# Leading Zero Anticipation for Latency Improvement in Floating-Point Fused Multiply-Add Units

MEI Xiao-Lu[1, 2]

[1](Institute of Computing Technology, Chinese Academy of Sciences Beijing 100080)
[2](Graduate School of Chinese Academy of Sciences, Beijing 100039)
Email: xlmei@ict.ac.cn
Tel: 13661375741

## Abstract

The leading zero anticipation (LZA) is vital in the floating-point fused multiply-add (FMA) units. The general LZA algorithms can only deal with 2 operands. It increases the critical path delay of high performance floating-point FMA units. The paper presents a novel LZA algorithm to deal with 3 operands directly and implemented the 106-bit leading zero anticipator in the high performance floating-point FMA with the general LZA algorithm and the proposed LZA algorithm respectively. Compared with the general leading zero anticipator, the proposed leading zero anticipator can reduce the delay of the critical path by 16.67% and reduce the area by 19.63% approximately.

**Key words**: leading zero anticipation; fused multiply-add

## 1 Introduction

The fused multiply-add (FMA) unit is very important in many scientific and engineering applications. It is a key feature of the floating-point unit (FPU), which greatly increases the floating-point performance and accuracy. The FPUs of several commercial processors have included a floating-point fused multiply-add units, such as Intel Itanium, IBM RS/6000, POWER3.

The fused multiply-add unit computes D = A+B*C. The leading zero anticipator predicts the location of the most significant bit of D. It's pivotal to the normalization of results for fused multiply-add units in high-performance floating point processors. General leading zero anticipation (LZA) algorithms deal with 2 operands. In fused multiply-add units, the result of multiply tree is 2 numbers, $SUM_{BC}$ and $CARRY_{BC}$. The addend $A$ needs to be added to $SUM_{BC}$ and $CARRY_{BC}$. In basic structures for floating-point FMA, the leading zero anticipator is not on the critical path. So $SUM_{BC}$, $CARRY_{BC}$ and $A$ first input a level of 3:2 Carry-Save Adders (CSAs) to be compressed into 2 operands, and then input the adder and leading zero anticipator in parallel. But, in high performance structure for floating-point FMA, the leading zero anticipator is on the critical path and the general LZA algorithms will increase the critical path delay.

This paper proposes a novel LZA algorithm, which can deal with 3 operands directly and doesn't need a level of 3:2 CSAs. This LZA algorithm is suitable for the high performance floating-point FMA units.

The paper is organized as follows: in Section 2, the general LZA algorithms in FMA units are described. The proposed LZA algorithm is presented in Section 3. In Section 4, we make a experiment and compare the general LZA and proposed LZA in terms of delay of the critical path and added hardware complexity. Finally, we draw a conclusion in Section 5.

## 2 General scheme of LZA in floating-point fused multiply-add units

Figure 1 shows the major building blocks of basic structure for the floating-point FMA units, implemented in several floating-point units of general purpose processors [1,2,3]. The multiply tree reduces partial products to 2 numbers. Because leading zero anticipator is not on the critical path, the $SUM_{BC}$, $CARRY_{BC}$ and addend $A$ first input to a level of 3:2 CSAs to be compressed into 2 numbers, and then the 2 numbers input the adder and leading zero anticipator in parallel.

