情報保護學會論文誌
第11券 第5號, 2001. 10

Montgomery 곱셈기를 이용한 효율적인 모듈라 멱승기 구조

하 재 철*, 문 상 재**

Efficient Architectures for Modular Exponentiation Using Montgomery Multiplier

Jae-Cheol Ha*, Sang-Jae Moon**

요 약

본 논문에서는 공개 키 암호시스템에서의 필수적인 연산인 모듈라 멱승을 처리하기 위한 멱승기의 회로 구조를 제안한다. 제안한 멱승기는 Montgomery 알고리듬을 사용한 곱셈기를 채택하였으며 멱승의 사전·사후 계산 과정을 쉽게 처리할 수 있도록 MUX를 이용한 것이 특징이다. 논문에서는 n비트의 모듈라 멱승을 가정하여 L-R 이진 방식과 R-L 이진 방식에 기초한 두 가지 형태의 설계 구조를 제안하였다. 구현에 사용된 곱셈기가 m번 클럭의 캐리 처리과정을 포함하여 $(n+m)$번의 클럭으로 모듈라 곱셈을 처리할 수 있다고 가정할 경우, L-R 방식 멱승기는 평균 $(1.5n+5)(n+m)$번의 클럭만에, R-L 방식 멱승기는 $(n+4)(n+m)$번의 클럭 시간에 멱승을 처리할 수 있다.

ABSTRACT

Modular exponentiation is an essential operation required for implementations of most public key cryptosystems. This paper presents two architectures for modular exponentiation using the Montgomery modular multiplication algorithm combined with two binary exponentiation methods, L-R(Left to Right) and R-L(Right to Left) algorithms. The proposed architectures make use of MUXes for efficient pre-computation and post-computation in Montgomery's algorithm. For an n-bit modulus, if multiplication with m carry processing clocks can be done $(n+m)$ clocks, the L-R type design requires $(1.5n+5)(n+m)$ clocks on average for an exponentiation. The R-L type design takes $(n+4)(n+m)$ clocks in the worst case.

keyword : *공개 키 암호, 모듈라 멱승기, Montgomery 알고리듬, 모듈라 곱셈기*

Ⅰ. 서 론

Diffie-Hellman 방식에서 유래한 이산 대수에 문제에 기반한 공개 키 암호시스템이나 RSA와 같이 소인수분해 문제에 기반한 공개 키 암호시스템에서는 512비트 이상의 큰 정수에 대한 모듈라 멱승(exponentiation)연산이 필수적이다[1,2]. 특히, IC 카드와 같은 정보보호용 장치를 사용할 경우에는 멱승과 같은 특수한 연산 알고리듬을 고속으로 수행해야 하는 반면 설계시 물리적인 조건이 제한적이다. 현재 사용되는 암호 IC카드의 경우, 25mm^2 이내의 면적에 주 마이크로 프로세서를 장착하고 있으며, 512바이트의 RAM, 16K바이트의 ROM 그리고 8K 바이트의 EEPROM 등을 포함하고 있다. 지금까지

* 나사렛대학교 전산정보학과(jcha@kornu.ac.kr)
** 경북대학교 공과대학 전자전기공학부(sjmoon@knu.ac.kr)

이런 제한적 조건에서도 큰 정수의 멱승을 빠르게 수행하기 위한 하드웨어 회로 연구가 많이 있었다[3,4,5].

고속 멱승을 위한 하나의 방법은 정보보호 장치 내에 모듈라 멱승을 효율적으로 처리할 수 있는 전용 연산 장치를 부착하는 것이다. 이 고속 연산 장치는 모듈라 멱승 자체를 빠르게 할 수도 있으며 멱승의 기본 연산인 모듈라 곱셈을 고속으로 수행하여 고속화될 수도 있다. 현재, 모듈라 곱셈 방법으로는 Montgomery 알고리듬[6,7]이 가장 많이 알려져 있는데 Montgomery 알고리듬은 곱셈기의 내부 연산이 규칙적이고 데이터 흐름이 일정한 구조를 가지고 있어 이 알고리듬을 이용한 VLSI 하드웨어 연구가 많았다[8,9,10,11]. 특히, Arazi가 제안한 DSD (Digital Signature Device)[12]는 가산기, 레지스터 그리고 멀티플렉서(multiplexer, MUX)만을 사용한 간단한 회로 구조로서 Montgomery 모듈라 곱셈을 수행할 수 있다.

한편, Montgomery 모듈라 곱셈기를 이용해 멱승기를 구현한다 하더라도 곱셈기 구조가 시스토릭 어레이(systolic array)에 기반한 형태인지, 큰 정수의 덧셈기에 기반한 형태인지에 따라, 또는 곱셈기의 입·출력 형태에 따라 여러 가지 구조로 설계될 수 있다. 그리고 어떤 멱승 알고리듬을 사용했는지에 따라 하드웨어 구조가 다르게 결정되며 제어 신호의 사용 방법에 따라서도 다양한 형태로 구현될 수 있다.

본 논문에서는 고속 멱승을 위해 Montgomery 모듈라 곱셈기 이용한 멱승기의 하드웨어 회로 구조를 제시한다. 멱승기 설계시에는 덧셈기에 기반한 Montgomery 곱셈기를 사용하는데, 이 곱셈기는 곱셈을 하는 두 연산 값 중 하나는 직렬로 다른 하나는 병렬로 입력되는 구조로 되어 있다. 실제 설계된 멱승기는 A와 E가 모두 가변적인 경우를 가정하여 $A^E \bmod N$을 계산하도록 고려하였다. 또한 Montgomery 곱셈기를 사용함에 따라 멱승기에는 사전·사후 계산 과정이 필요한데 이를 쉽게 처리할 수 있도록 MUX를 이용한 것이 특징이다.

논문에서는 모듈라 멱승 알고리듬 중 대표적인 L-R 이진 방식과 R-L 이진 방식을 기초로 두 가지 형태의 멱승기 구조를 제시하였다. 본 논문의 제 2장에서는 Montgomery 모듈라 곱셈 알고리듬 및 기본 구조를 설명하며, 제 3장에서는 멱승기에 사용할 멱승 알고리듬을 두 가지 형태로 기술한다. 제 4장에서는 멱승 알고리듬에 따라 설계된 멱승기 구조를 제안하며, 제 5장에서는 ALTERA MAX+PLUS II[13]을 이용하여 그래픽 편집기로 설계한 후 시뮬레이션하여 그 결과를 검증하였다.

II. Montgomery 곱셈기 구조

공개 키 암호시스템에 사용하는 모듈라 멱승 $A^E \bmod N$은 $AB \bmod N$형태의 모듈라 곱셈의 반복으로 이루어진다. 모듈라 곱셈 $AB \bmod N$은 두 수 A, B를 곱한 결과를 모듈러스 N으로 나눈 나머지를 취하는 연산이다. 이를 효과적으로 수행하기 위해 여러 곱셈 알고리듬들이 제안되었으나 Montgomery 알고리듬이 간단하며 고속이기 때문에 널리 쓰이고 있다. 그러나 Montgomery 모듈라 곱셈은 내부 연산이 규칙적이고 데이터 흐름이 일정한 구조를 가지고 있는 것에 반해 $AB \bmod N$가 아닌 $ABR^{-1} \bmod N$을 수행하는 알고리듬이다. 여기서 R은 N과 서로 소인 N보다 큰 정수이며 A, B 및 N은 모두 n비트라고 가정한다. 즉, 각 정수를 기저(base) r로 표현하면 $A=\sum_{i=0}^{l-1} A[i]r^i$, $B=\sum_{i=0}^{l-1} B[i]r^i$ 및 $N=\sum_{i=0}^{l-1} N[i]r^i$과 같이 l자리로 나타낼 수 있는데 $r=2$인 경우에는 l자리는 n비트와 같아진다.

