Journal of Korea Multimedia Society Vol. 20, No. 2, February 2017(pp. 289-297)
https://doi.org/10.9717/kmms.2017.20.2.289

GF(p)의 타원곡선 암호 시스템을 위한 효율적인 하드웨어 몽고메리 모듈러 역원기

최필주†, 김동규††

Efficient Hardware Montgomery Modular Inverse Module for Elliptic Curve Cryptosystem in GF(p)

Piljoo Choi†, Dong Kyue Kim††

ABSTRACT

When implementing a hardware elliptic curve cryptosystem (ECC) module, the efficient design of Modular Inverse (MI) algorithm is especially important since it requires much more computation than other finite field operations in ECC. Among the MI algorithms, binary Right-Shift modular inverse (RS) algorithm has good performance when implemented in hardware, but Montgomery Modular Inverse (MMI) algorithm is not considered in [1, 2]. Since MMI has a similar structure to that of RS, we show that the area-improvement idea that is applied to RS is applicable to MMI, and that we can improve the speed of MMI. We designed area- and speed-improved MMI variants as hardware modules and analyzed their performance.

Key words: Elliptic Curve Cryptosystem, Modular Inversion, Montgomery Inversion, Finite Field Operation

1. 서 론

제품과 기기들이 지능화되는 4차 산업 혁명에서는 보안이 더 중요해질 것으로 예상된다. 그러나 기존의 서버나 광통신 등의 환경과 달리 계산 능력과 무선 통신 전송량은 제한되어 있기 때문에 앞으로 더 짧은 키를 사용하면서도 비슷한 암호학적 강도[3, 4]를 갖는 타원 곡선 암호 시스템(ECC, Elliptic Curve Cryptosystem)의 사용이 늘어날 것으로 예상된다.

최근 요구되는 보안 강도에 따른 ECC의 키 길이는 256-bit로, 워드 길이인 32-bit 또는 64-bit보다 커 소프트웨어로 구현 시 워드 단위로 나누어 계산해야 한다. 이는 계산 능력이 뛰어난 서버급 환경에서는 큰 문제가 되지 않으나, 자원이 제한된 임베디드 시스템 환경의 경우 ECC 연산은 큰 부담으로 작용할 수 있다. 이와 달리 ECC를 하드웨어로 구현할 경우 ECC의 알고리즘 특성에 맞춘 전용 모듈로 구현할 수 있어 계산 속도를 크게 증가시킬 수 있다.

ECC도 다른 공개키 암호 알고리즘과 마찬가지로 유한체(finite field) 상에서의 연산인 모듈러 덧셈(MA, Modular Addition), 모듈러 역(MI, Modular Inversion), 모듈러 곱셈(MM, Modular Multiplication)을 사용한다. 이 중 MI는 다른 대표적인 공개키

※ Corresponding Author : Dong Kyue Kim, Address: (04763) Wangsimni-ro 222, Seongdong-gu, Seoul, Korea, TEL : +82-2-2220-2312, FAX : +82-2-2220-4926, E-mail : dqkim@hanyang.ac.kr
Receipt date : Jan 13, 2017, Approval date : Jan. 31, 2017
† Dept. of Electronic Eng., Hanyang University
(E-mail: pjchoi@hanyang.ac.kr)
†† Dept. of Electronic Eng., Hanyang University
※ This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2016-H8601-16-1005) supervised by the IITP(Institute for Information & communications Technology Promotion)

암호 알고리즘인 RSA에서는 거의 사용되지 않으나 ECC에서는 빈번하게 사용되며, MA나 MM에 비해 계산량이 많아 ECC 전체 성능에 큰 영향을 미친다. 따라서 ECC의 하드웨어 설계 시 MI의 효율적인 구현이 매우 중요하다.

MI의 하드웨어 구현에 대한 기존 연구[2]에서는 MI 알고리즘인 유클리드 모듈러 역(EM, Euclidean Modular inverse)[5], 이진 RS 모듈러 역(RS, binary Right-Shift modular inverse)[6, 7, 8], 이진 LS(LS, binary Left-Shift modular inverse)[9, 10] 중에서 RS가 가장 하드웨어 구현 시 가장 속도가 빠르다고 말하고 있다. 다만 하드웨어 구현 시 면적이 상대적으로 크다는 단점이 있으나 [1]에서 제시한 방법을 통해 면적을 크게 줄일 수 있다. 그러나 이들 논문에서 몽고메리 모듈러 역(MMI, Montgomery Modular Inverse) 알고리즘[11, 12, 13]은 고려되어 있지 않다. MMI는 MM을 계산하기 위해 몽고메리 곱(Montgomery product)를 사용할 때 함께 많이 사용하며, RS와 구조가 비슷하기 때문에 RS의 면적 개선 방법을 동일하게 적용할 수 있다.

