ISSN 0146-4116, Automatic Control and Computer Sciences, 2015, Vol. 49, No. 6, pp. 354–365. © Allerton Press, Inc., 2015.
Original Russian Text © N.I. Chervyakov, A.S. Molahosseini, P.A. Lyakhov, M.G. Babenko, I.N. Lavrinenko, A.V. Lavrinenko, 2015, published in Avtomatika i Vychislitel'naya Tekhnika, 2015, No. 6, pp. 47–62.

Comparison of Modular Numbers Based on the Chinese Remainder Theorem with Fractional Values

N. I. Chervyakov[a], A. S. Molahosseini[b], P. A. Lyakhov[a], M. G. Babenko[a], I. N. Lavrinenko[a], and A. V. Lavrinenko[a]

[a]*North Caucasus Federal University, Stavropol, 355009 Russia*
[b]*Department of Computer Engineering, Kerman Branch, Islamic Azad University, Kerman, Iran*
e-mail: k-fmf-primath@stavsu.ru

Received April 4, 2014

Abstract—New algorithms for determining the sign of a modular number and comparing numbers in a residue number system (RNS) have been developed using the Chinese remainder theorem with fractional values. These algorithms are based on calculations of approximate values of fractional values determined by moduli of the system. Instrumental implementations of the new algorithms are proposed and examples of their applications are given. Modeling these developments on Xilinx Kintex 7 FPGA showed that the proposed methods of decrease computational complexity of determining signs and comparing numbers in the RNS compared to that in well-known architectures based on the Chinese remainder theorem with generalized positional notation.

Keywords: residue number system, Chinese remainder theorem, modular arithmetic, positional characteristic, fractional values, approximate method, generalized positional notation

DOI: 10.3103/S0146411615060048

1. INTRODUCTION

Residue number systems (RNSs) are among promising directions of investigation in the field of computer science, which is confirmed by increasing interest of many researchers in these systems. RNSs are structures that provide parallel representation and processing of data [1–4]. Large number of publications devoted to the practical implementation of RNSs in digital filtration [5], image processing systems [6], neurocomputers [7], wireless communication networks [8], cloud computing [9] and other applications demonstrate the efficiency and usefulness of this approach.

RNS is essentially a nonpositional notation system that allows numbers of large size to be divided into several orders and accelerate computations by making them in parallel. In particular, RNSs offer the advantage of faster summation and multiplication as compared to all other notations, which accounts for the interest in using these systems in information and communication technologies that involve large volumes of computations. In addition, the use low-order numbers in RNS-based computations provides a significant reduction in energy consumption of computers [10]. This makes RNSs useful in designing computational facilities with parallel structures based on FPGAs and ASICs.

However, some operations, including inverse conversion to a positional notation form, sign determination, and the comparison and division of numbers in RNSs, encounter computational difficulties [11, 12]. The search for more effective algorithms of accomplishing these operations would provide new promising fields of practical RNS applications [13–16]. The present work proposes a new approach to accomplishing operations of determining sign and comparing numbers in RNSs, which is based on the modified Chinese reminder theorem (CRT) with fractional values.

2. INTRODUCTION TO RESIDUAL NUMBER SYSTEMS

For a fixed set of positive integers $p_1, p_2, \ldots, p_n$ called moduli, an RNS is an nonpositional notation in which any positive integer (natural number) A is represented by a set of residues obtained upon dividing the given number by these moduli: $A = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, where α_i are the minimum nonnegative residues with respect to moduli $p_1, p_2, \ldots, p_n$. Digits α_i in this representation for the selected moduli are determined as follows:

$$a_i = res + A(\mathrm{mod}\, p_i) = A - \left[\frac{A}{p_i}\right] p_i, \quad (\forall i \in [1, n]), \tag{1}$$

where $\left[\frac{A}{p_i}\right]$ are the integer divisors, and p_i are the moduli (relative primes). In the theory of numbers, the CRT ascertains that, if $\forall i \neq j\left(p_i, p_j\right) = 1$, then representation (is single-valued provided that $0 \leq A < P$, where $P = p_1 p_2 \ldots p_n = \prod_{i=1}^{n} p_i$ is the numerical representation range. In other words, there is a single number $A \in [0, P)$, for which

$$A \equiv \alpha_1 (\mathrm{mod}\, p_1); \quad A \equiv \alpha_2 (\mathrm{mod}\, p_2); \ldots; \quad A \equiv \alpha_n (\mathrm{mod}\, p_n). \tag{2}$$

For numbers in the range $[0, P)$, presented in the form of $A = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, the arithmetic operations of addition, subtraction, and multiplication are performed with mutually independent residues α_i according to the following simple rules as:

$$A \pm B = \left(\left|a_1 \pm b_1\right|_{p_1}, \ldots, \left|a_k \pm b_k\right|_{p_k}\right), \tag{3}$$

$$A \times B = \left(\left|a_1 \times b_1\right|_{p_1}, \ldots, \left|a_k \times b_k\right|_{p_k}\right). \tag{4}$$

Equations (3) and (4) reveal the parallel nature of RNS that is free of inter-order transitions. Based on the CRT, any number X can be reconstructed from its residues $\{x_1, x_2, \ldots, x_k\}$ as follows [4]:

$$X = \left|\sum_{i=0}^{k} \left|\left|P_i^{-1}\right|_{p_i} x_i\right|_{p_i} P_i\right|_P, \tag{5}$$

where $P_i = \frac{P}{p_i}$. Element $\left|P_i^{-1}\right|_{p_i}$ is the denoted multiplicative inverse of P_i with respect to p_i.

