Fast Parallel-Prefix Modulo $2^n + 1$ Adders

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

Abstract—Modulo $2^n + 1$ adders find great applicability in several applications including RNS implementations and cryptography. In this paper, we present two novel architectures for designing modulo $2^n + 1$ adders, based on parallel-prefix carry computation units. The first architecture utilizes a fast carry increment stage, whereas the second is a totally parallel-prefix solution. CMOS implementations reveal the superiority of the resulting adders against previously reported solutions in terms of implementation area and execution latency.

Index Terms—Binary adders, modulo $2^n + 1$ arithmetic, parallel-prefix adders, RNS.

1 Introduction

SEVERAL applications whose arithmetic operations are limited to addition, subtraction, and multiplication profit from the use of a Residue Number System (RNS) [1], [2]. Such applications include the design of Number Theoretic Transform, Discrete Fourier Transform, Discrete Cosine Transform, digital filters [2], [3], [4], [5], [6], [7], and communication components [8], [9].

An RNS is defined by a set of L moduli, suppose $\{d_1, d_2, \ldots, d_L\}$, that are pair-wise relative prime. Assume that $|X|_d$ denotes the modulo d of X, that is, the least nonnegative remainder of the division of X by d. Then, an integer X has a unique representation in the RNS, given by the set $(x_1, x_2, \ldots, x_L)$ of residues, where $x_i = |X|_{d_i}$ if $X \geq 0$ and $x_i = |D + X|_{d_i}$ if $X < 0$, with $D = d_1 \times d_2 \times \ldots \times d_L$. An RNS operation $\diamond$ is defined as $(z_1, z_2, \ldots, z_L) = (x_1, x_2, \ldots, x_L) \diamond (y_1, y_2, \ldots, y_L)$, where $z_i = |x_i \diamond y_i|_{d_i}$. Since the computation of z_i only depends on x_i, y_i, and d_i, each z_i is computed in parallel in a separate arithmetic unit, often called *channel*.

Moduli choices of the form $\{2^n, 2^n - 1, 2^n + 1\}$ have received significant attention because they offer very efficient implementations in the $\text{area} \times \text{time}^2$ product sense [10]. Addition in such systems is performed using three channels that, in fact, are a modulo $2^n - 1$ (equivalently one's complement), a modulo 2^n, and a modulo $2^n + 1$ adder. Thus, the design of an efficient modulo $2^n + 1$ adder is a vital task in RNS-based applications that use a channel of the $2^n + 1$ form.

Modulo $2^n + 1$ adders are also utilized as the last stage adder of modulo $2^n + 1$ multipliers. Modulo $2^n + 1$ multipliers find great applicability in pseudorandom number generation, cryptography [11], [12], [13], and in the Fermat number transform, which is an effective way to compute convolutions [14].

The addition delay in an RNS application which uses the $\{2^n, 2^n - 1, 2^n + 1\}$ moduli, is determined by the modulo $2^n + 1$ channel, since this handles $(n+1)$-bit wide operands. Therefore, the design of fast modulo $2^n + 1$ adders is of great significance. To overcome the problem of $(n+1)$-bit wide circuits for the modulo $2^n + 1$ channel, the diminished-one number system [15] has been proposed. Efficient adders with operands represented in this system have appeared in [16], [17], [18], [19]. However, its use rises new problems. One is the special treatment required for operands equal to 0. Another one is the requirement for converters from/to the normal to/from diminished-one representation that increases the execution latency and the required implementation area.

Modulo $2^n + 1$ adders for the normal binary number system can be derived by considering them as a special case of modulo m adders. The most efficient solutions for generalized modulo adders are reported in [20], [21], [22]. The solution proposed in [22] is based on a Carry Save Adder (CSA) and on the multiplexing of the carry generate (g) and propagate (p) signals before being driven to the carry computation unit. The architecture proposed in [22] has been shown experimentally to outperform the solutions proposed in [20] and [21] in both area and delay, and will therefore be used in our comparisons.

In this paper, we derive two novel architectures for modulo $2^n + 1$ adders, with operands represented in the normal binary number system. Both architectures are based on using parallel-prefix carry computation schemes. The first architecture utilizes a fast carry increment stage, according to [23]. The second eliminates the need for this separate carry increment stage by performing carry recirculation at each prefix level [19], [24]. We will hereafter refer to these architectures as PPFCI (Parallel-Prefix with Fast Carry Increment) and TPP (Totally Parallel-Prefix), respectively. Using implementations in a full static CMOS VLSI technology, we show that both architectures outperform the previously reported ones in both execution latency and implementation area.

