Floating–Point Fused Multiply–Add with Reduced Latency *

Tomas Lang
Dept. Electrical and Computer Eng.
University of California at Irvine
Irvine CA 92697, USA
tlang@uci.edu

Javier D. Bruguera
Dept. of Electronic and Computer Eng.
University of Santiago de Compostela
15706 Santiago de Compostela, Spain
bruguera@dec.usc.es

Abstract

We propose an architecture for the computation of the floating–point multiply–add–fused (MAF) operation $A+(B\times C)$. This architecture is based on the combined addition and rounding (using a dual adder) and on the anticipation of the normalization step before the addition. Because the normalization is performed before the addition, it is not possible to overlap the leading-zero-anticipator with the adder. Consequently, to avoid the increase in delay we modify the design of the LZA so that the leading bits of its output are produced first and can be used to begin the normalization. Moreover, parts of the addition are also anticipated.

We have estimated the delay of the resulting architecture for double–precision format, considering the load introduced by long connections, and estimate a reduction of about 15% *to* 20% *with respect to traditional implementations of the floating–point MAF unit.*

1 Introduction

The multiply-add operation is important in many scientific and engineering applications. Recently, the floating–point units of several commercial processors include a floating–point multiply–add–fused (MAF) unit [2, 7, 9], which executes the double–precision multiply–add $A + (B \times C)$ as a single instruction. This fused implementation has two advantages: (1) the operation is performed with only one rounding instead of two and (2) there is reduction in the delay and hardware required by sharing several components [2].

*This work was performed in part while J.D. Bruguera was with the University of California at Irvine. This work was supported in part by the *Sec. Xeral de I+D de Galicia* (Spain) under contract PGIDT99-PXI20602B, by Ministry of Science and Technology of Spain under contract MCYT-FEDER TIC2001-3694-C02 and by NSF grant MIP-9726244

We propose an architecture that reduces the overall delay with respect to previous implementations. This reduction is based, mainly, on the following architectural issues:

- The combination of the final addition with the rounding. This approach is being used currently to reduce the latency of the floating–point addition and multiplication [6, 8]. The rounding step is merged with the addition using a dual adder that computes simultaneously the *sum* and the $sum + 1$. Depending on rounding bits and the most–significant bits of *sum*, the output *sum* or $sum + 1$ is selected as the rounded result and then it is normalized.

- The anticipation of the normalization before the addition. In addition and multiplication with combined addition/rounding, the normalization of the result is performed after the rounding. However, this is not possible for the MAF operation because the rounding position is not know until the normalization has been performed. Therefore, we propose to perform the normalization before the addition. Moreover, to reduce the overall delay, we overlap the operation of the leading–zeros–anticipator (LZA) with the normalization shifter.

- However, despite the overlap, the operation of the normalization shifter cannot begin at the same time as the LZA; consequently, there is a time gap where the normalization shifter is waiting for the first digit of the normalization amount. We propose to fill this gap by anticipating part of the addition before the normalization

We estimate the delay of the proposed architecture, for a double-precision floating-point format, using a family of standard cells and compare this delay with an estimate for previous implementations. We esti-

