Diminished-One Modulo $2^n + 1$ Adder Design

Haridimos T. Vergos, Costas Efstathiou, and Dimitris Nikolos, *Member*, *IEEE*

Abstract—This paper presents two new design methodologies for modulo $2^n + 1$ addition in the diminished-one number system. The first design methodology leads to carry look-ahead, whereas the second to parallel-prefix adder implementations. VLSI realizations of the proposed circuits in a standard-cell technology are utilized for quantitative comparisons against the existing solutions. Our results indicate that the proposed carry look-ahead adders are area and time efficient for small values of n, while for the rest values of n the proposed parallel-prefix adders are considerably faster than any other already known in the open literature.

Index Terms—Modulo $2^n + 1$ addition, carry look-ahead addition, parallel-prefix adders, diminished-one number representation, VLSI adders.

1 Introduction

Modulo arithmetic has been used in digital computing systems for various purposes for many years. In particular, modulo $2^n + 1$ arithmetic appears to play an important role in many algorithms.

A first application field is in Residue Number Systems (RNS) [1], [2], [3], [4]. In an RNS based application, every number X is represented by a sequence of residues $(X_1, X_2, \ldots, X_M)$, where $X_i = X \bmod p_i$. The p_is, $1 \le i \le M$, comprise the base of the RNS and are pairwise relative prime integers. A two operand RNS operation, suppose $\diamondsuit$, is defined as

$$(Z_1, Z_2, \ldots, Z_M) = (X_1, X_2, \ldots, X_M) \diamondsuit (Y_1, Y_2, \ldots, Y_M),$$

where $Z_i = (X_i \diamondsuit Y_i) \bmod p_i$. For most RNS applications $\diamondsuit$ is either addition, subtraction, or multiplication. Since the computation of Z_i only depends upon X_i, Y_i, and p_i, each Z_i is computed in parallel in a separate arithmetic unit, often called *channel*. Moduli choices of the form $\{2^n - 1, 2^n, 2^n + 1\}$ [5] have received significant attention because they offer very efficient circuits in the $\text{area} \times \text{time}^2$ product sense [6]. Addition in such systems is performed using three channels that, in fact, are a modulo $2^n - 1$ (equivalently, one's complement), a modulo 2^n, and a modulo $2^n + 1$ adder [1], [4]. The addition delay in an RNS application which uses the above moduli is dictated by the modulo $2^n + 1$ channel. The latter means that, if we can cut down the time required for modulo $2^n + 1$ addition, we also cut down the addition time in an RNS application.

Modulo $2^n + 1$ adders are also utilized as the last stage adder of modulo $2^n + 1$ multipliers. Modulo $2^n + 1$ multipliers find applicability in pseudorandom number generation [7], cryptography [8], [9], [10], and in the Fermat number transform, which is an effective way to compute convolutions [11].

Leibowitz, in [12], has proposed the *diminished-one* number system. In the diminished-one number system, each number X is represented by $X^* = X - 1$. The representation of 0 is treated in a special way. Since the adoption of this system leads to modulo $2^n + 1$ adders and multipliers of n bits wide operands, it has been used for many residue number system implementations, for example, [13], [14].

Efficient VLSI implementations of modulo $2^n + 1$ adders for the diminished-one number system have recently been presented in [15], [16]. The adders presented in [15], [16], although fast, are, according to the comparison presented in [15], still slower than the fastest modulo 2^n adders or the fastest modulo $2^n - 1$ adders presented in [17]. Therefore, their use in an RNS application would still limit the performance of the system.

In this paper, we derive two new design methodologies for modulo $2^n + 1$ adders in the diminished-one number system. The first one leads to traditional Carry Look-Ahead (CLA), while the second to parallel-prefix adder architectures. Using implementations in a static CMOS technology, we show that the proposed CLA adder design methodology leads to more area and time efficient implementations than those presented in [15], [16] for small operand widths. For wider operands, the proposed parallel-prefix design methodology leads to considerably faster adder implementations than those presented in [15], [16] and as fast as the integer or the modulo $2^n - 1$ architecture presented in [17].

The rest of this paper is organized as follows: The foundations of speeding up the addition operation are revisited in the next section. The derivation of our new architectures is presented in Section 3. Comparative results that show the efficiency of the proposed architectures against the existing solutions are presented in Section 4. Conclusions are given in the last section.

2 Foundations

In order to speed up the addition operation, the carry computation time should be minimized. One solution is to use CLA adders [4], [18].

- *H.T. Vergos and D. Nikolos are with the Computer Engineering & Informatics Department, Patras University, 26 500, Greece, and with the Computer Technology Institute, 3Kolokotroni St., 26 221, Patras, Greece. E-mail: vergos@ceid.upatras.gr, nikolosd@cti.gr.*
- *C. Efstathiou is with the Informatics Department, TEI of Athens, Ag. Spyridonos St., 12 210, Egaleo, Athens, Greece. E-mail: cefsta@teiath.gr.*

Manuscript received 9 Apr. 2001; revised 10 Octo. 2001; accepted 14 Mar. 2002.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 113956.

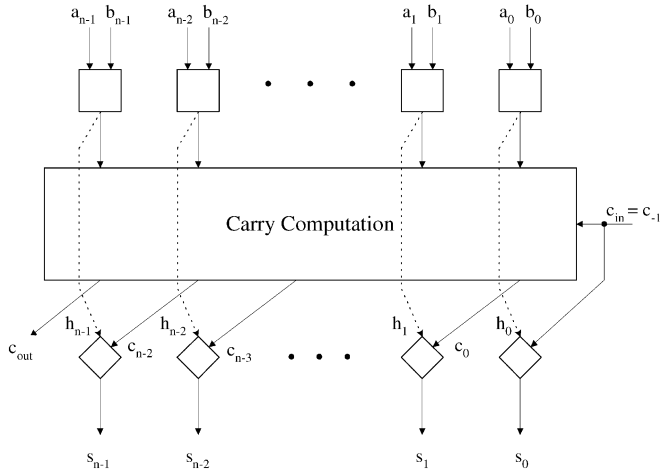

Fig. 1. Block diagram of a CLA adder.

Let $A = a_{n-1}a_{n-2}\ldots a_1a_0$ and $B = b_{n-1}b_{n-2}\ldots b_1b_0$ be two n-bit numbers and $S = s_{n-1}s_{n-2}\ldots s_1s_0$ their sum. For describing the carry computation problem, two terms are commonly used:

- the carry generate term, $g_i = a_i \cdot b_i$ and
- the carry propagate term, $p_i = a_i + b_i$.

The computation of the carries in the various CLA implementations is based on the recursive formula $c_i = g_i + p_i \cdot c_{i-1}$. If we denote the exclusive-OR operation by $\oplus$, the sum bits are given by $s_i = h_i \oplus c_{i-1}$, where $h_i = a_i \oplus b_i$ is the half-sum. (p_i may also be defined as $p_i = h_i = a_i \oplus b_i$, but, since this leads to somewhat slower implementations, we will not adopt it in this work).

The architecture of every CLA adder is given in Fig. 1. The implementation of the $\square$ and $\diamondsuit$ operators is presented in Fig. 2.

The computation of the carries c_i can be done in parallel by the carry computation unit of Fig. 1 by designing it to implement the formula:

$$c_i = g_i + \sum_{j=0}^{i-1}\left(\prod_{k=j+1}^{i} p_k\right)\cdot g_j + \left(\prod_{k=0}^{i} p_k\right)\cdot c_{in}.$$

However, when the operand length is large, the number of inputs to the high order gates of the carry computation unit also becomes quite large. In the case of wide operands, it is profitable to design the carry computation unit with more than one level of CLA [18]. Under this approach, at each level of the carry computation unit, the inputs are divided into groups, a smaller CLA unit is employed for each group, and collective CLA units are introduced for carry computations between groups.

