임베디드 SoC를 위한 Bus-splitting 기법 적용 ECC 보안 프로세서의 구현

*최선준, 장우영, 김영철
전남대학교 전자정보통신공학과 & RRC HECS
e-mail : msz009@neuron.chonnm.ac.kr, wyjang@neuron.chonnam.ac.kr, yckim@chonnam.ac.kr

An Implementation of ECC(Elliptic Curve Cryptographic)Processor with Bus-splitting method for Embedded SoC(System on a Chip)

*Seon-Jun Choi, Woo-Youg Chang, Young-Chul Kim
Dept. of Electronics and Information Communication ENG, Chonnam National University & RRC HECS

Abstract

In this paper, we designed ECC(Elliptic Curve Cryptographic) Processor with Bus-splitting mothod for embedded SoC. ECC SIP is designed by VHDL RTL modeling, and implemented reusably through the procedure of logic synthesis, simulation and FPGA verification. To communicate with ARM9 core and SIP, we designed SIP bus functional model according to AMBA AHB specification. The design of ECC Processor for platform-based SoC is implemented using the design kit which is composed of many devices such as ARM9 RISC core, memory, UART, interrupt controller, FPGA and so on. We performed software design on the ARM9 core for SIP and peripherals control, memory address mapping and so on.

I. 서론

SoC는 여러 기능을 필요로 하는 하나의 시스템을 칩 하나로 구현함으로써 생산 비용의 절감 및 시장 출하 시간을 단축할 뿐 아니라 칩 하나가 모든 컴퓨팅을 해결함으로써 시스템의 소형화와 저전력, 휴대성과 같은 다양한 이점들을 제공할 수 있다. 하지만, 복잡하고 난해한 시스템 집적 기술을 구현하는 데 얼마나 개발 시간을 단축하여 Time to Market을 최소할 수 있느냐 하는 것이 SoC설계 시의 중요한 문제점으로 부각되고 있다. 따라서 SoC 설계는 기존의 전통적인 설계 방식과는 다른 설계방식을 요구하며 그 대안으로서 떠오르고 있는 것이 IP(Intellectual Property)의 재사용이다. 또한 보안 모듈로서의 ECC에 대한 필요성은 같은 공개키 알고리즘인 RSA에 비해서 같은 키 길이로도 10배 이상의 보안강도를 갖기 때문에 새로운 공개키 보안 알고리즘으로서 각광받고 있으며 그 구현의 최적화에 대한 연구가 계속 진행되고 있다. 따라서 본 논문에서는 임베디드 SoC에 IP로서 적용할 수 있는 ECC 알고리즘 기반의 보안 프로세서를 Bus-splitting 기법을 사용해서 설계하였다[2]

구현 알고리즘에 있어서는 면적의 최소화와 동작속도의 최적화를 위해서 직렬구조이면서도 같은 형식인 Double-and-add 방식에 비해서 1/2의 연산량을 가지는 Radix-4 booth 알고리즘을 사용했고[1][6]구조에 있어서는 Bus 구조에서 저 전력 효과를 가지는 bus-splitting 기법을 사용해서 설계하였다[2]

본 논문은 정보통신부의 출연금 등으로 수행한 정보통신연구개발사업과 IDEC의 CAD Tool 지원사업의 연구결과임.

II. 타원곡선 암호알고리즘

타원곡선 암호 알고리즘은 타원곡선 이산 로그 문제(ECDLP: Elliptic Curve Discrete Logarithm Problem)에 근간을 두고 있다[4]. ECDLP는 타원곡선상의 임의의 한 점 P에 정수 K를 곱한 값이 Q = KP일 때, 점 Q와 P를 알고 있더라도 정수K를 계산하기 어려움을 나타낸다. 따라서 타원곡선 암호시스템을 구현하는데 필요한 핵심연산은 스칼라 곱셈, 즉 Q = KP를 구하는 것이다. 소수체(Prime Field)인 GF(p)와 유한체인 $GF(2^m)$상에서 모두 스칼라 곱셈이 정의되나, 유한체 덧셈연산에서는 캐리가 발생하지 않아 소수체보다 하드웨어 구현이 용이하다는 장점 때문에 대부분 $GF(2^m)$상에서의 연산을 사용하며, 본 논문에서도 $GF(2^m)$상에서 스칼라 곱셈을 계산할 수 있는 연산기를 하드웨어로 구현하였다.

