High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders

Lampros Kalampoukas, *Member*, *IEEE*, Dimitris Nikolos, *Member*, *IEEE*, Costas Efstathiou, Haridimos T. Vergos, and John Kalamatianos, *Student Member*, *IEEE*

Abstract—A novel parallel-prefix architecture for high speed modulo $2^n - 1$ adders is presented. The proposed architecture is based on the idea of recirculating the generate and propagate signals, instead of the traditional end-around carry approach. Static CMOS implementations verify that the proposed architecture compares favorably with the already known parallel-prefix or carry look-ahead structures.

Index Terms—Modulo $2^n - 1$ adders, parallel-prefix adders, carry look-ahead adders, VLSI design.

1 Introduction

Modulo adders are used in various applications, ranging from Residue Number System (RNS) applications [1], [2], [3], [4], fault-tolerant computer systems [5], [6], [7], [8], [9], and cryptography [10], [11].

In RNS logic, each operand is represented by its moduli with respect to a set of numbers comprising the base. Each of the numbers of the base must not have any common factor with any of the other numbers of the base. Moreover, since each operation is performed in parallel on the moduli in distinct design units (often called channels), the numbers of the base are chosen as close in magnitude as possible. This choice is mandatory for near equal delay among the channels. Most often, the base consists of three numbers: $2^n - 1$, 2^n, $2^n + 1$. Addition then is performed using three channels that in fact are a modulo $2^n - 1$ (equivalently one's complement), a modulo 2^n (the common integer) and a modulo $2^n + 1$ adder [1], [2].

In fault-tolerant computer systems, modulo adders are used for implementing residue, inverse residue, and product (AN) arithmetic codes. For low-cost implementations of such codes, modulo $2^n - 1$ adders are used both for the encoding process as well as for implementing the various arithmetic operations on the encoded operands [5], [6], [7].

In this paper, we propose a new architecture for designing high-speed parallel-prefix modulo $2^n - 1$ adders. The new architecture is based on the idea of modularizing the propagate and generate factors instead of using the traditional end-around carry scheme. This helps in optimizing the fan-out of the gates in the adder architecture and in reducing the number of stages of the adder. The proposed adders require only a small number of different modules interconnected in a regular manner.

This paper is organized as follows: In Section 2, we restate some background issues on the problem of speeding up the addition process. We focus our analysis on parallel-prefix carry computation. We also prove that any parallel-prefix integer adder with a carry input (c_{in}) can be modified to a modulo $2^n - 1$ adder without sequential behavior. Such a modification has also been referred to in [12]. In Section 3, we propose a novel modulo $2^n - 1$ parallel-prefix adder architecture. In Section 4, using a simple area and time model proposed in [13], we present comparative results that show the time efficiency of the proposed architecture against the faster parallel-prefix adders modified according to [12] for performing modulo $2^n - 1$ addition. Static CMOS implementations are also utilized for real comparisons of the proposed against both parallel-prefix and carry look-ahead modulo $2^n - 1$ adders.

2 Preliminaries

In order to speed up the addition operation, the latency of carry computation should be minimized. One solution is to use carry look-ahead (CLA) adders [14]. Unfortunately, when the operand length becomes large, a single level of CLA is insufficient since the number of inputs to the high order gates in the carry generation logic also becomes large. Since, in current VLSI technology, gates with a large number of inputs are either not available or too slow, a gate with a large number of inputs is usually implemented as two or more logic levels of gates with fewer inputs. Increasing the logic levels of a circuit, however, always results in a slower circuit. This problem can be attacked by designing the adders with two or more levels of CLA [14].

Another approach for speeding up the carry computation problem stems from the observation that carry propagation in binary addition is a prefix problem [15]. In

- *L. Kalampoukas is with Xebeo Communications, Inc., 1 Cragwood Rd., South Plainfield, NJ 07080. E-mail: lampros@ieee.org.*
- *D. Nikolos and H.T. Vergos are with the Department of Computer Engineering and Informatics, University of Patras, Patras 26500 Greece. E-mail: {nikolos, vergos}@cti.gr.*
- *C. Efstathiou is with the Department of Informatics, TEI of Athens, Ag. Spyridonos St., 12210 Egaleo, Athens, Greece. E-mail: cefsta@teiath.gr.*
- *J. Kalamatianos is with the Electrical and Computer Engineering Department, Northeastern University, Boston, MA 02115. E-mail: kalamat@ece.neu.edu.*

Manuscript received 1 Sept. 1999; revised 1 Feb. 2000; accepted 10 Mar. 2000. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 111788.

