멀티 세그먼트 카라츄바 유한체 곱셈기의 구현

Implementation of the Multi-Segment Karatsuba Multiplier for Binary Field

오종수*
(Jong Soo OH*)

Abstract - Elliptic Curve Cryptography (ECC) coprocessors support massive scalar multiplications of a point. We research the design for multi-segment multipliers in fixed-size ECC coprocessors using the multi-segment Karatsuba algorithm on $GF(2^m)$. ECC coprocessors of the proposed multiplier is verified on the SoC-design verification kit which embeds ALTERA EXCALIBUR FPGAs. As a result of our experiment, the multi-segment Karatsuba multiplier, which has more efficient performance about twice times than the traditional multi-segment multiplier, can be implemented as adding few H/W resources. Therefore the multi-segment Karatsuba multiplier which satisfies performance for the cryptographic algorithm, is adequate for a low cost embedded system, and is implemented in the minimum area.

Key Words : ECC, Computation Architecture, Information Security, FPGA, Verilog HDL

1. 서 론

암호 알고리즘은 연산자의 크기가 매우 커서 SoC 설계로 구현하면 암호 프로세서의 시스템 버스가 차지하는 면적이 매우 크다. 따라서 구현 면적을 줄이기 위하여 멀티 세그먼트 연산 방식을 사용하면 전체 유한체 크기의 시스템 버스의 수와 길이를 최소화할 수 있다. 가장 간단한 멀티 세그먼트 연산 방식에는 전통적인 교과서적 곱셈 방법(School Book Multiplication, SBM)이 있다. 세그먼트 수가 k 일 때 SBM은 $k\times k$ 번의 부분곱 연산이 필요하다. $GF(2^m)$에서는 멀티 세그먼트 Karatsuba 곱셈 방법(Multi-Segment Karatsuba Multiplication, MKM)을 사용하여 부분곱의 수를 $k(k+1)/2$로 줄일 수 있음이 알려져 있다[4]. 본 논문은 멀티 세그먼트 곱셈기를 이용하여 시스템 설계 요구에 따라 자유롭게 유한체 크기를 선택할 수 있는 타원곡선 암호 프로세서용 고정 크기 유한체 곱셈기에 집중되어 있다. MKM는 SBM에 비하여 약 2배정도 빠르다.

참고문헌 [4]에서 MKM을 사용하여 타원곡선 암호 프로세서를 구현하는 방법에 관하여 연구하였다. 그들이 사용한 데이터패스 구조는 2개의 입력 피연산자 레지스터, 조합 곱셈기, 부분곱 축적기, 그리고 이들 간 데이터 전송에 필요한 다수의 멀티플렉서들로 구성되어 있다. 실험을 통하여 저자들은 5-세그먼트 MKM와 23비트 조합 연산기를 사용하여 유한체 $GF(2^{113})$를 사용하는 타원곡선 암호 프로세서를 구현하였다. 본 논문에서는 이들의 연구를 이용하여 그래프 이론에 기반을 둔 조직적인 부분곱 스케줄 방법을 제시한다. 제안된 부분곱 스케줄 방법을 사용하면 조합 곱셈기의 크기와 세그먼트 수를 변경하여 임의의 유한체 연산을 지원하는 효율적인 곱셈기를 설계할 수 있다.

* 學生會員 : 慶北大學校 情報保護學科 碩士課程

2. 멀티 세그먼트 곱셈 알고리즘

$GF(2^n)$은 모듈로 특성 다항식 $P(x)$ 기저를 사용하며 차수가 $n-1$ 보다 작은 모든 다항식들의 유한집합과 동형이다. 즉 유한체의 각 원소는

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

로 나타내며 하드웨어 구현 시 n 비트 $GF(2)$ 벡터로 표현할 수 있다. $GF(2^n)$ 상의 "+"와 "×" 연산은 비트별 XOR과 AND 연산으로 구현된다. 유한체 $GF(2^n)$ 값 A 와 B 의 곱 C 는

$$C = A \cdot B = \sum_{k=0}^{2n-2} c_k x^k \mod P(x) \quad (2)$$

$$c_k = \sum_{i=0}^{k} a_i b_{k-i} \quad for\ 0 \le k \le 2n-2 \quad with\ a_i = 0,\ b_i = 0\ for\ i \ge n \quad (3)$$

로 나타내진다.

