Journal of Systems Architecture 55 (2009) 355–362

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

An efficient signed digit montgomery multiplication for RSA

Daesung Lim [a], Nam Su Chang [a], Sung Yeon Ji [a], Chang Han Kim [b,*], Sangjin Lee [a,1], Young-Ho Park [c,2]

[a] *Center for Information and Security Technologies (CIST), Korea University, Seoul, Republic of Korea*
[b] *Dept. of Information and Security, Semyeong University, Jechon, Republic of Korea*
[c] *Dept. of Information Security Systems, Sejong Cyber University, Seoul, Republic of Korea*

ARTICLE INFO

Article history:
Received 1 May 2007
Received in revised form 13 February 2009
Accepted 29 April 2009
Available online 9 May 2009

Keywords:
Montgomery multiplication
Signed digit adder
Signed digit number representation

ABSTRACT

In this paper we present an efficient Montgomery multiplier using the signed digit number representation suitable for modular exponentiation, which is the main operation of RSA. The multiplier consists of one level of signed digit adder plus multiplexer through a precomputation. We design the multiplier with the improved signed digit adder using SAMSUNG STD 130 0.18 μm 1.8 V CMOS Standard Cell Library and compare to multipliers with other previous adders. The proposed modular multiplier can be applied to public key cryptosystems based on integer arithmetic such as RSA, DSA or ECC.

1. Introduction

With recent developments in internet technology secure services, such as on-line shopping and data downloading, are growing their markets supported by digital signature and user authentication techniques with public key cryptosystems. RSA is the simplest and the most widely used public key cryptosystem [1]. Server systems which operate RSA must provide high-performance to process a large number of requests from various types of clients, and therefore a speed-up for RSA is essentially required.

It is the main operation of RSA to compute modular exponentiation by repeated modular multiplications. Many modular multiplication algorithms have been proposed in [2–4]. Montgomery multiplication is an efficient method for a modular multiplication with an arbitrary modulus and is particularly suitable for implementation on general purpose computers and embedded microprocessors [4,5].

In this paper we deal with Montgomery multiplication algorithms for modular exponentiation in RSA. There are three aspects. The first is how to avoid last subtractions, which are generated from each multiplication. The second is how to simply determine the addition with the modulus, and the last is how to efficiently calculate the addition with three operands. The problems of first and second aspects are solved in [6,7], respectively. We focus on the final aspect.

The core operation of Montgomery multiplication is two additions for three operands. In the addition of two numbers represented in the conventional binary number representation, the carry may propagate all the way from the least significant bit (LSB) to the most significant bit (MSB). The addition time is thus dependent on the bit length. To avoid the carry for the addition, there are two approaches using carry save (CS) number representation [8–10] and signed digit (SD) number representation [11–17]. In the SD number representation, a number can be represented in more than one way. Therefore, there can be more various gate expressions than CS number representation.

To realize a higher speed multiplier than previous multipliers, we design Montgomery multiplier using the SD number representation. Therefore, the input and output of the multiplier are SD numbers and the addition of three operands is performed with one level of SDA plus multiplexer through a precomputation. We implement the multiplier using SAMSUNG STD 130 0.18μm 1.8 V CMOS Standard Cell Library. The proposed modular multiplier can be applied to public key cryptosystems based on integer arithmetic such as RSA, DSA or ECC.

The paper is organized as follows. Section 2 presents Montgomery multiplication algorithm and its modifications for binary exponentiation, and the SD number representation is represented. In Section 3, we present an SD Montgomery multiplier with an improved SDA. We discuss the efficiency of our multiplier in Section 4. Finally, some conclusions are given in Section 5.

[*] Corresponding author. Tel.: +82 43 649 1281; fax: +82 43 649 1747.
E-mail addresses: daesung.lim@samsung.com (D. Lim), ns-chang@korea.ac.kr (N.S. Chang), jisy0522@cist.korea.ac.kr (S.Y. Ji), chkim@semyung.ac.kr (C.H. Kim), sangjin@korea.ac.kr (S. Lee), youngho@cybersejong.ac.kr (Y.-H. Park).

[1] This work was supported by the Second Brain Korea 21 Project.

[2] This work was supported by Grant No. R01-2005-000-11261-0 from Korea Science and Engineering Foundation in Ministry of Science and Technology.

1383-7621/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2009.04.001

2. Preliminaries

2.1. Montgomery multiplication

Let the modulus N be a n-bit odd integer, and let R be 2^n, i.e., $gcd(R,N) = gcd(2^n,N) = 1$. A method is described for computing $XYR^{-1} mod\ N$ without using classical modular reduction, we first define the N-residue of an integer X and Y such that $0 \leqslant X, Y < N$. Let $\widetilde{X} = XR\ mod\ N$ and $\widetilde{Y} = YR\ mod\ N$. Given two N-residues $\widetilde{X}$ and $\widetilde{Y}$, the montgomery multiplication is defined as the N-residue

$$\widetilde{A} = \widetilde{X}\tilde{\cdot}\widetilde{Y} mod\ N = \widetilde{X}\widetilde{Y}R^{-1} mod\ N = XRYRR^{-1} mod\ N = XYRmod\ N$$
$$= ARmod\ N,$$

where R^{-1} is the inverse of R modulo N. We use $\tilde{\cdot}$ to denote the montgomery multiplication. So applying montgomery multiplication to $\widetilde{X}$ in an exponentiation algorithm is going to produce $\widetilde{X^e}$ rather than $(\widetilde{X})^e$ [4].

