情報保護學會論文誌
第20卷 第1號, 2010. 2

η_T Pairing 알고리즘의 효율적인 하드웨어 구현*

이 동 건,[1†] 이 철 희,[1] 최 두 호,[2] 김 철 수,[3] 최 은 영,[4] 김 호 원[1‡]
[1]부산대학교 컴퓨터공학과, [2]한국전자통신연구원, [3]창신정보통신(주), [4]한국인터넷진흥원

Efficient Hardware Implementation of η_T Pairing Based Cryptography*

Dong-geoon Lee,[1†] Chul-hee Lee,[1] Dooho Choi,[2] Chulsu Kim,[3]
Eun Young Choi,[4] Howon Kim[1‡]
[1]Department of Computer Science Engineering, Pusan National University,
[2]Electronic and Telecommunications Research Institute(ETRI), Daejeon, KOREA,
[3]Chang Shin Infotel Co.,LTD, [4]Korea Internet & Security Agency

요 약

최근 무선 센서 네트워크 보안 분야에서는 키 교환을 위한 부가적인 통신이 필요 없이 통신 엔터티 상호간에 암호화를 수행할 수 있는 페어링 암호가 주목받고 있다. 본 논문에서는 이러한 페어링 암호의 한 종류인 η_T 페어링에 대한 효율적인 하드웨어 구현을 제시한다. 이를 위해 병렬 처리 및 레지스터/자원의 최적화에 기반한 η_T 페어링 알고리즘에 대한 효율적인 하드웨어 구조를 제안하며, 제안한 구조를 $GF(2^{239})$ 상에서 FPGA로 구현한 결과를 나타낸다. 제안한 구조는 기존의 구현 결과에 비해 Area Time Product에 있어 15% 나은 결과를 가진다.

ABSTRACT

Recently in the field of the wireless sensor network, many researchers are attracted to pairing cryptography since it has ability to distribute keys without additive communication. In this paper, we propose efficient hardware implementation of η_T pairing which is one of various pairing scheme. we suggest efficient hardware architecture of η_T pairing based on parallel processing and register/resource optimization, and then we present the result of our FPGA implementation over $GF(2^{239})$. Our implementation gives 15% better result than others in Area Time Product.

Keywords: η_T pairing, Eta-T pairing, finite field arithmetic, hardware coprocessor, FPGA, elliptic curve

I. 서 론

최근 페어링 기법은 겹선형성(Bilinearity)특성을 이용하여, 기존 암호화 알고리즘으로는 적용할 수 없는 새로운 암호화 기법에 적용되면서 많은 관심을 받고 있다. 페어링은 A. Menezes 등[1]과 G. Frey와 H.G. Rück[2]에 의해 암호학적인 도구로 사용되기 시작하였으며, 이후 ID 기반 암호화 기법[3,4]과 ID 기반 서명 기법[5,6,7], 삼자간 키 공유 기법[8], 짧은 서명 기법[9], ID 기반 인증키 공유 기법[10]등 다양한 암호학적 응용 분야에서 연구되었다.

페어링을 이용한 암호화 기법뿐만 아니라 페어링 연산을 위한 알고리즘에 관한 연구도 활발히 진행되고 있다. Barreto 등[11]과 Galbraith 등[12]은 기존의 Miller 알고리즘[13]에서 불필요한 연산을 제거함으로써 효율적인 알고리즘을 제시하였다. Duursma와 Lee[14]는 위수가 3인 필드 상에서의 페어링 연산에 대한 공식을 제시하였고, Kwon[15]은 위수

접수일(2009년 11월 17일), 게재확정일(2010년 1월 11일)
* 이 논문은 2009년 교육과학기술부로부터 지원받아 수행된 연구임. (지역거점연구단육성사업/차세대물류IT기술연구사업단)
* 본 연구는 2008년도 부산대학교 교내학술연구비(신임교수연구정착금)에 의한 연구임.
† 주저자, guneez@pusan.ac.kr
‡ 교신저자, howonkim@pusan.ac.kr

가 2인 필드상의 페어링 연산에 대한 공식을 제시하였다. Barreto 등[16]은 이후 Tate 페어링의 메인 루프(loop) 계산을 짧게 하기 위해 초특이 타원 곡선상의 Eta 페어링의 개념을 도입하였으며, Hess 등[17]은 이것을 일반 타원 곡선상의 Ate 페어링으로 확장시켰다.

페어링을 구현함에 있어 범용 마이크로프로세서나 마이크로컨트롤러를 사용하여 소프트웨어로 구현할 경우, 작은 위수를 가지는 유한체 연산에 적합하지 않고, 속도가 느리기 때문에 고성능의 전용 하드웨어 보조 연산기의 구현이 필요하게 되었다. Shu 등[18]과 Keller 등[19]은 유한체 $GF(2^m)$ 상의 Tate 페어링 구현에 관해 연구하였고, Ronan 등[20,21]은 초타원곡선 상의 Tate 페어링과 η_T 페어링의 구현을 연구하였으며, Jiang 등[22]은 $GF(3^m)$상의 η_T 페어링의 하드웨어 구현을 연구하였다.

본 논문에서는 η_T 페어링 알고리즘에 대한 효율적인 하드웨어 구현을 제시한다. II장에서는 구현하고자 하는 η_T 페어링 알고리즘을 간략히 언급하고, III장에서 제안하는 페어링 모듈 설계에 대하여 설명한 후, IV장에서 페어링 모듈의 구현에 필요한 $GF(2^m)$상의 유한체 연산기 구현에 관하여 설명한다. 그리고 V장에서 FPGA 합성 및 시뮬레이션 후 나타난 결과를 나타내고, 칩면적과 시간 측면에서 성능 분석을 한 뒤 기존 연구와 비교하고, VI장에서 결론을 맺는다.

II. η_T 페어링 알고리즘

이 장에서는 $GF(2^m)$상에서의 Tate 페어링을 변형한 형태인 η_T 페어링 알고리즘에 대해 다루고자 한다. 본 알고리즘은 참고문헌[23]에 제시되어 있다. 본 알고리즘은 Barreto 등[16]의 η_T 페어링 알고리즘을 개선하고, 제곱근 연산을 빠르게 수행할 수 있으며, Karatsuba[24] 기법을 적용하여 확장체 곱셈 연산을 적용하기에 용이하다. 또한 Torus 기반의 빠른 final exponentiation이 가능하다는 장점을 가진다. 또한, Tate 페어링으로 쉽게 변환 가능하다는 장점을 가진다.

본 연구에서 사용한 알고리즘이 알고리즘 1에 나타나 있으며, $GF(2^m)$ 상의 초특이 타원 곡선 E_b를 다음 식(1)과 같이 정의한다.

$$E_b/F_{2^m}: Y^2+Y=X^3+X+b, \quad where\ b=0 \text{ or } 1,\ m\ odd \tag{1}$$

알고리즘 1 F_{2^m}상의 곡선 E_b위에 변형된 η_T 페어링[23]

입력: $P=(\alpha,\beta)$, $Q=(x,y)$, 단 $\alpha,\beta,x,y\in F_{2^m}$
출력: $\eta_T(P,\psi(Q))$

1: $w\leftarrow\alpha+\frac{(m-1)}{2}$
2: $f\leftarrow w\cdot(x+\alpha+1)+y+(\beta+b+\epsilon_{(m+1)/2})+(w+x)s+t$
3: for $i=0$ to $(m-1)/2$ do
4: $w\leftarrow\alpha+\frac{(m+1)}{2}$
5: $g\leftarrow w\cdot(x+\frac{(m+1)}{2})+y+(\beta+b+\epsilon_{(m-1)/2})+(w+x)s+t$
6: $f\leftarrow f\cdot g$
7: if $i<(m-1)/2$ then
8: $\alpha\leftarrow\sqrt{\alpha},\beta\leftarrow\sqrt{\beta},x\leftarrow x^2,y\leftarrow y^2$
9: end if
10: end for
11: return $f^W=f^{(2^{2m}-1)(2^m+1-\epsilon^{(m+1)/2})}$
(Final Exponentiation)

본 논문에서 구현한 페어링 암호는 m이 239인 $GF(2^{239})$상에서 구현되었으며, b값은 1로 정하였다. 사용한 기약 다항식은 $p(z)=z^{239}+z^{36}+1$이다. $GF(2^m)$의 확장체인 $GF(2^{4m})$은 기저 $\{1,s,t,st\}$로 표현되며, $GF(2^{4m})$상의 원소 f는 $f=f_0+f_1s+f_2t+f_3st$과 같이 정의 된다.

알고리즘 1에서 ϵ_i값과 ϵ 값은 다음과 같이 정의된다.

$$\epsilon_i=\begin{cases}0 \text{ if } i\equiv 0(\bmod 4) \text{ or } 1(\bmod 4),\\ 1\ otherwise.\end{cases} \tag{2}$$

$$\epsilon=\begin{cases}-1, \text{ if } (m\equiv 1,7(\bmod 8) \text{ and } b=1)\\ \quad \text{or } (m\equiv 3,5(\bmod 8) \text{ and } b=0)\\ 1\ otherwise.\end{cases} \tag{3}$$

$m=239$인 경우, $\epsilon_{(m+1)/2}$ 값은 0이 되며, $\epsilon_{(m-1)/2}$은 1이 된다. 그리고 ϵ 값은 $m=239$인 경우 $m\equiv 7(\bmod 8)$이고, b값이 1이므로 ϵ 값은 -1로 고정 된다.

