Hybrid LZA: A Near Optimal Implementation of the Leading Zero Anticipator

Amit Verma[*]
amit.verma.nitrkl@gmail.com
Electronics and Communication Engineering
National Institute of Technology (NIT)
Rourkela, India

Ajay K. Verma
ajaykumar.verma@epfl.ch

Philip Brisk
philip.brisk@epfl.ch

Paolo Ienne
paolo.ienne@epfl.ch

Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
CH-1015 Lausanne, Switzerland

ABSTRACT

The *Leading Zero Anticipator* (LZA) is one of the main components used in floating point addition. It tends to be on the critical path, so it has attracted the attention of many researchers in the past. Most LZAs used today can be classified in two categories: exact and inexact. Inexact LZAs are normally preferred due to their shorter critical paths and reduced complexity; however, the inexact LZA requires an additional correct stage. In this paper we present a new LZA architecture that combines ideas taken from prior exact and inexact LZAs. Our new LZA improves the delay of floating point addition by 7–10% compared to state of art techniques as well as reduces hardware area in most cases. We also establish theoretical lower bounds on the delay of an LZA and we show that our LZA is very close to these bounds.

1. INTRODUCTION AND MOTIVATION

In a standard floating point representation (e.g., IEEE 754) a real number is represented using three values: a sign bit, mantissa and exponent; the value of the real number is

$$r = (-1)^{sign} \times 1.mantissa \times 2^{exponent}.$$

In this representation *most significant bit* (MSB) of the real number (which is always 1) is hidden. To add or subtract two floating point numbers, first their exponents must be matched by shifting $(1.mantissa)$ of one of the two numbers to the right. Then the shifted/unshifted $(1.mantissa)$ of the two numbers are added or subtracted. Since the result of this addition/subtraction may not be of the form $(1.xxx)$, it is necessary to shift the result to the left until the MSB is 1. This process is called *Normalisation*; in order to find the proper shift amount, a *Leading Zero Anticipator* (LZA) is required. Fig. 1 illustrates a simple floating point adder/subtracter.

Throughout the remainder of this paper, we assume that the final result of the floating point addition/subtraction is positive; in the case of a negative result, the leading ones, rather than the leading zeros, must be anticipated. In the general case one can use both an

[*]This work was performed while Amit Verma was with the Processor Architecture Laboratory, Swiss Federal Institute of Technology Lausanne.

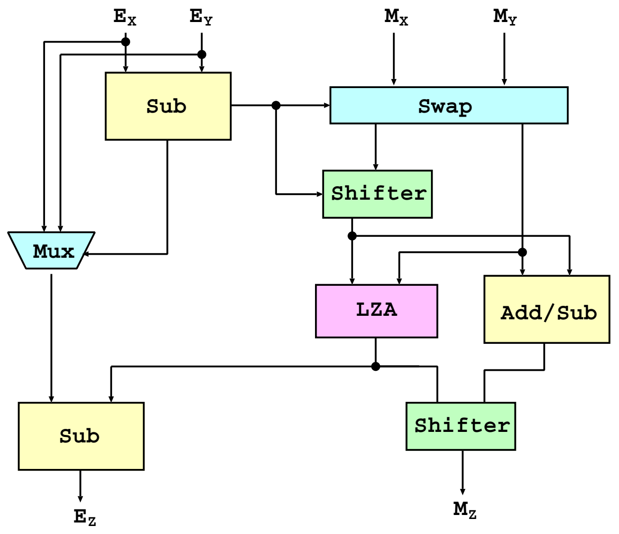

Figure 1: A straightforward implementation of floating point addition. The output of the LZA is an input to the final shifter. Therefore a slow LZA architecture will increase the critical path delay.

LZA and a *Leading One Anticipator* (LOA) and select the appropriate output based on the sign bit of the addition.

Formally, the inputs to an LZA are two integers and the output is the number of leading zeros in their sum, if they were to be added; the leading zeros do not include the carry-out bit. A naive implementation is to add the integers and then count the number of leading zeros using a *Leading Zero Detector* (LZD), i.e.:

$$LZA(A, B) = LZD(A + B).$$

The disadvantage of this approach is that leading zero detection cannot begin until after the two integers are added. To address this shortcoming, several LZA architectures have been proposed which work in parallel with the addition; all modern LZAs are based on this approach, and can be classified as either *exact* or *inexact* (E- or

