Trailing-Ones Anticipation for Reducing the Latency of the Rounding Incrementer in FP FMA Units

Toru Koizumi*†, Ryota Shioya*‡, Takuya Yamauchi*, Tomoya Adachi*, Ken Namura*, and Jun Makino*
*Preferred Networks, Inc., Tokyo, Japan
Email: {tyama, adachi, karuman, jmakino}@preferred.jp
†Department of Computer Science, Nagoya Institute of Technology, Aichi, Japan
Email: koizumi@nitech.ac.jp
‡Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan
Email: shioya@ci.i.u-tokyo.ac.jp

***Abstract*—Floating-point fused multiply-add (FMA) operations are fundamental in many fields such as scientific computing, graphics processing, and machine learning. In conventional floating-point FMA designs, the internal steps of multiplication and addition, which are performed using integer arithmetic, are followed by a post-processing stage. It has been known that the post-processing stage contributes to approximately 60% of the total latency of the double-precision FMA operation. We propose a novel trailing-ones anticipation technique that predicts trailing-ones bits of the mantissa before rounding, in parallel with the post-processing stage. With this technique, the rounding incrementer can be implemented using a single XOR operation, thereby reducing the total latency. We evaluated the latency using Synopsys Design Compiler for synthesis, confirming that the proposed technique reduced total latency by 4%.**

***Index Terms*—High-Speed Arithmetic**

I. Introduction

Fused multiply-add (FMA) operations are fundamental in many fields such as scientific computing, machine learning, and graphics. Despite their widespread adoption, designing an efficient FMA unit is significantly more difficult than implementing standalone addition or multiplication units. These challenges have driven extensive research efforts to refine FMA unit designs toward higher performance.

Our goal is to reduce the latency in FMA units. Figure 1 shows the architecture of a conventional FMA unit [1]. The figure focuses on the mantissa calculation, which constitutes the critical path. The computation is performed in the following steps: (1a) The partial products of the mantissa of the multiplicand (Mul-A) and multiplier (Mul-B) are generated. (1b) In parallel, the digits of the addend are shifted using the align shifter. (2) The partial products and the shifted addend are combined in a Wallace tree, and the result is obtained in a redundant binary representation. (3a) The redundant binary result is summed using a parallel prefix (dual) adder to obtain the absolute value in binary representation. (3b) In parallel, the number of leading zeros in the absolute value is predicted with at most one error using the leading zeros count anticipator (LZA). (4) The absolute value is shifted by the predicted number of digits using the normalizing shifter. (5) If there is an error in the predicted shift amount, it is corrected using the LZA error correction unit. (6) The rounding direction is determined. (7) If away-from-zero rounding is required, 1 is added to the result using the rounding incrementer.

In this design, optimizing the post-processing stages (stages (3a) and beyond) is important. Although the multiplier used in (1a) occupies a significant portion of the circuit area, the post-processing stages, despite their smaller footprint, make a notable contribution to the overall latency. Specifically, the latency arises from stages (3a/3b), (4), and (7), each requiring circuits with logic depth proportional to the logarithm of the mantissa bit width (N). Furthermore, these stages are sequentially dependent, increasing the overall latency. As a result, the post-processing stages contribute to a significant part of the overall latency of this design. According to Srinivasan *et al.*, the post-processing latency accounts for approximately 60% of the FMA calculation time [2].

Improvements of these post-processing stages have been widely studied. Lang *et al.* demonstrated a novel usage of a dual adder to serve both the computation of absolute values (taking the two's complement) and the addition of 1 in the rounding incrementer [3]. Lutz proposed methods to apply LZA even when the result is a denormal number and introduced an approach to determine errors in the shift amount at an early stage [4]. Sohn *et al.* showed that the computation of the absolute value and the addition of 1 can be shared without employing dual adder techniques. They also presented a method for handling denormal number inputs with hardly increasing the latency [5].

However, these techniques do not alter a fundamental characteristic of the post-processing stages; each of the stages (3a/3b), (4), and (7) has a logic depth of approximately $\log N$, resulting in a total logic depth of about $3 \log N$.

To address these challenges, we propose *trailing-ones anticipation (TOA)* designed to perform additional computations in parallel with stages (3a/3b) to (6). By leveraging the capabilities of TOA, the logic depth of stage (7) is effectively reduced to $O(1)$. Consequently, the total logic depth of the post-processing stages is reduced to approximately $2 \log N$. Our evaluation indicates that while the introduction of TOA increases the circuit area, it reduces the overall latency of the post-processing stages by 4%.

