유한 필드 $GF(2^m)$상의 비트-패러럴 시스톨릭 나눗셈기

김 창 훈† · 김 종 진† · 안 병 규† · 홍 춘 표††

요 약

본 논문에서는 유한 필드 $GF(2^m)$상에서 모듈러 나눗셈 $A(x)/B(x)$ mod $G(x)$을 수행하는 고속의 병렬 시스톨릭 나눗셈기를 제안한다. 제안된 나눗셈기는 이진 최대공약수(GCD) 알고리즘에 기반하며, FPGA 칩을 이용하여 구현 및 검증한다. 본 연구에서 제안된 나눗셈기는 연속적인 입력 데이터에 대해 초기 $5m$-2 클럭 사이클 지연후, 1 클럭 사이클 비율로 나눗셈 결과를 출력한다. 본 논문에서 제안된 나눗셈기를 기존의 병렬형 시스톨릭 나눗셈기들과 비교했을 때, 훨씬 적은 하드웨어의 사용으로 계산지연 시간을 상당히 감소 시켰다. 또한 제안된 나눗셈기는 기약 다항식의 선택에 어떠한 제약도 두지 않을 뿐 아니라 매우 규칙적이고 묘듈화 하기 쉽기 때문에 필드 크기 m에 대하여 높은 확장성 및 유연성을 제공한다. 따라서 제안된 구조는 VLSI 구현에 매우 적합하다.

Bit-Parallel Systolic Divider in Finite Field $GF(2^m)$

Chang Hoon Kim† · Jong Jin Kim† · Byeung Kyui An† · Chun Pyo Hong††

ABSTRACT

This paper presents a high-speed bit-parallel systolic divider for computing modular division $A(x)/B(x)$ mod $G(x)$ in finite fields $GF(2^m)$. The presented divider is based on the binary GCD algorithm and verified through FPGA implementation. The proposed architecture produces division results at a rate of one every 1 clock cycles after an initial delay of $5m$-2. Analysis shows that the proposed divider provides a significant reduction in both chip area and computational delay time compared to previously proposed systolic dividers with the same I/O format. In addition, since the proposed architecture 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. Therefore, the proposed divider is well suited to VLSI implementation.

키워드 : 유한 필드(Finite Field), 비트-병렬형 나눗셈기(Bit-Parallel Divider), 암호 시스템(Cryptographic System), 시스톨릭 배열(Systolic Array), VLSI

1. 서 론

유한 필드$GF(2^m)$상의 연산들은 오류 제어 코딩, 암호학 등 여러 분야에서 중요한 역할을 하고 있다[1, 2]. $GF(2^m)$상의 연산에는 덧셈, 뺄셈, 곱셈, 그리고 나눗셈 등이 있다. 여기서 덧셈과 뺄셈은 벡터간의 XOR연산으로 간단하게 수행할 수 있으나, 곱셈은 원시 기약 다항식 $G(x)$에 의해$A(x)\times B(x)$ mod $G(x)$로, 나눗셈은 $A(x)/B(x)$ mod $G(x)$로 각각 정의되기 때문에 복잡하다. 특히 나눗셈은 시간 및 하드웨어에 대해서 가장 복잡한 연산이기 때문에 이의 효율적인 구현은 $GF(2^m)$의 응용에 있어 매우 중요하다.

$GF(2^m)$상에서 표준 기저표기법을 사용할 경우 ① 선형 방정식 집합의 해를 구하는 방법[3], ② Fermat의 이론[4, 5], ③ Euclid[6] 혹은 바이너리 GCD 알고리즘[7]이 이용되어 왔다. 첫 번째 방법은 $2m$-1개의 미지수를 가진 $2m$-1개의 선형 방정식들의 해를 구함으로써 역원을 찾아낸다. 두 번째 방법은 Fermat의 이론을 이용하여 연속적인 제곱과 곱셈을 함으로써 역원을 찾는다. 즉 $A/B = AB^{-1} = AB^{2^m-1} = A(B(B(B\cdots B(B(B)^2)^2\cdots)^2)^2)^2$의 관계식을 이용하는데, 이 방법은 $m-1$ 번의 제곱연산과 $m-1$ 번의 곱셈연산을 필요로 한다[4]. 정규기저 표기법에 있어, 제곱연산은 순환 쉬프트 연산이기 때문에, 정규기저 표기법을 사용할 경우 많이 이용된다. 마지막 방법은 Euclide 혹은 바이너리 GCD 알고리즘을 이용하는 방법으로 GCD$(A(x), B(x)) = W(x)\cdot A(x) + U(x)\cdot B(x)$에서 $A(x)$에 기약 다항식, $G(x)$를 넣으면 GCD$(G(x), B(x)) = W(x)\cdot G(x) + U(x)\cdot B(x) = 1$에서 $U(x)\cdot B(x) \equiv 1$ mod $G(x)$가 되어 $U(x)$가 $B(x)$의 역원 즉, $B^{-1}(x)$가 된다. 최근 연구 결과에 의하면 표준 기저표기법을 사용할 경우 바이너리 GCD 알고리즘을 사용했을 때 가장 적은 하드웨어의 사용으로 가장 낮은 계산 지연시간을 가지는 것으로 나타났다[7].

