ISSN : 1226-7244 (Print)
ISSN : 2288-243X (Online)
논문번호 20-04-06

j.inst.Korean.electr.electron.eng.Vol.24,No.4,961~968,December 2020
http://dx.doi.org/10.7471/ikeee.2020.24.4.961

타원곡선 암호를 위한 고성능 모듈러 곱셈기
A High Performance Modular Multiplier for ECC

최 준 영*, 신 경 욱*★

Jun-Yeong Choe*, Kyung-Wook Shin*★

Abstract

This paper describes a design of high performance modular multiplier that is essentially used for elliptic curve cryptography. Our modular multiplier supports modular multiplications for five field sizes over GF(p), including 192, 224, 256, 384 and 521 bits as defined in NIST FIPS 186-2, and it calculates modular multiplication in two steps with integer multiplication and reduction. The Karatsuba-Ofman multiplication algorithm was used for fast integer multiplication, and the Lazy reduction algorithm was adopted for reduction operation. In addition, the Nikhilam division algorithm was used for the division operation included in the Lazy reduction. The division operation is performed only once for a given modulo value, and it was designed to skip division operation when continuous modular multiplications with the same modulo value are calculated. It was estimated that our modular multiplier can perform 6.4 million modular multiplications per second when operating at a clock frequency of 32 MHz. It occupied 456,400 gate equivalents (GEs), and the estimated clock frequency was 67 MHz when synthesized with a 180-nm CMOS cell library.

요 약

타원곡선 암호에 필수적으로 사용되는 모듈러 곱셈의 고성능 하드웨어 설계에 대해 기술한다. 본 논문의 모듈러 곱셈기는 NIST FIPS 186-2에 정의된 소수체 상의 5가지 체 크기(192, 224, 256, 384, 521 비트)의 모듈러 곱셈을 지원하며, 정수 곱셈과 축약의 두 단계 과정으로 모듈러 곱셈을 연산한다. 고속 정수 곱셈을 위해 카라추바-오프만 곱셈 알고리듬이 사용되었고, 축약 연산을 위해 Lazy 축약 알고리듬이 사용되었다. 또한, Lazy 축약에 포함된 나눗셈 연산을 위해 Nikhilam 나눗셈 알고리듬이 사용되었으며, 나눗셈 연산은 주어진 모듈러 값에 대해 처음 한 번만 연산되고, 모듈로 값이 고정된 상태로 연속적인 모듈러 곱셈이 수행되는 경우에는 나눗셈을 거치지 않도록 하였다. 설계된 모듈러 곱셈기는 32 MHz의 클록 주파수로 동작하는 경우에 초당 640만번의 모듈러 곱셈을 연산할 수 있는 것으로 평가되었으며, 180-nm CMOS 셀 라이브러리로 합성한 결과, 67 MHz의 클록 주파수로 동작이 가능하며, 456,400 등가 게이트로 구현되었다.

Key words : Modular multiplication, Elliptic curve cryptography (ECC), Karatsuba-Ofman algorithm, Nikhilam division algorithm, Lazy reduction algorithm

---

* School of Electronic Engineering, Kumoh National Institute of Technology
★ Corresponding author
Email : kwshin@kumoh.ac.kr, Tel : +82-54-478-7427
※ Acknowledgment
- This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2020R1I1A3A04038083)
- This research was supported by the KIAT(Korea Institute for Advancement of Technology) grant funded by the Korea Government(MOTIE : Ministry of Trade Industry and Energy). (No. N0001883, HRD Program for Intelligent semiconductor Industry)
- Authors are thankful to IDEC for supporting EDA software.

Manuscript received Nov. 5, 2020; revised Dec. 22, 2020; accepted Dec. 23. 2020.

Ⅰ. 서론

공개키 암호 (public key cryptography)는 키 관리 기능과 보안성이 우수하다는 장점이 있으나 계산이 복잡하여 연산에 많은 시간이 소요된다. 대표적인 공개키 암호 방식인 타원곡선 암호(elliptic curve cryptography : ECC)[1]는 키 길이가 비교적 짧으면서도 RSA 공개키 암호와 비슷한 보안성능을 가지며, 연산 속도가 빠르다는 장점을 갖는다. ECC는 EC-DSA(Elliptic Curve Digital Signature Algorithm), ECIES(Elliptic Curve Integrated Encryption Scheme), EC-DH(Elliptic Curve Diffie-Hellman) 등의 공개키 암호 프로토콜에 폭넓게 사용된다.

ECC 기반 공개키 암호의 응용분야가 확대됨에 따라 고성능 ECC 프로세서의 중요성이 증가하고 있다. ECC의 기본 연산인 타원곡선 상의 스칼라 곱셈은 점 연산(점 덧셈, 점 두배)의 반복으로 계산되며, 점 연산은 모듈러 연산(곱셈, 나눗셈, 역원, 덧셈, 뺄셈)을 기반으로 구현된다. 따라서 고속 모듈러 연산회로는 고성능 ECC 프로세서 설계를 위한 핵심 기능블록으로 사용된다. 특히, 모듈러 곱셈 연산은 타원곡선 점 연산에서 가장 많은 연산량을 차지하므로, 모듈러 곱셈기의 효율적인 설계가 매우 중요하다[2, 3].

