electronics

MDPI

Article

The Study of Monotonic Core Functions and Their Use to Build RNS Number Comparators

Mikhail Babenko [1,2,*], Stanislaw J. Piestrak [3], Nikolay Chervyakov [1] and Maxim Deryabin [4]

[1] Department of Computational Mathematics and Cybernetics, North-Caucasus Federal University, 355017 Stavropol, Russia; k-fmf-primath@stavsu.ru
[2] Institute for System Programming of the Russian Academy of Sciences, 109004 Moscow, Russia
[3] Nanomaterials, Electronic and Living Systems Department, Université de Lorraine, CNRS, IJL, F-54000 Nancy, France; stanislaw.piestrak@univ-lorraine.fr
[4] Computing Platform Lab, Samsung Advanced Institute of Technologies, Suwon 16678, Korea; max.deriabin@samsung.com
* Correspondence: mgbabenko@ncfu.ru

Abstract: A non-positional residue number system (RNS) enjoys particularly efficient implementation of addition and multiplication, but non-modular arithmetic operations in RNS-like number comparison are known to be difficult. In this paper, a new technique for designing comparators of RNS numbers represented in an arbitrary moduli set is presented. It is based on using the core function for which it was shown that it must be monotonic to allow for RNS number comparison. The conditions of the monotonicity of the core function were formulated, which also ensured the minimal range of the core function (essential to obtain the best characteristics of the comparator). The best choice is a core function in which only one coefficient corresponding to the largest modulus is set to 1 whereas all other coefficients are set to 0. It is also shown that the already known diagonal function is nothing else but the special case of the core function with all coefficients set to 1. Performance evaluation suggests that the new comparator uses less hardware and in some cases also introduces smaller delay than its counterparts based on diagonal function. The potential applications of the new comparator include some recently developed homomorphic encryption algorithms implemented using RNS.

Keywords: core function; high-speed arithmetic; magnitude comparison; number comparison; residue number system (RNS)

Citation: Babenko, M.; Piestrak, S.J.; Chervyakov, N.; Deryabin, M. The Study of Monotonic Core Functions and Their Use to Build RNS Number Comparators. *Electronics* **2021**, *10*, 1041. https://doi.org/10.3390/electronics10091041

Academic Editor: Alessandro Savino

Received: 17 March 2021
Accepted: 23 April 2021
Published: 28 April 2021

1. Introduction

Residue number system (RNS) is a non-positional representation of integers whose main advantage over its traditional positional 2's complement counterpart is particularly efficient implementation of the basic arithmetic operations like addition and multiplication, which are executed on shorter operands by parallel independent circuits [1,2]. Unfortunately, non-modular arithmetic operations in RNS like number comparison, sign and overflow detection are known to be difficult, because they require involvement of all residues. That execution of these and some other difficult operations does not have to resort to restore RNS numbers to their positional notation involving cumbersome operations of finding the remainder of the division by a large and awkward number was first shown by Akushskii et al. [3]. They introduced the core function whose major advantage is that it offers the possibility to reduce the range within which the remainder of the division is calculated and which contains some positional information about an RNS encoded number. Nevertheless, the main disadvantage of the core function remains that most of non-modular operations are hard to implement directly [4].

The simplest approach to RNS number comparison relies on converting them to the positional representations, which are then handled using an ordinary number comparator [1]. However, using a reverse converter for RNS number comparison involves

computations modulo a very large modulus M, which is both time- and power-consuming. Nevertheless, as for extra hardware, the latter has the advantage that only the ordinary a-bit number comparator ($a = \lceil \log_2 M \rceil$) is needed, because any RNS-based processor must use the reverse converter anyway. Since [3], several attempts to design stand-alone comparators (for arbitrary RNS moduli sets) using more sophisticated approaches have been proposed [5–11]. In [5], the algorithm for comparison of signed RNS numbers, based on using the core function from [3], was proposed. Unfortunately, it requires using a redundant modulus, which must be larger than the range of the core function used. Such a solution seems impractical due to its cost, because one extra residue datapath channel must be added just to allow for number comparison (although it can serve to facilitate execution of some other difficult RNS operations as well). Faster general RNS number comparators were based on using the diagonal function [6,7] and some monotone functions proposed in [8], although the latter requires including the modulus of the form 2^k. Some limitations of the comparators of [6,7] were pointed out in [9], and they also apply to those of [8]. The comparison algorithm suggested in [10], allows one to reduce the maximum size of modulo addition from M to approximately $\sqrt{M}$, but it suffers from excessive delay compared to other methods. Finally, a new approach based on the Modified Diagonal Function (MDF) was proposed recently in [11]. It allows replacing computations modulo a large and awkward a-bit number M with significantly simpler computations involving only a power of 2 modulus 2^N, although N is always larger than a. The MDF is a kind of extension of Vu's approach for sign detection and reverse conversion [12], which also can be reduced to the computations modulo 2^N [13]. The comparator of [11] was shown superior w.r.t. both area, speed, and power consumption compared to its existing counterparts.

The importance of availability of cost-efficient and fast RNS comparison algorithms stems from the following observations. Because comparison in RNS has been considered a complex operation, the most widespread applications of RNS are usually comparison-free. The potential improvement of the efficiency of RNS comparison techniques can have significant impact on novel applications of RNS wherein comparison cannot be avoided. These include image processing [14], RNS-based convolutional neural networks [15], and RNS-based error correction codes [16]. However, cryptography and data security is the most promising emergent and dynamically developing area using RNS to improve performance of computations involving very large numbers, whose lengths are counted in thousands of bits. These include integrity verification in RNS-based verifiable secret sharing schemes [17], RNS-based algorithms in cloud computing and in edge and fog devices [16,18], and modern post-quantum homomorphic cryptography algorithms based on algebraic lattices and Ring Learning With Errors (RLWE) assumption, whose execution can be accelerated using RNS [18–24]. The magnitude comparison is required for integrity control in [20,22–24]. Because all cryptographic schemes require computations involving polynomials of very large degree and with very large coefficients, RNS representation of coefficients and operands could allow significantly increase processing performance for such schemes. Nevertheless, in this context using even modulus 2^N with $N > a$ might also have some drawbacks. Although computations modulo 2^N are more efficient than modulo any other modulus, some cryptographic applications like homomorphic encryption algorithms based on RLWE (requiring comparison of encrypted numbers in RNS) are very sensitive to memory consumption, which can put executing computations on larger N-bit rather than a-bit operands on disadvantage. This is because such an approach requires a big amount of memory to represent ciphertext, the computations modulo 2^N could not necessarily be advantageous (nevertheless, the approach based on the MDF from [11] preserves its advantages, if applied to implement RNS algorithms involving smaller dynamic range size). All computations in RLWE-based cryptosystems (both in hardware and software) are based on some Number-Theoretic Transform (NTT) [18,25], since all ciphertexts are represented as polynomials in the cyclotomic ring. However, NTT requires representing numbers in RNS moduli sets composed only of prime numbers, so that involving any computations modulo 2^N could not be supported in general.

