On Reverse Converters for Arbitrary Multi-Moduli RNS

Piotr Patronik[a]

[a] *Dept. of Computer Engg, Faculty of Electronics (W-4/K-9), Wrocław University of Technology, 50–370 Wrocław, Poland*

Abstract

In this paper, we propose a new design of reverse converters for residue number systems with arbitrary moduli sets consisting of any number of odd moduli and one even modulus of the type 2^k. The new converters are arithmetic-based designs, that may be implemented using only arithmetic components without any read-only memories nor lookup tables. We tackle the problem of large modular reduction imposed by the properties of Chinese Remainder Theorem (CRT) employed in our method by calculating small correction factor in parallel with weighted sum of CRT in a set of constant multipliers followed by one two-operand modulo adder. Synthesis results show delay reduction over existing designs of up to 39.23% with area reductions of up to 28.48%.

Keywords: residue number system (RNS), Chinese Remainder Theorem, parallel architectures

1. Introduction

A residue number system (RNS) is an alternative, non-positional representation of numbers, where the calculations are performed in several independent modular *channels* [1, 32]. The calculations in each channel are performed modulo a positive integer *modulus*, and a set of pairwise prime moduli forms a *base* for the RNS. As the RNS is a non-positional number system, the arithmetic operations are performed without cross-channel interaction. Since the width of the individual channels is significantly smaller than the width of the positional datapath of comparable dynamic range, the main benefits of RNS stem from shorter propagation paths and smaller circuits, especially multipliers. On the flip side, it also requires *forward-* and *reverse conversions*.

The general architecture of the datapath with the RNS consists of the forward- and reverse converter and the set of datapath channels performing useful calculations. Since the conversions are pure overheads in the cost of the complete datapath (notably about 50% of the area of 64-bit arithmetic units for 4- and 5-moduli sets from [24]), then the cost of the converters should be minimized. On the other hand, to gain the advantage over positional datapath, the efficiency of the arithmetic circuits in the datapaths and the utilization of the available dynamic range should be maximized. Also, three key dimensions in the selection of the moduli for the moduli set are (i) the efficiency of the reverse converter, (ii) the efficiency of the implementation of datapath channels, (iii) the available bit width (memory) utilization. We will focus on the first two issues, with the memory utilization being additional benefit of arbitrary moduli selection enabled by the reverse converter. As the channels in the RNS datapath implement modular arithmetic operators, its performance and efficiency depend primarily on the implementation of the arithmetic circuits. The most efficient circuits are these for power-of-two modulus, next in a row are the circuits for $2^a - 1$, $2^a + 1$, and the others of the type $2^a \pm k$, k odd, follow. There also exist several well explored number-theoretic properties that facilitate the design of the reverse converters for the moduli sets with carefully selected special moduli [1]. However, the imposed special moduli set may severely restrain the performance of the rest of the datapath. Hence, the main motivation is to provide efficient converters for arbitrary moduli sets.

Email address: piotr.patronik@pwr.edu.pl (Piotr Patronik)

There exist four major forms of the conversion between residue and weighted representation [1]. Best known and most researched are two variants of Chinese Remainder Theorem (CRT) [32, 35] which are a modular inner product; the second is Mixed-Radix Conversion (MRC) [5, 6, 16, 21, 32], where an intermediate representation using a mixed radix is first constructed, and then the final value is calculated as an inner product using weighted representation. There were also attempts to use core functions, e.g. in [7], positional characteristics of the RNS number [2] and redundant representations [33]. These mathematical foundations are then used in the designs of dedicated hardware/software architectures for specific moduli sets. Each of the forms proposed has architectural implications: in the case of the MRC, the main obstacle is a serial order of calculations of individual mixed-radix digits, in the case of the CRT—despite its inherent parallelism—it is the necessity of the construction of large modular addition tree. Besides the most recently proposed architecture from [26] based on [35], the CRT approach enjoys the most interest when it comes to the design of the converters for arbitrary moduli sets.

Since CRT is an inner modular product with large modulus M, a major problem in its calculation with the binary representation of the components—extraction of the modulus in the summation and multiplication—has been tackled in a variety of ways. In [17, 30], the approximation of the result was proposed, which has then been extended in [20] to a fractional representation which has been optimized for the moduli of form $2^a \pm 1$ in [13], while in [12] the number of bits in fractional representation was reduced. In [35], in a scheme called New CRT-II, a tree decomposition with the modular reduction in each level of the tree has been proposed, however, it imposes a large cost of repeated modular reductions at each stage of the tree. A survey of a variety of suggestions from [3, 4, 8–10, 14, 15, 18, 22, 25, 27–29, 31, 34, 36] enable us to distinguish three basic stages of the CRT-based conversion: (i) the pre-processing step, where modular per-channel calculations are done; (ii) the addition stage, consisting of only positional constant multiplications, implemented in all recent converters as a carry-save adder (CSA) tree; (iii) the post-processing stage, in which modular reductions of the excess bits from the CSA tree are done and the final result is calculated in a two-operand modular adder.

Since modulus in its binary representation has a fixed number of bits, and CRT multiplication and subsequent accumulation in a multi-operand adder yields in wider result, the modular reduction boils down to the handling of the excess bits. In [4, 8, 14, 25, 27, 31, 36] the most-significant bits (MSB's) of CSA pair (i.e., these which weights are larger than the modulus) are reduced using a look-up table (LUT); in [9, 18, 29] least-significant bits (LSB's) from CSA are fed to LUT, and in [22] both MSB's and LSB's are reduced. However, the approach presented in [22] suggests using redundant modulus and channel. In [3, 10, 15, 34] the alternative number-theoretic approaches to generate a pair of the numbers to be added in one two-operand modular adder are suggested. Of these methods, the most efficient are these using LSB extraction since the size of the lookup tables is the smallest. However, the disadvantage of the methods presented in [9, 18] is the restriction of the moduli set without even modulus (especially the moduli in the form of 2^a). While the converter from [29] is designed to extend the former with the even moduli, it requires additional constant multiplication of the residue modulo 2^a and requires two modular reductions modulo large number.

In this paper, we provide a fully parallel reverse converter design conceived specifically for two types of the moduli sets: one consisting only of the moduli of type 2^a-1 since they have the best power/area/performance relation, and the other consisting of the moduli of the same width but very close values since they fit best to the available word size and hence provide the best memory utilization. Besides the above odd moduli, we aim at providing the converter for the moduli sets with even modulus of the type 2^k for arbitrary k, since the width of k, may be easily adjusted to match the cost/performance metrics of the even channel and the remaining odd channels. The proposed converters may be also used for any moduli sets with arbitrary pairwise prime odd moduli and one even modulus being the power of two.

Our design is architecturally the most similar to the converters presented in [8, 22, 29, 31] and in a smaller extent to [10], with the following differences and extensions:

- the presented converter does not need a redundant channel [22] specific for conversion,
- the presented converter uses LSBs unlike MSBs used in [8, 31]—thus eliminating the need for full carry-propagate addition before read-only memory (ROM) operation,

- the presented converter needs even channel on only one form (multiplied by the multiplicative inverse)—contrary to [29], where both direct and multiplied forms are needed,
- in the proposed converter, only one comparison and modulo reduction is required, contrary to the converter from [29], where two parallel modulo additions are needed (one for negative $\tilde{\varepsilon}$, and another for $\tilde{\varepsilon}$ larger than M [29]),
- the novelty is that the additional bits needed for the final CRT reduction are handled in a separate, fully arithmetic path containing only mod 2^a adders and multipliers without the need of any ROMs—unlike in [10], where this path is done in series and on the number of bits equal to the number of bits in the even channel.