Algorithm 1. Montgomery multiplication (X, Y, N)

```
Input: $X = \sum_{i=0}^{n-1} x_i 2^i, Y = \sum_{i=0}^{n-1} y_i 2^i, N = \sum_{i=0}^{n-1} n_i 2^i$
Output: $A = XYR^{-1}$mod $N$, where $R = 2^n$mod $N$
1: $A \leftarrow 0$. $(A = \sum_{i=0}^{n} a_i 2^i)$
2: For $i$ from 0 to $n-1$ do:
3:     $u_i \leftarrow a_0 + x_i y_0$mod2
4:     $A \leftarrow (A + x_i Y + u_i N)/2$
5: If $A \geqslant N$, then $A \leftarrow A - N$
6: Return($A$).
```

The Montgomery multiplication algorithm calculates the value $A = X \cdot Y \cdot R^{-1}\ mod\ N$ where R is a constant number usually $R = 2^n$. The n-bit value N has to be an integer satisfying the condition $\gcd(R,N) = 1$. If N is considered odd, like in RSA, the above constraint is always true.

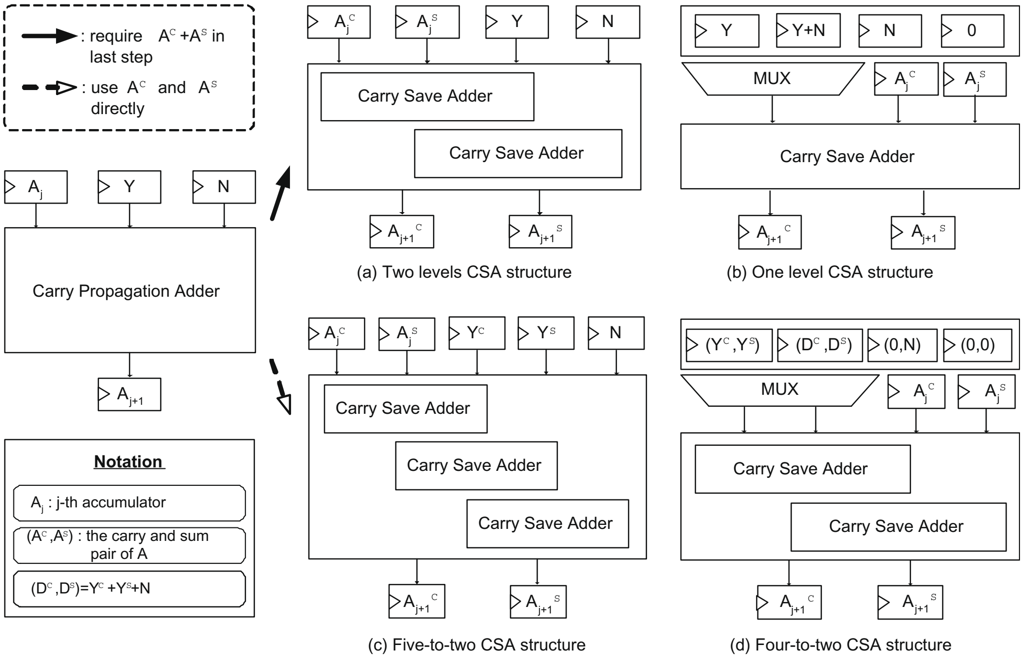
In the Montgomery multiplication, one reduction is needed for each multiplication. Because the input has the restriction $X, Y < N$ and the output A is bounded by $A < 2N$. As a consequence, if $A \geqslant N$, N must be subtracted so that the output can be used as input of the next multiplication. To avoid this subtraction, a bound for R is known such that for input $X, Y < 2N$ the output is also bounded by $A < 2N$ [6]. When u_i is calculated on step 3 of Algorithm 1, an addition of the modulus N is decided by checking the least significant bit (LSB) of A where Y is expanded one bit more and the LSB of Y is zero [7].

The most consuming part of the Algorithm 1 is the calculation of A given by the three input addition (step 4). It comes from the carry propagation of the very large operand additions and this has been avoided by using carry save adder (CSA) for additions [8–10]. The approach using CSA is represented in Fig. 1. Especially, in terms of time complexity, Fig. 1d method is well known for efficient architecture suitable for exponentiation [9].

2.2. Signed digit number representation

The SD number representation was first proposed by Avizienis to make it possible to perform the addition without carry propagation [11]. In the binary SD number representation, three possible digits $\{0, 1, -1\}$ can be encoded by using two bits. The best-known two encodings are Sign-and-Value and Plus-and-Minus methods. In the Sign-and-Value encoding, as it was in standard Sign-and-Value, the MSB of two is the sign and the other is the value, which denotes by $x_i = (x_i^s, x_i^v)$ [12,18,19]. In the Plus-and-Minus encoding, the value of each digit is calculated according to the equation $x_i = x_i^p - x_i^m$, which denotes by $x_i = (x_i^p, x_i^m)$, where x_i^p is a positive bit and x_i^m is a negative one [14,17].

The addition based on the SD number representation consists of two steps. In the first step, an intermediate carry c_{i+1} and sum s_i are generated, based on the operand digits x_i and y_i at each digit position i. In the second step, the summation $a_i = c_i + s_i$ is carried out to produce the final sum digit a_i. The important point is that it is always possible to select the intermediate carry c_i and sum s_i such that the summation in the second step does not generate a carry. Hence, the second step can also be executed in parallel for all the positions, yielding addition time, independent of the word length. Takagi et al. proposed a binary SD addition rule [16]. In the summation step, a carry is not generated because there aren't two cases



(a) Two levels CSA structure

(b) One level CSA structure

(c) Five-to-two CSA structure

(d) Four-to-two CSA structure

Fig. 1. Approach using carry save adder for the addition.