2 Fast Addition Basics

For speeding up the addition operation, the carry computation time should be minimized. In the following, we use the notations $\wedge, \vee, \oplus$, and $^-$ to denote AND, inclusive-OR, exclusive-OR, and complement operations, respectively. Assuming the addition of $A = a_{n-1}a_{n-2}\ldots a_0$ with $B = b_{n-1}b_{n-2}\ldots b_0$, two terms are commonly used for describing the carry computation problem: the carry generate term, g, and the carry propagate term, p. These terms are derived for every i, with $0 \leq i \leq (n-1)$ as $g_i = a_i \wedge b_i$ and $p_i = a_i \oplus b_i$. The bits of the sum $R = r_{n-1}r_{n-2}\ldots r_0$ are computed according to $r_i = p_i \oplus c_{i-1}$, where c is the carry signal. The carry at each bit position can be derived according to the well-known recursive equation $c_i = g_i \vee p_i \wedge c_{i-1}$. By unfolding the latter equation and implementing in parallel the resulting equations, the well-known carry-look ahead (CLA) adders result.

A special category of CLA adders, well-known as parallel-prefix adders, results by considering the carry computation in binary addition as a prefix problem [25]. Carry computation is transformed into a prefix computation by the introduction of the associative operator $\circ$, defined in [26] as:

$$(g_m, p_m) \circ (g_k, p_k) = (g_m \vee (p_m \wedge g_k), p_m \wedge p_k). \quad (1)$$

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

$$(G_i, P_i) = \begin{cases} (g_0, p_0), & \text{if } i = 0 \\ (g_i, p_i) \circ (G_{i-1}, P_{i-1}), & \text{if } 1 \leq i \leq n-1. \end{cases} \quad (2)$$

The association of two pairs of (g_x, p_x) terms using the $\circ$ operator is usually represented as a node and a whole carry computation unit is represented as a tree structured interconnection of such nodes. Several tree structures have been proposed [25], [26], [27], [28]. Both the Ladner-Fischer [25] and the Kogge-Stone [27] parallel-prefix adders offer the minimal logical depth property, that is, the prefix levels that they require are $\log_2 n$. Ladner-Fischer adders, however, require significantly less implementation area at the expense of a fan-out loading equal to the

- *C. Efstathiou is with the Informatics Department, TEI of Athens, Ag. Spyridonos St., 12210 Egaleo, Athens, Greece. E-mail: cefsta@teiath.gr.*
- *H.T. Vergos and D. Nikolos are with the Computer Engineering and Informatics Department, Patras University, 26500, Greece, and the Computer Technology Institute, 3 Kolokotroni St., 26221 Patras, Greece. E-mail: vergos@ceid.upatras.gr, nikolosd@cti.gr.*

Manuscript received 30 Mar. 2003; revised 6 Mar. 2004; accepted 2 Apr. 2004.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 118474.

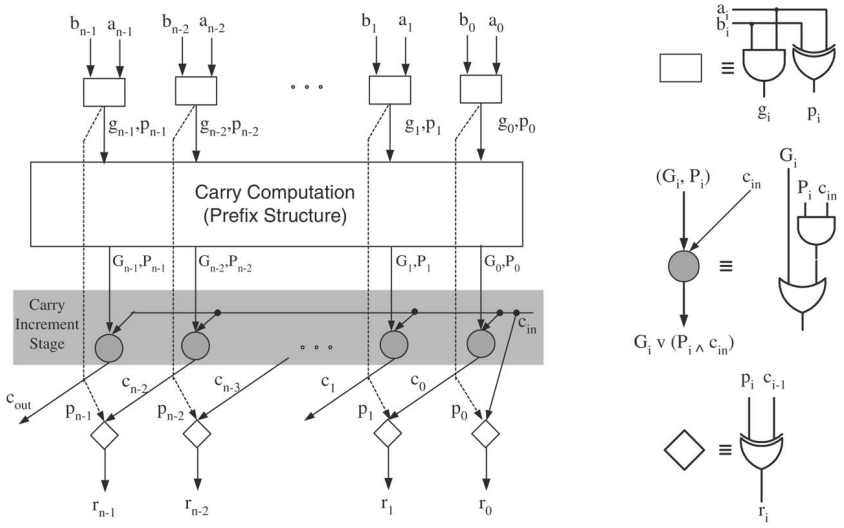

Fig. 1. Prefix adder structure with carry input.

operand length compared to Kogge-Stone adders that offer a fan-out of 2. Knowles in [28] examines combinations of the algorithms presented in [25], [27] and presents possible trade offs of fan-out and implementation area.

The addition of a carry input to a parallel-prefix adder can be achieved by adding to it a single row of n prefix operators [23]. We will call this extra row the carry increment stage. Fig. 1 presents the resulting structure.

