시스톨릭 어레이 구조를 갖는 효율적인 n-비트 Radix-4 모듈러 곱셈기 구조

박 태 근† · 조 광 원††

요 약

본 논문에서는 Montgomery 알고리즘을 기반으로 시스톨릭 어레이 구조를 이용한 효율적인 Radix-4 모듈러 곱셈기 구조를 제안한다. 제안된 알고리즘을 이용하여 모듈러 곱셈을 위한 반복의 수가 감소되었으며, 따라서 n-비트의 모듈러 곱셈을 수행하기 위하여 $(3/2)n+2$ 클럭이 소요된다. 그러나 하드웨어의 이용도를 감안할 때 두 개의 곱셈에 대한 중첩(interleaving) 연산이 가능하며, 가장 빠른 시기에 새로운 곱셈을 시작한다면 하나의 모듈러 곱셈을 수행하기 위하여 평균 $n/2$ 클럭이 필요하다. 제안된 구조는 시스톨릭 어레이 구조의 잇점으로 규칙성과 확장성을 갖기 때문에 효율적인 VLSI 구조로 설계하기가 용이하다. 기존의 다른 구조들과 비교하여 볼 때 제안된 구조는 상대적으로 적은 하드웨어들을 사용하여 높은 수행 속도를 보여주었다.

Efficient Architecture of an n-bit Radix-4 Modular Multiplier in Systolic Array Structure

Taegeun Park† · Kwangwon Cho††

ABSTRACT

In this paper, we propose an efficient architecture for radix-4 modular multiplication in systolic array structure based on the Montgomery's algorithm. We propose a radix-4 modular multiplication algorithm to reduce the number of iterations, so that it takes $(3/2)n+2$ clock cycles to complete an n-bit modular multiplication. Since we can interleave two consecutive modular multiplications for 100% hardware utilization and can start the next multiplication at the earliest possible moment, it takes about only $n/2$ clock cycles to complete one modular multiplication in the average. The proposed architecture is quite regular and scalable due to the systolic array structure so that it fits in a VLSI implementation. Compared to conventional approaches, the proposed architecture shows shorter period to complete a modular multiplication while requiring relatively less hardware resources.

키워드 : 모듈러 곱셈(Modular Multiplication), 시스톨릭 어레이(Systolic Array), 래딕스 수 체계(Radix Number System)

1. 서 론

네트워의 발달과 컴퓨터범죄의 증가로 인해 보안에 대한 필요성이 부각 되면서 데이터 암호화를 이용하여 다른 이로부터 자신의 정보를 지키는 것이 중요한 문제로 떠올랐다. 또한 최근에는 단순히 데이터 암호화만이 아니라 상호 인증을 통하여 보안이 필요한 다양한 분야들로 그의 응용이 확대되고 있다. 암호화와 전자인증 두 가지를 제공하고 있는 RSA 암호화 시스템은 1977년 Ron Rivest, Adi Shamir, Leonard Adleman에 의해 제안되었다[1]. RSA 암호화 시스템은 공개키와 비밀키를 가지는 비대칭 암호화 알고리즘의 대표적인 알고리즘이고, 소인수 분해의 어려움에 안전도의 근간을 두고 있다. 계산방식은 모듈러 멱승연산이 주를 이루고 있으며, 안전도의 측면에서 512-비트 이상의 키를 사용하고 있다. 모듈러 멱승 연산을 하기 위해서는 많은 계산량을 필요로 하는데 512-비트 이상의 큰 정수에 대한 모듈러 멱승을 실시간 내에 수행하기 위해서는 모듈러 멱승을 효율적으로 처리할 수 있는 전용 연산 장치를 부착하여 기본 연산인 모듈러 곱셈을 효율적으로 수행할 수 있는 회로를 하드웨어로 구현해야 한다.

큰 정수에 대해서 모듈러 곱셈을 수행하는 방식에는 크게 두 가지로 나누어 볼 수 있는데, 곱셈($X = A \times B$) 후 모듈러 감소 연산($X \bmod N$)을 하는 방법과 곱셈과 모듈러 감소 연산을 동시에($A \times B \bmod N$) 하는 방법이 있다. 후자 중에서도 현재 모듈러 곱셈을 수행함에 있어서 가장 널리 사용되는 Montgomery의 알고리즘은 규칙적이고 간단한 연산 구조를 가지며 시스톨릭 어레이를 기반으로 한 설계가 용이하다는 특징을 가지고 있다. 그러나 n-비트 모듈러 곱셈에서는 일반적으로 n번의 반복과 각 반복 마다 2번의 덧셈 및 후처리 과정이 필요하다[2]. Montgomery 알고리즘을 구현하기 위하여 시스톨릭 어레이 구조가 제안되었으며[3], Montgomery 알고리즘을 기반으로 하여 개선된 Montgomery 알고리즘은 각 반복마다 덧셈 연산의 수를 한번으로 줄였으나 반복이

† 정 회 원 : 가톨릭대학교 정보통신전자공학부 교수
†† 준 회 원 : 가톨릭대학교 대학원 컴퓨터공학
논문접수 : 2003년 5월 2일, 심사완료 : 2003년 8월 14일

