEEE Symposium on Computer Arithmetic*, (the 9th IEEE Symposium on Computer ArithmeticSanta Monica, California, USA) September 1989. p. .

[Montoye et al. (1990)] 'Second -Generation RISC Floating Point with Multiply-Add Fused'. R Montoye , E Hokenek , S Runyon . *IEEE journal of solid-state circuits* October 1990. 25 (5) p. .

[Oberman et al. ()] 'The SNAP Project: Design of Floating-Point Arithmetic Units'. S F Oberman , H Al-Twaijry , M J Flynn . *Proc. IEEE 13th Symp. Computer Arithmetic*, (IEEE 13th Symp. Computer Arithmetic) 1997. p. .