Educational Administration: Theory and Practice
2024, 30(5), 2305 - 2316
ISSN: 2148-2403
https://kuey.net/

Research Article

Efficient Implementation Of The Sum Of Residues Modular Reduction Using Arithmetic-Friendly RNS Moduli Set

Danial Alvani[1*], Mohammad Esmaeildoust[2], Amer Kaabi[3]

[1*]Faculty of Marine Engineering, Khorramshahr University of Marine Science and Technology, Khorramshahr, Iran,
Email: danial.alvani@kmsu.ac.ir
[2]Faculty of Marine Engineering, Khorramshahr University of Marine Science and Technology, Khorramshahr, Iran,
Email: m_doust@kmsu.ac.ir
[3]Department of Basic Sciences, Abadan Faculty of Petroleum Engineering, Petroleum University of Technology, Abadan, Iran,
Email: kabbi_amer@put.ac.ir

***Corresponding Author:** Danial Alvani
*Faculty of Marine Engineering, Khorramshahr University of Marine Science and Technology, Khorramshahr, Iran,
Email: danial.alvani@kmsu.ac.ir

Citation: Danial Alvani, et.al (2024), Efficient Implementation Of The Sum Of Residues Modular Reduction Using Arithmetic-Friendly RNS Moduli Set, *Educational Administration: Theory and Practice*, *30*(4), 2305 - 2316
Doi: 10.53555/kuey.v30i5.3278

ARTICLE INFO **ABSTRACT**

Introduction: Due to the importance of public-key cryptography in information and communication security, it is widely employed in various applications for secure communication.
Materials and Methods: multiplication and exponentiation of large numbers are used in cryptography algorithms such as RSA, ElGamal, and elliptic curve cryptography.
Results and Discussion: The residue number system in these algorithms is very efficient since calculations are performed on small residues, resulting in a fast arithmetic operation, as well as lower power consumption. The modular reduction of large numbers is one of the main operations in most public-key cryptography systems, which includes large computations in finite fields. This paper presents an efficient implementation of modular reduction. To this end, arithmetic-friendly moduli were selected and employed in the implementation of the improved sum of residues reduction algorithm. The SOR algorithm using the proposed moduli set was described in VHDL language and synthesized on the Xilinx virtex7 FPGA family using ISE14.7 software.
Conclusion: The results showed that, compared to the recent similar works, the implementation of the improved sum of residues algorithm using the proposed moduli set has achieved higher speed and uses less hardware resources.

Keywords: residue number system (RNS); modular reduction; modular multiplication; sum of residues (SOR) reduction; arithmetic residue.

INTRODUCTION

Public key cryptography algorithms have an important role in information and communication security and are widely employed in various applications [1-2]. Modular addition and multiplication are the most basic components of these algorithms. With the growth of the complexity of algorithms, computational cost dramatically increases. Nowadays, with the proliferation of portable computers and electronic devices as well as the use of general-purpose or special-purpose processors, low-power and high-speed computing are very much needed. The Residue Number System (RNS) is a non-weighted number system [3], which provides parallel and fast computational operations with high accuracy. In this system, calculations are performed on residues. Many works used RNS for implementing public-key cryptography algorithms such as RSA [4-5], ElGamal [6], and ECC [7-10] where modular multiplication and exponentiation are used for very large numbers. Therefore, one of the ways to significantly increase the speed and decrease the computing power in these algorithms is to utilize the residue number system. RNS usually is used for applications such as digital signal processing [11, 12], digital filters [13, 14], image processing [15, 16], and error correction systems [17, 18].

The Montgomery modular multiplication [19] is one of the efficient methods for modular multiplication and exponentiation in public-key cryptography algorithms. For simultaneous use of the advantages of the residue number system and Montgomery modular multiplication, RNS Montgomery modular multiplication is presented in [20]. A new algorithm and VLSI architecture for RNS Montgomery modular multiplication are stated in [21] and a survey of Montgomery reduction in the context of RNS arithmetic is reported in [22]. In [23] a modern architecture which comes with two new well-formed 4-moduli RNS bases $\{2^{n-2}+1, 2^{n-3}-1, 2^{n-3}+1, 2^{n-5}-1\}$, $\{2^{n}+1, 2^{n}, 2^{n-1}-1, 2^{n-1}+1\}$, for performing RNS Montgomery modular multiplication is offered. The main advantage of the RNS Montgomery modular multiplication method is the efficiency of this procedure in using hardware resources. The sum of residues algorithm is one of the new and important methods to perform modular reduction which was first introduced in [24]. Hardware implementation of the sum of residues algorithm was proposed later in [25]. The proposed implementation in [25] is very large in terms of area. In [10], the sum of residues algorithm [24] was employed to perform modular reduction, and general RNS moduli were chosen to implement the ECC processor. Further, by modified series/parallel implementation of the sum of residues, the proposed ECC processor has become more effective. In [10], the absence of balanced RNS moduli set was observed significantly. In [27], by introducing the correction factor k to obtain an accurate result, the sum of residues algorithm was improved. Furthermore, by using a balanced and efficient 8-moduli set $\{2^{66}-1, 2^{66}-2^{2}-1, 2^{66}-2^{3}-1, 2^{66}-2^{4}-1, 2^{66}-2^{5}-1, 2^{66}-2^{6}-1, 2^{66}-2^{8}-1, 2^{66}-2^{9}-1\}$, a new design was represented for improving the area compared to [25], as well as the timing of its design, was improved compared to the RNS Montgomery modular multiplication. In addition, in [27], two implementations were performed for the 256-bit prime field of the elliptic curve SEC2P256K1 and the 255-bit prime field of the ED25519 elliptic curve. In [28], a hardware architecture based on the residue number system that supports quick elliptic curve point doubling, point tripling, and point addition, based on the chosen 8-moduli set in [27], is presented.
Moduli in the forms of 2^{n} and $2^{n}\pm1$ can reduce the required arithmetic operations, leading to an efficient implementation of hardware in the reside number system [3]. The use of well-formed and balanced moduli in the forms 2^{n} and $2^{n}\pm1$ can significantly improve the system performance. In [9], parts of RNS bases were chosen in the forms 2^{n} and $2^{n}\pm1$ for achieving higher efficiency. In this paper, to obtain the advantages of the moduli in the forms 2^{n} and $2^{n}\pm1$ and the hardware implementation of the improved sum of residues reduction algorithm, the new 8-moduli set is selected as $\{2^{d+6}-1, 2^{d+4}-1, 2^{d}-1, 2^{d-2}-1, 2^{d-3}-1, 2^{d-4}-1, 2^{d-6}-1, 2^{d-8}-1\}$ where $d = 6b+1$, $b = 2, 3, 4, \ldots, 13$, and $\langle b \rangle_5 \neq 0, 2, 3$.
The novelties of the article are as follows:
1- Selection of the new and balanced 8-moduli set $\{2^{d+6}-1, 2^{d+4}-1, 2^{d}-1, 2^{d-2}-1, 2^{d-3}-1, 2^{d-4}-1, 2^{d-6}-1, 2^{d-8}-1\}$.
2- Hardware implementation of improved sum of residues reduction algorithm with new 8-moduli set.
The rest of this paper is organized as follows: In section 2, the related mathematical background is presented. Section 3 explains the proposed RNS moduli selection for SOR. In section 4, a performance evaluation with recent works is presented. Finally, section 5 concludes the paper.

