고속 모듈러 승산의 비교와 확장 가능한 시스톨릭 어레이의 설계

추 봉 조† · 최 성 욱††

요 약

본 논문은 고속 모듈러승산을 위한 몽고메리 알고리즘에 근거한 Walter 방식과 Iwamura 방식에 대해 병렬알고리즘을 유도하고, 각 병렬알고리즘의 데이터종속그래프를 비교한다. 비교결과에 따라 데이터의존 그래프 내에서 계산점이 적은 Walter 방식의 병렬알고리즘을 채택하여 이를 확장 가능한 비트레벨 시스톨릭어레이로 설계하기 위해 각 투영방향에 따른 파이프라인을 위한 공간 및 시간 다이어그램을 계산한다. 제안된 확장형 시스톨릭어레이의 정확한 내부동작을 위해 C++ 언어로 작성하여 검증하였다.

Comparison of High Speed Modular Multiplication and Design of Expansible Systolic Array

Bong-Jo Choo† · Sung-Wook Choi††

ABSTRACT

This paper derived Montgomery's parallel algorithms for modular multiplication based on Walter's and Iwamura's method, and compared data dependence graph of each parallel algorithm. Comparing the result, Walter's parallel algorithm has small computational index in data dependence graph, so it is selected and used to computed spatial and temporal pipelining diagrams with each projection direction for designing expansible bit-level systolic array. We also evaluated internal operation of proposed expansible systolic array with C++ language.

1. 서 론

정보의 다양성, 복잡성, 대용량성에 따라 이를 처리하기 위해서는 높은 계산능력이 요구되고 특히 각종 정보보호기술을 다루는 암호화 시스템의 경우에는 이러한 계산능력과 계산시간이 핵심적인 요소가 된다. 현대 공개키 암호시스템은 512비트 이상의 큰 수에 대해 모듈러 멱승을 요구함으로써 기본연산인 모듈러 승산의 고속화를 위한 VLSI 하드웨어 구현이 필요하게 되었다.[1][2]

† 정 회 원 : 김천대학 사무자동화과 교수
†† 정 회 원 : 한국해사정보통신주식회사 대표이사

논문접수 : 1998년 6월 10일, 심사완료 : 1999년 2월 26일

특히, 모듈러 승산은 RSA와 같은 공개키 암호시스템의 암호화 및 복호화 과정에서 $M^e \bmod N$의 모듈러 멱승을 구하기 위해 전체 계산의 대부분을 차지하며, 그 중에서 몽고메리 모듈러 승산 알고리즘이 가장 효과적인 것으로 알려져 있다.[3,4,5] 모듈러 승산은 문제 규모가 크고 많은 수의 메세지블럭이 연속적으로 계산되어야 하므로 입출력 채널의 수를 상수개로 고정시켜서 VLSI 칩으로 제작할 경우, 문제 크기에 관계없이 입출력 핀의 수를 고정시킬 수 있는 확장 가능한 모듈 구성이 매우 중요하다.

본 논문에서는 시스톨릭 어레이 설계기법을[7,9] 적용하여 Iwamura와 Walter가 제안한 몽고메리알고리즘을 근거로 병렬알고리즘으로 유도한 후 데이터 의존

그래프로 표현하고 이를 시간 및 공간변환을 수행하여 일차원 시스톨릭 어레이를 설계하였다. 그리고 설계한 일차원 시스톨릭 어레이를 확장 가능한 모듈의 구성을 고려하여 여러 가지 평가기준에 따라 비교, 분석하며 컴퓨터시뮬레이션을 수행하여 설계의 정확성과 유효함을 검증한다.

본 논문의 구성은 다음과 같다. 2절에서 RSA 암호시스템에 모듈러 승산과 몽고메리 병렬알고리즘, Walter과 Iwamura 알고리즘의 데이터의존그래프를 유도 비교한다. 3절에서는 모듈러 승산을 고속으로 하기 위한 비트레벨 시스톨릭 어레이를 유도 설계하고 4장에서 결론을 내린다.

