논문 01-26-3B-13 한국통신학회논문지 '01-3 Vol.26 No.3B

유한체 상에서 비트-직렬 곱셈기의 병렬화 기법

정회원 최 영 민*, 양 군 백*

Parallelism of the bit-serial multiplier over Galois Field

Young-min Choi*, Kun-paek Yang* *Regular Members*

요 약

유한체(Galois Field) 상에서의 곱셈(multiplication)을 구현하는 방법은 크게 병렬 곱셈기(parallel multiplier)와 직렬 곱셈기(serial multiplier)로 나누어 질 수 있는데, 구현시 하드웨어 면적을 작게 차지한다는 장점 때문에 직렬 곱셈기가 널리 사용된다. 하지만 이 직렬 곱셈기를 이용하여 계산을 하기 위해서는 병렬 곱셈기에 비해 많은 시간이 필요하게 된다. 직렬기법과 병렬기법의 결합이 이를 보완할 수 있게 된다. 본 논문에서는 복잡도는 직렬 곱셈기와 큰 차이가 없으면서 연산시간을 줄인 곱셈기(multiplier)를 제안하였다. 이 곱셈기를 사용하면 복잡도는 크게 늘어나지 않으면서 유한체 상에서의 곱셈을 하는데 필요한 시간을 줄이는 효과를 얻을 수 있다.

ABSTRACT

The method of implementing the multiplier over Galois Field is usually divided to "parallel multiplier" and "serial multiplier". Serial multiplier is widely used because the hardware area for implementation is small. But when using serial multiplier, we need more time for multiplication. Combination of serial and parallel scheme makes it possible to resolve the problem. In this paper, we propose the multiplier that has the similar complexity as serial multiplier and the small computation time. Using the proposed multiplier, we can get the effect that the time for multiplication lessens without the hardware's large increase.

I. 서 론

유한체(Galois field, or finite field)는 그 원소의 개수가 유한하기 때문에 많은 특징들을 가지고 있다. 유한체의 특징을 이용하기 위하여, 근래의 많은 시스템에서 데이터를 유한체 상에서 처리한다. 통신 시스템의 채널코딩(channel coding) 및 암호학이 그 예라고 할 수 있다. 특히 사용되는 거의 모든 시스템이 디지털 시스템이기 때문에 주로 고려되는 유한체는 원소의 개수가 2의 승수인, 즉 GF(2^N) 의 유한체이다. 유한체 상에서의 연산은 가·감산과 곱셈, 그리고 나눗셈이 있다. 이를 구현 측면에서 보면 가·감산은 간단해 곱셈과 나눗셈이 고려 대상이 된다. 그리고 나눗셈은 역을 구하여 곱하는 것으로 생각할 수 있으므로 곱셈을 구현하는 것이 가장 큰 문제가 된다.

유한체 상에서 곱셈을 구현하는 방법은 크게 두 가지로 나눌 수 있다. 비트-직렬 곱셈기(bit-serial multiplier)와 병렬 곱셈기(parallel multiplier)가 그것이다. 디지털 시스템에서 주로 메모리(memory, 특히 ROM)로 구현되는 병렬 곱셈기는 곱셈을 하기 위해 필요한 클럭(clock)의 수가 적지만 구현하는데 많은 하드웨어 면적을 차지한다는 단점이 있다. 이를 보완하기 위해 나온 것이 비트-직렬 곱셈기인데, 병렬 곱셈기에 비해 필요한 하드웨어 면적이 작기 때문에 널리 사용되고 있다. 하지만 이것은 곱셈을 하는데 필요한 클럭의 수가 많다는 단점을 가지고 있다. 디지털 시스템에서 하나의 연산에 필요한 클럭의 수가 크다면 그 시스템의 성능을 크게 악화시

* 데이콤 종합연구소 광대역무선개발팀(ymchoi@dacom.net),
논문번호 : 00345-0628, 접수일자 : 2000년 6월 28일

킬 수 있기 때문에 클럭의 수를 줄이는 방법을 찾아야 한다.

본 논문에서는 복잡도는 비트-직렬 곱셈기와 큰 차이가 없으면서 연산에 필요한 클럭의 수를 줄일 수 있는 병렬화된 비트-직렬 곱셈기(parallelized bit-serial multiplier)를 제안하였다. 이 곱셈기를 이용하면 복잡도는 크게 늘어나지 않으면서 유한체 상에서의 연산시간을 줄이는 효과를 얻을 수 있다. 또 병렬화 깊이(parallizing depth)를 조절할 수 있기 때문에, 그 시스템에서 곱셈을 하는데 사용할 수 있는 최대 클럭의 수를 만족시키는데까지만 병렬화를 함으로써 시스템의 최적화에도 기여할 수 있다.

