셀룰러 오토마타를 이용한 LSB 곱셈기 설계
(Design of LSB Multiplier using Cellular Automata)

하경주* 구교민**
(Kyeoung-Ju Ha) (Kyo-Min Ku)

요 약 GF(2^m)상에서 모듈러 곱셈은 공개키 암호 시스템과 같은 응용에서의 기본 연산으로 사용된다. 본 논문에서는 이와 같은 모듈러 곱셈 연산을 셀룰러 오토마타를 이용하여, GF(2^m)상에서 m 클럭 사이클만에 처리할 수 있는 연산기를 설계하였다. 이 곱셈기는 LSB 우선 방식으로 설계되었으며, 기존의 시스톨릭 구조를 이용한 곱셈기 보다 하드웨어 복잡도가 낮고 처리 시간이 빠른 장점이 있다. 그리고 설계된 곱셈기는 지수연산을 위한 하드웨어 설계에 효율적으로 이용될 수 있을 것이다.

Abstract Modular Multiplication in Galois Field GF(2^m) is a basic operation for many applications, particularly for public key cryptography. This paper presents a new architecture that can process modular multiplication on GF(2^m) per m clock cycles using a cellular automata. Proposed architecture is more efficient in terms of the space and time than that of systolic array. Furthermore it can be efficiently used for the hardware design for exponentiation computation.

1. 서 론

인터넷의 급성장으로 인해, 인터넷을 통한 데이터의 전송이 점차적으로 많아지고 있으며, 안전한 데이터의 전송을 위해서 반드시 해결해야 할 문제 중의 하나가 보안 문제이다. 특히 전자상거래 등에서는 개인 신상 정보 등의 유출 문제가 매우 심각하므로 이러한 보안문제가 반드시 해결되어야 한다. 따라서 보안기술의 핵심적인 요소라 할 수 있는 암호 시스템 구현의 중요성 또한 크게 부각되고 있는 실정이다. 특히 공개키 암호 시스템은 비밀키 시스템에서의 단점인 키 분배 등의 문제를 해결하고, 또한 전자 서명과 같은 많은 응용에 이용됨으로써 많은 장점을 가지고 있다. 이러한 공개키 암호화 시스템에서는 유한 필드 상의 모듈러 지수(modular exponetiation) 연산을 기본으로 하고 있으며, 대표적인 암호 알고리즘으로는 Diffie-Hellman key exchange 방식, ElGamal 암호 방식 등이 있다[2][3]. 특히 ECC와 같은 암호 시스템은 다른 암호시스템에 비해서 효율적인 하드웨어 복잡도를 가지며 시스템을 구성하는데 있어서 역원계산이 필요하다[4]. 또한 Diffie-Hellman key exchange 방식이나 ElGamal 암호 시스템은 암호화 복호화 하는데 있어서 지수 연산을 기본으로 하고 있다. 이러한 역원이나 지수 연산기를 구현하기 위해서는 모듈러 곱셈(modular multiplication) 연산이 기본적으로 사용된다. 따라서 이러한 연산들을 구현하기 위한 기본 구조로 효율적인 모듈러 곱셈기가 필요하다. GF(2^m) 상에서 곱셈기를 구현하기 위한 알고리즘으로는 LSB 우선 곱셈 알고리즘[5]과 MSB 우선 곱셈 알고리즘[6] 및 몽고메리(montgomery) 알고리즘[7] 등이 있다.

지금까지 이러한 기본 구조들은 주로 시스톨릭(systolic) 구조에서 개발되어져 왔다[5, 6]. 그러나 이들은 하드웨어 복잡도가 높고 처리 시간이 길어 암호 시스템의 응용에는 적합하지 못하다. 특히 근래에 들어 PC 카드 형태로 개발되어 사용되는 보안토큰(cryptographic token)은 다양한 보안서비스를 바탕으로 차세대 정보보호 기술의 핵심 기술로 떠오르고 있다. 보안토큰은 주로 카드 형태 즉 PCMCIA 카드와 스마트(smart) 카드의 형태로 개발되어 사용되고 있는데, 암호시스템이 보안토큰의 형태로 개발되기 위해서는 무엇보다도 하드웨어 복잡도가 중요하다.

본 논문에서는 기존의 시스톨릭 구조나 지금까지 CA 상에서 제시된 곱셈기 보다 하드웨어 복잡도가 낮고 처리 시간이 빠른 곱셈기를 설계하였다. 특히 3-이웃 셀룰

* 경산대학교 정보과학부
** 대구교육대학교

러 오토마타(Cellular Automata : CA)는 간단하고도 규칙적이며, 모듈화 하기 쉽고 계층화하기 쉬운 구조적 특징을 가지고 있어 암호화 시스템에서 대칭키의 암호화 복호화등 많은 응용 분야에 사용되고 있다[8][9].