2. 몽고메리 병렬알고리즘

2.1 RSA 암호시스템의 모듈러 승산

암호시스템에서 많이 사용되는 방법이 1978년 미국 MIT의 Rivest, Shamir, Adleman에 의하여 개발된 RSA 공개키 암호시스템개념이다. RSA암호시스템은 식 (1)과 같이 512-bit 이상 큰 정수의 소인수분해가 어렵고 많은 계산을 요구한다.[4][8]

$$T = \sum_{i=0}^{n-1} t_i r^i = t_0 + \cdots + t_{n-1} \times r^{n-1} \qquad (1)$$

where, $t_i \in \{0,1\}^v$ (i = 0, ···, n-1), $r = 2^v$.

RSA 암호시스템의 유사암호계열은 Rabin 암호, Williams 암호 등이 있고, 암호시스템의 안전성은 Euler totient 함수값 $\phi(N)$의 계산에 의존한다. 정수 N을 소인수 분해하여 $\phi(N)$를 구한 후 유클리드알고리즘을 사용하여 복호키 d를 쉽게 계산할 수 있다. 따라서 N의 크기는 최소한 512-bit(154-digit) 이상을 요구한다. RSA 암호시스템에서 암호화과정(C)과 복호화 과정(M)은 식 (2)와 식 (3)과 같다.

$$C = E(M) = M^e \bmod N \qquad (2)$$

$$M = D(C) = D(E(M)) = C^d \bmod N \qquad (3)$$

2.2 몽고메리 모듈러 승산 알고리즘

RSA 암호화시스템 등에 사용되는 $M^e \bmod N$ 모듈러 승산을 위한 몽고메리 모듈러 승산 알고리즘은 (그림 1)과 같다.

```
/* A,B,N,R,P : double integer */

/* Pi : i th accumulative value */
Algorithm : MM(A,B,N,R)
Input     : A, B, N, R.
Output    : P = A×B×R^-1 mod N
 1 :              no' ← -no^-1 mod r
 2 : P-1 ← 0
 3 : for i ← 0 to n-1
 4 :                  Pi ← Pi-1 + ai×B×r^i
 5 :                  mi ← pi×no' mod r
 6 :                  Pi ← Pi + mi×N×r^i
 7 : P ← Pn div R
 8 : return P
```

(그림 1) 몽고메리 모듈러 승산 알고리즘

일반적으로 다정도 정수 A, B에 대하여 몽고메리 모듈러 승산은 다음과 같이 수행된다.[3]

단계 1. 정규표현식 A, B를 몽고메리 표현식인 A', B'로 변환한다.

$A' = MM(A,R^2 \bmod N,N,R) = AR \bmod N$

$B' = MM(A,R^2 \bmod N,N,R) = BR \bmod N$

단계 2. 몽고메리 모듈러 승산을 계산한다.

$P' = MM(A',B',N,R) = A'B'R^{-1} \bmod N$

$= ABR \bmod N = (AB)'$.

단계 3. 결과값 P'을 정규표현식으로 변환한다.

$P = MM(P',1,N,R) = P'R^{-1} \bmod N$.

(그림 1)의 알고리즘에서 모듈러스의 배수 m_i는 p_i에 의하여 계산되고, 그 결과를 P_i의 계산에 이용하며 m_i를 뺄셈 연산하기보다는 덧셈 연산을 수행하고 피승수 A의 처리 순서를 역전시켜 A의 낮은 자리수부터 계산한다.

2.3 Walter 방식의 병렬알고리즘

(그림 1)의 몽고메리 모듈러 승산알고리즘에서 Walter는[6] 계산의 대부분을 차지하는 3~6행에 해당하는 순환부분을 병렬화시키기 위하여 아래와 같이 변형하였다.

```
for i ← 0 to n-1
    mi ← ((Pi-1 mod r) + ai × b0) × no' mod r
    Pi ← (Pi-1 + ai × B + mi × N) div r
```

위의 수식을 전개하면 $r^i P_i = a_i B + m_i N$이 되며, P

의 최종값 P_n은 $r^nP_n = AB + MN$이 되어 결과적으로 $P \equiv ABR^{-1} \bmod N$이 된다.

