Implementation of RSA Algorithm Based on RNS Montgomery Multiplication

Hanae Nozaki, Masahiko Motoyama, Atsushi Shimbo, and Shinichi Kawamura

Corporate Research and Development Center, Toshiba Corporation
1, Komukai Toshiba-cho, Saiwai-ku, Kawasaki 212-8582, Japan
{hanae.nozaki, masahiko.motoyama, atsushi.shimbo,
shinichi2.kawamura}@toshiba.co.jp

Abstract. We proposed a fast parallel algorithm of Montgomery multiplication based on Residue Number Systems (RNS). An implementation of RSA cryptosystem using the RNS Montgomery multiplication is described in this paper. We discuss how to choose the base size of RNS and the number of parallel processing units. An implementation method using the Chinese Remainder Theorem (CRT) is also presented. An LSI prototype adopting the proposed Cox-Rower Architecture achieves 1024-bit RSA transactions in 4.2 msec without CRT and 2.4 msec with CRT, when the operating frequency is 80 MHz and the total number of logic gates is 333 KG for 11 parallel processing units.

Keywords: RSA cryptography, residue number systems, Montgomery multiplication, modular exponentiation

1 Introduction

Computational performance of large integers is important in the implementation of public key cryptography and digital signature. We proposed a fast parallel Montgomery multiplication algorithm based on Residue Number Systems (RNS) [1]. In RNS, an integer is represented by a set of its residues in terms of base elements of RNS, and thus addition, subtraction, and multiplication can be independently carried out for every base element. On the other hand, Montgomery multiplication is a method for performing modular multiplication by substituting addition and multiplication for division. Therefore, the combination of RNS and Montgomery multiplication is expected to be well suited to parallel processing of modular exponentiation, and several studies concerning it have been reported [2], [3], [4].

The main purpose of our previous paper [1] was to improve the base transformation algorithm which consumes most of the processing time in the RNS Montgomery multiplication. We also proposed a hardware "Cox-Rower Architecture" suitable for the RNS Montgomery multiplication. The base transformation operation is efficiently realized by the Cox-Rower Architecture where Rower units perform parallel processing in cooperation with one Cox unit. Based on

Ç.K. Koç, D. Naccache, and C. Paar (Eds.): CHES 2001, LNCS 2162, pp. 364–376, 2001.

this architecture, the performance of 1 Mbps has been estimated for 1024-bit RSA cryptosystem at the operating frequency of 100 MHz.

In this paper, we investigate an implementation of RSA cryptosystem using the proposed RNS Montgomery multiplication algorithm, and design an RSA LSI to confirm the performance and feasibility of the proposed algorithm. As implementation methods, RSA decryption procedures without and with the Chinese Remainder Theorem (CRT) are presented. The Cox-Rower Architecture is characterized by the scalability for operating time and chip size depending on the number of Rower units. In implementation, the relation between the number of Rower units and the base size in RNS representation becomes important for the performance, because operations for each base element are performed in parallel at Rower units. For an LSI prototype using 0.25 μm CMOS, we obtain 4.2 msec for 1024-bit RSA cryptosystem without CRT and 2.4 msec with CRT. This result is comparable with the present best performance of commercial chips.

The organization of the paper is as follows: In the next section, the RNS Montgomery multiplication algorithm proposed in Ref. [1] is surveyed. In Sec. 3, we present RSA decryption procedures without and with CRT, and discuss the base size of RNS and the number of parallel processing units from the viewpoint of implementation. In Sec. 4, for the designed LSI, a hardware structure and its specifications are described. Finally, a short summary is given in Sec. 5.

2 Algorithm

2.1 Residue Number Systems

In RNS, an integer x is represented by

$$\langle x \rangle_a = (x[a_1], x[a_2], \ldots, x[a_n]), \tag{1}$$

where $x[a_i] = x \bmod a_i$. The set $a = \{a_1, a_2, \ldots, a_n\}$ is called a base and the number of elements n is its base size. The elements are required to satisfy $\gcd(a_i, a_j) = 1$ for $i \neq j$. CRT assures that the integer x which satisfies $0 \leq x < A$ ($A = \prod_{i=1}^{n} a_i$) is uniquely represented by $\langle x \rangle_a$.

The RNS representation has an advantage in which addition, subtraction, and multiplication can be realized by modular addition, subtraction, and multiplication for each RNS element as follows:

$$\langle x \pm y \rangle_a = ((x[a_1] \pm y[a_1])[a_1], \ldots, (x[a_n] \pm y[a_n])[a_n]), \tag{2}$$

$$\langle x \cdot y \rangle_a = ((x[a_1] \cdot y[a_1])[a_1], \ldots, (x[a_n] \cdot y[a_n])[a_n]), \tag{3}$$

which enables parallel computation using n processing units. However, we have not known how to perform comparison and division efficiently based on the RNS representation. To overcome this disadvantage, combination with Montgomery multiplication has been proposed [1], [2], [3], [4].

