유한 필드 $GF(2^m)$상의 시스톨릭 곱셈기/제곱기 설계

(Design of Systolic Multiplier/Squarer over Finite Field $GF(2^m)$)

유 기 영 † 김 정 준 ††
(Kee-Young Yoo) (Jung-Joon Kim)

요 약 본 논문에서는 유한 필드 $GF(2^m)$상에서 모듈러 곱셈 $A(x)B(x)$ mod $P(x)$을 수행하는 새로운 선형 문제-크기(full-size) 시스톨릭 어레이 구조인 LSB-first 곱셈기를 제안한다. 피연산자 $B(x)$의 LSB(least significant bit)를 먼저 사용하는 LSB-first 모듈러 곱셈 알고리즘으로부터 새로운 비트별 순환 방정식을 구한다. 데이타의 흐름이 규칙적인 순환 방정식을 공간-시간 변환으로 새로운 시스톨릭 곱셈기를 설계하고 분석한다. 기존의 곱셈기와 비교할 때 제안한 곱셈기의 면적-시간 성능이 각각 10%와 18% 향상됨을 보여준다. 또한 같은 설계방법으로 곱셈과 제곱연산을 동시에 수행하는 새로운 시스톨릭 곱셈/제곱기를 제안한다. 유한 필드상의 지수연산을 위해서 제안한 시스톨릭 곱셈/제곱기를 사용할 때 곱셈기만을 사용할 때보다 면적-시간 성능이 약 26% 향상됨을 보여준다.

Abstract In this paper, a new LSB-first multiplier with a linear full-size systolic array structure for computing the modular multiplication $A(x)B(x)$ mod $P(x)$ over finite field $GF(2^m)$ is presented. A bit-wise recurrence equation which is driven from LSB-first modular multiplication algorithm using first the LSB of multiplicand $B(x)$ is obtained.

By space-time transformation, a new systolic multiplier is designed from the recurrence equation with regular data flows, and then is analyzed. The area-time performance of the proposed systolic multiplier is improved by 10% and 18%, respectively compared to two well-known multipliers. Moreover, a new systolic multiplier/squarer for computing multiplication and square simultaneously is proposed by using the same design method. For the modular exponentiation over the finite field, the use of the proposed systolic multiplier/squarer shows 26% better area-time performance than the use of only multiplier.

1. 서 론

지난 30여년간 오류-제어 코딩, 디지털 신호 처리 및 암호학 등 여러 분야에서 유한 필드에 대한 연구가 이루어져 왔다[1]. 특히, 모듈러 지수연산에 근거한 Diffie-Hellman 키 교환이나 타원 곡선(elliptic curve) 암호시스템과 같은 암호학적 응용에 유한 필드상에서의 모듈러 곱셈 연산이 중요하게 인식되었다[2] [3].

유한 필드상에서 덧셈은 비트별 배타적 논리합(exclusive or) 연산으로 간단하다. 그러나, 곱셈, 역원산(inversion) 및 지수 연산은 복잡하다. 이들 연산에서 가장 기본이 되는 연산은 모듈러 곱셈 연산이다. 필드상에서 곱셈 연산 및 지수 연산에 대한 효율적인 하드웨어 및 소프트웨어 구현에 대해 많은 관심의 대상이 되었다. 특히 구조가 간단하고, 계산 시간이 짧으며, 좋은 성능을 가지는 모듈러 곱셈에 대한 회로 설계는 아주 중요한 과제로 연구되어 왔다[4-10].

Yeh[11] 등은 다항식 기저 표현(polynomial basis representation)으로 $GF(2^m)$ 곱셈기를 시스톨릭 어레이 구조로 설계하였는데, VLSI 구현에 편리한 구조이지만 두 개의 제어신호를 가진 구조이다. Wang[12]등은 유한 필드 상에서의 MSB-first 방법으로 시스톨릭 곱셈기를 설계하였다. 이 곱셈기는 VLSI로 구현하기 좋은

† 종신회원 : 경북대학교 컴퓨터공학과 교수
yook@bh.knu.ac.kr
†† 비 회 원 : 한국통신 가입자망연구소 무선ATM연구실장
jungkim@kt.co.kr
논문접수 : 2000년 2월 10일
심사완료 : 2001년 3월 30일

구조이다. 그 외에도 Hsu[13]등은 다항식 기저 표현뿐만 아니라, dual, 및 정규 기저 표현에 근거한 표현으로 $GF(2^n)$ 곱셈기를 설계하였다. 최근에는 Jain[14,15] 등과 Song[16]등은 LSB-first 및 MSB-first 방법으로 브로드캐스트(broadcast) 신호를 가지는 준 시스톨릭(semisystolic) 어레이 구조의 곱셈기를 설계하였다. 이 구조는 시간적으로 효율적이지만, 브로드캐스트 데이타 입력을 가지는 준 시스톨릭 곱셈기로 시스톨릭 어레이와는 다르다.

본 논문에서는 유한 필드 $GF(2^m)$상에서 모듈러 곱셈 $A(x)B(x)$ mod $P(x)$을 위한 새로운 선형 문제-크기 시스톨릭 어레이 구조의 LSB-first 곱셈기를 설계하고 분석한다. 모듈러 곱셈의 피연산자 $B(x)$의 LSB(least significant bit)로부터 먼저 사용한 LSB-first 모듈러 곱셈 알고리즘을 유도하고, 이를 비트별 순환 방정식(recurrence equation)으로 변환한다. 데이타의 흐름이 규칙적인 특성을 가지고 있는 순환 방정식으로부터 데이타 의존 그래프(data-dependency graph , DG)를 구한다. DG로부터 시스톨릭 곱셈기를 설계하기 위해 필요한 공간 변환벡터와 시간 변환벡터를 구하여 공간-시간 변환(space-time transformation)에 의해 새로운 LSB-first 시스톨릭 곱셈기를 유도한다. 이 논문에서 제안한 시스톨릭 곱셈기와 많은 논문에서 인용되고 있는 기본적인 기존의 시스톨릭 곱셈기[11][12]를 비교 분석한다. 제안한 시스톨릭 곱셈기는 기존의 시스톨릭 곱셈기보다 구조가 간단하며, 시간적으로 효율이 좋은 곱셈기임을 보여준다.

또한, LSB-first 방식의 곱셈 알고리즘의 특성과 곱셈기 설계에 적용한 같은 설계 방법을 이용하여 곱셈과 제곱연산을 동시에 수행하며 구조가 간단하고 효율적인 새로운 시스톨릭 곱셈/제곱기를 제안한다. 제안된 시스톨릭 곱셈/제곱기는 곱셈기보다 회로는 조금 더 복잡하지만 수행시간은 같아서 모듈러 지수연산에 아주 효율적으로 이용될 수 있음을 보여준다.

본 논문의 구성은 다음과 같다. 제2장에서는 유한 필드에 대해서 간단히 알아본다. 제3장에서 LSB-first 모듈러 곱셈 알고리즘을 설명하고, 이 알고리즘에서 정규 순환 방정식으로 변환하며, LSB-first시스톨릭 곱셈기를 설계하고, 비교 분석한다. 제4장에서는 곱셈과 제곱연산을 동시에 수행하는 순환 방정식을 유도한다. 같은 설계 방법으로 곱셈과 제곱연산을 동시에 수행하는 시스톨릭 곱셈/제곱기를 설계하고 분석한다. 마지막으로 제 5장에서 결론을 맺는다.

2. 유한 필드

유한 필드(finite field) 혹은 Galois 필드(Galois field)는 교환, 결합 및 분배 법칙에 대해 닫혀 있고, 덧셈, 뺄셈, 곱셈, 나눗셈 연산이 가능한 유한 개 원소들의 집합이다. 비록 유한 필드가 많은 소수(prime number)의 지수 차수에 대해서 존재하지만, 암호학에서 주로 사용되는 필드로는 소수 q에 대한 소수 유한 필드 $GF(q)$와 양수 m에 대한 이진 유한 필드 $GF(2^m)$이다. 본 논문에서는 이진 유한 필드상의 곱셈에 대해서만 고려한다. 유한 필드 $GF(2^m)$는 길이 m인 2^m개의 조합이 가능한 비트 문자열(string)으로 구성된다.

$$GF(2^m)=\{(a_{m-1}\ a_{m-2}\ \ldots\ a_1\ a_0)|a_i\in GF(2),\ 0\le i\le m-1\}$$

$GF(2^m)$의 각 비트 문자열은 $GF(2)$의 m개 원소들로 구성된다.

