논문 2005-42SD-8-3

멀티 세그먼트 곱셈 기반 저비용 타원곡선 암호 프로세서

(Low-Cost Elliptic Curve Cryptography Processor Based On Multi-Segment Multiplication)

이 동 호*

(Dong-Ho LEE)

요 약

본 논문에서는 효율적인 $GF(2^m)$ 멀티 세그먼트 곱셈 연산 구조를 제안하고 제안된 구조의 타원곡선 암호 프로세서 설계 응용을 연구한다. 제안된 멀티 세그먼트 곱셈 연산 구조는 유한체 크기 m에 비하여 아주 작은 워드 조합 곱셈기를 이용하여 부분곱을 계산하고 거의 모든 내부 버스는 워드 크기이며 m 비트 멀티플렉서와 m 비트 레지스터를 하나만 사용한다. 따라서 조합 곱셈기의 워드 크기 w를 줄이고 세그먼트 수 k를 크게 하여 전체 데이터패스 자원 사용량이 최소화할 수 있다. 제안된 곱셈기는 디지트 시리얼 곱셈기로 구현된 ECC 프로세서와 비교할 때 이론적으로 자원 효율성이 우수하다. 암호 프로세서의 자원 사용량은 구현에 필요한 기본 하드웨어 요소 수뿐만 아니라 구성 요소들의 배치와 연결 상태에도 의존한다. 제안된 프로세서의 실질적인 자원사용량을 디지트 시리얼 곱셈기 기반 암호 프로세서와 비교하기 위하여 두 종류의 프로세서를 FPGA 상에 구현하였다. 실험 결과로 제안된 멀티 세그먼트 곱셈기 기반 ECC 프로세서는 유사한 성능을 가지는 디지트 시리얼 곱셈기 기반 ECC 프로세서보다 자원 사용면에서 2배 정도 우수함을 보였다.

Abstract

In this paper, we propose an efficient $GF(2^m)$ multi-segment multiplier architecture and study its application to elliptic curve cryptography processors. The multi-segment based ECC datapath has a very small combinational multiplier to compute partial products, most of its internal data buses are word-sized, and it has only a single m bit multiplexer and a single m bit register. Hence, the resource requirements of the proposed ECC datapath can be minimized as the segment number increases and word-size is decreased. Hence, as compared to the ECC processor based on digit-serial multiplication, the proposed ECC datapath is more efficient in resource usage. The resource requirement of ECC processor implementation depends not only on the number of basic hardware components but also on the complexity of interconnection among them. To show the realistic area efficiency of proposed ECC processors, we implemented both the ECC processors based on the proposed multi-segment multiplication and digit serial multiplication and compared their FPGA resource usages. The experimental results show that the proposed multi-segment multiplication method allows to implement ECC coprocessors, requiring about half of FPGA resources as compared to digit serial multiplication.

Keywords: ECC, Finite Field Multiplication, Computation Architecture

I. 서 론

저비용 무선 임베디드 시스템의 보안 요구가 커짐에 따라 공개키 암호(public key cryptography) 알고리즘의 저비용 구현이 매우 중요하게 되었다. 타원곡선 암호(elliptic curve cryptography) 알고리즘은 유선 네트워크에서 널리 사용되는 RSA 알고리즘보다 소프트웨어 혹은 하드웨어적으로 저비용으로 구현할 수 있으므로 무선 임베디드 시스템의 공개키 알고리즘으로 널리 사용될 것으로 기대된다. 특히 유비쿼터스 컴퓨팅의 경우 저비용 임베디드 마이크로프로세서를 사용하게 되므로 ECC 알고리즘을 마이크로프로세서 상에서 소프트

* 평생회원, 경북대학교 전자전기컴퓨터학부
(School of Electrical Engineering and Computer Science, Kyungpook National University)
접수일자: 2005년2월19일, 수정완료일: 2005년7월11일

웨어로 구현하기 어려운 경우가 많다. 또한 시스템 온 칩 설계 기술의 발달로 ECC 전용 연산 하드웨어를 내장하기가 용이하고 전용 하드웨어를 사용하면 ECC 계산에 필요한 전력 소모가 마이크로프로세서를 사용하는 것보다 적다. 따라서 저비용 무선 임베디드 시스템의 보안 알고리즘의 구현에서 ECC 계산을 지원하는 전용 하드웨어 프로세서의 구현은 매우 중요하다.

ECC 기반의 공개키 암호 시스템에서 스칼라 곱 연산이 가장 긴 시간이 소비된다. 따라서 ECC를 위한 보조 프로세서는 주로 스칼라 곱 연산을 하드웨어적으로 수행하는 역할을 수행한다. 이제까지 ECC 스칼라 곱 알고리즘의 효율적인 구현에 대하여 많은 연구가 수행되어 왔다. ECC 스칼라 곱 알고리즘은 유한체 덧셈, 곱셈, 나눗셈을 사용하여 구현된다. 나눗셈 구현에는 많은 하드웨어가 사용된다. 나눗셈은 Fermat의 정리를 이용하면 곱셈을 이용하여 구현할 수 있다. 사영 좌표계(projective coordinates)를 사용하면 스칼라 곱 수행에 필요한 유한체 나눗셈의 수를 1회로 제한할 수 있다. 많은 ECC 하드웨어 구현이 사영 좌표계를 사용한다.

$GF(2^m)$ 유한체 연산은 정수 연산과 달리 캐리 전송이 없으므로 유한체 연산기 구현에 요구되는 자원은 정수 연산의 경우보다 아주 작다. 암호 하드웨어 구현에 널리 쓰이는 비트 serial 연산기는 각 비트당 AND 게이트 2개 XOR 게이트 2개가 소요된다. 따라서 비트 serial 곱셈기에 기반을 둔 ECC 프로세서는 자원 소요량 면에서 아주 우수하다. 디지트 serial 곱셈기는 비트 serial 곱셈기에 병렬 연산 기법을 적용하여 성능을 향상한 곱셈기이다. 디지트 시리얼 곱셈을 이용하면 m 비트 곱셈에 m*m/d 클록이 소용되며 하드웨어 구현에 필요한 AND 및 XOR 게이트는 비트 시리얼 곱셈기의 경우 보다 d배 증가한다.

멀티 세그먼트 곱셈은 작은 w × w 곱셈기를 이용하며 큰 유한체 곱셈을 수행한다. 멀티 세그먼트 곱셈기의 경우 유한체 크기 m은 세그먼트 수 k와 조합 곱셈기 워드 크기 w의 곱으로 표현된다. m이 w의 배수가 아닌 경우 k는 $m/w \leq k$를 만족하는 최소로 한다. 이 경우 연산에서 하드웨어 레지스터들은 일부만 사용된다. 저비용 유한체 연산기를 설계하기 위해서는 세그먼트 수 k가 큰 경우에도 효율적인 멀티 세그먼트 곱셈 하드웨어 구조를 사용하여야 한다. 본 논문에서는 Ernst, Jung, Madlener 등이 제안한 멀티 세그먼트 곱셈 하드웨어 구조를 확장하여 w가 작고 k가 큰 경우에도 효율적인 멀티 세그먼트 곱셈 하드웨어 구조를 제안하였다[9].

