타원곡선 암호프로세서의 재구성형 하드웨어 구현을 위한 GF(2^m)상의 새로운 연산기

(A Novel Arithmetic Unit Over GF(2^m) for Reconfigurable Hardware Implementation of the Elliptic Curve Cryptographic Processor)

김 창 훈 † 권 순 학 †† 홍 춘 표 ††† 유 기 영 ††††

(Chang Hoon Kim) (Soonhak Kwon) (Chun Pyo Hong) (Kee-Young Yoo)

요 약 유연성을 제공하지 못하는 ASIC 구현의 단점을 해결하기 위해, 본 논문에서는 타원곡선 암호프로세서의 재구성형 하드웨어(FPGAs) 구현을 위한 GF(2^m)상의 새로운 연산기를 제안한다. 제안된 연산기는 바이너리 확장 GCD 알고리즘과 MSB 우선 곱셈 알고리즘을 기반으로 하며, 전역신호 전파를 제거하기 위해 시스톨릭 구조로 설계되었다. 제안된 구조는 GF(2^m)상의 나눗셈 및 곱셈 모두를 수행 할 수 있다. 즉 연속된 입력 데이타에 대해 나눗셈 모드에서 초기 $5m-2$ 클럭 사이클 지연 후 매번 m 클럭 사이클 비율로 나눗셈의 결과를 출력하며, 곱셈 모드에서 초기 $3m$ 클럭 사이클 지연 후 매번 m 클럭 사이클 비율로 곱셈의 결과를 출력한다. 본 논문에서 제안된 연산기를 동일한 입출력 형태를 가지는 기존의 나눗셈기들과 비교 분석한 결과 기존의 나눗셈기들이 $O(m2)$ 혹은 $O(m \cdot (\log_2 m))$의 면적 복잡도를 가지는 반면 제안된 연산기는 $O(m)$의 면적 복잡도를 가진다. 또한 제안된 구조는 $O(m \cdot (\log_2 m))$의 면적 복잡도를 가지는 나눗셈기에 비해 훨씬 낮은 계산 지연시간을 가진다. 제안된 연산기를 Altera사의 FPGA칩인 EP2A70F1508C-7 디바이스에서 구현한 결과 GF(2^{571})상에서 최대 121MHz의 주파수에서 동작하였고, 52%의 칩 면적을 사용하였다. 따라서, 본 연구에서 제안된 산술 연산기는 FPGAs상에서 구현되는 타원곡선 암호프로세서의 나눗셈 및 곱셈 연산기로 매우 적합하다.

키워드 : GF(2^m) 나눗셈, GF(2^m) 곱셈, 타원곡선 암호시스템, 시스톨릭 어레이, 회로설계

Abstract In order to solve the well-known drawback of reduced flexibility that is associate with ASIC implementations, this paper proposes a novel arithmetic unit over GF(2^m) for field programmable gate arrays (FPGAs) implementations of elliptic curve cryptographic processor. The proposed arithmetic unit is based on the binary extended GCD algorithm and the MSB-first multiplication scheme, and designed as systolic architecture to remove global signals broadcasting. The proposed architecture can perform both division and multiplication in GF(2^m). In other word, when input data come in continuously, it produces division results at a rate of one per m clock cycles after an initial delay of $5m-2$ in division mode and multiplication results at a rate of one per m clock cycles after an initial delay of $3m$ in multiplication mode respectively. Analysis shows that while previously proposed dividers have area complexity of $O(m^2)$ or $O(m \cdot (\log_2 m))$, the proposed architecture has area complexity of $O(m)$, In addition, the proposed architecture has significantly less computational delay time compared with the divider which has area complexity of $O(m \cdot (\log_2 m))$. FPGA implementation results of the proposed arithmetic unit, in which Altera's EP2A70F1508C-7 was used as the target device, show that it ran at maximum 121MHz and utilized 52% of the chip area in GF(2^{571}). Therefore, when elliptic curve cryptographic processor is implemented on FPGAs, the proposed arithmetic unit is well suited for both division and multiplication circuit.

Key words : GF(2^m) Division, GF(2^m) Multiplication, Elliptic Curve Cryptosystem, Systolic Array, VLSI

· 본 연구는 한국과학재단 목적기초연구 (R05-2003-000-11573-0) 지원으로 수행되었음

† 비 회 원 : 대구대학교 컴퓨터정보공학과
chkim@dsp.daegu.ac.kr

†† 비 회 원 : 성균관대학교 수학과 교수
shkwon@math.skku.ac.kr

††† 비 회 원 : 대구대학교 정보통신공학부 교수
cphong@daegu.ac.kr

†††† 종신회원 : 경북대학교 컴퓨터공학과 교수
yook@knu.ac.kr

논문접수 : 2003년 2월 3일
심사완료 : 2004년 5월 4일

1. 서 론

최근 인터넷 및 무선통신의 급성장으로 정보 보호는 아주 중요한 문제로 인식되고 있다. 정보 보호를 위해선 암호 시스템들의 구현은 필요하다. 많은 암호시스템들 중 최근 타원곡선 암호 시스템(Elliptic Curve Cryptosystem: ECC)은 학계나 산업계로부터 많은 관심을 모으고 있다. ECC의 가장 주된 장점은 RSA나 ElGamal과 같은 다른 암호시스템들에 비해 현저히 작은 키를 사용하면서(약 1/6 정도) 동일한 안전도를 가진다[2-3]. 작은 키를 사용한다는 것은 계산 시간, 전력 소모 그리고 저장공간의 감소를 의미한다.

따라서 ECC의 효율적인 소프트웨어[3-5] 및 하드웨어 구현[6-10]에 대해 많은 연구 결과가 발표되었다. ECC의 소프트웨어 구현은 쉽고 높은 유연성을 제공하지만 실시간 응용을 위해선 적합하지 않을 뿐 아니라 사이드 채널 공격(타이밍 공격, 메모리 공격 등)에 약하다[9]. 따라서 보다 향상된 성능 및 안전성을 제공하기 위해 암호시스템의 하드웨어 구현은 바람직하다. 그러나 암호시스템은 다른 응용과는 달리 암호 알고리즘 자체가 바뀔 수도 있으며, 응용에 따라 필드 크기를 달리 할 수도 있으며, 때로는 다른 유한체를 선택하기도 한다. 비록 ASIC구현은 작은 크기, 빠른 속도 그리고 저 전력 등의 장점이 있지만 암호시스템이 요구하는 유연성을 제공하지 못한다. ASIC과 달리 FPGAs는 하드웨어의 재구성이 실행 중에도 가능하기 때문에 높은 유연성을 제공하고 소프트웨어 보다 훨씬 빠른 처리 속도를 보인다[12-13]. 또한 FPGAs 자체는 하드웨어로서 사이드 채널 공격을 쉽게 막을 수 있기 때문에 소프트웨어 구현보다 높은 안전도를 가진다[6,9]. 따라서 최근 암호시스템들은 ASIC 구현보다 FPGAs와 같은 재구성형 하드웨어 구현에 더 많은 노력을 기울이고 있다[6-9,11].

FPGAs 구현을 위한 연산기 설계에 있어 가장 중요한 설계 원칙은 전역신호 전파의 제거이다. FPGAs에서 전역신호는 단순한 와이어가 아니라 전파 지연시간을 갖는 라우팅 자원들에 의해 연결되어 진다. 일반적으로 ECC에서 m은 163 보다 크기 때문에 전역신호가 사용될 경우 심각한 속도 저하를 야기시킨다[14]. 따라서 전역 신호 전파가 없는 시스톨릭 구조는 비-시스톨릭 구조에 비해 FPGAs상에서 높은 성능을 보인다[13].