지금까지 GF(2^m)상에서 모듈러 곱셈 연산을 위해 개발된 연구 결과들은 다음과 같다. 먼저 1차원 시스톨릭 구조상에서는 LSB, MSB 우선 알고리즘의 경우 모두 m 셀을 사용하며 $3m$의 계산 지연시간을 가진다[5, 6]. CA 구조상에서는 [10]에서 m 셀을 사용하여 m 클럭 사이클에 모듈러 곱셈 연산을 제시하고 있으나, [10]에서는 완전한 구조를 제시하고 있지는 못하다.

본 논문에서는 m 셀, $2m$-1 AND 게이트, $2m$-1 XOR 게이트 그리고 $3m$-1 레지스터를 사용하여 m 클럭 사이클만에 모듈러 곱셈 연산을 수행할 수 있는 연산기를 설계하였다.

본 논문의 구성은 다음과 같다. 2장에서는 유한필드에 대한 기본적인 개념 및 CA에 대해 살펴보고, 3장에서는 GF(2^m) 상에서 LSB 우선 방식의 모듈러 곱셈을 위한 구조를 살펴본다. 4장에서는 CA를 이용한 모듈러 곱셈의 효율적인 구조를 제시하고, 5장에서 시뮬레이션 결과 및 성능에 대해 분석한다. 마지막으로 6장에서 결론을 맺는다.

2. 유한 필드 및 CA

2.1 유한필드

본 절에서는 유한 필드와 관련된 몇 가지 기본적인 개념과 표현법을 살펴본다[1]. 유한 필드는 Galois 필드(GF)로도 불리우며, 비록 유한 필드가 많은 소수의 지수 차수에 대해 존재하지만, 암호학에서 주로 사용되는 필드는 소수 q에 대한 소수 유한 필드 GF(q)와 약수 m에 대한 이진 유한 필드 GF(2^m)이다. 유한필드는 GF(2^m)은 길이가 m인 2^m개의 가능한 비트 스트링으로 구성된다.

$$GF(2^m)=\{(a_{m-1}a_{m-2}\cdots a_1a_0)|a_i\in GF(2),\ 0\le i\le m-1\}$$

여기서 GF(2)는 GF(2^m)의 하부필드(Sub - field)라 불리고, 유한필드 GF(2^m)은 GF(2)의 확장 필드라고 한다. 필드에서의 원소들을 표현하기 위해서는 정규 기저(normal basis) 표기법, 이원기저(dual basis) 표기법, 다항식기저(polynomial basis) 표기법 등의 세 가지 표기법이 있다. 그러나 정규 기저 표기법과 이원기저 표기법은 하드웨어 구현 시 규칙적이지 않기 때문에 즉, m에 따라 하드웨어 구조가 달라지기 때문에 비효율적이다. 본 논문에서는 필드상의 원소들을 다항식 기저 표기법으로 표현한다. 다항식 기저 표기법에서의 GF(2^m)의 각 원소는 다음과 같이 m 차수 미만의 다항식으로 표현된다.

$$A(x)=a_{m-1}x^{m-1}+\cdots+a_1x^1+a_0x^0,\ a_i\in GF(2),$$
$$0\le i\le m-1.$$

GF(2^m) 상에서 연산 후 연산 결과를 필드의 원소로 만들기 위해서는 차수 m의 기약 다항식(irreducible polynomial)이 필요하고, 이 기약 다항식을 이용한 모듈러 연산이 필요하다. 차수 m의 기약 다항식 $P(x)$ 는 다음과 같이 표현된다.

$$P(x)=p_mx^m+\cdots+p_1x^1+p_0x^0,\ \ p_i\in GF(2),$$
$$0\le i\le m\ [1].$$

2.2 CA

CA는 규칙적으로 상호 연결된 많은 셀들로 구성되어져 있으며, 각 셀들의 다음 상태는 각 셀들과 연결된 이웃의 현재 상태 값에 따라 달라지게 된다[8,9]. 각 셀이 두 개의 상태를 가지고 세 이웃을 가지는 다음과 같은 CA를 고려해 보자.

이웃의 상태 : 111 110 101 100 011 010 001 000
다음 상태 : 0 1 0 1 1 0 1 0

(법칙 90)
다음 상태 : 1 0 1 0 1 0 1 0
(법칙 170)

여기서, 이웃의 상태라 함은 시간 t에 3개의 이웃이 가질 수 있는 가능한 8개의 상태를 나타내며, 이를 나타내는 3개의 비트 중 가운데 비트가 자신의 상태를 나타내고, 이의 왼쪽, 오른쪽 비트는 각각 왼쪽, 오른쪽 이웃의 상태를 나타낸다. 법칙 90과 170은 각각 시간 t+1에 i번째 셀이 가지는 상태를 나타내고 있다. 여기서 90과 170의 의미는 다음 상태 8 비트를 십진수로 나타내었을 때의 값을 의미한다. 특히 법칙 170의 경우 본 논문에서 곱셈기 설계 시 사용되는 규칙이다. 두개의 법칙 90과 170은 다음과 같은 논리 함수로 나타낼 수 있다.