스칼라 곱셈이 이루어지는 타원곡선은 $GF(2^m)$상에서(x,y)인 점들로 구성되어지고, 타원곡선은 $y^2 + xy = x^3 + ax^2 + b$ 의 형태를 가진다. 여기서 $a,b \in GF(2^m)$, $b \neq 0$이다. 타원 곡선 위의 점들은 점 덧셈 연산에 대해서 군을 이루고, 무한 원점 O는 이 덧셈에 대한 가환군의 항등원이 된다. $GF(2^m)$ 상에서 정의된 타원 곡선 군은 유한 개의 원소를 가지게 되고 반올림에 따른 오차가 없기 때문에 이진 컴퓨터 연산에 많이 쓰이게 된다. P와 Q를 타원 곡선 E 위의 두 점이라 하면 아래 같은 연산 공식들이 성립한다 [5].

[$GF(2^m)$ 상의 점 덧셈 연산 알고리즘]

⊙ $P=(x_1, y_1)$와 $Q=(x_2, y_2)$는 타원 곡선 위의 두 점

⊙ 둘 중 하나가 무한 원점이면, 덧셈 결과는 나머지 한 점이다.

⊙ 만일 P = Q이면, 두배점 연산을 사용한다.

⊙ $P \neq Q$이면, $P+Q = R(x_3, y_3)$이고, x_3, y_3의 값은 다음과 같다.

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a,$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1,$$
$$\lambda = (y_1 + y_2) / (x_1 + x_2)$$

[$GF(2^m)$ 상의 두배점 연산 알고리즘]

⊙ $P(x_1, y_1) = Q(x_1, y_1)$는 타원 곡선 위의 같은 한 점

⊙ 만일 $x_1 = 0$이면, 결과 값 2P는 무한 원점 O이다.

⊙ 만일 $x_1 \neq 0$이면, 결과 값은 $2P(x_1, y_1) = R(x_3, y_3)$이고, x_3, y_3의 값은 다음과 같다.

$$x_3 = \lambda^2 + \lambda + a, \; y_3 = x_2$$
$$1 + (\lambda + 1)x_3, \; \lambda = (x_1 + y_1 / x_1)$$

[$GF(2^m)$상의 점 역원 연산 알고리즘]

⊙ $P(x_1, y_1)$는 타원 곡선 위의 한 점

⊙ $-P(x_1, y_1) = R(x_3, y_3)$ 이고, x_3, y_3의 값은 다음과 같다.

$$(x_3, y_3) = -(x_1, y_1) = (x_1, x_1+y_1)$$

위 공식들을 살펴보면 연산 알고리즘을 처리하는 데 필요한 유한체 연산이 얼마나 요구되는지 계산할 수 있다[6]. 점 덧셈 연산에서는 8번의 유한체 덧셈, 1번의 유한체 곱셈, 1번의 유한체 역원 연산, 1번의 유한체 제곱 연산이 필요하다는 것을 알 수 있고, 두배점 연산에서는 4번의 유한체 덧셈, 1번의 유한체 곱셈, 2번의 유한체 제곱 연산, 그리고 1번의 유한체 역원 연산이 필요하다. 이를 종합하여 타원곡선 암호시스템의 연산과정을 크게 네부분으로 나누어 도식하면 아래 그림 1 과 같다.

