공간 효율적인 비트-시리얼 제곱/곱셈기 및 AB^2-곱셈기

(Area Efficient Bit-serial Squarer/Multiplier and AB^2-Multiplier)

이 원 호 † 유 기 영 ††
(Won-Ho Lee) (Kee-Young Yoo)

요 약 현대 통신 분야에서 많이 응용되고 있는 유한 필드상의 중요한 연산은 지수승과 나눗셈, 역원 등이 있다. 유한 필드에서 지수 연산은 이진 방법을 이용하여 곱셈과 제곱을 반복함으로서 구현될 수 있고, 나눗셈이나 역원 연산은 AB^2 연산을 반복함으로서 구현될 수 있다. 그래서 이러한 연산들을 위한 빠른 알고리즘과 효율적인 하드웨어 구조 개발이 중요하다. 본 논문에서는 차수가 m인 기약 AOP에 의해 생성되는 GF(2^m)상의 제곱과 곱셈을 동시에 할 수 있는 새로운 구조의 비트-시리얼 제곱/곱셈기와 AB^2-곱셈기를 구현하였다. 제안된 연산기들은 지수기와 나눗셈 및 역원기의 핵심 회로로 사용될 수 있으며 기존의 연산기들과 비교하여 보다 작은 하드웨어 복잡도를 가진다. 그리고 제안된 구조는 정규성과 모듈성을 가지기 때문에 VLSI 칩과 같은 하드웨어로 쉽게 구현함으로써 IC 카드에 이용될 수 있다.

키워드 : 암호학, 유한 필드, 기약 AOP, 지수 연산, 곱셈, 나눗셈

Abstract The important arithmetic operations over finite fields include exponentiation, division, and inversion. An exponentiation operation can be implemented using a series of squaring and multiplication operations using a binary method, while division and inversion can be performed by the iterative application of an AB^2 operation. Hence, it is important to develop a fast algorithm and efficient hardware for this operations. In this paper presents new bit-serial architectures for the simultaneous computation of multiplication and squaring operations, and the computation of an AB^2 operation over GF(2^m) generated by an irreducible AOP of degree m. The proposed architectures offer a significant improvement in reducing the hardware complexity compared with previous architectures, and can also be used as a kernel circuit for exponentiation, division, and inversion architectures. Furthermore, since the proposed architectures include regularity and modularity, they can be easily designed on VLSI hardware and used in IC cards.

Key words : Cryptography, Finite Fields, Irreducible AOP, Exponentiation, Multiplication, Division

1. 서 론

현대 통신 분야에서 유한 필드(Finite Fields) 또는 갈로아 필드(Galois Fields: GF)는 에러 교정 코드(Error Correcting Code)[1]나 암호학(Cryptography)[2], 디지털 신호 처리(Digital Signal Processing)[3] 같은 곳에 많이 활용되고 있다. 이러한 유한 필드에서 중요한 연산은 덧셈(Addition), 곱셈(Multiplication), 지수승(Exponentation), 나눗셈(Division), 곱셈에 대한 역원(Multiplicative Inverse) 연산들이다. 이들 연산들 중에서 덧셈은 필드 원소들이 다항식 표현(Polynomial Representation) 방식으로 표현된다면 배타적 논리합(eXclusive OR: XOR)으로 매우 간단한 연산이다. 그러나 다른 연산들은 매우 복잡한 연산들이다. 지수 연산은 유한 필드 GF(2^m)상에서 이진 방법을 사용하여 제곱과 곱셈을 반복함으로서 구현할 수 있다[4]. 그리고 나눗셈이나 곱셈에 대한 역원 연산은 AB^2 연산을 반복함으로서 구현할 수 있다. 유한 필드 GF(2^m)상의 AB^2 연산을 위한 VLSI 구조는 두개의 곱셈기를 사용하여 쉽게 구현할 수 있다. 그러나 하드웨어 구현 측면에서 볼 때 두개의 곱셈기를 사용하는 것 보다 AB^2-곱셈기를 사용하

· 이 논문은 2003년도 두뇌한국21사업에 의하여 지원되었음
† 비 회 원 : 경북대학교 컴퓨터공학과
purmi@purple.knu.ac.kr
†† 종신회원 : 경북대학교 컴퓨터공학과 교수
yook@knu.ac.kr
논문접수 : 2003년 3월 27일
심사완료 : 2003년 9월 25일

는 것이 매우 좋다. 그래서 제곱과 곱셈, AB^2 연산을 위한 효율적이고 빠른 알고리즘과 하드웨어 개발이 매우 중요하므로 본 논문에서는 유한 필드 GF(2^m)상의 지수 연산과 나눗셈 및 역원 연산을 위해 작은 복잡도와 단일화된 구조를 가지는 제곱과 곱셈을 동시에 하는 제곱/곱셈기와 AB^2-곱셈기를 설계한다.

유한 필드 GF(2^m)상에서 빠른 연산과 적은 하드웨어 복잡도를 가지는 다양한 구조(Architecture)는 많이 연구 되었다[5-13]. 그러한 구조들은 서로 다른 기저를 사용하였는데 그 기저에는 정규 기저(Normal Basis)와 이중 기저(Dual Basis), 표준 기저(Standard Basis)가 있으며 각기 다른 특징들을 가진다. 이중에서 표준 기저의 구조는 적은 복잡도와 모듈성을 가지므로 본 논문에서는 표준 기저를 사용한다[5]. 또한, 최근 들어 유한 필드 GF(2^m)상의 연산에서 기약 다항식(Irreducible Polynomial)의 모든 항이 1인 기약 AOP(All-One Polynomial)를 사용하여 구현한 비트-페러럴(Bit-Parallel)과 비트-시리얼(Bit-Serial) 구조의 하드웨어가 많이 연구되고 있다[14-16]. 그래서 본 논문에서는 유한 필드 GF(2^m)상에서 기약 AOP를 사용하여 지수 연산을 위한 곱셈과 제곱을 동시에 하는 연산기와 나눗셈 및 역원 연산을 위한 AB^2-곱셈기를 설계한다. 제안된 구조는 이전의 구조와 비교했을 때 하드웨어 복잡도가 작으며, 정규성과 모듈성을 가지기 때문에 VLSI 칩과 같은 하드웨어로 쉽게 구현함으로써 IC 카드에 이용될 수 있다.