ECC에 사용되는 유한체는 GF(p), GF(p^m) 그리고 GF(2^m)이 있다(여기서 p는 소수이다). 이중 GF(2^m)은 0과 1을 원소로 같는 GF(2)의 m차원 확장 필드로 특히 하드웨어 구현에 적합하다. 가장 대표적인 GF(2^m)의 원소 표기법에는 정규기저(normal basis)와 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.

ECC에서 가장 중요한 연산은 kP이다. 여기서 k는 큰 정수이고 P는 타원곡선상의 한 포인트이다. kP를 연산하기 위해서는 또다시 포인트 덧셈과 배 연산이 필요한데 타원곡선이 GF(2^m)상에서 정의되고 affine 좌표계를 사용할 경우 포인트 덧셈은 GF(2^m)상에서 8번의 덧셈, 한번의 나눗셈, 한번의 곱셈 그리고 한번의 제곱으로 이루어지며, 배 연산은 6번의 덧셈, 한번의 나눗셈, 한번의 곱셈 그리고 한번의 제곱으로 이루어진다[2-3]. 여기서 덧셈은 비트별 XOR 연산이기 때문에 간단하게 구현이 된다. 반면 나머지 연산들은 계산 시간 및 하드웨어 비용 측면에서 복잡하다. 특히 나눗셈은 가장 복잡한 연산으로 실제 ECC의 성능을 좌우한다[9].

GF(2^m)상에서 나눗셈 연산을 하드웨어로 구현하기 위해 1) 선형방정식 집합의 해를 구하는 방법[15-16], 2) Fermat의 이론[17-18], 3) Euclid의 GCD 알고리즘[19-20]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 $A/B = AB^{-1} = AB^{2^m-2} = A(B(B(B\cdots B(B(B)^2)^2\cdots)^2)^2)^2$ 의 관계식을 이용하는데, 이 방법은 m번의 제곱연산과 약 $\log_2^{(m-1)}$번의 곱셈연산을 필요로 한다. 마지막 방법은 Euclide의 GCD 알고리즘을 이용하는 방법으로 GCD($G(x)$, $B(x)$) = 1 이라는 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 $U(x)\cdot B(x) \equiv 1 \bmod G(x)$가 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[20].

GF(2^m)상의 곱셈 알고리즘은 크게 LSB(Least Significant First) 우선 방식과 MSB(Least Significant First) 우선 방식이 있다[21]. LSB-우선 방식은 곱수의 LSB부터 처리되고, MSB-우선 방식은 MSB부터 처리된다. 이들 두 방식은 동일한 시간 및 면적 복잡도를 가진다.

이러한 나눗셈 및 곱셈 회로들은 비트-패러럴[17-18, 21-22] 혹은 비트-시리얼[15-16, 19-20] 구조를 가진다. 고속의 나눗셈 연산을 위해서는 비트-패러럴 구조가 적

절하나 ECC에서 필드의 크기 m은 적어도 163 이상이어야 하기 때문에 비트-패러럴 구조는 높은 면적 복잡도와 핀 개수 문제로 구현에 많은 어려움이 따를 뿐 아니라 비실용적이다. 따라서 ECC를 위해선 비트-시리얼 구조가 적합하다.

본 논문에서는 GF(2^m)상에서 표준 기저 표기법을 사용하여, 나눗셈 및 곱셈을 모두 수행할 수 있는 새로운 비트-시리얼 시스톨릭 어레이를 제안한다. 이를 위해 GF(2^m)상의 바이너리 확장 GCD 알고리즘과 MSB-우선 곱셈 방식으로부터 각각의 자료의존 그래프(Dependence Graph: DG)를 얻은 후, 곱셈 및 나눗셈을 모두 수행할 수 있는 새로운 DG를 설계한다. 이를 바탕으로 프로젝션을 통하여 일차원 신호흐름 그래프(Signal Flow Graph: SFG)를 얻은 후 컷-셋 시스톨릭화 기법(cut-set systolization technique)[23]을 적용하여 완전한 연산기를 얻는다. 제안된 어레이는 연속된 입력 데이타에 대해 나눗셈 모드에서 초기 $5m-2$ 클럭 사이클 지연 후 매번 m 클럭 사이클 비율로 나눗셈의 결과를 출력하며, 곱셈 모드에서 초기 $3m$ 클럭 사이클 지연 후 매번 m 클럭 사이클 비율로 곱셈의 결과를 출력한다. 본 논문에서 제안된 연산기를 동일한 입출력 형태를 가지는 기존의 비트-시리얼 시스톨릭 어레이들과 비교 분석한 결과 현저히 낮은 면적 복잡도를 가질 뿐만 아니라 거의 동일하거나 낮은 계산 지연시간을 가진다. 또한 제안된 연산기를 Altera사의 FPGA칩인 EP2A70 F1508C-7 디바이스에서 구현한 결과 GF(2^{571})상에서 최대 121MHz의 주파수에서 동작하였고, 52%의 칩 면적을 사용하였다. 따라서, 본 연구에서 제안된 산술 연산기는 FPGAs상에서 구현되는 타원곡선 암호프로세서의 나눗셈 및 곱셈 연산기로 매우 적합하다.

2. ECC를 위한 GF(2^m)상의 연산

ECC에서 가장 중요한 연산은 kP이다. 여기서 k는 큰 정수이고 P는 타원곡선 상의 한 포인터이다. kP를 계산하기 위해 우리는 포인트 P를 k번 더하면 된다[2-3]. GF(2^m)을 특성이 2인 유한체라하면, 무한정점 $\boldsymbol{O}$와 함께 식 (1)을 만족하는 모든 근의 집합은 GF(2^m)상의 non-supersingular 타원곡선 E(GF(2^m))이다. 여기서 $a_2, a_6 \in$ GF(2^m) 이고 $a_6 \neq 0$.

$$E : y^2 + xy = x^3 + a_2x^2 + a_6 \quad (1)$$

$P_1 = (x_1, y_1)$과 $P_2 = (x_2, y_2)$를 affine 좌표계에서 E(GF(2^m))상의 두 포인트라 하고, $P_1, P_2 \neq \boldsymbol{O}$ 그리고 $P_1 \neq -P_2$라고 가정하면, 두 포인트의 덧셈 $P_3 = (x_3, y_3) = P_1 + P_2$는 아래와 같이 계산된다.

If $P_1 \neq P_2$ (포인트 덧셈 연산)

$\lambda = (y_1 + y_2)/(x_1 + x_2)$,

$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a_2$

$y_3 = (x_1 + x_3)\lambda + x_3 + y_1$

If $P_1 = P_2$ (포인트 배연산)

$\lambda = y_1/x_1 + x_1$,

$x_3 = \lambda_2 + \lambda + a_2$

$y_3 = (x_1 + x_3)\lambda + x_3 + y_1$

위에 기술된 바와 같이, E(GF(2^m))상의 서로 다른 두 포인트 덧셈은 GF(2^m)상에서 8번의 덧셈, 한번의 나눗셈, 한번의 곱셈 그리고 한번의 제곱으로 이루어지며, 배 연산은 6번의 덧셈, 한번의 나눗셈, 한번의 곱셈 그리고 한번의 제곱으로 이루어진다. 여기서 덧셈은 비트별 XOR 연산이기 때문에 쉽게 구현이 되고, 제곱은 곱셈으로 대체 될 수 있다. 위의 포인트 덧셈 및 배 연산을 보면 데이타 의존 때문에 덧셈을 제외한 어떠한 연산도 동시에 수행 될 수 없다. 따라서 곱셈 및 나눗셈 연산기의 분리된 구현 보다 이들간의 하드웨어를 공유하는 것이 훨씬 바람직하다.