두 배로 늘어나 계산에는 동일한 시간이 필요하다[5]. Su[6], Jen[7] 등이 제안한 개선된 Montgomery 알고리즘은 곱셈의 결과를 상위 n-비트와 하위 n-비트로 나누어서 계산하는 구조로서 계산 시간과 성능를 어느 정도 향상시켰다. 현재까지 제안된 구조로는 효율적인 나머지 연산 처리와 모듈러스 크기에 독립적인 임계경로를 가지는 CSA(carry save adder)를 적용한 구조[8], 개선된 L-알고리즘의 반복을 줄였으나 이를 위하여 미리 계산한 값을 저장할 테이블이 필요한 구조[9], 확장 가능한 Digit-serial 방법을 적용한 구조[10], 테이블 룩업(table look-up)방식의 병렬 알고리즘을 적용한 구조[11] 등 다양한 형태의 구조가 제안 되었다.

본 논문에서는 개선된 Montgomery 알고리즘[6, 7]을 기반으로 한 모듈러 곱셈기를 위해 시스톨릭 어레이 구조를 이용하였으며, 효율적이고 규칙적인 Radix-4 모듈러 곱셈기로서 고속 연산을 수행할 수 있는 구조를 제안하였다. 제안된 구조는 시스톨릭 어레이 구조로서 임의의 n-비트 구조로 확장이 용이하며 Radix-4 연산을 하기위해 필요한 Booth 인코더기와 곱셈기, 모듈러스 연산 블록 역시 시스톨릭 구조로 구성되어 효율적으로 설계되었다. 또한 시스톨릭 구조이기 때문에 하나의 연산이 모두 끝나기 전에 다른 연산을 시작할 수 있다는 장점을 가지고 있다. 시스톨릭의 장점과 개선된 알고리즘[6, 7] 적용한 Radix-4 모듈러 곱셈기를 이용하여 연산 성능이 개선되었다.

논문의 구성은 다음과 같다. 2 장에서는 Montgomery 알고리즘과 개선된 Montgomery 알고리즘, 그리고 본 논문에서 제안된 Radix-4 모듈러 곱셈 알고리즘에 대해서 설명한다. 3장에서는 제안된 radix-4 모듈러 곱셈기와 모듈러스 처리 블록, Booth 인코딩, T_n 연산 구조 등을 설명하고 4장에서 시뮬레이션 결과 및 성능을 분석하며 5장에서 결론을 맺는다.

2. 모듈러 곱셈 알고리즘

현재까지 CSA(Carry Save Adder)나 CLA(Carry Look-Ahead Adder) 등을 적용한 구조, 시스톨릭 어레이를 이용한 구조 등 다양한 형태의 구조들이 제안되었다. 본 논문에서는 구조가 규칙적이고 확장이 용이하여 효율적으로 구현할 수 있고 고속 수행이 가능한 시스톨릭 어레이 구조를 이용한 Radix-4 모듈러 곱셈기 구조를 제안한다.

정수 A, B, N이 주어졌을 때 $P = A \cdot B \pmod N$을 모듈러 곱셈이라고 정의한다. 모듈러 곱셈 을 위한 알고리즘으로는 규칙적이고 간단한 Montgomery 알고리즘이 가장 널리 사용되며, 개선된 Montgomery 알고리즘들[5-7]이 제안되었고 효율적인 설계 및 구현에 대한 연구가 진행되어 왔다.

2.1 Montgomery 알고리즘

$0 < A, B < N$ 이고 $\gcd(N, R) = 1$의 관계를 가진 R이 주어질 때 Montgomery 알고리즘은 다음과 같이 계산된다.

$$\mathrm{MA}(A, B) = A \cdot B \cdot R^{-1} \bmod N$$

알고리즘은 R과 N이 서로 소수의 관계를 가질 때 어떠한 R 값에 대해서도 적용 가능하지만, 특히 R이 2의 멱승일 때는 빠르게 연산될 수 있다. 즉, $R = 2^k$이고 $A = (a_{k-1} a_{k-2} \cdots a_1 a_0)$, $B = (b_{k-1} b_{k-2} \cdots b_1 b_0)$인 두 개의 k-비트 정수가 N 보다 작고, N이 k-비트 홀수인 조건을 가진다면 위의 식은 다음과 같이 나타낼 수 있다.

$$2^{-k}(a_{k-1} a_{k-2} \cdots a_1 a_0) \cdot B (\bmod N) = 2^{-k} \sum_{i=0}^{k-1} a_i 2^i B (\bmod N)$$

$T = (a_0 2^0 + a_1 2^1 + \cdots + a_{k-1} 2^{k-1}) \cdot B$는 최상위 비트부터 시작하여 하위 비트로의 처리과정을 거쳐 계산되어 질 수 있으며 $2^{-k} A \cdot B$ 안에서 시프트 요소 2^{-k}는 합의 방향을 바꾸어 A의 최하위 비트부터 처리함으로써 $T = A \cdot B \cdot 2^{-k}$를 계산하기 위해서 이진수 더하기-이동(add-shift) 알고리즘을 이용할 수 있다.

