JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 21, No. 8 : 1471~1479 Aug. 2017

32-비트 몽고메리 모듈러 곱셈기 기반의 2,048 비트 RSA 공개키 암호 프로세서

조욱래 · 신경욱*

2,048 bits RSA public-key cryptography processor based on 32-bit Montgomery modular multiplier

Wook-Lae Cho · Kyung-Wook Shin*

School of Electronic Engineering, Kumoh National Institute of Technology, Gumi, Kyungbuk 39177, Korea

요 약

2,048 비트의 키 길이를 지원하는 RSA 공개키 암호 프로세서를 설계하였다. RSA 암호의 핵심 연산인 모듈러 곱셈기를 워드 기반의 몽고메리 곱셈 알고리듬을 이용하여 설계하였으며, 모듈러 지수승 연산은 Left-to-Right(LR) 이진 멱승 알고리듬을 이용하여 구현하였다. 모듈러 곱셈에 8,448 클록 사이클이 소요되며, RSA 암호화와 복호화에 각각 185,724 클록 사이클과 25,561,076 클록 사이클이 소요된다. 설계된 RSA 암호 프로세서를 Virtex 5 FPGA로 구현하여 하드웨어 동작을 검증하였다. 0.18 ㎛ CMOS 표준셀을 사용하여 100 MHz의 동작 주파수로 합성한 결과, RSA 암호 프로세서는 12,540 GE로 구현되었고, 12 kbit의 메모리가 사용되었다. 동작 가능한 최대 주파수는 165 MHz로 평가되었으며, RSA 암호화, 복호화 연산에 각각 1.12 ms, 154.91 ms가 소요되는 것으로 예측되었다.

ABSTRACT

This paper describes a design of RSA public-key cryptography processor supporting key length of 2,048 bits. A modular multiplier that is core arithmetic function in RSA cryptography was designed using word-based Montgomery multiplication algorithm, and a modular exponentiation was implemented by using Left-to-Right (LR) binary exponentiation algorithm. A computation of a modular multiplication takes 8,386 clock cycles, and RSA encryption and decryption requires 185,724 and 25,561,076 clock cycles, respectively. The RSA processor was verified by FPGA implementation using Virtex5 device. The RSA cryptographic processor synthesized with 100 MHz clock frequency using a 0.18 um CMOS cell library occupies 12,540 gate equivalents (GEs) and 12 kbits memory. It was estimated that the RSA processor can operate up to 165 MHz, and the estimated time for RSA encryption and decryption operations are 1.12 ms and 154.91 ms, respectively.

키워드 : RSA, 공개키 암호, 몽고메리 모듈러 곱셈, LR 이진 멱승

Key word : RSA, public-key cryptography, Montgomery modular multiplication, LR binary exponentiation

Received 12 April 2017, **Revised** 19 April 2017, **Accepted** 09 June 2017

* **Corresponding Author** Kyung-Wook Shin(E-mail:kwshin@kumoh.ac.kr, Tel:+82-54-478-7427)
School of Electronic Engineering, Kumoh National Institute of Technology, Gumi, Kyungbuk 39177, Korea

Open Access https://doi.org/10.6109/jkiice.2017.21.8.1471 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

데이터, 사람, 사물, 공간 등 모든 것이 서로 연결되어 정보교류를 하는 초연결사회(hyper-connected society)가 도래함에 따라 사물인터넷(Internet of Things : IoT)이 ICT(information and communication technology)의 핵심 기술로 부상하고 있다. IoT는 다양한 산업 분야에 적용이 되어 새로운 부가 가치를 창출하고 있으며, 홈/가전, 의료, 교통 등 다양한 분야에서 본격적인 시장 활성화가 진행되고 있다. IoT 기술의 활성화 및 신규 서비스 창출을 위해 정보보안은 반드시 제공해야 하는 필수적 요소이며, 정보보호가 담보되지 않은 환경에서 치명적인 보안 위협들이 발생할 것으로 예상된다.

정보보호를 위한 다양한 암호기술들이 사용되고 있으며, 대칭키 암호(symmetric key cipher), 해시(hash), 공개키 암호(public key cryptography) 등을 이용한 무결성(integrity), 기밀성(confidentiality), 사용자 및 기기 간의 인증(authentication), 키 분배(key distribution) 기술이 IoT 보안의 핵심요소이다. IoT 디바이스는 CPU 성능, 메모리 크기, 소비 전력 등의 제한된 자원을 가지므로, 기기의 성능과 보안 강도를 고려한 경량암호 기술이 필요하다[1].

최근 IoT, RFID의 보안에 적합하도록 제안된 대칭키 방식의 경량 암호(lightweight cryptography) 알고리듬으로 PRESENT[2], LEA[3] 등이 있으며, 이들 대칭키 알고리듬을 충분히 활용하기 위해서는 사용자 및 기기 간 인증과 키 분배 기술이 필수적으로 요구된다. 공개키 암호와 비밀키 암호는 정보보안에서 상호 보완적인 기능을 수행하며, 송·수신자 간의 인증, 키 분배, 무 결성 검증 등을 수행하는 공개키 암호시스템의 중요성은 날로 강조되고 있다.

