Paper Title Low Cost and Low Power Floating-point Fused Multiply-Add Unit Design with Proxy Bits and Weighted 2-Level Booth Encoding

Hyunpil Kim
School of Electrical and Electronic Engineering
Yonsei University
Seoul, Korea
Email: loimst.kim [AT] gmail.com

Abstract—**With the appearance of high performance mobile devices, low cost and low power consumption have become important issues in high performance processors. To meet the needs, low cost and low power floating-point fused multiply-add unit is proposed in this paper. According to the area and power consumption analysis, the multiplication part in fused multiply-add operation accounted for most power consumption and area. Thus, the proposed floating-point fused multiply-add unit used a new weighted 2-level Booth encoding algorithm to optimize the multiplier. In additional, proxy bits, which can represent redundant bits, from the cancellation in subtraction operation is proposed. The aim of this paper was to be achieved by optimizing a shifter, leading zero anticipator, and adder using the proxy bits.**

The synthesis and power consumption analysis results show that the proposed floating-point fused multiply-add unit reduced area by 37.3%, and improves the latency by 10.8% compared to a conventional floating-point fused multiply-add unit under no clock constraints. In addition, the proposed floating-point fused multiply-add unit reduced area and power consumption compared to the conventional unit by 31.6% and 23.3%, respectively under 2.5ns timing constraints. Therefore, the proposed floating-point fused multiply-add unit will contribute to server and mobile purpose processors' high performance, low power, and low cost requirements.. (Abstract)

Keywords- Floating-point unit, Fused multiply-add unit, Low power, Low cost, Weighted 2-level Booth encoding, Proxy bits

I. INTRODUCTION (HEADING 1)

The floating-point unit (FPU) is was developed for complex and precise point operations. It is larger, more complex, and consumes more power than an integer arithmetic unit consumes. As with any complex arithmetic operation, FPU is used in various fields such as communication, multimedia, and graphics. Recently, FPUs are not only used on server processors requiring high performance but also mobile processors requiring low cost and power consumption. Therefore, FPUs that consume low power and to have low cost and high performance are desirable.

IBM announced an FPU on the RISC System 6000 (IBM RS/6000) chip in 1990 [1, 2]. Recognizing that there would be scores of modern applications, IBM especially recursively performed a floating-point multiplication immediately followed by a floating-point addition as dot products. To enhance these applications' performance, IBM designed a new FPU that merged a floating-point multiplier and a floating-point adder into a single hardware. This FPU was called the multiply-add fused (MAF) unit at first, and then the MAF came to be called the fused multiply-add (FMA) unit by Thomas Lang and Javier D. Bruguera in 2005 [3].

The FMA unit has several advantages due to the merger of the floating-point multiplier and adder. First, the FMA unit can improve the performance of an application that recursively executes a multiplication followed by an addition. Second, the FMA unit can perform both addition and multiplication depending on a value of the input operands. Third, the FMA unit is smaller than general FPUs. Most FPUs have a floating-point adder and a floating-point multiplier in parallel, while the FMA unit is a structure sharing the rounding, normalization, and post-normalization module. Fourth, the FMA unit is capable of using integer FMA units in its design.

However, the FMA unit has several disadvantages. First, although an application may experience increased performance when a program requires multiplications followed by additions, others that involve single-instruction additions or single-instruction multiplications without the crossover experience a significant reduction in performance. Second, the additional hardware module for the merge imposes extra latency on the stand-alone instructions as compared to their original units. Third, any internal module of the FMA unit requires more bit-width and interconnection than that of modules found in floating-point adder and multiplier.

Despite these disadvantages, the FMA unit has been used in many applications, such as for DSP, graphics, and communication processing. To accommodate these needs for increased utilization of the FMA instruction, the FMA unit is now employed in high performance processors such as HP's

PA-8000, Hitachi's SH-4, Intel's Itanium, STI's CELL, Fujitsu's SPARC64v9, and ARM's VFPv4 and NEONv2.

