On Modulo $2^n + 1$ Adder Design

Haridimos T. Vergos, *Member*, *IEEE*, and Giorgos Dimitrakopoulos, *Member*, *IEEE*

Abstract—Two architectures for modulo $2^n + 1$ adders are introduced in this paper. The first one is built around a sparse carry computation unit that computes only some of the carries of the modulo $2^n + 1$ addition. This sparse approach is enabled by the introduction of the inverted circular idempotency property of the parallel-prefix carry operator and its regularity and area efficiency are further enhanced by the introduction of a new prefix operator. The resulting diminished-1 adders can be implemented in smaller area and consume less power compared to all earlier proposals, while maintaining a high operation speed. The second architecture unifies the design of modulo $2^n \pm 1$ adders. It is shown that modulo $2^n + 1$ adders can be easily derived by straightforward modifications of modulo $2^n - 1$ adders with minor hardware overhead.

Index Terms—Modulo arithmetic, residue number system (RNS), parallel-prefix carry computation, computer arithmetic, VLSI.

1 Introduction

Arithmetic modulo $2^n + 1$ has found applicability in a variety of fields ranging from pseudorandom number generation and cryptography [1], [2], [3], up to convolution computations without round-off errors [4], [5], [6]. Also, modulo $2^n + 1$ operators are commonly included in residue number system (RNS) applications [7], [8], [9]. The RNS is an arithmetic system which decomposes a number into parts (residues) and performs arithmetic operations in parallel for each residue without the need of carry propagation among them, leading to significant speedup over the corresponding binary operations. RNS is well-suited to applications that are rich of addition/subtraction and multiplication operations and has been adopted in the design of digital signal processors [7], [10], [11], FIR filters [12], [13], [14] and communication components [15], [16], [17], offering in several cases apart from enhanced operation speed, low-power characteristics [18].

The complexity of a modulo $2^n + 1$ arithmetic unit is determined by the representation chosen for the input operands. Three representations have been considered; namely, the normal weighted one, the diminished-1 [19] and the signed-LSB representations [20]. We only consider the first two representations in the following, since the adoption of the signed-LSB representation does not lead to more efficient circuits in delay or area terms. In every case, when performing arithmetic operations modulo $2^n + 1$ the input operands and the results are limited between 0 and 2^n.

In the normal weighted representation, each operand requires $n + 1$ bits for its representation but only utilizes $2^n + 1$ representations out of the 2^{n+1} that these can provide. A more dense encoding of the input operands and simplified arithmetic operations modulo $2^n + 1$ are offered by the diminished-1 representation. In the diminished-1 representation, A is represented as $a_z A^\star$, where a_z is a single bit, often called the zero indication bit, and $A^\star$ is an n-bit vector, often called the number part. If $A > 0$, then $a_z = 0$ and $A^\star = A - 1$, whereas for $A = 0, a_z = 1$, and $A^\star = 0$. For example, the diminished-1 representation of $A = 5$ modulo 17 is 00100_2.

Considering that the most common operations required in modulo $2^n + 1$ arithmetic are negation, multiplication by a power of two and addition [4], the adoption of the diminished-1 representation, allows to limit these operations to n bits. Specifically, negation is performed by complementing every bit of $A^\star$, if $a_z = 0$ and inhibiting any change when $a_z = 1$. Multiplication by 2^i is performed by an i-bit left rotation of the bits of $A^\star$, in which the re-entering bits are complemented, if $a_z = 0$ and inhibiting any change when $a_z = 1$. Finally, the addition of $a_z A^\star$ with $b_z B^\star$, boils down to an n-bit modular addition of $A^\star$ with $B^\star$ with some minor modifications. All operations involved in modulo $2^n + 1$ addition for diminished-1 operands are discussed in detail in Section 3.

1.1 Related Work

Several papers have attacked the problem of designing efficient diminished adders. The majority of them rely on the use of an inverted end around carry (IEAC) n-bit adder, which is an adder that accepts two n-bit operands and provides a sum increased by one compared to their integer sum if their integer addition does not result in a carry output. Although an IEAC adder can be implemented by using an integer adder in which its carry output is connected back to its carry input via an inverter, such a direct feedback is not a good solution. Since the carry output depends on the carry input, a direct connection between them forms a combinational loop that may lead to an unwanted race condition [21]. To this end, a number of custom solutions have been proposed for the design of efficient IEAC adders.

Considering the diminished-1 representation for modulo $2^n + 1$ addition, [4], [5] used an IEAC adder which is based on an integer adder along with an extra carry lookahead (CLA) unit. The CLA unit computes the carry output which is then inverted used as the carry input of the integer adder. Solutions that rely on a single carry computation unit have also been proposed. Zimmermann [22], [23] proposed IEAC adders that make use of a parallel-prefix carry computation

• *H.T. Vergos is with the Department of Computer Engineering and Informatics, University of Patras, GR 26500, Patras, Greece. E-mail: vergos@ceid.upatras.gr.*
• *G. Dimitrakopoulos is with the Department of Informatics and Communications Engineering, University of Western Macedonia, Karamanli and Lygeris, Kozani, GR 50100, Greece. E-mail: gdimitrak@uowm.gr.*

Manuscript received 27 Jan. 2010; revised 28 June 2010; accepted 14 Sept. 2010; published online 7 Dec. 2010.
Recommended for acceptance by P. Montuschi.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2010-01-0067.
Digital Object Identifier no. 10.1109/TC.2010.261.

unit along with an extra prefix level that handles the inverted end-around carry.

Although these architectures are faster than the carry-lookahead ones proposed in [24], for sufficiently wide operands, they are slower than the corresponding parallel-prefix integer adders because of the need for the extra prefix level. In [24], it has been shown that the recirculation of the inverted end around carry can be performed within the existing prefix levels, that is, in parallel with the carries' computation. In this way, the need of the extra prefix level is canceled and parallel-prefix IEAC adders are derived that can operate as fast as their integer counterparts, that is, they offer a logic depth of $\log_2 n$ prefix levels. Unfortunately, this level of performance requires significantly more area than the solutions of [22], [23], since a double parallel-prefix computation tree is required in several levels of the carry computation unit.

For reducing the area complexity of the parallel-prefix solutions, select-prefix [25] and circular carry select [26] IEAC adders have been proposed. Unfortunately, both these proposals achieve a smaller operating speed than the parallel-prefix ones of [24]. Recently, very fast IEAC adders that use the Ling carry formulation of parallel-prefix addition [27] have appeared in [28], that also suffer from the requirement of a double parallel-prefix computation tree.

Although a modulo 2^n+1 adder that follows the $(n+1)$-bit weighted representation can be designed following the principles of generic modulo adder design [29], specialized architectures for it have appeared in [30], [31]. However, it has been recently shown [32] that a weighted adder can be designed efficiently by using an IEAC one and a carry save adder (CSA) stage. As a result, improving the design for an IEAC adder would improve the weighted adder design as well.

1.2 Main Contribution

In this paper, two novel architectures for the design of modulo 2^n+1 adders are introduced. Both architectures target the IEAC part of modulo 2^n+1 addition. The remaining cases of diminished-1 modulo 2^n+1 addition that stem from the need to handle zero operands, are covered by a newly proposed method that imposes only simple modifications to the core IEAC as discussed in Section 3.

The first design is based on a new parallel prefix carry computation unit that is customized for the case of modulo 2^n+1 addition and eliminates the double parallel-prefix computation tree problem of previously fast designs [24], [28]. The proposed architecture is utilized for the design of sparse modulo 2^n+1 adders that offer significantly reduced cell area, wiring complexity and power consumption along with a high operation speed. Its introduction is based on the extension of the well-known idempotency property of the prefix operator, by proving its inverted circular nature in the case of modulo 2^n+1 addition.

The second alternative is a versatile architecture that unifies the design of modulo 2^n-1 and 2^n+1 adders. We prove that modulo 2^n+1 addition can be treated as a subcase of the corresponding modulo 2^n-1 addition. New relations are introduced that associate the sum bits of the adders of these two moduli. Therefore, modulo 2^n-1 parallel-prefix adders [33], [34] can be also used for the design of modulo 2^n+1 adders using a small amount of extra logic.

The rest of the paper is organized as follows: Section 2 discusses shortly the design of parallel-prefix adders. The basics of modulo $2^n \pm 1$ addition are presented in Section 3. We also discuss how a diminished-1 modulo 2^n+1 adder can be derived by minor modifications of an IEAC adder. Section 4 introduces the new sparse parallel prefix carry computation units customized for the case of IEAC adders, while their performance benefits in terms of delay, area, and power consumption are quantified in Section 5 for the design of both diminished-1 and weighted modulo 2^n+1 adders. Then, in Section 6, the new theory that unifies modulo 2^n-1 and IEAC (and consequently modulo 2^n+1) addition is developed. Conclusions are drawn in the last section.

2 Parallel-Prefix Addition Basics

Suppose that $A = A_{n-1}A_{n-2}\dots A_0$ and $B = B_{n-1}B_{n-2}\dots B_0$ represent the two numbers to be added and $S = S_{n-1}S_{n-2}\dots S_0$ denotes their sum. An adder can be considered as a three-stage circuit. The preprocessing stage computes the carry-generate bits G_i, the carry-propagate bits P_i, and the half-sum bits H_i, for every $i, 0 \le i \le n-1$, according to

$$G_i = A_i \cdot B_i \qquad P_i = A_i + B_i \qquad H_i = A_i \oplus B_i,$$

where $\cdot, +$, and $\oplus$ denote logical AND, OR, and exclusive-OR, respectively. The second stage of the adder, hereafter called the carry computation unit, computes the carry signals C_i, for $0 \le i \le n-1$ using the carry generate and carry propagate bits G_i and P_i. The third stage computes the sum bits according to

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

Carry computation is transformed into a parallel prefix problem using the $\circ$ operator, which associates pairs of generate and propagate signals and was defined in [35] as

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

In a series of associations of consecutive generate/propagate pairs (G, P), the notation $(G_{k:j}, P_{k:j})$, with $k > j$, is used to denote the group generate/propagate term produced out of bits $k, k-1, \dots, j$, that is,

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

Since every carry $C_i = G_{i:0}$, a number of algorithms have been introduced for computing all the carries using only $\circ$ operators. Fig. 1 presents the most well-known approaches for the design of an 8-bit adder, while Fig. 2 depicts the logic-level implementation of the basic cells used throughout the paper.

For large wordlengths, the design of sparse parallel prefix adders is preferred, since the wiring and area of the design are significantly reduced without sacrificing delay. The design of sparse adders relies on the use of a sparse parallel-prefix carry computation unit and carry-select (CS) blocks. Only the carries at the boundaries of the carry-select blocks are computed, saving considerable amount of area in the carry-computation unit [39]. A 32-bit adder with 4-bit sparseness is shown in Fig. 3a. The carry select block computes two sets of sum bits corresponding to the two possible values of the incoming carry. When the actual carry is computed, it selects the correct sum without any delay overhead. A possible logic-level implementation of a 4-bit carry-select block is shown in Fig. 3b.

