RSA 공개키 암호화시스템의 효율적인 Radix-4 시스톨릭 VLSI 구조

박 태 근*

Efficient Radix-4 Systolic VLSI Architecture for RSA Public-key Cryptosystem

Taegeun Park

요 약

본 논문에서는 RSA 공개키 암호화 알고리즘을 위한 효율적인 Radix-4 시스톨릭 VLSI 아키텍처를 제안하였다 모듈러 곱셈 알고리즘의 이터레이션 단순화와 효율적인 시스톨릭 매핑으로 제안된 구조는 n-비트 모듈러 멱승 연산을 n^2 클럭 싸이클에 수행한다. 각 지수 처리 단계에서 두 개의 모듈러 곱셈, M_i와 P_i는 중첩되어 연산되며 따라서 제안된 하드웨어의 이용도(hardware utilization)는 100%이다 또한 RSA 암호화를 위한 총 모듈러 곱셈의 횟수를 줄이기 위하여 지수를 Radix-4 SD(Signed Digit) 수체계를 이용하여 인코딩하였다 이로 인하여 지수의 NZ(non-zero) 디지트가 약 20% 감소되어 성능이 향상되었다. 기존의 방법들과 비교하였을 때, 제안된 구조는 비교적 적은 하드웨어를 사용하여 우수한 성능을 보였으며 개선된 Montgomery 알고리즘을 바탕으로 한 제안된 구조는 지역성, 규칙성, 확장성 등으로 VLSI 구현에 적합하다.

Key Words RSA, Cryptosystem, Modular multiplication, Systolic

ABSTRACT

In this paper, an efficient radix-4 systolic VLSI architecture for RSA public-key cryptosystem is proposed Due to the simple operation of iterations and the efficient systolic mapping, the proposed architecture computes an n-bit modular exponentiation in n^2 clock cycles since two modular multiplications for M_i and P_i in each exponentiation process are interleaved, so that the hardware is fully utilized We encode the exponent using Radix-4 SD (Signed Digit) number system to reduce the number of modular multiplications for RSA cryptography Therefore about 20% of NZ (non-zero) digits in the exponent are reduced Compared to conventional approaches, the proposed architecture shows shorter period to complete the RSA while requiring relatively less hardware resources The proposed RSA architecture based on the modified Montgomery algorithm has locality, regularity, and scalability suitable for VLSI implementation

I. 서 론

정보의 경제적 가치는 통신망의 확충과 함께 증가되고 있으며 그에 따라 정보통신은 산업구조의 핵심분야로 등장하고 있다. 정보 교류의 양적 팽창은 보다 광범위하고 효율적인 통신망의 수요를 낳고 있으며, 따라서 보다 빠르고 대용량의 고품질의 정보 서비스체계가 요구되고 있다 그런데 이와 더불어 컴퓨터 네트워크 및 정보에 대한 보안이 사회적인 시급한 문제로 부각되고 있다 정보 암호분야는 최근 단순히 데이터 암호화만이 아니라 상호 인증을 통하여 보안이 필요한 다양한 분야들로 그의 응용이 확대되고 있

* 가톨릭대학교 정보통신전자공학부 부교수
논문번호 KICS2004-06-038, 접수일자 2004년 6월 15일

다 1977년 제안된 RSA 암호화 시스템은 암호화와 전자인증 두 가지를 제공하며 현재 가장 대표적인 암호화 방법으로 그의 응용분야가 확대되고 있다[1] RSA 암호화 알고리즘은 큰 정수에 대한 소인수분해의 어려움에 근간을 두고 있다[11] 따라서 상대적으로 긴 키 길이(512비트 이상)가 권장되며 이에 따라 암호 및 복호화과정의 실시간 처리를 어렵게 한다. RSA 암호화 시스템의 주된 연산은 모듈러 멱승이고 가장 보편적인 처리 방법으로는 "square-and multiply"가 있으며 이는 반복된 모듈러 곱셈으로 가능하다

모듈러 멱승 연산을 하기 위해서는 많은 계산이 필요하며 순수한 소프트웨어로 연산하는 방법은 실시간 처리가 요구되는 응용분야에는 성능의 문제 때문에 적용하기 어렵다 따라서 512-비트 이상의 큰 정수에 대한 모듈러 멱승을 실시간 내에 수행하기 위해서는 모듈러 멱승을 효율적으로 처리할 수 있는 전용 연산 장치를 설계, 구현하여 기본 연산인 모듈러 곱셈을 효율적으로 수행할 수 있는 회로를 하드웨어로 구현해야 한다[3][5][6][7][8] 그러므로 효율적인 VLSI 구조 및 설계가 RSA 암호화시스템을 구현하는 데에는 필수적인 요소이다

RSA 암호화시스템은 반복적인 모듈러 지수 연산을 하며, 지수연산은 모듈러 곱셈을 기본연산으로 한다 현재까지 효율적인 모듈러 곱셈 알고리즘 및 구조가 제안되었다 Chiang은 곱셈연산의 회수를 줄이기 위하여 지수 값에 대한 멀티 비트 인코딩 방법을 제안하였다[7] Sheu는 각 세그먼트에서 덧셈과 뺄셈간의 데이터 상관관계를 없애주기 위하여 분할 방법을 적용하였다[5] 이로 인하여 현재 세그먼트와 이전 세그먼트의 나머지 계산 과정에서 파이프라인을 적용할 수 있었다. 개선된 Montgomery 알고리즘이[3] 제안되었는데, 이는 모듈러 곱셈 과정에서 곱셈과 모듈러 감소 과정을 분리하여 각 이터레이션 당 한번의 덧셈만 필요한 구조로 변환하였다 하지만 이터레이션의 수가 두 배로 증가하였기 때문에 총 수행시간은 변함이 없다 Montgomery 알고리즘의 이터레이션 수를 감소하기 위한 다양한 알고리즘들이 제안 되었다[4][6]

