고속 멱승을 위한 모듈라 곱셈기 회로 설계

°하 재철*, 오 중효*, 유 기영**, 문 상재*
* 경북대학교 전자전기공학부
** 경북대학교 컴퓨터공학과

Circuit Design of Modular Multiplier for Fast Exponentiation

°Jae-Cheol Ha*, Joong-Hyo Oh*, Kee-Young Yoo**, Sang-Jae Moon*

* School of Electronics and Electrical Eng., Kyungpook National University
** Dept. of Comuter Eng., Kyungpook National University

본 연구는 정보통신연구관리단의 국책기술개발 사업 지원에 의해서 이루어졌습니다

요 약 문

본 논문에서는 고속 멱승을 위한 모듈라 곱셈기를 시스토릭 어레이로 설계한다. Montgomery 알고리듬 및 시스토릭 어레이 구조를 분석하고 공통 피승수 곱셈 개념을 사용한 변형된 Montgomery 알고리듬에 대해 시스토릭 어레이 곱셈기를 설계한다. 제안 곱셈기는 각 처리기 내부 연산을 병렬화할 수 있고 연산 자체도 간단화할 수 있어 시스토릭 어레이 하드웨어 구현에 유리하며 기존의 곱셈기를 사용하는 것보다 멱승 전체의 계산을 약 0.4배내지 0.6배로 감소시킬 수 있다.

1. 서 론

RSA나 ElGamal형 공개 키 암호 시스템에서의 주 연산은 모듈라 멱승(exponentiation)이다[1,2]. 멱승 $A^E \bmod N$에 사용되는 A, E 및 N은 안전도에 따라 가변적이나 대부분 512비트 이상을 사용한다. 이와 같은 멱승 연산은 계산 시간이 많이 소요되어 이를 감소시키기 위한 많은 연구가 있었다[3,4,5]. 고속 멱승 연산을 위해서는 기본 연산인 모듈라 곱셈을 고속으로 처리하거나 요구되는 모듈라 곱셈수가 적은 멱승 방식을 사용해야 한다. 현재, 모듈라 곱셈 방법으로는 Montgomery 알고리듬[6]이 가장 고속인 것으로 알려져 있다. Montgomery 알고리듬은 곱셈기의 내부 연산이 규칙적이고 데이타 흐름이 일정한 구조를 가지고 있어 최근에는 이 알고리듬을 이용하여 VLSI 하드웨어로 시스토릭 어레이(systolic array)를 설계하는 연구도 있었다. C. D. Walter[7]는 모듈라 곱셈을 위한 평면 시스토릭 어레이를 제안하였으며 김 등[8]은 이를 분할(partitioning)기법에 기초하여 4가지 종류의 선형 시스토릭 어레이를 설계하였다.

본 논문에서는 C. D. Walter가 설계한 Montgomery 모듈라 곱셈기를 분석하고 고속 멱승을 위해 하[9] 등이 제안한 변형된 Montgomery 알고리듬을 시스토릭 어레이로 설계한다. 변형된 Montgomery 알고리듬은 공통 피승수 곱셈 개념을 사용한 것으로서 이를 시스토릭 어레이로 설계하고 처리기 개수, 수행 시간 및 데이타 흐름 등을 비교 분석한다.

2. Montgomery 알고리듬의 시스토릭 어레이

공개 키 암호 시스템에 사용하는 모듈라 멱승은 $AB \bmod N$형태의 모듈라 곱셈의 반복으로 이루어

진다. 현재까지의 모듈라 곱셈 알고리듬 중 Montgomery 알고리듬이 가장 고속이며 구현이 효율적인 것으로 알려져 있어 이를 이용한 소프트웨어 구현 및 VLSI 하드웨어 설계에 관한 많은 연구가 있었다. 시스토릭 어레이는 여러 개의 처리기(Processing Element, PE)가 망 구조로 연결되어 데이타를 규칙적으로 입력받고 이를 처리하여 이웃 처리기로 전달하는 형태이다. 각 PE들은 정규적으로 흐르는 데이타를 순차적으로 처리하며 최종 PE에서 곱셈 결과를 출력하게 된다.

한편, J. Sauerbrey[10]는 두개의 시스토릭 어레이로 Montgomery 모듈라 곱셈기를 만들고 이를 사용하여 모듈라 멱승기를 구현하였으며, K. Iwamura 등[11]은 Montgomery 모듈라 곱셈을 수행하는 두 종류의 일차원 시스토릭 어레이를 설계하였다. C. D. Walter는 모듈라 곱셈을 위한 평면 시스토릭 어레이를 제안하였는데 본 절에서 이를 분석하고 시스토릭 어레이 설계 방법을 기술한다.

2.1 Montgomery 알고리듬

Montgomery 모듈라 곱셈 $ABR^{-1} \bmod N$은 내부 연산이 규칙적이고 데이타 흐름이 일정한 구조를 가지고 있다. 여기서 R은 N과 서로 소인 N보다 큰 정수이이며 A, B 및 N은 모두 n비트이고 A와 B가 N보다 작다고 가정한다. 또 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 있다. C. D. Walter는 시스토릭 어레이로 쉽게 구현하기 위해 Montgomery 알고리듬을 다음과 같이 변형하였다.

