Nov. 2006, Vol.21, No.6, pp.901–906 J. Comput. Sci. & Technol.

Parallel Error Detection for Leading Zero Anticipation

Ge Zhang (张 戈), Wei-Wu Hu (胡伟武), and Zi-Chu Qi (齐子初)

Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences Beijing 100080, P.R. China

Graduate University of the Chinese Academy of Sciences, Beijing 100039, P.R. China

E-mail: gzhang@ict.ac.cn

Received October 25, 2004; revised November 28, 2005.

Abstract The algorithm and its implementation of the leading zero anticipation (LZA) are very vital for the performance of a high-speed floating-point adder in today's state of art microprocessor design. Unfortunately, in predicting "shift amount" by a conventional LZA design, the result could be off by one position. This paper presents a novel parallel error detection algorithm for a general-case LZA. The proposed approach enables parallel execution of conventional LZA and its error detection, so that the error-indication signal can be generated earlier in the stage of normalization, thus reducing the critical path and improving overall performance. The circuit implementation of this algorithm also shows its advantages of area and power compared with other previous work.

Keywords computer arithmetic, floating-point addition, leading zero anticipation

1 Introduction

Leading Zero Anticipation (LZA) is a technique that predicts the location of the most significant digit in a floating-point addition of the inputs given to the adder. This determination of the leading-zero result is performed in parallel with the addition step so as to facilitate immediate start of the normalization shift operation once the addition completes. Fig.1 shows the basic function of LZA in floating-point adder design. A great many approaches have been proposed for a faster and simpler LZA logic[1]. These LZA designs have been incorporated into most realistic floating-point processing units (FPU) and commercial processors[2−4]. The choice of determining LZA style is often dependent on the overall design of the floating-point addition unit, that is, on how subtraction is handled when the exponents are the same and how it detects and corrects possible one-bit error of the LZA[1].

The typical LZA logic consists of two main parts: a pre-encoding module which generates a string of bits with the most-significant digit "1" having the same position as the actual sum output; and a leading zero detector (LZD) which is then employed to encode the pre-encoding result. LZA is often used in floating-point adder when the operation is effective subtraction, for the result of effective addition need not be left shifted for normalization. Some LZAs are simply designed under the assumption that the result is always positive. Most of the LZAs are inexact because of one bit of error possible in their results[1].

In this paper, we propose a parallel error detection algorithm for a general-case leading zero anticipator. This algorithm works well for the leading zero predication regardless of positive or negative result of subtraction, so it is suitable for the high-speed dual-path based floating-point addition units[5]. In addition, the proposed error detection directly detects the two inputs of the subtraction, and thus has no impact on the original circuits of the adder and LZA.

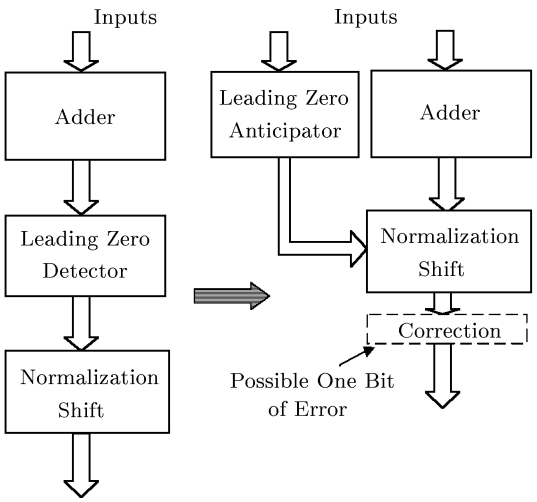

Fig.1. Function of LZA in floating-point adder.

The rest of the paper is organized as follows. Section 2 presents an overview of previous work on Leading Zero Anticipation. In Section 3, the proposed LZA design with a novel algorithm for error detection is presented and discussed. The experimental results are presented in Section 4, followed by a conclusion in Section 5.

2 Previous Work on LZA

Leading Zero Anticipation is a very well researched topic and a number of techniques have been presented in the literature[6−12]. The first LZA scheme in [6] can

Short Paper

Supported by the National High Technology Development 863 Program of China under Grant Nos. 2002AA111100 and the National Basic Research 973 Program of China under Grant No. 2005CB321600.

work for both leading zeros, when the result of a subtraction is positive, and leading ones, when the result is negative. A relatively complicated scheme for LZA is described in [7]. [8] provides a simpler circuit based on the assumption that the result of subtraction is always positive, which is only suitable for the cases where the two operands have been compared and exchanged with each other in the early stage of floating-point addition.

