The CSI Journal on
Computer Science and Engineering
Vol. 2, No. 2&4 (b), Summer & Winter 2004
Pages 31-39
Regular Paper

Improved RNS for RSA Hardware Implementation

Kooroush Manochehri Kalantari Saadat Pour Mozafari Babak Sadeghiyan

Computer Engineering Department, Amirkabir University of Technology
Tehran, Iran

Abstract

There are many methods for RNS implementation. The Bajard method is the fastest RNS implementation until now. One of the goals of this paper is to optimize this method to achieve higher performance for hardware implementation of RSA cryptosystem. Higher performance means increase in processing speed and less area. Proper hardware architectures for this method are proposed. For this purpose the number of multiplications is the criterion of the processing speed and the required memory for saving the constant values indicates the area required for this system. The number of multiplications is reduced by 400/(3k+11) percent (k is the number of modulus) in the final improved system and the number of constant values reduced by 50 percent.

Keywords: RNS, RSA, hardware implementation, cryptosystem, modular multiplication, modular exponentiation

1. Introduction

Residue number systems (RNS) have long been considered an alternative to weighted (Binary) representations for the implementation of digital signal processing and are used in cryptosystem and especially in RSA implementation [1,2,3,4] that uses modular exponentiation. RNS is capable to represent a very large integer as small independent integers that can be processed faster and easier. Also these small integers can be processed in parallel architectures and finally convert the results to a weighted large integer. This approach is attractive for the implementation of public key cryptosystems, as these typically require modular multiplication of very large integers.

In RNS, numbers are represented according to a base $\beta=(m_1,m_2,...,m_k)$ that all of the moduli are prime of each others, $\gcd(m_i,m_j)=1$, $i\neq j$, where k is the number of elements in this base. An integer a is represented by the sequence $(a_1,a_2,...,a_k)$ of positive integers, where $a_i=a \bmod m_i$, $i=1\ldots k$. The Chinese Remainder Theorem (CRT) ensures the uniqueness of this representation within the range $0\leq a <M$, where $M=\prod_{i=1}^{k} m_i$ [5]. The proof of this theorem can be used to convert back the numbers from its residue representation:

$$a=\sum_{i=1}^{k} a_i \times M_i \times \left|M_i^{-1}\right|_{m_i} \bmod M \qquad (1)$$

where $M_i=\frac{M}{m_i}$ and $\left|M_i^{-1}\right|_{m_i}$ is the inverse of M_i modulo m_i.

Modulus in the form of 2^n+1 and 2^n-1 are cost effective and usually are used in many RNS [6]. Residue calculation in this modulus is simple and faster than other moduli.

The most important advantage of RNS is that addition, subtraction and multiplication are very simple and can be implemented in constant time on a parallel architecture. If a

and b are given in their RNS form $(a_1,a_2,...,a_k)$ and $(b_1,b_2,...,b_k)$ the result of the mentioned operations are

$$a \pm b = \left(\left|a_1 \pm b_1\right|_{m_1},...,\left|a_k \pm b_k\right|_{m_k}\right)$$
$$a \times b = \left(\left|a_1 \times b_1\right|_{m_1},...,\left|a_k \times b_k\right|_{m_k}\right)$$

The disadvantages of RNS representation are two folds. First, one can not easily figure out a $(a_1,a_2,...,a_k)$ is greater than b $(b_1,b_2,...,b_k)$ or vise versa. Second, whether the overflow has occurred during the computation or not.. So division is difficult in this representation.

The above mentioned are not a drawback for cryptographic implementations including public key cryptography. In public key cryptography most of the algorithms perform the computations in a finite field or ring, which eliminates the overflow problem. Moreover, they do not require comparisons. Modular reduction (the computation of a mod N), multiplication (ab mod N) and exponentiation (a^b mod N) are the most important operations. They can be efficiently computed without division using Montgomery's algorithm.

The Bajard method for RNS implementation is the fastest method until now [2]. This is a general method and is independent of selecting modulus, but most of the RNS implementations and particularly RSA implementation by RNS used the modulus in the form of 2^n+1 and 2^n-1. The goal of this paper is to improve Bajard method for RSA hardware implementation by using the above-mentioned modulus. Also for implementing this improved method, proper architectures for each main processing unit are proposed. In designing of each unit the performance is mentioned.

In this process, speed increased while the used area is decreased. The number of multiplications is the criterion of processing speed and required memory for saving the constant values indicates the area required for this system.

This paper is organized as follows:

The Bajard method is presented briefly in second section. Section three presents the Bajard proposed method for implementation of RSA cryptosystem by RNS. In forth section, the improved Bajard method for hardware implementation of RSA is presented. In fifth section the hardware modules that are required for this implementation are proposed. The results are shown in section six. Finally the paper is concluded in section seven.

2. Bajard RNS Implementation

In Bajard method, Montgomery modular multiplication algorithm is used for modular multiplication (or modular exponentiation) and implemented by residue number system. This method is introduced in [2] and summarized in this section.

This method uses two RNS bases of size k $\beta = (m_1,...,m_k)$ and $\beta' = (m_{k+1},...,m_{2k})$. All of the m_i are prime of each other and the relationship $M' \geq M$ exist with the assumption that $M = \prod_{i=1}^{k} m_i$ and $M' = \prod_{i=k+1}^{2k} m_i$ should take place. Since this is based on Montgomery algorithm, the output are in the form of $a \times b \times M^{-1} \bmod(N)$ and like Montgomery method, multiplication by M gives favorite result, which is $a \times b \bmod N$. The Bajard RNS algorithm is as below.