$MMA_1\ (A, B, N, r)$

$n_0 = -N[0]^{-1} \bmod r$

$T = 0$

for $i = 0$ *to* $k-1$ *step* 1{

$M[i] = ((T[0] + A[i]B[0]n_0) \bmod r$

$T = (T + A[i]B + M[i]N) \text{ div } r$

}

return(T)

이 알고리듬에서 $r=2$로 하면 $k=n$이 되고 최종 단계의 T는 $r^nT = AB + MN$이 되므로 $T = ABr^{-n} \bmod N$이 된다. 그러나 T의 최대 값은 $T<2N$가 되어 이 결과를 바로 다음 계산에 이용하기가 힘들다. 따라서 $A[n]=0$를 A에 덧붙여 한번 더 수행하면 $T<N$이 되어 다음 입력으로 그대로 사용할 수 있다. Montgomery 알고리듬에서는 $R=r^n$를 잉여류 변환에 많이 사용하지만 이 경우에는 상기한 문제를 해결하기 위해 $R=r^{(n+1)}$을 사용하였다.

Montgomery 알고리듬 MMA_1에서는 A와 M을 인덱스 i로 하여 1차원 구조로 되어 있지만 B와 N을 인덱스 j로 하여 2차원 공간으로 확장할 수 있다. 상기한 알고리듬 MMA_1을 2차원으로 확장하여 연산이 병렬적으로 수행됨을 볼 수 있는 정규 순환 방정식으로 표현하면 아래와 같다. 여기서 $A[i,j]$는 A값의 i번째 인덱스 및 j번째 인덱스에 해당하는 자리 값을 표시하는 인덱스 점(index point)을 의미한다.

$MMA_2(A, B, N, r)$

$n_0[0,0] = -N[0]^{-1} \bmod r$

for $i = 0$ *to* n *step* 1

for $j=0$ *to* n *step* 1 {
  if($j=0$){
    $M[i,j]=((T[i,j]+A[i,j]B[i,0])n_0[0,0])\ \bmod r$
    $C_0[i,j+1]=(T[i,j]+A[i,j]B[i,j]+M[i,j]N[i,j])\ div\ r \bmod r$
    $C_1[i,j+1]=(T[i,j]+A[i,j]B[i,j]+M[i,j]N[i,j])\ div\ r\ div\ r$
    $M[i,j+1]=M[i,j]$
  }
  else {
    $C_0[i,j+1]=(T[i,j]+A[i,j]B[i,j]+M[i,j]N[i,j]$
      $+C_0[i,j]+C_1[i,j]r)\ div\ r \bmod r$
    $C_1[i,j+1]=(T[i,j]+A[i,j]B[i,j]+M[i,j]N[i,j])$
      $+C_0[i,j]+C_1[i,j]r)\ div\ r\ div\ r$
    $T[i+1,j-1]=(T[i,j]+A[i,j]B[i,j]+M[i,j]N[i,j]+C_0[i,j])\bmod r$
    $M[i,j+1]=M[i,j]$
  }
  $A[i+1,j]=A[i,j]\ \ B[i,j+1]=B[i,j]\ \ N[i+1,j]=N[i,j]$
}

MMA_2 알고리듬에 사용되는 2차원 배열 변수 A, B, N 및 T의 초기 값은 다음과 같다.

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

최종 계산 값은 $T[n+1,0]$에서 $T[n+1,n-1]$까지이며 각 PE에서 발생하는 carry 값은 두자리수가 된다. 이 알고리듬에서 $r=2$로 하면 비트 단위 연산이 가능하며 $n_0[0,0]$는 항상 1이 되어 연산을 생략할 수 있다.

2.2 시스토릭 어레이

C. D. Walter가 제안한 순환 정규 방정식 MMA_2를 종속 그래프(dependence graph)로 나타내면 그림 1과 같다. 종속 그래프의 계산 공간을 좌표 $[i,j]^T$로 나타내면 좌표가 $[i,0]^T$인 계산점에서는 MMA_2의 $j=0$인 부분이 수행되며, 그 나머지 계산점에서는 MMA_2의 $j\neq 0$인 부분이 수행된다. 데이타 $A[i]$는 $[i,0]^T$에서 입력되고 $[0,1]^T$방향으로 이동하며 M값은 제일 오른쪽에서만 계산되어 $[0,1]^T$방향으로 흐른다. Carry값은 모든 계산점에서 계산되어 $[0,1]^T$방향으로 전달된다. $B[j]$와 $N[j]$는 계산점 $[0,j]^T$에서 입력되고 $[1,0]^T$방향으로 흐른다. 그리고 T값은 $[0,j]^T$및 $[i,n]^T$의 계산점에서 입력되어 방향벡터 $[1,-1]^T$로 전달된다. 이 시스토릭 어레이는 $r=2$일 때 $(n+1)(n+1)$개의 처리기가 사용되며 계산점이 $[i,j]^T$와 $[i+1,j+2]^T$인 PE는 병렬로 데이타를 처리한다.

