Leading-Zero Anticipatory Logic for High-Speed Floating Point Addition

Hiroaki Suzuki, Hiroyuki Morinaka, Hiroshi Makino, Yasunobu Nakase, Koichiro Mashiko, *Member, IEEE,* and Tadashi Sumi

Abstract—This paper describes a new leading-zero anticipatory (LZA) logic for high-speed floating-point addition (FADD). This logic carries out the pre-decoding for normalization concurrently with addition for the significand. It also performs the shift operation of normalization in parallel with the rounding operation. The use of simple Boolean algebra allows the proposed logic to be constructed from a simple CMOS circuit. Its area penalty is as small as 30% of the conventional LZA method. The FADD core using the proposed logic was fabricated by 0.5 μm CMOS technology with triple metal interconnections and runs at 164 MHz under the condition of $V_{DD} = 3.3$ V.

I. INTRODUCTION

DEMANDS for fast floating-point computations are rapidly growing as the visualization of scientific simulations and multimedia entertainment using three–dimensional graphics are becoming increasingly popular. To meet these demands, high performance ASIC's including floating-point processing units (FPU's) are strongly required.

However, high-speed FPU cores have been difficult to design because they involve such functions as floating-point addition (FADD), multiplication (FMPY), division (FDIV), and square-roots (FSQT). Of these, the performances of the FADD and the FMPY are very important because their operation commands are issued more frequently than other commands. To improve the performance of floating-point arithmetic, several kinds of FPU core with a pipeline structure have recently been developed [1]–[13].

The FADD design is the most difficult because it has complicated procedures requiring many types of building blocks. The required procedures are alignment shift, addition for significands, normalization, and rounding to IEEE 754 Standard [14]. The decoding method for the shift amount and a fast circuit for multiple shifts such as the barrel shifter can improve the alignment shift delay time [1]. The normalization procedure also requires a fast barrel shifter. Fast adders reduce the delay time of significand addition [1], [15]–[17]. An optimized hardware rounding algorithm also improves the FADD delay time [6].

The normalization procedure occupies the dominant portion in a FADD. It consists mainly of the leading-zero counting and normalization shifts. Operating the multiple shift in parallel with the rounding procedure is effective in minimizing the delay time of this procedure [8]. Also, the leading-zero

anticipation (LZA) method is able to conceal the leading-zero counting in this procedure behind the addition for significands. Several floating-point chips currently in production have adopted the LZA approach to reduce their delay time. For example, Hokenek *et al.* have reported their LZA approach in detail, including the expression of Boolean algebra and its circuit implementations [2], [3].

This paper reports a new approach to the LZA method. Our method has a simple Boolean algebra structure and can be realized using fast logic circuits. In the second section, the basic FADD architecture using LZA is explained. It predecodes the shift amount for the normalization from the pair of source significands without summing them. In the third section, the proposed LZA logic is described using Boolean algebra and some examples. The actual LZA logic and leading-zero (LZ) counter circuits are also shown. Then, the fourth section compares the conventional methods. In the fifth section, the FADD core design using our LZA method is presented. This section also reports its fabrication and its cycle time test results. The sixth section concludes this paper.

II. BASIC ARCHITECTURE OF FADD CORE

In a FADD core, the subtraction produces the leading zeros that are on the left-hand side of the significand [18]. Then the FADD core has to shift the significand to the left until the first "1" appears on the left end as the leading bit. When the two numbers are close to each other, this procedure of normalization takes a large portion of the delay time in floating-point addition. If the consecutive bits of these leading zeros are counted in parallel with the addition/subtraction for significands, the total delay time for the FADD is greatly reduced [2], [3].

Fig. 1(a) shows an example of FADD design using LZA. Here, only the parts for addition and normalization for the significands are illustrated. The LZA unit anticipates the amount of the shift for normalization. Because this amount is calculated from the source operands of the addition/subtraction for significands, the LZA performs this operation in parallel with addition/subtraction. Immediately after the addition/subtraction, the shifter carries out the multiple shift for normalization in accordance with the anticipated result.

