情報保護學會論文誌
第12券 第2號, 2002. 4

Radix-2^k 모듈라 곱셈 알고리즘 기반의 RSA 지수승 연산기 설계*

권 택 원**, 최 준 림***

Implementation of RSA Exponentiator Based on Radix-2^k Modular Multiplication Algorithm

Taek-Won Kwon**, Jun-Rim Choi***

요 약

본 논문에서는 radix-2^k 모듈라 곱셈 알고리즘 기반의 고속 RSA 지수승 연산기의 구현 방법을 제시하고 검증하였다. Radix-2^k 모듈라 곱셈 알고리즘을 구현하기 위해 Booth recoding 연산 알고리즘[1]을 사용하였으며 최대 radix-16 연산을 위해 2K-byte 메모리와 2개의 전가산기와 3개의 반가산기의 지연을 갖는 CSA(carry-save adder) 어레이를 사용하였다. CSA 어레이 출력인 캐리와 합을 고속으로 가산하기 위해 마지막 덧셈기로써 캐리 발생과 지연시간이 짧은 가상 캐리 예측 덧셈기(pseudo carry look-ahead adder)[2,3]를 적용하였다. 또한, 주어진 공정에서 동작 주파수와 처리량의 관계를 통해 radix-2^k에서 설계 가능한 radix 값을 제시하였다. Altera FPGA EP2K1500E를 사용하여 기능을 검증한 후 삼성 0.35㎛ 공정을 사용하여 타이밍 시뮬레이션을 하였으며 radix-16 모듈라 곱셈 알고리즘을 사용할 경우 모듈라 곱셈에 (n+4+1)/4 의 클럭을 사용하여 1,024-bit RSA를 처리하는데 50㎒에서 5.38㎳의 연산 속도를 측정하였다.

ABSTRACT

In this paper, an implementation method of RSA exponentiator based on radix-2^k modular multiplication algorithm is presented and verified. We use Booth recoding algorithm[1] to implement radix-2^k modular multiplication and implement radix-16 modular multiplier using 2K-byte memory and CSA(carry-save adder) array - with two full adder and three half adder delays. For high speed final addtion we use a reduced carry generation and propagation scheme called pseudo carry look-ahead adder.[2,3] Furthermore, the optimum value of the radix is presented through the trade-off between the operating frequency and the throughput for given Silicon technology. We have verified 1,024-bit RSA processor using Altera FPGA EP2K1500E device and Samsung 0.35㎛ technology. In case of the radix-16 modular multiplication algorithm, (n+4+1)/4 clock cycles are needed and the 1,024-bit modular exponentiation is performed in 5.38㎳ at 50㎒.

keyword : *RSA, CSA, radix-2^k modular multiplier, pseudo carry look-ahead adder*

Ⅰ. 서 론

개방 네트워크의 신뢰성과 안전성을 제공하는 많은 기술들이 있지만 그 중에서도 공개키 암호 기술은 소인수분해의 어려움에 안전도의 근간을 갖고 있으며 암호화와 전자 서명을 위한 RSA[4] 암호 시스템이

* 본 논문은 ITRC 사업과 IDEC 사업의 지원으로 수행되었습니다.
** 경북대학교 전자공학과 대학원(ktw@palgong.knu.ac.kr)
*** 경북대학교 전자전기컴퓨터학부 조교수(jrchoi@ee.knu.ac.kr)

가장 널리 사용되고 있다. RSA 암호 시스템은 안전도를 높이기 위해 키 값을 1,024 비트 이상으로 높이고 있는 추세이며 이러한 키 값으로 인해 반복적인 모듈라 곱셈 연산 구조를 갖는 하드웨어 구현을 매우 어렵게 만들고 있다. 지수승 연산의 결과는 모듈라 곱셈의 반복 연산을 통해 얻어지며[4,5] 모듈라 곱셈기를 어떻게 설계하느냐에 따라 처리 속도는 달라진다. 일반적인 모듈라 곱셈기로써 systolic 어레이 구조와 CSA 덧셈기 구조를 사용한 모듈라 곱셈기가 제안되었다.[6~9] 또한, 하드웨어 소모는 많지만 고속 모듈라 연산이 가능한 RNS(residue number system) 기반과[10,11] radix 기반의[12~14] 몽고메리 모듈라 곱셈 알고리즘이 제안되었으며, 처리 속도는 느리지만 하드웨어 소모가 적은 분할형 워드 기반 모듈라 곱셈 알고리즘도[15,16]제안되고 있다.

