J Sign Process Syst (2018) 90:323–339
DOI 10.1007/s11265-017-1238-6

Design of RNS Reverse Converters with Constant Shifting to Residue Datapath Channels

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

Received: 8 December 2015 / Revised: 24 November 2016 / Accepted: 1 March 2017 /Published online: 14 March 2017

Abstract This paper presents a new general approach to simplify residue-to-binary (reverse) converters for a Residue Number System (RNS) composed of an arbitrary set of moduli. It is suggested to formulate the basic equation of the reverse converter in a form consisting of two separate parts: one depending on input variables of the converter whereas the other is a single constant. Then, the constant, instead of being added inside the reverse converter, can be shifted out to the residue datapath channels, in most cases at no hardware cost or extra delay. Thus, the hardware cost of the converter is reduced, because its multi-operand adder has one operand less to handle. To illustrate various design issues of this new design approach and to prove its efficiency, a new design method of the residue-to-binary (reverse) converters for the 3-moduli set $\{2^n-1, 2^n, 2^n+1\}$ is considered. Two versions of the new converters for the 3-moduli set $\{2^n-1, 2^n, 2^n+1\}$ as well as several of their known counterparts were synthesized for all dynamic ranges from 8 to 38 bits (i.e., for $3 \leq n \leq 13$). The results obtained suggest that, compared to the best of the state-of-the-art converters, at least one of two versions of our converters is superior with respect to area and power consumption, for all dynamic ranges considered, in some cases accompanied by slight delay reduction. The area is reduced from about 5 % to about 20 % and the largest savings are observed for the power consumption—from over 10 % up to 27 %.

Keywords Residue number system (RNS) · Reverse converter · Residue-to-binary converter · Computer arithmetic · Digital signal processing (DSP)

✉ Piotr Patronik
piotr.patronik@pwr.wroc.pl

Stanisław J. Piestrak
stanislaw.piestrak@univ-lorraine.fr

[1] Department of Computer Engineering (W-4/K-9), Wrocław University of Technology, 50–370 Wrocław, Poland

[2] Res. Team MAE, Institut Jean Lamour (UMR 7198 CNRS), Université de Lorraine, 54506 Vandœuvre-Les-Nancy, France

1 Introduction

The Residue Number System (RNS) offers several well documented advantages over the conventional 2's complement binary number system [16]. One of them is that the basic arithmetic operations like addition, subtraction, and multiplication can be carried out simultaneously in a number of parallel independent datapaths on relatively short numbers. It is therefore particularly well suited for hardware implementation of a typical computational problem in many Digital Signal Processing (DSP) systems, as the calculation of the vector inner product (the sum of products)

$$S_N = \sum_{j=0}^{N-1} C_j \cdot X_j, \tag{1}$$

where: S_N is the numerical value of the function computed, X_j is the j-th of the series of N input operands, and C_j is the j-th of the series of N a priori known coefficients (which could be loaded at the system initialization). The RNS representation allows to execute inner product computations like those given in Eq. 1 using virtually carry-free arithmetic allowing for area, time, and power consumption savings compared to its 2's complement positional counterpart.

Most digital systems operate on data using a positional representation of numbers, hence using a non-positional RNS representation of numbers in some computational blocks requires *conversions* of the numbers back and forth to RNS, performed respectively by *reverse* and *forward converters*. Because, unlike residue datapaths executing useful computations, both converters are a pure overhead, it is desirable to maximally reduce their area, delay, and power consumption. The forward conversion is conceptually relatively simple, because it is an extraction of residues from a positional number, which can be implemented in hardware using residue generators [21, 23]. On the other hand, the reverse conversion requires application of special methods, amongst which the most commonly used are the *Chinese Remainder Theorem (CRT)* and the *Mixed-Radix Conversion (MRC)* [16], whose efficient hardware implementations are significantly more difficult to design than forward conversion.

The main goal of this paper is to propose a new approach which could be taken into account to improve all characteristics of the reverse converters for an arbitrary set of moduli. It relies on the hypothesis that a reverse converter equation is given in the form in which the variable terms and a constant are separated. Although the mathematical expression of such an equation can be easily obtained, the main difficulty relies on its such a formulation that it can be efficiently implemented in hardware. Then, once such an equation has been found, the addition of the constant can be shifted from the reverse converter to the residue datapath channels, thus reducing by one the number of operands handled by the reverse converter. We will show that the latter can result in some area, delay, and power consumption savings, which can be achieved virtually at no cost. The above idea was already suggested by us for the first time in [19], but only for a particular case of reverse converters for two 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). Also, in [19], neither any discussion on the possible performance degradation of residue datapaths nor the feasibility of applying this approach to other RNS moduli sets have been presented.

The choice of the moduli set of an RNS significantly affects performance of residue datapaths. Particularly efficient implementations of all basic modulo arithmetic operations and binary-to-residue (forward) converters have powers-of-two related moduli of the forms 2^n and $2^n \pm 1$ which hence are considered low-cost. Consequently, in search for efficient RNS-based hardware implementations, several special moduli sets composed exclusively of low-cost moduli have been proposed. The most intensively investigated has been the 3-moduli set $\{2^n-1, 2^n, 2^n+1\}$ introduced in 1978 by Jenkins [14], which offers the $(3n-1)$-bit dynamic range with 3-bit resolution: e.g., for $n = 6$, 7, and 8, the dynamic ranges available are respectively 17, 20, and 23 bits. Throughout the years, several reverse converters with steadily improved parameters have been proposed for this RNS [2–5, 9, 11–13, 22, 31, 32, 34, 35]. Amongst them, the most hardware efficient and high-speed converters can be designed using the methods from [12, 34, 35]. To note also that the reverse converter for this moduli set can be also obtained using recently proposed design methods of the converter for the general 3-moduli set $\{2^n-1, 2^k, 2^n+1\}$ with flexible even modulus k [8, 36], applied for the special case of $k = n$. Some specific applications of the 3-moduli set $\{2^n-1, 2^n, 2^n+1\}$ include Finite Input Response (FIR) filters [10, 15, 24, 28]. However, introducing in 2007 the flexible 3-moduli set $\{2^n-1, 2^k, 2^n+1\}$, accompanied by an efficient reverse converter for $k \le 2n$ [8], has superseded the 3-moduli set with only a single parameter n. This is because for the dynamic range of $3n-1$ bits offered by the former, the equivalent 3-moduli set $\{2^{n-1}-1, 2^{n+1}, 2^{n-1}+1\}$ results in faster and more area-efficient residue datapaths. Although, no specific designs have been explicitly considered in the literature, the overall complexity figures of the latter can be easily obtained from complexity characteristics of the MACs mod $2^n \pm 1$ and 2^k for all the moduli sets concerned, provided in [10, 15, 25].

As a vehicle to present various facets of constant shifting to the residue datapath channels, to illustrate its feasibility, and to show its efficiency, we have chosen designing a new reverse converter for the 3-moduli RNS $\{2^n-1, 2^n, 2^n+1\}$. (It is worth to note that despite several reverse converter functions for this moduli set have been proposed to date, none of those referred earlier has been readily amenable for such a transformation.) Obviously, in the context of the state-of-the-art presented above, building another reverse converter for the 3-moduli set $\{2^n, 2^n-1, 2^n+1\}$, even with slightly improved performance, could seem hardly justifiable. Nevertheless, of significantly more interest is the possibility of using it not as a stand-alone circuit but rather as the main building block in various reverse converters for larger multi-moduli RNSs formed by extending the basic 3-moduli set. The design of the latter architectures is based on the premises of the MRC algorithm. The 3-moduli set $\{2^n, 2^n-1, 2^n+1\}$ is one of the most commonly used for such an approach. Examples of the latter are reverse converters proposed recently for the special balanced 4-moduli sets $\{2^n, 2^n-1, 2^n+1, 2^{n+1}-1\}$ (n even) [1, 7], $\{2^n, 2^n-1, 2^n+1, 2^{n-1}-1\}$ (n even) [7], $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$ (n odd) [1, 30], and $\{2^n+1, 2^n-1, 2^n, 2^{n-1}+1\}$ (n odd) [20], 5-moduli sets $\{2^{n+1}, 2^n-1, 2^n+1, 2^n+2^{(n+1)/2}+1, 2^n-2^{(n+1)/2}+1\}$ (n odd) [26], $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1, 2^{n-1}-1\}$ (n even) [6], $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1, 2^{n-1}+1\}$ (n odd) [18], as well as many more like e.g., those mentioned in [27]. Hence, the reverse converter for the above 3-moduli set is one of the most crucial parts, whose characteristics significantly affect performance of the whole family of converters.

This paper is organized as follows. In Section 2, the theoretical background on RNS and the basic properties of arithmetic mod $2^n - 1$ are presented. In Section 3, the general problem of designing a reverse converter for an arbitrary RNS moduli set, whose basic equation allows for shifting a constant to residue datapath channels, as well as some design suggestions are presented. In Section 4, the concept of shifting a constant to residue datapath channels is illustrated on the example of a new design method of the reverse converters for the 3-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, whose two versions are presented. In Sections 5 and 6, the complexity evaluation of the reverse converters for the 3-moduli set, those proposed here and their most efficient existing counterparts, is presented: it includes both the gate level estimations of complexity figures as well as more accurate evaluations of the delay, area, and power efficiency of all circuits synthesized in 65 nm technology. Finally, Section 7 presents some conclusions and suggestions for future research.

2 RNS Background

2.1 Basic Notions

Let X be an integer and m be a positive integer called a *modulus*. We define X *residue modulo (mod)* m as a result of the integer division of X by m, denoted $x = X \mod m$ or $x = |X|_m$, where usually $0 \le |X|_m \le m-1$ (one notable exception is widely accepted double representation of zero mod $2^n - 1$, which has two equivalent representations of n 0's $(0\ldots00)$ and n 1's $(1\ldots11)$, because in this case $0 \le |X|_{2^n-1} \le 2^n - 1)$. An RNS is a set of numbers defined by a set of r pairwise prime *moduli* $\{m_1, \ldots, m_r\}$. The *dynamic range* M of an RNS is a product of its moduli, i.e., $M = \prod_{i=1}^{r} m_i$, hence, a positional binary representation of any RNS number occupies $a = \lceil \log_2 M \rceil$ bits. Any number $0 \le X < M$ in an RNS can be represented by an ordered set of residues $X = \{x_1, \ldots, x_r\}$, where $x_i = |X|_{m_i}$, $1 \le i \le r$ is represented on $a_i = \lceil \log_2 m_i \rceil$ bits.

Let X, Y, and Z be integers $0 \le X, Y, Z < M$ represented by the sets of residues as above. Then, any arithmetic operation $\circ \in \{+, -, \times\}$ on these integers yielding $Z = |X \circ Y|_M$ is equivalent to the same arithmetic operation executed on their independent residues, i.e., $z_i = |x_i \circ y_i|_{m_i}$. These operations executed on relatively short operands are the main advantage of RNS representation over its 2's complement positional integer counterpart.

A *multiplicative inverse* of g mod m $(0 < g < m)$ is such an integer h $(0 < h < m)$ that $|hg|_m = 1$. It exists provided that g and m are co-prime. It can be written using the fractional notation as $h = |1/g|_m$. Here, two following multiplicative inverses will be useful: (i) $|1/2^n|_{2^{2n}-1} = 2^n$, and (ii) $|1/(2^n + 1)|_{2^n-1} = |1/(2^n - 1)|_{2^n+1} = 2^{n-1}$.

Reverse converters, which are used to convert from the RNS to the positional representation of numbers, can be designed using the following general methods [16, 33].

