IET Circuits, Devices & Systems

Research Article

IET Journals
The Institution of Engineering and Technology

General modular adder designs for residue number system applications

ISSN 1751-858X
Received on 30th October 2017
Revised 15th January 2018
Accepted on 30th January 2018
E-First on 22nd March 2018
doi: 10.1049/iet-cds.2017.0470
www.ietdl.org

Ahmad Hiasat[1] ✉

[1]*Department of Computer Engineering, School of Engineering, Princess Sumaya University for Technology, P.O. Box 1438, Amman, Jordan*
✉ *E-mail: a.hiasat@psut.edu.jo*

Abstract: Modular adders are very crucial components in the performance of residue number system-based applications. Most of the work published so far has been restricted to modulo $(2^n \pm 1)$ adders or modulo-specific adders. Less work has been dedicated to modulo-generic adders. This work presents new designs for modulo $(2^n \pm K)$ adders, where K is any integer in the range of $3 \leq K < 2^{n-1}$. The proposed structure merges two binary adder structures and maximises sharing of components, wherever possible. This merger permits shorter cell-interconnections, which results in space wastage reduction. Additionally, tristate-based multiplexers (MUXs) are used in lieu of the more demanding gate-based MUXs. As examined over a very practical range of n, $7 \leq n \leq 15$, and based on a 65 nm VLSI realisation, the circuit layouts of the proposed adders outperform considerably the most recent and competitive functionally identical published works. On average, the proposed designs have shown reductions in area, time, power, and energy of 23.7, 13.8, 22.9, and 33.6%, respectively.

1 Introduction

The residue number system (RNS) is a non-weighted representation. This representation is based on expressing any number using relatively prime positive integers, known as moduli. The product of all moduli defines the dynamic range in which numbers within this range are uniquely represented [1]. When it comes to additions, subtraction and multiplication, each modulus is independent from other moduli in processing its computations [1]. The data path in which computations take place with respect to any modulus is referred to as a channel. For an RNS representation that uses L moduli, there are L parallel channels. The relatively independent and parallel channels reduce considerably the time needed for all computationally demanding arithmetic applications that depend mainly on the above listed operations.

When a certain dynamic range is specified, the level of parallelism achieved is higher if the number of moduli L is increased. This implies dividing the total number of bits of a dynamic range over more channels, thus, having shorter word-lengths for each channel, where a shorter word-length can be processed faster than a longer one [1]. The level of parallelism achieved using three-moduli sets proved to be reasonable for some digital signal processing (DSP) applications. Other DSP and cryptography applications require higher levels that cannot be attained using three-moduli sets [1–6].

When considering the number of moduli, three and four moduli sets have been heavily researched [1, 7–11]. To achieve higher levels of parallelism, five-moduli sets have been introduced such as $\{2^{2n}, 2^n - 3, 2^n + 3, 2^n - 1, 2^n + 1\}$, $\{2^n, 2^n - 1, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$, and $\{2^n, 2^n - 1, 2^n + 1, 2^n - 2^{(n+1/2)} + 1, 2^n + 2^{(n+1/2)} + 1\}$ [12–14].

An application-specific RNS-based processor uses mainly adders and multipliers. The area, time, and power needed to perform modular additions are very critical factors in applications that use RNS. A modular multiplier is basically seen as a process of successive binary and modular additions [1, 15, 16]. Examining the form of different moduli introduced in three- four-, and five-moduli sets reveals that most of these moduli are of the form $(2^n \pm 1)$ [13], $(2^n \pm 3)$ [12], $(2^n \pm 2^{(n+1/2)} + 1)$ [13, 14] or more general forms such as $(2^n \pm K)$, where $3 \leq K < 2^{n-1}$ [1]. Most of the published research in designing RNS-based modular adders has been dedicated to the moduli of the form 2^n and $(2^n \pm 1)$ [17–23]. Significant improvements have been achieved in the proposed designs, where the delay requirements of modulo $(2^n \pm 1)$ adders were getting closer to the delay of a modulo 2^n binary adders. Such an advancement is crucial in improving the overall performance of any RNS-based processor. On the other hand, very limited work has been dedicated to general modular adders [24–32], or modulo-specific adders, other than modulo $(2^n \pm 1)$ [33–35].

This work is intended to propose modulo $(2^n \pm K)$ adders that can serve any moduli, where any moduli can be expressed using one of the two given forms. The intention is to reduce the area and time requirements of such adders to approach those of modulo $(2^n \pm 1)$. Any improvement in this direction would make higher-order moduli sets, such as $\{2^{2n}, 2^n - 1, 2^n + 1, 2^n - 2^{(n+1/2)} + 1, 2^n + 2^{(n+1/2)} + 1\}$, more appealing in DSP and cryptographic applications that require high-level of parallelism.

This work is organised as follows: In Section 2, the previous published works related to moduli of general form and specific forms, other than $(2^n \pm 1)$, are revisited. In the first part of Section 3, a general background and the basic concepts of binary addition and parallel-prefix adders are introduced. In the second and third parts of Section 3, the proposed modulo $(2^n \pm K)$ adders are introduced. Section 4 evaluates the performance of the proposed adders as compared with the most competitive functionally similar modular adders. The comparison is held with the modular adders of [28, 29] using unit-gate model and VLSI realisation tools.

2 Previous work

The authors of [24] proposed a general modulo m adder, where $m < 2^n$, $n = \lceil \log_2 m \rceil$, and $\lceil . \rceil$ is the smallest integer equal to or greater than (.). It consists of two consecutive binary adders [24]. The first serial stage adds the two n-bit binary operands A and B, $0 \leq A, B < m$, using a binary adder. The second serial stage adds A, B and $(2^n - m)$, using also a binary adder. A multiplexer (MUX) in the third stage selects either the output of the first stage or the output of the second-stage depending on the output carry resulting from the second stage. The work in [25] presented also a modulo m adder using a sequential circuit. Modular addition is performed in

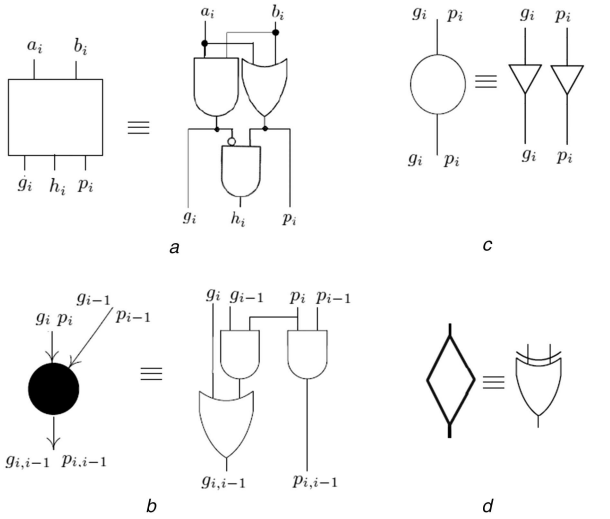

Fig. 1 *Block diagram and gate-level implementation of computational operators used in building different adders*
(a) A half-adder cell generating h_i, p_i, g_i, ***(b)*** Parallel-prefix black node operator, ***(c)*** Buffers, ***(d)*** Exclusive OR gate