All the LZA schemes mentioned above are off by an error of one bit in the pre-encoding module, and the actual count of leading zeros of the subtraction's result may be one bit more than that of the LZA's result. This error must be corrected after the addition and normalization shifts. In the evaluation performed in [8], this correction process accounts for a 12.5% delay of addition plus shifting. [9] describes an exact LZA logic incorporated into the adder, however its description is just a simplification of the actual design, and it is not always practical to use such LZA without affecting the design of the adder.

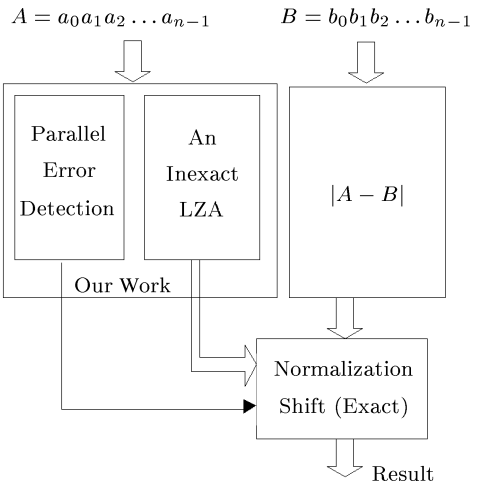

Fig.2. Proposed LZA architecture.

An alternative way to provide an exact LZA is to generate an error signal in parallel with the conventional LZA logic and then correct the leading zero's count before the required shift. It has been proved that this method is effective to reduce the critical path delay[10,11]. A conceptually simpler scheme for error detection was proposed in [12] for detecting the error through examination of each carry-in bit of the addition. However, the count of leading zeros cannot be generated in time, which deteriorates the critical path from addition to normalization. The error detection scheme proposed in [10, 11] uses a relatively independent binary tree to get error detection signal, but its detection tree only considers one case that the subtraction result is either positive or negative, so two separate detection trees must be used for a general case LZA. In the following sections, we will propose a novel design of parallel error detection for LZA, which is independent of the original inexact LZA and adder, and can work in any cases of subtraction results, positive or negative. Fig.2 shows the basic architecture of our proposed LZA with parallel error detection.

3 Proposed LZA with Parallel Error Detection

3.1 General-Case Inexact LZA

Firstly let us consider the subtraction of two inputs with magnitude representation: A and B,

$$A = a_0a_1a_2 \ldots a_{n-1}; \quad B = b_0b_1b_2 \ldots b_{n-1}.$$

Both A and B are considered positive here since they denote the magnitude of the inputs using IEEE-754 standard "sign-and-magnitude representation". Our purpose is to precisely predict the leading zeros of the subtraction $|A - B|$ without prior knowledge of which operand is greater.

To achieve this, firstly, for each bit position i of the input operands, the following functions are defined:

$$\begin{aligned} g_i &= a_i\bar{b}_i \qquad (a_i > b_i),\\ s_i &= \bar{a}_ib_i \qquad (a_i < b_i),\\ e_i &= a_ib_i + \overline{a_ib_i} \quad (a_i = b_i). \end{aligned} \tag{1}$$

Then we can get a string W with each bit w_i denoted by $g_i(a_i > b_i)$ or $s_i(a_i < b_i)$ or $e_i(a_i = b_i)$. Table 1 lists all possible pattern sequences of the string and corresponding most-significant "1"s positions for the result of the subtraction. In this table the superscript i, j or k denote the numbers of times for a pattern's appearance, the pattern $e^ig\cdots$ indicates a positive result and the pattern $e^is\cdots$ indicates a negative result. Furthermore, it is evident that the most-significant "1"s position appears in the result of $|A - B|$, when the pattern of W meets the cases of $e_{i-1}g_i\overline{s_{i+1}}$, $\overline{e_{i-1}}s_i\overline{s_{i+1}}$ (for $A > B$) or of $e_{i-1}s_i\overline{g_{i+1}}$, $\overline{e_{i-1}}g_i\overline{g_{i+1}}$ (for $A < B$).

