파이프라인 구조의 고속 RSA 암호화 칩 설계
(Design of a Pipelined High Performance RSA Crypto_chip)

이 석 용 † 김 성 두 †† 정 용 진 †††
(Seok-Yong Lee) (Seong-Doo Kim) (Yong-Jin Jeong)

요 약 본 논문에서는 RSA 암호 시스템의 핵심 과정인 모듈로 멱승 연산에 대한 새로운 하드웨어 구조를 제시한다. 본 방식은 몽고메리 곱셈 알고리즘을 사용하였으며 기존의 방법들이 데이타 종속 그래프(DG: Dependence Graph)를 수직으로 매핑한 것과는 달리 여기서는 수평으로 매핑하여 1차원 선형 어레이 구조를 구성하였다. 그 결과로 멱승시에 중간 결과값이 순차적으로 나와서 바로 다음 곱셈을 위한 입력으로 들어갈 수 있기 때문에 100%의 처리율(throughput)을 이룰 수 있고, 수직 매핑 방식에 비해 절반의 클럭 횟수로 연산을 해낼 수 있으며 컨트롤 또한 단순해 지는 장점을 가진다. 각 PE(Processing Element)는 2개의 전가산기와 3개의 멀티플렉서로 이루어져 있고, 암호키의 비트수를 k비트라 할 때 k+3개의 PE만으로 파이프라인구조를 구현하였다. 1024비트 RSA데이타의 암호 또는 복호를 완료하는데 $2k^2+12k+19$의 클럭수가 소요되며 클럭 주파수 100Mhz에서 약 50kbps의 성능을 보인다. 또한, 제안된 하드웨어는 내부 계산 구조의 지역성(locality), 규칙성(regularity) 및 모듈성(modularity) 등으로 인해 실시간 고속 처리를 위한 VLSI 구현에 적합하다.

Abstract This paper proposes a new hardware architecture for modular exponentiation which is a core arithmetic in RSA(Rivest-Shamir-Adleman) cryptosystem. Using Montgomery's algorithm, we projected the DG(Dependence Graph) to horizontal direction, although all the previous works have used vertical projection. As a result of horizontal mapping, the intermediate result during modular exponentiation appears serially at the output, i.e. LSB(Least Significant Bit) first, and is just ready for the input of the next multiplication operation. This fact enables us to accomplish 100% throughput with a simple control logic and to perform a full exponentiation with a half clock cycles of the vertical mapping. There are k+3 PE(Processing Element)s. Each PE has two full adders and three multiplexers. A complete 1024-bit RSA encryption and decryption requires $2k^2+12k+19$ clock cycles, and hence we have more than 50kbps (assuming 100Mhz clock frequency). Due to its locality, regularity and modularity of the PE's internal structure, the proposed architecture is adequate for high speed VLSI implementation.

1. 개 요

최근 몇 년간 전자상거래 등에 필요한 정보 암호화(encryption), 전자 서명 및 인증(electronic signature and authentication)에 대한 관심과 수요가 급증하고 있다. 공개키 방식 암호 시스템은 이러한 요구를 충족시키는 기본적인 알고리즘을 갖추고 있는데, 그 대표적인 것이 1978년에 R. Rivest, A. Shamir, L. Adleman에 의해서 제안된 RSA알고리즘이다[1]. RSA알고리즘은 1024비트 이상의 큰 정수를 기반으로 한 모듈로 연산에 의해 수행되며, 큰 정수 계수(modulus)의 소인수 분해가 매우 어려움에 그 안전성의 근거를 두고 있다. RSA를 위한 모듈로 연산은 내부에 곱셈과 나눗셈이 복합되어 있어 계산 구조가 복잡하고 워드 사이즈가 1024비트 이상으로 크기 때문에 고속 실시간 처리를 위한 하드웨어 모듈의 구현이 어렵다. 그러나, 근래 들어서는 VLSI 설계 기술의 발달과 함께 다양한 연산 알고리즘들이 연구되어 고속 RSA모듈 구현에 대한 연구가 활발해지고

· 이 논문은 1999년도 한국학술진흥재단의 연구비에 의하여 지원되었음. (KRF-99-003-E00376)
† 학생회원 : 광운대학교 전자통신공학과
sylee@explore.kwangwoon.ac.kr
†† 비 회 원 : 광운대학교 전자통신공학과
sdkim@explore.kwangwoon.ac.kr
††† 비 회 원 : 광운대학교 전자공학부 교수
yjjeong@daisy.kwangwoon.ac.kr
논문접수 : 2000년 4월 20일
심사완료 : 2001년 5월 4일

있으며[2][3][4] [5][6][7], 하드웨어로 구현했을 경우 키의 안전성 면에서도 소프트웨어보다 월등하다는 장점을 가진다.

RSA모듈 구현에 대한 기존의 방법들을 살펴보면, Jeong과 Burleson은 몽고메리 알고리즘과는 달리 중간 결과값에 대한 모듈로 감소(modulo reduction)과정에서 계수의 뺄셈을 결정할 때 MSB(Most Significant Bit) 우선 방식을 사용하였다[4]. 여기서는 미리 계수의 보수들을 계산해서 레지스터에 저장해 놓고 look-up하는 단순한 방식을 사용하였는데, 이 때문에 PE가 아주 간단해 져서 빠른 클럭 주기를 가능하게 한다. 미리 계산된 계수의 보수들을 저장하기 위한 추가의 레지스터가 더 들어가지만, 후처리(post processing)가 필요하지 않기 때문에 적은 양의 계산시에는 몽고메리 알고리즘보다 유리할 수 있다. 그러나, 멱승 계산처럼 연속된 곱셈을 요하는 경우에는 그 장점이 몽고메리 방법에 비해 크게 드러나지는 않는다. Walter는 몽고메리 알고리즘을 이용하여 모듈로 곱셈을 2차원 평면상에 시스톨릭 어레이(systolic array)로 구현하였다[5]. k비트의 입력에 대하여 처음으로 출력이 나오기 시작할 때까지 2k+2의 클럭 수가 소요되지만, 2차원 평면상에 구현된 것이라 그 크기 때문에 실제로 하드웨어로 구현하기는 힘들다.

