Exploration of High-Speed Modulo $2^n - 1$ Adders

Konstantinos Vasilas, Theofanis Vergos, Polykarpos Vergos and Haridimos T. Vergos
Department of Computer Engineering & Informatics,
University of Patras, Rio, 26504, Greece
E-mails: vasilas@ceid.upatras.gr, st1072560@ceid.upatras.gr,
st1072561@ceid.upatras.gr, vergos@upatras.gr

***Abstract*—Existing efficient parallel-prefix modulo 2^n-1 adder designs either rely on standard carry equations or apply a specific factorization of these equations only at the initial level. In this paper, we introduce two extensions to the design theory of parallel-prefix modulo $2^n - 1$ adders. We show that several factorizations are possible and that these factorizations can be applied at any prefix level, rather than being restricted to the initial one. Combined with the use of higher-valency operators, these extensions enable a broader exploration of the design space and lead to a larger family of adders. Previously proposed parallel-prefix architectures can be viewed as specific members of this generalized family. Experimental results show that several newly derived members of this adder family outperform previously reported designs in speed and achieve the best efficiency under the area $\times$ delay^2 metric.**

***Index Terms*—Modulo 2^n-1 adders, parallel-prefix computation, Ling adders, Residue Number System (RNS)**

I. Introduction

Modulo $2^n - 1$ arithmetic plays an important role in a wide range of computational systems where high-speed and area-efficient arithmetic operations are required. Its properties make it particularly attractive in applications such as fault-tolerant computing [1] and checksum computations in high-speed networks [2], where modular operations are frequently used to detect errors and verify data integrity.

In addition, modulo $2^n - 1$ arithmetic is a fundamental component of Residue Number Systems (RNS), which represent numbers using several smaller, independent residue channels. This representation enables carry-free arithmetic between channels and offers a high degree of parallelism, making RNS especially suitable for computationally intensive applications. Many RNS-based applications, including digital signal and image processing [3]–[5], and neural network acceleration [6]–[10], can benefit from this parallel arithmetic structure. In such systems, arithmetic units are often required to perform a large number of additions with low delay, small area, and reduced power consumption. Since addition is one of the most frequently used operations in these workloads, the design of efficient modulo $2^n - 1$ adders is essential and has a direct impact on the overall performance, area, and energy efficiency of the system. For this reason, modulo $2^n - 1$ adder design has become a widely studied research topic, and considerable effort has been devoted to improving the speed and hardware efficiency of these adders.

Several architectures have been proposed to address the design of efficient modulo 2^n-1 adders. Single- and two-level carry-lookahead (CLA) adders were introduced in [11], to reduce the addition delay, while a carry-select-based solution was explored in [12]. More efficient designs based on parallel-prefix carry computation were presented in [13] and [14]. The architecture in [13] performs the modulo operation by feeding the carry-out of a conventional parallel-prefix integer adder back to all carry positions through an additional prefix level. However, this feedback mechanism leads to a maximum wire fan-out of n, increasing the critical-path delay. To overcome this limitation, [14] proposed recirculating the carry-generate and carry-propagate signals within the existing prefix structure, thereby canceling the need for an extra prefix level.

Another important direction in fast adder design was introduced by Ling [15], who proposed a simplified carry formulation by factoring the most significant carry-propagate term out of each carry equation. The resulting carry definition, commonly referred to as the Ling carry, reduces the complexity of carry computation and has been used to design efficient parallel-prefix integer adders [16] and modulo 2^n-1 adders in [17]. The original Ling formulation considers only a single factorization, applied at the initial level of the adder. For integer adders, it was shown in [18] that the factorization introduced by Ling is one among several possible alternatives. Moreover, these alternative factorizations can be applied not only at the initial level, but also at any level of the parallel-prefix structure. As a result, further simplifications of the carry equations become possible, leading to faster integer adder implementations.

In this paper, we show that different factorizations of the modulo $2^n - 1$ carry equations are also possible, and that such factorizations can be applied not only at the initial level, but also at any prefix level of the adder. These observations, combined with the use of high-valency prefix operators, allow us to introduce a new family of modulo $2^n - 1$ adders. Each member of this family is characterized by the factorization degree and the valency of the prefix operators used at each prefix level. The previously proposed architectures in [14] and [17] can be viewed as special cases of the proposed design space. By exploring this extended design space, we identify new members of the proposed family that outperform all previously reported modulo $2^n - 1$ adders in terms of speed, while also achieving the best efficiency under the area $\times$ delay^2 metric.

The rest of this paper is organized as follows. Section II reviews the fundamentals of parallel-prefix integer and modulo

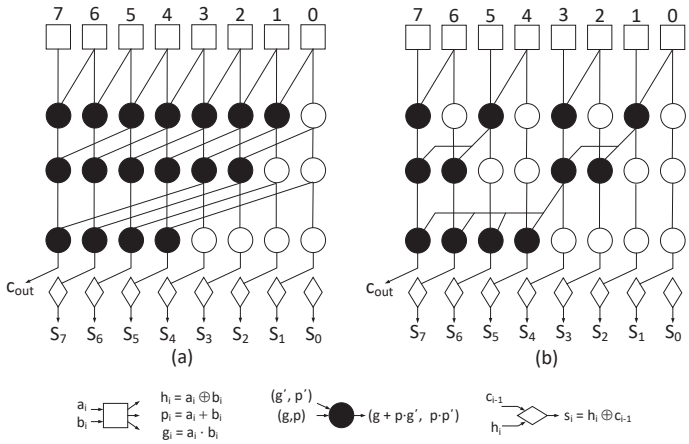

Fig. 1. 8-bit integer adder using a Kogge-Stone (a) or a Ladner-Fisher (b) parallel-prefix carry computation unit.