MATHEMATICAL BACKGROUND

RNS background

The RNS is described in terms of relatively prime moduli set $\{m_1, m_2, \ldots, m_n\}$ where gcd $(m_i, m_j) = 1$ for $i \neq j$. A weighted number X can be displayed as $X = (x_1, x_2, \ldots x_n)$, where,

$$x_i = X \mod m_i = \langle X \rangle_{m_i}, \quad 0 \leq x_i < m_i. \qquad (1)$$

Such a representation is unique for any integer X in the range [0, M-1], where M is the dynamic range of the moduli set $\{m_1, m_2, \ldots, m_n\}$, which is equal to the product of m_i terms $(M = m_1 \times m_2 \times \ldots \times m_n)$ [29]. The RNS generally includes three sections: the forward converter, arithmetic unit, and reverse converter [3]. In RNS, the weighted numbers are converted to their equivalent residue numbers by a forward converter [30]. The arithmetic unit of the residue number system includes the modular adder, multiplier, and subtractor for each modulus channel [31-32]. The residue numbers are converted to their weighted equivalents in the binary system by a reverse converter to utilize the outcomes of arithmetic operations [33-34]. Reverse converter algorithms are basically based on the mixed-radix conversion (MRC) [35], Chinese remainder theorem (CRT) [36-37], new Chinese remainder-1 [38-39], and new Chinese remainder-2 [40].

Chinese remainder theorem

The Chinese Remainder Theorem (CRT) [41] may be considered as one of the most fundamental results in the theory of residue number systems. Computing weighted number X from its RNS representation, i.e., $(x_1, x_2, \ldots, x_n)$, based on the moduli set $\{m_1, m_2, \ldots, m_n\}$ is as follows:

$$X = \left\langle \sum_{i=1}^{n} \langle x_i N_i \rangle_{m_i} M_i \right\rangle_M \qquad (2)$$

Where $M = m_1 \times m_2 \times \ldots \times m_n$, $M_i = M / m_i$, $N_i = \left|M_i^{-1}\right|_{m_i}$ is the multiplicative inverse of M_i, $i = 1, 2, \ldots, n$.

Sum of residues reduction background

Sum of residues RNS modular multiplication algorithm is a novel algorithm to perform modular multiplication in the residue number system [24-25]. In SOR, calculations are performed in RNS modules. SOR [24] is a rival to Montgomery modular multiplication [19]. CRT [41] is used to Conclude a RNS algorithm for the sum of residues reduction [10,24]. A brief description of the sum of residues reduction is provided for understanding [24].

Display of the integer X, $0 \le X < m$, using CRT is given in Eq. (2). Assume two l-bit integers, X and Y. Then the multiplication outcome $Z = X \times Y$ is a $2l$-bit integer.

The presentation of Z in the residue number system is:

$$RNS(Z) = \{z_1, z_2, \ldots, z_N\} \qquad (3)$$

Where, $z_i = \left|x_i . y_i\right|_{m_i}$, and N is equal to the number of moduli.

The CRT performs condition $Z<M$. otherwise, the N-tuple RNS set in Eq. (3) doesn't display integer Z. Defining $\gamma_i = \left|z_i M_i^{-1}\right|_{m_i}$, the integer Z can be offered as:

$$Z = \left\langle \sum_{i=1}^{N} \gamma_i M_i \right\rangle_M \qquad (4)$$

The α is an integer coefficient, which can be computed such as follows [42]:

$$Z = \sum_{i=1}^{N} \gamma_i M_i - \alpha M \qquad (5)$$

The reduction of Z by the modulus p is shown as follows:

$$Z \bmod p = \langle Z \rangle_p = \left\langle \sum_{i=1}^{N} \gamma_i M_i \right\rangle_p - \langle \alpha M \rangle_p . \qquad (6)$$

The computation of α has been debated in [25-43]. It is shown that selecting suitable constants q, Δ and performing boundary condition of Eq. (7), α can be computed using Eq. (8).

$$0 \le X < (1-\Delta)M . \qquad (7)$$

$$\alpha = \left\lfloor \frac{1}{2^q} \sum_{i=1}^{N} \left\lfloor \frac{\gamma_i}{2^{n-q}} \right\rfloor + 2^q . \Delta \right\rfloor \qquad (8)$$

In Eq. (8), Δ is a constant-point rectification term and q is an integer constant that determines the number of bits shorted of γ_i terms in the sum.

Improved Sum of residues reduction algorithm

The improved sum of residues reduction algorithm [27] is presented to compute the accurate value of "X mod p" straightly in the RNS representation of an integer.

Algorithm 1 shows the RNS modulus p multiplication $\{x_1, x_2, \ldots, x_N\} \times \{y_1, y_2, \ldots, y_N\} \bmod p$ over chosen moduli set using improved sum of residues manner [27].