$c_i = s_i = 1$ and $c_i = s_i = -1$. For eliminating two cases the next lower digit generates a signal to deliver to the upper digit. For example, when $(x_i = 0, y_i = 1)$ or $(x_i = 1, y_i = 0)$, if x_{i-1} and y_{i-1} are both nonnegative values a carry can be generated. Therefore, $c_{i+1} = 1$ and $s_i = -1$. But the number of the input and output of the cell increases because the signal must be calculated and delivered. It is important for the adder cell to reduce the number of input and output because a number of gates in the cell lead to a slower speed.

3. Signed digit montgomery multiplier

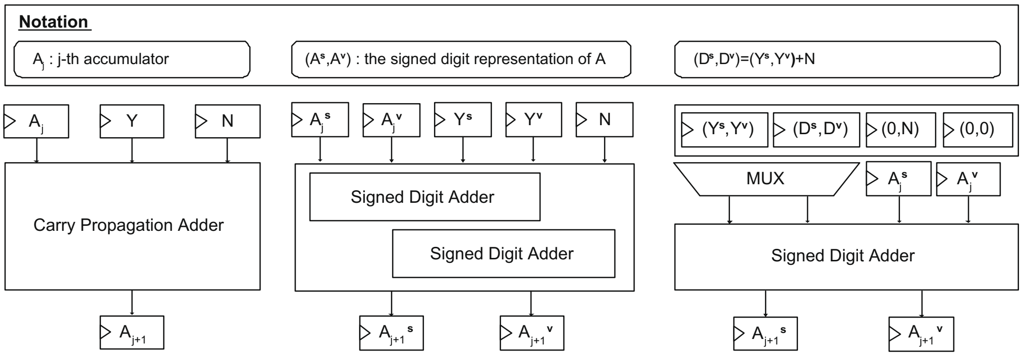
In this section, we design the SD Montgomery multiplier suitable for binary modular exponentiation using the signed digit number representation. The multiplier performs using SDA without carry propagation for the three input addition (step 4 of Algorithm 1). We can consider the method using multiplexer through precomputation like an approach using CSA in Fig. 1, and apply it to the multiplier as shown in Fig. 2b. One level signed digit adder (SDA) structure is similar to the four-to-two CSA structure of Fig. 1 in terms of total registers, the number of the input and output, and multiplexer. If one level SDA is more efficient than four-to-two CSA, a performance of the multiplier can be increased.

We propose an SD Montgomery multiplier. The input of the multiplier is two signed numbers X, Y, and a binary number N. The output is a signed number A. The multiplier consists of X_i_gen, MUX, u_i_gen, and an improved SDA (iSDA). The addition of Y is determined in accordance with x_i in Algorithm 1. We have to consider two cases both Y and $-Y$ because there is a case $x_i = -1$ in the SD number representation. We may need an extra operation and registers $-Y, -Y + N$. It may bring down a performance of the multiplier. To solve the above problem we propose X_i_gen. The input and output of our algorithms are represented in the SD number representation, but they are not negative numbers. That is, they can be converted to binary numbers. The operation of the X_i_gen is following. For example, let $X = (x_3, x_2, x_1, x_0) = (1, 0, -1, -1)$. When $x_0 = -1, X_0$ becomes 1 and the register which can be stored the carry becomes -1. When $x_1 = -1, X_1$ becomes 0 resulted in the addition with the carry and the register becomes -1. When $x_2 = 0, X_2$ becomes 1 and the register becomes -1 in the same way. Finally, when $x_3 = 1, X_3$ becomes 0 and the register becomes 0. Since we only need one bit of X for each iteration step of Montgomery multiplication, the X_i_gen is running in parallel and does not delay the algorithm. The architecture of X_i_gen is represented in Fig. 3. MUX can be implemented as two 4:1 multiplexers working in parallel, addressed by the two signals X_i and u_i. u_i_gen determines the addition of N and can be implemented as an XNOR gate. We deal with the proposed iSDA in 3.1.

The operation of the multiplier is following. The addition of Y and N is performed at the beginning of each multiplication and the result is stored at D. X_i_gen generates the signal X_i from the least significant digit to the most significant digit. The signal X_i is delivered to MUX. u_i is also generated from u_i_gen and delivered to MUX. The input of the iSDA consists of a signed number through MUX and an accumulator A. The iSDA outputs the result repeatedly. The multiplier outputs a signed number A after all loops. The SD Montgomery multiplier with the proposed iSDA is represented in Fig. 4.

3.1. Improved SDA

Almost all existing SDAs are constructed using just one method between Sign-and-Value and Plus-and-Minus encoding methods. The number of cases for the addition with two signed numbers is represented at each encoding as shown in Fig. 5. In Plus-and-Minus encoding, 0 has two expressions (1,1) and (0,0). Therefore, the



(a) Two levels SDA structure

(b) One level SDA structure

Fig. 2. Our approach using signed digit adder for the addition.

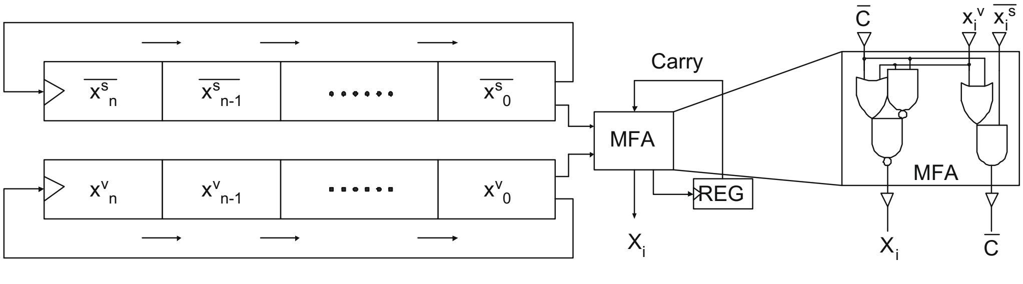


Fig. 3. The architecture of X_i_gen.