본 논문에서는 [1]의 면적 개선 방법을 MMI에 적용하고, 속도도 같이 개선하여 기존의 MMI보다 면적과 처리 속도 면에서 효율이 더 좋은 하드웨어 MMI 모듈을 제시하고자 한다. 본 논문의 구성은 다음과 같다. 2장에서 본 논문의 효과적인 하드웨어 MI 모듈의 필요성에 대한 기본 이론으로 ECC와 하드웨어 설계 시 ECC 유한체 연산의 특징에 대해 살펴본다. 3장에서는 기존의 MI 알고리즘에 대해서 RS와 MMI에 초점을 맞춰 알아보며, 4장에서는 제안하는 MMI를 설명한다. 5장에서는 하드웨어로 설계한 MI 모듈들의 구조를 보여주고 성능을 평가하도록 한다. 마지막으로 6장에서 결론을 맺는다.

2. Preliminary

기본 이론으로써 ECC의 기본적인 내용에 대해 설명한 후 ECC를 하드웨어로 설계할 때 유한체 연산의 특징을 살펴본다.

2.1 타원 곡선 암호 알고리즘과 P-256 타원 곡선

p가 소수일 때 유한체 GF(p) 상에서의 타원 곡선은 $y^2 \pmod p = x^3 + ax + b \pmod p$를 만족하는 (x, y)점들의 집합과 항등원인 무한대 점 O를 의미한다. 많이 사용되는 표준 타원 곡선인 [14]의 P-256을 사용할 때 $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, $a = -3$, b는 16진수 값 5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B를 갖는다. 타원 곡선 상에서 두 점에 대한 연산은 포인트 합(PA, Point Addition), 포인트 승(PD, Point Doubling)으로 나뉘며, 이는 Fig. 1에 나타나 있다.

PA는 서로 다른 두 점 간의 덧셈으로, 두 점을 지나는 직선이 타원 곡선과 만나는 다른 한 점을 구한 후 이를 다시 x축 대칭하여 구할 수 있다. PD는 서로 같은 두 점 간의 덧셈으로, 이 두 점에서 타원 곡선의 접선이 타원 곡선과 만나는 다른 한 점을 구한 후 이를 다시 x축 대칭하여 구할 수 있다. 이러한 PA와 PD를 반복하여 포인트 곱(PM, Point Multiplication)을 수행할 수 있으며, PM을 수행하여 ECC 기반 암호 프로토콜인 ECDH[15], ECDSA[16] 등을 수행할 수 있다.

2.2 하드웨어 설계 시 ECC의 유한체 연산의 특징

하드웨어 설계 시엔 병렬 처리가 가능할 뿐만 아니라 연산 단위를 키 길이에 맞춰 설계하여 ECC 모듈의 구조를 더 단순화할 수 있다. 연산 단위가 256-

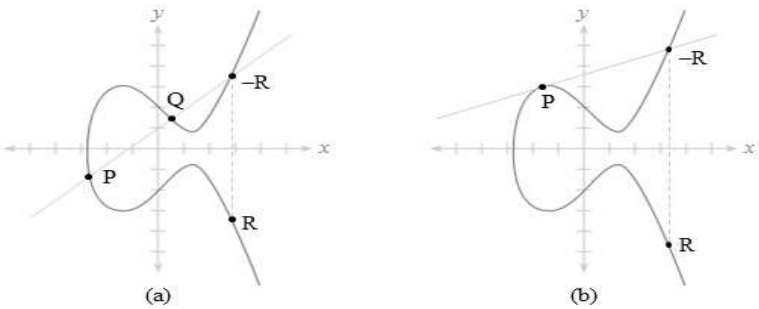

Fig. 1. EC Point operations. (a) EC Point Addition (PA) and (b) EC Point Doubling (PD).

bit가 되도록 설계하더라도 100~200MHz에서 동작이 가능할 뿐만 아니라, 임베디드 시스템의 경우 동작 주파수는 100MHz보다 훨씬 낮은 편이다. 따라서 본 논문에서는 연산 단위가 키 길이와 동일한 설계를 기본으로 한다.