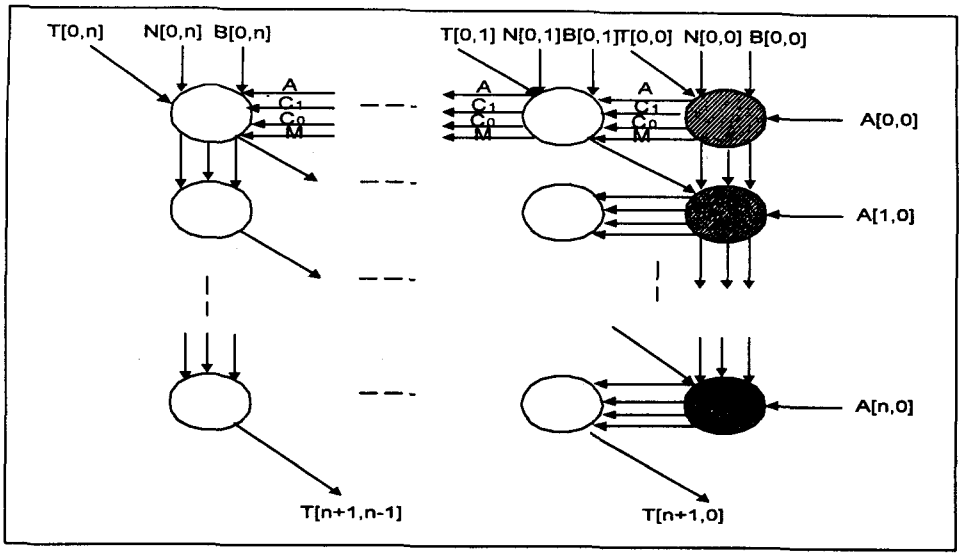


그림 1. Montgomery 곱셈기의 시스토릭 어레이 구조

3. 고속 멱승을 위한 시스토릭 어레이 설계

본 논문에서는 고속 멱승을 위해 하[9] 등이 제안한 변형된 Montgomery 알고리듬을 시스토릭 어레이로 설계한다. 변형된 Montgomery 알고리듬은 공통 피승수 곱셈 개념을 사용한 것으로서 소프트웨어로 이진 멱승 방식을 구현했을 경우 멱승 시간을 약 16%정도 줄일 수 있었다. 본 절에서는 시스토릭 어레이 설계를 위해 이 모듈라 곱셈 방식을 순환 방정식 및 종속 그래프로 표현한다. 종속 그래프로부터 자료흐름 벡터들과 초기 값의 위치를 구한 다음 2차 시스토릭 어레이 곱셈기를 설계한다.

3.1 Montgomery 알고리듬에 대한 공통 피승수 곱셈

이진방식은 가장 일반적인 멱승 방식으로서 n비트인 지수 E를 이진수로 표현한 후 지수를 탐색하면서 모듈라 곱셈을 반복하는 방식이다. 즉, $E=(e_{n-1}e_{n-1}\cdots e_0)_2$, $e_i\in\{0,1\}$으로 표현한 수 있으며 이진수로 표현된 E의 "1"과 "0"의 개수는 동일하다고 가정한다. Montgomery 알고리듬을 사용한 L-R형 및 R-L형 이진 방식을 나타낸 것이 그림 2이다. 여기서 $REDC(AB)$는 $ABR^{-1} \bmod N$을 나타낸다.

L-R Binary : $A^E \bmod N$	R-L Binary : $A^E \bmod N$

$B=R \bmod N$
$A=AR \bmod N$
 	
for $i=n-1$ *to* 0 *step* -1 {
$B=REDC(BB)$
if($e_i=1$) $B=REDC(AB)$
}
$B=REDC(B)$
return(B)
	$B=R \bmod N$
	$A=AR \bmod N$
	for $i=0$ *to* $n-1$ *step* 1 {
	if($e_i=1$){ $B=REDC(AB)$ $A=REDC(AA)$ }
	else $A=REDC(AA)$
	}
	$B=REDC(B)$
	return(B)

그림 2. 이진 멱승 방식

그림 2의 L-R형이나 R-L형 모두 평균 $1.5n$번의 모듈라 곱셈이 필요하다. 그러나 R-L형에서는 e_i가 1일 때에는 $REDC(AB)$과 $REDC(AA)$를 수행하는데 이와 같이 동일한 A에 관한 두번의 모듈라 곱셈을 공통 피승수 모듈라 곱셈(common-multiplicand modular multiplication)이라 한다. R-L형에서 이 두번의 곱셈은 병렬 처리가 가능하므로 두 개의 모듈라 곱셈기를 설계하여 e_i가 1일 때에는 $REDC(AB)$과 $REDC(AA)$를 독립적으로 계산할 수 있다. 이와 같은 병렬처리는 멱승 전체의 소요 시간은 n번의 모듈라 곱셈기를 구동한 시간과 같아져 L-R형 모듈라 멱승 시간의 2/3정도이지만 두 개의 곱셈기를 사용해야 하므로 하드웨어적인 부담이 크다.

