개선된 확장 유클리드 알고리듬을 이용한 유한체 나눗셈 연산기의 하드웨어 설계

이 광호°, 강 민섭
안양대학교 컴퓨터공학과
mskang@aycc.anyang.ac.kr

Hardware Design of Finite Field Divider Using Modified Extended Euclidian Algorithm

K.H. Lee°, M.S. Kang
Dept. of Computer Engineering
Anyang University

요 약

본 논문에서는 GF(2^m) 상에서 나눗셈 연산을 위한 고속 알고리듬을 제안하고, 제안한 알고리듬을 기본으로 한 나눗셈 연산기의 하드웨어 설계 및 구현에 관하여 기술한다. 나눗셈을 위한 모듈러 연산은 개선된 이진 확장 유클리드 알고리듬 (Binary Extended Euclidian algorithm) 을 기본으로 하고 있다. 성능 비교 결과로부터 제안한 방법은 기존 방법에 비해 지연시간이 약 26.7 % 정도 개선됨을 확인하였다.

1. 서 론

Galois Field GF(2^m) 상의 산술 연산은 코딩 이론, 이산대수, 암호 시스템 등과 같이 다양한 분야에 적용되고 있다. 오늘날 많이 사용하고 있는 RSA 와 Elliptic Curve Cryptosystem(ECC) 와 같은 암호시스템 효율적인 나눗셈 연산은 매우 중요한 역할을 하고 있다. 특히, ECC 알고리듬에 있어서 확장 유클리드 알고리듬은 역수와 나눗셈 연산을 수행하는데 많이 이용되고 있다[1-3]. 기존의 유클리드 알고리듬을 기본으로 하여 역수와 나눗셈 연산을 위한 많은 연구가 수행되었다[2-4]. 비록 GF(2^m)상에서 역수와 나눗셈 연산을 위한 효율적인 이진 확장 최대공약수 (Binary Extended Greatest Common Divisor (GCD) 알고리듬[2]이 발표되었지만, 하드웨어로 구현이 어렵다는 문제점이 있다[4]. 이러한 문제를 해결하기 위해서 [4]에서 제안된 알고리듬은 VLSI 구현에 적합하지만, 이 알고리듬은 최종 나눗셈 결과를 얻는데 2m-1 번 (m 은 iteration time 을 나타냄)의 클럭이 소요된다[4].

본 논문에서는 GF(2^m) 상에서 나눗셈 연산을 위한 고속 알고리듬을 제안하고, 제안한 알고리듬을 기본으로 한 나눗셈 연산기의 하드웨어 설계 및 구현에 관하여 기술한다. 나눗셈을 위한 모듈러 연산은 개선된 이진 확장 유클리드 알고리듬을 기본으로 하고 있다.

2. 확장 유클리드 알고리듬[2]

GF(2^m)상의 두 원소 A(x), B(x)를 다항식으로 표현되며, G(x)는 차수 m 에서의 기약다항식이고, B(x) ≠ 0 일 때, P(x)는 A(x)/B(x) mod G(x)의 결과를 나타낸다.

그림 1 은 나눗셈 결과 P(x)를 얻기 위한 이진확장 GCD 을 나타낸다[2].

```
Input : G(x), A(x), B(x)
Output : U  has P(x) = A(x) / B(x) mod G(x)
Initialization : R = B(x), S =G= G(x), U = A(x), V = 0
while S ≠ 0 do
(1)  while (r0 == 0) do
          R = R / x
          if u0 == 0 then U = U / x
          else U = (U + G) / x; end if
     end while
(2)  while (s0 == 0) do
          S = S / x
          if v0 == 0 then V = V / x
          else V = (V + G) / x; end if
     end while
(3)  if  S ≥ R then
          (S, R)=(S+R, R), (V, U)=(U+V, U);
     else
          (S, R)=(S, S+R), (V, U) = (V, U+V);
     end if
end while
```

그림 1 이진 확장 GCD 알고리듬[2]

그림 1 의 알고리듬은 외부 while loop 내에서 세 개의 Step 으로 구성되었다. 첫 번째 클럭은, 세 개의 레지스터를 초기화하고, Step(1)에서는 제어비트 r_0 와 u_0를 사용하여 (R, U) 값을 갱신한다. 그리고 다시 제어비트 r_0를 검사하게 되고, (R, U) 값은 r_0가 0 이 될 때까지 갱신되고, 값이 갱신될 때 마다 하나의 클럭이 소요된다. 결과적으로 반복횟수만큼의 클럭이 사용됨을 알

†본 연구는 중소기업청 "2005 년도 산학연 공동기술개발 컨소시엄사업" 과 IDEC 지원으로 수행되었음.