Advantages of the representation and processing of numbers in RNSs include the order of residues, which makes it possible to effectively use the table-based methods of data processing in RNSs. RNS-based computational systems are characterized by high efficiency and reliability. However, serious complications arise in the realization of nonpositional procedures, such as finding the residue of a number; determining its sign (in RNS, the sign is only implicitly defined); comparing modular numbers; detecting overflow; and operations of division, scaling, expansion, correcting errors, etc. The time of carrying out these operations can be reduced to the time required for carrying out the multiplication (along with addition, subtraction, and multiplication) and scaling (along with expansion).

Well-known algorithms of comparing modular numbers in RNSs are realized using a Mixed Radix Conversion (MRC). Number $X < P$ has the form of $\{x'_1, x'_2, \ldots, x'_k\}$, $0 < x'_i \leq p_i$ in MRC provided that

$$X = x'_1 + x'_2 p_1 + x'_3 p_1 p_2 + \ldots + x'_n \prod_{i=1}^{n-1} p_i, \tag{6}$$

where $x'_i \in [0, p_i)$ are digits of X in MRC such that

$$\begin{aligned}
&x'_1 = x_1 \bmod p_1, \\
&x'_2 = (x_2 - x'_1) c_{12} \bmod p_2, \\
&x'_3 = ((x_3 - x'_1) c_{13} - x'_2) c_{23} \bmod p_3, \\
&\ldots, \\
&x'_n = (\ldots((x_n - x'_1) c_{1n} - x'_2) c_{2n} - \ldots - x'_{n-1}) c_{n-i,n} \bmod p_n
\end{aligned} \tag{7}$$

and c_{ij} constants are multiplicative inverse elements for p_i with respect to module p_j for all $1 \leq i \leq j \leq n$ (i.e., $c_{ij} p_i = 1 \bmod p_j$ for $1 \leq i \leq n$) and can be calculated, e.g., using the Euclidean algorithm.

3. APPROXIMATE CALCULATION OF THE RELATIVE MODULAR NUMBER AS RECONSTRUCTED FROM RESIDUES

In order to simplify the comparison of modular numbers, let us consider the approximate method [15] based on taking the ratio of absolute values of these numbers to the total RNS range. Upon dividing the left and right sides of Eq. (5) by constant P corresponding to this range, we obtain an approximate value of

$$F(A) = \left|\frac{A}{P}\right|_1 = \left|\sum_{i=1}^{n} \frac{\left|P_i^{-1}\right|_{p_i}}{p_i} \alpha_i\right|_1 \approx \left|\sum_{i=1}^{n} k_i \alpha_i\right|_1, \tag{8}$$

where $k_i = \frac{\left|P_i^{-1}\right|_{p_i}}{p_i}$ are constants of the selected set, α_i are the orders of the number as represented in the RNS, and $F(A) = \left|\frac{A}{P}\right|_1 \in [0,1)$. The number of orders in the fractional part of the number is determined by the maximum possible difference between adjacent numbers. For the correct comparison of numbers A and B, it is necessary to find $F\left(\frac{A}{P}\right)$ and $F\left(\frac{B}{P}\right)$ by summing n fractional values, each one of $\left\lceil \log_2\left(P\left(-n + \sum_{i=1}^{n} p_i\right)\right)\right\rceil$ bits, where p_i are the RNS moduli, n is the number of these moduli, and $P = p_1 p_2 \ldots p_n$ is the RNS range [6, 12]. The obtained approximate values of $F\left(\frac{A}{P}\right)$ and $F\left(\frac{B}{P}\right)$ based on the RNS with fractional values will be used for correct comparison of the modular numbers, provided that the fractional vales are sufficiently accurate.

Let us assume that the RNS contains n moduli and there are n modular processors operating simultaneously in parallel (to perform summation per unit time), each module representing a residue of $b_i = [\log p_i - 1]$ bits. In order to increase the efficiency of the algorithm and convenience of the analysis of complexity, let us assume that the sizes of moduli are approximately the same, which implies that $b_i = b$.

The procedure of comparison employs LUT tables with dimensions $O(n2^b \log n)$ bit and temporal duration $O(\log n)$. For comparison, it should be noted that the conversion in MRC requires a table with dimensions $O(n^2 2^b)$ and temporal duration $O(n)$. The LUT tables contain some fractional values x rounded to the $-t$th bit, which will be denoted $[x]_{2^{-t}}$. The exact number is determined by inequalities $[x]_{2^{-t}} \le x \le [x]_{2^{-t}} + 2^{-t}$. Every operation of the determining sign and comparing numbers takes a time of $O\log(n)$ for the summation of x bit values.

4. ALGORITHM FOR COMPARING MODULAR NUMBERS

Let numbers $A = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $B = (\beta_1, \beta_2, \ldots, \beta_n)$ be set in the RNS with respect to moduli $p_1, p_2, \ldots, p_n$. To compare these numbers, it is necessary to determine the weighted value by some means from their residues. The comparison can be either exact or approximate. A straightforward comparison of modular numbers based on the passage from residues to weighted representation and the subsequent usual comparison is expensive [3, 4].

Many algorithms that can be used for an exact comparison are based on the methods employing orthogonal basis sets, the estimation of number intervals using the Euler function, universal positional characteristics represented by MRC coefficients, rank functions, number kernels, etc. These methods have been considered in [1–4], where it is shown that the necessary information is extracted in all cases from residue representation, which leads to both temporal and instrumental complexity of computing.

In traditional computers, a comparison of the absolute values of two numbers, A and B, is performed by computing the value of $A - B$ and determining the sign of this difference. In the RNS, it is not sufficient to determine the sign as $|A - B|_P$ because the values of $A - B$ can fall outside the interval $\left[-\frac{P}{2}, \frac{P}{2} - 1\right)$, which will lead to erroneous result.

