Classification of Moduli Sets for Residue Number System With Special Diagonal Functions

PETER BOYVALENKOV\(^1\), NIKOLAY I. CHERVYAKOV\(^2\), PAVEL LYAKHOV\(^3\),\(^4\), NATALYA SEMYONOVA\(^2\), ANTON NAZAROV\(^2\), MARIA VALUEVA\(^2\), (Graduate Student Member, IEEE), GEORGI BOYVALENKOV\(^3\), DANIL BOGAEVSKIY\(^4\), AND DMITRII KAPLUN\(^3\), (Member, IEEE)

\(^1\)Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, 1113 Sofia, Bulgaria
\(^2\)Department of Applied Mathematics and Mathematical Modeling, North-Caucasus Federal University, 355000 Stavropol, Russia
\(^3\)Bosch.IO LTD, 1124 Sofia, Bulgaria
\(^4\)Department of Automation and Control Processes, Saint Petersburg Electrotechnical University “LETI”, 197376 Saint Petersburg, Russia

Corresponding author: Dmitrii Kaplun (dikaplun@etu.ru)

This work was supported by the Russian Science Foundation under Grant 19-19-00566. The work of Peter Boyvalenkov was supported by the National Scientific Program through the Information and Communication Technologies for a Single Digital Market in Science, Education and Security (ICTinSES) of the Bulgarian Ministry of Education and Science.

ABSTRACT

The paper presents algorithms for the generation of Residue Number System (RNS) triples with \(SQ = 2^k - 1\) and quadruples with \(SQ = 2^k\) for some \(k\). Triples and quadruples allow us to design efficient hardware implementations of non-modular operations in RNS such as division, sign detection, comparison of numbers, reverse conversion with using of a diagonal function from requiring division with the remainder by the diagonal module \(SQ\). Division with a remainder in the general case is the most complex arithmetic operation in computer technology. However, the consideration of special cases can significantly simplify this operation and increase the efficiency of hardware implementation. We show that there are 5573 good RNS triples (2301 even and 2372 odd) with elements less than 10 000, as the values of \(SQ\) vary from \(2^5 - 1\) to \(2^{27} - 1\). In contrast, RNS quadruples with \(SQ = 2^k\) seem to be quite rare. Restricting our search to sums of the elements in a quadruple less than 4000 we find that exactly 31 such quadruples exist. Their values of \(SQ\) vary between \(2^{20}\) and \(2^{30}\) with always even exponent. We suggest the measure of RNS balance and find perfectly balanced RNS among triples according to this measure. We demonstrate the advantages of more balanced quadruples by means of hardware implementation.

INDEX TERMS

Residue number system, non-modular operations, diagonal function, triples, quadruples, RNS balance, average bit-width, hardware implementation, FPGA.

I. INTRODUCTION

The current level of computer technology requires the development of parallel computing architectures and methods for organizing calculations on them. One of the ways of parallel organization of calculations at the arithmetic level is the transition from the traditional binary number system to the Residue Number System (RNS). The main idea of such a replacement is the ability to quickly and parallel processing the residues of a small bit-width when performing arithmetic operations of addition, subtraction and multiplication. This approach is very promising for practical applications that require intensive execution of mainly these operations: digital signal and image processing [1], [2], cryptography [3] and machine learning [4]. The disadvantage of RNS is the high computational complexity of performing non-modular operations, which include division, sign detection and comparison of numbers [5], [6]. These limitations exist because RNS is a non-positional number system, and magnitude comparison of numbers in RNS form is impossible, so the division operation consists of a magnitude comparison operation that is also a problematic operation. Finding faster algorithms would allow detecting more promising new areas to apply RNS.

One way to implement non-modular operations in RNS is to use a diagonal function from [7]–[10] requiring division with the remainder by the diagonal module \(SQ\). Division
with a remainder in the general case is the most complex arithmetic operation in computer technology. However, the consideration of special cases can significantly simplify this operation. Satisfying the condition $SQ = 2^k$ allows to calculate the remainder of dividing by $SQ$ by simply choosing the $k$ least significant bits of the dividend, while the quotient is determined by the remaining most significant bits of the dividend. Under the condition $SQ = 2^k - 1$ the remainder of the division by $SQ$ is calculated as the sum of the $k$-bit parts of the dividend modulo $SQ$ [11]. Algorithms for RNS constructing with special $SQ$ forms based on number-theoretical properties are presented in [7] and [8]. The main drawbacks of those algorithms is the impossibility of RNS construction with predetermined dynamic ranges as well as obtaining unbalanced RNSs (with too much difference in modules bit-width) in most cases.

In this paper, we present algorithms for generation of RNS triples with $SQ = 2^k - 1$ and quadruples with $SQ = 2^k$ for some predetermined $k$. By “predetermined” we mean that designers of RNSs could use the results of our classification by choosing in advance $k$ in certain range (for example, between 5 and 21 with a few exceptions for triples from Appendix A, and any even $k$ between 20 and 30 for quadruples from Appendix B). Moreover, the lists from Appendixes A and B are complete for the ranges given in Theorems 4 and 6 below.

Our approach is based on careful dealing with the exponent of 2 in the expressions which naturally arise in targeting corresponding representations of $SQ$ and related quantities. We show that there are 5573 good RNS triples (2301 with three even modules and 2372 with two odd and one even modules) with elements less than 10 000, as the values of $SQ$ vary from $2^{51}$ to $2^{27,31}$. In contrast, RNS quadruples with $SQ = 2^k$ seem to be quite rare. We restrict our search to sums of the elements in a quadruple less than 4000 and find that exactly 31 such quadruples exist. Their values of $SQ$ vary between $2^{20}$ and $2^{30}$ with always even exponent.

The rest of the paper is organized as follows. Section II presents the construction of RNS with a convenient form of diagonal function. Sections III and IV present effective algorithms for computing triples and quadruples, respectively, to build effective RNS blocks based diagonal function. Section V discusses approach to measuring RNS balance and the methods of RNS construction using proposed approach. Section VI presents hardware simulation results. Discussion is presented in Section VII. The conclusion of the paper is reported in Section VIII.

II. RNS MATH BACKGROUND. METHODOLOGY

Let $m_1, m_2, \ldots, m_n, n \geq 3$, be mutually co-prime positive integers and

$$X \equiv x_i \quad (mod \ m_i), \quad i = 1, 2, \ldots, n, \quad (1)$$

be a corresponding Chinese Remainder Theorem. In RNS, a solution $X$ of (1) is associated to the $n$-tuple $(x_1, x_2, \ldots, x_n)$ and that $n$-tuple is used in operations instead of $X$ [12, Section 3.4].

Denote

$$M = \prod_{i=1}^{n} m_i, \quad M_i = \frac{M}{m_i}, \quad i = 1, 2, \ldots, n.$$

Then

$$SQ := \sum_{i=1}^{n} M_i$$

is called diagonal modulus. The diagonal modulus is important in computations as it is instrumental in the definition of the diagonal function

$$D(x) := \sum_{i=1}^{n} k_i x_i \quad (mod \ SQ),$$

where the integers $k_i$ are defined by $k_i m_i \equiv -1 \quad (mod \ SQ)$.

It was observed that diagonal modulus of special binary representations (very low or very high Hamming weight) are useful in the construction of RNS with good performance. This point was described in [7].

Our goal is to provide certain classification results for systems $(m_1, m_2, \ldots., m_n)$ for small $n$. Such classification could be useful for designers of RNS in their search for good performance. Thus, we also discuss performance issues, adding here that predetermined range of $M$ (see above for predetermined $k$) implied from our Appendixes could be very useful.

We shall use many times the following simple lemma. We use the notation $v_2(m)$ for the maximum degree of 2 which divides a positive integer $m$ (i.e., $m/2^{v_2(m)}$ is an odd integer).

