유한체 GF(2^m)의 응용을 위한 새로운 나눗셈 회로

김 창 훈[†] · 이 남 곤[††] · 권 순 학[†††] · 홍 춘 표[††††]

요 약

본 논문에서는 유한체 GF(2^m)의 응용을 위한 새로운 비트-시리얼 나눗셈 회로를 제안한다. 제안된 나눗셈 회로는 수정된 바이너리 최대 공약수 알고리즘에 기반하며, $2m$-1 클락 사이클 비율로 나눗셈 결과를 출력한다. 본 연구에서 제안된 회로는 기존의 비트-시리얼 나눗셈 회로에 비해 속도에서 43%, 칩 면적에서 20%의 성능 개선을 보인다. 또한 제안된 회로는 기약다항식의 선택에 있어 어떠한 제약 조건도 두지 않을 뿐 아니라 매우 규칙적이고 모듈화 하기 쉽기 때문에 필드 크기 m에 대해 높은 유연성 및 확장성을 제공한다. 따라서 본 논문에서 제안된 나눗셈 회로는 저면적을 요구하는 GF(2^m)의 응용에 매우 적합하다.

키워드 : 유한체 나눗셈, 바이너리 GCD, 시리얼 구조, VLSI

New Division Circuit for GF(2^m) Applications

Chang Hoon Kim[†] · Nam Gon Lee[††] · Soonhak Kwon[†††] · Chun Pyo Hong[††††]

ABSTRACT

In this paper, we propose a new division circuit for GF(2^m) applications. The proposed division circuit is based on a modified the binary GCD algorithm, and produce division results at a rate of one per $2m$-1 clock cycles. Analysis shows that the proposed circuit gives 47% and 20% improvements in terms of speed and hardware respectively. In addition, since the proposed circuit does not restrict the choice of irreducible polynomials and has regularity and modularity, it provides a high flexibility and scalability with respect to the field size m. Thus, the proposed divider is well suited to low-area GF(2^m) applications.

Key Words : Finite Field Division, Elliptic Curve Cryptosystem, Binary GCD, VLSI

1. 서 론

최근 유한체 연산은 오류제어 코드, 암호 시스템등 다양한 분야에 응용되고 있다. 이러한 응용에 사용되는 유한체는 GF(p), GF(p^m) 그리고 GF(2^m)이 있다 (여기서 p는 소수이다). 이중 GF(2^m)은 0과 1을 원소로 갖는 GF(2)의 m차원 확장 체로 특히 하드웨어 구현에 적합하다. 또한 가장 대표적인 GF(2^m)의 원소 표기법에는 정규기저(normal basis) 및 표준기저(standard basis) 표기법이 있다. 각 기저표기법은 장단점을 가지는데, 정규기저 표기법을 사용할 경우 제곱연산이 쉽게되는 반면 곱셈연산이 매우 복잡하며 하드웨어구조가 규칙적이지 못하고 모듈성이 떨어지기 때문에 하드웨어 구현에는 표준기저 표기법이 더 많이 사용된다.

※ 이 논문은 2004년도 한국학술진흥재단의 지원에 의하여 연구되었음(KRF-2004-002-D00312)
† 준 회 원 : 대구대학교 컴퓨터공학과 박사과정
†† 정 회 원 : 대구대학교 정보통신공학과 석사과정
††† 정 회 원 : 성균관대학교 수학과 부교수
†††† 정 회 원 : 대구대학교 정보통신공학부 교수
논문접수 : 2004년 11월 11일, 심사완료 : 2005년 5월 18일

GF(2^m)상의 연산은 덧셈, 곱셈, 지수, 나눗셈이 있다. 여기서, 덧셈은 비트별 XOR 연산으로 쉽게 구현이 가능하지만, 나머지 연산들은 복잡하다. 특히 나눗셈 연산은 시간 및 공간 측면에서 가장 복잡하다. GF(2^m)상의 나눗셈 연산은 소프트웨어를 이용하여 쉽게 구현이 가능하지만 실시간 응용을 위해선 적합하지 않을 뿐 아니라 암호 시스템의 경우 사이드 채널 공격(타이밍 공격, 메모리 공격 등)에 약하다 [1]. 따라서 보다 향상된 성능 및 안전성을 제공하기 위해 나눗셈 연산기의 하드웨어 구현은 바람직하다.

GF(2^m)상의 나눗셈 연산을 하드웨어로 구현하기 위해 1) 선형방정식 집합의 해를 구하는 방법[4-5], 2) Fermat의 이론[6-7], 3) Euclid의 GCD 알고리즘[8-9]이 이용되어 왔다. 첫 번째 방법은 $2m$-1개의 미지수를 가진 $2m$-1개의 선형방정식들의 해를 구함으로써 역원을 찾아낸다. 두 번째 방법은 Fermat의 이론을 이용하여 연속적인 제곱과 곱셈을 함으로써 역원을 찾는다. 즉 $A/B = AB^{-1} = AB^{2^m-2} = A(B(B(B\cdots B(B(B)^2)^2\cdots)^2)^2)^2$ 의 관계식을 이용하는데, 이 방법은 m번의 제곱연산과 약 $\log_2{}^{(m-1)}$번의 곱셈연산을 필요로 한다.