※ 본 연구는 2002학년도 대구대학교 학술연구비에 의하여 수행되었음.
† 준 회 원 : 대구대학교 대학원 컴퓨터정보공학과
†† 정 회 원 : 대구대학교 정보통신공학부 교수
논문접수 : 2003년 11월 28일, 심사완료 : 2004년 3월 24일

이러한 나눗셈기 들은 비트-시리얼 또는 비트-패러럴 형태의 구조를 가진다. 비트-시리얼 형태의 구조는 적은 하드웨어의 사용을 가능하게 하지만 속도가 느리기 때문에 고속의 연산을 필요로 하는 응용에서는 적절하지가 못하다. 이에 비해서 비트-패러럴 형태의 구조는 고속의 나눗셈 연산을 사용하기에 적합한 형태이다. 또한 나눗셈기는 시스톨릭 구조 또는 비 시스톨릭 형태로 구현이 가능한데, 비 시스톨릭 구조는 시스톨릭 구조에 비해 낮은 칩 면적을 가지지만, m이 커질 경우 전역 신호전파 때문에 심각한 속도 저하를 초래할 뿐 아니라, 신뢰성을 제공하지 못한다[4].

본 논문에서는 $GF(2^m)$상의 표준 기저 표기법을 사용하여, 나눗셈을 위한 비트-패러럴 형태의 시스톨릭 구조를 제안한다. 제안한 나눗셈기는 이진 GCD 알고리즘에 기반하며, 이 알고리즘으로부터 자료의존 그래프 (DG : Dependenc Graph)를 얻은 후 컷셋 시스톨릭화 기법(cut-set systolization techniques)[8]을 적용하여, $GF(2^m)$상의 완전한 나눗셈 연산기를 얻는다. 제안된 나눗셈기는 $O(1)$의 시간복잡도와 $O(m^2)$의 공간 복잡도를 가지며, 연속된 입력 데이터에 대하여, 초기 $5m$-2 사이클의 지연 후, 1 사이클 마다 나눗셈의 결과를 출력한다.

본 연구에서 제안된 시스톨릭 나눗셈기를 동일한 입출력 형태를 가지는 기존의 시스톨릭 나눗셈기들과 비교 분석한 결과 칩 면적 및 계산 지연시간 모두에 있어 상당한 개선을 보인다. 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.

2. $GF(2^m)$상의 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 $A(x)/B(x)$ mod $G(x)$의 결과라고 하면, 각각의 다항식은 다음과 같이 표현되며 계수들은 이진수 0 혹은 1이다.

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

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

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

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

기존 연구에서는 $GF(2^m)$상의 나눗셈 연산 $A(x)/B(x)$ mod $G(x)$를 수행하기 위하여 아래의 (알고리즘 Ⅰ)을 사용했으며, 기본적으로 GCD($G(x)$, $B(x)$)의 결과가 항상 1이라는 사실에 기반 한다. 또한 (알고리즘 Ⅰ)은 다음과 같은 3가지의 특성을 가지고 있다.

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

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

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

(알고리즘 Ⅰ)은 간단하지만 반복 종료 조건이 가변적이며, 변수 R과 S에 대한 비교과정(단계 12)이 있기 때문에 하드웨어로 구현하기엔 적합하지 않다. 본 논문에서는 이러한 두 가지의 문제점을 해결함으로써 VLSI 구현에 적합한 새로운 $GF(2^m)$상의 나눗셈 알고리즘을 제안하며, 그 결과는 (알고리즘 Ⅱ)에 기술되어 있다.

```
Input : G ( x ), A( x ), B ( x )
Output : U has P (x)=A (x) / B (x) mod G (x)
Initialize : R = B (x), S = G = G (x), U = A(x), V = 0
1. while S ≠ 0 do
2.    while r0 == 0 do
3.       R = R/x
4.       if u0 == 0 then U = U/x ; else U = (U + G )/x ;
5.       end if
6.    end while
7.    while s0 == 0 do
8.      S = S/x
9.      if v0 == 0 then V = V/x ; else V = (V + G)/x ;
10.     end if
11.   end while
12.   if S ≥ R then
13.     (S, R) = (S + R, R) ;  (V, U ) = (V, U + V ) ;
14.   else
15.     (S, R) = (R + S, R) ;  (V, U ) = (U + V, U) ;
16.   end if
17. end while
```