또한 $GF(2^n)$의 한 원소 A 는 다음과 같이 k 개의 세그먼트로 나누어 표현할 수 있다.

$$A = \sum_{i=0}^{k-1} A_i \hat{x}^i \quad with\ \hat{x} = x^{n/k},\ A_i \in GF(2^{n/k}) \quad (4)$$

예를 들어 세그먼트를 k=3으로 할 때 곱 C=$A\times B$는 식 5와 같이 SBM 방법 혹은 MKM 방법으로 계산될 수 있다. 그림 5를 통하여 SBM 방법을 사용하면 부분곱의 수는 $k\times k$ (=9)이나 MKM 방법을 사용하면 필요한 부분곱의 수는 $k\times(k+1)/2$(=6)이다. 즉 k가 증가하면 필요한 곱셈의 수는 MKM 방법이 SBM 방법의 1/2에 접근한다.

$$
\begin{aligned}
AB &= \left(A_2x^{2n/3} + A_1x^{n/3} + A_0\right)\left(B_2x^{2n/3} + B_1x^{n/3} + B_0\right) \\
&= (A_2B_2)x^{4n/3} \\
&\quad \oplus (A_2B_1 \oplus A_1B_2)x^{n} \\
&\quad \oplus (A_2B_0 \oplus A_1B_1 \oplus A_0B_2)x^{2n/3} \\
&\quad \oplus (A_1B_0 \oplus A_0B_1)x^{n/3} \\
&\quad \oplus (A_0B_0) \\
&= (A_2B_2)x^{4n/3} \\
&\quad \oplus ((A_2 \oplus A_1)(B_2 \oplus B_1) \oplus A_2B_2 \oplus A_1B_1)x^{n} \\
&\quad \oplus ((A_2 \oplus A_1 \oplus A_0)(B_2 \oplus B_1 \oplus B_0) \\
&\qquad \oplus (A_2 \oplus A_1)(B_2 \oplus B_1) \\
&\qquad \oplus (A_1 \oplus A_0)(B_1 \oplus B_0))x^{2n/3} \\
&\quad \oplus ((A_1 \oplus A_0)(B_1 \oplus B_0) \oplus A_1B_1 \oplus A_0B_0)x^{n/3} \\
&\quad \oplus (A_0B_0)
\end{aligned}
\tag{5}
$$

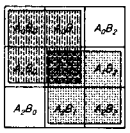
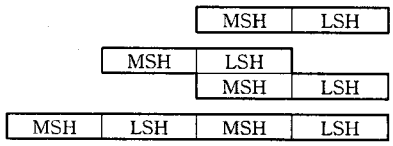


그림 1. MKM 곱셈 항 계산 방법

그림 1의 우상좌하 대각선 $A_0B_2+A_1B_1+A_2B_0$는 대각선 상의 3셀 A_0B_2, A_1B_1, A_2B_0만 정사각 블록들에 덮인 회수가 홀수이도록 해서 알 수 있다. 이렇게 MKM 곱셈 항들을 계산하는 일반적인 계산식은 아래처럼 나타난다.

$$
\begin{aligned}
&S_{l,m} = M_{l,m} \oplus R_{l,m},\ m > 0,\ l \geq 0 \\
&R_{l,m} = \begin{cases} M_{l,m-1} \oplus M_{l+1,m-1} \oplus R_{l+1,m-2}, & m > 1 \\ 0, & m = 1 \end{cases} \\
&M_{l,m} = \left(\bigoplus_{i=0}^{m-1} A_{l+i}\right) \otimes \left(\bigoplus_{i=0}^{m-1} B_{l+i}\right),\ m > 0
\end{aligned}
\tag{6}
$$



(패턴 1)

(패턴 2)

(패턴 3)

그림 2. 3가지 가산 패턴

MKM 방법은 식 5를 사용하여 얻은 곱셈 항들을 가산 위치를 고려하여 그룹화하는데, 위 3가지 패턴을 쉽게 얻는다. SBM 방법의 경우 패턴 1만 사용하는 곱셈 계산 방법이다.