3. FPGAs 구현을 위한 GF(2^m)상의 새로운 연산기

3.1 나눗셈을 위한 GF(2^m)상의 새로운 DG

$A(x)$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.

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

$$B(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1x + b_0 \quad (3)$$

$$G(x) = x^m + g_{m-1}x^{m-1} + \cdots + g_1x + g_0 \quad (4)$$

$$P(x) = p_{m-1}x^{m-1} + p_{m-2}x^{m-2} + \cdots + p_1x + p_0 \quad (5)$$

GF(2^m)상의 나눗셈을 위해 [알고리즘 1]은 사용될 수 있다[24]. [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, $U(x) \cdot B(x) \equiv 1$ mod 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 $P(x) = A(x)/B(x)$ mod $G(x)$를 얻을 수 있다[24].

[알고리즘 1]의 VLSI 설계에 앞서 핵심연산 및 제어함수에 대해서 고려한다. [알고리즘 1]의 S와 R은 차수가 m인 다항식 그리고 U와 V는 차수가 m-1인 다항식으로 각각 표현할 수가 있으며, 각 다항식은 아래와 같다.

$$R = r_mx^m + r_{m-1}x^{m-1} + \cdots + r_1x + r_0 \quad (6)$$

$$S = s_mx^m + s_{m-1}x^{m-1} + \cdots + s_1x + s_0 \quad (7)$$

$$U = u_{m-1}x^{m-1} + u_{m-2}x^{m-2} + \cdots + u_1x + u_0 \quad (8)$$

$$V = v_{m-1}x^{m-1} + v_{m-2}x^{m-2} + \cdots + v_1x + v_0 \quad (9)$$

[알고리즘 1] GF(2^m)상의 나눗셈을 위한 바이너리 확장 GCD 알고리즘

Input: $G(x)$, $A(x)$, $B(x)$
Output: V has $P(x)=A(x)/B(x)$ mod $G(x)$
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
```

[알고리즘 1]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

$$R' = r'_m x^m + r'_{m-1}x^{m-1} + \cdots + r'_1 x + r'_0 \quad (10)$$

r_0, 식 (6) 그리고 식 (7)로부터 아래의 식들을 얻을 수 있다.

$$r'_m = 0 \quad (11)$$

$$r'_{m-1} = r_0 S_m = r_0 S_m + 0 \quad (12)$$

$$r'_i = r_0 S_{i+1} + r_{i+1},\ 0 \le i \le m-2 \quad (13)$$

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

$(U + V)$를 U''라 두면

$$U'' = u''_{m-1}x_{m-1} + \cdots + u''_1 x + u''_0 = U + V \quad (14)$$

r_0, 식 (8) 그리고 식 (9)로부터 우리는 아래의 식 (15)를 얻을 수 있다.

$$u''_i = r_0 v_i + u_i,\ 0 \le i \le m-1 \quad (15)$$

x가 기약다항식 $G(x)$의 근이기 때문에 식 (16)을 얻을 수 있다.

$$x^m = g_{m-1}x^{m-1} + g_{m-2}x^{m-2} + \cdots + g_1 x + g_0 \quad (16)$$

식 (16)으로부터 g_0는 항상 1이기 때문에 (17)와 같이 다시 쓸 수 있다.

$$1 = (x^{m-1} + g_{m-1}x^{m-2} + g_{m-2}x^{m-3} + \cdots + g_2 x + g_1)x \quad (17)$$

식 (17)으로부터, 양변에 x를 나누면, 우리는 아래의 식 (18)을 얻을 수 있다.

$$x^{-1} = x^{m-1} + g_{m-1}x^{m-2} + g_{m-2}x^{m-3} + \cdots + g_2 x + g_1 \quad (18)$$

U/x의 결과를 U'''라 두면

$$U''' = u'''_{m-1}x^{m-1} + \cdots + u'''_1 x + u'''_0 = U/x \quad (19)$$

식 (8)과 식 (18)을 식 (19)에 대입하면, 아래의 식들을 유도 할 수 있다.

$$u'''_{m-1} = u_0 = u_0 g_m + 0 \quad (20)$$

$$u'''_i = u_{i+1} + u_0 g_{i+1},\ 0 \le i \le m-2 \quad (21)$$

마지막으로 U''/x의 결과를 U'라 두면

$$U' = u'_{m-1}x^{m-1} + \cdots + u'_1 x + u'_0 = U''/x \quad (22)$$

식 (14)와 식 (19)로부터, U의 임시적인 결과를 아래와 같이 얻을 수 있다.

$$u'_{m-1} = r_0 v_0 + u_0 = (r_0 v_0 + u_0)g_m + r_0 0 + 0 \quad (23)$$

$$u'_i = (r_0 v_{i+1} + u_{i+1}) + (r_0 v_0 + u_0)g_{i+1},\ 0 \le i \le m-2 \quad (24)$$