법칙 90 : $q_i(t+1) = q_{i-1}(t)\oplus q_{i-1}(t)$
법칙 170 : $q_i(t+1) = q_{i-1}(t)$

여기서 $\oplus$은 XOR 연산을 나타내며, $q_i(t)$은 시간 t에서 i번째 셀의 상태이며, $q_{i-1}(t)$과 $q_{i-1}(t)$은 i번째 셀의 왼쪽, 오른쪽 이웃의 상태를 나타낸다.

CA가 위의 법칙과 같이 XOR만으로 나타낼 수 있을

때 이를 선형(linear) CA라고 하며, 다음은 법칙 90과 170 이외의 선형 CA 법칙을 나타낸다.

법칙 60 : $q_i(t+1) = q_{i-1}(t) \oplus q_i(t)$
법칙 102 : $q_i(t+1) = q_i(t) \oplus q_{i-1}(t)$
법칙 150 : $q_i(t+1) = q_{i-1}(t) \oplus q_i(t) \oplus q_{i-1}(t)$
법칙 204 : $q_i(t+1) = q_i(t)$
법칙 240 : $q_i(t+1) = q_{i-1}(t)$

그리고 CA의 구조에서 경계 조건(boundary condition)을 고려했을 때, 가장 왼쪽 셀과 가장 오른쪽 셀이 서로 이웃한 것으로 간주하는 CA 를 PBCA(Periodic Boundary CA) 라한다.

n개의 셀을 가지는 CA의 현재 상태는 n-벡터 $x=(x_0x_1\cdots x_{n-1})$로 나타낼 수 있다. 여기서 x_i는 셀 i의 값이며, x_i는 GF(2)의 원소이다. 선형 CA에서 다음의 상태는 특성 행렬과 현재 상태의 벡터를 곱함으로써 결정될 수 있는데, 이 때 특성 행렬은 CA의 전체적인 법칙을 나타낸다. x^t를 시간 t에서의 CA의 상태라 하고(하나의 열벡터로 간주), 특성 행렬을 T라하면, 시간 $t+1$에서의 CA의 상태는 다음과 같이 표현된다.

$$x^{t+1} = T\,x^t$$

여기서의 산술 연산은 GF(2) 상에서 일어난다.

법칙 90에 대한 PBCA인 경우 특성 행렬은 다음과 같다.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

위의 예에서 1행 2열의 원소가 1임이 의미하는 것은 CA의 첫 번째 셀의 다음상태가 그의 오른쪽 이웃인 두 번째 셀의 상태와 연관이 있음을 나타낸다. 즉 i번째 행 j번째 열에 있는 행렬의 요소 1은 i번째 셀이 j번째 셀과 이웃에 대한 의존도가 있음을 나타낸다.

3. GF(2^m) 상에서 곱셈 연산을 위한 알고리즘

모듈러 곱셈에는 두 가지 알고리즘 즉 피연산자 $B(x)$의 LSB부터 먼저 연산을 시작하는 LSB 우선 알고리즘과 MSB부터 먼저 연산을 시작하는 MSB 우선 곱셈 알고리즘이 있다[5,6]. 본 논문에서는 LSB 우선 방식의 곱셈기를 설계하므로, 본 장에서는 LSB 우선 연산 방식의 곱셈에 대하여 알아본다.

곱셈은 차수 m의 원시다항식에 의해 정의된다. 두 원소의 곱셈은 단순히 두 다항식을 곱한 뒤 원시 기약 다항식인 $P(x)$로 모듈러 연산을 취한다. $A(x)$와 $B(x)$를 GF(2^m) 상의 한 원소라 하자. 그리고 $P(x)$는 차수 m의 원시 기약 다항식이라 하자[1]. 그러면 세 개의 다항식 $A(x)$, $B(x)$, $P(x)$는 다음과 같다.

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

다항식 $M(x)$는 $A(x)$와 $B(x)$의 LSB부터 처리한 곱셈을 $P(x)$로 모듈러 연산을 수행한 결과이다.

$$\begin{aligned} M(x) &= A(x)B(x) \bmod P(x) \\ &= b_0A(x)+b_1[A(x)x \bmod P(x)] + \\ &\quad \cdots + b_{m-1}[A(x)x^{m-1} \bmod P(x)] \end{aligned} \quad (1)$$

식 (1)을 비트별 연산으로 고침으로서 다음의 비트별 LSB 우선 방식의 모듈러 곱셈 알고리즘을 유도할 수 있다[11].