기존의 radix-2^k의 모듈라 곱셈기는[12,13] systolic array 기반의 radix-2 모듈라 곱셈기[7]를 확장한 것으로써 radix-2^k를 구현하기 위하여 중간 값 제어와 저장을 위한 인터리빙 제어 신호와 파이플라인 레지스터를 사용하여 하드웨어 소모가 기존의 radix-2보다 50% 증가하는 단점이 있었다. 본 논문은 이러한 문제점을 해결하기 위하여 하드웨어 소모가 적고 연산 속도를 높일 수 있는 CSA 기반의 곱셈기[9]와 radix-2^k의 부분 곱과 몽고메리 reduction 연산을 위해 look-up 테이블 방식을 사용하여 설계하였다. 또한, 주어진 공정에서의 동작 주파수와 처리 속도 관점에서 본 논문에서 제시한 하드웨어 구조에 맞는 최적의 radix 값을 제시한다. 먼저 본문 Ⅱ장에서는 하드웨어 구현 관점에서 radix-2^k 몽고메리 모듈라 곱셈 알고리즘을 기술하였다. 본문 Ⅲ장에서는 radix-2^k 모듈라 곱셈기의 구조, 가상 캐리 예측 덧셈기, 그리고 구현 가능한 최적의 radix 값을 제시하였다. 본문 Ⅲ장에서 제안된 하드웨어 구조를 Altera FPGA EP2K1500E와 삼성 0.35㎛ 공정을 사용하여 검증하였으며 칩의 면적과 연산 속도 차이를 본문 Ⅳ장에서 참고문헌과 비교, 검토하였다.

Ⅱ. Radix-2^k 모듈라 곱셈 알고리즘

RSA 지수승 연산은 몽고메리 모듈라 곱셈 알고리즘의 반복 연산이다.[5] 따라서, RSA 지수승 연산기는 모듈라 곱셈기를 어떻게 설계하느냐에 따라 속도와 설계 면적이 결정된다. 본 논문에서는 설계 면적을 최소화하면서 연산 속도를 높일 수 있는 radix-2^k 모듈라 곱셈기를 적용하여 RSA 지수승 연산기의 설계 방안을 제안한다.

2.1 Radix-2^k 모듈라 곱셈 알고리즘

CSA 기반의 radix-2^k 몽고메리 모듈라 곱셈 알고리즘은 Booth recording 알고리즘과 부호 발생 알고리즘[1]을 적용하였으며 n+k+1 비트 길이의 승수(A), n+k 비트 길이의 피승수(B), n+k 비트 길이의 모듈러스(N), 그리고 n+k 비트 길이의 매핑 계수(r)에 대하여 $A \cdot B \cdot r^{-1} \bmod N$을 연산하며 CSA 기반의 radix-$2^k$ 모듈라 곱셈 알고리즘은 [그림 1]과 같다.[12~14]

Function : $R2^kMM(A, B, N)$
Input : n+k+1 비트 승수(A), n+k 비트 피승수(B) n+k 비트 모듈러스(N) Output : $A \cdot B \cdot r^{-1} \bmod N$
Initialization
1: $Y \leftarrow \{-2^{k-1}, -2^{k-1}+1, \cdots, 0, \cdots, 2^{k-1}-1, 2^{k-1}\} \times B$ 2: $RT \leftarrow \{-2^{k-1}, -2^{k-1}+1, \cdots, 0, \cdots, 2^{k-1}-1\} \times N$
Modular Multiplication
3: $S_0, C_0 \leftarrow 0$ 4: for(i=0 ; i< $\lceil (n+k+1)/k \rceil$; i++) { 5: $s \leftarrow BE(A), PP_i \leftarrow SE(Y_s)$ 6: $T \leftarrow (S_i + C_i + PP_i) \pmod{2^k}$ 7: $S_i, C_i \leftarrow (S_i + C_i + PP_i)$ 8: $S_{i+1}, C_{i+1} \leftarrow (S_i + C_i + PP_i + RT(T)) / 2^k$} 9: $R \leftarrow S + C$ 10: Return R

[그림 1] Radix-2^k 모듈라 곱셈 알고리즘

CSA 기반의 radix-2^k 모듈라 곱셈 알고리즘은 [그림 1]에서와 같이 합(S)과 캐리(C)를 분리하여 연산하며 승수(A)와 피승수(B)의 radix 부분 곱(partial product : Y)과 radix reduction decoding (T) 값에 의한 reduction 연산을 위해 reduction table(RT)을 미리 연산하여 메모리에 저장한다. 모듈라 곱셈은 단계 5에서 승수(A)로부터 Booth encoding (BE) 값을 s에 전달하여 Y 메모리로부터 부분 곱(Y_s)을 호출하여 부호 확장된 PP_i를 얻으며, 반복 모듈라 곱셈 연산을 위한 부호 확장(sign extension : SE)된 radix-2^k 부분 곱[1]은 식 (1)에서 (4)와 같다.

$$Y = \left(\sum_{s=0}^{(n+k)/k-1} D_s \right) \cdot B \qquad (1)$$

$$D_s = A_{k\cdot i-1} + \sum_{j=1}^{k-1} A_{k\cdot i+j-1}\cdot 2^{j-1} - A_{k(s+1)-1}\cdot 2^{k-1} \quad (2)$$

$$PP_0 = \sum_{l=n+k+1}^{n+2k} 2^l + 2^{n+k} + \overline{Y_{s(n+k-1)}}\cdot 2^{n+k} + Y_s \quad (3)$$

$$PP_i = \sum_{l=n+k+1}^{n+2k} 2^l + \overline{Y_{s(n+k-1)}}\cdot 2^{n+k} + Y_s \ (i>0) \quad (4)$$

Radix-2^k의 부분 곱(Y)은 〔표 1〕의 Booth encoding (D_s)의 값에 피승수(B)를 곱한 값이며 이를 메모리에 미리 연산하여 저장한다. 부호 확장된 Booth 부분 곱(PP_i)은 부분 곱(Y_s)의 n+k-1 번째의 비트를 2진 보수를 취하여 부호를 확장하며 모듈라 곱셈 연산을 위해 식 (3)과 (4)에 따라 부호가 확장된다. 본 논문에서는 몽고메리 모듈라 reduction 연산을 위해 〔그림 1〕의 단계 6에서 k 비트를 얻어 RT의 메모리 주소를 발생한다. 예를 들어 radix-4와 radix-8에 대한 RT는 〔표 2〕와 〔표 3〕과 같다.