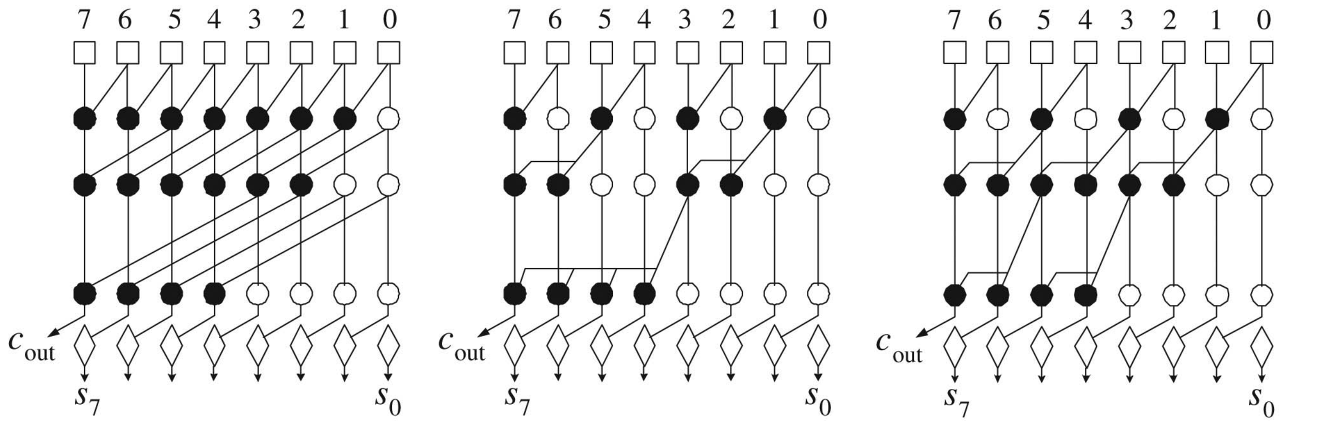


Fig. 1. Examples of 8-bit parallel-prefix structures for integer adders. (a) Kogge-Stone [36], (b) Ladner-Fischer [37], and (c) one representative of the Knowles [38] family of adders.

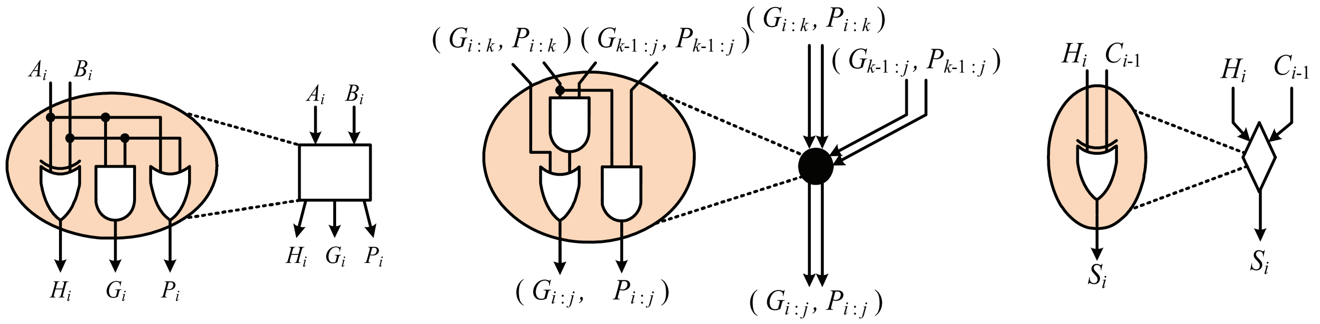


Fig. 2. The logic-level implementation of the basic cells used in parallel-prefix adders.

3 Modulo $2^n \pm 1$ Addition Basics

3.1 Modulo $2^n - 1$ Adders

The computation of modulo 2^n-1 addition is, in fact, a conditional operation defined as

$$(A+B)\text{mod}(2^n-1) = \begin{cases} (A+B), & A+B<2^n, \\ (A+B+1)\text{mod } 2^n, & A+B \geq 2^n. \end{cases}$$

A modulo 2^n-1 adder can be implemented using an integer adder that increments also its sum when the carry output is one, that is, when $A+B \geq 2^n$. This is also equivalent to feeding the carry-input of the adder with the carry-output of the first addition. The conditional increment can be implemented by an additional carry increment stage as shown in Fig. 4a. In this case, one extra level of $\bullet$ cells driven by the carry output of the adder, is required.

Depending on the implementation of the modulo 2^n-1 adder, for bitwise-complementary inputs, i.e., when $A+B=2^n-1$, the adder may produce an all 1s output vector, in place of the expected result which is equal to zero. In most applications, this is acceptable as a second representation for zero.

The implementation of a modulo 2^n-1 adder requires the connection of the carry output $C_{n-1}=G_{n-1:0}$ of an integer adder to its carry-input port. The carries of the modulo 2^n-1 adder C_i^- that takes also a carry-input port are equal to $C_i^- = G_{i:0} + P_{i:0} \cdot C_{in}$. Therefore, connecting the carry output to the carry input leads to $C_i^- = G_{i:0} + P_{i:0} \cdot G_{n-1:0}$. This relation contains many redundant terms and according to [40] can be simplified to

$$C_i^- = G_{i:0} + P_{i:0} \cdot G_{n-1:i+1}. \tag{1}$$

This simpler equation can be equivalently expressed using the $\circ$ operator as follows:

$$C_i^- \leftrightarrow (G_i, P_i) \circ \cdots (G_0, P_0) \circ (G_{n-1}, P_{n-1}) \circ \cdots \circ (G_{i+1}, P_{i+1}). \tag{2}$$

Relation (2) that computes the modulo 2^n-1 carries has a cyclic form and, in contrast to integer addition, the number of generate and propagate pairs (G_i, P_i) that need to be associated for each carry is equal to n. This means that the parallel-prefix carry computation unit of a modulo 2^n-1 adder has significantly increased area complexity than that of a corresponding integer adder. In terms of delay, the carries C^- can be computed in $\log_2 n$ levels using regular parallel prefix structures, as the one shown in Fig. 4b. At each level of the parallel prefix structure, larger groups of (G_i, P_i) are progressively associated and the carries C^- are computed at the last level. The final sum bits S_i^- are equal to $H_i \oplus C_{i-1}^-$.

The above form of modulo 2^n-1 adder suffers from the double representation of zero. Few solutions have been reported on the design of a modulo 2^n-1 adder with a single zero representation. Those proposed by [40] have an increased delay compared to those with a double zero representation since they rely on using H_i instead of P_i as

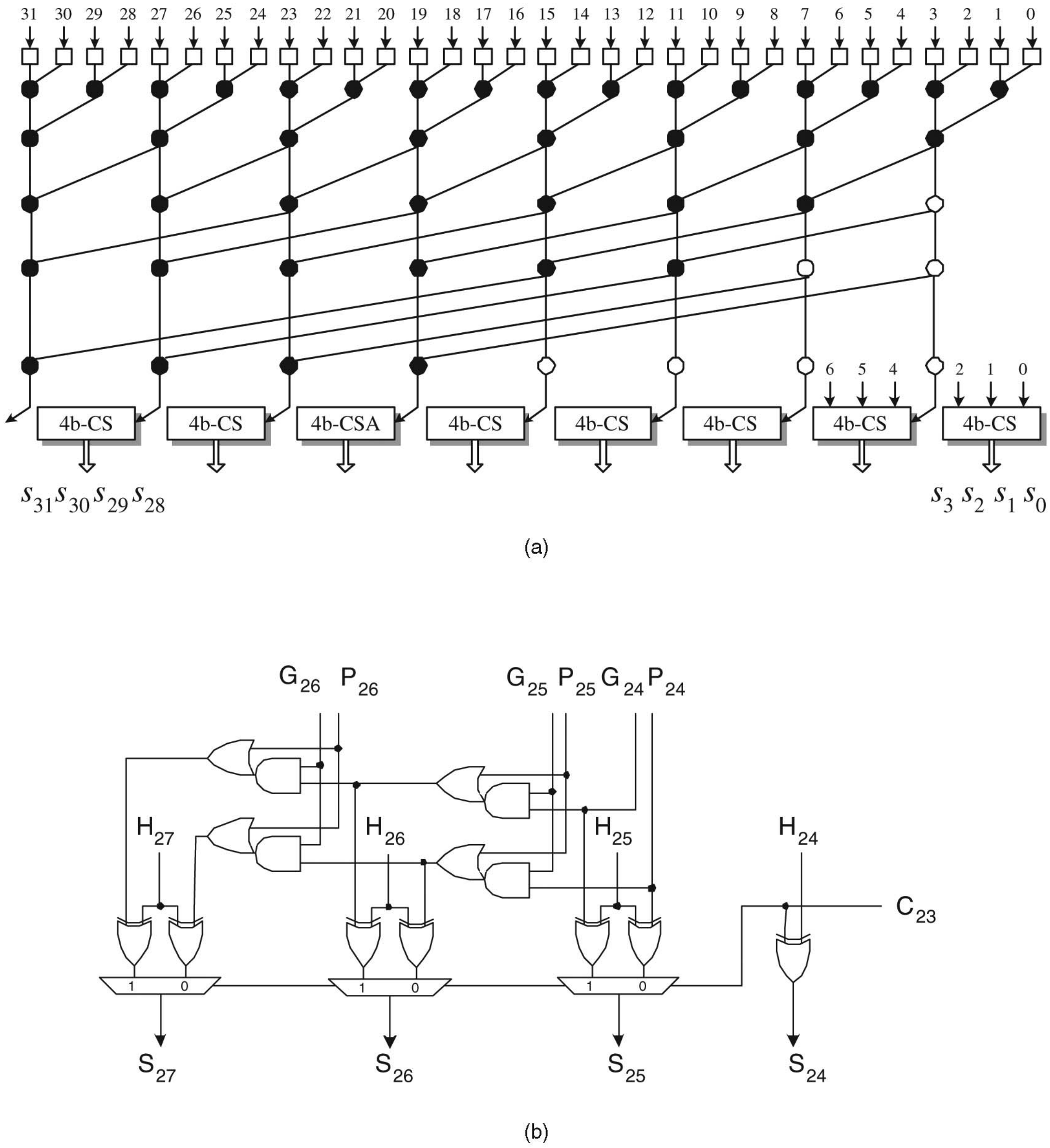


Fig. 3. (a) Sparse-4 parallel-prefix structure for a 32-bit integer adder and (b) the logic level implementation of the CS block.

the carry propagate signal, while those proposed in [41] compute the modulo carries C^- as

$$C_i^- \leftrightarrow (G_i, P_i) \circ \cdots (G_0, P_0) \circ (G_{n-1}, P_{n-1}) \circ \cdots \circ (P_{i+1}, P_{i+1})$$

that is, by using P_{i+1} instead of G_{i+1}. Although this change seems minor, it ruins the regularity of the adders, and the resulting implementation suffers from increased cell and interconnect area. In the rest of this paper, we consider modulo $2^n - 1$ adders with a double representation for zero.