알고리즘 1. LSB 우선 모듈러 곱셈 알고리즘[11]

입력 : $A(x)$, $B(x)$, $P(x)$
출력 : $M(x)=A(x)B(x) \bmod P(x)$
1 : $M^{(0)}(x)=0$, $A^{(0)}(x)=0$
2 : *for* $i=1$ *to* m
3 : $A^{(i)}(x)= A^{(i-1)}(x)x \bmod P(x)$
4 : $M^{(i)}(x)= M^{(i-1)}(x)+b_{i-1}A^{(i-1)}(x)$
5 : *end for*

위 알고리즘 1에서 3과 4번 문은 독립적으로 수행될 수 있다.

4. CA를 이용한 모듈러 곱셈 연산기

본 장에서는 CA를 이용하여 GF(2^m) 상에서 $A(x)B(x) \bmod P(x)$를 LSB 우선 방식으로 빠른 시간에 계산 할 수 있는 구조를 제시하고자 한다.

3장에서 제시한 모듈러 곱셈인 $M(x) = A(x)B(x) \bmod P(x)$는 식 (1)과 같이 표현될 수 있으며, 식(1)은 다음과 같은 순환(recurrence) 형태로 다시 나타낼 수 있다.

$$M^{(i)}(x)=M^{(i-1)}(x)+b_{i-1}A^{(i-1)}(x),$$
$$A^{(i)}(x)=A^{(i-1)}(x)x \bmod P(x), \text{ for } 1\le i\le m \quad (2)$$

여기서 $A^{(0)}(x)=A(x)$, $M^{(0)}(x)=0$, $M^{(i)}(x) = b_0A(x) + b_1[A(x)\ x \bmod P(x)] + b_2[A(x)\ x^2 \bmod P(x)] + \cdots + b_{i-1}[A(x)x^{i-1} \bmod P(x)]$이고, $i=m$ 인 경우, $M^{(m)}(x)=M(x)=A(x)B(x) \bmod P(x)$ 이다.

(2) 식에서 두 개의 식은 병렬로 수행될 수 있다.

먼저 $A(x)$와 $B(x)$를 $GF(2^m)$ 상의 한 원소라 하면, 이들을 두 개의 이진 m-튜플로 다음과 같이 나타낼 수 있다.

$$A(x) = a_{m-1} \cdots a_2\ a_1a_0 \quad (3)$$
$$B(x) = b_{m-1} \cdots b_2\ b_1b_0 \quad (4)$$

다음은 모듈러 곱셈에서 수행되는 $A^{(i)}(x) = A^{(i-1)}(x)x \bmod P(x)$ $(1 \le i \le m)$를 수행하기 위한 기본적인 연산이다.

연산 1 : $A^{(i-1)}(x)x$ 를 수행하기 위하여

$A(x)$의 m-tuple을 cyclic left shift 한다. 그 결과는 다음과 같다.

$$(a_{m-2}, \cdots, a_1, a_0, a_{m-1}) \quad (5)$$

연산 2 : 모듈러 연산을 수행하기 위하여

만약 $a_{m-1}=1$ 이면 (5)의 결과와 $(p_{m-1}, \cdots, p_2, p_1, p_0+a_{m-1})$를 비트별로 모듈러 2 더하기 연산을 행한다. 즉 다음과 같은 연산을 행한다.

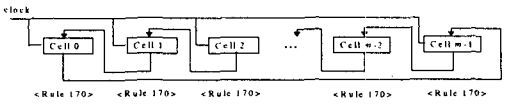
$$(a_{m-2}, \cdots, a_1, a_0, a_{m-1}) +$$
$$a_{m-1}(p_{m-1}, \cdots, p_2, p_1, p_0+a_{m-1})$$

연산 1을 수행하기 위하여, m개의 셀을 가지는 1차원 PBCA를 이용한다. 이 때 CA의 각 셀들의 다음 상태는 현재 자신의 오른쪽 이웃의 값에 따라 결정되는데 이를 각 경우별로 표현하면 다음과 같다.

이웃의 상태	111	110	101	100	011	010	001	000
다음 상태(법칙 170)	1	0	1	0	1	0	1	0

위와 같이 연산 1을 수행하기 위한 CA는 법칙 170을 가지게 되며 cyclic left shift를 위해 가장 왼쪽 셀과 가장 오른쪽 셀이 이웃한 PBCA의 특성을 가지게 된다. 이러한 CA의 특성을 나타내는 특성행렬 T는 다음과 같으며, 그 구조도는 <그림 1>과 같다.