Table 1. Pattern Sequences for Leading One Position

Pattern Sequence of W		Leading One
$A > B$	$B > A$	Position
$e^igg\ldots$	$e^iss\ldots$	i
$e^ige^jg\ldots$	$e^ise^js\ldots$	i
e^ige^j	e^ise^j	i
$e^ige^js\ldots$	$e^ise^jg\ldots$	$i+1$
$e^igs^jg\ldots$	$e^isg^js\ldots$	$i+j$
$e^igs^je^kg\ldots$	$e^isg^je^ks\ldots$	$i+j$
$e^igs^je^k$	$e^isg^je^k$	$i+j$
e^igs^j	e^isg^j	$i+j$
$e^igs^je^ks\ldots$	$e^isg^je^kg\ldots$	$i+j+1$

So we can construct a binary string whose number of leading zeros is approximately the same as that of the actual result of subtraction, denoted by F:

$$\begin{aligned} F_i = {} & e_{i-1}g_i\overline{s_{i+1}} + \overline{e_{i-1}}s_i\overline{s_{i+1}} + e_{i-1}s_i\overline{g_{i+1}}\\ & + \overline{e_{i-1}}g_i\overline{g_{i+1}}, \quad \text{when } i = 0,\ e_{i-1} = 0. \end{aligned} \tag{2}$$

The binary string F can be calculated by a well-known process called "pre-encoding" using the values of each digit and its two neighbors of the string W. After this, an LZD[13,14] can be used to count the leading zeros of string F and generate the final output of LZA.

Unfortunately, the LZA model described above is still inaccurate. It is evident in Table 1 that when the pattern sequence of string W is $e^i g e^j s \cdots, e^i s e^j g \cdots$, $e^i g s^j e^k s \cdots, e^i s g^j e^k g \cdots$, the leading one's position in binary string F would be off by another position to the left from the actual leading one's position. To solve this problem, we will discuss a proposed error detection scheme in the following subsections.

3.2 Error Detection of LZA

In this subsection we present an algorithm which can recognize all the four pattern sequences leading to error prediction. Firstly, we provide a new string that combines all of the four patterns into one. To do this, the following functions are defined:

$$\begin{aligned} p_i &= (e_{i-1}|e_{i-2}g_{i-1}|\overline{e_{i-2}}s_{i-1})g_i, \\ n_i &= (e_{i-1}|e_{i-2}s_{i-1}|\overline{e_{i-2}}g_{i-1})s_i, \\ z_i &= \overline{p_i|n_i}, \\ \text{when} \quad & i = 0,\ e_{i-1} = e_{i-2} = 1,\ g_{i-1} = s_{i-1} = 0. \end{aligned} \tag{3}$$

A new string H represented by p_i, n_i and z_i is constructed. Table 2 lists all possible pattern sequences of H in correspondence with Table 1. It can be found that the patterns leading to error prediction have been uniquely denoted by the pattern $z^i x z^j y \cdots$, where x, y indicate the first appearances of p, n or n, p in string H respectively.

Table 2. Pattern Sequences of H

Pattern Sequence		Error
Original W	Denoted by H	Prediction
$e^i g g \ldots$	$z^i p p \ldots$	No
$e^i g e^j g \ldots$	$z^i p z^j p \ldots$	No
$e^i g e^j$	$z^i p z^j$	No
$\boldsymbol{e^i g e^j s \ldots}$	$\boldsymbol{z^i p z^j n \ldots}$	**Yes**
$e^i g s^j g \ldots$	$z^i p z^j p \ldots$	No
$e^i g s^j e^k g \ldots$	$z^i p z^{j+k} p \ldots$	No
$e^i g s^j e^k$	$z^i p z^{j+k}$	No
$\boldsymbol{e^i g s^j e^k s \ldots}$	$\boldsymbol{z^i p z^j n \ldots}$	**Yes**
$e^i s s \ldots$	$z^i n n \ldots$	No
$e^i s e^j s \ldots$	$z^i n z^j n \ldots$	No
$e^i s e^j$	$z^i n z^j$	No
$\boldsymbol{e^i s e^j g \ldots}$	$\boldsymbol{z^i n z^j p \ldots}$	**Yes**
$e^i s g^j s \ldots$	$z^i n z^j n \ldots$	No
$e^i s g^j e^k s \ldots$	$z^i n z^{j+k} n \ldots$	No
$e^i s g^j e^k$	$z^i n z^{j+k}$	No
$\boldsymbol{e^i s g^j e^k g \ldots}$	$\boldsymbol{z^i n z^j p \ldots}$	**Yes**

A novel binary tree is constructed here to detect the pattern $z^i x z^j y \cdots$ of H. The basic theory of this binary tree is compressing three adjacent bits of H into two adjacent bits at each level, and then finally converting H to a two-bit representation which only holds the first and second encounters of p or n of H. Fig.3 describes an example of this binary tree, and the error-indicating signal can be obtained by $Y = P^l N^r | N^l P^r$ in the last level nodes of the tree.