3.2 Modulo $2^n + 1$ Adders

Diminished-1 modulo $2^n + 1$ addition is more complex since special care is required when at least one of the input operands is zero $(1\,00\ldots0)$. The sum of a diminished-1 modulo adder is derived according to the following cases:

1. When none of the input operands is zero ($a_z, b_z \neq 0$) their number parts $A^\star$ and $B^\star$ are added modulo $2^n + 1$. This operation as discussed in the following, can be handled by an IEAC adder.
2. When one of the two inputs is zero the result is equal to the nonzero operand.
3. When both operands are zero, the result is zero.

In any case that the result is equal to zero (cases 1 or 3), the zero-indication bit of the sum needs to be set and the number part of the sum should be equal to the all-zero vector. According to the above, a true modulo addition in a diminished-1 adder is needed only in case 1, while in the other cases the sum is known in advance.

When none of the input operands is zero, $a_z, b_z \neq 1$, the number part of the diminished-1 sum is derived by the number parts $A^\star$ and $B^\star$ of the input operands as follows:

$$\begin{aligned} S^+ &= (A^\star + B^\star) \bmod (2^n + 1) \\ &= \begin{cases} (A^\star + B^\star + 1) \bmod 2^n, & A^\star + B^\star < 2^n, \\ (A^\star + B^\star) \bmod 2^n, & A^\star + B^\star \geq 2^n. \end{cases} \end{aligned} \quad (3)$$

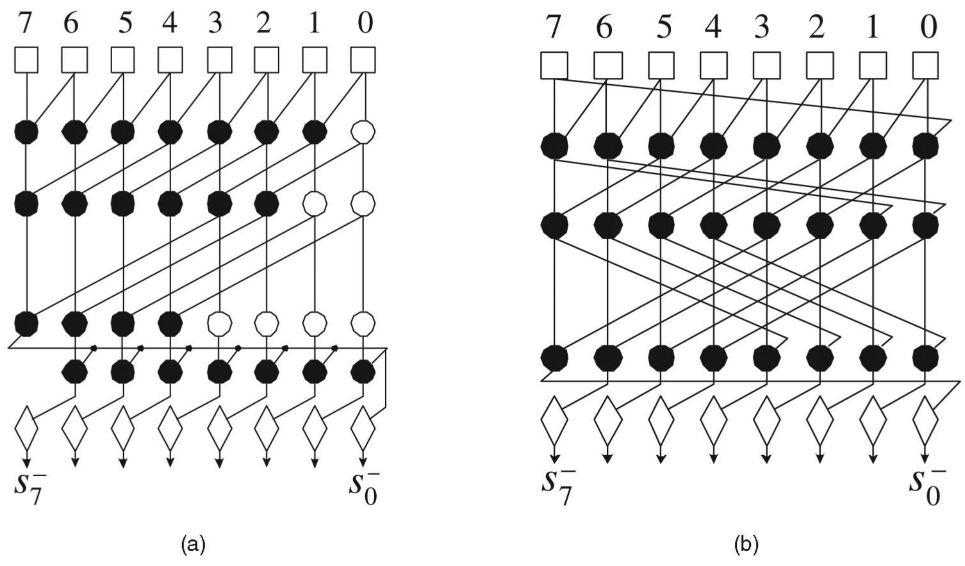


Fig. 4. Parallel prefix modulo $2^8 - 1$ adders: (a) using an additional carry-increment stage and (b) recirculating the end around carry within the existing $\log_2 n$ prefix levels.

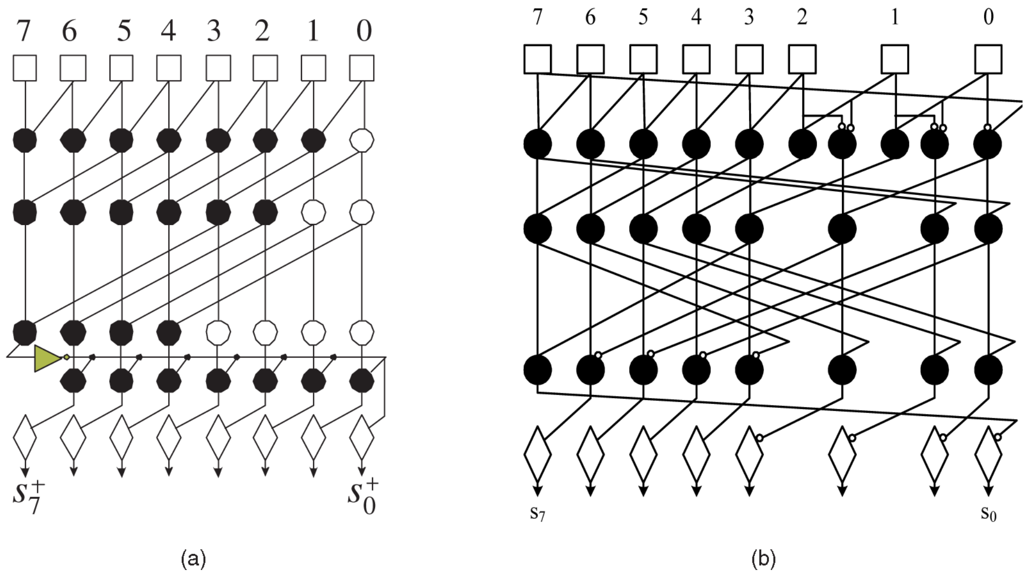


Fig. 5. Parallel prefix modulo $2^8 + 1$ adders: (a) using an additional carry-increment stage and (b) recirculating the inverted end around carry within the existing $\log_2 n$ prefix levels.

Equation (3) reveals that an IEAC adder can be used for providing the number part in this case. Fig. 5a [22], [23] presents the implementation of an IEAC adder by the addition of a carry increment stage to an integer parallel prefix adder.

In an analogous way to that of the modulo $2^n - 1$ case, [24] has shown that the carry C_i^+ at the ith bit position of an IEAC adder, when feeding the carry input $C_{in} = C_{-1}^+$ with the inverted carry out $\overline{C_{n-1}} = \overline{G_{n-1:0}}$, can be computed more simply by

$$C_i^+ = G_{i:0} + P_{i:0} \cdot \overline{G_{n-1:i+1}}. \tag{4}$$

Equivalently, using the $\circ$ operator the IEAC addition carries can be expressed as

$$C_i^+ \leftrightarrow (G_i, P_i) \circ \cdots (G_0, P_0) \circ \overline{(G_{n-1}, P_{n-1}) \circ \cdots \circ (G_{i+1}, P_{i+1})},$$

where by definition, $\overline{(g, p)}$ is equal to $(\overline{g}, p)$, and the final sum bits S_i^+ are equal to $H_i \oplus C_{i-1}^+$. Using the above simplified carry equations, the parallel prefix graph shown in Fig. 5b can be derived for case 1 of the diminished-1 addition.

When comparing Figs. 4b and 5b, it is evident that developing a regular parallel prefix structure for the computation of C_i^+ is extremely complicated due to the inversion of the term $G_{n-1:i+1}$ that appears in (4). The solution provided in [24] doubles up several operators of the computation unit in order to compute each C_i^+ in at most $\log_2 n$ prefix levels, while its regularity vanishes for large values of n and when n is not a power of two. These drawbacks do not appear in the case of modulo $2^n - 1$ addition, where regular and efficient parallel-prefix units can be designed for every n, using in many cases less than $n \log_2 n$ prefix operators when n is not a power of two [33], [42].

In the following, we discuss how an IEAC adder, which is used for the addition of the number parts of case 1, can be modified for handling the other two cases of the input operands (cases 2 and 3) in a unified way. Apart from small, the modifications should not hurt the parallel prefix structure of the adder. These modifications are required since the IEAC adder has been built to work correctly only when $a_z, b_z \neq 0$. In this case, all zeros vector in the number parts A^*, B^*, represents the integer 1 and not a zero operand. Therefore, when one or both the input operands is zero ($a_z = 1$ or $b_z = 1$) and the IEAC has not been properly modified, it will return a number part increased by one compared to the expected.

In order to avoid having a wrongly incremented output, when the IEAC is used for true zero operands (cases 2 and 3), we propose to set all the carries C_i^+ equal to zero, when at least one of the input operands is equal to zero ($a_z = 1$ or $b_z = 1$). The carries C_i^+, with $i > 0$ follow the form $G_{i:0} + P_{i:0} \cdot \overline{G_{n-1:i+1}}$, while C_{-1}^+ used for the derivation of S_0^+ is equal to $\overline{G_{n-1:0}}$. In cases 2 and 3, it can be easily verified that all carry generate bits G_i are equal to zero since at least one of the number parts of the input operands is equal to the all zeros vector. Therefore, forcing all carry propagate bits P_i to zero, when $a_z = 1$ or $b_z = 1$, suffices to set $C_i^+ = 0$, with $i > 0$. However, this approach does not alter C_{-1}^+ which remains equal to one. Therefore, we choose just to invert S_0^+ when at least one of the input operands is zero, thus canceling in a simple manner the fact that C_{-1}^+ remains equal to one. This can be easily performed by computing S_0^+ as $S_0^+ = [H_0 \oplus (a_z + b_z)] \oplus C_{-1}^+$.

The above modifications do not alter the parallel-prefix structure of the IEAC shown in Fig. 5b and their delay overhead is very small. More specifically, the only delay overhead is caused by the transformation of the 2-input OR gates of the preprocessing stage into OR-AND compound gates. The modification at S_0^+ does not add to the critical path of the IEAC.

The adders of [4], [5], [23] could have also used this approach. However, since they rely on an additional carry-increment stage as shown in Fig. 5a it is simpler to use as a re-entering carry the $C_{-1}^+ = \overline{a_z + b_z + C_{n-1}}$ that also covers efficiently cases 2 and 3 of diminished-1 modulo addition.

Finally, the last point that remains to be discussed is the generation of the zero-indication flag of the sum s_z when the result is equal to zero. This condition arises in the following cases:

- When both input operands are zero $a_z = b_z = 1$ and $A^* = B^* = 0$.
- When none of the input operands are zero and A^*, B^* are bitwise complementary that is $A^* + B^* = 2^n - 1$.

Both these conditions can be encoded in the following Boolean relation:

$$s_z = a_z \cdot b_z + \overline{a_z + b_z} \cdot \overline{G_{n-1:0}} \cdot P_{n-1:0},$$

where $\overline{G_{n-1:0}} \cdot P_{n-1:0}$ detects that the input operands are bitwise complementary [43].

In the rest of the paper, we focus only on the parallel-prefix structures that implement an IEAC adder similar to the circuit shown in Fig. 5b, which according to the above discussion is the core of a modulo $2^n + 1$ adder. The small extra hardware required to cover the additional cases arising from true zero input operands and the computation of the s_z flag in a diminished-1 adder is assumed to coexist with the reported circuits and explicitly considered only when necessary.

4 New Sparse Modulo $2^n + 1$ Adders

In this section, we focus on the design of diminished modulo adders with a sparse parallel-prefix carry computation stage that can use the same carry-select blocks as the sparse integer adders.