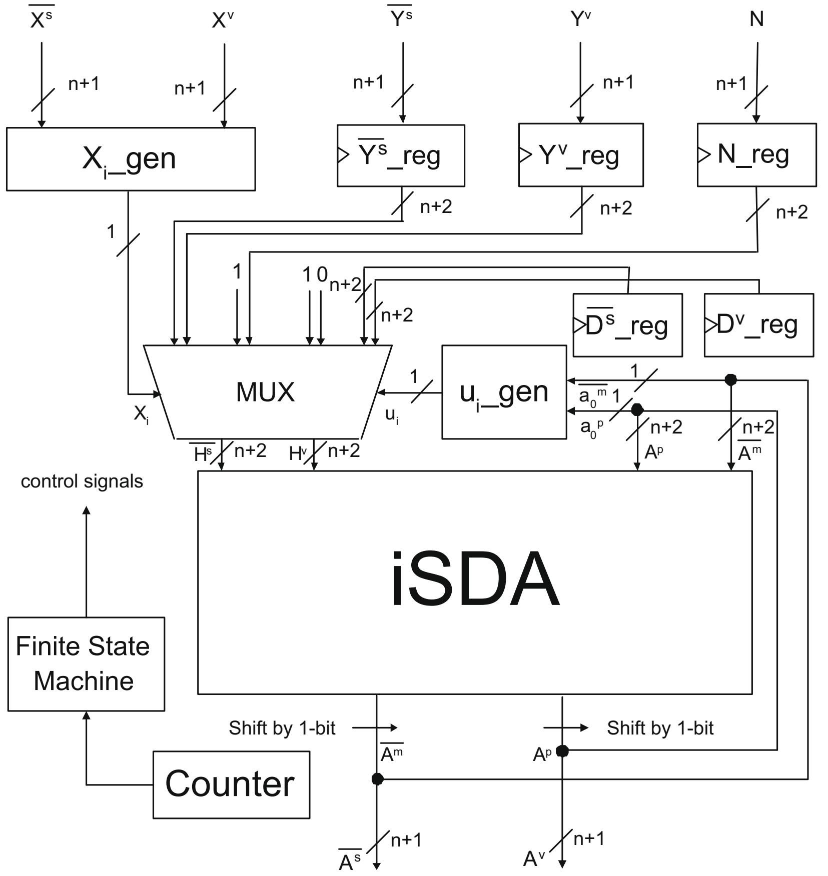
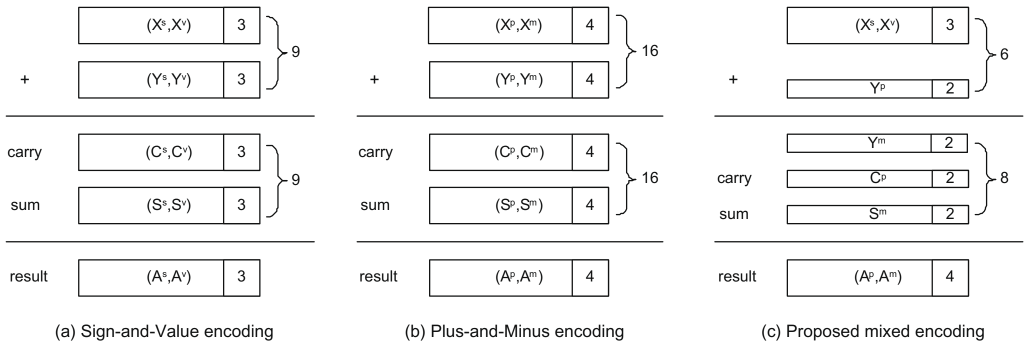


Fig. 4. The architecture of our SD Montgomery multiplier.



(a) Sign-and-Value encoding (b) Plus-and-Minus encoding (c) Proposed mixed encoding

Fig. 5. Number of cases for the addition with two signed numbers.

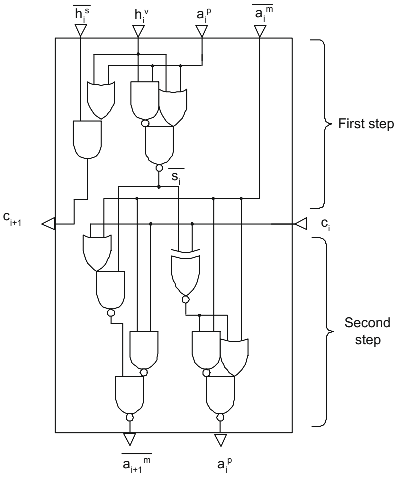
number of cases is increased, but the method has more gate expressions (see Fig. 6).

The number of cases for the addition is declined using the proposed mixed encoding as shown in Fig. 5c. For an improved SDA, we calculate the addition with two signed numbers using both Sign-and-Value and Plus-and-Minus representations. The input of the iSDA consists of H_j and A_j, and the output is A_{j+1}, where H_j and A_j are the input of the j-th loop. We use H as the input of the iSDA through multiplexer, which is one value among $\{Y+N, Y, N, 0\}$ and denote by $H = \sum_{i=0}^{n+2} h_i 2^i, h_i = (\overline{h_i^s}, h_i^v)$. A is an accumulator and denoted by $A = \sum_{i=0}^{n+2} a_i 2^i, a_i = (a_i^p, \overline{a_i^m})$. We use invert values as like $\overline{h_i^s}$ and $\overline{a_i^m}$ for efficient gate expression.