After IBM first developed a FMA unit in 1990 [1, 2], several related studies were published. The FMA unit has been developed in order structure to support high performance [3-5] multiple path [6], multiple precision [7], and a single instruction multiple data (SIMD) [8]. Lately, there has been a trend toward studying application-specific FMA [9-11] or a FMA with a modified pipeline stage to improve program throughput [12].

This paper proceeds from the viewpoint of the implementation at the architectural level to reduce the power consumption and area of FMA units. In chapter 2, the weighted 2-level Booth encoding and proxy bits, which are key ideas of this paper are described. Then, the proposed FMA unit is explained in chapter 3. Chapter 4 shows the results of the implementation and power analysis. Findings on cost and power reduction in comparison with the conventional FMA unit are presented. Finally, chapter 5 summarizes the proposed FMA unit and highlights the benefits and weakness.

II. MOTIVATION AND KEY IDEAS

To analyze power consumption and area, Lang's FMA [5], which has been the most commonly cited since 2000, is designed with Verilog HDL and synthesized with Synopsys Design Compiler and SAED 32nm Library. Fig.1 shows the ratio of total power consumption. As seen in the figures, the modules that consume the most power are, in order, the multiplier (Booth encoder and CSA tree), shifters (align_shifter and norm_shifter), and the final adder. The previous three modules occupy the most area on the FMA unit. Peculiarly, the Booth encoder and CSA tree for the floating-point multiplication account for approximately 50% of the total area and total power consumption, respectively. Therefore, there is a need to optimize the multiplier and bit-width of the internal data to achieve a low cost FMA with low power consumption.

A. Weighted 2-level Booth encoding

According to Swee and Hai, the radix-4 Booth multiplier offers the best performance and area features [13]. However, the advantage of a higher radix Booth multiplier is that the number of partial products is reduced. The small number of partial products is especially favorable when implementing a pipeline multiplier. Another study [14] proposes an Itanium processor that uses four clock cycles for the pipeline multiplier. In such a case, large pipeline registers are required, because the Wallace (or carry-save adder) tree is divided by the pipeline registers.

Although a higher radix multiplier has advantages, there are two main problems related to its implementation. First, extra adder logics are required for a higher radix, unlike the radix-2 and radix-4. Second, larger input multiplexers (MUXs) are required in a higher radix multiplier. These problems result in a low performance and a high consumption of both resources and power. We propose that using a higher radix to reduce the number of partial products is the most beneficial strategy, and that a new Booth encoding algorithm should be developed to reduce the burden of the encoder from the higher radix. Here, we propose a weighted two-level Booth algorithm.

First, the partial products of a given radix can be defined by the following operation:

$$[\![x_{i+k+n+1}, x_{i+k+n}, \dots, x_{i+k-1}]\!] = \quad (1)$$
$$-2^{k+n+1}x_{i+k+n+1} + 2^{k+n}x_{i+k+n} + \cdots + 2^{k}x_{i+k} + 2^{k}x_{i+k-1}$$

where n represents a 2n+1-base radix, k indicates the number of encoding bits, and xi is binary. For example, the radix-4, radix-8, and radix-16 partial products are respectively derived by (2), (3), and (4) as follows:

$$[\![x_{i+1}, x_i, x_{i-1}]\!] = -2^1 x_{i+1} + 2^0 x_i + 2^0 x_{i-1} \quad (2)$$

$$[\![x_{i+2}, x_{i+1}, x_i, x_{i-1}]\!]$$
$$= -2^2 x_{i+2} + 2^1 x_{i+1} + 2^0 x_i + 2^0 x_{i-1} \quad (3)$$

$$[\![x_{i+3}, x_{i+2}, x_{i+1}, x_i, x_{i-1}]\!]$$
$$= -2^3 x_{i+3} + 2^2 x_{i+2} + 2^1 x_{i+1} + 2^0 x_i + 2^0 x_{i-1} \quad (4)$$