Lemma 1: Let $k$ be a positive integer and $2^k = a + b$, where $a$ and $b$ are positive integers. Then $v_2(a) = v_2(b)$.

Proof: Assume for a contradiction that $v_2(a) \neq v_2(b)$. Without loss of generality, let $v_2(a) > v_2(b)$. It is clear that $v_2(b) < k$ (otherwise $a + b > 2^{v_2(b)} \geq 2^k$). Then it follows that $0 \equiv 2^k = a + b \equiv 2^{v_2(b)} b_1 \quad (mod \ 2^{v_2(a)})$, which is impossible ($b_1 = b/2^{v_2(b)}$ is odd).

Remark: It is clear from the proof that the conclusion $v_2(a) = v_2(b)$ of Lemma 1 is true whenever $\max(v_2(a), v_2(b)) < k$ (i.e., the condition for $a$ and $b$ both being positive integers is dropped). We will use this fact once in our analysis of RNS quadruples.

Our approach is based on applications of Lemma 1 with appropriate representations of $SQ$ and its relatives. This leads to significant simplifications which allow us to develop efficient algorithms achieving our goals.

III. RNS TRIPLES WITH $SQ = 2^k - 1$

A. THREE ODD $m_i$

We are interested in diagonal modulus $SQ = 2^k - 1$ with three odd $m_1, m_2, m_3$. We call such triples odd.

Starting with co-prime odd $m_1$ and $m_2$, we require

$$v_2(m_1 + m_2) = v_2(m_1 m_2 + 1) = \omega \geq 1 \quad (2)$$

Since

$$SQ + 1 = 2^k = m_3 (m_1 + m_2) + (m_1 m_2 + 1),$$
it follows from Lemma 1 that (2) is a necessary condition. Moreover, we have
\[
\frac{(m_1 + m_2 - 2\omega, m_1m_2 + 1)}{2\omega} = 1. \tag{3}
\]
Indeed, otherwise (2) implies that any nontrivial prime common divisor of \(\frac{(m_1 + m_2)}{2\omega}, \frac{m_1m_2 + 1}{2\omega}\) will be an odd prime which will divide \(2^k\), a contradiction.

It follows from the above that
\[
m_1 + m_2 = 2^\omega r, \quad m_1m_2 + 1 = 2^\omega s
\]
where \(r\) and \(s\) are co-prime odd integers. Then
\[
2^k - \omega = s + rm_3
\]
and we have to seek solutions of
\[
2^l \equiv s \pmod{r}
\]
with respect to \(l\). Each such solution defines a candidate for \(m_3\) by
\[
2^l = s + rm_3 \tag{5}
\]
This \(m_3\) is approved if it is co-prime to both \(m_1\) and \(m_2\).

**Theorem 2:** The so chosen \(m_1, m_2, m_3\) form a good RNS triple with \(SQ = 2^k - 1\), where \(k = \ell + \omega\). Each odd triple can be found this way.

**Proof:** For \(m_1, m_2, m_3\) as chosen above we have consecutively by (4) and (6)
\[
SQ = M_1 + M_2 + M_3 = m_1m_2 + m_3 (m_1 + m_2)
= 2^\omega s - 1 + \frac{2^l - s}{r} \cdot 2^\omega r = 2^l + \omega - 1,
\]
as required.

Theoretical investigations can be separated into two cases depending on whether (5) always or not always has solutions.

1. \(r = \rho^\omega\), where \(\rho\) is an odd prime, and 2 is primitive root modulo \(\rho^\omega\). In this case (5) has always solutions.
2. \(r\) is an odd integer such that (5) has solutions. This includes two sub-cases:
   1. \(r = \rho^\omega\), where \(\rho\) is an odd prime, but 2 is not primitive root modulo \(\rho^\omega\) and
   2. \(r\) is divisible to at least two distinct primes.

However, it is not necessarily important to distinguish between these cases. In particular, we may skip the check whether 2 is a primitive root and just care for (5) having a solution.

Therefore, the classification itself can be organized as follows.

Step 1. For fixed even positive integer \(A\), generate all pairs \((m_1, m_2)\) of odd positive integers such that \(m_1 + m_2 = A, m_1 < m_2\).

Step 2. Given a pair \((m_1, m_2)\), check whether (2) and (3) are satisfied; otherwise consider next pair.

Step 3. For \((m_1, m_2)\) satisfying (2) find \(\omega, r\) and \(s\) and check whether (5) has a solution.

Step 4. Check in increasing order solutions of (5) until suitable \(m_3 = (2^l - s)/r\) is found.

**Output:** \((m_1, m_2, m_3), k = \omega + l, SQ = 2^{\omega + l} - 1\).

This algorithm was implemented by a program in C++. The results are described in the end of the section.

**B. TWO ODD AND ONE EVEN \(m_i\)**

It is clear that at most one of \(m_1, m_2, m_3\) can be even. Thus, we complete consideration of triples with diagonal modulus \(SQ = 2^k - 1\) by analysis of triples \((m_1, m_2, m_3)\), where \(m_1\) and \(m_2\) are odd and \(m_3\) is even. We call such triples even. Following the above scheme, we write
\[
m_3 = \rho^\omega q, \quad m_1 + m_2 = 2^\omega r, \quad m_1m_2 + 1 = 2^\omega s, \tag{7}
\]
where \(q, r, s\) are odd and mutually coprime. Now
\[
SQ + 1 = 2^k = m_3 (m_1 + m_2) + (m_1m_2 + 1) = 2^{\omega + \rho} qr + 2^\omega s
\]
and Lemma 1 implies that
\[
v_2(m_1 + m_2) + v_2(m_3) = v_2(m_1m_2 + 1) \iff \omega_1 + \rho = \omega_2. \tag{8}
\]
Note that \(\omega_1 < \omega_2\). So far
\[
2^k = 2^{\omega_1 + \rho} qr + 2^\omega s = 2^\omega (qr + s).
\]
This means that we need to choose \(q\) as a solution of \(qr + s = 2^l\) for some \(\ell\), thus getting
\[
SQ = 2^{\omega_2 + \ell} - 1.
\]
Noting that \(q = m_3/2^\omega\) shows the similarity with the case of three odd \(m_i\) (it can be deduced from here by setting \(\rho = 0\)).

Similarly, to above, the following statement follows.

**Theorem 3:** The so chosen \(m_1, m_2, m_3\) form a good RNS triple with \(SQ = 2^k - 1\), where \(k = \ell + \omega_2\). Each even triple can be obtained this way.

In this case the classification can be organized as follows.

Step 1. For fixed even positive integer \(A\), generate all pairs \((m_1, m_2)\) of odd positive integers such that \(m_1 + m_2 = A, m_1 < m_2\).

Step 2. Given a pair \((m_1, m_2)\), find \(\omega_1, \omega_2, r\) and \(s\) as (7) requires and check whether the conditions \(\omega_2 > \omega_1\) (see (8)) and \((r, s) = 1\) are satisfied; otherwise consider next pair.

Step 3. For \((m_1, m_2)\) from Step 2, solve \(2^l \equiv s \pmod{r}\) with respect to \(\ell\).

Step 4. Check in increasing order solutions \(q = (2^l - s)/r\) with \(\ell's from Step 3, finding \(m_3 = 2^{\omega_2 - \omega_1} q\), where \((q, r) = (q, s) = 1\).

**Output:** \((m_1, m_2, m_3), k = \omega_2 + l, SQ = 2^{\omega_2 + l} - 1\).

**C. RESULTS FOR RNS TRIPLES WITH \(SQ = 2^k - 1\)**

We implemented both algorithms for finding all suitable triples with \(m_i \leq 1000, i = 1, 2, 3\). With running time less than a second, the program produced 412 triples shown in Appendix A. These results were also confirmed by a brute force search. Further, there are 5573 good triples (2301 even and 2372 odd) with \(m_i \leq 10000\), generated by our algorithm in few hours, as \(SQ = 2^{27} - 1\) appears as largest.
Theorem 4: There are exactly 412 triples \((m_1, m_2, m_3)\) (as given in Appendix A) such that \(m_i \leq 1000, i = 1, 2, 3\), and \(SQ = 2^k - 1\) for some positive integer \(k\). Further, there are exactly 5573 good triples \((2301\text{ with two odd and one even } m_i \text{ and } 2372\text{ with three odd } m_i)\) with \(m_i \leq 10000, i = 1, 2, 3,\) and \(SQ = 2^k - 1\).

