새로운 유한체 나눗셈 알고리즘

김 의 석, 정 용 진
광운대학교 전자통신공학과
전화 : 02-940-5551 / 핸드폰 : 011-454-7194

A New Finite Field Division Algorithm

Eui-Seok Kim, Yong-Jin Jeong
Dept. of Electronics & Communications Engineering, Kwangwoon University
E-mail : check95@explore.kw.ac.kr

요 약

본 논문에서는 확장 유클리드 알고리즘을 이용하여 VLSI 구현에 적합한 GF(2^m)에서의 나눗셈 알고리즘을 제안하였다. 제안하는 나눗셈 알고리즘은 GF(2^m)에서 2m-2번의 반복적인 비트 연산을 필요로 하며 입력 데이터에 의존적인 하드웨어 구조를 새로운 (m+1)-bit의 유한체 G와 H를 도입하여 간단하게 제어하도록 구현하였다. 본 논문에서 제안하는 알고리즘은 유한체 곱셈과 나눗셈이 요구되는 Error Correction Code와 암호 알고리즘에 효율적으로 적용이 가능하다. 현재 대표적으로 사용되는 기존 나눗셈 알고리즘과 비교해 볼 때 연산 시간은 비슷하지만 2-bit의 제어신호만을 필요로 하기 때문에 입력 데이터에 독립적인 O(1)의 complexity를 가짐으로 O($\log_2$ (m+1))의 컨트롤을 갖는 다른 두 알고리즘에 비해 하드웨어 리소스 면에서 월등한 결과를 보인다.

I. 서론

유한체 나눗셈($A \cdot B^{-1}$: $A, B \in GF(2^m)$)은 Reed-Solomn Code와 같은 Error Correction Code, 그리고 ECC(Elliptic Curve Cryptography)와 같은 암호 알고리즘들에서 많이 사용되는 연산이다.

본 논문은 광운대학교 IDEC 센터의 툴 지원으로 수행되었습니다.

유한체 나눗셈을 구현하기 위한 대표적인 알고리즘으로는 확장 유클리드 알고리즘과 페르마 이론에 근거한 방법이 있다. 하드웨어로 구현할 경우 확장 유클리드 알고리즘이 페르마 이론보다 빠른 성능을 보인다. 예를 들면 확장 유클리드 알고리즘의 경우 연산을 위해 2m번의 반복적인 덧셈 연산이 필요하지만 페르마 방법의 경우 m($\log_2$ (m-1)+HW(m-1)-1) 혹은 m($\log_2$ (m-2)+1)번의 유한체 덧셈이 필요하다[3][4]. 여기에서 HW()는 Hamming weight를 의미한다. 따라서 확장 유클리드 알고리즘의 성능이 보다 빠르다.

확장 유클리드 알고리즘을 이용하여 나눗셈기를 구현한 대표적인 방법들로는 H. Brunner[1]와 J. Guo[2]의 논문이 있다. 두 논문에서의 알고리즘은 나눗셈 연산을 제어하기 위해 컨트롤 로직으로 $\log_2$ (m+1)-bit up/down counter와 zero check를 사용한다. 또한 연산 시간에서 Brunner 알고리즘의 경우 2m번의 반복적인 연산이 필요하며 Guo 알고리즘의 경우 2m-2번의 반복적인 연산이 필요하다. 따라서 입력 데이터의 크기가 커질수록 제어 신호를 위한 컨트롤 로직 부분의 사이즈가 커지게 된다. 또한 counter의 결과 값을 zero check에서 사용하기 때문에 보다 긴 연산 시간을 필요로 하게 된다.

본 논문에서는 입력 데이터 사이즈 m에 무관한 고정된 크기의 컨트롤 로직을 가지는 확장 유클리드 알고리즘 기반의 나눗셈기를 설계하였다. 제안하는 나눗셈 알고리즘은 위의 두 방법과 유사하게 GF(2^m)에

서 2m-2번의 반복적인 연산이 필요하다. 하지만 입력 데이터에 의존적인 하드웨어 구조를 2-bit의 제어 신호를 이용하여 간단하게 제어하도록 구현하였다. 2-bit의 제어 신호만을 필요로 하기 때문에 입력 데이터에 독립적인 O(1) complexity의 하드웨어 리소스를 필요로 한다. 앞의 두 방법이 $O(\log_2 (m+1))$의 컨트롤 로직을 필요로 하는 것에 비해 본 논문의 방법이 훨씬 작은 하드웨어 리소스를 요구함을 알 수 있으며, 필드 사이즈 m이 커질수록 그 차이는 더욱 커지게 된다.