One of the most significant characteristics of this operation is that the law of separation is adopted. For example, the radix-16 partial products are derived by the summation of the two-stage radix-4 operations, that is,

$$[\![x_{i+3}, x_{i+2}, x_{i+1}]\!] + [\![x_{i+1}, x_i, x_{i-1}]\!]$$
$$= -2^3 x_{i+3} + 2^2 x_{i+2} + 2^2 x_{i+1} + (-2^1 x_{i+1} + 2^0 x_i + 2^0 x_{i-1})$$
$$= -2^3 x_{i+3} + 2^2 x_{i+2} + 2^1 x_{i+1} + 2^0 x_i + 2^0 x_{i-1} \quad (5)$$

$$[\![x_{i+3}, x_{i+2}, x_{i+1}, x_i, x_{i-1}]\!]$$
$$= [\![x_{i+3}, x_{i+2}, x_{i+1}]\!] + [\![x_{i+1}, x_i, x_{i-1}]\!]. \quad (6)$$

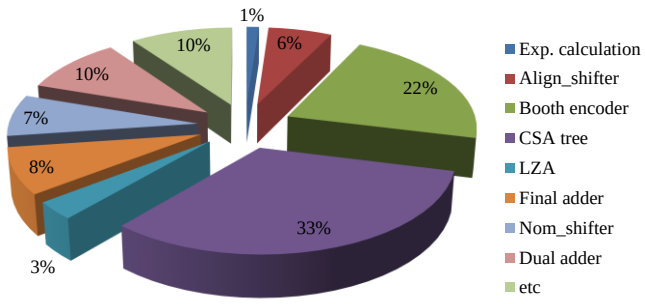

Fig. 1 Total power consumption ratio for the each module

B. Proxy bits

There are many redundant bits in floating-point arithmetic. These redundant bits has little effect on the results. In floating-point subtraction, the bits for the result out of the precision is padded with zeros. Regardless of whether the loss of accuracy is complete or partial, the phenomenon is called cancellation. Most floating-point units, however, execute an addition or subtraction for this cancellation by two times of the precision plus 1. We think these bits for the cancellation is redundant.

According to Seidel [6], pre-determined data ranges exist that can perform different operations in parallel. These pre-determined data ranges are classified into five distinct cases that comply with the IEEE 754 double-precision standards. All five cases are based on the differences in the exponents (d = Aexp + Bexp – Cexp + bias). We analyzed indispensable bits that directly affected the effective results of the FMA operations for the five distinct and propose two proxy bits which represent the redundant bits to reduce the executed bits.

In a floating-point arithmetic, the sticky bit is generated by processing a logical OR function for the right-side bits of the round bit during the normalization stage. Adopting the proxy bits generates two sticky bits. The first sticky bit (S') is only generated from the proxy bits in the normalization stage. The second sticky (S") is calculated from the remaining bits, which are not processed in the general processing but are calculated by the independent sticky-bit generator for the input data. In the rounding stage, rounding is determined by the round bits, S', S", and the rounding mode. However, the round bit can be influenced by the remaining bits when a carry propagation is performed by the remaining bits. If a carry is propagated from the least significant bits, the effective result will be reflected in the rounding module through a compensating circuit.

When proxy bits are used, the five distinct cases of [6] are described by the four cases shown in Fig. 2 because cancellation is not necessary since the proxy is a representative of the redundant bits. Each case is described below, and S' is generated by a logical OR function for the proxy bits in each case.

(a) The right-side bits of the proxy do not affect the effective result, because the exponent difference between the addend and multiplication is large. Thus, the right-side bits of the proxy do not need to be added. S" is generated from 105 bits of the multiplication result. In this case, the carry propagation from the redundant bits is not generated.

(b) Addition is required between the aligned C and multiplication result, but the right-side bits of the proxy do not need to be added. Similar to case (a), S" is generated from the multiplication result, and the carry propagation from the redundant bits is not generated.

(c) Addition is required between the aligned C and multiplication results. S" is generated from the aligned C and multiplication results. A carry propagation is detected by LZA.

(d) The significand of the effective result is the most significant 53 bits of the multiplication result, because the exponent difference between the addend and multiplication is large. However, S" is generated from the aligned C and multiplication results, and a carry propagation is detected by LZA, in contrast to case (a).