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$



<그림 1> 특성 행렬 T를 가지는 PBCA 구조

이제부터 <그림 1>의 구조를 가지는 CA를 특성 행렬 T를 가지는 PBCA라 하자. 그리고 연산 1의 결과로 얻어진 m-튜플(tuple)을 z라 하자.

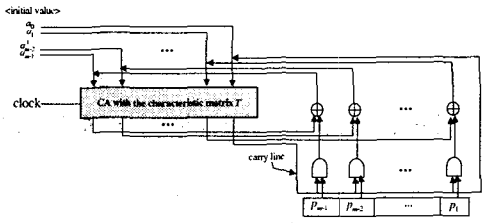
그 다음 연산 2를 수행하기 위해서는, 연산 1의 결과로 얻어진 m-튜플 z의 LSB가 1이면, 즉 a_{m-1}이 1이면, m-튜플 z와 $(p_{m-1}, \cdots, p_2, p_1, p_0 \oplus a_{m-1})$를 비트별로 XOR 연산을 행한다. 즉, a_{m-1}의 값에 따라 <표 1>과 같은 연산이 일어난다.

<표 1> 연산 2의 결과

a_{m-1}	연산 2의 결과
0	$(a_{m-2}, \ldots, a_1, a_0, a_{m-1})$
1	$(a_{m-2} \oplus p_{m-1}, \ldots, a_1 \oplus p_2, a_0 \oplus p_1, a_{m-1} \oplus p_0 \oplus a_{m-1})$

<표 1>의 결과를 살펴보면 a_{m-1}이 "0"인 경우 결과의 맨 마지막 비트 값은 "0"이 되며, "1"인 경우 결과의 맨 마지막 비트 값은 $a_{m-1} \oplus p_0 \oplus a_{m-1}$="1⊕1⊕1=1"이 됨을 알 수 있다. 따라서 연산 2의 결과인 모듈러 리덕션 연산 시 맨 마지막 비트는 별도의 AND 게이트나 XOR 게이트가 필요하지 않고 단지 a_{m-1}의 값을 그대로 출력하면 된다.

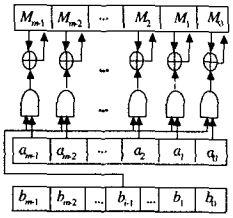
연산 1과 연산 2를 기본으로 하여, 알고리즘 1의 3번 문장을 수행하는 구조도는 다음 <그림 2>와 같다.



<그림 2> $A^{(i-1)}(x)x \bmod P(x)$ 연산을 위한 구조도

다음으로 $M^{(i)}(x)=M^{(i-1)}(x)+b_{i-1}A^{(i-1)}(x)$ $(1 \le i \le m)$ 을 수행하기 위하여, 먼저 $b_{i-1}A^{(i-1)}(x)$ 연산을 살펴본다.

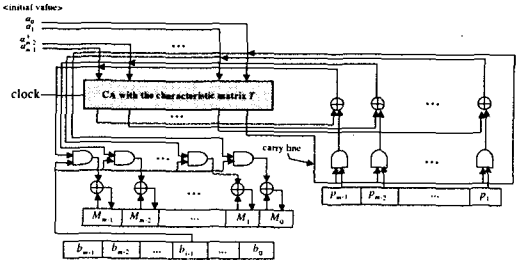
$b_{i-1}A^{(i-1)}(x)$ $(1 \le i \le m)$연산을 위하여 $A(x)$ 레지스터의 각 비트를 m개의 AND 게이트의 한 입력으로 하고, 나머지 하나의 입력은 b_{i-1}로 한다. 그 다음 그 결과와 $M^{(i-1)}(x)$를 XOR하여 그 결과를 다시 $M^{(i-1)}(x)$에 저장한다. 초기의 각 $M^{(i-1)}(x)(1 \le i \le m)$ 레지스터의 값은 "0"으로 초기화한다. 이를 위한 구조는 다음 <그림 3>과 같다. <그림 3>에서는 i번째 클럭에서의 연산을 나타내고 있다.



<그림 3> $M^{(i)}(x)=M^{(i-1)}(x)+b_{i-1}A^{(i-1)}(x)$ 연산을 위한 구조도

이제 식 (2)에서 두개의 식이 병렬로 처리될 수 있다고 하였으므로, <그림 2>, <그림 3>에서 제시된 구조도를 결합하면 모듈러 곱셈 연산을 수행 할 수 있는 <그림 4>와 같은 구조도가 생성된다. 여기서 <그림 3>에서의 $A(x)$ 레지스터는 CA가 된다. <그림 4>는 i번째 클럭에서의 연산을 나타내고 있으며, 첫 번째 연산에 앞서 특성행렬 T를 가지는 PBCA와 $B(x)$, $M(x)$, $P(x)$ $(0 \le i \le m-1)$ 레지스터에 초기 값이 설정된다.

