情報保護學會論文誌
第13券 第5號, 2003. 10

삼항 다항식을 이용한 효율적인 비트-병렬 구조의 곱셈기

정 석 원*, 이 선 옥*, 김 창 한**

Design of an Efficient Bit-Parallel Multiplier using Trinomials

Seok Won Jung*, Seon Ok Lee*, Chang Han Kim**

요 약

최근 빠른 하드웨어의 구현은 속도의 효율성을 중시하는 환경에서 큰 관심의 대상이 되고 있다. 유한체 연산기는 연산과정이 복잡한 곱셈 연산에 의해 속도가 결정된다. 연산 수행 속도를 빠르게 개선하기 위해 본 논문에서는 하드웨어 구조를 기존의 Mastrovito방법을 이용하여 제안하고자 한다. 삼항기약다항식(trinomial) $p(x)=x^m+x^n+1$를 이용하여 제안하는 곱셈기의 시간 복잡도를 기존의 복잡도[14] $T_A+(\lfloor (m-2)/(m-n) \rfloor +1+\lceil \log_2(m) \rceil)T_X$에서 $T_A+(1+\lceil \log_2(m-1)+\lceil n/2 \rceil \rceil)T_X$으로 감소시킨다. 그러나 공간 복잡도를 살펴보면 AND 게이트 수가 기존의 복잡도와 m^2으로 같지만, XOR 게이트의 수는 기존 복잡도[14]인 m^2-1에서 $m^2+(n^2-3n)/2$으로 기약다항식의 중간항 차수인 n에 따라 약간 증가된다. 기약다항식의 최고차 항을 표준에서 권장하는 차수와 그에 준하는 다항식의 차수에 대해 XOR 공간 복잡도가 평균적으로 1.18% 증가하는 데 비해, 시간 복잡도는 평균적으로 9.036% 정도 감소한다.

ABSTRACT

Recently efficient implementation of finite field operation has received a lot of attention. Among the $GF(2^m)$ arithmetic operations, multiplication process is the most basic and a critical operation that determines speed-up hardware. We propose a hardware architecture using Mastrovito method to reduce processing time. Existing Mastrovito multipliers using the special generating trinomial $p(x)=x^m+x^n+1$ require m^2-1 XOR gates and m^2 AND gates. The proposed multiplier needs m^2 AND gates and $m^2+(n^2-3n)/2$ XOR gates that depend on the intermediate term xn. Time complexity of existing multipliers is $T_A+(\lfloor (m-2)/(m-n) \rfloor +1+\lceil \log_2(m) \rceil)T_X$ and that of proposed method is $T_A+(1+\lceil \log_2(m-1+\lceil n/2 \rceil \rceil)T_X$. The proposed architecture is efficient for the extension degree m suggested as standards: SEC2, ANSI X9.63. In average, XOR space complexity is increased to 1.18% but time complexity is reduced 9.036%.

keyword : *Elliptic Curve Cryptosystem, Bit-Parallel Multiplier, Mastrovito Multiplication Method*

Ⅰ. 서 론

효율적인 연산기는 코딩 이론(coding theory)과 컴퓨터 대수(computer algebra), 타원곡선 암호 시스템(elliptic curve cryptosystem)등의 분야에 널리 적용되고 있다.[6,9,10] 하드웨어의 효율성은 연산 속도와 하드웨어가 차지하는 공간의 크기로 표현된다. 그러나 하드웨어 집적 기술의 발전으로 하드웨어에 대한 관심 대상은 수행 시간의 최적화에 초점이 맞추어지고 있다. 효율적인 연산기를 개발하기 위한 연구는 암호시스템을 보다 효과적으로 구현하는 데 기여한다.

효율성의 척도는 복잡도로 표현되며, 복잡도(com-

* 고려대학교 정보보호대학원({jsw, seonognim}@cist.korea.ac.kr)
** 세명대학교 인터넷 정보학부(chkim@semyung.ac.kr)

plexity)는 시간 복잡도(time complexity)와 공간 복잡도(space complexity)로 나누어진다. 유한체 $GF(2^m)$ 위에서 곱셈기를 구성할 경우 공간 복잡도는 덧셈에 관련되어 있는 *XOR* 게이트와 곱셈에 관련되어 있는 *AND* 게이트의 수로서 측정하고, 시간 복잡도는 회로의 총 게이트 시간 지연(gate delay)으로 결정한다.

유한체의 연산은 원소의 표현과 유한체를 구성하는 기약다항식과 관련되어 있다. 특히, 연산 중에서 복잡한 곱셈을 살펴보면 두 단계로 이루어져 있다. 이는 다항식 곱셈을 수행하는 단계와 그 곱셈의 결과를 유한체 원소로 표현하기 위한 모듈러 감산 연산단계이다.[7,8] 이와 달리 단계적으로 분리되어 있는 과정을 곱셈 행렬을 이용하여 곱셈과 모듈로 감산 연산을 통합하는 연산 방식이 Mastrovito에 의해 제안되었다[7]. Mastrovito방식을 이용할 경우 [14]에서는 일반적인 형태의 모든 삼항 기약다항식

$$x^m + x^n + 1, (n = 1, 2, 3, \cdots, m-1, m \neq 2n)$$

으로 구성한 곱셈기의 경우 $m^2 - 1$ *XOR* 게이트와 m^2 *AND* 게이트가 필요하다는 것을 보였다.