알고리즘 2는 알고리즘 1을 $m=239$, $b=1$인 경우로 적용한 것이다. 알고리즘 1의 step 4에 있는 $\frac{(m+1)}{2}(\bmod 2)$은 0이 되므로, step 5의 w는 α로 치환할 수 있다. 또한 하드웨어로 구현될 때는 알고리즘 1의 step 8(알고리즘 2의 step 7)을 위한 로직이 만들어 지고, 이것은 다른 연산과 동시에 수행될 수 있기 때문에, 마지막 $i=120$ 일 때 이 부분이 실행되어도 계산 결과에는 아무런 영향이 없으며, 이 연산으로 인한 칩 면적이나 시간에 있어서도 아무런 영향을 미치지 않는다. 따라서 알고리즘 1에 있는 step 7(알고리

알고리즘 2 $F_{2^{239}}$상의 곡선 E_1위에 변형된 η_T 페어링

입력: $P=(\alpha,\beta), Q=(x,y)$, 단 $\alpha,\beta,x,y\in F_{2^{239}}$
출력: $\eta_T(P,\psi(Q))$
1: $w\leftarrow\alpha+1$
2: $f\leftarrow w\cdot(x+\alpha+1)+y+(\beta+1)+(w+x)s+t$
3: for $i=0$ to 119 do
4: $g\leftarrow(\alpha\cdot x)+(y+\beta)+(\alpha+x)s+t$
5: $f\leftarrow f\cdot g$
6: if $i<119$ then
7: $\alpha\leftarrow\sqrt{\alpha},\beta\leftarrow\sqrt{\beta},x\leftarrow x^2,y\leftarrow y^2$
8: end if
9: end for
10: return $f^W=f^{(2^{2n}-1)(2^n+1+2^{(n+1)/2})}$
(Final Exponentiation)

즘 2의 step 6)의 조건 확인 과정은 생략될 수 있다. 다음 절에서는 제안하는 η_T 페어링 모듈에 관하여 설명할 것이다.

III. 제안하는 η_T 페어링 모듈 구현

이 장에서는 II장에서 살펴보았던 η_T 페어링 알고리즘을 이용하여 어떠한 방법으로 페어링 모듈을 구현하였는지에 대하여 설명할 것이다. 구현에서 사용된 세부 연산 모듈의 구현 방법은 IV장에서 제시될 것이다.

우리는 페어링 모듈을 구현함에 있어 가능한 많은 연산을 병렬적으로 처리하도록 하였다. 이는 병렬처리에 강한 하드웨어 구현의 장점을 최대한 활용하기 위함이다. 덧셈 연산이 그 대표적인 예인데, 덧셈 연산의 경우 XOR 회로 하나를 두고 MUX를 이용해 선택을 바꾸어 가며 연산하는 방식을 취하지 않고, 모든 필요한 피연산자 조합에 대해 XOR 회로를 각각 따로 구현하였다. 이로써 XOR 연산을 최대한 병렬적으로 수행할 수 있도록 하였다. 하지만, 곱셈 연산의 경우 칩의 면적을 많이 차지하므로, 곱셈 연산모듈의 입력에 MUX를 배치하여 필요할 때 마다 입력을 바꾸어 가며 사용할 수 있도록 하였다. 이는 병렬성은 떨어지지만, 칩의 면적을 줄이기 위함이다. 본 구현에서 면적을 적게 차지하는 덧셈과 제곱연산은 필요한 조합의 수만큼 연산 모듈을 배치하였으며, 곱셈 연산은 입력에 MUX를 이용해 입력을 선택할 수 있도록 하였다. 또한 제곱근 연산의 경우 앞서본 그림 3과 같이 이미 배치된 곱셈기를 이용할 수 있도록 하였다. 본 구현에서는 곱셈 연산 모듈 2개를 배치하여 곱셈 연산에서의 속도 향상을 꾀하였다. 역승 연산 모듈 역시 면적을 많이 차지하는 모듈 중 하나이지만, 페어링 연산에 있어 단 한번만 사용하는 모듈이기에 하나의 모듈만을 배치하였다.

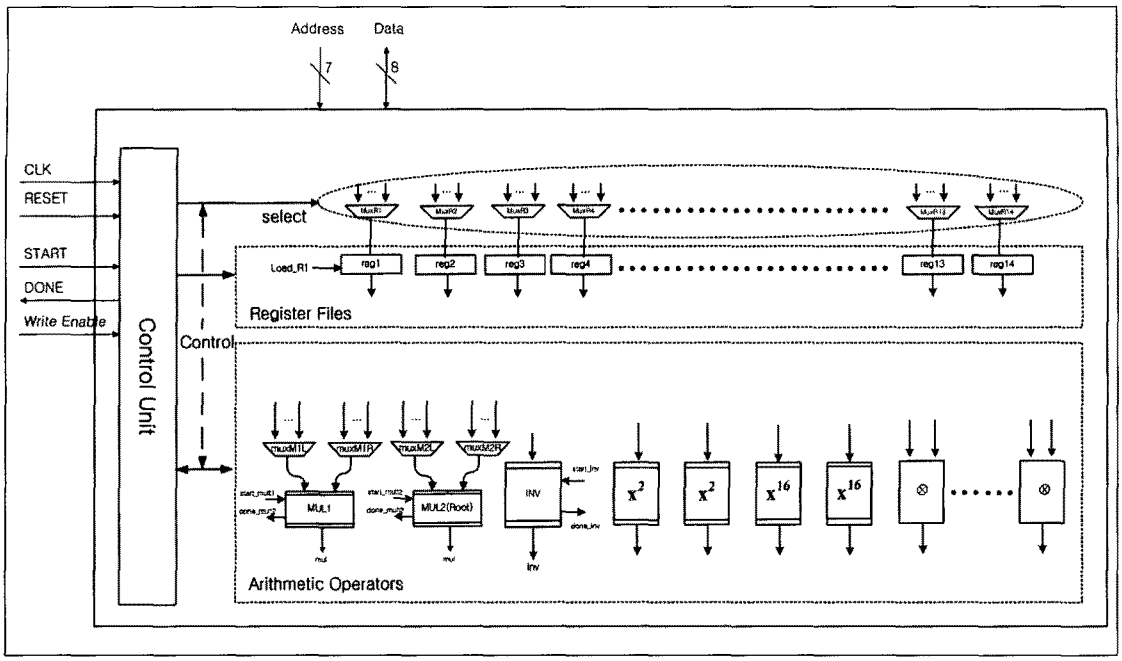
병렬처리를 통해 연산 속도를 증가시키는 것과 함께, 레지스터 배치의 최적화를 통해 모듈의 면적을 줄이려고 하였다. 본 구현에서 $GF(2^{239})$상의 필드 연산을 수행하므로, 연산의 중간값을 저장하기 위한 레지스터는 모두 239 비트가 되어야 한다. 레지스터 하나가 추가될 때마다 최소 239개의 플립플롭을 생성하게 되므로, 중간값 저장을 위한 레지스터 하나를 줄이게 되면 많은 수의 플립플롭을 절약하게 되면서 칩의 면적을 줄일 수 있게 된다. 본 구현에서는 14개의 239 비트 레지스터만을 사용하여 페어링 연산 모듈을 구현하였다.

하나의 레지스터에 대한 입력으로 여러 개의 연산 모듈로 부터의 결과를 MUX를 통해 선택적으로 받도록 하였다. MUX의 입력 하나당 239개의 데이터 경로(data path)가 필요하므로, MUX의 입력으로 받는 다른 모듈의 출력 수가 많으면 그 만큼 MUX의 크기가 커지게 된다. 따라서 각각의 레지스터에 최소한의 입력을 넣기 위해 같은 특정 연산을 특정 레지스터에 집중시키는 방법으로 MUX를 최적화 하였다.

또한 XOR 모듈의 경우 필요한 연산의 수만큼 만들어 배치하였지만, 이것 역시 XOR 모듈 하나를 생성할 때마다 239 비트 단위로 모듈이 생성되므로 최소한의 숫자만 배치해야 했다. 따라서 주로 연산의 입력으로 사용되는 레지스터를 미리 정해두어 무분별하게 XOR 모듈이 많이 생성되는 것을 방지함과 동시에 XOR 모듈을 다음 연산에서 재사용 가능하게 하였다.

3.1 전체 모듈 설계

전체적인 모듈의 대략적인 개요가 그림 1에 나타나 있다. 우선 제어신호를 발생시키는 제어 유닛(control unit)과 입출력 값과 중간 계산 결과를 저장할 레지스터 파일(register file), 그리고 $GF(2^{239})$ 필드 연산을 수행하는 연산 모듈들로 구성되'연산다. 연산 모듈의 경우 egister file연산 모듈은 적은 수rol uniMUX를 이용하여 입력을 바꾸'연가며 사용할 수 있도록 하였고, 그렇지 않은 경우 필요한 수만큼 l uni연산의 병렬성계산 결과를 저제어 유닛은 페어링 연산의 순서에 따라 제어신호를 생성하여 반복적으로 레지스터-연산 모듈을 구동하여 페어링 연산을 수행한다.