$GF(2^m)$에서 곱셈을 수행하기 위해서는 먼저 필드상에서 각 비트 문자열을 해석하기 위한 기저 표현(basis representation)이 선행되어져야 한다. 기저 표기법에는 크게 다항식 기저 표현법과 정규 기저 표현이 있다. 본 논문에서는 다항식 기저 표현으로 필드상의 원소들을 표현한다. 다항식 기저 표현에서는 $GF(2^m)$의 각 원소는 m차수 미만의 다항식으로 표현된다.

$$GF(2^m)=\{a_{m-1}x^{m-1}+\ldots+a_1x+a_0\,|\,a_i\in GF(2),\ 0\le i\le m-1\}$$

$GF(2)$상에서 차수 m의 기약 다항식(irreducible polynomial) $P(x)$는 필드 $GF(2^m)$을 구성하는데 필요하다. 유한 필드 $GF(2^m)$의 2^m개 다항식들은 $GF(2)$의 원소를 계수로 가지는 모든 다항식들을 기약 다항식 $P(x)$로 나머지 연산을 행한 결과이다. 유한 필드$GF(2^m)$에서 비트 문자열 $A=(a_{m-1}\ a_{m-2}\ \ldots\ a_1\ a_0)$는 다항식 $A(x)=\ a_{m-1}x^{m-1}+\ldots+a_1x+a_0$에 일대일(one-to-one)로 대응되며, 앞으로 필드 원소 A의 비트 문자열 표현과 $A(x)$의 다항식 표현은 동치로 간주한다.

3. 곱셈 알고리즘 및 시스톨릭 곱셈기

이 장에서 유한 필드 $GF(2^m)$ 상에서 LSB-first모듈러 곱셈 알고리즘을 분석하고, 정규 순환 방정식을 유도한다. 비트별 정규 순환 방정식으로부터 공간-시간 변환으로 LSB-first 문제-크기 시스톨릭 곱셈기를 설계한다.

앞으로 유한 필드 $GF(2^m)$의 원소들은 대문자로 표기하며, $GF(2)$의 원소들은 첨자(subscript)를 가진 대문자와 소문자로 표기한다. 그리고 덧셈과 곱셈은 각각 비트

별 배타적 이항 논리합(XOR)과 논리곱(AND)을 의미한다.

3.1 곱셈 알고리즘

곱셈은 차수 m의 원시 다항식 $P(x)$에 의해서 정의된다. 두 원소의 곱셈은 단순히 두 다항식을 곱한 뒤에 $P(x)$로 모듈러 연산을 취해주면 된다. $A(x)$와 $B(x)$는 $GF(2^m)$의 원소이고, $P(x)$는 차수 m의 원시 다항식이라 하면, 다항식 $A(x)$, $B(x)$ 및$P(x)$는 다음과 같이 표현된다.

$$A(x)=a_{m-1}x^{m-1}+a_{m-2}x^{m-2}+...+a_1x+a_0$$
$$B(x)=b_{m-1}x^{m-1}+b_{m-2}x^{m-2}+...+b_1x+b_0$$
$$P(x)=x^m+p_{m-1}x^{m-1}+p_{m-2}x^{m-2}+...+p_1x+p_0$$

다항식 $M(x)$를 $A(x)$와 $B(x)$의 곱셈 결과를 $P(x)$로 모듈러 연산을 수행한 결과라면 다음과 같다.

$$\begin{aligned}M(x)&=A(x)\ B(x)\ \bmod P(x)\\&=b_0\ A(x)+b_1[A(x)\ x\ \bmod P(x)]\\&\quad+b_2[A(x)\ x^2\ \bmod P(x)]+...\\&\quad+b_{m-1}[A(x)\ x^{m-1}\ \bmod P(x)]\end{aligned}\quad(1)$$

식 (1)을 보면 B의 LSB(Least Significant Bit)부터 먼저 사용됨을 알 수 있다. 이를 LSB-first 곱셈이라 한다. 식 (1)을 다음과 같은 순환식으로 변환할 수 있다. ($1\le i\le m$)

$$\begin{aligned}A^{(i)}(x)&=A^{(i-1)}\ (x)\ x\ \bmod P(x)\\M^{(i)}(x)&=M^{(i-1)}(x)+b_{i-1}A^{(i-1)}(x)\end{aligned}\quad(2)$$

여기서 $A^{(0)}(x)=A(x)$이고, $M^{(0)}(x)=0$이다. 식 (2)에서 $A^{(i)}(x)$은 식 (1)에서 i 번째 [ ] 안에 있는 식을 의미하며, $M^{(i)}(x)$은 식 (1)에서 i 번째 덧셈 연산자+전까지의 부분합(partial sum)을 나타낸다. 위 첨자 i의 값이 m일 때, $M^{(m)}(x)=M(x)=A(x)B(x)\ \bmod P(x)$이다. 식 (2)에 있는 두 순환식은 병렬로 계산될 수 있음을 쉽게 알 수 있다.

식 (2)의 계산은 다음의 식 (3)과 같이 비트 문자열의 형태로 표현된다.

$$\begin{aligned}A^{(i)}&=sh(A^{(i-1)})\ \bmod P\\M^{(i)}&=M^{(i-1)}+b_{i-1}A^{(i-1)}\end{aligned}\quad(3)$$

여기서, 비트 문자열 $A^{(0)}=(a_{m-1}\ a_{m-2}\ ...\ a_1\ a_0)$이고, $M^{(0)}=(0\ 0\ ...\ 0\ 0\ )$이다.

식 (3)에서 계산 $sh(A^{(i-1)})$은 비트 문자열 $A^{(i-1)}$을 1 비트 왼쪽 쉬프트 연산을 의미하며, 다음 식 (4)와 같다.

$$sh(A^{(i-1)})=(A^{(i-1)}_{(m-1)},\ A^{(i-1)}_{(m-2)},...,A^{(i-1)}_1,\ A^{(i-1)}_0,\ 0)\quad(4)$$

비트별 순환식을 쉽게 유도하기 위해서 식 (4)의 비트 문자열을 다항식으로 표현하면 차수가 m인 아래의 다항식 $A^{(i-1)}(x)x$ 와 같은데, 다항식의 차수가 m이므로 기약 다항식 $P(x)$로 모듈러 연산을 취해 주어야 한다.

$$sh(A^{(i-1)})=(A^{(i-1)}_{(m-1)},\ A^{(i-1)}_{(m-2)},...,A^{(i-1)}_1,\ A^{(i-1)}_0,\ 0)\quad(5)$$

한편, 기약 원시 다항식 $P(x)$으로부터 $x^m=\ p_{m-1}x^{m-1}+p_{m-2}x^{m-2}+...+p_1x+p_0$임을 이용하여 이를 식 (5)의 최고 차수의 항에 대입하면 아래와 같은 식 (6)을 구할 수 있다.

$$\begin{aligned}A^{(i)}(x)&=A^{(i-1)}_{(m-1)}x^m+A^{(i-1)}_{(m-2)}x^{m-1}+...\\&\quad+A^{(i-1)}_1x^2+A^{(i-1)}_0x\ \bmod P(x)\\&=A^{(i-1)}_{(m-1)}(p_{m-1}x^{m-1}+p_{m-2}x^{m-2}+...+p_1x+p_0)\\&\quad+A^{(i-1)}_{(m-2)}x^{m-1}+...+A^{(i-1)}_1x^2+A^{(i-1)}_0x\\&=(A^{(i-1)}_{(m-2)}+A^{(i-1)}_{(m-1)}p_{m-1})x^{m-1}\\&\quad+(A^{(i-1)}_{(m-3)}+A^{(i-1)}_{(m-1)}p_{m-2})x^{m-2}+...\\&\quad+(A^{(i-1)}_0+A^{(i-1)}_{(m-1)}p_1)x+A^{(i-1)}_{(m-1)}p_0\end{aligned}\quad(6)$$

식 (6)을 다시 비트 문자열로 표현하면 다음의 식 (7)과 같다.

$$\begin{aligned}A^{(i)}=(&A^{(i-1)}_{(m-2)}+A^{(i-1)}_{(m-1)}p_{m-1},\ A^{(i-1)}_{(m-3)}+A^{(i-1)}_{(m-1)}p_{m-2},...,\\&A^{(i-1)}_0+A^{(i-1)}_{(m-1)}p_1,A^{(i-1)}_{(m-1)}p_0)\end{aligned}\quad(7)$$

식 (3)의 첫번째 순환식으로부터 식 (7)을 유도한다. 식 (7)과 식 (3)의 두 번째 순환식을 비트별 연산으로 고쳐서 다음의 비트별 LSB-first 모듈러 곱셈 알고리즘을 유도한다.