(그림 1)의 알고리즘에서는 P의 최종값을 R(= r^n)을 나누어서 P를 구했으나 비트 레벨(bit level)에서 처리가 가능하도록 최종 결과값을 구하기 위하여 나눗셈연산을 하는 대신 각각의 부분 결과값에서 나눗셈 연산을 수행하였고, 병렬처리가 용이하도록 각 계산점에서 의존관계의 수를 줄였다. 결과값 P를 생성하는 한 부분인 비트레벨의 피승수 a_i와 연산되는 승수 B를 단지 다정도상수로 연산하는 과정과 비트레벨로 인덱스를 확장한 b_j에서의 승산과정이 동일함은 다음의 예로 설명된다.

```
   5 3              1 1 0 1 0 1   = A
 × 4 2            × 1 0 1 0 1 0   = B
   12 6             1 0 1 0 1 0   = a0B×r^0
20 10              0 0 0 0 0 0   = a1B×r^1
                 1 0 1 0 1 0     = a2B×r^2
               0 0 0 0 0 0       = a3B×r^3
             1 0 1 0 1 0         = a4B×r^4
           1 0 1 0 1 0           = a5B×r^5
       1 0 0 0 1 0 1 1 0 0 1 0   = A×B
```

위의 예에서 각 부분승산값은 한 단계씩 아래로 이동되어 더해진다. 이것은 이차원 i, j 인덱스공간에서 a는 (0, 1)방향으로, b는 (1, 0)방향으로, P는 (1, -1)방향으로 값이 이동함을 의미하며, 같은 방법으로 m은 a와 같이 (0, 1)방향으로, n은 b와 같이 (1, 0)방향으로 이동되면서 계산된다. 따라서 각 몽고메리 부분결과값을 생성하는데 있어서 앞 단계에서 계산된 P의 부분결과값은 $P_{i-1,j+1}$이 된다. 특히 모듈러스의 배수 m_i는 앞 단계의 부분 결과값 $P_{i-1,j+1}$과 $a_i \times b_{i-1,0}$를 합한 결과의 최하위 비트에 의존하기 때문에 j = 0인 인덱스에서만 계산된다.

몽고메리 알고리즘에서 부분결과값 $P_{i,j}$를 계산하는 과정에서 두비트의 캐리가 발생하게 되는데 이중 상위 캐리비트 $cal_{i,j}$는 다음 단계의 캐리 계산에 이용되고, 하위 캐리비트 $ca0_{i,j}$는 다음 단계의 부분결과값 $P_{i,j}$의 계산에 이용된다. 이를 병렬알고리즘으로 구성하면 (그림 2)와 같다.

```
1 :  begin
2 :  for all 0 ≤ i ≤ n-1, 0 ≤ j ≤ n+1 do in parallel
3 :   if i = 0 then
4 :     b_{i,j} ← b_j; n_{i,j} ← n_j; p_{i,j} ← 0;
5 :   else
6 :     b_{i,j} ← b_{i-1,j}; n_{i,j} ← n_{i-1,j};
7 :   if j = n+1 then
8 :     p_{i,j} ← 0;
9 :   else
10:     p_{i,j} ← p_{i-1,j+1};
11:   end if
12:   end if
13:   if j = 0 then
14:     a_{i,j} ← a_i; 0_{i,j} ← 0; cal_{i,j} ← 0;
15:     m_{i,j} ← ((p_{i,j} mod r) + a_i×b_{i-1,j}) mod r;
16:   else
17:     a_{i,j}←a_{i,j-1}; m_{i,j}←m_{i,j-1};
        ca0_{i,j}←ca0_{i,j-1}; cal_{i,j}←cal_{i,j-1};
18:   end if
19:   p_{i,j} ← (p_{i,j} + a_{i,j}×b_{i,j} + m_{i,j}×n_{i,j} + ca0_{i,j}) mod r;
20:   ca0_{i,j} ← ((p_{i,j} + a_{i,j}×b_{i,j} + m_{i,j}×n_{i,j} + ca0_{i,j} + cal_{i,j}
        ×r) div r) mod r;
21:   cal_{i,j}←(p_{i,j} + a_{i,j}×b_{i,j} + m_{i,j}×n_{i,j} + ca0_{i,j} +cal_{i,j}×r)
        div r^2;
22:  all for
23:  end
```