As shown in Fig. 1(b), the FADD without the LZA has to count the number of leading zeros after the addition/subtraction for significands. The LZ counter adds to the total delay the non-negligible delay of 3.5 ns in the 0.5-μm CMOS circuit. This is one half of the normalization
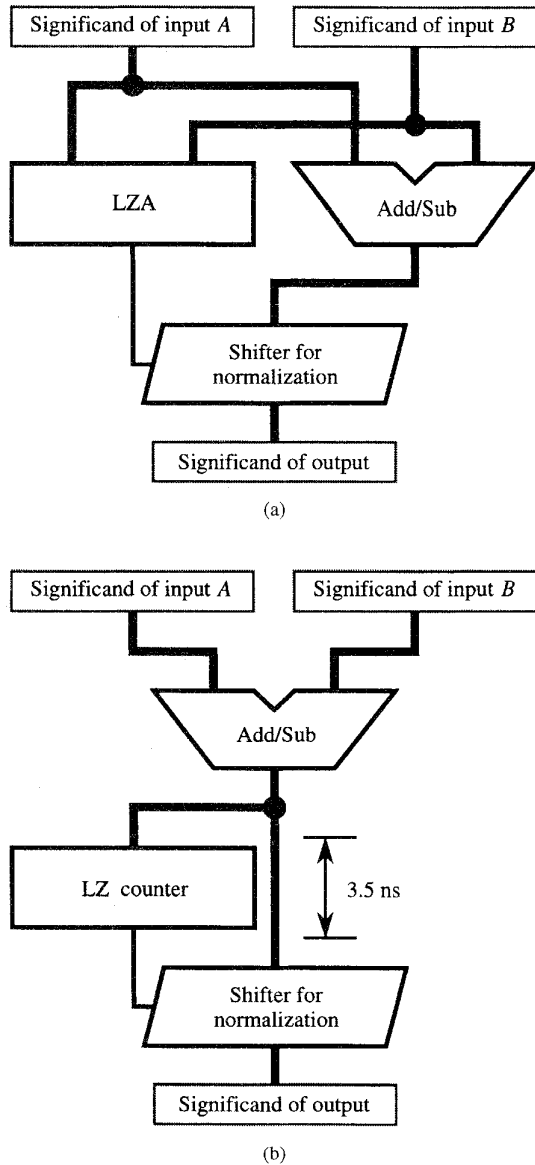
Fig. 1. Basic block diagram of the FADD architecture. (a) Example with the LZA method. (b) Example with only the LZ counter, i.e., LZA methods are not used.

process. Because the LZA conceals this delay behind the addition/subtraction, this basic LZA idea improves the speed performance of the FADD. However, conventional logic for this anticipation has been difficult to design due to its complexity and the large area penalty [2], [3]. In the next section, we propose a new approach for realizing the anticipation function with simple Boolean algebra and a small circuit.

III. PROPOSED LZA LOGIC

A. Boolean Algebra

To predecode the normalization shift amount, our method takes two steps. First, the proposed LZA logic prepares the

predicted consecutive bits of the leading zeros from the two source operands of the summation. Second, the LZ counter decodes these consecutive bits to drive the big shifter for the normalization shift.

The rule required for the proposed LZA logic is as follows:

- The smaller significand is always subtracted from the larger one: that is,

$$\text{if } A \geq B \text{ then } S = A - B = A + (-B),$$
$$\text{if } A < B \text{ then } S = B - A = (-A) + B$$

where A and B are the pair of significands shown in Fig. 1, and S is their summation.

The floating-point format of the IEEE 754 Standard has three parts, namely, the sign bit, the biased exponent, and the significand [14]. In the calculation of a significand, the floating-point add/sub operation results in the addition of the sign magnitude data because the significand is defined as an absolute value. When the sign control unit calculates the sign-bit, two absolute values are added or subtracted. This rule denotes the popular method of executing the subtraction for absolute data formats such as the IEEE Standard [6]–[11]. Having the comparison process before the subtraction guarantees that the result of the significand procedure always shows an absolute value.

To execute the subtraction, the adder generates a negative value by adding "1" to the carry input after inverting the bits as the 1's complement. That is, the calculation of $\{A + (-B)\}$ is executed as the operation of $(A + \bar{B} + 1)$. Here, we define A' and B' as follows:

$$\text{if } A \geq B \text{ then } A' = A, B' = \bar{B},$$
$$\text{if } A < B \text{ then } A' = \bar{A}, B' = B$$

where the upper line denotes the NOT operator. The two inputs A' and B' of the LZA logic are also used by the adder. It is evident that the case of $A' = A$ and $B' = B$ gives the operation of $(A + B)$.

For the LZA logic (E_i), we derived the following expression:

$$E_i = \overline{A'_i \oplus B'_i} \bullet (A'_{i-1} + B'_{i-1})$$
$$= (A'_i \bullet B'_i + \overline{A'_i + B'_i}) \bullet (A'_{i-1} + B'_{i-1}). \quad (1)$$

Here, $\bullet$, $+$, and $\oplus$ denote AND, OR, and Exclusive-OR operators, respectively, and E_i represents the most significant "1," that is, the left-end digit of "1" located at the next bit of the leading zeros. This logic can be theoretically derived from the binary equations for the sum. Below we explain some of the behavior of the E_i.

B. Behavior of LZA Logic

Fig. 2 shows four examples of E's and S's behavior. As E_0 cannot be defined by (1), the mark "_" is used to denote it. The examples use the data when $A \geq B$, that is, $A' = A$ and $B' = \bar{B}$. Fig. 2(a) shows the first case where two significands are close to each other and some upper A and $\bar{B}$ bits are identical. In this example, the sum of $A'_i = $ "0" and $B'_i = $ "0" causes the leading zeros. At the ninth digit, both E and S give the left-end digit as "1" next to the leading zeros. The status

```
A :  +      1000 0000 1010 0001
B :  -      0111 1111 0110 1001
              ⟱
A' : + 0    1000 0000 1010 0001
B' : + 1    1000 0000 1001 0110  +1
─────────────────────────────────
E :    0    0000 0001 0100 100_
S :    0    0000 0001 0011 1000
              (a)


A :  +      1001 1100 1010 0001
B :  -      1001 1100 1000 1001
              ⟱
A' : + 0    1001 1100 1010 0001
B' : - 1    0110 0011 0111 0110  +1
─────────────────────────────────
E :    0    0000 0000 0010 100_
S :    0    0000 0000 0001 1000
              (b)


A :  +      1001 1100 1010 1000
B :  +      1001 1100 1010 0111
              ⟱
A' : + 0    1001 1100 1010 1000
B' : + 1    0110 0011 0101 1000  +1
─────────────────────────────────
E :    0    0000 0000 0000 000_
S :    0    0000 0000 0000 0001
              (c)


A :  +      1001 1100 0010 0001
B :  +      0011 1011 1100 1001
              ⟱
A' : + 0    1001 1100 0010 0001
B' : + 0    0011 1011 1100 1001  +0
─────────────────────────────────
E :    1    0101 1000 0011 001_
S :    0    1101 0111 1110 1010
              (d)
```

