通信情報保護學會論文誌
第9卷 第1號, 1999. 3

시스토릭 어레이를 이용한 Montgomery 모듈라 곱셈기 설계

하 재철*, 문 상재**

(Design of Montgomery Modular Multiplier based on Systolic Array)

Jae-cheol Ha,* Sang-Jae Moon**

요 약

공개 키 암호 시스템에서의 주 연산은 멱승 연산이며 이는 모듈라 곱셈의 반복으로 이루어져 있다. 본 논문에서는 고속 모듈라 곱셈을 위해 Montgomery 알고리듬에 기반한 선형 시스토릭 어레이 곱셈기를 제안하고 이를 설계하였다. 제안 곱셈기는 각 처리기 내부 구조를 간소화할 수 있어 기존 곱셈기에 비해 하드웨어 설계에 필요한 논리 게이트를 약 14%정도 줄일 수 있을 뿐만 아니라 모듈라 곱셈 속도를 약 20%정도 감소시킬 수 있다.

Abstract

Most public key cryptosystems are constructed based on a modular exponentiation, which is further decomposed into a series of modular multiplications. We design a new systolic array multiplier to speed up modular multiplication using Montgomery algorithm. This multiplier with simple circuit for each processing element will save about 14% logic gates of hardware and 20% execution time compared with previous one.

Key word : 공개 키 암호, 모듈라 곱셈기, 시스토릭 어레이

Ⅰ. 서 론

정보의 암호화나 디지털 서명 등을 위해 사용하는 공개 키 암호 시스템(public key cryptosystem)의 주 연산은 512비트 이상의 큰 정수에 대한 모듈라 멱승(modular exponentiation)이다[1, 2]. 그러나 멱승 연산은 계산 시간이 많이 소요되므로 처리 속도의 지

* 나사렛대학교 전산정보학과
** 경북대학교 전자전기공학부

연이 공개 키 암호 시스템의 문제가 되고 있다. 이를 해결하기 위해 고속 모듈라 멱승 및 곱셈 알고리듬들이 개발되었으며 모듈라 연산만을 위한 전용 칩도 개발되고 있다[3~6]. 멱승 연산은 AB mod N과 같은 모듈라 곱셈의 반복 수행으로 구현할 수 있으므로 멱승의 고속화를 위해 모듈라 곱셈기 자체를 고속으로 구현하는 것이 효과적인 대안이다.

고속 모듈라 곱셈을 위해서는 고전적인 방법[7], Barret 알고리듬[8] 그리고 Montgomery 알고리듬[9] 등이 사용된다. 이 중 Montgomery 알고리듬은 곱셈기의 내부 연산이 규칙적이고 데이타 흐름이 일정한 구조를 가지고 있어 이 알고리듬을 이용한 소프트웨어 구현은 물론 VLSI 하드웨어인 시스토릭 어레이(systolic array)를 설계하는 연구가 많이 있었다[10~12]. Sauerbrey는 Montgomery 모듈라 곱셈 알고리듬에 근거하여 두 개의 시스토릭 어레이를 사용한 멱승기를 설계하였으며[13], Iwamura 등은 Montgomery 모듈라 곱셈을 수행하는 일차원 시스토릭 어레이를 설계하였다[14]. 그러나 이 두 가지 방법은 처리기 구조가 복잡하고 많은 입력 채널이 필요하다는 단점이 있다. 한편, Walter에 의해 제안된 평면 시스토릭 어레이는 여러 모듈라 곱셈 알고리듬 중 Montgomery 알고리듬만이 가지고 있는 규칙성과 병렬성을 잘 이용하고 있다. 특히, 김 등[15]은 Walter의 평면형 곱셈기를 선형으로 설계하여 하드웨어 게이트 수를 크게 감소시킬 수 있는 방안을 제시한 바 있다.

본 논문에서는 Walter 그리고 김 등이 제안한 Montgomery 모듈라 곱셈기를 분석하고 이를 개선한 새로운 곱셈기를 선형 시스토릭 어레이로 설계한다. 기존 곱셈기를 분석한 결과 모듈라 곱셈시 발생하는 carry를 효과적으로 처리하지 않고 있는 점에 기초하여 곱셈기의 carry 처리 방식을 변형하였다. 제안한 곱셈기는 기존 곱셈기에 비해 하드웨어 부담을 약 14%정도 감소시킬 수 있으며 모듈라 곱셈 시간을 20%정도 개선할 수 있다. 본 논문의 2장은 Montgomery 알고리듬을 분석하고 3장에서 Walter의 시스토릭 어레이를 설계하는 기법을 설명한다. 4장에서는 곱셈기내의 처리기에 필요한 게이트를 줄일 수 있는 기본 개념과 연산 속도 개선을 위한 기법을 설명하고 기존 곱셈기와 비교 검토한다. 마지막으로 5장에서 결론을 맺는다.

Ⅱ. Montgomery 알고리듬 분석

일반적으로 n비트라고 가정한 수 A, B 그리고 N을 기수(radix) r로 표현할 때 $A=\sum_{i=0}^{k-1} A[i]r^i$, $B=\sum_{i=0}^{k-1} B[i]r^i$ 및 $N=\sum_{i=0}^{k-1} N[i]r^i$와 같이 k자리로 나타낼 수 있다. 단, 여기서 A와 B는 N보다 작다고 가정한다. 일반적으로 모듈라 곱셈 AB mod N은 A와 B를 곱한 후 몫 추정 기법에 의한 나눗셈을 수행한 뒤 그 나머지를 취하는 방법을 사용하는데 고전적인 방법과 Barret 알고리듬이 대표적이다. 그러나 이 방법들은 나눗셈을 이용하므로 몫 추정 과정이 복잡하다는 점과 하드웨어 구현이 어려운 단점을 지니고 있다. 반면 Montgomery 알고리듬은 모듈라 감소(modular reduction)시 반복적인 나눗셈을 하는 대신 모듈라 N잉여 집합(residue class modulo N)을 N-residue로 정의하고 이 집합내에서 덧셈과 곱셈만으로 빠른 모듈라 감소를 수행한다.