일반적으로 R은 n이 N의 비트 수일 때 div R이나 mod R을 간단히 계산할 수 있도록 r^n으로 하는 것이 일반적이다. 본 논문에서는 하드웨어 구현의 용이성을 위해 $r=2$라 두고 $R=2^n$이라 가정하며 Montgomery 곱셈은 $AB2^{-n} \bmod N$으로 구체화한다. 따라서 Montgomery 모듈라 곱셈 알고리듬은 $r=2$이고, N이 홀수인 경우 [그림 1]과 같이 간단히 나타낼 수 있다.

[그림 1]의 알고리듬은 n번의 루프를 돌면서 각 루프마다 2개의 n비트 가산기를 사용하고 이 출력을

```
REDC(A · B)
1. T = 0
2. for i = 0 to n−1 step 1
2a. M[i] = (T[0] + A[i]B[0]) mod 2
2b. T = T + A[i]B
2c. T = T + M[i]N
2d. T = T/2
3. if(T≥N) T = T−N
4. return (T)
```

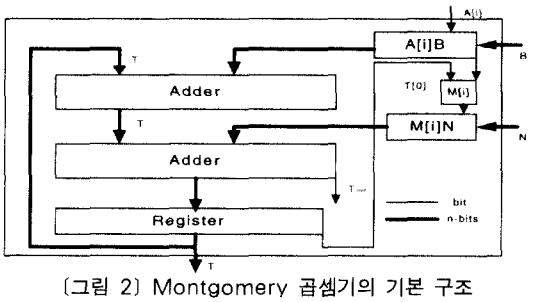
[그림 1] 간소화된 Montgomery 모듈라 곱셈 알고리듬 ($r=2$)

오른쪽으로 쉬프트시킴으로써 구현이 가능하다. 그림에서 단계 2b는 A와 B를 서로 곱하는 과정이며 단계 2c는 모듈라 감소를 하는 단계인데 덧셈과 모듈라 감소를 나누어서 수행하면 단계 2b에서 한자리의 캐리가, 단계 2c에서도 한자리의 캐리가 발생하게 된다. 알고리듬의 단계 2c와 2d는 $M[i]=0$일 경우 T를 단순히 쉬프트만 시키고, $M[i]=1$일 경우 T에 N을 더한 후 오른쪽으로 쉬프트시키는 것과 동일하다. 또한, n번째 루프가 끝난 후 단계 2d에서의 최종 계산값 T는 $T=AB2^{-n} \bmod N$가 됨을 알 수 있다. 따라서 이 알고리듬은 덧셈과 쉬프트만으로 구성되어 있어 하드웨어 구현이 용이하다. n번의 루프가 끝나고 출력되는 결과 T는 N보다 클 수도 있는데 이 경우는 간단한 뺄셈기를 이용하거나 추가적으로 한번 더 루프를 실행하여 최종결과의 범위를 조절하기도 한다.

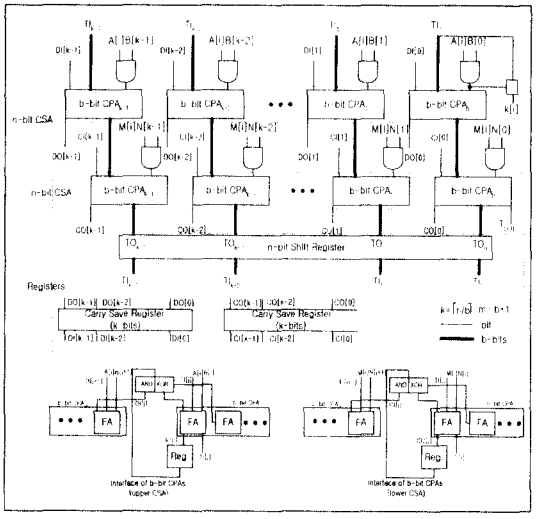
〔그림 1〕의 Montgomery 모듈라 곱셈 알고리듬을 그대로 적용한 모듈라 곱셈기 구조를 나타낸 것이 〔그림 2〕이다. 이 곱셈기의 첫번째 가산기에서는 중간 계산값을 저장하는 레지스터의 출력 T에 $A[i]B$를 더하고, 두 번째 가산기에서는 이전 가산기의 출력과 $M[i]N$을 더한다. 즉, 첫번째 가산기는 〔그림 1〕의 단계 2b를 수행하고, 두 번째 가산기는 〔그림 1〕의 단계 2c를 수행한다. 이렇게 해서 얻은 출력의 마지막 비트를 제외하고 레지스터의 입력으로 궤환(feedback)시킴으로써 오른쪽으로 쉬프트시키는 것과 똑같은 효과를 얻을 수 있다. 이 과정을 피승수 A의 각 비트 $A[i]$에 따라 1번 수행하면 $A[i]B2^{-1} \bmod N$이 되고 n번 수행함으로써 $AB2^{-n} \bmod N$이 계산된다.

이 곱셈기 구조에서 입·출력 관계를 살펴보면 n비트의 B와 N이 병렬로 입력이 되고 있으며 A가 직렬로 입력되고 있다. 여기에 사용된 가산기는 캐리 전파 문제를 해결할 수 있어야 하는데, 따라서 CPA (Carry Propagation Adder)의 사용은 어렵다. 다른 대안으로 CSA(Carry Save Adder) 형태를 사용할 수 있지만 이 경우에는 곱셈 결과가 캐리와 합(sum)의 형태로 나누어지게 되므로 최종 결과를 낼 수 있는 회로 및 추가적인 클럭 시간이 필요하다. 본 논문에서 사용한 곱셈기는 n번의 루프를 수행하는데 n클럭을 소요하며 캐리와 합을 더하여 최종 결과를 얻는데 m번의 클럭을 소요하는 것이므로 한번의 곱셈에는 $(n+m)$번의 클럭이 필요한 곱셈기이다. 또, 〔그림 2〕에서와 같이 모듈러스 N은 병렬로 입력되며, 승수와 피승수 중 하나는 병렬로 다른 하나는 매 클럭마다 한 비트씩 직렬로 입력되는 것을 사용한다. 모듈라 곱셈기의 최종 결과는 n루프 후의 T로서 병렬로 출력된다.

〔그림 2〕의 곱셈기 기본 구조를 구체적으로 블록화하여 곱셈기 회로로 나타낸 것이 〔그림 3〕이다. 〔그림 3〕에서 n비트의 덧셈기는 캐리를 저장할 수 있는 CSA를 사용하였는데 캐리를 매 FA(Full Adder)마다 발생하는 것이 아니라 일정한 크기의 CPA마다 하나의 캐리를 발생하도록 하였다. 즉, b개의 FA를 묶어 b비트용 CPA를 만든 후 클럭이 입력될 때마다 하나의 CPA에서 하나의 캐리를 발생시키는 구조를 사용하고 있다. 이러한 구조는 매 FA마다 캐리를 발생시키는 기존의 CSA 구조보다 발생하는 캐리 수가 적어 m클럭만으로 캐리를 처리할 수 있어 한번의 곱셈에는 $(n+m)$번의 클럭만이 필요하게 된다. 여기서, $b=\lceil n/k \rceil$ 이며 $m=b+1$이다.