The iSDA operates with two steps. The first step is the addition with (h_i^s, h_i^v) and a_i^p. The binary carry is generated from an addition of a signed number and a binary number. We also use an invert value $\overline{s_i}$ for efficient gate expression. The first step is represented in step 2 Algorithm 2. The second step is the addition with a binary carry from lower digit, a negative result of the first step, and a_i^m. Therefore, a negative carry is generated. The output of the iSDA is $(a_i^p, \overline{a_i^m})$. The second step is represented in step 3 of Algorithm 2.

Algorithm 2. Improved Signed Digit Addition (iSDA)

```
Input: Sign-and-Value $X = (\overline{X^s}, X^v)$, Plus-and-Minus $Y = (Y^p, \overline{Y^m})$
Output: Plus-and-Minus $A = X + Y$
1. $A \leftarrow 0$
2. $c_{i+1} \leftarrow (h_i^v \vee a_i^p) \wedge \overline{h_i^s}, \overline{s_i} \leftarrow (h_i^v \wedge a_i^p) \vee \overline{(h_i^v \vee a_i^p)}$
3. $\overline{a_{i+1}^m} \leftarrow (\overline{a_i^m} \vee c_i) \wedge \overline{s_i} \vee (\overline{a_i^m} \wedge c_i), a_i^p$
   $\leftarrow \overline{\{(\overline{s_i} \oplus c_i) \vee a_i^m\}} \vee (\overline{s_i} \oplus c_i) \wedge a_i^m$
4. Return $A = (A^p, \overline{A^m})$
```



(a) The architecture of i-th iSDA cell

Input					Output	
h_i	a_i^p	$\overline{h_i^s}$	h_i^v	a_i^p	c_{i+1}	$\overline{s_i}$
0	0	1	0	0	0	1
	1	1	0	1	1	0
1	0	1	1	0	1	0
	1	1	1	1	1	1
-1	0	0	1	0	0	0
	1	0	1	1	0	1

(b) Logic of first step

$\overline{a_i^m}$	$\overline{s_i}$	c_i	$\overline{a_{i+1}^m}$	a_i^p
1	1	0	1	0
1	1	1	1	1
1	0	0	0	1
1	0	1	1	0
0	1	0	0	1
0	1	1	1	0
0	0	0	0	0
0	0	1	0	1

(c) Logic of second step

Fig. 6. The architecture of iSDA and logics of each step.

3.2. Efficient conversion

The iSDA outputs a Plus-and-Minus number A as the final result. But A needs format conversion to Sign-and-Value representation to use next multiplication. Therefore, the conversion may need an extra hardware. To solve the conversion problem, we use the iSDA as a converter. If we use the iSDA for the conversion, the extra hardware is not needed. The cases generated by converting Plus-and-Minus to Sign-and-Value representation for the final result of the multiplication in Fig. 7b. While, (a) of Fig. 7 shows cases generated by calculating the second step of the iSDA. In Fig. 7, we can recognize that (b) is identical to the right four columns of (a). If we use proper values, the iSDA can convert without any extra hardware.

We can use the iSDA as a converter easily. In step 2 of Algorithm 2, when $h_i^v = 1, c_{i+1} = \overline{h_i^s}$ and $s_i = a_i^p$. Let us change the input of the iSDA (c) into (d) of Fig. 7. The results for calculating the first step of the iSDA, c_{i+1}, s_i, are $c_{i+1} = A_{i+1}^p$ and $s_i = \overline{A_i^m}$. Therefore, Plus-and-Minus representation is converted easily to Sign-and-Value representation.

3.3. Modular exponentiation

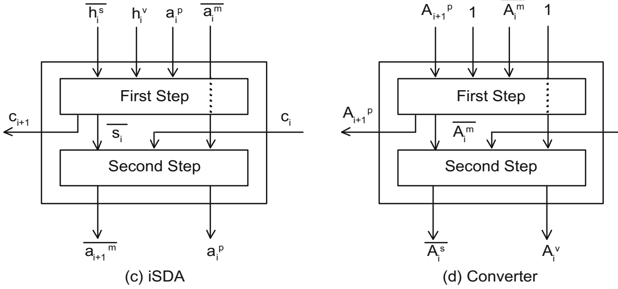
The SD Montgomery multiplication with the proposed iSDA is represented in Algorithm 3. For modular exponentiation, two modifications [6,7] are applied to the multiplication.

$\overline{a_i^m}$	$\overline{s_i}$	c_i	$\overline{a_{i+1}^m}$	a_i^p
1	1	0	1	0
1	1	1	1	1
1	0	0	0	1
1	0	1	1	0

(a) Cases of calculating second step

$\overline{A_i^m}$	A_i^p	$\overline{A_i^s}$	A_i^v
1	0	1	0
1	1	1	1
0	0	0	1
0	1	1	0