Thomas Blum과 Christof Paar는 몽고메리 알고리즘을 이용한 멱승구조를 하나의 FPGA(Field Programmable Gate Array)에 들어갈 수 있도록 PE들의 한 줄만으로 매핑하여 구현하였다[6]. $(R_i+a_iB+q_iN)/2$ 의 계산을 매번 반복하기 보다는 미리 B+N값을 계산해 놓고 a_i와 q_i값에 따라서 0, N, B, B+N값을 멀티플렉서를 이용하여 선택하는 방식을 취함으로써 덧셈기를 하나로 줄였다. 그러나, 멱승시에는 매 곱셈이 끝날 때마다 B+N을 다시 계산해야 하고, 처리율을 높이기 위해 제곱과 곱셈의 인자를 인터리빙하기 때문에 추가의 레지스터와 컨트롤이 복잡해지는 단점이 있다. k비트의 입력에 대해서 출력이 나오기까지는 2(k+2)(k+4)의 클럭 사이클이 소요되며, 1024비트에 대해 전체 모듈로 멱승 수행시 50Mhz의 주파수로 약 26Kbps의 성능을 보인다. Shand와 Vuillemin은 속도를 높이기 위한 다양한 방법으로 CRT (Chinese Remainder Theorem), carry completion adder, quotient pipelining을 사용하여 그들이 새로이 제안한 프로그래머블 능동 메모리(PAM: Programmable Active Memory)구조에 구현하였다[7]. 그러나, RSA암호화와 복호화시 서로 다른 PAM디자인을 사용하였고, 계수가 곱셈기안에 hard-wired되어 있어서 계수가 바뀔 때마다 아키텍쳐를 다시 구성해야 하는 단점을 가지고 있다. Carry save형태의 redundant binary representation을 사용하였기 때문에 한번의 곱셈이 끝난후 다음의 곱셈과정으로 들어가기 위해 non-redundant 형태로 바꾸어야 하는데, 이 과정에서의 딜레이를 줄이기 위해 asynchronous carry completion detection 회로를 사용하였다. CRT를 사용하여 복호화시 1024비트의 키에 대해서 약 165kbps의 속도를 이루어 최근까지 발표된 결과 중 가장 좋은 성능을 갖는 것으로 보인다.

본 논문에서는 처리 속도를 높이기 위해 파이프라인 1차원 어레이 구조를 갖도록 곱셈기를 설계하였으며, 파이프라인 구현시 캐리의 전달과 데이타의 처리순서가 일치하는 몽고메리 알고리즘을 사용하였다. 또한, 기존의 발표된 방법들과는 달리 DG에 대한 수평매핑을 통하여 100%의 처리율을 갖도록 하였다. 제안된 구조는 이진 연산을 기본으로 하고 있으며, 동일한 설계 방법론 및 하드웨어 구조를 하이 래딕스(high radix)연산에 적용할 경우 추가적인 성능향상이 예상된다. 본 논문의 구성은 다음과 같다. 2장에서는 RSA알고리즘에 대해 설명하고, 몽고메리 모듈로 곱셈 알고리즘을 분석하며 이를 이용하여 곱셈기를 설계하는 과정을 설명한다. 3장에서는 멱승과정을 효율적으로 처리하기 위한 방법과 2장에서 구현한 곱셈기를 이용해서 전체 RSA암호 시스템을 구현하는 과정에 대해 설명한다. 마지막으로 4장에서는 결론으로 제안된 하드웨어 구조의 시간과 공간 복잡도 및 성능을 분석한다.

2. RSA 알고리즘과 모듈로 연산

RSA 암호시스템은 1024비트 이상의 크기를 갖는 계수 N과 $M=M^{DE}$ mod N인 관계를 갖는 두 개의 키(key) D, E를 가진다. 계수 N은 두 개의 큰 소수 P, Q의 곱으로 이루어지며 키 E는 $\phi(N)$이 Euler's totient function 즉, $\phi(N)=(P-1)(Q-1)$일때 gcd(E, $\phi(N)$)=1인 관계를 갖는 $1<E<\phi(N)$의 수 중에서 하나를 선택한다. 키 D는 E^{-1} mod $\phi(N)$으로 계산 되고, E와 N을 공개키로서 공개하며, D와 P, Q는 비밀키로서 공개되지 않는다. 공개하지 않는 P와 Q를 계수 N으로부터 정확히 소인수분해하기는 1024비트 이상의 크기 때문에 사실상 불가능하다. 이 키들을 이용한 RSA 암호시스템 알고리즘의 핵심이 되는 암호화 및 복호화 연산은 식 1과 같다.

$$C=M^E \bmod N,\ \text{where } E=\sum_{i=0}^{k-1} e_i \times 2^i$$
$$M=C^D \bmod N,\ \text{where } D=\sum_{i=0}^{k-1} d_i \times 2^i \quad (1)$$

식 1에서 보는 바와 같이 RSA암호시스템은 공개키 E나 비밀키 D에 대해 모듈로 멱승을 취함으로써 암호화(C) 및 복호화(M)가 이루어 지며, 두 과정이 인자만 바뀌고 동일한 연산으로 이루어짐을 알 수 있다.