본 논문의 구성을 살펴보면, 2장에서 기존에 널리 사용되고 있는 비트-직렬 곱셈기의 구조를 살펴보았다. 3장에서는 병렬화 기법을 이용한 비트-직렬 곱셈기를 제안하였고, 결론으로 4장에서 병렬화 기법의 이점 및 사용용도에 대해서 설명하였다.

Ⅱ. 비트-직렬 곱셈기

비트-직렬 곱셈기(bit-serial multiplier)는 Berlekamp가 Reed-Solomon 코드의 부호화기(encoder)를 만들면서 사용한 것이다[1]. 이 비트-직렬 곱셈기의 구조를 이해하기 위해 약간의 대수학적 기초가 필요하다. 본 장에서는 본 논문을 이해하는데 필요한 대수학적 기초를 먼저 설명하고 Berlekamp의 비트-직렬 곱셈기의 일반적인 구조를 살펴본다.

1. 대수학적 기초

정의 1 : q^m개의 원소를 가지고 있는 GF(q^m)의 기저(basis) $\{\mu_i\}$는 GF(q^m) 안에 있는 m개의 선형적으로 독립된 원소들(linearly independent elements)의 집합이다.

γ을 GF(q^m) 상의 임의의 원소라고 하자.

정의 2 : 만일 집합 $\{\gamma^k\}$, $0 \le k \le m-1$가 GF(q^m)의 기저(basis)라면, 기저 $\{\gamma^k\}$, $0 \le k \le m-1$를 규범적(canonical) 또는 다항식(polynomial) 기저라고 부른다.

정의 3 : GF(q^m)에 속한 한 원소 β의 형적(trace)은 다음 식 (1)과 같이 정의된다.

$$tr(\beta) = \sum_{i=0}^{m-1} \beta^{q^i} \tag{1}$$

그 형적(trace)은 다음 식 (2)와 같은 특성을 가진다.

$$\begin{aligned} &1)\ tr(\beta+\gamma) = tr(\beta)+tr(\gamma) \\ &2)\ tr(\beta)\text{는 GF}(q^m)\text{ 상의 값을 가진다.} \\ &3)\ tr(\beta^q) = tr(\beta)^q = tr(\beta) \\ &4)\ tr(1) = m \pmod q \end{aligned} \tag{2}$$

정의 4 : 두 개의 기저(basis) $\{\mu^k\}$와 $\{\lambda_i\}$는 아래의 식 (3)과 같은 조건하에서 서로 보상적이다(complementary) 또는 이중적이다(dual) 라고 말한다.

$$tr(\mu_i \lambda_k) = \begin{cases} 1, & \text{if } i=k \\ 0, & \text{if } i \neq k \end{cases} \tag{3}$$

정의 5 : 위너-호프(Wiener-Hopf)의 이산식(Discrete-time Equation)은 다음 식 (4) 형태의, t개의 미지수 a_i($i=0, 1, \cdots, t-1$)와 $2t-1$개의 계수 s_i($i=0, 1, \cdots, 2t-2$), 그리고 t개의 상수 b_i($i=0, 1, \cdots, t-1$)로 구성되는 t개의 선형 비등가식(linear inhomogeneous equation)으로 구성되는 시스템으로 정의된다. 여기서 식 (4)를 DTWHE(Discrete-Time Wiener-Hopf Equation)이라고 부른다. 단, 여기서 a_i, s_i, 그리고 b_i는 GF(q)의 원소들이다.

$$\begin{bmatrix} s_{2t-2} & s_{2t-3} & \cdots & s_{t-1} \\ s_{2t-3} & s_{2t-4} & \cdots & s_{t-2} \\ \vdots & \vdots & \ddots & \vdots \\ s_{t-1} & s_{t-2} & \cdots & s_0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_3 \end{bmatrix} \tag{4}$$

2. Berlekamp의 비트-직렬 곱셈 알고리듬

Berlekamp는, GF(q^m) 상에서 다항식 기저(polynomial basis)와 그것의 이중 기저(dual basis) 사이의 관계를 이용하여, 효율적인 Reed-Solomon 부호화기(encoder)의 구현방법을 제안하였다[1]. 이 부호화기는 비트-직렬 곱셈기(bit-serial multiplier)의 한 응용 예라 할 수 있다. 다음 정리는 비트-직렬 곱셈 알고리듬의 이론적인 배경을 제공한다.

