논문 2004-41SC-6-4

$GF(2^m)$ 상에서의 효율적인 지수제곱 연산을 위한 VLSI Architecture 설계

(Design of VLSI Architecture for Efficient Exponentiation on $GF(2^m)$)

한 영 모*

(Young mo Han)

요 약

유한 필드, 즉 Galois 필드는 에러 정정 코드, 디지털 신호처리, 암호법(cryptography)와 같은 광범위한 응용 분야에 사용되고 있다. 이 응용들은 종종 $GF(2^m)$에서 지수제곱 연산을 필요로 한다. 기존에 제안되었던 방법들은 지수제곱 연산을 반복, 순환적인 곱셈으로 구현하여 계산시간이 많이 걸리거나, 또는 구현 시 하드웨어 구조가 복잡하여 하드웨어 비용이 큰 경우가 많았다. 본 논문에서는 지수제곱 연산을 하는 효과적인 방법을 제안하고 이를 VHDL로 구현하였다. 이 회로는 지수의 각 비트에 해당하는 곱셈 항들을 계산하고 이 들을 곱함으로써 지수제곱 연산을 계산한다. 과거에는 이 알고리즘이 원시 다항식의 근의 지수제곱 연산을 계산하는 데 사용되는 것으로 국한되어 있었으나, 본 논문에서는 이 알고리즘을 $GF(2^m)$의 임의의 원소의 지수제곱 연산으로 확장하였다.

Abstract

Finite or Galois fields have been used in numerous applications such as error correcting codes, digital signal processing and cryptography. These applications often require exponetiation on $GF(2^m)$ which is a very computationally intensive operation. Most of the existing methods implemented the exponetiation by iterative methods using repeated multiplications, which leads to much computational load, or needed much hardware cost because of their structural complexity in implementing. In this paper, we present an effective VLSI architecture for exponentiation on $GF(2^m)$. This circuit computes the exponentiation by multiplying product terms, each of which corresponds to an exponent bit. Until now use of this type algorithm has been confined to a primitive element but we generalize it to any elements in $GF(2^m)$.

Keywords : Galois fields, exponenitator, VLSI, VHDL

I. 서 론

1970년대 이후에 신호처리 기술이 급속도로 발전됨에 따라 기존의 아날로그 방식만으로는 영상 및 음성 신호의 저장과 전송을 원활히 수행할 수 없게 되었다. 또한 과거의 통신 분야에서는 문자, 데이터, 음성 신호 등이 정보원이었으나 오늘날에는 영상과 함께 이들이 혼합된 멀티미디어 신호가 많이 사용되고 있으며, 그 매개체로는 디지털 방식이 보편화 되고 있는 추세이다. 디지털 방식은 아날로그 방식에 비해 신뢰도, 보안성, 그리고 정보의 질 측면에서 우수한 특성을 나타낸다. 하지만, 이러한 특성을 유지하기 위해서는 기존의 아날로그 방식에 비해 많은 정보량을 필요로 한다. 따라서 저장하거나 전송할 때 정보의 신뢰도를 높이기 위한 연구가 많이 진행되었다. 그리고 이 연구들 중 상당수는 Galois 필드 상의 알고리즘에 집중되고 있다.

* 정회원, 이화여자대학교 정보통신공학과
(Dept. of Information Electronics Ewha Womans University)

※ 이 논문은 2004년도 두뇌한국21사업에 의하여 지원되었음.
접수일자: 2004년4월13일, 수정완료일: 2004년11월3일

에러 정정 부호, 디지털 신호처리, 암호법(crypto-graphy)와 같은 통신 분야에서의 중요하고 실질적인 응용 때문에 Galois 필드는 최근 들어 주목을 받고 있다. 특히 Galois 필드 $GF(2^m)$은 각 원소들이 m개의 이진수로 표현될 수 있고, 그 산술의 구현이 쉽기 때문에 많은 연구가 행해져 왔다.

덧셈 같은 간단한 연산은 상대적으로 명백하지만 곱셈, 지수제곱과 같은 복잡한 연산은 큰 수를 계산할 때 효과적으로 수행되기 어렵다. 암호법(Cryptography)의 경우를 살펴보면, 많은 private key와 public key 알고리즘을 수행하는데 고수준의 안정성을 얻기 위해서 $GF(2^{2000})$과 같은 큰 필드에서의 연산이 필요하다. 그러므로 $GF(2^m)$에서의 연산을 위한 효과적인 알고리즘의 개발이 요구된다. 그리고 큰 필드를 사용할 때, 32 또는 64 비트 컴퓨터에서 동작하는 소프트웨어를 범용 프로세서에서 돌리는 것은 상당히 비효율적이기 때문에, 높은 산출율 (throughput rate)을 얻을 수 있는 하드웨어 구현이 주목받고 있다. 최근 VLSI 기술의 발달로 집적도가 증가함에 따라 $GF(2^m)$에서 큰 수의 연산을 하나의 칩에서 실현 할 수 있게 되었다. 이러한 취지에서 몇 가지 회로들이 제안되었다.

본 논문에서는 $GF(2^m)$에서의 지수제곱 연산을 효과적으로 수행하는 VLSI 를 제안 하고자 한다. 지금 까지 연구된 지수제곱 연산 회로는 크게 두 가지가 있다. 첫째로 α를 $GF(2^m)$에서의 원시 다항식(primitive polyno-mial)의 근 이라 할 때 α^E을 구하는 것과[2][3], 둘째로 A를 $GF(2^m)$에서의 임의의 원소라 할 때 A^E를 구하는 것이다. 전자는 주로 1980년대와 그 이전에 많이 연구 되었고 후자는 주로 1990년대에 들어오면서 많이 연구 되고 있다. 본 연구 에서는 후자의 회로를 만드는 데 전자의 목적으로 개발된 아이디어를 도입해 발전, 변형시켜 사용하고자 한다. 여기서 주목하고자 하는 아이디어는 α^E를 구할 때 E의 각 비트별로 계산해서 그 결과들을 곱한다는 것인데, 그 내용을 정리하면 다음과 같다.