연산 단위를 키 길이와 동일하도록 설계하더라도 MM과 MI는 계산량이 많아 보통 수십~수백 싸이클이 소모된다. 보통 ECC 알고리즘에서 사용되는 modulus의 값은, 나머지만 취하는 reduction 연산에 용이하도록 설정되어 있어, 빠른 MM 연산이 가능할 뿐만 아니라, 몽고메리 곱(Montgomery product)와 같은 빠르게 MM을 수행하는 알고리즘도 존재한다. 이에 반해 MI의 경우 modulus와 divisor의 최대 공약수(GCD, Great Common Divisor)에 기반을 둔 알고리즘이 보통 사용되나 modulus의 형태에 상관없이, MM에 비해 속도가 매우 느리다. 이러한 이유로 MI를 사용하지 않아도 되는 좌표계인 Jacobian coordinates 등의 사영 좌표계(projective coordinates) [17]가 많이 사용된다. 그러나 MI가 제거된 대신 더 많은 MM가 필요하고 수식이 복잡해짐에 따라 이를 구현할 때 면적이 늘어난다. 또한 사영 좌표계에서 마지막에 원래 좌표계인 아핀 좌표계(affine coordinates)로 변환하는 과정에서 MI가 필요하며, ECDSA 서명 생성과 검증 시에도 MI가 필요하다. 따라서 MI는 ECC 하드웨어 모듈의 필수 하위 모듈이며, 만약 빠른 속도의 MI를 구현할 수 있다면, 사영 좌표계의 사용 없이 간단한 구조의 ECC를 설계할 수 있다.

3. 기존의 모듈러 역 알고리즘

기존의 GCD 기반의 모듈러 역 알고리즘 중에서 하드웨어 설계 시 성능이 우수한 이진 RS 모듈러 역(RS, binary Right-Shift modular inverse) 알고리즘에 대해 살펴보고, 비슷한 구조의 몽고메리 모듈러 역 알고리즘(MMI, Montgomery Modular Inverse algorithm)도 함께 살펴본다.

3.1 이진 RS 모듈러 역 알고리즘의 특징

RS는 EM과 LS와는 다른 두 가지 특징이 있다. 첫 번째는 두 연산 중 하나를 선택하는 것이 그 중 한 연산의 결과에 의해 결정된다는 것이다. RS 내 두 가지 변수 U와 V간 연산인 $U-V$와 $V-U$의 선택이 $U > V$에 의해 결정되는 것이 이에 속한다. 이 경우 소모 클럭 싸이클 수를 늘리지 않기 위해서는 두 연산을 동시에 수행해야 하며, 이는 덧셈기 증가로 이어진다. 두 번째 특징은 홀수를 right-shift하기 위해 modulus m을 더하는 $+m$ 연산이다. 이 두 가지 특징으로 인해 소프트웨어 구현 시 RS가 느린 편이나 하드웨어 설계 시엔 덧셈기를 추가하여 병렬 처리하면 소모 클럭 싸이클 수가 감소되며, 이 경우 오히려 RS 알고리즘 특성으로 인해 EM이나 LS에 비해 속도가 더 빠르다[2]. 다만 추가한 덧셈기들로 인해 면적이 크게 늘어나나 [1]에서 적용한 방법으로 상대적으로 늘어난 면적의 상당부분을 줄일 수 있다.

3.2 이진 RS 모듈러 역 알고리즘의 하드웨어 설계 최적화

Algorithm 1은 RS와 RS의 면적을 개선한 알고리즘으로, 음영 표시의 왼쪽은 원래 RS를, 오른쪽은 RS의 면적을 개선한 RS variant를 나타낸다. 나머지 부분은 공통 사용 부분이다. Algorithm 1의 while문의 루프 불변자(loop invariant)는

$$aR \equiv U \pmod{m},\ aS \equiv V \pmod{m} \qquad (1)$$

이다. Algorithm 1의 첫 번째 줄에서 R, S, U, V는 각각 0, 1, m, a로 초기화되며, while문 내에서 right-shift와 뺄셈을 통해 U와 V에 저장된 값이 지속적으로 감소된다. $V = 0$일 때 U의 값이 0이면 $a^{-1} \pmod{m}$이 존재하지 않으며, U의 값이 1이면 $aR \equiv 1 \pmod{m}$이 되어, R에 저장되어 있는 값이 $a^{-1} \pmod{m}$이 된다. ECC 알고리즘에서 modulus m은 항상 소수이므로 역원이 존재하지 않는 경우는 없다. 본 논문에서는 역원이 존재하는 경우만 다루도록 한다.