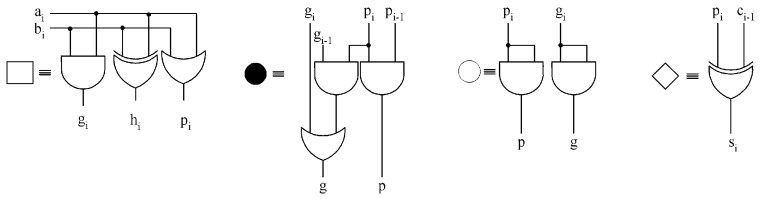

Fig. 1. Prefix logic operators and their implementations.

a prefix problem, n inputs (suppose $x_{n-1}, x_{n-2}, \ldots, x_0$) and an arbitrary associative operator o are used for computing n outputs (suppose $y_{n-1}, y_{n-2}, \ldots, y_0$) according to the relation $y_i = x_i o x_{i-1} o \ldots o x_0$, for $i = 0, \ldots, n-1$.

Carry computation can be transformed into a prefix problem using the associative operator o defined in [16] as follows:

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

Note that o is not commutative since its left argument is treated differently from its right argument. Specifically, let $a_{n-1}a_{n-2}\ldots a_0$ and $b_{n-1}b_{n-2}\ldots b_0$ denote two n-bit operands. Defining the carry generate term g_i and the carry propagate term p_i, for $i = 0, 1, \ldots, n-1$ as:

$$g_i = a_i \cdot b_i$$
$$p_i = a_i + b_i,$$

the carry c_i for each bit position obeys the relation $c_i = G_i$, where

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

After the computation of the carries c_i, the sum bits s_i can be computed in a straightforward way as

$$h_i = a_i \oplus b_i, s_i = h_i \oplus c_{i-1} \text{ for } i \neq 0 \text{ and } s_0 = h_0.$$

($\oplus$ indicates the exclusive-OR operation).

For dealing with the parallel carry computation (assumed as a prefix) problem, several tree structures have been proposed, for example, [16], [17], [18]. Adders that are constructed according to these tree structures are known as parallel-prefix adders. All these adders feature layout regularity, but each structure has its own implementation area, speed, and maximum fan-out characteristics. A full comparison among the already known parallel-prefix adders can be found in [19].

Most often, prefix structures and adders are represented as directed acyclic graphs with the edges representing signals or signal pairs and the nodes standing for the four logic operators presented in Fig. 1.

Using the above representation, any prefix adder structure without a carry input (equivalently, with a carry input equal to 0) has the form of Fig. 2. The carry computation unit can be any of the tree structures already presented in the open literature. The tree structures proposed by Kogge and Stone [17] and Sklansky [18] are the fastest among those already proposed [19]. Figs. 3 and 4 present these structures for $n = 8$. Parallel-prefix adders that use a Kogge-Stone or a Sklansky tree structure will hereafter be denoted as KS or SKL adders, respectively.

The above analysis assumes that the adder has no carry input. The prefix design can be extended to include a carry input $c_{in} = c_{-1}$.

Lemma 1. *Let G'_i, P'_i be the new group generate and the group propagate functions assuming a carry input $c_{in} = c_{-1} \in \{0, 1\}$, where*

$$(G'_i, P'_i) = \begin{cases} (g_0 + p_0 \cdot c_{-1}, p_0), & \text{if } i = 0 \\ (g_i, p_i)\ o\ (G'_{i-1}, P'_{i-1}), & \text{if } 1 \le i \le n-1. \end{cases} \quad (2)$$

Then, for the new carries of the addition, we have that $c'_i = G'_i$ for $0 \le i \le n-1$.

Lemma 1 leads to the implementation of the parallel-prefix adder with c_{in} given in [12, Fig. 2] and its proof is similar to that of Lemma 1 in [16].

Lemma 2. *Consider that (G_i, P_i) and (G'_i, P'_i) are, respectively, defined according to (1) and (2). Then,*

$$(G'_i, P'_i) = (G_i + P_i \cdot c_{-1}, P_i). \quad (3)$$

Proof. The proof will be given by induction on i. The relation holds for $i = 0$, since

$$(G'_0, P'_0) = (g_0 + p_0 \cdot c_{-1}, p_0) = (G_0 + P_0 \cdot c_{-1}, P_0).$$

Suppose that the relation holds for $i = k-1$, that is, suppose that $(G'_{k-1}, P'_{k-1}) = (G_{k-1} + P_{k-1} \cdot c_{-1}, P_{k-1})$. Then, for $i = k$ we get:

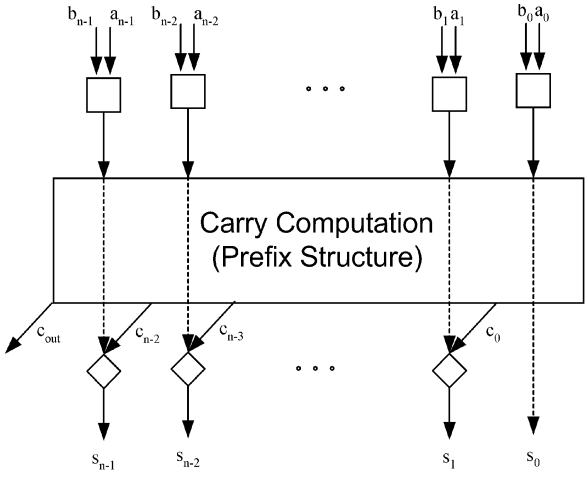

Fig. 2. General structure of a prefix adder (carry input is assumed 0).