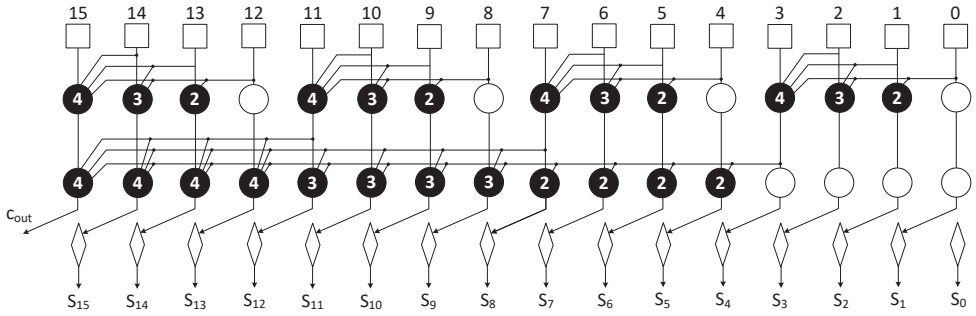

Fig. 2. 16-bit integer adder with up to valency–4 prefix operators.

$2^n - 1$ addition, along with the different carry formulations used in such adders. Section III introduces the proposed extended design theory for parallel-prefix modulo $2^n - 1$ adders. Section IV presents quantitative comparison results against previously proposed architectures.

II. Background

A. Parallel-prefix addition

Let $A = \{a_{n-1}, a_{n-2}, ..., a_0\}$ and $B = \{b_{n-1}, b_{n-2}, ..., b_0\}$ represent the two n-bit operands to be added, and let $S = \{s_{n-1}, s_{n-2}, ..., s_0\}$ denote their sum. An adder that performs the addition can be considered as a circuit composed by the initial (preprocessing) stage, the carry computation unit and the summation stage.

The initial stage computes the carry generate (g_i), the carry propagate (p_i) and the half-sum (h_i) bits, according to $g_i = a_i \cdot b_i$, $p_i = a_i + b_i$ and $h_i = a_i \oplus b_i$, for $0 \leq i \leq n-1$, where $\cdot, +$ and $\oplus$ denote the logical AND, OR, and exclusive-OR operations, respectively. The carry computation stage 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. Finally, the summation stage computes the sum bits according to $s_i = h_i \oplus c_{i-1}$. Carry computation is transformed into a parallel-prefix problem using the $\circ$ operator, which associates pairs of generate and propagate signals and was defined in [19] as $(g, p) \circ (g', p') = (g + p \cdot g', p \cdot p')$. In a series of associations of consecutive generate and propagate pairs the notation $(G_{k:j}, P_{k:j})$, is used to denote the group generate and group propagate terms, respectively, produced out of bits $k, k-1, ..., j+1, j$, that is,

$$\begin{aligned}(G_{k:j}, P_{k:j}) &= (g_k, p_k) \circ (g_{k-l}, p_{k-1}) \circ \ldots \circ \\ &\quad \circ (g_{j+1}, p_{j+1}) \circ (g_j, p_j) \end{aligned} \quad (1)$$

In integer addition, the carries c_k are equal to the group generate $G_{k:0}$.

The parallel-prefix carry computation structures are often represented as directed acyclic graphs. In these graphs the prefix operators are represented by black nodes which are placed on a grid of rows (prefix levels) and bit columns. As an example, Fig. 1 presents the parallel-prefix structures proposed by Kogge-Stone [20] and Ladner-Fisher [21] for the design of an 8-bit adder (buffering nodes used are indicated as white circles). Both these approaches use $\lceil \log_2 n \rceil$ prefix levels for the design of an n-bit integer adder with no carry input, when valency–2 prefix operators are used.

For achieving a lower number of prefix levels and in some cases lower logical levels, higher valency operators can be used [22]. For example, Fig. 2 presents the parallel-

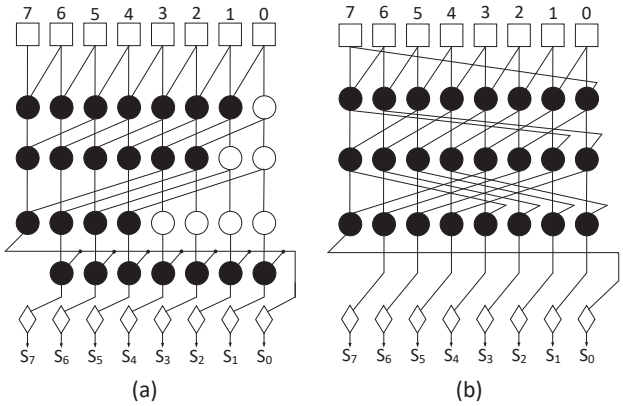

Fig. 3. Modulo $2^8 - 1$ parallel-prefix adder designs

prefix graph of a 16-bit integer adder that requires only two prefix levels by using prefix operators of various valencies. The valency of the operators used is shown as a number within each operator. It is noted that in all parallel-prefix graphs, the connection order of the (g, p) inputs to a prefix operator is important. For example, a valency–4 operator produces $(G_{3:0}, P_{3:0})$, if $(g_3, p_3), (g_2, p_2), (g_1, p_1)$ and (g_0, p_0) are driven left to right at its inputs as shown in the graph. In the rest of this paper we consider that valency–2 or valency–4 operators are used.

