# Bilinear map 기반 센서네트워크 보안프로토콜을 위한 Pairing용 곱셈 최적화 기법[1)]

서화정*, 이동건*, 김호원*
부산대학교 컴퓨터 공학과*
hwajeong@pusan.ac.kr, guneez@gmail.com, howonkim@pusan.ac.kr

# Optimization of multiplication-techniques for a Pairing for sensor network security protocol based on bilinear map

Hwa-Jeong Seo*, Dong-Geon Lee*, Ho-won Kim*
Department of Computer Science Engineering Pusan National University*

**요 약**

센서네트워크는 유비쿼터스 환경을 실현할 수 있는 기술로서, 최근 무인 경비 시스템이나 에너지 관리, 환경 모니터링, 홈 자동화, 헬스케어 응용 등과 같은 다양한 응용 분야에 활용되고 있다. 하지만 자신의 정보가 무선통신상에 쉽게 노출됨으로써 도청과 전송 메시지에 대한 위변조, 서비스 거부 공격을 받을 위험이 있다. 더욱이 센서네트워크의 자원 제약성(적은 메모리, 컴퓨팅 성능의 제약)과 키분배·관리의 어려움으로 인해 기존의 공개키, 대칭키 기반의 보안프로토콜을 대체할 수 있는 프로토콜이 필요하다. 그러므로 키분배·관리에 장점을 가지는 Bilinear map 기반 프로토콜은 적합한 대안이다. 하지만 프로토콜에 사용되는 Pairing연산은 높은 컴퓨팅 성능이 요구된다. 따라서 제한된 성능을 가진 센서상의 구현을 위해서는 Computation Cost를 줄이고 연산 수행 속도를 가속화 할 필요성이 있다. 본 논문에서는 프로토콜 구현에 필요한 Pairing의 핵심 연산인 Multiplication을 대표적인 센서노드 프로세서인 MSP430상에서 최적화 구현함으로써 성능을 개선한다.

## 1. 서 론

센서네트워크를 통한 유비쿼터스 환경의 실현은 기술의 발전으로 인해 점차 가속화되어 가고 있다. 센서들간의 통신에 사용되는 사물통신(M2M platform)은 사물과 사물, 사물과 사람간의 통신을 일컬으며 방송통신융합 ICT(Information and Communication Technologies)와 결합해 원격지에 있는 사물의 정보를 수집·활용이 가능케 하고 있다.[1] 해당 기술은 4대강 유역의 수질 측정 관리와 전력 생산 및 소비 정보를 양방향·실시간으로 유통함으로써 에너지 효율을 극대화 할 수 있는 스마트 그리드 기술에 적용이 가능하다.[2] 이처럼 통신상에 유통되어 지는 정보는 자원적인 제약성을 가진 센서를 통하여 전달되지만 정보의 중요성을 생각해 볼 때 사용자의 정보가 목적지에 안전하게 전달될 수 있도록 기밀성을 보장해 주어야 한다. 하지만 무선 통신의 특성상 자신의 정보가 공간상에 무방비로 노출될 뿐 아니라 현재 센서 상에는 완전한 보안장치를 제공하고 있지 않다. 따라서 ECC를 통한 타원곡선 암호 및 RSA 암호 등 다양한 PKC(Public Key Cryptography)방식이 연구되고 있다. 하지만 센서의 메모리 부족과 컴퓨팅 성능의 제한으로 원활히 연산이 수행되지 못하며 상대방의 모든 키를 분배받고 관리하는데 문제가 있다. 이러한 키문제는 자신이 가진 ID값을 통해 암호화가 수행되어 서로간에 키분배가 필요하지 않은 IBE(Identity Based Encryption)프로토콜을 통해 해결가능하다. 하지만 IBE수행에 요구되는 높은 컴퓨팅 성능에 대한 대안이 필요하다. 본 논문에서는 Pairing을 구성하는 핵심 연산인 Multiplication을 대표적인 센서노드 프로세서인 MSP430 상에 최적화 구현하여 Pairing이 사용되는 IBE프로토콜을 개선한다. 논문의 구성은 2장에서 Pairing Identity based Protocol에 대해 알아보며, 3장에서 최적화 곱셈기법을 구현한 프로세서인 MSP430을 다른 프로세서인 Atmega128과 비교하여 장단점을 분석한다. 4장에서는 최적화 곱셈기법을 제안하여 개선된 기법을 소개하며 마지막 5장에서는 전체적인 성능을 다른 기법과 비교분석을 통해 제안기법의 우수성을 증명한다.