(알고리즘 Ⅰ) $GF(2^m)$상의 바이너리 확장 GCD 알고리즘

(알고리즘 Ⅰ)로부터 (알고리즘 Ⅱ)를 유도하는 과정은 다음과 같이 요약할 수 있다.

(1) (알고리즘 Ⅰ)에서 S의 초기값은 $G(x)$이고, R의 초기값은 $B(x)$이기 때문에 상수값 s_0는 항상 1이다. 또한 (알고리즘 1)의 첫 번째 반복에서는 $S \geq R$의 관계식이 항상 성립하기 때문에 $r_0=1$이면 스텝 13이 수행되며, $r_0=0$이면 $r_0=1$이 될 때까지 스텝 3이 수행된다. 따라서 스텝 13이 수행될 때 $(S,R)=(S+R,R)$ 대신에 $(S,R)=(R,R+S)$을 적용할 수 있다. 이 경우 (알고리즘 Ⅰ)이 종료되면 S는 GCD(S,R) 값을 가지기 때문에 s_0는 항상 1이 된다. 만약 (알고리즘 Ⅱ)에서 $s_0=1$로 설정한다면 (알고리즘 Ⅰ)의 단계 7부터 단계 11을 제거할 수 있다. 이를 구현하기 위해 r_0 값에 따라 GCD(S,R)은 세 가지 다른 방법으로 구할 수 있다. 만약 $r_0=0$이면 GCD $(S,R)=\mathrm{GCD}(S,R/x)$를 적용하고, 만약 $r_0=1$이면 (알고리즘 Ⅰ)의 스텝 12의 결과에 따라 GCD $(S,R)=\mathrm{GCD}$ $(R,(S+R)/x)$ 또는 GCD$(S,R)=\mathrm{GCD}(S,(S+R)/x)$를 적용한다. 결과적으로 s_0의 값은 항상 1이기 때문에 우리는 단지 r_0 값이 1 또는 0인지만 검사하면 된다. (알고리즘 Ⅱ)에서는 이들이 반영되었으며, (알고리즘 Ⅰ)의 단계 7부터 단계 11이 제거되었음을 알 수 있다.

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** $r_0 == 1$ **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** $r_0 == 1$ **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. **if** $u_0 == 0$ **then** $U = U/x$ **else** $U = (U+G)/x$
19. **end if**
20. **end for**

(알고리즘 Ⅱ) New Division Algorithm in $GF(2^m)$

(2) (알고리즘 Ⅰ)의 첫 번째 반복에서는 S의 차수는 m이고 R의 차수는 기껏해야 m-1이기 때문에 각 반복에 있어 S 혹은 R의 차수를 1씩 감소시킨다면 $2m$의 반복 후 R은 항상 0이 되고 S는 1이 된다. 또한 R이 n만큼 차수가 감소 된 후 r_0가 1일 때 $\mathrm{GCD}(S, R) = \mathrm{GCD}(R, (S + R)/x)$를 적용한다. 다음의 n회 반복에 있어서는 만약 $r_0 = 0$이면 $\mathrm{GCD}(S, R) = \mathrm{GCD}(S, R/x)$를 적용하고, $r_0 = 1$이면 $\mathrm{GCD}(S, R) = \mathrm{GCD}(S, (S + R)/x)$를 적용한다. 이들 관계식을 구현하고 (알고리즘 Ⅰ)의 비교 부분을 제거하기 위해서 새로운 변수 *count*와 *state*를 추가한다. (알고리즘 Ⅱ)에 기술된 것처럼 *state*와 r_0에 따라 네 가지의 다른 조건을 적용한다 : ① *state* = 0이고 $r_0 = 0$이면, 식 (6)을 적용하고 $r_0 = 1$이 될 때까지 R의 차수는 감소하고 *count*는 증가한다, ② *state* = 0이고 $r_0 = 1$이면, $\mathrm{GCD}(S, R) = \mathrm{GCD}(R, (S + R)/x)$를 적용한다. ③ *state* = 1이고 $r_0 = 0$이면, 식 (6)을 적용하고 *count*는 감소한다. 또한 *count* = 0이면, 초기의 반복 조건과 같기 때문에 *state* = 0으로 설정한다. ④ *state* = 1이고 $r_0 = 1$이면, $\mathrm{GCD}(S, R) = \mathrm{GCD}(S, (S + R)/x)$를 적용한다. 지금까지 기술된 것처럼 위의 4가지 경우에 대하여 R/x는 항상 실행되며, *state*와 r_0 따라 S와 R은 바뀌기 때문에 우리는 R과 S의 차수를 재귀적으로 감소시킬 수 있다. 따라서 알고리즘의 $2m$회 반복 후에는 $R = 0$, $S = 1$, *state* = 0, *count* = 0이 된다.