대표적으로 사용되는 공개키 암호 알고리듬으로는 RSA(Rivest, Shamir, Adleman)[4]와 타원곡선 암호(Elliptic Curve Cryptography: ECC)[5] 알고리듬 등이 있다. RSA의 안전성은 두 개의 큰 정수를 곱한 수의 인수분해가 어려운 점에 기반을 두며, ECC는 타원 곡선군의 이산대수 문제에 안정성을 둔다. RSA 공개키 암호는 많은 국제기구에서 표준으로 지정하고 있고 산업 표준으로도 권장되고 있으며, 현재 에너지/건설/제조/환경·재난/교통/헬스케어/스마트 홈 등 분야별로 다양하게 활용되고 있다. 한국인터넷진흥원(KISA)에서는 2011년부터 2,048 비트 이상의 키 길이를 지원하는 RSA 암호를 권장하고 있다.

본 논문에서는 2,048 비트의 키 길이를 지원하는 RSA 공개키 암호 프로세서를 설계하고 FPGA 구현을 통해 하드웨어 동작을 검증하였다. Ⅱ장에서는 RSA 공개키 암호 알고리듬을 설명하고, Ⅲ장에서는 RSA 암호의 핵심 연산인 모듈러 곱셈 알고리듬에 관해서 설명하며, Ⅳ장에서 RSA-2048 프로세서 설계에 대해 설명한다. 설계된 RSA 프로세서의 기능검증과 FPGA 검증 결과는 Ⅴ장에 서술했으며, Ⅵ장에서 결론을 맺는다.

Ⅱ. RSA 공개키 암호 알고리듬[4]

Rivest, Shamir, Adleman에 의해 개발된 RSA는 공개키 암호 중 하나로, 암호화, 전자서명, 인증 등의 정보보안에 광범위하게 활용되고 있다. RSA 알고리듬의 안정성은 두 개의 큰 소수(prime number)의 곱에 대한 인수분해가 어렵다는 것에 기반을 두고 있다. RSA 알고리듬의 암호화와 복호화에 사용되는 공개키 N과 e, 그리고 개인키 d는 다음의 과정으로 생성된다.

· 두 개의 서로 다른 소수 p, q (단, $p \neq q$)를 선택한다.
· 두 소수의 곱 $N = p \cdot q$를 구한다.
· $\Phi(N) = (p-1) \cdot (q-1)$를 구한다.
· 정수 e (단, $e < \Phi(N)$이고, e와 $\Phi(N)$은 서로소)를 찾는다.
· $d \cdot e = 1 \bmod \Phi(N)$를 만족하는 정수 d를 구한다.

암호화 연산에 필요한 (e, N)은 공개키로 사용되고, 복호화 연산에 필요한 d는 개인키이며, 타인에게 노출되지 않아야 한다. 식 (1)은 공개키 (e, N)를 사용하여 평문 M을 RSA 암호화하여 암호문 C가 생성되는 연산을 나타낸다. 식 (2)는 개인키 d를 사용하여 암호문 C를 복호화하여 평문 M이 생성되는 연산을 나타낸다.

$$C = M^e \bmod N \tag{1}$$

$$M = C^d \bmod N \tag{2}$$

식 (1)과 식 (2)에 의하면, RSA 암호화/복호화는 모듈러 멱승(modular exponentiation) 연산으로 수행되며,

모듈러 멱승 연산은 반복적인 모듈러 곱셈으로 구현된다. 일반적으로 RSA 암호체계의 안정성을 보장받기 위해서는 키 길이가 2,048 비트 또는 3,072 비트로 매우 커야 한다. 큰 정수에 대한 반복적인 모듈러 곱셈은 연산 시간이 길고, 하드웨어 복잡도가 크다. 따라서 RSA 공개키 암호의 효율적인 하드웨어 구현을 위해서는 모듈러 곱셈 및 멱승 알고리듬에 대한 고려가 필요하다.

Ⅲ. 몽고메리 모듈러 곱셈 알고리듬[6]

RSA 공개키 암호 알고리듬의 핵심 연산인 모듈러 곱셈을 위한 다양한 알고리듬들이 연구되었으며, 그 중 몽고메리 모듈러 곱셈 알고리듬이 가장 널리 사용되고 있다. 그림 1은 기본 몽고메리 모듈러 곱셈 알고리듬의 슈도코드(pseudo code)를 나타내며, 모듈러 곱셈을 나눗셈이나 반복적인 뺄셈 없이 단순 덧셈과 시프트 연산만으로 구현할 수 있다. 큰 정수 A와 B에 대한 몽고메리 모듈러 곱셈 알고리듬은 식 (3)과 같이 표현된다. 입력 A와 B는 N 보다 작고, $R=2^k$과 N은 서로 소이며, N의 범위는 $2^{k-1} < N < 2^k$이다. 여기서 k는 키 길이를 나타낸다.

$$Z = A \cdot B \cdot R^{-1} \bmod N \tag{3}$$

그림 1의 몽고메리 모듈러 곱셈은 승수의 최하위 비트(LSB)부터 1 비트 단위로 스캔하여 k 비트의 가산이 수행되며, 시프트를 통하여 리덕션(reduction)을 한다. 이를 직접 하드웨어로 매핑하여 구현하는 경우, k 비트의 레지스터가 필요하다. RSA에서 키 길이 k는 1,024 비트, 2,048 비트 또는 3,072 비트로 매우 큰 값을 가지므로, k 비트의 레지스터는 큰 하드웨어 자원을 필요로 한다. 또한, k 비트 가산의 캐리전파 경로에 의한 지연이 매우 커지게 되며, 가산 과정의 캐리 지연을 고려하여 데이터패스를 작게 분할하여 구현하면 소요 사이클 수가 크게 증가하게 된다. 따라서 기본 몽고메리 모듈러 곱셈 알고리듬을 하드웨어로 직접 매핑하여 구현하면 회로 성능이 저하된다.