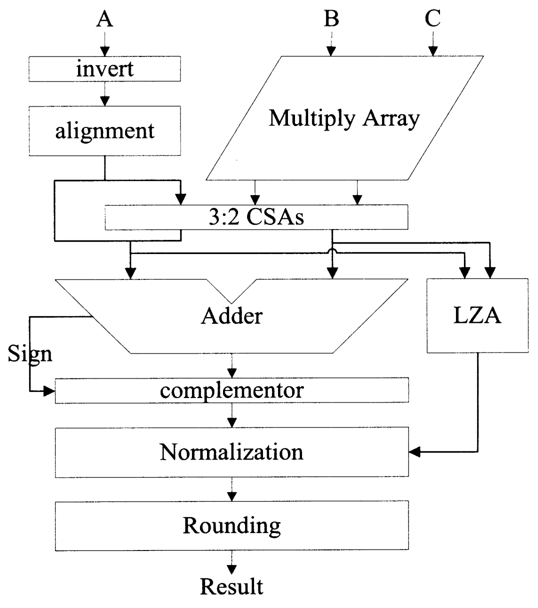


Figure 1. Basic structure for FMA units

The leading zero anticipator includes 2 sequential parts: pre-encoding module and encoding tree. A pre-encoding module provides a string of zeros and ones, with the most-significant 1 defining the leading-one position. After this leading one, the rest of the string is not significant. An encoding tree encodes the position of

This work was supported by the National 863 Program fund (2002AA110010 and 2002AA111100)

the most-significant 1 in a way that is used to control the normalization shifter.

Paper [4] shows a typical LZA algorithm among many previous LZA algorithms. Assuming the result of 3:2 CSAs are named *A* and *B*.

$$A = a_0 a_1 \ldots a_{n-1}, B = b_0 b_1 \ldots b_{n-1}, a_i, b_i \in \{0,1\}$$

*A*, *B* are 2's complement signed numbers. For each bit position i of operands *A* and *B*, the following functions are defined.

$$t_i = a_i \otimes b_i$$
$$g_i = a_i \cdot b_i$$
$$z_i = \overline{a_i} \cdot \overline{b_i}$$

Then, the indicator F is defined.

$$f_i = t_{i-1} \cdot (g_i \cdot \overline{z_{i+1}} + z_i \cdot \overline{g_{i+1}}) + \overline{t_{i-1}} \cdot (z_i \cdot \overline{z_{i+1}} + g_i \cdot \overline{g_{i+1}})$$

If the indicator is set in position i and no other digit of greater significance has its indicator set, the leading digit is either i or i+1.

Once the string *F* has been obtained, the position of the leading one of *F* has to be encoded by means of a LOD tree [5]. Then, the position is sent to the normalization shifter.

## 3 Proposed scheme of LZA in floating-point fused multiply-add units

Recently, Lang & Bruguera [6] proposed a new structure for floating-point FMA units, whose delay is less than the basic structures by 15% to 20%. See Figure 2. In Lang & Bruguera's structure, normalization is executed before addition. The leading zero anticipator is on one of the critical paths. If general LZA algorithm is used, $SUM_{BC}$, $CARRY_{BC}$ and *A* must input a level of 3:2 CSAs at the first step, which will increase delay of the critical path.

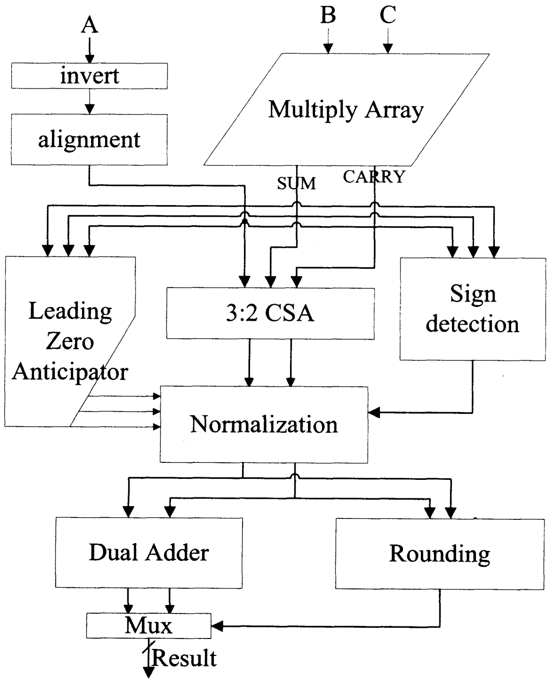


Figure 2. Lang & Bruguera's structure for FMA units

For clarity, we rename $SUM_{BC}$, $CARRY_{BC}$ and addend *A* as *A, B, C*.