Mastrovito방식으로 나타난 곱셈 행렬은 유한체를 구성하는 삼항 기약다항식에 대해서 일반적인 수식을 이끌어 낼 수 있다. 곱셈 행렬의 각 행에 관한 수식을 살펴보면, 각 행을 구성하는 성분의 요소가 모듈로 감산과정과 관계없이 독립적으로 이루어진 성분과 모듈로 감산 연산에 의해 연관되어 나타난 성분으로 나뉜다. 우리는 곱셈행렬을 모듈러 연산과 관련된 부분과 관련이 없는 부분으로 분할한다. 분할한 행렬에 대해 곱할 원소간의 곱셈연산과 원소간의 덧셈연산 중에 우선순위를 선택한다. 이러한 순위를 이용하면 제안하는 곱셈의 결과 값이 계산 과정에서 병렬구조로 수행될 수 있다. 우선순위에 의해 연산 수행과정을 보다 효율적으로 배치하면 시간의 지연을 개선할 수 있다.

논문의 구성은 다음과 같다. 2장은 Mastrovito 연산의 구체적인 방법을 절로 나누어 설명한다. 2.1절에서는 Mastrovito 곱셈 행렬을 구하기 위한 구성 요소와 정의를 기술하고, 2.2절에서는 본 논문에서 이용할 Mastrovito 곱셈 방법의 원리를 설명한다. 3장은 시간 복잡도 개선을 위한 Mastrovito 곱셈기를 제안하고자 한다. 3.1절에서는 효율적인 비트-병렬 구조의 곱셈기를 제시하고 구성 요소를 설명한다. 3.2절에서는 제안한 곱셈기의 복잡도를 살펴보고 이를 바탕으로 3.3절에서 기존 논문에서 제시된 곱셈의 방법과 제안한 곱셈 방법의 효율성을 비교 분석하고 4장에서 결론을 맺는다.

II. Mastrovito 연산 방법

2.1 구성요소와 정의

$GF(2^m)$은 2^m개의 원소를 가지고 있는 유한체이다. 이 유한체는 $GF(2)$ 위의 m차원 벡터 공간으로 볼 수 있으므로 다항식 기저,

$$B = \{1, x, x^2, x^3, \cdots, x^{m-1}\}$$

을 이용하여 원소들을 살펴보면,

$$GF(2^m) = \{a_{m-1}x^{m-1} + \cdots + a_1 x + a_0 \mid a_i \in GF(2)\}$$

이다. 유한체의 산술 연산 중 덧셈은 비트 별로 수행되기 때문에 *XOR* 게이트로 충분히 표현된다. 유한체 $GF(2^m)$의 두 원소에 대한 곱셈은 다항식 곱셈 단계와 모듈로 감산 연산 단계로 나누어진다.

본 논문에서는 항의 개수가 세 개이면서 기약인 삼항기약다항식 $p(x) = x^m + x^n + 1$ $(m > 2n)$로 만들어진 유한체 $GF(2^m)$의 원소들의 곱셈에 대해 알아본다. 여기서 $m > 2n$임을 가정하여도 일반성을 잃지 않는다.[4] 삼항기약다항식 $p(x)$에 의해서 구성된 $GF(2^m)$의 원소를 $A(x)$와 $B(x)$라고 하고,

$$A(x) = \sum_{i=0}^{m-1} a_i x^i, \quad B(x) = \sum_{i=0}^{m-1} b_i x^i$$

로 표현하자. 여기서 a_i , $b_i \in GF(2)$이다. 두 원소 $A(x) \times B(x)$의 결과를 얻기 위한 과정을 살펴보면 다음과 같다.

■ 다항식의 곱셈:

$C'(x) = A(x) \times B(x)$라 하고 이를 행렬로 표현하면, $c' = M \cdot b$ 이다. 이때,

$$b = [b_0, b_1, \cdots, b_{m-1}]^T, \; c' = [c'_0, c'_1, \cdots, c'_{m-1}]^T$$

이고, $0 \leq i \leq 2m-2$에 대해 c'_i들은 $C'(x)$의 계수이

다. 그리고 M행렬은 다항식 $A(x)$의 원소로 구성된 $(2m-1)\times m$행렬이다.

$$M=\begin{pmatrix} a_0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & \cdots & 0 & 0 \\ a_2 & a_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-1} & a_{m-2} & \cdots & a_1 & a_0 \\ 0 & a_{m-1} & \cdots & a_2 & a_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & a_{m-1} \end{pmatrix}$$

■ 모듈로 감산:

두 다항식 $A(x)$와 $B(x)$의 곱 $C'(x)$에 대한 $p(x)$로의 모듈로 감산 값을

$$C(x)\equiv C'(x) \bmod p(x)$$

라 하자. 이를 수식으로 표현하면,

$$C(x) = \left\{ \sum_{i=0}^{2m-2} c'_i x^i \right\} \bmod p(x)$$

$$= \sum_{i=0}^{m-1} c'_i x^i + \sum_{i=m}^{2m-2} c'_i (x^i \bmod p(x))$$

$m\le i\le 2m-2$에 대해서 $x^m = 1 + x^n \bmod p(x)$인 사실을 이용하여 $x^i \bmod p(x)$를 표현하면,

$$x^{m+k} = x^k + x^{n+k}, \quad 0\le k\le m-n-1$$

$$x^{2m-n+l} = x^{m-n+l} + x^l + x^{n+l}, \quad 0\le l\le n-2$$

의 두 경우로 각각 얻어진다.

위의 곱셈과 모듈로의 두 단계를 하나의 행렬 Z로 $c = Z\cdot b$ 로 단일화할 수 있다[7]. 여기서,

$c=[c_0, c_1, \cdots, c_{m-2}, c_{m-1}]^T$는 $C(x)$의 계수로 이루어진 벡터이다. 행렬 Z를 곱셈 행렬이라 부른다. 이를 구하는 알고리듬은 다음 절에서 설명한다. 우선 곱셈 행렬 Z를 구할 때 사용되는 구성 요소와 몇가지 정의를 살펴 보자. 행렬 M의 i번째 행을 M_i라 하자. $M[\downarrow k]$는 행렬을 아래쪽 방향으로 k만큼 이동하고 빈자리는 0으로 채우는 연산을 의미한다. 예를 들어,

$$M[\downarrow 2]=\begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ a_0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-1} & a_{m-2} & \cdots & a_1 & a_0 \\ 0 & a_{m-1} & \cdots & a_2 & a_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{m-2} & a_{m-3} \end{pmatrix}$$

행렬의 오른쪽 이동을 표현하기 위한 연산 $M[\rightarrow k]$와 행렬의 위쪽 방향의 이동을 표현하기 위한 연산 $M[\uparrow k]$도 $M[\downarrow k]$의 연산과 유사하게 정의할 수 있다. 행렬 X는 행렬 M의 윗 부분인 $m\times m$행렬이고, 행렬 T은 $m\times m$ 행렬로서 M행렬의 아래 부분으로 이루어져 있다. 즉,

$$X=\begin{pmatrix} a_0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & \cdots & 0 & 0 \\ a_2 & a_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-2} & a_{m-3} & \cdots & a_0 & 0 \\ a_{m-1} & a_{m-2} & \cdots & a_1 & a_0 \end{pmatrix},$$

$$T=\begin{pmatrix} 0 & a_{m-1} & \cdots & a_2 & a_1 \\ 0 & 0 & \cdots & a_3 & a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{m-1} & a_{m-2} \\ 0 & 0 & \cdots & 0 & a_{m-1} \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

2.2 Mastrovito 곱셈 방법의 원리

[14]에는 삼항기약다항식 $p(x)$에 대한 Mastrovito 곱셈행렬을 생성하는 방법이 소개되어 있는데 이를 알고리듬화 하면 다음과 같다.

[알고리듬] 곱셈 행렬 M을 구하는 알고리듬

입력: 기약다항식 $p(x)$, 행렬 X, 행렬 T.

출력: 곱셈 행렬 Z

1. $T1\leftarrow T[\uparrow (m-n)]$;
2. $U\leftarrow T[\downarrow n]$;
3. $U1\leftarrow U[\rightarrow (m-n)]$;
4. $Y\leftarrow T+T1+U+U1$;
5. $Z\leftarrow X+Y$;

각 단계의 변수들은 모두 $m\times m$행렬들이다. 마지막 과정에서 X는 기약다항식 $p(x)$의 모듈로 감산이 나타나지 않는 $m-1$차 이하의 항들과 관련되어 있는 행렬이다. Y행렬은 기약다항식 $p(x)$의 모듈로 감산과 $A(x)$의 계수들이 관련되어 생성된다.

[예1] 알고리듬의 적용

$A(x)=a_4x^4+a_3x^3+a_2x^2+a_1x+a_0$, $B(x)=b_4x^4+b_3x^3+b_2x^2+b_1x+b_0$ 이고 기약다항식 $p(x)=x^5+x^2+1$ 일 때, 곱셈 행렬 Z를 구해보자. 우선 M 을 구해보면

$$M=\begin{pmatrix} a_0 & 0 & 0 & 0 & 0 \\ a_1 & a_0 & 0 & 0 & 0 \\ a_2 & a_1 & a_0 & 0 & 0 \\ a_3 & a_2 & a_1 & a_0 & 0 \\ a_4 & a_3 & a_2 & a_1 & a_0 \\ 0 & a_4 & a_3 & a_2 & a_1 \\ 0 & 0 & a_4 & a_3 & a_2 \\ 0 & 0 & 0 & a_4 & a_3 \\ 0 & 0 & 0 & 0 & a_4 \end{pmatrix}$$