확장된 멀티 세그먼트 곱셈기를 이용한 ECC 프로세서의 효율성을 보이기 위하여 새로운 멀티 세그먼트 곱셈기 기반 데이터패스 구조로 구현한 ECC 프로세서와 디지트 시리얼 곱셈기로 구현된 ECC 프로세서와의 비교 실험을 하였다. 설계 검증은 Altera EXCALIBUR와 Xilinx Vertex-II FPGA를 포함한 시스템 온 칩 개발 환경에서 수행되었다. 두 프로세서 구조의 자원 사용량은 FPGA 논리 요소 사용량으로 비교하였다. 실험 결과 제안된 멀티 세그먼트 곱셈기가 ECC 프로세서의 구현에 디지트 serial 곱셈기보다 경제적임을 보였다.

II. 관련 연구

참고문헌 [1, 2]에는 최근까지 연구된 $GF(2^m)$ 기저 스칼라 곱 알고리즘과 $GF(p)$ 기저 스칼라 곱 알고리즘의 소프트웨어 구현에 대한 연구가 모두 소개되어 있다. 스칼라 곱의 하드웨어 구현에도 이들 알고리즘을 사용할 수 있으나 소프트웨어 구현의 경우와는 달리 VLSI 구현 특성이 우수한 알고리즘을 사용하여야 한다. 먼저 하드웨어 구현에는 이진 확장 유한체(binary extension field)인 $GF(2^m)$를 사용하는 것이 소수 유한체(prime field)인 $GF(p)$를 사용하는 것보다 유리하다. $GF(p)$ 연산에는 정수 곱 연산과 Montgomery reduction과 같은 modular reduction 연산이 필요하다. ECC의 하드웨어 구현에서 유한체 연산기 구현이 가장 중요하므로 $GF(p)$ ECC 보조 프로세서의 하드웨어 구조는 RSA의 경우와 유사한 점이 많다.

$GF(2^m)$ 유한체 연산의 구현 방법은 크게 유한체 원소의 표현 방법에 따라 구별된다. 참고문헌 [3, 4]에는 정규 기저(normal basis)를 사용하여 ECC 알고리즘을 소프트웨어/하드웨어로 구현하는 방법이 소개되어 있다. 정규 기저 알고리즘은 제곱 계산을 간단한 쉬프트 연산으로 수행할 수 있는 장점이 있으나 곱셈과 나눗셈 구현이 복잡하여 최근에는 다항식 기저(polynomial basis) 표현 방법을 주로 사용한다[1]. 참고문헌 [1]에 소개된 많은 알고리즘이 사영 좌표계를 사용하고 있다. Lopez와 Dahab은 다항식 기저와 사영 좌표계를 이용한 하드웨어 구현에 효율적인 ECC 스칼라 곱 알고리즘에

대한 연구하였다[5]. 다항식 기저 알고리즘의 곱셈에 대하여 많은 연구가 있었으나 Song과 Parhi의 디지트 serial 곱셈 알고리즘을 기반으로 하는 ECC 스칼라 곱 알고리즘이 하드웨어 구현에 널리 사용된다[6]. 이 알고리즘은 비트 serial 곱셈 알고리즘의 간단한 확장으로 디지트 크기 D에 따라 연산 속도가 비례하여 증가한다. Orlando와 Paar는 디지트 serial 연산과 FPGA의 reprogramming을 이용한 고속 ECC 스칼라 곱 연산 프로세서를 연구하였다[7].

본 연구와 직접 관련된 연구로 먼저 Savas, Tenca, Koc 등은 scalable 구조를 이용하여 GF(p)와 $GF(2^m)$를 모두 사용할 수 있는 통합 아키텍처에 대하여 연구하였다[8]. 이 논문에서 제안된 scalable 구조는 $GF(2^m)$ 유한체 연산기의 자원 소요량을 줄이는 데에는 효과가 작을 것으로 보인다. 그 이유는 $GF(2^m)$ 연산기의 경우 GF(p) 연산기와는 달리 연산보다 데이터 이동에 많은 자원이 사용되며 이들 저자들이 제안한 구조는 이러한 점에서 비효율적이기 때문이다[8].

Ernst, Madlener 등은 Karatsuba 멀티 세그먼트 곱셈기를 이용하여 저비용 ECC 프로세서를 설계하는 방법을 연구하였다[9]. 기본적인 멀티 세그먼트 곱셈 알고리즘은 k*k 회의 서브워드 곱(subword product)을 사용한다. Karatsuba 멀티 세그먼트 곱셈(KMM) 방법을 사용하면 이 부분 곱의 횟수를 $k(k+1)/2$로 줄일 수 있다[9]. 이와 같이 요구되는 부분 곱 계산의 수를 줄이기 위해서는 한 클록에 두 워드를 동시에 읽어낼 수 있는 데이터패스 구조와 정확한 부분곱 스케줄이 필요하다. 이들 저자들은 실험에서 23 비트 조합 곱셈기와 5 세그먼트 멀티 세그먼트 곱셈 스케줄을 사용하여 115 비트 유한체 곱셈기를 사용하는 ECC 스칼라 곱 연산기를 구현하였다[9]. 본 논문에서는 이들의 곱셈기 설계 방법을 확장하여 k가 16 혹은 그 이상까지 증가하여도 효율적인 멀티 세그먼트 곱셈 연산기 구조을 제안하고 이를 이용하여 기존의 디지트 serial 유한체 연산기를 사용하는 ECC 프로세서보다 자원 사용량이 적은 ECC 프로세서를 설계하였다.

Ⅲ. 타원곡선 암호 프로세서 설계

1. 타원곡선 암호 알고리즘

$GF(2^m)$에 기반한 ECC 알고리즘은 다음의 Weierstrass 형식의 타원곡선을 사용한다.

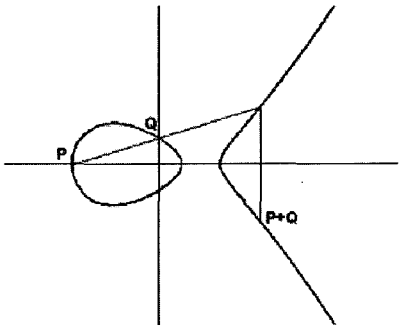


그림 1. 타원곡선 상의 그룹합 연산
Fig. 1. Group addition on the elliptic curve.

표 1. 타원곡선 암호 그룹 연산의 계산
Table 1. Calculating ECC group operations.

$P+Q=(x_0,y_0)+(x_1,y_1)$	$2P=(x_1,y_1)+(x_1,y_1)$
$\lambda=\frac{(y_0+y_1)}{(x_0+x_1)}$	$\lambda=x_1+\frac{y_1}{x_1}$
$x_2=\lambda^2+\lambda+a+x_0+x_1$	$x_2=\lambda^2+\lambda+a$
$y_2=(x_0+x_2)\lambda+x_2+y_1$	$y_2=(x_1+x_2)\lambda+x_2+y_1$

$$y^2+xy=x^3+ax^2+b \quad (1)$$

식 (1)에서 알 수 있는 바와 같이 하나의 타원곡선은 계수 a와 b에 의하여 결정된다. 곡선 상의 한점은 두개의 유한체 값의 쌍으로 나타낸다. 타원곡선 상의 유한개의 점들과 무한점(infinity point) 사이에 기하학적으로 정의된 합 연산에 의하여 그룹을 정의한다. <그림 1>에서 그룹의 합 P+Q가 실수 상에서 정의된 타원곡선 상에서 예시되어 있다. 점 P를 자신에게 더하는 연산은 근접한 서로 다른 두점 가산의 극한치로 정의되어 doubling이라 부르며 2P로 표시된다. 그룹 합과 doubling 연산은 <표 1>에 나타낸 유한체 연산식으로 계산할 수 있다 .

