RSA 암호 시스템의 고속 처리를 위한 새로운 모듈로 연산 알고리즘 및 하드웨어 구조

정 용 진
광운대학교 전자공학부, 서울 노원구 월계동 447-1

A New Modular Arithmetic Algorithm and its Hardware Structure for RSA Cryptography System

Yong-Jin Jeong (yjjeong@daisy kwangwoon ac kr)
School of Electronics Engineering, Kwangwoon University, Seoul, Korea

요 약

본 논문에서는 RSA 암호 알고리즘의 핵심 개산 과정인 모듈로 곱셈 연산의 효율적인 하드웨어 구현을 위해 새로운 알고리즘과 하드웨어 구조를 제시한다. 기존의 몽고메리 알고리즘이 LSB 우선 방법을 사용한 것과는 달리 여기서는 MSB 우선 방법을 사용하였으며, RSA 암호 시스템에서 키가 일정 기간 동안 변하지 않고 유지된다는 점에 착안해 계수(Modulus)에 대한 보수(Complements)를 미리 계산해 놓고 이를 이용하여 모듈로 감소 처리를 간단히 덧셈으로 치환하도록 하였다. 보수들을 저장할 몇 개의 레지스터와 그들 중 하나를 선택하기 위한 간단한 멀티플렉서(Multiplexer)만을 추가함으로써 몽고메리 알고리즘이 안고 있는 홀수 계수 조건과 사후 연산이라는 번거로움을 없앨 수 있다 본 논문에서 제안하는 알고리즘은 하드웨어 복잡도가 몽고메리 알고리즘과 비슷하며 그 내부 계산 구조를 보여주는 DG(Dependence Graph)의 지역 연결성 (Local Connection), 모듈성(Modularity), 데이터의 규칙적 종속성 (Regular Data Dependency) 등으로 인해 실시간 고속 처리를 위한 VLSI 구현에 적합하다.

1. 개요

고속 인터넷의 확장과 전자 상거래의 실시에 있어 정보의 암호화(Encryption)와 전자서명 및 인증(Electronic Sign and Authentication)은 필수 요구 사항이다 공개키 방식 암호 시스템은 이를 위한 기본 알고리즘을 갖추고 있으며 이의 대표적인 것이 RSA 알고리즘이다. RSA 알고리즘은 비트 수가 1024 혹은 2048 비트나 되는 큰 정수를 기반으로 한 모듈로 연산에 의해 수행된다. RSA를 위한 모듈로 연산은 내부에 곱셈과 나눗셈이 복합되어 있어 계산 구조가 복잡하고, 워드 사이즈가 크기 때문에 고속 실시간 처리를 위한 하드웨어 모듈의 구현이 매우 어렵다. 그러나, 근래 들어서는 VLSI 설계 기술의 발달과 함께 다양한 연산 알고리즘들이 연구되어 고속 RSA 모듈 구현에 대한 연구가 활발해지고 있다.[2][3][4][5][6][7][8] 하드웨어로 구현했을 경우의 또 하나의 장점은 소프트웨어보다 키의 안전성 문제에서 월등하다는 점이다 본 논문에서는 고속 RSA 모듈 구현을 위한 새로운 모듈로 연산 알고리즘과 이의 VLSI 구현을 염두에 둔 하드웨어 구조를 제안한다

2. RSA 알고리즘과 모듈로 연산

RSA 알고리즘의 용도는 크게 암호화(Encryption and Decryption) 기능과 인증(Authentication) 기능으로 구분할 수 있지만 내부 연산 구조는 동일하다. RSA 연산에 사용되는 계수 n 은 두 개의 큰 소수의 곱으로 만들어지며, 연산 과정은 메시지 M 을 $[0,n-1]$ 의 정수로 변환한 다음 공개키 e 니 비밀키 d 에 대해 모듈로 멱승 $C = M^e \mod n$ (또는 $M = C^d \mod n$)을 취하는 것이다.[9][10] 그러므로 RSA 알고리즘의 기본 연산은 모듈로 멱승 계산(Modular Exponentiation)이며, 이것은 다시 모듈로 곱셈 (Modular Multiplication)의 반복에 의해 계산되므로 RSA 구현의 핵심은 모듈로 곱셈기의 설계라 할 수 있다. 이를 위해 지금까지 수많은 알고리즘들이 연구 발표되었으며, 현재는 대부분 1985 년 발표된 몽고메리(Montgomery) 알고리즘을 나름대로 변형하여 사용하고 있다. [1][3]

