2000년 한국정보처리학회 추계 학술발표논문집 제7권 제2호

Hybrid 가산기를 이용한 고속 모듈러 곱셈기의 설계*

이재철, 임권묵, 강민섭
안양대학교 정보통신 · 컴퓨터 공학부
e-mail:jclee@cs2.anyang.ac.kr

Design of High Speed Modular Multiplication Using Hybrid Adder

Jae-Chul Lee, Kwon-Mook Lim, Min-Sup Kang
Div. of Computer & Electronic Engineering, Anyang University

요약

본 논문에서는 RSA 암호 시스템의 Montgomery 모듈러 곱셈 알고리듬을 개선한 고속 모듈러 곱셈 알고리듬을 제안하고, Hybrid 구조의 가산기를 사용한 고속 모듈러 곱셈 알고리듬의 설계에 관하여 기술한다. 기존 Montgomery 알고리듬에서는 부분합계산시 2번의 덧셈연산이 요구되지만 제안된 방법에서는 단지 1번의 덧셈 연산으로 부분 합을 계산할 수 있다. 또한 덧셈 연산 속도를 향상시키기 위하여 Hybrid 구조의 가산기를 제안한다. Hybrid 가산기는 기존의 CLA(Carry Look-ahad Adder)와 CSA(Carry Select Adder)알고리듬을 혼합한 구조를 기본으로 하고 있다. 제안된 고속 모듈러 곱셈기는 VHDL(VHSIC Hardware Description Language)을 이용하여 모델링하였고, Synopsys™사의 Design Analyzer를 이용하여 논리합성(Altera 10K lib. 이용)을 수행하였다. 성능 분석을 위하여 Altera MAX+ PLUS II 상에서 타이밍 시뮬레이션을 수행하였고, 실험을 통하여 제안한 방법의 효율성을 입증하였다.

1. 서론

최근 통신 기술의 발전으로 컴퓨터 통신망을 통한 정보 교환이 활발히 이루어지면서 컴퓨터 통신망의 불법적인 접근을 통한 정보 노출이 용이하기 때문에 암호 알고리듬에 대한 연구가 활발히 진행 중에 있다[1,2,4]. 암호 알고리듬은 키 사용 방법에 따라 공통키(common key encryption) 방식과 공개키(public key encryption) 방법으로 나눌 수 있다[2]. 1976년에 Diffie와 Hellman에 의해 공개키 알고리듬[3]이 최초로 제안되었고, 그후 공개키 개념을 실현하기 위한 다양한 알고리듬이 발표되었지만, 현재 안정성을 인정받고 있는 대표적인 알고리듬은 RSA와 Diffie-Hellman 알고리듬이다. RSA 알고리듬 ($M = C^d \bmod N$)은 모듈러 N에 대한 소인수 분해의 어려움에 그 안정성을 두고 있으며, 공개하는 키 중 하나인 모듈러 N 정수의 소인수 분해는 NP-문제로 알려져 있다[2].

RSA 암 · 복호 알고리듬은 모듈러 멱승 연산을 기본 연산으로 사용하고 있고, 이 모듈러 멱승 연산은 모듈러 곱셈 연산을 원자적 연산으로 하여 수행된다. 기존의 RSA 알고리듬을 이용할 경우 사용하는 킷값이 증가하면 모듈러 멱승을 위한 연산 속도가 감소되므로 고속연산을 위해서는 멱승에 필요한 모듈러 곱셈 수를 가능한 한 최소한으로 줄여야 한다. 따라서 공개키 암호 알고리듬의 실시간 처리를 위해서는 알고리듬 레벨에서의 고속화 기법 연구와 더불어 그 알고리듬의 구현 방법에 대한 연구도 매우 중요하다[5-7].