Affine 좌표계 구현에서는 곱셈과 나눗셈을 하드웨어로 구현하여 <표 1>의 식을 이용하여 스칼라 곱을 구한다. 사영 좌표계 구현에서는 유한체 연산의 특성을 이용하여 스칼라곱 연산 iteration 내에서 곱셈만을 사용하고 최종 결과를 나눗셈을 이용하여 affine 좌표계로 변경하는 방법을 사용한다. 사영 좌표계를 이용한 효율적인 스칼라 곱 계산법은 [5]에서 연구되었다. 본 논문에서는 [5]에서 제안된 방법을 사용한다.

2. 멀티 세그먼트 곱셈 알고리즘

다항식 기저(polynomial basis)의 유한체 $GF(2^m)$ 연

산은 미리 정해진 m차 원시 다항식(primitive) $P(x)$를 사용하는 modulo 연산으로 생각할 수 있다. 이때 유한체 $GF(2^m)$의 각 원소는 $m-1$차 이하의 이진 다항식(binary polynomial)으로 표현할 수 있다.

$$a_{m-1}x^{m-1}+\ldots+a_2x^2+a_1x^1+a_0 \in GF(2^m) \quad (2)$$

유한체 $GF(2^m)$ 연산을 하드웨어로 구현하려면 다항식 기저의 원소를 비트 벡터로 표현하는 방법을 사용한다. $GF(2^m)$의 원소는 m-비트 $GF(2)$ 벡터로 표현이 가능한데, 이는 $GF(2^m)$ 원소와 정수는 하드웨어적으로 같은 표현법을 가짐을 보여준다. $GF(2)$에서 덧셈 "+"은 XOR 연산으로, 곱셈 "·"은 AND 연산으로 구현된다. $GF(2^m)$에서 덧셈 "+"은 쉽게 비트별 XOR 연산으로 구현이 가능하나 곱셈은 조금 복잡한 다음의 식으로 정리된다.

$$C=A\cdot B=\sum_{k=0}^{2m-2}c_kx^k \bmod P(x) \quad (3)$$

$$c_k=\sum_{i=0}^{k}a_ib_{k-1} \text{ for } 0\le k\le 2m-2, a_i=0, b_i=0, i\ge m$$

멀티 세그먼트 곱셈 방법에서 두 입력치는 k개의 부분 워드로 나누어져서 다수의 부분 워드 곱셈들이 계산되고 이들은 각각 결과 레지스터(accumulator)에 축적된다. 예를 들어 두 입력치 A, B가 3의 배수 길이를 갖는 원소라고 한다면 그들의 곱 C는 <그림 2>과 같이 나타내진다. <그림 2>에서 기초적인 멀티 세그먼트 곱셈(elementary multi-segment multiplier) 방법은 9번의 곱셈을, KMM(Karatsuba multi-segment multiplier) 방법은 6번의 곱셈을 이용함을 알 수 있다.

$$\begin{aligned}AB&=(A_2x^{2n/3}+A_1x^{n/3}+A_0)(B_2x^{2n/3}+B_1x^{n/3}+B_0)\\&=(A_2B_2)x^{4n/3}\\&\quad\oplus(A_2B_1\oplus A_1B_2)x^n\\&\quad\oplus(A_2B_0\oplus A_1B_1\oplus A_0B_2)x^{2n/3}\\&\quad\oplus(A_1B_0\oplus A_0B_1)x^{n/3}\\&\quad\oplus(A_0B_0)\\&=(A_2B_2)x^{4n/3}\\&\quad\oplus((A_2\oplus A_1)(B_2\oplus B_1)\oplus A_2B_2\oplus A_1B_1)x^n\\&\quad\oplus((A_2\oplus A_1\oplus A_0)(B_2\oplus B_1\oplus B_0)\\&\qquad\oplus(A_2\oplus A_1)(B_2\oplus B_1)\\&\qquad\oplus(A_1\oplus A_0)(B_1\oplus B_0))x^{2n/3}\\&\quad\oplus((A_1\oplus A_0)(B_1\oplus B_0)\oplus A_1B_1\oplus A_0B_0)x^{n/3}\\&\quad\oplus(A_0B_0)\end{aligned}$$

EMM

KMM

그림 2. EMM 곱셈과 KMM 곱셈
Fig. 2. EMM multiplication and KMM multiplication.

유한체의 크기 m을 k개의 세그먼트로 나눈다면($m=k\times w$, w는 서브워드의 크기), 일반적으로 EMM 방법의 곱셈 횟수는 $k\times k$번, KMM 방법은 $k(k+1)/2$번이 필요하다[9]. KMM 방법은 덧셈의 수는 많으나 곱셈 수는 매우 적다. 덧셈 비용이 곱셈 비용보다 현저히 낮기 때문에 KMM 방법이 EMM 방법보다 계산하는 데 더 효율적이다. KMM 하드웨어를 효과적으로 구현하기 위하여 추가적인 하드웨어 자원 없이 $k(k+1)/2$번의 부분곱을 축적하는 효과적인 방법을 고안해내야 한다.

<그림 3>에는 $k(k+1)/2$번의 부분곱을 사용하여 k 세그먼트 멀티 세그먼트 곱, $MSK_k(A,B)$를 구하는 recurrence relation들이 주어져 있다. 그림에서 $\hat{x}$는 $x^{m/k}$을 나타낸다.

참고문헌 [9]에서 저자들은 요구되는 곱셈의 횟수를 k*k에서 k*(k+1)/2로 줄이기 위하여 KMM 곱에 사용되는 세 가지 패턴으로 정의하고 이들을 heuristic한 방법으로 스케줄을 하여 얻은 곱셈 제어기를 이용하여 멀티 세그먼트 곱셈기를 설계하였다. 즉, <그림 3>를 사용하여 얻은 곱셈 항들을 가산 위치를 고려하여 그룹화하여 <그림 4>의 3 가지 패턴을 얻는다.

<그림 4>에서 MSH(most significant half)는 부분곱 결과의 상위부를 나타내며 LSH(least significant half)는 부분곱의 하위부를 나타낸다. 예를 들어 <그림 4>의 (패턴 2)는 하나의 w 비트 곱셈으로 2*w 비트 결과를 얻어서 LSH와 MSH를 더하여 좌우에 MSH와 LSH을 붙여서 3*w비트 결과를 만들어 축적 레지스터(accumulator)에 더하는 것을 나타낸다. EMM 방법은 KMM 곱셈 방법에서 (패턴 1)만을 사용하는 곱셈 계산

$$S_{l,n}=M_{l,n}\oplus R_{l,n},\ n>0,\ l\ge 0$$

$$R_{l,n}=\begin{cases}M_{l,n-1}\oplus M_{l+1,n-1}\oplus R_{l+1,n-2}, & n>1\\ 0, & n=1\end{cases}$$

$$M_{l,n}=\left(\bigoplus_{i=0}^{n-1}A_{l+i}\right)\otimes\left(\bigoplus_{i=0}^{n-1}B_{l+i}\right),\ n>0$$

$$MSK_k(A,B)=\bigoplus_{i=1}^{k}S_{0,i}\hat{x}^{i-1}\oplus\bigoplus_{i=1}^{k-1}S_{i,k-i}\hat{x}^{i-1+k}$$

그림 3. KMM 곱을 계산하는 recurrence relation들
Fig. 3. Recurrence relations for computing KMM products.

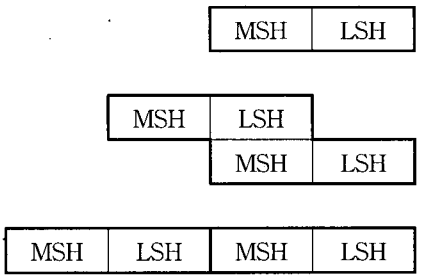


그림 4. 3 가지 서브워드 가산 패턴
Fig. 4. 3 accumulation patterns for subword products.