〔그림 1〕 η_T 페어링 모듈 블럭도

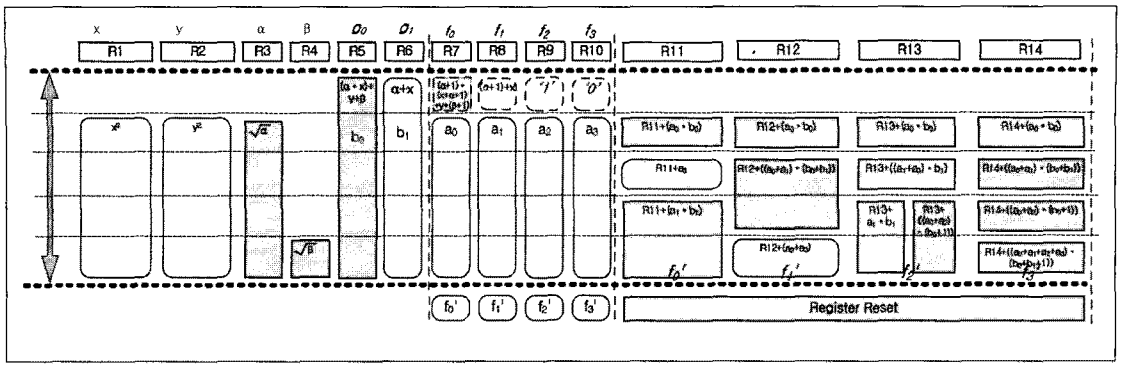
14개의 레지스터를 이용하여 페어링 모듈의 입출력 값과 연산의 중간값을 저장하기 때문에, 알고리즘에 있는 덧셈 연산은 두 개의 레지스터에 있는 값을 입력으로 받는 연산으로 바뀌게 된다. 레지스터를 재사용함으로써, 덧셈 연산 역시 재사용할 수 있게 된다.

모든 연산 모듈의 출력은 각 레지스터의 입력 전에 있는 MUX로 입력되어 제어 유닛의 제어신호에 따라 선택적으로 레지스터에 입력된다. 덧셈 연산 모듈인 XOR의 출력은 레지스터로 입력될 뿐만 아니라, 곱셈 연산 모듈의 입력으로 사용되어 덧셈 결과가 곱셈의 입력으로 즉시 사용될 수 있도록 하였다.

η_T모듈의 경우 기본적으로 clk, reset 신호와 페어링 연산의 시작을 알려주는 start 신호, 페어링 연산이 끝났음을 알려주는 done 신호 외에 입출력으로 $GF(2^{4\cdot 239})$상의 원소를 사용하기 때문에, 239 비트의 입력 4개와 출력 4개가 필요하게 된다. 따라서 이를 각각의 핀으로 설계할 경우 2000개에 가까운 입출력 핀이 필요하게 된다. 이러한 구현이 꼭 필요한 특수한 경우를 제외하면 일반적으로 효율적이지 못하기 때문에, 본 구현에서는 입출력 핀의 개수를 줄이기 위해 입출력에 관여되는 레지스터의 접근을 주소 지정을 통해 가능하도록 하였으며 7 비트의 address핀과 8 비트의 data 핀을 배치하였다. 8 비트 단위로 레지스터에 읽고 쓸 수 있게 하였기 때문에, 하나의 레지스터당 $\lceil 239/8 \rceil = 30$개의 주소가 필요하며 5 비트의 주소선이 필요하다. 총 4개의 레지스터에 $P=(\alpha,\beta)$와 $Q=(x,y)$의 좌표 α, β, x, y를 R1, R2, R3, R4의 4개 레지스터로 입력받아야 하므로 2개의 주소선이 추가로 필요하다. 페어링 결과 출력을 위해 R11, R12, R13, R14의 다른 4개 레지스터를 사용하지만 1개의 추가 주소선을 사용하지 않고, Write Enable 신호를 이용해 〔R1,R2,R3,R4〕와 〔R11, R12,R13,R14〕의 레지스터 셋 중 하나를 선택하도록 하였다.

3.2 페어링의 각 단계에 따른 구현

이번 절에서 우리는 제안하는 모듈을 구현함에 있어 II장에서 제시한 알고리즘의 각 단계가 어떠한 방법을 통해 구현되었는지를 설명하고자 한다. 페어링의 각 단계는 Control Unit 내부에 있는 유한 상태 기계(FSM, finite state machine)에 의해 제어된다. 유한 상태 기계는 각 단계에 맞는 제어신호를 생성한다. 전체적인 동작을 설명할 때, 시간에 따른 레지스터간의 데이터 이동을 그림으로 나타낼 것이다. 그림에 사용되는 각 상자는 특정 연산 모듈의 연산 결과가 해당 레지스터에 입력되는 것을 나타낸다. 어떤 연산이 실행되는지는 상자의 색깔로 구별하며, 이를



〔그림 2〕 for loop 반복 연산 과정

표 1에 나타내었다.

네모난 상자의 연산은 여러 클록 사이클에 걸쳐 연산을 하는 모듈이며, 둥근 모서리 상자의 연산은 조합 회로로만 구성이 가능하므로 한 클록 사이클에 연산이 가능한 것을 나타낸다. 페어링 연산 모듈의 동작 시간 순서를 나타냄에 있어 일반적으로 곱셈 연산이 XOR이나 Squaring 연산보다 시간이 오래 걸리므로, 시간 변화의 순서를 곱셈 연산의 순서로 나타냈다. XOR와 Squaring 등의 연산은 한 클록에 가능하므로 주로 곱셈 연산을 하는 동안 함께 수행되며, 곱셈이 끝나기 전에 레지스터에 결과 값이 반영된다.

〔표 1〕 연산 모듈별 상자 색상

상자	연산 모듈
	Multiplier 1
	Multiplier 2
	Inversion Module
	XOR, Squaring

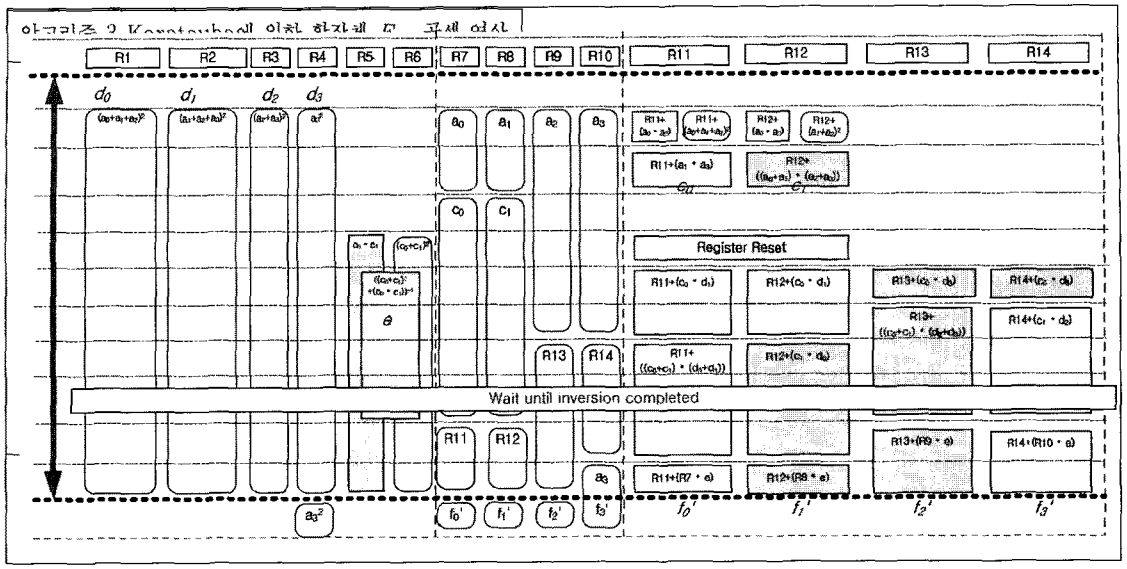
3.2.1 for loop의 시작, 다항식 f 및 g의 계수 계산

먼저 알고리즘 2의 for loop 부분을 그림 2에 나타내었다. R1, R2, R3, R4는 입력 P와 Q의 좌표 x, y, α, β를 입력 받고, R5, R6에는 다항식 g의 계수 중 g_0과 g_1을 계산하여 저장하며, R7, R8, R9, R10에는 다항식 f의 계수 f_0, f_1, f_2, f_3를 계산하여 저장한다. 먼저 다항식 f의 계수 계산을 위해 알고리즘 2의 step 1에서 $w \leftarrow \alpha+1$ 연산을 수행하며, 연산과정에서는 면적을 줄이기 위해 LSB에만 반전기(inverter)를 사용하였다. 이 연산뿐만 아니라 페어링 연산 구현에 등장하는 모든 +1 연산에는 모두 LSB에만 반전기를 사용하는 방식을 사용하였다. 알고리즘 2의 step 1과 step 2는 한 번의 곱셈과 여러 개의 XOR 연산을 통해 계산될 수 있다. 곱셈기의 입력으로 $(\alpha+1)$과 $(x+\alpha+1)$을 각각 입력하면 되므로 step 1과 step 2는 한 번에 실행될 수 있다. 본 구현에서 곱셈기 2개를 사용하였기 때문에, step 4의 다항식 g 계수 계산 부분에 있는 곱셈 연산 $(\alpha \cdot x)$를 함께 실행할 수 있다. 그래서 step 1과 step 2를 for loop 내부에 포함시켰으며, f의 계수 계산의 경우 처음 한번만 실행되고 이후에는 실행되지 않도록 하였다. 다항식 g의 계수 계산에 있어 g_2, g_3의 경우 알고리즘 2의 step 4에서 보이듯이 각각 0과 1로 고정되므로 따로 저장할 필요가 없으며, 이에 따라 확장체 곱셈 연산에 있어 축약된 형태를 쓸 수 있다.