Let us consider examples of correct and incorrect comparisons of modular numbers.

Example 1. Consider a set of moduli $p_1 = 2,\ p_2 = 3,\ p_3 = 5,\ p_4 = 7$ with the dynamic range $P = 2 \times 3 \times 5 \times 7 = 210$ and assume that this RNS will represent only positive numbers. The corresponding partial products are $P_1 = \frac{P}{p_1} = 105$, $P_2 = \frac{P}{p_2} = 70$, $P_3 = \frac{P}{p_3} = 42$, and $P_4 = \frac{P}{p_4} = 30$. Let us compare two integers, $A = 25$ and $B = 30$, as represented in RNS with moduli p_1, p_2, p_3, p_4 as follows: $A = (1, 1, 0, 4)$, $B = (0, 0, 0, 2)$. For this purpose, let us determine the constants $k_i = \frac{\left|P_i^{-1}\right|_{p_i}}{p_i}$. To simplify the analysis, calculations will be performed in the decimal notation

$$k_1 = \frac{\left|\frac{1}{105}\right|_2}{2} = \frac{1}{2} = 0.5; \quad k_2 = \frac{\left|\frac{1}{70}\right|_3}{3} = \frac{1}{3} \approx 0.3333; \quad k_3 = \frac{\left|\frac{1}{42}\right|_5}{5} = \frac{3}{5} = 0.6; \quad k_4 = \frac{\left|\frac{1}{30}\right|_7}{7} = \frac{4}{7} \approx 0.5714.$$

Formula (8) yields

$$F\left(\frac{A}{P}\right) \approx |1 \times 0.5 + 1 \times 0.3333 + 0 \times 0.6 + 4 \times 0.5714|_1 \approx 0.1189,$$

$$F\left(\frac{B}{P}\right) \approx |0 \times 0.5 + 0 \times 0.3333 + 0 \times 0.6 + 2 \times 0.5714|_1 \approx 0.1428.$$

Since $F\left(\frac{B}{P}\right) > F\left(\frac{A}{P}\right)$, or $0.1428 > 0.1189$, then $B > A$; indeed, $30 > 25$.

Now let us consider the case where the total working range is subdivided into two intervals, i.e., $\left[0, \frac{P}{2}\right)$ for positive numbers and $\left[-\frac{P}{2}, \frac{P}{2} - 1\right)$ for negative numbers.

Example 2. This is a variant of an incorrect comparison of modular numbers based on the sign determination. Let $A = \frac{P}{3}$ and $B = -\frac{P}{3}$, so that obviously $A > B$. Let us use (8) to determine $F\left(\frac{A}{P}\right)$ and $F\left(\frac{B}{P}\right)$ for RNS with the same moduli as in Example 1. Then, using additional code, we obtain $A = (0, 1, 0, 0)$, $B = (0, 2, 0, 0)$, and

$$F\left(\frac{A}{P}\right) \approx 1 \times 0.3333 = 0.3333; \quad F\left(\frac{B}{P}\right) \approx 2 \times 0.3333 = 0.6666.$$

It follows from this that $B > A$, but the result is incorrect since number B falls into the negative interval $\left[-\frac{P}{2}, \frac{P}{2} - 1\right)$. Therefore, this comparison yielded the incorrect result $A < B$.

For the guaranteed correct comparison, it is necessary first to check for the signs of A and B, so the algorithm of comparison is as follows:

(1) Determine the signs of A and B.

(2) If both A and B are deprived of signs, then the positive value of the difference of relative values indicates the greater number.

(3) If A and B are of the same sign, then check for $\left|\frac{A}{P} - \frac{B}{P}\right|_1$.

(4) If A and B have opposite signs, then

$0 \le \left|\frac{A}{P} - \frac{B}{P}\right|_1 < 0.5$ for $A < B$ and $0.5 \le \left|\frac{A}{P} - \frac{B}{P}\right|_1 < 1$ for $A > B$.

Thus, a correct comparison of numbers with signs requires a preliminary analysis of signs of the numbers to be compared.

As is known [1–4], the negative part of the dynamic range in the case of encrypting with additional code occurs near the upper boundary of the total dynamic range. Positive numbers of the dynamic range are mapped onto region $\left[0, \frac{P+1}{2}\right)$ for odd P and onto region $\left[0, \frac{P}{2}\right)$ for even P. Figure 1 shows mapping the dynamic range onto the corresponding region for an excessive RNS code.

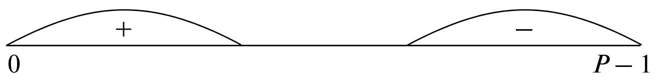

Fig. 1. Diagram of positive and negative numbers mapped onto the range of excessive RNS code.

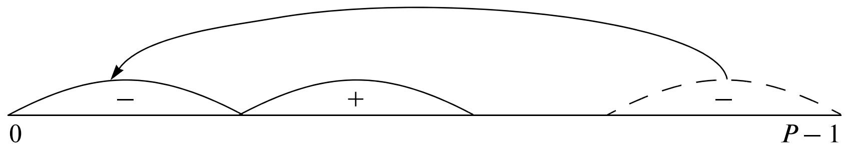

Fig. 2. Scheme of RNS polarity shift.

This circumstance can lead to errors of the comparison (as in the above example), since negative numbers fall onto the upper part of the total range and, hence all these numbers would produce errors because of expansion of the dynamic range. In order to eliminate this difficulty, it is necessary to shift the negative region by rotating the residue circle to a position indicated in Fig. 2 (where dash curve shows the region that was shifted). As a result, negative numbers will be mapped onto the beginning of the total dynamic range.