본 논문에서는 RSA 암 · 복호 알고리듬의 핵심 부분인 고속 모듈러 곱셈 알고리듬을 제안하고, 연산 속도 향상을 위해 Hybrid 구조의 가산기를 사용한 고속 모듈러 곱셈 기의 설계 및 구현에 관하여 기술한다. 제안한 고속 모듈러 곱셈기는 기존의

*이 논문은 IDEC의 지원을 받아 연구되었음.

Montgomery 알고리듬[11]을 기본으로 하고 있으나, 부분 합의 계산 방법을 개선하였으며, 덧셈 연산 속도를 향상을 위해 Hybrid 구조의 고속 가산기를 사용하였다. 제안된 고속 모듈러 곱셈기는 VHDL(VHSIC Hardware Description Language)을 이용하여 설계하였고, Synopsys™사의 Design Analyzer를 이용하여 논리합성(Altera 10K 라이브러리 이용)을 수행하였다. FPGA 구현을 위하여 Altera MAX+ PLUS II 상에서 타이밍 시뮬레이션을 수행하였다.

2. Montgomery 알고리듬

고속 RSA 암호 시스템을 구현하기 위해서는 커다란 승수를 갖는 모듈러 곱셈 연산에 대한 고속 연산 알고리듬의 설계가 매우 중요한 요소로 작용한다. 가장 널리 사용되는 modular 감소 알고리듬의 하나인 Montgomery 알고리듬[11]은 수 체계의 변환을 통해 modular 감소가 쉽게 되는 수 체계에서 연산한 후 원래의 수체계로 역변환시키는 방법을 이용한다. 즉, modular 감소가 쉬운 수 $R = 2^n$를 이용하여 modular multiplication을 비교적 빠르게 할 수 있는 알고리듬이다. 일반적으로 Montgomery 알고리듬은 RSA 암·복호 알고리듬의 H/W구현에 가장 널리 사용되는 알고리듬으로 알려져 있다[6, 7].

Montgomery 알고리듬은 $MonPro(A, B) = A \cdot B \cdot r^{-1} \bmod N$을 계산한다. 여기서 N은 솟수이며, $0 \le A$ and $B < N$ 이라 가정하고, r은 N과 서로 소인 N 보다 큰 수이다. 그리고 $A = \sum_{j=0}^{n-1} a_j \times 2^j$ 라고 하면 Montgomery 알고리듬을 이용하여 부분합인 S[0], S[1], …, S[n]을 구할 수 있다. (그림 1)에 보인 함수 MonPro()는 C. D. Water에 의해서 H/W 구현에 적합하도록 변형된 Montgomery 알고리듬을 나타낸다.

```
MonPro(A, B, N)
  S[0] := 0;
   for i in 0 to n-1
    q := (Si + Ai×B) mod 2;
    S[i+1] := (S[i] + Ai×B + q×N) div 2;
  return S[i+1];
```

(그림 1) Montgomery 알고리듬

(그림 1)의 알고리듬에서 S[0]= 0으로 초기화되며, for loop은 i의 값에 의해서 제어된다. 이러한 결과는 "$2^nS=A \cdot B + M \cdot N$" 을 만족 시키며 "$S=A \cdot B \cdot 2^{-n} \bmod N$" 이 된다. 이때 S는 "S<2N" 이 되므로 출력값은 S+N 또는 S가 된다.

3. 고속 모듈러 곱셈 알고리듬

(그림 1)에 나타난 기존의 알고리듬[7]은 킷값이 커지면 커질수록 반복문 내에서 처리를 위해 커다란 연산량이 요구되므로 실시간 처리에 문제점을 가지고 있다. 이러한 문제점을 해결하기 위해서는 가장 커다란 연산량이 되는 덧셈 연산의 횟수를 감소시켜야 한다. (그림 2)에 나타난 함수 AM()은 기존의 Montgomery 알고리듬을 개선한 제안된 고속 모듈러 곱셈 알고리듬을 나타낸다.

```
AM(A, B, N)
   S[0] := 0;
   BN := B + N;
   for i in 0 to n-1
       q := (S[i] + Ai×B) mod 2;
       case(Ai & q) of
           "11" : opd := BN;
           "10" : opd := B;
           "01" : opd := N;
           "00" : opd := 0;
        end
        S[i+1] := (S[i] + opd) div 2;
   return S[i+1];
```