The structure of this paper is as follows. Section II provides

2576-2265/25/$31.00 ©2025 IEEE
DOI 10.1109/ARITH64983.2025.00014

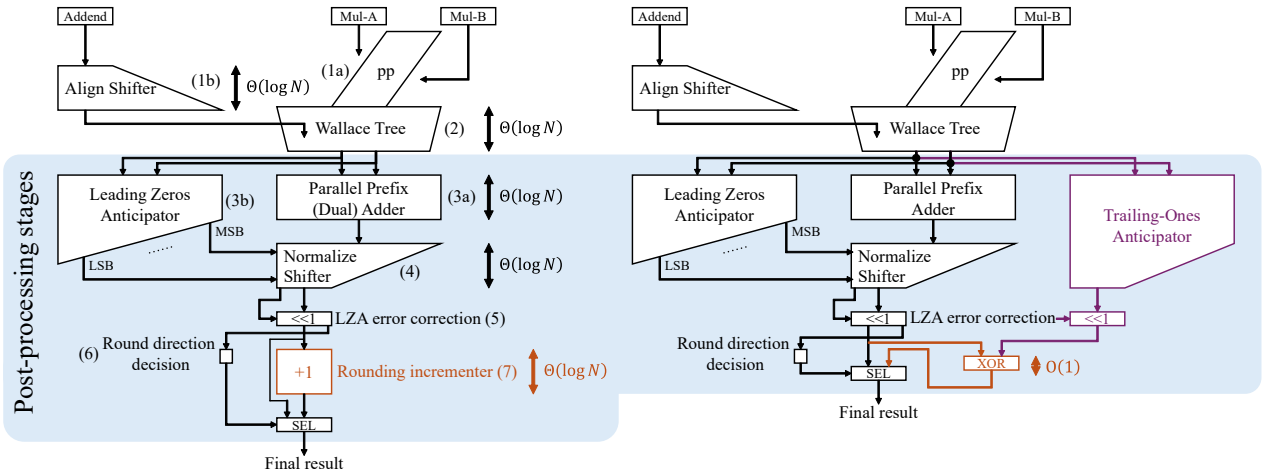


Fig. 1. Floating-point fused multiply-add unit architectures. Our proposed trailing-ones anticipator effectively reduces the rounding incrementor delay to $O(1)$.

a review of the conventional FMA architecture. In Section III, we describe the proposed method, trailing-ones anticipation, and Section IV presents its evaluation.

II. Conventional FMA Architecture

A. Partial product generator (1a)

The partial product generator produces partial products from the mantissa of the multiplier and multiplicand. To optimize this process, Radix-4 Booth encoding is commonly employed, as it reduces the number of partial products, thereby minimizing both latency and circuit area. Although the latency of the partial product generator, which includes the Booth encoder and selector, is often assumed to be $O(1)$, it is actually $\Theta(\log N)$. This latency arises from the need for multiple buffers to handle high-fanout signals, such as the selection signals.

To further reduce circuit area, higher-radix Booth encoding methods, such as radix-8 and radix-16, are sometimes employed [5]. However, these methods require hard multipliers to compute values that cannot be generated by simple shifting, such as multipliers that calculate three, five, or seven times the multiplicand. The delay of these hard multipliers is $\Theta(\log N)$.

The input to the Booth encoder depends on whether the multiplier or multiplicand is a normalized or denormalized number. Consequently, determining whether the number is normalized is on the critical path. Sohn *et al.* proposed a speculative approach that assumes a number is normalized, begins the computation, and applies corrections if the assumption proves incorrect. This approach effectively eliminates the determination of whether the number is a normalized number from the critical path [5].

B. Addend alignment shifter (1b)

The addend alignment shifter aligns the mantissa of the addend to the multiplier output. The shift operation is performed sequentially, controlled by each bit of the shift amount, from the least significant bit (LSB) toward the most significant bit (MSB). This approach optimizes both delay and circuit area. The circuit comprises $\log_2(3N+O(1))$ 2:1 selectors arranged in series, resulting in a delay time of $\Theta(\log N)$.

C. Wallace tree (2)

The Wallace tree adds all the partial products generated in (1a) with the aligned addend generated in (1b), outputting the result in a redundant binary representation. A Wallace tree is constructed from numerous 3:2 carry-save adders (CSAs). To optimize overall delay, the aligned addend from (1b) is added after partial products from (1a) have been partially summed. The critical path of this circuit involves the application of 3:2 CSAs to reduce the number of partial products to 2, resulting in a delay time of $\Theta(\log N)$.

While a Wallace tree is used as a representative example here, many alternative network structures are available. Depending on factors such as delay and wiring length, a different network structure may provide better performance [6].

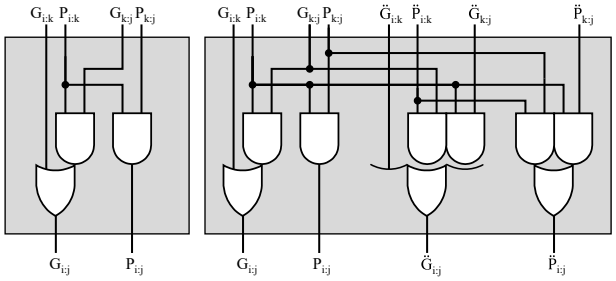
D. Parallel prefix dual adder (3a)

First, we will describe a simple parallel prefix adder (PPA) instead of a parallel prefix dual adder.

1) Parallel prefix adder: The parallel prefix adder computes $G_{i:0}$ $(0 < i \leq N)$ using the propagate (P) and generate (G) signals, which are defined as follows.

$$P_{i:j} = \bigwedge_{k=j}^{i-1} a_k \veebar b_k \tag{1}$$

$$G_{i:j} = \bigvee_{k=j}^{i-1} P_{i:k+1} \wedge a_k \wedge b_k \tag{2}$$



(a) Conventional prefix box. (b) Proposed prefix box.