(3) 나눗셈의 결과를 얻기 위해, [8]에 설명된 것과 동일한 확장 방법으로 이진 GCD 알고리즘을 확장하였다. 결과적으로, $2m$번의 알고리즘 반복 후 V는 나눗셈 결과 $P(x) = A(x)/B(x) \bmod G(x)$를 가진다.

제안된 (알고리즘 Ⅱ)에서 마지막 반복의 초기에는 '*count* = 1 및 *state* = 1'의 관계식을 만족시켜야 하기 때문에 나눗셈 결과 V는 항상 $2m$-1번의 반복 후에 얻어진다. <표 1>은 $m = 4$, $G(x) = x^4 + x - 1$, $A(x) = x^3 + x^2 + x$, $B(x) = x^3 + x + 1$일 때 (알고리즘 Ⅱ)의 실행결과를 보여준다. <표 1>에 기술된 것처럼 ($2m$-1 = 7)번의 반복 후에 V는 $A(x)/B(x) \bmod G(x) = x + 1$의 나눗셈 결과를 얻는다.

〈표 1〉 (알고리즘 Ⅱ)에 따른 $GF(2^4)$상에서의 나눗셈 연산

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

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

(알고리즘 Ⅱ)의 VLSI 구현을 위하여 우선 핵심연산 및 제어함수에 대해서 고려한다. (알고리즘 Ⅱ)의 R과 S는 차수가 m인 다항식, 그리고 U와 V는 차수가 m-1인 다항식으로 각각 표현 할 수가 있으며, 각 다항식은 아래와 같다.

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

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

$$U = u_{m-1} x^{m-1} + u_{m-2} x^{m-2+} \cdot + u_1 x + u_0 \tag{10}$$

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

(알고리즘 Ⅱ)에 기술된 것처럼 S와 V는 *state*와 r_0에 따른 간단한 R과 U의 교환 연산이다. 반면에 R과 U는 각각 2개의 연산을 가진다. 첫째, R의 연산을 고려해 해보면 r_0값에 따라 (R/x) 또는 $((R + S)/x)$가 수행되며, R의 중간결과를 다음과 같이 얻을 수 있다.

$$\begin{aligned} R' &= r'_m x^m + r'_{m-1} x^{m-1} + \cdots + r'_1 x + r'_0 \\ &= (R + r_0 S)/x \end{aligned} \tag{12}$$

식 (8) 및 식 (9) 그리고 r_0에 의해 다음의 결과를 얻을 수 있다.

$$r'_m = 0 \tag{13}$$

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

$$r'_i = r_0 s_{i+1} + r_i + 1, \; 0 \le i \le m-2 \tag{15}$$

둘째, U의 중간결과를 얻기 위해서 U의 연산을 고려해 보면, 두 가지 연산 $U = (U + V)$와 U/x이다.

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

식 (10) 및 식 (11) 그리고 r_0에 의해 다음의 결과를 얻을 수 있다.

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

(알고리즘 Ⅱ)의 스텝 18로부터 식 (18), 식 (19)를 유도할 수 있으며, 식 (20)을 이용하여 식 (21)과 식 (22)를 유도할 수 있다.