Algorithm 1: improved sum of residues reduction

```
Require: $p, \Delta, q, \beta = \{m_1, \ldots, m_N\}, m_1 > m_2 > \cdots > m_N, n = \lceil \log_2 m_1 \rceil$
$W = \lceil \log_2 p \rceil, T, N \ge \left\lceil \frac{2W}{n} \right\rceil$
Require: $M = \prod_{i=1}^{N} m_i, \hat{M} = (1-\Delta)M, M_i = \frac{M}{m_i}$ for $i = 1$ to N
```

Require: pre-computed tables $\begin{bmatrix} \langle M_1^{-1} \rangle_{m_1} \\ \langle M_2^{-1} \rangle_{m_2} \\ \vdots \\ \langle M_N^{-1} \rangle_{m_N} \end{bmatrix}$, $\begin{bmatrix} \langle -p \rangle_{m_1} \\ \langle -p \rangle_{m_2} \\ \vdots \\ \langle -p \rangle_{m_N} \end{bmatrix}$, and $\begin{bmatrix} \left\lfloor \frac{\langle M_1 \rangle_p}{2^{W-T}} \right\rfloor \\ \vdots \\ \left\lfloor \frac{\langle M_N \rangle_p}{2^{W-T}} \right\rfloor \end{bmatrix}$

Require: pre-computed table $\begin{bmatrix} \langle \langle M_i \rangle_p \rangle_{m_1} \\ \langle \langle M_i \rangle_p \rangle_{m_2} \\ \vdots \\ \langle \langle M_i \rangle_p \rangle_{m_N} \end{bmatrix}$ for $i = 1$ to N.

Require: pre-computed table $\begin{bmatrix} \langle \alpha.\langle -M \rangle_p \rangle_{m_1} \\ \langle \alpha.\langle -M \rangle_p \rangle_{m_2} \\ \vdots \\ \langle \alpha.\langle -M \rangle_p \rangle_{m_N} \end{bmatrix}$ for $\alpha = 1$ to $N - 1$

input : Integers X and Y, $0 \le X, Y < \hat{M}$ in form of RNS: $\{x_1, \cdots, x_N\}$ and $\{y_1, \cdots, y_N\}$.
output : presentation of $Z = XY \mod p$ in RNS: $\{z_1, \cdots, z_N\}$.

```
1. for $i = 1$ to $N$ do
   | $xy_i \leftarrow \langle x_i . y_i \rangle_{m_i}$.
end
2. for $i = 1$ to $N$ do
   | $\gamma_i \leftarrow \left\langle x.y_i \langle M_i^{-1} \rangle_{m_i} \right\rangle_{m_i}$.
end
3. for $i = 1$ to $N$ do
   | for $j = 1$ to $N$ do
   |   | $\Upsilon_{ij} \leftarrow \gamma_i \left\langle \langle M_i \rangle_p \right\rangle_{m_j}$.
   | end
end
4. for $i = 1$ to $N$ do
   | 4.1 $\alpha \leftarrow \left\lfloor \frac{1}{2^q} (\sum_{i=1}^{N} \left\lfloor \frac{\gamma_i}{2^{n-q}} \right\rfloor + 2^q \Delta) \right\rfloor$.
   | 4.2 $k \leftarrow \left\lfloor \frac{1}{2^T} \sum_{i=1}^{N} \gamma_i \left\lfloor \frac{\langle M_i \rangle_p}{2^{W-T}} \right\rfloor \right\rfloor$.
end
```

```
5. for $i = 1$ to $N$ do
    5.1 Calculate $\langle k.\langle -p\rangle_{m_i}\rangle_{m_i}$ .
    5.2 Read $\langle \alpha\langle -M\rangle_p\rangle_{m_i}$ from the table.
    5.3 $\text{sum}_i \leftarrow \left\langle \sum_{j=1}^{N} Y_{ji}\right\rangle_{m_i}$ .
end
6. for $i = 1$ to $N$ do
    $z_i \leftarrow sum_i + \langle \alpha\langle -M\rangle_P\rangle_{m_i} + \langle k\,\langle -p\rangle_{m_i}\rangle_{m_i}$ .
end
```

RNS moduli selection for SOR

As discussed in section 1, many works are done to select efficient RNS moduli set for effective implementation of SOR. In the work reported in [25], the *N* general moduli are chosen with same word length. This shows that dynamic range of the residue number system is equally dispensed into *N* moduli. The SOR implementation of [25], later is employed in the implementation of elliptic curve point multiplication reported in [10]. This work only concentrated on general moduli to show that rapid implementation of modular multiplication and missed the special attributes of the well-formed moduli. In [27], for efficient implementation of the SOR, moduli set in form of $2^n - 2^{t_i} - 1$ (*n*=66) are selected. Moduli in the form of $2^n - 2^{t_i} - 1$, first presented in [44] and modular reduction in RNS addition and multiplication can be realized in a adder base structure.
In order to improve efficiency of SOR implementation and employ the property of moduli in the form of 2^n-1, a new 8-moduli set, namely β = {2^{d+6}-1, 2^{d+4}-1, 2^d-1, 2^{d-2}-1, 2^{d-3}-1, 2^{d-4}-1, 2^{d-6}-1, 2^{d-8}-1} where, $d = 6b+1$, b= 2, 3, 4, …, 13, and $\langle b\rangle_5 \neq 0,2,3$, are selected. With substituting b = 11, moduli set {2^{73}-1, 2^{71}-1, 2^{67}-1, 2^{65}-1, 2^{64}-1, 2^{63}-1, 2^{61}-1, 2^{59}-1} is resulted which provides 523- bit dynamic range. Since RNS modular multiplication in the 256-bit prime field requires a dynamic range of at least 512 bits, the provided 523- bit dynamic range makes it suitable for RNS modular multiplication in 256- bit prime field. The selected moduli set for 256- bit prime field are shown in Table 1.

In the following, theorem to prove that the selected modules are relatively prime is included.
Theorem 1. Let $a, b \in \mathbb{Z}$. If there exist integers x and y such that $ax+by$=1 then gcd (a, b) = 1.
Proof. Let $a, b \in \mathbb{Z}$ such that d= gcd (a, b). Then $d \mid a$ and $d \mid b$.
Hence $d \mid (ax + by)$, thus $d \mid 1$. Which implies $d = \pm 1$, since gcd is the greatest, d=1.
Because modules are large numbers, a Python program is coded for calculations and verification.
Due to the fact that the selected modules are in the form 2^n-1, it leads to fast and simple arithmetic operations in RNS [45-46] compared to the modules provided in [25] and [27].