4.1 Partially Regular Sparse Parallel-Prefix Adders

We will first show that the carries of the diminished-1 modulo $2^n + 1$ addition are associated in the very same way as the carries of the integer addition. To this end, the *inverted circular idempotency* property is introduced by the following Theorem:

Theorem 1.

$$\begin{aligned}(G_{i:0}, P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1}) \circ (G_{i:0}, P_{i:0})} \\ = (G_{i:0}, P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1})}\end{aligned}$$

with $\overline{(G, P)} = (\overline{G}, P)$, according to the definition given in [24].

Proof. At first, we ungroup the $\circ$ operators with their equivalent Boolean relations as

$$\begin{aligned}&(G_{i:0}, P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1}) \circ (G_{i:0}, P_{i:0})} \\ &\quad = (G_{i:0}, P_{i:0}) \circ (\overline{G_{n-1:i+1} + P_{n-1:i+1} \cdot G_{i:0}}, P_{n-1:i+1} \cdot P_{i:0}) \\ &\quad = (G_{i:0} + P_{i:0} \cdot (\overline{G_{n-1:i+1} + P_{n-1:i+1} \cdot G_{i:0}}), \\ &\qquad P_{i:0} \cdot P_{n-1:i+1} \cdot P_{i:0}).\end{aligned}$$

In the following, in the generate part of the prefix relation we expand the inversion operation, while in the propagate part we simplify the double appearance of the term $P_{i:0}$ as

$$\begin{aligned}&(G_{i:0}, P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1}) \circ (G_{i:0}, P_{i:0})} \\ &\quad = (G_{i:0} + P_{i:0} \cdot (\overline{G_{n-1:i+1}} \cdot (\overline{P_{n-1:i+1}} + \overline{G_{i:0}})), \\ &\qquad P_{i:0} \cdot P_{n-1:i+1}) \\ &\quad = (G_{i:0} + (P_{i:0} \cdot \overline{G_{n-1:i+1}} \cdot \overline{P_{n-1:i+1}}) \\ &\qquad + (P_{i:0} \cdot \overline{G_{n-1:i+1}} \cdot \overline{G_{i:0}}),\ P_{i:0} \cdot P_{n-1:i+1}).\end{aligned}$$

The terms $G_{i:0} + P_{i:0} \cdot \overline{G_{n-1:i+1}} \cdot \overline{G_{i:0}}$ are reduced to $G_{i:0} + P_{i:0} \cdot \overline{G_{n-1:i+1}}$ which simplifies the initial relation as follows:

$$\begin{aligned}&(G_{i:0}, P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1}) \circ (G_{i:0}, P_{i:0})} \\ &\quad = (G_{i:0} + (P_{i:0} \cdot \overline{G_{n-1:i+1}} \cdot \overline{P_{n-1:i+1}}) + (P_{i:0} \cdot \overline{G_{n-1:i+1}}), \\ &\qquad P_{i:0} \cdot P_{n-1:i+1}).\end{aligned}$$

Observing the last two terms of the generate part, we see that the product $P_{i:0} \cdot \overline{G_{n-1:i+1}}$ appears redundantly twice. Therefore, our relation can be simplified as follows:

$$\begin{aligned}(G_{i:0}, P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1}) \circ (G_{i:0}, P_{i:0})} \\ = (G_{i:0} + P_{i:0} \cdot \overline{G_{n-1:i+1}}, P_{i:0} \cdot P_{n-1:i+1}).\end{aligned}$$

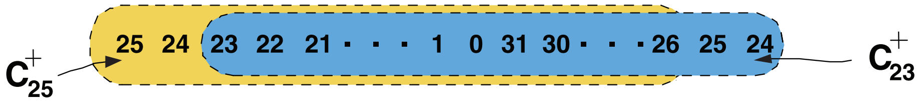


Fig. 6. The circular overlap of the modulo carries C^+_{25} and C^+_{23} in case of modulo $2^{32}+1$ addition.

If we reuse the $\circ$ operator, the equation is rewritten in the following way:

$$\begin{aligned}&(G_{i:0},P_{i:0}) \circ \overline{(G_{n-1:i+1},P_{n-1:i+1}) \circ (G_{i:0},P_{i:0})}\\ &= (G_{i:0},P_{i:0}) \circ (\overline{G_{n-1:i+1}},P_{n-1:i+1})\\ &= (G_{i:0},P_{i:0}) \circ \overline{(G_{n-1:i+1},P_{n-1:i+1})},\end{aligned}$$

which concludes the proof. □

The inverted circular idempotency property indicates that we can repeat (G_i, P_i) terms that appear at the front of a prefix relation of the form suggested by (4) *inverted* at its tail. For example, consider the prefix relation for C^+_1 in the case of a 2^4+1 diminished adder:

$$(G_1,P_1) \circ (G_0,P_0) \circ \overline{(G_3,G_3) \circ (G_2,P_2)}.$$

According to the inverted circular idempotency property, this relation is also equivalent to the following ones:

$$\begin{aligned}&(G_1,P_1) \circ (G_0,P_0) \circ \overline{(G_3,P_3) \circ (G_2,P_2) \circ (G_1,P_1)}\\ &(G_1,P_1) \circ (G_0,P_0) \circ \overline{(G_3,P_3) \circ (G_2,P_2) \circ (G_1,P_1) \circ (G_0,P_0)}.\end{aligned}$$

Armed with the inverted circular idempotency, we will present the proposed methodology by using as an example the design of a sparse-4 parallel-prefix modulo $2^{32}+1$ adder. Since we assume a sparsity of 4, only one every four carries is generated at positions 3, 7, 11, 15, 19, 23, 27, and 31 or equivalently -1. In order to use the carry-select block of Fig. 3b, we need to show that this internally derives the required modulo addition carries based on the available ones.

For example, C^+_{25} should be derived by the available by the sparse parallel-prefix unit C^+_{23} and the carry-select block logic. According to (4) we know that

$$\begin{aligned}C^+_{25} &= G_{25:0} + P_{25:0} \cdot \overline{G_{31:26}},\\ C^+_{23} &= G_{23:0} + P_{23:0} \cdot \overline{G_{31:24}}.\end{aligned}$$

We will show that although C^+_{23} contains the group generate term $G_{25:24}$ which partially overlaps with the group generate term $G_{25:0}$ of C^+_{25}, in a circular manner, as shown in Fig. 6, it is still possible to derive C^+_{25} from C^+_{23}. Utilizing the $\circ$ operator C^+_{25} can be equivalently expressed as

$$C^+_{25} \leftrightarrow (G_{25:0},P_{25:0}) \circ \overline{(G_{31:26},P_{31:26})}.$$

Expanding $(G_{25:0},P_{25:0})$ in two smaller carry generate/propagate groups we have that

$$C^+_{25} \leftrightarrow (G_{25:24},P_{25:24}) \circ (G_{23:0},P_{23:0}) \circ \overline{(G_{31:26},P_{31:26})}.$$

Based on Theorem 1 the term

$$(G_{25:24},P_{25:24}) \circ (G_{23:0},P_{23:0}) \circ \overline{(G_{31:26},P_{31:26})}$$

that appears in the relation of C^+_{25} can be equivalently expanded to

$$(G_{25:24},P_{25:24}) \circ (G_{23:0},P_{23:0}) \circ \overline{(G_{31:26},P_{31:26}) \circ (G_{25:24},P_{25:24})}.$$

Regrouping together the contiguous carry generate and propagate terms $(G_{31:26},P_{31:26})$ and $(G_{25:24},P_{25:24})$ we get that

$$C^+_{25} \leftrightarrow (G_{25:24},P_{25:24}) \circ (G_{23:0},P_{23:0}) \circ \overline{(G_{31:24},P_{31:24})}.$$

It can be easily identified that $(G_{23:0},P_{23:0}) \circ \overline{(G_{31:24},P_{31:24})}$ corresponds to C^+_{23} and thus

$$C^+_{25} = G_{25:24} + P_{25:24} \cdot C^+_{23}. \quad (5)$$

From (5), we conclude that although carries C^+_{25} and C^+_{23} refer to diminished-1 modulo 2^n+1 addition, the relation that associates them is the same as in the case of integer addition. Therefore, the carry select block used for integer adders can be used without modifications for the design of sparse modulo 2^n+1 adders.

Fig. 7 presents the architecture proposed in [24] (a) and the architecture (b) that can be derived using the above developed theory for modulo $2^{16}+1$ diminished adders. The sparse architecture offers the same prefix levels as the architecture of [24], but the carry computation unit is far simpler, since it requires significantly less prefix operators and wiring.

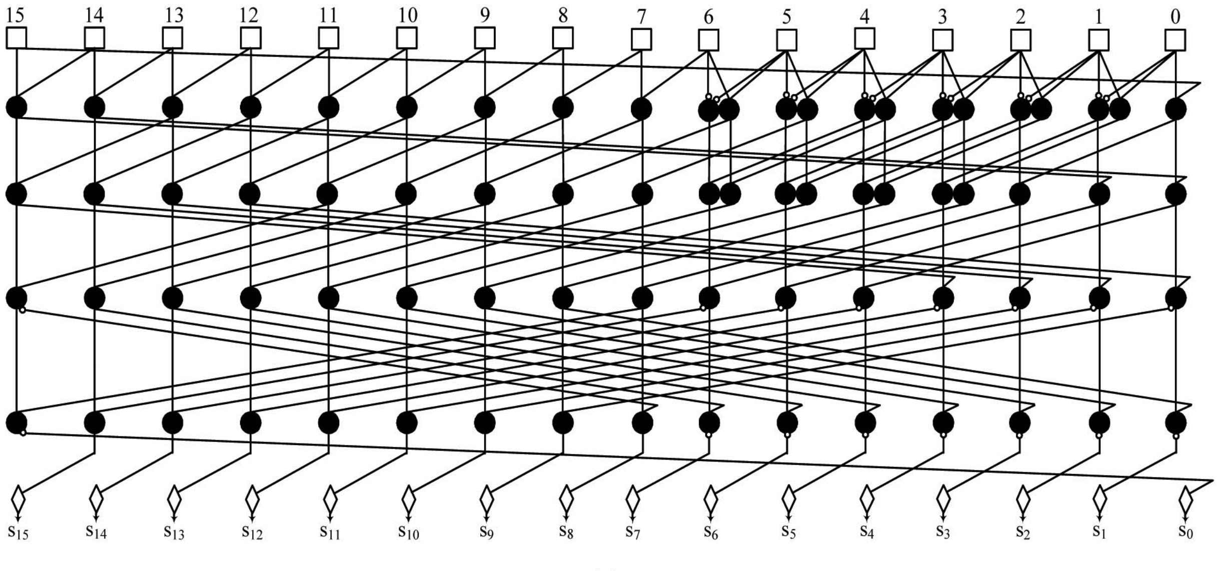
4.2 Totally Regular Parallel-Prefix Units

The methodology presented in [24] is the only approach known so far that can organize the computation of the carries C^+, in case of modulo 2^n+1 addition, in a parallel-prefix-like form with $\log_2 n$ prefix levels. As also shown in Fig. 7a some prefix operators are doubled up, since two carry computations need to be performed in parallel; one on normal propagate and generate signals, while the other on their complements. The problem gets worse when the input operands' width is not a power of two. Although, the sparse version of the parallel-prefix adders introduced in this paper alleviates a lot the regularity and the area-overhead problem, as it can be verified from Fig. 7b, there is still a lot of space for improvement.