모듈러 곱셈은 정수의 곱셈 연산과 모듈러 축약(reduction)의 두 단계 연산으로 구현될 수 있으며, Montgomery 곱셈[4], Karatsuba-Ofman 곱셈[5], residue number system (RNS) 기반 모듈러 곱셈[6], Barret 축약 기반 모듈러 곱셈[7] 등 다양한 알고리듬이 제안되고 있다[8, 9].

본 논문에서는 N-비트 두 정수 곱셈 연산의 복잡도를 $O(N^2)$에서 $O(N^{\log_2 3})$로 줄여 큰 정수의 곱셈에 적합한 것으로 평가되는 카라추바-오프만 곱셈 알고리듬과 하나의 회로만으로 다양한 모듈러 값에 대한 모듈러 축약 연산을 수행할 수 있는 Lazy 축약 알고리듬[10]을 적용한 고성능 모듈러 곱셈기 구현 방법을 제안한다. Ⅱ장에서는 소수체 상의 모듈러 곱셈 연산에 사용된 카라추바-오프만 곱셈 알고리듬, Lazy 축약 알고리듬, Nikhilam 나눗셈 알고리듬을 설명하고, Ⅲ장에서는 이 알고리듬들을 적용한 고성은 모듈러 곱셈기 설계에 대해 설명한다. Ⅳ장에서는 본 논문에서 구현한 ModMul의 RTL 기능검증 결과 및 성능평가에 대해 기술하고, Ⅴ장에서 결론을 맺는다.

Ⅱ. GF(p) 상의 모듈러 곱셈 연산

2.1. 모듈러 곱셈 알고리듬

모듈러 곱셈은 ECC의 점 연산 구현에 사용되는 핵심 유한체 연산이므로, 모듈러 곱셈기 설계를 위해서는 타원곡선이 정의되는 유한체의 크기가 고려되어야 한다. 대표적인 ECC 표준인 NIST FIPS 186-2[11]에 정의된 소수체 상의 5가지 체 크기(field size) 192, 224, 256, 384, 521 비트를 지원하는 모듈러 곱셈 알고리듬을 제안한다. 본 논문에서 제안하는 모듈러 곱셈 알고리듬의 슈도코드는 그림 1과 같으며, n 비트의 피연산자 X와 Y를 $q=n/2$ 비트의 상위 절반 X_1, Y_1과 하위 절반 X_0, Y_0로 분할하여 연산하는 방법을 기반으로 한다. 병렬 곱셈기는 피연산자 비트 수(체 크기)의 제곱에 비례하여 하드웨어 복잡도가 커지므로, 384 비트, 521 비트 곱셈기는 매우 많은 게이트를 필요로 하여 비효율적이다. 본 논문에서는 고정된 264 비트 크기의 곱셈기에서 5가지 체 크기의 모듈러 곱셈이 연산될 수 있도록, 체 크기가 192, 224, 256 비트인 경우의 그림 1-(a) 슈도코드와 체 크기가 384, 521 비트인 경우의 그림 1-(b) 슈도코드로 분리하여 적용하였다.

그림 1-(b)의 슈도코드의 단계-1에서 단계-5까지에 포함된 곱셈은 카라추바-오프만 곱셈 알고리듬을 적용하여 계산되며, 2.2절에서 상세히 설명한다. 슈도코드의 나머지 단계에서는 Lazy 축약 알고리듬을 이용하여 모듈러 축약 연산이 수행되며, 2.3절에서 상세히 설명한다. 모듈러 곱셈 연산 초기에 주어진 모듈러스 값 M에 대해 계산되는 μ 값은 Nikhilam 나눗셈 알고리듬을 이용하여 계산되며, 이에 대해서는 2.4절에서 설명한다.

2.2. 카라추바-오프만 곱셈 알고리듬

피연산자가 100 비트 이상인 큰 정수의 곱셈을 하드웨어로 구현하는 경우, 회로 복잡도와 연산 시간이 피연산자 비트 수의 제곱에 비례해서 커지게 된다. 카라추바-오프만 곱셈 알고리듬[5]은 피연산자를 적절한 크기로 분할하고, 분할된 피연산자에 대해 곱셈 연산을 반복 처리함으로써 곱셈기의 회로 복잡도와 연산 시간을 줄이는 고속 알고리듬의 한 형태이다. 예를 들어, n 비트의 피연산자 X와 Y

$Input$: X , Y , M
X = $\{x_{n-1}, x_{n-2}, \cdots, x_1, x_0\}$,
Y = $\{y_{n-1}, y_{n-2}, \cdots, y_1, y_0\}$,
M = $\{m_{n-1}, m_{n-2}, \cdots, m_1, m_0\}$
$Output$: T = $(X \times Y)\ mod\ M$
$Precomputed$: $\mu = \lfloor 2^{2n+\alpha+\beta}/M \rfloor$

01 : $C := X \times Y$;
02 : s := $\lfloor C\ /2^{n+\alpha} \rfloor \times \mu$;
03 : m := $\lfloor s\ /\ 2^{n+\beta} \rfloor \times M$;
04 : T := $C - m$;
05 : $if\ (T < 0)\ then$
06 : T := $T + M$;
07 : $return\ T$