Proof: The necessary conditions from above mean that all possible triples are generated as described. The sufficiency follows from Theorems 2 and 3.

The list of all 5573 triples from Theorem 4 is available upon request.

IV. RNS QUADRUPLES WITH \(SQ = 2^k\)

For \(n = 4\) the targeted \(SQ\) (odd or even) determines the type of the quadruple \((m_1, m_2, m_3, m_4)\). Since we need \(SQ = 2^k\), all \(m_i\) must be odd.

Since

\[2^k = m_1m_2(m_3 + m_4) + m_3m_4(m_1 + m_2),\]

it follows from Lemma 1 that the first necessary condition is

\[v_2(m_1 + m_2) = v_2(m_3 + m_4) = \omega_1,\]

\[m_1 + m_2 = 2^{\omega_1}r, \quad m_3 + m_4 = 2^{\omega_2}s,\] (9)

where \(r\) and \(s\) are odd and coprime. Now

\[2^k = 2^{\omega_1}[m_3m_4 + sm_1m_2]
= 2^{\omega_1}[m_3(2^{\omega_1}s - m_3) + sm_1(2^{\omega_1}r - m_1)]
= 2^{\omega_1}\left[rs2^{\omega_1}(m_1 + m_3) - (rm_2^2 + sm_1^2)\right],\]

whence \(2^{k-\omega_1} = rs2^{\omega_1}(m_1 + m_3) - (rm_2^2 + sm_1^2)\). Using Lemma 1 again (see the remark), we require

\[\omega_1 + v_2(m_1 + m_3) = v_2\left(\frac{rm_2^2 + sm_1^2}{2}\right) = \omega_1 + \omega_2,\]

\[m_1 + m_3 = 2^{\omega_2}q, \quad rm_2^2 + sm_1^2 = 2^{\omega_1+\omega_2}t,\] (10)

where \(q\) and \(t\) are odd coprime positive integers (in fact, all four numbers \(r, s, q, t\) are odd and mutually coprime).

Plugging these, we obtain \(2^{k-\omega_1} = 2^{\omega_1+t+\omega_2}\) (rsq-t).

Finally, we have to search for \(rsq - t = 2^{\omega_3}\) for some positive integer \(\omega_3\).

Therefore, we have proved the following statement.

Theorem 5: The so chosen \(m_1, m_2, m_3, m_4\) form a good RNS quadruple with \(SQ = 2^k\) where \(k = 2^{\omega_1} + \omega_2 + \omega_3\).

Proof: For \(m_1, m_2, m_3, m_4\) as chosen above we obtain

\[SQ = 2^{2^{\omega_1+\omega_2}}(rsq - t) = 2^{\omega_1+\omega_2+\omega_3},\]

as required.

Remark: Note that the slight modification \(rsq - t = 2^{\omega_3} - 1\) in the last step leads to

\[SQ = 2^{2^{\omega_1+\omega_2+\omega_3}} - 2^{2^{\omega_1+\omega_2}},\]

which could be interesting in some applications.

Thus, we propose classification as follows.

Step 1. Find all quadruples \((m_1, m_2, m_3, m_4)\) of odd positive integers (less than 1000), such that (9) is satisfied.

Step 2. For any quadruple \((m_1, m_2, m_3, m_4)\) from Step 1, compute \(m + n\) and \(m^2 + sm^2\), where \(m \in \{m_1, m_2\}\) and \(n \in \{m_3, m_4\}\) and check if (10) is satisfied.

Step 3. Check if \(rsq - t\) is a power of 2.

Output. If \(rsq - t = 2^{\omega_3}\) in Step 3, the output is \((m_1, m_2, m_3, m_4), 2\omega_1 + \omega_2 + \omega_3, SQ = 2^{2^{\omega_1+\omega_2+\omega_3}}\).

We implemented the above algorithms for finding all suitable quadruples with \(m_1 + m_2 + m_3 + m_4 \leq 4000\). The running time was a few hours on a home PC. The program produced 31 quadruples shown in Appendix B. The results with \(m_i \leq 1000\) were confirmed by a brute force program working 9 hours on GPU.

Theorem 6: There are exactly 31 quadruples \((m_1, m_2, m_3, m_4)\) (as given in Appendix B) such that \(m_1 + m_2 + m_3 + m_4 \leq 4000\) and \(SQ = 2^k\) for some positive integer \(k\).

Proof: The necessary conditions from above mean that all possible quadruples are generated as described. The sufficiency follows from Theorem 5.

V. BALANCE METRIC FOR BUILDING EFFECTIVE COMPUTATIONAL SYSTEMS

The practical implementation of the arithmetic operations of addition, subtraction and multiplication in RNS with modules \((m_1, m_2, \ldots, m_n)\) is based on the parallel execution of operations for each of the modules \(m_i, i = 1, 2, \ldots, n\). In the general case, the addition of two numbers modulo \(m_i\) has computational complexity \(\sim O(b_i)\), where \(b_i = \lfloor \log_2 m_i \rfloor\) is modulo \(m_i\) bit-width. Multiplication of two numbers modulo \(m_i\) generally has computational complexity \(\sim O(b_i^2)\). If all RNS modules have a very different bit-width, then this will lead to a long idle time of the computational elements for low-bit-width modulo while computing for modules of higher bit-width [13]. This phenomenon is called unbalanced RNS [14].

The triplets and quadruples found in the Sections III and IV, listed in Appendices A and B, are not equally balanced. We introduce the concept of a measure of the RNS balance. Let the RNS be defined by modules \((m_1, m_2, \ldots, m_n)\) with bit-widths \((b_1, b_2, \ldots, b_n)\). Denote average bit-width of RNS modules a

\[
\bar{b} = \frac{\sum_{i=1}^{n} b_i}{n}. \tag{11}
\]

Obviously, the larger \(\bar{b}\) implies the greater range \(M\) of RNS.

Let us define a measure of RNS balance, due to absolutely absence of metrics for RNS balance measuring in the literature, as a quantity \(\beta\), determined by the formula

\[
\beta = \frac{\sum_{i=1}^{n} (b_i - \bar{b})^2}{n}, \tag{12}
\]

and calculated as the dispersion of bit-widths of the RNS modules. We assume the more balanced of the two different RNSs that one, in which \(\beta\) is smaller.

Definition 1: An RNS is called perfectly balanced if \(\beta = 0\).
TABLE 1. Modeling results.

| Moduli set          | LUTs | Delay, ns | Power, W |
|---------------------|------|-----------|----------|
| Magnitude comparison|      |           |          |
| {43,51,79,91}       | 596  | 12.607    | 35.342   |
| (5,29,93,313)       | 423  | 13.118    | 21.926   |

Example: Consider quadruples (43, 51, 79, 91), (23, 43, 87, 143), (5, 29, 93, 313) and (3, 7, 43, 2323) from App. 2. They all have $SQ = 2^{20}$.

For quadruple (43, 51, 79, 91) we have $M = 15765477$, $\log_2 M \approx 23.9$, $(b_1, b_2, b_3, b_4) = (6, 6, 7, 7)$, $\bar{b} = 6.5$, $\beta = 0.25$.

For quadruple (23, 43, 87, 143) we have $M = 12304149$, $\log_2 M \approx 23.6$, $(b_1, b_2, b_3, b_4) = (5, 6, 7, 8)$, $\bar{b} = 6.5$, $\beta = 1.25$.