〔표 1〕 Radix-2^k에 대한 Ds의 집합[11]

k	2	3	m-1
D_s	{-2,-1,0, 1,2}	{-4,-3,⋯ 0,⋯3,4}	$\{-2^{m-2},-2^{m-2}+1,\cdots,0,\cdots,2^{m-2}-1,2^{m-2}\}$

〔표 2〕 Radix-4의 reduction table(RT)

$N(n_1n_0)$ \ $T(t_1t_0)$	00	01	10	11
01	0	-N	-2N	N
11		N		-N

〔표 3〕 Radix-8의 reduction table(RT)

$N(n_2n_1n_0)$ \ $T(t_2t_1t_0)$	000	001	010	011
001	0	-N	-2N	-3N
011		-3N	2N	-N
101		3N	-2N	N
111		N	2N	3N
$N(n_2n_1n_0)$ \ $T(t_2t_1t_0)$	100	101	110	111
001	-4N	3N	2N	N
011		N	-2N	3N
101		-N	2N	-3N
111		-3N	-2N	-N

Radix-2^k의 모듈라 reduction 연산은 T값과 N값에 의해 주소가 발생되어 RT 메모리에서 그 주소 값에 대한 값을 호출하며 radix-4인 경우 $t_1t_0n_1n_0$의 4비트를, radix-8인 경우 $t_2t_1t_0n_2n_1n_0$의 6비트를 주소로 사용한다.

2.2 Radix-2^k 모듈라 지수승 알고리즘

Radix-2^k 모듈라 지수승 연산은 Radix-2^k 모듈라 곱셈의 반복 연산으로 이루어진다. 본 논문에서는 radix-2^k 모듈라 지수승 연산을 위해 〔그림 2〕에서와 같이 지수 e가 오른쪽에서 왼쪽으로 스캔되는 R-L (right to left) 이진 연산 방식을 사용한다.

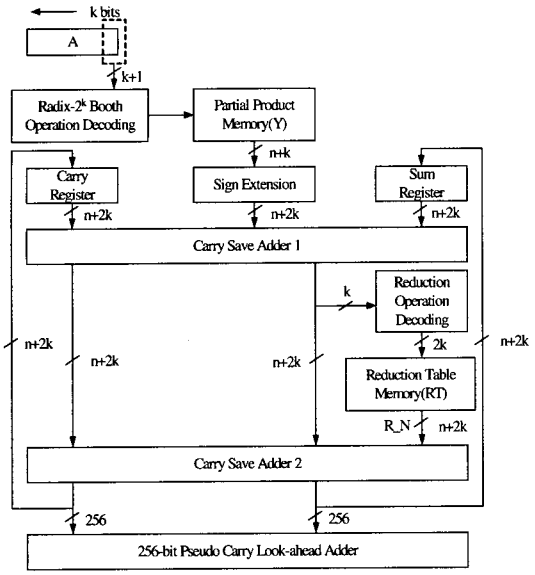
〔그림 2〕에서의 R-L 이진 연산 방식은 모듈라 지수승 연산이외에 mapping 연산으로써 M을 M·r mod N으로 변환하는 연산과 R의 초기값을 계산하는 과정을 포함한다. 그리고 M^E mod N값을 얻기 위하여 마지막에 remapping 연산을 수행한다. R-L 이진 연산 방식은 단계 4와 5과정이 서로 독립적으로 수행된다는 것이 R-L 이진 연산 방식의 특징으로서 실제 하드웨어로 구현 시 radix-2^k 모듈라 곱셈기 2개가 필요한 병렬 구조가 된다. 따라서 L-R(left-to-right) 이진 연산 방식보다 최대 100%, 지수 값의 분포도에 의해 평균 33%가 더 빠를 수 있으나 곱셈 연산과 제곱 연산을 동시에 수행하기 위하여 모듈라 곱셈기가 병렬로 두 개가 필요하므로 모듈라 곱셈기의 하드웨어 면적은 L-R 이진 연산 방식의 2배가 되는 단점이 있다.[9]

```
Function : R2^kME(M, E, N)
Input : 모듈러스(N), 지수(E), 평문(M),
        상수(C) ← 2^(2(n+k)) mod N
Output : M^E mod N
1: M'← R2^kMM(M, C, N),
   R ← R2^kMM(1, C, N) // mapping
2: for (i = 0; i<n ; i++) {
3:   If (e_i = 1) then
4:     R ← R2^kMM(R, M', N) // 곱셈
5:   M'← R2^kMM(M', M', N) } // 제곱
6: R ← R2^kMM(R, 1, N) // re-mapping
7: Return R
```

〔그림 2〕 Radix-2^k 모듈라 지수승 알고리즘

Ⅲ. Radix-2^k 모듈라 곱셈기 하드웨어 구조

Radix-2^k 몽고메리 모듈라 곱셈기는 앞서 언급한



〔그림 3〕 Radix-2^k 모듈라 곱셈기

바와 같이 두 입력 승수 A, 피승수 B, 그리고 모듈러스 N에 대해서 $R2^kMM(A, B, N)=A \cdot B \cdot r^{-1} \bmod N$을 출력하는 블록이며 전체 구조는 〔그림 3〕과 같다.