Fig. 2. Some examples of E's and S's behavior. (a) $A_i = \bar{B}_i$ in upper bits. (b) $A_i = B_i$ in upper bits. (c) $A - B = 1$. (d) $A + B$.

of E_0 does not affect the result. The position of their most significant "1" is coincidence. Thus, E's logic anticipates the position of the most significant "1" correctly.

Fig. 2(b) shows the second case where the two significands A and B are close to each other and their upper bits are identical. In this example, the leading zeros appear in the upper bits of the sum because $A'_i \oplus B'_i$ is asserted there in series. Here E gives the significant digit at the sixth digit, while S gives it at the fifth digit. Also, in this case, the status of E_0 does not affect the result. The position of this anticipated digit is likely to deviate by a single bit in this case. More correctly, the wrongly anticipated position is always one bit left of the correct one.

Fig. 2(c) shows the third case where the difference between two significands is "00...001." This example results in $S =$ "00...001" and $E =$ "00...00_." In this case, we have to consider the status of E_0. If E_0 is always asserted, this

example will be correctly anticipated. However, the case of $S =$ "00...000" will lead to the wrong anticipation. This error shows a shift of one bit left of the correct position. That is, it is the same mode mentioned in the second case. On the other hand, if E_0 is always de-asserted, it causes the shift of one bit right of the correct position. Because this error is different from the error mentioned in the second case, it requires another compensation shifter. If E_0 is always asserted, the compensation becomes easier.

Fig. 2(d) shows the fourth case where the positive values of A and B are summed. In this case, the E value at the sign bit is asserted, because the left-end digits representing the sign bits of A and B are zero. Therefore, if the circuits for anticipation are extended to the sign bit, this logic gives an un-shift signal to the add operation for significands.

The examples in Fig. 2(a)–(d) exhaust all the possible behavior of the proposed LZA logic. Any combination of input patterns falls into one of these patterns. Thus, it is confirmed that the anticipated position of the leading bit is equal to or a single bit larger than the significant digit of S.

When the anticipated position is a single bit left of the correct one, our method has to additionally shift one bit to the left. However, this overhead of the single-bit shifter is not so large in the actual VLSI implementation of the FADD. In some cases, the additional circuit can be shared with the other shifter in the rounding block.

As mentioned above, the proposed LZA logic has the possibility of anticipating wrongly. However, if we permit this error, it simplifies the logic dramatically. Thus, it led to a fast operating circuit.

C. Circuit Implementation

The LZA logic E_i given in (1) is realized by simple CMOS gates, as shown in Fig. 3. This circuit is constructed of three gate stages. The transistor count for each bit is only 16. Here, the NOR gate of the $(i - 1)$th digit is shared with the next block of the ith digit. It should be emphasized that there are no signal paths spread over many bits like a carry signal in binary adders. The circuit in our design uses conventional CMOS gates because it gives sufficient speed in comparison with conventional adders. Although the use of faster type circuits, such as the pass-transistor logics, would enhance the speed greatly, it also increases the transistor count. As shown in Fig. 2(c), we decided to always assert E_0, thus $\overline{A_0 + B_0}$ are generated for the first block of E_1.