The RNS Montgomery modular multiplication algorithm

Inputs: Two RNS bases $\beta=(m_1,...,m_k)$ and $\beta'=(m_{k+1},...,m_{2k})$, such that $M = \prod_{i=1}^{k} m_i < M' = \prod_{i=1}^{k} m_{k+i}$ and gcd(M,M')=1; a redundant moduli m_r, $\gcd(m_r,m_i)=1$, i=1…2k; a positive integer N represented in both RNS bases such that $0 < (k+2)^2 N < M < M'$ and gcd(N,M)=1, gcd(N,M')=1; two positive integers a,b represented in both RNS bases with ab<MN.

Outputs: A positive integer $\hat{r}$ represented in both RNS bases, such that $\hat{r} = a \times b \times M^{-1} \ (Mod\ N)$ and $\hat{r} < (k+2)N$.

MM(a,b,N):

1- $q \leftarrow (a \times b) \times (-N^{-1})$ in β

2- $[q \text{ in } \beta] \rightarrow [\hat{q} \text{ in } \beta']$ (First base extension)

3- $\hat{r} \leftarrow (a \times b + \hat{q} \times N) \times M^{-1}$ in β' and m_r

4- $[\hat{r} \text{ in } \beta] \leftarrow [\hat{r} \text{ in } \beta']$ *(Second base extension)*

Instructions in steps 1 and 3 can be performed in parallel by full RNS operations. As a consequence the complexity of the algorithm clearly relies on the two base extensions on steps 2 and 4.

If the number of multiplications is the criterion for speed, two modular multiplications in line 1, and three multiplications in line 3 are needed. Also if the number of memory for saving constant values is a criterion for area, one memory in line 1 for saving the values of $-N^{-1}$ in each modulus and two memories in line 3 for saving the values of N and M^{-1} in each modulus, are needed.

RSA cryptosystem requires modular exponentiation, so one can use this algorithm and reuse its output each time as its input until achieve the desired exponent.

The instruction in line 2 (first base extension) consists of converting q obtained in its RNS form $(q_1,\ldots,q_k)$ in the base β to its RNS representation in base β'. For this purpose the following relations are used. In these relationship q_i represents q mod m_i.

$$\sigma_i = q_i \times \left|M_i^{-1}\right|_{m_i} \bmod m_i \ , i = 1,...,k \qquad (2)$$

$$\hat{q}_j = \left|\sum_{i=1}^{k} \left|M_i\right|_{m_j} \times \sigma_i\right|_{m_j} \ , \ j = k+1,...,2k \qquad (3)$$

So σ_i must be first computed and then used in next relation. In this step a modular multiplication is needed for calculating σ_i and k modular multiplications for $\hat{q}_j$. Furthermore required one memory for saving the value of M_i^{-1} in each modulo m_i and one for saving M_i in each modulo m_j.

In line 4 (second base extension) first the value of ξ_j must be computed from the following relation:

$$\xi_j = \hat{r}_j \times \left|M_j'^{-1}\right|_{m_j} \text{Mod}\, m_j, \quad j = k+1,...,2k \tag{4}$$

and then the desired resultant value can be computed from:

$$\left|\hat{r}\right|_{m_i} = \left|\sum_{j=1}^{k} \left|M_j'\right|_{m_i} \times \xi_j - \left|\alpha \times M'\right|_{m_i}\right|_{m_i}, \; i = 1,...,k \tag{5}$$

where α is computed from:

$$\alpha = \left|\left|M'^{-1}\right|_{m_r} \times \left(\sum_{j=1}^{k} \left|M_j'\right|_{m_r} \times \xi_j - \left|\hat{r}\right|_{m_r}\right)\right|_{m_r} \tag{6}$$

For simplicity and ease of the calculation in the last equation, m_r usually assumed as power of 2 such that $m_r \geq k$, k represents the number of moduli in each base set.

3. Bajard RSA Implementation

Implementation of RSA with Montgomery modular multiplication algorithm is as the following. The final result of this algorithm is $c=a^e \bmod N$ [7].

RSA (a,e,n)
Step 1. $\bar{a}$ *:= a×R mod N*
Step 2. $\bar{c}$ *:= 1×R mod N*
Step 3. for i = n-1 downto 0 do
Step 4. $\bar{c}$ *:= MonPro (*$\bar{c}$*,*$\bar{c}$*)*
Step 5. if e_i *= 1 then* $\bar{c}$ *:= MonPro (*$\bar{a}$*,*$\bar{c}$*)*
Step 6. c := MonPro ($\bar{c}$*,1)*
Step 7. return c

Monpro represents the Montgomery modular multiplication algorithm and R represents the auxiliary moduli. The number of bits in N (or e) is shown in this algorithm by n.
The rules for algorithm is that the input is converted to Montgomery representation then according to number of bits of exponentiation the internal loop is repeated in order to calculate the desired modular exponentiation and finally the result of this loop is converted back to binary representation by the last Montgomery modular multiplication.
Note that in this algorithm 'a' represent the plaintext that must be encrypted. As mentioned before RNS final result is a single modular multiplication where in RSA a modular exponentiation is needed.
Since the Bajard RNS method is based on Montgomery algorithm, the previous RSA algorithm can be used except for Monpro, the RNS Montgomery modular multiplication algorithm (MM(a,b,N)) must be substituted and redundant residue (R) should be substituted by M (multiplications of modulus).
By this substitution the final result is less than (k+2)N, since it must be less than N so one must correct this difference. The solution that is proposed by Bajard [2] is that since this error is obtained by the calculation of the first base extension approximately, for the last call of multiplication algorithm (that is used for convert back to binary representation) this base extension should be calculated exactly. The Mixed Radix System (MRS) can be used for this exact base extension. The relations of MRS are as below:

$$\left|q\right|_{m_j} = \left|t_1 + t_2 m_1 + t_3 m_1 m_2 + ... + t_k m_1 ... m_{k-1}\right|_{m_j} \tag{7}$$

In other word in this system the weight of each number t_i is 1, m_1, m_1m_2, … , $m_1m_2m_3$ … m_k. The value of t_i can be computed as below:

$t_1 = q \bmod m_1 = q_1$
$t_2 = (q_2 - t_1)c_{12} \bmod m_2$
.
.
.
$t_k = (...((q_k - t_1)c_{1k} - t_2)c_{2k} - ... - t_{k-1})c_{(k-1)k} \bmod m_k$

c_{ij} represents $m_i^{-1} \bmod m_j$ and q_i is the value from step 1 of RNS algorithm which is equal to $q_i = q \bmod m_i$.