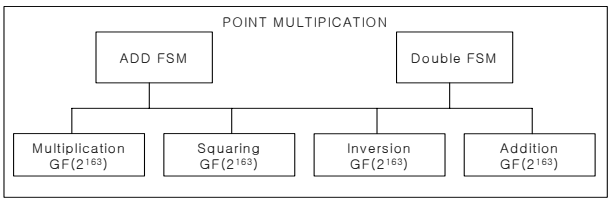

그림 1. 스칼라 곱셈기의 연산 구조

III. 타원곡선상의 스칼라 곱셈

스칼라 곱셈을 한번 수행할때는 여러번의 덧셈과 두배점 연산이 필요하다. 여기에서는 일반적인 Double and Add 방식과 Booth 알고리즘을 이용한 Radix-4 스칼라 곱셈 방식 두 가지를 비교한다.

3.1 Double and Add 방식

m비트의 최상위비트부터 차례로 읽으면서 두배점 연산과 점 덧셈 연산을 수행하는 방식으로 최소 m-1번의 두배점 연산에 더하여 k를 이진수로 표현했을때의 hamming weight만큼의 점 덧셈연산이 필요하다[7].

⊙점 덧셈 연산 : add(), 두배점 연산 : double()

표 1. Double and Add 방식

```
kP :    k = Σ b_i 2^i (b_i ∈ 0,1)
        P: =P(x1, y1)        Q: =P
        for i from m-1 downto 0 do
                Q: =double(Q)
                if b_i = 1 then
                Q: =add(P, Q)
        end (Q = kP)
```

3.2 Radix-4 스칼라 곱셈 방식 방식

kP를 효과적으로 구현하기 위해 modified Booth 알고리즘을 적용하여 Radix-4 스칼라 곱셈방식을 사용한다. 이 방식은 오퍼랜드의 비트열 3비트를 참고하면서 한번에 2비트씩 연산을 처리하여 부분곱의 계산량을 1/2으로 줄인다.[8]. 표 2는 Radix-4 Modified Booth 알고리즘을 나타낸다.

표 2. Radix-4 스칼라 곱셈방식

k_i	k_{i-1}	k_{i-2}	k'_i	k'_{i-1}	동작
0	0	0	0	0	+0
0	1	0	0	1	+P
1	0	0	1	0	+P
1	1	0	0	1	-2P
0	0	1	0	1	-P
0	1	1	1	0	+P
1	0	1	0	1	+2P
1	1	1	0	0	-0

기존의 double-and-add 방식은 m 비트로 이루어진 k의 Hamming weight 값에 맞추어 연산 단계의 수가 결정된다. 임의의 정수 k의 비트열에는 '1'이 몇 번 나올지 알 수가 없기 때문에 이를 확률적으로 계산하기로 한다. 첫 번째 +P 가 기본적으로 선택되고 이후 m-1번의 double( ) 연산이 소요되는데 m비트 중 '1'이 있을 확률은 0.5이므로 m/2번의 add( ) 연산이 소요된다고 예상 할 수 있다. 여기서 Booth 알고리즘을 적용하면 최상위 비트 자리의 0으로 인해 $\lceil (m+1)/2 \rceil$ 번의 연산 단계가 필요하다. Modified Booth 알고리즘을 이용한 Radix-4 스칼라곱셈 방식의 알고리즘은 다음 표3과 같다.

표 3. Radix-4 스칼라곱셈 알고리즘

SET P <- $P(x_1, y_1)$
  Q <- {0, +P, +2P}
  2P <- P+P
  sub <- $P(x_1, y_1+x_1)$
For i from $\lceil m/2 \rceil$ -1 downto 0
  Q <- R^4P
    if R^4 = +P then Q <- Q+P
    if R^4 = +2P then Q <- Q+2P
    if R^4 = -P then Q <- Q+sub(P)
    if R^4 = -2P then Q <- Q+sub(2P)
end (Q =kP)