3.2.2 확장체 곱셈

알고리즘 2의 step 3 조건에 따라 for loop는 120번 반복하게 된다. step 5에 있는 확장체 F_{2^m} 연산의 경우에는 Karatsuba 알고리즘을 사용하였는데, 그 이유는 Karatsuba 알고리즘을 사용하면 많은 레지스터들이 사용되지만 곱셈 연산의 횟수를 줄일 수 있는 특징이 있기 때문이다. 그리고 알고리즘 내의 곱셈 모듈은 합성하였을 때 면적을 많이 차지하기 때문에 곱셈 모듈 두 개만을 사용하여 같은 모듈을 재사용하도록 하여 연산 병렬성과 칩 면적의 축소를 동시에 꾀하려 하였다. 알고리즘 3의 연산은 총 9번의 필드 곱셈 연산을 필요로 한다.

알고리즘 2의 step 4에서 g의 t항은 1로 항상 고정되며, st항은 0으로 고정되어 있다. 이를 이용하면 Karatsuba 확장체 곱셈 연산에서의 필드 곱셈 횟수



〔그림 3〕 Final Exponentiation Step 1

를 9번에서 6번으로 줄일 수 있다. g의 t항을 1로, st 항을 0으로 하였을 때의 알고리즘을 알고리즘 4에 나타내었다. 알고리즘 2의 for loop 안에서는 알고리즘 4를 이용하였으며, 알고리즘 3은 Final Exponentiation에서 사용된다.

확장체 곱셈의 중간 연산 결과 및 최종 곱셈 결과인 다항식 f'의 계수 f_0', f_1', f_2', f_3'는 R11, R12, R13, R14에 저장된다. 이는 다시 R7, R8, R9, R10로 옮겨져 다음 loop에서 확장체 곱셈 연산의 입력으로 사용된다. R11, R12, R13, R14에서의 연산들은 주로 곱셈의 결과나 덧셈의 결과를 기존 레지스터에 있는 값에서 누적되는 형태로 제작되었다. 다음 보게 될 Final Exponentian의 중간 계산 값들도 R11, R12, R13, R14에 누적의 형태로 저장되도록 하였는데, 이는 레지스터 입력 선택 MUX의 입력 수를 줄이기 위함이며, 비슷한 연산을 같은 레지스터로 모아서 하도록 하여 입력 수를 최적화시키기 위함이다. 연산의 결과가 누적되어야 하기 때문에, 하나의 loop가 끝나거나 새로운 연산을 하기 위해서 이들 레지스터들을 reset하여 값을 초기화 할 필요가 있다.

알고리즘 2의 step 6에서 step 8까지 걸친 $\alpha \leftarrow \sqrt{\alpha}$, $\beta \leftarrow \sqrt{\beta}$, $x \leftarrow x^2$, $y \leftarrow y^2$ 연산은 II장에서 설명한 것과 같이 step 6의 조건을 검색하지 않고, 매 loop 마다 실행된다. (마지막 결과는 이후 어떠한 연산에도 쓰이지 않기 때문에 이것이 가능하게 된다.) 이 과정은 g가 생성된 이후라면 언제 실행되어도 관계없기 때문에, step 5의 확장체 연산과 함께 실행되어도 관계

알고리즘 4 축약된 Karatsuba에 의한 확장체 $F_{2^{6m}}$ 곱셈 연산

입력: $A = a_0 + a_1 s + (a_2 + a_3 s)t$,
$B = b_0 + b_1 s + (b_2 + b_3 s)t$, $where\ b_2 = 1,\ b_3 = 0$
출력: $A \cdot B$
1: $x \leftarrow (a_0 + a_1)(b_0 + b_1)$,
$z \leftarrow (a_0 + a_1 + a_2 + a_3)(b_0 + b_1 + 1)$
2: $c_0 \leftarrow a_0 b_0 + a_1 b_1$, $c_1 \leftarrow x + a_0 b_0$
3: $t_0 \leftarrow (a_0 + a_2)(b_0 + 1) + (a_1 + a_3)b_1$,
$t_1 \leftarrow z + (a_0 + a_2)(b_0 + 1)$
4: $f_0' \leftarrow c_0 + a_3$, $f_1' \leftarrow c_1 + a_2 + a_3$
5: $f_2' \leftarrow t_0 + c_0$, $f_3' \leftarrow t_1 + c_1$
6: $f' \leftarrow (f_0' + f_1' s) + (f_2' + f_3' s)t$
7: return f'

알고리즘 5 Final Exponentiation - 1단계

입력: $A = a_0 + a_1 s + (a_2 + a_3 s)t$
출력: $A^{2^{3m}-1}$
1: $c_0 \leftarrow (a_0 + a_1 + a_3)^2 + a_0 a_2 + a_1 a_3$,
$c_1 \leftarrow (a_1 + a_2)^2 + (a_0 + a_1)(a_2 + a_3) + a_0 a_2$
2: $d_0 (a_0 + a_1 + a_2)^2$, $d_1 (a_1 + a_2 + a_3)^2$,
$d_2 (a_2 + a_3)^2$, $d_3 \leftarrow a_3^2$
3: $e \leftarrow ((c_0 + c_1)^2 + c_0 c_1)^{-1}$
4: $x \leftarrow (c_0 + c_1)(d_0 + d_1)$, $y \leftarrow (c_0 + c_1)(d_2 + d_3)$
5: $f_0' \leftarrow e(x + c_0 d_1)$, $f_1' \leftarrow e(c_0 d_1 + c_1 d_0)$,
$f_2' \leftarrow e(y + c_0 d_3)$, $f_3' \leftarrow e(c_0 d_3 + c_1 d_2)$
6: $f' \leftarrow f_0' + f_1' s + (f_2' + f_3' s)t$
7: return f'

가 없다. 이 과정은 R1, R2, R3, R4에서 일어나며, 연산의 결과는 다음 loop의 입력으로 쓰여, 다음 loop의 다항식 g의 계수 계산에 사용된다. 제곱근 연산에는 곱셈 모듈이 필요한데, 이는 이미 페어링 연산을 위해 만들어진 곱셈 연산 모듈을 재사용한다. 따라서 loop 내에서 곱셈의 횟수는 f및 g의 계수 계산에 필요한 2번, 확장체 곱셈에 필요한 6번, 제곱근 연산에 필요한 2번을 더하여 총 10번(처음 f 계수 계산을 제외하면 9번) 필요하며, 곱셈 연산 모듈을 2개 배치하였으므로, loop 안에는 총 5개의 단계로 구성된다.

3.2.3 Final Exponentiation

알고리즘 2의 step 10은 Final Exponentiation 단계로 $f^W = f^{(2^{2m}-1)(2^m+1+2^{(m+1)/2})}$의 지수 누승 연산을 하기 위해서 크게 3단계로 나누어 수행한다. 첫 번째로 $X = f^{(2^{2m}-1)}$을 계산하고, 두 번째 단계에서 이 결과인 X를 이용해 $X^{(2^{m}+1)}$과 $X^{2^{(m+1)/2}}$을 각각 계산하여, 세 번째 단계에서 이 두 결과를 곱하게 된다.

첫 번째 연산 단계를 알고리즘 5에 나타내었으며 이의 구현을 그림 3에 나타내었다. R7, R8, R9, R10에 있는 이전 단계의 결과 다항식을 입력으로 받아 알고리즘 5의 step 1에 있는 c_0, c_1의 계산과 step 2에 있는 d_0, d_1, d_2, d_3의 계산이 각각 R11, R12와 R1, R2, R3, R4에서 동시에 진행된다. c_0, c_1 계산이 끝나면 e값 계산을 위한 $(c_0+c_1)^2$ 과 $c_0 \cdot c_1$계산을 각각 R5와 R6에서 진행한다. 이 후 이 두 값을 XOR한 값이 역승 연산 모듈의 입력으로 사용되어 step 3의 역승 연산이 시작된다. 역승 연산과 동시에 병렬적으로 step 4의 x, y 계산과 step 5의 $c_0 \cdot d_1$, $c_1 \cdot d_0$, $c_0 \cdot d_3$, $c_1 \cdot d_2$의 계산이 R11, R12, R13, R14에서 진행된다. 다만 step 3의 e값의 계산이 오래 걸리므로, 곱셈 계산이 먼저 끝나더라도 역승 연산이 끝나고 e 값이 계산되어져 나올 때까지 다음 step을 진행하지 않고 기다려야 한다. e값 계산이 끝나면 step 5의 e와의 곱셈을 하고, $A^{(2^{2m}-1)}$ 계산이 완료된다. 계산 결과인 다항식 f'은 R11, R12, R13, R14에 저장되며, 다음 단계에서 사용되기 위해 R7, R8, R9, R10으로 이동된다.