이다. 이를 이용하여 곱셈 행렬을 만들기 위해 기약 다항식 $p(x)$에 대한 모듈로 감산의 특징을 살펴 보면,

$$\begin{aligned} x^5 &= x^2+1, \\ x^6 &= x^3+x, \\ x^7 &= x^4+x^2, \\ x^8 &= x^2+1+x^3. \end{aligned}$$

위의 식을 살펴 보면 5차항의 경우 2차항과 상수항에 영향을 주고, 다음 6차, 7차, 8차의 경우도 각각에 해당하는 차수의 계수에 영향을 준다. 즉, 가장 오른쪽의 열을 해석해 보면 M_5행은 곱셈 행렬 Z를 생성할 때 Z_0행에 영향을 주고 M_6행은 Z_1행에 영향을 준다. 그 아래의 행들도 마찬가지로 순차적으로 영향을 준다. 순차적인 행렬들의 특징은 행렬 T로 표현할 수 있다. 즉, M_5행에서 M_7행까지 나타나는 순차적인 행의 영향은 T행렬로 표현된다. 첫번째 모듈로 감산이 수행되는 경우는 T, U행렬로 표현되고, 두번째 모듈로 감산 경우는 $T1$, $U1$행렬로 표현된다. 따라서 Y의 행렬이 모듈로 감산 효과를 표현하는 행렬이라 할 수 있다.

입력: $$X=\begin{pmatrix} a_0 & 0 & 0 & 0 & 0 \\ a_1 & a_0 & 0 & 0 & 0 \\ a_2 & a_1 & a_0 & 0 & 0 \\ a_3 & a_2 & a_1 & a_0 & 0 \\ a_4 & a_3 & a_2 & a_1 & a_0 \end{pmatrix},$$

$$T=\begin{pmatrix} 0 & a_4 & a_3 & a_2 & a_1 \\ 0 & 0 & a_4 & a_3 & a_2 \\ 0 & 0 & 0 & a_4 & a_3 \\ 0 & 0 & 0 & 0 & a_4 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

1. $T1 \leftarrow [T\uparrow 3] = \begin{pmatrix} 0 & 0 & 0 & 0 & a_4 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$.

2. $U \leftarrow T[\downarrow 2] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & a_4 & a_3 & a_2 & a_1 \\ 0 & 0 & a_4 & a_3 & a_2 \\ 0 & 0 & 0 & a_4 & a_3 \end{pmatrix}$.

3. $U1 \leftarrow U[\rightarrow 3] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_4 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$.

4. $Y=\begin{pmatrix} 0 & a_4 & a_3 & a_2 & a_1+a_4 \\ 0 & 0 & a_4 & a_3 & a_2 \\ 0 & a_4 & a_3 & a_2+a_4 & a_1+a_3+a_4 \\ 0 & 0 & a_4 & a_3 & a_2+a_4 \\ 0 & 0 & 0 & a_4 & a_3 \end{pmatrix}$.

5. $Z = X+Y$

$$= \begin{pmatrix} a_0 & a_4 & a_3 & a_2 & a_1+a_4 \\ a_1 & a_0 & a_4 & a_3 & a_2 \\ a_2 & a_1+a_4 & a_0+a_3 & a_4+a_2 & a_3+a_1+a_4 \\ a_3 & a_2 & a_1+a_4 & a_0+a_3 & a_4+a_2 \\ a_4 & a_3 & a_2 & a_1+a_4 & a_0+a_3 \end{pmatrix}.$$

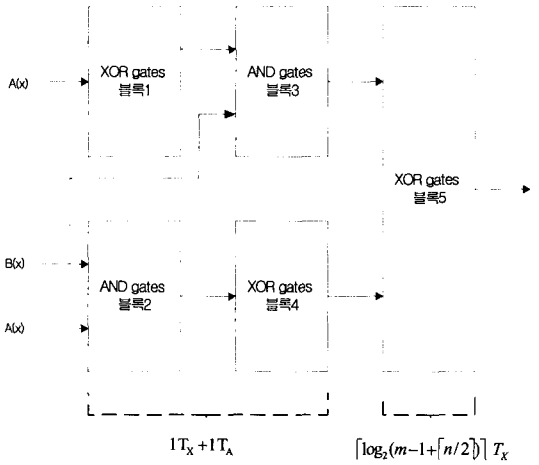
Ⅲ. Mastrovito 곱셈기의 개선

3.1 효율적인 비트-병렬 구조의 곱셈기

2.2절에서 Mastrovito 곱셈 행렬 Z를 구하는 방법을 소개하였다. 일반적으로 유한체에서 두 원소의 곱을 구하는 곱셈기는 곱셈 행렬 Z를 먼저 구한후 b와 곱해서 $c=Z\cdot b$를 계산한다. 그러나 제안하는 곱셈기는 곱셈행렬을 모듈러 연산과 관련된 부분과 관련이 없는 부분으로 분할한다. 분할한 행렬에 대해 곱할 원소 간의 곱셈연산과 원소 간의 덧셈연산 중에 우선순위를 선택하여 시간 복잡도를 개선한다. 행렬 Z는 홀수 개인 원소의 덧셈으로 이루어진 행렬 OD와 짝수 개인 원소의 합으로 구성된 행렬 EV로 나뉜다. 행렬 OD는 행렬 EV의 특정한 부분 원소로 이루어진 부분 행렬 EVP와 그 원소들을 제외한 단일항으로 구성된 행렬 S를 분리할 수 있다. 독립적으로 분리된 행렬들은 각각의 병렬 연산을 수행할 수 있는 계산의 형식을 제시한다.