3 Novel Architectures for Modulo $2^n + 1$ Addition

In this section, we propose two architectures for modulo $2^n + 1$ addition that are based on the following theorem:

Theorem 1. *Let X and Y denote two $(n+1)$-bit binary numbers in the range $[0, 2^n + 1)$. Then,*

$$|X+Y|_{2^n+1} = \begin{cases} \left| |X+Y+2^n-1|_{2^{n+1}} + 2^n + 1 \right|_{2^{n+1}}, & \text{if } X+Y+2^n-1 < 2^{n+1} \\ |X+Y+2^n-1|_{2^{n+1}}, & \text{otherwise.} \end{cases}$$

Proof.

$$\begin{aligned} |X+Y|_{2^n+1} &= \begin{cases} X+Y, & \text{if } X+Y < (2^n+1) \\ X+Y-(2^n+1), & \text{otherwise.} \end{cases} \\ &= \begin{cases} |X+Y+2^{n+1}|_{2^{n+1}}, & \text{if } X+Y \\ & -(2^n+1) < 0 \\ X+Y+2^{n+1}-(2^n+1)-2^{n+1}, & \text{otherwise.} \end{cases} \\ &= \begin{cases} |X+Y+2^n-1+2^n+1|_{2^{n+1}}, & \text{if } X+Y+2^{n+1} \\ & -(2^n+1) < 2^{n+1} \\ X+Y+2^n-1-2^{n+1}, & \text{otherwise.} \end{cases} \end{aligned} \tag{3}$$

Since $X, Y \in [0, 2^n + 1)$, we have that

$$X+Y+2^n-1-2^{n+1} \leq 2^n + 2^n + 2^n - 1 - 2^{n+1} = 2^n - 1 < 2^{n+1},$$

therefore,

$$X+Y+2^n-1-2^{n+1} = |X+Y+2^n-1|_{2^{n+1}}.$$

Then, from (3), we get:

$$|X+Y|_{2^n+1} = \begin{cases} \left| |X+Y+2^n-1|_{2^{n+1}} + 2^n + 1 \right|_{2^{n+1}}, & \text{if } X+Y+2^n-1 < 2^{n+1} \\ |X+Y+2^n-1|_{2^{n+1}}, & \text{otherwise.} \end{cases}$$

□

Theorem 1 reveals that a two-stage combinational circuit can be utilized for the modulo addition. The first stage computes an intermediate sum $M = X + Y + 2^n - 1$. Since $M \in [2^n - 1, 2^{n+1} + 2^n - 1]$, it has a $(n+2)$ bit binary representation, suppose $m_{n+1}m_n \ldots m_1 m_0$. If $M < 2^{n+1}$ or, equivalently, if the most significant bit of M, $m_{n+1} = 0$, the term $2^n + 1$ is added modulo 2^{n+1} to the n least significant bits of M by the second stage.

For computing M, a CSA that outputs a carry vector, C, and a sum vector, S, followed by an $(n+1)$-bit parallel adder, which performs $2 \times C + S$, can be utilized. Since $2^n - 1$ in its $(n+1)$-bit binary representation has a 0 at its leftmost position and 1s at all the other positions, the CSA is composed of one half-adder (HA) at its leftmost position and n semihalf-adders (HA*) [22]. An HA* is equivalent to a full-adder, with one of its inputs driven at 1.

3.1 The Proposed Parallel-Prefix with Fast Carry Increment (PPFCI) Architecture

Our first proposed architecture is based on the adder architecture of Fig. 1. For using it for modulo $2^n + 1$ addition, in the following, we introduce several modifications.

The $(n+1)$-bit parallel-adder that computes $2 \times C + S$, can be viewed as a module composed of an n-bit adder for the least significant bits and an exclusive-NOR gate. This gate receives the carry output, suppose c_{out}, of the n-bit adder and the most significant bit, c_n, of the carry vector and produces $\overline{m_{n+1}}$. Note that c_n is equal to 1, when both x_n and y_n are equal to 1. In this case, the n-bit adder computes $(X - 2^n) + (Y - 2^n) + 2^{n+1} - (2^n + 1) = X + Y - (2^n + 1)$. However, considering that $X, Y < (2^n + 1), X + Y - (2^n + 1) < (2^n + 1) < 2^{n+1}$. Therefore, c_{out} cannot, in this case, be equal to 1. We therefore conclude that c_{out} and c_n cannot be simultaneously equal to 1 and the exclusive-NOR gate can be replaced equivalently by an NOR gate.

To further increase the speed of the PPFCI architecture, we note that in the $(n+1)$-bit parallel adder, we have $b_0 = 0$ and, therefore, $g_0 = 0$ and $p_0 = s_0$. We can therefore use an n-bit parallel-prefix

