通信情報保護學會論文誌
第 10 券 第 3 號, 2000. 9

분할형 CSA를 이용한 Montgomery 곱셈기

하 재 철*, 문 상 재**

The Montgomery Multiplier Using Scalable Carry Save Adder

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

요 약

본 논문에서는 Montgomery 곱셈을 수행하기 위해 작은 크기의 CSA를 반복적으로 사용하는 새로운 모듈라 곱셈기를 제안한다. 이 곱셈기는 설계공간이나 계산 시간에 따라 분할 가능한 형태로 구현할 수 있어 공개 키 암호 시스템에서의 큰 정수의 곱셈을 수행하는데 적응적으로 활용할 수 있다. 논문에서는 워드 단위의 Montgomery 알고리듬을 기술하고 알고리듬에 따른 분할형 CSA를 이용한 곱셈기 설계 기법을 제안한다. 제안한 곱셈기는 IC 카드와 같은 제한된 공간내에서 곱셈의 연산시간을 고려하여 다양하게 설계할 수 있다.

ABSTRACT

This paper presents a new modular multiplier for Montgomery multiplication using iterative small carry save adder. The proposed multiplier is more flexible and suitable for long bit multiplication due to its scalable property according to design area and required computing time. We describe the word-based Montgomery algorithm and design architecture of the multiplier. Our analysis and simulation show that the proposed multiplier provides area/time tradeoffs in limited design area such as IC cards.

keyword : *modular multiplier, Montgomery algorithm, scalable CSA, area/time tradeoffs*

Ⅰ. 서 론

공개 키 암호 시스템에서는 512비트나 1024비트 이상의 큰 정수에 대한 모듈라 멱승(exponentiation) 연산이 필요한데 대부분 수 백번 이상의 모듈라 곱셈을 반복함으로써 처리할 수 있다.[1,2] 특히, 암호 IC 카드와 같은 물리적으로 설계 공간이 제한된 경우에도 실시간 정보보호 서비스를 위해 고속, 저용량의 모듈라 곱셈 회로를 필요로 하는 경우가 있다.

현재까지 개발된 모듈라 곱셈기는 Sedlak, Quisquater, Montgomery 알고리듬 등에 기반하는데 Montgomery 알고리듬은 곱셈기 내부 연산이 규칙적이고 데이터 흐름이 일정한 구조를 가지고 있어 하드웨어 구현이 용이하다.[3~7] Montgomery 알고리듬을 이용한 곱셈기에는 Arazi가 제안한 DSD (Digital Signature Device)와 Walter가 제안한 평면 시스토릭 어레이(systolic array) 등이 있다.[8,9]

Arazi가 제안한 DSD 곱셈기는 가산기(adder), 레지스터(register), 멀티플렉서(MUX)만을 사용함으로써 Montgomery 모듈라 곱셈을 수행할 수 있으나 여기에서는 사용된 구체적인 가산기를 제시하지 않고 있어 캐리 전파(carry propagation) 문제를

* 나사렛대학교 전산정보학과(jcha@dove.nazarene.ac.kr)
** 경북대학교 공과대학 전자전기공학부(sjmoon@ee.knu.ac.kr)

해결했다고 할 수 없다. 한편, 하 등[10,11]은 일반적인 모듈라 곱셈시 발생하는 캐리 전파 문제를 해결하면서 연산 속도를 높이기 위한 곱셈기를 제시한 바 있다.

그러나 지금까지 제시된 대부분의 곱셈기들은 회로 설계에 필요한 하드웨어 용량이 연산하고자 하는 수의 크기에 정비례한 특징을 가지고 있다. 즉, n비트의 곱셈에 필요한 하드웨어는 한 비트 처리를 위한 회로의 n배 정도가 필요한 구조로 되어 있다. 본 논문에서는 이러한 점을 고려하여 연산 비트가 n일 경우 필요한 하드웨어 용량이 한 비트 처리를 위한 회로의 n배보다 훨씬 적게 소요되는 곱셈기를 제안하고자 한다. 결국, 제안 곱셈기는 계산 시간과 회로 복잡도가 상호 완충적인(tradeoffs) 관계를 가지게 되어 하드웨어 구현 조건이 충분하지 않은 시스템에 적응적으로(flexible) 활용할 수 있다.