모듈러 곱셈기의 성능을 개선하기 위한 다양한 구현 방법들이 발표되었으며, 수정형 Booth 기반 몽고메리 곱셈 [7, 8], High-Radix 몽고메리 곱셈 [9-11], Koc [12]이 발표한 5가지 몽고메리 곱셈 알고리듬의 CIOS (Coarsely Integrated Operand Scanning) [13], FIPS (Finely Integrated Product Scanning) [14] 등이 있다. 수정형 Booth 알고리듬은 기본 몽고메리 곱셈 알고리듬에 비해 연산량이 감소되나, k 비트 레지스터를 사용해야 하는 점은 동일하여 많은 하드웨어 자원을 필요로 한다. Koc [12]이 발표한 5가지 곱셈 알고리듬(SOS,

Input : $A = \{a_{k-1}, \cdots, a_1, a_0\}_2$, $B = \{b_{k-1}, \cdots, b_1, b_0\}_2$, $N = \{n_{k-1}, \cdots, n_1, n_0\}_2$
Output : $Z = \{z_{k-1}, \cdots, z_1, z_0\}_2 = ABR^{-1} \bmod N$

```
1 : Z ← 0;
2 :   for i = 0 to k − 1 do
3 :     Z ← Z + a_i × B;
4 :     Z ← Z + z_0 × N;
5 :     Z ← Z/2;
6 :   end for
7 : if Z > N then Z ← Z − N;
```

Fig. 1 Montgomery modular multiplication algorithm

Input : $A = \{a_{s-1}, \cdots, a_1, a_0\}_{2^w}$, $B = \{b_{s-1}, \cdots, b_1, b_0\}_{2^w}$, $N = \{n_{s-1}, \cdots, n_1, n_0\}_{2^w}$, $R = 2^k$, $W = 2^w$, $n' = -n_0^{-1} \bmod W$
Output : $Z = \{z_{s-1}, \cdots, z_1, z_0\}_{2^w} = ABR^{-1} \bmod N$

```
1 : Z = 0; u = 0; S = 0;
2 : for i = 0 to s − 1 do
3 :   S = z_0 + a_i b_0;
4 :   C_a = S/W; z_0 = S mod W;
5 :   x_i = z_0 n' mod W;
6 :   S = z_0 + x_i n_0;
7 :   C_b = S/W; z_0 = S mod W;
8 :   for j = 1 to s − 1 do
9 :     S = z_j + a_i b_j + C_a;
10:     C_a = S/W; z_j = S mod W;
11:     S = z_j + x_i n_j + C_b;
12:     C_b = S/W; z_{j−1} = S mod W;
13:   end for
14:     S = C_a + C_b + u;
15:     u = S/W; z_{s−1} = S mod W;
16: end for
17: if Z > N then Z = Z − N;
```

Fig. 2 Word-based Montgomery multiplication algorithm

$MM(Montgomery\ Multiplication)$
Mapping : $MM(A, R^2) = A \times R^2 \times R^{-1} = AR\ mod\ N$
Mapping : $MM(B, R^2) = B \times R^2 \times R^{-1} = BR\ mod\ N$
Multiplication : $MM(AR, BR) = AR \times BR \times R^{-1} = ABR\ mod\ N$
re-mapping : $MM(ABR, 1) = ABR \times 1 \times R^{-1} = AB\ mod\ N$

Fig. 3 Mapping and re-mapping process for Montgomery modular multiplication

CIOS, FIOS, FIPS, CIHS)의 경우, SOS를 제외한 나머지 4가지 알고리듬은 성능이 비슷하며, 메모리의 활용이 가능하다는 장점이 있다.

그림 2는 워드 기반(word-based) 몽고메리 모듈러 곱셈 알고리듬의 슈도코드를 나타내며, FIOS와 형태가 유사하다[8]. 각각의 k 비트 정수(A, B, N)를 w 비트의 워드 s 개로 분할하여 연산을 진행한다. 두 개의 반복 루프로 구성되며, 외부 반복루프에서는 승수를 워드 단위로 스캔하여 내부 반복루프에 사용될 부분곱(partial product)을 생성한다. 내부 반복루프에서는 매 반복마다 부분곱의 가산과 리덕션이 연산된다.

식 (3)과 그림 1에서 볼 수 있듯이, 몽고메리 모듈러 곱셈 알고리듬의 곱셈 결과는 R^{-1}을 포함하므로, 매 곱셈 연산마다 R을 곱해서 R^{-1}을 제거해야 하는 단점이 있다. 이를 개선하기 위해 그림 3과 같이 전처리 매핑(mapping)과 후처리 역매핑(re-mapping)을 적용하면, 몽고메리 곱셈을 보다 효율적으로 구현할 수 있다. A, B를 각각 $AR\ mod\ N$과 $BR\ mod\ N$으로 매핑하고, 매핑된 두 수를 몽고메리 곱셈하면 $ABR\ mod\ N$이 얻어진다. 최종적으로, $ABR\ mod\ N$을 역매핑하면 모듈러 곱셈 결과 $AB\ mod\ N$이 얻어진다.

