情報保護學會論文誌
第17卷 第1號, 2007. 2

고속 RSA 하드웨어 곱셈 연산과 하드웨어 구조*

장남수[1†], 임대성[1], 지성연[1], 김창한[2‡], 윤석봉[3]
[1]고려대학교, [2]세명대학교, [3]동의대학교

Fast RSA Montgomery Multiplier and Its Hardware Architecture

Nam Su Chang[1†], Daesung Lim[1], Sung Yeon Ji[1], Chang Han Kim[2‡], Suk Bong Yoon[3]
[1]Korea University, [2]Semyung University, [3]Dongeui University

요 약

몽고메리 곱셈 방법을 이용한 고속 연산은 RSA 암호 시스템의 설계에 중요한 부분을 차지한다. 몽고메리 곱셈은 두 번의 덧셈 연산으로 구성되며 CSA를 이용한 방법과 RBA를 이용한 방법이 있다. CSA의 경우 4-2 CSA 또는 5-2 CSA를 이용하여 구현하며, RBA의 경우 기존 이진 방법과 달리 잉여 이진체계를 이용한다는 특징을 가진다. [1] 에서는 기존의 RBA와 다른 새로운 이진 체계와 하드웨어 구조를 제안하고 몽고메리 곱셈에 적용하였다. 본 논문에서는 [1] 에서 제안한 RBA의 로직 구조를 재구성하여 시간 복잡도 뿐만 아니라 결합기가 필요하지 않도록 구성하여 공간 복잡도를 크게 줄였다. 또한 입·출력 값을 변형시켜 지수승 연산에 적합하도록 설계하였다. 그 결과 제안하는 RBA는 삼성 STD130 0.18μm 1.8V 표준 셀 라이브러리에서 지원하는 게이트들을 사용하여 설계하는 환경에서, 기존의 4-2 CSA 보다 공간과 시간 복잡도를 각각 18.5%와 25.24%를, 기존의 RBA 보다 6.3%와 14%를 감소시킨다. 또한 [1] 의 RBA와 비교시 44.3%, 2.8%의 감소된 복잡도를 갖는다.

ABSTRACT

A fast Montgomery multiplication occupies important to the design of RSA cryptosystem. Montgomery multiplication consists of two addition, which calculates using CSA or RBA. In terms of CSA, the multiplier is implemented using 4-2 CSA or 5-2 CSA. In terms of RBA, the multiplier is designed based on redundant binary system. In [1], A new redundant binary adder that performs the addition between two binary signed-digit numbers and apply to Montgomery multiplier was proposed. In this paper, we reconstruct the logic structure of the RBA in [1] for reducing time and space complexity. Especially, the proposed RB multiplier has no coupler like the RBA in [1]. And the proposed RB multiplier is suited to binary exponentiation as modified input and output forms. We simulate to the proposed NRBA using gates provided from SAMSUNG STD130 0.18μm 1.8V CMOS Standard Cell Library. The result is smaller by 18.5%, 6.3% and faster by 25.24%, 14% than 4-2 CSA, existing RBA, respectively. And Especially, the result is smaller by 44.3% and faster by 2.8% than the RBA in [1].

Keywords : *Montgomery multiplication, Redundant binary adder, Signed-digit system*

접수일: 2006년 9월 26일; 채택일: 2006년 11월 7일
* "본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (IITA-2006-(C1090-0603-0025))
* 본 논문은 제18회 정보보호와 암호에 관한 학술대회(WISC2006) 최우수 논문임.
† 주저자, nschang@cist.korea.ac.kr
‡ 교신저자, chkim@semyung.ac.kr

Ⅰ. 서 론

RSA 암호 시스템은 미국 M.I.T. 대학의 Rivest, Shamir, Adleman에 의해서 1978년에 고안된 암호 시스템으로 Diffie, Hellman이 제안한 공개키 암호 시스템에 대한 개념을 가장 충실히 반영한 것으로 알려져 있다. RSA 암호 시스템은 매우 큰 정수에 대한 소인수분해가 어렵고 해를 찾아내는데 많은 시간을 요구한다는 점을 이용한 암호 시스템이다. 일반적으로 RSA 암호 시스템은 시스템의 안전성을 높이기 위해 1,024비트 이상의 큰 수를 기반으로 한 모듈러 지수승 연산을 수행한다. 모듈러 지수승 연산은 반복된 모듈러 곱셈으로 구성되어 있으므로 효율적인 모듈러 곱셈 알고리즘의 하드웨어 구현은 암호 시스템의 효율성을 크게 향상시킨다.

본 논문에서는 암호 시스템의 하드웨어 구현에 적합한 모듈러 곱셈 알고리즘을 중심으로 논한다. 모듈러 곱셈 알고리즘 중 몽고메리 곱셈 알고리즘은 모듈러 연산을 반복적으로 수행하는 RSA 암호 시스템 구현에 널리 사용되고 있다. 몽고메리 곱셈 알고리즘에서 연산 속도를 결정짓는 클럭 주기에 가장 큰 영향을 미치는 세 피연산자의 덧셈을 효율적으로 수행하기 위해 캐리 저장 덧셈기(Carry Save Adder, CSA)와 잉여 이진 덧셈기(Redundant Binary Adder, RBA)가 제시되었다. CSA를 이용한 방법은 다음 곱셈의 입력 값으로 사용하기 위해 출력 결과 캐리 C와 합 S를 (C,S)형태로 보존한다. 이 경우 캐리를 저장할 레지스터가 추가로 필요하지만 덧셈을 수행하지 않으므로 고속 연산에 적합하다. McIvor는[3]에서 X,Y가 모두 (C,S)형태이므로 5개의 입력 값과 2개의 출력 값을 갖는 5-2 CSA 구조와 Y+M을 사전 계산하여 4개의 입력 값과 2개의 출력 값을 갖는 4-2 CSA 구조를 제시하였다. 5-2 CSA 구조는 FA 3단계로 구성되고 4-2 CSA 구조는 FA 2단계와 FA 1단계보다 시간 복잡도가 낮은 MUX로 구성되므로 곱셈기에 적용할 경우 전체 시간 복잡도는 4-2 CSA 구조를 적용한 방법이 더 적합하다[3].