모듈로 곱셈은 모듈로 덧셈의 연속이며 계산해 가는 순서에 따라 MSB 우선 방법과 LSB 우선 방법으로 나눌 수 있다. 몽고메리 알고리즘은 LSB 우선 방법에 속하며, 이진 숫자일 경우 각 반복 단계마다 중간 결과를 2 의 배수, 즉 LSB 를 0 으로 만들어 소거함으로써 결과가 워드 크기 k 를 넘지 않도록 한다. 이 때문에 계수가 항상 홀수여야 하는 조건이 생기며, 올바른 결과를 위해서는 최종 결과에 워드 크기만큼 2 를 곱하여 다시 모듈로 감소를 취해 주어야 하는 사후 처리가 필요하다. RSA 의 경우에는 계수가 두 소수의 곱으로 항상 홀수이며, 후자의 제한 조건도 미리 입력 변수에 보상을 해 놓고 곱셈을 하여 해결할 수 있기 때문에 위의 두 가지 조건이 큰 문제가 되지는 않는다.[3] 그러나, 원하는 연산이 한 번의 모듈로 곱셈일 경우와 계수의 제한조건 없이 일반적으로 사용하고자 할 경우에는 앞뒤로 처리해 주어야 하는 과정이 큰 부담으로 남는 것은 당연하다 참고로 짝수인 계수에 대해서 몽고메리 알고리즘을 사용하고자 할 경우에는 계수를 인수 분해하여 CRT (Chinese Remainder Theorem)를 이용하면 되지만 여기에는 많은 하드웨어 오버헤드가 뒤따른다 그럼에도 불구하고, 몽고메리 알고리즘의 가장 큰 장점은 LSB 위치에서 모듈로 감소 처리를 위한 결정이 이루어지기 때문에 캐리의 전달

로 인한 지연 시간이 필요 없다는 점이며 이것이 하드웨어 구현 측면에서 큰 이득으로 작용한다

본 논문에서 제안하는 알고리즘은 위와는 달리 MSB 우선 방법을 취하고 있다. 기본 아이디어는 중간 계산결과 중 워드 사이즈, 즉, k 를 벗어나는 항에 대해 미리 계산해 놓은 보수들을 이용하여 필요한 때에 모듈로 감소를 행함으로써 중간 결과를 워드 범위 안에 유지시킨다는 것이다 그리므로, 계수에 대한 제한 조건이나 사후 연산이 필요 없으며, 대신에 보수들을 저장할 메모리와 그 중 하나를 선택하기 위한 멀티플렉서가 필요하다 이러한 접근 방법은 문헌 [2]에서 시도되었지만 몽고메리 알고리즘에 비해 하드웨어 복잡도가 커 실제 사용에는 한계가 있었다. 본 논문의 알고리즘은 [2]의 알고리즘을 보완 발전시킨 것으로 하드웨어 자원 및 계산 시간 측면에서 몽고메리 알고리즘과 대등한 결과를 보인다.

3. 새로운 알고리즘 제안

모듈로 곱셈, $U=AB \bmod n$, 의 입력 변수 A, B, n 의 워드 사이즈를 k-비트라고 하고, 각 변수의 비트 정보를 a_i, b_i, n_i ($\in \{0,1\}$)로 표시하면 호너의 규칙(Horner's Rule)을 적용하여 식 (1)과 같이 반복 수식(Iterated equation) 으로 알고리즘을 표현할 수 있다.

$$\begin{aligned}&\text{(i)}\quad P_{(0)} = 0\\ &\text{(ii)}\quad P_{(j+1)} = 2P_{(j)} + b_{k-j}A \bmod n\\ &\text{(iii)}\quad U = P_{(k)} \bmod n\end{aligned} \tag{1}$$

앞에서 언급한 것처럼 모듈로 처리를 효율적으로 덧셈으로 대체하기 위해 계수의 보수(complements of the modulus) D_h를 식 (2)와 같이 정의한다[2]. 즉, 모듈로 곱셈 과정에서 나타나는 중간 결과들에 대해 워드 사이즈 k 를 초과해 나타나는 2^k 항 대신 그 weight 에 따라 미리 계산해 놓은 D_h 중 하나를 더함으로써 모듈로 감소 처리를 덧셈으로 대치하기 위한 것이다 보수 D_i 은 계수 n 의 2's complement 를 구하는 방법과 동일하다

