# A Family of Parallel-Prefix Modulo $2^n - 1$ Adders

G. Dimitrakopoulos[†] , H. T. Vergos[†‡], D. Nikolos[†‡], and C. Efstathiou[§]

[†]Computer Engineering and Informatics Dept., University of Patras, 26500 Patras, Greece

[‡]Computer Technology Institute, 3 Kolokotroni Str., 26221 Patras, Greece

[§]Informatics Dept., TEI of Athens, 12210 Egaleo, Athens, Greece.

**Abstract**

*In this paper we at first reveal the cyclic nature of idempotency in the case of modulo $2^n - 1$ addition. Then based on this property, we derive for each $n$, a family of minimum logic depth modulo $2^n - 1$ adders, which allows several trade-offs between the number of operators, the internal wire length, and the fanout of internal nodes. Performance data, gathered using static CMOS implementations, reveal that the proposed architectures outperform all previously reported ones in terms of area and/or operation speed.*

## 1 Introduction

Modulo $2^n - 1$, or equivalently one's complement addition, plays an essential role in Residue Number System (RNS) applications [1], in fault-tolerant computer systems [2], in error detection in computer networks [3], and in floating-point arithmetic [4], [5].

In RNS each operand is encoded as a vector of residues, computed with respect to a set of $M$ pairwise relatively prime integers called the *moduli*. The later form a set $\mathcal{W} = \{m_1, m_2, \ldots, m_M\}$, which is called the *base* of the RNS. All integers $A, B$ with $0 \leq A, B < \prod_{i=1}^{M} m_i$ have a unique RNS representation $A \overset{RNS}{\longleftrightarrow} \{A_1, A_2, \ldots, A_M\}$ and $B \overset{RNS}{\longleftrightarrow} \{B_1, B_2, \ldots, B_M\}$, where $A_i = \langle A \rangle_{m_i}$, $B_i = \langle B \rangle_{m_i}$ for $i = 1, 2, \ldots, M$, and $\langle x \rangle_{m_i}$ denotes the residue of $x$ modulo $m_i$. Multiplication, addition, and subtraction in RNS are described by $Z = A \diamond B \overset{RNS}{\longleftrightarrow} \{Z_1, Z_2, \ldots, Z_M\}$, where $Z_i = \langle A_i \diamond B_i \rangle_{m_i}$ and $\diamond$ denotes any of the aforementioned operations. Significant speedup over the corresponding binary operations can be achieved, since the $Z_i$s are computed in parallel, each in a separate arithmetic unit (channel), because their computation depends only on $A_i$, $B_i$, and $m_i$, and no carry propagation among the channels is required. Among the most popular three-moduli bases are the $\{2^n, 2^n - 1, 2^n + 1\}$ and the $\{2^n, 2^n - 1, 2^{n-1} - 1\}$ [6]–[8]. Therefore, a modulo $2^n - 1$ adder is essential in the most popular RNS implementations.

Modulo $2^n - 1$ adders also find great applicability in fault-tolerant computer systems. They are commonly used for implementing residue, inverse residue, product (AN) and checksum arithmetic codes. For low-cost implementations of such codes, modulo $2^n - 1$ adders are used both for the encoding and the checking process [2]. Such codes, are also used extensively in checksum computation, and error detection in TCP/IP networks [3].

Recently, one's complement addition has also been employed in the design of high-speed floating-point arithmetic units [4], [5]. In [4] an 161-bit end-around carry (EAC) adder was used in the design of a single pass floating-point multiplier, while in the dual pass version of the design, it

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

was substituted by an 81-bit EAC adder. Additionally, Pillai *et al.* in [5] have presented a triple-datapath architecture for floating point addition, which employs 1's complement adders, and offers significant savings in the power dissipation.

Several proposals have already appeared to the fast modulo $2^n - 1$ adder design problem. Single and two-level carry lookahead modulo $2^n - 1$ adders have been presented in [9]. To achieve even higher speeds, parallel-prefix carry-computation design approaches have been adopted in [10]–[14]. In [10] and [12] the required end-around carry operation is achieved by feeding back the carry output of a parallel-prefix integer addition unit via an extra prefix level. This technique apart from adding an extra prefix level also suffers from the unlimited fanout loading problem at the re-entering carry line. In [11] it was shown that modulo $2^n - 1$ addition can be performed by recirculating the generate and propagate signals, instead of the traditional end-around carry approach. In this way, the need for an additional prefix level is cancelled, and parallel-prefix modulo $2^n - 1$ adders with minimum logic depth (equal to the fastest integer parallel prefix adders) are derived. Although the fundamental theory and a general architecture were presented in [11], no straightforward design method was given when $n \neq 2^k$. This task was left to the intuition of the designer. Extending the work of [11], in [14] a method was given to produce Kogge-Stone like modulo $2^n - 1$ adders for every $n$. Finally in [13], parallel-prefix adders, similar to those of [11], using prefix operators of valency greater or equal to 2 were presented, only for the case that $n$ is of the $2^k$ form.

