Patronik and Piestrak
EURASIP Journal on Advances in Signal Processing (2023) 2023:92
https://doi.org/10.1186/s13634-023-01037-8

EURASIP Journal on Advances in Signal Processing

RESEARCH Open Access

Design of reverse converters for the general RNS 3-moduli set $\{2^k, 2^n - 1, 2^n + 1\}$

Piotr Patronik[1*] and Stanisław J. Piestrak[2]

*Correspondence:
piotr.patronik@pwr.wroc.pl

[1] Department of Computer Engineering, Faculty of Electronics (W-4/K-9), Wrocław University of Technology, 50–370 Wrocław, Poland
[2] Institut Jean Lamour (UMR 7198 CNRS), Université de Lorraine, Campus ARTEM, 54011 Nancy, France

Abstract

This paper presents a new design method of the reverse (residue-to-binary) converter for the flexible 3-moduli residue number system (RNS) set $\{2^k, 2^n - 1, 2^n + 1\}$, where k and n are a pair of arbitrary integers ≥ 2. The basic equation of the reverse converter is formulated in two alternative forms, each of which consists of two separate parts: one depending on input variables of the converter, and the other being a single constant. The constant can be either added inside the reverse converter or shifted out to the residue datapath channels, in most cases at no hardware cost or extra delay. From the set of basic functions, which are essentially different than those of the only two known general converters proposed for this moduli set, four versions of a converter can be designed for any pair of k and n. Experimental results obtained using the commercial 65-nm low-power design kit and industrial synthesis tools for all dynamic ranges from 8 to 40 bits suggest that, compared to the state-of-the-art designs, at least one version of the newly proposed converters is superior w.r.t. delay, power consumption, and area, for all dynamic ranges considered. The savings for the best versions (these with constants moved to the datapath channels) are up to 12.7% for the area and from 2.5% to 14% (5.8 % on average) for the delay, while the power consumption is reduced up to 23.2% (5.6% on average).

Keywords: Application-specific integrated circuit (ASIC), Residue arithmetic, Residue number system (RNS), Residue-to-binary converter, Reverse converter

1 Introduction

Several computational problems require high-throughput computations that aggregate multiplications and additions in long uninterrupted series. These include various digital signal processing (DSP) algorithms that rely on sum-of-products computations, such as: digital convolution [15], filtering [31], image processing [34], discrete Fourier transform [33], number theoretic transforms [11], and discrete wavelet transform [1, 16, 30]. One of the most important approaches to throughput improvement relies on selecting data representation which is the most appropriate for a given computational problem. This is because it could allow not only for reduction of the number of executed operations, the length of operands, the activity of data, and the number and/or the length of the global connections, but also area, delay, latency, and power dissipation. A valuable alternative to the commonly used 2's complement integer arithmetic is the non-positional

residue number system (RNS) defined by a set of pairwise prime positive integers, called *moduli*, whose product determines its *dynamic range*. Its potential efficiency advantages over the 2's complement representation stem from the decomposition of operands of arithmetic operations into small and independent *residues*, which are processed independently in relatively narrow, parallel, and modular *residue datapath channels*. In comparison with the positional *2's complement* integer arithmetic, RNS computations enjoy the reduced carry propagation within each of independent residue datapaths, which also handle smaller partial product matrices. RNS adders, multipliers, and multiplier–accumulators (MACs) can simultaneously operate faster, utilize less area, and dissipate less power than their 2's complement integer counterparts. However, to communicate with predominantly used 2's complement hardware, RNS datapaths must be accompanied by the binary-to-residue (*forward*) and the residue-to-binary (*reverse*) converters, of which the latter is significantly more complex to design. Unfortunately, these converters are pure overhead, which partially consumes potential benefits of RNS. Therefore, it is crucial to minimize their adverse impact on the overall efficiency of an RNS-based computing unit. As for performance of the RNS datapath, it depends heavily on the choice of the moduli set defining the RNS. Particularly efficient implementations of all arithmetic circuitry enjoy three classes of moduli (thus called *low cost*): even moduli of the form $m = 2^n$ (of which only one can be used) and odd moduli of the form $m = 2^n \pm 1$. To take the maximal advantage of RNS, including performance of the RNS datapath as a whole, it is also desirable that residue datapath channels are *balanced*, i.e., such that the area and the delay of all residue channels are as close as possible.

There have been proposed many RNS moduli *special sets*, composed only of low-cost moduli, accompanied by specifically developed design methods of reverse converters. The most studied has been the classic 3-moduli set $2^n, 2^n - 1, 2^n + 1$ [8, 9, 24, 26, 32, 36, 37], offering the $(3n - 1)$-bit dynamic range with 3-bit resolution as all three moduli depend solely on n. (*Note:* here we assume that the dynamic range of a bits guarantees that all 2^a integers can be represented in this RNS.) However, such a moduli set with similar channel widths led to the *imbalance problem*, where for a given n, the channel mod 2^n is significantly less complex and faster than remaining two odd channels mod $2^n \pm 1$. That in turn means that the width of the even channel can be larger than the widths of the odd channels without any negative impact on the delay of the complete RNS datapath, and with relatively little impact on its area and power.

The channel imbalance problem has been addressed in [4, 6, 12], where the authors considered a new generalized flexible 3-moduli set $\{2^k, 2^n - 1, 2^n + 1\}$, in which the size k of the even modulus 2^k is independent on the size n of two odd moduli $2^n \pm 1$. For a given dynamic range, the latter moduli set enjoys two main advantages: (i) by properly selecting $k > n$, the performance of the even channel mod 2^k could be made similar to that of the odd channels (which, for a given dynamic range, become less complex and faster than for the classic 3-moduli set), and (ii) the exact 1-bit dynamic range resolution is ensured. The first known partially general design method of the reverse converter for the 3-moduli set $\{2^k, 2^n - 1, 2^n + 1\}$ with k limited to $n < k \leq 2n$ was proposed in [4]. The design from [4] followed the approach from the previous converters for the non-flexible moduli

set, specifically [26] with two CSA levels adding the bit-level manipulations needed for the extension of the even modulus. Then, the bit-level manipulations were optimized in [12], where the two CSA stages have been reduced to one with bit-level manipulations partially incorporated into the remaining CSA structure.

Some other special RNSs defined using low-cost and balanced moduli sets with flexible even modulus 2^k, accompanied by the design methods of reverse converters were also proposed:

(i) for the 4-moduli sets:

- in [2], for $\{2^k, 2^n - 1, 2^n + 1, 2^{n+1} - 1\}$ (n even and arbitrary k such that $n \leq k \leq 2n$);
- in [38], for $\{2^{n-1} - 1, 2^{n+1} - 1, 2^k, 2^n - 1\}$ (n even and $k > 2$)
- in [20], for two classes of 4-moduli sets $\{2^n - 1, 2^k, 2^n + 1, 2^{n+1} - 1\}$ and $\{2^n - 1, 2^k, 2^n + 1, 2^{n-1} - 1\}$ (n even and arbitrary k),

(ii) in [22], for the 5-moduli set $\{2^k, 2^n - 1, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$ (n even and arbitrary k).

To note that the 3-moduli converter for the 3-moduli set $\{2^k, 2^n - 1, 2^n + 1\}$ is an integral part of the converters from [20] and [22].

1.1 Contribution of this paper

This article proposes a novel general design method of a reverse converter for the flexible 3-moduli set $\{2^k, 2^n - 1, 2^n + 1\}$ with unrestricted k, which leads to circuits enjoying better performance than the existing counterparts. The converter equations will be derived using the New Chinese remainder theorem I, which allowed us to formulate converter equation in a form consisting of two separate parts: one depending on input variables of the converter and the other being a single constant. Then, the constant can be either added inside the reverse converter or shifted out to the residue datapath channels. Experimental results suggest that, compared to the state-of-the-art designs, at least one of four proposed versions of the newly proposed converters is superior w.r.t. delay, power consumption, and area, for all dynamic ranges considered.

The remainder of this paper is organized as follows. In Sect. 2, the basic properties of the RNS and the necessary mathematical background are revisited as well as various application and hardware implementation aspects for the RNS moduli set considered here are discussed. In Sect. 3, the converter's equations with a formal proof of the correctness of the design are given. They are accompanied by the exploration of the various possibilities leading to obtaining the most efficient hardware implementation. In Sect. 4, first, the detailed evaluation of the delay and area performance as well as the power efficiency is presented. Then, the synthesized versions of our converters are compared against the existing counterparts as well as against the best-known converters for the classic $\{2^n - 1, 2^n, 2^n + 1\}$ moduli set. Finally, in Sect. 5, some conclusions are drawn and some directions for future research are proposed.

2 Preliminaries

2.1 Basic properties of RNS

The RNS is defined by the *moduli set* $\{m_1, \ldots, m_l\}$ of l positive pairwise prime integers. According to the CRT, its *dynamic range*, i.e., the number of integers X that can be uniquely represented, is $M = \prod_{i=1}^{l} m_i$; thus, for instance, any nonnegative integer $0 \le X \le M-1$ has a unique representation. In particular, all 2^a integers can be represented in this RNS, where $a = \lfloor \log_2 M \rfloor$. For the 3-moduli set $\{2^k, 2^n-1, 2^n+1\}$ considered here, $M = 2^k(2^{2n}-1)$, which suffices to represent all $(k+2n-1)$-bit wide binary integers. Any integer $0 \le X \le M-1$ is represented in RNS as an ordered l-tuple $\{x_1, \ldots, x_l\}$, where the *residue* mod m_i $x_i = |X|_{m_i}$ (also denoted $X = \bmod m_i$) is the remainder of the integer division of X by m_i, $1 \le i \le l$. If two integers X and Y are, respectively, represented by the sets of residues $\{x_1, \ldots, x_l\}$ and $\{y_1, \ldots, y_l\}$, then the set of residues $\{z_1, \ldots, z_l\}$ computed as $z_i = \left|x_i \circ y_i\right|_{m_i}$, $1 \le i \le l$, uniquely represents $Z = |X \circ Y|_M$ for any operation $\circ \in \{+, -, \times\}$. If $|X \cdot Y|_m = 1$, then X is a *multiplicative inverse* of $Y \,(\bmod m)$ and Y is a multiplicative inverse of $X \,(\bmod m)$.

2.2 Properties of arithmetic mod $2^n - 1$

Arithmetic operations mod $2^n - 1$ have the following properties which will be used widely later. Let Z denote an integer mod $2^n - 1$ and $(z_{n-1} \ldots z_0)$ its n-bit representation.

- $-Z$, the negative value of Z, can be calculated as the 1's complement (bitwise complement) of the binary representation of Z, i.e., $\bar{Z} = (\bar{z}_{n-1} \ldots \bar{z}_0)$ represents $-Z$. Recall, however, that computing $-Z$ as 1's complement results in a double representation of 0, which can appear as both $(0 \ldots 0)_b$ and $(1 \ldots 1)_b$. (In all numerical examples, the parentheses and the index b will be used to differentiate the binary representation from its integer (decimal) value.)
- If d is a positive integer, then the result of the multiplication $\left|2^d Z\right|_{2^n-1}$ can be obtained by the left cyclic shift of Z by d positions.
- If d is a positive integer, then 2^d and $2^n - 1$ are obviously relatively prime. Hence, because $|2^{-d}|$ and $|2^d|$ are the multiplicative inverses of each other mod $2^n - 1$, the result of the multiplication $\left|2^{-d} Z\right|_{2^n-1}$ can be obtained by the right cyclic shift of Z by d positions.