〔그림 2〕 Montgomery 곱셈기의 기본 구조



〔그림 3〕 Montgomery 곱셈기 회로

Ⅲ. Montgomery 멱승 알고리듬

멱승 $A^E \bmod N$을 계산하는 알고리듬으로는 이진(binary) 방식[14], 이진 잉여(binary redundant) 방식, m-ary 방식, window 방식 및 지수 folding을 이용하는 방식[15,16] 등이 있다. 그러나 이진 잉여 방식은 역수를 구하는 과정이 필요하여 전체적인 멱승 속도가 다소 늦은 편이다. 그리고 m-ary나 window 방식은 이진 방식에 비해 고속이지만 많은 임시 메모리를 필요로 하고 알고리듬이 복잡하여 하드웨어 구현에서는 많이 사용하지 않는다.

하드웨어 구현에서는 알고리듬이 간단한 이진 방식을 주로 사용하는데 이진 방식은 지수 E를 이진수로 표현한 후 지수를 탐색하면서 모듈라 곱셈을 반복하는 방식이다. 지수의 탐색 순서에 따라 두 가지 형태로 구분할 수 있는데 하나는 지수의 최상위부터 탐색하는 Left-to-Right 방식(L-R형)이며 다른 하나는 최하위부터 탐색하는 Right-to-Left 방식(R-L형)이다. 일반적으로 이진 방식은 n비트의 지수를 사용할 경우 L-R형이나 R-L형 모두 평균 $1.5n$번의 모듈라 곱셈이 필요하다.

그런데 멱승에 사용된 곱셈 알고리듬이 Montgomery 알고리듬일 경우에는 사전·사후 계산 과정이 필요하다. 즉, 멱승 $A^E \bmod N$을 계산할 경우 먼저 A를 N-잉여류로 변환하는 사전 과정이 필요하고 이 값을 멱승한 후 다시 일반 수로 변환하는 사후 처리 과정이 필요하다. 이를 정리하면 다음과 같은 3단계로 표현할 수 있다. 여기서 $REDC(A \cdot B) = ABR^{-1} \bmod N$로 정의한다.

단계 1(사전처리) :

$$A' = REDC(A \cdot R^2) = AR \bmod N$$

단계 2(멱승과정) :

$$C' = A'^E \bmod N = A^E \cdot R \bmod N$$

단계 3(사후처리) :

$$C = REDC(C' \cdot 1) = A^E \bmod N$$

단계 1의 사전 처리 과정에서는 $R' = R^2 \bmod N$을 미리 계산하여 두었다가 $A = REDC(A \cdot R')$와 계산하면 쉽게 구할 수 있다. 그리고 Montgomery 방식으로 멱승 처리가 끝난 수에 대해서는 $C = REDC(C' \cdot 1)$을 계산함으로써 $C = A^E \bmod N$을 얻을 수 있다. Montgomery 곱셈 알고리듬을 이용한 이진 멱승 방식을 기술한 것이 [그림 4]이다.

```
L-R Binary : A^E mod N
A' = REDC(A · R^2);
C' = REDC(1 · R^2);
for i = n-1 to 0 step -1{
    C' = REDC(C' · C');
    if( e_i = 1) then C' = REDC(C' · A');}
C = REDC(C' · 1);
return( C);
```

```
R-L Binary : A^E mod N
C' = REDC(1 · R^2);
S' = REDC(A · R^2);
for i = 0 to n-1 step 1{
    if( e_i = 1) then C' = REDC(C' · S');
    S' = REDC(S' · S'); }
C = REDC(C' · 1);
return( C);
```

[그림 4] 이진 멱승 방식

멱승기를 하드웨어로 구현할 경우 L-R 방식과 R-L 방식의 구조는 다르게 설계될 수 있다. [그림 4]에서 보면, 멱승기의 주요 부분인 모듈라 곱셈 부분을 처리할 경우 L-R 방식의 경우는 결과 값 C'이 연속적으로 이용되고 있으므로 하나의 모듈라 곱셈기를 반복해서 이용할 수 있다는 장점이 있다. 이 경우 모듈라 곱셈 횟수는 언급한 바와 같이 평균 $1.5n$번이 된다. 반면, R-L 방식의 경우, 주요 멱승이 수행되는 동안 두 개의 결과 값 C'과 S'을 구해야 한다. 하나의 모듈라 곱셈기를 이용하여 설계할 경우 이 두 값을 번갈아 가면서 구해야 하므로 하드웨어가 복잡해질 수 있다. 그러므로 R-L 방식을 사용하는 경우에는 두 개의 모듈라 곱셈기를 이용하여 C'과 S'을 병렬로 구할 수 있도록 하는 구조가 가능하다. 물론, 이 경우에는 n번의 모듈라 곱셈을 수행하는 시간과 거의 같게 된다. 따라서 회로의 설계공간이 충분하고 처리 시간의 중요도가 높은 경우에는 R-L 방식이 선호된다. 반면, 처리 시간은 다소 지연되더라고 하나의 곱셈기를 활용함으로써 설계공간을 줄이고자 하는 경우에는 L-R 방식이 더 유용하다.

Ⅳ. 새로운 멱승기 구조 설계

본 장에서는 L-R 이진 방식 및 R-L 이진 방식을 사용하면서 Montgomery 모듈라 곱셈기를 채택한

새로운 멱승기 구조를 제안한다. 일반적으로 L-R 이진 방식은 직렬형으로 설계하며 R-L 이진 방식은 병렬형으로 설계하는데 본 논문에서도 이 형태를 사용한다. 그러나 실제 회로를 구성함에 있어서는 Montgomery 멱승기의 핵심인 사전 계산, 멱승 과정 그리고 사후 계산을 효율적으로 처리할 수 있는 회로 구조를 제안한다.

4.1 L-R 이진 멱승기 구조

L-R 이진 멱승을 위해 필요한 레지스터를 보면, [그림 4]의 알고리듬에서 보는 바와 같이 $R^2 \bmod N$을 저장할 수 있는 공간이 필요하다. 이 값은 멱승시의 모듈러스 N이 결정되면 사전에 계산될 수 있다. 두 번째로, $A' = AR \bmod N$ 값을 저장할 수 있는 레지스터가 필요하다. 마지막으로 임시 변수 C'을 저장할 수 있는 레지스터와 곱셈기의 병렬 입력을 위한 임시 레지스터가 필요하다. 모듈라 곱셈기와 저장 레지스터를 이용하여 설계한 모듈라 멱승기 구조는 [그림 5]와 같으며 각 구성요소에 대한 설명은 다음과 같다. 단, 여기서 Set 신호는 레지스터 값을 "1"로 초기화하는 제어신호를 의미한다.