Fig. 2. Prefix boxes

Intuitively, $P_{i:j}$ means that if a carry is generated in the j-th digit, it propagates through to the i-th digit. Similarly, $G_{i:j}$ means that based solely on the digits from j to $i-1$, it can be determined that a carry is generated in the i-th digit. For any k such that $i > k > j$, the following equations hold:

$$P_{i:j} = P_{i:k} \wedge P_{k:j} \tag{3}$$

$$G_{i:j} = G_{i:k} \vee (P_{i:k} \wedge G_{k:j}) \tag{4}$$

Using this formula, the circuit area can be optimized by extracting the common parts from the computation process of $G_{i:0}$ $(0 < i \leq N)$. The circuit implementing equations (3) and (4) is shown in Figure 2(a) and is referred to as a prefix box.

It is well known that circuit area, fan-out, and logic depth cannot be simultaneously optimized [7]. The Kogge-Stone configuration is recognized for compromising on circuit area, the Sklansky configuration for compromising on fan-out, and the Brent-Kung configuration for compromising on logic depth. Additionally, many intermediate configurations that balance these trade-offs are also known [7].

This circuit of the simple PPA requires at least $\log_2 N$ prefix boxes in series, resulting in a delay time of $\Theta(\log N)$.

2) Parallel prefix dual adder: The parallel prefix dual adder is a high-speed binary adder implemented using end-around-carry, capable of simultaneously computing $A + B$ and $A + B + 1$ [8]–[10]. This functionality is essential when taking the two's complement of a result that is negative. The dual adder achieves this by computing not only $G_{i:0}$ but also $P_{i:0}$ $(0 < i \leq N)$. Since the circuits required for these calculations are largely shared, the increase in circuit area is minimal.

3) Simplified Two's Complement Handling in a PPA: In contrast, Sohn *et al.* proposed a method that requires only a simple PPA, eliminating the need for a dual adder by utilizing a rounding incrementer [5]. This method leverages the fact that the increment for rounding and the increment for taking the two's complement occur mutually exclusively, thereby removing the need to compute the two's complement within the PPA. Instead, computing the ones' complement is sufficient within the PPA.

E. Leading zeros count anticipator (LZA) (3b)

LZA is a circuit that computes the leading zeros count of $A+B$ with an error of at most 1, without requiring the binary representation of $A + B$. In practice, a circuit known as an absolute leading zeros count anticipator is used to compute the leading zeros of $|A + B + 1|$ with the same conditions [11].

This circuit consists of two components: a circuit that computes the predicted value L and a circuit that determines the leading zeros count of L. The circuit for computing the predicted value L is an $O(1)$-stage circuit that outputs a value satisfying the following conditions, with a possible single-bit position error:

- In the leading zeros portion of $|A + B + 1|$: 0
- In the digit where the most significant 1 of $|A + B + 1|$ appears: 1
- In the digits below that: arbitrary

The delay time to output the MSB of the predicted value of the leading zeros count is $\Theta(\log N)$, as it requires a NOR operation with an input size of $3N + O(1)$. Additionally, to output the LSB of the predicted value, a stage count approximately twice that of the MSB is required. To reflect this delay, the LZA is depicted in Figure 1 (3b) with a diagonal cutout. The critical path is not determined solely by the LZA but is influenced by its combination with the normalizing shifter, as described below.

Lutz proposed a method for generating predicted values to handle irregular shifts when the result is a denormalized number. Specifically, this is achieved by setting a 1 at a specific bit position to prevent additional shifts [4].

F. Normalizing shifter (4)

The normalizing shifter is a circuit that shifts the absolute value computed in (3a) by the leading zeros count predicted in (3b). Delay and circuit area can be optimized by shifting sequentially according to each bit of the shift amount, starting from the MSB. This circuit consists of $\log_2(3N + O(1))$ 2:1 selectors arranged in series, resulting in a delay time of $\Theta(\log N)$.

G. LZA error correction (5)

Since the LZA may output a shift amount that is one less than the correct value, the LZA error correction circuit shifts the output from (4) by one additional bit if its first bit is 0. The delay of this circuit is often considered $O(1)$ from a pure logic perspective. However, in practice, it is actually $\Theta(\log N)$ because multiple buffers are required to drive high-fanout selector signals for the shift.

To address this issue, Lutz proposed a method for determining early whether this adjustment is necessary [4]. In the method, the bit corresponding to the first bit of the output of (4), the selection signal, is extracted using an OR tree in parallel with the shift operation in (4). To achieve this, a mask is created using a one-hot vector with only a bit corresponding to the first bit of (4). This process can be performed in parallel with (3a/3b), with a delay time of $\Theta(\log N)$.