– The *Chinese Remainder Theorem (CRT)*:

$$X = \left| \sum_{i=1}^{r} \hat{m}_i x_i \left| \frac{1}{\hat{m}_i} \right|_{m_i} \right|_M, \tag{2}$$

where $\hat{m}_i = \prod_{j=1, j\ne i}^{r} m_j$, i.e., $\hat{m}_i$ is a product of all moduli but m_i.

– The *New CRT*:

$$\begin{aligned} X &= |x_1 + k_1(x_2 - x_1)m_1 + k_2(x_3 - x_2)m_1 m_2 \\ &\quad + \cdots + k_{r-1}(x_r - x_{r-1})m_1 \cdots m_{r-1}|_M \end{aligned} \tag{3}$$

where $k_i = \left| \frac{1}{\prod_{j=1}^{i} m_j} \right|_{\prod_{j=i+1}^{r} m_j}$ with $1 \le i \le r - 1$.

– The *Mixed-Radix Conversion (MRC)*, which is an iterative method given by

$$X = d_1 + d_2 m_1 + d_3 m_2 m_1 + \cdots + d_r \prod_{i=1}^{r-1} m_i, \tag{4}$$

where the *mixed-radix digits* r-tuple $\{d_1, d_2, \ldots, d_r\}$ is defined by

$$\begin{aligned} d_1 &= X_1 \\ d_2 &= \left| (X_2 - d_1) \cdot \left| m_1^{-1} \right|_{m_2} \right|_{m_2}, \cdots \\ d_i &= \left| \left(\cdots \left((X_i - d_1) \left| m_1^{-1} \right|_{m_i} - d_2 \right) \left| m_2^{-1} \right|_{m_i} \right) \right. \\ &\quad \left. - \cdots - d_{r-1} \left| m_{r-1}^{-1} \right|_{m_i} \right|_{m_i}, 3 \le i \le r. \end{aligned}$$

For the special case of the 2-moduli set $\{m_1, m_2\}$ and two respective residues $\{x_1, x_2\}$, which will be needed in Section 4, the simplified version of Eq. 4 is

$$X = x_1 + m_1 \left| (x_2 - x_1) \frac{1}{m\,1} \right|_{m_2}. \tag{5}$$

2.2 Properties of Arithmetic Modulo $2^n - 1$

For Readers' convenience, all properties needed in the case study given in Section 4 are presented here. Let n, s, and d be arbitrary positive integers, and z, x, y be positive integers such that $0 \le z, x \le 2^n - 1$ and $0 \le y \le 2^{sn} - 1$. The binary representations of z, x, and y are respectively $(z_{n-1} \ldots z_0)$, $(x_{n-1} \ldots x_0)$, and $(y_{sn-1} \ldots y_0)$. As for y, if it is more than $(s - 1)n$-bit and less than sn-bit number, then it is preceded by some leading zeros on the most significant bit positions. The symbol $\|$ denotes the concatenation of binary vectors. Then, some basic arithmetic operations mod $2^n - 1$ are performed as follows.

– The sign change of z mod $2^n - 1$ (the additive inverse of z) is obtained by bit-by-bit complementing of all bits, i.e.,

$$|-z|_{2^n-1} = |\bar{z}|_{2^n-1} = |(\bar{z}_{n-1}\ldots\bar{z}_0)|_{2^n-1}.$$

– The multiplication mod $2^n - 1$ of z by 2^d is obtained by the left cyclic shift of z by d positions, i.e., $|2^d z|_{2^n-1} = |(z_{n-d-1}\ldots z_0 z_{n-1}\ldots z_{n-d})|_{2^n-1}$, and easily implemented at no hardware cost.

– The multiplication mod $2^n - 1$ of z by $2^{-d} = 1/2^d$ (i.e., the multiplication mod $2^n - 1$ by the multiplicative inverse of 2^d) is obtained by the right cyclic shift of z by d positions (equivalent to the left cyclic shift of z by $n - d$ positions, because $|2^{-d} z|_{2^n-1} = |2^{n-d} z|_{2^n-1}$), i.e.

$$|2^{-d} z|_{2^n-1} = |z/2^d|_{2^n-1} = |(z_{d-1}\ldots z_0 z_{n-1}\ldots z_d)|_{2^n-1}.$$

– For any non-negative integer i, $|2^{in}|_{2^n-1} = 1$. Therefore, for an (sn)-bit number y we have

$$\begin{aligned}|y|_{2^n-1} &= \left|\sum_{i=0}^{s-1} 2^{ni}(y_{n(i+1)-1}\ldots y_{ni})\right|_{2^n-1}\\ &= \left|\sum_{i=0}^{s-1}(y_{n(i+1)-1}\ldots y_{ni})\right|_{2^n-1},\end{aligned}$$

i.e., to obtain the residue y mod $2^n - 1$, it simply suffices to partition y into s n-bit parts and sum up all of them mod $2^n - 1$.

3 General Scheme of Constant Shifting for an Arbitrary RNS Moduli Set

In this section, we will present the general approach which can be used to simplify reverse converters for an arbitrary RNS moduli set, which relies on shifting constants from a reverse converter to residue datapath channels. Specifically, any RNS reverse converter can be seen as a particular modular datapath composed of a number of constant multiplications, bit-level manipulations, and finally additions. Because these arithmetic and logic operations may involve some variable or fixed operands, whose number and length determine the amount of required hardware and performance, any cost reduction boils down to the reduction of the number of executed operations and specifically a number of addition operands. Our goal is to consider the possibility of transforming the functions of the reverse converter to such a form that all constants are accumulated to a single one, which then could be shifted out to the residue datapath, thus reducing the number of operands handled inside the reverse converter. As each reverse converter for a given moduli set is a kind of custom designed datapath, isolating the constants can be done only on a case-by-case basis, depending on the moduli set and the architecture of the converter. An inspiration to present such a general framework for an arbitrary RNS reverse converter was our earlier results obtained for the reverse converters for two 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) [19].

The inspection of the basic formulas for the reverse conversion given by Eqs. 2–4 reveals that the positional representation of the output variable X could only be the sum mod M (for the CRT and the New CRT) or the simple sum (for MRC) of products of only one residue x_i by some constant b_i ($1 \le i \le r$), i.e., there never occurs any single term involving the products of two residues like $x_i x_j$. In particular, any CRT-based reverse converter function can be expressed in a generic form as

$$X = \left|\left(\sum_{i=1}^{r} b_i x_i\right) + C_k\right|_M, \tag{6}$$

where b_i are some constant coefficients of the residues x_i and C_k is some total constant.

It is important to recall that, because it is implicitly assumed that the RNS considered is a non-redundant one, each of r moduli contributes to the dynamic range $M = \prod_{i=1}^{r} m_i$. Therefore, X must depend on all coefficients b_i, which implies that all $b_i \neq 0$ and the final addition has at least r operands. The actual number of operands of the latter, $p \ge r$, depends on the possibility of obtaining simple binary expressions for $|b_i x_i|_M$ or $b_i x_i$ (for MRC). Although either of the latter products can be implemented using ROM look-up tables, on one hand, it could be prohibitively complex for larger a_i, and on the other hand, for many moduli sets with arbitrary a_i the most efficient implementations have been obtained using arithmetic circuits (adders and subtractors) and some bit manipulations.

Obviously, a simple arithmetic expression like Eq. 6 does not necessarily imply that any simple hardware implementation with the smallest possible number of operands can be obtained even through skillful bit manipulations. Nevertheless, the motivation for reducing the number of operands stems from the fact that it can contribute not only to reducing area and power consumption (which seems quite obvious: eliminating one operand results in one up to a-bit CSA less, i.e., up to a FAs or HAs less, depending on whether an operand is a variable or a constant), but even in some cases it can be accompanied by reducing by one the number of CSA stages on a CSA tree that processes multiple operands to obtain the final result X.

Now consider the RNS-based implementation of Eq. 1, which takes the form

$$|S_N|_{m_i} = \left|\sum_{j=0}^{N-1}\left(\left||C_j|_{m_i}\cdot|X_j|_{m_i}\right|_{m_i}\right)\right|_{m_i}, 1 \le i \le r. \quad (7)$$

We assume that Eq. 7 is implemented using N Multiply-Accumulate units (MACs), in which the j-th stage of the datapath ($1 \le j \le N$) is composed of r MACs mod m_i, $1 \le i \le r$, each of which realizes the function

$$\left|S_j\right|_{m_i} = \left||S_{j-1}|_{m_i} + |C_{j-1}|_{m_i}\cdot|X_{j-1}|_{m_i}\right|_{m_i}, \quad (8)$$

where the initial values are generally assumed $|S_0|_{m_i} = 0$.

If the function of the reverse converter can be expressed as in Eq. 6, then, the residue datapath channels followed by the reverse converter can be implemented as shown in the upper part of Fig. 1. Now observe that the addition of the constant $|C_k|_M$ in the reverse converter is equivalent to the set of additions of r constants $|c_i|_{m_i}$ in the residue datapaths, $1 \le i \le r$, where $|c_i|_{m_i} = |C_k|_{m_i}$. Therefore, the addition of the constant $|C_k|_M$ can be shifted out from the $(p+1)$-operand CSA tree mod M of the reverse converter, because it can be taken into account earlier at the stage of RNS datapath computations modulo all moduli m_i, resulting in hardware implementation shown in the lower part of Fig. 1. The case study presented in the following sections will show that in most cases such a modification can be done at no hardware cost.

To facilitate finding the reverse converter function amenable for shifting out constants to residue datapaths, we suggest to proceed as follows.

– Identify all modular datapaths inside the reverse converter.

  The reverse converter is a network of interconnected modular datapaths, whose architecture depends both on the moduli set and the general approach taken by a designer of the converter, resulting in a composition of CRT, New CRT, and MRC techniques. As such, each part of a converter is a chain of additions and multiplications of the residues x_i by a constant. Hence, the first step is to identify such parts in the converter.
– Determine the value of constants added and multiplied inside these datapaths and calculate the cumulative

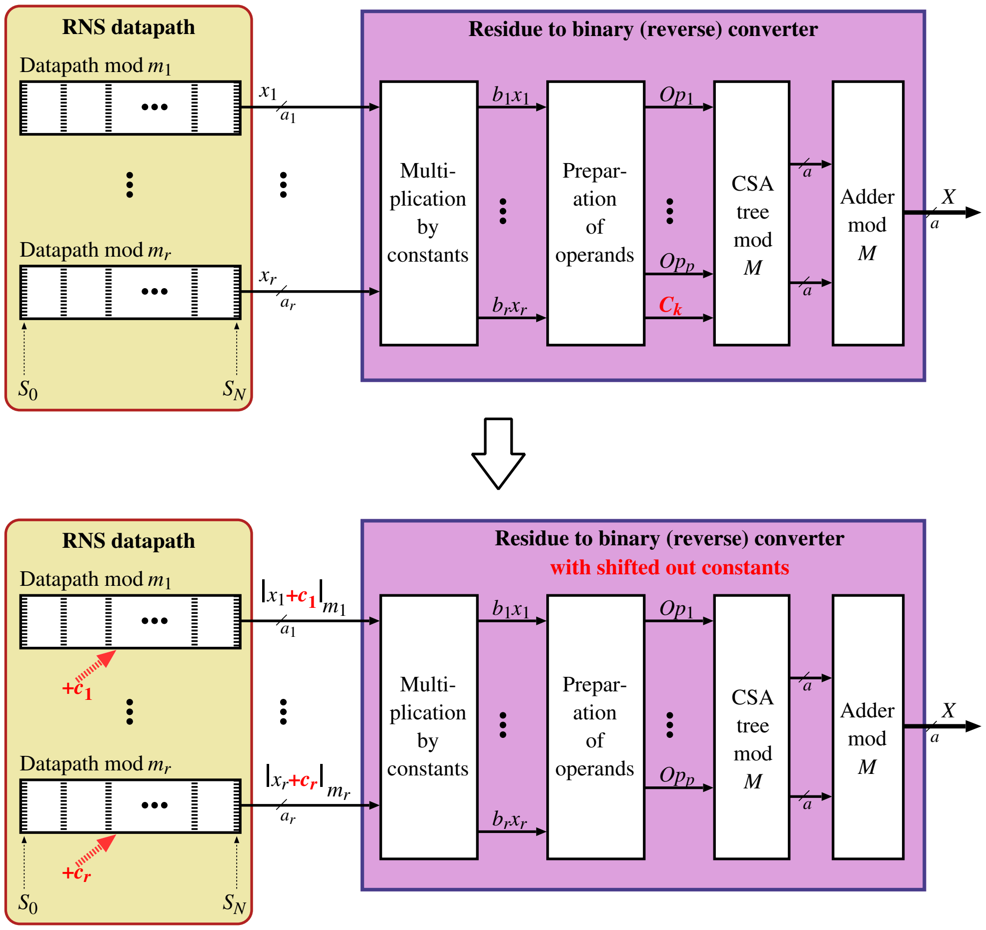


Figure 1 General scheme of shifting constants from the reverse converter for an arbitrary RNS moduli set to residue datapath channels, according to Eq. 6.

value of the constant for each of the residue datapaths $m_i, 1 \leq i \leq r$.

