저 면적 타원곡선 암호프로세서를 위한 GF(2^m)상의 새로운 산술 연산기†

정회원 김 창 훈*, 권 순 학**, 홍 춘 표*

A New Arithmetic Unit Over GF(2^m) for Low-Area Elliptic Curve Cryptographic Processor

Chang-Hoon Kim*, Soon-Hack Kwon**, Chun-Pyo Hong* *Regular Members*

요 약

본 논문에서는 저 면적 타원곡선 암호프로세서를 위한 GF(2^m)상의 새로운 산술 연산기를 제안한다. 제안된 연산기는 바이너리 확장 최대공약수 알고리즘과 MSB(Most Significant Bit) 우선 곱셈 알고리즘으로부터 하드웨어 공유를 통하여 LFSR(Linear Feed Back Shft Register)구조로 설계되었으며, 나눗셈 및 곱셈 모두를 수행 할 수 있다. 즉 나눗셈 모드에서 $2m$-1 클락 사이클 지연 후 나눗셈의 결과를 출력하며, 곱셈 모드에서 m 클락 사이클 지연 후 곱셈 결과를 각각 출력한다. 본 논문에서 제안된 연산기를 기존의 나눗셈기들과 비교 분석한 결과 적은 트랜지스터의 사용으로 계산 지연시간을 감소 시켰다. 또한 제안된 연산기는 기약다항식의 선택에 어떠한 제약도 두지 않을 뿐 아니라 매우 규칙적이고 묘듈화 하기 쉽기 때문에 필드 크기 m 에 대하여 높은 확장성 및 유연성을 제공한다. 따라서, 본 연구에서 제안된 산술 연산기는 타원곡선 암호프로세서의 나눗셈 및 곱셈 연산기로 사용될 수 있다. 특히 스마트 카드나 무선통신기기와 같은 저 면적을 요구하는 응용들에 매우 적합하다.

Key Words : Elliptic Curve Cryptosystem, Finite Field Arithmetic, Standard Basis, VLSI

ABSTRACT

This paper proposes a novel arithmetic unit over GF(2^m) for low-area elliptic curve cryptographic processor. The proposed arithmetic unit, which is linear feed back shift register (LFSR) architecture, is designed by using hardware sharing between the binary GCD algorithm and the most significant bit (MSB)-first multiplication scheme, and it can perform both division and multiplication in GF(2^m). In other word, the proposed architecture produce division results at a rate of one per $2m$-1 clock cycles in division mode and multiplication results at a rate of one per m clock cycles in multiplication mode. Analysis shows that the computational delay time of the proposed architecture, for division, is less than previously proposed dividers with reduced transistor counts. In addition, since the proposed arithmetic unit does not restrict the choice of irreducible polynomials and has regularity and modularity, it provides a high flexibility and scalability with respect to the field size m. Therefore, the proposed novel architecture can be used for both division and multiplication circuit of elliptic curve cryptographic processor. Specially, it is well suited to low-area applications such as smart cards and hand held devices.

* 대구대학교 컴퓨터정보공학과 (chkim@dsp.taegu.ac.kr), ** 성균관대학교 수학과 (shkwon@math.skku.ac.kr)
논문번호 : 030038-0127, 접수일자 : 2003년 1월 27일
† 본 연구는 한국과학재단 목적기초연구(R05-2003-000-11573-0) 지원으로 수행되었음

I. 서 론

최근 인터넷 및 무선통신의 급성장으로 정보 보호는 아주 중요한 문제로 인식되고 있다. 정보 보호를 위해선 암호 시스템의 구현은 필요하다. 암호시스템은 소프트웨어를 이용하여 쉽게 구현이 가능하지만 실시간 응용을 위해선 적합하지 않을 뿐 아니라 사이드 채널 공격(타이밍 공격, 메모리 공격 등)에 약하다[1]. 따라서 보다 향상된 성능 및 안전성을 제공하기 위해 암호시스템의 하드웨어 구현은 바람직하다.

많은 암호시스템들 중 최근 타원곡선 암호시스템(Elliptic Curve Cryptosystem: ECC)은 학계나 산업계로부터 많은 관심을 모으고 있다. ECC의 가장 주된 장점은 RSA나 ElGamal과 같은 다른 암호 시스템에 비해 현저히 작은 키를 사용하면서(약 1/6 정도) 동일한 안전도를 가진다[2-3]. 작은 키를 사용한다는 것은 계산 시간, 전력 소모 그리고 저장공간의 감소를 의미한다. 따라서 ECC는 스마트 카드나 무선통신 기기와 같은 저 면적을 요구하는 응용들에 매우 적합하다.

ECC는 유한체 상에서 구현되며, 사용되는 유한체는 GF(p), GF(p^m) 그리고 GF(2^m)이 있다 (여기서 p는 소수이다). 이중 GF(2^m)은 0과 1을 원소로 같는 GF(2)의 m차원 확장 필드로 특히 하드웨어 구현에 적합하다. 가장 대표적인 GF(2^m)의 원소 표기법에는 정규기저(normal basis) 및 표준기저(standard basis) 표기법이 있다. 각 기저표기법은 장단점을 가지는데, 정규기저 표기법을 사용할 경우 제곱연산이 쉽게 되는 반면 곱셈연산이 매우 복잡하며 하드웨어 구조가 규칙적이지 못하고 모듈성이 떨어지기 때문에 하드웨어 구현에는 표준기저 표기법이 더 많이 사용된다.

ECC에서 가장 중요한 연산은 kP이다. 여기서 k는 큰 정수이고 P는 타원곡선상의 한 포인트이다. kP를 연산하기 위해서는 또다시 포인트 덧셈과 배 연산이 필요한데 타원곡선이 GF(2^m)상에서 정의되고 Affine 좌표계를 사용할 경우 포인트 덧셈은 GF(2^m)상에서 8번의 덧셈, 한번의 나눗셈, 한번의 곱셈 그리고 한번의 제곱으로 이루어지며, 배 연산은 6번의 덧셈, 한번의 나눗셈, 한번의 곱셈 그리고 한번의 제곱으로 이루어진다[2-3]. 여기서 덧셈은 비트별 XOR 연산이기 때문에 간단하게 구현이 된다. 반면 나머지 연산들은 계산 시간 및 하드웨어 비용 측면에서 복잡하다. 특히 나눗셈은 가장 복잡한 연산으로 실제 ECC의 성능을 좌우한다[1].