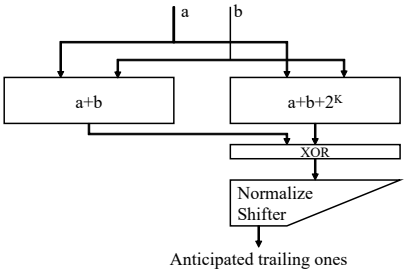


Fig. 3. Overview of trailing-ones anticipation

H. Round decider (6)

The round decider is a circuit that examines the least significant part of the bit sequence obtained in (5) to determine whether the number should be rounded away or toward zero. While the delay time of this circuit is $O(1)$ with respect to the bit width, its complexity may place it on the critical path.

Lutz proposed a method to determine the rounding direction early by extracting the required bits like that described in the previous section [4].

I. Rounding incrementer (7)

The rounding incrementer adds 1 to the mantissa when stage (6) determines that rounding away from zero is required. The delay of this circuit is $\Theta(\log N)$.

III. Proposed method — trailing-ones anticipation

A. Key idea

We propose a trailing-ones anticipator (TOA), a circuit designed to predict the trailing ones of the output from a normalizing shifter, thereby reducing the latency of the rounding incrementer. Figure 1(b) presents a simplified block diagram of the proposed FMA circuit incorporating TOA. TOA significantly reduces the delay of the rounding incrementer compared to the conventional design shown in Figure 1(a).

The key idea relies on the observation that if the trailing ones of a value are known, the addition of 1 can be performed using a simple XOR operation. For example, consider incrementing the 4-bit value $0\mathrm{b}1011$, which has trailing ones $0\mathrm{b}0011$. By XORing the original value with the result of shifting a 1-bit to the left ($0\mathrm{b}0110$) and then inverting the least significant bit (LSB) of the result, the incremented value $0\mathrm{b}1100$ is obtained. This process intuitively demonstrates that the carry resulting from adding 1 propagates only to higher-order bits within the trailing-ones section.

B. Overview

The circuit overview implementing TOA is shown in Figure 3. A special adder is introduced to compute $a + b + 2^K$ for inputs a and b in redundant binary representation. Here, 2^K corresponds to the power of two associated with the least significant bit of the mantissa after shifting by the amount predicted by the LZA. This special adder directly computes $a + b + 2^K$ using a parallel prefix network, without explicitly calculating K or 2^K. By XORing $a + b + 2^K$ with $a + b$ and then performing a normalization shift, the anticipated trailing ones are obtained.

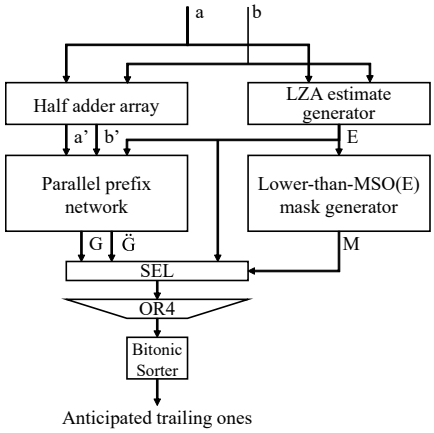


Fig. 4. Architecture of trailing-ones anticipator

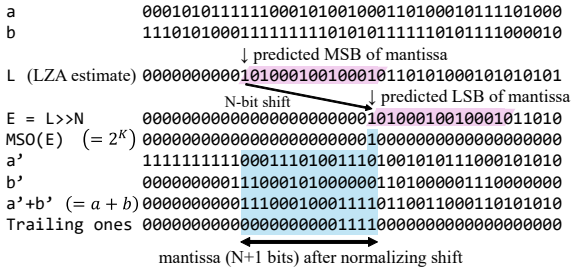


Fig. 5. The computation example of trailing ones

The special adder used in TOA differs from the existing dual adder described in Section II-D2. While that dual adder is designed to compute both $a+b$ and $a+b+1$ at the same time, it can only add 1 to the number *before* shifting. Consequently, it cannot compute $a+b+2^K$, which corresponds to adding 1 to the number *after* shifting. The core contribution of this paper is the development of this special adder, which is capable of computing $a + b + 2^K$.

C. Architecture of trailing-ones anticipator

Figure 4 illustrates the detailed circuit architecture of TOA, while Figure 5 provides an example bit sequence used in the subsequent explanation. First, TOA uses an $O(1)$-stage circuit to generate an estimate E, which approximates 2^K. In parallel, a and b are processed through a half-adder array to convert them into a' and b'. Next, $a'+b'+\mathrm{MSO}(E)$ is computed using a parallel prefix network that incorporates a custom prefix box. Here, $\mathrm{MSO}(x)$ denotes a binary number in which all bits except the most significant 1 of x are set to 0. The result $a' + b' + \mathrm{MSO}(E)$ represents the desired intermediate output. The following sections provide a detailed explanation of each module.

D. Generation of estimating value of 2^K

The procedure for obtaining 2^K poses challenges in reducing total latency. This is because computing 2^K concretely requires a circuit with a logical depth of $\Theta(\log N)$, where N is the bit width of the mantissa. To address this issue, we adopt an approach that uses $E = L \gg N$ as a surrogate for 2^K. Here, we leverage the fact that the predicted value L output by the LZA closely approximates 2^K when shifted N bits to the right.

E. Half adder array

For general inputs a and b, it is not possible to compute $a + b + \mathrm{MSO}(E)$ using existing parallel prefix adders. This limitation arises because two separate carries can be generated in the same digit.

