타원곡선 암호를 위한 시스톨릭 Radix-4 유한체 곱셈기의 설계

김주영, 박태근
가톨릭대학교 정보통신전자공학부
e-mail : mog016@catholic.ac.kr, parktg@catholic.ac.kr

Design of a systolic radix-4 finite-field multiplier for the elliptic curve cryptosystem

Ju-Young Kim, Tae-Geun Park
School of Information, Communication, and Electronic Engineering
The Catholic University of Korea

Abstract

The finite-field multiplication can be applied to the wide range of applications, such as signal processing on communication, cryptography, etc. However, an efficient algorithm and the hardware design are required since the finite-field multiplication takes much time to compute. In this paper, we propose a radix-4 systolic multiplier on $GF(2^m)$ with comparative area and performance. The algorithm of the proposed standard-basis multiplier is mathematically developed to map on low-cost systolic cell, so that the proposed systolic architecture is suitable for VLSI design. Compared to the bit-serial and digit-serial multipliers, the proposed multiplier shows relatively better performance with low cost. We design and synthesis $GF(2^{193})$ finite-field multiplier using Hynix 0.35㎛ standard cell library and the maximum clock frequency is 400㎒.

I. 서론

유한체(finite-field)는 보통 Galois Field(GF)라고 하는데, Reed-solomon 디코더 및 타원곡선 암호 등의 암호시스템에서 핵심적인 역할을 수행한다[1][2]. 유한체 연산은 많은 응용에서 실시간 처리가 요구되는 반면 연산과정이 복잡하고 면적과 전력 소비의 제한이 따르는 경우가 많기 때문에 효율적인 알고리즘 및 이에 대한 하드웨어 설계가 필요하다.

유한체 $GF(2)$의 확장체인 $GF(2^m)$은 VLSI 설계 시 효율적인 구조가 가능하며 고속으로 동작하는 시스템을 설계하는데 매우 효과적이다. $GF(2^m)$에서의 덧셈은 매우 간단하지만 곱셈은 복잡하기 때문에 많은 처리 시간과 자원을 필요로 한다. 따라서 $GF(2^m)$상의 유한체 곱셈기를 구현하기 위한 다양한 알고리즘이 연구되고 있다.

본 연구에서는 매우 큰 소수 m을 사용하는 타원곡선 암호 시스템에 적용 가능한 효율적인 유한체 곱셈기를 설계하기 위한 VLSI 구조를 제안한다. 효율적인 면적과 연산시간을 얻기 위하여 m비트의 승수를 수학적 전개를 통하여 2비트씩 묶어서 동시에 연산함으로써 기존의 Radix-2 직렬 곱셈기의 연산블록과 레이턴시를 절반으로 줄이면서도 처리량은 약 2배가 되었으며 기존의 2 digit-serial 곱셈기와 비교했을 때 유사한 성능을 내면서도 Latency는 약 2/3로 감소되었고 하드웨어 복잡도는 감소되었다. 제안된 곱셈기는 표준기저 방식과 시스톨릭 구조를 이용하였다.

Ⅱ. 제안된 Radix-4 시스톨릭 알고리즘

1. $GF(2^m)$ 유한체 곱셈

$GF(2^m)$ 유한체 상의 곱셈을 위하여 승수와 피승수를

각각 $A(x)$와 $B(x)$라하고 기약 다항식을 $F(x)$라고 할 경우, 곱셈의 결과 $P(x)=A(x)B(x)modF(x)$는 아래와 같이 표현할 수 있다.

$$A(x)=a_{m-1}x^{m-1}+\cdots+a_1x+a_0$$
$$B(x)=b_{m-1}x^{m-1}+\cdots+b_1x+b_0$$
$$F(x)=x^m+b_{m-1}x^{m-1}+\cdots+f_1x+1$$
$$P(x)=p_{m-1}x^{m-1}+\cdots+p_1x+p_0$$

본 연구에서는 효율적인 유한체 곱셈기를 구현하기 위하여 아래의 Radix-4 시스톨릭 알고리즘을 제안한다.