This paper is organized as follows. In Sect. 2 we provide basic number-theoretic properties needed to design our converter, which we present in Sect. 3. In Sect. 4 we evaluate and compare against the other designs the complexity and performance of our proposed converter. Sect. 5 concludes the paper.

2. Preliminaries

The base for an RNS is a set of r moduli m_i, $i = \{1, \ldots, r\}$. If all moduli m_i are pairwise prime (which we will assume throughout this paper), the *dynamic range* M of the RNS is a product of r moduli m_i, i.e. $M = \prod_{i=1}^{r} m_i$. An integer x_i, $i = \{1, \ldots, r\}$, is a remainder from the division of integer X by m_i and is called a *residue*, denoted as $x_i = |X|_{m_i}$. The Chinese Remainder Theorem (CRT) states that the representation of an integer $0 \leq X < M$ in r-moduli RNS by a set of r residues $\{x_1, \ldots, x_r\}$, where $0 \leq x_i < m_i$ is unique. Moreover, given integers Y and Z, $0 \leq Y, Z < M$, represented by respective residue sets $\{y_1, \ldots, y_r\}$ and $\{z_1, \ldots, z_r\}$, an operation $\circ \in \{+, -, \times\}$ on integers X, Y, and Z such that $Z = X \circ Y$ is equivalent to the same operation performed on their respective residues and modulo respective modulus, i.e., $z_i = |x_i \circ y_i|_{m_i}$, $i = \{1, \ldots, r\}$. A modular *multiplicative inverse* of an integer a with respect to and coprime with m is such an integer b that $|ab|_m = 1$ and $b = \left|a^{-1}\right|_m$. We will also extensively use fractional notation, i.e., $b = |1/a|_m$. Consequently, when a multiplicative inverse is multiplied by integer c, it is denoted by $|cb|_m = |c \cdot 1/a|_m = |c/a|_m$, which is equivalent to $|cb|_m = \left|ca^{-1}\right|_m$ (i.e., $|c/a|_m = \left|ca^{-1}\right|_m$). Now, a cornerstone relation between an integer X and its representation with a set of residues $\{x_1, \ldots, x_r\}$ in a base $\{m_1, \ldots, m_r\}$ (i.e., for $0 \leq x_i < m_i$, $i = \{1, \ldots, r\}$) is given with CRT equation with additional variables $\hat{m}_i = \prod_{j=i, j\neq i}^{r} m_i$ as

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

As said, the reduction of the above inner product (note that it is equal at most $r \cdot M$) poses the major problem in the practical calculations of the CRT. To ease its calculation, we propose the following lemmas, whose proofs can be found in the appendix.

Lemma 1. *Let s, u, v be positive integers, q_i be odd positive pairwise prime integers, a_i be integers, $i = \{1, \ldots, s\}$, $u > v$, $p = \prod_{i=1}^{s} q_i$, $\hat{q}_i = p/q_i$. Then*

$$\left| 2^u \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{2^u \hat{q}_i} \right|_{q_i} \right|_{2^u p} = \left| 2^{u-v} \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{2^{u-v} \hat{q}_i} \right|_{q_i} + 2^{u-v} p \right| - \sum_{i=1}^{s} \frac{\left| \frac{a_i}{2^{u-v} \hat{q}_i} \right|_{q_i}}{q_i} \Bigg|_{2^v} \Bigg|_{2^u p} . \tag{2}$$

Lemma 2. *Let u, v be positive integers such that $u > v$, p be odd positive integer, a, b be integers. Then*

$$\left| 2^{u-v} p \left(|a|_{2^v} + b \right) \right|_{2^u p} = \left| 2^{u-v} p \left(|a + b|_{2^v} \right) \right|_{2^u p} . \tag{3}$$

3. Proposed Architecture

3.1. Mathematical Foundation

Assume we have an RNS composed of a set of odd moduli $\{m_1, \ldots, m_{r-1}\}$. Let $M_h = \prod_{i=1}^{r-1} m_i$ and $\hat{m}_i = M_h/m_i$, $i = \{1, \ldots, r-1\}$. In this RNS, an integer X_h is represented by a set of residues $\{x_1, \ldots, x_{r-1}\}$ using CRT as

$$X_h = \left| \sum_{i=1}^{r-1} \hat{m}_i \left| \frac{x_i}{\hat{m}_i} \right|_{m_i} \right|_{M_h}. \tag{4}$$

Now we introduce without the loss of generality $m_r = 2^k$ and rewrite (4) with residue x_r and using two integers A_1 and A_2 as

$$\begin{aligned} X &= \left| 2^k \left| \frac{X_h}{2^k} \right|_{M_h} + M_h \left| \frac{x_r}{M_h} \right|_{2^k} \right|_{2^k M_h} \\ &= \left| \sum_{i=1}^{r-1} 2^k \hat{m}_i \left| \frac{x_i}{2^k \hat{m}_i} \right|_{m_i} + M_h \left| \frac{x_r}{M_h} \right|_{2^k} \right|_{2^k M_h} \\ &= \left| \underbrace{2^k \sum_{i=1}^{r-1} \hat{m}_i \left| \frac{x_i}{2^k \hat{m}_i} \right|_{m_i}}_{A_1} + \underbrace{M_h \left| \frac{x_r}{M_h} \right|_{2^k}}_{A_2} \right|_{2^k M_h}. \end{aligned} \tag{5}$$

Next we rewrite the term A_1 from (5) with Lemma 1 as

$$A_1 = \left| 2^{k-d} \textstyle\sum_{i=1}^{r-1} \hat{m}_i \left| \frac{x_i}{2^{k-d}\hat{m}_i} \right|_{m_i} + 2^{k-d} M_h \right| - \sum_{i=1}^{r-1} \frac{\left| \frac{x_i}{2^{k-d}\hat{m}_i} \right|_{m_i}}{m_i} \Bigg|_{2^d} \Bigg|_{2^k M_h}. \tag{6}$$

As $x'_r = |x_r/M_h|_{2^k}$ is a k-bit integer we can split it into two parts, one $k-d$-bit x_q and one d-bit x_p such that (a "$\|$" is a bit vector concatenation operator)

$$\begin{aligned} A_2 &= M_h x'_r = M_h(x_p \| x_q) \\ &= 2^{k-d} M_h x_p + M_h x_q, \end{aligned} \tag{7}$$

and we have inner sums rewritten as two terms that we will then partially merge.

Following Lemma 2 we can merge the second term of (6) with the first term of (7) and obtain

$$X = \left| 2^{k-d} \textstyle\sum_{i=1}^{r-1} \hat{m}_i \left| \frac{x_i}{2^{k-d}\hat{m}_i} \right|_{m_i} + M_h x_q + 2^{k-d} M_h \right| - \sum_{i=1}^{r-1} \frac{\left| \frac{x_i}{2^{k-d}\hat{m}_i} \right|_{m_i}}{m_i} + x_p \Bigg|_{2^d} \Bigg|_{2^k M_h}. \tag{8}$$

Let $0 \leq x'_i < m_i$ ($i = \{1, \ldots, r-1\}$), $0 \leq x'_p < 2^d$ be integers. Now, the modular terms

$$x'_i = \left| x_i/(2^{k-d}\hat{m}_i) \right|_{m_i} \tag{9}$$

may be pre-calculated in a datapath channel or using LUTs at the input of the converter. Moreover, let

$$x'_p = \left| x_p - \sum_{i=1}^{r-1} x'_i/m_i \right|_{2^d}. \tag{10}$$

To simplify notation, we rewrite X from (8) using x'_i and x'_p as

$$\begin{aligned} X &= \left| 2^{k-d}\sum_{i=1}^{r-1}\hat{m}_i x'_i + M_h x_q + 2^{k-d} M_h \underbrace{\left| -\sum_{i=1}^{r-1} x'_i/m_i + x_p \right|_{2^d}}_{x'_p} \right|_{2^k M_h} \\ &= \left| 2^{k-d}\sum_{i=1}^{r-1}\hat{m}_i x'_i + M_h x_q + 2^{k-d} M_h x'_p \right|_{2^k M_h}. \end{aligned} \tag{11}$$

Analyzing the form from (11), we see that a general data flow of the conversion is just a number of constant multiplications and additions with the only modular term left in the calculation of x'_p (besides the outer modulo reduction by $2^k M_h$). We will now continue with (11) to determine the value of d.