이 알고리즘의 결과값으로 $T = A \cdot B \cdot 2^{-k}$ 얻을 수 있지만, $S = A \cdot B \cdot 2^{-k} (\bmod N)$를 계산하는 것이 우리가 관심을 가져야 할 부분이다. N은 홀수라는 조건이 있으므로 S가 홀수 일 때 N을 더하여 짝수가 되며, 짝수일 때는 그대로 놓여진다. 그러므로 S는 시프트하기 전에 항상 짝수가 되어 다음과 같이 계산될 수 있다.

$$S = S \cdot 2^{-1} \pmod N$$
$$S = 2 \cdot V \cdot 2^{-1} = V \pmod N, \quad \text{where } S = 2V$$

결과적으로 $S = A \cdot B \cdot 2^{-k} \pmod N$을 위한 Montgomery 알고리즘은 다음과 같이 정의할 수 있다.

```
MA (A, B, N)
{
    S0 = 0 ;
    for (i = 0 to k-1){
        if (Si + aiB is even)
            Si+1 = (Si + aiB)/2 ;
        else
            Si+1 = (Si + aiB + N)/2 }
    return Sk ;
}
```

이 알고리즘에서 각 반복 마다 $S_1, S_2, S_3, \cdots, S_{k-1}, S_k$가 생성되고, 모든 $i = 0, 1, 2, \cdots, k-1$에 대해서 $0 \le S_k < A + B < 2N$의 조건을 만족한다. 이와 같은 조건에서는 S_k의 최대 값이 N 보다 클 수 있기 때문에 새로운 모듈러 곱셈 계산을 하기 전에 결과값을 N과 비교하여 N보다 크다면 N을 한번 빼주어야 한다. 그러므로 $A \cdot B \cdot R^{-1} \pmod N$은 S_k와 $S_k - N$의 값들 중 하나를 가진다. Montgomery 알고리즘은 각 반복 마다 2 번의 덧셈과 S_k의 값이 N 범위 안에 들어오는지 비교해야 하는 단점이 있어 이를 보완하고 구현을 용이하게 하기 위해서 $X = A \times B$를 먼저 계산하는 개선된 Montgomery 알고리즘이 제안되었다[5]. 이 알고리즘은 각 반복 마다 한번의 덧셈으로 계산이 가능하고, $0 \le S_{2k} < N$ 값을 만족하기 때문에 후처리 계산 과정이 필요 없다. 그러

나 반복 수가 두 배로 늘어나 계산 시간의 차이는 없다. 이를 보완하여 먼저 계산되는 $2k$-비트 $X=A\times B$ 값을 상위 k-비트와 하위 k-비트로 나누어서 연산하는 알고리즘이 제안 되었다[6, 7]. 이 알고리즘들은 Montgomery 알고리즘의 단점을 없애고 k 번의 반복으로 연산을 할 수 있다. 본 논문에서는 Radix-4의 수 체계를 적용하여 두 비트씩 효율적으로 인코딩함으로써 전체 모듈러 곱셈의 수행시간을 반으로 줄이고 고속 수행이 가능한 시스톨릭 어레이 구조를 이용한 Radix-4 모듈러 곱셈기 구조를 제안한다.

2.2 제안된 Radix-4 모듈러 곱셈 알고리즘

$X=A\times B$의 하위 k-비트 X_L를 개선된 Montgomery[6, 7] 알고리즘에 적용시키면 다음과 같이 나타낼 수 있다.

$$R\cdot S_k = X_L (\mathrm{mod}N)$$
$$X = R\cdot X_M + R\cdot S_k (\mathrm{mod}N)$$
$$X_M + S_k = R^{-1}\cdot X (\mathrm{mod}N)$$

Radix-4를 이용한 모듈러 곱셈기는 곱셈의 부분곱을 반으로 감소시켜 곱셈의 계산을 $k/2$ 번으로 줄일 수 있으며 이것은 Booth 인코더기[10]를 이용하여 구현할 수 있다.

A와 B를 k-비트인 2의 보수로 확장하고 $-N < A, B < N$이라고 하자. 여기에서 고려해야 할 것은 Booth 인코더에 의해 피승수 A의 값이 $2A$의 값을 가질 수 있으며 Radix-4 알고리즘에 의해 모듈러스 N의 값이 $2N$ 값이 나올 수 있으므로 오버플로(overflow)가 생기지 않도록 A, N 값에 주의하여 부호 비트를 제외한 최상위 비트의 범위를 고려해야 한다. 값의 크기에 따라서 $(k+4)$-비트로 연산 범위를 확장하여 해결할 수 있다. $X=A\times B$로부터 $-N^2 < A\times B < N^2$가 되며, X를 k-비트인 2의 보수로 표현하면 다음과 같이 나타낼 수 있다.