따라서, RSA 연산의 핵심은 모듈로 멱승(modular exponentiation)이며 이는 모듈로 곱셈의 연속으로 이루어진다. 또한 모듈로 곱셈은 연속된 덧셈 연산으로 수행될 수 있는데, 곱셈 연산시 곱셈을 먼저 한 후 모듈로 연산을 하는 곱셈 후 모듈로 감소(modulo reduction after multiplication)방식과 곱셈 중에 모듈로 연산을 반복하는 곱셈중 모듈로 감소(modulo reduction during multiplication)방식이 있다. 모듈로 곱셈기의 하드웨어 구현시 전자는 입력의 비트수를 k 비트라 할 때, k×k 비트의 곱셈기와 2k비트의 레지스터 그리고 2k×k비트의 나눗셈기 등 리소스를 많이 차지하기 때문에 주로 후자의 방법을 사용한다. 곱셈중 모듈로 감소 방법에는, 연속적인 덧셈에 의해서 만들어지는 중간 값(partial product)에 대해 MSB방향으로 커지는 비트를 보고 적당한 계수의 배수를 뺌으로써 새로 늘어난 비트를 제거하는 방법으로 전체적인 비트수를 감소, 유지시키는 MSB(JB 알고리즘)우선 방식[4]과 LSB(Least Significant Bit)부분을 '0'으로 만드는 적당한 계수의 배수를 더한 다음 오른쪽으로 쉬프트하는 방법으로 LSB부분을 제거함으로써 전체적인 비트수를 감소, 유지시키는 LSB(몽고메리 알고리즘)우선 방식[3]이 있다. 이 중에 몽고메리 알고리즘은 계수의 덧셈에 대한 선택이 하위 비트에서 이루어지기 때문에 파이프라인으로 구현시 캐리의 지연을 고려치 않아도 되는 장점이 있어 주로 사용되는 방법이다.

이진 연산 방식의 몽고메리 알고리즘을 이용한 모듈로 곱셈의 계산은 다음과 같다.

MonPro(A, B)
  $P_{(0)}=0$
  for i=0 **to** k−1 **do**
    $P^{*}_{(j+1)}=(P_{(j)}+b_j \cdot A)$
    if $P^{*}_{(j+1)}$ is odd, **then** $P_{(j+1)}=(P^{*}_{(j+1)}+N)\cdot 2^{-1}$
    else $P_{(j+1)}=(P^{*}_{(j+1)})\cdot 2^{-1}$
  return $P_{(k)}$

위에서 보는 바와 같이 몽고메리 알고리즘 MonPro는 중간 결과 P_j가 홀수이면 계수 N을 더하여 짝수로 만든 후 오른쪽으로 쉬프트하는 방법으로 비트수를 줄여나가는데 이는 RSA 암호시스템에서 계수 N이 홀수이기 때문에 가능하다. 마지막 결과값은 기수(radix)로 k(키의 비트수)번 나눈 결과이므로 후처리로 기수를 k번 곱해야 정확한 결과를 얻게 된다. 즉, $R^{*}=A\cdot B\cdot 2^{-k} \bmod N$이므로 후처리로 $R^{*}\cdot 2^{k} \bmod N$을 수행하여 원하는 결과값 $R=A\cdot B \bmod N$을 얻는다.

앞에서 밝힌 바와 같이 RSA의 주된 연산인 모듈로 멱승은 연속된 모듈로 곱셈으로 구성되므로 효율적이고 빠른 RSA 암호 시스템 구현의 관건은 모듈로 곱셈기의 설계에 있음을 알 수 있다. MonPro알고리즘의 내부 계산 구조를 보여주는 DG를 k=4비트에 대해서 그림 1에 보였다.

몽고메리 곱셈의 입력 A, B, N이 k비트일때, MonPro알고리즘의 반복문에서 중간 결과값이 k+2비트까지 나올수 있기 때문에 MSB의 캐리를 염두에 두면 DG의 한 행은 k+1비트가 되어야 하므로 $a_k=0$, $n_k=0$

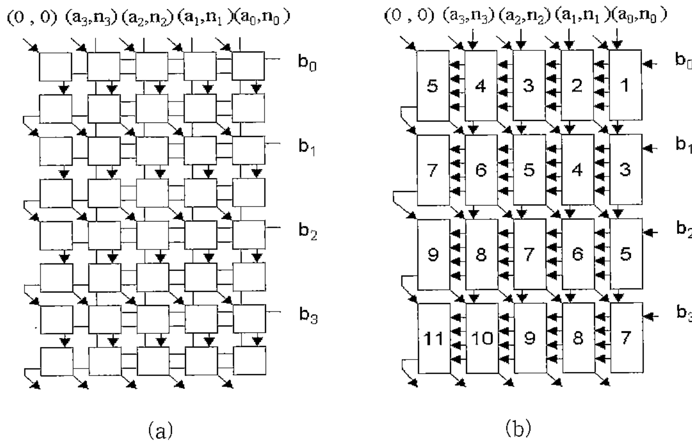
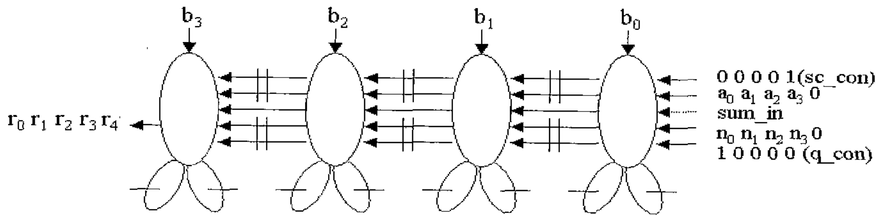


