IET Circuits, Devices & Systems

Research Article

New energy-efficient hybrid wide-operand adder architecture

ISSN 1751-858X
Received on 13th March 2019
Revised 22nd July 2019
Accepted on 6th August 2019
E-First on 12th November 2019
doi: 10.1049/iet-cds.2019.0084
www.ietdl.org

Fereshteh Jafarzadehpour[1], Amir Sabbagh Molahosseini[1] ✉, Azadeh Alsadat Emrani Zarandi[2], Leonel Sousa[3]

[1]Department of Computer Engineering, Kerman Branch, Islamic Azad University, Kerman, Iran
[2]Department of Computer Engineering, Shahid Bahonar University of Kerman, Kerman, Iran
[3]INESC-ID, Instituto Superior Tecnico (IST), University of Lisbon, Lisbon, Portugal
✉ *E-mail: amir@iauk.ac.ir*

Abstract: The consumer electronics markets have increased the demand for high-speed and low-power adders with large operands to be integrated in modern portable systems. Traditional fast adder architectures, such as parallel-prefix adders, exhibit high-power consumption for large operands. The hybrid design is one of the most promising techniques to achieve a trade-off between the delay and power-consumption for the addition of large operands. This study presents a new hybrid adder architecture, specifically designed for large operands, based on the premise that in large parallel-prefix adders the least-significant carries are produced much sooner than the most-significant ones. Therefore, the authors avoid the incorporation of fast architectures related with the application of carries to the final summation least-significant bits, with no impact on the critical path. This leads to a reduction in the area of the summation blocks in the least-significant positions without compromising the speed. Moreover, the complement of the carries is generated and propagated inside the carry network of the proposed adder in order to decrease the delay. VLSI implementation results on the 65-nm-TSMC technology show that the proposed adder achieves >25% of energy savings, and a reduction of over 30% of the area-delay-product in comparison with state-of-the-art wide-operand adders.

1 Introduction

Adders are the most important building blocks in modern processors, since they are in the critical delay path and support other arithmetic operations, such as multiplication and division. Furthermore, the processor datapath consumes over 30% of the power and adders are the most used components in it [1]. There are three main levels of investigation to improve the efficiency of adders, namely: (i) the adoption of unconventional number systems [2] such as the Residue Number System [3] that limit carry-propagation, (ii) the design of efficient multi-bit adder architectures [4] such as parallel-prefix adders [5], and (iii) the improvement of the design of full adders (FAs) which are a core part of many adder architectures [1, 6].

Up to now, distinct structures have been introduced for multi-bit binary adders. The simplest and low-cost structure is the ripple carry adder (RCA) [7, 8]. However, the high delay of the RCA structure makes it inefficient for modern applications. Other structures like the carry select (CSL) [9–11], carry skip (CSK) [12–15], and carry-look ahead (CLA) adders [16–18] are faster than RCAs at the expense of increased area and power consumption. Moreover, parallel prefix (PPF) adders are fast structures composed of three main components: (i) the preparation unit that produces generate and propagate signals; (ii) the carry network that creates and propagates carry bits; and (iii) the last component that computes the value of the sum based on the bitwise XORing of the carry bits (the output of ii), and the two input bits. Based on the cell types, fan-out and depth of the tree structures, different structures of this class have been suggested. One of these structures is the Kogge-Stone [19], wherein all the nodes of the structure have a fan-out of two. This structure has the minimum depth among the structures with fan-out of two, but requires the highest area. Other structures with lower number of nodes are the Brent-Kung [20], the Lander-Fisher [21], the Han-Carlson [22], and the Sklansky [23].

During recent years, due to increasing amounts of processed data, and stricter requirements pertaining precision and speed, in particular for cryptography, fast adders with large operands have been intensively investigated [24–27]. In order to increase the performance of the addition of large numbers, hybrid adders have been suggested, since the regular horizontal extension of conventional architectures such as parallel-prefix and carry-select adders leads to excessive power-consumption. In this direction, recently, 64 and 128-bit adders using PPF, CSL, and CSK adders have been presented in [28], while in [29], ripple carry and CLA adder structures were combined. The adder structures of [30] use carry save adders and CLA, and, in [31], CSL is applied in conjunction with the Kogge-Stone structure.

In the designs proposed herein, the PPF structure is adopted for the generation and propagation of the *complement of carries*. Two different types of blocks, designated *sum producer type 1* and *sum producer type 2*, are applied in the proposed hybrid adders. There is no propagation of carry in these two types of sum producer blocks. The *sum producer type 1* is designed for the situations where the complement of the input carry bit becomes ready sooner, resulting in a significant area reduction. Moreover, in order to increase the speed, AND/OR gates in the critical delay path of PPF structures are replaced by NOR/NAND gates.

This paper is organised as follows. Section 2 reviews the PPF adders briefly. The structures of the proposed adders and performance evaluation are presented in the third and fourth sections, respectively. Finally, Section 5 concludes the paper.

2 Background

The first stage of a PPF adder, named the preparation unit, produces generate (g), alive (a), and propagate (p) signals. Let $X = x_{n-1}x_{n-2}\dots x_0$ and $Y = y_{n-1}y_{n-2}\dots y_0$ be the two operands. The g, a, and p signals for each bit position can be calculated as follows [4]:

$$g_i = x_i \cdot y_i \quad (1)$$

$$a_i = x_i + y_i \quad (2)$$

$$p_i = x_i \oplus y_i \tag{3}$$

In the above formulas, i is the bit position, and ‘ . ’, ‘ + ’, and ‘$\oplus$’ denote bitwise AND, OR, and exclusive-OR operations, respectively. The second stage accommodates the main network, a tree structure, of parallel prefix adders. The associative operator o is applied to associate pairs of consecutive generate and propagate bits as follows: $(g_{i+1},\ a_{i+1})o(g_i,\ a_i) = (g_{i+1} + g_i \cdot a_{i+1},\ a_{i+1} \cdot a_i)$. The notation $(G_{i:j},\ A_{i:j})$ is used when the o operator is applied to a series of consecutive pairs of generate and propagate bits from the j-bit to ith bit positions ($j < i$):

$$(G_{i:j},\ A_{i:j}) = (g_i,\ a_i)o(g_{i-1},\ a_{i-1})o\ldots o(g_{j+1},\ a_{j+1})o(g_j,\ a_j) \tag{4}$$

By using the idempotency property of the o operator [32], two groups, $(G_{i:k'+1},\ A_{i:k'+1})$ and $(G_{k':j},\ A_{k':j})$, with $i > k' \geq j$, can be associated as follows:

$$(G_{i:j}, A_{i:j}) = (G_{i:k'+1}, A_{i:k'+1})o(G_{k':j}, A_{k':j}) \tag{5}$$

Through the iterative application of (4) and (5) in the second stage of the PPF adder, the carry-out signal of any i bit-position (C_{i+1}) can be obtained as ($C_{i+1} = G_{i:0}$).

In the last stage of the PPF adder (the sum production part), the value of the sum is produced from the input carries at each bit position (the outputs of the PPF tree structure) and the corresponding p signals. Based on the node numbers, the depth of the PPF, and the fan-out of each node, distinct topologies with different characteristics have been introduced for designing parallel prefix adders. As an example, the Brent-Kung structure [20] occupies the least area, while Sklansky [23] is the fastest one.

In order to increase the speed of PPFs, the depth of the tree structure should be reduced, which requires increasing the number of nodes and interconnections or the nodes' fan-outs. As a result, faster adder structures, like Kogge-Stone, consume more power and area in comparison with other similar structures. Fig. 1 shows the structure of an 8-bit Kogge-Stone adder.

PPFs are not efficient for large operands due to their high power-consumption. Therefore, hybrid PPF-based structures are usually used for wide adders. One of these efficient adders is presented in [28], which is composed of a parallel-prefix/carry select architecture and a skip adder (PPF/CSSA). This wide adder can improve the speed by balancing the delays of the PPF network and CSL/CSSA blocks. The structure of [28] for a 64-bit adder is shown in Fig. 2. It consists of three main types of blocks: pre-processing, PPF, and sum production blocks. In the sum production blocks, illustrated in Fig. 3, two 4-bit RCAs are used for each possible value of the input carry (0 or 1). The carry-in of the second RCA is produced directly from PPF signals, improving the speed of this 64-bit adder in comparison with adders of similar bit-width.

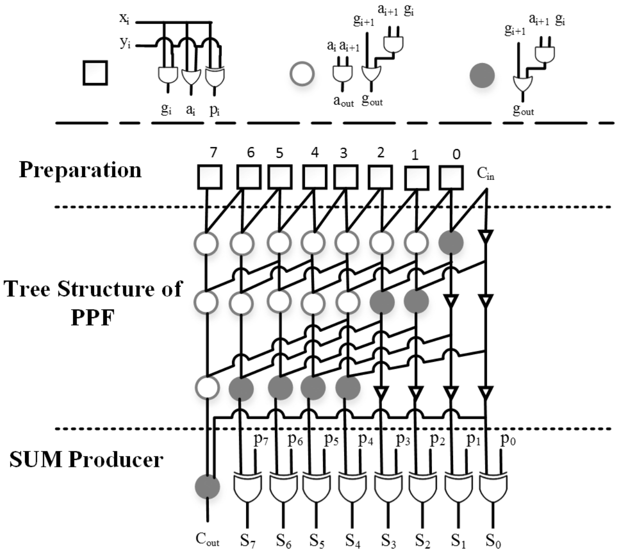


Fig. 1 *8-bit Kogge-Stone adder structure*

3 Proposed adder architectures

The following points are considered to improve the area and speed in the proposed design.

(i) In order to improve the area, the proposed design adopts two kinds of blocks for computing the sum. The *Sum Producer type 2* is a sort of carry select adder with a different structure. It is known that the delay of CSL adder is less than other adders and corresponds roughly to the delay of a single multiplexer (MUX). However, the high speed of the CSL adder is achieved with the cost of high area. Unlike the design of [28] that uses the same structure for all sum producer blocks, herein different sum producer blocks are used. For the places where the complement of the input carry bit, $\bar{c}_i$ becomes ready sooner, in order to reduce the area, the *Sum Producer type 1* is used, which has a simpler structure compared to the second type. Although the delay of the type 1 block is higher, the final addition results are obtained faster for these blocks in the proposed adder, since the complement of the carry-in bits affecting these blocks becomes ready sooner.

