IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 16 November 2020; revised 23 February 2021; accepted 21 March 2021.
Date of publication 25 March 2021; date of current version 16 September 2021.

Digital Object Identifier 10.1109/TETC.2021.3068729

Sum Propagate Adders

GIORGOS DIMITRAKOPOULOS, KLEANTHIS PAPACHATZOPOULOS, AND VASSILIS PALIOURAS, (Member, IEEE)

Giorgos Dimitrakopoulos is with the Electrical and Computer Engineering Department, Democritus University of Thrace, 67100, Xanthi Greece
Kleanthis Papachatzopoulos and Vassilis Paliouras are with the Electrical and Computer Engineering Department, University of Patras, 26504 Patras, Greece
CORRESPONDING AUTHOR: GIORGOS DIMITRAKOPOULOS (dimitrak@ee.duth.gr)

ABSTRACT Binary adders are present in every digital computer system. Even if their structure has evolved significantly over the last decades following the progress in logic and circuit design, the scaling of implementation technologies, and the improvement of logic synthesis tools, the fundamental carry-propagation algorithm that guides their operation remains unchanged. This work takes a different path and explores the possibility of performing addition by propagating directly the sum bits of previous bit positions instead of carries. The transformation of binary carry-propagate addition to an equivalent sum propagate addition opens up a whole new design space that spans from ripple-sum to sum-lookahead adders. New parallel-prefix structures that follow the sum-propagation paradigm are presented using a newly introduced associative prefix operator. Sum-propagate and carry-propagate adders have asymptotically the same area and delay complexity. In practice, however, carry propagate adders exhibit better characteristics when implemented in currently established implementation technologies. This gap is expected to reduce in the future using multiple-independent-gate transistors that are promising functionality-enhanced beyond CMOS device technologies, and allow the cost-efficient implementation of AND-XOR operations involved in sum-propagate adders.

INDEX TERMS Binary addition, parallel prefix adders, FPGA adders, computer arithmetic, logic design

I. INTRODUCTION

Adder design is under a constant evolution following the evolution of the implementation technologies and the discovery of new logic and circuit design techniques [1]–[3]. Irrespective of how many transformations we have witnessed to the logic design of adders the last decades, the only attribute that remained unchanged is the fundamental algorithm of carry propagation used for implementing binary addition. In all cases, for computing the sum of two numbers, carries are generated locally and propagated to the next bit positions producing new carries that in turn compute the needed sum bits.

This work, explores, for the first time—to the best of our knowledge—the possibility of *performing addition by propagating directly the sum bits of the previous bit positions, instead of carries*. Carry generation/propagation *is actually hidden* and the computation of each sum bit is based solely on local information and the sum bit of the previous bit position. This *new sum-only recursive bit-level algorithm* allows the design of new *sum-propagate adders* that can be useful in both ASIC and FPGA chips. For instance, in the case of FPGA chips, the introduced sum-propagation chains can readily replace the existing hard carry chains in FPGAs with simpler sum-only chain structures.

To enable fast and regular sum propagate adders, a novel associative parallel prefix operator is introduced that allows the design of a new family of sum-lookahead adders based on the parallel-prefix paradigm.

The introduced formulation of sum-propagation addition allows the design of sum-propagate adders that are asymptotically equivalent, in terms of logic levels and number of gates, to carry-propagate adders. However, their internal structure differs and this leads to measurable performance differences in terms of delay. The operation of sum-propagate adders is based on a series of AND-XOR operations and thus are slower in currently available CMOS standard-cell libraries than the AND-OR operations involved in carry-propagate adders. This result is verified experimentally by synthesizing both sum-propagate and carry-propagate adders with the Cadence digital implementation flow and using commercial grade 28nm standard-cell libraries under two operating voltages.

The delay overhead of sum-propagate adders is expected to reduce in the future with the adoption of multiple-independent

gate nanowire transistors [4] that are promising functionality-enhanced beyond CMOS device technologies [5]. To support this argument, we implemented the same adders under comparison using a 10nm triple-independent-gates CMOS technology [6], [7] and measured the delay of each design in Spectre-Spice. The availability of three-independent gates per transistor reduces the logic depth of sum propagate adders, which in effect reduces the delay gap across various adder architectures.

In overall, the introduced direct sum-propagation paradigm and its parallel-prefix computation, is an interesting theoretical result that can be proven useful: (a) for adder design in future implementation technologies or FPGAs, (b) for the design of approximate adders or in-memory adders and other addition-related operations or even (c) for handling addition in logic synthesis flows that focus on the optimization of AND-XOR logic for preserving security properties [8], such as the ones introduced in [9], [10].

The remainder of the paper is organized as follows: Section II revisits the basics of carry-propagate addition. Section III introduces sum-propagate adders. Section IV organizes sum-lookahead computation in a parallel-prefix structure, while Section V presents the experimental results. Finally, conclusions are drawn in Section VI.

II. BASICS OF CARRY-PROPAGATE ADDERS

When adding two n-bit binary numbers $A = A_{n-1}A_{n-2}\ldots A_0$ and $B = B_{n-1}B_{n-2}\ldots B_0$, the sum bit S_i at the ith bit position is computed by combining the modulo-2 sum (exclusive OR) of bits A_i and B_i, i.e., $H_i = A_i \oplus B_i$ (XOR), and the carry C_{i-1} computed in the previous bit position

$$S_i = H_i \oplus C_{i-1}. \tag{1}$$

For computing the sum bits of the following bit positions, the incoming carry C_{i-1} needs to propagate to the next position. The carry out of the ith bit position C_i is computed using the local carry generate and propagate bits $G_i = A_i B_i$ (AND) and $P_i = A_i + B_i$ (OR) and the fundamental recursive carry propagation formula

$$C_i = G_i + P_i C_{i-1}. \tag{2}$$

For short bit widths, ripple-carry adder structures [2] offer compact implementations. Figure 1 depicts two versions of ripple-carry cells [3]. In Figure 1(a) carry propagation follows the AND-OR logic of (2) where H_i is legally used in place of P_i, while in Figure 1(b) carry out is computed using equivalent MUX-based logic. The latter structure is adopted in FPGA chips that employ dedicated carry chains for carry propagation instead of mapping addition solely on lookup tables (LUTs) [11].

For increased bit widths, carry-skip adders [12], [13] have been proposed that improve the linear growth of carry chain delay by allowing carries to skip across blocks of bits, rather than rippling through them [14].

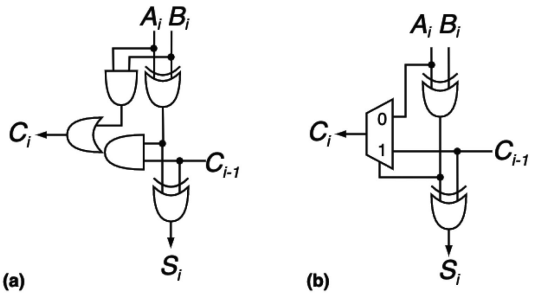

FIGURE 1. (a) Ripple-carry adder using AND-OR carry propagation (b) MUX-based equivalent logic employed at FPGA hard carry chains.