그림 1 몽고메리 알고리즘의 Dependence Graph

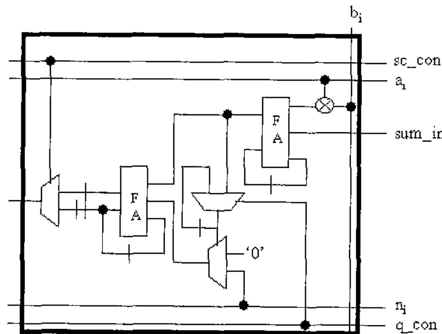
으로 A, N을 k+1비트로 확장 표현하였다. 그림 1의 (a)는 $P_{(j)}+b_j A$모듈과 N을 더하여 쉬프트하는 모듈을 나누어 그린 것으로 각 노드는 한 개의 전가산기를 가지고 있으며, (b)는 (a)의 두개의 모듈을 하나로 묶은 간결화된 DG이다. 각각의 화살표는 그 방향으로의 데이타 흐름을 나타내고 각 행은 MonPro알고리즘에서 곱셈과 쉬프트의 반복문을 의미한다. 각 셀(cell)들은 하나의 비트에 대하여 곱셈과 쉬프트를 수행하고 제일 우측 열은 N을 더할 것이지를 판단하는 부분만 제외하고는 다른 셀들과 동일한 동작을 수행한다. 또한 PE안의 숫자는 각 셀들 사이에 얼마의 딜레이가 존재해야 하는 지를 보여준다. 그림 1에서 보는 바와 같이 DG의 내부 계산 구조는 매우 규칙적이고 정형적인 구조를 가지는데, 이를 이용하여 여러 종류의 모듈로 곱셈기의 구현이 가능하다[4] [5] [6] [7].

지금까지의 모든 논문들이 모듈로 곱셈기를 구현할 때 그림 1의 DG를 수직으로 매핑한 것과는 달리 본 논문에서는 수평으로 매핑하였다. 그로 인해서 얻어지는 장점은 수직 매핑의 50% 처리율을 개선하여 1024비트 전체에 대해 100% 처리율을 갖는 파이프라인 구조가 가능하다는 것이다. 수직 매핑은 데이타 흐름이 양방향이고, 결과는 2k부터 나오지만 옆에서 들어가는 입력은 2k-1에 걸쳐서 한 클럭씩 쉬면서 들어가므로 100% 처리율을 구현하기가 어렵다. 100% 처리율을 이루기 위해서는 Paar[6]의 경우처럼 인터리빙방식으로 사이사이에 제곱과 곱셈에 들어가는 인자를 스위칭하면 되지만 그럴 경우 입력값을 저장할 레지스터가 2개 필요하고 또한 결과가 한클럭씩 지연되면서 오버랩(overlap)되어 나오기 때문에 결과 또한 2개의 레지스터에 나누어서 저장해야 한다. 결과적으로 출력이 병렬로 나오는 효과를 가져온다. 반면, 수평 매핑은 데이타 흐름이 단방향이고, 결과는 수직매핑과 똑같이 2k부터 나오지만 옆에서 들어가는 입력은 매 클럭마다 하나씩 k의 구간에 들어가므로 나머지 k의 시간에 다른 계산(곱셈 또는 제곱)의 인자를 넣을 수 있다. 또한 위에서 병렬로 들어가는 입력은 1, 3, 5, 7,…의 순서로 제 시간에 들어가기만 하면 되므로 레지스터와 멀티플렉서를 써서 시리얼로 나오는 중간 결과값을 바로 받을 수 있도록 구현이 가능하다. 따라서 수직매핑에 비해 컨트롤이 한결 쉬워지며 100% 처리율을 가지면서도 구현이 용이한 장점이 있다. 반면, 파이프라인을 위한 레지스터가 증가하는 단점도 존재한다.

그림 1을 수평으로 매핑하여 얻은 SFG(Signal Flow Graph)와 PE는 그림 2와 같으며 'ㅣ'(bar)는 플립플롭을 나타낸다. MonPro알고리즘의 반복문을 간단히 표현하면 $P_{(i+1)}=(P_i+b_iA+q_iN)/2$으로 표현할 수 있는데, 여기서 q_i는 $P_{(i+1)}$가 2로 나누어 떨어지도록 선택된다. 즉,



(a) Signal Flow Graph



(b) Processing Element

그림 2 몽고메리 곱셈의 수평매핑에 대한 SFG 및 PE

$P_i + b_iA$ 값에 N을 더할 것인가를 결정하는 것이다. 몽고메리 알고리즘에서는 이러한 결정이 LSB에서 이루어지기 때문에 그림 2에서와 같이 q_con을 (1, 0, ⋯, 0)으로 넣어서 최하위 비트에서 N을 더할 것인지를 결정하고 그 결정이 전체 a_i에 대해서 유지 되도록 구현하였다. 또한 A의 MSB까지 계산 되었을 때는 출력의 캐리까지 의미있는 값으로 받아줘야 하기 때문에 sc_con을 넣어서 sum과 carry를 선택하도록 하였으며 sc_con과 sum이 한 클럭의 딜레이 차이가 나는 것은 중간 결과값의 최하위 비트는 쉬프트 되면서 없어지기 때문이다.