$$u'''_{m-1} = u_0 \quad (18)$$

$$u'''_i = u_{i+1} + u_0 g_{i+1},\ 0 \le i \le m-2 \quad (19)$$

$$U' = u'_{m-1}x^{m-1} + \cdots + u'_1 x + u'_0 = U''/x \quad (20)$$

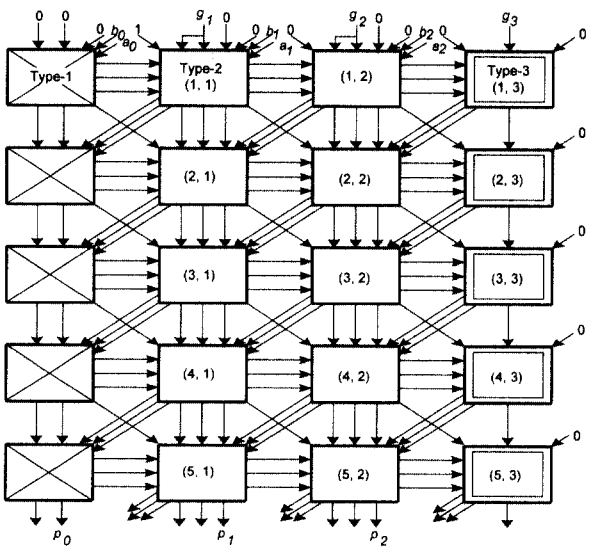
$$u'_{m-1} = r_0 v_0 + u_0 = (r_0 v_0 + u_0) g_m \quad (21)$$

$$u'_i = (r_0 v_{i+1} + u_{i+1}) + (r_0 v_0 + u_0) g_{i+1},\ 0 \le i \le m-2 \quad (22)$$

3. $GF(2^m)$상의 시스톨릭 나눗셈기 설계

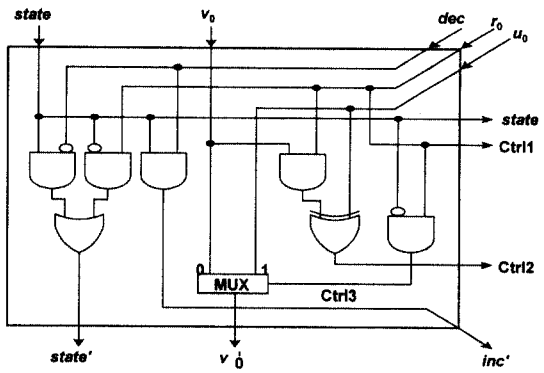
2장의 $GF(2^m)$ 상의 나눗셈 알고리즘 및 핵심 연산으로부터 $GF(2^m)$상의 나눗셈 연산을 위한 DG를 구하면 (그림 1)과 같다. (그림 1)의 DG는 $(2m-1)$개의 Type-1 셀과 $(2m-1)\times(m-1)$개의 Type-2 셀, 그리고 $(2m-1)$개의 Type-3 셀들로 구성되며, Type-1, Type-2 및 Type-3 셀의 구조는 각각 (그림 2), (그림 3), (그림 4)와 같다. (그림 1)에서 입력 값 $R(x)$, $U(x)$, 그리고 $G(x)$는 어레이의 첫 번째 열에 병렬 형태로 입력된다.

또한 어레이의 i번째 열은 알고리즘의 i번째 반복을 수행하며, 나눗셈 결과 $V(x)$는 $2m-1$회의 반복 후에 마지막 열로부터 얻을 수 있다. 참고로 (알고리즘 Ⅱ)에서 s_0는 항상 1이고 r_m은 항상 0이기 때문에 이들 연산을 위한 회로는 필요 없다.



(그림 1) 제안된 나눗셈 알고리즘의 $GF(2^3)$에 대한 자료의존 그래프

각 셀에 대한 자세한 설명에 앞서 우선 *count*의 구현에 대해 고려해보면, (알고리즘 Ⅱ)에서 *count*의 값은 기껏해야 m까지 증가하기 때문에 m-비트 크기의 양방향 쉬프트 레지스터를 이용해서 이를 구현할 수 있다. 다시 말해서, 현재의 *count* 값이 n이면$(0 \le n \le m)$, n번째 레지스터의 값은 1이고 나머지는 모두 0으로 처리하고, 현재의 *count* 값이 n이라고 가정하면 된다. 따라서 각각의 Type-2 셀과 Type-3 셀에 2-to-1 멀티플렉서, *inc* 및 *dec* 신호를 를 추가한다. 알고리즘의 i번째 반복에서는 m-bit의 양방향 쉬프트 레지스터가 *state*의 값에 따라 좌/우로 쉬프트 된다. (그림 3), (그림 4)에서 cnt_n이 1이면 *count* 값이 $n(1 \le n \le m)$이 되었음을 의미한다. 따라서 동일한 열에 있는 모든 Type-2 셀과 Type-3 셀의 cnt_n은 0이 되고, (그림 2)의 *inc'* = 1, *state* = 0이 된다.



