# Brief Contributions

## High-Speed Parallel-Prefix VLSI Ling Adders

Giorgos Dimitrakopoulos and
Dimitris Nikolos, *Member*, *IEEE*

**Abstract**—Parallel-prefix adders offer a highly efficient solution to the binary addition problem and are well-suited for VLSI implementations. In this paper, a novel framework is introduced, which allows the design of parallel-prefix Ling adders. The proposed approach saves one-logic level of implementation compared to the parallel-prefix structures proposed for the traditional definition of carry lookahead equations and reduces the fanout requirements of the design. Experimental results reveal that the proposed adders achieve delay reductions of up to 14 percent when compared to the fastest parallel-prefix architectures presented for the traditional definition of carry equations.

**Index Terms**—Adders, parallel-prefix carry computation, computer arithmetic, VLSI design.

## 1 Introduction

BINARY addition is one of the primitive operations in computer arithmetic. VLSI integer adders are critical elements in general-purpose and digital-signal processing processors since they are employed in the design of Arithmetic-Logic Units, in floating-point arithmetic datapaths and in address generation units. They are also employed in encryption and hashing function implementation.

A large variety of algorithms and implementations have been proposed for binary addition [1], [2], [3]. When high operation speed is required, tree structures, like parallel-prefix adders, are used [4], [5], [6], [7], [8], [9]. Parallel-prefix adders are suitable for VLSI implementation since they rely on the use of simple cells and maintain regular connections between them. The prefix structures allow several trade offs among the number of cells used, the number of required logic levels, and the cells' fanout. A recent comparison of the most efficient adder architectures has been presented in [10].

Several variants of the carry-lookahead equations, like Ling carries [11], have been presented that simplify carry computation and can lead to faster structures. The simplified form of Ling equations has been exploited for the design of multilevel block carry lookahead adders [11], [12], [13], [14], [15], [16], [17], [18]. Nevertheless, no systematic methodology for designing parallel-prefix structures for Ling carry computation that takes full advantage of the simplicity of Ling equations has been presented. Although, in [19], a method was presented that computes the Ling carries using a parallel-prefix network, the design proposed requires an extra OR gate compared to the traditional carry lookahead parallel-prefix structures and is therefore against the inherent fast computation of Ling carries. In this work, we propose the parallel-prefix formulation of Ling addition. The proposed adders are implemented using one less logic level compared to the parallel prefix structures proposed for the traditional carry equations, while they also reduce the fanout requirements of the design. Following the proposed methodology, any parallel-prefix architecture can be employed for the design of high-speed Ling adders. The proposed parallel-prefix adders are compared to the widely adopted prefix structures proposed for the traditional definition of carry equations using static CMOS implementations. In all cases, the proposed adders are the fastest. The delay reductions achieved range from 10 percent to 14 percent.

The rest of the paper is organized as follows: Section 2 gives a brief description of the parallel-prefix formulation of binary addition and the appropriate definitions concerning Ling addition. Section 3 introduces the proposed parallel-prefix Ling adders, while, in Section 4, experimental results are given. Finally, conclusions are drawn in Section 5.

• *The authors are with the Computer Engineering and Informatics Department, University of Patras, 26500 Patras, Greece.*
*E-mail: dimitrak@ceid.upatras.gr, nikolosd@cti.gr.*

*Manuscript received 9 Dec. 1003; revised 14 Oct. 2004; accepted 21 Oct. 2004; published online 15 Dec. 2004.*
*For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0275-1203.*

## 2 Background and Definitions

### 2.1 Parallel-Prefix Addition

Assume that $A = a_{n-1}a_{n-2}\ldots a_0$ and $B = b_{n-1}b_{n-2}\ldots b_0$ represent the two numbers to be added and $S = s_{n-1}s_{n-2}\ldots s_0$ denotes their sum. An adder can be considered as a three stage circuit. The preprocessing stage computes the carry-generate bits $g_i$, the carry-propagate bits $p_i$, and the half-sum bits $d_i$, for every $i$, $0 \leq i \leq n-1$, according to: $g_i = a_i \cdot b_i$, $p_i = a_i + b_i$, and $d_i = a_i \oplus b_i$, where $\cdot$, $+$, and $\oplus$ denote the logical AND, OR and exclusive-OR operations, respectively. The second stage of the adder computes the carry signals $c_i$ using the carry generate and propagate bits $g_i$ and $p_i$, while the final stage computes the sum bits according to, $s_i = d_i \oplus c_{i-1}$.

A parallel-prefix circuit with $n$ inputs $x_1, x_2, \ldots, x_n$ computes, in parallel, $n$ outputs $y_1, y_2, \ldots, y_n$ using an arbitrary associative operator $\odot$ as follows [2]: $y_1 = x_1$, $y_2 = x_1 \odot x_2$, $y_3 = x_1 \odot x_2 \odot x_3$, $\ldots$, $y_n = x_1 \odot x_2 \odot \cdots \odot x_n$. Carry computation can be transformed to a prefix problem [6] using the associative operator $\circ$, which associates pairs of generate and propagate bits as follows: $(g,p) \circ (g',p') = (g + p \cdot g', p \cdot p')$.

In a series of consecutive associations of generate and propagate pairs $(g,p)$, the notation $(G_{k:j}, P_{k:j})$ is used to denote the group generate and propagate term produced out of bits $k, k-1, \ldots, j$, that is,

$$(G_{k:j}, P_{k:j}) = (g_k, p_k) \circ (g_{k-1}, p_{k-1}) \circ \ldots \circ (g_{j+1}, p_{j+1}) \circ (g_j, p_j). \quad (1)$$

Following the above definitions, each carry $c_i$ is equal to $G_{i:0}$.

