Low-Power Leading-Zero Counting and Anticipation Logic for High-Speed Floating Point Units

Giorgos Dimitrakopoulos, *Member, IEEE*, Kostas Galanopoulos, Christos Mavrokefalidis, *Student Member, IEEE*, and Dimitris Nikolos, *Member, IEEE*

***Abstract*—In this paper, a new leading-zero counter (or detector) is presented. New boolean relations for the bits of the leading-zero count are derived that allow their computation to be performed using standard carry-lookahead techniques. Using the proposed approach various design choices can be explored and different circuit topologies can be derived for the design of the leading-zero counting unit. The new circuits can be efficiently implemented either in static or in dynamic logic and require significantly less energy per operation compared to the already known architectures. The integration of the proposed leading-zero counter with the leading-zero anticipation logic is analyzed and the most efficient combination is identified. Finally, a simple yet efficient technique for handling the error of the leading-zero anticipation logic is also presented. The energy-delay behavior of the proposed circuits has been investigated using static and dynamic CMOS implementations in a 130-nm CMOS technology.**

***Index Terms*—Floating-point unit, leading-zero anticipation (LZA), leading-zero counter (LZC), microprocessor, normalization.**

I. Introduction

FLOATING-POINT units are a dominant part of modern microprocessors and both their delay and energy consumption should be aggressively optimized in order to get an overall efficient design. Several high-speed floating-point units have been recently presented in open literature [1]–[5]. One of the main operations of floating-point datapaths is normalization. During normalization the outcome of a floating-point operation is brought to its normalized form, i.e., $1.xx\ldots x$, $x \in \{0,1\}$, according to the IEEE-754 standard [6]. Normalization involves the use of a leading-zero counting (LZC), or detection unit, and a normalization shifter, while in almost all cases, leading-zero anticipation (LZA) logic is also employed to speedup the computation.

The problem of normalizing the result can be solved in two ways. The first one involves counting the number of leading zeros of the result and then shifting the result to the left according to the outcome of the LZC unit. The derived leading-zero count is also required in order to correctly update the exponent part of the result. This method is slow and is rarely preferred. The second way is to try to predict in parallel with the true operation (addition/subtraction) a pseudo result that will have almost equal number of leading zeros as the true result. In this way, both the true operation and leading-zero counting on the pseudo result can occur simultaneously. The predicted leading zero count is given to a shifter in order to normalize the true result.

The prediction of the pseudo result is performed by the LZA logic and its design has attracted a lot of interest in the past few years [7]–[11]. A clear and thorough overview of the most efficient solutions can be found in [11]. LZA logic also handles the case of leading ones when the result of the true operation is negative. Several methods have been presented that try to handle leading zeros as well as leading ones in a unified manner [8], [9], [11], [12]. Up to now, mostly due to delay reasons, distinct units are employed to handle separately leading zeros and leading ones [2], [10]. In other cases, duplicate adders and LZC units are used to compute both the negative and the positive version of the same result, e.g., $A-B$ and $B-A$, and the final outcome is selected using the actual sign of the result [4]. The predicted leading-zero count is not always exact and may differ from that of the true result by one position. This error of the LZA logic can be handled in several ways [11].

Counting the leading zeros of a word is also useful to many other cases besides floating-point datapaths. Almost all instruction sets of contemporary microprocessors include a count leading zeros (CLZ) instruction for fixed-point operands. Also, several hardware-based function evaluation algorithms require the inclusion of a LZC unit to speed up the computation; see for example [13]. The same holds for some variable-length-code decoding architectures used in data compression [14].

In this paper, the design of a new and energy-efficient leading-zero counting and anticipation logic is discussed. In Section II, we briefly describe the functionality and the implementation of existing LZC architectures. Then, in Section III, following a mathematical approach, we derive new boolean relations that describe the bits of the leading-zero count. The form of the derived equations shows that the leading-zero count can be efficiently computed using standard carry-lookahead techniques. Based on this observation, we propose two organizations for the new LZC unit that are presented in Section IV. In Section V, the integration of the proposed LZC logic to the already known LZA architectures is explored. We describe the LZA approaches used in state-of-the-art floating-point units and the techniques involved for their implementation according to the chosen LZC unit. Our goal is to clarify which part of the prediction circuit that consists of the LZA logic and the LZC unit, is more critical in terms of energy and delay for the performance of the whole

Manuscript received August 28, 2006; revised August 19, 2007. The work of G. Dimitrakopoulos was supported by the "D. Maritsas" graduate scholarship of the Research Academic Computer Technology Institute, Patras, Greece.

The authors are with the Technology and Computer Architecture Laboratory, Computer Engineering and Informatics Department, University of Patras, Patras 26500, Greece (e-mail: dimitrak@ceid.upatras.gr).

Digital Object Identifier 10.1109/TVLSI.2008.2000458

1063-8210/$25.00 © 2008 IEEE

circuit. The combination of the LZA logic with the new LZC unit leads to new and efficient designs. Additionally, we propose a simple way to handle the one bit prediction error of the LZA logic. The benefits of the new LZA error handling method are analyzed and compared to previously known techniques. The efficiency of the proposed circuits has been validated using static and dynamic CMOS implementations in a standard performance 130-nm technology. The experimental methodology followed and the derived results are presented in Section VI. Finally, conclusions are drawn in Section VII.

II. Review of LZC Architectures

Leading-zero counting is the procedure of encoding in binary representation the number of consecutive zeros that appear in a word before the first more significant bit that is equal to one. The latter is called leading digit. The opposite holds for the case of leading ones. The LZC unit assumes n bits as input $A = A_{n-1}A_{n-2}, \ldots, A_0$, where A_{n-1} is the most-significant bit (MSB), and produces the $\log_2 n$ bits of the leading-zero count Z and a flag V that denotes the all-zero case for the input A.

The first method for determining the leading-zero count of a word is based on a two-step encoding procedure [11]. At first, the position of the leading digit of the input operand is marked and the remaining bits are set to zeros (one-hot representation). For example, for the input 00110100 the position of the leading digit is determined by the codeword 00100000. To derive the one-hot representation, an intermediate S string is at first produced. The bits of S that follow the leading digit are set to one, while the other more-significant bits remain to zero. For the same input, S is equal to 00111111. The ith bit of S denoted as S_i, $0 \leq i \leq n-1$, is defined as follows:

$$S_i = A_{n-1} + A_{n-2} + \cdots + A_{i+1} + A_i \tag{1}$$

where $+$ denotes the logical OR operation. Each bit S_i reveals the existence of at least one bit equal to 1 between the MSB and the ith bit position. The one-hot representation of the leading digit (L word) is determined by detecting the case (0,1) for two consecutive bits of S. Hence

$$L_i = \overline{S}_{i+1} \cdot S_i = \overline{S}_{i+1} \cdot A_i, \quad \text{for} \quad 0 \leq i \leq n-2 \tag{2}$$

and $L_{n-1} = S_{n-1}$, where $\cdot$ and $\overline{x}$ denote the logical AND and complement operations, respectively. The circuit that computes the L word is called a priority encoder [15]. The L word is then given to an encoder that translates the number of leading zeros to its weighted binary representation [11]. For the case of an 8-bit input operand the number of leading zeros, Z bits, are given by (3)–(5)

$$Z_2 = L_3 + L_2 + L_1 + L_0 \tag{3}$$

$$Z_1 = L_5 + L_4 + L_1 + L_0 \tag{4}$$

$$Z_0 = L_6 + L_4 + L_2 + L_0. \tag{5}$$

The all-zero flag V is set to $\overline{S}_0$ showing that no bit equal to one, exists in A. Up to now, this approach is mostly preferred in dynamic CMOS floating-point-unit implementations.

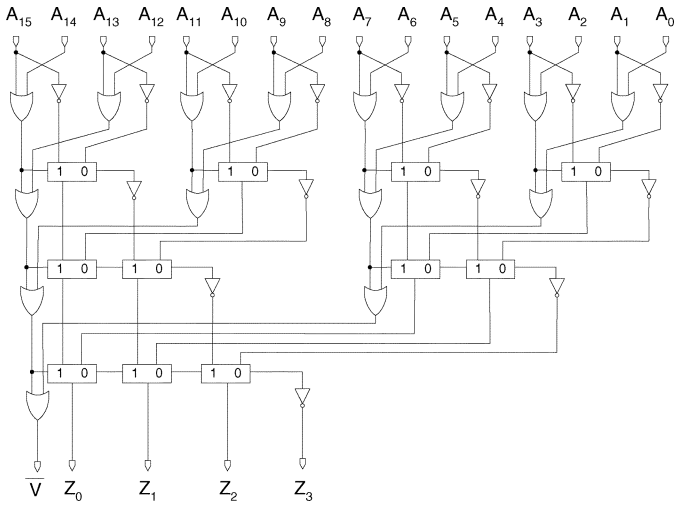


Fig. 1. 16-bit LZC unit following the architecture presented in [16].

The leading-zero count is computed in few logic stages by employing wide dynamic OR gates for the computation of both the S and the Z bits. In the rest of this paper, we will refer to this LZC architecture as the encoder-based LZC unit.

The second method for computing the leading-zero count is based on an algorithmic approach and it was introduced by Oklobdzija in [16]. As it was shown in [17], the same approach has also been used in [18]. At first, the input is partitioned in $n/2$ two-bit groups of adjacent bits. For each group, a 2-bit leading-zero count is generated. The most significant of the two bits also acts as an all-zero indicator for the bits of the group. At the next level, neighbor groups are combined and either the leading-zero count of the left or the right group is selected using a set of multiplexers. The selection is performed based on the value of the most-significant bit of the left group. Additionally, the two all-zero indicators are combined and a new all-zero flag is produced, which indicates when the bits of the new double size group are all equal to zero. This procedure continues recursively for $\log_2 n$ levels. The circuit is modular and is the fastest solution up-to-now for static CMOS implementations. The implementation of the 16-bit LZC unit is shown in Fig. 1.

