Circuits Syst Signal Process (2017) 36:1224–1246
DOI 10.1007/s00034-016-0354-z

Sign Detection and Number Comparison on RNS 3-Moduli Sets $\{2^n - 1, 2^{n+x}, 2^n + 1\}$

Leonel Sousa[1] · **Paulo Martins**[1]

Received: 21 October 2015 / Revised: 8 June 2016 / Accepted: 9 June 2016 /
Published online: 18 June 2016

Abstract Number comparison, sign identification and overflow detection are important operations, especially for digital signal processing, but hard to perform using the residue number system (RNS). In this paper, a new method is proposed for sign identification and number comparison based on an optimized version of the mixed radix conversion for the augmented 3-moduli sets $\{2^n + 1, 2^n - 1, 2^{n+x}\} (0 \leq x \leq n)$. Notably, most of the computations are directly performed on the moduli channels, thus allowing to easily adapt this new method to any RNS processor. Accordingly, this paper proposes an efficient unified very large scale integration architecture based on the presented methodology, which can be used not only to design application specific integrated circuits (ASICs) but also to configure field-programmable gate arrays (FPGAs). The implementation results that were obtained using 65 nm CMOS technologies show that the proposed architecture provided comparators that are more efficient than the related state of the art, by considering as a figure of merit the area time product. More specifically, the considered ASIC and FPGA implementations provide relative improvements in the efficiency of up to 57 and 38 %, respectively. The experimental assessment also shows that the power consumption of the proposed

This work was supported by national funds through FCT (Fundação para a Ciência e a Tecnologia) under project UID/CEC/50021/2013 and by the Ph.D. Grant with reference SFRH/BD/103791/2014.

Electronic supplementary material The online version of this article (doi:10.1007/s00034-016-0354-z) contains supplementary material, which is available to authorized users.

✉ Paulo Martins
paulo.sergio@netcabo.pt

Leonel Sousa
las@inesc-id.pt

[1] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Rua Alves Redol, 9, 1000-029 Lisbon, Portugal

circuits is significantly lower than the related state of the art, with relative reductions of up to 50 %.

Keywords Residue number systems · Mixed radix conversion · Sign detection · Number comparison

1 Introduction

An RNS allows the decomposition of a large integer number into a set of smaller integers, called residues. For these residues, calculations are independent and can be performed in parallel. This type of number representation introduces parallelism in digital computer arithmetic [16], which is quite useful, for example, in cryptography [1] and digital signal processing applications [19]. However, while addition, subtraction and multiplication are operations easily and directly performed in parallel on the residues, other arithmetic operations are difficult to implement in RNS, such as reverse conversion, scaling, magnitude comparison, and sign and overflow detection [16].

Several of those operations that are difficult to implement in RNS are fundamental to develop processors with practical interest. For example, not only sign and overflow detection are important in general for fixed-point arithmetic but also number comparison is often required namely for several nonlinear procedures, such as median and rank-order filtering [2]. In some cases, mixed RNS-binary architectures have been adopted to partially overcome these difficulties, as for example for adaptive filtering, when large variable finite impulse response (FIR) filters are implemented in RNS and the adaptation algorithm is implemented on weighted binary systems, in order to avoid inefficient RNS scaling operations [3].

The traditional approaches for RNS sign detection and comparison are supported on techniques for RNS reverse conversion, namely the Chinese remainder theorem (CRT) and the mixed radix conversion (MRC) [22]. Concretely, numbers are converted from RNS to a positional representation system where the comparison operation can be efficiently computed. This approach was followed in [9]. As mentioned in detail later in this paper, these approaches either require large modular operations or sequential processing.

Several different approaches have been tried to perform RNS comparison, mostly for unsigned numbers. Methods based on new properties of the CRT have been proposed to overcome the drawbacks referred above. The architecture of [6] targets the binary conversion, but it restricts the dynamic range of the input, and thus cannot be adapted to integer comparison. [29] applies the CRT II in a divide-and-conquer approach for RNS number comparison, mainly targeting software-based implementations. A general method for RNS magnitude comparison based on a mixed radix form of the CRT is proposed in [4]. Although this approach was applied in [4] to the 3-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, it can be used to perform number comparison for other RNS moduli sets. Other works have tried to mitigate the effects of converting the numbers to the mixed radix system (MRS). In [13,27], intermediary computations of the conversion to the MRS are exploited to perform number comparison and avoid the negative impacts of performing a full conversion. Nevertheless, these architectures require a large amount of comparators, which decrease the effectiveness of the design.

Various other methods have been proposed to perform RNS number comparison. A method specifically derived for odd dynamic ranges has been proposed in [20]; if $A \geq B$, we have that $A - B \bmod M \bmod 2 = A + B \bmod 2$, otherwise $A - B \bmod M \bmod 2 = A + B + M \bmod 2$. However, the restrictions imposed on the dynamic range (DR) parity prevents the application of this method to most of the hitherto proposed moduli sets, namely the ones that include a power of two modulus. Some other methods have adopted 'core functions' [12] or 'diagonal functions' [7,18], to obtain a magnitude order of the numbers for RNS comparison. In [7] the function was computed using a suitable extra modulus, which can be inserted in the moduli set to avoid redundancy, and [18] does not require any special modulus. The technique of using monotonic functions to perform RNS-to-binary conversion was introduced in [14]. While interesting from a theoretical point of view, this technique requires the use of expensive dividers and thus is costly in terms of area and delay.

The works [21,24,30] can be highlighted for RNS sign identification, which can be considered a particular case of RNS number comparison. The former is based on the MRS and supports augmented 3-moduli sets as the ones targeted in this paper. [24] is supported on the CRT II, but is restricted to the traditional $\{2^n - 1, 2^n, 2^n + 1\}$ moduli set, and the sign detector of [30] is underpinned by the moduli set $\{2^{n+1} - 1, 2^n - 1, 2^n\}$. The particular properties of these moduli sets [10] have made them the most adopted, not only for sign identification and number comparison but also for scaling [23]. However, they only cover a limited DR. Sets providing larger DRs have been proposed, some with higher number of moduli [15,17], sometimes by extending the moduli beyond related power of two moduli [11], and also with 3-moduli sets with the augmented exponent of the power of two modulo [5].

In this paper, we improve upon and extend previous works on sign identification [21,24], to design efficient unified arithmetic units for performing both sign detection and comparison on the augmented 3-moduli sets $\{2^n - 1, 2^{n+x}, 2^n + 1\}(0 \leq x \leq n)$. Most of the computation is directly performed in the RNS channels. The main contributions are summarized as follows:

- a novel and effective algorithm for performing number comparison and sign identification, which can be applied also for overflow detection, directly on the RNS residues, with a quite low overhead;
- a unified architecture proposed to perform efficient RNS number comparison and sign identification in hardware;
- the description of the RNS comparators implemented in ASIC and FPGA are made publicly available; this description can be considered a third contribution since it is fundamental for showing the superiority of the proposed architectures and circuits when compared to the state of the art; it is worth mentioning that it also facilitates the evaluation of the relative performance of future proposals.

This paper is organized as follows. Section 2 provides the background that supports the work presented in this paper. In Sect. 3, algorithms are proposed for number comparison and sign identification, which are used in Sect. 4 to design VLSI architectures for number comparison and sign identification. In Sect. 5, an architecture supported on a direct application of the MRS will be presented, which will be used as a baseline architecture for the assessment of the proposed architecture, and Sect. 6 presents the main characteristics of the related state of the art. Section 7 assesses and experimentally evaluates the efficiency of the proposed, the baseline and state-of-the-art RNS arithmetic units. Final conclusions are drawn in Sect. 8.

2 Background

The following notation is adopted in this paper, considering an RNS based on a moduli set $\{m_1, m_2, \ldots, m_N\}$.

- For an n-bit natural binary representation of a generic value α_i, bits are referred from the most significant bit (MSB) to the least significant bit (LSB) as $\alpha_{i(n-1)}, \cdots, \alpha_{i(0)}$; and the sequence of bits from k to j is represented as $\alpha_{i(k:j)}$.
- x_i denotes the residue of X for m_i: x_i is the least positive remainder of the division of X by m_i ($x_i = \langle X \rangle_{m_i}$), for which the n-bit natural binary representation is $x_{i(n-1)}, \cdots, x_{i(0)}$.
- $\left\langle m_j^{-1} \right\rangle_{m_i}$ is the multiplicative inverse of m_j with respect to m_i, i.e.

$$\left\langle m_j \times m_j^{-1} \right\rangle_{m_i} = 1.$$

- The designation 'channel m_i' is adopted to refer the set of modulo m_i operations.
- For a bit b, we use $\bar{b}$ to denote $1 - b$. The bitwise logical AND and OR operations are represented by the symbols $\wedge$ and $\vee$, respectively.

$M = \prod_{i=1}^{N} m_i$ different numbers can be individually represented in RNS when $m_i (1 \leq i \leq N)$ are pairwise co-prime numbers. An unsigned integer X within the DR $(0 \leq X < M)$ is represented in RNS by the N-tuple of residues $(x_1, x_2, \cdots x_N)$. For representing signed numbers in RNS, the DR is divided in two halves: the lower half is representative of positive numbers while the upper half is reserved for negative numbers. For an even M, which is the case for the moduli set $\{2^n - 1, 2^{n+x}, 2^n + 1\}$, a positive number X is defined in the range $0 \leq X < \frac{M}{2}$ (0 is considered a positive number). The additive inverse of X, $-X$, is represented by the complement of X, which corresponds to $Y = M - X$.

An unsigned integer number X is mapped into the RNS by computing the residues x_i for each channel $(1 \leq i \leq N)$. To represent negative integers, we follow the definition: $-X = M - X (\text{mod } M)$, for $0 < X \leq \frac{M}{2}$, which is equal to $(\langle m_1 - x_1 \rangle_{m_1}, \langle m_2 - x_2 \rangle_{m_2}, \ldots, \langle m_N - x_N \rangle_{m_N})$.