In this paper, we will study monotonic core functions and, in particular, their properties which would make them suitable for efficient RNS number comparison. These newly discovered properties will provide the way for construction of the RNS comparison algorithm based on the core function with the smallest possible range. The general context is that all computations of a new function can be assumed as computations in the new RNS in which one of the moduli of the original RNS is excluded. It could serve as a theoretical basis for NTT-based cryptographic algorithms requiring the use of prime moduli only, aiming at accelerating such algorithms as homomorphic comparison of numbers in encrypted form.

The main contributions of this paper are twofold. One is a new systematic design approach to number comparison in RNS, which is based on the newly defined minimum-range monotonic core function and which is applicable to an arbitrary general RNS moduli set. Its major advantage is that its hardware implementation is less complex and in some cases it could be also faster than any previous similar design. Formulated will be the conditions of the monotonicity of the core function (necessary to execute comparison), which will also ensure its minimal range (essential to obtain the best characteristics of the comparator). The second is our finding that the diagonal function, previously used for number comparison in RNS and reverse conversion, is actually nothing else but the special case of the core function with all coefficients set to 1.

This paper is organized as follows. Sections 2 and 3 present the basic properties of RNS and the core functions, respectively. Section 4 details the theoretical background of the core functions, allowing for number comparison in RNS. Performance evaluation and comparison against existing circuits are provided in Section 5. Finally, some conclusions and suggestions for future research are given in Section 6.

2. Properties of RNS

The RNS is defined by the set of n pairwise prime moduli $\{m_1, m_2, \cdots, m_n\}$, which are here arranged in the increasing order (i.e., $m_1 < m_2 < \cdots < m_n$). The dynamic range of this RNS is $M = \prod_{i=1}^{n} m_i$, i.e., any a-bit integer X ($a = \lceil \log_2 M \rceil$) such that $0 \leq X < M$ can be uniquely represented in RNS as $X = \{x_1, x_2, \cdots, x_n\}$, written $X \xrightarrow{\text{RNS}} \{x_1, x_2, \cdots, x_n\}$, where $x_i = |X|_{m_i}$ (also written $x_i = X \bmod m_i$) is the a_i-bit remainder of an integer division of X by m_i ($a_i = \lceil \log_2 m_i \rceil$).

Let $M_i = M/m_i$. M_i^{-1}, the multiplicative inverse of M_i mod m_i ($0 < M_i^{-1} < m_i$) is such an integer that $|M_i^{-1} \cdot M_i|_{m_i} = 1$. To obtain the number X back from RNS to the positional form, the Chinese remainder theorem (CRT) can be used [1]

$$\begin{aligned} X &= \left| \left(M_i \left| M_i^{-1} \right|_{m_i} \right) x_i \right|_M \\ &= \left| \sum_{i=1}^{n} B_i x_i \right|_M, \end{aligned} \tag{1}$$

where the set of n CRT constants defined by

$$B_i = M_i \left| M_i^{-1} \right|_{m_i}, \; 1 \leq i \leq n, \tag{2}$$

is called the orthogonal basis [3].

3. Properties of the Core Function

The core function was defined [3] as

$$C(X) = \sum_{i=1}^{n} \omega_i \left\lfloor \frac{X}{m_i} \right\rfloor \tag{3}$$

or equivalently

$$C(X) = X\sum_{i=1}^{n}\frac{\omega_i}{m_i} - \sum_{i=1}^{n}\frac{x_i\omega_i}{m_i}, \tag{4}$$

where ω_i, $1 \le i \le n$, are integer constants which can be selected arbitrarily. For a given set of moduli, the core function can be characterized by:

- the value $C_M = C(M)$, which can be selected arbitrarily and usually such that $C_M \ll M$, and
- its range $G = C_{max} - C_{min}$, where C_{max} and C_{min} are respectively its maximum and minimum, which occur for some X_{max} and X_{min} [26].

The main attraction of the core function is that its range can vary and, similarly to C_M, it can be significantly smaller than M. Replacing X by M in Equation (4) yields

$$C_M = \sum_{i=1}^{n}\omega_i M_i. \tag{5}$$

Because $|M_j|_{m_i} = 0$ for $i \neq j$, the constant coefficients ω_i can be determined by the equation

$$\omega_i \equiv \left(C_M \cdot M_i^{-1}\right) \bmod m_i,\ 1 \le i \le n. \tag{6}$$

Note that in Equation (6), which also defines a residue class for each i, $1 \le i \le n$, the coefficients ω_i can assume both positive or negative values.

Now we will show how to obtain a practically useful formula to compute $C(X)$ for any X. As $M \overset{\text{RNS}}{\longrightarrow} \{0, 0, \cdots, 0\}$, then setting $X = M$ in Equation (4) yields

$$C_M = M\sum_{i=1}^{n}\frac{\omega_i}{m_i}. \tag{7}$$

Because Equation (3) is not practical, the value of $C(X)$ can be calculated by using remainders of X in the CRT according to Equation (1)

$$X = \sum_{i=1}^{n} B_i x_i - \alpha M,$$

where $\alpha = \lfloor X/M \rfloor$. Substituting this expression in Equation (4) and using Equation (7) leads to

$$\begin{aligned} C(X) &= \frac{C_M}{M}\left(\sum_{i=1}^{n} B_i x_i - \alpha M\right) - \sum_{i=1}^{n}\frac{x_i\omega_i}{m_i} \\ &= \sum_{i=1}^{n} x_i\left(\frac{B_i C_M}{M} - \frac{\omega_i}{m_i}\right) - \alpha C_M. \end{aligned} \tag{8}$$

Then, setting $X = B_i$ into Equation (4) with B_i from Equation (1) leads to

$$C(B_i) = \frac{B_i C_M}{M} - \sum_{j=1}^{n}\frac{\left|M_i\left|M_i^{-1}\right|_{m_i}\right|_{m_j}\omega_j}{m_j} = \frac{B_i C_M}{M} - \frac{\omega_i}{m_i}. \tag{9}$$

Now the most convenient formula for calculating $C(X)$ is obtained by substituting Equation (9) in Equation (8), which yields

$$C(X) = \left|\sum_{i=1}^{n} x_i C(B_i)\right|_{C_M}. \tag{10}$$

4. Number Comparison Using the Core Function

4.1. Monotonic Properties of the Core Function

It has been shown [5,27] that a core function, generally, is not monotonic. Now we will determine the necessary conditions for its monotonicity, which would make it useful for RNS number comparison. First, let us express $C(X-1)$ as a function of $C(X)$ for $0 < X < M$. According to Equations (4) and (7)

$$C(X) = \frac{XC_M}{M} - \sum_{i=1}^{n} \frac{x_i \omega_i}{m_i}. \tag{11}$$

Because for any x_i, $1 \le i \le n$, the following condition is met:

$$|x_i - 1|_{m_i} = \begin{cases} x_i - 1, \text{if } x_i > 0 \\ m_i - 1, \text{if } x_i = 0, \end{cases}$$

we can substitute $X-1$ in Equation (11):

$$\begin{aligned} C(X-1) &= \frac{(X-1)C_m}{M} - \sum_{i=1}^{n} \frac{|x_i - 1|_{m_i} \omega_i}{m_i} \\ &= \frac{XC_M}{M} - \frac{C_M}{M} - \sum_{\substack{i=1 \\ x_i \neq 0}}^{n} \frac{(x_i - 1)\omega_i}{m_i} - \sum_{\substack{i=1 \\ x_i = 0}}^{n} \frac{(m_i - 1)\omega_i}{m_i} \\ &= \frac{XC_M}{M} - \frac{C_M}{M} - \sum_{\substack{i=1 \\ x_i \neq 0}}^{n} \frac{x_i \omega_i}{m_i} - \sum_{\substack{i=1 \\ x_i = 0}}^{n} \frac{m_i \omega_i}{m_i} + \sum_{i=1}^{n} \frac{\omega_i}{m_i}. \end{aligned} \tag{12}$$