서론에 이어 본 논문의 구성은 다음과 같다. 2장에서는 유한 필드 GF(2^m)상의 기본 연산에 대하여 간단히 살펴본다. 3장에서는 곱셈과 제곱 연산을 동시에 하는 알고리즘과 AB^2 연산을 위한 알고리즘을 설명하고 이 알고리즘들에서 순환 방정식을 유도하여 지수 연산을 위한 곱셈과 제곱을 동시에 하는 제곱/곱셈기와 나눗셈 및 역원 연산을 위한 AB^2-곱셈기를 새로운 하드웨어 구조로 제시한다. 4장에서는 제안된 연산기들을 시뮬레이션을 하여 검증하고, 이를 이용하여 지수기와 나눗셈 및 역원기를 설계하여 기존의 연산기들과 비교 분석을 한다. 마지막으로 5장에서 결론을 맺는다.

2. 유한 필드 GF(2^m)상의 연산

유한 필드 GF(2^m)상의 원소는 여러 가지 방식으로 표기될 수 있다. 그 표기법들 중에서 다항식 표기법은 일반적으로 가장 많이 쓰이고 하드웨어와 소프트웨어 구현에 적당하다. 그래서 본 논문에서는 이 다항식 표기법을 사용하기로 한다.

만약 $f(x)=x^m+x^{m-1}+\cdots+x+1$을 모든 각각의 항이 1인 기약 AOP라 가정하고, α를 $f(x)$의 근(Root)이라고 가정하자. 만약 2가 오더(Order) m을 모듈러 (m+1)을 했을 때의 원소이고 (m+1)이 소수(Prime)이면 $f(x)$는 GF(2^m)을 위한 기약 AOP로 선택할 수 있다[14, 16]. 그러면 GF(2^m)을 위한 다항식 기저(Polynomial Basis)의 폼은 {1, α, $\cdots$, α^{m-1}}이고 GF(2^m)상의 원소인 A는 수식 (1)과 같이 표현된다.

$$\begin{aligned} A &= \sum_{i=0}^{m-1} A_i \alpha^i \\ &= A_{m-1}\alpha^{m-1} + A_{m-2}\alpha^{m-2} + \cdots + A_1\alpha^1 + A_0 \end{aligned} \quad (1)$$

여기서 A_i는 GF(2)의 원소이다. 기약 AOP는 $f(\alpha)=\alpha^m+\alpha^{m-1}+\cdots+\alpha+1$이 된다. 만약 {1, α, $\cdots$, α^{m-1}, α^m}의 폼을 확장 다항식 기저(Extended Polynomial Basis)라 하면 필드 원소 A는 수식 (2)와 같이 표현될 수 있다. 수식 (2)에서 a_i는 GF(2)의 원소이다.

$$A = \sum_{i=0}^{m} a_i \alpha^i = a_m\alpha^m + a_{m-1}\alpha^{m-1} + \cdots + a_1\alpha^1 + a_0 \quad (2)$$

다항식 기저와 확장 다항식 기저의 각 항인 A_i와 a_i의 관계는 수식 (3)과 같다. 다항식 기저의 각 항(A_i)은 확장 다항식 기저의 최고차항(a_m)과 나머지 각 항(a_i)과 더한 것과 같다. 그리고 확장 다항식 기저 상에서는 기약 다항식은 $f(\alpha)=\alpha^{m+1}+1$이 된다[14,16].

$$A_i = a_i + a_m \qquad (0 \le i < m\text{-}1) \quad (3)$$

유한 필드 원소의 제곱 연산인 A^2은 수식 (4)와 같이 표현된다. 수식 (4)에서 보는 것과 같이 확장 다항식 기저 상에서의 제곱 연산(A^2)은 단지 계수들을 규칙에 맞게 재 나열을 하는 것과 동일하다. 이는 제곱 연산에서 추가적인 연산이 필요 없이 단지 각 항의 계수들을 재 나열만 하면 된다는 것이다[14,16].

$$\begin{aligned} & A^2 \pmod{\alpha^{m+1}+1} \\ &= (a_m\alpha^{2m} + a_{m-1}\alpha^{2m-2} + \\ &\qquad \cdots + a_0) \pmod{\alpha^{m+1}+1} \\ &= a_m\alpha^{m-1} + a_{m-1}\alpha^{m-3} + \\ &\qquad \cdots + a_{m/2+1}\alpha^1 + a_{m/2}\alpha^m + a_{m/2-1}\alpha^{m-2} + \\ &\qquad \cdots + a_0 \\ &= a_{m/2}\alpha^m + a_m\alpha^{m-1} + a_{m/2-1}\alpha^{m-2} + \\ &\qquad \cdots + a_1\alpha^2 + a_{m/2+1}\alpha^1 + a_0 \end{aligned} \quad (4)$$

A와 B, P를 유한 필드 GF(2^m)상의 원소라 하자. 즉, $A, B, P \in$ GF(2^m). 그러면 A와 B, P를 확장 다항식 기저 상에서 표현을 하면 $A=\sum_{i=0}^{m} a_i\alpha^i$, $B=\sum_{i=0}^{m} b_i\alpha^i$, $P=\sum_{i=0}^{m} p_i\alpha^i$와 같다. 이때 곱셈 연산은 $P=AB \pmod{\alpha^{m+1}+1}$로 나타낼 수 있다. 곱셈의 결과 P 또한 유한 필드 GF(2^m)상의 원소가 되고, 각 항은 수식 (5)와 같이 나타낼 수 있다.