정리 1 : 두 개의 기저(basis) $\{\mu^k\}$와 $\{\lambda_i\}$가 보상적(complementary)이라고 하면, 원소 Z는 이중기저(dual basis) $\{\lambda_i\}$ 상에서 다음 식 (5)와 같이 표현될 수 있다.

$$Z = \sum_{k=0}^{m-1} z_k \lambda_k = \sum_{k=0}^{m-1} tr(Z\mu_k)\, \lambda_k \tag{5}$$

(단, $z_k = tr(Z\mu_k)$는 이중기저의 k번째 계수)

정리 1의 내용을 명확히 하기 위해 간단한 예를 하나 보자. GF(2^4)의 원시함수(primitive polynomial)가 $p(z)=z^4+z+1$이고, α가 $p(\alpha)=0$를 만족하는 GF(2^4) 상의 원시원소(primitive element)라고 한다면, GF(2^4) 상의 다항식 기저(polynomial basis)는 집합 $\{1, \alpha, \alpha^2, \alpha^3\}$이다. 그러면 그것의 보상적 기저(complementary basis)는 집합 $\{\alpha^{15}, \alpha^2, \alpha, 1\}$이다. u와 v가 GF(2^4)의 원소라면 이 둘은 다음과 같은 식 (6)으로 표현될 수 있다. 여기서 u_k와 v_k는 GF(2)의 원소이다.

$$\begin{aligned} u &= u_0\cdot 1+u_1\cdot\alpha+u_2\cdot\alpha^2+u_3\cdot\alpha^3 \\ v &= v_0\cdot\alpha^{14}+v_1\cdot\alpha^2+v_2\cdot\alpha+v_3\cdot 1 \end{aligned} \tag{6}$$

그리고 u와 v의 곱 $s(s=u\cdot v)$는 GF(2^4)의 원소이며 다음 식 (7)과 같이 표현될 수 있다.

$$s = s_0\cdot\alpha^{14}+s_1\cdot\alpha^2+s_2\cdot\alpha+s_3\cdot 1 \tag{7}$$

정리 1과 형적(trace)의 특성으로부터 위의 $\{s_0, s_1, s_2, s_3\}$를 구할 수 있는데 이를 식 (8)에 나타내었다.

$$\begin{aligned} s_0 &= tr(s\cdot 1) \\ &= tr(u\cdot v\cdot 1) \\ &= u_0\cdot v_0+u_1\cdot v_1+u_2\cdot v_2+u_3\cdot v_3 \\ s_1 &= tr(s\cdot\alpha) \\ &= tr(u\cdot v\cdot\alpha) \\ &= u_1\cdot v_0+u_2\cdot v_1+u_3\cdot v_2 \\ &\quad +(u_1+u_2)\cdot v_3 \\ s_2 &= u_2\cdot v_0+u_3\cdot v_1+(u_0+u_1)\cdot v_2 \\ &\quad +(u_1+u_2)\cdot v_3 \\ s_3 &= u_3\cdot v_0+(u_0+u_1)\cdot v_1 \\ &\quad +(u_1+u_2)\cdot v_2+(u_2+u_3)\cdot v_3 \end{aligned} \tag{8}$$

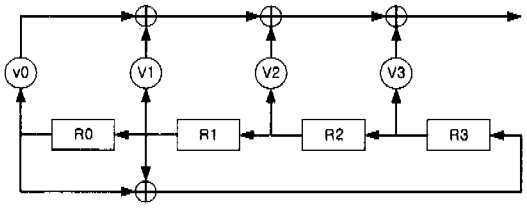

그림 1. GF(16) 상에서 비트-직렬 곱셈기의 예

따라서 위 식 (8)는 다음 식 (9)와 같이 행렬로 표현될 수 있다. 이 행렬을 살펴보면 2.1절에서 설명한 위너-호프의 이산식(DTWHE) 임을 알 수 있다. 이 식 (9)를 풀기 위한 회로는 쉬프트 레지스터(shift register)의 형태로 쉽게 구현 할 수 있는데 이를 그림 1에 나타내었다.

$$\begin{bmatrix} u_0 & u_1 & u_2 & u_3 \\ u_1 & u_2 & u_3 & (u_0+u_1) \\ u_2 & u_3 & (u_0+u_1) & (u_1+u_2) \\ u_3 & (u_0+u_1) & (u_1+u_2) & (u_2+u_3) \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \tag{9}$$