The last method for computing the leading-zero count was presented by Bruguera and Lang in [19]. The algorithm recursively calculates the ith bit of the leading-zero count based on the precomputed more significant bit Z_{i+1}. Assume, for example, the case of an 8-bit input operand A. At first, the OR of the $n/2$ high-order bits is computed. This signal will be equal to zero when $A_7A_6A_5A_4$ are all zero. Hence, this signal inverted indicates that there are at least four leading zeros in A, and thus $Z_2 = 1$. At the next step, the algorithm uses the value of Z_2 to determine the next bit Z_1 of the leading-zero count. In the case that $Z_2 = 1$, it means that we should search for more leading-zeros to the bits $A_3A_2A_1A_0$. In the opposite case, $Z_2 = 0$, less than four leading zeros exist and they are placed on the upper part of the word $A_7A_6A_5A_4$. Therefore, in the following, the OR of the bits A_3, A_2, and A_7, A_6 is computed, respectively, to determine whether Z_1 should be asserted. The value of Z_1 is computed via a multiplexer that selects the OR of either A_3, A_2 or A_7, A_6 according to the value of the more

significant bit Z_2. Since the OR of the smaller group of bits is computed earlier than Z_2, the computation of bits Z_2 and Z_1 is overlapped in time. In the next levels, the algorithm continues using the same principle. The four possible values of Z_2Z_1 determine, using a tree of multiplexers, if an extra leading zero exists in bits A_7, A_5, A_3, or A_1.

According to the design of [19], the more significant bits of the leading-zero count are computed earlier than the corresponding less significant bits. This approach was followed since the more significant bits are used in the first stages of the shifter that perform the coarse normalizing steps. The less significant bits can be delayed since they are not used until the last shifting stages of the normalization shifter. This technique may be beneficial in some cases, however it cannot be applied when the leading-zero counter and the normalization shifter belong to different pipeline stages. Also, producing the bits of the leading-zero count with a certain delay difference between them, increases the delay of the exponent update logic that subtracts the leading-zero count from the precomputed exponent value. The adoption of this technique depends on the specific design choices made for the complete floating-point unit such as the number of pipeline stages and the clock period. Also, delaying the computation of the bits of the leading-zero count can be also applied to all other LZC circuits by properly sizing the gates of the circuit. The extra time slack provided to the less significant bits can be used for reducing the power dissipation of the circuit.

The last two methods assume that, when the input operand is equal to zero, the V flag is asserted and the Z bits are treated as don't care values. Since $A = 0$ any redundant normalization shift does not change the result. This approach is also followed by the proposed method. The implementation details will be clarified in the following sections.

III. Proposed LZC Algorithm

In this section, the proposed LZC unit will be presented. Following a mathematical approach, we simplify the boolean relations that describe the Z bits of the leading-zero count. The proposed method will be presented using an example of an 8-bit LZC unit. Using (2) in (3)–(5), we get

$$Z_2 = \overline{S}_4 \cdot S_3 + \overline{S}_3 \cdot S_2 + \overline{S}_2 \cdot S_1 + \overline{S}_1 \cdot S_0 \quad (6)$$
$$Z_1 = \overline{S}_6 \cdot S_5 + \overline{S}_5 \cdot S_4 + \overline{S}_2 \cdot S_1 + \overline{S}_1 \cdot S_0 \quad (7)$$
$$Z_0 = \overline{S}_7 \cdot S_6 + \overline{S}_5 \cdot S_4 + \overline{S}_3 \cdot S_2 + \overline{S}_1 \cdot S_0. \quad (8)$$

Since the S string is monotonically increasing, i.e., it has the form $00\ldots01\ldots11$, then a pair of bits (S_i, S_j) with $i > j$ can never take the value (1,0). This fact leads to two significant properties of the bits of S. For $i > j$, we have $S_i \cdot S_j = S_i$ and $S_i + S_j = S_j$. Using these properties, the equations that describe the Z bits can be further simplified. For example, the term $\overline{S}_4 \cdot S_3 + \overline{S}_3 \cdot S_2$ of (6) can be written as $\overline{S}_4 \cdot \left(S_3 + \overline{S}_3 \cdot S_2\right)$ since $\overline{S}_4 \cdot \overline{S}_3 = \overline{S}_3$. Then, the term in the parentheses is reduced to $S_3 + S_2$ that is equal to S_2, due to the second property of the S bits. Such modifications can be recursively applied to the rest bits of the leading-zero count, leading to the following relations:

$$Z_2 = \overline{S}_4 \cdot S_0 \quad (9)$$
$$Z_1 = \overline{S}_6 \cdot S_4 + \overline{S}_2 \cdot S_0 \quad (10)$$
$$Z_0 = \overline{S}_7 \cdot S_6 + \overline{S}_5 \cdot S_4 + \overline{S}_3 \cdot S_2 + \overline{S}_1 \cdot S_0. \quad (11)$$

These relations have also been presented in [11] without providing any proof for their derivation or any further optimization. According to (9)–(11), when the input is equal to zero, the Z bits are also set to zero, indicating that no normalization is required. However, as long as the V flag is asserted, we can map the Z bits to any other value. Therefore, we chose to set each bit Z_i to 1 when $V = \overline{S}_0 = 1$. In this way, (9)–(11) can be written as follows:

$$Z_2 = \overline{S}_4 \cdot S_0 + \overline{S}_0 \quad (12)$$
$$Z_1 = \overline{S}_6 \cdot S_4 + \overline{S}_2 \cdot S_0 + \overline{S}_0 \quad (13)$$
$$Z_0 = \overline{S}_7 \cdot S_6 + \overline{S}_5 \cdot S_4 + \overline{S}_3 \cdot S_2 + \overline{S}_1 \cdot S_0 + \overline{S}_0. \quad (14)$$

We can replace redundant terms of the form $\overline{S}_i \cdot S_0 + \overline{S}_0$, $i > 0$ with their equivalent terms $\overline{S}_i + \overline{S}_0$. Based on the first property of the S bits, these terms are further simplified to $\overline{S}_i$. After applying such simplifications, the Z bits are given by (15)–(17)

$$Z_2 = \overline{S}_4 \quad (15)$$
$$Z_1 = \overline{S}_6 \cdot S_4 + \overline{S}_2 \quad (16)$$
$$Z_0 = \overline{S}_7 \cdot S_6 + \overline{S}_5 \cdot S_4 + \overline{S}_3 \cdot S_2 + \overline{S}_1. \quad (17)$$

We are interested in computing the bits of the leading-zero count directly from the input operand A. Therefore, we express each bit S_i of (15)–(17) as a function of the input bits A_i following (1). Inverting the Z bits and performing some simple boolean algebra manipulations, we derive the final equations that describe the bits of the leading-zero count in the case of an 8-bit input operand

$$\overline{Z}_2 = A_7 + A_6 + A_5 + A_4 \quad (18)$$
$$\overline{Z}_1 = A_7 + A_6 + \overline{A}_5 \cdot \overline{A}_4 \cdot (A_3 + A_2) \quad (19)$$
$$\overline{Z}_0 = A_7 + \overline{A}_6 \cdot A_5 + \overline{A}_6 \cdot \overline{A}_4 \cdot A_3 + \overline{A}_6 \cdot \overline{A}_4 \cdot \overline{A}_2 \cdot A_1. \quad (20)$$

The Z bits are all equal to one when $A = 0$. This is the reason why the least significant bit A_0 of the input does not participate in any relation concerning the Z bits of the leading-zero count. A_0 is only used, via S_0, for the computation of the V flag.

The least-significant bit Z_0 determines whether the leading-zero count is an odd or even number. We would like to exploit this simple property. Therefore, a new single output operator $F()$ is defined that has the same form as the equation describing bit Z_0

$$\begin{aligned} F(X_{n-1}, X_{n-2}, \ldots, X_1, X_0) &= X_{n-1} + \overline{X}_{n-2} \cdot X_{n-3} + \\ &+ \overline{X}_{n-2} \cdot \overline{X}_{n-4} \cdot X_{n-5} \\ &+ \cdots + \overline{X}_{n-2} \cdot \overline{X}_{n-4} \cdot \overline{X}_{n-6} \cdot \cdots \cdot \overline{X}_2 \cdot X_1. \end{aligned} \quad (21)$$

When $F()$ is asserted it means that the leading-zero count of the input string X is an even number. The least-significant bit X_0 is unused. The reason is that when X is equal to zero, we are allowed to treat the Z bits as don't care values. The Z bits are computed by the application of the operator $F()$ to different

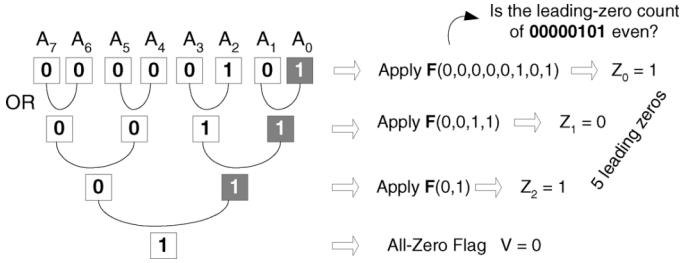


Fig. 2. Example of the new method for determining the leading-zero count of an 8-bit input operand.

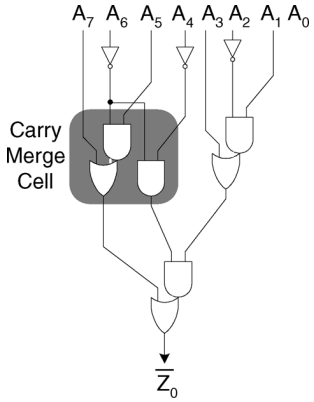


Fig. 3. Single-output carry-lookahead tree that computes Z_0 in the case of an 8-bit input operand.

groups of bits of the input operand A. For the 8-bit leading-zero count, we have that

$$\overline{Z}_0 = F(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$$
$$\overline{Z}_1 = F(A_7 + A_6, A_5 + A_4, A_3 + A_2, A_1 + A_0)$$
$$\overline{Z}_2 = F(A_7 + A_6 + A_5 + A_4, A_3 + A_2 + A_1 + A_0)$$
$$\overline{V} = A_7 + A_6 + A_5 + A_4 + A_3 + A_2 + A_1 + A_0.$$

An example of the application of the proposed method to the input operand 00000101 is shown in Fig. 2. By applying $F()$ to all the bits of the input operand, we determine whether its leading-zero count is an even number. This is false and so Z_0 is set equal to 1. This operation requires several logic stages to complete. Thus, in parallel, we perform a bitwise OR operation between the neighbor bits of the input operand and a new half-size string is derived. Again, using $F()$, we determine for the new string 0011 whether its leading-zero count is even. Since, 0011 has an even number of leading zeros, bit Z_1 is set equal to 0. In this case, $F()$ assumes half inputs compared to the computation of Z_0 allowing the computation of Z_0 and Z_1 to finish almost simultaneously. While, Z_0 and Z_1 are evaluated, we can OR the adjacent bits of the intermediate string 0011 and apply the $F()$ operator to the new word 01 which gives that $Z_2 = 1$. Thus, in parallel to the computation of Z_0 and Z_1, we can also derive the bit Z_2 of the leading-zero count. Also, at the same time the computation of the binary OR tree is also completed and its final output determines the all-zero V flag of the LZC unit. The output of the new LZC unit is equal to $Z_2Z_1Z_0 = 101$ that correctly encodes the five leading-zeros of the example input. In Fig. 2, the least significant bits of all intermediate words produced by the binary OR tree are shaded. The reason is that according to the definition of the $F()$ operator they are not used for the derivation of the Z bits. They participate only in the computation of the V flag.