Let us analyze the sum of the two terms, $M_h x_q$ and $2^{k-d} M_h x'_p$. Since x_q is a $k-d$-bit integer and x'_p is a d-bit integer, we will concatenate them to form one k-bit integer denoted as x''_r, i.e., $0 \le x''_r < 2^k$ and

$$x''_r = 2^{k-d} x'_p + x_q = (x'_p \| x_q). \tag{12}$$

We rewrite (11) as

$$X = \left| 2^{k-d}\sum_{i=1}^{r-1}\hat{m}_i x'_i + M_h x''_r \right|_{2^k M_h}. \tag{13}$$

Because the summation modulo $2^k M_h$ is a nontrivial, though necessary task, we will try to reduce the number of reductions so that the inner term will be smaller than $2 \cdot 2^k M_h$. Hence we have

$$0 \le 2^{k-d}\sum_{i=1}^{r-1}\hat{m}_i x'_i + M_h x''_r < 2 \cdot 2^k M_h. \tag{14}$$

In the worst case, when all variable terms assume their largest values, a rough estimation yields

$$2^{k-d}\underbrace{\sum_{i=1}^{r-1}\hat{m}_i(m_i-1)}_{\approx (r-1)M_h} + \underbrace{M_h(2^k-1)}_{\approx 2^k M_h} < 2 \cdot 2^k M_h. \tag{15}$$

Rewriting we obtain d as

$$2^{k-d}(r-1)M_h < 2^k M_h \Rightarrow d > \lceil \log(r-1) \rceil,$$

i.e., the number d of excess bits to calculate the sums in (11) depends only on the number of moduli.

3.2. Converter Implementation

In Fig. 1, we show the general architecture of our converter. In the first row of the blocks from the top, there are modular multiplications by the constants, resulting in r x'_is. Then, there is a pre-processing layer, in which d lower bits of each x'_i ($i \in [1, r-1]$) are multiplied by $\left|\frac{1}{m_i}\right|_{2^d}$ and added up modulo 2^d with d most significant bits of x'_r. The resulting d-bit result is concatenated with $k-d$ least significant bits of x'_r to form x''_r. These x'_is ($i \in [1, r-1]$) and x''_r are then multiplied by the constants $\hat{m}_i$ and M_h and added in one CSA tree. Note that the organization (the balance) of the CSA tree may be adjusted to compensate for the additional delay imposed by the calculation of the most significant bits of x''_r. As a result, from the CSA tree, we obtain two vectors X_s and X_c whose sum lies in the range from 0 to $2 \cdot 2^k M_h - 1$ (i.e., $0 \le X_c + X_s < 2 \cdot 2^k M_h$).

The sum value is then determined by considering two cases: either $0 \le X_c + X_s < 2^k M_h$ or $2^k M_h \le X_c + X_s < 2 \cdot 2^k M_h$. In the first case, the addition provides the correct result. In the second case, the result

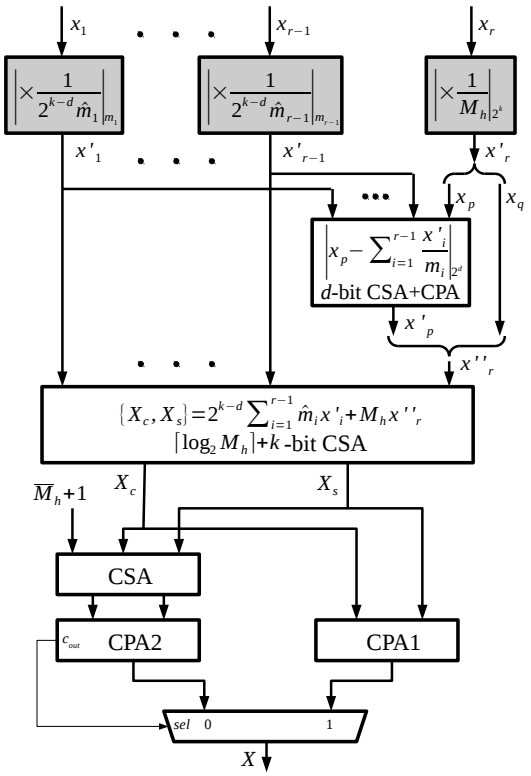

Figure 1: Block diagram of the proposed converter

of addition requires subtracting $2^k M_h$. We achieve both correction and comparison at once by preparing two alternative results—corrected and uncorrected—and then selecting the correct one by verifying whether subtracting correction yielded in a negative result, which in turn can be checked by adding two's complement value of M_h and checking the carry bit from the adder performing addition with correction value [27]. Hence, the last block of the converter is one CSA layer with a pair of regular carry-propagate adders (CPA1, CPA2), followed by the multiplexer presented in the bottom part of Fig. 1.

Alternatively, depending on the architecture of the modular adder used, we can assume the dynamic range of the sum as $3 \cdot 2^k M_h$ and rewrite (15) accordingly as

$$2^{k-d}(r-1)M_h + 2^k M_h < 3 \cdot 2^k M_h. \tag{16}$$

In this case, we have d as

$$2^{k-d}(r-1)M_h < 2 \cdot 2^k M_h \Rightarrow d > \left\lceil \log \frac{(r-1)}{2} \right\rceil.$$

Also, the use of another architecture of a modular adder (as presented e.g. in [19, 23]), enables the use of the value of d smaller by one bit with relevant change of the complexity of the rest of the converter.

3.3. Example

The RNS in this example is given by its base of $\{m_1, \ldots, m_6\} = \{23, 25, 27, 29, 31, 128\}$, hence $k = 7$ and $M = 1786492800$. We have $M_h = 23 \cdot 25 \cdot 27 \cdot 29 \cdot 31 = 13956975$ and the weights $\{\hat{m}_1, \ldots, \hat{m}_5\} = \{606825, 558279, 516925, 481275, 450225\}$. The residue values on the input which we take for this example are $\{x_1, \ldots, x_6\} = \{1, 2, 3, 4, 5, 6\}$. Since there are $r = 6$ residues, the value of d is $\lceil \log_2(6-1) \rceil = 3$. We calculate x'_is, $i = \{1, \ldots, r-1\}$, from (9) and the pair $\{x_p, x_q\}$ from (7). We have $\{x'_1, \ldots, x'_5\} = \{\left|\frac{1}{2^4 \cdot 606825}\right|_{23},$

$\left|\frac{2}{2^4 \cdot 558279}\right|_{25}$, $\left|\frac{3}{2^4 \cdot 516925}\right|_{27}$, $\left|\frac{4}{2^4 \cdot 481275}\right|_{29}$, $\left|\frac{5}{2^4 \cdot 450225}\right|_{31}\} = \{8, 18, 12, 4, 6\}$ and $(x_p \| x_q) = \left|\frac{6}{13956975}\right|_{128} = 90 = (1011010)_{\text{bin}}$. Next, we calculate x'_p from (10) as $\left|5 - (\frac{8}{23} + \frac{18}{25} + \frac{12}{27} + \frac{4}{29} + \frac{6}{31})\right|_8 = 1$ and further obtain x''_r from (12) as $(001 \| 1010)_{\text{bin}} = 26$. Now, as we have all the components, we can calculate the dot product from (13) as $2^4 \cdot 8 \cdot 606825 + 2^4 \cdot 18 \cdot 558279 + 2^4 \cdot 12 \cdot 516925 + 2^4 \cdot 4 \cdot 481275 + 2^4 \cdot 6 \cdot 450225 + 26 \cdot 13956975 = 774612102$. Since $774612102 < 1786492800$, the final reduction is not needed. Hence, we get $X = 774612102$, which may be easily verified by calculating $|774612102|_{\{23,25,27,29,31,128\}} = \{1, 2, 3, 4, 5, 6\}$.