(ii) To reduce the delay of the proposed adder structure, the complement of the carry, $\bar{c}_i$ is generated and propagated instead of $c_i \cdot \bar{c}_i$ is considered instead of c_i to make use of the kill (k_i) signal ($k_i = \bar{x}_i \cdot \bar{y}_i$) and the complement of the generate ($\bar{g}_i$) instead of the generate (g_i) and alive (a_i) signals that are used in traditional PPF designs. k_i and $\bar{g}_i$ can be computed for each bit-position by using NOR and NAND gates. Therefore, the speed of the circuit is improved in the Preparation level compared to the usual PFF designs. Moreover, in the suggested adder, NAND/NOR gates are

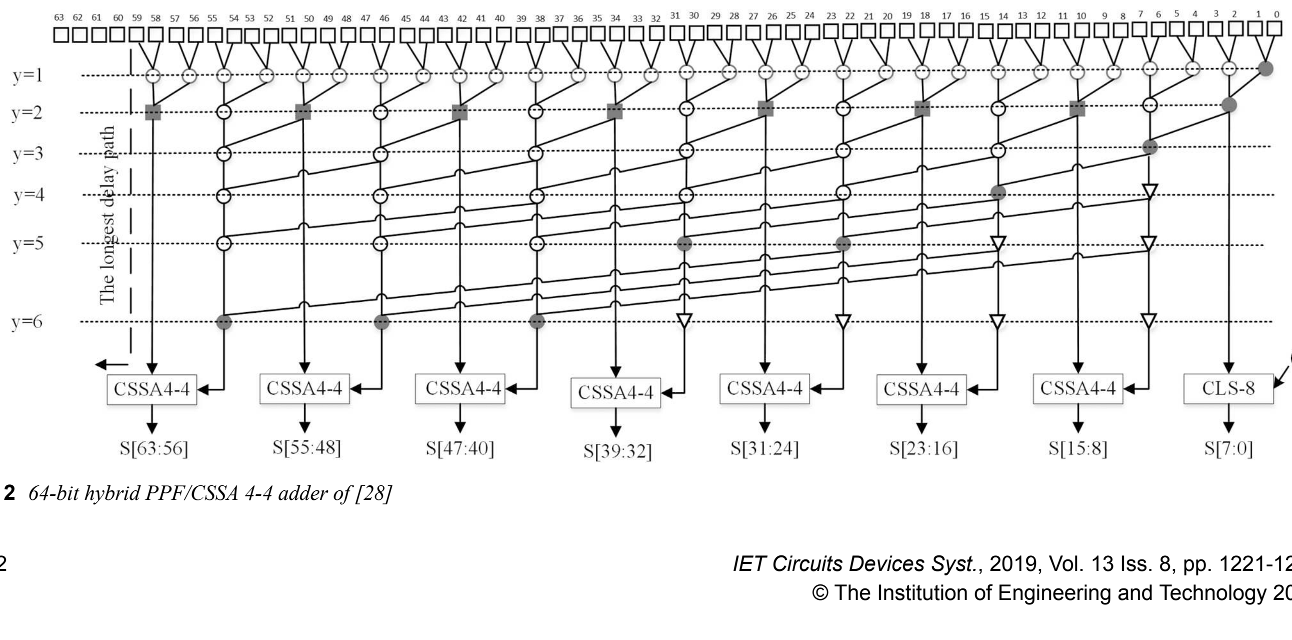


Fig. 2 *64-bit hybrid PPF/CSSA 4-4 adder of [28]*

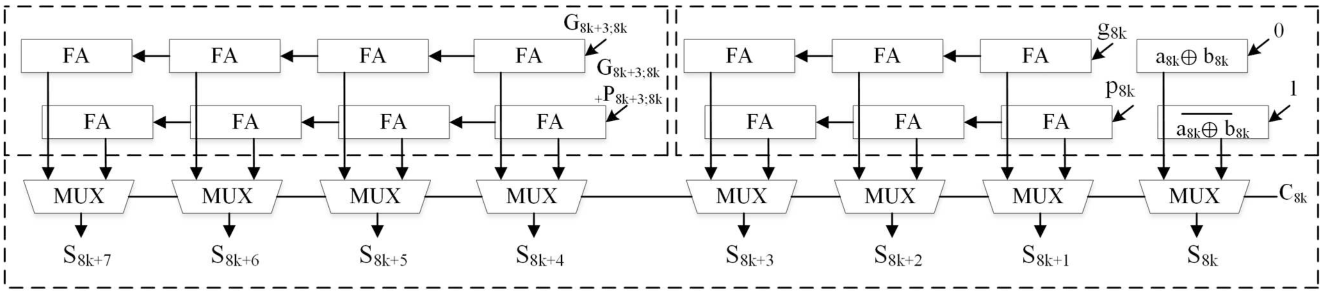
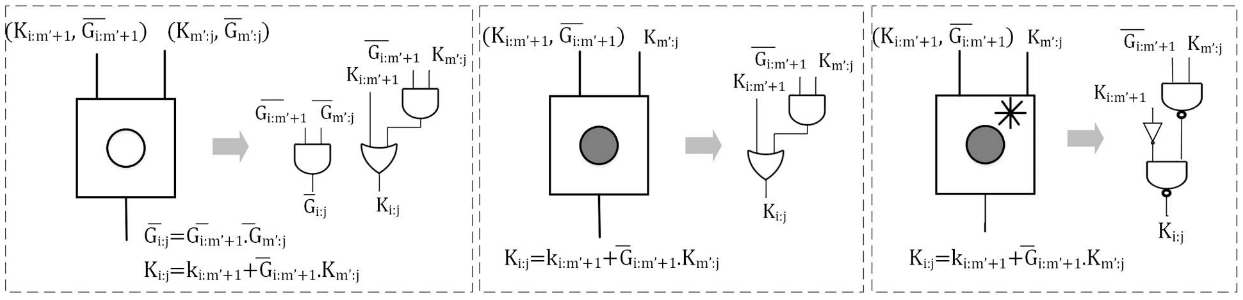


Fig. 3 *Structure of 4-4 bit CSSA of [28]*



$(K_{i:m'+1}, \bar{G}_{i:m'+1})$ $(K_{m':j}, \bar{G}_{m':j})$
$\bar{G}_{i:j}=\bar{G}_{i:m'+1}.\bar{G}_{m':j}$
$K_{i:j}=k_{i:m'+1}+\bar{G}_{i:m'+1}.K_{m':j}$
$K_{i:j}$
$\bar{G}_{i:j}$

Fig. 4 *Structure for the nodes in the proposed design*

used as much as possible instead of OR/AND gates, especially in the critical path.

Let $X = x_{n-1}x_{n-2}\dots x_0$ and $Y = y_{n-1}y_{n-2}\dots y_0$ be two input n-bit operands. The following formulae are used as the basis for the computation of the complemented values of the carries ($\bar{c}_i's$):

$$k_i = \bar{x}_i \cdot \bar{y}_i = \overline{x_i + y_i} \tag{6}$$

$$\bar{g}_i = \overline{x_i \cdot y_i} \tag{7}$$

Since $c_{i+1} = x_i \cdot y_i + c_i \,.\, (x_i + y_i)$, $\bar{c}_{i+1}$ can be calculated as follows:

$$\begin{aligned}\bar{c}_{i+1} &= \overline{x_i \cdot y_i + c_i \cdot (x_i + y_i)} = \overline{(x_i \cdot y_i)} \cdot \overline{(c_i \cdot (x_i + y_i))}\\ &= \overline{(x_i \cdot y_i)} \cdot \left(\overline{c_i} + \overline{(x_i + y_i)}\right)\\ &= \overline{(x_i \cdot y_i)} \cdot \overline{c_i} + \overline{(x_i \cdot y_i)} \cdot \overline{(x_i + y_i)} = \overline{(x_i \cdot y_i)} \cdot \overline{c_i}\\ &\quad +(\overline{x_i} + \overline{y_i}) \cdot (\overline{x_i} \cdot \overline{y_i}) = \overline{(x_i \cdot y_i)} \cdot \overline{c_i} + (\overline{x_i} \cdot \overline{y_i})\end{aligned}$$

Therefore:

$$\bar{c}_{i+1} = \bar{g}_i\bar{c}_i + k_i \tag{8}$$

Equations (6) to (8) can be interpreted as follows: whenever the value of x_i and y_i in any bit position becomes 0, the value of k_i for that position is 1 and c_{i+1} and $\bar{c}_{i+1}$ are 0 and 1, respectively. In this situation, it can be concluded that $\bar{c}_{i+1}$ is *generated*.

When the pair x_iy_i takes a value in {00, 01, 10}, $\bar{g}_i$ will be 1. In this case, if the complement of $c_i \cdot (\bar{c}_i)$ has the value of 1, it can be concluded that $\bar{c}_{i+1} = 1$. In other words, $\bar{g}_i$ has the role of propagating $\bar{c}_i$ just like the alive signal $(x_i + y_i)$ propagates c_i in common parallel prefix adders.

Similarly to the common PPF adder, we can use the associative operator o to associate pairs of $\bar{g}_i$ and k_i bits as follows: $(k_{i+1}, \bar{g}_{i+1})o(k_i, \bar{g}_i) = (k_{i+1} + k_i \cdot \bar{g}_{i+1} \cdot \bar{g}_{i+1} \cdot \bar{g}_i)$. The notation $\left(K_{i:j}, \bar{G}_{i:j}\right)$ is also used to denote the group *kill* and *not generate* produced by the input bits i, i–1, …, $j+1$,j. $(K_{i:j}, \bar{G}_{i:j})$ can also be obtained by associating two groups, $\left(K_{i:m'+1}\ \bar{G}_{i:m'+1}\right)$ and $\left(K_{m':j}, \bar{G}_{m':j}\right)$, with $i > m' \geq j$, as follows:

$$\left(K_{i:j}, \bar{G}_{i:j}\right) = \left(K_{i:m'+1}, \bar{G}_{i:m'+1}\right)o\left(K_{m':j}, \bar{G}_{m':j}\right) \tag{9}$$

The structure of nodes used in the proposed design is shown in Fig. 4. It should be mentioned that the functionalities of the black node and the one with a star are the same. Since the power consumption of the starred one is higher than the other, it is only used in the critical path to improve latency.