$$X = -x_{2k-1}2^{2k-1} + \sum_{i=0}^{2k-2} x_i 2^i = R\cdot X_M + X_L$$

$$X_M = -x_{2k-1}2^{k-1} + \sum_{i=0}^{k-2} x_{i+k}2^i, \quad X_L = \sum_{i=0}^{k-1} x_i 2^i$$

```
R4MA (X_L, N)
{
    Y_0 = 0 ;
     for ( i = 0 to (k / 2)-1 ) {
      (m_1 m_0) = (Y_i + x_{2i+1}x_{2i}) mod 4 ;
       if ( m_0 = 0 ) {
        if ( m_1 = 0 )
          Y_{i+1} = ( Y_i + x_{2i+1}x_{2i} ) / 4 ;
        else Y_{i+1} = ( Y_i + x_{2i+1}x_{2i} + 2N ) / 4 ; }
       else {
        if ( m_1 = n_1 )
          Y_{i+1} = ( Y_i + x_{2i+1}x_{2i} + 3N ) / 4 ;
        else Y_{i+1} = ( Y_i + x_{2i+1}x_{2i} + N ) / 4 ; } }
      return Y_{k/2} ;
}
```

$Y_i + x_{2i+1}x_{2i}$의 최하위 두 비트 $m_1 m_0$과 모듈러스 N의 하위 두 번째 비트의 값 n_1을 이용하여 모듈러 정리의 값 $(0, N, 2N, 3N)$을 정할 수 있다. 개선된 Montgomery 알고리즘[6, 7]에서와 같이 R4MA의 결과는 $0 \le Y_{k/2} \le N$의 조건을 만족하며 X의 상위 k-비트 X_M는 $-N < X_M < N$과 같으므로 다음과 같은 식을 얻을 수 있다.

$$T_n = X_M + Y_{k/2} \quad \text{if } A\times B < 0$$
$$T_n = X_M + Y_{k/2} - N \quad \text{if } A\times B \ge 0$$

위와 같이 $A\times B \ge 0$이라면 $0 \le X_M + Y_{k/2} \le 2N$이므로 N을 빼주어서 $-N < T_n < N$의 조건을 만족하도록 해야 한다. 따라서 결과값 T_n은 입력과 마찬가지로 $-N < A, B < N$의 범위를 만족하므로 후처리 과정이 필요치 않다. $A\times B$의 부호는 각각의 부호 비트를 이용하여 쉽게 알아낼 수 있다. 위와 같은 모든 연산을 마친 후 모듈러스 곱셈의 결과값은 다음과 같이 나타낼 수 있다.

$$T_n = R^{-1}\cdot X (\mathrm{mod}N)$$

3. Radix-4 시스톨릭 모듈러 곱셈기 구조

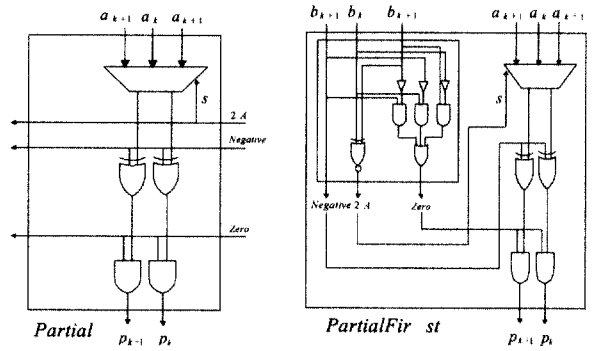
3.1 Radix-4 Booth 어레이 곱셈기.

일반적으로 곱셈기는 2차원의 어레이 구조로 구성할 수 있다. 2차원 구조는 게이트의 수가 많아 비실용적이므로 이를 1차원 구조로 시스톨릭 매핑할 수 있다. 시스톨릭 구조에서는 한 행의 시스톨릭 어레이만 구성하여 매 클럭마다 PE (Processing Element)가 수행되기 때문에 하드웨어 효율이 높아 고속 수행이 가능하다. 본 논문에서는 개선된 Booth 알고리즘을 적용하고 이를 시스톨릭 어레이로 구현하여 효율적인 곱셈을 수행하도록 한다. <표 1>은 개선된 Booth 알고리즘을 나타내며 승수의 3비트를 표와 같이 인코딩하여 곱셈의 부분곱을 생성하여 준다.

<표 1> 개선된 Booth 알고리즘

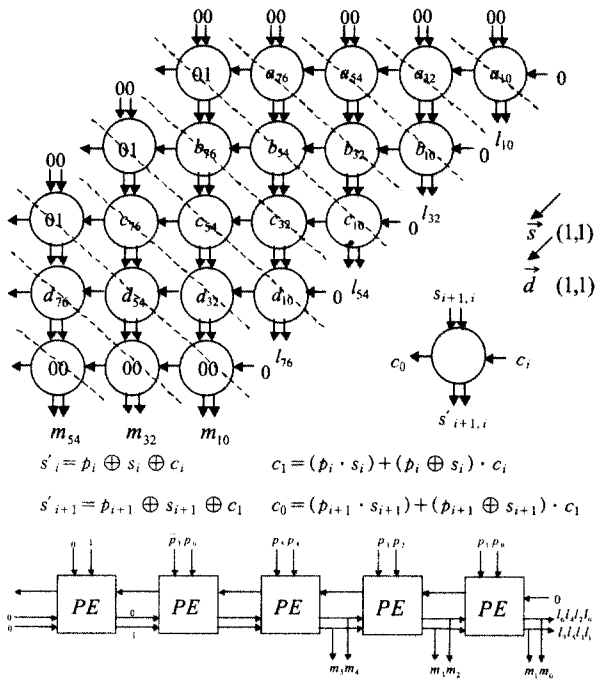
b_{K+1}	b_K	b_{K-1}	-	2A	0	*Action*
0	0	0	0	1	0	0
0	0	1	0	0	1	*A*
0	1	0	0	0	1	*A*
0	1	1	0	1	1	2*A*
1	0	0	1	1	1	-2*A*
1	0	1	1	1	1	-*A*
1	1	0	1	0	1	-*A*
1	1	1	1	1	0	0