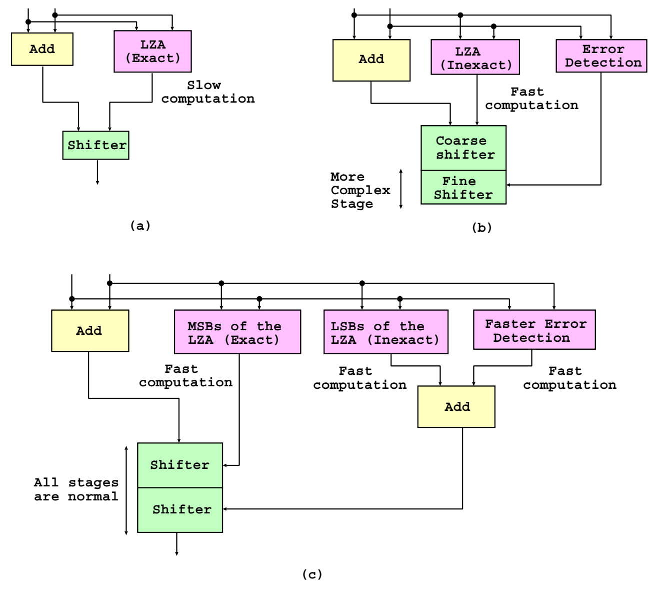

Figure 2: An illustration of the various implementations of the normalisation step in floating point addition. (a) The first method uses an E-LZA and suffers from a longer critical path delay. (b) The second method uses an I-LZA, which is faster, but suffers from the overhead of the fine-grained shifter.(c) The third implementation is the one proposed here, which combines ideas from both E- and I-LZA architectures.

I-LZA).

The E-LZA always produces a correct result, but may be slower and larger than an I-LZA; this is detrimental to performance because the LZA and final shifter are on the critical path of the floating point adder. The I-LZA, in contrast, is faster and smaller, but produces a result that may differ by 1 from the exact result; this requires an additional shifting stage to compensate.

Two methods have been proposed to handle the extra shift. The first method shifts the output by an addition bit only if the MSB is zero. This is accomplished by performing the two shifts in parallel, and then use a multiplexer (mux) to select the result according to the error detection bit. Note that the used multiplexer comes in the critical path.

The second method, which is more common, employs a concurrent error detection mechanism. The final shifter is divided into a separate *coarse-* and *fine-grained shifter* (CGS and FGS). The CGS shifts by a multiple of four; the FGS shifts the input operand by k, where k varies from 0 to 4. The output of the I-LZA is sent to both shifters. The CGS shifts the addenda according to the MSBs of the I-LZA. The FGS, meanwhile, computes the exact shift count using the two *least significant bits* (LSBs) of the I-LZA and the error signal and shifts the output of the CGS appropriately.

In the I-LZA architecture, the delay of the error detection circuit is non-critical, as it is only required by the FGS; the FGS, however, is more complex than the normal two stage shifter, because the shift count can vary from 0 to 4, which requires three bits to represent instead of two. Generic E- and I-LZA architectures are shown in Fig. 2 (a) and (b).

This paper contributes a novel *Hybrid LZA* (H-LZA) architecture that eliminates the shortcomings of prior E- and I-LZAs. The H-LZA is based on several key theoretical points, enumerated here, which are described in the following sections of this paper:

- The MSBs of an LZA can be computed significantly faster using the E-LZA. We show that a few MSBs of the LZA can be computed before the addenda is ready.

- Current error detection methods are slow because the error detector is not on the critical path as shown in Fig. 2 (b). In the proposed H-LZA architecture, the error detector becomes critical and a faster mechanism is needed. A novel error detection architecture is presented that is significantly simpler than current methods and reduces the delay of error detection below that of an adder.

- A small adder can add the output of an I-LZA to the result of the error detection circuit to compute the LSBs of the LZA. The delays of both components are reduced and the bitwidth of this adder is small; therefore, the delay of computing the LSBs is not significant. These bits are non-critical as they control the later stages of shifter.

Based on these contributions, a novel H-LZA architecture, shown in Fig. 2 (c), has been implemented. The MSBs are computed using an E-LZA and the LSBs are computed using an I-LZA in conjunction with an error detector. A two-stage normal shifter replaces the hybrid CGS/FGS and shifts the addition of the inputs by the LZA output. Since the MSBs of the LZA arrive before the addenda is ready, they are non-critical; likewise, the LSBs are non-critical because they are required only in the second stage of the shifter.

As a secondary contribution, we present two theoretical lower bounds on the delay of the LZA and floating point addition; we show that the critical path delay of our H-LZA and floating point addition is very close to these bounds.

The rest of the paper is organised as follows: Section 2 summaries related work on LZA design. The following section presents lower bounds on the delays of LZA. Section 4 presents an overview of the H-LZA and the requirements that must be fulfilled in order to achieve the theoretical lower bounds. Sections 5 and 6 discuss optimisations to fulfil these requirements for E- and I-LZAs respectively. Finally, Section 7 describes the algorithm implemented by the H-LZA. Section 8 presents experimental results and Section 9 concludes the paper.

2. STATE OF THE ART

A large amount of literature exit on leading zero anticipator. A discussion of various LZAs can be found in the work of Schmookler *et al.* [10]. We have already discussed that the leading zero anticipators can be classified into two categories: exact and inexact LZA. Although Most of the work has been done on inexact LZAs, a few exact LZAs has also been proposed. The LZAs presented by Ng [7], and Inoue [4] fall in this category. Most of the exact LZAs are slow with the exception of LZA proposed by Gerwig and Kroener [2]. We use this LZA as a starting point for the exact LZA, and further improve it using Boolean properties.