The numbers of indispensable bits in cases (a), (b), (c), and (d) shown in Fig. 2 are the most significant 56, 56, 57, and 57 bits, respectively.

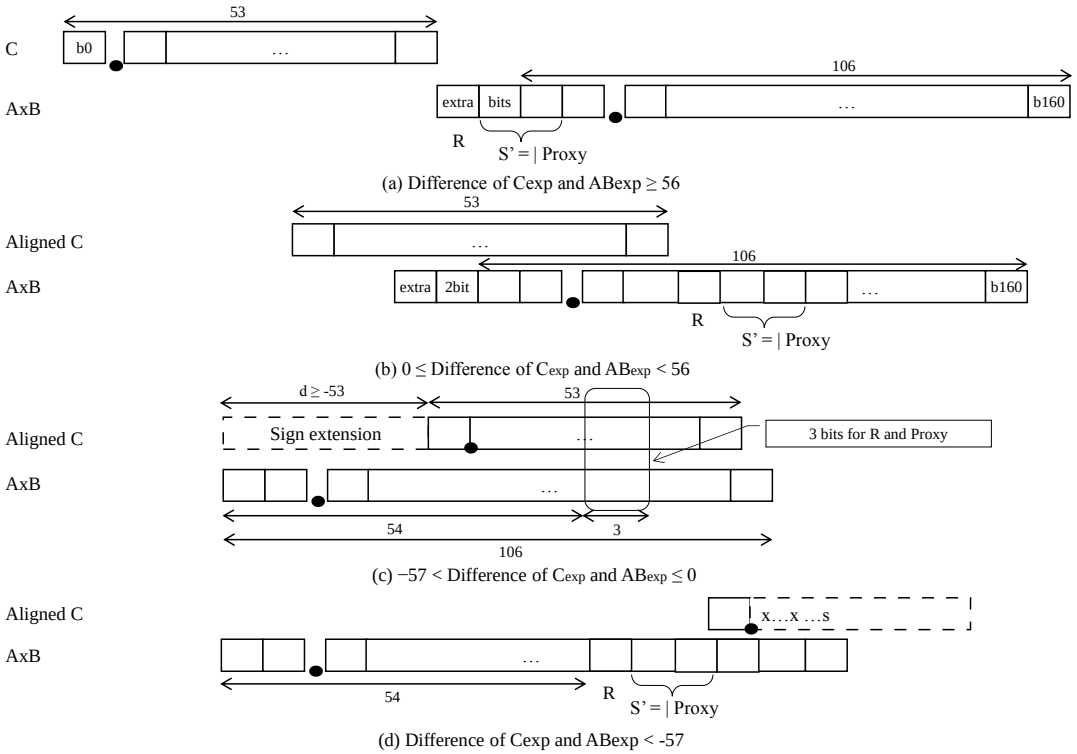

Fig. 1 Proposed four distinct cases with *proxy* bits

III. THE PROPOSED FMA UNIT

Fig. 3 shows the architecture of the proposed FMA unit which based on the dual path FMA and Lang's FMA [5]. The proposed FMA unit supports the double precision of the IEEE

754 standard. To achieve low cost and low power, a radix-16 multiplier with weighted two-level Booth encoding and proxy bits instead of many redundant bits is used.

Generally, conventional FMA units use a 161-bit final adder, a 161-bit LZA, and a 105-bit carry save adder (CSA) tree. The proposed FMA unit with the proxy bits can reduce the resources used in processing the redundant bits. For example, the 57-bit CSA, final adder, normalization shifter, and 54-bit LZA are used instead of the 161-bit module such as the CSA, final adder, LZA, and normalization shifter of the baseline FMA unit. Meanwhile, to compensate for the rounding error caused by the proxy bits, a 52-bit LZA and a sticky-bit generator are added. However, the resources added to these modules are much smaller than the ones reduced by the proxy bits, because the bit width of these modules (in which the area is proportional to the bit width) are reduced by approximately two-thirds. The proposed FMA unit also uses three input LZAs to reduce the processing time [15].