본 논문에서는 Radix-4 수 체계를 이용하여 개선된 모듈러 곱셈 알고리즘을 바탕으로 규칙적이고 확장성 있는 RSA 암호화시스템을 위한 효율적인 VLSI 구조를 제안한다 제안된 모듈러 곱셈 알고리즘은 효율적인 시스톨릭 구조로 매핑되었으며 그의 구조가 규칙적이고 임의의 n-비트로 확장이 용이하므로 VLSI 설계에 적합하다. 또한 RSA 암호화 과정을 처리할 때 모듈러 곱셈 연산의 횟수는 지수에 포함된 NZ(Non-zero) 비트의 수에 관계된다 따라서 본 논문에서는 지수의 NZ 디지트를 최소화 하기 위한 Radix-4 SD(Signed Digit) 인코딩 알고리즘을 제안하였다. 제안된 구조는 시스톨릭 인터리빙의 적용과 이터레이션 수의 감소로 인하여 n-비트 모듈러 지수 연산를 n2 클럭 사이클에 수행할 수 있다 이에 따라 제안된 구조는 100%의 하드웨어 이용도(hardware utilization)를 보여준다

본 논문의 구성은 다음과 같다. 2장에서는 RSA 암호화 시스템과 그의 핵심 알고리즘인 모듈러 곱셈 및 지수 연산 알고리즘에 대한 고찰과 본 논문에서 제안하는 Radix-4 모듈러 곱셈 및 효율적인 지수연산을 위한 SD 인코딩 알고리즘을 설명한다. 3장에서는 모듈러 곱셈기, 모듈러스 처리블럭 및 RSA 암호화시스템 구조에 대하여 설명한다. 시뮬레이션 및 분석은 4장에서 다루고 마지막으로 5장에서 결론을 맺는다.

Ⅱ RSA 암호화 알고리즘

현재 다양한 정보에 대한 보안을 위해 널리 이용되고 있는 RSA 암호 알고리즘에 대한 간단한 내용은 다음과 같다[11] 대략적으로 비슷한 크기의 임의의 큰 두 소수 p, q에 대하여 $n = p \cdot q$, $\phi = (p-1)(q-1)$의 관계에 있는 n, ϕ를 구한다 또한, $1 < e < \phi$ 와 $\gcd(e, \phi) = 1$을 만족하는 랜덤 정수 e를 선택하고 확장된 Euclidean 알고리즘을 이용하여 $e \cdot d \equiv 1 \pmod{\phi}$를 만족하는 특정한 정수 $d(1 < d < \phi)$를 구한다. 이 때 e를 공개키, d를 비밀키라고 한다

위에서 생성된 두 키 e와 d를 이용하여 암호화와 복호화를 하기 위해서는 다음과 같은 연산이 필요하며 이 때, C를 암호문(cipher text), M을 평문(plain text)라고 부른다

$$C = M^e \pmod{n}, \qquad M = C^d \pmod{n}$$

위의 모듈러 지수연산은 모듈러 곱셈의 반복적인 수행으로 처리할 수 있다 따라서 RSA 암호화 알고리즘을 효율적으로 수행하기 위해서는 효율적인 모듈러 곱셈기 설계가 필수적이라 하겠다

2 1 Montgomery 모듈러 곱셈 알고리즘

A, B는 n-비트 정수이고, N은 n-비트홀수 일 때, $0 < A, B < N$ 의 조건을 가진다면 Montgomery 모듈러 곱셈 알고리즘[2]은 다음과 같다

```
MM(A, B, N)
{S_0=0 ,
  for (i=0 to n-1){
   if (S_i + a_iB is even)
       S_{i+1} =( S_i + a_iB)/2,
   else
       S_{i+1} =( S_i + a_iB + N)/2}
return S_n ; }
```

위의 알고리즘에서 각 이터레이션은 세 개의 n-비트의 덧셈과 나눗셈으로 연산 되어진다 이것은 두 번의 n-비트 덧셈과 오른쪽 쉬프트 연산으로 구현 할 수 있다. 여기서 $2^j S_j \equiv (\sum_{i=0}^{j-1} a_i 2^i)B \pmod N$ 이고 $0 \le S_j \le N+B<2N$ 이다 이 때 $R=2^n$이라고 하면 $R \cdot S_n = A \cdot B \pmod N$이 된다. 그러나 S_n 의 값이 초기의 $(0, N)$ 범위를 넘어 $(0, 2N)$의 값을 가지기 때문에 S_n이 N보다 크다면 다음 모듈러 곱셈을 하기 전에 S_n에서 N을 빼는 후처리 과정이 필요하다.