For quadruple (5, 29, 93, 313) we have $M = 4220805$, $\log_2 M \approx 22.0$, $(b_1, b_2, b_3, b_4) = (3, 5, 7, 9)$, $\bar{b} = 6$, $\beta = 5$.

For quadruple (3, 7, 43, 2323) we have $M = 2097669$, $\log_2 M \approx 21.0$, $(b_1, b_2, b_3, b_4) = (2, 3, 6, 12)$, $\bar{b} = 5.75$, $\beta = 15.1875$.

The cases examined show that quadruples (5, 29, 93, 313) and (3, 7, 43, 2323) are least suitable for practical use, since they, firstly, have the smallest ranges ($\log_2 M < 23$), and secondly, are very unbalanced, as they have the largest values $\beta$. Quadruples (43, 51, 79, 91) and (23, 43, 87, 143) have large and approximately equal ranges ($\log_2 M > 23$), however, in practice preference should be given to the quadruple (43, 51, 79, 91), since it is very well balanced, which is confirmed by the smallest value $\beta$.

It is shown in Fig.1 the distribution of all 412 triples according to value of $\beta$. It can be noticed that we have 22 perfectly balanced triples.

In Fig.2 the distribution of 31 quadruples according to value of $\beta$ is presented. We can’t see any perfectly balanced quadruples but have one quadruple with $\beta = 0.25$.

In the previous work [7], only triples (3, 5, 14) and (7, 9, 12) and quadruple (5, 29, 93, 31) were analyzed in the experimental part. These moduli sets have $\beta = 0.67$, $\beta = 2.89$ and $\beta = 5.00$, respectively. In this paper we found a large number of better balanced RNSs with lower $\beta$, which will be more effective in practice than these from [7].

When developing a computing system in RNS, it is necessary to take into account the range and requirements for the number of modules. After determining these parameters from Appendices A and B, the most promising RNS moduli sets are these with the lowest possible $\beta$. These results can be used for building effective parallel computational systems [15] based on computers with parallel structure like FPGA and GPU [16], [17]. The basic idea of a hardware implementation is that an algorithm (division, sign detection, comparison of numbers, reverse conversion) based on a diagonal function requires division by $SQ$. Since we were able to find such quadruples for which $SQ = 2^n$, for such RNS the algorithm based on the diagonal function will be extremely better than an algorithm based on Chinese remainder theorem (CRT) [18], CRT with fractional values (CRTf) [19] and mixed radix conversion (MRC) [20]. For example, division with the remainder by $2^n$, in fact, costs nothing, unlike division by $M$ in CRT or multiplication by $M$, as in CRTf or different operations on the modules $m_1, m_2, \ldots, m_n$, as in MRC. This is confirmed by our previous studies, so in [7] there is an example of the implementation of non-modular operations of comparison and reverse conversion for triples and quadruples, which demonstrates the advantage of our proposals for the hardware implementation of systems based on RNS in FPGA.

VI. HARDWARE MODELING

The hardware modeling goal is comparison of circuits on the example of a problematic comparator device with the known {5, 29, 93, 313} moduli set from [7] and the proposed {43, 51, 79, 91} moduli set which has the same $SQ = 2^{20}$ and the lowest $\beta$. In this regard, the operation of magnitude comparison two numbers in RNS was implemented in FPGA. All simulated circuits were described in very high-speed integrated circuit (VHDL) hardware description language (VHDL). Hardware modeling was performed on Xilinx Artix
TABLE 2. Triples with $\text{SQ} = 2^k - 1$.

| RNS modules | RNS parameters | $k$ | $\beta$ |
|-------------|----------------|-----|---------|
| $m_1$ $m_2$ $m_3$ | $M$ | $\log_2 M$ | |
| 3 5 2 | 30 | 4 | 5 | 0.67 |
| 3 5 14 | 210 | 7 | 7 | 0.67 |
| 7 9 4 | 252 | 7 | 7 | 0.67 |
| 3 101 2 | 606 | 9 | 9 | 6.89 |
| 3 5 62 | 930 | 9 | 9 | 2.89 |
| 3 37 10 | 1110 | 10 | 9 | 2.67 |
| 3 17 23 | 1173 | 10 | 9 | 2.00 |
| 13 27 4 | 1404 | 10 | 9 | 1.56 |
| 15 17 8 | 2040 | 10 | 9 | 0.67 |
| 5 291 2 | 2910 | 11 | 11 | 11.56 |
| 3 5 254 | 3810 | 11 | 11 | 6.89 |
| 5 9 143 | 6435 | 12 | 11 | 4.67 |
| 7 9 124 | 7812 | 12 | 11 | 2.89 |
| 5 51 32 | 8160 | 12 | 11 | 1.56 |
| 9 103 10 | 9270 | 13 | 11 | 2.00 |
| 9 19 67 | 11457 | 13 | 11 | 1.56 |
| 9 29 47 | 12267 | 13 | 11 | 0.67 |
| 9 743 2 | 13374 | 13 | 13 | 14.00 |
| 15 17 56 | 14280 | 13 | 11 | 0.67 |
| 53 147 2 | 15582 | 13 | 13 | 8.67 |
| 31 33 16 | 16368 | 13 | 11 | 0.67 |
| 19 37 24 | 16872 | 14 | 11 | 0.22 |
| 3 7 817 | 17157 | 14 | 13 | 12.67 |
| 3 17 407 | 20757 | 14 | 13 | 8.22 |
| 3 325 22 | 21450 | 14 | 13 | 8.22 |
| 3 37 202 | 22422 | 14 | 13 | 6.22 |
| 3 197 38 | 22458 | 14 | 13 | 6.22 |
| 3 47 161 | 22701 | 14 | 13 | 6.22 |
| 3 79 97 | 22989 | 14 | 13 | 5.56 |
| 5 627 8 | 25080 | 14 | 13 | 10.89 |
| 25 279 4 | 27900 | 14 | 13 | 8.22 |
| 11 23 113 | 28589 | 14 | 12 | 1.56 |
| 7 9 508 | 32004 | 14 | 13 | 6.89 |
| 5 21 311 | 32655 | 14 | 13 | 6.22 |
| 13 427 6 | 33306 | 15 | 13 | 6.89 |
| 5 47 153 | 35955 | 15 | 13 | 4.22 |
| 5 99 74 | 36630 | 15 | 13 | 3.56 |
| 7 13 405 | 36855 | 15 | 13 | 6.89 |
| 9 13 367 | 42939 | 15 | 13 | 5.56 |
| 7 33 199 | 45969 | 15 | 13 | 4.22 |