2. 확장 유클리드 알고리즘

유한체 나눗셈은 역수 연산을 수행하는 확장 유클리드 알고리즘을 이용하여 연산한다. 기본적인 확장 유클리드 알고리즘은 식 (1)과 같다.

$$R=\sum_{i=0}^{m} r_i x^i,\ r_i \in \{0,1\};\ D=\sum_{i=0}^{m} d_i x^i,\ d_i \in \{0,1\};$$

$$U=\sum_{i=0}^{m} u_i x^i,\ u_i \in \{0,1\};\ V=\sum_{i=0}^{m} v_i x^i,\ v_i \in \{0,1\};$$

$R=F;\ D=B;\ U=1;\ V=0;$
While$(R!=0)$ *begin*
  $R=R-Q\cdot D;$
  $R\leftrightarrow D;$
  $V=Q\cdot U+V;$
  $V\leftrightarrow U;$
end
return $V(=B^{-1})$ (1)

여기서 ↔는 두 변수의 값을 교환하는 것을 나타낸다. 알고리즘 (1)을 살펴보면 유한체 나눗셈과 곱셈 후에 두 다항식의 값을 서로 교환하는 과정이 필요하고 나눗셈과 곱셈을 위해 deg(R)-deg(D)(deg(x)는 x의 차수)번의 반복적인 계산과정이 필요하기 때문에 입력되는 값에 따라 필요한 연산 시간이 바뀌게 된다. 연산을 위한 루프가 끝날 때까지 몇 번의 반복적인 연산이 필요한지 알 수가 없기 때문에 구현상에 어려움이 있다. 본 논문에서는 이것을 해결하기 위해 유한체 나눗셈이 계산 단계인지 아니면 계산을 위한 준비 단계인지를 나타내는 mode신호와 두 다항식의 값이 언제 교환될지를 나타내는 swap신호를 이용하여 해결하였다.

3. 제안하는 나눗셈 알고리즘

본 논문에서 제안하는 나눗셈 알고리즘은 다음과 같다.

$D=F;\ R=B;\ U=A;\ V=0;$
$swap=0;\ mode=1;\ G=(0\cdots 0)_2;\ H=(10\cdots 0)_2;$
for $i=1$ *to* $2k-2$ *begin*
  $q=r_m;$
  $R\_p=R;$
  $R=R\oplus(r_m\cdot D);$
  $swap\_p=(h_m+\overline{mode})\,r_{m-1};$
  $mode=r_{m-1}+mode\cdot\overline{h_m};$
  $U=\{(q\cdot V)\oplus(h_m\cdot T)\oplus U\} \bmod F;$
  if$(g_m==0)$
    $U=U\cdot x;\ V=V;\ T=T;$
  else
    $U=0;\ V=U;\ T=V;$
  end
  $g_m\leftrightarrow h_m;$
  if$(swap==1)$ *begin*
    $g_{[m-1:0]}\leftrightarrow h_{[m-1:0]};$
    $D=R\_p;$
  end
  $R=R\cdot x;\ G=G/x;\ H=H\cdot x;$
  $swap=swap\_p;$
end (2)

알고리즘 (2)는 (1)의 확장 유클리드 알고리즘을 비트 단위로 계산하도록 표현하여 하드웨어 구현이 용이하도록 한 것이다. 여기서 ⊕는 XOR 연산, 그리고 $A_{[x:y]}$는 A 다항식의 x번째 항에서 y번째 항까지의 비트 열을 나타낸다. $GF(2^m)$에서 2m-2번의 반복적인 계산이 필요하며 제어 부분과 유한체 나눗셈 그리고 곱셈의 세 부분으로 나눌 수 있다.

표 1에서 $x^2+x+1/x^3+x+1 \bmod x^4+x+1$ (A/B mod F)의 예를 들어 제안하는 알고리즘을 설명하였다. 유한체 나눗셈과 곱셈 후에 두 다항식의 교환이 이루어진다. 두 다항식이 교환되는 시점은 mode와 swap 그리고 새롭게 도입한 G와 H의 다항식에 의해 결정되는데 G와 H는 $(0...0)_2$와 $(10...0)_2$로 초기화 된다. 이 후 반복적인 연산 동안 g_m와 h_m이 교환되며 swap 신호가 1인 경우에만 $g_{[m-1:0]}$과 $h_{[m-1:0]}$이 교환된다. 여기서 H의 최상위 항의 값이 1일 때 유한체 나눗셈의 끝임을 나타내며, mode와 swap신호를 결정하는데 사용된다.

