확장 이진 GCD 알고리듬을 이용한 개선된 유한체 나눗셈 연산기의 FPGA 설계

박지원*, 강민섭
*안양대학교 컴퓨터공학과
e-mail : mskang@anyang.ac.kr

FPGA Design of Modified Finite Field Divider Using Extended Binary GCD Algorithm

Ji-Won Park*, Min-Sup Kang
*Dept. of Computer Science Engineering, Anyang University

요 약

본 논문에서는 확장 이진 최대공약수 알고리듬 (Extended Binary GCD algorithm)을 기본으로 GF(2^m) 상에서 유한체 나눗셈 연산을 위한 고속 알고리듬을 제안하고, 제안한 알고리듬을 기본으로 한 나눗셈 연산기의 FPGA 설계 구현에 관하여 기술한다. 제안한 알고리듬은 Verilog HDL 로 기술하였고, Xilinx FPGA virtex4-xc4vlx15 디바이스를 타겟으로 하였다.

1. 서론

최근 유무선 통신기술의 발달로 정보 보안은 아주 중요한 문제로 인식되고 있다. 정보 보안에 필요한 암호화 알고리듬은 키 사용 방법에 따라 공통키(common key encryption) 방법과 공개키(public key encryption) 방법으로 나눌 수 있다. 오늘날 많이 사용하고 있는 암호 시스템은 RSA 와 Elliptic Curve Crypto system(ECC)이다. ECC 암호 시스템은 다른 암호 시스템에 비해 짧은 비트 길이의 키 값을 사용하면서 안전도가 동일한 장점을 가진다. ECC 에 사용되는 유한체는 GF(p), GF(p^m), GF(2^m)이 있다(여기서 p 는 소수이다). 이중 GF(2^m)은 0 과 1 을 원소로 갖는 GF(2)의 m 차원 확장 필드로 특히 하드웨어 구현에 적합하다[4]. ECC 암호 시스템의 연산은 기본적으로 유한체 연산을 포함하고 있으면 이 연산에 따라 암호 시스템의 효율성이 결정된다[5]. 따라서 유한체 GF(2^m)상에서 정의된 덧셈, 뺄셈, 곱셈, 나눗셈 연산은 매우 중요하다. 이중 덧셈, 뺄셈은 XOR 연산으로 간단히 연산이 가능하지만 곱셈, 나눗셈 연산은 매우 복잡한 연산 구조를 이루고 있으며 특히 나눗셈 연산은 곱셈 연산보다 복잡한 구조를 이루고 있어 효율적인 나눗셈 연산은 이와 같은 암호화 시스템에 매우 중요한 역할을 하고 있다[6]. 유한체 GF(2^m)상에서 나눗셈 연산을 수행하기 위해서는 확장 유클리드 알고리듬(Extended Euclidean Algorithm(EEA)과 확장 이진 최대공약수 알고리듬(Extended Binary GCD Algorithm(EBGA) 등이 현재까지 사용되어 왔다. 이중 확장 이진 GCD 알고리듬은 나눗셈과 곱셈 연산을 시프트 연산으로 대체함으로써 확장 유클리드 알고리듬보다 좋은 성능을 보여준다.

본 논문에서는 확장 이진 GCD 알고리듬을 이용하여 GF(2^m) 상에서 나눗셈 연산을 위한 고속 알고리듬을 제안하고, 제안한 알고리듬을 기본으로 한 나눗셈 연산기의 FPGA 설계 및 구현에 관하여 기술한다.

2. 확장 이진 최대공약수 알고리듬[2]

GF(2^m)상의 두 원소 A(x), B(x)는 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 확장 이진 GCD 알고리듬을 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) 확장 이진 GCD 알고리듬[2]

(그림 1)의 알고리듬은 외부 while loop 내에서 세

개의 Step 으로 구성 되었다. 첫 번째 클럭에서 네 개의 레지스터를 초기화 하고, Step(1)에서는 제어비트 r_0 가 0 이 될 때까지 (R, U)를 연산하고, 값이 연산될 때마다 하나의 클럭이 사용된다. 결과적으로 반복횟수만큼의 클럭이 사용됨을 알 수 있다. Step(2)에서는 제어비트 s_0 가 0 이 될 때까지 (S, V)가 연산한다. 마지막으로, Step(3)에서는 R 과 S 의 크기를 비교하여 (R, U)와 (S, V)를 연산하며 하나의 클럭을 사용한다.

