A Novel Design of Leading Zero Anticipation Circuit With Parallel Error Detection

Ge Zhang Zichu Qi Weiwu Hu
Institute of Computing Technology, CAS, Beijing, 100080, China
Graduate School of the CAS, Beijing, 100039, China
{gzhang, qizichu, hww}@ict.ac.cn

***Abstract*—The algorithm and its implementation of the Leading zero anticipation (LZA) is 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 error has to be detected and corrected by additional logic in the post-normalization process, resulted in longer critical path and impacted overall performance of floating-point unit. This paper presents a novel algorithm and its design for error detection of LZA. The proposed approach enables parallel execution of conventional LZA and its error detection operation, so that the error-indication signal can be generated in the earlier stage of normalization, thus reduced the critical path and improved overall performance. In addition, the proposed LZA with parallel error detection logic can work with general case of operands regardless of polarity of subtraction result, positive or negative. This means that the proposed scheme can be nicely adapted to the CLOSE path design of dual-path based floating-point adder [1]. The circuit implementation and evaluation of this algorithm are also presented in this paper.**

I. INTRODUCTION

Leading Zero Anticipation (LZA) is a technique that predicts the location of the most significant digit in a floating-point addition given the inputs to the adder. This determination of the leading-zero result is performed in parallel with the addition step so as to facilitate immediately start of the normalization shift operation once the addition completes. A great many approaches have been proposed for a faster and simpler LZA logic [2]-[8]. These LZA designs have been incorporated in most realistic floating-point processing units (FPU) and commercial processors. 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 for the possible one-bit error of the LZA [2].

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 same position as the actual sum output; a leading zero detector (LZD) which is then employed to encode the pre-encoding result. The LZA is often used in floating-point adder when the operation is effective subtraction. Leading zeros occur when the result of subtraction is positive, and leading ones occur when the result of subtraction is negative. Some LZAs are simply designed under the assumption that the result is always positive [3] [4]. Most of the LZAs are inexact because of possible one bit of error in their results [2].

In this paper, we propose a general-case leading zero anticipator with parallel error detection based on pattern matching technique. This anticipator works well for the leading zeros predication regardless of positive or negative subtraction result, so it is suitable for the high-speed dual-path based floating-point addition units [1]. In addition, the proposed error detection operation is accomplished directly from the two inputs of the subtraction, thus has no impact to the original circuits of the adder and LZA.

The rest of the paper is organized as follows: Section Ⅱ presents an overview of previous work on Leading Zero Anticipation. In Section Ⅲ, the proposed LZA design with a novel algorithm for error detection is presented. Some experimental results based on physical layout implementation are presented in Section Ⅳ, followed by conclusion in Section Ⅴ.

II. PREVIOUS WORK ON LZA

Leading Zero Anticipation is a very well researched topic and a number of techniques have been presented in the literature [2]-[8]. The first LZA scheme was proposed in [5] , which can 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 [6]. Paper [3] provides a simpler circuit based on the assumption that the result of subtraction is always positive, which is only suitable for the cases that 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 greater than the LZA's result. This error must be

Supported by the National High Technology Development 863 Program of China under Grant No. 2002AA111100 & No. 2002AA110010

corrected after the addition and normalization shift. In the evaluation performed in [3], this correction process accounts for 12.5% of the delay of addition plus shifting. Paper [7] describes an exact LZA logic that incorporates with the adder, but the problem is that the count of leading zeros can't be generated as soon as the addition step finishes, which deteriorates the critical path from addition to normalization and also leads to circuit complexity.

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. This method has been proved as an effective way to reduce the critical path delay [4]. A conceptually simpler scheme for error detection was proposed in [8] to detect the error through an examination of each carry-in bit of the addition. However, it is not always practical to use such carries without affecting the design of the adder. The error detection scheme proposed in [4] uses a binary tree to match a special string pattern, which is only dependent on the two inputs of the subtraction. This method effectively reduced the delay and isolated the detection logic from other circuits. However, it is only applicable to the case where the result of subtraction is positive. In the following sections, we will propose a novel design for LZA with parallel error detection, which is independent on original inexact LZA and adder and can work with any case of subtraction results, positive or negative.