4. Improved Bajard Algorithm for Hardware Implementation

The Bajard algorithm for RNS implementation uses less clock cycles and has more speed for calculation than other methods. In this section improved Bajard algorithm is used and optimized for RSA hardware implementation to reduce number of clock cycles and the area.
The values that are calculated during internal calculations and are not necessary for RSA final results, are combined with other calculations to reduce the calculation time, as well as the area. For further clarity the optimizations for the main values are presented as follows.

4.1 Calculation of s_i in the first base extension

As shown in the first base extension of the algorithm, it gets the value of $q=(a\times b)\times(-N^{-1})$ in the RNS representation in base β and calculates the value of $\hat{q}$. So in RNS algorithm the value of q is not needed and can be combined with the first base extension to reduce calculation cycles. The new relationship for σ_i is as follow:

$$\begin{aligned} \sigma_i &= q_i \times \left|M_i^{-1}\right|_{m_i} \bmod m_i \\ &= (a \times b \times \left|-N^{-1}\right|_{m_i}) \times \left|M_i^{-1}\right|_{m_i} \bmod m_i \\ &= (a \times b) \times \left|-N^{-1} \times M_i^{-1}\right|_{m_i} \end{aligned} \tag{8}$$

In order to calculate a×b, then multiply by M_i^{-1}, multiply the final result by $-N^{-1}$ and assuming each multiplication requires one cycle, it demands at least 3 cycles. While by combined relation these cycles reduce to 2 cycles. As a result the speed is increased and area decreased, because in the new relation the value of $\left|-N^{-1} \times M_i^{-1}\right|_{m_i}$ must be saved instead of saving two values $\left|M_i^{-1}\right|_{m_i}$ and $\left|-N^{-1}\right|_{m_i}$.

4.2 Calculation of x_j

The values of ξ_j play an important role in the calculation of the final results. The equations that are needed to calculate ξ_j are listed below.

$$\xi_j = \hat{r}_j \times \left|M_j'^{-1}\right|_{m_j} \bmod m_j$$

$$\hat{r}_j = (a \times b + \hat{q}_j \times N) \times M^{-1} \bmod m_j$$

$$\hat{q}_j = \left(\sum_{i=1}^{k} \left|M_i\right|_{m_j} \times \sigma_i\right) \bmod m_j$$

If one starts with the first equation then substitute the values, it implies that:

$$\xi_j = ((a\times b+\hat{q}_j\times N)\times M^{-1})\times\left|M_j'^{-1}\right|_{m_j} \bmod m_j$$

$$=(a\times b\times\left|M^{-1}\times M_j'^{-1}\right|_{m_j})+$$

$$(\hat{q}_j\times\left|N\times M^{-1}\times M_j'^{-1}\right|_{m_j})\bmod m_j$$

$$=(a\times b\times\left|M^{-1}\times M_j'^{-1}\right|_{m_j})+$$

$$(\left|\sum_{i=1}^{k}\left|M_i\right|_{m_j}\times\sigma_i\right|_{m_j}\times\left|N\times M^{-1}\times M_j'^{-1}\right|_{m_j})\bmod m_j$$

$$=\underbrace{(a\times b\times\left|M^{-1}\times M_j'^{-1}\right|_{m_j})}_{\sigma_j}+$$

$$\underbrace{\sum_{i=1}^{k}(\sigma_i\times\left|M_i\times N\times M^{-1}\times M_j'^{-1}\right|_{m_j})\bmod m_j}_{\delta_j} \quad (9)$$

In this relationship the first multiplication is named σ_j and the second one is named δ_j. The calculation of σ_j with the calculation of σ_i can be computed parallel as a result the computation time is reduced. Also the area is reduced due to combining the fixed values, as a result instead of saving four values $\left|M_i\right|_{m_j}$, $\left|N\right|_{m_j}$, $\left|M^{-1}\right|_{m_j}$ and $\left|M_j'^{-1}\right|_{m_j}$ just save two values $\left|M^{-1}\times M_j'^{-1}\right|_{m_j}$ and $\left|M_i\times N\times M^{-1}\times M_j'^{-1}\right|_{m_j}$. Furthermore instead of k+4 multiplication cycles (k multiplication cycles for calculation of sigma for $\hat{q}_j$ and 4 multiplication cycles for other calculations) only k+2 cycles (k multiplication cycles for δ_j and two cycles for σ_j) are required.