2.2 Montgomery Multiplication

Montgomery multiplication is known to be an efficient method for implementing modular exponentiation used in public key cryptographies. In the algorithm shown below, the inputs are x, y, and N ($x, y < N$) and the output is $w \equiv xyR^{-1} \pmod{N}$ ($w < 2N$), where $\gcd(R, N) = 1$ and $N < R$.

1: $s \leftarrow x \cdot y$
2: $t \leftarrow s \cdot (-N^{-1}) \bmod R$
3: $u \leftarrow t \cdot N$
4: $v \leftarrow s + u$
5: $w \leftarrow v/R$

The Montgomery constant R is chosen so as to make division in steps 2 and 5 simple. For example, R is generally set to 2's power in a radix 2 representation.

It is characteristic of Montgomery multiplication to perform modular multiplication by substituting addition and multiplication for division. Since the advantage of RNS is that addition, subtraction, and multiplication can be independently performed for each RNS element, the combination of RNS and Montgomery multiplication is expected to realize fast parallel processing effectively.

2.3 RNS Montgomery Multiplication

The RNS Montgomery multiplication algorithm proposed in Ref. [1] is briefly described in this subsection. The above-mentioned Montgomery multiplication procedure is rewritten by using RNS as shown in Fig. 1. Two bases a and b are introduced, and B ($= \prod_{i=1}^{n} b_i$) is used as the Montgomery constant. We assume here both a and b have the base size n, and denote the RNS representation of x based on a and b by $\langle x \rangle_{a \cup b}$ or simply by $\langle x \rangle$. The bases a and b satisfy $A, B > 8N$, $\gcd(A, B) = 1$, and $\gcd(B, N) = 1$.

Function: $\langle w \rangle_{a\cup b} = \mathrm{MM}(\langle x \rangle_{a\cup b}, \langle y \rangle_{a\cup b}, N)$	
Input: $\langle x \rangle_{a\cup b}$, $\langle y \rangle_{a\cup b}$ $(x, y < 2N)$	
Output: $\langle w \rangle_{a\cup b}$ $(w \equiv xyB^{-1} \pmod{N},\ w < 2N)$	
Base a operation	Base b operation
1: $\langle s \rangle_a \leftarrow \langle xy \rangle_a$	$\langle s \rangle_b \leftarrow \langle xy \rangle_b$
2:	$\langle t \rangle_b \leftarrow \langle s(-N^{-1}) \rangle_b$
3:	$\langle t \rangle_a \leftarrow \mathrm{BT}(\langle t \rangle_b, 0)$
4: $\langle u \rangle_a \leftarrow \langle tN \rangle_a$	
5: $\langle v \rangle_a \leftarrow \langle s + u \rangle_a$	
6: $\langle w \rangle_a \leftarrow \langle vB^{-1} \rangle_a$	
7:	$\langle w \rangle_b \leftarrow \mathrm{BT}(\langle w \rangle_a, 0.5)$

Fig. 1. RNS Montgomery multiplication algorithm.

Steps 3 and 7 in Fig. 1 are base transformation (BT) between a and b (see Fig. 2). According to CRT, x in radix representation is calculated from $\langle x \rangle_a$ by

$$x = \sum_{i=1}^{n} \xi_i A_i \bmod A, \tag{4}$$

where $\xi_i = x[a_i]A_i^{-1}[a_i] \bmod a_i$ and $A_i = A/a_i$. Equation (4) is rewritten by

$$x = \left(\sum_{i=1}^{n} \xi_i A_i \right) - kA, \tag{5}$$

with an unknown parameter k. Dividing both sides of Eq. (5) by A, we obtain

$$k = \left\lfloor \sum_{i=1}^{n} \frac{\xi_i}{a_i} \right\rfloor, \tag{6}$$

from $0 \le x/A < 1$ and $k \le \sum_{i=1}^{n} \xi_i/a_i < k+1$. Figure 2 shows a procedure of the base transformation from a to b. In this procedure, k is approximated by

$$\hat{k} = \left\lfloor \sum_{i=1}^{n} \frac{\mathrm{trunc}(\xi_i)}{2^r} + \alpha \right\rfloor, \tag{7}$$

where $\mathrm{trunc}(\xi_i)$ is a function to approximate ξ_i by its most significant g $(< r)$ bits: i.e. $\mathrm{trunc}(\xi_i) = \xi_i \bigwedge (\overbrace{1 \ldots 1}^{g} \overbrace{0 \ldots 0}^{(r-g)})_{(2)}$, $\bigwedge$ means a bitwise AND operation, and r is the bit length of processing units. An offset value α is required as a correction caused by the approximation, and is set to 0 at step 3 and 0.5 at step 7 in Fig. 1. The parameter $\hat{k}$ is computed recursively by k_i as shown at steps 4-6 in Fig. 2, where k_i satisfies $\hat{k} = \sum_{i=1}^{n} k_i$ and $k_i \in \{0, 1\}$.