Montgomery 모듈라 곱셈 알고리듬에서는 먼저 N보다 크고 N과 서로 소(relative prime)인 R을 정의하는데 연산기의 처리 단위를 고려하여 mod R과 div R이 용이하도록 일반적으로 $R=r^k$을 사용한다. 단, 여기서 $r=2$일때는 $R=r^k=r^n$이다. 또한 이 알고리듬에서는 임의의 수 x에 대한 N-residue를 xR mod N으로 정의하며, 주된 연산은 N-residue로 변환된 A와 B에 대한 모듈라 곱셈 ABR^{-1} mod N을 수행하

는 것이다. 여기서 AB는 단순히 두 수의 곱셈 과정이며 이 결과에 대해 R^{-1}를 곱하는 과정이 모듈라 감소에 해당한다.

Montgomery 알고리듬 $ABR^{-1} \bmod N$을 수행하는 방법은 크게 두 가지로 나눌 수 있는데 그 하나는 곱셈만을 먼저 수행한 후 모듈라 감소를 수행하는 방법이며, 다른 하나는 곱셈과 모듈라 감소를 동시에 수행하는 방법이다[4]. 첫번째 방법을 알고리듬으로 나타내면 그림 1과 같으며 단계 2와 3이 곱셈을 그리고 단계 4에서 6이 모듈라 감소를 수행하는 부분이다. 단, 여기서 $n_0 = -N[0]^{-1} \bmod r$이며 extended Euclidean 알고리듬으로 사전에 계산한다고 가정한다[7].

MMA_1 (A, B, N, r)
1. $T=0$
2. for $i=0$ to $k-1$ step 1 {
3. $\quad T=T+A[i]B$ }
4. for $i=0$ to $k-1$ step 1 {
5. $\quad M[i]=T[0]n_0 \bmod r$
6. $\quad T=(T+M[i]N)$ div r }
7. if$(T \geq N)$ $T-N$
8. return (T)

그림 1. 곱셈과 모듈라 감소가 분리된 Montgomery 알고리듬
Fig. 1. Montgomery algorithm with distinctive multiplication and reduction.

그림 1의 모듈라 곱셈 알고리듬을 수행하는 과정에서 발생하는 carry는 두 종류로 나눌 수 있는데 단계 3의 곱셈 $T=T+A[i]B$를 수행할 때 발생하는 곱셈용 carry와 단계 6의 모듈라 감소 $T=(T+M[i]N)$ div r을 수행할 때 발생하는 모듈라 감소용 carry가 있다. 이 알고리듬의 단점은 곱셈과 모듈라 감소가 분리되어 있어 큰 길이의 저장장소가 필요하며 하드웨어 구현이 복잡하다는 것이다.

Walter는 이 단점을 없애기 위해 곱셈 및 모듈라 감소를 동시에 수행하도록 Montgomery 알고리듬을 변형하였으며 이에 근거하여 시스토릭 어레이로 곱셈기를 설계하였다[11]. Walter가 제시한 알고리듬을 요약하면 그림 2와 같은데 단계 4의 $T=(T+A[i]B+M[i]N)$ div r에서 보는 바와 같이 곱셈 및 모듈라 감소를 동시에 수행하고 있다. 이 알고리듬에서 곱셈용 carry와 모듈라 감소용 carry는 혼합되어 동시에 처리되고 있으며 단계 5이전의 T값은 최대 $T<2N$가 되어 T가 N보다 클 경우에는 $T-N$을 수행해야 한다.

MMA_2 (A, B, N, r)
1. $T=0$
2. for $i=0$ to $k-1$ step 1 {
3. $\quad M[i]=((T[0]+A[i]B[0])n_0)$ mod r
4. $\quad T=(T+A[i]B+M[i]N)$ div r}
5. if$(T \geq N)$ $T-N$
6. return(T)

그림 2. 곱셈과 모듈라 감소를 동시에 수행하는 Montgomery 알고리듬
Fig. 2. Montgomery algorithm with integrated multiplication and reduction.

Walter 알고리듬 MMA_2에서는 A와 M을 인덱스 i로 하여 1차원 구조로 두고 기술하였으나 B와 N을 인덱스 j로 하여 2차원으로 확장할 수 있다[11]. 또한 비트 단위의 하드웨어 구현을 위해 $r=2$로 하면 자리 수 k는 비트 수 n과 같으며 N이 홀수인 경우에는 $n_0[0, 0] = -N[0]^{-1} \bmod r$는 항상 1이 되어 다른 값과 곱하는 연산은 생략할 수 있다. 알고리듬 MMA_2를 2차원 공간으로 확장하여 비트 단위로 수행되도록 기술하면 그림 3과 같다[12, 15].