Fig. 3 indicates the block diagram of a two-level CLA adder. The carry computation unit of this adder is composed of a Group Propagate and Generate (GPG) unit, a Between Groups Carry Look-Ahead (BGCLA) unit, and a Group Carry Look-Ahead (GCLA) unit [18], [19]. The GPG unit is composed of $m = \lceil n/k \rceil$ subunits. Each subunit processes k bits of the input operands, except the last, which may process less if n is not exactly divisible by k. The jth

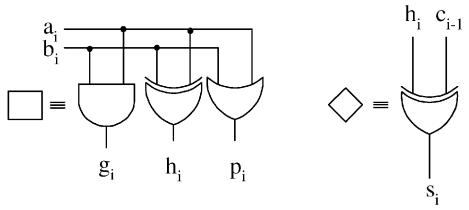

Fig. 2. Implementation of the operators of Fig. 1.

subunit implements the following equations for the group generate (gg_j) and group propagate (gp_j):

$$gg_j = g_{kj+k-1} + p_{kj+k-1}\cdot g_{kj+k-2} + \ldots + p_{kj+k-1}\cdot p_{kj+k-2}\cdot\ldots \cdot p_{kj+1}\cdot g_{kj}$$

and

$$gp_j = p_{kj+k-1}\cdot p_{kj+k-2}\cdot\ldots\cdot p_{kj+1}\cdot p_{kj},$$

where $0 \le j \le m-1$. The BGCLA unit implements the following equations for the group carries (gc_j):

$$gc_j = gg_j + \sum_{t=0}^{j-1}\left(\prod_{f=t+1}^{j} gp_f\right)gg_t + \left(\prod_{f=0}^{j} gp_f\right)gc_{-1},$$

where $0 \le j \le m-1$.

The GCLA unit is composed of m subunits, the jth of which implements the equation:

$$c_{kj+u} = g_{kj+u} + \sum_{t=0}^{u-1}\left(\prod_{f=t+1}^{u} p_{kj+f}\right)g_{kj+t} + \left(\prod_{t=0}^{u} p_{kj+t}\right)gc_{j-1},$$

for $u = 0, 1, \ldots, k-2$.

If carry computation in binary addition is treated as a prefix problem [20], another category of adders results, known as parallel-prefix adders. Carry computation is transformed into a prefix computation if the associative operator $\circ$ is defined according to [21] as:

$$(g_m, p_m)\circ(g_k, p_k) = (g_m + p_m\cdot g_k, p_m\cdot p_k).$$

Then, the carries are given by $c_i = G_i$, where G_i is the first member of the group relation (assuming that carry input $c_{in} = 0$):

$$(G_i, P_i) = \begin{cases} (g_0, p_0), & \text{if } i = 0 \\ (g_i, p_i)\circ(G_{i-1}, P_{i-1}), & \text{if } 1 \le i \le n-1. \end{cases}$$

The $\circ$ operation on a pair of (g_x, p_x) terms is usually represented as a node (node $\bullet$ of Fig. 4) and a whole carry computation unit is represented as a tree structured interconnection of such nodes. Several tree structures have been proposed in the past [20], [21], [22], [23]. Figs. 5 and 6 present, for n = 8, the tree structures proposed by Ladner-Fischer [20] and Kogge-Stone [22], respectively. The gate level implementation of the nodes is given in Fig. 4. The insertion of the buffering nodes, $\bigcirc$, is not mandatory. The adders that result following one of the proposed tree structures feature regular layout, but each structure has distinct implementation area, speed, and fan-out characteristics. For example, adders with a Ladner-Fischer prefix

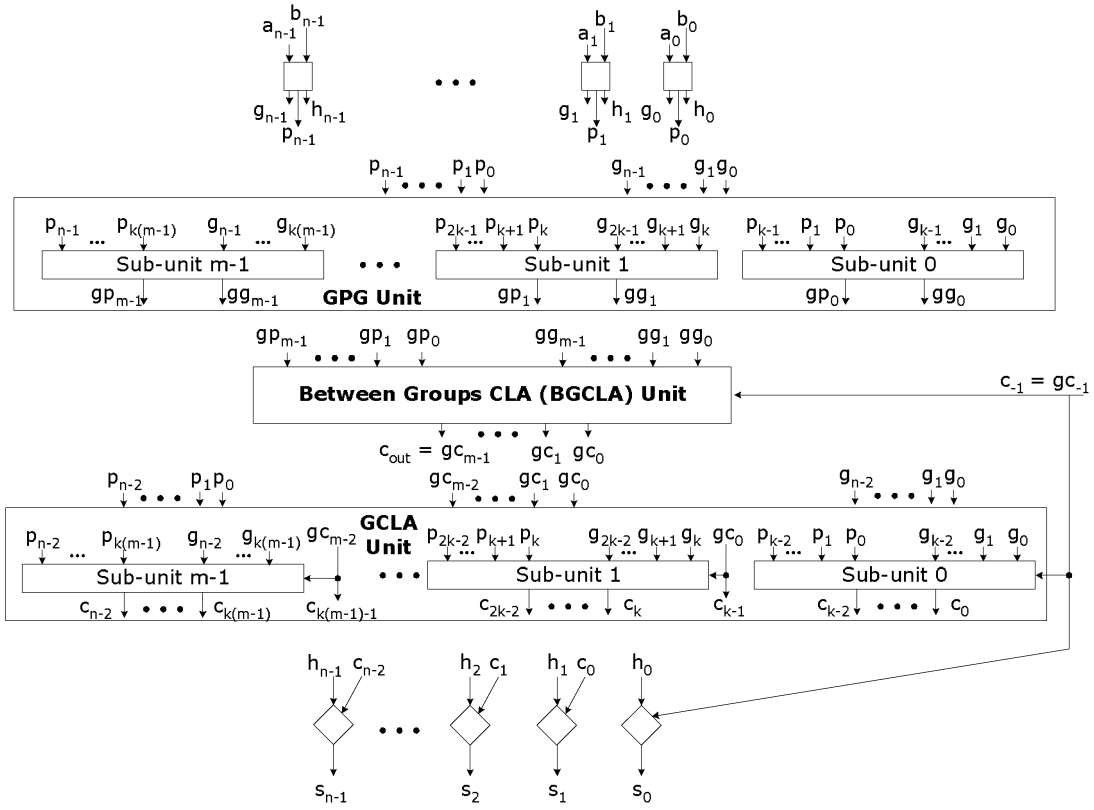

Fig. 3. Block diagram of a two-level CLA adder.

structure require less implementation area, but have unlimited fan-out compared to adders with a Kogge-Stone prefix structure. On the other hand, adders with a Kogge-Stone prefix structure are the faster reported [16], [23]. Knowles, in [23], examines combinations of the algorithms presented in [20], [22] and presents possible tradeoffs of fan-out and implementation area. Ladner-Fischer and Kogge-Stone prefix structures, according to [23], are the end cases of minimum implementation area and maximum speed, respectively, of a large family of addition structures, which all offer the minimum logical depth property.

Let X^* denote the representation of X in the diminished-one number system, that is, let $X^* = X - 1$. If S is the sum of A and B, it was shown in [15] that:

$$S^* \bmod (2^n + 1) = \begin{cases} (A^* + B^*) \bmod 2^n, & \text{if } A^* + B^* \geq 2^n \\ A^* + B^* + 1 & \text{otherwise.} \end{cases}$$

The above relation reveals that a diminished-one modulo $2^n + 1$ adder can be implemented by incrementing the sum by one when the carry output $c_{out} = 0$. This can be achieved by connecting the carry output via an inverter back to the carry input. To avoid combinational loops that, in some cases, lead to unwanted race conditions, the architecture of Fig. 7 has been proposed in [15], [16]. This architecture is based on the parallel-prefix adders with fast carry processing, proposed in [24].

It is obvious that the architecture of Fig. 7 results in diminished-one modulo $2^n + 1$ adders slower than the corresponding modulo 2^n adders and the fastest modulo $2^n - 1$ adders [17], [19] because:

- of the extra stage of • operators that is required and
- the reentering carry has a large fan-out.