수식 (5)의 관계를 바탕으로 유한 필드 GF(2^m)상의 원소인 A와 B의 곱셈은 수식 (6)과 같이 행렬의 곱으로 나타낼 수 있다. 수식 (6)에서 보는 것과 같이 곱셈의 결

과 값인 각 항 p_i는 승수(Multiplier)의 각 항 b_i를 한 비트씩 좌측으로 순환 시프트(One-Bit Left Circular Shift)를 하여 승수의 각 항을 피승수(Multiplicand)의 각 항 a_i와 곱한 후 서로 더하여 구하면 된다.

$$\begin{cases} p_m = a_0 b_m + a_1 b_{m-1} + a_2 b_{m-2} + \\ \quad \cdots + a_{m-2} b_2 + a_{m-1} b_1 + a_m b_0 \\ p_{m-1} = a_m b_m + a_0 b_{m-1} + a_1 b_{m-2} + \\ \quad \cdots + a_{m-3} b_2 + a_{m-2} b_1 + a_{m-1} b_0 \\ \quad \vdots \\ p_1 = a_2 b_m + a_3 b_{m-1} + a_4 b_{m-2} + \\ \quad \cdots + a_m b_2 + a_0 b_1 + a_1 b_0 \\ p_0 = a_1 b_m + a_2 b_{m-1} + a_3 b_{m-2} + \\ \quad \cdots + a_{m-1} b_2 + a_m b_1 + a_0 b_0 \end{cases} \quad (5)$$

$$\begin{bmatrix} p_m \\ p_{m-1} \\ \vdots \\ p_1 \\ p_0 \end{bmatrix} = \begin{bmatrix} b_0 & b_1 & \cdots & b_m \\ b_m & b_0 & \cdots & b_{m-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_2 & b_3 & \cdots & b_1 \\ b_1 & b_2 & \cdots & b_0 \end{bmatrix} \begin{bmatrix} a_m \\ a_{m-1} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} \quad (6)$$

유한 필드 GF(2^m)상의 지수 연산은 곱셈과 제곱 연산을 반복적으로 적용하는 이진 제곱과 곱셈 알고리즘(Binary Square and Multiply Algorithm)을 사용하여 구현될 수 있다[4]. 이 알고리즘은 지수 연산 A^E을 계산할 때 지수 E를 $E = \sum_{i=0}^{n-1} e_i 2^i$로 표현을 하여 각 지수 비트에 따라 지수 연산을 수행한다. 단, $e_i \in 0, 1$이다. 그리고 이 알고리즘은 RL방식(Right-to-Left Method)과 LR방식(Left-to-Right Method)의 두 가지 방식이 있다. 이 두 방식들 중에서 RL방식은 곱셈과 제곱을 할 때 서로 의존관계가 없이 독립적으로 하기 때문에 두 연산은 동시에 수행을 할 수 있다. 그래서 본 논문에서는 이 RL방식을 사용하여 지수기를 설계한다. RL방식을 이용한 이진 제곱과 곱셈의 알고리즘은 알고리즘 1에 나타나 있다.

[알고리즘 1] RL방식의 지수 연산 알고리즘
입력 : A, E 출력 : $C = A^E$
1. $C = 1$
2. **for** $i = 0$ **to** $n - 1$ **do**
3. **if** $e_i = 1$ **then** $C = CA$
4. $A = A^2$

나눗셈은 곱셈과 곱셈에 대한 역원을 사용하여 할 수 있다. 즉 $A/B = AB^{-1}$이기 때문이다. 역원을 구하는 방법에는 테이블 룩업 방법(Table Lookup Method)과 유클리드 알고리즘(Eucluid's Algorithm)을 이용하는 방법 등이 있다. 테이블을 이용한 방법은 m의 값이 증가함에 따라 테이블 사이즈가 너무 커지는 단점이 있고, 유클리드 알고리즘을 이용한 방식은 VLSI로 구현하기가 쉽지가 않다. 역원 연산은 지수 연산의 특별한 경우라 할 수 있다. 왜냐하면 페르마 이론(Fermat Theorem)에 의해역원 연산 $B^{-1} = B^{2^m-2} = (B(B(B\cdots B(B(B)^2)^2)^2)^2)^2$이기 때문이다. 이러한 특징을 이용하여 나눗셈이나 역원을 구하는 알고리즘은 알고리즘 2에 나타나 있다. 이 알고리즘에서 AB^2 연산이 단계 4에서 사용되고 있으며, 만약 A의 값이 '1'이면 결과 값은 역원이 된다.

[알고리즘 2] 나눗셈 및 역원 연산 알고리즘
입력 : A, B 출력 : $R = A/B = AB^{-1}$
1. $R = B$
2. **for** $i = 1$ **to** $m-1$ **do**
3. **if** $i = m-1$ **then** $B = A$
4. $R = BR^2$

3. 비트-시리얼 연산기 설계

지수 연산이나 나눗셈 및 곱셈에 대한 역원을 구하기 위해서는 제곱과 곱셈을 위한 곱셈기와 AB^2-곱셈기가 필요하다. 이러한 연산기들은 비트-패러럴과 비트-시리얼의 구조가 있는데 하드웨어 복잡도를 우선으로 하기 위해 비트-시리얼 구조를 선택한다. 그래서 본 절에서는 이러한 연산기들을 위해 연산 알고리즘에서 순환 방정식을 유도하여 단일화된 새로운 비트-시리얼 구조의 회로를 설계한다.

3.1 단일 구조의 비트-시리얼 제곱/곱셈기 설계

지수 연산을 위하여 이진 제곱과 곱셈 알고리즘을 구현하는 방법으로 하나의 곱셈기를 이용하여 제곱과 곱셈을 하는 방식과 아니면 제곱과 곱셈을 위해 두개의 곱셈기를 사용하는 방식이 있을 수 있다. 그러나 이러한 방법들은 시간 복잡도 또는 하드웨어 복잡도를 증가시킨다. 그러므로 이 절의 주 목적은 이진 제곱과 곱셈 알고리즘을 사용하여 지수 연산을 할 경우 요구되는 제곱과 곱셈의 연산을 동시에 하는 새로운 단일화된 구조의 회로를 설계하여 하드웨어 복잡도를 줄이는 것이다.