| 7 73 96 | 49056 | 15 | 13 | 3.56 |
| 19 709 4 | 53884 | 15 | 14 | 10.89 |
| 57 719 8 | 54264 | 15 | 13 | 2.89 |
| 9 35 179 | 56385 | 15 | 13 | 2.67 |
| 9 167 38 | 57114 | 15 | 13 | 2.67 |
| 9 79 85 | 60435 | 15 | 13 | 2.00 |
| 15 17 248 | 63240 | 15 | 13 | 2.89 |
| 5 23 581 | 66815 | 16 | 14 | 8.67 |
| 13 25 207 | 67275 | 16 | 13 | 2.89 |
| 13 27 196 | 68796 | 16 | 13 | 2.89 |
| 13 31 177 | 71331 | 16 | 13 | 2.89 |
| 13 139 42 | 75894 | 16 | 13 | 2.67 |
| 17 23 195 | 76245 | 16 | 13 | 2.00 |
| 13 63 97 | 79443 | 16 | 13 | 1.56 |
| 13 75 82 | 79950 | 16 | 13 | 2.00 |
| 7 19 625 | 83125 | 16 | 14 | 8.67 |
| 17 143 36 | 87516 | 16 | 13 | 1.56 |
| 3 65 479 | 93405 | 16 | 15 | 8.67 |
| 21 31 145 | 94395 | 16 | 13 | 2.00 |
| 17 63 89 | 95319 | 16 | 13 | 0.67 |
| 47 113 18 | 95598 | 16 | 13 | 0.67 |
| 51 101 20 | 102529 | 16 | 14 | 4.67 |
| 25 33 127 | 104775 | 16 | 13 | 0.67 |
| 3 53 72 | 133560 | 17 | 13 | 0.22 |
| 47 53 57 | 141987 | 17 | 13 | 0.00 |
| 19 37 280 | 196840 | 17 | 14 | 2.89 |
| 19 73 163 | 226081 | 17 | 14 | 1.56 |
| 19 85 142 | 229330 | 17 | 14 | 1.56 |
| 79 145 22 | 252010 | 17 | 14 | 1.56 |
| 173 747 2 | 258462 | 17 | 17 | 14.89 |
| 243 533 2 | 259038 | 17 | 17 | 14.89 |
| 23 89 128 | 262016 | 17 | 14 | 0.89 |
| 13 33 703 | 301878 | 18 | 15 | 6.22 |
| 71 137 32 | 312646 | 18 | 14 | 1.56 |
| 19 789 22 | 329802 | 18 | 15 | 5.56 |
| 35 89 107 | 333305 | 18 | 14 | 0.22 |
| Triples with $SQ = 2^k - 1$. |  |  |  |  |  |
|---|---|---|---|---|---|
| 13 79 345 | 354 315 | 18 15 | 4.22 | 23 729 152 | 2 548 584 |
| 47 65 119 | 363 545 | 18 14 | 0.22 | 295 393 22 | 2 550 570 |
| 13 171 166 | 369 018 | 18 15 | 3.56 | 23 165 677 | 2 569 215 |
| 3 901 142 | 383 826 | 18 17 | 11.56 | 23 177 635 | 2 585 085 |
| 3 223 577 | 386 013 | 18 17 | 11.56 | 23 537 212 | 2 618 412 |
| 3 229 562 | 386 094 | 18 17 | 11.56 | 73 133 271 | 2 631 139 |
| 3 287 449 | 386 589 | 18 17 | 10.89 | 23 257 447 | 2 642 217 |
| 27 605 26 | 424 710 | 18 15 | 5.56 | 23 377 306 | 2 653 326 |
| 19 309 82 | 481 422 | 18 15 | 2.67 | 23 327 353 | 2 654 913 |
| 19 145 183 | 504 165 | 18 15 | 2.00 | 177 623 26 | 2 867 046 |
| 183 697 4 | 510 204 | 18 17 | 11.56 | 293 387 26 | 2 948 166 |
| 27 397 52 | 557 388 | 19 15 | 2.89 | 27 173 632 | 2 952 072 |
| 27 79 289 | 616 437 | 19 15 | 2.67 | 113 879 32 | 3 178 464 |
| 33 59 335 | 652 245 | 19 15 | 2.00 | 33 107 911 | 3 216 741 |
| 27 131 185 | 654 345 | 19 15 | 2.00 | 33 127 793 | 3 323 463 |
| 101 227 30 | 687 810 | 19 15 | 1.56 | 37 103 909 | 3 464 209 |
| 13 73 751 | 712 699 | 19 16 | 6.00 | 33 191 557 | 3 510 771 |
| 13 331 178 | 765 934 | 19 16 | 4.67 | 33 203 527 | 3 530 373 |
| 37 99 214 | 783 882 | 19 15 | 0.67 | 37 117 823 | 3 562 767 |
| 7 873 142 | 867 762 | 19 17 | 8.67 | 35 149 684 | 3 567 060 |
| 7 169 738 | 873 054 | 19 17 | 8.67 | 33 247 439 | 3 578 289 |
| 7 211 589 | 878 199 | 19 17 | 8.67 | 33 415 262 | 3 588 090 |
| 7 291 433 | 882 021 | 19 17 | 8.00 | 33 287 380 | 3 598 980 |
| 65 71 207 | 955 305 | 19 15 | 0.22 | 37 135 733 | 3 661 335 |
| 71 201 68 | 970 428 | 19 15 | 0.22 | 45 971 86 | 3 757 770 |
| 59 125 138 | 1 017 750 | 19 15 | 0.67 | 37 579 178 | 3 813 294 |
| 313 823 4 | 1 030 396 | 19 18 | 12.67 | 37 183 565 | 3 825 615 |
| 127 129 64 | 1 048 512 | 19 15 | 0.67 | 113 815 42 | 3 867 990 |
| 47 881 26 | 1 076 582 | 20 16 | 4.67 | 47 89 933 | 3 902 739 |
| 11 173 702 | 1 335 906 | 20 17 | 6.22 | 37 243 436 | 3 920 076 |
| 11 237 518 | 1 350 426 | 20 17 | 6.22 | 37 403 264 | 3 936 504 |
| 149 803 12 | 1 435 764 | 20 17 | 6.22 | 37 271 393 | 3 940 611 |
| 151 793 12 | 1 436 916 | 20 17 | 6.22 | 37 307 348 | 3 952 932 |
| 13 123 952 | 1 522 248 | 20 17 | 6.00 | 47 93 905 | 3 955 755 |
| 13 157 759 | 1 549 119 | 20 17 | 6.22 | 63 65 992 | 4 062 249 |
| 13 667 180 | 1 560 780 | 20 17 | 6.22 | 55 81 931 | 4 147 605 |
| 13 327 373 | 1 585 623 | 20 17 | 5.56 | 47 113 786 | 4 174 446 |
| 15 161 731 | 1 765 365 | 20 17 | 6.22 | 47 123 737 | 4 260 597 |
| 15 337 358 | 1 809 690 | 20 17 | 5.56 | 77 923 60 | 4 264 260 |
| 241 495 16 | 1 908 720 | 20 17 | 4.67 | 59 909 80 | 4 290 480 |
| 17 143 804 | 1 954 524 | 20 17 | 4.22 | 51 103 817 | 4 291 701 |
| 49 127 337 | 2 097 151 | 20 16 | 1.56 | 57 89 863 | 4 377 999 |
| 21 115 946 | 2 284 590 | 21 17 | 4.22 | 49 123 727 | 4 381 629 |
| 23 117 917 | 2 467 647 | 21 17 | 4.22 | 47 149 633 | 4 432 899 |
| Triple | Triples with $SQ = 2^k - 1$. |
|--------|--------------------------------|
| 45     | 209 479 4504995 22 17 1.56 77 123 608 5758368 22 17 2.00 |
| 45     | 217 463 4521195 22 17 1.56 63 257 359 5812569 22 17 2.00 |
| 49     | 145 639 4540905 22 17 2.67 73 147 547 5869857 22 17 1.56 |
| 47     | 177 548 4558812 22 17 2.67 65 237 383 5900115 22 17 0.67 |
| 47     | 191 513 4605201 22 17 2.67 157 523 72 5911992 22 17 1.56 |
| 47     | 497 198 4625082 22 17 1.56 95 609 104 6016920 22 17 2.00 |
| 103    | 777 58 4641798 22 17 2.89 193 447 70 6038970 22 17 0.67 |
| 65     | 831 86 4645290 22 17 2.00 73 175 477 6093675 22 17 0.67 |
| 51     | 145 631 4666245 22 17 2.67 79 145 534 6116970 22 17 1.56 |
| 47     | 225 443 4684725 22 17 1.56 135 553 82 6121710 22 17 1.56 |
| 47     | 233 429 4697979 22 17 1.56 73 423 202 6237558 22 17 0.67 |
| 109    | 747 58 4722534 22 17 2.89 77 173 471 6274191 22 17 0.67 |
| 47     | 293 345 4750995 22 17 2.00 71 291 305 6301605 22 17 0.89 |
| 227    | 437 48 4761552 22 17 1.56 123 557 92 6303012 22 17 2.00 |
| 63     | 97 781 4772691 22 17 2.89 73 237 367 6349467 22 17 0.67 |
| 51     | 565 166 4783290 22 17 2.67 73 327 268 6397428 22 17 0.89 |
| 61     | 105 751 4810155 22 17 2.89 105 109 559 6397755 22 17 2.00 |
| 73     | 807 82 4830702 22 17 2.00 205 411 76 6403380 22 17 0.67 |
| 57     | 127 673 4871847 22 17 2.89 77 197 423 6416487 22 17 0.67 |
| 49     | 295 339 4900245 22 17 2.00 85 151 501 6430335 22 17 0.67 |
| 51     | 197 488 4902936 22 17 1.56 81 463 172 6450516 22 17 0.67 |
| 51     | 257 383 5019981 22 17 2.00 103 117 541 6519591 22 17 2.00 |
| 51     | 341 290 5043390 22 17 2.00 89 149 495 6564195 22 17 0.67 |
| 55     | 177 523 5091405 22 17 2.67 81 191 425 6575175 22 17 0.67 |
| 57     | 173 527 5196747 22 17 2.67 103 127 513 6710553 22 17 2.00 |
| 61     | 603 142 5223186 22 17 2.67 81 271 310 6804810 22 17 0.89 |
| 55     | 217 438 5227530 22 17 1.56 81 287 293 6811371 22 17 0.89 |
| 55     | 409 254 5263830 22 17 1.56 113 119 507 6817629 22 17 0.89 |
| 73     | 103 702 5278338 22 17 2.00 85 387 208 6842160 22 17 0.67 |
| 71     | 681 110 5318610 22 17 2.00 89 183 422 6873114 22 17 0.67 |
| 59     | 183 497 5366109 22 17 1.56 117 499 118 6889194 22 17 0.89 |
| 63     | 577 148 5379948 22 17 2.67 235 349 84 6889260 22 17 0.67 |
| 81     | 95 701 5394195 22 17 2.00 89 203 387 6991299 22 17 0.67 |
| 57     | 235 403 5398185 22 17 1.56 103 457 150 7060650 22 17 0.67 |
| 161    | 543 62 5420226 22 17 2.67 93 187 406 7060746 22 17 0.67 |
| 61     | 171 520 5424120 22 17 2.67 203 381 92 7115556 22 17 0.67 |
| 71     | 117 653 5424471 22 17 2.00 113 135 467 7124085 22 17 0.67 |
| 57     | 311 308 5459916 22 17 2.00 127 129 448 7339584 22 17 0.67 |
| 59     | 219 425 5491425 22 17 1.56 103 177 403 7347093 22 17 0.67 |
| 81     | 103 667 5564781 22 17 2.00 95 257 303 7397745 22 17 0.89 |
| 65     | 159 539 5570565 22 17 1.56 97 223 342 7397802 22 17 0.67 |
| 81     | 655 106 5623830 22 17 2.00 247 313 96 7421856 22 17 0.67 |
| 73     | 127 609 5646039 22 17 2.00 107 173 402 7441422 22 17 0.67 |
| 61     | 271 345 5703195 22 17 2.00 233 327 98 7466718 22 17 0.67 |
### TABLE 2. (Continued.) Triples with $SQ = 2^k - 1.$