Applying Equations (7) and (11) to Equation (12) yields

$$\begin{aligned} C(X-1) &= \frac{XC_M}{M} - \sum_{\substack{i=1 \\ x_i \neq 0}}^{n} \frac{x_i \omega_i}{m_i} - \sum_{\substack{i=1 \\ x_i = 0}}^{n} \omega_i \\ &= C(X) - \sum_{\substack{i=1 \\ x_i = 0}}^{n} \omega_i. \end{aligned} \tag{13}$$

In other words, the value of the core function for the preceding value of X (i.e., $X-1$) is equal to the sum of the core function for X (i.e., $X-1$) and the sum of all those coefficients ω_i for which $x_i = |X|_{m_i} = 0$. The latter observation immediately leads to the following property.

Property 1. *The core function is monotonic if and only if all its coefficients ω_i are non-negative, $1 \le i \le n$.*

Thus, Property 1 undeniably limits the design space exploration range to only those core functions which could be suitable for RNS number comparison.

We will consider two special cases of the core function $C(X)$: (i) with all coefficients $\omega_i = 1$ for $1 \le i \le n$, i.e., $\{\omega_1, \ldots, \omega_{n-1}, \omega_n\} = \{1, \ldots, 1, 1\}$; and (ii) with only one coefficient set to 1, $\omega_n = 1$, corresponding to the largest modulus m_n, i.e., $\{\omega_1, \ldots, \omega_{n-1}, \omega_n\} = \{0, \ldots, 0, 1\}$.

For $\{\omega_1, \ldots, \omega_{n-1}, \omega_n\} = \{1, \ldots, 1, 1\}$, the range of the core function $C(X)$ equals to $C_M = \sum_{i=1}^{n} M_i$, i.e., it is nothing else but the sum of quotients SQ introduced in 1993 [6] also for number comparison. Consequently, the diagonal function $D(X)$ of [6] is nothing else but the special case of the core function $C(X)$ with $C_M = SQ$: the fact which has been unnoticed for several years until now.

For $\{\omega_1, \ldots, \omega_{n-1}, \omega_n\} = \{0, \ldots, 0, 1\}$ we obtain the monotonic core function with the minimum range G equal to $C_M = M_n$, which is obtained by setting $\omega_i = 0$ for $1 \leq i \leq n-1$ and $\omega_n = 1$ in Equation (3)

$$C_{m_n}(X) = \left\lfloor \frac{X}{m_n} \right\rfloor, \tag{14}$$

i.e., it is nothing else but the quotient obtained by dividing X by the largest modulus m_n and such that C_M is the smallest compared to any other $C_M = M_i$, $i < n$. Henceforth, the above function will be called the Minimum-Range Monotonic Core Function (MMCF).

Note that:

- the core function cannot be strictly monotonic because $C_M < M$, hence some other sufficiently large parameter must be used for comparison to resolve the case of $C(X) = C(Y)$ for some compared numbers X and Y; and
- for any $C_M < M_n$ the number comparison becomes impossible, because M_n is the minimal possible range for core functions with $\omega_i \geq 0$ according to Equation (5), so that the number of combinations available to differentiate numbers is only $m_n \cdot C_M < M$.

Finally, we compare our results obtained in this section against those of [8], where a new class of monotonic functions was proposed for number comparison and residue-to-binary conversion. A closer look reveals that the function $F_I(X)$ proposed in this paper (where I is a non-empty subset of indices $1 \leq i \leq n$) is nothing else but the core function with all coefficients ω_i set to 1 for any $i \in I$. Besides, the theory of the functions $F_I(X)$ presented in [8] has the following limitations.

(1) No proof is given that the function F_I is indeed monotonic. We have formally proven (Property 1) that all coefficients ω_i must be non-negative to guarantee that any core function $C(X)$ is monotonic.
(2) It is assumed that one modulus must be even 2^k, although no justification for this assumption is given. For the class of core functions considered here, the set of RNS moduli can be arbitrary (i.e., all moduli can be odd as well).
(3) No suggestions are given, how to choose the function F_I to obtain the most efficient comparator. We have shown how to construct the MMCF.

4.2. New Comparison Algorithm and Its Hardware Implementation

Because $C_{m_n}(X)$ of Equation (14) can be computed using Equation (10), the comparison of RNS integers X and Y can be formally summarized as the following algorithm.

Figure 1 shows the hardware implementation of Algorithm 1. The n-operand modular adder (MOMA) can be implemented e.g., using the methods of [28,29]. Two ordinary a_{M_n}- and a_n-bit number comparators ($a_{M_n} = \lceil \log_2 M_n \rceil$) which work in parallel as shown, can be designed e.g., according to [30] (pp. 45–47). Obviously, the basic principle of Algorithm 1 and its hardware implementation are the same as for RNS comparators proposed in [9,11]: they only differ in the modulus used by the n-operand MOMA, which generates the equivalent representation of compared numbers, sufficient to perform comparison. The modulus M_n proposed here is the smallest amongst them and this is the main contribution here.

Example 1. *Consider a sample 3-moduli set* $\{5, 7, 8\}$. *Its dynamic range is* $M = 280$ *and two basic sets of constants are:* $M_1 = 56$, $M_2 = 40$, $M_3 = 35$; *and* $|M_1^{-1}|_5 = |56|_5 = 1$, $|M_2^{-1}|_7 = |40|_7 = 3$, $|M_3^{-1}|_8 = |35|_8 = 3$. *The MMCF is obtained for* $C_M = M_3 = 35$, $\omega_1 = \omega_2 = 0$, *and* $\omega_3 = 1$. *The extra constants needed for comparison using the core function with* $C_M = M_3 = 35$ *are:* $B_1 = 56$, $B_2 = 120$, $B_3 = 105$, $C_{m_n}(B_1) = 7$, $C_{m_n}(B_2) = 15$, *and* $C_{m_n}(B_3) = 13$. *Now let us compare two numbers:* $X = 5 \xrightarrow{\text{RNS}} \{0, 5, 5\}$ *and* $Y = 6 \xrightarrow{\text{RNS}} \{1, 6, 6\}$, *for which the core functions are:*

$$C_{m_n}(5) = |0 \cdot 7 + 5 \cdot 15 + 5 \cdot 13|_{35} = |140|_{35} = 0$$
$$C_{m_n}(6) = |1 \cdot 7 + 6 \cdot 15 + 6 \cdot 13|_{35} = |175|_{35} = 0$$

Obviously, because $C_{m_n}(5) = C_{m_n}(6) = 0$*, the result of comparison of* $x_3 = 5 < y_3 = 6$ *is needed to conclude comparison.*

Algorithm 1: Comparison of RNS numbers using the core function with the minimal range.