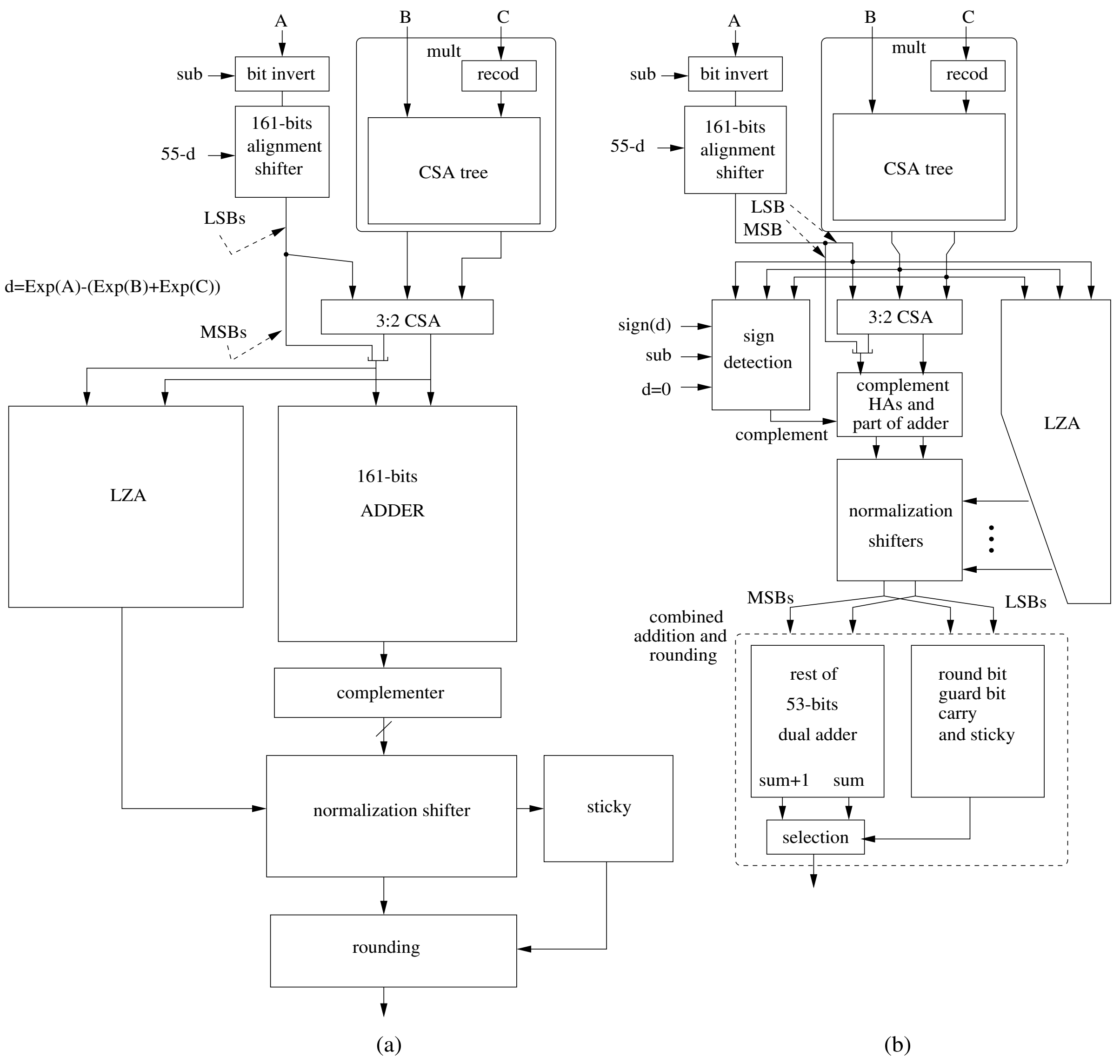

Figure 1. Schemes for the MAF operation. (a) Basic. (b) Proposed

mate that the proposed architecture results in a delay reduction of about 15% to 20%.

1.1 Basic Scheme of the MAF Unit

The MAF architecture proposed before [3], implemented in several floating–point units units of general-purpose processors [2, 3, 7], is shown in Figure 1(a). The necessary steps are:

1. Multiplication of B and C to produce an intermediate product $B \times C$ in carry-save representation.

2. Bit inversion and Alignment. To reduce the latency, the bit inversion and alignment of A is done in parallel with the multiplication. The alignment is implemented as a right shift by placing A to the left of the most–significant bit of $B \times C$. Two extra bits are placed between A and $B \times C$ to allow correct rounding when A is not shifted. This alignment requires a 161-bit right shifter.

3. The aligned A is added to the carry-save representation of $B \times C$. Only the 106 least-significant-bits of the aligned A are needed as input to the 3:2 CSA, because the product has only 106 bits. The 55 most-significant bits of the aligned A are concatenated at the output of the CSA to obtain the 161-bit sum.

4. Addition of the carry and sum words in a 161–bit adder and determination of the shift amount for normalization by means of a LZA (Leading Zero Anticipator).

5. Normalization and rounding of the result.

2 Proposed MAF: General Structure

The objective of the proposed MAF architecture is to reduce the overall delay of the MAF operation. Since in floating–point addition and multiplication one of the approaches to reduce latency has been to combine addition with rounding [1, 6, 8], 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 MAF because before the normalization the rounding position is not known. The solution we explore is to perform the normalization before the addition.