B. Modulo $2^n - 1$ parallel-prefix adders

In the following we assume modulo $2^n - 1$ adders with double representation of zero, that is, adders that may produce all 0s or all 1s at their output to indicate a zero result. Modifications for adders that only produce the all 0s output to indicate a zero result have been presented in [14], [23]. We further assume that a normal weighted binary encoding is used for the inputs and outputs. Other encodings that have been considered (for example signed-lsb [24]) and that require forward / reverse converters from / to the normal representation are out of the scope of this paper.

The operation of modulo $2^n - 1$ adder is defined as:

$$|A + B|_{2^n-1} = \begin{cases} |A + B|_{2^n}, & \text{if } A + B < 2^n \\ |A + B|_{2^n} + 1, & \text{if } A + B \geq 2^n. \end{cases}$$

or equivalently, $|A+B|_{2^n-1} = |A+B|_{2^n} + c_{n-1}$ where c_{n-1} is the carry output of the integer addition of A with B.

Although a direct connection of the carry output of an integer adder back to its carry input results into a modulo $2^n - 1$ adder, it also leads to a circuit that suffers from oscillations [25] and requires significant time to settle to its correct output due to races. Therefore, in [13] it was suggested to perform the addition of c_{n-1} to the $A + B$ sum by an additional carry increment stage as shown in Fig. 3(a). Since the carry increment stage requires an extra level of prefix operators which are all driven by the carry output of the integer adder, such an adder design suffers from increased delay.

In [14], it was shown that the i-th carry c_i^* (* is used in the following to differentiate modulo $2^n - 1$ carries against integer ones) of the modulo $2^n - 1$ addition is given by:

$$c_i^* = G_{i:0} + P_{i:0} G_{n-1:i+1} \tag{2}$$

which can be computed using the $\circ$ operator as:

$$(g_i, p_i) \circ ... \circ (g_0, p_0) \circ (g_{n-1}, p_{n-1}) \circ ... \circ (g_{i+1}, p_{i+1}).$$

In the parallel-prefix carry computation proposed by [14], and shown in Fig. 3(b) for the case that $n = 8$, the carries are computed by recirculating the intermediate generate and propagate terms in the existing prefix levels of a parallel-prefix tree, resulting in faster adders than those proposed in [13].

It is obvious that carry computation in a modulo 2^n-1 adder is more complex than that of an integer adder. In the modulo case all generate and propagate terms are associated in the computation of every carry in a circular manner compared to only those of the current and the less significant bit positions in the integer adder case. Because of this circular form and for the ease of presentation, when referring to modulo $2^n - 1$ adders in the following we use the notations X_i even for negative values of i, to denote signal $X_{|i|_n}$. We also use $G_{i:j}$ and $P_{i:j}$ even when $j > i$, to denote the group generate and group propagate out of group of bits in a circular way, that is, in these cases $G_{i:j} = G_{i:0} + P_{i:0} G_{n-1:j}$ and $P_{i:j} = P_{i:0} P_{n-1:j}$.

C. Ling factorization

By observing that $g_i = p_i g_i$, Ling [15] proposed that a propagate term is factored out of the carry equation. In this way the carry computation becomes simpler at the expense of a more complicated preprocessing stage. Since the analysis is similar for both integer and modulo $2^n - 1$ adders, we exemplify this factorization by considering the modulo $2^8 - 1$ case. According to (2) the carry c_3 of a modulo $2^8 - 1$ adder is equal to:

$$\begin{aligned} c_3^* &= g_3 + p_3 g_2 + \ldots + p_3 p_2 p_1 g_0 + \\ &\quad p_3 p_2 p_1 p_0 g_7 + \ldots + p_3 p_2 p_1 p_0 p_7 p_6 p_5 g_4 \end{aligned} \tag{3}$$

Using that $g_3 = p_3 g_3$, (3) can be rewritten as:

$$\begin{aligned} c_3^* &= p_3 (g_3 + g_2 + \ldots + p_2 p_1 g_0 + \\ &\quad + p_2 p_1 p_0 g_7 + ... + p_2 p_1 p_0 p_7 p_6 p_5 g_4) = p_3 H_3, \end{aligned}$$

where H_3 is often called the Ling carry. H_3 can be further rewritten [17] as:

$$\begin{aligned} H_3 &= (g_3 + g_2) + (p_2 p_1)(g_1 + g_0) + \\ &\quad + (p_2 p_1)(p_0 p_7)(g_7 + g_6) + \\ &\quad + (p_2 p_1)(p_0 p_7)(p_6 p_5)(g_5 + g_4) \end{aligned} \tag{4}$$

Defining the new generate terms $R_i = g_i + g_{i-1}$ and the new propagate terms $Q_i = p_i p_{i-1}$, and using them in (4) the following equation can be derived for H_3

$$H_3 = R_3 + Q_2 R_1 + Q_2 Q_0 R_7 + Q_2 Q_0 Q_6 R_5 \tag{5}$$

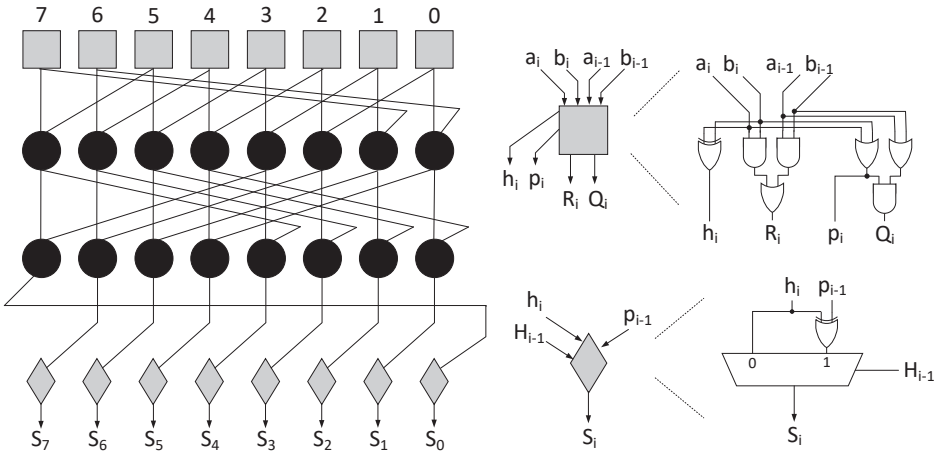

Fig. 4. Ling modulo $2^8 - 1$ parallel-prefix adder

Using the $\circ$ operator in (5), H_3 can also be expressed as the group generate term of:

$$H_3 \longrightarrow (R_3, Q_2) \circ (R_1, Q_0) \circ (R_7, Q_6) \circ (R_5, Q_4)$$

The corresponding expressions for the remaining Ling modulo carries H_i can be derived in a similar manner. In the general case, H_i is the group generate term of

$$H_i \longrightarrow (R_i, Q_{i-1}) \circ (R_{i-2}, Q_{i-3}) \circ \cdots \circ (R_{i+2}, Q_{i+1})$$

Utilizing the above, in [16] and [17] parallel-prefix Ling carry computation units have been proposed for integer and modulo $2^n - 1$ adders, respectively. The parallel-prefix architecture for the Ling modulo $2^8 - 1$ adder proposed in [17] is shown in Fig 4. The preprocessing stage of these adders is more complicated since it also needs to compute the R_i and the Q_i signals. However, these can be derived efficiently by using and-or-invert (AOI) and or-and-invert (OAI) compound gates. The summation stage is actually a multiplexer which uses the Ling carry as the select signal of two precomputed sum bits. Considering that a multiplexer offers almost the same delay as an XOR gate and that both its inputs are available before the selecting Ling carry H_{i-1}, it is evident that this modified summation stage does not impose any delay penalty over the previous cases. On the other hand, significant savings in delay over the previous designs result in the carry computation unit which in this case requires one less prefix level compared to the adders shown in Fig. 3.b.

III. Proposed family of modulo $2^n - 1$ adders

The family of modulo $2^n - 1$ adders presented in this section is based on two extensions of the existing design theory, which are introduced by design examples in the following.

The first is that the factorization considered by Ling is just one of many alternatives. This extension alone is sufficient in providing us with a new architecture that relies on an even simpler carry equation. We introduce this using as an example the c_5^* carry of a modulo $2^8 - 1$ adder, which according to [14] is equal to:

$$\begin{aligned} c_5^* &= g_5 + p_5 g_4 + p_5 p_4 g_3 + p_5 p_4 p_3 g_2 + \\ &\quad + p_5 p_4 p_3 p_2 g_1 + p_5 p_4 p_3 p_2 p_1 g_0 + \\ &\quad + p_5 p_4 p_3 p_2 p_1 p_0 g_7 + p_5 p_4 p_3 p_2 p_1 p_0 p_7 g_6 \end{aligned} \quad (6)$$

Given that $g_i = p_i g_i$, some of its possible factorizations are:

$$\begin{aligned} c_5^* &= p_5(g_5 + g_4 + p_4 g_3 + p_4 p_3 g_2 + p_4 p_3 p_2 g_1 + \\ &\quad + p_4 p_3 p_2 p_1 g_0 + p_4 p_3 p_2 p_1 p_0 g_7 + \\ &\quad + p_4 p_3 p_2 p_1 p_0 p_7 g_6) \quad (7) \\ &= (g_5 + p_5 p_4)(g_5 + g_4 + g_3 + p_3 g_2 + p_3 p_2 g_1 + \\ &\quad + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 g_7 + p_3 p_2 p_1 p_0 p_7 g_6) \quad (8) \\ &= (g_5 + p_5 g_4 + p_5 p_4 p_3)(g_5 + g_4 + g_3 + g_2 + p_2 g_1 + \\ &\quad + p_2 p_1 g_0 + p_2 p_1 p_0 g_7 + p_2 p_1 p_0 p_7 g_6) \quad (9) \end{aligned}$$

Therefore, the adders of [14] can be viewed as architectures in which no factorization is applied to the carry equations. Ling adders [17] correspond to architectures in which only the factorization of (7) is considered. As a result, they reduce the complexity of the carry equations by only a single propagate term.

The adders of the proposed family may employ any alternative factorization for reducing the complexity even more. This is demonstrated by considering the factorization of (9), where three terms are factored out of the carry equation. Let $D_i = (g_i + p_i g_{i-i} + p_i p_{i-1} p_{i-2})$ denote the term inside the first parenthesis of (9). This term can be efficiently computed in the preprocessing stage using a compound AOI gate, with a delay smaller than that of two prefix levels. By grouping the terms in the second parenthesis, we obtain:

$$\begin{aligned} c_5^* &= D_5[(g_5 + g_4) + (g_3 + g_2) + (p_2 p_1)(g_1 + g_0) + \\ &\quad + (p_2 p_1)(p_0 p_7)(g_7 + g_6)] \end{aligned}$$