In previous chapter, the five distinct cases are reduced to four with proxy bits. These four distinct cases are described in the proposed FMA unit as follows:

- Case (a): AC[160:108] is bypassed by the result of the exponent calculation module if the difference in the exponent between C and the multiplication result is much greater than 56. In this case, R is zero, and the rounding module rounds according to the result of the sticky-bit generator and the round mode.

- Case (b): The final adder adds AC[108:52], SC[108:52], and SS[108:52] from the 57-bit CSA. The result of the final adder is concatenated to the bypassed AC[160:108]. In this case, the 54-bit LZA calculates the position of the leading one and transfers it to the normalization module. R and S' are determined in the normalization module. The rounding module determines the rounding using the round mode, R, S', and S", which are from the sticky-bit generator. The position of the carry propagation is determined from the 52-bit LZA.

- Case (c): The final adder adds AC[108:52], SC[108:52], and SS[108:52] from the 57-bit CSA. Unlike that in case (b), AC[108:52] is not concatenated to the result of the final adder. However, normalization and rounding are processed, similar to that in case (b).

- Case (d): The final adder adds both SC[108:52] and SS[108:52]. The value of AC is reflected in the rounding module through the 52-bit LZA and sticky-bit generator.

The original sticky bit is generated by the logical OR function of the LSBs from the result in the normalization module. The S' bit shown in Fig. 2 is generated by the logical OR function of the proxy bits. Thus, a sticky-bit error may be generated by the other LSBs instead of the proxy bits. We modified the sticky-bit generator to process the other LSBs and placed it parallel to the LZA and the final adder. The sticky-bit generator consists of an OR tree and generates the S" from the redundant bits that do not affect the effective result from the align shifter and the CSA tree. The result (S") from the sticky-bit generator is used to compensate the error in the round module using the logical OR with the S' bit. If an error is generated by the proxy bit, the proposed FMA unit can compensate the error using the sticky-bit generator.

