논문 2008-45SD-12-5

삼항 기약다항식을 이용한 $GF(2^n)$의 효율적인 저면적 비트-병렬 곱셈기

(Low Space Complexity Bit Parallel Multiplier For Irreducible Trinomial over $GF(2^n)$)

조 영 인*, 장 남 수*, 김 창 한**, 홍 석 희***

(Young In Cho, Nam Su Chang, Chang Han Kim, and SeokHie Hong)

요 약

유한체 $GF(2^n)$ 연산을 바탕으로 구성되는 암호시스템에서 유한체 곱셈의 효율적인 하드웨어 설계는 매우 중요한 연구 분야이다. 본 논문에서는 공간 복잡도가 낮은 병렬 처리 유한체 곱셈기를 구성하기 위하여 삼항 기약다항식(Trinomial) $f(x)=x^n+x^k+1$의 모듈러 감산 연산 특징을 이용하였다. 또한 연산 수행 속도를 빠르게 개선하기 위해 하드웨어 구조를 기존의 Mastrovito 곱셈 방법과 유사하게 구성한다. 제안하는 곱셈기는 n^2-k^2 개의 AND 게이트와 n^2-k^2+2k-2개의 XOR게이트로 구성되므로 이는 기존의 n^2AND게이트, n^2-1 XOR게이트의 합 $2n^2-1$에서 $2k^2-2k+1$만큼의 공간 복잡도가 감소된 결과이다. 시간 복잡도는 기존의 $T_A+(1+\lceil \log_2(2n-k-1) \rceil)T_X$와 같거나 $1T_X$ 큰 값을 갖는다. 최고차 항이 100에서 1000 사이의 모든 기약다항식에 대해 시간복잡도는 같거나 $1T_X$(10%~12.5%)정도 증가하는데 비해 공간복잡도는 최대 25% 까지 감소한다.

Abstract

The efficient hardware design of finite field multiplication is an very important research topic for and efficient $f(x)=x^n+x^k+1$ implementation of cryptosystem based on arithmetic in finite field $GF(2^n)$. We used special generating trinomial to construct a bit-parallel multiplier over finite field with low space complexity. To reduce processing time, The hardware architecture of proposed multiplier is similar with existing Mastrovito multiplier. The complexity of proposed multiplier is depend on the degree of intermediate term x^k and the space complextity of the new multiplier is $2k^2-2k+1$ lower than existing multiplier's. The time complexity of the proposed multiplier is equal to that of existing multiplier or increased to $1T_X$(10%~12.5%) but space complexity is reduced to maximum 25%.

Keywords : Polynomial Multiplication, Bit-Parallel Multiplier, Mastrovito Multiplication.

* 학생회원, *** 정회원,
고려대학교 정보경영공학전문대학원
(Graduate School of Information Management and Security, Korea University)
** 정회원-교신저자, 세명대학교 정보통신학부
(School of Information & Communication systems, Semyung University)
※ 이 연구에 참여한 연구자(의 일부)는 '2단계BK21사업'의 지원비를 받았음.
접수일자: 2008년7월9일, 수정완료일: 2008년12월2일

I. 서 론

유한체는 공개키 암호시스템에 널리 활용되며, 공개키 암호시스템 중에서 타원곡선 암호시스템(Elliptic Curve Cryptosystem), 페어링 기반의 암호시스템(Pairing Based Cryptosystem) 등은 유한체 연산중 곱셈 연산을 주된 연산으로 한다. 유한체 연산은 주로 덧

셈과 곱셈으로 구성되며, 그 외 지수승, 역원 연산 등은 덧셈과 곱셈의 반복 연산으로 구성된다. 그리고 곱셈 연산은 덧셈 연산에 비하여 시간-공간 자원을 많이 사용한다. 따라서 시스템 설계의 관점에서 효율적인 곱셈기 설계는 매우 중요하다.

유한체 연산에서 곱셈은 주요한 연산 중 하나이고, 유한체 연산의 하드웨어 구현은 공간 복잡도와 시간 복잡도를 통하여 효율성을 비교한다. 따라서 하드웨어에서 곱셈 연산의 복잡도를 효율적으로 구성하는 것은 전체 암호 시스템의 복잡도 감소에 많은 영향을 준다[4]. Leone은 Karatsuba-Ofman Algorithm(KOA) 방법을 적용하여 시간 복잡도와 공간 복잡도의 교환을 이용하여 시간 복잡도는 증가하지만 낮은 공간 복잡도를 가지는 병렬 처리 곱셈기를 제안하였다[8]. Sunar 와 Koc은 1999년에 시간 복잡도를 감소시키기 위한 방법으로 곱셈 행렬을 이용하여 다항식 곱셈과 모듈로 감산 연산을 통합하는 Mastrovito 곱셈 방법을 제시하였다[1]. 그리고 2005년에 Fan과 Dai가 다항식 기저(Polynomial Basis-PB) Q를$\{x^i \mid 0 \le i \le n-1\}$라 할 때 이 다항식 기저에 x^{-v}를 곱한 기저 $x^{-v}Q=\{x^{i-v} \mid 0 \le i \le n-1\}$를 Q의 Shifted Polynomial Basis(SPB)라 정의하고, 이 기저와 Mastrovito 곱셈 방법을 이용하여 공간 복잡도는 기존의 Mastrovito 곱셈 방법과 같고 모든 삼항 기약다항식에서 시간 복잡도가 $T_A+(1+\lceil \log_2 n \rceil)T_X$와 같음을 보였다[6]. 또한, 다시 [9]에서 제안된 시간 지연을 개선하는 방법으로 삼항 기약다항식이 $f(x)=x^n+x^k+1$이고 $k<n/2$일 때 시간 복잡도가 $T_A+(\lceil \log_2(2n-k-1) \rceil)T_X$까지 감소할 수 있음을 보였다[7].

본 논문에서는 다항식 기저를 사용한 곱셈기에 대하여 논하도록 하며 모듈로 감산 연산의 효율성을 고려하여 삼항 기약다항식을 사용한다. 유한체 원소를 분할하여 곱셈을 하는 기존의 KOA 곱셈 방법에서 착안한 것으로 유한체 $GF(2^n)$의 원소를 분할하여 곱셈 연산을 하며 기존의 방법과 같이 시간 복잡도를 줄이기 위해 다항식 곱셈과 감산 연산을 통합하는 Mastrovito 곱셈 방법과 유사하게 곱셈 연산을 실행한다. 제안하는 곱셈 방법을 사용할 경우 $k<n/2$인 모든 삼항 기약다항식에서 공간 복잡도가 기존의 방법에 비해 감소함을 보이고 이로써 곱셈기의 효율성을 제시한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서 $GF(2^n)$의 기존의 곱셈 방법을 설명하고 Ⅲ장에서 새로운 곱셈 방법을 제안한다. Ⅳ장에서는 제안하는 곱셈기의 구조를 설명하고 Ⅴ장에서 제안하는 곱셈기의 복잡도를 보인다. 마지막으로 Ⅵ장에서 기존의 곱셉기와 비교한 결과를 제시하고 결론을 맺는다.

Ⅱ. GF(2ⁿ)의 유한체 곱셈

본 장에서는 기존의 유한체 $GF(2^n)$에서의 곱셈 방법에 대하여 살펴본다. n차 기약다항식 $f(x)$에 의해 생성된 유한체 $GF(2^n)$에서 $x \in GF(2^n)$를 $f(x)$의 근이라 하면 $\{1, x, x^2, ..., x^{n-1}\}$는 $GF(2^n)$의 다항식 기저가 된다. 즉, 유한체 $GF(2^n)$의 원소 A는 $A=a_0+a_1x+a_2x^2+...+a_{n-1}x^{n-1}$, $a_i \in GF(2)$와 같이 유일하게 표현된다. A의 계수 벡터를 $a=[a_0, a_1, ..., a_{n-1}]^T$와 같이 표현하면

$$A=X^Ta \ , \ X=[1, x, x^2, ..., x^{n-1}]^T$$

로 쓸 수 있다. 유한체 $GF(2^n)$의 원소 A와 B의 다항식 곱셈을 하면 다음과 같이 $2n-2$차의 다항식 S로 나타나며

$$S=A\times B=s_0+s_1x+...+s_{2n-2}x^{2n-2},$$

$s_i \in GF(2^n)$ 이다. 그러면 A와 B의 곱 C는 다음과 같이 모듈로 감산 연산으로 얻을 수 있다.

$$C=\sum_{i=0}^{n-1} c_ix^i \equiv S \bmod f(x).$$

본 논문에서 사용하는 기호를 정리하면 다음과 같다.