알고리즘 6 Final Exponentiation - 2-(1)단계

입력: $A = a_0 + a_1 s + (a_2 + a_3 s)t \in T_2(F_{2^{2m}})$
출력: A^{2^m+1}
1: $x_0 \leftarrow -1 + a_3^2 + (a_0 + a_1 + a_2)(a_1 + a_2), x_1 \leftarrow -1 + a_3^2 + (a_0 + a_2)a_1, x_2 \leftarrow -1 + a_3^2 + a_0(a_1 + a_3), x_3 \leftarrow -1 + a_3^2(a_0 + a_1 + a_3)(a_1 + a_2 + a_3)$
2: $b_0 \leftarrow x_1, b_1 \leftarrow x_2 + x_3, b_2 \leftarrow x_1 + x_2, b_3 \leftarrow x_0 + x_1 + x_2 + x_3$
3: $B \leftarrow (b_0 + b_1 s) + (b_2 + b_3 s)t$
4: return B

알고리즘 7 Final Exponentiation - 2-(2)단계

입력: $A = a_0 + a_1 s + (a_2 + a_3 s)t \in T_2(F_{2^{2m}})$
출력: $A^{2^{(m+1)/2}}$
1: $x \leftarrow a_0^{(\frac{m+1}{2})}, x_1 \leftarrow a_1^{(\frac{m+1}{2})}, x_2 \leftarrow a_2^{(\frac{m+1}{2})}, x_3 \leftarrow a_3^{(\frac{m+1}{2})}$
2: $c_0 \leftarrow x_0 + \frac{m+1}{2}x_1 + \epsilon_{(m+1)/2}x_2 + \frac{m+1}{2}(\epsilon_{(m+1)/2}+1)x_3$, $c_1 \leftarrow x_1 + \frac{m+1}{2}x_2 + \epsilon_{(m+1)/2}x_3$, $c_2 \leftarrow x_2 + \frac{m+1}{2}x_3$, $c_3 \leftarrow x_3$
3: $C \leftarrow c_0 + c_1 s + (c_2 + c_3 s)t$
4: return C

Final Exponentiation의 두 번째 연산 단계는 다시 두 개의 계산으로 나누어지며, 이를 알고리즘 6과 7에 각각 나타내었으며, 이의 구현을 그림 4에서 보여준다. 위 두 연산은 구현에서 첫 번째 연산 단계의 결과를 입력으로 받아 병렬로 동시에 계산하게 된다. Step 2-(1)은 R7, R8, R9, R10에 있는 다항식을 이용하여 연산하며, R1, R2, R3, R4에 중간 결과인 x_0, x_1, x_2, x_3을 저장하여 이를 이용해 세 번째 단계의 입력이 되는 다항식 $B = b_0 + b_1 s + b_2 t + b_3 st$의 계수 b_0, b_1, b_2, b_3를 R5, R6, R3, R4에 저장하게 된다. Step 2-(2)는 역시 입력 R7, R8, R9, R10을 이용하여, R11, R12, R13, R14에서 수행되며, 연산의 결과인 다항식 $C = c_0 + c_1 s + c_2 t + c_3 st$의 계수 c_0, c_1, c_2, c_3는 R7, R8, R9, R10으로 이동된다. 알고리즘 7의 경우 $a_i^{2^{(m+1)/2}}$, $(0 \le i \le 3)$ 연산에서 $m = 239$이므로, 각각 a_0, a_1, a_2, a_3에 대해 제곱 연산을 120번 반복 수행하게 된다. step 2에서 $(m+1)/2$는 0이며, $\epsilon_{(m+1)/2}$역시 0이 되므로, $c_1 \leftarrow x_1, c_2 \leftarrow x_2, c_3 \leftarrow x_3, c_4 \leftarrow x_4$가 된다. 이 부분은 제곱 연산 모듈의 반복적인 수행을 통해 제곱 연산을 120번 반복 수행해야 한다. 이 부분에서는 제곱 연산의 횟수를 줄이기 위해 III장에서 보았던 16승 연산 모듈을 별도로 구현하여 수행횟수를 30번으로 줄였다.

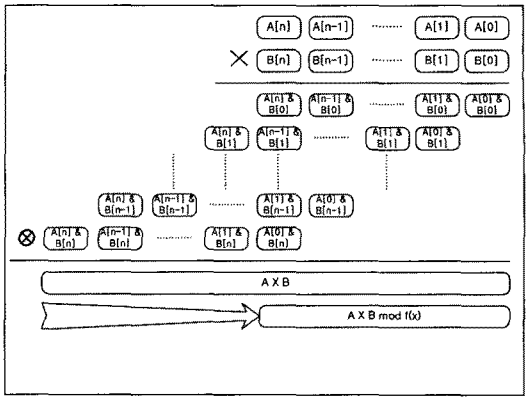
Final Exponentiation의 세 번째 단계에서는 앞서 두 번째 단계인 Step 2-(1)과 Step 2-(2)에서 나온 결과들을 곱하여 주며 이에 관한 구현을 그림 5에서 보여준다. 이 과정에서는 알고리즘 4의 축약된 Karatsuba 알고리즘을 사용할 수 없기 때문

에, 알고리즘 3의 확장체 곱셈 연산을 사용한다. 이 중 $a_0 \cdot b_0$, $(a_0+a_1)\cdot(b_o+b_1)$, $a_1 \cdot b_1$, 의 경우 축약된 경우와 축약 되지 않은 경우 모두 사용되기 때문에, 특정 연산을 두 개의 곱셈 연산 모듈 중 어떤 곱셈 연산 모듈에 배정할지를 결정할 때, 이전 for loop에서 사용한 곱셈 연산 모듈 배치를 그대로 따르도록 하였다. 또한 사용하는 레지스터가 비슷한 $(a_1+a_3)\cdot b_1$과 $(a_1+a_3)\cdot(b_1+b_3)$, $(a_0+a_2)\cdot(b_0+1)$과 $(a_0+a_2)\cdot(b_0+b_2)$, $(a_0+a_1+a_2+a_3)\cdot(b_0+b_1+1)$과 $(a_0+a_1+a_2+a_3)\cdot(b_0+b_1+b_2+b_3)$도 비슷한 연산끼리 곱셈 연산 모듈 배치를 같게 하였다. 이렇게 한 이유는 각 곱셈 연산 모듈로 들어가는 레지스터 입력을 최소화 하여 데이터 경로의 개수 및 곱셈 연산 모듈의 입력 MUX의 크기를 최소화 하고자 함이다.

Final Exponentiation의 최종 연산이 끝나게 되며, 결과인 다항식 f^w의 계수는 R11, R12, R13, R14에 저장된다.

IV. 필드 연산 모듈의 구현

이 장에서는 η_T 페어링 알고리즘을 하드웨어로 구현하기 위해 필드 연산 로직을 어떻게 구현하였는지에 대해 세부적으로 설명하고자 한다. II장에서 제시

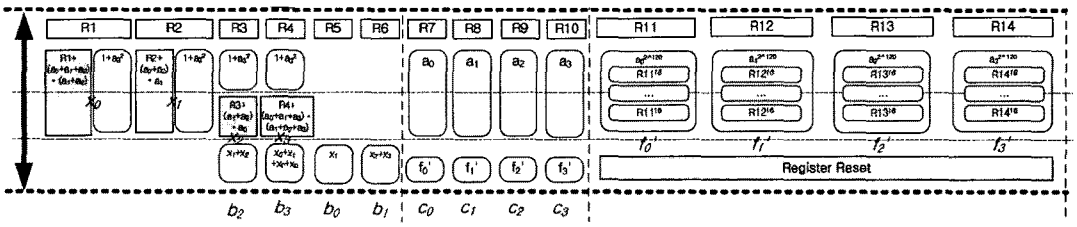


〔그림 6〕 곱셈 연산 1

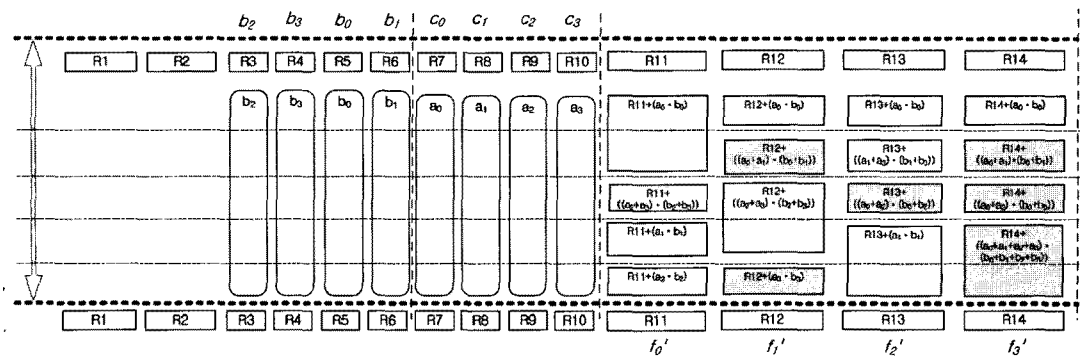
한 알고리즘 2에서는 덧셈, 곱셈, 제곱, 제곱근, 역승연산이 사용되며, 각각의 연산 모듈의 구현은 다음과 같다.