(a) for field sizes of n=192, 224, 256 bits

$Input$: X = $\{x_{n-1}, x_{n-2}, \cdots, x_1, x_0\} = \{X_1, X_0\}$
$X_1 = \{x_{n-1}, x_{n-2}, \cdots, x_q\}$
$X_0 = \{x_{q-1}, x_{q-2}, \cdots, x_0\}$, (q = n/2)
$Y = \{y_{n-1}, y_{n-2}, \cdots, y_1, y_0\} = \{Y_1, Y_0\}$
$Y_1 = \{y_{n-1}, y_{n-2}, \cdots, y_q\}$
$Y_0 = \{y_{q-1}, y_{q-2}, \cdots, y_0\}$, (q = n/2)
M = $\{m_{n-1}, m_{n-2}, \cdots, m_1, m_0\}$
$Output$: T = $(X \times Y)\ mod\ M$
$Precomputed$: $\mu = \lfloor 2^{2n+\alpha+\beta}/M \rfloor$

01 : $P_{00} := X_0 \times Y_0$; $Y_{sum} := Y_0 + Y_1$;
02 : $P_{11} := X_1 \times Y_1$; $X_{sum} := X_0 + X_1$;
03 : $P_{ss} := X_{sum} \times Y_{sum}$; $C_1 := \{P_{11}, P_{00}\} - P_{00} \times 2^q$;
04 : $C_2 := C_1 - P_{11} \times 2^q$;
05 : $C := C_2 + P_{ss} \times 2^q$;
06 : $s_0 := \lfloor C\ /2^{n+\alpha} \rfloor_{[(\frac{(n-\alpha)}{2}-1):0]} \times \mu_{[(\frac{(n+4)}{2}-1):0]}$;
07 : $s_1 := \lfloor C\ /2^{n+\alpha} \rfloor_{[(n-\alpha):\frac{(n-\alpha)}{2}]} \times \mu_{[(\frac{(n+4)}{2}-1):0]}$;
$s_{t1} := s_0$;
08 : $s_2 := \lfloor C\ /2^{n+\alpha} \rfloor_{[(\frac{(n-\alpha)}{2}-1):0]} \times \mu_{[(n+4):\frac{(n+4)}{2}]}$;
$s_{t2} := s_{t1} + s_1 \times 2^{\frac{(n+4)}{2}}$;
09 : $s_3 := \lfloor C\ /2^{n+\alpha} \rfloor_{[(n-\alpha):\frac{(n-\alpha)}{2}]} \times \mu_{[(n+4):\frac{(n+4)}{2}]}$;
$s_{t3} := s_{t2} + s_2 \times 2^{\frac{(n+4)}{2}}$;
10 : $s := s_{t3} + s_3 \times 2^{(n+4)}$;
11 : $m_0 := \lfloor s\ /\ 2^{n+\beta} \rfloor_{[(\frac{(n+(5-\beta)}{2}-1):0]} \times M_{[(q-1):0]}$;
12 : $m_1 := \lfloor s\ /\ 2^{n+\beta} \rfloor_{[(n+(5-\beta):\frac{(n+(5-\beta)}{2}]} \times M_{[(q-1):0]}$;
$m_{t1} := m_0$;
13 : $m_2 := \lfloor s\ /\ 2^{n+\beta} \rfloor_{[(\frac{(n+(5-\beta)}{2}-1):0]} \times M_{[n:q]}$;
$m_{t2} := m_{t1} + m_1 \times 2^q$;
14 : $m_3 := \lfloor s\ /\ 2^{n+\beta} \rfloor_{[(n+(5-\beta):\frac{(n+(5-\beta)}{2}]} \times M_{[n:q]}$;
$m_{t3} := m_{t2} + m_2 \times 2^q$;
15 : m := $m_{t3} + m_3 \times 2^n$;
16 : T := $C - m$;
17 : $if\ (T < 0)\ then$
18 : T := $T + M$;
19 : $return\ T$

(b) for field sizes of n=384, 521 bits

Fig. 1. Proposed pseudocode for modular multiplication.
그림 1. 제안된 모듈러 곱셈의 슈도코드

를 상위 부분 X_1, Y_1과 하위 부분 X_0, Y_0로 분할하여 식 (1-a), 식 (1-b)와 같이 다항식으로 표현하면, 두 수의 곱 XY는 식 (2)와 같에 계산될 수 있다. n 비트의 승수와 피승수를 q 비트 (단, $q=n/2$)로 분할하여 연산을 진행하므로, 곱셈기의 하드웨어 복잡도가 작아진다. 식 (2)에서 변수 z는 식 (3)과 같으며, z_0와 z_2의 계산 결과를 이용하여 식 (3-b)과 같이 z_1을 계산할 수 있으므로, 곱셈 횟수를 1회 줄일 수 있다[12].