마지막 방법은 Euclide의 GCD 알고리즘을 이용하는 방법으로 GCD($G(x)$, $B(x)$) = 1이라는 사실을 이용한다. 여기서 $B(x)$는 GF(2^m)상의 0이 아닌 원소이고 $G(x)$는 GF(2^m)을 정의하는 기약 다항식이다. 확장된 Euclide의 GCD 알고리즘은 $W(x) \cdot G(x) + U(x) \cdot B(x) = 1$을 만족하는 $W(x)$와 $U(x)$를 찾는다. 따라서 $U(x) \cdot B(x) \equiv 1 \bmod G(x)$이 되어 $U(x)$는 $B(x)$의 역원 즉, $B^{-1}(x)$가 된다. 최근 연구 결과에 의하면 표준 기저표기법을 사용할 경우 Euclide의 GCD 알고리즘을 사용했을 때 가장 적은 하드웨어의 사용으로 가장 낮은 계산 지연시간을 가진다[9].

이러한 나눗셈 회로들은 비트-패러럴[6-7] 혹은 비트-시리얼[4-5], [8-9] 구조를 가진다. 고속의 나눗셈 연산을 위해서는 비트-패러럴 구조가 적절하나 ECC에서 필드의 크기 m은 적어도 163 이상이어야 하기 때문에 비트-패러럴 구조는 높은 면적 복잡도와 핀 개수 문제 때문에 구현에 많은 어려움이 따를 뿐 아니라 비실용적이다. 따라서 ECC를 위해선 비트-시리얼 구조가 적합하다.

본 논문에서는 GF(2^m)상에서 표준 기저 표기법을 사용하여, 비트-시리얼 구조의 나눗셈기를 제안한다. 제안된 나눗셈기는 수정된 바이너리 GCD 알고리즘에 기반하며, 이 알고리즘으로부터 핵심 연산 및 제어함수를 설계한 후, 완전한 나눗셈 회로를 얻는다. 제안된 나눗셈기는 $O(m)$의 시간 복잡도와 $O(m)$의 공간 복잡도를 가지며, $2m$-1 사이클 마다 나눗셈 결과를 출력한다.

본 연구에서 제안된 나눗셈기를 기존의 나눗셈기들[8][9]과 비교 분석한 결과 칩 면적 및 계산 지연시간 모두에 있어 상당한 개선을 보였다. 따라서 제안된 나눗셈기는 적은 하드웨어를 사용하면서 고속으로 나눗셈 연산을 수행할 수 있기 때문에 GF(2^m)상의 나눗셈 연산기로 매우 적합하다. 또한 제안된 구조는 기약다항식 선택에 있어 어떤 제약도 두지 않고, 단 방향의 신호흐름을 가지면서, 매우 규칙적이기 때문에 필드크기 m에 대해 높은 유연성 및 확장성을 제공한다.

본 논문의 구성은 다음과 같다. 2절에서 VLSI 구현을 위한 GF(2^m)상의 새로운 나눗셈 알고리즘을 유도한다. 3절에서는 새로운 알고리즘에 기반 하여, 비트-시리얼 나눗셈기를 설계하고 FPGA를 이용하여 구현 및 그 기능을 검증한다. 4절에서 기존의 나눗셈 회로들과 성능을 비교 및 분석하고, 5절에서 결론을 맺는다.

2. VLSI 구현을 위한 GF(2^m)상의 새로운 나눗셈 알고리즘

$A(x)$와 $B(x)$는 GF(2^m)상의 두 원소이고, $G(x)$는 GF(2^m)을 정의하는, 즉 GF(2^m)≅GF(2)[x]/$G(x)$, 차수 m의 기약 다항식이고, $P(x)$는 $A(x)/B(x) \bmod G(x)$의 결과라고 하면, 각각의 다항식은 다음과 같이 표현되며 계수들은 이진수 0 혹은 1이다.

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0 \quad (1)$$

$$B(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1x + b_0 \quad (2)$$

$$G(x) = x^m + g_{m-1}x^{m-1} + \cdots + g_1x + g_0 \quad (3)$$

$$P(x) = p_{m-1}x^{m-1} + p_{m-2}x^{m-2} + \cdots + p_1x + p_0 \quad (4)$$

GF(2^m)상의 나눗셈을 위해 [알고리즘 1]은 사용될 수 있다. [알고리즘 1]은 GCD($G(x)$, $B(x)$) = 1 이라는 사실을 이용하며, 아래의 3가지 사실에 기반한다[10-11].

$$\text{If } R \text{ and } S \text{ are both even, then GCD}(S, R) = x\text{GCD}(S/x, R/x) \quad (5)$$

$$\text{If } R \text{ is even and } S \text{ is odd, then GCD}(S, R) = \text{GCD}(S, R/x) \quad (6)$$

$$\text{If } R \text{ and } S \text{ are both odd, then GCD}(S, R) = \text{GCD}((S-R)/x, R) \quad (7)$$

[알고리즘 1]에서 $B(x)$는 GF(2^m)상의 0이 아닌 원소이고 $W(x) \cdot G(x) + U(x) \cdot B(x) = 1$을 만족하는 $W(x)$와 $U(x)$를 찾으면, $U(x) \cdot B(x) \equiv 1 \bmod G(x)$가 되어 $U(x)$는 $B(x)$의 역원 즉, $B^{-1}(x)$이 된다. 이때 $U(x)$에 초기 값 1 대신 $A(x)$를 대입하면 알고리즘의 종료 후 나눗셈 결과 $P(x) = A(x)/B(x) \bmod G(x)$를 얻을 수 있다[9].