The RNS reverse conversion is a more complex operation that requires not only to apply modular arithmetic but also to combine residues from the different channels. Considering the two main different approaches used to perform RNS reverse conversion, the CRT allows parallel computation of the terms of a sum, but it requires large modulo M arithmetic [19], while the MRS applies modular arithmetic with only the width of the RNS channels, but it requires a sequential computation [31]. By adopting the MRS, an integer X can be obtained from its residues $(x_1, x_2, \ldots, x_N)$ by computing the mixed radix digits step by step (1) and then weighing the MRS digits using (2) to obtain X.

$$\begin{aligned}
d_1 &= x_1 \qquad (1)\\
d_2 &= \left\langle (x_2 - d_1) \times \left\langle m_1^{-1} \right\rangle_{m_2} \right\rangle_{m_2}\\
d_3 &= \left\langle \left((x_3 - d_1) \times \left\langle m_1^{-1} \right\rangle_{m_3} - d_2 \right) \times \left\langle m_2^{-1} \right\rangle_{m_3} \right\rangle_{m_3}\\
&\cdots\\
X &= d_1 + m_1 \times (d_2 + m_2 \times (d_3 + m_3 \times (\cdots))) \ . \qquad (2)
\end{aligned}$$

Moreover, for signed integers, M should be subtracted from X when X is in the range $\frac{M}{2} \leq X < M$:

$$\tilde{X} = \begin{cases} X & \text{if } 0 \leq X < \frac{M}{2} \\ X - M & \text{if } \frac{M}{2} \leq X < M \end{cases} \qquad (3)$$

2.1 Properties of Powers of Two-Related 3-Moduli

Properties of modular arithmetic for related powers of two are widely exploited in this paper, namely the useful, simple to prove, equalities expressed in Property 1.

Property 1

$$\begin{aligned}
\langle -x_i \rangle_{2^n-1} &= \overline{x}_{i(n-1)}, \ \ldots, \ \overline{x}_{i(0)}\\
\left\langle 2^n \times x_i \right\rangle_{2^n+1} &= \langle -x_i \rangle_{2^n+1}\\
\left\langle 2^k \times x_i \right\rangle_{2^n-1} &= x_{i(n-k-1)}, \ \ldots, \ x_{i(0)}, \ x_{i(n-1)}, \ \ldots, \ x_{i(n-k)}
\end{aligned}$$

This paper takes advantage of the characteristics of the moduli set $\{m_1 = 2^n + 1, m_2 = 2^n - 1, m_3 = 2^{n+x}\}(0 \leq x \leq n)$ to efficiently perform number comparison based on the MRS approach. The multiplicative inverses for computing the MRS on the augmented moduli set are provided in (4), (5) and (6).

$$\left\langle m_1^{-1} \right\rangle_{m_2} = \left\langle [2^n + 1]^{-1} \right\rangle_{2^n-1} = \left\langle 2^{n-1} \right\rangle_{2^n-1} \qquad (4)$$

$$\left\langle m_1^{-1} \right\rangle_{m_3} = \left\langle [2^n + 1]^{-1} \right\rangle_{2^{n+x}} = \left\langle -2^n + 1 \right\rangle_{2^{n+x}} \qquad (5)$$

$$\left\langle m_2^{-1} \right\rangle_{m_3} = \left\langle [2^n - 1]^{-1} \right\rangle_{2^{n+x}} = \left\langle -2^n - 1 \right\rangle_{2^{n+x}} \qquad (6)$$

Table 1 Sign of operands (S_A, S_B) and result S_C ($C = A - B$) for the purpose of comparison (COMP)

S_A	1	0	1	1	0	0
S_B	0	1	1	1	0	0
S_C	*	*	1	0	1	0
COMP	$A < B$	$A > B$	$A < B$	$A \geq B$	$A < B$	$A \geq B$
	Ls	**Gr**	**Ls**	**GrEq**	**Ls**	**GrEq**

The sign is represented by a single bit; the value 1 is used for '−' and 0 for '+'; '*' designates do not care

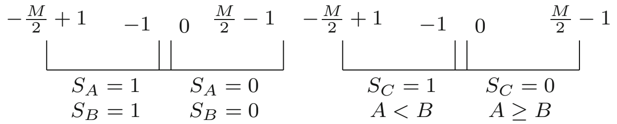


Fig. 1 Representation of signed numbers A and B modulo M to be compared, and their difference C for $S_A = S_B$

3 RNS Number Comparison and Sign Detection

Since RNS is not a weighted number representation, it is hard to establish a monotone function to map the residues into integers, in order to perform magnitude comparison. However, the comparison of any two numbers can be performed by subtracting them, and analysing the signs (S_A, S_B, and S_C) of the operands (A, B) and result (C), whenever only the two conditions $A < B$ and $A \geq B$ are targeted, as presented in Table 1. In the first two columns in Table 1, the values of S_A and S_B are different so the subtraction result is not required to conclude that the largest number is the one with a positive value ($S_A = 0$ or $S_B = 0$). The remaining columns in Table 1 correspond to the four different cases where $S_A = S_B$. For these cases, the operation $C = A - B$ is performed in the channels and the values of S_A, S_B, and S_C are used to perform a comparison (COMP) according to Table 1. When the conditions $A = B$ and $A > B$ are to be individually considered, it has to be additionally tested if the result takes the value zero.

The distribution of RNS signed integers within the DR is depicted in Fig. 1. For the moduli sets targeted in this paper, since M is even, negative numbers are mapped exactly into the highest half segment of the whole range $S = 1$. The result of the subtraction is depicted in Fig. 1 considering that A and B have the same sign ($S_A = S_B$). It is noticeable from the first two columns in Table 1 that when the signs of the operands differ, the comparison output does not depend on the sign of the result (S_C). Therefore, no consequences arise if the result is not correctly represented, *i.e.* $C \notin]\frac{-M}{2}, \frac{M}{2}[$.

Therefore, the comparison can be performed directly in the RNS domain by identifying the signs of $A(S_A)$, $B(S_B)$ and $C = A - B(S_C)$ whenever $S_A = S_B$. While subtraction is an operation efficiently applied in parallel in the RNS channels, the same is not true for the identification of the signs of numbers. In this section, a method based on a mixed radix representation is proposed to overcome this drawback of the RNS representation and perform number comparison in this domain.

3.1 Sign Detection

The identification of the sign of a number is easily performed in radix-based representations, and for a number represented in RNS the simplest way to achieve it is by using a mixed radix system [22]. Instantiating the MRS expression (2) for an integer (R) represented in RNS by the residues r_1, r_2, r_3 on the considered 3-moduli set $\{2^n+1, 2^n-1, 2^{n+x}\}$:

$$\begin{aligned} R &= d_3(2^{2n}-1) + d_2(2^n+1) + d_1 \\ \tilde{R} &= \begin{cases} R & \text{if } 0 \le R < \frac{M}{2} \\ R - M & \text{if } \frac{M}{2} \le R < M \end{cases} \end{aligned} \tag{7}$$

One can see from (7) that since $0 \le R < M$, $\tilde{R}$ will be less than zero when $R \ge \frac{M}{2}$, where $M = 2^{n+x} \times (2^{2n}-1)$:

$$\begin{aligned} \tilde{R} < 0 \Rightarrow &\left(d_3 \times \left(2^n-1\right) + d_2\right) \times (2^n+1) + d_1 \\ \ge &\left(2^{n+x-1} \times \left(2^n-1\right) + 0\right) \times (2^n+1) + 0 \end{aligned}$$

Since $0 \le d_1 < 2^n+1$ and $0 \le d_2 < 2^n-1$, from (2) it can be concluded that:

$$\tilde{R} < 0 \Rightarrow d_3 \ge 2^{n+x-1} \tag{8}$$

A similar reasoning leads to:

$$\tilde{R} \ge 0 \Rightarrow d_3 < 2^{n+x-1} \ , \tag{9}$$

and for:

$$\begin{aligned} \tilde{R} &= 0 \Rightarrow d_3 = d_2 = d_1 = 0 \\ &\Leftrightarrow r_3 = r_2 = r_1 = 0 \ . \end{aligned} \tag{10}$$

The last step missing to complete the sign identification process is the computation of the MRS digits (d_1, d_2, d_3). (11) to (13) are derived from (1) by using the multiplicative inverses provided in (4) to (6). When $x = 0$, we define $d_{2(x-1:0)}$ to be 0.

$$d_1 = r_1 \tag{11}$$

$$d_2 = \left\langle (r_2 - r_1) \times 2^{n-1} \right\rangle_{2^n-1} \tag{12}$$

$$d_3 = \left\langle \left((r_3 - r_1) \times (-2^n+1) - d_2 \right) \times (-2^n-1) \right\rangle_{2^{n+x}}$$

$$= \left\langle (r_1 - r_3) + d_{2(x-1:0)} \times 2^n + d_2 \right\rangle_{2^{n+x}} \tag{13}$$

Although the value of d_3 depends on d_2, it will be shown in the next section that with a small amount of redundant computation, d_3 can be computed independently of d_2. The digits of the mixed radix representation for the three numbers involved in the comparison $(A; B; C = A - B)$ are identified from now on as $(e_1, e_2, e_3; f_1, f_2, f_3; g_1, g_2, g_3)$, respectively.

The method derived for sign detection of RNS numbers is used in the next subsection on an algorithm for RNS number comparison.