가. SW controller :
- 지수 E에 따라 4:1 MUX의 출력을 선택하도록 하는 선택 신호 발생기
- 사전 계산 과정 동안에는 SW1=0, SW0=1을 발생
- 지수 비트 값 $E[i]$가 0이면 SW1=0, SW0=0 발생하여 자승을 하도록 함
- 지수 비트 값 $E[i]$가 1이면 SW1=1, SW0=1 발생하여 AR Reg.의 값을 곱함
- 사후 계산 과정 동안에는 SW1=1, SW0=0을 발생

나. 4:1 MUX : 선택 스위치에 의해 곱셈기로 입력되는 직렬 신호를 제어

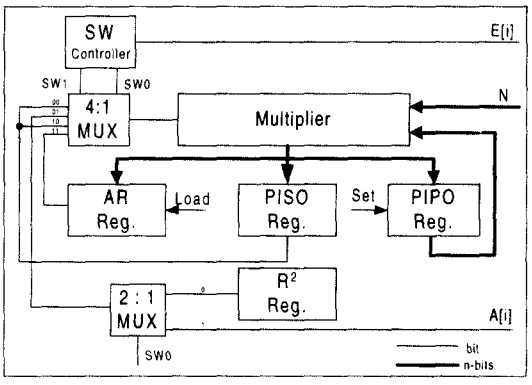
다. 2:1 MUX : 선택 스위치에 의해 사전 계산과 멱승과정 동안 입력되는 값을 결정

라. AR Reg. : $AR \bmod N$이 계산되어 저장된 좌측 순환(Left Rotator)형 레지스터

마. R2 Reg. : $R^2 \bmod N$이 사전에 저장된 좌측 순환형 레지스터

바. PISO Reg. : 병렬 입력/직렬 출력형 레지스터, 곱셈기의 출력을 병렬로 받아 직렬로 변환하여 출력하는 회로

사. PIPO Reg. : 병렬 입력/병렬 출력형 레지스터, 곱셈기의 출력을 병렬로 받아 다음 루프 곱셈의 병렬 입력으로 사용하기 위한 회로



[그림 5] L-R 이진 멱승기의 기본 구조

Montgomery 곱셈 알고리듬을 이용한 멱승기로서 동작하게 하는 가장 핵심적인 회로가 MUX인데 멱승이 진행되는 동안의 선택 스위치 및 곱셈기의 입·출력 절차는 [표 1]과 같다. 이 멱승기에서 가장 큰 특징은 두 개의 MUX를 곱셈기의 직렬 입력단 앞에 연결함으로써 Montgomery 멱승시 발생하는 곱셈기의 입력 스위칭 작용을 원활하게 한 것이다. 즉, 곱셈기의 직렬 입력단은 MUX를 이용하여 수시로 입력값을 바꿀 수 있도록 하고 병렬 입력단은 입력값을 바꾸지 않고 가급적 이전 루프 값을 반복 사용하도록 하였다. 단, 병렬 입력단의 입력값을 "1"로 초기화시키는 것은 간단한 작용이므로 설계시 이를 고려하였다. 만약, 곱셈기의 병렬 입력단에 입력값을 바꿀 수 있도록 설계하였다면 병렬 입력단의 각 비트마다 MUX가 필요하게 되어 하드웨어적으로 많은 회로가 필요하게 될 것이다. 따라서 곱셈기의 병렬 입력단에 MUX를 두지 않고 직렬 입력단 앞에 둠으로서 단지 두 개의 MUX로만 멱승기의 사전·사후 계산시 수반되는 입력 스위칭 문제를 쉽게 해결하였다.

[표 1]의 사전계산 단계 중 [그림 4]의 알고리듬에 있는 $A' = REDC(R^2 \cdot A) = AR \bmod N$를 바로 구하지 않고 앞에 두 단계를 더 두는 것도 역시 사용하는 곱셈기의 입력이 하나는 병렬이고 하나는 직렬인 점을 고려한 것이다. 즉, $REDC(R^2 \cdot A)$를 바로 계산하기 위해서는 초기값으로 R^2이나 A값 중 하나를 병렬 입력단으로 입력해야 하는데 이를 위해서는

〔표 1〕 L-R 이진 멱승기의 동작 과정

연산 과정		4 : 1 MUX	2 : 1 MUX	곱셈기 입력		연산 동작	비 고
				직렬	병렬		
사전 계산		01	1	A	1(Set)	$REDC(A \cdot 1) = AR^{-1} \bmod N$	
			0	R^2	$AR^{-1} \bmod N$	$REDC(R^2 \cdot AR^{-1}) = A \bmod N$	
			0	R^2	$A \bmod N$	$REDC(R^2 \cdot A) = A' = AR \bmod N$	AR Reg.에 Load
			0	R^2	1(Set)	$REDC(R^2 \cdot 1) = C' = R \bmod N$	
멱승	자승	00	×	C'	C'	C'	
	곱셈	11	×	$AR \bmod N$	C'	C'	
사후 계산		10	×	C'	1(Set)	$REDC(C' \cdot 1) = C = A^E \bmod N$	

n비트의 2:1 MUX가 있어야 한다. 그러나 곱셈기의 병렬 입력을 "1"로 만들기는 어렵지 않으므로 앞의 두 단계를 더 두어 $AR \bmod N$을 쉽게 구할 수 있게 하였다.

그러면 모듈라 멱승기를 구현할 경우에 필요한 논리 게이트의 개수를 산출해 보자. 먼저 〔그림 5〕에서 보는 바와 같이 크게 하나의 모듈라 곱셈기, n비트용 레지스터가 2개(R2 Reg., PIPO Reg.), 그리고 2개의 병렬 입력/직렬 출력형 레지스터(PISO Reg., AR Reg.)가 필요하다. 또한 2개의 MUX가 필요하지만 하드웨어 전체 용량에 비해 큰 영향을 미치지는 않는다.

본 논문에서 사용한 모듈라 곱셈기는 문헌 〔17〕과 〔18〕에서 제안한 것을 사용하였는데 구체적인 회로는 〔그림 3〕과 같다[17,18]. 〔그림 3〕에서 보면 중간 계산값 T와 캐리를 저장하기 위해 $n+2k$개의 D 플립플롭을 사용하였으며, 2번의 n비트 덧셈을 위해서는 $2n$개의 전가산기를, $A[i]B$와 $M[i]N$의 계산에는 각각 n비트 AND 게이트를, 그리고 캐리 처리를 위한 CPA간의 인터페이스에는 $4k$개의 AND 혹은 XOR 게이트를 사용하였다. 따라서 곱셈기의 하드웨어 용량을 산출해 보면 다음과 같다.

곱셈기의 게이트 수 = D플립플롭$(n+2k)$
+전가산기$(2n)$ + AND/XOR$(2n+4k)$
$= (5+10+2)n + (10+4)k = 17n + 14k$ 게이트

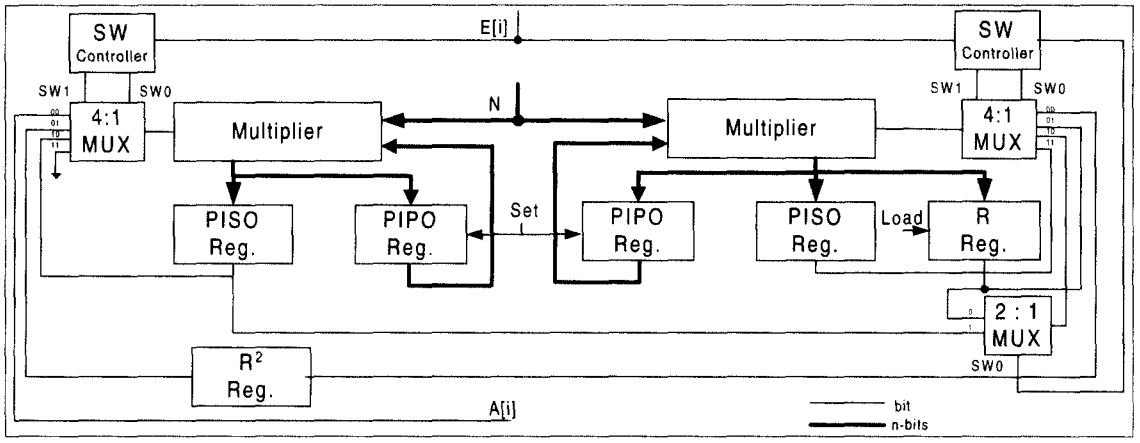
문헌 〔18〕에서는 n-비트 CSA를 이용한 곱셈기를 설계하였는데 위 계산식의 k는 $k = \lceil n/b \rceil$로서 CPA를 가지는 블록의 개수를 나타내는 인자이다. 여기서 b는 하나의 CPA안에 있는 FA의 개수를 의미한다. 본 논문에서는 $m = (b+1)$로 표기하는데 주로 CSA를 이용한 모듈라 곱셈기에서 캐리를 처리하는데 소요된 클럭 수를 의미한다. 논문에서는 한번의 곱셈을 처리하는데 $(n+m)$번의 클럭이 필요한데 상기한 바와 같이 n번의 클럭은 n번의 덧셈 루프를 위해 m클럭은 CSA의 캐리를 처리하는데 필요한 클럭 수이다.