III. PROPOSED LZA WITH PARALLEL ERROR DETECTION

Now let's consider the subtraction of two inputs with magnitude representation: A and B,

$$A = a_0\, a_1\, a_2\, ...a_{n-1};\quad B = b_0\, b_1\, b_2\, ...b_{n-1};$$

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 LZA design's 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:

$$\begin{aligned} g_i &= a_i \overline{b_i} && (a_i > b_i) \\ s_i &= \overline{a_i} b_i && (a_i < b_i) \\ e_i &= a_i b_i + \overline{a_i}\,\overline{b_i} && (a_i = b_i) \end{aligned} \qquad (1)$$

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. The superscript i, j or k denote the number of times for a pattern's appearance in this table and 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

Table 1. Pattern sequences for leading one position

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

the pattern of W meets the cases $e_{i-1}\overline{g_i}\,\overline{s_{i+1}}$, $\overline{e_{i-1}}\,\overline{s_i}\,\overline{s_{i+1}}$ (for A>B) or the cases $e_{i-1}\overline{s_i}\,\overline{g_{i+1}}$, $\overline{e_{i-1}}\,\overline{g_i}\,\overline{g_{i+1}}$ (for A<B). So we can construct a binary string whose number of leading zeros is approximately the same as the number of leading zeros of the actual result of subtraction, denoted by F:

$$f_i = e_{i-1}g_i\overline{s_{i+1}} + \overline{e_{i-1}}\,\overline{s_i}\,\overline{s_{i+1}} + e_{i-1}s_i\overline{g_{i+1}} + \overline{e_{i-1}}\,\overline{g_i}\,\overline{g_{i+1}}$$

$$\text{when } i = 0,\ e_{i-1} = 0 \qquad (2)$$

The binary string F can be calculated by a well-known process called "pre-encoding" using the value of each digit and its two neighbors of the string W. After this, an LZD [9] 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. This is evident from Table.1 that when the pattern sequence of string W is $e^ige^js^{\cdots}$, $e^ise^jg^{\cdots}$, $e^igs^je^ks^{\cdots}$, $e^isg^je^kg^{\cdots}$, and the leading one's position in binary string F could be off by one position to the left from the actual leading one's position. To solve this problem, a proposed new scheme will be discussed in the following subsections.

Error Detection of LZA

In this subsection we'll present a 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:

$$\begin{aligned} p_i &= (e_{i-1} | e_{i-2}g_{i-1} | \overline{e_{i-2}}\ s_{i-1})g_i \\ n_i &= (e_{i-1} | e_{i-2}s_{i-1} | \overline{e_{i-2}}\ g_{i-1})s_i \\ z_i &= \overline{p_i \mid n_i} \end{aligned}$$

$$\text{when } i = 0,\ e_{i-1}=e_{i-2}=1,\ g_{i-1}= s_{i-1} = 0 \qquad (3)$$

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 $z^ixz^jy^{\cdots}$, where x, y indicate the first appearances of p,n or n,p in string H respectively.

A binary tree is constructed here to detect the pattern $z^ixz^jy^{\cdots}$ of H. The basic theory of this binary tree is through

Table 2. Pattern sequences of H

Pattern Sequence		Error Prediction
Original W	Denoted by H	
$e^i gg\ldots$	$z^i pp\ldots$	No
$e^i ge^j g\ldots$	$z^i pz^j p\ldots$	No
$e^i ge^j$	$z^i pz^j$	No
$e^i ge^j s\ldots$	**$z^i pz^j n\ldots$**	**Yes**
$e^i gs^j g\ldots$	$z^i pz^j p\ldots$	No
$e^i gs^j e^k g\ldots$	$z^i pz^{j+k} p\ldots$	No
$e^i gs^j e^k$	$z^i pz^{j+k}$	No
$e^i gs^j e^k s\ldots$	**$z^i pz^j n\ldots$**	**Yes**
$e^i ss\ldots$	$z^i nn\ldots$	No
$e^i se^j s\ldots$	$z^i nz^j n\ldots$	No
$e^i se^j$	$z^i nz^j$	No
$e^i se^j g\ldots$	**$z^i nz^j p\ldots$**	**Yes**
$e^i sg^j s\ldots$	$z^i nz^j n\ldots$	No
$e^i sg^j e^k s\ldots$	$z^i nz^{j+k} n\ldots$	No
$e^i sg^j e^k$	$z^i nz^{j+k}$	No
$e^i sg^j e^k g\ldots$	**$z^i nz^j p\ldots$**	**Yes**

compressing three adjacent bits of H to two adjacent bits for each level, and then finally converting H to a two-bit presentation which only holds the first and second encounters of p or n of H. The logic equations to form the binary tree are defined here,