Table 1. Co-prime moduli set β

2^{73}-1	2^{71}-1	2^{67}-1	2^{65}-1
2^{64}-1	2^{63}-1	2^{61}-1	2^{59}-1

Proposed RNS adder and multiplier circuits

Modular addition is basic operation in residue number system, since the modular adders are essential building blocks for sketching modular multipliers and as well as modular subtraction can be perform by using modular adders [3].
Figure. 1-(a), shows the design of a *n*-bit RNS adders.
The formula for performing the (A+B mod 2^n-1) is [3]:

$$(A+B)\bmod\ (2^n-1) = \begin{cases} (A+B+1)\bmod\ 2^n & \text{if } A+B+1 \geq 2^n \\ A+B & \text{if } A+B+1 < 2^n \end{cases} \tag{9}$$

Due to the condition $x + y + 1 \geq 2^n$, both additions in Eq. (9) are performed in parallel, and the correct answer is selected by a multiplexer as shown in figure 1-(a).

In this paper, n = 73 is considered.

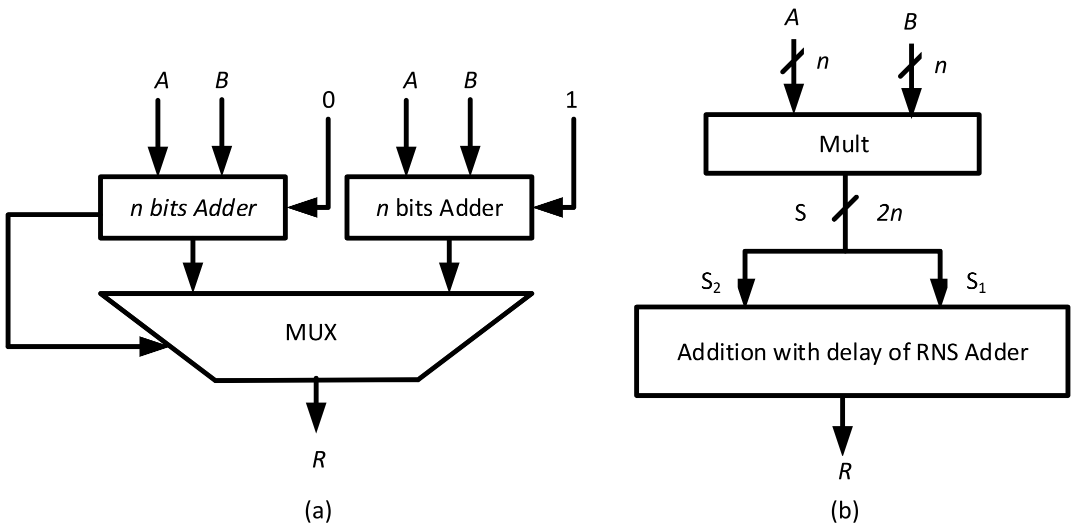

Figure 1. a) Modulus 2^n-1 RNS adder, b) Modulus 2^n-1 RNS multiplier

Figure. 1-(b), shows the design of a n-bit RNS multiplier circuit.
The formula for performing the multiplication circuit is

$$S = A \times B \tag{10}$$

The result S has $2n$-bit length and divided into two n-bit as S_1 and S_2.
Since moduli in the form of 2^n-1 are selected, RNS adders and multiplier circuits are less complex compared to the circuits employed in [27] for moduli in the form of $2^n - 2^{t_i} - 1$ (n = 66). RNS adder and multiplier for n =73 (largest selected moduli) are implemented on the different Xilinx FPGAs family which is shown in Table 2. The results are listed based on maximum logic, net and combinational delays of the RNS adder and multiplier. Although the selected moduli set have higher bit length compared to moduli set selected in [27], the simple hardware results in noticeable improvement in delay of RNS addition and multiplication.

Table 2. Implementation results of SOR components on ARTIX7 & VIRTEX7 FPGAs series.

Unit	Device	Max. Logic Delay (ns)	Max. Net Delay (ns)	Combinational Delay (ns)
RNS Multiplier [27]	ARTIX 7	16.206	5.112	21.318
RNS Multiplier (n =73)	ARTIX 7	14.987	2.790	17.777
RNS Adder [27]	ARTIX 7	6.017	2.303	8.32
RNS Adder (n =73)	ARTIX 7	3.724	1.550	5.274
RNS Multiplier [27]	VIRTEX 7	11.525	3.793	15.264
RNS Multiplier (n =73)	VIRTEX 7	10.58	2.499	13.079
RNS Adder [27]	VIRTEX 7	3.931	1.469	5.4
RNS Adder (n=73)	VIRTEX 7	2.087	1.393	3.481

In order to achieve efficient modular multiplication on FPGA, DSP modules are used for implementation of 73×73-bit, 71×71-bit, 67×67-bit, 65×65-bit, 64×64-bit, 63×63-bit, 61×61-bit and 59×59-bit multipliers, that are followed by a combinational reduction logic to construct the RNS multiplier. The total number of 116 DSP resources are used for a RNS multiplier. Table 3 presents number of DSP 48E1s for multipliers.

Table 3. Number of DSP 48E1s for multipliers

Multiplier	DSP 48E1s
73×73-bit	20
71×71-bit	20
67×67-bit	16

65×65-bit	12
64×64-bit	12
63×63-bit	12
61×61-bit	12
59×59-bit	12

Numerical Example

The algorithm 1 can be more clarify with the help of a numerical example with the following inputs and pre-computations. The python program is used to attain algorithm outputs.

Inputs:

$T = 72$, $q = 8$, $\Delta = 1/2^4$, $n = 73$ (largest moduli in selected RNS moduli set), $N = 8$, $w = 256$,

$p = 2^{256} - 2^{32} - 977$

$\Rightarrow p$=115792089237316195423570985008687907853269984665640564039457584007908834671663

$X = 2^{256} - 2^{35} - 977$

$\Rightarrow X$=115792089237316195423570985008687907853269984665640564039457584007878769900591

$Y = 2^{256} - 2^{37} - 977$

$\Rightarrow Y$=115792089237316195423570985008687907853269984665640564039457584007775690685487

Z=$X \times Y$=13407807929942597099574024998205846127479365820592393377723561443701871100390897768745943162383372889379037440209302592119473661603124793281120775636422817

Moduli = $\{2^{d+6}-1, 2^{d+4}-1, 2^{d}-1, 2^{d-2}-1, 2^{d-3}-1, 2^{d-4}-1, 2^{d-6}-1, 2^{d-8}-1\}$ when $d = 67$