4. Evaluation & Comparison

The analysis in this section is split into two parts: first, we create a synthetic formulation of the complexity of the converters and perform the comparison on the level of the synthetic components; then, we compare the results obtained from logic synthesis done targeting real gate library. To clarify the hardware evaluation, we first put some comments about hardware component architecture. The converters for arbitrary moduli sets based directly on CRT (specifically [10, 29] and presented by us) are composed of two major hardware components; the first being the modular multiplication by the multiplicative inverses (shown as gray blocks in Fig. 1 with the second component performing the modular sum of products. Since the first multiplication stage is constant multiplication modulo the moduli respective for the individual channels, it is customary [10, 13, 29] to apply this step as a part of a datapath without instantiating dedicated hardware multipliers at all (achieving the goal of multiplied residues via e.g. filter coefficients selected accordingly or additional multiplication instruction issued in microprocessor code [24]).

As a reference, we have selected the converters from [10–12, 29] as they are the most state-of-the-art converters for arbitrary, unrestricted moduli sets consisting of any number of odd moduli with one power-of-two modulus. All the analyzed converters are fully arithmetic designs, except [29] which is ROM based. As noted above, the architectures of our proposed converter and the converters from [10, 29] consist of a series of the constant modular multiplications of input residues by the multiplicative inverses, followed by the weighted sum of previously obtained numbers and the two- or three stage (in case of [29]) modular reduction. The converter from [12] uses weighted sum modulo power-of-two, followed by constant multiplication with truncation of the least significant bits. In the converter from [11], the weighted sum component is truncated, multiplied by a constant, and used to reduce the magnitude of the other weighted sum (two weighted sums and one constant multiplication in total are computed). These architectures are reflected by the list of the hardware components listed in Tab. 1. The table is organized into two groups, the first making up the components comprising the total area of the converters, and the second listing the components along a critical path expressing the delay. To facilitate comparison, in the table we have included only the most dominant constant multipliers; the individual adders are disregarded except the final modular adders.

From Tab. 1, it can be seen that the area differences primarily lie in an additional set of d-bit multipliers in our implementation. There is also another difference in the number of modular multipliers. Since the modular multipliers are the part of the datapath, we can omit them from the calculation with one exception—in the converters from [10, 29], one of the modular multipliers (the most efficient is even modulus 2^k) is required to be the part of the converter, since it is used for the modulo reduction. Hence, our converter enables the full exploitation of the modular multiplication in the channels, while our counterparts may take this advantage only partially. Note that the converter from [29] uses a three-operand final adder mod M while the proposed converter and [10] uses a two-operand one. The converters from [12] and [11] use different architectures, in which the multipliers have larger width, but in this case, the input residues have a direct (not scaled) form. Hence, we expect the area of the converter from [11] to be larger than the area of the one from [12], and both of them to be larger than the areas of the former ones ([10, 29]) and the proposed.

To estimate the delay, which is the sum of delays of components present on the critical path, we have assumed the same types of components as in the area part. Hence, in our converter in the critical path, there are three basic components: modular multiplication (which we show separately, but we assume to be a part of the datapath, hence in parentheses), two multipliers—one of these is d-bit wide tree and one two-operand modular adder. The modular reduction of the CRT addition in the CSA tree is done as the addition of the results of the multiplications of the d-bit results from the multipliers by the constants. Since

Table 1: Major hardware components of the evaluated converters

	Element	[10]	[12]	[11]	[29]	New
Area	Multiplier mod m_i: $\lceil \log_2 m_i \rceil \times \lceil \log_2 m_i \rceil$	$(r-1)^{\ddagger}$	-	-	$(r-1)^{\ddagger}$	(r)
	Multiplier: $\lceil \log_2 m_i \rceil \times \lceil \log_2 \hat{m}_i \rceil$	r	-	-	r	r
	Multiplier: $k \times k$	1	-	-	-	-
	Multiplier: $(k-d) \times (k-d)$	-	-	-	1	-
	Multiplier: $d \times d$	-	-	-	-	$r-1$
	Multiplier: $k \times \lceil \log_2 M_h \rceil$	1	-	-	-	-
	Multiplier: $d \times \lceil \log_2 M_h \rceil$	-	-	-	-	1
	Multiplier: $N_a \times \lceil \log_2 m_i \rceil$	-	r	-	-	-
	Multiplier: $N_a \times \lceil \log_2 M \rceil$	-	1	-	-	-
	Multiplier: $N_b \times \lceil \log_2 m_i \rceil$	-	-	r	-	-
	Multiplier: $N_b \times \lceil \log_2 M \rceil$	-	-	1	-	-
	Multiplier: $\lceil \log_2 m_i \rceil \times \lceil \log_2 M \rceil$	-	-	r	-	-
	ROM: $d \times \lceil \log_2 M_h \rceil$	-	-	-	1	-
	Adder mod M: 2-operand	1	1	1	-	1
	Adder mod M: 3-operand	-	-	-	1	-
Delay	Multiplier mod m_i: $\lceil \log_2 m_i \rceil \times \lceil \log_2 m_i \rceil$	(1)	-	-	(1)	(1)
	Multiplier: $\lceil \log_2 m_i \rceil \times \lceil \log_2 \hat{m}_i \rceil$	1	-	-	1	1
	Multiplier: $k \times k$	1	-	-	-	-
	Multiplier: $(k-d) \times (k-d)$	-	-	-	1	-
	Multiplier: $d \times d$	-	-	-	-	1
	Multiplier: $k \times \lceil \log_2 M_h \rceil$	1	-	-	-	-
	Multiplier: $N_a \times \lceil \log_2 m_i \rceil$	-	1	-	-	-
	Multiplier: $N_a \times \lceil \log_2 M \rceil$	-	1	-	-	-
	Multiplier: $N_b \times \lceil \log_2 m_i \rceil$	-	-	1	-	-
	Multiplier: $N_b \times \lceil \log_2 M \rceil$	-	-	1	-	-
	Multiplier: $\lceil \log_2 m_i \rceil \times \lceil \log_2 M \rceil$	-	-	1	-	-
	ROM: $d \times \lceil \log_2 M_h \rceil$	-	-	-	1	-
	Adder mod M: 2-operand	1	1	1	-	1
	Adder mod M: 3-operand	-	-	-	1	-

Notes:

All the multipliers listed are constant multipliers;

$d = \lceil \log_2(r-1) \rceil$;

$N_a = \lceil \log_2 \left(M \left(\sum_{i=1}^{r} m_i - r \right) \right)$ [12] ($N_a = \lceil \log_2 M + N_b \rceil$);

$N_b = \lceil \log_2 (\sum_{i=1}^{r} m_i - r)$ [11];

‡ — except multiplier mod 2^k.

d is the logarithm of the number of moduli, we assume that the addition of this result does not impact the overall delay of the tree.