$$X = X_1 2^q + X_0 \tag{1-a}$$

$$Y = Y_1 2^q + Y_0 \tag{1-b}$$

$$\begin{aligned} XY &= (X_1 2^q + X_0)(Y_1 2^q + Y_0) \\ &= z_2 2^{2q} + z_1 2^q + z_0 \end{aligned} \tag{2}$$

$$z_0 = X_0 Y_0 \tag{3-a}$$

$$\begin{aligned} z_1 &= X_1 Y_0 + X_0 Y_1 \\ &= (X_0 + X_1)(Y_0 + Y_1) - z_2 - z_0 \end{aligned} \tag{3-b}$$

$$z_2 = X_1 Y_1 \tag{3-c}$$

2.3. Lazy 축약 알고리듬

그림 1-(a)의 슈도코드 단계-1과 그림 1-(b)의 슈도코드의 단계-1～단계-5에서 얻어진 2n 비트의 곱셈결과 C는 축약 연산을 통해 n 비트의 모듈러 곱셈결과 $T = (X \times Y) \bmod M$으로 출력된다. 그림 1-(a)의 슈도코드에서 단계-2～단계-6 그리고 그림 1-(b)의 슈도코드에서 단계-6～단계-18은 축약 연산과정을 나타내고 있으며, 바렛 축약 연산 [7]을 기반으로 하는 Lazy 축약 알고리듬[10]을 사용한다. 고정된 크기의 곱셈기를 사용하여 연산이 이루어지므로, 체 크기에 따라 축약 연산과정이 달라지며, 그림 1-(b)의 슈도코드는 s와 m을 위해 사용되는 피연산자의 비트 길이가 곱셈기의 입력 비트보다 큰 경우에 적용된다.

Lazy 축약은 두 입력 X, Y의 곱을 식 (4)와 같이 연산하며, δ는 뺄셈 연산이 최대 한번만 수행되도록 $\delta = 2$로 결정되어 X, Y는 $0 \le X, Y < \delta M$의 범위를 갖는다.

$$0 \le (X \times Y) \bmod M < \delta M \tag{4}$$

축약 연산을 수행하기 위하여 $1/M$ (M은 모듈러스 값)의 추정 값과 2의 멱승 값을 이용하여 대략 2n 비트 길이의 μ를 미리 계산한다. 파라미터 α, β는 모듈러 축약 연산내의 근사 값을 조정하여 정확한 결과 값을 구하기 위해 사용되며, 각각 식 (5), 식 (6)으로 정의된다[10].

$$\alpha = \lfloor \log_2(\delta-1) \rfloor - 2 \quad (5)$$

$$\beta = \left\lfloor \log_2\left(\frac{k\delta^2}{\delta-1}\right) \right\rfloor - \lfloor \log_2(\delta-1) \rfloor + 3 \quad (6)$$

2.4. Nikhilam 나눗셈 알고리듬

그림 1의 슈도코드에서 사전에 계산되는 μ 값을 계산하기 위해서는 $2^{2n+\alpha+\beta}$를 모듈러스 M으로 나누는 나눗셈 연산이 필요하며, 본 논문에서는 이를 위해 Nikhilam 나눗셈 알고리듬[13]을 적용하였다. Nikhilam 나눗셈은 일반적인 나눗셈 연산과 달리 2의 멱승과 제수(divisor)의 차를 이용하는 방법이다. 본 논문에서 사용된 Nikhilam 나눗셈 알고리듬의 슈도코드는 그림 2와 같으며, 체 크기 k 비트에 대해 $(2k+4)$ 비트의 피제수 (dividend)와 k 비트의 제수를 입력받아 나눗셈 연산을 출력한다. 고정된 크기의 곱셈기를 사용하여 연산이 이루어지므로, 체 크기가 384 비트 이상인 경우에는 단계-3에서 단계-10까지의 연산이 수행된다. 단계-14는 단계-13의 연산 결과값을 비교하는 과정으로 결과값이 k 비트 이상일 경우 단계-2로 되돌아가 연산을 반복하며, 그렇지 않을 경우 단계-18로 가서 제수와 값을 비교한 후 최종 결과값인 몫을 출력한다. Nikhilam 나눗셈 연산의 결과로 몫과 나머지가 얻어지지만 모듈러 축약 연산에는 몫만 필요하므로, 단계-2의 연산의 결과만 사용된다.

Input : *Dividend* $X = \{x_{2k+3}, x_{2k+2}, \cdots, x_1, x_0\}$,
Divisor $Y = \{y_{k-1}, y_{k-2}, \cdots, y_1, y_0\}$,
$k = field\ size$
Precomputed: $BASE = 2^k - Y$
Output : Q = $\frac{X}{Y}$'s quotient