GF(2^m)상에서 나눗셈 연산을 하드웨어로 구현하기 위해 1) 선형방정식 집합의 해를 구하는 방법[4-5], 2) Fermat의 이론[6-7], 3) Euclid의 GCD 알고리즘[8-9]이 이용되어 왔다. 첫 번째 방법은 $2m$-1개의 미지수를 가진 $2m$-1개의 선형 방정식들의 해를 구함으로써 역원을 찾아낸다. 두 번째 방법은 Fermat의 이론을 이용하여 연속적인 제곱과 곱셈을 함으로써 역원을 찾는다. 즉 $A/B = AB^{-1} = AB^{2^m-2} = A(B(B(B\cdots B(B(B)^2)^2\cdots)^2)^2)^2$ 의 관계식을 이용하는데, 이 방법은 m번의 제곱연산과 약 $\log_2{}^{(m-1)}$번의 곱셈 연산을 필요로 한다. 마지막 방법은 Euclide의 GCD 알고리즘을 이용하는 방법으로 GCD($G(x)$, $B(x)$) = 1 이라는 사실을 이용한다. 여기서 $B(x)$는 GF(2^m)상의 0이 아닌 원소이고 $G(x)$는 GF(2^m)을 정의하는 기약 다항식이다. 확장된 Euclide의 GCD 알고리즘은 $W(x)\cdot G(x) + U(x)\cdot B(x) = 1$을 만족하는 $W(x)$와 $U(x)$를 찾는다. 따라서 $U(x)\cdot B(x) \equiv 1 \bmod G(x)$이 되어 $U(x)$는 $B(x)$의 역원 즉, $B^{-1}(x)$가 된다. 최근 연구 결과에 의하면 표준 기저표기법을 사용할 경우 Euclide의 GCD 알고리즘을 사용했을 때 가장 적은 하드웨어의 사용으로 가장 낮은 계산 지연시간을 가진다[9].

GF(2^m)상의 곱셈 알고리즘은 크게 LSB(Least Significant First)-우선 방식과 MSB-우선 방식이 있다[10]. LSB-우선 방식은 곱수의 LSB부터 처리되고, MSB-우선 방식은 MSB부터 처리된다. 이들 두 방식은 동일한 시간 및 면적 복잡도를 가지지만 LSB-우선 방식이 3개의 쉬프트 레지스터를 사용하는 반면, MSB-우선 방식은 2개의 쉬프트 레지스터를 사용한다. 따라서 전력 사용측면에서 MSB-우선 방식이 LSB-우선 방식보다 효율적이다[1].

이러한 나눗셈 및 곱셈 회로들은 비트-패러럴[6-7], [10] 혹은 비트-시리얼[1], [4-5], [8-9] 구조를 가진다. 고속의 나눗셈 연산을 위해서는 비트-패러럴 구조가 적절하나 ECC에서 필드의 크기 m은 적어도 163 이상이어야 하기 때문에 비트-패러럴 구조는 높은 면적 복잡도와 핀 개수 문제로 구현에 많은 어려움이 따를 뿐 아니

라 비실용적이다. 따라서 ECC를 위해선 비트-시리얼 구조가 적합하다.

본 논문에서는 GF(2^m)상에서 표준 기저 표기법을 사용하여, 비트-시리얼 구조의 새로운 산술 연산기를 제안한다. 제안된 연산기는 바이너리 최대공약수(Greatest Common Divisior: GCD) 알고리즘과 MSB-우선 곱셈 방식으로부터 하드웨어 공유를 통하여 이루어 졌으며, 나눗셈 모드에서 $2m$-1 클락 사이클 지연 후 나눗셈의 결과를 출력하고, 곱셈 모드에서 m 클락 사이클 지연 후 곱셈 결과를 각각 출력한다.

본 논문에서 제안된 연산기를 기존의 나눗셈 기들과 비교 분석한 결과 적은 트랜지스터의 사용으로 계산 지연시간을 감소 시켰다. 또한 제안된 연산기는 기약다항식의 선택에 어떠한 제약도 두지 않을 뿐 아니라 매우 규칙적이고 묘듈화 하기 쉽기 때문에 필드 크기 m 에 대하여 높은 확장성 및 유연성을 제공한다. 따라서, 본 연구에서 제안된 연산기는 ECC 프로세서의 나눗셈 및 곱셈 연산기로 사용될 수 있다. 특히 스마트 카드나 무선통신기기와 같은 저 면적을 요구하는 응용들에 매우 적합하다.

본 논문의 구성은 다음과 같다. 2절에서 ECC를 위한 GF(2^m)상의 연산에 대해서 알아본 후, 3절에서는 바이너리 확장 GCD 알고리즘과 MSB-우선 곱셈 알고리즘으로부터 나눗셈 및 곱셈을 모두 수행 할 수 있는 LFSR 구조의 새로운 연산기를 설계하고 FPGA 구현을 통하여 그 기능을 검증한다. 4절에서 기존의 나눗셈 회로들과 성능을 비교 및 분석하고, 5절에서 결론을 맺는다.

Ⅱ. ECC를 위한 GF(2^m)상의 연산

ECC에서 가장 중요한 연산은 kP이다. 여기서 k는 큰 정수이고 P는 타원곡선 상의 한 포인터이다. kP를 계산하기 위해 우리는 포인트 P를 k번 더하면 된다[2-3]. GF(2^m)을 특성이 2인 유한체라 하면, 무한정점 $\boldsymbol{O}$와 함께 식 (1)을 만족하는 모든 근의 집합은 GF(2^m)상의 non-supersingular 타원곡선 E(GF(2^m))이다. 여기서 a_2, $a_6 \in$ GF(2^m) 이고 $a_6 \neq 0$.

$$E : y^2 + xy = x^3 + a_2x^2 + a_6 \qquad (1)$$

$P_1 = (x_1, y_1)$과 $P_2 = (x_2, y_2)$를 affine 좌표계에서 E(GF(2^m))상의 두 포인트라 하고, P_1, $P_2 \neq \boldsymbol{O}$ 그리고 $P_1 \neq -P_2$라고 가정하면, 두 포인트의 덧셈 $P_3 = (x_3, y_3) = P_1 + P_2$는 아래와 같이 계산된다.