As a result, the moduli set is equal to

Moduli = (9444732965739290427391, 2361183241434822606847, 147573952589676412927, 36893488147419103231, 18446744073709551615, 9223372036854775807, 2305843009213693951, 576460752303423487)

Pre-computations:

$M = m_1 \times m_2 \times \ldots \times$ m8

$\Rightarrow M$=27459190640522438794974387438372958734738904476932930589760680023722449847623641334027717419099877483026047907644978036993241544937013360639638063158042558465

$M_i = M / m_i$ (for i = 1 to 8)

$\Rightarrow M_i$=
(29073548971824275553204138375875441908591661777802903028608387254731036809437107039550467831103459337125896795533952565739688154009436 15,,
47634102635436893148996612957022260346580939661181848074937606749193991788617841420843036101050378169680592236974730005947115434580373405695)

$\langle M_i^{-1} \rangle_{m_i}$ = (5386417757290016831506, 9305007039937802644 55, 1880316078530007173 3, 2047261872306870902 6, 3400417297457863576, 3683224619310100934, 540231240598819487, and 308591480262646799)

$x_i = \langle X \rangle_{m_i}$

x_i = (103079214127, 8761733282863, 36028762659224623, 2305842974853954607, 18446744039349812271, 9223372002495036478, 2305842974853958702, 576460717944732718)

$y_i = \langle Y \rangle_{m_i}$

y_i = (9444732965739290426414, 8658654067759, 36028659580009519, 2305842871774739503, 18446743936270597167, 9223371899415821374, 2305842871774743598, 576460614865517614)

Step1:

for i=1 to N

$$xy_i \leftarrow \langle x_i . y_i \rangle_{m_i}$$

$$xy_1 \leftarrow 9444732865030898225312$$

$$xy_2 \leftarrow 2361166221716380126754$$

$$xy_3 \leftarrow 77173859857994780864$$

$$xy_4 \leftarrow 32426085153653035174$$

$$xy_5 \leftarrow 167675524188672$$

$$xy_6 \leftarrow 165098543782273$$

$$xy_7 \leftarrow 2305307169103575200$$

$$xy_8 \leftarrow 396485711994204320$$

Step 2:
For i=1 to N

$$\gamma_i \leftarrow \left\langle xy_i \left\langle M_i^{-1} \right\rangle_{m_i} \right\rangle_{m_i}$$

$$\gamma_1 \leftarrow 2409161764343951463886$$
$$\gamma_2 \leftarrow 1135107945428528244106$$
$$\gamma_3 \leftarrow 135899592520116905117$$
$$\gamma_4 \leftarrow 789822594998113784 9$$
$$\gamma_5 \leftarrow 5936354770407377172$$
$$\gamma_6 \leftarrow 865739859861763338$$
$$\gamma_7 \leftarrow 1313261649358731301$$
$$\gamma_8 \leftarrow 83291400022900075$$

Step 3:
For i= 1 to N
For j=1 to N

$$Y_{ij} \leftarrow \gamma_i \left\langle \left\langle M_i \right\rangle_p \right\rangle_{m_j}$$

$Y_{11} \leftarrow 681867765270114117420133393408490086262906$, $Y_{12} \leftarrow 14110999000615748585725027912095192 02960572$
$Y_{21} \leftarrow 2659261440602693086991442234037531836892694$, $Y_{22} \leftarrow 13767849490879109300725036976658737031 43752$
$Y_{31} \leftarrow 980204674115271573830804255676847542450458$, $Y_{32} \leftarrow 4704508928771512517555434529802326450 0134$
$Y_{41} \leftarrow 5974525928778090626338736042222879520850$, $Y_{42} \leftarrow 621939042396340198485819794393968992 2946$
$Y_{51} \leftarrow 3613829521572029835320595981293910046000 8$, $Y_{52} \leftarrow 5607603186449701501359797610557855591304$
$Y_{61} \leftarrow 1852740078360359150364953302955205235416$, $Y_{62} \leftarrow 1499065839605647225112824798436085160056$
$Y_{71} \leftarrow 5039251287888810431290568139652218946652$, $Y_{72} \leftarrow 1283872360271851651556491573291643483678$
$Y_{81} \leftarrow 491689769989300091524375965533261748475$, $Y_{82} \leftarrow 1245001950254418850036006829160151738 75$

$Y_{13} \leftarrow 2771567538788877003155946711337771481478 54$, $Y_{14} \leftarrow 6031455705067204460454919467162898619 8936$
$Y_{23} \leftarrow 11321320756411111666047826669757276652514$, $Y_{24} \leftarrow 14676688015072631794298638578584883620874$
$Y_{33} \leftarrow 16134696147550311472725944718296707584618$, $Y_{34} \leftarrow 3691070020767749267947470135952686755595$
$Y_{43} \leftarrow 6775868829898882661684615485811483 59269$, $Y_{44} \leftarrow 2727648289395755373018329982814870182 74$
$Y_{53} \leftarrow 7095629195915895439147504983174863742 84$, $Y_{54} \leftarrow 408219245897641520342234385942423143 64$
$Y_{63} \leftarrow 2867420084858173731171677179416206146 2$, $Y_{64} \leftarrow 192936980594767590662814718044158563 8$
$Y_{73} \leftarrow 8741293408750378537250368265717169558$, $Y_{74} \leftarrow 2081156555452729991347494319775665372 1$
$Y_{83} \leftarrow 611119128411221256767698377804925825$, $Y_{84} \leftarrow 6717415463676356444102345540271959 00$
$Y_{15} \leftarrow 2777544291298458401581887747504380829296$, $Y_{16} \leftarrow 1336705836095123673701830761452697007544 0$
$Y_{25} \leftarrow 13086337797715795213691802211956674897 68$, $Y_{26} \leftarrow 1063276267409231305401892130362762706924$
$Y_{35} \leftarrow 2036758186019779579201711235058398273323$, $Y_{36} \leftarrow 9731768706953119432203237321469874064$
$Y_{45} \leftarrow 3640477820883153884265384523917729261 9$, $Y_{46} \leftarrow 3869036675509395972544059701011751768 1$
$Y_{55} \leftarrow 3099130368167116439847095760223557237 6$, $Y_{56} \leftarrow 3413446951590187223582237943761751396$
$Y_{65} \leftarrow 1596756489595295087973896302686573276 0$, $Y_{66} \leftarrow 6114140002827327687978607407424477548$
$Y_{75} \leftarrow 2422143162596688793930291568729629246 3$, $Y_{76} \leftarrow 6054263937375943593673675653646583405$
$Y_{85} \leftarrow 1536200930104473728887011569798006725$, $Y_{86} \leftarrow 7680940228013972297564272809122581 75$