표 2. KMM 곱셈 패턴
Table 2. KMM multiplication patterns.

순서	입력	패턴 형식	가산 위치
1	A_2B_2	패턴 2	m
2	$(A_2+A_1)(B_2+B_1)$	패턴 2	$(2/3)m$
3	$(A_2+A_1+A_0)(B_2+B_1+B_0)$	패턴 1	$(2/3)m$
4	$(A_1+A_0)(B_1+B_0)$	패턴 2	$(1/3)m$
5	A_1B_1	패턴 3	$(1/3)m$
6	A_0B_0	패턴 2	0

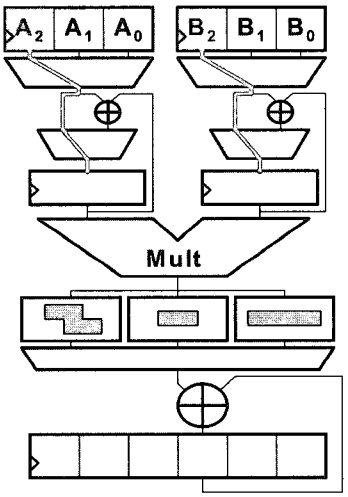


그림 5. Ernst의 멀티 세그먼트 데이터패스
Fig. 5. Ernst's multi-segment datapath.

방법이다.

KMM 곱셈 항들을 다시 그룹화 하여 3 가지 패턴을 구하는 것은 쉽다. <표 2>에 k가 3일 경우의 모든 곱셈 패턴이 나타나 있다. <표 2>의 패턴을 순서대로 축적 레지스터에 더하면서 가산 위치가 변경될 때 (1/3)m 비트 단위로 쉬프트하면 <그림 2>의 KMM 곱셈을 얻을 수 있음을 알 수 있다.

참고문헌 [9]에서는 <그림 5>의 데이터패스를 이용하여 멀티 세그먼트 곱셈을 수행하였다. <그림 5>에는 하나의 출력 레지스터와 두개의 입력 축적 레지스터가 있다. <그림 5>의 데이터패스 상에서 <표 2>의 곱셈 패턴들을 이용하여 곱셈을 수행할 때 곱셈 결과를 축력 레지스터에 축적하고 입력 쌍은 <그림 5>의 입력 축적 레지스터에 축적되어야 한다. 또한 <표 2>의 (패턴 i)에서 (패턴 i+1)로 움직일 때 오직 두 워드만 새로 입력하면 된다. <그림 5>의 출력 레지스터는 on-the-fly reduction 기법을 사용하면 그 크기를 2*k에서 k로 줄일 수 있다.

3. 패턴 이동 그래프를 이용한 멀티 세그먼트 곱셈기 스케줄

<그림 5>의 데이터패스와 참고문헌 [9]에서 Ernst, Jung, Madlener 등이 사용한 곱셈 스케줄 방법을 이용하여 KMM 곱셈기를 설계하면 <그림 6>에서 보인 바와 같이 k가 6 이상인 경우에는 곱셈에 필요한 클록 사이클의 수가 현격히 증가한다. 그 이유는 스케줄에 따른 곱셈 데이터 입력을 위해서는 4 워드를 입력하여야 하므로 한 클록에 2 워드 입력이 가능한 <그림 5>의 데이터패스를 사용하면 데이터 입력에 두 클록이 소요되며 이 두 클록 사이클 동안 하나의 부분 곱 워드만 출력 레지스터에 축적된다. 서론에서 언급한 바와 같이 Ernst, Jung, Madlener 등은 k가 5인 멀티 세그먼트 곱셈기만 실험에서 사용하였기 때문에 곱셈을 완료하는 동안 이러한 파이프라인 동결 현상은 한 클록 동안만 발생한다[9]. 이에 대한 간단한 해결책으로는 <그림 5>의 데이터패스 상에 2 개의 서브워드 멀티플렉서를 더 두는 것이다. 하지만 이 경우 하드웨어 비용이 크게 증가하게 된다.

더 효율적인 곱셈기를 설계하기 위하여 <그림 5>의

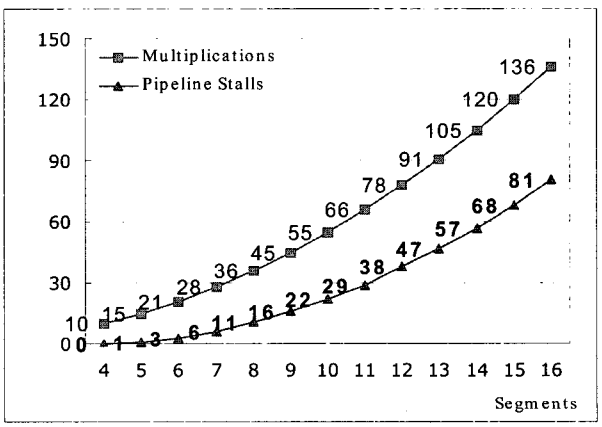
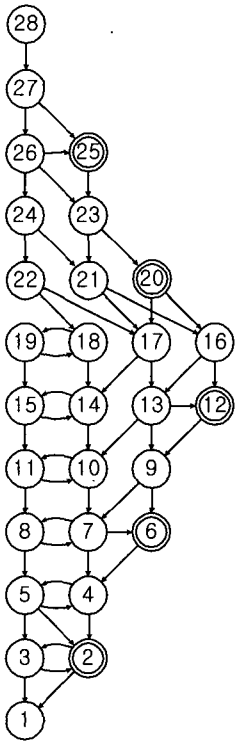


그림 6. 파이프라인 정지 횟수
Fig. 6. Number of pipeline stalls.

Pattern No.	$(\oplus Ai)\cdot(\oplus Bi)$	Pattern Type
1	0	2
2	1	3
3	01	2
4	12	3
5	012	2
6	2	3
7	123	3
8	0123	2
9	23	3
10	1234	3
11	01234	2
12	3	3
13	234	3
14	12345	3
15	012345	2
16	34	3
17	2345	3
18	123456	2
19	0123456	1
20	4	3
21	345	3
22	23456	2
23	45	3
24	3456	2
25	5	3
26	456	2
27	56	2
28	6	2

(a)



(b)

그림 7. 7-세그먼트 KMM 패턴과 패턴 이동 그래프
Fig. 7. 7-segment KMM pattern and its movement graph.