The most common case of a constant addition is when numbers are subtracted, following the identity $-x_i = \bar{x}_i - 1$. Alternatively, the addition mod m_i of a negative number is expressed as $-x_i = m_i - x_i$. Note also that each residue datapath channel could have its own cumulative constant.

– Calculate the constants for each channel.

Each datapath has its own modulus m_i, which can be either the individual modulus from the moduli set (e.g. as in Eq. 6) or the product of the moduli—if the hierarchical structure of the converter is used (one example can be found in [19]). In the latter case, the value of the constant for each respective channel is the residue of the constant for the datapath; otherwise, the constant is added only in one channel. Because various datapaths inside a converter may result in various constants, in such a case, the constants are accumulated in channels modulo their respective moduli and added only once.

– Reformulate and simplify the functions of the converter with shifted out constants, including reorganization of modular CSAs and new alternative bit-level manipulations.

With the constants eliminated from a given datapath of the reverse converter, ones can be replaced with zeros on all relevant bits. The latter is done by replacing FAs in one stage of the CSA tree with HAs or even by completely eliminating one stage of them. Moreover, if some bit combinations in datapaths do not occur, FAs can be replaced with simpler AND/OR logic operators.

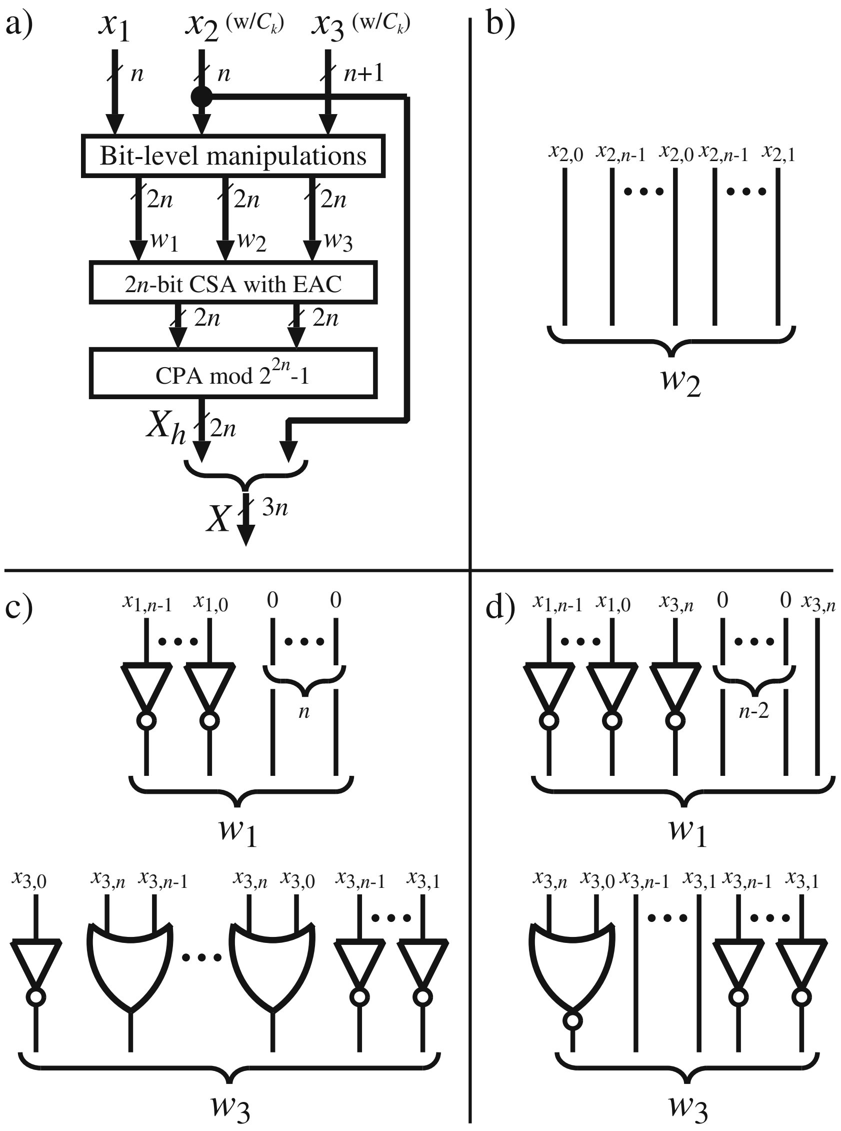

a)
x_1
x_2 (w/C_k)
x_3 (w/C_k)
n
n
$n+1$
Bit-level manipulations
$2n$
w_1
w_2
w_3
$2n$-bit CSA with EAC
CPA mod 2^{2n}-1
X_h
X
$3n$
b)
$x_{2,0}$ $x_{2,n-1}$ $x_{2,0}$ $x_{2,n-1}$ $x_{2,1}$
w_2
c)
$x_{1,n-1}$ $x_{1,0}$ 0 0
n
w_1
$x_{3,0}$ $x_{3,n}$ $x_{3,n-1}$ $x_{3,n}$ $x_{3,0}$ $x_{3,n-1}$ $x_{3,1}$
w_3
d)
$x_{1,n-1}$ $x_{1,0}$ $x_{3,n}$ 0 0 $x_{3,n}$
n-2
w_1
$x_{3,n}$ $x_{3,0}$ $x_{3,n-1}$ $x_{3,1}$ $x_{3,n-1}$ $x_{3,1}$
w_3

Figure 2 Block diagram of the new converter for the 3-moduli set $\{2^n-1, 2^n, 2^n+1\}$: **a** general schema; **b** bit-level manipulations for variable w_2; **c**, **d** bit-level manipulations for variables w_1 and w_3 respectively for Version 1 and 2.

In the next section, we will show how to apply these general steps to the special case of the new reverse converter for the 3-moduli set $\{2^n-1, 2^n, 2^n+1\}$.

4 Design of Reverse Converters for the 3-Moduli Set $\{2^n-1, 2^n, 2^n+1\}$

In this section, we will first detail a new method for designing reverse converters that implements a newly introduced set of equations in which variable and constant terms could be separated. Then, we will explain a new approach to improving converter performance that relies on shifting constants added by the converter out to the datapath channels and will argue that the latter operation in most cases can be done at no cost. A general logic scheme of the new converter is presented in Fig. 2.

4.1 New Basic Functions

The basic functions of the reverse converter for the set of three moduli $\{m_1, m_2, m_3\} = \{2^n, 2^n-1, 2^n+1\}$ have been given in many papers, notably in [12, 34, 35] which proposed designs currently considered the most efficient. The latter three methods start with the CRT (or the so-called New CRT in case of [34]) and then bring the problem of the conversion to a Multi-Operand Mod $2^{2n}-1$ Addition (MOMA mod $2^{2n}-1$). Because the final addition mod $2^{2n}-1$ seems inevitable in this RNS, all the efforts to increase the efficiency of the converter have been concentrated on the problem of reducing the number of MOMA operands. There are three operands in the case of [12, 35] and four in the case of [8, 34, 36] (although in [34], the fourth operand is reduced to one bit). The varying composition of MOMA operands is mainly achieved by various bit-level manipulations, with the exception of the converter of [36]. In all previous designs, the MOMA operands are composed of variable parts depending on three input residues x_1, x_2, and x_3 mixed up with some constants. We have analyzed equations of all of the above converters and we have not found any obvious method to separate them. Here, our idea is to show that could a constant be isolated from variable parts, it would be unnecessary to add that constant within the converter. Therefore, we propose first to accumulate the constants to a single total constant and then to move its addition out of the converter to the residue datapath channels. It will be seen that this approach gives new options of bit-level manipulations being either a simplified version of the existing design from [35] or a new proposition which has not yet been explored in the open literature. In Section 4.4.2, we will argue that the addition of the constants in the residue datapaths in most cases can be done at no cost and that in a few special cases its cost is negligible in comparison to the benefits resulting from the simplification of the converter. Our new design method requires execution of the following two steps that are eventually merged in a single arithmetic block.

The first step relies on applying the CRT of Eq. 2 to the 2-moduli set $\{m_2, m_3\} = \{2^n-1, 2^n+1\}$ which provides $X_{23} = \{x_2, x_3\}$ $(0 \le X_{23} < m_2 m_3)$:

$$\begin{aligned} X_{23} &= \left| m_3 x_2 \left| \frac{1}{m_3} \right|_{m_2} + m_2 x_3 \left| \frac{1}{m_2} \right|_{m_3} \right|_{m_2 m_3} \\ &= \left| m_3 x_2 \left| \frac{1}{2^n+1} \right|_{2^n-1} + m_2 x_3 \left| \frac{1}{2^n-1} \right|_{2^n+1} \right|_{m_2 m_3} \\ &= \left| 2^{n-1}(2^n+1)x_2 + 2^{n-1}(2^n-1)x_3 \right|_{2^{2n}-1} \\ &= \left| 2^{n-1}\left(\left(2^n+1\right) x_2 + \left(2^n-1\right) x_3 \right) \right|_{2^{2n}-1}. \end{aligned} \tag{9}$$

The second step relies on applying the MRC of Eq. 5 (similarly as in [29]) to the 2-moduli set $\{m_1, m_2 m_3\}$, which provides the final result $X = \{x_1, X_{23}\}$ given by

$$\begin{aligned} X &= x_1 + m_1 \left| (X_{23} - x_1) \frac{1}{m_1} \right|_{m_2 m_3} \\ &= x_1 + 2^n \underbrace{\left| (X_{23} - x_1) 2^{-n} \right|_{2^{2n}-1}}_{X_h} = x_1 + 2^n X_h, \end{aligned} \tag{10}$$

where the term X_h can be computed by replacing X_{23} with its expression of Eq. 9 resulting in

$$\begin{aligned} X_h &= \left| (X_{23} - x_1)\, 2^{-n} \right|_{2^{2n}-1} \\ &= \left| \left(2^{n-1}\left(\left(2^n+1\right) x_2 + \left(2^n-1\right) x_3 \right) - x_1 \right) 2^{-n} \right|_{2^{2n}-1} \\ &= \left| -x_1 2^{-n} + x_2 \left(2^n+1\right) 2^{-1} + x_3 \left(2^n-1\right) 2^{-1} \right|_{2^{2n}-1}. \end{aligned} \tag{11}$$

Three terms which appear in the above equation will be transformed through some bit-level manipulations to obtain expressions not only better adapted for implementations using logic gates, but also having the advantage that variable parts (i.e., those depending on three input residues $\{x_1, x_2, x_3\}$) will be separated from constant components.