The formulations to form the binary tree are defined here:

$$\begin{aligned} P_l &= P^l + Z^l P^m, \\ P_r &= \bar{Z}^l P^m + P^r (Z^l + Z^m), \\ N_l &= N^l + Z^l N^m, \\ N_r &= \bar{Z}^l N^m + N^r (Z^l + Z^m), \\ Z_l &= Z^l Z^m, \\ Z_r &= Z^r (Z^l + Z^m), \end{aligned} \tag{4}$$

where the superscripts l, m and r respectively denote the left input, the middle input and the right input from the upper level, and the subscripts l and r respectively denote the left output and the right output to the current level.

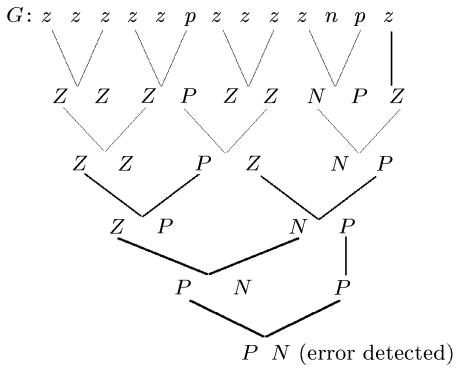

Fig.3. Binary tree to find the pattern $z^i x z^j y \cdots$ of H.

To evident the correctness of the formulations defined in (4), Table 3 shows all the states of transitions from three bits at upper level to two bits at current level for a node of the binary tree according to (4). We can find that in all possible state transitions, three adjacent bits are fairly compressed into two bits by holding the first and second encounters of p or n. So in this way, level by level, all error patterns in Table 2 will be uniquely and exclusively compressed into $P^l N^r$ or $N^l P^r$ finally, and the error-indicating signal can be obtained by $Y = P^l N^r | N^l P^r$ in the last level nodes of the tree.

Table 3. Function of a 3-2 Binary Tree Node

State Transition of the Binary Tree					
Upper Level	Current Level	Upper Level	Current Level	Upper Level	Current Level
PPP	*PP*	*ZPP*	*PP*	*NPP*	*NP*
PPZ	*PP*	*ZPZ*	*PZ*	*NPZ*	*NP*
PPN	*PP*	*ZPN*	*PN*	*NPN*	*NP*
PZP	*PP*	*ZZP*	*ZP*	*NZP*	*NP*
PZZ	*PZ*	*ZZZ*	*ZZ*	*NZZ*	*NZ*
PZN	*PN*	*ZZN*	*ZN*	*NZN*	*NN*
PNP	*PN*	*ZNP*	*NP*	*NNP*	*NN*
PNZ	*PN*	*ZNZ*	*NZ*	*NNZ*	*NN*
PNN	*PN*	*ZNN*	*NN*	*NNN*	*NN*

Fig.4 shows one style of circuit implementation constructed by simple AND and OR gates according to (4). It should be pointed out that in our actual implementation, it is not necessary to calculate all the N's logic for each level, since $N = \overline{P|Z}$ and the output Y can be generated by $Y = P^l N^r | N^l P^r = (P^l \wedge P^r)\overline{Z^l}\,\overline{Z^r}$.

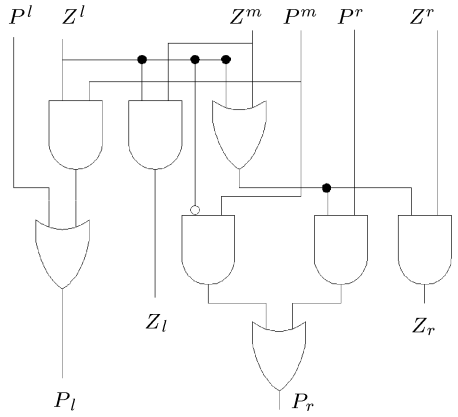

Fig.4. Implementation of the binary tree.