모듈라 곱셈기를 제외한 모듈라 멱승을 위한 하드웨어로 용량을 산출해 보자. 논문에 사용된 병렬 입력/직렬 출력 레지스터는 n개의 D플립플롭과 $2n$개의 AND 게이트 그리고 n개의 OR 게이트로 구성할 수 있는데 모두 2개가 사용되었다. 또한 저장용 레지스터는 단지 n개의 D플립플롭으로 구성할 수 있는데 모두 2개가 사용되었다. 따라서 총 $4n$개의 D플립플롭과 약 $6n$개의 AND 혹은 OR 게이트가 사용되었음을 알 수 있다. 단, 2개의 MUX와 SW 발생기는 전체회로에 비해 게이트 용량이 많지 않아 계산에서 제외하였다. 여기서 사용된 D플립플롭은 5개의 기본 논리 게이트로, 전가산기는 5개의 게이트로 구현할 수 있다고 가정하여 멱승기 구현에 필요한 게이트를 산출하였는데 총 $43n+14k$개의 게이트가 필요하게 된다.

멱승기의 게이트 수
= 곱셈기 + 저장 혹은 순환형 레지스터
= (D플립플롭$(n+2k)$ + 전가산기$(2n)$ +
AND/XOR$(2n+4k)$) + (D플립플롭$(4n)$
+AND/OR$(6n)$)
$= (17+26)n + (10+4)k = 43n + 14k$ 게이트

4.2 R-L 이진 멱승기 구조

이 절에서는 R-L 이진 멱승 알고리듬을 이용하여 병렬 구조로 설계하고자 한다. 〔그림 4〕의 R-L 이진 알고리듬에서 보는 바와 같이 S'을 연속하여 자승하는



〔그림 6〕 R-L 이진 멱승기의 기본 구조

곱셈기와 임시 계산값 C'에 $E[i]$의 값에 따라 S'을 곱하는 곱셈기가 필요하다. 모듈라 곱셈기와 저장 레지스터를 이용한 R-L형 모듈라 멱승기 구조는 〔그림 6〕과 같으며 각 구성요소에 대한 설명은 다음과 같다. 각 레지스터 및 제어신호는 L-R형 멱승기에서 설명한 바와 같으며 차이가 있는 부분은 다음과 같다.

가. SW controller :
- 지수 E에 따라 4:1 MUX의 출력을 선택하도록 하는 선택 신호 발생기
- 사전 계산 과정 동안에는 SW1=0, SW0=0 혹은 SW1=0, SW0=1 을 발생
- 멱승 연산 과정에는 SW1=1, SW0=0 을 발생
- 사후 계산 과정 동안에는 SW1=1, SW0=1 을 발생

나. 2:1 MUX : 선택 스위치에 의해 멱승 계산과정 동안 입력되는 값을 결정
- 지수 비트 값 $E[i]$가 0이면 SW0=0 발생하여 $R \bmod N$ 곱함
  그러나 실제로는 곱셈이 일어나지 않음
  (예, $REDC(T' \cdot R) = T'$)
- 지수 비트 값 $E[i]$가 1이면 SW0=1 발생하여 곱셈 $C' = REDC(C' \cdot S')$을 수행

다. R Reg. : $R \bmod N$이 계산되어 저장된 좌측 순환형 레지스터

Montgomery 곱셈 알고리듬을 이용한 R-L형 멱승기의 동작과정을 선택 스위치 및 곱셈기 입·출력 절차에 따라 정리한 것이 〔표 2〕이다. 〔그림 4〕의 R-L 이진 멱승 알고리듬의 사전 계산 단계를 왼쪽에 있는 곱셈기와 오른쪽 곱셈기가 독립적으로 각각 처리한다. 이후의 실제 멱승 과정동안에는 왼쪽 곱셈기는 멱승만 수행하고 오른쪽 곱셈기는 $E[i]$에 따라 실제 결과값 C'을 계산한다. 그리고 이 과정을 n번 반복된 후 오른쪽 곱셈기는 사후 계산과정을 거쳐 최종 결과 $C = A^E \bmod N$를 출력하게 된다.

〔표 2〕 R-L 이진 멱승기의 동작과정

연산 과정	4 : 1 MUX	왼쪽 곱셈기 입력 및 동작			2 : 1 MUX	오른쪽 곱셈기 입력 및 동작		
		직렬	병렬	연산 동작		직렬	병렬	연산 동작
사전 계산	00	A	1(Set)	$AR^{-1} \bmod N$	×	R^2	1(Set)	$R \bmod N$
	01	R^2	$AR^{-1} \bmod N$	$A \bmod N$	×	$R \bmod N$	$R \bmod N$	$R \bmod N$
	01	R^2	$A \bmod N$	$S' = AR \bmod N$	×	$R \bmod N$	$R \bmod N$	$R \bmod N$
멱승	10	S'	S'	S'(자승)	0	$R \bmod N$	$C' \bmod N$	$C' \bmod N$
					1 (곱셈)	S'	$C' \bmod N$	$C' \bmod N$
사후 계산	11				×	$C' \bmod N$	1(Set)	$REDC(C' \cdot 1) = C = A^E \bmod N$

여기서 R-L형 모듈라 멱승기를 구현할 경우에 필요한 논리 게이트의 개수를 산출해 보자. 우선 [그림 6]에서 보는 바와 같이 두 개의 모듈라 곱셈기, n비트용 레지스터가 3개(R2 Reg., PIPO Reg.), 그리고 3개의 병렬 입력/직렬 출력형 레지스터(PISO Reg., R Reg.)가 필요하다. 추가적으로 3개의 MUX가 필요하지만 하드웨어 전체 용량에 비해 큰 영향을 미치지는 않는다. 그러므로 모듈라 곱셈기들을 제외한 모듈라 멱승을 위한 하드웨어로는 총 $6n$개의 D플립플롭과 약 $9n$개의 AND 혹은 OR 게이트로 요약할 수 있다.

그러므로 R-L형 멱승기를 구현할 경우, 곱셈기 회로는 2배로 늘어나고 레지스터 부분에서는 2/3배가 추가로 증가하게 되어 총 $73n+28k$개의 게이트가 필요하게 된다. 이는 L-R형보다 약 70%가 증가한 결과이다.