In this paper a novel carry-computation architecture for parallel-prefix modulo $2^n - 1$ adders, for arbitrary values of $n$, is introduced. The proposed architecture actually describes a whole family of adders, which exhibits minimum logic depth and small operator count. At first the basic theory of idempotency is extended for the case of modulo $2^n - 1$ addition and it is shown that terms produced in a parallel-prefix tree can be further associated in a circular manner. Then, a systematic methodology for designing a family of modulo $2^n - 1$ adders is presented. Static CMOS implementations are finally utilized for real comparisons of the proposed structures against previously reported parallel-prefix modulo $2^n - 1$ adders. Experimental results indicate that several members of the proposed family of modulo $2^n - 1$ adders can significantly reduce the area penalty of previously reported designs [14], while maintaining the highest speed compared to [10].

The rest of the paper is organized as follows. Some background material on parallel-prefix addition and the notation used are given in Section 2. The extension of the idempotency property is introduced in Section 3, while the family of modulo $2^n - 1$ adders is presented in Section 4. Quantitative results are presented in Section 5. Finally, conclusions are drawn in Section 6.

## 2 Background and Definitions

Suppose that $A = a_{n-1}a_{n-2}\ldots a_0$ and $B = b_{n-1}b_{n-2}\ldots b_0$ represent the two numbers to be added and $S = s_{n-1}s_{n-2}\ldots s_0$ 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 $i$, $0 \leq i \leq n-1$, 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 the logical AND, OR and exclusive-OR operations respectively. The second stage of the adder, hereafter called the carry-computation unit, computes the carry signals $c_i$, for $0 \leq i \leq n-1$ using the carry generate and carry propagate bits $g_i$ and $p_i$. The third stage computes the sum bits according to:

$$s_i = h_i \oplus c_{i-1}.$$

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

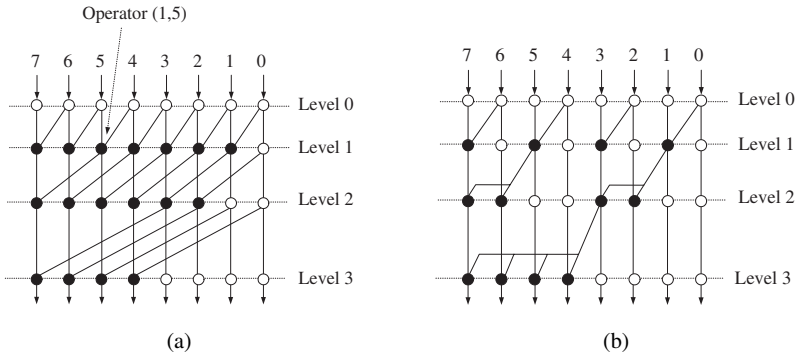


**Figure 1. The (a) Kogge-Stone and (b) Ladner-Fischer Parallel-Prefix Structures.**

Since in all adders the first and the third stage are identical, in the following we concentrate on the carry-computation unit. Several algorithms have been proposed for the carry computation problem. Carry computation is transformed into a prefix problem using the operator $\circ$, which associates pairs of generate and propagate signals and was defined in [15] as follows,

$$(g, p) \circ (g', p') = (g + p \cdot g', p \cdot p'). \tag{1}$$

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

$$(G_{k:j}, P_{k:j}) = (g_k, p_k) \circ (g_{k-1}, p_{k-1}) \circ \ldots \circ (g_{j+1}, p_{j+1}) \circ (g_j, p_j). \tag{2}$$

Although the $\circ$ operator is not commutative, the idempotency property [16],

$$(G_{i:\,j}, P_{i:j}) = (G_{i:\,k}, P_{i:k}) \circ (G_{m:\,j}, P_{m:j}) \tag{3}$$

holds for it, where $i > m \geq k > j$.

We define as *length* of a group generate/propagate term (or simply length), the number of distinct generate/propagate pairs $(g_k, p_k)$ that have been associated for its computation. The length of the group generate/propagate term $(G_{k:j}, P_{k:j})$ is obviously $k - j + 1$, $k > j$. When two group signals are further associated the result will have a length equal to the sum of the lengths of the two operands minus any *overlapping* terms due to idempotency.

The parallel-prefix structures proposed by Kogge-Stone [17] and Ladner-Fisher [18] for an 8-bit carry-computation unit are shown in Figure 1. The operator $\circ$ is represented as a node ($\bullet$), while white nodes are buffering nodes. In any parallel-prefix graph we will number the prefix levels from 0 (the $(g, p)$ signals-pair level) up to $m$ (the level that produces the carries) and the bit columns from 0 up to $n - 1$. Since the $\bullet$ nodes are placed on the grid of rows and columns we can refer to any of them by the pair (prefix level, bit column). For example in Figure 1(a) the operator $(1, 5)$ is pointed. The prefix structures proposed by Kogge-Stone [17], Ladner-Fischer [18] and Knowles [19] are of special practical interest, since they offer minimum logic-depth solutions to the carry-computation problem.

## 3 The Case of Modulo $2^n - 1$ Addition

In this section the basic theory introduced in [11] is revisited, and a novel property of idempotency is introduced. According to [11] in the case of modulo $2^n - 1$ addition, each carry $c_i$,

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

$0 \leq i \leq n-1$, is produced by combining the carry generate and propagate pairs using the formula,

$$G_i = (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \ldots \circ (g_0, p_0) \circ (g_{n-1}, p_{n-1}) \circ \ldots \circ (g_{i+1}, p_{i+1}) \tag{4}$$

where, $c_i = G_i$ and $c_{-1} = c_{n-1}$. It should be noted that in contrast to integer addition, the number of pairs $(g_k, p_k)$ that have to be associated for the generation of each carry is equal to $n$.