(그림 2) Walter 방식의 병렬알고리즘

2.4 Iwamura 방식의 병렬알고리즘

(그림 1)의 몽고메리 모듈러 승산알고리즘에서 Iwamura 등은 알고리즘의 병렬화를 위해 3~6행에 해당하는 순환부분을 병렬화시키기 위하여 아래와 같이 변형하였다.

```
for  i ← 0  to  n
       m_i ← (P_{i-1}  mod r)×n_o'  mod r
       P_i ← (P_{i-1} + a_i×B×r + m_i×N) div r
```

Walter 방식의 병렬알고리즘과 차이점은 모듈러스의 배수 m_i를 계산하기 위하여 요구되었던 a_i, b_0의 승산부분이 제거되었고, 승수 B값을 인덱스 j인 2차원 계산공간으로 확장한 상태에서 최하위 비트(LSB) 한 자리수를 left-shift하여 입력하거나 $r=2^v$ 일 때 $a_{i,j-1} \times b_{i+1,j}$의 결과값을 최상위비트(MSB) 방향으로 v비트만큼 쉬프트하여 전체적으로 루프를 한번 더 수행한다. 이를 비트레벨에서 병렬알고리즘을 구성하면 (그림 3)과 같다.

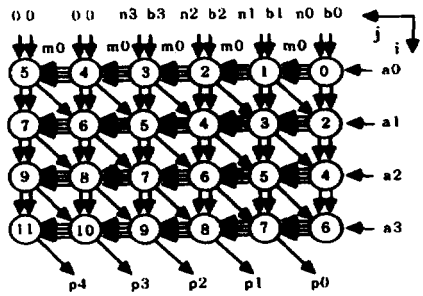
```
1 : begin
2 :    for all 0 ≤ i ≤ n-1,  0 ≤ j ≤ n+2
       do    in parallel
3 :       if i = 0 then
4 :          b_{i,j} ← b_j; n_{i,j} ← n_j; p_{i,j} ← 0;
5 :       else
6 :          b_{i,j} ← b_{i-1,j}; n_{i,j} ← n_{i-1,j};
7 :          if j = n+2  then
```

```
8 :              p_{i,j} ← 0;
9 :           else
10:              p_{i,j} ← p_{i,j-1};
11:           end if
12:        end if
13:        if j = 0 then
14:           a_{i,j} ← a_i; ca0_{i,j} ← 0; cal_{i,j} ← 0;
15:           m_{i,j} ← p_{i,j} mod r;
16:        else
17:           a_{i,j}←a_{i,j-1}; m_{i,j}←m_{i,j-1};
              ca0_{i,j}←ca0_{i,j-1}; cal_{i,j}←cal_{i,j-1};
18:        end if
19:        p_{i,j} ← (p_{i,j} + a_{i,j}×b_{i,j}×r + m_{i,j}×n_{i,j} +
              ca0_{i,j}) mod r;
20:        ca0_{i,j}← ((p_{i,j} + a_{i,j}×b_{i,j}×r + m_{i,j}×n_{i,j}
           + ca0_{i,j} + cal_{i,j} × r) div r) mod r;
21:        cal_{i,j} ← (p_{i,j} + a_{i,j}×b_{i,j}×r + m_{i,j}×n_{i,j}
           + ca0_{i,j} + cal_{i,j}×r) div r^2;
22:     all for
23: end
```