For the design of arbitrary wordlength LZC units, we do not need to derive general expressions that describe the bits of the leading-zero count. A general n-bit LZC unit can be designed as a simplified version of the corresponding 2^k-bit LZC unit, where $k = \lceil \log_2 n \rceil$. For example, an 11-bit LZC unit can be easily derived by the corresponding 16-bit design. In this case, we need to form a 16-bit word that will have the same number of leading zeros with the corresponding 11-bit input. To achieve this, we generate a 16-bit word where the 11 bits of the input are put to the more significant bit positions and the five least significant bits are set to zero. We can derive the corresponding 11-bit LZC circuit by propagating the five least significant zeros to the equivalent 16-bit LZC unit and removing the gates that evaluate to a constant value. This approach produces simplified circuits with reduced delay.

IV. LZC Unit Organization

In the following, we will present two methods for the organization of the proposed LZC unit and we will analyze in detail their implementation in both static and dynamic CMOS logic. It can be easily observed that the relation (21) of the $F()$ operator resembles the well know carry-lookahead equation $G_i + P_iG_{i-1} + \cdots + P_iP_{i-1}, \ldots, P_1G_0$, where in place of the bits G_i and P_i we use the bits of the string X. Each bit Z_i of the leading-zero count is computed independently using a separate single-output carry lookahead tree. The structure of such a tree that computes the least significant bit of the leading-zero count Z_0 in the case of an 8-bit input operand is shown in Fig. 3. The basic block of the carry-lookahead tree is the well-known carry merge (CM) cell. Please notice that the right part of the design that combines the less significant bits requires a simplified form of the CM cell, which is composed only of an AND-OR gate.

Besides Z_0 that is computed directly from the input bits, the carry-lookahead trees that compute the remaining bits of the leading-zero count, assume as input the OR function of specific groups of the input bits according to the algorithm described in Fig. 2. Therefore, a complete binary tree of OR gates is required. The intermediate results produced at each level of the binary OR tree are given as input to the corresponding carry-lookahead trees that compute the bits of the leading-zero count. Moreover, the final output of the binary OR tree represents the all-zero flag V that is also required by the leading-zero counter. In our design, $\log_2 n$ independent carry-lookahead trees are required to compute the bits of the leading-zero count of an n-bit input operand. Each tree combines a different number of bits. Z_0 is computed directly from the $n-1$ bits of the input operand since A_0 does not participate in the computation, while Z_1 assumes as input the $n/2-1$ pairs $A_{n-1} + A_{n-2}, A_{n-3} + A_{n-4}, \ldots, A_2 + A_1$. In the general case, the computation of the bit Z_i of the leading-zero count is performed using a carry-lookahead tree of

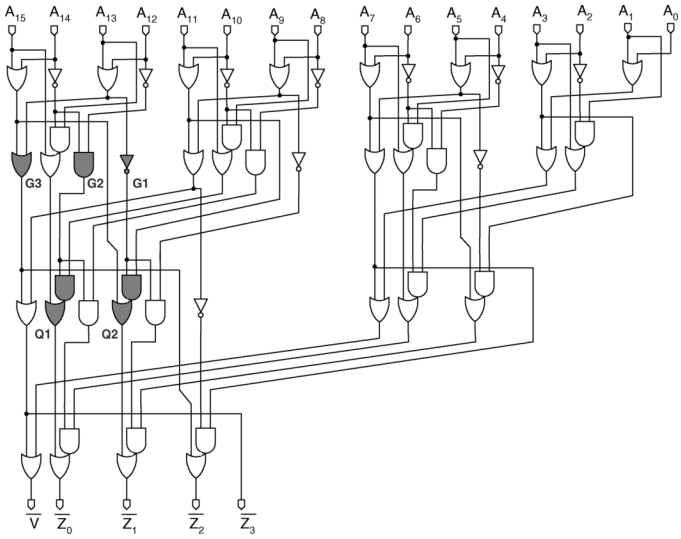


Fig. 4. Proposed 16-bit LZC unit.

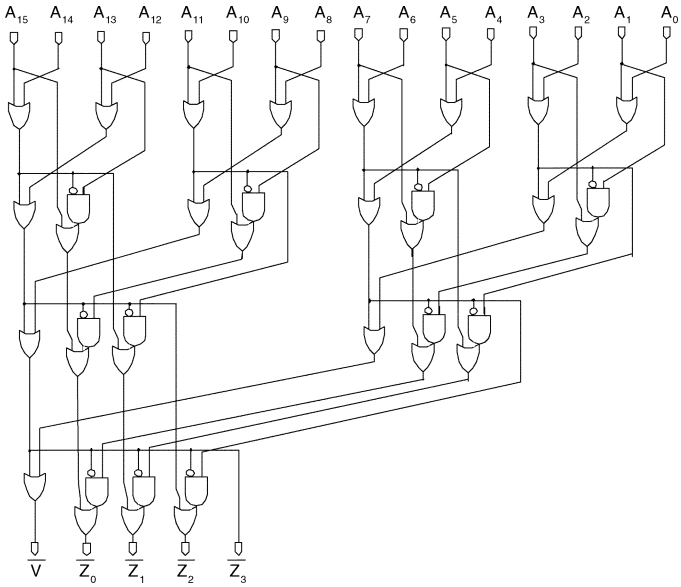


Fig. 5. Second variant of the proposed 16-bit LZC using the shared carry-propagate approach.

$n/2^i - 1$ inputs. A complete implementation of a 16-bit LZC unit is shown in Fig. 4.

Based on the properties of the carry propagate signals that are produced at each level of the circuit shown in Fig. 4, an alternative implementation can be derived. As shown in Fig. 4, gate $Q1$ of the carry-lookahead tree that is used for the computation of bit Z_0, computes the term

$$A_{15} + \overline{A}_{14} \cdot A_{13} + \overline{\underbrace{(A_{14} + A_{12})}} \cdot (A_{11} + \overline{A}_{10} \cdot A_9)$$

using $\overline{A_{14} + A_{12}}$ as a carry-propagate bit. This signal comes from the output of AND gate $G2$. Also, the output of gate $Q2$ of the carry-lookahead tree that computes bit Z_1, is equal to

$$A_{15} + A_{14} + \overline{\underbrace{(A_{13} + A_{12})}} \cdot (A_{11} + A_{10}).$$

In this case, the term $\overline{A_{13} + A_{12}}$ is used as a carry-propagate bit and is computed by the inverter $G1$. In both cases, we observe that we can use the output of gate $G3$, i.e., $\overline{A_{15} + A_{14} + A_{13} + A_{12}}$, of the binary OR tree instead of the outputs of gates $G1$ and $G2$ without altering the result of gates $Q1$ and $Q2$. For example, for gate $Q2$, if we use $\overline{A_{15} + A_{14} + A_{13} + A_{12}}$ as the carry propagate bit, we get

$$\begin{aligned} A_{15} + A_{14} + \overline{A_{15} + A_{14} + A_{13} + A_{12}} \cdot (A_{11} + A_{10}) \\ = A_{15} + A_{14} + \overline{(A_{13} + A_{12})} \cdot (A_{11} + A_{10}) \end{aligned}$$

which is equal to the original equation describing the output of gate $Q2$. Thus, the terms produced at the intermediate nodes of the binary OR tree can be reused and act as shared carry propagate signals for the carry trees that generate the bits of the leading-zero count. The same property holds for all levels of the carry logic and the rest terms produced by the binary OR tree. The circuit that follows the shared-carry-propagate architecture is shown in Fig. 5.

The two proposed solutions for the LZC unit are equally modular and regular and represent the two extremes of the design space. The straightforward computation of the bits of the leading-zero count (see Fig. 4) has a shorter critical path and the fanout of each gate is limited to 2. On the contrary, the shared-carry propagate approach has the smaller number of gates but the internal nodes of the circuit are more loaded due to the larger fan-out of the shared carry-propagate signals. This difference in the load of the internal nodes is in favor of the straightforward implementation, which is expected to be slightly faster than the second variant of the proposed LZC unit (see Fig. 5). However, the reduced number of gates of the shared-carry propagate approach will give more energy efficient solutions when the circuit is sized for larger delays.

The second variant of the proposed LZC unit is a simplified form of the circuit presented by Oklobdzija in [16], while the first variant of the proposed designs (see Fig. 4) follows a completely new circuit topology. The new designs have simplified critical paths due to the carry-lookahead approach used for the computation of the bits of the leading-zero count and also lead to more energy efficient solutions. The energy and delay reductions achieved by both the proposed solutions compared to previously known LZC units will be quantified by the experimental data given in Section VI.

The form of the equations describing the bits of the leading-zero count imposes some difficulties in the case of dynamic CMOS implementations of the proposed circuits. The difficulty stems from the fact that certain AND-OR gates of the circuit (as Gate $Q2$ shown in Fig. 4) need to combine signals of different polarity in order to compute a relation of the form $A + \overline{B} \cdot C$. Assuming a totally single-rail implementation and that each dynamic stage is followed by one stage of static logic, the signals A and C will be low in the precharge phase and possibly be high in the evaluate phase. On the contrary, we cannot guarantee that $\overline{B}$ will not perform a High-to-Low transition during the evaluate phase thus possibly deteriorating the output of the dynamic gate. Therefore, the new circuits should be designed appropriately to overcome this monotonicity problem.