제안하는 모듈라 곱셈은 공통 피승수 모듈라 곱셈 $REDC(AB)$과 $REDC(AA)$의 공통 계산 부분을 추출하여 한번만 계산함으로써 계산 효율을 높이면서 하나의 곱셈기를 사용하여 구현하자는 것이다. 이를 위해 먼저 Montgomery 곱셈을 다음과 같이 변형한다. 여기서 $R=r^{(k+1)}$이라 둔다.

$$REDC(AB) = ABR^{-1} \bmod N$$
$$= A(B[k-1]r^{k-1}+B[k-2]r^{k-2}+\cdots+B[0]r^{0})r^{-(k+1)} \bmod N$$
$$= A(B[k-1]+B[k-2]r^{-1}+\cdots+B[0]r^{k-1})r^{-2} \bmod N$$
$$= (B[k-1]A+B[k-2]Ar^{-1}+\cdots+B[0]Ar^{k-1})r^{-2} \bmod N$$
$$= T_B r^{-2} \bmod N$$

그러나 이 경우 최종 결과를 N보다 작게 하기 위해서는 T_B의 크기를 고려해야 한다. 만약, N이 $n=(\log_2 r)k$비트이면 T_B는 최대 $n+\log_2 r+\log_2 k$비트가 되고 $k+1+\lceil(\log_2 k)/(\log_2 r)\rceil$자리가 된다. 여기서 $\log_2 r$은 한자리 수를 표현하는 데 필요한 비트 수이다. 예를 들어 $n=512$이고 $r=2^{16}$이면 $k=32$가 되어 T_B는 이는 34자리가 된다. 그러므로 $T_B r^{-2} \bmod N$은 k자리가 된다. 그러나 $n=512$이고 $r=2$이면 $k=n=512$가 되어 T_B는 최대 522자리가 된다. 그러므로 $T_B r^{-2} \bmod N$을 수행하면 520자리가 된다. 이를 다시 k자리로 줄이기 위해서는 8번의 모듈라 감소 과정이 더 필요하다. 그러므로 $r=2$일 경우에는 최종 $REDC(AB)=T_B r^{-10} \bmod N$로 하여 최종 결과를 N보다 작게 할 수 있다. 물론 A, B 및 N의 크기 및 r의 선택에 따라 T_B의 자리수는 달라지나 본 논문에서는 N은 512비트이고 $r=2$일 경우로 가정한다. 따라서 $r=2$일 때에는 $R=r^{(n+9)}$으로 두고 공통 피승수 곱셈 $REDC(AB)$과 $REDC(AA)$를 나타내면 다음과 같다.

$$REDC(AB) = ABR^{-1} \bmod N$$
$$= A(B[n-1]r^{n-1}+B[n-2]r^{n-2}+\cdots+B[0]r^{0})r^{-(n+9)} \bmod N$$
$$= (B[n-1]A+B[n-2]Ar^{-1}+\cdots+B[0]Ar^{n-1})r^{-10} \bmod N$$
$$= T_B r^{-10} \bmod N$$
$$REDC(AA) = AAR^{-1} \bmod N$$
$$= A(A[n-1]r^{n-1}+A[n-2]r^{n-2}+\cdots+A[0]r^{0})r^{-(n+9)} \bmod N$$
$$= (A[n-1]A+A[n-2]Ar^{-1}+\cdots+A[0]Ar^{n-1})r^{-10} \bmod N$$
$$= T_A r^{-10} \bmod N$$

위의 두 식에서 $Ar^{-i}(0\le i\le n-1)$가 공통적으로 사용되므로 다음과 같이 한번만 계산하여 사용할 수 있다.

$$A_0=A,\ A_1=A_0r^{-1} \bmod N,\ A_2=A_1r^{-1} \bmod N,\ \cdots\ A_{n-1}=A_{n-2}r^{-1} \bmod N$$

먼저 소프트웨어로 구현시 제안한 모듈라 곱셈의 계산량은 분석하면 다음과 같다. 먼저 하나의 A_i값을 계산하는데 $k+1$번의 작은 수 곱셈이 필요하므로 모든 A_i값을 계산하는데 $(k-1)(k+1)$번의 작은 수 곱셈이 필요하다. 그러므로 $REDC(AB)$를 계산하는데 $k^2+(k-1)(k+1)+(1+\lceil(\log_2 k)/(\log_2 r)\rceil)(k+1)$번의 곱셈이 필요하다. 그러나 $REDC(AA)$를 계산할 때에는 공통 계산 부분을 생략할 수 있으므로 $k^2+(1+\lceil(\log_2 k)/(\log_2 r)\rceil)(k+1)$번의 곱셈이 필요하다. 그러므로 제안한 방식은 Montgomery 곱셈에 비해 공통의 피승수를 가지는 두 번의 모듈라 곱셈의 계산량을 약 0.75배로 줄일 수 있다.

제안한 공통 피승수 곱셈 $X=REDC(AB)$ 및 $Y=REDC(AA)$를 $r=2$로 한 정규 순환 방정식으로 표현하면 아래와 같다. 물론 R-L형 이진 멱승에서 e_i가 "0"일 경우에는 $Y=REDC(AA)$만 필요하므로 출력된 X는 무시하면 된다.