In the next section, we present two novel design methodologies for diminished-one modulo $2^n + 1$ adders. The first one leads to CLA architectures and is derived by using the reentering carry's equation for simplifying the carry computation unit's equation. The second methodology leads

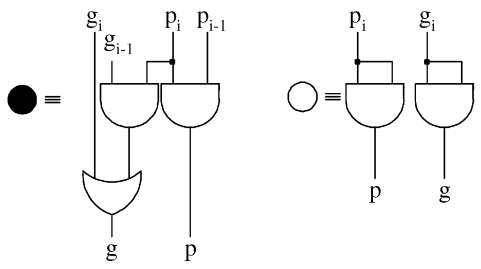

Fig. 4. Implementations of prefix logic operators.

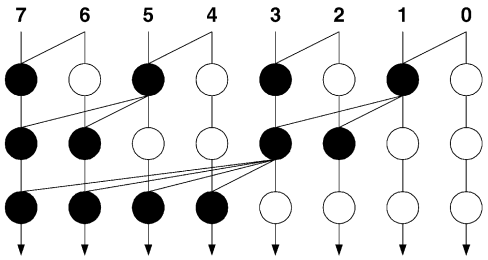

Fig. 5. Ladner-Fischer prefix structure.

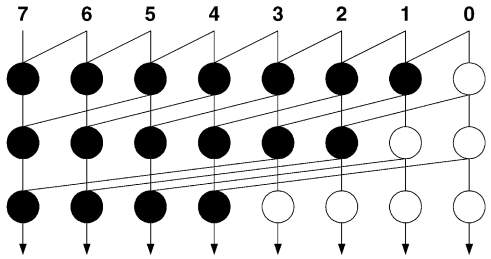

Fig. 6. Kogge-Stone prefix structure.

to new and faster parallel-prefix implementations derived by recirculating the carries in each level of the carry computation unit instead of having a final stage of reentering carry.

We conclude this section by the observation that all diminished-one adders suffer from the problem of interpreting correctly the zero output since it may either represent a valid zero output (that is, an addition with a result of 1) or a real zero output (that is, an addition with a result of 0). The following example clarifies these two possibilities.

Example 1. Consider the diminished-one modulo 9 addition of $A = 6$ with $B = 4$ and $C = 5$ with B. We then have: $A^* = 101_2$, $B^* = 011_2$, $C^* = 100_2$. The diminished-one modulo additions are presented in Table 1 (we denote the complement of x with $\overline{x}$).

The real zero output results only when the two inputs are complementary. This can be detected by the logical AND of the logical XOR of a_i and b_i. The Exclusive-OR gates required for the detection circuit are already present in the $\boxplus$ operators.

3 Novel Modulo $2^n + 1$ Adder Design

In this section, we propose novel diminished-one modulo $2^n + 1$ adder architectures. In the first and second subsections, we introduce conventional one and two level CLA adder architectures, respectively, while, in the third, we extend our theory to parallel-prefix implementations.

3.1 Novel One Level CLA Modulo $2^n + 1$ Adders

As explained in the previous section, the addition of the operands A^*, B^* in modulo $2^n + 1$ arithmetic involves the carry output (c_{out}) computation and its complement's addition. Therefore, we can assume that the diminished-one addition is a two cycles operation. During the first cycle, a normal binary addition takes place with zero

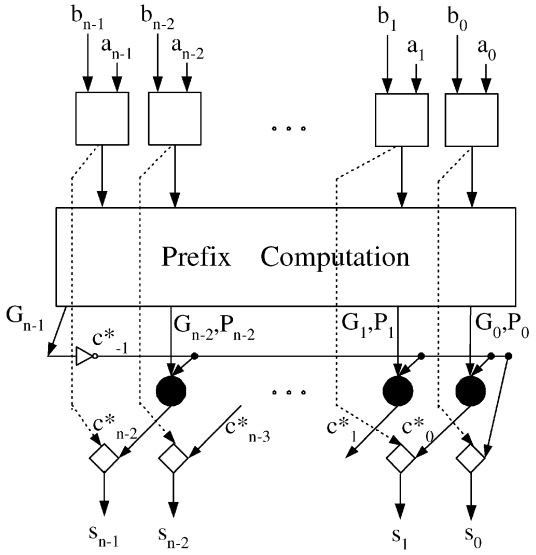

Fig. 7. Modulo $2^n + 1$ adder architecture for diminished-one arithmetic [15], [16].

carry input. During the second addition, we have that $c_{-1} = c_{in} = \overline{c_{out}} = \overline{c_{n-1}}$.

Lemma 1.

$$\overline{c_k} = \overline{p_k} + \overline{g_k} \cdot \overline{c_{k-1}}, \text{ for } 1 \le k \le n-1. \tag{2}$$

Proof. Since $\overline{g_i} = \overline{a_i \cdot b_i}$ and $\overline{p_i} = \overline{a_i + b_i}$, $\overline{g_i} \cdot \overline{p_i} = \overline{p_i}$ holds. Then:

$$\begin{aligned}\overline{c_k} &= \overline{(g_k + p_k \cdot c_{k-1})} = \overline{g_k} \cdot (\overline{p_k} + \overline{c_{k-1}}) \\ &= \overline{g_k} \cdot \overline{p_k} + \overline{g_k} \cdot \overline{c_{k-1}} = \overline{p_k} + \overline{g_k} \cdot \overline{c_{k-1}}.\end{aligned}$$

□

The inverted carry output that is added to $A^* + B^*$ by the operators of the last stage of Fig. 7 can then be derived by applying (2) recursively (note that $c_0 = g_0$):

$$\begin{aligned}c^*_{-1} &= \overline{c_{n-1}} = \overline{p_{n-1}} + \overline{g_{n-1}} \cdot \overline{c_{n-2}} \\ &= \overline{p_{n-1}} + \overline{g_{n-1}} \cdot (\overline{p_{n-2}} + \overline{g_{n-2}} \cdot \overline{c_{n-3}}) \\ &= \overline{p_{n-1}} + \overline{g_{n-1}} \cdot \overline{p_{n-2}} + \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \overline{c_{n-3}} = \ldots = \\ &= \overline{p_{n-1}} + \overline{g_{n-1}} \cdot \overline{p_{n-2}} + \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \overline{p_{n-3}} + \ldots \\ &\quad + \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \ldots \cdot \overline{g_2} \cdot \overline{p_1} + \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \ldots \cdot \overline{g_2} \cdot \overline{g_1} \cdot \overline{g_0}\end{aligned} \tag{3}$$

or, equivalently:

$$\begin{aligned}c^*_{-1} = \overline{p_{|n-1|_n}} + \overline{g_{|n-1|_n}} \cdot \overline{p_{|n-2|_n}} + \ldots + \overline{g_{|n-1|_n}} \cdot \overline{g_{|n-2|_n}} \cdot \ldots \\ \cdot \overline{g_{|2|_n}} \cdot \overline{p_{|1|_n}} + \overline{g_{|n-1|_n}} \cdot \overline{g_{|n-2|_n}} \cdot \ldots \cdot \overline{g_{|2|_n}} \cdot \overline{g_{|1|_n}} \cdot \overline{g_{|0|_n}},\end{aligned}$$

TABLE 1
Diminished-One Modulo Additions

A^*	=	101	C^*	=	100
B^*	=	011 +	B^*	=	011 +
c_{out}	=	1 000	c_{out}	=	0 111
$\overline{c_{out}}$	=	0 +	$\overline{c_{out}}$	=	1 +
Correct Result	=	000	Result indicating real zero	=	000

where $|X|_Y$ denotes the modulo Y of X.