$MMA_3(A,B,N,r)$
1. for $i=0$ to n step 1
2. for $j=0$ to n step 1
3. if $(j=0)$ {
4. $M[i, j]=((T[i, j]+A[i, j]B[0, j])n_0[0, 0])$ mod r
5. $C_0[i, j+1]=(T[i, j]+A[i, j]B[i,j]+M[i, j]N[i, j])$ div r mod r
6. $C_1[i, j+1]=(T[i, j]+A[i,j]B[i,j]+M[i, j]N[i, j])$ div r div r
7. M$[i, j+1]$=M$[i, j]$
8. else {
9. $C_0[i, j+1]=(T[i, j]+A[i, j]B[i, j]+M[i, j]N[i, j]+C_0[i, j]+C_1[i, j]r)$ div r mod r
10. $C_1[i, j+1]=(T[i, j]+A[i, j]B[i, j]+M[i, j]N[i, j]+C_0[i, j]+C_1[i, j]r)$ div r div r
11. $T[i+1, j-1]=(T[i, j]+A[i, j]B[i, j]+M[i, j]N[i, j]+C_0[i, j])$ mod r
12. $M[i, j+1]=M[i, j]$
13. $A[i, j+1]=A[i, j]$
14. $B[i+1, j]=B[i, j]$
15. $N[i+1, j]=N[i, j]$
16. }

그림 3. 2차원 Montgomery 알고리듬
Fig. 3. Two-dimensional Montgomery algorithm.

여기서 $A[i, j]$는 A값의 i번째 인덱스 및 j번째 자리 값을 표시하는 2차원 인덱스 점(index point)을 의미한다. 한편, Walter는 그림 2의 단계 5를 하드웨어로 쉽게 구현하는 방법으로 $A[n]=0$를 A에 덧붙여 모듈라 감소를 한번 더 수행함으로써 자연스럽게 $T<N$이 되게 하여 다음 입력으로 사용할 수 있도록 하였다. 그러므로 곱셈기의 최종 결과는$ABr^{-(n+1)}$ mod N이 되고 이런 이유로 원래의 Montgomery 알고리듬에서 $R=r^n$를 잉여류 변환에 사용한 것과 달리 $R=r^{(n+1)}$를 사용한다.

MMA_3 알고리듬에 사용되는 2차원 배열 변수 A, B, N 및 T의 초기값은 다음과 같다.

$T[i, j]=0$ $(0\le i\le n,\ 0\le j\le n)$
$A[i, 0]=A[i]$ $(0\le i<n)$, $A[n, 0]=0$
$B[0, j]=B[j]$ $(0\le j<n)$, $B[0, n]=0$
$N[0, j]=N[j]$ $(0\le j<n)$, $N[0, n]=0$

이 알고리듬에서 최종 결과는 $T[n+1,0]$에서 $T[n+1, n-1]$까지의 결과이다. 그림 3의 알고리듬에서 특히 주의할 점은 처리하는 carry가 최대 두 자리가 될 수 있다는 것이다. 그 이유는 앞서 기술한 바와 같이 곱셈용 carry와 모듈라 감소용 carry가 동시에 처리되므로 단계 11과 같이 $T[i, j]+A[i, j]B[i, j]+M[i, j]N[i, j]+C_0$ $[i, j]$의 연산 결과가 최대 3비트가 될 수 있기 때문이다. 여기서 T의 하위 carry는 C_0로 표시하였고 상위 carry는 C_1으로 나타내었다.

Ⅲ. 시스토릭 어레이 설계

시스토릭 어레이는 여러 개의 처리기가 망 구조로 연결되어 데이타를 규칙적으로 입력받

이를 처리한 후 이웃 처리기로 전달하는 형태이다. 각 처리기들은 정규적으로 흐르는 데이타를 순차적으로 처리하며 최종 처리기에서 모듈라 곱셈 결과를 출력하게 된다. 그림 3의 알고리듬 MMA_3를 구현하기 위한 종속 그래프(dependence graph)를 나타내면 그림 4와 같다. 여기서 종속 그래프의 계산 공간을 좌표 $[i, j]^T$로 나타낼 때 좌표가 $[i, 0]^T$인 오른쪽의 빗금친 계산점들은 MMA_3 알고리듬의 단계 4에서 7까지의 $j=0$인 부분이 수행되는 처리기이며, 그 나머지 계산점은 단계 9에서 15까지의 $j\neq 0$인 부분이 수행되는 처리기이다.

