Radix-16 Montgomery Modular 곱셈 알고리즘의 FPGA 구현과 전력 소모 비교에 관한 연구

*김판기, 김기영 , 김석윤
숭실대학교 컴퓨터학과
e-mail : kim_panki@msn.com, kky@ic.ssu.ac.kr, ksy@comp.ssu.ac.kr

A Study on FPGA Implementation of Radix-16 Montgomery Modular Multiplication and Comparison of Power Dissipation

*Pan-Ki Kim, Ki-Young Kim, Seok-Yoon Kim
School of Computing
Soongsil University

Abstract

In last several years, the need for the right of privacy and mobile banking has increased. The RSA system is one of the most widely used public key cryptography systems, and its core arithmetic operation is modular multiplication. P. L. Montgomery proposed a very efficient modular multiplication technique that is well suited to hardware implementation.

In this paper, the montgomery modular multiplication algorithms(CIOS, SOS, FIOS) , developed by Cetin Kaya Koc, is presented and implemented using radix-16 and Altera FPGA. Also, we undertake comparisons of power dissipation using Quatrus II PowerPlay Power Analyzer.

I. 서론

정보기술의 발달과 함께 휴대 기기를 이용한 전자 화폐 거래, 은행 거래가 활발해지고 있다. 이러한 발달과 함께 암호화에 관한 관심이 날로 높아져가고 있다. 일반적으로 암호화 알고리즘으로 쓰이는 RSA 같은 알고리즘을 위해서는 지수승의 나머지 계산이 이루어져야 한다. P. L. Montgomery[1]는 이 문제를 해결하는 하드웨어에 적합한 알고리즘을 제시했고 Koc[2]은 이 몽고메리 알고리즘 기술을 기반으로 몇 가지 효율적인 알고리즘을 개발했다. Koc은 이 알고리즘들을 32bit의 워드 크기를 사용하여 소프트웨어로 구현, 비교하였다.

또한, 휴대기기에서 암호화 시스템을 구현할 때 신중히 고려해야 하는 부분 중 하나가 시스템의 전력 소모량이다. 전자 인증 등의 빈도를 고려했을 때 FPGA 상에서 암호화 칩을 재구성하는 것이 많은 경제성을 가지기 때문에, 실제로 휴대폰 등의 모바일 기기에는 FPGA를 주로 장착하고 있다.

본 논문에서는 Koc이 실험하였던 것과 마찬가지로 32-bit의 워드 크기를 사용하여 Radix-16 방식으로 SOS, CIOS, FIOS를 Altera FPGA에서 Combinational Logic으로 구현하였다. 또한, Altera사에서 제공하는 Quatrus II PowerPlay Power Analyzer를 이용하여 알고리즘들의 전력소모를 비교 분석 해보았다.

II. 관련 연구

2.1 RSA 공개키 암호 알고리즘

1976년 Diffie와 Hellman[3]에 의해 공개키 암호방식이 제안된 이후, RSA 암호 알고리즘[4]은 공개키암호 시스템에 가장 많이 사용되는 알고리즘 중 하나가 되었다. 모듈러 값 N과 메시지 M를 사용하며, 키 생성 과정을 통해 생성한 공개키 e와 $M=M^{ed} \bmod N$인 관계를 갖는 개인키d를 가지게 된다.

$$C = M^e \pmod N \quad (1)$$
$$M = C^d \pmod N$$

식 (1)에서 볼 수 있듯이 RSA 암호 알고리즘은 모듈러 멱승 연산이 핵심이다. 일반적인 모듈러 곱셈 연산은 곱셈의 반복적 연산으로 시스템 자원을 많이 필요로 하게 된다. 또한 나눗셈 연산은 회로의 구현에 복잡성을 증가시킨다.

2.2 Montgomery 알고리즘

1985년에 P. L. Montgomery[1]는 모듈로 연산을 나눗셈으로 처리하지 않고 쉬프트연산과 덧셈연산으로 처리하여 일반적인 하드웨어에 알맞은 알고리즘을 개발했다.

몽고메리 알고리즘은 A×B mod N 대신 $A \times B \times R^{-1} \bmod N$ 을 결과로 한다. 여기에는 A, B < N이며, R은 기수 그리고 N과 R은 서로소이여야 한다. 이 조건을 만족하기위해 일반적으로 R을 2의 지수승으로 하고 N을 홀수로 취함으로써 이 조건을 만족하고, 또한 shift 연산만을 통해 일반 모듈러 곱셈보다 간단하게 구현할 수 있다. [3][5][6][7]

MMM(A ,B , N, R)

where $A = \sum_{i=0}^{s-1} A^i \times r^i$, $B = \sum_{i=0}^{s-1} B^i \times r^i$, $r^{s-1} \le N < r^s$

이고, $R = r^s$. $R \times R^{-1} - N \times N' = 1$

```
T = A × B
M = T × N′ mod R
T = ( T + M × N ) / R
if T>=N then return T-N
     else return T
```

알고리즘 1. 몽고메리 알고리즘

2.3 Koc의 몽고메리 알고리즘 분류

Koc은 몽고메리 알고리즘을 곱셈 단계와 감소단계의 위치에 따라 구별하였다.[2] 이 각각의 분류된 알고리즘은 다음과 같다.