■ 기호

계수 벡터 $v=[v_0, v_1, ..., v_{n-1}]$, $w=[w_0, w_1, ..., w_{n-1}]$와 $(n\times m)$ 행렬 M에 대하여

- $(v \rightarrow i)$: 벡터 v를 오른쪽으로 i만큼 이동하고 빈 위치에 0을 채움. 즉, $(v \rightarrow 3)=[0, 0, 0, v_0, v_1, ..., v_{n-4}]$와 같이 표현함.
- $(v \leftarrow j)$: 벡터 v를 왼쪽으로 j만큼 이동하고 빈 위치에 0을 채움.
- M_i : 행렬 M의 i행.
- $M_{i...j}$: 행렬 M의 i부터 j행.
- v_s : 벡터 v의 모든 원소의 합, $v\_s=\sum_{i=0}^{n-1} v_i$.

- $v^{(j)}$: v의 j번 째 원소.
- $v \wedge w$: $[v_0 \times w_0, v_1 \times w_1, \ldots, v_{n-1} \times w_{n-1}]$, $\times$는 $GF(2)$의 곱셈.
- $(M \wedge v)$: M의 각 행에 v를 $\wedge$ 연산한 행렬.
- $(M \wedge v)\_s = [\sum_{j=0}^{m-1} M_0^{(j)} \times v^{(j)}, \sum_{j=0}^{m-1} M_1^{(j)} \times v^{(j)}, \ldots, \sum_{j=0}^{m-1} M_{n-1}^{(j)} \times v^{(j)}]^T$.

1. Polynomial Basis Mastrovito 곱셈 방법

본 절에서는 다항식 곱셈 단계와 모듈로 감산 연산 단계를 통합한 Mastrovito 곱셈 방법에 대하여 살펴본다. 우선 유한체 $GF(2^n)$의 원소 A와 B의 다항식 곱 S의 계수벡터 $[s_0, s_1, \ldots, s_{2n-2}]^T$를 나타내는 행렬을 구성한다.

$$M = \begin{pmatrix} a_0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & 0 & 0 & \cdots & 0 & 0 \\ a_2 & a_1 & a_0 & 0 & \cdots & 0 & 0 \\ \vdots & & \vdots & & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & a_{n-4} & a_{n-5} & \cdots & a_0 & 0 \\ a_{n-1} & a_{n-2} & a_{n-3} & a_{n-4} & \cdots & a_1 & a_0 \\ 0 & a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_2 & a_1 \\ \vdots & & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_{n-1} & a_{n-2} \\ 0 & 0 & 0 & 0 & \cdots & 0 & a_{n-1} \end{pmatrix}$$

이 행렬을 M이라 하면 이 행렬 A의 계수 벡터 a의 원소로 구성되며 B의 계수 벡터를 $b = [b_0, b_1, \ldots, b_{n-1}]^T$라 할 때 $s_i = M_i \cdot b$이다.

이제 다항식 A와 B의 다항식 곱 S를 $f(x) = x^n + x^k + 1$로 모듈로 감산한 값 C는 $n \le i \le 2n-2$에 대하여 $x^n = x^k + 1 \bmod f(x)$이므로 $x^i \bmod f(x)$는

$$x^{n+r} = x^{k+r} + x^r \ (0 \le r \le n-k-1),$$
$$x^{2n-k+s} = x^{n-k+s} + x^{k+s} + x^s \ (0 \le s \le k-2),$$

인 두 경우를 얻을 수 있다. 즉, $x^i (n \le i \le 2n-2)$ 항들의 계수들이 모듈로 감산 연산에 의해 위의 두 가지 경우로 아래와 같이 해당 $n-1$차 이하 항의 계수에 더해진다. 따라서

$$s_{k+r} \Leftarrow s_{k+r} + s_{n+r} \ (0 \le r \le n-k-1),$$
$$s_r \Leftarrow s_r + s_{n+r} \ (0 \le r \le n-k-1),$$
$$s_{n-k+s} \Leftarrow s_{n-k+s} + s_{2n-k+s} \ (0 \le s \le k-2),$$
$$s_{k+s} \Leftarrow s_{k+s} + s_{2n-k+s} \ (0 \le s \le k-2),$$
$$s_s \Leftarrow s_s + s_{2n-k+s} \ (0 \le s \le k-2)$$

와 같이 된다. $c = [c_0, c_1, \ldots, c_{n-2}, c_{n-1}]^T$를 C의 계수 벡터라 할 때 위의 곱셈과 모듈로 감산 연산 두 단계를 하나의 행렬 $c = Z \cdot b$으로 통합할 수 있다[10]. 이 때 행렬 Z를 곱셈 행렬이라 부르며 이 행렬은 M을 이용하여 다음과 같이 구할 수 있다.

$$M_{k+r} \Leftarrow M_{k+r} + M_{n+r} \ (0 \le r \le n-k-1),$$
$$M_r \Leftarrow M_r + M_{n+r} \ (0 \le r \le n-k-1),$$
$$M_{n-k+s} \Leftarrow M_{n-k+s} + M_{2n-k+s} \ (0 \le s \le k-2),$$
$$M_{k+s} \Leftarrow M_{k+s} + M_{2n-k+s} \ (0 \le s \le k-2),$$
$$M_s \Leftarrow M_s + M_{2n-k+s} \ (0 \le s \le k-2).$$

행렬 M의 각 행이 위의 연산을 수행하고 나면 M의 n행 이하의 행은 모두 0이 되므로 $(n \times n)$행렬인 곱셈 행렬 Z가 생성된다.

2. Shfited Polynomial Basis Mastrovito 곱셈 방법

SPB를 Mastrovito 곱셈 방법에 적용할 경우 기존과 비교하여 곱셈 행렬 Z를 생성함에 있어 $1 < k < n/2$일 때 최단 지연 경로(Critical Path Delay)를 줄여서 $1T_X$의 시간 지연을 줄일 수 있다. 그리고 [9]에서 제안된 방법으로 기약다항식의 중간 항 차수 k에 따라 시간 복잡도를 더욱 줄일 수 있다. 감산 연산시 C의 계수 벡터를 구할 때 XOR 연산으로 인한 시간을 최소화 할 수 있어 기존의 결과들 중 공간 복잡도가 PB Mastrovito 곱셈 방법과 같을 때 가장 낮은 시간 복잡도를 갖는다. 자세한 내용은 [6~7]에 나와 있으며 본 논문 PB 기반의 병렬 곱셈기에 대하여 기술하므로 이를 생략한다.

III. 제안하는 곱셈 방법

본 장에서는 제안하는 곱셈 방법을 설명한다. 제안하는 곱셈 방법을 사용할 경우 모듈로 감산 연산에 의해 상쇄되는 원소가 많아 실제 연산해야할 양이 감소함을 보인다. 그리고 다항식 곱셈과 모듈로 감산 연산의 단계로 나뉘어져 있는 과정을 통합하기 위해 앞에서 설명한 Mastrovito 곱셈 방식과 유사하게 곱셈 행렬을 이용하여 시간 지연을 줄인다. 본 논문에서는 모듈로 감산

연산을 위해 중간항의 차수가 최고차 항의 차수의 절반보다 작은 삼항 기약다항식을 이용하고 있다. 즉, 삼항 기약다항식이 $f(x)=x^n+x^k+1$라 할 때, $1\le k<n/2$인 $f(x)$를 이용한다. $k=n/2$인 경우 기존의 KOA방법과 같고, $k>n/2$인 경우는 reciprocal성질에 의하여 $f(x)=x^n+x^{n-k}+1$인 기약다항식이 항상 존재하여 효율성을 고려하여 $k<n/2$인 k를 사용하므로 $k>n/2$인 경우는 본 논문에서 논의하지 않는다[5].