2.3 The new CRT

Given a moduli set $\{m_1, \ldots, m_l\}$ and the set of residues $\{x_1, \ldots, x_l\}$, the value of X can be calculated using two classic methods like the CRT and the mixed-radix conversion (MRC) algorithms [1]. However, here we will use the New CRT-I [35], according to which X is calculated using the additional variable V leading to a simpler equation

$$X = x_1 + m_1 V, \tag{1}$$

where

$$V = \left| q_1(x_2 - x_1) + \sum_{i=2}^{l-1} \left[q_i \Big(\prod_{j=2}^{i} m_j \Big) (x_{i+1} - x_i) \right] \right|_{\prod_{j=2}^{l} m_j} \tag{2}$$

and the set of multiplicative inverses $\{q_1, \ldots, q_{l-1}\}$ is computed according to

$$q_i = \left| \frac{1}{\prod_{j=1}^{i} m_j} \right|_{\prod_{u=i+1}^{l} m_u}, \quad 1 \le i \le l-1. \tag{3}$$

Because here $l = 3$, Eqn (2) reduces to

$$X = x_1 + m_1 \cdot \underbrace{|q_1(x_2 - x_1) + q_2 m_2 (x_3 - x_2)|_{m_2 m_3}}_{V} \tag{4}$$

with

$$q_1 = \left| \frac{1}{m_1} \right|_{m_2 m_3}, \tag{5}$$

$$q_2 = \left| \frac{1}{m_1 m_2} \right|_{m_3}. \tag{6}$$

It has been observed in [35] that for the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, the complexity of a reverse converter varies with different assignments of moduli to positions m_1, m_2, and m_3 in (4). In particular, if $m_1 = 2^n$, then (4) simplifies to $X = x_1 + 2^n V$, i.e., it can be computed as a simple concatenation of the binary representations of V and x_1, denoted $(V||x_1)$, at no hardware cost. Moreover, since $(2^n - 1)(2^n + 1) = 2^{2n} - 1$, the value of V can be conveniently computed modulo $2^{2n} - 1$ [25]. Finally, if $m_2 = 2^n + 1$ and $m_3 = 2^n - 1$, the coefficients q_1 and q_2 are the multiplicative inverses of $2^n \pmod{2^{2n} - 1}$ and $2^n(2^n + 1) \pmod{2^n - 1}$, respectively. Because the same advantages can also be observed for the flexible moduli set $\{2^k, 2^n - 1, 2^n + 1\}$, we have chosen similar moduli assignment in our design.

2.4 Advantages and applications of the 3-moduli set $\{2^k, 2^n - 1, 2^n + 1\}$

The reverse converters considered here may have twofold applications. Basically, they can be used as an integral part of any RNS-based processor using the 3-moduli set $\{2^k, 2^n - 1, 2^n + 1\}$. Nevertheless, not less important is their use as the main building block in various reverse converters for larger multi-moduli RNSs formed by extending the basic 3-moduli set with one or two extra moduli. Sample architectures of the latter, constructed on the premises of the MRC algorithm are those proposed for larger flexible moduli sets: the 4-moduli sets $\{2^k, 2^n - 1, 2^n + 1, 2^{n+1} - 1\}$ (n even and arbitrary k such that $n \le k \le 2n$) [2] and $\{2^k, 2^n - 1, 2^n + 1, 2^{n+1} - 1, 2^{n\pm1} - 1\}$ (n even and $k > 2$) [20] as well as for the 5-moduli set $\{2^k, 2^n - 1, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$ (n even) [22].

Unquestionable advantages of the flexible moduli set provide the following argument. Recall that MACs are the basic building blocks which dominate hardware usage of RNS-based processors for numerous applications like those mentioned for DSP. Therefore, their complexity can serve as a reference for comparison of efficiency of various RNS moduli sets providing a given dynamic range (see, e.g., the parameters of residue MACs and RNS datapaths reported in [7, 17, 28, 29]). For instance, the dynamic range of $3n-1$ bits can be ensured by two sets of moduli: the classic non-flexible set $\{2^n, 2^n-1, 2^n+1\}$ and its flexible counterpart $\{2^k, 2^p-1, 2^p+1$ with $k=n+2$ and $p=n-1$ considered here. Now, let us compare the complexity of two respective sets of three residue MACs, which form one stage of the residue datapath in each of these two cases. It can be easily found that the overall complexity of the set of three MACs mod 2^n, 2^n-1, and 2^n-1 is larger than for the set of three MACs mod 2^{n+2}, $2^{n-1}+1$, and $2^{n-1}+1$. (As for the forward converters, the pair of residue generators mod $2^{n-1}-1$ and $2^{n-1}+1$ actually requires a few full-adders more than their counterparts.) Clearly, for a given dynamic range of $3n-1$ bits and any $n \geq 3$, the estimated overall complexity of three MACs, respectively, mod 2^n and $2^n \pm 1$ forming one stage of the residue datapath is larger than, e.g., of three MACs mod 2^{n+3} and $2^{n-1} \pm 1$. Should performance of reverse converters for the flexible 3-moduli set be at least comparable to that of its non-flexible counterpart $\{2^n, 2^n-1, 2^n+1\}$, the former would supersede the latter and can be considered the best choice among all known 3-moduli sets.

Hardware implementations of all basic modular arithmetic circuits for odd low-cost moduli of the form $m=2^n \pm 1$ have been proposed: residue 2-operand adders [5, 13, 14, 19, 39], residue multipliers [10, 18, 39], MACs [7, 17], residue generators (used to build forward converters), and multi-operand adders [21, 25, 27]. Recall, however, that all arithmetic circuits mod 2^n are significantly faster and less hardware-consuming than for any comparable odd moduli (i.e., using the same number of bits n to represent all valid residue values), because any carry-out signal generated of the most significant bit (MSB) of weight 2^{n-1} is simply ignored. Unfortunately, only one even modulus can be used. Clearly, should the same bit width be used for all three datapath channels mod 2^n and $2^n \pm 1$, it would result in underutilization of the even channel mod 2^n. Therefore, to maximize its performance advantage, the authors of [4] suggested that the even channel should be at least a few bits wider than odd channels. The advantages of the moduli sets using one even modulus 2^k larger by a few bits than all odd moduli have been observed for the applications reported, e.g., in [3, 4, 23]. In [3, 4], the 3-moduli set $\{2^{2n}, 2^n-1, 2^n+1\}$ was considered and the RISC DSP based on the 3-moduli RNS $\{2^{16}, 2^8-1, 2^8+1\}$ was implemented. Recently, an RNS-based coprocessor using one larger even modulus 2^k and at least two pairwise prime moduli of the type $2^{n_i}-1$ was proposed in [23].

3 Reverse converter design

Let $m_1=2^k$, $m_2=2^n+1$, and $m_3=2^n-1$, and x_1, x_2, and x_3 be their respective residues. For this moduli set, we will try to find the function of the reverse converter which can be expressed as

$$X = x_1 + 2^k V, \tag{7}$$

so that it can be implemented as a simple concatenation $X = (V||x_1)$. Hence, the design problem reduces to finding such an expression for V, which would lead to its most efficient hardware implementation. Additionally, following our encouraging results from [24], obtained for the simple 3-moduli set $\{2^n, 2^n - 1, 2^n + 1\}$, we will attempt to obtain an expression for V in which the variable and the constant terms are separated. The design procedure which will be presented below can be summarized as follows.

(1) Derive an expression for V which involves three terms v_i, each of which depends only on its respective residue $x_i, 1 \le i \le 3$.
(2) For each of three terms v_i, determine the variable part containing only expressions which are the functions of the respective residues $x_i, 1 \le i \le 3$, each accompanied by a constant.
(3) Find the basic equation for V with a single separated constant and three variable terms $v_i, 1 \le i \le 3$.
(4) Execute bit-level optimization of V for further hardware and delay reduction.
(5) Implement the function of the reverse converter (7) using optimized V in hardware. □

3.1 Expression for *V* with separated residues

From (5) we have $q_1 = |1/2^k|_{2^{2n}-1} = |2^{-k}|_{2^{2n}-1}$. Now, let t be a nonnegative integer such that $t = |k|_n$, i.e., $0 \le t \le n-1$, which implies that for any positive k we have $\left|2^k\right|_{2^n-1} = 2^t$. Then, from (6) we have $q_2 = |1/[2^k(2^n+1)]|_{2^n-1} = 2^{n-1-t}$. We will consider two cases: (i) $|k|_{2n} = t$ and (ii) $|k|_{2n} = n + t$. In the first case, we rewrite the term V from (4) as

$$\begin{aligned} V &= \left|2^{-t}(x_2 - x_1) + 2^{n-1-t} \cdot (2^n+1)(x_3 - x_2)\right|_{2^{2n}-1} \\ &= \left|\frac{x_2 - x_1 + 2^{n-1} \cdot (2^n+1)(x_3 - x_2)}{2^t}\right|_{2^{2n}-1} \\ &= \left|\frac{(1 - 2^{2n-1} - 2^{n-1})x_2 + 2^{n-1}(2^n+1)x_3 - x_1}{2^t}\right|_{2^{2n}-1}. \end{aligned}$$

Since $\left|1 - 2^{2n-1}\right|_{2^{2n}-1} = 2^{2n-1}$, we have

$$V = \left|\frac{2^{n-1}(2^n-1)x_2 + 2^{n-1}(2^n+1)x_3 - x_1}{2^t}\right|_{2^{2n}-1}. \tag{8}$$

In the second case, we rewrite the term V from (4) as

$$
\begin{aligned}
V &= \left|2^{-n-t}(x_2-x_1)+2^{n-1-t}\cdot(2^n+1)(x_3-x_2)\right|_{2^{2n}-1}\\
&= \left|\frac{2^{-n}(x_2-x_1)+2^{n-1}\cdot(2^n+1)(x_3-x_2)}{2^t}\right|_{2^{2n}-1}\\
&= \left|\frac{\begin{pmatrix}(2^{-n}-2^{2n-1}-2^{n-1})x_2\\ +2^{n-1}(2^n+1)x_3-2^{-n}x_1\end{pmatrix}}{2^t}\right|_{2^{2n}-1}\\
&= \left|\frac{\begin{pmatrix}2^{-n}(1-2^{2n-1}-2^{n-1})x_2\\ +2^{-n}\cdot 2^{2n-1}(2^n+1)x_3-2^{-n}x_1\end{pmatrix}}{2^t}\right|_{2^{2n}-1}.
\end{aligned}
$$

From $\left|2^{2n-1}(2^n+1)\right|_{2^{2n}-1}=\left|2^{n-1}(2^n+1)\right|_{2^{2n}-1}$, we have

$$
V=\left|\frac{2^{n-1}(2^n-1)x_2+2^{n-1}(2^n+1)x_3-x_1}{2^{n+t}}\right|_{2^{2n}-1}. \tag{9}
$$

As $k=2n\lfloor k/2n\rfloor+n+t$ or $k=2n\lfloor k/2n\rfloor+t$ and $\left|\lfloor k/2n\rfloor\right|_{2^{2n}-1}=1$, we can merge both cases from Eqns (8) and (9) and rewrite V using three separate variables v_1, v_2, and v_3 as

$$
\begin{aligned}
V &= \left|2^{-k}\left(2^{n-1}(2^n+1)x_3+2^{n-1}(2^n-1)x_2-x_1\right)\right|_{2^{2n}-1}\\
&= \Bigg|\underbrace{\left|2^{-k}\cdot 2^{n-1}(2^n+1)x_3\right|_{2^{2n}-1}}_{v_3}\\
&\quad+\underbrace{\left|2^{-k}\cdot 2^{n-1}(2^n-1)x_2)\right|_{2^{2n}-1}}_{v_2}+\underbrace{\left|-2^{-k}x_1\right|_{2^{2n}-1}}_{v_1}\Bigg|_{2^{2n}-1}.
\end{aligned} \tag{10}
$$

The terms v_1, v_2, and v_3 can be determined separately and then added mod $2^{2n}-1$ to obtain the value of V.

3.2 Separation of variable and constant parts

Further simplification of Eqn (10) relies on determining in each of three terms v_1, v_2, and v_3: (a) the variable part containing only expressions which are the functions of three residues x_1, x_2, and x_3; and (b) the remaining part composed only of constants. The goal of finding such a representation of Eqn (10) is twofold:

(i) should a single global constant be determined, it would allow to reduce the number of operands involving constants to be added to only one; and

(ii) as suggested in [24], the addition of the global constant can be shifted to the datapath channels, thus further reducing the total number of operands to be added by the carry–save adder (CSA) tree of the converter (moreover that it can be done at no cost).

Here, we will show how to extract all constants from three terms v_1, v_2, and v_3 to be added in (10), so that their sum can appear as the global constants C_1 or C_2. First, we will simplify the expressions for v_1, v_2, and v_3 by taking into account that: (i) the multiplication mod $2^{2n}-1$ by 2^{n-1} is the $(n-1)$-bit left-hand cyclic rotation of the $2n$-bit vector; (ii) the multiplication mod $2^{2n}-1$ by $|2^{-k}|_{2^{2n}-1}$ is nothing else but the cyclic right-hand rotation of the $2n$-bit vector; and (iii) the addition mod $2^{2n}-1$ of the term x_1 with negative sign (which is the subtraction of x_1 mod $2^{2n}-1$) relies on complementing x_1 and then padding it with leading ones, so that the width of the resulting vector is a multiple of $s=\lceil k/2n\rceil$ and $2n$ (i.e., the total length of the term becomes $2sn$ bits).