세 피연산자의 덧셈을 효율적으로 수행하는 또 다른 방법인 RBA는 부호 자리 수 체계(Signed Digit Number System, SD)에서 설계된다[4]. SD 수 체계에서는 덧셈 수행 시 캐리 전파를 제거함으로써 효율적인 RBA 설계가 가능하다. 그러나 SD 수를 표현하기 위해서는 부호 비트와 값 비트를 표시해야 하므로 한 자리를 표현하는 데 두 비트가 사용된다. 따라서 레지스터의 사용량이 증가하나(C,S)형태의 구조와 동일한 레지스터를 사용하고 RBA와 4-2 CSA는 입력 값의 개수와 출력 값의 개수가 동일하므로 대체할 수 있다. 따라서 기존의 RBA를 분석하고 보다 효율적인 게이트 구성으로 RBA를 설계하고 곱셈기에 적용하면 전체 복잡도를 낮출 수 있다.[1]에서는 Takagi의 이진 SD 덧셈 규칙[5]을 적용한 기존의 RBA에 비해 효율성이 높은 RBA를 설계하기 위해 새로운 덧셈 규칙을 제안하고 덧셈기에 적용하였다. 또한 삼성 STD130 0.18μm 1.8V CMOS 셀 라이브러리[6]에서 제공하는 게이트를 사용하여 RBA를 설계하고 몽고메리 곱셈기에 적용하였다.[1]에서는 곱셈의 결과를 다음 곱셈의 입력 값으로 사용하기 위해 결합기를 사용해야 한다. 따라서 하드웨어 구조로 설계할 경우 공간 복잡도가 커질 수밖에 없다. 본 논문에서는[1]에서 제안한 덧셈 규칙을 적용하여 효율적인 RBA 로직구조를 재구성한다. 또한 제안한 RBA는 주연산인 덧셈뿐만 아니라 결합기의 역할을 수행할 수 있도록 설계한다. 따라서[1]에서 공간 복잡도가 늘어나는 문제를 해결할 수 있다. 그 결과 4-2 CSA에 비해 약 18.5%의 공간 복잡도 감소와 약 25.3%의 시간 복잡도 감소를 보였다. 또한 기존의 RBA에 비해 최대 약 39.8%, 최소 약 14%의 시간 복잡도 감소를 보였다. 공간 복잡도 또한 기존의 RBA에 비해 최대 약 62.7%, 최소 약 6.3%의 감소를 보였다. 또한[1]의 RBA와 비교시 공간 복잡도는 44.3%, 시간 복잡도는 2.8%의 감소한다. 비록 제안한 RBA를 몽고메리 곱셈기에 적용할 경우 4-2 CSA 몽고메리 곱셈기와 RBA 몽고메리 곱셈기에 비해 클럭 사이클 수는 2회 더 증가하나 이는 1,024비트 또는 2,048비트 연산에서 비중이 크지 않다. 또한 동일한 레지스터 수와 MUX 및 게이트를 사용하며 덧셈기의 크기는 작고 단위 시간은 짧으므로 반복된 곱셈으로 이루어진 RSA 암호 시스템과 같은 정수 기반 암호 시스템에서 효율성을 크게 발휘할 수 있고 하드웨어 칩으로 구현할 경우 다양한 환경에서 사용될 수 있을 것이다.

2장에서는 이진 몽고메리 곱셈 알고리즘과 복잡도를 줄일 수 있는 방법에 대해 설명하고 3장에서는 캐리 전파를 제거할 수 있는 방법인 CSA와 RBA를 분석하고 이를 적용한 곱셈기를 설명한다. 4장에서는 두 이진 SD 수의 덧셈을 수행하는 새로운 로직구조로 재구성한 RBA를 제안한다. 또한 결합기의 역할 수행에 대해 설명하고 몽고메리 곱셈에 적용하며 RSA 암호 시스템을 설계한다. 5장에서는 기존의 결과 비교하고 결론을 내

린다.

Ⅱ. Montgomery 곱셈 구조

M을 n-비트 모듈러스라 하자. 주어진 M보다 작은 정수 U, V에 대하여 몽고메리 곱셈을 수행하기 위해서 $R=2^n$을 이용하여 식 (1)을 통해 몽고메리 도메인의 원소인 X, Y로 변환시킨다.

$$X=U\times R(\mathrm{mod}M),\quad Y=V\times R(\mathrm{mod}M) \tag{1}$$

R^{-1}을 모듈러 M에 대하여 R의 역원이라 할 때, X, Y의 몽고메리 곱셈은 식 (2)와 같이 정의한다.

$$Mont(X,Y)=X\times Y\times R^{-1}=U\times V\times R(\mathrm{mod}M) \tag{2}$$

곱셈을 단지 한 번 수행한다고 할 때 이 과정에서 소요되는 복잡도는 일반적인 곱셈에 비해 크다. 그러나 몽고메리 곱셈은 RSA 암호 시스템에서의 지수승 연산과 같이 반복적인 모듈러 곱셈을 요구되는 환경에서 효율적이다. 지수승 연산에 적합하도록 몽고메리 곱셈 알고리즘을 보다 효율적으로 구성할 수 있다. 몽고메리 곱셈 알고리즘의 단계 3에서 수행되는 비교 연산과 뺄셈은 큰 연산량을 차지하게 되므로 X와 Y의 범위를 기존의 $0\le X, Y<M$에서 $0\le X, Y<2M$으로 대체함으로써 제거될 수 있다[7]. 이는 입력 값과 출력 값의 크기를 동일하게 하므로 출력 값을 다음 단계의 입력 값하여 반복적인 곱셈 연산을 수행할 수 있도록 한다. 따라서 X와 Y는 한 비트 늘어난 $n+1$비트가 되고, 반복 연산이 증가한다. 중간 결과 A의 크기를 $0\le A<2M$로 만들기 위해 X의 최상위 비트 x_{n+1}을 0이라 하고 반복 연산을 한 번 더 수행한다. 반복 연산이 두 번 더 수행되지만 곱셈기의 구성상 기존 방법 보다 효율적이다[7]. 또한 Y를 한 비트 늘려 최하위 비트를 0으로 만들어 반복 연산을 한 번 더 수행함으로써 몽고메리 곱셈 알고리즘의 단계 2.1의 u_i를 계산은 간단히 수행될 수 있다[8].