수 있다. Step (2)도 Step (1)과 같은 처리과정을 거쳐 (S, V)를 구한다. 마지막으로, Step (3)에서는 하나의 클럭을 소요한다.

제안한 알고리듬과 기존의 두 알고리듬을 비교하기 위하여 [3, 4]]에서 제공된 입력의 데이터를 사용하였다. 표 1 은 GF(2^4)상의 나눗셈 연산 연산 과정을 나타내며, m=4, $G(x)=x^4+x+1$, $A(x)=x^3+x^2+x$, $B(x)=x^3+x+1$ 의 데이터를 입력으로 사용하였다.

표 1 그림 1의 알고리듬에 대한 연산 과정

It	Step	Clk	S (=G(x))	R (=B(x))	V (=0)	U (=A(x))
1	(1)	1	X^4+x+1	x^3+x+1	0	x^3+x^2+x
	(2)	2	X^4+x+1	x^3+x+1	0	x^3+x^2+x
	(3)	3	x^4+x^3	x^3+x+1	x^3+x^2+1	x^3+x^2+x
2	(1)	4	x^4+x^3	x^3+x+1	x^3+x^2+1	x^3+x^2+x
	(2)-1	5	x^3+x^2	x^3+x+1	x^2+x+1	x^3+x^2+x
	(2)-2	6	x^2+x	x^3+x+1	x^3+x	x^3+x^2+x
	(2)-3	7	x+1	x^3+x+1	x^2+1	x^3+x^2+x
	(3)	8	x+1	x^3	x^2+1	x^3+x+1
3	(1)-1	9	x+1	x^2	x^2+1	X^3+x^2
	(1)-2	10	x+1	x	x^2+1	x^2+x
	(1)-3	11	x+1	1	x^2+1	x+1
	(2)	12	x+1	1	x^2+1	x+1
	(3)	13	x	1	x^2+1	x+1
4	(1)	14	x	1	x+1	x+1
	(2)	15	1	1	x+1	x+1
	(3)	16	0	1	0	x+1

표 1 에서 알 수 있듯이 S = G(x), R = B(x), U = A(x), V = 0 으로 초기화 되었고, It 는 바깥의 while loop 의 반복 횟수를, Clk 는 연산에 사용된 클럭을 나타낸다. 첫번째 It 에서는 Step (1), (2), (3)을 수행하는데 총 3 클럭이 사용되었다. 두 번째 It 의 경우 Step (2)는 while 루프를 3 번 반복하면서 3 클럭이 사용되었기 때문에 이 It 에서는 총 5 클럭이 사용되었다. 이 알고리듬은 4 번째 It 이후에 종료가 되며, 나눗셈의 결과 U=x+1 을 얻는데 총 16 클럭이 사용되었다.

3. 개선된 확장 유클리드 알고리듬

유한체 연산에 있어서 역원연산과 나눗셈 연산은 가장 많은 시간과 공간을 사용하는 부분이다.

```
Input : G(x), A(x), B(x)
Output : U  has P(x) = A(x) / B(x) mod G(x)
Initialization : R = B(x), S =G= G(x), U = A(x), V = 0
while S ≠ 0 do
(1)    while (r0 == 0) or (s0 == 0) do
          if r0 == 0 then
             R = R / x, U = (U + u0·G) / x;
          end if
          if s0 == 0 then
           S = S / x, V = (V + v0·G) / x;
          end if
       end while
(2)    if R > S then
          R = S + R, U = U + V;
       else
          S = S + R, V = U + V;
       end if
end while
```

그림 2 제안하는 유한체 나눗셈 알고리듬

GF(2^m) 상의 연산 속도향상을 위하여 여기에서는 [2]에서 제안된 확장 유클리드 알고리듬을 기반으로 한 고속 나눗셈 연산 알고리듬을 제안한다.
그림 2 는 GF(2^m) 상에서의 제안하는 나눗셈 알고리듬을 나타낸다.