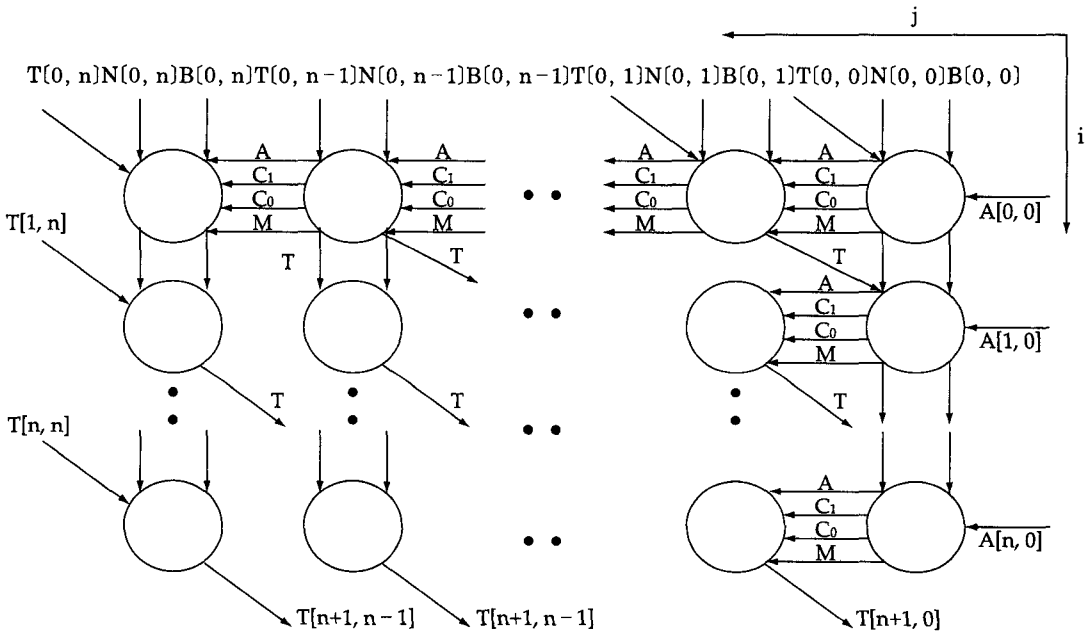


그림 4. 곱셈기의 종속 그래프
Fig. 4. Dependance Graph of the multiplier.

각 데이터의 입출력 과정 및 흐름을 살펴보면, 데이타 $A[i, j]$는 $[i, 0]^T$에서 입력되고 $[0, 1]^T$ 방향으로 이동하며 $M[i, j]$ 값은 제일 오른쪽 처리기에서만 계산되어 $[0, 1]^T$ 방향으로 흐른다. 두 개의 carry값 C_0, C_1은 모든 계산점에서 계산되어 $[0, 1]^T$방향으로 전달된다. $B[i, j]$와 $N[i, j]$는 계산점 $[0, j]^T$에서 입력되고 $[1, 0]^T$ 방향으로 흐른다. 그리고 $T[i, j]$값은 $[0, j]^T$ 및 $[i, n]^T$ 의 계산점에서 입력되어 방향벡터 $[1, -1]^T$ 로 전달된다. 또 이 시스토릭 어레이 곱셈기에서 계산점$[i, j]^T$와 $[i+1, j+2]T$인 처리기는 병렬로 동시에 데이타를 처리한다. 이 곱셈기는 모두 $(n+1)(n+1)$개의 처리기가 사용되며 $(2n+2)t_w$초 후에 최하위 비트가 출력되고 $(2n+2+n)t_w$초 후에 모든 결과가 출력된다. 단, 여기서 t_w는 하나의 처리기를 통과하는데 소요된 시간을 의미한다.

한편, 각 처리기의 내부 구조는 알고리듬 MMA_3를 논리식으로 표현하여 설계할 수 있다. 즉, 알고리듬 MMA_3 중에서 $j\neq 0$인 단계 9에서 11까지를 구하는 논리식을 구하면 다음과 같다. 물론 $j=0$일 때의 오른쪽의 처리기도 같은 방법으로 논리식을 구하면 더 간단히 설계할 수 있다.

$$C_0[i, j+1] = ((C_0[i, j] \wedge T[i, j]) \vee C_1[i, j]) \oplus (M[i, j] \wedge N[i, j] \wedge A[i, j] \wedge B[i, j]) \oplus ((T[i, j] \oplus C_0[i, j]) \wedge (M[i, j] \wedge N[i, j] \oplus A[i, j] \wedge B[i, j]))$$

$$C_1[i, j+1] = (((C_0[i, j] \wedge T[i, j]) \vee C_1[i, j]) \oplus (M[i, j] \wedge N[i, j] \wedge A[i, j]) \wedge B[i, j]) \wedge ((T[i, j] \oplus C_0[i, j]) \wedge (M[i, j] \wedge N[i, j] \oplus A[i, j] \wedge B[i, j])) \vee (((C_0[i, j] \wedge T[i, j]) \vee C_1[i, j]) \wedge (M[i, j] \wedge N[i, j] \wedge A[i, j] \wedge B[i, j]))$$

$$T[i+1, j-1] = T[i, j] \oplus (A[i, j] \wedge B[i, j]) \oplus (M[i, j] \wedge N[i, j]) \oplus C_0[i, j]$$

각 처리기를 논리 게이트로 나타낸 것이 그림 5인데 하나의 처리기에는 모두 14개의 논리 게이트가 필요하다. 또 여기서 고려할 점은 하나의 처리기가 통과하는데 소요되는 시간이다. 만약 각 게이트가 비슷한 응답시간을 가진다고 가정하면 그림 5에 나타낸 바와 같이 XOR 게이트 1개, OR 게이트 그리고 AND 게이트 2개씩을 통과해야 한 처리기의 완전한 결과가 출력됨을 알 수 있다[11].

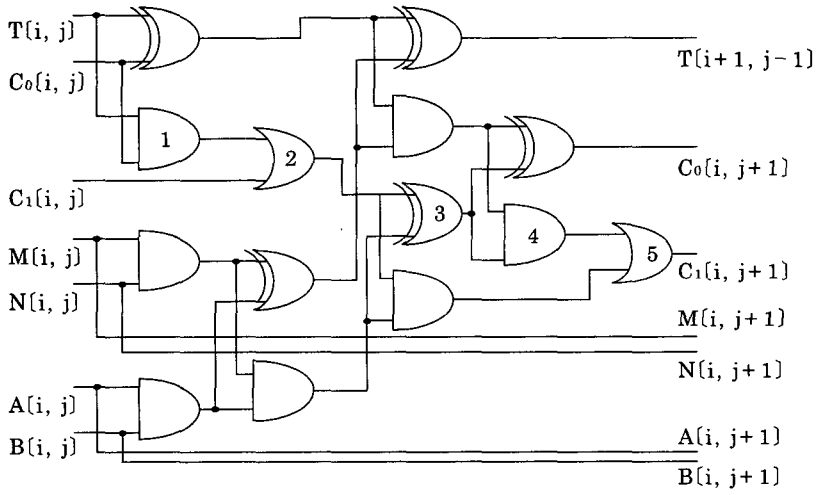


그림 5. 처리기의 내부 회로
Fig. 5. Circuit of the typical processing element.