동일한 가산 위치의 부분곱 축적 순서를 매기는 한가지 방법으로 가산 위치가 높은 순서로 부분곱을 축적하고 같은 차수의 경우 임의의 순서를 사용한다. MKM 방법의 경우 부분곱 축적 알고리즘은 부분곱 축적뿐만 아니라 부분곱 곱셈기의 입력 축적도 고려하여야 한다. 하나의 입력 축적에서 다른 입력 축적으로 이동하는 방법은 두가지 방법이 있다. 새로운 패턴이 2개의 유한체 서브워드만 사용하는 경우에는 그림 3의 입력 선택 멀티플렉서로부터 직접 입력한다. 아니면 새로운 패턴(패턴 3)의 서브워드 합의 곱으로 나타나있는 경우 그 전 패턴(패턴 2)에서 새로운 입력 A_0, B_0을 각각 비트별 XOR 연산을 하여 얻는다.

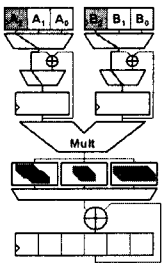


그림 3. MKM 곱셈기 데이터패스

참고문헌 [4]에서 사용한 스케줄 방법의 경우 도착 패턴의 가산 위치가 시작 패턴의 가산 위치와 같거나 보다 한 워드만큼 낮은 경우에만 이동할 수 있다. 그러나 더 큰 유한체 연산을 위해 k가 증가하면 정확한 스케줄을 구하는 것은 매우 흥미로운 문제가 된다. 그림 3의 곱셈기 데이터패스를 보면 입력 축적과 곱 연산 결과 축적이 파이프라인화 되어 있어 k=3인 경우 곱셈은 패턴의 수에 1을 더한 7클록만에 곱셈을 완료할 수 있음을 알 수 있다. 하드웨어 구현에서는 유한체 정의 원시 다항식에 따른 다항식 축소 쉬프트 회로를 추가하면 k 서브워드 크기의 출력 레지스터를 사용할 수 있다. 이는 그림 3에서 출력 레지스터 크기를 3개의 서브워드로 줄일 수 있으며 별도의 다항식 축소 회로가 불필요하다는 것을 의미한다. 이 데이터패스 사용하여 부분곱을 축적하는 스케줄을 구하면, k가 6이상이 되면 부분곱 입력을 위해 추가적인 클록 사이클이 필요하며 그 동안 출력 레지스터는 고정되어 있어야 한다. 이를 우리는 파이프라인 중지라 말한다.

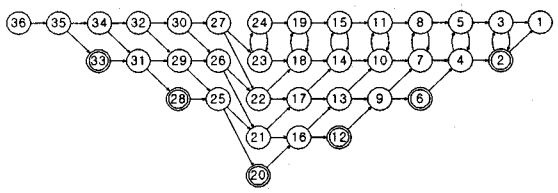


그림 4. MKM 패턴 이동 그래프 (k=8)