<표 1>은 (그림 1)에 나타난 기존 알고리듬[2]의 연산 과정을 나타낸다.

<표 1> 기존 알고리듬[2]의 연산 과정

#Clk	Step	S =(G(x))	R =(B(x))	V =(0)	U =(A(x))
1	init	x^4+x+1	x^3+x+1	0	x^3+x^2+x
2	(1)	x^4+x+1	x^3+x+1	0	x^3+x^2+x
3	(2)	x^4+x+1	x^3+x+1	0	x^3+x^2+x
4	(3)	x^4+x^3	x^3+x+1	x^3+x^2+1	x^3+x^2+x
5	(1)	x^4+x^3	x^3+x+1	x^3+x^2+1	x^3+x^2+x
6	(2)-1	x^3+x^2	x^3+x+1	x^2+x+1	x^3+x^2+x
7	(2)-2	x^2+x	x^3+x+1	x^3+x	x^3+x^2+x
8	(2)-3	x+1	x^3+x+1	x^2+1	x^3+x^2+x
9	(3)	x+1	x^3	x^2+1	x^3+x+1
10	(1)-1	x+1	x^2	x^2+1	X^3+x^2
11	(1)-2	x+1	x	x^2+1	x^2+x
12	(1)-3	x+1	1	x^2+1	x+1
13	(2)	x+1	1	x^2+1	x+1
14	(3)	x	1	x^2+1	x+1
15	(1)	x	1	x+1	x+1
16	(2)	1	1	x+1	x+1
17	(3)	0	1	0	x+1

제안한 알고리듬과 기존 알고리듬[2]을 비교하기 위하여 [3,4]에서 제공된 데이터(m=4, $G(x)=x^4+x+1$, $A(x)=x^3+x^2+x$, $B(x)=x^3+x+1$)를 입력으로 사용하였다. <표 1>에서 알 수 있듯이, S=G(x), R=B(x), U=A(x), V=0 으로 초기화 되었고, #Clk 는 연산에 사용된 클럭을 나타낸다. 외부 while loop 의 첫 번째 반복에서 네 개의 레지스터가 초기화 되었고 두 번째 반복에서 Step (1), (2), (3)을 수행하는데 총 3 클럭이 사용되었다. 세 번째 반복에서는 Step (2) while loop 가 3 번 반복 되면서 3 클럭이 사용되었기 때문에 총 5 클럭이 사용되었다. 이 알고리듬은 외부 while loop 가 5 번 반복되고 종료되며, 나눗셈 결과 U=x+1 을 구하는데 총 17 클럭이 사용되었다.

3. 개선된 유한체 나눗셈 알고리듬

본 논문에서는 $GF(2^m)$ 상의 연산 속도 향상을 위하여 (그림 1)에서 제안된 확장 이진 GCD 알고리듬을 기반으로 고속 유한체 나눗셈 연산 알고리듬을 제안한다. (그림 2)는 제안하는 고속 유한체 나눗셈 알고리듬을 나타낸다.

```
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
(S1)    if r0 == 0 or s0 == 0 do
(1)-1     if r0 == 0 then
              R = R / x;
              U = (U + u0·G) / x;
          end if
(1)-2     if s0 == 0 then
              S = S / x;
              V = (V + v0·G) / x;
          end if
(S2)    else
(2)-1     if R > S then
              R = R + S;
              U = U + V;
(2)-2     if S >= R then
              S = S + R;
              V = V + U;
          end if
        end if
end while
```

(그림 2) 제안하는 개선된 유한체 나눗셈 알고리듬

제안하는 알고리듬은 두 개의 Step 으로 구성되며, (그림 1)에서 제안된 알고리듬과 비교할 때 두 개의 내부 while loop 가 제거된 구조를 갖는다. 첫 번째 클럭에서 네 개의 레지스터를 초기화 하고, GCD 연산은 Step(1)에서 제어비트 r_0 와 s_0 가 각각 0 또는 1 인지를 검사함으로써 시작된다. Step(1)에서는 $r_0=0$ or $s_0=0$ 인 조건일 때 값 (R, U)와 (S, V)를 연산하고 다음 while loop 반복에서 Step(1)을 $r_0=1$ and $s_0=1$ 이 될 때까지 수행하고, 반복 횟수만큼 클럭을 사용하게 된다. $r_0=1$ and $s_0=1$ 이 되면 Step(2)에서 (S, R)과 (V, U)의 연산을 위하여 하나의 클럭을 사용하게 된다. <표 2>는 제안하는 알고리듬의 $GF(2^4)$상에서의 연산 과정을 나타내며, <표 1>에서 사용된 것과 동일한 입력을 사용하였다.