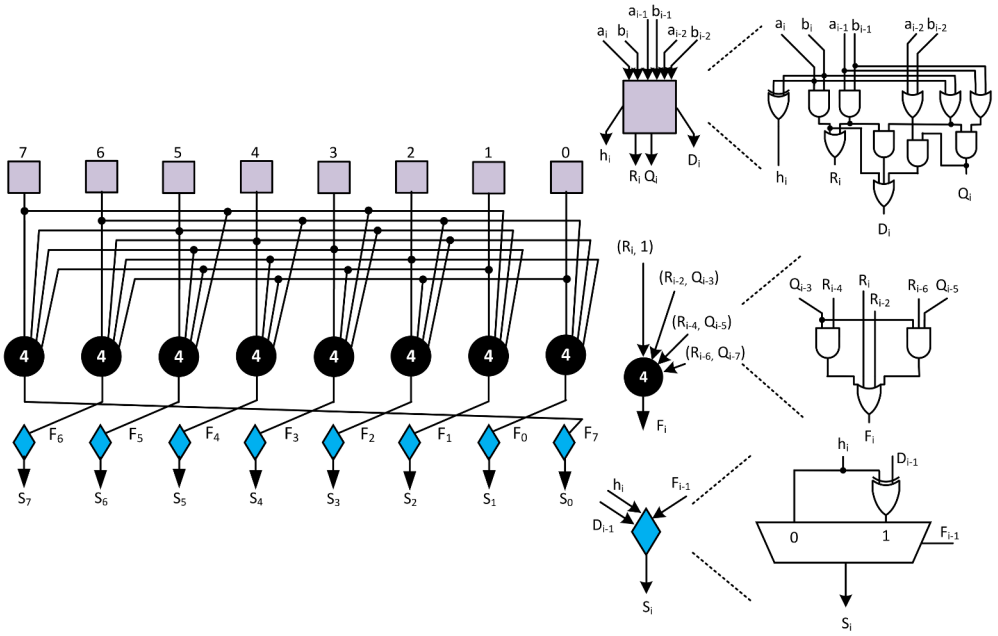

Fig. 5. Modulo $2^8 - 1$ adder based on (9) with valency–4 prefix operators.

Substituting $R_i = g_i + g_{i-1}$, and $Q_i = p_i p_{i-1}$ into the above expression, we obtain:

$$c_5^* = D_5(R_5 + R_3 + Q_2 R_1 + Q_2 Q_0 R_7) = D_5 F_5$$

where F_5 denotes the new simplified form of the carry, and is defined as the group generate of:

$$F_5 \longrightarrow (R_5, 1) \circ (R_3, Q_2) \circ (R_1, Q_0) \circ (R_7, Q_6)$$

The term F_5 can be computed in either one or two prefix levels, depending on whether valency–4 or valency–2 prefix operators are used, respectively. Moreover, since the propagate component of the $(R_5, 1)$ pair is constant, the corresponding prefix operators can be further simplified. Generalizing the above, the carries c_i^* of a modulo $2^8 - 1$ adder that adopts the factorization of (9) and uses valency–4 prefix operators can be computed as:

$$\begin{aligned} c_i^* &= D_i(R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}) \\ &= D_i F_i \end{aligned}$$

Comparing F_i against H_i, it is obvious that due to the greater factorization, its logic equation is simpler and can therefore be computed earlier.

Since D_i is always computed earlier than F_i, additional delay in the summation stage can be avoided by precomputing the two possible sum values, h_i and $h_i \oplus D_{i-1}$. The late-arriving signal F_{i-1} is then used to select the correct result, according to:

$$\begin{aligned} s_i &= h_i \oplus c_{i-1}^* = h_i \oplus (D_{i-1} F_{i-1}) \\ &= F_{i-1}(h_i \oplus D_{i-1}) + \overline{F_{i-1}} h_i \end{aligned}$$

This also dictates that D_i should be computed at least one XOR gate delay earlier than F_{i-1}.

The derived adder is illustrated in Fig. 5. Compared against the architecture of Fig. 4 it uses a more complex preprocessing stage but requires only a single prefix level with simplified valency–4 operators. It is noted that the simplistic unit-gate model [26] provides an equal delay estimation of 8 equivalent gates for both the architectures of Figures 4 and 5. However, such estimations can not be considered fair since they ignore routing and they do not properly account for the area and delay of compound gates available in every CMOS design library.

In the above we considered only the possible factorizations at the initial level of the adder. Our second extension is that factorization should not be constrained just to the initial stage of the adder; it can be performed at every level. We introduce this second extension, using the modulo $2^{16} - 1$ adder as our presentation vehicle and discuss its implications. As an example, let us consider c_7^* of a modulo $2^{16} - 1$ adder in which the factorization of (7) is performed at both the initial and the first prefix level of the adder. We consider that valency–4 and valency–2 operators are used at the first and the second prefix level of the adder, respectively.

The factorization at the initial level leads to:

$$\begin{aligned} c_7^* &= p_7 H_7 = p_7(R_7 + Q_6 R_5 + Q_6 Q_4 R_3 + Q_6 Q_4 Q_2 R_1 \\ &\quad + Q_6 Q_4 Q_2 Q_0 R_{15} + Q_6 Q_4 Q_2 Q_0 Q_{14} R_{13} \\ &\quad + Q_6 Q_4 Q_2 Q_0 Q_{14} Q_{12} R_{11} \\ &\quad + Q_6 Q_4 Q_2 Q_0 Q_{14} Q_{12} Q_{10} R_9) \end{aligned} \quad (10)$$