The prefix operator $\circ$ is idempotent, i.e., $(g,p) \circ (g,p) = (g,p)$. The generalization of the idempotency property [20] allows a group term $(G_{i:j}, P_{i:j})$ to be derived by the association of two overlapping terms, $(G_{i:k}, P_{i:k})$ and $(G_{m:j}, P_{m:j})$, with $i > m \geq k > j$, since

$$(G_{i:j}, P_{i:j}) = (G_{i:k}, P_{i:k}) \circ (G_{m:j}, P_{m:j}). \quad (2)$$

Representing the operator $\circ$ as a node ● and the signal pairs $(G_{i:j}, P_{i:j})$ as the edges of a graph, parallel-prefix carry-computation units can be represented as directed acyclic graphs. Fig. 1 presents the 8-bit parallel-prefix adders, proposed by Kogge and Stone [4], Ladner and Fisher [5], and one representative of the Knowles' adders [8]. The logic-level implementation of the basic cells used in a parallel-prefix adder is shown in Fig. 2, while white nodes ○ are buffering nodes. The last node ● of each bit column requires a simpler implementation (one AND-OR gate) since only a group generate term of the form $G_{i:0}$ needs to be produced.

### 2.2 Ling Adders

Ling proposed a simplified form of carry lookahead equations that rely on adjacent bit pairs $(a_i, b_i)$ and $(a_{i-1}, b_{i-1})$. The $i$th Ling carry

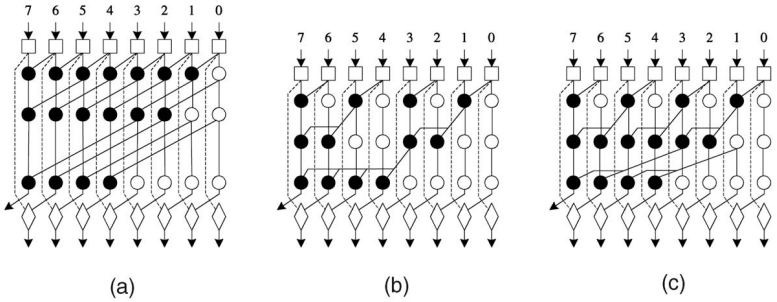


Fig. 1. The (a) Kogge-Stone, (b) Ladner-Fischer, and (c) one representative of Knowles' adders.

$H_i$ was defined in [11] as $H_i = c_i + c_{i-1}$. In this way, each $H_i$ can be expressed as

$$H_i = g_i + g_{i-1} + p_{i-1} \cdot g_{i-2} + \ldots + p_{i-1} \cdot p_{i-2} \cdot \ldots \cdot p_1 \cdot g_0. \quad (3)$$

The Ling carries $H_i$ can be computed faster than the corresponding carries $c_i$ since they rely on a simpler Boolean function. Consider, for example, the case of $c_3$ and $H_3$

$$c_3 = g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0$$
$$H_3 = g_3 + g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0.$$

Assuming the use of two-input logic gates, the calculation of $c_3$ requires four logic levels for the fastest implementation, while, for $H_3$, only three logic levels suffice.

Although the computation of the bits $H_i$ is simpler, the derivation of the final sum bits $s_i$ using the Ling carries is complicated compared to the case where the traditional carries are used, i.e., $s_i = d_i \oplus c_{i-1}$. Since $c_i = p_i \cdot H_i$, it holds that $s_i = d_i \oplus c_{i-1} = d_i \oplus (p_{i-1} \cdot H_{i-1})$. According to [13], the computation of the bits $s_i$ can be transformed as follows:

$$s_i = \overline{H}_{i-1} \cdot d_i + H_{i-1} \cdot (d_i \oplus p_{i-1}), \quad (4)$$

which can be implemented using a multiplexer that selects either $d_i$ or $(d_i \oplus p_{i-1})$ according to the value of $H_{i-1}$. The notation $\overline{x}$ denotes the complement of bit $x$. Taking into account that, in general, an XOR gate is of almost equal delay to a multiplexer and that both $d_i$ and $(d_i \oplus p_{i-1})$ are computed in fewer logic levels than $H_{i-1}$, then no extra delay is imposed by the use of Ling carries for the computation of the sum bits $s_i$. In fact, the sum bits are computed faster because of the faster computation of Ling carries.

## 3 Parallel-Prefix Formulation of Ling Addition

In the following, we will present a systematic methodology that allows the parallel-prefix computation of Ling carries. In order to describe the proposed approach, at first an 8-bit adder will be used as an example. The Ling carries at the fourth and the fifth bit positions are equal to

$$H_4 = g_4 + g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0, \quad (5)$$

$$H_5 = g_5 + g_4 + p_4 \cdot g_3 + p_4 \cdot p_3 \cdot g_2 + p_4 \cdot p_3 \cdot p_2 \cdot g_1 + p_4 \cdot p_3 \cdot p_2 \cdot p_1 \cdot g_0. \quad (6)$$

Since $g_i \cdot p_i = g_i$, then (5) and (6) can be written as

$$H_4 = g_4 + g_3 + p_3 \cdot p_2 \cdot (g_2 + g_1) + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot g_0, \quad (7)$$

$$H_5 = g_5 + g_4 + p_4 \cdot p_3 \cdot (g_3 + g_2) + p_4 \cdot p_3 \cdot p_2 \cdot p_1 \cdot (g_1 + g_0). \quad (8)$$

Assuming that

$$G_i^* = g_i + g_{i-1} \qquad \text{and} \qquad P_i^* = p_i \cdot p_{i-1}, \quad (9)$$