(b) Cases of format conversion



(c) iSDA

(d) Converter

Fig. 7. Efficient format conversion.

```
Algorithm 3. SD Montgomery Multiplication (SDMM)

Input: $X = (00x_{n-1} \cdots x_0), Y = (0y_n \cdots y_0 0), N = (n_{n-1} \cdots n_0)$
Output: $A = XYR^{-1}$ mod $N$, where $R = 2^{n+3}$ mod $N$
1. $A \leftarrow 0, D \leftarrow 0$
2. $(D^p, \overline{D^m}) \leftarrow iSDA(\overline{Y^s}, Y^v, N, 0)$
3. $(\overline{D^s}, D^v) \leftarrow iSDA(D^p, 1, \overline{D^m}, 1)$
4. For $i$ from 0 to $n+2$ do:
5. $u_i \leftarrow (a_0^p + \overline{a_0^m})$ mod 2
6.     If $X_i = 0$ and $u_i = 0$ then $(A_{i+1}^p, \overline{A_{i+1}^m}) \leftarrow iSDA(1, 0, A_i^p, \overline{A_i^m})/2$
7.     Else If $X_i = 1$ and $u_i = 0$ then
         $(A_{i+1}^p, \overline{A_{i+1}^m}) \leftarrow iSDA(\overline{Y_i^s}, Y_i^v, A_j^p, \overline{A_j^m})/2.$
8.     Else If $X_i = 0$ and $u_i = 1$ then
         $(A_{i+1}^p, \overline{A_{j+1}^m}) \leftarrow iSDA(1, N, A_i^p, \overline{A_i^m})/2$
9.     Else $(\overline{A_{i+1}^p, A_{i+1}^m}) \leftarrow iSDA(\overline{D_i^s}, D_i^v, A_i^p, \overline{A_i^m})/2$
10. $(\overline{A^s}, A^v) \leftarrow iSDA(A^p, 1, \overline{A^m}, 1)$
11. Return $A = (\overline{A^s}, A^v)$
```

The RSA encryption and decryption functions are given by $C = M^e(mod\ N)$ and $M = C^d(mod\ N)$ respectively, where M is a plaintext message block, C is a ciphertext block, N is the n-bit modulus, and e and d are the public and private exponents respectively. The equation $ed = 1(mod\ (p-1) \times (q-1))$ must also hold, where p and q are two large prime numbers ($n/2$-bits in length) and $N = pq$. Modular exponentiation with the SD Montgomery multiplication is represented in Algorithm 4.

```
Algorithm 4. Modular exponentiation($M, e, N$)

Input: message $M$, encryption key $e$, modulus $N$
Output: $C = M^e\ mod\ N$
1. $K = 2^{2n}\ mod\ N$. (computed externally)
2. $(\overline{J_0^s}, J_0^v) \leftarrow SDMM(\overline{K}, 0, \overline{C}, 0, N)$
3. $(\overline{T_0^s}, T_0^v) \leftarrow SDMM(\overline{K}, 0, 0, 0, N)$
4. For $i$ from 0 to $n_e - 1$ do: ($n_e$ is the bit lengths of the public
   exponent)
5.     $(\overline{J_{i+1}^s}, J_{i+1}^v) \leftarrow SDMM(\overline{J_i^s}, J_i^v, \overline{J_i^s}, J_i^v, N)$
6.     If $e_i = 1$ then $(T_{i+1}^s, T_{i+1}^v) \leftarrow SDMM(\overline{T_i^s}, T_i^v, \overline{J_i^s}, J_i^v, N)$
7. $(\overline{C^s}, C^v) \leftarrow SDMM(0, 0, \overline{T^s}, T^v, N)$
8. $C \leftarrow$ Convert$(\overline{C^s}, C^v)$
9. Return $C$.
```

Table 1
Comparison of other adders and iSDA.

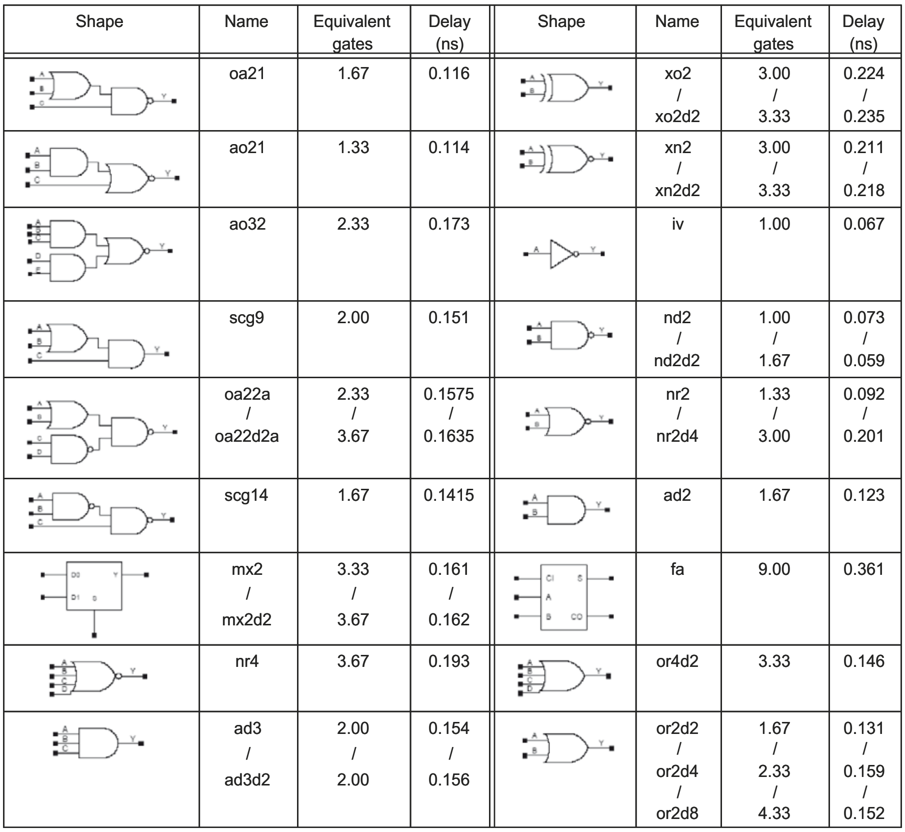
	Total delay(ns)	Components	Equivalent gates	Components
Four-to-two CSA [9]	0.7220	2FAs	18.00	2FAs
SDA [13]	0.8025	1nr2d4 + 2iv +1nr4 + 1ad2 +1ad3d2	39.34	1or2d4 + 2ad3 + 1nr2d4 +1or2d8 + 2ad3d2 + 1or4d2 +1nr4 + 2iv + 4ad2
SDA [17]	0.8960	1or2d2 + 1iv +1xo2d2 + 2xn2d2 +2ad2	22.34	2or2d2 +1nr2 + 2iv +1xn2d2 + 4ad2 + 1nd2 +1xo2d2 + 1ao21
SDA [20]	0.6930	2xo2d2 + 1xo2	20.66	3xo2d2 + 1xo2 +1mx2d2 +1mx2
SDA [14]	0.6815	2nd2 + 2xo2d2 +1iv	18.33	6nd2 + 2xo2d2 + 1mx2d2 +2iv
SDA [19]	0.6270	1xo2d2 + 1xn2d2 +1nd2d2 + 1oa21	15.66	1nr2 + 1nd2 + 1xo2d2 +1ao32 + 1xn2d2 + 1nd2d2 +1iv +1oa21
iSDA	**0.5390**	**1oa22d2a + 1oa22a** **+1xn2d2**	**14.67**	**1scg9 + 1oa22d2a** **+1oa21 + 1xn2d2** **+1scg14 + 1oa22a**