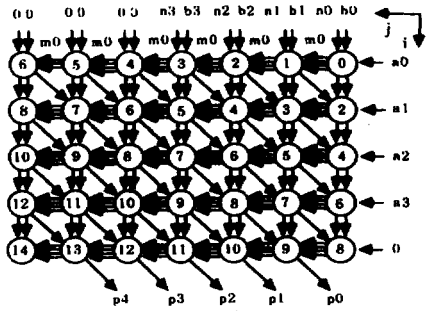
(그림 3) Iwamura 방식 병렬알고리즘

2.5 데이터 의존 그래프

(그림 4)의 (a)는 문제 크기가 4인 경우 Walter방식에 의한 데이터 의존 그래프를 나타내며, j=0인 열의 계산점이 나머지 계산열과 다르고 인덱스 j의 방향에서 캐리 두 비트를 계산에 포함시키기 위하여 문제의 크기보다 두 개의 계산열이 더 요구된다. (그림 4)의 (b)는 Iwamura 방법에 의한 데이터 의존 그래프로서 (a)에 비해서 한 행과 한 열이 더 요구된다. 그러므로 두 방식 중 j=0인 계산열의 계산과정은 (b)가 단순하나 나머지 계산열은 동일하므로 전체적으로 볼 때 (a)의 Walter 방식이 더 단순하고 계산행과 계산열의 수도 각각 한 개씩 적게 요구되어 시스톨릭 어레이에서 처리요소 수가 적어진다. 본 논문에서는 Walter 방식을 기본알고리즘으로 채택하여 시스톨릭 어레이를 설계한다.



(a) Walter 방식 의존그래프



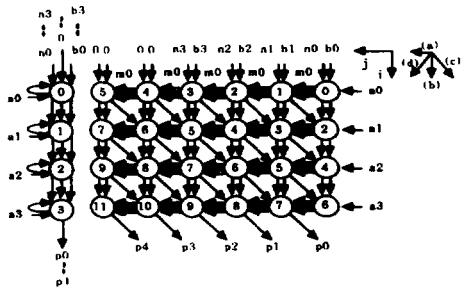
(b) Iwamura 방식 의존그래프

(그림 4) 데이터 의존 그래프 (n=4)

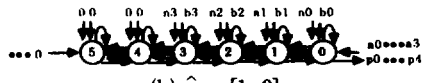
3. 몽고메리 모듈러 승산을 위한 시스톨릭 어레이의 설계

3.1 일차원 시스톨릭 어레이의 설계

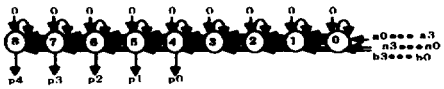
(그림 4)의 데이터 의존 그래프에서 각 노드의 공간변환을 수행하고 각각의 시간변환에서 유도된 아크들의 방향으로 일차원시스톨릭 어레이를 설계하면 (그림 5)와 같다.[8] 여기서 확장 가능한 모듈의 구성을 고려하여 볼 때 (그림 5)의 (d)가 가장 유리하나 처리요소 및 데이터 통신 패스(path)의 수가 (a)의 경우보다 많다. 그러므로 처리요소의 수와 데이터 통신 패스의 연결형태 등에서 유리한 (그림 5)의 (a)에서 입력 데이터 a의 흐름을 b, n과 같은 방향으로 직렬(serial) 형태로 변형하고 입력 데이터 a는 한 단위시간의 전달지연으로 다음 처리요소로 이동되며, 동작을 제어하기 위하여 두 단위시간의 전달지연을 갖는 제어신호(ctl)를 입력시켜 사용한다. 또한 데이터 a의 최초의 입력되는 두 비트는 두 단위시간의 지연이 요구되므로 블럭 파이프라인의 주기는 '3'이 된다. 또한 데이터 통신 패스는 단일 방향으로만 존재하므로 확장 가능한 모듈을 구성하고 어레이의 VLSI 구현에서 결함허용성이 용이하다.



(a) $\hat{p}$ = [0, 1]



(b) $\hat{p}$ = [1, 0]



(c) $\hat{p}$ = [1, -1]

(d) $\hat{p}$ = [1, 1]

(그림 5) 일차원 시스톨릭 어레이 설계(n=4)

(그림 5)의 4가지 투영방향 [0, 1], [1, 0], [1, -1], [1, 1]에 대해 각 투영 방향별 파이프라인 구성을 위한 시간 및 공간 다이어그램은 <표 1>과 같다.