Carry-lookahead adders can reduce the delay further [15]–[17]. In this case, carries are computed in parallel and addition is implemented in logarithmic logic depth. Mapping carry-lookahead adders to parallel-prefix structures maximized their efficiency and increased also their placement and wiring regularity [18]–[23]. Carry computation is transformed to a prefix problem [18] by using the associative operator $\circ$ that associates pairs of generate and propagate bits as follows:

$$(G, P) \circ (G', P') = (G + P\,G', P\,P').$$

In a series of consecutive associations of generate and propagate pairs, $(G_{k:j}, P_{k:j})$ denotes the group generate and propagate term produced out of bits $k, k-1, \ldots, j$

$$(G_{k:j}, P_{k:j}) = (G_k, P_k) \circ (G_{k-1}, P_{k-1}) \circ \cdots \circ (G_j, P_j), \tag{3}$$

and carry C_i corresponds to $G_{i:0}$.

Figure 2 depicts two parallel-prefix carry-propagate adders and Figure 3 highlights the logic-level implementation of their basic blocks. Each adder is organized in three stages. The pre-processing stage computes bits G_i, P_i and H_i. Parallel prefix carry computation is done in the second stage using black and grey nodes that implement (3), whereas the last stage computes the sum bits according to (1).

To avoid the large wiring overhead of some parallel-prefix structures, hybrid designs have been proposed that compute only a subset of the required carries and the intermediate ones are pre-computed in parallel using conditional sum computation [24]–[26]. Conditional sum can be simplified using select-prefix/carry increment blocks [27]. Also, fully unrolling the concept of conditional sum computation [28] allows the design of adders using parallel-prefix structures that consist of multiplexer-based prefix operators.

Ling in [29] simplified the definition of each carry by propagating the OR of two consecutive carries. This allowed the design of reduced logic depth parallel-prefix adders [30] as well as fast ripple-carry adders for small bitwidths [31], [32]. Exploring similar concepts Dimitrakopoulos *et al.* in [33] introduced carries that exhibit lower switching activity than normal carries and can lead to reduced-complexity parallel-prefix structures.

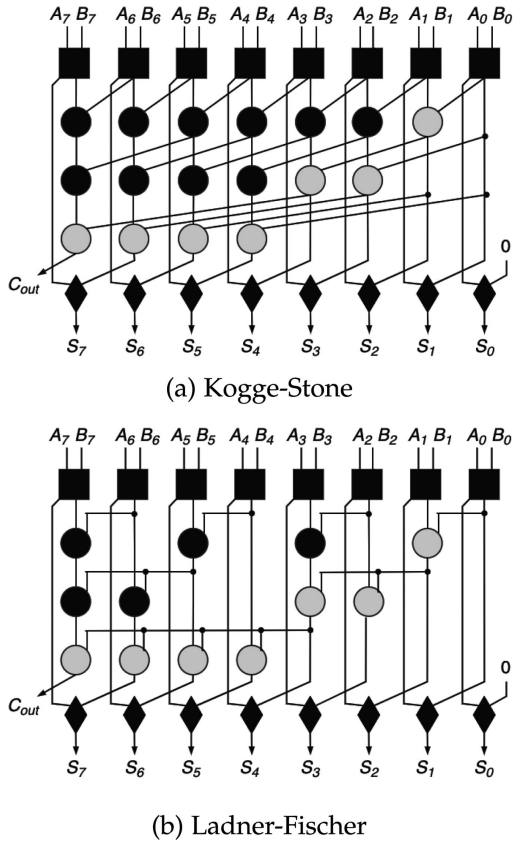

FIGURE 2. The (a) Kogge-Stone[20] and (b) Ladner-Fischer [19] parallel prefix carry-propagate adders.

III. SUM-PROPAGATE ADDITION

To transform binary addition from a carry-propagate operation to a sum-propagate operation, we need to associate the computation of each sum bit S_i directly with the sum bit of the previous bit position S_{i-1} without using a carry bit as an intermediate. This is achieved by Lemma 1, assuming that boolean AND takes precedence of boolean XOR operation, i.e., $a \oplus bc$ should be read as $a \oplus (bc)$.

Lemma 1. $S_i = X_i \oplus H_{i-1} S_{i-1}$, *where* $X_i = H_i \oplus P_{i-1}$.

Proof. First, it will be helpful to express the fundamental carry operation (2), in a XOR-AND form. To do this we use the property that for any two bits a, b it holds that $a + b = a \oplus b \oplus ab$

$$C_i = G_i + P_i C_{i-1} = G_i \oplus P_i C_{i-1} \oplus G_i P_i C_{i-1}.$$

Since $G_i P_i = G_i$ and $P_i \oplus G_i = H_i$

$$\begin{aligned} C_i &= G_i \oplus P_i C_{i-1} \oplus G_i C_{i-1} = G_i \oplus (P_i \oplus G_i) C_{i-1} \\ &= G_i \oplus H_i C_{i-1}. \end{aligned} \tag{4}$$

Second, using (1), we can express the carry bits C_i and C_{i-1} as a function of the corresponding sum bits they are meant to compute. For instance, by adding modulo-2 (xor) the term H_i to both sides of (1) we get

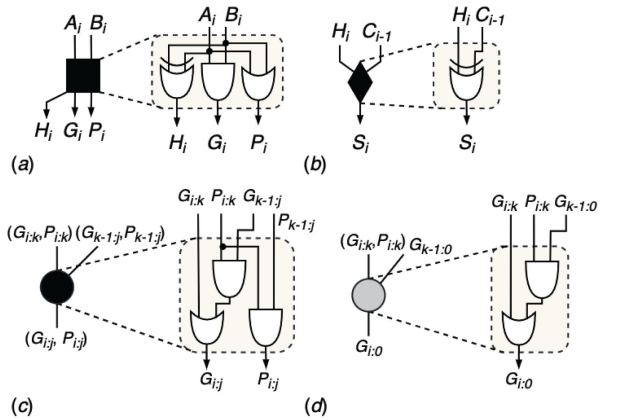

FIGURE 3. The blocks used in the (a) first and (b) in the last stage of a parallel-prefix adder. (c) The logic-level implementation of the prefix operator ∘. (d) The simplified operator ∘ used in the last node of each column.

$$H_i \oplus S_i = H_i \oplus H_i \oplus C_{i-1}.$$

Simplifying H_i from the right side we get that

$$H_i \oplus S_i = C_{i-1}. \tag{5}$$

Equivalently, for the next bit position we can write that

$$C_i = S_{i+1} \oplus H_{i+1}. \tag{6}$$

Replacing (5) and (6) to the xor-based recursive carry formula (4) and recognizing that $G_i \oplus H_i = P_i$ we get

$$\begin{aligned} S_{i+1} \oplus H_{i+1} &= G_i \oplus H_i (S_i \oplus H_i) \\ &= G_i \oplus H_i \oplus H_i S_i \\ &= P_i \oplus H_i S_i. \end{aligned} \tag{7}$$

Adding modulo-2 (xor) the term H_{i+1} in both sides of (7) and simplifying H_{i+1} from the left side, we get

$$S_{i+1} = H_{i+1} \oplus P_i \oplus H_i S_i.$$

Defining $X_{i+1} = H_{i+1} \oplus P_i$ (or equivalently $X_i = H_i \oplus P_{i-1}$) we can write the sum bit of the ith bit position as $S_i = X_i \oplus H_{i-1} S_{i-1}$. □

Lemma 1 manages to remove the dependency of addition on carry bits, while the actual information of carry propagation is handled implicitly by the direct propagation of previous sum bits. The addition is carry free in the sense that no carry is generated or propagated, while still the addition remains non-redundant. The arithmetic example in Figure 4 shows the values of the intermediate signals H_i, P_i and X_i needed for the recursive computation of the sum bits. As highlighted in Figure 4, the sum bit at position 4, S_4, is computed using X_4, H_3 and the previous sum bit S_3. For bit position 0, we assume that $H_{-1} = P_{-1} = 0$.