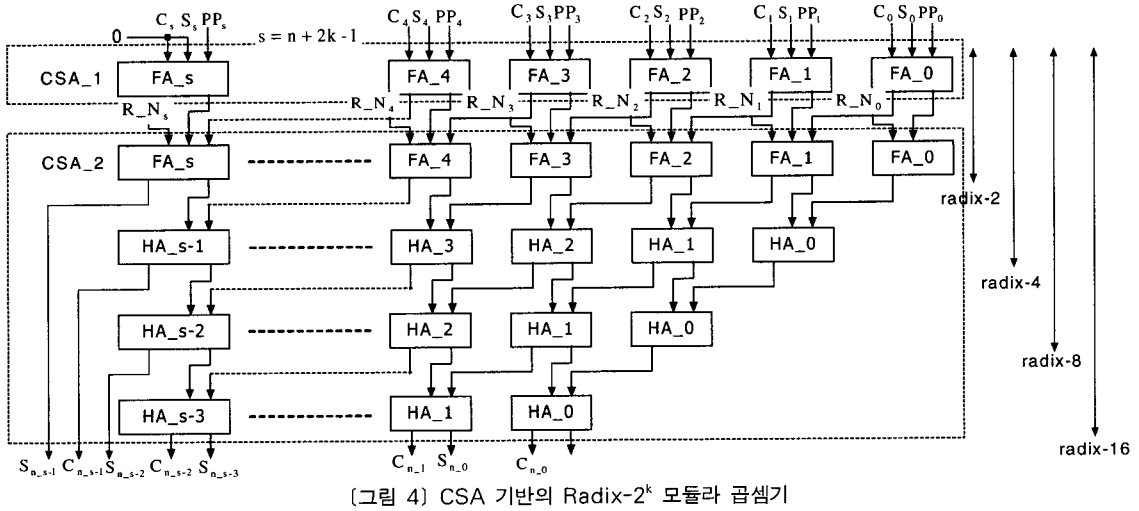
〔그림 3〕에서 CSA1은 〔그림 1〕의 단계 6과 7를, CSA2는 단계 8을 처리한다. 연산 과정 중간값은 carry와 sum 값으로 나누어져 있기 때문에 본 논문에서는 고속 덧셈을 수행하기 위해 PCLA(pseudo carry look-ahead adder : 가상 캐리 예측 덧셈기)[2,3]를 사용하여 마지막 덧셈을 수행한다.

3.1 Radix-2^k 몽고메리 모듈라 곱셈기

CSA 기반의 radix-2^k 모듈라 곱셈기는 〔그림 4〕에서와 같이 2FA(full adder)+(k-1)HA(half adder)+α(memory access time)의 지연을 가진다. 여기에서 α는 radix 부분 곱을 미리 계산하여 메모리에 저장하고 읽어오는 시간과 RT 메모리로부터 reduction 값을 읽어오는 시간을 합한 지연시간이며, radix 값에 의해 HA의 깊이가 결정된다. 〔그림 4〕에서 나타낸 모듈라 곱셈기는 1 클럭안에 합(S)와 캐리(C)를 출력하며 총 $\lceil (n+k+1)/k \rceil$번 반복 수행한다. 따라서 본 논문에서 제안한 〔그림 4〕와 같은 CSA 기반의 radix-2^k 모듈라 곱셈기는 설계 시 사용하는 공정과 동작 주파수에 따라 radix 값과 HA의 깊이가 결정되며 이 구조를 지수승 연산에 사용하면 총 $n \times \lceil (n+k+1)/k \rceil$의 클럭을 사용하여 동작 주파수 f에 대하여 연산속도는 식 (5)와 같이 결정된다.

$$(n \times \lceil (n+k+1)/k \rceil)/f \tag{5}$$

식 (5)에서와 같이 동작 주파수와 radix 값은 연산속도를 결정하는 중요한 값이며 〔그림 4〕의 구조로부터 설계 시 사용하는 공정에 따라 최적의 radix 값을 결정할 수가 있다. 본 논문에서는 삼성 0.35㎛ 공정을 사용하여 식 (5)와 같은 상관관계를 고려하여 최적의 radix 값을 시뮬레이션을 통해 결정하였다. 실제 시뮬레이션 결과 HA의 깊이에 따른 지연은 크게 변하지 않지만 Booth encoding 연산, 메



〔그림 4〕 CSA 기반의 Radix-2^k 모듈라 곱셈기

〔표 4〕 Radix 값에 대한 연산 속도 비교

n=1,024	지연 (ns)	최대 동작 주파수(㎒)	연산속도 (㎳)
k=3	12.79	67	3.92
k=4	16.83	59	3.57
k=5	22.35	44	3.99
k=6	39.58	25	7.03