(그림 1)의 오른쪽 그림은 <표 1>의 승수 세 비트를 처리하는 인코더기와 부분곱을 만드는 첫번째 블록이고 왼쪽 그림은 첫번째 블록에서 만들어진 인코딩 값을 사용하여 부분곱을 만드는 블록이다. 시스톨릭 구조를 이루고 있기 때문에 두 번째 블록 이후부터는 PartialFirst에서 만들어진 인코딩 값을 이용하여 부분곱을 생성할 수 있어 인코더가 필요 없고, 앞단에서 받은 인코딩 값을 이용하여 부분곱을 생성한



(그림 1) 피승수 부분곱 생성기

후 다음 블록으로 전달해주게 된다. 또한 PartialFirst 블록에서는 승수의 하위 세 비트에 대한 Booth 인코딩을 하기 위해서 승수의 LSB 오른쪽 B_{-1}의 값에 '0'을 추가하고, 피승수 $2A$의 값을 만들어주기 위한 피승수의 LSB 오른쪽 A_{-1}의 값을 '0'으로 채워주어야 한다.



$$s'_i = p_i \oplus s_i \oplus c_i \qquad c_1 = (p_i \cdot s_i) + (p_i \oplus s_i) \cdot c_i$$

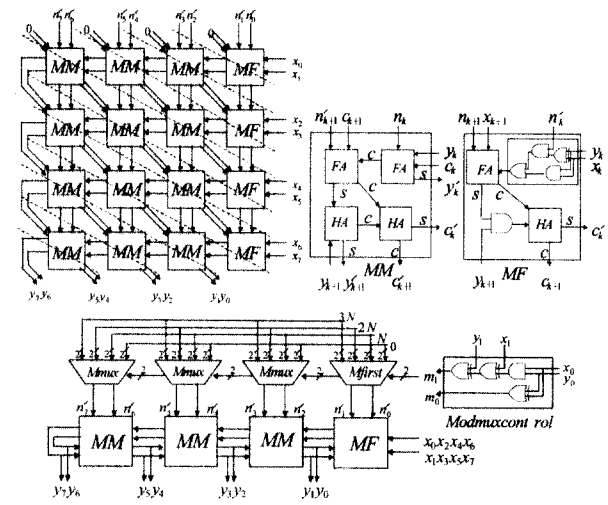
$$s'_{i+1} = p_{i+1} \oplus s_{i+1} \oplus c_1 \qquad c_0 = (p_{i+1} \cdot s_{i+1}) + (p_{i+1} \oplus s_{i+1}) \cdot c_1$$

(그림 2) Radix-4 8×8 Booth 시스톨릭 어레이 곱셈기 구조

(그림 2)는 부분곱 생성기에서 만들어진 값을 이용한 2의 보수 Baugh-Wooley 8×8 어레이 곱셈기[4]의 신호 흐름 그래프(SFG : signal flow graph)와 매핑된 시스톨릭 구조를 나타낸다. 이때 프로젝션 벡터 $\vec{p}$과 스케쥴링 벡터 $\vec{s}$은 각각 (1, 1)과 (1, 1)을 적용하였다. 각처리기(PE) 들은 FA를 기본으로 하여 설계되었다. 점선은 같은 시간에 처리 되어지는 연산들을 나타낸다. 부분곱을 계산하여 올바른 곱셈값을 얻기 위해서는 부호 비트를 확장해야 하는데 하드웨어의 크기를 줄이고 연산의 효율성을 높이기 위해서 부호확장 제거기법[4]을 사용하였다. 각 처리기에 부분곱 생성기에서 만들어진 부분곱 두 비트가 입력이 되어 덧셈연산을 수행한다. a, b, c, d는 인코더기의 출력으로 생성되는 부분곱을 나타내며 (그림 2) 하단의 시스톨릭 구조에서는 p_i로 표시되었다. 또한 m은 곱셈 X의 상위 비트, l은 하위 비트를 나타낸다. 모든 연산은 두 비트씩 처리가 되기 때문에 곱셈의 하위 비트는 (그림 2)의 오른쪽 첫번째 PE에서 두 클럭마다 두 비트씩 출력이 되고, 상위 비트가 출력이 될 때는 매 클럭마다 두 비트씩 연산을 하기위해서 왼쪽으로 PE를 이동하면서 연산이 수행된다.