Input: $X \xrightarrow{\text{RNS}} \{x_1, x_2, \cdots, x_n\}$, $Y \xrightarrow{\text{RNS}} \{y_1, y_2, \cdots, y_n\}$.
Output: $(1\,0\,0)$ if $X < Y$, $(0\,1\,0)$ if $X = Y$, and $(0\,0\,1)$ if $X > Y$.
Step 1. Compute $C_{m_n}(X)$ and $C_{m_n}(Y)$.
Step 2. Compare the values of $C_{m_n}(X)$ and $C_{m_n}(Y)$:
(1) If $C_{m_n}(X) < C_{m_n}(Y)$, then $X < Y$;
(2) If $C_{m_n}(X) > C_{m_n}(Y)$, then $X > Y$;
(3) If $C_{m_n}(X) = C_{m_n}(Y)$, then:
(a) if $x_n < y_n$, then $X < Y$;
(b) if $x_n > y_n$, then $X > Y$;
(c) if $x_n = y_n$, then $X = Y$.

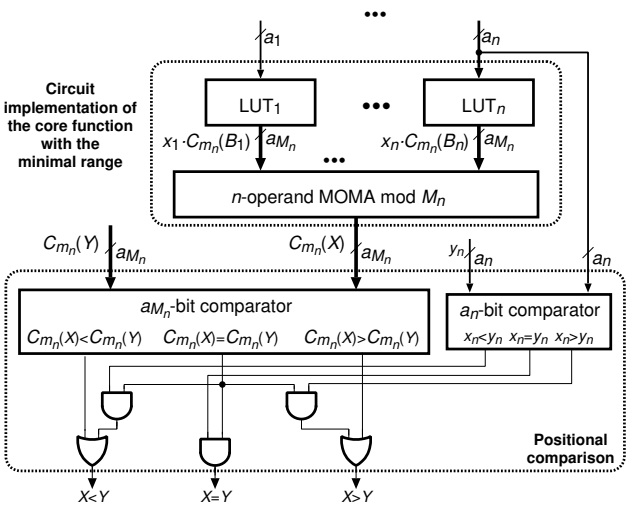

Figure 1. Hardware implementation of residue number system (RNS) numbers comparison algorithm.

5. Performance Evaluation

5.1. General Analysis

Here, we will compare the new comparators against their most efficient known counterparts based on the diagonal function [7], the modified diagonal function [11], and those based on CRT [1]. (Because in [9], it was shown that the CRT-based version Case (a) of Figure 3.1 in [7] was actually the fastest, we will consider only the latter version for comparison.) The two latter comparators have similar structures as one proposed here on Figure 1. The only differences are the following: (i) for the diagonal function and for the CRT the n-operand MOMA mod SQ and mod M is used, respectively; and (ii) for the CRT, the positional comparison consists only of a simple a-bit comparator. Because in all three cases an n-operand MOMA with varying modulus is the main building block, the impact of the size of the modulus on the hardware complexity will be analyzed, using the characteristics summarized in Table 1.

Table 1. Parameters of multi-operand modular adders (MOMAs) used in residue number system (RNS) numbers comparators.

Comparator Type	Modulus	Operand Size [bits]
CRT-based	$M = \Pi_{i=1}^{n} m_i$	$a_M = \lceil \log_2 M \rceil$
Diagonal function	$SQ = \sum_{i=1}^{n} \Pi_{j=1, j\neq i}^{n} m_j$	$a_{SQ} = \lceil \log_2 SQ \rceil$
Modified diagonal function	2^N	$N = \lceil \log_2 [SQ \cdot (m_n - 1)] \rceil$
New MMCF	$M_n = \Pi_{i=1}^{n-1} m_i$	$a_{M_n} = \lceil \log_2 M_n \rceil$

First, we compare the sizes of operands handled by RNS numbers comparators built using standard CRT-based implementation and the MMCF. By setting $M_n = M/m_n$ and taking the logarithms of both sides, we obtain $\log_2 M_n = \log_2 M - \log_2 m_n$, which leads to the following inequality.

$$a_M - \lceil \log_2 m_n \rceil \leq a_{M_n} \leq a_M - \lfloor \log_2 m_n \rfloor. \tag{15}$$

In particular, if $m_n = 2^k$ then $a_{M_n} = a_M - k$. Clearly, the bigger is the largest modulus m_n, the relatively shorter are operands of the MOMA (a_{M_n} vs. a_M) and more hardware savings ($(n-2) \cdot (a_M - a_{M_n})$ FAs less in the CSA tree alone of the MOMA) are observed compared to the CRT-based implementation. No savings are observed in the positional comparison, because the sizes of the a-bit comparator for the CRT and the total size of two comparators for the MMCF M_n- and a_n-bit are similar, although the latter requires a few extra final gates.

To compare the sizes of operands handled by comparators built using the diagonal function and the MMCF notice the following.

$$\frac{SQ}{M_n} = \sum_{i=1}^{n} \frac{m_n}{m_i} = 1 + m_n \sum_{i=1}^{n-1} \frac{1}{m_i}, \tag{16}$$

from which we get

$$\log_2 \frac{SQ}{M_n} = \log_2 SQ - \log_2 M_n = \log_2 \left(1 + m_n \sum_{i=1}^{n-1} \frac{1}{m_i} \right). \tag{17}$$

From the latter equation we obtain the lower- and upper bounds on the number of bits saved in our design

$$\left\lfloor \log_2 \left(1 + m_n \sum_{i=1}^{n-1} \frac{1}{m_i} \right) \right\rfloor \leq a_{SQ} - a_{M_n} \leq \left\lceil \log_2 \left(1 + m_n \sum_{i=1}^{n-1} \frac{1}{m_i} \right) \right\rceil. \tag{18}$$

In summary, because $n \geq 2$, then obviously $SQ = \sum_{i=1}^{n} M_i > M_n$. The resulting inequality $a_{M_n} < a_{SQ}$ implies the following general observations.

- The MOMA mod M_n operates on shorter operands than the MOMA mod SQ, so that both internal carry-save adders (CSAs) as well as the final CPAs of the former are shorter by $a_{SQ} - a_{M_n}$ bits: therefore hardware savings in adders are about $n(a_{SQ} - a_{M_n})$ FAs, i.e., they grow with both the number of moduli n and the size of the largest modulus m_n (see (18)).
- Up to $a_{SQ} - a_{M_n}$ less outputs from each of n input look-up tables (LUTs) (usually implemented using ROMs) imply less area due to connections.
- Selection of the largest modulus m_n for extra comparison to resolve the ambiguity, which occurs if $C_{m_n}(X) = C_{m_n}(Y)$, does not affect the delay of the whole RNS comparator, because it can be done in parallel with the comparison of $C_{m_n}(X)$ and $C_{m_n}(Y)$ (cf. Figure 1).