한편, 본 논문과는 독립적으로 Tenca 등은 분할(scalable) 덧셈을 이용하면서 병렬 처리 기법을 사용한 Montgomery 곱셈기를 제안한 바 있다.[12] 그러나 이 곱셈기는 병렬 처리를 위한 추가적인 회로가 더 필요하고 CSA(Carry Save Adder)를 이용했을 경우 발생하는 캐리와 합(sum)을 보정하는 문제를 제시하지 않고 있다. 본 논문에서는 병렬 처리 기법을 사용하지 않음으로써 회로를 단순화하였으며 CSA를 이용하면서도 최종 결과값을 보정하여 단일 결과를 출력하는 방법을 제안한다.

본 논문의 2장은 비트 단위의 Montgomery 알고리듬을 설명하고 이를 이용한 모듈라 곱셈기의 설계 예를 기술한다. 제 3장에서는 워드 단위의 모듈라 곱셈 알고리듬을 제안하고 이를 기초로 새로운 곱셈기 설계 기법을 설명한다. 제 4장에서는 설계된 회로를 ALTERA MAX+PLUS II[13]로 시뮬레이션 하여 동작의 정확성 및 성능을 검증하고 기존의 곱셈기와 비교 분석한다. 마지막으로 결론을 맺는다.

II. 기본적인 Montgomery 곱셈기

1. Montgomery 알고리듬

공개 키 암호 시스템에 사용하는 모듈라 멱승 $A^E \bmod N$은 모듈라 곱셈의 반복으로 이루어진다. 일반적인 모듈라 곱셈 $AB \bmod N$은 두 수 A, B를 곱한 결과를 N으로 나눈 나머지를 취하는 연산이다. Montgomery 모듈라 곱셈은 $AB \bmod N$가 아닌 $ABR^{-1} \bmod N$을 수행하는데 여기서 R은 N과 서로 소인 N보다 큰 정수이다. 논문에서는 A, B 및 N를 모두 n비트라고 가정한다. 또, 각 정수를 기저(base) r로 표현하면 $A=\sum_{i=0}^{k-1}A[i]r^i$, $B=\sum_{i=0}^{k-1}B[i]r^i$ 및 $N=\sum_{i=0}^{k-1}M[i]r^i$ 과 같이 k자리로 나타낼 수 있는데 $r=2$인 경우에는 k자리는 n비트와 같아진다.

Montgomery 알고리듬은 먼저 N보다 크고 N과 서로 소인 R을 선택하는데 div R이나 mod R을 간단히 계산할 수 있도록 $R=r^n$으로 하는 것이 일반적이다. 본 논문에서는 $r=2$이라 가정하며 Montgomery 곱셈은 $AB2^{-n} \bmod N$으로 구체화 한다. 따라서 Montgomery 모듈라 곱셈 알고리듬은 $r=2$이고, N이 홀수인 경우 그림 1과 같이 간단히 나타낼 수 있다.[5]

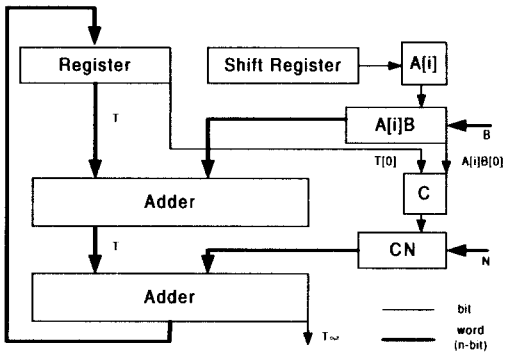
그림에서 단계 2b는 A와 B를 서로 곱하기 위한 반복 과정이며 단계 2c는 모듈라 감소를 위한 반복 단계이다. 또, 단계 2c 및 2d는 $C=0$일 경우에는 T를 단순히 오른쪽으로 쉬프트시키고, $C=1$일 경우에는 T에 N을 더한 후 오른쪽으로 쉬프트시키는 것과 동일하다. 그림 1의 알고리듬은 n번의 루프를 돌면서 각 루프마다 2개의 n비트 가산기를 사용하고 이 출력을 오른쪽으로 쉬프트 시킴으로써 하드웨어로 구현이 가능하다. 이 알고리듬에서 n번째 루프 후의 최종 계산 값은 $T = AB2^{-n} \bmod N$가 됨을 확인할 수 있다.