The reduction of the CSA tree in the converter from [10] is performed using k-bit wide multiplier followed by k-bit carry-propagate addition, which is done in series with the CSA tree, whereas the converter from [29] contains one $(k-d)$-bit wide multiplier, one ROM and one three-operand adder mod M. Because the critical path of the proposed converter contains one d-bit wide multiplier and one two-operand adder, the overall delay with smaller k and larger d (which is the case with smaller moduli and larger moduli sets), the proposed converter may have larger delay than the one presented in [29]. The converters from [11, 12] contain two or three wide (at least $\lceil \log M \cdot \log m_i \rceil$ bits) multipliers. Also, we expect that the converters with the smallest delay will be either one of [29] or the new one, with the converter from [10] be the next one, followed by the ones from [11, 12].

To experimentally verify the claimed properties of the new converter, we have synthesized our converters and their counterparts from [10–12, 29]. All but one of these converters can be implemented using arithmetic descriptions and operators (except [29], which requires using ROM or combinational functions). The motivation to use such a level of description is that it offers us the possibility of taking advantage of the rich library of the arithmetic components (CSAs, adders, and constant multipliers), as well as automatic datapath synthesis features, both provided as the components of the synthesis tools. Moreover, modular

components needed to perform reductions modulo 2^d and 2^k in the other converters are synthesized directly by the tool.

The synthesis has been carried on a logic level using Cadence RTL Compiler (SOC_8.1USR2-s273) over low-power 65nm library from ST Micro (CMOS065-5.2.2). Power consumption was given based on the estimations provided by the synthesis tool. The converters were written in Verilog register transfer level (RTL) language and, to simulate real implementation conditions, the inputs, and outputs of the converters were connected to registers. Next, for each converter, the smallest target delay for which the logic synthesizer reported nonnegative slack was found, and this delay was reported as the minimal delay. The resulting gate-level description was then used to report the area (as a sum of the cell and routing areas), while power estimation was estimated internally by the tool using 50% switching activity (random input vectors).

As we propose the new reverse converter especially for programmable datapaths and arithmetic units, we have selected three dynamic ranges to best fit typical microprocessor and DSP word sizes, i.e., 16/32/64 bits. The number of moduli in the moduli set is based on two premises: the larger the number of moduli in the set, the channels are more fine-grained and hence are faster and more efficient, on the other hand, the more complex and less efficient is the reverse converter. Thus, the numbers of the moduli in the sets have been chosen from the range of three to nine. The specific moduli sets have been selected to meet one from two criteria: either to best fill the available dynamic range (e.g. $\{11, 13, 15, 16\}$) or to allow for the most efficient implementation of datapaths or arithmetic units (i.e., $2^a - 1/2^k$, e.g. $\{7, 31, 256\}$). The moduli sets, along with the other synthesis results, are presented in Tab. 2. The reductions of the power and area were referred to the smallest respective value of the delay, area, and power of the other converters from [10–12, 29], clearly, Red. $= \frac{\min\big([10\text{–}12,\ 29]\big)\text{–New}}{\min\big([10\text{–}12,\ 29]\big)} \cdot 100\%$.

The analysis of the delay and area from Tab. 2 resulting from the synthesis mostly confirms our observations of the arithmetic architecture sketched above. The delay reduction of our converter range from -1.69% to up to 39.23% and are accompanied by the area reductions of -11.43% to 28.48% with power reductions of -12.95% to 48.69%. The other delay and area differences are the largest with small moduli sets and wide even channels and tend to get smaller with the larger moduli sets. Our proposed converter falls at area disadvantage for some cases, although these cases occur when the other converter is slower (except one case for the 9-moduli set). This is due to both dominating effect of the positional multiplication by $\hat{m}_i$ as well as larger overhead from larger d, as from $r = 9$, it is equal to 4.

Upon closer examination of detailed synthesis results, we observe that the proposed converter has a larger delay than its closest counterpart from [29] for moduli sets with a large number of odd moduli and with small even modulus. In the experiment, the proposed converter has marginally smaller delay than the one from [29] for the 6-moduli ($\{23, 25, 27, 29, 31, 128\}$) set and the 7-bit even modulus (128)–0.27% reduction, and for the 9-moduli set ($\{107, 109, 112, 119, 121, 123, 125, 127, 256\}$) and the 8-bit even modulus (256), the new converter has the delay actually larger by 1.69%. Notably, these two specific cases have also larger power consumption. Also, the cases with a large number of moduli show relatively large area cost of an additional number of d-bit multipliers in the presented converter, which are larger (d grows with the number of moduli), and they occur in the higher number—hence, the negative reduction of the area in the cases for large moduli sets with 64-bit dynamic range, where the area of the additional multipliers adds up to the area of wide multiplication by M_h. That latter area impact is also visible for two last sets in Tab. 2, where area reduction drops from 20% to almost 0% (a similar impact can be also observed in power consumption for the same cases). For smaller dynamic ranges, there are also two cases for which the area and two cases for which the power consumption are larger than for its counterparts. These cases occur for very small converters, where the synthesizer has been able to merge and map parts of the arithmetic paths into direct logic functions.

The difference between the new converter and the one from [10] results from the two factors mentioned earlier: (i) the serial order of calculations where CSA tree result with carry-propagation (X_{medium} in [10]) is calculated in the first order, and then this value (after subtraction of the even residue) is multiplied modulo 2^k by the k-bit multiplicative inverse ($\hat{r}_n^{-1}$ in [10]), and then again by another wide constant ($\hat{M}$ in [10]), and (ii) the reduction modulo M of the CRT summation is done in a datapath of the width of the channel

with the even modulus (instead of the logarithm of the number of moduli, as in [29] and the proposed). Hence, the reduction of the delay grows with the width of the even modulus—from 6% for the 7-bit channel (the moduli set $\{23, 25, 27, 29, 31, 128\}$) to 45% (the moduli set $\{2^9 - 1, 2^{10} - 1, 2^{11} - 1, 2^{34}\}$).