3. 비트 직렬 곱셈기의 구조

2.2절에서 설명한 Berlekamp의 비트-직렬 곱셈기(bit-serial multiplier)의 구조를 설명하기 위해 또다른 예를 살펴본다. 원시함수(primitive polynomial)가 $p(z)=z^8+z^4+z^3+z^2+1$인 GF(2^8) 상에서의 곱셈을 생각해보자. 이 유한체(Galois field)에 포함되는 원소는 8비트(bit) 즉 1바이트(byte)로 표현가능하기 때문에 채널코딩에서 가장 흔히 발견할 수 있는 경우가 된다. 이 GF(2^8) 상에서의 비트-직렬 곱셈기는 다음 그림 2와 같은 구조를 가진다. 그림 2에서 살필 수 있듯이 비트 곱셈이 이루어지는 곳에 저장될 원소($\{v_0, v_1, \cdots, v_7\}$) 는 규범적(canonical) 형태를 가지고 있고 쉬프트 레지스터(shift register)에 새로이 저장될 원소($\{r_0', r_1', \cdots, r_7'\}$)는 이중적(dual) 형태를 가진다. 그리고 출력 역시 이중적 형태가 된다.

본 절의 예에서 규범적(canonical) 형태의 원소와 이중적(dual) 형태의 원소 사이의 변환은 다음 식 (10)을 통해서 이루어질 수 있다. 식 (10)에서 알 수 있듯이 규범적 형태의 원소와 이중적 형태의 원소사이의 변환은 그리 복잡하지 않기 때문에 규범적 형태의 원소와 이중적 형태의 원소를 같이 사용하는 것은 성능에 큰 영향을 미치지 않는다. 따라서 이후 회로의 복잡도를 계산하는데 이들 사이의 변환은 고려하지 않는다.

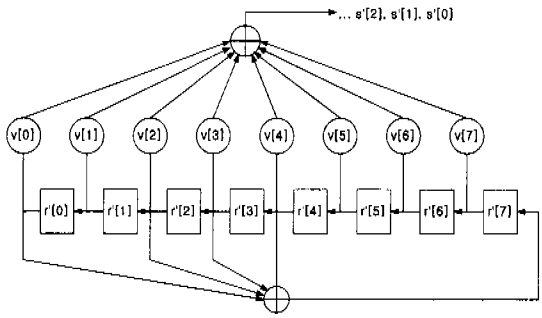

그림 2. GF(256) 상에서 비트-직렬 승산기의 예

다음 그림 3은 그림 2의 회로를 이용하여 곱셈을 할 때 예상되는 타이밍도(timing diagram)이다. 이러한 구조의 곱셈기를 이용하여 곱셈을 하는데 필요한 클럭(clock)의 수는 그림 3에서 확인할 수 있듯이 8개가 필요하게 된다. 그리고 회로를 구현하는데 레지스터(register) 8개와 한 개의 곱의 합(sum of product)을 수행하는 회로, 그리고 XOR을 수행하는 한 개의 회로가 필요하다. 이 구조의 곱셈기는 회로의 복잡도는 작지만 필요한 클럭의 수가 많다는 단점을 가지고 있다. 따라서 만일 어떤 시스템 내에서 곱셈을 하는데 허락된 시간이 8클럭 미만이라면, 제대로 동작하는 시스템을 만들 수 없기 때문에 메모리(memory, 특히 ROM)를 이용한 병렬 곱셈기(parallel multiplier)를 사용해야 한다. 하지만 병렬 곱셈기를 사용하면 2^{16}비트의 메모리를 가져야 하기 때문에 많은 하드웨어를 차지하게 된다. 이를 해결하기 위해 다음 장에서는 이러한 비트-직렬 곱셈기(bit-serial multiplier)를 병렬화한 곱셈기를 제안하였다.

$$
\begin{aligned}
x'_0 &= x_2 + x_0 \\
x'_1 &= x_1 \\
x'_2 &= x_0 \\
x'_3 &= x_7 \\
x'_4 &= x_6 \\
x'_5 &= x_5 \\
x'_6 &= x_4 \\
x'_7 &= x_3 + x_7
\end{aligned}
\tag{10}
$$

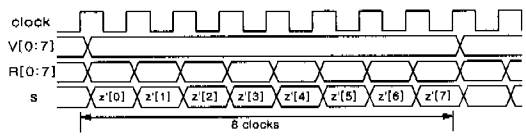

그림 3. GF(256) 상에서 비트-직렬 승산기의 timing 도

Ⅲ. 병렬화된 비트-직렬 곱셈기

본 장에서는 비트-직렬 곱셈기(bit-serial multiplier)를 병렬화하는 방법을 제시하였다. 이를 일반화하기 전에, 먼저 2.3절에서 제시한 예를 이용하여 병렬화를 설명한다. 1절에서는 2.3절의 예를 병렬화할 때 병렬화 깊이(parallelizing depth)에 따른 곱셈기의 구조와 하드웨어 복잡도를 살펴보고 이를 2절에서 일반화 한다.