Ⅲ. 기존의 곱셈기

알고리즘 1 : 변형된 몽고메리 곱셈(X, Y, M)

입력 : $X=(00x_{n-1}\cdots x_1x_0)_2$, $Y=(0y_{n-1}\cdots y_1y_00)_2$,
$M=(m_{n-1}\cdots m_1m_0)_2$, $R=2^{n+3}$,
$\gcd(M,2)=1$, $0\le X, Y<2M$.

출력 : $A=XYR^{-1}\mathrm{mod}M=(a_{n-1}\cdots a_1a_0)_2$.

1. $A\leftarrow 0$.
2. For i from 0 to $n+2$ do :
   2.1. $u_i\leftarrow a_0 \mathrm{mod}2$.
   2.2. $A\leftarrow (A+x_iY+u_iM)/2$.
3. Return(A).

알고리즘 1의 임계 경로 지연 시간은 세 피연산자의 계산에서 발생하는 A에서 세 연산자의 덧셈과정에서 발생하는 캐리 전파를 처리하는 과정이 알고리즘 지연 시간의 대부분을 차지한다.

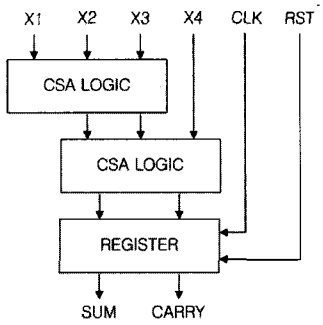
$$A\leftarrow (A+x_iY+u_iM)/2. \tag{3}$$

식 (3)과 같이 알고리즘 1에서 단계 2.2의 세 입력 값에 대한 덧셈을 수행하는 과정이 알고리즘 1의 시간 복잡도에서 가장 큰 부분을 차지한다. 그러나 이 문제는 CSA 또는 RBA를 사용하여 효율적인 계산을 수행할 수 있다.

3.1 CS 곱셈기

첫 번째 해결 방법은 세 개의 입력 값에 대하여 덧셈을 수행하고 두 개의 값(캐리: c_{i+1}, 합: s_i)을 출력하는 덧셈기인 CSA를 이용한 방법이다[3]. 알고리즘 1의 단계 2.2에서 $(A+x_iY+u_iM)$의 계산 결과는 CSA를 이용하여 캐리 A_C와 합 A_S으로 나타낼 수 있고, $A=(A_C+A_S)$는 다음 곱셈의 입력으로 사용되므로 Y 또한 $Y=(Y_C+Y_S)$으로 표현 한다. 따라서 $(A+x_iY+u_iM)$의 연산은 x_i를 X의 i번째 비트라 할 때 $((A_C+A_S)+x_i(Y_C+Y_S)+u_iM)$으로 나타낼 수 있으므로 다섯 개의 입력 값과 두 개의 출력 값을 갖는 McIvor의 1 방법[3]이 된다.

$((A_C+A_S)+x_i(Y_C+Y_S)+u_iM)$의 연산에서 x_i와 u_i의 값에 따라 덧셈에 필요한 피 연산자의 조합은 달라진다. $x_i=u_i=0$인 경우 단지 (A_C+A_S)의 연산이 되고 $x_i=1, u_i=0$인 경우 $(A_C+A_S)+(Y_C+Y_S)$의 연산이 되며 $x_i=0, u_i=1$의 경우 $(A_C+A_S)+M$의 연산이 된다. 하지만 $x_i=y_i=1$의 경우 다섯 개의 피 연산자로 구성 된 $(A_C+A_S)+(Y_C+Y_S)+M$의 연산을 수행하게 된다. 피 연산자의 항의 개수가 증가하는 문제점을 해결하기 위하여 사전 연산을 통해 [그림 1]과 같이 항의 개수를 줄인 McIvor의 2 방법이 제안되었다. 곱셈이 시



〔그림 1〕 McIvor의 2 방법

작하기 전 사전 연산을 통하여$(Y_C+Y_S)+M$의 연산을 수행하고 그 결과 캐리 D_C와 합 D_S를 저장한 후 *MUX*를 이용하여 x_i와 u_i의 경우에 선택적으로 사용하는 방법이다.

3.2 RB 곱셈기

암호 시스템 구성에 적합한 4-2 CSA 몽고메리 곱셈은 CSA가 주된 연산을 이룬다. 4-2 CSA의 입력 값은 (A1,A2)와 (D1,D2)이다. A=(A1+A2)이고 D=(D1+D2)이므로 캐리 전파 없이 덧셈 수행이 가능하다면 입력 값은 두 개가 될 것이다.

캐리 전파 문제를 해결하는 방법에는 이진 수 체계(Binary Number System)가 아닌 잉여 이진 수 체계(Redundant Binary Number System)로 확장하여 부호 비트를 표현하는 방식이 있다. 간단한 예를 들면 두 비트 x_i, y_i에 대하여 덧셈을 수행할 때 경우의 수는 〔표 1〕과 같다.

x_i와 합 y_i이 모두 1인 경우 캐리가 발생할 수 있다. 따라서 〔표 1〕과 같이 수행하면 캐리 전파를 제거할 수 있다. 그러나 $\bar{1}$을 나타내기 위해서는 부호 비트를 사용해야 한다. RBA는 입력 값이 두 개이고 출력 값이 한 개이지만 부호 비트를 포함하므로 4-2 CSA와 동일한 형태를 갖추지만 내부 구조를 간단히 하여 효율성을 높