3.2 Number Comparison

Algorithm 1 is proposed for RNS number comparison. Two alternative approaches may be adopted to identify the signs of A (S_A) and B (S_B), which are expressed in Algorithm 1: (i) any time an RNS operation is performed, the sign of the result is detected and stored and (ii) the signs of A ($S_A \equiv e_{3(n+x-1)}$) and B ($S_B \equiv f_{3(n+x-1)}$) are computed in the scope of Algorithm 1 for the purpose of number comparison. Option (i) is clearly the most cost-effective since it avoids having three times the same hardware or execution time to perform comparison. First, it should be noted that for the case of multiplication the sign of the product is trivially obtained from the sign of the operands. Moreover, attaching the sign detection circuit to the adders or subtractors may not impose any overhead, neither in time nor in cost. Assuming a single-cycle

Algorithm 1 Comparison of two integers represented in RNS based on the 3-moduli set $\{2^n + 1, 2^n - 1, 2^{n+x}\}$

```
Require: Arithmetic modulo $2^{n+x}$ and $2^n - 1$
1: function COMPARE($a_1, a_2, a_3, b_1, b_2, b_3$)
2:                                        ▷ $S_A \equiv e_{3(n+x-1)}$, $S_B \equiv f_{3(n+x-1)}$ available
3:                  ▷ Otherwise, code up to line 8 is repeated to compute $e_3$ and $f_3$ as well
4:     $(c_1, c_2, c_3) \leftarrow (a_1 - b_1, a_2 - b_2, a_3 - b_3)$          ▷ $C = A - B$
5:     $g_1 \leftarrow c_1$                                                  ▷ radix $2^0$
6:     $g_2 \leftarrow \left\langle (c_2 - c_1) \times 2^{n-1} \right\rangle_{2^n-1}$          ▷ radix $2^n + 1$
7:     $g_3 \leftarrow \langle (c_1 - c_3) + g_{2(x-1:0)} \times 2^n + g_2 \rangle_{2^{n+x}}$    ▷ radix $2^{2n} - 1$
8:                                     ▷ $g_3$ can also be computed in parallel with $g_2$ (17)
9:                                                  ▷ Implement Function in Table 1
10:     if $c_1 = 0 \wedge c_2 = 0 \wedge c_3 = 0$ then
11:         COMP ← Eq                                                     ▷ $A = B$
12:     else
13:         if $(S_A = 1 \wedge S_B = 0) \vee (S_A = S_B \wedge g_{3(n+x-1)} = 1)$ then
14:             COMP ← Ls                                                 ▷ $A < B$
15:         else
16:             if $(S_A = 0 \wedge S_B = 1) \vee (S_A = S_B \wedge g_{3(n+x-1)} = 0)$ then
17:                 COMP ← Gr                                             ▷ $A > B$
18:             end if
19:         end if
20:     end if
21:     return COMP
22: end function
```

processor, the critical path will include the multiplier, whose execution time is greater than that of performing an addition or subtraction plus sign identification. If the comparison operation is implemented as a single instruction, the addition and subtraction units can use the same sign identifier circuit as the comparator, since their execution times will not overlap.

As a matter of simplicity, we assume the approach *i*), which means $S_C \equiv g_{3(n+x-1)}$ has to be computed, but not $e_{3(n+x-1)}$ and $f_{3(n+x-1)}$ that are supposed to be available as inputs of the algorithm. From line 4 to line 8 in Algorithm 1, the residues of the two numbers to be compared are subtracted and the mixed radix digits g_1, g_2 and g_3 are computed. The value of g_3 can be computed by using (13). Nevertheless, this expression will be optimized in the next section to achieve more efficient hardware structures. Given the values of S_A, S_B and S_C, the conditions expressed in Table 1 are checked in lines 10–20 of Algorithm 1. In line 10 of Algorithm 1 the values of the three residues c_i, computed in line 4 of this algorithm, are compared with zero, and when the three are zero the flag A **Eq** B is assigned. For the remaining of Algorithm 1: conditions corresponding to the first, third and fifth columns in Table 1 are tested in line 13, which correspond to the case A **Ls** B when $A < B$; the other conditions are tested in line 16, but in practice this test is not required since it includes all the other conditions, for which the flag A **Gr** B is activated.

4 Proposed VLSI Architectures

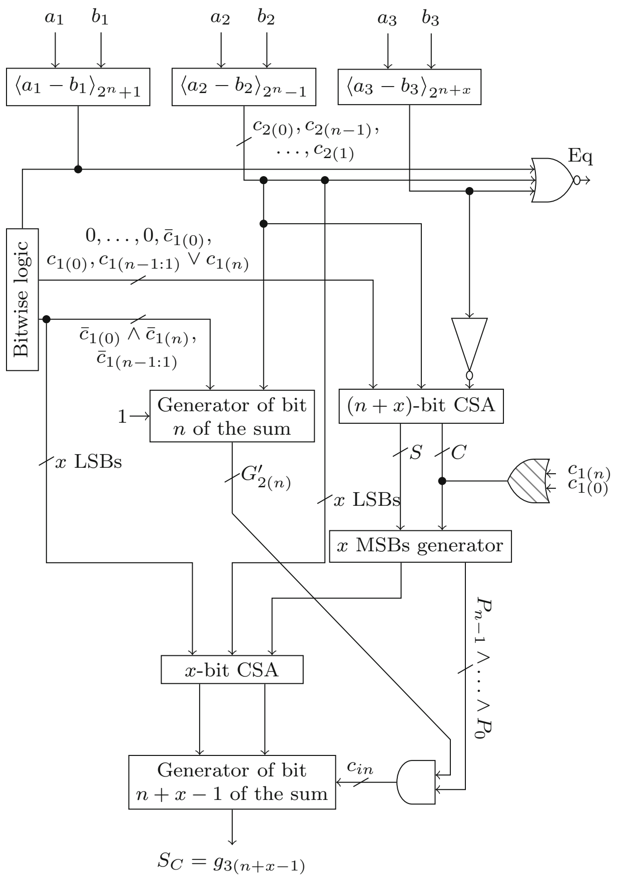
The circuit whose architecture is presented in Fig. 2 performs number comparison by computing Algorithm 1, with some further simplifications in particular for the hardware that computes the MRS digits. The numbers under comparison (A, B) are subtracted in parallel, in the three channels for the computation of $(c_1, c_2, c_3) = (a_1 - b_1, a_2 - b_2, a_3 - b_3)$ and ultimately $g_{3(n+x-1)}$. Herein, we describe the computation of $g_{3(n+x-1)}$ but $e_{3(n+x-1)}$ and $f_{3(n+x-1)}$ can also be similarly computed.

On the channel $2^n + 1$ no further computation is required since $g_1 = c_1$. For computing g_2, we start by noting that the following is valid using Property 1:

$$\begin{aligned}\left\langle -2^{n-1}c_1\right\rangle_{2^n-1} &= \left\langle -2^{n-1}((c_{1(n-1)},\ldots,c_{1(0)}) + c_{1(n)})\right\rangle_{2^n-1}\\ &= \left\langle (\bar{c}_{1(0)},\bar{c}_{1(n-1)},\ldots\bar{c}_{1(1)}) + (2^n-1-2^{n-1})c_{1(n)}\right\rangle_{2^n-1}\\ &= \left\langle (\bar{c}_{1(0)},\bar{c}_{1(n-1)},\ldots\bar{c}_{1(1)}) + (2^{n-1}-1)c_{1(n)}\right\rangle_{2^n-1}\end{aligned}$$

Thus, (12) can be rewritten as:

$$\begin{aligned}g_2 &= \left\langle (c_2-c_1)\times 2^{n-1}\right\rangle_{2^n-1}\\ &= \big\langle (c_{2(0)},c_{2(n-1)},\ldots,c_{2(1)})\\ &\quad + (\bar{c}_{1(0)},\bar{c}_{1(n-1)},\ldots,\bar{c}_{1(1)}) + (2^{n-1}-1)\times c_{1(n)}\big\rangle_{2^n-1}\end{aligned} \tag{14}$$



a_1
b_1
a_2
b_2
a_3
b_3
$\langle a_1 - b_1 \rangle_{2^n+1}$
$\langle a_2 - b_2 \rangle_{2^n-1}$
$\langle a_3 - b_3 \rangle_{2^{n+x}}$
$c_{2(0)}, c_{2(n-1)}, \dots, c_{2(1)}$
Eq
Bitwise logic
$0, \dots, 0, \bar{c}_{1(0)}, c_{1(0)}, c_{1(n-1:1)} \vee c_{1(n)}$
$\bar{c}_{1(0)} \wedge \bar{c}_{1(n)}, \bar{c}_{1(n-1:1)}$
1
Generator of bit n of the sum
$(n+x)$-bit CSA
S
C
$c_{1(n)}$
$c_{1(0)}$
x LSBs
$G'_{2(n)}$
x LSBs
x MSBs generator
$P_{n-1} \wedge \dots \wedge P_0$
x-bit CSA
Generator of bit $n+x-1$ of the sum
c_{in}
$S_C = g_{3(n+x-1)}$

Fig. 2 Optimized architecture proposed for number comparison, the sign of $C = A - B$ is identified as $g_{3(n+x-1)}$

The last two terms in (14) can be fused by using bitwise logic operators instead of the addition. Since $0 \le c_1 < 2^n + 1$, when $c_1(n) = 1$ then $c_1 = 2^n$, and therefore, $\left\langle \overline{c}_{1(0)}, \overline{c}_{1(n-1)} \dots, \overline{c}_{1(1)} \right\rangle_{2^n-1} = \langle 2^n - 1 \rangle_{2^n-1} = 0$. Hence, when $c_{1(n)} = 1$, we have that $\left\langle \overline{c}_{1(0)}, \overline{c}_{1(n-1)}, \dots, \overline{c}_{1(1)} + (2^{n-1} - 1) \times c_{1(n)} \right\rangle_{2^n-1} = 2^{n-1} - 1$, which can be obtained by the expression $\overline{c}_{1(0)} \wedge \overline{c}_{1(n)}, \overline{c}_{1(n-1)}, \dots, \overline{c}_{1(1)}$. When $c_{1(n)} = 0$, this latter expression still produces valid results. Therefore, the value c_2 can be computed with (15), by applying only bitwise logic operators and a modulo $2^n - 1$ carry propagate adder (CPA).