$MMA_3(A,B,N,r)$

$n_0[0,0]=-N[0]^{-1} \bmod r$

for $i=0$ *to* n *step* 1

  for $j=0$ *to* $n+9$ *step* 1 {

    if$(j=0)$ {

      $X[i+1,j]=(X[i,j]+T[i,j]B[i,0]) \bmod r$

      $X_0[i,j+1]=(X[i,j]+T[i,j]B[i,j])$ *div* r

      $Y[i+1,j]=(Y[i,j]+T[i,j]A[i,0]) \bmod r$

      $Y_0[i,j+1]=(Y[i,j]+T[i,j]A[i,j])$ *div* r

      $M_X[i,j]=(T[i,j]n_0[0,0]) \bmod r$

      $C_0[i,j+1]=(T[i,j]+M_X[i,j]N[0,j])$ *div* r

      $M_X[i,j+1]=M_X[i,j]$

    }

    else {

      $X[i+1,j]=(X[i,j]+T[i,j]B[i,0]+X_0[i,j]) \bmod r$

      $X_0[i,j+1]=(X[i,j]+T[i,j]B[i,j]+X_0[i,j])$ *div* r

      $Y[i+1,j]=(Y[i,j]+T[i,j]A[i,0]+Y_0[i,j]) \bmod r$

      $Y_0[i,j+1]=(Y[i,j]+T[i,j]A[i,j]+Y_0[i,j])$ *div* r

      $C_0[i,j+1]=(T[i,j]+M_X[i,j]N[0,j]+C_0[i,j])$ *div* r

      $T[i+1,j-1]=(T[i,j]+M_X[i,j]N[0,j]+C_0[i,j]) \bmod r$

      $M_X[i,j+1]=M_X[i,j]$

    }

  }

for $j=0$ *to* $n+9$ *step* 1 { $X[0,j]=X[n+1,j]$ $Y[0,j]=Y[n+1,j]$}

for $i=0$ *to* 10 *step* 1

  for $j=0$ *to* $n+9$ *step* 1 {

    if$(j=0)$ {

$M_X[i,j]=(X[i,j]n_0[0,0])$ mod r
$X_0[i,j+1]=(X[i,j]+M_X[i,j]N[0,j])$ *div* r
$M_X[i,j+1]=M_X[i,j]$
$M_Y[i,j]=(Y[i,j]n_0)$ mod r
$Y_0[i,j+1]=(Y[i,j]+M_Y[i,j]N[0,j])$ *div* r
$M_Y[i,j+1]=M_Y[i,j]$
}
else{
$X_0[i,j+1]=(X[i,j]+M_X[i,j]N[0,j]+X_0[i,j])$ *div* r
$X[i+1,j-1]=(X[i,j]+M_X[i,j]N[0,j]+X_0[i,j])$ mod r
$M_X[i,j+1]=M_X[i,j]$
$Y_0[i,j+1]=(Y[i,j]+M_Y[i,j]N[0,j]+Y_0[i,j])$ *div* r
$Y[i+1,j-1]=(Y[i,j]+M_Y[i,j]N[0,j]+Y_0[i,j])$ mod r
$M_Y[i,j+1]=M_Y[i,j]$
}

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

$T[0,j]=A[j]$ $(0\le j<n)$ and $T[0,j]=0$ $(n\le j\le n+9)$
$N[0,j]=N[j]$ $(0\le j<n)$ and $N[0,j]=0$ $(n\le j\le n+9)$
$A[n-1-i,0]=A[i]$ $(0\le i<n)$ and $A[n,0]=0$
$B[n-1-i,0]=B[i]$ $(0\le i<n)$ and $B[n,0]=0$
$X[0,j]=0$ $(0\le i\le n+9)$,
$Y[0,j]=0$ $(0\le i\le n+9)$

REDC(AB)의 최종 계산 값은 $X[10,0]$에서 $X[10,n-1]$까지이며, *REDC*(AA)의 최종 계산 값은 $Y[10,0]$에서 $Y[10,n-1]$까지이다. 이 알고리듬에서도 역시 $r=2$로 하면 비트 연산이 가능하며 $n_0[0,0]$는 항상 1이 되어 연산을 생략할 수 있다. 그러나 MMA_3에서 발생하는 모든 carry는 C. D. Walter의 곱셈기와 달리 항상 1자리이다. 그 이유는 Montgomery 알고리듬 MMA_2에서 수행하는 최대로 큰 연산은 $T[i,j]+A[i,j]B[i,j]+M[i,j]N[i,j]+C_0[i,j]$이므로 결과 값을 표현하는데 4비트가 필요하며 carry가 두자리 발생한다. 반면, MMA_3의 최대 연산은 $X[i,j]+T[i,j]B[i,0]+X_0[i,j]$가 되어 한 항이 적다. 그러므로 이를 표현하는 데는 3비트면 충분하며 carry가 한자리가 발생한다. MMA_3 알고리듬의 또 하나의 특징은 Ar^{-i}를 한 단계 앞서 구한 후 이 결과를 A나 B의 각 자리수와 곱할 수 있으므로 각각 독립적인 연산으로 분리할 수 있다. 즉, MMA_3의 첫번째 "for"문에서 보는 바와 같이 이전 단계에서 구한 Ar^{-i}를 A나 B의 자리수와 곱하고 다음 단계의 $Ar^{-(i+1)}$를 구한다.

3.2 시스토릭 어레이 설계

순환 정규 방정식 MMA_3를 종속 그래프로 나타내면 그림 3과 같다. 그림 3의 A부분은 MMA_3알고리듬의 인덱스 i에 대한 첫번째 "for"문을 수행하는 것이고 B부분이 두번째 "for"문을 수행하는 것이다.