4.1 $GF(2^{239})$ 덧셈 연산

덧셈 연산은 $m=239$개의 XOR 게이트 만으로 쉽게 구현된다. 알고리즘 2의 step 1과 step 2에서 상수 1을 더하는 연산이 있는데, 여기서는 전체 비트에 XOR 게이트를 취하지 않고, LSB(Least Significant Bit)에만 반전기를 취하는 방법으로 구현하여



〔그림 4〕 Final Exponentiation Step 2-(1), 2-(2)



〔그림 5〕 Final Exponentiation Step 3

하드웨어 복잡도를 줄이고자 하였다.

4.2 곱셈 연산

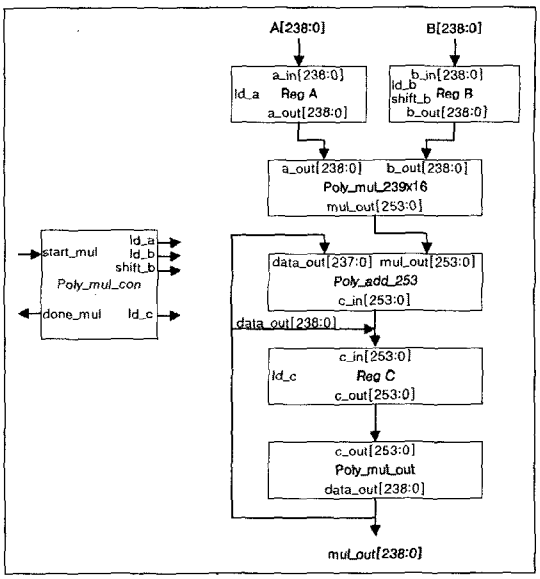
곱셈 연산은 그림 6과 같이 A를 B의 각 비트에 곱하고, 각 비트와 곱한 값들의 합을 모두 더하여 그 결과를 축약(reduction)시키는 방법으로 구현할 수 있다. 이를 로직으로 구성할 때, 두 개의 피연산자를 입력 받아 하나의 출력을 내는 조합회로로 만들 수 있지만, 이와 같은 구현은 조합회로 내의 지연 시간을 길어지게 하기 때문에 동기 회로에 적용할 경우 전체 시스템의 동작 주파수를 낮추는 원인이 된다.
본 구현에서는 그림 7과 같이, B를 일정 길이로 나누고, A와 곱 B의 일부 비트를 곱하는 회로를 여러 사이클에 걸쳐 반복적으로 사용하는 방법을 사용하였다. 곱셈 계산을 위해 여러 사이클이 필요하지만, 회로를 재사용 할 수 있어 칩의 면적을 줄일 수 있다. 곱셈 단위인 디지트(digit)의 크기가 증가할수록 곱셈을 위한 회로는 커지게 되며, 필요한 사이클은 줄어들게 된다. 곱셈을 위한 조합회로가 커지게 되면, 그 만큼 임계 경로(critical path)가 길어지게 되며, 이는 동작 주파수를 낮아지게 하는 원인이 된다. 따라서 디지트를 몇 비트로 하느냐에 따라 곱셈기의 크기와 계산에 필요한 사이클 간 상충 관계(trade-off) 및 최대 동작 주파수와 계산에 필요한 사이클 간에 상충 관계가 발생한다. 본 논문에서는 30 비트, 60 비트, 120 비트로 디지트를 사용하는 곱셈기를 사용하였다.

4.3 제곱 및 16승 연산

제곱연산의 경우 알고리즘 8과 같이 입력 비트 사이에 0을 추가하고, 이를 다시 기약 다항식 $p(z)$를 이용해 축약시킨다. 이 과정을 미리 계산하면 식 (4)과 같이 입력의 각 비트에 따라 입력 비트 중 두 개의 비트를 XOR 연산을 하거나 혹은 단순히 입력된 비트의 위치를 바꾸는 방식으로 구현된다.

$$B_0 = A_0 \veebar A_{221},\ B_1 = A_{120},\ B_2 = A_0 \veebar A_{222} \cdots$$
$$B_{237} = A_{220} \veebar A_{238},\ B_{238} = A_{119} \quad (4)$$

V장에서 설명할 Final Exponentiation의 2-(2) 단계에서 $A^{2^{120}}$ 연산은 식 (5)와 같이 제곱 연산을 120번 수행하게 된다.



〔그림 7〕 곱셈 연산 2

알고리즘 8 $F_{2^{239}}$상의 제곱 계산
입력: 이진 다항식 $a(z) = \sum_{i=0}^{238} a_i z^i = (a_{238}, a_{237}, \ldots, a_0)$.
출력: $c(z) = a(z)^2$
1: $b \leftarrow (a_{238}, 0, a_{237}, 0, \ldots, a_1, 0, a_0, 0) \in F_2[z]$
2: $c \leftarrow b \bmod f(z)$
3: c를 return

$$A^{2^{120}} = (\ldots(\underbrace{((A^2)^2)^2)^2\ldots)^2}_{120} \quad (5)$$

이 때, $A^{16} = (((A^2)^2)^2)^2$을 한 번에 계산하도록 하면, 네 번의 제곱 계산을 한 번에 하는 셈이므로, 식 (6)과 같이 연산 횟수를 30번으로 줄일 수 있다.

$$A^{2^{120}} = (\ldots(\underbrace{((A^{16})^{16})^{16})\ldots)^{16}}_{30} \quad (6)$$

16승 연산 모듈을 만드는 방법 역시 16승 연산 후 축약시킨 결과를 식으로 나타내면 식(4)와 비슷한 형태로 XOR 연산과 비트 위치 변경 로직으로 구성된다. 다만 XOR 연산이 연속적으로 일어나기도 하기 때문에, 조합회로의 단계(depth)가 늘어나지만, 결

과적으로 $A^{2^{120}}$연산에 필요한 사이클을 120 사이클에서 30사이클로 줄여준다.

4.4 제곱근 연산

제곱근 연산을 위해 K. Fong 등[24]의 F_{2^m}상의 제곱근 연산 방법을 참조하였다. $a(z)=\sum_{i=0}^{m-1} a_i z^i \in F_{2^m}$라 하였을 때, $\sqrt{a(z)}$를 다음 제시된 식(7)과 같이 계산할 수 있다.

$$\begin{aligned} a^{\frac{1}{2}} &= (a^{2^m})^{\frac{1}{2}} = a^{2^{m-1}} = (\sum_{i=0}^{m-1} a_i z^i)^{2^{m-1}} = \sum_{i=0}^{m-1} a_i (z^{2^{m-1}})^i \\ &= \sum_{i=0}^{\frac{m-1}{2}} a_{2i} (z^{2^{m-1}})^{2i} + \sum_{i=0}^{\frac{m-3}{2}} a_{2i+1} (z^{2^{m-1}})^{2i+1} \\ &= \sum_{i=0}^{\frac{m-1}{2}} a_2 z^i + \sqrt{z} \sum_{i=0}^{\frac{m-3}{2}} a_{2i+1} z^i \end{aligned} \quad (7)$$

위의 식을 이용하여 다음과 같은 알고리즘 9를 도출할 수 있다.

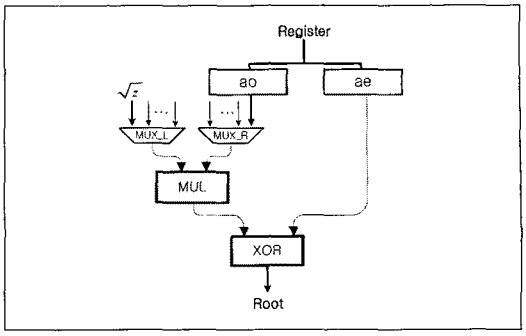
위의 알고리즘 9에서 우리는 기약 다항식 $p(z)=z^{239}+z^{36}+1$을 사용하므로 다음과 같은 방법을 통해 $\sqrt{z}$를 미리 계산하였다.

$$\begin{aligned} & z^{239} = z^{36}+1 \pmod{p(z)} \\ & z = z^{-(238)}(z^{36}+1) \pmod{p(z)} \\ & \sqrt{z} = z^{-119}(z^{18}+1) \pmod{p(z)} \\ & z^{-119} = z^{-83} + z^{120}, \qquad z^{-83} = z^{-47} + z^{156}, \\ & z^{-47} = z^{-11} + z^{192}, \quad z^{-11} = z^{25} + z^{228} \\ & \sqrt{z} = (z^{228}+z^{192}+z^{156}+z^{120}+z^{25}) \cdot (z^{18}+1) \\ & \sqrt{z} = z^{228}+z^{210}+z^{192}+z^{174}+z^{156} \\ & \qquad\qquad + z^{138}+z^{120}+z^{25}+z^{7} \end{aligned} \quad (8)$$

따라서 a_e와 a_o를 만들기 위해 비트 순서를 다시 배열하는 회로를 만들었으며, 이 외에 $\sqrt{z} \cdot a_o$ 곱셈 연산 후 마지막으로 a_e와 $\sqrt{z} \cdot a_o$를 더하기 위한 덧셈 연산을 만들었다. 곱셈 연산을 위해 2절에서 살펴본 곱셈 연산 모듈을 사용하며, 실제 페어링 회로 구현에서는 제곱근 연산 모듈 내부에 곱셈 연산 모듈을 포함하지 않고, 페어링 연산 중 사용되는 곱셈 모듈의 입력을 제곱근 연산에 필요한 $\sqrt{z}$와 a_o로 바꾸어, 그림 8과 같이 재사용 한다.