Equation (3) is used in the sequel for deriving simplified equations for the carries generated during the second cycle, that is, the carries of the modulo $2^n + 1$ addition. Suppose that these carries are denoted by c_i^*, with $0 \le i \le n-2$. Then, they are given by the recursive formula $c_i^* = g_i + p_i \cdot c_{i-1}^*$. For $i = 0$ and by substituting c_{-1}^* by (3), we get:

$$c_0^* = g_0 + p_0 \cdot c_{-1}^* = g_0 + p_0 \cdot (\overline{p_{n-1}} + \overline{g_{n-1}} \cdot \overline{p_{n-2}} + \dots + \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2} \cdot \overline{p_1} + \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2} \cdot \overline{g_1} \cdot \overline{g_0}). \tag{4}$$

Given that

$$g_0 + p_0 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2} \cdot \overline{g_1} \cdot \overline{g_0} = g_0 + p_0 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2} \cdot \overline{g_1}$$

and that

$$p_0 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2} \cdot \overline{g_1} + p_0 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2} \cdot \overline{p_1} = p_0 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2} \cdot \overline{g_1},$$

(4) becomes:

$$\begin{aligned} c_0^* &= g_0 + p_0 \cdot \overline{p_{n-1}} + p_0 \cdot \overline{g_{n-1}} \cdot \overline{p_{n-2}} + \dots + p_0 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2} \cdot \overline{g_1} \\ &= g_{|n+0|_n} + p_{|n-1+1|_n} \cdot \overline{p_{|n-2+1|_n}} + \dots + p_{|n-1+1|_n} \cdot \overline{g_{|n-2+1|_n}} \cdot \dots \cdot \overline{g_{|1+1|_n}} \cdot \overline{g_{|0+1|_n}}. \end{aligned} \tag{5}$$

For $i = 1$ and by substituting c_0^* from (5), we get:

$$\begin{aligned} c_1^* &= g_1 + p_1 \cdot c_0^* \\ &= g_1 + p_1 \cdot (g_0 + p_0 \cdot \overline{p_{n-1}} + p_0 \cdot \overline{g_{n-1}} \cdot \overline{p_{n-2}} + \dots + p_0 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2} \cdot \overline{g_1}). \end{aligned} \tag{6}$$

Given that

$$g_1 + p_1 \cdot p_0 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2} \cdot \overline{g_1} = g_1 + p_1 \cdot p_0 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2}.$$

and that

$$p_1 \cdot p_0 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2} + p_1 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_3} \cdot \overline{p_2} = p_1 \cdot p_0 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2},$$

(6) becomes:

$$\begin{aligned} c_1^* &= g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot \overline{p_{n-1}} + p_1 \cdot p_0 \cdot \overline{g_{n-1}} \cdot \overline{p_{n-2}} + \dots + p_1 \cdot p_0 \cdot \overline{g_{n-1}} \cdot \overline{g_{n-2}} \cdot \dots \cdot \overline{g_2} \\ &= g_{|n+1|_n} + p_{|n-1+2|_n} \cdot g_{|n-2+2|_n} + \dots + p_{|n-1+2|_n} \cdot p_{|n-2+2|_n} \cdot \overline{g_{|n-3+2|_n}} \cdot \dots \cdot \overline{g_{|1+2|_n}} \cdot \overline{g_{|0+2|_n}}. \end{aligned} \tag{7}$$

We then can use c_1^* for computing c_2^*, c_2^* for computing c_3^*, and so on, up to:

$$\begin{aligned} c_{n-2}^* &= g_{n-2} + p_{n-2} \cdot g_{n-1} + \dots + p_{n-2} \cdot p_{n-3} \cdot \dots \cdot p_1 \cdot g_0 + p_{n-2} \cdot p_{n-3} \cdot \dots \cdot p_1 \cdot p_0 \cdot \overline{g_{n-1}} \\ &= g_{|n+n-2|_n} + p_{|n-1+n+1|_n} \cdot g_{|n-2+n-1|_n} + \dots + p_{|n-1+n-1|_n} \cdot p_{|n-2+n-1|_n} \cdot \dots \cdot p_{|1+n-1|_n} \cdot \overline{g_{|0+n-1|_n}}. \end{aligned}$$

The above equations sum up to the following general formula, which defines our proposed one level CLA diminished-one modulo $2^n + 1$ architecture:

$$c_i^* = g_{|n+i|_n}^* + \sum_{j=0}^{n-2} \left(\prod_{k=j+1}^{n-1} p_{|k+i+1|_n}^* \right) \cdot g_{|j+i+1|_n}^*, \tag{8}$$

where

$$g_i^* = \begin{cases} \overline{p_j}, & \text{if } j > i+1 \\ \overline{g_j}, & \text{if } j = i+1, \\ g_j, & \text{otherwise} \end{cases} \qquad p_i^* = \begin{cases} \overline{g_j}, & \text{if } j > i+1 \\ p_j, & \text{otherwise} \end{cases}$$

and $-1 \le i \le n-2$.

3.2 Novel Two Level CLA Adders

Thinking again of the modulo $2^n + 1$ addition as a two cycles operation, during the first cycle, a normal binary addition takes place with zero carry input. During the addition of the second cycle, we have that $c_{-1} = c_{in} = \overline{c_{out}} = \overline{gc_{m-1}}$.

Lemma 2. *If* $\overline{gq_j} = \overline{gg_j} + \overline{gp_j}$, *then* $\overline{gc_j} = \overline{gq_j} + \overline{gg_j} \cdot \overline{gc_{j-1}}$, *for* $1 \le j \le m-1$.

Proof.

$$\begin{aligned} \overline{gc_j} &= \overline{gg_j + gp_j \cdot gc_{j-1}} = \overline{gg_j} \cdot (\overline{gp_j} + \overline{gc_{j-1}}) \\ &= \overline{gg_j + gp_j} + \overline{gg_j} \cdot \overline{gc_{j-1}} = \overline{gq_j} + \overline{gg_j} \cdot \overline{gc_{j-1}}. \end{aligned}$$

□

The $\overline{gq_j}$, for $1 \le j \le m-1$, can be implemented by modifying slightly the *j*th GPG subunit, that is, by adding an OR-gate. Applying Lemma 2 recursively, $\overline{gc_{m-1}}$ can be expressed by (note that $gc_0 = gg_0$):

$$\overline{gc_{m-1}} = \overline{gq_{m-1}} + \overline{gg_{m-1}} \cdot \overline{gq_{m-2}} + \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \overline{gq_{m-3}} + \dots + \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \dots \cdot \overline{gg_1} \cdot \overline{gg_0}.$$

Therefore, during the modulo $2^n + 1$ addition, the BGCLA unit should obey the following equation:

$$gc_{-1}^* = \overline{gq_{m-1}} + \overline{gg_{m-1}} \cdot \overline{gq_{m-2}} + \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \overline{gq_{m-3}} + \dots + \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \dots \cdot \overline{gg_1} \cdot \overline{gg_0}$$

or, equivalently:

$$\begin{aligned} gc_{-1}^* = {} & \overline{gq_{|m-1|_m}} + \overline{gg_{|m-1|_m}} \cdot \overline{gq_{|m-2|_m}} + \dots \\ & + \overline{gg_{|m-1|_m}} \cdot \overline{gg_{|m-2|_m}} \cdot \dots \cdot \overline{gg_{|2|_m}} \cdot \overline{gq_{|1|_m}} \\ & + \overline{gg_{|m-1|_m}} \cdot \overline{gg_{|m-2|_m}} \cdot \dots \cdot \overline{gg_{|1|_m}} \cdot \overline{gg_{|0|_m}}. \end{aligned}$$

Using this equation, we can also get the equations for the rest of the carries that the modulo $2^n + 1$ adders' BGCLA unit should produce. The group carries of the modulo $2^n + 1$ addition are given by the recursive formula $gc_i^* = gg_i + gp_i \cdot gc_{i-1}^*$. For $i = 0$ and using the above equation, we get:

$$gc_0^* = gg_0 + gp_0 \cdot (\overline{gq_{m-1}} + \overline{gg_{m-1}} \cdot \overline{gq_{m-2}} + \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \overline{gq_{m-3}} + \ldots + \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \ldots \cdot \overline{gg_1} \cdot \overline{gg_0}). \quad (9)$$