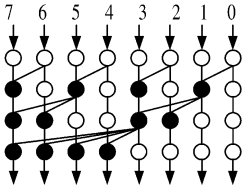

Fig. 3. Sklansky prefix structure.

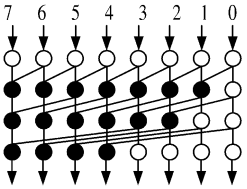

Fig. 4. Kogge-Stone prefix structure.

$$\begin{aligned}(G'_k, P'_k) &= (g_k, p_k)\ o\ (G'_{k-1}, P'_{k-1}) \\ &= (g_k, p_k)\ o\ (G_{k-1} + P_{k-1} \cdot c_{-1}, P_{k-1}) \\ &= (g_k + p_k \cdot (G_{k-1} + P_{k-1} \cdot c_{-1}), p_k \cdot P_{k-1}) \\ &= ((g_k + p_k \cdot G_{k-1}) + p_k P_{k-1} \cdot c_{-1}, P_k) \\ &= (G_k + P_k \cdot c_{-1}, P_k).\end{aligned}$$

Thus, $(G'_i, P'_i) = (G_i + P_i \cdot c_{-1}, P_i)$. □

Lemma 2 implies that a possible design method for a parallel-prefix adder with $c_{in} \in \{0, 1\}$ is to add a single row of logic blocks to the output of the parallel-prefix carry computation without c_{in}. Fig. 5 depicts the structure of such an adder. In [12], [19], a similar adder design was presented without any formal foundation.

The standard implementation of a modulo $2^n - 1$ adder (equivalently, one's complement adder) uses a conventional binary adder with the carry output connected to the carry input to achieve the end-around carry. The adder is either implemented as a ripple-carry, CLA, or prefix-adder (Fig. 2 in [12]). The direct connection of the carry output to carry input in effect turns the adder into an asynchronous sequential circuit. The sequential behavior of the end-around carry adder is the cause of a practical timing problem. Due to the potential race between two stable states, the end-around carry adders may exhibit long delays. In CLA adders, the problem is solved by using as carry input the logical OR of the carry out and the propagate P signals [20] or the logical OR of propagate P and generate G signals [21].

In [22], combinational modulo $2^n - 1$ adders have been presented based on the one-level and the two-level CLA adder structures, respectively. As was shown in [22], the adders given there are significantly faster than the adders given in [20], [21]. Although fast, they lack any regularity. Moreover, a large number of possible combinations among the number of CLA levels and the implementation of each CLA level may have to be investigated for each different implementation technology in order to reach the fastest design.

Modulo $2^n - 1$ addition can be performed in two cycles. During the first cycle, a regular addition takes place. During the second cycle, the carry output of the first addition is added to the result of the first cycle [23]. A simple control circuit (e.g., a multiplexor) is required for data routing.

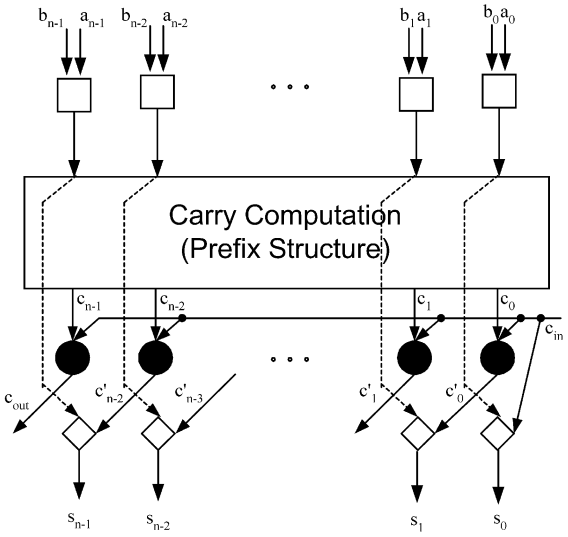

Fig. 5. Parallel-prefix adder structure with c_{in}.

For the adder given in Fig. 5, during the first cycle of the addition we have that $c_{-1} = 0$. Hence, from (3), we get $(G'_{n-1}, P'_{n-1}) = (G_{n-1}, P_{n-1})$. During the second cycle, we have that $c_{-1} = G'_{n-1} = G_{n-1}$. For $c_{-1} = G_{n-1}$, (3) gives

$$(G'_{n-1}, P'_{n-1}) = (G_{n-1} + P_{n-1} \cdot G_{n-1}, P_{n-1}) = (G_{n-1}, P_{n-1}).$$

The above analysis implies that the carry output $c'_{n-1} = G'_{n-1}$ is stable during both cycles of the addition. From (3), we get $(G'_i, P'_i) = (G_i + P_i \cdot G_{n-1}, P_i)$.