2. Montgomery 곱셈기의 기본 구조

그림 1의 Montgomery 모듈라 곱셈 알고리듬을 따라 설계한 모듈라 곱셈기 구조를 나타낸 것이 그림 2이다.[10] 이 곱셈기의 첫 번째 가산기는 중간 계산 값을 저장하는 레지스터의 출력 T와 $A[i]B$를 더하는 것이고, 두 번째 가산기는 이전 가산기의 출력과 C값에 따라 CN을 더하는 것이다. 즉, 첫 번째 가산기는 그림 1의 단계 2b를 수행하고, 두 번째 가산기는 그림 1의 단계 2c를 수행한다. 이렇게 해서 얻은

```
1. T = 0
2. i = 0 for to n-1 step 1 {
   2a. C = (T[0] + A[i]B[0]) mod 2
   2b. T = T + A[i]B
   2c. T = T + CN
   2d. T = T/2 }
3. return (T)
```

[그림 1] Montgomery 모듈라 곱셈 알고리듬



〔그림 2〕 Montgomery 곱셈기의 기본 구조

출력의 마지막 비트 T_{out}을 제외하고 레지스터의 입력으로 다시 궤환(feedback)시키게 되는데 이는 오른쪽으로 1비트씩 쉬프트시키는 것과 똑같은 효과를 얻을 수 있어 그림 1의 단계 2d를 수행하게 된다. 이 과정을 피승수 A의 각 비트 $A[i]$에 따라 n번 반복 수행함으로써 최종적으로 $AB2^{-n} \bmod N$을 계산한다.

이 곱셈기 구조에서는 큰 정수에 대한 덧셈을 수행하는 두 개의 가산기가 사용되는데, 만약 CPA (Carry Propagation Adder)를 사용할 경우에는 캐리 전파 문제가 생긴다. 특히, 계산에 사용되는 수가 512비트 이상의 큰 정수인 점을 고려하면 캐리 전파 문제로 인해 CPA를 사용하기는 현실적으로 어렵다. 캐리 전파 문제를 해결하기 위한 방법으로 CSA를 사용할 수 있는데 CSA는 3개의 비트 입력에 대해 이들을 더한 값을 캐리와 합(sum)으로 나누어 2비트로 결과 값을 출력한다. 가산기로 CSA를 사용함으로써 각 비트들을 독립적으로 처리할 수 있어 캐리 전파는 해결할 수 있지만 올바른 최종 계산 값을 얻기 위해서는 캐리와 합을 다시 더해야 한다는 단점이 있다. 그러나 최근 별도의 하드웨어나 추가적인 회로 구현이 없이도 CSA의 캐리와 합을 더하여 보정된 최종 결과를 얻을 수 있는 방법이 하 등[10,11]에 의해 제시된 바 있다.

Ⅲ. 분할 CSA를 이용한 모듈라 곱셈기

1. 워드 단위별 모듈라 곱셈 알고리듬

그러나 위에서 제시한 설계 방법은 가산기의 크기가 연산하고자 하는 수의 크기에 정비례하여 늘어나게 되는 단점이 있다. 즉, n비트의 곱셈에 필요한 하드웨어는 한 비트 처리를 위한 회로의 정확히 n배가 필요한 구조이다. 이러한 곱셈기 구조는 큰 길이의 정수에 대한 곱셈을 수행해야 하지만 설계 공간이 일정하게 제한된 곳에는 실제 활용하기가 어려운 점이 있다. 본 논문에서는 이러한 점을 고려하여 제한된 설계 공간에서도 큰 정수의 곱셈 연산이 가능한 분할 구현이 가능한 곱셈기를 설계하고자 한다.

n비트에 대한 모듈라 곱셈을 가정할 때, 곱셈기의 핵심 회로인 큰 정수에 대한 가산기를 n비트를 처리하는 크기로 설계하는 것이 아니고 n보다 작은 b비트 크기로 설계하면 계산 시간은 늘어나지만 설계에 필요한 가산기는 $m = \lceil n/b \rceil$ 정도로 줄일 수 있다. 단, 여기서 회로의 간소화를 위해 n/b가 정수가 되도록 설계하는 것이 효과적이다. 예를 들어, 512비트의 정수에 대한 가산을 256비트나 128비트, 혹은 64비트씩의 가산기를 사용하여 반복적으로 이용할 수 있도록 하자는 것이다.