[알고리즘 1]과 핵심연산으로부터 유도된 GF(2^m)상의 나눗셈연산을 위한 DG는 그림 1과 같다 (여기서 m=3). 일반적으로 그림 1의 DG는 $(2m-1)$개의 Type-1셀과 $(2m-1)\cdot m$개의 Type-2셀로 구성되며, Type-1 및 Type-2셀의 회로는 각각 그림 2와 그림 3과 같다. DG의 i번째 열은 알고리즘의 i번째 반복을 수행하며, 첫 번째 열은 $U(x)$, $R(x)$ 그리고 $G(x)$를 입력으로 받고,

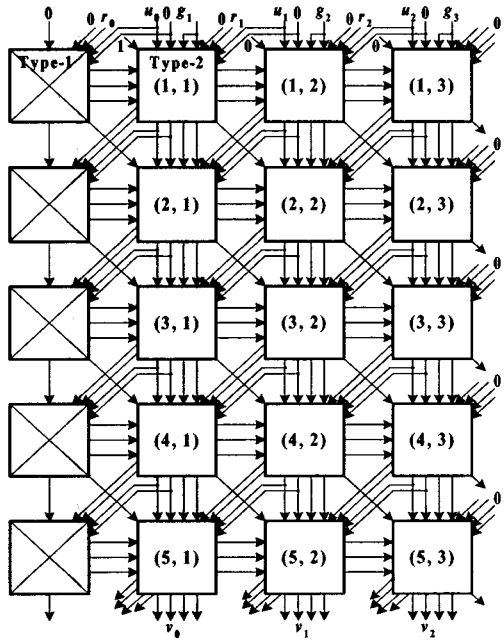


그림 1 나눗셈을 위한 GF(2^3)상의 DG

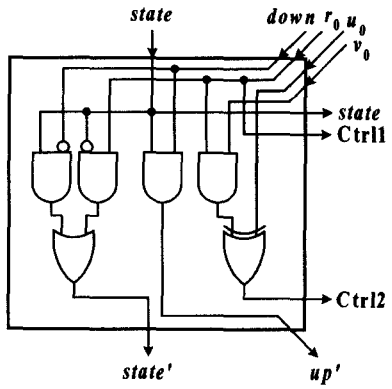


그림 2 그림 1의 Type-1셀의 회로도

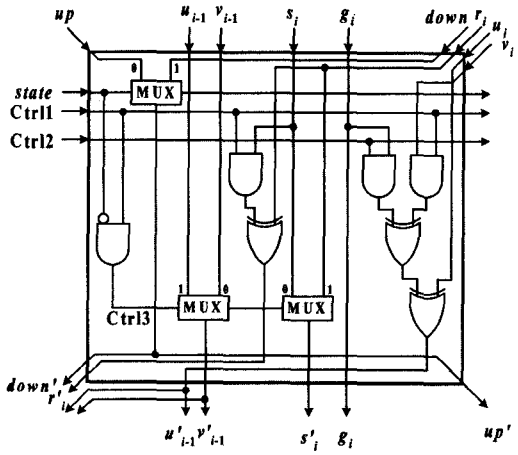


그림 3 그림 1의 Type-2셀의 회로도

마지막 열은 나눗셈 결과인 $V(x)$를 출력한다. 핵심 연산으로부터 r_0는 항상 0이고 s_0는 항상 1이기 때문에 그림 1의 DG에서 이들은 제거되었다.

제안된 알고리즘으로부터 Type-1셀은 현재와 다음 반복을 위한 제어신호들을 생성하고, Type-2셀은 하나의 제어신호를 포함하며, Type-1셀의 제어신호로부터 [알고리즘 1]의 *count* 및 나눗셈 연산을 수행한다.

각 셀에 대한 자세한 설명에 앞서 우리는 *count*의 구현에 대해 고려한다. [알고리즘 1]에서 *count*의 값은 최대 m까지 증가한다. 이를 구현하기 위해서는 Type-1셀에 $\log_2^{(m+1)}$-비트의 덧셈 및 뺄셈기를 추가하여야 한다. 이 경우, [23]의 투영절차에 의해 일차원 SFG를 얻으면, 각 처리기(Processing Element: PE)는 $\log_2^{(m+1)}$-비트의 덧셈 및 뺄셈기를 포함해야 한다. ECC와 같이 m이 크다면 이는 분명히 비실용적이다. 이 문제를 해결하기 위해 우리는 아래와 같이 *count*의 값을 계산한다.

*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, 이를 현재의 *count* 값이 n이라고 가정하면 된다. 따라서 그림 1에 있는 DG의 각 열에 m 비트 크기의 양방향 쉬프트 레지스터를 구현하기 위해 각각의 Type-2 셀에 2-to-1 멀티플렉서를 추가하고, (1, 1)의 Type-2 셀에 1을 입력하고, 첫 번째 열의 나머지 Type-2 셀에는 0을 입력한다. 이것을 현재의 *count* 값이 0이라고 간주하고, *state*로 *count*값을 증감시킨다. *count*의 구현 결과와 함께 Type-1 및 Type-2 셀의 기능에 대해 아래에 기술한다.

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

$$\text{Ctrl1} = (r_0 == 1) \tag{25}$$

$$\text{Ctrl2} = u_0 \text{ XOR } (v_0 \ \& \ r_0) \tag{26}$$

$$inc' = (state == 1) \ \& \ (dec == 1) \tag{27}$$

$$state = \overline{state}, \text{ if} \begin{cases} ((r_m == 1) \& (state == 0)) \text{ or} \\ ((dec == 1) \& (state == 1)) \end{cases} \tag{28}$$

또한 *state*가 1이 되고 *down*이 1이 되면, *count*가 0이 되는 것을 의미한다. 따라서 *state*는 다시 0이 되고, $up' = 1$이 된다. 또한 그 열의 모든 Type-2 셀의 $up' = down' = 0$이 되어 결국 알고리즘이 시작될 때와 같이 *count* = 0이 된다.

2) Type-2 셀: Type-1 셀로부터 *state*, Ctrl1 그리고 Ctrl2를 받아 각 다항식의 계수 및 *count* 값을 계산한다. 또한 [알고리즘 1]의 단계 5와 11을 수행하기 위해 식 (29)의 제어신호 Ctrl3을 생성한다. 그림 3에 나타나듯이 *state*가 0이면, *up*이 선택되어 *count* = *count* +1이 되고 그렇지 않으면 *down*이 선택되어 *count* = *count* -1이 된다.

$$\text{Ctrl3} = (state == 0) \ \& \ \text{Ctrl1} \tag{29}$$

3.2 GF(2^m)상의 MSB 우선 곱셈을 위한 DG

$A(x)$, $B(x)$ 그리고 $P(x)$는 GF(2^m)상의 원소이고, $G(x)$는 차수 m의 기약 다항식이라 하면, 곱셈 결과 $P(x) = A(x)B(x) \bmod G(x)$ 는 아래와 같은 방법으로 계산할 수 있다[21].

$$\begin{aligned} P(x) &= A(x)B(x) \bmod G(x) \\ &= \{\cdots[A(x)b_{m-1}x \bmod G(x) + A(x)b_{m-2}]x \bmod G(x) \\ &\quad + \cdots + A(x)b_1\}x \bmod G(x) + A(x)b_0 \end{aligned} \tag{30}$$

식 (30)으로부터 우리는 아래 식 (31)과 같이 $P(x)$의 임시적인 결과를 계산 할 수 있다. 여기서 $P^{(0)}(x) = 0$ 그리고 $1 \le i \le m$.

$$\begin{aligned} P^{(i)}(x) &= (P^{(i-1)}(x)x + b_{m-i}A(x)) \bmod G(x) \\ &= (P^{(i-1)}(x)x \bmod G(x)) + b_{m-i}A(x) \end{aligned} \tag{31}$$

식 (31)로부터 $(P^{(i-1)}(x)x \bmod G(x))$는 아래와 같이 계산될 수 있다.

$$(P^{(i-1)}(x)x \bmod G(x)) = \sum_{k=0}^{m-1} p_k^{(i-1)} x^{(k+1)}$$