According to Fig.4, three-gate delay is introduced at each level of the binary tree. Furthermore, we can find that at each level, Z_l, Z_r are generated by two-gate delay at most, and the P_l, P_r are generated by three-gate delay at most. On the other hand, the P^l, P^m, P^r are only from the second gate level. So for each P and Z level by level, only two-gate delay is introduced instead of three actually.

Fig.5 shows the whole architecture of our proposed LZA with parallel error detection scheme. We can see that the LZA has two main independent parts. The right part is a general case inexact LZA design, and the left part generates the error correction signal in parallel by using a binary tree, which spends about $O(\log_{1.5}^{n})$ time on generating the error-indicating signal, where n is the length of the input operands.

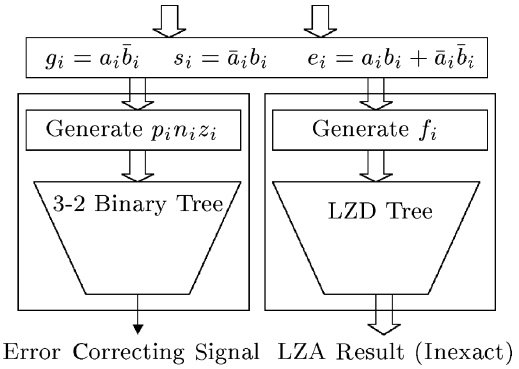

Fig.5. Architecture of our proposed LZA.

4 Experimental Results

We implemented the proposed LZA with error detection algorithm described in Section 3. We also applied our LZA scheme to the design of a double precision floating-point addition unit. The structure of the floating-point addition unit is based on [5], but some differences exist in both FAR and CLOSE path in the implementation of our design.

Fig.6 shows the details how the LZA with concurrent correction scheme is used in the CLOSE path for our dual-path based floating-point adder. The input operands FA and FB are 53-bit long according to IEEE double precision floating-point mantissa. After the small align operation, both of the LZA and normalization are constructed with 54-bit input wide. The LZA scheme generates a 6-bit output for counting leading zeros and a one-bit error-indicating signal denoted by Y. The normalization is divided into three stages, and each stage performs a shifter from 1 to 4 positions (4-1 MUX). The correction is performed at the last stage of normalization shift, so that the shifter at the last stage has to be modified to shift from 1 to 5 positions, this should have a little effect on the delay at the last stage. On the other hand, the demand for the arrival time of error signal Y is loose because the correction is processed only at the last stage of normalization.

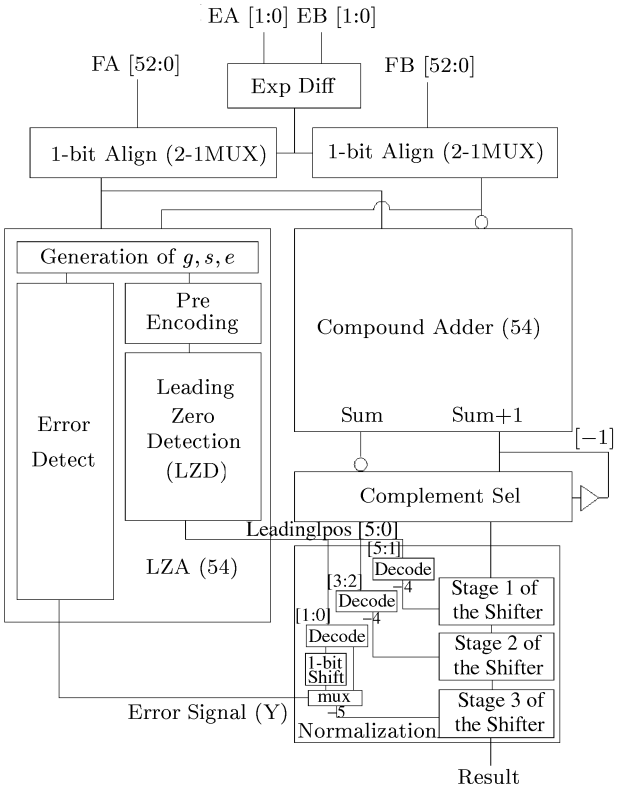

Fig.6. CLOSE path of FP adder using our LZA.

4.1 Timing Critical Path Analysis

Table 4 shows the timing and area of each part of the CLOSE path for floating-point addition in Fig.6. The delay and area of the post-normalization are also given in the table for comparison. The post-normalization occurs only when the LZA and normalization is inexact and a separate correction stage is needed, and hence it is not needed in our design. All results in Table 4 are derived from Synopsys P&R tools with 90nm typical standard cell library for ST technology.