이를 위해 먼저 B와 N을 각각 b비트씩 묶어 하나의 워드로 하고 이를 순차적으로 처리한다. 즉, A, B, 그리고 N은 각각 $A = \sum_{i=0}^{m-1} A[i]2^{ib}$, $B = \sum_{i=0}^{m-1} B[i](2^b)^i$ 및 $N = \sum_{i=0}^{m-1} N[i](2^b)^i$로 표현할 수 있다. 여기서 $B[j]$와 $N[j]$는 n비트를 m개의 블록으로 나눈 것으로 가정할 때 j번째의 블록에 있는 b비트로 구성된 계수(coefficient)를 의미한다. 정수 B와 N의 워드 단위별 처리를 위해 그림 1의 Montgomery 알고리듬을 2차원 형태로 다시 표현한 것이 그림 3이다. 그림 3의 2b에서 2d까지의 작은 정수의 가산하는 과정은 각각 그림 1의 2b에서 2d까지의 과정과 대응된다. 그림 3에서의 중요한 특징은 과정 2b에서 발생하는 캐리(D_1)와 2c에서 발생하는 캐리(D_2)가 각각 최대 1비트가 되도록 함으로써 $A[i]B[j]$와 $CN[j]$를 동시에 $T[j]$에 더하는 방식보다 캐리 처리를

```
1.  T = 0
2.  for i = 0 to n-1 step 1 {
2a. C = (T[0] + A[i]B[0]) mod 2
    D1 = D2 = 0
    for j = 0 to m-1 step 1 {
2b  T[j] = (T[j] + A[i]B[j] + D1) mod 2^b
    D1 = (T[j] + A[i]B[j] + D1) / 2^b
2c  T[j] = (T[j] + CN[j] + D2) mod 2^b
    D2 = (T[j] + CN[j] + D2) / 2^b
2d  T[j] = T[j] / 2 }}
3.  return (T)
```

〔그림 3〕 반복적인 워드 단위 가산을 이용한 Montgomery 알고리듬

쉽게 설계하였다. 만약 $A[i]B[j]$와 $CN[j]$를 동시에 $T[j]$에 더한다면 캐리는 최대 2비트가 되어 하드웨어 구현시 캐리 처리가 쉽지 않게 된다.

2. 분할형 CSA를 이용한 곱셈기 구조

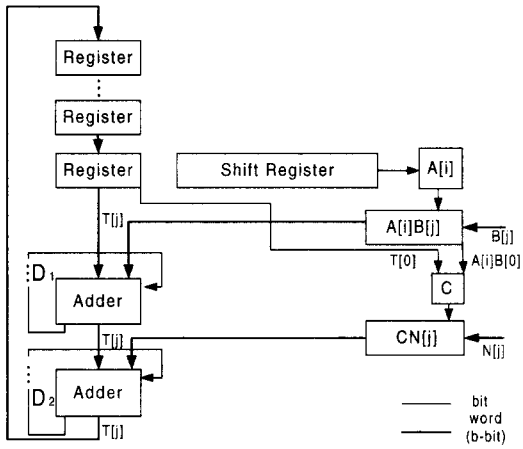
그림 3에 기술한 알고리듬을 구현하는 곱셈기의 블록도를 나타낸 것이 그림 4이다. 그림 2와 그림 4를 비교해 볼 때, 그림 2는 큰 정수에 대한 가산을 할 때 하나의 큰 가산기로 n비트씩 가산을 하는 것에 비해 그림 4는 n보다 작은 b비트씩을 m번 가산하는 구조로 되어 있음을 알 수 있다. 그림에서 T를 저장하기 위한 m개의 레지스터와 가산기는 모두 b비트 즉, 워드 단위로 데이터를 처리한다. 또한, $A[i]$는 비트 단위로 n번 입력되며 그때마다 $B[j]$와 $N[j]$는 워드 단위로 입력된다.

그림 2와 같은 방법으로 가산기를 사용할 경우 한번의 곱셈을 위해 필요한 클럭 수는 n클럭이다. 반면 그림 4의 경우는 최소 $n \cdot m$ 클럭이 필요하고 이는 기존의 것보다 클럭이 m배 늘어나게 될 것이다. 결국, 가산기의 크기는 $1/m$로 줄일 수 있는 반면 필요한 클럭 수는 그 만큼 늘어나므로 상호 완충적인 관계가 있음을 알 수 있다.