〈표 1〉 파이프라인을 위한 공간 및 시간 다이어그램

공간 \ 시간	0	1	2	3	4	5	6	7	8	9	10	11
0	T_1	T_2	T_3	T_4	T_5	T_6						
1			T_1	T_2	T_3	T_4	T_5	T_6				
2					T_1	T_2	T_3	T_4	T_5	T_6		
3							T_1	T_2	T_3	T_4	T_5	T_6
4									T_1	T_2	T_3	T_4
5											T_1	T_2

(a) $\widehat{p_1}$ = [0 1]

공간 \ 시간	0	1	2	3	4	5	6	7	8	9	10	11
0	T_1		T_1		T_1		T_1					
1		T_2		T_2		T_2		T_2				
2			T_3		T_3		T_3		T_3			
3				T_4		T_4		T_4		T_4		
4					T_5		T_5		T_5		T_5	
5						T_6		T_6		T_6		T_6

(b) $\widehat{p_2}$ = [1 0]

공간 \ 시간	0	1	2	3	4	5	6	7	8	9	10	11
0	T_1											
1		T_2	T_1									
2			T_3	T_2	T_1							
3				T_4	T_3	T_2	T_1					
4					T_5	T_4	T_3	T_2				
5						T_6	T_5	T_4	T_3			
6								T_6	T_5	T_4		
7										T_6	T_5	
8												T_6

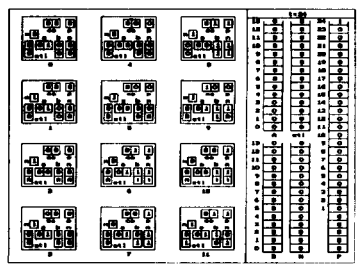
(c) $\widehat{p_3}$ = [1 −1]

공간 \ 시간	0	1	2	3	4	5	6	7	8	9	10	11
-5						T_6						
-4					T_5			T_6				
-3				T_4			T_5			T_6		
-2			T_3			T_4			T_5			T_6
-1		T_2			T_3			T_4			T_5	
0	T_1			T_2			T_3			T_4		
1			T_1			T_2			T_3			
2					T_1			T_2				
3							T_1					

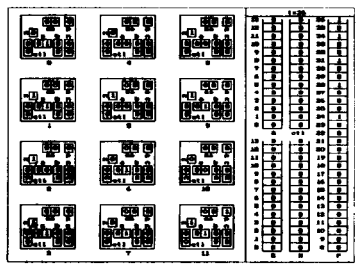
(d) $\widehat{p_4}$ = [1 1]

3.2 결과 및 고찰

설계한 모듈러 승산을 위한 일차원 시스톨릭 어레이의 정확한 내부동작과 평가를 검증하기 위하여 문제크기를 입력받고 입력받은 두 수를 비트레벨로 표시하여 이를 연속적으로 입력받아 각 시간별로 처리상태를 표시하기 위해 C++ 언어를 이용하여 동작과정을 시뮬레이션 하였다. 문제크기가 12인 경우의 시뮬레이션 과정에서의 화면 구성은 (그림 6)과 같다.



(a) t=24일 때 처리요소 상태



(b) t=36일 때 처리요소 상태

(그림 6) 문제크기 12일 때의 시뮬레이션 상태

(그림 6)은 문제의 크기가 n = 12이고, 단일 입력 데이터 A = 2763, B = 3582, N = 4077이며, 예상되는 계산결과의 출력 데이터는 P = 5265 또는 1188인 경우, 컴퓨터 시뮬레이션 하는 과정에서 t = 24, 36일 때

어레이내의 각 처리요소들과 입출력 큐의 내용을 나타내면 각각 (그림 6)의 (a), (b)와 같다.

(그림 6)의 (a)는 t = 24 단위시간에 각 처리요소의 상태를 나타내고 결과값의 첫비트가 이때부터 마지막 처리요소에서 출력되어 출력 큐에 저장되기 시작한다. (그림 6)의 (b)는 t = 36 단위시간에 결과값의 마지막 비트가 마지막 처리요소에서 출력되어 그 최종결과값은 출력큐 P에서 출력된 결과를 보인다. 연속적인 입력값이나 문제의 크기가 다른 여러 가지 경우의 입력값에 대해서도 컴퓨터시뮬레이션을 수행하였고, 정확한 계산결과가 출력됨을 확인하였다.