$$\begin{aligned} g_2 &= \langle G_2 \rangle_{2^n-1} \\ G_2 &= (c_{2(0)}, c_{2(n-1)}, \dots, c_{2(1)}) \\ &\quad + (\overline{c}_{1(0)} \wedge \overline{c}_{1(n)}, \overline{c}_{1(n-1)}, \dots, \overline{c}_{1(1)}) \end{aligned} \tag{15}$$

There is only one dependence between channels, from g_2 to g_3. To break this dependence, the value of g_2 in (15) is computed directly in the channel m_3 this time using modulo 2^{n+x} arithmetic (16):

$$g_2 = \left\langle G_2 + G'_{2(n)} - G'_{2(n)} \times 2^n \right\rangle_{2^{n+x}} \tag{16}$$

where $G'_2 = G_2 + 1$ and $G'_{2(n)}$ is the n-th bit of G'_2. When $G_2 \geq 2^n - 1$, since G'_2 will satisfy $G'_2 = G_2 + 1 \geq 2^n$, $G'_{2(n)}$ will take the value of 1. In this case, the end-around carry is added $(+G'_{2(n)})$ while the carry-out bit of the sum G_2 is eliminated $(-G'_{2(n)} \times 2^n)$.

Therefore, g_3 can be computed independently from g_2, by applying (16) and (15) to (13), obtaining (17).

$$\begin{aligned}
g_3 &= \left\langle (c_1 - c_3) + g_{2(x-1:0)} \times 2^n + G_2 + G'_{2(n)} - G'_{2(n)} \times 2^n \right\rangle_{2^{n+x}} \\
&= \langle c_1 + (\overline{c}_{1(0)} \wedge \overline{c}_{1(n)}, \overline{c}_{1(n-1)}, \ldots, \overline{c}_{1(1)}) - c_3 + (c_{2(0)}, c_{2(n-1)}, \ldots, c_{2(1)}) \\
&\quad + g_{2(x-1:0)} \times 2^n - G'_{2(n)} \times 2^n + G'_{2(n)} \rangle_{2^{n+x}} \\
&= \langle Y - c_3 + (c_{2(0)}, c_{2(n-1)}, \ldots, c_{2(1)}) \\
&\quad + g_{2(x-1:0)} \times 2^n - G'_{2(n)} \times 2^n + G'_{2(n)} \rangle_{2^{n+x}}
\end{aligned} \tag{17}$$

where

$$Y = \langle c_1 + (\overline{c}_{1(0)} \wedge \overline{c}_{1(n)}, \overline{c}_{1(n-1)}, \ldots, \overline{c}_{1(1)}) \rangle_{2^{n+x}}. \tag{18}$$

The dependence on g_2 in (17) is confined to $g_{2(x-1:0)}$ and $G'_{2(n)}$, with the latter corresponding to the output of the 'Generator of bit n of the sum' in Fig. 2. The various operands in (17), except the ones derived from bits of g_2 could be added by using Carry Save Adders (CSAs) and a CPA, all $n + x$-bit width binary adders.

Channel 2^{n+x} is, however, overloaded, which represents a penalty both in terms of cost and delay. In order to avoid this penalty, let us simplify the addition of the terms in Y in (17). We first consider the case when $c_{1(n)} = 0$ and afterwards extend the formula for $c_{1(n)} \in \{0, 1\}$:

$$\begin{aligned}
Y|_{c_{1(n)}=0} &= \langle c_1 + (\overline{c}_{1(0)}, \overline{c}_{1(n-1)}, \ldots, \overline{c}_{1(1)}) \rangle_{2^{n+x}} \\
&= \left\langle c_1 + 2^n - 1 - c_{1(0)} 2^{n-1} - \frac{c_1 - c_{1(0)}}{2} \right\rangle_{2^{n+x}} \\
&= \left\langle \left(c_1 - \frac{c_1 - c_{1(0)}}{2} \right) + 2^{n-1} \left(2 - c_{1(0)} \right) - 1 \right\rangle_{2^{n+x}} \\
&= \left\langle \left(\frac{2c_1 - c_1 + c_{1(0)} + c_{1(0)} - c_{1(0)}}{2} \right) + 2^{n-1} \left(2\bar{c}_{1(0)} + c_{1(0)} \right) - 1 \right\rangle_{2^{n+x}} \\
&= \left\langle \left(\frac{c_1 - c_{1(0)}}{2} + c_{1(0)} \right) + 2^n \bar{c}_{1(0)} + 2^{n-1} c_{1(0)} - 1 \right\rangle_{2^{n+x}} \\
&= \left\langle (\overbrace{0, \;\cdots\cdots, \; 0}^{\text{bits } n+x-1,\ldots,n+1}, \bar{c}_{1(0)}, c_{1(0)}, c_{1(n-1)}, \ldots, c_{1(1)}) + c_{1(0)} - 1 \right\rangle_{2^{n+x}}
\end{aligned} \tag{19}$$

Equation (19) will now be extended for a generic $c_{1(n)} \in \{0, 1\}$. When $c_{1(n)} = 1$, it is observed, from (18) that $Y = 2^n + 2^{n-1} - 1$. In that case, it is necessary to force the $n - 1$ LSBs of Y to $(2^{n-1} - 1)$, the $(n - 1)$-th bit to '0' and the (n)-th bit to '1', while all the other $x - 1$ MSBs of Y keep the value '0':

$$\begin{aligned} Y = \langle(&\overbrace{0, \quad \cdots\cdots, \quad 0}^{\text{bits } n+x-1\cdots n+1}, \bar{c}_{1(0)}, c_{1(0)}, c_{1(n-1)} \vee c_{1(n)}, \cdots, c_{1(1)} \vee c_{1(n)}) \\ &+ (c_{1(0)} \vee c_{1(n)}) - 1\rangle_{2^{n+x}} \end{aligned} \tag{20}$$

Adding $\langle -c_3 \rangle_{2^{n+x}} = \langle \bar{c}_3 + 1 \rangle_{2^{n+x}}$ and $c_{2(0)}, c_{2(n-1)}, \ldots, c_{2(1)}$ to Y in (17) to compute g_3, we obtain (21).

$$\begin{aligned} &\left\langle Y - c_3 + (c_{2(0)}, c_{2(n-1)}, \ldots, c_{2(1)}) \right\rangle_{2^{n+x}} \\ &\quad = \langle(\overbrace{0, \quad \cdots\cdots, \quad 0}^{\text{bits } n+x-1\cdots n+1}, \bar{c}_{1(0)}, c_{1(0)}, c_{1(n-1)} \vee c_{1(n)}, \ldots, c_{1(1)} \vee c_{1(n)}) \\ &\qquad + (c_{2(0)}, c_{2(n-1)}, \ldots, c_{2(1)}) + \bar{c}_3 + (c_{1(0)} \vee c_{1(n)}) \rangle_{2^{n+x}} \end{aligned} \tag{21}$$

The first three terms of (21) are added with the $(n + x)$-bit CSA in Fig. 2, whereas the conditional increment derived from the last operand is performed by setting $C_0 = 1$ (corresponding to the LSB of the output carry of the CSA) when $c_{1(0)} \vee c_{1(n)} = 1$, as implemented in Fig. 2 by the OR gate with grey stripes.

For summing the last three terms of (17), we can use (15) and apply Property 1:

$$\begin{aligned} \left\langle g_{2(x-1:0)} \times 2^n \right\rangle_{2^{n+x}} &= \left\langle \langle G_2 \rangle_{2^n - 1(x-1:0)} \times 2^n \right\rangle_{2^{n+x}} \\ &= \left\langle (G_{2(x-1:0)} - G'_{2(n)}(2^n - 1)) \times 2^n \right\rangle_{2^{n+x}} \\ &= \left\langle (G_{2(x-1:0)} + G'_{2(n)}) \times 2^n \right\rangle_{2^{n+x}} \end{aligned} \tag{22}$$

leading to:

$$\begin{aligned} &\left\langle g_{2(x-1:0)} \times 2^n - G'_{2(n)} \times 2^n + G'_{2(n)} \right\rangle_{2^{n+x}} \\ &\quad = \left\langle G_{2(x-1:0)} \times 2^n + G'_{2(n)} \right\rangle_{2^{n+x}} \end{aligned} \tag{23}$$

The value of $G_{2(x-1:0)}$ is computed based on (15) by setting the x LSBs of $c_{2(0)}, c_{2(n-1)}, \ldots, c_{2(1)}$ and $\bar{c}_{1(0)} \wedge \bar{c}_{1(n)}, \bar{c}_{1(n-1)}, \ldots, \bar{c}_{1(1)}$ as two of the inputs of the x-bit CSA in Fig. 2. The third input of this CSA corresponds to the x MSBs of (21), because only the value of the bit $g_{3(n+x-1)}$ is meaningful.