two consecutive binary-based addition cycles using the same binary adder. MUXs are also used to select either the sum resulting from the first addition cycle, or to select the corrected sum resulting from the second cycle. The work of [26] proposed a modulo m adder as a last stage in a residue-to-binary conversion process. This modular adder was based on using a carry-save adder (CSA), two binary adders, and an MUX. The CSA computes $(A + B - m)$ as a sum $\boldsymbol{S}$ vector and a carry vector $\boldsymbol{C}$. The two binary adders operate simultaneously. The first binary adder computes the sum of $A + B$, while the second binary adder computes the sum of $\boldsymbol{S} + \boldsymbol{C}$. The MUX selects one of the two output results. The structure of the modulo m adder introduced in [27] is based on producing the propagate and generate vectors of $(A + B)$ and producing the propagate and generate vectors of $(A + B - m)$. The output carry resulting from a carry look-ahead structure selects either the propagate and generate vectors of $(A + B)$ or the ones of $(A + B - m)$. The selected vectors are then processed using a binary adder.

The modulo m adder introduced in [28] utilises two binary adders. The first adder computes $(A + B)$. The CSA, which is embedded in the second adder, reduces the three operands to two using n half-adder/pseudo half-adder circuits (a pseudo half-adder is a half-adder that adds two bits while assuming that the third added bit is (i). The second adder adds the sum and carry vectors resulting from the CSA. The outputs of both adders are applied to an MUX that selects one of the outputs based on the output carry. The adder suggested in [29] is built using three main circuits. The first circuit is a simplified CSA circuit. The second one is a double adder with shared hardware carry-propagate adder (CPA) circuit. The third part is an MUX circuit. The first circuit receives A and B as input operands and creates the carry-save representation of $A + B$, and the carry-save representation of $A + B + T$, where T is the two's complement of modulus m. This CSA circuit is similar to the CSA used in [26–28]. The simplicity of this circuit depends greatly on the binary pattern of T. The second circuit computes both sums $A + B$ and $A + B + T$ in parallel. This circuit permits the sharing of arithmetic sub-circuits in the CPA. The MUX circuit, built using n (2×1) MUXs, selects one of the two n-bit sums resulting from the second circuit. The modular adder presented in [30] uses a special representation, referred to as δ representation. Although the proposed adder is efficient in terms of area and time, this special representation requires additional hardware when converting the δ representation to RNS. The modulo $(2^n \pm k)$ adders introduced in [31, 32] deals with the case of two-, three-, or four-operands addition (i.e. residue generators), which is not within the scope of the work presented here nor the works in [24–30].

Modulo-specific modular adders, other than $(2^n \pm 1)$, were also introduced. A modulo $(2^n - 2^{n-2} - 1)$ was presented that deals with this specific modulus [33]. A modulo $(2^n \pm 3)$ has been proposed in [34]. A more flexible modulus-specific modular adder is presented in [35], which deals with the modulus $(2^n - 2^k - 1)$, $1 \leq k \leq n - 2$.

3 Proposed modulo $(2^n \pm K)$ adders

As outlined in the Section 2, the basic concept adopted in most previous general modular adders was the use of two different n-bit binary adders; one to compute $A + B$ and one to compute $A + B - m$. Input, output, and internal submodule interconnections were routed in two distant groups, where each group dealt with one adder. Additionally, a gate-based multiplexing stage was used to select one of the two outputs. Different alterations to this capacious concept were introduced in [24–29] to reduce either area, time, or both.

The approach proposed in this paper merges the two binary adders into one, which allows component sharing, particularly in the preprocessing and the sum-computation stages. This merger produces also shorter interconnections from one standard cell to the abutting one(s). Additionally, this work proposes the use of the relatively-unconventional tristate-based selectors in modular adders instead of the conventional gate-based selectors in the multiplexing stage.

3.1 Background and basic concepts

Assuming that n is a positive integer and that A and B are two n-bit non-negative integers where $A = a_{n-1}, \ldots, a_1 a_0$ and $B = b_{n-1}, \ldots, b_1 b_0$, the sum $\boldsymbol{S} = A + B$, where $\boldsymbol{S} = s_n s_{n-1}, \ldots, s_1 s_0$, is evaluated in three stages: the preprocessing stage, the parallel-prefix stage, and the final sum stage [17–23], described as follows:

- *Preprocessing stage*: This stage creates three $\boldsymbol{n}$ bit vectors, namely the half-sum vector $\boldsymbol{H}$, the carry-propagate vector $\boldsymbol{P}$, and the carry-generate vector $\boldsymbol{G}$, defined as

$$\begin{aligned} &H = h_{n-1}, \ldots, h_1 h_0, \\ &h_i = a_i \oplus b_i \end{aligned} \tag{1}$$

$$\begin{aligned} &\boldsymbol{P} = p_{n-1}, \ldots, p_1 p_0, \\ &p_i = a_i \vee b_i \end{aligned} \tag{2}$$

$$\begin{aligned} &\boldsymbol{G} = g_{n-1}, \ldots, g_1 g_0, \\ &g_i = a_i \wedge b_i \end{aligned} \tag{3}$$

  where $0 \leq i \leq n - 1$. The symbols $\oplus$, $\vee$, and $\wedge$ refer to exclusive OR, OR, and AND logic operators, respectively. The block diagram and gate-level implementation of a half-adder cell generating h_i, p_i, and g_i are shown in Fig. 1*a*. Furthermore, the preprocessing stage of a binary adder is shown in Fig. 2 for the case $n = 7$.
- *Parallel-prefix stage*: This stage creates the n-bit carry vector $\boldsymbol{C} = c_{n-1}, \ldots, c_0$, where the basic component in this stage is a parallel-prefix cell that receives two pairs of bits, (p_i, g_i) (p_{i-1}, g_{i-1}), and computes the pair $(p_{i,i-1}, g_{i,i-1})$, where:

$$\begin{aligned} p_{i,i-1} &= p_i \wedge p_{i-1} \\ g_{i,i-1} &= g_i \vee (g_{i-1} \wedge p_i) \end{aligned}$$

Fig. 1*b* shows the block diagram the of the basic parallel-prefix black node cell and its gate-level implementation. The parallel-prefix stage for the case $n = 7$, using Ladner–Fischer structure [36], is shown in Fig. 2.