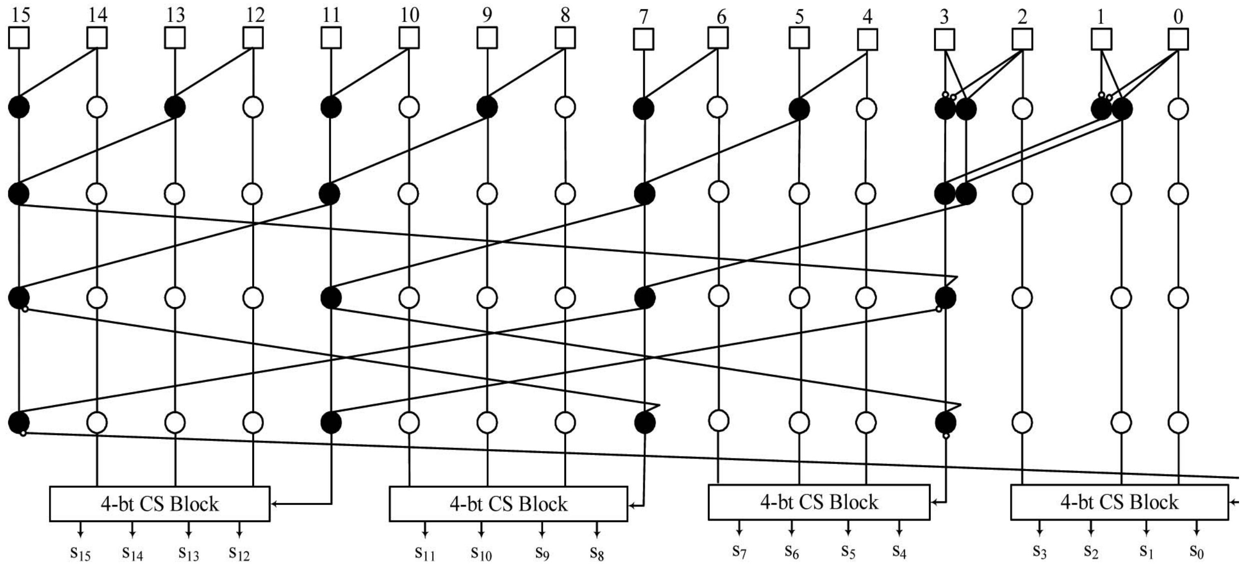
In the following, we attack this problem by introducing a new prefix operator and an even simpler parallel-prefix carry computation unit. The new technique will be presented via an example. Let us consider the design of a sparse-4 diminished-1 modulo $2^{16}+1$ adder. In this case, we need a carry computation unit that implements the following prefix equations:

$$\begin{aligned}C^+_{15} &\leftrightarrow \overline{(G_{15:0},P_{15:0})},\\ C^+_{11} &\leftrightarrow (G_{11:0},P_{11:0}) \circ \overline{(G_{15:12},P_{15:12})},\\ C^+_{7} &\leftrightarrow (G_{7:0},P_{7:0}) \circ \overline{(G_{15:8},P_{15:8})},\\ C^+_{3} &\leftrightarrow (G_{3:0},P_{3:0}) \circ \overline{(G_{15:4},P_{15:4})}.\end{aligned}$$

For computing all carries within $\log_2 n$ prefix levels in [24] it was shown that



(a)



(b)

Fig. 7. Modulo $2^{16}+1$ diminished adders. (a) Proposal of [24] and (b) using a sparse carry computation unit.

$$(g,p)\circ\overline{(G,P)} = \overline{(\overline{p},\overline{g})\circ(G,P)}$$

and therefore the parallel-prefix carry computation unit of Fig. 7a equivalently computes C_3^+ and C_{11}^+ by

$$C_3^+ \leftrightarrow \overline{((\overline{p_3},\overline{g_3})\circ(\overline{p_2},\overline{g_2})\circ(\overline{p_1},\overline{g_1})\circ(\overline{p_0},\overline{g_0}))\circ(G_{15:12},P_{15:12})}$$
$$\circ\,\overline{(G_{11:4},P_{11:4})},$$
$$C_{11}^+ \leftrightarrow (G_{11:4},P_{11:4})\circ\overline{((\overline{p_3},\overline{g_3})\circ(\overline{p_2},\overline{g_2})\circ(\overline{p_1},\overline{g_1})\circ(\overline{p_0},\overline{g_0}))}$$
$$\circ\,\overline{(G_{15:12},P_{15:12})}.$$

This approach unfortunately doubles up several operators, since $(g_1,p_1)\circ(g_0,p_0)$ and $(\overline{p_1},\overline{g_1})\circ(\overline{p_0},\overline{g_0})$ need to be computed in parallel and the same is true for $(g_3,p_3)\circ(g_2,p_2)$ with $(\overline{p_3},\overline{g_3})\circ(\overline{p_2},\overline{g_2})$ and $(G_{3:0},P_{3:0})$ with $(\overline{p_3},\overline{g_3})\circ(\overline{p_2},\overline{g_2})\circ(\overline{p_1},\overline{g_1})\circ(\overline{p_0},\overline{g_0})$. For larger adders, significantly more operators need to be doubled up, leading to increased area and wiring. To overcome this problem, we need a prefix operator that can associate $(G_{k:0},P_{k:0})\circ\overline{(G_{n-1:r},P_{n-1:r})}$ with $\overline{(G_{r-1:m},P_{r-1:m})}$ for computing $(G_{k:0},P_{k:0})\circ\overline{(G_{n-1:m},P_{n-1:m})}$. In our example case, this will permit us to compute C_3^+ by associating

$$(G_{3:0},P_{3:0})\circ\overline{(G_{15:12},P_{15:12})} \quad \text{with} \quad \overline{(G_{11:4},P_{11:4})}$$

in order to produce

$$C_3^+ \leftrightarrow (G_{3:0},P_{3:0})\circ\overline{(G_{15:12},P_{15:12})\circ(G_{11:4},P_{11:4})}.$$

To this end, we introduce a new operator, hereafter called *gray* operator. The implementation of a gray operator is given in Fig. 8. It accepts five inputs and produces four outputs. Three of the inputs of a gray operator residing at prefix level $j-1$, namely, G_V^{j-1}, P_V^{j-1} and T_V^{j-1} form the operator's vertical input bus, while the rest two G_L^{j-1} and P_L^{j-1} form

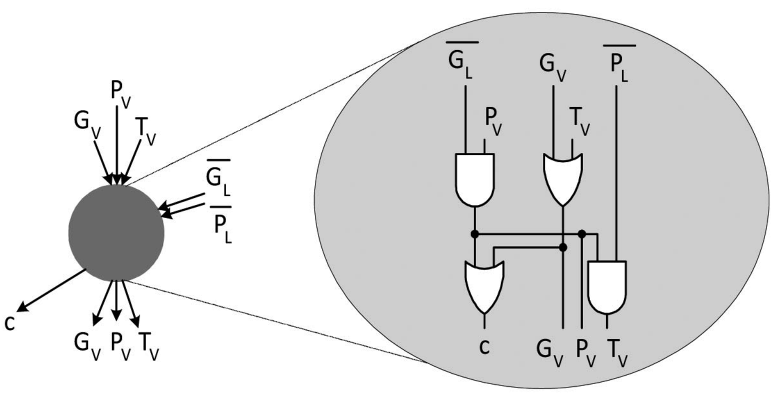


Fig. 8. Gray prefix operator; notation and implementation.

its lateral input bus. The lateral bus signals are driven inverted to the operator. The gray operator produces three signals for its vertical successor of prefix level j (G_V^j, P_V^j and T_V^j) and one (c^j) for its lateral successor. Note that compared to the $\circ$ prefix operator, the gray one requires one extra gate, but does not require extra logic levels. Considering a sparse-2^k parallel-prefix carry computation unit, gray operators will not be used in the first k prefix levels, since these need only compute the group generate and propagate terms out of 2^k adjacent bit positions.

The logic equations performed by a gray operator residing at prefix level $j-1$ are

$$\begin{aligned} G_V^j &= G_V^{j-1} + T_V^{j-1}, \\ P_V^j &= P_V^{j-1} \cdot \overline{G_L^{j-1}}, \\ T_V^j &= P_V^j \cdot \overline{P_L^{j-1}}, \\ c^j &= G_V^j + P_V^j. \end{aligned}$$

Consider now that we connect

$$\begin{aligned} &T_V^{j-1} \text{ to } 0, \\ &G_V^{j-1} \text{ and } P_V^{j-1} \text{ to } G_{k:0} \text{ and } P_{k:0}, \text{respectively, and} \\ &G_L^{j-1} \text{ and } P_L^{j-1} \text{ to } G_{n-1:r} \text{ and } P_{n-1:r}, \text{ respectively.} \end{aligned}$$

Then, the gray operator will provide $c^j = G_{k:0} + P_{k:0} \cdot \overline{G_{n-1:r}}$ at its lateral output. More importantly, it will also provide at its vertical outputs:

$$\begin{aligned} G_L^j &= G_{k:0}, \\ P_L^j &= P_{k:0} \cdot \overline{G_{n-1:r}}, \\ T_L^j &= P_{k:0} \cdot \overline{G_{n-1:r}} \cdot \overline{P_{n-1:r}}, \end{aligned}$$

an information which as we will show in the following suffices for the vertical successor to compute $(G_{k:0}, P_{k:0}) \circ \overline{(G_{n-1:m}, P_{n-1:m})}$ out of $(G_{k:0}, P_{k:0}) \circ \overline{(G_{n-1:r}, P_{n-1:r})}$ and $\overline{(G_{r-1:m}, P_{r-1:m})}$. Consider the vertical successor of the aforementioned gray operator which resides in prefix level j. By using a gray operator in its place in which we connect G_L^j and P_L^j to $G_{r-1:m}$ and $P_{r-1:m}$, respectively, and G_V^j, P_V^j and T_V^j to the vertical output of the gray operator of level $j-1$ mentioned above, that is,

$$\begin{aligned} &G_V^j \quad \text{to} \quad G_{k:0}, \\ &P_V^j \quad \text{to} \quad P_{k:0} \cdot \overline{G_{n-1:r}}, \text{ and} \\ &T_V^j \quad \text{to} \quad P_{k:0} \cdot \overline{G_{n-1:r}} \cdot \overline{P_{n-1:r}}. \end{aligned}$$

The lateral output of the operator will be equal to

$$\begin{aligned} c^{j+1} &= G_V^j + P_V^j = G_{k:0} + P_{k:0} \cdot \overline{G_{n-1:r}} \cdot \overline{P_{n-1:r}} + P_{k:0} \\ &\quad \cdot \overline{G_{n-1:r}} \cdot \overline{G_{r-1:m}}, \\ &= G_{k:0} + P_{k:0} \cdot (\overline{G_{n-1:r}} \cdot (\overline{P_{n-1:r}} + \overline{G_{r-1:m}})), \\ &= G_{k:0} + P_{k:0} \cdot (\overline{G_{n-1:r}} \cdot (\overline{P_{n-1:r} \cdot G_{r-1:m}})), \\ &= G_{k:0} + P_{k:0} \cdot (\overline{G_{n-1:r} + P_{n-1:r} \cdot G_{r-1:m}}), \\ &= G_{k:0} + P_{k:0} \cdot \overline{G_{n-1:m}}, \end{aligned}$$

and will provide at its vertical outputs:

$$\begin{aligned} G_V^{j+1} &= G_{k:0}, \\ P_V^{j+1} &= P_{k:0} \cdot \overline{G_{n-1:m}}, \\ T_V^{j+1} &= P_{k:0} \cdot \overline{G_{n-1:m}} \cdot \overline{P_{n-1:m}}. \end{aligned}$$

Applying the same procedure recursively, the lateral output of the last vertical successor of a gray operator will be equal to

$$G_{k:0} + P_{k:0} \cdot \overline{G_{n-1:k+1}},$$

that is, equal to C_k^+.