Ⅳ. RSA-2048 프로세서 설계

4.1. 전체 구조

그림 4-(a)는 2,048 비트의 키 길이를 지원하는 RSA-2048 프로세서의 전체 구조이며, WMM(Word-based Montgomery Multiplier) 블록, 제어블록, 2,048 비트 메모리 6개로 구성된다. 메모리 KeyM은 2,048 비트의 키(공개키 또는 개인키) 값을 저장하며, 메모리 Msg는 입력 평문/암호문 메시지를 저장한다. WMM 블록은 몽고메리 모듈러 곱셈을 32 비트 단위로 연산하며, 그 중간 결과는 메모리 MMz에 저장된다. 메모리 RSAt

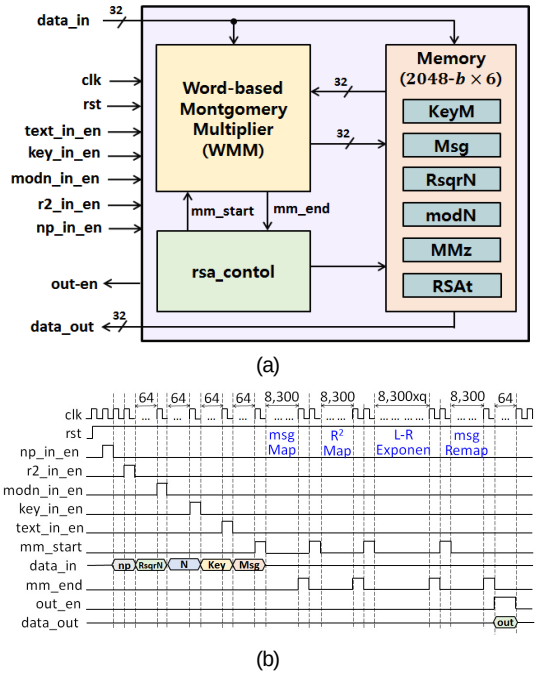

Fig. 4 (a) Overall architecture and (b) timing diagram of the RSA-2048 processor

는 RSA 암호화/복호화의 중간 결과를 저장한다. 메모리 RsqrN은 매핑인자 $R^2 \bmod N$을 저장하며, 메모리 modN은 모듈러 상수 N을 저장한다.

RSA-2048의 동작 타이밍 도는 그림 4-(b)와 같다. 매핑 인자 $R^2 \bmod N$, 모듈러 상수 N, 공개키/개인키, 그리고 평문/암호문이 각각 32 비트 단위로 64 클록 주기에 걸쳐 순차적으로 입력되어 해당 메모리에 저장된다. 데이터 입력이 완료된 후, mm_start 신호에 의해 전처리 매핑, 모듈러 멱승 연산, 역매핑이 순차적으로 진행되어 RSA 암호 또는 복호 연산이 수행된다. 연산이 완료되면, 암호/복호문이 32 비트 단위로 64 클록 주기 동안 out_en 신호와 함께 출력된다.

4.2. L-R 이진 모듈러 멱승 연산

멱승을 구현함에 있어서 가장 간단한 방법은 모듈러 곱셈을 지수만큼 반복 수행하는 것이다. 그러나 RSA 암호화/복호화 연산의 지수에 해당하는 키는 1,024, 2,048 또는 3,072 비트로 매우 큰 정수이므로, 모듈러 곱셈을 지수만큼 반복하는 것은 비효율적이다. 멱승을 계산하는 대표적인 알고리듬은 이진 방법(binary

$Input:$ $M = \{m_{s-1}, \cdots, m_1, m_0\}_{2^w}$
$G = \{g_{s-1}, \cdots, g_1, g_0\}_{2^w}$
$N = \{n_{s-1}, \cdots, n_1, n_0\}_{2^w}$
$E = \{e_{k-1}, \cdots, e_1, e_0\}_2$
$Output:$ $T = \{t_{s-1}, \cdots, t_1, t_0\}_{2^w}$
$= M^E \bmod N$

```
1 :  M ← MontMult(M, G, N);
2 :  T ← MontMult(1, G, N);
3 :  for i = k − 1 downto 0 do
4 :     T ← MontMult(T, T, N);
5 :     if e_i = 1 then
6 :        T ← MontMult(T, M, N);
7 :     end if
8 :  end for
9 :  T ← MontMult(1, T, N);
```

Fig. 5 Left-to-right binary method for modular exponentiation

method), m-ary 방법, 슬라이딩 윈도우(sliding window) 방법 등이 있으며, 이진 방법은 추가적인 레지스터가 필요하지 않아 효율적인 것으로 알려져 있다.

이진 방법은 지수를 최상위 비트(MSB)부터 스캔하는 L-R(Left-to-Right) 알고리듬과 최하위 비트(LSB)부터 스캔하는 R-L(Right-to-Left) 알고리듬으로 구분된다. R-L 알고리듬은 연산량이 절반으로 줄어드는 장점이 있지만, 두 배의 하드웨어 자원을 필요로 한다. 본 논문에서는 하드웨어 자원을 최소화하기 위해 L-R 알고리듬을 사용하여 설계하였다.