Table 2
Comparison of multipliers.

Adders	Clock cycle	Critical path delay	Area complexity
Four-to-two CSA	$n+4$	2FA + 4:1 MUX +1xo2	9Register + 2*4:1 MUX +1xo2 + 2FA
Existing SDAs	$n+4$	1SDA + 4:1 MUX +1xo2	9Register + 2*4:1 MUX +1xo2 + 1RBA
iSDA	**n + 6**	**1iSDA + 4:1 MUX** **+1xn2**	**9Register + 2*4:1 MUX** **+1xn2 + 1iSDA**

4. Comparison

For comparison, we use the technology "SAMSUNG STD 130 0.18 μm 1.8 V CMOS Standard Cell Library". The comparison of adders is presented in Table 1. Table 1 consists of total delay(ns), components of the delay step, equivalent gates. First, we deal with adders in terms of time complexity. Table 1 shows that the iSDA is more than about 25.3% faster than the four-to-two CSA [9]. The iSDA is also more than between 39.8% (SDA [17]) and 14% (SDA



Shape	Name	Equivalent gates	Delay (ns)	Shape	Name	Equivalent gates	Delay (ns)
	oa21	1.67	0.116		xo2 / xo2d2	3.00 / 3.33	0.224 / 0.235
	ao21	1.33	0.114		xn2 / xn2d2	3.00 / 3.33	0.211 / 0.218
	ao32	2.33	0.173		iv	1.00	0.067
	scg9	2.00	0.151		nd2 / nd2d2	1.00 / 1.67	0.073 / 0.059
	oa22a / oa22d2a	2.33 / 3.67	0.1575 / 0.1635		nr2 / nr2d4	1.33 / 3.00	0.092 / 0.201
	scg14	1.67	0.1415		ad2	1.67	0.123
	mx2 / mx2d2	3.33 / 3.67	0.161 / 0.162		fa	9.00	0.361
	nr4	3.67	0.193		or4d2	3.33	0.146
	ad3 / ad3d2	2.00 / 2.00	0.154 / 0.156		or2d2 / or2d4 / or2d8	1.67 / 2.33 / 4.33	0.131 / 0.159 / 0.152

Fig. 8. Cell representations.

[19]) faster than existing SDAs. Second, in terms of area complexity, the iSDA is the most efficient in compete adders. The iSDA is more than about 18.5% smaller than the four-to-two CSA. The iSDA is also more than between 62.7% (SDA [13]) and 6.3% (SDA [19]) smaller than existing SDAs. Refer to the Appendix for details about cell representations used in tables.

The comparison of multipliers is presented in Table 2. Table 2 consists of used adder, clock cycle, critical path delay, and area complexity. Table 2 shows that multipliers are similar in terms of critical path delay and area complexity except for the adder. The SD Montgomery multiplier with the iSDA is the most efficient multiplier compared the four-to-two CSA and existing SDAs.

However, we need to consider clock cycles. While the multipliers with the four-to-two CSA and existing SDAs have $n+4$ clock cycles, the SD Montgomery multiplier has $n+6$ clock cycles. Two clock cycles are used in step 3 and 10 in Algorithm 3 for converting Plus-and-Minus representation to Sign-and-Value representation. While 2 clock cycles are used in step 3 and 10 in Algorithm 3, these 2 clock cycles are negligible for 1024 or 2048 bits exponentiation because of the shortest unit time of multiplication. Therefore, the SD Montgomery multiplier is more efficient in large bits environment.

5. Conclusion

In this paper, we design an efficient SD Montgomery multiplier and circuit architecture suitable for modular exponentiation using the SD number representation. The addition stages are performed using the iSDA, which is the most efficient in compared adders. We implement the multipliers with the iSDA and other adders using SAMSUNG STD 130 0.18 μm 1.8 V CMOS Standard Cell Library. The proposed modular multiplier can be applied to public key cryptosystems based on integer arithmetic such as RSA, DSA or ECC.

Appendix A

In this appendix, we present cell representations, which are used in Tables 1 and 2. Fig. 8 shows shape, name, equivalent gates, and delay for cells. Though some cells have the same shape, they have a different name, equivalent gates and delay according to the number of fan-out. For example, The cell ‘‘nd2d2” means that 2-inputs NAND gate with 2 fan-outs. In another example, The cell ‘‘ad3” means that 3-inputs AND gate with 1 fan-out.

References

[1] Ronald L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and publickey cryptosystems, in: Communications of the ACM, vol. 21, 1978, pp. 120–126.

[2] G.R. Blakley, A computer algorithm for the product ab modulo m, IEEE Trans. Comput. C-32 (1983) 497–500.