The latter relation means that the parallel-prefix adder of Fig. 5 is transformed into a modulo $2^n - 1$ adder using G_{n-1} as c_{in}. Such a design has already been proposed in [12, Fig. 4]. Apart from adding an extra logic operator stage, this modulo $2^n - 1$ adder design has the disadvantage that the reentering carry has a fan-out of n.

3 A Novel Parallel-Prefix Modulo $2^n - 1$ Adder Implementation

In the following, we will show, using the operator o, that the computation of the carries c_i^* of the modulo $2^n - 1$ adder, for $-1 \le i \le n-2$, can be transformed into a parallel computation. To this end, the following theorem is necessary:

Theorem 1. *Let*

$$(G_i^*, P_i^*) = \begin{cases} (G_{n-1}, P_{n-1}) & \text{if } i = -1 \\ (g_i, p_i)\ o\ (G_{i-1}^*, P_{i-1}^*) & \text{for } 0 \le i \le n-2, \end{cases} \quad (4)$$

where G_{n-1} and P_{n-1} are computed by (1). Then, $c_i^* = G_i^*$ for $-1 \le i \le n-2$, where c_i^* are the carries of the modulo $2^n - 1$ addition.

Proof. We will use induction on i in this case also.

1. For $i = -1$, we have $(G_{-1}^*, P_{-1}^*) = (G_{n-1}, P_{n-1})$. As explained previously, $c_{-1} = G_{n-1}$. Therefore, $c_{-1}^* = G_{-1}^*$.

2. We assume that (4) holds for $i = k-1$, with $k \geq 0$. This means that $c^*_{k-1} = G^*_{k-1}$. We will prove that it holds for $i = k$, too. By definition, we have that $(G^*_k, P^*_k) = (g_k, p_k)\ o\ (G^*_{k-1}, P^*_{k-1})$. Taking into account that $G^*_{k-1} = c^*_{k-1}$, we get:

$$\begin{aligned}(G^*_k, P^*_k) &= (g_k, p_k)\ o\ (c^*_{k-1}, P^*_{k-1}) \\ &= (g_k + p_k \cdot c^*_{k-1}, p_k \cdot P^*_{k-1}),\end{aligned}$$

that is, $G^*_k = g_k + p_k \cdot c^*_{k-1}$. The second part of this equation is the definition of c'_k. Therefore, the theorem has been proven and $c^*_i = G^*_i$ for $-1 \leq i \leq n-2$. □

In the sequel, we will present a new architecture for modulo $2^n - 1$ adders. The resulting adders can operate at higher speeds, retain a regular layout based on the above building blocks, and each block has a fan-out less than or equal to 2, without the need of buffer insertion. In order to introduce the formal description of the proposed architecture, we will make use of the following lemma.

Lemma 3. $(G_i, P_i)\ o\ (g, p)\ o\ (G_i, P_i) = (G_i, P_i)\ o\ (g, p)$.

Proof.

$$\begin{aligned}(G_i, P_i)\ o\ (g, p)\ o\ (G_i, P_i) &= (G_i + P_i \cdot g, P_i \cdot p)\ o\ (G_i, P_i) \\ &= (G_i + P_i \cdot g + P_i \cdot p \cdot G_i, P_i \cdot p \cdot P_i) \\ &= [G_i(1 + P_i \cdot p) + P_i \cdot g, P_i \cdot p] \\ &= (G_i + P_i \cdot g, P_i \cdot p) \\ &= (G_i, P_i)\ o\ (g, p).\end{aligned}$$

□

From (4), we have that

$$\begin{aligned}(G^*_i, P^*_i) &= (g_i, p_i)\ o\ (G^*_{i-1}, P^*_{i-1}) = \ldots = \\ &= (g_i, p_i)\ o\ (g_{i-1}, p_{i-1})\ o \ldots o\ (g_0, p_0)\ o\ (G_{n-1}, P_{n-1}) \\ &= (g_i, p_i)\ o \ldots o\ (g_0, p_0)\ o\ (g_{n-1}, p_{n-1})\ o\ (G_{n-2}, P_{n-2}) \\ &= (g_i, p_i)\ o \ldots o(g_0, p_0)\ o\ (g_{n-1}, p_{n-1})\ o \ldots \\ &\quad o\ (g_{i+1}, p_{i+1})\ o\ (g_i, p_i)\ o \ldots o(g_0, p_0) \\ &= (G_i, P_i)\ o\ (g_{n-1}, p_{n-1})\ o \ldots o\ (g_{i+1}, p_{i+1})\ o\ (G_i, P_i).\end{aligned}$$

According to Lemma 3, we get:

$$(G^*_i, P^*_i) = (G_i, P_i)\ o\ (g_{n-1}, p_{n-1})\ o \ldots o\ (g_{i+1}, p_{i+1}).$$