If $P_1 \neq P_2$ (포인트 덧셈 연산)

$\lambda = (y_1 + y_2)/(x_1 + x_2)$,

$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a_2$

$y_3 = (x_1 + x_3)\lambda + x_3 + y_1$

If $P_1 = P_2$ (포인트 배연산)

$\lambda = y_1/x_1 + x_1$,

$x_3 = \lambda_2 + \lambda + a_2$

$y_3 = (x_1 + x_3)\lambda + x_3 + y_1$

위에 기술된바와 같이, E(GF(2^m))상의 서로 다른 두 포인트 덧셈은 GF(2^m)상에서 8번의 덧셈, 한번의 나눗셈, 한번의 곱셈 그리고 한번의 제곱으로 이루어지며, 배 연산은 6번의 덧셈, 한번의 나눗셈, 한번의 곱셈 그리고 한번의 제곱으로 이루어진다. 여기서 덧셈은 비트별 XOR 연산이기 때문에 쉽게 구현이 되고, 제곱은 곱셈으로 대체 될 수 있다. 위의 포인트 덧셈 및 배 연산을 보면 데이터 의존 때문에 덧셈을 제외한 어떠한 연산도 동시에 수행 될 수 없다. 따라서 곱셈 및 나눗셈 기능의 분리된 구현 보다 이들간의 하드웨어 공유를 통한 구현이 훨씬 바람직하다.

Ⅲ. ECC 프로세서를 위한 새로운 산술 연산기

3.1 GF(2^m)상의 새로운 나눗셈기

$A(x)$와 $B(x)$는 GF(2^m)상의 두 원소이고, $G(x)$는 GF(2^m)을 정의하는, 즉 GF(2^m)≅ GF(2)[x]/$G(x)$, 차수 m의 기약 다항식이고, $P(x)$는 $A(x)/B(x)$ mod $G(x)$의 결과라고 하면, 각각의 다항식은 다음과 같이 표현되며 계수들은 이진수 0 혹은 1이다.

$$A(x) = a_{m-1}x^{m-1}+a_{m-2}x^{m-2}+\cdots+a_1x+a_0 \qquad (2)$$

$$B(x) = b_{m-1}x^{m-1}+b_{m-2}x^{m-2}+\cdots+b_1x+b_0 \qquad (3)$$

$$G(x) = x^m+g_{m-1}x^{m-1}+\cdots+g_1x+g_0 \qquad (4)$$

$$P(x) = p_{m-1}x^{m-1}+p_{m-2}x^{m-2}+\cdots+p_1x+p_0 \qquad (5)$$

GF(2^m)상의 나눗셈을 위해 [알고리즘 1]은 사용될 수 있다[11]. [알고리즘 1]에서 $B(x)$는 GF(2^m)상의 0이 아닌 원소이고 $W(x) \cdot G(x) + U(x) \cdot B(x) = 1$을 만족하는 $W(x)$와 $U(x)$를 찾으면, $U(x) \cdot B(x) \equiv 1 \bmod G(x)$가 되어 $U(x)$는 $B(x)$의 역원 즉, $B^{-1}(x)$이 된다. 이때 $U(x)$에 초기값 1 대신 $A(x)$를 대입하면 알고리즘의 종료 후 나눗셈 결과 $P(x) = A(x)/B(x) \bmod G(x)$를 얻을 수 있다[11].

[알고리즘 1] GF(2^m)상의 나눗셈을 위한 바이너리 확장 GCD 알고리즘

Input: $G(x)$, $A(x)$, $B(x)$
Output: V has $P(x)=A(x)/B(x) \bmod G(x)$
Initialize: $R=B(x)$, $S=G=G(x)$, $U=A(x)$, $V=0$, $count=0$, $state=0$

1. **for** $i = 1$ **to** $2m$ **do**
2. **if** $state == 0$ **then**
3. $count = count+1$;
4. **if** $r_0 == 1$ **then**
5. $(S, R)=(R, R+S)$; $(V, U)=(U, U+V)$;
6. $state = 1$;
7. **end if**
8. **else**
9. $count = count-1$;
10. **if** $r_0 == 1$ **then**
11. $(S, R)=(S, R+S)$; $(V, U)=(V, U+V)$;
12. **end if**
13. **if** $count == 0$ **then**
14. $state = 0$;
15. **end if**
16. **end if**
17. $R = R/x$;
18. $U = U/x$;
19. **end if**
20. **end for**

[알고리즘 1]의 VLSI 설계에 앞서 핵심연산 및 제어함수에 대해서 고려한다. [알고리즘 1]의 S와 R은 차수가 m인 다항식 그리고 U와 V는 차수가 m-1인 다항식으로 각각 표현 할 수가 있으며, 각 다항식은 아래와 같다.

$$R = r_m x^m + r_{m-1}x^{m-1} + \cdots + r_1 x + r_0 \quad (6)$$

$$S = s_m x^m + s_{m-1}x^{m-1} + \cdots + s_1 x + s_0 \quad (7)$$

$$U = u_{m-1}x^{m-1} + u_{m-2}x^{m-2} + \cdots + u_1 x + u_0 \quad (8)$$

$$V = v_{m-1}x^{m-1} + v_{m-2}x^{m-2} + \cdots + v_1 x + v_0 \quad (9)$$

[알고리즘 1]의 단계 5와 11에 나타나듯이 S와 V연산은 $state$와 r_0에 따른 간단한 교환 연산이다. 반면 R과 U는 두 개의 연산을 가진다. 우선 R의 연산을 살펴보면, r_0에 따라 $((S + R)/x)$ 또는 (R/x)를 계산해야한다. 따라서 우리는 아래와 같이 R의 임시적인 결과를 얻을 수 있다.