[알고리즘 1] GF(2^m)상의 바이너리 확장 GCD 알고리즘

Input: $G(x)$, $A(x)$, $B(x)$
Output: U has $P(x) = A(x)/B(x) \bmod G(x)$
Initialize: $R = B(x)$, $S = G = G(x)$, $U = A(x)$, $V = 0$
1. **while** $S \neq 0$ **do**
2. **while** $r_0 == 0$ **do**
3. $R = R/x$;
4. $U = U/x \bmod G$;
5. **end while**
6. **while** $s_0 == 0$ **do**
7. $S = S/x$;
8. $V = V/x \bmod G$;
9. **end while**
10. **if** $S \geq R$ **then**
11. $(S, R) = (S + R, R)$; $(V, U) = (U + V, U)$;
12. **else**
13. $(S, R) = (S, S + R)$; $(V, U) = (V, U + V)$;
14. **end if**
15. **end while**

[알고리즘 1]은 비록 간단하지만, 반복 회수가 고정되어있지 않을 뿐만 아니라 R과 S에 대한 비교 부분이 있기 때문에 VLSI 구현에 적합하지 않다. 따라서 우리는 이런 문제들을 해결하기 위해 식 (5)부터 식 (7)에 기술되어 있는 GCD의 사실을 이용하여 [알고리즘 1]을 수정한다. 수정된 결과는 [알고리즘 2]에 주어지며, 유도 과정은 아래와 같다.

1) [알고리즘 1]의 6단계부터 9단계의 제거 : [알고리즘 1]이 시작 할 때 S는 $G(x)$이고 R은 $B(x)$이기 때문에 s_0는 1이고 r_0는 1 혹은 0이고, 알고리즘이 종료 할 때 s_0는 0이고 r_0는 1이 된다. 또한 [알고리즘 1]이 시작 할 때 S는 항상 R보다 크기 때문에 r_0가 1이면 단계 11을 수행하고 r_0가 0이면 r_0가 1이 될 때까지 단계 3을 수행한 후 단계 11을 수행한다. 여기서 단계 11을 수행 할 때 $(S, R) = (S + R, R)$ 대신 $(R, S) = (S + R, R)$를 적용한다면, R과 S가 서로 바뀌기 때문에 알고리즘이 종료되면 S가 GCD(S, R)의 결과를 가진다. 즉 s_0는 항상 1이 된다. 이 조건을 적용할 경우, s_0는 알고리즘이 시작 할 때도 1이고 종료 될 때도 1이 된다. 따라서 알고리즘의 중간 단계에서 s_0를 항상 1로 유지한다면, [알고리즘 1]의 6단계부터 9단계는 제거 될 수 있다. 이 조건을 만족하기 위해서, 우리는 r_0에 따라 서로 다른 세 연산 식을 적용한다. 만약 r_0가 0이면, $(S, R) = (S, R/x)$을 적용하고, r_0가 1이면 [알고리즘 1]의 단계 10에 따라 $(S, R) = (R, (S + R)/x)$ 또는 $(S, R) = (S, (S + R)/x)$을 적용한다. 결과적으로, s_0는 항상 1이 되기 때문에 우리는 r_0가 0인지 1인 지만 체크하면 된다. 이 결과를 바탕으로 우리는 [알고리즘 2]에서 [알고리즘 1]의 6단계부터 9단계를 제거한다.