우선 $B(x)$의 값들을 2비트씩 묶어서 수식을 정리하면 식 (1)과 같다.

$$\begin{aligned}P(x)&=A(x)B(x)modF(x)\\&=A(x)\{b_{m-1}x^{m-1}+\cdots+b_1x+b_0\}modF(x)\\&=A(x)\{b_{m-1}x^{m-1}+b_{m-2}x^{m-2}\}modF(x)+\cdots+\\&\quad A(x)\{b_2x^2+b_1x\}modF(x)+A(x)b_0modF(x)\end{aligned}\tag{1}$$

식 (1)의 각각의 항을 $K_i\,(i=1,2,\cdots,m)$으로 다시 정리하면 아래와 같다.

$$\begin{aligned}K_1&=A(x)\{b_{m-1}x^{m-1}+b_{m-2}x^{m-2}\}modF(x)\\&=[A(x)\{b_{m-1}x+b_{m-2}\}modF(x)]x^{m-2}modF(x)\\&=P_1x^{m-2}modF(x)\\&\vdots\\K_i&=K_{i-2}+A(x)\{b_{m-i}x^{m-i}+b_{m-i-1}x^{m-i-1}\}modF(x)\\&=[P_{i-2}x^2+A(x)\{b_{m-i}x+b_{m-i-1}\}modF(x)]x^{m-i-1}\\&\quad modF(x)=P_ix^{m-i-1}modF(x)\\&\vdots\\K_m&=K_{m-2}+A(x)b_0modF(x)\\&=(P_{m-2}x+A(x)b_0)modF(x)=P_m\end{aligned}$$

위 식의 결과를 보면 알 수 있듯이 결국 K_m은 $P(x)=A(x)B(x)modF(x)$와 같으며 이를 정리하면 알고리즘 1과 같다.

알고리즘 1

READ $A, B, F, M, P=0; i=M-1$
DO WHILE $i>2$
  IF $b_ib_{i-1}=00$ THEN $A\leftarrow 0$
  ELSE IF $b_ib_{i-1}=01$ THEN $A\leftarrow A$
  ELSE IF $b_ib_{i-1}=10$ THEN $A\leftarrow Ax+F$
  ELSE IF $b_ib_{i-1}=11$ THEN $A\leftarrow (Ax+F)+A$
  END IF
  $P\leftarrow P+A$
  IF $i>2$ THEN
    $P\leftarrow (PxmodF)xmodF$; $i\leftarrow i-2$
  END IF
END DO
$P\leftarrow (PxmodF)+Ab_0$
P is the Product

위의 알고리즘을 구현하기 위해서는 $Ax+F$, $(Ax+F)+A$와 Px^2을 처리하는 부분이 필요하다. $Ax+F$를 얻기 위해 $A(x)$에 x를 곱하면

$$A(x)x=a_{m-1}x^m+a_{m-2}x^{m-1}+\cdots+a_1x^2+a_0x\tag{2}$$

이며 이 때, x^m은 $F(x)$를 이용하여 식 (3)과 같이 구할 수 있다.

$$x^m=f_{m-1}x^{m-1}+f_{m-2}x^{m-2}+\cdots+f_1x+f_0\tag{3}$$

위 식 (2)와 (3)을 이용하여 $Ax+F$는 식 (4)와 같이 구할 수 있으며 $(Ax+F)+A$는 식 (4)에 A를 한번 더 XOR 연산한 것과 같다.

$$\begin{aligned}A(x)x+F&=a_{m-1}(f_{m-1}x^{m-1}+\cdots+f_0)+a_{m-2}x^{m-1}\\&\quad+\cdots+a_1x^2+a_0x\\&=(a_{m-1}f_{m-1}+a_{m-2})x^{m-1}+\cdots+(a_{m-1}f_0)\end{aligned}\tag{4}$$

Px^2을 구하기 위해 $P(x)$에 x^2을 곱한 결과는 식 (5)와 같다.

$$P(x)x^2=p_{m-1}x^{m+1}+p_{m-2}x^m+\cdots+p_1x^3+p_0x^2\tag{5}$$

식 (3)을 이용하여 x^{m+1}을 구하면

$$x^{m+1}=f_{m-1}x^m+f_{m-2}x^{m-1}+\cdots+f_1x^2+f_0x\tag{6}$$

와 같으며 식 (3)과 (6)을 식 (5)에 대입함으로써 다음의 결과를 얻을 수 있다.

$$\begin{aligned}P(x)x^2=p_{m-1}&(f_{m-1}x^m+f_{m-2}x^{m-1}+\cdots+f_0x)+\\&p_{m-2}(f_{m-1}x^{m-1}+f_{m-2}x^{m-1}+\cdots+f_0)+\\&p_{m-3}x^{m-1}+\cdots+p_1x^3+p_0x^2\end{aligned}$$