- Some delay saving can be observed for any moduli set for which $\lceil \log_2 a_{M_n} \rceil < \lceil \log_2 a_{SQ} \rceil$, because all fast carry-propagate adders (CPAs) used by the MOMA mod M_n have a few gate levels less than their counterparts used by the MOMA mod SQ. Examples of such RNS moduli sets will be given below.

As for the modified diagonal function, we will see that N is always significantly larger than a_{M_n}, which would make the new RNS comparators of interest for some cryptographic applications mentioned in the Introduction.

5.2. Complexity Analysis for Sample RNS Moduli Sets

To reveal differences between the sizes of MOMAs used in various RNS number comparators (hence, hardware savings) depending on the number of moduli n and the dynamic range DR, the parameters of several sample RNS moduli sets $S_{n,i}$ are listed in Table 2 (where n is the number of moduli and i is the number of a particular n-moduli set). To note that amongst them, the sets $S_{6,1}$ and $S_{11,1}$ are the maximal sets of the largest relatively prime moduli respectively of size $a_i \leq 4$ and $a_i \leq 5$.

Table 2. Operand sizes of the MOMAs required to implement comparators for various RNS moduli sets.

n	Moduli Set	M	SQ	M_n	a_M	a_{SQ}	a_{M_n}	SQ/M_n	N
3	$S_{3,1} = \{63, 65, 256\}$	1,048,320	36,863	4095	20	16	**12**	9.00	24
	$S_{3,2} = \{127, 129, 512\}$	8,388,096	147,455	16,383	23	18	**14**	9.00	27
4	$S_{4,1} = \{7, 15, 17, 64\}$	114,240	32,441	1785	17	15	11	*18.17*	21
	$S_{4,2} = \{15, 17, 31, 64\}$	505,920	87,713	7905	19	17	**13**	11.10	23
	$S_{4,3} = \{63, 65, 127, 512\}$	266,273,280	10,939,777	520,065	28	24	19	21.03	33
	$S_{4,4} = \{251, 253, 255, 256\}$	4.15×10^9	65,351,153	16,193,265	32	26	24	*4.04*	34
	$S_{4,5} = \{507, 509, 511, 512\}$	6.75×10^{10}	529,816,561	131,870,193	36	29	27	*4.02*	38
5	$S_{5,1} = \{5, 7, 9, 11, 13\}$	45,045	28,009	3465	16	15	12	*8.08*	19
	$S_{5,2} = \{5, 7, 9, 11, 16\}$	55,440	33,673	3465	16	16	12	9.72	19
	$S_{5,3} = \{5, 7, 9, 11, 31\}$	107,415	61,993	3465	17	16	12	*17.89*	21
	$S_{5,4} = \{7, 15, 17, 31, 32\}$	1,770,720	587,623	55,335	21	20	**16**	10.62	25
	$S_{5,5} = \{7, 15, 17, 31, 64\}$	3,541,440	1,119,911	55,335	22	21	**16**	20.24	27
	$S_{5,6} = \{7, 15, 17, 31, 128\}$	7,082,880	2,184,487	55,335	23	22	**16**	39.48	29
	$S_{5,7} = \{31, 63, 65, 127, 256\}$	4.13×10^9	310,764,191	16,122,015	32	29	24	19.28	37
6	$S_{6,1} = \{5, 7, 9, 11, 13, 16\}$	720,720	493,189	45,045	20	19	**16**	10.95	23
	$S_{6,2} = \{5, 7, 9, 11, 13, 32\}$	1,441,440	941,333	45,045	21	20	**16**	*20.90*	25
	$S_{6,3} = \{7, 9, 11, 13, 16, 17\}$	2,450,448	1,330,897	144,144	22	21	18	*9.23*	25
	$S_{6,4} = \{7, 9, 11, 13, 31, 32\}$	8,936,928	4,337,167	279,279	24	23	19	15.53	28
	$S_{6,5} = \{5, 7, 9, 11, 13, 256\}$	11,531,520	7,215,349	45,045	24	23	**16**	*160.18*	31
7	$S_{7,1} = \{5, 7, 9, 11, 13, 17, 32\}$	24,504,480	17,444,101	765,765	25	25	20	22.78	30
	$S_{7,2} = \{19, 23, 25, 27, 29, 31, 64\}$	1.70×10^{10}	4.34×10^9	265,182,525	34	33	28	16.35	38
8	$S_{8,1} = \{5, 7, 9, 11, 13, 17, 19, 32\}$	465,585,120	355,942,399	14,549,535	29	29	24	24.46	34
11	$S_{11,1} = \{7, 11, 13, 17, 19, 23, 25, 27, 29, 31, 32\}$	1.55×10^{14}	8.49×10^{13}	4.85×10^{12}	48	47	43	*17.51*	52

The comparison of the comparators based on the diagonal function against their CRT-based counterparts reveals the following:

(i) Any significant advantages of the diagonal function $a_M - a_{SQ} \geq 4$ are observed only for a few moduli sets which, additionally, are only the smallest moduli sets composed of $n = 3$ or 4 moduli: $S_{3,1}$, $S_{3,2}$, $S_{4,3}$, and $S_{4,4}$.

(ii) For $n \geq 6$, the difference (if any) between a_M and a_{SQ} becomes insignificant, which implies that the diagonal function actually does not offer any meaningful advantages over standard CRT-based implementation of the RNS numbers comparator.

The comparison of the MMCF-based comparators proposed here against their counterparts based on the diagonal function reveals the following:

(i) Should the even modulus $m_n = 2^k$ be used, for $n \geq 7$, at least $(k-1)n$ FAs are saved in our design compared to its counterpart based on the diagonal function. The inspection of the last column SQ/M_n of Table 2 reveals that the upper-bound

on the MOMA operand reduction (cf. Equation (18)) is obtained for most of the sample RNS moduli sets listed; the cases of lower-bounds are distinguished by italics.

(ii) For all moduli sets for which $\lceil \log_2 a_{SQ} \rceil \leq \lceil \log_2 a_{M_n} \rceil$ (marked in bold in the column of a_{M_n}) the new comparator is also faster, because it requires one stage less circuitry of the CPA. For instance, for $S_{6,1}$ the comparator based on the diagonal function uses the adder mod 493,189 operating on 19-bits, so that the delay of a CPA used is 12 gate delays; on the other hand, the MMCF-based comparator uses the adder mod 45,045 operating on 16-bits, so that the delay of a CPA used is 8 gate delays.

(iii) Finally, notice that the data of Table 2 show why we failed to find any closed formula to evaluate the upper-bound of $a_{SQ} - a_{M_n}$, which would be simpler than (18). Should it depend e.g., on $\lceil \log_2 m_n \rceil$ alone, notice that although for $S_{6,4} = \{7, 9, 11, 13, 31, 32\}$ we have $\lceil \log_2 m_6 \rceil = \lceil \log_2 32 \rceil = 5$ and $a_{SQ} - a_{M_n} = 4$ (which is quite close), for $S_{3,1} = \{63, 65, 256\}$ a significant difference occurs: $\lceil \log_2 m_3 \rceil = \lceil \log_2 256 \rceil = 8$ and $a_{red} = a_{SQ} - a_{M_n} = 4$.