The rotation depicted in Fig. 2 is also known as a polarity shift, and can be implemented by adding constant quantity $C = \frac{P-1}{2}$ (for odd P) or $C = \frac{P}{2}$ (for even P) to each $A \in [0, P)$ prior to the comparison.

If $C_i \equiv |C|^+_{p_i}$, then the polarity shift within the RNS turns out to be a simple residue that is determined by formula $\alpha_{ic} = |\alpha_i + c_i|^+_{p_i}$, where α_{ic} denotes residue digits upon the shift.

5. DETERMINING THE SIGN OF A MODULAR NUMBER

As is known [1–4, 16], the sign of a modular number is determined using the numbers of intervals in which the given number falls, which allows this number to be estimated to within the interval size. The interval P can be divided into p_i subintervals as follows:

$$\left[j\frac{P}{p_i}, (j+1)\frac{P}{p_i}\right], \quad j = 1, 2, \ldots, p_i. \tag{9}$$

The second computer zero (machine epsilon) is set at the point $\frac{p_{n+1}}{2}\frac{P}{p_n}$. Numbers falling in the subintervals $\left[0, \frac{p_{n+1}}{2}\frac{P}{2}\right)$ and $\left[\frac{p+1}{2}\frac{P}{p_n}, P\right)$ are considered as having different (opposite) signs.

For a given representation $(\alpha_1, \alpha_2, \ldots, \alpha_n)$, the sign of this number is determined by establishing to which interval it belongs. In the case of $p_i = 2$, it is sufficient to check whether the this number belongs to the first $\left[0, \frac{P}{2}\right)$ or second $\left[\frac{P}{2}, P\right)$ half of the total range $[0, P)$. This task is solved by comparing the given representation to that of $\frac{P}{2}$, provided that $p_1 = 2$. All of the known methods implement this algorithm based on the absolute values; however, here, we prefer to use relative values. This approach significantly simplifies transformations while retaining the main functional possibilities.

Figure 3 shows a scheme of determining the sign of a modular number represented by residues with respect to moduli $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, and $p_4 = 7$. This scheme comprises input registers RG_i, $\forall i = [1\ldots4]$ for the temporal storage of the corresponding residues, relatively small look-up tables LUT_i, $\forall i = [1\ldots4]$

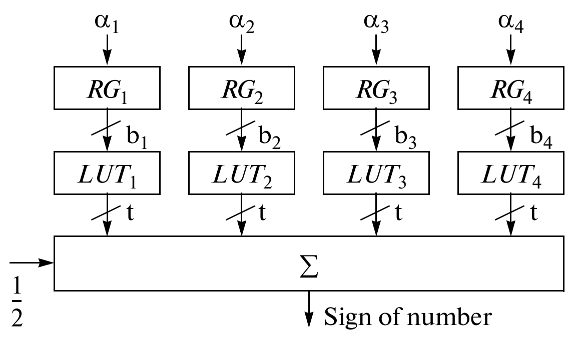

Fig. 3. Scheme of determining the sign of a number.

for the storage of products $\left|\frac{P_i^{-1}}{p_i}\right|_{p_i}\alpha_i$, and a parallel adder. The look-up tables (LUTs) are introduced in order to accelerate the multiplication operations.

The process of interval determination can be reduced to checking whether a given number belongs to one of two halves of the total range [0, P), i.e., first $\left[0, \frac{P}{2}\right)$ or second $\left[\frac{P}{2}, P\right)$, where $\frac{P}{2}$ is treated as zero. This task is solved by comparing the relative value of $\frac{A}{P}$ to that of $\frac{K}{P} = \frac{P}{2P} = \frac{1}{2}$. The initial number is classified positive for $\frac{A}{P} < \frac{1}{2}$ and negative for $\frac{A}{P} \geq \frac{1}{2}$.

The scheme operates as follows. The code of number A, for which the interval, i.e., the sign, has to be determined, enters input registers RG_i in binary code (each RNS digit is binary coded for the bus width $\lceil \log_2 p_i \rceil$ bits). The output signals enter the inputs of LUTs in which products of constants k_i and residues α_i (i.e., $\frac{\left|P_i^{-1}\right|_{p_i}}{p_i}\alpha_i$) in the form of natural binary fractions are stored in additional code with the bus width $\left\lceil \log_2 \left(P\left(\sum_{i=1}^{n} p_i - n\right)\right)\right\rceil$ bits, where p_i, $i = 1, 2, ..., n$ are RNS moduli and $P = p_1 p_2 ... p_n$ is the RNS dynamic range. In the case of Example 1, the binary fraction format is $\left\lceil \log_2 210\left(\sum_{i=1}^{4} p_i - 4\right)\right\rceil = 12$. The bus width with this format provides guaranteed and correct sign determination and a comparison of numbers in the RNS [6, 12].

Finally, the input signals of LUTs in the additional binary code enter the adder, in which constant 0.5 is written during initial setup (the additional code is used in order to replace the operation of subtraction by addition). The result of addition determines the interval (first vs. second) and, hence, the sign of the given number.

Example 3. Consider a set of moduli $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, and $p_4 = 7$ with $P = 210$. The corresponding constants k_i are $k_1 = 0.5$, $k_2 \approx 0.3333$, $k_3 = 0.6$, and $k_4 \approx 0.5714$, respectively. The given number is $A = (1, 1, 2, 0)$, and the task is to determine its sign.