기약다항식의 특성상 $f_{m-1}=0$이므로 다시 정리하면 아래의 식과 같다.

$$\begin{aligned}P(x)x^2=&(p_{m-1}f_{m-2}+p_{m-3})x^{m-1}+\\&(p_{m-1}f_{m-3}+p_{m-2}f_{m-2}+p_{m-4})x^{m-2}+\\&\cdots+(p_{m-1}f_2+p_{m-2}f_3+p_1)x^3+\\&(p_{m-1}f_1+p_{m-2}f_2+p_0)x^2+\\&(p_{m-1}f_0+p_{m-2}f_1)x+p_{m-2}f_0\end{aligned}$$

위의 과정을 통해 얻은 수식을 이용하여 $A(x)$와 $P(x)$의 값을 MSB부터 2비트씩 묶어서 처리하면 알고리

즘 2를 얻을 수 있다.

알고리즘 2

READ $A, B, F, m, P = 0, i = j = m-1$

$a_{-1} = f_{-1} = f_{-2} = p_{-1} = p_{-2} = p_{-3} = p_{m-1} = p_{m-2} = 0$

DO WHILE $i \geq 2$

 DO WHILE $j \geq 0$

  IF $b_i b_{i-1} = 00$ THEN $a_j a_{j-1} \leftarrow 00$

  ELSE IF $b_i b_{i-1} = 01$ THEN $a_j a_{j-1} \leftarrow a_j a_{j-1}$

  ELSE IF $b_i b_{i-1} = 10$ THEN $a_j a_{j-1} \leftarrow a_{j-1} a_{j-2}$

  ELSE IF $b_i b_{i-1} = 11$ THEN $a_j a_{j-1} \leftarrow (a_{j-1} \oplus a_j)(a_{j-2} \oplus a_{j-1})$

  END IF

  $p_j p_{j-1} \leftarrow ((p_{m-1} f_j \oplus (a_{m-1} b_j \oplus p_{m-2}) f_j \oplus p_{j-2}) \oplus a_j)$

  $((p_{m-1} f_{j-2} \oplus (a_{m-1} b_j \oplus p_{m-2}) f_{j-1} \oplus p_{j-3}) \oplus a_{j-1})$

  $j \leftarrow j-2$

 END DO

 $i \leftarrow i-2; j = m-1$

END DO

DO WHILE $j \geq 0$

 $p_j p_{j-1} \leftarrow (p_{m-1} f_j \oplus p_{j-1} \oplus b_0 a_j)(p_{m-1} f_{j-1} \oplus p_{j-2} \oplus b_0 a_{j-1})$

 $j \leftarrow j-2$

END DO

Ⅲ. $GF(2^m)$ 유한체 곱셈기의 Radix-4 시스톨릭 어레이 구조

제안한 알고리즘 2에 대한 DG(dependency graph)는 그림 1과 같으며 모두 $((m+1)/2)^2$개의 PE로 구성되어 있다. 그래프를 바탕으로 projection vector $\vec{d} = (1,0)$과 scheduling vector $\vec{s} = (1,-2)$를 이용하여 시스톨릭 매핑하면 행방향의 양방향 데이터 흐름이 제거되어 그림 2와 같은 시스톨릭 구조를 얻을 수 있다. '●' 은 한 사이클의 지연을 갖는 소자를 의미하며 시스톨릭 구조에서 셀마다 적절한 입력과 출력을 맞추기 위해서 삽입된다. 이 때 2차원 배열 상의 PE와 1차원 배열 상의 PE의 내부 구조는 동일하다.

그림 2의 PE($m-1$)부터 PE(2)까지의 A 셀 내부회로는 그림 3(a)와 같고 A 셀 내부의 PE의 내부회로는 그림 3(b)와 같다. A 셀은 ctrl 신호가 0일 때 $a_{m-1}, p_{m-1}, p_{m-2}, b_i$ 및 b_{i-1}의 값을 저장하며 1일 때 각각의 셀에서 곱셈 연산을 수행한다. 또한, 입력 단과 출력 단 모두에 13개의 플립플롭이 사용되며 2 사이클이 소요된다. 그림 3(b)의 출력단의 실선은 입력단의 x_i 값이 바로 출력단의 x_o로 연결되었음을 의미한다.