Ⅳ. 시뮬레이션 및 고찰

본 논문에서는 큰 정수에 대한 가산을 작은 크기의 가산기를 반복적으로 이용하는 기법을 사용하여 모듈라 곱셈기 회로를 설계하였다. 그리고 VLSI 설계 도구인 ALTERA MAX+PLUS II를 사용하여 schematic 방법으로 시뮬레이션하였다. 여기에서는 회로 설명의 간결성을 위해 8비트의 A, B, N을 사용하였으며, $AB2^{-8}$ mod N을 계산하는 곱셈기의 회로를 ALTERA MAX+PLUS II의 그래픽 편집기(graphic editor)로 설계하였다. 또한, $n=8$비트의 정수 B와 N은 $b=4$비트씩 묶어 $m=2$개의 블록으로 처리하도록 구성하였다.

회로 구현에서 중요한 부분은 그림 4에서 사용된 두 개의 b비트 가산기 회로이다. CPA는 캐리 전파 문제가 생기게 되므로 이러한 큰 정수의 덧셈에는 사용하지 않는 것이 일반적이다. 다른 방법으로 CSA를 사용할 수 있는데 이 가산기는 캐리 전파 문제는 완전히 해결할 수 있는 장점에 비해 그 결과가 하나의 결과 형태로 나오는 것이 아니라 캐리와 합이 독립적으로 출력되는 단점이 있었다. 그러나 멱승 연산과



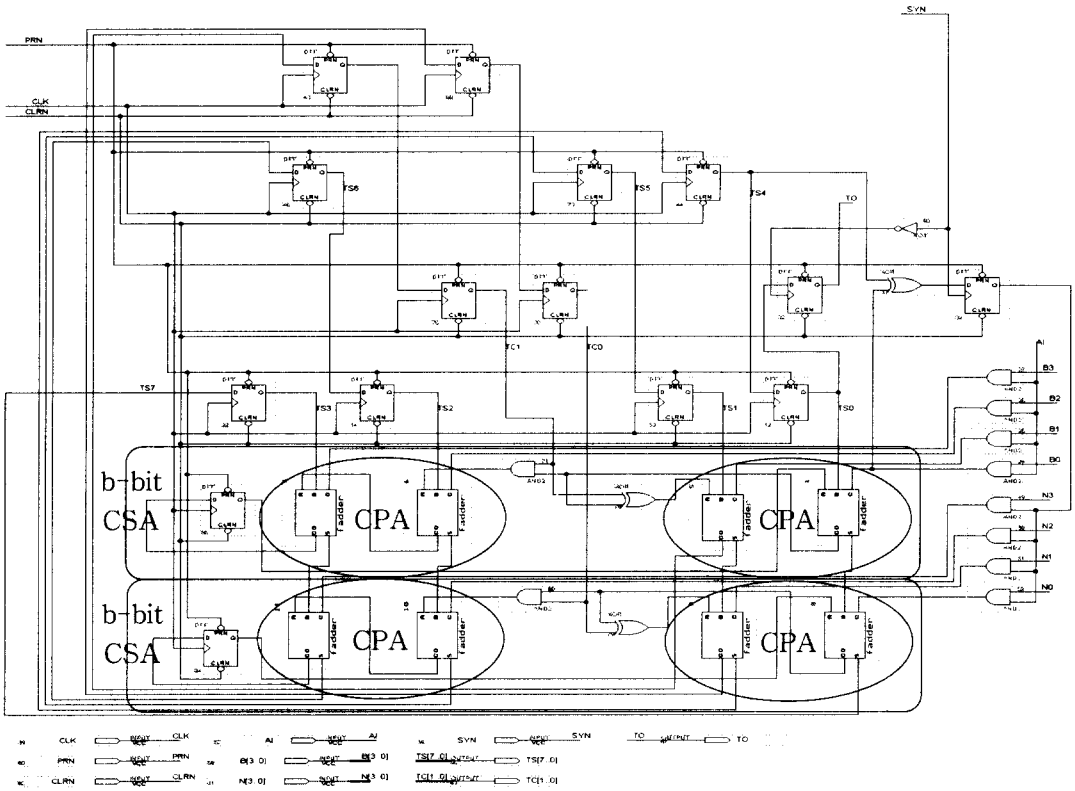
〔그림 4〕 제안하는 Montgomery 곱셈기 구조

같이 반복적인 곱셈이 필요한 경우에는 한번의 곱셈이 끝나게 되면 최종 결과는 캐리와 합의 형태가 아니라 두 결과를 더한 단일 결과로 만들어야 한다.