A generic procedure to construct an adder with the proposed design encompasses the following steps:

(i) In the first step, the *Preparation 1* unit is applied to produce k_i, $\bar{g}_i$, and p_i (in this paper $0 \leq i < n = 64$ or 128) for each bit-position of the operands.
(ii) In the second step, [n/8] *Preparation 2* blocks are applied to the n values of k_i and $\bar{g}_i$ (the *Preparation2* blocks are designed for 8-bit inputs). Each of these blocks produces all the values of K and $\bar{G}$. For example, the block number b $(0 \leq b < [n/8])$ receives k_{8*b} to k_{8*b+7}, and $\bar{g}_{8*b}$ to $\bar{g}_{8*b+7}$ as inputs and produces the group values $K_{8*b+j:8*b}$ and $\bar{G}_{8*b+j:8*b}$ for each j bit-position of the block $(0 \leq j < 7)$.
(iii) In the third step, a parallel prefix structure, named intermediate PPF, is used, which receives the values of K and $\bar{G}$ from the *Preparation 2* blocks and produces the complement of the input carry bits required in the fourth step.
(iv) In the fourth step, a simple 8-bit block, [n/16]-1 *Sum Producer_type1* blocks and [n/16] *Sum Producer_type2* blocks are used to produce the final values of SUM. In the *Sum Producer_type1* blocks, the values of SUM are calculated based on the compliment of the input carry bits provided by the third step, while the *Sum Producer_type2* is a sort of carry select adder that selects the value of SUM based on the actual value of $\bar{c}_{\text{in}}$.

Note: some structures of step3 and step4 work simultaneously.

3.1 Proposed structure for a 64-bit adder

The general structure of the proposed 64-bit adder is shown in Fig. 5 (in this figure $K_{i:j}$ and $\bar{G}_{i:j}$ are represented by $K(i{:}j)$ and $\bar{G}(i{:}\,j)$, respectively). The proposed adder consists of four parts, namely *Preparation 1*, *Preparation 2*, intermediate PPF, and the sum production blocks. A description of each part follows:

Preparation 1: in this part, the values of $p_i = (x_i \oplus y_i)$, $k_i = (\overline{x_i + y_i})$, and $\bar{g}_i = (\overline{x_i \cdot y_i})$ are computed for each bit. The related structures of p_i, k_i, and $\bar{g}_i$ are shown in Fig. 6. As it can be seen, NAND gates

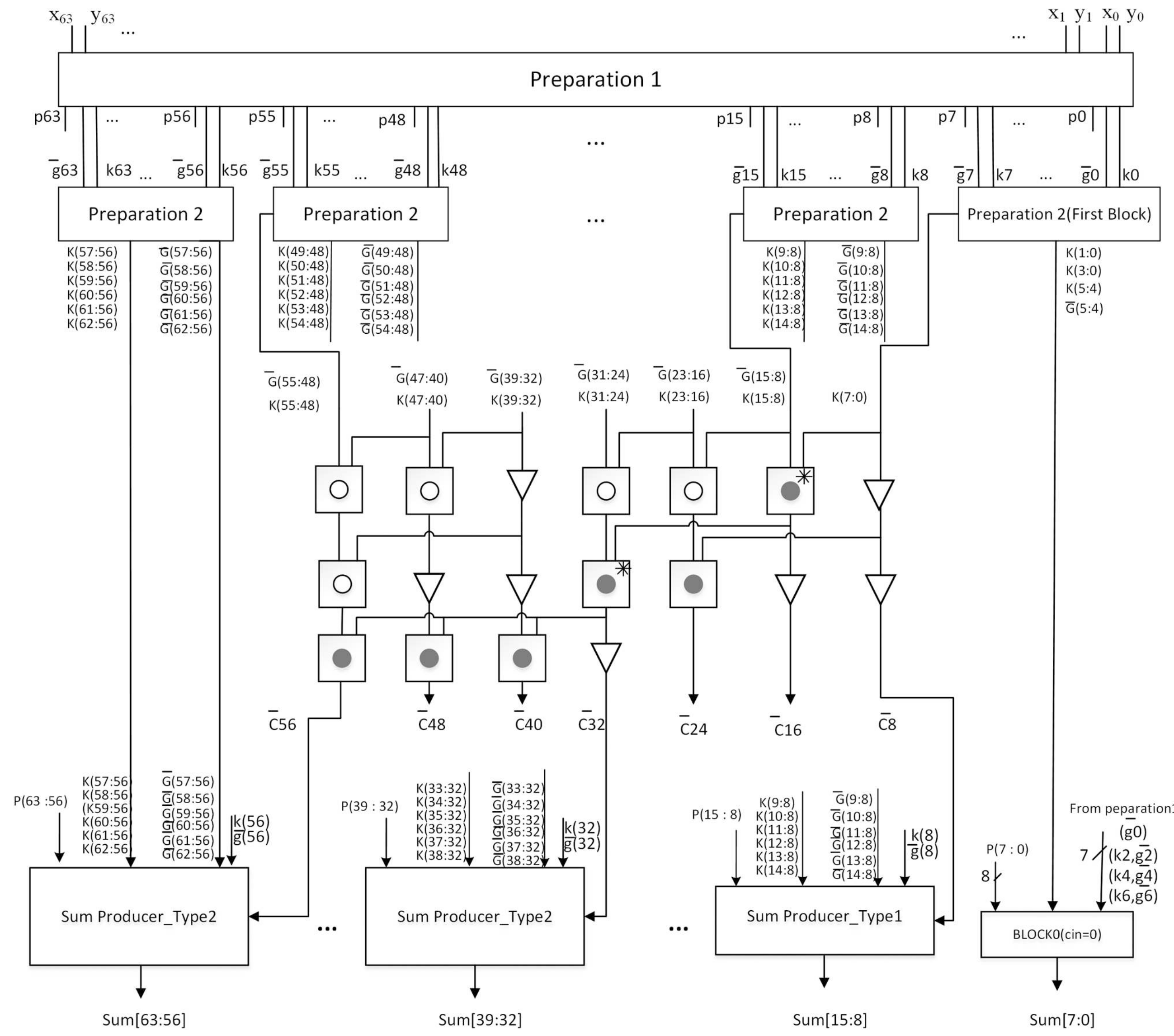


Fig. 5 *Proposed 64-bit adder*

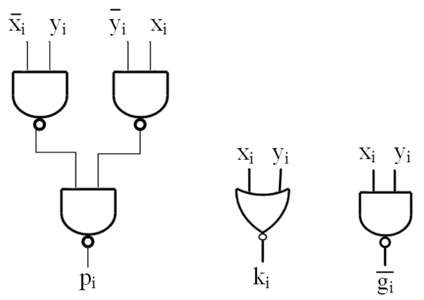


Fig. 6 *Circuits for producing p_i, k_i, and $\bar{g}_i$ in Preparation 1*

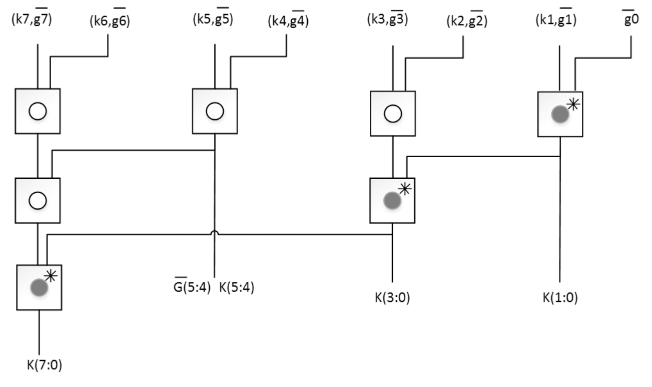


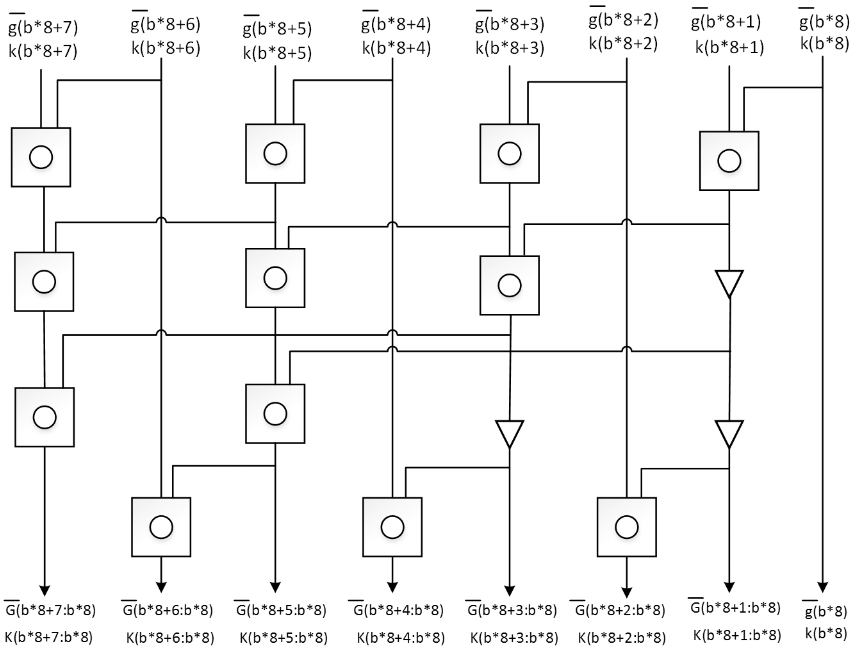
Fig. 7 *Structure of Preparation 2 for the first block*

are used for producing p_i, since the delay of two concurrent NAND gates is less than an AND gate and an OR gate.

Preparation 2: The structure of preparation 2 for the first block is shown in Fig. 7. Assuming that c_{in} is zero, a simplified circuit is used in that block to produce the values of K and $\bar{G}$. If the x_0y_0 pair has the value of 00, 01, or 10, the value of $\bar{c}_1$ becomes 1, and otherwise is 0. By considering that $k_0 = \overline{g_0} = \overline{x_0 \cdot y_0}$,the delay of the critical path is reduced by one gate, when compared to the general case where the carry-in cannot be assumed to be zero.

We have used the starred black node in the first block *of Preparation 2*, since these nodes belong to the critical delay path. $K(1{:}0)$, $K(3{:}0)$, $K(5{:}4)$, and $\bar{G}(5{:}4)$ are outputted by this structure and sent to sum production. The $K(7;0)$ signal, also generated by this block, is sent to the intermediate structure of PPF.