지수 E의 이진수 표현을 $(E_{m-1}, E_{m-2}, \cdots, E_0)$라 하고, $P_i = \alpha^{E_i \cdot 2^i}$ 라 하면, $\alpha^E = \prod_{i=0}^{m-1} P_i$ 로 표현될 수 있다. 즉 지수제곱 연산을 지수의 각 비트별로 병렬 형태로 계산할 수 있다는 것이다. 이에 덧붙여 본 논문에서는 P_i 의 특성을 분석하여, P_i 의 계산법을 좀 더 단순화 하는 방법을 제시한다. 이러한 아이디어를 바탕으로 본 논문에서는 병렬 비트 곱셈기 나무구조를 사용해서 $A^E = \prod_{i=0}^{m-1} P_i$ 을 계산하는 회로를 VHDL로 구현한다. 그리고 곱셈기 나무구조의 latency를 줄이기 위해서, 1996년에 개발된 빠르고 구조가 단순한 병렬 비트 곱셈기를 사용하는 방법을 검토한다.

Ⅱ. 기존의 지수제곱 연산기의 구조

1. 연구 동향

현재까지 연구되어 온 지수제곱 연산 회로는 크게 두 가지가 있다. 먼저 1980년대 이전에는 주로 $GF(2^m)$의 원시 다항식(primitive polynomial)의 근을 α라 할 때 α^E를 구하는 회로가 연구 되었다. 그리고 1990년대에 들어와서 $A = a_{m-1}\alpha^{m-1} + \cdots + a_1\alpha + a_0$로 표현 되는 $GF(2^m)$의 임의의 원소 A에 대해, A^E를 구하는 회로가 비로소 연구 되었다. 본 논문에서는 과거에 전자에 사용되었던 아이디어를 발전시켜 후자의 회로를 만드는 것을 그 기본 내용으로 한다.

2. α^E를 구하는 회로

$GF(2^m)$에서 지수제곱 연산을 구하는데 $O(log_2(log_2 E))t$의 계산 시간이 걸리는 병렬 입력-병렬 출력 회로가 1982년에 발표 되었다[4]. 여기서 t는 한 번의 곱셈에 소요되는 시간이다. 이 전의 계산이 완료 될 때까지 다음 계산이 지연되어야 하므로 이 회로는 한 클럭에 한 개의 출력율(throughput rate)을 얻도록 파이프라인 될 수가 없다.

1988년에 이르러 Scott이 지수제곱을 수행하는 직렬 회로(sequential circuit)와 병렬 계산 회로를 제안 하였다[3]. 직렬 회로는 곱셈기를 반복적으로 사용하는 방식에 기초를 두었으며 $O(m^2)$의 계산 시간이 소요된다.

병렬 계산 회로는 $(m-1)$개의 곱셈기를 사용하며 $(m + t \cdot \log_2 m)$ 클럭의 latency를 갖는다. 그리고 이 회로의 출력율은 m 클럭 당 한 번의 지수제곱 연산이다.

본 논문에서 사용한 지수 E의 각 비트별로 product term을 계산하고 이 들을 곱한다는 아이디어는 바로 이 회로에 사용된 것이다.

Ghafoor와 Singh는 병렬 곱셈기를 이용해서 지수제곱 연산을 계산하는 시스톨릭 회로를 1989년에 발표 하였다[5]. 이 회로는 각각의 곱셈기가 m^2개의 cell을 필요로 하는 (2^m-1)개로 이루어진 곱셈기 나무구조를 사용한다. $3m^2$ 클럭의 지연시간을 가지며 매 클럭 당 한 개의 지수제곱 연산을 계산할 수 있다. 1990년에 이르

러 직렬 입력-직렬 출력(serial-in serial-out)지수제곱 연산기가 발표 되었다[6]. 이 회로는 하드웨어 가격과 한 번의 지수제곱 연산을 수행할 때의 속력 면에서는 상당히 효율적이지만, 출력율이 m에 역비례 하는 단점이 있다.

3. A^E를 구하는 회로

가. 제곱과 곱셈 알고리즘(square-and-multiply)[12]

지수 E의 이진수 표현이 $(E(1), E(2), \cdots, E(n))$이라 하면, A^E의 값은 다음과 같이 구할 수 있다.

```
B=1;
for (j =1; j<=n; j++)
  {
    B = B * B;
    if( E(j) = 1) B = B*A;
  }
```

이때 B의 최종 값이 A^E이다.

$GF(2^m)$에서 계산할 때, A와 B는 이진수 계수를 가진 $(m-1)$차 이하의 다항식이다. 곱셈은 그 필드의 m차의 생성 다항식(generating polynomial)의 modulo 연산에 의해 수행된다. 제곱과 덧셈 연산 모두 LFSR (Linear Feedback Shift Register)에 의한 곱셈으로 이루어 질 수 있다[7].

나. Pattern Matching Method를 이용한 지수제곱 연산기[8]

$GF(2^m)$의 0이 아닌 모든 원소는 $1, \alpha, \alpha^2, \cdots, \alpha^{2^m-2}$로 표현될 수 있고 각 원소는 conventional basis로 알려진 $\{1, \alpha, \alpha^2, \cdots, \alpha^{(m-1)}\}$의 선형조합으로 나타낼 수 있다.