For the first term, we obtain the equation

$$\begin{aligned} \left| -x_1 2^{-n} \right|_{2^{2n}-1} &= \left| \overbrace{(\underbrace{1\ldots1}_{n} \| \bar{x}_1)}^{2n} 2^{-n} \right|_{2^{2n}-1} \\ &= \left| \overbrace{(0\ldots0 \| \bar{x}_1)}^{2n} 2^{-n} + (1-2^n) 2^{-n} \right|_{2^{2n}-1} \\ &= \left| (2^{n+1} \bar{x}_1) 2^{-1} + (2^{-n} - 1) \right|_{2^{2n}-1} \end{aligned} \tag{12}$$

which contains the variable term depending on x_1 and the constant term equal to $|2^{-n}-1|_{2^{2n}-1}$. The former term can be presented conveniently as

$$\begin{aligned}&\left|2^{-1}(2^{n+1}\bar{x}_1)\right|_{2^{2n}-1}\\&=\left|2^{-1}((\bar{x}_{1,n-2}\ldots\bar{x}_{1,0})\|\underbrace{(0\ldots0)}_{n}\|\bar{x}_{1,n-1})\right|_{2^{2n}-1}.\end{aligned}\tag{13}$$

Note: because the multiplication by 2^{-1} will appear in modified expressions for all three variable terms, we have left it intentionally instead of replacing it with the right cyclic shift by one bit.

In the second term, because $x_2(2^n+1)=2^nx_2+x_2$ and x_2 is an n-bit number, the addition of 2^nx_2 to x_2 is a simple concatenation of two variables x_2, which yields

$$\left|x_2\left(2^n+1\right)2^{-1}\right|_{2^{2n}-1}=\left|2^{-1}\left(x_2\|x_2\right)\right|_{2^{2n}-1}.\tag{14}$$

Finally, the third term can be rewritten as

$$\begin{aligned}&\left|x_3\left(2^n-1\right)2^{-1}\right|_{2^{2n}-1}=\left|2^{-1}\left(2^nx_3-x_3\right)\right|_{2^{2n}-1}\\&=\left|2^{-1}\begin{pmatrix}((x_{3,n-1}\ldots x_{3,0})\|\underbrace{(0\ldots0)}_{n-1}\|x_{3,n})\\+(\underbrace{(1\ldots1)}_{n-1}\|(\bar{x}_{3,n}\ldots\bar{x}_{3,0}))\end{pmatrix}\right|_{2^{2n}-1}\\&=\left|2^{-1}\begin{pmatrix}((x_{3,n-1}\ldots x_{3,0})\|(\bar{x}_{3,n-1}\ldots\bar{x}_{3,0}))\\+(\underbrace{(1\ldots1)}_{n-1}\|(\bar{x}_{3,n}\|\underbrace{0\ldots0}_{n-1})\|x_{3,n})\end{pmatrix}\right|_{2^{2n}-1}\\&=\left|2^{-1}\begin{pmatrix}((x_{3,n-1}\ldots x_{3,0})\|(\bar{x}_{3,n-1}\ldots\bar{x}_{3,0}))\\+(\underbrace{(0\ldots0)}_{n-1}\|\bar{x}_{3,n}\|\underbrace{(0\ldots0)}_{n-1}\|x_{3,n})\\+(1-2^{n+1})\end{pmatrix}\right|_{2^{2n}-1}.\end{aligned}\tag{15}$$

Now, consider the variable part of Eq. 15 involving x_3, denoted by A:

$$\begin{aligned}A&=\left(x_{3,n-1}\ldots x_{3,0}\|\bar{x}_{3,n-1}\ldots\bar{x}_{3,0}\right)\\&\quad+(\underbrace{0\ldots0}_{n-1}\|\bar{x}_{3,n}\|\underbrace{0\ldots0}_{n-1}\|x_{3,n}).\end{aligned}\tag{16}$$

First, note that if $x_3<2^n$ then $x_{3,n}=0$, so that

$$\begin{aligned}A&=\left((x_{3,n-1}\ldots x_{3,0})\|(\bar{x}_{3,n-1}\ldots\bar{x}_{3,0})\right)\\&\quad+\overbrace{(\underbrace{(0\ldots0)}_{n-1}\|1\|\underbrace{(0\ldots0)}_{n})}^{2^n}.\end{aligned}\tag{17}$$

Otherwise, if $x_{3,n}=1$, then $x_{3,i}=0$ for $0\le i\le n-1$, so that

$$\begin{aligned}A&=\left((0\ldots0)\|\underbrace{(1\ldots1)}_{n}\right)+(\underbrace{(0\ldots0}_{2n-1}\|1))\\&=\overbrace{(\underbrace{(0\ldots0)}_{n-1}\|1\|\underbrace{(0\ldots0)}_{n})}^{2^n}.\end{aligned}\tag{18}$$

Consequently, we consider two special cases depending on the value of $x_{3,n}$. First, note that if $x_{3,n}=1$, then all other bits of x_3 are equal to 0. We will try to explore this fact either to reduce the size of A from two vectors in Eq. 16 to one vector, or to change the distribution of the bits of x_3 in such a way that it would enable merging it with the other operands, e.g., with the vector of Eq. 13. We have found two expressions for variable A of which the first is inspired by [35], while the second has not been explored in the open literature yet:

1. Addition of the constant 2^n and selective setting of n most-significant bits of the first vector as in Eqs. 17 and 18 yields

$$A=\left((x_{3,n-1}\vee x_{3,n})\ldots(x_{3,0}\vee x_{3,n})\|\bar{x}_{3,n-1}\ldots\bar{x}_{3,0}\right)+2^n.\tag{19}$$

2. Merging of two cases from Eqs. 17 and 18 by clearing one least-significant bit and adding $x_{3,n}$ on the position 1 yields

$$\begin{aligned}A&=(x_{3,n-1}\ldots x_{3,0}\|\bar{x}_{3,n-1}\ldots\bar{x}_{3,1}\|(\overline{x_{3,0}\vee x_{3,n}}))\\&\quad+(\underbrace{0\ldots0}_{n-1}\|\bar{x}_{3,n}\|\underbrace{0\ldots0}_{n-2}\|x_{3,n}\|0).\end{aligned}\tag{20}$$

One can easily verify that Eq. 17 (18) can be obtained mod $2^{2n}-1$ by setting $x_{3,n}=0$ ($x_{3,n}=1$) in Eqs. 19 and 20. Now, depending on which of Eqs. 19 and 20 is selected, two alternative versions of the converter can be obtained.

4.2 Version 1

First, we merge the constants $|(2^{-n}-1)|_{2^{2n}-1}$, $|2^{-1}(1-2^{n+1})|_{2^{2n}-1}$, and $|2^n2^{-1}|_{2^{2n}-1}$ which appear in respective Eqs. 12, 15, and 19 to obtain one total constant C_k given by

$$\begin{aligned}C_k&=\left|(2^{-n}-1)+2^{-1}(1-2^{n+1})+2^{n-1}\right|_{2^{2n}-1}\\&=\left|2^n-1+2^{-1}-2^n+2^{n-1}\right|_{2^{2n}-1}\\&=\left|2^{n-1}+2^{-1}-1\right|_{2^{2n}-1}=2^{2n-1}+2^{n-1}-1\\&=(1\underbrace{0\ldots0}_{n+2}\underbrace{1\ldots1}_{n-1}).\end{aligned}\tag{21}$$

Second, we define new variables w_1, w_2, and w_3 involving variable terms from respective Eqs. 12, 14, and 19, each involving respectively exactly one of the input residues x_1, x_2, and x_3. For convenience, when rewriting Eqs. 12, 14, and 19 we have performed multiplication by $|2^{-1}|_{2^{2n}-1}$ (here, it is a right cyclic shift by one bit over $2n$ bits) to obtain three following raw binary vectors:

$$w_1 = (\bar{x}_{1,n-1} \dots \bar{x}_{1,0} \| \underbrace{(0 \dots 0)}_{n}), \tag{22}$$

$$w_2 = (x_{2,0} \| x_{2,n-1} \dots x_{2,0} \| x_{2,n-1} \dots x_{2,1}), \tag{23}$$

$$w_3 = \big(\bar{x}_{3,0} \| (x_{3,n-1} \vee x_{3,n}) \dots (x_{3,0} \vee x_{3,n}) \\ \| \bar{x}_{3,n-1} \dots \bar{x}_{3,1}\big). \tag{24}$$

Now Eq. 11 can be rewritten using these new variables w_1, w_2, w_3, and the constant C_k as

$$\begin{aligned} X_h &= \left| -x_1 2^{-n} + x_2 \left(2^n + 1\right) 2^{-1} + x_3 \left(2^n - 1\right) 2^{-1} \right|_{2^{2n}-1} \\ &= |w_1 + w_2 + w_3 + C_k|_{2^{2n}-1}. \end{aligned} \tag{25}$$

Version 1 of the converter can be obtained by substituting in Eq. 10 X_h with its expression of Eq. 25 and designing all circuitry as shown in Fig. 2a)–c). We have intentionally omitted C_k in Fig. 2, as it is expected to be already included in input residues x_2 and x_3, as described below in Section 3.4.

4.3 Version 2

First, we combine the variable terms from Eqs. 13 and 15 according to Eq. 20 and subtract the constant $(1 - 2^{n+1})$ from Eq. 15 to obtain

$$\begin{aligned} &\left| 2^{n+1}\bar{x}_1 + (2^n - 1)x_3 - (1 - 2^{n+1}) \right|_{2^{2n}-1} \\ &= \left| \begin{array}{c} (\bar{x}_{1,n-2} \dots \bar{x}_{1,0} \| (\underbrace{0 \dots 0}_{n} \| \bar{x}_{1,n-1}) \\ +(\underbrace{0 \dots 0}_{n-1} \| \bar{x}_{3,n} \| \underbrace{0 \dots 0}_{n-2} \| x_{3,n} \| 0) \\ + \big(x_{3,n-1} \dots x_{3,0} \| \bar{x}_{3,n-1} \dots \bar{x}_{3,1} \| (\overline{x_{3,0} \vee x_{3,n}})\big) \end{array} \right|_{2^{2n}-1} \\ &= \left| \begin{array}{c} (\bar{x}_{1,n-2} \dots \bar{x}_{1,0} \| \underbrace{(\bar{x}_{3,n} \| 0 \dots 0) \| x_{3,n}}_{n+1} \| \bar{x}_{1,n-1}) \\ + \big(x_{3,n-1} \dots x_{3,0} \| \bar{x}_{3,n-1} \dots \bar{x}_{3,1} \| (\overline{x_{3,0} \vee x_{3,n}})\big) \end{array} \right|_{2^{2n}-1}. \end{aligned} \tag{26}$$

Next, we merge the constants $|2^{-n} - 1|_{2^{2n}-1}$ and $|2^{-1}(1 - 2^{n+1})|_{2^{2n}-1}$ from respective Eqs. 12 and 15 into one total constant C_k given by

$$\begin{aligned} C_k &= \left| (2^{-n} - 1) + 2^{-1}(1 - 2^{n+1}) \right|_{2^{2n}-1} \\ &= \left| 2^n - 1 + 2^{-1} - 2^n \right|_{2^{2n}-1} = \left| 2^{-1} - 1 \right|_{2^{2n}-1} \\ &= 2^{2n-1} - 1 = 0\underbrace{1 \dots 1}_{2n-1}). \end{aligned} \tag{27}$$

As in Version 1, we define two variables w_1 and w_3 involving the input residues x_1 and x_3 from Eq. 26 (which are actually slightly different than in Version 1) whereas the variable w_2 remains the same as before in Eq. 23. When rewriting Eq. 26 we have performed the multiplication by $|2^{-1}|_{2^{2n}-1}$ (the right cyclic shift by one bit over $2n$ bits) to obtain two following raw binary vectors:

$$w_1 = ((\bar{x}_{1,n-1} \dots \bar{x}_{1,0}) \| \bar{x}_{3,n} \| \underbrace{(0 \dots 0)}_{n-2} \| x_{3,n}), \tag{28}$$

$$w_3 = \big((\overline{x_{3,0} \vee x_{3,n}}) \| x_{3,n-1} \dots x_{3,0} \| (\bar{x}_{3,n-1} \dots \bar{x}_{3,1})\big). \tag{29}$$

Version 2 of the converter, whose circuitry is shown in Fig. 2a, b, and d, can be obtained by substituting in Eq. 10 X_h with its expression of Eq. 25 wherein the variables w_1, w_3, and C_k are given respectively by Eqs. 28, 29, and 27.