Kershaw *et al.* [5, 3] introduced the first I-LZAs. They recognise that to determine whether any *specific* bit can be the first leading bit, it suffices to examine only the bit at that position and its neighbour bits; this is accomplished using a constant depth circuit that examines the input integers from left-to-right and ignores the carry coming from the right.

Knowles [6] extended this idea to compute the exact expression of the indicator function for each bit position, which is then used to detect the position of the first leading digit. The first digit from the left at which the indicator function is set is the position of the leading digit with a possible error of one bit. The same expression also appears in a paper by Bruguera [1]. The position of the leading

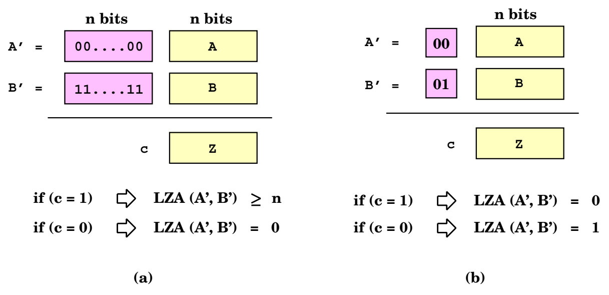
n bits
n bits
A' =
00....00
A
B' =
11....11
B
c
Z
if (c = 1) ⇨ LZA (A', B') ≥ n
if (c = 0) ⇨ LZA (A', B') = 0
(a)
n bits
A' =
00
A
B' =
01
B
c
Z
if (c = 1) ⇨ LZA (A', B') = 0
if (c = 0) ⇨ LZA (A', B') = 1
(b)

Figure 3: The carry output of an n-bit addition can be computed using (a) a $2n$-bit LZA, and (b) an $n+2$ bit LZA.

digit can then be found using a faster LZD in lieu of an LZA, such as the one described by Oklobzija [8].

The preceding techniques work for both signed and unsigned floating point addition and subtraction. If the final result is guaranteed to be positive then the circuit can be simplified as described by Suzuki *et al.* [11]; our work uses this inexact LZA.

To correct the one bit error, Suzuki *et al.* [11] perform an extra stage that shifts the final output by one bit if the MSB is not one. Other approaches use a concurrent error detection circuit to avoid this extra stage. The exact expression for the error detector was described by Kershaw *et al.* [5, 3], which was simplified by Quach *et al.* [9]. Our work further simplifies this expression under the assumption that the final result is positive; a similar expression can be derived for the negative case.

Although the use of concurrent error detection removes the extra stage, it adds a layer of complexity to the barrel shifter, which is decomposed into a CGS and an FGS; our approach, in contrast, uses normal shifters at both stages.

3. LOWER BOUNDS ON THE LZA DELAY

This section presents two lower bounds on the critical path delay of an E-LZA. These bounds will be used in the following sections to estimate whether or not the H-LZA presented in this paper is close to optimal. The following two theorems indicate that the delay of an n-bit LZA cannot be reduced significantly below the delay of an n-bit adder.

THEOREM 1. *The carry output of an n-bit addition can be computed using a $2n$-bit LZA, where n is a power of 2.*

PROOF. Assume that we want to compute the carry output of the addition of two n-bit integers A and B. We introduce two integers A' and B', where A' is formed by putting n zeros before A and B' is formed by putting n ones before B as shown in Fig. 3 (a). If $A+B$ produces a carry, then it will propagate to the carry-output of $A'+B'$ as well. In this case, the n MSBs of $A'+B'$ are zero, i.e.: $LZA(A', B') \geq n$.

On the other hand, if $A+B$ does not produce a carry, then $A'+B'$ will produce no leading zeros, i.e.: $LZA(A', B') = 0$. Since n is a power of 2, it follows that the carry output of $A+B$ is equal to the MSB of the LZA of A' and B' □

Theorem 1 proves that the delay of a $2n$-bit LZA cannot be smaller than that of the fastest carry computation in an n-bit adder. Theorem 2 tightens this bound.

THEOREM 2. *The carry output of an n-bit addition can be computed using an $(n+2)$-bit LZA.*

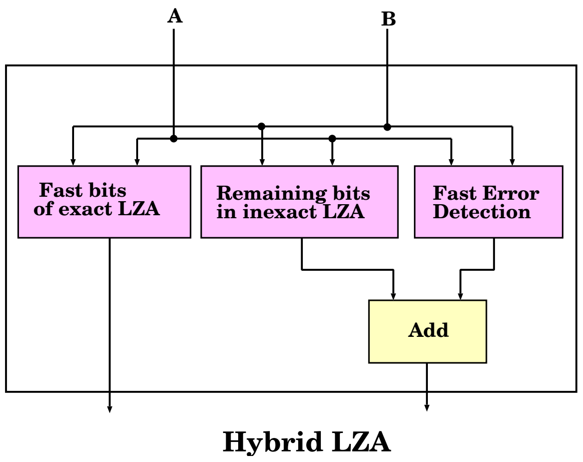

Figure 4: A faster hybrid LZA architecture that uses both E- and I-LZA.