몽고메리 알고리즘 MonPro에서 초기 입력 조건을 A, B<2N이라 할 때, $P_{(k)}<3N$이다. 이렇게 되면 멱승 연산시 다음 곱셈을 위한 입력값으로는 적합하지가 않기 때문에 여분의 '0'를 입력에 패딩(padding)하는 방법을 사용하여 k+3비트를 입력으로 넣어주면 $P_{(k)}<2N$으로 항상 입력조건에 부합하게 되어 멱승시에 중간 결과값에 대한 추가의 모듈로 감소 과정을 거칠 필요가 없어지게 된다[6]. 그러므로, 여분의 3비트를 더 붙임으로써 3클럭이 더 소요되지만 추가적인 처리과정 없이 바로 다음 계산의 입력으로 들어갈 수 있어 효율적인 파이프라인 구조의 구현이 가능해진다.

그림 1의 DG를 수평 매핑하고 비트수를 k+3으로 확장하여 그림 3에 다시 보인다. 그러나 이것을 그대로 2차원으로 구현하면 하드웨어 복잡도(hardware complexity)가 k^2이 되어 상당한 리소스를 필요로 하기 때문에 하드웨어 복잡도가 k가 되도록 한 행만으로 파이프라인 구조의 1차원 선형 어레이 곱셈기(linear array multiplier)를 구현하였다.

3. 새로운 RSA 모듈로 멱승기의 제안

RSA암호 알고리즘의 핵심이 되는 모듈로 멱승연산은 곱셈의 연속으로 수행시 계산량이 너무 많아지게 된다. 계산량을 줄이기 위한 방법으로 지수에 따라 일련의 제곱과 곱셈으로 멱승을 수행하는 방법이 있는데, 그 중 하나인 binary method는 지수(exponent)를 스캔하는 방향에 따라 MSB부터 스캔해 나가는 LR-method (Horner's rule)와 LSB부터 스캔해 나가는 RL-method의 두가지 방법이 있다[2].

수평으로 매핑한 경우 입력이 (k+3)클럭에 걸쳐 들어간후 결과가 2(k+3)부터 나오기까지 (k+3)의 클럭을

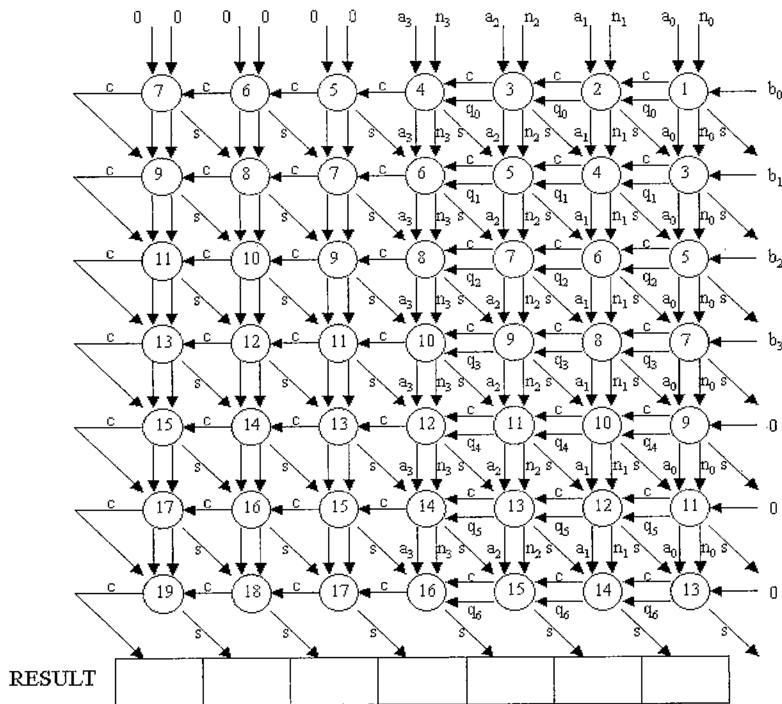


그림 3 k+3비트로 확장된 수평매핑 DG

쉬기 때문에 이를 이용하기 위하여 그림 4처럼 제곱과 곱셈이 서로 독립적으로 병렬 처리될 수 있는 RL-method를 이용하여 멱승을 구현하였으며, 그림 4가 작동하는 예를 표 1에 보였다.

그림 4와 표 1에서 보는 바와 같이 멱승은 제곱과 곱셈을 번갈아 함으로써 연산이 가능하기 때문에, 지금까지 논의해 온 곱셈기를 그대로 사용하며, 데이타 흐름의 적절한 컨트롤로써 수행된다.

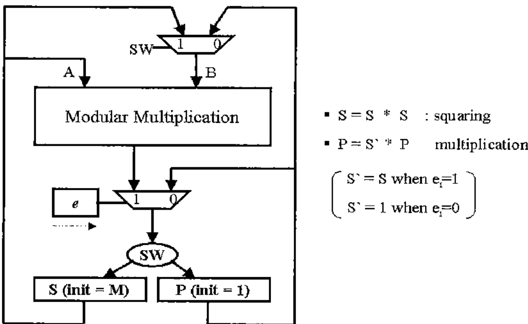


그림 4 RL-method

표 1 RL-method의 예 <M^1011 mod N>

	$E_0=1$		$E_1=1$		$E_2=0$		$E_3=1$			
SW	1	0	1	0	1	0	1	0	1	0
A	M	M	M^2	M^2	M^4	M^4	M^8	M^8	M^{16}	M^{16}
B	M	1	M^2	M	M^4	M^3	M^8	M^3	M^{16}	M^{11}
P	1	1	M	M	M^3	M^3	M^3	M^3	M^{11}	M^{11}

몽고메리 알고리즘은 곱셈시 후처리를 꼭 해 주어야 하기 때문에 적은 양의 곱셈시에는 오히려 비효율적이다. 그러나, 멱승과 같이 여러 번의 곱셈을 수행할 경우에는 전처리와 후처리를 멱승의 처음과 마지막에 1회만 해주면 되기 때문에 그 오버헤드는 상대적으로 무시될 수 있을 정도로 줄어들게 된다. 이에 대한 모듈로 멱승 알고리즘 MonExp는 다음과 같다.