$$D_h \equiv h \cdot D \bmod n, \text{ where } D \equiv 2^k \bmod n \tag{2}$$

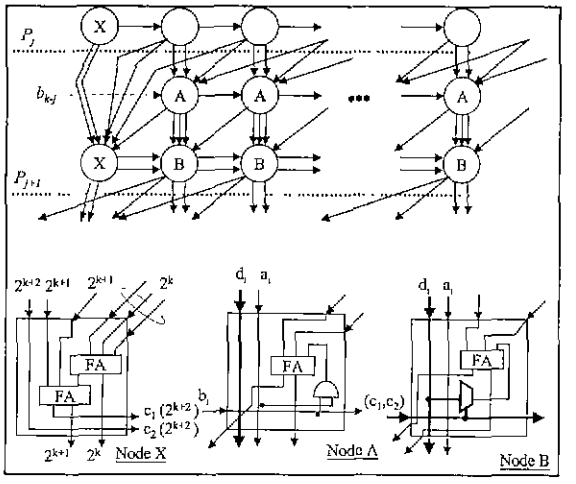
식 (1)을 살펴 보면 모듈로 연산이 하나의 덧셈 변수가 되기 때문에 (ii) 부분에서 더해야 할 변수가 3 개임을 알 수 있다 이것은 캐리를 고려하면 한 개의 Full Adder로 구현이 어려움을 나타내기 때문에 (ii) 부분을 분리하여 식 (3)으로 변형하였다 이 방법은 모듈로 감소를 위한 결정이 LSB 위치에서 일어나는 몽고메리 알고리즘과는 달리 더할 값에 대한 결정이 MSB 비트 위치에서 일어나기 때문에 하드웨어 구현시 치명적인 약점이 된다. 이것을 해결하기 위해서는 캐리의 전달 지연을 피할 수 있도록 Carry Save Adder (CSA) 를 사용하면 된다. 알고리즘 (3)은 [2]에 나타난 것보다 하드웨어 측면에서 한층 향상된 것이라 할 수 있지만 아직도 많은 개선의 여지를 안고 있다 즉, CSA 로 식 (3)을 구현했을 경우 매 반복 단계마다 일어날 수 있는 h 의 최대 범위가 5 가 되어 D_1, D_2, D_3, D_4, D_5 의 사전 계산은

$$\begin{aligned}&\text{(ii)}'\ P^*_{(j+1)} = 2P_{(j)} + b_{k-j}A\\ &\quad \text{if } P^*_{(j+1)} \geq 2^k,\ h = P^*_{(j+1)} \bmod 2^k\\ &\qquad P_{(j+1)} = P^*_{(j+1)} - h \cdot 2^k + D_h\\ &\quad \text{if } P^*_{(j+1)} < 2^k,\ P_{(j+1)} = P^*_{(j+1)}\end{aligned} \tag{3}$$

$$\begin{aligned}&\text{(i)}\quad P_{(0)} = 0,\ h_0 = 0\\ &\text{(ii)}\quad P^*_{(j+1)} = 2P_{(j)} + b_{k-j}A\\ &\qquad h_{j+1} = P^*_{(j+1)} \bmod 2^k\\ &\qquad \text{if } h_{j-1} == K, \text{ then}\\ &\qquad\qquad P_{(j+1)} = P^*_{(j+1)} - K \cdot 2^k + D_K\\ &\text{(iii)}\quad U = P_{(k)} \bmod n\end{aligned} \tag{4}$$

물론 각 비트 위치에서 h 값에 따라 이들을 선택하기 위한 6:1 Mux 가 필요하다는 것이다

위의 알고리즘을 보완 향상시킨 알고리즘이 식 (4)에 나타나 있다. 여기서의 특징은 매 단계마다 해당 h 값에 따라 D_h 를 선택하여 더하는 것이 아니라 h 가 어느 값 K 에 도달할 때까지 기다렸다가 덧셈으로 치환하기 때문에 D_K 만 있으면 되고 그에 따르는 멀티플렉서의 크기 (2 1 Mux)도 훨씬 작아지게 된다. 그러나, K를 계산하기 위해 카운터를 사용할 경우 h를 계산하는 컨트롤 모듈의 하드웨어가 커져 전체적인 클럭 스피드를 낮추는 요인이 된다. 앞의 두 가지 요인을 모두 고려하여 본 논문에서는 두 개의 컨트롤 신호를 이용해 K 를 나타내었다 즉, K 의 값을 하나로 고정하지 않고 4,8 중 어느 값을 갖도록 하여 보상을 한 다음 K를 뺀 h 값의 나머지 부분은 다음



(그림 1) 새로운 모듈로 연산 알고리즘의 내부 계산 구조

반복 단계로 넘기는 것이다. 그림 1 에 알고리즘 (4)의 내부 데이터 흐름을 나타내는 DG (Dependence Graph) 와 각 노드 기능이 나타나 있다.