데이터패스 상에서 가능한 서브워드 곱 스케줄들을 분석하였다. 효율적인 스케줄 연구를 위하여 서브워드 곱 스케줄을 방향성 그래프(digraph)로 모델링 하였다. 각 패턴은 노드로, 패턴 간 가능한 이동은 에지로 표현하였다. 스케줄 그래프는 <그림 7>의 (b)에서 보인 바와 같이 하나의 source와 하나의 sink를 가지며 곱셈 수행에 필요한 최소 쉬프트 수의 레벨(level)을 가진다. <그림 7(a)>는 k가 7일 경우 KMM 곱셈 패턴을 차수와 부분합의 크기별로 정렬한 것이다. 이 그래프에서 각 노드들은 <그림 7(a)>에서 부여한 고유 번호를 나타낸 패턴들에 해당한다. <그림 7(b)>에서 이중원 노드 2, 6, 12, 20, 25는 EMM의 경우처럼 직접 입력이 가능한 노드로서 현재 레벨과 그전 레벨의 모든 노드로부터 이동이 가능하다. <그림 7(b)>에서 완전한 곱셈 스케줄은 시작 노드 28부터 마지막 노드 1까지 모든 노드를 커버

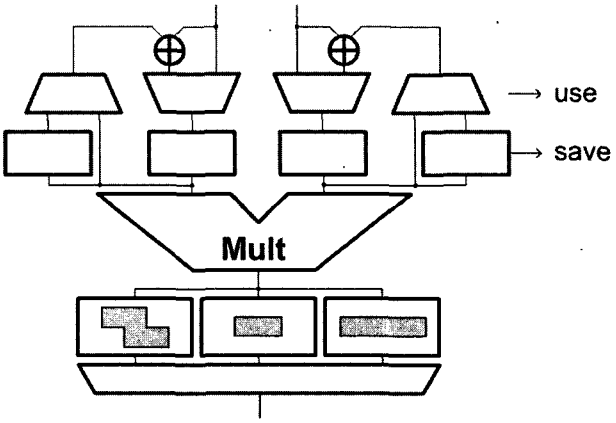


그림 8. SAVE 스케줄을 위한 데이터패스 변경
Fig. 8. Datapath modification for SAVE scheduling.

하는 경로로 나타낼 수 있다. <그림 7(b)>의 경우 이러한 경로는 존재하지 않는다. <그림 6>에서 알 수 있는 바와 같이 에지를 따르지 않는 6개의 노드 간 이동을 사용하면 곱셈이 필요한 스케줄을 얻을 수 있다.

<그림 7(b)>의 그래프의 분석해 보면 그래프의 에지만을 사용하는 완전한 스케줄을 얻기 어려운 이유는 같은 레벨에서 패턴 간의 이동이 어려운 점임을 알 수 있다. 이 그래프를 자세히 관찰하면 많은 노드들이 그전 레벨의 노드들로부터 두개의 에지가 있음을 알 수 있다. 본 논문에서는 이를 이용하며 현재 레벨에서 얻은 입력 축적 패턴을 몇 개의 서브워드 레지스터에 저장하여 두었다가 다음 레벨에서 사용할 수 있도록 하는 간단하지만 효율적인 스케줄을 고안하였다. 본 논문에서는 이 스케줄을 SAVE 스케줄(save schedule)이라 한다.

SAVE 스케줄 알고리즘을 <그림 7(b)>에 적용하면 하나의 입력 축적 저장 레지스터를 이용하여 스케줄 28−27−26−25(S)−23−24(U)−21(S)−20−22(US)−18−19−16(U)−17(US)−14−15−12−13(US)−10−11−9(U)−7−8−6−4−5−2−3−1을 얻을 수 있다. 이 스케줄에서 괄호 속의 S는 직전 입력 축적을 저장(save)하는 것을 나타내며, U는 직전 입력 축적을 사용하는 대신 저장된 입력 축적을 사용(use)하는 것을 나타낸다. 저장과 사용을 동시에 할 수 있는 이유는 플립플롭의 특성에 따른 것이다.

SAVE 스케줄을 위하여 저장 레지스터를 추가하여 <그림 5>의 데이터패스를 변경하는 방법이 <그림 8>에 나타나 있다. 필요한 저장 서브레지스터 수는 k가 증가할수록 증가한다. 그러나 k가 8일 때까지는 한 쌍

의 서브레지스터만 사용하는 스케줄이 있으며 k가 16일 때까지는 두 쌍의 서브레지스터만 사용하는 스케줄이 있음을 확인하였다.

4. ECC 데이터패스 설계

본 논문에서는 Ernst, Jung, Madlener 등과 같이 곱셈기에 덧셈, 제곱 연산 기능을 추가하여 사영 좌표계를 사용하는 타원곡선 암호 프로세서를 설계하였다[7].

가. 멀티 세그먼트 곱셈을 사용한 ECC 데이터패스

<그림 9>에는 타원곡선 암호 프로세서를 위한 개념적으로 간단한 연산 회로가 나타나 있다. ECC 프로세서는 이 연산회로에 RAM 블록과 제어 블록들을 추가하여 구성된다. <그림 9>에서 a, b 레지스터는 전체 유한체 크기의 레지스터이고 MULT는 멀티 세그먼트 곱셈기를 나타낸다. Ernst, Jung, Madlener 등은 저비용 ECC 프로세서를 구현하기 위하여 이와 유사한 마이크로 아키텍처를 사용하였다[9]. <그림 9>을 살펴보면 전통적인 디지트 serial 곱셈기가 MULT 블록과 입력 선택 멀티플렉서로 표현되었음을 알 수 있다.

FPGA 자원 사용 비교 실험 결과 <그림 9>의 방법으로 멀티 세그먼트 곱셈기를 사용하는 것은 전통적인 디지트 serial 곱셈기와 비교하여 자원 사용면에서 장점이 없음을 알게 되었다. 본 논문에서는 이 마이크로 아키텍처를 변경하여 자원 사용을 극소화하는 마이크로 아키텍처를 설계하였다. 이를 위하여 먼저 RAM 블록을 서브워드 단위로 직접 접근하면 <그림 9>에서 a, b 레지스터와 곱셈기 입력 선택 멀티플렉서는 생략될 수 있다. 이 경우 유한체 곱셈은 서브워드 단위로 수행되며 최종 결과는 곱셈기 내부의 p 레지스터에 나타나며 멀티 클록을 사용하여 RAM 블록 내의 목적 주소들로 전송된다.

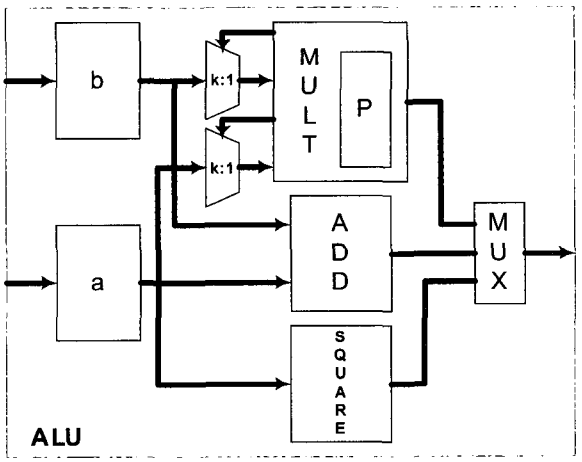


그림 9. 멀티 세그먼트 곱셈을 이용한 유한체 연산 데이터 패스

Fig. 9. The finite field arithmetic datapath using multi-segment multiplier.

유한체 덧셈 또한 멀티 클록에 수행하여야 한다. 또한 저비용 타원곡선 암호 연산기의 경우에 제곱 연산기는 곱셈 연산으로 대체할 수 있다. 마지막으로 이러한 멀티 세그먼트 곱셈을 이용한 설계는 보조 프로세서 내의 전체 유한체 크기 버스의 수도 감소시켜 구현 면적을 줄이는 데 도움을 준다.

나. 최적화된 멀티 세그먼트 데이터패스

<그림 10>에 본 논문에서 제안하는 유한체 연산기의 데이터패스 구조를 나타내었다. <그림 10>의 두꺼운 선은 크기가 m인 버스를, 가는 선은 크기가 $w(=m/k)$인 버스를 나타낸다. 여기서 MEMD 블록은 <그림 5>에서 보인 Ernst의 데이터패스에서 축적 레지스터 p와 관련 논리회로를 제외하고 <그림 8>에서처럼 SAVE 레지스터들을 추가한 멀티 세그먼트 Karatsuba 곱셈 방법을 이용한 데이터패스 구현 블록이다.