Solution. The input registers are set as $RG_1 = 1$, $RG_2 = 1$, $RG_3 = 2$, $RG_4 = 0$. The register outputs are addressed at LUT storage elements that take the following values:

$$LUT_1 = 0.5, \quad LUT_2 = 0.3333 \times 1 = 0.3333, \quad LUT_3 = |0.6 \times 2|_1 = |1.2|_1 = 0.2, \quad LUT_4 = 0,$$

which enter the adder inputs. Summation of these signals yields

$$F\left(\frac{A}{P}\right) = |0.5 + 0.333 + 0.2|_1 = |0.033|_1,$$

which implies $F\left(\frac{K}{P}\right) - F\left(\frac{A}{P}\right) = 0.5 - 0.033 = 0.467$. This difference is positive, so that $A < \frac{P}{2}$, number A falls into the first interval and, hence, is positive.

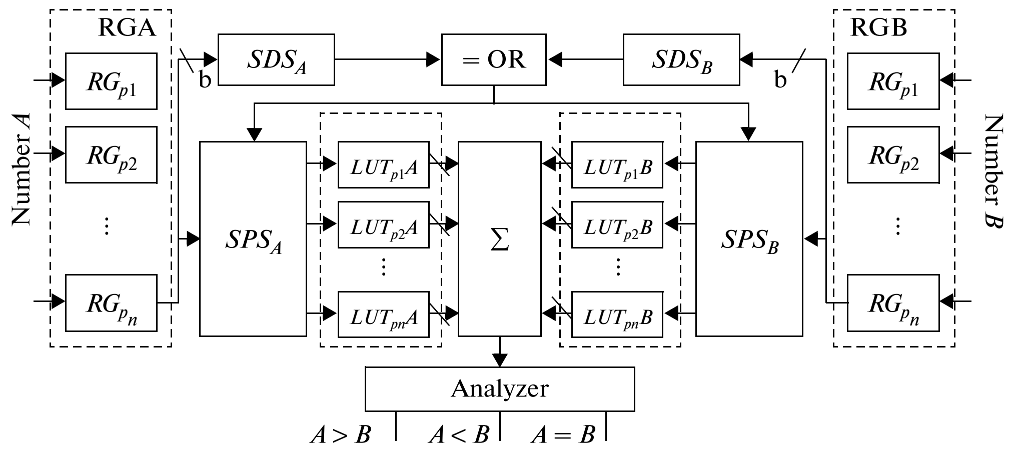

Fig. 4. Scheme of a comparator of modular numbers.

The determination of the sign requires $O(n)$ adding operations, where n is the number of RNS moduli. In order to reduce the temporal complexity to $O(\log n)$, the summation can be realized using the tree (recursive doubling) principle. For comparison, the procedure of sign determination using the Tanaka algorithm has a temporal complexity of $O(n^2)$. Therefore, the proposed scheme of sign determination reduces the instrumental and temporal complexity compared to well-known MRC systems.

6. CARRYING OUT A COMPARISON OF MODULAR NUMBERS

A comparison of two modular numbers, A and B, is aimed at determining which one is greater (smaller) or whether they are equal. The numbers of intervals in which these numbers fall can be determined by various methods [1–4]. Let number A occur in interval j_1, while number B occurs in interval j_2. Then, the operation of comparison in the case of $j_1 \neq j_2$ can be realized by simply comparing the numbers of intervals: if $j_1 > j_2$, then $A > B$; if $j_1 < j_2$, then $A < B$. An exception is presented by the case of $j_1 = j_2$. Here, finding the greater number requires determining the number j_3 of the interval in which the difference $A - B$ occurs; if $0 \le j_3 < \frac{p_{n+1}}{2}$, the difference is negative and, hence, $A < B$; if $\frac{p_{n+1}}{2} \le j_3 < p_n$, the difference is positive and, hence, $A > B$.

In the case of $A - B = 0$, the two numbers are equal in magnitude and have the same sign. The existing methods of determining intervals have been considered in [1, 2, 16]. All of these methods are based on exact calculations, encounter considerable computational difficulties, and are time consuming.

Let us consider the implementation of the proposed method of comparing numbers in RNSs based on the use of their relative values. Figure 4 shows a scheme of the modular comparison of numbers that comprises input registers *RGA* and *RGB* for storing numbers A and B to be compared, a scheme of determining signs of numbers A and B (SDS_A and SDS_B), an “exclusive or” operator, schemes of polarity shift (SPS_A and SPS_B), memory look-up tables ($LUT_{p_i}A$ and $LUT_{p_i}B$, $i = [1..n]$), an adder of LUT output signals, and a difference sign comparator (analyzer) for forming signals $A = B$, $A < B$, and $A > B$.

Initial numbers in b-bit RNS representation with respect to moduli $p_1, p_2, ..., p_n$ enter the input registers *RGA* and *RGB*. The corresponding output signals enter the schemes of sign determination (SDS_A and SDS_B). The output signals of these schemes are fed into the “excluding OR” operator. For different signs of numbers A and B, the output signal of this logical element (assigned value c_i) is fed into schemes of polarity shift (SPS_A and SPS_B), the outputs of which are addressed at the inputs of the LUTs ($LUT_{p_i}A$ and $LUT_{p_i}B$). The LUT memory elements store constants $k_i = \frac{\left|P_i^{-1}\right|_{p_i}}{p_i}\alpha_i$, $k_i = \frac{\left|P_i^{-1}\right|_{p_i}}{p_i}\beta_i$, where $\alpha_i \equiv A \bmod p_i$, $\beta_i \equiv B \bmod p_i$, $\forall i \in [1, 4]$. The output signals of $LUT_{p_i}A$ and $LUT_{p_i}B$ are fed into the adder for the weighted summation of $\frac{A}{P}$ and $\frac{B}{P}$ and the formation of a signal of the sign of difference $A - B$, which is analyzed in the comparator to yield the result ($A = B$, $A < B$, or $A > B$). Below, we consider an example of a comparison of modular numbers in this scheme.