2) 비교단계의 제거 및 고정된 반복조건 : [알고리즘 1]이 시작할 때 S의 차수는 m이고 R의 차수는 기껏해야 m-1이다. 또한 위의 수정단계에서 r_0와 [알고리즘 1]의 단계 10에 따라 각 반 복 단계에 있어 (S, R)에 대하여 $(S, R/x)$ 또는 $(R, (S + R)/x)$ 또는 $(S, (S + R)/x)$의 연산이 수행된다. 따라서 각 반복에 있어 정확히 S 혹은 R의 차수를 1씩 감소시킨 다면, $2m$ 번의 반복 후 $(R = 0)$, $(S = 1)$이 된다. 또한 비교 단계를 없애기 위해 [알고리즘 1]과 위의 수정 과정을 살펴보면, 초기에 R의 차수가 n만큼 감소 된 후 r_0가 1이면 S가 R보다 크기 때문에 $(S, R) = (R, (S + R)/x)$을 적용한다. 다음 반복에 있어 우리는 새로운 R의 차수를 감소시킨다. 이때 r_0가 0이면 $(S, R) = (S, R/x)$을 적용한다. 여기서 새로운 R의 차수가 n만큼 감소되지 않은 상황에서 r_0가 1이면 R이 S보다 크기 때문에 [알고리즘 1]의 13단계로부터 $(S, R) = (S, (S + R)/x)$을 적용해야만 한다. 이를 구현하기 위해 우리는 새로운 변수 *count*와 *state*를 추가한다. [알고리즘 2]에 기술된 것처럼 우리는 *state*와 r_0에 따라 네 가지의 다른 조건을 적용한다: ① *state*가 0이고 r_0가 0이면 $(S, R) = (S, R/x)$을 적용하고 r_0가 1이 때까지 R의 차수를 1씩 감소시키고 *count*는 1씩 증가시킨다, ② *state*가 0이고 r_0가 1면, $(S, R) = (R, (S + R)/x)$을 적용한다, ③ *state*가 1이고 r_0가 0이면, $(S, R) = (S, R/x)$을 적용하고 *count*를 감소시킨다. 또한 *count*가 0이면, 초기의 반복 조건과 같기 때문에 *state*는 0으로 설정한다, ④ *state*가 1이고 r_0가 1이면, $(S, R) = (S, (S + R)/x)$을 적용한다. 위의 4가지 조건을 보면 $(S, R/x)$는 항상 적용되고 *state*와 r_0 따라 S 와 R은 바뀌기 때문에 우리는 각 반복에 있어 R과 S의 차수를 재귀적으로 1씩 감소시킬 수 있다. 따라서 $2m$ 반복 후 $R = 0$, $S = \text{GCD}(S, R) = 1$, *count* = 0 그리고 *state* = 0이 된다.

3) 나눗셈을 위한 수정된 알고리즘의 확장 : 나눗셈의 결과를 얻기 위해, 위에서 기술된 수정 결과와 함께 변수 U와 V를 추가하여 [10]에서 제안된 방법과 동일하게 확장하면 [알고리즘 2]를 얻을 수 있다. 지금까지의 수정 과정을 보면, 단지 R 과 S의 위치를 바꾸었을 뿐 식 (5)부터 식 (7)의 GCD 사실에는 위배되지 않았기 때문에 [알고리즘 2]는 $2m$ 반복 후 V는 정확한 나눗셈 결과 $P(x)=A(x)/B(x) \bmod G(x)$를 가진다.

[알고리즘 2]를 분석해보면 알고리즘의 마지막 반복을 시작할 때 *count*와 *state*는 항상 1이 된다는 사실을 알 수 있다. 따라서 V값에는 아무런 영향을 미치지 못하기 때문에 실제로 $2m$-1의 반복 후 우리는 정확한 나눗셈결과 V를 얻을 수 있다. 표 1에 [알고리즘 2]를 수행하는 한 예를 보인다. <표 1> 나타나듯이 $7(2m-1)$번 반복 후 V는 정확한 나눗셈 결과인 $x + 1$을 가진다.

[알고리즘 2] VLSI 구현을 위한 GF(2^m)상의 수정된 바이너리 확장 GCD 알고리즘

Input: $G(x)$, $A(x)$, $B(x)$
Output: V has $P(x)=A(x)/B(x) \bmod G(x)$
Initialize: $R=B(x)$, $S=G=G(x)$, $U=A(x)$, $V=0$,
count=0, *state*=0

```
1.  for i = 1 to 2m do
2.      if state == 0 then
3.          count = count+1;
4.          if r0 == 1 then
5.              (S, R)=(R, R+S); (V, U)=(U, U+V);
6.              state = 1;
7.          end if
8.      else
9.          count = count-1;
10.         if r0 == 1 then
11.             (S, R)=(S, R+S); (V, U)=(V, U+V);
12.         end if
13.         if count == 0 then
14.             state = 0;
15.         end if
16.     end if
17.     R = R/x;
18.     U = U/x;
19.     end if
20. end for
```

〈표 1〉 알고리즘 2에 기반 한 GF(24)상의 나눗셈 계산의 예

($G(x) = x^4 + x + 1$, $A(x) = x^3 + x^2 + x$, $B(x) = x^3 + x + 1$)

i	$state$	$count$	R	S	U	V
Init.	0	0	$x^3 + x + 1$	$x^4 + x + 1$	$x^3 + x^2 + x$	0
1	1	1	$x^3 + x^2$	$x^3 + x + 1$	$x^2 + x + 1$	$x^3 + x^2 + x$
2	0	0	$x^2 + x$	$x^3 + x + 1$	$x^3 + x$	$x^3 + x^2 + x$
3	0	1	$x + 1$	$x^3 + x + 1$	$x^2 + 1$	$x^3 + x^2 + x$
4	1	2	x^2	$x + 1$	$x^3 + x^2$	$x^2 + 1$
5	1	1	x	$x + 1$	$x^2 + x$	$x^2 + 1$
6	0	0	1	$x + 1$	$x + 1$	$x^2 + 1$
7	1	1	1	1	$x + 1$	$x + 1$
8	0	0	0	1	0	$x + 1$