The structure of *Preparation 2* for the other blocks is shown in Fig. 8. In this structure, b is the number of the block $(1 \leq b \leq 7)$. According to the unit-gate model [5], the basic two-input gates (AND, OR, NAND, and NOR) count as one unit for the delay, and inverters are not considered. An XOR/XNOR gate also has two units of delay [5]. Therefore, the total value of K for each *Preparation 2* block has the delay of six units. This value, with the total value of $\bar{G}$, is sent to the intermediate structure of the parallel prefix.



g(b*8+7) k(b*8+7)
g(b*8+6) k(b*8+6)
g(b*8+5) k(b*8+5)
g(b*8+4) k(b*8+4)
g(b*8+3) k(b*8+3)
g(b*8+2) k(b*8+2)
g(b*8+1) k(b*8+1)
g(b*8) k(b*8)
G(b*8+7:b*8) K(b*8+7:b*8)
G(b*8+6:b*8) K(b*8+6:b*8)
G(b*8+5:b*8) K(b*8+5:b*8)
G(b*8+4:b*8) K(b*8+4:b*8)
G(b*8+3:b*8) K(b*8+3:b*8)
G(b*8+2:b*8) K(b*8+2:b*8)
G(b*8+1:b*8) K(b*8+1:b*8)
g(b*8) k(b*8)

Fig. 8 *Structure of Preparation 2 in the proposed 64-bit adder (except for the first block)*

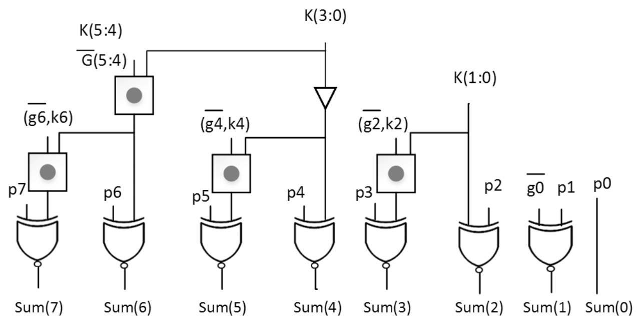


Fig. 9 *Structure of Block 0 in the Sum Producer*

It is indifferent whether the values of K and $\bar{G}$ for the other bit positions (except for the last one) are calculated with the maximum delay of six units or seven, since they are not in the critical path. Therefore, in order to simplify the hardware, it has been decided to produce them with higher delay (e.g. seven units of delay for producing $K_{b*8+6:b*8}$). Due to the structures of sum producer type 1 and sum producer type 2, which are introduced in the following sections, the critical path in the proposed adder has 15 units of delay (one unit for preparation 1, six units for preparation 2 (for producing the total value of K for the block), six units for the intermediate PPF, and two units for the MUX of *Sum Producer type 2*).

The Intermediate PPF: The outputs of the last bit positions of the preparation 2 blocks are the inputs of this block (in the middle part of Fig. 5). It produces carry bits (for blocks 1 to 7 of sum producer blocks) as its outputs ($\bar{c}_8$, $\bar{c}_{16}$, $\bar{c}_{24}$, $\bar{c}_{32}$, . .). Two starred black nodes are used in this part to enhance performance.

Considering the intermediate PPF structure, $\bar{c}_8$, $\bar{c}_{16}$, $\bar{c}_{24}$ are computed with the maximum of 3 units of delay, while three $\bar{c}_{32}$, $\bar{c}_{40}$, $\bar{c}_{48}$ and $\bar{c}_{56}$ will be ready with the maximum of six units of delay. Therefore, two types of sum generator blocks are introduced, namely *Sum Producer type 1* and *Sum Producer type 2*; the first one occupies less area at the cost of higher delay compared to type 2. In the sum production part, the blocks numbered 1, 2, and 3 are of type 1 and the 4, 5, 6, and 7 blocks are designed based on the second type. The first block of sum production has a different structure.

Sum Producer type 1 requires less hardware than the second type, and its output becomes ready with the delay of four units after carry-in becomes available. On the other hand, the output of the *Sum Producer type 2* becomes ready with the delay of a MUX (2 units of delay). It should be mentioned that having the value of $\bar{c}_8$, $\bar{c}_{16}$ and $\bar{c}_{24}$ in advance is used to reduce the hardware required for the blocks numbered 1, 2 and 3 of the sum producer, without any impact on the delay of the whole circuit. These structures are explained in the next section.

Sum Producer Blocks: This part consists of eight blocks (0 to 7), where each one computing eight bits of the final sum. As it was noted before, the blocks numbered 1, 2, and 3 are of the first type and blocks numbered 4, 5, 6, and 7 are designed based on *Sum Producer type 2*.

The structure of Block 0 in sum production is shown in Fig. 9. As it can be seen, K(1;0), K(3;0), K(5;4), $\bar{G}$(5;4), p_0 *to* p_7, $\bar{g}_0$, k_6, $\bar{g}_6$, k_4, $\bar{g}_4$, k_2 and $\bar{g}_2$ are inputted, and the values of sum bits (from 0 to 7) are computed and outputted. The value of $c_{\rm in}$ for this block is assumed to be zero (i.e. $\bar{c}_{\rm in} = 1$). With this condition, the value of Sum(0) will be equal $p(0)$, and $\bar{c}_1$ is equal to $\bar{g}_0$, which means that if x_0y_0 is not equal to 11, then $\bar{c}_1 = 1$. The fact that $\bar{c}_{\rm in} = 1$ simplifies the computation associated with the first bit position, allowing one to set $K(1;0)$ to $\bar{c}_2$, among other simplifications.

Since the adder operates with the complement of carry values ($\bar{c}_i s$) . XNOR gates, instead of XORs, are used to compute the Sum. We have used NAND and OR gates to implement a XNOR gate, leading to a delay of two units.

The structure of the *Sum Producer type 1* is shown in Fig. 10. The Ks and $\overline{Gs}$ in this structure are the outputs of the corresponding block in preparation 2. As it was mentioned before, this structure is used in the 1, 2, and 3 blocks of sum production, and its outputs are available with the delay of four units. Unlike [28], these blocks do not compute two values for Sum (for $c_{\rm in} = 1$ and $c_{\rm in} = 0$). Therefore, the required hardware is reduced.

The structure of the *Sum Producer type 2,* used in blocks numbered 4, 5, 6, and 7 of the sum production unit, is shown in Fig. 11. The values of S_0 and S_1 correspond to the sum bits assuming $\bar{c}_{\rm in} = 0$ and $\bar{c}_{\rm in} = 1$, respectively. Finally, based on the actual value of $\bar{c}_{\rm in}$, one of the S_0 and S_1 is selected as the result.

The structure of the *Sum Producer type 2* is as follows. For $\bar{c}_{\rm in} = 0$, if $K(i$:0$) = 1$, then $\bar{c}_{i+1}$ will be 1 (in this situation, the bit positions 0 to i of the block produce $\bar{c}_{i+1}$). Similarly, for $\bar{c}_{\rm in} = 1$, if one of the values of K(i;0) or $\bar{G}$(i;0) is one, $\bar{c}_{i+1}$ will be one. We have used NAND gates to implement the MUX for the *Sum Producer type 2* block. Therefore, the output of *Sum Producer type 2* becomes ready with the delay of two units (the delay of a MUX).

As it was mentioned before, in the intermediate PPF structure of the proposed adder, $\bar{c}_8$, $\bar{c}_{16}$ and $\bar{c}_{24}$ become ready with the maximum delay of three units and $\bar{c}_{32}$, $\bar{c}_{40}$, $\bar{c}_{48}$, and $\bar{c}_{56}$ are ready with the

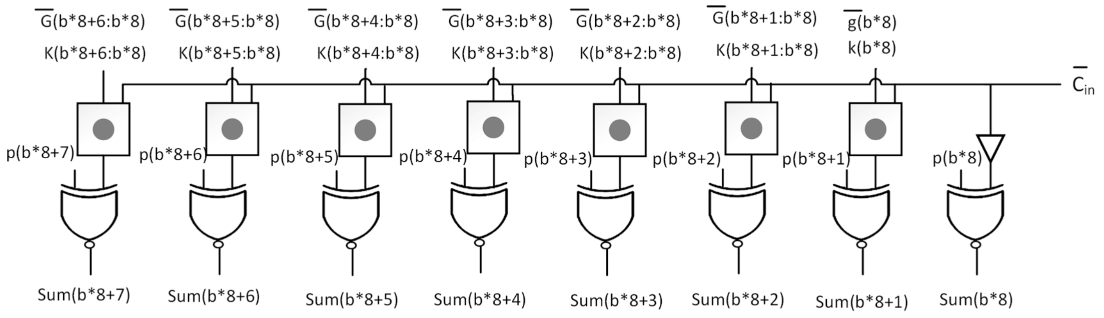
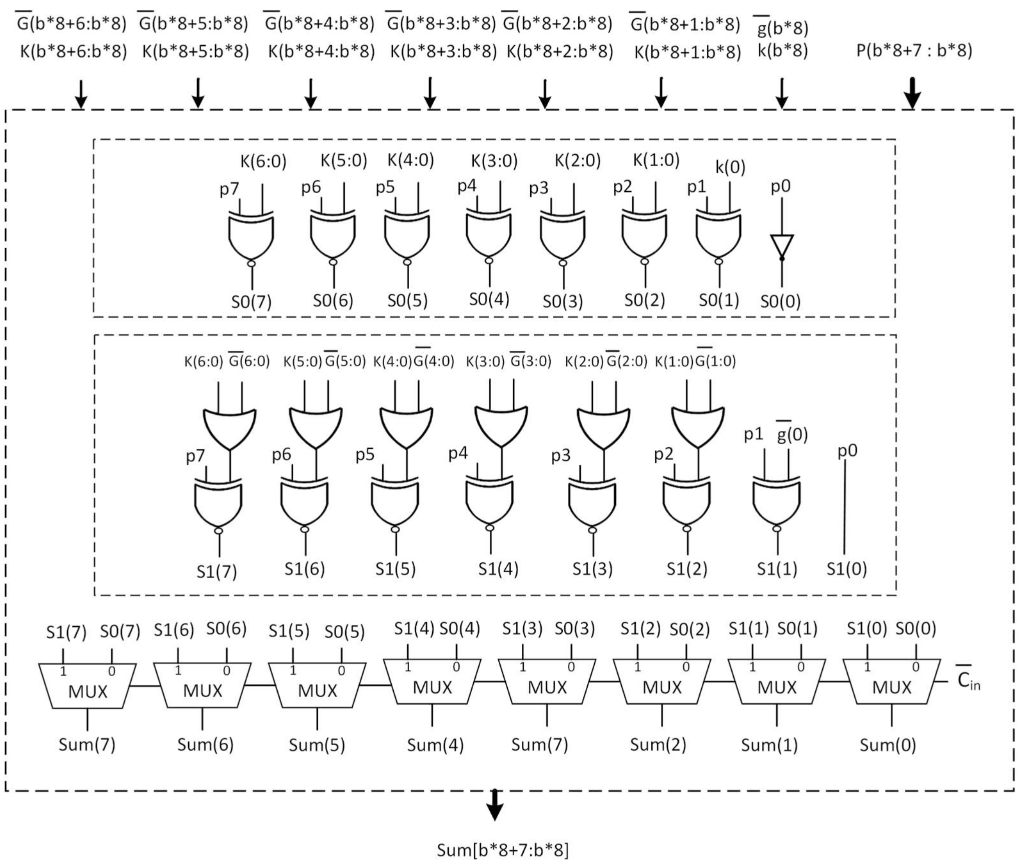