Since we consider valency–4 operators in the first prefix level, we group the terms of (10) in groups of 4, and by performing

a second factorization of the term $R_7 + Q_6$, we get that (10) is equivalent to:

$$\begin{aligned} c_7^* &= p_7(R_7+Q_6)\Big\{\big[R_7+R_5+Q_4R_3+Q_4Q_2R_1\big]+ \\ &\quad +\big[Q_4Q_2Q_0(R_{15}+Q_{14})\big] \\ &\quad \big[R_{15}+R_{13}+Q_{12}R_{11}+Q_{12}Q_{10}R_9\big]\Big\} \quad (11) \\ &= D_7^1(F_7^1+Q_4^1F_{15}^1) \\ &= D_7^1F_7 \quad (12) \end{aligned}$$

with

$$\begin{aligned} D_7^1 &= p_7(R_7+Q_6), \\ F_7^1 &= R_7+R_5+Q_4R_3+Q_4Q_2R_1, \\ Q_4^1 &= Q_4Q_2Q_0(R_{15}+Q_{14}), \\ F_{15}^1 &= R_{15}+R_{13}+Q_{12}R_{11}+Q_{12}Q_{10}R_9, \text{ and} \\ F_7 &= F_7^1+Q_4^1F_{15}^1. \end{aligned}$$

The superscript in each signal indicates the prefix level in which this is computed (F_7 is also therefore F_7^2). Considering that we opted for the use of valency–4 prefix operators at the first prefix level of the adder, F_{15}^1, F_7^1 are the group generate propagate signals of:

$$\begin{aligned} F_{15}^1 &\longrightarrow (R_{15},1)\circ(R_{13},Q_{12})\circ \\ &\quad \circ(R_{11},Q_{10})\circ(R_9,Q_8(R_7+Q_6)) \quad (13) \\ F_7^1 &\longrightarrow (R_7,1)\circ(R_5,Q_4)\circ \\ &\quad \circ(R_3,Q_2)\circ(R_1,Q_0(R_{15}+Q_{14})) \quad (14) \end{aligned}$$

while Q_{12}^1 and Q_4^1 are the group propagate signals of (13) and (14), respectively. Having them ready from the first prefix level, the carry is then given by:

$$F_7 = F_7^2 \longrightarrow (F_7^1,Q_4^1)\circ(F_{15}^1,Q_{12}^1)$$

that is, it is the group generate of a valency–2 operator of the second prefix level.

Generalizing the above, we can conclude that the carry c_i^* of a modulo $2^{16}-1$ adder which adopts the factorization of one term at the initial and the first prefix level, and which uses valency–4 and valency–2 operators at the first and the second prefix level, respectively, are the group generate signals of:

$$c_i^* = D_i^1F_i^2 \longrightarrow p_i(R_i+Q_{i-1})[(F_i^1,Q_{i-3}^1)\circ(F_{i-8}^1,Q_{i-11}^1)],$$

where:

$$\begin{aligned} F_i^1 &= R_i+R_{i-2}+Q_{i-3}R_{i-4}+Q_{i-3}Q_{i-5}R_{i-6}, \text{ and} \\ Q_i^1 &= Q_iQ_{i-2}Q_{i-4}(R_{i-5}+Q_{i-6}) \end{aligned}$$

Although the recursive factorization of terms at the various prefix levels removes complexity from the carry computation, it also results in increased complexity of the D_i signals. Viewed in a more abstract way, the adders of the proposed family may have two computation trees; that of the D_i and that of the F_i signals. Every factorization makes the F_i tree shallower at the cost of the D_i tree becoming deeper. Under this abstraction, the adders of [14] can be considered as those with a zero–depth D_i tree and those of [17] as those that have a D_i tree of a single gate.

To ensure that no delay is added at the summation stage, the computation of the D_i signals must be complete at least by the delay of an XOR gate earlier than the F_i signals. That is, the D_i tree computation must be shallower than that of the F_i one. Therefore, the factorization as well as the valency of the operators at every prefix level used must be chosen carefully, in order to reach the adder that offers the least delay. The fastest organization also depends on the availability of compound gates in the design library as well as on their delay difference against an XOR gate.

To better illustrate this, Table I presents the computations performed at the prefix levels of two members of the modulo $2^{32}-1$ adder family. Both adders use valency–4 prefix operators at both their prefix levels and use (9) for factorization at the preprocessing level. Compared to Adder 2, Adder 1 uses an extra factorization of a single term at the first prefix level. As a result, the computation of F_i^2 is simpler in Adder 1 than that of Adder 2. Although this allows to use simplified valency–4 operators in both Adder 1 prefix levels, while Adder 2 uses simplified operators only at the first prefix level, Adder 1 requires more delay to compute D_i^1 which in our experiments led to a larger delay than Adder 2. From the above, it becomes obvious that the introduced extensions provide for each wordlength an extended previously unexplored design space. This space needs to be explored for every distinct design library for finding the fastest member of the proposed family. Given the cells available in each library, one has to examine the factorization degree along with the valency of the operators of each level. Such an exploration is presented next.

IV. Comparisons

As explained in Section III, the proposed adder family offers many alternatives for each wordlength, depending on the chosen factorization and the valency of the operators of each prefix level. For determining the fastest possible design at each wordlength, we built a generator that provided all candidate adders built on valency–2 or valency–4 prefix operators. Although all possible factorizations at each level were considered, only those that result in the D_i tree being shallower than the F_i tree were selected. For examining adders with an increasing number of prefix levels, wordlengths of 8, 16, 32 and 64 bits were considered.

Each adder was mapped into a 32nm standard cell technology [27], using the Synopsys® Design Compiler® tool. We assumed that each adder's input and output is driven by the output of a D flip flop and drives the input of a D flip flop of the same implementation library, respectively. A typical corner (1.05V, 25^{o}C) was considered for all designs. For our delay and area results, the mapped designs were recursively optimized for speed until no performance gains were possible. An area recovery step was finally applied. For power estimations an operation frequency of 2 GHz and a switching probability of 0.5 at each input were considered.