Then,

$$\begin{aligned}(G^*_i, P^*_i) = (g_i, p_i)\ o\ (g_{i-1}, p_{i-1})\ o \ldots o\ (g_0, p_0) \\ o\ (g_{n-1}, p_{n-1})\ o \ldots o(g_{i+1}, p_{i+1}).\end{aligned} \quad (5)$$

Relation (5) defines the proposed parallel-prefix architecture for designing modulo $2^n - 1$ adders. It shows that a modulo $2^n - 1$ addition can be implemented if the group generate and the group propagate terms, G^*_i $(= c^*_i)$ and P^*_i, respectively, at each bit position i $(-1 \leq i \leq n-2)$ are designed as a function, using the operator o, not only of the propagate and generate terms from 0 through i, but also of the terms from $i+1$ through $n-1$. A possible design method to achieve this is to modify a Kogge-Stone prefix structure such that, at every stage m, with $1 \leq m \leq \log_2 n$, the (g, p) output terms of the highest order 2^{m-1} operators of the previous stage are also fed to the lowest order 2^{m-1} operators of stage m. Viewing it from a higher abstraction level, (5) shows that the feedback of P, G signals, required for performing the modulo $2^n - 1$ addition, can be done at each existing prefix level instead of assigning it to only a last extra one. Therefore, the extra prefix level (the last row of • in Fig. 5) can be removed, leading to the proposed faster design. As an example, the outcoming modulo $2^n - 1$ adder for $n = 8$ is presented in Fig. 6. Several observations can be made on the proposed architecture. First, the number of prefix levels is equal to those required by an integer adder. Then, the regularity of the prefix structure is not disturbed and the fan-out of every logic operator is less than or equal to 2.

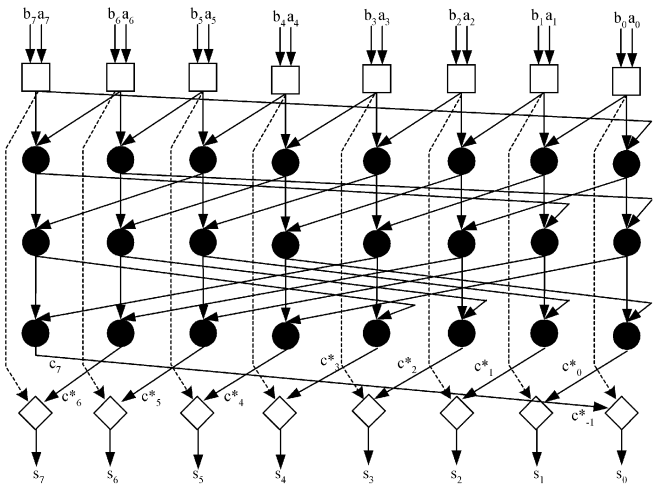

Fig. 6. Proposed modulo $2^8 - 1$ adder architecture.

The modulo $2^n - 1$ adders designed according to the proposed procedure have two representations of zero (all 0s and all 1s). In some cases, this is desirable [6], [8], [9]. In an application where the all 1s representation is undesirable, its elimination can be achieved by minor modifications of the proposed architecture. The proposed adder produces the all 1s representation of zero when both input operands are all 1s or when the input operands are complementary. In an application with a single representation of zero, the proposed adders will produce an all 1s output only if the input operands are complementary. In the case of complementary inputs, note that $h_i = 1$, for $i = 0, 1, \ldots, n-1$, and also $c^*_i = 0$, for $i = -1, 0, \ldots, n-2$. A modification that avoids the all 1s output is to produce the final sum as $s_i = (h_i \cdot \overline{H_{n-1}} \oplus c^*_{i-1}$, for $0 \leq i \leq n-1$, instead of $s_i = h_i \oplus c^*_{i-1}$, where $H_{n-1} = h_1 \cdot h_2 \cdot \ldots h_{n-1}$ and $\overline{H_i}$ is the complement of H_i. A possible way to achieve this is to define $p_i = h_i = a_i \oplus b_i$ and modify the last row logic operators • in order to implement their p output as $p = \overline{\overline{p_i} + p_{i-1}}$. The resulting modulo $2^n - 1$ adders with single zero representation have one more gate level than the proposed modulo $2^n - 1$ adders with double zero representation. The above analysis, which is based on $p_i = a_i + b_i$, is also valid if we define $p_i = h_i = a_i \oplus b_i$, but, in this case, the adders are slightly slower since, in the first stage, an exclusive-OR is used instead of an OR gate.

TABLE 1
Adder Area and Delay Model Estimations

Adder	Area	Delay
Integer from [13]	$\frac{3}{2}n\log n + 4n$	$2\log n + 3$
m-KS	$3n\log n + 4n + 1$	$2\log n + 5$
m-SKL	$\frac{3}{2}n\log n + 7n - 2$	$2\log n + 5$
Proposed	$3n\log n + 4n$	$2\log n + 3$

4 Comparison Results

For our comparisons, we use both a simple model and actual static CMOS implementations. We use the notation m-SKL and m-KS for parallel-prefix adders with a Sklansky and a Kogge-Stone prefix structure, respectively, modified to perform modulo $2^n - 1$ addition as proposed in [12, Fig. 4] and [19].