그림 5는 L-R 이진 멱승 연산 알고리듬의 슈도코드이다. M은 메시지이며, $G = R^2 \bmod N$는 매핑을 위한 매핑 인자이다. 단계-1과 -2는 몽고메리 곱셈을 위해 M과 '1'을 초기 매핑하는 단계이며, 단계-3~8은 키를 MSB부터 스캔하여 조건에 따라 몽고메리 곱셈기를 사용하여 제곱 연산과 곱셈 연산을 반복하는 단계이다. 단계-9는 몽고메리 곱셈을 마치고 최종 결과값 $M^E \bmod N$을 얻기 위한 역매핑 과정이다.

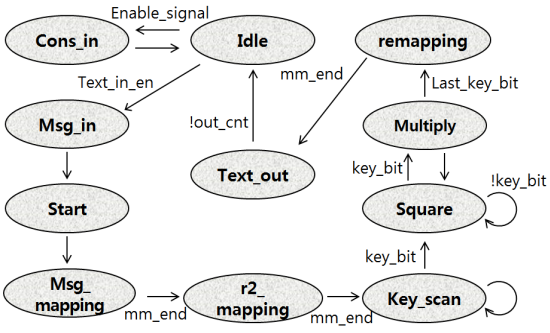

Fig. 6 State transition diagram for computing exponentiation

RSA-2048 프로세서의 제어블록은 그림 6의 상태천이도를 갖는 유한상태머신(finite state machine)으로 설계되었다. Cons_in 상태에서 모듈러 상수 N, 개인키 또는 공개키 값, 매핑인자 $R^2 \bmod N$ 등을 입력받고, Msg_in 상태에서 평문/암호문을 입력받는다. Start 상태를 거쳐 Msg_mapping, R2_mapping 상태로 천이된다. 일반적으로, RSA의 암호화에 사용되는 공개키 e는 개인키에 비해 매우 작은 비트(통상, 17 비트 정도를 사용함)의 정수가 사용되므로, 공개키 e의 비트 수를 확인하기 위해 Key_scan 상태를 거친다. Key_scan 상태에서는 메모리 KeyM에 저장된 키 값의 MSB부터 스캔하여 '1'이 처음 나오는 비트 위치를 찾는다. Key_scan을 위해 약 2,000 사이클 정도 소요되지만, 수십~수백만 사이클이 소요되는 RSA 연산에서 차지하는 비율이 매우 낮으므로, RSA 연산 성능에 미치는 영향이 매우 작다. Key_scan 상태에서 키 길이가 확인되면, Square 상태로 천이되어 그림 5의 L-R 이진 방법에 의해 멱승 연산이 시작된다. 메모리 KeyM에 저장된 공개키 또는 개인키의 비트별 값에 따라 Square 상태에서 제곱 연산(그림 5의 단계-4)과 Multiply 상태에서 곱셈 연산(그림 5의 단계-6)을 반복한다. 멱승 연산이 완료되면 Remapping 상태로 천이되며, 역매핑 후에는 Text_out 상태로 천이되어 최종 암호문 또는 평문을 출력한다.

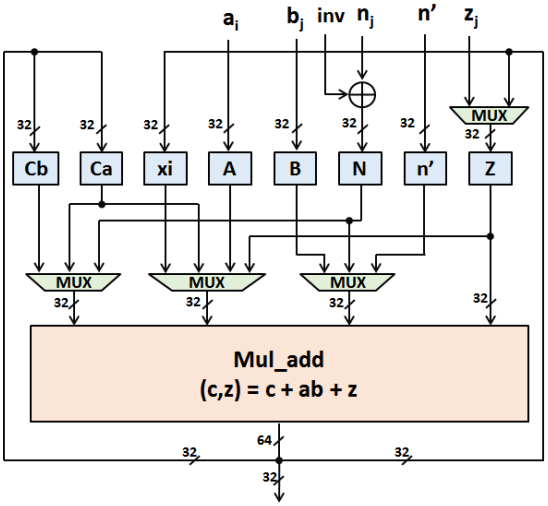

Fig. 7 Word-based Montgomery multiplier(WMM)

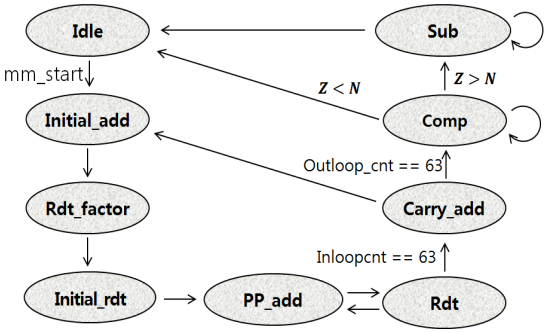

Fig. 8 State transition diagram for WMM

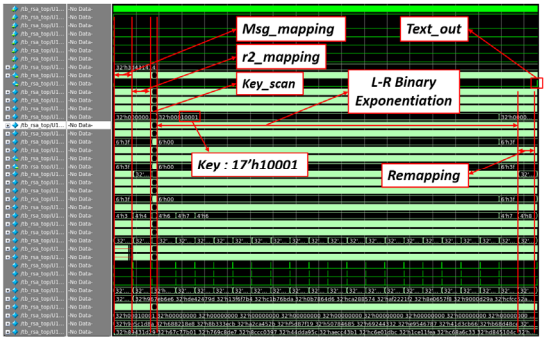

Fig. 9 RTL simulation result of RSA-2048 processor

4.3. 워드 기반 몽고메리 모듈러 곱셈기

입력 A, B, N을 받아 모듈러 곱셈을 계산하여 $Z = ABR^{-1} \bmod N$을 출력하는 워드 기반 몽고메리 모듈러 곱셈기(WMM)의 내부 구조는 그림 7과 같으며, 32 비트 레지스터 8개와 Mul_add 블록으로 구성된다. Mul_add 블록은 그림 2의 슈도코드가 나타내는 연산을 수행하며, Mul_add의 출력 64 비트 중, 상위 32 비트는 Ca 또는 Cb 레지스터에 저장되고, 하위 32 비트는 xi 또는 Z 레지스터에 저장된다.