제안하는 알고리듬은 두 개의 Step 으로 구성되며, [2]에서 제안한 알고리듬과 비교할 때 하나의 while 구문을 제거된 구조를 갖는다. GCD 연산은 Step (1) 에서 parity 비트 r_0 와 s_0 이 각각 0 인지 1 인지를 검사함으로써 시작되며, 이때 한 개의 클럭이 사용된다. Step (1)에서는 $r_0=s_0=0$ 인 조건일 때 값 (R, S)가 갱신된다. 다시 Step(1)에서는 r_0 와 s_0 를 검사하게 되고, 이 과정에서 반복 횟수 만큼 클럭 이 사용된다. Step (1)이 완료되면 Step (2)에서는 GCD(S, R)과 GCD(V, U)의 갱신을 위해 하나의 클럭이 필요하게 된다. 표 2 는 제안하는 알고리듬의 GF(2^4)상에서의 연산 과정을 나타내며, 표 1 에서 사용된 동일한 파라메터를 사용하였다.

표 2 제안하는 알고리듬의 연산 과정

It	Step	Clk	S (=G(x))	R (=B(x))	V (=0)	U (=A(x))
1	(1)	1	x^4+x+1	x^3+x+1	0	x^3+x^2+x
	(2)	2	x^4+x^3	x^3+x+1	x^3+x^2+x	x^3+x^2+x
2	(1)-1	3	x^3+x^2	x^3+x+1	x^2+x+1	x^3+x^2+x
	(1)-2	4	x^2+x	x^3+x+1	x^3+x	x^3+x^2+x
	(1)-3	5	x+1	x^3+x+1	x^2+1	x^3+x^2+x
	(2)	6	x+1	x^3	x^2+1	x^3+x+1
3	(1)-1	7	x+1	x^2	x^2+1	x^3+x^2
	(1)-2	8	x+1	X	x^2+1	x^2+x
	(1)-3	9	x+1	1	x^2+1	x+1
	(2)	10	x	1	x^2+x	x+1
4	(1)	11	1	1	x+1	x+1
	(2)	12	0	1	0	x+1

표 2 에서 알 수 있듯이, 첫 번째 반복 루프에서 각 Step 마다 하나의 클럭이 사용되었다. 두 번째 루프의 Step (1)에서는 while 루프가 3 번 반복하므로 3 개의 클럭이 사용되고, Step (2)에서 하나의 클럭이 사용되었다.

제안하는 알고리듬은 4 번의 반복 후에 나눗셈 결과인 U=x+1 를 얻게 되며, 이를 위해 12 개의 클럭이 사용되었다. 제안하는 알고리듬은 최대로 3m 의 클럭을 필요로 하며, m 번의 루프가 반복된 후에 최종 나눗셈 결과를 얻게 된다.
그리하여, 표 1 의 나눗셈 결과를 얻는데 기존의 알고리듬[2]은 적어도 15 클럭이 필요한 반면에, 제안하는 알고리듬은 12 클럭이면 충분하다.

4. 하드웨어 설계 및 구현 결과

4.1 하드웨어 설계

이 장에서는 제안하는 유한체 나눗셈 알고리듬을 기반으로 새로운 GF(2^m)상의 나눗셈을 위한 연산기 구조를 설계한다. 그림 3 은 제안하는 고속 유한체 알고리듬을 기본으로 한 나눗셈 연산기의 구조를 나타낸다.

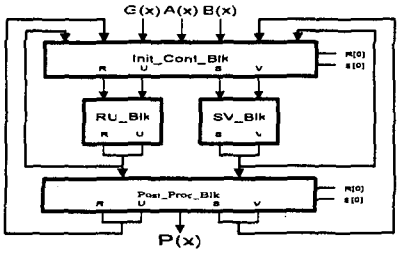

그림 3 제안한 나눗셈 연산기의 구조

그림 3 은 G(x), A(x), 그리고 B(x)의 세 개의 입력을 가지며, " A(x) /B(x) mod G(x)" 의 연산 결과인 P(x)를 얻는다. RU_Blk 과 SV_Blk 모듈은 제안하는 알고리듬의 Step (1)의 if 구문에서 수행된다. 이 두 모듈은 병렬 처리가 가능하므로, RU_Blk 에서는 (U, R), SV_Blk 에서는 (V, S)를 단지 하나의 클럭을 사용하여 각각의 parity 비트의 상태에 따라 처리하게 된다.

그리고 Post_Proc_Blk 는 Step(2)의 연산을 수행하게 된다. 이 모듈은 이전 Step 에서 계산된 네 개의 파라메터들을 비교, 덧셈 연산을 통하여 나눗셈의 연산 결과를 얻는다.

하드웨어 구현에 있어서, 종래의 알고리듬[2]은 Step (1)의 루프에서는 (U, R)을 계산하고, Step (2)의 루프에서는 (S, V)를 계산하기 때문에 2 개의 클럭이 소요되어 많은 처리 시간이 필요하다. 그러나, 제안하는 알고리듬에서는 하나의 클럭로 두 개의 블록이 제어되므로 병렬처리(concurrent processing)가 가능하게 된다.