〔표 1〕 부호화된 두 이진수의 합

x_i	y_i	c_{i+1}	s_i
0	0	0	0
0	1	1	$\bar{1}$
1	0	1	$\bar{1}$
1	1	1	0

〔표 2〕 이진 SD 덧셈 규칙

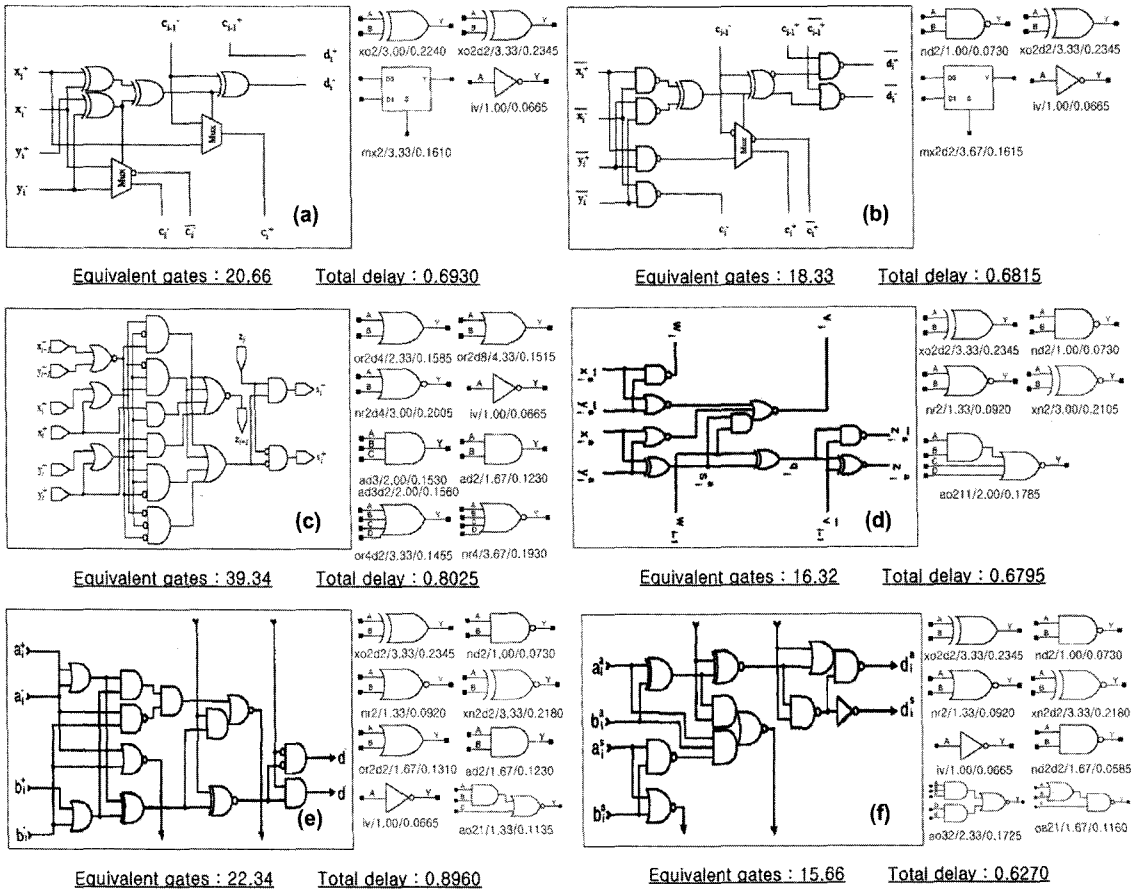
타입	x_i	y_i	(x_{i-1}, y_{i-1})	c_{i+1}	s_i
1	1	1	-	1	0
2	1	0	둘 다 음수가 아닐 경우	1	$\bar{1}$
	0	1	그렇지 않으면	0	1
3	0	0	-	0	0
4	1	$\bar{1}$	-		
	$\bar{1}$	1			
5	0	$\bar{1}$	둘 다 음수가 아닐 경우	0	$\bar{1}$
	$\bar{1}$	0	그렇지 않으면	$\bar{1}$	1
6	$\bar{1}$	$\bar{1}$	-	$\bar{1}$	0

일 수 있다는 장점을 가진다.

SD 수 체계와 일반적인 수 체계는 각 자리 수에 따른 표현 방식에 차이가 있다. r진법에서 일반적인 수 체계는 {0,1,2,...,r-1}의 r개의 원소로 표현되는 반면, SD 수 체계는 {-(r-1),-(r-2),....,0,..,r-2,r-1}을 이용한다. SD 수 체계에서 각 자리들이 양의 값뿐만 아니라 음의 값도 가지게 함으로써 SD 수 체계는 하나의 수가 서로 다르게 표현될 수 있는 잉여한 특성을 가지게 된다. 일반적인 수 체계에서는 음수를 표현하기 위해 수의 부호와 크기를 구분해 표현하는 부호-크기 표현 방식, 1의 보수 표현 방식, 또는 2의 보수 표현 방식 등을 사용하나 SD 수 체계는 각각의 자리가 부호에 대한 정보를 함께 가지게 된다. 1985년 Takagi에 의해 제안된 SD 연산체계는 〔표 2〕와 같다[5].

중간 합과 이전 캐리가 동시에 1 또는 $\bar{1}$이 되는 경우를 방지함으로써 캐리의 수평적 전달을 방지한다. 따라서 〔표 2〕의 덧셈 규칙을 따름으로써 SD 수 체계의 장점인 캐리 전파 제거 특성을 이진 SD 수 체계에서도 유지시킬 수 있다. SD 수 체계에서 설계된 기존의 RBA에 대해 살펴보자. 본 논문에서는 삼성 STD130 0.18μm 1.8V CMOS 셀 라이브러리에서 제공하는 게이트를 사용하여 게이트 수와 지연 시간을 비교하고자 한다. 〔그림 3〕은 RBA1부터 RBA6까지를 표현한 것이다. 좌측은 두 SD 수의 한 자리 덧셈을 수행하는 그림이고 우측은 셀에 사용된 게이트를 나타낸 것이다. 사용된 게이트는 게이트 이름, 게이트의 크기, 게이트의 지연 시간 순으로 표현한다. 그리고 셀의 전체 게이트 크기와 지연 시간을 하부에 표현한다.