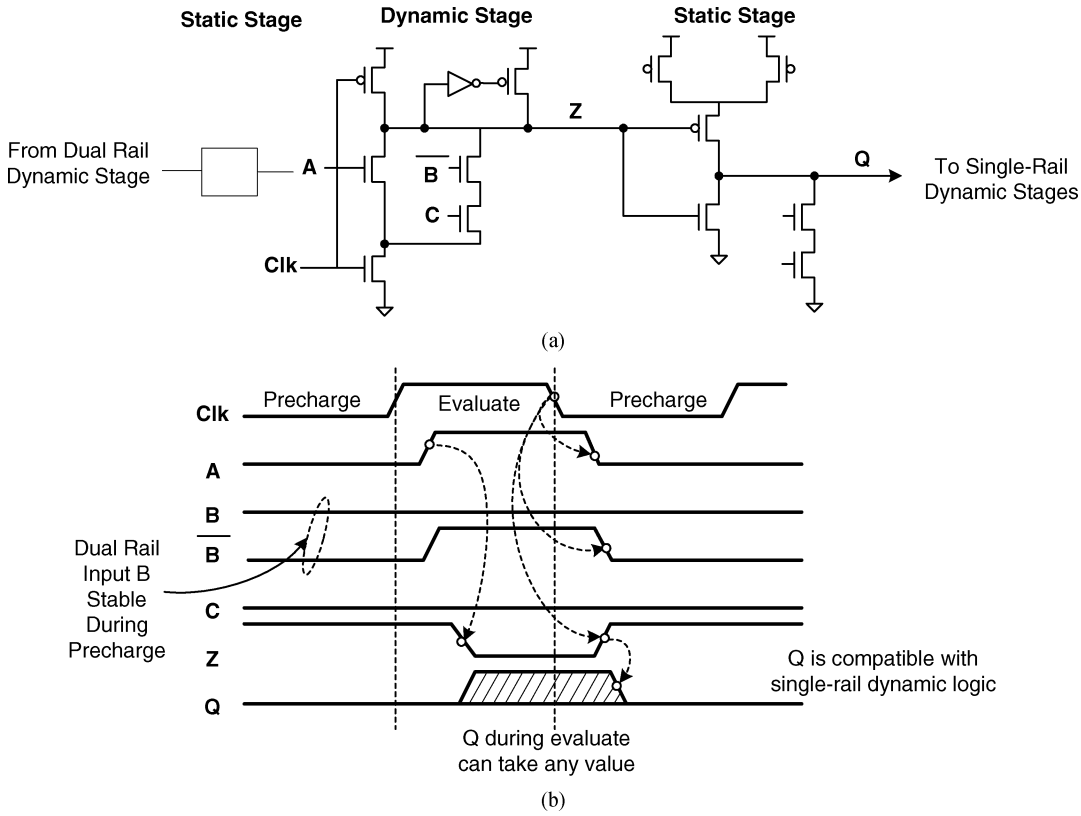


Fig. 6. Example of the hybrid dual and single rail implementation of the proposed LZC units.

One straightforward approach is to use dual-rail dynamic logic. In that case, both the true and the complement versions of all signals are available in dynamic-logic compatible form. Nevertheless, the adoption of dual-rail dynamic logic duplicates the gates of the circuit and increases significantly its power consumption. Another approach that allows us to keep the design in full single-rail form, but requires some tricky and less robust circuit techniques, is to directly cascade dynamic gates and delay the clock to the latter gate in order to ensure that the inputs are monotonic during evaluation [15].

The approach we adopted relies on a hybrid technique where dual-rail dynamic logic is used only in specific parts of the circuit, while the majority of the gates produces single rail outputs. The derivation of this technique is based on a simple observation. In both circuits of Figs. 4 and 5, we can see that the AND-OR gates that compute relations of the form $A + \overline{B} \cdot C$, receive in all cases the inverted B input either from the input bits or the output of the OR gates of the binary OR tree. Therefore, the only part of the circuit that should provide stable dual-rail signals is either the input of the circuit, or the gates of the binary OR tree. The gates of the binary OR tree are duplicated and compute both the true and the complement versions of their outputs giving the dynamic-compatible signals to the corresponding gates of the carry-lookahead trees. The rest gates of the circuit that compute the bits of the leading-zero count in a carry-lookahead fashion are implemented as single-rail dynamic gates without causing any monotonicity problems. Such techniques are not uncommon to the high-speed microprocessor design community, where both single and dual-rail dynamic signals are used for improving the speed of the design [20].

The proposed designs consist of alternating dynamic and static carry-merge stages. Also, the precharge and evaluate phases of the dynamic stages are orchestrated by overlapped clocks following the design principles described in [21], in order to minimize any timing overhead and improve skew tolerance. An example of the functionality of the mixed single and dual-rail dynamic CMOS implementations is shown in Fig. 6. In the rest of this paper, this approach for designing the proposed LZC units in dynamic CMOS, will be denoted as *single-rail* technique in order to clearly differentiate from the full dual-rail approach that we also investigated in the experimental results.

V. LZA Logic

In this section, we review several LZA architectures and we present how the proposed LZC unit can be integrated with the LZA logic. Together with the experimental results presented in Section VI, we aim to identify which one of the LZA techniques can be more efficiently used along with the proposed LZC unit. Moreover, we investigate which part of the prediction circuit among the LZA logic and the LZC unit, is more critical in terms of delay and energy for the performance of the whole circuit. Finally, a new method for handling the error of LZA logic is introduced that further reduces the complexity of the normalization circuit.

LZA logic tries to produce from the input operands A and B a prediction string of n bits that will have almost the same number of leading zeros or ones as the outcome of the true operation. Leading zeros occur when the result is positive, and leading ones occur when the result is negative. Each bit of the prediction string is equal to the value of an indicator f_i that is computed using the carry propagate $P_i = A_i \oplus B_i$, carry generate $G_i = A_i \cdot B_i$, and carry kill $K_i = \overline{A}_i \cdot \overline{B}_i$ functions of the input bits A_i and B_i. Many algorithms have been presented so far for the design of the LZA logic. Their difference lies in the boolean relations that describe the function of the indicator f_i. These techniques can be roughly separated in two categories. The first category contains the circuits that detect the case of either leading zeros or ones using a single set of indicators. The second category separates the case of leading zeros from the case of leading ones and uses two distinct sets of indicators. For both LZA architectures, the prediction string is given to a single or two separate LZC units, respectively, in order to encode the number of leading zeros or ones. Any of the already described LZC units can be used for the encoding.

In the first class of LZA logic two techniques have been proposed [8], [12], that both have almost the same complexity. The most widely used is the one presented in [8]. In this case, the indicator f_i predicts the position of the leading digit using information from both the left $i+1$ and the right $i-1$ neighbor bits irrespective of the sign of the true result. The definition of the indicator is given in (22)

$$f_i = P_{i+1} \cdot (G_i \cdot \overline{K}_{i-1} + K_i \cdot \overline{G}_{i-1}) + \overline{P}_{i+1} \cdot (K_i \cdot \overline{K}_{i-1} + G_i \cdot \overline{G}_{i-1}) \quad (22)$$

and $f_{n-1} = \overline{P}_{n-1} \cdot P_{n-2}$. If the indicator of the ith bit position is asserted and no other more significant indicator is also asserted then the leading digit is either in position i or in $i-1$. The prediction string produced by the indicators of (22) does not need any further processing and can be given directly to a LZC unit that will encode in weighted binary representation the number of leading zeros.

In the second class of LZA logic that employs two separate prediction units, the indicators are simpler since they do not need to detect both leading zeros and leading ones. The method presented in [11] predicts the leading zeros and ones using two separate units. The outputs of the two units are combined to produce a single prediction string. Each unit has its own set of indicators. For the case of leading zeros the indicator $f_i^z = P_i \oplus \overline{K}_{i-1}$ is used. In the opposite case, leading ones are detected using the indicator $f_i^o = P_i \oplus \overline{G}_{i-1}$. The indicators in each unit are ORed from left to right to create two monotonic strings of zeros followed by ones, e.g., $00\ldots01\ldots11$. The ith bit of each monotonic string equals the OR of all the bits from the most-significant position down to position i, like (1). Then, the two monotonic strings are bitwise ANDed to create a single prediction string whose left-most one (more significant) predicts the position of the leading digit. Again, in the case of a miss prediction, the leading digit of the true result may be in the following less significant bit position. Monotonic strings have also been used in [9] for the design of an finite-state machine based LZA logic. However, the design of [11] is preferred because of its higher speed and lower implementation cost. Encoding the number of leading zeros of the derived prediction string is better facilitated by the last part of the encoder-based LZC unit since the prediction string is already in monotonic form.

For this LZA architecture [11], an alternative LZC unit has been proposed in [10]. This circuit follows a hybrid structure where monotonic strings of bits are generated in groups of 6 bits, instead of the whole word, while the final encoding is selected by appropriately combining the output of each group. The organization of the LZC unit followed in [10] resembles a high-radix implementation of the circuit presented by Oklobdzija in [16]. Instead of combining groups of 2 bits, the final result is derived by examining 6-bit groups of the pseudo-result computed by the LZA logic and the selection between the groups is performed using 6-to-1 multiplexers. One encoding is derived for each one of the two LZA prediction units. The correct encoding (leading zeros or leading ones) is selected using the sign of the true result that is computed separately from the input operands. This method is efficient in the case of dynamic-CMOS implementations.

If we want to integrate the two-prediction-unit LZA logic with the proposed LZC unit, then the weighted binary representation of the leading-zero count can be computed directly from the indicators f^z and f^o. In the original version of the LZA logic with two separate units, the indicators of each unit were first ORed and two monotonically increasing strings of the form $00\ldots01\ldots11$ were generated. These strings have exactly the same form as the definition of the S string given in (1). The only difference is that in place of the bits of the input the leading-zero-indicator bits appear. We have shown that departing from the monotonic string S, a simplified form of relationships can be derived that compute the leading-zero count directly from the input bits. Therefore, the generation of a monotonic string from the indicators either f^z or f^o is redundant. The leading-zero count can be directly computed from f^z and f^o, respectively. For the case of split LZA prediction, our approach is similar to the method followed in [10] using a different LZC unit organization.