3.2 Radix-4 8×8 모듈러 연산기(modular reduction) 구조

(그림 3)은 R4MA 알고리즘를 연산하는 부분으로서 곱셈기에서 만들어진 하위 k-비트인 X_L의 값에 대한 모듈러 연산을 수행한다. 모듈러 연산블록 역시 FA, HA를 기본으로 설계 되었다. 우선 모듈러 N 값을 Radix-4에 적용시키기 위해 0, N, $2N$, $3N$ 값을 선택해야 하는데 이를 위한 블록이 Modmuxcontrol이며 $(Y_i + x_{2i+1}x_{2i}) \pmod 4$의 결과값 두 비트 m_1m_0을 멀티플렉서 선택신호로 Mfirst에 입력하고 매 클럭마다 Mmux로 전달해 준다.

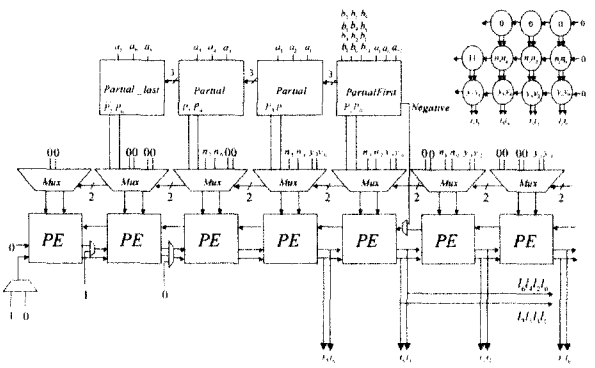


(그림 3) Radix-4 모듈러 연산기구조

MF(Mod First)에서는 곱셈기에서 출력되는 X_L의 두 비트, 모듈러 N의 두 비트, 첫 번째 MM 단에서 오는 합의 두 비트로 연산하여 캐리 두 비트를 모듈 MM으로 되돌려 준다. MM(Mod middle)에서는 오른쪽 MM 또는 MF에서 넘어오는 캐리 두 비트와 왼쪽 MM 단에서 출력되는 합의 두 비트, N의 두 비트를 연산해서 오른쪽 MM 단에 캐리 비트와 왼쪽의 MM 단에 합의 두 비트 값을 만들어준다. Radix-4 모듈러 블록에 곱셈기에서 만들어진 X_L의 값이 입력되어 k+1 클럭 후에 $Y_{k/2}$의 LSB y_0, y_1 값이 나오기 시작함과 동시에 곱셈기에서는 X의 상위 k-비트 X_M의 LSB 값이 나오므로 k+2 클럭이 지나면서부터 최종 결과값인 T_n의 연산을 시작 할 수 있다.

3.3 T_n의 연산 구조

(그림 4)의 오른쪽 상단에 SFG는 곱셈연산의 다음 단계를 나타내는 것이다. Booth 인코더기와 곱셈기의 연산으로 만들어진 곱셈 결과값의 하위 X_L은 모듈러 연산블록을 거치면서 $Y_{k/2}$ 값이 생성된다. k+2 클럭이 지나면서 나오는 곱셈의 상위 X_M 값에 $Y_{k/2}$를 더해주는 단계를 거쳐 $X > 0$ 이면 모듈러 N을 빼주고, $X < 0$ 이면 '0'를 더하여 최종값인 T_n의 값이 출력되기 시작한다. 이때 모듈러 N의 상위 k-1, k-2 비트 자리에 "11"을 넣어 준 것은 곱셈기에서 곱셈의 상위 X_M의 출력을 (k-3)-비트까지 고려함으로 X가 '0'보다 크다면 X_M의 부호 비트를 포함한 상위 두 비트가 "00"이 되고, 따라서 N을 빼주어야 하기 때문에 N의 상위 두 비트는 보수가 되어 "11"이 된다. 만일 곱셈 X가 '0'보다 작다면 N은 '0'이 되고, X의 상위 k-비트인 X_M의 부호 비트를 포함한 상위 두 비트가 음수가 됨으로 역시 "11"이 되어 올바른 계산이 된다.



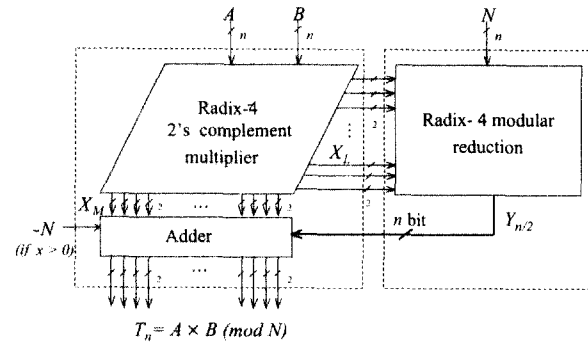
(그림 4) 8×8 Radix-4 Booth 곱셈기 및 T_n의 연산 구조

(그림 4)는 Booth 인코더기와 곱셈기 그리고 T_n 계산을 위한 구조를 나타낸다. Booth 인코더를 이용하여 부분곱과 다음 Partial 블록 부분곱의 제어 신호를 만들어주는 Partial First 블록에서 피승수의 값이 $-A$, $-2A$의 값을 가질 경우 2의 보수를 취하기 위해서 피승수 값의 보수를 취한다. 이때 올바른 2의 보수값을 만들어주기 위해서 '1'을 더해 주어야 하는데 곱셈기에 Negative라는 신호를 보내어 '1'을 더하여 주면 된다. 또한 SFG에서와 같이 알맞은 시간에 제어 신호를 올바르게 만들어 주기 위해서 카운터의 값을 이용하여 부분곱과 $Y_{k/2}$, N 값을 PE에 넣어주어 T_n 값을 연산하게 된다.