멱승기의 게이트 수
= 2개의 곱셈기 + 저장 혹은 순환형 레지스터
= 2(D플립플롭($n+2k$) + 전가산기($2n$)
+AND/XOR($2n+4k$))
+(D플립플롭($6n$) + AND/OR($9n$))
= $(34+39)n+2(10+4)k=73n+28k$ 게이트

V. 시뮬레이션 및 검토

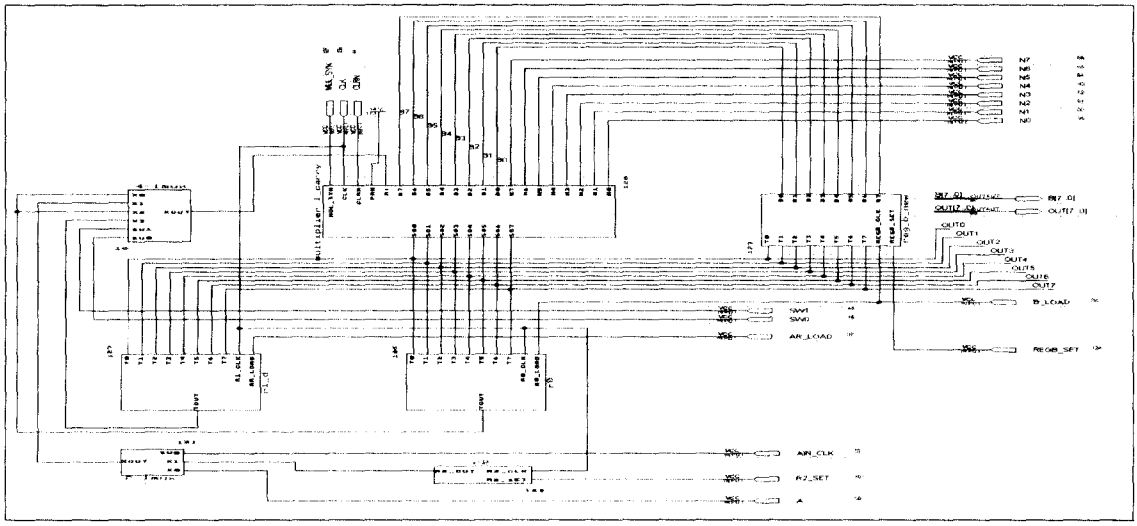
상기한 Montgomery 곱셈기를 사용한 멱승기의 정확한 동작을 검증하기 위해 멱승기 회로를 VLSI 설계 도구인 ALTERA MAX+PLUS II를 사용하여 schematic 방법으로 구현하였다. 논문에서는 검증은 편의를 위해 8비트의 A, E, N을 사용하였으며 $A^E \bmod N$을 계산하는 멱승 회로를 ALTERA MAX+PLUS II의 그래픽 편집기(graphic editor)로 설계하였다. 그래픽 편집기로 설계한 것이 [그림 7]이며 그림의 (a)가 L-R형이며, (b)가 R-L형의 멱승기의 설계도이다. 그림 (a)의 중앙에 있는 큰 회로가 곱셈기이며 곱셈기의 좌측이 4:1 MUX, 그리고 나머지는 제어 신호 및 레지스터들이다. L-R형 알고리듬에 기반한 것은 직렬형 구조로 되어 있어 하나의 곱셈기를 반복해서 사용한다. 따라서 L-R 방식 멱승기는 평균 $(1.5n+5)$번의 모듈라 곱셈을 수행하며 최대 $(2n+5)$번의 모듈라 곱셈을 수행한다.

반면 [그림 7]의 (b)인 R-L형 멱승기는 중앙에 크게 위치해 있는 두 개의 곱셈기를 사용하며 좌측의 곱셈기는 [표 2]에 나타낸 것과 같이 대부분 자승 연산을 수행하며 오른쪽 곱셈기는 곱셈 연산을 수행한다. 이 멱승기는 병렬형 구조로 되어 있어 총 $(n+4)$번의 모듈라 곱셈으로 최종 결과를 얻을 수 있다.

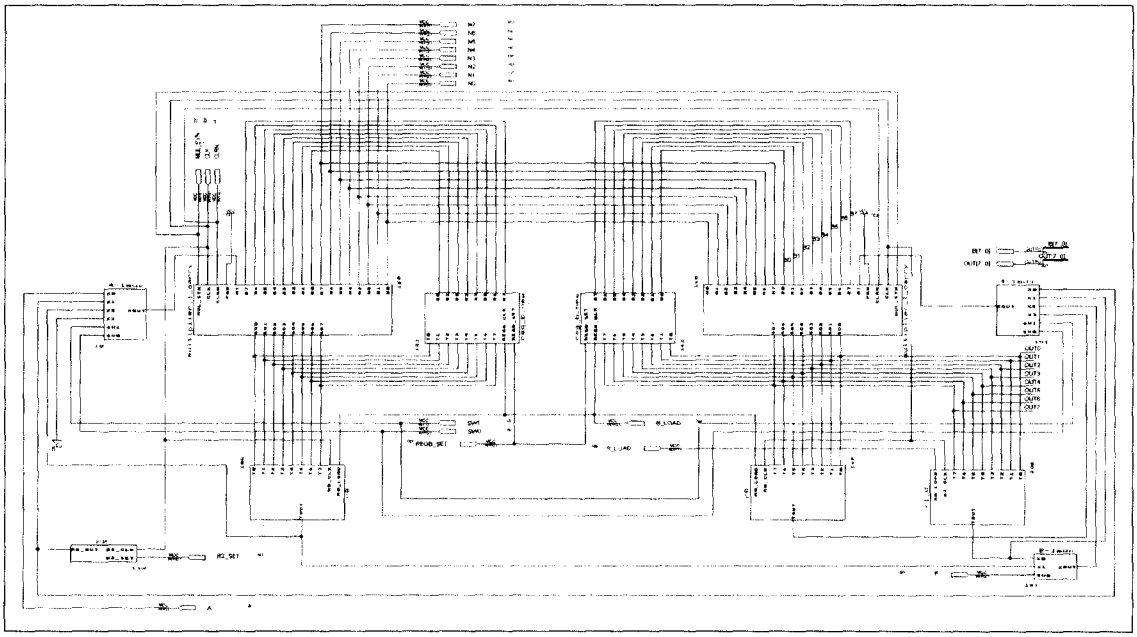
위의 멱승기를 이용하여 시뮬레이션 한 결과의 예가 [그림 8]에 나타나 있다. 예에서는 8비트를 기준으로 $A = 10000101_2 = 133_{10}$, $E = 10100101_2 = 165_{10}$, $N = 11101001_2 = 233_{10}$인 $A^E \bmod N$을 계산한 결과인데 이론 값은 $133^{165} \bmod 233 = 01101000_2 = 104_{10}$ 이다. 그림에서 CLK는 레지스터에 인가되는 주 클럭으로서 주파수를 5MHz(한 클럭 시간 200 ns)로 하였다. 한번의 곱셈을 수행하는 데에는 $(n+m)$번의 클럭이 필요한데 예에서는 $m=5$로 하였으므로 한번의 곱셈은 $(8+5)\cdot 200ns = 2.6\mu s$에 마치게 된다. 그림의 파형은 R-L형을 기준으로 한 것인데 처음 3번의 모듈라 곱셈시간 동안은 사전 계산 과정, 다음 8번의 모듈라 곱셈시간 동안은 멱승과정, 나머지 한번은 사후계산 과정을 나타낸다. 따라서 $(3+8+1)\cdot 2.6\mu s = 31.2\mu s$만에 최종 결과, $133^{165} \bmod 233 = 01101000_2 = 104_{10}$가 출력되어 이론 값과 같음을 [그림 8]의 OUT[7..0] 단자에서 확인할 수 있다. 물론 L-R형 멱승기도 계산 시간이 다를 뿐 동일한 결과를 확인할 수 있었다.