R'을 아래와 같이 R의 임시적인 결과라 두면

$$R' = r'_m x^m + r'_{m-1}x^{m-1} + \cdots + r'_1 x + r'_0 \quad (10)$$

r_0, 식 (6) 그리고 식 (7)로부터 아래의 식들을 얻을 수 있다.

$$r'_m = 0 \quad (11)$$

$$r'_{m-1} = r_0 s_m = r_0 s_m + 0 \quad (12)$$

$$r'_i = r_0 s_{i+1} + r_{i+1},\ 0 \le i \le m-2 \quad (13)$$

U의 연산에 관해 살펴보면, $(U + V)$ 그리고 U/x을 계산해야 한다.
$(U + V)$를 U''라 두면

$$U'' = u''_{m-1}x_{m-1} + \cdots + u''_1 x + u''_0 = U + V \quad (14)$$

r_0, 식 (8) 그리고 식 (9)로부터 우리는 아래의 식 (15)를 얻을 수 있다.

$$u''_i = r_0 v_i + u_i,\ 0 \le i \le m-1 \quad (15)$$

x가 기약다항식 $G(x)$의 근이기 때문에 식 (16)을 얻을 수 있다.

$$x^m = g_{m-1}x^{m-1} + g_{m-2}x^{m-2} + \cdots + g_1 x + g_0 \quad (16)$$

식 (16)으로부터 g_0는 항상 1이기 때문에 (17)와 같이 다시 쓸 수 있다.

$$1 = (x^{m-1} + g_{m-1}x^{m-2} + g_{m-2}x^{m-3} + \cdots + g_2 x + g_1)x \quad (17)$$

식 (17)으로부터, 양변에 x를 나누면, 우리는 아래의 식 (18)을 얻을 수 있다.

$$x^{-1} = x^{m-1} + g_{m-1}x^{m-2} + g_{m-2}x^{m-3} + \cdots + g_2 x + g_1 \quad (18)$$

U/x의 결과를 U'''라 두면

$$U''' = u'''_{m-1}x^{m-1}+\cdots+u'''_1x+u'''_0 = U/x \quad (19)$$

식 (8)과 식 (18)을 식 (19)에 대입하면, 아래의 식들을 유도 할 수 있다.

$$u'''_{m-1} = u_0 = u_0g_m + 0 \quad (20)$$

$$u'''_i = u_{i+1} + u_0g_{i+1}, \quad 0 \le i \le m-2 \quad (21)$$

마지막으로 U''/x의 결과를 U'라 두면

$$U' = u'_{m-1}x^{m-1}+\cdots+u'_1x+u'_0 = U''/x \quad (22)$$

식(14)와 식 (19)로부터, U의 임시적인 결과를 아래와 같이 얻을 수 있다.

$$u'_{m-1} = r_0v_0+u_0 = (r_0v_0+u_0)g_m+r_00+0 \quad (23)$$

$$u'_i = (r_0v_{i+1}+u_{i+1})+(r_0v_0+u_0)g_{i+1}, \ 0\le i\le m-2 \quad (24)$$

또한 [알고리즘 1]과 핵심 연산들을 위한 제어 함수들은 아래와 같다.

$$\text{Ctrl1} = (r_0==1) \quad (25)$$

$$\text{Ctrl2} = u_0 \text{ XOR } (v_0 \ \& \ r_0) \quad (26)$$

$$\text{Ctrl3} = (state == 0)\&(r0==1) \quad (27)$$

$$count' = \begin{cases} count+1, if\ state = 0 \\ count-1, if\ state = 1 \end{cases} \quad (28)$$

$$state = \overline{state}, \text{ if} \begin{cases} ((r_m==1)\&(state==0)) \text{ or} \\ ((count==1)\&(state==1)) \end{cases} \quad (29)$$

핵심 연산 및 제어함수로부터 우리는 그림 1과 같은 LFSR 구조의 새로운 나눗셈기를 얻을 수 있다.

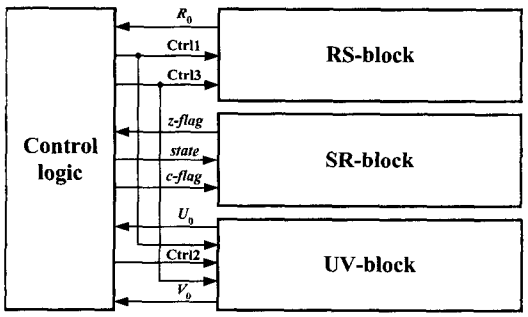
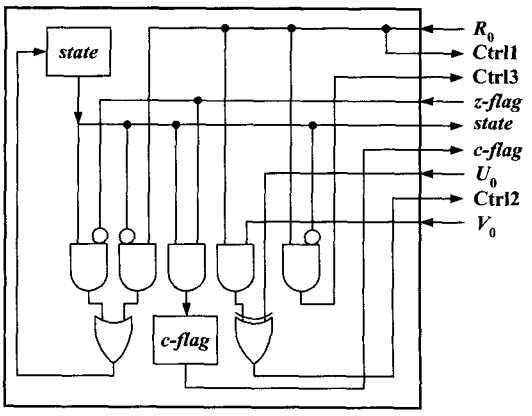


그림 1. GF(2^m)을 위한 새로운 나눗셈 구조

그림 1의 나눗셈기는 Control logic, RS-block, SR-block 그리고 UV-block으로 구성되어있다. 각각에 해당하는 기능블록은 그림 2, 그림 3, 그림 4 그리고 그림 5와 같다. 핵심 연산으로부터 s_0는 항상 1이고 r_m은 항상 0 이기 때문에 그림 3에서 이 계수들을 위한 연산 회로는 제거하였다. 그림 4에 나타나듯이 [알고리즘 2]의 *count* 값을 위해서 $\log_2(m+1)$-비트 카운터 대신 m-비트 양방향 쉬프트 레지스터를 사용하였다. 이렇게 함으로 해서 약간의 하드웨어 비용은 증가하나 높은 클락 주파수를 얻을 수 있다. Control logic은 현재의 반복을 위해 Ctrl1, Ctrl2, Ctrl3 그리고 *c-flag*의 신호를 생성하고 다음 반복을 위해 *state* 및 *c-flag*값을 갱신한다. 1-비트 *c-flag* 레지스터는 SR-block과 함께 동작하며 나눗셈을 시작할 때 1로서 초기화된다. 그림 3의 RS-block은 [알고리즘 2]의 R 과 S에 관한 연산을 수행하고 Control logic에 r_0신호를 전파한다. 그림 4에 나타나듯이 양방향 쉬프트 레지스터는 *state*에 따라 왼쪽 혹은 오른쪽으로 쉬프트 되며, cnt_m이 1일 때 *count* 값은 m이 된다. 또한 *count*가 0으로 감소하게되면, *z-flag*는 1이 되고 모든 cnt_m은 0이 되고, *c-flag*는 1이 되고 *state*는 0이 된다. 결국 이것은 나눗셈의 첫 번째 연산 조건과 동일하다. 그림 5의 UV-block은 U와 V에 관한 연산을 수행하며 Contrl logic에 u_0 와 v_0 신호를 전파한다.