Function: $\langle x \rangle_b = \mathrm{BT}(\langle x \rangle_a, \alpha)$	
Input: $\langle x \rangle_a$, $\alpha = 0$ or 0.5	
Output: $\langle x \rangle_b$	
Precomputation: $\left\langle A_i^{-1} \right\rangle_a$, $\langle A_i \rangle_b$, $\langle -A \rangle_b$	
1:	$\xi_i = x[a_i]A_i^{-1}[a_i] \bmod a_i$
2:	$\sigma_0 = \alpha$, $y_{i0} = 0$
3:	For $j = 1, \ldots, n$
4:	$\sigma_j = \sigma_{j-1} + \mathrm{trunc}(\xi_{(i+1-j)})/2^r$
5:	$k_{(i+1-j)} = \lfloor \sigma_j \rfloor$
6:	$\sigma_j = \sigma_j - k_{(i+1-j)}$
7:	$y_{ij} = y_{i(j-1)} + \xi_{(i+1-j)} \cdot A_{(i+1-j)}[b_i] + k_{(i+1-j)} \cdot (-A)[b_i]$
8:	Next j
9:	$x[b_i] = y_{in} \bmod b_i$

Fig. 2. Base transformation algorithm.

Function: $\langle y \rangle_{a \cup b} = \mathrm{MEXP}(\langle x \rangle_{a \cup b}, d, N)$

Input: $\langle x \rangle_{a \cup b}$, $d = (d_\kappa, \ldots, d_1)_{(2^4)}$

Output: $\langle y \rangle_{a \cup b}$, s.t. $y = x^d B^{-(d-1)} \bmod N$

Precomputation: $\langle B_N \rangle_{a \cup b}$, s.t. $B_N = B \bmod N$

1: $\left\langle x_N^0 \right\rangle \leftarrow \langle B_N \rangle$
2: $\left\langle x_N^1 \right\rangle \leftarrow \langle x \rangle$
3: $\left\langle x_N^{i+1} \right\rangle \leftarrow \mathrm{MM}(\left\langle x_N^i \right\rangle, \langle x \rangle, N)$ (for $i = 1, \ldots, 14$)
4: $\langle y \rangle \leftarrow \left\langle x_N^{d_\kappa} \right\rangle$
5: For $i = \kappa - 1, \ldots, 1$
6: For $j = 1, \ldots, 4$
7: $\langle y \rangle \leftarrow \mathrm{MM}(\langle y \rangle, \langle y \rangle, N)$
8: Next j
9: $\langle y \rangle \leftarrow \mathrm{MM}(\langle y \rangle, \left\langle x_N^{d_i} \right\rangle, N)$
10: Next i

Fig. 3. RNS modular exponentiation algorithm.

As compared with the previous RNS Montgomery multiplication algorithm [2], the above-mentioned algorithm has an advantage in that the base transformation at step 7 in Fig. 1 is error-free and does not need extra steps for error correction. Moreover, the correction factor k_i is computed only by an adder as will be described in Sec. 4.1, which can make the hardware structure simpler than that in Ref. [2].

An exponentiation algorithm based on a 4-bit window method is realized by the RNS Montgomery multiplication as shown in Fig. 3. It is assumed that an input variable has been transformed previously into $x' = xB \bmod N$, because of the essential feature of the Montgomery multiplication in which the Montgomery constant B is introduced. From this assumption, we obtain $y = x^d B \bmod N$ as an output.

The clocks to perform the RNS Montgomery multiplication are $O(n^2/u)$, where u is the number of parallel processing units. The RNS modular exponentiation (MEXP) is realized as the iteration operation of the RNS Montgomery multiplication (MM), and the number of iterations is proportionate to the key length $|N|$. From $n \propto |N|$, the performance of the RNS modular exponentiation is consequently estimated by $O(n^3/u)$. This relation means that there is the scalability for performance and chip size depending on the number of parallel processing units u, since the chip size is determined by u.