알고리즘 9 $F_{2^{239}}$상의 제곱근 계산

입력: 이진 다항식 $a(z)=\sum_{i=0}^{238} a_i z^i = (a_{238}, a_{237}, \ldots, a_0)$.
출력: $c(z)=\sqrt{a(z)}$
1: $\sqrt{z}=z^{2^{238}}$을 먼저 계산
2: $a_e \leftarrow (0,0,\ldots,0,a_{238},a_{236},\ldots,a_2,a_0)$
3: $a_o \leftarrow (0,0,\ldots,0,0,a_{237},\ldots,a_3,a_1)$
4: $c \leftarrow a_e + \sqrt{z} \cdot a_o$
5: c를 return



[그림 8] 제곱근 연산 모듈의 곱셈 연산 모듈 재사용

4.5 역승 연산

역승 연산은 Hankerson 등[25]의 F_{2^m}상의 역승 연산 알고리즘을 참고하였다. 그들의 논문에서 EEA (Extended Euclidean Algorithm)와 AIA(Almost Inverse Algorithm), 그리고 MAIA(Modified AIA)에 대해 언급하고 있다. 이 중 우리는 MAIA 알고리즘을 변형하여 구현하였다.

위의 알고리즘 10을 하드웨어로 구현할 때 시간적으로 더 좋은 성능을 낼 수 있도록 step 2에서 1 비트 씩 처리하지 않고, 4 비트 씩 처리가 가능하도록

알고리즘 10 $F_{2^{239}}$상의 Modified AIA 알고리즘

입력: $a \in F_{2^{239}}$, $a \neq 0$.
출력: $a^{-1} \bmod f(x)$.
1: $b \leftarrow 1$, $c \leftarrow 0$, $u \leftarrow a$, $v \leftarrow f$.
2: While x divides u do:
  2.1 $u \leftarrow u/x$.
  2.2 If x divides b then $b \leftarrow b/x$:
    else $b \leftarrow (b+f)/x$.
3: If $u=1$ then return(b).
4: If $\deg(u) < \deg(v)$ then: $u \leftrightarrow v, b \leftrightarrow c$.
5: $u \leftarrow u+v, b \leftarrow b+c$.
6: Goto step 2.

변경하였으며, step 4와 step 5를 동시에 처리하도록 구현하였다. 또한 step 5는 shift 로직으로 구현할 수 있기 때문에, step 5와 step 2가 동시에 이루어지도록 하였다. 또한, step 2부터 step 5까지의 동작이 한 클록에 수행되도록 알고리즘을 변형하였다.

V. 구현 결과 및 비교

이번 절에서는 η_T 페어링 연산을 FPGA로 구현하였을 때의 성능을 제시하고, 기존 연구 결과와 비교 분석할 것이다.

5.1 구현 결과

η_T 페어링 알고리즘을 FPGA로 구현한 후 합성되었을 때의 면적과 최대 동작 주파수를 살펴보았다. 또한 실행에 걸리는 클록 사이클 수를 시뮬레이션을 통해 얻은 후, 실행에 걸리는 시간을 계산 및 예측해 보았다. 결과를 얻기 위해 Xilinx ISE 10.1 tool suite을 이용하였으며, 다른 결과와의 비교를 위해 Xilinx Vertex-II 6000 및 Virtex-II Pro 100을 Target으로 하여 합성을 진행하였으며, speed-grade는 -6을 사용하였다.

〔표 2〕 Virtex-II Pro 100을 이용한 합성 결과

D	Slice	#cycle	MHz	Latency (us)	AT Product
30	22173	6498	140.032	46.40	1.02
60	28218	4058	133.63	30.36	0.85
120	41116	2838	128.923	22.00	0.90

〔표 3〕 Virtex-II 6000을 이용한 합성 결과

D	Slice	#cycle	MHz	Latency (us)	AT Product
30	22185	6498	135.825	47.83	1.06
60	28793	4058	127.555	31.80	0.91

〔표 4〕 다른 구현들과의 비교 분석

Author	Pairing	Curve	Field	Level of Security	Device	D	Slice	#cycle	MHz	Latency (us)	AT Product
Shu[18]	Tate	EC	$F_{2^{239}}$	956	XC2VP100	16	18202	5500	100	55	1.0011
Shu[18]	Tate	EC	$F_{2^{239}}$	956	XC2VP100	32	31719	3569	83	43	1.3639
Shu[18]	Tate	EC	$F_{2^{283}}$	1132	XC2VP100	16	22706	7308	84	87	1.9754
Shu[18]	Tate	EC	$F_{2^{283}}$	1132	XC2VP100	32	37803	4392	72	61	2.3060
Shu[18]	Tate	EC	$F_{2^{239}}$	956	XC2VP100	16	25287	3444	84	41	1.0368
Shu[18]	Tate	EC	$F_{2^{283}}$	1132	XC2VP100	16	33252	4368	56	78	2.5937
Ronan[20]	Tate	HEC	$F_{2^{103}}$	1236	XC2VP100	4	21021	10506	51	206	4.3303
Ronan[20]	Tate	HEC	$F_{2^{103}}$	1236	XC2VP100	8	24290	6992	46	152	3.6921
Ronan[20]	Tate	HEC	$F_{2^{103}}$	1236	XC2VP100	12	27182	5805	43	135	3.6696
Ronan[20]	Tate	HEC	$F_{2^{103}}$	1236	XC2VP100	16	30464	5412	41	132	4.0212
Ronan[21]	η_T	EC	$F_{2^{313}}$	1252	XC2VP100	4	34675	11165	55	203	7.0390
Ronan[21]	η_T	EC	$F_{2^{313}}$	1252	XC2VP100	8	41078	6200	50	124	5.0937
Ronan[21]	η_T	EC	$F_{2^{313}}$	1252	XC2VP100	12	44060	4818	33	146	6.4328
Keller[19]	Tate	EC	$F_{2^{251}}$	1004	XC2V6000	1	16621	322000	50	6440	107.0392
Keller[19]	Tate	EC	$F_{2^{251}}$	1004	XC2V6000	6	21955	110940	43	2580	56.6439
Keller[19]	Tate	EC	$F_{2^{251}}$	1004	XC2V6000	10	27725	94800	40	2370	65.7083
Keller[19]	Tate	EC	$F_{2^{283}}$	1132	XC2V6000	1	18599	399000	50	7980	148.4200
Keller[19]	Tate	EC	$F_{2^{283}}$	1132	XC2V6000	4	22636	158270	49	3230	73.1143
Keller[19]	Tate	EC	$F_{2^{283}}$	1132	XC2V6000	6	24655	132070	47	2810	69.2806
This Work	η_T	EC	$F_{2^{239}}$	956	XC2VP100	60	28218	4058	133.63	30.36	0.85
This Work	η_T	EC	$F_{2^{239}}$	956	XC2V6000	60	28793	4058	127.555	31.80	0.91

곱셈 연산 모듈의 경우 각각 30 비트, 60 비트, 120 비트 크기의 디지트를 사용하는 곱셈 연산을 사용하였으며, 역승 연산 모듈의 경우 4 비트 단위 연산을 하는 것을 사용하였다. Virtex-II Pro 100을 Target으로 한 합성 결과를 표 2에, Virtex-II 6000을 Target으로 한 합성 결과를 표 3에 각각 나타내었다.

5.2 결과 비교

기존 연구와의 구현 결과 비교를 표4에 나타내었다. 비슷한 안전성을 제공하는 Shu 등[18]의 구현, Ronan 등[20,21]의 구현, 그리고 Keller 등[19]의 구현에 비해 칩 면적이나 시간적인 측면에서 우수한 성능을 보이고 있음을 확인할 수 있다. 특히, 같은 크기의 필드 상에서 구현된 결과 중 가장 성능이 좋은 Shu 등의 구현에 비해 AT(Area·Time) Product가 15% 가량 향상되었음을 확인할 수 있다.

