Circuits Syst Signal Process (2017) 36:1027–1051
DOI 10.1007/s00034-016-0336-1

Highly Parallel Modular Multiplier for Elliptic Curve Cryptography in Residue Number System

Shahzad Asif[1] · **Yinan Kong**[1]

Received: 6 July 2015 / Revised: 3 May 2016 / Accepted: 4 May 2016 / Published online: 19 May 2016

Abstract This article proposes a novel architecture to perform modular multiplication in the Residue Number System (RNS) by using sum of residues. The highly parallel architecture is implemented using VHDL and verified by extensive simulations in ModelSim SE. The pipelined and non-pipelined versions of the design are implemented on ASIC and FPGA platforms to allow a broad comparison. The proposed architecture requires only one iteration to complete modular multiplication and achieves 12–90 % less delay as compared to the existing RNS and binary modular multipliers. The complexity of the proposed design is also less than the existing state-of-the-art RNS-based modular multipliers. The high scalability and flexibility of the proposed architecture allows it to be used for a wide range of high-speed applications.

Keywords Residue Number System (RNS) · Modular multiplier · Montgomery multiplier · Cryptosystem · RSA · Elliptic curve cryptography (ECC) · High speed

1 Introduction

1.1 Background

One can argue that public-key cryptosystems become more secure as the advances in hardware speed up the computation of cryptographic algorithms. Take the RSA cryptosystem as an example. The effort of cracking RSA through factorisation of the product of two large primes approximately doubles for every 35 bits at a key size of 2^{10} bits [12]. However, adding 35 bits to the key only increases the work

✉ Shahzad Asif
shahzad.asif@mq.edu.au

[1] Department of Engineering, Macquarie University, Sydney, NSW 2109, Australia

involved in decryption by 10 %. Thus, speeding up the hardware by just 10 % makes the cryptosystem about twice as strong without any other extra resources [26]. Speed, therefore, is an important goal for public-key cryptosystems. Indeed, it is essential not just for cryptographic strength but also to clear the large number of transactions performed by central servers in electronic commerce systems.

This work aims to speed up public-key cryptosystems by accelerating their fundamental operation: the multiplication $X = A \times B$ followed by a reduction modulo M, $X \mod M = \langle X \rangle_M$, where A, B and the modulus M are all n-bit positive integers. This is the most frequent operation in elliptic curve cryptosystems (ECC) [15]. In RSA, it is the only operation required to implement the modular exponentiations which constitute encryption and decryption [36]. The Residue Number System (RNS) [41] offers advantages for long wordlength arithmetic of this kind by representing integers in independent short wordlength channels.

Indeed, implementing public-key cryptosystems using RNS is an interesting avenue of research [6,25]. The drawback to this approach is RNS modular reduction which is a computationally complex operation. Early publications [42] avoided it altogether by converting from RNS representation back to a positional system, performing modular reduction there, and converting the result back into RNS. Later, algorithms using look-up tables [23,39,40,43] were proposed to perform short wordlength modular reduction. Most of these avoided converting numbers from RNS to positional systems, but were limited to 32-bit inputs [9,19,46] by tables available. The work in [1] uses Chinese remainder theorem (CRT) to perform modular reduction within RNS channels; however, no implementation results are given for their proposed algorithm. Another alternative is the use of core function to perform RNS-based modular multiplication [27]. More recently, variations of Montgomery's reduction algorithm [31] have been developed which work entirely within a RNS [5,17,34].

Montgomery's reduction algorithm is only one of the alternatives available in positional number systems [28]. This raises a question: can any of the other reduction algorithms from positional number systems be applied to RNS? This paper provides an answer in the affirmative by presenting an RNS reduction architecture which uses sum of residues reduction with a fast implementation on FPGA.

Early attempts in positional number systems reduce $Z = A \times B$ modulo M by finding a sum of residues modulo M [16,44]. If $Z = \sum_i Z_i$ then we have $\sum_i \langle Z_i \rangle_M \equiv Z \mod M$. Although this does not produce a fully reduced result, it is possible to determine bounds for intermediate values such that the output from one modular multiplication can be used as the input to subsequent modular multiplications without overflow.

We use this sum of residues method for modular multiplication regarding large integers in the RNS, with the advantage that all of the residues $\langle Z_i \rangle_M$ can be evaluated in parallel. The proposed novel algorithm performs the modular reduction completely within the RNS channels without any conversion to/from binary number system ensuring high-speed operation.

The rest of the paper is arranged as follows: Section 1.2 highlights our contribution to the topic of modular multiplication in RNS. Section 2 briefly explains the representation and benefits of residue number system. Section 3 describes the Barrett algorithm used in the proposed design to perform modulus operation within each RNS channel.

Section 4 explains the development of the proposed algorithm to perform modular multiplication in RNS. Section 5 describes the implementation of the proposed algorithm and comparison with RNS-based modular multipliers. The work is concluded in Sect. 6.

1.2 Contribution

This paper makes the following contributions.

1. Confinement of all the computations of modular multiplication within the RNS channels without any long wordlength operations or conversion to a positional number system.
2. Proposal of a novel algorithm that can perform modular multiplication in a single iteration. The proposed architecture − based on this algorithm − is the first hardware implementation of a single iteration modular multiplication.
3. A high scalability of the proposed algorithm and architecture of the modular multiplier. The dynamic range of the modular multiplication can be easily decreased or increased by changing the RNS channel width or number of channels in the proposed algorithm. This allows easy scale-up of the architecture by adding additional RNS channels to the existing architecture. The paper provides the detailed analysis and criteria to compute the pre-computed values required to construct modular multipliers of different wordlengths.

2 The Residue Number System

A Residue Number System [42] is characterised by a set of N co-prime moduli $\{m_1, ..., m_N\}$ with $m_1 < m_2 < \cdots < m_N$. In the RNS a non-negative integer A is represented in N channels: $A = \{a_1, a_2, ..., a_N\}$, where a_i is the residue of A with respect to m_i, i.e. $a_i = \langle A \rangle_{m_i} = A \mod m_i$. The wordlength of m_i (in bits) − which defines the RNS channel width − is denoted by w. Within the RNS there is a unique representation of all integers in the range $0 \leq A < D$ where $D = m_1 m_2 ... m_N$. D is therefore known as the dynamic range of the RNS. Two other values, D_i and $\langle D_i^{-1} \rangle_{m_i}$ are commonly used in RNS computations and are worth defining here. $D_i = D/m_i$ and $\langle D_i^{-1} \rangle_{m_i}$ is its multiplicative inverse with respect to m_i such that $\langle D_i \times D_i^{-1} \rangle_{m_i} = 1$.

If A, B and C have RNS representations given by $A = \{a_1, a_2, \ldots, a_N\}$, $B = \{b_1, b_2, \ldots, b_N\}$ and $C = \{c_1, c_2, \ldots, c_N\}$, then denoting * to represent the operations +, − or ×, the RNS version of $C = A * B$ satisfies

$$C = \{\langle a_1 * b_1 \rangle_{m_1}, \langle a_2 * b_2 \rangle_{m_2}, \ldots, \langle a_N * b_N \rangle_{m_N}\}. \quad (1)$$

Thus, addition, subtraction and multiplication can be concurrently performed on the N residues within N parallel channels, and it is this high-speed parallel operation that makes the RNS attractive. There is, however, no such parallel form of the modular reduction regarding large modulus used in public-key cryptosystems. In order to implement RNS-based public-key cryptosystems, it is of utmost importance to devise an algorithm which can perform fast modular multiplication in RNS.

3 The Modular Reduction within RNS Channels

In Eq. (1), all the operations are accomplished by performing basic operations (addition, subtraction or multiplication) first and a reduction modulo for a channel modulus m_i second. Compared with the modular reduction, these basic operations of addition, subtraction and multiplication are trivial. This section explains how Barrett modular reduction algorithm [8] is used in our implementation to perform this modular reduction within RNS channels.

The relationship between division and modular reduction is made explicit in Eq. (2).

$$z = c \mod m = c - \left\lfloor \frac{c}{m} \right\rfloor \times m. \tag{2}$$

where c is $2w$ bits, m is the w-bit modulus and $\lfloor x \rfloor$ returns the largest integer smaller than or equal to x. To differ from the large modular multiplication over the whole RNS discussed in Sect. 4, lower case letters are used here to imply this is an operation running within a RNS channel m_i. Barrett algorithm, proposed for positional number system in [7] and [8], gives a fast computation of the division $y = \lfloor \frac{c}{m} \rfloor$ as

$$y = \left\lfloor \frac{c}{m} \right\rfloor = \left\lfloor \frac{\frac{c}{2^{w+v}} \frac{2^{w+u}}{m}}{2^{u-v}} \right\rfloor,$$

where u and v are two parameters. Furthermore, the quotient y can be estimated with an error of at most 1 from

$$\hat{y} = \left\lfloor \frac{\left\lfloor \frac{c}{2^{w+v}} \right\rfloor \left\lfloor \frac{2^{w+u}}{m} \right\rfloor}{2^{u-v}} \right\rfloor.$$

The value $K = \left\lfloor \frac{2^{w+u}}{m} \right\rfloor$ is a constant and can be pre-computed.