Table 2: Delay, area and power of the synthesized converters

DR (bits)	Moduli Set	Delay, ps						Area, kμm^2						Power, mW					
		[10]	[12]	[11]	[29]	New	Red. %	[10]	[12]	[11]	[29]	New	Red. %	[10]	[12]	[11]	[29]	New	Red. %
16	$\{11, 13, 15, 16\}$	1988	2233	2327	1928	1753	9.08	5.44	6.33	6.09	4.32	4.65	-7.64	0.688	0.686	0.531	0.502	0.567	-12.95
	$\{7, 13, 15, 32\}$	2468	2428	2822	2022	1849	8.56	6.33	8.06	10.41	5.21	5.07	2.69	0.829	0.962	1.315	0.640	0.573	10.47
	$\{7, 31, 256\}$	2443	2172	2635	2115	1587	24.96	5.96	7.89	8.00	3.81	3.86	-1.31	0.685	0.771	0.806	0.347	0.341	1.73
	$\{3, 7, 31, 64\}$	2363	2393	2677	2070	1758	15.07	5.80	7.92	8.88	4.83	4.78	1.04	0.745	0.966	0.963	0.601	0.555	7.65
32	$\{123, 125, 127, 2048\}$	3162	2895	3318	2713	1996	26.43	13.81	23.31	24.19	12.96	11.27	13.04	2.340	3.732	3.032	2.013	1.676	16.74
	$\{251, 253, 255, 256\}$	2710	2934	3265	2488	1996	19.77	13.06	20.73	23.69	13.22	11.3	13.48	2.129	3.142	3.385	1.927	1.712	11.16
	$\{55, 59, 61, 63, 256\}$	2979	3029	3328	2446	2166	11.45	15.99	22.12	33.77	18.15	14.85	7.13	2.880	3.286	4.943	3.277	2.618	9.10
	$\{23, 25, 27, 29, 31, 128\}$	2790	3195	3372	2628	2621	0.27	14.99	22.76	31.57	17.38	12.57	16.14	2.178	4.535	4.590	2.776	2.204	-1.19
	$\{127, 255, 131072\}$	2836	2288	2709	2586	1699	25.74	13.72	14.67	19.71	9.92	7.46	24.80	1.771	1.463	2.306	1.173	0.678	42.20
	$\{15, 31, 127, 65536\}$	3414	2889	3332	3056	2139	25.96	16.62	24.96	27.47	13.33	11.41	14.40	2.897	3.862	4.246	2.008	1.482	26.20
	$\{7, 127, 255, 16384\}$	3246	3007	3303	3109	2113	29.73	15.51	24.61	26.86	12.93	10.99	15.00	2.486	4.014	3.114	1.737	1.480	14.80
	$\{7, 15, 31, 127, 8192\}$	3386	3048	3356	2770	2133	23.00	15.28	23.82	30.11	16.25	13.35	12.63	2.543	3.350	4.546	2.495	1.989	20.28
	$\{7, 31, 127, 255, 512\}$	3124	3168	3473	2601	2230	14.26	17.24	26.08	31.82	17.68	15.5	10.09	3.086	5.373	4.133	2.972	2.308	22.34
64	$\{1013, 1015, 1019, 1021, 1023, 16384\}$	3846	4110	4309	3573	2636	26.22	44.07	93.3	108.81	48.9	45.36	-2.93	11.497	27.069	22.716	10.706	9.539	10.90
	$\{499, 503, 505, 507, 509, 511, 1024\}$	3583	4000	4237	3430	2706	21.11	44.59	86.26	112.51	49.49	45.36	-1.73	10.682	27.062	24.724	11.559	10.127	5.20
	$\{233, 239, 241, 247, 251, 253, 255, 256\}$	3239	4065	4204	3085	2669	13.48	50.64	90.4	113.76	56.65	53.33	-5,31	11.953	28.146	23.785	12.597	11.146	6.75
	$\{107, 109, 113, 119, 121, 123, 125, 127, 256\}$	3303	3913	4183	2663	2708	-1.69	50.14	80.07	116.49	60.71	55.87	-11.4	11.586	23.961	23.068	13.938	11.754	-1.45
	$\{2^{15}-1, 2^{16}-1, 2^{33}\}$	3123	2515	2938	2765	1839	26.88	28.04	30.98	38.99	21.31	15.24	28.48	3.462	2.821	4.961	2.524	1.295	48.69
	$\{2^{9}-1, 2^{10}-1, 2^{11}-1, 2^{34}\}$	3819	3441	3987	3413	2074	39.23	32.81	66.38	79.38	26.94	21.43	20.45	6.991	13.251	14.934	4.884	2.889	40.85
	$\{2^{7}-1, 2^{9}-1, 2^{10}-1, 2^{11}-1, 2^{27}\}$	4111	3668	4085	3352	2368	29.36	37.73	74.92	96.29	36.93	27.69	25.02	9.563	16.958	18.501	7.700	4.989	35.21
	$\{2^{5}-1, 2^{7}-1, 2^{11}-1, 2^{12}-1, 2^{13}-1, 2^{16}\}$	3705	3904	4168	3579	2567	28.28	42.40	79.91	97.39	46.00	42.07	0.78	10.208	21.164	22.451	10.425	8.476	16.97
	$\{2^{3}-1, 2^{5}-1, 2^{7}-1, 2^{8}-1, 2^{11}-1, 2^{13}-1, 2^{17}\}$	4159	4106	4429	3731	2623	29.70	49.29	99.36	109.64	51.2	48.91	0.77	12.551	28.603	25.113	13.588	10.312	17.84

Note: Red. $= \frac{\min\big([10\text{–}12,\ 29]\big) - \text{New}}{\min\big([10\text{–}12,\ 29]\big)} \cdot 100\%$

5. Conclusions

In this paper, we propose the general method for designing the reverse converters for moduli sets consisting of arbitrary odd moduli and one modulus being a power of two. For the odd moduli, the only restriction is the natural condition of being coprime. Our approach features fully-arithmetic architecture in which we have reduced conventional CRT to the form of two modular reductions, one being a power of two with a small number of bits (depending on the logarithm of the number of moduli), and the second being the reduction from the number being smaller than twice of the modulus. The remaining component is a positional CSA tree. Thanks to its fully arithmetic form, our design may be easily implemented using well-known two's complement arithmetic circuits and synthesis methods and tools. Specifically, the converter may be simply written in the register transfer level language using only two's complement addition and constant multiplication operators. To verify these properties, we have implemented, verified, and synthesized our proposed design using typical industrial tools and libraries along with the state-of-the-art converters for arbitrary moduli sets. We have obtained delay reduction of up to 39.23%, accompanied both by area and power consumption reductions of up to 28.48% and 48.69%, respectively.

Thanks to its low delay, the proposed method seems to be particularly well-suited to implement arithmetic units of microprocessors and output conversions needed in digital filters where the delay of the proposed reverse converter may match the delay of the other arithmetic components. Other potential applications (as well as potential research directions) are fixed-point RNS representation, fast conversions needed in implementations of RNS-based elliptic curve cryptography (ECC), and fault-tolerant number representation with redundant RNS channels.

6. Acknowledgments

The author is grateful to Professor Stanislaw J. Piestrak and anonymous reviewers for their helpful suggestions.

Appendix

To provide the proof of Lemma 1, we provide additionally Properties 1 and 2 and Lemmas 3 and 4.

Property 1. *Let b be an arbitrary integer and a and m be positive integers. Then $|ab|_{am} = a\,|b|_m$.*

Property 2. *(Common factor cancellation) Let a, d be positive integers coprime with a positive integer m. Then we write*

$$\left|d \cdot \frac{1}{ad}\right|_m = \left|\frac{\not{d}}{a\not{d}}\right|_m = \left|\frac{1}{a}\right|_m.$$

Proof of Property 2. We note that

$$\left|\frac{1}{ad}\right|_m = \left|\left(\frac{1}{a}\right) \cdot \left(\frac{1}{d}\right)\right|_m,$$

where $\left|\frac{1}{a}\right|_m$ and $\left|\frac{1}{d}\right|_m$ denote integer numbers such that $0 < \left|\frac{1}{a}\right|_m, \left|\frac{1}{d}\right|_m < m$ and $\left|a \cdot \left(\frac{1}{a}\right)\right|_m = \left|d \cdot \left(\frac{1}{d}\right)\right|_m = 1$. Such numbers exist since a and d are coprime with m and their product ad is also coprime with m. Then, since modular multiplication is associative and $\left|d \cdot \left(\frac{1}{d}\right)\right|_m = 1$, we write

$$\left|d \cdot \frac{1}{ad}\right|_m = \left|d \cdot \left(\frac{1}{a}\right) \cdot \left(\frac{1}{d}\right)\right|_m = \left|\left(\frac{1}{a}\right) \cdot d \cdot \left(\frac{1}{d}\right)\right|_m = \left|\frac{\not{d}}{a\not{d}}\right|_m = \left|\frac{1}{a}\right|_m.$$