그림 4는 k=8인 경우 MKM 패턴 사이의 이동 가능 관계를 보여주는 방향성 그래프로 나타낸 것이다. 그림에서 이중원 노드 33, 28, 20, 12, 6, 2는 SBM의 경우처럼 직접 입력이 가능한 노드로 현재 레벨과 그전 레벨의 노드로부터 이동이 가능하다. 이 그래프에 그림 3의 데이터패스를 사용하여 구한 곱셈 연산 스케줄은 36−35−34−33−31−32−30−28−29−25−26−27−22−23−24−20−21−16−17−18−19−15−14−12−13−9−10−11−8−7−6−4−5−3−2−1로 나타낼 수 있다. 이 스케줄에서 31→32, 28→29, 25→26, 26→27, 22→23, 20→21, 16→17, 17→18, 12→13, 9→10의 이동에는 입력 축적에 2클록이 소요된다. 따라서 곱셈 계산 시간은 47클록이 소요된다. 참고로 SBM 방법의 경우 65클록이 소요된다. k가 증가할수록 파이프라인 중지로 인한 추가 클록 사이클이 소요되는 노드 이동이 급격히 증가한다.

그림 3의 데이터패스에서 출력 레지스터를 좌우로 이동할 수 있으면 그림 4는 에지의 화살표를 제거하여 무방향 그래프로 MKM 패턴 이동을 표현하게 할 수 있다. 이 경우의 이중원 노드들은 현재 레벨과 전후 레벨로부터 직접 이동이 가

능하고, 스케줄 결정 문제는 Hamilton 회로 문제로 귀결된다. 이 경우 36-35-34-33-31-28-25-29-32-30-27-22-26-21-20-16-12-9-13-17-14-18-23-24-19-15-11-10-7-8-6-4-5-3-2-1로 스케줄이 나타난다. 이 그래프 문제는 매우 흥미있는 문제로, 이 문제에 대하여 자세히 연구한 결과 k=16까지 곱셈 연산 스케줄이 있음을 확인하였다. 유한체 정의 다항식의 특성 상 우측 쉬프트 회로는 좌측 쉬프트보다 다소 복잡하나 추가되는 쉬프트 회로에 의한 로직 부담은 그리 크지 않다. 다만 연산 결과 레지스터에 추가적인 멀티플렉서가 필요한 이유로 좌우 이동으로 얻을 수 있는 입력 선택 멀티플렉서의 절감 효과가 반감된다.

3. 구현 및 실험 결과

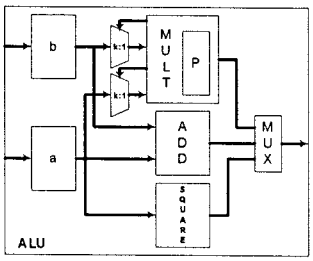


그림 5. 저비용 ECC 프로세서를 위한 연산 회로

그림 5에는 본 논문에서 연구한 저비용 타원곡선 암호 프로세서를 위한 연산 회로가 블록 단위로 나타나 있다. a, b 레지스터는 전체 유한체 크기의 레지스터이다. 본 논문에서는 참고문헌 [1]에서 연구된 서브블록 조합 회로 곱셈기를 이용하여 저비용 타원곡선 암호 프로세서를 설계한다.

실험에서는 MKM 곱셈 방법을 FPGA 상에서 구현하여 SBM 곱셈 방법의 구현 결과와 비교한다.

표 1. 여러가지 곱셈기들의 FPGA 구현 결과 ($GF(2^{128})$)

		SBK	UPD	LRS
Resource Usage (Logic Elements)	ALU	1603	1707	1840
	MULT	830	806	1067
	CKM	172	228	229
	CTRL	113	82	70
	RED	22	21	57
	SQUARE	7	7	7
CLK FREQ (MHz)		47.87	43.72	40.42
CLK COUNT (Clocks)		73932	48760	48760

표 1에는 이들 유한체 곱셈 방법들을 이용한 128비트 타원곡선 암호 프로세서의 FPGA 구현에서 연산 회로부에 사용된 로직 셀의 개수를 나타내었다. 여기서 연산 회로 부분은 대체로 그림 5의 회로에 해당하며 곱셈부(MULT)는 그림 3의 회로에서 a, b 레지스터와 입력 선택 멀티플렉서를 제외한 회로에 해당한다. 제외된 부분은 그림 5에서처럼 연산 회로부(ALU)에 포함되었다.