$Y_{17} \leftarrow 2256773885564848812199077039693822416028$, $Y_{18} \leftarrow 867990622203321764436991473908291277512$
$Y_{27} \leftarrow 1063300957183257316768953115869437506490$, $Y_{28} \leftarrow 4089631622718745699839063778616024843 62$
$Y_{37} \leftarrow 1664571068777580863528244649252574415 61$, $Y_{38} \leftarrow 978821942552281951598068595423104791 9$
$Y_{47} \leftarrow 227382519365801494827103747718351516 9$, $Y_{48} \leftarrow 227583080737841227639926951491027839 7$
$Y_{57} \leftarrow 341983236725666692581846583584114038 4$, $Y_{58} \leftarrow 342150947533640860247972157649691577 6$
$Y_{67} \leftarrow 997786237018025833654249540432449978$, $Y_{68} \leftarrow 498978312413921098839608395114602468$
$Y_{77} \leftarrow 6706425526817308951346249761570787 95$, $Y_{78} \leftarrow 756907429454713464804174183322887736$
$Y_{87} \leftarrow 192022288185827710460559323881629025$, $Y_{88} \leftarrow 327842200944202103585696774150642 75$

Step 4:
For *i*=1 to *N*

$$4.1\ \alpha \leftarrow \left\lfloor \frac{1}{2^q}\left(\sum_{i=1}^{N}\left\lfloor \frac{\gamma_i}{2^{n-q}}\right\rfloor + 2^q\Delta\right)\right\rfloor.$$

$\alpha \leftarrow 0$

$$4.2\ k \leftarrow \left\lfloor \frac{1}{2^T}\sum_{i=1}^{N}\gamma_i\left\lfloor \frac{|M_i|_p}{2^{W-T}}\right\rfloor\right\rfloor.$$

$k \leftarrow 18464022786947346774 77$

Step 5:
For *i*=1 to *N*

$$5.1\ \text{Calculate } \left\langle k.\langle -p\rangle_{m_i}\right\rangle_{m_i}.$$

Phase 1.1 = 82834665965104427875 59, phase 1.2 = 37418006676867636279 7
Phase 1.3 = 9239368927439859188 2, phase 1.4 = 2028256847905996249 1
Phase 1.5 = 1635684631875253669 9, phase 1.6 = 888552096566847584 1
Phase 1.7 = 1215072663256523127, phase 1.8 = 428518828954403760

$$5.2\ \text{Read } \left\langle \alpha\langle -M\rangle_p\right\rangle_{m_i} \text{ from the table.}$$

phase 2.1=0, phase 2.2=0, phase 2.3=0, phase 2.4=0, phase 2.5=0, phase 2.6=0, phase 2.7=0, phase 2.8=0

$$5.3\ \text{sum}_i \leftarrow \left\langle \sum_{j=1}^{N} Y_{ji}\right\rangle_{m_i}.$$

$$\text{sum}_1 \leftarrow 5217897958331825282912$$
$$\text{sum}_2 \leftarrow 1761286061702129779386$$
$$\text{sum}_3 \leftarrow 751360334343319140 86$$
$$\text{sum}_4 \leftarrow 1305362183577013839 8$$
$$\text{sum}_5 \leftarrow 1713859891008834642 0$$
$$\text{sum}_6 \leftarrow 2195256151132184606$$
$$\text{sum}_7 \leftarrow 531093517150379851$$
$$\text{sum}_8 \leftarrow 271871109103727507$$

Step 6:
For *i*=1 to *N*

$$z_i \leftarrow sum_i + \left\langle \alpha\langle -M\rangle_p\right\rangle_{m_i} + \left\langle k\langle -p\rangle_{m_i}\right\rangle_{m_i}.$$

z_1 = 13501364554842268070471, z_2 = 2135466128470806142183, z_3 = 1675297227087305059 68
z_4 = 333361903148301008 89, z_5 = 3349544522884088311 9, z_6 = 1108077711680066044 7
z_7 = 1746166180406902978, z_8 = 700389938058131267
The results are verified as follows:

$$\left\langle \left\langle Z \right\rangle_P \right\rangle_{m_1} = 13501364554842268070471$$

$$\left\langle \left\langle Z \right\rangle_P \right\rangle_{m_2} = 2135466128470806142183$$

$$\left\langle \left\langle Z \right\rangle_P \right\rangle_{m_3} = 1675297227087305059 68$$

$$\left\langle \left\langle Z \right\rangle_P \right\rangle_{m_4} = 3333619031483010088 9$$

$$\left\langle \left\langle Z \right\rangle_P \right\rangle_{m_5} = 3349544522884088311 9$$

$$\left\langle \left\langle Z \right\rangle_P \right\rangle_{m_6} = 1108077711680066044 7$$

$$\left\langle \left\langle Z \right\rangle_P \right\rangle_{m_7} = 1746166180406902978$$

$$\left\langle \left\langle Z \right\rangle_P \right\rangle_{m_8} = 700389938058131267$$

Hardware implementation of the sum of residues reduction algorithm

Three different architectures for implementation of the SOR algorithm discussed in Section 2 are introduced in [27], named non-pipe-lined (SOR_1M_N), pipe-lined (SOR_1M_P), and two parallel pipelined (SOR_2M). In this paper, these architectures are implemented using the proposed 8-moduli set. Table 4, shows a comparison between the implementation of SOR algorithm using the proposed moduli set and the most advanced RNS-based modular multipliers.