The proposed method also requires a fast circuit for the leading-zero counting. To achieve the fastest operation, a tree type LZ counter circuit is adopted. The tree structure using a pass transistor circuit multiplexer reduces the delay of this circuit. The features of this circuit structure are illustrated in Fig. 4. This figure shows an 8-b building block with two 4-b blocks for the bottom stage in the tree structure. The upper block of the 4-b tree generates the primely encoded value of $D4_{m,\langle 1:0\rangle}$ and the 4-b logical OR of $OR4_m$. The lower block of the 4-b tree encodes the inputs of $E_{\langle m-4:m-7\rangle}$ to $D4_{m-4,\langle 1:0\rangle}$ and $OR4_{m-4}$. The compressor circuit in the gray-lined box generates $D8_{m,\langle 2:0\rangle}$ and $\overline{OR8_m}$ from these values of the 4-b blocks. Here $D8_{m,2}$ is derived from $OR4_m$, and $D8_{m,\langle 1:0\rangle}$
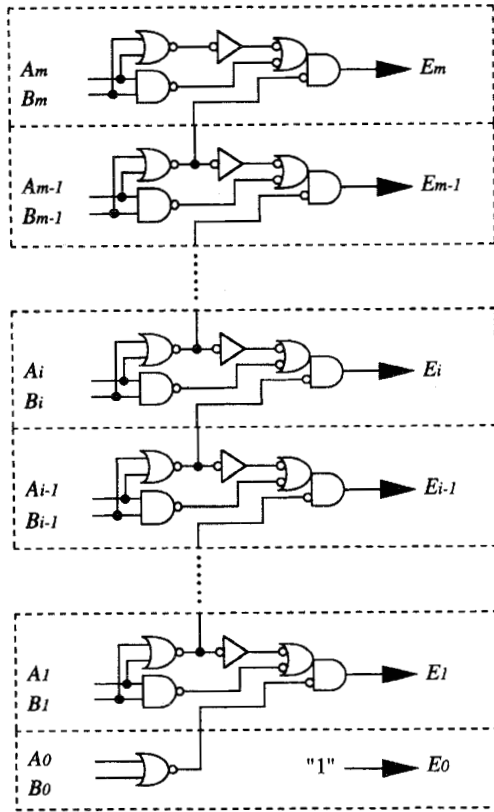
Fig. 3. CMOS circuit of the proposed LZA logic. The E signals between the first and mth digit are generated from the same blocks of the circuit. E_0 is always asserted and $\overline{A_0 + B_0}$ are generated for the first block of E_1.
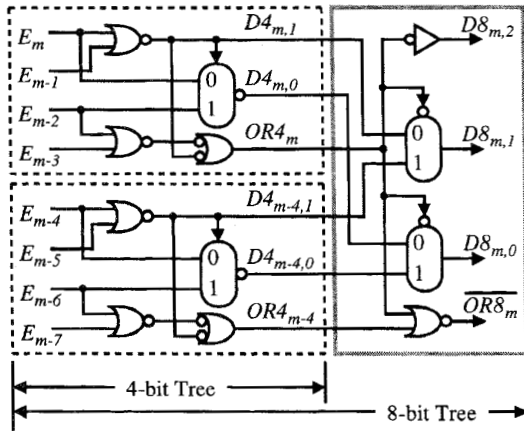

Fig. 4. Basic structure of LZ counter. This figure represents only the block of the 8-b tree circuit composed of the two blocks of the 4-b tree circuit.

is selected from $D4_{m,\langle 1:0 \rangle}$ or $D4_{m-4,\langle 1:0 \rangle}$ by $OR4_m$. Also $\overline{OR8_m}$ is used for the 16-b building block. The 16-b tree is composed of a pair of 8-b trees by the additional compressor. In this way, a bigger LZ counter tree can be designed.

IV. COMPARISON WITH CONVENTIONAL METHODS

Some FADD cores use the LZA method to reduce the delay time, and the others omit using it to avoid its area penalty. Fig. 5(a) and (b) compare the delay times and transistor counts

(a)

(b)

■ *Significand adder* ▨ *Multiple Shifter (1st) / Multiple Shifter (hex)*
▧ *LZA logic LZA* ◈ *Multiple Shifter (2nd)*
▨ *LZ Counter* ▤ *Compensate decoder / 4-bit LZ decoder*
□ *Shift decoder* ▨ *Compensate shifter / Multiple Shifter (binary)*
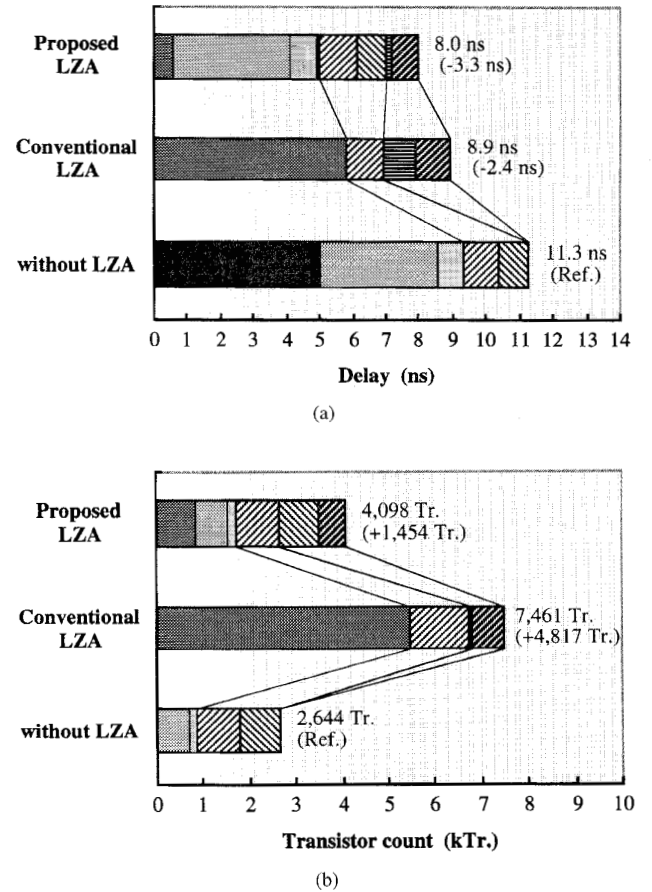□ *Waiting delay*

Fig. 5. Comparison chart of (a) delay time and (b) transistor count of the proposed LZA method, the conventional LZA method, and the method without LZA.

of three methods, namely, using the proposed LZA, using a conventional LZA [2], [3], and without LZA [1], [8]–[13]. The exact values of individual circuits are also summarized in Table I. Here, these comparisons are derived from the prelayout design of 0.5-μm CMOS 64-b FADD cores.