VI. 결 론

본 논문에서는 η_T 페어링 알고리즘을 하드웨어에 최적화할 수 있도록 변형하여 FPGA를 이용해 하드웨어로 구현해 보았다. 우리가 구현한 η_T 페어링 모듈의 특징은 수행 속도를 빠르게 하기 위해서 많은 연산 모듈을 병렬적으로 배치하면서도 사용하는 레지스터의 수는 최대한 줄일 수 있도록 최적화를 한 것이다. 그래서 칩 면적의 최적화를 구현함과 동시에 시간 성능 또한 좋은 결과를 얻었다. 하지만 아직까지 더 연구해 볼 부분이 많이 존재한다. 먼저 곱셈 연산에서는 다항식끼리의 곱에서 한 번에 곱하는 디지트의 수에 따라 성능이 달라 질 것이다. 또 많은 수행시간이 소요되는 역승 연산 모듈은 한 번에 처리하는 비트수를 얼마로 정하느냐에 따라 성능이 달라질 것이다. 또한 두 개 보다 많은 수의 곱셈기를 사용하였을 때는 또 다른 성능을 나타낼 것이다. 이처럼 아직 구현의 설계부분에 있어서 성능의 최적화를 위해 실험 할 부분도 많고 성능에 대해 비교, 분석할 부분도 많다. 이 부분에 대한 연구는 향후 지속적으로 연구해 나갈 과제이다.

참 고 문 헌

[1] A. Menezes, T. Okamoto, and S. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field," IEEE Trans. Inform. Theory, vol. 39, no. 5, pp. 1639-1646, Sep. 1993.

[2] G. Frey and H.G. Rück, "A remark concening m-divisibility and the discrete logarithm in the divisor class group of curves," Math. Comput., vol. 62, no. 206, pp. 865-874, Apr. 1994.

[3] D. Boneh and M. Franklin, "Identity based encryption from the weil pairing," SIAM J. on Computing, vol. 32, no. 3, pp. 586-615, Mar. 2003.

[4] R. Sakai and M. Kasahara, "ID based cryptosystems with pairing on elliptic curve," IACR ePrint 2003-054, Mar. 2003.

[5] J.C. Cha and J.H. Cheon, "An Indentity-Based Signature from Gap Diffie-Hellman Groups," PKC 2003, LNCS 2567, pp. 18-30, 2003.

[6] F. Hess, "Exponent group signature schemes and efficient identity based signature schems based on pairing," SAC 2002, LNCS 2595, pp. 310-324, 2002.

[7] K.G. Paterson, "ID-based signature from pairings on elliptic curves," Electronics Letters, vol. 38, no. 18, pp. 1025-1026, Aug. 2002.

[8] A. Joux, "A One Round Protocol for Tripartite Diffie-Hellman," Journal of Cryptology, vol. 17, no. 4, pp. 263-276, Sep. 2004.

[9] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," Journal of Cryptology, vol. 17, no. 4, pp. 297-319, Sep. 2004.

[10] N.P. Smart, "An identity based authentication key agreement protocol based on pairing," Electronics Letters, vol. 38, no. 13, pp. 630-632, June 2002.

[11] P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott, "Efficient algorithms for pairing-based cryptosystems," CRYPTO 2002, LNCS 2442, pp. 354-368, 2002.

[12] S.D. Galbraith, K. Harrison, and D. Soldera, "Implementing the Tate pairing," ANTS V, LNCS 2369, pp. 324-337, 2002.

[13] V. Miller, "Short Programs for Functions on Curves," unpublished manuscript, 1986.

[14] I. Duursma and H.S. Lee, "Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$," Asiacrypt 2003, LNCS 2894, pp. 111-123, 2003.

[15] S. Kwon, "Efficient Tate Pairing Computation for Elliptic Curves over Binary Fields," ACISP 2005, LNCS 3574, pp. 134-145, 2005.

[16] P.S.L.M. Barreto, S. Galbraith, C. Ó hÉigeartaigh, and M. Scott, "Efficient Pairing Computation on Supersingular Abelian Varieties," IACR ePrint 2004-375, Sep. 2005.

[17] F. Hess, N. Smart, and F. Vercauteren, "The eta pairing revisited," IEEE Transactions on Information Theory, vol. 52, no. 10, pp. 4595-4602, Oct. 2006.

[18] C. Shu, S. Kwon, and K. Gaj, "FPGA accelerated Tate pairing based cryptosystem over binary fields," Proceedings of the 2006 IEEE International Conference on Field Programmable Technology, pp. 173-180, Dec. 2006.

[19] M. Keller, T. Kerins, F. Crowe, and W.P. Marnane, "FPGA implementation of a $GF(2^m)$ Tate pairing architecture," In K. Bertels, J.M.P. Cardoso, and S. Vassiliadis, editors, International Workshop on Applied Reconfigurable Computing (ARC 2006), number 3985 in Lecture Notes in Computer Science, pp. 358 - 369, Mar. 2006.

[20] R. Ronan, C. Murphy, T. Kerins, C. Ó hÉigeartaigh, and P.S.L.M. Barreto, "A flexible processor for the characteristic 3 η_T pairing," International Journal of High Performance Systems Architecture, vol. 1, no. 2, pp. 79 - 88, Oct. 2007.

[21] R. Ronan, C. Ó hÉigeartaigh, C. Murphy, M. Scott, and T. Kerins, "FPGA acceleration of the Tate pairing in characteristic 2," Proceedings of the 2006 IEEE International Conference on Field Programmable Technology, pp. 213 - 220, Dec. 2006.

[22] J. Jiang, "Bilinear pairing (Eta T Pairing) IP core," City University of Hong Kong - Department of Computer Science, May 2007.

[23] D.H. Choi, D.G. Han, and H.W. Kim, "Construction of Efficient and Secure Pairing Algorithm and Its Application," Journal of Communications and Networks, vol. 10, no. 5, pp. 437-443, Dec. 2008.

[24] K. Fong, D. Hankerson, J. López, and A. Menezes, "Field inversion and point halving revisited," CORR 2003-18, University of Waterloo, 2002.

[25] D. Hankerson, J.L. Hernandez, and A. Menezes, "Software Implementation of Elliptic Curve Cryptography Over Binary Fields," CHES 2000, LNCS 1965, pp. 1-24, 2000.

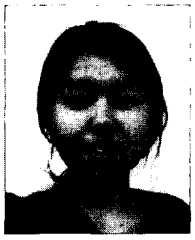
〈著者紹介〉

이 동 건 (Dong-geon Lee) 학생회원
2009년 2월: 부산대학교 정보컴퓨터공학부 졸업
2009년 3월~현재: 부산대학교 컴퓨터공학과 석사과정
〈관심분야〉 RFID/USN 정보보호 기술, 페어링 기반 암호 이론, VLSI 설계, embedded system

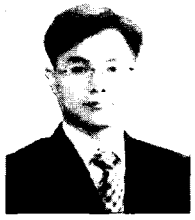
이 철 희 (Chul-hee Lee) 학생회원
2009년 2월: 동명대학교 멀티미디어공학과 졸업
2009년 3월~현재: 부산대학교 컴퓨터공학과 석사과정
〈관심분야〉 부채널 공격 기법 및 대응, 페어링 기반 암호 이론, VLSI 설계

최 두 호 (Dooho Choi) 정회원
1994년: 성균관대학교 수학과 졸업(학사)
1996년: 한국과학기술원(KAIST) 수학과 석사(이학석사)
2002년: 한국과학기술원(KAIST) 수학과 박사(이학박사)
2002년 1월~현재: 한국전자통신연구원(ETRI) 정보보호연구본부 선임연구원/팀장
〈관심분야〉 페어링 기반 암호 이론, 암호시스템 안전성 증명, 비가환군 암호 이론

김 철 수 (Chulsu Kim)
2001년 2월: 한밭대학교 전자계산학과 졸업(학사)
2003년 2월: 충남대학교 컴퓨터과학과 석사(이학석사)
2002년 12월~2004년 5월: 메타빌드(주) 기업부설연구소 주임연구원
2004년 5월~2009년 4월: 한국전자통신연구원(ETRI) RFID/USN 연구본부 연구원
2009년 4월~현재: 창신정보통신(주) 기업부설연구소 연구소장
〈관심분야〉 USN 미들웨어, embedded system



최 은 영 (Eun Young Choi)
2001년 8월: 고려대학교 수학과 졸업(학사)
2003년 8월: 고려대학교 정보보호대학원 공학석사
2004년 3월~2009년 8월: 고려대학교 정보경영공학전문대학원 공학박사
2007년 10월~현재: 한국인터넷진흥원 연구원
〈관심분야〉 암호 이론, 정보보호 이론, RFID 정보보호 기술, 유비쿼터스



김 호 원 (Ho Won Kim) 종신회원
1993년 2월: 경북대학교 전자공학과졸업(학사)
1995년 2월: 포항공과대학교 전자전기공학과 석사(공학석사)
1999년 2월: 포항공과대학교 전자전기공학과박사(공학박사)
1998년 12월~2008년 2월: 한국전자통신연구원(ETRI) 정보보호연구단 선임연구원/팀장
2008년 3월~현재: 부산대학교 정보컴퓨터공학부 조교수
〈관심분야〉 RFID/USN 정보보호 기술, 타원곡선 및 초타원곡선 암호 이론, VLSI 설계, embedded system