Feeding the indicators f^z and f^o to the proposed LZC units two distinct weighted binary representations are produced. The first encodes the predicted number of leading zeros and the other the predicted number of leading ones. Which one of the two binary representations contains the correct normalization information can be selected in two ways. The first one is based on the sign of the true result. If the result is positive then the encoding of the leading-zero indicators is selected. In the opposite case the encoding of the leading-one indicators contains the information that is needed for the normalization of the result. This method was followed in [10], where the actual sign is computed by a separate circuit that runs in parallel to the leading-zero counting procedure. However, in case that an end-around-carry adder is used, the output carry, which is used for determining whether to recomplement the output of the adder, could also be used to select the proper LZA count.

The second way to get the valid number of leading zeros or ones is to compare the outputs of the two LZC units and select the maximum. This approach is directly derived by the functionality of the LZA logic with two separate prediction units

presented in [11]. Recall that the last step of the original algorithm is to perform a bitwise AND between the two monotonically increasing strings derived by each unit. After this step the result is equal to the string that has the smallest number of consecutive ones to its least significant part, i.e., larger number of leading zeros. In our case, we do not have two monotonic strings but the corresponding encodings of the number of leading zeros. Therefore, selecting the maximum weighted binary representation of the two possible leading-zero counts is equivalent to the bitwise-AND operation performed by the original method. Selecting the maximum between the two representations has more delay than selecting the correct representation using the sign of the true result. However, depending on the design constraints, in some cases it may be beneficial to use the comparison-based approach.

When the selection of the correct representation of the number of leading zeros is based on the sign of the true result, we could use simpler indicators than f^z and f^o [18]. The indicators $fs_i^z = \overline{P}_i \cdot \overline{K}_{i-1}$ and $fs_i^o = \overline{P}_i \cdot \overline{G}_{i-1}$ assume that always the smaller operand has been subtracted from the larger one and the result of the true operation is always positive. Indicators fs^z detect the case of leading zeros, while fs^o detect the case of leading ones. Therefore, we can use the split LZA architecture using two LZC units driven by fs^z and fs^o, respectively. The correct encoding is selected by the sign of the true result.

So far, we have described several alternatives the designer has for the design of the LZA logic and the corresponding LZC unit. Which combination of the LZA logic and the LZC unit gives the more efficient implementation will be derived by the energy-delay comparisons presented in Section VI. We expect the proposed LZC unit to offer significant savings in terms of energy per operation to the whole prediction circuit, without increasing the delay. Also, using the simulation data, we will clarify whether the architectures with two separate prediction units, offer any benefit compared to the LZA logic that employs the combined indicator of (22).

A. LZA Error Handling

In certain cases, the prediction of the position of the leading digit may differ from that of the true result by one. Then, the result is not correctly normalized and an additional shift left by one position should take place. The exponent should be also decreased by one. Many methods have been proposed so far to handle the one-bit error of the LZA logic.

The first one (LZE I) involves checking the most-significant bit of the output of the normalization shifter. In case that it is equal to zero, it means that the result is not normalized and an additional shift is required. More bits from the intermediate levels of the shifter can be checked to reduce the delay overhead in the last shifting stage. Although this technique is conceptually simple it imposes a significant delay overhead to the normalization shifter. In some cases, like [22], this delay overhead causes the designers to move the correction shift to the next pipeline stage following the normalization shift, so as not to increase the clock cycle.

The second approach (LZE II) combines the information produced by the LZA logic with signals from the adder that produces the true unnormalized result. In some cases the output of the adder is ANDed with the one-hot encoding of the anticipated leading digit. If all the bits of the derived word are equal to zero then the predicted position of the leading digit is not correct. Hence, an all-zero detector is used to detect the miss-prediction in parallel to the shifter. This approach is used in [23]. In other cases, the carries at each bit of the adder are used together with the one-hot representation of the leading digit that is produced by the LZA logic to generate an error indication signal e_i [12]. If at least one of the error indicators is asserted then the predicted leading-zero count is off by one. The error is detected using a binary OR tree that runs in parallel to the normalization shifter. In both cases, when an error is detected, the normalization should perform an additional shifting operation to correct the error.

Both variants of LZE II rely on the generation of n-bit one-hot string, where the only 1 denotes the predicted position of the leading digit. This property limits the designer's choices concerning the LZC unit that could be used for encoding the number of leading zeros of the pseudo result that was generated by the LZA logic. For example, if an LZC unit like the proposed one is used, or the one presented in [16], then the output of the LZA logic would be the $\log_2 n$ bits of the weighted binary representation of the number of leading zeros of the pseudo result. This representation can be directly used by the normalization shifter and the exponent update logic without any further processing. However, since LZE II requires the position of the leading digit to be encoded in one-hot form an extra $\log_2 n$-to-n decoder is needed to perform the needed transformation. In this case, the extra decoder increases significantly the delay and the energy per operation of the error detection logic. Therefore, the only LZC unit that could be used, should be similar to the encoder-based LZC unit, described in Section II, where the one-hot representation of the leading digit is already produced as an intermediate result of the encoding procedure.

The last approach (LZE III) generates an error indication signal in parallel with the adder and the LZA logic [7], [24]. This approach uses the indicators of the LZA logic to detect specific patterns of the input bits that cause the error. The circuits that implement this form of pattern detection have significant delay and energy cost, compared to LZE I and II.

We propose a new LZA error detection and correction method that is as simple as the LZE II techniques without imposing any limitations on the selection of the LZC unit. The structure of the proposed error handling method is shown in Fig. 7. The only circuit added is a single-output carry tree that computes the least significant bit (LSB) of the leading-zero count of the true unnormalized result. This circuit is equivalent to the one shown in Fig. 3. The circuit runs in parallel with the shifter and together with the value of the LSB of leading-zero count that is predicted by the LZA logic, controls the last shifting stage of the normalization shifter. When the LSB of the leading-zero count of the true result, denoted as L, is different from the LSB of the predicted leading-zero count, denoted as Z_0, then an error has occurred.

If the LZA was exact, the last stage of the normalization shifter would perform either no shift or a left shift by one position. As described in Section V, we assume that in the case of an error, the predicted leading-zero count is smaller than the

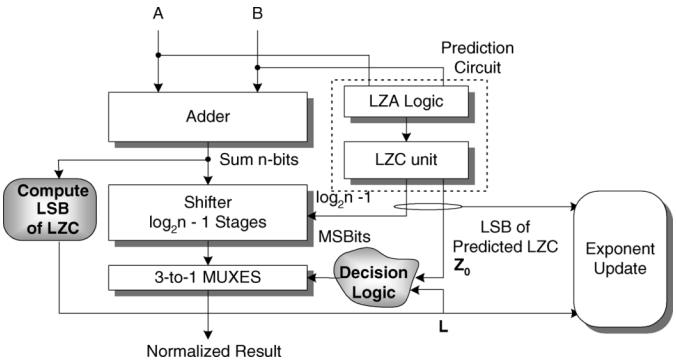


Fig. 7. Proposed method for detecting and correcting the 1-bit error of the LZA logic.

leading-zero count of the true result by 1. Therefore, to correct the one position error, we would either add one extra shift stage or we could transform the last stage of the normalization shifter to perform a left shift by zero, one, or two positions, according to the output of the decision logic. We chose the second approach since it leads to a faster circuit. When (L, Z_0) is equal to (0,0) or (1,1) no error occurs. In the first case, the result is already normalized, while in the second case a left shift by one position is required. On the contrary, when (L, Z_0) is equal to (0,1) or (1,0) an error has occurred and a left shift by 2 or 1 position, respectively, is required to correct it.

VI. Energy-Delay Comparisons

The proposed architectures have been evaluated using CMOS implementations in UMC 130-nm standard performance CMOS technology [25]. All measurements were performed for the typical process corner at a temperature of 70 °C, assuming a nominal supply voltage of 1.2 V (1 FO4 $\approx$ 62 ps). In order to explore the energy-delay space for each design, we performed gate sizing for several delay targets, beginning from the circuit's minimum achievable delay. Optimization is performed using an in-house tool developed around the geometric programming solver of [26] and following the gate sizing methodology presented in [27]. For the derived gate sizes, the energy and the delay of each circuit have been measured in HSpice. During optimization and measurements we assumed that the outputs of the circuit are loaded with a capacitance of 100 fF that roughly corresponds to the capacitance of a 350-μm metal-2 wire in our technology. Interstage wiring loads, both capacitance and resistance, have also been taken into account, assuming for the design a bit slice of 16 metal-1 tracks as the one used in state-of-the-art microprocessors [23]. To get reasonable delays, all compared designs have been optimized assuming that the maximum allowable input capacitance of each circuit is less than 25 fF which corresponds to a ratio of 4 for the circuit's output capacitance to input capacitance.

A. LZC Units

At first, the proposed 64-bit LZC units were compared to the most efficient architecture [16] for static CMOS implementations. The energy-delay behavior of the circuits is shown in Fig. 8(a). It can be observed that the proposed methodology leads to circuits that are slightly faster than the design of [16], having a smaller (by 4%) minimum achievable delay. The main benefit of the proposed designs compared to previous approaches is their energy efficiency. For equal delay measurements, the energy savings range from 10% to 49%. This result stems from the reduced number of gates required to compute the leading-zero count and the simpler gates that appear on the critical path. The shared-carry propagate approach (see Fig. 5) requires more energy than the straightforward implementation of the proposed LZC unit (see Fig. 4) for delays smaller than 8.5 FO4. This behavior is explained by the fact that the second variant of the proposed LZC unit has more gates on the critical path and larger fanout of the internal nodes of the circuit compared to the straightforward implementation. Therefore, the derived design has increased gate sizes that also increase the energy requirements of the circuit. For delay targets greater than 8.5 FO4, the shared-carry propagate approach gives the most energy efficient circuit because it requires the smallest number of gates. Nevertheless, for all delay targets the proposed designs, either the first or the second variant, offer the most energy efficient implementation compared to the design of [16].

Our experiments showed that the minimum achievable delay of the encoder-based LZC unit and the design of [19] in static CMOS implementations is almost twice the delay of the LZC units under comparison (around 11 FO4 for the encoder-based LZC unit and the design of [19]). Therefore, simulation results for the encoder-based LZC unit and the design of [19] have not been included in static CMOS, since no valid comparisons can be made. Due to the structure of the design of [19] the bits of the leading-zero count are computed with a certain delay difference. Although the critical path is above 11 FO4 the more significant bits are computed earlier with a delay of 8.5 FO4, which still is 22% worse than the proposed design. At this minimum delay point the energy of the circuit is well above the proposed LZC units, which if sized for a delay target of 8.5 FO4 require less than 2 pJ of energy per operation.