4.3 Calculation of $\left|\hat{r}\right|_{m_r}$

Since the value of $\left|\hat{r}\right|_{m_r}$ is required for calculation of α, its value must be computed while computing other values. The formula for this computation is as below:

$$\left|\hat{r}\right|_{m_r}=(a\times b+\hat{q}\times N)\times M^{-1}\bmod m_r \quad (10)$$

$\hat{q}$ can be computed by the relation $\hat{q}_r=\sum_{i=1}^{k}\left|M_i\right|_{m_r}\times\sigma_i \bmod m_r$. So with substitution in the $\left|\hat{r}\right|_{m_r}$ relation, the new formula is as follows:

$$\left|\hat{r}\right|_{m_r}=\underbrace{(a\times b\times M^{-1})}_{\sigma_r}+\underbrace{\sum_{i=1}^{k}\sigma_i\times\left|M_i\times N\times M^{-1}\right|_{m_r}\bmod m_r}_{\delta_r} \quad (11)$$

In this new equation the first term is named as σ_r and second one is named δ_r. The calculation of σ_r can be done while calculating σ_i. Due to combining the relations, the number of multiplications is reduced from k+3 (k multiplications for calculation of $\hat{q}_r$ and 3 for $\left|\hat{r}\right|_{m_r}$ calculation) to k+1 (k multiplication cycles for δ_r and one for σ_r calculation), furthermore the number of memory for saving the constant value is reduced from 3 to 2.

After computing $\left|\hat{r}\right|_{m_r}$ the value of α can be computed as follow:

$$\alpha=\left|M'^{-1}\right|_{m_r}\times\left(\sum_{j=1}^{k}\left|M_j'\right|_{m_r}\times\xi_j-\left|\hat{r}\right|_{m_r}\right)\bmod m_r$$

$$=\underbrace{\sum_{j=1}^{k}\xi_j\times\left|M'^{-1}\times M_j'\right|_{m_r}}_{\alpha_1}-\left|\hat{r}\right|_{m_r}\times\left|M'^{-1}\right|_{m_r}\bmod m_r \quad (12)$$

In this equation the first term is named as α_1 for individual calculation.

4.4 Calculation of $\hat{r}$ in the final result

After the calculations of previous values now the final result or the output of algorithm ($\hat{r}$) is obtained. Due to removing some intermediate values for optimization, the values of $\left|\hat{r}\right|_{m_j}$ must be computed, as well as computation of $\left|\hat{r}\right|_{m_i}$. As a result the following relations can be used:

$$\left|\hat{r}\right|_{m_i}=\underbrace{\sum_{j=1}^{k}\left|M_j'\right|_{m_i}\times\xi_j}-\left|\alpha\times M'\right|_{m_i}\bmod m_i \quad (13)$$

$$\left|\hat{r}\right|_{m_j}=\underbrace{\xi_j}_{\rho}\times\left|M_j'\right|\bmod m_j \quad (14)$$

The first term in $\left|\hat{r}\right|_{m_i}$ is named as ρ. The relation of $\left|\hat{r}\right|_{m_j}$ is obtained from the fact that the value of ξ_j is equal to $\hat{r}_j\times\left|M_j'^{-1}\right|_{m_j}\bmod m_j$ and as a result $\left|\hat{r}\right|_{m_j}$ is produced by multiplying it with the value of $\left|M'_j\right|_{m_j}$. With this method one redundant multiplication is added but due to reducing the multiplication cycles that is obtained by other optimizations, this overhead can be omitted and the overall reduction in multiplication cycles is achieved.

If the numbers of clock cycles are mentioned, all calculations those are required for RNS method can be categorized as follows:

$$1)\ \sigma_i = (a\times b)\times\left|-N^{-1}\times M_i^{-1}\right|_{m_i} \bmod m_i \ ,i=1,...,k$$

$$\sigma_j = a\times b\times\left|M^{-1}\times M_j'^{-1}\right|_{m_j} \bmod m_j \ , j=k+1,...,2k$$

$$\sigma_r = a\times b\times M^{-1} \bmod m_r$$

$$2)\ \xi_j = \sigma_j + \sum_{i=1}^{k}(\sigma_i\times\left|M_i\times N\times M^{-1}\times M_j'^{-1}\right|_{m_j}) \bmod m_j$$

$$\left|\hat{r}\right|_{m_r} = \sigma_r + \sum_{i=1}^{k}\sigma_i\times\left|M_i\times N\times M^{-1}\right|_{m_r} \bmod m_r$$

$$3)\ \rho = \sum_{j=1}^{k}\left|M_j'\right|_{m_i}\times\xi_j \bmod m_i$$

$$\alpha_1 = \sum_{j=1}^{k}\xi_j\times\left|M'^{-1}\times M_j'\right|_{m_r} \bmod m_r$$

$$4)\ \alpha = \alpha_1 - \left|\hat{r}\right|_{m_r}\times\left|M'^{-1}\right|_{m_r} \bmod m_r$$

$$5)\ \left|\hat{r}\right|_{m_i} = \rho - \alpha\times M' \bmod m_i$$

$$\left|\hat{r}\right|_{m_j} = \xi_j\times\left|M'_j\right|_{m_j} \bmod m_j$$

In this categorization the relations in each category can be computed in parallel because they are independent of each other. Due to equal number of multiplication and summation in each category, the relations in one category are finished in the same cycles and their results can be used in the next category.

It's obvious that in the first category, 3 multiplications, second category, k multiplications and one addition, third category, k multiplications, forth category one multiplication and one subtraction, fifth category, k multiplications and for $\left|\hat{r}\right|_{m_i}$ a redundant subtraction are needed.

5. Implementation of Hardware Modules

In this section the main modules for hardware implementation of RSA by RNS method is presented. Also the modulus selection in both bases β and β', to improve its performance, are presented.

5.1 Selection of modulus

As mentioned before the modulus in the form of 2^n-1 and 2^n+1 are more useful and cost effective for RNS implementations. For selecting these modulus from the set $\{2^{n_1}-1,2^{n_1}+1,2^{n_2}-1,2^{n_2}+1,...,2^{n_L}-1,2^{n_L}+1\}$ one must notice that all of these modulus are prime of each other. For this purpose due to the proof that is presented in [8] all the n_i i=1,…,L must be prime of each other. Also for having a similar modulus length, these parameters should be as close as possible.