$\beta = \beta_{m-1}\alpha^{m-1} + \beta_{m-2}\alpha^{m-2} + \cdots + \beta_1\alpha + \beta_0$

를 $GF(2^m)$의 0이 아닌 원소라 하고 $F(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + \cdots + f_1x + 1$를 원시 다항식(primitive polynomial)이라 하자. 이 때 $\beta_i, f_i \in GF(2)$이다. 이제 circle rotation 함수 ρ를 다음과 같이 정의하자.

$$\rho(\beta) = \alpha\beta(moduloF(\alpha)) \tag{1}$$

그러면 다음의 성질이 성립한다.

$$\rho^i(\beta) = \alpha^i\beta \tag{2}$$

이러한 circle rotation 함수를 회로로 구현해 보자. $\beta = \beta_3\alpha^3 + \beta_2\alpha^2 + \beta_1\alpha^1 + 1$와 $\delta = \rho(\beta) = \delta_3\alpha^3 + \delta_2\alpha^2 + \delta_1\alpha + 1$가 $GF(2^4)$의 원소라 하고 $F(x) = x^4 + x + 1$이라 하자. 그러면 β와 δ 사이에는 다음의 관계식이 성립한다.

$$\begin{aligned}\delta = \rho(\beta) &= \alpha\beta(\bmod ulo\ \alpha^4 + \alpha + 1)\\ &= \beta_3\alpha^4 + \beta_2\alpha^3 + \beta_1\alpha^2 + \beta_0\alpha(\bmod ulo\ \alpha^4 + \alpha + 1)\\ &= \beta_3(\alpha + 1) + \beta_2\alpha^3 + \beta_1\alpha^2 + \beta_0\alpha\\ &= \beta_2\alpha^3 + \beta_1\alpha^2 + (\beta_0 \oplus \beta_3)\alpha + \beta_3\end{aligned} \tag{3}$$

그러므로 다음 관계를 얻는다.

$$\delta_3 = \beta_2\ ,\ \delta_2 = \beta_1\ ,\ \delta_1 = \beta_0 \oplus \beta_3\ ,\ \delta_0 = \beta_3 \tag{4}$$

$\gamma = \alpha^j$가 주어진 $GF(2^m)$의 원소라 하자. 그러면 이 선택에 의해 $GF(2^m)$의 모든 0이 아닌 원소들은 몇 개의 부분집합으로 나누어 진다. 그 부분집합의 시작 원소는 α^{jn} , $n \in \{0, 1, \cdots, \lfloor 2^m - 2 \rfloor\}$이고 원소의 갯수는 j 개 (단, 마지막 부분집합은 j 보다 적은 원소를 가질 수도 있다.)이다. 그리고 $GF(2^m)$의 임의의 원소 $\beta = \alpha^i$ 는 다음과 같이 표현할 수 있다.

$$\beta = \rho^k(\alpha^{jn})\ ,\ n \in \{0, 1, 2, \cdots, \lfloor 2^m - 2\ /j \rfloor\} \tag{5}$$

이 때 $i = |k + jn| \bmod (2^m - 1)$, $0 \le k \le 2^m - 2$, $k < j$ 이고, $\lfloor x \rfloor$는 x보다 작거나 같은 가장 큰 정수이다. 그리고 α^{jn}을 pattern 이라 부른다. $\beta = \rho^k(\alpha^{jn})$일 때 다음의 정리가 성립한다.

$$\beta^e = \rho^r(\alpha^{js}) \tag{6}$$

여기서 $r = |\,|(jn + k)e|\ mod(2^m)|\ modj$

$$s = \lfloor (|(jn + k)e| \bmod (2^m - 1))/j \tag{7}$$

이다. 그리고 곱 $(k + jn)e$는 다음과 같이 계산 되어질 수 있다.

$$(k + jn)e = |\sum_i e_i f_i(k + jn)|\ mod(2^m - 1),\ 0 \le i \le (m - 1) \tag{8}$$

여기서 f_i는 $(k + jn)$을 circular left shift하는 것을 말한다.

이와 같은 이론을 바탕으로 주어진 밑 A에 pattern matching method를 적용하여 지수 형 표현을 찾은 후, circular left shift를 이용하여 지수제곱 연산을 구하고, 마지막으로 다시 pattern matching method를 이용해 다항식 표현을 찾는다.

II. 제안하는 지수제곱 연산 알고리즘

$GF(2^m)$의 임의의 원소 A의 다항식 표현이

$(a_{m-1}, \cdots, a_1, a_0)$라 하고 지수 E의 이진수 표현이 $(e_{m-1}, \cdots, e_1, e_0)$라 할 때, A^E를 구해 보자. 먼저 A와 E는 다음과 같이 표현될 수 있다.

$$A = a_{m-1}\alpha^{m-1} + \cdots + a_1\alpha + a_0 \quad (9)$$

$$E = e_{m-1}2^{m-1} + \cdots + e_1 2 + e_0 \quad (10)$$

이 두 식을 이용하면 A^E는 다음과 같이 계산 될 수 있다.

$$\begin{aligned} A^E &= A^{e_{m-1}2^{m-1} + \cdots + e_1 2 + e_0} \\ &= (A^{e_{m-1}2^{m-1}}) \times \cdots \times (A^{e_1 2}) \times (A^{e_0}) \\ &= pt_{m-1} \times \cdots pt_1 \times pt_0 \end{aligned} \quad (11)$$

$$A^E = \prod_{i=1}^{m} pt_{m-i} \quad (12)$$

즉, m번의 곱셈만으로 A^E를 계산할 수 있다.

(12)식에서 pt_{m-i}는 다음과 같이 계산 될 수 있다.

$$\text{if } e_{m-i} = 0 \text{ then } pt_{m-i} = 1 \quad (13)$$

$$\begin{aligned} &\text{elsief} \quad e_{m-i} = 1 \ , \\ &\text{then} \quad pt_{m-i} = A^{2^{m-i}} \\ &\qquad = (a_{m-1}\alpha^{m-1} + \cdots + a_1\alpha + a_0)^{2^{m-i}} \end{aligned} \quad (14)$$

(14)식을 간단하게 하기 위해서 몇 가지 정리를 도입해 보자.