(그림 2) 제안된 고속 모듈러 곱셈 알고리듬

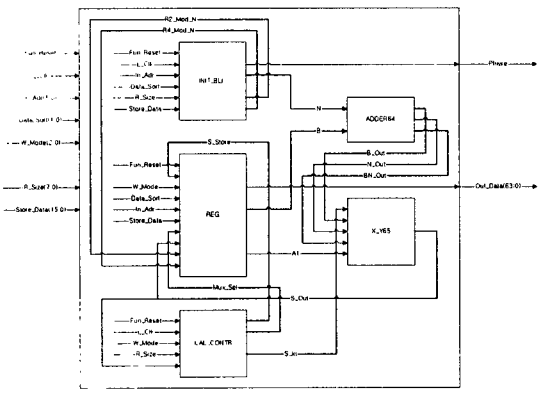
기존 알고리듬(그림 1)에 있어서 S[i+1]의 연산은 for 루프에서 두 번의 덧셈 연산이 요구된다. 그러나 제안된 알고리듬(그림 2)은 고정된 값으로 A_i와 q에 의해서 덧셈 연산의 operand로 사용되어지기 때문에 단지 한번의 덧셈 연산만이 필요하다. 즉, 입력 값인 B, N과 0 값은 특별한 연산 및 기억 공간을 필요로 하지 않고, 단지 for 루프 문에 들어가기 전에 B+N 값을 한번 계산하여 특정의 기억 공간에 저장시키고, A_i와 q에 의해서 선택된 값으로 S[i]와 덧셈 연산을 수행하면 된다. 결과적으로 단지 한 번의 덧셈 연산이 요구되므로 모듈러 곱셈의 고속화가 가능하게 된다.

4. 고속 모듈러 곱셈기의 하드웨어 설계

(그림 3)은 제안된 고속 모듈러 곱셈기의 top level 블록도를 나타낸다. 제안된 고속 모듈러 곱셈기는 (그림 3)에서 BN값을 연산(그림 2)하는 ADDER64와 systolic array부분인 X_Y65, 초기화와

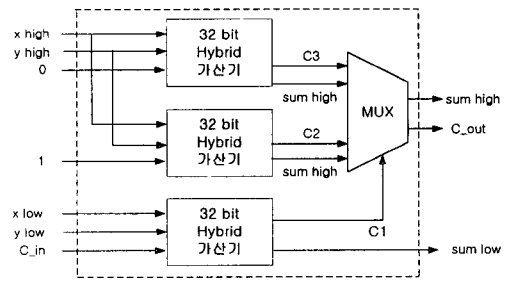
뺄셈 연산을 담당하는 INIT_BLK, 연산의 제어를 담당하는 CAL_CONTR, 그리고 레지스터 총 5개의 블록으로 구성된다.

제안된 고속 모듈러 곱셈기는 H/W area 문제를 해결하기 위하여 외부 클럭에 맞추어 출력 값이 순회적으로 입력되어지는 계산 형태를 취하고 있다.



(그림 3) 고속 모듈러 곱셈기의 블록도

덧셈 연산의 성능 향상을 위해 ADDER64와 INIT_BLK에서 사용된 64비트 Hybrid 가산기의 구조는 (그림 4)와 같다. 3개의 CLA와 MUX로 구성되며, 최하위의 기본적인 셀(Cell)은 4-비트 CLA로 구성되어 있다.