The outputs of the x-bit CSA in Fig. 2 are then combined using an x-bit CPA; this CPA is designated 'Generator of bit $n + x - 1$ of the sum', as only computing the MSB of g_3, corresponding to $g_{3(n+x-1)}$, is required. Moreover, the term $G'_{2(n)}$ should be added to the overall sum. Therefore, the input carry of the latter CPA is set to '1' when the n LSBs of (21) and $G'_{2(n)}$ are '1'; otherwise, it is set to '0'. It should be noted

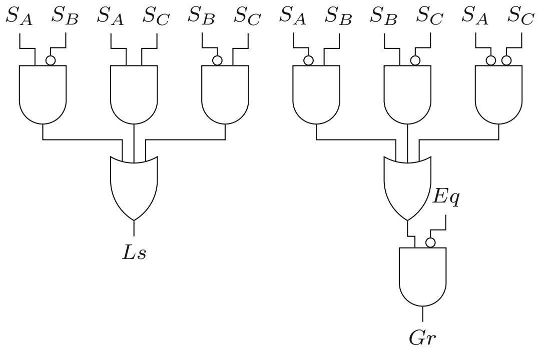


Fig. 3 Derivation of the signal L which is '1' when $A < B$, and G which is '1' when $A > B$

that the n LSBs of (21) are '1' when the value of $P_{n-1} \wedge \ldots \wedge P_0$ (P_i represents the Propagate bit in location i) in Fig. 2 is '1'.

Furthermore, to take into account the equality test, it is evaluated whether the outputs of the subtractors in the three channels are all equal to '0', using a NOR gate for all bits (see generation of Eq in Fig. 2). Finally, the scheme in Fig. 3, which takes as input $S_C = g_{3(n+x-1)}$, S_A, S_B and Eq, is used to produce the signals Gr and Ls that are '1' when $A > B$ and $A < B$, respectively.

In the next section, we will describe a straightforward implementation of both a sign identifier and a comparator exploiting the MRS. Next, the related state of the art will be described. Finally, the performance of the proposed sign identifier and number comparator will be compared with the baseline architecture and related art in Sect. 7.

5 Baseline Architecture

We start by considering a straightforward design of a sign identifier using the MRS. With this architecture a number is converted to the MRS, and the mixed radix digits are compared with $\frac{M}{2}$, i.e. half of the dynamic range, to check if a number is negative or positive. Concretely, a number A was represented by the residues $a_i = \langle A \rangle_{m_i}$ for $i = \{1, 2, 3\}$, and $m_1 = 2^{n+x}$, $m_2 = 2^n + 1$, and $m_3 = 2^n - 1$. The mixed radix digits of A, $d_i^{(A)}$, are produced such that $A = (d_3^{(A)}(2^n + 1) + d_2^{(A)})2^{n+x} + d_1^{(A)}$, and have the following values:

$$\begin{aligned} d_1^{(A)} &= a_1 \\ d_2^{(A)} &= \langle 2^{2n-x}(a_1 - a_2) \rangle_{2^n+1} \\ d_3^{(A)} &= \langle 2^{n-1}(2^{2n-x}(a_3 - a_1) - d_2^{(A)}) \rangle_{2^n-1} \end{aligned} \tag{24}$$

The order of the moduli was chosen arbitrarily, and it will be shown that this is not a choice as good as the one adopted for the proposed architecture. In fact, the evaluation of the sign of a number is now a much more complex operation. Since $L = \frac{M}{2}$ is represented by $d_3^{(L)} = 2^{n-1} - 1$, $d_2^{(L)} = 2^{n-1}$ and $d_1^{(L)} = 2^{n+x-1}$, one needs to compute all three mixed radix digits of A, $d_3^{(A)}$, $d_2^{(A)}$, and $d_1^{(A)}$ instead of just

$d_3^{(A)}$ as with the proposed architecture. Afterwards, to check whether A is a negative number, one needs to check whether $d_3^{(A)} > d_3^{(L)}$, or $d_3^{(A)} = d_3^{(L)}$ and $d_2^{(A)} > d_2^{(L)}$, or $d_3^{(A)} = d_3^{(L)}, d_2^{(A)} = d_2^{(L)}$ and $d_1^{(A)} \geq d_1^{(L)}$.

Similarly, we consider a straightforward implementation of a comparator for unsigned integers using the MRS. With this design, two numbers are converted from RNS to the MRS, and the mixed radix digits are compared from the most significant to the least so as to determine which number is the largest. As soon as the mixed radix digits of the two numbers to be compared, A ($d_1^{(A)}, d_2^{(A)}$ and $d_3^{(A)}$) and B ($d_1^{(B)}, d_2^{(B)}$ and $d_3^{(B)}$), are available, they are compared as follows: if $d_3^{(A)} > d_3^{(B)}$, or $d_3^{(A)} = d_3^{(B)}$ and $d_2^{(A)} > d_2^{(B)}$, or $d_3^{(A)} = d_3^{(B)}, d_2^{(A)} = d_2^{(B)}$ and $d_1^{(A)} > d_1^{(B)}$, then $A > B$; if $d_3^{(A)} = d_3^{(B)}, d_2^{(A)} = d_2^{(B)}$ and $d_1^{(A)} = d_1^{(B)}$, then $A = B$; and if none of the before mentioned conditions is satisfied, then $A < B$. We will consider this design as a baseline for the evaluation of the performance of the proposed architecture.

6 Related State of the Art

The work in [21] for sign identification is based on a similar approach to what is herein presented, except for the computation of d_2, which was not integrated in that of d_3 and therefore is less efficient. In [24], restricted to the traditional 3-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, the New CRT II formula [28] is distorted so that computing the sign of X is the same as computing the MSB of $\langle t' \rangle_{2^n}$, where t' is as follows:

$$\begin{aligned} Y &= \langle 2^{n-1}(x_1 - x_3) \rangle_{2^n - 1} \\ t' &= x_3 - x_2 + Y \end{aligned}$$

with $\{m_1 = 2^n - 1, m_2 = 2^n, m_3 = 2^n + 1\}$.

For RNS number comparison, [18] extends [7] by establishing a 'diagonal function' without the need of any extra modulus. The function F_I is defined so that for two unsigned numbers $A, B \in [0, M[, A < B \iff (F_I(A) < F_I(B)) \; or \; (F_I(A) = F_I(B) \;\text{ and }\; a_i < b_i, \;\text{ for any }\; i \in I)$. The subscript I stands for the indexes of a subset of the moduli set, and J is used to denote the indexes of the complementary subset. For $I = \{1, 2\},\; J = \{3\}$ with $\{m_1 = 2^n + 1, m_2 = 2^n - 1, m_3 = 2^{n+x}\}$, F_I is instantiated as follows:

$$\begin{aligned} M_I &= M_1 + M_2 = 2^{2n+x+1} \;;\; M_3 = 2^{2n} - 1 \\ -\beta_1 &= \left\langle m_1^{-1} \right\rangle_{M_I} = 2^{2n+x+1} - 2^{3n} + 2^{2n} - 2^n + 1 \\ -\beta_2 &= \left\langle m_2^{-1} \right\rangle_{M_I} = 2^{2n+x+1} - 2^{3n} - 2^{2n} - 2^n - 1 \\ S_{INV} &= \sum_{i=1}^{2} \left\langle m_i^{-1} \right\rangle_{M_I} = 2^{2n+x+2} - 2^{3n+1} - 2^{n+1} \\ \beta_3 &= \left\langle M_3 \times S_{INV} \times \left\langle M_3^{-1} \right\rangle_{m_3} \right\rangle_{M_I} = 2^{2n+x+1} - 2^{n+1} \end{aligned}$$

$$
\begin{aligned}
F_I(X) &= \left\langle \sum_{i=1}^{3} \beta_i \times x_i \right\rangle_{M_I} \\
&= \Big\langle (2^{3n} - 2^{2n} + 2^n - 1)x_1 \\
&\quad +(2^{3n} + 2^{2n} + 2^n + 1)x_2 - 2^{n+1}x_3 \Big\rangle_{2^{2n+x+1}} \qquad (25)
\end{aligned}
$$

Other subsets I and J can be considered, but at the cost of more complex reduction operations. For example, if $I = \{3\}$ and $J = \{1, 2\}$, F_I is computed with modulo $M_I = 2^{2n} - 1$ arithmetic, and for $I = \{1, 3\}$ and $J = \{2\}$ the modulo is $M_I = 2^{2n+x} - 2^{n+x} + 2^{2n} - 1$.

Other approach for RNS magnitude comparison was proposed in [4]. It was supported on the 3-moduli set $\{m_1 = 2^n + 1, m_2 = 2^n, m_3 = 2^n - 1\}$ but can be extended to the augmented moduli set $\{m_1 = 2^n + 1, m_2 = 2^{n+x}, m_3 = 2^n - 1\}$. It relies on the computation of the mixed radix digits $\alpha_A, \alpha_B, \beta_A$ and β_B so that (26) holds, by comparing the unsigned numbers A and B in the augmented RNS moduli set.

$$
\begin{aligned}
\alpha_X &= \left\langle \left\lfloor 2^{2n-x-1}(x_3 - x_1) + \frac{x_1 - x_2}{2^x} + \frac{x_2 - x_1}{2^{n+x}} \right\rfloor \right\rangle_{2^n-1} \\
\beta_X &= \left\langle (2^n - 1)x_1 - (2^n - 1)x_2 \right\rangle_{2^{n+x}} \\
A &= \alpha_A(2^n + 1)2^{n+x} + \beta_A(2^n + 1) + x_1 \\
B &= \alpha_B(2^n + 1)2^{n+x} + \beta_B(2^n + 1) + y_1 \qquad (26)
\end{aligned}
$$

The condition $A > B$ is verified *iff*: $\alpha_A > \alpha_B$; or $\alpha_A = \alpha_B$ and $\beta_A > \beta_B$; or $\alpha_A = \alpha_B$, $\beta_A = \beta_B$ and $x_1 > y_1$.

To evaluate the performance and the efficiency of the algorithm and architectures proposed in this paper, the state-of-the-art methods in [21,24] are used as references for sign identification, and in [4,18] for RNS number comparison.

7 Assessment and Experimental Evaluation

In this section, we start by presenting a comparative evaluation of the proposed and baseline sign identifiers as well as those from the literature, and argue that the proposed architecture is specially tuned for the RNS comparator. Similarly, the efficiency of the proposed architectures for RNS number comparison is evaluated and compared with the baseline architecture as well as other proposals in the literature. To derive a technology-independent evaluation of the performance of the RNS comparators, we adopt the commonly applied simple unit-gate (U-G) model [25], complemented with experimental results from ASIC and FPGA implementations for RNS number comparison.