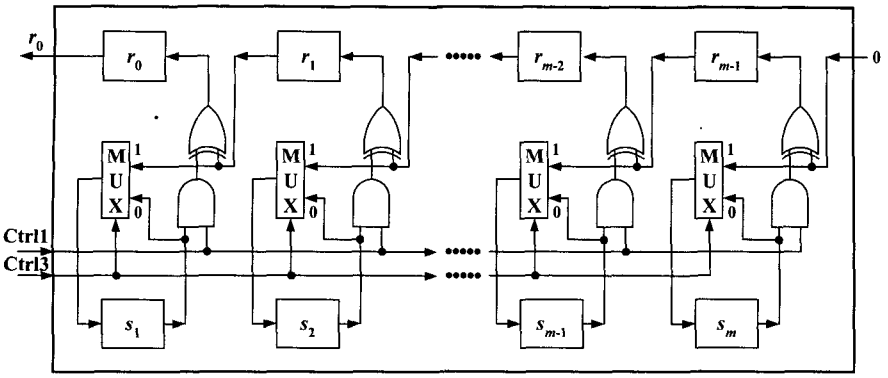


그림 3. 그림 1의 RS-block 회로

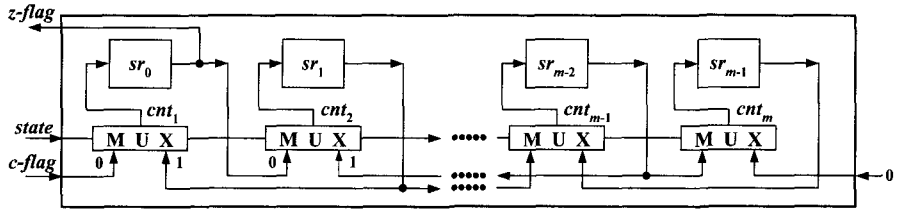


그림 4. 그림 1의 SR-block 회로

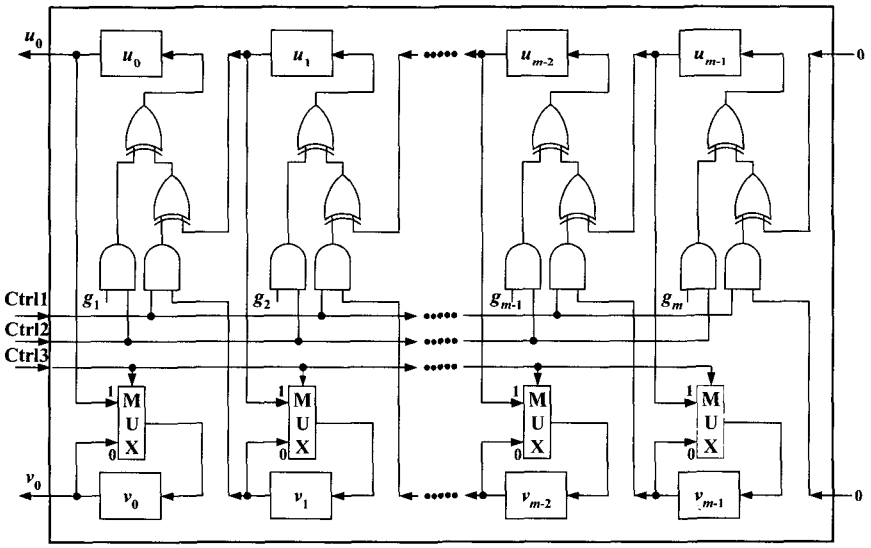


그림 5. 그림 1의 UV-block 회로

그림 2. 그림 1의 Control logic 회로

3.2 GF(2^m)상의 MSB 우선 곱셈기

$A(x)$, $B(x)$ 그리고 $P(x)$는 GF(2^m)상의 원소이고, $G(x)$는 차수 m의 기약 다항식이라 하면, 곱셈 결과 $P(x) = A(x)B(x) \bmod G(x)$ 는 아래와 같은 방법으로 계산할 수 있다[10].

$$\begin{aligned} P(x) &= A(x)B(x) \bmod G(x) \\ &= \{\cdots[A(x)b_{m-1}x \bmod G(x) \\ &\quad + A(x)b_{m-2}]x \bmod G(x) + \cdots \\ &\quad + A(x)b_1\}x \bmod G(x) + A(x)b_0 \end{aligned} \quad (30)$$

식 (30)으로부터 우리는 아래 식 (31)과 같이 $P(x)$의 임시적인 결과를 계산 할 수 있다. 여기

서 $P^{(0)}(x) = 0$ 그리고 $1 \le i \le m$.

$$P^{(i)}(x) = (P^{(i-1)}(x)x + b_{m-i}A(x)) \bmod G(x)$$
$$= (P^{(i-1)}(x)x \bmod G(x)) + b_{m-i}A(x) \quad (31)$$

식 (31)로부터 $(P^{(i-1)}(x)x \bmod G(x))$는 아래와 같이 계산 될 수 있다.

$$(P^{(i-1)}(x)x \bmod G(x))$$
$$= \sum_{k=0}^{m-1} p_k^{(i-1)} x^{(k+1)}$$
$$= p_{m-1}^{(i-1)} x^m + \sum_{k=0}^{m-2} p_k^{(i-1)} x^{(k+1)} \quad (32)$$

식 (16)으로부터 $x^m = \sum_{k=0}^{m-1} g_k x^k$. 따라서 차수 감소를 위해 식 (32)에 $\sum_{k=0}^{m-1} g_k x^k$ 을 x^m에 대입하면, 아래의 식을 유도 할 수 있다.

$$p_{m-1}^{(i-1)} \sum_{k=0}^{m-1} g_k x^{(k)} + \sum_{k=1}^{(i-1)} x(k)$$

$$= \sum_{k=0}^{m-1} (p_{m-1}^{(i-1)} g_k + p_{k-1}^{(i-1)}) x^k (\text{with } p_{-1}^{i-1} \equiv 0) \quad (33)$$

식 (31)과 식 (33)으로부터 우리는 아래의 MSB 우선 곱셈 알고리즘을 얻을 수 있다.