이 곱셈 알고리즘의 수행을 2차원 평면에 그래프로 표현할 수 있다. 각 인덱스 점(index point) (i, j), ($1\le i\le m$, $1\le j\le m$)은 계산이 수행되는 곳으로 계산 점(computation point)이라고도 하는데, 그래프에서 한 개의 노드(node)로 표현되며, 각 계산 점에서 계산에 필요한 데이타의 흐름은 그래프에서 에지(edge)로 표현된다. 에지는 각 계산에 필요한 데이타의 의존관계를 의미하므로 이 그래프를 데이타 의존 그래프(data dependency graph, DG)라 한다. 또한 이 그래프를 Signal Flow Graph라 부르기도 한다.

[비트별 LSB-first 모듈러 곱셈 알고리즘]

입력 : $A=(a_{m-1},\ a_{m-2},\ ...\ ,\ a_1,\ a_0)$

$B=(b_{m-1},\ b_{m-2},\ ...\ ,\ b_1,\ b_0)$

$P=(1,\ p_{m-1},\ p_{m-2}\ ,\ ...\ ,\ p_1,\ p_0)$

출력 : $M^{(m)}=M(x)=A(x)B(x)\ \bmod P(x)$

초기치 : $M^{(0)}=(M_{m-1}^{(0)}, M_{m-2}^{(0)}, \ldots, M_1^{(0)}, M_0^{(0)})$
$=(0, 0, \ldots, 0, 0)$
$A^{(0)}=(A_{m-1}^{(0)}, A_{m-2}^{(0)}, \ldots, A_1^{(0)}, A_0^{(0)}, A_{-1}^{(0)})$
$=(a_{m-1}, a_{m-2}, \ldots, a_1, a_0, 0)$
$A_{-1}^{(i)}\equiv 0,\ 1\le i\le m$

순환식 : **for** i=1 **to** m
for j=1 **to** m
$A_{m-j}^{(i)}=A_{m-1-j}^{(i-1)}+A_{m-1}^{(i-1)}\ p_{m-j}$
$M_{m-j}^{(i)}=M_{m-j}^{(i-1)}+b_{i-1}A_{m-j}^{(i-1)}$

위의 비트별 LSB-first 모듈러 곱셈 알고리즘에 대한 DG는 다음 그림 1과 같다.

그림 1의 DG에서 데이타의 의존관계가 규칙적이고, 또한 서로 반대 방향으로 진행하는 에지들이 없어서 시스톨릭 어레이 구조의 하드웨어로 설계하기가 적합하다. 이와 같이 정규적인 데이타의 의존 관계를 가지는 이 알고리즘을 정규 순환 방정식(regular recurrence equation)이라고도 한다.

그림 1에서 보는 바와 같이 계산 점 (i, j), $(1\le i\le m, 1\le j\le m)$에 데이타 $A_{m-j}^{(i-1)}$, $M_{m-j}^{(i-1)}$와 P_{m-j}는 계산 점 $(i, j-1)$로 부터 전달되어 오는데, i=1일 때 이들 데이타가 외부로부터 입력된다. 또한 데이타 b_{i-1}는 왼쪽에서부터 계산 점 (i, j)에 전달되는데, j=1일 때는 외부에서 입력된다. 한편 데이타 $A_{m-1-j}^{(i-1)}$는 대각으로 계산 점 (i, j)에 전달된다. 특히, i=1이거나 $j=m$일 때는 초기값이 입력된다. 첫번째 열에 있는 계산 점$(i, 1)$에서 계산된 데이타 $A_{m-1}^{(i)}$은 대각으로 계산 점$(i+1, 1)$에 입력되어 그 행에서 값이 변하지 않고 오른쪽으로 전달된다. 각 계산 점에 전달되는 데이타 P_{m-j}와 b_{i-1}는 계산에 사용만 되며 변하지 않고 이웃 계산 점으로 전달된다. 각 계산 점 (i, j)으로 전달된 값을 사용하여 알고리즘에 따라 $A_{m-j}^{(i)}$와 $M_{m-j}^{(i)}$의 새로운 값이 계산되어 하단으로, 대각으로 다음 계산 점으로 각각 전달된다. 마지막 행으로부터 출력되는 $M_j^{(m)}$는 곱셈의 결과 값의 j번째 비트이다.

3.2 시스톨릭 곱셈기의 설계

이 장에서는 앞장에서 언급한 곱셈 알고리즘으로부터 시스톨릭 곱셈기를 설계한다. 그림 1의 DG로부터 해석적이고 체계적인 절차로 시스톨릭 어레이를 설계할 수 있다[17][18]. DG로 표현된 알고리즘으로부터 시스톨릭 어레이를 설계하기 위해서 만족되어져야 하는 공간 변환벡터와 시간 변환벡터를 구한다. 공간 및 시간 변환벡

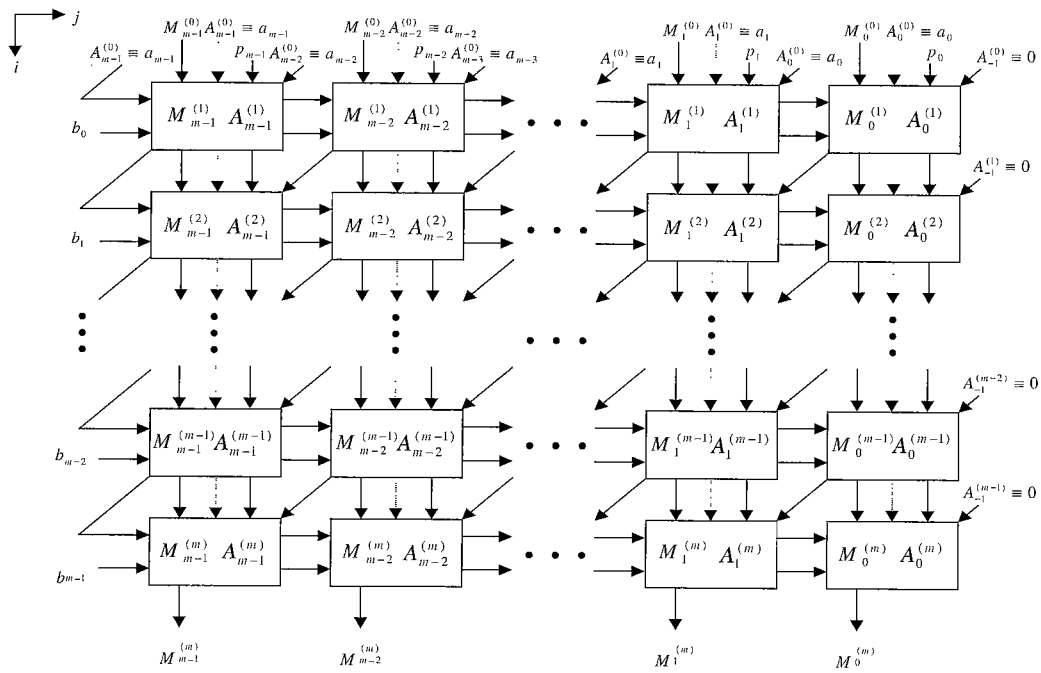

그림 1 자료 의존 그래프

터로 구성된 공간-시간 변환으로 알고리즘에서 시스톨릭 어레이 구조의 곱셈기를 유도한다.