$$= p_{m-1}^{(i-1)} x^m + \sum_{k=0}^{m-2} p_k^{(i-1)} x^{(k+1)} \quad (32)$$

식 (16)으로부터 $x^m = \sum_{k=0}^{m-1} g_k x^k$. 따라서 차수 감소를 위해 식 (32)에 $\sum_{k=0}^{m-1} g_k x^k$을 x^m에 대입하면, 아래의 식을 유도할 수 있다.

$$p_{m-1}^{(i-1)} \sum_{k=0}^{m-1} g_k x^{(k)} + \sum_{k=1}^{(i-1)} x(k)$$

$$= \sum_{k=0}^{m-1} (p_{m-1}^{(i-1)} g_k + p_{k-1}^{(i-1)}) x^k \ (\text{with } p_{-1}^{i-1} \equiv 0) \quad (33)$$

식 (31)과 식 (33)으로부터 우리는 아래의 MSB 우선 곱셈 알고리즘을 얻을 수 있다[21].

[알고리즘 2] GF(2^m)상의 MSB 우선 곱셈 알고리즘

Input: $G(x)$, $A(x)$, $B(x)$
Output: $P(x) = A(x) \,/\, B(x) \bmod G(x)$
1. $p_k^{(0)}=0$, **for** $0 \le k \le m-1$
2. $p_{-1}^{(i)}=0$, **for** $1 \le i \le m$
3. **for** i=1 **to** m **do**
4. **for** $k=m-1$ **to down to** 0 **do**
5. $p_k^{(i)} = p_{m-1}^{(i-1)} g_k + b_{m-i} a_k + p_{k-1}^{(i-1)}$
6. **end**
7. **end**
8. $P(x) = p^{(m)}(x)$

위의 알고리즘에서, $p_k^{(i)}$은 $P^{(i)}(x)$의 k번째 계수를 나타내고 a_k, b_k 그리고 g_k는 각각 $A(x)$, $B(x)$ 그리고 $G(x)$의 k번째 계수를 나타낸다.

[알고리즘 2]의 MSB 우선 곱셈 알고리즘으로부터 그림 4와 같은 GF(2^m)상의 곱셈을 위한 DG를 얻을 수 있다[22]. 일반적으로 곱셈을 위한 DG는 $m \times m$개의 기본셀들로 구성되며, 그림 4에서 m은 3이고 기본셀의 회로는 그림 5와 같다. i번째 열의 기본셀들은 $P^{(i)}(x)$의 계수들을 계산하고 곱셈결과 $P^{(i)}(x)$는 $P^{(m)}(x)$와 같다.

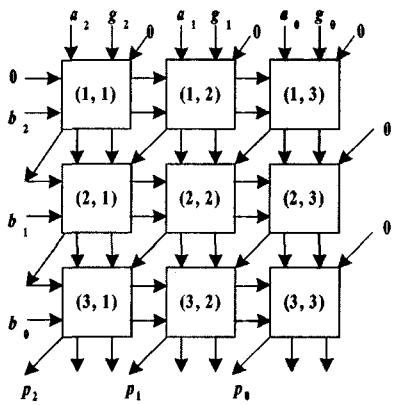


그림 4 GF(2^m)상의 MSB 우선 곱셈을 위한 DG[22]

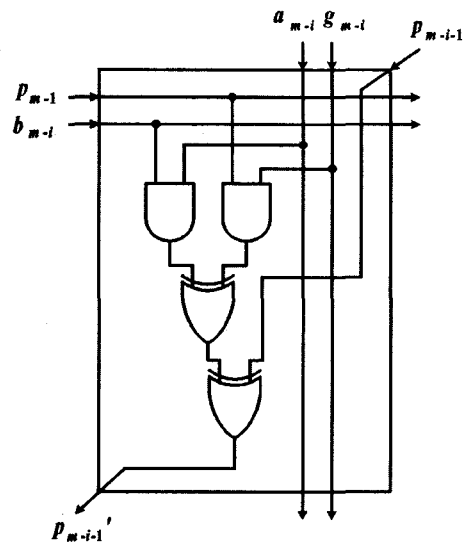


그림 5 그림 4의 기본셀 회로도[22]

3.3 GF(2^m)상의 나눗셈 및 곱셈 모두를 위한 DG

그림 1의 m개의 Type-1셀과 $m \times m$개의 Type-2셀을 가지고 곱셈을 수행하기 위해 우리는 그림 1의 DG를 수정하며 그 결과는 그림 6과 같다. 또한 그림 6의

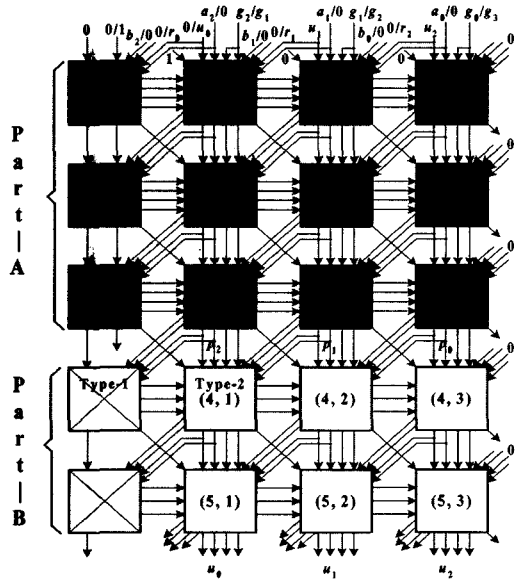


그림 6 GF(2^3)상의 나눗셈 및 곱셈 모두를 위한 DG

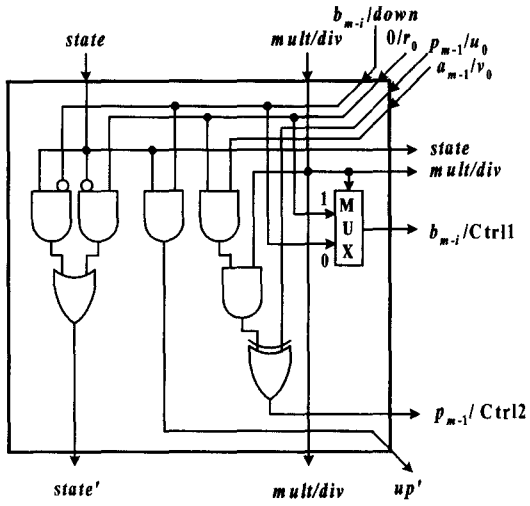


그림 7 그림 6의 Modified Type-1셀의 회로도

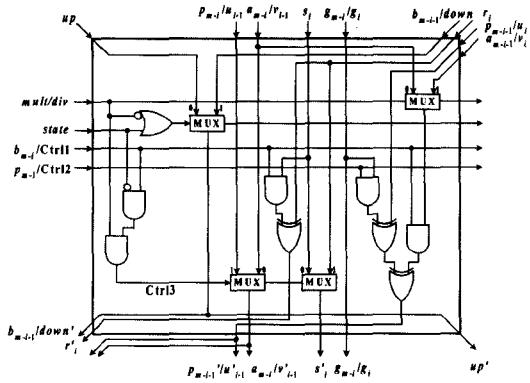


그림 8 그림 6의 Modified Type-2셀의 회로도

Modified Type-1셀 및 Modified Type-2셀의 회로는 각각 그림 7과 그림 8에 주어진다. 그림 6에 기술되듯이 새로운 DG는 m개의 Modified Type-1, $m\times m$개의 Modified Type-2, m-1개의 Type-1 그리고 $(m-1)\times m$개의 Type-2셀들로 구성된다. 즉 Part-A에서는 나눗셈 및 곱셈 모두를 수행하고 Part-B에서는 나눗셈만 수행한다. Part-A에서 곱셈을 수행하기 위해 우리는 Modified Type-1셀에 *mult/div* 신호와 하나의 2-to-1 AND 게이트 그리고 하나의 2-to-1 멀티플렉서를 추가한다. 그리고 Modified Type-2셀에 *mult/div* 신호, 하나의 2-to-1 OR 게이트, 하나의 2-to-1 AND 게이트 그리고 하나의 2-to-1 멀티플렉서를 추가한다. 그림 7에 보이듯이 곱셈을 수행할 때 *mult/div*에 0을 입력하면 b_{m-i}/Ctrl1은 b_{m-i} 그리고 p_{m-1}/Ctrl2은 p_{m-1}을 각각 가진다. 그림 8에서도 역시 곱셈을 수행할 때 p_{m-i} 대신 a_{m-i}이 선택되고, b_{m-i-1}은 왼쪽으로 전파되고 그리고 a_{m-i-1} 대신 a_{m-i}이 선택된다. 결과적으로 우리는 그림 6의 Part-A를 가지고 그림 4의 곱셈을 수행할 수 있다. 또한 그림 7과 그림 8로부터 나눗셈을 수행할 때 *mult/div*에 1을 입력하면 그림 1의 나눗셈 DG와 동일한 기능을 수행한다. 따라서 그림 6의 DG는 GF(2^m)상에서 나눗셈 및 곱셈 모두를 수행할 수 있다.

3.4 FPGAs 구현을 위한 GF(2^m)상의 새로운 산술 연산기