4. 결론

본 논문에서는 RSA 암호 알고리즘 구현에 필요한 모듈로 연산 알고리즘을 제안하고 고속 하드웨어 모듈 설계를 위한 내부 구조를 보였다 제안된 알고리즘은 기존의 몽고메리 알고리즘이 가지고 있는 계수 제약 조건과 사후 처리 과정을 제거하면서 내부 하드웨어 복잡도는 거의 대등하기 때문에 앞으로 활용 가능성이 높다. 본 알고리즘의 특징으로는 D_k 의 사전 계산이 필요하고 계산 결과가 CSA 형태로 나온다는 점이다 그러나, 사전 계산은 RSA 시스템의 키가 바뀔 때에만 한 번 해주면 되기 때문에 전혀 문제가 되지 않는다.

RSA 시스템의 특성상 워드 사이즈가 1024 비트 혹은 2048 비트로 방대하기 때문에 본 논문에서 보인 어레이 구조를 그대로 하나의 칩에 구현하는 것은 무리가 따르며 파이프라인 구조를 갖는 일차원 어레이 형태로 매핑 (Mapping) 하는 과정이 필요하다. 이 과정은 제안된 알고리즘의 내부 연산 구조가 규칙적이고 모듈화되어 있기 때문에 쉽게 얻을 수 있어 본 논문에서는 생략하였다. 현재 상술한 특징들과 매핑 과정을 통하여 VHDL 시뮬레이션을 마치고 FPGA 설계 진행 중에 있으며 앞으로 PC 인터페이스와 컨트롤을 추가하여 RSA 보드를 설계할 예정이다. 본 논문에서 제안한 알고리즘의 구현을 위해 필요한 게이트 수는 설계 스타일에 따라 큰 차이가 있지만 ASIC 으로 구현시 (1024-bit RSA 시스템) 대략 6 만~7 만 게이트로 예상된다.

아울러, FPGA 혹은 ASIC 구현을 위해 추가적으로 이루어져야 할 최적화 작업으로는 1-차원 어레이 구현을 위한 최적의 워드 분할 (partitioning) 및 매핑 과정, 모듈로 멱승 계산을 위한 곱셈과 제곱의 interleaving scheme, 멱지수 (exponent) 의 bit-pattern 활용 방안, 그리고 더 큰 워드 사이즈로의 확장성 (extendability)에 대한 고려가 필요하다 RSA 시스템에 사용하기 위해서는 암호화 및 복호화에 사용되는 멱지수, 즉 공개키 및 비밀키의 특징을 최대한 활용하는 것도 하드웨어의 복잡도를 더 줄이는데 도움이 될 것이다.

참고문헌

[1] P. Montgomery, *modular multiplication without trial division*, Mathematics of Computation, vol.44, pp.519-521, 1985.

[2] Yong-Jin Jeong and Wayne Burleson, "VLSI Array Algorithms and Architectures for RSA Modular Multiplication," *IEEE Trans. on VLSI Systems*, vol.5, pp 211-217, June 1997

[3] Cetin Kaya Koc, "High-Speed RSA Implementation," RSA Data Security Inc. *Technical Report RSA_TR201, November 1994*

[4] Colin Walter, "Systolic Modular Multiplication," IEEE Trans On Computers, vol 42, no.3, March 1993.

[5] P Konerup, "A *Systolic Linear-array multiplier for a class of Right-Shift* algorithms," Trans On Computers, vol.43, no 8, pp.892-898, August 1994.

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

[7] H.Orup, "Simplifying quotient determination in high-radix modular multiplication,". *Proceedings of the 12th Symposium on Computer* Arithmetic, IEEE, pp 252-259, 1995.

[8] Thomas Blum and Christof Paar, "Montgomery Modular Exponentiation on Reconfigurable Hardware," Proceedings of the 14th Symposium on Computer Arithmetic (IEEE), 1999.

[9] Douglas Stinson, Cryptography: Theory and Practice, CRC Press, 1995.

[10] Bruce Schneier, Applied Cryptography, John Wiley & Sons Pub , 1996