앞 장에서 제시한 알고리즘 1을 이용해 지수 연산을 할 때 필요한 단일화된 비트-시리얼 구조를 위한 제곱과 곱셈의 알고리즘은 수식 (4)와 (5)의 활용과 레지스터 및 시프트 연산을 사용하여 알고리즘 3과 같이 나타낼 수 있다. 이 알고리즘에서 레지스터와 시프트 연산을 사용하여 연산하기 때문에 크게 두 과정이 필요하다. 하나는 레지스터에 값을 저장하기위한 과정(단계 2~4)과 연산을 위한 과정(단계 5~8)이다. 그리고 수식 (4)에서 보는 것과 같이 제곱 연산은 단지 각항의 계수들을 재배치하는 연산이므로 첫 과정의 마지막 단계에서 수행이 가능하다. 그래서 알고리즘 3의 단계 4에서 수식 (4)를 이용하여 제곱을 하고, 단계 6에서 수식 (5)를 이용하여 곱셈을 한다. 알고리즘 3에서 C_L_Shift(X)는 X를 한 비트 좌측으로 순환 시프트를 하는 연산을 나타

낸다. 그리고 a_i와 b_i, p_i, s_i는 각각 A와 B, P, S의 i-번째 항을 나타낸다. 첫 번째 순환문(단계 2~4)에서 변수 A와 B는 레지스터 Y와 X에 저장되고, 루프의 마지막에 수식 (4)를 이용한 제곱 연산이 일어나 레지스터 Z에 저장된다. 두 번째 순환문(단계 5~8)에서 매 순환마다 수식 (5)를 이용한 곱셈의 결과 p_i와 제곱의 결과인 s_i가 한 비트씩 출력이 된다.

[알고리즘 3] 단일 구조를 위한 비트-시리얼 제곱과 곱셈 알고리즘
입력 : A, B 출력 : $P = AB$, $S = A^2$

```
1. P = S = 0
2. for i= m downto 0 do
3.     y_i = a_i, x_i = b_i
4.     if i = 0 then Z = Y^2  /* 수식 (4) 사용 */
5. for i= m downto 0 do
6.     p_i = YX  /* 수식 (5) 사용 */
7.     s_i = z_i
8.     C_L_Shift(X)
```

알고리즘 3에서 제곱 연산시 각 항의 계수들의 재배치 속성과 수식 (6)을 이용하여 단일 구조의 비트-시리얼 제곱/곱셈기를 위한 순환 방정식을 유도할 수 있으며 알고리즘 4에 나타나 있다. 알고리즘 4에서 L_Shift(X, b_i)는 b_i비트를 오른쪽 끝에 입력으로 하여 X를 한 비트 좌측으로 시프트(One-Bit Left Shift)를 하는 연산을 나타낸다.

알고리즘 4의 순환 방정식을 이용하여 유한 필드 GF(2^m)상의 단일 구조의 비트-시리얼 제곱/곱셈기는 그림 1과 같이 구현할 수 있다. 여기서 m=4이다. 그림 1에서 AND-XOR 트리는 그림 2에 나타나 있다. y_i 레지스터를 기준으로 위쪽은 제곱 연산을 아래쪽은 곱셈 연산을 수행한다. 처음 (m+1)번의 지연동안은 A와 B의 값이 y_i와 x_i의 레지스터에 각각 저장이 되고, (m+1)번째 단계에서 제곱 연산이 수행되어 z_i 레지스터에 저장이 된다. (m+1)번째의 지연이 있는 후 (m+2)번째부터 제곱과 곱셈의 결과가 하나씩 나와 (2m+2)번째에 마지막 결과가 나온다. 입력 값 A와 B를 레지스터에 저장하고 제곱과 곱셈 연산을 위해 제어신호(ctl)가 하나 필요한데 (m+1)개의 '1'과 (m+1)개의 '0'으로 구성되어 진다. 즉 ($1^{m+1}0^{m+1}$)이다. 제어신호가 '1'일 동안 A와 B의 값이 레지스터에 시프트 되면서 저장이 되고, 마지막 (m+1)번째에 제곱연산이 수행되어 진다. 나머지 '0'일 동안에 곱셈이 수행되어 매번마다 제곱과 곱셈의 결과 한 비트씩 값이 출력된다.

[알고리즘 4] 단일 구조의 비트-시리얼 제곱/곱셈기를 위한 순환 방정식
입력 : A, B 출력 : $P = AB$, $S = A^2$

```
1.  P = S = 0
2.  for i = m downto 0 do
3.      if i = 0 then  /* 단계 3~9는 동시수행 */
4.          z_0 = a_0
5.          for j = 1 to m do
6.              if j <= (m/2) then  z_2j = y_j-1
7.              else z_2(j-m/2)-1 = y_j-1
8.      L_Shift(Y, a_i)
9.      L_Shift(X, b_i)
10. for i = m downto 0 do
11.     for j = 0 to m do  /* 단계 11~14는 동시수행 */
12.         p_i = x_j × y_m-j + p_i
13.     s_i = z_i
14.     C_L_Shift(X)
```

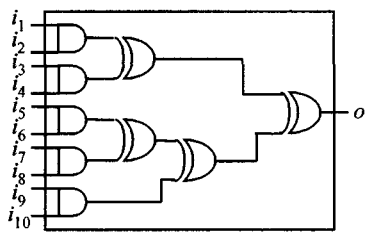


그림 2 그림 1의 AND-XOR 트리

3.2 단일 구조의 비트-시리얼 AB^2-곱셈기 설계

나눗셈 연산은 $A/B = AB^{-1}$이기 때문에 곱셈에 대한

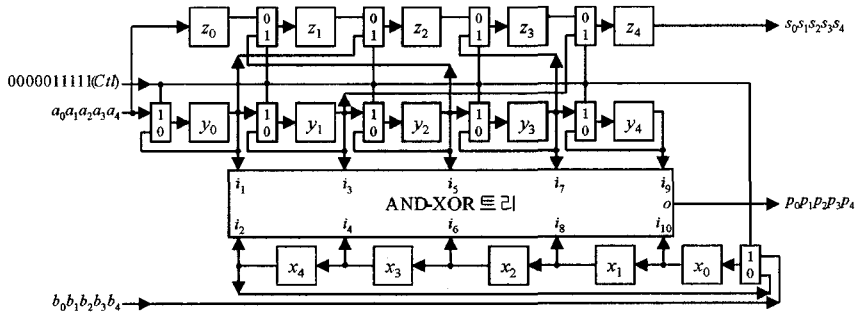


그림 1 GF(2^m)상의 비트-시리얼 제곱/곱셈기(m=4)

역원을 이용할 수 있다. 역원을 구하기 위해서는 앞 장에 제시한 알고리즘 2를 이용하면 된다. 알고리즘 2에서는 AB^2 연산이 필요하므로 이 절에서는 이 연산을 수행할 수 있는 작은 하드웨어 복잡도의 단일 구조 비트-시리얼 AB^2-곱셈기를 설계한다.