1) 이 논문 또는 저서는 2010년 교육과학기술부로부터 지원받아 수행된 연구임 (지역거점연구단육성사업/차세대물류IT기술연구사업단)

## 2. Pairing Identity based Protocol

### 2.1 Identity Based Cryptography의 특성

표 1 IBE구현을 위한 Pairing의 특성 요약

| |
|---|
| $G_1$은 *cyclic additional group*이다., |
| $G_2$은 *a cyclic multiplicative group*이다. |
| $G_1$, $G_2$는 동일한 *prime* order $q$상에 정의된다. |
| $\hat{e}$는 $G_1 \times G_1 \rightarrow G_2$로 정의한다. |
| $Bilinearity: P, Q \in G_1$와 $a,b \in Z$상에서 $\hat{e}(aP, bQ)$ $=\hat{e}(P, bQ)^a = \hat{e}(aP, Q)^b = \hat{e}(P, Q)^{ab}$이 성립한다. |
| $Non-degeneracy: P, Q \in G_1$는 $\hat{e}(P,Q) \neq 1$을 만족한다. |
| $Computability: P,Q \in G_1$상에서 $\hat{e}(P,Q)$를 효율적으로 연산하기 위한 알고리즘이 있다. |

1984년, Shamir에 의해 제안된 IBE는 Pairing의 Bilinearity, Non degeneracy, Computability 특성에 의해 구현되며 자신이 가진 고유의 정보(IP, email address, serial number, hw-address)를 통해 보안을 제공한다. [4][5] 이중에서 Bilinearity는 pairing의 승수연산을 스칼라 곱셈연산으로 만들어 준다. 이를 통해 PBC상에서 상대방과 정보 교환시 자신의 정보 제공없이 서로간에 정보를 주고 받을 수 있다. Non-degeneracy는

Pairing연산이 수행되고 있는 Group을 벗어나지 않도록 한다. 마지막으로 Efficient-Computability은 Pairing연산상에서의 스칼라곱셈연산이 Doubling과 Add를 통해 효율적으로 계산가능하게 한다.

### 2.2 Eta-T Pairing을 통한 Pairing Based Cryptography

현재 IBE구현에 사용되는 bilinear 함수는 크게 4가지로 나누어 볼 수 있다. 그 종류에는 초창기에 제안된 Wëil, Tate pairing과 2004년, Barreto에 의해 제안된 Eta-t pairing 그리고 2006년, Hess에 의해 제안된 Ate pairing이 있다.[5] PBC(Pairing Based Cryptography)의 장점은 센서 상호간에 인증 과정을 통하지 않고 비밀 키를 생성할 수 있으며, [6][7] 자신의 고유정보를 통한 암호화가 가능하므로 초기에 PKI에서 필요했던 불필요한 Public key분배 과정이 생략된다. 또한 인증서 생성 시 기존의 320-bit DSA에서 제공해 주던 signature를 160-bit PBC에서 Short-signature로 구현이 가능하다.[8][9] 하지만 센서 상에서의 암호화 구현은 프로토콜의 암호학적 장점과 함께 Pairing연산 수행에 사용되는 Computational Cost에 대한 고려도 함께 병행되어야 한다. 이에 대한 분석을 위해 여러 종류의 Pairing 연산중 센서 상에서 가장 최적화된 성능을 보이는 eta-t Pairing을 기준으로 분석해보면 [표 2]와 같다. [표 2]와 같이 Pairing연산은 크게 squaring, addition, reduction, inversion, multiplication으로 구성된다. 이 중에서 multiplication연산이 Pairing전체연산에서 차지하는 비중은 75%에 달한다. 이는 전체 성능이 multiplication에 의해 좌우된다는 것을 의미하므로 성능개선을 위해 multiplication에 대한 이루어 져야한다.