MonExp(M, E, N)

/*($z=r^2$ mod N) is precalculated, where $r=2^{(k+3)}$ */

$M^*=M \cdot r$ mod N=MonPro(M, z)

$X^*=1 \cdot r$ mod N=MonPro(1, z)

for i=0 **to** k-2 **do**

  if $e_i=1$ **then** X^*=MonPro(M^*, X^*)

  M^*=MonPro(M^*, M^*)

if $e_{k-1}=1$ **then** X^*=MonPro(M^*, X^*)

X=MonPro(X^*, 1)

return X

MonPro(A,B)$=A \cdot B \cdot 2^{-(k+3)}$ mod N이므로 올바른 중간 결과값을 얻기 위해서는 후처리로 $2^{(k+3)}$의 곱셈이 더 필요하다. 이를 중간 결과값들에 대해 매번 수행하게 되면 매우 비효율적이므로 그림 5처럼 초기 입력값을 $2^{2(k+3)}$으로 미리 곱해 놓고 계산하는 것이 편리함을 알 수 있다. 모든 중간 결과값은 R^*=MonPro(A^*, B^*)$=A \cdot B \cdot 2^{(k+3)}$ mod N처럼 $2^{(k+3)}$의 factor를 가지게 되므로 최종 결과값에 간단히 1을 곱하는 후처리를 해 주면 원하는 결과값을 얻게된다.

전처리 :

A^*=MonPro(A, $2^{2(k+3)}$)$=A \cdot 2^{(k+3)}$ mod N

B^*=MonPro(B, $2^{2(k+3)}$)$=B \cdot 2^{(k+3)}$ mod N

후처리 :

R=MonPro(R^*, 1)$=R^* \cdot 1 \cdot 2^{-(k+3)}$ mod N

$=A \cdot B$ mod N

본 논문에서 제안하는 방식은 그림 2의 새로운 선형 모듈로 곱셈기를 기본으로 하며, MonExp알고리즘의 멱승계산을 효율적으로 처리하도록 하기 위한 모듈로 멱승의 과정을 입력값을 받아서 결과가 출력될 때까지의 전체 RSA연산 프로세스로써 각 구간별로 나누면 그림 5와 같다.

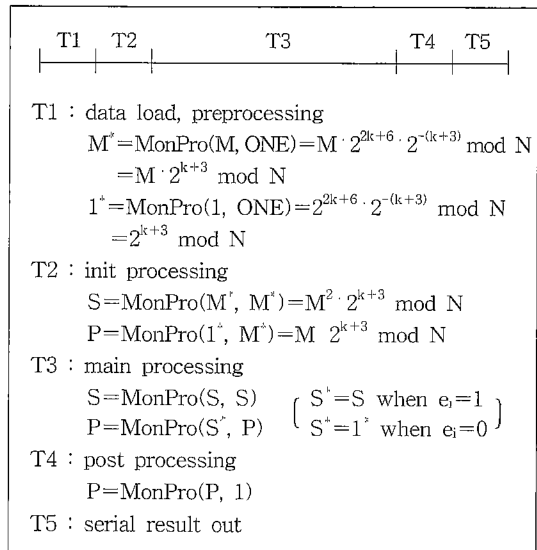


그림 5 모듈로 멱승기의 전체 Process

T1 동안에는 각 키 데이타들을 레지스터에 로드하고 앞에서 기술한 것처럼 중간의 처리과정을 없애기 위해서 초기 입력 데이타에 2^{k+3}의 factor를 붙이는 과정이고, T2는 전처리의 결과값을 M과 ONE 레지스터에 시리얼로 로딩하는 프로세스이다. 이 때 M과 ONE레지스터는 곱셈기를 거쳐 계산된 값을 받아야 하기 때문에 다음 계산에 입력으로 들어가기 전에 k+3클럭의 쉬는 시간이 존재하게 된다. T3는 주된 연산인 모듈로 멱승을 수행하고, T4는 후처리로 원하는 결과값을 얻어내기 위해 최종 결과값에 1을 곱하는 과정이며, 그 결과가 T5에 시리얼로 나오게 된다. 여기서 주목할 것은 암호칩의 계산 모듈이 쉬지 않고 100%의 처리율로 계속해서 데이타를 받아 처리한다는 점이다. 이 구조의 경우 필요한 메시지와 키를 로드한 다음 시작 신호만 주고 정해진 클럭 수만큼 후에 데이타를 읽으면 암호화 혹은 복호화가 마무리된다. 즉, 암호화 과정에 소프트웨어의 컨트롤이 들어가지 않기 때문에 빠른 처리 속도를 이룰 수 있는 것이다.

지금까지 상술한 매핑과정과 프로세스에 따라 구현한 멱승기의 전체구조는 그림 6과 같다. 칩 내부는 크게 데이타 패스, 컨트롤, 그리고 레지스터 블록의 세 부분으로 나누어져 있다. 컨트롤 블럭에는 그림 5에 보인 구간별 동작을 제어하기 위한 카운터들이 들어 있으며, 데이타 패스는 bit-slice 개념으로 잘 정렬되어 있다.