We will at first make use of the model originally presented in [13], which was also used in [12], under the notation "unit-gate model." 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-in and fan-out; therefore, the validation of the estimates that it produces will be carried out later by CMOS static implementations.

Table 1 presents the gate-count and gate-delay estimates as a function of the word length n for m-KS, m-SKL and the proposed adders. In Table 1, we have also included the results of [12] for an integer (2's complement) adder with a Sklansky parallel-prefix structure. The white nodes in the carry computation prefix structure of Figs. 3 and 4, have not been taken into account in the derivation of the unit gate area formulas.

As far as the implementation area is concerned, the above table reveals that the proposed architecture requires approximately the same area as an m-KS architecture. The estimations for the area required by the m-SKL and integer adders are smaller. We have to note though, that these estimations do not take into account the area that may be required for buffer insertion needed to alleviate the unlimited fan-out problem of Sklansky prefix structures, as well as the more complex wiring of the Kogge-Stone prefix structures.

In order to obtain more realistic results m-KS, m-SKL [12], [19], the CLA adders proposed in [22] and the proposed modulo $2^n - 1$ adders for $n = 8$, 16, 32, 64, and 128 were described in HDL. We used the Synopsys tools [24] driven by the AMS CUB implementation technology (0.6 μm, 2-metal layer, 5.0 V) for our implementations. All results associated with circuit delay were gathered assuming worst case process parameters and are measured in ns. All area results use mils^2 as a metric. We have to note that:

1. The use of the specific implementation technology is not advantageous to the adders proposed in this work. The latter could profit more by a technology with more than two interconnection layers. Therefore, we chose to present the area (prelayout estimations) results in three columns corresponding to the cell-area, the routing area and the total area.
2. The CLA adders proposed in [22] have also been included in our comparisons. The author of [12] asserts that: a) These adders have a complex circuit structure and b) they result in large implementation area. We did not face any particular problem in modeling the CLA adders of [22] and our simulations showed that assertion b) above is incorrect in several cases. Our simulation results for all examined values of n (8, 16, 32, 64, or 128) indicated that two-level CLA modulo $2^n - 1$ adders lead to better delay results compared to single level CLA adders. Therefore, in Tables 2 and 3, we present only results for two-level CLA adders and only the specific organization of them that resulted in the best delay results. We also present in parentheses the number of groups of the GPG unit [22] that provided the minimum delay results.

We simulated two different design approaches: a bottom-up approach aiming netlist regularity and an aggressive optimization approach.

For the first approach, each building block (the GPG, BGCLA, GCLA units from [22], as well as the three logic operators of Fig. 1 excluding $\bigcirc$) was mapped to the target technology aiming at the minimum delay. A "don't touch" primitive was then applied to all optimized building blocks. The architectures under comparison were then described as an interconnection of these optimized blocks. We instructed the tool to optimize the outcoming adders for speed by inserting buffers where necessary or appropriate for dealing with fan-out problems, but keeping intact the building blocks. Table 2 presents the results obtained using this approach.

The results of Table 2 indicate that the proposed adders are faster compared to m-KS, m-SKL, and the CLA adders by, respectively, 14.7 percent, 11.8 percent, and 10.1 percent on the average. They require approximately the same area as a m-KS architecture, but approximately 34.5 percent and 105.7 percent more implementation area than an m-SKL and the CLA adders on the average.

For our second approach, we removed the "don't touch" primitive from the building blocks. This effectively led to a flattened netlist. All the netlists were then optimized for speed. As a secondary target, the tool was instructed to try to recover as much area as possible. Table 3 lists the results obtained by this second approach.

The results of Table 3 indicate that that the proposed architecture leads to 16.2 percent, 16 percent, and 21.8 percent faster designs on the average compared to m-KS, m-SKL, and two stage CLA modulo $2^n - 1$ architectures. The area overhead in this case is 15.5 percent, 50.6 percent, and 27.5 percent compared, respectively, to m-KS, m-SKL, and CLA architectures.

We also synthesized the proposed adders targeting a delay equal to or less than the delay of the fastest among the rest of the modulo $2^n - 1$ adders. The results are presented in Table 4. As we can see from Tables 3 and 4, the proposed