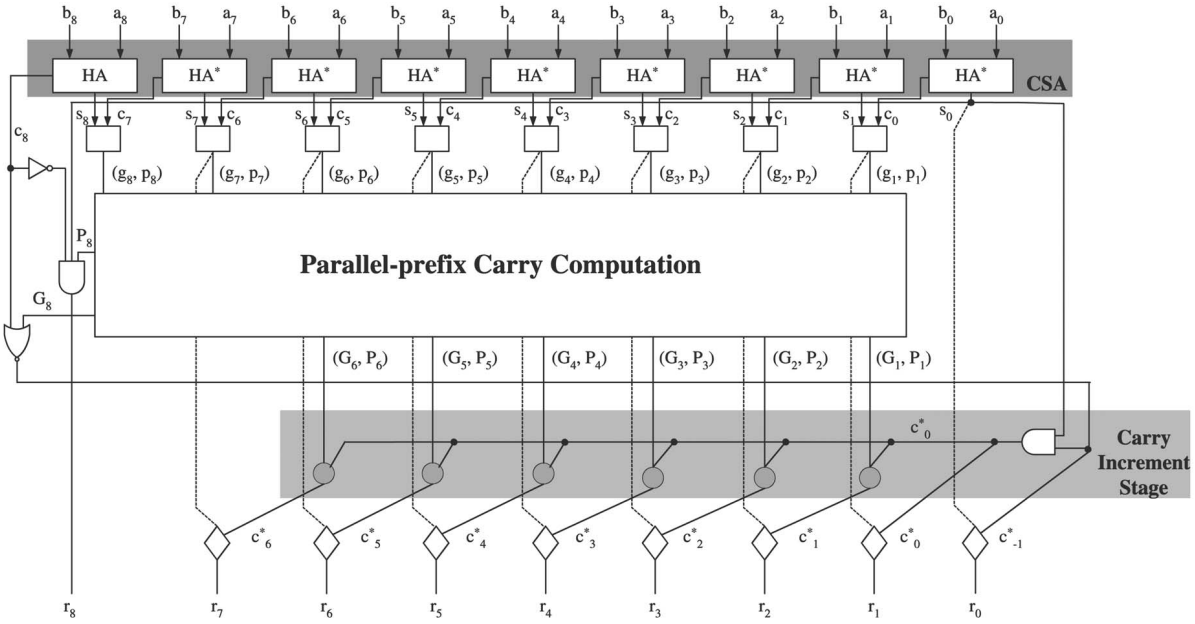

Fig. 2. PPFCI Modulo 2^8+1 adder.

adder with a carry input instead. The carry computation unit of this adder produces:

$$\begin{aligned}(G_1,P_1) &= (g_1,p_1)\\ (G_2,P_2) &= (g_2,p_2)\circ(g_1,p_1)\\ &\cdots\\ (G_n,P_n) &= (g_n,p_n)\circ(g_{n-1},p_{n-1})\circ\ldots\circ(g_1,p_1)\end{aligned}$$

and small modifications in the carry increment stage are required. Specifically, supposing that the carries of the modulo (2^n+1) addition are denoted by c_i^*, we have that:

$$\begin{aligned}c_{-1}^* &= c_{in}\\ c_0^* &= g_0\vee(p_0\wedge c_{-1}^*) = s_0\wedge c_{in}\\ c_i^* &= G_i\vee(P_i\wedge c_0^*),\end{aligned}$$

which means that an AND gate is required for producing c_0^*, while the rest operators of the carry increment stage (see Fig. 1) now receive c_0^*, instead of c_{in}.

Finally, we observe that the conditional addition of (2^n+1) when $m_{n+1}=0$, is equivalent to the addition of $2^n\times\overline{m_{n+1}}+\overline{m_{n+1}}$. The addition of $\overline{m_{n+1}}$ can be performed directly by connecting it to the carry input of the carry increment stage. The addition of the term $2^n\times\overline{m_{n+1}}$ can be omitted since this only affects the most significant bit, r_n, of the result. Taking into account that r_n should be set to 1 if $X+Y=2^n$ or, equivalently, if $M=2^{n+1}-1$, we conclude that r_n can be computed straightforwardly as $\overline{c_n}\wedge P_n\wedge s_0$.

Fig. 2 presents an example of the proposed PPFCI architecture for the case that $n=8$.

3.2 The Proposed Totally Parallel Prefix (TPP) Architecture

Instead of having a dedicated single stage for the reentering carry, it has been proposed in [24], [19] to perform carry recirculation at each existing prefix level. In this way, there is no need for the extra carry increment stage. As a result, dedicated totally parallel-prefix adder architectures are derived with one less prefix level, compared to those derived by the PPFCI architecture.

In our case, in which the reentering carry is given by the expression $\overline{c_n\vee G_n}$, following the procedure of [24], we can derive that the carries c_i^* of the modulo 2^n+1 addition are equal to G_i^*, where G_i^* is computed by the prefix equations:

$$(G_i^*,P_i^*) = \begin{cases}\overline{((g_n\vee c_n,p_n)\circ(G_{n-1,1},P_{n-1,1}))}, & \text{if } i=-1\\ s_0\wedge(G_{-1}^*,P_{-1}^*), & \text{if } i=0\\ (G_{i,1},P_{i,1})\circ s_0\wedge\overline{(G_{n-1,i+1},P_{n-1,i+1})}, & \text{if } 1\le i\le(n-2),\end{cases}\quad(4)$$

where:

- $\overline{(G,P)}=(\overline{G},P)$.
- $G_{a,b}$ and $P_{a,b}$, with $a>b$, are respectively the group generate and propagate signals for the group $a,a-1,a-2,\ldots,b+1,b$, computed by:

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

- $s\wedge(G,P)=(s\wedge G,P)$.