3 Implementation

Figure 4 shows an RSA decryption procedure using the RNS modular exponentiation algorithm. In steps 1 and 2, modular arithmetic based on the radix 2 representation is required. We assume that this modular arithmetic is performed at a dedicated divider unit. In step 3, $\left\langle -N^{-1} \right\rangle_b$ which is used in MM() is calculated from $b_i - (N^{\lambda_i - 1} \bmod b_i)$, where λ_i, the Carmichael function [5] of

Input: C, d, N	
Output: $m = C^d \bmod N$	
1:	$B_N \leftarrow B \bmod N$
2:	$B_N^2 \leftarrow B^2 \bmod N$
3:	Compute $\left\langle -N^{-1} \right\rangle_b$
4:	Radix-RNS conversion: N, B_N, B_N^2, C
5:	$\langle C' \rangle \leftarrow \mathrm{MM}(\langle C \rangle, \left\langle B_N^2 \right\rangle, N)$
6:	$\langle m' \rangle \leftarrow \mathrm{MEXP}(\langle C' \rangle, d, N)$
7:	$\langle m \rangle \leftarrow \mathrm{MM}(\langle m' \rangle, \langle 1 \rangle, N)$
8:	RNS-Radix conversion: $\langle m \rangle_a$
9:	$m \leftarrow m - cN \quad (c = 0 \text{ or } 1)$

Fig. 4. RSA decryption algorithm.

b_i, is precomputed and stored in ROMs. Since steps 1–4 (except for C at step 4) depend only on the key N, it is effective to precompute these steps, if N is not changed frequently. As mentioned above, it is assumed in MEXP() that the input and the output are variables multiplied by the Montgomery constant B. From this condition, steps 5 and 7 are required as a transformation to get $C' = CB \bmod N$ and as its inverse transformation, respectively. Finally, step 9 is a correction to assure $m < N$, because m obtained in step 7 is less than $2N$.

Radix-RNS and RNS-Radix conversions are defined by

$$x[a_i] = \left(\sum_{j=0}^{n-1} x(j) \cdot 2^{r \cdot j}[a_i] \right) \bmod a_i, \tag{8}$$

$$x = (2^{r \cdot (n-1)}, \ldots, 2^r, 1) \sum_{i=1}^{n} \left[\xi_i \begin{pmatrix} A_i(n-1) \\ \vdots \\ A_i(1) \\ A_i(0) \end{pmatrix} - k_i \begin{pmatrix} A(n-1) \\ \vdots \\ A(1) \\ A(0) \end{pmatrix} \right], \tag{9}$$

where the notation $x(i)$ means the radix-2^r representation of x: i.e. $x = \sum_{i=0}^{n-1} x(i) \cdot 2^{r \cdot i}$. In the RNS-Radix conversion, carry propagation is needed after the summation has been finished.

3.1 Base Size and Number of Parallel Units

The operation shown in Fig. 4, except for steps 1 and 2 performed at the additional divider unit and the carry propagation in the RNS-Radix conversion, is independently carried out for every base element a_i and b_i at parallel processing units. Here, the base size n has the relation $n \geq \lceil (|N| + r)/r \rceil$ with the bit length $|N|$ of the modulus N. The number of parallel processing units u can be chosen in the range of $1 \leq u \leq n$. If $u < n$, time-sharing processing for some base elements is performed in each unit. When the base size n is fixed, RSA

transaction performance improves in proportion to u. Obviously, it is efficient to set u as a divisor of n in order to control all processing units by the same procedure. By choosing u appropriately, a variety of chips can be realized in terms of performance and size.

In the implementation, it is realistic that all parameters which depend only on the base sets a and b are precomputed and stored in ROMs. A chip loaded with base sets for an RSA key length L can deal with key lengths which are shorter than L. However, processing time of a key length l ($< L$) is reduced only to l/L as compared with that of the key length L, although $(l/L)^3$ is achieved ideally. The overhead time is caused by the fact that the performance of RNS Montgomery multiplication is determined from the base size, and thus the amount of operations does not decrease.

In order to perform an efficient computation for shorter key lengths, it is necessary to prepare some base sets for typical key lengths: e.g. 512, 1024, and 2048 bits. For these key lengths, minimum base sizes become 17, 33, and 65 in the case of $r = 32$. There are several implementation methods to deal with different-sized base sets. Among them, it is advantageous to set a base size to a multiple of u from the viewpoint of the simplicity of a control circuit. Therefore, if $u = 11$, appropriate base sizes are 22, 33, and 66 for key lengths 512, 1024, and 2048 bits, respectively. In this case, it is expected that 1024-bit and 2048-bit RSA processing has good performance, whereas 512-bit processing has overhead time.