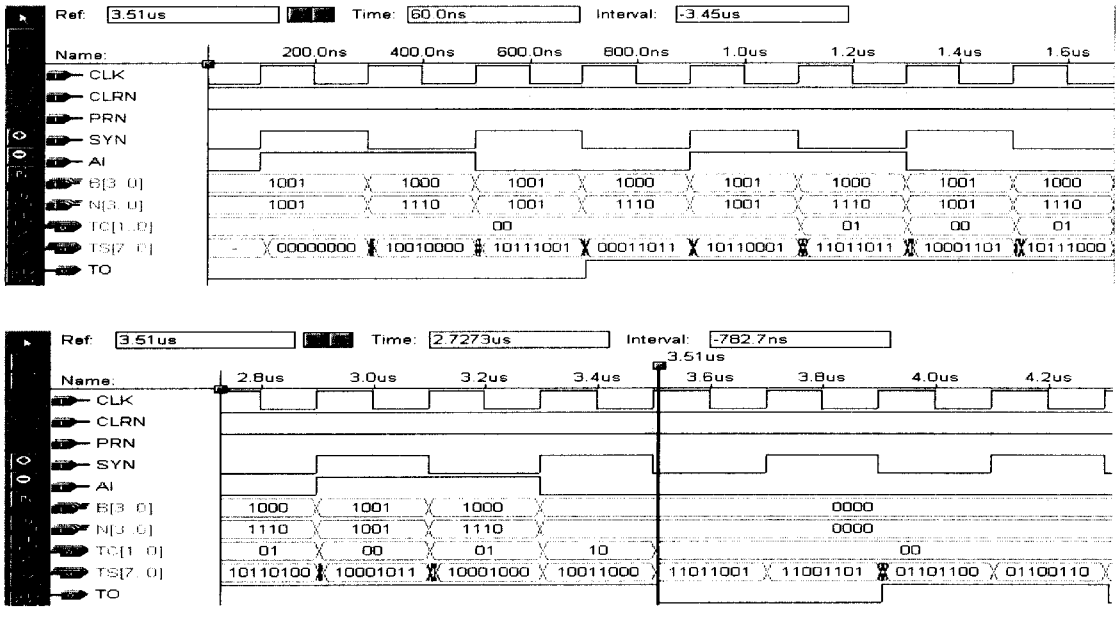
단일한 결과를 출력하기 위해서는 별도의 회로와 클럭이 필요한데 문헌 〔10〕과 〔11〕에서는 추가적인 회로 없이도 약간의 클럭만 더 인가하여 보정된 최종 결과를 얻을 수 있는 방법을 제안한 바 있다. 좀 더 구체적으로 설명하면 b비트의 가산기를 설계할 경우에도 b비트를 다시 $c=\lceil b/d \rceil$로 나누어 캐리 전파가 전체 회로에 영향을 주지 않을 크기의 d비트 CPA를 c개 설계한다. 그리고 d비트씩 묶은 c개의 CPA는 CSA 형태로 연결하는 기법이다. 본 논문에서는 $c=2$ 그리고 $d=2$로 간소화하여 시뮬레이션 결과를 표시하였다. 물론 추가적인 하드웨어 없이 최종 결과를 얻기 위해서는 m개의 블럭당 c번의 클럭이 더 필요하게 된다. 자세한 내용은 문헌 〔10〕과 〔11〕에 나타나 있으며 최종 결과 회로를 나타낸 것이 그림 5이다.

예로서 $A=133=10000101_2$, $B=137=10001001_2$, $N=233=11101001_2$로 하여 시뮬레이션 한 그 결과 파형은 그림 6과 같다. 이 경우 R^{-1} mod $N=2^{-8}$ mod $233=152$이며, ABR^{-1} mod N에 대한 이론값은 $133 \cdot 137 \cdot 152$ mod $233=154$이다. 그림에서 CLK는 레지스터에 인가되는 클럭으로서 주파수를 5MHz(한 클럭당 200 ns)로 하였으며, AI는 A의 각 비트를 나타낸다.