그림 6의 DG를 [23]의 투영절차에 따라 오른쪽으로 투영하면 그림 9과 같은 일차원 SFG를 얻을 수 있으며, 그림 9의 Type-1 처리기(Processing Element: PE) 및 Type-2처리기의 회로는 각각 그림 10와 그림 11에 주어진다. 그림 9에 있어 '•'은 1-비트 래치를 나타낸다. 그림 9의 SFG 어레이는 길이 m인 011…11의 비트열에 의해 제어된다. 프로젝션 절차에 따라 Type-1 PE는 그림 6의 Modified Type-1셀과 Modified Type-2셀을 Type-2 PE는 Type-1셀과 Type-2셀을 각각 포함 해야한다. 그림 6의 DG로부터, 각각의 열 반복에서 Ctrl1, Ctrl2, 그리고 *state*는 i번째 열의 모든 Modified Type-2셀들과 Type-2셀들에 전파되어야 하기 때문에 우리는 각각의 PE에 세 개의 2-to-1 멀티플렉서와 세 개의 1-비트 래치를 추가한다. 또한 그림 6 DG의 가장 오른쪽 셀에는 4개의 0이 입력되기 때문에 4개의 2-to-1 AND게이트를 추가하였으며, 제어 논리가 0일때 각각 0을 생성한다. 참고로 *mult/div* 신호 역시 i번째 열의 모든 Modified Type-2셀들에 전파되어야 하지만 이 신호는 각각의 연산 모드에서 1혹은 0의 상수 값을 가지기 때문에 생략될 수 있다.

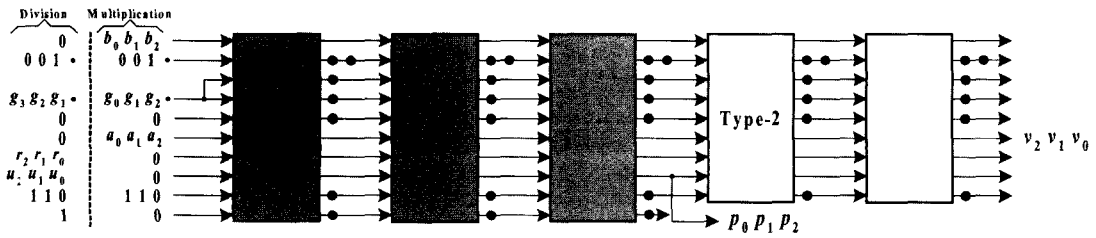


그림 9 GF(2^3)상의 나눗셈을 위한 일차원 SFG 어레이

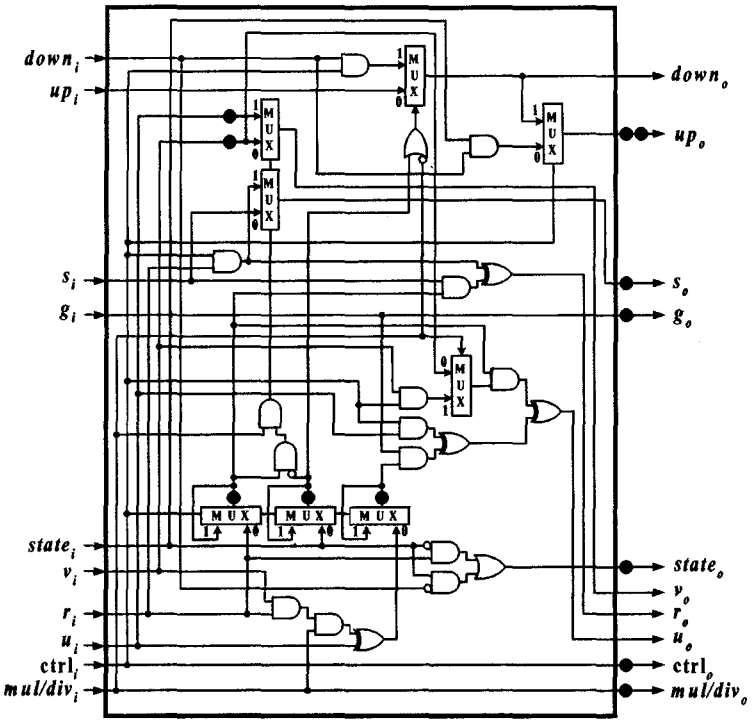


그림 10 그림 9의 Type-1 PE 회로도

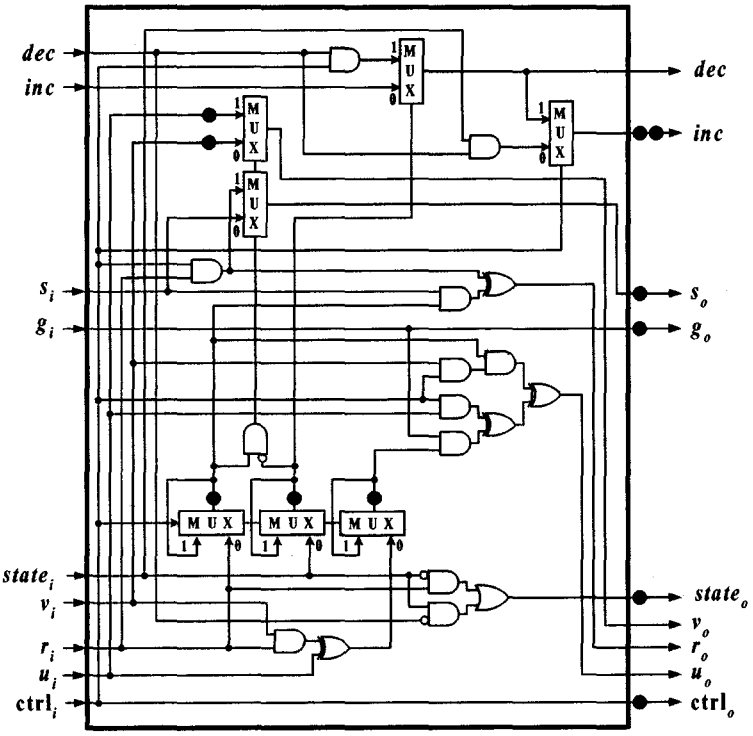


그림 11 그림 9의 Type-2 PE 회로도

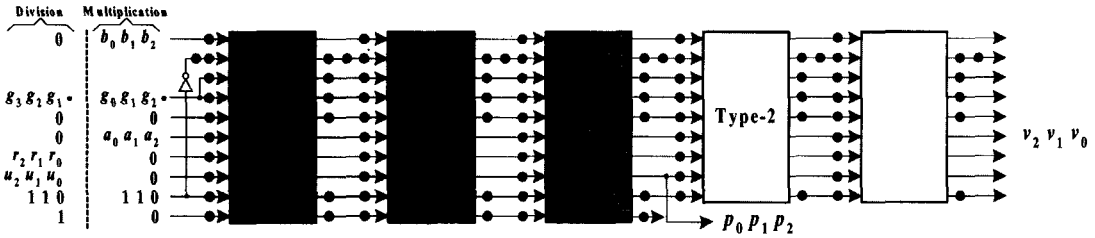


그림 12 GF(2^3)상의 나눗셈 및 곱셈 모두를 위한 비트-시리얼 시스톨릭 어레이

GF(2^m)상의 완전한 비트-시리얼 시스톨릭 어레이를 얻기 위해 컷-셋 시스톨릭화 기법[23]을 적용하면, 그림 12와 같은 구조를 얻을 수 있다. 그림 12에 보이듯이 나눗셈을 위해선 LSB부터 입력되고 LSB부터 출력되며, 곱셈을 위해선 MSB부터 입력되고 MSB부터 출력된다. 또한 그림 12의 연산기는 연속된 입력 데이타에 대해 나눗셈 모드에서 초기 $5m$-2 클럭 사이클 지연 후 매번 m 클럭 사이클 비율로 나눗셈의 결과를 출력하며, 곱셈 모드에서 초기 $3m$ 클럭 사이클 지연 후 매번 m 클럭 사이클 비율로 곱셈의 결과를 출력한다. 즉 *mult/div*에 1을 인가하면 나눗셈을 0을 인가하면 곱셈을 수행한다.

3.5 제안된 시스톨릭 어레이의 FPGA 구현

본 논문에 제안된 GF(2^m)상의 새로운 연산기의 FPGA 구현 및 기능 검증을 위해 VHDL로 회로를 기술하였고, Synopsys사의 합성 툴(FPGA-Express Version : 2000.11-FE 3.5.1)을 사용하여 회로를 합성, Net-list 파일을 추출한 후, Mento Graphics사의 Design View(VHDL Chipsim)를 이용하여 시뮬레이션 하였다. 또한 Altera사의 Quartus II 1.1을 이용하여 Place & Route 과정을 거친 후, 타이밍 및 칩 사용율에 대해 분석하였다. 여기서 Altera사의 300만 게이트급인 EP2A70F1508C-7을 대상 디바이스로 선택하였다. 그림 13과 그림 14는 GF(2^8)상에서 각각 곱셈 및 나눗셈에 대한 시뮬레이션 결과이며, 아래의 연속된 입력 데이타 집합을 사용하였다.

$A(x)/U(x) = (x^5 + x^3 + x + 1,\ x^6 + x^2 + x + 1)$ (33)

$B(x)/R(x) = (x^6 + x^3 + x^2 + x,\ x^5 + x^4 + x^3 + x^2 + x + 1)$ (34)

$G(x) = (x^8 + x^4 + x^3 + x^2 + 1,\ x^8 + x^4 + x^3 + x^2 + 1)$ (35)

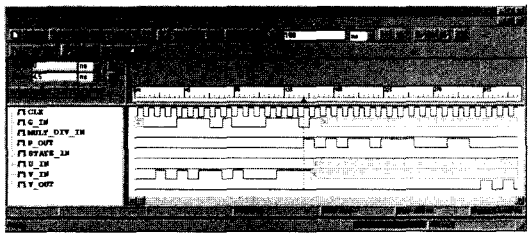


그림 13 GF(2^8)상의 곱셈에 대한 시뮬레이션 결과

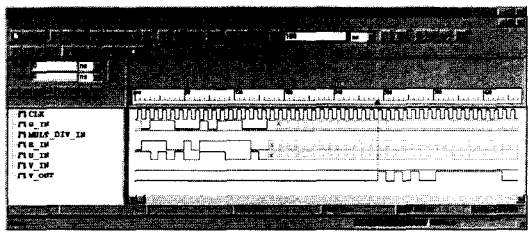


그림 14 GF(2^8)상의 나눗셈에 대한 시뮬레이션 결과

그림 13과 그림 14에 나타나듯이 곱셈의 경우 초기 $3m$ 클락 사이클 지연 후 m 사이클마다 정확한 곱셈 결과인 $P(x) = (x^7 + x^5 + x^3 + x^2 + 1,\ x^7 + x^6 + x^2 + x)$을 출력하고 나눗셈의 경우 초기 $5m$-2 클락 사이클 지연 후 m 사이클마다 정확한 나눗셈 결과 $V(x) = (1 + x^2 + x^4 + x^7,\ 1 + x + x^2 + x^3 + x^4 + x^5 + x^6)$을 출력한다. 또한 본 연구에서는 다양한 m(최대 571)에 대해서도 시뮬레이션을 수행하였으며, 그때마다 동일한 처리율로 정확히 동작함을 확인 하였다. 참고로 571은 NIST[25]에서 권고하는 필드 크기 중 가장 큰 값이다.

표 1에 제안된 연산기의 FPGA 구현 결과에 대해 요

표 1 제안된 연산기의 FPGA 구현 결과

항목 \ m	163	233	409	571
LE 갯수	9920	14209	24909	34950
최대 동작 주파수(MHz)	138.87	127.99	127.99	121.11
칩 사용율(%)	14.76	21.21	21.21	52.01

※ LE(Logic Element)는 하나의 4-to-1 룩업테이블, 하나의 플립플롭, 고속 캐리 로직 그리고 프로그램 가능한 멀티플렉서들로 구성되어 있다[26].

약하였다. 표 1에 기술되듯이 제안된 연산기는 전역신호 전파가 없는 시스톨릭 구조이기 때문에 큰 필드 크기에 대해서도 높은 동작 주파수를 얻을 수 있다. 뿐만 아니라 571비트에 대해서도 52%의 칩 사용율을 보인다. 따라서 타원곡선 암호프로세서를 위한 콘트롤 및 래지스터 파일 부분을 추가하더라도 하나의 칩에 구현이 가능하다.

4. 성능 분석