■ 정의

- $a^L=[a_0,a_1,...,a_{k-1}]$.
- $a^H=[a_k,a_{k+1},...,a_{n-1}]$.
- $\overline{a_{n-k+i}}=a_{n-k+i}+a_i,\ i=0,1,...,k-1$.
- $a^S=[a_k,a_{k+1},...,a_{n-k-1},\overline{a_{n-k}},...,\overline{a_{n-1}}]$.
- $a^L(x)=0\times x^{-n+2k}+0\times x^{-n+2k+1}+...+0\times x^{-1}+a_0+a_1x+...+a_{k-1}x^{k-1}$.
- $a^H(x)=a_kx^{-n+2k}+a_{k+1}x^{-n+2k+1}+...+a_{n-1}x^{k-1}$.
- $X1=[1,x,...,x^{n+k-2}]$.
- $X2=[x^{-n+3k},x^{-n+3k+1},...,x^{2n-2}]$.
- $X3=[x^{-n+3k},x^{-n+3k+1},...,x^{n+k-2}]$.

(1) 분할과 곱셈 방법

유한체 $GF(2^n)$의 원소 A를 $n-k$개의 항을 갖는 $a^L(x)$와 $a^H(x)x^{n-k}$으로 분할하되 $a^L(x)$에 포함된 존재하지 않는 항들에 대해서는 계수를 0으로 하여 정의에 의하여 다음과 같이 분할한다.

$$\begin{aligned}A=&(0\times x^{-n+2k}+0\times x^{-n+2k+1}+...+0\times x^{-1}\\&+a_0+a_1x+...+a_{k-1}x^{k-1})\\&+(a_kx^{-n+2k}+a_{k+1}x^{-n+2k+1}+\\&...+a_{n-1}x^{k-1})x^{n-k}\\=&\ a^L(x)+a^H(x)x^{n-k}.\end{aligned}$$

이때, $k<n/2$이므로 기존 KOA와 달리 항상 k에 의존하는 비 균등한 분할이 되어 시간-공간의 trade-off 시 발생하는 시간 지연의 증가를 삼항 기약다항식의 특징을 이용하여 최소화한다.

유한체 $GF(2^n)$의 원소 A와 B의 다항식 곱셈 S는 다음과 같이 표현할 수 있다. $S1$, $S2$, $S3$를

$$\begin{aligned}S1&=a^L(x)b^L(x)+a^L(x)b^L(x)x^{n-k},\\S2&=a^H(x)b^H(x)x^{n-k}+a^H(x)b^H(x)x^{2n-2k},\\S3&=(a^L(x)+a^H(x))(b^L(x)+b^H(x))x^{n-k}\end{aligned}$$

라 정의하면 $S=A\times B$는 $S1$, $S2$, $S3$에 의하여 다음과 같다.

$$\begin{aligned}S&=A\times B\\&=a^L(x)b^L(x)+((a^L(x)b^H(x)+a^H(x)b^L(x))x^{n-k}\\&\quad+a^H(x)b^H(x)x^{2n-2k}\\&=a^L(x)b^L(x)+((a^L(x)+a^H(x))(b^L(x)+b^H(x))\\&\quad+a^L(x)b^L(x)+a^H(x)b^H(x))x^{n-k}+a^H(x)b^H(x)x^{2n-2k}\\&=S1+S2+S3\ .\end{aligned}$$

따라서 $S1,S2,S3$의 계수를 구하기 위한 행렬을 각각 $M1,M2,M3$라 하면

$$s1_i=M1_i\bullet b^L,\ s2_i=M2_i\bullet b^H,\ s3_i=M3_i\bullet b^S$$

이다. $M1$은 $(n+k-1)\times k$행렬, $M2$는 $(3n-3k-1)\times(n-k)$행렬, $M3$는 $(2n-2k-1)\times(n-k)$행렬이며 이들은 다음과 같다.

$$M1=\begin{pmatrix}a_0&0&0&\cdots&0\\a_1&a_0&0&\cdots&0\\\vdots&\vdots&\ddots&\ddots&\vdots\\a_{k-2}&a_{k-3}&\cdots&a_0&0\\a_{k-1}&a_{k-2}&\cdots&a_1&a_0\\0&a_{k-1}&\cdots&a_2&a_1\\\vdots&\vdots&\ddots&\vdots&\vdots\\a_0&0&\cdots&a_{k-1}&a_{k-2}\\a_1&a_0&\cdots&0&a_{k-1}\\\vdots&\vdots&\ddots&\vdots&\vdots\\a_{k-2}&a_{k-3}&\cdots&a_0&0\\a_{k-1}&a_{k-2}&\cdots&a_1&a_0\\0&a_{k-1}&\cdots&a_2&a_1\\\vdots&\vdots&\ddots&\vdots&\vdots\\0&0&\cdots&a_{k-1}&a_{k-2}\\0&0&\cdots&0&a_{k-1}\end{pmatrix},$$

$$M2=\begin{pmatrix}a_k&\cdots&0&\cdots&0\\\vdots&\ddots&\vdots&\ddots&\vdots\\a_{n-k}&\cdots&a_k&\cdots&0\\\vdots&\ddots&\vdots&\ddots&\vdots\\a_{n-1}&\cdots&a_{n-k}&\cdots&a_k\\a_k&\ddots&\vdots&\ddots&\vdots\\\vdots&\ddots&a_{n-1}&\cdots&a_{n-k}\\a_{n-k}&\cdots&a_k&\ddots&\vdots\\\vdots&\ddots&\vdots&\ddots&a_{n-1}\\a_{n-1}&\cdots&a_{n-k}&\cdots&a_k\\\vdots&\ddots&\vdots&\ddots&\vdots\\0&\cdots&a_{n-1}&\cdots&a_{n-k}\\\vdots&\ddots&\vdots&\ddots&\vdots\\0&\cdots&0&\cdots&a_{n-1}\end{pmatrix},$$

$$M3=\begin{pmatrix} a_k & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \underline{a_{n-k}} & \cdots & a_k & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \underline{a_{n-1}} & \cdots & \underline{a_{n-k}} & \cdots & a_k \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \underline{a_{n-1}} & \cdots & \underline{a_{n-k}} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \underline{a_{n-1}} \end{pmatrix}.$$

(2) 곱셈 행렬

본 절에서는 $M1, M2, M3$를 이용하여 곱셈 행렬을 구하는 과정과 그 과정 중에 $GF(2)$에서 상쇄되어 연산할 필요가 없는 부분을 제거하는 과정을 설명한다. 이로 인하여 실제 계산해야할 계수들이 줄어듬을 볼 수 있다. 먼저 위에서 구한 행렬 $M1, M2, M3$를 분할 방법에 따른 영역으로 나눈다. $M1$은 $M1^L$와 $M1^H$ 로 나누는데 이는 $S1$의 두 항에 따라 나눈 것이다.

$$M1_i^L=\begin{cases}[a_i,...,a_0,0,...,0], & (0\le i\le k-1)\\ M1_{k-1}^L(\rightarrow i-(k-1)), & (k\le i\le 2k-2),\end{cases}$$

$$M1_i^H=M1_i^L \ (0\le i\le 2k-2).$$

마찬가지로 $M2$도 아래와 같이 나눈다.

$$M2_i^L=\begin{cases}[a_{k+i},...,a_k,0,...,0], & (0\le i\le n-k-1)\\ M2_{n-k-1}^L(\rightarrow i-(n-k-1)), & (n-k\le i\le 2n-2k-2),\end{cases}$$

$$M2_i^H=M2_i^L \ (0\le i\le 2n-2k-2).$$

$M1^H, M2^H$는 각각 $M1, M2$의 $n-k$행부터 시작된다. 이때, $M2_i=M3_i\,(0\le i\le n-2k-2)$이므로 다항식 곱셈 $S=S1+S2+S3=X1\cdot(M1\cdot b^L)+X2\cdot(M2\cdot b^H)+X3\cdot(M3\cdot b^S)$를 구하는 과정에서 $GF(2)$의 연산인 것을 감안하여 상쇄되는 부분이다. 따라서 이 부분은 불필요한 연산으로 제거 가능하고 편의를 위해

$$\begin{aligned}s_i&=s1_i+s2_i+s3_i\\&=\begin{cases}M1_i\cdot b^L+M2_i\cdot b^H+M3_i\cdot b^S, & (0\le i\le n+k-2)\\ M2_i\cdot b^H, & (n+k-1\le i\le 2n-2)\end{cases}\end{aligned}$$