[알고리즘 2] GF(2^m)상의 MSB 우선 곱셈 알고리즘

Input: $G(x)$, $A(x)$, $B(x)$
Output: $P(x) = A(x) / B(x) \bmod G(x)$
1. $p_k^{(0)}=0$, **for** $0 \le k \le m-1$
2. $p_{-1}^{(i)}=0$, **for** $1 \le i \le m$
3. **for** $i=1$ **to** m **do**
4. **for** $k=m-1$ **to down to** 0 **do**
5. $p_k^{(i)} = p_{m-1}^{(i-1)} g_k + b_{m-i} a_k + p_{k-1}^{(i-1)}$
6. **end**
7. **end**
8. $P(x) = p^{(m)}(x)$

위의 알고리즘에서, $p_k^{(i)}$은 $P^{(i)}(x)$의 k번째 계수를 나타내고 a_k, b_k 그리고 g_k는 각각 $A(x)$, $B(x)$ 그리고 $G(x)$의 k번째 계수를 나타낸다. MSB 우선 곱셈 알고리즘으로부터 우리는 그림 6과 같은 LFSR 구조의 곱셈기를 얻을 수 있다.

3.3 ECC 프로세서를 위한 새로운 산술 연산기

지금까지 설계된 나눗셈기 및 곱셈기로부터 우리는 다음과 같은 유사점들을 발견할 수 있다. 1) 나눗셈기의 U에 관한 연산은 곱셈기의 P에 관한 연산과 동일하다 2) 그림 4의 SR 레지스터가 양방향으로 쉬프트 되는 반면, 곱셈기의 B 레지스터는 오른쪽 방향으로만 쉬프트 된다.

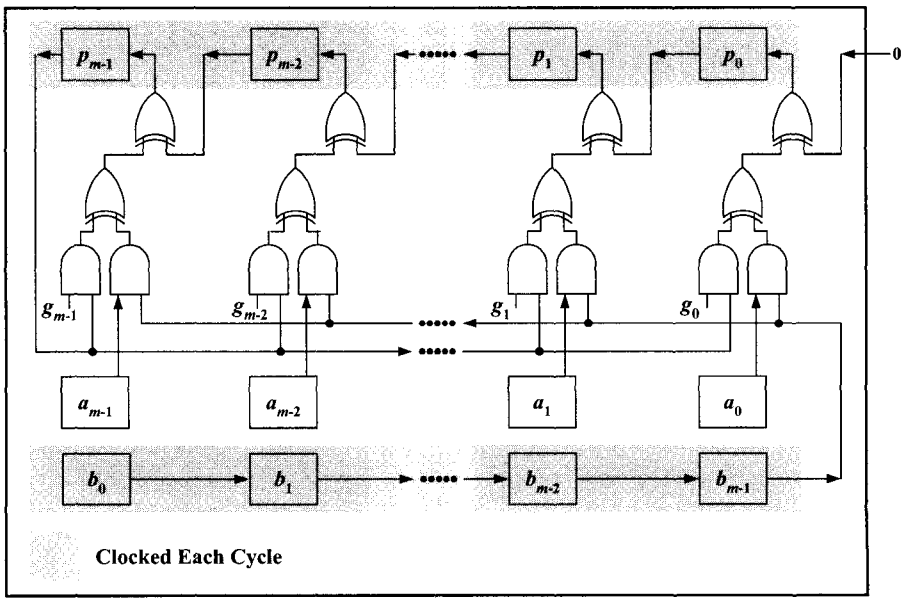


그림 6. GF(2^m)상의 MSB 우선 곱셈기

따라서 그림 1의 나눗셈기로 곱셈을 수행하기 위해 그림 1의 SR-block과 UV-block을 수정하였으며, 그 결과는 각각 그림 7과 그림 8에 주어지며, 수정 절차는 아래와 같다.

(1) SR-block : 그림 7에 나타나듯이 곱셈 연산을 수행 할 때 그림 7의 레지스터를 단지 왼쪽 방향으로만 쉬프트 시키기 위해 우리는 하나의 2-to-1 OR 게이트와 *mult/div* 신호를 추가하였다. 즉 곱셈을 수행할 때는 *mult/div*에 0을 입력하고 나눗셈을 수행할 때는 1을 입력한다. 따라서 그림 7의 레지스터는 나눗셈 모드에서는 *state*의 값에 따라 양방향으로 쉬프트 되고 곱셈 모드에서는 왼쪽 방향으로만 쉬프트 된다.

(2) UV-block : 그림 8에 보이듯이, 그림 5에 비해 두 개의 2-to-1 멀티플렉서, 두 개의 2-to-1 AND 게이트 그리고 *mult/div* 신호가 추가되었다. 그림 8에 나타나듯이 두 개의 2-to-1 멀티플렉서를 추가함으로서 곱셈 모드에선 Ctrl2와 *z-flag*대신 p_{m-1}과 b_i가 각각 선택된다. 또한 숫자 1로 표기된 AND 게이트는 곱셈 모드에서 항상 0을 생성하기 때문에 각각의 a_i/b_i 레지스터는 자기 자신의 값을 선택하고 숫자 2로 표기된 AND 게이트는 나눗셈 모드에서 항상 0을 생성한다.