The algorithm used in our implementation is shown in Algorithm 1 where u and v are set to $w+3$ and -2, respectively, as suggested by [14] and [13]. The bounds on the quotient, input and output for these specific values of u and v are calculated to be $w+3$, $2w+2$ and $w+1$, respectively [13].

Algorithm 1 Barrett modular reduction algorithm

Require: m ▷ RNS channel modulus
Require: $u = w+3$, $v = -2$
Require: $K = \left\lfloor \frac{2^{2w+3}}{m} \right\rfloor$
Ensure: $z \equiv c \mod m$
$c_1 = \left\lfloor \frac{c}{2^{w-2}} \right\rfloor$
$c_2 = c_1 \times K$
$y = \left\lfloor \frac{c_2}{2^{w+5}} \right\rfloor$
$z = c - y \times m$

4 Modular Multiplication in Residue Number System

This section derives our main RNS modular multiplication (MM) algorithm using a sum of residues. More upper case variables reappear denoting large operands involved in modular multiplication over the whole RNS.

4.1 Moduli Selection

In our application, RNS is used to accelerate a 256-bit modular multiplication; therefore, the dynamic range D of the RNS should be no smaller than 512 bits so that the product of two 256-bit numbers does not overflow. One important rule to be considered is the uniform distribution of this 512-bit dynamic range into the N moduli. The smaller the RNS channel width w, the faster the computation within RNS and the more remarkable the advantage of RNS. Therefore, we want w to be as small as possible. In this paper, the N moduli are selected to be the same wordlength. This means that the dynamic range of the RNS system is evenly distributed into the N moduli.

A lot of work in the literature has been using the moduli in special forms, e.g. pseudo Mersenne numbers [10], or in the form of $2^w \pm 1$ [32]. However, this work only focuses on general moduli rather than special ones to demonstrate that fast implementation of modular multiplication does not have to rely on the special characteristics of the moduli, which is shown by our proposed algorithm.

4.2 Sum of Residues Reduction in the RNS

To define an RNS modular reduction algorithm, we start with the Chinese remainder theorem (CRT) [42]. Using the CRT, an integer X can be expressed as

$$X = \left\langle \sum_{i=1}^{N} D_i \langle D_i^{-1} x_i \rangle_{m_i} \right\rangle_D, \tag{3}$$

where D, D_i and $\langle D_i^{-1} \rangle_{m_i}$ are pre-computed constants. Defining $\gamma_i = \langle D_i^{-1} x_i \rangle_{m_i}$ in (3) yields,

$$\begin{aligned} X &= \left\langle \sum_{i=1}^{N} \gamma_i D_i \right\rangle_D \\ &= \sum_{i=1}^{N} \gamma_i D_i - \alpha D. \end{aligned} \tag{4}$$

Reducing this modulo the long wordlength modulus M yields

$$Z = \sum_{i=1}^{N} \gamma_i \langle D_i \rangle_M - \langle \alpha D \rangle_M$$

$$
\begin{aligned}
&= \sum_{i=1}^{N} Z_i - \langle \alpha D \rangle_M \\
&\equiv X \mod M
\end{aligned} \tag{5}
$$

where $Z_i = \gamma_i \langle D_i \rangle_M$. Thus, we have expressed $Z \equiv X \mod M$ as a sum of residues Z_i modulo M and a correction factor $\langle \alpha D \rangle_M$.

Note that $\gamma_i = \langle D_i^{-1} x_i \rangle_{m_i}$ can be found using a single RNS multiplication as $\langle D_i^{-1} \rangle_{m_i}$ is just a pre-computed constant. For the same reason, only one RNS multiplication is needed for $Z_i = \gamma_i \langle D_i \rangle_M$ as $\langle \langle D_i \rangle_M \rangle_{m_i}$ can be pre-computed.

In addition, to avoid negative residues resulted in the RNS channels from the subtraction in (5), the $-\langle \alpha D \rangle_M$ can be replaced by $+\langle -\alpha D \rangle_M$, which in the RNS is also a set of N pre-computed residues $\langle \langle -\alpha D \rangle_M \rangle_{m_i}$. This makes the last operation in (5) a simple RNS addition and (5) becomes

$$
Z = \sum_{i=1}^{N} \gamma_i \langle D_i \rangle_M + \langle -\alpha D \rangle_M, \tag{6}
$$

A further expansion to an expression of vectors of the pre-computed residues will make this equation clearer:

$$
\begin{aligned}
\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix} &= \sum_{i=1}^{N} \langle D_i^{-1} x_i \rangle_{m_i} \begin{pmatrix} \langle \langle D_i \rangle_M \rangle_{m_1} \\ \langle \langle D_i \rangle_M \rangle_{m_2} \\ \vdots \\ \langle \langle D_i \rangle_M \rangle_{m_N} \end{pmatrix} \\
&\quad + \alpha \begin{pmatrix} \langle \langle -D \rangle_M \rangle_{m_1} \\ \langle \langle -D \rangle_M \rangle_{m_2} \\ \vdots \\ \langle \langle -D \rangle_M \rangle_{m_N} \end{pmatrix}
\end{aligned} \tag{7}
$$

4.3 Approximation of α

Now α becomes the only value yet to be found. Here the method provided by Kawamura [24] is improved by decomposing its approximations, and more accuracy is achieved by permitting exact γ_i.

Dividing both sides of (4) by D yields

$$
\alpha + \frac{X}{D} = \frac{\sum_{i=1}^{N} \gamma_i D_i}{D} = \sum_{i=1}^{N} \frac{\gamma_i}{m_i}. \tag{8}
$$

Since $0 \le X/D < 1$, $\alpha \le \sum_{i=1}^{N} \frac{\gamma_i}{m_i} < \alpha + 1$ holds. Therefore,

$$\alpha = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{m_i} \right\rfloor. \tag{9}$$

In subsequent discussions, $\hat{\alpha}$ is used to approximate α. Firstly, an approximation of $\hat{\alpha} = \alpha$ or $\alpha - 1$ will be given. Secondly, some extra work will exactly assure $\hat{\alpha} = \alpha$ under certain prerequisites.

4.3.1 Deduction of $\hat{\alpha} = \alpha$ or $\alpha - 1$

The first approximation is introduced here: a denominator m_i in (9) is replaced by 2^w, where w is the RNS channel width and $2^{w-1} < m_i \leq 2^w - 1$. Then the estimate of (9) becomes

$$\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{2^w} \right\rfloor. \tag{10}$$

The error incurred by this denominator's approximation is denoted as

$$\epsilon_i = \frac{(2^w - m_i)}{2^w}.$$

Then,

$$2^w = \frac{m_i}{1 - \epsilon_i}.$$

According to the definition of RNS in Sect. 2, the RNS moduli are ordered such that $m_i < m_j$ for all $i < j$. Therefore, the largest error

$$\epsilon = \max(\epsilon_i) = \frac{(2^w - m_1)}{2^w}.$$

The accuracy of $\hat{\alpha}$ can be investigated:

$$\begin{aligned} &0 \leq \gamma_i \leq m_i - 1 \\ \Rightarrow\ &0 \leq \sum_{i=1}^{N} \frac{\gamma_i}{m_i} < N. \end{aligned} \tag{11}$$

Therefore,

$$\begin{aligned} \sum_{i=1}^{N} \frac{\gamma_i}{2^w} &= \sum_{i=1}^{N} \frac{\gamma_i (1 - \epsilon_i)}{m_i} \\ &= \sum_{i=1}^{N} \frac{\gamma_i}{m_i} - \epsilon \sum_{i=1}^{N} \frac{\gamma_i}{m_i} \end{aligned} \tag{12}$$

$$\Rightarrow \sum_{i=1}^{N} \frac{\gamma_i}{2^w} > \sum_{i=1}^{N} \frac{\gamma_i}{m_i} - N\epsilon. \tag{13}$$

The last inequality holds due to Eq. (11). If $0 \leq N\epsilon \leq 1$, then $\sum_{i=1}^{N} \frac{\gamma_i}{m_i} - N\epsilon > \sum_{i=1}^{N} \frac{\gamma_i}{m_i} - 1$. Thus, $\sum_{i=1}^{N} \frac{\gamma_i}{2^w} > \sum_{i=1}^{N} \frac{\gamma_i}{m_i} - 1$. In addition, obviously $\sum_{i=1}^{N} \frac{\gamma_i}{2^w} < \sum_{i=1}^{N} \frac{\gamma_i}{m_i}$. Therefore,

$$\sum_{i=1}^{N} \frac{\gamma_i}{m_i} - 1 < \sum_{i=1}^{N} \frac{\gamma_i}{2^w} < \sum_{i=1}^{N} \frac{\gamma_i}{m_i}. \tag{14}$$

Then,

$$\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{2^w} \right\rfloor = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{m_i} \right\rfloor = \alpha,$$

or,

$$\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{m_i} \right\rfloor - 1 = \alpha - 1.$$

when $0 \leq N\epsilon \leq 1$.