It should be pointed out that other parallel-prefix structures, such as the structures of [37–39], can also be used, where the basic building block is the black node operator shown in Fig. 1. In this paper, we have selected the Ladner–Fischer structure [36] for

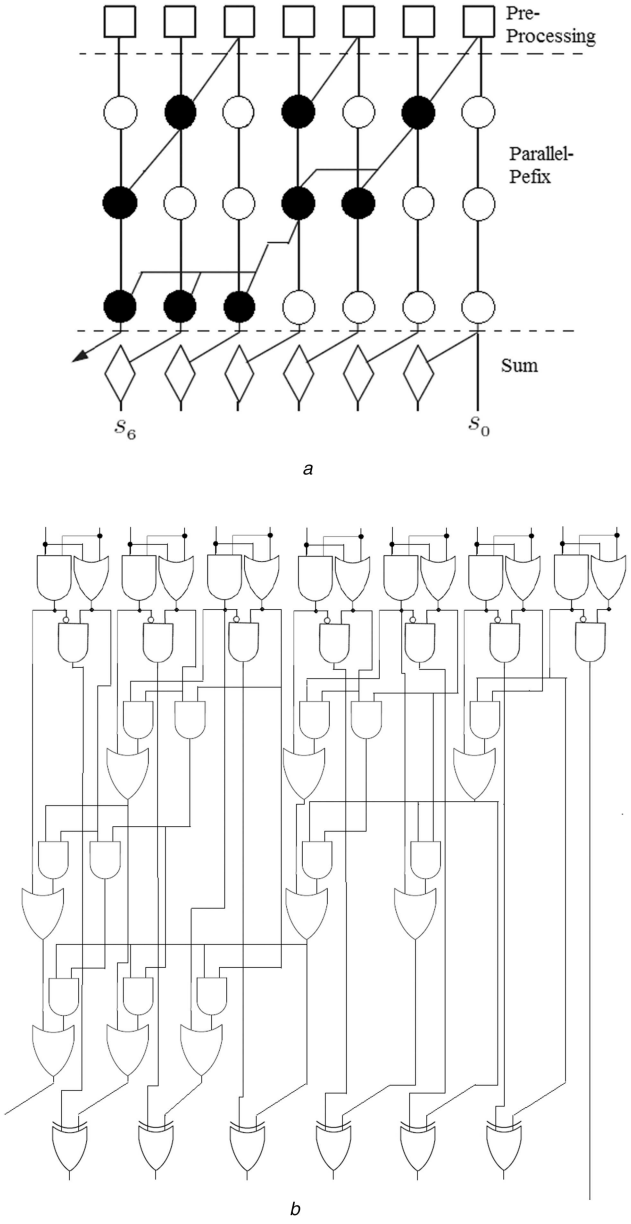

Fig. 2 *Seven-bit binary adder using Ladner–Fischer parallel-prefix structure [36]*
(a) Block diagram, ***(b)*** Logic-level implementation using different cells of Fig. 1, where buffers are not shog′wn

comparison purposes, where the other two competitive general modular adders [28, 29] have used the structure of Ladner–Fischer [36].

• *Sum computation stage*: Computing the final sum S of adding the two n-bit operands A and B is performed at this level, where:

$$s_0 = h_0 \tag{4}$$

$$s_i = h_i \oplus c_{i-1}, \quad 1 \le i \le n-1 \tag{5}$$

$$s_n = c_{n-1} \tag{6}$$

3.2 Proposed modulo $(2^n - K)$ adder

An RNS-based modulo $(2^n - K)$ adder is defined as

$$\boldsymbol{S} = \langle A + B \rangle_{m_1} \tag{7}$$

where $m_1 = (2^n - K)$, $3 \le K \le 2^{n-1} - 1$.

Equation (7) can be rewritten as

$$\boldsymbol{S} = \begin{cases} A + B & \text{if } A + B < m_1 \\ A + B - (2^n - K) & \text{if } A + B \ge m_1 \end{cases} \tag{8}$$

Observing in both cases of the last equation that $\boldsymbol{S} < m_1 < 2^n$, then applying modulus 2^n to both cases of (8) results in:

$$\boldsymbol{S} = \begin{cases} \langle A + B \rangle_{2^n} & \text{if } c_{\text{out}} = 0 \\ \langle A + B + K \rangle_{2^n} & \text{if } c_{\text{out}} = 1 \end{cases} \tag{9}$$

where c_{out} is the output carry resulting from computing $(A + B + K)$. It should be observed that applying modulus 2^n to any non-negative integer implies considering just the least significant n bits of the binary representation of the integer.

The basic concept used in this paper for evaluating S in (9) is to compute, simultaneously, both output cases (i.e. $A + B$ and $A + B + K$), determine c_{out} and select one of the two outputs via tristate-based MUXs.

The output of the first case of (9) (i.e. $\langle A + B \rangle_{2^n}$) is simply performed by a structure similar to Fig. 2. However, in order to compute the second case of (9), the three stages of the adder of Fig. 2 (namely preprocessing stage, parallel-prefix stage, and the sum computation stage) need to be modified as follows:

i. *Preprocessing stage*: The binary representations of A, B and K of (9), where $K = \overbrace{0k_{n-2}k_{n-3}, \ldots, k_1k_0}^{n}$, are expressed as

$$\begin{aligned} A \rightarrow \quad & \overbrace{a_{n-1}a_{n-2}, \ldots, a_1a_0}^{n} \\ +B \rightarrow \quad & \overbrace{b_{n-1}b_{n-2}, \ldots, b_1b_0}^{n} \\ +\widetilde{K} \rightarrow \quad & \overbrace{0\ k_{n-2}, \ldots, k_1k_0}^{n} \\ A' \rightarrow \quad & a'_{n-1}, \ldots, a'_1a'_0 \\ +B' \rightarrow \quad & b'_nb'_{n-1}, \ldots, b'_2b'_10 \end{aligned} \tag{10}$$

where

$$a'_i = \begin{cases} a_i \oplus b_i & \text{if } k_i = 0 \\ a_i \odot b_i & \text{if } k_i = 1 \end{cases} \tag{11}$$

and

$$b'_{i+1} = \begin{cases} a_i \wedge b_i & \text{if } k_i = 0 \\ a_i \vee b_i & \text{if } k_i = 1 \end{cases} \tag{12}$$

where $\odot$ refers to an exclusive NOR logic operator. Adding a_i to b_i is performed using a half-adder circuit if $k_i = 0$. However, adding a_i, b_i, and k_i is performed using a pseudo half-adder circuit if $k_i = 1$. The half-adder and pseudo half-adder circuits are shown in Fig. 3*a*.

The preprocessing stage of the new proposed adder is shown in Fig. 4. It consists of two consecutive phases as follows:

- The first phase, which consists of n hashed cells of Fig. 3*b*, receives the two n-bit vectors $\boldsymbol{A}$ and $\boldsymbol{B}$, and produces the three n-bit vectors $\boldsymbol{H}$, $\boldsymbol{P}$ and $\boldsymbol{G}$ needed for computations of the first case of (9). As a byproduct, the same hardware can produce the two vectors $\boldsymbol{A}' = a'_{n-1}, \ldots, a'_1a'_0$ and $\boldsymbol{B}' = b'_n, \ldots, b'_2b'_10$, which are needed to reduce the three vectors of the second case of (9) to two.
- The second phase consists of n envelope cells of Fig. 3*c*. These cells receive the two vectors $\boldsymbol{A}'$ and $\boldsymbol{B}'$ resulting from phase 1 and produce the corresponding half-sum $\boldsymbol{H}'$ vector, carry-propagate $\boldsymbol{P}'$ vector, and carry-generate $\boldsymbol{G}'$ vector, which are defined as

$$\begin{aligned} \boldsymbol{H}' &= h'_{n-1}, \ldots, h'_1h'_0, \\ h'_i &= a'_i \oplus b'_i \end{aligned} \tag{13}$$