PROOF. Similar to the proof of Theorem 1, we introduce two new integers A' and B', where A' is formed by concatenating 00 to A and B' is formed by concatenating 01 to B, as shown in Fig. 3 (b). It is straightforward to see that the LZA of A' and B' is 0 or 1 depending on the carry output of $A+B$; specifically, the carry output of $A+B$ is equal to the negation of the LSB of the LZA of A' and B'. □

In Fig. 2 (a) there is no benefit from reducing the delay of an LZA below the delay of an adder because the final shifter uses the results of both; reducing the delay of the LZA removes it from the critical path, which is then dominated by the adder and shifter; further reducing the delay of the LZA will not reduce the critical path of the circuit.

4. MAIN IDEA AND ITS REQUIREMENTS

This section introduces the fundamental ideas underlying the H-LZA. The critical path of a floating point adder is through the shifter. To ensure that a shifter is $normal$, each stage should be implemented using a series of $2:1$ muxes.

The output of an LZA is an unsigned integer in the range 0 to $n-1$; to ensure that each stage of the final shifter is normal, the bits of the E-LZA should be used to control the multiplexers at each stage of the shifter. This is because the use of a $2:1$ multiplexer at each stage ensures that each control signal is a binary digit. If n is a power of 2, then in binary system, each number from 0 to $n-1$ has a unique representation. The control signals in the shifter stages, therefore, should correspond to the output bits of the LZA.

The implication is that an I-LZA cannot be used in the shifter without introducing complications; however, the I-LZA *can* effectively compute some of the bits of the E-LZA.

The underlying principle of the H-LZA is to compute its output bits using a hybrid structure that employs both E- and I-LZAs. The first few stages of the shifter use the bits from the E-LZA. The novelty of our approach is a new method to reduce the delays of some of the output bits of the E-LZA down to the delay of an adder as described in Section 5.

The only way to compute the E-LZA from I-LZA is to add the output of I-LZA with output of error detection; the remaining bits are computed in this fashion. Since these bits are used in the late stages of the shifter, and they can tolerate increased delay, as long as they do not constraint the critical path. This can be achieved

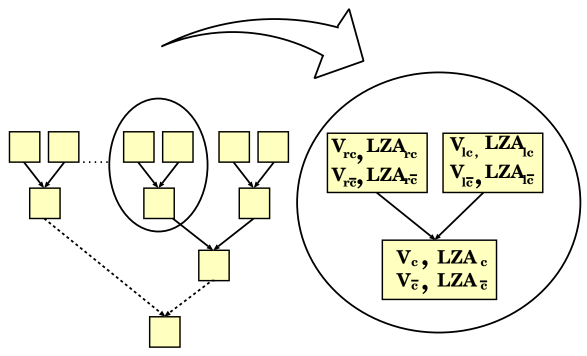
V_{rc}, LZA_{rc}
$V_{r\overline{c}}$, $LZA_{r\overline{c}}$
V_{lc}, LZA_{lc}
$V_{l\overline{c}}$, $LZA_{l\overline{c}}$
V_c, LZA_c
$V_{\overline{c}}$, $LZA_{\overline{c}}$

Figure 5: An implementation of an E-LZA organised in carry-select fashion.

by reducing the delay of both the I-LZA and the error detection circuit. Section 6 describes a new method to compute the error detection signal whose critical path is lower than that of an adder.

Fig. 4 illustrates the H-LZA, in which some bits are computed using an E-LZA, while others are computed using an I-LZA in conjunction with an error detector.

5. EXACT LZA AND ITS IMPROVEMENT

This sections describes the design of an E-LZA with modifications to reduce the delay of the computation of some of its output bits. Our baseline E-LZA is similar to the one suggested by Gerwig and Kroener [2], which is similar in principle to a *Carry Select Adder*.

First, the input integers are divided into blocks of 2 bits; for each block, the following values are computed:

- LZA_c: the LZA of the block, assuming that there is an incoming carry bit.
- v_c: A single bit that is true if and only if all output bits at the bit positions corresponding to this block are zero, assuming that there is an incoming carry bit.
- $LZA_{\overline{c}}$: the LZA of the block, assuming that there is no incoming carry bit.
- $v_{\overline{c}}$: a single bit which is true if and only if at all output bits at the bit positions corresponding to this block are zero, assuming that there is no incoming carry bit.

Next, we merge two adjacent blocks into a single block and compute the corresponding values for this block. This process repeats until only one block remains; then the value of $LZA_{\overline{c}}$ is returned as the output of LZA. This process is illustrated in Fig. 5. The process of merging $block_l$ (the left block) and $block_r$ (the right block) is performed as follows:

$$v_c = v_{lc}(\overline{K_l} v_{rc} + K_l v_{r\overline{c}}),$$
$$LZA_c = \langle y_c X_c \rangle, \text{ i.e., concatenation of } y_c \text{ and } X_c \text{ where}$$
$$X_c = \overline{K_l}(\overline{v_{rc}} LZA_{rc} + v_{rc} LZA_{lc}) +$$
$$K_l(\overline{v_{r\overline{c}}} LZA_{r\overline{c}} + v_{r\overline{c}} LZA_{lc}),$$
$$y_c = \overline{K_l} v_{rc} + K_l v_{r\overline{c}}.$$

In the above equations, which are defined recursively, '+' denotes the logical OR operation. The symbols P, G, and K denote the propagate, generate, and kill signals for the corresponding blocks.