Due to (4) the definition of a group generate/propagate term $(G_{k:j}, P_{k:j})$ is extended here to the case where $k < j$, $0 \leq k, j \leq n-1$, and is defined as,

$$(G_{k:j}, P_{k:j}) = (G_{k:0}, P_{k:0}) \circ (G_{n-1:j}, P_{n-1:j}). \tag{5}$$

Therefore, in the general case ($k > j$ or $k < j$), the length of a group generate/propagate term $(G_{k:j}, P_{k:j})$ is equal to $\langle k - j + 1 \rangle_n$. Assuming an intermediate index $k$, $0 \leq k \leq n-1$, each carry $c_i$ of (4) can be expressed as,

$$G_i = (G_{i:k}, P_{i:k}) \circ (G_{k-1:i+1}, P_{k-1:i+1}). \tag{6}$$

The following Theorem reveals the cyclic nature of the idempotency property in the case of modulo $2^n - 1$ addition, and is used as the basis for the design of the family of adders proposed in this paper.

**Theorem 1.** *Let*

$$(G_{i:k}, P_{i:k}) = \begin{cases} (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \ldots \circ (g_k, p_k), & \text{if } i \geq k \\ (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \ldots \circ (g_0, p_0) \circ (g_{n-1}, p_{n-1}) \circ \ldots \circ (g_k, p_k), & \text{if } i < k \end{cases}.$$

*Then it holds that*

$$(G_{i:k}, P_{i:k}) \circ (G_{k-1:i+1}, P_{k-1:i+1}) \circ (G_{i:k}, P_{i:k}) = (G_{i:k}, P_{i:k}) \circ (G_{k-1:i+1}, P_{k-1:i+1}).$$

*Proof.* Unrolling the prefix operator $\circ$ it follows that,

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

□

Theorem 1 simplifies group generate/propagate terms of length greater than $n$ to terms of length equal to $n$. For example assume the case of a modulo $2^5 - 1$ adder. The association of $(G_{4:1}, P_{4:1})$ (length 4 term) with $(G_{1:3}, P_{1:3})$ (length 4 term) is expected to lead to a group term of size 7, since under the normal definition of idempotency only the overlapping term $(g_1, p_1)$ can be simplified. However, due to Theorem 1 the resulting term is $(G_{4:0}, P_{4:0})$, which is a length-5 term, since,

$$\begin{aligned} (G_{4:1}, P_{4:1}) \circ (G_{1:3}, P_{1:3}) &= (G_{4:3}, P_{4:3}) \circ (G_{2:1}, P_{2:1}) \circ (G_{1:0}, P_{1:0}) \circ (G_{4:3}, P_{4:3}) \\ &= (G_{4:3}, P_{4:3}) \circ (G_{2:0}, P_{2:0}) \circ (G_{4:3}, P_{4:3}) \\ &= (G_{4:3}, P_{4:3}) \circ (G_{2:0}, P_{2:0}) = (G_{4:0}, P_{4:0}). \end{aligned}$$

Theorem 1 is an extension of the basic idempotency property presented in [16]. The assumption that two group generate/propagate terms must meet or overlap in order to be associated can be also considered in a circular manner. Figure 2 explains the circular meet-or-overlap for the case of $(G_{4:1}, P_{4:1})$ and $(G_{1:3}, P_{1:3})$.

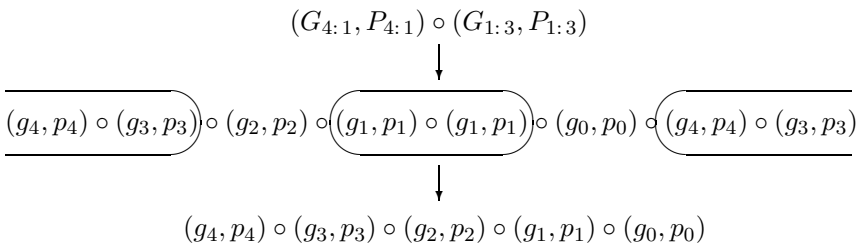


**Figure 2. Traditional and Circular Idempotency.**

## 4 The Proposed Design Methodology

A systematic methodology for designing a family of area-time efficient parallel-prefix modulo $2^n - 1$ adders is introduced in this section. All derived family members, i.e., prefix structures, have minimum logic depth equal to $m = \lceil \log_2 n \rceil$ prefix levels, and the number of operators employed for carry generation can vary according to the value of $n$ and the design selected in each case.

According to Eq. (4) the length of all carry equations in a modulo $2^n - 1$ adder is equal to $n$. Therefore, among the possible lengths of group generate/propagate terms that can be generated in *at most* $m - 1$ prefix levels the ones that allows us to build, at the $m$th level, group terms of length greater than $n$, due to Theorem 1, or equal to $n$ are sought. In any other case the generation of a valid carry relation is impossible. Let $S_a$ and $S_b$ represent the length of any two group terms, which are generated in the first $m - 1$ prefix levels, and are selected to complete carry generation in the $m$th level. Then, the selected $S_a$ and $S_b$ should satisfy the following condition,

$$S_a + S_b \geq n. \tag{7}$$

The way group generate/propagate terms can be produced in the first $m - 1$ prefix levels of the carry-computation unit can be graphically represented via a graph, called the Length Dependency Graph. The Length Dependency Graph is the same for all values of $n$ with the same logic depth $m = \lceil \log_2 n \rceil$, and is denoted as $\text{LDG}_m$. The $\text{LDG}_m$ consists of $m$ levels and contains one vertex for each possible length of the group terms that can be produced in the first $m - 1$ levels of the carry-computation unit. The value inside each vertex is equal to the length that it represents. For example $\text{LDG}_4$ is drawn in Figure 3(a). At level 0 only one vertex exists with value equal to 1, which represents the length-1 terms, i.e., the generate/propagate pairs $(g, p)$.