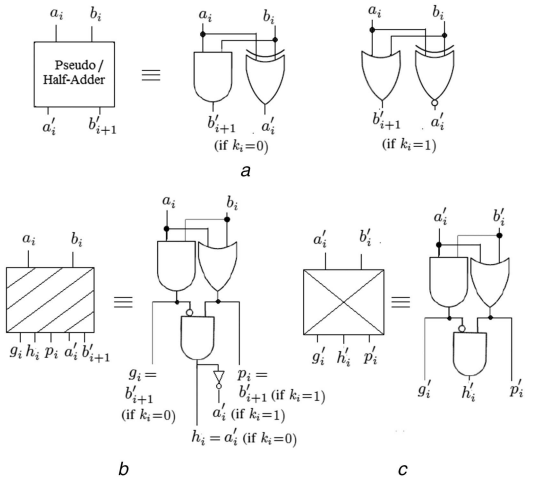

Fig. 3 *Block diagram and gate-level implementation of*
(a) Half-adder and a pseudo half-adder, ***(b)*** Hashed cell that generates h_i, p_i, g_i, a'_i, b'_{i+1}, ***(c)*** Envelope cell that generates h'_i, p'_i, where the envelope cell is functionally identical to the square cell of Fig. 1*a*

$$\begin{aligned} \boldsymbol{P}' &= p'_{n-1}, \ldots, p'_1 p'_0, \\ p'_i &= a'_i \vee b'_i \end{aligned} \tag{14}$$

$$\begin{aligned} \boldsymbol{G}' &= g'_{n-1}, \ldots, g'_1 g'_0, \\ g'_i &= a'_i \wedge b'_i \end{aligned} \tag{15}$$

where $0 \le i \le n-1$ and $b'_0 = 0$. Moreover, since, $k_{n-1} = 0$, then $b'_n = a_{n-1} \wedge b_{n-1}$. Hence, b'_n can be easily incorporated as an input to the last OR gate of the parallel-prefix cell producing c_{out}.

Therefore, the preprocessing stage, shown in Fig. 4, accepts $\boldsymbol{A}$ and $\boldsymbol{B}$ and produces six vectors, namely $\boldsymbol{H}$, $\boldsymbol{H}'$, $\boldsymbol{P}$, $\boldsymbol{P}'$, $\boldsymbol{G}$, and $\boldsymbol{G}'$.

ii. *Parallel-prefix stage*: This stage is shown in Fig. 5. It receives two pairs of vectors, $(\boldsymbol{P}, \boldsymbol{G})$ and $(\boldsymbol{P}', \boldsymbol{G}')$. The first pair $(\boldsymbol{P}, \boldsymbol{G})$ produces the carry vector $\boldsymbol{C} = c_{n-1}, \ldots, c_1 c_0$. The second pair $(\boldsymbol{P}', \boldsymbol{G}')$ produces the carry vector $\boldsymbol{C}' = c'_{n-1}, \ldots, c'_1 c'_0$, where $c_{\text{out}} = c'_{n-1}$.

iii. *Multiplexing and sum computation stage*: Based on the output carry bit c_{out}, one of the two vectors $\boldsymbol{H}$ and $\boldsymbol{H}'$, produced in the preprocessing stage, will be selected. If $c_{\text{out}} = 0$, $\boldsymbol{H}$ is selected. Otherwise, $\boldsymbol{H}'$ is selected. Similarly, one of the two vectors $\boldsymbol{C}$ and $\boldsymbol{C}'$, produced in the parallel-prefix stage, will be selected at this stage, where $\boldsymbol{C}$ is selected if $c_{\text{out}} = 0$, and $\boldsymbol{C}'$ is selected otherwise. The two selected vectors are then exclusive ORed using (4) and (5). Fig. 6 shows the proposed multiplexing and sum computation circuit for $n = 7$. Multiplexing is realised using unidirectional tri-state buffers. A total of $2n$ tri-state buffers are needed to select either the half-sum vector$\boldsymbol{H}$, or to select the half-sum vector$\boldsymbol{H}'$. Similarly, a total of $2(n-1)$ tri-state buffers are also needed to select either the carry vector $\boldsymbol{C}$ or to select the vector $\boldsymbol{C}'$. Producing the final sum requires a total of $(n-1)$ exclusive OR gates.

The block diagram of the proposed modulo $(2^n - K)$, which combines the above described stages for both cases of (9), is shown in Fig. 5.

3.3 Proposed modulo $(2^n + K)$ adder

Using a similar approach to the one followed in the previous subsection, a residue-based modulo $(2^n + K)$ adder can be defined as

$$\boldsymbol{S} = \langle A + B \rangle_{m_2} \tag{16}$$

where $m_2 = (2^n + K)$, $3 \le K \le 2^{n-1} - 1$, and $A, B \in [0, m_2 - 1]$.

Re-expressing (16)

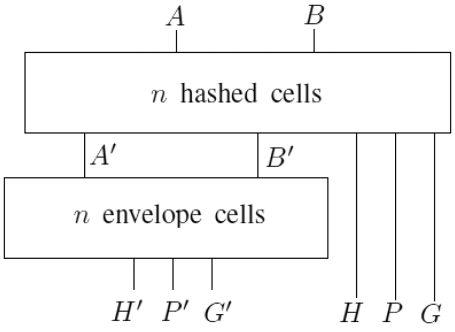

Fig. 4 *Block diagram of the preprocessing stage of the proposed modular adder. The first phase consists of n hashed cells of Fig. 3b. The second phase consists of n envelope cells of Fig. 3c*

$$\boldsymbol{S} = \begin{cases} A + B & \text{if } A + B < m_2 \\ A + B - (2^n + K) & \text{if } A + B \ge m_2 \end{cases} \tag{17}$$

where $\boldsymbol{S} < m_2 < 2^{n+1}$.

Observing that $\langle -(2^n + K) \rangle_{2^{n+1}} = \langle 2^{n+1} - (2^n + K) \rangle_{2^{n+1}} = \langle 2^n - K \rangle_{2^{n+1}}$, and applying modulus 2^{n+1} to the two cases of (17):

$$\boldsymbol{S} = \begin{cases} \langle A + B \rangle_{2^{n+1}} & \text{if } c_{\text{out}} = 0 \\ \langle A + B + \widetilde{K} \rangle_{2^{n+1}} & \text{if } c_{\text{out}} = 1 \end{cases} \tag{18}$$

where $\widetilde{K} = 2^n - K$ and where c_{out} is the output carry, whose binary weight is 2^{n+1}, resulting from computing $(A + B + \widetilde{K})$.