Example 4. Let us use the RNS with set of moduli $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, and $p_4 = 7$ to compare modular numbers $A = (1, 2, 2, 3)$ and $B = (1, 2, 1, 2)$. The initial numbers are stored in registers *RGA* and *RGB*, from which they enter the schemes of determining signs (SDS_A and SDS_B) for comparison with constant $\frac{P}{2}$. Without losing generality, the calculations are assumed to be carried out in decimal notation.

The sign of number A is determined by the following calculations:

$$F\left(\frac{A}{P}\right) \approx |k_1 \times 1 + k_2 \times 2 + k_3 \times 2 + k_4 \times 3|_1 = |0.5 \times 1 + 0.3333 \times 2 + 0.6 \times 2 + 0.5714 \times 3|_1 \approx 0.0808;$$

$$F\left(\frac{K}{P}\right) - F\left(\frac{A}{P}\right) = 0.5 - 0.0808 = 0.4192.$$

The difference of these values is positive, which implies that $F\left(\frac{A}{P}\right) < F\left(\frac{K}{P}\right)$ and $A < K$, so that number A falls into the first interval and, hence, is positive.

Analogous calculations determine the sign of B as follows:

$$F\left(\frac{B}{P}\right) \approx |k_1 \times 1 + k_2 \times 2 + k_3 \times 1 + k_4 \times 2|_1 = |0.5 \times 1 + 0.3333 \times 2 + 0.6 \times 1 + 0.5714 \times 2|_1 \approx 0.9094;$$

$$F\left(\frac{K}{P}\right) - F\left(\frac{B}{P/2}\right) = 0.5 - 0.9094 = -0.4094.$$

The difference between these values is negative, which implies that number B falls into the second interval and, hence, is negative.

The results of determining the signs of numbers A and B are fed into the "excluding OR" element, the output signal of which enters the schemes of polarity shift (SPS_A and SPS_B), the outputs of which yield the values

$$A = (1, 2, 2, 3) + (1, 0, 0, 0) = (0, 2, 2, 3) \text{ and } B = (1, 2, 1, 2) + (1, 0, 0, 0) = (0, 2, 1, 2),$$

respectively. The output signals of the polarity shift schemes are addressed at the inputs of *LUT*s ($LUT_{p_i}A$ and $LUT_{p_i}B$), which select the constants $k_i = \frac{\left|P_i^{-1}\right|_{p_i}}{p_i}\alpha_i$. For the given example, the corresponding values are as follows:

$$LUT_{p_i}A\colon\ (0; 0.3333 \times 2; 0.6 \times 2; 0.5714 \times 3);$$

$$LUT_{p_i}B\colon\ (0; 0.3333 \times 2; 0.6 \times 1; 0.5714 \times 2).$$

The output signals of $LUT_{p_i}A$ and $LUT_{p_i}B$ enter the adder and are summed to yield:

$$F\left(\frac{A_i}{P}\right) - F\left(\frac{B_i}{P}\right) \approx |0 + 0.3333 \times 2 + 0.6 \times 2 + 0.5714 \times 3|_1 - |0 + 0.3333 \times 2 + 0.6 \times 1 + 0.5714 \times 2|_1 = 0.1724.$$

The difference is positive, which implies that $A > B$, which is in agreement with the fact that $A = 17$ and $B = -19$.

The results of adding are analyzed in the comparator, which yields the decisions that:

—If the difference is 0, then $A = B$,

—If the difference is positive, then $A > B$,

—If the difference is negative, then $A < B$.

7. SIMULATION OF THE PROPOSED SCHEME OF MODULAR COMPARISON

The proposed scheme of comparison was simulated using algorithms for reconstructing the weighted values of modular numbers based on (i) the Chinese remainder theorem with a classical orthogonal basis set (CRTcl), (ii) the Mixed Radix Conversion (MRC), and (iii) the Chinese remainder theorem with fractional quantities (CRTfr).

Table 1. Kintex 7 XC7K70T packaging

Slice logic items	Available
Number of Slice Registers	82000
Number of Slice LUTs	41000
Number of Slices	10250
Number of IOBs	300
Number of DSP48E1s	240

Table 2. FPGA board performance necessary for realization of the proposed algorithms at a fixed number of moduli ($n = 4$)

Set of moduli	19, 23, 29, 31		1009, 1013, 1019, 1021		32713, 32717, 32719, 32749		1048549, 1048559, 1048571, 1048573	
Range bit count	19		40		60		80	
Modulus size (bit)	5		10		15		20	
	Slices	DSP48E1	Slices	DSP48E1	Slices	DSP48E1	Slices	DSP48E1
CRTfr	13	7	77	29	118	42	236	69
CRTcl	92	4	352	16	706	20	1133	32
MRC	54	9	224	13	576	17	1108	24

Table 3. FPGA board performance necessary for realization of the proposed algorithms with various numbers of 6-bit moduli

Set of moduli	47, 53, 59, 61		41, 43, 47, 53, 59, 61		31, 37, 41, 43, 47, 53, 59, 61	
Range bit count	24		34		45	
Number of moduli	4		6		8	
	Slices	DSP48E1	Slices	DSP48E1	Slices	DSP48E1
CRTfr	17	10	46	30	129	43
CRTcl	117	4	213	12	285	32
MRC	75	11	164	16	191	45

The modeling was performed using Xilinx Kintex 7 XC7K70T FPGA (Table 1) based on ISE Design Suit 4.7 WebPack. The main criteria used to assess the quality of algorithms were device utilization and the final total delay of the algorithm. The device utilization was characterized by the number of slices that represent the base computing units of Xilinx Series 7 devices. In addition, we have studied variants of algorithm realization with and without using DSP48E1 slices intended for optimizing special-structure algorithms.