먼저 간단한 SBK 방법을 구현하였다. 표 1에서 SBK 열은 SBK 방법 구현에 사용된 로직 셀 자원의 수를 나타내었다. 표 1에서 보는 바와 같이 전체 연산 회로 중 약 절반이 곱셈 회로(MULT)에 사용되었다. 나머지는 그림 5의 a, b 레지스터, 덧셈기, 제곱기, 곱셈 데이터 서브워드 선택 멀티플렉서, 그리고 연산 결과 선택 멀티플렉서로 구성되어 있다. 표 1에 보는 바와 같이 SBK 설계는 73932클록이 사용되나 다른 설계들은 48760클록이 사용되었다. UPD 열에는 그림 3의 데이터패스에서 입력 선택 멀티플렉서를 2개에서 4개로 증가시킨 데이터패스를 이용한 곱셈기의 구현 결과를 나타내었다. MULT 행에서 알 수 있는 바와 같이 곱셈기의 크기가 최소임을 알 수 있다. LRS 열에는 좌우 쉬프트가 가능한 논리 구조를 가진 곱셈 연산기에 대한 구현 결과가 나타나 있다. RED 행의 논리 셀 수가 다른 경우보다 많은 이유는 다른 설계와는 달리 오른쪽 쉬프트 축소 논리 회로가 포함되어 있기 때문이다. 이 설계는 오른쪽 쉬프트 기능을 제공하기 위하여 전체 유한체 크기에 해당되는 멀티플렉서를 하나 더 사용해야 한다는 부담이 있다.

4. 결 론

멀티 세그먼트 곱셈(MKM)은 암호 연산에 사용되는 크기가 매우 큰 유한체를 k개의 부분으로 나누어 각각의 곱을 이용하여 전체 곱을 계산하는 방법이다. 연구 결과로 UPD 스케줄 방법을 제안하여 곱셈기의 크기를 최소화하여 SBM보다 2배의 성능을 가진 MKM가 최소한의 자원 사용으로 구현될 수 있음을 보였다. 본 논문에서 제안된 타원곡선 암호 회로용 곱셈기는 설계 면적을 최소화하면서 암호 알고리즘의 수행 속도를 최대화할 수 있게 한다.

하드웨어 설계의 효율성은 연산 구조뿐만 아니라 기반 구현 기술에 많은 영향을 받는다. ALTERA FPGA는 내부 버스 설계를 오직 멀티플렉서만 사용해서 구현한다. 따라서 제안된 방법을 ALTERA FPGA 개발 환경에서 고성능 ECC 프로세서 설계에 적용할 경우 전통적인 디지트 시리얼 구현 방법보다는 우수하지 못하다. 하지만 그림 5에서 a, b 레지스터와 곱셈기 입력 선택 멀티플렉서를 제거하고, 이중 포트 메모리로 대체하여 비교해볼 수 있다. 그밖에 ASIC 구현 기술이나 XILINX FPGA 구현의 경우 내부 버스에 삼상 버퍼를 사용할 수 있으므로 효율적인 내부 버스를 설계할 수 있다.

참 고 문 헌

[1] H. Brunner, A. Curiger, and M. Hofstetter, "On Computing Multiplicative Inverses in $GF(2^m)$," IEEE Transactions on Computers, 42(8), pp. 1010-1015, August 1993.

[2] M. Rosing, "Implementing Elliptic Curve Cryptography," Manning Publications Co., 1999.

[3] J. López and R. Dahab, "Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation," CHES '99, Springer-Verlag, pp. 316-327, August 1999.

[4] M. Ernst, M, Jung, F. Madlener, S. Huss, and R. Bluemel, "A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over $GF(2^m)$," CHES 2002, Springer-Verlag, pp. 382-399, August 2002.