In the case of the proposed LZA method, the values are calculated from the circuits of the LZA logic, LZ counter, shift decoder, multiple shifters, and compensation shifter. The shift decoder translates the number counted by the LZ counter to the drive signals of the multiple shifters. Here, the normalization shifter is composed of two blocks of multiple shifters. The first block performs the coarse shift for the byte blocks, and the second block performs the binary shift in the 8-b range. In the case of the conventional LZA method, the values are calculated from the logic circuit, multiple shifter (hex), 4-b LZ decoder for the hex data and multiple shifter (binary). The conventional LZA has to decode the binary position of the leading-zero before the binary shift because it calculates only the hexadecimal position of leading-zero. In the case without

TABLE I
COMPARISON OF DELAY TIME AND TRANSISTOR COUNT OF THE PROPOSED LZA
METHOD, THE CONVENTIONAL LZA METHOD, AND THE METHOD WITHOUT LZA

	Delay (ns)	Transistor count
Proposed LZA	**8.0**	**4,098**
- LZA logic	0.6	884
- LZ counter	3.5	696
- Shift decoder	0.8	152
- Waiting delay	0.1	-
- Multiple shifter (1st)	1.1	928
- Multiple shifter (2nd)	0.9	868
- Compensate decoder	0.2	2
- Compensate shifter	0.8	568
Conventional LZA	**8.9**	**7,461**
- LZA	5.8	5,446
- Multiple shifter (hex)	1.1	1,288
- 4-bit LZ decoder	1.0	39
- Multiple shifter (binary)	1.0	688
Without LZA	**11.3**	**2,644**
- Significand adder	5.0	-
- LZ counter	3.5	696
- Shift decoder	0.8	152
- Multiple shifter (1st)	1.1	928
- Multiple shifter (2nd)	0.9	868

the LZA method, the value of the delay time is calculated from the circuits of the significand adder, the LZ counter, and multiple shifters. The shifter structure is same as that of the proposed LZA.

The proposed LZA is smaller and faster than the conventional LZA. The proposed LZA method is 0.9 ns faster than the conventional LZA method, and 3.3 ns faster than the method without LZA. Besides, its LZA path is faster than the significand addition whose delay time is 5.0 ns. Therefore, its operation is concealed behind the adder operation, and it has to wait 0.1 ns during the significand adder operation. On the other hand, the delay time of the conventional LZA is 0.8 ns larger than that of the significand addition. Besides, the shift operation of the proposed method including the compensation shift is 0.1 ns faster than that of the conventional method. Although the proposed method needs an additional shifter to compensate for its wrong anticipation, the delay time of its whole shift operation is less than that of the conventional LZA. The delay penalty of the compensation is as small as the LZ decoding for the binary shift in the conventional method, because the wrong anticipation is limited to the 1-b left position and can be easily compensated for by the simple logic and small shifter. The multiple shifter (second) is faster than that of the multiple shifter (binary), because its drive circuit operates concurrently with the first shifter. As for the area penalty, the proposed LZA increases the number of transistors over non-LZA methods to 1454 including the compensation

circuits. This corresponds to 30% of the 4817 transistors of the conventional LZA method.

Using LZA methods has the advantage of delay time and the penalty of transistor count. The proposed LZA has 3.3 ns of speed advantage and 1454 transistor of area penalty. On the other hand, the conventional LZA has 2.4 ns of speed advantage and 4817 transistor of area penalty. Thus, the proposed method, giving a larger advantage with a smaller penalty, changes the tradeoff point toward the use of the LZA method. In the pipelining approach, the largest delay time of the building blocks deeply affects the cycle time of the pipeline. Each of the largest delay times is 5.0 ns of the significand adder in the proposed method and 5.8 ns of the LZA unit in the conventional method. From this point of view, the proposed method has a speed advantage of 0.8 ns compared with the conventional method. This advantage corresponds to 14% of the maximum delay time of the building blocks, thus it deeply affects the cycle time of the pipeline approach such as is described following section.

V. FADD CORE DESIGN

A. Architecture and Circuits

We designed a FADD core using the proposed LZA method. Fig. 6 shows a block diagram. This core is divided into five execution pipeline stages. The building blocks of the proposed LZA as mentioned above are distributed in the third, fourth, and fifth stages.

In the first stage, the input aligner suitably aligns the source operands for the given instructions. A pair of selectors swaps the two significands depending on the comparison result of their exponents. Concurrently, the larger exponent is sent to the exponent subtracter in the fourth stage.

In the second stage, the right shifter shifts the significand of the smaller data to equalize its exponent value. In parallel with this, these significands are compared and ordered so that the subtraction result becomes positive. One of the bit-inverters inverts the smaller data. The carry-input signal (not shown) is asserted to the adder for the subtraction. In most cases, the right bit-inverter works and the left bit-inverter is idle because the swapper in the first stage selects the significand with the smaller exponent. The left bit-inverter works only if the left significand is smaller than the right significand when their original exponents are the same.