In several cases, the equations indicated by (4) require more than $\log_2 n$ prefix levels for their implementation. These equations can be transformed into equivalent ones that can be implemented within $\log_2 n$ prefix levels. The transformation required uses Theorem 2 of [19], as well as the new Theorem 2 that we will introduce below. For the sake of completeness, we present the main idea of the theory in [19] below. Let t denote the inclusive-OR of the corresponding bits of the two addition operands. If $(G^x,P^x)=(g,p)\circ\overline{(G,P)}$ and $(G^y,P^y)=(\overline{(\overline{t},\overline{g})\circ(G,P)})$, since

$$\begin{aligned}G^x &= g\vee p\wedge\overline{G}=\overline{\overline{(g\vee p\wedge\overline{G})}}=\overline{\overline{g}\wedge(\overline{p}\vee G)}=\overline{(\overline{g}\wedge\overline{p}\vee(\overline{g}\wedge G))}\\ &=\overline{(\overline{t}\vee\overline{g}\wedge G)}\end{aligned}$$

and $G^y=\overline{(\overline{t}\vee\overline{g}\wedge G)}$, we get that $G^x=G^y$. This implies that a carry equal to the generate term which is expressed by a prefix equation of the form $(g,p)\circ\overline{(G,P)}$ is also equal to the generate term of an equation of the form $\overline{(\overline{(\overline{t},\overline{g})\circ(G,P)})}$.

For expressing the terms that have the form $(g_1,p_1)\circ(s_0\wedge(G,P))$ in prefix notation the following theorem is also required:

Theorem 2. *If* $(G^z,P^z)=(g_1,p_1)\circ s_0\wedge\overline{(G,P)}$ *and* $(G^w,P^w)=\overline{(\overline{T_1},\overline{g_1},)}\circ(G,P)$, *where* $T_1=s_1\vee(a_0\wedge b_0)$, *then* $G^z=G^w$.

Proof. Since

$$(g_1,p_1)\circ s_0\wedge\overline{(G,P)}=(g_1,p_1)\circ s_0\wedge(\overline{G},P)=(g_1,p_1)\circ(s_0\wedge\overline{G},P),$$

we have that:

$$\begin{aligned}
G^z &= g_1 \vee (p_1 \wedge s_0 \wedge \overline{G}) = \\
&= \overline{\overline{g_1 \vee (p_1 \wedge s_0 \wedge \overline{G})}} = \\
&= \overline{(\overline{g_1} \wedge (\overline{p_1} \vee \overline{s_0} \vee G))} = \\
&= \overline{((\overline{g_1} \wedge \overline{p_1}) \vee (\overline{g_1} \wedge \overline{s_0}) \vee (\overline{g_1} \wedge G))} = \\
&= \overline{(\overline{t_1} \vee (\overline{g_1} \wedge \overline{s_0}) \vee (\overline{g_1} \wedge G))} = \\
&= \overline{(\overline{t_1} \vee \overline{(g_1 \vee s_0)} \vee (\overline{g_1} \wedge G))} = \\
&= \overline{\overline{((s_1 \vee c_0) \vee \overline{((s_1 \wedge c_0) \vee s_0)} \vee (\overline{g_1} \wedge G))}} = \\
&= \overline{\overline{((s_1 \vee c_0) \vee \overline{((s_1 \vee s_0) \wedge (c_0 \vee s_0))} \vee (\overline{g_1} \wedge G))}} = \\
&= \overline{\overline{((s_1 \vee c_0) \vee \overline{(s_1 \vee s_0)} \vee (\overline{g_1} \wedge G))}} = \\
&= \overline{\overline{((s_1 \vee c_0) \wedge (s_1 \vee s_0)} \vee (\overline{g_1} \wedge G))} = \\
&= \overline{\overline{((s_1 \vee (c_0 \wedge s_0))} \vee (\overline{g_1} \wedge G))} = \\
&= \overline{\overline{((s_1 \vee (a_0 \wedge b_0))} \vee (\overline{g_1} \wedge G))} = \\
&= \overline{(\overline{T_1} \vee (\overline{g_1} \wedge G))}, \text{ where } T_1 = s_1 \vee (a_0 \wedge b_0).
\end{aligned} \tag{5}$$

The latter is the logic equation of G^w. □

As shown in [19] for area-time efficient implementations, the above transformations should be applied j times (the introduced Theorem 2 is applied once and Theorem 2 of [19] $j-1$ times) to those equations of (4) that have the form $(g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \cdots \circ (g_1, p_1) \circ \overline{(G_{n,i+1}, P_{n,i+1})}$. j should be such that:

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

The carry equations resulting from the above modification can be implemented by a prefix structure that has $\log_2 n$ levels. The last (the closest to the outputs of the adder) prefix level requires $n-1$ prefix operators, whereas each prefix level i, with $1 \leq i \leq \log_2 n - 1$, requires $\frac{3}{2}n - 2^i$ prefix operators. The total number of prefix operators required by the TPP architecture is $\frac{3}{2}n(\log_2 n - 1) + 1$.