$$\begin{aligned} Z &= OD+EV \\ &= S+EVP+EV. \end{aligned}$$

구하고자 하는 곱셈 $Z\cdot b$는 각 행의 곱과 덧셈 연산



〔그림 1〕 비트-병렬 구조의 곱셈기 $A(x)\times B(x)$

으로 결합되어 있다. 즉,

$$Z \cdot b = S \cdot b + (EV + EVP) \cdot b.$$

위의 수식은 [그림 1]과 같은 구조로 표현된다.

■ 비트-병렬 구조:

$Z \cdot b$를 구하기 위한 곱셈기는 $S \cdot b$, EV 생성, $(EV+EVP) \cdot b$를 수행하는 부분 그리고 두 수식을 더하는 요소로 구성된다.

3.1.1 블록1과 블록3의 과정

블록1과 블록3은 $(EV+EVP) \cdot b$를 수행하는 과정이다. 곱셈 행렬 Z가 n번째 행인 Z_n으로 생성되므로[14] EV의 n번째 행인 EV_n의 원소만으로 EV와 행렬 EVP를 구성할 수 있다. 즉, 행렬 EV가 일어나는 모듈로 감산 과정은 기약 삼항다항식에 따라 순차적으로 영향을 주기 때문에 모듈로 감산이 시작되는 0행과 n행중에서 행렬 EV를 구성하는 요소가 모두 EV_n에서 나타난다.

EV_n을 구하기 위해 블록1에서 $A(x)$의 원소끼리의 비트별 덧셈이 일어난다. 덧셈이 일어난 원소에 $(EV+EVP)$의 행렬에 따른 b의 곱연산은 블록3에서 일어난다. $(EV+EVP)$의 n행에서 수행되는 모듈로 감산과정의 원소 수는 2차 모듈로 감산과정의 수를 뺀 차로 표현된다. 이는 2차 모듈로 감산과정의 b의 곱셈위치가 중복되어 나타나기 때문이다.

3.1.2 블록2와 블록4의 과정

단일항으로 이루어진 행렬 S에서는 블록1과는 달리 독립적으로 비트 별 곱셈이 일어난다. 즉, $S \cdot b$를 수행한다. 블록2의 결과와 블록3의 결과는 c_i를 계산하기 위해 XOR가 필요하다. 블록1, 블록3과 블록2의 수행시간을 비교하면 소모적으로 기다려야 하는 시간($1T_X$)이 있다. 이를 활용하기 위하여 $S \cdot b$와 $(EV+EVP) \cdot b$를 더하기 위해 구성한 블록5의 연산 중에서 블록2의 출력 값만 해당되는 덧셈을 블록4에서 분산 처리한다.

3.1.3 블록5의 과정

남아있는 원소의 덧셈을 XOR 트리구조를 이용하여 블록5에서 덧셈을 수행한다.

구체적으로 [예1]에서 곱셈행렬 Z는 2.2절의 알고리듬을 통해 구할 수 있고, 행렬 OD와 행렬 EV는 다음과 같이 된다.

$$OD = \begin{pmatrix} a_0 & a_4 & a_3 & a_2 & 0 \\ a_1 & a_0 & a_4 & a_3 & a_2 \\ a_2 & 0 & 0 & 0 & a_3+a_1+a_4 \\ a_3 & a_2 & 0 & 0 & 0 \\ a_4 & a_3 & a_2 & 0 & 0 \end{pmatrix},$$

$$EV = \begin{pmatrix} 0 & 0 & 0 & 0 & a_1+a_4 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & a_1+a_4 & a_0+a_3 & a_4+a_2 & 0 \\ 0 & 0 & a_1+a_4 & a_0+a_3 & a_4+a_2 \\ 0 & 0 & 0 & a_1+a_4 & a_0+a_3 \end{pmatrix}$$