(정리 1) 모든 솟수 p와 모든 $k \in \{1, 2, 3, \cdots, p\}$에 대해 $p \mid \binom{p}{k}$이다[1].

(정리 2) $\alpha_1, \alpha_2, \cdots, \alpha_t$가 $GF(p^m)$의 원소라 하면 다음의 관계가 성립한다.(단, p는 솟수)[1].

$$(\alpha_1 + \alpha_2 + \cdots + \alpha_t)^{p^r} = \alpha_1^{p^r} + \alpha_2^{p^r} + \cdots \alpha_t^{p^r}$$
$$\text{for } r = 1, 2, 3, \cdots$$

(14)식에 (정리 2)를 적용해 보자.

$$\begin{aligned} pt_{m-i} &= A^{2^{m-i}} \\ &= (a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + \cdots + a_1\alpha + a_0)^{2^{m-i}} \\ &= a_{m-1}\alpha^{2^{(m-i)}(m-1)} + a_{m-2}\alpha^{2^{(m-i)}(m-2)} + \cdots + a_1\alpha^{2^{(m-i)}} + a_0 \end{aligned}$$

이 때 m이 짝수이면 다음과 같이 두 항씩 묶여진다. (m이 홀수이면 a_0를 빼고 두 항씩 묶는다.)

$$pt_{m-i} = A^{2^{m-i}} = \sum_{k=1}^{m/2} \alpha^{(m-2k)2^{m-i}} (a_{m-2k+1}\alpha^{2^{m-i}} + a_{m-2k})$$

$\rho()$를 circle rotation 함수라 하면 다음을 얻을 수 있다.

$$\begin{aligned} pt_{m-i} &= A^{2^{m-i}} \\ &= \sum_{k=1}^{m/2} \rho^{(m-2k)2^{m-i}} (a_{m-2k+1}\alpha^{2^{m-i}} + a_{m-2k}) \end{aligned} \quad (15)$$

여기서

$$\begin{aligned} &(a_{m-2k+1}\alpha^{2^{m-i}} + a_{m-2k}) \\ &= \begin{cases} a_{m-2k} & if \ a_{m-2k+1} = 0 \\ \alpha^{2^{m-i}} + a_{m-2k} & if \ a_{m-2k+1} = 1 \end{cases} \end{aligned} \quad (16)$$

이 조건문은 2:1 mux로 구현될 수 있다. 그리고 $\alpha^{2^{m-i}}$는 주어진 $(m-i)$에 대해 상수이다.

이와 같이 하여 pt_{m-i} for $m-i = m-1, \cdots, 1$을 구한다. 그리고 pt_0는 다음과 같이 구한다.

$$pt_0 = \begin{cases} 1 & if \ e_0 = 0 \\ A & if \ e_0 = 1 \end{cases}$$

이 조건식은 2:1 mux 한 개로 구현할 수 있다.

III. 제안하는 지수제곱 연산기의 VLSI 구현

이 절에서는 앞 절에서 고안한 알고리즘을 VLSI로 구현하는 방법을 예시하고자 한다. 이를 위해 $GF(2^4)$에서의 지수제곱 연산기를 VHDL 로 구현해 본다. 사용되는 원시 다항식은 $F(x) = x^4 + x + 1$로 한다.

1. 회로의 전체 구성

$m=4$의 경우 지수제곱 연산 함수는 (12)식으로부터 다음과 같이 얻어 진다.

$$A^E = pt_3 \times pt_2 \times pt_1 \times pt_0 \quad (17)$$

우리가 구현한 회로에서 입력값은 E[0:3](E의 4 비트 표현) 와 A[0:3](A의 4 비트 다항식 표현)이고, 출력값은 OUT[0:3](A^E의 4 비트 다항식 표현)이다.

그림 1은 그 내부구조를 보인 것이다. DPT3,

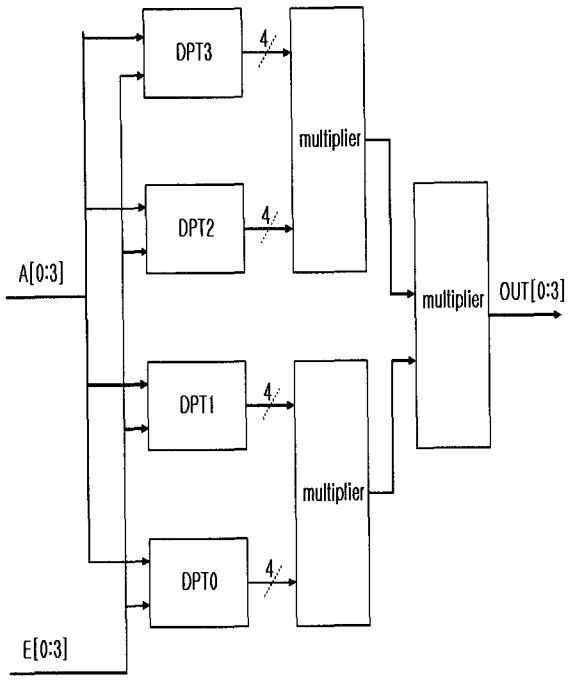

그림 1. 지수제곱 연산기의 내부 구조 (최상위 hierarchy)

Fig. 1. Internal structure of the exponentiator. (top hierarchy)

DPT2, DPT1, DPT0는 각각 pt_3, pt_2, pt_1, pt_0를 계산하는 회로들이다. 곱셈기로서는 $GF(2^m)$의 standard basis와 normal basis 그리고 dual basis에서 설계된 다양한 회로들이 있는데, 여기서는 한 예로서 1996년도에 발표된 dual basis bit parallel multiplier[10]을 사용한다. 이 경우 곱셈기 나무구조의 latency가 줄어드는 장점이 있다. 그러나 standard basis와 normal basis로의 변환과정이 불편한 단점이 생기는데, 다행히 특정$GF(2^m)$에 대해서는 dual basis가 standard basis를 단순히 순서만 바꾼 것으로 표현될 수 있다[11]. 그리고 이 절에서 살펴보고자 하는 지수제곱 연산의 예가 바로 그 경우에 해당한다. 이 내용을 요약해 보면 다음과 같다.