개선된 Montgomery 알고리즘[4]은 $X = A \times B$를 미리 계산함으로써 한 번의 덧셈과 후처리 과정이 필요하지 않다. 그러나 이터레이션이 두 배로 늘어나기 때문에 기존의 알고리즘에 비해 계산시간은 같아진다 이를 보완하기 위하여 먼저 계산되는 $2n$-비트 $X(= A \times B)$ 값을 상위 n-비트와 하위 n-비트로 나누어서 연산하는 알고리즘이 제안 되었다[4][6] 이 알고리즘은 Montgomery 알고리즘의 단점을 없애고 n 번의 반복으로 연산을 할 수 있다 또한 상대적으로 간단한 PE(processing element)로 설계할 수 있으므로 시스톨릭 어레이로 매핑하기에 용이하고 고성능 처리가 기대된다

2 2 제안된Radix-4 모듈러 멱승 알고리즘

Radix-4 모듈러 곱셈기는 하위 부분곱인 X_L을 얻기 위한 $n/2$ 번으로 곱셈을 처리하며 이것은 Booth 인코더기[10]를 이용하여 구현된다 n-비트 A와 B의 범위를 $-N < A, B < N$ 이라고 하자 Booth 인코더에 의해 피승수 A의 값이 $2A$의 값을 가질 수 있으며 Radix-4 알고리즘에 의해 모듈러스 N의 값이 $2N$ 값이 나올 수 있으므로 오버플로우가 생기지 않도록 부호 비트를 제외한 최상위 비트의 범위를 고려해야 하며 값에 따라서 비트를 확장하여 해결한다. $X = A \times B$로부터 $-N^2 < X = A \times B < N^2$가 되며, X를 n-비트 2의 보수로 표현하면 다음과 같이 나타낼 수 있다.

$$X = -x_{2n-1}2^{2n-1} + \sum_{i=0}^{2n-2} x_i 2^i = RX_M + X_L$$

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

```
R4MM(X_L, N)
{Y_0= 0 ,
   for ( i=0 to n/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 ,
      elseY_{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 ,
       elseY_{i+1} = ( Y_i + x_{2i+1}x_{2i} + N) / 4 , }
 return Y_{n/2} , }
```

위의 알고리즘은 제안된 Radix-4 모듈러 곱셈 알고리즘을 나타내며, X의 하위 n-비트 X_L에 대한 모듈러 연산 $R \cdot Y_{n/2} = X_L \pmod N$은 $0 < Y_{n/2} < N$ 범위를 갖는다 또한 X의 상위 n-비트 X_M 의 범위가 $-N < X_M < N$ 이고 $-R(N+1) < N^2 - X_L < R \cdot X_M < N^2 - X_L < R \cdot N$이기 때문에 제안된 알고리즘의 지수연산 마지막 $T_n \equiv R^{-1} \cdot X \pmod N$은 다음과 같이 계산 되어진다.

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

위와 같이 $A \times B \ge 0$ 이라면 $-N \le X_M < N$과 $0 < Y_{n/2} < N$로부터 $0 \le X_M + Y_{n/2} \le 2N$이므로 T_n에서 N을 빼주어서 $-N < T_n < N$의 범위 내로 결과값을 보정해 주어야 하며 T_n은 입력과 같은 범위를 가지므로 후처리 과정이 필요치 않다. 위의 알고리즘은 각 반복마다 곱셈 결과 X의 두 비트만이 필요하기 때문에 X_L의 최하위 두 비트가 생성되자마자 시작할 수 있다 X_L의 모든 비트가 차례로 R4MM()를 거쳐서 $Y_{n/2}$ 의 최하위 비트가 나오는 시점과 X_M의 최하위 비트가 나오는 시점이 같으므로 T_n 은 지연시간 없이 최하위 비트부터 차례로 계산되어 질 수 있으며, 이것은 다음에 설명되어질 시스톨릭 어레이에 의해서 효율적으로 구현될 수 있다. T_n 을 계산하기 위해서는 n 번의 이터레이션이 소요되고, 각 이터레이션마다 한 번의 덧셈 연산이 필요하다

```
ME(M,E,C,N)
{
  C = 1,
 for ( i = n-1  to  0 )                    {
    C = C · C (mod N)
    if  e*i = 1     C = C · M (mod N) )
return  C ,
}
```

M^E mod N 의 지수연산을 하기 위한 방법에는 *L-R*(Left to Right) 알고리즘과 *R-L*(Right to Left) 알고리즘이 있다. 지수 $E = e_{n-1}e_{n-2} \cdots\cdots e_1e_0$ 와 같이 나타낼 수 있는데 *L-R* 알고리즘은 지수의 최상위 비트(MSB)부터 처리하여 해당 비트가 '1'이면 제곱과 곱셈을 연산하고 '0'이면 제곱만 연산하며 반대로 *R-L* 알고리즘은 최하위 비트부터 처리한다. 위에 기술된 ME()는 *L-R* 알고리즘을 나타낸다

$C=R^2$ (mod N)일 때 위의 모듈러 멱승 알고리즘은 반복적으로 P_n을 생성하고 이 값은 M^E (mod N)의 결과 값을 나타내게 된다 모듈러 지수 연산 과정에서 중간 결과 값 M_i , P_i은 $(-N, N)$ 범위를 가진다 최종 결과인 P_n은 구간 $(0, N)$ 안에 들어와야 하므로 $P_n < 0$인 경우에는 P_n에 N을 더해주어야 올바른 값을 얻을 수 있다