3. GF(2^m)을 위한 새로운 나눗셈 회로의 설계

3.1 핵심 연산 및 제어함수

2절에서 제안된 알고리즘을 구현하기 전에 제안된 알고리즘의 핵심연산 및 제어함수에 대해서 고려한다. [알고리즘 2]의 S와 R은 차수가 m인 다항식 그리고 U와 V는 차수가 m-1인 다항식으로 각각 표현 할 수 있으며, 각 다항식은 아래와 같다.

$$R = r_m x^m + r_{m-1}x^{m-1} + \cdots + r_1 x + r_0 \quad (8)$$

$$S = s_m x^m + s_{m-1}x^{m-1} + \cdots + s_1 x + s_0 \quad (9)$$

$$U = u_{m-1}x^{m-1} + u_{m-2}x^{m-2} + \cdots + u_1 x + u_0 \quad (10)$$

$$V = v_{m-1}x^{m-1} + v_{m-2}x^{m-2} + \cdots + v_1 x + v_0 \quad (11)$$

[알고리즘 2]의 단계 5와 11에 나타나듯이 S와 V연산은 $state$와 r_0에 따른 간단한 교환 연산이다. 반면 R과 U는 두 개의 연산을 가진다. 우선 R의 연산을 살펴보면, r_0에 따라 $((S + R)/x)$ 또는 (R/x)를 계산해야한다. 따라서 우리는 아래와 같이 R의 임시적인 결과를 얻을 수 있다.

R'을 아래와 같이 R의 임시적인 결과라 두면

$$R' = r'_m x^m + r'_{m-1}x^{m-1} + \cdots + r'_1 x + r'_0 \quad (12)$$

r_0, 식 (8) 그리고 식 (9)로부터 아래의 식들을 얻을 수 있다.

$$r'_m = 0 \quad (13)$$

$$r'_{m-1} = r_0 s_m = r_0 s_m + 0 \quad (14)$$

$$r'_i = r_0 s_{i+1} + r_{i+1},\ 0 \le i \le m-2 \quad (15)$$

U의 연산에 관해 살펴보면, $(U + V)$ 그리고 U/x을 계산해야 한다.

$(U + V)$를 U''과 같이 두면

$$U'' = u''_{m-1}x_{m-1} + \cdots + u''_1 x + u''_0 = U + V \quad (16)$$

r_0, 식 (10) 그리고 식 (11)로부터 우리는 아래의 식 (17)을 얻을 수 있다.

$$u''_i = r_0 v_i + u_i,\ 0 \le i \le m-1 \quad (17)$$

x가 기약다항식 $G(x)$의 근이기 때문에 식 (18)을 얻을 수 있다.

$$x^m = g_{m-1}x^{m-1} + g_{m-2}x^{m-2} + \cdots + g_1 x + g_0 \quad (18)$$

식 (18)로부터 g_0는 항상 1이기 때문에 (19)와 같이 다시 쓸 수 있다.

$$1 = (x^{m-1} + g_{m-1}x^{m-2} + g_{m-2}x^{m-3} + \cdots + g_2 x + g_1)x \quad (19)$$

식 (19)로부터, 양변에 x를 나누면, 우리는 아래의 식 (20)을 얻을 수 있다.

$$x^{-1} = x^{m-1} + g_{m-1}x^{m-2} + g_{m-2}x^{m-3} + \cdots + g_2 x + g_1 \quad (20)$$

U/x의 결과를 U'''라 두면

$$U''' = u'''_{m-1}x^{m-1} + \cdots + u'''_1 x + u'''_0 = U/x \quad (21)$$

식 (10)과 식 (20)을 식 (21)에 대입하면, 아래의 식들은 유도 될 수 있다.

$$u'''_{m-1} = u_0 = u_0 g_m + 0 \quad (22)$$

$$u'''_i = u_{i+1} + u_0 g_{i+1},\ 0 \le i \le m-2 \quad (23)$$

마지막으로 U''/x의 결과를 U'라 두면

$$U' = u'_{m-1}x^{m-1} + \cdots + u'_1 x + u'_0 = U''/x \quad (24)$$

식 (16)과 식 (21)로부터, U의 임시적인 결과를 아래와 같이 얻을 수 있다.

$$u'_{m-1} = r_0v_0 + u_0 = (r_0v_0 + u_0)g_m + r_00 + 0 \quad (25)$$

$$u'_i = (r_0v_{i+1} + u_{i+1}) + (r_0v_0 + u_0)g_{i+1}, \quad 0 \le i \le m-2 \quad (26)$$

또한 [알고리즘 2]와 핵심 연산들을 위한 제어 함수들은 아래와 같다.

$$\text{Ctrl1} = (r_0==1) \quad (27)$$

$$\text{Ctrl2} = u_0 \text{ XOR } (v_0 \ \& \ r_0) \quad (28)$$

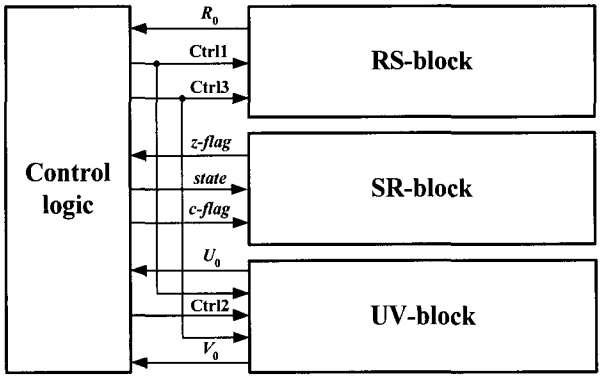
$$\text{Ctrl3} = (state == 0)\&(r0==1) \quad (29)$$

$$count' = \begin{cases} count+1, if\ state == 0 \\ count-1, if\ state == 1 \end{cases} \quad (30)$$

$$state = \overline{state},\ if \begin{cases} ((r_m == 1)\&(state == 0))or \\ ((count == 0)\&(state == 1)) \end{cases} \quad (31)$$