〔그림 3〕의 (a)는 RBA1[2]은 Takagi의 덧셈 규칙을 수정하여 RBA의 내부 구조를 단순화 하고자 하였다.



〔그림 3〕 기존의 RB 곱셈기

하위 자리로부터의 캐리 발생 가능성의 경우를 좀 더 세분화하여 하위 자리의 두 입력 x_{i-1}, y_{i-1}이 각각 $\bar{1}$,1 또는 1,$\bar{1}$인 경우는 비록 두 하위 자리 하나가 $\bar{1}$이지만 이 경우 캐리와 중간 합을 RBA 내부로직을 단순화시키는 방향으로 정하여 줌으로써 RBA를 최적화하는 방식이다. (b)의 RBA2[9]는 입력 값과 출력 값을 모두 전환한 값으로 사용하여 게이트를 효율적인 구성으로 설계하였다. (c)의 RBA3[10]는 이전 비트 정보를 고려해야 하므로 전체 게이트 크기가 본 논문에서 비교하는 RBA 중 가장 크다. (d)의 RBA4[11]는 v_i값을 그대로 사용할 수 없다. 따라서 전환 연산이 추가로 필요하게 된다. 그림 (f)의 RBA5[9]는 본 논문에서 비교하는 RBA 중 가장 큰 지연 시간이 걸린다. 그림(f)의 RBA6[12]는 비교하는 RBA 중 게이트 크기가 가장 작고 지연 시간이 가장 적게 걸린다.

Ⅳ. 제안하는 RSA 연산

4.1. 제안하는 RB 곱셈기

본 절에서는 세 피연산자의 덧셈을 효율적으로 구성하는 새로운 RBA(NRBA)를 제안하고 몽고메리 곱셈기에 적용하여 효율적인 RSA 연산을 구성한다. 제안하는 RB 곱셈기는 4-2 CSA를 사용한 CS 곱셈기와 유사하나 핵심 구성요소인 NRBA의 입·출력 표현을 새로 정의하여 효율적인 구조 설계가 가능해진다. 또한 부호 자리 수에 대해서 BRFA는 변형된 BRFA(MBRFA)로 대체된다.

4.1.1 연산 정의

본 논문에서 다음과 같은 기호를 사용한다.

· $A \vee B$: A와 B의 OR 연산

- $A \wedge B$: A와 B의 AND 연산
- $A \oplus B$: A와 B의 XOR 연산
- $\overline{A}$: A의 전환(Invert) 연산

4.1.2 NRBA

제안하는 덧셈은 다음 두 단계로 구성된다.

- 1단계 : $Sig + Bin_1 + Neg_1 \rightarrow Neg_1 + Neg_2 + Bin_2$ ($Sig + Bin_1$을 수행하여 캐리 Bin_2와 합 Neg_2을 생성한다.)
- 2단계 : $Neg_1 + Neg_2 + Bin_2 \rightarrow Neg_3 + Bin_3$ (Neg_1과 1단계에서 생성된 Neg_2와 이전 자리의 캐리 Bin_2의 덧셈을 수행하여 캐리 Neg_3와 합 Bin_3을 생성한다.)

$x_i = (x_i^s, x_i^v) \in Sig, y_i \in Bin_1$이라 하고 이때 발생하는 캐리 $c_{i+1} \in Bin_2$, 합 $s_i \in Neg_2$라 할때 1단계에서 발생할 수 있는 경우의 수는 〔표 3〕의 (a)와 같이 총 6가지가 된다. 알고리즘을 효율적으로 구성하기 위해 〔표 3 (a)〕의 x_i^s와 s_i 대신 각각의 전환한 값을

적용하면, 즉 $\overline{x_i^s}$와 $\overline{s_i}$를 적용하면 〔표 3(b)〕와 같다.

〔표 3〕 1단계의 진리표

x_i^s	x_i^v	y_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	1	0	0	1
1	1	1	0	0

(a)

$\overline{x_i^s}$	x_i^v	y_i	c_{i+1}	$\overline{s_i}$
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1
0	1	0	0	0
0	1	1	0	1

(b)

〔표 4〕 2단계 진리표

z_i	$\overline{s_i}$	c_i	a_{i+1}	b_i
0	1	0	0	0
0	1	1	0	1
0	0	0	1	1
0	0	1	0	0
1	1	0	1	1
1	1	1	0	0
1	0	0	1	0
1	0	1	1	1

(a)

$\overline{z_i}$	$\overline{s_i}$	c_i	$\overline{a_{i+1}}$	b_i
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

(b)

알고리즘 2 : 제안하는 덧셈(X, Y, Z)

입력 : X, Y, Z

출력 : $A, B (A + B = X + Y + Z)$.

1. $A, B \leftarrow 0$.
2. Process($\overline{x_i^s}, x_i^v, y_i, \overline{z_i}$)
   Begin
   2.1. $c_{i+1} \leftarrow (x_i^v \vee y_i) \wedge \overline{x_i^s}$,
   $\overline{s_i} \leftarrow \sim (x_i^v \wedge y_i) \wedge (x_i^v \vee y_i)$.
   2.2. $\overline{a_{i+1}} \leftarrow \sim [\sim (\overline{z_i} \vee c_i) \wedge \overline{s_i} \wedge \sim (\overline{z_i} \wedge c_i)]$,
   $b_i \leftarrow \sim [\sim \{\sim (\overline{s_i} \oplus c_i) \wedge \overline{z_i}\} \wedge \{\sim (\overline{s_i} \oplus c_i) \vee \overline{z_i}\}]$.
   End process.
3. Return(A, B).

c_{i+1}와 $\overline{s_i}$는 〔표 3〕의 경우를 고려하면 식 (4)와 같이 계산된다.

$$c_{i+1} \leftarrow (x_i^v \vee y_i) \wedge \overline{x_i^s}, \overline{s_i} \leftarrow \sim (x_i^v \wedge y_i) \wedge (x_i^v \vee y_i). \quad (4)$$

식 (4)에서 $c_{i+1} \in Bin_2, \overline{s_i} \in Neg_2$이므로 두 번째 단계는 $Neg_1 + Neg_2 + Bin_2$의 연산이 된다. $z_i \in Neg_1$이라 하고, 이때 발생하는 캐리 $a_i \in Neg_3$, 합 $b_i \in Bin_3$이라 할 때 경우의 수는 (표 4 (a))와 같다. 1단계와 마찬가지로 알고리즘을 효율적으로 구성하기 위해 (표 4 (a))의 z_i와 a_{i+1}대신 각각의 전환한 값을 적용하면, 즉 $\overline{z_i}$와 $\overline{a_{i+1}}$를 적용하면 (표 4 (b))와 같다.