- ▶ SOS(Separated Operand Scanning)
- ▶ CIOS(Coarsely Integrated Operand Scanning)
- ▶ FIOS(Finely Integrated Operand Scanning)
- ▶ FIPS(Finely Integrated Product Scanning)
- ▶ CIHS(Coarsely Integrated Hybrid Scanning)

이 알고리즘에서 워드 크기는 임의적으로 선택할 수 있지만 선택에 따라 알고리즘의 특성이 영향을 받게 된다. Koc은 32bit로 워드 크기를 선택해서 Pentium -60 Linux 시스템에서 각 알고리즘을 비교 분석 하였고, 이 결과 CIOS 알고리즘이 가장 성능이 우수하며, SOS와 FIOS 또한 비교적 성능이 우수한 것으로 확인하였다.

2.4 Quatrus II PowerPlay Power Analyzer

Quatrus II PowerPlay Power Analyzer[8]는 static power dissipation과 dynamic power dissipation을 예상하여 보여줌으로써 개발자들에게 더 좋은 설계를 도와준다.. 이 툴은 설계한 디자인의 구현단계에서의 전력소모를 예측할 수 있게 하며, 입력 값에 따라 동작하는 dynamic 파워 역시 계산할 수 있게 한다. 이러한 계산들은 이미 실제 모델에서 측정되어진 데이터들을 기초로 하여 제작되어져서 정확한 전력 소모를 예측할 수 있게 도와준다.

III. OPERAND SCANNING 모듈러 곱셈의 구현

T의 값이 N의 값보다 같거나 크게 된다면 T에서 N을 빼주는 연산이 필요하게 된다. 하지만 이 마지막 연산은 나머지 알고리즘에서도 공통으로 쓰이게 되므로 이 논문에서는 알고리즘의 비교를 위해 마지막 연산은 고려하지 않는다. 또한 이 연산은 특정 조건을 제외하고는 거의 쓰이지 않는다는 것이 증명되었다.[9][10] 이 논문에서는 공통적으로 Radix W의 크기를 Radix-16 방식으로 4비트씩 연산을 수행하도록 구현하였다.

3.1 SOS 알고리즘 구조

SOS 알고리즘은 곱셈 단계와 감소 단계가 완전히 분리된 두개의 루프를 통해 차례로 수행된다. 이 논문에서 구현을 위하여 크게 두개의 모듈로 나누었다. 첫번째 32비트 곱셈기로 이 결과가 SOS AU 모듈에 64비트의 값을 전달하게 된다. SOS AU모듈은 8개의 SOS AU Element로 구성되어 있으며 이들은 각각 32x4 곱셈기와 Adder 그리고 m값을 구하기 위한 연산이 들어있다(그림 1).

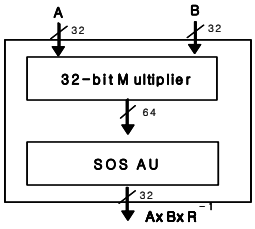

그림 1. SOS의 구조

3.2 CIOS 알고리즘 구조

CIOS는 곱셈과 감소단계가 통합되어진 방법이다. FIOS와의 차이점은 곱셈단계와 감소단계 사이에서 중간값 m을 계산한다.

이 논문에서는 CIOS 알고리즘의 구현을 위해 8개의

CIOS Element 만들고 구성을 이루었다. 이들 각 Element들은 32x4곱셈기와 Shifter와 연산부를 각각 가지고 있다. 그리고 각각의 연산부(AU)에는 m값을 구하기 위해 곱셈의 마지막 4bit만을 계산하는 부분과 Adder들로 구성되어지게 된다.