모든 레지스터는 T1이 시작하자 마자, 데이타가 로딩되어야 하므로 병렬 로딩기능이 있어야 하고 특히, M과 ONE레지스터는 전처리된 값을 다시 받아야 하므로 시리얼 로딩 기능도 가져야 한다. 구현한 시스템은 결과

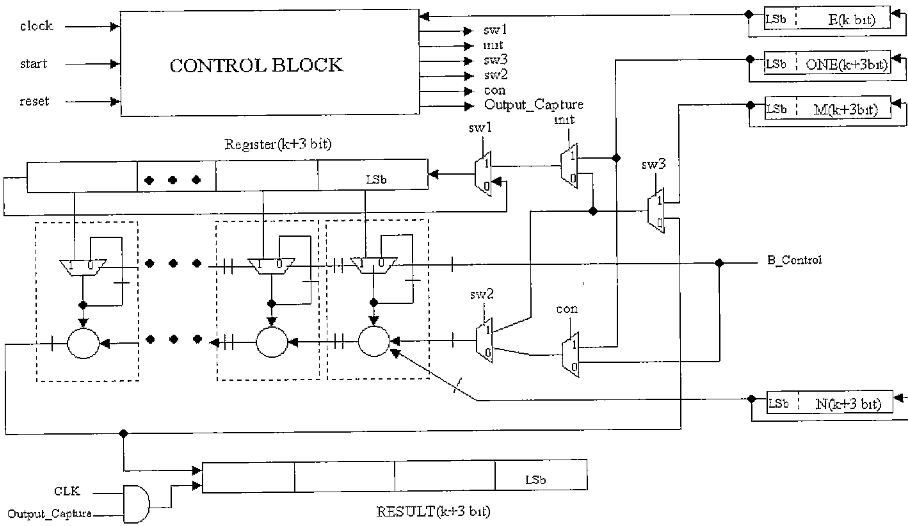


그림 6 멱승기의 전체 구조

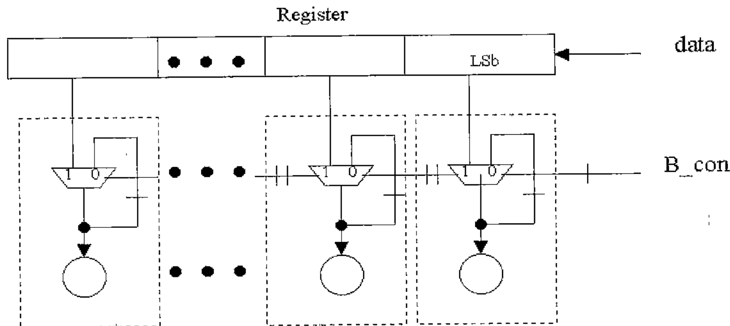


그림 7 serial to parallel 의 구현

값이 시리얼로 나오고 멱승시 입력으로도 시리얼로 들어가게 된다.

그러나, 위에서 들어가는 데이타는 DG에서 보는 것처럼 1,3,5,7,..의 적절한 시간에 각 PE에 도착하여 k+3 클럭 동안 값을 유지하고 있어야 하기 때문에 그림 7에서처럼 쉬프트 레지스터와 멀티플렉서를 사용하여 구현하였다. 또 B_con 입력의 패턴은 q_con과 동일하기 때문에 q_con을 이용함으로써 B_con의 파이프라이닝 플립플롭을 제거할 수 있었다.

4. 결 론

본 논문에서 제시한 1024비트 RSA암호 시스템 멱승기의 하드웨어 리소스, 전체 모듈로 멱승 계산을 위한 클럭 수 및 최장 지연 패스(critical path delay)를 표 2에 보였으며, 아울러 현 시점에서 실제의 칩으로 구현되었다고 보고된 가장 최근의 논문인 Paar의 논문[6]과 비교하였다.

Paar는 FPGA의 CLB(Configurable Logic Block)의 수를 줄이기 위해 PE를 1 비트가 아닌 u=4, 8, 16비트로 매핑하여 k/u개의 PE를 구성하였기 때문에 동일한 비교 조건을 만들기 위해서 하나의 PE를 1비트라고 가정하고 비교하였다. 표 2에서 클럭수는 입력데이타가 들어가기 시작해서 출력 데이타의 첫 비트가 나올때까지의 클럭수를 의미하며, 게이트 카운트는 삼성 스탠다드 셀 라이브러리[8]를 근거로 2입력 NAND 게이트를 1로 하여 계산한 것이다. 하드웨어 리소스에서는 RAM도 레지스터 카운트에 포함시켰는데 Paar논문의 PE외부 구성요소중에 (k+3)비트 레지스터가 10+α인 것은 6개의 레지스터와 4개의 RAM 중에 Dual Port RAM이 2개 있기 때문이다.

Paar의 논문에서 Time*Area Product값이 최적이라고 제안한 u=8비트의 경우에 대해, PE당 1비트라고 가정할 때 약 58개의 게이트를 가지고 있으며, 본 논문은 약90개의 게이트를 가진다.

그러나 PE외부의 구성요소에서 RAM과 Dual Port RAM및 디코더때문에 하드웨어 리소스는 본 논문과 거의 비슷함을 알 수 있다. 클럭의 주기를 결정하는 최장 지연 패스에 있어서는 삼성 라이브러리를 근거로 계산한 결과, Paar의 논문은 약 19ns의 지연을 가지고 본 논문은 약 8.5ns의 지연을 가진다. 이는 본 논문에서 제안한 시스템이 약 2배 이상의 빠른 클럭 주파수에서 동작할 수 있음을 보여주며, 이러한 차이는 Paar논문의 u 비트 덧셈기의 캐리 지연 때문이다. 데이타 처리 속도(data rate)로 두가지 경우의 성능을 표현하면, Paar는 50Mhz의 클럭 주파수에서 약 25.6kbps의 속도를 보이고 본 논문은 120Mhz의 주파수에서 약 56.8kbps의 성능을 가진다.