좌표가 $[i,0]^T$인 계산점에서는 MMA_3의 $j=0$인 부분이 수행되며, 그 나머지 계산점에서는 MMA_3의 $j\neq 0$인 부분이 수행된다. 첫번째 "for"문에서 데이타 A와 B는 $[i,0]^T$에서 입력되고 $[0,1]^T$방향으로 이동하며 M값은 제일 오른쪽에서만 계산되어 $[0,1]^T$방향으로 흐른다. Carry값은 모든 계산점에서 계산되어 $[0,1]^T$방향으로 전달된다. $B[j]$와 $M[j]$는 계산점 $[0,j]^T$에서 입력되고 $[1,0]^T$방향으로 흐른다. 그리고 T값은 $[0,j]^T$및 $[i,n+9]^T$의 계산점에서 입력되어 방향벡터 $[1,-1]^T$로 전달된다. 첫번째 "for"문을 통과한 결과 값은 $n+10$자리가 된다. 그러므로 두번째 "for"문에서 이를 n자리로 감소하는 연산이 필요하다. 이 시스토릭 어레이는 $r=2$일 때 A부분은 $(n+1)(n+10)$개, B부분은 $10(n+10)$의 처리기가 사용되므로 총 $(n+11)(n+10)$가 필요하므로 C. D. Walter 설계방법보다는 PE의 개수가 늘어난다.

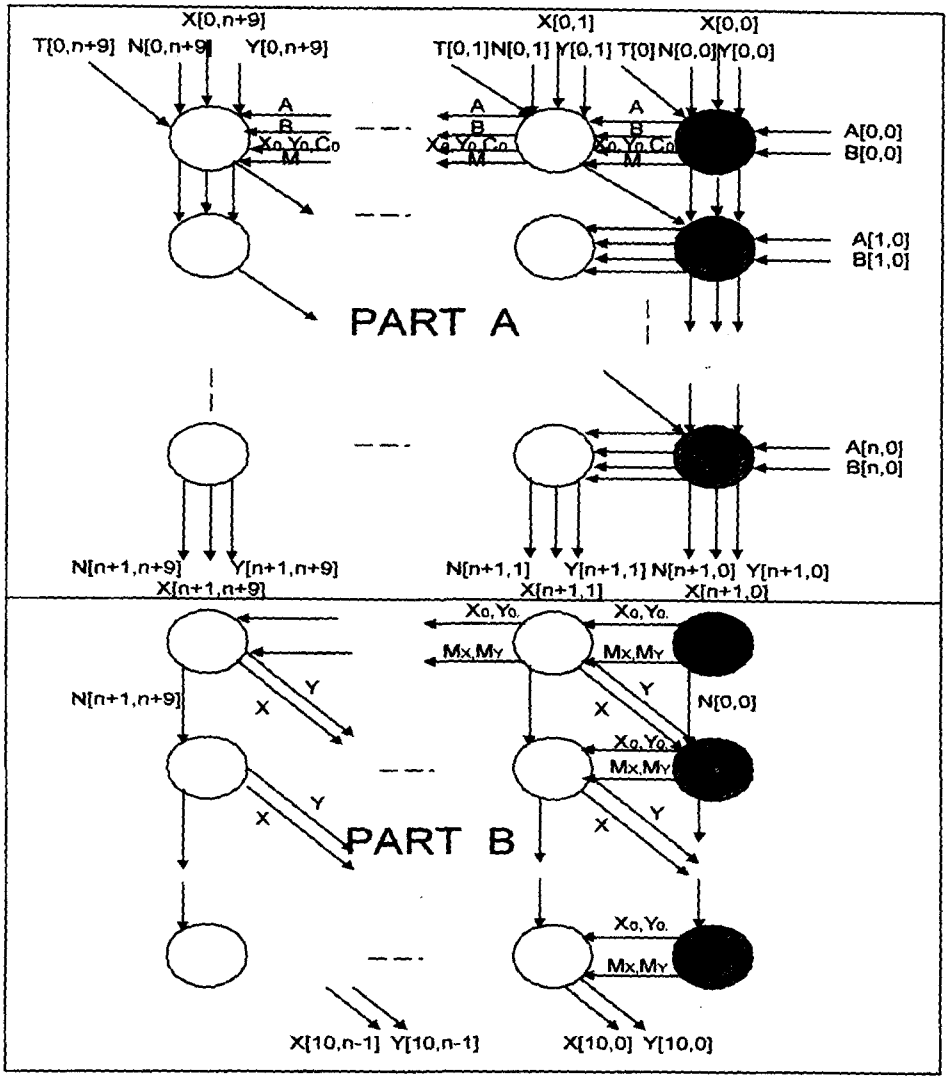
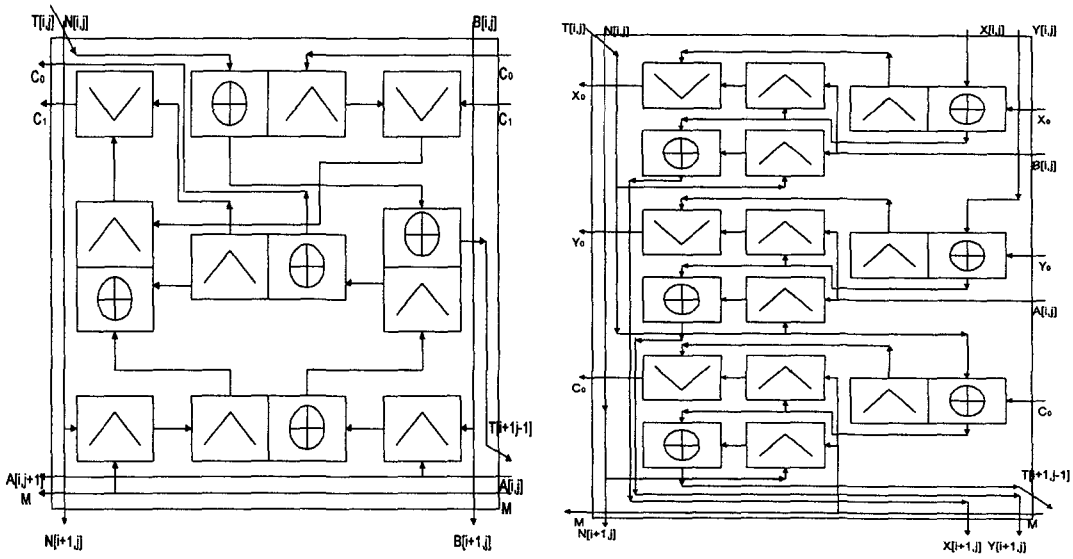