2 3 Radix-4 지수 인코딩 방법

앞의 모듈러 멱승 알고리즘에서 볼 수 있듯이 RSA 암호화 과정에서 수행되는 총 모듈러 곱셈의 수는 비트가 '0'일 경우는 제곱연산(S) 만이 필요하며 비트가 '1'일 경우는 제곱연산(S)과 곱셈(M)이 필요하므로 지수가 n-비트일 경우 $n + E_1$이며 평균 $3/2n$(최대 $2n$) 번의 모듈러 곱셈이 필요하다 이때 n은 지수의 비트 수 이고 E_1은 지수 E의 이진 표현에서 '1'의 비트 수 이다 따라서 RSA 암호화 알고리즘을 수행하기 위하여 많은 양의 계산이 필요하게 되며 전체 수행시간에도 영향을 주게 된다 본 논문에서는 지수를 Radix-4로 적절히 인코딩하여 처리해야 하는 모듈러 곱셈의 횟수를 단축함으로써 성능을 개선하였다

우선 지수를 비트단위로 처리할 경우와 두 비트씩 묶어서 처리할 경우의 필요한 곱셈 횟수를 아래의 표에 비교하였다 즉, 두 비트씩 묶어서 처리할 경우에는 제곱연산의 횟수는 동일하지만 곱셈은 M, M^2, M^3의 값을 직접 곱함으로써 횟수를 줄일 수 있다. M^2은 M 값을 쉬프트하면 되므로 쉽게 구할 수 있으며 M^3의 값만 미리 계산하여 저장해 놓으면 된다 이로 인한 평균 곱셈의 횟수는 $11/8n$이다

표 1 한 비트와 두 비트로 묶어 처리할 경우의 필요한 계산량

	비트 단위로 처리할 경우	두 비트씩 묶어서 처리할 경우
00	S, S	S, S
01	S, S, M	S, S, M
10	S, M, S	S, S, M^2
11	S, M, S, M	S, S, M^3

본 연구에서는 곱셈의 횟수를 좀 더 줄이기 위하여 SD(Signed Digit) 수체계로 지수를 인코딩하는 방법을 적용하였다 SD 수체계는 하나의 수를 표현하기 위해서 적어도 하나 이상의 표현 방법이 존재하며, 캐리의 전파를 제한하는 고속 병렬 덧셈기 설계에 적용될 수 있다 [10] 이러한 수체계를 바탕으로 임의의 수에 대한 SD로 표현된 수는 각 디지트가 부호를 갖으므로 데이터를 저장하기 위한 공간이 커지고 인코딩및 디코딩이 필요하다는 단점이 있다. 그러나 최소의 NZ 수를 갖는 CSD(Canonic Signed Digit)로 표현할 경우, 상수 곱셈이 많이 적용되는 신호처리용 필터 설계에서 부분곱의 수가 줄기 때문에 효율적인 연산을 기대할 수 있다[12]. 또한 CSD의 효율적인 표현은 모듈러 지수 연산의 수행 속도를 개선하며 Radix-2 RSA 암호화 알고리즘에 적용한 연구가 진행되었다[9].

지수 E 의 값을 SD 수체계의 이진수로 표현하면 다음과 같이 나타낼 수 있다

$$E = \sum_{i=0}^{n-1} e_i^* 2^i \qquad where \qquad e_i^* \in \{-1,0,1\}$$

$$E = \sum e_i^{'} 2^i - \sum e_i^{''} 2^i$$

$$where \quad e_i^{'} = e_i^* \quad if \quad e_i^* = 1 \quad and$$

$$e_i^{''} = -e_i^* \quad if \quad e_i^* = -1$$

연속된 $111\cdots11$ $(2^i+2^{i-1} \cdots 2^j, i > j)$를 $1000 \quad 0\bar{1}$ $(2^{i+1} - 2^j, i > j)$와 같이 부호비트 '$\bar{1}$'를 이용하여 NZ 비트 수를 줄일 수 있다 ($\bar{1}$ = -1) NZ 비트 수를 줄인다는 것은 모듈러 지수 연산에 있어서 모듈러 곱셈의 수를 줄이는 중요한 요소이다

모듈러 곱셈의 지수연산을 Radix-2에서 Radix-4로 확장하기위하여 e_i를 두 비트씩 묶어서 Radix-4 e_i^* 로 나타내면 다음과 같이 표현할 수 있다.

$E = e_{(n/2)-1} e_{(n/2)-2} \cdots e_1 e_0$, $\quad e_i \in \{0, 1, 2, 3\}$

$E = e^*_{(n/2)-1} e^*_{(n/2)-2} \cdots e^*_1 e^*_0$, $\quad e^*_i \in \{-3, -2, -1, 0, 1, 2, 3\}$

표 2 Radix-4 SD 인코딩 방법 (i = 0 *to* $(n/2)$-1)

Present state(e_i)		Next state(e^*_i)	
i+1	i	i+1	i
3	0	3	0
3	1	*C*, 0	-3
3	2	*C*, 0	-2
3	3	*C*, 0	-1