그림 6에서 보면 $n \cdot m$번째 클럭, 즉, 16번째 클럭이 끝나는 시점(3.3μs 바로 다음)에서의 최종 결과는 합(TS로 표기)이 10011000이고 캐리(TC로 표기)가



〔그림 5〕 분할형 CSA를 이용한 Montgomery 곱셈기



〔그림 6〕 시뮬레이션 결과 파형

10이므로 최종 결과는 이들을 합한 $10011010_2 = 154$이 되어 이론 값과 같음을 확인할 수 있다. 그러나 실제 하드웨어 회로를 이용하여 합과 캐리를 더하는 연산은 c번의 클럭(시뮬레이션에서는 $c=2$)이 더 필요하게 되며 이때 추가되는 c클럭 동안에는 A, B 그리고 N은 모두 0이 입력된다. 추가되는 c클럭 동안 출력의 최하위 비트는 그림의 TO 단자를 통해 한 비트씩 출력된다. 본 논문에서는 총 $(n+c)\cdot m = (8+2)\cdot 2 = 20$번의 클럭이 지난 4.1$\mu s$ 바로 다음에는 합과 캐리가 모두 더해져서 결과는 TO를 통해 하위 두 비트 그리고 $TS0 \sim TS5$까지의 6비트를 연접한 결과가 된다.

표 1은 기존의 DSD 곱셈기, n비트 다정도 CSA를 이용한 곱셈기[10,11] 그리고 제안한 반복 가산을 이용한 곱셈기를 비교한 것이다. 단, 이 표에서는 회로 개선도를 비교하기 위해 세 방식에서 공통적으로 사용하는 A, B, 그리고 N을 저장하기 위한 레지스터는 고려하지 않았다. DSD의 경우 수행시간이 $n+1$클럭이라고 표기하였으나 이는 DSD에 CPA를 사용했을 경우를 가정한 것인데 이것은 큰 정수의 연산에서는 이용하기 힘들다고 볼 수 있다. CSA를 사용할 경우에는 지적한 바와 같이 캐리와 합의 결과를 올바른 값으로 보정하기 위한 회로 장치가 필요하지만 구체적인 방법은 제시하지 않고 있다. n-비트 다정도 CSA를 이용한 곱셈기에서 k는 CPA를 가지는 블록의 개수를 나타내는데 이 곱셈기는 하드웨어 구성 요소 중 레지스터의 용량이 많지 않으며 MUX를 사용하지 않으므로 DSD에 비해 집적 논리 게이트를 많이 줄일 수 있다. D 플립플롭은 5개의 기본 논리 게이트, 전가산기는 5개의 게이트 그리고 4:1 MUX는 10개의 게이트가 필요하다고 가정하고 회로 개선도를 비교해 보자. 각 방식에 따라 구현에 필요한 게이트 산출 방식은 다음과 같다.

① DSD :
D 플립플롭$(3n)$+전가산기(n)+MUX(n)
$=(15+5+10)n=30n$ 게이트

② n-비트 CSA 곱셈기 :
D 플립플롭$(n+2k)$+전가산기$(2n)$+
AND/XOR$(2n+4k)$
$=(5+10+2)n+(10+4)k=17n+14k$ 게이트

③ 제안 곱셈기 :
D 플립플롭$(n+2c)$+전가산기$(2n/m)$+
AND/XOR$(2n/m+4c)$
$=5n+10n/m+2n/m+14c$
$=5n+12n/m+14c$ 게이트
(여기서 $b=n/m$, $c=b/d$이며 특히, $m=1$인 경우 $c=k$로 n-비트 CSA 곱셈기와 동일)

〔표 1〕 기존 곱셈기와 제안 곱셈기의 비교

		DSD	다정도 CSA 곱셈기	제안 곱셈기
수행시간(클럭)		$n+1$ (캐리 전파)	$n+k$ (k:CPA수)	$(n+c)\cdot m$ (m:분할수 $m=n/b$) (c:CPA수 $c=b/d$)
하드웨어 구성 요소	레지스터 (D플립플롭)*	$3n$	$n+2k$	$n+2c$
	전가산기	n	$2n$	$2n/m$
	4:1 MUX	n	0	0
	AND/XOR	0	$2n+4k$	$2n/m + 4c$
	합계 (기본게이트)	$30n$	$17n+14k$	$5n+12n/m+14c$

* : A, B, N을 저장하는 레지스터는 모두 공통적으로 동일함

위의 산출식에서 보는 바와 같이 제안 곱셈기는 DSD나 n-비트 다정도 CSA를 이용하여 구현하는 것보다 회로를 설계하는데 필요한 게이트가 적음을 알 수 있다. 만약 제안 곱셈 방식에서 가산기를 그대로 n비트 크기로 설계한다면 $m=1$이 되고 $c=k$가 되어 다정도 CSA를 이용하는 경우와 같아진다. 만약 가산기를 $1/m$로 분할한 것을 사용하게 되면 레지스터로 사용되는 D 플립플롭을 제외한 나머지 부분의 게이트 수는 n-비트 CSA 곱셈기보다 약 1/m로 줄어든다. 이에 반해 제안 곱셈기의 곱셈 계산 시간은 약 m배 정도 늘어남을 알 수 있다.