| k  | 103 | 205 | 357 | 15 | 577 | 871 | 103 | 219 | 337 | 117 | 163 | 400 | 113 | 177 | 383 | 103 | 265 | 282 | 105 | 227 | 323 | 113 | 197 | 351 | 109 | 225 | 319 |
|----|-----|-----|-----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|-----|-----|
|    | 103 | 205 | 357 | 15 | 577 | 871 | 103 | 219 | 337 | 117 | 163 | 400 | 113 | 177 | 383 | 103 | 265 | 282 | 105 | 227 | 323 | 113 | 197 | 351 | 109 | 225 | 319 |
| 15 | 105 | 227 | 323 | 113 | 197 | 351 | 109 | 225 | 319 | 117 | 191 | 353 | 117 | 323 | 519 | 135 | 361 | 166 | 117 | 323 | 519 | 135 | 361 | 166 | 117 | 323 | 519 |
| 156112 | 7538055 | 22 | 17 | 0.67 | 35 | 617 | 771 | 16649745 | 23 | 19 | 3.56 | 17967001 | 24 | 18 | 0.89 | 18040676 | 24 | 18 | 1.56 | 18282810 | 24 | 19 | 3.56 | 19122195 | 24 | 19 | 3.56 | 20185169 | 24 | 18 | 0.89 | 21632215 | 24 | 18 | 0.89 | 21799660 | 24 | 18 | 0.22 | 22381485 | 24 | 19 | 3.56 | 22834350 | 24 | 19 | 3.56 | 23053020 | 24 | 19 | 2.89 | 23597860 | 24 | 18 | 0.22 | 24226870 | 24 | 18 | 0.22 | 29129925 | 24 | 19 | 2.00 | 31217355 | 24 | 19 | 2.00 | 32638683 | 24 | 19 | 2.00 | 33005805 | 24 | 19 | 1.56 | 33859215 | 25 | 19 | 2.00 | 34139892 | 25 | 19 | 1.56 | 35150220 | 25 | 19 | 2.00 | 35643432 | 25 | 19 | 1.56 | 36261270 | 25 | 19 | 1.56 | 37890237 | 25 | 19 | 1.56 | 38228190 | 25 | 19 | 2.00 | 38285415 | 25 | 19 | 1.56 | 39308700 | 25 | 19 | 1.56 | 39447342 | 25 | 19 | 2.00 | 39620280 | 25 | 19 | 1.56 | 40591785 | 25 | 19 | 2.00 | 41718765 | 25 | 19 | 2.00 | 42769965 | 25 | 19 | 2.00 | 44434005 | 25 | 19 | 1.56 | 44595810 | 25 | 19 | 1.56 | 47999643 | 25 | 19 | 0.67 | 48563610 | 25 | 19 | 0.89 | 48671820 | 25 | 19 | 0.89 | 48794883 | 25 | 19 | 0.67 | 49594440 | 25 | 19 | 0.67 | 49841220 | 25 | 19 | 0.67 | 50673000 | 25 | 19 | 0.67 | 50850267 | 25 | 19 | 0.67 | 51485133 | 25 | 19 | 0.89 | 52316565 | 25 | 19 | 0.67 | 52714892 | 25 | 19 | 0.67 |
TABLE 2. (Continued.) Triples with $\text{SQ} = 2^k - 1$.

| 197  | 275  | 996 |
|------|------|-----|
| 183  | 887  | 336 |
| 173  | 411  | 776 |
| 169  | 465  | 703 |
| 185  | 351  | 857 |
| 183  | 385  | 799 |
| 253  | 955  | 234 |
| 177  | 539  | 599 |
| 179  | 527  | 609 |
| 181  | 535  | 597 |
| 265  | 903  | 244 |
| 217  | 855  | 316 |
| 255  | 257  | 896 |
| 217  | 319  | 849 |
| 191  | 483  | 641 |
| 193  | 463  | 663 |
| 207  | 385  | 751 |
| 219  | 395  | 713 |
| 243  | 325  | 784 |
| 211  | 565  | 522 |
| 217  | 439  | 654 |
| 215  | 509  | 573 |
| 231  | 401  | 683 |
| 227  | 441  | 635 |
| 225  | 607  | 466 |
| 265  | 319  | 753 |
| 253  | 351  | 721 |
| 243  | 389  | 680 |
| 247  | 381  | 685 |
| 281  | 311  | 738 |
| 235  | 621  | 442 |
| 511  | 513  | 256 |
| 267  | 605  | 416 |
| 337  | 655  | 306 |
| 291  | 361  | 643 |
| 399  | 593  | 290 |
| 321  | 355  | 607 |
| 301  | 507  | 460 |
| 305  | 459  | 503 |
| 395  | 551  | 326 |
| 321  | 433  | 511 |
| 433  | 495  | 334 |
| 343  | 505  | 414 |

| 731  | 405  | 482 |
|------|------|-----|
| 387  | 485  | 386 |
| 399  | 407  | 449 |
| 163  | 859  | 889 |
| 211  | 901  | 772 |
| 233  | 683  | 971 |
| 317  | 677  | 839 |
| 593  | 911  | 338 |
| 373  | 643  | 796 |
| 427  | 577  | 799 |
| 487  | 499  | 817 |
| 629  | 707  | 452 |
| 607  | 993  | 934 |
| 641  | 911  | 975 |
| 647  | 887  | 993 |
| 697  | 859  | 963 |
| 757  | 823  | 933 |
| 747  | 893  | 872 |
| 775  | 873  | 862 |
| 797  | 849  | 863 |