Fig. 10 *Sum Producer type 1*



$\overline{G}$(b*8+6:b*8) K(b*8+6:b*8)
$\overline{G}$(b*8+5:b*8) K(b*8+5:b*8)
$\overline{G}$(b*8+4:b*8) K(b*8+4:b*8)
$\overline{G}$(b*8+3:b*8) K(b*8+3:b*8)
$\overline{G}$(b*8+2:b*8) K(b*8+2:b*8)
$\overline{G}$(b*8+1:b*8) K(b*8+1:b*8)
$\overline{g}$(b*8) k(b*8)
P(b*8+7 : b*8)
K(6:0) K(5:0) K(4:0) K(3:0) K(2:0) K(1:0) k(0)
p7 p6 p5 p4 p3 p2 p1 p0
S0(7) S0(6) S0(5) S0(4) S0(3) S0(2) S0(1) S0(0)
K(6:0) $\overline{G}$(6:0) K(5:0) $\overline{G}$(5:0) K(4:0) $\overline{G}$(4:0) K(3:0) $\overline{G}$(3:0) K(2:0) $\overline{G}$(2:0) K(1:0) $\overline{G}$(1:0)
p7 p6 p5 p4 p3 p2 p1 $\overline{g}$(0) p0
S1(7) S1(6) S1(5) S1(4) S1(3) S1(2) S1(1) S1(0)
S1(7) S0(7) S1(6) S0(6) S1(5) S0(5) S1(4) S0(4) S1(3) S0(3) S1(2) S0(2) S1(1) S0(1) S1(0) S0(0)
1 0
MUX
$\overline{C}_{in}$
Sum(7) Sum(6) Sum(5) Sum(4) Sum(7) Sum(2) Sum(1) Sum(0)
Sum[b*8+7:b*8]

Fig. 11 *Sum Producer type 2*

maximum delay of six units. If the delay of the intermediate PPF is considered together with the delay of the sum production unit, the final results of the 1, 2, and 3 blocks become ready with the delay of seven units, and the final results of the 4, 5, 6, and 7 blocks have the delay of eight units. To compute the total delay, preparation 1 and preparation 2 should be considered. This will be done in Section 4.

3.2 Proposed structure for the 128-bit adder

The suggested structure for a 128-adder is shown in Fig. 12. As it can be observed, the delay of the intermediate PPF for 128-bit operands is two units more than for the 64-bit adder. This extra delay is kept, but the structure of *Preparation 2* is optimised to improve the area compared to the previous *Preparation 2* in the 64-bit adder. This structure can be seen in Fig. 13. The outputs of the preparation 2 unit in the 64-bit adder becomes ready after the delay of seven units with 12 nodes, while for the 128-bit adder, the same block has the delay of eight units (this is the delay of $K_{b*8+6:b*8}$) with 11 nodes (15 nodes less compared to a straightforward adaptation to a preparation 2 block for a 128-bit adder). Like the proposed 64-bit adder, the total value of K for each block are achieved after six units of delay. The structures of the first block of the *Preparation 2* unit and *Sum Production* are like the ones proposed for the 64-bit adder.

As it can be seen in Fig. 12, the values of $\bar{c}_8$, $\bar{c}_{16}$, …, $\bar{c}_{56}$ are computed at least one unit of delay faster than $\bar{c}_{64}$, $\bar{c}_{72}$, …, $\bar{c}_{120}$. Therefore, in the sum production unit, for blocks 1 to 7, *Sum Producer type 1* is used while for the other blocks the second type of sum producers is used. The structure of block 0 is the same as for the 64-bit adder.

Generally, the algorithm 1 (see Fig. 14) can be used for constructing the proposed adder.

4 Performance evaluation

In this section, the performance of the proposed 64- and 128-bit adders is evaluated and compared with the Kogge-Stone adder and the latest wide hybrid adders of [28, 29].

4.1 Analysis

The delay of the suggested 64-bit adder, based on gates in the critical path and without considering fan-out is:

$$d_{\text{Proposed Adder\_64}} = \max\left(d_{\bar{c}_{40}}, d_{\bar{c}_{48}}, d_{\bar{c}_{56}}\right) + d_{\text{mux}} \qquad (10)$$

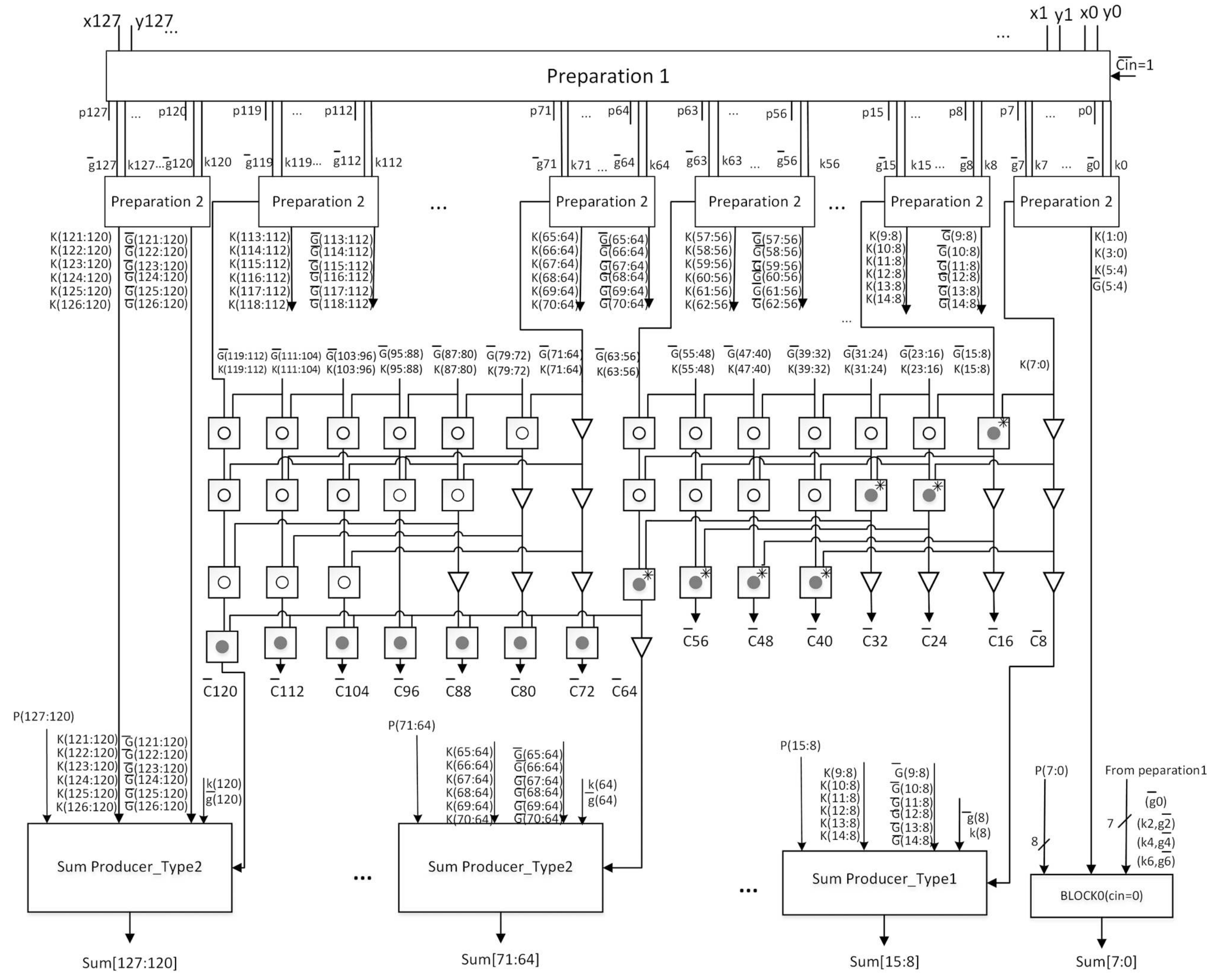


Fig. 12 *Proposed 128-bit adder*

As it was mentioned before, with the assumption that $\bar{c}_{\text{in}} = 1$, the value of k_0 is equal with $\bar{g}_0$. Considering $k_0 = \overline{a_0 \cdot b_0}$, the delay for achieving $\bar{c}_{40}$, $\bar{c}_{48}$, and $\bar{c}_{56}$ is:

$$
\begin{aligned}
d_{\bar{c}_{40}} = d_{\bar{c}_{48}} = {} & d_{\bar{c}_{56}} \\
= {} & d_{k0} + d_{\text{preparation2 (firstBlock)}} + d_{\text{middle PPF}} = \underbrace{d_{\text{NAND}}}_{k_0} \\
& + \underbrace{3 * (2d_{\text{NAND}})}_{\text{Preparation2\_first block}} + \underbrace{(2 * (2d_{\text{NAND}}) + d_{\text{AND}} + d_{\text{OR}})}_{\text{middle ppf}} \\
= {} & 11d_{\text{NAND}} + d_{\text{AND}} + d_{\text{OR}}
\end{aligned} \tag{11}
$$

In the intermediate PPF, three black nodes are placed in the critical path: two of them are starred black nodes (with the delay of two NANDs) while the other are normal nodes (with the delay of OR + AND). Therefore, the total delay of the suggested 64-bit adder based on the gates on the critical path is equal to:

$$
\begin{aligned}
d_{\text{proposed adder 64 bit}} = {} & \max\left(d_{\bar{c}_{40}},\ d_{\bar{c}_{48}},\ d_{\bar{c}_{56}}\right) + d_{\text{MUX}} \\
= {} & 11d_{\text{NAND}} + d_{\text{AND}} + d_{\text{OR}} + d_{\text{NOT}} \\
& + 2d_{\text{NAND}} = 13d_{\text{NAND}} + d_{\text{AND}} + d_{\text{OR}} + d_{\text{NOT}}
\end{aligned} \tag{12}
$$