4. 시뮬레이션 및 결과분석

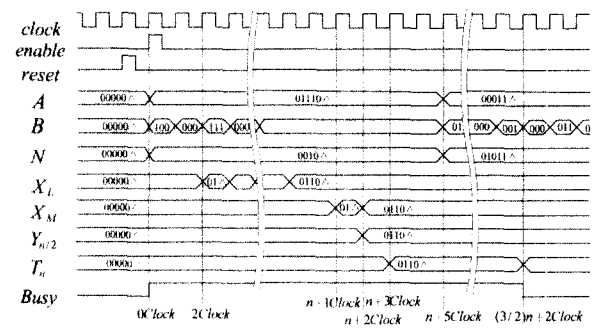
4.1 전체적인 시스톨릭 구조 및 결과분석

제안된 구조는 Verilog HDL로 모델링 되었으며 Modelsim SE-EE 5.4e 환경에서 시뮬레이션되어 그 동작이 검증되었다. (그림 5)는 전체 블록 다이어그램을 보여준다. Enable 신호가 인가된 후 모듈러 곱셈이 수행될 때 피승수 A와 모듈러스 N의 값은 n-비트씩 입력되고, 승수 B는 3 비트씩 입력된다. 전체적인 모듈러 곱셈기는 (그림 5)와 같이 개념적으로 두 개의 블록으로 나누어 볼 수 있다.



(그림 5) 개념적인 Radix-4 모듈러 곱셈기 블록도

우선 왼쪽 곱셈기 블록에는 Booth 인코딩 블록이 포함되어있고 이 블록에서 부분곱을 생성하여 연산한다. 곱셈결과 생성되는 하위 두 비트씩 모듈러스 연산 블록으로 보내어 $Y_{k/2}$을 연산하고 곱셈기에서 생성하는 곱셈의 상위 비트들과 더하여 마지막 결과값 T_n을 생성한다. 이때 곱셈의 상위 비트와 모듈러스 연산된 $Y_{k/2}$ 값을 더하는 덧셈기 블록의 실제적인 계산은 시스톨릭 곱셈기 내부의 덧셈기에서 처리할 수 있으므로 추가되는 하드웨어는 없다. 본 논문에서 제안한 Radix-4 구조를 이용하여 곱셈의 부분곱을 반으로 감소시켰으나 Radix-4 Booth 곱셈기 내부에 인코딩하는 블록이 첨가되어 Radix-2에 시스톨릭 구조에 비해서 하드웨어 복잡도는 다소 증가되었다.



(그림 6) 제안된 Radix-4 모듈러 곱셈기의 타이밍 다이어그램

제안된 구조는 하나의 모듈러 곱셈만 고려한다면 모듈러 계산을 완전히 마치기 위하여 (3/2)n+2 클럭이 필요하다. 그러나 (그림 6)에서 보여 주는 것과 같이 시스톨릭 구조를 갖기 때문에 n+5 클럭이 지난 후부터 다음 모듈러 곱셈의 상위 비트 계산을 시작할 수 있으므로약 평균 n 사이클에 하나의 모듈러 곱셈을 수행할 수 있다. 즉, Radix-4 수체계를 사용하여 두 비트를 한번에 코딩하여 처리함으로써 곱셈의 부분곱을 반으로 줄였기 때문에 성능이 향상되었으며 시스톨릭 구조를 적용하여 고속수행이 가능하다. 또한 제안된 구조는 간단한 FA를 기반으로 구성되어 간단하게 설계할 수 있다. 시스톨릭 구조로 인해 하나의 연산이 끝나기

전에 다음 연산을 수행 할 수 있으며 규칙성과 확장성을 갖기 때문에 효율적인 VLSI 구조로 설계가 용이한 장점을 가지고 있다.

현재 제안된 시스톨릭 구조의 하드웨어 효율은 50%이다. 따라서 홀수 번째 클럭과 짝수 번째 클럭에서 두 개의 모듈러 곱셈을 각각 중첩(interleaving)하여 연산할 수 있으며 RSA 암호화시스템과 같이 연속된 모듈러 곱셈을 수행할 경우 하나의 지수 비트를 처리할 경우 두 개의 모듈러 곱셈을 해야 하므로 이 경우 제안된 구조에서는 한 번에 처리할 수 있다. 따라서 하나의 모듈러 곱셈을 평균 $n/2$ 클럭 싸이클에 처리할 수 있다. 전체구조를 살펴보면 DFF, FA, 그리고 MUX(2-to-1 MUX로 환산하였을 경우)가 각각 $7n$, $2.5n$, $4n$개 정도 필요하며 Booth 인코더 블록이 필요하다. 또한 약간의 제어 블록과 정해진 시간에 필요한 제어 신호를 만들기 위하여 카운터가 필요하다. 제안된 구조에서 512-비트를 기준으로 단일의 모듈러 곱셈을 수행하면 약 $(1.5\times512)+2=770$ 클럭이 소요된다. 그러나 RSA의 암호화 또는 복호화를 연산하기 위하여 모듈러 곱셈을 중첩한다면 단일의 모듈러 곱셈의 수는 약 $(1\times512/2)+2=258$ 클럭이 소요되며, 512-비트 RSA를 수행하기 위해서는 평균 $1.5\times512\times1\times258=0.2M$ 클럭, 최고 $2\times512\times1\times258=0.27M$ 클럭이 소요된다. <표 2>에서 기존 구조들과 비교해 볼 때, 제안된 모듈러 곱셈기 구조는 적은 하드웨어를 이용하여 높은 성능을 얻을 수 있음을 보여준다.