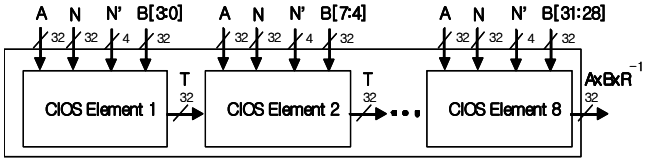

그림 2. CIOS의 구조

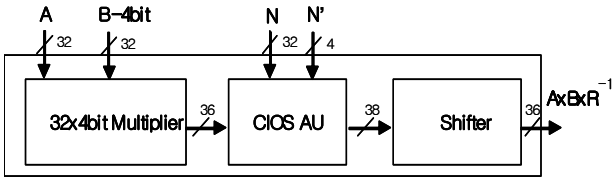

그림 3. CIOS Element의 구조

3.3 FIOS 알고리즘 구조

FIOS방식은 곱셈 단계와 감소단계가 하나의 루프에서 수행되어, 하나의 루프로 합쳐져 있으며 m계산을 제일 먼저 수행하게 된다.

FIOS 알고리즘의 구현은 CIOS와 비슷하게 8개의 FIOS Element들로 나누어진다. 이 각각의 Element에 또 하나의 작은 Element(SE)들로 구성되고 SE는 4bit 곱셈기와 m을 구하기 위한 곱셈의 4비트만 구해지는 모듈 그리고 Adder들로 구성되어있다.

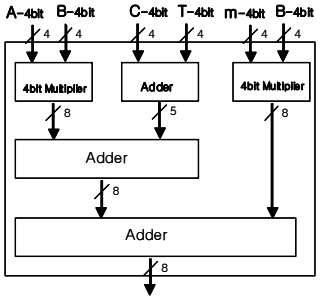

그림 4. FIOS SE의 구조

Ⅳ. 실험결과

SOS, CIOS, FIOS 알고리즘은 ALTERA사의 FPGA 상에 구현하였으며, 각 FPGA Family마다 하나의 Device를 선정하여 실험을 수행하였다. 표 1은 실험에 사용한 Device들을, 그림 5는 알고리즘들을 구현하기 위해 사용된 FPGA Logic Element의 수를, 그림 6은 구현한 각 알고리즘의 Longest Propagation Delay를 나타내고 있다. 그림에서 보는 바와 같이, Area의 측면에서는 SOS와 CIOS가 우수하였으며, 특히 Stratix II Family에서 가장 적은 Logic Element를 사용하였다. Performance의 측면에서는 SOS가 가장 우수한 특성을 보였고, CIOS가 그 뒤를 이었다. 특히 Stratix와 Stratix II Family에서의 성능이 우수하였다.

Family	Device	Core Voltage (V)
Stratix	EP1S80F1508C5	1.5
Stratix II	EP2S60F1020C5ES	1.2
Stratix GX	EP1SGX40GF1020C7	1.5
Cyclone	EP1C20F400C8	1.5

표 1. 구현에 사용한 Device들의 정보

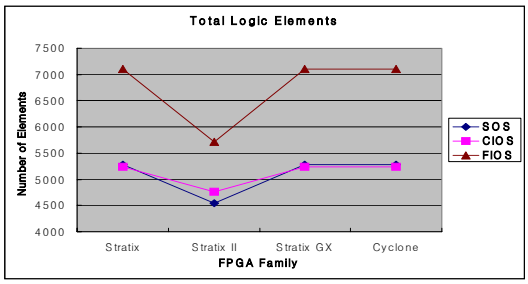

그림 5. 알고리즘 구현에 사용한 Logic Element의 수

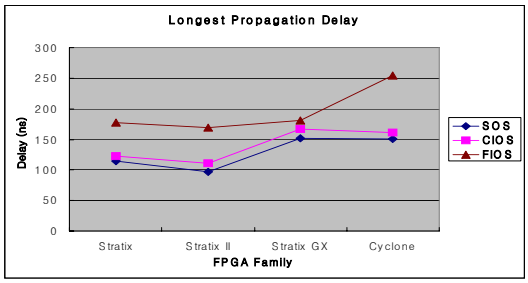

그림 6. 구현한 알고리즘의 Longest Propagation Delay

구현한 알고리즘들의 전력소모를 비교하기 위해서, 1000개의 입력값을 난수생성(Random Generation)하여 실험한 후, 그 값들의 평균치를 분석에 이용하였다.

그림 7은 구현한 각 알고리즘의 Static Thermal Power Dissipation을, 그림 8은 1000개의 난수 입력에 대한 각 알고리즘의 평균 Dynamic Thermal Power Dissipation을, 그림 9는 각 알고리즘의 Total Thermal Power Dissipation을 나타내고 있다.

그림에서 보는 바와 같이, Static Power의 측면과 Dynamic Power의 측면에서 모두 SOS가 가장 우수하고, CIOS 근소한 차이로 그 뒤를 잇는 것을 알 수 있다. 또한, 전력소모가 가장 적은 Family는 Cyclone이고, 가장 많은 Family는 Stratix 임을 확인할 수 있었다.