The hardware requirement of the suggested 64-bit adder can be calculated as follows:

$$
\begin{aligned}
h_{\text{Proposed Adder\_64}} = {} & h_{\text{Preparation 1}} + h_{\text{Preparation 2\_fb}} \\
& + 7 * h_{\text{Preparation 2}} + h_{\text{PPF}} + h_{\text{BLOCK0}} \\
& + 3 * h_{\text{Sum Produce - Type1}} + 4 * h_{\text{Sum Produce - Type2}}
\end{aligned}
$$

where

$$
\begin{aligned}
h_{\text{Preparation 1}} = {} & 64\text{XOR} + 63\text{NAND} + 62\text{NOR} \\
= {} & 64 * (2\text{NOT} + 3\text{NAND}) + 63\ \text{NAND} + 62\text{NOR}
\end{aligned}
$$

$$
\begin{aligned}
h_{\text{Preparation 2\_fb}} = {} & 4 * \text{node}_{\text{white}} + 3 * \text{node}_{\text{black - star}} \\
= {} & 4 * (\text{OR} + 2\text{AND}) + 3 * (2\text{NAND} + \text{NOT})
\end{aligned}
$$

$$
\begin{aligned}
h_{\text{Preparation 2}} = {} & 12 * \text{node}_{\text{white}} + 3 * \text{buffer} \\
= {} & 12 * (\text{OR} + 2\text{AND}) + 3 * (2\text{NOT})
\end{aligned}
$$

$$
\begin{aligned}
h_{\text{PPF}} = {} & 5 * \text{node}_{\text{white}} + 3 * \text{node}_{\text{black - star}} + 3 * \text{node}_{\text{black}} \\
& + 7 * \text{buffer} = 5 * (\text{OR} + 2\text{AND}) + 3 * (2\text{NAND} + \text{not}) \\
& + 3 * (\text{AND} + \text{OR}) + 7 * (2\text{NOT})
\end{aligned}
$$

$$
\begin{aligned}
h_{\text{BLOCK0}} = {} & 4 * \text{node}_{\text{black}} + 7 * \text{XNOR} + \text{buffer} \\
= {} & 4 * (\text{OR} + \text{AND}) + 7 * (\text{OR} + 2\text{NAND}) + 2\text{NOT}
\end{aligned}
$$

$$
\begin{aligned}
h_{\text{Sum Produce - Type1}} = {} & 7 * \text{node}_{\text{black}} + 8 * \text{XNOR} + \text{buffer} \\
= {} & 7 * (\text{OR} + \text{AND}) + 8 * (\text{OR} + 2\text{NAND}) + 2\text{not}
\end{aligned}
$$

$$
\begin{aligned}
h_{\text{Sum Producer - Type2}} = {} & 14 * \text{XNOR} + 6\text{OR} + \text{NOT} + 8\text{MUX} \\
= {} & 14 * (\text{OR} + 2\text{NAND}) + 6\text{OR} + \text{NOT} \\
& + 8 * (\text{not} + 3\text{NAND})
\end{aligned}
$$

Therefore

$$h_{\text{Proposed Adder\_64}} = 537\ \text{NAND} + 62\ \text{NOR} + 214\ \text{AND} + 232\ \text{OR} + 234\ \text{NOT} \quad (13)$$

The delay of [28] can be calculated as follows (the critical path is shown in Fig. 2):

$$d_{[28]\_64\,\text{bit}} = d_{\text{generate}} + d_{\text{black round node}} + d_{\text{black square node}} + d_{4-4\text{CSSA(gates in critical path)}} + d_{\text{mux}}$$

where

$$d_{\text{generate}} = d_{\text{AND}}, \quad d_{\text{MUX}} = d_{\text{NOT}} + d_{\text{AND}} + d_{\text{OR}}$$

$$d_{\text{black round node}} = d_{\text{AND}} + d_{\text{OR}}$$

$$d_{\text{black square node}} = d_{\text{AND}} + 2 * d_{\text{OR}}$$

$$\begin{aligned} d_{4-4\text{CSSA(gates in critical path)}} &= 3 * d_{\text{black round node}} + d_{\text{xor}} \\ &= 3 * d_{\text{AND}} + 3 * d_{\text{OR}} + d_{\text{not}} \\ &\quad + d_{\text{AND}} + d_{\text{OR}} = 4 * d_{\text{AND}} \\ &\quad + 4 * d_{\text{OR}} + d_{\text{NOT}} \end{aligned}$$

Therefore, the total delay of [28] is equal with:

$$d_{[28]\_64\,\text{bit}} = 8 * d_{\text{AND}} + 8 * d_{\text{OR}} + 2 * d_{\text{NOT}} \quad (14)$$

From (12) and (14), it can be concluded that theoretically the delay of the proposed adder is improved compared to [28] (note that the delay of a NAND gate is lower than that of OR/AND gates in a VLSI implementation).

The hardware requirement of the 64-bit adder in [28] can be calculated as follows:

$$h_{[28]\_64} = h_{\text{Preparation}} + h_{\text{PPF\_[28]}} + 7 * h_{4-4\text{CSSA}} + h_{\text{CLS}-8}$$

where

$$\begin{aligned} h_{\text{Preparation}} &= 64\text{XOR} + 63\text{AND} + 62\text{OR} \\ &= 64 * (2\text{NOT} + 2\text{AND} + \text{OR}) \\ &\quad + 63\ \text{AND} + 62\text{OR} \end{aligned}$$

$$\begin{aligned} h_{\text{PPF\_[28]}} &= 50 * \text{node}_{\text{white}} + 9 * \text{node}_{\text{black}} \\ &\quad + 7 * \text{node}_{\text{black square node}} + 7 * \text{buffer} \\ &= 50 * (\text{OR} + 2\text{AND}) + 9 * (\text{AND} + \text{not}) \\ &\quad + 7 * (2\text{AND} + 2\text{OR}) + 14\text{NOT} \end{aligned}$$

$$\begin{aligned} h_{4-4\text{CSSA}} &= 10 * \text{node}_{\text{black}} + 14 * \text{XOR} + 8 * \text{MUX} \\ &= 10 * (\text{OR} + \text{AND}) + 14 * (2\text{AND} + 2\text{NOT} + \text{OR}) \\ &\quad + 8 * (2\text{AND} + \text{NOT} + \text{OR}) \end{aligned}$$

$$\begin{aligned} h_{\text{CLS}-8} &= 5 * \text{node}_{\text{black}} + 7 * \text{XOR} = 10 * (\text{OR} + \text{AND}) \\ &\quad + 7 * (2\text{AND} + 2\text{NOT} + \text{OR}) \end{aligned}$$

Therefore

$$h_{[28]\_64} = 711\ \text{AND} + 435\ \text{OR} + 408\ \text{NOT} \quad (15)$$

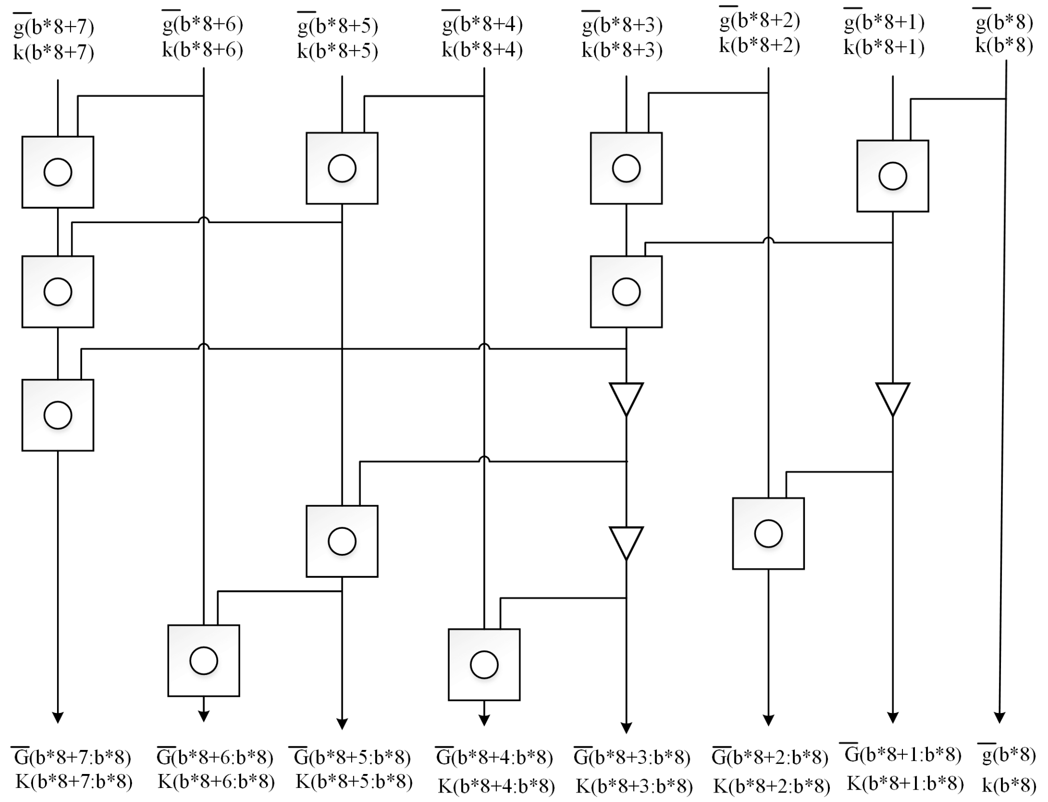
Equations (13) and (15) show that the proposed adder can reduce the hardware requirements compared to [28]. The total number of NAND/NOR/AND/OR and NOT gates in the proposed adder is 101 and 174, respectively, less than the AND/OR and NOT gates of [28].