Ⅳ. 새로운 모듈라 곱셈기

지금까지 기술한 Montgomery 알고리듬을 이용한 시스토릭 모듈라 곱셈기는 곱셈과 모듈라 감소를 동시에 처리하도록 하였다. 즉, 그림 3의 단계 9부터 11에서 보는 바와 같이 $T[i, j]$에 $A[i, j]B[i, j]$를 더하는 과정이 순수하게 곱셈을 하는 부분이며 $M[i, j]N[i, j]$를 더하는 과정이 모듈라 감소를 하는 부분이다. 기존의 곱셈기는 여기에서 발생하는 각각의 carry를 한꺼번에 처리하였는데 그 결과, 곱셈용 carry 와 모듈라 감소용 carry가 구분되지 않고 상위 두 자리 carry가 발생하게 되었다. 그에 따라 처리기 구조도 그림 5와 같이 복잡하게 설계되었다.

그러나 본 논문에서는 이를 개선하여 곱셈시 발생하는 carry와 모듈라 감소시 발생하는 carry를 각각 분리함으로써 각 중간 결과값의 carry가 한자리씩만 발생하도록 하고자 한다. 즉, 곱셈용 carry와 모듈라 감소용 carry를 구

분하여 한 자리씩만 발생하면 처리기 구조를 간소화할 수 있다.

이 개념에 기초하여 제안하는 2차원 모듈라 곱셈 알고리듬은 그림 6과 같다. 단, 여기서 C_M은 곱셈용 carry를 그리고 C_R은 모듈라 감소용 carry를 의미하며 그 역할은 서로 독립적이

$MMA_4(A,B,N,r)$

1. for $i=0$ to n step 1
2. for $j=0$ to n step 1 {
3. if $(j=0)$ {
4. $M[i, j] = ((T[i, j]+A[i, j]B[0, j])n_0[0, 0])$ mod r
5. $C_M[i, j+1] = (T[i, j]+A[i, j]B[i, j])$ div r
6. $T[i, j] = (T[i, j]+A[i, j]B[i, j])$ mod r
7. $C_R[i, j+1] = (T[i, j]+M[i, j]N[i, j])$ div r
8. $M[i, j+1] = M[i, j]$ }
9. else {
10. $C_M[i, j+1] = (T[i, j]+A[i, j]B[i, j]+C_M[i, j])$ div r
11. $T_m[i, j] = (T[i, j]+A[i, j]B[i, j]+C_M[i, j])$ mod r
12. $C_R[i, j+1] = (T_m[i, j]+M[i, j]N[i, j]+C_R[i, j])$ div r
13. $T[i+1, j-1] = (T_m[i, j]+M[i, j]N[i, j]+C_R[i, j])$ mod r
14. $M[i, j+1] = M[i, j]$}
15. $A[i, j+1] = A[i, j]$
16. $B[i+1, j] = B[i, j]$
17. $N[i+1, j] = N[i, j]$
18. }

그림 6. 독립 carry를 가지는 Montgomery 알고리듬
Fig. 6. Montgomery algorithm with two independent carries.

다. 또한 이 알고리듬의 정확성을 검증하기 위해 PC상에서 그림 1, 2, 3 그리고 그림 6의 알고리듬을 각각 C 언어로 구현하여 시뮬레이션한 결과 모두 정확하게 동작함을 확인하였다.

모듈라 곱셈 알고리듬 MMA_4를 2차원 종속 그래프로 나타낼 경우 데이터 흐름 및 처리 방향은 그림 4와 동일하다. 다만 carry의 구분에서 C_0와 C_1이 각각 C_M과 C_R로 대치된다. 그러나 각 처리기의 내부 구조는 내부 연산 알고리듬에 따라 달라진다. 제안 곱셈 처리기의 주 연산은 그림 6의 단계 10에서 13인데 이는 그림 3의 단계 9에서 11에 해당한다. 그림 6의 단계 10부터 13에서 $A[i, j]B[i, j]$와 $M[i, j]N[i, j]$은 AND 게이트로 간단하게 설계할 수 있으며 각 항들을 더하는 것은 전가산기(full adder)로 간단히 설계할 수 있어 다음과 같이 표현할 수 있다.

$$C_M[i,j+1] = ((T[i, j] \oplus C_M[i, j]) \wedge (A[i, j] \wedge B[i, j])) \vee (T[i, j] \wedge C_M[i, j])$$
$$T_m[i, j] = T[i, j] \oplus C_M[i, j] \oplus (A[i, j] \wedge B[i, j])$$

$C_R[i, j+1] = ((T_m[i, j] \oplus C_R[i, j]) \wedge (M[i, j] \wedge N[i, j])) \vee (T_m[i, j] \wedge C_R[i, j])$

$T[i+1, j-1] = T_m[i, j] \oplus C_R[i, j] \oplus (M[i, j] \wedge N[i, j])$

이 논리식에 근거하여 그림 6의 j≠0일 때의 처리기 구조를 설계한 것이 그림 7의 (a)인데 모두 2개의 AND 게이트와 2개의 전가산기로 구현할 수 있다. 그러므로 곱셈기내의 하나의 처리기는 총 12개의 게이트로 구성할 수 있다.