Similar conclusions can be derived if we consider the dynamic CMOS implementations of the proposed LZC unit and the already known architectures. We implemented a 64-bit encoder-based LZC unit using 8-bit-wide dynamic OR gates in each stage. Also, we included in the comparisons the LZC unit presented in [10] assuming a radix-8 implementation. We chose a radix-8 circuit since it gives a faster solution in our technology compared to the radix-6 design originally proposed in [10]. The energy-delay characteristics of those two dynamic LZC units are shown in Fig. 8(b). The straightforward implementation of the proposed circuit (see Fig. 4) has been also implemented in dynamic logic following the circuit techniques described in Section IV. We have included in the experiments a full dual-rail implementation of the proposed LZC in order to quantify the savings earned by the single-rail technique.

The energy-delay diagram of Fig. 8(b) reveals that the proposed approach leads to significantly faster and more energy efficient solutions. The proposed single-rail implementation of the LZC unit is 12% and 40% faster than the circuits proposed in [10] and the encoder-based implementation, respectively. Also, in all cases the proposed dynamic LZC unit requires the least energy per operation. The reductions achieved by the proposed

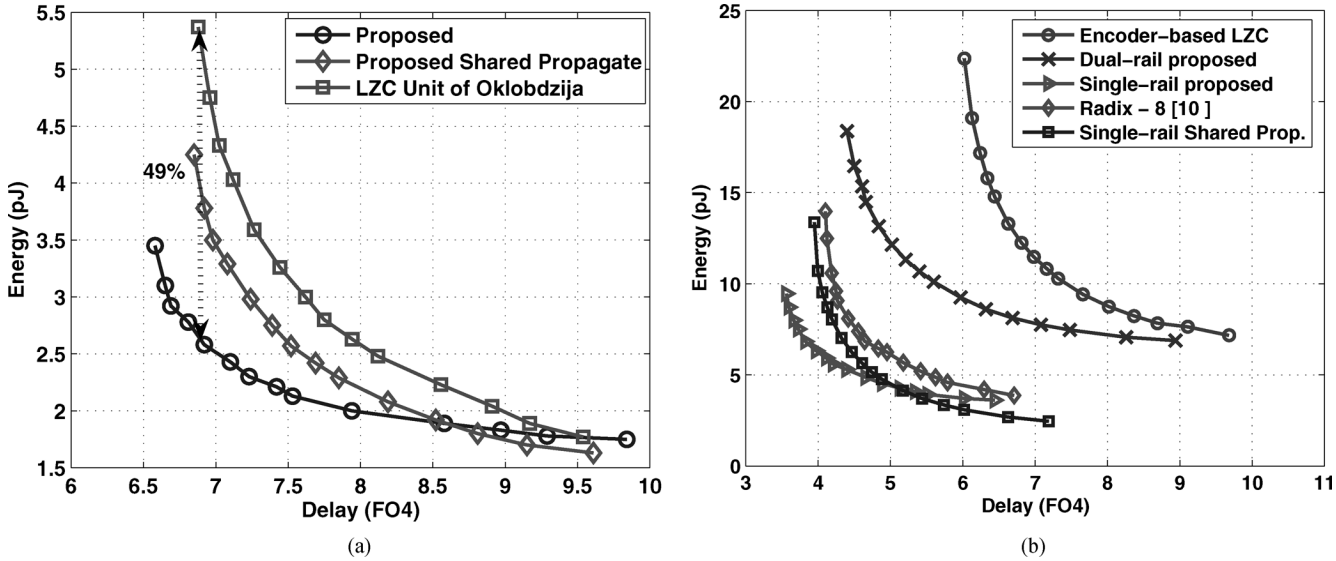


Fig. 8. Energy-delay curves for (a) the two variants of the proposed 64-bit LZC units and the circuit proposed by Oklobdzija in [16] and (b) the dynamic CMOS implementations of proposed counters as well as the encoder-based LZC unit and the design of [10].

single rail implementation reach 55% compared to the most efficient so far design in dynamic CMOS [10]. The corresponding energy reductions compared to the encoder-based LZC unit exceed 80%. The energy savings mostly come from the simple structure of the proposed LZC unit and the reduced number of gates on the critical path that leads to both faster solutions and reduced gate sizes. Finally, from Fig. 8(b) it is evident that the single-rail implementation of the proposed LZC unit saves more than 45% of energy compared to the corresponding dual-rail implementation. The single-rail form of the second variant of the proposed LZC unit is slower than the straightforward implementation, but gives the most energy efficient designs for delay targets larger than 5 FO4.

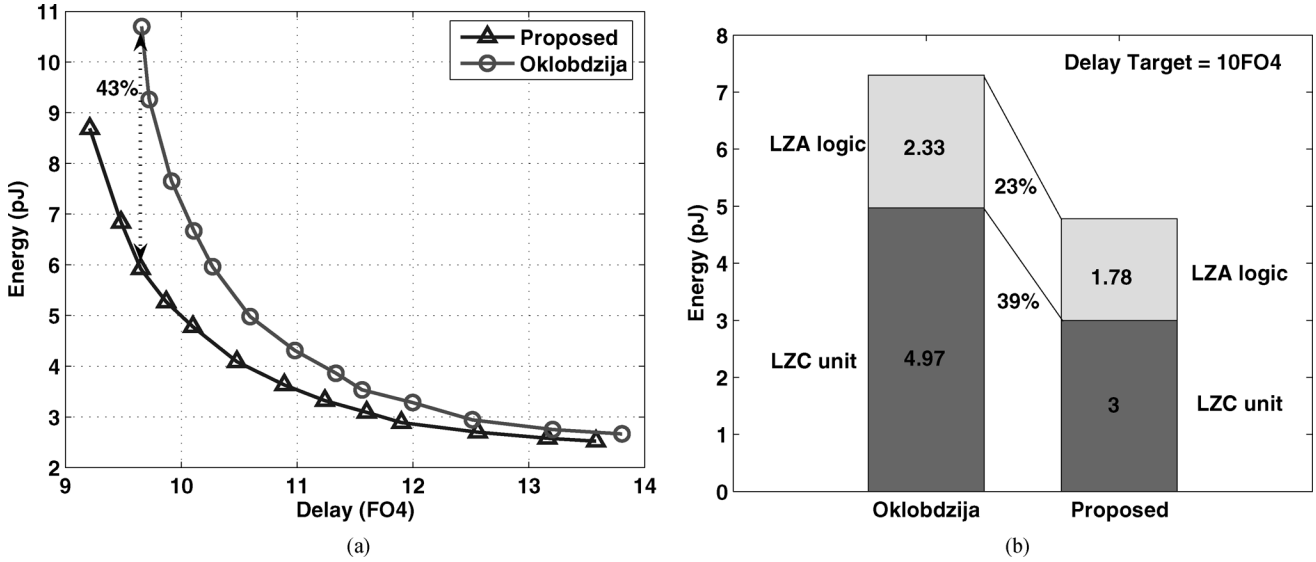
B. LZA Circuits

The minimum delay $\approx$ 6.6 FO4 achieved in static CMOS by the new LZC unit (see Fig. 4) leaves a lot of room for the insertion of any form of LZA logic. The available delay slack is determined by the speed of the adder that computes the true result. In our technology a 64-bit static CMOS radix-2 Ling adder [28] following the Kogge-Stone architecture [29] has a minimum delay of 9.3 FO4 requiring at this point 25 pJ per addition. At first, we are interested in quantifying the overhead imposed by the combined-indicator LZA logic to the whole prediction circuit.[1] Thus, two circuits are implemented. In the first case, the combined indicators drive the proposed LZC unit while in the second case they drive the LZC unit proposed by Oklobdzija in [16]. The energy-delay behavior of both circuits is shown in Fig. 9(a). From Fig. 9(a), it can be derived that for all delay targets the prediction circuit that uses the proposed LZC unit requires the smallest energy compared to the prediction circuit that uses the LZC unit proposed in [16]. For small delays, the energy savings are more than 40%. For larger delay targets it is better to employ the shared-carry propagate approach reducing further the energy of the prediction circuit.

Using the simulation data gathered for both circuits, we are interested in investigating which part of the prediction circuit, either the LZA logic or the LZC unit, is more critical in terms of energy and delay. For all cases of Fig. 9(a), including both circuits under comparison, the simulations show that the critical path is unevenly distributed between the LZA logic and the LZC unit. The LZA logic is responsible for roughly the 1/3 of the delay of the critical path, while the LZC unit contributes to the 2/3 of the total delay. Therefore, as far as delay is concerned, the more critical part of the prediction circuit that needs to be better optimized is the LZC unit and not the LZA logic. The same distribution roughly holds for energy also. The LZC unit is responsible for more than 60% of the total energy per operation required by the prediction circuit. Therefore, the optimization of the LZC unit is more important than the optimization of the combined-indicator LZA logic. This result is better shown using the diagram of Fig. 9(b). In Fig. 9(b), we present the energy breakdown of both circuits compared in Fig. 9(a), when they are both optimized for a delay target of 10 FO4 under the same input and output capacitance configuration used in deriving the energy-delay curves of Fig. 9(a). In each case the combined-indicator LZA logic is only responsible for roughly the 32% of the energy required per operation. The proposed LZC unit offers a 39% reduction in the energy of the LZC part of the circuit. Also, the simpler LZC circuit has as a consequence the reduction of the energy of the LZA part by 23%, since, for the same delay target, it reduces the capacitance driven by the LZA logic, leading to smaller gate sizes.