With the U-G model, a two-input monotonic gate, such as a NAND gate, is considered to have one unit of area and one unit of delay. An XOR gate is deemed to require two units of area and to impose a delay of two units. The area and delay of an inverter are a negligible fraction of a unit, and hence, it is assumed to require zero units of area and delay. A full adder (FA) has seven units of area and four units of delay. Mainly binary adders, and $2^n - 1$ and $2^n + 1$ modulo adders are required to implement the architectures proposed in Sect. 4, and in the related state of the art. The adopted Ling modulo $2^n - 1$ adders [8] require $3n\lceil\log_2 n - 1\rceil + 12n$ units of area and impose a delay of $2\lceil\log_2 n - 1\rceil + 3$ units. Modulo $2^n + 1$ adders for the diminished-1 representation [26] require $4.5n\lceil\log_2 n\rceil + 0.5n + 6$ units of area and impose a delay of $2\lceil\log_2 n\rceil + 3$ units. For binary adders, $1.5n\lceil\log_2 n\rceil + 5n$ units of area and $2\lceil\log_2 n\rceil + 3$ units of time are accounted [26].

7.1 Technology-Independent Evaluation

We start by discussing the U-G model of the baseline sign identifier. We apply some simplifications so as to reduce the complexity of the final expressions and enhance their intelligibility, without compromising the derived conclusions. For the implementation of the architecture described in Sect. 5, the computation of the value of $d_2^{(A)}$ requires the use of 3 stages of modulo $2^n + 1$ CSAs, and a modulo $2^n + 1$ CPA. The computation of $d_3^{(A)}$ required 3 additional modulo $2^n - 1$ CSA stages, and a modulo $2^n - 1$ CPA. It should be noted that for these estimations we have to consider that a_1 and a_2 correspond to two n-bit operands and a constant correction term has to be added. The condition of $d_3^{(A)} > d_3^{(L)}$ requires solely the checking of the $(n-1)^{\text{th}}$ bit of $d_3^{(A)}$. In contrast, two n-bit equality testers were used to check if $d_3^{(A)} = d_3^{(L)}$ and $d_2^{(A)} = d_2^{(L)}$. The logic value of $d_2^{(A)} > d_2^{(L)}$ is true when the previous equality test fails ($d_2^{(A)} \neq d_2^{(L)}$) and the $(n-1)$th bit of $d_2^{(A)}$ is 1 and false otherwise. Finally, the condition $d_1^{(A)} \geq d_1^{(L)}$ can be inferred from the $(n+x-1)^{\text{th}}$ bit of $d_1^{(A)}$. Furthermore, we assume that the equality testers are composed by an array of n XNOR gates, and a tree-like structure of AND gates. Thus, we approximate the area of the baseline sign identifier by $7.5n\lceil\log_2(n)\rceil + 60.5n + 6$, and its delay by $5\lceil\log_2(n)\rceil + 28$.

Regarding the proposed sign identifier (Fig. 2) the area of the generators of bits $n + x - 1$ and n of the sum, and the block to generate the x MSBs of the sum and to test whether the first n bits of the result are '1' are approximated by half the area of the corresponding CPAs of $n + x$, n and x bits, respectively. The two CSAs used in Fig. 2 require an area of $7(n + 2x)$; n gates are required for the bitwise logic block, and an AND and OR gates to generate the carries. Thus, the area adds up to $0.75(n+x)\lceil\log(n+x)\rceil + 0.75n\lceil\log n\rceil + 0.75x\lceil\log x\rceil + 13n + 19x + 2$. Concerning the delay of the proposed architecture, the critical path is composed by: two CSAs (with a total delay of 8 units), the x MSBs generator (whose delay we approximate by half of that of a x-bit CPA), and the generator of the bit $n+x-1$ of the sum (approximated with a delay of a $n+x$-bit CPA). The resulting delay is $\lceil\log_2(n+x)\rceil + \lceil\log_2(x)\rceil + 11$. When comparing these results with those of the baseline architecture, we can conclude that the different moduli order as well as the simplification of the testing formula reduce the complexity of sign detection and increase the efficiency of the proposed scheme.

The architecture of [21] requires two adders, one modulo $2^n - 1$ and another modulo 2^{n+x}, whose areas approximately sum up to $3n\lceil\log_2 n\rceil + 1.5(n + x)\lceil\log(n + x)\rceil + 17n + 5x$; and a generator of the bit $n + x$ of a sum, whose area we approximate by half of that of a $n+x$-bit adder, corresponding to $0.75(n+x)\lceil\log_2(n+x)\rceil+2.5(n+x)$. The total area is $2.25(n+x)\lceil\log_2(n+x)\rceil+3n\lceil\log_2 n\rceil+19.5n+7.5x$. We apply the same accounting used for the previous architecture and approximate the delay by $3\lceil\log(n+x)\rceil+4.5$.

Finally, we analyse the design of the sign identifier of [24], which is restricted to the moduli set $\{2^n-1, 2^n, 2^n+1\}$. The area of the architecture, which consists of a CSA, two carry generators and some additional gates, is herein modelled as a n-bit CSA, and two half n-bit CPAs, and sum up to $1.5n\lceil\log n\rceil+12n$. Further, the delay is modelled as the delay of a CSA, a half n-bit CPA, and four levels of logic gates corresponding to $\lceil\log n\rceil+9.5$.

We can conclude that the architecture in [24] is the most efficient for the moduli set $\{2^n-1, 2^n, 2^n+1\}$. Nevertheless, the architecture herein proposed was able to extend the sign identifier to moduli sets with the form $\{2^n-1, 2^{n+x}, 2^n+1\}$ that provide greater dynamic ranges, without a significant increase in area or delay—in fact, we can see that, asymptotically, for $x=0$, the area of both architectures is dominated by $1.5n\lceil\log n\rceil$, and the delay by $\lceil\log n\rceil$. Additionally, we enhanced the architecture in [21] by combining the computation of g_2 and g_3, resulting in a decrease of delay and area.

We now analyse the RNS comparators. The baseline architecture consists of the reconstruction of the MRS digits of the two numbers that are being compared. Each reconstruction requires 3 modulo 2^n+1 CSAs, 1 modulo 2^n+1 CPA, 3 modulo 2^n-1 CSAs, and a modulo 2^n-1 CSA as previously discussed for the sign identifier. This reconstruction is followed by a magnitude comparison of the digits of the MRS. We approximate the area and delay of the digit comparators by the area and delay of a binary adder. The final formula for the area is given in Table 2. Moreover, the critical path, comprised of the path to compute d_3, and of the comparator of two most significant mixed radix digits, has the delay depicted in Table 3.

The area of the proposed architecture consists of the three channel subtractors, with an approximate total area of $1.5(n+x)\lceil\log(n+x)\rceil+7.5\lceil\log n\rceil+17.5n+5x+6$; the previously described sign identifier; and $3n+x$ gates to perform the NOR operation over all the bits of the moduli. Concerning the delay of the proposed architecture, the critical path is composed by the $n+x$-bit subtractor, with a delay of $2\left\lceil\log_2(n+x)\right\rceil+3$; and the sign identifier. The resulting area and delay of the system are found in Tables 2 and 3. We can see that there is a significant reduction in the occupied area when compared with the baseline architecture. This is a result of the proposed algorithm that first subtracts the operands before applying the MRS, instead of more traditional scheme of converting the number to the MRS and afterwards comparing them. Furthermore, the simplification of the MRS formulas enabled not only a further reduction in the area size, but also of the delay. In particular, for large values of n, and with $x=0$, one expects a speed up of up to 2.

Considering the architecture in [18], only binary addition and bitwise logic operations are required to implement (25). Since multiplying by powers of two corresponds to left shifts, only four-input $(2n+x+1)$-bit terms have to be added given the operands in (25) are properly organized, which can be implemented by two CSAs and a CPA. In terms of delay an additional $2n+x+1$-bit binary subtractor is required, to compare $F_I(A)$ and $F_I(B)$; in terms of area, four $(2n+x+1)$-bit binary CSAs and two CPAs

Table 2 Circuit area (A) for number comparison (U-G model)

Architecture	A
[18]	$4.5(2n+x+1)\lceil\log_2(2n+x+1)\rceil+3n\lceil\log_2(n)\rceil+99n+44x+42$
[4]	$1.5(4n+1)\lceil\log_2(4n+1)\rceil+6(n+x+2)\lceil\log_2(n+x+2)\rceil+4.5(n+x)\lceil\log_2(n+x)\rceil+3(n+1)\lceil\log_2(n+1)\rceil+7.5n\lceil\log_2(n)\rceil+200n+64x+41$
Proposed	$2.25(n+x)\lceil\log_2(n+x)\rceil+8.25n\lceil\log_2(n)\rceil+0.75x\lceil\log_2(x)\rceil+33.5n+25x+8$
Baseline	$1.5(n+x)\lceil\log_2(n+x)\rceil+1.5(n+1)\lceil\log_2(n+1)\rceil+15n\lceil\log_2(n)\rceil+124n+5x+29$

Table 3 Circuit delay (D) for number comparison (U-G model)

Architecture	D
[18]	$4\lceil\log_2(2n+x+1)\rceil+14$
[4]	$\lceil\log_2(4n+1)\rceil+2\lceil\log_2(n+x+2)\rceil+4\lceil\log_2(n)\rceil+22.5$
Proposed	$3\lceil\log_2(n+x)\rceil+\lceil\log_2(x)\rceil+14$
Baseline	$6\lceil\log_2 n\rceil+29$

are required to compute $F_I(A)$ and $F_I(B)$, as well as two binary subtractors to check if $a_i < b_i,\ i \in I$. Also $2n+x+1, n$, and $n+1$ input NOR gates are required for detecting equality. The total cost and delay for number comparison for [18] are presented in Tables 2 and 3.