Table 4. Comparison of 256-bit Modular Multipliers

Design	Platform	Latency (ns)	Area (KLUTs), (DSP)
(MM-PA-P) [25]	VIRTEX-6	14.20	(36.5), (2016)
(MM-PA-N) [25]	VIRTEX-6	47.25	(34.34), (2016)
(MM-PA-P) [26]	VIRTEX-7	48.3	(29.17), (2799)
(MM-SPA) [26]	VIRTEX-7	239.2	(11.43), (512)
(SOR-1M-N) [27]	VIRTEX-7	241	(8.17), (140)
(SOR-1M-P) [27]	VIRTEX-7	173	(8.73), (140)
(SOR-2M) [27]	VIRTEX-7	140	(10.11), (280)
[23]	VIRTEX-7	120.16	(9.21), (248)
(SOR-1M-N) with Proposed Moduli set	VIRTEX-7	197.2	(7.16), (116)
(SOR-1M-P) with Proposed Moduli set	VIRTEX-7	132.6	(7.57), (116)
(SOR-2M) with Proposed Moduli set	VIRTEX-7	105.7	(8.93), (232)

In the designs proposed in [25-26], the SOR algorithm introduced in [24] was used to perform the modular reduction. Barrett reduction [47], was used in these two designs for modular multiplication at any channel. As stated in [27], the Barrett reduction involves two multiplications and one subtraction, which isn't an optimal solution for high-speed designs. The design in [25] has a simultaneous structure to execute the modular reduction in one clock cycle. The hardware needed in this design is presented in [26], which is equal to (34.34 KLUTs, 2016 DSP) for Modular Multiplier Parallel Architecture (Non-pipelined) (MM_PA_N) and (36.5 KLUTs, 2016 DSPs) for Modular Multiplier Parallel Architecture (Pipelined) (MM_PA_P).

The design in [27] is more effective than previous works [25-26], because it consumes less hardware resources and has lower latency. The amount of hardware area needed in this work [27], for the sum of residues reduction non-pipe-lined (SOR_1M_N) design, the sum of residues reduction with pipe-lined (SOR_1M_P) design, and the sum of residues reduction using two parallel pipe-lined (SOR_2M) design is equal to (8.17 KLUTs, 140 DSPs), (8.73 KLUTs, 140 DSPs) and (10.11 KLUTs, 280 DSPs), respectively. The hardware needed for the parallel architecture of the RNS Montgomery multiplier reported in [23] is equal to (9.21 KLUT, 248 DSP), showing its improvement over previous work [27].

As mentioned in literature, a lot of works are introduced about modular multipliers for high-speed performance, but due to the dissimilar implementation technology, a direct comparison is not always conceivable. In this paper, to have a straight comparison, SOR algorithm using the proposed moduli set is implemented on the Xilinx Virtex-7 FPGA similar the previous works. Finally, from the results presented in table 4, it can be concluded that the using proposed moduli set in the implementation of the modified SOR algorithm proposed in [27], it is more efficient in terms of latency and area compared to the previous works [23,27].

In comparison with the most advanced implementations on Virtex-7 FPGA, presented in [23], SOR_2M architecture using the proposed moduli set has achieved 32.5% and 13.7% faster than SOR_2M architecture proposed in [27] and [23], respectively. Further, SOR_1M_N using the proposed moduli set, has achieved 14.1% improvement compare to SOR_1M_N [27] in terms of area.

CONCLUSION

This paper presents an efficient implementation of modular reduction. A new balanced and well-formed eight-moduli set $\{2^{d+6}-1, 2^{d+4}-1, 2^{d}-1, 2^{d-2}-1, 2^{d-3}-1, 2^{d-4}-1, 2^{d-6}-1, 2^{d-8}-1\}$ is selected and employed in the implementation of the improved sum of residues algorithm. SOR algorithm using the proposed moduli set is described in VHDL language and synthesized by the ISE14.7 software on Xilinx virtex7 FPGA. The synthesis results illustrated that, compared to the latest work in literature, SOR_2M using the proposed moduli set has achieved 13.7% improvement in speed and SOR_1M_N with proposed moduli set, uses less hardware resources compared to the best work in literature.

Conflict of interest

The authors declare that there are no conflict of interests.

REFERENCES

1. Malik M, Dutta M, Granjal J. A survey of Key bootstrapping protocols based on Public Key Cryptography in the Internet of Things. IEEE Access 2019
2. Kavitha K, Alphonse PJA, Reddy YV. An improved authentication and security on efficient generalized group key agreement using hyper elliptic curve based public key cryptography for IOT health care system, J. Med. Syst. 2019 Jul;43.
3. Navi K, Molahosseini A, Esmaeildoust M. How to Teach Residue Number System to Computer Scientists and Engineers, IEEE. 2011.54: 156–163.
4. Rivest RL, Shamir A, Adleman LM. A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM. 1978.21 (2): 120–126.
5. Bajard JC, Imbert L. A full RNS implementation of RSA. IEEE Trans. on Compute. 2004 June; 53(6): 769-774.
6. ELGamal T. A public key cryptosystem and a signature scheme based on discrete logarithms," IEEE Trans. Inf. Theory. 1985 Jul; 31(4): 469–472.
7. Miller VS. Advances in Cryptology – CRYPTO '85 Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, Ch. Use of Elliptic Curves in Cryptography. 1986:417–426.
8. Koblitz N. Elliptic curve cryptosystems. Math. Compute. 1987. 48:203-209.
9. Esmaeildoust M, Schinianakis D, Javashi H, Stouraitis T, and Navi K. Efficient RNS implementation of elliptic curve point multiplication over GF (p). IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 2013 Aug; 21 (8):1545-1549.
10. Asif S, Hossain M, Kong Y,and Abdul W. A fully RNS based ECC processor", Integr. VLSI J. 2018. 61:138-149.
11. CChang CH, Molahosseini AS, Zarandi AAE, and Tay TF. Residue number systems: A new paradigm to datapath optimization for low-power and high-performance digital signal processing applications. IEEE Circuits Syst. Mag. 2015. 15 (4): 26–44.
12. Cardarilli GC, Nannarelli A, and Re M. RNS applications in digital signal processing in Embedded Systems Design with Special Arithmetic and Number Systems, Cham, Switzerland: Springer.2017: 181-215.
13. Veligosha AV, Linets GI, Kaplun DI, Klionskiy DM, and Bogaevskiy DV. Implementation of non-positional digital filters", Proceedings of the XIX IEEE International Conference on Soft Computing and Measurements (SCM).2016 May; 148-150.
14. Cardarilli GC, Nunzio LD, Fazzolari R, Nannarelli A, Petricca M, and Re M. Design space exploration based methodology for residue number system digital filters implementation. IEEE Trans. Emerg. Topics Comput. 2020. May.
15. Younes D, Steffan P. Efficient image processing application using residue number system. Proc. 20th Int. Conf. Mixed Design Integr. Circuits Syst. (MIXDES). 2013 Jun; 468-472.
16. Chervyakov N, Lyakhov P. RNS-Based Image Processing" in Embedded Systems Design with Special Arithmetic and Number Systems, Cham, Switzerland: Springer. 2017: 217-245.
17. Chu J, Benaissa M. Error detecting AES using polynomial residue number systems. Microprocessors Microsyst. 2013 March; 37(2): 228-234.
18. Veligosha AV, Kaplun DI, Klionskiy DM, Bogaevskiy DV, Gulvanskiy VV, and Kalmykov IA. Error Correction of Digital Signal Processing Devices using Non-Positional Modular Codes", Automatic Control and Computer Sciences.2017. 51 (3): 167-173.
19. Montgomery PL. Modular multiplication without trial division. Mathematics of Computation.1985. 44: 519–521.
20. Bajard JC, Didier LS, and Kornerup P. An RNS Montgomery modular multiplication algorithm," IEEE Trans. Comput. 1998 Jul; 47 (7): 766–776.