The results of simulations are summarized in Tables 2 and 3 and illustrated in Figs. 5 and 6. A comparison of the well-known classical reconstruction methods based on CRTcl and MRC shows that MRC favors higher device utilization and requires a smaller chip area, while CRTcl ensures lower delay. The

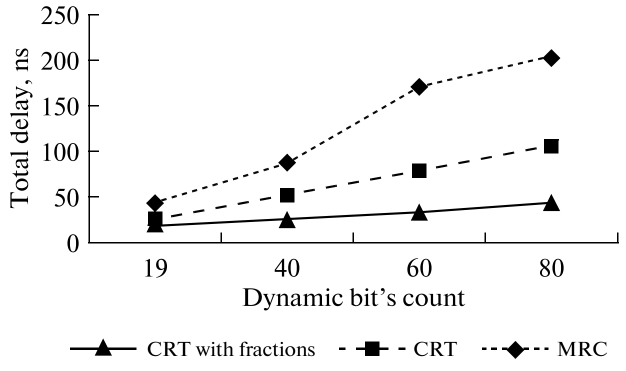

Fig. 5. Plots of the total delay of algorithm vs. modulus bit count for DSP48E1 slices with four moduli.

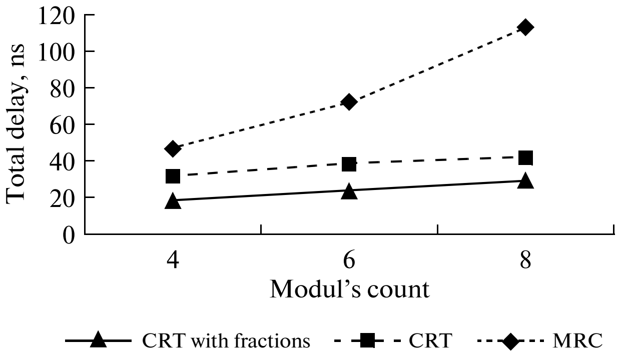

Fig. 6. Plots of the total delay of algorithm vs. number of 6-bit moduli in DSP48E1 slices.

MRC-based method involves a greater number of small-digit operations. A longer pathway of the signal from input to output, related to the greater number of operations, makes this method the slowest in this comparative simulation. Compared to CRTcl and MRC based methods, the proposed CRTfr exhibited both the lowest area requirements for the chip and shortest delay.

Let us also consider the issue of minimal RNS moduli sets necessary to cover the ranges of computing devices using maximum eight-bit buses. Table 4 presents different variants of RNS moduli sets that allow covering 8-, 16-, and 32-bit ranges.

Figure 7 shows the total delay and device utilization for all algorithms under comparison with moduli sets from Table 4. Note that the proposed method exhibits the best results despite the fact that computations exceed the range bit count. For example, in the 32-bit range, the exact calculation of a positional number will require computing up to 42-bit numbers.

The use of DSP48E1digital signal processing slices on the board allows the useful area occupied by the algorithm to be significantly reduced (Fig. 8b), but this is accompanied by a decrease in the speed of algorithm operation (Fig. 8a). In different practical situations, this circumstance can be used to optimize the desired process characteristic.

The experimental data show the efficiency of the proposed method from the standpoint of algorithm operation speed, which is achieved due to the absence of calculations of the residues of division and a small number of operations. In addition, for all sets of RNS moduli, this method requires a minimum amount of computational resources as compared to the two well-known methods.

8. CONCLUSIONS

We have proposed a procedure for representing and processing modular numbers using relative fractional values, which are used to determine the signs of modular numbers and comparing them in the RNS. The basic operation in comparing numbers is their weighted representation and summation of fractional values, which allows the instrumental and temporal complexity to be reduced as compared the well-

Table 4. Minimal RNS moduli sets covering computer ranges

Computer range	Set of moduli	RNS range	Fractional number size (bit)
$2^8 = 256$	7, 37	259	14
$2^{16} = 65536$	11, 59, 101	65549	24
$2^{32} = 4294967296$	19, 67, 89, 167, 227	4294975973	42

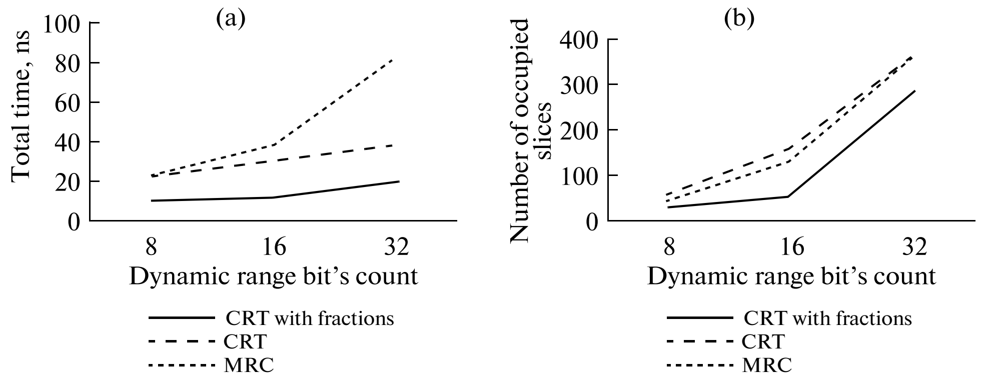

Fig. 7. Comparison of (a) total delay and (b) device utilization for algorithms with moduli sets from Table 4 (without using DSP48E1 slices).