〔표 5〕 모듈라 곱셈기의 지연 비교

radix	radix-4	radix-16
delay	3FA[12]	5.4FA[13]
	2FA+1HA*	2FA+3HA*

* : proposed

모리 저장, 그리고 메모리에서 값을 읽어 오는 시간이 전체 모듈라 곱셈기 지연에 지대한 영향을 미쳐 radix-32에서 오히려 성능이 저하됨을 알게 되었다. 전체 시뮬레이션 결과는 〔표 4〕와 같으며 본 논문에서 제시한 radix-2^k 모듈라 곱셈기 구조를 사용할 경우 radix-16이 최적이라 사료된다. 그러나 이는 사용 공정에 따라 그 지연시간에 영향을 받으므로 공정이 발전함에 따라 클럭 주파수를 높이면 radix를 높일수록 속도의 향상이 기대된다.

본 논문에서 제시한 CSA 구조는 〔표 5〕에서와 같이 systolic array 기반의 모듈라 곱셈기보다 지연이 작아 동작 주파수를 높일 수 있어 전체 연산속도를 향상 시킬 수 있다.

3.2 Pseudo Carry Look-ahead Adder[2,3]

CSA 기반 radix-2^k 모듈라 곱셈 알고리즘은 출력을 합(S)과 캐리(C)의 벡터형으로 출력하므로 최종 결과를 얻기 위해서는 합(S)과 캐리(C)를 더해야한다. 본 논문에서는 가상 캐리 예측 덧셈기(psedo carry look-ahead adder : PCLA)를 사용하여 고속 덧셈을 수행한다. PCLA에서 입력을 합(S)과 캐리(C)라 하면 i번째 PCLA의 출력은 식 (6)과 같다.

$$\mathrm{Sum}_i = H_{i-1} \cdot P_{i-1} \oplus \overline{P}_i \tag{6}$$

식 (6)에서 $P_i(=C_i+S_i)$는 캐리 전파 신호이며 $\overline{P}_i$는 $C_i \oplus S_i$이며, H_i는 가상 캐리이다. 모든 P_i와 $\overline{P}_i$ 신호는 병렬로 연산되며 H_i는 이전 식에 의해서 연산되며 일반식은 식 (7)과 (8)과 같다.

$$H_0 = G_0 + C_{in}, \tag{7}$$

$$H_i = G_i + H_{i-1} \cdot P_{i-1}. \tag{8}$$

일단 H_i가 구해지면 합을 고속으로 얻을 수 있으며 식 (9)와 같이 선택기(multiplexer)로 구현할 수 있다.

$$\mathrm{Sum}_i = \begin{cases} P_i, & \text{if } H_{i-1}=0 \\ P_{i-1} \oplus \overline{P}_i, & \text{if } H_{i-1}=1 \end{cases} \tag{9}$$

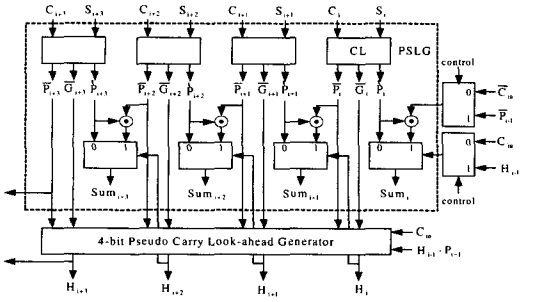
식 (9)에서 $P_{i-1} \oplus \overline{P}_i$는 $\overline{P}_{i-1} \odot \overline{P}_i$로 다시 쓸 수 있다. 식 (6)부터 (9)를 사용하여 4 비트 PCLA를 구성하면 〔그림 5〕와 같다. 일반적으로 캐리 예측 덧셈기를 설계할 때와 마찬가지로 긴 워드에 대하여 PCLA를 구성할 경우에는 〔그림 5〕에서와 같은 작은 블록들을 조합하여 다단계 그룹 PCLA를 구성할 수 있다.

〔그림 5〕를 블록으로 구성하면 그림 6과 같이 합을 출력하는 4 비트 pseudo sum look-ahead generator (PSLG)와 가상 캐리를 출력하는 4 비트 pseudo carry look-ahead generator (PCLG)로 구성할 수 있으며 〔그림 6〕의 4 비트 PCLA를 이용하여 16 비트 2 단계 그룹 PCLA를 〔그림 7〕과 같이 구성할 수 있다. 〔그림 6〕과 〔그림 7〕에서 Hg와 Ig는 식 (10), (11)과 같으며 다음 그룹 PCLA의 가상 캐리를 발생시킨다. Hg와 Ig에 의해 발생되는 그룹 가상 캐리는 식 (12)와 같다.