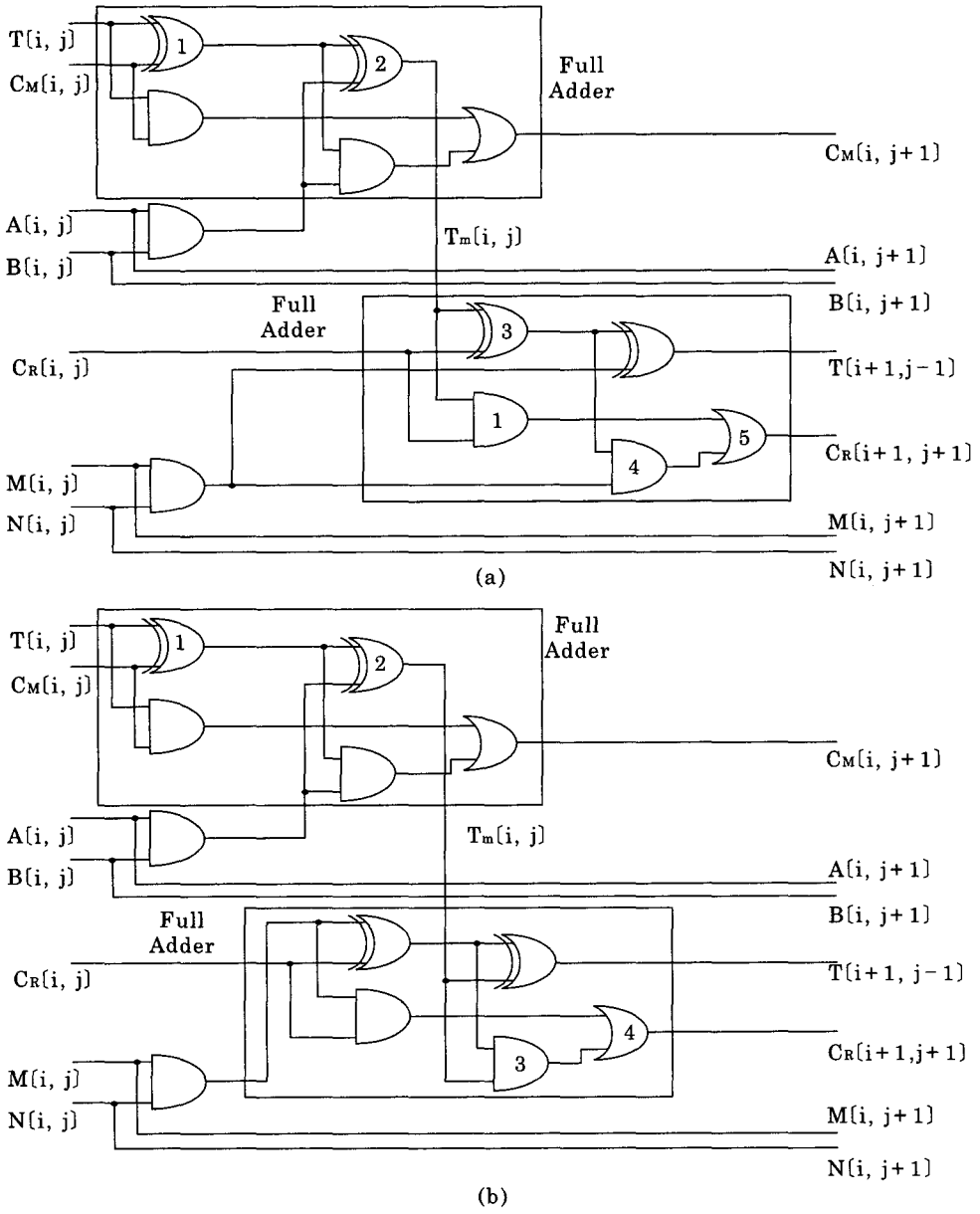


그림 7. 제안 곱셈기의 처리기 구조

Fig. 7. Circuit of the proposed processing element.

한편, 그림 7의 (a)에서 한 처리기의 연산이 종료되기 위해서는 모두 5개의 논리 게이트를 통과해야 한다. 이는 그림 6의 단계 11이 완전히 종료되어 중간 값 T_m $[i, j]$이 출력된 후 단계 12와 단계 13에서 $M[i, j]N[i, j]$와 $C_R[i, j]$을 더한다고 가정했기 때문이다. 그러나 단계 11과 12에서 $A[i, j]B[i, j]$와 $C_M[i, j]$를 더하는 과정과 단계 13과 14에서 $M[i, j]N[i, j]$와 C_R $[i, j]$를 더하는 과정이 독립적이므로 단계 12의 $T_m[i, j]$이 계산되기 전에 $M[i, j]N[i, j]$와 $C_R[i, j]$에 관한 덧셈을 먼저 수행할 수 있다. 즉, $C_R[i,j+1]$과 $T[i+1, j-1]$를 구하는 위의 논리식에서 $T_m[i, j]$와 $M[i, j]N[i, j]$의 위치를 바꾸어 아래와 같이 변형하여도 동일한 결과가 된다.

$$C_R[i, j+1] = (((M[i, j] \wedge N[i, j]) \oplus C_R[i, j]) \wedge T_m[i, j]) \vee ((M[i, j] \wedge N[i, j]) \wedge C_R[i, j])$$
$$T[i+1, j-1] = (M[i, j] \wedge N[i, j]) \oplus C_R[i, j] \oplus T_m[i, j]$$

이러한 사실에 근거하여 하나의 처리기를 통과하는데 필요한 시간을 줄이기 위한 회로가 그림 7 (b)이다. 그림에서 보는 바와 같이 단계 11에서 12까지, 그리고 단계 13에서 14까지의 서로 독립적인 연산은 병렬로 처리함으로써 최종 결과가 출력되기까지 총 4개의 논리 게이트를 통과하도록 설계할 수 있다.