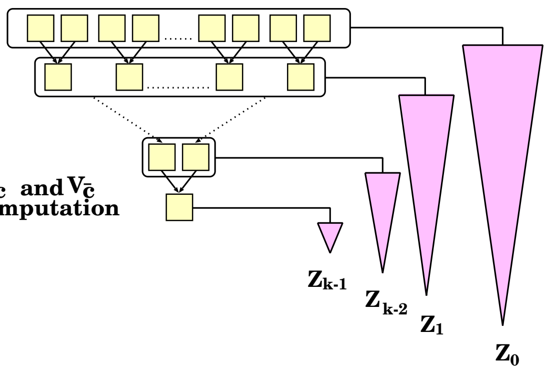

Figure 6: Dependency of the LZA output bits on the v_c and $v_{\overline{c}}$ values computed at various stages. The Z_i's denote the different output bits of the LZA.

The above recurrences compute v_c and LZA_c; $v_{\overline{c}}$ and $LZA_{\overline{c}}$ can be computed similarly.

Only v_c and $v_{\overline{c}}$ values are used to compute y_c and $y_{\overline{c}}$; thus, the second LSB of the LZA can be computed using propagate, generate, v_c, and $v_{\overline{c}}$ values computed in the first stage. In general, the r^{th} LSB of the LZA can be computed using the propagate, generate, v_c, and $v_{\overline{c}}$ bits computed in the $(r-1)$-th stage, as shown in Fig. 6.

If we can find a faster way to compute v_c and $v_{\overline{c}}$, then the delays of the MSBs of the LZA could improve significantly; the following two theorems describe how to accomplish this.

THEOREM 3. *The v_c of a block is the same as the propagate of the block, i.e., $v_c = P$.*

PROOF. Let R and S denote the portion of the input integers inside the block. If k is the width of the block, then the v_c value for the block is true if and only if the k LSBs of $R + S + 1$ are zero. In other words,

$$R + S + 1 \equiv 0(\text{mod } 2^k), \text{ i.e.,}$$
$$R + S \equiv -1(\text{mod } 2^k), \text{ i.e.,}$$
$$R + S = 2^k - 1 = 11 \cdots 1.$$

Note that $R + S = 2^k - 1$ if and only if the sum of R and S at each bit position is 1, i.e., the propagate signal is true at each bit position of the block, or equivalently, the propagate signal P of the whole block is true. □

Let p_{begin} be the propagate signal of the rightmost bit of the block.

THEOREM 4. *The expression of $v_{\overline{c}}$ of a block can be given like this: $v_{\overline{c}} = \overline{p_{begin}} \prod_i (p_i \oplus k_{i-1})$.*

PROOF. $v_{\overline{c}}$ of a block is true if and only if the block has the form $[p^*g]k^*$. Here p^* means a string of p's of any length (including zero), and $[x]$ denotes a string which can be either the same as x, or can be an empty string.

At the leftmost position of the block, any of the three cases, propagate, generate, or kill, may occur. A kill or a generate signal at the leftmost position is followed by all kill signals at every remaining bit position in the block. A propagate signal at the leftmost position is followed by a chain of propagate signals at higher bit positions, followed by a generate and then a chain of kill signals.

Likewise, the block has the form $[p^*g]k^*$ if and only if it satisfies the following three conditions:

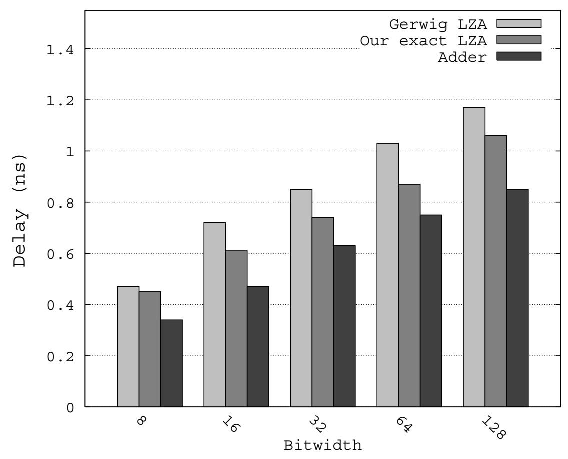

Figure 7: A comparison of the delays of the E-LZA of Gerwig and Kroener [2] and the E-LZA described here.

- a propagate signal is not be followed by a kill signal,
- any signal other than the propagate is followed by a kill signal, and
- p_{begin} is false.