4. 결 론

모듈러 승산은 문제규모가 크고 연속적인 메시지블럭을 계산함으로 입출력 채널의 수를 상수 개로 고정시켜서 VLSI 칩 크기를 최소화 할 수 있도록 문제 크기에 관계없이 입출력 핀의 수를 고정시킬 수 있는 확장 가능한 모듈 구성의 개념이 매우 중요하다.

본 논문에서는 [8]에서 설계된 시스톨릭 모듈러 승산을 위한 몽고메리 병렬알고리즘을 Walter 방식과 Iwamura 방식을 각각 병렬알고리즘으로 재구성한 후 데이터 의존 그래프를 구하여 비교한 후, 비교 결과에 따라 계산 행과 계산열의 수가 한 개씩 적고 시스톨릭 어레이의 설계에서 처리요소의 수가 적게 요구되는 Walter 방식을 선택하여 시간변환 및 공간변환을 수행하는 체계적인 방법으로 일차원 시스톨릭 어레이들을 설계하였다.

설계된 시스톨릭어레이로 검증하기 위하여 각 투영방향에 따른 파이프라인을 위한 공간 및 시간 다이어그램을 계산하였고, 정확한 내부동작과 평가를 위해 C++ 언어로 작성하여 검증하였다.

참 고 문 헌

[1] H. S. Hwang, C. H. Lim, and P. J. Lee, "An Implementation and Analysis of Modular Multiplication on PC," KIISC Review, Vol.4, No.3, pp. 34-60, 1994.

[2] Giuseppw Alia and Enrico Martinelli, "A VLSI Modulo m Multiplier," IEEE Trans. on Computers, Vol.40, No.7, July, 1991.

[3] S. R. Dussé and B. S. Kaliski Jr. "A Cryptography Library for the Motorola DSP 56000," Proc. EUROCRYPT '90, pp.230-244, 1990.

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

[5] K. Iwamura, T. Matsumoto, and H. Imai, "Systolic Arrays for Modular Exponentiation Using Montgomery Method," Proc. EUROCRYPT '92, pp.477-481, 1992.

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

[7] S. Y. Kung, "VLSI Array Processor," Prentice Hall, 1986.

[8] Po-Song Chen, Shin Arn Hwang, "A Systolic RSA Public Key Cryptosystem," IEEE International Sysmposium on Circuit and System, pp. 408-411, 1996.

[9] 최성욱, 추봉조, 우종호, "모듈러 곱셈을 위한 모듈러 확장가능한 시스톨릭 어레이의 설계", 대한전자공학회, 1995년도 하계논문집, pp.353-356, 1995.

추 봉 조

e-mail : bjchoo@kimcheon.ac.kr

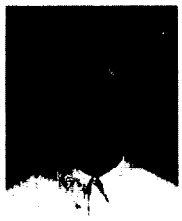
1990년 경성대학교 전자계산학과 졸업(이학사)

1992년 경성대학교 대학원 전자계산학과 졸업(이학석사)

1996년 부경대학교 대학원 전자공학과 박사 수료

1996년~현재 김천대학 사무자동화과 부교수

관심분야 : 분산 및 병렬처리, 영상처리, 고속통신망



최 성 욱

e-mail : victor@maritel.co.kr

1992년 동아대학교 산업공학과 졸업(공학사)

1995년 부경대학교 산업대학원 전자 및 컴퓨터공학과 졸업(공학석사)

1995년~1998년 한국해양대학교 대학원 전자통신공학과 박사

1990.년~1993년 해군 제 3함대사령부 전산실 근무

1997년~1998년 부경대학교, 동명정보대, 한국해양대학교 강사

1998년~현재 한국해사정보통신주식회사 대표이사

관심분야 : 분산 및 병렬처리, 지능망, 영상처리