두개의 곱셈 데이터 입력치를 각 클록 사이클마다 읽어 오기 위해서 듀얼 포트 RAM 메모리를 사용하였다. 여기서 듀얼 포트 RAM은 두개의 주소, 즉 하나의 읽고 쓰는 연산에 사용되는 주소와 다른 하나의 읽기만을 하

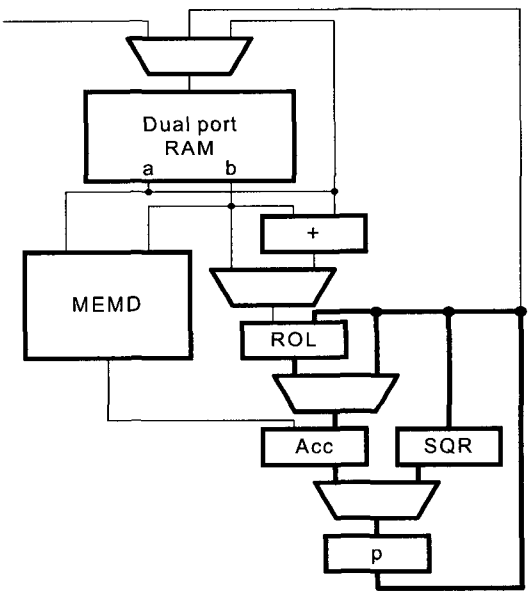


그림 10. 최적화된 멀티 세그먼트 데이터패스

Fig. 10. The optimized multi-segment datapath.

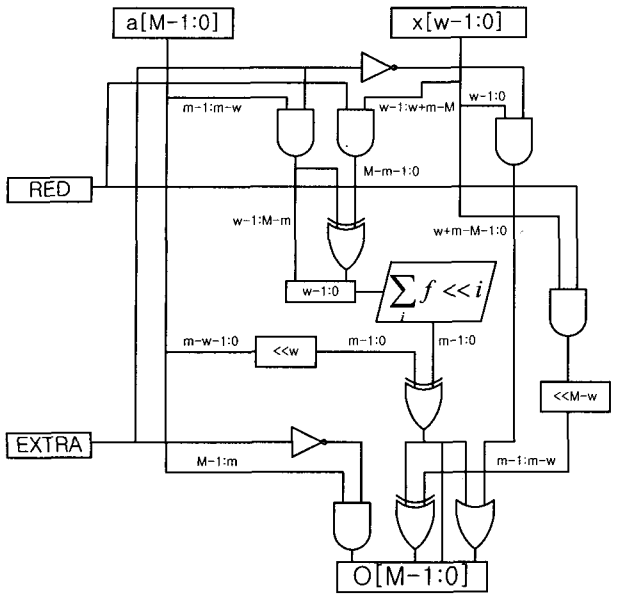


그림 11. ROL 블록 논리 회로도
Fig. 11. ROL block circuit diagram.

는 주소를 가진다고 가정한다. 이 듀얼 포드 RAM으로는 3-주소 덧셈은 불가능하다. 덧셈을 하기 위하여 입력치 하나는 덧셈 연산 전 k 클록에 레지스터 p에 옮겨져 있어야 한다. 덧셈 연산 동안에 레지스터 p의 값은 k번 좌로 쉬프트 되어 두 번째 소스 입력치와 비트별 XOR 되고 목적 주소에 쓰여 진다. 이러한 유한체 덧셈 연산에는 $2k$ 클록이 걸린다. 만일에 3-주소 RAM 블록을 사용할 수 있다면 k 클록에 완성되는 유한체 덧셈 연산을 구현할 수 있을 것이다.

제곱 연산에 사용된 자원을 절약하기 위하여 <그림 11>의 ROL 블록을 이용하여 제곱 연산을 구현할 수 있다. <그림 11>에서 M은 하드웨어 레지스터 크기, m은 유한체 크기, w는 워드 크기를 나타낸다. ROL 블록은 유한체 연산 쉬프트, 논리 쉬프트, 그리고 제곱 연산에 사용되는 추가적인 워드 합 기능을 포함하고 있다. <그림 11>에서 AND 및 XOR 게이트는 멀티 비트 구조이며 입력이 RED나 EXTRA와 같이 1 비트 제어 입력인 경우 다른 입력 비트 수만큼 중복된다. ROL 블록을 사용하여 제곱 연산을 하는 경우 <그림 10>의 SQR 블록과 관련된 멀티플렉서는 불필요하다.

ROL 블록을 사용하여 제곱 연산을 수행하기 위하여 우선 소스 입력치를 읽어 와서 제곱하여 두개의 서브워드로 나눈다. 그 중 상위 워드는 레지스터 p에 로드하고 하위 워드는 목적 주소에 쓴다. 레지스터 p는 k번 좌로 쉬프트 되어 목적 주소에 더해진다. 바로 이 쉬프트에 의해서 하위 워드에 더해지기 전에 상위 워드의 값은 하위 워드와 같은 비중(weight)을 가지게 된다. 이러한 방식으로 제곱을 구현하면 제곱 연산에 적어도 $4k$ 클록이 소요된다. Karatsuba 곱셈 방법으로 곱셈 연산을 수행하는 데 $k(k+1)/2$ 클록이 걸리므로 이러한 제곱 방법으로는 k가 8보다 큰 경우에만 유용하다.

다. 데이터패스 자원 사용량 비교

멀티 세그먼트 데이터패스의 자원 사용량은 간단한 수식으로 표현하기 어렵다. <표 3>에서는 실험에서 사용된 8 비트, 16 비트, 그리고 32 비트 조합 곱셈기의 자원 사용량을 보였다[9]. ROL 블록의 자원 사용량을 세그먼트 크기 w, 하드웨어 레지스터 크기 M, 유한체 크기 m의 식으로 표현하였다. 여기서 CKM 블록은 <그림 5>의 Ernst 멀티 세그먼트 곱셈기의 순수한 조합회로 곱셈기를 의미하며 <그림 5>와 <그림 6>에서 MULT 블록으로 표현되어 있다.

<표 4>에서는 디지트 시리얼 데이터패스와 제안된 멀티 세그먼트 데이터패스의 자원 사용량을 비교하였다. <표 4>에서 p와 s는 유한체를 정의하는 원시 다항식의 항의 수와 SAVE 스케줄을 위한 추가적인 워드 크기의 레지스터 수를 나타내며, a(w)와 x(w)는 각각 CKM 조합 회로 곱셈기 구현에 요구되는 AND2와 XOR2 게이트의 수를 나타낸다.

<표 4>에서 보는 바와 같이 멀티 세그먼트 곱셈기의 자원 사용량은 세그먼트 수가 크면 w가 작아지므로

표 3. CKM 블록과 ROL 블록의 자원 사용량
Table 3. Resource usage of CKM block and ROL block.

	CKM8	CKM16	CKM32	ROL
AND2	27	81	243	3w+M-m
XOR2	100	360	1207	M+w(1+p)
OR2	0	0	0	w

표 4. 디지트 시리얼 데이터와 멀티 세그먼트 데이터 패스 자원 비교
Table 4. Comparison of resource usage of digit-serial datapath and multi-segment datapath.

	Digit Serial Datapath	Multi-segment Datapath
AND2	mD	6w+3w+M-m+a(w)
XOR2	mD+(2D-1)p+D(D-1)/2	7w+M+w(1+k)+x(w)
MUX2	7m	8w+m
OR2	0	w
DFF	2m	2sw+m

CKM 조합 회로의 사용 게이트 수는 디지트 시리얼 곱셈기의 사용 게이트 수보다 작아진다. 그러나 디지트 시리얼 곱셈기는 구조가 매우 간단하므로 배선이 매우 효율적이다. 따라서 실질적인 자원 사용량은 회로 합성 후 배선 효율성을 고려하여야 한다.