To address this issue, a half-adder array is employed to transform a and b into two values, a' and b', such that $a+b = a' + b'$. By passing $a + b$ through the half-adder array, the following inequality is satisfied for any i:

$$\sum_{k=0}^{i} 2^k (a'_k + b'_k) < 3 \cdot 2^i \tag{5}$$

This ensures that when $a' + b' + \mathrm{MSO}(E)$ is computed, the carry from any digit will be at most 1 (and will not be 2). This is guaranteed because the following inequality holds:

$$\sum_{k=0}^{i} 2^k (a'_k + b'_k + \mathrm{MSO}(E)_k) < 4 \cdot 2^i \tag{6}$$

F. Prefix network

The proposed method utilizes a prefix network with a prefix box that computes the following $\ddot{P}$ and $\ddot{G}$ in addition to P and G. The structure of this prefix box is illustrated in Figure 2(b).

$$\ddot{P}_{i+1:i} = a_i \veebar b_i \veebar E_i \tag{7}$$

$$\ddot{G}_{i+1:i} = (a_i \wedge b_i) \vee (b_i \wedge E_i) \vee (E_i \wedge a_i) \tag{8}$$

$$\ddot{P}_{i:j} = (\ddot{P}_{i:k} \wedge P_{k:j}) \vee (P_{i:k} \wedge \ddot{P}_{k:j}) \tag{9}$$

$$\ddot{G}_{i:j} = \ddot{G}_{i:k} \vee (\ddot{P}_{i:k} \wedge G_{k:j}) \vee (P_{i:k} \wedge \ddot{G}_{k:j}) \tag{10}$$

Intuitively, $\ddot{G}_{i:j}$ indicates a generate signal when $p_{\mathrm{MSO}}(E)$ is between bit positions $i-1$ and j, and similarly, $\ddot{P}_{i:j}$ indicates a propagation signal under the same condition. Here, $p_{\mathrm{MSO}}(X)$ represents the position of the most significant 1 in X.

The right-hand sides of the equations for $\ddot{P}$ and $\ddot{G}$ are formed by summing terms derived from the corresponding equations for P and G, with exactly one variable in each term replaced by its umlaut version. If a term contains multiple variables, it is split into multiple terms, each with exactly one variable modified. The resulting equation cannot contain a term with more than one umlaut-modified variable (such as $\ddot{P}_{i:k} \wedge \ddot{G}_{k:j}$) because $p_{\mathrm{MSO}}(E)$ corresponds to a single bit position and thus it cannot simultaneously exist between both i and k and between k and j.

G. Lower-than-MSO(E) mask

A lower-than-MSO(E) mask M can be generated by the following simple procedure.

```
M = E >> 1;
for shamt in { 1, 2, 4, 8, 16, ... }:
  M |= M >> shamt;
```

Note that the *smask* that Lutz introduced [4] is equivalent to this. In addition, the one-hot mask described by Lutz [4] can be calculated by `E&~M` and is equivalent to $\mathrm{MSO}(E)$.

H. Select G/$\ddot{G}$

The generate signal for $a' + b' + \mathrm{MSO}(E)$ is defined as $G_{i:0}$ when $i < p_{\mathrm{MSO}}(E)$, and $\ddot{G}_{i:0}$ otherwise. In the following, we call this G^e. Using the lower-than-MSO(E) mask, this can be expressed as `(G&M)|(Ğ&~M)`.

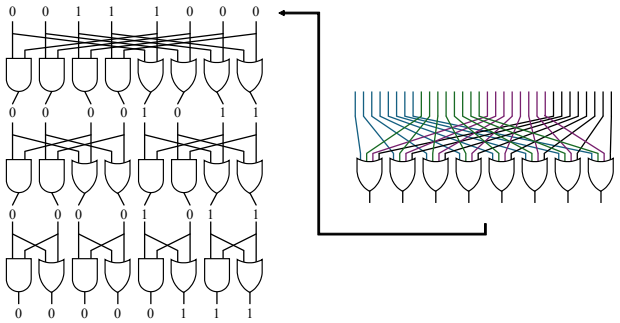
I. Anticipated trailing ones

The XOR of $a'+b'$ and $a'+b'+\mathrm{MSO}(E)$ gives the predicted value of trailing ones. $a'+b'$ can be expressed as $a' \oplus b' \oplus (G \ll 1)$ and $a' + b' + \mathrm{MSO}(E)$ can be expressed as $a' \oplus b' \oplus \mathrm{MSO}(E) \oplus (G^e \ll 1)$. $\mathrm{MSO}(E)$ is `E&~M`. Thus, the predicted value of trailing ones, before normalization shifting, can be determined using `(E&~M)|(((G^Ğ)&~M)<<1)`. The value of $a + b + 2^K$ itself does not need to be explicitly computed.

J. Bitonic sort-based shifter

The value output above is a bitonic sequence, which enables a highly efficient normalization shift. A bitonic sequence is defined here as a number represented by the regular expression $0^*1^*0^*$. For such a bitonic sequence, bitonic sorting can be performed to shift the 1s to the right in $\Theta(\log N)$ steps. Since trailing ones, by their nature, start from the least significant end, sorting is functionally equivalent to normalization shifting. Therefore, the normalization shifter for general numbers can be replaced with a bitonic sorter.