앞서 설명한 바와 같이 RS는 14째 줄의 $U > V$의 결과에 따라 $U-V$와 $V-U$의 선택이 달라진다. $U > V$ 연산은 결국 $U-V > 0$이므로 $U-V$ 연산은 기본적으로 수행되며, 그 결과가 음수일 경우 $V-U$를 위한 추가 연산이 필요하다. 이들은 서로 덧셈의 역원 관계로 $V-U = -(U-V)$로도 계산이 가능하나, 2의 보수를 계산하는 과정 역시 덧셈기가 필요하다. 별도의 클럭 싸이클을 소모하지 않으려면 $U-V$와 $V-U$ 각각에 덧셈기가 하나씩 필요하다. $R-S$와 $S-R$ 연산을 수행하는 것도 마찬가지 이유로 2개의 덧셈기가 필요하다. 그리고 홀수 R과 S를 right-

Algorithm 1: Right-Shift binary inversion algorithm

Input: modulus m, divisor a.
Output: $a^{-1} \pmod m$.

```
1:  U ← m; V ← a; R ← 0; S ← 1;                    ⇒  U ← -m; V ← a; R ← 0; S ← 1;
2:  while (V > 0)
3:      if (U0 = 0)                                    // even U
4:          U ← U/2;                                   // halving U
5:          if (R0=0)       R ← R/2;                   // halving even R
6:          else            R ← (R + m)/2;             // halving odd R
7:      else if (V0 = 0)                               // even V
8:          V ← V/2;                                   // halving V
9:          if (S0=0)       S ← S/2;                   // halving even S
10:         else            S ← (S + m)/2;             // halving odd S
11:     else                                           // odd U and odd V
12:
13:
14:         if (U > V)                                 VU ← U + V;
15:             U ← U - V;                             SR ← R + S;
16:             R ← R - S;                             if (VU < 0)
17:             if (R < 0) R ← R + m;                      U ← VU;
18:         else                                           R ← SR;
19:             V ← V - U;                             else
20:             S ← S - R;                                 V ← VU;
21:             If (S < 0) S ← S + m;                      S ← SR;
22: if (U ≠ 1)      return 0;
23: if (R > m)      R ← R - m;
24: return R;
```

shift하기 위한 $+m$ 연산(6번째 줄과 10번째 줄)까지 고려하면 소모 클럭 싸이클 수와 최상 경로(critical path)를 최소화하면서 RS를 하드웨어로 설계하는 것은 적어도 5~6개의 덧셈기가 필요하다.

[1]에서는 간단한 수식의 수정만으로 요구되는 덧셈기의 수를 반으로 줄였으며 달라진 주요 부분을 Algorithm 1의 첫 번째 줄과 12~21번째 줄(음영 표시)의 오른쪽에 표현하였다. 변경된 알고리즘에서는 변수 U와 R에 $-U$와 $-R$을 저장한다. 이를 위해 U가 m이 아닌 $-m$으로 초기화되며, 초기 값이 0인 R에서는 아무런 변화가 없다. 변경 결과 덧셈 연산만 존재하게 되어, $U+V$, $R+S$에 각 하나, 홀수 R과 S에 대한 $+m$ (공용 사용)을 위해 하나, 총 3개의 덧셈기가 필요하다.

3.3 기존의 몽고메리 모듈러 역 알고리즘

몽고메리 모듈러 역(MMI, Montgomery Modular Inverse) 알고리즘은 RS와 비슷한 구조를 가지고 있어 앞서 RS에 적용한 면적 개선 방법을 MMI에도 적용할 수 있다. 엄밀히 말하면 총 두 단계로 수행되는 MMI의 첫 번째 단계인 almost Montgomery modular inverse (AMMI) 알고리즘에 적용할 수 있다. AMMI에서는 $a^{-1} \cdot 2^k \pmod m$을 수행하며, 자세한 알고리즘은 다음과 같다.