로 표현 될 수 있도록 $M2, M3$행렬을 위 쪽 방향으로 $n-3k$만큼 이동하고 빈자리는 제거한다. 그러면

$$M2_i^L=\begin{cases}[a_{i+(n-2k)},...,a_k,0,...,0], & (k\le i\le 2k-1)\\ M2_{(2k-1)}^L(\rightarrow i-(2k-1)), & (2k\le i\le n+k-2),\end{cases}$$

$$M2_i^H=\begin{cases}[a_{k+i},...,a_k,0,...,0], & (-n+3k\le i\le k-1)\\ M2_i^H=M2_i^L, & (k\le i\le n+k-2)\end{cases}$$

이므로 $M2, M3$는 다음과 같다.

$$M2_i=\begin{cases}[0,...,0], & (0\le i\le k-1)\\ M2_i^L, & (k\le i\le 2k-1)\\ M2_i^L+M2_{i-(n-k)}^H, & (2k\le i\le n+k-2)\\ M2_{i-(n-k)}^H, & (n+k-1\le i\le 2n-2)\end{cases}$$

$$M3_i=\begin{cases}[\overline{a_{i+(n-2k)}},...,a_k,0,...,0], & (k\le i\le 2k-1)\\ M3_{(2k-1)}(\rightarrow i-(2k-1)), & (2k\le i\le n+k-2).\end{cases}$$

이제 행렬 $M1, M2, M3$을 이용하여 곱셈 행렬 $Z1, Z2, Z3$를 구해보자. 이는 각 행렬의 n행 이상의 행들을 사용하는 삼항 기약다항식에 따라 각 행렬의 $n-1$행 이하의 해당 행에 더해주고 n행 이상의 행들을 제거함으로써 얻을 수 있다. 그러면 각 행렬은 모두 n행을 갖는 곱셈 행렬이 된다. 따라서

$$Z1_i=\begin{cases}M1_i+M1_{n+i}=M1_i^L+M1_{k+i}^H, & (0\le i\le k-2)\\ M1_i=M1_i^L, & (i=k-1)\\ M1_i+M1_{i+(n-k)}, & (k\le i\le 2k-2)\\ M1_i=M1_{i-(n-k)}^L, & (n-l\le i\le n-1)\end{cases} \quad (1)$$

와 같다. 그런데 (1)에서 $M1_{i+(n-k)}=M1_i^L=M1_i$이므로 $M1_i+M1_{i+(n-k)}\,(k\le i\le 2k-2)$는 $GF(2)$에서 연산할 필요가 없으므로 $Z1_i=[0,...,0]\ (k\le i\le n-k-1)$이 된다. 결과적으로 $Z1$은 $M1^L$으로만 이루어진다.

$$Z1_i=\begin{cases}M1_i^L+M1_{k+i}^H=M1_i^L+M1_{k+i}^L, & (0\le i\le k-2)\\ M1_i^L, & (i=k-1)\\ M1_{i-(n-k)}^L, & (n-k\le i\le n-1).\end{cases}$$

또한, $M2$에서는 $M2_{n\cdots 2n-2}$행이 감산되어야 하는데 여기에서 $M2_{n\cdots 2n-2}$는 $M2_{n\cdots n+k-2}^L$와 $M2_{k\cdots n+k-2}^H$ 로 이루어져 있고 먼저 $M2_{k\cdots n+k-2}^H$를 감산하기 위해서 $M2_i^L+M2_i^H\ (k\le i\le n+k-2)$를 계산하게 되는데 이때 $M2_i^L=M2_i^H\,(k\le i\le n+k-2)$이므로 이 부분 역시 $GF(2)$에서 연산할 필요가 없어서 제거 가능하다. 이때, $M2_{n\cdots n+k-2}^L$도 함께 상쇄되므로 나머지 $M2_i+M2_{k+i}^H\ (0\le i\le n-2)$의 감산 연산이 진행되면 $Z2$에는 아래와 같이 $M2^L$는 모두 제거되고 $M2^H$만 남는다.

$$Z2_i=\begin{cases}M2_{k+i}^H, & (0\le i\le 2k-1)\\ M2_{i-(n-k)}^H+M2_{k+i}^H, & (2k\le i\le n-2)\\ M2_{k-1}^H, & (i=n-1).\end{cases}$$

마지막으로 $Z3$를 생성하기 위해 $M3_{n\cdots n+k-2}$가 감산되어야 하며

$$Z3_i=\begin{cases}M3_{n+i}, & (0\le i\le k-2)\\ [0,...,0], & (i=k-1)\\ M3_i+M3_{i+(n-k)}, & (k\le i\le 2k-2)\\ M3_i, & (2k-1\le i\le n-1)\end{cases}$$

와 같이 생성된다. 각 $Z1, Z2, Z3$은 다음과 같다.

$$Z1=\begin{pmatrix} a_0 & a_{k-1} & \cdots & a_1\\ \vdots & \ddots & \ddots & \vdots\\ a_{k-2} & \cdots & a_0 & a_{k-1}\\ a_{k-1} & \cdots & a_1 & a_0\\ 0 & \cdots & 0 & 0\\ \vdots & \ddots & \vdots & \vdots\\ a_0 & \cdots & 0 & 0\\ \vdots & \ddots & \vdots & \vdots\\ a_{k-2} & \cdots & a_0 & 0\\ a_{k-1} & \cdots & a_1 & a_0\end{pmatrix},$$

$$Z2=\begin{pmatrix} a_{n-k} & \cdots & a_k & 0 & \cdots & 0\\ \vdots & \ddots & \vdots & \ddots & \ddots & \vdots\\ a_{n-2} & \cdots & a_{2k-2} & \cdots & a_k & 0\\ a_{n-1} & \cdots & a_{2k-1} & \cdots & a_{k+1} & a_k\\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots\\ 0 & \cdots & a_{n-1} & \cdots & a_{2k+1} & a_{2k+2}\\ a_k & \cdots & 0 & a_{n-1} & \cdots & a_{2k+1}\\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots\\ a_{n-k-2} & \cdots & a_k & 0 & \cdots & a_{n-1}\\ a_{n-k-1} & \cdots & a_{k+1} & a_k & \cdots & 0\end{pmatrix},$$

$$Z3=\begin{pmatrix} 0 & \cdots & 0 & \overline{a_{n-1}} & \cdots & \overline{a_{n-k+1}}\\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots\\ 0 & \cdots & 0 & 0 & \cdots & \overline{a_{n-1}}\\ 0 & \cdots & 0 & 0 & \cdots & 0\\ \overline{a_{n-k}} & \cdots & a_k & \overline{a_{n-1}} & \cdots & \overline{a_{n-k+1}}\\ \vdots & \ddots & \vdots & \ddots & \ddots & \vdots\\ \overline{a_{n-2}} & \cdots & a_{2k-2} & \cdots & a_k & \overline{a_{n-1}}\\ \overline{a_{n-1}} & \cdots & a_{2k-1} & \cdots & a_{k+1} & a_k\\ \vdots & \ddots & \overline{\vdots} & \ddots & \overline{\vdots} & \vdots\\ 0 & \cdots & \overline{a_{n-1}} & \cdots & \overline{a_{n-k+1}} & \overline{a_{n-k}}\end{pmatrix}.$$

예제) 삼항 기약다항식이 $f(x)=x^5+x^2+1$이고, $A(x)=a_0+a_1x+a_2x^2+a_3x^3+a_4x^4$, $B(x)=b_0+b_1x+b_2x^2+b_3x^3+b_4x^4$일 때, 곱셈 행렬 $Z1, Z2, Z3$는 불필요한 원소 a_2와 $M1^H, M2^L$를 제외하고, 앞에서 설명한 것처럼 아래와 같이 구할 수 있다.

$$Z_1=\begin{pmatrix} a_0 & a_1\\ a_1 & a_0\\ 0 & 0\\ a_0 & 0\\ a_1 & a_0\end{pmatrix},\ Z_2=\begin{pmatrix} a_3 & a_2 & 0\\ a_4 & a_3 & a_2\\ 0 & a_4 & a_3\\ 0 & 0 & a_4\\ a_2 & 0 & \overline{0}\end{pmatrix},\ Z_3=\begin{pmatrix} 0 & 0 & \overline{a_4}\\ \overline{0} & 0 & \overline{0}\\ \overline{a_3} & \overline{a_2} & \overline{a_4}\\ \overline{a_4} & \overline{a_3} & a_2\\ 0 & \overline{a_4} & \overline{a_2}\end{pmatrix}$$

IV. 제안하는 곱셈기 구조

앞 장에서 곱셈 행렬 $Z1, Z2, Z3$을 구하는 방법을 살펴보았다. 일반적인 곱셈기는 곱셈 행렬 Z를 구한 후 $Z\bullet b$를 수행하여 결과 벡터를 구한다. 제안하는 곱셈기는 곱셈 행렬 Z의 원소 $Z_i^{(j)}$에 b_j 또는 $\overline{b_j}$를 곱한 후 공간 복잡도를 줄이기 위하여 세 개의 XOR 트리를 구성하여 c을 구한다. 이 세 개의 XOR 트리를 구성하기 위해 정리1에 의하여 $c=Z1\bullet b^L+Z2\bullet b^H+Z3\bullet b^S$ 대신 $c=(Z1\wedge b^L)_s+(Z2\wedge b^H)_s+(Z3\wedge b^S)_s$를 계산하기로 한다.