The edges of $\text{LDG}_m$ describe the way that each possible length of group terms can be produced. For example, the edge $(4) \rightarrow (6)$ with weights $\{2, 3, 4\}$ implies that a length-6 group term can be produced on the 3rd prefix level by associating a length-4 term of the second level with a suitable term of length either 2 or 3 or 4. The associations of terms of length 4 and 3, and 4 and 4, require the use of idempotency in order to produce a length-6 term.

Therefore, for each pair of lengths $\{S_a, S_b\}$ that satisfies condition (7), and by following the connections of $\text{LDG}_m$ a parallel-prefix carry-computation unit for a modulo $2^n - 1$ adder can be constructed. To simplify the design procedure, for each selected pair $\{S_a, S_b\}$ the Design Graph $\text{DG}_{n,\{S_a,S_b\}}$ is extracted from the corresponding $\text{LDG}_m$. The $\text{DG}_{n,\{S_a,S_b\}}$ is a subgraph of $\text{LDG}_m$, and it is derived by following the paths of the $\text{LDG}_m$ that depart from the vertices with values $S_a$ and $S_b$, respectively, up to level 0. The vertices that do not belong to any of these paths are excluded.

For example the $\text{DG}_{10,\{8,5\}}$ in case of modulo $2^{10} - 1$ addition is derived from $\text{LDG}_4$ of Figure 3(a), and is presented in Figure 3(b). Since the pair $\{8, 5\}$ satisfies condition (7) for the

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

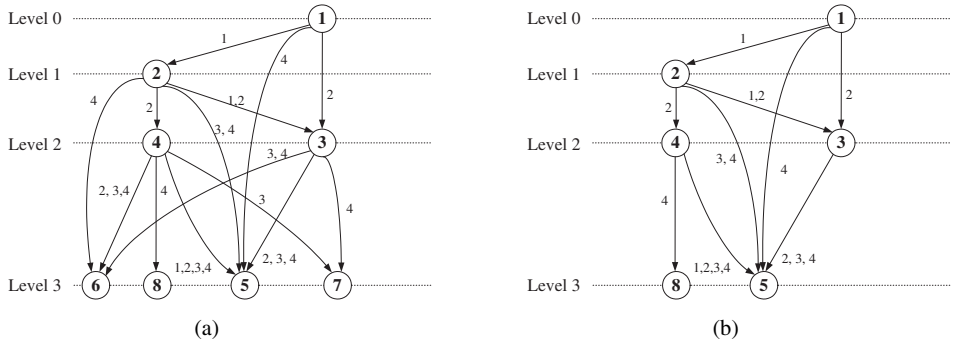


**Figure 3. (a) The Length Dependency Graph for $4$ prefix levels implementations $LDG_4$ and (b) the corresponding Design Graph $DG_{10,\{8,5\}}$.**

case of modulo $2^{10}-1$ addition, it guarantees that the carries $c_i$ can be generated in the $4$th prefix level. As shown by Figure 3(b), the construction of group terms of length $8$ and $5$ in the 3rd prefix level leaves many choices to the designer, especially for the length-5 terms. It should be noted that the generation of carries the carries $c_i$ in the $4$th prefix level requires the existence of group terms with all possible prefixes $(g_i, p_i)$, $i = 0, 1, \ldots, 9$, produced either from group terms of length $8$ or terms of length $5$, respectively.

In general, even after the selection of a valid pair $\{S_a, S_b\}$ the design space is left with numerous solutions. Based on LDG$_m$ we can produce exhaustively all possible solutions of modulo $2^n-1$ adders and select the one that best matches our design constraints. Since the design-solutions space is huge we set certain rules that allow only a subset of all possible solutions to be derived.

### 4.1 Reduction Rules

The proposed systematic design procedure is based on treating the even-indexed $\{0, 2, 4, \ldots\}$ and the odd-indexed $\{1, 3, 5, \ldots\}$ bit columns of the prefix tree separately. Specifically, all group generate/propagate terms produced on the even-indexed columns of the $i$th prefix level have the same length denoted as $L_{\text{even}}(i)$. Additionally, all group terms generated on the odd-indexed columns of the $i$th prefix level are also of equal length, and their length is denoted as $L_{\text{odd}}(i)$. On the last, i.e., $m$th, prefix level, the group terms from the even and the odd-indexed columns are properly associated, in order the carries of the modulo $2^n-1$ addition, according to Eq. (4), to be produced.

The reduction rules concern the length of the generate/propagate terms that can be produced on the even or the odd-indexed columns, and are applied in all the prefix levels up to the $(m-1)$st level. The input connections to the operators of the $m$th level are treated separately.

REDUCTION RULES FOR THE EVEN-INDEXED BIT COLUMNS

E1. On the $\lceil \frac{n}{2} \rceil$ even-indexed columns only group terms of even length are produced.