그림 3. 제안 Montgomery 곱셈기의 시스토릭 어레이 구조

4. 성능 및 효율성 분석

제안하는 모듈라 곱셈기는 공통의 피승수를 가지는 두번의 곱셈을 병렬적으로 처리함으로써 R-L형 멱승 수행속도를 고속화할 수 있도록 설계하였다. 먼저 두 곱셈기의 PE구조 및 수행 속도를 비교하고 멱승 전체의 속도를 분석한다. 단, 본 논문에서는 512비트의 곱셈기로 가정하고 비교한다. 그림 4는 C. D. Walter의 곱셈기와 본 논문에서 제안한 곱셈기의 PE 구조를 나타낸 것이다. 그림 4의 (a)는 $j \neq 0$일 때의 PE구조이며 (b)는 본 논문의 첫번째 "for"문의 $j \neq 0$일 때의 PE구조이다. 두번째 "for"문의 구조도 (b)와 비슷하게 설계할 수 있으며 한 PE를 통과하는 시간은 (b)와 동일하다.



(a) 기존 곱셈기의 PE (b) 제안 곱셈기의 PE

그림 4. PE 구조 비교

그림 4의 (a)는 총 14개의 게이트로 구성되어 있으나 (b)는 18개, 두번째 "for"문의 PE는 12개로 구성되어 하드웨어적인 부담은 (b)가 크다. 또 전체 PE의 개수는 (a)의 경우 513×513개이므로 총 513×513×14=3684336개의 게이트가 필요하다. (b)의 경우는 513×522×18+10×522×12=4882788개가 필요하게 되어 전체적으로 약 33%가 증가한다.

그러나 하나의 PE를 처리하는데 걸리는 시간을 보면 (b)는 3개만 통과하면 결과 값이 출력된다. 반면 (a)는 5개의 게이트를 통과해야 한다. 그 이유는 (b)에서 수행하는 연산이 병렬화 되어 있고 carry가 한자리만 발생하므로 그 처리시간을 줄일 수 있기 때문이다. 그러므로 (b)는 (a)에 비해 한 PE당 처리시간을 0.6배로 줄일 수 있다.

모듈라 곱셈을 수행한 최종 결과가 나오는 시점 측면에서 볼 때 C. D. Walter의 곱셈기는 $(2n+2)t$초 후에 최하위 비트가 출력되고 $(2n+2+n)t$초 후에 모든 결과가 출력된다. 여기서 t는 하나의 PE를 통과하는데 소요된 시간이다. 제안 곱셈기에서는 $(2n+22)t$초 후에 최하위 비트가 출력되고 $(2n+22+n)t$초 후에 모든 결과가 출력된다. 그러므로 최종 곱셈 결과를 출력하는 시간 비는 제안 방식이 $(3n+22)/(3n+2)\times 0.6$배 빠르다. 결국 제안 곱셈기는 n이 512일 경우 기존 곱셈기에 비해 약 0.61배 빠르게 된다.

전체 멱승시 C. D. Walter의 곱셈기를 사용하면 평균 $1.5n$번의 모듈라 곱셈이 필요하며 제안 방식은

곱셈기가 병렬 구조로 되어 있으므로 약 n번의 곱셈시간이면 충분하다. 그러므로 전체 멱승 시간 비로 환산하면 $n/1.5n \times 0.61 = 0.4$배가 된다. 결국, 제안한 곱셈기는 두번의 공통 피승수 곱셈을 병렬처리함으로써 하드웨어적인 부담은 있지만 멱승 속도를 60%정도 개선할 수 있다. 만약, 이 곱셈 알고리듬으로 한번의 곱셈 $REDC(AB)$만 수행하도록 설계하면 그림 4의 (b)는 12개, 두번째 "for"문의 PE도 6개로 구성할 수 있고 $513 \times 522 \times 12 + 10 \times 522 \times 6 = 3244752$개가 필요하게 되어 기존의 곱셈기보다 하드웨어적인 부담을 약 12%정도 더 줄일 수 있다. 이 경우 전체 멱승에 필요한 시간은 0.6배로 줄일 수 있다.