TABLE I
ALTERNATIVE MODULO $2^{32}-1$ ADDERS

	Adder 1	Adder 2
Initial level	$R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$	$R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$
Level 1	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$ $D_i^1 = D_i (F_i^1 + Q_{i-3}^1)$	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$
Level 2	$F_i^2 = F_i^1 + F_{i-8}^1 + Q_{i-11}^1 F_{i-16}^1$ $+ Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$	$F_i^2 = F_i^1 + Q_{i-3}^1 F_{i-8}^1 + Q_{i-3}^1 Q_{i-11}^1 F_{i-16}^1$ $+ Q_{i-3}^1 Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$
Summation stage	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}^1) + \overline{F_{i-1}^2} h_i$	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}) + \overline{F_{i-1}^2} h_i$

TABLE II
FASTEST MODULO $2^n - 1$ ADDERS ORGANIZATION AND LOGIC EQUATIONS

	$n = 8$ Organization: $M_4^{(9),(-)}$	$n = 16$ Organization: $M_{4,4}^{(-),(7),(-)}$
Initial level	$h_i = a_i \oplus b_i$ $p_i = a_i + b_i$ $g_i = a_i \cdot b_i$ $R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$	$h_i = a_i \oplus b_i$ $p_i = a_i + b_i$ $g_i = a_i \cdot b_i$
Level 1	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} + Q_{i-5} R_{i-6}$	$F_i^1 = g_i + g_{i-1} + p_{i-1} g_{i-2} + p_{i-1} p_{i-2} g_{i-3}$ $Q_i^1 = p_i p_{i-1} p_{i-2} p_{i-3}$ $D_i^1 = p_i F_i^1 + p_{i-1} Q_i^1$
Level 2		$F_i^2 = F_i^1 + F_{i-4}^1 + Q_{i-5}^1 F_{i-8}^1 + Q_{i-5}^1 Q_{i-9}^1 F_{i-12}^1$
Summation stage	$s_i = F_{i-1}^1 (h_i \oplus D_{i-1}) + \overline{F_{i-1}^1} h_i$	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}^1) + \overline{F_{i-1}^2} h_i$

	$n = 32$ Organization: $M_{4,4}^{(9),(-),(-)}$	$n = 64$ Organization: $M_{4,4,4}^{(-),(7),(7),(-)}$
Initial level	$h_i = a_i \oplus b_i$ $p_i = a_i + b_i$ $g_i = a_i \cdot b_i$ $R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$	$h_i = a_i \oplus b_i$ $p_i = a_i + b_i$ $g_i = a_i \cdot b_i$
Level 1	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$	$F_i^1 = g_i + g_{i-1} + p_{i-1} g_{i-2} + p_{i-1} p_{i-2} g_{i-3}$ $Q_i^1 = p_i p_{i-1} p_{i-2} p_{i-3}$ $D_i^1 = p_i F_i^1 + p_{i-1} Q_i^1$
Level 2	$F_i^2 = F_i^1 + Q_{i-3}^1 F_{i-8}^1 + Q_{i-3}^1 Q_{i-11}^1 F_{i-16}^1$ $+ Q_{i-3}^1 Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$	$F_i^2 = F_i^1 + F_{i-4}^1 + Q_{i-5}^1 F_{i-8}^1 + Q_{i-5}^1 Q_{i-9}^1 F_{i-12}^1$ $Q_i^2 = Q_i^1 Q_{i-4}^1 Q_{i-8}^1 (F_{i-11}^1 + Q_{i-12}^1)$ $D_i^2 = D_i^1 (F_i^2 + Q_{i-5}^2)$
Level 3		$F_i^3 = F_i^2 + F_{i-16}^2 + Q_{i-21}^2 F_{i-32}^2 + Q_{i-21}^2 Q_{i-37}^2 F_{i-48}^2$
Summation stage	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}) + \overline{F_{i-1}^2} h_i$	$s_i = F_{i-1}^3 (h_i \oplus D_{i-1}^2) + \overline{F_{i-1}^3} h_i$

Table II lists the family members that led to the best delay results, along with the logic equations that are implemented at each level, for $n = 8, 16, 32$, or 64. The adder organization is denoted by $M_{x,y,\ldots}^{a,b,\cdots}$, where the superscript indicates the factorization used in the initial and the subsequent levels of the adder, whereas the subscript indicates the valency of the operators used in each prefix level. A $(-)$ indicates that no factorization is performed at a certain level. a For example, the organization $M_{4,4,4}^{(-),(7),(7),(-)}$ that led to the best delay at $n = 64$ denotes an adder with three valency–4 prefix levels and in which a single term is factored at the first and the second prefix level of the adder, according to (7).

Table III lists our attained results of the fastest adders that our design space exploration provided, against those of the proposals of [14] and [17]. Delay results (T) are given in ps, area results (A) in μm^2, and average power estimations (P) in μW. The delay savings that were attained compared against the adders proposed in [14] range from 9.8% up to 12.3%. The savings offered against the adders of [17] range from 5.9% up to 6.1%. The results reveal that the introduced extensions

TABLE III
DELAY (T) IN PS, AREA (A) IN μm^2 AND POWER (P) IN μW RESULTS

		[14]	[17]	Proposed
$n = 8$	Delay (T)	307	286	269
	Area (A)	244.9	265.5	261.3
	Power (P)	27.3	26.4	26.2
$n = 16$	Delay (T)	341	313	294
	Area (A)	873.4	941.2	938.4
	Power (P)	102.7	104.2	97.9
$n = 32$	Delay (T)	401	381	355
	Area (A)	2150.2	2103.9	2168.3
	Power (P)	296.6	275.4	263.0
$n = 64$	Delay (T)	458	440	413
	Area (A)	6124.1	6150.5	6311.2
	Power (P)	867.8	844.4	807.5

TABLE IV
AREA x DELAY^2 NORMALIZED RESULTS

(n)	[14]	[17]	Proposed
8	1.220	1.148	1.000
16	1.252	1.137	1.000
32	1.265	1.107	1.000
64	1.193	1.106	1.000

enabled us to derive faster adders than the already proposed in every examined wordlength case.