For each bit position i of inputs operands *A, B* and *C*, the following functions are defined.

$$s_i = \overline{a_i} \cdot \overline{b_i} \cdot c_i$$
$$e_i = \overline{a_i} \cdot \overline{b_i} \cdot \overline{c_i} + (a_i \oplus b_i) \cdot c_i$$
$$t_i = a_i \cdot b_i \cdot \overline{c_i}$$
$$w_i = a_i \oplus b_i \oplus c_i$$
$$x_i = a_i \cdot b_i + (a_i + b_i) \cdot \overline{c_i}$$

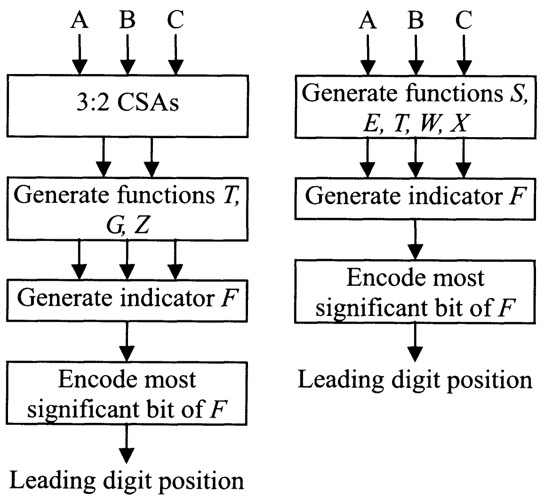
Then, the indicator F is defined.

$$f_i(pos) = e_i \cdot t_{i+1} + w_i \cdot e_{i+1} + w_{i+1} \cdot x_{i+2},$$
$$f_i(neg) = s_i \cdot \overline{x_{i+1}} + \overline{w_i} \cdot t_{i+1}$$

The proposed LZA algorithm needs the sign of the result to determine whether *F(pos)* or *F(neg)* should be chosen.

If the indicator is set in position i and no other digit of greater significance has its indicator set, the leading digit is either i or i+1. Once the string *F* has been obtained, the position of the leading one of *F* has to be encoded by means of a LOD tree [5], like the general LZA.

The difference of the general LZA algorithm described in Section 2 and the LZA algorithm we present here is showed in Figure 3.


A B C
3:2 CSAs
Generate functions T, G, Z
Generate indicator F
Encode most significant bit of F
Leading digit position
A B C
Generate functions S, E, T, W, X
Generate indicator F
Encode most significant bit of F
Leading digit position

a) general LZA b) proposed LZA

Figure 3. General LZA and proposed LZA

To achieve the above formulas, we perform a radix-2 signed-digit subtraction of the significands. We obtain

$$P = A + B - C$$

This operation is done on each bit slice (without carry propagation), that is,

$$p_i = a_i + b_i - c_i, \quad p_i \in \{-1, 0, 1, 2\}$$

For clarity, the -1 will be represented as $\bar{1}$.

To simplify the discussion, we consider the cases P≥0 and P<0 separately. The notation used throughout the paper is the following: x denotes an arbitrary sub-string and $0^k$, $1^k$, and $\bar{1}^k$ denote strings of k 0s, 1s, and $\bar{1}$s, respectively, with $k \geq 0$.

**Case $P \geq 0$**

Since $P \geq 0$, the first digit $p_i$ different from 0 has to be equal 1 or 2. The P strings and corresponding pre-encoding patterns are showed in Table 1.