Making use of the relations

$$gg_0 + gp_0 \cdot \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \ldots \cdot \overline{gg_1} \cdot \overline{gg_0} = gg_0 + gp_0 \cdot \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \ldots \cdot \overline{gg_1}$$

and

$$gp_0 \cdot \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \ldots \cdot \overline{gg_1} + gp_0 \cdot \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \ldots \cdot \overline{gq_1} = gp_0 \cdot \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \ldots \cdot \overline{gg_1},$$

(9) can be simplified to

$$gc_0^* = gg_0 + gp_0 \cdot \overline{gq_{m-1}} + gp_0 \cdot \overline{gg_{m-1}} \cdot \overline{gq_{m-2}} + \ldots + gp_0 \cdot \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \ldots \cdot \overline{gg_2} \cdot \overline{gg_1}$$

or, equivalently:

$$gc_0^* = gg_{|m+0|_m} + gp_{|m-1+1|_m} \cdot \overline{gq_{|m-2+1|_m}} + \ldots + gp_{|m-1+1|_m} \cdot \overline{gg_{|m-2+1|_m}} \cdot \ldots \cdot \overline{gg_{|1+1|_m}} \cdot \overline{gg_{|0+1|m}}.$$

Working in the same way, we substitute the equation derived above for gc_0^* in the equation $gc_1^* = gg_1 + gp_1 \cdot gc_0^*$ and get:

$$gc_1^* = gg_1 + gp_1 \cdot gg_0 + gp_1 \cdot gp_0 \cdot \overline{gq_{m-1}} + \ldots + gp_1 \cdot gp_0 \cdot \overline{gg_{m-1}} \cdot \overline{gg_{m-2}} \cdot \ldots \cdot \overline{gg_3} \cdot \overline{gg_2}$$

or, equivalently:

$$gc_1^* = gg_{|m+1|_m} + gp_{|m-1+2|_m} \cdot gg_{|m-2+2|_m} + \ldots + gp_{|m-1+2|_m} \cdot gp_{|m-2+2|_m} \cdot \ldots \cdot \overline{gg_{|1+2|_m}} \cdot \overline{gg_{|0+2|m}}.$$

We then can use gc_1^* for computing gc_2^* and so on, up to:

$$gc_{m-2}^* = gg_{m-2} + gp_{m-2} \cdot gg_{m-3} + \ldots + gp_{m-2} \cdot gp_{m-3} \cdot \ldots \cdot gp_1 \cdot gg_0 + gp_{m-2} \cdot gp_{m-3} \cdot \ldots \cdot gp_1 \cdot gp_0 \cdot \overline{gg_{m-1}}$$

or, equivalently:

$$gc_{m-2}^* = gg_{|m+m-2|_m} + gp_{|m-1+m-1|_m} \cdot gg_{|m-2+m-1|_m} + \ldots + gp_{|m-1+m-1|_m} \cdot gp_{|m-2+m-1|_m} \cdot \ldots \cdot gp_{|1+m-1|_m} \cdot \overline{gg_{|0+m-1|m}}.$$

The above equations can be expressed by the following general formula, which defines the logic of the BGCLA unit of a two-level CLA diminished-one modulo $2^n + 1$ adder when the GPG unit has been modified for producing the gq signals:

$$gc_j^* = gg_{|m+j|_m}^* + \sum_{t=0}^{m-2} \left(\prod_{f=t+1}^{m-1} gp_{|f+j+1|_m}^* \right) \cdot gg_{|t+j+1|_m}^*,$$

where

$$gg_z^* = \begin{cases} \overline{gq_j}, & \text{if } z > j+1 \\ \overline{gg_j}, & \text{if } z = j+1, \\ gg_j, & \text{otherwise} \end{cases} \qquad gp_z^* = \begin{cases} \overline{gg_j}, & \text{if } z > j+1 \\ gp_j, & \text{otherwise} \end{cases}$$

and $-1 \le j \le m-2$.

3.3 Novel Parallel-Prefix Modulo $2^n + 1$ Adders

The last stage • operators of Fig. 7 accept the reentrant carry $c_{-1}^* = \overline{G_{n-1}}$ and produce the modulo $2^n + 1$ carries $c_i^* = G_i + P_i \cdot \overline{G_{n-1}}$, for $0 \le i \le n-2$. By defining the complement of (G, P), which we will hereafter denote by $\overline{(G, P)}$, to be equal to $(\overline{G}, P)$, the modulo $2^n + 1$ carries can be expressed in parallel-prefix form as $c_i^* = G_i^*$, where:

$$(G_i^*, P_i^*) = \begin{cases} \overline{(G_{n-1}, P_{n-1})}, & \text{if } i = -1 \\ (G_i, P_i) \circ \overline{(G_{n-1}, P_{n-1})} & 0 \le i \le n-2 \end{cases} \quad (10)$$

and the terms G_{n-1}, P_{n-1} are computed according to (1).

As we have mentioned earlier, the design of Fig. 7, apart from adding an extra logic operator stage, also has the disadvantage that the reentering carry has a fan-out of n. Therefore, in the sequel, we utilize the idea of carry recirculation in each prefix level, which was introduced in [17], for transforming the computation of the carries c_i^* of the modulo $2^n + 1$ adder, for $-1 \le i \le n-2$, in a parallel-prefix computation problem. To this end, we need to define the group generate and group propagate functions $G_{a,b}$ and $P_{a,b}$ for the group of bits $a, a-1, \ldots, b$, with $a > b$, as

$$(G_{a,b}, P_{a,b}) = (g_a, p_a) \circ (g_{a-1}, p_{a-1}) \circ \ldots \circ (g_b, p_b).$$

Note that, according to (1), $(G_{a,0}, P_{a,0}) = (G_a, P_a)$.

The novel parallel-prefix modulo $2^n + 1$ adder design method which we propose in this paper is based on the following theorem:

Theorem 1. $(G_i^*, P_i^*) = (G_i, P_i) \circ \overline{(G_{n-1,i+1}, P_{n-1,i+1})}$ *for* $0 \le i \le n-2$.

Proof. From (10), we have:

$$\begin{aligned} (G_i^*, P_i^*) &= (G_i, P_i) \circ \overline{(G_{n-1}, P_{n-1})} = (G_i, P_i) \circ (\overline{G_{n-1}}, P_{n-1}) \\ &= (G_i + P_i \cdot \overline{G_{n-1}}, P_i \cdot P_{n-1}) \\ &= (G_i + P_i \cdot (\overline{G_{n-1,i+1} + P_{n-1,i+1} \cdot G_i}), P_{n-1}) \\ &= (G_i + P_i \cdot \overline{G_{n-1,i+1}}(\overline{P_{n-1,i+1}} + \overline{G_i}), P_{n-1}) \\ &= (G_i + P_i \cdot \overline{G_{n-1,i+1}} \cdot \overline{P_{n-1,i+1}} + P_i \cdot \overline{G_{n-1,i+1}} \cdot \overline{G_i}, P_{n-1}) \\ &= (G_i + P_i \cdot \overline{G_{n-1,i+1}}, P_i \cdot P_{n-1,i+1}) \\ &= (G_i, P_i) \circ (\overline{G_{n-1,i+1}}, P_{n-1,i+1}) \\ &= (G_i, P_i) \circ \overline{(G_{n-1,i+1}, P_{n-1,i+1})}. \end{aligned}$$

□

Theorem 1 implies that

$$(G_i^*, P_i^*) = (g_i, p_i) \circ \ldots \circ (g_0, p_0) \circ \overline{(g_{n-1}, p_{n-1}) \circ \ldots \circ (g_{i+1}, p_{i+1})}$$

holds for $0 \le i \le n-2$, that is, the carries in a modulo $2^n + 1$ adder can be computed using a prefix structure in which the carry at each position i not only depends on bits i to 0 but also on the bits $n-1$ through $i+1$. This can be achieved by recirculating the carries at each prefix level, instead of having a single final stage for the reentrant carry, as proposed in [17]. For achieving carry recirculation at each prefix level, more logic operators must be added to a

Kogge-Stone prefix structure and the outputs of the highest order 2^{m-1} operators of stage $m-1$ need to be driven to the lowest order 2^{m-1} operators of stage m.