AMMI는 (1)과 다른 루프 불변자를 가지며 다음과 같다.

$$aR \cdot 2^{-k} \equiv U \pmod m,\ aS \cdot 2^{-k} \equiv V \pmod m \quad (2)$$

(2)에 의해 $U=1$일 때 $aR \cdot 2^{-k} \equiv 1$이 되어, R에 결과물인 $a^{-1} \cdot 2^k \pmod m$가 저장된다. 이 값은 몽고메리 모듈러 역 알고리즘의 두 번째 단계에서 $a^{-1} \pmod m$로 변환된다. 이 때 m을 더해가며 bit 단위로 right-shift를 반복하거나, 몽고메리 곱(Montgomery product) 모듈이 있을 경우 2~3번의 몽고메

리 곱 연산을 수행하여 처리할 수 있다[13].

4. 제안하는 almost Montgomery modular inverse 알고리즘

이번 장에서는 앞서 살펴본 몽고메리 모듈러 역 알고리즘의 첫 번째 단계인 AMMI의 면적과 속도를 개선한 알고리즘에 대해 설명한다.

4.1 면적을 개선한 almost Montgomery modular inverse 알고리즘 (AMMI variant-1)

Algorithm 2를 살펴보면 R과 S간의 연산은 덧셈이기 때문에 $U > V$의 영향을 받지 않으나 U와 V간의 연산은 여전히 서로 뺄셈 관계이다. 따라서 앞서 RS에서 적용했던 방법을 적용하여 덧셈기의 수를 줄일 수 있다.

Algorithm 3을 살펴보면 $U+V$, $R+S$에 각각 하나씩, 총 2개의 덧셈기만 사용됨을 알 수 있다. 속도면에서, RS와 AMMI에서의 U, V간 연산 방법은 동일하므로 소모 클럭 싸이클 수는 동일하다. 다만 while문의 탈출 조건을 기존의 $V = 0$에서 $V \leq 1$로 바꾸어, V가 1일 때 U도 1이 되길 기다렸다가 $V = V - U = 0$을 수행하는 불필요한 싸이클을 제거하였다. 이 변경으로 인해 $k < n$이 되는 상황이 발생할 수 있으나 이는 13째 줄의 두 번째 while문을 통해 해결된다.

Algorithm 2: Almost Montgomery Modular Inverse [11]

Input: modulus m, divisor a.
Output: $a^{-1} \cdot 2^k \pmod m$. $(n \leq k \leq 2n)$

```
1:  U ← m; V ← a; R ← 0; S ← 1;
2:  k ← 0
3   while (V > 0)
4:      if (U0=0)                U ← U/2;          S ← 2S;
5:      else if (V0=0)           V ← V/2;          R ← 2R;
6:      else if (U > V)          U ← (U - V)/2;    S ← 2S;     R← R + S;
7:      else                     V ← (V - U)/2;    R ← 2R;     S← R + S;
8:      k ← k + 1
9:  if (U ≠ 1)        return  0;
10: if (R > m)        R ← R - m;
11: return R and k;
```

Algorithm 3: Almost Montgomery Modular Inverse variant-1 (AMMI variant-1)

Input: modulus m, divisor a.
Output: $a^{-1} \cdot 2^k \pmod m$. $(n \leq k \leq 2n)$

```
1:  U ← - m; V ← a; R ← 0; S ← 1;
2:  k ← 0
3   while (V > 1)
4:      if (V0=1)                TU ← U;           TR ← R;
5:      else                     TU ← 0;           TR ← 0;
6:      if (U0=1)                TV ← V            TS ← S;
7:      else                     TV ← 0;           TS ← 0;
8:      VU ← TU + TV;
9:      SR ← TR + TS;
10:     if (VU < 0)              U ← VU/2;         S ← 2S;     R ← SR;
11:     else                     V ← VU/2;         R ← 2R;     S ← SR;
12:     k ← k + 1;
13: while (k < n)                S ← 2S;           k ← k + 1;
14: if (V = 0)        return 0;
15: return S and k;
```

4.2 속도를 개선한 almost Montgomery modular inverse 알고리즘 (AMMI variant-2, 3)

소모 클럭 싸이클 수를 더 줄이기 위하여 Algorithm 3의 8번째 줄에서 VU의 최하위 2-bit이 0이면 1-bit shift 대신 2-bit shift를 수행하도록(AMMI variant-2), VU의 최하위 3-bit이 0이면 3-bit shift 하도록(AMMI variant-3) 수정할 수 있다.