실제 응용의 예로서 $n=512$비트의 멱승 시간을 산출해 보자. 논문에서 사용한 모듈라 곱셈기에서 CPA의 FA 수인 b를 16으로 두면 $m=(b+1)=17$이 된다. 따라서 한번의 멱승을 수행하는데 L-R형은 평균 $(1.5n+5)(n+m)=0.41$M번의 클럭 사이클이 필요하며 최대 $(2n+5)(n+m)=0.54$M번, 그리고 R-L형은 최대 $(n+4)(n+m)=0.27$M번이 필요하다. 만약 주 클럭을 5MHz로 사용하면 한 클럭 시간이 200ns가 되므로 L-R형은 평균 $(1.5n+5)(n+m)200ns = 82ms$가 소요되며 최대 109$ms$가 소요된다. 또한, R-L형은 최대 $(n+4)(n+m)200ns = 55ms$가 소요되어 두 멱승기 모두 실시간 처리에는 문제가 없어 보인다. 물론 주 클럭을 빠른 것으로 사용함으로써 한 클럭 시간을 줄이고 전체적인 멱승 속도를 높일 수 있으나 클럭 사이클 수는 변하지 않아 이를 논문에서 비교 기준으로 설정하였다. 산출식에서와 같이 비트 증가에 따른 클럭 수의 증가는 $O(n^2)$의 위수 관계를 갖는다.

멱승기 구현에 사용된 게이트 수는 L-R형일 경우에는 4.1절에서 언급한 바와 같이 $43n+14k$개의



(a) L-R형 멱승기

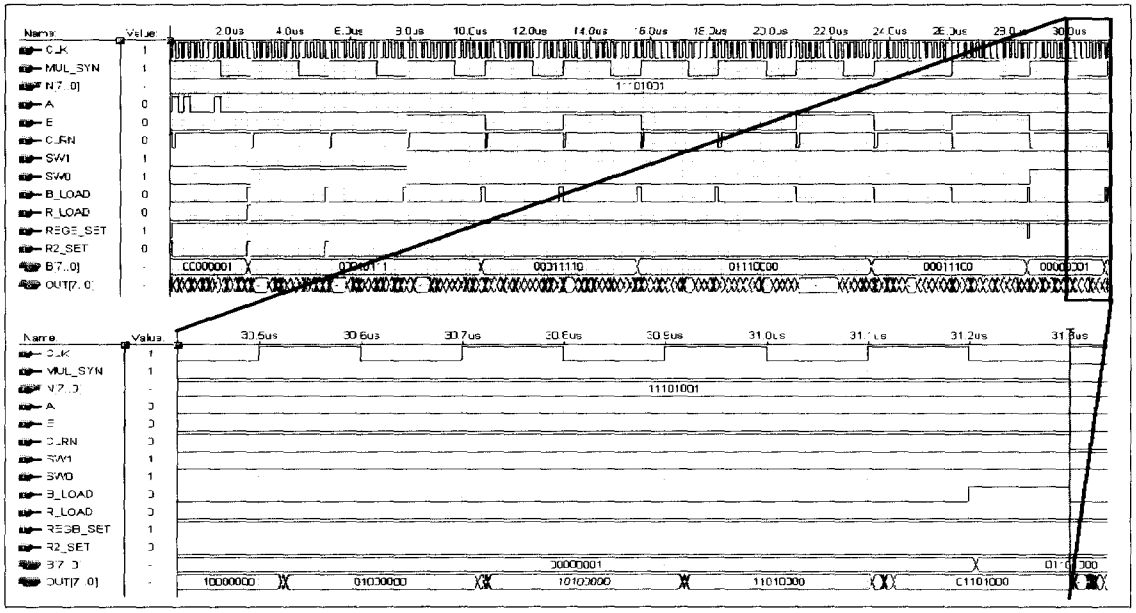


(b) R-L형 멱승기

〔그림 7〕 Montgomery 곱셈기를 사용한 멱승기 회로

게이트가 소요되는데 $n=512$이고 $k=32$(즉, $b=16$)인 경우에는 약 22K개의 게이트가 필요하다. 비슷한 방법으로 R-L형일 경우에는 4.2절에서 언급한 바와 같이 $73n+28k$개의 게이트가 소요되는데 이 경우에는 약 38K개의 게이트가 필요하다. 계산식에서 보는 바와 같이 비트 증가에 따른 게이트 수의 증가는 $O(n)$의 위수 관계를 갖는다.

위의 멱승기와 기존의 멱승기와의 성능을 비교한 것이 〔표 3〕이다. 비교에 있어 고려할 점은 실제 구현에 사용된 주 클럭 주파수가 각각 다르므로 정확한 성능 비교를 위해 필요한 클럭 사이클 수를 비교 요소로 채택하였다. 참고 문헌의 대부분이 512비트를 기준으로 구현되어 있어 이를 기준으로 하였으며 512비트 구현이 아닌 경우에는 비트가 2배 증가함에



〔그림 8〕 시뮬레이션 파형

〔표 3〕 512비트 멱승시의 성능 비교

구 분			년도	클럭 사이클 수	게이트 수	논문에서 구현 예
본 논문 (k=32 m=17)	L-R형	평균	2001	0.41M	22K	512비트, 주클럭 5MHz
		최대	2001	0.54M		512비트, 주클럭 5MHz
	R-L형		2001	0.27M	38K	512비트, 주클럭 5MHz
Victor[20]			1994	0.125M	75K	512비트, 주클럭 25MHz
NTT[21]			1994	1.00M	53K	1024비트, 주클럭 40MHz
Chen[22]			1995	1.05M	77K	512비트, 주클럭 50MHz
Yang *et al.*[4]			1998	0.54M	74K	512비트, 주클럭 125MHz
Guo *et al.*[5]			1998	0.26M	132K	512비트, 주클럭 143MHz
Leu *et al.*[19]			1999	0.53M	64K	512비트, 주클럭 115MHz

따라 클럭 사이클 수는 4배, 필요한 게이트 수는 2배로 증가하는 것으로 가정하였다[4,5,19]. 실제로 구현 비트 수 n의 증가에 따라 하드웨어 용량은 정비례하고 계산 시간은 n^2에 비례하는 것이 일반적 견해이다.

본 논문의 L-R형 멱승기에서 필요한 클럭 사이클 수는 Yang이나[4] Leu[19]가 구현한 것과 비슷한 성능을 보인다. 또 병렬 구조인 R-L형은 Guo가 구현한 것과 비슷한 클럭 사이클 수를 보이는데 그 이유는 두 구조가 모두 병렬형을 취하고 있기 때문이다. 그러나 Guo의 것은 곱셈기가 시스토릭 어레이에 기반한 것을 사용한 반면 본 논문에서는 덧셈기에 기반한 것을 사용하였다. 본 논문에서의 필요한 게이트 수가 적은 이유는 다음 두 가지 착안점이 중요한 역할을 하였다. 첫째, 곱셈기를 설계할 때 무엇보다도 설계 공간이 최소화 될 수 있도록 덧셈기에 기반한 구조를 채택하였고 그러면서도 캐리 전파 문제를 쉽게 해결할 수 있는 독특한 구조를 사용하였다. 둘째, 멱승의 사전·사후 계산시 필요한 입력의 스위칭 동작을 곱셈기의 직렬 입력단 앞의 MUX를 이용하여 간소화하였다. 다른 논문의 경우, 곱셈기를 만들 때 캐리 전파문제를 해결하기 위해 시스토릭 어레이 구조를 사용하거나 별도의 n비트용 CPA를 두고 있기 때문에 성능 저하 및 설계 공간의 부담으로 작용하였다. 결국, 본 논문에서 설계한 멱승기는 타 구현에 비해 수행속도나 설계 공간 측

면에서 대등하거나 우수한 성능을 보이고 있음을 알 수 있다.