<표 2> 제안하는 알고리듬의 연산 과정

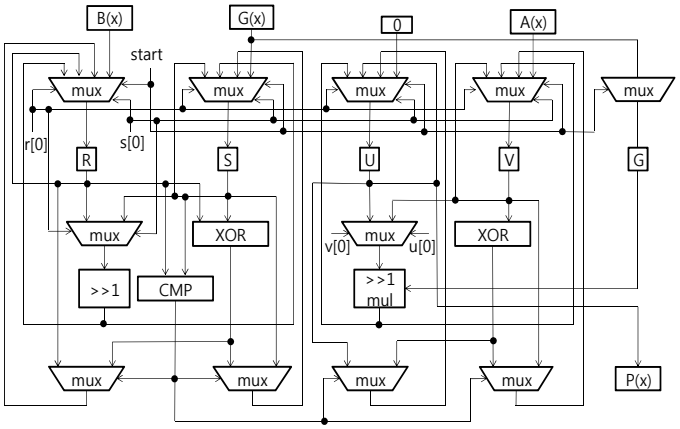
#Clk	Step	S =(G(x))	R =(B(x))	V =(0)	U =(A(x))
1	init	x^4+x+1	x^3+x+1	0	x^3+x^2+x
2	(2)-2	x^4+x^3	x^3+x+1	x^3+x^2+x	x^3+x^2+x
3	(1)-2	x^3+x^2	x^3+x+1	x^2+x+1	x^3+x^2+x
4	(1)-2	x^2+x	x^3+x+1	x^3+x	x^3+x^2+x
5	(1)-2	x+1	x^3+x+1	x^2+1	x^3+x^2+x
6	(2)-1	x+1	x^3	x^2+1	x^3+x+1
7	(1)-1	x+1	x^2	x^2+1	x^3+x^2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

<표 2>에서 알 수 있듯이, 각 반복 루프마다 하나의 클럭이 사용되므로 나눗셈의 결과 U=x+1 을 구하는데 총 12 클럭이 필요하다. 제안하는 알고리듬은 단일 while loop 를 사용함으로써 최대 3m 의 클럭을 사용하게 된다. 따라서, 제안한 알고리듬은 기존 이진 확장 GCD 알고리듬 보다 실제 연산에 필요한 클럭의

수를 줄임으로써 고속 연산이 가능하게 된다.

4. 하드웨어 설계 및 구현결과

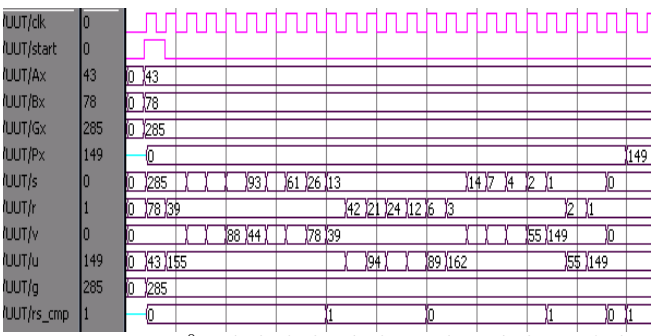
이 장에서는 제안하는 개선된 유한체 나눗셈 알고리듬을 기반으로 새로운 GF(2^m)상의 나눗셈을 위한 연산기 구조를 설계한다. (그림 3)은 제안하는 고속 유한체 알고리듬을 기본으로 한 나눗셈 연산기의 구조를 나타낸다.