$GF(2^4)$상의 primitive polynomial이 x^4+x+1, standard basis가 $\{1, \alpha, \alpha^2, \alpha^3\}$라 하면 dual basis는 $\{1, \alpha^3, \alpha^2, \alpha\}$이 된다.

2. DPT3, DPT2, DPT1, DPt0

pt_3, pt_2, pt_1을 계산하는 식은 식(15)을 사용함으로써 다음과 같이 얻을 수 있다.

$$pt_3 = \rho(a_3\alpha^8 + a_2) + (a_1\alpha^8 + a_0) \tag{18}$$

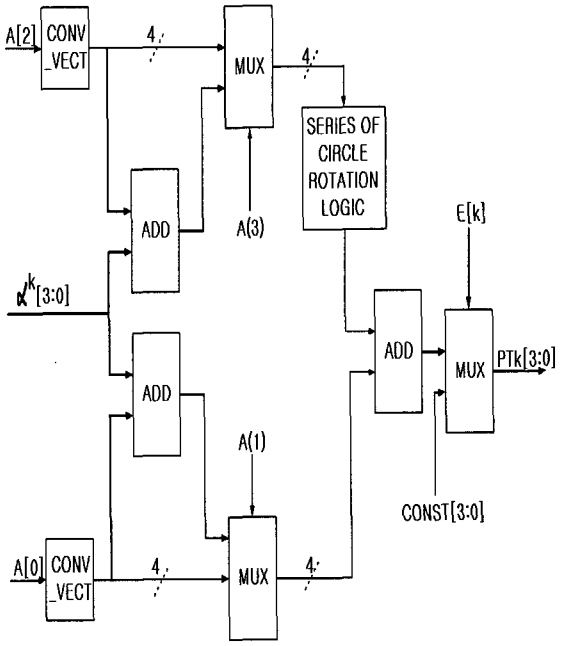

그림 2. DPTk, k=1,2,3의 schematic

Fig. 2. Schematic of DPTk, k=1,2,3.

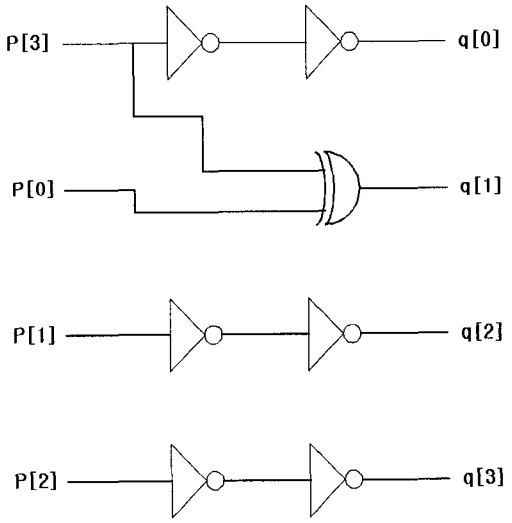

그림 3. Circle rotation logic. k=1, 2, 3 일 때 SERIES OF CIRCLE ROTATION LOGIC에 사용되는 circle rotation logic의 개수는 각각 4개, 8개, 1이다.

Fig. 3. Circle rotation logic.

$$pt_2 = \rho^8(a_3\alpha^4 + a_2) + (a_1\alpha^4 + a_0) \tag{19}$$

$$pt_1 = \rho^4(a_3\alpha^2 + a_2) + (a_1\alpha^2 + a_0) \tag{20}$$

위의 3식을 비교하여 보면 차이점은 단지 circle rotation ρ와 α의 power가 다를 뿐임을 알 수 있다. 따라서 $pt_k, k=1,2,3$ 회로는 k값에 따라 변하는 ρ와 α 블록을 제외한 나머지 부분에서 공통 구조를 가지게 된다. 이

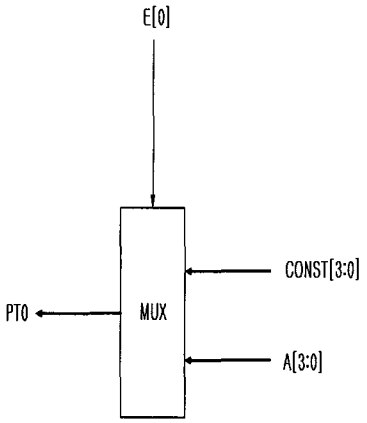

그림 4. DPT0 의 schematic
Fig. 4. Schematic of DPT0.

공통 구조를 구현한 회로, 즉 $DPT_k, k=1,2,3$ 이 그림 2 에 주어져 있다.

이 회로에서 입력값은 E[k](지수 E의 k 번째 bit), A[3](A의 3 번째 bit), A[2](A의 2 번째 bit), A[1](A의 1번째 bit), A[0](A의 0번째 bit)이다. CONST[0:3]에는 0001이 hardwired 되어 있다. 출력값은 PTk[0:3](pt_3의 4비트 표현)이다.