The most primitive form of a binary adder that can be designed using this new formulation is the ripple-sum adder. Figure 5 depicts a 4-bit ripple-sum adder. The H and P bits

	7	6	5	4	3	2	1	0	
A:	0	0	1	0	0	1	0	0	
B:	0	1	1	0	1	1	1	0	
H:	0	1	0	0	1	0	1	0	$H_i = A_i \oplus B_i$
P:	0	1	1	0	1	1	1	0	$P_i = A_i + B_i$
X:	1	0	0	1	0	1	1	0	$X_i = H_i \oplus P_{i-1}$
S:	1	0	0	1	0	0	1	0	$S_i = X_i \oplus H_{i-1}\, S_{i-1}$

FIGURE 4. Arithmetic example of adding two 8-bit numbers using the recursive formula of Lemma 1.

are computed directly from the input bits in one logic level (G bits used in a carry-propagate adder are not needed). Then those signals are used to compute the X bits according to the definition in Lemma 1. The computation of the ith sum bit S_i requires the pair (X_i, H_{i-1}) and the sum bit computed in the previous bit position S_{i-1}. To achieve this, the sum produced at each bit position drives the adder's output and feeds also the next bit position in a ripple manner.

A. CARRY INTERFACES

For backward compatibility sum-propagate adders need to support carry in and out interfaces. The easiest way to incorporate a carry-in signal to the proposed sum-propagate adders is to connect it to P_{-1}. By setting $P_{-1} = C_{\text{in}}$ and keeping $H_{-1} = 0$, we get that

$$S_0 = X_0 = H_0 \oplus P_{-1} = H_0 \oplus C_{\text{in}}, \tag{8}$$

which matches the definition of the least-significant sum bit according the basic definition (1). In this way, the carry-in signal is incorporated to the computation of the rest sum bits, since they are recursively derived from S_0, following Lemma 1.

A carry out bit can be derived by the P and H bits of the most significant bit position and the most significant sum bit according to Lemma 2.

Lemma 2. $C_{out} = C_{n-1} = P_{n-1} \oplus H_{n-1} S_{n-1}$

Proof. From (6) $C_{n-1} = S_n \oplus H_n$. Replacing S_n with its definition from Lemma 1, we get that

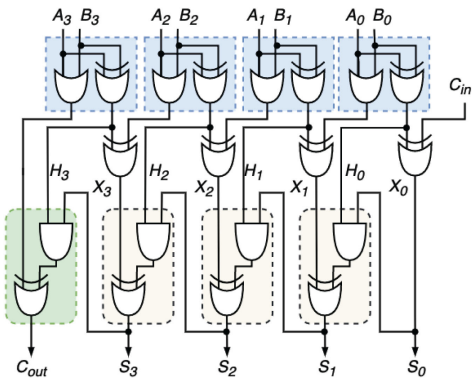

FIGURE 5. A 4-bit ripple-sum adder. Addition is based solely on sum bit propagation. Carry input/output interfaces also included.

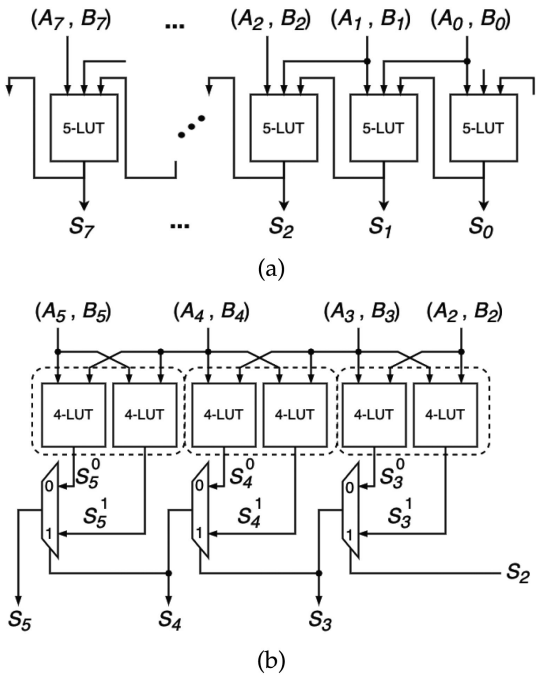

FIGURE 6. (a)Ripple-sum adder mapped in a series of 5-input LUTs (b) Sum-propagation multiplexer chain that can replace FPGA carry chains.

$$C_{n-1} = (X_n \oplus H_{n-1}\, S_{n-1}) \oplus H_n.$$

Expanding the definition of X_n and simplifying redundant terms H_n, we can get the needed result

$$\begin{aligned} C_{n-1} &= H_n \oplus P_{n-1} \oplus H_{n-1}\, S_{n-1} \oplus H_n \\ &= P_{n-1} \oplus H_{n-1}\, S_{n-1}. \end{aligned}$$

□

Figure 5 highlights also the extra circuit needed in the least and the most-significant bit position for handling the carry-in and carry-out signals, respectively.

B. RIPPLE SUM ADDERS FOR FPGAS

The introduced sum-propagate adders open up new possibilities on how adders can be implemented in FPGAs. At the moment, FPGAs, besides the adders built inside DSP blocks, can implement binary addition either using only LUTs or using a combination of LUTs and the built-in carry chains for higher speed [11].

In the first case, of using only LUTs for implementing addition, the removal of carry propagation allows the design of ripple-sum adders built only with 5-input LUTs as shown in Figure 6. The sum bits at the output of each LUT can feed the next 5-input LUT without any additional logic.

In the second case, we argue that hard carry chains are redundant and can be replaced by equivalent sum-propagate chains thus removing the need for additional logic that would compute sum bits from the computed carries. Actually, the multiplexer structures used in those carry chains can be

replaced by equivalent multiplexer-based structures for propagating directly the corresponding sum bits. Using two 4-input LUTs per bit position, one can produce from Lemma 1 two speculative variants of each sum bit, i.e., S_i^0 and S_i^1, assuming that S_{i-1} is equal to 0 or 1, respectively

$$S_i^0 = X_i \quad \text{and} \quad S_i^1 = X_i \oplus H_{i-1}.$$

Then, as shown in Figure 6(b), the correct sum bit is derived by selecting one of the two speculative results. The selection is done by the sum bit of the previous bit position that drives the select signal of the corresponding multiplexer.

IV. PARALLEL PREFIX SUM LOOKAHEAD ADDERS

Unrolling the recursive definition of sum bits given in Lemma 1 allows the design of sum-lookahead adders. In this case, the sum bits are computed in parallel directly from the input bits without the need for ripple-like sum propagation

$$S_i = X_i \oplus \left(\bigoplus_{j=0}^{i-1} \left(\prod_{k=j}^{i-1} H_k \right) X_j \right). \tag{9}$$

For example, in the case of a 4-bit adder the sum bits are given by the following equations:

$$\begin{aligned} S_0 &= X_0 \\ S_1 &= X_1 \oplus H_0 X_0 \\ S_2 &= X_2 \oplus H_1 X_1 \oplus H_1 H_0 X_0 \\ S_3 &= X_3 \oplus H_2 X_2 \oplus H_2 H_1 X_1 \oplus H_2 H_1 H_0 X_0. \end{aligned}$$

In traditional carry-propagate adders, parallel-prefix carry computation introduces structure and regularity to the lookahead computation of carries.