Table 1은 Algorithm 3의 10~12번째 줄(음영 표시 부분)이 AMMI variant-2와 AMMI variant-3에서 어떻게 바뀌는지를 나타낸다. Table 1대로 설계 시 multiplexer만 추가될 뿐이기에 이로 인한 면적 및 최상 경로 길이의 증가는 크지 않을 것으로 예상된다. RS에도 비슷한 방법을 적용 가능하나, 홀수 R과 S를 right-shift하기 위한 부분 때문에 AMMI variant-2와 AMMI variant-3처럼 간단한 수정은 불가능하다. 2-bit shift까지는 덧셈기가 추가되지 않는 설계가 가능하므로 RS에서는 AMMI와 달리 최대 2-bit shift 정도가 적당하다.

5. MI 모듈의 구현 결과

3, 4장에서 살펴본 RS, RS variant와 AMMI, AMMI variant-1~3을 하드웨어로 설계하였다. 이 장에서는 이들의 구조와 성능, 즉 면적과 처리속도를 살펴보도록 한다.

5.1 하드웨어 모듈의 구조

RS, RS variant와 AMMI, AMMI variant-1을 하드웨어로 구현하였을 때 구조는 Fig. 2와 같다. Fig. 2에 포함되어 있지 않은 AMMI variant-2와 AMMI

Table 1. AMMI variant-2 and AMMI variant-3

Algorithm	Codes that replace the codes in line 10-12 of Algorithm 3

AMMI variant-2	**if** ($UV_{1:0}$ = 0) // $UV_{1:0}$: two least significant bits of UV
	  if (VU < 0) $U \leftarrow VU/4$; $S \leftarrow 4S$; $R \leftarrow SR$;
	  else $V \leftarrow VU/4$; $R \leftarrow 4R$; $S \leftarrow SR$;
	  $k \leftarrow k + 2$;
	else *codes in line 10-12 of Algorithm 3*;

AMMI variant-3	**if** ($UV_{2:0}$ = 0) // $UV_{2:0}$: three least significant bits of UV
	  if (VU < 0) $U \leftarrow VU/8$; $S \leftarrow 8S$; $R \leftarrow SR$;
	  else $V \leftarrow VU/8$; $R \leftarrow 8R$; $S \leftarrow SR$;
	  $k \leftarrow k + 3$;
	else *AMMI varian-2's replacement codes*;

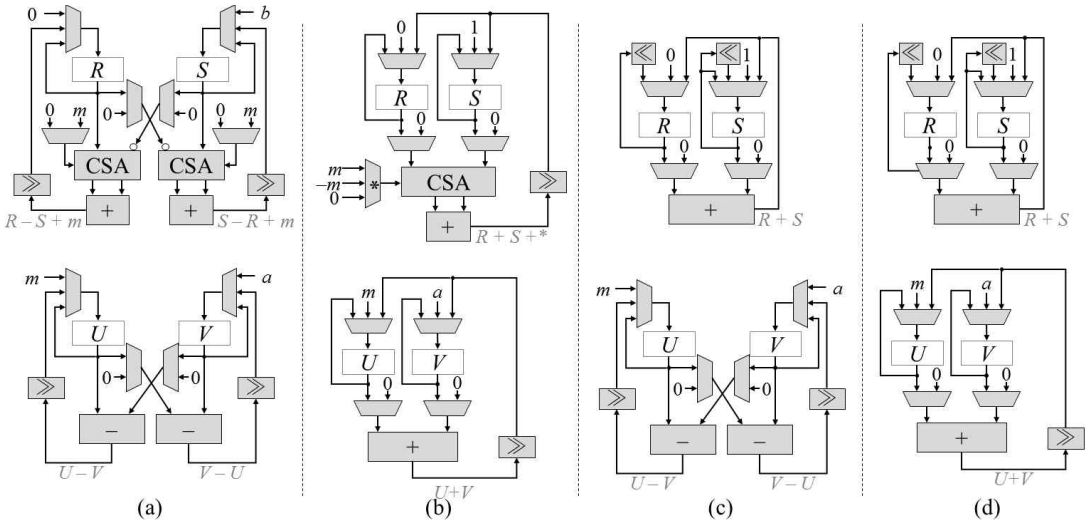

Fig. 2. Structures of hardware modular inversion modules. (a) RS, (b) RS variant, (c) AMMI, and (d) AMMI variant-1.