Similar to the previous subsection, the output of the case $\langle A + B \rangle_{2^{n+1}}$ of (18) is computed using the adder of Fig. 2 if $n = 6$, or with adding one more bit to that structure if $n = 7$. However, when considering the case of $\langle A + B + \widetilde{K} \rangle_{2^{n+1}}$ of (18), the implementation proceeds as follows: Since A and B are integers that belong to the range $[0, m_2 = 2^n + K)$, the binary representations of A, B and $\widetilde{K}$ of (18) are expressed in $(n+1)$ bits each. That is:

$$\begin{aligned} \boldsymbol{A} \rightarrow \quad & \overbrace{a_n a_{n-1} a_{n-2}, \ldots, a_1 a_0}^{n+1} \\ +\boldsymbol{B} \rightarrow \quad & \overbrace{b_n b_{n-1} b_{n-2}, \ldots, b_1 b_0}^{n+1} \\ +\widetilde{K} \rightarrow \quad & \overbrace{0\ \widetilde{k}_{n-1} \widetilde{k}_{n-2}, \ldots, \widetilde{k}_1 \widetilde{k}_0}^{n+1} \\ = \ & - - - - - - - - - - - - - - - - \\ \boldsymbol{A}' \rightarrow \quad & a'_n a'_{n-1}, \ldots, a'_1 a'_0 \\ +\boldsymbol{B}' \rightarrow \quad & b'_{n+1} b'_n b'_{n-1}, \ldots, b'_2 b'_1 0 \end{aligned} \tag{19}$$

$$a'_i = \begin{cases} a_i \oplus b_i, & \text{if } \widetilde{k}_i = 0 \\ a_i \odot b_i, & \text{if } \widetilde{k}_i = 1 \end{cases} \tag{20}$$

and

$$b'_{i+1} = \begin{cases} a_i \wedge b_i, & \text{if } \widetilde{k}_i = 0 \\ a_i \vee b_i, & \text{if } \widetilde{k}_i = 1 \end{cases} \tag{21}$$

The same proposed structure of Fig. 5 for the case $n = 7$ is still usable as modulo $(2^n + K)$ adder but after being extended by one bit. Alternatively, the same structure of Fig. 5 can be used as modulo $(2^n + K)$ adder for the case $n = 6$.

4 Evaluation and VLSI realisation

In this section, the hardware and time requirements of the proposed adder along with the most competitive non-modulo-specific adders

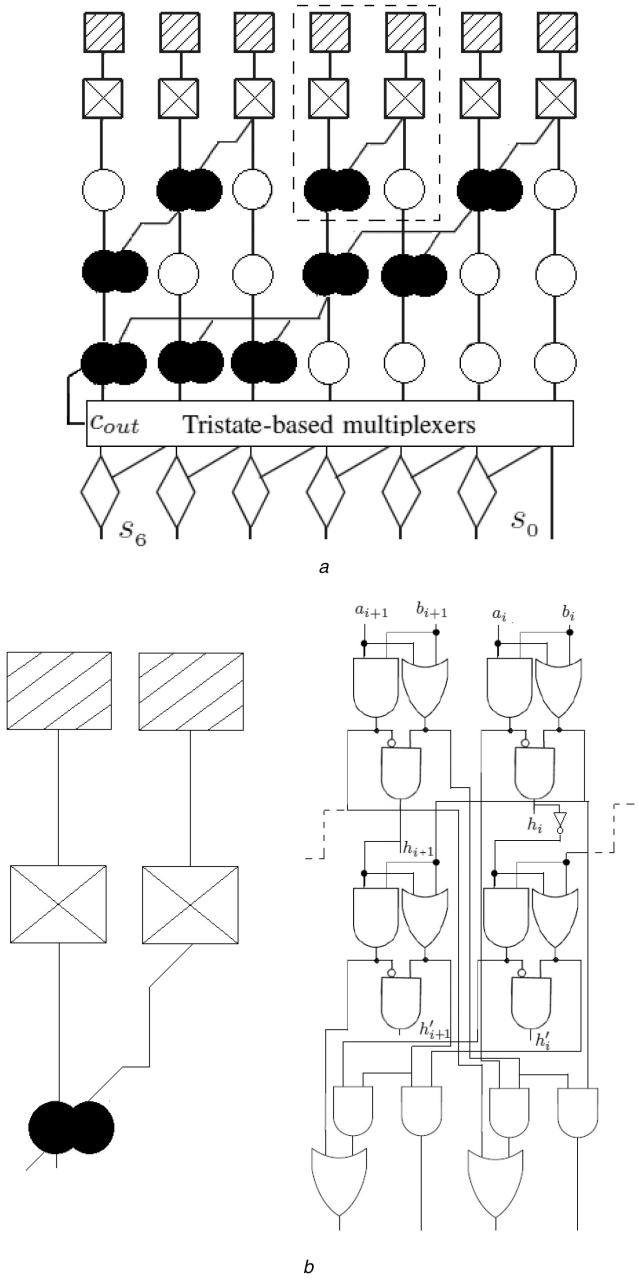

Fig. 5 *The proposed general modular adder*

(a) Block diagram of the proposed modulo $(2^n \pm K)$, ***(b)*** Logic-level implementation of a representative part of (a) using the cells shown in Figs. 1 and 3, where buffers are not shown. It has been assumed that $k_i = 1$ and $k_{i+1} = 0$. The gate-level implementation of the tristate-based MUXs and the sum computation stage are shown in Fig. 6

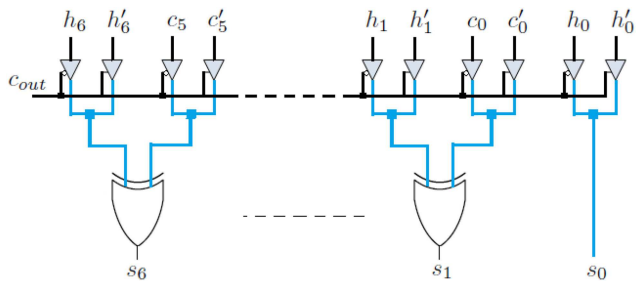

Fig. 6 *Gate-level implementation of the tristate-based MUXs and the sum stage of Fig. 5 for the case n = 7*

will be evaluated and compared using two methods, namely, the unit gate model and the VLSI realisation.

4.1 Evaluation based on the unit-gate model

Following the technology-independent approach introduced in [17, 40] and adopted by many researchers dealing with computer arithmetic and modular components, the unit-gate model is used to theoretically estimate the area and time delay of many arithmetic digital circuits [11–23, 29–33, 35]. The model uses a monolithic two-input gate as a reference, where the area of such a gate is considered to be 1 area-unit (A_g) and its delay is 1 time-unit (τ_g). This include two-input AND, NAND, OR, and NOR. The model ignores inverters and considers an exclusive OR and exclusive NOR to have an area of $2A_g$ and delay of $2\tau_g$ [17, 40]. A (2×1) MUX realised using two unidirectional tri-state CMOS buffers has the area of $2A_g$ and the delay of $1\tau_g$ [41].