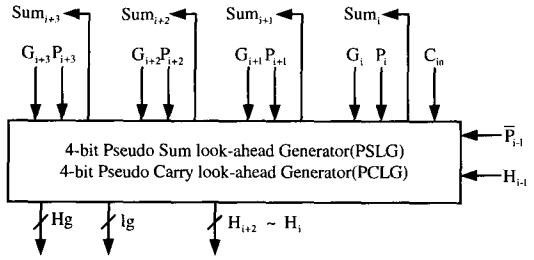
$$Hg = G_3 + G_2 + P_2 G_1 + P_2 P_1 G_0 \tag{10}$$

$$Ig = P_2 P_1 P_0 P_{i-1} \tag{11}$$

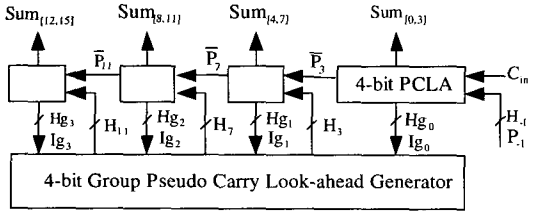
$$H_{i+3} = Hg + Ig \cdot H_{i-1} \tag{12}$$



〔그림 5〕 4 비트 가상 캐리 예측 덧셈기



〔그림 6〕 4 비트 그룹 PCLA 블록다이어그램



〔그림 7〕 2 단계 16 비트 GPCLA

〔표 6〕 게이트 지연 비교

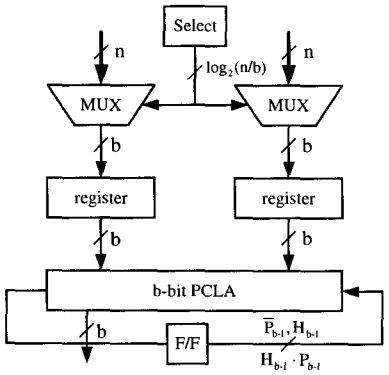
Adder	gate delay 일반식	gate delays 64 비트(r=4)
PCLA	$4\log_r k+1$	13
CLA[17]	$4\log_r k+1$	13
RCA	2k	128

〔그림 7〕의 게이트 지연을 살펴보면 다음과 같다.

1 gate delay(각 비트 위치에서 G와 P를 출력) +2 gate delays(4 비트 블럭의 입력을 위한 가상 캐리 신호 H_3, H_7, 그리고 H_{11}을 예측)+2 gate delays (각 4 비트 블록마다 내부 가상 캐리 예측)+2 gate delays(Hg와 Ig를 출력)+2 gate delays(합을 계산) =9 gate delays

〔표 6〕은 64 비트 리플 캐리 덧셈기(ripple-carry adder : RCA)와 캐리 예측 덧셈기(carry look-ahead adder : CLA)를 PCLA와 게이크 지연을 비교하였다. CLA와는 게이트 지연이 같지만 그룹 캐리(Cg)와 그룹 가상 캐리(Hg)를 구할 때 Cg는 4개의 fan-in, 10개의 항, 20개의 입력을 가짐에 비해,[17] Hg는 3개의 fan-in, 7개의 항, 14개의 입력을 가지므로 속도의 향상 및 면적을 절감할 수 있다.

본 논문에서는 CSA 기반 radix-2^k 모듈라 곱셈기 곱셈기를 위해 〔그림 8〕과 같이 PCLA를 구성하여 n 비트 크기의 연산자에 대해 w= ⌈(n+k)/(b=256)⌉ 번을 반복 수행하여 ABr^{-1} mod N의 최종 결과를 출력한다.



〔그림 8〕 PCLA 하드웨어 구조

Ⅳ. 성능 분석 및 PCI 인터페이스를 통한 보드 테스트

4.1 연산 속도 및 설계 면적 비교

본 논문에서 제시된 R-L 이진 연산 방식의 RSA 지수승 연산기를 삼성 0.35㎛ 공정을 사용하여 radix-8과 radix-16에 대하여 각각 설계하였다. radix-8과 radix-16은 각각 mapping과 remapping 연산에 총 $2\times(\lceil(n+3+1)/3\rceil+5)$의 클럭과 $2\times(\lceil(n+4+1)/4\rceil+5)$의 클럭을 사용하였으며, 모듈라 지수승 연산에 radix-8은 $n\times(\lceil(n+3+1)/3\rceil+5)$ 클럭을, radix-16은 $n\times(\lceil(n+4+1)/4\rceil+5)$ 클럭을 사용하였다. 따라서 총 클럭 수는 radix-8의 경우 $(n+2)\times(\lceil(n+3+1)/3\rceil+5)$가 되며, radix-16의 경우 $(n+2)\times(\lceil(n+4+1)/4\rceil+5)$가 되어 1,024 비트 RSA 지수승 연산 속도는 동작 주파수 50㎒에서 각각 7.13㎳와 5.38㎳의 성능을 보였으며 〔표 7〕과 같이 systolic array 기반의 모듈라 곱셈기를 사용한 RSA 지수승 연산기를 비교한 결과 전체적으로 약 2배의 연산속도를 가짐을 알 수 있다. 또한, 현재 시장에 출시된 RSA 암호 프로세서들[18,19]과 동

〔표 7〕 1,024 비트 RSA 지수승 연산을 위한 Clock cycles 수와 동작 주파수 50㎒에서의 연산 속도

radix	radix-2	radix-4	radix-16
Systolic	$2n^2$ [7]	n^2 [12]	$0.56n^2+72n$[13]
	42㎳[7]	20㎳[12]	13.2㎳[13]
CSA	n^2+36n[9]	$0.5n^2+7.5n$*	$0.25n^2+6.75n$*
	21.7㎳[9]	10.6㎳*	5.38㎳*

* : proposed

〔표 8〕 1,024 비트 RSA 프로세서 비교

RSA	[14]	[18]	[19]	radix-16*
동작 주파수(㎒)	45.6	55	50	50
동작 속도(㎳)	11.95	8.2	5.25	5.38

* : proposed

작 주파수에 따른 연산 속도의 성능을 〔표 8〕에서 비교한 결과 동일한 주파수에서 대등한 성능을 가짐을 알 수 있다.