TABLE 2. (Continued.) Triples with $\text{SQ} = 2^k - 1$.

| 731  | 405  | 482 |
|------|------|-----|
| 387  | 485  | 386 |
| 399  | 407  | 449 |
| 163  | 859  | 889 |
| 211  | 901  | 772 |
| 233  | 683  | 971 |
| 317  | 677  | 839 |
| 593  | 911  | 338 |
| 373  | 643  | 796 |
| 427  | 577  | 799 |
| 487  | 499  | 817 |
| 629  | 707  | 452 |
| 607  | 993  | 934 |
| 641  | 911  | 975 |
| 647  | 887  | 993 |
| 697  | 859  | 963 |
| 757  | 823  | 933 |
| 747  | 893  | 872 |
| 775  | 873  | 862 |
| 797  | 849  | 863 |

7 xc7a200tfbg484-2 in Vivado 2018.3 and the strategy of synthesis was highly area optimized. The modeling results were taken from an implementation run report.

The Fig. 3 shows a block diagram of the magnitude comparison operation using the diagonal function of the form $2^k$. The moduli set $\{m_1, m_2, \ldots, m_n\}$ has bit widths $a_1, a_2, \ldots, a_n$. Multiplication by constants is performed using the compression technique from [19]. First, partial product generator (PPG) forms partial products. Then, they are summed by carry save adder (CSA) tree and Kogge-Stone adder (KSA). The results of modeling are shown in Table 1.

The simulation of magnitude comparison shows that the using $\{43, 51, 79, 91\}$ moduli set allows to reduce delay of device by 4%, but requires 29% more number of LUTs and 37.96% more power consumption compare to the using $\{5, 29, 93, 313\}$ moduli set. We do not consider an area parameter since better balance is not necessarily the best bit-width solution. The known $\{5, 29, 93, 313\}$ moduli set has bit-widths $\{3, 5, 7, 9\}$ of computing channels, and the proposed $\{43, 51, 79, 91\}$ moduli set has bit widths $\{6, 6, 7, 7\}$. This fact explains the delay advantage of the comparator with $\{43, 51, 79, 91\}$ moduli set.

These results confirm our conclusions based on values of $\log_2 M$ and $\beta$. As for hardware resources, in [20] the authors
shown that selection of the appropriate moduli set should be based on the final usage of the RNS system. At first, we need to cover required dynamical range but for usage of hardware resources dynamical range is the most critical. For the {43, 51, 79, 91} moduli set we have $M = 15765477$, $\log_2 M \approx 23.9$, and for the {5, 29, 93, 313} we have it much fewer $M = 4220805$, $\log_2 M \approx 22.0$. Values of modeling results were normalized using division by bit width. The normalized results show that the use of {43, 51, 79, 91} moduli set allows to reduce delay of device by 11.58%, but requires 23% more number of LUTs and 32.59% more power consumption compare to the using {5, 29, 93, 313} moduli set. This point defines the difference in hardware usage for these moduli sets. In other words, a suitable trade-off is needed according to dynamical range and balance, that is why it is not possible to introduce single moduli set which is best for all applications.

VIII. DISCUSSION

The methodology developed in Sections 2-4 can be applied for investigations in other cases. It could be quite easily replicated for classification results for triples with $SQ = 2^k + 1$. The application for quadruples with $SQ = 2^k \pm 1$ could be very similar to our treatment of the case of triples with two odd and one even modules. Indeed, in this case we can start (signifying $m_4$ to be even) with the representation

$$2^k = m_4 (m_1 m_2 + m_2 m_3 + m_1 m_3) + (m_1 m_2 m_3 \pm 1),$$

in order to apply Lemma 1 with the even integers $a = m_4 (m_1 m_2 + m_2 m_3 + m_1 m_3)$ and $b = (m_1 m_2 m_3 \pm 1)$.

Other, more complicated, forms of $SQ$ will require suitable generalizations (in fact, consideration of different cases) of Lemma 1.

The work [21] raises the question of finding the best parameters of the RNS in terms of the number of modules and their performance. The approaches proposed in this paper allow us to answer the question about the best RNS moduli sets in terms of the performance of algorithms based on the diagonal function. In our opinion, the best practical solution for performance is the choice of a RNS with four modules covering the given range $M$ and having the smallest $\beta$. Suitable cases for ranges from 21 to 36 bits can be found in Appendix B. Such ranges are usually sufficient to solve most practical problems in digital processing of signals and images. Another important issue in the RNS theory is the problem of the effective implementation of the reverse conversion. As shown in [7], even unbalanced triples and quadruples can show good results for reverse converters based on a diagonal function.
Therefore, the triples and quadruples found in this paper can further improve the result for reverse converters, due to the greater balance of the RNS modules.

Further research will be related with testing algorithms using SQ on FPGA and GPU. These algorithms will be used to develop faster methods of digital signal processing, cryptography, machine learning using the proposed quadruples for RNS with SQ. It would also be interesting to study the relationship between $\beta$ and loss due to downtime in an unbalanced RNS. We can determine the connection between $\beta$ and losses due to equipment downtime in an unbalanced RNS, but this requires a very large number of hardware implementations and this is a topic for a separate study. But even in this case, for each system, in order to classify by $\beta$, it will be necessary to first determine acceptable levels of losses. In other words, it would be interesting to theoretically or experimentally determine the threshold for $\beta$ below which the RNS could be considered well balanced, and above which the RNS could be considered poorly balanced. Another interesting area of further research is the question of finding an RNS with a large number of modules (6, 8 etc.) and with $SQ = 2^k$. For example, it can be shown that there exist a unique such 6-tuple with elements less than 500. The main problem in this way is the increasing computational complexity of the search algorithm.

VIII. CONCLUSION

We presented heuristic algorithms for generation of RNS triples with $SO = 2^k - 1$ and quadruples with $SO = 2^k$ for some $k$. Such classification results could be useful for designers of RNS in their search for good performance. Thus, we also discussed performance issues. Our approach is based on careful dealing with the exponent of 2 in the expressions which naturally arise in targeting the corresponding form of $SO$. The measure of RNS balance was proposed. Also, perfectly balanced RNS were defined and found among triples.

APPENDIX A

See Table 2.

APPENDIX B

See Table 3.

Remark: $k$ is even, between 20 and 30.

REFERENCES

[1] C.-H. Chang, A. S. Molahasoeini, A. A. E. Zarandi, and T. F. Tay, “Residue number systems: A new paradigm to data path optimization for low-power and high-performance digital signal processing applications,” IEEE Circuits Syst. Mag., vol. 15, no. 4, pp. 26–44, 4th Quart., 2015.

[2] N. Chervyakov and P. Lyakhov, “RNS-Based Image Processing,” in Embedded Systems Design with Special Arithmetic and Number Systems, Cham, Switzerland: Springer, 2017, pp. 217–245.

[3] S. Asif, O. Andersson, J. Rodrigues, and Y. Kong, “65-nm CMOS low-energy RNS modular multiplier for elliptic-curve cryptography,” IET Comput. Digit. Techn., vol. 12, no. 2, pp. 62–67, Mar. 2018.