표 2는 Radix-4 SD 수체계에서 NZ 디지트를 최소화하기 위한 인코딩 방법을 나타낸다. 하위 비트에서 상위비트로 2비트씩 묶어 Radix-4 수체계에서 i+1 번째의 값이 '3'이고 i 번째의 값이 '1', '2', '3' 일 때 위의 표를 적용하면 i+1 번째의 값이 '0'이 되고 i 번째의 값이 '-3', '-2', '-1'되며 i+2 번째의 값에 C(Carry) 즉, '1'이 더해져 상위비트에 대한 인코딩에 적용된다 제안된 Radix-4 SD 수체계에서의 지수 인코딩 방법을 적용하여 NZ 디지트수의 변화를 표3에 분석하였다 여러 가지 비트 수에 대하여 랜덤하게 만들어진 임의의 수를 위에서 제안한 방법으로 인코딩하여 평균 NZ 디지트 수를 비교하였다

표 3 Radix-4 인코딩 후의 NZ 디지트 수 비교

비트 수 (디지트 수)	인코딩 전 *NZ* 디지트의 평균 수	인코딩 후 *NZ* 디지트의 평균 수	*NZ* 감소율
32(16)	11	9	18 2%
64(32)	23	18	21 7%
128(64)	46	37	19 6%
256(128)	95	77	18 9%
512(256)	197	162	17 8%

표 3에서 제안된 Radix-4 인코딩 방법을 적용할 경우, 인코딩 전에비하여 인코딩 후에 평균 NZ디지트 수는 약 18% - 21% 정도 감소되었으며, Montgomery 알고리즘을 바탕으로 한 모듈러 지수 연산에 제안된 인코딩 방법을 적용한다면 약 20% 가까운 모듈러 곱셈을 줄일 수 있다. 지수 E를 SD Radix-4 수체계로 인코딩하여 새로 구성된 *L-R* 알고리즘은 아래에 설명되어 있다. 그러나 이와 같은 방법을 사용하기 위하여는 M^{-1}, M^3, M^3의 값을 미리 계산 하여야 하는 단점이 있다(단, M^2, M^{-2}는 쉬프트 연산에 의해 쉽게 구할 수 있다).

```
R4ME ( M, E, C, N)
{
 C = 1,
 for ( i = n/2-1 to 0 ) {
  C_temp = C·C (mod N)
   C = C_temp C_temp (mod N)
   if e*_2i e*_2i-1 = 0   C = C
   if e*_2i e*_2i-1 = 1   C = C M (mod N)
   if e*_2i e*_2i-1 = 2   C = C M^2 (mod N)
   if e*_2i e*_2i-1 = 3   C = C·M^3 (mod N)
   if e*_2i e*_2i-1 = -1  C = C M^-1 (mod N)
   if e*_2i e*_2i-1 = -2  C = C M^-2 (mod N)
   if e*_2i e*_2i-1 = -3  C = C M^-3 (mod N) }
return C,
}
```

Ⅲ. 제안된 RSA 암호화 시스템 구조

3 1 Radix-4 2의 보수 Baugh-Wooley 어레이 곱셈기

Radix-4 2의 보수 Baugh-Wooley 어레이 곱셈기를 설계 하였다 그림 1 은 승수 세 비트를 처리하는 Booth 인코더기(BE)와 부분곱(PP)을 만드는 블록도이다 중첩된 세 비트를 처리하여 0, ±*A*, ±2*A*를 결정하며 각 단으로 전달되며 각 단에서는 부분곱을 만들어 Radix-4 Baugh-Wooley 곱셈기에 입력한다.

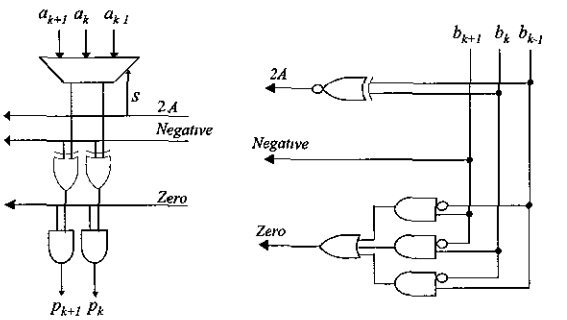


그림 1 피승수 부분곱 생성기

그림2는 8×8 Radix-4 2의 보수 시스톨릭

Baugh-Wooley 곱셈기와 PE(processing element)의 구조를 나타낸다. 로직을 간단하게 하고 연산의 효율성을 높이기 위하여 부호확장제거 기법[10]이 사용되었으며 PE를 구성하는 셀은 전가산기를 기본으로 구성되어 구조가 간단하고 확장하기 용이하다. 아래의 구조에서 곱셈 연산은 두 비트씩 처리되며 곱셈 결과의 하위 n-비트 l은 오른쪽 PE 에서 두 클럭마다 두 비트씩 출력이 되고, 상위 n-비트 m은 매 클럭마다 두 비트씩 출력된다. 아래의 시스톨릭 곱셈기의 하드웨어 이용도는 50%이며 RSA 암호화에 적용될 경우 지수처리 시에 두 개의 곱셈을 중첩처리 함으로써 이용도를 100%로 높일 수 있다