$\overline{a_{i+1}}$와 b_i는 식 (5)와 같이 계산된다.

$$\overline{a_{i+1}} \leftarrow \sim [\sim (\overline{z_i} \vee c_i) \wedge \overline{s_i} \wedge \sim (\overline{z_i} \wedge c_i)],$$
$$b_i \leftarrow \sim [\sim \{\sim (\overline{s_i} \oplus c_i) \wedge \overline{z_i}\} \wedge \{\sim (\overline{s_i} \oplus c_i) \vee \overline{z_i}\}]. \quad (5)$$

입력 값 $X = (x_{n-1} x_{n-2} \cdots x_0) \in Sig$ (단, $x_i = (\overline{x_i^s}, x_i^v)$)와 $Y = (y_{n-1} y_{n-2} \cdots y_0) \in \mathrm{Bin}$, $\overline{Z} = (\overline{z_{n-1}} \overline{z_{n-2}} \cdots \overline{z_0}) \in Neg$에 대한 덧셈의 결과로 $\overline{A} = (\overline{a_{n-1}} \overline{a_{n-2}} \cdots \overline{a_0}) \in Neg$, $B = (b_{n-1} b_{n-2} \cdots b_0) \in Bin$를 출력하는 덧셈은 알고리즘 2와 같다.

4.1.3 NRBA의 하드웨어 구조

한 자리 덧셈에 대한 하드웨어 구조는 기존 RBA의 비교와 동일하게 삼성 STD130 0.18um 1.8V CMOS 셀 라이브러리에서 제공되는 6개의 셀로 구성된다. 〔그림 4〕에서는 번호에 따라 셀이 나누어지며 번호, 게이트, 지연 시간 순으로 표시되어 있다. 2번과 6번은 동일

〔표 5〕 $Neg_3 + Bin_3 \rightarrow Sig$진리표

$\overline{a_i}$	b_i	$\overline{x_i^s}$	x_i^v
1	0	1	0
1	1	1	1
0	0	0	1
0	1	1	0

한 셀이나 출력의 개수에 따라 게이트, 지연 시간이 달라지므로 다른 이름을 사용한다.

출력 결과 A, B는 $A \in Neg$, $B \in Bin$이며 A는 전환 형태 $\overline{A}$를 가진다. 이 두 값은 곱셈에서 중간 결과로써 사용된다. 그러나 다음 곱셈의 입력으로 재사용되기 위해서는 A, B를 하나의 Sig값으로 생성하는 결합기(Coupler)를 사용해야 한다[1]. 그러나 본 논문에서 제안하는 RBA는 추가적인 하드웨어 없이 한 번

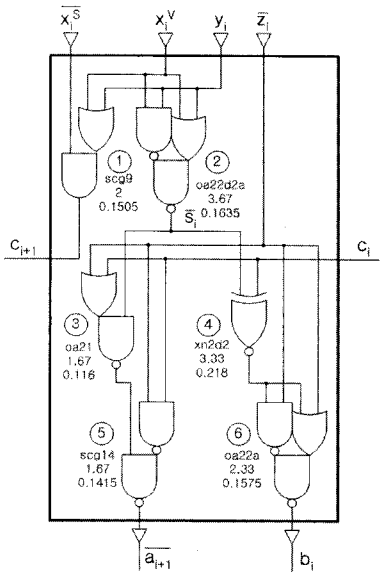
의 반복 동작으로 결합기의 역할을 수행한다. 덧셈 결과의 두 출력 값 $\overline{a_i} \in Neg_3$, $b_i \in Bin_3$에 대해, 두 값을 하나의 값 $x_i = (\overline{x_i^s}, x_i^v) \in Sig$로 결합한다고 할 때 경우의 수는 〔표 5〕와 같다.

알고리즘 3 : 제안하는 RB 몽고메리 곱셈

입력 : $X=(X^s, X^v)$, $Y=(Y^s, Y^v)$, M
$R=2^{n+3}$, $\gcd(M,2)=1$, $0 \le X, Y < 2M$.
출력 : $A=(A^s, A^v)=XYR^{-1} \bmod M$.

1. $A \leftarrow 0$.
2. $(D^s, D^v) \leftarrow NRBA(Y^s, Y^v, M, 0)$.
3. $(D^s, D^v) \leftarrow NRBA(D^v, 1, D^s, 1)$.
4. for i in 0 to $n+2$ loop
   4.1 $u_i \leftarrow (a_0^s + a_0^v) \bmod 2$.
   4.2 if $X_i = 0$ and $u_i = 0$ then
   $(a_{i+1}^s, a_{i+1}^v) \leftarrow NRBA(0,0,a_i^s, a_i^v)/2$.
   Else if $X_i = 1$ and $u_i = 0$ then
   $(a_{i+1}^s, a_{i+1}^v) \leftarrow NRBA(y_i^s, y_i^v, a_i^s, a_i^v)/2$.
   Else if $X_i = 0$ and $u_i = 1$ then
   $(a_{i+1}^s, a_{i+1}^v) \leftarrow NRBA(m_i, 0, a_i^s, a_i^v)/2$.
   Else $(a_{i+1}^s, a_{i+1}^v) \leftarrow NRBA(d_i^s, d_i^v, a_i^s, a_i^v)/2$.
5. $(A^s, A^v) \leftarrow NRBA(A^v, 1, A^s, 1)$.
6. Return(A).