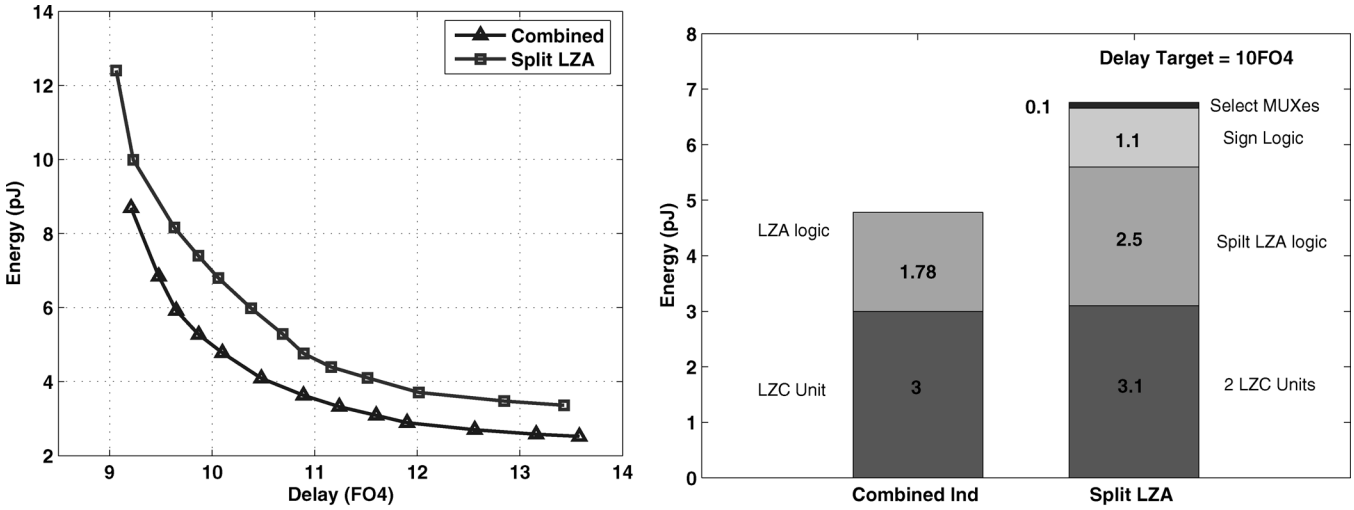
In the following, we analyze the case of split LZA architectures and compare them to the prediction circuit that uses a combined-indicator LZA logic in static CMOS. In all cases we use the proposed LZC units since they require significantly less energy per operation compared to the best previous implementation [16]. The split prediction circuits of [10] and [11] are

[1]The term *prediction circuit* denotes the pair of the LZA logic and the LZC unit that predicts the position of the leading digit, as shown in Fig. 7.



(a) (b)

Fig. 9. (a) Energy-delay curves for the static CMOS implementation of the prediction circuit that uses the combined-indicator LZA logic along with the proposed LZC unit and the design of Oklobdzija [16], respectively. (b) The energy breakdown of the prediction circuits using different LZC units and sized for a delay of 10 FO4.



(a) (b)

Fig. 10. (a) Energy-delay curves for the static CMOS implementation of the prediction circuit using the combined-indicator LZA logic and the split LZA technique. In both cases the proposed LZC units are used. (b) The energy breakdown of the prediction circuits that use the combined-indicator LZA logic and the split LZA logic, respectively, along with the proposed LZC units, when sized for a delay of 10 FO4.

more suitable for dynamic CMOS implementations taking advantage of the efficient implementation of wide OR functions, while they do not fit well with static CMOS. Therefore, they are not included in this set of experiments. We implemented the prediction circuit that uses two sets of indicators, as described in Section V. We examined the case of the indicators (fs^z, fs^o) (Split LZA) since they lead to more efficient circuits compared to the more complex indicators (f^z, f^o). For the implementation of the split-LZA prediction circuits, we assumed that the correct number of leading zeros is selected according to the value of the true sign of the addition using 2-to-1 multiplexers. The sign is computed by a separate carry lookahead tree, whose energy and delay overhead has been also included in the designs. The energy-delay diagrams derived are shown in Fig. 10(a), where we have also included the prediction circuit that uses the combined-indicator LZA logic and the proposed LZC unit.

From Fig. 10(a), we can see that the split architectures are faster only by 3% but they require 18% more energy on average than the prediction circuit with the combined-indicator LZA logic. Fig. 10(b) depicts the energy breakdown of the prediction circuits that use the combined-indicator LZA logic and the split LZA logic, respectively, utilizing in both cases the proposed LZC units. Both circuits are optimized for a delay of 10 FO4. We can see that the energy of the LZA logic of the split-LZA prediction unit is more than the energy of the combined-indicators LZA logic. This happens because the indicators fs^z and fs^o that share the carry bits *G*, *P*, *K*, have increased output load since they drive two separate LZC units. The *G*, *P*, and *K* bits are also used by the carry tree that computes the sign of the

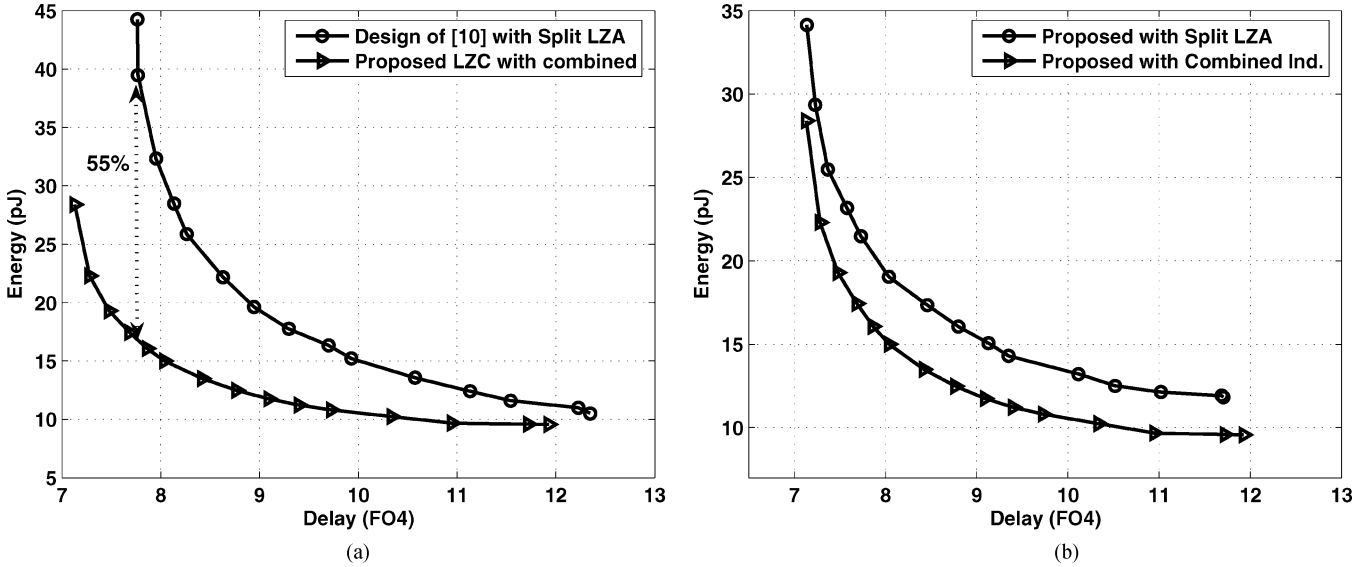


Fig. 11. (a) Energy-delay curves for the dynamic CMOS implementation of the prediction circuits using the combined-indicator LZA logic and the proposed single-rail LZC unit, along with the prediction circuit of [10] using split LZA technique. (b) The same predictions circuits using in all cases the proposed single-rail LZC unit.

true result. The energy of the two LZC units is almost equal to the energy of the single LZC unit required in the combined-indicators approach. The energy of the two LZC units does not increase, since, after delay optimization, they get less input capacitance compared to the LZC unit with the combined indicator. The capacitance removed from the two LZC units goes to the carry tree that computes the sign of the true result, so that all paths have almost equal delay. Therefore, the energy overhead imposed by the split LZA architecture and the needed sign detection logic is significant compared to the combined-indicators LZA logic. Again, the LZA architecture is responsible for roughly the one-third of the total energy of the prediction circuit. Concluding with the static CMOS prediction circuits, we can say that, when using the proposed LZC unit that gives the most energy efficient designs, there is no meaning to use the split LZA architectures since in all cases they give less efficient circuits compared to the combined-indicator LZA approach.

The same conclusion can be derived for the case of dynamic CMOS implementations of the prediction circuit. For this case, we performed two sets of experiments. At first, we compared the prediction circuit that uses the combined-indicator LZA logic and the single-rail implementation of the proposed LZC unit, with the most efficient previous architecture of [10]. The combined-indicator LZA logic is implemented in full dual-rail dynamic CMOS which then drives the proposed single-rail LZC unit. For the design of [10], we implemented the simpler split LZA approach.[2] The obtained energy-delay curves are shown in Fig. 11(a). In all cases, the prediction circuit with the combined-indicators LZA logic and the proposed LZC unit provides the most efficient designs. For small delays, which is the area of interest in dynamic CMOS, the energy savings are significant and over 55%. In the second set of experiments, we want to determine, which LZA architecture offers the most efficient implementations in dynamic CMOS implementations. Therefore, we investigated two prediction circuits each one using a different LZA technique, i.e., Split LZA and the combined-indicators LZA logic. In both cases the number of leading zeros is encoded using the proposed single-rail LZC unit. The results derived are shown in Fig. 11(b). Clearly, the most efficient prediction circuit remains the combined-indicators LZA logic when using the proposed LZC unit.

[2]In [10], the indicators (f^z, f^o) were used for the design of the prediction circuit, which offer less efficient designs compared to the simpler indicators (fs^z, fs^o) that are implemented in this paper and compared to the proposed solutions.

C. LZA Error Handling Methods

A set of experiments have also been performed in order to quantify the energy and the delay requirements of the proposed and the previous LZA error handling methods. As a baseline of our comparisons we assume the energy and the delay of a standalone shifter, which is composed of $\log_2 n$ stages of 2-to-1 multiplexers. All error handling methods should try to keep to minimum the delay and the energy overhead they add to the normalization shifter. Using the same parameters as in all other circuits under comparison, it is derived that the minimum delay achieved by the shifter equals 9.5 FO4. At this delay point the energy of the circuit is roughly equal to 32 pJ.

At first, we designed the technique LZE I. Examining the MSB of the output of the shifter and controlling the extra shifting stage according to its value, imposes a significant delay overhead. Although the signal is appropriately buffered, the delay overhead is over 18%. The extra delay is caused because of the large fan-out imposed by the select line. The energy of the normalization shifter used in LZE I is 27 pJ. However, due to the large delay difference no valid conclusions can be made. Sizing the normalization shifter for a delay equal to the shifter of LZE I reduces its energy to 18 pJ.