4.4 Elimination of the Constant C_k

In this section, we will show how the general ideas presented in Section 3 can be applied in practice.

4.4.1 Theoretical Background

Besides three variable components w_1, w_2, and w_3, Eq. 25 for X_h contains one global constant C_k given by Eq. 21 for Version 1 and by Eq. 27 for Version 2. Notice however that the addition of C_k in the calculation of X_h has the same effect as the addition of $2^n C_k$ to X_{23}. Even more, the latter may be also achieved by adding simultaneously $c_2 = |2^n C_k|_{2^n-1} = |C_k|_{2^n-1}$ to x_2 mod $2^n - 1$ and $c_3 = |2^n C_k|_{2^n+1} = |-C_k|_{2^n+1}$ to x_3 mod $2^n + 1$ (the even residue x_1 is left unmodified because $|2^n C_k|_{2^n} = 0$). In summary, the constant C_k indeed can be added either by the CSA part of the reverse converter or beforehand in the datapath channels mod $2^n \pm 1$. The latter possibility is attractive and feasible, because the addition of these constants may be performed at no cost, as they may be initially loaded as the starting values or merged with other constants in the residue datapath channels.

Henceforth, the addition of C_k in X_h is omitted and Eq. 25 for X_h can be rewritten as

$$X_h = |w_1 + w_2 + w_3|_{2^{2n}-1}, \tag{30}$$

which clearly shows that MOMA mod $2^{2n} - 1$ has only three operands. The closed-form expressions for the constants to be added by the datapath channels are as follows.

For Version 1, from Eq. 21 we obtain

$$c_2 = |2^n (2^{2n-1} + 2^{n-1} - 1)|_{2^n-1} = 0, \tag{31}$$

$$\begin{aligned} c_3 &= |2^n (2^{2n-1} + 2^{n-1} - 1)|_{2^n+1} \\ &= |2^n (-2^{n-1} + 2^{n-1} - 1)|_{2^n+1} = 1. \end{aligned} \tag{32}$$

It means that the datapath channel mod $2^n - 1$ does not have to be modified at all, whereas the value produced by the datapath channel mod 2^n+1 only needs to be incremented by $c_3 = 1$. For Version 2, from Eq. 27 we obtain

$$c_2 = |2^n(2^{2n-1}-1)|_{2^n-1} = 2^{n-1}-1 = (0\underbrace{1\ldots1}_{n-1}), \quad (33)$$

$$c_3 = |2^n(2^{2n-1}-1)|_{2^n+1} = |2^n(-2^{n-1}-1)|_{2^n+1} = 2^{n-1}+1 = (01\underbrace{0\ldots0}_{n-2}1). \quad (34)$$

4.4.2 Shifting of Constants to the Datapath Channels

Figure 3 shows the RNS datapath using the 3-moduli set $\{2^n-1, 2^n, 2^n+1\}$ and the reverse converter which realizes Eq. 10 in which X_h is given by Eq. 30. We assume that the initial version of the RNS datapath (prior shifting out the constants from the reverse converter) implements the set of $r = 3$ equations (7), in which the j-th stage of the datapath ($1 \le j \le N$) is composed of three MACs mod m_i, $1 \le i \le 3$, each of which realizes Eq. 8, where the initial values are generally assumed $|S_0|_{m_i} = 0$. All MACs (mod 2^n, 2^n-1, and 2^n+1) can be implemented using the design approach presented in [10]. Also, we suggest two following solutions for adding the constants c_2 and c_3 in their respective datapath channels mod 2^n-1 and 2^n+1.

Solution 1 Suppose that each computation is initialized by activating the Clear signals in the registers $|S_0|_{m_i}$ containing the initial values mod m_i, $i = \{1, 2, 3\}$. Then, the effect of adding mod 2^n-1 and 2^n+1 the respective constants c_2 and c_3 in the residue datapaths mod $m_2 = 2^n-1$ and $m_3 = 2^n+1$ can be obtained without any extra delay and virtually with no hardware cost by activating in these registers the Preset signal allowing to load the constants c_2 and c_3 in the channels mod 2^n-1 and 2^n+1.

Solution 2 Should the above simplest solution be unfeasible for some reason, which seems rather unlikely, the constants c_2 and c_3 can be added at any other stage of computation in respective residue datapath channels. We will detail the case of Version 2, because adding mod 2^n-1 and 2^n+1 of the respective constants $c_2 = 2^{n-1}-1$ and $c_3 = 2^{n-1}+1$ is more difficult than for the case of Version 1, for which always $c_2 = 0$ and $c_3 = 1$, and which will appear later as a special case.

First, consider the datapath channels mod 2^n-1. Conceptually the simplest MAC mod 2^n-1 [10] is nothing else but the array of n 2-input AND gates followed by the $(n+1)$-operand MOMA mod 2^n-1. Obviously, no modifications are required for Version 1, because $c_2 = 0$. For Version 2, the addition of c_2 requires introducing somewhere in the datapath one CSA stage composed of n half-adders (with carry outputs non-complemented or complemented, depending on whether a given bit of c_2 is 0 or 1, respectively). It can be done by modifying an arbitrarily selected MAC of the datapath mod 2^n-1. Such a modification would not incur any increase of the number of CSA stages

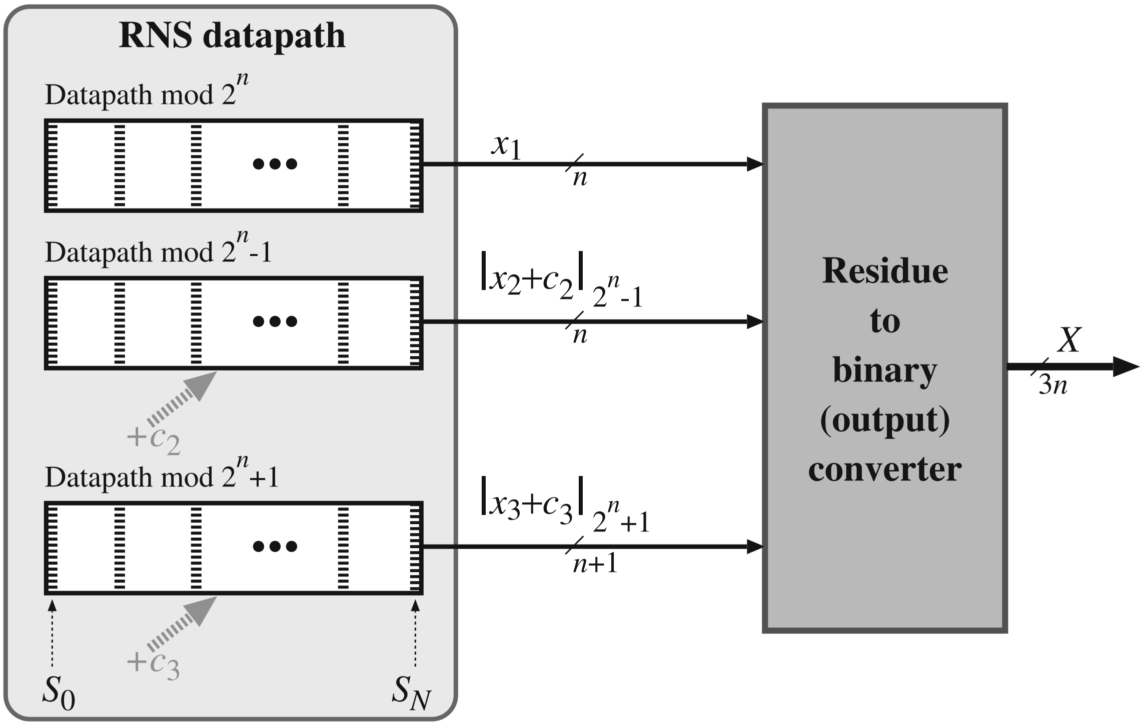


Figure 3 Scheme of shifting constants to residue datapath channels in the reverse converter for the 3-moduli set $\{2^n-1, 2^n, 2^n+1\}$.

of the MAC mod $2^n - 1$, except for those n for which $\theta(n+2) = \theta(n+1)+1$, where $\theta(p)$ denotes the number of CSA stages on a CSA tree that processes p input operands), i.e., only for $n = 3, 5, 8, 12, 18, 27$, etc.

As for the datapath channel mod 2^n+1, several possibilities exist, which do not involve any changes which could result in the increasing of the area or delay. However, to clarify this issue, we must first recall some details regarding the internal logic structure of a MAC mod $2^n + 1$, e. g. one proposed in [10]. The latter design employs the tree of n-bit CSAs with some signals complemented and, in particular, complemented End-Around Carry (EAC) signals. In a given arithmetic circuit mod $2^n + 1$, any complemented signal of weight 2^i requires to add mod $2^n + 1$ the corrective constant equal to $|-2^i|_{2^n+1}$ [21]. For such a circuit, a cumulative total constant $|C_{total}|_{2^n+1}$ can be calculated and added whenever the most convenient, i.e., it does not have to be added at each stage of the datapath mod $2^n + 1$, unless an intermediate unbiased result is actually needed. In the best case, a cumulative total constant $|C_{total}|_{2^n+1}$ can be added only once, e.g., at the final stage of the datapath channel mod 2^n+1, when the final result $|S_N|_{2^n+1} = x_3$ is produced, which is one of three input signals of the reverse converter considered here (recall also that x_3 assumes only the valid values mod $2^n + 1$, i.e., $0 \le x_3 \le 2^n$). Obviously, for all cases when $|C_{total}|_{2^n+1} \neq 0$, the addition of c_3 can be done at no cost by replacing the addition of $|C_{total}|_{2^n+1}$ with the addition of $|C_{total} + c_3|_{2^n+1}$. In the remaining special case of $|C_{total}|_{2^n+1} = 0$, the adder mod $2^n + 1$ used at the final stage of the datapath channel mod $2^n + 1$, which adds the pair of n-bit vectors in the carry-save form $C^* = (c^*_{n-2}\dots c^*_0\overline{c^*_{n-1}})$ and $S^* = (s^*_{n-1}\dots s^*_0)$ to compute $x_3 = |C^*+S^*|_{2^n+1}$, must be modified, as it should produce $|C^*+S^*+c_3|_{2^n+1}$ now. The most obvious general implementation of the latter involves introducing one n-bit CSA with complemented EAC (the latter imposes to subtract -1, so that the adder actually computes $|C^*+S^*+c_3-1|_{2^n+1}$) followed by the adder mod 2^n+1. Such a modification involves the extra delay of one half-adder and the area cost of n extra half-adders (which still reduces by a half the total of $2n$ half-adders which would be used, should this constant be added within the reverse converter). Finally, the special case of $|C_{total} + c_3|_{2^n+1} = 1$ (which is also the case of Version 1) involves no extra cost at all, because the addition $|C^* + S^* + 1|_{2^n+1}$ can be implemented using the n-bit adder with inverted EAC of [37].