For v_1 from (10), we obtain

$$\begin{aligned} v_1 &= \left| 2^{-k}\left((\underbrace{1\ldots1}_{2sn-k}) \| \bar{x}_1 \right) \right|_{2^{2n}-1} \\ &= \left| \overbrace{2^{-k}(\underbrace{0\ldots0}_{2sn-k}\| \bar{x}_1)}^{v_{1a}} + \overbrace{2^{-k}(\underbrace{1\ldots1}_{2sn-k}\| \underbrace{0\ldots0}_{k})}^{v_{1b}} \right|_{2^{2n}-1}, \end{aligned} \tag{11}$$

where

$$v_{1a} = \left| 2^{-k}(\underbrace{0\ldots0}_{2sn-k}\| \bar{x}_1) \right|_{2^{2n}-1} \tag{12}$$

and

$$\begin{aligned} v_{1b} &= \left| 2^{-k}(\underbrace{1\ldots1}_{2sn-k}\| \underbrace{0\ldots0}_{k}) \right|_{2^{2n}-1} \\ &= \left| 2^{-k}(2^{sn}-2^{k}) \right|_{2^{2n}-1} = \left| 2^{-k}-1 \right|_{2^{2n}-1}, \end{aligned} \tag{13}$$

with the indices a and b denoting the variable and the constant terms, respectively.

The term $(2^n-1)x_2$ from v_2 in (10)

$$v_2 = \left| 2^{-k}\cdot 2^{n-1}(2^n-1)x_2) \right|_{2^{2n}-1} \tag{14}$$

can be split into two separate components $|2^n x_2|_{2^{2n}-1}$ and $|-x_2|_{2^{2n}-1}$, which can be presented conveniently for three special cases depending on the value of x_2:

$$|2^n x_2|_{2^{2n}-1} = \begin{cases} 0 & \text{if } x_2 = 0, \\ (x_{2,(n-1)}\ldots x_{2,0}) \| (\underbrace{0\ldots0}_{n}) & \text{if } 0 < x_2 < 2^n, \\ 1 & \text{if } x_2 = 2^n \end{cases}$$

and

$$|-x_2|_{2^{2n}-1} = \begin{cases} 0 & \text{if } x_2 = 0, \\ (\underbrace{1\ldots1}_{n}) \| (\bar{x}_{2,(n-1)} \ldots \bar{x}_{2,0}) & \text{if } 0 < x_2 < 2^n, \\ \underbrace{1\ldots1}_{n-1} 0 \underbrace{1\ldots1}_{n} & \text{if } x_2 = 2^n. \end{cases}$$

Now, the two above terms can be added separately for the constant and variable terms for respective cases. The constant term is $(\underbrace{1\ldots1}_{n}\underbrace{0\ldots0}_{n})$. Because the constant ones have been removed, the variable component is formed from the concatenation of the vectors with the bits of x_2. To properly handle the case of $x_2 = 2^n$, n bits of the variable vector have to be cleared, i.e., they have to be set to 0 for $x_{2,n} = 1$. It is achieved by ANDing the respective bits (from $\bar{x}_{2,n}$ with $\bar{x}_{2,(n-1)}$ up to $\bar{x}_{2,n}$ with $\bar{x}_{2,0}$) to yield

$$\begin{aligned} |2^n x_2 - x_2|_{2^{2n}-1} = \Big| \big((x_{2,(n-1)} \ldots x_{2,0}) \| \\ (\bar{x}_{2,n}\bar{x}_{2,(n-1)} \ldots \bar{x}_{2,n}\bar{x}_{2,0}) \big) + (\underbrace{1\ldots1}_{n}\underbrace{0\ldots0}_{n}) \Big|_{2^{2n}-1}. \end{aligned} \tag{15}$$

Finally, (14) can be rewritten as $v_2 = |v_{2a} + v_{2b}|_{2^{2n}-1}$, where

$$\begin{aligned} v_{2a} = \Big| 2^{n-1}2^{-k} \cdot \big((x_{2,(n-1)} \ldots x_{2,0}) \| \\ (\bar{x}_{2,n}\bar{x}_{2,(n-1)} \ldots \bar{x}_{2,n}\bar{x}_{2,0}) \big) \Big|_{2^{2n}-1} \end{aligned} \tag{16}$$

and

$$\begin{aligned} v_{2b} &= \Big| 2^{n-1}2^{-k} (\underbrace{1\ldots1}_{n}\underbrace{0\ldots0}_{n}) \Big|_{2^{2n}-1} \\ &= \Big| 2^{n-1}2^{-k}(1-2^n) \Big|_{2^{2n}-1}. \end{aligned} \tag{17}$$

In (10), the extraction of the terms 2^{-k} and 2^{-1} for v_3 yields

$$\begin{aligned} v_3 &= \Big| 2^{-k-1}(x_3 + 2^n x_3) \Big|_{2^{2n}-1} \\ &= \Big| 2^{-k}2^{-1}x_3\big(2^n+1\big) \big) \Big|_{2^{2n}-1}. \end{aligned}$$

First, because x_3 is the n-bit vector and $|x_3(2^n+1)|_{2^{2n}-1}$ is nothing else but the simple concatenation of two n-bit vectors x_3, we first obtain $|x_3(2^n+1)|_{2^{2n}-1} = |(x_3\|x_3)|_{2^{2n}-1}$. Then, due to $|(x_3\|x_3)|_{2^{2n}-1} = |2^n(x_3\|x_3)|_{2^{2n}-1}$, we obtain the final form (containing only the variable term)

$$v_3 = \Big| 2^{n-1}2^{-k}(x_3\|x_3) \Big|_{2^{2n}-1}. \tag{18}$$

Putting new variables v_{1a}, v_{1b}, v_{2a}, v_{2b}, and v_3 (expressed by the respective Eqns (12), (13), and (16)– (18)) into Eqn (10) yields the final desired form with separate constant and variable terms

$$\begin{aligned} V &= |\nu_{1a} + \nu_{1b} + \nu_{2a} + \nu_{2b} + \nu_3|_{2^{2n}-1} \\ &= |\nu_{1a} + \nu_{2a} + \nu_3 + C_1|_{2^{2n}-1}, \end{aligned} \tag{19}$$

where

$$C_1 = \nu_{1b} + \nu_{2b} \tag{20}$$

is the total constant involved in the addition yielding V as a primary part of the final result of conversion.

3.3 Basic hardware implementation

The hardware implementation of the converter using (19), which will be called Version 1, requires a CSA mod $2^{2n} - 1$ with $s \geq 1$ operands for ν_{1a} and 3 operands for ν_{2a}, ν_3, and C_1, i.e., the total of at least 4 operands in two CSA layers (in the special case of $k \leq 2n$), followed by the final adder modulo $2^{2n} - 1$.

To obtain the final addition without C_1 (which can be shifted to the datapath channels), first in (7) for the final value X, V is substituted by its form given by (19) and m_1 with its actual value 2^k, which yields

$$\begin{aligned} X &= x_1 + 2^k |\nu_{1a} + \nu_{2a} + \nu_3 + C_1|_{2^{2n}-1} \\ &= \left| x_1 + 2^k (\nu_{1a} + \nu_{2a} + \nu_3) + 2^k C_1 \right|_{2^k (2^{2n}-1)}. \end{aligned} \tag{21}$$

It is seen from (21) that the final addition given by (19) corresponds to the addition modulo the dynamic range of the RNS, as $2^k(2^{2n} - 1)$ is the product of all moduli. As it was shown in [22], the above addition of a number modulo the dynamic range of the RNS is equivalent to the addition of the respective residues. Therefore, the addition of the constant $2^k C_1$ in (21) is also equivalent to the addition of its residue representation to the input variables of the converter, thus yielding a new set of input residues of the reverse converter.

Note: Henceforth, the superscript c will be used to distinguish the input residues coming from the residue datapath channels with the constant C_1 added beforehand (e.g., x_2 is written as x_2^c).

Because $|2^k C_1|_{2^k} = 0$, no constant is added by the even channel x_1. Therefore, we do not introduce residue x_1^c as the residue x_1 can be passed directly to the converter. Next, we get

$$x_2^c = |x_2 + 2^k C_1|_{2^n+1} \tag{22}$$

$$x_3^c = |x_3 + 2^k C_1|_{2^n-1}. \tag{23}$$

Consequently, we introduce two new variables ν_{2a}^c and ν_3^c, composed similarly as defined by their respective expressions (16) and (18) for ν_{2a} and ν_3, but using only the bits from x_2^c and x_3^c:

$$\begin{aligned} \nu_{2a}^c = \Big| & 2^{n-1} 2^{-k} \cdot \left((x_{2,(n-1)}^c \ldots x_{2,0}^c) \| \right. \\ & \left. (\bar{x}_{2,n}^c \bar{x}_{2,(n-1)}^c \ldots \bar{x}_{2,n}^c \bar{x}_{2,0}^c) \right) \Big|_{2^{2n}-1}, \end{aligned} \tag{24}$$

$$v_3^c = \left| 2^{n-1} 2^{-k} \left(x_3^c \| x_3^c \right) \right|_{2^{2n}-1}. \tag{25}$$

Rewriting (19) using the constant-free notation yields

$$V = \left| v_{1a} + v_{2a}^c + v_3^c \right|_{2^{2n}-1}. \tag{26}$$

The hardware implementation of the converter using (26), which will be called Version 2, requires a CSA with the maximum of $s+2$ operands. Note that at least 3 operands (v_{1a}, v_{2a}^c, and v_3^c) occur for the special case of $k \leq 2n$, handled by one CSA layer, compared to at least two CSA layers in Version 1.

3.4 Further bit-level Optimization

The hardware implementation of v_{2a} and v_{2a}^c (given by (16) and (24)) requires n 2-input AND gates, each of which is driven by $\bar{x}_{2,n}$ or $\bar{x}_{2,n}^c$—the MSB of x_2 (the residue modulo 2^n+1), thus putting high load on its output and requiring the synthesizer either to build a buffer tree or to use gates with higher fanout (which lowers power/delay efficiency). These AND gates also consume power and area, while their only role is to provide the correct value for the only special case of $\bar{x}_{2,n}=0$ or $\bar{x}_{2,n}^c=0$ (i.e.,for $x_2=2^n$). To eliminate them, we have considered alternative compositions of operands and bit-level manipulation schemes. Figure 1 shows bit-level details of the variables v_1 and v_2 as well as their new equivalents w_{1a} and w_{2a} with varying organization for various values of n and k. Recall also that here we deal only with $|k|_{2n}$ least significant bits (LSBs) of x_1, because for $k > 2n$, all the remaining MSBs of x_1 (i.e.,$(x_{1,k} \dots x_{1,|k|_{2n}})$), occupy $s-1$ $2n$-bit slices, each of which is added as a separate CSA operand.

Notes: (i) The terms v_3 and w_3 are not shown, because they are always composed in the same manner. (ii) In Fig. 1, we have omitted multiplications by factors 2^{-k} and 2^{n-1}, as they are simple bit rotations applied uniformly to all operands.

Because bit-level manipulations in the optimized operands w_{2a} and w_{1a} depend on n and k, we analyze four separate cases depending on the value of $|k|_{2n}$: (i) $0 < |k|_{2n} < n-1$ (Fig. 1a); (ii) $n-1 \leq |k|_{2n} < 2n-1$ (Fig. 1b); (iii) $|k|_{2n} = 2n-1$ (Fig. 1c); and (iv) $|k|_{2n} = 0$ (Fig. 1d). For readers' convenience, in Figs. 1a and 1b we show both the initial and the final versions of vectors v_2, v_1 and w_{2a}, w_{1a}, w_{2b}, v_{1b} (the upper and lower part of each figure, respectively); then, only the final versions of the vectors w_{1a} and v_{1b} are given in Fig. 1c and d. The upper part of Fig. 1a shows the initial bit assignment of v_1 and v_2 for $0 < |k|_{2n} < n-1$. By introducing the $2n$-bit all-0's vector and reorganizing all constant bits, we have obtained the final versions of w_{2a} and w_{1a} and the constant vector C_2, shown in the bottom part of Fig. 1a.