1. 병렬화 깊이에 따른 비트-직렬 곱셈기의 예

1.1 병렬화 깊이가 1인 경우

2.3절 그림 2안의 쉬프트 레지스터(shift regitster)를 병렬화함으로써 곱셈을 수행하는데 필요한 클럭(clock)의 수를 줄일 수 있다. 그림 4는 병렬화 깊이(parallelizing depth)를 1로 하였을 때, 병렬화된 비트-직렬 곱셈기(parallelized bit-serial multiplier)의 회로도이다. 먼저 $\{v[0], v[1], \cdots, v[7]\}$와 $\{r'[0], r'[1], \cdots, r'[7], r'[8]\}$가 각각의 레지스터(register)에 저장된다. $\{v[0], v[1], \cdots, v[7]\}$는 규범적(canonical) 형태의 GF(256) 상에 있는 원소이고 $\{r'[0], r'[1], \cdots, r'[7]\}$는 이중적(dual) 형태의 GF(256) 상에 있는 원소이다. 그리고 $r'[8]$는 초기에 $r'[0]\hat{}r'[2]\hat{}r'[3]\hat{}r'[4]$로 저장된다. 이렇게 두 값이 각각의 레지스터에 저장되고 나면 곱셈된 결과 바이트(byte)의 첫 번째 비트(bit)와 두 번재 비트를 얻을 수 있다. 이들은 $s'[t]$와 $s'[t+1]$를 통해서 출력된다. 그리고 이 회로에 다시 클럭이 인가되면 쉬프트 레지스터는 이동된 또는 새로운 값을 가지고 되고 그때의 $s'[t]$와 $s'[t+1]$는 곱셈된 결과 바이트의 다음 비트들이 되어 출력된다.

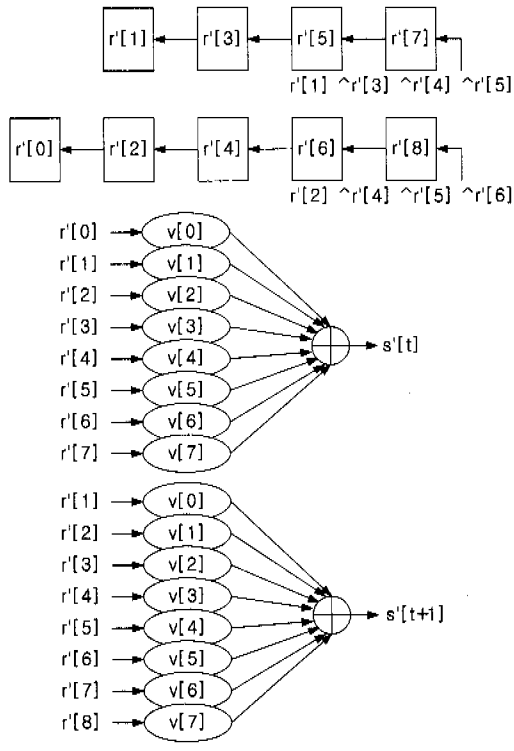

그림 4. 병렬화된 비트-직렬 승산기의 구조 (병렬화 깊이=1)

이런 방식으로 8비트의 결과 바이트를 구하게 되면 이 회로를 통한 곱셈은 끝나게된다. 따라서 다음 곱셈을 위해 새로운 초기값을 레지스터에 저장하거나 이 회로를 비활성화시키면 된다. 이 회로가 동작할 때 각각의 신호에 따라 시간영역에서 예상되는 타이밍도(timing diagram)가 그림 5이다.

그림 4에서 알 수 있듯이 그림 2의 비트-직렬 곱셈기(bit-serial multiplier)의 구현시보다 추가로 필요한 회로는 1개의 레지스터와 곱의 합(sum of product)을 계산하는 회로 1개, 그리고 XOR 를 수행하는 회로 2개가 필요하게 된다(초기에 $r'[8]$ 를 저장할 때 쓰인 XOR 은 그림 4에 나타나 있지 않다). 반면에 곱셈을 수행하는데 필요한 클럭의 수는 그림 5에서 볼 수 있듯이 4개로 2.3절의 비트-직렬 곱셈기를 이용하여 곱셈을 할 때의 반절로 줄게 된다. 즉 약간의 회로를 추가함으로서 곱셈을 수행하는데 걸리는 시간을 줄였다.