〔표 5〕는 〔표 4〕에서 $\overline{z_i}=1$인 경우 오른쪽 4열과 동일함을 알 수 있다. 따라서 $\overline{z_i}=1$를 고정하고 $\overline{s_i}$에 $\overline{a_i}$를 대입하고 c_i에 b_i를 대입하면 결합기의 역할을 수행할



〔그림 4〕 제안하는 RBA 셀 구조

수 있게 된다. 또한 입력 값 $x_i^v=1$를 고정하면 c_{i+1}는 $\overline{x_i^s}$ 값이 출력되고, $\overline{s_i}$는 y_i값이 출력된다. 따라서 최하위 자리에서 $c_0 \leftarrow b_0$을 대입하고 $\overline{x_0^s} \leftarrow b_1$을 대입한다. $y_i \leftarrow \overline{a_0}$을 대입하면 c_1은 b_1이 되고 $\overline{s_0}$은 $\overline{a_0}$가 된다. 따라서 덧셈의 2단계의 경우의 수와 동일하게 되므로 결합기의 역할을 수행할 수 있다.

알고리즘 4 : 변형된 지수승 연산(C,d,M)

입력 : C,d,M
출력 : $X = C^d \bmod M$.

1. $K = 2^{2n} \bmod M$ (computed externally)
2. $(P_0^s, P_0^v) \leftarrow NRBMontMul(K,0,C,0,M)$
   $(T_0^s, T_0^v) \leftarrow NRBMontMul(K,0,1,0,M)$
3. for i in 0 to $n_d - 1$ loop (n_d : d의 비트 길이)
   $(P_{i+1}^s, P_{i+1}^v) \leftarrow NRBMontMul(P_i^s, P_i^v, P_i^s, P_i^v, M)$.
   if $d_i = 1$ then
   $(T_{i+1}^s, T_{i+1}^v) \leftarrow NRBMontMul(T_i^s, T_i^v, P_i^s, P_i^v, M)$.
4. $(X^s, X^v) \leftarrow NRBMontMul(1,0,T_n^s, T_n^v, M)$.
5. $X \leftarrow Convert(X^s, X^v)$
6. Return(X).

4-2 CSA를 제안하는 RBA로 대체함으로써 RB 몽고메리 곱셈기의 구조는 4-2 CSA 몽고메리 곱셈기와 Input/Output Buffer, Y+M, Y, M, 0의 레지스터, 4:1 MUX, 입력 값과 출력 값이 동일하다. 제안하는 RB 몽

고메리 곱셈은 알고리즘 3과 같다.

4.2 변형된 지수승 연산

일반적으로 RSA 암호화 및 복호화 함수는 X가 평문, C가 암호문, M이 n비트 모듈러스라 하고 e,d가 공개 및 비밀 지수라 할 때, 각각 $C=X^e \pmod M$와 $X=C^d \pmod M$와 같다. 또한 $M=pq$이고 p,q가 약 $n/2$ 비트 수라 할 때, $ed=1 \pmod{(p-1)\times(q-1)}$를 만족해야 한다. 본 논문에서 제안하는 RB 몽고메리 곱셈을 적용한 지수승 연산은 알고리즘 4와 같다.

V. 비교 및 결과

본 장에서는 본 논문에서 제안한 RBA와 이를 적용한 RB 몽고메리 곱셈기의 효율성에 대해 논한다. 제안하는 RBA를 NRBA라 할 때, [표 6]는 4-2 CSA를 비롯하여 기존의 RBA와 NRBA를 비교한 것이다. 4-2 CSA는 FA 2개로 구성되어 있고 지연 시간 또한 2단계의 FA가 소요된다. NRBA의 전체 지연 시간은 4-2 CSA에 비해 약 25.2% 감소되었고 가장 지연 시간이 높은 RBA5에 비해 약 39.8% 감소되었다. 기존의 RBA 중 가장 지연 시간이 낮은RBA6에 비해 약 14% 감소하였다. 공간 복잡도는 4-2 CSA에 비해 약 18.5% 감소하였고 가장 공간 복잡도가 높은 RBA3에 비해 62.7% 감소하였다. 기존의 RBA 중 가장 공간 복잡도가 낮은 RBA6에 비해 약 6.3% 감소하였다.[1]의 RBA와 비교 시 19.28% 공간 복잡도 감소와 2.8% 시간 복잡도 감소를 보였다. 그러나[1]의 RBA를 몽고메리 곱셈기에 적용할 경우 결합기를 사용해야 하므로 공간 복잡도는 증가한다. 동등 게이트 수는 26.35로써 공간 복잡도 감소폭은 약 44.3%로 증가한다. 4-2 CS 몽고메리 곱셈기와 기존의 RB 몽고메리 곱셈기 및 제안하는 RB 몽고메리 곱셈기를 [표 7]에서 비교한다. 4-2 CS 몽고메리 곱셈기와 기존의 RB 몽고메리 곱셈기는 클럭 사이클이 n+4임에 반해 제안하는 RB 몽고메리 곱셈기는 n+6이다. 그 이유는 알고리즘 알고리즘의 단계 2와 5에서 결합기의 역할을 수행하기 때문이다. 그러나 1024비트 또는 2048비트 곱셈에서 곱셈의 단위 시간이 가장 짧으므로 반복 회수 2회는 크게 비중을 차지하지 않는다. 공간 복잡도 또한 비교 대상과 동일한 레지스터 수, MUX 및 게이트를 가지고 있으므로 2FA와 1RBA에 비해 크기가 작은 1NRBA는 비트의 수가 증가할수록 큰 의미를 가진다.

참고문헌

[1] 임대성, 장남수, 지성연, 김성경, 이상진, 구본석, "새로운 잉여 이진 Montgomery 곱셈기와 하드웨어 구조", *정보보호학회*, *Vol. 16, No. 4*, pp. 33-41, 2006.