$$w_{1a} = \left| 2^{n-1} 2^{-k} \begin{cases} \underbrace{(0\dots0)}_{n-1-k} \| \underbrace{\bar{x}_{1,(k-1)} \dots \bar{x}_{1,0}}_{k} \| \bar{x}_{2,n} \| \underbrace{0\dots0}_{n-1} \| x_{2,n} & \text{if } 0 < |k|_{2n} \leq n-1 \\ \underbrace{(\bar{x}_{1,(n-2)} \dots \bar{x}_{1,0})}_{n-1} \| \bar{x}_{2,n} \| \underbrace{(0\dots0)}_{n-|k|_n-2} \| x_{2,n} \| \underbrace{(\bar{x}_{1,(k-1)} \dots \bar{x}_{1,(n-1)})}_{k-n+1} & \text{if } n-1 < |k|_{2n} \leq 2n-2 \\ \underbrace{(\bar{x}_{1,(n-2)} \dots \bar{x}_{1,0})}_{n-1} \| 0 \| \underbrace{(\bar{x}_{1,(k-1)} \dots \bar{x}_{1,(n-1)})}_{k-n+1} & \text{if } |k|_{2n} = 2n-1 \\ \underbrace{(\bar{x}_{1,(n-2)} \dots \bar{x}_{1,0})}_{n-1} \| \underbrace{(\bar{x}_{1,(k-1)} \dots \bar{x}_{1,(n-1)})}_{k-n+1} & \text{if } |k|_{2n} = 0 \end{cases} \right|_{2^{2n}-1} \tag{27}$$

$$w_{2a} = \left| 2^{n-1}2^{-k} \begin{cases} (x_{2,(n-1)} \dots x_{2,0}) \| (\bar{x}_{2,(n-1)} \dots \bar{x}_{2,0}) & \text{if } 0 < |k|_{2n} \le n-1 \\ (x_{2,(n-1)} \dots x_{2,0}) \| (\bar{x}_{2,(n-1)} \dots \bar{x}_{2,t}) \| \underbrace{(\bar{x}_{2,n}\bar{x}_{2,(t-1)} \dots \bar{x}_{2,n}\bar{x}_{2,0})}_{t=|k|_{2n}-n+1} & \text{if } n-1 < |k|_{2n} \le 2n-2 \\ (x_{2,(n-1)} \dots x_{2,0}) \| (\bar{x}_{2,n}\bar{x}_{2,(n-1)} \dots \bar{x}_{2,n}\bar{x}_{2,0}) & \text{if } |k|_{2n} \in \{0, 2n-1\} \end{cases} \right|_{2^{2n}-1} \tag{28}$$

$$w_{2b} = \left| 2^{n-1}2^{-k} \begin{cases} (\underbrace{1 \dots 1}_{n} \underbrace{0 \dots 0}_{n}) = (1-2^n) & \text{if } |k|_{2n} \in \{0, 2n-1\} \\ (\underbrace{1 \dots 1}_{n-1} \underbrace{0 \dots 0}_{n+1}) = (1-2^{n+1}) & \text{otherwise} \end{cases} \right|_{2^{2n}-1} \tag{29}$$

$$w_{1a}^d = \left| 2^{n-1}2^{-k} \begin{cases} (\underbrace{0 \dots 0}_{n-1-k}) \| \underbrace{\bar{x}_{1,(k-1)} \dots \bar{x}_{1,0}}_{k} \| \bar{x}_{2,n}^d \| \underbrace{0 \dots 0}_{n-1} \| x_{2,n}^d & \text{if } 0 < |k|_{2n} \le n-1 \\ (\underbrace{\bar{x}_{1,(n-2)} \dots \bar{x}_{1,0}}_{n-1}) \| \bar{x}_{2,n}^d \| \underbrace{(0 \dots 0)}_{n-|k|_n-2} \| x_{2,n}^d \| \underbrace{(\bar{x}_{1,(k-1)} \dots \bar{x}_{1,(n-1)})}_{k-n+1} & \text{if } n-1 < |k|_{2n} \le 2n-2 \\ (\underbrace{\bar{x}_{1,(n-2)} \dots \bar{x}_{1,0}}_{n-1}) \| 0 \| \underbrace{(\bar{x}_{1,(k-1)} \dots \bar{x}_{1,(n-1)})}_{k-n+1} & \text{if } |k|_{2n} = 2n-1 \\ (\underbrace{\bar{x}_{1,(n-2)} \dots \bar{x}_{1,0}}_{n-1}) \| \underbrace{(\bar{x}_{1,(k-1)} \dots \bar{x}_{1,(n-1)})}_{k-n+1} & \text{if } |k|_{2n} = 0 \end{cases} \right|_{2^{2n}-1} \tag{30}$$

$$w_{2a}^d = \left| 2^{n-1}2^{-k} \begin{cases} (x_{2,(n-1)}^d \dots x_{2,0}^d) \| (\bar{x}_{2,(n-1)}^d \dots \bar{x}_{2,0}^d) & \text{if } 0 < |k|_{2n} \le n-1 \\ (x_{2,(n-1)}^d \dots x_{2,0}^d) \| (\bar{x}_{2,(n-1)}^d \dots \bar{x}_{2,t}^d) \| \underbrace{\left(\bar{x}_{2,n}^d \bar{x}_{2,(t-1)}^d \dots \bar{x}_{2,n}^d \bar{x}_{2,0}^d\right)}_{t=|k|_{2n}-n+1} & \text{if } n-1 < |k|_{2n} \le 2n-2 \\ (x_{2,(n-1)}^d \dots x_{2,0}^d) \| (\bar{x}_{2,n}^d \bar{x}_{2,(n-1)}^d \dots \bar{x}_{2,n}^d \bar{x}_{2,0}^d) & \text{if } |k|_{2n} \in \{0, 2n-1\} \end{cases} \right|_{2^{2n}-1} \tag{31}$$

Prior analyzing all remaining cases of $|k|_{2n}$, we consider the possibilities of some simplifications related to $x_{2,n}$—the MSB of x_2. Because $\bar{x}_{2,n} = 1$, for all values of x_2 but $x_2 = 2^n$, the addition of a pair of bits $\bar{x}_{2,n}$ and $x_{2,n}$ (see w_{1a} in Fig. 1a) is actually executed only for $x_2 = 2^n$, resulting in $|v_2 - w_{2b}|_{2^{2n}-1} = 2^n$. The same result is obtained by clearing n rightmost bits of w_{2a} and replacing $\bar{x}_{2,n}$ of v_2 with constant one, using the same approach as already used in the Subs. 3.2 while deriving v_2 in (16).

For $n-1 \le |k|_{2n} < 2n-1$, similarly as in the previous case, the upper and the bottom parts of Fig. 1b, respectively, show the initial bit assignment of v_1 and v_2, the final versions of w_{2a} and w_{1a}, and the constant vector C_2. However, because in this case there was no place for $x_{2,n}$, as $t = |k|_{2n} - n + 1$ rightmost bit positions in w_{1a} are occupied by the bits of x_1, the following extra bit manipulations were needed. We have shown above that because $\bar{x}_{2,n}$ and $x_{2,n}$ are actually added only for $x_2 = 2^n$, it resulted in $|v_2 - w_{2b}|_{2^{2n}-1} = 2^n$. The same result is obtained by clearing t rightmost bits of w_{2a} when $x_2 = 2^n$ and moving $x_{2,n}$ to the tth position.

In the remaining two cases, i.e.,for $|k|_{2n} = 2n-1$ and $|k|_{2n} = 0$, there is no available 2-bit field to insert the pair $\bar{x}_{2,n}$ and $x_{2,n}$ into $w_{1,a}$. Again, in either case, the fact that the bit position occupied by $\bar{x}_{2,n}$ is equal to one for all values of x_2 but $x_2 = 2^n$ was taken into account by augmenting w_{2b} with another constant one at the nth bit position of weight 2^n (i.e., $w_{2b,n} = 1$). The final bit distributions are shown in Fig. 1c and d, respectively.

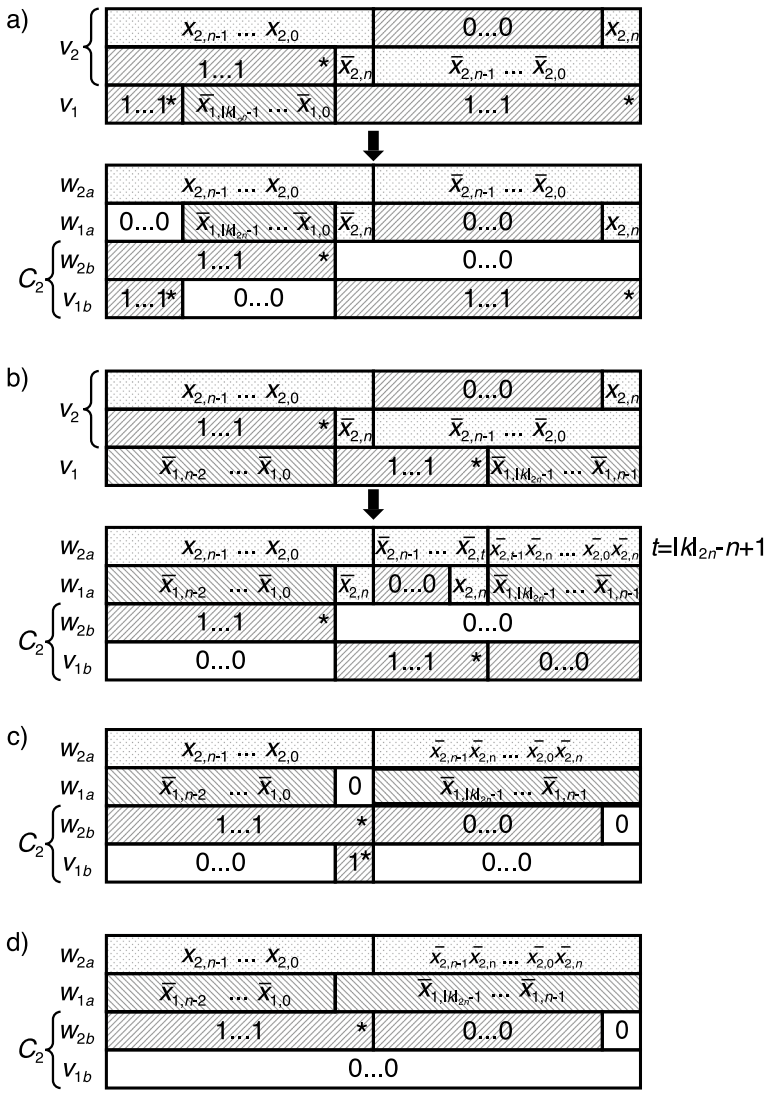

Fig. 1 Bit-level manipulations (excluding v_3) leading to hardware simplifications: **a** $|k|_{2n} \leq n-1$; **b** $n-1 < |k|_{2n} < 2n-1$; **c** $|k|_{2n} = 2n-1$; and **d** $|k|_{2n} = 0$. Asterisk "*" fields are zeroed when the constant addition is shifted from the converter to the datapath channels

In summary, depending on k, three cases occur: (i) no AND gates for $0 < |k|_{2n} < n-1$; (ii) $t = |k|_{2n} - n - 1$ AND gates for $n-1 \leq |k|_{2n} < 2n-1$ (clearly, for many practical cases $t = n-k$ is small); and (iii) n AND gates for $|k|_{2n} = 2n-1$ and $|k|_{2n} = 0$.

These modifications have led to the new equation for V

$$V = |w_{1a} + w_{2a} + v_3 + C_2|_{2^{2n}-1}, \tag{32}$$

where

$$C_2 = v_{1b} + w_{2b} \tag{33}$$

is the new constant, w_{1a}, w_{2a}, and w_{2b} are given by Eqns (27)–(29), and v_3 is given by (18). The hardware implementation of the converter using (32) will be called Version 3.