Example 1. For the implementation of a modulo 257 TPP adder, (4) provides us with the following set of prefix equations:

$$\begin{aligned}
c^*_{-1} &= \overline{((g_8 \vee c_8, p_8) \circ (g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1))} \\
c^*_0 &= s_0 \wedge c^*_{-1} \\
c^*_1 &= (g_1, p_1) \circ (s_0 \wedge \overline{((g_8 \vee c_8, p_8) \circ (g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2))}) \\
c^*_2 &= (g_2, p_2) \circ (g_1, p_1) \circ (s_0 \wedge \overline{((g_8 \vee c_8, p_8) \circ (g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3))}) \\
c^*_3 &= (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ (s_0 \wedge \overline{((g_8 \vee c_8, p_8) \circ (g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4))}) \\
c^*_4 &= (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ (s_0 \wedge \overline{((g_8 \vee c_8, p_8) \circ (g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5))}) \\
c^*_5 &= (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ (s_0 \wedge \overline{((g_8 \vee c_8, p_8) \circ (g_7, p_7) \circ (g_6, p_6))}) \\
c^*_6 &= (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ (s_0 \wedge \overline{((g_8 \vee c_8, p_8) \circ (g_7, p_7))}).
\end{aligned}$$

A parallel prefix carry computation unit that requires only three prefix levels can be derived by transforming the equations describing c^*_1, c^*_2, c^*_3, c^*_5, and c^*_6 into the following:

$$\begin{aligned}
c^*_1 &= \overline{((\overline{t_1}, \overline{g_1}) \circ (g_8 \vee c_8, p_8) \circ (g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2))} \\
c^*_2 &= \overline{((\overline{t_2}, \overline{g_2}) \circ (\overline{T_1}, \overline{g_1}) \circ (g_8 \vee c_8, p_8) \circ (g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3))} \\
c^*_3 &= \overline{((\overline{t_3}, \overline{g_3}) \circ (\overline{t_2}, \overline{g_2}) \circ (\overline{T_1}, \overline{g_1}) \circ (g_8 \vee c_8, p_8) \circ (g_7, p_7) \circ (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4))} \\
c^*_5 &= (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ (g_2, p_2) \circ \overline{((\overline{T_1}, \overline{g_1}) \circ (g_8 \vee c_8, p_8) \circ (g_7, p_7) \circ (g_6, p_6))} \\
c^*_6 &= (g_6, p_6) \circ (g_5, p_5) \circ (g_4, p_4) \circ (g_3, p_3) \circ \overline{((\overline{t_2}, \overline{g_2}) \circ (\overline{T_1}, \overline{g_1}) \circ (g_8 \vee c_8, p_8) \circ (g_7, p_7))}.
\end{aligned}$$

The attained TPP implementation for the modulo $2^8 + 1$ adder case is given in Fig. 3.

4 Comparisons

In this section, we compare the proposed PPFCI and TPP adders against those designed according to [22]. We remind that the architecture proposed in [22] was shown to outperform those of [20] and [21] in terms of implementation area and execution latency. In the case of the PPFCI architectures, a Ladner-Fischer [25] parallel-prefix carry computation scheme is adopted. We assume modulo adders, with n = 4, 8, and 16. All architectures were described in HDL and mapped to the UMC-VST 25 implementation technology (0.25 μm, up to 5-metal layers, $1.8/3.3$V), using the Design Compiler® tool set of Synopsys®. All area implementation results are given in μm^2, while all propagation delay results are in ns.

For evaluating the speed efficiency of each architecture, each mapped design was recursively optimized for speed until the tool was unable to produce a faster design. Area recovery steps followed. This methodology leads to the fastest designs attained by each architecture. The obtained results are listed in Table 1. As we can see both proposed architectures lead to faster implementations than that of [22]. This is due to the fact that the architecture of [22] requires a delay of two CLA computation units connected in series. The proposed architectures, on the other hand, rely on a single carry computation unit. As the adder size increases, the speed advantage of the proposed architectures also increases because the delay of each CLA computation unit required in the architecture of [22] increases as well. The TPP architecture offers the fastest designs reported in the open literature for modulo $2^n + 1$ adders that use the normal binary number system. It is faster than the PPFCI architecture because it is composed of one less prefix level and because of the increased fan-out requirements of the reentering carry in the PPFCI architecture. Note that the latter is also the reason that the speed gap between the TPP and the PPFCI architectures increases along with the adder size.

For comparing the area complexities of the different architectures, we optimized all designs until they reached the same operating speed. We considered as targets, the latencies offered by the architecture of [22]. Then, recursive area optimization steps were applied to the designs, until the tool was unable to provide a smaller implementation. This procedure leads to the smallest adder provided by each architecture that is capable to operate at the target speed. The attained results are indicated in Table 2. The proposed PPFCI and TPP adders are far more area efficient than those of [22]. As n increases both proposed architectures become more attractive.