본 논문에서는 R-L 이진 지수승 방식과 radix-16 모듈라 곱셈기를 적용한 1,024 비트 RSA 코어를 삼성 0.35㎛ 공정 사용하여 설계하였다. 그리고 모듈라 곱셈기의 설계 면적을 systolic 어레이 기반의 곱셈기[7,12,13]와 비교하기 위하여 같은 삼성 0.35㎛ 공정으로 설계한 결과 CSA를 기반으로 한 곱셈기가 설계 면적에서 약 50%의 이득을 얻었으며 그 결과를 〔표 9〕에 나타내었다. 그리고 본 논문에서 설계한 RSA 지수승 연산기의 각 블록에 대한 설계 면적을 〔표 10〕에 나타내었다. 〔표 10〕으로부터 본 논문에서 제시한 RSA 코어의 설계면적 분포를 알 수 있으며, 이중 메모리 부분을 포함한 radix-16 몽고메리 모듈라 곱셈기의 크기는 전체의 60%이상 차지한다. 따라서 전체 RSA 코어의 설계 면적을 줄이는 것은 모듈라 곱셈기의 최적화에 달려 있으며

〔표 9〕 모듈라 곱셈기의 설계 면적 비교

modular multiplier	radix-2	radix-4	radix-16
Systolic	67K[7]	111K[12]	142K[13]
CSA	60K[9]	78K*	92K*

* : proposed

〔표 10〕 각 블록의 게이트 수

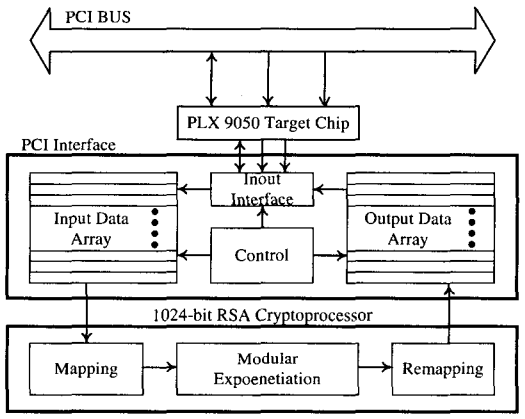
Blocks		Gate Counts
2 × Modular Multiplier	Register	2 × 10K
	CSA	2 21K
	PCLA	2 × 9K
	RAM(2Kbyte)	52K
Buffer		2 × 6K
Mux		12K
Controller		10K
Input Interface		4 × 6K
Output Interface		10K
Total		200K

본 논문에서는 look-up table을 사용하여 설계 면적을 줄일 수 있었다.

4.2 PCI 인터페이스를 통한 동작 검증 및 테스트

본 논문에서 설계된 radix-16 모듈라 곱셈기 기반의 1,024 비트 RSA 암호프로세서의 사용을 위하여 PCI 인터페이스 보드를 부가적으로 설계하였다. PCI 인터페이스 보드는 로컬 버스에 132MB/sec의 버스트 전송을 제공할 수 있는 PLX 9050 칩을 타겟으로 제작되었다.

PCI 인터페이스 보드는 그림 9에서와 같이 33㎒에서 동작하며 32-bit I/O, 입/출력 데이터 전송을 위한 데이터 어레이, 그리고 이러한 블록들을 제어하는 제어 블록으로 구성되어 있다. RSA 기능 검증을 위해 본 논문에서는 Altera FPGA EP2K1500E를 사용하여 설계하였으며 1,024 비트 RSA 암호프로세서는 〔그림 10〕과 같이 PCI 인터페이스 보



〔그림 9〕 1024 비트 RSA 암호프로세서를 위한 PCI 인터페이스

〔그림 10〕 1,024 비트 RSA 암호 프로세서의 FPGA 보드 테스트

드를 통하여 메시지 M, 공개키 e, 그리고 모듈러스 N을 전송하여 기능을 테스트하였다.

V. 결 론

본 논문에서는 네트워크 보안을 위한 RSA 시스템의 핵심 연산블록인 1,024 비트의 RSA 모듈라 지수승 프로세서를 radix-2^k 모듈라 곱셈 알고리즘을 적용하여 R-L 이진 지수승 연산 방식으로 설계하고 성능을 비교 검증하였다. 본 논문에서 제안된 radix-2^k 모듈라 곱셈기는 부호 확장된 부분 곱을 얻기 위해 메모리를 사용하고 있으며, 또한 모듈라 reduction 연산을 미리 저장된 reduction table을 통하여 처리하도록 하였다. 그리고 마지막 모듈라 연산 후 합(S)과 캐리(C)를 고속으로 처리할 수 있는 가상 캐리 예측 덧셈기(PCLA)를 사용하여 연산 처리 속도를 높일 수 있었다. Radix-2^k 모듈라 곱셈기는 설계 시 사용되는 공정에 매우 의존적이며 전가산기의 지연 시간과 메모리 접근 지연 시간에 따라 최적의 radix 값이 결정된다. 본 논문에서는 삼성 0.35㎛ 공정을 사용하여 16의 radix 값을 결정할 수 있었으며 동작 주파수 50㎒에서 5.38㎳ 연산 속도를 보였고, systolic array 기반의 모듈라 곱셈기보다 약 50%의 면적 감소와 2배의 연산 속도 증가를 가져왔다.