정리1. 행렬 Z가 $n\times m$행렬일 때 C의 계수 벡터는 $c=Z\bullet b=(Z\wedge b)_s$이다.

증명) $c=Z\bullet b$이면 $c_i=Z_i\bullet b=\sum_{j=0}^{m-1}(Z_i^{(j)}\times b_j)=[Z_i^{(0)}\times b_0, Z_i^{(1)}\times b_1, ..., Z_i^{(m-1)}\times b_{m-1}]_s = [Z_i\wedge b]_s$이므로 $c=(Z\wedge b)_s$이다.□

■ 비트-병렬 구조

$c=(Z1\wedge b^L)_s+(Z2\wedge b^H)_s+(Z3\wedge b^S)_s$을 구하기 위한 곱셈기는 7개의 블록으로 구성되며 개략적인 구조는 다음과 같다. 제안하는 곱셈기는 $Z1\wedge b^L$를 계산하는 블록1, $Z3$와 b^S를 생성하기 위해 사전계산을 하는 블록2와 사전계산이 끝난 후 $Z3\wedge b^S$ 을 계산하는 블록3, $Z2\wedge b^H$를 계산하는 블록4, 그리고 블록2,3이 계산하는 동안 블록1,4의 원소를 더하는 블록5와 중복 데이터 활용을 위하여 세 개의 트리를 형성하여 계산하는 블록6, 그리고 그 결과를 더하여 C를 구하는 블록7로 구성된다.

(1) 블록1의 과정

블록1에서는 $Z1\wedge b^L$를 계산한다.

$$Z1_i\wedge b^L=\begin{cases}(M1_i^L+M1_{k+i}^L)\wedge b^L, & (0\le i\le k-2)\\ M1_i^L\wedge b^L, & (i=k-1)\\ M1_{i-(n-k)}^L\wedge b^L, & (n-k\le i\le n-1).\end{cases}\quad (2)$$

이때 (2)에서 $(M1_i^L+M1_{k+i}^L)\wedge b^L$는 $(M1_i^L\wedge b^L)+(M1_{k+i}^L\wedge b^L)$로 표현 할 수 있는데 이 중 $(M1_i^L\wedge b^L)$ $(0\le i\le k-1)$과 $M1_{i-(n-k)}^L\wedge b^L (n-k\le i\le n-1)$이 동일하다.

(2) 블록2와 블록3의 과정

블록2는 $Z3$와 b^S의 원소들을 곱하기 전 사전 계산을 하는 과정으로써 $\overline{a_{n-k+i}} = a_{n-k+i} + a_i$, $i=0,1,...,k-1$, $\overline{b_{n-k+i}} = b_{n-k+i} + b_i$, $i=0,1,...,k-1$을 생성하기 위해 XOR 연산을 하게 된다.

블록3은 블록2에서 $Z3$와 b^S가 생성되면 $Z3 \wedge b^S$를 계산하는 과정이다.

$$Z3_i \wedge b^S = \begin{cases} M3_{n+i} \wedge b^S, & (0 \le i \le k-2) \\ [0,...,0], & (i=k-1) \\ (M3_i + M3_{i+(n-k)}) \wedge b^S, & (k \le i \le 2k-2) \\ M3_i \wedge b^S, & (2k-1 \le i \le n-1). \end{cases} \quad (3)$$

(3)에서 $(M3_i + M3_{i+(n-k)}) \wedge b^S$은 $(M3_i \wedge b^S) + (M3_{i+(n-k)} \wedge b^S)$와 같이 쓸 수 있고 이중 $(M3_{i+(n-k)} \wedge b^S)(k \le i \le 2k-2)$이 $M3_{n+i} \wedge b^S (0 \le i \le k-2)$와 동일하다.

(3) 블록4의 과정

블록4는 $Z2 \wedge b^H$를 계산한다.

$$Z2_i \wedge b^H = \begin{cases} M2^H_{k+i} \wedge b^H, & (0 \le i \le 2k-1) \\ (M2^H_{i-(n-k)} + M2^H_{k+i}) \wedge b^H, & (2k \le i \le n-2) \\ M2^H_{k-1} \wedge b^H, & (i=n-1) \end{cases} \quad (4)$$

(4)에서 $Z2_0 \wedge b^H$를 살펴보면 $M2^H_k \wedge b^H = [a_{n-k} \times b_k, a_{n-k-1} \times b_{k+1}, ..., a_k \times b_{n-k}]$로써 $M3_k \wedge b^S = [\overline{a_{n-k}} \times b_k, a_{n-k-1} \times b_{k+1}, ..., a_k \times \overline{b_{n-k}}]$와 처음과 끝 원소를 제외하고 동일함을 알 수 있다. 따라서 공간 복잡도를 최소화하기 위해 중복되는 일부 원소를 재사용할 수 있다. $M2^H_{k+i} \wedge b^H$와 $M3_{k+i} \wedge b^S$에 동일한 부분은 $Z2_{0...n-2k-2}$와 $Z3_{k...n-k-2}$에 나타난다. 이들을 각각 $(M2^H_{k+i} \wedge b^H)'$ $(M3_{k+i} \wedge b^S)'$ $(0 \le i \le n-2k-2)$라 하자.

(4) 블록5의 과정

블록2,3이 사전 계산과 곱셈 연산을 하고 나서 블록1,4의 결과와 더해서 c_i를 구하기 위한 XOR연산을 해야 하는데 블록2,3의 결과가 생성되기 까지는 $1T_A + 1T_X$의 시간이 필요하므로 블록1,4에서는 곱셈 연산을 하고 난 후 $1T_X$의 시간이 소모될 수 있다. 따라서 이 시간지연을 활용하기 위해서 블록1,4의 결과가 블록2,3과 통합되어 새로 분류되기 전에 한번의 XOR연산을

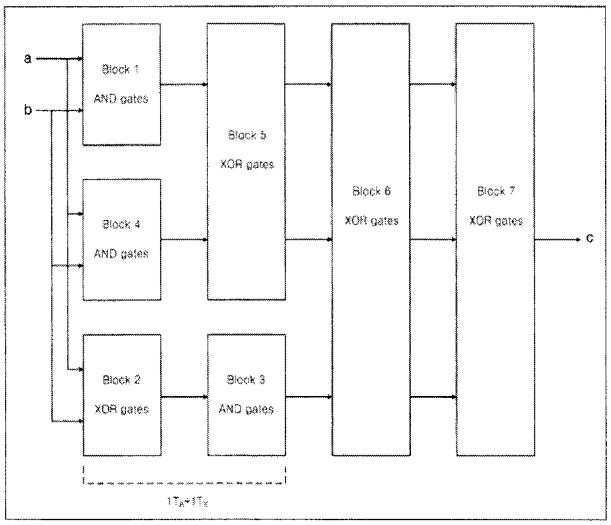


그림 1. 비트-병렬 곱셈기 구조
Fig. 1. Bit-Parallel Multiplier Architecture.

수행할 수 있다. 블록5에서는 블록1,4의 결과 중 재사용될 부분과 아닌 부분으로 분류하여 한번의 XOR연산을 실행한다. 블록1의 $(M1^L_i \wedge b^L)$ $(0 \le i \le k-1)$의 원소들이 재사용되므로 블록1의 결과는 이들과 $(M1^L_{k+i} \wedge b^L)$ $(0 \le i \le k-2)$로 분류하고 블록4에서는 $(M2^H_{k+i} \wedge b^H)'$ $(0 \le i \le n-2k-2)$을 재사용 할 수 있으므로 이들과 나머지로 분류하여 한 번의 XOR 연산을 수행한다.