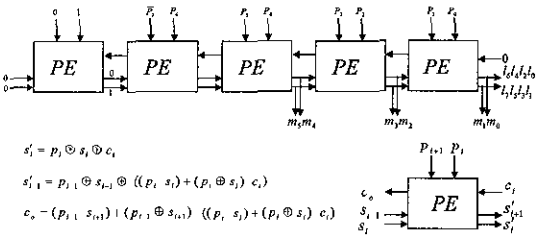


그림 2 8×8 Radix-4 2의 보수 Baugh-Wooley 시스톨릭 어레이 곱셈기

3.2 Radix-4 모듈러 감소와 T_n

그림 3은 8×8 Radix-4 모듈러 감소(modular reduction) 어레이 구조에 대한 시스톨릭 구조를 보여준다. 이 블록은 2의 보수 어레이 곱셈기에 의해서 계산되어진 하위 n-비트 X_L 의 모듈러 감소를 처리한다. FA와 HA로 이루어진 MF(Mod First), MM(Mod Middle), 와 ML(Mod Last)은 모듈러스 N 을 이용하여 서로에게 캐리와 합을 이웃셀들에게 전해준다 MF에서는 곱셈기에서 출력되는 X_L의 두 비트, 모듈러 N의 두 비트, 첫번째 MM 단에서 오는 합의 두 비트로 연산하여 캐리 두 비트를 모듈 MM으로 전해준다. MM에서는 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 만들어주며 선택 신호인 (m_1,m_0)에 의해서 $(0, N, 2N, 3N)$ 값들을 적절하게 선택할 수 있다. 곱셈기에서 만들어진 X_L의 값이 모듈러 감소 블록에 입력되어 n+1 클럭 후에 $Y_{n/2}$의 하위 두 비트인 y_0, y_1 값이 나오기 시작함과 동시에 곱셈기에서는 X의 상위 n-비트 X_M의 LSB값이 나오기 시작하므로 n+2 클럭부터 최종 결과값인 T_n의 연산을 시작 할 수 있다

그림 4는 8×8 Radix-4 Booth 부호화 곱셈기와 T_n의 구조를 보여준다. l_0에서부터 l_7 값이 모듈러 감소 블록으로 전해지고, 적절한 카운터의 신호 의해서 올바른 연산이 수행된다. 곱셈기로부터 X 의 하위 X_L 의 값이 출력되어 모듈러 감소 블록으로 전해지고 n+1 클럭이 지나면서 $Y_{n/2}$ 값이 출력되며, n+2 클럭에 X 의 상위 X_M이 나오기 시작함과 동시에 마지막 결과 값인 T_n 값이 연산된다. 만약 $X > 0$ 이면 N 를 빼주어 T_n 값을 $-N < T_n < N$ 의 범위 안에 들어오게 한다. 이때 모듈러 N의 상위 두 비트 자리에 "11"을 넣어 준 것은 곱셈기에서의 결과값 X가 '0'보다

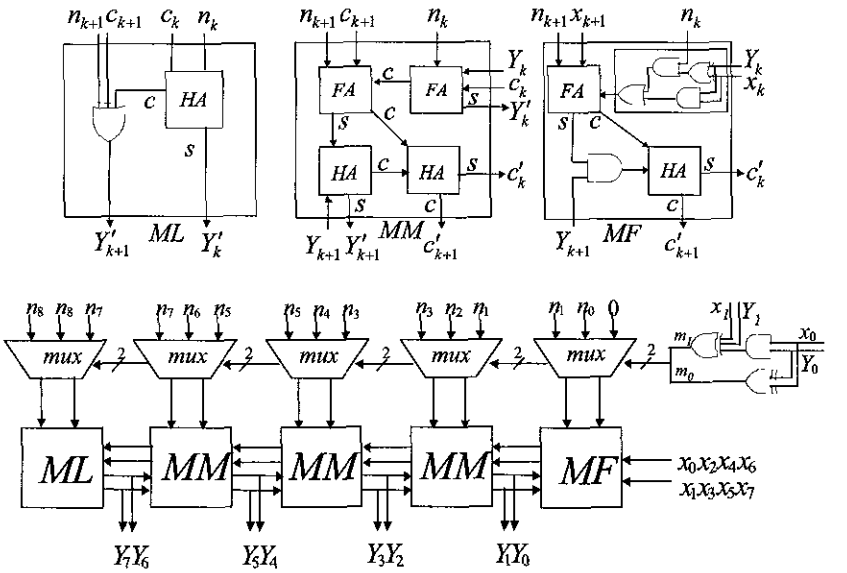


그림 3 모듈러 감소 블록의 시스톨릭 매핑

크거나 작을 경우에도 올바른 계산이 되도록 하기 위함이다. Booth 인코더를 이용하여 부분곱과 다음 Partial 블록 부분곱의 제어 신호를 만들어주는 PartialFirst블록에서 피승수의 값이 -A, -2A의 값을 가질 경우 2의 보수를 취하기 위해서 피승수 값의 보수를 취한다. 이때 올바른 2의 보수 값을 만들어주기 위해서 '1'을 더해 주어야 하는데 곱셈기에 Negative라는 신호를 보내어 '1'을 더한다.