In [4], the comparator is implemented for $x=0$, which enabled a simplification of the expression that computes α_X in (26): the floor function was omitted, and instead of computing $\frac{x_2-x_1}{2^n}$, 1 was added to the result when $x_2 > x_1$. However, for a generic value of x all the fractions need to be accurately computed. As such, a $n+1$-bit CPA was used to compute x_3-x_1, a $n+x+2$-bit CPA to compute x_1-x_2, and a $n+x+1$-bit CPA to compute x_2-x_1. The result $2^{3n-1}(x_3-x_1)+2^n(x_1-x_2)+x_2-x_1$ is then calculated, using a $4n+1$-bit CSA. In addition, a CPA is used to compute the $3n+1-x$ most significant bits of the result, corresponding to the result of the floor function. This value is reduced modulo 2^n-1 by rewriting the result as $R=\sum_{i=0}^{3} R_i 2^{in}$, and computing the sum $\left\langle \sum_{i=0}^{3} R_i \right\rangle_{2^n-1}$. Finally, if R is negative, and given that it is initially represented in two's complement, -1 is added to the reduction. The reduction is thus computed by a direct application of the formula, which is accomplished with two n-bit CSAs for $x>0$ (since $R_3=0$), or three n-bit CSAs for $x=0$, and an n-bit CPA, all modulo 2^n-1 adders. As for β_X, its value is computed using two $n+x$-bit CSAs when $x>0$, or a single $n+x$-bit CSA when $x=0$, and a $n+x$-bit CPA.

The area required by the architecture in [4] is even larger than the one proposed in [18], as it can be observed in Table 2. This is mostly due to the fixed-point approach required to compute the value of the floor function in (26). The complexity of the circuit also leads to a larger delay.

According to the U-G model, the architecture in [18] is the most efficient from the related state of the art. It presents a higher delay than the proposed architecture, and the difference in delays gets shorter as x gets closer to n. The proposed architecture is also supported on smaller circuits. In fact, it is observed in Table 2 that the proposed architectures require the smallest area if compared with the related state of the art. Considering the area–time (AT) product (AT is the common designation of the product of the area and the delay of the circuit) as a figure of efficiency, and $x \in \{0, \frac{n}{2}, n\}$, $n \in \{4, 8, 16, 32\}$, the proposed architecture improves the AT parameter by up to 83.87 % when compared with the baseline, by up to 74.36 % if compared to [18], and by up to 90.38 % if compared to [4].

7.2 Experimental Evaluation

The proposed and state-of-the-art circuits were described in synthesizable VHDL and their functionality was thoroughly tested[1]. A well-known library of arithmetic units [32], also described in synthesizable VHDL, was used. This library contains a structural specification of components, namely optimized prefix adders, which were employed to describe and implement the comparators. Experimental results were obtained both for ASICs using a TSMC standard cell library tailored for the 65 nm CMOS LOGIC General Purpose Plus technology and FPGAs on a Xilinx Virtex5 XC5VLX220 FF1760 -2. Tables describing these results in detail can be found in Online Resource 1.

We start by implementing the baseline and the proposed sign identifier, and those of [21,24]. However, it is not possible to perform a thorough comparison, due to the fact that [24] can only be used when $x = 0$, and that those papers do not have the same coverage of the proposed architecture. Therefore, we decided to limit the implementation results to the ASIC technology, and for $x = 0$. The values obtained for performance, presented in Online Resource 1, are as predicted using the U-G model. First, the proposed architecture outperforms the baseline architecture, showing a relative improvement of the AT of up to 93.5 %. Furthermore, the implementation of [21] consistently performs worse than the proposed architecture, if the AT metric is considered. Moreover, the proposed design shows a similar performance to that of [24], with a maximum increase in the AT of about 30 % for $n = 8$, and an improvement of 20 % for $n = 32$. Thus, the proposed architecture presents the best features to underpin RNS comparison, since it supports the moduli set $\{2^n - 1, 2^{n+x}, 2^n + 1\}$, and has a performance comparable to that of [24].

Regarding the experimental results obtained for the implementation of the RNS comparators on ASIC and FPGA technologies, which are presented in Online Resource 1, as expected from the analysis based on the U-G model, the delay for the baseline architecture is almost constant for a constant value of n, and varying values of x, and is at most two times larger than the proposed architecture. The area of the baseline architecture is larger than the area of the proposed architecture for all considered values of n and x. Also, similar delays are obtained with the proposed architecture and the architecture in [18] for number comparison. However, with the proposed architecture there is a significant reduction in the circuit area in comparison with the one in [18]. If compared to the architecture in [4], the proposed implementation is up to 83 and

[1] The HDL specification of the proposed architecture will be made publicly available at http://sips.inesc-id.pt/las/prototypes/RNScomparators/.

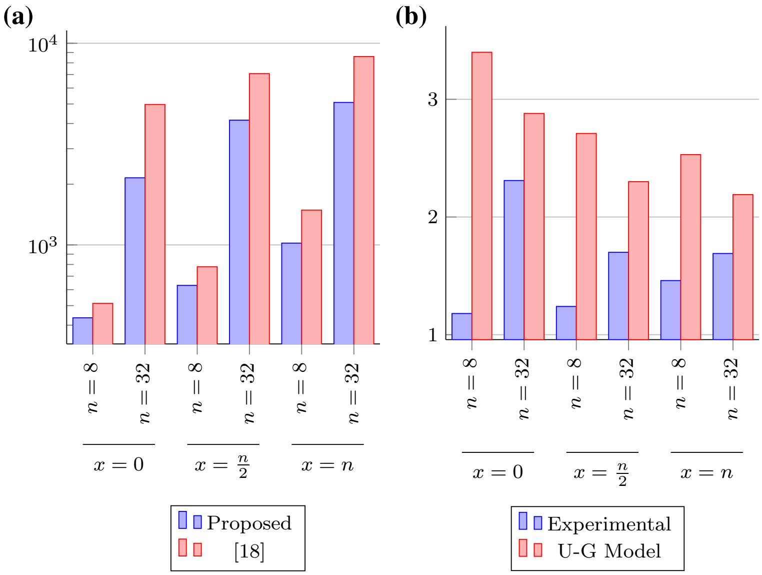


Fig. 4 Results obtained for the ASIC Area × Time product (AT) for the proposed and [18] circuits (AT corresponds to the product of the area and the delay of the circuits). **a** Experimental results [$\mu m^2 ns$]. **b** Experimental and theoretical results for [18], normalized with the proposed architecture

77 % more efficient when considering the AT for the ASIC and FPGA technologies, respectively.

Graphs in Figs. 4 and 5 present experimental results of the area–time (AT) product efficiency metric when the proposed architecture and the one in [18] were implemented in ASIC and FPGA, respectively. Values of $x \in \{0, \frac{n}{2}, n\}$ are considered and in Figs. 4a and 5a values in the y axes are represented in a logarithmic scale. It is experimentally observed that the efficiency of the proposed architecture for number comparison is always significantly higher in comparison with the one in [18], achieving gains of up to 57 % on the ASIC and up to 38 % on the FPGA.

In order to contrast the experimental values obtained for the efficiency with the values predicted with the U-G model, we have plotted in the graphs of Fig. 4b) and 5b) both the experimental and theoretical results obtained for [18], normalized by taking as reference the proposed architecture. We can see that the U-G model approximation is more accurate when x takes the value n. The discrepancies are due to approximation errors of the U-G model, such as the modelling of the block to generate the bit n of a sum as a block with half the area and delay of an n bit adder. Furthermore, the U-G model does not take into account the area occupied by interconnections. Although it predicts a relative improvement of performance of [18] as the circuit gets larger, the ASIC experimental results show that the area occupied by the interconnections increases in such a way that the relative efficiency decreases when x and n increase. For the FPGA technology, this effect is more pronounced when $x = 0$ and n increases.

Finally, experimental results were obtained for power consumption in ASIC, from the placed-and-routed circuit specifications for 20 % of switching activity. The

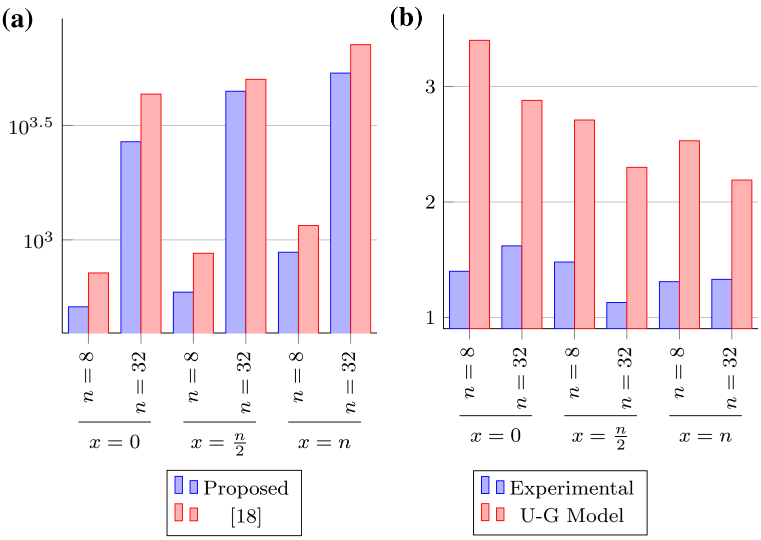


Fig. 5 Results obtained for the FPGA Area × Time product (AT) for the proposed and [18] circuits (AT corresponds to the product of the area and the delay of the circuits). **a** Experimental results [(slices ×)ns]. **b** Experimental and theoretical results for [18], normalized with the proposed architecture