시스톨릭 어레이의 설계에 대한 설명을 간단히 하기 위해서 예를 들어 유한 필드 $GF(2^4)$상에서의 시스톨릭 곱셈기의 설계를 설명한다. 일반적인 $GF(2^m)$상의 시스톨릭 어레이의 설계는 같은 방법으로 확장하여 적용하면 된다. 그림 1에서 제일 왼쪽에 있는 계산 점에서 대각으로 전달되어 오는 데이타의 흐름을 규칙적으로 하기 위해서 한 열의 계산 점을 추가하여 그림 2와 같이 개선된 DG를 만들 수 있다. 그림 2에서 실선 에지는 데이타가 변하지 않고 전달됨을, 점선 에지의 데이타 흐름은 각 PE에서 계산되어 새로운 값이 전달됨을 의미한다.

그림 2에서 보는 것과 같이 데이타 의존관계로 점선상에 있는 계산 점들은 병렬로 수행될 수 있다. 이런 점선을 동시간 초평면(equitemporal hyperplane)이라 한다. 즉, 초평면상에 있는 계산 점은 같은 시간스텝(time step)에 수행될 수 있음을 의미한다. 병렬로 계산의 수행이 완전히 끝날 때까지는 12 시간스텝이 걸림을 알 수 있다. 일반적으로 $GF(2^m)$상에서 곱셈 알고리즘의 계산 점들이 최대로 병렬성을 살려서 수행될 때 걸리는 시간스텝은 $3m$이다.

그림 2에서 초평면상에 있는 계산 점들이 설계될 시스톨릭 어레이에서 동시에 수행될 수 있게 시간 변환벡터를 정해야 한다. 시간벡터는 초평면과 수직인 벡터를 구하면 초평면상의 계산 점의 시간스텝이 같게 된다. 따라서, 동시간 초평면의 벡터가 [1, -2]이므로 시간 변환벡터는 $t=[2, 1]$이다.

한편, 그림 2의 각 계산 점이 선형(linear) 시스톨릭 어레이에 대응하도록 공간 변환벡터를 구하여야 한다. 그림 2의 DG를 i축, j축 및 대각선을 투영벡터(projection vector)로 하여 각 투영벡터상의 계산 점들이 시스톨릭 어레이의 같은 처리기(processing element, PE)에서 수행되도록 공간 변환벡터를 구할 수 있다. DG를 i축으로 투영하면 m개의 PE로 매핑(mapping)되지만 결과값이 각 PE에 머무르게 되어 설계에 어려움이 있으며, j축으로 투영하면 DG는 m개의 PE로 매핑되며 결과값이 흐르게 되어 i축으로 투영할 때 보다 좋은 구조의 시스톨릭 어레이가 유도된다. 한편, 대각으로 투영하면 $2m-1$개의 PE로 매핑되어 전자의 경우보다 어레이 구조가 나쁘다. 그 중에서 j축으로 투영된 어레이 구조가 더 좋음 알 수 있다. j축을 투영벡터로 할 때 투영벡터가 [0, 1]이므로 투영벡터에 수직인 공간 변환벡터는 $s=[1, 0]$이다.

따라서, 공간 변환벡터와 시간 변환벡터로 구성된 공간-시간 변환행렬 T는 다음과 같다.

$$T=\begin{pmatrix} t \\ s \end{pmatrix}=\begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix} \tag{8}$$

변환 T는 2차원의 DG를 선형의 시스톨릭 어레이로 매핑시킨다. 따라서, 시간 변환벡터는 각 계산 점을 유도될 시스톨릭 어레이 상에서 그 계산점이 수행되어야

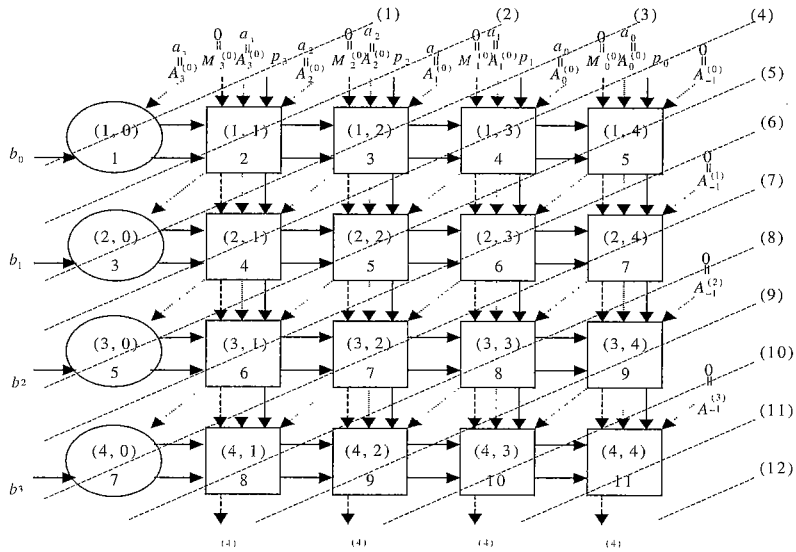

그림 2 데이타 의존 그래프의 예($m=4$)

할 시간스텝에 대응시키며, 각 데이타가 전달되는 에지를 시스톨릭 어레이에서 PE사이에 데이타가 전달되는데 걸리는 시간에 대응시킨다. 또 공간 변환벡터는 각 계산점을 유도될 시스톨릭 어레이 상에서 그 계산 점의 계산이 수행될 PE에 대응시키며, 각 에지를 시스톨릭 어레이의 연결 링크에 대응시킨다. 변환행렬에 의해서 시스톨릭 어레이로 대응될 때 각 계산 점은 같은 PE에 같은 시간스텝에 수행될 수 없는데, 즉, 서로 다른 두 계산 점 p와 q에 대해서 $Tp \neq Tq$이어야 하는데 변환행렬 T는 이 조건을 만족함을 쉽게 알 수 있다.

변환행렬 T에 의해서 시스톨릭 어레이로 변환될 때, 그림 2에서 같은 행에 있는 계산 점들은 같은 PE에 대응되는데, 사각형의 계산 점과 원으로 표시된 계산 점을 수행해야 하기 때문에 유도된 시스톨릭 어레이의 각 PE는 두 가지의 다른 계산을 한다. 이와 같이 각 PE는 두 가지 다른 계산을 구별하기 위해서 제어신호(그림 3의 ctl)가 필요하다. 각 PE가 처음으로 계산할 때, 즉 ctl=1일 때, 외부에서 입력되거나($A_3^{(0)}$) 이웃 PE에서 계산이 된 값이 입력되는 값을 저장하고, 처음이 아닐 때, 즉 ctl=0일 때는 저장된 값을 사용한다. 그림 2의 DG에서 첫 열에 있는 원으로 표시된 계산 점들의 시간스텝의 차는 2이다. 따라서 제어신호도 PE를 따라 흐를 때 시간차를 두기 위해서 지연이 필요하다. DG에서 왼쪽에서 오른쪽으로 흐르는 데이타(A, b_i)의 흐름에는 지연이 필요 없고, 위에서 아래로 흐르는 데이타(p, M)의 흐름에는 지연이 필요하다.

앞에서 구한 변환행렬 T로 그림 2의 DG를 변환하면 다음의 그림 3와 같이 준 시스톨릭 어레이(semisystolic array) 구조의 곱셈기를 유도할 수 있다. 그림 3에서 각 PE에 동시에 입력되는 데이타 b_i를 다른 데이타처럼 같은 방향인 왼쪽에 오른쪽으로 흐르게 하여야 한다. 각 PE에서 제어신호(ctl)가 "1"인 경우에, 다시 말해서 각 PE가 처음 수행될 경우, 데이타 b_i를 입력해서 저장하여 사용하면 된다. 그림 3에서 기호(■)는 시간지연을 위한 래치(latch)이다.

그림 3에서 각 PE에 동시에 입력되는 데이타 b_i를 제거하여 다음 그림 4와 같은 시스톨릭 곱셈기를 설계할 수 있다.

한 PE의 구조를 도시한 회로는 그림 5와 같다.