TABLE 2
Area-Delay Results for Bottom-Up Approach

n	Architecture	Cell Area	Routing Area	Total Area	Delay
8	m-KS	113.1	155.1	268.1	6.42
	m-SKL	92.0	130.2	222.2	6.20
	CLA (4)	73.7	111.2	184.9	6.13
	Proposed	108.7	152.4	261.2	5.27
16	m-KS	289.5	393.3	682.8	7.86
	m-SKL	215.3	302.2	517.5	7.74
	CLA (4)	126.4	218.6	345.0	7.57
	Proposed	294.7	398.0	692.7	6.65
32	m-KS	706.9	965.4	1668.1	9.28
	m-SKL	500.9	961.2	1198.5	9.03
	CLA (8)	252.5	448.3	700.8	8.81
	Proposed	735.0	965.4	1700.5	7.97
64	m-KS	1847.9	2414.1	4262.1	10.85
	m-SKL	1277.6	1699.6	2977.1	10.38
	CLA (8)	687.5	1213.7	1901.2	9.86
	Proposed	1774.7	2303.5	4078.2	9.35
128	m-KS	4174.4	5480.9	9655.3	12.22
	m-SKL	2819.1	3760.1	6579.3	11.70
	CLA (16)	1457.5	2624.8	4082.3	11.90
	Proposed	4071.7	5284.5	9356.2	10.73

TABLE 3
Area-Delay Results for Flattened Implementations Targeting Minimal Delay and Then Area Recovery

n	Architecture	Cell Area	Routing Area	Total Area	Delay
8	m-KS	66.5	122.3	188.8	4.02
	m-SKL	55.9	108.5	164.4	3.89
	CLA (4)	68.2	134.0	202.2	3.57
	Proposed	67.4	129.1	196.6	3.30
16	m-KS	164.7	304.9	469.5	4.75
	m-SKL	130.2	246.7	376.9	4.84
	CLA (4)	153.4	298.5	452.0	5.17
	Proposed	202.8	385.9	588.7	4.05
32	m-KS	352.3	670.1	1022.4	5.90
	m-SKL	247.1	476.4	723.4	5.90
	CLA (8)	346.0	673.3	1019.3	6.67
	Proposed	449.9	891.3	1341.2	4.97
64	m-KS	980.2	1854.7	2834.9	6.98
	m-SKL	761.1	1326.4	2087.6	6.68
	CLA (8)	784.5	1434.4	2218.9	7.90
	Proposed	973.7	2040.5	3014.2	5.66
128	m-KS	2118.5	4122.2	6240.7	7.54
	m-SKL	1740.5	2941.8	4682.3	7.85
	CLA (16)	1704.9	3146.2	4851.1	8.76
	Proposed	2318.8	4588.0	6906.8	6.49

adders offer at least the same speed as the best of the rest of the architectures. At the same time, they achieve a smaller implementation area in all but one cases.

The estimations of Table 1 reveal that the proposed modulo $2^n - 1$ adders, apart from being the fastest among the proposed parallel-prefix modulo $2^n - 1$ adders, are also as fast as an integer parallel-prefix adder using a Sklansky tree structure (which, together with the Kogge-Stone tree structure, are considered to be the fastest [19]). For realistic comparisons, Sklansky and Kogge-Stone integer adders for

TABLE 4
Area-Time Optimization Results of the Proposed Adders

n	Cell Area	Routing Area	Total Area	Delay
8	57.2	99.0	156.2	3.56
16	130.0	242.9	372.9	4.69
32	274.2	519.8	794.0	5.90
64	555.5	1185.6	1741.2	6.54
128	1437.4	3048.8	4486.2	7.54

$n = 8$, 16, 32, 64, and 128 were designed following both the bottom-up and the aggressive optimization approaches. The results obtained show that the proposed modulo $2^n - 1$ adders are at most 0.32 and 0.45 ns slower than the integer ones designed using the first and the second design approach, respectively. As a consequence, in an RNS application, the modulo $2^n - 1$ channel can perform addition with almost the same speed as the modulo 2^n (the normal integer) channel.

5 Conclusions

In this paper, we have presented a novel architecture for designing high-speed modulo $2^n - 1$ adders. The proposed adders have a parallel-prefix carry computation structure and therefore can be implemented by interconnecting only a small number of different modules, leading to an overall regular design. Static CMOS implementations have shown that the proposed modulo $2^n - 1$ adders compare favorably with the other already known parallel-prefix or carry look-ahead adders. Moreover, the modulo $2^n - 1$ adders designed according to the proposed architecture are almost as fast as the fastest parallel-prefix integer (modulo 2^n) adders, making them ideal for RNS applications.

References

[1] M.A. Sonderstrand et al., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing.* New York: IEEE Press, 1986.

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

[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] M.A. Bayoumi et al., "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.

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

[6] B.J. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems.* Addison-Wesley, 1989.

[7] T.R.N. Rao, *Error Coding for Arithmetic Processors.* Academic Press, 1974.

[8] D. Nikolos et al., "Efficient Design of Totally Self-Checking Checkers for All Low Cost Arithmetic Codes," *IEEE Trans. Computers,* vol. 37, no. 7, pp. 807-814, July 1988.

[9] I.L. Sayers and D.J. Kinniment, "Low-Cost Residue Codes and Their Application to Self-Checking VLSI Systems," *IEE Proc. Computers and Digital Techniques,* vol. 132, no. 4, pp. 197-202, 1985.

[10] 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.

[11] A. Curiger, "VLSI Architectures for Computations in Finite Rings and Fields," PhD thesis, Swiss Federal Inst. of Technology (ETH), Zurich, 1993.