Ⅵ. 결 론

본 논문에서는 정보보호 서비스를 제공할 경우에 필수적인 모듈라 멱승을 효율적으로 처리하기 위해 새로운 모듈라 멱승기 회로를 제안하였다. 제안한 멱승기에는 큰 정수의 덧셈기에 기반한 Montgomery 모듈라 곱셈기를 사용하였는데 이 곱셈기는 모듈러스 N을 병렬로 입력하고, 승수와 피승수 중 하나는 병렬로, 다른 하나는 직렬로 입력하는 구조를 가지고 있다. 또한, Montgomery 곱셈기를 적용한 멱승기에서의 필수 연산인 사전·사후 계산 과정을 효과적으로 처리할 수 있도록 입력 스위칭용 MUX를 활용한 것이 중요한 특징이다.

제안하는 모듈라 멱승기는 L-R형과 R-L형 이진 멱승 방식에 기초하여 각각 설계하였으며 하드웨어 설계 부담을 줄이는 부분에 중점을 두었다. 만약, n 비트의 모듈라 멱승을 가정하고 곱셈기가 $(n+m)$번의 클럭으로 모듈라 곱셈을 처리할 수 있다면, L-R 방식 멱승기는 평균 $(1.5n+5)(n+m)$번의 클럭만에, R-L 방식 멱승기는 최대 $(n+4)(n+m)$번의 클럭 시간에 멱승을 완전히 처리할 수 있다. 제안된 멱승기를 ALTERA MAX+PLUS II의 그래픽 편집기로 설계하여 시뮬레이션 하였으며 그 동작이 정확하였다.

참 고 문 헌

[1] R. Rivest, A.Shamir and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Comm. of ACM*, Vol. 21, No. 2, pp. 120~126, Feb. 1978.

[2] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inform. Theory*, Vol. 31, No. 4, pp. 469~472, July 1985.

[3] H. Handschuh and P. Paillier, "Smart card crypto-coprocessors for public-key cryptography," *Cryptobytes*, Vol. 4, No. 1, pp. 6~10, 1998.

[4] C. C. Yang, T. S. Chang, and C. W. Jen, "A new RSA cryptosystem hardware design based on Montgomery's algorithm," *IEEE Transactions on Circuits & Systems II-Analog & Digital Signal Processing*, Vol. 45, No. 7, 1998.

[5] J. H. Guo, C. L. Wang, and H. C. Hu, "Design and implementation of an RSA public-key cryptosystem," *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems*, Vol. 1, 1999.

[6] Peter L. Montgomery, "Modular multiplication without trial division," *Math. of Comp.* Vol. 44, No. 170, pp. 519~521, April 1985.

[7] S. R. Dusse and B. S. Kaliski, "A cryptographic library for the Motorola DSP 56000," *Advances in cryptology, Proc. EUROCRYPT '90*, pp. 230~244, 1990.

[8] C. D. Walter, "Systolic Modular Multiplication," *IEEE Trans. on Computers*, Vol. 42, pp. 376~378, 1993.

[9] S. E. Eldridge and C. D. Walter, "Hardware implementation of Montgomery's modular multiplication algorithm," *IEEE Trans. Computers*, Vol. 42, No. 6, pp. 693~699, 1993.

[10] S. E. Eldridge, "A Faster Modular Multiplication Algorithm," *Intern. J. Computer Math.*, Vol. 40, pp. 63~68, 199.

[11] C. Y. Su, S. A. Hwang, P. S. Chen, and C. W. Wu, "An improved Montgomery's algorithm for high-speed RSA public-key cryptosystem," *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, Vol. 7, No. 2 , 1999.

[12] B. Arazi, *Digital Signature Device*, US Patent #5448639, 1995.

[13] ALTERA, *MAX+PLUS II Getting Started*, ALTERA Corp., July, 1998.

[14] D. E. Knuth, *The Art of Computer Programming*, Vol. 2, Seminumerical Algorithms, 2nd Edition, Addison-Wesley, 1981.

[15] C. N. Zhang, "An Improved Binary

Algorithm for RSA," *Computer Math. Applic.* Vol. 25, No. 6, pp. 15~24, 1993.

[16] D. C. Lou and C. C. Chang, "Fast exponentiation method obtained by folding the exponent in half," *Electronics Letters*, Vol. 32, No. 11, pp. 984~985, May, 1996.

[17] J. C. Ha and S. J. Moon, "A Design of Modular Multiplier Based on Multi-precision Carry Save Adder," *Joint Workshop on Information Security and Cryptology (JWISC'2000)*, pp. 45~51, Jan. 2000.

[18] 하재철, 문상재, "*분할형 CSA를 이용한 Montgomery 곱셈기*," 통신정보보호학회 논문지, 제10권, 제3호, 2000. 9

[19] J. J. Leu and A. Y. Wu, "A scalable low-complexity digit-serial VLSI architecture for RSA cryptosystem," *Proceedings of IEEE Workshop on Signal Processing Systems*, pp. 586~595, 1999.

[20] H. Orup, "A 100Kbits/s single chip modular exponentiation processor," *in HOT Chips VI, Symp. Rec.*, pp. 53~59, 1994.

[21] S. Ishii, K. Ohyama, and K. Yamanaka, "A single-chip RSA processor implemented in a 5μm rule gate array," *in Proc. 7th Annu. IEEE Int. ASIC Conf. Exhibit*, pp. 433~436, 1994.

[22] P. S. Chen, S. A. Hwang, and C.W. Wu, "A systolic RSA public key cryptosystem," *in Proc IEEE International Symp. on Circuit and Systems*, Vol. 38, No. 4, April, 1996.

〈著者紹介〉

하 재 철 (JaeCheol Ha) 종신회원

1989년 2월 : 경북대학교 전자공학과 졸업(학사)
1993년 8월 : 경북대학교 대학원 전자공학과 졸업(석사)
1998년 2월 : 경북대학교 대학원 전자공학과 졸업(박사)
1998년 3월~2000년 2월 : 나사렛대학교 전자계산소장
1998년 3월~현재 : 나사렛대학교 전산정보학과 조교수
2000년 3월~현재 : 나사렛대학교 학술정보관장
2001년 2월~현재 : 한국정보보호학회 총무이사
〈관심분야〉 정보보호, 부호이론, 네트워크 보안

문 상 재(SangJae Moon) 종신회원

1972년 2월 : 서울대학교 공업교육(전자)과 졸업(학사)
1974년 2월 : 서울대학교 대학원 전자공학과 졸업(석사)
1984년 6월 : 미국 UCLA 전기공학과 졸업(박사)
1984년 7월~1985년 6월 :UCLA Postdoctoral 근무
1984년 7월~1985년 6월 : 미국 OMNET 컨설턴트
1974년 12월~현재 : 경북대학교 공과대학 전자전기컴퓨터학부 교수
2000년 8월~현재 : 경북대학교 이동네트워크 정보보호기술 연구센터 소장
2001년 2월~현재 : 한국정보보호학회 회장
〈관심분야〉 정보보호, 디지털 통신, 이동 네트워크