$$P_l = P^l + Z^l P^m$$
$$P_r = \overline{Z^l}\, P^m + P^r (Z^l + Z^m)$$
$$N_l = N^l + Z^l N^m$$
$$N_r = \overline{Z^l}\, N^m + N^r (Z^l + Z^m)$$
$$Z_l = Z^l Z^m$$
$$Z_r = Z^r (Z^l + Z^m) \qquad (4)$$

where the superscript l, m and r respectively denote the left input, the middle input and the right input from the upper level, and the subscript l and r respectively denote the left output and the right output to the current level. Table 3 shows all the states of transitions from three bits of upper level to two bits of current level for a node of the binary tree. Fig. 1 describes an example of this binary tree, and the error-indicating signal can be obtained by $Y = P^l N^r \mid N^l P^r$ in the last level nodes of the tree.

Table 3. Function of a 3-2binary 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

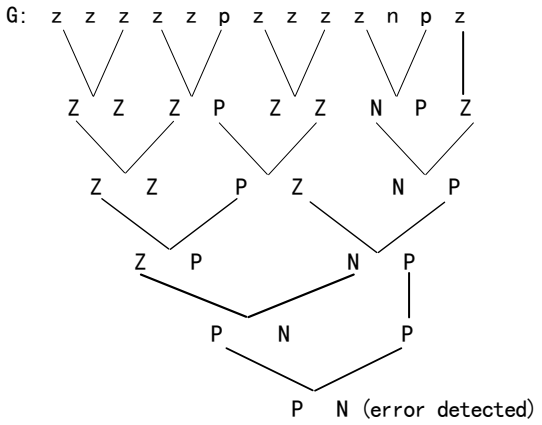

Figure 1. A binary tree to find the pattern $z^i x z^j y\ldots$.of H.

The algorithm described above spends $O(\log_{1.5}{}^n)$ time to generate the error-indicating signal, where n is the length of the input operands.

IV. Experiment Results

We implemented the proposed LZA with error detection algorithm described in Section Ⅲ. 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 [1], but some differences exist in both FAR and CLOSE path in the implementation of our design.

Fig. 2 shows the details how the LZA circuit with concurrent correction scheme is used in the CLOSE path for our dual-path based high speed 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 6-bit output for counting of leading zeros and 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 of last stage has to be modified to shift from 1 to 5 positions, this should have a little effect on the delay of 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 in the last stage of normalization.

Table 4 shows the timing and area of each part of the CLOSE path for floating-point addition in Fig. 2. The delay and area of the post-normalization are also given in the table for analysis. The post-normalization occurs only when the LZA and normalization is inexact and separate correction stage is needed, and hence it is not needed in this design. All results in Table 4 are derived from Synopsys P&R tools with Artisan's 0.18um standard cell library for SMIC. It should be pointed out here that in our implementation of the binary tree according to equation (4), it is not necessary to calculate the

N's logic for each level since $N = \overline{P \mid Z}$ and the output Y will be generated by $Y = (P^{l} \wedge P^{r})\,\overline{Z^{l} Z^{r}}$.

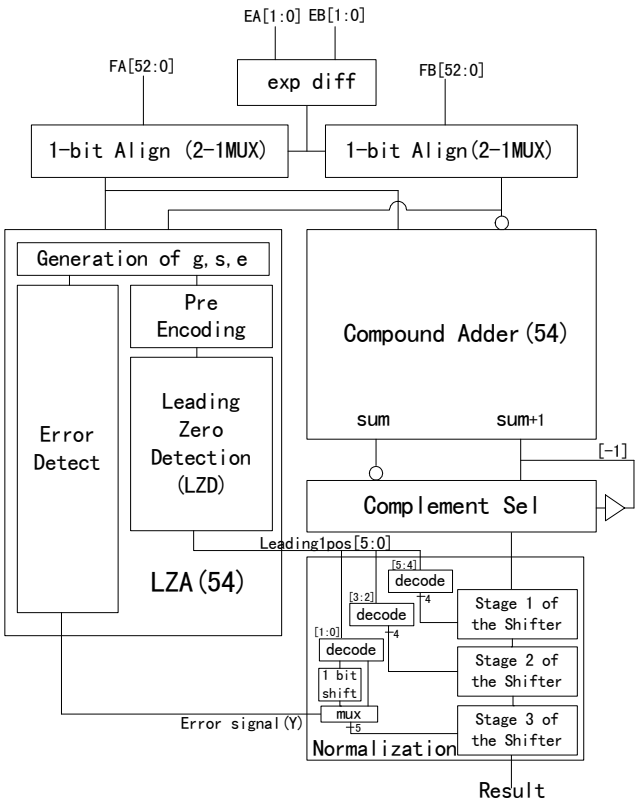

Figure 2. CLOSE path of FP adder using our LZA

The total delay of the CLOSE path in Fig. 2 can be calculated according to Table 4,