E2. The even-length group terms of the $i$th prefix level are produced by associating group terms of length $2^{i-1}$ of the $(i-1)$st prefix level, possibly by using idempotency. This rule implies that the operators placed at the even-indexed columns of the $i$th level associate only terms of lenght $2^{i-1}$, steming from the even-indexed columns of the previous level. For example the generation of a length-6 group term on the 3rd prefix level imposes the association of two group terms of length $4$ that have been generated on the 2nd level.

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

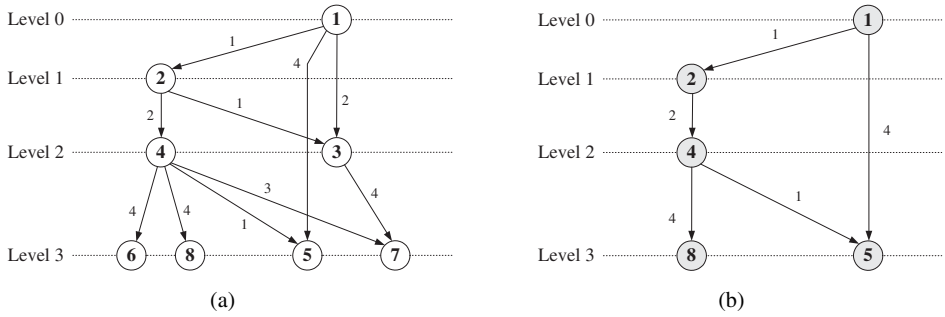


**Figure 4. (a) The Simplified LDG for $4$ prefix levels implementations $SLDG_4$ and (b) the corresponding simplified design graph $SDG_{10,\{8,5\}}$.**

REDUCTION RULES FOR THE ODD-INDEXED BIT COLUMNS

O1. On the $\left\lfloor \frac{n}{2} \right\rfloor$ odd-indexed columns only group terms of odd length are produced.

O2. The odd-length, $L_{\text{odd}}(i)$, group terms of the $i$th prefix level are generated by associating a group term of even length $2^{i-1}$ of the $(i-1)$st level and a term of length $L_{\text{odd}}(i) - 2^{i-1}$ generated on the $k$th level, with $k = \left\lceil \log_2 \left( L_{\text{odd}}(i) - 2^{i-1} \right) \right\rceil$, $k < i$. This rule implies that the operators placed on the odd-indexed columns of the $i$th prefix level associate only terms of length $2^{i-1}$ that appear on the even-indexed columns of the $(i-1)$st level and terms of odd lenght, i.e., $L_{\text{odd}}(i) - 2^{i-1}$, generated on the odd-indexed columns of any previous level.

Design rules E.2 and O.2 determine the exact way each term, of even or odd length, will be generated in the prefix tree. They are applied in a bottom-up fashion beginning from the $(m-1)$st level up to the first level, in order to predetermine the length of all intermediate group terms that need to be produced.

The separate treatment of the odd the even-indexed columns, along with the introduced design rules, specify a subset of all possible solutions that can be derived by $\text{LDG}_m$. Applying the reduction rules to $\text{LDG}_m$ we produce a simplified length dependency graph denoted as $\text{SLDG}_m$. The $\text{SLDG}_4$ is shown in Figure 4(a). The vertices of $\text{SLDG}_m$ are separated in two sets, namely $V_{\text{even}}$ (vertices with even values) and $V_{\text{odd}}$ (vertices with odd values), which correspond to the even and the odd-length group terms that can be produced by the parallel-prefix carry-computation unit.

Similar to $S_a$ and $S_b$, we define $S_{\text{even}} \in L_{\text{even}}(i)$ and $S_{\text{odd}} \in L_{\text{odd}}(i)$, $1 \leq i \leq m-1$, as the length of the group terms that are selected from the even and the odd-indexed columns, respectively, to complete carry generation, and $d_n$ to be defined as,

$$d_n = \begin{cases} n+1, & \text{if } n \text{ is odd} \\ n, & \text{if } n \text{ is even} \end{cases}. \tag{8}$$

Following relation (7) the selected $S_{\text{even}}$ and $S_{\text{odd}}$ should satisfy the following condition,

$$S_{\text{even}} + S_{\text{odd}} \geq d_n. \tag{9}$$

The variable $d_n$ is used, since a more strict bound than condition (7) is required by the proposed methodology when $n$ is odd. Therefore for each pair of even or odd lengths $\{S_{\text{even}}, S_{\text{odd}}\}$ that satisfy condition (9) a simplified design graph ($\text{SDG}_{n,\{S_{\text{even}},S_{\text{odd}}\}}$) can be derived from the $\text{SLDG}_m$. The $\text{SDG}_{10,\{8,5\}}$ extracted from $\text{SLDG}_4$ is shown in Figure 4(b). The $\text{SDG}_{10,\{8,5\}}$ allows the design of a modulo $2^{10}-1$ adder in a straightforward manner, and it is less complex than $\text{DG}_{10,\{8,5\}}$ of Figure 3(b), since several solutions are omitted due to the adoption of the reduction rules.

### 4.2 Design Procedure

After the derivation of $\text{SDG}_{n,\{S_{\text{even}},S_{\text{odd}}\}}$, the proposed design procedure is described by the following steps, including the connections of the $m$th prefix level, which completes generation of all carries $c_i$ according to Eq. (4).