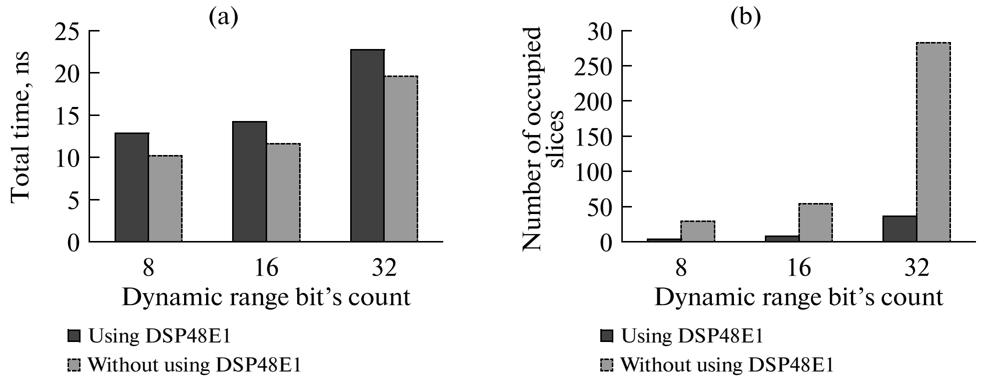

Fig. 8. Comparison of (a) total delay and (b) device utilization for the proposed approximate method with moduli sets from Table 4 (with and without using DSP48E1 slices).

known solutions, such as those using MRC. The results of simulation showed the high efficiency of the proposed architecture of comparing modular numbers.

Subsequent investigations will be aimed at adapting the proposed algorithms for particular applications, using special sets of RNS moduli, and increasing the stability of modular structures.

ACKNOWLEDGMENTS

This work was supported by the Russian Foundation for Basic Research, project no. 14-07-31004-mol-a.

REFERENCES

1. Chervyakov, N.I., Sakhnyuk, P.A., Shaposhnikov, A.V., and Makokha, A.N., *Neirokomp'yutery v ostatochnykh klassakh. Uchebnoe posobie dlya vuzov* (Neurocomputers in Residual Classes. Textbook for High Schools), Moscow: Radiotekhnika, 2003.
2. Chervyakov, N.I., Sakhnyuk, P.A., Shaposhnikov, A.V., and Ryadnov, S.A., *Modulyarnye parallel'nye vychislitel'nye struktury neiroprotsessornykh system* (Modular Parallel Computing Structures of Neuroprocessor Systems), Moscow: Fizmatlit, 2003.
3. Szabo, N.S. and Tanaka, R.I., *Residue Arithmetic and Its Application to Computer Technology,* McGraw-Hill, 1967.
4. Omondi, A. and Premkumar, B., *Residue Number Systems. Theory and Implementation,* London: Imperial College Press, 2007.
5. Reddy, K.S., Akshit, S., and Sahoo, S.K., A new approach for high performance RNS-FIR filter using the moduli set, *Proc. IEEE Symp. on Computer Applications and Industrial Electronics (ISCAIE),* Penang, Malaysia, 2014, pp. 136–140.

6. Chervyakov, N.I., Lyakhov, P.A., and Babenko, M.G., Digital filtering of images in a residue number system using finite-field wavelets, *Autom. Control Comput. Sci.,* 2014, vol. 48, no. 3, pp. 180–189.
7. Alia, G. and Martinelli, E., NEUROM: A ROM based RNS digital neuron, *Neuron Networks,* 2005, no. 18, pp. 179–189.
8. Yatskiv, V., Su, J., Yatskiv, N., Sachenko, A., and Osolinskiy, O., Multilevel method of data coding in WSN, *Proc. IEEE 6th Int. Conf. on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS),* 2011, pp. 863–866.
9. Gomathisankaran, M., Tyagi, A., and Namuduri, K., HORNS: A homomorphic encryption scheme for cloud computing using residue number system, *Proc. IEEE 45th Annual Conference on Information Sciences and Systems (CISS),* 2011, pp. 1–5.
10. Akkal, M. and Siy, P., A new mixed radix conversion algorithm MRC-II, *J. Syst. Archit.,* 2007, vol. 53, no. 9, pp. 577–586.
11. Navi, K., Esmaeildoust, M., and Molahosseini, A.S., A general reverse converter architecture with low complexity and high performance, *IEICE Trans. Inf. Syst.,* 2011, vol. E94-D, no. 2, pp. 264–273.
12. Chervyakov, N.I., Babenko, M.G., Lyakhov, P.A., and Lavrinenko, I.N., An approximate method for comparing modular numbers and its application to the division of numbers in residue number systems, *Cybern. Syst. Anal.,* 2014, vol. 50, no. 6, pp. 977–984.
13. Molahosseini, A.S., Sorouri, S., and Zarandi, A.A.E., Research challenges in next-generation residue number system architectures, *Proc. IEEE 7th International Conference on Computer Science & Education (ICCSE),* 2012, pp. 1658–1661.
14. Gbolagade, K.A. and Cotofana, S.D., An O(n) residue number system to mixed radix technique, *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS 2009),* 2009, pp. 521–524.
15. Hung, C.Y. and Parhami, B., An approximate sign detection method for residue numbers and its application to RNS division, *Comput. Math. Appl.,* 1994, vol. 27, no. 4, pp. 23–25.
16. Chervyakov, N.I., Methods and principles of modular neural computers, in *"50 let modulyarnoi arifmetike". Sbornik nauchnykh trudov* (50 Years of Modular Arithmetic. Collection of Scientific Papers), Moscow: OAO Angstrem, MIET, 2005.

Translated by P. Pozdeev