The bitonic sorter is the circuit shown in Figure 6(a). It should be noted that the number of stages can be halved by using logic elements such as NAND4 and OAI22 without increasing transistor count.



(a) Bitonic sorter (b) OR4 preprocess

Fig. 6. Bitonic sorter-based shifter

The width of the sorting circuit can be reduced. This circuit can correctly sort inputs such as $1^*0^*1^*$ as well as $0^*1^*0^*$. In addition, the input ($4N+O(1)$ bits) does not contain more than $N+1$ 1 bits. Therefore, the same result can be obtained by dividing the output value into four parts, performing a bitwise OR on them, followed by sorting, as illustrated in Figure 6(b).

The whole trailing-ones anticipator circuit is faster than the other paths due to the very lightweight bitonic sorter-based shifter circuit. The parallel prefix adder inside the trailing-ones anticipator has a longer latency than the main parallel prefix adder because of the prefix box complexity. Still, the total latency is shorter than the other paths thanks to the low-latency shifter circuit.

K. Anticipation-error correction

The predicted values of the trailing-ones output above may require correction. The incorrect predictions come from the fact that the predicted value L in the leading zeros count anticipator may be off by one bit. If the predicted value may have an error of one bit, it means that we predicted the trailing ones from the second last place of the mantissa before rounding. This can be dealt with by classifying the cases according to whether the pre-rounded mantissa is even or odd, or in other words, whether the last place is 0 or 1, as follows.

If the last place of the mantissa before rounding is 0, +1 can be achieved by simply changing the bit at this position to 1. In this case, the predicted trailing ones are not used.

If the last place of the mantissa before rounding is 1, then +1 can be achieved by setting the last position to 0 and performing an XOR operation on the other parts with the anticipated trailing ones shifted 1 bit to the left.

L. Negative case handling

Calculating $a+b+\mathrm{MSO}(E)$ by itself is insufficient to calculate trailing ones when $a+b$ is negative; therefore, it is also necessary to calculate $-a-b+\mathrm{MSO}(E)$. We have been unable to find a circuit configuration that can simultaneously compute both $a+b+\mathrm{MSO}(E)$ and $-a-b+\mathrm{MSO}(E)$. Research into such circuit configurations will be the subject of future work. In this paper, we address this problem by presenting a circuit that calculates $\tilde{a}+\tilde{b}+\mathrm{MSO}(E)$, in addition to a circuit that calculates $a+b+\mathrm{MSO}(E)$.

M. Whole architecture

Figure 7 shows the whole architecture of the trailing-ones anticipator and final mantissa calculation. LZA error correction, selection with special values, e.g., Inf or NaN, and rounding selection are all critical paths. We found that the latency could be optimized by performing these steps in the order shown in the figure.

First, XOR the output of the normalization shifter with the anticipated trailing ones. Even if the LZA is mispredicted, the combination of bit positions to be XORed remains the same, thus the XOR can be performed before the LZA correction. Next, correct the LZA error. This procedure is performed

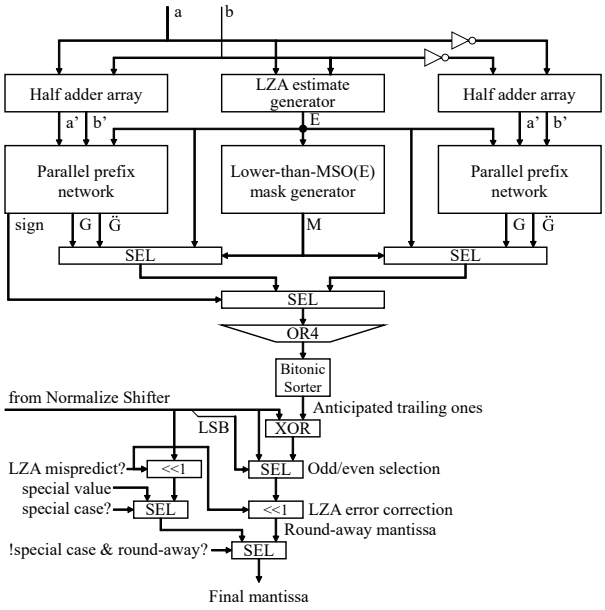


Fig. 7. Whole architecture of trailing-ones anticipator and final mantissa calculation

second, considering the latency involved in buffering signals with high fan-out. In parallel, the special values are selected on the low-delay side. Finally, the selection is made according to the rounding direction. The rounding direction takes longer to determine and has more fan-out; thus the selection using it is done last.

IV. Evaluation

A. Methodology

To evaluate the effectiveness of the proposed method, we synthesized IEEE 754-compliant double-precision floating-point FMA circuits for multiple target frequencies, then analyzed their latency and area. The FMA circuits, based on both the existing and proposed techniques, were implemented in SystemVerilog. The synthesis was performed using Synopsys Design Compiler with a standard cell library derived from the ASAP7 7nm finFET predictive process design kit (PDK) [12].