The total delay of the 64-bit Kogge-Stone adder can be calculated as follows:

$$\begin{aligned} d_{\text{Kogge\_Stone\_64}} &= \underbrace{d_{\text{AND}}}_{g_0} + \underbrace{6 * (d_{\text{AND}} + d_{\text{OR}})}_{\text{ppf}} \\ &\quad + \underbrace{d_{\text{NOT}} + d_{\text{AND}} + d_{\text{OR}}}_{\text{xor}} = 8d_{\text{AND}} + 7d_{\text{OR}} + d_{\text{NOT}} \end{aligned} \quad (16)$$

Moreover, the hardware requirement of the 64-bit Kogge-Stone adder is:

$$h_{\text{Kogge\_Stone\_64}} = h_{\text{Preparation}} + h_{\text{PPF}} + 64 * h_{\text{xor}}$$



$\overline{g}$(b*8+7) k(b*8+7)
$\overline{g}$(b*8+6) k(b*8+6)
$\overline{g}$(b*8+5) k(b*8+5)
$\overline{g}$(b*8+4) k(b*8+4)
$\overline{g}$(b*8+3) k(b*8+3)
$\overline{g}$(b*8+2) k(b*8+2)
$\overline{g}$(b*8+1) k(b*8+1)
$\overline{g}$(b*8) k(b*8)
$\overline{G}$(b*8+7:b*8) K(b*8+7:b*8)
$\overline{G}$(b*8+6:b*8) K(b*8+6:b*8)
$\overline{G}$(b*8+5:b*8) K(b*8+5:b*8)
$\overline{G}$(b*8+4:b*8) K(b*8+4:b*8)
$\overline{G}$(b*8+3:b*8) K(b*8+3:b*8)
$\overline{G}$(b*8+2:b*8) K(b*8+2:b*8)
$\overline{G}$(b*8+1:b*8) K(b*8+1:b*8)
$\overline{g}$(b*8) k(b*8)

Fig. 13 *Structure of the Preparation 2 in the proposed 128-bit adder (except the first block)*

where

$$h_{\text{Preparation}} = 64\text{XOR} + 63\text{AND} + 62\text{OR}$$
$$= 64 * (2\text{not} + 2\text{AND} + \text{OR}) + 63\ \text{AND} + 62\text{OR}$$

$$h_{\text{ppf}} = 258 * \text{node}_{\text{white}} + 63 * \text{node}_{\text{black}} + 63 * \ \text{buffer}$$
$$= 258 * (2\text{AND} + \text{OR}) + 63 * (\text{AND} + \text{OR}) + 126\ \text{NOT}$$

Therefore

$$h_{\text{Kogge\_Stone\_64}} = 898\ \text{AND} + 511\ \text{OR} + 382\ \text{NOT} \tag{17}$$

Comparing (16) and (17) with (12) and (13) shows that the total number of gates in the critical delay path of the proposed adder is similar to Kogge-Stone (the only difference is the gate types) while the proposed adder significantly reduces the hardware requirements.

The structure of the 64-bit adder of [29] uses 2-input NAND gates, and consists of three main parts, namely Stage 1, Stage 2, and Stage 3. The delay of this adder can be calculated as follows:

$$d_{[29]\_64} = d_{\text{Stage 1}} + d_{\text{Stage 2}} + d_{\text{Stage 3}}$$

where

$$d_{\text{Stage 1}} = d_{\text{XOR}} = 4d_{\text{NAND}}$$

$$d_{\text{Stage 2}} = d_{\text{p, g Block}} + d_{\text{C's Producer Block}}$$

$$d_{\text{p, g Block}} = d_{\text{g\_out}} = 6\ d_{\text{NAND}} + 4d_{\text{NOT}}$$

$$d_{\text{C's Producer Block}} = 6\ d_{\text{NAND}} + 4d_{\text{NOT}}$$

$$d_{\text{Stage 3}} = 7 * (2d_{\text{NAND}}) + d_{\text{XOR}} = 18d_{\text{NAND}}$$

Therefore:

$$d_{[29]\_64} = 34\ d_{\text{NAND}} + 8\ d_{\text{NOT}} \tag{18}$$

From (18) and (12), it can be concluded that the proposed adder is much faster than [29]. The hardware requirements of [29] can be estimated as:

$$h_{[29]\_64} = h_{\text{Stage 1}} + h_{\text{Stage 2}} + h_{\text{Stage 3}}$$

where:

$$h_{\text{Stage 1}} = h_{\text{(P\&G) Blocks}} = 64(\text{XOR}) + 63(\text{AND}) = 319\ \text{NAND}$$

$$h_{\text{Stage 2}} = h_{\text{p, g Blocks}} + h_{\text{C's Producer Block}}$$

$$h_{\text{p, g Blocks}} = \left(\underbrace{35\text{NAND} + 28\text{NOT}}_{g}\right) + 6 * \left(\underbrace{42\text{NAND} + 35\ \text{NOT}}_{g, p}\right) = 287\text{NAND} + 238\ \text{NOT}$$

In the previous equation, for the first block, only the generation of the *g* signal is considered (for the all the adders in this paper, we assume that $C_{\text{in}} = 0$). We also disregard the computation of *p* and *g* for the last block (we eliminate the parts of the circuit that calculate C_{out} in all adders to perform a fair comparison).

In [29], carries are partially computed from a Carry-Look-ahead structure:

$$h_{\text{C's Producer Block}} = h_{C16} + h_{C24} + h_{C32} + h_{C40} + h_{C48} + h_{C56}$$

$$h_{C16} = 2\text{NAND} + \text{NOT}$$

$$h_{C24} = 5\text{NAND} + 3\text{NOT}$$

$$h_{C32} = 9\text{NAND} + 6\text{NOT}$$

$$h_{C40} = 14\text{NAND} + 10\text{NOT}$$

$$h_{C48} = 20\text{NAND} + 15\text{NOT}$$

$$h_{C56} = 27\text{NAND} + 21\text{NOT}$$

$$h_{\text{C's Producer Block}} = 77\ \text{NAND} + 56\ \text{NOT}$$

$$h_{\text{Stage 2}} = 364\ \text{NAND} + 294\ \text{NOT}$$

$$h_{\text{Stage 3}} = h_{\text{(S\&C)Blocks}} = 8 * (8\text{XOR} + 7(\text{AND} + \text{OR}))$$
$$= 8 * (8(4\text{NAND}) + 7(2\text{NAND})) = 368\text{NAND}$$

Therefore

$$h_{[29]\_64} = 1051\ \text{NAND} + 294\ \text{NOT} \tag{19}$$

Transistor count comparison is herein performed by counting the number of transistors required to implement each digital gate. We

```
1: **Require** Input: $n$
2: b ← $\lceil \frac{n}{8} \rceil$; // b is the number of *preparation 2* blocks
3: Constructing the middle PPF (for $\lceil \frac{n}{8} \rceil$-bit operands) by using the Kogge-Stone structure with 2* $Log^{\lceil \frac{n}{8} \rceil}$ units of delay.
4: Constructing a proper structure for *preparation 2* block with a delay of less than $D_{MUX\_pre2}$. // $D_{MUX\_pre2}$ is the maximum delay that *preparation 2* can have theoretically (it can be calculated as follows):
    $D_{MUX\_pre2} \leftarrow D_{total\ value\ of\ K\ in\ pre2} + D_{PPF} - D_{S1}$;
    where:
    $D_{total\ value\ of\ K\ in\ pre2} \leftarrow 6$; // this is the delay of
        *preparation 2* block to produce the total value of K for the block (this value is sent to the PPF structure).
    $D_{PPF} \leftarrow 2^* Log^{\lceil \frac{n}{8} \rceil}$; // this is the delay of middle PPF
    $D_{S1} \leftarrow 3$; // This is the delay of *Sum Producer type2*
        for producing S1 signals.
// **note**: in the practice the delay of *preparation 2* must be
        shorter than $D_{MUX\_pre2}$, since we do not consider the delay of inverters and fan-out in the Unit-Gate model).
5: n1 ← $\frac{1}{2} * \lceil \frac{n}{8} \rceil - 1$; // n1 is the number of *Sum Producer type1* blocks (*Sum Producer type1* has the same structure in the both proposed 64-bit adder and 128-bit adder).
6: n2 ← $\frac{1}{2} * \lceil \frac{n}{8} \rceil$; // n2 is the number of *Sum Producer type2* blocks (*Sum Producer type2* has the same structure in the both proposed 64-bit adder and 128-bit adder).
7: **return** b, n1, n2;
```

Fig. 14 *Algorithm 1: Adder construction*

Table 1 Performance comparison of 64-bit adders based on VLSI implementations

Adder_64 bit	Delay, ns	Power, mW	Allocated area, μm^2	Energy, pJ	ADP
proposed adder	0.561	6.391	10,383	3.585	5824.863
(PPF/CSSA4_4) [28]	0.638	7.508	13,202	4.790	8422.876
Kogge-Stone	0.598	9.621	14,424	5.753	8625.552
(hybrid RCA/HCLA_8^{64}) [29]	0.816	5.864	10,465	4.785	8539.44

Table 2 Performance comparison of 128-bit adders based on VLSI implementations

Adder_128 bit	Delay, ns	Power, mW	Allocated area, μm^2	Energy, pJ	ADP
proposed adder	0.637	11.46	20,966	7.3	13,355.342
(PPF/CSSA 6_3) [28]	0.647	12.59	21,723	8.146	14,054.781
Kogge-Stone	0.687	17.37	31,500	11.933	21,640.5
hybrid RCA/HCLA_{12}^{128} [29]	1.07	10.51	25,661	11.246	27,457.27

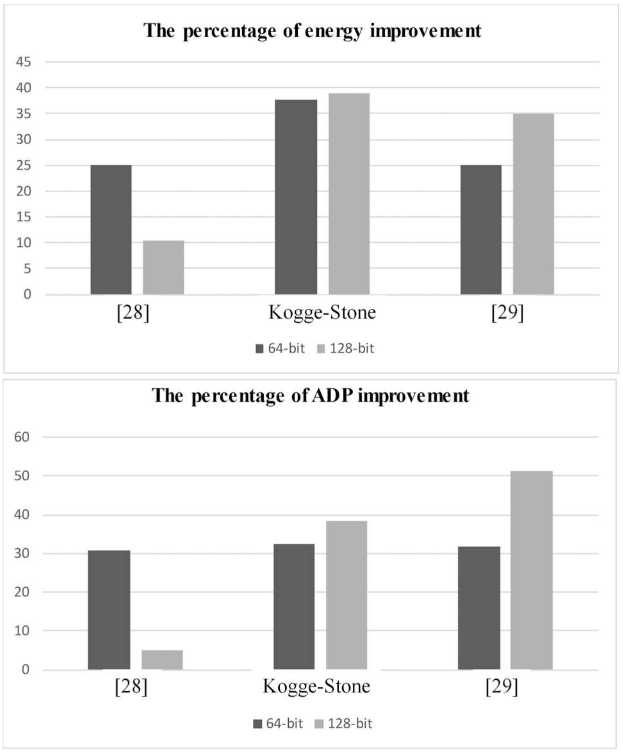


Fig. 15 *Percentage of energy and ADP improvement of the proposed 64-bit and 128-bit adders in comparison with [28], Kogge-Stone, and [29]*

considered complementary metaloxide semiconductor (CMOS) designs for all the required gates. Therefore, each two-input NAND and NOR gate can be implemented with four transistors [33]. A NOT gate requires two transistors, and six transistors are required for each two-input AND and OR gate. From (13), (15), (17), and (19), the total number of transistors is 5540, 7692, 9218, and 4792 for the proposed adder, [28], Kogge-Stone, and [29], respectively. Therefore, except for [29], which is implemented with about 13% less transistors, the proposed adder requires in average around 52% less transistors than the other adders in the state-of-the-art.