(그림 2) Type-1 셀의 회로도

다음으로 Type-1, Type-2 그리고 Type-3 셀의 기능은 아래와 같이 요약할 수 있다.

(1) Type-1 셀 : Type-2 셀을 제어하기 위해 아래의 네 가지 제어신호를 생성한다.

$$\text{Ctrl1} = (r_0 == 1) \quad (23)$$

$$\text{Ctrl2} = u_0 \ \text{XOR}\ (v_0\ \&\ r_0) \quad (24)$$

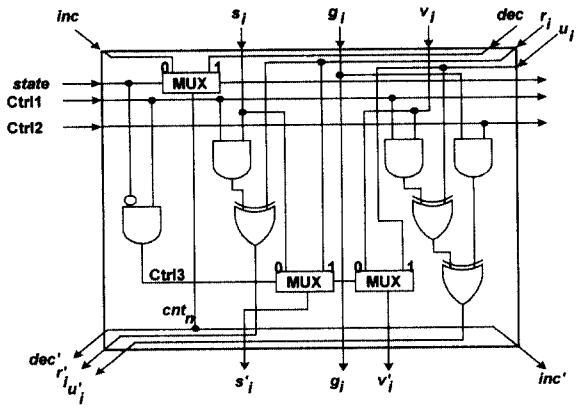
$$inc' = (state == 1)\ \&\ (dec == 1) \quad (25)$$

$$\dot{state} = \overline{state},\ \text{if}\ \begin{cases} ((r_m == 1)\ \ (state == 0)) or \\ ((dec == 1)\ \ (state == 1)) \end{cases} \quad (26)$$

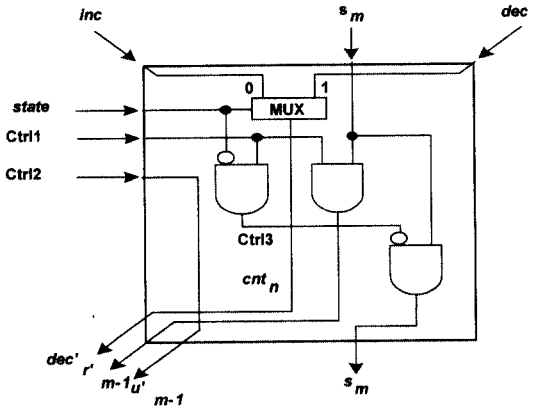
Ctrl2는 (알고리즘 Ⅱ)의 스텝 18에 있는 u_0를 결정하기 위해 사용되며, *state*와 *dec*가 1이 되면, *count*가 0이 되는 것을 의미한다. 따라서 *state*는 다시 0이 되고, *inc'* = 1이 된다. 또한 그 열의 모든 Type-2 셀과 Type-3셀의 *inc'* = *dec'* = 0이 되어 결국 알고리즘이 시작될 때와 같이 *count* = 0이 된다.

(2) Type-2 및 Type-3 셀 : Type-1 셀로부터 *state*, Ctrl1 그리고 Ctrl2를 받아 2장에 설명된 연산들 및 *count*의 값을 계산한다. 또한 (알고리즘 Ⅱ)의 스텝 5와 스텝 11을 수행하기 위해 식 (27)의 제어신호 Ctrl3을 생성한다. (그림 3)에 나타나듯이 *state*가 0이면, *inc*가 선택되어 *count* = *count* + 1이 되고 그렇지 않으면 *dec*가 선택되어 *count* = *count* -1이 된다.