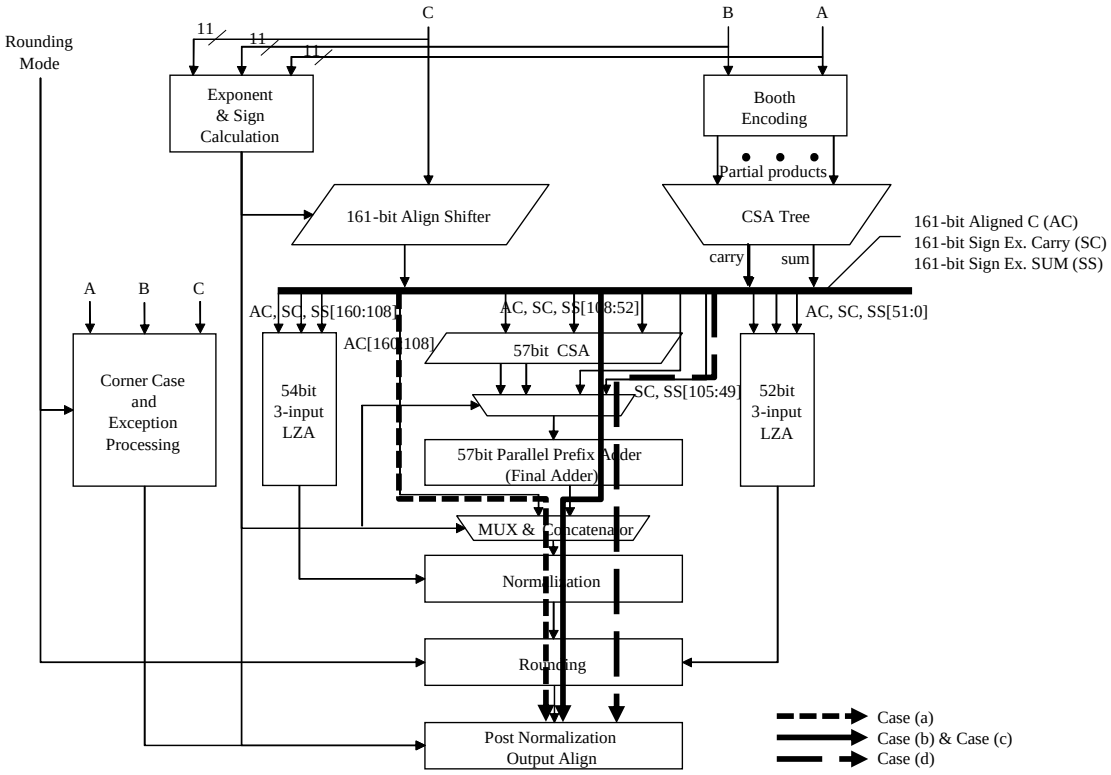

Fig. 1 Architecture of the proposed FMA unit

IV. RESULTS

The proposed floating-point FMA unit was compared to the Lang's FMA unit [5]. They were designed at the register-transfer level using HDL Verilog, and simulated by NC-Verilog of Cadence. The synthesis was carried out by the Design Compiler of Synopsys in the SAED 32-nm library.

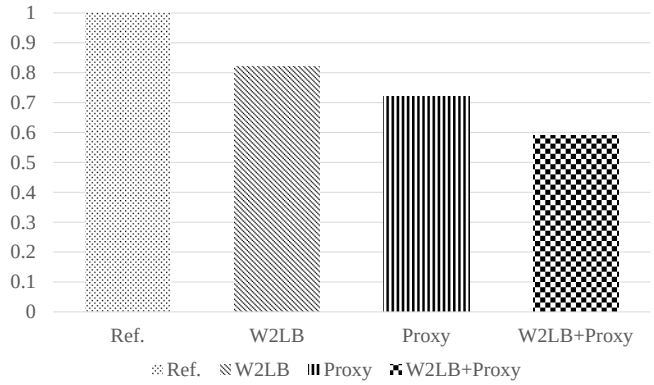

Fig. 1 Effect of the proposed FMA: normalized AD product to [5], where W2LB is weighted 2 level Booth encoding.

Fig. 4 depicts the area-delay (AD) product of Ref. 4 and the proposed FMA unit of three version, which are normalized to Ref. 4 to make it easy to compare the features of the FMA unit at a glance. The proposed three version FMAs are adopted the weighted 2 level Booth encoding, proxy bits, and both of all. The AD product a figure of merit related to the cost efficiency of a logic block and it is proportional to the power-delay (PD) product. Generally, an increase in the logic block area causes the latency to reduce and the power consumed to increase. In other words, the latency is a trade-off relationship with area and power consumption. Therefore, a lower AD product indicate more efficient area and power consumption under identical operating frequency. The proposed FMA units adopted new Booth algorithm and proxy bits shows approximately 40% more cost-efficient and power-efficient than [5]. Since redundant bits are eliminated as using proxy bits, these efficient features are induced without trade-off.

TABLE 1
SYNTHESIS RESULTS

[4]			Proposed		
Module	Latency (ns)	Area (um^2)	Module	Latency (ns)	Area (um^2)
Exponent calculator	1.15	782	Exponent calculator	1.15	782
Align shifter	0.78	3027	Align shifter	0.78	3027
Booth encoder	no critical path	5374	Booth encoder	no critical path	5633
CSA tree	no critical path	11382	CSA tree	no critical path	4770
Final adder	no critical path	3477	Final adder	no critical path	1042
			Concatenator	no critical path	2503
LZA	3.98	2964	LZA	2.91	1046
Normalization	1.13	4429	Normalization	0.72	1312
Output align.	1.78	3928	Output align	2.31	1435
Total	8.82	34363	Total	7.87	21550

Table 1 lists the synthesis results for between Ref. 4 and the proposed FMA unit under no clock constraints. In summary, the proposed FMA unit is approximately 37% smaller, and we observe an approximately 10% improvement in the performance relative to the unit described in [5] because of use of the modules adopted proxy bits. This is because the latencies of the final adder, LZA, and normalization shifter are proportional to their bit widths.