This raises the question: is it easy to satisfy the condition $0 \leq N\epsilon \leq 1$ in a RNS? The answer is: the larger the dynamic range of the RNS, the easier. This is contrary to most published techniques that are only applicable to RNS with a small dynamic range [9,19,39,43].

Given $0 \leq N\epsilon \leq 1$ and $\epsilon = \frac{(2^w - m_1)}{2^w}$,

$$\frac{N-1}{N} \leq \frac{m_1}{2^w} \leq 1,$$

which means there must be at least N co-prime numbers existing within the interval $I = [\frac{N-1}{N}2^w, 2^w]$ for the use of RNS moduli. Apart from this, it is also easy to satisfy the harsher condition $0 \leq N\epsilon \leq \frac{1}{2}$. This requires

$$\frac{2N-1}{2N} \leq \frac{m_1}{2^w} \leq 1,$$

which can be derived using the process above. Thus, the new interval for RNS moduli is given by Eq. (15) and will be used for further developments in the next subsection.

$$I = \left[\frac{2N-1}{2N}2^w, 2^w\right] \tag{15}$$

Table 1 lists the maximum N against different w from 4 to 24 within the interval $I = [\frac{2N-1}{2N}2^w, 2^w]$. It is evident that the number of available channels N increases dramatically along with the linear increase of the channel width w. This is because the span of interval I is $2^w - \frac{N-1}{N}2^w = \frac{2^w}{N}$. 2^w increases much faster than N, which gives a sharp increase of the span of I with more primes existing within it as the dynamic range D of the RNS increases.

The actual problem now is $\hat{\alpha}$ could be α or $\alpha - 1$. From Eq. (4), $\hat{X}$ could be X or $X + D$. Then two values of $X \mod M$ will result, and it is difficult to tell the correct one. Thus, $\hat{\alpha}$ needs to be the exact α.

Table 1 Maximum possible N against w in new RNS modular multiplication

w (bits)	Max. N	D (bits)	w (bits)	Max. N	D (bits)
4	2	8	5	3	15
6	3	18	7	5	35
8	6	48	9	9	81
10	12	120	11	17	187
12	21	252	13	29	377
14	40	560	15	49	735
16	69	1104	17	95	1615
18	128	2304	19	180	3420

D represents the number of bits for dynamic range

4.3.2 Ensuring $\hat{\alpha} = \alpha$

To make sure $\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{2^w} \right\rfloor$ in (10) is equal to α instead of $\alpha - 1$, a correction factor Δ can be added to the floor function. Equation (10) becomes

$$\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{2^w} + \Delta \right\rfloor. \tag{16}$$

Substituting Eq. (8) in Eqs. (13) and (14) yields

$$\alpha + \frac{X}{D} - N\epsilon < \sum_{i=1}^{N} \frac{\gamma_i}{2^w} < \alpha + \frac{X}{D}.$$

Adding Δ on both sides yields

$$\alpha + \frac{X}{D} - N\epsilon + \Delta < \sum_{i=1}^{N} \frac{\gamma_i}{2^w} + \Delta < \alpha + \frac{X}{D} + \Delta. \tag{17}$$

If $\Delta \geq N\epsilon$, then $\Delta - N\epsilon \geq 0$ and $\alpha + \frac{X}{D} - N\epsilon + \Delta \geq \alpha$. If $0 \leq X < (1 - \Delta)D$, then $\frac{X}{D} + \Delta < 1$ and $\alpha + \frac{X}{D} + \Delta < \alpha + 1$. Hence,

$$\alpha < \sum_{i=1}^{N} \frac{\gamma_i}{2^w} + \Delta < \alpha + 1. \tag{18}$$

Therefore,

$$\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{2^w} + \Delta \right\rfloor = \alpha$$

holds. The two prerequisites obtained from the deduction above are

$$\begin{cases} N\epsilon \leq \Delta < 1 \\ 0 \leq X < (1-\Delta)D. \end{cases} \tag{19}$$

It has already been shown in the previous section that the first condition $N\epsilon < \Delta < 1$ is easily satisfied as long as Δ is not too small. For example, Δ could be $\frac{1}{2}$. The second one is not that feasible at first sight as it requires X be less than half the dynamic range D in the case of $\Delta = \frac{1}{2}$. However, $\frac{1}{2}D$ is just one bit shorter than D, which is a number over two thousand bits. Therefore, this can be easily achieved by extending D by several bits to cover the upper bound of X. This is deduced in the following subsection. Hence, we have obtained an $\hat{\alpha} = \alpha$.

4.4 Bound Deduction

The RNS dynamic range to do a 256-bit multiplication should at least be 512 bits. However, RNS algorithms always require some redundant RNS channels. This subsection is dedicated to confirming how many channels are actually needed for the new RNS modular multiplication algorithm. Note that the result Z in Eq. (6)—the basis of the RNS modular multiplication algorithm—may be greater than the modulus M and would require subtraction of a multiple of M to be fully reduced. Instead, the dynamic range D of the RNS can be made large enough that the results of modular multiplications can be used as operands for subsequent modular multiplications without overflow.

Given that $\gamma_i < m_i < 2^w$, $\langle D_i \rangle_M < M$ and $\langle \alpha D \rangle_M \geq 0$,

$$Z = \sum_{i=1}^{N} \gamma_i \langle D_i \rangle_M - \langle \alpha D \rangle_M < N 2^w M. \tag{20}$$

Thus, take operands $A < N2^w M$ and $B < N2^w M$ such that $X = A \times B < N^2 2^{2w} M^2$.

According to Eq. (19), we must ensure that X does not overflow $(1-\Delta)D$. If it is assumed M can be represented in h channels so that $M < 2^{wh}$, then

$$X < N^2 2^{2wh+2w}.$$

$X < (1-\Delta)D$ is required for

$$D > 2^{wN-1},$$

which will be satisfied if

$$N^2 2^{2wh+2w} < (1-\Delta) 2^{wN-1}.$$

This is equivalent to

$$N > 2h + 2 + \frac{1 + 2\log_2 \frac{N}{1-\Delta}}{w}.$$

For example, for $w \geq 32$, $N < 128$ and $\Delta = \frac{1}{2}$, it will be sufficient to choose $N \geq 2h+7$. Note that this bound is conservative and fewer channels may be sufficient for a particular RNS. This is because the bound of Z can be directly computed as

$$Z = \sum_{i=1}^{N} \gamma_i \langle D_i \rangle_M - \langle \alpha D \rangle_M \leq \sum_{i=1}^{N} (m_i - 1)\langle D_i \rangle_M$$

using the pre-computed RNS constants, m_i and $\langle D_i \rangle_M$, instead of worst case bounds N and M as in (20).

4.5 The New RNS Modular Multiplication Algorithm

4.5.1 Another Approximation

The computation of α in Eq. (16) can be optimised by representing γ using its most significant q bits, where $q < w$. Hence, the approximated γ can be written as

$$\hat{\gamma}_i = 2^{w-q} \left\lfloor \frac{\gamma_i}{2^{w-q}} \right\rfloor. \tag{21}$$

The error incurred by this numerator's approximation is denoted as

$$\delta_i = \frac{\gamma_i - \hat{\gamma}_i}{m_i}.$$

Then

$$\hat{\gamma}_i = \gamma_i - \delta_i m_i.$$

The largest possible error will be

$$\delta = \frac{2^{w-q} - 1}{m_1}.$$

Note that this approximation, treated as a necessary part of the computation of α in [24], is actually not imperative. It has been shown the algorithm works fine without this approximation in previous discussions although it does simplify the computations in hardware.