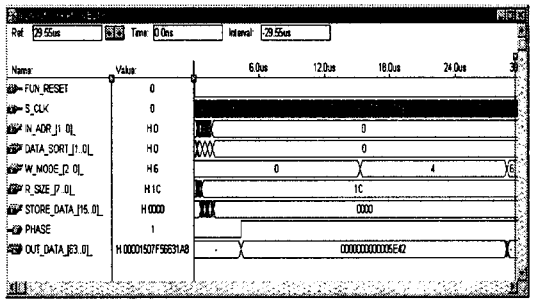


(그림 4) 64비트 Hybrid 가산기의 구조

5. 구현 결과 및 성능 분석

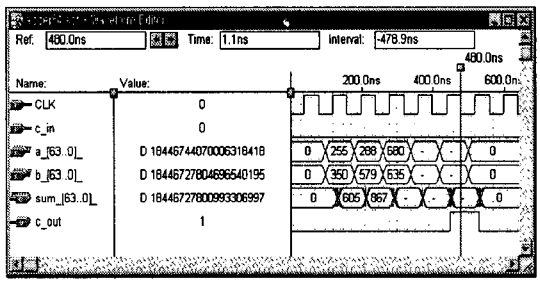
본 논문에서 제안된 고속 모듈러 곱셈기는 크게 5개의 기능 모듈로 구성되어 있으며 Axil-320 워크스테이션 상에서 VHDL(VHSIC Hardware Description Language)을 이용하여 설계하였다. 제안된 고속 모듈러 곱셈기의 Front-end 설계는 Synopsys^{TM}사의 Design Analyzer를 이용하여 시뮬레이션 및 논리 합성을 수행하였고, 논리 합성 시 Altera 10K 라이브러리를 이용하였다. FPGA 구현을 위하여 Altera MAX+ PLUS II 상에서 타이밍 시뮬레이션을 수행하였다.

(그림 5)는 제안된 고속 모듈러 곱셈기의 타이밍 시뮬레이션 결과를 나타낸다.



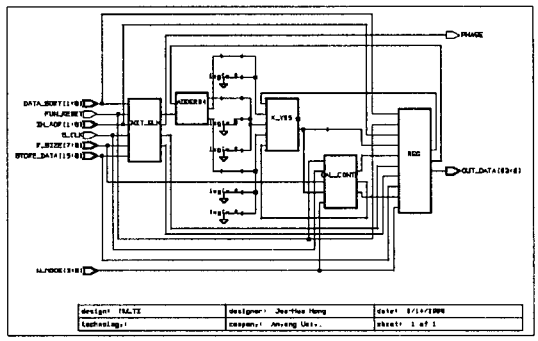
(그림 5) 모듈러 곱셈기의 타이밍 시뮬레이션 결과

(그림 6)은 제안된 Hybrid 가산기의 타이밍 시뮬레이션 결과를 나타낸다.



(그림 6) Hybrid 가산기의 타이밍 시뮬레이션 결과

(그림 7)은 논리 합성을 수행한 고속 모듈러 곱셈기의 top level 블록도를 나타낸다.



(그림 7) 모듈러 곱셈기의 top level 블록도

제안된 시스템에서 모듈러 곱셈연산을 위한 블록은 64비트 단위로 처리되며, 이러한 블록들이 체인

으로 구성되어 있다.

<표 1>은 64비트 모듈러 곱셈 연산에 대해 기존의 방법과 제안된 방법과의 성능 분석 결과를 나타낸다. 분석 항목에서 Used area는 Synopsys™사의 Design analyzer를 이용하여 분석한 결과이며, Critical path는 Altera MAX+ PLUS II 상에서 Timing analyzer를 이용한 결과를 나타낸다.

<표 1> 모듈러 곱셈 연산의 성능분석

Algorithms / Items	C. D. Walter[7]	Proposed method	Remark
Used Area (Cell)	592	304	48.6 %
Max delay (ns)	318.5	266.5	16.3 %

성능 분석 결과에서 Max delay인 임계 경로(critical path)는 약 16.3 % 정도 개선되었고, 칩 면적은 약 48.6 %가 감소되었다.