Table 1 lists the different detailed requirements of the adder proposed in Fig. 5 using unit-gate model parameters. Using the same parameters, Table 2 compares the area and delay of the proposed work with the most competitive general modular adders that can deal with the moduli of any form without any restriction of any form [28, 29]. The adder in [30] has been excluded as it requires an additional area and time to convert the special δ representation to RNS. Table 2 also lists the area and time requirements of the two modulo-specific adders [33, 35], as references. The table also shows that the modulo-specific adder of [35] with one parallel-prefix tree ($1.5n \log n$ unit gates) is more area-efficient than the proposed modulo-generic adder with two-parallel-prefix trees ($3n \log n$ unit gates). Since the modulo-specific adders in [33, 35] are, in fact, special cases of the more general ones, no further comparison is carried with these types of adders. Fig. 7 shows the area-time product using the technology-independent data of Table 2 of the proposed adder along with the modular adders of [28, 29].

4.2 VLSI realisation and evaluation

In order to get a better evaluation of the performance of the proposed adders and the most competitive functionally identical ones, VLSI tools were used to realise a technology-based circuit layout. To achieve this goal, the proposed designs and the generic-moduli modular adders of [28, 29] were modelled using Verilog Hardware Description Language. Randomly selected prime numbers for values of $(n = 7, 9, 11, 13, 15)$ that have no specific form were considered. For $n = 7$, the selected prime moduli were 83 $(2^7 - 45)$ and 167 $(2^7 + 39)$. For $n = 9$, the selected prime numbers were 419 $(2^9 - 93)$ and 757 $(2^9 + 245)$. Similarly, for $n = 11$, 13 and 15, the corresponding selected prime numbers were $\{1777\ (2^{11} - 271),\ 2579\ (2^{11} + 531)\}$, $\{6173\ (2^{13} - 2019),\ 11353\ (2^{13} + 3161)\}$, and $\{27073\ (2^{15} - 5695),\ 35083\ (2^{15} + 2315)\}$, respectively.

The Verilog Hardware Description Language was used to model all structures under consideration (i.e. proposed [28, 29]). The Synopsys Design Compiler (Version G-2012.06) was used to perform the synthesis (at 1.8 V for the core voltage and 25°C for the temperature). All synthesised circuits were also mapped to the 65 nm Synopsys DesignWare CMOS Digital Logic Libraries. Using Synopsys Prime Time PX, the Monte-Carlo power simulations were conducted utilising randomly produced inputs [42]. The designs were simulated till the power estimated values were within a confidence interval of 96% and an error smaller than 2%.

Table 3 lists the main VLSI parametric circuit layout results of area, delay and power after the place-and-route phase. The percentage of reductions obtained by the proposed designs as compared with each of the other two competitive designs [28, 29] are also listed in Table 4. Upon examining the values of Table 4, it can be concluded that the area, time and power reductions of the proposed adder, as compared with [28], are given in the ranges of (19.5–33.8), (13.8–20.5) and (17.9–34.4)%, respectively, where the competitive work is used as a reference in all cases. When compared with the design of [29], the new design has achieved reductions in area, time and delay expressed in the ranges of (15.4–25.8), (9.6–14.1) and (14.8–24.8)%, respectively.

Table 1 Area and delay of the proposed $(2^n - K)$ modular adder using unit-gate model. The same results apply to $(2^n + K)$ modular adder, by replacing n by $(n + 1)$

Stage	Type of cell	Number of cells	Number of gates	Critical delay
preprocessing	hatched	n	$3n$	2
	envelope	n	$3n$	1[a]
parallel-prefix	black[b]	$n\log n$	$3n\log n - 2(n-1)$	$2\log n$
multiplex	tristate-based (2×1) MUX	$2n - 1$	$4n - 2$	1
final sum	exclusive OR	$n - 1$	$2(n-1)$	2
total for mod $(2^n - K)$	—	—	$3n\log n + 10n - 2$	$2\log n + 6$
total for mod $(2^n + K)$	—	—	$3(n+1)\log(n+1) + 10n + 8$	$2\log(n+1) + 6$

[a]The envelope cell requires a critical delay of 1 unit to produce g_i' and p_i'. Producing h_i' is not in the critical path of the preprocessing stage of the proposed adder because h_i' will be used, later on, in the multiplexing stage.

[b]The bottom two cells in every column of the parallel-prefix structure of Fig. 5 consist of just two gates each, where the carry propagate variables p need not be produced at this level.

Table 2 Area and delay of the proposed $(2^n - K)$ modular adder and other competitive published work

Modular adder[a]	Area (A_g)	Delay (τ_g)
proposed[b]	$3n\log n + 10n - 2$	$2\log n + 6$
[28]	$3n\log n + 16n - 4 - \log n$	$2\log n + 8$
[29]	$(4n-7)\log n + 15n - 11$	$2\log n + 7$
[33][c]	$3n\log n + 11n - 2 + 2^{\log n - 1}$	$2\log n + 5$
[35][d]	$1.5n\log n + 13n - 5k - 8$	$2\log n + 7$

[a]To ease comparison, it has been assumed that $\log n \simeq \log(n-1)$.

[b]Area and delay are for modulo $(2^n - K)$ adder. For modulo $(2^n + K)$, n is replaced with $(n + 1)$.

[c]Modulo-specific adder: modulus $(2^n - 2^{n-2} - 1)$.

[d]Modulo-specific adder: modulus $(2^n - 2^k - 1)$, $1 \le k \le n - 2$.

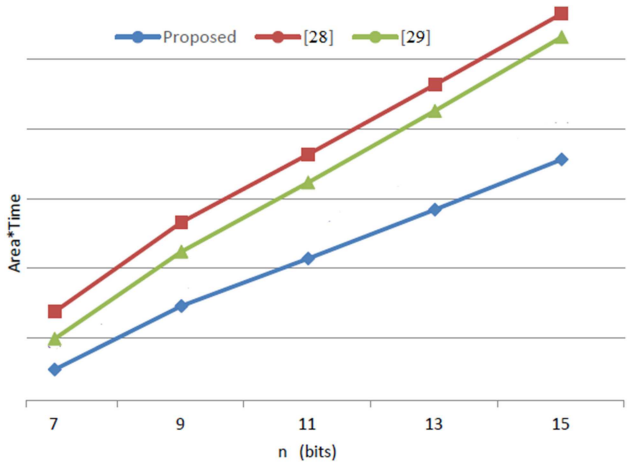

Fig. 7 *Area–time product as computed using the gate unit model based on the data of Table 2*

For further analysis, Fig. 8 shows the area–time product of all designs over the range of the moduli selected. The proposed design is less area–time product by (31.0–44.0)% compared with [28], and by (23.5–34.2)% compared with [29]. Similarly, Fig. 9 shows the power–time product (energy) over the same range, which indicates that the new design is significantly energy-efficient. The proposed design is more energy-efficient than [28, 29] by (29.5–46.4), and (25.4–34.5)%, respectively. Both figures, Figs. 8 and 9, are plotted based on the data of Table 3.

As compared with all modular adders in [19–29], the above significant improvements are achieved due to three main reasons:

- Merging two binary-adder structures into one structure, thus, producing shorter interconnections between abutted standard cells, and resulting in space wastage reduction.
- Sharing components in preprocessing and sum computation stages, wherever possible.