본 논문에서는 수평 매핑을 통하여 본문에서 언급한 수직 매핑의 단점을 피하고 성능을 향상시키는 방법을 제시하였다. 제안된 하드웨어 구조는 수평으로 매핑함에 따라서 파이프라이닝 플립플롭의 수가 증가하는 단점이 있지만, 표 3에서 보는 것처럼 제곱과 곱셈의 입력이 단순하게 들어가고 결과 역시 순차적으로 나오기 때문에 100%의 처리율을 쉽게 이룰 수 있었다. 또한, 컨트롤이 더 쉬워지고, 그에 따른 PE의 내부 구조도 더 간단해 졌으며, 데이타 및 컨트롤 신호의 지역성으로 인해 더 빠른 클럭 속도로써 성능을 향상시킬 수 있었다.

표 2 Paar와 성능 비교

	Paar		본 논 문	
클럭수	$2k^2 + 12k + 16$		$2k^2 + 12k + 19$	
하 드 웨 어 리소스	각 PE당	Gates	각 PE당	Gates
	MUX(4:1) 1개 MUX(3:1) 1개 MUX(2:1) 2개 Full Adder 1개 3x8 Decoder 1개 Flip Flop 8개	5.5 4.5 5 5.5 9.8/u 36	AND Gate 1개 MUX(2:1) 4개 Full Adder 2개 Flip Flop 15개	1.3 10 11 67.5
	PE외부 구성요소 (k+3)bit Register 10+α개 State Machine 1개		PE외부 구성요소 (k+3)bit Register 6개 State Machine 1개	
최장지연패스	MUX(4:1) + u-bit Full Adder		Full Adder 2개 + MUX(2:1) 3개	

표 3 Paar와 데이타 입출력 플로우 비교

	데이터 입력*	결과 출력**
본논문	○○○○…××××	$S_0\ S_1 \cdots S_k\ M_0\ M_1 \cdots M_k$
Paar	○×○×…○×○×	$S_0\ S_1 \cdots S_k$ $M_0\ M_1 \cdots M_k$

* O : 제곱을 위한 입력
X : 곱셈을 위한 입력
** Si : 제곱 결과의 i번째 비트
Mi : 곱셈 결과의i번째 비트

본 논문에서는 각 비트당 하나의 PE로 구현하였으며, 클럭 사이클 타임(clock cycle time)을 결정하는 요소인 최장 지연 패스가 두개의 전가산기와 세개의 멀티플렉서로 이루어져 있어 100 MHz이상의 클럭까지 구현이

가능하다. 위에서 보인 성능 지표는 0.5um기술인[8]을 근거로 하였지만 이에 0.25 내지는 0.18um기술을 적용한다면 속도나 크기에 있어서 좀더 향상 효과를 볼 수 있다. 제안된 하드웨어 구조를 동일한 설계 방법을 사용하여 하이 래딕스로 구현하면 PE당 3비트 혹은 4비트(r=8 or 16)까지 가능하여 클럭 사이클 수를 현저히 줄일 수 있어 더 한층 성능을 향상(약 3배~4배)시킬 수 있다. 속도를 수 Mbps까지 올리기 위해서 하이 래딕스를 사용하여 ASIC으로 구현하는 중이며, 아울러 스마트 카드에의 적용을 위해 low frequency, low power설계를 위한 연구가 진행중이다.

참 고 문 헌

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

[2] Cetin Kaya Koc, "RSA Hardware Implementation," *RSA Laboratories*, August 1995.

[3] P. Montgomery, "Modular multiplication without trial division," *Mathmatics of computation*, vol. 44, pp.519-521, 1985.

[4] Y.Jeong, W.Burleson, " VLSI array algorithms and architectures for RSA modular multiplication," *IEEE Tran. On VLSI Systems*, vol.5, pp.211-217, June 1997.

[5] Colin Walter, "Systolic modular multiplication," *IEEE Tans. On Computers*, vol.42, March 1993.

[6] Thomas Blum, Christof Paar, "Montgomery Modular Exponentiation on Reconfigurable Hardware," *IEEE Symposium on Computer Arithmetic*, April 14-16, 1999, Adelaide, Australia.

[7] M. Shand, J. Vuillemin, "Fast Implementations of RSA Cryptography," *Proceedings 11th IEEE Symposium on Computer Arithmetic*, pp.252-259, 1993.

[8] Samsung Electronics, "ASIC STD85/STDM85 0.5um High Density CMOS Standard Cell Library," September 1997.

이 석 용

2000년 2월 광운대학교 전자공학부 졸업. 2000년 3월 ~ 현재 광운대학교 전자통신공학과 석사과정. 관심분야는 통신용 칩 설계, 무선 통신, 정보보호

김 성 두

2000년 2월 서울산업대학교 전자공학과 졸업. 2000년 3월 ~ 현재 광운대학교 전자통신공학과 석사과정. 관심분야는 통신용 칩 설계, 무선 통신, 정보보호

정 용 진

1983년 2월 서울대학교 제어계측공학과 졸업. 1983년 3월 ~ 1989년 7월 전자통신 연구소(ETRI). 1991년 5월 미국 UMASS 전자전산공학과 석사. 1995년 2월 미국 UMASS 전자전산공학과 박사. 1995년 4월 ~ 1999년 2월 삼성전자 반도체 수석 연구원. 1999년 3월 ~ 현재 광운대학교 전자공학부 조교수. 관심분야는 컴퓨터 연산 알고리즘, ASIC 설계, 무선 통신, 정보보호