$0 \le i \le n-1$, with $g_{-1} = p_{-1} = 0$, $G_k^* = P_k^* = 0$, for $k < 0$, then (7), (8) can be expressed as

$$H_4 = G_4^* + P_3^* \cdot G_2^* + P_3^* \cdot P_1^* \cdot G_0^* \quad (10)$$

$$H_5 = G_5^* + P_4^* \cdot G_3^* + P_4^* \cdot P_2^* \cdot G_1^*. \quad (11)$$

Equations (10) and (11) can be written, using the $\circ$ operator, as

$$H_4 = (G_4^*, P_3^*) \circ (G_2^*, P_1^*) \circ (G_0^*, P_{-1}^*)$$
$$H_5 = (G_5^*, P_4^*) \circ (G_3^*, P_2^*) \circ (G_1^*, P_0^*).$$

Therefore, by using the intermediate generate and propagate pairs $(G_i^*, P_{i-1}^*)$ and by treating separately the Ling carries of the even and the odd-indexed bit positions, each carry $H_i$, in the case of an 8-bit adder, can be derived using the operator $\circ$ as follows:

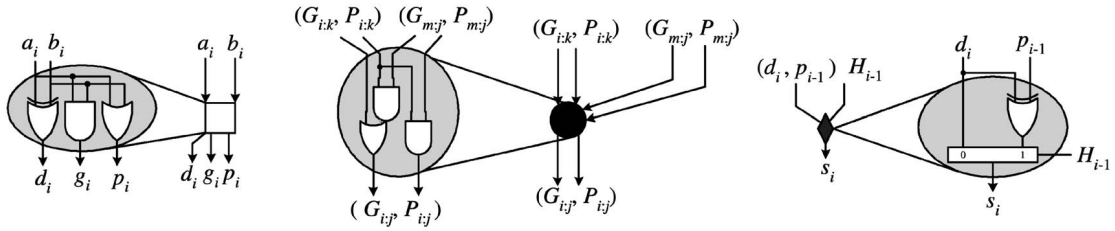


Fig. 2. The logic-level implementation of the basic cells used in parallel-prefix carry computation.

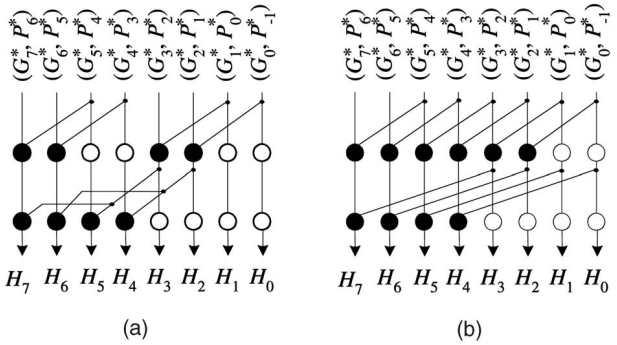


Fig. 3. The (a) Lander-Fischer and (b) Kogge-Stone parallel-prefix structure, using the pairs $(G_i^*, P_{i-1}^*)$, for the computation of the Ling carries, in the case of an 8-bit adder.

$$
\begin{aligned}
H_0 &= (G_0^*, P_{-1}^*),\\
H_2 &= (G_2^*, P_1^*) \circ (G_0^*, P_{-1}^*),\\
H_4 &= (G_4^*, P_3^*) \circ (G_2^*, P_1^*) \circ (G_0^*, P_{-1}^*),\\
H_6 &= (G_6^*, P_5^*) \circ (G_4^*, P_3^*) \circ (G_2^*, P_1^*) \circ (G_0^*, P_{-1}^*),\\
H_1 &= (G_1^*, P_0^*),\\
H_3 &= (G_3^*, P_2^*) \circ (G_1^*, P_0^*),\\
H_5 &= (G_5^*, P_4^*) \circ (G_3^*, P_2^*) \circ (G_1^*, P_0^*),\\
H_7 &= (G_7^*, P_6^*) \circ (G_5^*, P_4^*) \circ (G_3^*, P_2^*) \circ (G_1^*, P_0^*).
\end{aligned}
$$

This formulation allows the parallel-prefix computation of the Ling carries $H_i$ using separate prefix trees for the even and the odd-indexed bit positions. As shown in Fig. 3, after the generation of the pairs $(G_i^*, P_{i-1}^*)$, at most two prefix levels are required for the computation of each $H_i$.

Based on the form of the equations that compute the Ling carries $H_i$, in the case of the 8-bit adder, the following observations can be made: The new preprocessing stage that computes the pairs $(G_i^*, P_{i-1}^*)$ requires one extra logic level compared to the logic-levels needed to derive the bits $(g, p)$ in the traditional case. However, the number of terms $(G_i^*, P_{i-1}^*)$ that need to be associated is reduced to half compared to the traditional approach, where the pairs $(g, p)$ are used. Therefore, one less prefix level (two logic levels) is required for the computation of the bits $H_i$, which leads to a reduction by one logic level in total. Also, the terms $H_i$ of the odd and the even-indexed bit positions are computed independently, thus directly reducing the fanout of the parallel-prefix structure, a fact that also contributes to the reduction of the delay.