The fastest parallel-prefix modulo 2^n and modulo $2^n - 1$ adders are capable of computing the carries within $m = \log_2 n$ prefix levels. For not delaying the addition operation in an RNS environment, the proposed adders should also be able to compute the carries within $\log_2 n$ prefix levels. However, the equations in the form produced by Theorem 1 cannot always be implemented in $\log_2 n$ prefix levels. Consider, for example, the following equation produced for c_0^* by Theorem 1 in a modulo 257 ($= 2^8 + 1$) adder:

$$c_0^* = (g_0, p_0) \circ \overline{(g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1)}$$

Since this equation has eight terms that need to be associated by the use of operator $\circ$, which treats its left and right operands distinctly, it is clear that the computation of c_0, as indicated by the above equation, cannot be achieved within the three parallel-prefix levels required by the corresponding modulo 2^8 (modulo 256) and modulo $2^8 - 1$ (modulo 255) adders. This is obvious since, for computing $\overline{(G_{7,1}, P_{7,1})}$, the three prefix levels are exhausted and a fourth level is required. To overcome this problem, we introduce the following theorem:

Theorem 2. *Suppose that* $(G_x, P_x) = (g, p) \circ \overline{(G, P)}$ *and* $(G_y, P_y) = \overline{((\overline{p}, \overline{g}) \circ (G, P))}$. *Then,* $G_x = G_y$.

Proof. Since

$$G_x = g + p \cdot \overline{G} = \overline{\overline{(g + p \cdot \overline{G})}} = \overline{\overline{g} \cdot (\overline{p} + G)} = \overline{(\overline{g} \cdot \overline{p} + \overline{g} G)} = \overline{(\overline{p} + \overline{g} G)}$$

and $G_y = \overline{(\overline{p} + \overline{g} G)}$, we get the required $G_x = G_y$. □

For area-time efficient parallel-prefix modulo $2^n + 1$ adder implementations, Theorem 2 needs to be applied j times recursively to the equations of the form

$$(g_i, p_i) \circ \ldots \circ (g_0, p_0) \circ \overline{(G_{n-1,i+1}, P_{n-1,i+1})}$$

produced by Theorem 1, until:

$$n - 1 - i + j = \begin{cases} n, & \text{if } i > \frac{n}{2} - 1 \\ \frac{n}{2}, & \text{if } i \leq \frac{n}{2} - 1. \end{cases}$$

Example 2. Consider the two 8-bit operands

$$A = a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$$

and

$$B = b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$$

in the diminished-one number representation. For constructing a parallel-prefix modulo 257 adder for them, from Theorem 1, we get the following equations:

$$c_{-1}^* = \overline{((g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ \overline{(g_1, p_1) \circ (g_0, p_0)})}$$

$$c_0^* = (g_0, p_0) \circ \overline{((g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ \overline{(g_2, p_2) \circ (g_1, p_1)})}$$

$$c_1^* = (g_1, p_1) \circ (g_0, p_0) \circ \overline{((g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2))}$$

$$c_2^* = (g_2, p_2) \circ (g_1, p_1) \circ (g_0, p_0) \circ \overline{((g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3))}$$

$$c_3^* = (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ (g_0, p_0) \circ \overline{((g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4))}$$

$$c_4^* = (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ (g_0, p_0) \circ \overline{((g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5))}$$

$$c_5^* = (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ (g_0, p_0) \circ \overline{((g_7, p_7) \circ (g_6, p_6))}$$

$$c_6^* = (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ (g_0, p_0) \circ \overline{((g_7, p_7))}.$$

As mentioned earlier these cannot be computed within three prefix levels. Applying Theorem 2, we get the following set of equations:

$$c_{-1}^* = \overline{((g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ \overline{(g_1, p_1) \circ (g_0, p_0)})}$$

$$c_0^* = \overline{((\overline{p_0}, \overline{g_0}) \circ (g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ \overline{(g_2, p_2) \circ (g_1, p_1)})}$$

$$c_1^* = \overline{((\overline{p_1}, \overline{g_1}) \circ (\overline{p_0}, \overline{g_0}) \circ (g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ \overline{(g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2))}}$$

$$c_2^* = \overline{((\overline{p_2}, \overline{g_2}) \circ (\overline{p_1}, \overline{g_1}) \circ (\overline{p_0}, \overline{g_0}) \circ (g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ \overline{(g_4, p_4) \circ (g_3, p_3))}}$$

$$c_3^* = (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ (g_0, p_0) \circ \overline{((g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4))}$$

$$c_4^* = (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ \overline{((\overline{p_0}, \overline{g_0}) \circ (g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5))}$$

$$c_5^* = (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ \overline{((\overline{p_1}, \overline{g_1}) \circ (\overline{p_0}, \overline{g_0}) \circ (g_7, p_7) \circ (g_6, p_6))}$$

$$c_6^* = (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ \overline{((\overline{p_2}, \overline{g_2}) \circ (\overline{p_1}, \overline{g_1}) \circ (\overline{p_0}, \overline{g_0}) \circ (g_7, p_7))}.$$

Fig. 8 presents the implementation indicated by the above equations. Note that only three prefix levels are required. Modified operators are presented in gray color in Fig. 8.

In the general case of n bits wide operands, the carries for a diminished-one modulo $2^n + 1$ adder can be computed using Theorems 1 and 2 in $\log_2 n$ stages, where stage i, $1 \leq i \leq (\log_2 n - 2)$, requires $3\frac{n}{2} - 2^i$ operators and the last two stages n operators.