Table 3 VLSI realisation results of area, delay, and power of the proposed modular adder and the adders previously presented in [28, 29]

n (modulus)	Structure	Area, μm^2	Delay, ps	Power, μW
7	proposed	532	385	343
(83)	[28]	738	485	508
	[29]	628	426	416
7	proposed	561	417	361
(167)	[28]	847	494	550
	[29]	718	486	423
9	proposed	744	465	480
(419)	[28]	1044	560	719
	[29]	963	520	637
9	proposed	856	497	525
(757)	[28]	1178	592	724
	[29]	1059	574	653
11	proposed	973	476	627
(1777)	[28]	1306	561	867
	[29]	1262	546	835
11	proposed	1125	512	685
(2579)	[28]	1398	597	889
	[29]	1373	582	874
13	proposed	1140	490	735
(6173)	[28]	1622	565	975
	[29]	1503	550	919
13	proposed	1282	515	781
(11,353)	[28]	1704	600	998
	[29]	1612	596	933
15	proposed	1312	491	846
(27,073)	[28]	1750	572	1030
	[29]	1768	553	1037
15	proposed	1472	522	870
(35,083)	[28]	1867	605	1072
	[29]	1845	597	1069

- Replacing conventional gate-based MUXs used in all earlier works of [19–29] with tristate-based MUXs.

5 Conclusions

The potential of RNS in DSP and cryptographic applications can be highly enhanced by improving the performance of modular adders. This paper presented new designs for modulo-generic modular adders. The moduli can take any form of $(2^n \pm K)$, $3 \le K \le 2^{n-1} - 1$. Using VLSI tools, the proposed structures proved to be significantly more efficient than other functionally identical modular adders, where no restriction on the form of moduli are imposed. Compared to other general modular adders, the new presented adders reduced area requirements by the range

Table 4 Percentages of area, delay, and power reductions achieved by the proposed modular adder compared with the adders of [28, 29]

n (modulus)	Proposed comp. with	Area reduction, %	Delay reduction, %	Power reduction, %
7	[28]	27.9	20.5	32.5
(83)	[29]	15.4	9.6	17.5
7	[28]	33.8	15.5	34.4
(167)	[29]	21.9	14.1	14.8
9	[28]	28.7	17.0	33.2
(419)	[29]	22.7	10.6	24.6
9	[28]	27.3	16.1	27.5
(757)	[29]	19.1	13.4	19.6
11	[28]	25.5	15.2	27.6
(1777)	[29]	22.9	12.8	24.8
11	[28]	19.5	14.3	23.0
(2579)	[29]	18.1	12.0	21.6
13	[28]	29.7	13.3	24.6
(6173)	[29]	24.2	10.9	20.0
13	[28]	24.8	14.2	21.7
(11,353)	[29]	20.5	13.6	16.3
15	[28]	25.1	14.2	17.9
(27,073)	[29]	25.8	11.3	18.9
15	[28]	21.2	13.8	18.8
(35,083)	[29]	20.2	12.6	18.6

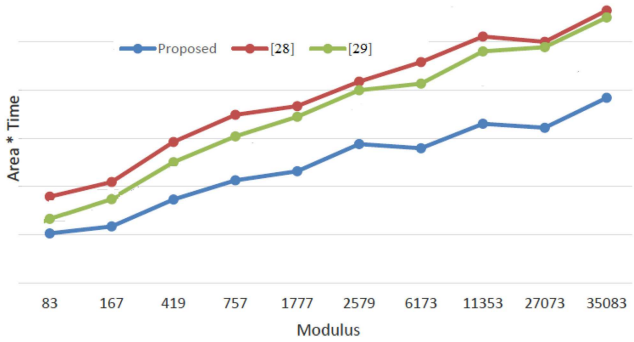

Fig. 8 *Area–time product based on the VLSI layout results of Table 3*

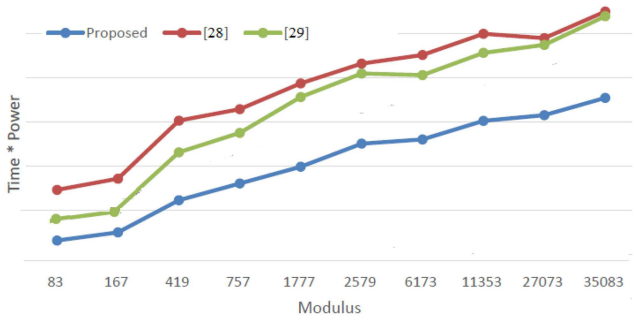

Fig. 9 *Time–power product based on the VLSI layout results of Table 3*

of (15.4–33.8)%, time requirements by the range of (9.6–20.5)%, power dissipation by the range of (14.8–34.4)%, and energy requirements by the range of (23.5–46.4)%. The new structures enabled improving the level of parallelism needed by many RNS-based applications and the overall computational speed.

6 References

[1] Mohan, P.V.A.: '*Residue number systems: theory and applications*' (Birkhauser, Basel, Switzerland, 2016)

[2] Hiasat, A., Abdel-Aty-Zohdy, H.: 'Design and implementation of a fast and compact residue-based semi-custom VLSI arithmetic chip'. Proc. 37th Midwest Symp. Circuits Systems, August 1994, pp. 428–431

[3] Toivonen, T., Heikkil, J.: 'Video filtering with fermat number theoretic transforms using residue number system', *IEEE Trans. Circuits Syst. Video Technology*, 2006, **16**, (1), pp. 92–101

[4] Vun, C., Premkumar, A., Zhang, W.: 'A new RNS based DA approach for inner product computation', *IEEE Trans. Circ. Syst. CAS-I*, 2013, **60**, (9), pp. 2139–2152

[5] Esmaeildoust, M., Schinianakis, D., Javashi, H., *et al.*: 'Efficient RNS implementation of elliptic curve point multiplication over GF(p)', *IEEE Trans. Very Large Scale Integr. Syst.*, 2013, **21**, (8), pp. 1545–1549

[6] Zheng, X., Wang, B., Zhou, C., *et al.*: 'Parallel DNA arithmetic operation with one error detection based on 3-moduli set', *IEEE Trans. Nanosci.*, 2016, **15**, (5), pp. 499–507

[7] Hiasat, A.: 'New designs for a sign detector and a residue to binary convertor', *IEE Proc. G Circ. Dev. Syst.*, 1993, **140**, (4), pp. 247–252

[8] Wang, W., Swamy, M., Ahmad, M., *et al.*: 'A study of the residue-to-binary converters for the three-moduli sets', *IEEE Trans. Circuits Syst. I*, 2003, **50**, (2), pp. 235–243

[9] Hiasat, A.: 'An efficient reverse converter for the three-moduli set $(2^{n+1}-1, 2^n, 2^n-1)$', *IEEE Trans. CAS-II*, 2017, **64**, (8), pp. 962–966

[10] Molahosseini, A., Navi, K., Dadkhah, C., *et al.*: 'Efficient reverse converter designs for the new 4-moduli sets $(2^n-1, 2^n, 2^n, +1, 2^{2n+1}-1)$ and $(2^n-1, 2^{2n}, 2^n, +1, 2^{2n}+1)$ based on new CRTs', *IEEE Trans. Circuits Syst. I*, 2010, **57**, (4), pp. 823–835