앞 절에서 단일화된 비트-시리얼 구조를 위한 제곱과 곱셈의 알고리즘을 유도한 방식과 같이 앞에서 제시한 알고리즘 2를 이용해 나눗셈 연산을 할 때 필요한 단일화된 비트-시리얼 구조를 위한 AB^2 연산의 알고리즘은 수식 (4)와 (5)의 활용과 레지스터 및 시프트 연산을 사용하여 알고리즘 5와 같이 나타낼 수 있다. 이 알고리즘에서도 레지스터와 시프트 연산을 사용하여 연산하기 때문에 레지스터에 값을 저장하기 위한 과정(단계 2~4)과 연산을 위한 과정(단계 5~7)이 필요하다.

[알고리즘 5] 단일 구조를 위한 비트-시리얼 AB^2 알고리즘
입력 : A, B 출력 : $P = AB^2$

1. $P = 0$
2. **for** $i = m$ **downto** 0 **do**
3. $y_i = a_i$, $x_i = b_i$
4. **if** $i = 0$ **then** $X = X^2$ /* 수식 (4) 사용 */
5. **for** $i = m$ **downto** 0 **do**
6. $p_i = YX$ /* 수식 (5) 사용 */
7. C_L_Shift(X)

알고리즘 6은 알고리즘 5에서 제곱 연산의 재배치 속성과 수식 (6)을 이용하여 단일 구조의 비트-시리얼 AB^2-곱셈기를 위한 순환 방정식을 유도한 것이다.

알고리즘 6의 순환 방정식을 이용하여 유한 필드 GF(2^m)상의 단일화된 비트-시리얼 AB^2-곱셈기는 그림 3과 같이 구현할 수 있다. 여기서 m=4이다. 처음 (m+1)번의 지연동안은 A와 B의 값이 y_i와 x_i의 레지스터에 시프트 되면서 저장이 되고, (m+1)번째 단계에서 제곱 연산이 수행되어 x_i 레지스터에 저장이 된다. (m+1)번째의 지연이 있는 후 (m+2)번째부터 AB^2 연산의 결과가 하나씩 나와 (2m+2)번째에 마지막 결과가 나온다. 입력 값 A와 B를 레지스터에 저장하고 곱셈 연산을 위해 제어신호($ctl1$)가 필요한데 (m+1)개의 '1'과 (m+1)개의 '0'으로 구성되어진다. 즉 $(1^{m+1}0^{m+1})$이다. 제어신호($ctl1$)가 '1'일 동안 A와 B의 값이 레지스터에 저장이 되고, 나머지 '0'일 동안에 곱셈이 수행되어 매번마다 곱셈 결과가 한 비트씩 출력된다. 그리고 제곱 연산을 위해 제어신호($ctl2$)가 하나 더 필요한데 m개의 '0'과 1개의 '1', (m+1)개의 '0'으로 구성되어 진다. 즉 $(0^m1^10^{m+1})$이다. 제어신호($ctl2$)가 '1'일 때, 즉 (m+1)번째에 제곱 연산이 일어나고, '0'일 때는 x_i 레지스터의 값이 한 비트 좌측으로 순환 시프트를 한다.

[알고리즘 6] 단일 구조의 비트-시리얼 AB^2-곱셈기를 위한 순환 방정식
입력 : A, B 출력 : $P = AB^2$

1. $P = 0$
2. **for** $i = m$ **downto** 0 **do**
3. **if** $i = 0$ **then** /* 단계 3~9는 동시수행 */
4. $x_0 = b_0$
5. **for** $j = 1$ **to** m **do**
6. **if** $j <= (m/2)$ **then** $x_{2j} = x_{j-1}$
7. **else** $x_{2(j-m/2)-1} = x_{j-1}$
8. L_Shift(Y, a_i)
9. L_Shift(X, b_i)
10. **for** $i = m$ **downto** 0 **do**
11. **for** $j = 0$ **to** m **do** /* 단계 11~13은 동시수행 */
12. $p_i = x_j \times y_{m-j} + p_i$
13. C_L_Shift(X)

4. 제안된 구조의 검증과 응용 및 분석

이 장에서는 앞 장에서 설계한 연산기들을 시뮬레이션을 하여 검증하고, 제안한 연산기들을 이용하여 지수기와 나눗셈 및 역원 연산기를 설계한다. 그리고 기존의 연산기들과 비교 분석을 한다.

4.1 제안된 구조의 시뮬레이션

제안된 제곱/곱셈기와 AB^2-곱셈기를 검증하기 위하

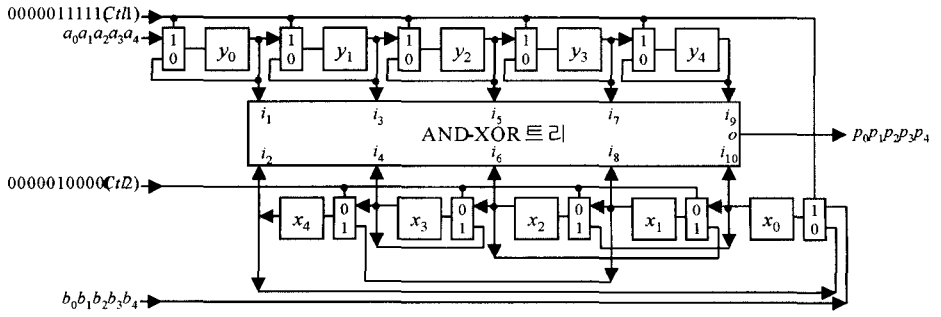


그림 3 GF(2^m)상의 비트-시리얼 AB^2-곱셈기(m=4)