참 고 문 헌

[1] H. Sam, and A. Gupta, "A generalized Multibit Coding of Two's Complement Binary Numbers and Its Proof with Application in Multiplier Implementation," *IEEE Trans. Computers*, Vol. 39, No. 8, Aug., 1990.

[2] Flynn, M., "Topics in arithmetic for digital systems designers," (Preliminary Second Edition) pp. 104~105, 1995.

[3] Ling. H., "High speed binary adder," IBM J.Reasearch. Dev., Vol. 25, No. 3, pp. 156, May, 1981.

[4] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, 21(2):120-126, February, 1978.

[5] C. K. Koc, "RSA Hardware Implementation," Technical Report TR 801, *RSA Laboratories*, April 1996.

[6] C. K. Koc and C. Y. Hung, "Bit-level systolic arrays for modular multiplication," *Journal of VLSI Signal Processing*, pp. 215~223, 1991.

[7] C.Y. Su, S.A. Hwang, P.S. chen, and C.W. Wu, "An improved Montgomery algorithm for high-speed RSA public-key cryptosystem," *IEEE Trans. VLSI Systems*, Vol. 7, pp. 280~284, June 1999.

[8] T. Blum and C. Paar, "Montgoemry modular exponentiation on reconfigurable hardware," *14th IEEE Symposium on Computer Arithmatic*, pp. 70~77, 1999.

[9] T.W. Kwon, J.R. Choi and etc, "Two implementation methods of a 1024-bit RSA cyptoprocessor based on modified Montgomery algorithm," *Circuits and Systems, ISCAS 2001*, Vol. 4, pp. 650~653, Sydney, 2001.

[10] A.A. Hiasat, "New efficient structure for a modular multiplier for RNS," *Computers, IEEE Trans*, Vol. 49, pp. 170~174, Feb., 2000.

[11] W.L. Freking, K.K. Parhi, "Montgomery modular multiplication and exponentiation in the residue number system," *Signal, Systems, and Computers*, vol. 2, pp. 1312~1316, 1999.

[12] J.H. Hong and C. W. Wu, "Radix-4 modular multiplication and exponentiation algorithm," for the RSA public-key cryptosystem, *ASP-DAC* pp. 565~570, 2000.

[13] C.H. Wu, M.D. Shieh, C.-H. Wu, and etc, "A VLSI architecture of fast high-radix modular multiplication for RSA cryptosystem," Circuits and Systems, ISCAS 1999, Vol. 1, pp. 504~507, 1999.

[14] T. Blum and C. Paar, "High-radix Montgoemry exponentiation on reconfigurable hardware," *IEEE Trans. on Computers*,

Vol. 50, pp. 759~764, 2001.
[15] F. Tenca and C. K. Koc, "A scalable architecture for Montgomery multiplication," In C. K. Koc and C. Paar, editors, *Cryptographic Hardware and Embedded Systems*, Lecture Notes in Computer Science No. 1717, pp. 94~108, Springer, Berlin, Germany, 1999.
[16] F. Tenca, G. Todorov, and C. K. Koc, "High-radix design of a scalable modular multiplier," In C. K. Koc and C. Paar, editors, *CHES2001*, pp. 185~201, Springer, Paris, France, 2001.
[17] Lynch, T., and E. Swartzlander, "A spanning tree carry lookahead adder," *IEEE Trans. on Computers*, Vol. 41, No. 8, pp. 931~939, 1992.
[18] STI Security Techologies inc., SCC102, http://www.stitec.com/product/ecryptoxpress.html.
[19] Hi/fn Company, Hi/fn 6500, http://www.hifn.com/docs/6500.pdf.

〈著者紹介〉

권 택 원 (Taek-Won Kwon)
1998년 2월 : 경북대학교 전자공학과 졸업
2000년 2월 : 경북대학교 전자공학과 석사
2000년 3월~현재 : 경북대학교 전자공학과 박사과정
〈관심분야〉 암호칩설계, 정보보호 및 응용

최 준 림 (Jun-Rim Choi) 정회원
1986년 2월 : 연세대학교 전기공학과 졸업
1988년 8월 : 미국 Cornell 대학교 전자전기공학과 석사
1991년 7월 : 미국 Minnesota 대학교 전자전기공학과 박사
1987년 6월~1988년 5월 : 미국 National Nanofabrication Facility 연구원
1988년 6월~1989년 5월 : 미국 center for Microtechnology(MEIS) 연구원
1991년 7월~1997년 2월 : LG전자기술원
1997년 3월~현재 : 경북대학교 전자전기컴퓨터학부 조교수
〈관심분야〉 암호칩설계, 영상압축칩설계,