$GF(2^4)$상의 시스톨릭 어레이 구조의 LSB-first 곱셈기는 그림 4와 같지만, 지금까지 설명된 설계과정은 일반적인 $GF(2^m)$상에서의 곱셈 알고리즘에도 그대로 적용할 수 있다. 그림 4의 시스톨릭 곱셈기는 4개의 PE를 가지지만, $GF(2^m)$상의 시스톨릭 곱셈기는 m개의 PE를 가진다.

설계된 시스톨릭 곱셈기에서 올바른 계산이 수행되도록 입력 데이타가 정확한 때에 정확한 PE에 입력이 되어야 한다. 시간 변환벡터와 공간 변환벡터에 의해서 PE의 링크방향과 지연시간이 결정되며, 또한 초기에 어레이의 왼쪽에 입력되는 데이타의 일련의 순서가 이들

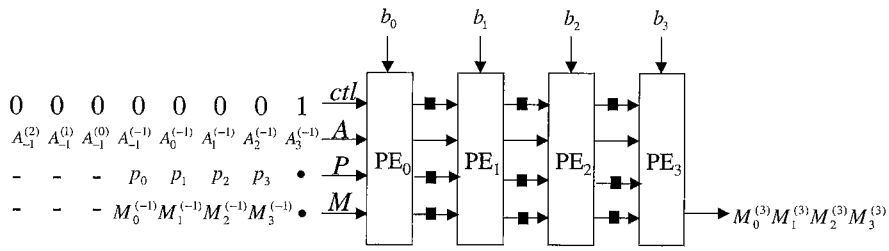

그림 3 $GF(2^4)$상의 준 시스톨릭 어레이 구조인 곱셈기

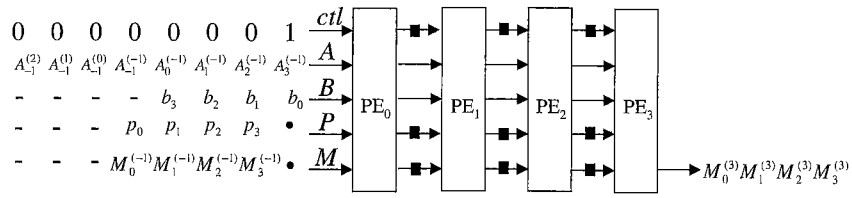

그림 4 $GF(2^4)$상의 시스톨릭 어레이 구조인 곱셈기

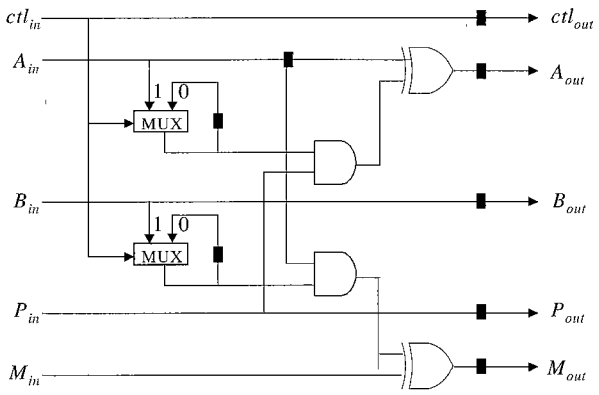

그림 5 시스톨릭 곱셈기의 PE 구조

에 의해서 결정된다. $GF(2^m)$상의 시스톨릭 곱셈기에 입력되는 데이타 ctl, A, B, P, M의 초기 입력 배치를 I_{ctl}, I_A, I_B, I_P, I_M이라 하면 각각은 다음과 같다. 여기서, | |은 수열의 길이이다.

$I_{ctl}=\{1, 0, 0, ..., 0\}$, $|I_{ctl}|=2m$

$I_A=\{a_{m-1}, a_{m-2}, ..., a_1, a_0, 0, 0, ..., 0\}$, $|I_A|=2m$

$I_B=\{b_0, a_1, ..., b_{m-2}, b_{m-1}\}$, $|I_B|=m$

$I_P=\{\bullet, p_{m-1}, p_{m-2}, ..., p_1, p_0\}$, $|I_P|=m+1$

$I_M=\{\bullet, 0, 0, ..., 0, 0\}$, $|I_M|=m+1$

입력 데이타의 수열 I_M의 값이 전부 0이므로 외부에서 입력하지 않고 시스톨릭 곱셈기의 첫 번째 PE에서 생성하여 계산에 사용하는 것이 가능하다.

곱셈기의 계산이 시작되어 $2m$ 시간스텝 후에 출력 링크상에 모듈러 곱셈의 결과 값의 최상위 비트가 출력되기 시작하여 m 시간스텝 후에 최하위 비트까지 전부가 계산되어진다.

3.3 시스톨릭 곱셈기의 분석

본 논문에서는 제안한 LSB-first문제-크기 시스톨릭 곱셈기를 Yeh와 Wang의 곱셈기와 비교하였는데, 이는 이들의 곱셈기가 가장 기본적이고 제안한 것과 같은 문제-크기 비트별 곱셈기이기 때문이고 최근 발표되는 같은 방식의 문제-크기 시스톨릭 곱셈기가 없기 때문이다. Yeh[11] 등은 LSB-first 방식으로 시스톨릭 곱셈기를 설계하였지만, Yeh의 곱셈 알고리즘은 본 논문에서 유도한 알고리즘보다 좀 더 복잡하다. 그래서 곱셈기의 PE는 2개의 제어신호를 가지는 구조이며, 임계 경로(critical path)는 한 개의 래치, AND, 및 XOR 게이트로 이루어졌다.

한편, Wang[12]등은 MSB-first 방식으로 시스톨릭 곱셈기를 설계하였다. 이 곱셈기는 시스톨릭 곱셈기의 PE의 복잡도에서는 본 논문에서 제안한 곱셈기와 거의 같으나, 한 PE내에서의 임계경로는 한 개의 래치와 AND 게이트, 및, 두 개의 XOR 게이트로 구성되어 한 개의 XOR 케이트를 더 통과하여야 하므로 한 PE의 지연시간이 더 길어진다. 표 1은 제안한 LSB-first 시스톨릭 곱셈기와 기존의 곱셈기와의 특성 비교를 보여준다.

제안한 시스톨릭 곱셈기의 면적(area)과 계산시간(time)을 다른 곱셈기와 비교하기 위해서 설계에서 많이 인용되는 각 회로 소자에 대한 지연시간(delay time)과 면적에 관해서 다음과 같이 가정한다[19].

표 1 $GF(2^m)$상의 시스톨릭 곱셈기의 특성 비교

	Yeh의 시스톨릭 곱셈기	Wang의 시스톨릭 곱셈기	제안한 시스톨릭 곱셈기
PE의 개수	m	m	m
수행 시간	$3m$	$3m$	$3m$
PE 복잡도	3 AND 게이트 2 XOR 게이트 12 Latchs 2 MUX(1X2)	3 AND 게이트 2 XOR 게이트 10 Latchs 2 MUX(1X2)	2 AND 게이트 2 XOR 게이트 11 Latchs 2 MUX(1X2)
입력 port 수	6	5	5
제어신호 개수	2	1	1
임계 경로 길이	2	3	2

$T_{2AND}=2\Delta$, $A_{2AND}=2A_U$ $T_{2XOR}=4\Delta$, $A_{2XOR}=3A_U$

$T_{2MUX}=3\Delta$, $A_{2MUX}=3A_U$ $T_L=7\Delta$, $A_L=8A_U$

여기서 T_{XGATE}와 A_{XGATE}는 X개의 입력을 가지는 게이트의 계산시간과 면적을 나타내며, T_L와 A_L는 한 비

트 래치(latch)의 지연 시간(delay)과 면적을 나타낸다. 또 Δ와 A_U는 invert 회로의 지연시간과 면적을 각각 의미한다.

다음의 표 2와 3는 기존의 문제-크기 시스톨릭 곱셈기와 제안한 시스톨릭 곱셈기의 전체 면적과 전체 수행 시간을 각각 보여준다.