6. 결 론

본 논문에서는 기존의 Montgomery 알고리듬을 개선한 고속 모듈러 곱셈 알고리듬을 제안하고, 연산 속도의 향상을 위해 Hybrid 구조의 가산기를 사용한 고속 모듈러 곱셈 알고리듬의 설계에 관하여 기술하였다. Hybrid 가산기는 기존의 CLA(Carry Look-ahad Adder)와 CSA(Carry Select Adder)알고리듬을 혼합한 구조를 사용하였다.

기존 Montgomery 알고리듬에서는 부분 합 계산 시 2번의 덧셈연산이 요구된다. 그러나 제안된 방법에서는 단지 1번의 덧셈 연산으로 부분합을 계산할 수 있으며, 덧셈 연산시 Hybrid 구조의 가산기를 사용하여 덧셈 연산 속도를 향상하였다.

제안된 고속 모듈러 곱셈기는 Synopsys™사의 Design Analyzer를 이용하여 논리합성을 수행하였으며, Altera MAX+ PLUS II 상에서 Timing analyzer를 이용하여 시뮬레이션을 수행하였다. 성능 분석 결과 임계경로(critical path)는 약 16.3 % 정도 개선되었고, 칩 면적은 약 48.6 %가 감소되었다.

참 고 문 헌

[1] NBS, *Data Encryption Standard,* FIPS Pub, 46, U.S, National Bureau of Standard, Washington DC, Jan. 1977.

[2] R. L. Rivest, A. Shrmir, and L. Adleman, " A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Communications of the ACM*, Vol. 21, No. 2, pp. 120-126, Feb. 1978.

[3] A. Sorkin, "LUCIFER, A Cryptographic Algorithm," *Cryptologia*, Vol. 8, No. 1, pp. 22-24, 1973.

[4] ANSI, "Data Encryption Algorithm," American National Standard X3, 92. NY. 1981.

[5] J. Sauerbrey, "A Modular Exponentiation Unit Based on Systolic Arrays," *Proc. of AUSCRYPT'92*, pp. 12.19-12.24, 1992.

[6] K. Iwamura, T. Matsumoto, and H. Imai, "Systolic Arrays for Modular Exponentiation using Montgomery Method" *Proc. of Euro CRYPT'92*, pp. 477-481, 1992.

[7] C. D. Walter, "Systolic Modular Multiplication," *IEEE Trans. on Computers*, Vol. 42, pp. 376-378, 1993.

[8] 박태규, 황대준, "다중 프로세서를 위한 RSA 병렬 암호화 알고리즘 설계," *한국정보과학회 논문지*, 제22권 제1호, pp. 49-56, 1995.

[9] 강창구, 김대영, "디지틀 다중서명 방식 비교", *한국통신정보보호학회 학회지*, 제2권, 4호, pp. 7-16, 1992.

[10] E. F. Brickell, "A Survey of Hardware Implementation of RSA," *Proc. of CRYPTO89*, pp. 368-370, 1989.

[11] P. L. Montgomery, "Modular Multiplication without Trial Division." *Mathemat. of Computat.*, vol. 44, pp. 519-521, 1985.

[12] 황효선, 임채훈, "공개키 암호 시스템의 고속 구현", *한국통신정보보호학회 종합학술발표회논문집*, Vol. 7, No. 1, pp. 232-247, 1997.

[13] J. Bobs and M. Coster, "Addition Chain Heuristics." *Advances in Cryptology—CRYPTO '89, Proceedings*, Lecture Notes in Computer Science, No. 435, pp. 400-407, New York, NY: Springer-Verlag, 1989.

[14] C. K. Cok, *High-Speed RSA Implementation*, TR 201, November 1994.

[15] 최용락, 소우영, 이재광, 이임영, *통신망 정보보호*, 그린 출판사, 1996.