3.2 CRT Mode

An RSA decryption procedure with CRT is given by

$$\begin{aligned} m &= (C^{d_p} \bmod p)(q^{-1} \bmod p)\, q + (C^{d_q} \bmod q)(p^{-1} \bmod q)\, p \pmod N \\ &= [(C^{d_p} \bmod p)(q^{-1} \bmod p) \bmod p]\, q \\ &\quad + [(C^{d_q} \bmod q)(p^{-1} \bmod q) \bmod q]\, p \pmod N, \end{aligned} \tag{10}$$

where $N = pq$, $d_p = d \bmod (p-1)$, and $d_q = d \bmod (q-1)$. A procedure to perform Eq. (10) is shown in Fig. 5. The operations for p and q are carried out sequentially. Precomputations of steps 1, 2, 4, and 5 (except for C_p and C_q) are effective, if the secret keys p and q are not changed frequently. Here, it should be noted that we need the RNS representation of m by means of the base $a \cup b$ in the RNS-Radix conversion at step 11, because the modulus N in RSA processing with CRT is represented uniquely not by a single base a or b but by the base $a \cup b$. In contrast, only $\langle m \rangle_a$ (or $\langle m \rangle_b$) is sufficient for the RNS-Radix conversion in Fig. 4. Step 12 is a correction to assure $m < N$ the same as step 9 in Fig. 4. In this case, m obtained in step 10 is less than $4N$, because u_p and u_q ($< 2N$) are added to each other without mod N operation.

The processing time of Fig. 5 is dominated by MEXP() at step 7. Since the base size n can be reduced to $n/2$ by adopting CRT, the processing time of MEXP() becomes 1/8 of that in Fig. 4. As a result, reduction of about 1/4 is achieved in total processing time, because the operations for p and q are

Input: C, d_p, d_q, N, p, q, $q_{\mathrm{inv}}(=q^{-1} \bmod p)$, $p_{\mathrm{inv}}(=p^{-1} \bmod q)$
Output: $m = C^d \bmod N$

	Operation for p	Operation for q
1:	$B_p \leftarrow B \bmod p$	$B_q \leftarrow B \bmod q$
2:	$B_p^2 \leftarrow B^2 \bmod p$	$B_q^2 \leftarrow B^2 \bmod q$
3:	$C_p \leftarrow C \bmod p$	$C_q \leftarrow C \bmod q$
4:	Compute $\left\langle -p^{-1} \right\rangle_b$	Compute $\left\langle -q^{-1} \right\rangle_b$
5:	Radix-RNS: p, q_{inv}, B_p, B_p^2, C_p	Radix-RNS: q, p_{inv}, B_q, B_q^2, C_q
6:	$\left\langle C_p' \right\rangle \leftarrow \mathrm{MM}(\langle C_p \rangle, \left\langle B_p^2 \right\rangle, p)$	$\left\langle C_q' \right\rangle \leftarrow \mathrm{MM}(\langle C_q \rangle, \left\langle B_q^2 \right\rangle, q)$
7:	$\left\langle m_p' \right\rangle \leftarrow \mathrm{MEXP}(\left\langle C_p' \right\rangle, d_p, p)$	$\left\langle m_q' \right\rangle \leftarrow \mathrm{MEXP}(\left\langle C_q' \right\rangle, d_q, q)$
8:	$\langle t_p \rangle \leftarrow \mathrm{MM}(\left\langle m_p' \right\rangle, \left\langle q_{\mathrm{inv}} \right\rangle, p)$	$\langle t_q \rangle \leftarrow \mathrm{MM}(\left\langle m_q' \right\rangle, \left\langle p_{\mathrm{inv}} \right\rangle, q)$
9:	$\langle u_p \rangle \leftarrow \mathrm{MUL}(\langle t_p \rangle, \langle q \rangle)$	$\langle u_q \rangle \leftarrow \mathrm{MUL}(\langle t_q \rangle, \langle p \rangle)$
10:	$\langle m \rangle \leftarrow \mathrm{ADD}(\langle u_p \rangle, \langle u_q \rangle)$	
11:	RNS-Radix conversion: $\langle m \rangle_{a \cup b}$	
12:	$m \leftarrow m - cN \quad (c = 0, 1, 2, \text{or } 3)$	

Fig. 5. RSA decryption algorithm with CRT.

performed sequentially. The same reduction ratio is obtained in a general case based on the radix 2 representation.

4 Prototype

We prototyped an LSI adopting the Cox-Rower Architecture. In this section, an outline of the LSI is described.

4.1 Architecture

The Cox-Rower Architecture was proposed as a hardware suitable for the RNS Montgomery multiplication [1]. The name is derived from its original structure where plural "Rower" units perform parallel processing in cooperation with one "Cox" unit which computes a correction factor in the base transformation.

A hardware structure of the Cox-Rower Architecture in this work is shown in Fig. 6. It consists of u sets of Rower units which individually have a multiplier-and-accumulator with modular reduction unit by base element a_i and b_i. Figure 6 is different from the original structure proposed in Ref. [1] in regard to the following two points:

(i) Rower units are connected by ring connection instead of by bus connection.
(ii) A Cox unit is embedded in every Rower unit.