(5) 블록6의 과정

$1T_A + 1T_X$의 시간 후에 블록3과 블록5의 결과가 생성되면 이들을 통합하여 재사용할 부분과 아닌 부분으로 구분하여 세 개의 XOR 트리를 생성한다. 이들을 각각 $T1'$, $T2'$, $T3'$라 하면 $T1'$과 $T2'$가 재사용이 가능한 원소들의 합을 생성하는 XOR 트리이고 $T3'$은 재사용되지 않는 원소들의 합을 생성하는 XOR 트리이다. 먼저 $T1'$은 블록5의 결과 중 $(M1^L_i \wedge b^L)$ $(0 \le i \le k-1)$와 관련된 원소들의 합을 생성하게 되는데 결과적으로 $(M1^L_i \wedge b^L)_s (0 \le i \le k-1)$를 계산한다. $T2'$는 블록5의 결과 중 $(M2^H_{k+i} \wedge b^H)'$ $(0 \le i \le n-2k-2)$의 원소들과 관련된 부분과 블록3에서 재사용이 가능한 $M3_{n+i} \wedge b^S$ $(0 \le i \le k-2)$의 원소들의 합을 생성하게 된다. 즉, $(M2^H_{k+i} \wedge b^H)'_s + (M3_{n+i} \wedge b^S)_s$를 계산한다. 마지막으로 $T3'$는 원소의 합이 재사용되지 않는 블록 3,5의 결과 원소들의 합을 계산한다. 예를 들어 $n+1/3 \le k < 2/n$인 경우에 $T3$는 다음을 계산하게 된다.

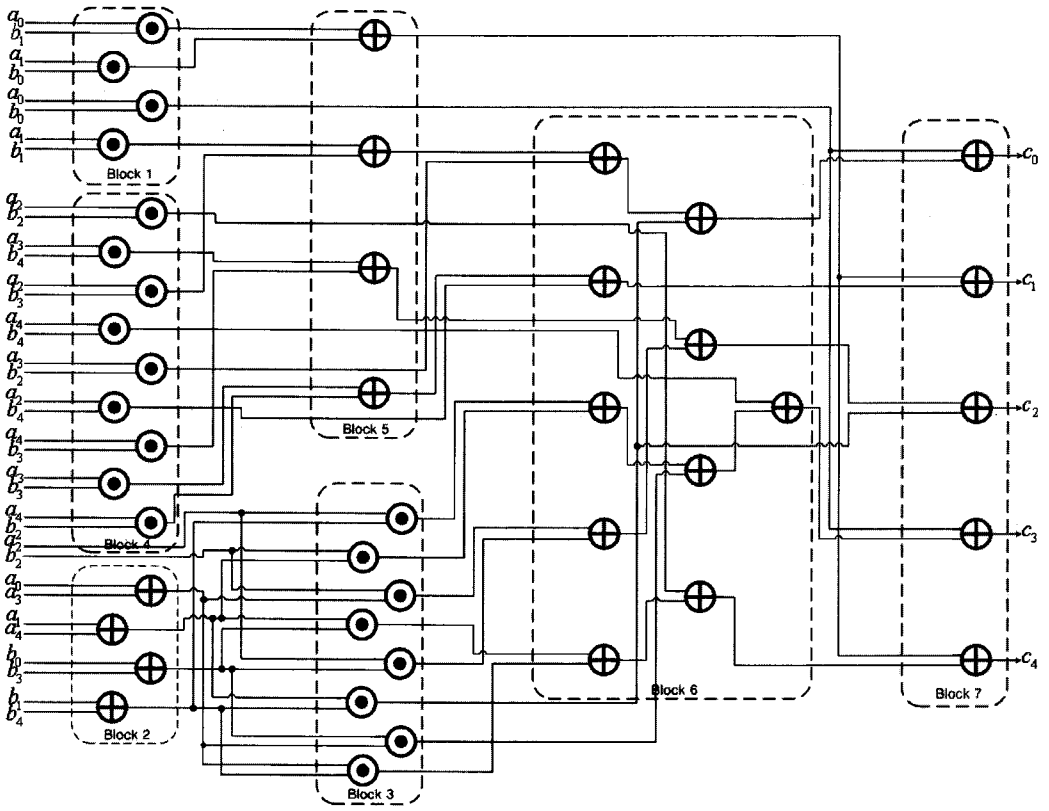


그림 2. $f(x)=x^5+x^2+1$ 일 때 제안하는 곱셈기 구조

Fig. 2. Proposed Bit-Parallel Multiplier Architecture using $f(x)=x^5+x^2+1$.

- $(M1^L_{k+i}\wedge b^L)\_s+((M2^H_{k+i}\wedge b^H)-(M2^H_{k+i}\wedge b^H)')\_s,$ $(0\le i\le n-2k-2),$
- $(M1^L_i\wedge b^L)\_s+(M2^H_{k+i}\wedge b^H)\_s,$ $(n-2k-1\le i\le k-2),$
- $(M2^H_{k+i}\wedge b^H)\_s,$ $(i=k-1),$
- $(M2^H_{k+i}\wedge b^H)\_s+((M3_i\wedge b^S)-(M3_i\wedge b^S)')\_s,$ $(k\le i\le n-k-2),$
- $(M2^H_{k+i}\wedge b^H)\_s+(M3_i\wedge b^S)\_s,$ $(n-k-1\le i\le 2k-1),$
- $((M2^H_{i-(n-k)}+M2^H_{k+i})\wedge b^H)\_s+(M3_i\wedge b^S)\_s,$ $(2k\le i\le n-2),$
- $(M2^H_{k-1}\wedge b^H)\_s+(M3_i\wedge b^S)\_s,$ $(i=n-1).$

(6) 블록7의 과정

마지막 블록7에서 $T1', T2', T3'$의 결과를 더하여 c_i를 구한다.

예제)의 $f(x)=x^5+x^2+1$일때 제안하는 곱셈기의 구조는 그림 2와 같다.

V. 제안하는 곱셈기의 복잡도

본 장에서는 제안하는 곱셈기의 공간복잡도와 시간복잡도에 대하여 살펴본다. 공간 복잡도를 최소화하기 위해 XOR 트리를 세 개로 분할한 것이므로 이것은 $Z1, Z2, Z3$에 중복되어 나타나는 원소들을 한 곳에만 존재하도록 하여 곱셈을 하고 $T1', T2'$의 결과를 재사용하여 $C=(Z1\wedge b^L)_s+(Z2\wedge b^H)_s+(Z3\wedge b^S)_s$를 계산하는데 드는 게이트 비용을 측정하는 것과 같다. 블록1에서는 $Z1\wedge b^L$, 블록4에서는 $Z2\wedge b^H$을 계산하는데 재사용되는 부분을 제외하면 각각 $M1^L\wedge b^L$과 $M2^H\wedge b^H$만을 계산하면 되므로 k비트 곱셈과 $n-k$비트 곱셈이 필요하다. 블록3에서는 $Z3\wedge b^S$을 위해 $M3\wedge b^S$을 계산해야 하므로 $n-k$비트 곱셈이 필요하나 이 들 중에서 $M2^L$과 같아서 불필요한 연산으로 제거된 부분과 $(M2^H_{k+i}\wedge b^H)'$으로 대신할 수 있는 $(M3_{k+i}\wedge b^H)'$ $(0\le i\le n-2k-2)$를 제외하면 $(n-k)^2-(n-2k)^2=2nk-3k^2$의 AND연산만이 필요하다. 그리고 블록2에서 수행하는 사전 계산에는 k비트 덧셈이 두 번 필요하고 $(Z1\wedge b^L)_s$을 계산하기 위해서는 (2)으로부터 $(k-1)^2+(k-1)$의 덧셈 연산이 필요하고 $(Z2\wedge b^H)_s$를 계산하기 위해 (4)로 부터 $(n-k-1)^2+(n-2k)$의 덧셈이 필요하다. $(Z3\wedge b^S)_s$를 계산하기 위해서는 (3)으로부터 $(n-k-1)^2-(n-2k-1)^2+k-1$의 덧

셈 연산이 필요하다. 여기서 $(n-2k-1)^2$는 불필요한 부분과 재사용하는 부분에 필요한 덧셈 연산수이다. 이제 최종결과인 c_i를 구하는데 필요한 덧셈 연산을 살펴보면 $(Z1 \wedge b^L)_s + (Z2 \wedge b^H)_s + (Z3 \wedge b^S)_s$을 계산하여 $c_0 \sim c_{k-2}$를 구하는데 $2(k-1)$, c_{k-1}을 구하는데 1, $c_k \sim c_{n-k-1}$에 $n-2k$, $c_{n-k} \sim c_{n-1}$에 $2k$의 덧셈이 필요하다. 그러므로 제안하는 곱셈기의 공간 복잡도는 다음과 같다.