4.4.3 Example

Consider an RNS-based implementation of the 16-tap FIR filter with 8-bit input operands and 8-bit coefficients. The required dynamic range of 20 bits is guaranteed by the 3-moduli set with $n = 7$ $\{m_1, m_2, m_3\} = \{128, 127, 129\}$. Eq. 1 is implemented using three residue datapaths mod m_i, $1 \le i \le 3$, each of which realizes the function

$$|S_{16}|_{m_i} = \left|\sum_{j=0}^{15}\left(\left||C_j|_{m_i}\cdot|X_j|_{m_i}\right|_{m_i}\right)\right|_{m_i}, \tag{35}$$

where the initial value of $|S_0|_{m_i}$ is usually set to 0. Three basic MAC units mod 128, 127, and 129, required to built these residue datapaths, can be implemented using the design approach presented in [10], although some slightly improved versions proposed in [25] can be used as well. The general scheme is as shown in Fig. 3, assuming that the following constants are added inside the residue datapath channels: $c_2 = 0$ and $c_3 = 1$—for Version 1, and $c_2 = 63$ and $c_3 = 65$—for Version 2. Should it be feasible for an actually used architecture, at the beginning of computations the residue datapaths registers containing the initial values $|S_0|_{2^n-1}$ and $|S_0|_{2^n+1}$ are loaded with the suitable constants required by the reverse converter rather than with 0's, i.e., $|S_0|_{2^n-1} = c_2$ and $|S_0|_{2^n+1} = |C_{total} + c_3|_{2^n+1}$. Should loading of initial non-zero constants c_2 and c_3 be unfeasible, some of the alternative methods proposed in the previous subsection could be used. Further, in this example, we disregard the actual value of $|C_{total}|_{2^n+1}$ (the only other constant to be added, which can be calculated according to the internal structure of the MAC mod $2^n + 1$), take into account c_2 and c_3, and assume that the result of the calculations from the residue datapaths $\{x_1, x_2, x_3\} = \{1, 2, 3\}$.

For Version 1, adding the values of $c_2 = 0$ and $c_3 = 1$ to the appropriate datapath channels results in $\{x_1, x_2, x_3\} = \{1, 2, |1 + 3|_{129}\} = \{1, 2, 4\}$, which is in binary $\{x_1, x_2, x_3\} = \{(0000001), (0000010), (00000100)\}$. From Eqs. 22, 23, and 24 respectively, we have:

$$\begin{aligned}
w_1 &= ((\bar{x}_{1,n-1}\dots\bar{x}_{1,0})\|\underbrace{(0\dots0)}_{n})\\
&= (11111100000000) = 16128,\\
w_2 &= \left(x_{2,0}\|(x_{2,n-1}\dots x_{2,0})\|(x_{2,n-1}\dots x_{2,1})\right)\\
&= (0\|0000010\|000001) = 129,\\
w_3 &= \left((\bar{x}_{3,0}\|(x_{3,n-1}\vee x_{3,n})\dots(x_{3,0}\vee x_{3,n}))\|(\bar{x}_{3,n-1}\dots\bar{x}_{3,1})\right)\\
&= (1\|0000100\|111101) = 8509.
\end{aligned}$$

From Eq. 30 we obtain $X_h = |w_1 + w_2 + w_3|_{2^{2n}-1} = |16128 + 129 + 8509|_{16383} = 8383$, which substitutes X_h in Eq. 10 to provide the final value of X given by $X = x_1 + 2^n X_h = 1 + 2^7 \cdot 8383 = 1073025$.

For Version 2, adding the values of $c_2 = 63$ and $c_3 = 65$ to the appropriate datapath channels results in $\{x_1, x_2, x_3\} = \{1, |2 + 63|_{127}, |3 + 65|_{129}\} = \{1, 65, 68\}$,

$\{x_1, x_2, x_3\} = \{(0000001), (1000001), (01000100)\}$. From Eqs. 28, 23, and 29 respectively, we have:

$$\begin{aligned} w_1 &= ((\bar{x}_{1,n-1} \dots \bar{x}_{1,0}) \| \bar{x}_{3,n} \| \underbrace{(0 \dots 0)}_{n-2} \| x_{3,n}) \\ &= (11111101 \| 00000 \| 0) = 16192, \\ w_2 &= (x_{2,0} \| (x_{2,n-1} \dots x_{2,0} \| x_{2,n-1} \dots x_{2,1})) \\ &= (1 \| 1000001 \| 100000) = 12384, \\ w_3 &= \left((\overline{x_{3,0} \vee x_{3,n}}) \| (x_{3,n-1} \dots x_{3,0}) \| (\bar{x}_{3,n-1} \dots \bar{x}_{3,1})\right) \\ &= (1 \| 1000100 \| 011101) = 12573. \end{aligned}$$

From Eq. 30 we obtain the same value of $X_h = |w_1 + w_2 + w_3|_{2^{2n}-1} = |16192 + 12384 + 12573|_{16383} = 8383$.

In summary, we have shown the feasibility of avoiding the cost of one additional CSA stage in our new converters, by assuming that the appropriate constants are already added to the input residues x_2 and x_3 of the converter.

4.5 Constant Removal From the Reverse Converters for 4- and 5-moduli sets

The same rules of shifting out constants as presented above can be directly applied to other reverse converters for the special multi-moduli sets constructed by extending the classic 3-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, like those of [1, 6, 20], in which the reverse converter for the 3-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ can be used as a subcircuit. For instance, it can be done by first dividing the moduli set into at least two subsets, one of which is the 3-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. The residues corresponding to this moduli set are transformed using the reverse converter to obtain one number, which is an intermediate result for this moduli set. (In particular, it can be the converter proposed above, provided that the necessary constants are added in residue datapath channels mod $2^n \pm 1$.)

Next, the two-moduli MRC from Eq. 5 is applied once or twice to complete the reverse converter for the 4- or 5-moduli set, respectively. The application of the MRC in this step is nothing else but the modulo generation from the number obtained in the preceding step followed by the modular subtraction and multiplication by the multiplicative inverse. Depending on a specific implementation of this step, a number of constants can appear. As this part of the converter is a kind of residue datapath modulo one of the two last moduli, the constants involved in these steps (or one last step, in the case of the 4-moduli set) can then be shifted out to the residue datapath channels corresponding to these moduli.

5 Complexity Evaluation

In this section, we will evaluate the gate-level complexity of a number of converters for the 3-moduli set $\{2^n, 2^n - 1, 2^n + 1\}$, including: Versions 1 and 2 of our converters, three specific designs which in the literature have been considered to be the most efficient [12, 34, 35], and two converters for the general 3-moduli set $\{2^k, 2^n - 1, 2^n + 1\}$ designed for the special case of $k = n$ [8, 36]. (To note that because the expressions used to implement all the converters of [12, 34, 35] contain no explicit constants, our technique of shifting the constants to the datapath channels can be applied only to the converters proposed here.) Here, we count logic gates, based on a number of basic primitives used in the architectural design detailed in the previous section, and then we estimate the circuit delay as the number of logic primitives present on the critical path. In the next section, we will evaluate the results of logic synthesis of all converters using commercial design tools, which would provide not only more accurate estimations of the area and delay but also of the power consumption.

5.1 Hardware Complexity Evaluation

In Table 1, which summarizes the gate-level hardware complexity, we can distinguish three groups of components. The first one consists of bit-level manipulation components like primitive gates, inverters and MUXes (which we also treat as primitive gates). As in every converter mentioned there is a 3-operand [12, 35] or a 4-operand [8, 34, 36] addition mod $2^{2n} - 1$, the second group are full-adders (FAs) and

Table 1 Hardware complexity (in [8] and [36] $k = n$).

Element	Ver.1	Ver.2	[8]	[12]	[34]	[35]	[36]
OR	n	1	1	1	–	n	–
AND	–	–	–	1	–	–	–
XOR	–	–	–	–	1	–	–
NOT	$2n$	$2n+1$	$2n+1$	$2n+1$	$3n-1$	$2n$	$2n+1$
MUX	–	–	–	–	2	–	–
Full-adder (FA)	n	$n+2$	$2n+1$	$2n$	$2n$	$2n$	$2n+1$
Half-adder (HA)	n	$n-2$	$2n-1$	–	$2n$	–	$2n-1$
Add. mod $2^{2n}-1$	1	1	1	1	1	1	1

half-adders (HAs) which form $2n$-bit CSA stages producing the carry-save vector pair. The third group is the final adder mod $2^{2n}-1$ reducing the carry-save pair into the final result. The differences between converters are revealed in the first two groups of components, because the same final adder mod $2^{2n}-1$ occurs in all converters. It is seen that Version 1 of our converter is comparable with the converter from [35] (as it uses similar design assumptions), while in the other converters (notably in Version 2 of our converter) the number of two-input gates is fixed and does not depend on n. A real advantage of our new design (resulting from removal of the constants) may be seen in the total number of full- and half-adders. While in all converters there are at least $2n$ full-adders accompanied in [8, 34, 36] by about $2n$ half-adders, in both versions of our design the number of full- and half-adders is about n.

In summary, the above estimations suggest that our converters should occupy slightly smaller area resulting from smaller number of full- and half-adders.

5.2 Gate-Level Delay Evaluation

Table 2 provides the estimations of the delay of all converters considered. Each column shows the data of one converter, with delay components appearing in consecutive rows according to the order of calculations. Besides all the logic primitives included in the analysis of the hardware complexity, we have also included some fanout characteristics (expressed by the number of inputs driven by one signal), because of the following reasons. As gates have limited fanout, should it be exceeded, the logic synthesis tool either builds a buffer tree which drives larger number of gates or uses slower gates with higher output fanout. This may result in a synthesized circuit which is actually slower than it would indicate direct delay analysis of all components present on the critical path (indeed, most estimations presented in the open literature count only the number of gate levels, regardless on fanout). To note, however, that in all 3-moduli converters considered here, the only signals with high fanouts are the primary inputs to the converter. To expose the high fanout issue, we have found the maximal fanouts on the converter input (which we understood simply as the number of occurrences of some literals $x_{i,j}$ $(1 \le i \le 3)$ in bit-level manipulation expressions) and included them in the first row of Table 2 as $d_f(a)$, where a is the maximal number of gate inputs driven by the primary input.

Prior comparing delay, notice that any significant delay reduction seems unfeasible, because the most significant part of the overall delay is contributed by the final adder mod $2^{2n}-1$, whose usage could not have been avoided in any design proposed to date. It is seen that besides fanout-related delay, any differences between the delays of various converters stem from the number of CSA stages on the critical path. The first (faster) group of designs includes two versions of our converters and the converters from [12, 35] which have only one FA stage, whereas the second (slower) group with two FA stages includes the converters from [8, 34, 36]. In the faster group, the delay of Version 2 is comparable with the delay of the converter from [12], which is due to small fanout and using similar gates for bit-level manipulations. The second best are Version 1 and the converter from [35] with the fanout equal to n in either case. In the slower group, the fastest seems to be the converter from [36], while the slowest is the converter from [34] which has large fanout depending on n and uses a XOR gate for its bit-level manipulations.