We would like to take advantage of the favorable properties of parallel-prefix computation to design efficient and regular sum-lookahead adders. To do this, we introduce the operator $\odot$, defined as

$$(x, h) \odot (x', h') = (x \oplus h\,x', h\,h'). \tag{10}$$

Then, by leveraging operator $\odot$ and Lemmas 3 and 4, we can compute the sum bits following the parallel-prefix paradigm.

Lemma 3. Let

$$(X_{i:0}, H_{i-1:0}) = \begin{cases} (X_0, H_{-1}), & \text{if } i = 0 \\ (X_i, H_{i-1}) \odot (X_{i-1:0}, H_{i-2:0}), & \text{if } 1 \le i \le n-1 \end{cases},$$

then $S_i = X_{i:0}$, *for all* $i = 0, 1, \ldots, n-1$.

Proof. We prove the lemma by induction on i. For $i = 0$, $S_0 = X_0 = H_0$ which is true according to (1) assuming a carry-in (C_{-1}) equal to zero and $H_{-1} = P_{-1} = 0$.

For $i > 1$, let's assume that $S_{i-1} = X_{i-1:0}$. Then

$$\begin{aligned} (X_{i:0}, H_{i-1:0}) &= (X_i, H_{i-1}) \odot (X_{i-1:0}, H_{i-2:0}) \\ &= (X_i, H_{i-1}) \odot (S_{i-1}, H_{i-2:0}) \\ &= (X_i \oplus H_{i-1}\,S_{i-1}, H_{i-1}\,H_{i-2:0}). \end{aligned}$$

Thus $X_{i:0} = X_i \oplus H_{i-1}\,S_{i-1}$. Since, according to Lemma 1, $S_i = X_{i:0}$ the result follows by induction. □

Lemma 4. The operator $\odot$ *is associative*

Proof. For any $(X_3, H_2) \odot (X_2, H_1) \odot (X_1, H_0)$ we have

$$\begin{aligned} &[(X_3, H_2) \odot (X_2, H_1)] \odot (X_1, H_0) \\ &\quad = (X_3 \oplus H_2 X_2, H_2 H_1) \odot (X_1, H_0) \\ &\quad = (X_3 \oplus H_2 X_2 \oplus H_2 H_1 X_1, H_2 H_1 H_0) \end{aligned}$$

$$\begin{aligned} &(X_3, H_2) \odot [(X_2, H_1) \odot (X_1, H_0)] \\ &\quad = (X_3, H_2) \odot (X_2 \oplus H_1 X_1, H_1 H_0) \\ &\quad = (X_3 \oplus H_2 X_2 \oplus H_2 H_1 X_1, H_2 H_1 H_0). \end{aligned}$$

The right-hand sides of both expressions are equal. □

Using the operator $\odot$ and the definition of Lemma 3, in a series of consecutive associations of X and H bits the notation $(X_{k:j}, H_{k-1:j-1})$ is used to denote the group term produced out of bits $k, k-1, \ldots, j$

$$(X_{k:j}, H_{k-1:j-1}) = (X_k, H_{k-1}) \odot (X_{k-1}, H_{k-2}) \odot \cdots \odot (X_j, H_{j-1}).$$

According to Lemma 1, for the computation of the ith sum bit S_i only the term $X_{i:0}$ is needed. For example, in the case of 4 bits, the four sum bits can be computed using the $\odot$ operator as follows:

$$\begin{aligned} S_0 &: (X_0, H_{-1}) \\ S_1 &: (X_1, H_0) \odot (X_0, H_{-1}) \\ S_2 &: (X_2, H_1) \odot (X_1, H_0) \odot (X_0, H_{-1}) \\ S_3 &: (X_3, H_2) \odot (X_2, H_1) \odot (X_1, H_0) \odot (X_0, H_{-1}). \end{aligned}$$

Figure 7 presents six variants of the new 8-bit parallel-prefix sum-propagate adders using the topologies proposed by Ladner-Fischer [19], Kogge-Stone [20], one representative of the Knowles' adders [21] and Roy [23] for parallel carry computation. Also, a Han-Carlson [34] and a sum-increment adder [2] (analogous to carry-increment topologies) are presented.

The logic-level implementation of the basic cells used in a parallel-prefix sum-propagate adder is shown in Figure 8. The first stage computes in parallel, bits P_i and H_i, while the second stage, computes in one logic level the pairs (X_i, H_{i-1}). Then, the sum bits are produced in a parallel-prefix manner using $\log_2 n$ levels. The last node of each bit column requires a simpler implementation of the $\odot$ operator since only a group X term of the form $X_{i:0}$ needs to be computed.

The proposed parallel-prefix sum-propagate adders can take many forms leading to a whole family of parallel-prefix adders. However, not all structures are possible since the operator $\odot$ *is not idempotent*, i.e., $(x, h) \odot (x, h) \neq (x, h)$.