The resulting scheme is shown in Figure 1(b). This implementation shares several characteristics with the basic implementation: (1) The alignment is performed in parallel with the multiplication, by shifting the addend A and (2) the result of multiplication is in carry-save representation and a 3:2 CSA is used to obtain a redundant representation of the unnormalized and unrounded result. On the other hand, the proposal is different in the following aspects

- Normalization before the addition. The two resulting vectors of the 3:2 CSA are normalized (before the addition) and the add/round operation is performed. By performing the normalization before the addition the final result is always normalized, which simplifies the rounding step. Note that, in this case there are two normalization shifters, instead of one as in the basic approach. Moreover, to avoid an increase in delay, it is necessary to overlap the LZA and the normalization shift. To achieve this overlap, the shift count is obtained starting from the most–significant bit, and once the first bit (MSB) is obtained, the normalization shift can start.

- Dual adder of 53 bits. To obtain the rounded result, the 53 most-significant bits of the *sum* and the $sum+1$ of the two resulting vectors of the normalization are required. Therefore, the add/round module consists of a dual adder of 53 bits; this reduced adder width is possible because the normalization is performed before. Therefore, the inputs to the add/round module are split into two parts: the 53 most-significant bits are input to the dual adder; the remaining least-significant bits are inputs to the logic for the calculation of the carry into the most–significant part and for the calculation of the rounding and sticky bits. As explained later, an additional row of HA is required before the dual adder to perform the rounding.

- Sign detection. A more complicated selection between *sum* and $sum + 1$ is required if the adder output can be negative. One way to avoid this is to detect the sign of $A-(B\times C)$ and complement the outputs of the CSA, the sum and carry words, when the result would be negative.

- Advance part of the adder. Since the sign detection as well as the part of the LZA that cannot be overlapped have a significant delay, it might be convenient to place the HA and to perform some parts of the dual adder before the inversion/normalization.

Comparing with the basic architecture we can expect that the total delay is reduced and, therefore, the latency of a pipelined implementation should be also reduced. A more detailed delay analysis is presented in section 4.

3 Architecture of the Proposed MAF

Figure 2 shows the block diagram of the MAF unit. This figure is similar to Figure 1(b) but with a larger level of detail[1]. In addition, Figure 3 shows the number of bits needed in some parts of the MAF. Bits shifted out during the alignment are used to compute the sticky bit of the bits to the right of position 160 ($st1$). In the case of an effective subtraction these bits can propagate a carry into the 161-bit operands; so $st1$ and sub ($sub = 1$ if effective subtraction) are used at the output of the 3:2 CSA to produce this carry (Figure 3(c)).

To avoid additional delays in the bit invert of the carry and sum words at the output of the CSA, this operation is carried out conditionally (see Figure 2). The two 1's needed to complete the 2's complement of the operands, in case of negative result, are introduced as the LSB of the carry word and in the add/round module.

The 162-bit normalization shifter is implemented as a 54-bits shift followed by a 108-bit shifter, in such a way that the most–significant bit of the shift, the control of the 54-bit shift, can be obtained from the exponent difference and the LZA computes the control

[1]A more detailed description can be found in [5]