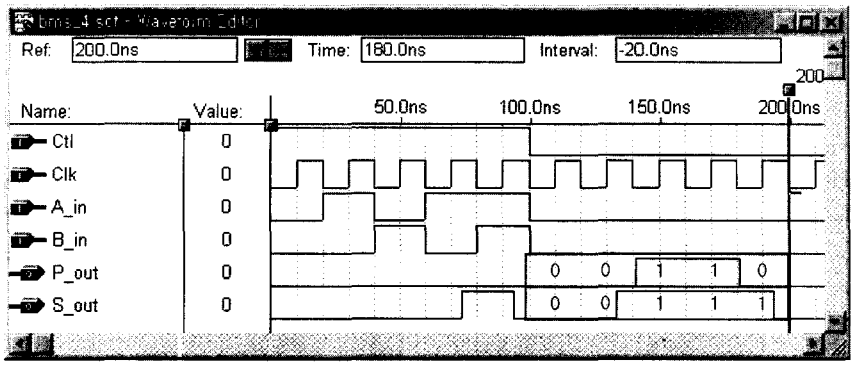


그림 4 비트-시리얼 제곱/곱셈기의 시뮬레이션 결과(m=4)

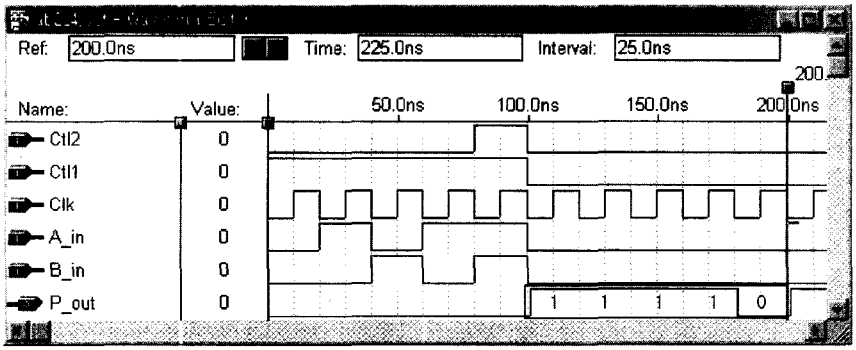


그림 5 비트-시리얼 AB^2-곱셈기 시뮬레이션 결과(m=4)

여 ALTERA의 MAX+PLUS(Multiple Array Matrix Programmable Logic User System) II 툴을 사용해서 회로를 설계하고, 클락 주기(Clock Period)를 20ns로 하여 시뮬레이션을 하였다. 시뮬레이션은 m=4일 경우를 한 것으로 입력 값은 $A=x^3+x+1$(01011)과 $B=x^2+1$(00101)을 사용하였다. 그림 4는 앞 장에서 제안된 그림 1의 제곱/곱셈기를 시뮬레이션 한 결과이다. 곱셈(AB)의 결과 값인 $P=x^2+x$(00110)와 제곱(A^2)의 결과 값인 $S=x^2+x+1$(00111)이 바로 나온 걸 확인할 수 있다. 그림 5는 앞 장에서 제안한 그림 3의 AB^2-곱셈기를 시뮬레이션 한 결과로 AB^2 연산의 결과 값인 $P=x^4+x^3+x^2+x$(11110)가 바로 나온 걸 확인할 수 있다.

4.2 제안된 구조의 응용

지수기와 나눗셈기를 구현하는 방법에는 두 가지 방식이 있다. 첫 번째 방식은 하드웨어 복잡도를 즉 공간을 최소화하는 시리얼로 처리하는 방식이다. 두 번째 방식은 페러럴로 처리하는 방식으로 계산 시간을 최소화하는 방식이다. 그리고 지수기와 나눗셈기를 설계할 때 사용할 곱셈기의 수도 고려되어야 한다. 본 논문에서는 공간을 최소화 하는 시리얼 방식을 채택하고, 제안한 단일화된 제곱/곱셈기와 AB^2-곱셈기를 사용하여 지수기와 나눗셈기를 각각 설계한다.

그림 6은 알고리즘 1을 사용하여 GF(2^m)상의 지수기를 설계한 것이다. 그림 6의 제곱/곱셈기 회로가 본 논문에서 제안한 그림 1의 단일 구조의 비트-시리얼 제곱/곱셈기 회로이다. 이 지수기는 물론 곱셈기 두개를 사용하는 방법과 곱셈기 하나와 제곱기 하나를 사용하는 방법에도 사용될 수 있다. A 레지스터와 C 레지스터, C' 레지스터의 초기값은 각각 A와 1, 1의 값이 저장된다. 그림 6에서 보는 것과 같이 지수 연산시 다음 단계의 값을 선택하기 위해 곱셈 부분에 멀티플렉서(Multiplexer)가 하나 필요하다. 만약 지수의 비트가 '0'이면 전 곱셈 결과의 값이 곱셈 부분에 입력이 되고, '1'이면 현재의 결과가 입력으로 들어간다.

그림 7은 알고리즘 2를 사용하여 GF(2^m)상의 나눗셈 및 역원기를 설계한 것이다. 그림 7의 AB^2-곱셈기는 본 논문에서 제안한 그림 3의 단일 구조의 비트-시리얼 AB^2-곱셈기이다. 이 나눗셈 및 역원기 구조는 물론 곱셈기 두개를 사용하는 방법과 곱셈기 하나와 제곱기 하

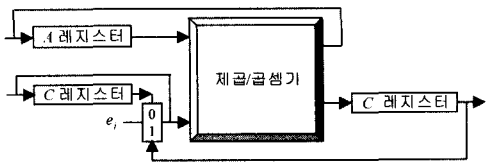


그림 6 알고리즘 1을 이용한 GF(2^m)상의 지수기

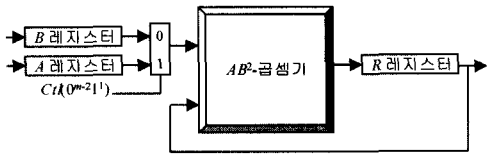


그림 7 알고리즘 2를 이용한 GF(2^m)상의 나눗셈 및 역원기

나를 사용하는 방법에도 사용될 수 있다. A 레지스터와 B 레지스터, R 레지스터의 초기 값은 각각 A와 B, B의 값이 저장된다. 만약 A의 값이 1이면 곱셈에 대한 역원 연산을 수행한다. 제안된 나눗셈 및 역원기는 다음 단계에 쓰일 값을 결정하기 위해 멀티플렉서 하나와 제어신호(ctl)가 필요하다. 제어신호는 (m-2)개의 '0'과 한 개의 '1'로 구성된다. 즉 제어신호는 $(0^{m-2}1^1)$이고, (m-1)단계 후에 결과 값($A/B = AB^{-1}$)이 나온다.