21. Schinianakis D, Stouraitis T. A RNS Montgomery Multiplication Architecture. In Proceedings of the IEEE International Symposium of Circuits and Systems (ISCAS), Rio de Janeiro, Brazil. 2011 May; 15-18.
22. Bajard JC, Eynard J, and Merkiche N. Montgomery reduction within the context of residue number system arithmetic. J. Cryptogr. Eng. 2018. 8: 189–200.
23. Ahsan J, Esmaeildoust M, Kaabi A, and Zarei V. Efficient FPGA of RNS Montgomery multiplication using balanced RNS bases, Integration. 2022 May; 84: 72-83.
24. Phillips BJ, Kong Y, and Lim Z. Highly parallel modular multiplication in the residue number system using sum of residues reduction. Appl. Algebra Eng. Commun. Comput. 2010. 21(3):249–255.
25. Asif S and Kong Y. Highly Parallel Modular Multiplier for Elliptic Curve Cryptography in Residue Number System. Circuits Syst. Signal Process. 2017. 36: 1027–1051.
26. Asif A. High-Speed Low-Power Modular Arithmetic for Elliptic Curve Cryptosystems Based on the Residue Number System. Ph.D. Thesis, Macquarie University, Sydney, Australia. 2016.
27. Mehrabi M. Improved sum of residues modular multiplication algorithm. Cryptography. 2019. 3 (2): 1-16.
28. Mehrabi MA, Doche C, and Jolfaei A. Elliptic curve cryptography point multiplication core for hardware security module. IEEE Trans. Comput. 2020. 69: 1707–1718.
29. Taylor FJ. Residue Arithmetic: A Tutorial with Examples", Computer. 1984 May; 50-62.
30. Guan G and Jones EV. Fast conversion between binary and residue numbers. Electron. Lett. 1988. 24 (19): 1195–1197.
31. Vergos HT, and Efstathiou C. On the design of efficient modular adders. J. Circuits, Syst., Comput. 2005. 41 (5): 965–972.
32. Soderstrand MA, Johnson TG, and Clark GA. Use of generalized quadratic RNS arithmetic in adaptive filters. In Proc. IEEE Asilomar Conf. Circuits, Syst., Comput. Pacific Grove, CA. 1985 Nov; 102–106.
33. Hosseinzadeh M, Molahosseini AS, and Navi K. A fully parallel reverse converter. Int. J. Elect., Comput. Syst. Eng.. 2007. 1 (3):183–187.
34. Molahosseini AS, Navi K, and Rafsanjani MK. Efficient forward and reverse converters for a new high-radix moduli set. In Proc. 3rd IEEE Int. Symp. Inf. Technol. 2008: 1–4.
35. Chakraborti N, Soundararajan J, and Reddy A. An implementation of mixed-radix conversion for residue number applications", IEEE Trans. Comput 1986 Aug; 35: 762-764.
36. Van Vu T. Efficient implementations of the Chinese reminder theorem for sign detection and residue decoding. IEEE Trans. Comput. 1985 Jul; C-34: 646-651.
37. Arhami B. Computer Arithmetic: Algorithms and Hardware Design. Oxford, U.K.: Oxford Univ. Press. 2000.
38. Cao B, Chang CH, and Srikanthan T. An efficient reverse converter for the 4-moduli set {2^n-1, 2^n, 2^n+1, 2^{2n}+1} based on the new chinese remainder theorem. IEEE Trans. Circuits Syst. I, Reg. Papers.2003 Oct; 50 (10): 1296–1303.
39. Wang y. Residue-to-binary converters based on new Chinese remainder theorems. IEEE Trans. Circuits Syst. II, Exp. Briefs. 2000 fEB; 47 (2):197–205.
40. Wang Y, Song X, Aboulhamid M, and Shen H. A new algorithm for RNS magnitude comparison based on new chinese remainder theorem II," in Proc. 9th Great Lakes Symp. VLSI. 1999: 362–365.
41. Omondi A, and Premkumar B. Residue Number Systems: Theory and Implementations, Imperial College Press. 2007.
42. Mohan PVA. Residue Number Systems: Theory and Applications, Cham, Switzerland: Birkhäuser. 2016.
43. Kawamura S, Koike M, Sano F, and Shimbo A. Cox-Rower Architecture for Fast Parallel Montgomery Multiplication", Advances in Cryptology Proc. EUROCRYPT. 2000 May; 523-538.
44. Bajard JC, Kaihara M, and Plantard T. Selected RNS bases for modular multiplication. 19th IEEE Symposium on Computer Arithmetic. IEEE. 2009: 25-32.
45. Abdallah M and Skavantzos A. On multimoduli residue number systems with moduli of forms r/sup a/, r/ sup b/-1, r/ sup c/+1, IEEE Trans. Circuits Syst. I Reg. Papers. 2005 Jul; 52 (7): 1253-1266.
46. Skavantzos A, Abdallah M, Stouraitis T, and Schinianakis D. Design of a balanced 8-modulus RNS", Proc. IEEE 16th Int. Conf. Electronics Circuits Syst. 2009: 61-64.
47. Barrett P. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In Proceedings of the Conference on the Theory and Application of Cryptographic Techniques, Linkoping, Sweden. 1986 May 20–22.