〈표 2〉 n-비트 모듈러 곱셈기에 대한 하드웨어 및 성능 분석

Architectures	Hardware Requirement				Time
	DFF	FA	MUX	RAM	clock
Yang[7]	$12n$	$2n$	$8n$	0	n
Sheu[8]	$10n$	$3.2n$	$9n$	0	$6n/5$
Chiang[9]	$6.7n$	$1.5n$	$2n$	10*512	$n/2$
Leu[10]	$12.3n$	$2n$	$3n$	0	$n/2$
Ours	$7n$	$2.5n$	$4n$	0	$n/2$

5. 결 론

본 논문에서는 Montgomery 알고리즘을 기반으로 효과적으로 연산을 수행할 수 있는 Radix-4 모듈러 알고리즘과 그를 위한 고속 시스톨릭 구조를 제안하였다. Booth 인코딩을 이용하여 부분곱을 효율적으로 연산하기 위한 구조를 설계하였으며, 제안된 Radix-4 모듈러 곱셈 알고리즘은 두 수의 곱을 상, 하위 비트로 나누어서 Radix-4 수 체계로 하위 k-비트만 모듈러 처리하고 상위 k-비트를 더하여 올바른 결과값을 얻음으로써 계산시간을 줄일 수 있었다. 또한 시스톨릭 구조를 적용함으로써 규칙적이고 확장성 있게 설계할 수 있었다. 기존의 다른 구조들과 비교하여 볼 때 상대적으로 적은 하드웨어들을 사용하여 높은 수행 속도를 보여주었다. 이 Radix-4 모듈러 곱셈기를 RSA 암호화시스템에 적용하여 설계하면 효율적인 구조를 기대할 수 있다.

참 고 문 헌

[1] R. L. Rivest, A. Shamir and L. Adelman, "A Method for Obtaining DigitalSignatures and Public-Key Cryptosystem," Commun. ACM, Vol.21, No.2, pp.120-126, Feb., 1978.

[2] P. L. Montgomery, "Modular Multiplication Without Trivial Division," Math. Comput, Vol.44, pp.519-521, 1985.

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

[4] K. Hwang, "Computer Arithmetic : Principles, Architecture, and Design," Wiley, 1979.

[5] P. S. Chen, S. A. Hwang and C. W. Wu, "A systolic RSA Public Key Cryptosystem," in Proc. IEEE Int. Symp. Circuits Syst, pp.408-411, May, 1996.

[6] 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 Trans. VLSI Syst, Vol.7, No.2, June, 1999.

[7] C. C. Yang, T. S. Chang and C. W. Jen, "A New RSA Cryptosystem HardwareDesign Based on Montgomery's Algorithm," IEEE Trans. Circuits and Systems II, Vol.45, No.7, pp.908-913, July, 1998.

[8] J. L. Sheu, M. D. Shieh, C. H. Wu and M. H. Sheu, "A PipelinedArchitecture of Fast Modular Multiplication for RSA Cryptography," Proc. IEEE ISCAS, Vol.2, pp.121-124, 1998.

[9] J. S. Chiang and J. K. Chen, "An Efficient VLSI Architecture for RSA Public-Key Cryptosystem," Proc. IEEE Int. Symp. on Circuits and Systems, pp.496-499, 1999.

[10] J. J. Leu and A. Y. Wu, "Design Methodology for Booth-encoded Montgomery Module Design for RSA Cryptosystem," Proc. IEEE ISCAS, pp.357-360, 2000.

[11] C. W. Chiou, T. C. Yang, "Parallel Modular Multiplication with Table Look-up," Int. J. Computer Math., Vol.69, pp. 49-55, 1997.

박 태 근

e-mail : parktg@catholic.ac.kr

1985년 연세대학교 전자공학(학사)

1988년 Syracuse Univ. 컴퓨터공학(석사)

1993년 Syracuse Univ. 컴퓨터공학(박사)

1991년~1993년 Coherent Research Inc. VLSI 설계 엔지니어

1994년~1998년 현대전자 System IC 연구소 책임연구원

1998년~현재 가톨릭대학교 정보통신전자공학부 부교수

관심분야 : VLSI 설계, CAD, 병렬처리

조 광 원

e-mail : luischo@catholic.ac.kr

2002년 가톨릭대학교 정보통신공학(학사)

2002년~현재 가톨릭대학교 컴퓨터공학 석사과정

관심분야 : VLSI 설계, 신호처리, 암호화 시스템