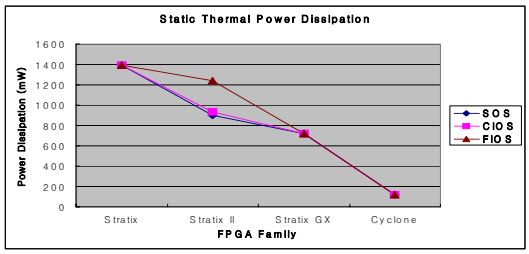

그림 7. 구현한 알고리즘의 Static Thermal Power Dissipation

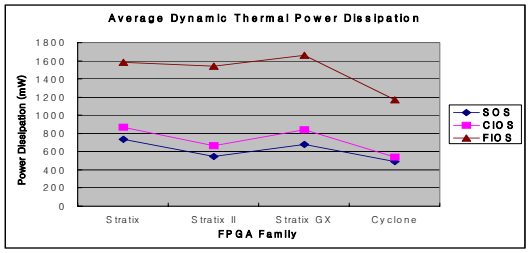

그림 8. 구현한 알고리즘의 평균 Dynamic Thermal Power Dissipation

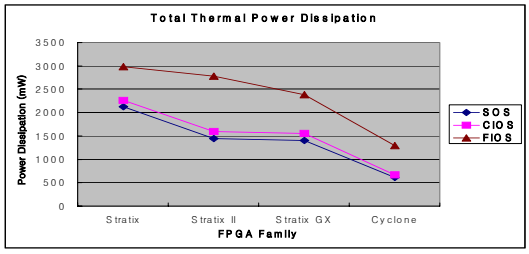

그림 9. 구현한 알고리즘의 Total Thermal Power Dissipation

V. 결론

휴대기기의 암호화 시스템 등에서 일반적으로 쓰이는 RSA 같은 알고리즘을 위해서
는 지수승의 나머지 계산이 이루어져야 한다. P. L. Montgomery는 이 연산을 위해 하드웨어에 적합한 알고리즘을 제시했고 Koc은 이 몽고메리 알고리즘 기술을 기반으로 몇 가지 효율적인 알고리즘을 개발했다.
본 논문에서는, Koc이 개발한 알고리즘들 중에서 성능이 입증된 SOS, CIOS, FIOS 알고리즘을 ALTERA FPGA 상에 구현하였다. 구현한 알고리즘을 다양한 환경에서 실험한 결과, 성능이나 면적, 전력소모 등에 대한 각 알고리즘의 특성을 확인해 볼 수 있었다. 성능, 면적, 전력소모의 모든 측면에서, SOS 알고리즘이 가장 우수한 특성을 나타내는 것을 확인하였고, CIOS 알고리즘이 근소한 차이로 그 뒤를 이었다. 또한, ALTERA FPGA Family 중 성능과 면적의 측면에서는 Stratix II가 가장 우수한 특성을 보였고 전력소모의 측면에서는 Cyclone이 가장 우수한 특성을 보였다.

본 논문에서 도출한 이와 같은 특성을 이용하여 RSA 알고리즘의 하드웨어 구현 시 활용한다면, 적합한 알고리즘의 선택에 큰 도움을 줄 것이라 예상한다.

VI. 감사의 글

본 연구는 한국과학재단 특정기초연구(R01-2005-000-11215-0(2005))지원에 의해 수행되었음.

참고문헌

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

[2] Koc, C. K, Acar, T. and Kaliski, jr. B. S. "Analyzing and Comparing Montgomery Multiplication Algorithms," pp. 26-33, IEEE Micro, June, 1996.

[3] W.Diffle, M. Hellman, "New Directions in Crytography" IEEE Trans. on Info. Theory, vol, IT-22(6) pp.644-654, 1976

[4] Rivest, R. L, Shamir, A. Adleman, L, "A Method for Obtaning Digital Signatures and Public-key Cryptosystems", Communication of the ACM, 21, pp. 120-126. 1978

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

[6] McIvor. C., McLoone. M., McCanny. J.V.,"FPGA Montgomery multiplier architectures - a comparison"", IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 279 - 282 , April 20-23, 2004

[7] 안준언, 유기영 "고속 RSA 암호 시스템을 위한 몽고메리 알고리즘의 구현 및 분석",한국통신정보보호학회 논문지 92 (p.61 -71), 1999.

[8] http://www.altera.com/products/software/products/quartus2/design/qts-power_analysis.html

[9] C.D. Walter, " Montgomery's Multiplication Technique : How to make it Smaller and Faster", Workshop on Cryptographic Hardware and Embedded Systems, LNCS 1717, pp80-93, August 1999, USA

[10] C.D. Walter, " Montgomery Exponentiation Needs no Final Subtrations", Electronics Letters, 35(21): 1831-1832, October 1999.