We also have to compare the proposed adders against modulo $2^n + 1$ adders with operands in diminished-one representation. A TPP adder has the same prefix levels as the adders proposed in [19], without requiring converter circuits to and from the normal binary system, which increase both the execution latency and the implementation area. Moreover, no special treatment is required for zero operands. Therefore, the proposed TPP adders are more efficient than the fastest modulo $2^n + 1$ adders which handle operands in diminished-one representation.

Finally, it is useful to compare the execution latency of the proposed adders against modulo 2^n and modulo $2^n - 1$ adders, since their speed differences will determine the execution speed in an RNS implementation that adopts the three moduli set $\{2^n, 2^n - 1, 2^n + 1\}$. For this comparison, we adopt the unit gate model [29]. This model assumes that each gate excluding exclusive-OR gates has a delay of 1. Exclusive-OR gates have a delay of 2. The execution latencies of the fastest reported modulo 2^n [25], [27], [28] and modulo $2^n - 1$ [24] adders, according to the considered model, are equal to $(2 \times \log_2 n + 3)$. The proposed PPFCI

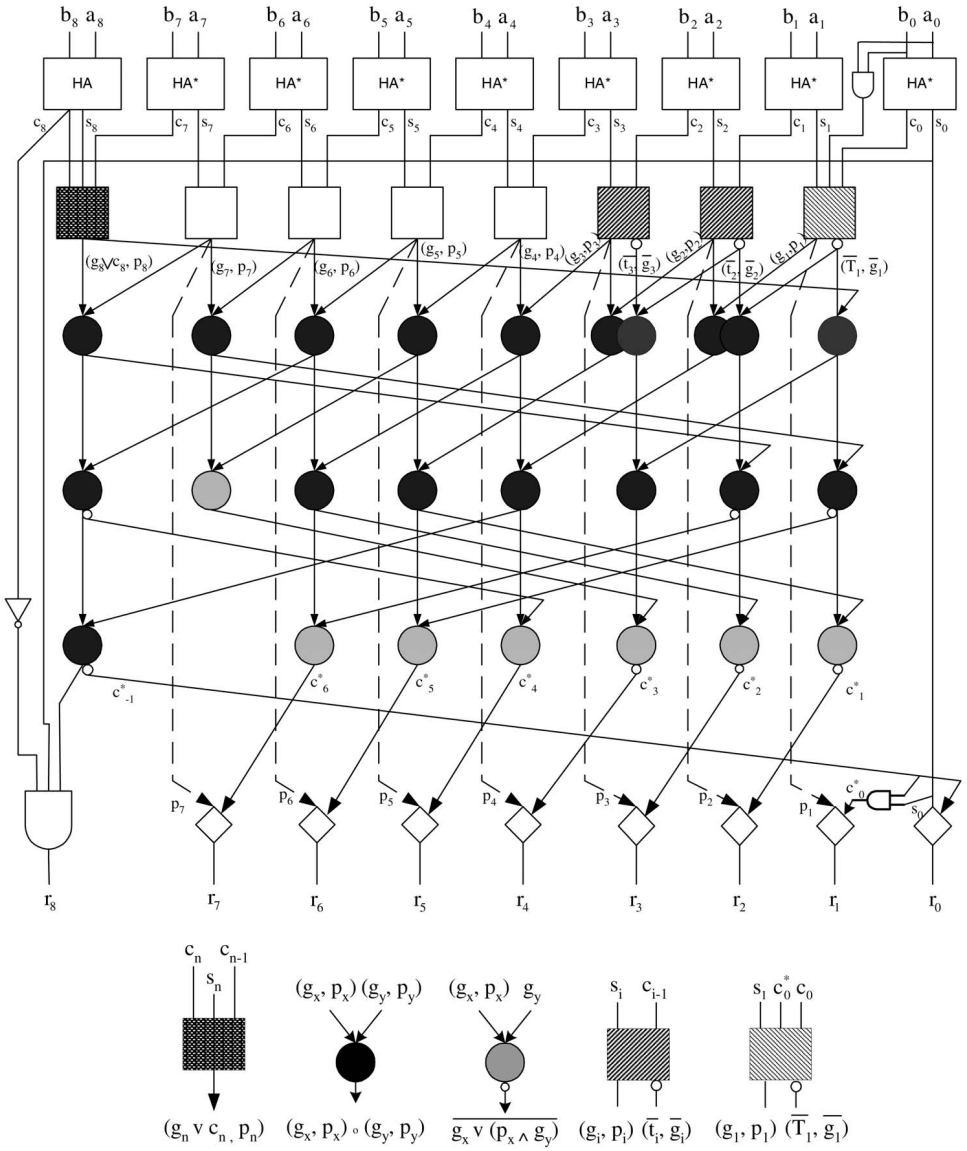

Fig. 3. TPP modulo $2^8 + 1$ adder.