The third stage performs 56-b addition and predecoding for the normalization shift. This predecoding is carried out by the proposed LZA logic and LZ counter. This LZA circuit is extended to the sign bits. Because this extension enables the LZA circuit to generate the unshift signal at the add operation shown in Fig. 2(d), the unshifted data pass through the barrel shifter. The rounding procedure uses this unshifted data as the rounded-down signal in the following stage. That is, the extension of the sign bit enables the signal path of the round-down data to be shared with the signal path of the normalization shifter. The 56-b adder circuit has seven 8-b carry lookahead blocks with the carry-skip structure one of the seven blocks.
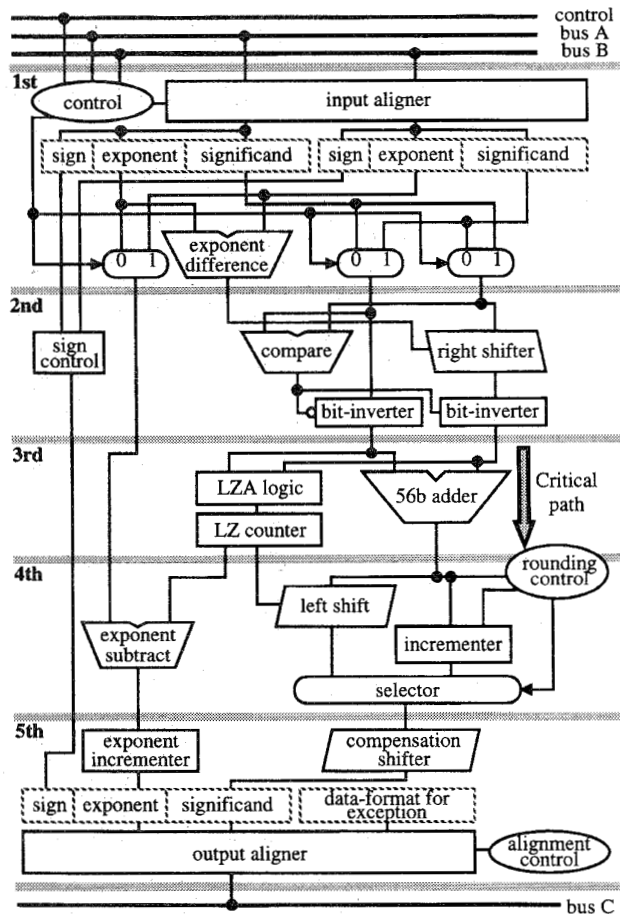
Fig. 6. The FADD block diagram using the proposed LZA method. It has five execution pipeline stages.

Fig. 7. Photomicrograph of the FADD core which is fabricated with 0.5-μm CMOS technology with triple metal interconnections. The die size is 3.5 mm × 3.6 mm.

The fourth stage performs the left shift for normalization and the increment for the rounded-up signal. The incrementer speculatively generates the round-up signal without the rounding conditions. Concurrently, the rounding controller estimates the condition of round-up/round-down. In the case of round-down, the incremented data must be canceled by the bypassed data as the round-down signal. The normalization shifter includes the path for bypassing. For example, in the addition, the extended bit generates the control signal to bypass the nonshifted data as mentioned in Fig. 2(d). That is, the rounded-down signal passes through the normalization shifter, and the control signals of the rounding select these paths according to the rounding mode and the requirements of the normalization shift.

The fifth stage performs the small significand shift and increments the exponent to compensate for the leading-bit position. These blocks adjust any mis-shifts caused by the wrong anticipation. Additionally, they also compensate a single-bit shift caused in the rounding procedures. In this design, the compensation shifter is shared with the small shifter for the rounding procedure. This shifter adjusts the 1-b over-flow/underflow in the rounding procedure. Therefore, the penalty of our LZA method is as small as 884 transistors. This corresponds to 1.8% of the total. Finally, the output aligner
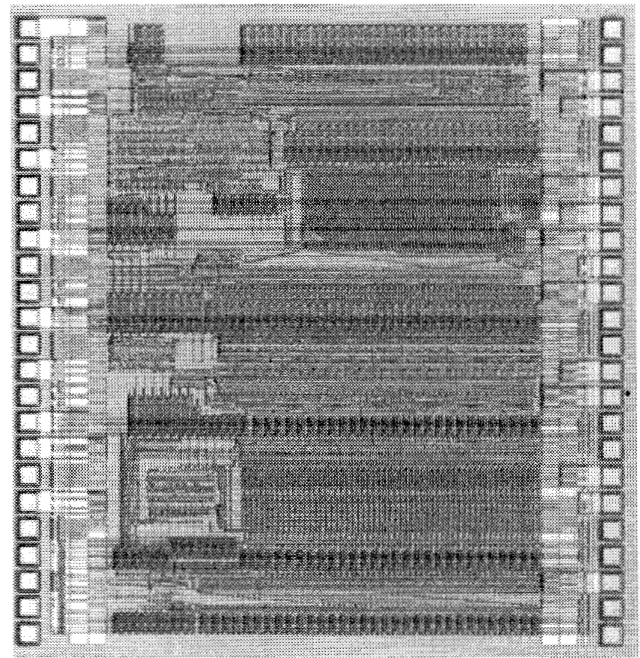
aligns the output data according to the destination type of the given instruction.