<그림 4>에서 제시된 구조를 사용 시 m개의 셀과 $2m$-1 AND 게이트 그리고 $2m$-1 XOR 게이트, $3m$-1 레지스터를 사용하여 m 클럭 사이클만에 곱셈 연산을 수행할 수 있다.



- PBCA 초기값: $A(x) = a_{m-1} \cdots a_2\ a_1\ a_0$
- $B(x)$ 레지스터 초기값 : $b_{m-1} \cdots b_2\ b_1\ b_0$
- $P(x)$ 레지스터 초기값 : $p_{m-1} \cdots p_2\ p_1$
- $M(x)$레지스터 초기값 : all 0

<그림 4> CA를 이용하여 모듈러 곱셈 연산을 수행하는 구조도

<표 2> 여러 구조 상에서의 연구 결과 비교

구 조	시스톨릭 구조		CA 구조	
	[5]	[6]	[10]	본 논문
수행 연산	$A(x)B(x) \bmod P(x)$			
입출력 구조	serial-in serial- out		parallel-in parallel-out	
셀의 수	m	m	m	m
AND 게이트 수	$3m$	$3m$	$2m$	$2m$-1
XOR 게이트 수	$2m$	$2m$	$2m$	$2m$-1
래치 수	$12m$	$10m$	0	0
MUX 수	$2m$	$2m$	0	0
레지스터 수	0	0	$4m$	$3m$-1
셀 전파지연시간	$T_{2AND}+$ $T_{2XOR}+$ $T_{2MUX}+$ T_{1Latch}	$T_{2AND}+$ $2T_{2XOR}+$ $T_{2MUX}+$ T_{1Latch}	$T_{2AND}+$ $2T_{2XOR}+$ T_{1FF}	$T_{2AND}+$ $T_{2XOR}+$ T_{1FF}
수행 시간	$3m$	$3m$	m	m

5. 성능분석 및 시뮬레이션 결과

5.1 성능 분석

본 논문에서는 3-이웃 CA를 이용하여, GF(2^m)상에서 모듈러 곱셈 연산을 빠른 시간에 처리 할 수 있는 구조를 제안하였다.

본 논문의 결과를 지금까지의 연구결과와 비교해 보면 <표 2>와 같다.

본 논문에서 제시된 구조를 사용하면, m개의 셀과 $2m$-1개 AND 게이트와 $2m$-1개 XOR 게이트 그리고 $3m$-1 레지스터를 사용하여 m 클럭 사이클만에 곱셈 연산을 수행할 수 있다. 결과적으로 본 논문에서 제시한 구조가 시스톨릭 구조 [5], [6]과 비교시 시간적인 면에서나 공간적인 면에서 훨씬 효율적임을 알 수 있다.

[10]의 경우는 전체적인 곱셈의 구조는 제시하지 않

고 있으며, $A^{(i)}(x)=A^{(i-1)}(x)x \bmod P(x)(1\le i\le m)$의 연산에 사용되는 구조를 제시하고 이 때 사용된 게이트, 레지스터의 수를 분석하고 있다. 또한 본 논문에서는 <표 1>에서 나타난 것과 같이 CA를 이용하여 한 비트 쉬프트한 결과 생기는 캐리의 성질을 이용하여, AND 게이트와 XOR게이트, one-bit 레지스터 각각 1개씩을 줄일 수 있었다. 그러나 [10]의 경우는 캐리를 이용하기 위해 CA 외에 $A(x)$를 저장하는 별도의 레지스터를 두고 있으며, CA에서 순환 시프트한 결과 캐리가 생기지 않고 항상 "0"이 되도록 특성 행렬을 둠으로써, AND 게이트, XOR 게이트가 하나씩 더 필요하게 된다. 또한 곱셈을 위한 나머지 연산($M^{(i)}(x)= M^{(i-1)}(x) + b_{i-1}A^{(i-1)}(x)(1\le i\le m)$)에 대해서는 구조도가 제시되어 있지 않아 이를 수행하기 위해서는 추가의 게이트와 레지스터가 필요하게 되어 전체 m 셀, $2m$개의 AND 게이트, $2m$개의 XOR 게이트, $4m$개 레지스터가 필요하다. 그러므로, 본 논문에서 제시한 구조와 비교해 볼 때 시간적인 면에서는 동일하나 공간적인 면에서 비효율적임을 알 수 있다.