Table 1. Pre-encoding patterns when P≥0

| *Class of P* | *Leading One Location* | *Pre-encoding pattern* |
|---|---|---|
| $0^k2(x)$ | k *or* k+1 | $p_ip_{i+1}=0_i2_{i+1}$ |
| $0^k12(x)$ | k *or* k+1 | $p_{i+1}p_{i+2}=1_{i+1}2_{i+}$ |
| $0^k11^l(\bar{1},0,2)(x)$ | k *or* k+1 | $p_{i+1}p_{i+2}=1_{i+1}1_{i+}$ |
| $0^k10(x)$ | k+1 *or* k+2 | $p_ip_{i+1}=1_i0_{i+1}$ |
| $0^k1\bar{1}^l2(x)$ | k+l *or* k+l+1 | $p_{i+1}p_{i+2}=\bar{1}_{i+1}2_i$ |
| $0^k1\bar{1}^l1^m(x)$ | k+l *or* k+l+1 | $p_{i+1}p_{i+2}=\bar{1}_{i+1}1_{i+}$ |
| $0^k1\bar{1}^l0(x)$ | k+l+1 *or* k+l+2 | $p_ip_{i+1}=\bar{1}_i0_{i+1}$ |

Combining the above patterns, we can obtain the indicator *F(pos)*.

$$\begin{aligned} f_i(pos) &= 0_i2_{i+1}+1_i0_{i+1}+\bar{1}_i0_{i+1}+1_{i+1}1_{i+2} \\ &\quad +1_{i+1}2_{i+2}+\bar{1}_{i+1}1_{i+2}+\bar{1}_{i+1}2_{i+2} \\ &= 0_i2_{i+1}+(1_i+\bar{1}_i)0_{i+1}+(1_{i+1}+\bar{1}_{i+1})(1_{i+2}+2_{i+2}) \end{aligned}$$

After introducing intermediate numbers *S, E, T, W, X*, which are defined previously, we can simplify the indicator *F(pos)* as follows.

$$f_i(pos)=e_i\cdot t_{i+1}+w_i\cdot e_{i+1}+w_{i+1}\cdot x_{i+2}$$

**Case $P < 0$**

We introduce the opposite number of *P* to analyse pre-encoding patterns, as following.

$Q=-P=C-A-B>0$

For each bit slice,

$q_i=-p_i=c_i-a_i-b_i,\ \ q_i\in\{\bar{2},\bar{1},0,1\}$ 。

Since $Q > 0$, the first digit $q_i$ different from 0 has to be equal 1. The Q strings and corresponding pre-encoding patterns are showed in Table 2.

Combining the above patterns, we can obtain the indicator *F(neg)*.

$$\begin{aligned} f_i(neg) &= \bar{1}_i\bar{1}_{i+1}+\bar{1}_i0_{i+1}+0_i2_{i+1}+2_i2_{i+1} \\ &= \bar{1}_i(\bar{1}_{i+1}+0_{i+1})+(0_i+2_i)2_{i+1} \end{aligned}$$

The indicator *F(pos)* can be simplified as follows.

$$f_i(neg)=s_i\cdot\overline{x_{i+1}}+\overline{w_i}\cdot t_{i+1}$$

Table 2. Pre-encoding patterns when P<0

| *Class of Q* | *Leading-One Location* | *Pre-encoding pattern* | |
|---|---|---|---|
| | | $q_iq_{i+1}$ | $p_ip_{i+1}$ |
| $0^k11(x)$ | k+1 *or* k+2 | $1_i1_{i+1}$ | $\bar{1}_i\bar{1}_{i+1}$ |
| $0^k10(\bar{1},0,1)(x)$ | k+1 *or* k+2 | $1_i0_{i+1}$ | $\bar{1}_i0_{i+1}$ |
| $0^k10\bar{2}^l(x)$ | k+l+1 *or* k+l+2 | $0_i\bar{2}_{i+1}$ *or* $\bar{2}_i\bar{2}_{i+1}$ | $0_i2_{i+1}$ *or* $2_i2_{i+1}$ |
| $0^k1\bar{1}^l(1,0)(x)$ | *As above* | | |
| $0^k1\bar{1}^l\bar{2}(x)$ | *As above* | | |
| $0^k(1\bar{2})^l(x)$ | *As above* | | |

## 4 Experiment Results

In this section, the LZA algorithm we proposed is compared with the general algorithm discussed in Section 2, in terms of delay of the critical path and added hardware complexity. The 2 LZA algorithms both give the position with a possible error of one bit.