(그림 3) 제안한 유한체 나눗셈 연산기의 구조

(그림 3)은 B(x), G(x), 그리고 A(x)의 세 개의 입력을 가지며, " A(x)/B(x) mod G(x)"의 연산 결과인 P(x)를 얻는다. 제안한 알고리듬의 if 구문은 각각의 mux 블록에서 수행된다. 하드웨어 구현에 있어서, 기존의 알고리듬[2]은 Step(1)의 loop 에서 (R, U)를 연산하고, Step(2)의 loop 에서 (S, V) 연산을 처리하기 때문에 2 개의 클럭이 사용되어 많은 처리 시간이 필요하다. 그러나, 제안된 알고리듬에서는 r_0 와 s_0 각각의 제어 비트의 상태에 따라 각 mux 블록이 공용 클럭을 사용하여 처리 되므로 각 레지스터의 갱신이 하나의 클럭으로 처리된다.

제안한 알고리듬은 Verilog HDL 로 기술하였고, 하드웨어 구현을 위하여 Xilinx FPGA virtex4-xc4vlx15 디바이스를 타겟으로 합성을 수행하였다. 또한, 설계된 회로의 검증을 위한 시뮬레이션은 ISE 10.1 과 Mentor Graphics 사의 ModelSim 을 사용하였고, 본 연구에서 사용된 툴들은 KAIST, IDEC 에서 지원되었다. (그림 4)는 제안한 알고리듬을 기본으로 하여 설계된 m=8 인 연산기의 타이밍 시뮬레이션 결과를 나타낸다.



(그림 4) GF(2^8) 연산기의 타이밍 시뮬레이션 결과

시뮬레이션에 사용된 입력 매개변수는 $A(x)=x^5+x^3+x+1$, $B(x)=x^6+x^3+x^2+x$, $G(x)=x^8+x^4+x^3+x^2+1$ 이다. 시뮬레이션 결과에서도 알 수 있듯이 3m 클럭인 24 클럭 후에 시뮬레이션 결과인 $P(x)=x^7+x^4+x^2+1$(decimal 149)를 얻을 수 있다. <표 3>은 기존 알고리듬[2,7]과 제안한 알고리듬의 성능평가를 위한 m=163 인 연산기의 합성결과를 나타낸다.

<표 3> 알고리듬의 성능평가 결과

항목 \ 알고리듬	기존 방법[2]	기존 방법[7]	제안한 방법
Min. Period(ns)	8.584	8.578	6.824
No. of Flip Flops	981	981	832
No. of Slices	1775	1590	904

세 알고리듬의 비교를 위하여 기존의 알고리듬[2,7]을 Verilog HDL 로 직접 설계하였으며, 합성 및 시뮬레이션을 수행하였다. <표 3>에서 Min. Period 는 최종 결과가 나오는데 걸린 지연시간이며, 기존 알고리듬보다 약 20% 개선되었다. 또한 하드웨어 요구량 비교에서도 약 43% 정도 감소되었다. 세 알고리듬의 성능평가를 통하여 제안한 방법이 기존 방법[2, 7]보다 성능이 개선됨을 확인하였다.

5. 결 론

본 논문에서는 기존의 확장 이진 GCD 알고리듬을 기반으로 하여 개선된 유한체 나눗셈 연산 알고리듬을 제안하였다. 제안한 알고리듬은 Verilog HDL 로 기술하였고, 설계 검증은 ModelSim 을 사용하였다. 시뮬레이션 결과로부터 설계된 구조가 정확히 동작함을 확인 하였다. 또한 알고리듬의 성능평가에서 제안한 알고리듬이 기존 알고리듬[2,7] 보다 개선되었다.

참고문헌

[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, and C. P. Hong, "A New Arithmetic Unit Over GF(2^m) for Low-Area Elliptic Curve Cryptographic Processor" 03-7 Vol.28 No.7A

[5] S. S. Kim, N. S. Chang, and C. H. Kim, "A Fast Inversion for Low-Complexity System over GF(2^m)" 2005-42SD-9-7

[6] B. C. Jeon,, J. S. Bae, S. J. Park, M. S. Kang, "Hardware Deisgn of Advanced Finite Field Divider" 2009

[7] K. H. Lee, M. S. Kang, "Hardware Design of Finite Field Divider Using Modified Extended Euclidian Algorithm", 2005 Vol.32, No. 2(I)