3.5 Optimized hardware implementation

To obtain the final addition without C_2 (which can be shifted to the datapath channels), we proceed similarly as for Version 2. First, Eqn (32) allows to substitute V in (4) to obtain

$$\begin{aligned} Z &= x_1 + 2^k |w_{1a} + w_{2a} + v_3 + C_2|_{2^{2n}-1} \\ &= \left| x_1 + 2^k (w_{1a} + w_{2a} + v_3) + 2^k C_2 \right|_{2^k 2^{2n}-1}. \end{aligned} \tag{34}$$

Then, the residues with added constants in odd channels (again, no constant is added in the even channel x_1) are

$$\begin{aligned} x_2^d &= |x_2 + 2^k C_2|_{2^n+1} \\ x_3^d &= |x_3 + 2^k C_2|_{2^n-1}. \end{aligned}$$

Consequently, we introduce three new variables defined by their respective Eqns (27), (28), and (18), but using only the bits from x_1, x_2^d and x_3^d: w_{1a}^d (Eqn (30)), w_{2a}^d (Eqn (31)), and

$$v_3^d = \left| 2^{n-1} 2^{-k} \left(x_3^d \| x_3^d \right) \right|_{2^{2n}-1}. \tag{35}$$

Rewriting (32) using constant-free notation yields

Table 1 Summary of hardware options

Version	Input operands of CSA	Equations	Constant in channels 2 & 3
1	v_{1a}, v_{2a}, v_3, C_1	(12), (16), (18), (20)	–
2	v_{1a}, v_{2a}^c, v_3^c	(12), (24), (25)	$2^k C_1$
3	w_{1a}, w_{2a}, v_3, C_2	(27), (28), (18), (33)	–
4	$w_{1a}^d, w_{2a}^d, v_3^d$	(30), (31), (35)	$2^k C_2$

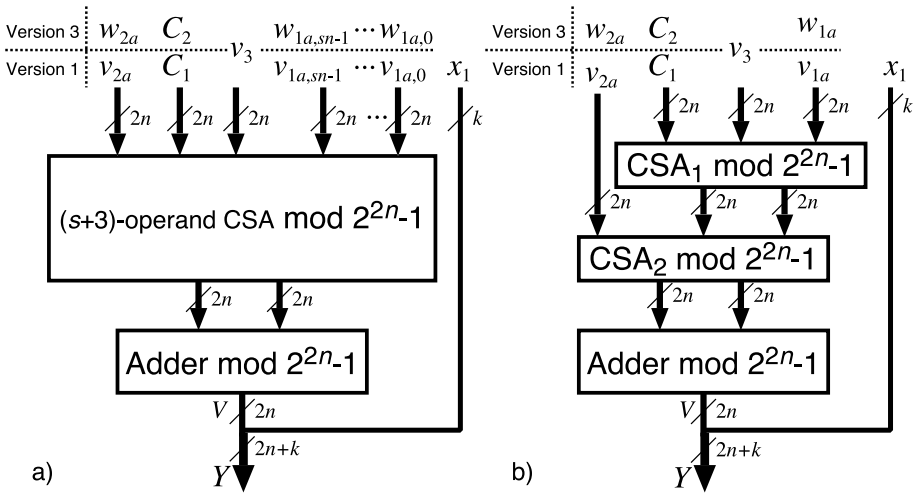

Fig. 2 Converter architecture: **a** general case, **b** special case of $k \leq 2n$

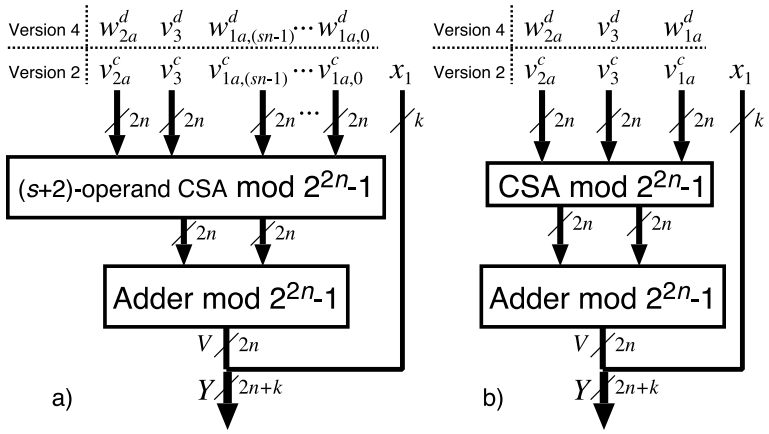

Fig. 3 Converter architecture with constants shifted to the RNS datapath channels: **a** general case, **b** special case of $k \leq 2n$

$$V = \left| w_{1a}^{d} + w_{2a}^{d} + v_{3}^{d} \right|_{2^{2n}-1}, \tag{36}$$

which is used in the hardware implementation of Version 4.

The details of the four versions proposed here are summarized in Table 1—detailing input operands to the CSA tree of the converter, and Figs. 2 and 3—showing their hardware implementations. In Versions 1 and 3, the constants are added by the converter, whereas in Versions 2 and 4, the constants are added in the datapath channels. Versions 3 and 4 feature optimized bit-level manipulations resulting in lower number of AND gates. Also, in Versions 1 and 2, the number of AND gates increases from 3 to 13 jumping by one for every value of n for the dynamic ranges from 8 to 40 bits, whereas in Version 3 and 4, it is kept between 1 and 3. Each version of the converter uses a CSA with end-around carry (EAC) modulo $2^{2n} - 1$ with the appropriate number of operands, followed by the 2-operand adder modulo $2^{2n} - 1$.

3.6 Example

The methods proposed above will be illustrated for the sample moduli set with $n = 3$ and $k = 5$, i.e., $\{2^5, 2^3 + 1, 2^3 - 1\} = \{32, 9, 7\}$. V from (19) will be calculated mod $2^6 - 1$. The binary representations of the residues mod 32, 9, and 7 are, respectively, $x_1 = (x_{1,4} \ldots x_{1,0})$, $x_2 = (x_{2,3} \ldots x_{2,0})$, and $x_3 = (x_{3,2} \ldots x_{3,0})$.

According to the proposed scheme, to compute v_{1a} from (12) we: (i) extend x_1 to six bits $(0 \| (x_{1,4} \ldots x_{1,0}))$, (ii) bitwise complement the variable bits to obtain $(\bar{x}_{1,4} \ldots \bar{x}_{1,0})$, and (iii) apply the right cyclic shift by five positions (i.e.,multiply by 2^{-5}), which results in $v_{1a} = ((\bar{x}_{1,4} \ldots \bar{x}_{1,0}) \| 0)$. Next, the term v_{2a} from (16) is constructed by concatenating the inverted and the simple copy of three LSBs of x_2, each multiplied by the inverted bit $x_{2,3}$, resulting in $v_{2a} = ((\bar{x}_{2,3}\bar{x}_{2,2} \ldots \bar{x}_{2,3}\bar{x}_{2,0}) \| (x_{2,2} \ldots x_{2,0}))$. Six bits of the term v_3 (Eqn (18) are obtained by concatenating two copies of the residue x_3, i.e., $v_3 = ((x_{3,2} \ldots x_{3,0}) \| (x_{3,2} \ldots x_{3,0}))$. The constant term C_1 is calculated from v_{1b} and v_{2b} (Eqns (13) and (17)) as $C_1 = |\underbrace{2^{-5} - 1}_{v_{1b}=1} + \underbrace{2^{3-1} \cdot 2^{-5}(1 - 2^3)}_{v_{2b}=7}|_{63} = 8$ (in Fig. 4, it is added by HA^{+1}). The bit distribution for all four components is illustrated in Table 2.

Table 2 Bit distribution of the sample converter for $n = 3$ and $k = 5$ after constants integration

v_1	$\bar{x}_{1,4}$	$\bar{x}_{1,3}$	$\bar{x}_{1,2}$	$\bar{x}_{1,1}$	$\bar{x}_{1,0}$	0
v_{2a}	$\bar{x}_{2,3} \cdot \bar{x}_{2,2}$	$\bar{x}_{2,3} \cdot \bar{x}_{2,1}$	$\bar{x}_{2,3} \cdot \bar{x}_{2,0}$	$x_{2,2}$	$x_{2,1}$	$x_{2,0}$
C_1	0	0	1	0	0	0
v_3	$x_{3,2}$	$x_{3,1}$	$x_{3,0}$	$x_{3,2}$	$x_{3,1}$	$x_{3,0}$

Now we will consider the following sample values of the input residues: $\{x_1, x_2, x_3\} = \{1, 2, 3\}$. Then, $v_{1a} = \left|2^{-5}(011110)_b\right|_{63} = (111100)_b = 60$, $v_{2a} = \left|2^{3-1} \cdot 2^{-5}(010101)_b\right|_{63} = (101010)_b = 42$, $v_3 = \left|2^{3-1} \cdot 2^{-5}(011011)_b\right|_{63} = (011011)_b = 27$. Next, $V = |60 + 42 + 27 + 8|_{63} = 11$ and the final value $Z = 1 + 11 \cdot 32 = 353$. Figure 4 shows the detailed scheme of this design with the above sample values.

For the version with the constants shifted to the datapath channels, we have $x_2^c = |x_2 + 2^5 \cdot 8|_9 = |x_2 + 4|_9$ and $x_3^c = |x_3 + 2^5 \cdot 8|_7 = |x_3 + 4|_7$. v_{1a} is the same as above, whereas $v_{2a}^c = \left((\bar{x}_{2,3}^c \bar{x}_{2,2}^c \ldots \bar{x}_{2,3}^c \bar{x}_{2,0}^c) \| (x_{2,2}^c \ldots x_{2,0}^c)\right)$ and $v_{3a}^c = \left((x_{3,2}^c \ldots x_{3,0}^c) \| (x_{3,2}^c \ldots x_{3,0}^c)\right)$. Now the sample input residues, which actually appear on the reverse converter inputs, can be modified from $\{x_1, x_2, x_3\}$ to $\{x_1, x_2^c, x_3^c\} = \{1, |2 + 4|_9, |3 + 4|_7\} = \{1, 6, 0\}$. Then $v_{2a}^c = \left|2^{3-1} \cdot 2^{-5}(110001)_b\right|_{63} = (001110)_b = 14$, $v_3^c = \left|2^{3-1} \cdot 2^{-5}(000000)_b\right|_{63} = (000000)_b = 0$, so that $V = |60 + 14 + 0|_{63} = 11$, and the final value is $Z = 1 + 11 \cdot 32 = 353$. Figure 5 shows the detailed scheme of this design for the above sample values.

4 Complexity evaluation

Basically, the flexible 3-moduli set $\{2^k, 2^n - 1, 2^n + 1\}$ offers for any dynamic range (DR) a number of alternative choices spanning over a range of n and k (in case of [4, 12] the choices are limited to $n \leq k \leq 2n$), whereas the conventional 3-moduli set

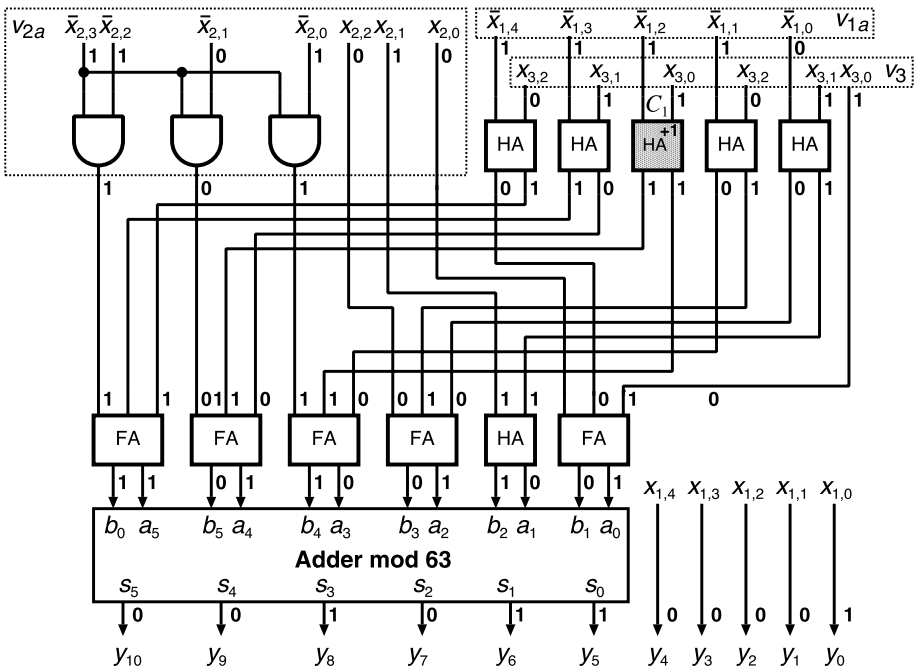

Fig. 4 Detailed scheme of the basic sample converter for $n = 3$ and $k = 5$

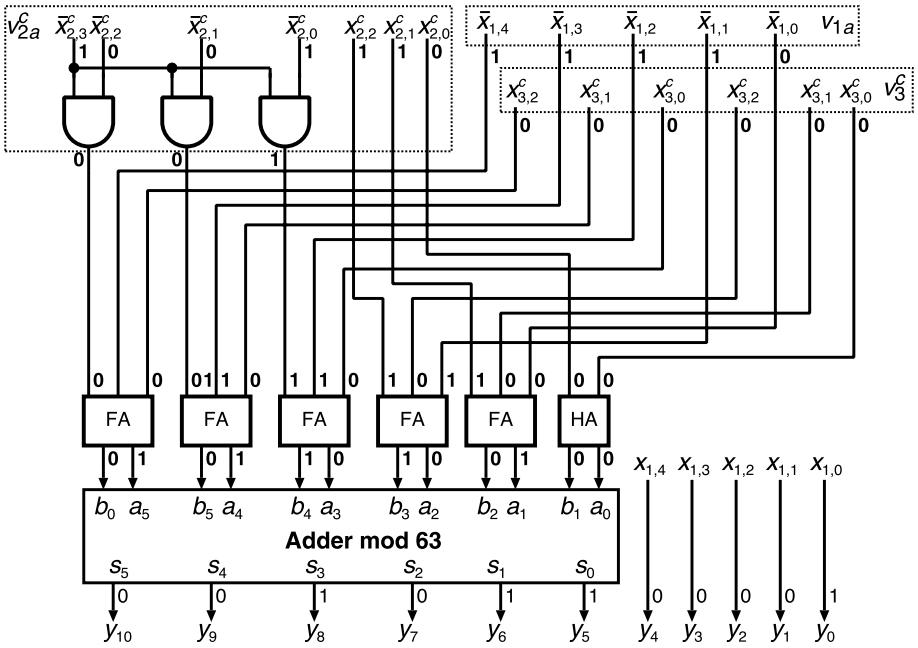

Fig. 5 Detailed scheme of the sample converter for $n = 3$ and $k = 5$ with the constants shifted to the datapath channels

$\{2^n, 2^n - 1, 2^n + 1\}$ allows for only one solution for a given n, thus providing the dynamic range of $3n - 1$ bits. Because in practical applications the converters for the sets of well-balanced moduli seem of the main interest, for any dynamic range we have considered only one pair (k, n) with $k = \{n + 1, n + 2, n + 3\}$. For instance, to cover the dynamic ranges of 18–20 bits, we have considered three following pairs: (7, 6), (8, 6), and (9, 6).