[3] E.F. Brickell, A fast modular multiplication algorithm with application to two-key cryptography, in: Proceedings of the CRYPTO'82 Adcances Cryptology, 1982, pp. 51–60.

[4] P.L. Montgomery, Modular multiplication without trial division, Math. Comput. 44 (1985) 519–521.

[5] C.K. Koc, T. Acar, S. Burton, S. Kaliski Jr., Analyzing and comparing montgomery multiplication algorithms, IEEE Micro. 16 (3) (1996) 26–33.

[6] C.D. Walter, Montgomery exponentiation needs no final subtractions, Electronic Lett. 35 (21) (1999) 1831–1832.

[7] K. Manochehri, S. Pourmozafari, Modified radix-2 Montgomery modular multiplication to make it faster and simpler, in: ITCC 2005, 2004, pp. 598–602.

[8] T.W. Kwon, C.S. You, W.S. Heo, Y.K. Kang, J.R. Choi, Two implementation methods of a 1024-bit RSA cryptoprocessor based on modified montgomery algorithm, in: IEEE Interantional Symposium On Circuits and Systems (ISCAS), vol. 4, 2001, pp. 650–653.

[9] Ciaran McIvor, Maire McLoone, John V. McCanny, Alan Daly, Fast Montgomery modular multiplication and RSA cryptographic processor architectures, in: ACCSC 2003, 2003, pp. 379–384.

[10] A. Cilardo, A. Mazzeo, L. Romano, G.P. Saggese, Carry-save montgomery modular exponentiation on reconfigurable hardware, IEEE Proc. On DATE'04 03 (3) (2004) 206–211.

[11] A. Avizienis, Signed-digit number representations for fast parallel arithmetic, IRE Trans. Electron. Comput. EC-IO (9) (1961) 389–400.

[12] S. Kuninobu, T. Nishiyama, T. Taniguchi, High speed MOS multiplier and divider using redundant binary representation and their implementation in a microprocessor, IEICE Trans. Electron E76-C (March) (1993) 436–445.

[13] Anders Lindstrom, Michael Nordseth, Lars Bengtsson, 0.13 m CMOS Synthesis of Common Arithmetic Units, Technical Report No. 03–11, Department of Computer Engineering, Chalmers University of Technology, August 2003.

[14] H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara, K. Mashiko, An 8.8-ns 54 × 54 bit multiplier with high speed redundant binary architecture, IEEE J. Solid-State Circuit 31 (6) (1996) 773–783.

[15] D.S. Phatak, I. Koren, Hybrid signed-digit number systems: a unified framework for redundant number representations with bounded carry propagation chains, IEEE Trans. Comput. 43 (8) (1994) 880–891.

[16] Naofumi Takagi et al., High-speed VLSI multiplication algorithm with a redundant binary additon tree, IEEE Trans. Comput. C-34 (9) (1985) 789–796.

[17] M. Tonomura, High-speed digital circuit of discrete cosine transform, SP94-11, DSP94-66, Tech. Rep. IEICE Japan, 1994, pp. 39-46.

[18] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi, N. Takagi, Design of high speed MOS multiplier and divider using redundant binary representation, in: IEEE Proceedings Of the 8th Symposium on Computer Arithmetic (ARITH8), 1987, pp. 80–86.

[19] H. Edamatsu, T. Taniguchi, T. Nishiyaina, S. Kuninobu, A 33 MFLOPS floating point processor using redundant binary representation, Dig. Tech. Papers of 1988 ISSCC, 1988, pp. 152–153.

[20] J.W. Hong, Real/Complex Number Multiplier Using Redundant Binary Arithmetic, Dept. Elec. and Comput. Engin., Grad. School of Yonsei Univ., Master Thesis.

Daesung Lim received the B.S. degree in Mathematics from Kangnam University, Soowon, Korea, in 2004 and the M.S. degree of engineering in Information Security from Korea University, Seoul, Korea in 2006, respectively. His research interests include fast implementation of public key cryptosystems.

Nam Su Chang received the B.S. degree in Mathematics from University of Seoul, Seoul, Korea, in 2002 and the M.S. degree of engineering in Information Security from Korea University, Seoul, Korea in 2004, respectively. He is currently an doctor degree student at Graduate School of Information Management and Security, Korea University, and also a researcher of Center for Information Security Technologies (CIST). His research interests include algorithms and architectures for computations in Galois fields and fast implementation of public key cryptosystems.

Sung Yeon Ji received the B.S. degree in mathematics from Hanshin University, Osan, Korea, in 2005 and the M.S. degree of engineering in Information Security from Korea University, Seoul, Korea in 2007, respectively. His research interests include fast implementation of public key cryptosystems.

Chang Han Kim received the B.S. and M.S. degree in Mathematics from Korea University of Korea in 1985 and 1987, respectively, and the Ph.D. degree in Mathematics from Korea University of Korea in 1992. He is now a professor with the school of Information and Technology Systems at Semyung University, Jecheon, Korea. His research interests include algorithms and architectures for computations in Galois fields and fast implementation of public key cryptosystems.

Sangjin Lee received the B.S. and M.S. degree in Mathematics from Korea University of Korea in 1987 and 1989, respectively, and the Ph.D. degree in Mathematics from Korea University of Korea in 1994. He is now a professor at Graduate School of Information Management and Security, Korea University, Seoul, Korea. His research interests include design and analysis of block cipher and computer forensics.

Young-Ho Park received the B.S. and M.S. degree in Mathematics from Korea University of Korea in 1990 and 1993, respectively, and the Ph.D. degree in Mathematics from Korea University of Korea in 1997. He is currently an associate professor at Sejong Cyber University, Korea. His research interests include number theory, cryptography, information security and side channel attacks.