그림 7과 그림 8로부터, 나눗셈 모드에서 우리는 *B/SR* 과 *A/V* 레지스터를 클리어하고, *P/U* 레지스터에 *U*를 로딩하고 그리고 *mult/div* 신호에 1을 인가한다. 또한 곱셈 모드에선 *P/U* 레지스터를 초기화하고, *A/V*와 *B/SR* 레지스터에 각각 *A*와 *B*를 로딩하고 그리고 *mult/div* 신호에 0을 인가한다. 따라서 나눗셈 모드일 경우 $2m-1$ 클락 사이클 지연 후 *A/V* 레지스터가 나눗셈 결과를 가지고 곱셈 모듈일 경우 m 클락 사이클 지연 후 *P/U* 레지스터가 곱셈 결과를 가진다.

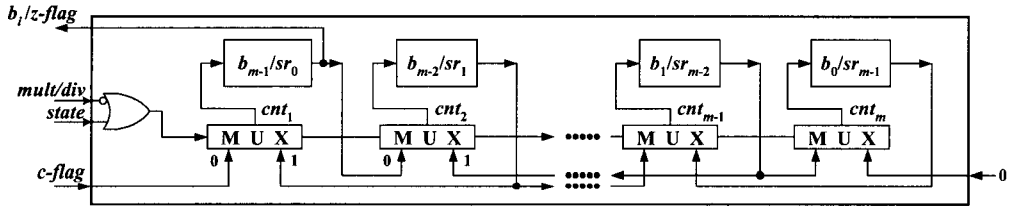


그림 7. 수정된 SR-block

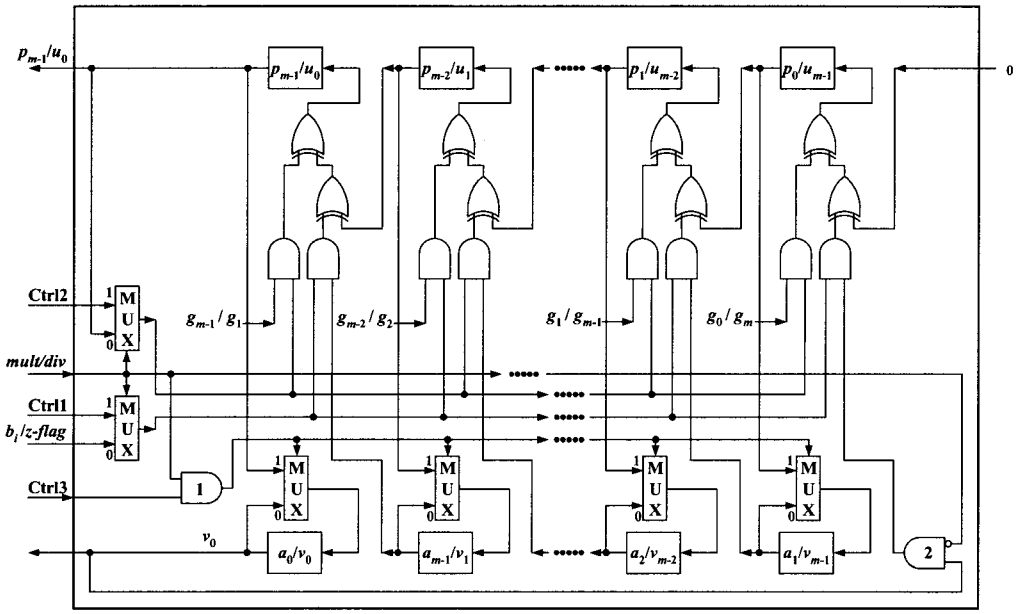


그림 8. 수정된 UV-block

표 1. GF(2^m)상의 나눗셈 회로들의 성능 비교

	Brunner 등 [8]	제안된 연산기
처리율 (1/cycles)	$1/2m$	$1/2m-1$: 나눗셈 $1/m$: 곱셈
지연시간(cycles)	$2m$	$2m-1$: 나눗셈 m : 곱셈
최대 처리기 지연시간	Tzero-detector + $2T_{AND2}$ + $2T_{XOR2}$ + $2T_{MUX2}$	$2T_{AND2}$ + $2T_{XOR2}$ + T_{MUX2}
회로의 구성요소들	AND_2: $3m+\log_2(m+1)$ XOR_2: $3m+\log_2(m+1)$ Latch: $4m+\log_2(m+1)$ MUX_2: $8m$	Inverter : 5 AND_2: $3m+7$ XOR_2: $3m+1$ OR_2: 2 Latch: $5m+2$ MUX_2: $3m+2$
TR 갯수	$110m+18\log_2(m+1)$	$88m+84$
연산	나눗셈	나눗셈 및 곱셈

AND_i:i-input AND gate
XOR_i:i-input XOR gate
OR_i:i-input OR gate
MUX_i:i-to-1 multiplexer
T_{ANDi}:the propagation delay through one AND_i gate
T_{XORi}:the propagation delay through one XOR_i gate
T_{MUXi}:the propagation delay through one MUX_i gate
Tzero-detector: the propagation delay of $(\log_2^{m+1})$-bit zero-detector

3.4 제안한 연산기의 FPGA 구현 및 검증

본 연구에서 제안된 산술 연산기의 기능 검증을 위해, VHDL로 회로를 기술하였고, Mento Graphics사의 합성 툴(LeonardSpectrum Version: 2002c.15)을 사용하여 회로를 합성, Net-list 파일을 추출한 후, Altera사의 Quartus II 2.1을 이용하여 Place & Route 과정을 거친 후, 타이밍 분석 및 시뮬레이션을 수행하였다. 여기서 Altera사의 150만 게이트급인 EP2A40F1020C7을 대상 디바이스로 선택하였으며, 필드 크기 m을 최대 571까지 확장하여 다양하게 시뮬레이션을 수행한 결과 모두 정확한 결과를 얻었다.

Ⅳ. 성능분석

본 연구에서 제안한 연산기를 기존에 제안된 나눗셈기와 비교하였다. 표 1에 성능 비교 결과를 요약하였으며, 조금 더 자세한 비교를 위해 3-입력 XOR 게이트는 2개의 2-입력 XOR 게이트로 구성된다고 가정하였다. 또한 2-입력 AND 게이트, 2-입력 XOR 게이트, 2-to-1 MUX, 2-입력 OR 게이트, 그리고 1-비트 래치는 각각 4, 6, 6, 6 그리고 8개의 트랜지스터(TR)로 구성된다고 가정하였다[12].