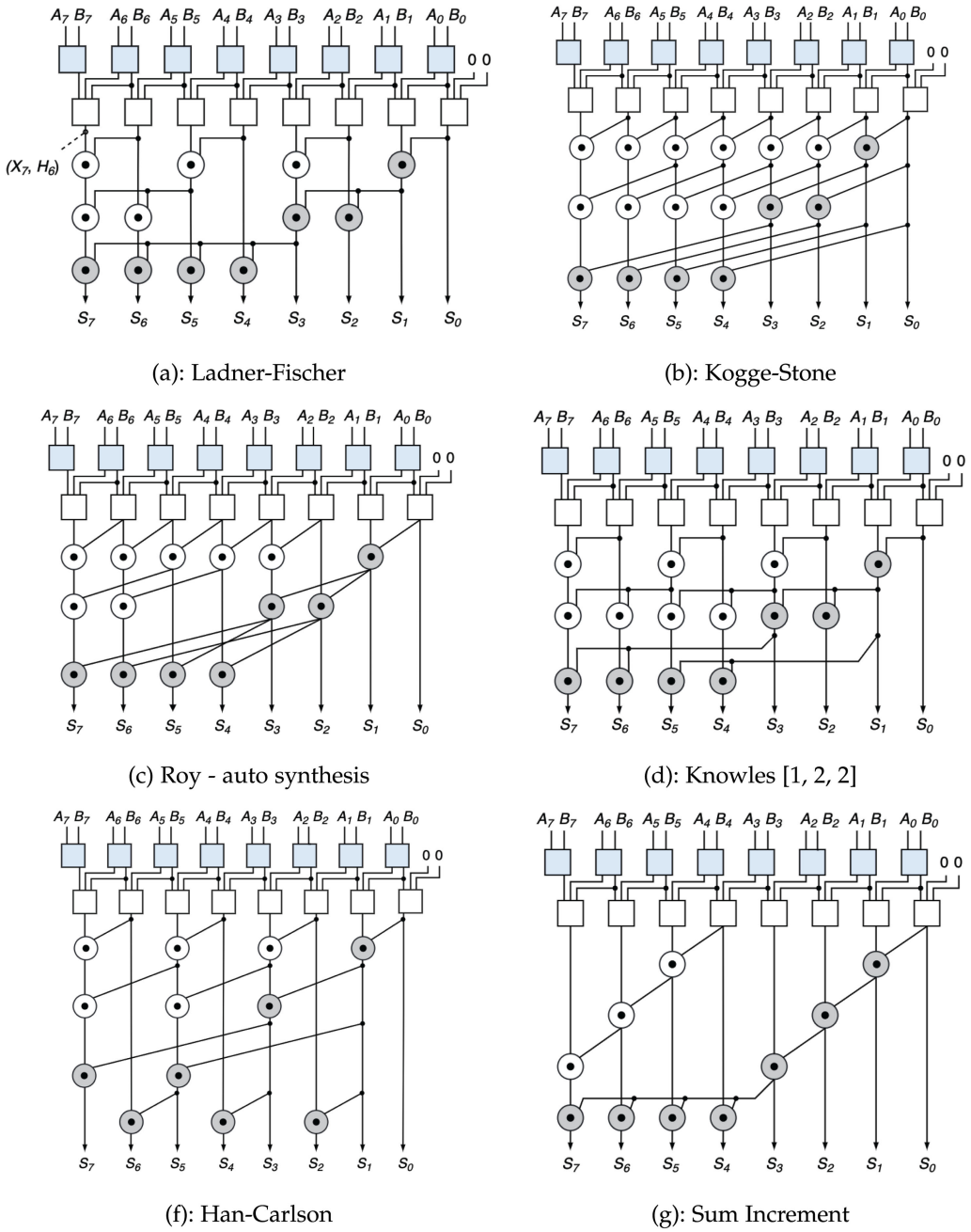

FIGURE 7. Examples of 8-bit sum-propagate parallel-prefix adders.

Therefore, parallel-prefix structures that compute overlapping groups of $(X_{i:j}, H_{i-1:j-1})$ should be avoided. For instance, in Figure 9(a) depicts a valid parallel-prefix structure, while the one presented in Figure 9(b) is illegal, since for the computation of group 2:0 it associates the terms 2:1 and 1:0 that overlap at bit position 1.

Finally, it should be noted that although the proposed parallel-prefix sum-lookahead adders resemble structurally the traditional carry-lookahead adders and they can be built using the same construction principles, their logic-level netlists are completely different. Their fundamental differences can be easily identified by Figure 10 that presents the unfolded logic-level implementation of an 8-bit Kogge-Stone sum-propagate adder.

V. EXPERIMENTAL RESULTS

The experimental results aim at highlighting the delay performance of conventional parallel-prefix carry-propagate (*CP*) adders and the introduced sum-propagate (*SP*) adders under two scenarios: In the first case, we employed a well-established digital design flow using a commercial-grade standard cell library. In the second case, the adders under comparison were implemented with multiple-independent gate nanowire transistors [4] that are promising functionality-enhanced

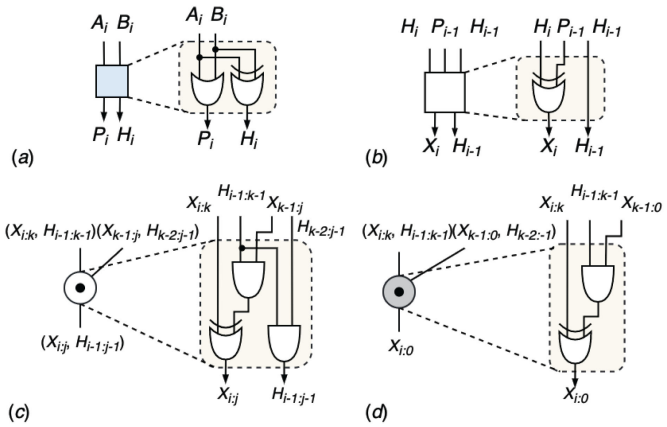

FIGURE 8. **The logic-level implementation of the basic blocks used to construct parallel-prefix sum-propagate adders.**

beyond CMOS device technologies and their delay was measured in Spice.

A. UNIT GATE MODEL ANALYSIS

Before moving on the actual hardware measurements, it would be interesting to discuss analytically the properties of SP adders relative to their CP counterparts. Asymptotically, SP and CP adders have the same properties. For instance, sum bits can be computed in as low as $2 + 2\log_2 n$ logic levels using minimum depth parallel-prefix SP or CP adders, assuming that each 2-input gate counts as one independent of its type. This can be easily verified by comparing the CP and SP Kogge-Stone adders of Figures 2(a) and 7(b), respectively.

Also, SP and CP adders can be built using the same number of 2-input gates. In both cases, every prefix operator consists of three or two gates and introduces a delay of two gates in series. This can be easily verified by the 8-bit example SP adder shown in Figure 10. The depicted adder consists of 70 gates which is equal to the number of gates used by the Kogge-Stone CP adder shown in Figure 2(a).

However, such favorable properties do not translate to equivalent performance in practice, since the basic AND-XOR sum propagation operation is fundamentally slower than the basic AND-OR carry propagation operation in current CMOS implementations. Also, implementing AND-XOR with discrete gates adds more transistors in overall than the combined AND-OR gates (either AND-OR-Invert or OR-AND-Invert) that can be used in the CP adders.

B. LOGIC SYNTHESIS RESULTS

To quantify the area-delay gap of SP adders with respect to CP adders, various SP and CP adder architectures were implemented in Verilog and synthesised to a commercial 28 nm standard-cell library using Cadence Genus. Synthesis was performed for two different operating voltages: a typical case of 1V and a low-voltage scenario of 0.72V. In both cases, cells with only regular threshold voltage were employed. To apply delay constraints in a uniform manner and enable an equivalent output loading for all adders under comparison, all inputs and outputs are assumed to be registered.

The minimum achievable delay of each architecture for various bitwidths and the area that corresponds to this delay point are depicted in Table 1. In all cases, SP adders are slower than CP adders. The reason for this delay gap is threefold. First, the AND-XOR operation involved in SP adders is inherently more complex than the AND-OR logic needed in CP adders. Second, AND-OR gates in standard-cell libraries are highly optimized logic cells relative to XOR gates that are considered less often used. Third, the intermediate representations used in logic synthesis tools [35] promote sum-of-product implementations relative to the direct synthesis of XOR logic thus missing additional possible optimizations [36].

For smaller bitwidths, where even ripple-carry/sum architectures make sense, SP adders are still slower. For instance, an 8-bit ripple-sum adder has a worst-case delay of 376 ps, while an equivalent 8-bit ripple-carry adder has a worst-case delay of 290 ps.

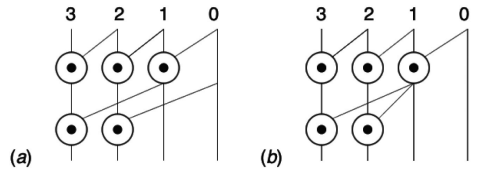

FIGURE 9. **Idempotency does not hold for the operator $\odot$. (a) Valid connection with no overlap (b) Invalid connectivity.**

C. ADDER DESIGN WITH THREE-INDEPENDENT-GATE FIELD-EFFECT TRANSISTORS

Here, we assess the delay performance of parallel-prefix adders leveraging Three-Independent-Gate Field-Effect Transistors (TIGFETs). The particular technology allows for increased transistor density by combining multiple MOS gates on a single device [37]–[40].