It can be easily proven by induction that, in case of an $n$-bit adder, the Ling carries $H_i$ and $H_{i+1}$ of consecutive even and odd bit positions $i$ and $i+1$, respectively, are given by

$$H_i = (G_i^*, P_{i-1}^*) \circ (G_{i-2}^*, P_{i-3}^*) \circ \ldots \circ (G_0^*, P_{-1}^*) \tag{12}$$

$$H_{i+1} = (G_{i+1}^*, P_i^*) \circ (G_{i-1}^*, P_{i-2}^*) \circ \ldots \circ (G_1^*, P_0^*). \tag{13}$$

The design of parallel-prefix Ling adders is summarized in the following steps:

- Generate the intermediate generate and propagate pairs $(G_i^*, P_{i-1}^*)$ either by combining the carry generate bits $g_i$ and the carry propagate bits $p_i$ according to (9) (Fig. 4a) or directly from the input bits $(a_i, b_i)$ and $(a_{i-1}, b_{i-1})$ using AND-OR and OR-AND gates that implement the equations $G_i^* = (a_i \cdot b_i) + (a_{i-1} \cdot b_{i-1})$ and $P_i^* = (a_i + b_i) \cdot (a_{i-1} + b_{i-1})$, respectively. The second method reduces the number of in-series transistors that appear on the critical path.
- Using the pairs $(G_i^*, P_{i-1}^*)$, produce two separate prefix-trees, one for the even and one for the odd-indexed bit positions that compute the Ling carries $H_i$ and $H_{i+1}$. Any parallel-prefix structure can be employed for the generation of the bits $H_i$, in $\log_2 n - 1$ prefix levels.
- Derive the sum bits $s_i$ according to (4). The ♦ cell that implements (4) is shown in Fig. 4b. The Ling carry $H_{n-1}$ produced from the most-significant bit position does not represent a valid carry output. In order to get the carry-out $c_{n-1}$, one extra AND gate should be added that computes $c_{n-1} = p_{n-1} \cdot H_{n-1}$, without affecting the critical path.

In Fig. 5, several architectures for the case of a 16-bit adder are presented. It can be easily verified that the proposed adders maintain all the benefits of the parallel-prefix structures, while, at the same time, they offer reduced delay and fanout requirements. Since there is no interference between the prefix trees of the even and the odd bit positions, separate prefix architectures can be used for each one of them. For example, the last prefix structure of Fig. 5 uses the Ladner-Fischer approach for the even bit positions and the Kogge-Stone structure for the odd bit positions.

The design of any parallel-prefix structure when its width $n$ is not a power-of-two is based on the idempotency property presented in [20] (2). We prove that the idempotency property is valid even if the intermediate pairs $(G_i^*, P_{i-1}^*)$ are used instead of the pairs $(g, p)$. Therefore, idempotency can also be employed in the case of the proposed Ling adders. Assume that the notation $(G_{i:j}^*, P_{i:j}^*)$ is used to denote a group term produced out of associations of consecutive intermediate generate and propagate pairs $(G_k^*, P_{k-1}^*)$ and is defined as

$$(G_{i:j}^*, P_{i:j}^*) = (G_i^*, P_{i-1}^*) \circ (G_{i-2}^*, P_{i-3}^*) \circ \ldots \circ (G_j^*, P_{j-1}^*), \tag{14}$$

where $i, j$ are both either odd or even numbers. The idempotency property for the proposed Ling adders is defined as follows:

**Theorem 1.** *If $i$, $m$, $k$, and $j$ are all either odd or even integers and $i > m \geq k > j$, then $(G_{i:j}^*, P_{i:j}^*) = (G_{i:k}^*, P_{i:k}^*) \circ (G_{m:j}^*, P_{m:j}^*)$.*

**Proof.** Since $m \geq k$ and $k > j$, it holds that

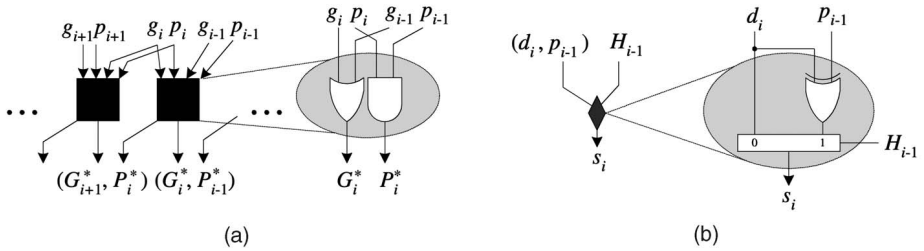


Fig. 4. The generation of the intermediate generate and propagate pairs $(G_i^*, P_{i-1}^*)$ and the new cell used for the computation of the sum bits in the case of a Ling adder.