**Step 1**: *Definitions*

Set $m = \lceil \log_2 n \rceil$, $\beta = \min\{S_{\text{even}}, S_{\text{odd}}\}$, and $\gamma = S_{\text{even}} + S_{\text{odd}}$. $L(u)$ denotes the value of vertex $u$ in the $\text{SDG}_{n,\{S_{\text{even}},S_{\text{odd}}\}}$.

**Step 2**: *First prefix level*

Place $\lceil \frac{n}{2} \rceil$ operators on the even-indexed columns of the first prefix level. Each operator $(1, j)$ with $j \in \{0, 2, \ldots, d_n - 2\}$ connects to the nodes $(0, j)$ and $(0, \langle j - 1 \rangle_{d_n})$. Add buffering nodes to the odd-indexed columns of the first prefix level.

**Step 3**: *Subsequent $m - 2$ prefix levels*

Examine the $i$th level of the $\text{SDG}_{n,\{S_{\text{even}},S_{\text{odd}}\}}$.

- If a vertex $u \in V_{\text{even}}$ exists, then place $\lceil \frac{n}{2} \rceil$ operators on the even-indexed columns of the $i$th prefix level. Each operator $(i, j)$, with $j \in \{0, 2, \ldots, d_n - 2\}$ connects to the operators $(i - 1, j)$ and $(i - 1, \langle j - L(u) + 2^{i-1} \rangle_{d_n})$.
- If a vertex $u \in V_{\text{odd}}$ exists, then place $\lfloor \frac{n}{2} \rfloor$ operators on the odd-indexed columns of the $i$th prefix level. Each operator $(i, j)$ with $j \in \{1, 3, \ldots, d_n - 3\}$ connects to the operators $(i - 1, j)$ and $(i - 1, \langle j - L(u) + 2^{i-1} \rangle_{d_n})$. Additionally if $n$ is even add the operator $(i, d_n - 1)$ and connect it to $(i - 1, d_n - 1)$ and $(i - 1, \langle 2^{i-1} - L(u) - 1 \rangle_{d_n})$, respectively.

Add buffering nodes to the remaining either even or odd columns of the $i$th prefix level.

**Step 4**: *Connections on the last prefix level*

Construct the $m$th prefix level consisting of $n$ operators.

- Each operator $(m, j)$ with $j \in \{0, 2, \ldots, d_n - 2\}$ connects to $(m - 1, j)$ and $(m - 1, \langle j - \beta + \gamma \rangle_{d_n})$.
- Each operator $(m, j)$ with $j \in \{1, 3, \ldots, d_n - 3\}$ connects to $(m - 1, j)$ and $(m-1, \langle j-1-\beta+\gamma \rangle_{d_n})$. Additionally if $n$ is even add the operator $(m, d_n - 1)$ and connect it to $(m - 1, d_n - 1)$ and $(m - 1, \langle \gamma - \beta - 2 \rangle_{d_n})$, respectively.

The family of parallel-prefix modulo $2^{10} - 1$ adders, designed according to the proposed design methodology, are shown in Figure 5, along with the corresponding simplified design graphs derived from $\text{SLDG}_4$. It can be verified that each solution has its own internal wire length and fan-out loading, while the number of operators, i.e., nodes $\bullet$, used in each case range from 30 to 35. The carry-computation units with the minimum number of operators in general have less-complex wiring and less nodes with increased fanout compared to the solutions with more operators. The same observations can be made for all carry-computation units that employ the minimum number of operators.

**Table 1. Area$(\mu m^2)$ and Time$(ns)$ Results using Static CMOS implementations.**

(a)

| $n$ | [10] LF | | [10] KS | | [14] | | Proposed | | |
|---|---|---|---|---|---|---|---|---|---|
| | Area | Time | Area | Time | Area | Time | $SDG_n$ | Area | Time |
| 5 | 3981 | 1.07 | 5559 | 1.07 | 5307 | 0.85 | $SDG_{5,\{4,3\}}$ | 5307 | 0.85 |
| 6 | 4464 | 1.12 | 6669 | 1.08 | 6523 | 0.84 | $SDG_{6,\{4,3\}}$ | 6523 | 0.84 |
| 9 | 7443 | 1.29 | 15668 | 1.24 | 12534 | 1.07 | $SDG_{9,\{6,5\}}$ | 8654 | 1.17 |
| 20 | 16985 | 1.64 | 26569 | 1.60 | 26943 | 1.31 | $SDG_{20,\{16,5\}}$ | 20250 | 1.36 |
| 24 | 18805 | 1.66 | 35099 | 1.56 | 30378 | 1.34 | $SDG_{24,\{16,9\}}$ | 24156 | 1.44 |
| 56 | 48382 | 1.91 | 77500 | 1.77 | 93289 | 1.56 | $SDG_{56,\{32,25\}}$ | 58081 | 1.64 |

(b)

| $n$ | Time | Area | | |
|---|---|---|---|---|
| | | [10] LF | [14] | Proposed |
| 5 | 1.07 | 3981 | 3013 | 3013 |
| 6 | 1.12 | 4464 | 3847 | 3847 |
| 9 | 1.29 | 7443 | 7088 | 5958 |
| 20 | 1.64 | 16985 | 16858 | 13738 |
| 24 | 1.66 | 18805 | 19168 | 14797 |
| 56 | 1.91 | 48382 | 59896 | 43675 |

## 5 Performance Evaluation

The proposed adders were compared against the modulo $2^n - 1$ adders proposed in [10] when either a Ladner-Fischer [18](LF) or a Kogge-Stone [17](KS) prefix tree is used, as well as, against the reduced modulo $2^n - 1$ adders proposed in [14]. Each adder was described in Verilog HDL and mapped on the UMC-VST 25 technology library ($0.25\mu m$, $1.8/3.3$V, up to 5 metal layers) using the Synopsys® Design Compiler. Each design was optimized for speed targeting a strict maximum delay of 0.8ns for $n = 5, 6, 9$ and 1.2ns for $n = 20, 24, 56$. The obtained results are shown in Table 1(a).