표 2 문제-크기 시스톨릭 곱셈기의 전체 면적

곱셈기	PE의 면적	전체면적, A
Yeh의 곱셈기	$3A_{2AND}+2A_{2XOR}+2A_{2MUX}+12A_L$ $=114A_U$	$114mA_U$
Wang의 곱셈기	$3A_{2AND}+2A_{2XOR}+2A_{2MUX}+10A_L=98A_U$	$98mA_U$
제안한 곱셈기	$2A_{2AND}+2A_{2XOR}+2A_{2MUX}+11A_L$ $=104A_U$	$104mA_U$

표 3 $GF(2^m)$상에서 시스톨릭 곱셈기의 전체 시간

곱셈기	한 PE의 지연시간	전체면적, T
Yeh의 곱셈기	$T_{2MUX}+T_{2AND}+T_{2XOR}+T_L=16\Delta$	$48m\Delta$
Wang의 곱셈기	$T_{2MUX}+T_{2AND}+2T_{2XOR}+T_L=20\Delta$	$20m\Delta$
제안한 곱셈기	$T_{2MUX}+T_{2AND}+T_{2XOR}+T_L=16\Delta$	$48m\Delta$

표 4 $GF(2^m)$상에서 제안한 시스톨릭 곱셈기의 AT 성능향상

곱셈기	AT 복잡도	성능향상
Yeh의 곱셈기	$114mA_U * 48m\Delta=5472m^2A_U\Delta$	10%
Wang의 곱셈기	$98mA_U * 60m\Delta=5880m^2A_U\Delta$	18%
제안한 곱셈기	$104mA_U * 48m\Delta=4992m^2A_U\Delta$	–

표 4에서 보는 바와 같이 본 논문에서 제안한 LSB-first 시스톨릭 곱셈기가 Yeh의 곱셈기와 Wang의 곱셈기보다 각각 10%와 18% 면적-시간 성능이 향상되었음을 알 수 있다. 더욱이, 제안한 모듈러 곱셈 알고리즘은 병렬로 수행할 수 있는 순환식으로 구성되어서 Wang 알고리즘보다 곱셈과 제곱(square) 연산을 동시에 수행하는 시스톨릭 어레이를 설계하기가 훨씬 더 쉽다.

4. 곱셈과 제곱 동시 계산 알고리즘과 시스톨릭 곱셈/제곱기

4.1 곱셈과 제곱을 동시 계산 하는 알고리즘

모듈러 곱셈을 이용하여 모듈러 지수 연산을 수행하는 가장 일반적인 방법은 Knuth의 이진 제곱과 곱셈(binary square and multiply) 알고리즘이다. 이 알고리즘에는 left-to-right 방식과 right-to-left 방식이 있다[1]. 지수연산($X=Y^e$)을 위한 right-to-left 방식은 지수의 최하위 비트에서 최상위 비트를 보면서 각 비트의 값에 따라 곱셈($Y=X*Y$)과 제곱연산($X=X*X$)을 반복한다.

Right-to-left방식에서는 각 반복마다 제곱 연산과 곱셈을 동시에 수행할 수 있다. 따라서 곱셈과 제곱 연산을 동시에 수행하는 하드웨어는 지수 연산에 아주 효율적으로 이용될 수 있다.

LSB-first 곱셈 알고리즘의 특성을 이용하여 곱셈과 제곱 연산을 병렬로 수행할 수 있는 시스톨릭 곱셈/제곱기를 설계한다.

곱셈 $M(x)=A(x)B(x) \bmod P(x)$와 제곱 연산 $S(x)=A(x)A(x) \bmod P(x)$의 계산과정을 전개하면 아래와 같다.

$$\begin{aligned} M(x)&=A(x)B(x) \bmod P(x)\\ &=b_0A(x)+b_1[A(x)x \bmod P(x)]\\ &\quad +b_2[A(x)x^2 \bmod P(x)]+\ldots+b_{m-1}[A(x)x^{m-1} \bmod P(x)]\\ S(x)&=A(x)A(x) \bmod P(x)\\ &=a_0A(x)+a_1[A(x)x \bmod P(x)]\\ &\quad +a_2[A(x)x^2 \bmod P(x)]+\ldots+a_{m-1}[A(x)x^{m-1} \bmod P(x)] \end{aligned}$$

위의 두 식에서 [ ]안에 있는 계산은 같음을 알 수 있다. 따라서 곱셈과 제곱 연산을 동시에 수행할 때 공통부분을 한번만 계산하면 효율적이다. 앞 절에서 설명한 곱셈에 대한 식 (2)와 같이 위의 제곱 연산을 같은 방법으로 식 (9)과 같이 유도할 수 있다. $(1\le i\le m)$

$$\begin{aligned} A^{(i)}(x)&=A^{(i-1)}(x)x \bmod P(x)\\ S^{(i)}(x)&=S^{(i-1)}(x)+a_{i-1}A^{(i-1)}(x) \end{aligned} \tag{9}$$

여기서 $A^{(0)}(x)=A(x)$이고, $S^{(0)}(x)=0$이다. 또 식 (2)와 같이 식 (9)의 두 순환식도 병렬로 수행될 수 있다.

식 (2)의 곱셈과 식 (9)의 제곱 연산에서 식 $A^{(i)}(x)=A^{(i-1)}(x)x \bmod P(x)$은 공통으로 포함되어 있다. 따라서 다음 식 (10)와 같이 곱셈과 제곱 연산을 동시에 수행하는 순환식을 쉽게 유도할 수 있다. $(1\le i\le m)$

$$\begin{aligned} A^{(i)}(x)&=A^{(i-1)}(x)\,x \bmod P(x)\\ M^{(i)}(x)&=M^{(i-1)}(x)+b_{i-1}A^{(i-1)}(x)\\ S^{(i)}(x)&=S^{(i-1)}(x)+a_{i-1}A^{(i-1)}(x) \end{aligned} \tag{10}$$

식 (10)으로부터 $i=m$일 때, $M^{(m)}=M(x)=A(x)B(x) \bmod P(x)$와 $S^{(m)}=S(x)=A(x)A(x) \bmod P(x)$으로 모듈러 곱셈 $M(x)$과 제곱 $S(x)$를 동시에 계산할 수 있다. 식 (10)에 있는 3개의 순환식은 병렬로 수행될 수 있다. 앞 절에서 설명한 같은 방법으로 비트별 LSB-first 모듈러 곱셈/제곱을 동시에 계산하는 알고리즘을 아래와 같이 유도한다.

이 알고리즘의 루프(loop)의 구조는 3장에서 제시된 비트별 LSB-first 모듈러 곱셈 알고리즘의 루프 구조와 같다. 이 알고리즘의 자료 의존관계를 보여주는 DG는 그림 1의 LSB-first 모듈러 곱셈의 DG에 두 개의 데이타 흐름만 더 추가하면 된다. 먼저, 그림 1의 DG에 추가되는 데이타 a_{i-1}는 곱셈에 사용되는 데이타 b_{i-1}과 같이 왼쪽으로부터 i 번째 행으로 입력되어 각 계산 점에서 계산에 사용만 되며, 오른쪽 계산 점으로 전달된다. 제곱에 필요한 새로운 데이타 $S_{m-j}^{(i)}$은 그림 1의 $M_{m-j}^{(i)}$와 같이 위에서 각 계산 점(i, j)에 입력되어 알고리즘에 따라

[비트별 LSB-first 모듈러 곱셈/제곱 알고리즘]

입력 : $A=(a_{m-1}, a_{m-2}, \ldots, a_1, a_0)$

$B=(b_{m-1}, b_{m-2}, \ldots, b_1, b_0)$

$P=(1, p_{m-1}, p_{m-2}, \ldots, p_1, p_0)$

출력 : $M^{(m)}=M(x)=A(x)B(x) \bmod P(x)$,

$S^{(m)}=S(x)=A(x)A(x) \bmod P(x)$

초기치 : $M^{(0)}=(M_{m-1}^{(0)}, M_{m-2}^{(0)}, \ldots, M_1^{(0)}, M_0^{(0)})$

$=(0, 0, \ldots, 0, 0)$

$S^{(0)}=(S_{m-1}^{(0)}, S_{m-2}^{(0)}, \ldots, S_1^{(0)}, S_0^{(0)})$

$=(0, 0, \ldots, 0, 0)$

$A^{(0)}=(A_{m-1}^{(0)}, A_{m-2}^{(0)}, \ldots, A_1^{(0)}, A_0^{(0)}, A_{-1}^{(0)})$