Replacing the γ_i in Eq. (16) by $\hat{\gamma}_i$ yields

$$\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\hat{\gamma}_i}{2^w} + \Delta \right\rfloor. \tag{22}$$

Then, Eq. (12) becomes

$$\begin{aligned}\sum_{i=1}^{N}\frac{\hat{\gamma}_i}{2^w} &= \sum_{i=1}^{N}\frac{(\gamma_i-\delta_i m_i)(1-\epsilon_i)}{m_i}\\ &= \sum_{i=1}^{N}\frac{\gamma_i(1-\epsilon_i)}{m_i}-\sum_{i=1}^{N}(1-\epsilon_i)\delta_i\\ &\geq (1-\epsilon)\sum_{i=1}^{N}\frac{\gamma_i}{m_i}-N\delta\\ \sum_{i=1}^{N}\frac{\hat{\gamma}_i}{2^w} &> \sum_{i=1}^{N}\frac{\gamma_i}{m_i}-N(\epsilon+\delta). \end{aligned} \tag{23}$$

This is because

$$\begin{aligned} 0<1-\epsilon_i=\frac{m_i}{2^w}<1\\ \Rightarrow 0<\sum_{i=1}^{N}(1-\epsilon_i)<N, \end{aligned}$$

Note that the only difference between Eqs. (13) and (23) is that the ϵ in the former is replaced by the $\epsilon+\delta$ in the latter. Following a similar development to Sect. 4.3, Eq. (17) becomes

$$\alpha+\frac{X}{D}-N(\epsilon+\delta)+\Delta<\sum_{i=1}^{N}\frac{\hat{\gamma}_i}{2^w}+\Delta<\alpha+\frac{X}{D}+\Delta. \tag{24}$$

The two prerequisites in (19) are now

$$\begin{cases} N(\epsilon+\delta)\leq\Delta<1\\ 0\leq X<(1-\Delta)D \end{cases} \tag{25}$$

This will again guarantee

$$\hat{\alpha}=\left\lfloor\sum_{i=1}^{N}\frac{\hat{\gamma}_i}{2^w}+\Delta\right\rfloor=\alpha.$$

Substituting (21) in Eq. (22) yields

$$\hat{\alpha}=\left\lfloor\sum_{i=1}^{N}\frac{\left\lfloor\frac{\gamma_i}{2^{w-q}}\right\rfloor}{2^q}+\Delta\right\rfloor. \tag{26}$$

This is the final equation used in the new algorithm to estimate α.

4.5.2 RNS Modular Multiplication Algorithm and Design Example

The new sum of residues modular multiplication algorithm in RNS is shown in Algorithm 2. It computes $Z \equiv A \times B \mod M$ using Eq. (6). Note that from Eqs. (9) and (11), $\alpha < N$. Thus, $\langle -\alpha D \rangle_M$ can be pre-computed in RNS for $\alpha = 0 \ldots N-1$.

Algorithm 2 RNS modular multiplication algorithm

Require: $M, N, w, \Delta, q, \{m_1, \ldots, m_N\}$
Require: $(N2^w M)^2 < (1-\Delta)D$, $N(\frac{(2^w - m_1)}{2^w} + \frac{2^{w-q}-1}{m_1}) \leq \Delta < 1$.
Require: pre-computed table $\langle D_i^{-1} \rangle_{m_i}$ for $i = 1, \ldots, N$
Require: pre-computed table $\begin{pmatrix} \langle \langle D_i \rangle_M \rangle_{m_1} \\ \langle \langle D_i \rangle_M \rangle_{m_2} \\ \vdots \\ \langle \langle D_i \rangle_M \rangle_{m_N} \end{pmatrix}$ for $i = 1, \ldots, N$
Require: pre-computed table $\langle \langle -\alpha D \rangle_M \rangle_{m_i}$ for $\alpha = 1, \ldots, N-1$ and $i = 1, \ldots, N-1$
Require: $A < N2^w M$, $B < N2^w M$
Ensure: $Z \equiv A \times B \bmod M$
1: $\{x_1, x_2, \ldots, x_N\} = \{\langle a_1 \times b_1 \rangle_{m_1}, \langle a_2 \times b_2 \rangle_{m_2}, \ldots, \langle a_N \times b_N \rangle_{m_N}\}$
2: $\gamma_i = \langle x_i D_i^{-1} \rangle_{m_i}$ for $i = 1, \ldots, N$
3: $\alpha = \left\lfloor \sum_{i=1}^{N} \left\lfloor \frac{\gamma_i}{2^{w-q}} \right\rfloor / 2^q + \Delta \right\rfloor$
4: $Y_i = \{\langle \gamma_i \times \langle D_i \rangle_M \rangle_{m_1}, \langle \gamma_i \times \langle D_i \rangle_M \rangle_{m_2}, \ldots, \langle \gamma_i \times \langle D_i \rangle_M \rangle_{m_N}\}$ for $i = 1, \ldots, N$
5: $Sum_i = \sum_{j=0}^{N-1} Y_{j,i}$ for $i = 0, \ldots, N-1$ ▷ *where $Y_{j,i}$ means i^{th} channel of Y_j*
6: $Z_i = \langle Sum_i + \langle \langle -\alpha D \rangle_M \rangle_i \rangle_{m_i}$ for $i = 0, \ldots, N-1$

The flow chart of the proposed algorithm is shown in Fig. 1. The thick lines represent RNS values, whereas thin lines are used to represent binary values of small wordlength.

The proposed Algorithm 2 can be further explained with the help of an example with the following inputs and pre-computed values:

- $moduli = [16183, 16187, \ldots, 16383]$
- $M = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ *(NIST standard for Koblitz curve)*
- $N = 40$, $w = 14$, $\Delta = 0.75$, $q = 8$ (from Tables 1 and 3)
- $D_i^{-1} = [1027, 13322, ..., 698]$
- $\langle D_i \rangle_M = \{[3064, 11630, \ldots, 14819], [2396, 10967, \ldots, 6494], \ldots, [10399, 1229, \ldots, 678]\}$
- $A_i = [11169, 1811, \ldots, 15]$
- $B_i = [6273, 5504, \ldots, 9258]$

The complete moduli set used in this example is given in Table 4. The formulae for D_i^{-1} and $\langle D_i \rangle_M$ are given in Sect. 2. The steps below show the computation of the operation $(A \times B \bmod M)$ where each step corresponds to the steps of Algorithm 2.

1. $x_i = [\langle 11169 \times 6273 \rangle_{16183}, \langle 1811 \times 5504 \rangle_{16187}, ..., \langle 15 \times 9258 \rangle_{16383}] = [6930, 12739, ..., 7806]$
2. $\gamma_i = [\langle 6930 \times 1027 \rangle_{16183}, \langle 12739 \times 13322 \rangle_{16187}, ..., \langle 7806 \times 698 \rangle_{16383}] = [12773, 4450, ..., 9432]$
3. $\alpha = \lfloor \frac{(199+69+91+...+147)}{2^8} + 0.75 \rfloor = 25$

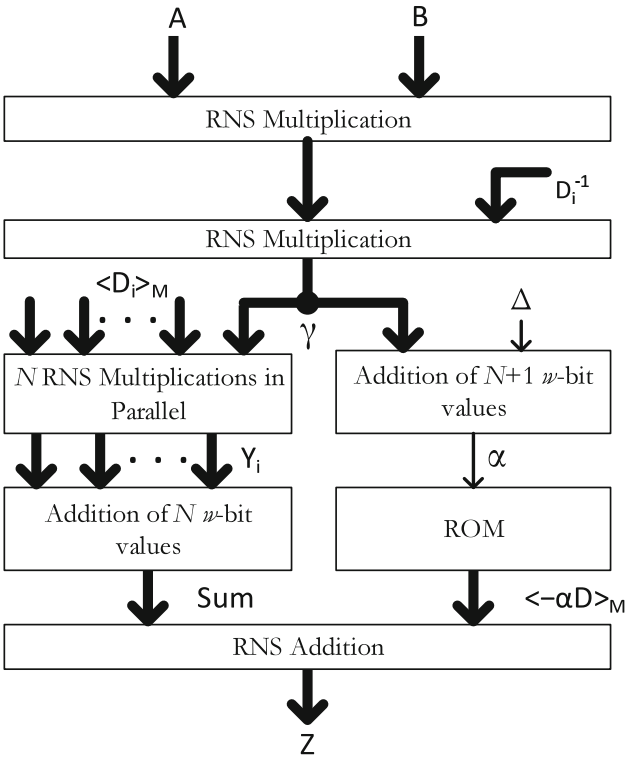

Fig. 1 RNS modular multiplication flow chart

4. $Y_i = \{[\langle 12773 \times 3064\rangle_{16183}, \langle 12773 \times 11630\rangle_{16187}, ..., \langle 12773 \times 14819\rangle_{16383}], [\langle 4450 \times 2396\rangle_{16183}, \langle 4450 \times 10967\rangle_{16187}, ..., \langle 4450 \times 6494\rangle_{16383}], ..., [\langle 9432 \times 10399\rangle_{16183}, \langle 9432 \times 1229\rangle_{16187}, ..., \langle 9432 \times 678\rangle_{16383}]\}$
   $Y_i = \{[5978, 1891, ..., 10288], [13786, 15532, ..., 15071], ..., [14388, 2036, ..., 5526]\}$
5. $Sum = [\langle 5978 + 13786 + ... + 14388\rangle, \langle 1891 + 15532 + ... + 2036\rangle, ..., \langle 10288 + 15071 + ... + 5526\rangle]$ $Sum = [446438, 403741, ..., 373271]$
6. $\langle -\alpha D\rangle_M = [13693, 3365, ..., 10031]$ $Z = [\langle 446438 + 13693\rangle_{16183}, \langle 403741 + 3365\rangle_{16187}, ..., \langle 373271 + 10031\rangle_{16383}]$ $Z = [7007, 2431, ..., 6493]$

The values of A, B and Z are given below in binary number system for better understanding.