It follows that:

$$v_{\overline{c}} = \overline{p_{begin}} \prod_i ((p_i + k_{i-1})(\overline{p_i} + \overline{k_{i-1}})), \text{ i.e.,}$$

$$v_{\overline{c}} = \overline{p_{begin}} \prod_i (p_i \oplus k_{i-1}).$$

□

As a consequence of Theorem 3 and 4, v_c and $v_{\overline{c}}$ can be computed for any block by computing the AND of a set of values whose depth is constant. Since these computations are extremely fast, compared to the adder, the delays of the few MSBs of the LZA is also smaller than that of the adder; in our experiments this property holds for the two or three MSBs. It also improves the overall delay of the E-LZA as shown in Fig. 7. This approach improves the delay by 10–15% compared to the E-LZA described by Gerwig and Kroener [2]. The overall delay of the E-LZA is also close to the delay of an adder of the same bitwidth.

6. INEXACT LZA AND FASTER ERROR DETECTION

This section describes the implementation of an I-LZA and error detection circuit. Leading zeros never occur in the addition of two normalised integers of same sign. On the other hand, The MSB propagate signal is true in the addition of two integers of opposite signs.

If the result is positive, then there is at least one leading zero: the sign bit. For this case, Suzuki *et al.* [11] and Quach *et al.* [9] proposed a fast I-LZA architecture based on the following property: when two normalised integers are added, leading zeros will occur if and only if the whole block has the form $p^i g k^j *$. In other words, if we look at the block from left (i.e., from MSBs) it should have a chain of propagates of length i, followed by a generate, and a chain of kills of length j, which can be followed by any string (as denoted by $*$). The number of leading zeros in this case is either $i + j$ or

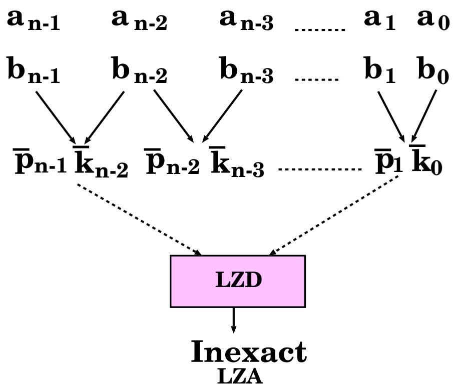
a_{n-1} a_{n-2} a_{n-3} .... a_1 a_0
b_{n-1} b_{n-2} b_{n-3} .... b_1 b_0
$\overline{p}_{n-1}\overline{k}_{n-2}$ $\overline{p}_{n-2}\overline{k}_{n-3}$ .... $\overline{p}_1\overline{k}_0$
LZD
Inexact LZA

Figure 8: Design of an I-LZA proposed by Suzuki *et al.* [11].

$i + j + 1$, where i and j are the largest values for which this form holds. In other words, an I-LZA must find the largest suffix (i.e., a block starting from MSB) of the input integers that has the form p^*gk^*. Under the assumption that the result is positive, the size of the largest such suffix can be computed as shown in Fig. 8.

First, a new integer, X, is generated as follows:

$$X_0 = p_0,$$
$$X_i = \overline{p_i}\overline{k_{i-1}} \quad (\text{for } i \geq 1).$$

The number of leading zeros in X is one less than the length of the largest suffix having the aforementioned form. Since the critical path delay of an LZD is small, the I-LZA can be computed quickly. More specifically the delay of an I-LZA differ only by a constant from the delay of an LZD of the same bitwidth.

6.1 Error Detection

Quach *et al.* [9] and Kershaw *et al.* [5, 3] observed that an error will occur in an I-LZA only if there is an incoming carry from right at the last bit position of the longest suffix of the form p^*gk^*. Let e be a Boolean variable that is true in the presence of an error; then e will be true when a suffix of the block is of the form $p^*gk^*p^*g$. Our method for error detection is based on the following theorem.

THEOREM 5. *An string made of $p, g,$ and k, and starting with p will have a suffix of the form $p^*gk^*p^*g$ (the pattern is written in the form starting from MSB) if and only if it has a suffix S where the following two conditions are true:*

- *S has at least two g's.*
- *The expression $(\overline{p_i} + \overline{k_{i-1}})$ is true at each bit position except the last one.*

PROOF. If a suffix R is of the form $p^*gk^*p^*g$, then R is trivially a suffix that satisfies the two conditions.

Now, consider a suffix S for which the two conditions are true; we will prove the existence of a suffix that has the desired form. Since S contains at least two g, there must be a suffix of S that has exactly two g's as well; let S' be the smallest such suffix of S, from which we can infer that S' ends with a g. In S', the expression $(\overline{p_i} + \overline{k_{i-1}})$ holds at each bit position except for the last; therefore, each contiguous sequence of p's must terminate with a g.

Since S' starts with p, it must terminate by a g, i.e., S' can be written as p^*gYg, where Y contains no g's. If Y begins with a p, then Y has the form p^*, since the chain of p's can only terminate with a g. Therefore S' has the form p^*gp^*g, which satisfies the desired form (note that k^* includes the empty string).