본 논문에서 제안한 연산기를 동일한 입출력을 가지는 기존의 비트-시리얼 시스톨릭 어레이들과 비교하였으며, 표 2에 그 결과를 요약하였다. 표 2에 기술된 바와 같이 기존의 어레이들이 $O(m^2)$ 혹은 $O(m \cdot (\log_2 m))$의 면적 복잡도를 가지는 반면 제안된 연산기는 $O(m)$의 면적 복잡도를 가진다. 또한 제안된 구조는 $O(m \cdot (\log_2 m))$의 면적 복잡도를 가지는 나눗셈기에 비해 훨씬 낮은 계산 지연시간을 가진다. [15-16]의 연산기들은 제안된 연산기 보다 T_{AND2} 만큼의 낮은 최대 처리기 지연시간을 가지지만 $O(m^2)$의 면적 복잡도 때문에 ECC의 구현에는 적합하지 않다. 또한 제안된 연산기는 [20]에 비해 $(T_{XOR2}+T_{MUX2})$만큼의 낮은 최대 처리기 지연시간을 가질 뿐만 아니라 계산 지연시간도 $3m+1$ 만큼 줄인다. 더욱이 제안된 어레이는 기존의 연산기와 달리 곱셈도 수행할 수 있다. 따라서 본 연구에서 제안된 어레이는 기존의 연구 결과들에 비해 훨씬 효율적이다.

5. 결 론

본 논문에서는 바이너리 확장 GCD 알고리즘과 MSB 우선 곱셈 방식으로부터 ECC 프로세서의 FPGAs 구현을 위한 GF(2^m)상의 새로운 비트-시리얼 시스톨릭 어레이를 제안하였다. 제안된 어레이는 GF(2^m)상에서 나눗셈 및 곱셈을 모두 수행할 수 있다. 즉 연속된 입력 데이타에 대해 나눗셈 모드에서 초기 $5m-2$ 클럭 사이클 지연 후 매번 m 클럭 사이클 비율로 나눗셈의 결과를 출력하며, 곱셈 모드에서 초기 $3m$ 클럭 사이클 지연 후 매번 m 클럭 사이클 비율로 곱셈의 결과를 출력한다.

제안된 어레이의 성능을 평가하고 그 기능을 검정하기 위해 Altera사의 FPGA칩인 EP2A70F1508C-7 디바이스를 사용하여 구현하였으며, 시뮬레이션을 통해 정확히 동작함을 확인 하였다. 또한 Altera사의 Quartus II 1.1을 이용하여 타이밍 및 칩 사용율을 분석한 결과 큰 필드 크기에 대해서도 높은 동작 주파수를 얻었으며, 낮

표 2 GF(2^m)상의 나눗셈을 위한 비트-시리얼 시스톨릭 어레이들과의 성능 비교

항목\회로	[15]	[16]	[20]	제안된 어레이
처리량 (1 /cycles)	$1/(2m-1)$	$1/m$	$1/m$	$1/m$
지연시간 (cycles)	$7m-3$	$5m-1$	$8m-1$	$5m-2$
최대 처리기 지연시간	$T_{AND2}+T_{XOR2}+T_{MUX2}$	$T_{AND2}+T_{XOR2}+T_{MUX2}$	$2T_{AND2}+2T_{XOR2}+2T_{MUX2}$	$2T_{AND2}+T_{XOR2}+T_{MUX2}$
회로 구성요소들	AND_2: $3m^2+3m-2$ XOR_2: $1.5m^2+1.5m-1$ MUX_2: $3m^2+m-2$ Latch: $6m^2+8m-4$	AND_2:$2m-1$ OR_2:$3m$ XOR_2: $0.5m^2+1.5m-1$ MUX_2: $0.5m^2+3.5m-2$ Latch: $2.5m^2+14.5m-6$	Inverter:$2m$ AND_2:$26m$ XOR_2:$11m$ MUX_2:$35m+2$ FILO(m-bit):4 Latch: $46m+4m(\log_2(m+1)$ zero-check $(\log_2(m+1)$-bit):$2m$ adder/subtractor $(\log_2(m+1)$-bit):$2m$	Inverter : $7m-2$ AND_2 : $26m-12$ OR_2 : $3m-1$ XOR_2 : $8m-4$ MUX_2 : $15m-7$ Latch : $44m-24$
면적 복잡도	$O(m^2)$	$O(m^2)$	$O(m(\log_2 m))$	$O(m)$
연산	나눗셈	나눗셈	나눗셈	나눗셈 및 곱셈

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_{AND_i}:the propagation delay through one AND_i gate
T_{XOR_i}:the propagation delay through one XOR_i gate
T_{MUX_i}:the propagation delay through one MUX_i gate
$T_{zero-detector}$: the propagation delay of $(\log_2 m+1)$-bit zero-detector

은 칩 사용율을 나타내었다. 또한 기존의 나눗셈기들이 $O(m^2)$ 혹은 $O(m \cdot (\log_2 m))$의 면적 복잡도를 가지는 반면 제안된 연산기는 $O(m)$의 면적 복잡도를 가지며, $O(m \cdot (\log_2 m))$의 면적 복잡도를 가지는 나눗셈기에 비해 훨씬 낮은 계산 지연시간 및 최대 처리기 지연시간을 가진다.

더욱이 제안된 어레이는 단방향의 신호흐름을 가지고, 기약다항식의 선택에 어떠한 제약도 두지 않을 뿐 아니라 매우 규칙적이고 묘듈화 하기 쉽기 때문에 필드 크기 m에 대하여 높은 확장성 및 유연성을 제공한다. 따라서, 본 연구에서 제안된 GF(2^m)상의 산술 연산기는 재구성형 하드웨어 상에서 구현되어지는 타원곡선 암호프로세서의 나눗셈 및 곱셈 연산기로 매우 적합하다.

참 고 문 헌

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

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

[3] M. Rosing, *Implementing Elliptic Curve Cryptography*, Manning, 1999.

[4] D. Hankerson, J. L. Hernandez, and A. Menezes, "Implementation of Elliptic Curve Cryptography Over Binary Fields," *CHES 2000*, LNCS 1965, Springer-Verlag, 2000.

[5] D. Bailey and C. Paar, "Efficient Arithmetic in Finite Field Extensions with Application in Elliptic Curve Cryptography," *J. of Cryptology*, vol. 14, no. 3, pp. 153-176, 2001.