Table 2. Performance comparison of MI algorithms

Algorithm		Area (GE) @ 100MHz	Max. Freq.	# of clock cycles		
				128-bit	256-bit	n-bit
RS	Algorithm 1	32,541	200MHz	180.0	360.7	1.41n
RS variant	-	23,405	222MHz			
AMMI	Algorithm 2	22,622	250MHz	179.4	360.2	$1.40n \sim 1.41n$
AMMI variant-1	Algorithm 3	19,378	250MHz			
AMMI variant-2	-	21,295	250MHz	120.0	240.6	$0.94n$
AMMI variant-3	-	22,345	222MHz	103.2	206.5	0.86n

variant-3은 AMMI variant-1의 1 bit shift부분(≪, ≫)에 2-bit shift, 3-bit shift도 선택할 수 있도록 multiplexer가 추가된다. 모두 소모 클럭 싸이클 수를 최소화하기 위해 충분한 수의 덧셈기가 사용되었으며, 덧셈 또는 뺄셈 이후 수행되는 shift는 덧셈 또는 뺄셈과 동일한 클럭 싸이클에 수행되도록 설계되었다. RS에서 사용된 CSA(carry save adder)[18]는 1비트 전 가산(1-bit full addition)의 지연만으로 세 수의 덧셈을 두 수의 덧셈으로 바꿔주는 역할을 수행한다. CSA를 포함하였을 때 RS variant가 필요한 덧셈기의 개수는 RS에 비해 1/2인 3개로, AMMI variant-1는 AMMI에 비해 덧셈기가 하나 더 적은 2개로 줄었다. 이와 함께 제어 회로도 더 단순해져 면적이 감소할 것으로 예상된다.

5.2 처리 속도와 면적 비교

RS, RS variant와 AMMI, AMMI variant-1~3을 0.18μm 공정을 사용하여 합성하였으며, 이들의 속도와 면적은 Table 2에 나타나 있다. Table 2에서 최대 클럭 주파수는 클럭 주기를 0.5 ns씩 줄여가며 합성하였을 때 합성 가능한 최대 주파수를 나타낸다. RS variant는 RS에 면적 감소 방법만 적용된 것이기 때문에 이 두 알고리즘의 소모 클럭 싸이클 수는 동일하다. AMMI와 AMMI variant-1의 실제 구현 시엔 while문의 탈출 조건을 AMMI variant-1에 맞춰 설계하였으며, 단지 면적 감소 방법의 적용 여부만 다르므로 이 둘의 소모 클럭 싸이클 수도 동일하다. AMMI variant-1~3은 최대 shift할 수 있는 bit양만 차이나며, 소모 클럭 싸이클 수가 33%, 14%씩 점차적으로 줄어드는 것을 볼 수 있다. 그에 따라 늘어나는 면적은 크진 않으나 소모 클럭 싸이클 수의 감소 정도가 점진적으로 줄어들기 때문에 shift할 수 있는 최대 bit의 양을 더 이상 늘리는 것은 비효율적일 것으로 예상된다. AMMI variant-3를 AMMI와 비교하면, 100MHz의 동작 주파수 기준으로 면적은 조금 더 작으며, 약 43% 적은 클럭 싸이클을 소모한다.

6. 결 론

본 논문은 ECC 모듈의 유한체 연산 중 전체 성능에 큰 영향을 미치는 MI 알고리즘의 하드웨어 구현에 대한 연구를 진행하였다. EM, LS, RS의 성능을 비교한 기존 연구로부터 하드웨어 설계 시 RS의 처리 속도가 좋은 편임을 알 수 있으며, 단점이었던 면적 문제 또한 해결 가능함을 알 수 있었다[1, 2]. 본 논문에서는 RS에 적용되었던 개선 방법을 비슷한 구조의 몽고메리 모듈러 역 알고리즘에도 적용 가능함을 보였으며, 처리 속도의 개선 방법도 함께 제시하고 이들의 면적과 속도를 분석하였다. 분석 결과 100MHz로 합성할 때를 기준으로, 속도와 면적이 개선된 AMMI variant-2와 AMMI variant-3이 면적 면에서 기존 AMMI 보다 각각 5.9%, 1.2% 작으면서도 각각 33.2%, 42.6% 적은 클럭 싸이클 수를 소모해 결과적으로 처리속도가 각각 49.7%, 74.4% 더 빠르다. 따라서 MM을 위해 몽고메리 곱을 사용하는 등 MMI가 필요할 때 제안한 AMMI variant들을 함께 사용하면 더 적은 면적에, 속도가 더 빠른 ECC 모듈을 설계할 수 있을 것으로 기대된다.