$$\text{Ctrl3} = (state == 0)\ \&\ \text{Ctrl1} \quad (27)$$

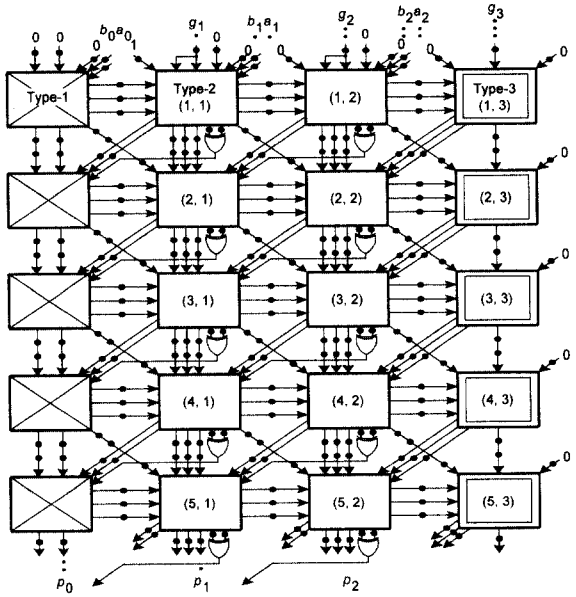


(그림 3) Type-2 셀의 회로도

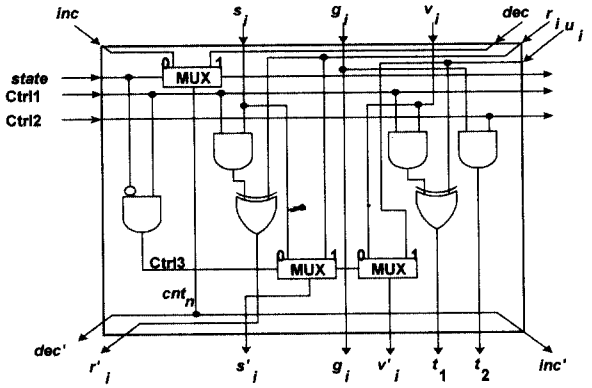


(그림 4) Type-3 셀의 회로도

이들을 종합하면 Type-1 셀은 현재와 다음 반복을 위한 제어신호들을 생성하고, Type-2 및 Type-3셀은 하나의 제어신호를 포함하며, Type-1 셀의 제어신호로부터 2장에 설명된 연산들 및 *count*의 값을 계산한다는 사실을 알 수 있다. (그림 2), (그림 3), (그림 4)로부터 임계경로는 Type-2 셀의 임시적인 u'을 계산하는 경로이다. 임계경로 u'의 계산 지연시간을 줄이기 위해서 분할 처리 기법 및 컷 셋 시스톨릭화 기법[8]을 적용하면 (그림 5)와 같은 병렬 시스톨릭 나눗셈기를 얻을 수 있고, 수정된 Type-2 셀은 (그림 6)과 같다.



(그림 5) $GF(2^3)$상의 병렬 시스톨릭 나눗셈기



(그림 6) (그림 5)의 Type-2 셀의 회로

〈표 2〉 $GF(2^m)$상의 비트-패러럴 시스톨릭 나눗셈기들의 특성 비교

	Wei *et al.* [4]	Wang *et al.* [5]	Guo *et al.* [6]	제안된 나눗셈기
처리율 (1/cycles)	1	1	1	1
지연시간 (cycles)	$3m^2-2m$	$2m^2-3m/2$	$8m-1$	$5m-2$
최대 처리기 지연시간	$T_{AND2}+2T_{XOR2}$	$T_{AND2}+3T_{XOR2}$	$T_{AND2}+2T_{XOR2}+2T_{MUX2}$	$2T_{XOR2}$
셀의 구성요소	$AND_2: 3m^3-3m^2$ $XOR_2: 3m^3-3m^2$ Latch : $13m^3-13m^2$	$AND_2: 3m^3-3m^2$ $XOR_2: 3m^3-3m^2$ Latch : $8.5m^3-8.5m^2$	inverter : $2m$ $AND_2: 7m^2+10m$ $XOR_2: 9m^2+4m$ $MUX_2: 16m^2+10m$ Latch : $34m^2+4m+4m\cdot\log_2(m+1)+2$ zero-check ($\log_2(m+1)$-bit) : $2m$ adder/subtractor($\log_2(m+1)$-bit) : $2m$	inverter : $2m^2+7m-4$ $AND_2: 8m^2-4m-4$ $XOR_2: 6m^2-7m+2$ $OR_2: 2m-1$ Latch : $32m^2-16m$ $MUX_2: 6m^2-5m+1$
면적 복잡도	$O(m^3)$	$O(m^3)$	$O(m^2)$	$O(m^2)$
시간-면적 복잡도	$O(m^3)$	$O(m^3)$	$O(m^2)$	$O(m^2)$

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 ANDi gate T_{XORi} : the propagation delay through one XORi gate
T_{MUXi} : the propagation delay through one MUXi gate