mode 신호는 유한체 나눗셈이 계산되고 있는지 아니면 유한체 나눗셈을 시작하기 위한 준비단계에 있는

표 1. 제안된 새로운 나눗셈 알고리즘의 GF(2^4)에서의 예제

	mode	swap	H	G	R	D	V	U	T
1	1	0	x^4	0	x^3+x^2+1	x^4+x+1	0	x^2+x+1	0
2	1	1	0	x^3	x^4+x^3+x	x^4+x+1	x^2+x+1	0	0
3	1	0	x^4	0	x^4+x	x^4+x^3+x	x^2+x+1	x^3+x^2+x	0
4	1	1	0	x^3	x^4	x^4+x^3+x	x^3+1	0	x^2+x+1
5	1	0	x^4	0	x^4+x^2	x^4	x^3+1	1	x^2+x+1
6	0	0	0	x^3	x^3	x^4	x^3+x^2+x+1	0	x^3+1
7	1	1	0	x^2	x^4	x^4	x^3+x^2+x+1	0	x^3+1
8	1	0	x^3	0	0	x^4	x^3+x^2+x+1	x^3+x^2+1	x^3+1

지를 나타내며 swap신호는 R과 D 다항식의 값이 언제 교환될지를 나타낸다. R과 D의 값은 유한체 나눗셈이 끝났을 때, R의 최상위 항의 값이 1일 경우에는 그 다음 단계에서 동작이 일어나지만 R의 최상위 항이 1이 아닌 경우에는 R 다항식을 좌측으로 이동한 후, 즉 mode신호가 0인 동안에 R 다항식의 최상위 항이 1이 되면 교환 동작이 일어나야 한다.

(2)에서 다항식 R의 값이 0이 될 때 A/B mod F는 유한체 곱셈의 우측 변에 있는 V값이 된다. V는 이전 반복 루프에서 U 항등식의 값이므로 마지막 유한체 나눗셈을 계산할 필요 없이 U의 값이 결과 값이 된다. 따라서 마지막 유한체 나눗셈은 비트 연산으로 계산하였을 때 최소 2번의 반복 루프를 필요로 하므로 2m-2번의 반복적인 연산만을 필요로 한다.

4. 전체 구조

제안한 알고리즘은 나눗셈과 곱셈 그리고 제어를 위한 세 부분으로 나눌 수 있다. 이 가운데 제어 부분은 유한체 나눗셈의 연산 이후 두 변수의 값을 교환하는 제어 신호를 결정하며 결정된 제어 신호에 따라 나눗셈과 곱셈을 연산한다.

그림 1은 제안하는 알고리즘의 전체 구조를 보여주고 있다. 그림 1에서 C는 제어 부분이며 D는 나눗셈 그리고 M은 곱셈을 위한 부분이다. C의 제어 로직은 연산 과정 중에 다항식을 교환하는 제어신호 swap와 mode를 결정한다. 또한 나눗셈과 곱셈을 위한 몫을 D와 M에 전달한다. D와 M은 제어신호를 받아 연산을 하며 다항식의 swap 신호에 의해 교환 과정을 거친다.

그림 2는 각각의 세부 로직을 나타내고 있다. 그림 2(a)는 그림 1에서 C의 세부로직이며 (b)는 D의 세부로직, 그리고 (c)는 M의 세부 로직이다. (a)에서 점선으로 나타낸 부분이 다음 연산을 위한 제어 신호를 결정하는 부분이며 나머지는 나눗셈과 곱셈의 몫을 결정하는 부분이다. (c)에서 점선으로 나타낸 부분은 modulus 연산을 할 때 사용되며, irreducible polynomial, F(x)의 계수가 '1'인 PE에만 필요하다.

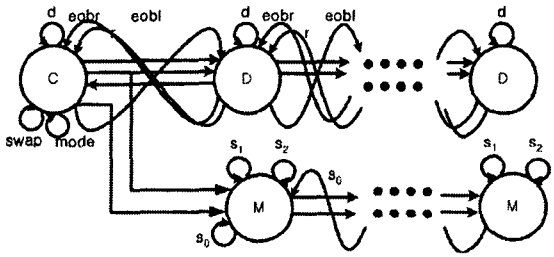
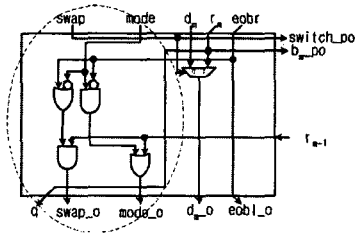
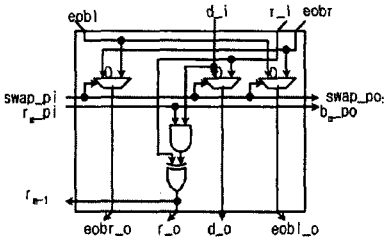