[6] L. Gao, S. Shrivastava and G. E. Solbelman, "Elliptic Curve Scalar Multiplier Design Using FPGAs," *CHES 2000*, LNCS 1717, Springer-Verlag, 1999.

[7] G. Orlando and C. Parr, "A High Performance Reconfigurable Elliptic Curve Processor for GF (2^m)," *CHES 2000*, LNCS 1965, Springer-Verlag, 2000.

[8] M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi, and J. Teich, "Reconfigurable Implementation of Elliptic Curve Crypto Algorithms," *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS02)*, pp. 157-164, 2002.

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

[10] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, "An Implementation for Elliptic Curve Cryptosystems Over F_2^{155}," *IEEE J. Selected Areas in Comm.*, vol. 11, no. 5, pp. 804-813, June 1993.

[11] T. Blum and C. Paar, "High Radix Montgomery Modular Exponentiation on Reconfigurable Hardware," *IEEE Trans. Computers.*, vol. 50, no. 7, pp. 759-764, July 2001.

[12] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171-210, June 2002.

[13] R. Tessier and W. Burleson, "Reconfigurable Computing for Digital Signal Processing: A Survey," *J. VLSI Signal Processing*, vol. 28, no. 1, pp. 7-27, May 1998.

[14] S.D. Han, C.H. Kim, and C.P. Hong, "Characteristic Analysis of Modular Multiplier for GF(2^m)," *Proc. of IEEK Summer Conference 2002*, vol. 25, no. 1, pp. 277-280, 2002.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] S.K. Jain, L. Song, and K.K. Parhi, "Efficient Semi-Systolic Architectures for Finite Field Arithmetic," *IEEE Trans. VLSI Syst.*, vol. 6, no. 1, pp. 101-113, Mar. 1998.

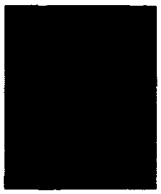
[22] C. L. Wang and J. L. Lin, "Systolic Array Implementation of Multipliers for Finite Field GF(2^m)," *IEEE Trans. Circuits and Syst.*, vol. 38, no. 7, pp. 796-800, July 1991.

[23] S. Y. Kung, *VLSI Array Processors*, Englewood Cliffs, NJ: Prentice Hall, 1988.

[24] 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.

[25] NIST, Recommended elliptic curves for federal government use, May 1999. http://csrc.nist.gov/encryption.

[26] Altera, APEX*TMII* Programable Logic Device *Family Data Sheet, Aug. 2000. http://www.altera.com/literature/lit-ap2.html*



김 창 훈
2001년 2월 대구대학교 컴퓨터정보공학부, 학사. 2003년 2월 : 대구대학교 컴퓨터정보공학과, 석사. 2004년 3월~현재 대구대학교 컴퓨터정보공학과, 박사과정 관심분야는 암호 시스템, Embedded System, 재구성형 컴퓨팅

권 순 학
1990년 2월 KAIST 수학과, 학사. 1992년 2월 서울대학교 수학과, 석사. 1997년 5월 Johns Hopkins University, 박사 1998년 3월~현재 성균관대학교 수학과, 조교수. 관심분야는 정수론, 암호이론, 회로이론

홍 춘 표
1978년 2월 : 경북대학교 전자공학과, 학사. 1986년 12월 Georgia Institute of Technology ECE, 석사. 1991년 12월 Georgia Institute of Technology ECE, 박사. 1994년 9월~현재 대구대학교 정보통신공학부, 교수. 관심분야는 DSP 하드웨어 및 소프트웨어, 컴퓨터 구조, VLSI 신호처리, Embedded System

유 기 영
정보과학회논문지 : 시스템 및 이론
제 31 권 제 1 호 참조