From the above analysis, we conclude that starting from a sparse architecture with doubled up operators, it suffices to

1. remove the doubled up operators that associate inverted signals,
2. replace the top operator of every column excluding the leftmost that accepts a feedback signal with a gray one, with its T_V input tied to zero, and
3. replace every vertical successor of a gray operator introduced by the previous step with a gray one,

to attain a diminished-1 modulo 2^n+1 adder, with an even simpler totally regular sparse parallel-prefix carry computation unit. Fig. 9 presents the resulting architecture for a diminished-1 modulo $2^{16}+1$ adder, in which two gray operators are used. The top one which resides at prefix level 3, accepts a feedback signal and therefore has its T_V^3 input tied to zero. This operator is used to compute $(G_{3:0}, P_{3:0}) \circ \overline{(G_{15:12}, P_{15:12})}$, which is necessary for the computation of

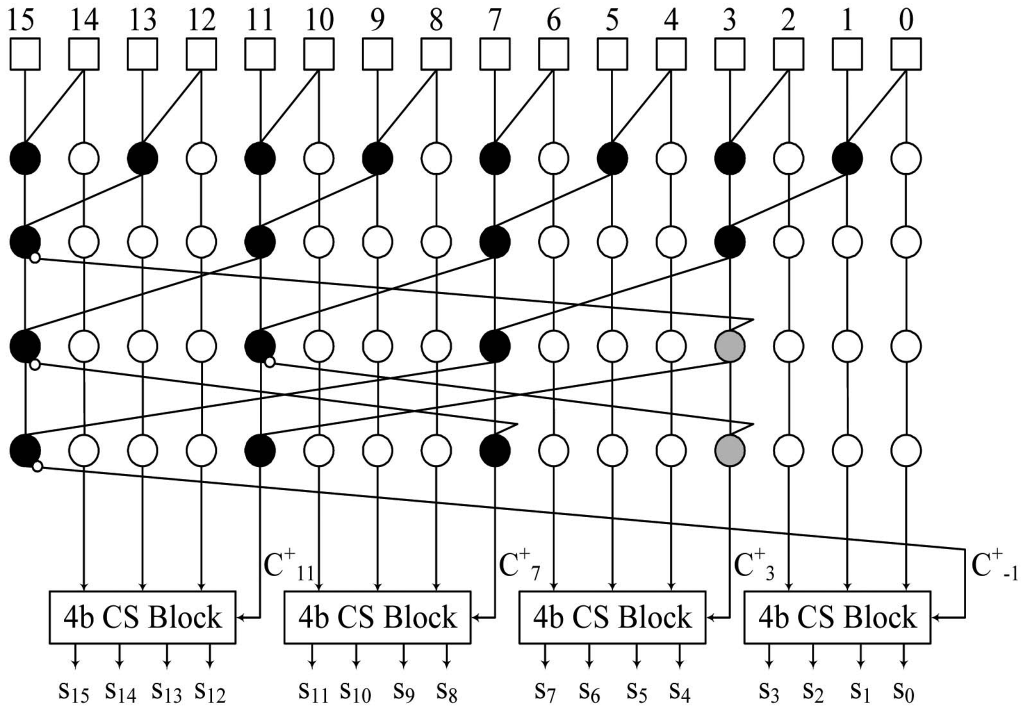


Fig. 9. Proposed sparse-4 modulo $2^{16}+1$ diminished-1 adder.

both C_3^+ and C_{11}^+. Its vertical successor is also replaced by a gray operator that computes the final:

$$C_3^+ \leftrightarrow (G_{3:0}, P_{3:0}) \circ \overline{(G_{15:12}, P_{15:12}) \circ (G_{11:4}, P_{11:4})}.$$

5 Comparisons

In this section, at first we compare the diminished-1 adders that use the totally regular parallel-prefix IEAC adders presented in the previous section (hereafter called proposed diminished-1) against the diminished-1 adders proposed by [4], [5] and those that use the IEAC proposed in [23], [24], [28]. We consider that all diminished-1 adders can handle true zero operands and indicate true zero results. That is, all diminished-1 adders that use an IEAC adder have all necessary modifications discussed in Section 3. For the adders of [4], [5], we consider that the carry output computed by the CLA unit is used as a late increment carry signal in the successor integer adder. For the latter, we consider that it follows the Ladner-Fisher (LF) proposal augmented by a carry increment prefix level. For the IEAC adders of [23], we consider that the first $\log_2 n$ prefix levels may either follow the Ladner-Fisher (LF) or the Kogge-Stone (KS) proposal. Finally, we examine both the reduced area parallel prefix (RAPP) and the full parallel prefix (FPP) architectures of the IEAC adders that use Ling carries [28].

For attaining our comparison data, we first generated structural Verilog descriptions of all adders under comparison. After extensive simulations that verified the correctness of each description, each design was synthesized and mapped in a power-characterized 90 nm implementation technology [44]. For the synthesis and mapping of the designs, we used the Synopsys® Design Compiler® tool in its topographical mode. In this mode, for achieving faster timing closure, the tool performs floorplanning in parallel with synthesis and mapping and the design is annotated with wiring lengths and fan-out and parasitic capacitances coming directly from the floorplan of the design and not from a wire load model. We assumed that each adder's input and output is driven by the output of a D flip-flop and drives the input of a D flip-flop of the same implementation library, respectively. A typical corner (1.2 V, 25°C) was considered.

Each adder was recursively optimized for speed until the tool was unable to produce a faster design. A final area recovery step was then applied. For obtaining power data, we followed a simulation driven approach, by applying 2^{16} random input vectors at a 500 MHz frequency at each netlist (the same vectors were applied at the corresponding netlists of the architectures under comparison) and measured the average power dissipation. The attained results are given in Table 1.

Our experimental data reveal that the proposed diminished-1 adders offer a higher operation speed than those of [4], [5] and the diminished-1 adders that result by using the IEAC adders proposed by [23] and the RAPP architecture of [28]. Their speed is also slightly higher than that of the adders resulting from the IEAC adders of [24], mainly because the reduction in the total number of critical paths. In this way, the true critical paths are favored receiving a larger drive strength and see less off-path loading. At the same time, wiring delays are also reduced due to more compact layout. The proposed diminished-1 adders are slightly slower (less than 5.5 percent) than those that use the FPP architecture of [28] but the difference is diminishing at the wider cases. Nevertheless, in this case, it should be noted that the proposed diminished-1 adders require significantly less area and consume significantly less power. The proposed adders require from 15.8 to 28.7 percent less area than the most compact earlier solution, from 18 to 52 percent less area than the adders resulting from the use of the IEAC adders of [24] and from 28 to 58 percent less

TABLE 1
Experimental Results for Diminished-1 Adders

	[4], [5]			[23] LF			[23] KS			[24]		
n	Delay	Area	Power	Delay	Area	Power	Delay	Area	Power	Delay	Area	Power
4	301	1,137	0.389	307	829	0.267	305	821	0.272	278	847	0.288
8	376	2,436	0.764	376	1,845	0.667	376	2,029	0.713	344	2,198	0.782
16	446	5,149	1.706	431	4,063	1.330	449	4,988	1.757	403	5,216	1.996
32	535	11,470	3.982	514	8,696	2.983	545	11,628	4.414	478	13,369	5.152

	[28] FPP			[28] RAPP			Proposed		
n	Delay	Area	Power	Delay	Area	Power	Delay	Area	Power
4	N/A			N/A			272	691	0.197
8	317	2,121	0.750	348	1,974	0.698	334	1,528	0.524
16	381	5,221	2.025	424	4,823	1.818	394	2,895	0.991
32	458	15,211	5.694	498	12,409	4.717	464	6,359	2.194

Delay results are given in picoseconds, area results in square micrometers, and average power results in milliwatts.

TABLE 2
Experimental Results for Weighted Adders

	[31]			[32] using the proposed diminished-1 adders		
n	Delay	Area	Power	Delay	Area	Power
4	420	1,009	0.277	387	887	0.209
8	484	2,344	0.720	447	1,796	0.461
16	550	5,709	1.901	512	3,217	0.886
32	623	14,161	5.049	584	6,745	1.855

Delay results are given in picoseconds, area results in square micrometers, and average power results in milliwatts.

area than those that use the IEAC FPP adder proposal of [28]. Finally, the power consumption savings offered range from 21 to 61.5 percent in the examined adders cases.

The results of Table 1 indicate that if we use the proposed IEAC adders as building blocks for weighted adders in the architecture proposed in [32], the resulting weighted adders will also be more efficient than those that use any other previously proposed IEAC adder architecture. Therefore, we compare the weighted adders that result by using the proposed IEAC adders, against the weighted adders of [31], which have been shown to be more efficient in both area and delay terms than the proposal of [30]. The comparison results given in Table 2, reveal that the weighted adders resulting from using the proposed IEAC adders in the scheme proposed in [32] are faster, smaller and consume less power than the adders of [31] throughout the examined range. The savings offered are about 7 percent in operation speed and range from 12 to 52 percent and from 24 to 63 percent in the required implementation area and average power consumption, respectively.

6 Unified Approach to the Design of Modulo $2^n \pm 1$ Adders

Several area-time-power efficient architectures (for example, [33], [34], [40]) have been proposed for the simpler case of modulo $2^n - 1$ addition. These architectures preserve all the benefits of parallel-prefix carry computation units and can be easily designed for every n. More specifically, Dimitrakopoulos et al. [33] generalized the design of such units for all values of n and has provided easy-to-follow topographical design rules. The resulting structures for $n \neq 2^k$ save significant amount of area without sacrificing delay.

We therefore conclude that mapping the diminished modulo $2^n + 1$ adder design problem to that of modulo $2^n - 1$ addition, would be beneficial given all the efficient architectures that have been proposed for the latter. In the following, we show that this mapping requires a constant time postprocessing stage and analyze its area and time overhead.

6.1 Modulo $2^n \pm 1$ Unification Theory

In order to unify the parallel-prefix modulo $2^n \pm 1$ addition principles, we need to explore the relation between the carries of these two addition operators, that is, between C_i^- and C_i^+.