01 : $A := X$
02 : $Q := Q + \{a_{2k+3}, a_{2k+2}, \cdots, a_k\}$
03 : *if* $(k = 384\ or\ k = 521)$ *then*
04 : $t := \lceil (k+4)/2 \rceil$
05 : $A' := \{a_{2k+3}, a_{2k+2}, \cdots, a_k\} = \{a'_{l-1}, a'_{l-2}, \cdots, a'_0\}$
06 : $B_0 := \{a'_{t-1}, a'_{t-2}, \cdots, a'_0\} \times \{base_{t-1}, base_{t-2}, \cdots, base_0\}$
07 : $B_1 := \{a'_{l-1}, a'_{l-2}, \cdots, a'_t\} \times \{base_{t-1}, base_{t-2}, \cdots, base_0\}$
08 : $B_2 := \{a'_{t-1}, a'_{t-2}, \cdots, a'_0\} \times \{base_{k-1}, base_{k-2}, \cdots, base_t\}$
09 : $B_3 := \{a'_{l-1}, a'_{l-2}, \cdots, a'_t\} \times \{base_{k-1}, base_{k-2}, \cdots, base_t\}$
10 : $B := B_3 \times 2^{2t} + B_2 \times 2^t + B_1 \times 2^t + B_0$
11 : else then
12 : B := $\{a_{2k+3}, a_{2k+2}, \cdots, a_k\} \times BASE$
13 : A := $B + \{a_{k-1}, a_{k-2}, \cdots, a_0\}$
14 : *if* $(\{a_{2k+3}, a_{2k+2}, \cdots, a_k\} > 0)$ *then*
15 : ***go to*** step 2
16 : else then
17 : ***go to*** step 18
18 : *if* $(A \geq Y)$ *then*
19 : A := $A - Y$
20 : $Q := Q + 1$
21 : ***go to*** step 24
22 : else then
23 : ***go to*** step 24
24 : ***Return*** Q

Fig. 2. Pseudocode for Nikhilam division algorithm.
그림 2. Nikhilam 나눗셈 알고리듬의 슈도코드

III. 고속 모듈러 곱셈기 설계

3.1. 모듈러 곱셈기 구조

Ⅱ장에서 설명된 그림 1과 그림 2의 슈도코드를 기반으로 NIST FIPS 186-2[11]에 정의된 $GF(p)$ 상의 5가지 체 크기(192, 224, 256, 384, 521 비트)를 지원하는 모듈러 곱셈기 코어 (ModMul)를 설계하였다. 그림 1과 그림 2의 슈도코드를 하드웨어로 구현함에 있어서 정수 곱셈기가 가장 많은 하드웨어 자원을 사용하므로, 연산성능과 하드웨어 복잡도 사이의 관계를 고려하여 정수 곱셈기를 264 비트로 구현하였다. 설계된 ModMul의 내부 구조는 그림 3과 같으며, 264-비트 정수 곱셈기(Mul_264b), 640-비트 및 1280-비트 캐리선택 가산기(CSA_640b, CSA_1280b), 중간 결과값 저장 레지스터, 그리고 제어블록(Cntl_FSM) 등으로 구성된다. 정수 곱셈기는 그림 1과 그림 2의 슈도코드에서 기호 ×로 표시된 곱셈 연산을 수행하며, 캐리선택 가산기는 슈도코드에서 기호 +로 표시된 가산 연산을 수행한다. 가산기 CSA_1280b는 나눗셈 연산과정에서 몫에 대한 중간결과 가산에 사용되며, 그 이외의 가산 혹은 감산 연산은 CSA_640b에서 연산된다.

설계된 ModMul 코어는 내부 곱셈기와 가산기의 비트 수가 커서 회로의 동작속도가 느려질 수 있으며, 내부의 최악경로 지연이 크면 회로 합성 과정에서 하드웨어 자원 사용이 증가한다. 본 논문에서는 그림 3과 같이 곱셈기와 가산기 사이에 파이프라인 레지스터 (mul_reg, add_reg)를 삽입하여 최

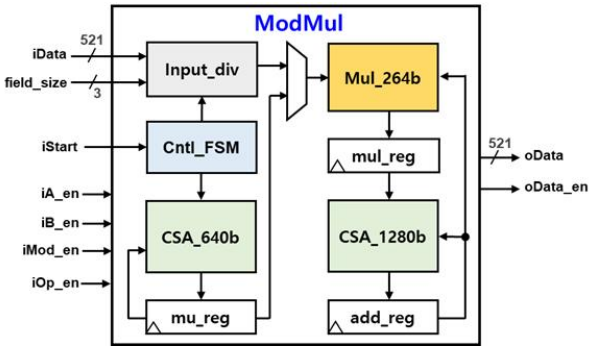


Fig. 3. Architecture of modular multiplier ModMul.
그림 3. 모듈러 곱셈기 ModMul의 구조

악경로 지연을 줄임으로써 동작속도 향상과 동시에 하드웨어 증가를 최소화시켰다.

3.2. 모듈러 곱셈 연산과정

ModMul 코어에서 모듈러 곱셈은 그림 4의 과정으로 연산된다. 모듈러 곱셈의 피연산자 X, Y와 모듈러스 값 M이 입력되면, 그림 1의 슈도코드에 사용되는 μ 값을 계산하는 초기화 단계가 실행된다. μ 값 계산을 위한 나눗셈 연산은 그림 2의 슈도코드에 의해 계산되며, 264-비트 정수 곱셈기 Mul_264b, 640-비트 캐리선택 가산기 CSA_640b와 1280-비트 캐리선택 가산기 CSA_1280b가 사용된다. μ 값 계산이 끝나면, 카라추바-오프만 곱셈 알고리듬에 의한 정수 곱셈 연산이 진행되며, 곱셈기 Mul_264b와 가산기 CSA_1280b가 사용된다. 정수 곱셈 연산이 완료되면, 곱셈 결과와 μ 값을 이용한 축약 연산을 거쳐 최종 모듈러 곱셈 결과 $T=(X\times Y)\bmod M$이 출력된다. 한편, 모듈러 곱셈 연산이 완료된 후, 모듈러스 값 M이 갱신되지 않은 상태로 새로운 피연산자 X, Y 값이 입력되는 경우에는 μ 값 계산 과정이 생략되도록 설계하였으며, 이를 통해 연속적인 모듈러 곱셈의 연산 성능이 향상되도록 하였다.