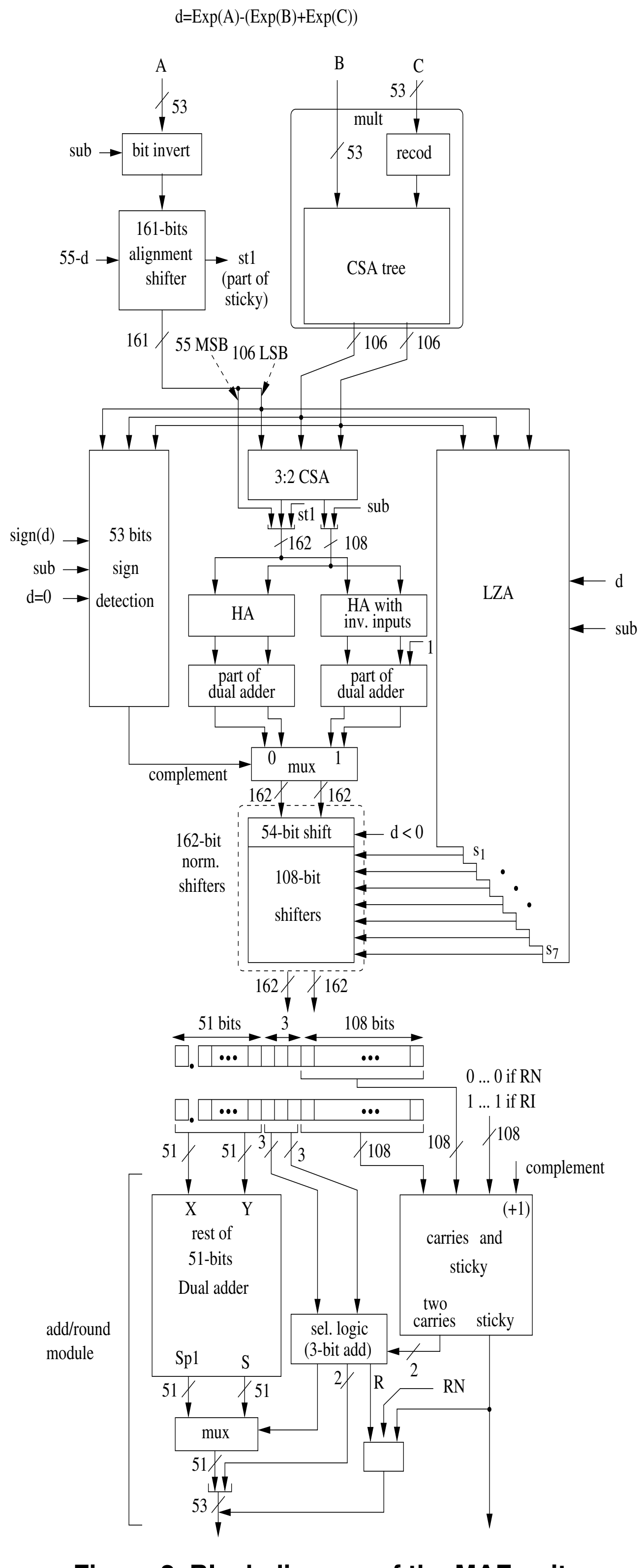

Figure 2. Block diagram of the MAF unit

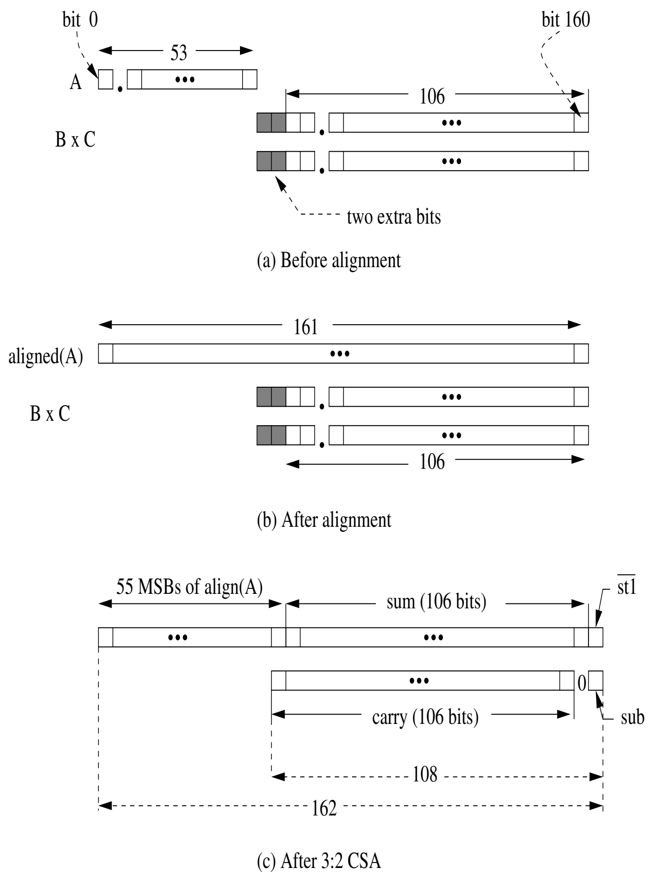

Figure 3. Wordlengths in the MAF unit

for the 108-bit shift starting from the most–significant bit. For effective addition the shift distance is $55 - d$ or $55 - d - 1$

In the add/round module the rounded and normalized MAF result is computed from the carry–save representation obtained at the output of the normalization shifter. This is a problem similar to the rounding of floating–point multipliers, where the carry–save representation of the exact product computed in the addition tree has to be rounded [1, 8, 10]; however, since the normalization has been performed before the addition, the result is always normalized, and in the case of a negative result, a $(+2)$ has to be added before the rounding. The add/round module implements the round to nearest/even (RN) and round to ∞ (RI) round modes. Moreover, the correct round to nearest/even result is obtained from the round to nearest/up changing the LSB [8]. We adapt the scheme proposed in [1, 10] for multiplication as follows:

1. The least–significant part (108 bits) is input to the logic that computes the carry into the upper part

and the sticky bit. The additional input operand to this block is the rounding injection, $0 \ldots 0$ if round to nearest or $1 \ldots 1$ if round to $\pm\infty$, which guarantees that any 1 in this lower part produces round–up of the upper part. With this additional input there are at most two carries to the upper part.