The relationship that connects the recirculating carry-out bits C_{n-1}^- and C_{n-1}^+ that are employed for the derivation of the sum bits on the least-significant position zero is trivial

$$C_{n-1}^+ = \overline{C_{n-1}^-} = \overline{G_{n-1:0}} = C_{-1}^+. \quad (6)$$

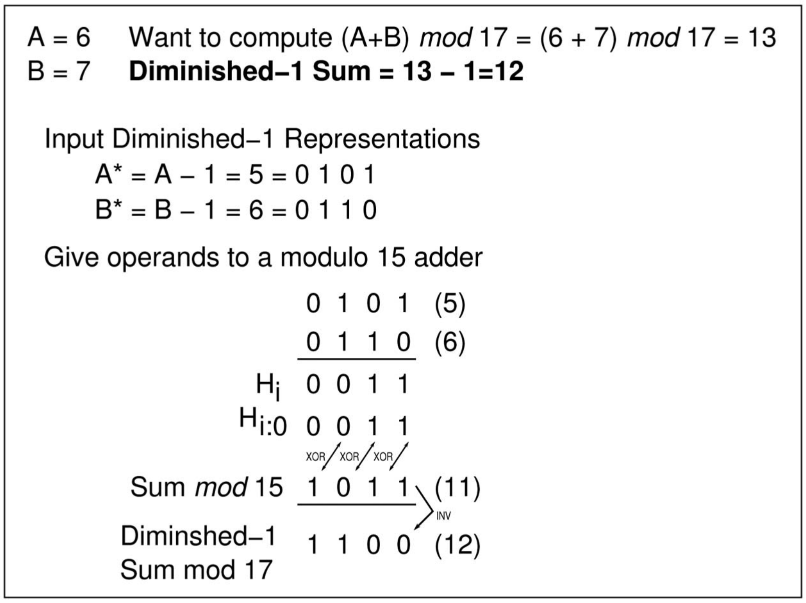

A = 6 Want to compute (A+B) mod 17 = (6 + 7) mod 17 = 13
B = 7 Diminished−1 Sum = 13 − 1=12
Input Diminished−1 Representations
A* = A − 1 = 5 = 0 1 0 1
B* = B − 1 = 6 = 0 1 1 0
Give operands to a modulo 15 adder
0 1 0 1 (5)
0 1 1 0 (6)
Hi 0 0 1 1
Hi:0 0 0 1 1
XOR XOR XOR
Sum mod 15 1 0 1 1 (11)
INV
Diminshed−1 Sum mod 17 1 1 0 0 (12)

Fig. 10. An arithmetic example of the new approach for the computation of the modulo 2^n+1 sum via the result of the corresponding modulo 2^n-1 addition.

Please note that in this cases both carries are considered as incoming carries from bit position -1.

For all other bit positions with $0 \le i < n-1$, the relation between C_i^- and C_i^+ is given by the following Theorem.

Theorem 2. *$C_i^+ = C_i^- \oplus (\overline{G_{i:0}} \cdot P_{i:0})$, with $i < n-1$.*

Proof. By the definition of the XOR operator $C_i^- \oplus (\overline{G_{i:0}} \cdot P_{i:0})$ can be written as

$$C_i^- \cdot (G_{i:0} + \overline{P_{i:0}}) + \overline{C_i^-} \cdot \overline{G_{i:0}} \cdot P_{i:0}.$$

Replacing C_i^- with its definition in the above relation we get

$$\begin{aligned}&(G_{i:0} + P_{i:0} \cdot G_{n-1:i+1}) \cdot (G_{i:0} + \overline{P_{i:0}})\\ &\quad + \overline{G_{i:0}} \cdot \left(\overline{P_{i:0}} + \overline{G_{n-1:i+1}}\right) \cdot \overline{G_{i:0}} \cdot P_{i:0}\\ &= G_{i:0} + \overline{G_{i:0}} \cdot P_{i:0} \cdot \overline{G_{n-1:i+1}} = G_{i:0} + P_{i:0} \cdot \overline{G_{n-1:i+1}}.\end{aligned}$$

Based on (4), the derived term is the definition of the carry signal C_i^+ in the case of diminished-1 modulo 2^n+1 addition. □

The direct consequence of the newly derived relationship is that we can compute the carries for the case of modulo 2^n+1 adders directly from a modulo 2^n-1 carry computation unit by a stage of XOR gates that will combine the carries C_i^- with the terms $\overline{G_{i:0}} \cdot P_{i:0}$.

At first, it may seem complicated to compute $\overline{G_{i:0}} \cdot P_{i:0}$ since it requires a complete carry tree to be added for the computation of $G_{i:0}$. However, based on Theorem 3 that we introduce in the following, the computation of $\overline{G_{i:0}} \cdot P_{i:0}$ is straightforward and can be implemented at low cost.

Theorem 3. *$\overline{G_{i:0}} \cdot P_{i:0} = H_{i:0}$, where $H_{i:0} = H_i \cdot H_{i-1} \cdot \dots \cdot H_0$.*

Proof. By definition we know that $G_{i:0} = G_i + P_i \cdot G_{i-1:0}$ and that $P_{i:0} = P_i \cdot P_{i-1:0}$. Therefore, the term $\overline{G_{i:0}} \cdot P_{i:0}$ can be written as

$$\overline{G_i} \cdot \left(\overline{P_i} + \overline{G_{i-1:0}}\right) \cdot P_i \cdot P_{i-1:0} = \overline{G_i} \cdot P_i \cdot \overline{G_{i-1:0}} \cdot P_{i-1:0}.$$

The term $\overline{G_i} \cdot P_i$ can be easily proven that is equal to H_i. Hence,

$$\overline{G_{i:0}} \cdot P_{i:0} = H_i \cdot \left(\overline{G_{i-1:0}} \cdot P_{i-1:0}\right).$$

In the same manner, the term in the parentheses $\overline{G_{i-1:0}} \cdot P_{i-1:0} = H_{i-1} \cdot \overline{G_{i-2:0}} \cdot P_{i-2:0}$ leading to

$$\overline{G}_{i:0} \cdot P_{i:0} = H_i \cdot H_{i-1}\left(\overline{G}_{i-2:0} \cdot P_{i-2:0}\right).$$

Applying the same rule recursively n times we get

$$\overline{G_{i:0}} \cdot P_{i:0} = H_i \cdot H_{i-1} \cdot \ldots \cdot H_0 = H_{i:0}.$$

□

Therefore, from the two newly introduced theorems, the diminished-1 modulo 2^n+1 sum can be derived from the corresponding modulo 2^n-1 sum as follows: by definition, we know that

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

Replacing C_{i-1}^+ with its new value $C_{i-1}^- \oplus H_{i-1:0}$ we get that

$$S_i^+ = H_i \oplus C_{i-1}^- \oplus H_{i-1:0}.$$

We identify that $H_i \oplus C_{i-1}^-$ is the corresponding sum S_i^-. Thus, it holds that

$$S_i^+ = S_i^- \oplus H_{i-1:0} \quad \text{for } i \neq 0. \tag{7}$$

Also, based on (6), the sum bit S_0^+ is simply equal to $\overline{S_0^-}$.

An arithmetic example illustrating the derivation of a diminished-1 modulo 17 sum via a modulo 15 adder and some extra logic is given in Fig. 10.

According to Section 3, one or both the input operands in case of diminished-1 representation may be equal to zero. We chose to handle this case by setting the corresponding diminished-1 carries C_i^+ to zero. However, when using a modulo 2^n-1 adder for the implementation of a diminished-1 modulo 2^n+1 adder an even simpler approach can be employed: when at least of the input operands is zero, i.e., $a_z=1$ or $b_z=1$, then we ignore the term $H_{i-1:0}$ for the derivation of the bit S_i^+ and keep the original sum of the modulo 2^n-1 adder. This simple condition can be efficiently implemented by the following equations:

$$\begin{aligned} S_i^+ &= S_i^- \oplus H_{i-1:0} \cdot \overline{(a_z+b_z)} \quad \text{for } i \neq 0 \text{ and} \\ S_0^+ &= S_0^- \oplus \overline{(a_z+b_z)}. \end{aligned} \tag{8}$$

6.2 Hardware Implementation Discussion

Equation (7) reveals that the modulo 2^n+1 sum can be computed using a modulo 2^n-1 adder, followed by an extra stage of XOR gates. The half sum bits H_i are already available by the modulo 2^n-1 adder. The only extra circuitry required is that needed to compute all the intermediate products $H_{i-1:0}$, that is, a parallel-prefix structure of AND gates. Either a Kogge-Stone or a Ladner-Fischer parallel-prefix structure as the ones shown in Fig. 1 can be employed, where each node consists of just a 2-input AND gate. Therefore, this structure can be embedded inside a complete modulo 2^n-1 parallel-prefix adder (as the one shown in Fig. 5), by enhancing the functionality of the common nodes of the parallel-prefix tree.

Alternatively, the area overhead of this approach can be reduced if instead of $P_{i:0}$ we use the equivalent $H_{i:0}$ for the computation of modulo 2^n-1 carries C_i^- [40]. The AND gates already present in the prefix ($\circ$) operators are in this way also used then for computing the $H_{i:0}$ terms.

It is evident that the implementation of the modulo 2^n+1 adder via an equal-width modulo 2^n-1 adder, requires at least one extra logic level compared to the modulo adders of [24] and the ones proposed in Section 4. Therefore, their delay is expected to be worse than the aforementioned designs. For large wordlengths, the delay overhead is diminishing and the performance of both architectures (unified modulo $2^n \pm 1$ adder and customized modulo 2^n+1 adder) converges fast. More importantly, mapping the problem of the diminished-1 modulo 2^n+1 addition to that of modulo 2^n-1 addition allows to reuse all the efficient architectures proposed for the modulo 2^n-1 addition case and opens a whole new unexplored design space for diminished-1 modulo 2^n+1 adders.

7 Conclusions

Efficient modulo 2^n+1 adders are appreciated in a variety of computer applications including all RNS implementations. In this paper, two contributions are offered to the modulo 2^n+1 addition problem.

A novel architecture has been proposed that uses a sparse totally regular parallel-prefix carry computation unit. This architecture was derived by proving the inverted circular idempotency property of the parallel-prefix carry operator in modulo 2^n+1 addition and by introducing a new prefix operator that eliminates the need for a double computation tree in the earlier fastest proposals. The experimental results indicate that the proposed architecture heavily outperforms the earlier solutions in implementation area and power consumption, while offering a high execution rate.

The modulo 2^n+1 addition problem was also shown to be related to the modulo 2^n-1 addition problem. The unifying theory presented in this paper revealed that a simple postprocessing stage composed of an XOR gate for each output bit needs to be added to a modulo 2^n-1 adder for attaining a modulo 2^n+1 adder. As a result, every architecture that has been and more importantly that will be proposed for designing modulo 2^n-1 adders, can be reused for the design of modulo 2^n+1 adders.