[4] N. I. Chervyakov, P. A. Lyakhov, N. N. Nagornov, M. V. Valueva, and G. V. Valuev, “Hardware implementation of a convolutional neural network using calculations in the residue number system,” Comput. Opt., vol. 43, no. 5, pp. 857–868, Oct. 2019.

[5] C. Chang, W. Lee, Y. Liu, B. Goi, R. W. Phan, “Signature gateway: Offloading signature generation to IoT gateway accelerated by GPU,” IEEE Internet Things J., vol. 5, no. 4, pp. 4448–4461, Nov. 2019.

[6] N. Chervyakov, M. Babenko, P. Lyakhov, and I. Lavrenenko, “An approximate method for comparing modular numbers and its application to the division of numbers in residue number systems,” Cybern. Syst. Anal., vol. 50, pp. 977–984, Apr. 2014.

[7] M. Valueva, G. Valuev, N. Semyonova, P. Lyakhov, N. Chervyakov, D. Kaplan, and D. Bogaevskiy, “Construction of residue number system using hardware efficient diagonal function,” Electronics, vol. 8, no. 6, p. 694, Jun. 2019.

[8] N. I. Chervyakov, P. A. Lyakhov, N. F. Semyonova, and M. V. Valueva, “On RNS with VLSI-friendly diagonal function,” in Proc. Int. Multi-Conference Eng., Comput. Inf. Sci. (SIBIRCON), 2017, pp. 131–134.

[9] G. Dimaro, S. Impedovo, and G. Pirlo, “A new technique for fast number comparison in the residue number system,” IEEE Trans. Comput., vol. 42, no. 5, pp. 608–612, May 1993.

[10] S. J. Piestrak, “A note on RNS architectures for the implementation of the diagonal function,” Inf. Process. Lett., vol. 115, no. 4, pp. 453–457, Apr. 2015.

[11] A. Onomdi and B. Premkumar, Residue Number Systems: Theory and Implementation, London, U.K.: Imperial College Press, 2007, p. 296.

[12] K. Ireland, M. Rosen. A Classical Introduction to Modern Number Theory, 2nd ed. New York, NY, USA: Springer-Verlag, 1990.

[13] P. Patronik and S. J. Piestrak, “Design of reverse converters for a new flexible RNS five-Moduli set 2.2+-1.2+1.2+1.2+-1.1+1 Even),” Circuits, Syst., Signal Process., vol. 36, no. 11, pp. 4593–4614, 2017.

[14] R. Chaves and L. Sousa, “Improving residue number system multiplication with more balanced moduli sets and enhanced modular arithmetic structures,” IET Comput. Digit. Techn., vol. 1, no. 5, pp. 472–480, Sep. 2007.

[15] I. A. Kalmykov, A. V. Veligosa, D. I. Kaplan, D. M. Klionsky, and V. V. Gulsansky, “Parallel-pipeline implementation of digital signal processing techniques based on modular codes,” in Proc. XIX IEEE Int. Conf. Soft Comput. Meas. (SCMi), May 2016, pp. 213–214.

[16] D. I. Kaplan, A. V. Voznesenskiy, V. V. Gulsansky, and D. V. Bogaevskiy, “Application of non-positional codes for FIR-filter implementation using computers with CUDA technology,” in Proc. IEEE Conf. Russian Young Researchers Electr. Electron. Eng. (EIConRus), Jan. 2018, pp. 1086–1089.

[17] K. Isupov, “Using floating-point intervals for non-modular computations in residue number system,” IEEE Access, vol. 8, pp. 58603–58619, 2020.

[18] P. Mohan, Residue Number Systems: Theory and Applications, Basel, Switzerland: Birkhauser, 2016.

[19] R. Matos, R. Paludo, N. Chervyakov, P. Lyakhov, and H. Pettenghi, “Efficient implementation of modular multiplication by constants applied to RNS reverse converters,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), Baltimore, MD, USA, May 2017, pp. 1–4.

[20] M. Akkal and P. Siy, “A new mixed radix conversion algorithm MRC-II,” J. Syst. Archit., vol. 53, pp. 577–586, Dec. 2007.

[21] A. S. Molahasoeini, S. Sorouri, and A. A. E. Zarandi, “Research challenges in next-generation residue number system architectures,” in Proc. 7th Int. Conf. Comput. Sci. Educ. (ICCSE), Jul. 2012, pp. 1658–1661.
NIKOLAY I. CHERVYAKOV received the Doctor of Engineering Sciences degree from the Department of Applied Mathematics and Mathematical Modeling, North-Caucasus Federal University. He is currently a Professor and the Head of the Department of Applied Mathematics and Mathematical Modeling, North-Caucasus Federal University. He was an Honored Worker of Science and Technology of the Russian Federation, an Academician of the International Academy of Informatization, and an Expert of the Russian Academy of Sciences. His research interests include modular arithmetic, artificial neural networks, digital image processing, and cryptography.

PAVEL LYAKHOV received the degree and Ph.D. degrees in mathematics from Stavropol State University, in 2009 and 2012, respectively. Since 2012, he has been working with North-Caucasus Federal University. He is the author of over 30 scientific papers. His research interests include high-performance computing, residue number systems, digital signal processing, and image processing.

NATALIYA SEMYONOVA is currently an Associate Professor with the Department of Higher Algebra and Geometry, North-Caucasus Federal University. Her research interests include modular arithmetic, high-performance computing, and signal processing.

ANTON NAZAROV received the bachelor's degree in mathematics from Stavropol State University, in 2011, and the master's degree in applied mathematics and informatics and the Ph.D. degree from NCFU, in 2015 and 2019, respectively. He is currently an Assistant with the Department of Applied Mathematics and Mathematical Modeling, North-Caucasus Federal University. He is the author of over 34 articles. His research interests include modular arithmetic, residue number systems, FPGA, computer arithmetic, high-performance computing, and fault-tolerant computing.

MARIA VALUEVA (Graduate Student Member, IEEE) received the bachelor’s and master’s degrees in applied mathematics and computer science from North-Caucasus Federal University, in 2016 and 2018, respectively, where she is currently pursuing the master’s degree with the Department of Applied Mathematics and Mathematical Modeling. Her research interests include high-performance computing, modular arithmetic, artificial neural networks, and digital image processing.

GEORGI BOYVALENKOV received the B.Sc. degree from Sofia University, in 2012. He is currently a Senior Software Developer/Engineer with IObosch. His research interests include the Internet of Things, device management, and OSGi.

DANIL BOGAEVSKIY received the M.S. degree in software engineering and computer applications (applied mathematics and informatics) from the Faculty of Computer Science and Technology, Saint Petersburg Electrotechnical University “LETI”, in 2017. He is currently pursuing the Ph.D. with the Department of Automation and Control Processes, Saint Petersburg Electrotechnical University “LETI”. He is the author of more than ten articles.

DMITRI KAPLUN (Member, IEEE) received the Ph.D. degree from Saint Petersburg Electrotechnical University “LETI”, Saint Petersburg, Russia, in 2009. He was an Associate Professor from Saint Petersburg Electrotechnical University “LETI”, in 2015. He is currently a Senior Researcher with Saint Petersburg Electrotechnical University “LETI”. He defended his Ph.D. thesis in digital signal processing with Saint Petersburg Electrotechnical University “LETI”, in 2009. He regularly takes part in different collaborative projects related with signal processing, artificial neural networks, and software-hardware implementation of digital signal processing algorithms. His most substantial results are in the fields of digital signal and image processing, embedded systems, artificial neural networks, and machine learning. He is the author of more than 50 papers in journals and conference proceedings. His current research and academic work are concerned with digital signal and image processing, radio monitoring and hydroacoustic monitoring applications, embedded and reconfigurable systems, computer vision, and machine learning.

***