2. The most–significant part (54 bits) is divided into two parts: the 51 most–significant bits are input to the dual adder and the 3 least–significant bits are added, together with the two carries and the 1 that always gets added in position 53 for rounding, to compute the two least–significant bits of the result[2]. A row of HA is used to limit the maximum value in bits 51 to 53 and assure only one carry to bit 50. As the HAs are placed before the normalization shifter to reduce the overall delay, the position of the most-significant bit of the result is not known when the HA operate; therefore, it is necessary to include HAs for the 162 bits.

   The carry of this 3-bit addition decides whether $S = X + Y$ or $Sp1 = X + Y + 1$ should be selected at the output of the dual adder.

4 Evaluation and Comparison

In this section we estimate the delay of the critical path of the proposed MAF architecture and compare it with the basic MAF implementation.

Critical path delay in the proposed MAF

As shown in Figure 2, after the multiplier there are three parallel paths whose maximum delay is part of the critical path delay: (1) sign detection, (2) calculation of the MSB of the shift count and (3) 3:2 CSA plus HA plus part of the adder. Then, the delay of the MAF unit is given by,

$$t_{maf} = t_{mult} + t_{par\ paths} + t_{norm\ sh} + t_{rest\ of\ adder} + t_{sel} \quad (1)$$

being t_{mult} the delay of the multiplier and $t_{par\ paths}$ the largest delay of the three parallel paths. $t_{norm\ sh}$, $t_{rest\ of\ adder}$ and t_{sel} are the delay of the normalization shifter, of the part of the adder that has not been anticipated, and of the final result selection, respectively.

[2]This scheme presents some variations with respect to the implementation in [10]. In [10] there is only one carry propagating from the least–significant part. Moreover, as the result may have 1 bit of overflow, an additional 1 has to be added in position 52.

Critical path delay in the basic MAF

To obtain a faster implementation of the basic MAF, the adder is implemented as a 161–bit one's complement adder with an end–around–carry adjustment [2, 3, 7]. Moreover, due to the alignment, it consists of two pieces: a 106-bit adder and a 53-bit incrementer for the most–significant part. This way, the adder delay is the delay of a 106–bits one's complement adder,

$$\begin{aligned} t_{maf\ basic} &= t_{mult} + t_{csa} + t_{106b\ adder} + t_{1'comp} \\ &\quad + t_{norm\ sh} + t_{round} \end{aligned} \quad (2)$$

being $t_{106b\ adder}$, $t_{1'comp}$ and t_{round} the delay of the one's complement adder, of the output bit inversion and of the rounding step (basically, a 54-bit adder), respectively.

4.1 Comparison

Comparing equations (1) and (2) we can determine the delay reduction achieved with the proposed MAF. There are two components that are part of the critical path in both architectures: the multiplier and the normalization shifter. Moreover, it can be considered that the delay of the dual adder result selection in the proposed MAF (t_{sel} in equation (1)) is equal to the delay of the XOR gate to complement the adder output in the basic MAF ($t_{1'comp}$ in equation (2)). Then, the proposed MAF is faster if

$$t_{par\ paths} + t_{rest\ of\ adder} < t_{csa} + t_{106b\ adder} + t_{round} \quad (3)$$

To determine if the proposed architecture is faster than the basic one, we estimate the delay of the left and right parts in equation (3). To show the potential benefits of the approach, we use the delays specified in [11] corresponding to a 0.5μm CMOS standard-cell library. The delay unit is the delay of a 2-input NAND with a load of 2. It has to be pointed out that this is only an *approximate* estimation, since the delay is heavily dependent on the technology.

Among the parallel paths, the delays of the (1) sign detection and (2) calculation of the MSB of the shift count are almost equal and roughly equal to the delay of a 54-bit adder. Therefore, we can approximate $t_{par\ paths} \approx t_{round}$. Equation (3) can be rewritten as

$$t_{maf} < t_{maf\ basic} \quad \text{if} \quad t_{rest\ of\ adder} < t_{csa} + t_{106b\ adder} \quad (4)$$

For the delays of the 54-bits adder[3] and the 106-bit 1's complement adder, we consider prefix adders

[3]The delay of a 54-bit adder is used to estimate $t_{rest\ of\ adder}$

in both cases[4]. Then, according to our estimations, $t_{54b\ adder} = 18\ t_{nd2}$ and $t_{106b\ adder} = 23\ t_{nd2}$.