For ease of implementation and speed improvement, modulus in the form of 2^n-1 is selected as first base set (M) and modulus in the form of 2^n+1 is selected as second base set (M'). This assumption satisfy the relation M<M'. Moreover by this selection the decision-making for hardware implementation of each step in the RNS algorithm can be done easily.

5.2 the 2^n+1 modular multiplier

the 2^n+1 modular multiplication has the main role in RNS implementation but operations in this moduli are not as easy as operations in 2^n-1 [9].

The diminished-1 representation of numbers was proposed by Leibowitz [10], as a convenient and efficient form for modulo 2^n+1 operations on binary numbers. If d(a) is the diminished-1 representation of 'a' then:

$$d(a) = a-1 \ (\text{mod } 2^n+1) \qquad (15)$$

The advantage of this representation is that zero is uniquely identified by MSB=1, for which all arithmetic operations are inhibited.

One has the following relationships for arithmetic operations within the representation:

$$d(a+b)=d(a)\oplus d(b) \qquad (16)$$
$$=d(a)+d(b)+1 \ (\text{Mod } 2^n+1)$$

$$d(a-b) = d(a)\oplus[-d(b)] \qquad (17)$$
$$= d(a)+\overline{d(b)}+1 (\text{mod}\, 2^n+1)$$

$$d\left(\sum_{i=1}^{n}a_i\right) = \sum_{i=1}^{n}{}_{\oplus}\, d(a_i) \qquad (18)$$
$$= d(a_1)\oplus...\oplus d(a_n)$$
$$= d(a_1)+...+d(a_n)+n-1 (\text{mod}\, 2^n+1)$$

In these relationships, $\oplus$ represents addition and [-x] represents negative of number x in diminished-1 format. $\overline{d(b)}$ represents the one's complement of d(b) and $\sum_{\oplus} d(a_k)$ used for summation of the numbers $d(a_k)$ in modulo 2^n+1 in diminished-1.

In [11] by using the Wallace tree and diminished-1 representation, an efficient method for modular multiplication modulo 2^n+1 is presented. By using this method there is no need to expand the adders to 2n bits for n bit numbers. The multiplication is done by a series of addition that in each addition step, carry must be complemented and used as a first bit for next addition step.

The formula for modular multiplication in diminished-1 of two numbers a and b modulo 2^n+1, that is used by this method, is as follow:

$$d(ba)=\left(\sum_{i=1}^{n-1}{}_{\oplus}\, b_i d(2^i a)\oplus\overline{Z}\oplus d_1(a)\right)+1 \ (\text{mod}\, 2^n+1) \qquad (19)$$

The first term in parenthesis should be done by diminished-1 addition and finally the result should be incremented by one.

A constraint for this method is that it cannot produce a true result if one of the operands was zero.

Zero can be detected by testing the most significant bit in diminished-1 representation so there is no need to add an additional hardware module for finding that an operand is 0 or not.