구체적인 예를 들어보자. 주 클럭을 5MHz로 하는 시스템을 사용한다면 한 클럭 시간은 200ns이다. 또한 256비트용 가산기를 사용하여 1024 모듈라 곱셈을 계산한다고 하고 또한, 256비트 가산기는 32비트를 하나의 CPA로 사용한다고 가정하자. 즉, $n=1024$, $b=256$, $m=4$, $d=32$, 그리고 $c=8$이다. 따라서, 한번 모듈라 곱셈의 시간은 $(n+c)\cdot m\cdot 200ns = 825\mu s$이며 평균 멱승 속도는 약 $1.5n\cdot 825\mu s = 1.27s$가 되므로 실시간 처리에는 크게 무리가 없다.

V. 결 론

본 논문에서는 정보보호 서비스를 제공할 경우에 필수적인 모듈라 멱승을 고속으로 처리하기 위해 새로운 모듈라 곱셈기 회로를 제안하였다. 제안한 곱셈기의 중요한 특징은 제한된 설계 공간 문제를 해결하기 위해 큰 정수에 대한 가산 회로를 $1/m$로 축소하여 설계하고 이 회로를 반복적으로 사용하도록 하였다. 실제로 사용하는 가산기는 캐리 전파 문제를 해결하기 위해 CSA를 사용할 수도 있으나 이는 캐리와 합을 보정한 결과로 출력되어야 한다. 본 논문에서는 한 클럭 동안에서만 캐리 전파가 일어나는 CPA와 CSA의 특징을 혼합한 가산기를 사용하으로써 캐리 전파 문제를 해결하였다. 제안 곱셈기는 n비트 모듈라 곱셈 연산을 할 경우 $(n+c)\cdot m$ 클럭만에 완전한 곱셈 결과를 얻을 수 있으므로 계산 속도는 다소 늦어질 수 있으나 다양한 설계 기준을 채택할 수 있어 회로 설계 조건이 제한적인 암호 IC 카드와 같은 응용 분야에 유용하게 사용될 수 있다.

참 고 문 헌

[1] R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public key crypto-systems," *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] P. L. Montgomery, "Modular multiplication without trial division," *Math. of Comp.* Vol. 44, No. 170, pp. 519~52, April 1985.

[4] H. Sedlak, "The RSA cryptography processor," *In Advances in Cryptology - EUROCRYPT '87*, Springer-Verlag, pp. 95~105, 1988.

[5] S. E. Eldridge, "A Faster Modular Multiplication Algorithm," *Intern. J. Computer Math.*, Vol. 40, pp. 63-68, 1991.

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

[7] D. Naccache, D. M'Raihi, "Cryptographic smart cards," *IEEE Micro*, pp. 14~24, June, 1996.

[8] C. D. Walter, "Systolic modular multiplication," *IEEE transactions on computers*, Vol. 42, pp. 376~378, Mar. 1993.

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

[10] 허준희, 하재철, 문상재, "*다정도 CSA를 이용한 고속 모듈라 곱셈기*," 한국 통신정보보호학회 학술대회, Vol. 9, No. 1, pp. 541-55, 1999. 4.

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

[12] A. F. Tenca and C. K. Koc, "A Scalable Architecture for Montgomery Multiplication," *Cryptographic Hardware and Embedded Systems-CHES'99*, Springer-Verlag, pp. 94~108, 1999.

[13] ALTERA, *MAX+PLUS II Getting Started*, ALTERA Corp., Nov. 1995.

〈著者紹介〉

하 재 철 (Jae-Cheol 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
〈관심분야〉 정보보호, 부호이론, 네트워크 보안

문 상 재 (Sang-Jae Moon) 종신회원

1972년 2월 : 서울대학교 공업교육과 졸업(학사)
1974년 2월 : 서울대학교 대학원 전자공학과 졸업(석사)
1984년 6월 : 미국 UCLA 통신공학과 졸업(박사)
1984년 7월 ~1985년 6월 :UCLA Postdoctor 근무
1984년 7월 ~1985년 6월 : 미국 OMNET 컨설턴트
1974년 12월 ~ 현재 : 경북대학교 공과대학 전자전기공학부 교수
〈관심분야〉 정보보호, 디지털 통신, 이동 네트워크