이 그림에서 CONV_VECT회로는 1비트 입력을 받아 4비트 벡터로 변환해 주는 회로이다. 이 때 1비트 입력값은 4비트 벡터의 LSB로 들어간다. ADD는 4비트 xor 회로이다. SERIES OF CIRCLE ROTATION LOGIC은 ρ의 power 연산을 수행하기 위해 circle rotation logic을 직렬로 연결(cascade connection)한 회로이다. Circle rotation logic의 좀더 구체적인 구조와 기능은 그림 3에 주어져 있다.

$pt_k, k=1,2,3$에 비해 pt_0는 훨씬 간단한 회로로 구현될 수 있는데, 그림4 에 그 구현 회로가 주어져 있다.

IV. 성능 평가와 검증

이 절에서는 앞 절에서 제안한 지수제곱 연산기 vexp2의 성능을 평가하고 검증하고자 한다. 먼저 vexp2의 동작을 check해 보기 위해 다음과 같이 test bench 파일을 작성하였다.

```
-- VHDL Test Bench Created from SGE Symbol vexp2.sym.sym
-- Nov  5 11:13:58 1996

library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_arith.all;
   use IEEE.std_logic_components.all;

entity E is
end E;

Architecture A of E is

   signal      DI_A : std_logic_vector (3 downto 0):="0000";
   signal      DI_E : std_logic_vector (3 downto 0):="0000";
   signal      DOUT : std_logic_vector (3 downto 0);

   component VEXP2
      Port (    DI_A : In     std_logic_vector (3 downto 0);
                DI_E : In     std_logic_vector (3 downto 0);
                DOUT : Out    std_logic_vector (3 downto 0) );
   end component;

begin
   UUT : VEXP2
      Port Map ( DI_A, DI_E, DOUT );

-- *** Test Bench - User Defined Section ***
   TB : block
   begin
      di_a(3) <= not di_a(3) after 100 ns;
      di_a(2) <= not di_a(2) after 200 ns;
      di_a(1) <= not di_a(1) after 300 ns;
      di_a(0) <= not di_a(0) after 400 ns;

      di_e(3) <= not di_e(3) after 500 ns;
      di_e(2) <= not di_e(2) after 600 ns;
      di_e(1) <= not di_e(1) after 700 ns;
      di_e(0) <= not di_e(0) after 800 ns;
   end block;
-- *** End Test Bench - User Defined Section ***

end A;

configuration CFG_TB_VEXP2_BEHAVIORAL of E is
   for A
      for UUT : VEXP2
         use configuration WORK.CFG_VEXP2;
      end for;

-- *** User Defined Configuration ***
      for TB
      end for;
-- *** End User Defined Configuration ***

   end for;
end CFG_TB_VEXP2_BEHAVIORAL;
```

그리고 A[0:3]의 최 상위 비트의 입력 값을 100 ns후에 반전시키고, 그 다음 비트의 입력 값은 200 ns후에 반전시키고, 그 다음 비트는 300 ns후에 반전시키는 방식으로 100 ns 만큼의 반전 시간차가 있게 하였다. 마찬가지로 E[0:3]의 최 상위 비트의 입력은 500 ns후에 반전 시키고, 그 다음 비트는 600 ns후에 반전시키는 방식으로 각 비트가 100 ns 씩의 반전 시간차를 갖도록 하였다. 그림5는 이 test bench 파일을 이용해 2000 ns 동안 시뮬레이션한 결과를 보여 주고 있다. 이 그림에서 di_e(3:0), di_a(3:0), dout(3:0)는 각각 그림1에서 E[0:3], A[0:3], OUT[0:3]에 해당한다.

예로서 1800 ns 와 1900 ns 사이의 결과를 살펴보자. di_e(3:0)의 값이 C인데 이는 이진수 표현으로 1100이다. di_a(3:0)의 값이 4(0100)인데 이 값은 다항식 표현이므로 지수제곱 표현으로 바꾸면 α^2이 된다. 이 값의 C승을 하고 mod(15)를 하면 $(\alpha^2)^{12 \bmod 15} = \alpha^9$가 된다. 그리고 α^9는 다항식 표현으로 1010(A)가 된다. 그러므로 출력 dout(3:0)의 값 A가 옳은 것임을 알 수 있다. 같은 방법으로 나머지 경우 들을 살펴 보면 이 회로가 모든 경우에 대하여 올바르게 동작하고 있음을 알 수 있다.

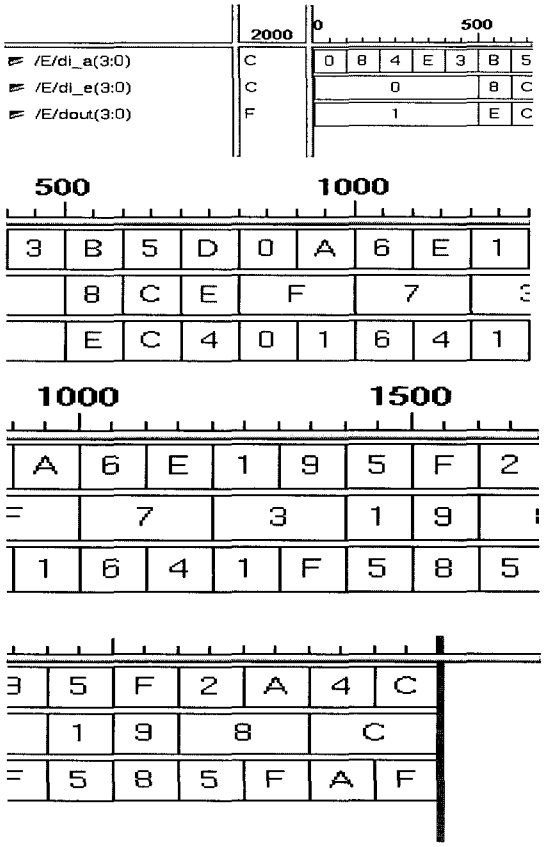

그림 5. vexp2회로의 시뮬레이션 결과.
이 그림에서 di_e(3:0), di_a(3:0), dout(3:0)은 각각 그림1에서 E[0:3], A[0:3], OUT[0:3]에 해당한다.