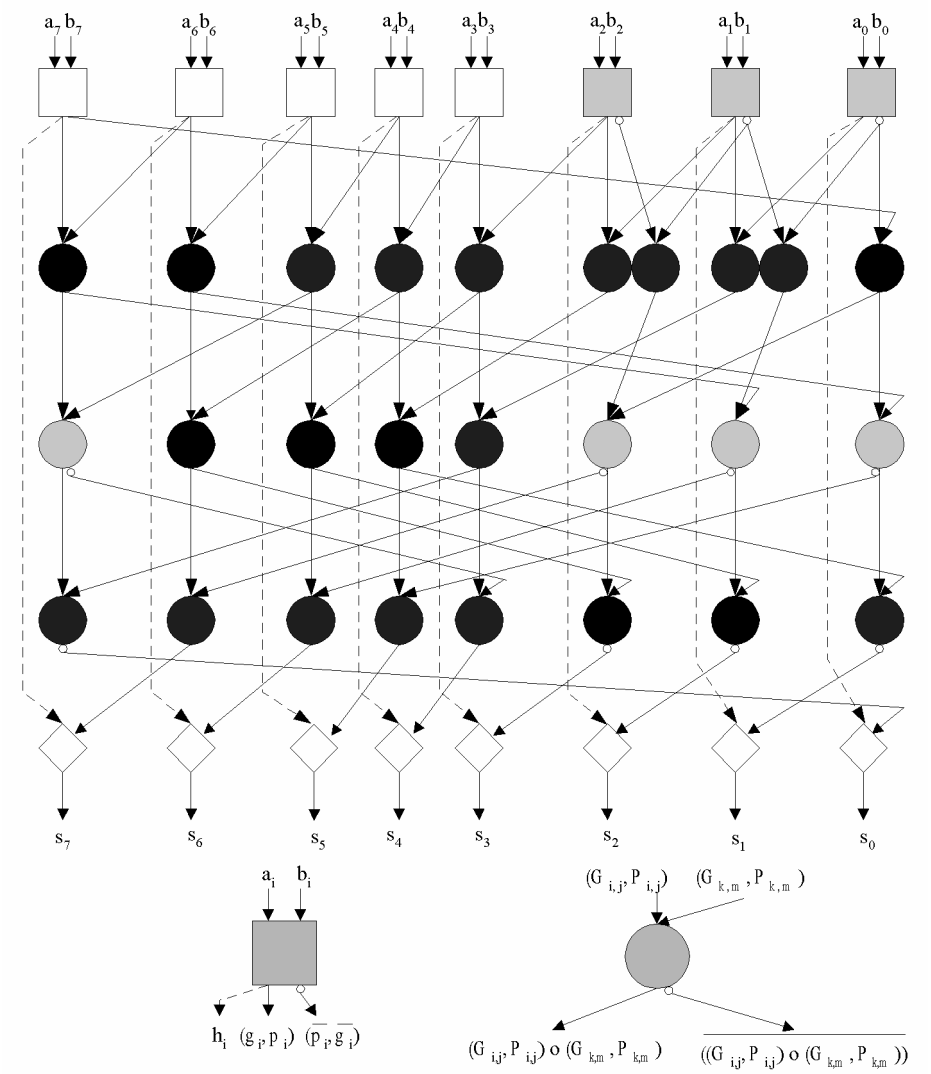

Fig. 8. Proposed parallel-prefix diminished-one modulo 257 adder.

4 Comparisons

In this section, we compare the proposed adder architectures against those proposed in [15], [16] both qualitatively and quantitatively. For our qualitative comparison, we will use a simple model, whereas, for the quantitative comparison, we will use actual static CMOS implementations. We use the notation LAF and KST for diminished-one modulo $2^n + 1$ adders with, respectively, a Ladner-Fischer and a Kogge-Stone prefix carry computation structure that follow the architecture proposed in [15], [16] and presented in Fig. 7. The proposed adders in Sections 3.1 and 3.2 will be denoted as CLA adders, whereas those proposed in Section 3.3 will be denoted as PPREF.

We will at first make use of the analytical model originally presented in [25], which was also used in [15], [16], [17], under the notation "unit-gate model," for comparing the proposed parallel-prefix adders against LAF and KST adders. This model assumes that each gate, excluding exclusive-OR, counts as one elementary gate for both area and delay. An exclusive-OR gate counts for two elementary gates for both area and delay. The model ignores fan-out, therefore, the validation of the estimates that it produces will be later carried out by CMOS static implementations. The estimates produced by the model are indicative in the case of parallel-prefix architectures since these are built solely by two-input gates. However, in CLA architectures, gates with multiple inputs may be required. That is why we do not include the proposed CLA architectures in our comparisons using the unit-gate model.

In Table 2, we present the area and delay estimates using this model as a function of the word length n. We have also included results for the fastest modulo 2^n and modulo $2^n - 1$ [17] adder. For the derivation of the area estimates, the ◯ buffering nodes in the carry computation prefix structure of Figs. 5 and 6 have not been taken into account.

Considering the delay, Table 2 not only reveals that the proposed adders are faster than both LAF and KST adders, but that they can operate as fast as the fastest known modulo 2^n and modulo $2^n - 1$ adders, which makes them ideal for use in an RNS application. From Table 2, we can see that, among the diminished-one modulo $2^n + 1$ adders, LAF adders require less area for their implementation. It should be noted, however, that the estimations produced by the adopted model do not take into account the area that may be required

TABLE 2
Adder Area and Delay Model Estimations

Adder	Area	Delay
Modulo 2^n	$\frac{3}{2}n\log n + 5n$	$2\log n + 3$
Fastest Modulo 2^n-1 [17]	$3n\log n + 5n$	$2\log n + 3$
LAF	$\frac{3}{2}n\log n + 8n - 1$	$2\log n + 5$
KST	$3n\log n + 7n$	$2\log n + 5$
PPREF	$\frac{9}{2}n\log n + \frac{1}{2}n + 6$	$2\log n + 3$

for buffer insertion needed to alleviate the unlimited fan-out problem of Ladner-Fischer prefix structures.

For more realistic evaluation, the proposed CLA and PPREF architectures and the LAF and KST adders were described in HDL for n = 4, 8, 16, and 32. For the CLA adders case, we modeled both the single and the two-level CLA adders, with various group sizes for the latter. For direct comparisons with the results presented in [17], we mapped our designs to the AMS CUB implementation technology ($0.6\mu m$, 2-metal layer, 5.0 V) using the Design Compiler® tool of Synopsys Inc. Each design was then recursively optimized for speed until the tool's algorithm was unable to provide a faster design. As a last stage of each recursive run, the tool was instructed to recover as much area as possible.

For the proposed CLA adders, we will present only the results for the fastest derived implementation. This fastest implementation when n = 4 is produced when a single level of CLA is used. However, for the rest of the examined cases (n = 8, 16, or 32), two-level CLA adders led to faster implementations. For the latter cases, we give, in parentheses, the number of subunits in the GPG and GCLA units that led to the minimum delay results. Table 3 lists the obtained results. All the delay results of Table 3 below assume worst case process parameters and are expressed in ns, whereas all area results are expressed in $mils^2$. Shaded cells indicate the best results in area or execution delay.

TABLE 3
Statis CMOS Implementations Area and Delay Results

n	Architecture	Area ($mils^2$)	Delay (ns)
4	LAF	66.7	2.70
	KST	87.2	2.69
	CLA	64.1	1.73
	PPREF	97.1	2.02
8	LAF	183.2	3.87
	KST	189.9	3.82
	CLA (4)	169.5	3.81
	PPREF	197.3	3.20
16	LAF	349.6	4.64
	KST	426.3	4.58
	CLA (4)	347.8	5.83
	PPREF	570.4	3.85
32	LAF	706.8	5.80
	KST	955.6	5.79
	CLA (8)	655.9	7.85
	PPREF	1276.6	4.70

Table 3 indicates that, for small values of n, the proposed CLA adder design methodology produces both faster and smaller implementations than LAF and KST. When n becomes large, however, LAF and KST adders are capable of leading to better performance with the penalty of increased area. PPREF adders are the fastest diminished-one modulo $2^n + 1$ adders when $n \geq 8$. Moreover, as n becomes larger, so does the performance difference between PPREF implementations against both LAF and KST adders. On the average of the examined cases, the proposed PPREF adders are approximately faster by 19 percent than LAF or KST adders. For reaching the fastest implementations, our proposed PPREF design methodology requires, on the average, 31 percent and 17 percent more implementation area over LAF and KST adders.

Comparing the results of Table 3 with those presented in Table III of [17], we can verify that the proposed PPREF adders, apart from being the fastest (excluding the case of very narrow operands in which the proposed CLA adders offer the best results) among those already proposed for diminished-one modulo $2^n + 1$ addition, they can also operate as fast as the fastest known integer (modulo 2^n) or modulo $2^n - 1$ adders. Therefore, they are highly applicable in RNS applications.

As we can see from Table 3, the proposed adders offer better delay results than LAF and KST adders, but, in some cases, with the penalty of increased implementation area. It is interesting to know whether, by applying the proposed architectures and targeting a performance equal to that of the faster among LAF and KST adders, the resulting adders can be implemented in less area. Therefore, in Table 4, we present results obtained by instructing the synthesis tool to find the

TABLE 4
Area-Time Constraint Driven Optimization Results

n	Architecture	Area	Delay
4	CLA	51.8	2.68
	PPREF	62.6	2.67
8	CLA (4)	169.5	3.81
	PPREF	153.63	3.80
16	PPREF	409.3	4.58
32	PPREF	1002.2	5.79

smaller implementation of the proposed adders that offers at least the performance of the fastest LAF or KST adders. Comparing the results of Table 3 and Table 4, we can see that, in all but one case, the proposed PPREF adders with smaller implementation area can lead to at least the same performance as LAF or KST adders. The proposed CLA architectures cannot lead to implementations as fast as the faster among LAF and KST adders in the n = 16 and n = 32 cases. That is why they do not appear in Table 4 in the corresponding rows.