In Table IV the well-known A $\times$ T^2 product is used as a metric. It lists A $\times$ T^2 results normalized over those offered by the fastest adders. Savings of 19.3% up to 26.5% are offered against the adders of [14]. The savings against the adders of [17] range from 10.6% up to 14.8%.

V. CONCLUSIONS

In this paper several extensions of the already developed theory on modulo $2^n - 1$ parallel-prefix adder design were introduced; namely, the use of higher valency operators along with alternative and more aggressive factorizations that can be performed in every prefix level and not only at the pre-processing stage of the adder. Based on them, a large family of previously unexplored adders was attained. Each adder of this family has its own factorization degree and prefix operator valency at each prefix level. The previously reported parallel-prefix modulo 2^n-1 adders can be considered as special cases of this adder family.

By exploring this extended design space in a specific design library for the most commonly used adder wordlengths, we were able to derive parallel-prefix modulo $2^n - 1$ adders that are faster than all previously reported ones by at least 5.9%, and also more efficient under the area x delay^2 metric by at least 10.6%.

REFERENCES

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

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

[3] G. N. Jyothi, K. Sanapala, and A. Vijayalakshmi, "ASIC Implementation of Distributed Arithmetic Based FIR Filter using RNS for High Speed DSP Systems," *International Journal of Speech Technology*, vol. 23, pp. 259–264, June 2020.

[4] G. Valuev, *et al.*, "Digital Filter Architecture Based on Modified Winograd Method $F(2 \times 2, 5 \times 5)$ and Residue Number System," *IEEE Access*, vol. 11, pp. 26 807–26 819, 2023.

[5] E. Vassalos, D. Bakalis, and H. T. Vergos, "RNS Assisted Image Filtering and Edge Detection," in *Proceedings of the 18th IEEE International Conference on Digital Signal Processing (DSP)*, Santorini, Greece, July 2013, pp. 1–6.

[6] S. Salamat *et al.*, "RNSnet: In-Memory Neural Network Acceleration Using Residue Number System," in *Proceedings of the IEEE International Conference on Rebooting Computing (ICRC)*, 2018, pp. 1–12.

[7] N. Samimi *et al.*, "Res-DNN: A Residue Number System-Based DNN Accelerator Unit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 2, pp. 658–671, 2020.

[8] V. Sakellariou *et al.*, "On Reducing the Number of Multiplications in RNS-based CNN Accelerators," in *Proceedings of the 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021, pp. 1–6.

[9] A. Roohi *et al.*, "RNSiM: Efficient Deep Neural Network Accelerator Using Residue Number Systems," in *Proceedings of the 2021 IEEE/ACM Int. Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[10] H. Nakahara and T. Sasao, "A High-Speed Low-Power Deep Neural Network on an FPGA based on the Nested RNS: Applied to an Object Detector," in *Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

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

[12] S.-H. Lin and M.-H. Sheu, "Area-Time Efficient Modulo $2^n - 1$ Adder Design using Hybrid Carry Selection," *IEICE Transactions on Information and Systems*, vol. E91-D, no. 2, pp. 361–362, 2008.

[13] R. Zimmermann, "Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication," in *Proceedings of the 14th IEEE Symposium on Computer Arithmetic (ARITH-14)*, 1999, pp. 158–167.

[14] L. Kalampoukas *et al.*, "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 673–680, 2000.

[15] H. Ling, "High-Speed Binary Adder," *IBM Journal of Research and Development*, vol. 25, no. 3, pp. 156–166, 1981.

[16] G. Dimitrakopoulos and D. Nikolos, "High-Speed Parallel-Prefix VLSI Ling Adders," *IEEE Transactions on Computers*, vol. 54, no. 2, pp. 225–231, 2005.

[17] G. Dimitrakopoulos *et al.*, "New Architectures for Modulo $2^N - 1$ Adders," in *Proceedings of the 12th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2005, pp. 1–4.

[18] R. Jackson and S. Talwar, "High Speed Binary Addition," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, 2004, pp. 1350–1353 Vol.2.

[19] Brent and Kung, "A Regular Layout for Parallel Adders," *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260–264, 1982.

[20] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, 1973.

[21] R. E. Ladner and M. J. Fischer, "Parallel Prefix Computation," *Journal of the ACM*, vol. 27, no. 4, pp. 831–838, 1980.

[22] A. Beaumont-Smith and C.-C. Lim, "Parallel prefix adder design," in *Proceedings 15th IEEE Symposium on Computer Arithmetic (ARITH-15)*, 2001, pp. 218–225.

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

[24] G. Jaberipur and B. Parhami, "Unified Approach to the Design of Modulo-$(2^n \pm 1)$ Adders Based on Signed-LSB Representation of Residues," in *Proceedings 19th IEEE Symposium on Computer Arithmetic (ARITH-19)*, 2009, pp. 57–64.

[25] J. J. Shedletsky, "Comment on the sequential and indeterminate behavior of an end-around-carry adder," *IEEE Transactions on Computers*, vol. C-26, no. 3, pp. 271–272, 1977.

[26] A. Tyagi, "A Reduced-Area Scheme for Carry-Select Adders," *IEEE Transactions on Computers*, vol. 42, no. 10, pp. 1163–1170, 1993.

[27] Synopsys Inc., "SAED 32nm EDK," Available: https://www.synopsys.com/apps/protected/university/members.html.