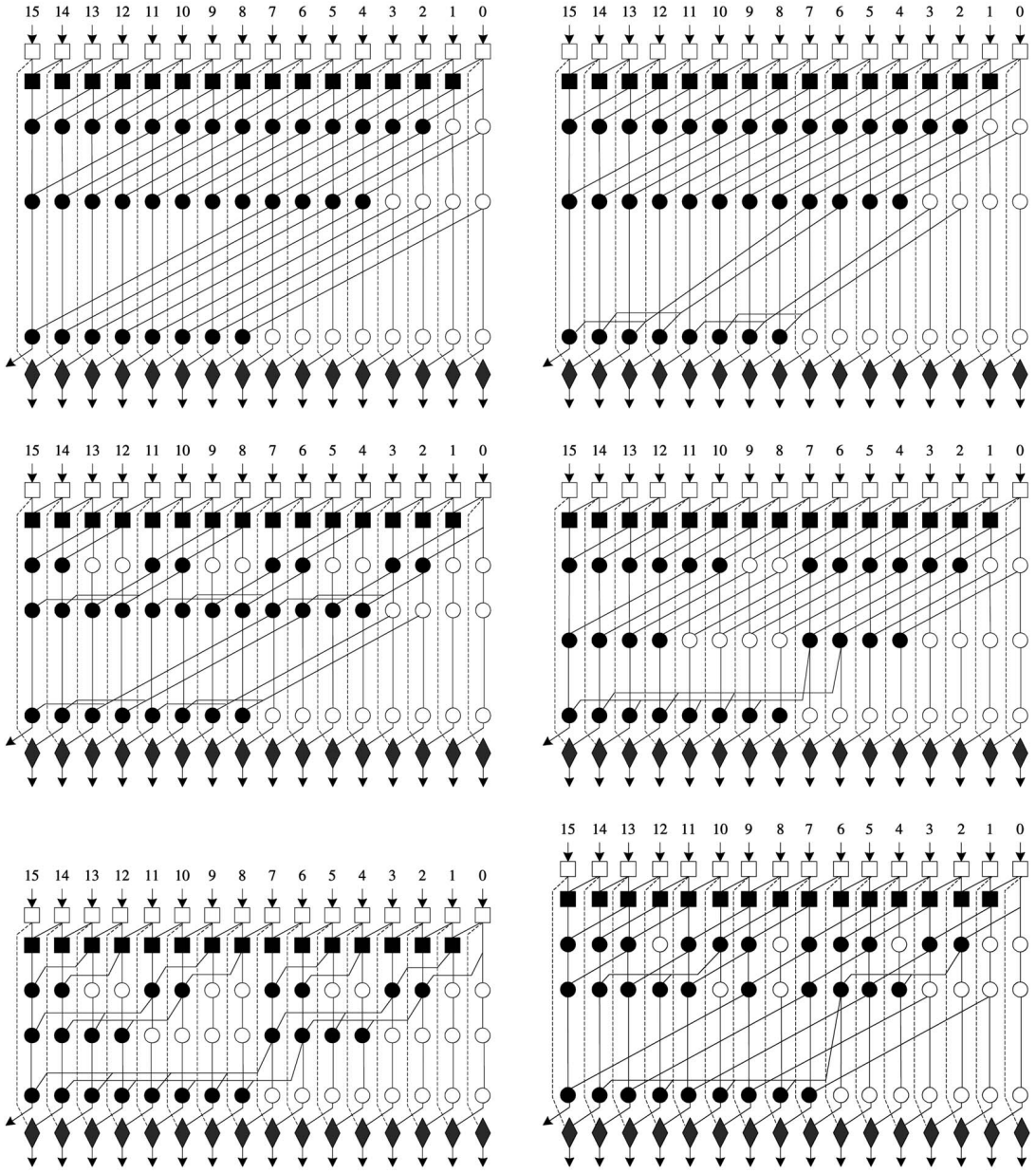


Fig. 5. Sixteen-bit minimum depth parallel-prefix Ling adders.

$$
\begin{aligned}
&(G^*_{i:k}, P^*_{i:k}) \circ (G^*_{m:j}, P^*_{m:j}) = \\
&= [(G^*_{i:m-2}, P^*_{i:m-2}) \circ (G^*_{m:k}, P^*_{m:k})] \circ [(G^*_{m:k}, P^*_{m:k}) \circ (G^*_{k-2:j}, P^*_{k-2:j})] \\
&= (G^*_{i:m-2}, P^*_{i:m-2}) \circ (G^*_{m:k} + P^*_{m:k} \cdot G^*_{m:k}, P^*_{m:k} \cdot P^*_{m:k}) \circ (G^*_{k-2:j}, P^*_{k-2:j}) \\
&= (G^*_{i:m-2}, P^*_{i:m-2}) \circ (G^*_{m:k}, P^*_{m:k}) \circ (G^*_{k-2:j}, P^*_{k-2:j}) \\
&= (G^*_{i:k}, P^*_{i:k}) \circ (G^*_{k-2:j}, P^*_{k-2:j}) = (G^*_{i:j}, P^*_{i:j}).
\end{aligned}
$$

□

Finally, we investigate ways for incorporating a carry-input signal to a Ling parallel-prefix structure. A discussion of the most efficient approaches for the traditional carries can be found in [21]. The carry-in bit can be included either by adding a fast carry increment stage or by treating $c_{\text{in}}$ as an extra bit of the preprocessing stage of the adder. The first case in shown in Fig. 6a. The second case can be derived by setting $g_{-1} = c_{\text{in}}$ and, according to (9), it follows that $G^*_{-1} = c_{\text{in}}$, $G^*_0 = g_0 + c_{\text{in}}$. Fig. 6b illustrates this approach for an 8-bit Ling adder.

### 3.1 Hybrid Parallel-Prefix/Carry-Select Ling Adders

The goal for high-speed adder architectures with reduced area and wiring has led to the design of hybrid parallel-prefix/carry-select adders. Fig. 7 illustrates a hybrid 32-bit adder which employs a Kogge-Stone parallel-prefix structure for the generation of the carries $c_{4k}$, $k = 1, 2, \ldots, n/4$, and 4-bit carry select blocks. The carry-select block computes two sets of sum bits, i.e., $s^0_i$, $s^1_i$, and the final sums are selected via a multiplexer according to the value of $c_{4k}$. The goal of such hybrid structures is to overlap the time required for the computation of the carries at the boundaries of the carry-select blocks with the time needed to derive the sum bits.