4. 성능분석

(그림 5)의 나눗셈기를 구현하기 위해 VHDL을 이용하여 회로를 기술하였고, Synopsys사의 합성 툴(FPGA-Express Version : 2000.11-FE 3.5)을 사용하여 Altera사의 EP2A70F 1508C-7을 대상 디바이스로 회로를 합성, Net-list 파일을 추출한 후, Mento-graphics사의 VHDL-ChipSim을 이용하여 시뮬레이션 하였다. 본 연구에서 제안된 나눗셈기의 기능을 검증 한 후, 동일한 입출력 형태를 가지는 기존의 나눗셈기 들과 시간 및 하드웨어 복잡도 측면에서 성능을 비교 분석하였다. <표 2>에 기술된 바와 같이, 본 연구에서 제안한 나눗셈기를 포함한 모든 비트-패러럴 나눗셈기는 동일한 시간복잡도 $O(1)$을 가진다. 하지만 기존의 나눗셈기의[4, 5] 면적 복잡도가 $O(m^3)$인데 비해 본 연구에서 제안한 나눗셈기는 $O(m^2)$의 복잡도를 가지며, 또한 가장 낮은 최대 처리기 지연시간 및 계산 지연시간을 가진다는 것을 알 수 있다. 참고로 성능 비교시 조금 더 자세한 비교를 위해 3-입력 XOR 게이트와 4-입력 XOR 게이트는 각각 2개의 2-입력 XOR 게이트 그리고 3개의 2-입력 XOR 게이트로 구성된다고 가정하였다.

5. 결 론

본 논문에서는 $GF(2^m)$ 상의 고속 병렬 시스톨릭 나눗셈기를 제안하였다. 제안된 구조는 바이너리 GCD 알고리즘과 2장에서 유도한 핵심 연산에 기반하며, 분할 처리 기법을 적용하여 최대 처리 지연시간을 줄였다. 본 연구에서 제안된 나눗셈기는 연속적인 입력 데이터에 대해 초기 $5m-2$ 클럭 사이클 지연후, 1 클럭 사이클 비율로 나눗셈 결과를 출력한다. 또한 제안된 나눗셈기를 동일한 입출력 형태를 가지는 기존의 시스톨릭 나눗셈기들과 비교 분석한 결과 속도 및 면적 모두에 있어 상당한 개선을 보였다. 또한 제안된 나눗셈기는 기약다항식의 선택에 어떠한 제약도 두지 않을 뿐 아니라 매우 규칙적이고 묘듈화 하기 쉽기 때문에 필드 크기 m에 대하여 높은 확장성 및 유연성을 제공한다. 따라서 제안된 구조는 VLSI 구현에 매우 적합하다.

참 고 문 헌

[1] R. E. Blahut, *Theory and Practice of Error Control Codes*, MA : Addison-Wesley, 1983.

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

[3] 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.

[4] 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.

[5] C.-L. Wang, "New Systolic Arrays for C + AB^2, Inversion,and Division in GF(2^m)," *IEEE Trans. Computers*, Vol.49, No.10, pp.1120-1125, Oct., 2000.

[6] J.-H. Guo and C.-L. Wang, "Systolic Array Implementation of Euclid's Algorithm for Inversion and Division in GF(2^m)," *IEEE Trans. Computers.*, Vol.47, No.10, pp. 1161-1167, Oct., 1998.

[7] C. H. Kim and C. P. Hong, "High-speed division architecture for GF(2^m)," Electronics Letters, Vol.38, No.15, pp. 835-836, July, 2002.

[8] S. Y. Kung, *VLSI Array Processors*, Englewood Cliffs, NJ : Prentice Hall, *Applications*, pp.113-117, 1988.

김 창 훈

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

2000년 대구대학교 컴퓨터정보공학부(학사)

2002년 대구대학교 컴퓨터정보공학과(석사)

2004년~현재 대구대학교 컴퓨터정보공학과 박사과정

관심분야 : 암호 시스템, 내장형 시스템, 재구성형 컴퓨팅

김 종 진

e-mail : jjkim@dsp.daegu.ac.kr

2003년 대구대학교 컴퓨터정보공학부(학사)

2003년~현재 대구대학교 컴퓨터정보공학과 석사과정

관심분야 : 암호 시스템, 내장형 시스템, 재구성형 컴퓨팅

안 병 규

e-mail : anbk007@hanmail.net

1991년 경일대학교 전자계산학과

1995년 경일대학교 산업대학원 전자계산학과(석사)

2001년 대구대학교 컴퓨터정보공학과 수료(박사)

관심분야 : 컴퓨터구조, 내장형 시스템, 암호 시스템

홍 춘 표

e-mail : cphong@daegu.ac.kr

1978년 경북대학교 전자공학과(학사)

1986년 Georgia Institute of Technology, Electrical and Computer Engineering, 석사

1991년 Georgia Institute of Technology, Electrical and Computer Engineering, 박사

1992년~현재 대구대학교 정보통신공학부 교수

관심분야 : DSP 하드웨어 및 소프트웨어, 프로세서 구조, 암호 시스템, 내장형 시스템