To estimate $t_{rest\ of\ adder}$ we have to determine the number of levels of the dual adder that can be placed before the normalization shifter. It is possible to anticipate the calculation of the p_i, g_i and $\bar{k}_i$ bits and two levels of the prefix tree. With these considerations, the delay of the rest of the adder has been estimated to be 12 t_{nd2} Then, replacing in equation (4), the delay reduction obtained with the proposed architecture is 15 t_{nd2}. To put in perspective this reduction, we estimate the global delay of the basic MAF architecture according to equation (2) with the following delays for the remaining components of the unit:

- 53×53 multiplier. We assume a radix–4 coding for the multiplier operand and a CSA tree with 4 levels of 4:2 CSA. We estimated a delay of 30 t_{nd2}.
- Normalization shifter. According to [2], the delay of the normalization shifter is about 40% of the delay of the 106–bits 1's complement adder, $t_{norm\ sh} \approx 10\ t_{nd2}$.

Then, the delay of the basic MAF unit is $t_{maf\ basic} \approx 87\ t_{nd2}$ and the proposed architecture achieves a delay reduction of between 15% to 20%.

5 CONCLUSIONS

An architecture for a floating–point Multiply–Add–Fused (MAF) unit that reduces the latency of the traditional MAF units has been proposed. The proposed MAF is based on the combination of the final addition and the rounding, by using a dual adder. However, this approach cannot be used directly for the MAF operation, because the rounding position is not know 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 MAF unit, the leading–zeros–anticipator (LZA) and the dual adder. The LZA is implemented so that its operation overlaps with the normalization shifter. Moreover, part of the dual addition, the calculation of g, p and $\bar{k}$ vectors and some levels of the prefix tree, is anticipated before the normalization. This permits to balance the delay of the several parallel paths in the MAF and to reduce the delay of the critical path.

The proposed architecture has been evaluated using the delays of a $0.5\mu m$ CMOS standard cell library and compared with the basic MAF architecture. The conclusion is that, for a double–precision floating–point representation, the proposed MAF reduces the delay by about $15\% - 20\%$, with respect to the basic implementation.

[4]To obtain more realistic estimations we consider the gate load and the additional load introduced by long wires. Following the evaluation in [4], the additional load of a connection of length 4 is equal to 3.

References

[1] 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. (2000).

[2] E. Hokenek, R. Montoye and P. W. Cook. *Second–Generation RISC Floating Point with Multiply–Add Fused.* IEEE J. Solid–State Circuits. Vol. 25, No. 5, pp. 1207–1213. (1990).

[3] R.M. Jessani and M. Putrino. *Comparison of Single– and Dual–Pass Multiply–Add Fused Floating–Point Units.* IEEE Trans. Computers. Vol. 47, No. 9. pp. 927–937. (1998).

[4] S. Knowles. *A Family of Adders.* IEEE 14^th Symp. on Computer Arithmetic (ARITH14). pp. 30–34. (1999).

[5] T. Lang and J.D. Bruguera. *Floating–Point Fused Multiply–Add with Reduced Latency.* Int. Rep. Univ. Santiago de Compostela (Spain). (2002). (available at www.ac.usc.es).

[6] S.F. Oberman, H. Al–Twaijry and M.J. Flynn. *The SNAP Project: Design of Floating–Point Arithmetic Units.* IEEE 13th Symp. on Computer Arithmetic (ARITH13). pp. 156–165. (1997).

[7] F.P. O'Connell and S.W. White. *POWER3: The Next Generation of PowerPC Processors.* IBM J. Research and Development. Vol. 44, No. 6. pp. 873–884. (2000).

[8] M.R. Santoro, G. Bewick and M.A. Horowitz. *Rounding Algorithms for IEEE Multipliers.* IEEE 9th Symp. on Computer Arithmetic (ARITH9). pp. 176–183. (1989).

[9] H. Sharangpani and K. Arora. *Itanium Processor Microarchitecture.* IEEE Micro Mag. Vol. 20, No. 5. pp. 24–43. (2000).

[10] R.K. Yu and G.B. Zyner. *167 MHz Radix–4 Floating–Point Multiplier.* IEEE 12th Symp. on Computer Arithmetic (ARITH12). pp. 149–154. (1995).

[11] LSI Logic. LCB500K Design Manual. (1995)