All four versions of our designs will be compared against the reverse converters for the same 3-moduli set $\{2^k, 2^n - 1, 2^n + 1\}$, designed using the methods from [4, 12], and the best-known converters for the 3-moduli set $\{2^n, 2^n - 1, 2^n + 1\}$ proposed recently by us in [24]. (To note that the converter equations from both [4] and [12] do not allow for shifting constants to datapath channels.)

4.1 Gate-level complexity evaluation

The gate-level complexity figures of all converters considered are summarized in Table 3. For the proposed designs, we give two figures: the one for general k and the second for k close to n. Here, FA, HA, and HA^{+1}, respectively, denote a full-adder, a half-adder, and a half-adder with increment (which is nothing else but the full-adder with one constant input 1); obviously, the complexity of an HA^{+1} is assumed the same as of an HA. The delay of an a-operand CSA tree is denoted by $d_{CSA}(a)$. When one of CSA operands is a constant, its delay is denoted by $d_{CSA}(a, 1_c)$. In the critical path delay column of Table 3, we neglect the delay of the final adder mod $2^{2n} - 1$, because it occurs in all converters considered here.

Because for k of most practical importance, i.e., $n < k \leq 2n$, we have $\lceil k/2n \rceil = 1$, the CSA tree used in any new converter consists of one CSA stage for Versions 2 and 4 and two CSA stages for Versions 1 and 3. In the latter case, the first CSA stage contains only HAs, so that the delay of the first CSA stage of the proposed design is one HA compared

Table 3 Gate complexity of the proposed converter and other known converters for some similar RNSs

Moduli Set/Design	FA	HA/HA^{+1}	AND/OR (XOR)	Adder mod $(2^{2n}-1)$	Inverters	Critical Path Delay (w/o $d_{add\ mod\ 2^{2n}-1}$)
$\{2^n, 2^n-1, 2^n+1\}$	n	n	n	1	$2n$	$d_{inv}+d_{OR}+d_{FA}$
Patronik and Piestrak [24]						
$\{2^k, 2^n-1, 2^n+1\}$	$2n-k+1$	$n+k$	2	1	$k+1$	$d_{inv}+d_{OR}+2d_{FA}$
Chaves and Sousa [4]						
Version 1	k	$4n-\lvert k\rvert_{2n}$	n	1	$n+k$	$d_{inv}+d_{AND}+d_{CSA}(\lceil \frac{k}{2n} \rceil, 1_c)$ $(d_{inv}+d_{AND}+d_{FA}+d_{HA})$
Version 2	k	$2n-\lvert k\rvert_{2n}$	n	1	$n+k$	$d_{inv}+d_{AND}+d_{CSA}(\lceil \frac{k}{2n} \rceil)$ $(d_{inv}+d_{AND}+d_{FA})$
Version 3	$k+2c$	$4n-\lvert k+2c\rvert_{2n}$	$f_A(n,k)^{\dagger}$	1	$f_I(n,k)^{\ddagger}+k$	$d_{inv}+d_{AND}+d_{CSA}(\lceil \frac{k+2c}{2n} \rceil, 1_c)$ $(d_{inv}+d_{AND}+d_{FA}+d_{HA})$
Version 4	$k+2c$	$2n-\lvert k+2c\rvert_{2n}$	$f_A(n,k)^{\dagger}$	1	$f_I(n,k)^{\ddagger}+k$	$d_{inv}+d_{AND}+d_{CSA}(\lceil \frac{k+2c}{2n} \rceil)$ $(d_{inv}+d_{AND}+d_{FA})$
[12] $p=0$	$2n$	0	2 (+1 XOR)	1	n	d_{FA}
[12] $p=1$	$2n$	0	$n+1$(+1 XOR)	1	n	$d_{FA}+d_{OR}+d_{inv}$
[12] $p>1$	$2n$	0	$n+p$ (+p XOR)	1	n	$d_{FA}+d_{OR}$

$c=\begin{cases}0 & \text{if } \lvert k\rvert_{2n}=\{0,2n-1\}\\ 2 & \text{otherwise}\end{cases}$
$t=\lvert k\rvert_{2n}-n+1$ $\dagger f_A(n,k)=\begin{cases}0 & \text{if } \lvert k\rvert_{2n}\le n-1\\ t & \text{if } n-1<\lvert k\rvert_{2n}<2n-1\\ n & \text{if } \lvert k\rvert_{2n}\in\{0,2n-1\}\end{cases}$ $\ddagger f_I(n,k)=\begin{cases}n \text{ if} \lvert k\rvert_{2n}\in\{0,2n-1\}\\ n+1 \text{ otherwise}\end{cases}$

one FA in [4]. Finally, The design from [12] has also one CSA stage composed of FAs complemented by XORs, although thanks to a special design technique its delay is one FA and between zero and one OR, depending on the width of the even modulus k. Our new converters with flexible dynamic range also exhibit delay close to the best designs of [24] for the classic 3-moduli set with fixed dynamic range $\{2^n, 2^n - 1, 2^n + 1\}$. As for the gate count, the proposed converters fall close to the gate count of their counterparts from [24] (in the case of constant-free versions) or are larger by $2n$ HAs. Hence, we expect the delay of our converters to be close to the converters from [12, 24] and smaller than those from [4]. Versions 1 and 2 of our converter contain a number of AND gates with one input driven by the same signal, which may cause some fanout problem not revealed by standard metrics. This issue, however, depends on the actual gate library used and may be eventually exposed by the actual synthesis results.

4.2 Synthesis assumptions and constraints

To obtain as much realistic as possible complexity figures, in an attempt to produce systematic and fairly comparable descriptions, all converters were implemented in parameterized structural Verilog, following the identical coding guidelines. In all cases, the multi-operand adder mod $2^{2n} - 1$ was implemented using the CSA with EAC, followed by the final 2-operand adder mod $2^{2n} - 1$ of [19]. The FAs used in the CSAs have been implemented using register transfer level (RTL) 1-bit adder ("+" operator in Verilog RTL code), except for the special FAs proposed in [12]. In the latter case, we have implemented two versions, using the special and regular (RTL) FAs.

All converters were synthesized using Cadence RTL Compiler v. 8.1 over the commercial CMOS065LP 65-nm low-power library from ST Microelectronics. Each combinational design was complemented with an input and output register for more realistic delay and power estimations. In each case of the version and dynamic range, the minimum delay was found, which we understood as the minimum delay constraint for which the logic synthesizer still reported nonnegative timing slack and with no violating paths. In both logical and physical synthesis stages, the fanout settings have been kept at the cell library and synthesizer defaults without additional restrictions.

Next, we have placed and routed our designs using Cadence EDI v. 10.11 (our place-and-route (PnR) tool) on rectangular cores of dimensions around 130–180 μm. To level out area and leakage power comparison, we have adjusted the core dimensions to keep the target density at 75–80%. Since the designs at small dynamic ranges have low gate count, there were a few isolated cases, for which we were unable to stay at our target density bounds, so we have left out smaller density of around 73%. Consequently, we have placed the area constraint while keeping the delay at the values obtained during the logic synthesis phase. Then, we have performed power simulations on the netlists from the PnR tool, with delay annotations for 1000 random vectors using Synopsys PrimeTime 12.12H. The synthesis and simulation results obtained for various converters for all dynamic ranges from 8 to 40 bits are visualized in Figs. 6, 7, and 8, which, respectively, show the delay, power, and area estimations.

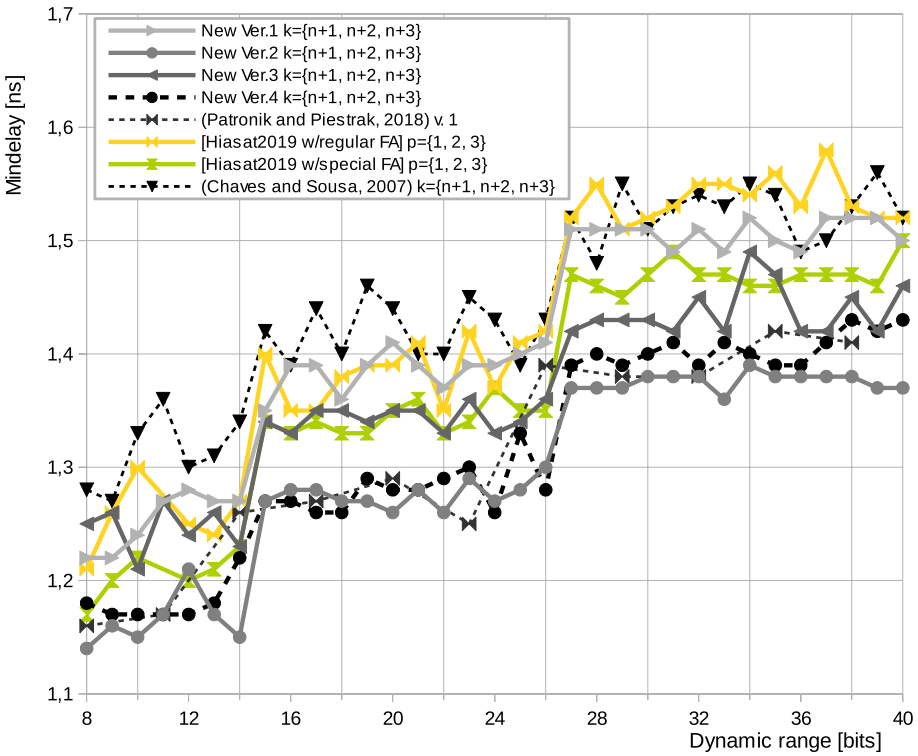

Fig. 6 Minimum delay as a function of the dynamic range

4.3 Delay evaluation

Figure 6 reveals the logarithmic growth of the delay with the dynamic range, which is the direct outcome of employing in all designs a logarithmic depth parallel prefix adder [19] for the final mod $2^{2n} - 1$ addition. Delay comparison of Versions 1 and 3 of our converter (in which the constants are added within the converter) against Versions 2 and 4 (with the constants shifted out to the datapath channels) clearly reveals the advantages of the latter: The compression of residues into three rather than four variables has allowed to avoid one CSA layer. Compared to [4, 12] for almost all dynamic ranges the Versions 2 and 4 are faster (except one case for Version 2). Also, for the dynamic ranges of $3n - 1$ bits offered by the 3-moduli set $\{2^n, 2^n - 1, 2^n + 1\}$, in most cases there is a version of the new converter which is faster compared to the reverse converter for the latter set from [24], which clearly reveals yet another advantage of using the flexible moduli set.