4.3 제안된 구조의 성능 분석

표 1은 Wang 등[5]과 Fenn 등[16]의 구조를 알고리즘 1을 사용하여 GF(2^m)상의 지수기 구조로 비교 분석한 것이다. 제안된 구조의 지수기는 본 논문에서 제안한 단일 구조의 비트-시리얼 제곱/곱셈기 회로를 사용하였다. 그리고 다른 두 지수기는 곱셈과 제곱 연산을 위해 Wang과 Fenn의 곱셈기 각각 두개를 사용한 것으로 가정하였다. 그러나 Wang의 곱셈기는 일반화된 기약 다항식을 사용한 비트-시리얼 방식이고, Fenn의 곱셈기와 제안된 제곱/곱셈기는 기약 AOP를 사용한 비트-시리얼 방식이다. 그리고 지수 연산시 모두 실행 시간 n을 가진다. 비록 Wang의 곱셈기는 임계경로(critical path)가 가장 짧지만 하드웨어 복잡도는 본 논문에서 제안한 구조가 Wang과 Fenn의 구조보다 작음을 알 수 있다. 이는 공간을 최소화하는 이슈를 가지는 곳에 중요하게 쓰일 수 있다.

표 2는 기존 GF(2^m)상의 연산기인 Wei[9]와 Wang 등[13], Fenn 등[16]의 연산기와 본 논문에서 제안한 그림 3의 AB^2-곱셈기를 비교한 것이다. Wei와 Wang의 연산기는 페러럴 입출력과 디지트-페러럴(Digit-Parallel) 입출력 구조를 각각 가지고 일반화된 기약 다항식을 사용한다. 반면에 Fenn의 연산기와 제안된 연산기는 시리얼 입출력 구조를 가지고, 기약 AOP를 사용한다. 그러나 Fenn의 연산기는 나눗셈이나 역원 연산기에 사용을 할 수가 없다. 비록 Wei의 연산기의 임계경로가 가장 짧고 제안한 연산기가 제어신호를 2개를 가지지만 하드웨어 복잡도면에서 월등히 좋음을 알 수 있다.

표 3은 기존 GF(2^m)상의 연산기인 유클리드 알고리즘을 이용하여 구현한 Guo 등[12]의 연산기와 페르마 이론을 이용하여 구현한 Wang 등[13]의 연산기와 본 논문에서 제안한 나눗셈 및 역원기를 비교한 것이다. 제안한 나눗셈 및 역원기는 페르마 이론을 이용하여 구현한 것으로 본 논문에서 제안한 비트-시리얼 AB^2-곱셈기를 이용하였다. 그런데 Guo와 Wang의 입출구조는 시리얼과 디지트-페러럴 구조를 가지나 제안된 구조는 시리얼을 가진다. 비록 제안된 구조가 제어신호를 3개 필요로 하지만 하드웨어 복잡도면에서 기존의 나눗셈기보다 월등히 좋음을 볼 수 있다.

표 1 알고리즘 1을 사용한 GF(2^m)상의 지수기 비교

	Wang 등[5]	Fenn 등[16]	제안된 구조(그림 6)
하드웨어 복잡도	AND_2: $6m$ XOR_3: $2m$ MUX_2: $2m$+1 REG.: $21m$	AND_2: $2m$+2 XOR_2: $2m$ MUX_2: $2m$+5 REG.: $7(m+1)$	AND_2: m+1 XOR_2: m MUX_2: $2m$+3 REG.: $6(m+1)$
곱셈 지연시간	$3m$	$2m$+2	$2m$+2
임계경로	$T_{AND2} + T_{XOR3}$	$T_{AND2} + \lceil \log_2 m + 1 \rceil T_{XOR2}$	$T_{AND2} + \lceil \log_2 m + 1 \rceil T_{XOR2}$
제어신호 수	2	2	2
입출력 형태	시리얼	시리얼	시리얼
구현 방법	곱셈기 × 2	곱셈기 × 2	단일 구조 제곱/곱셈기
지수 실행시간	n	n	n
기약 다항식	일반	AOP	AOP

AND_i : i-입력 AND 게이트(gate)
XOR_i : i-입력 XOR 게이트
MUX_i : i-1 멀티플렉서
REG. : 레지스터
T_{ANDi} : i-입력 AND 게이트 지연시간
T_{XORi} : i-입력 XOR 게이트 지연시간

표 2 GF(2^m)상의 연산기 비교

	Wei[9]	Wang 등[13]	Fenn 등[16]	제안된 구조(그림 3)
하드웨어 복잡도	AND_2: $3m^2$ XOR_2: m^2 XOR_3: m^2 REG.: $10m^2$	AND_2: $3m^2$ XOR_4: m^2 REG.: $8.5m^2$	AND_2: m+1 XOR_2: m MUX_2: m+2 REG.: 2(m+1)	AND_2: m+1 XOR_2: m MUX_2: 2(m+1) REG.: 2(m+1)
지연시간	$4m$	$2.5m$	$2m$+2	$2m$+2
임계경로	T_{AND2} + T_{XOR3}	T_{AND2} + T_{XOR4}	T_{AND2} + $\lceil \log_2 m + 1 \rceil$ T_{XOR2}	T_{AND2} + $\lceil \log_2 m + 1 \rceil$ T_{XOR2}
제어신호 수	0	0	1	2
입출력 형태	패러럴	디지트-패러럴	시리얼	시리얼
연산 종류	AB^2+C	AB^2+C	AB	AB^2
나눗셈 가능성	가능	가능	불가능	적합
기약 다항식	일반	일반	AOP	AOP