표 1은 NIST FIPS 186-2와 RFC 5639에 정의된

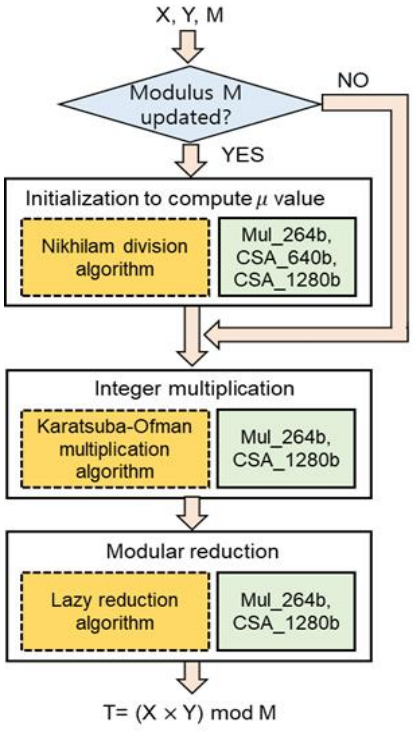


Fig. 4. Modular multiplication steps of ModMul.
그림4. ModMul의 모듈러 곱셈 과정

Table 1. Number of clock cycles used to compute μ value.
표 1. μ 값 계산에 소요되는 클럭 사이클 수

Field size (bits)		192	224	256	384	521
Initialization to compute μ (cycles)	Koblitz	7	7	7	-	-
	Pseudo-random	7	7	22	16	16
	Brainpool	194	175	335	2,026	-

모듈러스 값을 적용한 μ 값 계산에 소요되는 클록 사이클 수이다. Koblitz와 Pseudo-random 타원 곡선의 모듈러스 값은 비교적 큰 값을 가지므로, Brainpool 타원 곡선에 비해 μ 값 계산에 클록 사이클이 적게 소요된다. 체 크기가 384 비트, 521 비트인 경우에는 그림 1-(b)의 슈도코드와 같이 연산 과정이 더 늘어나 μ 값 계산에 소요되는 클록 사이클 수가 증가한다.

3.3. 카라추바-오프만 곱셈 알고리듬 구현

정수인 두 입력 데이터 X와 Y의 곱을 계산하기 위해 그림 1-(b) 슈도코드의 단계-1~단계-5을 기반으로 카라추바-오프만 곱셈기를 구현하였다. ModMul 코어 내부의 264-비트 곱셈기는 그림 1-(b) 슈도코드의 곱셈 연산을 위해 사용된다. 단, 단계-3~단계-5의 연산에서 2의 멱승 2^q을 곱하는 연산은 곱셈기 대신 단순 시프트 연산으로 계산되도록 하였다. 두 입력 X와 Y는 체 크기에 상관없이 264-비트의 상위 워드와 하위 워드로 나누어져 곱셈기 혹은 가산기에 입력되도록 설계하였으며, 이를 통해 체 크기에 따른 추가 회로를 구현할 필요가 없어 하드웨어 면적을 최소화하였다. 그림 1-(b) 슈도코드의 단계-3의 C_1 값을 구하는 연산은 식 (2)의 연산 중 하나로서 z_2B^{2q}와 z_0의 덧셈은 $m<n/2$인 경우, z_0가 (n+1) 비트 이상이 될 수 없다는 점을 이용하여 $\{P_{11},C\}$의 결합연산으로 구현하였다.

3.4. Lazy 축약과 Nikhilam 나눗셈 구현

모듈러 축약 연산은 Lazy 축약 알고리듬을 적용하여 하드웨어로 구현되었다. 체 크기가 256 비트 이하인 경우에는 그림 1-(a) 슈도코드의 단계-2~단계-6이 적용되며, 384 비트 이상인 경우에는 그림 1-(b) 슈도코드의 단계-6~단계-18이 적용된다. 설계된 ModMul 코어 내부의 곱셈기가 264-비트이므로, 체 크기가 384 비트 이상인 경우에는 피연산

자를 분할하여 축약 연산이 수행된다. 그림 1의 슈도코드에서 C와 s를 2의 멱승 $2^{n+\alpha}$ 또는 $2^{n+\beta}$로 나누는 과정은 두 값을 하위부터 $n+\alpha$ 또는 $n+\beta$ 비트만큼 분할한 후, 나머지 상위 비트의 값을 사용하는 방식으로 구현되었다. 체 크기가 384 비트 이상인 경우에는 s와 m을 구하기 위해 식 (2)와 식 (3)에 의한 곱셈과 가산 연산이 진행된다.