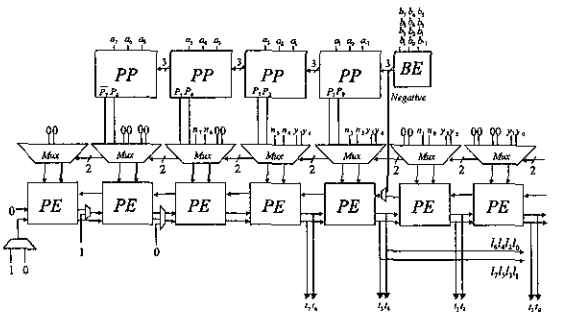


그림 4 Radix-4 Booth 부호화 곱셈기와 T_n

3 3 Radix-4 RSA 암호화 시스템 구조

그림 5는 제안된 RSA 암호화 시스템의 전체 블록도를 나타낸다. 버스 크기에 입력 데이터는 *sel* 신호에 의해서 선택되어 *load* 신호에 따라 각 레지스터에 적절하게 분배되어 저장되며, RSA 암호화 및 복호화 과정은 start 신호로 시작되어 키 값에 의해서 중간 결과값인 M_i와 P_i는 반복적으로 연산 저장된다. 승수 B는 Booth 부호화를 위해서P2S(parallel-to-serial) 블럭에 의해 3 비트씩 변환되어 모듈러 곱셈기로 전해지며 최종 결과값은 *oen* 신호와 함께 출력된다.

모듈러 곱셈기 블록에는 Booth 부호화기가 포함 되어있고 이 블록에서 부분곱을 생성하여 연산한다. 이 때 피승수 A와 모듈러스 N의 값은 n-비트씩 입력되고, 승수 B는 3 비트씩 입력된다. 곱셈기는 X의 하위 2비트씩 생성하여 모듈러 감소 블록으로 전해주어 $Y_{n/2}$ 을 생성한다. $Y_{n/2}$ 값이 나오고 n+1 클럭이 지나면서 X_M 과의 덧셈처리가 시작되고 마지막 결과값인 T_n을 연산하기 시작한다

Ⅳ. 실험 및 분석

제안된 RSA 암호화 시스템은 Verilog HDL로 설계되었으며 Modelsim 환경에서 검증되었다. 그림 6은 제안된 구조에 대한 타이밍도를 보여준다. 제안된 모듈러 곱셈기는 n-비트 모듈러 곱셈을 완전히 수행하기 위해서 (3/2)n+2 클럭을 필요로 한다. 그러나 n+5 클럭이 지난후

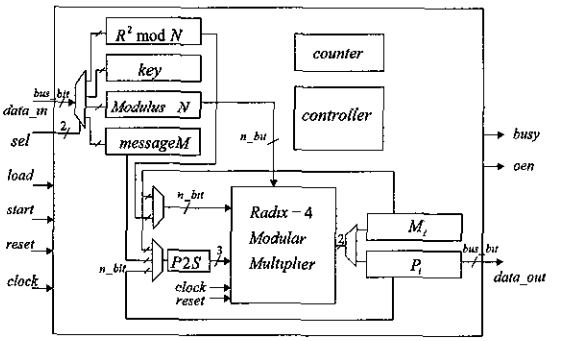


그림 5 RSA 암호화 시스템 구조

에 다음 모듈러 곱셈을 시작할 수 있기 때문에 n 비트 모듈러 곱셈을 평균적으로 n 클럭에 수행할 수가 있다. 설계된 Radix-4 시스톨릭 모듈러 곱셈기의 SFG(signal flow graph)를 분석해 보면 하드웨어 효율(hardware efficiency)은 50%이다. 이로 인한 비효율성은 RSA 암호화 과정에서 홀수와 짝수 클럭 싸이클에서 두 개의 곱셈 연산을 중첩(interleaving) 처리하면 개선될 수 있다 즉, 키 값 E_i가 '1'이면 중첩 연산에 의하여 중간 결과 값 M_i와 P_i를 동시에 처리하여 n-비트 RSA 연산을 n^2 클럭에 마칠 수 있다. 표 2 에서 본 논문의 성능과 다양한 RSA 구조를 DFFs, FAs, MUXs, Memory 와 처리시간 별로 비교하였다. 분석된 하드웨어는 모듈러 지수 연산기를 중심으로 분석되었고 그 이외에 제어기, 카운터 및 데이터 레지스터 등이 필요하다 제안된 구조가 n-비트 RSA를 적은 하드웨어 자원으로 효과적으로 수행하는 것을 보여준다 제안된 구조에서 1024-비트 RSA 암호화를 수행할 때 총 수행시간은 (1024+2)×1024 = 1 05M 클럭 싸이클이며 20MHz의 클럭을 가정할 때 52 5ms 정도 소요된다.

표 2 RSA 구조의 성능 비교

Architectures	Hardware Requirement				Time
	DFF	FA	MUX	RAM	clock
Yang[6]	12n	2n	8n	0	1 5n^2
Sheu[5]	10n	3 2n	9n	0	2 5n^2
Chiang[7]	6 7n	1 5n	n	10*512	2 7n^2
Su[8]	12n	2n	6n	0	2n^2
Ours	6 5n	2 5n	3 5n	0	n^2/2