In the base transformation, ξ_i $(i = 1, \ldots, n)$ which has been computed in each Rower unit needs to be transferred to the other Rower units. The original architecture uses bus connection for this transfer. We have found that ring connection can also realize the transfer of ξ_i's by sending them to an adjoining Rower unit in turn. In addition, since the original architecture has only one Cox unit, it also

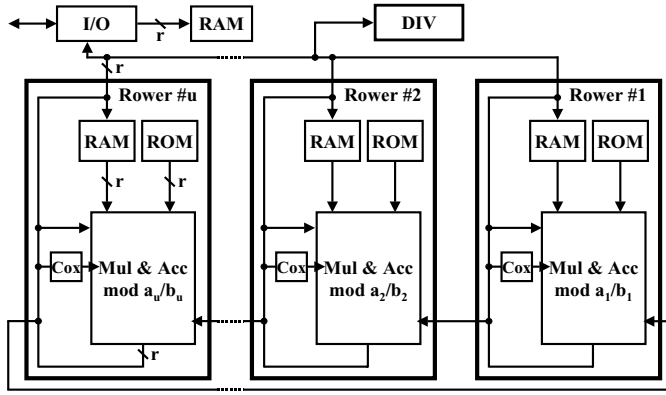

Fig. 6. Cox-Rower Architecture.

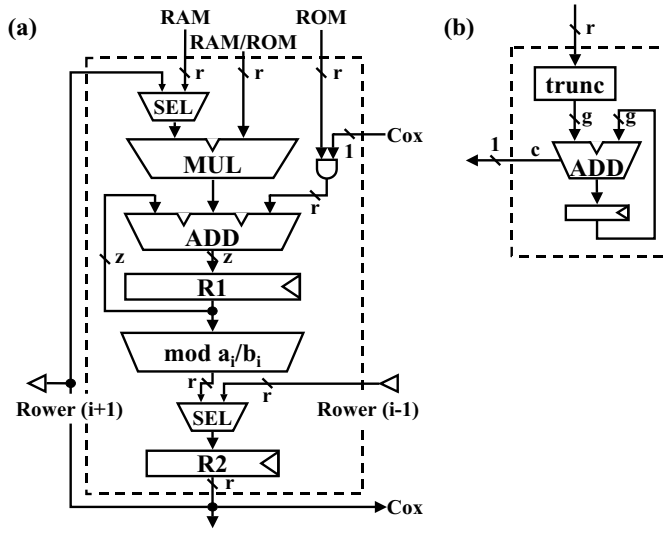

Fig. 7. Multiplier-and-accumulator (a) and Cox unit (b), where $r = 32$, $z = 72$, and $g = 9$.

needs bus connection to broadcast the correction factor k_i, which is computed in the Cox unit, to all Rower units. This broadcast is, however, avoidable by embedding a Cox unit in each Rower unit as shown in Fig. 6, which further enables us to control all Rower units by the same procedure. Consequently, we have adopted the ring connection in this work to lower data driving load and improve the modularity of Rower units.

Structures of the multiplier-and-accumulator and the Cox unit are shown in Fig. 7. The multiplier-and-accumulator has two stages: one is to accumulate a result of multiplication-and-addition and the other is to perform modular reduction by the base elements. The Cox unit consists of a truncation unit, a g-bit

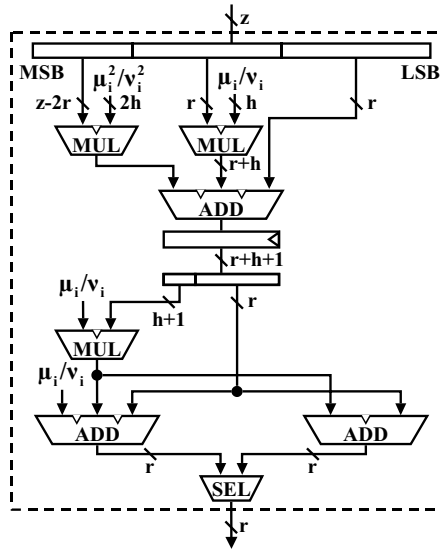

Fig. 8. Modular reduction unit, where $h = 10$.

adder, and a register in order to compute k_i in the base transformation according to steps 4–6 in Fig. 2. One of the advantages of the proposed RNS Montgomery multiplication algorithm is in this simple structure of the Cox unit.

Here, let us consider the base transformation procedure at Rower unit i. First, at step 1 in Fig. 2, ξ_i is calculated from $x[a_i]$ and $A_i^{-1}[a_i]$ and is stored in the register R2. Next, in the loop for $j = 1$, k_i is computed from ξ_i at the Cox unit, and then y_{i1} is obtained and stored in the register R1. Before the next loop for $j = 2$, ξ_{i-1} which has been computed at Rower unit $i - 1$ is transferred by ring connection and is stored in R2. Then, the loop for $j = 2$ is carried out based on ξ_{i-1}, and y_{i2} is obtained. After all loop processes for $j = 1, \ldots, n$ have been finished, $x[b_i]$ is computed from y_{in} at step 9 and is stored in R2.