$A = 1157920892373161954235709850086879078532699846656405640394575840079131296399 35$

$B = 7645500618709024670972844945214757001688688318560178355583814354233424294 2947072$

$Z = 2619465219992467549585156705197008821815356492891034885507922146802935738 2718268039$

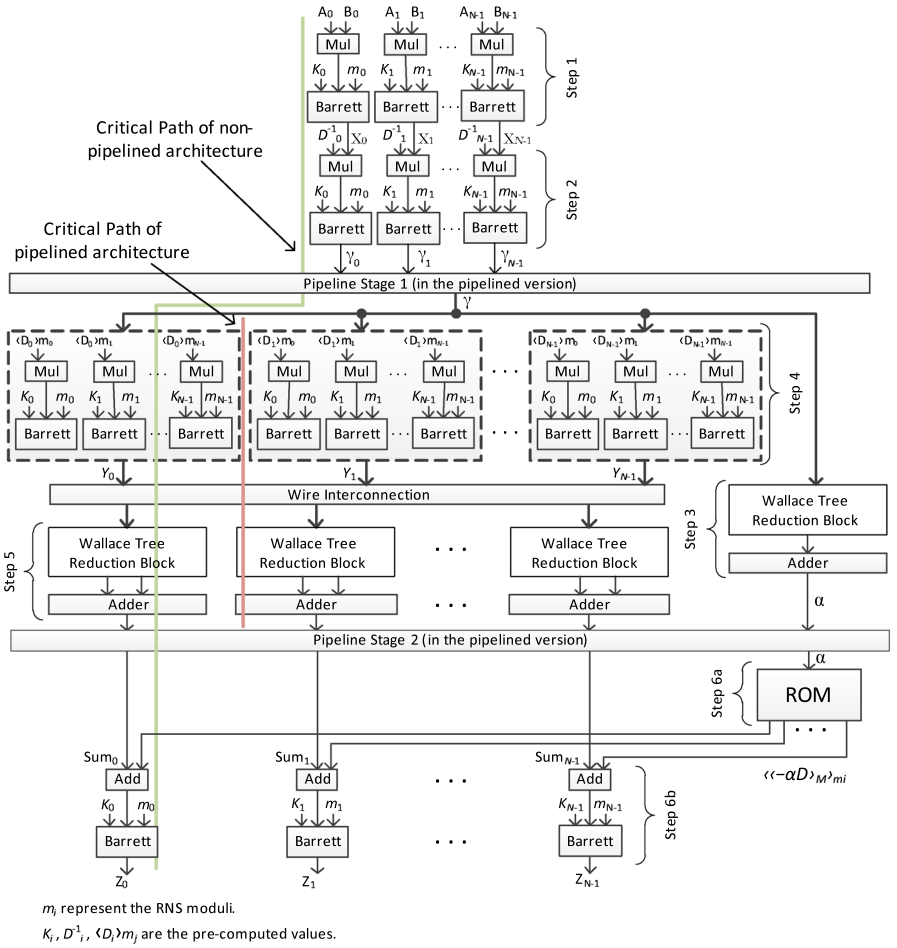

Fig. 2 Highly parallel architecture of RNS MM

5 Implementation and Synthesis Results

This section describes the implementation and synthesis results of the 256-bit modular multiplier (MM) in RNS using proposed algorithm.

5.1 Architecture of the RNS-Based MM

The architecture of the modular multiplier of Algorithm 2 is shown in Fig. 2. The pre-computed values required for the architecture are given in Table 2.

In Fig. 2, all of the computations are done in short wordlength (at most w bits, the RNS channel width) within the RNS. The architecture performs the following steps (the step numbers follow those in Algorithm 2).

Table 2 Pre-computed values for Algorithm 1 and Algorithm 2

i	0	1	…	N-1
K_i	K_0	K_1	…	K_{N-1}
$\langle D_i^{-1}\rangle_{m_i}$	$\langle D_0^{-1}\rangle_{m_0}$	$\langle D_0^{-1}\rangle_{m_1}$	…	$\langle D_{N-1}^{-1}\rangle_{m_{N-1}}$
	$\langle\langle D_0\rangle_M\rangle_{m_0}$	$\langle\langle D_0\rangle_M\rangle_{m_1}$	…	$\langle\langle D_0\rangle_M\rangle_{m_{N-1}}$
	$\langle\langle D_1\rangle_M\rangle_{m_0}$	$\langle\langle D_1\rangle_M\rangle_{m_1}$	…	$\langle\langle D_1\rangle_M\rangle_{m_{N-1}}$
$\big\langle\langle D_j\rangle_M\big\rangle_{m_i}$	.	.	.	.
	.	.	.	.
	.	.	.	.
	$\big\langle\langle D_{N-1}\rangle_M\big\rangle_{m_0}$	$\big\langle\langle D_{N-1}\rangle_M\big\rangle_{m_1}$	…	$\big\langle\langle D_{N-1}\rangle_M\big\rangle_{m_{N-1}}$
	0	0	…	0
	$\langle\langle -D\rangle_M\rangle_{m_0}$	$\langle\langle -D\rangle_M\rangle_{m_1}$	…	$\langle\langle -D\rangle_M\rangle_{m_{N-1}}$
	$\langle\langle -2D\rangle_M\rangle_{m_0}$	$\langle\langle -2D\rangle_M\rangle_{m_1}$	…	$\langle\langle -2D\rangle_M\rangle_{m_{N-1}}$
$\langle\langle -\alpha D\rangle_M\rangle_{m_i}$	.	.	.	.
	.	.	.	.
	.	.	.	.
	$\langle\langle -(N-1)D\rangle_M\rangle_{m_0}$	$\langle\langle -(N-1)D\rangle_M\rangle_{m_1}$	…	$\langle\langle -(N-1)D\rangle_M\rangle_{m_{N-1}}$

- In Step 1, the product $X = A \times B$ is computed within the RNS. This RNS multiplication involves three short wordlength multiplications and one subtraction.
- In Step 2, one RNS multiplication is performed to find the γ. This corresponds to three multiplications followed by one subtraction in the architecture of Fig. 2.
- Steps 3 and 4 are performed in parallel. RNS multiplications are used to compute the Y_is in Step 4, while the γ_is are used to generate α in Step 3. Note that the divisions in Step 3 are accomplished by simple right shifts.
- Step 5 and part of Step 6 are also performed simultaneously. The *sum* $\sum Y_i$ is computed in Step 5 using counter-based Wallace tree reduction, while $\langle -\alpha D\rangle_M$ is retrieved from memory in Step 6. It is to be noted that $\sum Y_i$ are simple additions without modulus operation; therefore, the channel width for *Sum* will be w+7 bits. This is not a problem because the result is still within the bounds for Barrett algorithm and can be reduced in the next step.
- Finally in the other part of Step 6, Z is produced by adding $\langle -\alpha D\rangle_M$ and the *Sum*.

Hence, this is a highly parallel structure with only 3 RNS multiplication, 1 Wallace reduction tree and 2 RNS additions in the critical path. It is therefore realised that all of the computations are done in the RNS channel width. In order to achieve a higher speed, the counter-based Wallace tree [4] is used which has less delay as compared to the conventional Wallace tree.