The design of hybrid parallel-prefix/carry-select Ling adders requires some minor modifications to the carry-select block. This is required since

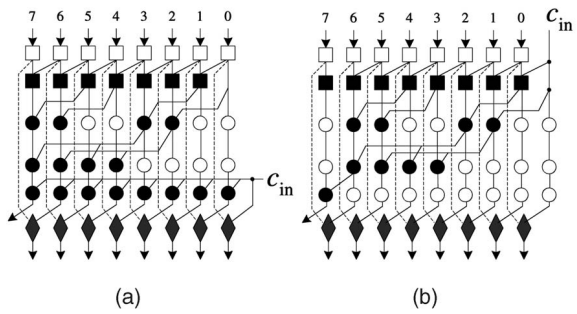


Fig. 6. Eight-bit Ling adders with a carry-in signal.

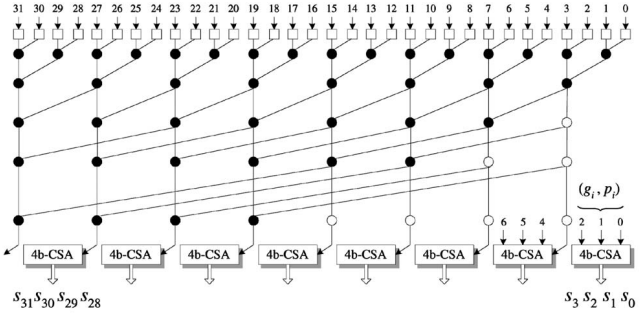


Fig. 7. Thirty-two-bit hybrid parallel-prefix/carry-select adder.

- The proposed prefix structures generate the Ling pseudo-carries $H_i$ instead of the real carries $c_i$ and, thus, a sum bit cannot be directly selected according to the value of $H_i$.
- The carries and the sum bits of the even and odd bit positions are generated separately.
- The carry-select blocks take as inputs the pairs $(G_i^*, P_{i-1}^*)$ and not the traditional $(g, p)$ pairs.

The equivalent 32-bit hybrid Ling adder is shown in Fig. 8. The Ling carries $H_{4k}$ and $H_{4k-1}$ are computed on the corresponding even and odd bit positions and used to select the final sum bits that have been concurrently produced by the 4-bit Modified Carry-Select Adders (MCSA). The design of the MCSA blocks will be explained via the following example.

Assume the case of the 4-bit MCSA that produces the sum bits $s_{30}$, $s_{28}$, $s_{26}$, and $s_{24}$ using as select signal the Ling carry $H_{23}$. For the sum bit $s_{28}$, it holds that

$$s_{28} = (p_{27} \cdot (G_{27}^* + P_{26}^* \cdot G_{25}^* + P_{26}^* \cdot P_{24}^* \cdot H_{23})) \oplus d_{28}.$$

According to the value of $H_{23}$, being 0 or 1, a set of two sum bits $\{s_{28}^0, s_{28}^1\}$ is derived

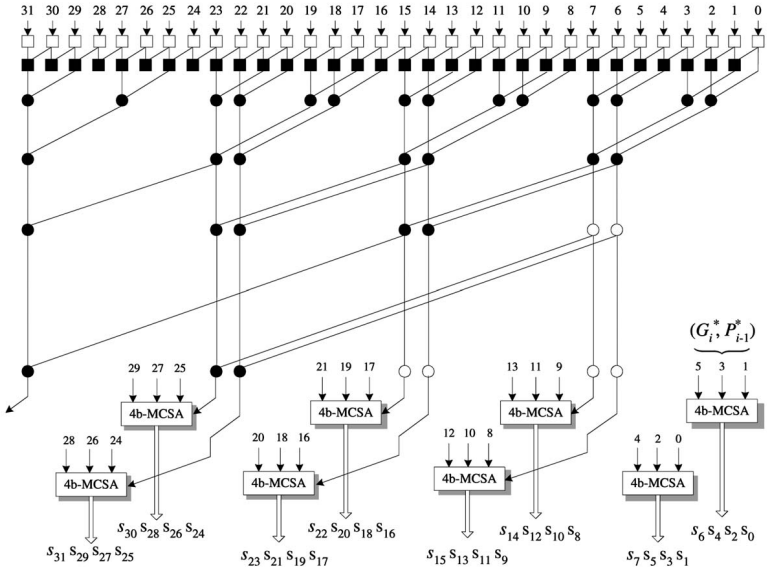


Fig. 8. A 32-bit hybrid parallel-prefix/carry-select Ling adder.

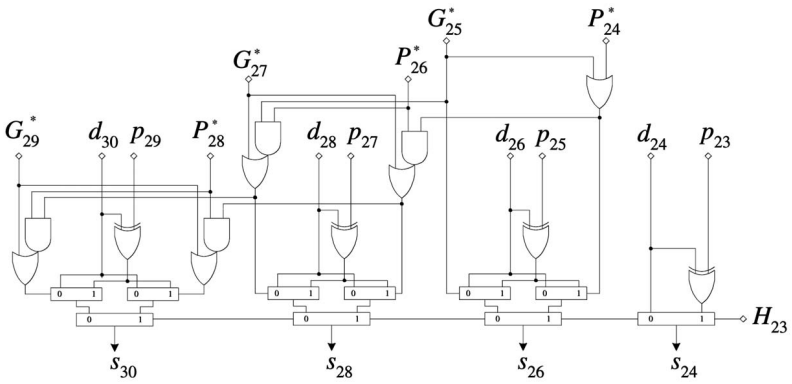


Fig. 9. The modified 4-bit carry-select block used for the design of hybrid Ling adders.