WMM 블록은 그림 8의 상태 천이도에 의해 그림 2의 모듈러 곱셈 연산을 수행한다. mm_start 신호에 의해서 곱셈 연산이 시작되며, Initial_add 상태에서 $a_i b_0$와 z_0로부터 C_a와 z_0가 계산되며, Rdt_factor 상태는 내부 루프에서 사용될 리덕션 계수를 생성한다. Initail_rdt 상태에서 리덕션 연산에 사용될 C_b와 z_0가 계산된 후, 내부 루프에 해당하는 PP_add와 Rdt 상태로 천이된다. 내부 루프에서는 부분곱 가산과 리덕션 연산이 차례로 수행되고, $s-1$번 반복 후에 Carry_add 상태로 천이되며, 캐리를 가산하여 z_0와 u에 저장한다. 외부 루프의 s번 반복 연산을 위해 Carry_add 상태에서 Initial_add 상태로 천이하며, Outloop_cnt의 값이 s가 되면 곱셈기의 출력 값과 모듈러 N의 값을 비교하기 위해 Comp 상태로 천이한다. Comp 상태에서는 곱셈기의 출력 값인 Z와 모듈러 N을 상위 32 비트부터 비교를 시작하며, $Z < N$이면 Idle 상태로 천이되고, $Z > N$이면 Sub 상태로 천이되어 하위 32 비트부터 Z와 N에 대한 뺄셈이 시작된다. 곱셈기 연산이 종료 되어 Idle 상태로 천이될 때, mm_end 신호가 생성되어 다음 mm_start 신호가 입력될 때 까지 Idle 상태를 유지한다. 모듈러 곱셈에 8,448 클록 사이클이 소요된다.

Ⅴ. 기능검증 및 FPGA 구현

Verilog HDL로 설계된 RSA-2048 프로세서를 RTL 시뮬레이션을 통한 기능 검증과, FPGA 구현으로 하드웨어 동작을 확인하였다. 그림 9는 RSA-2048 프로세서의 기능검증 결과의 일부를 보인 것이다. 데이터를 모두 입력 받으면, Msg_mapping을 수행한다. r2_mapping 연산 후에 Key_scan을 통해 유효 키 길이를 확인한 후, L-R 이진 멱승 연산을 시작하며, 연산이 완료되면 역매핑 후에 out_en 신호와 함께 결과를 32 비트씩 64 클록 주기에 걸쳐 출력한다.

RTL 시뮬레이션을 통해 검증된 RSA-2048 프로세서를 Virtex5 XC5VSX50T FPGA 디바이스에 구현하여 하드웨어 동작을 검증하였다. 그림 10은 FPGA 디바이스, UART 인터페이스, C# 기반의 전용 소프트웨어로