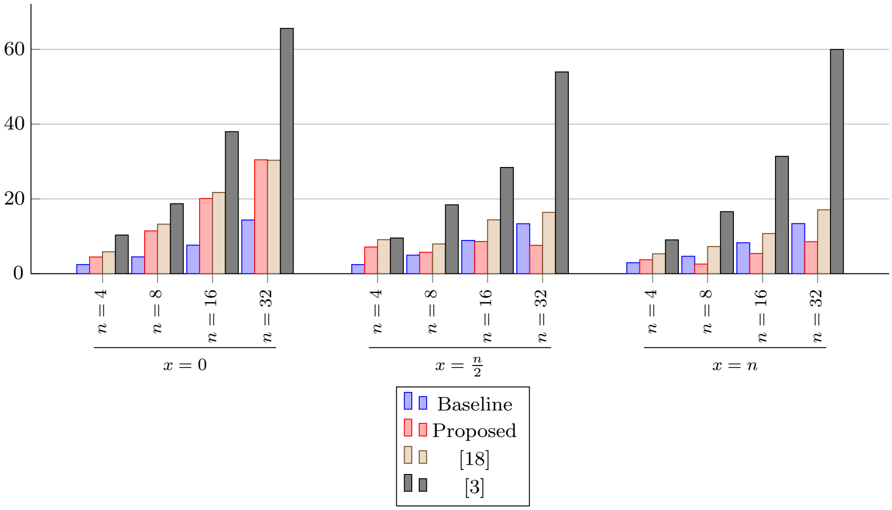


Fig. 6 Power consumption: experimental results [mW] for the proposed and related art circuits

Cadence Encounter built-in power reporting tool was employed, and the total power was measured, including the dynamic and leakage power. Results are presented in Fig. 6. The proposed architecture consumes more power than the baseline architecture for small values of x, and $n = \frac{n}{2}$, but this behaviour is inverted as x gets larger. Apart from the baseline architecture for small values of x, the proposed architecture offers the lowest power consumption. When $x = \frac{n}{2}$, the proposed architecture is 21 % times more efficient for $n = 4$, and 54 % for $n = 32$, if compared with the implementation of [18]. Furthermore, for the same values of x, it consumes 5 % times less power when $n = 4$, and 70 % when $n = 32$, in comparison with the architecture of [4].

8 Conclusions

In this paper, we have shown how to design efficient RNS comparators for large dynamic ranges, by exploiting the $\{2^n + 1, 2^n - 1, 2^{n+x}\}$ moduli set. Such design technique followed a two-step approach. First, the MRS was optimized for sign identification, by focusing on the extraction of the sign bit and on the parallelization of the hardware structures. It was shown that the sign identification module was an optimized extension to the most apt sign identifier for the $\{2^n + 1, 2^n - 1, 2^n\}$ moduli set in terms of efficiency, and further that it improved upon a previous proposal for the $\{2^n + 1, 2^n - 1, 2^{n+x}\}$ set. Then, we used the sign identification module to implement the number comparison operation. In terms of efficiency, the considered ASIC and FPGA proof of concept implementations exhibit relative improvements of up to 57 and 38 % regarding the related state of the art, respectively. Also, their power consumption is reduced by up to 50 %. These features make the proposed comparator an attractive hardware architecture for signal processing applications, such as nonlinear filtering, thus paving the way for more powerful solutions completely based on RNS.

References

1. S. Antão, L. Sousa, The CRNS framework and its application to programmable and reconfigurable cryptography. ACM Trans. Archit. Code Optim. **9**(4), 1–25 (2013)
2. K. Barner, G. Arce (eds.), *Nonlinear Signal and Image Processing: Theory, Methods, and Applications. Electrical Engineering & Applied Signal Processing* (CRC Press, Boca Raton, 2003)
3. G. Bernocchi, G. Cardarili, A. Nannarelli, M. Re, Low power adaptive filter based on RNS components. in *12th IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 3211–3214 (2007)
4. B.S. Bi, W. Gross, The mixed-radix chinese remainder theorem and its applications to residue comparison. IEEE Trans. Comput. **57**(12), 1624–1632 (2008)
5. R. Chaves, L. Sousa, Improving residue number system multiplication with more balanced moduli sets and enhanced modular arithmetic structures. IET Comput. Digital Techn. **1**(5), 472–480 (2007)
6. R. Conway, J. Nelson, Fast converter for 3 moduli RNS using new property of CRT. IEEE Trans. Comput. **48**(8), 852–860 (1999)
7. G. Dimauro, S. Impedovo, G. Pirlo, A new technique for fast number comparison in the residue number system. IEEE Trans. Comput. **42**(5), 608–612 (1993)
8. G. Dimitrakopoulos, D.G. Nikolos, H. Vergos, D. Nikolos, C. Efstathiou, New architectures for modulo $2^n - 1$ adders. in *12th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 1–4 (2005)
9. S.T. Eivazi, M. Hosseinzadeh, O. Mirmotahari, Fully parallel comparator for the moduli set $\{2^n, 2^n - 1, 2^n + 1\}$. IEICE Electron. Exp. **8**(12), 897–901 (2011)
10. K. Ibrahim, S. Saloum, An efficient residue to binary converter design. IEEE Trans. Circuits Syst. **35**(9), 1156–1158 (1988)
11. I. Kouretas, V. Paliouras, A low-complexity high-radix RNS multiplier. IEEE Trans. Circuits Syst. I Regul. Pap. **56**(11), 2449–2462 (2009)
12. D. Miller, R. Altschul, J. King, J. Polky, Analysis of the residue class core function of akushskii, burcev, and pak, in *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, ed. by S. Soderstrand, W. Jenkins, G. Jullien, F. Taylor (IEEE Press, Piscataway, NJ, 1986)

13. P.V. Mohan, Evaluation of fast conversion techniques for binary-residue number systems. IEEE Trans. Circuits Syst. I: Fundam. Theory Appl. **45**(10), 1107–1109 (1998)
14. P.V. Mohan, RNS to binary conversion using diagonal function and Pirlo and impedovo monotonic function. Circuits Syst. Signal Process. **35**(3), 1063–1076 (2015)
15. P.V. Mohan, A.B. Premkumar, RNS-to-binary converters for two four-moduli sets $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ and $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$. IEEE Trans. Circuits Syst. I Regul. Pap. **54**(6), 1245–1254 (2007)
16. B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, 2nd edn. (Oxford University Press, New York, 2010)
17. H. Pettenghi, R. Chaves, L. Sousa, RNS reverse converters for moduli sets with dynamic ranges up to (8n+1)-bit. IEEE Trans. Circuits Syst. I Regul. Pap. **60**(6), 1487–1500 (2013)
18. G. Pirlo, S. Impedovo, A new class of monotone functions of the residue number system. Int. J. Math. Models Methods Appl. Sci. **7**(9), 803–809 (2013)
19. M. Soderstrand, W. Jenkins, G. Jullien, F. Taylor (eds.), *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing* (IEEE Press, Piscataway, NJ, 1986)
20. L. Sousa, Efficient method for magnitude comparison in RNS based on two pairs of conjugate moduli. in *IEEE Symposium on Computer Arithmetic*, pp. 240–250 (2007)
21. L. Sousa, P. Martins, Efficient sign identification engines for integers represented in rns extended 3-moduli set $\{2^n-1, 2^{n+k}, 2^n+1\}$. Electron. Lett. **50**(16), 1138–1139 (2014)
22. N. Szabo, R. Tanaka (eds.), *Residue arithmetic and its application to computer technology* (McGraw-Hill, New York, 1967)
23. T. Tay, C. Chip-Hong, J. Low, Efficient VLSI implementation of 2^n scaling of signed integer in RNS $\{2^n-1, 2^n, 2^n+1\}$. IEEE Trans. VLSI Syst. **21**(10), 1936–1940 (2013)
24. T. Tomczak, Fast sign detection for RNS $\{2^n-1, 2^n, 2^n+1\}$. IEEE Trans. Circuits Syst. I Regul. Pap. **55**(6), 1502–1511 (2008)
25. A. Tyagi, A reduced-area scheme for carry-select adders. IEEE Trans. Comput. **42**(10), 1163–1170 (1993)
26. H. Vergos, C. Efstathiou, D. Nikolos, Diminished-one modulo 2^n+1 adder design. IEEE Trans. Comput. **51**(12), 1389–1399 (2002)
27. B. Vinnakota, V.V.B. Rao, Fast conversion techniques for binary-residue number systems. IEEE Trans. Circuits Systems I Fundam. Theory Appl. **41**(12), 927–929 (1994)
28. Y. Wang, New chinese remainder theorems. in *Signals, Systems and Computers, 1998. Conference Record of the Thirty-Second Asilomar Conference on, vol. 1*, vol. 1, pp. 165–171 (1998). doi:10.1109/ACSSC.1998.750847
29. Y. Wang, S. Xiaoyu, M. Aboulhamid, A new algorithm for RNS magnitude comparison based on New Chinese Remainder Theorem II. in *Ninth Great Lakes Symposium on VLSI*, pp. 362–365 (1999)
30. M. Xu, Z. Bian, R. Yao, Fast sign detection algorithm for the RNS moduli set $\{2^{n+1}-1, 2^n-1, 2^n\}$. IEEE Trans. VLSI Syst. **23**(2), 379–383 (2015)
31. H.M. Yassine, W.R. Moore, Improved mixed-radix conversion for residue number system architectures. IEE Proc. G Circuits Devices Syst. **138**(1), 120–124 (1991)
32. R. Zimmermann, Efficient VLSI implementation of modulo $2^n \pm 1$ addition and multiplication. in *14th IEEE Symposium on Computer Arithmetic (ARITH)*, pp. 158–167 (1999)