표 1에 기술된바와 같이, Brunner 등이 제안한 나눗셈기는 본 연구 결과와 거의 동일한 처리율과 계산 지연시간을 가지지만 훨씬 높은 최대 처리기 지연시간을 가지며 더 많은 TR을 사용한다. 더욱이 제안된 연산기는 기존에 제안된 LFSR 형태의 비트-시리얼 곱셈기와 동일한 계산 지연시간을 가지고 곱셈 연산도 할 수 있기 때문에 그림 6에 기술되어 있는 부가적인 하드웨어가(TR 개수 : $44m$) 필요 없다. 따라서 본 연구에서 제안된 나눗셈기는 기존의 연구 결과에 비해 시간 및 공간 모두에 있어 효율적이다.

Ⅴ. 결론

본 논문에서는 바이너리 확장 GCD 알고리즘과 MSB 우선 곱셈 방식으로부터 ECC 프로세서에 사용될 수 있는 GF(2^m)상의 새로운 연산기를 제안하였으며, FPGA 구현을 통하여 정확히 동작함을 확인하였다. 제안된 연산기는 LFSR 구조를 가지며, 기존의 연산기와는 달리 나눗셈 및 곱셈을 모두 수행 할 수 있다. 즉 나눗셈 모드에서 $2m$-1 클락 사이클 지연 후 나눗셈의 결과를 출력하며, 곱셈 모드에서 m 클락 사이클 지연 후 곱셈 결과를 각각 출력한다.

본 논문에서 제안된 연산기를 기존의 나눗셈기와 비교 분석한 결과 적은 트랜지스터의 사용

으로 계산 지연시간을 감소 시켰다. 또한 제안된 연산기는 기약다항식의 선택에 어떠한 제약도 두지 않을 뿐 아니라 매우 규칙적이고 묘듈화 하기 쉽기 때문에 필드 크기 m 에 대하여 높은 확장성 및 유연성을 제공한다. 따라서, 본 연구에서 제안된 산술 연산기는 타원곡선 암호 프로세서의 나눗셈 및 곱셈 연산기로 사용될 수 있다. 특히 스마트 카드나 무선통신기기와 같은 저 면적을 요구하는 응용들에 매우 적합하다.

참 고 문 헌

[1] J.R. Goodman, Energy Scalable Reconfigurable Cryptographic Hardware for Portable Applications, PhD thesis, MIT, 2000.

[2] IEEE P1363, *Standard Specifications for Publickey Cryptography*, 2000.

[3] I. F. Blake, G. Seroussi, and N. P. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.

[4] C.-L. Wang and J.-L. Lin, "A Systolic Architecture for Computing Inverses and Divisions in Finite Fields GF(2^m)," *IEEE Trans. Computers.*, vol. 42, no. 9, pp. 1141- 1146, Sep. 1993.

[5] M.A. Hasan and V.K. Bhargava, "Bit-Level Systolic Divider and Multiplier for Finite Fields GF(2^m)," *IEEE Trans. Computers*, vol. 41, no. 8, pp. 972-980, Aug. 1992.

[6] S.-W. Wei, "VLSI Architectures for Computing exponentiations, Multiplicative Inverses, and Divisions in GF(2^m)," *IEEE Trans. Circuits Syst. II*, vol 44, no. 10, pp. 847-855, Oct. 1997.

[7] A.V. Dinh, R.J. Bolton, and R. Mason, "A Low Latency Architecture for Computing Multiplicative Inverses and Divisions in GF(2^m)," *IEEE Trans. Circuits Syst. II*, vol. 48, no. 8, pp. 789-793, Aug. 2001.

[8] H. Brunner, A. Curiger and M. Hofstetter, "On Computing Multiplicative Inverses in GF(2^m)," *IEEE Trans. Computers.*, vol. 42, no. 8, pp. 1010-1015, Aug. 1993.

[9] J.H. Guo and C.L. Wang, "Bit-serial Systolic Array Implementation of Euclid's Algorithm for Inversion and Division in GF(2^m)," *Proc. 1997 Int. Symp. VLSI Tech., Systems and Applications*, pp. 113-117, 1997.

[10] S.K. Jain, L. Song, and K.K. Parhi, "Efficient Semi-Systolic Architectures for Finite Field Arithmetic," *IEEE Trans. VLSI Syst.*, vol. 6, no. 1, pp. 101-113, Mar. 1998.

[11] C.H. Kim and C.P. Hong, "High-speed division architecture for GF(2^m)," Electronics Letters, vol. 38, no. 15, pp. 835-836, July 2002.

[12] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A System Perspective*, 2nd ed. Reading, MA: Addison-Wesley, 1993.

김 창 훈(Chang-Hoon Kim) 정회원

2001년 2월 : 대구대학교 컴퓨터정보공학부, 학사
2003년 2월 : 대구대학교 컴퓨터정보공학과, 석사
2003년 3월~현재 : 대구대학교 컴퓨터정보공학과, 박사과정

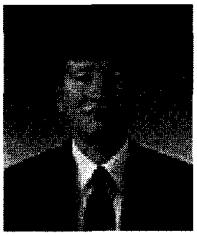
<주관심분야> 암호 시스템, Embedded System

권 순 학(Soon-Hack Kwon) 정회원

1990년 2월 : KAIST 수학과, 학사
1992년 2월 : 서울대학교 수학과, 석사
1997년 5월 : Johns Hopkins University, 박사
1998년 3월~현재 : 성균관대학교 수학과, 조교수

<주관심분야> 정수론, 암호이론, 회로이론

홍 춘 표(Chun-Pyo Hong) 정회원



1978년 2월 : 경북대학교 전자공학과, 학사
1986년 12월 : Georgia Institute of Technology ECE, 석사
1991년 12월 : Georgia Institute of Technology ECE, 박사
1994년 9월~현재 : 대구대학교 정보통신공학부, 교수

<주관심분야> DSP 하드웨어 및 소프트웨어, 컴퓨터 구조, VLSI 신호처리, Embedded System