We implemented the 106-bit leading zero anticipator in Lang & Brugurea's fused multiply-add unit, using general LZA scheme and proposed LZA scheme respectively. We describe the RTL code with Verilog HDL and synthesize the RTL code with Design Compiler of Synopsys (version 2003.06) and 0.13 micron technology library of Charter Semicon. Table 3 shows the experiment results. Note that, in Lang & Bruguera's structure of FMA, the sign of result is provided by the Sign Detection module and no additional delay would be increased to leading zero anticipator.

Table 3. Comparison of general LZA and proposed LZA

| | General LZA | Proposed LZA | improvement |
|---|---|---|---|
| Delay of critical path | 1.32 ns | 1.10 ns | 16.67% |
| Area | 76,191 um$^2$ | 61,236 um$^2$ | 19.63% |

In Lang & Bruguera's structure of fused multiply-add units, the general LZA scheme needs to pass 3:2 CSA first, which increases the area and the critical path delay. The LZA scheme we proposed can deal with 3 inputs directly. Because the leading zero anticipator is on the critical path of Lang & Bruguera's structure, the proposed LZA scheme can reduce the critical path delay of the whole FMA units. It will benefit the high-performance FMA units.

## 5 Conclusion

Leading zero anticipation (LZA) is a technology that permits the reduction of the delay of floating-point FMA units. The general LZA algorithms deal with 2 operands. In floating-point FMA units, if the general LZA algorithms is used, the leading zero anticipator needs a level of 3:2 CSAs to compress 3 operands into 2. In basic structure for FMA, the leading zero anticipator is not on the critical path. Introducing a level of 3:2 CSAs will not affect the critical path delay. But in Lang & Bruguera's structure for FMA, the leading zero anticipator is on the critical path. Using general LZA schemes will increase the critical path delay.

We have presented a novel LZA algorithm and its implementation, which can deal with 3 operands directly. The proposed LZA doesn't need a level of 3:2 CSAs and can improve the performance of the floating-point FMA units.

We implemented the 106-bit leading zero anticipator in Lang & Bruguera's FMA with the general LZA algorithm and the proposed LZA algorithm respectively. Compared with the design using the general LZA algorithm, the one using proposed LZA algorithm can reduce the critical path delay by 16.67% and reduce the area by 19.63% approximately.

## Refenrnce

[1] Hokenek, E.; Montoye, R.K.; Cook, P.W.; Second-generation RISC floating point with multiply-add fused. IEEE Journal of Solid-State Circuits, 1990, Volume 25, Issue 5, Page(s):1207 – 1213.

[2] F. P. O'Connell, and S. W. White, "Power3: The Next Generation of PowerPC Processors," IBM Journal of Research and Development. Vol. 44, no. 6, pp. 873 – 884, 2000.

[3] R. M. Jessani, and M. Putrino, "Comparison of Single- and Dual-Pass Multiply-Add Fused Floating-Point Units," IEEE Transactions on Computers. Vol. 47, no. 9, pp. 927 – 937, 1998.

[4] Martin S Schmookler, Kevin J Nowka. Leading Zero Anticipation and Detection – A Comparison of Methods. Vail, Colorado: Proceedings of the 15th IEEE Symposium on Computer Arithmetic, 2001, Page(s):7 – 12.

[5] Vojin G. Oklobdzija. An Algorithmic and Novel Design of a Leading Zero Detector Circuit: Comparison with Logic Synthesis. IEEE Transactions on VLSI Systems, 1994, Vol. 2, No. 1, pp. 124-128.

[6] Tomas Lang, Javier D Bruguera. Floating-Point Multiply-Add-Fused with Reduced Latency. IEEE Transactions on Computers, 2004, Volume 53, Issue 8, Page(s):988 – 1003.