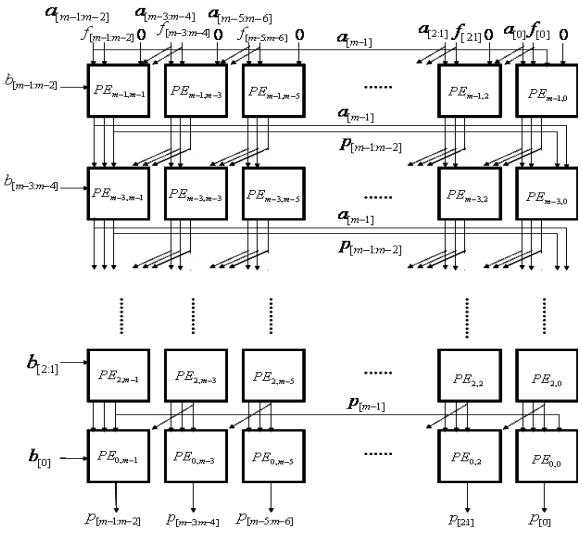
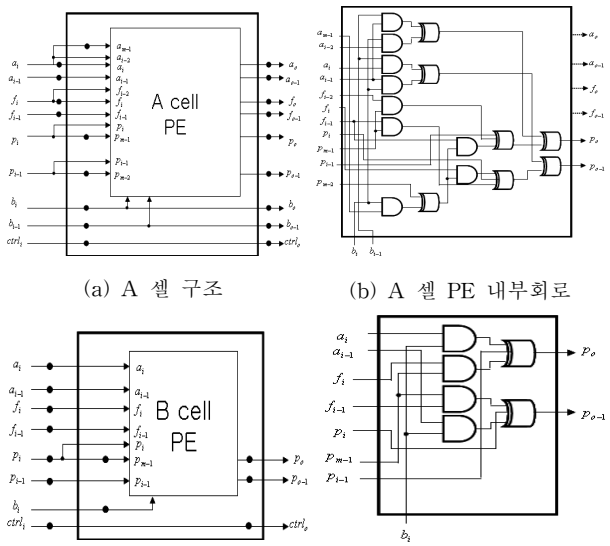


그림 1. $GF(2^m)$ Radix-4 유한체 곱셈기의 2차원 DG

m이 소수이기 때문에 마지막의 B 셀은 b_0 한 비트에 대한 유한체 곱셈 연산만을 수행하게 된다. 따라서 $(Px+F)+Ab_0$ 연산만 수행하면 되므로 그림 3(c)(d)와 같이 회로가 매우 단순하게 구현된다. 입력과 출력 단에 모두 10개의 플립플롭이 사용되며 B 셀에서 모두 2사이클이 소요된다.



(a) A 셀 구조 (b) A 셀 PE 내부회로

(c) B 셀 구조 (d) B 셀 내부 로직

그림 3. 셀 및 PE의 회로도

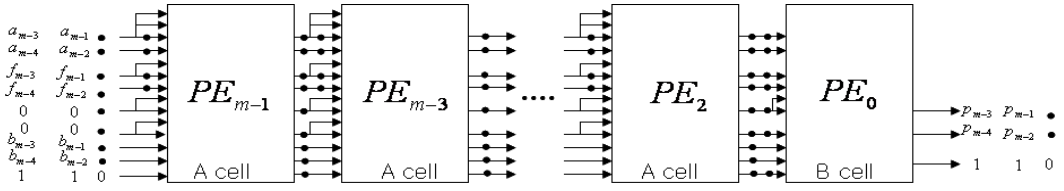


그림 2. $GF(2^m)$ Radix-4 유한체 곱셈기의 시스톨릭 어레이 구조

Ⅳ. 결과 및 성능분석

제안한 알고리즘을 이용하여 곱셈기의 동작을 검증하기 위해서 $GF(2^{193})$에서 동작하는 유한체 곱셈기를 설계하였으며 기약다항식은 $F(x)=x^{193}+x^{15}+1$을 사용하였다.

제안된 구조는 HDL로 모델링 되었으며 Synopsys Design Compiler를 이용하여 게이트 수준의 합성 결과를 얻었다. 각각의 셀은 두 사이클 동안 동작하므로 Latency는 $m+1$사이클이 되며 Radix-4를 사용하므로 $(m+3)/2$ 마다 곱셈 결과를 얻을 수 있다.

하이닉스 0.35㎛ 표준셀 라이브러리를 사용하여 합성한 결과 최대 동작 주파수는 400㎒이며 약 21K 게이트가 사용되었다.