Ⅳ. 실 험 결 과

멀티플렉서와 버스를 구현하는 논리 요소와 라우팅이 VLSI 자원의 많은 부분을 차지하는 유한체 $GF(2^m)$의 마이크로 아키텍쳐의 경우에는 VLSI 설계 완료 이전에 마이크로 아키텍쳐의 자원 요구량을 정확하게 예측하는 것은 어려운 일이다. 본 논문에서는 FPGA 구현을 이용하여 필요한 자원 요구량과 임계 경로 지연을 실험하였다. ASIC 설계와 FPGA 구현의 자원 사용 경향이 다소 다르나 FPGA 구현에 적합한 마이크로 아키텍쳐가 ASIC 구현에도 효율적일 것으로 기대된다. 먼저 <그림 9>의 ALU 설계를 사용하여 몇 가지 멀티 세그먼트 곱셈기를 구현하였다. 정확한 자원 요구량을 알기 위하여 설계된 ALU들을 이용하여 전체 ECC 프로세서를 구현하였다. 구현된 타원곡선 암호 프로세서는 Altera사의 EXCALIBUR EPXA10F1020C2 FPGA 상에서 검증하였다. Rosing의 ECC 소프트웨어 C 프로그램을 이 FPGA의 칩 스트라이프(stripe)에 존재하는 ARM 프로세서 상에서 수행하였다[3]. 스칼라 곱 연산 소프트웨어 함수를 AMBA 버스를 통한 ECC 보조 프로세서 상의 하드웨어 알고리즘 호출로 대체하였다. ARM 프로세서는 호스트 PC와 serial 통신으로 연결되어 있으므로 수행 과정 및 결과는 하이퍼 터미널 창을 통하여 확인할 수 있다.

<표 5>에서는 <그림 8>의 ALU 구조를 사용한 멀티 세그먼트 Karatsuba 곱셈기와 디지트 크기가 2와 4인 디지트 시리얼 곱셈기로 구현한 유한체 연산기들의 자원 요구량을 비교하였다. 이 실험의 유한체의 크기는 128 비트였다. 세그먼트의 수 k=8인 KMM 곱셈기는 한 번의 곱셈을 37 클록에 완료할 수 있다. 디지트 D=2, 4인 LSD 곱셈기는 곱셈 완료에 각각 64 클록, 32 클록이 소요된다. 두 번째 난에는 논리 요소 사용량을 4번째 난에는 스칼라 곱 연산에 필요한 클록수를 나타내었다.

표 5. 예비 구현 곱셈기와 LSD 곱셈기와의 비교
Table 5. Comparing preliminary implementations with LSD multipliers.

Multiplier	# of Logic Elements	Total Memory Bits	# of clks	Max Clk Freq
KMM(k=8)	2,804	2,048	94,016	57.22
LSD(D=2)	2,478	2,048	72,086	62.08
LSD(D=4)	2,694	2,048	43,372	59.34

세그먼트 수와 워드 곱셈기의 크기를 변경하며 많은 실험을 수행하였으나 <표 5>에서 본 것처럼 멀티 세그먼트 곱셈기를 사용한 연산 구조가 더 많은 자원을 사용하였다. <그림 9>의 데이터패스 구조에서 a, b 레지스터 전단의 입력 선택 멀티플렉서의 길이는 유한체 크기와 같은 m이다. 이들 전체 길이 입력 선택 멀티플렉서는 멀티 세그먼트 곱셈 방법의 최대 이점을 살리지 못하게 한다. 또한 <표 5>에서 주의할 점은 디지트 serial 곱셈기의 디지트 크기가 2에서 4로 증가하여도 필요한 곱셈기의 논리 요소의 수는 그다지 증가하지 않는다. 이는 $GF(2^m)$ 연산의 경우 연산보다는 데이터 이동에 많은 자원이 사용됨을 나타낸다. 이는 Orlando와 Paar의 실험 결과와도 일치한다[7]. 따라서 디지트 serial 곱셈기 구조와 비교할 경우 멀티 세그먼트 곱셈기의 서브 워드 구조의 장점이 자원 효율성으로 나타나지 않음을 알 수 있다.

다음에는 본 논문에서 제안된 <그림 10>의 유한체 연산 데이터패스를 사용하여 타원곡선 암호 프로세서를 설계하였다. 이 실험에서는 Xilinx Vertex-II FPGA

표 6. 최종 구현 곱셈기와 LSD 곱셈기와의 비교
Table 6. Comparing final implementations with LSD multipliers.

	# clks	period(ns)	# slices	# BRAM's
F128D1	129,622	6.89	972	2
F128D2	72,086	5.35	1,052	4
F128D4	43,366	11.2	1,155	4
F233D1	407,162	9.9	1,601	4
F233D2	217,598	7.18	2,188	4
F233D4	122,826	7.16	2347	4
LEU113*	166,783	22.2	1290	N.A.
LEU155*	246,443	27.8	1567	N.A.
LEU281*	474,504	30.3	2622	N.A.
C128M8	229,121	5.6	622	1
C128M16	94,061	5.59	694	1
C128M32	47,172	5.2	1,058	1
C233M16	381,535	5.70	933	1
C233M32	173,210	5.36	1,376	1

XC2V6000을 사용하였다. Vertex-II를 선택한 이유는 이것이 실제적인 듀얼 포트 메모리를 갖추고 있기 때문이다. Altera FPGA에서 제공하는 듀얼 포트 메모리는 싱글 포트 메모리 두개로써 구성되어 있다.

많은 ECC 구현이 서로 다른 크기의 유한체와 다른 종류의 FPGA를 사용하고 사용된 CAD 소프트웨어도 각각 다르므로 이들 논문에서 제시된 ECC 프로세서들의 성능과 자원 사용량을 비교하는 것은 쉽지 않다. 본 논문에서는 제안된 멀티 세그먼트 곱셈기를 사용하는 ECC 프로세서와 디지트 시리얼 곱셈기를 사용하는 ECC 프로세서를 직접 구현하여 성능과 자원 소요량을 비교하였다.

<표 6>에서 F128D1은 128 비트 시리얼 곱셈기를, F128D2는 디지트가 2인 디지트 시리얼 곱셈기를 사용한 ECC 프로세서 구현 결과이다. F233D1과 F233D2는 233 비트 시리얼 곱셈기와 디지트 시리얼 곱셈기를 사용한 구현 결과이다. 128 비트 구현에서는 원시 다항식 $x^{128}+x^7+x^2+x+1$을 사용하였으며 233 비트 구현에서는 원시 다항식 $x^{233}+x^{74}+1$을 사용하였다. 사용된 타원곡선의 파라메터 a는 0으로 고정하고 b는 난수 함수에서 얻었다. ECC 구현의 자원 사용량은 타원곡선 파라메터 a와 b에는 무관하다. LEU113, LEU155, 그리고 LEU281은 각각 Leung, Ma, Wong 등이 참고문헌 [12]에서 보고한 113, 155, 그리고 155 비트 ECC 프로세서들이다. 이 실험에서 사용된 유한체의 크기와 FPGA의 속도 차이를 고려할 때 본 논문에서 구현된 디지트 시리얼 ECC 프로세서 구현은 이들의 ECC 프로세서 구현과 비교하여 속도와 자원 소요량 면에서 유사함을 알 수 있다.