LZE II and the proposed error handling method have almost the same performance in terms of energy and delay. Their minimum achievable delay is equal to 9.9 FO4, which is only 4%

worse than the standalone normalization shifter. Also, the energy overhead is around 20% for both methods. For the case of LZE II, we included only the overhead imposed by n AND gates used to detect if the position of the predicted leading digit matches the leading digit of the true unnormalized result and the binary OR tree used to detect the presence of a bit equal to 1. We did not include any of the overhead imposed by the encoder-based LZC unit or the extra circuits that may be used to produce a monotonic string that denotes the position of the leading digit. If we at least include the overhead added by the encoder-based LZC unit that is required to produce the monotonic prediction string used by LZE II, we can safely conclude that the proposed method imposes significantly less energy overhead to the normalization shifter, while keeping the delay penalty to a minimum.

VII. Conclusion

Two new LZC circuits have been presented in this paper. Their design is based on a newly developed mathematical framework describing the bits of the leading-zero count, while their computation is reduced to well-known carry-lookahead techniques in a unified manner. Significant energy reductions are achieved by the proposed designs compared to the most efficient previous implementations both in static and dynamic CMOS logic. Based on the new LZC units, simplified prediction circuits are derived that outperform already reported architectures. From the presented analysis, the most efficient combination of LZA logic and LZC unit is derived for the design of the whole prediction circuit. Also, a novel technique for handling the possible error of LZA logic was described that imposes the minimum overhead to the normalization shifter without introducing any further limitation.

Acknowledgment

The authors would like to thank the reviewers for their constructive comments.

References

[1] A. Beumont-Smith, N. Burgess, S. Lefrere, and C. C. Lim, "Reduced latency IEEE floating-point standard adder architecture," in *Proc. 15th IEEE Symp. Comput. Arithmetic*, Jul. 2001, pp. 35–42.

[2] P. M. Seidel and G. Even, "Delay-optimized implementation of IEEE floating-point addition," *IEEE Trans. Comput.*, vol. 53, no. 2, pp. 97–113, Feb. 2004.

[3] S. M. Mueller, C. Jacobi, H.-J. Oh, K. D. Tran, S. R. Cottier, B. W. Michael, H. Nishikawa, Y. Totsuka, T. Namatane, N. Yano, T. Machida, and S. H. Dhong, "The vector floating-point unit in a synergistic processor element of a CELL processor," in *Proc. 17th IEEE Symp. Comput. Arithmetic*, Jun. 2005, pp. 59–67.

[4] G. Gerwig and M. Kroener, "Floating-point unit in standard cell design with 116 bit wide dataflow," in *Proc. 14th IEEE Symp. Comput. Arithmetic*, Apr. 1999, pp. 266–273.

[5] N. Ide, M. Hirano, Y. Edo, S. Yoshioka, H. Murakami, A. Kunimatsu, T. Sato, T. Kamei, T. Okada, and M. Suzuoki, "2.44-GFLOPS 300-MHz floating-point vector-processing unit for high-performance 3-D graphics computing," *IEEE J. Solid-State Circuits*, vol. 35, no. 7, pp. 1025–1033, Jul. 2000.

[6] *IEEE Standard for Binary Floating-Point Arithmetic*, Std 754-1985, American National Standards Institute and Institute of Electrical and Electronic Engineers, 1985.

[7] J. D. Bruguera and T. Lang, "Leading-one prediction with concurrent position correction," *IEEE Trans. Comput.*, vol. 48, no. 10, pp. 1083–1097, Oct. 1999.

[8] S. C. Knowles, "Arithmetic processor design for the T9000 transputer," in *Proc. SPIE*, Dec. 1991, vol. 1566, pp. 230–243.

[9] E. Hokenek and R. K. Montoye, "Leading-zero anticipator (LZA) in the IBM RISC system/6000 floating point execution unit," *IBM J. Res. Development*, vol. 34, pp. 71–77, Jan. 1990.

[10] K. T. Lee and K. J. Nowka, "1 GHz leading-zero anticipator using independent sign-bit determination logic," in *Symp. VLSI Circuits Dig. Techn. Papers*, 2000, pp. 194–195.

[11] M. S. Schmookler and K. J. Nowka, "Leading zero anticipation and detection : A comparison of methods," in *Proc. 15th IEEE Symp. Comput. Arithmetic*, Jul. 2001, pp. 7–12.

[12] N. Quach and M. J. Flynn, "Leading one prediction—Implementation, generalization, and application," Stanford University, Stanford, CA, Tech. Rep. CSL-TR-91-463, 1991.

[13] D. Harris, "An exponentiation unit for an OpenGL lighting engine," *IEEE Trans. Comput.*, vol. 53, no. 3, pp. 251–258, Mar. 2004.

[14] S. Sudharsanan and M. Sinnathamby, "Support for variable length decode on embedded processors," in *Workshop Media Signal Process. Embed. Syst. SoCs*, Sep. 2004, pp. 33–51.

[15] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Boston, MA: Addison Wesley, 2005.

[16] V. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 2, no. 1, pp. 124–128, Mar. 1994.

[17] V. Oklobdzija, "Comment on "Leading-zero anticipatory logic for high-speed floating point addition"," *IEEE J. Solid-State Circuits*, vol. 32, no. 2, pp. 292–293, Feb. 1997.

[18] H. Suzuki, H. Morinaka, H. Makino, Y. Nakase, K. Mashiko, and T. Sumi, "Leading-zero anticipatory logic for high-speed floating point addition," *IEEE J. Solid-State Circuits*, vol. 31, no. 8, pp. 1157–1164, Aug. 1996.

[19] T. Lang and J. D. Bruguera, "Floating-point multiply-add-fused with reduced latency," *IEEE Trans. Comput.*, vol. 53, no. 8, pp. 988–1003, Aug. 2004.

[20] M. Anders, R. Rios, K. Mistry, S. Mathew, R. Krishnamurthy, and K. Soumyanath, "Sub-500-ps 64-b ALUS in 0.18-μm SOI/BULK CMOS: Design and scaling trends," *IEEE J. Solid-State Circuits*, vol. 36, no. 11, pp. 1636–1646, Nov. 2001.

[21] D. Harris and M. Horowitz, "Skew-tolerant domino circuits," *IEEE J. Solid State Circuits*, vol. 32, no. 11, pp. 1702–1711, Nov. 1997.

[22] S. D. Trong, M. Schmookler, E. M. Schwarz, and M. Kroener, "P6 binary floating-point unit," in *Proc. 18th IEEE Symp. Comput. Arithmetic*, Jun. 2007, pp. 77–86.

[23] H.-J. Oh, S. M. Mueller, C. Jacobi, K. D. Tran, S. R. Cottier, B. W. Michael, H. Nishikawa, Y. Totsuka, T. Namatane, N. Yano, T. Machida, and S. H. Dhong, "A fully-pipelined single-precision floating point unit in the synergistic processor element of a CELL processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 4, pp. 759–771, Apr. 2006.

[24] G. Zhang, Z. Qi, and W. Hu, "A novel design of leading zero anticipation circuit with parallel error detection," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2005, pp. 676–679.

[25] Europractice IC Service, "UMC 130 nm CMOS FSG process," 2008.

[26] A. Mutapcic, K. K. , S. Kim, and S. Boyd, "GGPLAB: A MatLab Toolbox for geometric programming," 2006 [Online]. Available: http://www.stanford.edu/~boyd/ggplab/

[27] S. P. Boyd, S. J. Kim, D. Patil, and M. A. Horowitz, "Digital circuit optimization via geometric programming," *Operations Res.*, vol. 53, no. 6, pp. 899–932, Nov./Dec. 2005.

[28] G. Dimitrakopoulos and D. Nikolos, "High-speed parallel-prefix VLSI ling adders," *IEEE Trans. Comput.*, vol. 54, no. 2, pp. 225–231, Feb. 2005.

[29] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Comput.*, vol. C-22, no. 8, pp. 786–792, Aug. 1973.

Giorgos Dimitrakopoulos (M'07) received the Diploma in computer engineering and informatics and the M.Sc. in hardware/software integrated systems , and the Ph.D. degree in computer engineering from the University of Patras, Patras, Greece, in 2001, 2003, and 2007, respectively.

His research interests include VLSI design, computer architecture, energy optimization of digital systems, and design of media-enhanced microprocessors. He is currently fulfilling his military service.

Dr. Dimitrakopoulos is a member of the Technical Chamber of Greece.

Kostas Galanopoulos is currently an undergraduate student of the Computer Engineering and Informatics Department, University of Patras, Patras, Greece.

His research interests include microprocessor datapath design, energy-delay optimization, and low leakage circuit techniques.

Christos Mavrokefalidis (S'03) received the Diploma degree in computer engineering and informatics and the M.S. degree in signal processing from the University of Patras, Patras, Greece, in 2004 and 2006, respectively, where he is currently pursuing the Ph.D. degree in signal processing.

His research interests focus on the area of signal processing for cooperative communications.

Mr. Mavrokefalidis is a member of the Technical Chamber of Greece.

Dimitr Nikolos (M'95) received the B.Sc. degree in physics, the M.Sc. degree in electronics, and the Ph.D. degree in computer science from the University of Athens, Athens, Greece.

Since 1999, he has been a full Professor with the Computer Engineering and Informatics Department, University of Patras, Patras, Greece, and head of the Technology and Computer Architecture Laboratory. His main research interests include fault-tolerant computing, computer architecture, VLSI design, test, and design for testability. He has authored or co-authored more than 150 scientific papers and holds one U.S. patent.

Prof. Nikolos was a co-recipient of the Best Paper Award for his work "Extending the Viability of IDDQ Testing in the Deep Submicron Era" presented at the 3rd IEEE International Symposium on Quality Electronic Design (ISQED 2002). He has served as program co-chairman of the IEEE Int. On-Line Testing Workshops (1997–2001). He also served on the program committees for the IEEE Int. On-Line Testing Symposiums (2002–2006), for the IEEE International Symposiums on Defect and Fault Tolerance in VLSI Systems (1997–1999), for the Third and Fourth European Dependable Computing Conference, for the International Workshop on Power and Timing Modeling, Optimization, and Simulation-PATMOS (2005–2007), and for the DATE (2000–2007) Conferences. He was guest co-editor of the June 2002 Special Issue of Journal of Electronic Testing, Theory, and Applications (JETTA) devoted to the 2001 IEEE International On-Line Testing Workshop.