REFERENCE

[1] P. Choi, S. Lee, and D.K. Kim, “Design of Efficient Modular Inversion Module Using Resource Sharing,” *Proceeding of Korea Multimedia Society International Conference*

on Multimedia Information Technology and Applications, pp. 298-299, 2015.

[2] P. Choi, J. Kong, and D.K. Kim, "Analysis of Hardware Modular Inversion Modules for Elliptic Curve Cryptography," *Proceeding of International SoC Design Conference,* pp. 313-314, 2015.

[3] E. Barker and A. Roginsky, *NIST Special Publication 800-131A Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*, National Institute of Standards and Technology, 2011.

[4] W. Lee, C. Roh, and D. Ryu, "Critical Path Analysis for Codesign of Public Key Crypto-Systems," *Journal of Korea Multimedia Society*, Vol. 8, No. 1, pp. 78-87, 2016.

[5] N. Takagi, "A Modular Inversion Hardware Algorithm with a Redundant Binary Representation," *IEICE Transactions on Information and Systems*, Vol. E76-D, No. 8, pp. 863-869, 1993.

[6] X. Yan and S. Li, "Modified Modular Inversion Algorithm for VLSI Implementation," *Proceeding of International Conference on ASIC*, pp. 90-93, 2007.

[7] C. Chen and Z. Qin, "Fast Algorithm and Hardware Architecture for Modular Inversion in GF (p)," *Proceeding of International Conference on Intelligent Networks and Intelligent Systems*, pp. 43-45, 2009.

[8] S. Ma, Y. Hao, Z. Pan, and H. Chen, "Fast Implementation for Modular Inversion and Scalar Multiplication in the Elliptic Curve Cryptography," *Proceeding of International Symposium on Intelligent Information Technology Application*, pp. 488-492, 2008.

[9] R. Lórencz, "New Algorithm for Classical Modular Inverse," *Proceeding of International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 57-70, 2002.

[10] J. Hlaváč and R. Lórencz, "Arithmetic Unit for Computations in GF (p) with the Left-shifting Multiplicative Inverse Algorithm," *Proceeding of International Conference on Architecture of Computing Systems*, pp. 268-279, 2013.

[11] B.S. Kaliski, "The Montgomery Inverse and its Applications," *IEEE Transactions on Computers*, Vol. 44, No. 8, pp. 1064-1065, 1995.

[12] E. Savas and C.K. Koç, "The Montgomery Modular Inverse-revisited," *IEEE Transactions on Computers*, Vol. 49, Issue 7, pp. 763-766, 2000.

[13] R. Deng and Y. Zhou, "Improvement to Montgomery Modular Inverse Algorithm," *IEEE Transactions on Computers*, Vol. 55, No. 9, pp. 1207- 1210, 2006.

[14] FIPS PUB 186-2. *Digital Signature Standard*, National Institute of Standards and Technology, 2000.

[15] E. Barker, D. Johnson, and M. Smid, *NIST Special Publication 800-56A: Recommendation for Pair-wise Key Establishment Schemes Using Discrete Logarithm Cryptography*, Computer Security, National Institute of Standards and Technology, Vol. 114, 2007.

[16] *ANS X9. 62: 2005: Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm*, American National Standards Institute, Vol. 1430, 2005.

[17] D. Hankerson, A.J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer Science & Business Media, New York, USA, 2006.

[18] D. Galbi and A.K. Chan, *Four-to-two Adder Cell for Parallel Multiplication*, US4901270, US, 1990.

최 필 주

2010년 2월 한양대학교 융합전자공학부 학사 졸업
2012년 2월 한양대학교 융합전자공학부 석사 졸업
2012년 3월～현재 한양대학교 융합전자공학부 박사 과정

관심 분야 : 보안 SoC 설계, 암호 하드웨어 설계, 정보보안

김 동 규

1992년 2월 서울대학교 컴퓨터공학과 학사 졸업
1994년 2월 서울대학교 컴퓨터공학과 석사 졸업
1999년 2월 서울대학교 컴퓨터공학과 박사 졸업
1999년 9월～2006년 2월 부산대학교 정보컴퓨터공학부 조교수
2006년 3월～현재 한양대학교 융합전자공학부 교수
관심 분야 : 보안 SoC 설계, 암호 하드웨어 설계, Secure CPU 코어 설계, IoT용 디바이스 보안 시스템 설계