C128M8, C128M16, C128M32는 각각 8, 16, 32 비트 조합논리 Karatsuba 곱셈기를 이용한 멀티 세그먼트 곱셈기를 이용하여 ECC 프로세서를 구현된 것이다. C233M16과 C233M32의 구현은 유한체의 크기가 서브 워드 크기의 배수가 되지 않을 때에도 멀티 세그먼트 곱셈기를 어려움 없이 잘 구현할 수 있음을 보여준다. 이 경우 유한체 값들은 LSB 쪽으로 정렬하였다. 특히 C233M16 구현은 k가 홀수일 때도 제곱 연산이 효과적으로 구현이 됨을 추가적으로 보여준다. 이 경우 제곱 연산 과정에서 가운데 서브 워드의 상위 부는 p 레지스터의 LSW에 하위부는 목적 주소의 MSW에 저장된다. 타원곡선 스칼라곱 연산을 완료하는 데 걸리는 클록 수는 두 번째 열에 나타나 있다. 클록 주기는 Xilinx ISE에서 제공한 임계 경로의 지연 시간(critical path delays)이다. 네 번째 열에서는 자원 사용량을 보여준다. <표 6>의 F128D1과는 F128D2는 <표 5>의 LSD 곱셈기들를 이용한 설계를 Xilinx Vertex-II에서 합성한 결과이다.

F233D2 설계와 C233M32 설계 비교에서 알 수 있는 바와 같이 본 논문에서 제안된 멀티 세그먼트 곱셈기로 구현된 ECC 프로세서가 기존의 디지트 시리얼 곱셈기로 구현된 것보다 더 빠르고 면적은 적게 차지하는 경우가 많을 것임을 알 수 있다. 또한 F128D1 설계와 C128M16 설계 비교에서처럼 세그먼트의 수 k를 증가시킴으로써 (또는 곱셈기의 서브워드 크기를 감소시킴으로써) 비트 serial 곱셈기를 사용하는 경우보다 자원 사용량이 적으면서 성능이 우수한 ECC 프로세서를 설계할 수 있음을 알 수 있다.

V. 결론 및 차후 연구 방향

본 논문에서는 타원곡선 암호 응용을 위한 효율적인 멀티 세그먼트 $GF(2^m)$ 곱셈과 그 응용에 대하여 연구하였다. 연구 결과, 세그먼트 수 k가 커지더라도 참고문헌 [9]에서 Ernst, Jung, Madlener 등이 사용한 데이터 패스에서 최소한의 자원을 추가하면 효율적인 곱셈 스케줄을 얻을 수 있음을 보였다.

제안된 곱셈기를 사용한 타원곡선 암호 프로세서는 Xilinx FPGA 개발 환경에서 구현되었다. 본 논문에서 제안하는 멀티 세그먼트 곱셈기를 사용한 ECC 프로세서와 디지트 시리얼 곱셈기를 사용한 프로세서의 면적 및 임계 경로 지연을 측정하여 비교하였다. 실험 결과 멀티 세그먼트 곱셈기로 만든 ECC 프로세서가 디지트 시리얼 곱셈기의 프로세서보다 면적은 현저히 작고 속도는 빨랐다.

특히 듀얼 포트 RAM 구조가 효과적으로 제공될 때 제안한 멀티 세그먼트 곱셈 알고리즘은 아주 효율적이다. 듀얼 포트 RAM이 제공되지 않는다면 두개의 SRAM을 사용하고 본 논문에서 사용한 Lopez와 Dahab의 곱셈 알고리즘을 레지스터 할당 기법을 이용하여 변경하여야 한다[5]. 이 경우 다소의 성능 저하가 있을 것으로 보인다. 또한 전체 SoC 시스템에서 다른 설계 부분과 RAM 영역의 시간적 공유가 가능하다면 이 RAM

의 비용은 완전히 제거될 수 있다. 이 경우 ECC 수행 속도는 다소 저하된다. 추후 중요한 연구 과제로는 전력 사용면에서 본 논문에서 제안된 멀티 세그먼트 기반 타원 곡선 프로세서와 기존의 디지트 serial 기반 타원 곡선 프로세서를 비교 연구하는 것을 들 수 있다.

참 고 문 헌

[1] D. Hankerson, J. L. Hernandez, and A. Menezes, "Software implementation of elliptic curve cryptography over binary fields," *Cryptographic Hardware and Embedded Systems(CHES 2000), LNCS* 1965, Springer, pp. 2-24, Worcester, MA, USA, August 2000.

[2] M. Brown, D. Hankerson, J. López, and A. Menezes, "Software implementation of the NIST elliptic curves over prime fields," *CT-RSA 2001, LNCS* 2020, Springer, pp. 250-265, 2001.

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

[4] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, "An implementation of elliptic curve cryptosystems over $F_{2^{155}}$," *IEEE Journal on Selected Areas in Communications*, Vol. 11, no. 5, pp. 804-813, June 1993.

[5] J. López and R. Dahab, "Fast multiplication on elliptic curves over GF(2^m) without precomputation," *Cryptographic Hardware and Embedded Systems(CHES '99), LNCS* 1717, Springer, pp. 316-327, Worcester, MA, USA, August 1999.

[6] L. Song and K. K. Parhi, "Low-energy digit-serial/parallel finite field multipliers," *Journal of VLSI Signal Processing Systems*, Vol. 2, no. 22, pp. 1-17, August 1997.

[7] G. Orlando and C. Paar, "A high-performance reconfigurable elliptic curve processor for GF(2^m)," *Cryptographic Hardware and Embedded Systems(CHES 2000), LNCS* 1965, Springer, pp. 41-56, Worcester, MA, USA, August 2000.

[8] E. Savas, A. F. Tenca, and C. K. Koc, "A scalable and unified multiplier architecture for finite fields GF(p) and GF(2^m)," *Cryptographic Hardware and Embedded Systems(CHES 2000), LNCS* 1965, Springer, pp. 277-292, Worcester, MA, USA, August 2000.

[9] M. Ernst, M, Jung, F. Madlener, S. Huss, and R. Bluemel, "A reconfigurable system on chip implementation for elliptic curve cryptography over GF(2^m)," *Cryptographic Hardware and Embedded Systems(CHES 2002), LNCS* 2523, Springer, pp. 382-399, Worcester, MA, USA, August 2002.

[10] H. Wu, "Low complexity bit parallel finite field arithmetic using polynomial basis," *Cryptographic Hardware and Embedded Systems(CHES '99), LNCS* 1717, Springer, pp. 280-291, Worcester, MA, U.S.A, August 1999.

[11] Quartus-II S/W On Line Manual, Altera Corp, http://www.altera.com/product/software/pld/q2/qts-index.html

[12] K. H. Leung, K. W. Ma, W. K. Wong, and P. H. W. Leong, "FPGA implementation of a microcoded elliptic curve cryptography processor," 2000 IEEE Symposium on Field Programmable Custom Computing Machines, pp. 68-76, Napa Valley, CA, U.S.A, April 17-19, 2000.

저 자 소 개

이 동 호(평생회원)
1979년 서울대학교 전자공학과 학사 졸업.
1981년 한국과학기술원 전산학과 석사 졸업.
1992년 (미) Iowa대 컴퓨터과학과 박사 졸업.
1981년~1992년 ETRI 선임연구원
1992년~1993년 (미) Motorolar senior CAD engineer
1993년~현재 경북대학교 전자전기컴퓨터학부 부교수
<주관심분야 : 컴퓨터 구조, VLSI 설계, 디스플레이 하드웨어>