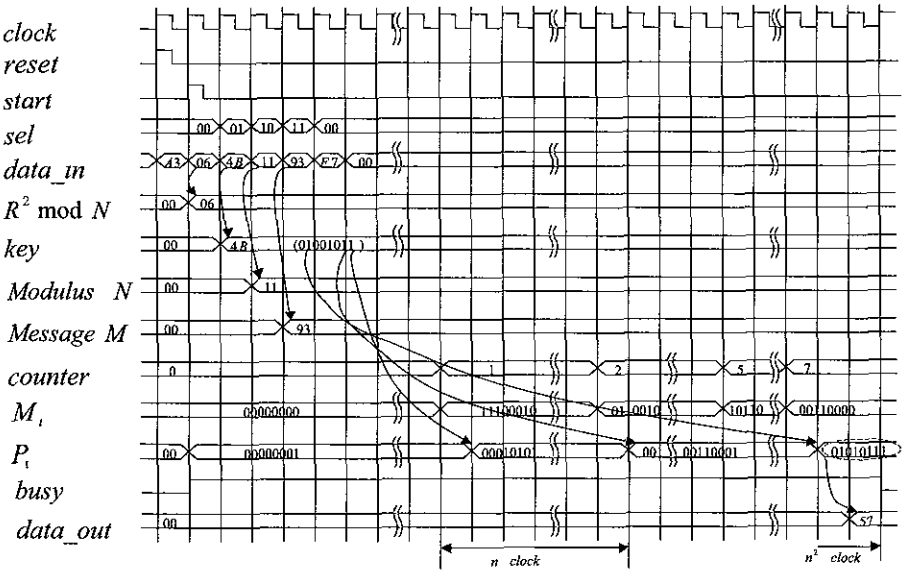


그림 6 제안된 RSA 암호화 구조의 타이밍 다이아그램

V. 결 론

본 논문에서는 보안성이 뛰어나고 그의 응용분야를 확대해 나가고 있는 RSA 암호화 시스템을 위한 효과적인 Radix-4 시스톨릭 VLSI 구조를 제안하였다. 그의 핵심 연산인 n-비트 모듈러 곱셈을 위하여 미리 계산되는 곱셈 $X(=A\times B)$의 상위 n-비트와 하위 n-비트를 나누어 처리하는 효율적인 Radix-4 모듈러 곱셈 알고리즘 및 그에 대한 시스톨릭 구조가 제안되었다. 하나의 지수 비트 처리를 위한 두 개의 곱셈을 중첩하여 연산하는 시스톨릭 인터리빙의 적용과 하위 n-비트만의 모듈러 감소 처리로 인한 전체 이터레이션 수의 감소로 인하여 제안된 구조는 n-비트 모듈러 지수 연산를 n^2 클럭 사이클에 수행할 수 있으며 100%의 하드웨어 효율을 보여준다. 또한 모듈러 곱셈의 횟수를 줄이기 위하여 지수를 Radix-4 SD 수체계를 이용하여 인코딩 하였으며 결과적으로 20% 정도의 NZ 디지트가 감소하였다. 다른 구조들과 비교한 결과 제안된 구조는 비교적 적은 하드웨어를 이용하여 우수한 성능을 보였으며 매우 규칙적이고 확장 가능하여 VLSI 구현에 적합하였다.

감사의 글

저자들은 본 연구를 위하여 FPGA 설계 환경을 제공하여 준 IDEC에 감사드린다.

참 고 문 헌

[1] R L Rivest, A. Shamir, and L Adelman "A method for obtaining digital signatures and public-key cryptosystem," Comm ACM, vol.21, no 2, pp. 120-126,1978

[2] P. L. Montgomery, "Modular multiplication without trivial division," Math. Comput Vol 44, pp.519-521, 1985

[3] P. S. Chen, S A Hwang, and C W Wu, "A systolic RSA public keycryptosystem," in Proc IEEE Int. Symp Circuits Syst, pp 408 411, May 1996

[4] C Y Su, S. A Hwang, P S.Chen and C W Wu "An Improved Montgomery'sAlgorithm for High-Speed RSA Public-Key Cryptosystem," IEEE Trans VLSI Syst,No.2, June 1999

[5] J. L. Sheu, M D Shieh, C H Wu, and M. H Sheu, " A pipelinedof fast modular multiplication for RSA cryptography," IEEE Int. Symp Circuits and Systems, vol.2, pp121-124, 1998

[6] C C Yang, T. S Chang, and C. W Jen, "A new RSA cryptosystem hardwarebased on Montgomery's algorithm," IEEE Trans. CAS II, vol 45, no.7, pp.908-913, July 1998.

[7] J S Chiang and J K. Chen, "An efficient

VLSI architecture for RSA public-key cryptosystem," Proc. IEEE Intl. Symp. on Circuits and Systems, pp.496-499, 1999.

[8] C. Y. Su and C. W. Wu, "A practical VLSI architecture for RSA public-key cryptography," Proc. 6th VLSI Design/CAD Symp., pp.273-276, 1995.

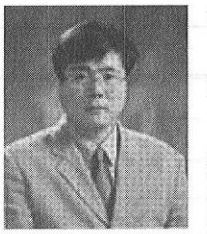
[9] C. Zhang, H Martin, and D. Yun, "Parallel algorithms and systolic array design for RSA cryptosystem," Proc. IEEE Intl. Conf. on Systolic Arrays, pp.341-350, 1988

[10] K. Hwang, "Computer arithmetic: principles, architecture, and design," Wiley, 1979

[11] A. Menezes, P. Oorschot, and S. Vanstone, "Handbook of applied cryptography," CRC Press, 1997

[12] K. Parhi, "VLSI digital signal processing systems: design and implementation," Wiley, 1999

박 태 근 (Taegeun Park)



1985년 연세대학교 전자공학 학사.

1988년 Syracuse Univ. Computer 공학석사.

1993년 Syracuse Univ. Computer 공학박사.

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

1994년 - 1998년 현대전자 System IC 연구소 책임연구원.

1998년 - 현재 가톨릭대학교 정보통신전자공학부 부교수.

<관심 분야> VLSI 설계, CAD, 병렬처리 등임.