Since the proposed adders do not suffer from the problem of the high fanout loading at the last stage and need one prefix level less than the adders proposed in [10], they are faster than them, regardless of which prefix structure, LF or KS, is used. On the average of the examined cases, the proposed adders are faster than those of [10] that use a LF or a KS prefix tree by $16\%$ and $13\%$, respectively. Considering the implementation area, the proposed adders, although faster in all examined cases, require significantly less area than the faster adders of [10], the ones with a KS prefix tree. On the average of the examined cases the area savings offered is $22\%$. The implementation area of the proposed adders is larger than that of [10] with a LF prefix tree by an average of $21\%$.

The results of Table 1(a) also reveal that the proposed adders are slightly slower than the adders proposed in [14]. This was expected since both architectures require the same prefix levels and the fanout loading is bounded. It should be noted that the Kogge-Stone-like modulo $2^n - 1$ adders proposed in [14] lead to the same prefix trees as the ones proposed in this paper when $n = 5, 6$. However, the proposed adders require significantly less prefix operators and hence implementation area for larger values of $n$. For example, in the case of $n = 56$, 84 less operators are required, which leads to an area reduction of $37\%$. On the average of the examined cases the area savings

offered by the proposed adders over the adders of [14] is $18.5\%$.

We also synthesized the proposed adders targeting a delay equal to the delay of the most area-efficient architecture, as derived from Table 1(a). The obtained results are shown in Table 1(b). It can be easily verified that the proposed architectures require less implementation area in all cases. The area reductions achieved are in average $13.6\%$ and $17.5\%$, when compared to the adders of [14] and of [10] with a LF prefix tree, respectively.

## 6 Conclusions

Fast and compact modulo $2^n - 1$ adders are greatly appreciated in RNS implementations, computer networks and fault-tolerant computer systems. In this paper, based on an extension of the idempotency property, we have introduced a new systematic design methodology, which leads to a family of parallel-prefix modulo $2^n - 1$ adders. All members of each family share the minimum logic depth property, whereas each member, has its own operator-count, fanout, and wire-length characteristics. Static CMOS implementations reveal that the proposed adders outperform all previously reported solutions in operation speed and/or implementation area.

## References


[1] I. Koren, *Computer Arithmetic Algorithms*, Prentice-Hall, 1993.

[2] T. R. N. Rao and E. Fujiwara, *Error Control Coding of Computer Systems*, Prentice-Hall, 1989.

[3] F. Halsall, *Data Communications, Computer Networks and Open Systems*, Addison Wesley, 1996.

[4] R. M. Jessani and M. Putrino, "Comparison of Single- and Dual-Pass Multiply-Add Fused Floating-Point Units," *IEEE Trans. on Computers*, vol. 47, no. 9, pp. 927–937, Sept. 1998.

[5] R. V. K. Pillai, D. Al-Khalili, and A. J. Al-Khalili, "A Low Power Approach to Floating Point Adder Design," in *Proc. of the IEEE International Conference on Computer Design*, Oct. 1997, pp. 178–185.

[6] A. A. Hiasat and H. S. Abdel-Aty-Zohdy, "Residue-to-binary arithmetic converter for the moduli set $(2^k, 2^k - 1, 2^{k-1} - 1)$," *IEEE Transactions on Circuits and Systems – Part II*, vol. 45, no. 2, pp. 204–209, Feb 1998.

[7] M. Abdallah and A. Skavantzos, "Implementation issues of the two-level residue number system with pairs of conjugate moduli," *IEEE Trans. on Signal Processing*, vol. 47, no. 3, pp. 826–838, Mar. 1999.

[8] Y. Wang, X. Song, M. Aboulhamid, and H. Shen, "Adder based residue to binary number converters for $(2^n - 1, 2^n, 2^n + 1)$," *IEEE Transactions on Signal Processing*, vol. 50, no. 7, pp. 1772–1779, Jul 2002.

[9] C. Efstathiou, D. Nikolos, and J. Kalamatianos, "Area-Time Efficient Modulo $2^n - 1$ Adder Design," *IEEE Trans. on Circuits and Systems II*, vol. 41, no. 7, pp. 463–467, Jul. 1994.

[10] R. Zimmerman, "Efficient VLSI Implementation of Modulo $(2^n \pm 1)$ Addition and Multiplication," in *Proc. of 14th IEEE Symposium Computer Arithmetic*, April 1999, pp. 158–167.

[11] L. Kalampoukas, D. Nikolos, C. Efstathiou, H. T. Vergos, and J. Kalamatianos, "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Trans. on Computers*, vol. 49, no. 7, pp. 673–680, Jul. 2000.

[12] N. Burgess, "The flagged prefix adder and its applications in integer arithmetic," *Journal of VLSI Signal Processing*, vol. 31, no. 3, pp. 263–271, Aug. 2002.

[13] A. Beaumont-Smith and C. C. Lim, "Parallel-prefix adder design," in *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, Apr. 2001, pp. 218–225.