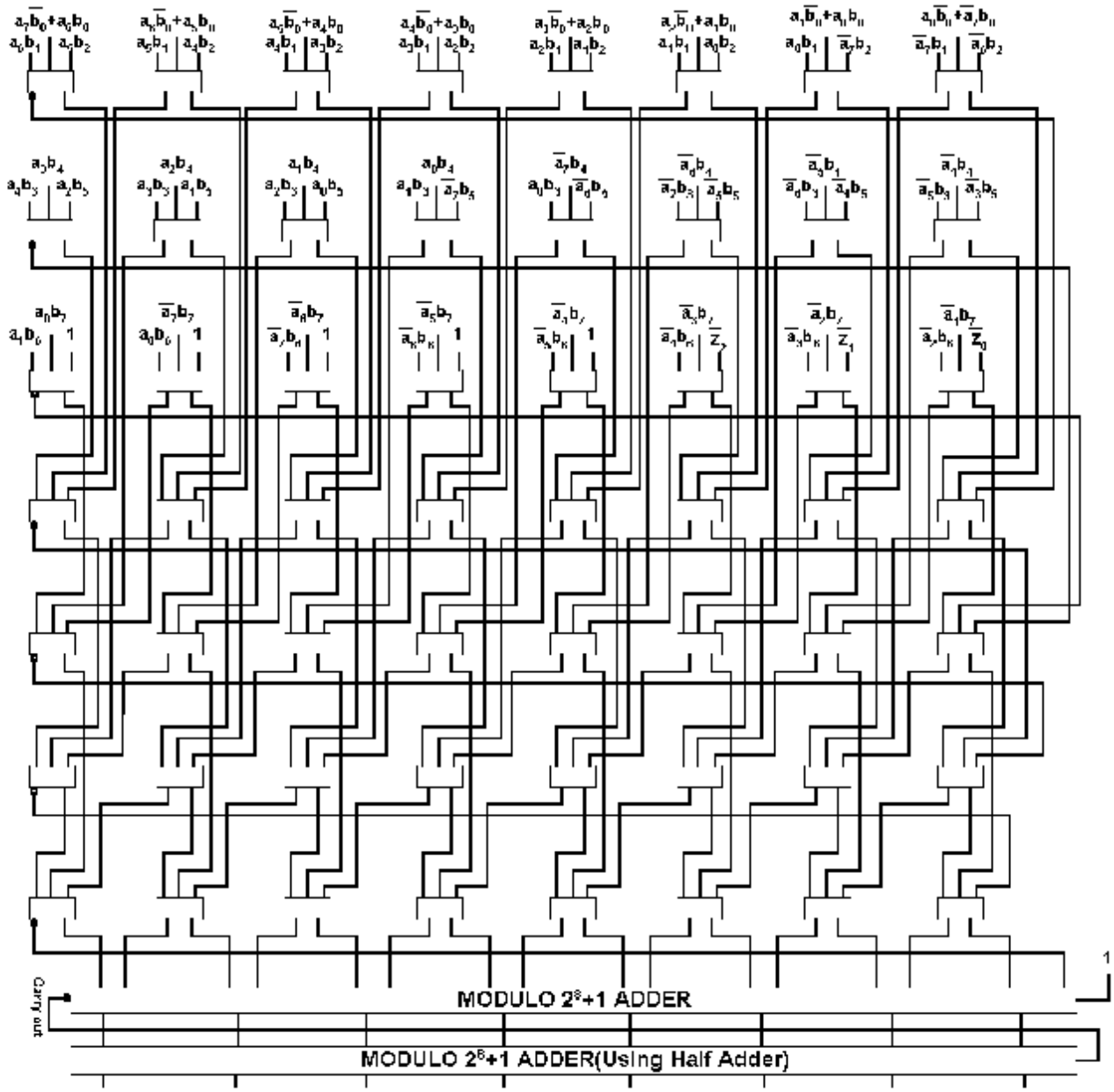

Figure 1. The 2^8+1 modular multiplication architecture

In this formula Z represent the number of zeros from bit b_1 to b_{n-1} and bar represent the one's complement of Z. $d_1(a)$ can be calculated by the following formula:

$$d_1(a) = \overline{b_0} d(a) + b_0 d(2a) \quad (20)$$

$d(2^i a)$ can be computed by the following relation:

$$d(2^i a) = \{a_{n-i-1}\, a_{n-i-2} \ldots a_0\, \overline{a_{n-1}}\, \overline{a_{n-2}} \ldots \overline{a_{n-i}}\} \quad (21)$$

in other words, multiplication by 2^i is accomplished by a cyclic shift of i bits to the left with the shifted bits being complemented.

An instance of the architecture for this multiplier modulo 2^8+1 is shown in Figure 1.

In this figure each rectangular represents a full adder block. As can be seen due to using Wallace tree, all additions are done by CSA architecture and there is no carry propagation in adders.

The additions that are taken in this multiplier are briefly shown in Figure 2.

For converting a binary number to diminished-1, one can use the methods that are proposed in [12].

For converting a binary number to diminished-1, one can use the methods that are proposed in [12].

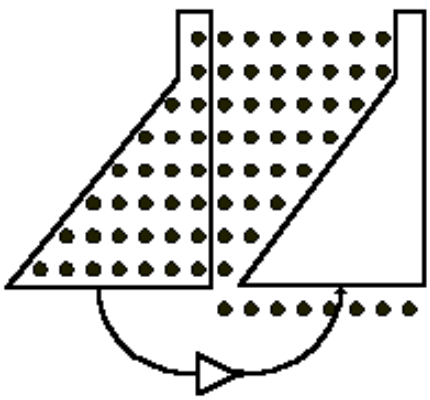

Figure 2. The additions that are taken in 2^8+1 multiplier

5.3 the 2^n-1 modular multiplier

In this section the architecture of 2^n-1 multiplier are proposed by using the architecture of 2^n+1 modular multiplier. This architecture is much similar to previous one but it's simpler and uses less area. Also in this implementation Wallace tree is used.

The property of operations in modulo 2^n-1 [9] differs from 2^n+1 multiplier as the following:

1) The output carry is used without any changes in first bit of addition, and no complement is necessary.

2)In 2^n-1 modular multiplier, there is no need to know number of zeros from b_1 to b_{n-1}. So this block is removed from architecture and as a result the speed is increased.

3)There is no need to compute $d_1(a)$, and the bits of b are used instead of it.

It is not required to distinguish zero from the other numbers. The output remains correct and the module produces zero number if each of the inputs were zero. So the hardware for zero detection and producing the zero output for this case, is removed.

With these differences one can find that this multiplier has more processing speed and less area than 2^n+1 multiplier. For instance the architecture of this multiplier is shown in Figure 3. In this figure two inputs are a and b. The additions that are done in this multiplier are shown in Figure 4.