TABLE 1
The Area and Delay Estimates for the Traditional and the Proposed Ling Adders Both Using a Ladner-Fischer [5] and a Kogge-Stone [4] Parallel-Prefix Structure

| $n$ | Ladner-Fischer | | | | | Kogge-Stone | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Area ($\mu m^2$) | | Delay (ns) | | | Area ($\mu m^2$) | | Delay (ns) | | |
| | Norm. | Prop. | Norm. | Prop. | Saving | Norm. | Prop. | Norm. | Prop. | Saving |
| 16 | 4336 | 4928 | 0.65 | 0.56 | 13.8% | 5716 | 5755 | 0.62 | 0.53 | 14.5% |
| 32 | 10250 | 11038 | 0.79 | 0.68 | 13.9% | 13404 | 13561 | 0.76 | 0.66 | 13.1% |
| 64 | 26808 | 28132 | 0.98 | 0.85 | 13.2% | 33205 | 33904 | 0.89 | 0.80 | 10.1% |

$$
\begin{aligned}
s_{28}^0 &= [p_{27} \cdot (G_{27}^* + P_{26}^* \cdot G_{25}^*)] \oplus d_{28} \\
&= \overline{(G_{27}^* + P_{26}^* \cdot G_{25}^*)} \cdot d_{28} + (G_{27}^* + P_{26}^* \cdot G_{25}^*) \cdot (p_{27} \oplus d_{28}), \\
s_{28}^1 &= [p_{27} \cdot (G_{27}^* + P_{26}^* \cdot (G_{25}^* + P_{24}^*))] \oplus d_{28} \\
&= \overline{(G_{27}^* + P_{26}^* \cdot (G_{25}^* + P_{24}^*))} \cdot d_{28} \\
&\quad + (G_{27}^* + P_{26}^* \cdot (G_{25}^* + P_{24}^*)) \cdot (p_{27} \oplus d_{28}).
\end{aligned}
$$

Based on the above formulation, both $s_{28}^0$ and $s_{28}^1$ can be computed using two-input multiplexers that select $d_{28}$ or $(p_{27} \oplus d_{28})$ according to the value of the terms $(G_{27}^* + P_{26}^* \cdot G_{25}^*)$ and $(G_{27}^* + P_{26}^* \cdot (G_{25}^* + P_{24}^*))$, respectively. Finally, an additional multiplexer produces the correct value of $s_{28}$ using the incoming carry $H_{23}$, as shown in Fig. 9. Since the bits $\{s_{28}^0, s_{28}^1\}$ are computed earlier than $H_{23}$, the critical path remains in the carry computation unit and the delay of a multiplexer is added to it. The computation of the rest sum bits is performed using an equivalent formulation. Finally, the early derived less significant carries $H_6$, $H_7$, $H_{14}$, and $H_{15}$ are buffered so as to match the delay required for the computation of the prospective sum bits of the MCSAs and to equalize the paths of the circuit for achieving minimum delay [22].

## 4 Experimental Results

The proposed adders are compared against the parallel-prefix structures proposed by Ladner and Fischer [5] and Kogge and Stone [4] for the traditional definition of carry equations. The two architectures represent the two extremes of Knowles adders [8]. The results, for the rest of the structures that Knowles proposed, are expected to be between the area-efficient Ladner-Fischer structure and the high-speed approach proposed by Kogge and Stone.

Each adder was described in Verilog HDL and mapped on a $0.18\mu m$ technology library [23] under typical conditions (1.8V, 25°C), using the Synopsys Design Compiler v.2003.06. Each design was recursively optimized for speed targeting the minimum possible delay. Then, the derived netlists and the design constraints were passed to Cadence Silicon Ensemble v.5.3 in order to perform the final placement and routing of the design. All design constraints, such as output load, floorplan initialization information (each $n$-bit adder is placed in $2n$ fixed-height rows), and pin placement, were held constant for each architecture. Final timing analysis was performed using PrimeTime of Synopsys after all RC parasitic information was extracted from the layout and back-annotated to the gate-level netlist. It should be noted that the proposed adders utilize the AND-OR and OR-AND complex gates of the library for the generation of the pairs $(G_i^*, P_{i-1}^*)$.

TABLE 2
The Area and Delay Estimates for 64-Bit Hybrid Adders Assuming 4-Bit Carry-Select Blocks

| 64-bit hybrid adders | Area ($\mu m^2$) | Delay (ns) |
|---|---|---|
| Ladner-Fischer Normal | 21228 | 0.91 |
| Ladner-Fischer Proposed | 23654 | 0.83 |
| Kogge-Stone Normal | 26754 | 0.89 |
| Kogge-Stone Proposed | 28385 | 0.81 |

The first part of Table 1 presents the area and delay estimates for the traditional and the proposed Ling adders, both using a Ladner-Fischer [5] parallel-prefix structure. The proposed adders have the minimum propagation delay in all examined cases. The proposed adders outperform the traditional Ladner-Fischer adders due to the half fanout requirements and the one-less logic level of implementation. When both the adders that implement the traditional carry-lookahead equations and the ones that compute the Ling carries have equal fanout, the proposed adders are faster in all cases, as shown in the second part of Table 1. The average delay reduction achieved for both parallel-prefix architectures is 13.1 percent.

Also, the traditional and the proposed hybrid adders are compared using a Kogge-Stone and a Ladner-Fischer parallel-prefix tree for the computation of the carries at the boundaries of the carry-select blocks. The experimental results gathered for the 64-bit hybrid adders are shown in Table 2. Again, the proposed adders are faster than the corresponding traditional hybrid adders by 8.8 percent.

For completeness, the adders proposed in [19] have been implemented using a Ladner-Fischer and a Kogge-Stone parallel-prefix structure. The results obtained are shown in Table 3. The proposed adders require two less logic levels than the adders of [19] and, according to Table 3, are faster by 13.8 percent on average. Finally, we remind the reader that the methodology presented in [19] cannot be applied to the design of hybrid parallel-prefix/carry-select adders.