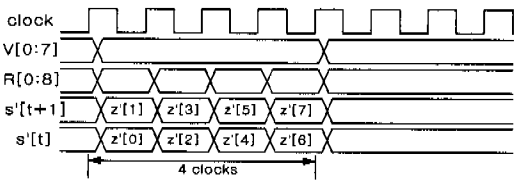

그림 5. 병렬화된 비트-직렬 승산기의 timing 도 (병렬화 깊이 = 1)

1.2 병렬화 깊이가 2인 경우

3.1.1절에서 병렬화했던 방법과 비슷하게 한번 더 병렬화를 할 수 있다. 그림 6는 병렬화깊이(parallelizing depth)가 2인 경우에 병렬화된 비트-직렬 곱셈기(parallelized bit-serial multiplier)의 회로도를 나타낸 그림이다. 우선 규범적(canonical) 형태의 GF(256) 상에 있는 원소인 $\{v[0], v[1], \cdots, v[7]\}$와 이중적(dual) 형태의 GF(256) 상에 있는 원소인 $\{r'[0], r'[1], \cdots, r'[7]\}$, 그리고 $\{r'[8], r'[9], r'[10]\}$이 각각의 레지스터(register)에 저장된다. $\{r'[8], r'[9], r'[10]\}$는 초기에 $\{r'[0]\hat{}r'[2]\hat{}r'[3]\hat{}r'[4],\ r'[1]\hat{}r'[3]\hat{}r'[4]\hat{}r'[5],\ r'[2]\hat{}r'[4]\hat{}r'[5]\hat{}r'[6],\ r'[3]\hat{}r'[5]\hat{}r'[6]\hat{}r'[7]\}$의 값으로 저장된다. 이렇게 저장되고 나면 곱셈된 결과 바이트(byte)의 첫 4비트(bit)를 얻을 수 있다. 이들은 $\{s'[t], s'[t+1], s'[t+2], s'[t+3]\}$를 통해서 출력된다. 그리고 이 회로에 다시 클럭(clock)이 인가되게 되면 쉬프트 레지스터(shift register)는 이동된 또는 새로운 값을 가지고 되고 그때 $s'[t]$, $s'[t+1]$, $s'[t+2]$, 그리고 $s'[t+3]$는 곱셈된 결과 바이트의 다음 비트들로 출력되게 된다. 이렇게 되면 이 회로를 통한 곱셈은 끝나게 된다. 따라서 다음 곱셈을 위해 새로운 값을 레지스터에 저장하거나 이 회로를 비활성화시키면 된다. 이회로가 동작할 때 각각의 신호에 따라 시간영역에서 예상되는 타이밍도(timing diagram)가 그림 7이다.

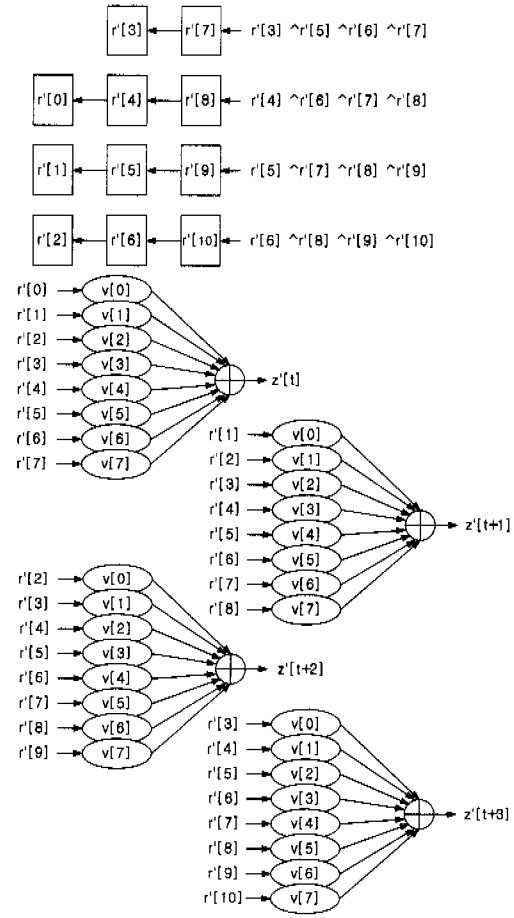

그림 6. 병렬화된 비트-직렬 승산기의 구조 (병렬화 깊이 = 2)

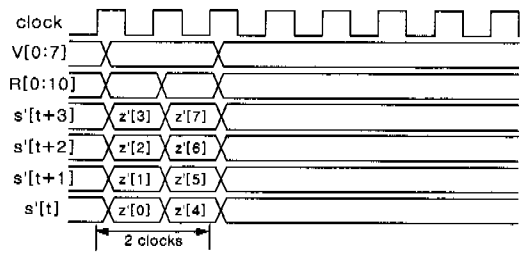

그림 7. 병렬화된 비트-직렬 승산기의 timing 도 (병렬화 깊이 = 2)