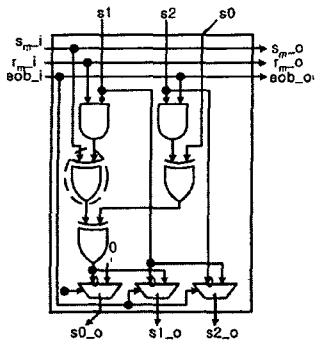
그림 1. 제안하는 나눗셈기의 구조



(a)



(b)



(c)

그림 2. 각 노드의 세부 로직
(a) control logic
(b) division logic
(c) multiplication logic

표 2는 확장 유클리드 알고리즘을 이용하여 나눗셈기를 구현한 대표적인 논문과 제안한 새로운 나눗셈 알고리즘을 비교한 표이다. [1]에서는 나눗셈 연산을 위해 2m의 반복적인 연산 과정이 필요하지만 [2]에서 제안한 알고리즘은 2m-2의 반복적인 연산과정이 필요하다. 두 알고리즘에서는 제어 신호를 결정하기 위해 $\log_2(m+1)$-bit의 up/down카운터와 zero check를 필요로 한다. 따라서 하드웨어로 설계 시 $O(\log_2(m+1))$ complexity의 하드웨어 리소스를 필요로 하며 입력 데이터의 크기가 커질수록 이에 비례해 하드웨어 리소스를 많이 필요로 하게 된다. 또한 zero check의 출력 값이 나눗셈 연산의 마지막에 결정되기 때문에 나눗셈기의 성능을 저하시키는 요인이 된다.

제안하는 알고리즘에서는 나눗셈을 제어하기 위한 컨트롤 부분을 두개의 (m+1)-bit의 유한체 G와 H를 도입함으로서 입력 데이터의 크기에 독립적인 2-bit의 제어 신호만을 필요로 하는 구조로 설계하였다. 또한 제안하는 알고리즘은 나눗셈 연산을 위하여 간단한 컨트롤 부분과 나눗셈 그리고 곱셈의 세 부분의 모듈로 구현이 가능하다. 따라서 파이프라인 설계에 유리하다.

표2. 각 구현 사례에 대한 비교

	[1]	[2]	제안한 알고리즘
iteration	O(2m)	O(2m-2)	O(2m-2)
control	$O(\log_2(m+1))$, zero check	$O(\log_2(m+1))$, zero check	O(1)
operation	division	division	division

5. 결 론

본 논문에서는 유한체 역수(B^{-1} mod F) 및 나눗셈(A/B mod F)을 계산 할 수 있는 새로운 알고리즘 및 하드웨어를 제안하였다. 제안한 알고리즘은 나눗셈 연산을 위해 2m-2의 반복적인 연산을 필요로 한다. 또한 유한체 나눗셈과 곱셈의 제어를 위해 도입한 G와 H의 새로운 유한체 원소를 사용함으로써 입력 데이터에 독립적이며 2bit의 고정된 제어 신호만으로 제어할 수 있다. 따라서 입력 데이터에 독립적인 O(1) complexity의 하드웨어 리소스를 필요로 하므로 $O(\log_2(m+1))$ complexity의 하드웨어 리소스를 필요로 하는 [1][2]에 비해 보다 좋은 성능을 가진다.

참고문헌

[1] H. Brunner, A. Curiger and M. Hofstetter, "On Computing Multiplicative Inverses in GF(2^m)," IEEE Transactions on Computers, Vol. 42, No. 8, pp. 1010-1015, Aug. 1993.

[2] J. Guo and C. Wang, "Hardware-Efficient Systolic Architecture for Inversion and Division in GF(2^m)," IEE Proceedings Computers and Digital Techniques, Vol. 145, No. 4, July 1998.

[3] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverse in GF(2^m) using Normal Basis," J. Society for Electronics Communication, pp. 31-36, 1986.

[4] S. Morilka and Y. Katayama, "O($\log_2$m) Iterative Algorithm for Multiplicative Inversion in GF(2^m)," International Symposium on Information Theory, 2000 Proceedings, IEE, pp. 449, 2000.