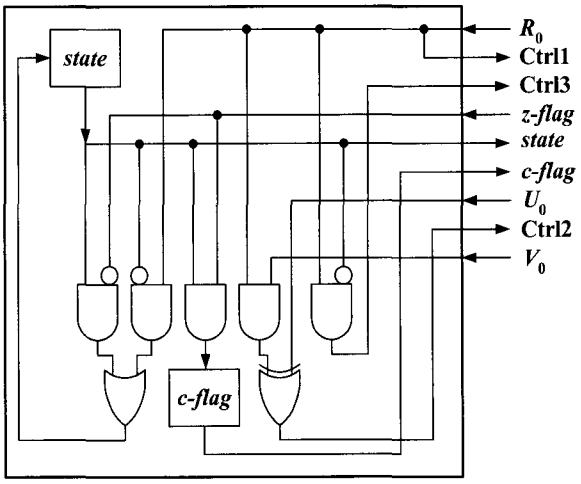
3.2 GF(2^m)을 위한 새로운 나눗셈 회로

핵심 연산 및 제어함수로부터 우리는 (그림 1)과 같은 새로운 나눗셈 구조를 얻을 수 있다.

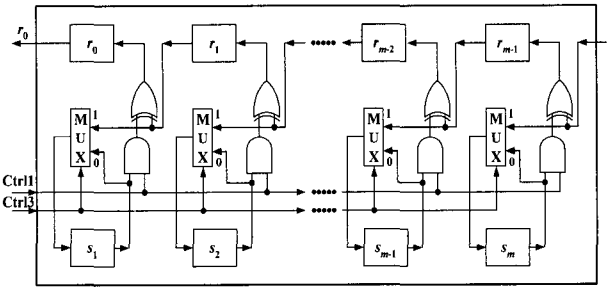


(그림 1) GF(2^m)을 위한 새로운 나눗셈 구조

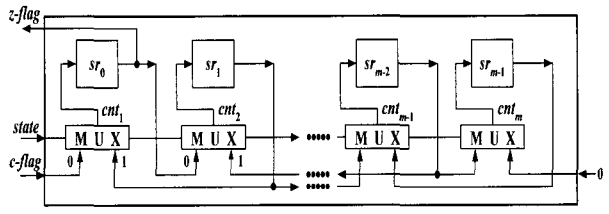
(그림 1)의 나눗셈기는 Control logic, RS-block, SR-block 그리고 UV-block으로 구성되어있다. 각각에 해당하는 기능 블록은 (그림 2), (그림 3), (그림 4) 그리고 (그림 5)와 같다. 핵심 연산으로부터 s_0는 항상 1이고 r_m은 항상 0 이기 때문에 (그림 3)에서 이 계수들을 위한 연산 회로는 제거하였다. (그림 4)에 나타나듯이 [알고리즘 2]의 *count* 값을 위해서 $\log_2(m+1)$-비트 카운터 대신 m-비트 양방향 쉬프트 레지스터를 사용하였다. 이렇게 함으로서 약간의 하드웨어 비용은 증가하나 높은 클락 주파수를 얻을 수 있다. Control logic은 현재의 반복을 위해 Ctrl1, Ctrl2, Ctrl3 그리고 *c-flag*의 신호를 생성하고 다음 반복을 위해 *state* 및 *c-flag*값을 갱신한다. 1-비트 *c-flag* 레지스터는 SR-block과 함께 동작하며 나눗셈을 시작할 때 1로서 초기화된다. (그림 3)의 RS-block은 [알고리즘 2]의 R 과 S에 관한 연산을 수행하고 Control logic에 r_0신호를 전파한다. 그림 4에 나타나듯이 양방향 쉬프트 레지스터는 *state*에 따라 왼쪽 혹은 오른쪽으로 쉬프트되며, cnt_m이 1일 때 *count* 값은 m이 된다. 또한 *count*가 0으로 감소하게되면, *z-flag*는 1이 되고 모든 cnt_m은 0이 되고, *c-flag*는 1이 되고 *state*는 0이 된다. 결국 이것은 나눗셈의 첫 번째 연산 조건과 동일하다. (그림 5)의 UV-block은 U와 V에 관한 연산을 수행하며 Contrl logic에 u_0 와 v_0 신호를 전파한다.