5 Conclusions

In this paper, we have presented two novel architectures for designing diminished-one modulo $2^n + 1$ adders. The first architecture leads to carry look-ahead adder implementations and was derived by associating the reentering carry equation with those for producing the carries of the modulo addition. The second architecture leads to parallel-prefix adders and was derived by recirculating the carries in each level of the prefix structure. Static CMOS implementations have shown that the proposed modulo $2^n + 1$ adders compare favorably with the other already known adder architectures. Moreover, the proposed parallel-prefix modulo $2^n + 1$ adders are as fast as the fastest integer (modulo 2^n) and modulo $2^n - 1$ adders, that is, they are suitable for RNS applications.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments. An earlier version of a part of this work was presented at the 15th IEEE Symposium on Computer Arithmetic (ARITH – 15), 11-13 June 2001, Vail Colorado. This research was partially supported by the Research Committee of Patras University within the framework of K. Karatheodoris scholarships program.

References

[1] M.A. Sonderstrand et al., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing.* New York: IEEE Press, 1986.
[2] M.A. Bayoumi, G.A. Jullien, and W.C. Miller, "A Look-Up Table VLSI Design Methodology for RNS Structures Used in DSP Applications," *IEEE Trans. Circuits and Systems,* vol. 34, pp. 604-616, June 1987.
[3] K.M. Elleithy and M.A. Bayoumi, "Fast and Flexible Architectures for RNS Arithmetic Decoding," *IEEE Trans. Circuits and Systems-II: Analog and Digital Signal Processing,* vol. 39, pp. 226-235, Apr. 1992.
[4] I. Koren, *Computer Arithmetic Algorithms.* Prentice Hall, 1993.
[5] W.K. Jenkins and B.J. Leon, "The Use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters," *IEEE Trans. Circuits and Systems,* vol. 24, no. 4, pp. 191-201, Apr. 1977.
[6] V. Paliouras and T. Stouraitis, "Novel High-Radix Residue Number System Multipliers and Adders," *Proc. 1999 IEEE Int'l Symp. Circuits and Systems VLSI (ISCAS '99),* pp. 451-454, 1999.
[7] D.H. Lehmer, *Proc. Second Symp. Large-Scale Digital Calculating Machinery,* pp. 141-146, 1951.
[8] R. Zimmermann, A. Curiger, H. Bonnenberg, H. Kaeslin, N. Felber, and W. Fichtner, "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.
[9] A. Curiger, "VLSI Architectures for Computations in Finite Rings and Fields," PhD thesis, Swiss Federal Inst. of Technology (ETH), Zurich, 1993.
[10] X. Lai and J.L. Massey, "A Proposal for a New Block Encryption Standard," *Proc. EUROCRYPT '90,* May 1990.
[11] Y. Ma, "A Simplified Architecture for Modulo $(2^n + 1)$ Multiplication," *IEEE Trans. Computers,* vol. 47, no. 3, Mar. 1998.
[12] L.M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform," *IEEE Trans. Acoustics, Speech, and Signal Processing,* vol. 24, pp. 356-359, 1976.
[13] W.K. Jenkins, "The Design of Specialized Residue Classes for Efficient Recursive Digital Filter Realization," *IEEE Trans. Acoustics, Speech, and Signal Processing,* vol. 30, pp. 370-380, 1982.
[14] W.K. Jenkins, "Recent Advance in Residue Number Techniques for Recursive Digital Filtering," *IEEE Trans. Acoustics, Speech, and Signal Processing,* vol. 27, pp. 19-30, 1979.
[15] R. Zimmermann, "Efficient VLSI Implementation of Modulo $(2^n \pm 1)$ Addition and Multiplication," *Proc. 14th IEEE Symp. Computer Arithmetic,* pp. 158-167, Apr. 1999.
[16] R. Zimmermann, "Binary Adder Architectures for Cell-Based VLSI and Their Synthesis," PhD thesis, Swiss Federal Inst. of Technology, Zurich, 1997.
[17] L. Kalamboukas, D. Nikolos, C. Efstathiou, H.T. Vergos, and J. Kalamatianos, "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Trans. Computers,* vol. 49, no. 7, special issue on computer arithmetic, pp. 673-680, July 2000.
[18] K. Hwang, *Computer Arithmetic: Principles, Architecture and Design.* John Wiley & Sons, 1979.
[19] C. Efstathiou, D. Nikolos, and J. Kalamatianos, "Area-Time Efficient Modulo $2^n - 1$ Adder Design," *IEEE Trans. Circuits and Systems-II,* vol. 41, no. 7, pp. 463-467, 1994.
[20] R.E. Ladner and M.J. Fischer, "Parallel Prefix Computation," *J. ACM,* vol. 27, no. 4, pp. 831-838, Oct. 1980.
[21] R.P. Brent and H.T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. Computers,* vol. 31, no. 3, pp. 260-264, Mar. 1982.
[22] P.M. Kogge and H.S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Trans. Computers,* vol. 22, no. 8, pp. 783-791, Aug. 1973.
[23] S. Knowles, "A Family of Adders," *Proc. 15th IEEE Symp. Computer Arithmetic,* pp. 277-281, Apr. 2001.
[24] J.A. Abraham and D.D. Gajski, "Easily Testable High-Speed Realization of Register-Transfer-Level Operations," *Proc. 10th Fault-Tolerant Computing Symp. (FTCS-10),* pp. 339-344, Oct. 1980.
[25] A. Tyagi, "A Reduced-Area Scheme for Carry-Select Adders," *IEEE Trans. Computers,* vol. 42, no. 10, pp. 1163-1170, Oct. 1993.

Haridimos T. Vergos received the Diploma and PhD degrees in computer engineering from the University of Patras in 1991 and 1996, respectively. During 1998, he worked on the development of the first IEEE 802.11 MAC processor for the Multimedia and Networking Group of Atmel Inc. He currently holds a lecturer position in the Department of Computer Engineering and Informatics at the University of Patras, Greece. Dr. Vergos has authored more than 30 scientific papers on computer arithmetic and architecture and on design for testability. He is a member of the Greek Computer Engineering Society and the Technical Chamber of Greece.

Costas Efstathiou received the BS degree in physics and the MS degree in electronics from the University of Athens and the PhD degree in computer science from the University of Thessaloniki. He is currently a professor in the Department of Informatics of the TEI of Athens, Greece. Previously, he worked in the Research Institute of the Hellenic Air Force and the Hellenic Telecommunications Organization. His research and teaching interests are in digital design and computer architecture with emphasis in computer arithmetic and fault-tolerant systems and in computer networks and telecommunications.

Dimitris Nikolos received the BSc degree in physics, the MSc degree in electronics, and the PhD degree in computer science, all from the University of Athens. He is currently a professor in the Computer Engineering and Informatics Department of the University of Patras, director of the Hardware and Computer Architecture Division, and head of the Technology and Computer Architecture Laboratory. He has served as program cochairman of five (1997-2001) IEEE International On-Line Testing Workshops. He also served on the program committees for the IEEE International Symposiums on Defect and Fault Tolerance in VLSI Systems (1997-1999), for the Third and Fourth European Dependable Computing Conference, and for the DATE (2000-2002) Conferences. He was guest coeditor of the June 2002 special issue of the *Journal of Electronic Testing, Theory, and Applications* (JETTA) devoted to the 2001 IEEE International On-Line Testing Workshop. His main research interests are fault-tolerant computing, computer architecture, VLSI design, test, and design for testability. Professor Nikolos has authored or coauthored more than 110 scientific papers and was corecipient of the Best Paper Award for his work "Extending the Viability of I_{DDQ} Testing in the Deep Submicron Era" presented at the Third IEEE International Symposium on Quality Electronic Design (ISQED 2002). He is a member of the IEEE and the IEEE Computer Society.