[12] 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.

[13] A. Tyagi, "A Reduced-Area Scheme for Carry-Select Adders," *IEEE Trans. Computers,* vol. 42, no. 10, pp. 1,163-1,170, Oct. 1993.

[14] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design.* John Wiley & Sons, 1979.

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

[16] 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.

[17] 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.

[18] J. Sklansky, "Conditional Sum Addition Logic," *IRE Trans. Electronic Computers,* vol. 9, no. 6, pp. 226-231, June 1960.

[19] R. Zimmermann, "Binary Adder Architectures for Cell-Based VLSI and Their Synthesis," PhD thesis, Swiss Federal Inst. of Technology, Zurich, 1997 (available at http://www.iis.ee.ethz.ch/~zimmi).

[20] J.J. Shedletsky, "Comment on the Sequential and Indeterminate Behavior of an End-Around-Carry Adder," *IEEE Trans. Computers,* vol. 26, pp. 271-272, Mar. 1977.

[21] J.F. Wakerly, "One's Complement Adder Eliminates Unwanted Zero," *Electronics,* pp. 103-105, Feb. 1976.

[22] C. Efstathiou, D. Nikolos, and J. Kalamatianos, "Area-Time Efficient Modulo $2^n - 1$ Adder Design," *IEEE Trans. Circuits and Systems-II: Analog and Digital Signal Processing,* vol. 41, no. 7, pp. 463-467, July 1994.

[23] M.A. Bayoumi, G.A. Jullien, and W.C. Miller, "A VLSI Implementation of Residue Adders," *IEEE Trans. Circuits and Systems,* vol. 34, pp. 284-288, Mar. 1987.

[24] Design Analyzer Tools Suite, version 1998.08, Synopsys Inc.

Lampros Kalampoukas received his Diploma in computer engineering and informatics from the University of Patras, Greece, his MS degree in 1995 and his PhD degree in 1997, both in computer engineering, from the University of California at Santa Cruz. Since 1997, he has been with the High Speed Networks Research Department at Bell Laboratories, where he has contributed to the architecture, design, and development of high-speed and QoS-capable IP switches. He is the winner of the 1998 Bell Laboratories President's Gold Award. He has published extensively in the areas of traffic management, congestion and flow control, and protocol performance. He is also the recipient of the Design SuperCon '97 Best Paper Award for his work on the design, development, and evaluation of highly efficient bandwidth and congestion management algorithms for the ABR service in ATM networks. He has more than 20 publications in technical journals and conferences and holds several patents. He currently works on the architecture and development of reliable and scalable multi-service optical switches and routers for the data networking and the communications transport infrastructure. His research interests are in the areas of high-speed router and switch architectures, as well as in transport and routing protocol analysis, modeling, and performance. He is a member of the IEEE and the IEEE Communications Society.

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, Greece. He is currently a full professor in the Computer Engineering and Informatics Department of the University of Patras and head of the Technology and Computer Architecture Laboratory. He has served as program cochairman of the last four IEEE On-Line Testing Workshops. He also served on the program committees for the IEEE International On-Line Testing Workshop in 1995 and 1996, for the 1997, 1998, and 1999 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, for the Third European Dependable Computing Conference, and for the DATE-2000 Conference. 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 90 scientific papers. He is a member of the IEEE.

Costas Efstathiou received the BS degree in physics in 1979 and the MS degree in electronics in 1982, both from the University of Athens, Greece. In 1985, he received the PhD degree in computer science from the University of Thessaloniki, Greece. From 1985 to 1986, he worked at the Research Institute of the Greek Air Force and, from 1988 to 1994, at the Hellenic Telecommunications Organization. Since 1994, he has been a professor in the Department of Informatics of the TEI of Athens, Greece. His current research interests include fault-tolerant computer systems, computer arithmetic and computer networks.

Haridimos T. Vergos received the Diploma in computer engineering and his PhD degree from the University of Patras, Greece, in 1991 and 1996, respectively. In 1998, he worked as an ASIC designer in the Multimedia and Networking Group of Atmel Inc. Currently, he holds a lecturer position in the Computer Engineering and Informatics Department of the University of Patras, Greece. His main research interests include fault-tolerant digital systems and reliable design and test of computer systems. He is a member of the Greek Computer Engineering Society and the Technical Chamber of Greece.

John Kalamatianos received his BS degree in computer engineering and informatics from University of Patras, Greece, in 1993 and his MS and PhD degrees in electrical and computer engineering from Northeastern University, Boston, in November 1995 and January 2000, respectively. His research interests are primarily in the area of processor microarchitecture, compiler optimizations, and VLSI design, with particular emphasis on cache and memory hierarchy design, branch and value prediction, prefetching, compiler optimizations for ILP, memory performance, and computer arithmetic. He has held summer internships at the IBM T.J. Watson Center and at Microsoft Research. He is a student member of the ACM and the IEEE.