V. CONCLUSIONS

In this paper, an FMA-related architectural-level research has been studied. To optimize the structure of multiplier, new Booth algorithm is proposed. We observed many redundant bits in FMA operations at the architectural level. We found that two proxy bits could substitute for long redundant bits. The proxy bits can improve the performance and reduce resources and power consumption without trade-off among area, power consumption, and latency. The proposed FMA unit has three main advantages owing to the proxy bits. First, a low-cost FMA unit with low power consumption can be designed by eliminating the processing of redundant bits. Second, the reduced bit width improves the performance. Third, parallelism of the sticky-bit generation reduces the latency of the critical path. We modified a conventional FMA unit using proxy bits. Compared with conventional FMA unit, the proposed FMA unit reduced the total area and latency by approximately 37.0% and 10%, respectively. We expect the proposed FMA unit to utilize in mobile or high performance processing unit for floating-point.

REFERENCES

[1] E. Hokenek, R. K. Hokenek, P. W. Montoye, and Cook, "Second-generation RISC floating point with multiply-add fused," IEEE Journal of Solid-State Circuits, vol. 25, pp. 1207-1213, 1990.

[2] R. K. Montoye, E. Hokenek, and S. L. Runyon, "Design of the IBM RISC System/6000 floating-point execution unit," IBM Journal of Research and Development, vol. 34, pp. 59-70, 1990.

[3] J. D. Bruguera and T. Lang, "Floating-point fused multiply-add: reduced latency for floating-point addition," in Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on, 2005, pp. 42-51.

[4] M. S. Schmookler and K. J. Nowka, "Leading zero anticipation and detection-a comparison of methods," in Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on, 2001, pp. 7-12.

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

[6] P. M. Seidel, "Multiple path IEEE floating-point fused multiply-add," in Circuits and Systems, 2003 IEEE 46th Midwest Symposium on, 2003, pp. 1359-1362 Vol. 3.

[7] H. Libo, S. Li, D. Kui, and W. Zhiying, "A New Architecture For Multiple-Precision Floating-Point Multiply-Add Fused Unit Design," in Computer Arithmetic, 2007. ARITH '07. 18th IEEE Symposium on, 2007, pp. 69-76.

[8] H. Libo, M. Sheng, S. Li, W. Zhiying, and X. Nong, "Low-Cost Binary128 Floating-Point FMA Unit Design with SIMD Support," Computers, IEEE Transactions on, vol. 61, pp. 745-751, 2012.

[9] K. Donghyun and K. Lee-Sup, "A Floating-Point Unit for 4D Vector Inner Product with Reduced Latency," Computers, IEEE Transactions on, vol. 58, pp. 890-901, 2009.

[10] R. Samy, H. A. H. Fahmy, R. Raafat, A. Mohamed, T. ElDeeb, and Y. Farouk, "A decimal floating-point fused-multiply-add unit," in Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium on, 2010, pp. 529-532.

[11] S. Z. Gilani, K. Nam Sung, and M. Schulte, "Energy-efficient floating-point arithmetic for software-defined radio architectures," in Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE International Conference on, 2011, pp. 122-129.

[12] S. Galal and M. Horowitz, "Latency Sensitive FMA Design," in Computer Arithmetic (ARITH), 2011 20th IEEE Symposium on, 2011, pp. 129-138.

[13] K. L. S. Swee and H. Lo Hai, "Performance comparison review of Radix-based multiplier designs," in Intelligent and Advanced Systems (ICIAS), 2012 4th International Conference on, 2012, pp. 854-859.

[14] R. Riedlinger, R. Arnold, L. Biro, B. Bowhill, J. Crop, K. Duda, et al., "A 32 nm, 3.1 Billion Transistor, 12 Wide Issue Itanium® Processor for Mission-Critical Servers," Solid-State Circuits, IEEE Journal of, vol. 47, pp. 177-193, 2012.

[15] X.-L. Mei, "Leading zero anticipation for latency improvement in floating-point fused multiply-add units," in ASIC, 2005. ASICON 2005. 6th International Conference On, 2005, pp. 53-56.