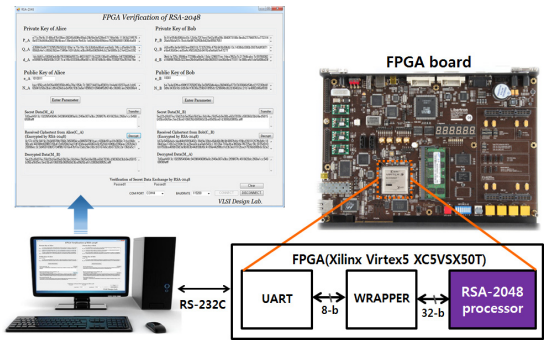

Fig. 10 FPGA verification system for RSA-2048 processor

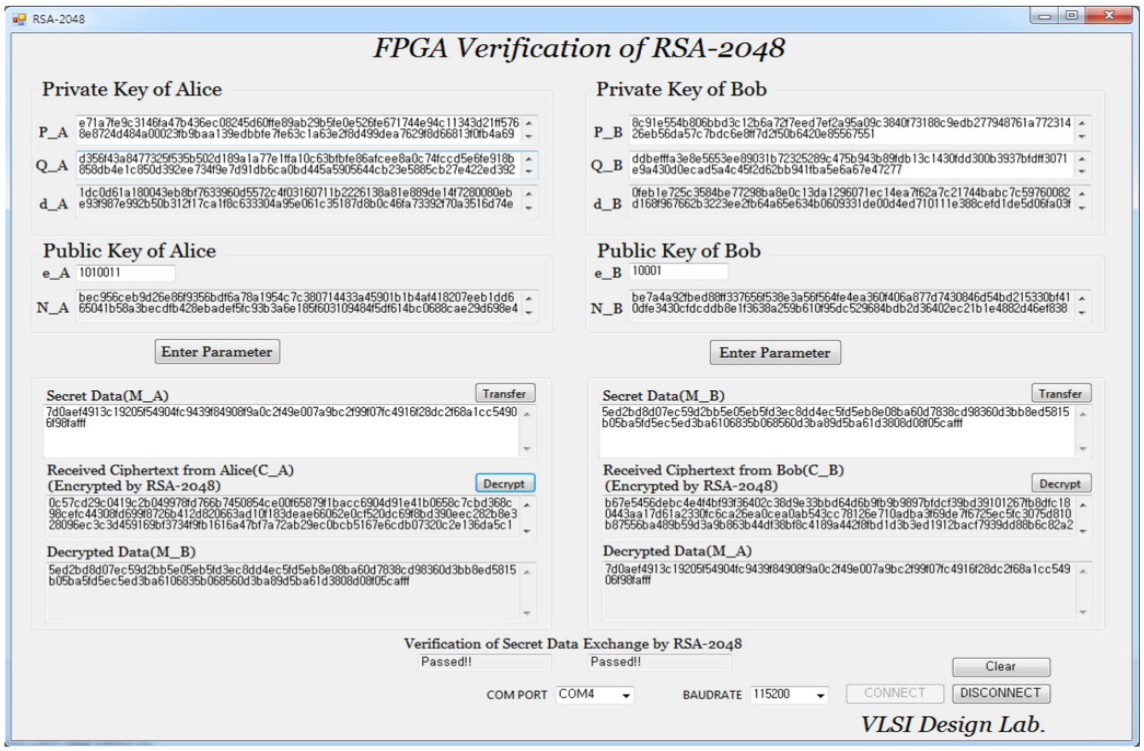

Fig. 11 FPGA verification results of RSA-2048 processor for key transmission protocol

구성된 FPGA 검증 시스템을 보이고 있다.

그림 11은 FPGA에 구현된 RSA-2048 프로세서를 이용하여 RSA 키 전송 프로토콜의 동작을 검증한 결과이다. Alice가 Bob과 통신을 원할 경우, Alice는 비밀키로 사용할 랜덤 정수 M_A를 생성하며, Bob의 공개키 (e_B, N_B)를 이용하여 M_A를 암호화한 후 Bob에게 전송한다. Bob은 전송받은 데이터를 Bob의 개인키 d_B를 이용하여 복호화 한다. Bob의 복호화 결과로 Alice가 생성한 랜덤 정수와 동일한 값이 얻어졌으며, 따라서 설계된 RSA-2048 프로세서가 올바르게 동작함을 확인할 수 있다.

Table. 1 Comparison of RSA cryptography processors

	[14]	[15]	[16]	Ours
key size [bit]	1,024	2,048	2,048	2,048
Max. frequency [MHz]	30	50	200	165
Area [GE]	30,000	38,000	61,000	12,540
Memory[bit]	N/A	N/A	N/A	12 k
Throughput [kbps]	9.23	18.4	57	13.2
APT	3.25	2.06	1.07	0.95

RSA-2048 프로세서는 공개키 $e(10001_{16})$를 사용한 암호화에 185,724 클록 사이클이 소요되고, 2,048 비트의 개인키 d를 사용한 복호화에 25,561,076 클록 사이클이 소요된다. 설계된 RSA-2048 프로세서를 0.18 ㎛ CMOS 표준 셀 라이브러리로 합성한 결과, 100 MHz의 동작 주파수에서 12,540 GE와 12 킬로비트(2,048 비트의 메모리 블록 6개)의 메모리로 구현되었다. 최대 동작 주파수 165 MHz에서 암호화 연산에 1.12 ms 가 소요되며, 복호화 연산에 157.91 ms 가 소요된다.

표 1은 문헌에 발표된 RSA 프로세서 사례들과 본 논문의 설계를 비교한 것이다. 문헌 [14-16]의 설계사례는 메모리를 사용하고 있으나, 메모리 크기에 관한 정보를 제공하지 않아 직접 비교할 수 없다. APT(Area Per Throughput)는 면적 대비 처리율을 의미하며, 낮을수록 성능이 좋다. 문헌 [14, 15]는 APT가 2.06 ~ 3.25

정도이고, [16]은 APT 수치가 준수하나, 높은 처리율만큼 많은 게이트를 필요로 하므로, 저면적이 중요한 응용분야에는 적합하지 않다. 본 논문의 RSA 프로세서는 APT 수치가 0.95 정도이며, 면적이 작고 데이터 처리율이 우수하여 저면적, 저전력 구현이 필요한 응용분야에 적합한 것으로 평가된다.

Ⅵ. 결 론

2,048 비트의 키를 지원하는 RSA 공개키 암호 프로세서를 설계하고, FPGA 구현을 통해 하드웨어 동작을 검증하였다. RSA 연산에 핵심이 되는 모듈러 곱셈기를 워드기반 몽고메리 모듈러 곱셈(WMM) 알고리듬을 이용하여 설계하였다. 곱셈기에 필요한 최소 레지스터만 사용하였으며, 큰 정수를 저장하기 위해 메모리를 사용함으로써 전체적인 하드웨어 자원을 최소화하였다. RSA-2048 프로세서를 0.18㎛ CMOS 공정으로 합성한 결과 100 MHz의 동작주파수에서 암호 프로세서는 12,540 GE, 12 킬로비트의 메모리로 구현되었다. 최대 165 MHz로 동작 가능하며, RSA 암호화와 복호화 연산에 각각 1.12 ms와 154.91 ms가 소요되는 것으로 예측되었다. 본 논문에서 설계된 2,048 비트 RSA 암호 프로세서는 제한된 하드웨어 자원을 갖는 디바이스 간의 인증 또는 키 관리를 위한 공개키 기반 보안 하드웨어 구현에 활용이 가능하다. 향후, 전력분석 공격 등 부채널 공격에 대한 대응 기법을 적용하는 연구를 할 예정이다.