[14] G. Dimitrakopoulos, H. T. Vergos, D. Nikolos, and C. Efstathiou, "A systematic methodology for designing area-time efficient modulo $2^n - 1$ adders," in *Proc. of IEEE International Symposium on Circuits and Systems, to appear*, May 2003.

[15] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. on Computers*, vol. 31, no. 3, pp. 260–264, Mar. 1982.

[16] T. Lynch and E. Swartzlander, "A Spanning Tree Carry Lookahead Adder," *IEEE Trans. on Computers*, vol. C-41, no. 8, pp. 931–939, Aug. 1992.




ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

[17] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. on Computers*, vol. C-22, pp. 786–792, Aug. 1973.

[18] R. E. Ladner and M. J. Fisher, "Parallel Prefix Computation," *Journal of The ACM*, vol. 27, no. 4, pp. 831–838, Oct. 1980.

[19] Simon Knowles, "A family of adders," in *Proc. of 14th IEEE Symp. on Computer Arithmetic*, Apr. 1999, pp. 30–34.

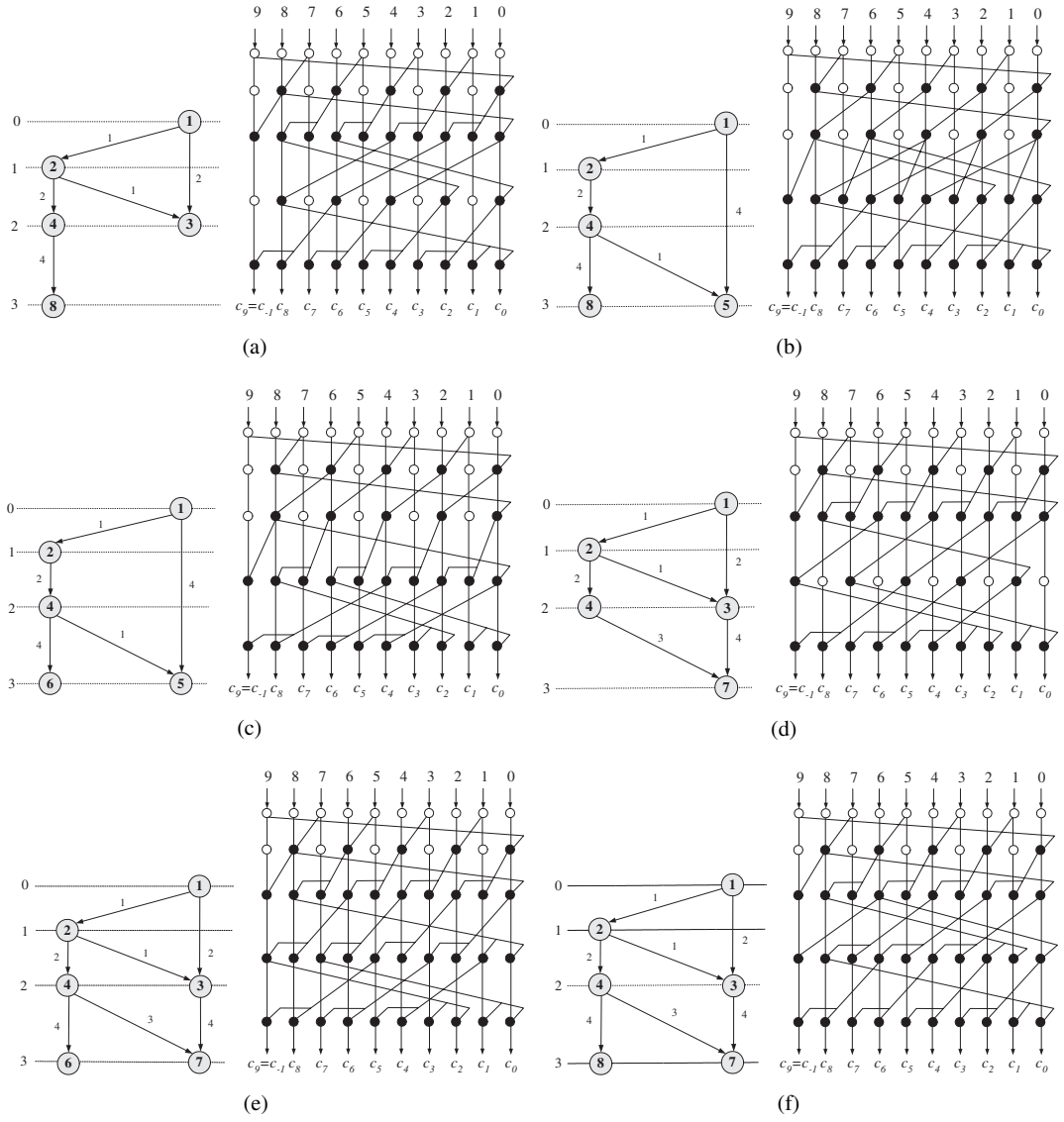


**Figure 5. The modulo $2^{10} - 1$ carry computation units using the (a) $SDG_{10,\{8,3\}}$, (b) $SDG_{10,\{8,5\}}$, (c) $SDG_{10,\{6,5\}}$, (d) $SDG_{10,\{4,7\}}$, (e) $SDG_{10,\{6,7\}}$, and (f) $SDG_{10,\{8,7\}}$.**