5.2 Design Specifications of 256-bit RNS MM

The architecture of Fig. 2 is used to implement a 256-bit modular multiplier in VHDL. The 256-bit modular multiplier consists of 40 RNS channels where each channel is of

Table 3 Design parameters of 256-bit RNS MM

Channels	Channel width	Dynamic range	q	Δ	*Modulus M*
40	15	560-bit	8	0.75	Koblitz curve

Table 4 RNS moduli set for w=14, N=40

16183	16187	16189	16193	16199	16217	16223	16229
16231	16241	16243	16249	16253	16259	16267	16271
16273	16277	16279	16301	16307	16309	16319	16321
16327	16333	16337	16339	16343	16349	16351	16361
16363	16367	16369	16373	16375	16379	16381	16383

15 bits. The size of each moduli is 14 bits, and one extra bit in channels is required due to the approximation used in the Barrett algorithm as mentioned in Sect. 3. The design parameters and the RNS moduli set of the architecture are given in Tables 3 and 4, respectively. The values of q and Δ are chosen according to the criteria discussed in Sect. 4.

5.3 Complexity Comparison

The complexity of the proposed architecture can be analysed from Algorithm 2 and Fig. 2 as follows:

1. Step 1 of Algorithm 2 performs one RNS multiplication. This is implemented by N w-bit multipliers followed by Barrett reductions as shown in Fig. 2. One Barrett reduction requires 2 w-bit multipliers and 1 w-bit subtractor. Hence, Step 1 requires $3N$ w-bit multipliers and N w-bit subtractors.
2. Step 2 also performs one RNS multiplication; therefore, the complexity of this step is same as Step 1.
3. Step 3 requires two division operations which can be easily implemented by simple right shifts because the divisor is a power of 2. These shifted values are then added together by using a Wallace tree reduction of N rows. The Wallace tree reduction block is followed by an adder to add the last two rows. Hence, this step requires $N + 1$ tree reduction block and one w-bit adder.
4. Step 4 performs N RNS multiplications in parallel. Each RNS multiplication requires $3N$ w-bit multipliers and N w-bit subtractors. Hence, the overall complexity of this step is $3N^2$ w-bit multipliers and N^2 w-bit subtractors.
5. Step 5 performs addition on the output of Step 4, i.e. N RNS values. This is done by adding channel 1 of Y_0 to Y_{N-1}, channel 2 of Y_0 to Y_{N-1}, and so on. Each addition is performed by using a Wallace tree reduction of N rows followed by one w-bit adder to add the last two rows. The complexity of this step is analysed as N Wallace tree reductions and N w-bit adders.

Table 5 Complexity analysis of the proposed architecture

Block type	No. of blocks	Critical path[a]
w-bit Mult	$3N^2 + 8N$	11
w-bit Add/Sub	$N^2 + 5N + 1$	6
Wallace tree (N rows)	$N + 1$	1

[a] Critical path is given for non-pipelined version

6. Step 6 of Algorithm 2 performs one RNS addition. This is implemented by N w-bit adders followed by Barrett reductions. Thus, this step requires $2N$ w-bit multipliers, N w-bit adders and N w-bit subtractors.

Based on this analysis, the complexity of the proposed architecture is summarised in Table 5. Note that the adders and subtractors are assumed to be of equal complexity for simplicity of analysis.

It can be seen from Table 5 that the critical path of the proposed architecture consists of only eleven and six w-bit multipliers and adders, respectively. It is important to note that the numbers of multipliers and adders in the critical path are independent of the number of channels. This property allows the scaling of the proposed architecture with very little decrease in the speed.

In order to compare the proposed architecture with other RNS-based modular multipliers, we evaluated the complexity in terms of w-bit modular multiplications and modular additions by following the approach given in [38]. The step-by-step analysis of the complexity is described as follows:

- Steps 1, 2: N modular multiplications are performed in parallel in these steps. Thus, a total of $2N$ modular multipliers are required. However, the critical path consists of only two modular multiplications.
- Step 3: Step 3 is responsible to add $N + 1$ values using the Wallace tree reduction where each value is $q + 1$ bits long. The q is usually a few bits less than w as discussed in detail in Sect. 4.5. Since the process is very similar to that of a multiplication (with the exception of partial products generation), therefore, for simplicity, we evaluate the complexity of Wallace tree in terms of multiplier. It is reasonable to say that a Wallace tree reduction of 9 (q+1 bits) columns and 41 $N + 1$ rows has similar complexity as of two 15-bit (w bits) multipliers. Hence, the complexity of Step 3 is estimated to be equivalent to a $\frac{2}{3}$ modular multiplication.
- Step 4: in this step N^2 modular multiplications are performed in parallel which means the delay of this step is same as the delay of one modular multiplier.
- Step 5: Step 5 is required to add N w-bit values using the Wallace tree reduction. Based on the above explanation we can estimate the complexity of Wallace tree reduction equivalent to three w-bit multipliers. Hence, the complexity of the Step 5 is estimated to be equivalent to one modular multiplication.
- Step 6: this step consists of one $w \times N \times N$ ROM and N modular additions. One modular addition is estimated to be equivalent to a $\frac{3}{4}$ modular multiplication.

Table 6 shows the comparison of the complexity of the proposed architecture with existing state-of-the-art RNS-based modular multipliers.

Table 6 Number of w-bit modular multiplications in the considered RNS MM algorithms

Design	Modular multiplications
[45]	$2N^2 + 5N$
[18] (with [24])[a]	$2N^2 + 6N$
[18] (with [5])[a]	$2N^2 + 5N$
[38]	$4N^2 + 20N + 7$
Proposed design	$N^2 + 3N + 2$

[a] 512-bit design

Table 7 RNS moduli set for w=17, N=62

130399	130409	130411	130423	130439	130447	130453	130457
130469	130477	130483	130489	130513	130517	130519	130523
130531	130547	130553	130561	130579	130589	130597	130607
130609	130619	130621	130631	130633	130639	130643	130649
130651	130657	130673	130681	130687	130693	130699	130717
130729	130733	130759	130763	130769	130771	130783	130787
130799	130807	130811	130813	130817	130829	130841	130843
130859	130861	130873	130901	130903	130909		

It can be seen from Table 6 that the proposed design requires about one-half and one-third the number of modular multiplications for the designs of [45] and [38], respectively.

The design in [18] implements a 512-bit modular multiplier; therefore, a detailed analysis is required in order to perform a fair comparison. To do this, we need to modify the proposed design such that it has the same dynamic range as of [18]. The dynamic range of [18] is 1055-bit with $N = 33$ and $w = 32$. Putting this value of N in Table 6 gives us $2N^2 + 5N$=2343 modular multiplications where each multiplication is of 32-bit.

In order to perform a fair comparison the proposed design needs to be scaled such that it has the same dynamic range as of [18]. This can be done by increasing the channel width (w+1) and/or number of channels (N). We propose N=62 and w=17 which will increase the dynamic range to 1054-bit with very little effect on the delay (note that the channel width $w + 1$ instead of w as explained in Section 3). The RNS moduli for the scaled-up design are given in Table 7.

Thus, the proposed scaled-up design requires $N^2 + 3N + 2$=4032 modular multiplications where each multiplication is of $w + 1 = 18$ bits. For simplicity, we can say that one 18-bit modular multiplier is equivalent to $\frac{18}{32} = 0.56$ 32-bit modular multiplier. Hence, the proposed design requires $4032 \times 0.56 = 2258$ 32-bit modular multiplications to compute one 512-bit modular multiplication. Based on this analysis, the complexity of the proposed design is 3.6% lower as compared to the [18].

5.4 Synthesis Results

The VHDL codes are developed for both pipelined and non-pipelined versions of the proposed 256-bit modular multiplier. The extensive simulations of the designs are performed using ModelSim SE. The designs are synthesised in Xilinx ISE 14.4 for Virtex-6 XC6VLX75T-3-FF784 FPGA with an optimisation goal of '*Speed*' and optimisation effort of '*Normal*'. The designs are also synthesised in Synopsys Design Compiler using 90-nm CMOS technology in order to do the comparison with the recent ASIC architectures. The designs are compiled using SAED90nm_typ library with typical process corner at a power supply of 1.2 V and 25 °*C* temperature. The compile effort of *medium* is used.

A large number of modular multipliers have been presented in the literature for high-speed operation, but a straightforward comparison is not always possible due to the difference in implementation technologies. The authors of [20] and [3] present the results only for the elliptic curve point multiplication and do not provide any results for the modular multiplication. Similarly, the work in [11] is focused on 128-bit pairing accelerators in RNS and analyse different types of pairing architectures. The unavailability of the modular multiplication results restricts these papers to join the comparison analysis. Yao et al. [47] perform a detailed analysis of the RNS parameter selection and its effects on the modular multiplication. The author proved the advantage of their proposed method by showing the clock cycles for one modular multiplication. However, the paper lacks the actual results and implementation details. Similarly, some recent modular multipliers [29,37,38,48] are designed for 1024-bit and cannot be used for the comparison.