Fig. 5. Simulation results for vexp2.

다음으로 본 논문에서 제안하는 방법과 기존의 방법의 성능을 비교해 보고자 한다. 먼저 square-and-multiply 방법의 경우를 살펴보자. 참고문헌 [12]에서 제시한 회로 중 한 개의 곱셈기를 이용한 구조의 synchronous mode 연산의 경우를 예로 들어 보자. 이 경우 지수제곱 연산기는 한 개의 곱셈기(multiplier)와 5개의 레지스터(square term register, product term register, irreducible polyno-mial register, expoenent register, output register)로 구성된다. 그리고 한 번의 지수제곱 연산을 하기 위해서는 $(m-1)$번의 제곱(squaring)과 m번의 곱셈(multiplica-tion)이 필요하다. 예로서 $m=4$인 경우의 성능값이 표1에 제시되어 있다. 이 표에서 multiplier latency는 참고문헌 [7]에서와 같이 $2m$ 클럭 사이클(clock cycle)로 가정하였다. 이에 덧붙여 본 논문에서는 각각의 연산 stage의 결과 값이 레지스터를 거쳐서 다음 연산 stage로 입력되는 시간을 1 클럭 사이클로 간주하였다. 그러면 지수제곱 연산기의 총 latency는 $(2m+1)((m-1)+m)$ 클럭 사이클, 즉 61 클럭 사이클이 된다.

다음으로 pattern matching 방법의 경우를 살펴보자. 참고문헌 [8]에서 구현한 $m=4, j=4$인 경우의 지수제곱 연산기를 예로 들어보면, 크게 pattern matching module M1, exponentiation module M2, result transfor-mation module M3로 구성되어 있다.

M1 module은 16개의 cell element PE와 polynomial selection flip-flop (PS FF)로 구성되어 있다. 16개의 PE들은 15개의 register에 의해 선형으로 pipeline(li-neary connected as a pipeline) 되어 있는데, 각각은 1 클럭 사이클 당 한 번의 pattern match를 수행할 수 있다. 각 PE 의 중요 구성 원소로는 comparator 1개, circle rotation logic 1개를 들 수 있다.

M2 module은 3개의 circular left shift register와 3개의 modulo adder로 구성되어 있다. 이 3개의 modulo adder는 2개(즉 $log_2 m$ 개)의 stage로 구성되어있다. 그리고 각 stage는 6개의 register를 사용해서 2단계로 pipeline되어 있다. 이 회로에서 modular adder array의 첫 번째 stage에는 circular left shift register의 결과값이 입력된다.

M3 module은 크게 pattern matching systolic array와 circle rotation systolic array로 구성되어 있다. Pa-ttern maching systolic array는 4개의 cell로 구성되어 있는데, 각 cell의 주요 성분으로는 1개의 comparator를 들 수 있다. 또한 circle rotation systolic array는 3개의 seg로 구성되어 있는데, 각 seg의 주요 성분으로는 1개의 compartor와 1개의 circle rotation logic을 들 수 있다. 그리고 M3는 7개의 register를 사용해서 7단계로 pipeline 되어 있다.

M1, M2, M3의 latency는 각각 15, 2, 7 클럭 사이클이므로, 지수제곱 연산기의 총 latency는 24 클럭 사이클이 된다.

지금까지 살표본 pattern matching방법의 $m=4, j=4$ 경우의 성능 값은 표 1에 요약되어 있다.

마지막으로 본 논문에서 제안한 방법의 경우를 살펴보자. $m=4$인 경우의 지수제곱 연산기는 크게 3개의 곱셈기와 4개의 DPT 회로(DPTk, k=0,1,2,3)로 구성된다. 여기서 3개의 곱셈기는 $\log_2 m$ 개, 즉 2개의 stage로 구성된 array 구조를 갖는데, 첫 번째 stage에 DPT 회로가 연결되어 있다. 이 구조는 앞에서 설명한 pattern matching 방법의 M2와 유사한 구조이므로, 앞의 경우와 같은 방법으로 6개의 레지스터에 의해 2단계로 파이프라인 되어 있는 것으로 간주한다. 본 논문에서는 곱셈기로서 **Dual Basis Bit Parallel Multip**

-lier[10] 를 사용하는 예를 보였는데, 이 경우 곱셈기가 조합회로(combinational circuit) 이므로 지수제곱 연산기의 총 latency는 $log_2 m$, 즉 2 클럭 사이클이 된다. 그런데 곱셈기로서 Dual Basis Bit Parallel Multiplier[10] 대신에 일반적인 standarad basis나 normal basis에서 구현된 곱셈기를 사용하는 경우를 생각할 수 있다. 이 경우 square-and -multiply 방법[12]에서와 같이 곱셈기의 latency를 $2m$으로 간주하면, 지수제곱 연산기의 총 latency는 $(2m+1)* log_2 m$, 즉 18 클럭 사이클이 된다.