TIGFET devices provide three independent gate terminals, namely, center gate (CG), polarity gate at source (PGS) and polarity gate at drain (PGD), as depicted in Figure 11(a). Specifically, the PGS and PGD terminals configure the Schottky junction's effective barrier height and allow either holes or electrons to flow through the channel of the device. Therefore, the device is configured either as an *n*-type or a *p*-type device by setting

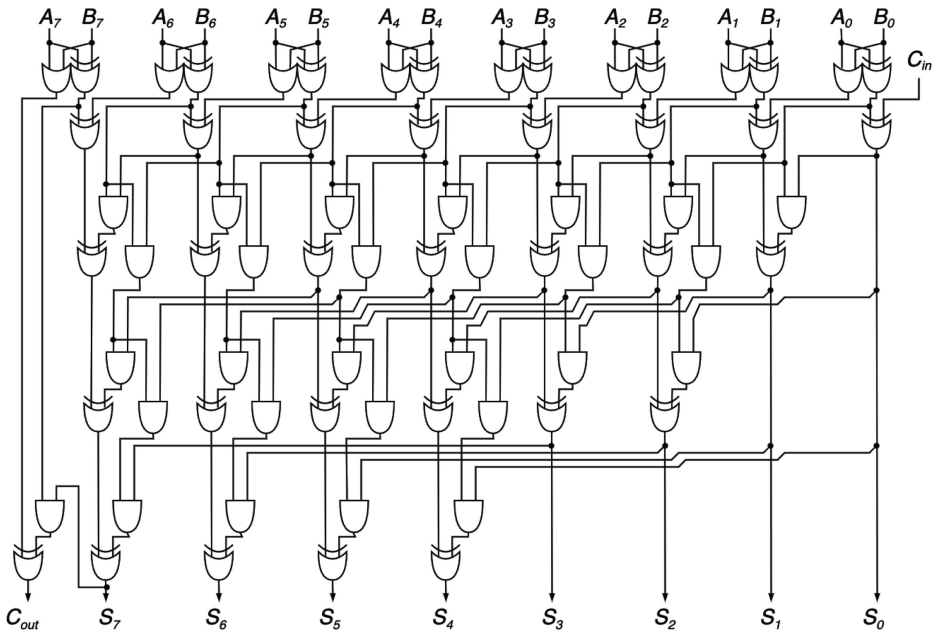

FIGURE 10. The gate-level implementation of an 8-bit Kogge-Stone sum-propagate adder.

TABLE 1. The minimum achievable delay of all adders under comparison and their corresponding area after logic synthesis at 28nm.

Architecture	16-bit				32-bit				64-bit			
	Delay (ps)		Area (μm^2)		Delay (ps)		Area (μm^2)		Delay (ps)		Area (μm^2)	
1 V	SP	CP	SP	CP	SP	CP	SP	CP	SP	CP	SP	CP
Kogge-Stone	309	236	155	115	352	272	284	262	400	292	694	675
Knowles [1,2,2,2]	305	240	120	101	356	268	300	262	399	296	725	638
Ladner-Fischer	308	244	97	89	355	269	208	202	405	302	473	451
Brent-Kung	370	277	84	68	448	320	170	148	534	369	377	299
Han-Carlson	337	254	108	88	391	283	212	203	426	314	465	438
0.72 V	SP	CP	SP	CP	SP	CP	SP	CP	SP	CP	SP	CP
Kogge-Stone	1283	1052	223	118	1438	1150	546	313	1626	1253	1221	693
Knowles [1,2,2,2]	1270	1067	171	111	1444	1158	421	285	1629	1277	981	654
Ladner-Fischer	1305	1055	138	100	1447	1173	307	213	1641	1298	679	475
Brent-Kung	1487	1199	117	76	1790	1366	232	165	2077	1550	496	327
Han-Carlson	1350	1130	164	98	1554	1218	354	235	1716	1330	914	520

the polarity gates at high or low logic state, respectively [5]. When all of CG, PGS and PGD terminals are biased on the same potential, the device turns on and carriers flow through the channel, while remainder bias condition lead to an off-state for the device (cf. [41, Figure 2]). When a TIGFET device is biased at an input and controlled by the other two, it operates as two series transistors, offering the possibility of a mapping between MOSFET and TIGFET devices. For example, Figure 11(b) depicts the two series *n*-type MOSFETs equivalent of a TIGFET device with the PGD terminal at high logic state.

The proposed 16-bit SP and CP parallel-prefix adders are implemented at a 10-nm TIGFET technology [6] publicly available in [42]. The most interesting cells for implementing both types of adders are depicted in Figure 12. Figure 12(a) depicts a 2-input XOR gate used in both types of adders, Figure 12(b) shows the simplified implementation of an AND-OR gate (its inverted version) used in the parallel prefix operator of CP addition, while Figure 12(c) shows the single-gate implementation of the AND-XOR operation used in the parallel-prefix operator of an SP adder. In all cases, stacked devices are completely avoided with the requirement that both non-inverting and inverting inputs (generated locally) are available.

The delay performance of the TIGFET-based adder implementation is assessed using 16-bit parallel-prefix SP and CP adders. The worst-case delay of its design is depicted in Figure 13. Delay measurements are performed in Cadence

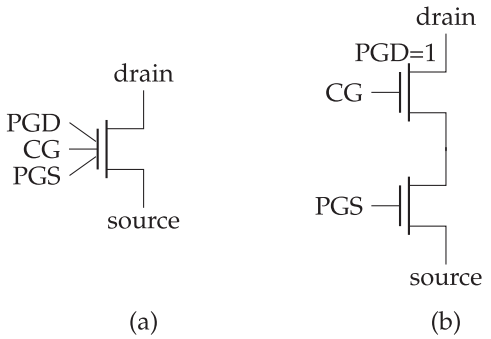

FIGURE 11. (a) TIGFET device symbol and (b) TIGFET device operating as two series *n*-type MOSFETs by setting PGD=1.

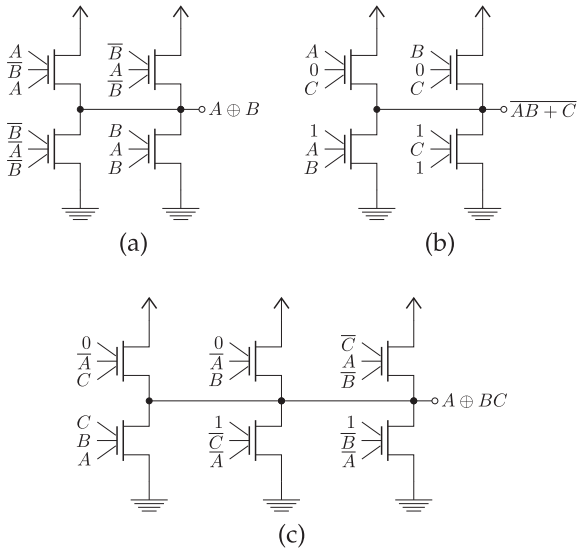

FIGURE 12. TIGFET-based transistor-level implementations of the main cells required for parallel-prefix SP and CP adders.