한편, 2차원으로 시스토릭 어레이 곱셈기를 설계할 경우에는 많은 논리 게이트를 필요로 하여 하나의 칩으로 구현하기는 실용성이 적다. 예로서 모듈라 곱셈수의 크기가 512비트를 가정할 때 필요한 처리기 개수는 513×513개이며 제일 오른쪽의 처리기를 제외한 각 처리기는 12개씩의 논리 게이트를 필요로 한다. 이 문제를 해결하는 한가지 방법은 김 등[15]이 제안한 선형 시스토릭 어레이(linear systolic array)로 구현하는 것이다. 그림 4에서 보는 바와 같이 하나의 처리기는 한번의 곱셈시 자신의 계산점에서 한번의 데이타 처리만을 위해 사용한다. 그러므로 그림 4의 한 행(row)에 해당하는 시스토릭 어레이만 구성하고 한 처리기에서 출력되어 흐르는 결과$T[i, j]$는 오른쪽으로 궤환(feedback)시켜 사용할 수 있다. 간단한 예로서 $n=3$인 경우의 선형 시스토릭 어레이 구조를 나타낸 것이 그림 8이다.

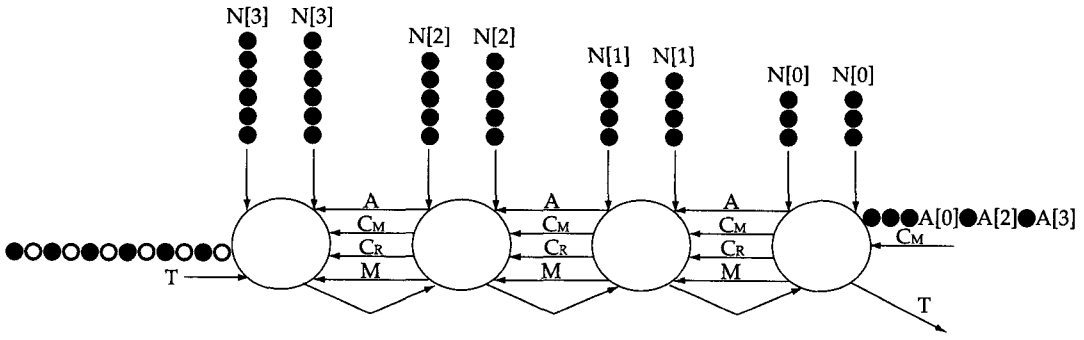


그림 8 선형 시스토릭 어레이
Fig. 8 Linear systolic array.

그림 8과 같이 입력 값 B와 N는 각 처리기에서 계속 머무르는 값이고 A, M 및 carry는 왼쪽으로 T값은 오른쪽으로 흐른다. 또 그림의 검정색 점은 하나의 처리기를 수행하는데 걸

리는 시간을 지연함을 의미하는데 이 지연이 필요한 이유는 그림 4에서 설명한 바와 같이 계산점 $[i, j]^T$와 $[i+1, j+2]^T$의 연산이 병렬로 이루어지기 때문이다. 즉, 하나의 처리기는 데이터를 입력받은 후 연산을 수행하고 인접 처리기가 실행되는 일정한 시간을 기다린 다음 다시 데이터를 입력받아 연산을 수행한다. Montgomery 곱셈기를 선형 시스토릭 어레이로 구성하면 하드웨어적인 부담을 약 $1/(n+1)$로 훨씬 감소시킬 수 있다. 모듈라 곱셈을 수행한 최종 결과가 나오는 시점 측면에서 볼 때 평면형 곱셈기는$(2n+2)t_w$초 후에 최하위 비트가 출력되고$(2n+2+n)t_w$초 후에 모든 결과가 출력되며 선형으로 구현할 경우에는 입출력단이 하나이므로 $2nt_w$만큼 지연된다. 이것은 제안 곱셈기에도 동일하게 적용되어 최종 결과는$(2n+2+n)t_p+(2n)t_p$초 후에 출력된다. 단, 여기서 t_w와 t_p는 각각 기존 곱셈기와 제안 곱셈기내의 처리기 통과 시간을 의미하며 $t_p=t_w \times 0.8$로 볼 수 있다.

결론적으로, 두 곱셈기가 모두 512 비트의 곱셈 연산을 수행한다고 가정하고 처리기 구조 및 수행 속도를 비교 분석한다. 기존의 방식으로 선형 시스토릭 어레이로 곱셈기를 구현할 경우 총 513개의 처리기가 필요하며 각각의 처리기는 총 14개의 게이트로 구성된다. 그러므로 하나의 곱셈기를 구현하는데 약 7182개의 논리 게이트가 필요하다. 반면, 제안 곱셈기의 각 처리기는 총 12개의 게이트로 구성되어 있어 하나의 곱셈기를 구현하는데 약 6156개의 논리 게이트가 필요하게 되어 하드웨어 용량을 약 14%정도 감소시킬 수 있다.