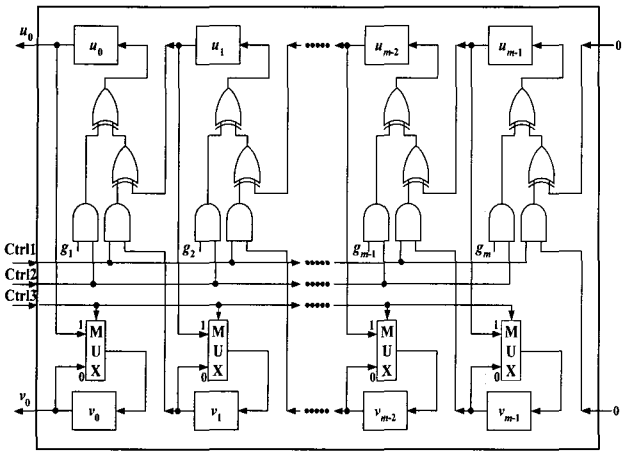
(그림 2) 그림 1의 Control logic 회로



(그림 3) 그림 1의 RS-block 회로



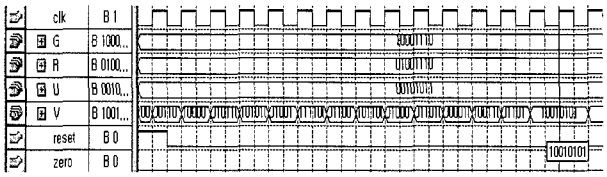
(그림 4) 그림 1의 SR-block 회로



(그림 5) 그림 1의 UV-block 회로

3.3 FPGA 구현 및 기능 검증

(그림 1)에 있는 나눗셈기의 기능 검증을 위해, VHDL로 회로를 기술하였고, Mento Graphics사의 합성 툴(Leonard Spectrum Version: 2002c.15)을 사용하여 회로를 합성, Net-list 파일을 추출한 후, Altera사의 Quartus II 2.1을 이용하여 Place & Route 과정을 거친 후, 타이밍 분석 및 시뮬레이션을 수행하였다. 여기서 Altera사의 150만 게이트급인 EP2A40F1020C7을 대상 디바이스로 선택하였다.



(그림 6) GF(2^8)상의 시뮬레이션 결과

(그림 6)은 GF(2^8)상의 나눗셈기 회로에 대한 시뮬레이션 결과를 나타내며, 314(MHz)의 최대 동작 주파수를 얻었다.

시뮬레이션 과정에서 입력된 데이터는 식 (32)부터 식 (34)와 같고 정확한 나눗셈 결과는 식 (35)와 같다.

$$U(x) = x^5 + x^3 + x + 1 \tag{32}$$

$$R(x) = x^6 + x^3 + x^2 + x \tag{33}$$

$$G(x) = x^8 + x^4 + x^3 + x^2 + 1 \tag{34}$$

$$V(x) = x^7 + x^4 + x^2 + 1 \tag{35}$$

(그림 6)에 나타나듯이 본 연구에서 제안된 나눗셈기는 $15(2m-1)$ 클락 사이클 후에 정확한 결과 값을 출력한다.

〈표 2〉 GF(2^m)상의 나눗셈 회로들의 성능 비교

	Brunner 등[8]	Guo 등[9]	제안된 나눗셈 회로
처리율 (1/cycles)	$1/2m$	$1/m$	$1/2m-1$
지연시간 (cycles)	$2m$	$5m$-4	$2m-1$
최대 처리기 지연시간	Tzero-detector + $2T_{AND2}$ + $2T_{XOR2}$ + $2T_{MUX2}$	$T_{AND2}+3T_{XOR2}+T_{MUX2}$	$2T_{AND2}$ + $2T_{XOR2}$
회로의 구성요소들	AND_2: $3m+\log_2(m+1)$ XOR_2: $3m+\log_2(m+1)$ Latch: $4m+\log_2(m+1)$ MUX_2: $8m$	AND_2 : $16m$-16 XOR_2 : $10m$-10 Latch : $44m$-43 MUX_2 : $22m$-22	AND_2: $3m$+5 XOR_2: $3m+1$ OR_2: 1 Latch: $5m+2$ MUX_2: $3m$
TR 갯수	$110m+18\log_2(m+1)$	$608m$-432	$88m$+48

AND_i: i-input AND gate
XOR_i: i-input XOR gate
OR_i: i-input OR gate
MUX_i: i-to-1 multiplexer
T_{ANDi}: the propagation delay through one AND_i gate
T_{XORi}: the propagation delay through one XOR_i gate
T_{MUXi}: the propagation delay through one MUX_i gate
Tzero-detector: the propagation delay of $(\log_2^{m+1})$-bit zero-detector

4. 성능 분석

본 연구에서 제안한 나눗셈기를 기존에 제안된 나눗셈기들과 비교하였다. <표 2>에 성능 비교 결과를 요약하였으며, 조금 더 자세한 비교를 위해 3-입력 XOR 게이트는 2개의 2-입력 XOR 게이트로 구성된다고 가정하였다. 또한 2-입력 AND 게이트, 2-입력 XOR 게이트, 2-to-1 MUX, 2-입력 OR 게이트, 그리고 1-비트 래치는 각각 4, 6, 6, 6 그리고 8개의 트랜지스터(TR)로 구성된다고 가정하였다[12].