6 Experimental Evaluation

6.1 Synthesis Assumptions

To obtain as much as possible realistic complexity figures, we have performed logic synthesis of both versions of our converter as well as their best known counterparts from [8, 12, 34–36]. In an attempt to produce systematic and fairly comparable descriptions, hardware description of all converters was done in parametrized structural Verilog, following identical coding guidelines and similar module/sub-module layout, i.e., we separated bit-level manipulations, CSAs, and adders mod $2^{2n}-1$ (designed according to [17]), and used hierarchy preserving feature of the logic synthesis tool.

Table 2 Delay estimations (in [8] and [36] $k=n$).

Ver. 1	Ver. 2	[8]	[12]	[34]	[35]	[36]
$d_f(n)+$	$d_f(3)+$	$d_f(3)+$	$d_f(3)+$	$d_f(n+1)+$	$d_f(n)+$	$d_f(2)+$
	$d_{OR}+$		$d_{NOT}+$			$d_{NOT}+$
$d_{OR}+$	$d_{NOT}+$	$d_{OR}+$	$d_{OR}+$	$d_{XOR}+$	$d_{OR}+$	
$d_{FA}+$	$d_{FA}+$	$2d_{FA}+$	$d_{FA}+$	$2d_{FA}+$	$d_{FA}+$	$2d_{FA}+$
$d_m(2n)$	$d_m(2n)$	$d_m(2n)$	$d_m(2n)$	$d_m(2n)$	$d_m(2n)$	$d_m(2n)$

$d_f(a)$—the delay of the fanout tree of size a
$d_m(2n)$—the delay of an adder mod $2^{2n}-1$

We have performed logic synthesis of all converters for a range of delay targets using Cadence RC Compiler (v. 8.10-s222_1) over the commercial CMOS065LP 65 nm low-power library from ST Microelectronics (CORE/ CLOCK65LPSVT, v. 5.2.2). As is customary, to obtain more realistic delay and power estimations, we have added input and output registers to all designs. For each design, the minimum delay was found, which we understood as the smallest delay target for which the logic synthesis tool still reported a non-negative timing slack. Next, we have performed physical synthesis on rectangular die whose size was selected to obtain the density about 75 % (we have achieved the values ranging from 70 to 80 %). Physical synthesis was performed using Cadence Encounter (v. 10.12-s181_1) and NanoRoute (v. 10.12-s010) tools. We have used the same scripts for synthesis of all converters, without any specific changes nor optimization.

The results of the place and route for all dynamic ranges (DR) from 8 to 38 bits, namely power- and area consumption at the minimal delay are visualized respectively in Figs. 5 and 6, and the minimal delay itself is shown in Fig. 4). It can be also observed that the results of place and route timing in general follow the estimations given by the logic synthesis tool alone, which seems to be the result of a relatively simple design size wherein the impact of placement and/or routing on the final timing is limited.

6.2 Delay Evaluation

Figure 4 reveals the logarithmic growth of the minimal delay with the dynamic range for all converters considered, which is the direct outcome of the logarithmic depth of the final parallel prefix adder [17] (which is the main delay contributor). It is also seen that delay differences remain fairly constant for all converters. The synthesis results confirm that our converters are the fastest for all dynamic ranges considered, which could be attributed to a smaller number of potentially critical paths in the final addition (our converters involve about two and a half of operands compared to three complete operands in those of [12, 35]), which in turn enabled the logic synthesis tool to use larger and faster gates (because they are fewer) having smaller overall impact on the final power and area.

While the delays of both Versions 1 and 2 of our converters are largely the same for most of the dynamic ranges larger than 8, a noticeable exception are the dynamic ranges of 35 and 38 bits: a likely explanation of larger delay of Version 1 than of Version 2 seems to be larger fanout in the former, resulting in a deeper buffer tree (or slower gates driving large fanout). The special cases of the converters of [8, 36] involve the addition of four operands resulting in an additional CSA stage contributing about 100 ps extra delay. The slowest converter amongst ASIC-specific designs is the converter from [34], in which further 50 ps is added by one XOR gate (see the fifth column of Table 2).

6.3 Power Consumption Evaluation

Figure 5 shows the power consumption of all designs considered. Power consumption presented was obtained as a sum of dynamic and leakage power from the simulation using 1000 random vectors using PrimeTime PX v2012.1H on the netlists provided by the physical synthesis tool. 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 is the result of the simulation

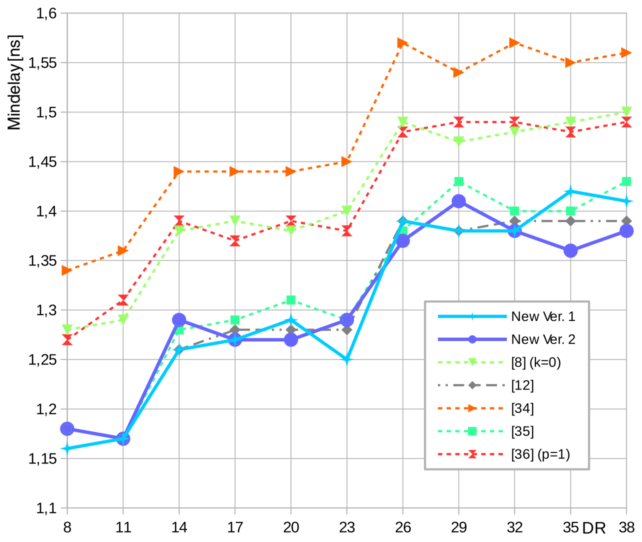


Figure 4 Minimum delay as a function of the dynamic range. (*Note:* For $DR = 8$ and 11, the delay of the converters of [12, 35] is the same as for new Version 2).

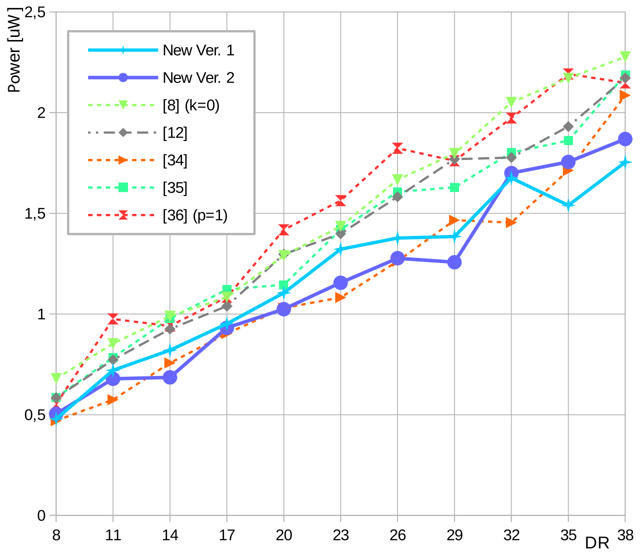


Figure 5 Power at minimal delay.

and internal estimations performed by the power simulation tool. Due to this, the power consumption estimations contain more nonlinearities and their contributing factors are more difficult to explain exclusively on the basis of the internal architecture.

In general, our converters have smaller power consumption (accompanied by smaller minimal delay) than all their counterparts. The power consumption of Version 1 is, on average, smaller than its counterpart of Version 2. Larger power consumption of the converters from [8] than those from [36], which both have very similar design and introduce nearly the same minimal delay, requires additional consideration. A likely reason is the presence of the additional OR gate on the critical path in [8] (compared to uniformly distributed NOT gates in [36]) that puts additional strain on all paths originating from this OR gate, as the logic synthesis tool must to scale up all gates on these paths which results in higher power consumption. Finally, it is noticeable that the converters from [34] enjoy relatively small power consumption which, unfortunately, is accompanied by the largest delay.

6.4 Area Evaluation

Figure 6 shows the area of all synthesized converters, obtained as the sum of reported areas occupied by logic cells and estimated area of interconnections. As these figures strictly depend on the numbers of cells used, they are more exact and consistent than power estimations.

The area occupied by all presented converters in general follows a slightly faster than linear trend resulting from the composition of $O(n \log n)$ complexity of the final adder and linear complexities of other components (cf. Table 1), so no significant differences are observed between various converters. A slight vertical shift from the linear trend of the area observed for nearly all converters between 23 and 26 bits results from the depth of the parallel prefix final adder [17] (as n changes from 8 to 9).

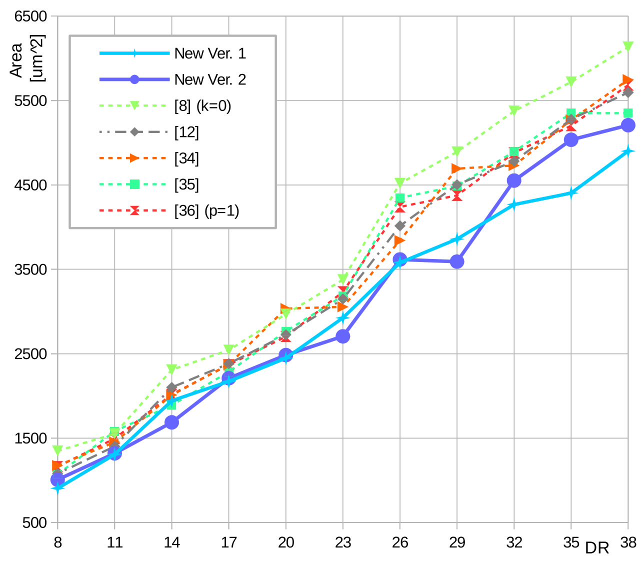


Figure 6 Area at minimal delay.

In general, the area of our converters is smaller than of all their counterparts for all dynamic ranges considered. We can observe also that delay, power, and area are somewhat correlated in our converters. For instance, for the dynamic ranges of 14, 23, and 29 bits, we observe higher delay of Version 2 accompanied by nearly the same power consumption and compensated by smaller area (smaller gates consume more power despite higher delay). Also, Version 2 of our converter occupies larger area for larger dynamic ranges (from 32 bits), which results from smaller delay achieved (despite that Version 1 uses a number of OR gates not used in Version 2). It seems that larger area of the converters from [12, 35] results from using $2n$ full-adders (Table 1 reveals that in our converters, there are roughly n full-adders and n half-adders). The area of the converters from [8, 34, 36] remains close to all other converters but, as it was observed before, it is accompanied by higher delay (thus the relatively small area results from the use of smaller and consequently slower gates).

6.5 Summary

Table 3 summarizes the improvement, in percentage terms, of our converters with respect to their best existing counterparts of [12, 35] as a function of the dynamic range: for a given parameter, the best of our two versions and of [12, 35] is compared. The delay of the new converters is either the same as in the previous designs or it is only slightly reduced, up to 2.34 %. It is not surprising, because in all converters considered the final adder mod $2^{2n}-1$ (preceded by at least two CSA stages) is the main delay contributor.

Table 3 Percentage assessment of improvements obtained.

DR	Delay	Area	Power	ADP	PDP
8	1.69	15.41	11.72	16.84	13.22
11	0.00	6.77	17.83	6.77	17.83
14	0.00	10.84	21.84	10.14	21.23
17	0.78	4.95	10.37	6.42	11.76
20	0.78	10.13	25.37	9.52	25.95
23	2.34	14.07	15.41	13.40	14.75
26	0.72	10.83	19.73	11.23	19.73
29	0.00	19.95	18.03	18.50	19.18
32	0.72	10.71	11.71	11.35	12.97
35	2.16	16.56	27.01	14.76	25.96
38	0.72	8.39	16.69	9.67	17.85
Average	0.90	11.69	17.79	11.69	18.22

Consequently, replacing one CSA stage with one stage of logic gates performing simple bit manipulations can only result in a relatively small delay reduction (0.9 % on average). Nevertheless, the area is reduced from about 5 % to about 20 % (on average by 11.69 %), and it is accompanied by even larger savings in the power consumption—from over 10 % up to 27 % (on average by about 17.79 %). Consequently, due to negligible delay reduction (if any), the reductions obtained for the area-delay and power-delay products are highly correlated with those observed for the area and power consumption: they range for the area-delay product from over 6 % up to over 18.5 % (on average by 11.69 %) and for the power-delay product from about 13 % up to about 26 % (on average by 18.2 %).