$D_{close} = D_{01}+D_{02}+D_{08}+D_{09}+2D_{10}+D_{11} = 2.43\text{ns}$

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

$D`_{close} = D_{01}+D_{02}+D_{08}+D_{09}+3D_{10}+D_{12} = 2.67\text{ns}$

Table 4. Delay and Cost of the CLOSE path

	Modules	Delay (ns)	Area (um^2)
01	Exp difference	0.25	136
02	1 bit align	0.31	1866
03	Generating of g, s, e	0.16	3885
04	Pre-encoding logic	0.27	6742
05	LZD	0.71	5845
07	Error detection	1.45	18261
08	Compound adder	1.04	25413
09	Complement select	0.29	1726
10	1-4 Shifter	0.16	3350
11	1-5 Shifter	0.22	3968
12	If post-normalization	0.30	2062

As shown from Table 4, the error detection schemes proposed in this paper can satisfy the timing requirement and wouldn't be a part of the critical path, and gives 9.0% reduction in delay for the CLOSE path of floating-point adder design. The adding of area for concurrent error detection is considerable for the LZA. However, its effect on the whole floating-point unit should be small and the layout should be easy because of its good circuit independence.

V. CONCLUSION

In conclusion, a novel design is proposed in this paper to detect the error prediction of conventional LZA and correct it in the normalization concurrently, thus resulted up to 9% reduction in critical path delay, as indicated by physical layout implementation. This design can work with general case for leading zeros of the absolute result of subtraction, and thus suitable for the design of high speed dual-path based floating-point adder.

ACKNOWLEDGMENT

The first author would like to thank Professor Ming Zeng from Intel Corp. He took time off his busy schedule to provide valuable suggestions and modifications for improving this paper.

REFERENCES

[1] Peter-Michael Seidel and Guy Even, "Delay-Optimized Implementation of IEEE Floating-Point Addition" IEEE Transactions on computers, Vol. 53, No. 2, February, 2004.

[2] Martin S. Schmookler and Kevin J. Nowka, "Leading Zero Anticipation and Detection – A Comparison of Methods", Proc. of the 15th IEEE Symposium on Computer Arihmetic, 2001.

[3] H. Suzuki, H. Morinaka, H. Makino, Y. Nakase, K. Mashiko, T. Sumi, "Leading-zero Anticipatory Logic for High-speed Floating Point Addition", IEEE Journal of Solid State Circuits, August , pp. 1157-1164, 1996.

[4] J. Bruguera and T. Lang, "Leading-One Prediction with Con- current Position Correction", IEEE Transactions on Computers, v. 48, No. 10, October pp. 298-305, 1999.

[5] W. Hays, R. Kershaw, L. Bays, J. Bodie, E. Fields, R. Freyman, C. Garen, J. Hartung, J. Klinikowski, C. Miller, K. Mondal, H. Moscovitz, Y. Rotblum, W. Stocker, L. Tran, "A 32-bit VLSI Digital Signal Processor", IEEE Journal of Solid State Circuits, October pp. 998-1004, 1985.

[6] E. Hokenek and R. Montoye, "Leading-Zero Anticipator (LZA) in the IBM RISC System/6000 Floating Point Execution Unit", IBM Journal of Research and Development, pp. 71-77, 1990.

[7] G. Gerwig and M. Kroener, "Floating Point Unit in standard cell design with 116 bit wide dataflow", IEEE Symposium on Computer Arithmetic. pp 266-273, 1999.

[8] N. Quach and M. Flynn, "Leading One Prediction – Implementation, Generalization, and Application", Technical Report CSL-TR-91-463, Stanford University, March, 1991.

[9] V. Oklobdzija, "An Implementation Algorithm and Design of a Novel Leading Zero Detector Circuit", 26th IEEE Asilomar Conference on Signals, Systems, and Computers, pp. 391- 395, 1992.