Spectre Simulator using appropriate pairs of input vectors and assuming a nominal supply voltage of 0.7 V, according to the available transistor models [6] and a fanout load of a minimum-sized inverter per sum output.

The delay differences between SP and CP adders are mainly attributed to the increased delay of the AND-XOR cell used in SP adders relative to the AND-OR cell used in CP adders. From the transistor-level implementation of the basic AND-XOR and AND-OR cells, it is evident that the AND-XOR operation is performed using only parallel transistors as in the case of an AND-OR cell, but with two more branches. These extra transistors increase both the capacitance of the output node as well as the input capacitance of the cell. In this way, and AND-XOR cell has both more output capacitance to drive and incurs also more delay to the previous stage that drives it.

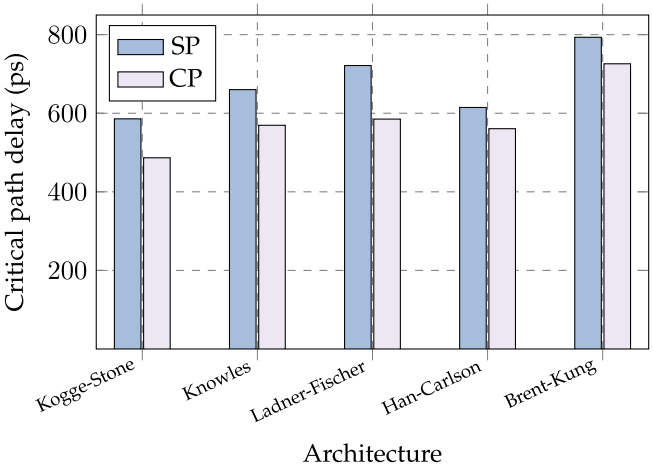

FIGURE 13. Worst-case delay of 16-bit TIGFET-based parallel-prefix adders at a 10-nm technology node and 0.7 V. Han-Carlson design has a smaller delay than Knowles and Ladner-Fischer adders, despite that Han-Carlson is comprised of an extra prefix stage, due to its lower internal signal fanout.

VI. CONCLUSION

This work redefined the decades-old carry propagate recursion using an equivalent sum-propagate recursion that works directly on sum bits. Based on this fundamental new formulation, various sum-propagate adder architectures have been presented that cover most of adder design space: from ripple sum adders to parallel prefix sum-lookahead architectures. The presented adders open up a whole new design space that can be proven useful in current and future implementation technologies. Also, the proposed formulation allows the replacement of the dedicated carry chains in FPGA chips with equivalent sum chains that can work directly on the sum bits without the need to introduce carry signals as an intermediate for computing fast the result of the addition.

At the moment, and using current state-of-the-art implementation technologies sum-propagate are slower than their carry-propagate counterparts. In the future, we expect this gap to diminish with the adoption of multi-gate transistors, examined in this work, and the enhancement of the intermediate representations of logic synthesis tools [10], [36] that would treat equally exclusive-or-rich logic.

In overall, we believe that this new formulation would help redefine several problems that are directly connected to carry propagation. For instance, solutions to well-known problems such as leading-zero anticipation [43] and parity prediction of addition [44] could be possibly revised and enhanced. Also, new approximate adders [45] with more tight error bounds are expected to be designed, while the in-memory computation of addition [46] can be possibly simplified by omitting the computation of carry bits and reusing previously computed sum bits.

REFERENCES

[1] B. Parhami, *Computer Arithmetic - Algorithms and Hardware Designs*. New York, NY, USA: Oxford Univ. Press, 2000.

[2] N. Weste and D. Harris, *CMOS VLSI Design a Circuits and Systems Perspective*, 3rd ed. Boston, MA, USA: Addison Wesley, 2010.

[3] V. G. Oklobdzija, "High-speed VLSI arithmetic units: Adders and multipliers," in *Design of High-Performance Microprocessor Circuits*.New York, NY, USA: IEEE Press, 2001, pp. 181–204.

[4] J. Trommer *et al.*, "Reconfigurable nanowire transistors with multiple independent gates for efficient and programmable combinational circuits," in *Proc. Des., Automat. Test Eur. Conf. Exhib.*, 2016, pp. 169–174.

[5] G. V. Resta, A. Leonhardt, Y. Balaji, S. De Gendt, P. Gaillardon, and G. De Micheli, "Devices and circuits using novel 2-D materials: A perspective for future VLSI systems," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 27, no. 7, pp. 1486–1503, Jul. 2019.

[6] G. Gore, P. Cadareanu, E. Giacomin, and P. Gaillardon, "A predictive process design kit for three-independent-gate field-effect transistors," in *Proc. IFIP/IEEE 27th Int. Conf. Very Large Scale Integration*, 2019, pp. 172–177.

[7] A. Antidormi, S. Frache, M. Graziano, P. Gaillardon, G. Piccinini, and G. De Micheli, "Computationally efficient multiple-independent-gate device model," *IEEE Trans. Nanotechnol.*, vol. 15, no. 1, pp. 2–14, Jan. 2016.

[8] J. Boyar, P. Matthews, and R. Peralta, "Logic minimization techniques with applications to cryptology," *J. Cryptol., vol.* 26, pp. 280–312, 2013.

[9] E. Testa, M. Soeken, L. Amarù, and G. De Micheli, "Reducing the multiplicative complexity in logic networks for cryptography and security applications," in *Proc. 56th Annu. Des. Automat. Conf.*, 2019, pp. 1–6.

[10] E. Testa, M. Soeken, H. Riener, L. Amaru, and G. D. Micheli, "A logic synthesis toolbox for reducing the multiplicative complexity in logic networks," in *Proc. Des., Automat. Test Eur. Conf. Exhib.*, 2020, pp. 568–573.

[11] K. E. Murray *et al.*, "Optimizing FPGA logic block architectures for arithmetic," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 6, pp. 1378–1391, Jun. 2020.