<표 3>은 제시된 구조와 기존의 시스톨릭 구조 및 CA에 대한 AT(Area and Time) 성능 분석이다. AT 성능분석[12]은 구현에 사용되는 트랜지스터의 수와 각 게이트 연산 시 소요되는 시간을 기준으로 하며, 그 단위는 각각 Φ와△로 나타낸다. T_{xGATE}는 x개의 입력을 가지는 게이트의 지연시간을 나타내며, A_{xGATE}는 x개의 입력을 가지는 게이트의 트랜지스터 수를 나타낸다. 본 논문에서는 1개의 레지스터와 CA의 하나의 셀은 1개의 플립 플랍으로 분석하였다. 각각의 게이트는 다음의 값을 가진다[12]

T_{2AND} = 2.4△ A_{2AND} = 6Φ
T_{2XOR} = 4.2△ A_{2XOR} = 14Φ
T_{2MUX} = 5.8△ A_{2MUX} = 20Φ
T_{1FF} = 3.8△ A_{1FF} = 18Φ
T_{1Latch} = 1.4△ A_{1Latch} = 8Φ

<표 3>AT 분석을 통한 곱셈기 연구결과 비교

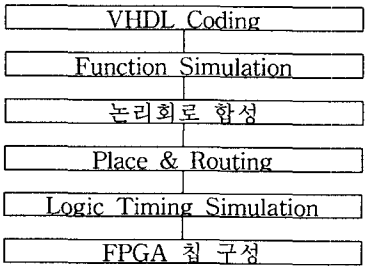
	Time	Area	Area× Time
[5]	$3m(T_{2AND}+T_{2XOR}+T_{2MUX}+T_{1Latch})=41.4m$△	$3mA_{2AND}+2mA_{2XOR}+2mA_{2MUX}+12mA1$ Latch=$182m$Φ	$7534.8m^2$ Φ△
[6]	$3m(T_{2AND}+2T_{2XOR}+T_{2MUX}+T_{1Latch})=54m$△	$3mA_{2AND}+2mA_{2XOR}+2mA_{2MUX}+10mA1$ Latch=$166m$Φ	$8964m^2$ Φ△
[10]	$m(T_{2AND}+2T_{2XOR}+T_{1FF})=14.6m$△	$2mA_{2AND}+2mA_{2XOR}+5mA_{1FF}=130m$Φ	$1898m^2$ Φ△
본 논문	$m(T_{2AND}+T_{2XOR}+T_{1FF})=10.4m$△	$2mA_{2AND}+(2m-1)A_{2XOR}+(4m-1)A_{1FF}=(112m-32)$Φ	$(1164.8m^2-332.8m)$ Φ△

가지는 게이트의 트랜지스터 수를 나타낸다. 본 논문에서는 1개의 레지스터와 CA의 하나의 셀은 1개의 플립 플랍으로 분석하였다. 각각의 게이트는 다음의 값을 가진다[12]

<표 3>에서 보는 바와 같이 본 논문에서 제시한 곱셈기의 구조가 AT 분석 결과에서도 다른 구조보다 우수함을 알 수 있다.

5.2 시뮬레이션 결과

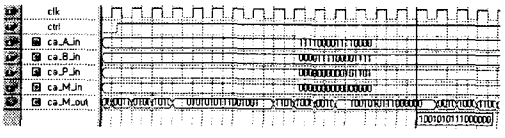
본 절에서는 유한 필드 GF(2^m)상에서 제시된 모듈러 곱셈기가 올바르게 동작됨을 보이기 위해 VHDL로 기술하여 이를 Altera MAX+II simulation tool을 써서 function 시뮬레이션하여 결과 값을 확인하였으며, Synopsys사의 FPGA-Epress를 사용하여 <그림 5>와 같은 절차로 제안된 곱셈기의 기능을 검증하였다. 이 때 Target device는 FLEX10K100ARC240-3를 사용하였다.



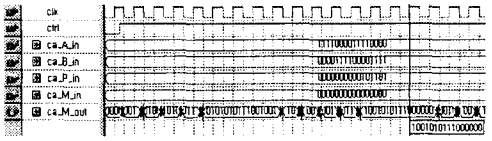
<그림 5> 시뮬레이션 수행 절차

그 결과 <그림 6>과 같은 결과를 얻음으로써, 본 논문에서 제시한 구조가 m클럭 사이클만에 모듈러 곱셈을 올바르게 수행함을 알 수 있었다.

(a) <그림 4>의 회로 합성 결과



(b) <그림 4>의 Function 시뮬레이션 결과



(c)<그림 4>의 Timing 시뮬레이션 결과

입력값			출력값	
	$A(x)$	1111000011110000		M(x)_out
	$B(x)$	0000111100001111		1001010111000000
	$P(x)$	0000000000101101		
	$M(x)$	0000000000000000		

(d) 16 bit에서 테스트 결과

<그림 6> $GF(2^{16})$상에서의 시뮬레이션 결과

위의 <그림 6>의 (b)와 (c)에서 ca_A_in, ca_B_in, ca_P_in, ca_M_in은 각각 $A(x)$, $B(x)$, $P(x)$, $M(x)$ 레지스터의 입력 값을 나타내며, ca_M_out은 $A(x)B(x)$ mod $P(x)$의 결과 값을 나타낸다. 본 논문에서는 시뮬레이션 시 16비트 데이터를 이용하였다. 따라서 (b)와 (c)를 통해 16 클럭 사이클 후에 올바른 결과 값이 출력됨을 알 수 있으며, (d)에서는 이들의 입출력 값을 보여주고 있다.