위에서 살펴 본 바와 같이 본 논문에서 제안한 지수제곱 연산기의 장점은 m이 증가할 때 급속도로 증가하는 요소가 없고, 속력이 매우 빠르다는 것이다. $m=4$ 인 경우에 세 가지 방법의 성능비교 결과를 표1에 요약하였다. 이 표에서도 알 수 있듯이 본 방법은 기존의 두 가지 방법 모두에 비해 latency가 짧고 빠르다. 그리고 하드웨어의 복잡도는 pattern matching 방법에 비해 대단히 간단하고, square-and-multiply 방법에 비해서도 그다지 크지 않다. 표1에는 제안하는 방법에 곱셈기로서 dual basis bit parallel multiplier 대신에 standard basis와 normal basis에서 설계된 일반 곱셈기를 사용하는 경우도 고려하여 괄호 안에 표시하였다. 이때 곱셈기의 latency는 square-and-multiply 방법에서와 같이 $2m$ 클럭 사이클로 간주하였다. 표1의 결과를 보면 일반 곱셈기를 사용하였을 때의 latency가 dual basis bit parallel multiplier를 사용했을 때 보다 조금 길어지기는 하지만, 여전히 기존의 두 방법에 비해 latency와 하드웨어 복잡도 면에서 우수한 특성을 가지고 있음을 확인할 수 있다.

그러므로 본 회로는 serial계산 방법이 하드웨어를 적게 사용하는 대신에 속력이 느리고, parallel 방법이 속력이 빠른 대신 하드웨어가 복잡하다는 trade-off 문제에 적절한 개선 방안을 제시한 것이라 할 수 있다.

표 1. $m=4$ 일 때 세 회로의 성능 비교
제안하는 방법의 latency 값 중 괄호 안의 수치는 일반 곱셈기(standard basis 또는 normal basis에서 설계된 곱셈기)를 사용한 경우를 표시하고 있다.

Table 1. Performance comparison of the three circuits when $m=4$.

	square-and-multiply method	pattern matching 방법($j=4$)	제안하는 방법
latency (clock cycle)	61	24	2(18)
레지스터	5	28	6
곱셈기	1	0	3
덧셈기	0	3	9
comparator	0	22	0
mux	0	22	10
Primitive Cell	0	-circle rotation 회로 18개 -PS FF 1개	-circle rotation 회로 13개

Ⅳ. 결 론

본 논문에서는 Galois field 상에서 지수제곱을 계산하는 효과적인 알고리즘을 제안하고, 이를 VLSI로 구현하는 예시를 보였다.

구현된 VLSI 회로는 기존의 방법에 비해 latency가 짧고 하드웨어의 복잡도도 그다지 크지 않다.

따라서 본 VLSI 회로는 serial계산 방법이 하드웨어를 적게 사용하는 대신에 속력이 느리고, parallel 방법이 속력이 빠른 대신 하드웨어가 복잡하다는 trade-off 문제에 적절한 개선 방안을 제시한 것이라 할 수 있다.

참 고 문 헌

[1] Stephen B. and Wicker, Error Control Systems for Digital Communication and Storage, Pentice Hall, 1996.

[2] D. E. Knuth, The Art of Computer Programming, Vol. 2, Seminumerical Algorithms. Readng, MA :Addisonp_Wesley, 1969.

[3] P. A. Scott, S. J. Simmons, S. E. Tavares, and L. E. Peppard, "Architectures for exponentiation in $GF(2^m)$," IEEE J. Select. Areas Commun., vol. 6, pp. 578-586, Apr. 1988.

[4] B. Fam and J. Kowalchuk, "A VLSI device for fast exponentiation in finite fields," in Proc, IEEE Int. Conf. Circuits Comput., New York, Sept. 1982, pp. 368-371

[5] A. Ghafoor and A. Singh, "Systolic architecture for finite field exponentiation," in Proc. IEEE, pt. E, Nov. 1989, vol. 136, pp. 465-470.

[6] C. C. Wang and D. Pei, "A VLSI design for computing exponetiations in $GF(2^m)$ and its app -lication to generate pseudornadom number sequences," IEEE Trans Comput., vol. 39, pp.258-262, Feb. 1990.

[7] B. Arzi, "Architectures for Exponentiation Over $GF(2^n)$ Adopted for Smatcard Application," IEEE Trans. Comput., vol. 42, pp.494-497, April 1993.

[8] Mario Kovac and N. Rangnathan, "ACE: A

VLSI Chip for Galois Field $GF(2^m)$ Based Exponentiation," IEEE Trans. Circuts and Systems., vol. 43, pp289-297, April 1996.
[9] C. L. Wang and J. L. Lin, "A systolic architecture fo computing inverses and divisions in finite fields $GF(2^m)$," IEEE Trans. Comput., vol. 42, pp. 1141-1145, Sept. 1993.
[10] Sevastian T. J. Fenn, Mohammed Benaissa, and David Taylor, " $GF(2^m)$ Multiplication and Divi -sion Over the Dual Basis," IEEE Trans. Com -put., vol. 45, pp. 319-327, March 1996.
[11] M. Morii, M. Kasahara, and D. L. Whiting, "Efficient Bit-Serial Multiplication and the Discerte-Time Wiener-Hopft Equation over Finite Fields," IEEE Trans. Information Theory, vol. 35, pp. 1,177-1,183, Nov. 1989.
[12] P. Andrew Scott, Stanley J. Simmons, Stafford E. Tavares, LLoyd E. Peppard, "Architectures for Exponentiation in $GF(2^m)$," IEEE Trans. Selected Areas In Communications, vol. 6, no. 3, pp. 578-586, April 1988.

저 자 소 개

한 영 모(정회원)
1992년 서울대학교 물리교육학과 학사 졸업.
1995년 서울대학교 제어계측공학과 학사 졸업.
1998년 서울대학교 전기공학부 석사 졸업.
2002년 서울대학교 기계항공공학부 박사 졸업
2002년~2003년 단국대학교 기계공학과 시간강사,
세종-록히드마틴 우주항공연구소 연구원, 연구교수
2004년~현재 이화여자 대학교 정보통신공학과 연구전임강사
<주관심분야: 컴퓨터비전, 컴퓨터비전 관련 영상처리 및 신호처리, Human-computer Interaction, robotic system control, 전기, 기계, 물리 분야의 융합테마(fusion theme)>