The delay time of the third stage restricts the cycle time of the FADD because the 56-b adder has the largest delay time. On the other hand, as the LZA logic and the LZ counter have a smaller delay than the adder as mentioned above, the normalization process is hidden behind the concurrent processes of the adder and the rounding.

B. Fabrication and Test

The FADD core is fabricated with 0.5-μm CMOS process technology with triple metal interconnections. Fig. 7 shows a photomicrograph of the chip. Using symbolic elements, we designed the layout manually with the aid of a commercial layout synthesizer. The layout synthesizer carried out the generation of the leaf cells from the design information of the circuit structures. We manually placed and routed them. The layout of this core has triple metal levels. The third metal level is used only for power supply and clock distribution. The inner cells are routed by the first and second metal levels. The inner cells do not have fat power lines inside the cell because this layout methodology guarantees sufficient power supply through the third metal level. The device features are summarized in Table II.

The core has scan-path circuit designed to investigate the cycle time and various functions. The source and destination registers of each pipeline stage are serially connected and occur from the serial-input pin to the serial-output pin. The main clock and the test clock control these registers according to the test program. These connected registers serve as the shift registers of the scan-paths under the control of the test clock. Also they serve as the pipeline registers under the control

TABLE II
DEVICE FEATURES OF THE FADD CORE

Process technology	Triple metal CMOS
Design rule	0.5 μm
Transistor count	54 k
Die size	3.5 mm × 3.6 mm
Active area	2.5 mm × 3.5 mm
Clock rate	164 MHz
Power dissipation	3.24 mW/MHz at 3.3 V

Fig. 8. Schmoo plot of the FADD core. This plot shows the cycle time of 6.1 ns at the supply voltage of $V_{DD} = 3.3$ V. It correspond to the 164 MHz operation.

of the main clock of the circuit operation. The test program serially inputs the input data into the source registers by the shift operation. This scan-path operation is controlled by the test clock. After the circuit operation of the core, the calculated values are stored in the destination registers. The test program outputs the data in the destination register to the output pin by the shift operation of the scan-path. The data of the output pins are compared synchronously with the expected values in the test patterns.

In the cycle time evaluation, the test program controls the timings of the clock signals. The difference between the clock signals of the source and destination registers corresponds to the measured values of the cycle time. The minimum cycle time is evaluated from a schmoo plot of the cycle times.

To generate the test patterns, we use commercial tools for path analysis and logic simulations. The path analysis tool determines the critical paths of the delay time. According to the results of the path analysis, 1000 of the test patterns that make the critical paths active are generated. The logic simulator confirms the path activated by these patterns. It also generates the expected values. Additionally, 100 000 random patterns were generated to confirm the functions of this core.

The test results at room temperature are shown in Fig. 8. The chip has a cycle time of 6.1 ns at the supply voltage of $V_{DD} = 3.3$ V. This corresponds to 164 MHz operation. The minimum supply voltage of the operation range is $V_{DD} = 2.2$ V, which is limited by the minimum operation voltage of the barrel shifter, because it consists of NMOS pass-transistor logic.

VI. CONCLUSION

We have proposed a new LZA logic. The LZA logic carries out predecoding for normalization shifts in parallel with addition for the significand. The proposed LZA logic is defined by simple Boolean algebra, and is composed of a small number of CMOS transistors. Its area penalty is as small as 30% of the conventional LZA method by transistor count. Because the delay time of the proposed circuit is smaller than that of the adder operation for the significand, the predecoding for the normalization shift is completely hidden behind the significant addition. The logic was applied to the design of a 64-b floating-point-adder core fabricated with 0.5-μm CMOS process technology with triple metal interconnections. These innovations enable the core to operate at 164 MHz. The proposed LZA method is effective for reducing the delay time of FADD cores with only a small area penalty.
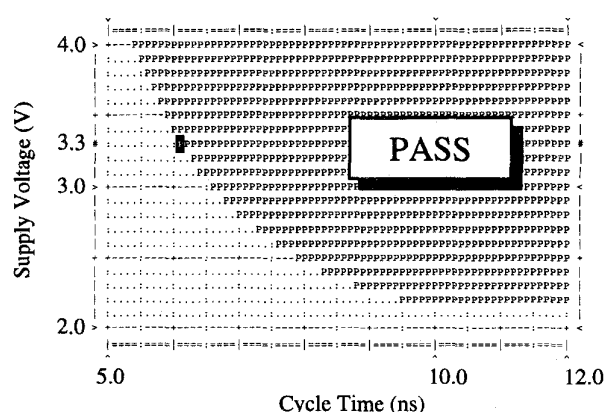
REFERENCES