[12] M. Lehman and N. Burla, "Skip techniques for high-speed carry-propagation in binary arithmetic units," *IRE Trans. on Electron. Comput.*, vol. EC-10, no. 4, pp. 691–698, Dec. 1961.
[13] V. Kantabutra, "Designing optimum one-level carry-skip adders," *IEEE Trans. Comput.*, vol. 42, no. 6, pp. 759–764, Jun. 1993.
[14] I. Koren, *Computer Arithmetic Algorithms*.Natick, MA, USA: A. K. Peters, 2002.
[15] J. L. S. Weinberger, "A logic for high-speed addition," in *Proc. Nat. Bureau Stand. Circ. 591*, 1958, pp. 3–12.
[16] B. Lee and V. Oklobdzija, "Improved CLA scheme with optimized delay," *J. VLSI Sign Process. Syst. Signal, Image Video Technol.*, no. 3, pp. 265–274, 1991.
[17] J. P. Fishburn, "A depth-decreasing heuristic for combinational logic; or how to convert a ripple-carry adder into a carry-lookahead adder or anything in-between," in *Proc. 27th ACM/IEEE Des. Automat. Conf.*, 1990, pp. 361–364.
[18] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Trans. Comput.*, vol. C-31, no. 3, pp. 260–264, Mar. 1982.
[19] R. E. Ladner and M. J. Fisher, "Parallel prefix computation," *J. ACM*, vol. 27, no. 4, pp. 831–838, Oct. 1980.
[20] 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, pp. 786–792, Aug. 1973.
[21] S. Knowles, "A family of adders," in *Proc. IEEE Symp. Comput. Arithmetic*, Apr. 1999, pp. 30–34.
[22] R. Zimmermann, "Binary adder Architectures for cell-based VLSI and their synthesis," Ph.D. dissertation, Swiss Federal Inst. Technol. Zurich, 1998.
[23] S. Roy, M. Choudhury, R. Puri, and D. Z. Pan, "Towards optimal performance-area trade-off in adders by synthesis of parallel prefix structures," in *Proc. 50th ACM/EDAC/IEEE Des. Automat. Conf.*, 2013, pp. 1–8.
[24] T. Lynch and E. E. Swartzlander, "A spanning tree carry lookahead adder," *IEEE Trans. Comput.*, vol. 41, no. 8, pp. 931–939, Aug. 1992.
[25] S. K. Mathew, M. A. Anders, B. Bloechel, T. Nguyen, R. K. Krishnamurthy, and S. Borkar, "A 4-GHz 300-mW 64-bit integer execution ALU with dual supply voltages in 90-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 44–51, Jan. 2005.
[26] A. Beaumont-Smith and C. C. Lim, "Parallel-prefix adder design," in *Proc. IEEE Symp. Comput. Arithmetic*, 2001, pp. 218–225.
[27] A. Tyagi, "A reduced-area scheme for carry-select adders," *IEEE Trans. Comput.*, vol. 42, no. 10, pp. 1163–1170, Oct. 1993.
[28] H. Lindkvist and P. Andersson, "Techniques for fast CMOS-based conditional sum adders," in *Proc. IEEE Int. Conf.Comput. Des.: VLSI Comput. Processors*, 1994, pp. 626–635.
[29] H. Ling, "High-speed binary adder," *IBM J. Res. Develop.*, vol. 25, no. 3, pp. 156–166, Mar. 1981.
[30] G. Dimitrakopoulos and D. Nikolos, "High-speed parallel-prefix VLSI ling adders," *IEEE Trans. Comput.*, vol. 54, no. 2, pp. 225–231, Feb. 2005.
[31] N. Burgess, "Fast ripple-carry adders in standard-cell CMOS VLSI," in *Proc. IEEE Symp. Comput. Arithmetic*, 2011, pp. 103–111.
[32] J. Grad and J. E. Stine, "New algorithms for carry propagation," in *Proc. ACM Great Lakes Symp. VLSI*, 2005, pp. 396–399.
[33] G. Dimitrakopoulos, P. Kolovos, P. Kalogerakis, and D. Nikolos, "Design of high-speed low-power parallel-prefix VLSI adders," in *Proc. Power Timing Model., Optim. Simul.*, 2004, pp. 248–257.
[34] T. Han and D. Carlson, "Fast area-efficient VLSI Adders," in *Proc. IEEE Symp. Comput. Arithmetic*, 1987, pp. 49–56.
[35] D. S. Marakkalage, E. Testa, H. Riener, A. Mishchenko, M. Soeken, and G. De Micheli, "Three-input gates for logic synthesis," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Sys.*, early access, 2020, doi: 10.1109/TCAD.2020.3032625.
[36] A. K. Verma and P. Ienne, "Improving XOR-dominated circuits by exploiting dependencies between operands," in *Proc. Asia South Pacific Des. Automat. Conf.*, 2007, pp. 601–608.
[37] J. Zhang, X. Tang, P.-E. Gaillardon, and G. De Micheli, "Configurable circuits featuring dual-threshold-voltage design with three-independent-gate silicon nanowire FETs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 10, pp. 2851–2861, Oct. 2014.
[38] D. Vana, P. Gaillardon, and A. Teman, "C^2TIG: Dynamic C^2MOS design based on three-independent-gate field-effect transistors," *IEEE Trans. Nanotechnol.*, vol. 19, pp. 123–136, Jan. 2020.
[39] E. Giacomin, J. R. Gonzalez, and P. Gaillardon, "Low-power multiplexer designs using three-independent-gate field effect transistors," 2017, pp. 33–38.
[40] H. G. Mohammadi, P.-E. Gaillardon, J. Zhang, G. D. Micheli, E. Sanchez, and M. S. Reorda, "A fault-tolerant ripple-carry adder with controllable-polarity transistors," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 2, Dec. 2016, Art. no. 16.
[41] J. Romero-González and P.-E. Gaillardon, "An efficient adder architecture with three-independent-gate field-effect transistors," in *Proc. IEEE Int. Conf. Rebooting Comput.*, 2018, pp. 1–8.
[42] A 10-nm TIGFET PDK. Accessed: Jan., 2021. [Online]. Available: https://github.com/LNIS-Projects/TIGFET-10nm-model
[43] M. S. Schmookler and K. J. Nowka, "Leading zero anticipation and detection: A comparison of methods," in *Proc. 15th IEEE Symp. Comput. Arithmetic*, 2001, pp. 7–12.
[44] M. Nicolaidis, "Carry checking/parity prediction adders and ALUs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 1, pp. 121–128, Feb. 2003.
[45] H. Jiang, J. Han, and F. Lombardi, "A comparative review and evaluation of approximate adders," in *Proc. Great Lakes Symp. VLSI*, 2015, pp. 343—348.
[46] A. Velasquez and B. Shaia, "Spatially efficient in-memory addition through destructive and non-destructive operations," *IEEE Intern. Symp. Circuits Syst.*, 2019, pp. 1–5.

GIORGOS DIMITRAKOPOULOS received the BS, MS, PhD degrees in computer engineering from the University of Patras, Patras, Greece, in 2001, 2003, and 2007, respectively. He is currently an associate professor with the Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece. His research interests include the design of digital integrated circuits, electronic design automation, computer arithmetic and computer architecture, with emphasis in the design of energy efficient systems.

KLEANTHIS PAPACHATZOPOULOS received the diploma degree in electrical and computer engineering and the MSc degree in integrated hardware-software systems in 2016 and 2018, respectively, from the University of Patras, Patras, Greece, where he is currently working toward the PhD degree. He is currently a research assistant with the VLSI Design Laboratory, ECE Department, University of Patras. His current research interests include VLSI architectures for signal processing and computer arithmetic.

VASSILIS PALIOURAS (Member, IEEE) is currently a Full Professor with Electrical and Computer Engineering Department, University of Patras, Patras, Greece. He is an advisor to three PhD students, and has supervised four PhD, 36 master's, and 43 diploma theses. He has authored or coauthored more than 150 research articles in international journals, conferences, and book chapters, and has edited three books. His research interests include VLSI architectures for signal processing and communications, low-power digital design, and computer arithmetic. Prof. Paliouras was the recipient of the IEEE CASS Guillemin—Cauer Best-Paper Award for the year 2000. He was the general co-chair for International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS 2004). He was also the technical program chair of PATMOS 2005, the IEEE Workshop on Signal Processing Systems Implementation (SiPS) 2005, and the Technical Program Co-Chair of the IEEE International Conference on Electronics Circuits and Systems (ICECS) 2010 and a European liaison for the IEEE ISCAS 2012, South Korea. He was on the editorial boards of journals and technical program committees of numerous conferences in the areas of signal processing, circuits, systems, and communications.