4.4 Power consumption evaluation

Figure 7 shows the power consumption of all designs considered. While the minimal delay obtained was relatively easy to explain, as it was strictly related to the number of logic levels, and hence, it was easily traceable by evaluating the critical paths, the power consumption figures are the results of simulations performed on placed and routed designs using the sets of random vectors. Due to this, the power consumption estimation and its contributing factors are to be explained on the basis of the internal architecture, the size of the circuit, and the rate of recharging gates, resulting from the target delay. Additionally, because all converters considered are relatively small

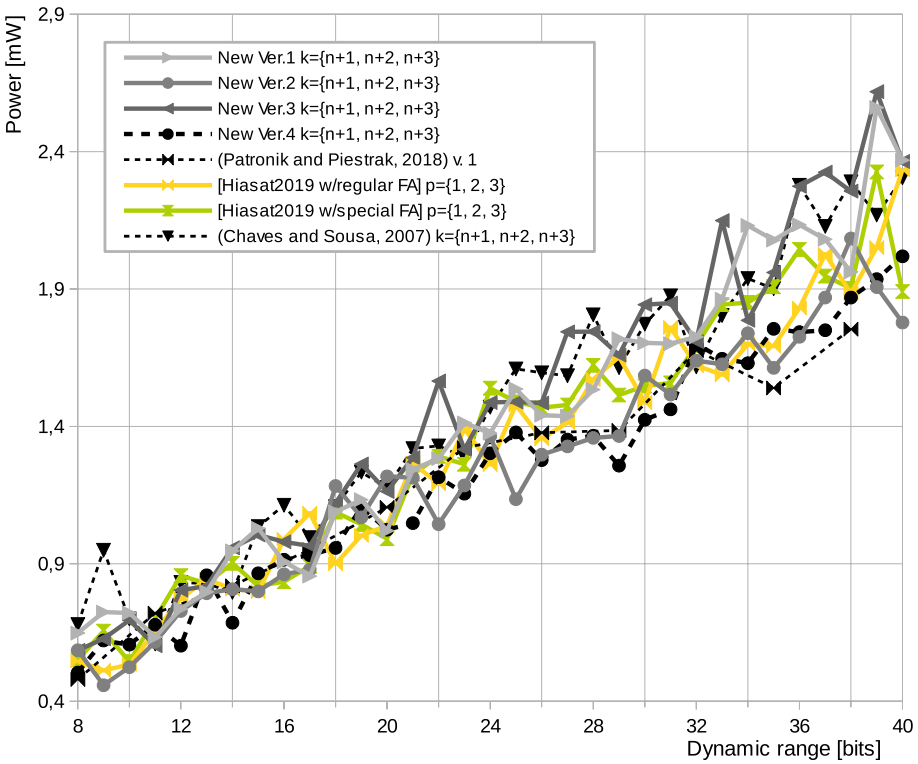

Fig. 7 Power at the minimum delay

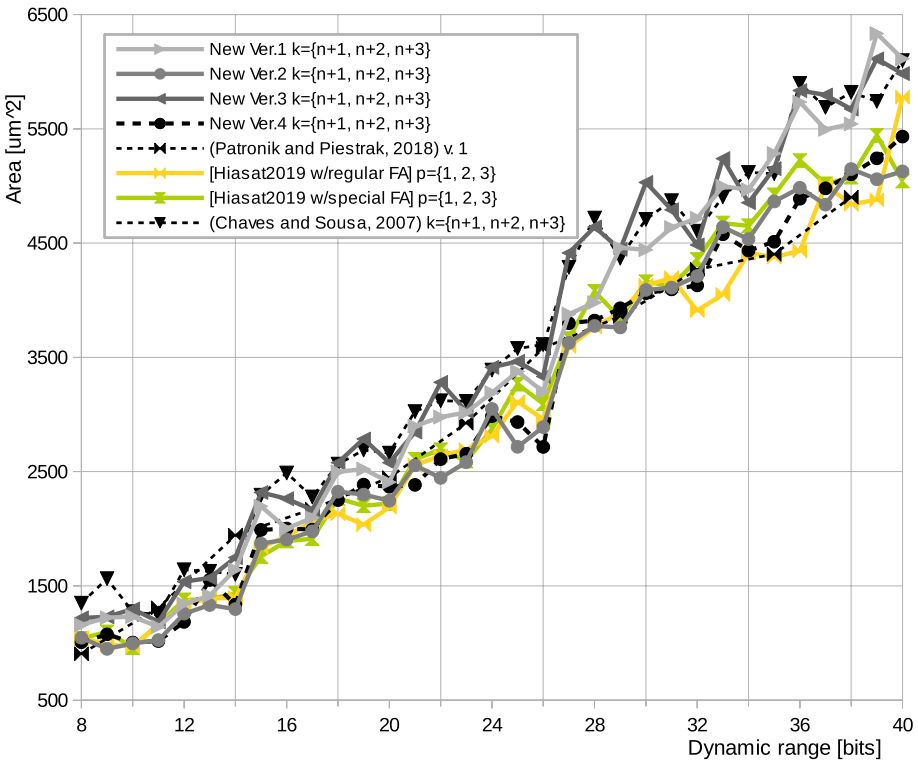

Fig. 8 Area at the minimum delay

designs (containing at most a few hundreds of logic cells), every difference between the circuits may have noticeable impact on power simulation results. As a result, the power curves may seem appear a little noisy, with overall trend resulting from the area complexity of the circuit and some local deviations negatively correlated with delay (i.e., the smaller delay, the larger power consumption). On the one hand, we can observe the general linear growth of power consumption with n, which is due to the linear growth (also with n) of the number of adders in CSA layers and almost linearly ($n \log n$) growing area of the final adder mod $2^{2n} - 1$. On the other hand, we can also observe local trends with fixed n but changing k, as there are additional cells coming with additional input lines in the even channel mod 2^k. The abrupt growths resulting from discrete logarithmic dependence of the delay of the adder are still observable for n changing with the power of 2, but they are mostly lost in the overall trend. Next, the power curves may be seen as separated into two groups. The first group, with smaller power consumption, contains the power curves of Versions 2 and 4 of the proposed design as well as of the converter from [24], whereas the second group contains Versions 1 and 3 of the proposed design as well as of the design from [4]. The converters from [12] falls between these two groups. Since both the proposed design and the converter from [24] have constants removed, they have similar power consumption. On the other hand, in most cases at least one from our proposed designs enjoys smaller power consumption than its direct counterparts from [4, 12].

4.5 Area evaluation

Figure 8 shows the area of the synthesized converters. The overall trend is almost linear, which is consistent with the complexity of the final adder mod $2^{2n} - 1$ ($O(n \log n)$) along with the number of additional gates required with growing n and k. There are two noticeable increments on all area curves, namely for the dynamic ranges changing from 14 to 15 bits and from 26 to 27 bits (corresponding to n changing from 4 to 5 and from 8 to 9, respectively), resulting directly from the increments of the number of prefix operator layers in the final adder mod $2^{2n} - 1$. In the general case, such incrementations would occur at any change of n from 2^r to $2^r + 1$. In turn, this is consistent with changing number of gates along the critical path and consequent area increase of the whole circuit (as the synthesizer tries to select larger gates in order to regain delay). As these figures strictly depend on the numbers of cells used, they are more consistent than power estimations: slower/faster designs tend to have smaller/larger area due to smaller/larger cells, respectively. Due to the smaller number of additions (there are no constants to be added in the converter), our circuits are generally smaller than their counterparts from [4]. In comparison with [12], some of the new converters proposed here are larger, especially for larger dynamic ranges. This is most likely the result of larger arithmetic components used by the synthesizer to achieve smaller delay.

4.6 Improvement assessment

Table 4 summarizes percentage improvements of our converters with respect to their counterparts of [4, 12]. For a given dynamic range, the best of our versions is selected for comparison, although—to emphasize the advantages of shifting constants to the

Table 4 Percentage assessment of improvements

	Area [μm^2]					Delay [ns]					Power [μW]				
	Best		CS	Red.[%]		Best		CS	Red.[%]		Best		CS	Red.[%]	
DR	{v1,v3}	{v2,v4}	or H	{v1,v3}	{v2,v4}	{v1,v3}	{v2,v4}	or H	{v1,v3}	{v2,v4}	{v1,v3}	{v2,v4}	or H	{v1,v3}	{v2,v4}
8	1158	1007	1039	− 11.5	3.1	1.22	1.14	1.17	− 4.3	2.6	0.585	0.504	0.546	− 7.1	7.7
9	1222	949	984	− 24.2	3.6	1.22	1.16	1.2	− 1.7	3.3	0.627	0.458	0.512	− 22.5	10.5
10	1232	997	960	− 28.3	− 3.9	1.21	1.15	1.22	0.8	5.7	0.695	0.524	0.534	− 30.1	1.9
11	1142	1017	1246	8.3	18.4	1.27	1.17	1.36	6.6	14.0	0.602	0.616	0.608	1.0	− 1.3
12	1342	1185	1340	− 0.1	11.6	1.24	1.17	1.2	− 3.3	2.5	0.736	0.602	0.773	4.8	22.2
13	1415	1333	1383	− 2.3	3.6	1.26	1.17	1.21	− 4.1	3.3	0.798	0.793	0.828	3.6	4.3
14	1642	1297	1397	− 17.5	7.2	1.23	1.15	1.23	0.0	6.5	0.947	0.685	0.796	− 17.2	15.2
15	2197	1870	1758	− 25.0	− 6.4	1.34	1.27	1.34	0.0	5.2	1.005	0.799	0.799	− 25.8	− 0.1
16	2001	1906	1893	− 5.7	− 0.7	1.33	1.27	1.33	0.0	4.5	0.907	0.862	0.835	− 8.6	− 3.2
17	2097	1977	1913	− 9.6	− 3.3	1.35	1.26	1.34	− 0.7	6.0	0.855	0.879	0.891	4.0	1.4
18	2498	2251	2133	− 17.1	− 5.5	1.35	1.26	1.33	− 1.5	5.3	1.089	0.957	0.9	− 21.0	− 6.4
19	2522	2299	2036	− 23.9	− 12.9	1.34	1.27	1.33	− 0.8	4.5	1.133	1.069	1.003	− 13.0	− 6.6
20	2408	2245	2188	− 10.1	− 2.6	1.35	1.26	1.35	0.0	6.7	1.02	1.024	0.991	− 2.9	− 3.3
21	2846	2384	2556	− 11.3	6.7	1.35	1.28	1.36	0.7	5.9	1.243	1.048	1.226	− 1.4	14.5
22	2977	2445	2639	− 12.8	7.4	1.33	1.26	1.33	0.0	5.3	1.283	1.044	1.191	− 7.7	12.3
23	3018	2583	2593	− 16.4	0.4	1.36	1.29	1.34	− 1.5	3.7	1.316	1.155	1.262	− 4.3	8.5
24	3191	2986	2810	− 13.6	− 6.3	1.33	1.26	1.37	2.9	8.0	1.375	1.302	1.261	− 9.0	− 3.3
25	3376	2717	3112	− 8.5	12.7	1.34	1.28	1.35	0.7	5.2	1.489	1.135	1.478	− 0.7	23.2
26	3200	2716	2960	− 8.1	8.2	1.36	1.28	1.35	− 0.7	5.2	1.441	1.277	1.355	− 6.3	5.8
27	3879	3627	3595	− 7.9	− 0.9	1.42	1.37	1.47	3.4	6.8	1.437	1.327	1.414	− 1.6	6.2
28	3980	3775	3764	− 5.7	− 0.3	1.43	1.37	1.46	2.1	6.2	1.534	1.358	1.571	2.4	13.6
29	4445	3762	3845	− 15.6	2.2	1.43	1.37	1.45	1.4	5.5	1.660	1.257	1.514	− 9.6	17.0
30	4442	4062	4129	− 7.6	1.6	1.43	1.38	1.47	2.7	6.1	1.704	1.424	1.487	− 14.6	4.2

Table 4 (continued)

	Area [μm^2]					Delay [ns]					Power [μW]				
	Best		CS	Red.[%]		Best		CS	Red.[%]		Best		CS	Red.[%]	
DR	{v1,v3}	{v2,v4}	or H	{v1,v3}	{v2,v4}	{v1,v3}	{v2,v4}	or H	{v1,v3}	{v2,v4}	{v1,v3}	{v2,v4}	or H	{v1,v3}	{v2,v4}
31	4637	4095	4124	− 12.4	0.7	1.42	1.38	1.49	4.7	7.4	1.701	1.462	1.558	− 9.2	6.2
32	4484	4130	3909	− 14.7	− 5.7	1.45	1.38	1.47	1.4	6.1	1.709	1.639	1.617	− 5.4	− 1.1
33	4998	4577	4050	− 23.4	− 13.0	1.42	1.36	1.47	3.4	7.5	1.863	1.626	1.588	− 17.3	− 2.4
34	4854	4434	4412	− 10.0	− 0.5	1.49	1.39	1.46	− 2.1	4.8	1.786	1.63	1.7	− 5.1	4.1
35	5152	4512	4377	− 17.7	− 3.1	1.47	1.38	1.46	− 0.7	5.5	1.960	1.613	1.692	− 15.8	4.7
36	5734	4888	4432	− 29.4	− 10.3	1.42	1.38	1.47	3.4	6.1	2.134	1.726	1.831	− 16.5	5.7
37	5494	4837	5006	− 9.7	3.4	1.42	1.38	1.47	3.4	6.1	2.081	1.750	1.947	− 6.9	10.1
38	5543	5102	4840	− 14.5	− 5.4	1.45	1.38	1.47	1.4	6.1	1.962	1.869	1.885	− 4.1	0.8
39	6112	5059	4879	− 25.3	− 3.7	1.42	1.37	1.46	2.7	6.2	2.560	1.907	2.051	− 24.8	7.0
40	5983	5128	5044	− 18.6	− 1.7	1.46	1.37	1.5	2.7	8.7	2.364	1.778	1.891	− 25.0	6.0
			Min	− 29.4	− 13.0				− 4.3	2.5				− 30.1	− 6.6
			Max	− 0.1	12.7				6.6	14.0				4.8	23.2
			Average	− 14.3	− 0.4				0.7	5.8				− 9.6	5.6

datapath channels—two separate comparisons are presented: one for the converters with the integrated constants (Versions 1 and 3) and another for the converters with the separated constants (Versions 2 and 4). On the other side, we have selected the best of earlier designs of [4] and [12] with special and regular FAs (respectively, denoted as *CS or H*). For Versions 1 and 3, the delay of the new converters is reduced on average by 0.7%, from −4.3% up to 6.6%. The area is increased by 14.3% on average, up to about 29.4%. The delay savings are accompanied by increase in the power consumption—on average by about 9.6%, up to 30.1%. For Versions 2 and 4, higher gains are observed: the delay is reduced on average by 5.8%, from 2.5% up to 14.0%; the area is increased on average by 0.4%, up to 13% (but the maximal reduction achieved for $DR = 25$ is 12.7%); and the power consumption is reduced on average by about 5.6%, up to 23.2%. Higher savings in the converters with separated constants stem directly from smaller number of gates needed to add constants inside the converter.