## 5 Conclusions

A systematic methodology for designing parallel-prefix Ling adders has been introduced in this paper. The proposed adders preserve all the benefits of the traditional parallel-prefix carry-computation units, while, at the same time, offering reduced delay and fanout requirements. Hence, high-speed datapaths of modern microprocessors can truly benefit from the adoption of the proposed adder architecture.

## Acknowledgments

G. Dimitrakopoulos has been supported by the "D. Maritsas" Graduate Scholarship.

## References

[1] I. Koren, *Computer Arithmetic Algorithms.* A.K. Peters, Ltd., 2002.
[2] B. Parhami, *Computer Arithmetic—Algorithms and Hardware Designs.* Oxford Univ. Press, 2000.

TABLE 3
The Area and Delay Estimates for the Adders Proposed in [19] and the Proposed Ling Adders Both Using a Ladner-Fischer [5] and a Kogge-Stone [4] Parallel-Prefix Structure

| $n$ | Ladner-Fischer | | | | | Kogge-Stone | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Area ($\mu m^2$) | | Delay (ns) | | | Area ($\mu m^2$) | | Delay (ns) | | |
| | [19] | Prop. | [19] | Prop. | Saving | [19] | Prop. | [19] | Prop. | Saving |
| 16 | 4730 | 4928 | 0.67 | 0.56 | 16.4% | 4770 | 5755 | 0.63 | 0.53 | 15.8% |
| 32 | 10289 | 11038 | 0.80 | 0.68 | 15.0% | 11432 | 13561 | 0.77 | 0.66 | 14.2% |
| 64 | 27596 | 28132 | 0.95 | 0.85 | 10.5% | 29962 | 33904 | 0.90 | 0.80 | 11.1% |


[3] M. Ergecovac and T. Lang, *Digital Arithmetic.* Morgan-Kauffman, 2003.
[4] P.M. Kogge and H.S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Trans. Computers,* vol. 22, no. 8, pp. 786-792, Aug. 1973.
[5] R.E. Ladner and M.J. Fisher, "Parallel Prefix Computation," *J. ACM,* vol. 27, no. 4, pp. 831-838, Oct. 1980.
[6] R.P. Brent and H.T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. Computers,* vol. 31, no. 3, pp. 260-264, Mar. 1982.
[7] T. Han and D. Carlson, "Fast Area-Efficient VLSI Adders," *Proc. Symp. Computer Arithmetic,* pp. 49-56, May 1987.
[8] S. Knowles, "A Family of Adders," *Proc. 14th Symp. Computer Arithmetic,* pp. 30-34, Apr. 1999. Reprinted in ARITH-15, pp. 277-281.
[9] A. Beaumont-Smith and C.C. Lim, "Parallel-Prefix Adder Design," *Proc. 15th Symp. Computer Arithmetic,* pp. 218-225, June 2001.
[10] V.G. Oklobdzija et al., "Energy-Delay Estimation Technique for High-Performance Microprocessor VLSI Adders," *Proc. 16th Symp. Computer Arithmetic,* pp. 15-22, June 2003.
[11] H. Ling, "High-Speed Binary Adder," *IBM J. R&D,* vol. 25, pp. 156-166, May 1981.
[12] R.W. Doran, "Variants of an Improved Carry-Lookahead Adder," *IEEE Trans. Computers,* vol. 37, pp. 1110-1113, 1988.
[13] S. Vassiliadis, "Recursive Equations for Hardwired Binary Adders," *J. Electronics,* vol. 67, no. 2, pp. 201-213, Aug. 1989.
[14] N.T. Quach and M.J. Flynn, "High-Speed Addition in CMOS," *IEEE Trans. Computers,* vol. 41, no. 12, pp. 1612-1615, Dec. 1992.
[15] S. Naffziger, "A Sub-Nanosecond 0.5$mu$m 64b Adder Design," *Proc. IEEE Solid-State Circuits Conf.,* pp. 362-363, Feb. 1996.
[16] D. Phatak and I. Koren, "Intermediate Variable Encodings that Enable Multiplexor Based Implementations of Two Operand Addition," *Proc. Symp. Computer Arithmetic,* pp. 22-29, Apr. 1999.
[17] O. Kwon, E. Swartzlander, and K. Nowka, "A Fast Hybrid Carry-Lookahead/Carry-Select Adder Design," *Proc. Great Lakes Symp. VLSI,* pp. 149-152, Apr. 2001.
[18] Y. Wang, C. Pai, and X. Song, "The Design of Hybrid Carry-Lookahead/Carry-Select Adders," *IEEE Trans. Circuits and Systems II,* vol. 49, no. 1, Jan. 2002.
[19] C. Efstathiou, H.T. Vergos, and D. Nikolos, "Ling Adders in Standard CMOS Technologies," *Proc. IEEE Int'l Conf. Electronics, Circuits, and Systems (ICECS),* vol. 2, pp. 485-488, Sept. 2002.
[20] T. Lynch and E. Swartzlander, "A Spanning Tree Carry Lookahead Adder," *IEEE Trans. Computers,* vol. 41, no. 8, pp. 931-939, Aug. 1992.
[21] A. Goldovsky et al., "A 1.0-nsec 32-bit Prefix Tree Adder in 0. 25-$mu$m Static CMOS," *Proc. Midwest Symp. Circuits and Systems,* vol. 2, pp. 608-612, Aug. 1999.
[22] I. Sutherland, R. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits.* Morgan Kaufmann, 1999.
[23] UMC-18, *eSi-Route/11 0.8 Standard Cell Library,* Virtual Silicon Technology, Jan. 2001.