모듈러 축약 연산에 사용되는 μ 값을 계산하기 위해 그림 2 슈도코드 기반의 Nikhilam 나눗셈 알고리듬을 적용하여 나눗셈 연산 회로를 구현하였다. BASE 값 계산에 사용되는 2의 멱승 값은 외부로부터 입력받지 않고 룩-업 테이블을 이용하여 구현하였으며, 체 크기가 384-비트 이상인 경우에는 슈도코드의 B를 계산하기 위해 카라추바-오프만 곱셈 알고리듬을 적용하였다. 단계-18에서 제수와 중간 결과값 A의 비교 연산은 비교기를 사용하지 않고 다음과 같이 구현하였다. 제수가 r-비트라고 했을 때, k 비트의 A를 하위부터 r-비트만큼 분할한 후, 나머지 상위 (k-r) 비트의 값 R이 $R>1$이면 다음 단계로 넘어가 몫의 값을 출력하고, 만약 $R=0$이면 A의 하위 r-비트 S와 제수 D를 감산한 후, $S-D>0$이면 $S>D$이므로 몫 값 Q에 1을 더해주고, $S-D\leq 0$인 경우에는 그 다음 단계로 넘어가 몫의 값을 출력한다.

Ⅳ. RTL 기능검증 및 성능평가

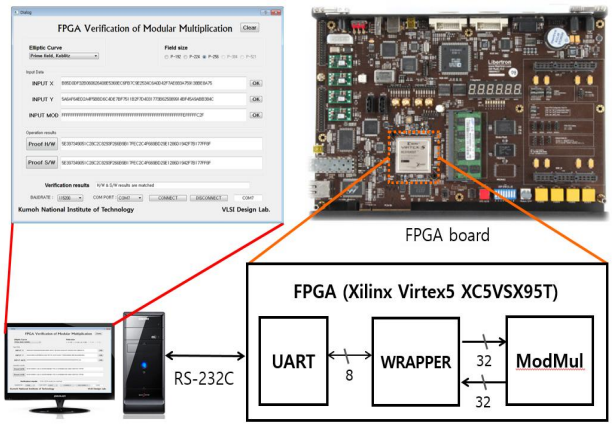
설계된 ModMul 코어를 Virtex-5 FPGA 디바이스에 구현하여 하드웨어 동작을 검증하였으며, FPGA 검증 시스템의 구성도는 그림 5-(a)와 같다. 그림 5-(b)는 FPGA 검증결과의 화면캡처이며, 체 크기 256 비트의 모듈러 곱셈 연산에 대한 동작을 보이고 있다. FPGA에 구현된 ModMul 코어에서 계산된 모듈러 곱셈 결과가 소프트웨어로 계산된 결과와 일치함을 확인할 수 있으며, 다른 체 크기에 대해서도 ModMul 코어가 정상 동작함을 확인하였다.

설계된 모듈러 곱셈기 코어 ModMul을 180-nm CMOS 셀 라이브러리로 합성한 결과, 67 MHz의 동작 주파수에서 456,400 등가 게이트로 구현되었으며, 체 크기에 따라 모듈러 곱셈 연산에 소요되는 클록 사이클 수는 표 2와 같다. 본 논문의 ModMul 코어는 내부의 정수 곱셈기와 가산기 크기가 256 비트 이하의 체 크기를 지원하도록 설계

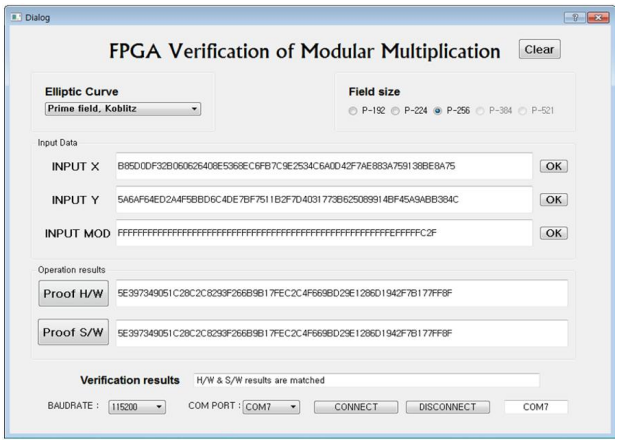
Table 2. Number of clock cycles required to calculate modular multiplication
표 2. 모듈러 곱셈 계산에 소요되는 클록 사이클 수

Field Size (bits)	192	224	256	384	521
Integer multiplication (cycles)	1	1	1	5	5
Modular reduction (cycles)	4	4	4	12	12
Total (cycles)	5	5	5	17	17

되어 체 크기가 384 비트, 521 비트인 경우에는 피연산자를 분할하여 연산하므로 소요 클록 사이클 수가 3배 정도 증가한다. 표 3은 문헌에 발표된 모듈러 곱셈기와의 비교를 보이고 있다. 문헌 [14]의 사례는 본 논문의 ModMul 코어에 비해 사용된 LUT 수가 적지만 모듈러 곱셈에 약 4.9 배의 시간이 소요되어 연산성능이 낮다. 문헌 [15]의 사례는 본 논문의 ModMul 코어 보다 연산 소요시간이 적



(a) FPGA verification setup



(b) Screenshot of FPGA verification (field size=256 bits)

Fig. 5. FPGA verification results of ModMul.
그림 5. ModMul의 FPGA 검증 결과

지만 약 7.3배의 LUT를 사용하여 하드웨어 복잡도가 크다. 또한, 문헌 [14], [15]의 모듈러 곱셈기는 256-비트 이하의 체 크기만 지원하나, 본 논문의 ModMul 코어는 384 비트와 521 비트를 지원하므로 다양한 분야에 적용될 수 있다는 장점이 있다.