References

[1] X. Lai and J.L. Massey, "A Proposal for a New Block Encryption Standard," *EUROCRYPT,* D.W. Davies, ed., vol. 547, pp. 389-404, Springer, 1991.

[2] R. Zimmermann et al., "A 177 Mb/s VLSI Implementation of the International Data Encryption Algorithm," *IEEE J. Solid-State Circuits,* vol. 29, no. 3, pp. 303-307, Mar. 1994.

[3] H. Nozaki et al., "Implementation of RSA Algorithm Based on RNS Montgomery Multiplication," *Proc. Third Int'l Workshop Cryptographic Hardware and Embedded Systems,* pp. 364-376, 2001.

[4] Y. Morikawa, H. Hamada, and K. Nagayasu, "Hardware Realisation of High Speed Butterfly for the Maximal Length Fermat Number Transform," *Trans. IECE,* vol. J66-D, no. 1, pp. 81-88, 1983.

[5] M. Benaissa, S.S. Dlay, and A.G.J. Holt, "CMOS VLSI Design of a High-Speed Fermat Number Transform Based Convolver/Correlator Using Three-Input Adders," *Proc. IEE,* vol. 138, no. 2, pp. 182-190, Apr. 1991.

[6] V.K. Zadiraka and E.A. Melekhina, "Computer Implementation of Efficient Discrete-Convolution Algorithms," *Cybernetics and Systems Analysis,* vol. 30, no. 1, pp. 106-114, Jan. 1994.

[7] M.A. Soderstrand et al., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing.* IEEE Press, 1986.

[8] P.V.A. Mohan, *Residue Number Systems: Algorithms and Architectures.* Springer-Verlag, 2002.

[9] A. Omondi and B. Premkumar, *Residue Number Systems: Theory and Implementations.* Imperial College Press, 2007.

[10] J. Ramirez et al., "RNS-Enabled Digital Signal Processor Design," *Electronics Letters,* vol. 38, no. 6, pp. 266-268, Mar. 2002.

[11] J. Ramirez et al., "Design and Implementation of High-Performance RNS Wavelet Proccessors Using Custom IC Technologies," *J. VLSI Signal Processing Systems,* vol. 34, no. 3, pp. 227-237, July 2003.

[12] J. Ramirez et al., "High Performance, Reduced Complexity Programmable RNS-FPL Merged FIR Filters," *Electronics Letters,* vol. 38, no. 4, pp. 199-200, Feb. 2002.

[13] G.C. Cardarilli, A. Nannarelli, and M. Re, "Reducing Power Dissipation in FIR Filters Using the Residue Number System," *Proc. 43rd IEEE Midwest Symp. Circuits and Systems,* pp. 320-323, Aug. 2000.

[14] Y. Liu and E.M.-K. Lai, "Moduli Set Selection and Cost Estimation for RNS-Based FIR Filter and Filter Bank Design," *Design Automation for Embedded Systems,* vol. 9, no. 2, pp. 123-139, June 2004.

[15] U. Meyer-Bäse, A. Garcia, and F. Taylor, "Implementation of a Communications Channelizer Using FPGAs and RNS Arithmetic," *J. VLSI Signal Processing Systems,* vol. 28, nos. 1/2, pp. 115-128, May/June 2001.

[16] J. Ramirez et al., "Fast RNS FPL-Based Communications Receiver Design and Implementation," *Proc. 12th Int'l Conf. Field Programmable Logic,* pp. 472-481, 2002.

[17] M. Panella and G. Martinelli, "An RNS Architecture for Quasi-Chaotic Oscillators," *J. VLSI Signal Processing Systems,* vol. 33, no. 1, pp. 199-220, Jan./Feb. 2003.

[18] R. Chokshi, K.S. Berezowski, A. Shrivastava, and S.J. Piestrak, "Exploiting Residue Number System for Power-Efficient Digital Signal Processing in Embedded Processors," *Proc. Int'l Conf. Compilers, Architecture, and Synthesis for Embedded Systems (CASES '09),* pp. 19-28, 2009.

[19] L.M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform," *IEEE Trans. Acoustics, Speech and Signal Processing,* vol. ASSP-24, no. 5, pp. 356-359, Oct. 1976.

[20] G. Jaberipur and B. Parhami, “Unified Approach to the Design of Modulo-($2^n \pm 1$) Adders Based on Signed-LSB Representation of Residues,” *Proc. 19th IEEE Symp. Computer Arithmetic,* pp. 57-64, 2009.

[21] J.J. Shedletsky, “Comment on the Sequential and Indeterminate Behavior of an End-Around-Carry Adder,” *IEEE Trans. Computers,* vol. C-26, no. 3, pp. 271-272, Mar. 1977.

[22] R. Zimmermann, “Binary Adder Architectures for Cell-Based VLSI and Their Synthesis,” PhD dissertation, Swiss Fed. Inst. of Technology, 1997.

[23] R. Zimmerman, “Efficient VLSI Implementation of Modulo ($2^n \pm 1$) Addition and Multiplication,” *Proc. 14th IEEE Symp. Computer Arithmetic,* pp. 158-167, Apr. 1999.

[24] H.T. Vergos, C. Efstathiou, and D. Nikolos, “Diminished-One Modulo $2^n + 1$ Adder Design,” *IEEE Trans. Computers,* vol. 51, no. 12, pp. 1389-1399, Dec. 2002.

[25] C. Efstathiou, H.T. Vergos, and D. Nikolos, “Modulo $2^n \pm 1$ Adder Design Using Select Prefix Blocks,” *IEEE Trans. Computers,* vol. 52, no. 11, pp. 1399-1406, Nov. 2003.

[26] S.-H. Lin and M.-H. Sheu, “VLSI Design of Diminished-One Modulo $2^n + 1$ Adder Using Circular Carry Selection,” *IEEE Trans. Circuits and Systems II,* vol. 55, no. 9, pp. 897-901, Sept. 2008.

[27] G. Dimitrakopoulos and D. Nikolos, “High-Speed Parallel-Prefix VLSI Ling Adders,” *IEEE Trans. Computers,* vol. 54, no. 2, pp. 225-231, Feb. 2005.

[28] H.T. Vergos and C. Efstathiou, “Efficient Modulo $2^n + 1$ Adder Architectures,” *Integration, the VLSI J.,* vol. 42, no. 2, pp. 149-157, Feb. 2009.

[29] M. Bayoumi, G. Jullien, and W. Miller, “A VLSI Implementation of Residue Adders,” *IEEE Trans. Circuits and Systems,* vol. CAS-34, no. 3, pp. 284-288, Mar. 1987.

[30] A. Hiasat, “High-Speed and Reduced-Area Modular Adder Structures for RNS,” *IEEE Trans. Computers,* vol. 51, no. 1, pp. 84-89, Jan. 2002.

[31] C. Efstathiou, H.T. Vergos, and D. Nikolos, “Fast Parallel-Prefix Modulo $2^n + 1$ Adders,” *IEEE Trans. Computers,* vol. 53, no. 9, pp. 1211-1216, Sept. 2004.

[32] H.T. Vergos and C. Efstathiou, “A Unifying Approach for Weighted and Diminished-1 Modulo $2^n + 1$ Addition,” *IEEE Trans. Circuits and Systems II,* vol. 55, no. 10, pp. 1041-1045, Oct. 2008.

[33] G. Dimitrakopoulos, H.T. Vergos, D. Nikolos, and C. Efstathiou, “A Family of Parallel-Prefix Modulo $2^n - 1$ Adders,” *Proc. IEEE Int'l Conf. Application-Specific Systems, Architectures and Processors,* pp. 326-336, 2003.

[34] J. Chen and J.E. Stine, “Parallel Prefix Ling Structures for Modulo $2^n - 1$ Addition,” *Proc. 20th IEEE Int'l Conf. Application-Specific Systems, Architectures and Processors,* pp. 16-23, July 2009.

[35] R.P. Brent and H.T. Kung, “A Regular Layout for Parallel Adders,” *IEEE Trans. Computers,* vol. C-31, no. 3, pp. 260-264, Mar. 1982.

[36] P.M. Kogge and H.S. Stone, “A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations,” *IEEE Trans. Computers,* vol. C-22, no. 8, pp. 786-792, Aug. 1973.

[37] R.E. Ladner and M.J. Fischer, “Parallel Prefix Computation,” *J. ACM,* vol. 27, no. 4, pp. 831-838, 1980.

[38] S. Knowles, “A Family of Adders,” *Proc. 14th IEEE Symp. Computer Arithmetic,* pp. 30-34, 1999.

[39] S. Mathew, M. Anders, R.K. Krishnamurthy, and S. Borkar, “A 4-GHz 130-nm Address Generation Unit with 32-bit Sparse-Tree Adder Core,” *J. Solid-State Circuits,* vol. 38, no. 5, pp. 689-695, May 2003.

[40] L. Kalampoukas et al., “High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders,” *IEEE Trans. Computers,* vol. 49, no. 7, pp. 673-680, July 2000.

[41] R.A. Patel, M. Benaissa, and S. Boussakta, “Fast Parallel-Prefix Architectures for Modulo $2^n - 1$ Addition with a Single Representation of Zero,” *IEEE Trans. Computers,* vol. 56, no. 11, pp. 1484-1492, Nov. 2007.

[42] G. Dimitrakopoulos, D.G. Nikolos, D. Nikolos, H.T. Vergos, and C. Efstathiou, “New Architectures for Modulo $2^n - 1$ Adders,” *Proc. IEEE Int'l Conf. Electronics, Circuits, and Systems,* 2005.

[43] H.T. Vergos and D. Bakalis, “On the Use of Diminished-1 Adders for Weighted Modulo $2^n + 1$ Arithmetic Components,” *Proc. 11th Euromicro Conf. Digital System Design,* pp. 752-759, Sept. 2008.

[44] Synopsys Inc., “SAED 90 nm EDK,” https://www.synopsys.com/apps/protected/university/members.html, 2011.

Haridimos T. Vergos received the diploma in computer engineering, and the PhD degree from the Department of Computer Engineering and Informatics, University of Patras, Greece, where he currently holds an associate professor and the Head of the Hardware and Computer Architecture Division. He was a member of Atmel Multimedia and Communications Group, and worked on the development of the first IEEE 802.11 compliant wireless MAC device. His research interests include computer arithmetic and architecture, dependable system architectures, and low power design and low power test. He holds one worldwide patent and has authored or coauthored more than 70 scientific papers. He is a member of the IEEE.

Giorgos Dimitrakopoulos received the PhD degree in computer engineering from the University of Patras, Greece, in 2007. He is a lecturer in the Department of Informatics and Communications Engineering at the University of West Macedonia, Kozani, Greece. In 2008-2010, he held a postdoctoral research position at the Computer Architecture and VLSI Systems Laboratory, Institute of Computer Science, of the Foundation for Research and Technology-Hellas (FORTH), in Heraklion, Crete, Greece, and he taught digital design related courses to the Computer Science Department of the University of Crete. His interests are in the design of digital integrated circuits focusing on on-chip interconnection network architectures, media-enhanced microprocessors, and computer arithmetic as well as ultra low voltage circuit design. He is a member of the technical chamber of Greece and the IEEE.