To convey a reader with some ideas about the size difference between N of their counterparts based on the modified diagonal function [11] and the MMCF-based comparators proposed here, we have included the last column N in Table 2. The size of the even modulus N used by the modified diagonal function of [11] for all cases considered is significantly larger than a_{M_n} by from 7 up to 15 bits (for $S_{6,5}$).

5.3. Detailed Complexity Evaluation

Every RNS comparator considered here has the same general structure as shown in Figure 1, whose basic blocks are:

- $L(l, a)$—a look-up table of 2^l locations with a-bit output word length (with a time delay denoted $t_{L(l,a)}$);
- $MOMA(n, a)$—a multi-operand modular adder (MOMA) for n operands with a-bit word length (with a time delay denoted $t_{MOMA(n,a)}$); and
- $C(a)$—a binary comparator of a-bit integers (with a time delay denoted $t_{C(a)}$).

We have made the following assumptions regarding the basic building blocks and we will follow the same notation as previously used in [9]. Similarly as in [7,9], the complexity of various implementations is evaluated in terms of the number of bits for look-up tables (L), the number of full adders (FA), and time delay (TD). To evaluate the time delay, Δ the delay of a NAND gate is used as a unit, and it is assumed that $t_{FA} = t_{MUX} = 2\Delta$ and $t_{XOR} = 1\Delta$.

The delay of an l-input a-output look-up table (implemented as Read-Only Memory (ROM)) (expressed in Δ) can be approximated by using e.g., a formula from [31]

$$t_{L(l,a)} = 2 + \lceil \log_f (l/2) \rceil + \lceil \log_f 2^{(l/2)} \rceil, \tag{19}$$

for a given maximum fan-in f. In particular, for $f = 3$, we have $t_{L(l,a)} = 5\Delta$ for $4 \leq a \leq 6$, 7Δ for $7 \leq a \leq 9$, and 8Δ for $10 \leq a \leq 12$.

The delay of the fastest available implementations of positional number comparators like those from [30] (pp. 45–47) is $t_{C(a)} = 4\Delta$ for $2 \leq a \leq 4$, 8Δ for $5 \leq a \leq 24$, and 12Δ for $25 \leq a \leq 120$.

The delay of an a-bit carry-look-ahead (CLA) adder [30] is $t_{CLA(a)} = 6\Delta$ for $a \leq 8$, 8Δ for $9 \leq a \leq 16$, and 12Δ for $17 \leq a \leq 64$.

However, unlike in [9], in all complexity evaluations will be used here the same MOMA from [32], which is actually faster than the MOMA from [28] used in [9], although at a little hardware cost. For readers' convenience, its block scheme is detailed here with delay evaluations in Figure 2. It is seen that the n-operand CSA tree of this MOMA produces a pair of vectors S and C, which are partitioned into two pairs of the most significant bits (MSBs) and the least significant bits $S = \{S_H, S_L\}$ and $C = \{C_H, C_L\}$ such that $\max\{S_L + C_L\} < M$. The actual exact total numbers of the bits in S and C as well as the upper-bound on the number of MSBs which could make inputs to the MSB

converter ($\max\{h_s + h_c\}$) can be found in Table 3. The MSB converter is nothing else but an $L(h_s + h_c, 2a)$ look-up table), which generates $|S_H + C_H|_M$. The delay of the whole MOMA in which CLAs are used to implement CPAs equals to

$$t_{MOMA(n,a)} = (\theta(n) + 1)t_{FA} + t_{L(h_C+h_S,a)} + t_{XOR} + t_{CPA(a)} + t_{MUX}, \tag{20}$$

where $\theta(n)$ denotes the minimal number of stages on a CSA tree that processes n input operands, for which some sample values are listed in Table 4.

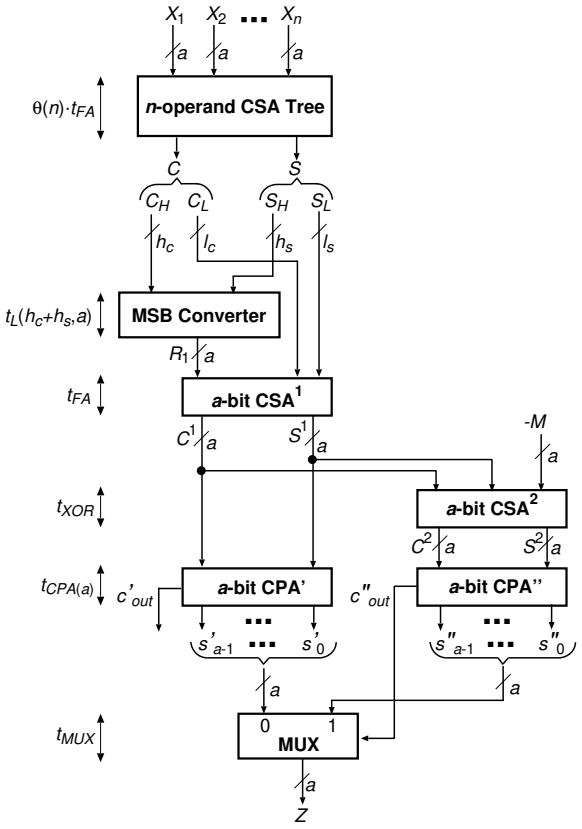

Figure 2. Detailed structure of the n-operand modular adder (MOMA) mod M.

Table 3. Outputs of the n-operand CSA tree of the MOMA from [32].

r	S	C	$\max\{h_s + h_c\}$
4	$s_a \ldots s_0$	$c_{a-1} \ldots c_0$	6
5, 6	$s_a \ldots s_0$	$c_a \ldots c_1$	7
7, 8	$s_{a+1} \ldots s_0$	$c_a \ldots c_1$	8
$9 \div 12$	$s_{a+1} \ldots s_0$	$c_{a+1} \ldots c_2$	9
13	$s_{a+2} \ldots s_0$	$c_{a+1} \ldots c_2$	10
$14 \div 18$	$s_{a+2} \ldots s_0$	$c_{a+2} \ldots c_3$	11

Table 4. The minimal number of carry-save adder (CSA) stages $\theta(k)$ required by the k-operand MOMA.

k	**3**	**4**	**5–6**	**7–9**	**10–13**	**14–19**	**20–28**
$\theta(k)$	1	2	3	4	5	6	7

Example 2. *Consider the 6-moduli set* $S_{6,1} = \{5, 7, 9, 11, 13, 16\}$, *which is the maximal set of the largest relatively prime 4-bit moduli, whose all basic parameters can be found in Table 2. Its*

dynamic range $M > 2^{20}$ is sufficient for many DSP applications. We will evaluate performance of two different comparator versions. Table 5 details the characteristics of all basic blocks used to build these comparators as well as the delays of the whole comparators. Note that the delay of both input look-up tables (LUTs) and the MOMA is counted twice, because we assume that for each pair of compared numbers their positional values or their core functions are computed serially by the same circuitry.