Figure 8 shows a structure of the modular reduction unit in Fig. 7(a). The base elements a_i and b_i can be given as $2^r - \mu_i$ and $2^r - \nu_i$. Small integers μ_i and ν_i ($\ll 2^r$) are chosen so as to make the base elements coprime. In this case, modular computation for a_i and b_i can be realized by multipliers and adders, where the multipliers perform multiplication by μ_i and ν_i as shown in Fig. 8. The maximum bit length h of μ_i and ν_i is 10 bits for the base size $n = 66$. With respect to the output y of the operation $x \bmod a_i$, the modular reduction unit has the condition $y < 2^r$ instead of $y < a_i$, i.e. if $x \bmod a_i < \mu_i$, $y = x \bmod a_i + a_i$. We have ascertained that this condition does not affect the RNS Montgomery multiplication algorithm.

The Cox-Rower Architecture additionally has a divider which performs division based on the radix 2 representation. This divider is required for steps 1 and 2 in Fig. 4 and steps 1–3 in Fig. 5.

As described above, the Cox-Rower Architecture is designed to be suitable for the RNS Montgomery multiplication algorithm, particularly for the base transformation algorithm. The other operations such as the Radix-RNS and the RNS-Radix conversions can also be implemented efficiently in this architecture.

4.2 Specifications

The specifications of the LSI prototype are summarized in Table 1. In the LSI, the standard length of 32 bits is adopted as the bit length of processing units r. The number of Rower units u is set to 11 from the consideration for chip size. We can use the base sizes of 22, 33, and 66, which realize maximum key lengths of 672, 1024, and 2080 bits, respectively. Therefore, key lengths up to 2048 bits without CRT and 4096 bits with CRT are available.

SHA-1 which is required as a Hash function in digital signature is additionally implemented in the LSI. The SHA-1 core has an MGF1 function used in RSA standard spec PKCS#1 Ver.2.0 [6].

In Rower unit i ($i = 1, \ldots, 11$), operations for the base elements a_j and b_j are performed, where $j = i + 11\ell$ ($\ell = 0, \ldots, 5$). Thus, parameters in terms of a_j and b_j are stored in ROM of Rower unit i. Table 2 lists the parameters. These parameters for the three base sizes $n = 22$, 33, and 66 are prepared in the LSI, which needs the memory size shown in Table 3. In this table, memory sizes in the case of shorter maximum key lengths are also estimated. Since the LSI has been designed to provide long key lengths such as 2048 and 4096 bits, the increase in memory size causes a big core size. It is possible to reduce the memory size depending on maximum key lengths as shown in Table 3.

Figure 9 shows the details of the processing time. The transactions of I/O and precomputation for keys are negligible in the total processing time and

Table 1. Specifications.

Process	0.25 μm CMOS
Operating frequency	80 MHz
Operating voltage	2.5 V
Functions	RSA without and with CRT
	SHA-1 Hash code generation
	MGF1 (PKCS#1 Ver.2.0)
Performance	1024-bit RSA: 4.2 ms / 2.4 ms
(without/with CRT)	2048-bit RSA: 29.2 ms / 8.9 ms
	4096-bit RSA: — / 60.4 ms
Core size	6.9 mm × 6.9 mm
No. of Rower units	11
No. of logic gates	333 KG (Total)
	221 KG (RSA core)
	36 KG (Divider)
	57 KG (SHA-1)
	19 KG (I/O etc.)

Table 2. Parameters stored in ROM.

Base transformation	
$\langle x \rangle_a \Longrightarrow \langle x \rangle_b$	$A_j^{-1}[a_j]$, $A_1[b_j]$, ... , $A_n[b_j]$, $-A[b_j]$
$\langle x \rangle_b \Longrightarrow \langle x \rangle_a$	$B_j^{-1}[b_j]$, $B_1[a_j]$, ... , $B_n[a_j]$, $-B[a_j]$
Radix-RNS conversion	
$x \Longrightarrow \langle x \rangle_a$	$2^r[a_j]$, $2^{r \cdot 2}[a_j]$, ... , $2^{r \cdot (n-1)}[a_j]$
$x \Longrightarrow \langle x \rangle_b$	$2^r[b_j]$, $2^{r \cdot 2}[b_j]$, ... , $2^{r \cdot (n-1)}[b_j]$
RNS-Radix conversion	
$\langle x \rangle_a \Longrightarrow x$	$A_1(j-1)$, ... , $A_n(j-1)$, $-A(j-1)$
$\langle x \rangle_{a \cup b} \Longrightarrow x$ (CRT mode)	$(AB/a_j)^{-1}[a_j]$, $(AB/b_j)^{-1}[b_j]$, $(AB/a_1)(j-1)$, ... , $(AB/a_n)(j-1)$, $(AB/b_1)(j-1)$, ... , $(AB/b_n)(j-1)$, $-AB(j-1)$

Table 3. Memory size.