According to Fig.6, there exist three possible timing critical path totally in the floating-point adder design:

• path A goes through compound adder and normalization shift;

• path B goes through the conventional inexact LZA and normalization shift; and

• path C goes through the error detection logic of LZA and the last stage of the normalization shift.

Table 4. Delay and Cost of the CLOSE Path

	Modules	Delay (ns)	Area (μm^2)
01	Exp Difference	0.11	50.5
02	1-bit Align	0.15	800.1
03	Generating g, s, e	0.08	1817.6
04	Pre-Encoding Logic	0.13	3870.1
05	LZD	0.24	1521.3
06	Error Detection	0.53	7181.6
07	Compound Adder	0.40	8808.2
08	Complement Select	0.14	1573.9
09	2-4 Decoder	0.05	26.6
10	4-1 Shifter	0.08	1749.6
11	5b MUX	0.06	169.1
12	5-1 Shifter	0.10	1948.2
13	If Post-Normalization	0.14	981.2

The delay of each path can be calculated respectively according to Table 4:

$$\begin{aligned} D_A &= D_{01} + D_{02} + D_{07} + D_{08} + 2D_{10} + D_{12} = 1.06\text{ns}, \\ D_B &= D_{01} + D_{02} + D_{03} + D_{04} + D_{05} + D_{09} + 2D_{10} \\ &\quad + D_{12} = 1.02\text{ns}, \\ D_C &= D_{01} + D_{02} + D_{03} + D_{06} + D_{11} + D_{12} = 1.03\text{ns}. \end{aligned}$$

We can find that only the path A is a critical timing path, and the proposed LZA with parallel error detection is not a timing critical part because $D_B < D_A$ and $D_C < D_A$. So the total delay of the CLOSE path is:

$$D_{\text{close}} = \max(D_A, D_B, D_C) = 1.06\text{ns}.$$

If we do not use parallel error detection scheme proposed in this paper, the separate error correction stage (post-normalization) must be used, so the total delay of the CLOSE path with post-normalization should be calculated by:

$$D'_{\text{close}} = D_{01} + D_{02} + D_{07} + D_{08} + 3D_{10} + D_{13} = 1.18ns.$$

Now we can find that $(D'_{\text{close}} - D_{\text{close}})/D'_{\text{close}} = 10.2\%$ delay reduction can be achieved with our parallel error detection scheme in total delay for the CLOSE path of floating-point adder design.

4.2 Comparison with Other Work

The most similar work has been proposed in [10, 11]. In [10], a 2-1 compressing tree is constructed to detect error patterns, but it only considers the positive subtraction result, which is only suitable for the cases in which the two operands have been compared and exchanged with each other in the early stage of floating point addition. So in [11], the authors extend their work by using two separate detection trees, to implement a general case error detection logic, one for positive result and the other for negative result, thus introduce it to much area and circuit complexity. Fig.7 shows the architecture of its two-tree based correction scheme described elaborately in [11].

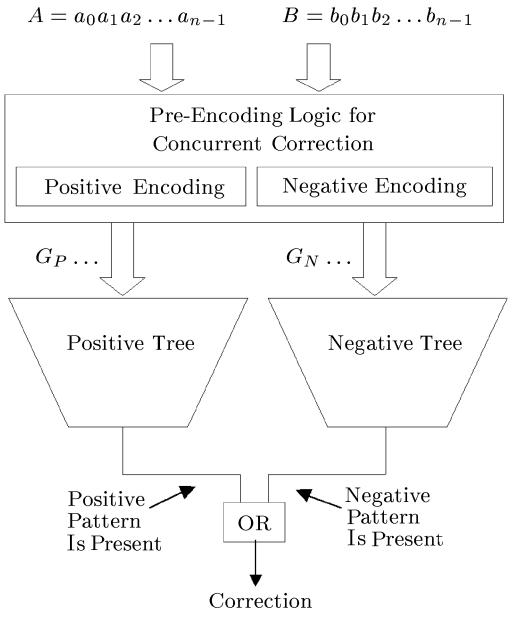

Fig.7. Architecture of error detection scheme in [11].