ACKNOWLEDGMENTS

- This work was supported by Korea Institute for Advancement of Technology (KIAT) grant funded by Korean government (Ministry of Trade, Industry & Energy, HRD Program for Software-SoC convergence) (No. N0001883).
- The authors also are thankful to IDEC for EDA software support.

REFERENCES

[1] Korea Internet & Security Agency (KISA). IoT Common Security Principle v1.0 [Internet]. Available: http://www.kisa.or.kr/public/laws/laws3_View.jsp?mode=view&p_No=259&b_No=259&d_No=67&ST=T&SV=/.

[2] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, “PRESENT: An Ultra-Lightweight Block Cipher,” *Cryptographic Hardware and Embedded Systems (CHES 2007)*, vol. 4727 LNCS, Springer, pp. 450-466, Aug. 2007.

[3] TTA std. TTAK.KO-12.0223, *128-Bit Block Cipher LEA*, Telecommunications Technology Association, 2013.

[4] R. Rivest, A. Shamir, and L. Adleman, “A method for obtaining Digital Signatures and Public-Key Cryptosystems,” *Communications of Association for Computing Machinery (ACM)*, vol. 21, no. 2, pp. 120-126, Feb. 1978.

[5] FIPS PUB 186-2, *Digital Signature Standard (DSS)*, National Institute of Standard and Technology (NIST), Jan. 2000.

[6] P. L. Montgomery, “Modular multiplication without trial division,” *Mathematics of computation*, vol. 44, no. 170, pp. 519-521, Apr. 1985.

[7] A. Kauther, S. Sami, and A. Ahmed, “Enhancement of hardware modular multiplier radix-4 algorithm for fast RSA cryptosystem,” *International Conference on Computing, Electrical and Electronic Engineering (ICCEEE)*, Khartoum, Sudan, pp. 692-696, Aug. 2013.

[8] S. Rohith, and C. Mahesh, “FPGA implementation of 16 bit RSA cryptosystem for text message,” *International Journal of Computer Applications*, vol. 92, no. 8, Apr. 2014.

[9] A. Miyamoto, N. Homma, T. Aoki, and A. Satoh, “Systematic design of RSA processors based on high-radix montgomery multipliers,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 7, pp. 1136-1146, Jul. 2011.

[10] D. M. Wang, Y. Y. Ding, J. Zhang, J. G. Hu and H. Z. Tan, “Area-efficient and ultra-low-power architecture of RSA processor for RFID,” *Electronics letters*, vol. 48, no. 19, pp. 1185-1187, Oct. 2012.

[11] A. Rezai and P. Keshavarzi, “High-Throughput Modular Multiplication and Exponentiation Algorithms Using Multibit-Scan-Multibit-Shift Technique,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23,

no. 9, pp. 1710-1719, Sep. 2015.

[12] C. K. koc, T. Acar, and B. S. Kaliski, “Analyzing and comparing Montgomery multiplication algorithms,” *IEEE Micro*, vol. 16, no. 3, pp. 26-33, Jan. 1996.

[13] S. Tamura, C. Yamada and S. Ichikawa, “Implementation and Evaluation of modular multiplication based on Coarsely Integrated Operand Scanning,” *IEEE 2012 Third International Conference on Networking and Computing (ICNC)*, Tamilnadu, India, pp. 334-335, 2012.

[14] J. Shao, L. Wu and X. Zhang, “Design and implementation of RSA for dual interface bank IC card,” *2013 IEEE 10th International Conference on ASIC (ASICON)*, Shenzhen, China, pp. 1-4, 2013.

[15] M. S. Kim, Y. S. Kim and H. S. Cho, “Design of Cryptographic Hardware Architecture for Mobile Computing,” *Journal of Information Processing Systems*, vol. 5, no. 4, pp. 187-196, Dec. 2009.

[16] X. Zheng, Z. Liu and B. Peng, “Design and Implementation of Ultra low power RSA coprocessor,” *IEEE 4th International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM’08*, Dalian, China, pp. 1-5, 2008.

조욱래(Wook-Lae Cho)

2016년 2월 금오공과대학교 전자공학부(공학사)
2016년 3월~현재 금오공과대학교 대학원 전자공학과 석사과정 재학 중
※관심분야 : 통신 및 신호처리용 반도체 IP 설계, 정보보호용 반도체 IP 설계

신경욱(Kyung-Wook Shin)

1984년 2월 한국항공대학교 전자공학과(공학사)
1986년 2월 연세대학교대학원 전자공학과(공학석사)
1990년 8월 연세대학교대학원(공학박사)
1990년 9월~1991년 6월 한국전자통신연구소 반도체연구단(선임연구원)
1991년 7월~현재 금오공과대학교 전자공학부(교수)
1995년 8월~1996년 7월 University of Illinois at Urbana-Champaign(방문교수)
2003년 1월~2004년 1월 University of California at San Diego(방문교수)
2013년 2월~2014년 2월 Georgia Institute of Technology(방문교수)
※관심분야 : 통신 및 신호처리용 SoC 설계, 정보보호 SoC 설계, 반도체 IP 설계