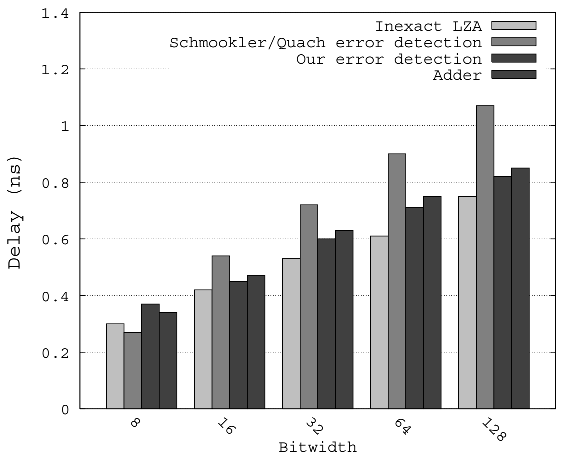

Figure 9: Comparison of the delays of the I-LZA and error detection, described here and by Schmookler *et al.* [10] and Quach *et al.* [9].

The other possibility is that the first character in Y is a k; then Y has the form k^*p^* because Y cannot contain a g by construction; hence, the chain of k's must terminate with p; however, after the first p, the chain of p's must continue until the g at the end of Y; therefore, S' has the desired form.

In both cases, $R = S'$. $\square$

From Theorem 5 it follows that a simpler expression for each error bit, e_i, can be computed for the ith suffix; each of these error bits can be ORed to find the overall expression for e:

$$\begin{aligned} e_i &= g_i(g_n + g_{n-1} + \cdots g_{i+1})(\overline{p_{n-1}} + \overline{k_{n-2}}) \cdots (\overline{p_{i+1}} + \overline{k_i}), \\ e_i &= g_i(g_n + g_{n-1} + \cdots g_{i+1}) \\ &\quad (\overline{p_{n-1}} + \overline{k_{n-2}}) \cdots (\overline{p_{i+2}} + \overline{k_{i+1}}), \\ e &= e_0 + e_1 + \cdots + e_{n-1}. \end{aligned}$$

Note that $\mathsf{NOT}(e)$ is added to the output of I-LZA in order to get the correct value of LZA.

Each of the e_i's can be computed using a circuit similar to a Manchester carry; using this method, the critical path delay was improved by 20% compared to the prior error detection mechanisms of Schmookler *et al.* [10] and Quach *et al.* [9], as illustrated in Fig. 9. Moreover, the delay of our error detection method was reduced below the delay of an adder.

7. H-LZA CONSTRUCTION ALGORITHM

The algorithm to construct a fast H-LZA is as follows. First, we compute the delays of each output bit of the E-LZA from Section 5, and the delay of the addition of the I-LZA and error detection signal from Section 6 by synthesising the corresponding circuit. In the E-LZA, the delays increase in order from the MSB to the LSB; in the addition of I-LZA and error detection circuit, the delays are in decreasing order from the MSB to the LSB.

It is straightforward to find some value j such that it is advantageous to compute the MSBs up to and including j using the E-LZA and the remaining bits using the I-LZA with error detection. In our experiments we observed that typically two to three MSBs are advantageous to compute using E-LZA, and for the rest of them I-LZA based method is faster. Using this value of j we construct the hybrid LZA which is used for normalisation. In the normalisation process shifter starts from MSBs to LSBs.

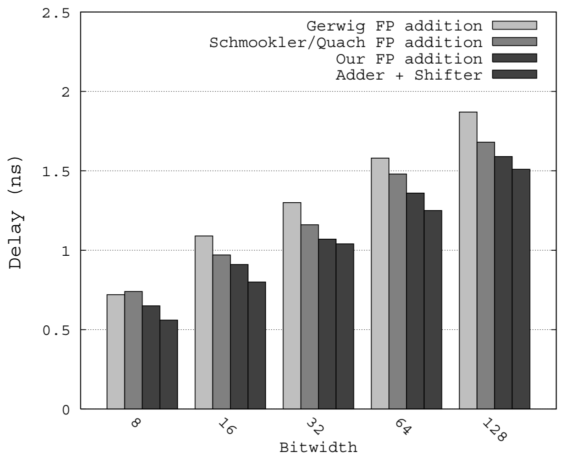

Figure 10: Comparisons of the delays of a FP adder using Gerwig and Kroener's LZA [2], Suzuki *et al.*'s LZA [11] with Quach *et al.*'s error detection [9], our proposed H-LZA, and the delay of an adder + shifter, which represents the lower bound.

8. EXPERIMENTAL RESULTS

We wrote a C++ program that implements the H-LZA construction algorithm described in the preceding section; the output is VHDL. A second C++ program then constructs a FP adder using the H-LZA. These circuits are synthesised using a standard cell library for UMC $0.18\mu m$ CMOS technology.

Fig. 10 compares the delay of a FP adder built using the H-LZA described in this paper with similar adders built from the E-LZA of Gerwig *et al.* [2] and a normal shifter, the I-LZA of Suzuki *et al.* [11] with Quach *et al.*'s error detector [9] and a hybrid CGS/FGS shifter, and an adder-shifter combination of the same bitwidth that represents the lower bound of a floating point addition.