The delays of the MOMAs to build the comparators using the diagonal function and MMCF (calculated according to Equation (20)) are respectively as follows:

$$t_{MOMA-DF} = (\theta(6)+1)\cdot t_{FA} + t_{L(7,19)} + t_{XOR} + t_{CPA(19)} + t_{MUX} = 3\cdot 2 + 7 + 1 + 12 + 2 = 28$$

$$t_{MOMA-MMCF} = (\theta(6)+1)\cdot t_{FA} + t_{L(7,19)} + t_{XOR} + t_{CPA(16)} + t_{MUX} = 3\cdot 2 + 7 + 1 + 8 + 2 = 24$$

Clearly, the data of Table 5 show that the new comparator is faster as it introduces smaller delay by 8Δ. It is also less complex as it uses less FAs (27), HAs (3), 2:1 MUXes (3), and the final comparator shorter by 3 bits.

Table 5. Complexity estimation of comparators for a sample RNS $S_{6,1} = \{5, 7, 9, 11, 13, 16\}$.

		Using Diagonal Function [6]	**New Using MMCF**
LUTs		2 $L(3,19)$, 4 $L(4,19)$	2 $L(3,16)$, 4 $L(4,16)$
MOMA	CSA Tree	$6 \times 19 = 114$ FAs	$6 \times 16 = 96$ FAs
	MSB Converter	$L(7,19)$	$L(7,16)$
	2 CSAs	19 FAs + 19 HAs	16 FAs + 16 HAs
	2 CPAs	$2 \times 19 = 38$ FAs	$2 \times 16 = 32$ FAs
	MUX	19 2:1 MUXes	16 2:1 MUXes
Comparator		$C(19+3)+2$	$C(16)+C(4)$
Time delay		$2(t_{L(4,19)} + t_{MOMA-DF}) + 10 = 2(5+28)+10 = 76$	$2(t_{L(4,16)} + t_{MOMA-MMCF}) + 10 = 2(5+24)+10 = 68$

In general, smaller delay can be observed for any moduli set for which at least one of the below conditions holds.

(i) Each of the pair of a-bit CPAs of a MOMA is faster, which occurs if $\lceil \log_2 a_{M_n} \rceil < \lceil \log_2 a_{SQ} \rceil$. Besides the moduli set $S_{6,1}$ considered above, the inspection of the columns a_{SQ} and a_{M_n} of Table 2 reveals that several other moduli sets meet this condition.

(ii) A relatively rare case, when the final a-bit comparator is faster, occurs for most cases of practical interest e.g., if $a_{M_n} \leq 24$ and $a_{SQ} > 24$, when the delay is reduced by 4Δ. In Table 2, only the set $S_{8,1}$ meets this condition.

5.4. Final Remarks

The complexity evaluation and comparison of the new comparators against their most efficient known counterparts presented in this section allows to formulate the following conclusions. To allow number comparison, the new comparators based on the MMCF use the smallest modulus of all circuits considered. As a result, the operands added by the MOMA are also the shortest. Because the MOMA is the principal contributor to the complexity of any comparator, the presented complexity analysis proves that the new comparators are the least complex. There are also indicated some cases, when the new comparators are also faster than their counterparts. The data presented in Table 2 reveal significant impact of selecting possibly the largest modulus on improved performance of new comparators.

To maximally benefit of handling RNS data by a set of independent residue datapath channels mod m_i ($1 \leq i \leq n$), it is desirable that the latter are balanced as much as possible,

i.e., they introduce similar delay and consume similar amount of hardware resources. Particularly advantageous are moduli sets in which the largest modulus m_n is even of the type 2^p. This is because despite the modulus 2^p is larger by a few bits than all remaining odd moduli, the delay and hardware complexity of the residue datapath channel mod 2^p could still be comparable to those for the largest odd moduli. The latter has been already observed for the special moduli sets composed only of low-cost moduli of the form $2^k \pm 1$ and 2^p [33–35]. Indeed, here we have shown that selecting an even modulus 2^p as the largest one is also more advantageous to built efficient comparators proposed here and for arbitrary RNS moduli sets, including those containing other odd moduli than those of the form $2^k \pm 1$ (at least $(p \cdot (n+1)$ FAs are saved).

6. Conclusions

In this paper, a new general method for comparison of numbers represented using residue number system (RNS) was proposed. The method is based on using the core function, for which it was shown that it must be monotonic and use only non-negative coefficients to be suitable for RNS number comparison. Formulated were the conditions of the monotonicity of the core function, which also ensure the minimal range of the core function (essential to obtain the best characteristics of the comparator). It was found that the Minimum-range Monotonic Core Function (MMCF) has only one coefficient set to 1 (corresponding to the largest modulus) whereas all other coefficients are set to 0. It is also shown that the already known diagonal function, previously suggested to implement RNS numbers comparison and other RNS non-modular operations, is nothing else but the special case of the core function with all coefficients set to 1. Performance evaluation suggests that the new comparator uses less hardware and in some cases also introduces smaller delay than its counterparts based on diagonal function. It is likely that hardware savings could result in smaller power consumption as well. Some new previously undisclosed limitations of the diagonal function are also revealed. The new comparator could be of interest in all applications in which the use of the even modulus 2^N must be excluded to implement comparison, like in some recent cryptographic applications. We believe that the presented study of the monotonic core functions will deepen the understanding of their properties and hence will allow to apply the presented theory to improve implementations of other non-modular RNS operations, thus contributing to extension of the applicability of RNS in different fields.

Author Contributions: Formal analysis, M.B., M.D. and S.J.P.; funding acquisition, M.B. and M.D.; investigation, M.B., M.D. and S.J.P.; methodology, S.J.P.; project administration, M.B.; software, M.D. and S.J.P.; supervision, N.C. and M.B.; validation, M.B. and S.J.P.; writing—original draft, M.D. and S.J.P.; writing—review and editing, M.B. and S.J.P. All authors have read and agreed to the published version of the manuscript.

Funding: The research was supported by the Russian Science Foundation Grant No. 19-71-10033.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Szabó, N.S.; Tanaka, R.I. *Residue Arithmetic and Its Application to Computer Technology*; McGraw-Hill: New York, NY, USA, 1967.
2. Ananda Mohan, P.V. *Residue Number Systems: Algorithms and Architectures*; Birkhäuser: Basel, Switzerland, 2016.
3. Akushskii, I.J.; Burcev, V.M.; Pak, I.T. A new positional characteristic of non-positional codes and its application. In *Coding Theory and the Optimization of Complex Systems*; Amerbaev, V.M., Ed.; Nauka: Alma-Ata, Kazakhstan, 1977. (In Russian)
4. Abtahi, M.; Siy, P. Core function of an RNS number with no ambiguity. *Comput. Math. Appl.* **2015**, *50*, 459–470. [CrossRef]
5. Miller, D.D.; Altschul, R.E.; King, J.R.; Polky, J.N. Analysis of the residue class core function of Akushskii, Burcev, and Pak. In *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing (Paper 7–2)*; Soderstrand, M.A., Jenkins, W.K., Jullien, G.A., Taylor, F.J., Eds.; IEEE Press: New York, NY, USA, 1986; pp. 390–401.
6. Dimauro, G.; Impedovo, S.; Pirlo, G. A new technique for fast number comparison in the residue number system. *IEEE Trans. Comput.* **1993**, *42*, 608–612. [CrossRef]
7. Dimauro, G.; Impedovo, S.; Pirlo, G.; Salzo, A. RNS architectures for the implementation of the 'diagonal function'. *Inf. Process. Lett.* **2000**, *73*, 189–198. [CrossRef]