[1] H. Fujii, C. Hori, T. Takada, N. Hatanaka, T. Demura and G. Ootomo, "A floating-point cell library and a 100-MFLOPS image signal processor," IEEE J. Solid-State Circuits, vol. 27, pp. 1080–1087, July 1992.
[2] E. Hokenek, R. Montoye, and P. Cook, "Second-generation RISC floating point with multiply-add fused," IEEE J. Solid-State Circuits, vol. 25, pp. 1207–1213, Oct. 1990.
[3] E. Hokenek and R. Montoye, "Leading-zero anticipator (LZA) in the IBM RISC system/6000 floating-point execution unit," IBM J. Res. Develop., vol. 34, pp. 71–77, Jan. 1990.
[4] K. Molnar, C. Ho, D. Staver, B. Davis, and R. Jerdonek, "A 40 MHz 64-bit floating-point co-processor," in ISSCC Dig. Tech. Papers, Feb. 1989, pp. 48–49.
[5] B. Benschneider, W. Bowhill, E. Cooper, M. Gavrielov, P. Gronowski, V. Maheshwari, V. Peng, J. Pickholtz, and S. Samudrala, "A 50 MHz uniformly pipelined 64 b floating-point arithmetic processor," in ISSCC Dig. Tech. Papers, Feb. 1989, pp. 50–51.
[6] L. Kohn and S. Fu, "A 1,000,000 transistor microprocessor," in ISSCC Dig. Tech. Papers, Feb. 1989, pp. 54–55.
[7] D. Steiss, S. Mangelsdort, P. Groves, D. Bural, M. Gill, R. Gratias, M. Jassowski, R. Luebs, B. Naas, A. Reynolds, H. Rothermel, W. Walker, and T. Wolf, "A 65 MHz floating-point coprocessor for a RISC processor," in ISSCC Dig. Tech. Papers, Feb. 1991, pp. 94–95.
[8] N. Ide, H. Fukuhisa, Y. Kondo, T. Yoshida, M. Nagamatsu, J. Mori, I. Yamazaki, and K. Ueno, "A 320-MFLOPS CMOS floating-point processing unit for superscalar processors," IEEE J. Solid-State Circuits, vol. 28, pp. 352–361, Mar. 1993.
[9] G. Taylor, A. Rekow, J. Radke, and G. Thompson, "A 100 MHz floating point/integer processor," in Proc. IEEE Custom Integrated Circuits Conf., May 1990, pp. 24.5.1–24.5.4.
[10] F. Okamoto, Y. Hagihara, C. Ohkubo, N. Nishi, H. Yamada, and T. Enomoto, "A 200-MFLOPS 100-MHz 64-b BiCMOS vector-pipelined processor (VPP) ULSI," IEEE J. Solid-State Circuits, vol. 26, pp. 1885–1892, Dec. 1991.
[11] H. Nakano, M. Nakajima, Y. Nakamura, T. Yoshida, Y. Goi, Y. Nakai, R. Segawa, T. Kishida, and H. Kadota, "An 80-MFLOPS (Peak) 64-b microprocessor for parallel computer," IEEE J. Solid-State Circuits, vol. 27, pp. 365–372, Mar. 1992.
[12] S. Komori, H. Takata, T. Tamura, F. Asai, T. Ohno, O. Tomisawa, T. Yamasaki, K. Shima, H. Nishikawa, and H. Terada, "A 40MFLOPS 32-bit floating-point processor," in ISSCC Dig. Tech. Papers, Feb. 1989, pp. 46–47.
[13] Y. Shimazu, T. Kengaku, T. Fujiyama, E. Teraoka, T. Ohno, T. Tokuda, O. Tomisawa, and S. Tsujimichi, "A 50 MHz 24 b floating-point DSP," in ISSCC Dig. Tech. Papers, Feb. 1989, pp. 44–45.

[14] ANSI/IEEE Standard 754—1985 for Binary Floating-Point Arithmetic, IEEE Computer Society Press, Los Alamos, CA, 1985.
[15] D. Dobberpuhl, R. Witek, R. Allmon, R. Anglin, D. Bertucci, S. Britton, L. Chao, R. Conrad, D. Dever, B. Gieseke, S. Hassoun, G. Hoeppner, K. Kuchler, M. Ladd, B. Leary, L. Madden, E. McLellan, D. Meyer, J. Montanaro, D. Priore, V. Rajagopalan, S. Samudrala, and S. Santhanam, "A 200-MHz 64-b dual-issue CMOS microprocessor," *IEEE J. Solid-State Circuits*, vol. 27, pp. 1555–1567, Nov. 1992.
[16] M. Suzuki, N. Ohkubo, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, and Y. Nakagome, "A 1.5-ns 32-b CMOS ALU in double pass-transistor logic," *IEEE J. Solid-State Circuits*, vol. 28, pp. 1145–1151, Nov. 1993.
[17] A. Inoue, Y. Kawabe, Y. Asada, and S. Ando, "A 0.4 μm 1.4 ns 32 b dynamic adder using nonprecharge multiplexers and reduced precharge voltage technique," in *Symp. VLSI Circuits Dig. Tech. Papers*, June 1995, pp. 9–10.
[18] J. Hennessy and D. Patterson, *Computer Organization and Design: The Hardware/Software Interface.* San Mateo, CA: Morgan Kaufmann, pp. 230–233, 1994.

Hiroyuki Morinaka, for a photograph and biography, see p. 513 of the April issue of this JOURNAL.

Hiroshi Makino, for a photograph and biography, see p. 512 of the April issue of this JOURNAL.

Yasunobu Nakase, for a photograph and biography, see p. 513 of the April issue of this JOURNAL.

Koichiro Mashiko (M'88), for a photograph and biography, see p. 513 of the April issue of this JOURNAL.

Hiroaki Suzuki, for a photograph and biography, see p. 513 of the April issue of this JOURNAL.

Tadashi Sumi, for a photograph and biography, see p. 513 of the April issue of this JOURNAL.