TABLE 1
Time Optimization Comparison Results

	[22]		*PPFCI*			*TPP*		
n	Area	Time	Area	Time	Time Savings (%)	Area	Time	Time Savings (%)
4	4661	1.19	6063	1.07	10.08	4899	0.99	16.81
8	13153	1.48	11969	1.27	14.19	14009	1.15	22.30
16	29487	2.03	21280	1.64	19.21	31665	1.45	28.57

TABLE 2
Area Optimization Comparison Results

		[22]	*PPFCI*		*TPP*	
n	Target Latency	Area	Area	Area Savings (%)	Area	Area Savings (%)
4	1.19	4661	3685	20.94	3351	28.11
8	1.48	13153	8098	38.43	6715	48.95
16	2.03	29487	11562	60.79	11009	62.66

and TPP adders offer execution latencies equal to $(2 \times \log_2 n + 9)$ and $(2 \times \log_2 n + 6)$, respectively, that is, only few gates more than the rest channels. The use of the proposed architectures therefore minimizes the overhead imposed on the RNS operation speed.

5 Conclusions

Two new architectures have been presented for designing modulo $2^n + 1$ adders, based on the use of a parallel-prefix carry computation unit. The PPFCI architecture utilizes a fast carry increment stage, whereas in the TPP architecture, there is no need for such a stage since reentering carry recirculation is performed within each existing prefix level. Full static CMOS implementations have revealed the proposed adders' efficiency against previous solutions in both implementation area requirements and execution latency. The proposed architectures lead to the fastest reported modulo $2^n + 1$ adders, with execution latencies closer to the execution latency of the fastest modulo 2^n and modulo $2^n - 1$ adders than any other reported solution.

References

[1] N. Szabo and R. Tanaka, *Residue Arithmetic and Its Aplications to Computer Technology*. McGraw-Hill, 1967.

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

[3] F. Taylor, "A Single Modulus ALU for Signal Processing," *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. 33, pp. 1302-1315, 1985.

[4] E.D. Claudio et al., "Fast Combinatorial RNS Processors for DSP Applications," *IEEE Trans. Computers*, vol. 44, pp. 624-633, 1995.

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

[6] "High Performance, Reduced Complexity Programmable RNS-FPL Merged FIR Filters," *Electronics Letters*, vol. 38, no. 4, pp. 199-200, 2002.

[7] P.G. Fernandez et al., "A RNS-Based Matrix-Vector-Multiply FCT Architecture for DCT Computation," *Proc. 43rd IEEE Midwest Symp. Circuits and Systems*, pp. 350-353, 2000.

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

[9] T. Keller et al., "Adaptive Redundant Residue Number System Coded Multicarrier Modulation," *IEEE J. Selected Areas in Comm.*, vol. C-18, no. 11, pp. 2292-2301, 2000.

[10] V. Paliouras and T. Stouraitis, "Novel High-Radix Residue Number System Multipliers and Adders," *Proc. IEEE Int'l Symp. Circuits and Systems*, pp. 451-454, 1999.

[11] 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, 1994.

[12] A. Curiger, "VLSI Architectures for Computations in Finite Rings and Fields," PhD dissertation, Swiss Federal Inst. of Technology, 1993.

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

[14] Y. Ma, "A Simplified Architecture for Modulo $(2^n + 1)$ Multiplication," *IEEE Trans. Computers*, vol. 47, no. 3, pp. 333-337, Mar. 1998.

[15] L.M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform," *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. 24, pp. 356-359, 1976.

[16] R. Zimmermann, "Binary Adder Architectures for Cell-Based VLSI and Their Synthesis," PhD dissertation, Swiss Federal Inst. of Technology, 1997.

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

[18] C. Efstathiou et al., "On the Design of Modulo $2^n \pm 1$ Adders," *Proc. Eighth IEEE Int'l Conf. Electronics, Circuits & Systems*, pp. 517-520, 2001.

[19] H.T. Vergos et al., "Diminished-One Modulo $2^n + 1$ Adder Design," *IEEE Trans. Computers*, vol. 51, pp. 1389-1399, 2002.

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

[21] M. Dugdale, "VLSI Implementation of Residue Adders Based on Binary Adders," *IEEE Trans. Circuits Systems II*, vol. 39, pp. 325-329, 1992.

[22] A.A. Hiasat, "High-Speed and Reduced Area Modular Adder Structures for RNS," *IEEE Trans. Computers*, pp. 84-89, 2002.

[23] J.A. Abraham and D.D. Gajski, "Easily Testable High-Speed Realization of Register-Transfer-Level Operations," *Proc. 10th Fault Tolerant Computing Symp.*, pp. 339-344, 1980.

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

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

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

[27] 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, pp. 786-792, 1973.

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

[29] A. Tyagi, "A Reduced-Area Scheme for Carry-Select Adders," *IEEE Trans. Computers*, vol. 42, no. 10, pp. 1163-1170, Oct. 1993.