$=(a_{m-1}, a_{m-2}, \ldots, a_1, a_0, 0)$

$A_{-1}^{(i)} \equiv 0,\ 1 \le i \le m$

순환식 : **for** $i=1$ **to** m

for $j=1$ **to** m

$$A_{m-j}^{(i)}=A_{m-1-j}^{(i-1)}+A_{m-1}^{(i-1)}\, p_{m-j}$$

$$M_{m-j}^{(i)}=M_{m-j}^{(i-1)}+b_{i-1}A_{m-j}^{(i-1)}$$

$$S_{m-j}^{(i)}=S_{m-j}^{(i-1)}+a_{i-1}A_{m-j}^{(i-1)}$$

새로운 값이 계산되어서 j 열을 따라 아래의 계산 점으로 전달된다. 특히, $j=0$일 때는 외부에서 값 0이 입력된다. 마지막 행에 있는 각 계산 점으로부터 출력되는 $S_j^{(m)}$는 제곱 연산의 결과 값의 j 번째 비트이다. 곱셈과 제곱을 동시에 계산하는 알고리즘에 대한 DG도 그림 1의 DG처럼 거의 같은 규칙적인 데이타 흐름이어서 시스톨릭 어레이 구조의 곱셈/제곱기를 설계하기가 용이하다.

4.2 시스톨릭 곱셈/제곱기의 설계

곱셈과 제곱을 동시에 수행하는 알고리즘의 DG는 앞 절에서의 설명처럼 모듈러 곱셈의 DG에 같은 방향으로 새로운 2개의 데이타 흐름만 더 추가되므로 시스톨릭 곱셈기와 같은 공간-시간 변환을 적용하여 시스톨릭 곱셈/제곱기를 설계할 수 있다. 시스톨릭 곱셈기의 설계에 사용되었던 식 (8)에 있는 공간-시간 변환 T를 사용하여 설계된 $GF(2^4)$상의 시스톨릭 곱셈/제곱기는 다음 그림 6과 같다. 그림 4와 그림 6를 비교해 보면, 그림 6에서는 A^*와 S 데이타의 입력이 더 추가되었음을 알 수 있다. 데이타 S는 데이타 M과 같은 패턴으로 PE사이를 흐르며, 데이타 A^*는 데이타 B와 같은 패턴으로 흐른다.

그림 6의 시스톨릭 곱셈/제곱기의 각 PE 구조는 다음 그림 7과 같다.

일반적으로 $GF(2^m)$상의 시스톨릭 곱셈/제곱기는 그림 6에서 PE의 개수가 m개로 증가되고, 각 PE의 구조는 그림 7과 같다. 첫 PE에 입력되는 입력 데이타 ctl, A, B, P, M의 각각 초기 입력배치 I_{ctl}, I_A, I_B, I_P, I_M는 시스톨릭 곱셈기의 초기 입력배치와 같다. 새로 추가되는 입력 데이타인 A^*와 S의 초기 입력배치는 다음과 같다.

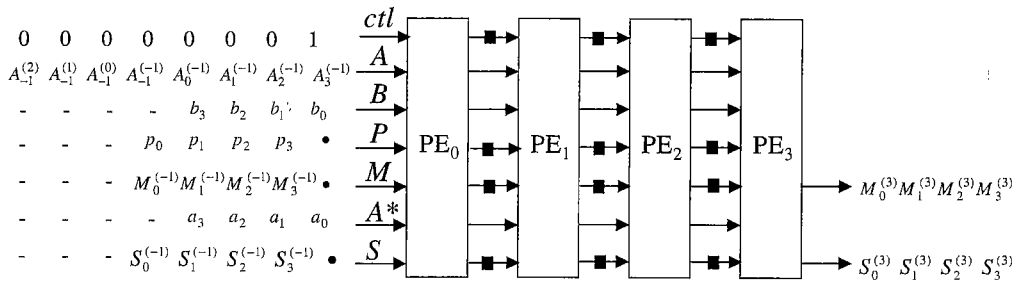

그림 6 $GF(2^4)$상의 시스톨릭 곱셈/제곱기

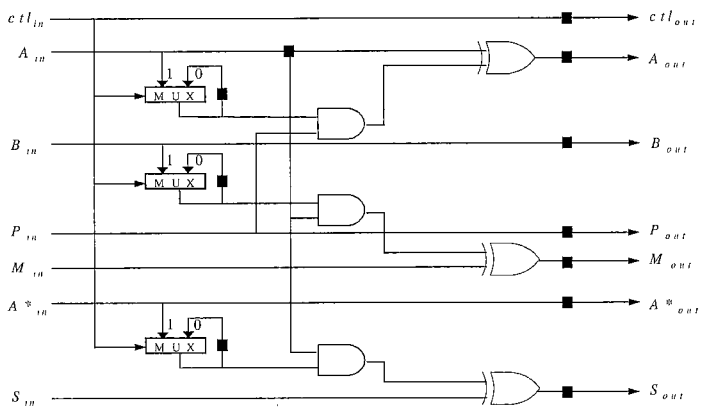

그림 7 시스톨릭 곱셈/제곱기의 PE 구조

$I_{A^*}=\{a_0, a_1, ..., a_{m-2}, a_{m-1}\}$, $|I_{A^*}|=m$

$I_S=\{\bullet, 0, 0, ..., 0, 0\}$, $|I_S|=m+1$

4.3 시스톨릭 곱셈/제곱기의 분석

논문에서 제안한 $GF(2^m)$상의 LSB-first 시스톨릭 곱셈/제곱기는 m개 PE로 구성되며, 각 PE의 구조는 그림 7과 같은데, 그림 5의 시스톨릭 곱셈기에 1개의 논리곱, 1개의 배타적 논리합, 1개의 MUX 및 2개의 래치만 더 추가하면 된다. 입력이 주어지고, 초기 지연 시간 $2m$ 시간스텝이 지난 후부터 한 스텝마다 곱셈과 제곱의 결과가 한 비트씩 출력되어져서 시간스텝 $3m$에 계산이 끝난다. 제안한 시스톨릭 곱셈/제곱기는 LSB-first 곱셈 알고리즘의 특성을 살려서 공통 부분을 한번만 계산하므로 PE의 구조가 간단하다. 또한 제안한 시스톨릭 곱셈/제곱기의 한 PE의 지연시간은 제안한 LSB-first 곱셈기와 같다. Wang이 제안한 MSB-first 곱셈 알고리즘을 이용한 곱셈/제곱기는 곱셈과 제곱을 동시에 수행하고자 할 때 같은 공통 부분이 없어서 MSB-first 곱셈 알고리즘을 두 번 이용하여야만 곱셈/제곱기를 설계할 수 있다.

제안한 LSB-first 시스톨릭 곱셈/제곱기와 Wang의 MSB-first 곱셈 알고리즘을 두 번 이용한 곱셈/제곱기의 성능의 다음 표 5와 같다.

표 5 $GF(2^m)$상의 시스톨릭 곱셈/제곱기의 AT 복잡도

곱셈/제곱기	전체 면적, A	전체 시간, T	AT 복잡도
Wang 곱셈/제곱기	$196mA_U$	$60m\Delta$	$11{,}760m^2A_U\Delta$
제안한 곱셈/제곱기	$124mA_U$	$48m\Delta$	$5{,}952m^2A_U\Delta$

표 5에서 보는 것과 같이 제안한 시스톨릭 곱셈/제곱기는Wang과 같이 MSB-first 곱셈 알고리즘을 이용한 시스톨릭 어레이 구조의 곱셈/제곱기보다 AT 복잡도에서 거의 2배 가까이 성능이 좋음을 알 수 있다.

제안한 곱셈/제곱기는 지수연산에 아주 효율적으로 이용될 수 있다. 지수연산에서 m 비트의 지수가 1과 0이 반반이라 가정할 때, 지수연산에 곱셈/제곱기를 사용할 때는 m번의 반복으로 곱셈/제곱기를 수행하면 된다. 한편, 곱셈기만을 사용할 때는 m번의 제곱과 $m/2$번의 곱셈을 반복 수행하여야 하므로 $3/2m$번의 곱셈기를 사용한다. 모듈러 지수연산을 수행할 때 제안한 곱셈/제곱기를 사용하는 경우와 앞에서 언급한 3개의 곱셈기중에서 AT 복잡도에서 가장 좋은 본 논문에서 제안한 곱셈기 하나만을 사용할 경우의 AT 복잡도 비교는 다음 표 6과 같다.