[11] Hiasat, A.: 'A residue-to-binary converter for the extended four-moduli set $\{2^n-1, 2^n+1, 2^{2n}+1, 2^{2n+p}\}$', *IEEE Trans. Very Large Scale Integ. Syst.*, 2017, **25**, (7), pp. 2188–2192

[12] Ahmadifar, H., Jaberipur, G.: 'A new residue number system with 5-moduli set: $(2^{2q}, 2^q \pm 3, 2^q \pm 1)$', *Comp. J.*, 2015, **58**, (7), pp. 1548–1565

[13] Pettenghi, H., Chaves, R., Sousa, L.: 'RNS reverse converters for moduli sets with dynamic ranges up to $(8n+1)$-bits', *IEEE Trans. Circuits Syst. I*, 2013, **60**, (6), pp. 1487–1500

[14] Hiasat, A.: 'A reverse converter and sign detectors for an extended RNS five moduli se', *IEEE Trans. Circ. Syst. TCAS-I*, 2017, **64**, (1), pp. 111–121

[15] Hiasat, A.: 'A suggestion for a fast residue multiplier for a family of moduli of the form $(2^n-(2^p \pm 1))$', *Comp. J.*, 2004, **47**, (1), pp. 93–102

[16] Pettenghi, H., Cotofana, S., Sousa, L.: 'Efficient method for designing modulo $(2^n \pm k)$ multipliers', *J. Circ. Syst. Comp.*, 2014, **23**, (1), pp. 1–20

[17] Zimmermann, R.: 'Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication'. Proc. 14th IEEE Symp. Computational Arithmetic, April 1999, pp. 158–167

[18] Kalamboukas, L., Nikolos, D., Efstathiou, C., *et al.*: 'High-speed parallel-prefix modulo $2^n - 1$ adders', *IEEE Trans. Comput.*, 2000, **49**, (7), pp. 673–680

[19] Vergos, H.T., Efstathiou, C., Nikolos, D.: 'Diminished-one modulo $2^n + 1$ adder design', *IEEE Trans. Comput.*, 2002, **51**, (12), pp. 1389–1399

[20] Dimitrakopoulos, G., Nikolos, D.: 'High-speed parallel-prefix VLSI ling adders', *IEEE Trans. Comput.*, 2005, **54**, (2), pp. 225–231

[21] Patel, R.A., Benaissa, M., Boussakta, S.: 'Fast parallel-prefix architectures for modulo $2^n - 1$ addition with a single representation of zero', *IEEE Trans. Comput.*, 2007, **56**, (11), pp. 1484–1492

[22] Jaberipur, G., Parhami, B.: 'Unified approach to the design of modulo- $2^n \pm 1$ adders based on signed-LSB representation of residues'. Proc. 19th IEEE Symp. Computer Arithmetic, April 2009, pp. 57–64

[23] Vergos, H.T., Dimitrakopoulos, G.: 'On modulo $2^n + 1$ adder design', *IEEE Trans. Comput.*, 2012, **61**, (2), pp. 173–186

[24] Bayoumi, M., Jullien, G., Miller, W.: 'A VLSI implementation of residue adders', *IEEE Trans. Circuits Syst.*, 1987, **34**, (3), pp. 284–288

[25] Dugdale, M.: 'VLSI implementation of residue adders based on binary adders', *IEEE Trans. Circuits Syst. II Analog Digit. Signal Process.*, 1992, **39**, (5), pp. 325–329

[26] Piestrak, S.J.: 'Design of residue generators and multioperand modular adders using carry-save adders', *IEEE Trans. Comput.*, 1994, **43**, (1), pp. 68–77

[27] Hiasat, A.: 'High-speed and reduced-area modular adder structures for RNS', *IEEE Trans. Comput.*, 2002, **51**, (1), pp. 84–89

[28] Vergos, H., Efstathiou, C.: 'On the design of efficient modular adders', *J. Circuits Syst. Comput.*, 2005, **14**, (5), pp. 965–972

[29] Patel, R., Benaissa, M., Boussakta, S.: 'Novel power-delay-area-efficient approach to generic modular addition', *IEEE Trans. Comput.*, 2007, **56**, (6), pp. 1279–1292

[30] Jaberipur, G., Parhami, B., Nejati, S.: 'On building general modular adders from standard binary arithmetic components'. Proc. 45th Asilomar Conf. Signals Systems and Computers, 2011, pp. 6–9

[31] Matutino, P.M., Pettenghi, H., Chaves, R., *et al.*: 'RNS arithmetic units for modulo $(2^n \pm k)$'. Proc. Euromicro Conf. Digital System Design, September 2012, pp. 795–802

[32] Low, J., Chang, C.-H.: 'A new approach to the design of efficient residue generators for arbitrary moduli', *IEEE Trans. Circ. Syst. TCAS-I*, 2013, **60**, (9), pp. 2366–2374

[33] Patel, R., Benaissa, M., Boussakta, S.: 'Fast modulo $2^n-(2^{n-2}+1)$ addition: a new class of adder for RNS', *IEEE Trans. Circ. Syst. I.*, 2007, **56**, (6), pp. 1279–1292

[34] Matutino, P.M., Chaves, R., Sousa, L.: 'Arithmetic units for RNS moduli $(2^n - 3)$ and $(2^n + 3)$ operations'. Proc. 13th Euromicro Conf. Digital System Design: Architecture, Methods and Tools (DSD), September 2010, pp. 243–246

[35] Ma, S., Hu, J.-H., Wang, C.-H.: 'A novel modulo adder for 2^n-2^k-1 residue number system', *IEEE Trans. Circ. Syst.*, 2013, **60**, (11), pp. 2962–2972
[36] Ladner, E., Fischer, M.J.: 'Parallel prefix computation', *J. ACM*, 1980, **27**, (4), pp. 831–838
[37] Sklansky, J.: 'Conditional sum addition logic', *IRE Trans. Electron. Comput.*, 1960, **9**, (6), pp. 226–231
[38] Kogge, P.M., Stone, H.S.: 'A parallel algorithm for the efficient solution of a general class of recurrence equations', *IEEE Trans. Comput.*, 1973, **22**, (8), pp. 786–792
[39] Brent, R.P., Kung, H.T.: 'A regular layout for parallel adders', *IEEE Trans. Comput.*, 1982, **31**, (3), pp. 260–264
[40] Tyagi, A.: 'A reduced area scheme for carry-select adders', *IEEE Trans. Comp.*, 1993, **42**, (10), pp. 1163–1170
[41] Alioto, M., Di Cataldo, G., Palumbo, G.: 'Optimized design of high fan-in multiplexers using tri-state buffers', *IEEE Trans. Circ. Syst. I*, 2002, **49**, (10), pp. 1500–1505
[42] Burch, R., Najm, F., Yang, P., *et al.*: 'A monte carlo approach for power estimation', *IEEE Trans. Very Large Scale Integr. Syst.*, 1993, **1**, (1), pp. 63–71