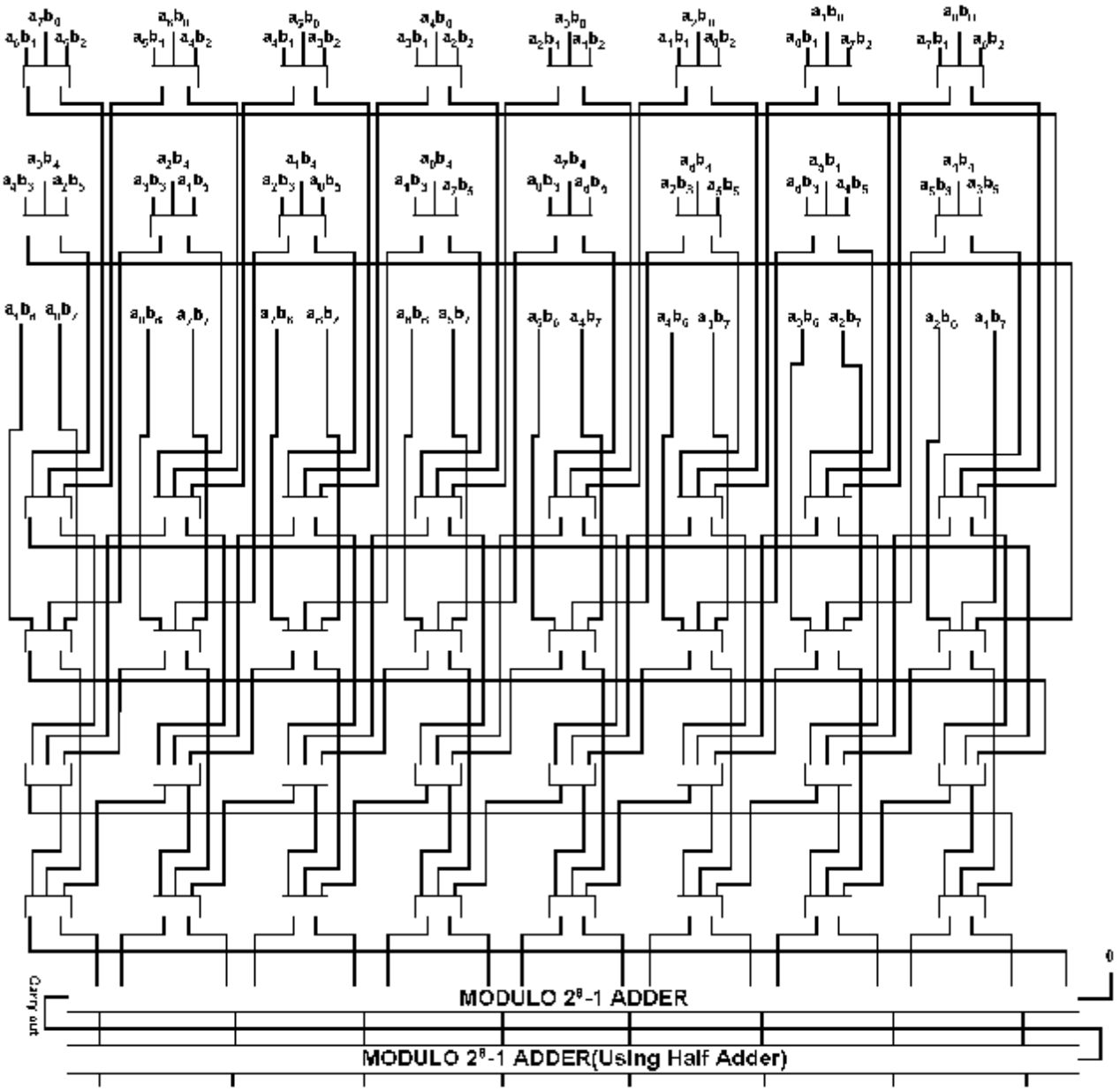

Figure 3. The 2^8-1 modular multiplication architecture

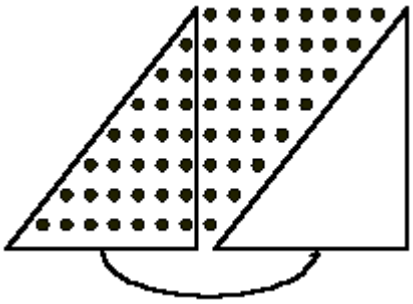

Figure 4. The additions that are taken in 2^8+1 multiplier

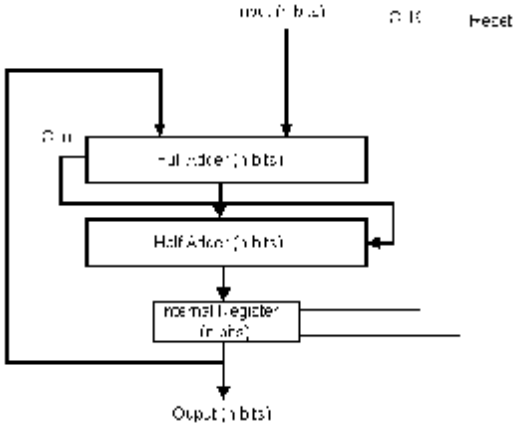

Figure 5. Modulo 2^n-1 adder architecture

5.4 the 2^n+1 modular adder

In this section the required architecture for 2^n+1 modular adder for RSA implementation in RNS is presented.

To use diminished-1 representation as input of this adder, the input has n+1 bits for a number with n bits. The task of this block is to add the input number with the value of internal register in modulo 2^n+1 and save the new result to this register and send it out.

The internal register is set to zero when resetting this block. Zero in diminished-1 represents the number 1, so the output is incremented by one. This fact changes the number from diminished-1 representation to binary number.

One maybe thinks that this representation change has an incorrect result, but as can be seen, this can be helpful for hardware implementations because the addition modulo 2^n+1 is the last operation of the second base extension and the result of these adders might be used as an input to system or send out as a result. If it is used as an input, it goes to modulo 2^n-1 multipliers so in both cases the number should not be in diminished-1 representation.

In this modular adder block the zero number should be discriminated while it mustn't change the output value. Testing the MSB can do this issue. For instance, this adder block is shown in Figure 5. In this architecture the clock is stopped when a MSB=1 this is done by a simple circuit that control the register from loading the new value. Two first additions are used for diminished-1 summation and carry correction.

5.5 the 2^n-1 modular adder

This adder is much similar to previous one but this architecture does not use diminished-1 representation, so there is no need to do carry correction that should be done in diminished-1 addition by complementing the carry and use it as a first bit of the addition.

A proposed architecture for this modular adder is shown in Figure 6. As shown there is no need to add a circuit to distinguish zero from other numbers and it can be used like the other numbers.

6. Comparisons

The results of the improved RNS are shown in Table 1 and 2 respectively for the number of modular multiplication and the memory for saving the constant values.

Table 1. Comparison based on the number of modular multiplications

	The number of needed multiplication	
Calculation of the value:	**Bajard method**	**Improved method**
σ_i	3	2
ξ_j	k+4	k+2
$\lvert\hat{r}\rvert_{m_r}$	k+3	k+2
α	k+1	k+1

Table 2. Comparison based on the number of memory

	The number of needed memory	
Calculation of the value:	**Bajard method**	**Improved method**
σ_i	2	1
ξ_j	4	2
$\lvert\hat{r}\rvert_{m_r}$	3	2
α	2	2

The results in these tables can be computed as shown in the context of the paper.