We implemented the following two FMA designs based on existing technology to compare with the proposed method. (1) First, we implemented the FMA design presented in [5], which is described as a design for the Intel E-Core processor. This design is oriented towards saving circuit area and was not intended to pursue high-speed operation. (2) Second, building on the above design, we implemented an improved version aimed at achieving higher speed. Specifically, we replaced Radix-16 Booth encoding by Radix-4 Booth encoding. Furthermore, we improved the LZA error correction and rounding decision circuits using Lutz's mask method, as described in Sections II-G and II-H.

In our evaluation, the following components were automatically generated using Synopsys Design Ware. (1) For the

Wallace tree part: we observed that unconventional circuits were generated by the tool to optimize latency. (2) For the leading zeros count part: the generated circuits seemed to be faster than the LZC circuits described in [5], but slightly larger in area. (3) For all adders: it was observed that unconventional circuits were generated by the tool to optimize latency, in particular, for the main parallel prefix adder.

Since the prefix network used in the proposed trailing-ones anticipator cannot be automatically synthesized by Design Ware, we manually implemented Kogge-Stone, Sklansky, and Brent-Kung networks and evaluated their performance.

B. Result

Figure 8 presents the evaluation results of latency and area under various timing constraints. The results indicate that the FMA unit using the proposed method achieves a latency reduction of over 30 ps compared to models employing existing acceleration methods. However, the circuit area increased by 26% when the PPA within the proposed method was implemented using the Brent-Kung or Sklansky configurations. Although the Kogge-Stone configuration is generally regarded as fast, its use in the PPA within the proposed method did not yield additional latency improvement and unnecessarily increased the circuit area. There was no significant difference in area between the Brent-Kung and Sklansky configurations. This appears to be due to the optimization in Design Compiler, which significantly altered the network structure.

Table I shows a breakdown of the critical path latency from the synthesis results. The proposed method significantly reduced latency by replacing the rounding incrementer with a simple XOR operation. Additionally, we confirmed that the trailing-ones anticipation circuit was not on the critical path.

Table II shows the synthesis results for the area of each module. Modules added by the proposed method are highlighted in bold. The core of the proposed method, the special PPA, is approximately 3.4 times larger than a standard PPA. This increase can be attributed to two factors: the presence of two PPAs and the larger size of the prefix box. On the other hand, the bitonic sorter-based shifter is approximately one-fourth the size of a conventional shifter, which aligns with predictions based on its circuit structure.

V. Related work

Even and Seidel proposed an injection-based rounding method for floating-point multipliers [13]. Their method achieves fast rounding by adding an appropriate value, determined by the rounding mode, to the product before the parallel prefix dual adder. The correct value to add differs based on whether the normalized result is less than two or not, but employing the dual adder and a selector can absorb the difference. These approaches share similarities with our proposed method.

A key difference between multiplication and FMA is whether the position of the least significant bit after normalization can be known before addition. In multiplication, the position of the least significant bit is known with at most a one-bit error. By contrast, in the FMA, this position cannot be determined before the addition. Therefore, determining the appropriate value to add is not straightforward, motivating the development of the special adder.

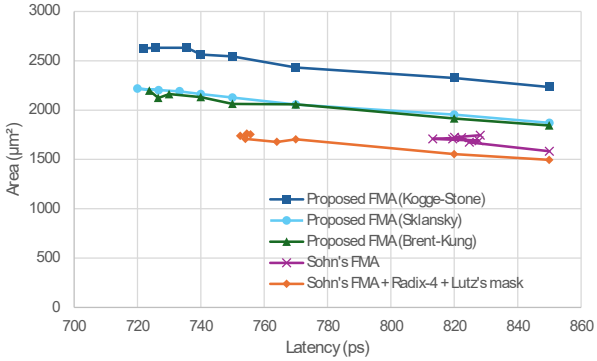


Fig. 8. Latency and Area

Conversely, our special adder is limited to adding numbers that are powers of two. This leaves implementing injection-based rounding methods requiring other values as an open problem for FMA units.

VI. Conclusion

The FMA operation is one of the most commonly used floating-point operations, and significant efforts have been made over the years to accelerate its implementation. However, the post-processing stages consist of three stages arranged in series, each requiring a latency of approximately $\log N$, and have not seen direct improvements until now.

To address this issue, we proposed a technique named trailing-ones anticipation. The core innovation of this technique is a special adder capable of computing $a + b + 2^K$. With this adder, we achieved a reduction in the post-processing stages to about two $\log N$ stages. Furthermore, since TOA is largely orthogonal to existing techniques, it can be combined with them to achieve further optimizations.

Our evaluation demonstrated that the proposed method reduces latency by about 4%, with a 26% increase in circuit area.

References

[1] S. D. Trong, M. Schmookler, E. M. Schwarz, and M. Kroener, "P6 binary floating-point unit," in *18th IEEE Symposium on Computer Arithmetic (ARITH '07)*, 2007, pp. 77–86.