Table 3. Comparison of modular multipliers implemented on FPGA.

표 3. FPGA에 구현된 모듈러 곱셈기 비교

	FPGA Platform	Field Size (bits)	LUTs	Freq (MHz)	Latency (ns)
[14]	Virtex-6	256	5.3k	166	770
[15]	Virtex-6	256	187.9k	68	14.7
This paper	Virtex-6	256	25.6k	32	156
		521			532

V. 결론

소수체 상의 모듈러 곱셈 연산을 고속 하드웨어로 구현하기 위해 Nikhilam 나눗셈 알고리듬, 카라추바-오프만 곱셈 알고리듬 그리고 Lazy 축약 알고리듬을 적용한 고성능 모듈러 곱셈기 설계 방법을 제안하고, 하드웨어를 설계하였다. 설계된 모듈러 곱셈기 코어 ModMul는 Virtex-6 FPGA 디바이스에서 약 25.6 kLUTs로 구현되었으며, 32 MHz의 클록 주파수로 동작하여 초당 640만 번의 모듈러 곱셈을 연산할 수 있는 것으로 평가되었다. 본 논문의 모듈러 곱셈기 코어는 타원곡선 암호의 고성능 하드웨어 구현에 핵심 연산장치로 사용될 수 있다.

References

[1] N. Koblitz, “Elliptic curve cryptosystems,” *Mathematics of Computation*, vol.48, no.177, pp. 203-209, Jan. 1987.

[2] D. Basu Roy and D. Mukhopadhyay, “High-Speed Implementation of ECC Scalar Multiplication in GF(p) for Generic Montgomery Curves,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.27, no.7, pp.1587-1600, July 2019. DOI: 10.1109/ TVLSI.2019.2905899.

[3] M. R. Hossain and M. S. Hossain, “Efficient FPGA Implementation of Modular Arithmetic for Elliptic Curve Cryptography,” *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, Cox’sBazar, Bangladesh, pp.1-6, 2019. DOI: 10.1109/ECACE.2019.8679419.

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

[5] A. Karatsuba and Y. Ofman, “Multiplication of many-digital numbers by automatic computers,” *Proceedings of the USSR Academy of Sciences*, vol.145, no.2, pp.293-294, 1962.

[6] J.-C. Bajard, L.-S. Didier and P. Kornerup, “An RNS Montgomery modular multipli-cation algorithm,” in *IEEE Transactions on Computers*, vol.47, no.7, pp.766-776, July 1998.
DOI: 10.1109/ 12.709376.

[7] P. Barrett, “Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor,” *In: Odlyzko A.M. (eds) Advances in Cryptology-CRYPTO’86. Lecture Notes in Computer Science*, Springer, vol.263, pp.311-323, Aug. 1986.
DOI: 10.1007/3-540-47721-7_24

[8] M. M. Islam, M. S. Hossain, M. Shahjalal, M. K. Hasan and Y. M. Jang, “Area-Time Efficient Hardware Implementation of Modular Multiplication for Elliptic Curve Cryptography,” *IEEE Access*, vol.8, pp.73898-73906, 2020.
DOI: 10.1109/ACCESS.2020.2988379.

[9] E. Öztürk, “Design and Implementation of a Low-Latency Modular Multiplication Algorithm,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol.67, no.6, pp.1902-1911, June 2020. DOI: 10.1109/TCSI.2020.2966755.

[10] S. Li and Z. Gu, “Lazy Reduction and Multi-Precision Division Based on Modular Reductions,” *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, Chengdu, 2018, pp.407-410.

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

[12] X. Feng and S. Li, "A high performance fpga implementation of 256-bit elliptic curve cryptography processor over GF(p)," *IEICE Transactions on Fundamentals of Electronics Communications & Computer Sciences*, vol. E98.A, no.3, pp.863-869, 2015.
[13] J. S. S. B. K. T. Maharaja, *Vedic Mathematics*, MotilalBanarsidass, New Delhi, India, 1994.
[14] K. Javeed, X. Wang and M. Scott, "Serial and parallel interleaved modular multipliers on FPGA paltform," *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pp.1-4, Sept 2015.
[15] R. Liu and S. Li, "A design and implementation of Montgomery modular multiplier," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp.1-4, May 2019.

BIOGRAPHY

Jun-Yeong Choe (Member)

2019 : BS degree in Electronic Engineering, Kumoh National Institute of Technology.
2019~ : Graduate student, Kumoh National Institute of Technology

Kyung-Wook Shin (Member)

1984 : BS degree in Electronic Engineering, Korea Aerospace University
1986 : MS degree in Electronic Engineering, Yonsei University
1990 : Ph.D. degree in Electronic Engineering, Yonsei University
1990~1991 : Senior Researcher, Semiconductor Research Center, Electronics and Telecommunications Research Institute (ETRI)
1991~Present : Professor in School of Electronic Engineering, Kumoh National Institute of Technology
1995~1996 : University of Illinois at Urbana-Champaign (Visiting Professor)
2003~2004 : University of California at San Diego (Visiting Professor)
2013~2014 : Georgia Institute of Technology (Visiting Professor)