For bitwidths of 16 and above, the E-LZA is the slowest, followed by the I-LZA, our H-LZA, and then the adder-shifter; the E-LZA is faster than the I-LZA for 8-bit addition. This is because for smaller bitwidths the delay of complex FGS is not negligible and makes the I-LZA slower than E-LZA. Among the three FP adders, our H-LZA is uniformly the fastest; moreover, it is just marginally slower compared to the lower bound, indicating that our H-LZA is not on the critical path of the circuit.

Fig. 11 compares the area of the four implementations. The adder-shifter, which is a lower bound, is uniformly the smallest, as it does not contain any LZA. For 8-bit FP addition, the E-LZA is the largest, followed by our H-LZA and then the I-LZA; for 16-bit FP addition, the E- and I-LZAs have comparable size, while our H-LZA is smaller; for 32-, 64-, and 128-bit FP addition, the E-LZA is the largest, followed by the I-LZA, and then ours H-LZA. The difference between the I-LZA and our H-LZA is marginal for 32- and 64-bit FP addition, but becomes more pronounced when the bitwidth increases to 128. This savings in area is primarily due to the simplified expression for error detection and the use of the improved E-LZA for the MSBs.

9. CONCLUSION

Previous E-LZA architectures suffer from unnecessarily complex logic stages, while previous I-LZA architectures suffer from the use of a hybrid CGS/FGS shifter. To address these concerns, we have presented a novel H-LZA that combines exact and inexact

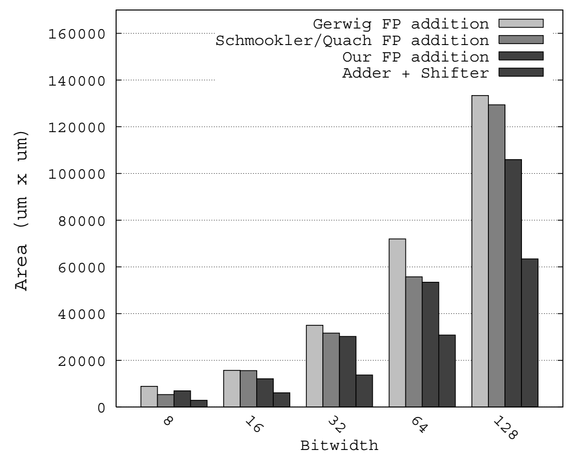

Figure 11: Comparisons of the areas of a FP adder using Gerwig and Kroener's LZA [2], Suzuki *et al.*'s LZA [11] with Quach *et al.*'s error detection [9], our proposed H-LZA, and the area of an adder + shifter.

LZAs for different bits. This new LZA no longer remains on the critical path of the normalisation process in a floating-point adder. Our experiments show that the H-LZA reduces the delay of floating point addition by 7–10% compared to prior E- and I-LZA architectures, with either minimal impact or a reduction in area. Note that we are improving only LZA, while floating point addition consist of adder, LZA and shifter, hence, an improvement of 7–10% in delay is significant. We have also established theoretical lower bounds on the delays of an LZA and of the floating point addition. The delay of our proposed H-LZA is very close to that of the optimal lower bounds.

10. REFERENCES

[1] J. Bruguera and T. Lang. Leading-one prediction with concurrent position correction. In *International Conference on Computers*, pages 298–305, Oct. 1999.

[2] G. Gerwig and M. Kroener. Floating point unit in standard cell design with 116 bit wide dataflow. In *IEEE Symposium on Computer Arithmetic*, pages 266–73, 1999.

[3] 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, and L. Tran. A 32-bit vlsi digital signal processor. In *IEEE Journal of Solid State Circuits*, pages 998–1004, Oct. 1985.

[4] G. Inoue. Leading one anticipator and floating point addition/subtraction apparatus. In *US Patent US5343413*, Aug. 1994.

[5] R. Kershaw, L. Bays, R. Freyman, J. Klinikowski, C. Miller, K. Mondal, H. Moscovitz, Y. Rotblum, W. Stocker, and L. Tran. A programmable digital signal processor with 32-bit floating point arithmetic. In *IEEE Solid State Circuits Conference, Digest of Papers*, pages 92–93, 1985.

[6] S. Knowles. Arithmetic processor design for the t9000 transputer. In *SPIE*, pages 230–243, 1991.

[7] K. Ng. Exact leading zero predictor for a floating point adder. In *US Patent US5204825*, Feb. 1993.

[8] V. G. Oklobdzija. An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, VLSI-2(1), Mar. 1994.

[9] N. Quach and M. Flynn. Leading one prediction—implementation, generalization, and application. In *Technical report CSL-TR-91-463, Stanford University*, Mar. 1991.

[10] M. Schmookler and K. Nowka. Leading zero anticipation and detection—a comparison of methods. In *IEEE Symp. Computer Arithmetic*, pages 7–12, June 2001.

[11] H. Suzuki, H. Morinaka, H. Makino, Y. Nakase, K. Mashiko, and T. Sumi. Leading-zero anticipatory logic for high-speed floating point addition. In *International Journal of Solid State Circuits*, pages 1157–64, Aug. 1996.