행렬 OD를 행렬 S와 행렬 EVP로 분리하면,

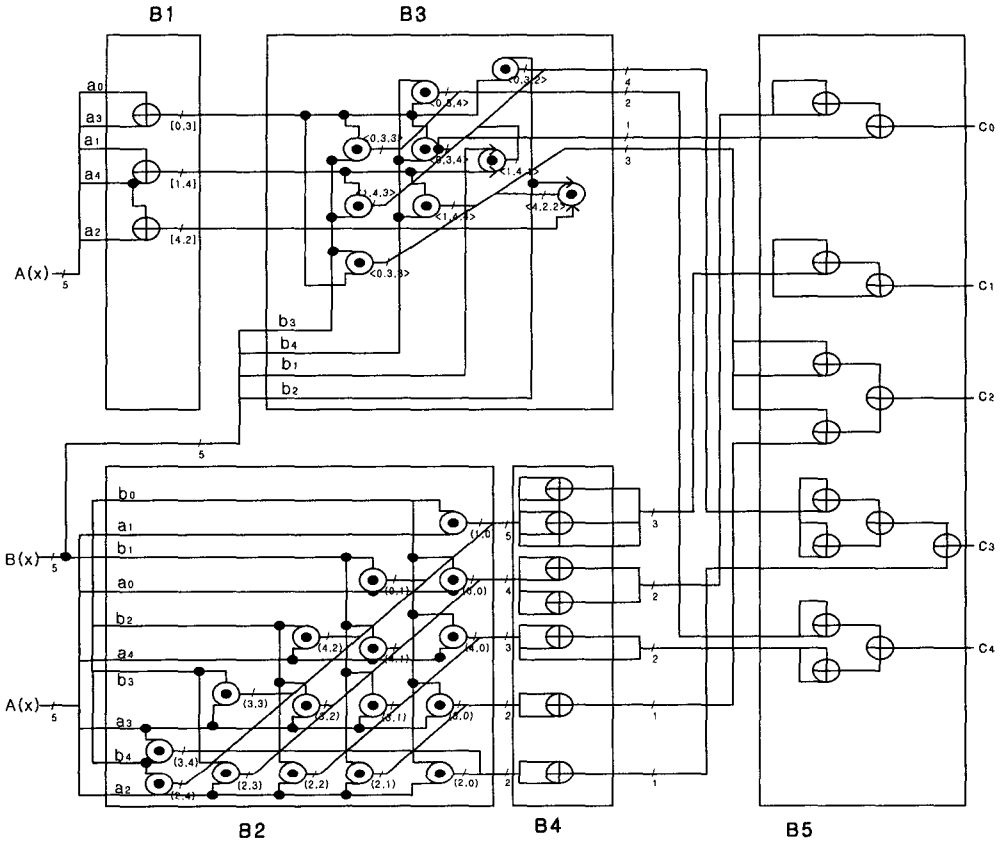
$$S = \begin{pmatrix} a_0 & a_4 & a_3 & a_2 & 0 \\ a_1 & a_0 & a_4 & a_3 & a_2 \\ a_2 & 0 & 0 & 0 & a_3 \\ a_3 & a_2 & 0 & 0 & 0 \\ a_4 & a_3 & a_2 & 0 & 0 \end{pmatrix},$$

$$EVP = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_1+a_4 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

를 얻을 수 있다.

k번째 행 S_k와 $B(x)$계수의 곱인 $a_i \cdot b_i$는 블록2에서 수행되며, 구조의 설명을 간단하게 표현하기 위해 순서쌍 (i,j)으로 항을 표현하자. (그림 2 참조)

$$\begin{cases} (0,0),(4,1),(3,2),(2,3) & \text{if } k=0, \\ (1,0),(0,1),(4,2),(3,3),(2,4) & \text{if } k=1, \\ (2,0),(3,4) & \text{if } k=2, \\ (3,0),(2,1) & \text{if } k=3, \\ (4,0),(3,1),(2,2) & \text{if } k=4. \end{cases}$$



〔그림 9〕 $p(x)=x^5+x^2+1$일 때 구성한 비트 병렬 구조의 곱셈기

행렬 EV를 구성하기 위해 반복적으로 나타나는 항 a_i+a_j의 합을 순서쌍 $[i,j]$으로 표현하면,

$$[1,\ 4],\ [0,\ 3],\ [4,\ 2]$$

이다. 이는 블록1에서 수행된다. 블록3에서는 블록1의 결과와 $B(x)$의 계수와 대응되어 곱셈연산이 일어난다. 수행되는 과정인 $(a_i+a_j)\cdot b_k$를 순서쌍 $\langle i,j,k\rangle$로 나타내면,

$$\langle 1,4,1\rangle,\ \langle 1,4,2\rangle,\ \langle 1,4,3\rangle,\ \langle 1,4,4\rangle,$$
$$\langle 0,3,2\rangle,\ \langle 0,3,3\rangle,\ \langle 0,3,4\rangle,$$
$$\langle 4,2,3\rangle,\ \langle 4,2,4\rangle$$

이다.

블록4에서는 블록2에서 일어난 곱셈결과에 대해 한번의 덧셈이 수행된다. 이는 곱셈 연산기의 수행시간을 효율적으로 사용하기 위한 것이다. 블록5에서 *XOR*트리구조를 이용하여 $A(x)\times B(x)$를 구한다.

3.2 제안하는 곱셈기의 복잡도 분석

제안하는 곱셈기의 복잡도를 살펴 보기 위해 블록 단위로 생각해 보자. 하드웨어의 공간 복잡도를 계산하면, 블록1은 한 번 이상 쓰이는 비트별 덧셈이 수행된다.