표 6 모듈러 지수연산의 AT 복잡도

곱셈/제곱기	전체 면적, A	전체 시간, T	AT 복잡도
Wang 곱셈/제곱기	$104mA_U$	$3/2m*48m\Delta$ $=72m^2\Delta$	$7{,}488m^3A_U\Delta$
제안한 곱셈/제곱기	$124mA_U$	$m*48m\Delta$ $=48m^2\Delta$	$5{,}952m^3A_U\Delta$

표 6은 제안한 시스톨릭 곱셈/제곱기를 모듈러 지수연산에 사용할 때는 곱셈기만을 사용할 때 보다 AT 복잡도에서 약 26%의 향상이 있음을 보여준다.

5. 결 론

본 논문에서는 유한 필드 $GF(2^m)$상에서 모듈러 곱셈

$A(x)B(x)$ mod $P(x)$을 위한 새로운 선형 시스톨릭 어레이 구조인 최적의 LSB-first 시스톨릭 곱셈기를 제안하고, 또 곱셈과 제곱연산을 동시에 수행하는 새로운 시스톨릭 곱셈/제곱기를 제안하였다. 모듈러 곱셈에서 피연산자 $B(x)$의 LSB로부터 먼저 사용한 LSB-first 모듈러 곱셈 알고리즘으로부터 비트별 순환 방정식으로 유도한다. 이 순환 방정식은 자료의 흐름이 정규적이고 규칙적인 특성을 가지고 있어서 시스톨릭 어레이 구조의 곱셈기를 설계가 용이하였다. 순환 방정식으로부터 자료의 흐름을 쉽게 분석하기 위해 DG를 도시하고, DG로부터 시스톨릭 어레이를 설계하기 위해 필요한 공간 변환벡터와 시간 변환벡터를 구하여 공간-시간 변환으로 선형의 LSB-first 시스톨릭 곱셈기를 설계하였다. 본 연구에서 설계된 시스톨릭 곱셈기를 기존의 모듈러 곱셈기와 비교하여 제안한 시스톨릭 곱셈기가 기존의 시스톨릭 곱셈기보다 구조가 간단하거나 시간적으로 성능이 좋은 곱셈기임을 보였다. 제안한 곱셈기는 Yeh의 곱셈기와 Wang의 곱셈기보다 AT 복잡도면에서 각각 10%와 18%의 성능이 좋아졌음을 밝혔다.

더욱이, LSB-first 방식의 곱셈 알고리즘의 특성으로부터 곱셈과 제곱연산을 동시에 수행할 때 같은 공통의 식을 이용할 수 있어서 곱셈/제곱기를 설계할 때 다른 곱셈기들 보다 이득이 있다. 본 논문에서 곱셈기의 설계 방법과 같은 공간-시간 변환으로 구조적으로나 시간적으로 효율이 좋은 시스톨릭 곱셈/제곱기를 제안하였다. 제안한 곱셈/제곱기는 MSB-first 곱셈 알고리즘을 이용한 곱셈/제곱기보다는 약2배의 성능이 좋음을 보였다. 모듈러 지수연산에 제안한 곱셈/제곱기를 사용할 때는 곱셈기만을 사용할 때 보다 AT 복잡도면에서 26%의 성능향상이 됨을 보였다. 제안한 곱셈/제곱기가 지수연산에 아주 효율적으로 활용될 수 있었어 암호용 프로세서 설계에 활용될 것으로 기대된다.

참 고 문 헌

[1] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*, New York: Kluwer-Academic, 1987.

[2] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Trans. on Info. Theory*, vol. IT- 22(6) pp. 644-654, 1976.

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

[4] S. T. Fenn, M. Benaissa, and D. Taylor, "*GF*(2^m) Multiplication and division over the dual base," *IEEE Trans. Computers*, vol. 4, no. 3, pp. 319-327, Mar. 1996.

[5] I. S. Hsu, I. S. Reed, T. K. Truong, K. Wang, C. S. Yeh, and L. J. Deutsch, "The VLSI implementation of a Reed-Solomon encoder using Berlekamp's bit-serial multiplier algorithm," *IEEE Trans. Computer*, vol. C-33, pp. 906-911. Oct. 1984.

[6] T. Itoh and S. Tsujii, "Structure of parallel multipliers for Galois Fields *GF*(2k)," *Information and Computers*, vol. 83, pp. 21-40, 1989.

[7] C. K. Koc and T. Acar, "Montgomery Multiplication in *GF*(2k)," *Proceedings of Third Workshop on Selected Area in Cryptography*, pp. 95-106, queen's University, Kinston, Ontario, Canada, August 15-16 1996.

[8] E. D. Mastrovito, "VLSI Design for multiplication over Finite Fields *GF*(2^m)," *Lecture Notes in Computer Science 357*, pp. 297-303, Berlin: Springer-Verlag, Mar. 1989.

[9] P. A. Scott, S.E. Tavares, and L.E. Peppard, "A fast VLSI multiplier for *GF*(2^m)," *IEEE J. Selected Areas in Comm.*, vol. 4. pp.62-66, Jan., 1986.

[10] C. C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI architectures for computing multiplications and inverses in *GF*(2^m)," *IEEE Trans. Computer*, vol. C-34, pp. 709-717, Aug., 1985.

[11] C. S. Yeh, I. S. Reed, and T. K. Truong, "Systolic multipliers for finite fields *GF*(2^m)," *IEEE Trans. Computer*, vol. C-33, pp.357-360, Apr., 1984.

[12] C. L. Wang and J. L. Lin, "Systolic Array Implementation of Multipliers for finite fields *GF*(2^m)," *IEEE Trans. Circuits Systems*, vol. 38, pp. 796-800, July 1991.

[13] I. S. Hsu. T. K. Truong, L.J. Deutsch, and I. S. Reed, "A Comparison of VLSI architecture of finite field multipliers using dual, normal, standard bases," *IEEE Trans. Computer*, vol. 37, pp. 735-739, June 1988.

[14] S. K. Jain and K. K. Parhi, "Low latency standard basis *GF*(2^m) multiplier and squarer architectures, *Proc. IEEE ICASSP*, pp. 2747-2750, 1995.

[15] S. K. Jain, L. Song, and K. K. Parhi, "Efficient Semisystolic Architectures for Finite-Field Arithmetic," *IEEE Trans. On VLSI Systems*, VOL. 6, NO. 1, pp. 101-113, 1998.

[16] L. Song and K. K. Parhi, "Effiecient finite field serial/parallel multiplication," *Proc. Int. Conf.*

Application Specific Syst., Architectures and Processors, Chicago, IL, pp. 72-82, 1996.
[17] S. Y. Kung, *VLSI Array Processors*, Prentice-Hall, 1987.
[18] K. Y. Yoo, *"A Systolic Array Design Methodology for Sequential Loop Algorithms,"* Ph.D. thesis, Rensselaer Polytechnic Institute, New York, 1992.
[19] K. Hwang, Computer Arithmetic, John Wiley & Sons, 1979.

유 기 영

1976년 경북대학교 수학교육학과 졸업(이학사). 1978년 한국과학기술원 전산학과 졸업(공학석사). 1992년 미국 Rensselaer Polytechnic Institute 졸업(이학박사). 1978년 ~ 현재 경북대학교 컴퓨터공학과에 재직. 관심분야는 병렬처리, array processor 설계, 암호칩 설계, 스마트 카드, 정보보호 등임.

김 정 준

1981년 경북대학교 전자공학과 졸업(공학사). 1983년 한국과학기술원 전기 및 전자공학과 졸업(공학석사). 1997년 루이지애나주립대 전기 및 컴퓨터공학과 졸업(공학박사). 1984년 ~ 현재 한국통신 가입자망연구소 무선ATM연구실장. 관심분야는 무선ATM, 무선망보안, 스마트카드 기술