Note that the number of memory in table 2 is shown for one moduli so if one multiply this values by the number of modulus that are used in RNS, the difference between the Bajard method and the improved one is more distinguishable.

The results show, the number of multiplication is reduced about 33% in σ_i calculations and in other cases this reduction has been varied by the value of k (number of modulus in RNS bases). As can be seen in Bajard method, it requires 3k+11 multiplication but in improved method 3k+7 multiplication is required, so the number of multiplication is reduced by 400/(3k+11) percent.

As shown previously in improved method some redundant modules are removed and by this changes there is no change in clock period, also in some cases it decreases latency of circuit.

The number of used memories for saving the constant values is reduced about 50% in more cases. For instance if the number of modulus is 10, in σ_i calculation instead of saving 20 constants (2 constants for each moduli), 10 constants should be saved (1 constant for each moduli), a reduction of 50%.

This improved method is implemented by VHDL language for RSA-1024 with k=10 (number of modulus) and

synthesized with Leonardo spectrum 2002 for CMOS 0.6. The result of synthesis is shown in Table 3.

Table 3. Result of implementation

Area (Gates)	*1) Clock Speed (MHz)*	**Throughput Rate (Mb/s)**
878908	3.29	0.23

By comparison one can find this throughput is much better than Montgomery method without RNS. Also if one compares this new method by Bajard method with same architecture, the Bajard method has less throughput due to its extra multiplications and clock cycles.

7. Conclusions

The results show that Bajard RNS method, which is the fastest method reported till now, can be improved by some modifications in its algorithm's calculations and using the proper and fast hardware modules in each processing units. By this proposed architecture the RNS performance in hardware implementation of RSA cryptosystem is increased.
The new method presented in this paper increased the processing speed by reducing the number of multiplications, also decreased the used area by reducing number of constants that used the extra number of memories.
The new-presented architecture is the proper hardware implementation for the various main processing units of the modified RNS method.

References

[1] M. Soderstrom et al. ED, "RNS Arithmetic: Modern Application in DSP," IEEE Press, 1986.

[2] J. C. Bajard, L. Imbert, "A Full RNS Implementation of RSA," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 769-774, 2004.

[3] D. Pearson, "A Parallel Implementation of RSA, " *SAC'96*, 1996.

[4] K. C. Posch and R. Posch, "Residue Number System: a Key to Parallelism in Public Key Cryptography," *IEEE Symposium on parallel distributed processing*, pp. 432-435, 1992.

[5] T. V. Vu, "Efficient Implementation of the Chinese Reminder Theorem for Sign Detection and Residue Deconding," *IEEE Transactions on computers*, vol. C-34, pp. 646-651, 1985.

[6] A. Skavantzos and M. Abdallah, "Implementation Issues of the Two-level Residue Number System with Pairs of Conjugate Moduli," *IEEE Trans. on signal processing*, vol. 47, no. 3, 1999.

[7] C. McIvor, M. McLoone, J. V. McCanny, A. Daly and W. Marnane, "Fast Montgomery Modular Multiplication and RSA Cryptographic Processor Architectures, " Department of electrical and electronic engineering, University college Cork, 2003.

[8] A. Skavantzos and M. Abdallah, "Implementation Issues of the Two-level Residue Number System with Pairs of Conjugate Moduli," *IEEE Trans. on Signal Processing*, vol. 47, nol. 3, pp. 826- 838, 1999.

[9] B. Parhami, "Computer Arithmetic, " Oxford university press, 2000.

[10] L. M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform," *IEEE Trans. Acoust. Speech, Signal Processing*, vol. ASSP-24, pp. 356-359, 1976.

[11] Z. Wang, G. A. Jullien and W. C. Miller, "An Efficient Tree Architecture for Modulo 2^n+1 Multiplication," *Journal of VLSI signal processing*, vol. 14, no. 3, pp. 241-8, 1996.

[12] K. Manochehri and S. Pour Mozafari, "Verification in Methods of Converting to Diminished-1 in 2n+1 Modular Multiplication," *IKT2005*, Iran, 2005

Kooroush Manochehri Kalantari received the B.S. degree in computer engineering from the Azad university (Central branch), Iran, in 2001 and the M.S. degree in Computer engineering (with honors) from the department of computer engineering at Amirkabir University of Technology, Iran, in 2005. He is a Ph.D. candidate in Computer engineering at the department of computer engineering at Amirkabir University of Technology. His research interests are in the areas of cryptography, quantum computing and arithmetic processors.
Email: Kmanochehri@ce.aut.ac.ir

Saadat Pour Mozafari is currently an assistant professor at the department of computer engineering and IT, Amirkabir university of technology. He received his M.S. degree in microelectronics from ASU (Arizona State University) and his Ph.D. degree from New Mexico State University (New mexico USA) in electrical and computer engineering in 1999. His current research interest includes cryptography, computer testing and testable design, VLSI design, fabrication and fault tolerant systems.
Email: saadat@ce.aut.ac.ir

Babak Sadeghiyan received his M.S. in Electrical Engineering from Amirkabir University of Technology, Iran, in 1989, and his Ph.D. in Computer Science from University College, University of New South Wales, Australia, in 1993. His major research interests are cryptology, especially design and cryptanalysis of Block Ciphers, and network security, especially on Intrusion Detection Systems. He is currently an Associate Professor of Computer Engineering and IT Department at Amirkabir University of Technology.
Email: basadegh@ce.aut.ac.ir