6. 결 론

본 논문에서는 CA를 이용하여, $GF(2^m)$상에서 모듈러 곱셈 연산을 빠른 시간에 처리 할 수 있는 구조를 제안하였다. $GF(2^m)$상에서 모듈러 곱셈 연산은 지수승 연산을 수행하기 위해 반드시 필요한 기본적인 연산이며, 지수승 연산은 Diffie-Hellman key exchange, ElGamal과 같은 대부분의 공개키 시스템의 기본이 되는 연산이다.

본 논문에서 제시한 곱셈기 구조는 [12]에서 제시한 AT 분석법을 통해 분석 해 본 바와 같이 그 결과가 $(1164.8m^2-332.8m)\Phi\Delta$로써, 시스톨릭 구조의 결과인 $7534.8m^2\Phi\Delta$[5]와 $8964m^2\Phi\Delta$[6]보다 효율적이며, CA 구조의 결과인$1898m^2\Phi\Delta$[10]보다도 역시 효율적인 구조임을 알 수 있다.

따라서 본 논문에서 제시된 곱셈기를 지수기 설계에 활용한다면 효율적인 지수기를 설계 할 수 있다. 또한 본 논문에서 제시된 모듈러 곱셈 연산기를 기반으로 하면 모듈러 지수기 뿐만 아니라, 나눗셈기, 곱셈의 역원기 등을 효율적으로 구현할 수 있다.

앞으로의 연구과제는 본 논문에서 설계된 곱셈기를 바탕으로, 공개키 암호 시스템을 위한 지수기를 설계하고, 이를 이용하여 효율적인 공개키 암호시스템을 구축하는 것이다.

참 고 문 헌

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

[2] W. Diffie and M.E. Hellman, New directions in cryptography, *IEEE Trans. on Info. Theory*, vol. 22, pp.644-654, November 1976.

[3] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. on Info. Theory*, vol. 31(4). pp. 469-472, July 1985.

[4] A.J. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.

[5] C. S. YEH, IRVING S. REED, T.K. TRUONG, Systolic Multipliers for Finite Fields $GF(2^m)$, *IEEE Trans. on Computers*, vol. C-33, no. 4, pp. 357-360, April 1984.

[6] Chin-Liang Wang, Jung-Lung Lin, Systolic Array Implementation of Multipliers for Finite Fields $GF(2^m)$, *IEEE Trans. on Circuits and Systems*, vol. 38, no. 7, pp. 796-800, July 1991.

[7] P.L. Montgomery, Modular multiplication without trial division, *Mathematics of Computation*, 44(170):519-521, April 1985.

[8] M. Delorme, J. Mazoyer, *Cellular Automata*, KLUWER ACADEMIC PUBLISHERS 1999.

[9] STEPHEN WOLFRAM, *Cellular Automata and Complexity*, Addison -Wesly Publishing Company, 1994.

[10] P. P. Choudhury, R. Barua, Cellular Automata Based VLSI Architecture for Computing Multiplication And Inverse In $GF(2^m)$, *IEEE Proceedings of the 7th International Conference on VLSI Design*, pp.279-282, January 1994.

[11] C.S. Yeh, I.S. Reed, and T.K. Truong, Systolic Multipliers for Finite Fields $GF(2^m)$, *IEEE Trans. on Computers*, vol. C-33, no. 4, pp. 357-360, 1984.

[12] D. D. Gajski, *Principles of Digital Design*, Prentice-Hall International, Inc., 1997.

하경주

경북대학교 공과대학 컴퓨터공학과 졸업(공학사)

경북대학교 대학원 공과대학 컴퓨터공학과 졸업(공학석사)

경북대학교 대학원 공과대학

컴퓨터공학과 졸업(공학박사)
한국전자통신연구소 선임연구원
현재 경산대학교 정보과학부 조교수
<관심분야 > 정보보호, 암호화시스템, 병렬처리

구교민
경북대학교 공과대학 컴퓨터공학과 졸업(공학사)
경북대학교 대학원 공과대학 컴퓨터공학과 졸업(공학석사)
경북대학교 대학원 공과대학 컴퓨터공학과 졸업(공학박사)
한국 과학기술정보연구원 연구원
현재 대구교육대학교 전임강사
<관심분야 > 정보보호, 암호화시스템, 병렬처리