5 Conclusion

A new general method to design reverse converters for the flexible 3-moduli $\{2^k, 2^n - 1, 2^n + 1\}$ residue number system (RNS), applicable for arbitrary pairs of positive integers n and k, was proposed. From the set of the basic functions, four versions of the converter can be designed for any n and k. Two of them, the most efficient ones, take advantage of the separation of the variable and constant parts, so that the addition of the constants can be shifted out to the residue datapath channels, thus reducing the number of added operands by one. (Elsewhere, we have already shown that the latter can be done in most cases at no cost.) The synthesis results were obtained for all dynamic ranges from 8 to 40 bits for consecutive values of n and $n + 1 \leq k \leq n + 3$ and compared against the known similar designs. In most cases, the presented versions are both faster and less hardware-consuming. The savings for the best versions (these with constants moved to the datapath channels) are up to 12.7% for the area and from 2.5% to 14% (5.8 % on average) for the delay, while the power consumption is reduced up to 23.2% (5.6% on average).

Finally, the importance of the designs proposed stems from the fact that the 3-moduli set $\{2^k, 2^n - 1, 2^n + 1\}$ can be extended to form multi-moduli special sets, and its reverse converters can be reused as building blocks to construct converters for these multi-moduli systems based on the premises of the mixed-radix conversion. Hence, the converters proposed here can contribute to improved performance of the whole family of converters already known as well as those which will be proposed in the future for some other extended flexible multi-moduli sets.

Abbreviations

ADP	Area–delay product
CRT	Chinese remainder theorem
CSA	Carry–save adder
DR	Dynamic range
DSP	Digital signal processing
EAC	End-around carry
FA	Full-adder
FIR	Finite impulse response
HA	Half-adder
LSB	Least significant bit

MAC	Multiplier–accumulator
MSB	Most significant bit
PDP	Power–delay product
PnR	Place-and-route
RNS	Residue number system
RTL	Register transfer level

List of symbols

C_1, C_2	The total constant to be added
k	The size (bits) of even modulus
l	Number of RNS channels
M	Dynamic range of RNS
m_i	The ith modulus
n	The size (bits) of odd modulus
X, Y, Z	Integers
x_i	Residue of X mod m_i

Acknowledgements
The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the article.

Author contributions
PP was responsible for conceptualization, software, data curation, visualization, investigation and writing—reviewing and editing. SJP participated in writing—original draft preparation, reviewing and editing, validation, methodology and supervision.

Funding
The research or reporting received no specific funding.

Availability of data and materials
Please contact authors for data requests.

Declarations

Consent for publication
Not applicable.

Ethics approval and consent to participate
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 9 January 2023 Accepted: 21 June 2023
Published online: 04 September 2023

References

1. P. V. Ananda Mohan, *Residue number systems: Algorithms and architectures*, Birkhäuser, Switzerland, (2016)
2. G. Chalivendra, V. Hanumaiah, and S. Vrudhula, A new balanced 4-moduli set $\{2^k, 2^n - 1, 2^n + 1, 2^{n+1} - 1\}$ and its reverse converter design for efficient FIR filter implementation, in *Proceedings of the ACMGreat Lakes SymposiumVLSI (GLSVLSI)* (Lausanne, 2011), 2–4, pp. 139–144
3. R. Chaves and L. Sousa, RDSP: A RISC DSP based on residue number system, in *Proceedings of Euromicro Symposium on Digital SystemDesign (DSD)* (Antalya, 2003), pp. 128–135
4. R. Chaves, L. Sousa, Improving residue number system multiplication with more balanced moduli sets and enhanced modular arithmetic structures. IET Proc. Comput. Digit. Tech. **1**(5), 472–480 (2007)
5. J. Chen, J. E. Stine, Parallel prefix Ling structures for modulo $2^n - 1$addition, in *Proceedings of the 20th IEEE International Conference Application-specific Systems, Architectures and Processors (ASAP'09)* (Boston, 2009), 7–9, pp. 16–23
6. R. Chokshi, K. S. Berezowski, A. Shrivastava, S. J. Piestrak, Exploiting residue number system for power-efficient digital signal processing in embedded processors, in *Proceedings of the 2009 international conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)* (Grenoble, 2009), 11–16, pp. 19–27
7. R. Conway, J. Nelson, Improved RNS FIR filter architectures. IEEE Trans. Circuits Syst. II **51**(1), 26–28 (2004)
8. A. Dhurkadas, Comments on "An efficient residue to binary converter design" by K. M. Ibrahim and S. N. Saloum, IEEE Trans. Circuits Syst. **37**, no. 6, 849–850 (1990)
9. A. Dhurkadas, Comments on "A high speed realization of a residue to binary number system converter." IEEE Trans. Circuits Syst. II **45**(3), 446–447 (1998)
10. C. Efstathiou, H.T. Vergos, D. Nikolos, Modified Booth modulo $2^n - 1$multipliers. IEEE Trans. Comput. **53**(3), 370–374 (2004)
11. H.K. Garg, *Digital signal processing algorithms: Number theory, convolution, fast fourier transforms, and applications* (CRC Press, Boca Raton, FL, USA, 1998)

12. A. Hiasat, A residue-to-binary converter with an adjustable structure for an extended RNS three-moduli set. J. Circuits Syst. Comput. **28**(8), 1–24 (2019)
13. G. Jaberipur, B. Parhami, Unified approach to the design of modulo-($2^n \pm 1$) adders based on signed-LSB representation of residues, in *Proceedings 14th IEEE Symposiumon Computer Arithmetic*. (Portland, 2009), 8–10, pp. 57–64
14. T.-B. Juang, C.-C. Chiu, M.-Y. Tsai, Improved area-efficient weighted modulo $2^n + 1$adder design with simple correction schemes. IEEE Trans. Circuits Syst. II **57**(3), 198–202 (2010)
15. K.P. Lim, A.B. Premkumar, A modular approach to the computation of convolution sum using distributed arithmetic principles. IEEE Trans. Circuits Syst. II **46**(1), 92–96 (1999)
16. Y. Liu, E.M.-K. Lai, Design and implementation of an RNS-based 2-D DWT processor. IEEE Trans. Consum. Electron. **50**(1), 376–385 (2004)
17. Y. Liu, E.M.-K. Lai, Moduli set selection and cost estimation for RNS-based FIR filter and filter bank design. Des. Aut. Embed. Syst. **9**(2), 123–139 (2004)
18. R. Muralidharan, C.H. Chang, Area-power efficient modulo $2^n - 1$and modulo $2^n + 1$multipliers for $\{2^n - 1, 2^n, 2^n + 1\}$ based RNS, IEEE Trans. Circuits Syst. I Reg. Papers **59**(10), 2263–2274 (2012)
19. R.A. Patel, S. Boussakta, Fast parallel-prefix architectures for modulo $2^n - 1$addition with a single representation of zero. IEEE Trans. Comput. **56**(11), 1484–1492 (2007)
20. P. Patronik, S.J. Piestrak, Design of reverse converters for general RNS moduli sets $\{2^k, 2^n - 1, 2^n + 1, 2^{n-1} - 1\}$ and $\{2^k, 2^n - 1, 2^n + 1, 2^{n+1} - 1\}$ (n even), IEEE Trans. Circuits Syst. I Reg. Papers **61**(6), 1687–1700 (2014)
21. P. Patronik and S. J. Piestrak, Design of residue generators with CLA/compressor trees and multi-bit EAC, in *Proceedings of the 2017 IEEE 8th Latin American Symposium on Circuits and Systems (LASCAS)* (San Carlos de Bariloche, 2017), 20–23, pp. 77–80
22. P. Patronik, S.J. Piestrak, Design of reverse converters for a new flexible RNS 5-moduli set $\{2^k, 2^n - 1, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$ (n even), Circuits. Syst. Signal Process. **36**(11), 4593–4614 (2017)
23. P. Patronik, S.J. Piestrak, Hardware/software approach to designing low-power RNS-enhanced arithmetic units, IEEE Trans. Circuits Syst. I Reg. Papers **64**(5), 1031–1039 (2017)
24. P. Patronik, S.J. Piestrak, Design of RNS reverse converters with constant shifting to residue datapath channels. J. Sign. Process. Syst. **90**(3), 323–339 (2018)
25. S.J. Piestrak, Design of residue generators and multioperand modular adders using carry-save adders. IEEE Trans. Comput. **43**(1), 68–77 (1994)
26. S.J. Piestrak, A high-speed realization of a residue to binary number system converter. IEEE Trans. Circuits Syst. II **42**(10), 661–663 (1995)
27. S.J. Piestrak, Design of multi-residue generators using shared logic, in *Proceedings of the IEEE International Symposium of Circuits and Systems (ISCAS)* (Rio de Janeiro, 2011), 15–18, pp. 1435–1438
28. S.J. Piestrak, K.S. Berezowski, Architecture of efficient RNS-based digital signal processor with very low-level pipelining, in *Proceedings of the IET Irish Signals and Systems Conference (ISSC)* (Galway, 2008), 18–19 , pp. 127–132
29. S.J. Piestrak, K.S. Berezowski, Design of residue multipliers-accumulators using periodicity, in *IET Irish Signals and Systems Conference (ISSC)* (Galway, 2008), 18–19, pp. 380–385
30. J. Ramírez, A. García, U. Meyer-Bäse, F. Taylor, A. Lloris, Implementation of RNS-based distributed arithmetic discrete wavelet transform architectures using field-programmable logic. J. VLSI Signal Process. **33**(1–2), 171–190 (2003)
31. T.K. Shahana, R.K. James, B.R. Jose, K.P. Jacob, S.Sasi, Performance analysis of FIR digital filter design: RNS versus traditional, in *Proceedings of the International Symposium on Communications and Information Technologies (ISCIT)* (Sydney, 2007), 17–19, pp. 1–5
32. A. Sweidan, A. Hiasat, A new efficient memoryless residue to binary converter. IEEE Trans. Circuits Syst. **35**(11), 1441–1444 (1988)
33. F.J. Taylor, An RNS discrete Fourier transform implementation, IEEE Trans. Acoust., Speech Signal Process. **38**, 8, 1386–1394 (1990)
34. W.Wang, M.N.S. Swamy, M.O. Ahmad, RNS application for digital image processing, *Proceedings of the IEEE international workshop on system-on-chip for real-time applications*, (Banff, Alberta, 2004), 19–21, pp. 77–80
35. Y. Wang, Residue-to-binary converters based on new Chinese Remainder Theorem. IEEE Trans. Circuits Syst. II **47**(3), 197–205 (2000)
36. Y. Wang, X. Song, M. Aboulhamid, H. Shen, Adder based residue to binary number converters for $(2^n - 1, 2^n, 2^n + 1)$. IEEE Trans. Signal Process. **50**(7), 1772–1779 (2002)
37. Z. Wang, G.A. Jullien, W.C. Miller, An improved residue-to-binary converter. IEEE Trans. Circuits Syst. I **47**(9), 1437–1440 (2000)
38. M.Wesołowski, P.Patronik, K.Berezowski, J. Biernat, Design of a novel flexible 4-moduli RNS and reverse converter, Proc. 23nd IET Irish Sign. Syst. Conf. (ISSC) (Maynooth, 2012), 28–29 , pp. 1–6
39. R. Zimmerman, Efficient VLSI implementation of modulo $2^n \pm 1$addition and multiplication, in Proceedings of the IEEE Symposium on Computer Arithmetic, (Adelaide, 1999), 14–16, pp. 158–167

Publisher's Note