AND = n^2-k^2,
XOR = $n^2-1-(k-1)^2$,
TOT = $2n^2-2k^2+2k-2$.

시간 복잡도는 먼저 블록1,4,5와 블록2,3이 수행되는데 각각 한 번의 AND와 XOR 연산이 필요하므로 $1T_A+1T_X$의 시간 지연이 생기고 그 이후에 필요한 시간 지연은 모듈로 감산 연산에 이용하는 삼항 기약다항식 $f(x)=x^n+x^k+1$의 중간항의 차수 k에 따라 달라진다. 곱셈기가 블록6에서 세 개의 XOR 트리를 구성하여 덧셈을 하고 있으므로 세 개의 트리가 계산을 끝내는데 소요되는 시간 지연이 제안하는 곱셈기의 시간 복잡도가 된다. 여기서 $1T_A+1T_X$이후에 블록3의 결과와 블록5의 결과를 분류하여 생성한 트리가 결과를 계산하는데 걸리는 시간은 $1T_A$이후에 블록1,4의 원소와 블록3의 원소의 두 배를 분류하여 $T1, T2, T3$트리로 생성하여 계산하는 시간과 같다. 또한 마지막으로 $T1, T2, T3$의 결과를 더할 때 $T3$의 결과가 먼저 생성되었어도 재사용을 위해 생성된 트리 $T1, T2$의 결과에 더해져서 영향을 미쳐서는 안 된다. 따라서 $T1, T2$의 결과는 재사용을 위해 보존되어야 하므로 이들의 결과가 생성되면 $T3$의 결과와 더해야 한다. 그러므로 $1T_A$ 이후에 $T1, T2, T3$의 원소의 수가 각각 t1, t2, t3라 하면 시간 복잡도는

$$1T_A+(\lceil \log_2(t3+2^{\lceil \log_2(t1)\rceil}+2^{\lceil \log_2(t2)\rceil})\rceil)T_X$$

와 같다. 곱셈 행렬의 형태는 사용되는 삼항 기약다항식에 따라 다음과 같이 여섯 가지 경우로 나눌 수 있다.

1. $k=1$,
2. $1<k\le(n-1)/4$,
3. $(n-1)/4<k\le(n-2)/3$,
4. $k=(n-1)/3$,
5. $k=n/3$,
6. $(n+1)/3\le k<n/2$.

첫 번째 $k=1$의 경우를 살펴보자. k가 1일 때 각 c_i를 구하기 위해 세 개의 XOR 트리가 더해야할 원소의 개수는 아래와 같다.

$$c_0 \Rightarrow \begin{cases} T1: 1 \\ T2: n-3 \\ T3: 2 \end{cases}, \quad c_i \Rightarrow \begin{cases} T2: 2n-6-2t \\ T3: 6+t \end{cases},$$

$$c_{n-3} \Rightarrow \begin{cases} T2: 1 \\ T3: n+2 \end{cases}, \quad c_{n-2} \Rightarrow T3: n+2,$$

$$c_{n-1} \Rightarrow \begin{cases} T1: 1 \\ T3: n+2 \end{cases}, (1\le i\le n-4,\ t=i-1).$$

그러면 각 계수를 구하는데 소요되는 시간은

$$c_0 \Rightarrow 1T_A+\lceil \log_2(2+2^{\lceil \log_2(n-3)\rceil}+2^{\lceil \log_2(1)\rceil})\rceil T_X,$$
$$c_i \Rightarrow 1T_A+\lceil \log_2((6+t)+2^{\lceil \log_2(2n-6-2t)\rceil})\rceil T_X,$$
$$c_{n-3} \Rightarrow 1T_A+\lceil \log_2((n+2)+2^{\lceil \log_2(1)\rceil}))\rceil T_X,$$
$$c_{n-2} \Rightarrow 1T_A+\lceil \log_2(n+2)\rceil T_X,$$
$$c_{n-1} \Rightarrow 1T_A+\lceil \log_2((n+2)+2^{\lceil \log_2(1)\rceil}))\rceil T_X,$$
$$(1\le i\le n-4,\ t=i-1)$$

와 같게 된다. 이 중 가장 큰 값이 $k=1$일 때 곱셈기의 시간 복잡도가 된다. 정리 2에 의해서 $n=2^m+u$ $(1\le u\le 2^m)$, $k=1$ 일 때 c_{u-3} 또는 $c_{u-3+2^{m-1}}$에서 $1T_A+\lceil \log_2((6+t)+2^{\lceil \log_2(2n-6-2t)\rceil})\rceil T_X$가 최대값을 가지며 이는 C의 다른 계수를 구하기 위한 시간보다 크므로, 이때 곱셈기의 시간 복잡도는

$$1T_A+\lceil \log_2((u+2)+2^{\lceil \log_2(2^{m+1}-14)\rceil})\rceil T_X$$

또는 $1T_A+\lceil \log_2((u+2+2^{m-1})+2^{\lceil \log_2(2^m-14)\rceil})\rceil T_X$

임을 알 수 있다.

정리2. $n=2^m+u(1\le u\le 2^m)$이고 $k=1$일 때 c_{u-3} 또는 $c_{u-3+2^{m-1}}$을 구할 때 최대 시간 지연이 생긴다.

증명) $u-3>0$인 경우 $2n-6-2t=2^{m+1}+2u-6-2t$이므로 $2u-6-2t>0$이면 $\lceil \log_2(2n-6-2t)\rceil = m+2$이다. $1T_A+\lceil \log_2((6+t)+2^{\lceil \log_2(2n-6-2t)\rceil})\rceil T_X$는 $2u-6-2t>0$를 만족하는 t가 최대일 때 최대값을 가지므로 이때 $t=u-4$이다. 따라서 c_{u-3}를 계산할 때 최대값을 갖는다. $u-3\le 0$인 경우에는 $2u-6-2t$

표 1. 기존의 곱셈기와 제안하는 곱셈기의 시간 및 공간복잡도 비교
Table 1. Comparing complexity of Mastrovito parallel multiplier.

곱셈기	k의 범위	시간복잡도	공간복잡도 (#AND+#XOR)
PB Mastrovito[2]	$1 < k < n/2$	$T_A + (2 + \lceil \log_2(n) \rceil) T_X$	$2n^2 - 1$
SPB Mastrovito[3]	$1 \le k < n/2$	$T_A + (\lceil \log_2(2n-k-1) \rceil) T_X$	$2n^2 - 1$
Proposed	$k=1$	$\max\{T_A + (\lceil \log_2(6+t+2^{\lceil \log_2(2n-6-2t) \rceil}) \rceil) T_X\}$, $(0 \le t \le n-5)$	$2n^2 - 2k^2 + 2k - 2$
	$1 < k \le \frac{n-1}{4}$	$\max\{T_A + (\lceil \log_2(6k+t+4+2^{\lceil \log_2(n-3k-t) \rceil + 1}) \rceil) T_X\}$, $(0 \le t \le n-4k-2)$	
	$\frac{n-1}{4} < k \le \frac{n-2}{3}$	$\max\{T_A + (\lceil \log_2(2k+4t+4+2^{\lceil \log_2(n-3-3t) \rceil + 1}) \rceil) T_X\}$, $(0 \le t \le n-3k-2)$	
	$k = \frac{n-1}{3}$	$\max\{T_A + (\lceil \log_2(n-k+3+3t+2^{\lceil \log_2(n-3-3t) \rceil}) \rceil) T_X\}$, $(0 \le t \le n-2k-2)$	
	$k = \frac{n}{3}$	$\max\{T_A + (\lceil \log_2(n-k+3+3t+2^{\lceil \log_2(n-3-3t) \rceil}) \rceil) T_X\}$, $(0 \le t \le k-2)$	
	$\frac{n+1}{3} \le k < \frac{n}{2}$	$\max\{T_A + (\lceil \log_2(n-k+3+3t+2^{\lceil \log_2(n-3-3t) \rceil}) \rceil) T_X\}$, $(0 \le t \le n-2k-2)$ 또는 $\max\{T_A + (\lceil \log_2(4n-7k+t+1+2^{\lceil \log_2(-2n+6k-2t-1) \rceil} + 2^{\lceil \log_2(1+t) \rceil}) \rceil) T_X\}$, $(0 \le t \le -n+3k-1)$	