[2] S. Srinivasan, K. Bhudiya, R. Ramanarayanan, P. S. Babu, T. Jacob, S. K. Mathew, R. Krishnamurthy, and V. Errgauntla, "Split-path fused floating point multiply accumulate (FPMAC)," in *2013 IEEE 21st Symposium on Computer Arithmetic*, 2013, pp. 17–24.

[3] T. Lang and J. Bruguera, "Floating-point fused multiply-add with reduced latency," in *Proceedings. IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 2002, pp. 145–150.

[4] D. R. Lutz, "Optimized leading zero anticipators for faster fused multiply-adds," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, 2017, pp. 741–744.

	Sohn's FMA [5]	Sohn's + Radix-4 + Lutz's mask	Proposed FMA
Exponent Difference & Alignment	Exponent adder (95) Alignment shifter (135) Complement (15)	Exponent adder (85) Alignment shifter (135) Complement (15)	Exponent adder (95) Alignment shifter (140) Complement (15)
Multiplier	(rest of) Wallace tree (155)	(rest of) Wallace tree (80)	(rest of) Wallace tree (100)
Main Adder		PPA (120) Complement (55)	
LZA	LZA input generation (40) LZC (75)		LZA input generation (70) LZC (70)
Normalization	Normalize shifter (115)	Normalize shifter (95)	Normalize shifter (155)
Rounding	Adjust shifter (40) Incrementer (95) Result selector (10)	Adjust shifter (20) Incrementer (95) Result selector (10)	Adjust shifter (20) XOR (10) Result selector (10)
Other	Flip-flop setup (40)	Flip-flop setup (40)	Flip-flop setup (35)
Total	810	750	**720 (fastest)**

TABLE I

LATENCY COMPARISON. THE UNIT ps IS OMITTED. VALUES REPRESENT SYNTHESIS RESULTS, AND THE LATENCY FOR THE SAME MODULE MAY VARY DUE TO OPTIMIZATION TO MEET TIMING CONSTRAINTS.

	Sohn's FMA [5]	Sohn's + Radix-4 + Lutz's mask	Proposed FMA
Exponent Difference & Alignment	Exponent adder (30) Alignment shifter (120) Complement (25) Sticky (10)	Exponent adder (30) Alignment shifter (120) Complement (25) Sticky (10)	Exponent adder (30) Alignment shifter (120) Complement (25) Sticky (10)
Multiplier	Hard multiplier (145) Booth selector (540) Denormal adjust (5) Wallace tree (300)	Booth selector (380) Denormal adjust (5) Wallace tree (455)	Booth selector (370) Denormal adjust (5) Wallace tree (430)
Main Adder	PPA (150) Complement (25)	PPA (120) Complement (30)	PPA (115) Complement (30) **Special PPA (410)**
LZA	LZA input generation (100) LZC (40)	LZA input generation (100) LZC (40) Lutz's mask generation (55)	LZA input generation (100) LZC (40) Lutz's mask generation (55)
Normalization	Normalize shifter (100) Exponent adder (5) Sticky/All ones (10)	Normalize shifter (100) Exponent adder (5) Sticky/All ones (10) Lutz's mask tree (90)	Normalize shifter (85) Exponent adder (5) Sticky/All ones (10) Lutz's mask tree (85) **Bitonic sorter-based shifter (25)**
Rounding	Adjust shifter (10) Round decision (5) Incrementer (25) Result selector (20)	Adjust shifter (10) Round decision (5) Incrementer (25) Result selector (20)	Adjust shifter (15) Round decision (5) **XOR (10)** Result selector (20)
Other	Special case handler (25) Exception flags (5)	Special case handler (25) Exception flags (5)	Special case handler (25) Exception flags (5)
Total	1695	1665	2030

TABLE II

AREA COMPARISON. THE UNIT μm^2 IS OMITTED. VALUES REPRESENT SYNTHESIS RESULTS, AND THE CIRCUIT AREA FOR THE SAME MODULE MAY VARY DUE TO OPTIMIZATION TO MEET TIMING CONSTRAINTS.

[5] J. Sohn, D. K. Dean, E. Quintana, and W. S. Wong, "Enhanced floating-point multiply-add with full denormal support," in *2023 IEEE 30th Symposium on Computer Arithmetic (ARITH)*, 2023, pp. 143–150.

[6] S. Galal, O. Shacham, J. S. Brunhaver II, J. Pu, A. Vassiliev, and M. Horowitz, "Fpu generator for design space exploration," in *2013 IEEE 21st Symposium on Computer Arithmetic*, 2013, pp. 25–34.

[7] D. Harris, "A taxonomy of parallel prefix networks," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, vol. 2, 2003, pp. 2213–2217.

[8] A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. Lim, "Reduced latency ieee floating-point standard adder architectures," in *Proceedings 14th IEEE Symposium on Computer Arithmetic*, 1999, pp. 35–42.

[9] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Addison-Wesley Publishing Company, 2010.

[10] V. G. Oklobdzija and R. K. Krishnamurthy, *High-performance energy-efficient microprocessor design*. Springer Science & Business Media, 2007.

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

[12] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002626921630026X

[13] G. Even and P.-M. Seidel, "A comparison of three rounding algorithms for ieee floating-point multiplication," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 638–650, 2000.