[표 6] 기존 결과와의 비교 결과

	Total delay	Components	Equivalent gates	Components
4-2 CSA [3]	0.7220	2FAs	18.00	2FAs
RBA1 [2]	0.6930	2xo2d2+1xo2	20.66	3xo2d2+1xo2+1nx2d2+1mx2
RBA2 [9]	0.6815	2nd2+2xo2d2+1iv	18.33	6nd2+2xo2d2+1mx2d2+2iv
RBA3 [10]	0.8025	1nr2d4+2iv+nr4+ad2 +1ad3d2	39.34	1or2d4+2ad3+1nr2d4+1or2d8 +2ad3d2+1or4d2+1nr4+2iv+4ad2
RBA4 [11]	0.6795	2xo2d2+1xo2	16.32	2xo2d2+1xn2+2nr2+2nd2+1ao211
RBA5 [10]	0.8960	1or2d2+1iv+1xo2d2 +2xn2d2+2ad2	22.34	2or2d2+1nr2+2iv+1xn2d2+4ad2 +1nd2+1xo2d2+ao21
RBA6 [12]	0.6270	1xo2d2+1xn2d2+1nd2d2 +1oa21	15.66	1nr2+1nd2+1xo2d2+1ao32 +1xn2d2+1nd2d2+1iv+1oa21
RBA [1]	0.5550	3nr2+3nr2b	20.01	7nr2+7nd2+3nr2b+1iv
NRBA	0.5390	1oa22da+1oa22a+1xn2d2	14.67	1scg9+1oa22d2a+1oa21+1xn2d2 +1scg14+1oa22a

[2] 홍종욱, "Redundant Binary 연산을 이용한 실수/복소수 승산기", *연세대학교 대학원, 전기 · 컴퓨터 공학회*, 1999.

[3] Ciaran McIvor, Maire McLoone, John V McCanny, Alan Daly, "Fast Montgomery modular multiplication and RSA cryptographic processor architectures", ACCSC 2003, pp 379-384, 2003.

[4] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic", IRE Trans. Electron. Comput., vol. EC-IO, no. 9, pp. 389-400, Sept. 1961.

[5] Naofumi Takagi, et. al., "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree", IEEE Trans. on Computers, Vol. C-34, No. 9, pp. 789-796, Sep. 1985.

[6] SAMSUNG STD130 0.18㎛ 1.8V CMOS Standard Cell Library for Pure Logic Products.

[7] Walter C. D., "Montgomery Exponentiation Needs No Final Subtractions", *Electronics Letters*, 35 (21) : pp. 1831-1832, 1999.

[8] Manochehri, K, Pourmozafari. S, "Modified radix-2 Montgomery modular multiplication to make it faster and simpler", *ITCC 2005*. pp. 598-602, 2004.

[9] H. Makino,Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara, and K. Mashiko, "An 8.8-ns 54×54 bit multiplier with high speed redundant binary architecture", *IEEE J. Solid-State Circuit*, vol. 31, no. 6, pp. 773-783, June 1996.

[10] Anders Lindstrom, Michael Nordseth and Lars Bengtsson, "0.13㎛ CMOS Synthesis of Common Arithmetic Units", *Technical Report No. 03-11*, Department of electrical and electronic engineering, University college Cork, 2003.

[11] D. S. Phatak and I. Koren, "Hybrid signed-digit number systems: A unified framework for redundant number representations with bounded carry propagation chains", *IEEE Transactions on Computers*, 43(8):880-891, Aug. 1994.

[12] H. Edamatsu, T. Taniguchi, T. Nishiyaina and S. Kuninobu, "A 33 MFLOPS floating point processor using redundant binary representation", *Dig. Tech. Papers of 1988 ISSCC*. pp. 152-153, Feb. 1988.

[13] B. Kaliski, "TWIRL and RSA Key Size", *RSA Labs Tech Note*, May 2003.

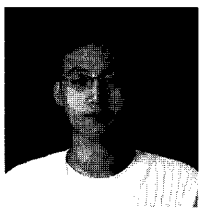
[14] Christof Paar, Thomas Blum, "High radix Montgomery modular exponentiation on reconfigurable hardware", *IEEE Transactions on Computers*, vol. 50, No. 7, pp. 759-764, 2001.

[15] I. Koren, "Computer Arithmetic Algorithms", *Englewood Cliffs*, NJ:Prentice-Hall, 1993.

〈著者紹介〉

장 남 수(Nam Su Chang) 학생회원
2002년 2월: 서울시립대 수학과 학사
2004년 8월: 고려대학교 정보보호대학원 석사
2005년 2월~현재: 고려대학교 정보보호대학원 박사과정
<관심분야> 공개키 암호 알고리즘, 무선 LAN 기술, 암호칩 설계 기술

임 대 성(Daesung Lim) 학생회원
2004년 2월: 강남대학교 수학과 학사
2006년 8월: 고려대학교 정보보호대학원 석사
<관심분야> 정보보호, 공개키 암호이론, 부채널 공격, 암호칩 설계



지 성 연(Sung Yeon Ji) 학생회원
2005년 2월: 한신대학교 수학과 학사
2005년 3월~현재: 고려대학교 정보보호대학원 석사과정
<관심분야> 공개키 암호, 암호 칩 설계 기술, 부채널 공격

김 창 한(Chang Han Kim) 평생회원
1985년 2월: 고려대학교 수학과 학사
1987년 2월: 고려대학교 수학과 석사
1992년 2월: 고려대학교 수학과 박사
1992년 3월~현재: 세명대학교 정보통신학부 교수
<관심분야> 정수론, 공개키 암호, 암호 프로토콜

윤 석 봉(Suk Bong Yoon) 평생회원
1985년 2월: 동의대학교 수학과 학사
1988년 2월: 동국대학교 수학과 석사
1992년 2월: 동국대학교 수학과 박사수료
1999년 2월: 경북대학교 수학과 박사
2000년 3월~현재: 동의대학교 수학과 교수
<관심분야> 정수론, 공개키 암호, 암호 프로토콜