처리 속도 측면에서 하나의 처리기가 걸리는 시간을 보면 기존의 곱셈기는 5개의 게이트를 통과하는 시간이다. 반면 제안 곱셈기는 그림 7의 (b)에서와 같이 4개의 게이트만 통과하면 결과가 출력된다. 이 경우 각 게이트의 처리 시간이 일정하다고 가정하면 하나의 처리기를 통과하는 시간이 20%정도 감소됨을 알 수 있다. 그러므로 제안하는 곱셈기를 선형 시스토릭 어레이로 구성하면 하드웨어적인 부담을 줄이고 고속 곱셈을 실현할 수 있다. 상기 분석 내용을 요약한 것이 표 1이다.

표 1. 하드웨어 용량 및 수행속도 비교($n=512$, $r=2$)

Table 1. Comparison of required hardware amount and execution time.($n=512$, $r=2$)

구 분		기존 곱셈기	제안 곱셈기
하드웨어	처리기당 게이트	14개 (XOR:5, AND:7, OR:2)	12개 (XOR:4, AND:6, OR:2)
	전체 게이트	7182개	6156개
	용량비율	1	0.86
곱셈속도	처리기 게이트(시간)	5개(t_w) (XOR:1, AND:2, OR:2)	4개(t_p) (XOR:2, AND:1, OR:1)
	최종 출력시간	$2562t_w$초	$2562t_p$초
	수행 시간비율	1	0.8

V. 결 론

본 논문에서는 Montgomery 모듈라 곱셈 알고리듬에 기반한 선형 시스토릭 어레이를 분석하고 이를 하드웨어 용량 및 처리 속도측면에서 개선하여 설계하였다. 기존의 곱셈기는 두 수의 곱셈과 모듈라 감소시 발생하는 carry를 동시에 혼합하여 처리하였는데 이로 인해

상위 두 자리의 carry가 발생하게 되고 각 처리기의 회로가 복잡해지는 단점이 있었다. 제안 곱셈기에서는 곱셈과 모듈라 감소시 발생하는 carry를 상호 독립적으로 처리함으로써 각 처리기의 내부회로를 간소화하였다.

설계한 선형 시스토릭 어레이 곱셈기를 기존의 것과 처리기 개수 및 모듈라 곱셈 시간을 비교 분석한 결과, 구현에 필요한 하드웨어 부담을 14%정도 줄일 수 있고 모듈라 곱셈 시간을 20%정도 개선할 수 있음을 확인하였다. 제안 선형 시스토릭 어레이는 고속이면서 하드웨어적인 부담이 크지 않아 정보 보호를 위한 곱셈기의 하드웨어 구현에 효과적으로 사용될 수 있을 것이다.

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. Info. Theory, Vol. 31, No. 4, pp. 469-472, July 1985.

[3] A. Bosselaers, R. Govaerts and J. Vandewalle, "Comparison of three modular reduction functions," Advances in Cryptology, Proc. CRYPTO '93, pp. 175-186, 1993.

[4] 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.

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

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

[7] D. E. Knuth, The Art of Programming, Vol.2 :Seminumerical Algorithms, 2nd Ed. Addison-Wesley, 1981.

[8] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," In Advances in Cryptology-CRYPTO' 86, pp. 311-323, Springer-Verlag, 1987.

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

[10] D. Naccache and D. M'Raihi, "Arithmetic co-processors for public-key cryptography : The state of the Art, IEEE Micro, Vol. 14, No 3, pp. 14-24, June, 1996

[11] C. D. Walter, "Systolic Modular Multiplication," IEEE Trans. on Computers, Vol. 42, No. 3, pp. 376-378, 1993.

[12] 하재철, 오중효, 유기영, 문상재, "고속 멱승을 위한 모듈라 곱셈기 회로 설계," 통신정보 보호학회 학술대회 논문집, 제 7권 1호, pp. 222-231, 1997. 11.

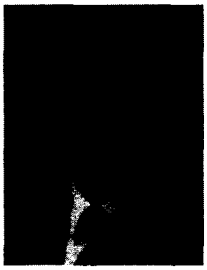
[13] J. Sauerbrey, "A Modular Exponentiation Unit based on Systolic Arrays," Abst. AUSCRYPT '92, pp. 12.19-12.24, 1992.

[14] K. Iwamura, T. Matsumoto, and H

imai, "Systolic arrays for modular exponentiation using Montgomery method," Proc. EUROCRYPT '92, pp. 477-481, 1992.

[15] 김현철, 허영준, 유기영, "모듈러 곱셈을 위한 고정-크기 선형 시스토릭 어레이 설계," 정보과학회 병렬처리시스템 학술발표대회 논문집, 제 7권 2호, pp. 21-32, 1996.

□ 著者紹介



하 재 철

1985. 3 ~ 1989. 2 경북대학교 전자공학과 (전자공학, 공학사)
1989. 3 ~ 1991. 6 육군통신장교 근무
1989. 3 ~ 1993. 8 경북대학교 대학원 전자공학과 (정보통신, 공학석사)
1998. 2 ~ 경북대학교 대학원 전자공학과(정보통신, 공학박사)
1998. 3 ~ 현재 나사렛대학교 전산정보학과 전임강사

※ 주관심분야 : 디지탈 통신, 정보보호, 부호이론

문 상 재

1972년 2월 서울대학교 공과대학 공업교육과(전자공학, 공학사)
1974년 2월 서울대학교 대학원 전자공학과(통신공학, 공학석사)
1984년 6월 미국 UCLA(통신공학, 공학박사)
1984년 6월 ~ 1985년 6월 UCLA Postdoctor 근무
1996년 12월 ~ 1985년 6월 미국 OMNET 컨설턴트
1994년 ~ 현재 경북대학교 공과대학 전기전자공학부 교수

※ 주관심분야 : 정보보호, 디지털 통신, 정보통신망