그림 6에서 알 수 있듯이 그림 2의 비트-직렬 곱셈기(bit-serial multiplier)의 구현시와 비교할 때, 추가로 3개의 레지스터(register)와 곱의 합(sum of product)을 계산하는 회로 3개, 그리고 XOR 를 수행하는 회로 6개가 필요하게 된다(초기에 $\{r[8], r[9], r[10]\}$ 를 저장할 때 쓰인 XOR 은 그림 6의 회로에는 나타나 있지 않다). 하지만 곱셈을 하는데 필요한 클럭(clock)의 수는 그림 7의 타이밍도(timing diagram)에서 볼 수 있듯이 2개로 2.3절의 비트-직렬 곱셈기를 이용하여 곱셈을 할 때보다 훨씬 작다. 즉 추가의 회로가 곱셈을 수행하는데 걸리는 시간을 상당히 줄였다고 할 수 있다.

2. 병렬화된 비트-직렬 곱셈기의 일반화

3.1절에서 GF(256) 상에서 곱셈을 살펴 보았다. 이 예는 가장 쉽게 발견할 수 있는 예이므로 자세히 살펴보는 것은 큰 의미를 가진다. 다음 표는 GF(256) 상에서의 구현방법에 따른 하드웨어의 양과 필요한 클럭(clock)의 수를 나타낸 것이다. 비트-직렬 곱셈기(bit-serial multiplier)의 경우, 차지하는 하드웨어의 양은 많지 않지만 필요한 클럭의 수가 8개로 크다는 것을 알 수 있다. 또 병렬 곱셈기를 이용할 경우 필요한 클럭의 수는 1개지만 차지하는 하드웨어는 32Kbit 메모리로 상당히 크다는 것을 알 수 있다. 이들의 중간인 병렬화된 비트-직렬 곱셈기(parallelized bit-serial multiplier)를 이용하면 필요한 클럭의 수가 비트-직렬 곱셈기(bit-serial multiplier)와 병렬 곱셈기의 중간이지만 증가하는 하드웨어의 양은 그리 많지 않다. 따라서 어떤 시스템 내에서 곱셈기가 필요하다면 그 시스템 내에서 허용되는 클럭의 수가 나타날만큼만을 병렬화하면 될 것이다.

표 1. 직렬, 병렬, 그리고 병렬화된 곱셈기의 비교 (DFF : D flip-flop, SOP : Sum of Product, XOR : 4-input Exclusive OR)

곱셈기의 종류	필요한 하드웨어	클럭 수
비트-직렬 승산기	8·DFF, 1·SOP, 1·XOR	8
병렬화된 비트-직렬 승산기(P = 1)	9·DFF, 2·SOP, 3·XOR	4
병렬화된 비트-직렬 승산기(P = 2)	11·DFF, 4·SOP, 7·XOR	2
병렬곱셈기	2^{16} Memory (32 Kbits)	1

이제까지는 GF(256) 상에서의 곱셈을 살펴보았다. 위의 예를 토대로 GF(256)가 아닌 다른 유한체 상에서의 곱셈으로 일반화할 수 있다. 임의의 유한체 GF(2^N) 상에서의 곱셈을 가정해보자. Berlekamp의 비트-직렬 곱셈기(bit-serial multiplier)를 이용하여 구현할 경우 그 회로를 이용하여 곱셈을 마치는데 필요한 클럭(clock)의 개수는 N개이고, 필요한 회로는 N비트(bit) 쉬프트 레지스터(shift register)와 곱의 합(Sum of Product) 등을 수행하는 조합(combinational) 회로이다. 반대로 메모리를 이용한 병렬 곱셈기를 이용할 경우 필요한 메모리의 양은 2^{2N}비트로 N이 커짐에 따라 그 크기는 지수함수적으로 증가하게 된다. 따라서 N이 큰 값을 때 큰 하드웨어를 차지하기 때문에 병렬 곱셈기를 사용하기 힘들게 된다. 하지만 비트-직렬 곱셈기로 구현하면 곱셈을 하는데 필요한 클럭의 개수가 N이 되어 필요한 시간 역시 커지게 된다.