이 연산은 행렬 EV_n에서 나타난다. 따라서 블록1의 *XOR*게이트는 $m-n$이다. 블록2의 곱셈은 각각 S행렬의 행마다 나타나므로 *AND*게이트는 행마다 계산되며 그 연산 수는,

$$m^2 - (m-1)m/2 + n(n-1)/2$$

이다. 위의 식은 전체에서 $(EV+EVP)$가 차지하는 원소의 개수를 뺀 값이다. 블록3은 블록1의 결과와 $(EV+EVP)\cdot b$가 일어나므로 *AND* 게이트에 대한 공간복잡도는

$$(m-1)m/2 - (n-1)n/2$$

이다. 블록4에서는 블록2의 결과를 각 행마다 두 개씩 묶어 한 번의 덧셈을 수행한다. 따라서 *XOR*게이트의 수는 처음 0행부터 *n-1*행까지, *n*행부터 *2n-1*행까지 그리고 나머지 행들 별로 차례대로 계산되며 수식은 다음과 같이 나타난다.

$$[n/2]n+\sum_{i=1}^{n}[(m-n+i)/2]+\sum_{i=1}^{m-2n}[(n+i)/2]$$

블록5에서는 블록3의 결과와 블록4의 결과가 덧셈으로 결합된다. 블록5의 *XOR*게이트의 수는 블록4의 수와 블록3에서 채워지는 원소의 개수들에 의한 관계식으로 표현되며 다음과 같다.

$$\sum_{i=1}^{n}\{\lceil(m-n+i)/2\rceil+\lceil n/2\rceil\}+\sum_{i=1}^{m-2n}\{\lceil(n+i)/2\rceil\}+m(m-1)/2-m$$

시간복잡도를 계산하면 블록1과 블록3에서 각각 덧셈과 곱셈을 한 번씩 수행하므로 T_X+T_A이다. 블록5의 시간 복잡도는 블록3와 블록4의 원소의 수가 가장 많은 것에 의존한다. 행 Z_n이 계산하는 경로가 가장 길기 때문에 Z_n을 고려하여 블록5에서의 시간 복잡도를 계산하면

$$\lceil \log_2(m-1+\lceil n/2\rceil)\rceil T_X$$

이다.

3.3 제안하는 곱셈기의 효율성 분석

[표 1]은 기약다항식이 $p(x)=x^m+x^n+1$ 일 때, Sunar[14]가 제안한 곱셈기의 시간 복잡도와 공간 복잡도이다. 제안한 방법의 복잡도는 [표 2]와 같다.

제안된 곱셈기의 *XOR* 공간 복잡도는 Sunar가 제안한 곱셈기보다 *XOR*게이트가 부가적으로 더 소요된다. 증가된 복잡도는 기약다항식의 중간항 n에 의존한다. 그러나 일반적으로 n의 값이 작은 기약다항식을 사용하므로 그 증가율은 적다. [표 3]은 제안한 곱셈기의 개선효과를 살펴 보기 위해 공간 복잡도와 시간 복잡도의 비율을 표준에서 권장하고 있는 기약다항식의 차수에 대해 비교한 표이다. 공간 복잡도의 경우 *AND*게이트의 수가 같으므로 이를 제외한 *XOR* 게이트수를 비교한다. 권장하고 있는 기약다항식에 대해 삼항기약다항식이 존재하지 않을 경우 표준의 가장 근접되어 있는 차수에 대해서 조사하였다. 표3에서 *XOR*공간 복잡도는 평균적으로 1.18% 증가 하는데 비해 시간 복잡도가 9.036% 감소하므로 전체 복잡도 측면에서 개선된다.

〔표 1〕 Sunar의 곱셈기의 복잡도

시간 복잡도	
$T_A+(\lfloor(m-2)/(m-n)\rfloor+1+\lceil\log_2(m)\rceil)T_X$	
공간 복잡도	
AND게이트	m^2
XOR게이트	m^2-1

〔표 2〕 제안된 곱셈기의 복잡도

시간 복잡도	
$T_A+(1+\lceil\log_2(m-1+\lceil n/2\rceil)\rceil)T_X$	
공간 복잡도	
AND게이트	m^2
XOR게이트	$m^2+(n^2-3n)/2$

〔표 3〕 복잡도의 비교

m차	n차	XOR공간 복잡도			시간 복잡도		
		기존[14]	제안	비율(1)	기존[14]	제안	비율(2)
113	9	12768	12796	0.21%	9	8	11.11%
132	17	17423	17543	0.68%	10	9	10.00%
135	11	18224	18269	0.24%	10	9	10.00%
167	6	27888	27898	0.05%	10	9	10.00%
170	11	28899	28944	0.12%	10	9	10.00%
191	9	36480	36508	0.07%	10	9	10.00%
193	15	37248	37339	0.24%	10	9	10.00%
233	74	54288	56916	4.83%	10	10	0%
239	36	57120	57715	1.04%	10	9	10.00%
289	21	83520	83710	0.22%	11	10	9.09%
359	68	128880	131091	1.71%	11	10	9.09%
409	87	167280	170935	2.18%	11	10	9.09%
431	120	185760	192781	3.77%	11	10	9.09%

Ⅳ. 결 론

본 논문에서 시간 복잡도를 줄이기 위한 병렬구조를 가지는 곱셈기를 제안하였다.