To compare this work with our proposed work in this paper fairly, we also implement its algorithm described in [11] with the same technology and physical design flow as incorporated into our floating-point adder described before. We have also successfully checked the logic formality between its circuits with our proposed circuits using Synopsys Formality$^{\text{TM}}$ tool. The delay and area results are given in Table 5, Column (1) represents the experimental results of error detection scheme in [11], and Column (2) represents the experimental result of the proposed scheme in this paper. Furthermore, we also simulated the circuits with 100,000 random input vectors to calculate their average power using Synopsys PrimePower$^{\text{TM}}$ tool, which includes dynamic power and leakage power respectively.

Table 5. Comparison Result with Previous Work in [11]

	(1)	(2)	Reduction
Delay (ns)	0.42	0.53	−26.2%
Area (μm^2)	9902.5	7181.6	27.5%
Dynamic Power	6.056	3.888	35.9%
Leakage Power	2.919	2.248	23.0%
Total Power (mw)	8.975	6.136	31.6%

From Table 5, firstly we can find that the timing path of our proposed scheme is a little longer than that of [11], because our method uses about $\log^n_{1.5}$ level 3-2 binary tree for error detection, while the algorithm in [11] uses two parallel $\log^n_2$ level 2-1 binary trees for error detection. However, just as has been analyzed before, our proposed error detection scheme still can satisfy the timing requirement and would not be a part of the critical path. But the error detection scheme in [11] cannot improve more on the timing path in the floating-point

adder. On the other hand, 27% and 28% reduction of the area and power by our scheme can be achieved respectively, compared with its scheme for we use only one detection binary tree instead of two. Therefore, as a whole, the parallel error detection scheme proposed in this paper has improved distinctly on the work in [11].

5 Conclusion

In conclusion, a novel design is given in this paper to detect the error prediction of conventional LZA and correct it in the normalization concurrently, and thus results in up to 10.2% reduction in critical path delay. This design can work with general case for leading zeros of the absolute result of subtraction, and thus is suitable for the design of high-speed dual-path based floating-point adder. The circuit implementation also shows its advantages of area and power over other previous algorithms.

References

[1] Schmookler M S, Nowka K J. Leading zero anticipation and detection—A comparison of methods. In *Proc. 15th IEEE Symposium on Computer Arithmetic*, Vail, CO, USA, June 11–13, 2001, pp.7–12.

[2] Montoye R K, Hokenek E, Runyon S L. Design of the IBM RISC system/6000 floating-point execution unit. *IBM Journal of Research and Development*, 1990, 34(1): 59–71.

[3] Oberman S. Floating-point arithmetic unit including an efficient close data path. US Patent 6094668, AMD, 2000.

[4] Gorshtein V, Grushin A, Shevtsov S. Floating-point addition methods and apparatus. US Patent 5808926, Sun Microsystems, 1998.

[5] Peter Michael Seidel, Guy Even. Delay-optimized implementation of IEEE floating-point addition. *IEEE Transactions on Computers*, 2004, 53(2): 97–113.

[6] Hays W *et al.* A 32-bit VLSI digital signal processor. *IEEE Journal of Solid State Circuits*, October 1985, 20(5): 998–1004.

[7] Hokenek E, Montoye R. Leading-zero anticipator (LZA) in the IBM RISC system/6000 floating point execution unit. *IBM Journal of Research and Development*, 1990, 34(1): 71–77.

[8] Suzuki H *et al.* Leading-zero anticipatory logic for high-speed floating point addition. *IEEE Journal of Solid State Circuits*, 1996, 31(8): 1157–1164.

[9] Gerwig G, Kroener M. Floating-point unit in standard cell design with 116-bit wide dataflow. In *Proc. 14th IEEE Symposium on Computer Arithmetic*, Adelaide, Australia, April 14–16, 1999, pp.266–273.

[10] Bruguera J, Lang T. Leading-one prediction scheme for latency improvement in single datapath floating-point adders. In *Proc. International Conference on Computer Design*, Austin, Texas, USA, October 5–7, 1998, pp.298–305.

[11] Bruguera J, Lang T. Leading-one prediction with con-current position correction. *IEEE Transactions on Computers*, 1999, 48(10): 298–305.

[12] Quach N, Flynn M. Leading one prediction—Implementation, generalization, and application. Technical Report CSL-TR-91-463, Stanford University, March 1991.

[13] Oklobdzija V. An implementation algorithm and design of a novel leading zero detector circuit. In *Proc. 26th IEEE Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, USA, October 26–28, 1992, pp.391–395.

[14] Oklobdzija V. An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis. *IEEE Transactions on VLSI Systems*, 1993, 2(1): 124–128.