<표 2>에 기술된바와 같이, Brunner 등이 제안한 나눗셈기는 본 연구 결과와 거의 동일한 지연시간을 가지지만 2배 이상의 최대 처리기 지연시간을 가지며 더 많은 TR을 사용한다. 또한 Guo 등이 제안한 나눗셈기는 본 연구에서 제안한 나눗셈기 보다 높은 최대 처리기 지연시간 및 약 2.5배의 계산 지연시간과 약 6.9배의 더 많은 TR을 사용한다. 따라서 본 연구에서 제안된 나눗셈기는 기존의 연구 결과들에 비해 시간 및 공간 모두에 있어 효율적이다.

5. 결 론

본 연구에서는 ECC를 위한 새로운 나눗셈기를 제안하였다. 제안된 나눗셈기는 수정된 바이너리 확장 GCD 알고리즘으로부터 설계되었으며, FPGA 구현을 통하여 정확히 동작함을 확인하였다.

제안된 나눗셈기는 $O(m)$의 시간 및 면적 복잡도를 가지며, $2m$-1 사이클 비율로 나눗셈 결과를 출력한다. 제안된 나눗셈기를 기존의 나눗셈기들과 비교 분석한 결과 칩 면적 및 계산 지연시간 모두에 있어 상당한 개선을 보였다. 또한 제안된 구조는 기약다항식의 선택에 있어 어떠한 제약 조건도 두지 않을 뿐만 아니라 매우 규칙적이고 모듈화 하기 쉽기 때문에 필드 크기 m에 대해 높은 유연성 및 확장성을 제공한다. 따라서 본 논문에서 제안된 나눗셈 회로는 저면적을 요구하는 GF(2^m)의 응용에 매우 적합하다.

참 고 문 헌

[1] J.R. Goodman, Energy Scalable Reconfigurable Cryptographic Hardware for Portable Applications, PhD thesis, MIT, 2000.

[2] IEEE P1363, *Standard Specifications for Publickey Cryptography*, 2000.

[3] I. F. Blake, G. Seroussi, and N. P. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.

[4] C.-L. Wang and J.-L. Lin, "A Systolic Architecture for Computing Inverses and Divisions in Finite Fields GF(2^m)," *IEEE Trans. Computers.*, Vol.42, No.9, pp.1141-1146, Sep., 1993.

[5] M.A. Hasan and V.K. Bhargava, "Bit-Level Systolic Divider and Multiplier for Finite Fields GF(2^m)," *IEEE Trans. Computers*, Vol.41, No.8, pp.972-980, Aug., 1992.

[6] S.-W. Wei, "VLSI Architectures for Computing exponentiations, Multiplicative Inverses, and Divisions in GF(2^m)," *IEEE Trans. Circuits Syst. II*, Vol.44, No.10, pp.847-855, Oct., 1997.

[7] A.V. Dinh, R.J. Bolton, and R. Mason, "A Low Latency Architecture for Computing Multiplicative Inverses and Divisions in GF(2^m)," *IEEE Trans. Circuits Syst. II*, Vol. 48, No.8, pp.789-793, Aug., 2001.

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

[9] J.H. Guo and C.L. Wang, "Bit-serial Systolic Array Implementation of Euclid's Algorithm for Inversion and Division in GF(2^m)," *Proc. 1997 Int. Symp. VLSI Tech., Systems and Applications*, pp.113-117, 1997.

[10] D.E. Knuth, *The art of computer programming: Seminumerical algorithms*, 3rd ed. Reading, MA: Addison-Wesley, 1998.

[11] E. Bach and J. Shallit, *Algorithmic Number Theory - Volume I: Efficient Algorithms*, MIT Press, 1996.

[12] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A System Perspective*, 2nd ed. Reading, MA: Addison-Wesley, 1993.

김 창 훈

e-mail : chkim@dsp.taegu.ac.kr

2001년 대구대학교 컴퓨터정보공학부(학사)

2003년 대구대학교 컴퓨터정보공학과(석사)

2003년~현재 대구대학교 컴퓨터정보공학과 박사과정

관심분야 : 암호 시스템, Embedded System, RFID/USN 보안

이 남 곤

e-mail : nglee@daegu.ac.kr

2005년 대구대학교 컴퓨터정보공학부(학사)

2005년~현재 대구대학교 정보통신공학과 석사과정

관심분야 : 암호 시스템, RFID/USN 시스템 보안

권 순 학

e-mail : shkwon@skku.edu
1990년 KAIST 수학과(학사)
1992년 서울대학교 수학과(석사)
1997년 Johns Hopkins University(박사)
1998년~현재 성균관대학교 수학과, 부교수
관심분야 : 정수론, 암호론, Cryptographic Hardware

홍 춘 표

e-mail : cphong@daegu.ac.kr
1978년 경북대학교 전자공학과(학사)
1986년 Georgia Institute of Technology ECE(석사)
1991년 Georgia Institute of Technology ECE(박사)
1994년~현재 대구대학교 정보통신공학부, 교수
관심분야 : DSP 하드웨어 및 소프트웨어, 컴퓨터 구조, VLSI 신호처리, Embedded System