표 3 GF(2^m)상의 나눗셈기 비교

	Guo 등[12]	Wang 등[13]	제안된 구조(그림 7)
하드웨어 복잡도	AND_2: 7(m+1) XOR_2: 4(m+1) MUX_2: 10(m+1) REG.: 19(m+1)	AND_2: $6m^2(m-1)/2$ XOR_2: $2m^2(m-1)/2$ REG.: $17m^2(m-1)/2$	AND_2: m+1 XOR_2: m MUX_2: $2m$+3 REG.: 5(m+1)
제어신호 수	1	1	3
입출력 형태	시리얼	디지트-패러럴	시리얼
나눗셈 구현 방법	유클리드 알고리즘	페르마 이론	페르마 이론
기약 다항식	일반	일반	AOP

5. 결 론

본 논문에서는 GF(2^m)상에서 기약 AOP를 사용하는 두 가지 구조를 제안하였다. 하나는 단일 구조의 비트-시리얼 제곱/곱셈기이고, 다른 하나는 단일 구조의 비트-시리얼 AB^2-곱셈기이다. 확장 다항식 기저 상에서 표현되는 원소들의 곱셈과 제곱의 특성들을 이용하여 알고리즘을 기술하였고, 그 알고리즘으로부터 순환 방정식을 유도하여 새로운 구조의 비트-시리얼 연산기들을 설계하였다. 곱셈은 피승수를 순환 시프트 특성을 이용하였고, 제곱은 계수들의 재배치를 이용하였다. 제안된 단일 구조의 비트-시리얼 제곱/곱셈기와 AB^2-제곱기는 지수기와 나눗셈기의 핵심회로로 각각 사용될 수 있었다.

제안된 연산기들은 기약 다항식을 AOP로 사용함으로써 범용 응용에 사용이 제약될 수 있다. 그러나 기존의 연산기들과 비교하여 하드웨어 복잡도면에서 월등히 좋음을 알 수 있었다. 그리고 제안된 연산기들을 지수기와 나눗셈기에 활용함으로써 기존의 지수기와 나눗셈기보다 하드웨어 복잡도면에서 월등함을 보였다. 더 나아가 제안된 구조들은 모듈성과 정규성을 가지고 있어서 쉽게 VLSI로 구현할 수 있으며 IC 카드에 사용될 수 있다.

참 고 문 헌

[1] W. W. Peterson and E. J. Weldom, *Error-Correcting Codes*. Cambridge, MA: MIT Press, 1972.

[2] D. E. R. Denning, *Cryptography and Data security*. Reading MA: Addison-Wesley, 1983.

[3] I. S. Reed and T. K. Truong, "The use of finite fields to compute convolutions," *IEEE Trans. Inform. Theory*, vol. IT-21, pp.208~213, Mar. 1975.

[4] D. E. Knuth, *The art of computer programming, Vol. 2: seminumerical algorithms*. Addison- Wesley, Reading, Mass., 2nd edition, 1981.

[5] C. L. Wang, and J. L. Lin, "Systolic array implementation of multipliers for finite field GF(2^m)," *IEEE Trans. Circuits System*, vol. 38, pp. 796~800, July, 1991.

[6] J. H. Guo and C. L. Wang, "Digit-serial systolic multiplier for finite fields GF(2^m)," *IEE Proc. Compu., Digit. Tech.*, vol. 145, pp.143~148, 1998.

[7] E. D. Mastrovito, "VLSI architecture for computations in Galois fields," Ph. D. dissertation, Dept. Elec. Eng., Linkoping Univ., Linkoping, Sweeden, 1991.

[8] C. K. Koc and B. Sunar, "Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields," *IEEE Trans. Computer*, vol. C-47, pp.353~356, Mar. 1998.

[9] S. W. Wei, "A systolic power-sum for GF(2^m)," *IEEE Trans. Computer*, vol. 43, pp.226~229, Feb. 1994.

[10] C. L. Wang and J. L. Lin, "A systolic architecture

for inverses and divisions in GF(2^m)," *IEEE Trans. Computer*, vol. 42, pp.1141~1146, Sep. 1993.

[11] S. T. J. Fenn, M. G. Parker, M. Benaissa, and D. Taylor, "GF(2^m) multiplication and division over the dual basis," *IEEE Trans. Computer*, vol. 45, pp.319~327, Mar. 1996.

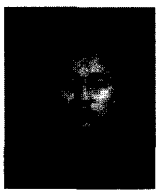
[12] J. H. Guo and C. L. Wang, "Bit-serial Systolic Array Implementation of Euclid's Algorithm for Inversion and Division in GF(2^m)," *Proc. 1997 Int. Symp. VLSI Technology, Systems, and Applications*, pp.113~117, 1997.

[13] C. L. Wang and J. H. Guo, "New Systolic Array for $C + AB^2$, Inversion and Division in GF(2^m)," *IEEE Trans. Computer*, vol. 49, pp.1120~1125, Oct., 2000.

[14] T. Itoh, and S. Tsujii, "Structure of parallel multipliers for a class of fields GF(2^m)," *Info. Trans.*, pp.21~40, 1989.

[15] M. A. Hasan, M. Z. Wang, and V. K. Bhargava, "A modified Massey-Omura parallel multipliers for a class of finite fields," *IEEE Trans. Computer*, C-42, pp.1278~1280, 1993.

[16] S. T. J. Fenn, M. G. Parker, M. Benaissa, and D. Taylor, "Bit-serial multiplication in GF(2^m) using irreducible all-one polynomials," *IEE Proc. Compu., Digit. Tech.*, vol. 144, pp.391~393, 1997.



이 원 호

1998년 대구대학교 전자계산학과 공학사 2000년 경북대학교 컴퓨터공학과 공학석사. 2002년 경북대학교 컴퓨터공학과 공학박사 수료. 관심분야는 정보보호, 암호학, 어레이 프로세서 설계.

유 기 영

1976년 경북대학교 이과대학 수학교육과(이학사). 1978년 한국 과학 기술원 컴퓨터공학과(공학석사). 1993년 New York Rensselaer Polytechnic Institute 컴퓨터과학과(이학박사). 1978년~현재 경북대학교 공과대학 컴퓨터공학과 교수. 관심분야는 암호연산, 병렬처리, 암호화 프로토콜, 정보보호