Maximum RSA key length	ROM (KByte)	RAM (KByte)
2048 & 4096 (CRT) *	209	24
2048 & 2048 (CRT)	138	20
1024 & 2048 (CRT)	57	12

* Designed LSI

are not exhibited in this figure. It is found that the contribution from division based on the radix 2 representation becomes large in the processing of shorter key lengths and in CRT mode. The latter condition means that the number of parameters which are computed in the divider increases in CRT mode. In a comparison between the performance without and with CRT, reduction ratio of 0.3 is obtained in 2048-bit RSA processing, which is close to an ideal ratio of 1/4. However, reduction ratio becomes 0.5 in 1024-bit processing. This increase in reduction ratio is due to the use of a redundant base size in CRT mode. In non-CRT mode of 1024-bit processing, we use the base size $n = 33$ which is optimum for this key length. In contrast, the base size $n = 22$ used in CRT mode is too long for the key length of 512 bits. These results indicate that base

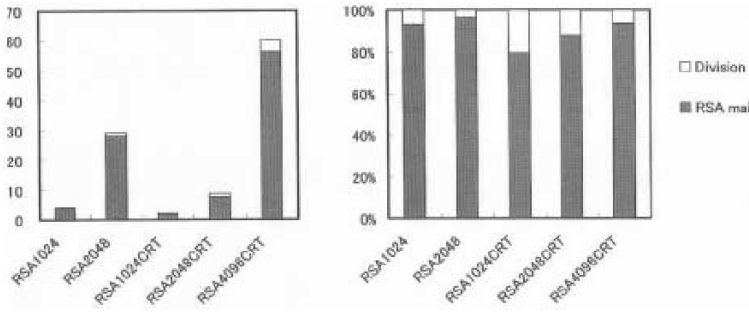

Fig. 9. Performance.

sizes strongly affect the performance of the Cox-Rower Architecture as discussed in Sec. 3.1.

At present, the best performance of 1024-bit RSA processing in commercial chips is, as far as we know, reported for Rainbow's chip (5 msec with CRT) [7] and Pijnenburg's chip (3 msec and 1.5 msec with CRT) [8], which is comparable with the performance in this work. The Cox-Rower Architecture can equip up to 33 Rower units for 1024-bit RSA processing. In that case, three-times speedup can be realized and the processing time which is less than 1 msec becomes feasible.

5 Conclusions

This paper presented the implementation of RSA algorithm based on the RNS Montgomery multiplication. We showed RSA decryption procedures and discussed the relation between the base size of RNS and the number of parallel processing units. The designed LSI adopting the Cox-Rower Architecture can deal with key lengths up to 4096 bits in CRT mode. Using 11 Rower units, we obtained 1024-bit RSA transactions in 4.2 msec without CRT and 2.4 msec with CRT, at the operating frequency of 80 MHz. This result gives us a prospect of realizing a high performance. Downsizing of chips and speedup by using more Rower units are subjects to be tackled in the next phase of this work.

Acknowledgment

The authors would like to thank Hidekazu Shimizu of Toshiba Information Systems Corporation for his collaboration in LSI design.

References

1. S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-Rower Architecture for Fast Montgomery Multiplication," EUROCRYPT 2000, pp. 523–538 (2000).
2. K. C. Posch and R. Posch, "Modulo Reduction in Residue Number Systems," IEEE Tr. Parallel and Distributed Systems, Vol. 6, No. 5, pp. 449–454 (1995).
3. J.-C. Bajard, L.-S. Didier, and P. Kornerup, "An RNS Montgomery Multiplication Algorithm," Proceedings of ARITH13, IEEE Computer Society, pp. 234-239 (1997).
4. P. Paillier, "Low-Cost Double-Size Modular Exponentiation or How to Stretch Your Cryptoprocessor," PKC99, pp. 223–234 (1999).
5. E. Kranakis, "Primality and Cryptography," Wiley-Teubner Series in Computer Science, John Willy & Sons (1986).
6. RSA Laboratories, "PKCS#1 Ver.2.0: RSA Cryptography Standard," Oct. 1 (1998).
7. `http://www.rainbow.com/cryptoswift`.
8. `http://www.pcc.pijnenburg.nl/pcc-ises.htm`.