표 2. n이 100에서 126사이 일 때 기존의 곱셈기(SPB Mastrovito multiplier)와 제안하는 곱셈기의 복잡도 비교
Table 2. Comparing complexity of SPB Mastrovito parallel multiplier.

n	k	XOR 시간복잡도			공간복잡도(#AND+#XOR)		
		SPB(T_X)	제안(T_X)	비율(%)①	SPB(개)	제안(개)	비율(%)②
100	49	8	8	100	19999	15294	76.47
102	37	8	8	100	20807	18142	87.19
105	49	8	8	100	22049	17344	78.66
106	15	8	8	100	22471	22050	98.12
108	45	8	8	100	23327	19366	83.01
110	33	8	9	112.5	24199	22086	91.26
111	49	8	8	100	24641	19936	80.90
113	30	8	9	112.5	25537	23796	93.18
118	45	8	8	100	27847	23886	85.77
119	38	8	9	112.5	28321	25508	90.06
121	30	8	9	112.5	29281	27540	94.05
123	2	8	9	112.5	30257	30252	99.98
124	55	8	9	112.5	30751	24810	80.68
126	49	8	9	112.5	31751	27046	85.18

※ 비율(%)① - 제안 XOR 시간 복잡도/기존 XOR 시간 복잡도×100,
※ 비율(%)② - 제안 공간 복잡도/기존 공간 복잡도×100

는 짝수 이므로 t가 증가하여 $2u-6-2t$가 -2^m+2까지 감소하여도 $\lceil \log_2(2n-6-2t) \rceil = m+1$로 일정하므로 $t = u-4+2^{m-1}$일 때, 즉 $c_{u-3+2^{m-1}}$를 구할 때

$$1T_A + \lceil \log_2((6+t)+2^{\lceil \log_2(2n-6-2t) \rceil}) \rceil T_X$$

가 최대값을 갖는다.□

나머지 경우도 정리2와 유사하게 구할 수 있으므로 생략하며 결과를 정리하면 표 1과 같다.

VI. 비교 및 결론

본 장에서는 기존의 곱셈 방법과 제안하는 곱셈 방법의 복잡도를 비교하여 제안하는 곱셈 방법의 효율성에 대하여 논한다. 제안하는 곱셈 방법은 공간 복잡도가 $2n^2-1$일 때 가장 낮은 시간 복잡도를 갖는 기존의 SPB Mastrovito 곱셈 방법과 시간 복잡도와 같거나 $1T_X$ 크게 되나 공간 복잡도는 곱셈에 필요한 AND 게이트와 XOR 게이트 수의 합인 기존의 공간복잡도 $2n^2-1$에 비해 $2k^2-2k+1$만큼 감소하며 또한 k가 $n/2$에 가까울수록 공간 복잡도 감소의 폭이 더욱 커짐을 알 수 있다. 모듈로 감산 연산에 사용되는 삼항 기약 다항식의 최고차 항의 차수 n의 범위가 $100 \le n < 1000$이고 중간항의 차수 k가 $k < n/2$인 1323개의 삼항 기약 다항식 중 40% 정도인 533개에서 시간 복잡도가 기존의 결과와 같았고 나머지 790개에서는 시간 복잡도가 10%~12.5% 증가하였다. 그러나 시간 복잡도의 증가 비율은 n이 증가할 수 록 감소한다. 또한 공간 복잡도는 위의 조건을 만족하는 모든 기약다항식에서 낮아져 최대 25%까지 감소하였다. 따라서 제안하는 곱셈기는 낮은 공간 복잡도를 요구하는 스마트카드와 모바일 폰 등의 환경에 효과적으로 적용될 수 있다.

참 고 문 헌

[1] B. Sunar and C. K. Koc, "Mastrovito multiplier for all trinomials," *IEEE Trans. Comput.*, vol. 48, no. 5, pp. 522-527, May 1999.

[2] A. Halbutogullari and C.K. Koc, "Mastrovito Multiplier for General Irreducible Polynomials," *IEEE Trans. Computers*, vol. 49, no. 5, pp. 503-518, May 2000.

[3] T. Zhang and K. K. Parhi, "Systematic design of original and modified Mastrovito multipliers for general irreducible polynomials," *IEEE Trans. Comput.*, vol. 50, no. 7, pp. 734-749, Jul. 2001.

[4] A. Reyhani-Masoleh and M. A. Hasan, "Low complexity bit parallel architectures for polynomial basis multiplication over $GF(2^n)$," *IEEE Trans. Comput.*, vol. 53, no. 8, pp. 945-959, Aug. 2004.

[5] R. Lidl and H. Niederreiter, "Introduction to finite fields and its applications," *Cambridge Univ. Press*, 1994.

[6] H. Fan and Y. Dai, "Fast bit parallel $GF(2^n)$ multiplier for all trinomials," IEEE Trans. Comput., vol. 54, no. 4, pp. 485-490, Apr. 2005.

[7] H. Fan and M. A. Hasan, "Fast Bit Parallel Shifted Polynomial Basis Multipliers in $GF(2^n)$," *IEEE Trans. Circuits & Systems-I*, vol. 53, no. 12, pp. 2606-2615, Dec. 2006.

[8] M. Leone, "A New Low Complexity Parallel Multiplier for a Class of Finite Fields,"in *Workshop on Cryptographic Hardware and Embedded Systems (CHES'01)*, LNCS 2162, pp. 160-170, 2001.

[9] S. O. Lee, S. W. Jung, C. H. Kim, J. Yoon, J. Koh, and D. Kim, "Design of bit parallel multiplier with lower time complexity," in Proc. ICICS, 2004, pp. 127-139.

[10] E. D. Mastrovito, "VLSI architectures for Computation in Galois Fields", PhD thesis, Linkoping University, Department of Electrical Engineering, Linkoping, Sweden, 1991./parallel finite field multipliers", *Journal of VLSI Signal Processing*, Vol.19, No.2, pp.149-166, July 1998.

[11] C. Shu, S. Kwon, and K. Gaj, "FPGA accelerated Tate pairing based cryptosystemover binary fields", *Cryptography ePrint Archive, Report 2006/179*, 2006.

저 자 소 개

조 영 인(학생회원)

2006년 2월 한양대학교 수학과 학사.

2007년 3월~현재 고려대학교 정보경영공학전문대학원 석사과정.

<주관심분야 : 공개키 암호, 암호칩 설계 기술>

김 창 한(정회원)

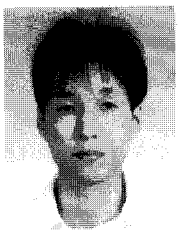
1985년 2월 고려대학교 수학과 학사

1987년 2월 고려대학교 수학과 석사

1992년 2월 고려대학교 수학과 박사

2002년 2월~현재 세명대학교 정보통신학부 정교수

<주관심분야 : 정수론, 공개키암호, 암호프로토콜>



장 남 수(학생회원)

2002년 2월 서울시립대학교 수학과 학사.

2004년 8월 고려대학교 정보보호대학원 석사.

2005년~현재 고려대학교 정보경영공학전문대학원 박사과정.

<주관심분야 : 공개키 암호, 암호칩 설계 기술, 부채널 공격 방법론>

홍 석 희(정회원)

1995년 고려대학교 수학과 학사

1997년 고려대학교 수학과 석사

2001년 고려대학교 수학과 박사

1999년 8월~2004년 2월 (주)시큐리티 테크놀로지스 선임연구원

2003년 3월~2004년 2월 고려대학교 시간강사

2004년 4월~2005년 2월 K.U. Leuven 박사후연구원

2005년~현재 고려대학교 정보경영전문대학원 부교수

<주관심분야> 대칭키 암호 알고리즘, 공개키 암호 알고리즘, 포렌식