4.2 구현 결과

제안한 알고리듬은 Verilog HDL 로 기술하였고, 하드웨어 구현을 위하여 Xilinx VirtexII XC2V-8000 FPGA 디바이스를 타겟으로 합성을 수행하였다. 또한, 설계된 회로의 검증을 위한 시뮬레이션은 ISE 6.x 와 Mentor Graphics 사의 ModelSim 을 사용하였다. 그림 4 는 제안한 알고리듬을 기본으로 하여 설계된 m = 8 인 연산기의 타이밍 시뮬레이션 결과를 나타낸다.

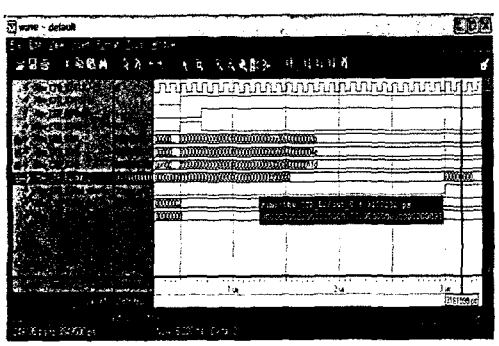

그림 4 타이밍 시뮬레이션 결과

시뮬레이션에 사용된 입력 파라메터는 $A(x)=x^5+x^3+x+1$, $B(x)=x^6+x^3+x^2+x$, $G(x)=x^8+x^4+x^3+x^2+1$ 이다. 시뮬레이션 결과에서 알 수 있듯이 3m 개의 클럭 사이클 후에 시뮬레이션 결과인 $P(x)=x^7+x^4+x^2+1$ 을 얻을 수 있었다.

표 3 은 기존 알고리듬[2]과 제안한 알고리듬의 성능평가를 위한 실험결과를 나타낸다.

표 3 두 알고리듬의 성능 비교 결과

항목 / 알고리듬		기존 방법[2]	제안한 방법
사이클 수	for m=8	8	8
지연시간(ns)		33	23
사이클 수	for m=16	55	42
지연시간(ns)		693	525
사이클 수	for m=32	92	68
지연시간(ns)		1159	850

공정한 비교를 위하여, 저자들이 직접 기존의 알고리듬[2]을 Verilog HDL 로 설계하였으며, 또한, 합성 및 시뮬레이션을 수행하였다. 표 3 에서 사이클 수는 최종 결과가 나오는데 필요한 클럭 수를, 전체지연은 최종 결과가 나오는데 걸린 시간을 나타내낸다. 또한, m 은 $GF(2^m)$의 지수를 나타내며, m 값을 다르게 하여 수행한 시뮬레이션 한 결과를 나타낸다. 표 3 의 사이클 수의 비교에서 M=16 과 M=32 인 경우 제안된 방법은 기존 방법에 비해 각각 23.6%와 26.1% 감소하였다. 지연시간의 경우는 각각 24.2% 와 26.7 %가 개선됨을 확인하였다.

5. 결 론

본 논문에서는 기존의 확장 유클리드 연산 알고리듬을 기반으로 하여 개선된 유한체 나눗셈 연산 알고리듬을 제안하였다. 제안한 알고리듬은 Verilog HDL 로 기술하였고, 설계된 나눗셈 연산기의 검증은 ModelSim 을 이용하였다. 시뮬레이션 결과로부터 설계된 구조가 정확히 동작함을 확인하였다. 또한, 지연시간의 성능 비교를 통하여 제안한 알고리듬이 기존의 알고리듬[2] 보다 약 26.7 % 정도 개선되었다.

6. 참고 문헌

[1] Stallings, William. Cryptography and Network Security: Principles and Practice, 2nd Edition. New Jersey: Prentice Hall Inc., 1999.

[2] Knuth, D. E.: ' The art of computer programming: seminumerical algorithms' (Addison-Wesley, 3rd ed. Reading, MA, 1998)

[3] J. Guo, and C. Wang, " Systolic Array Implementation of Euclidian' s Algorithm for Inversion and Division in GF, " IEEE Trans. Computers, Vol. 47, No. 10, pp. 1161-1167, Oct. 1998.

[4] C. H. Kim, S. Kwon, J. J Kim, and C. P. Hong, " A Compact and Fast Division Architecture for a *Finite Field*," *ICCSA2003, LNCS, Vol. 2667,* pp.855 - 864, Aug. 20.