Ⅳ. Bus-splitting 기법

Bus 구조를 사용할 때 문제가 되는 점은 임의의 하위 모듈이 Bus를 선점하고 데이터를 전송할 경우 그 데이터는 Bus 라인을 타고 그 데이터가 필요없는 모듈까지 전송된다는 점에 있다. 따라서 이때 Bus라인의 적절한 위치에 Gate를 설계하고 데이터의 흐름을 차단함으로서 전력 소모를 줄이는 기법이 Bus-splitting 이다. 그 개요도는 그림 2와 같다.[2]

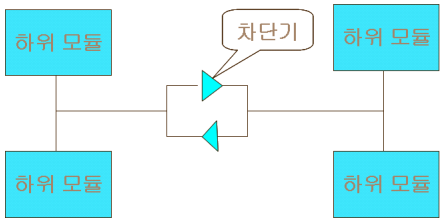

그림 2. bus-splitting 기법

Ⅴ. 임베디드 SoC를 위한 ECC 구현

1. 유한체 곱셈기

유한체 곱셈기는 연산량을 줄이기 Montgomery 알고리즘을 이용하였다.
Montgomery 알고리즘을 이용한 곱셈기는 Reduction 연산을 수행할 필요가 없다. 그리고 Binary Field일 경우 연산과정은 단지 Shifter와 XOR 연산으로만 구현되기 때문에 하드웨어 구현이 용이하다.

2. 유한체 나눗셈기

본 논문에서는 구현한 유한체 나눗셈기는 Brunner의 나눗셈 알고리즘을 이용하여 나눗셈기를 구현하였다.
먼저, Brunner의 알고리즘에서 R,S,U,V 레지스터를 포함하는 회로 블록의 기능은 다음 표 6과 같다. R 레지스터와 S 레지스터의 최상위 비트인 r_m을 참고하여 표에 따라 각 기능 셀 별로 정해진 연산을 수행한다. 그림 2에서는 유한체 나눗셈기의 구조를 나타내고 있다.

표 4. R, S, U, V 블록의 기능

제어신호			레지스터				카운터
rm	sm	δ=0	R	S	U	V	δ
0	x	x	xR	S	xU	V	δ+1
1	0	0	xS	R	xV	U	δ+1
1	0	1	R	xS	U/x	V	δ-1
1	1	0	x(S-R)	R	x(V-U)	U	δ+1
1	1	1	R	x(S-R)	U/x	V-U	δ-1

3. 타원곡선 암호프로세서

본 논문에서 제시한 Radix-4스칼라 곱셈 알고리즘을

이용하는 타원곡선 암호프로세서의 연산 과정은 처음 P, +2P를 연산하기 위해 두배점 연산과 점 덧셈 연산을 스칼라 연산을 하기 전에 수행한다. 그후 m 비트의 k 값을 2비트씩 참조하여 Booth recording을 수행하여 스칼라 연산을 한다. 실제적으로 m 비트의 k 값이 2비트씩 쉬프트하므로 두배점 연산을 3번 하는 대신 4배점 연산을 수행한다. 하드웨어 설계에서는 두배점과 연산과 점 덧셈 연산, 네배점 연산과정은 스칼라 연산이며 유한체상의 연산 덧셈, 곱셈, 나눗셈의 연산만으로 구현된다. 구조 설계에 있어서는 앞서 설명한 Bus-splitting 기법을 사용하였다. 게이트는 그림 3에서 보다시피 연산부와 데이터부 중간에 위치하고 있다. 이것은 Radix-4 Booth 알고리즘에 있어서 최초 값이 입력된 후 연산시에는 그 값들을 필요로 하지 않는 것에 기인한다.