□

Lemma 3. *Let m, p be a pair of coprime positive integers, b be an integer. Then*

$$m\left|\frac{b}{m}\right|_p = \left|b + p\left|-\frac{b}{p}\right|_m\right|_{mp}. \tag{17}$$

Proof of Lemma 3. Consider an integer a such that $a = \left|\frac{b}{m}\right|_p$ so that $b = |am|_p + kp$, where k is an integer. Then, we can rewrite (17) by using a and by transforming left hand side with Property 1 to get

$$|ma|_{mp} = \left| |ma|_p + kp + p \left| -\frac{|ma|_p + kp}{p} \right|_m \right|_{mp}.$$

As there exists a nonnegative integer j such that $|ma|_p = ma - jp$, we write

$$|ma|_{mp} = \left| ma - jp + kp + p \left| -\frac{\underbrace{ma}_{=0} - jp + kp}{p} \right|_m \right|_{mp}$$

and then by dropping the term ma since $|ma|_m = 0$ and by cancelling common factors p as shown in Property 2, we get

$$\begin{aligned} |ma|_{mp} &= \left| ma - jp + kp + p \left| -\frac{(-j+k)\cancel{p}}{\cancel{p}} \right|_m \right|_{mp} \\ &= |ma - jp + kp + p\,|j-k|_m|_{mp} \\ &= |ma - p\,(j-k - |j-k|_m)|_{mp}\,. \end{aligned}$$

Since $j-k = m\,\lfloor (j-k)/m \rfloor + |j-k|_m \Rightarrow (j-k) - |j-k|_m = m\,\lfloor (j-k)/m \rfloor$, and observing that $\left| mp \left\lfloor \frac{j-k}{m} \right\rfloor \right|_{mp} = 0$, we have

$$\begin{aligned} |ma|_{mp} &= \left| ma - \underbrace{mp \left\lfloor \frac{j-k}{m} \right\rfloor}_{=0} \right|_{mp} \\ &= |ma|_{mp}. \end{aligned}$$

□

Lemma 4. *Let q_i be coprime positive integers and a_i be integers, $i = \{1, \ldots, s\}$, $p = \prod_{i=1}^{s} q_i$, $\hat{q}_i = p/q_i$, m be positive integer coprime with p. Then*

$$\left| \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{m\hat{q}_i} \right|_{q_i} \right|_p = \left| \frac{\sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{\hat{q}_i} \right|_{q_i}}{m} \right|_p.$$

Proof of Lemma 4. Let A be a nonnegative integer such that $0 \le A < p$. Then the set of integers $\{a_1, \ldots, a_s\}$ is a residue representation of A in base $\{q_1, \ldots, q_s\}$. From the properties of residue arithmetic, a set of integers $\{\left|\frac{a_1}{m}\right|_{q_1}, \ldots, \left|\frac{a_s}{m}\right|_{q_s}\}$ is a residue representation of $\left|\frac{A}{m}\right|_p$. By inserting both representations to the CRT equation (1), we get

$$\begin{aligned} \left| \frac{\underbrace{\sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{\hat{q}_i} \right|_{q_i}}_{A}}{m} \right|_p &= \left| \sum_{i=1}^{s} \hat{q}_i \left| \frac{\frac{a_i}{m}}{\hat{q}_i} \right|_{q_i} \right|_p \\ &= \left| \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{m\hat{q}_i} \right|_{q_i} \right|_p \end{aligned}$$

which is the form to be proven. □

Now we proceed to the proof of Lemma 1.

Proof of Lemma 1. To simplify notation, we rewrite the lemma for a more general case, with l and m being arbitrary positive integers coprime with p. We rewrite (2) with $l = 2^{u-v}$ and $m = 2^v$ as

$$\left| lm \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{lm\hat{q}_i} \right|_{q_i} \right|_{lmp} = \left| l \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{l\hat{q}_i} \right|_{q_i} + lp \right| - \sum_{i=1}^{s} \frac{\left| \frac{a_i}{l\hat{q}_i} \right|_{q_i}}{q_i} \Bigg|_m \Bigg|_{lmp}.$$

By using Property 1 to extract l out of modulo operation followed by division by l we get

$$\left| m \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{lm\hat{q}_i} \right|_{q_i} \right|_{mp} = \left| \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{l\hat{q}_i} \right|_{q_i} + p \right| - \sum_{i=1}^{s} \frac{\left| \frac{a_i}{l\hat{q}_i} \right|_{q_i}}{q_i} \Bigg|_m \Bigg|_{mp}.$$

Then, noting that $\left| \frac{1}{q_i} \right|_m = \left| \frac{\hat{q}_i}{p} \right|_m$, we get

$$\left| m \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{lm\hat{q}_i} \right|_{q_i} \right|_{mp} = \left| \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{l\hat{q}_i} \right|_{q_i} + p \right| - \frac{\sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{l\hat{q}_i} \right|_{q_i}}{p} \Bigg|_m \Bigg|_{mp}. \tag{18}$$

By using Property 1 to extract m out of left hand side modulo operation we get

$$\left| m \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{lm\hat{q}_i} \right|_{q_i} \right|_{mp} = m \left| \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{lm\hat{q}_i} \right|_{q_i} \right|_p$$

and then by using Lemma 4 to extract m out of the sum term we have

$$m \left| \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{lm\hat{q}_i} \right|_{q_i} \right|_p = m \left| \frac{\sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{l\hat{q}_i} \right|_{q_i}}{m} \right|_p.$$

By inserting the latter expression into the left hand side of (18) and by substituting the positive integer b as $b = \sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{l\hat{q}_i} \right|_{q_i}$ we get

$$m \left| \frac{\underbrace{\sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{l\hat{q}_i} \right|_{q_i}}_{b}}{m} \right|_p = \left| \underbrace{\sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{l\hat{q}_i} \right|_{q_i}}_{b} + p \right| - \frac{\underbrace{\sum_{i=1}^{s} \hat{q}_i \left| \frac{a_i}{l\hat{q}_i} \right|_{q_i}}_{b}}{p} \Bigg|_m \Bigg|_{mp},$$

i.e.,

$$m \left| \frac{b}{m} \right|_p = \left| b + p \left| -\frac{b}{p} \right|_m \right|_{mp}.$$

Now the equality follows directly from Lemma 3. □

Proof of Lemma 2. As above, for brevity we consider the more general case with $l = 2^{u-v}$, $m = 2^v$ (where both are coprime with odd p), hence rewriting (3) with l, m we have

$$|lp\,(|a|_m + b)|_{lmp} = |lp\,(|a+b|_m)|_{lmp}\,. \tag{19}$$

Now we observe that the left hand side sum may be rewritten as two components, of which one is in the range of $[0; m-1]$,

$$|a|_m + b = m\left\lfloor \frac{|a|_m + b}{m} \right\rfloor + |a+b|_m\,, \tag{20}$$

i.e. $0 \leq |a+b|_m < m$. Rewriting left hand side of (19) with (20) and observing that $\left|lmp\left\lfloor \frac{|a|_m+b}{m} \right\rfloor\right|_{lmp} = 0$ we obtain

$$\begin{aligned} \left|lp\left(m\left\lfloor \frac{|a|_m + b}{m} \right\rfloor + |a+b|_m\right)\right|_{lmp} &= \\ \left|\underbrace{lmp\left\lfloor \frac{|a|_m + b}{m} \right\rfloor}_{=0} + lp\,|a+b|_m\right|_{lmp} &= |lp\,|a+b|_m|_{lmp}\,. \end{aligned}$$

□

References

[1] Ananda Mohan, P.V. Residue Number Systems: Algorithms and Architectures. Switzerland: Birkhäuser, 2016.