표 1에서 기존 곱셈기구조와 제안한 구조를 비교하였다. [3]에서 제안한 직렬 곱셈기와 비교해서 셀 당 복잡도는 증가하였지만 절반 정도의 셀만 사용하므로 2배의 성능을 가지며 Latency는 1/3로 감소하였다. digit-serial 시스톨릭 유한체 곱셈기를 구현한 논문 [4][5][6]과 비교했을 때 셀 수와 처리량은 거의 동일하다. 하지만 수학적 정리를 통해서 셀의 내부 구조 최적화함으로써 셀의 복잡도와 셀의 최대 지연시간이 감소되었다. 또한 레이턴시가 $m+1$사이클에 그치게 되어 기존의 2 digit-serial에 비하여 약 2/3배의 속도에서 결과를 얻을 수 있다.

Ⅴ. 결론

본 연구는 효율적인 면적과 연산시간을 가지고 동작하는 $GF(2^m)$ 유한체 곱셈기 구현하기 위하여 수학적 정리를 통하여 승수와 피승수를 2 비트씩 묶어서 처리하는 Radix-4 알고리즘 및 VLSI 설계를 위한 시스톨릭 어레이 구조를 제안하였다. 본 연구에서 제안한 Radix-4 시스톨릭 어레이 구조의 유한체 곱셈기는 기존의 구조보다 적은 면적에서 고속으로 동작할 수 있으므로 많은 응용에 적용 가능할 것으로 본다.

감사의 글

저자들은 본 연구를 위하여 설계 환경을 제공하여 준 IDEC(IC Design Education Center)에 감사드린다.

참고문헌

[1] G.B.Agnew, R.C.Mullin, I.M.Onyszchuk, and S.A.Vanstone, "An Implementation for a Fast Public-Key Cryptosystem," Journal of Cryptology, vol.3, pp.63-79, 1991

[2] E.R.Berlekamp, "Bit-Serial Reed-Solomon Encoders," IEEE Trans. on Information Theory, vol.28, issue.6, pp.869-874, Nov. 1982

[3] Chin-Liang Wang, and Jung-Lung Lin, "Systolic Array Implementation of Multipliers for Finite Fields $GF(2^m)$," IEEE Trans. on Circuits and Systems, vol.38, No.7, pp.796-800, 1991

[4] J.H.Guo, and C.L.Wang, "Digit serial systolic multiplier for finite fields $GF(2^m)$," IEE Proc.-comput. Digit. Tech., vol.145, No.2, pp.143-148, 1998

[5] K.W.Kim, K.J.Lee, and K.Y.Yoo, "A New digit-serial systolic multiplier for finite fields $GF(2^m)$," Info-tech and Info-net, 2001. proc. ICII 2001. vol.5, pp.128-133, 2001

[6] C.H.Kim, S.D.Han, and C.P.Hong, "An efficient digit-serial systolic multiplier for finite fields $GF(2^m)$," ASIC/SOC conference, 2001. Proc. 14th Annual IEEE international, pp.361-365, 2001

표 1. 기존 유한체 곱셈기와 제안된 곱셈기간의 성능비교

	Ref.[3] SISO	Ref.[4] L=2	Ref.[5] L=2	Ref.[6] L=2	proposed
PE 개수	m	$(m+1)/2$	$(m+1)/2$	$(m+1)/2$	$(m+1)/2$
Throughput	$1/m$	$2/(m+1)$	$2/(m+1)$	$2/(m+1)$	$2/(m+3)$
Latency($cycles$)	$3m$	$3(m+1)/2$	$(3(m+1)/2)+1$	$3(m+1)/2$	$m+1$
A cell 복잡도	3 AND2 2 XOR2 10 FFs 2 MUX2	11 AND2 9 XOR2 20 FFs 4 MUX2	8 AND2 8 XOR2 18 FFs 4 MUX2	10 AND2 8 XOR2 21 FFs 4 MUX2	9 AND2 9 XOR2 19 FFs
Maximum cell delay	$T_{AND2}+2T_{XOR2}+T_{MUX2}$	$2T_{AND2}+5T_{XOR2}+T_{MUX2}$	$2T_{AND2}+4T_{XOR2}+T_{MUX2}$	$3T_{AND2}+2T_{XOR2}+T_{MUX2}$	$2T_{AND2}+4T_{XOR2}$