In order to analyse the benefits of the proposed modular multiplier, it is compared with state-of-the-art modular multiplier implementations on ASIC and FPGA. Table 8 compares the synthesis results of the proposed and reference modular multipliers. The clock cycles represent the total cycles required to perform one modular multiplication, whereas the cycle time represents the minimum time period of the clock. The results in Table 8 are divided in two sections for ASIC- and FPGA-based implementations.

The results in Table 8 show that the proposed modular multiplier is faster than the existing high-speed modular multipliers. Note that the average time for one modular multiplication of the proposed architectures is same as their minimum cycle time. This is because the proposed architectures require only one iteration to compute one modular multiplication. The latency of the proposed pipelined architecture is $25.0 \times 3 = 75$ ns and $14.2 \times 3 = 42.6$ ns for ASIC and FPGA implementations, respectively, due to additional cycles required to fill the pipeline registers. The latency of non-pipelined architecture is same as its average time for one modular multiplication.

The comparison of the proposed architecture with other FPGA implementations is straightforward because they are implemented on the same FPGA device. The proposed MM outperforms the existing MM architecture implementations on FPGA in terms of speed and clock cycles. The closest FPGA implementation of existing MM is [2] which is 0.03 μs slower than the proposed architecture. The proposed architecture is 37, 93 and 94% faster than [2,22] and [21], respectively.

The comparison results of ASIC implementations include two designs among which [18] is implemented on an advanced 45-nm CMOS technology library, whereas the

Table 8 Performance comparison of the 256-bit modular multipliers

Design	Platform	Cycle time (ns)	Iterations	Avg. time (μs/MM)	Throughput (Mbps)
Neto [33]	90-nm CMOS	20.0	43	0.850	301.2
Gandino [18][a]	45-nm CMOS	1.12	80	0.090	2844.4
Proposed[b]	90-nm CMOS	25.0	1	0.025	10240.0
Proposed[c]	90-nm CMOS	72.7	1	0.073	3506.0
Javeed [22]	Virtex 6	10.42	128	1.300	196.9
Javeed [22]	Virtex 6	6.02	128	0.770	332.5
Javeed [21]	Virtex 6	11.55	129	1.487	172.0
Javeed [21]	Virtex 6	14.08	66	0.930	273.0
Alrimeih [2]	Virtex 6	10.00	8	0.080	3200.0
Marzouqi [30]	Virtex 5	6.24	213	1.330	192.5
Rahimzadeh [35]	Virtex 5	2.37	128	0.303	844.9
Proposed[b]	Virtex 6	14.20	1	0.014	18028.2
Proposed[c]	Virtex 6	47.25	1	0.05	5120.0

[a] 512-bit design
[b] 2-stage pipelined design
[c] Non-pipelined design

design in [33] uses the same technology. The proposed architecture outperforms both architectures in terms of throughput, clock cycles and average time for one modular multiplication. The cycle time of [33] is smaller than the proposed design; however, the proposed design can be modified to operate on much higher frequency by an increase in the pipeline stages. The main advantage of the proposed architecture is its less clock cycles which enables it to provide a higher throughput in spite of its slow clock frequency.

The design in [18] performs 512-bit modular multiplication, and the proposed design can be scaled up for 512-bit operation as explained in Sect. 5.3 which enables a fair analysis of the delay. The increase in the delay of the scaled design can be accurately computed by considering the scaling on the critical path as given in Table 5. It is evident from Table 5 that the number of channels has very little impact on the critical path of the architecture which means that the delay for a larger modular multiplier will approximately be same as of 256-bit modular multiplier. The impact of an increased N and w on the critical path is explained as follows:

- w-bit multiplier: the critical path consists of eleven w-bit multipliers. The value of w is increased from 15 to 18 in the scaled design which will result in a minor increase in the delay due to the interconnection and a slightly larger final adder in the multiplier. In order to have more precise results, we synthesised the 15-bit and 18-bit multipliers separately using the same device and synthesis parameters as for the original design.
- w-bit adder/subtractor: the critical path of the proposed modular multiplier consists of 6 w-bit adders/subtractors. The effect of a larger w on this part of the critical

Table 9 Synthesis results for the critical path delay of the proposed architecture

Design (ns)	Mult (ns)	Add/Sub (ns)	Tree reduction (ns)	Total delay (ns)
256-bit	4.14	0.81	6.29	56.69
512-bit	4.83	0.91	9.50	68.09

path will be even less than the above-mentioned component. The delay increase for this component is also analysed by the synthesis of 15-bit and 18-bit adders separately.

– Tree reduction of N rows: the critical path consists of one tree reduction to add N values of w bits. This component requires five reduction stages for $N = 40$ as well as for $N = 62$; therefore, the only increase in the delay will be due to the more interconnection wires.

Table 9 shows the synthesised delay of the different components of the critical path for the 256-bit and 512-bit versions of the proposed architecture. Equation (27) is used to calculate the total delay.

$$Delay_{\mathrm{MM}} = 11 \times Delay_{\mathrm{mult}} + 6 \times Delay_{\mathrm{add}} + Delay_{\mathrm{tree}} \tag{27}$$

The total delay of the 256-bit architecture calculated in Table 9 is slightly less than the actual result given in Table 8. The reason for this is the absence of interconnection delay which are not modelled in the calculated result; however, this can also be estimated by calculating the percentage of the interconnection delay in the 256-bit architecture and then adding same percentage in the calculated results of 512-bit architecture. The synthesised delay of 256-bit architecture is 72.7 ns which is 22 % more than the calculated delay of 56.69 ns. Based on this percentage the total delay for 512-bit architecture is estimated to be $68.09 + (0.22 \times 68.09) = 83.07$ ns. Hence, we can claim that the proposed architecture is 12% faster than [18]. It is also be noted that the design of [18] is implemented on a much advanced technology; therefore, our design is expected to show significant increase in performance if implemented on the same advanced technology of 45-nm CMOS. Furthermore, the use of pipeline stages can greatly improve the throughput of the proposed architecture. The advantage of the proposed architecture over [18] in terms of complexity is already proved in Sect. 5.3.

6 Conclusion and Future Work

A highly parallel and scalable architecture has been described to perform modular multiplication in the RNS using sum of residues. The algorithm performs the modular multiplication completely in the RNS channels without any need of conversion to the positional number system.

A 256-bit modular multiplier is implemented in a 40-channel RNS where each channel is of 15 bits. Pre-computed values are stored in the look-up tables to speed up the operations. The pipelined and non-pipelined versions of the architecture are

implemented in VHDL and synthesised in Xilinx ISE for Virtex-6 FPGA as well as on ASIC using 90-nm CMOS technology in Synopsys Design Compiler. The comparison shows that the delay and complexity of the proposed modular multiplier are 12 and 3.6 % lower, respectively, as compared to the state-of-the-art RNS modular multiplier.

As a future work, we plan to implement a 2048-bit modular multiplier based on the proposed algorithm. The proposed 256-bit and 2048-bit modular multipliers can be used to implement the RNS-based ECC and RSA cryptosystems, respectively.