[2] Babenko, M., Tchernykh, A., Chervyakov, N., Kuchukov, V., Miranda-López, V., Rivera-Rodriguez, R., Du, Z., Talbi, E.G. Positional characteristics for efficient number comparison over the homomorphic encryption. Programming and Computer Software 2019;45(8):532–543.

[3] Barsi, F., Pinotti, M.C. A fully parallel algorithm for residue to binary conversion. Information Process Lett 1994;50(1):1–8.

[4] Bhardwaj, M., Srikanthan, T., Clark, C.T. VLSI costs of arithmetic parallelism: a residue reverse conversion perspective. In: Proc. IEEE Symp. Comput. Arithm. 1999. p. 176–184.

[5] Bi, S., Gross, W.J. The mixed-radix Chinese Remainder Theorem and its applications to residue comparison. IEEE Trans Comput 2008;57(12):1624–1632.

[6] Bi, S., Wang, W., Al-Khalili, A. New modulo decomposed residue-to-binary algorithm for general moduli sets. In: Proc. Int. Conf. Acoust. Speech Signal Process. volume 5; 2004. p. V–141–V–144.

[7] Burgess, N. Scaled and unscaled residue number system to binary conversion techniques using the core function. In: Proc. Int. Symp. Comput. Arithm. 1997. p. 250–257.

[8] Burgess, N. Efficient RNS-to-binary conversion using high-radix SRT division. IET Proc Comput & Digit Tech 2001;148(1):49–52.

[9] Cardarilli, G.C., Re, M., Lojacono, R. Efficient modulo extraction for CRT based residue to binary converters. In: Proc. Int. Symp. Circ. & Syst. volume 3; 1997. p. 2036–2039.

[10] Chen, J.W., Yao, R.H. Efficient CRT-based residue-to-binary converter for the arbitrary moduli set. Sci China Information Sciences 2011;54(1):70–78.

[11] Chervyakov, N.I., Babenko, M., Tchernykh, A., Kucherov, N., Miranda-López, V., Cortés-Mendoza, J.M. AR-RRNS: Configurable reliable distributed data storage systems for Internet of Things to ensure security. Future Gen Comput Syst 2017;92(3):1080–1192.

[12] Chervyakov, N.I., Molahosseini, A.S., Lyakhov, P.A., Babenko, M.G., Deryabin, M.A. Residue-to-binary conversion for general moduli sets based on approximate Chinese Remainder Theorem. Int J Comput Math 2016;94(9):1833–1849.

[13] Conway, R., Nelson, J. New CRT-based RNS converter using restricted moduli set. IEEE Trans Comput 2003;52(5):572–578.

[14] Czyżak, M. An improved high-speed residue-to-binary converter based on the Chinese Remainder Theorem. Pomiary, Automatyka, Kontrola 2007;53(4):72–73.

[15] Elleithy, K.M., Bayoumi, M.A. Fast and flexible architectures for RNS arithmetic decoding. IEEE Trans Circuits Syst II: Analog and Digital Signal Proc 1992;39(4):226–235.

[16] Jullien, G.A. Residue number scaling and other operations using ROM arrays. IEEE Trans Comput 1978;100(4):325–336.

[17] Kim, J.Y., Park, K.H., Lee, H.S. Efficient residue-to-binary conversion technique with rounding error compensation. IEEE Trans Circuits Syst 1991;38(3):315–317.

[18] Lojacono, R., Cardarilli, G.C., Nannarelli, A., Re, M. Residue arithmetic techniques for high performance DSP. In: Proc. World Multi-conf. Circ. Commun. Comput. 2000. p. 314–318.
[19] Low, J.Y.S., Chang, C.H. A new approach to the design of efficient residue generators for arbitrary moduli. IEEE Trans Circuits Syst I, Reg Papers 2013;60(9):2366–2374.
[20] Meehan, S.J., O'Neil, S.D., Vaccaro, J.J. An universal input and output RNS converter. IEEE Trans Circuits Syst 1990;37(6):799–803.
[21] Miller, D.F., McCormick, W.S. An arithmetic free parallel mixed-radix conversion algorithm. IEEE Trans Circuits Syst II: Analog and Digital Signal Proc 1998;45(1):158–162.
[22] Omondi, A.R. Fast residue-to-binary conversion using base extension and the Chinese Remainder Theorem. J Circuits, Syst & Comput 2007;16(3):379–388.
[23] Patronik, P., Piestrak, S.J. Design of residue generators with CLA/compressor trees and multi-bit EAC. In: Proc. Latin America Symp. Circ. & Syst. 2017. p. 1–4.
[24] Patronik, P., Piestrak, S.J. Hardware/software approach to designing low-power RNS-enhanced arithmetic units. IEEE Trans Circuits Syst I, Reg Papers 2017;64(5):1031–1039.
[25] Pettenghi, H., Chaves, R., Sousa, L. Method to design general RNS reverse converters for extended moduli sets. IEEE Trans Circuits Syst II, Exp Briefs 2013;60(12):877–881.
[26] Pettenghi, H., Sousa, L. RNS reverse converters based on the new Chinese Remainder Theorem I. In: Proc. Int. Symp. Circ. & Syst. 2015. p. 830–833.
[27] Piestrak, S.J. Design of high-speed residue-to-binary number system converter based on Chinese Remainder Theorem. In: Proc. Int. Conf. Comput. Des. Cambridge, MA, USA; 1994. p. 508–511.
[28] Radhakrishnan, D., Srikanthan, T., Mathew, J. Using the 2^n property to implement an efficient general purpose residue-to-binary converter. In: Proc. Soft Comput. Conf. 1999. p. 183–186.
[29] Re, M., Nannarelli, A., Cardarilli, G.C., Lojacono, R. FPGA realization of RNS to binary signed conversion architecture. In: Proc. Int. Symp. Circ. & Syst. volume 4; 2001. p. 350–353.
[30] Soderstrand, M.A., Vernia, C., Chang, J.H. An improved residue number system digital-to-analog converter. IEEE Trans Circuits Syst 1983;30(12):903–907.
[31] Srikanthan, T., Bhardwaj, M., Clarke, C.T. Area-time-efficient VLSI residue-to-binary converters. IEE Proc Comput & Digit Tech 1998;145(3):229–235.
[32] Szabo, N.S., Tanaka, R.I. Residue Arithmetic and its Applications to Computer Technology. USA: McGraw Hill, 1967.
[33] Tchernykh, A., Babenko, M., Chervyakov, N., Miranda-López, V., Avetisyan, A., Drozdov, A.Y., Rivera-Rodriguez, R., Radchenko, G., Du, Z. Scalable data storage design for non-stationary IoT environment with adaptive security and reliability. IEEE Internet of Things J 2020;PP.
[34] Van Vu, T. Efficient implementations of the Chinese Remainder Theorem for sign detection and residue decoding. IEEE Trans Comput 1985;C-34(7):646–651.
[35] Wang, Y. Residue-to-binary converters based on new Chinese Remainder Theorem. IEEE Trans Circuits Syst II 2000;47(3):197–205.
[36] Zhang, C.N., Shirazi, B., Yun, D.Y.Y. Residue number conversion. In: Proc. Fall Joint Comput. Conf. 1987. p. 390–396.