4.2 Assessment

The experimental results presented in this section are achieved using Cadence tools. All adder circuits were implemented using 65-nm TSMC CMOS logic salicide process (1-poly, 9-metal). Cadence RTL Compiler tools version v11.20-s012_1 were used for synthesising the designs and the Cadence Encounter and NanoRoute tools (versions v09.12-s159 and v09.12-s013, respectively) for placing and routing. The experimental results were obtained without imposing constrains on the delay or area.

Power consumption was obtained from the placed-and-routed circuit specifications, for 20% of switching activity. The Cadence Encounter power reporting tool was used to measure the total power, including the dynamic and leakage power.

The delay of the 64-bit adder was theoretically analysed in the previous section. The experimental results for the proposed 64-bit adder are shown in Table 1. In order to have a comprehensive assessment, Hybrid RCA/HCLA_8^{64} [28, 29], and Kogge-Stone adders are also considered. Due to the emphasis of [29] on using 2-input NAND gates, the structure of the Hybrid RCA/HCLA_8^{64} is only designed with this gate and a few necessary NOT gates.

Based on the results in Table 1, it can be concluded that the proposed 64-bit adder can improve the Energy by about 25% when compared to [28, 29] and by >37% in comparison with Kogge-Stone. The improvements of the Area-Delay Product (ADP) are about 31, 32.5, and 32% when compared to [28], Kogge-Stone, and [29], respectively. The proposed 64-bit adder not only has less delay but also enhances the energy consumption and the circuit area.

The experimental results for the proposed 128-bit adder and those of related art are shown in Table 2. According to the results, the proposed 128-adder improves the energy consumption by about 10.38, 38.83, and 35% in comparison with [28], Kogge-Stone, and [29], respectively. Moreover, the ADP is reduced by about 5, 38, and 51% when compared to [28], Kogge-Stone, and [29].

Fig. 15 represents the percentage of the improvement of the power-delay product, i.e. energy, and area-delay-product of the proposed 64-bit and 128-bit adders in comparison with [28], Kogge-Stone, and [29].

For the 64-bit adder, the delay of the proposed adder is equivalent to the Kogge-Stone while its area is as low as [29]. Therefore, the proposed adder has the lowest ADP among all. As mentioned before, Kogge-Stone has the minimum depth among PPF adders with the cost of high area that leads to high power consumption for this adder. The proposed adder has achieved the performance of Kogge-Stone while reducing the area and power consumption significantly.

For the 128-bit adders, the critical path of the proposed adder, Kogge-Stone, and [28] goes through the PPF network (with the difference that gates in the critical path have different types). Therefore, the delay of these three adders is almost the same. The main advantage of the proposed adder, in comparison with the others, is the reduced cost, which results in less power consumption and consequently better energy efficiency.

5 Conclusions

In this paper, a new efficient adder structure is introduced for large operands, based on the PPF and on improving the sum producer blocks. Two types of these latter blocks were introduced in this paper to enhance the performance and area cost. The first type of the sum-producer block, in which the complement of the carry-in becomes ready sooner, has a simpler structure with lower hardware cost. The second type of sum producer features a CSL architecture with an efficiently designed structure. In the proposed structure, the complements of the carry bits are generated and propagated to improve performance. Experimental results show that the proposed

64-bit adder can improve the energy and ADP by about 25–37% and 31–32%, respectively, in comparison with the related state of the art.

6 Acknowledgments

This work was partially supported by Portuguese funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2019.

7 References

[1] Goel, S., Kumar, A., Bayoumi, M.A.: 'Design of robust, energy-efficient full adders for deep-submicrometer design using hybrid-CMOS logic style', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 2006, **14**, (12), pp. 1309–1321

[2] Molahosseini, A.S., Sousa, L., Chang, C.H. (Eds.): '*Embedded systems design with special arithmetic and number systems*' (Springer, New York, NY, USA, 2017)

[3] Chang, C.H., Molahosseini, A.S., Zarandi, A.A.E., *et al.*: 'Residue number systems: A new paradigm to datapath optimization for low-power and high-performance digital signal processing applications', *IEEE Circuits Syst. Mag.*, 2015, **15**, (4), pp. 26–44

[4] Parhami, B.: '*Computer arithmetic: algorithms and hardware designs*' (Oxford University Press, New York, NY, USA, 2010)

[5] Zimmermann, R.: 'Binary Adder Architectures for Cell-Based VLSI and their Synthesis'. Ph.D. thesis, Swiss Federal Institute of Technology, 1997

[6] Jafarzadehpour, F., Keshavarzian, P.: 'Low-power consumption ternary full adder based on CNTFET', *IET Circuits Devices Syst.*, 2016, **10**, (5), pp. 365–374

[7] Hauck, S., Hosler, M., Fry, T.: 'High performance carry chains for FPGAs', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 2000, **8**, (2), pp. 138–147

[8] Huang, C., Wang, J., Yeh, C., *et al.*: 'The CMOS carry-forward adders', *IEEE J. Solid-State Circuits*, 2004, **39**, (2), pp. 327–336

[9] Singh, S., Kumar, D.: 'Design of area and power efficient modified carry select adder', *Int. J. Comp. Appl. (0975-8887)*, 2011, **33**, (3), pp. 14–18

[10] Mohanty, B.K., Patel, S.K.: 'Area-delay-power efficient carry select adder', *IEEE Trans. Circuits Syst.-II: Express Briefs*, 2014, **61**, (6), pp. 418–422

[11] Kumar, U.S., Salih, K.K., Sajith, K.: 'Design and implementation of carry select adder without using multiplexer'. Proc. IEEE Int. Conf. on Emerging Technology Trends in Electronics Communication and Networking, Gujarat, India, 2012, vol. A247, pp. 529–551

[12] Pang, Y., Wang, J., Wang, S.: 'A 16-bit carry skip adder designed by reversible logic'. Proc. 5th Int. Conf. on Biomedical Engineering and Informatics, Chongqing, China, 2012, pp. 1332–1335

[13] Chirca, K.: 'A static low-power, high-performance 32-bit carry skip adder'. Proc. Euromicro Symp. on Digital System Design, Rennes, France, 2004. pp. 1–4

[14] Lin, Y.S.: 'Delay efficient 32-bit carry-skip adder'. 13th IEEE Int. Conf. on Electronics, Circuits and Systems, ICECS '06, Nice, France, 2006, pp. 506–507

[15] Burgess, N.: 'Accelerated carry-skip adders with low hardware cost'. Proc. Conf. on Signals, Systems and Computers, Pacific Grove, CA, USA, 2011, pp. 852–853

[16] Balasubramanian, P., Edwards, D.A., Toms, W.B., *et al.*: 'Self-timed section-carry based carry look ahead adders and the concept of alias logic', *J. Circuits Syst. Comput.*, 2013, **22**, (4), pp. 1–24

[17] Zlatanovici, R., Kao, S., Nikolic, B.: 'Energy-delay optimization of a 64-bitcarry – look ahead adders with a 420ps 90 nm CMOS design example', *IEEE J. Solid-State Circuits*, 2009, **44**, (2), pp. 569–583

[18] Morrison, M., Lewandowski, M., Meana, R., *et al.*: 'Design of a novel reversible ALU using an enhanced carry look-ahead adder'. 11th IEEE Conf. on Nanotechnology (IEEE-NANO), Portland, OR, USA, 2011, pp. 1436–1440

[19] Kogge, P.M., Stone, H.S.: 'A parallel algorithm for the efficient solution of a general class of recurrence equations', *IEEE Trans. Comput.*, 1973, **C-22**, (8), pp. 786–793

[20] Brent, R.P., Kung, H.T.: 'A regular layout for parallel adders', *IEEE Trans. Comput.*, 1982, **C-31**, (3), pp. 260–264

[21] Ladner, R.E., Fischer, M.J.: 'Parallel prefix computation', *J. ACM*, 1980, **27**, (4), pp. 831–838

[22] Han, T., Carlson, D.A.: 'Fast area-efficient VLSI adders'. Proc. IEEE 8th Symp. on Computer Arithmetic (ARITH), Como, Italy, 1987, pp. 49–56

[23] Sklansky, J.: 'Conditional-sum addition logic', *IRE Trans. Electron. Comput.*, 1960, **EC-9**, pp. 226–231

[24] Verma, A.K., Brisk, P., Ienne, P.: 'Variable latency speculative addition: A new paradigm for arithmetic circuit design'. Proc. Design, Automation and Test in Europe, Munich, 2008, pp. 1250–1255

[25] Cilardo, A.: 'A new speculative addition architecture suitable for two's complement operations'. Proc. Design, Automation & Test in Europe Conf. & Exhibition, Nice, 2009, pp. 664–669

[26] Del Barrio, A.A., Hermida, R., Memik, S.O., *et al.*: 'Multispeculative addition applied to datapath synthesis', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2012, **31**, (12), pp. 1817–1830

[27] Del Barrio, A.A., Hermida, R., Memik, S.O.: 'Exploring the energy efficiency of multispeculative adders'. Proc. IEEE 31st Int. Conf. on Computer Design (ICCD), Asheville, NC, 2013, pp. 309–315

[28] Cui, X., Liu, W., Wang, S., *et al.*: 'Design of high-speed wide-word hybrid parallel-prefix/carry-select and skip adders', *J. Signal. Process. Syst.*, 2018, **90**, (3), pp. 409–419

[29] Ibrahim, A., Gebali, F.: 'Optimized structures of hybrid ripple carry and hierarchical carry lookahead adders', *Microelectron. J.*, 2015, **46**, (9), pp. 783–794

[30] Javali, R., Nayak, R.J., Mhetar, A.M., *et al.*: 'Design of high speed carry save adder using carry lookahead adder'. Proc. Int. Conf. on Circuits Communication Control and Computing (I4C), Bangalore, India, 2014, pp. 33–36

[31] Du, K., Varman, P., Mohanram, K.: 'High performance reliable variable latency carry select addition'. Proc. Design Automation and Test in Europe (DATE '12), Dresden, Germany, 2012, pp. 1257–1262

[32] Dimitrakopoulos, G., Nikolos, D.: 'High-speed parallel-prefix VLSI ling adders', *IEEE Trans. Comput.*, 2005, **54**, (2), pp. 225–231

[33] Weste, N.H.E., Harris, D.M.: '*CMOS VLSI design: A circuits and systems perspective*' (Addison-Wesley, Reading, MA, USA, 2011, 4nd edn.)