기존 논문[14]의 시간 복잡도는,

$T_A+([(m-2)/(m-n)]+1+\lceil \log_2(m) \rceil)T_X$

이었으나, 제안된 방법의 시간 복잡도는

$T_A+(1+\lceil \log_2(m-1+\lceil n/2 \rceil) \rceil)T_X$

으로 개선되었다. 시간 복잡도와 공간 복잡도의 효율성 교환에 의해 공간 복잡도가 다소 늘지만, 시간 복잡도의 개선은 고속 연산을 요구하는 가속기나 전용 하드웨어에 적합할 것으로 보인다.

비율(1)=(제안 *XOR* 복잡도 - 기존 *XOR* 복잡도) / 기존 *XOR* 복잡도

비율(2)=(기존 *XOR* 복잡도 - 제안 *XOR* 복잡도) / 기존 *XOR* 복잡도

참 고 문 헌

[1] J. Guajardo and C. Parr. "*Efficient algorithms for elliptic curve cryptosystem*", Advances in Cryptology CRYPTO'97, Lecture Notes in Com puter Science, No. 1294, pp.342~356, 1997.

[2] G. H. Golub and C. F. van Loan, "*Matrix Computations*", The Johns Hopkins University Press, 1996.

[3] A. Halbutogullari and C. K. Koc. "*Mastrovito multiplier for general irreducible polynomials*", Applied Algebra, Algebraic Algorithms and Error Correcting Codes, Lecture Notes in Computer Science No.1719, pp.498~507, 1999.

[4] IEEE P1363. Standard Specifications for Public Key Cryptography. Institute of Electrical and Electionics Engineers, 2000.

[5] C. K. Koc and B. Sunar. "*Low-complexity bit parallel canonicaland normal basis multipliers for a class of finite fields*", IEEE Transactions on Computers, 47(3):353~356, March 1998.

[6] R. Lidl and H. Niederreiter, "*Introduction to Finite Fields and Their Applications*", New York, Cambridge University Press, 1994.

[7] E. D. Mastrovito, "*VLSI architectures for Computation in Galois Fields*", PhD thesis, Linkoping University, Department of Electrical Engineering, Linkoping, Sweden, 1991.

[8] E. D. Mastrovito, "*VLSI architectures for multiplication over finite field*", In T.Mora,editor, Applied Algebra, Algebraic Algorithms, and Error Correcting Codes, 6th International Conference, AAECC-6, Lecture Notes in Computer Science, No.357, pp. 297~309, Rome, Italy, July 1998.

[9] A. J. Menezes, "*Applications of Finite Fields*," Boston, MA:Kluwer Academic Publishers, 1993.

[10] A. J. Menezes. "*Elliptic Curve Public Key Cryptosystems*", Boston, MA: Kluwer Academic Publishers, 1993.

[11] C. Parr. "*Efficient VLSI Architectures for Bit Parallel Computation in Galois Fields*", PhD thesis, Universitat GH Essen, VDI Verlag, 1994.

[12] C. Parr. "*A New architecture for a parallel finite field multiplier with low complexity based on composite fields*", IEEE Transactions on Computers, 45(7):856-861, July 1996.

[13] SEC2. Recommended Elliptic Curve Domain Parameters, September 20, 2000.

[14] B. Sunar and C. K. Koc, "*Mastrovito Multiplier for All Trinomial*", IEEE Transactions on Computers, 48(5):522~527, May 1999.

[15] ANSI X.9.63-1998, Public Key Cryptography for the Financial Services Industry: Elliptic Curve Digital Signature Algorithm.

〈著者紹介〉

정 석 원 (Seok Won Jung) 정회원
1991년 2월 : 고려대학교 수학과 학사
1993년 2월 : 고려대학교 수학과 석사
1997년 2월 : 고려대학교 수학과 박사
1997년 5월~1997년 11월 : 한국전자통신연구원 박사후연구원
1999년 2월~2001년 2월 : (주)텔리맨 책임연구원
2002년 3월~현재 : 고려대학교 정보보호대학원 조교수
<관심분야> 암호칩 설계, 부채널 공격 방법론, 공개키 암호알고리즘, 디지털 방송 보안

이 선 옥 (Seon Ok Lee) 학생회원
2002년 2월 : 서울 시립대학교 수학과 졸업
2002년 3월~현재 : 고려대학교 정보보호대학원 정보보호학과 석사과정
<관심분야> 공개키 암호이론, 암호칩 설계

김 창 한 (Chang Han Kim) 정회원
1985년 2월 : 고려대학교 수학과 학사
1987년 2월 : 고려대학교 수학과 석사
1992년 2월 : 고려대학교 수학과 박사
2000년 2월~현재 : 세명대학교 인터넷 정보학부 부교수
<관심분야> 정수론, 공개키 암호, 암호 프로토콜