References

1. O. Aichholzer, H. Hassler, A fast method for modulus reduction in residue number system. In: Proceedings Economical Parallel Processing, pp. 41–54. Vienna, Austria (1993)
2. H. Alrimeih, D. Rakhmatov, Pipelined modular multiplier supporting multiple standard prime fields. In: Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on, pp. 48–56 (2014). doi:10.1109/ASAP.2014.6868630
3. S. Antão, L. Sousa, A flexible architecture for modular arithmetic hardware accelerators based on RNS. J. Signal Process. Syst. **76**(3), 249–259 (2014). doi:10.1007/s11265-014-0879-y
4. S. Asif, Y. Kong, Design of an algorithmic Wallace multiplier using high speed counters. In: Computer Engineering Systems (ICCES), 2015 10th International Conference on, pp. 133–138 (2015). doi:10.1109/ICCES.2015.7393033
5. J.C. Bajard, L.S. Didier, P. Kornerup, Modular multiplication and base extensions in residue number systems. In: Proceedings 15th IEEE Symposium on Computer Arithmetic, vol. 2 pp. 59–65 (2001)
6. J.C., Bajard, L. Imbert, A full RNS implementation of RSA. IEEE Trans. Comput. 53(6), 769–774 (2004)
7. P. Barrett, Communications authentication and security using public key encryption—a design for implementation. Master's thesis, Oxford University (1984)
8. P. Barrett, Implementing the Rivest, Shamir and Adleman public-key encryption algorithm on a standard digital signal processor, *Advances in Cryptology - Crypto 86*, vol. 263, Lecture Notes in Computer Science (Springer, Berlin/Heidelberg, 1987), pp. 311–323
9. F. Barsi, M.C. Pinotti, Fast base extension and precise scaling in RNS for look-up table implementations. IEEE Trans. Signal Process. **43**(10), 2427–2430 (1995)
10. B. Cao, C.H. Chang, T. Srikanthan, A residue-to-binary converter for a new five-moduli set. IEEE Trans. Circuits Syst. I Regul. Pap. **54**(5), 1041–1049 (2007)
11. R.C.C. Cheung, S. Duquesne, J. Fan, N. Guillermin, I. Verbauwhede, G.X. Yao, FPGA implementation of pairings using residue number system and lazy reduction, *Proceedings of the 13th International Conference on Cryptographic Hardware and Embedded Systems*, CHES'11 (Springer-Verlag, Berlin, Heidelberg, 2011), pp. 421–441
12. R. Crandall, C. Pomerance, *Prime Numbers, A Computational Perspective* (Springer, Berlin, 2001)
13. J.F. Dhem, Modified version of the Barrett modular multiplication algorithm. Tech. rep, UCL Crypto Group, Louvain-la-Neuve (1994)
14. J.F. Dhem, Design of an efficient public-key cryptographic library for RISC based smart cards. Ph.D. thesis, Université Catholique de Louvain (1998)
15. C. Doche, L. Imbert, Extended double-base number system with applications to elliptic curve cryptography. In: Progress in Cryptology - INDOCRYPT 2006, vol. 4329. Springer (2006)
16. P.A. Findlay, B.A. Johnson, Modular exponentiation using recursive sums of residues, Advances in Cryptology—Crypto 89, vol. 435, Lecture Notes in Computer Science. (Springer, Berlin/Heidelberg, 1990), pp. 371–386
17. W.L. Freking, K.K. Parhi, Modular multiplication in the residue number system with application to massively-parallel public-key cryptography systems. In: Proc. 34th Asilomar Conference on Signals, Systems and Computers, vol. 2, pp. 1339–1343 (2000)
18. F. Gandino, F. Lamberti, G. Paravati, J. Bajard, P. Montuschi, An algorithmic and architectural study on Montgomery exponentiation in RNS. IEEE Trans. Comput. **61**(8), 1071–1083 (2012). doi:10.1109/TC.2012.84

19. A. Garcia, A. Lloris, A look-up scheme for scaling in the RNS. IEEE Trans. Comput. **48**(7), 748–751 (1999)
20. N. Guillermin, A high speed coprocessor for elliptic curve scalar multiplications over F_p, Proceedings of the 12th International Conference on Cryptographic Hardware and Embedded Systems, CHES'10 (Springer-Verlag, Berlin, Heidelberg, 2010), pp. 48–64
21. K. Javeed, X. Wang, Radix-4 and radix-8 Booth encoded interleaved modular multipliers over general Fp. In: Field Programmable Logic and Applications (FPL), 2014 24th International Conference on, pp. 1–6 (2014). doi:10.1109/FPL.2014.6927452
22. Javeed, K., Wang, X., Scott, M.: Serial and parallel interleaved modular multipliers on FPGA platform. In: Field Programmable Logic and Applications (FPL), 2015 25th International Conference on, pp. 1–4 (2015). doi:10.1109/FPL.2015.7293986
23. G.A. Jullien, Residue number scaling and other operations using ROM arrays. IEEE Trans. Comput. **27**(4), 325–336 (1978)
24. S. Kawamura, M. Koike, F. Sano, A. Shimbo, Cox-Rower architecture for fast parallel Montgomery multiplication. In: Advances in Cryptology—Eurocrypt 2000, Lecture Notes in Computer Science, vol. 1807, pp. 523–538. Springer (2000)
25. S.I. Kawamura, K. Hirano, A fast modular arithmetic algorithm using a residue table, *Advances in Cryptology - Eurocrypt 88*, vol. 330, Lecture Notes in Computer Science. (Springer, Berlin/Heidelberg, 1988), pp. 245–250
26. N. Koblitz, A Course in Number Theory and Cryptography. Graduate Texts in Mathematics 114. Springer-Verlag (1987)
27. Y. Kong, S. Asif, M. Khan, Modular multiplication using the core function in the residue number system. Appl. Algebra Eng. Commun. Comput. **27**(1), 1–16 (2016). doi:10.1007/s00200-015-0268-1
28. Y., Kong, B. Phillips, Residue number system scaling schemes. In: S.F. Al-Sarawi (ed.) Proc. SPIE, Smart Structures, Devices, and Systems II, vol. 5649, pp. 525–536 (2005)
29. S.R. Kuang, J.P. Wang, K.C. Chang, H.W. Hsu, Energy-efficient high-throughput montgomery modular multipliers for RSA cryptosystems. IEEE Trans. VLSI Syst. **21**(11), 1999–2009 (2013). doi:10.1109/TVLSI.2012.2227846
30. H. Marzouqi, M. Al-Qutayri, K. Salah, D. Schinianakis, T. Stouraitis, A high-speed FPGA implementation of an RSD-based ECC processor. IEEE Trans. VLSI Syst. **24**(1), 151–164 (2016). doi:10.1109/TVLSI.2015.2391274
31. P.L. Montgomery, Modular multiplication without trial division. Math. Comput. **44**(170), 519–521 (1985)
32. R. Muralidharan, C.H. Chang, Radix-8 Booth encoded modulo $2^n - 1$ multipliers with adaptive delay for high dynamic range residue number system. IEEE Trans. Circuits Syst. I Regul. Pap. **58**(5), 982–993 (2011)
33. J. Neto, A. Ferreira Tenca, W. Ruggiero, A parallel and uniform k -partition method for Montgomery multiplication. IEEE Trans. Comput. **63**(9), 2122–2133 (2014). doi:10.1109/TC.2013.89
34. K.C. Posch, R. Posch, Modulo reduction in residue number systems. IEEE Trans. Parallel Distrib. Syst. **6**(5), 449–454 (1995)
35. L. Rahimzadeh, M. Eshghi, S. Timarchi, Radix-4 implementation of redundant interleaved modular multiplication on FPGA. In: Electrical Engineering (ICEE), 2014 22nd Iranian Conference on, pp. 523–526 (2014). doi:10.1109/IranianCEE.2014.6999599
36. R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
37. D. Schinianakis, T. Stouraitis, Multifunction residue architectures for cryptography. IEEE Trans. Circuits Syst. I Regul. Pap. **61**(4), 1156–1169 (2014). doi:10.1109/TCSI.2013.2283674
38. D. Schinianakis, T. Stouraitis, An RNS Barrett modular multiplication architecture. In: Circuits and Systems (ISCAS), 2014 IEEE International Symposium on, pp. 2229–2232 (2014). doi:10.1109/ISCAS.2014.6865613
39. A. Shenoy, R. Kumaseran, A fast and accurate RNS scaling technique for high speed signal processing. IEEE Trans. Acoust. Speech Signal Process. **37**(6), 929–937 (1989)
40. A. Shenoy, R. Kumaseran, Fast base extension using a redundant modulus in RNS. IEEE Trans. Comput. **38**(2), 292–297 (1989)
41. M.A. Soderstrand, W. Jenkins, G. Jullien, Residue number system arithmetic: Modern applications. Digital Signal Processing (1986)

42. N.S. Szabo, R.H. Tanaka, *Residue Arithmetic and its Applications to Computer Technology* (McGraw Hill, New York, 1967)
43. F.J. Taylor, C.H. Huang, An autoscale residue multiplier. IEEE Trans. Comput. **31**(4), 321–325 (1982)
44. A. Tomlinson, Bit-serial modular multiplier. Electron. Lett. **25**(24), 1664 (1989)
45. Y. Tong-jie, D. Zi-bin, Y. Xiao-Hui, Z. Qian-jin, An improved RNS Montgomery modular multiplier. In: Computer Application and System Modeling (ICCASM), 2010 International Conference on, vol. 10, pp. V10 144–V10 147 (2010). doi:10.1109/ICCASM.2010.5622857
46. Z.D. Ulman, M. Czyzak, Highly parallel, fast scaling of numbers in nonredundant residue arithmetic. IEEE Trans. Signal Process. **46**, 487–496 (1998)
47. G. Yao, J. Fan, R. Cheung, I. Verbauwhede, Novel RNS parameter selection for fast modular multiplication. IEEE Trans. Comput. **63**(8), 2099–2105 (2014). doi:10.1109/TC.2013.92
48. G. Zervakis, N. Eftaxiopoulos, K. Tsoumanis, N. Axelos, K. Pekmestzi, A high radix Montgomery multiplier with concurrent error detection. In: Design Test Symposium (IDT), 2014 9th International, pp. 199–204 (2014). doi:10.1109/IDT.2014.7038613