그러나 상기한 모듈라 곱셈기는 필요한 PE 개수 및 게이트 수가 너무 많아 하나의 칩으로 구현하기는 어렵다. 이를 해결하는 한 방법은 김 등[8]이 설계한 선형 시스토릭 어레이(linear systolic array)로 구현하는 것이다. 저자들이 제시한 4가지 방법 중 두번째 방법으로 기술하면 그림 3에서 보는 바와 같이 하나의 PE는 자신의 계산점에서 한번의 데이타 처리만을 위해 사용한다. 그러므로 한 열(row)에 해당하는 $n+10$개의 시스토릭 어레이로만 구성하고 한 PE에서 출력되어 흐르는 결과를 오른쪽으로 궤환(feedback)시키면 된다. 이 경우 계산점 $[i,j]^T$와 $[i+1,j+2]^T$의 연산이 병렬로 이루어지므로 한 PE를 통과하는 시간을 기다린 후 연산을 수행한다. 그림 3의 A부분에서는 X, Y 및 N는 각 PE에서 머무르는 값이고 T, A, B 및 carry는 흐르는 값이 된다. 이와 같은 선형적 구조는 입출력 단이 하나인 관계로 최초 입력 시간 및 최종 출력 시간이 필요하므로 $2(n+10)t$ 시간이 추가된다. 선형적으로 시스토릭 어레이를 구성하면 하드웨어적인 부담이 훨씬 감소하며 전체 멱승의 시간도 개선할 수 있다. 이와 같은 결과는 Montgomery 알고리듬을 시스토릭 어레이로 설계할 경우 carry가 한자리만 발생하도록 함으로써 그에 따른 게이트 수를 감소할 수 있기 때문이다. 상기 분석 내용을 요약한 것이 표 1이다.

표 1. 곱셈기 성능비교(512비트 모듈라 곱셈시)

구 분			C. D. Walter의 단일 곱셈기	제안 곱셈기	
				병렬 곱셈기	단일 곱셈기
2차원 시스토릭 어레이	하드웨어	PE당 게이트 수	14개	18 혹은 12개	12 혹은 6개
		전체 PE수	263169	273528	273528
		전체 게이트 수	3684336	4882788	3244752
	멱승 속도	PE당 출력 통과 게이트 수	5	3	3
		최종 출력 시간 비*	$(2n+2+n)t$	$(2n+22+n)t$	$(2n+22+n)t$
		멱승시 곱셈기 구동 회수	$1.5n$	n	$1.5n$
		멱승 시간 비	1	0.4	0.6
선형 시스토릭 어레이	하드웨어	PE당 게이트 수	14개	18 혹은 12개	12 혹은 6개
		전체 PE수	513	522×2	522×2
		전체 게이트 수	7182	15660	9396
	멱승 속도	PE당 출력 통과 게이트 수	5	3	3
		최종 출력 시간 비*	$(2n+2+n)t$ $+(2n)t$	$(2n+22+n)t$ $+2(n+10)t$	$(2n+22+n)t$ $+2(n+10)t$
		멱승시 곱셈기 구동 회수	$1.5n$	n	$1.5n$
		멱승 시간 비	1	0.4	0.6

* t : 하나의 PE를 통과하는데 소요된 시간

5. 결 론

본 논문에서는 C. D. Walter가 설계한 Montgomery 모듈라 곱셈기를 분석하고 고속 멱승을 위해 변형된 Montgomery 알고리듬 제안하여 이를 시스토릭 어레이로 설계한다. 변형된 알고리듬은 공통 피승수 곱셈 개념을 사용하여 두번의 곱셈을 병렬로 처리할 수 있도록 하였다. 또한 내부 처리되는 연산이 간소화되고 병렬적으로 처리할 수 있어 하나의 처리기를 통과하는 시간을 단축할 수 있었다.

설계한 시스토릭 어레이를 처리기 개수, 수행 시간 및 데이타 흐름 등을 비교 분석하고 멱승 전체의 시간을 분석하였다. 그 결과, 두번의 곱셈을 동시에 처리하는 경우에는 전체 게이트 수가 증가하여 기존의 곱셈기에 비해 하드웨어적인 측면에서는 다소 불리하나 멱승시간을 약 0.4배까지 줄일 수 있다. 제안한 알고리듬으로 한번의 곱셈을 수행하는 곱셈기를 설계하면 멱승시간을 약 0.6배까지 줄일 수 있다. 특히, 이를 선형적으로 설계하면 VLSI 시스토릭 어레이 회로 구현이 가능할 것이다. 결국 제안 모듈라 곱셈 알고리듬은 각 처리기 내부 연산을 병렬화할 수 있고 연산 자체도 간단화할 수 있어 하드웨어 구현에 유리하며 고속 멱승을 실현할 수 있다.

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

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

[5] 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.

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

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

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

[9] 하재철, 문상재, "Montgomery 모듈라 곱셈을 변형한 고속 멱승," 통신학회 논문지, 제 22권 5호, pp. 1036-1044, 1997. 5.

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

[11] K. Iwamura, T. Matsumoto, and H. Imai, "Systolic Arrays for Modular Exponentiation using Montgomery Method," *Proc. Euro CRYPT '92*, pp. 477-481, 1992.