8. Pirlo, G.; Impedovo, D. A new class of monotone functions of the residue number system. *Int. J. Math. Models Methods Appl. Sci.* **2013**, *7*, 802–809.
9. Piestrak, S.J. A note on RNS architectures for the implementation of the diagonal function. *Inf. Process. Lett.* **2015**, *115*, 453–457. [CrossRef]
10. Wang, Y.; Song, X.; Aboulhamid, M. A new algorithm for RNS magnitude comparison based on new Chinese remainder theorem II. In Proceedings of the Ninth Great Lakes Symposium on VLSI (GLSVLSI), Ypsilanti, MI, USA, 4–6 March 1999; pp. 362–365.
11. Babenko, M.; Deryabin, M.; Piestrak, S.J.; Patronik, P.; Chervyakov, N.; Tchernykh, A.; Avetisyan, A. Design Method of a High-Speed RNS Number Comparator Based on a Modified Diagonal Function. *Electronics* **2020**, *9*, 1784. [CrossRef]
12. Vu, T.V. Efficient implementation of the Chinese remainder theorem for sign detection and residue decoding. *IEEE Trans. Comput.* **1985**, *C-34*, 646–651.
13. 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.* **2017**, *94*, 1833–1849. [CrossRef]
14. Chervyakov, N.I.; Lyakhov, P.A. RNS-Based Image Processing. In *Embedded Systems Design with Special Arithmetic and Number Systems*; Molahosseini, A.S., de Sousa, L.S., Chang, C.H., Eds.; Springer: Berlin/Heidelberg, Germany, 2017; Chapter 9, pp. 217–245. [CrossRef]
15. Valueva, M.; Nagornov, N.; Lyakhov, P.; Valuev, G.; Chervyakov, N. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Math. Comput. Simul.* **2020**, *177*, 232–243. [CrossRef]
16. Singh, T. Residue number system for fault detection in communication networks. In Proceedings of the 2014 International Conference on Medical Imaging, m-Health and Emerging Communication Systems (MedCom), Greater Noida, India, 7–8 November 2014; pp. 157–161.
17. Deryabin, M.; Chervyakov, N.; Tchernykh, A.; Babenko, M.; Kucherov, N.; Miranda-López, V.; Avetisyan, A. Secure verifiable secret short sharing scheme for multi-cloud storage. In Proceedings of the 2018 International Conference on High Performance Computing & Simulation (HPCS), Orleans, France, 16–20 July 2018; pp. 700–706.
18. Kim, S.; Lee, K.; Cho, W.; Nam, Y.; Cheon, J.H.; Rutenbar, R.A. Hardware Architecture of a Number Theoretic Transform for a Bootstrappable RNS-based Homomorphic Encryption Scheme. In Proceedings of the 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Fayetteville, AR, USA, 3–6 May 2020; pp. 56–64. [CrossRef]
19. Mkhinini, A.; Maistri, P.; Leveugle, R.; Tourki, R.; Machhout, M. A flexible RNS-based large polynomial multiplier for Fully Homomorphic Encryption. In Proceedings of the 2016 11th International Design & Test Symposium (IDT), Hammamet, Tunisia, 18–20 December 2016; pp. 131–136. [CrossRef]
20. Alagic, G.; Dulek, Y.; Schaffner, C.; Speelman, F. Quantum Fully Homomorphic Encryption with Verification. In Proceedings of the Advances in Cryptology—ASIACRYPT 2017, Hong Kong, China, 3–7 December 2017; Takagi, T., Peyrin, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2017; pp. 438–467.
21. Cheon, J.H.; Han, K.; Kim, A.; Kim, M.; Song, Y. A Full RNS Variant of Approximate Homomorphic Encryption. In Proceedings of the Selected Areas in Cryptography—SAC 2018, Calgary, AB, Canada, 15–17 August 2018; Volume 11349, pp. 347–368.
22. Chialva, D.; Dooms, A. Conditionals in Homomorphic Encryption and Machine Learning Applications. *arXiv* **2019**, arXiv:1810.12380.
23. El-Yahyaoui, A.; Ech-Cherif El Kettani, M.D. A Verifiable Fully Homomorphic Encryption Scheme for Cloud Computing Security. *Technologies* **2019**, *7*, 21. [CrossRef]
24. Tan, B.H.M.; Lee, H.T.; Wang, H.; Ren, S.Q.; Khin, A.M.M. Efficient Private Comparison Queries over Encrypted Databases using Fully Homomorphic Encryption with Finite Fields. *IEEE Trans. Dependable Secur. Comput.* **2020**, 1–15. [CrossRef]
25. Sinha Roy, S.; Turan, F.; Jarvinen, K.; Vercauteren, F.; Verbauwhede, I. FPGA-Based High-Performance Parallel Architecture for Homomorphic Computing on Encrypted Data. In Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), Washington, DC, USA, 16–20 February 2019; pp. 387–398. [CrossRef]
26. Burgess, N. Scaled and unscaled residue number system to binary conversion techniques using the core function. In Proceedings of the 13th IEEE Sympsoium on Computer Arithmetic (ARITH'97), Asilomar, CA, USA, 6–9 July 1997; pp. 250–257. [CrossRef]
27. Gonnella, J. The application of core functions to residue number systems. *IEEE Trans. Signal Process.* **1991**, *39*, 69–75. [CrossRef]
28. Piestrak, S.J. Design of residue generators and multioperand modular adders using carry-save adders. *IEEE Trans. Comput.* **1994**, *43*, 68–77. [CrossRef]
29. Piestrak, S.J. A high-speed realization of a residue to binary number system converter. *IEEE Trans. Circuits Syst. II* **1995**, *42*, 661–663. [CrossRef]
30. Hwang, K. *Computer Arithmetic: Principles, Architecture and Design*; Wiley: New York, NY, USA, 1979.
31. Waser, S.; Flynn, M.J. *Introduction to Arithmetic for Digital Systems Designers*; Holt, Rinehart and Winston: New York, NY, USA, 1982.
32. Piestrak, S.J. Design of high-speed residue-to-binary number system converter based on Chinese Remainder Theorem. In Proceedings of the 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors, Cambridge, MA, USA, 10–12 October 1994; pp. 508–511.
33. Conway, R.; Nelson, J. Improved RNS FIR filter architectures. *IEEE Trans. Circuits Syst. II* **2004**, *51*, 26–28. [CrossRef]
34. Piestrak, S.J.; Berezowski, K.S. Architecture of efficient RNS-based digital signal processor with very low-level pipelining. In Proceedings of the IET Irish Signals and Systems Conference (ISSC 2008), Galway, Ireland, 18–19 June 2008; pp. 127–132.
35. Patronik, P.; Piestrak, S.J. Hardware/software approach to designing low-power RNS-enhanced arithmetic units. *IEEE Trans. Circuits Syst. I Reg. Pap.* **2017**, *64*, 1031–1039. [CrossRef]