7 Conclusion

In this paper, a new general approach to improving characteristics of reverse (residue-to-binary) converters for a Residue Number System (RNS) composed of arbitrary moduli sets was suggested. The main idea is to formulate the basic equation of the reverse converter in a form consisting of two separate parts: one depending on input variables of the reverse converter whereas the other is a single constant. Such a separation allows to reduce the number of operands added inside the reverse converter by one, because the constant, instead of being added inside the reverse converter, can be shifted out to the residue datapath channels. We have argued that adding the constant in the residue datapath channels, in most cases can be done at no hardware cost or extra delay, so that applying this design approach can lead to overall power and area reduction and in some cases also to decreasing delay. Some suggestions which facilitate obtaining the reverse converter equation with separated constants were also given.

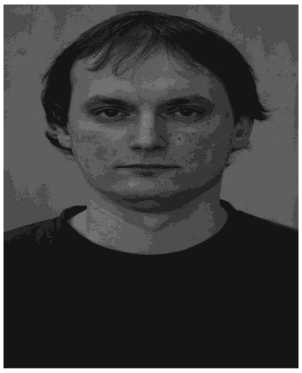
To illustrate various design issues of this new design approach and to prove its efficiency, a new design method of the residue-to-binary (reverse) converters for the popular classic 3-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ was considered. Investigations of different bit manipulations of the converter's input operands resulted in its two new versions. Unlike any of previous designs, the new sets of equations contain separated constants, which can be shifted out from the converter to the datapath channels at no cost, thus reducing the cost of the tree of carry-save adders (CSAs) of the converter. Experimental results suggest that compared to all of the state-of-the-art converters for this 3-moduli set, the converters obtained using the newly proposed approach are superior with respect to the area and power consumption which are reduced on average by about 12 % and 18 %, respectively, while delay is the same or slightly smaller. As several larger multi-moduli RNSs have been proposed by extending the set $\{2^n - 1, 2^n, 2^n + 1\}$, for which reverse converters have been constructed using the converter for the classic 3-moduli set as a basic building block, all of the latter converters (including those that will be proposed in the future) could also enjoy better performance, once their basic building block is improved. Future research will include considering the possibility of formulating equations of reverse converters for other moduli sets, in which variable and constant parts could be separated, which would allow to add constants within the residue datapath channels and would likely result in more efficient converters.

References

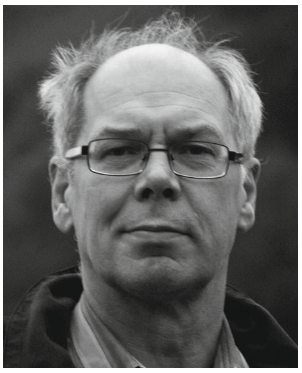
1. Ananda Mohan, P.V., & Premkumar, A.B. (2007). RNS-to-binary converters for two four-moduli sets $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ and $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *54*(6), 1245–1254.
2. Ananda Mohan, P.V., Premkumar, A.B., & Bhardwaj, M. (2001). Comments on "Breaking the $2n$-bit carry-propagation barrier in residue to binary conversion for the $[2^n - 1, 2^n, 2^n + 1]$ moduli set" and Author's Reply. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *48*(8), 1031.
3. Bernardson, P. (1985). Fast memoryless, over 64 bits, residue-to-binary convertor. *IEEE Transactions on Circuits and Systems*, *CAS-32*(3), 298–300.
4. Bhardwaj, M., Premkumar, A.B., & Srikanthan, T. (1998). Breaking the $2n$-bit carry propagation barrier in residue to binary conversion for the $[2^n - 1, 2^n, 2^n + 1]$ modula set. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, *45*(9), 998–1002.
5. Bi, G., & Jones, E.V. (1988). Fast conversion between binary and residue numbers. *Electronics Letters*, *24*(19), 1195–1197.
6. Cao, B., Chang, C.H., & Srikanthan, T. (2007). A residue-to-binary converter for a new five-moduli set. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *54*(5), 1041–1049.
7. Cao, B., Srikanthan, T., & Chang, C.H. (2005). Efficient reverse converters for the four-moduli sets $\{2^n, 2^n - 1, 2^n + 1, 2^{n+1} - 1\}$ and $\{2^n, 2^n - 1, 2^n + 1, 2^{n-1} - 1\}$. *IEE Proceedings - Computers and Digital Techniques*, *152*(5), 687–696.
8. Chaves, R., & Sousa, L. (2007). Improving residue number system multiplication with more balanced moduli sets and enhanced modular arithmetic structures. *IEE Journal on Computers and Digital Techniques*, *1*(5), 472–480.
9. Conway, R., & Nelson, J. (1999). Fast converter for 3 moduli RNS using new property of CRT. *IEEE Transactions on Computers*, *48*(8), 852–860.
10. Conway, R., & Nelson, J. (2004). Improved RNS FIR filter architectures. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *51*(1), 26–28.
11. Dhurkadas, A. (1990). Comments on "An efficient residue to binary converter design" by K. M. Ibrahim and S. N.

Saloum. *IEEE Transactions on Circuits and Systems*, *37*(6), 849–850.

12. Dhurkadas, A. (1998). Comments on “A high speed realization of a residue to binary number system converter”. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, *45*(3), 446–447.
13. Ibrahim, K.M., & Saloum, S.N. (1988). An efficient residue to binary converter design. *IEEE Transactions on Circuits and Systems*, *35*(9), 1156–1158.
14. Jenkins, W.K. (1978). Techniques for residue-to-analog conversion for residue-encoded digital filters. *IEEE Transactions on Circuits and Systems*, *CAS-25*(7), 555–562.
15. Liu, Y., & Lai, E.M.K. (2004). Moduli set selection and cost estimation for RNS-based FIR filter and filter bank design. *Design Automation for Embedded Systems*, *9*(2), 123–139.
16. Omondi, A., & Premkumar, B. (2007). *Residue number systems: theory and implementation*. London: Imperial College Press.
17. Patel, R., & Boussakta, S. (2007). Fast parallel-prefix architectures for modulo $2^n - 1$ addition with a single representation of zero. *IEEE Transactions on Computers*, *56*(11), 1484–1492.
18. Patronik, P., Berezowski, K., Biernat, J., Piestrak, S.J., & Shrivastava, A. (2012). Design of an RNS reverse converter for a new five-moduli special set. In *Proceedings on ACM Great Lakes symposium VLSI (GLSVLSI)* (pp. 67–70). Salt Lake City.
19. Patronik, P., & Piestrak, S.J. (2014). 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 Transactions on Circuits and Systems I: Regular Papers*, *61*(6), 1687–1700.
20. Patronik, P., & Piestrak, S.J. (2014). Design of reverse converters for the new RNS moduli set $\{2^n + 1, 2^n - 1, 2^n, 2^{n-1} + 1\}$ (n odd). *IEEE Transactions on Circuits and Systems I: Regular Papers*, *61*(12), 3436–3449.
21. Piestrak, S.J. (1994). Design of residue generators and multi-operand modular adders using carry-save adders. *IEEE Transactions on Computers*, *43*(1), 68–77.
22. Piestrak, S.J. (1995). A high-speed realization of a residue to binary number system converter. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, *42*(10), 661–663.
23. Piestrak, S.J. (2011). Design of multi-residue generators using shared logic. In *IEEE international symposium on circuits and systems (ISCAS)* (pp. 1435–1438). Rio de Janeiro.
24. Piestrak, S.J., & Berezowski, K.S. (2008). Architecture of efficient RNS-based digital signal processor with very low-level pipelining. In *IET Irish signals and systems conference* (pp. 127–132). Galway.
25. Piestrak, S.J., & Berezowski, K.S. (2008). Design of residue multipliers-accumulators using periodicity. In *IET Irish signals and systems conference* (pp. 380–385). Galway.
26. Skavantzos, A. (1998). Efficient residue to weighted converter for a new residue number system. In *Proc. IEEE Great Lakes symposium VLSI (GLSVLSI)* (pp. 185–191). Lafayette.
27. Skavantzos, A., Abdallah, M., & Stouraitis, T. (2007). Large dynamic range RNS systems and their residue to binary converters. *Journal of Circuits, Systems and Computers*, *16*(2), 267–286.
28. Smitha, K.G., & Vinod, A.P. (2012). A reconfigurable channel filter for software defined radio using RNS. *Journal of Signal Processing Systems*, *67*(3), 229–237.
29. Sousa, L., & Antão, S. (2012). MRC-based RNS reverse converters for the four-moduli sets $\{2^n + 1, 2^n - 1, 2^n, 2^{2n+1} - 1\}$ and $\{2^n + 1, 2^n - 1, 2^{2n}, 2^{2n+1} - 1\}$. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *59*(4), 244–248.
30. Sousa, L., Antão, S., & Chaves, R. (2013). On the design of RNS reverse converters for the four-moduli set $\{2^n + 1, 2^n - 1, 2^n, 2^{n+1} + 1\}$. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *21*(10), 1945–1949.
31. Sweidan, A., & Hiasat, A. (1988). A new efficient memoryless residue to binary converter. *IEEE Transactions on Circuits and Systems*, *35*(11), 1441–1444.
32. Taylor, F.J., & Ramnarayan, A. (1981). An efficient residue-to-decimal converter. *IEEE Transactions on Circuits and Systems*, *CAS-28*(12), 1164–1169.
33. Wang, Y. (2000). Residue-to-binary converters based on new Chinese Remainder Theorem. *IEEE Transactions on Circuits and Systems II*, *47*(3), 197–205.
34. Wang, Y., Song, X., Aboulhamid, M., & Shen, H. (2002). Adder based residue to binary number converters for $(2^n - 1, 2^n, 2^n + 1)$. *IEEE Transactions on Signal Processing*, *50*(7), 1772–1779.
35. Wang, Z., Jullien, G.A., & Miller, W.C. (2000). An improved residue-to-binary converter. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, *47*(9), 1437–1440.
36. Wey, C.L. (2006). Residue-to-binary converters for high-speed digital signal processing. In *Proceedings on IEEE international conference on electro-information technology* (pp. 421–426). East Lansings.
37. Zimmerman, R. (1999). Efficient VLSI implementation of modulo $2^n \pm 1$ addition and multiplication. In *Proceedings on the IEEE symposium on computer arithmetic* (pp. 158–167). Adelaide.



Piotr Patronik received the MSc and PhD degrees both in computer engineering from the Wroclaw University of Technology in 2001 and 2006, respectively. He is currently an Assistant Professor at the Faculty of Electronics of the Wroclaw University of Technology, Wroclaw, Poland. During academic year 2012–2013 he was on leave at the Institut Jean Lamour, Université de Lorraine, Nancy, France, where he participated in the ARDyT project funded by the ANR. His research interests include design of VLSI digital circuits, computer arithmetic, fault-tolerant computing, and parallel computing.



Stanisław J. Piestrak is a Professor at the Institut Jean Lamour/Université de Lorraine, Nancy/Metz, France. He received the Ph.D. and Habilitation degrees both in computer science in 1982 and 1996, respectively from the Wroclaw University of Technology and Gdansk University of Technology, in Poland. His research interests include design of VLSI digital circuits, fault-tolerant computing (self-checking circuits design, coding theory, and reconfigurable systems), and computer arithmetic (design of hardware for appications using residue number system (RNS)).