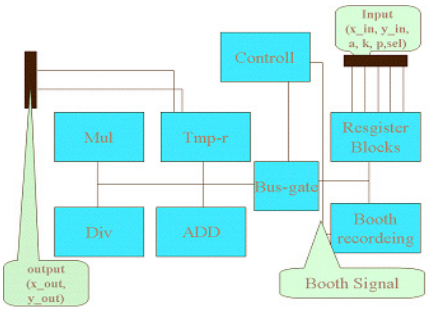

그림 3. 구현된 ECC 구조

검증은 상위레벨 동작 검증을 Modelsim에서 하고 FPGA 검증은 Xilinx 40만 게이트 FPGA칩에서 검증하였다.[7]그 결과는 그림 4와 같다. 최종적인 SoC 환경상에서의 검증은 Quartus4 Tool을 사용해서 한백전자의 SoC-Entry2상의 Excalibur 40만 게이트 칩에서 검증하였다.[4][5][6]구현결과는 그림 5와 같다.

VI. 결 론

본 논문에서는 타원곡선 암호화 알고리즘을 적용한 보안 프로세서를 Bus-splitting 기법을 적용해서 구현한 다음 플랫폼 기반의 SoC 설계장비 위에 설계하고 동작을 검증하였다. 특히, 타원곡선 암호 알고리즘의 구현에 있어 스칼라 곱셈의 경우는 Radix-4를 이용한 Modified Booth 알고리즘을 이용하여 설계하였고 구조에 있어서는 Bus-spliting 기법을 적용함으로서 저전력 설계를 하였다. 임베디드 SoC 환경상에서의 검증은 ARM9 RISC 코어와 FPGA가 하나의 칩으로 구성된 알테라의 Excalibur 칩을 이용해서 검증하였다.

본 논문의 임베디드 SoC를 위한 타원곡선 암호 프로세서는 현재 휴대용 SoC나 임베디드 시스템에서 자주 야기되는 보안상의 문제점을 해결할 수 있는 대안으로 기대된다. 향후 연구 계획은 설계한 프로세서를 임베디드 SoC설계를 위한 임베디드 OS 포팅, 디바이스 드라이버 제작 그리고 응용 소프트웨어의 설계 개발을 수행하는 것이다.

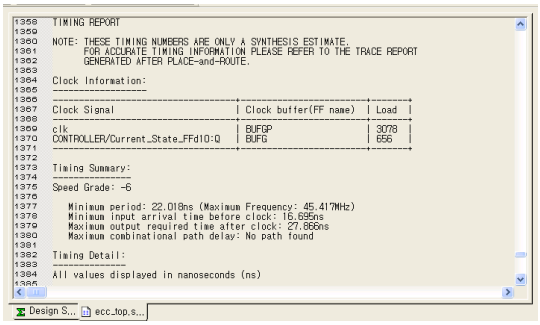

그림 4. Xilinx 합성 결과

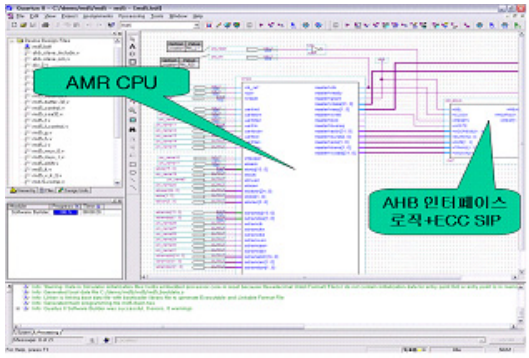

그림 5. Quartus4에서 구현

참고문헌

[1] 문상국, "타원 곡선 암호용 프로세서를 위한 고속 VLSI 알고리즘의 연구와 구현," 연세대학교 전기전자공학과 박사학위 논문, 2001, 12.

[2] 신현철, 2005년도 IT SoC 사업단 저전력 SoC 설계기술 1, 2005년도 하계설계특론 2005, 7

[3] 황정태, "플랫폼 기반 SoC 설계를 위한 보안IP 및 통합 보안 모듈 구현에 관한 연구", 전남대학교 전자정보통신공학과 석사학위 논문, 2004, 12

[4] http://www.arm.com,

[5] http://www.certicom.com

[6] Excalibur Devices, "Hardware Reference Manual Version 3.0" http://www.altera.com, July 2002

[7] Xilinx FPGA Manual,"http://www.xilinx.com"