병렬화된 비트-직렬 곱셈기를 이용할 경우 위 두 경우의 중간에 있게된다. 병렬화 깊이(parallelizing depth)를 P라고 하자. P가 1이면 곱셈을 하는데 필요한 클럭의 수가 비트-직렬 곱셈기에서 필요한 클럭의 수의 절반이 된다(만일 나누어지지 않을 경우, 올림을 하면 된다). 그리고 P가 1씩 증가할 때마다 곱셈을 하는데 필요한 클럭의 수는 또다시 절반으로 줄어들게 되고, 결국 필요한 클럭의 수가 2가 될 때까지 병렬화 할 수 있다. 따라서 병렬화된 비트-직렬 곱셈기가 곱셈을 마치는데 필요한 클럭의 수는 다음 식 (11)로 요약될 수 있다.

$$n_{p-bs} = \lceil \frac{N}{2^P} \rceil \tag{11}$$

2장에서 제시한 비트-직렬 곱셈기를 병렬화하는 예에서는 병렬화를 할 때 증가하는 회로의 모양이 일정하였다. 이는 GF(256)의 원시함수(primitive polynomial)가 $p(z) = z^8+z^4+z^3+z^2+1$로 z^8 밑으로 z^4 이상의 아무런 항이 없기 때문에 나타난 현상이다. 즉 이는 회로로 구현될 때 쉬프트 레지스터(shift register)에 나중에 입력되는 값의 탭(tap)을 이용하지 않기 때문이다. 하지만 원시함수의 z^N항 밑으로 $z^{N/2}$보다 큰 항이 존재한다면 병렬화를 함으로서 추가되는 회로는 조금 더 복잡해질 것이다. 따라서 유한체(Galois field)의 크기는 정해져 있고 그 유한체를 만드는 원시함수를 선택할 수 있다면, 원시함수의 z^N항 밑으로 $z^{N/2}$보다 큰 항이 없는

원시함수를 선택하는 것이 병렬화된 비트-직렬 곱셈기를 구현하는데 도움이 될 것이다.

Ⅳ. 결 론

본 논문에서는 유한체(Galois field) 상에서의 곱셈을 수행하는 회로로 병렬화된 비트-직렬 곱셈기(parallelized bit-serial multiplier)를 제안하였다. 이는 가장 흔히 사용되는 Berlekamp의 비트-직렬 곱셈기(bit-serial multiplier)가 곱셈을 수행하는데 걸리는 시간이 크다는 단점을 극복하는 방안으로 제안하였다. 비트-직렬 곱셈기가 쉬프트 레지스터(shift register)를 이용하고 있다는 점을 착안하여 이 쉬프트 레지스터를 병렬화 함으로써 곱셈에 필요한 클럭(clock)의 수를 줄인 것이다. 이를 위해 회로가 추가되어야 하는데 이는 하드웨어의 복잡도를 크게 늘리지 않았다.

유한체 상에서의 곱셈기가 한 시스템 내에서 구현될 때 곱셈을 하는데 허용된 클럭의 개수가 제한되는 경우가 있다. 이런 경우 만일 비트-직렬 곱셈기의 곱셈시간이 그 허용된 클럭의 수를 초과한다면 본 논문에서 제안된 병렬화 방법은 하나의 해결책이 될 것이다. 병렬화를 진행시킬수록 필요한 클럭의 수가 작아지므로, 시스템내의 허용된 클럭의 수 밑으로 떨어질 때까지 병렬화를 함으로서 시스템을 최적화한다는 의미도 가진다.

또 본 논문에서 제시한 GF(256) 상의 예는 채널 코딩 등의 범주에서 가장 흔히 발견할 수 있는 것으로 이 경우의 구현 방법을 자세히 살펴본 것은 큰 의미를 가진다고 생각된다.

참 고 문 헌

[1] E. R. Berlekamp, "Bit-serial Reed-Solomon encoder," IEEE Trans. Inform. Theory, vol. IT-28, pp. 869-874, Nov. 1982.

[2] R. Lidl and Niederreiter, Finite field, Addison-Wesley Publishing Co., Reading, MA, 1983.

[3] M. Morii and M. Kasahara, "Efficient bit-serial multiplication and the discrete-time Wiener-Hopf equation over finite field," IEEE Trans. Inform. Theory, vol. IT-35, pp. 1177-1183, Nov. 1989.

[4] M. Wang and I. F. Blake, "Bit serial multiplication in finite field," SIAM J. DISC. MATH, vol. 3, pp. 140-148, Feb. 1990.

최 영 민(Young-min Choi) 정회원

1996년 2월 : 인하대학교 전자공학과 공학사
1998년 2월 : 포항공과 대학교 전자전기공학과 석사
1998년 1월~현재 : 데이콤 종합연구소 광대역무선 개발팀 연구원
<주관심 분야> 부호이론, 통신이론, 통신 H/W설계

양 군 백(Kun-paek Yang) 정회원

1982년 2월 : 광운대학교 통신공학과 공학사
1985년 2월 : 동국대학교 전자계산공학과 공학석사
1983년~현재 : 데이콤 종합연구소 광대역무선개발팀 책임연구원
<주관심 분야> 초고속 인터넷가입자 전송기술, Radio Propagation