표 2 eta-t pairing을 구성하는 연산의 부하 분석[3]

| $Eta-T\ Pairing\ over\ GF(2^{239})$ | |
|---|---|
| 연산 | 연산의 부하비중(%) |
| multiplication | 75 |
| squaring | 4.7 |
| addition | 2.3 |
| reduction | 6.3 |
| inversion | 4.5 |

## 3. Target Platform

### 3.1 MSP430과 Atemga128 특성

현재 대표적인 센서 프로세서에는 MSP430과 Atmega128이 있다. 두 프로세서는 저전력, 고성능으로 설계되어 센서네트워크의 구성에 적합하며 memory와 Rom 그리고 CPU의 성능이 비슷하다. 두 프로세서가 가지는 차이점은 register의 구성 bit와 개수, 그리고 Assemly coding에 사용하는 Instruction set의 구성이다. 표 3을 보면 MSP430은 Atmega128에 비해 general register의 수가 16개 적다. 하지만 register로 표현 가능한 bit는 16bit이므로 Atmega128의 8bit에 비해 크다. 이러한 특성은 register에 동일한 크기의 정보를 불러오고, 저장하는 경우, Atmega128에 비해 적은 memory access로도 동일한 연산수행이 가능하다는 것을 의미한다. MSP430과 Atmega128에서 사용되어지는 Instruction set은 전체적인 기본 연산은 비슷하다. 하지만 동일한 명령을 수행할 때 사용되어 지는 register의 개수와 operation 수행결과에 있어서 상이하다.

표 3 MSP430, Atmega128의 성능비교

| | clock frequency | register bit<br>register 범위<br>register 개수 | program ROM<br>flash memory |
|---|---|---|---|
| MSP430 | 16 Mhz | 16 bit<br>/ r0 - r15<br>/ 16 | 120KB<br>/ 60KB |
| Atmega128 | 16 Mhz | 8 bit<br>/ r0 - r31<br>/ 32 | 64KB<br>/64KB |

표 4 MSP430 Instruction set

| | 명령어 | 설명 |
|---|---|---|
| 1 | BIT Rd, Rr | Source register의 bit를 Destination register와 비교하여 동일한 위치에 1인 bit가 존재할 경우 Carry가 발생한다. |
| 2 | JNC Ad | Carry bit이 0인 경우 목적지주소로 PC Counter를 이동시킨다. |
| 3 | ADC Rd | Destination Register의 값과 Carry Register의 값을 더하여 Destination Register에 저장한다. |

표 5 Atmega128 Instruction set

| | 명령어 | 설명 |
|---|---|---|
| 1 | SBRS Rr, b | Source register의 b번째 Register Bit이 1인 경우 $PC \leftarrow PC+2$이 수행되어 다음에 오는 명령이 수행되지 않는다.. 0인 경우 $PC \leftarrow PC+1$이 수행되어 순차적으로 다음에 오는 명령을 수행한다. |
| 2 | RJMP Ad | RJMP는 목적지주소로 PC Counter를 이동시킨다. |
| 2 | ADC Rd, Rr | Source register의 값과 Destination register의 값 그리고 Carry register의 값을 모두 합하여 Destination register에 저장한다. |

표 6 사용되어진 파라미터 정리

| | |
|---|---|
| Rr | Register Source 주소 |
| Rd | Register Destination 주소 |
| Ad | 코드 상의 Destination 주소 |
| b | Register의 bit 위치 |

[표 4], [표 5]의 첫 번째와 두번째 명령어는 분기문을 작성하기 위해 사용되는 instruction set이다. MSP430의 경우 BIT를 통해 Source register의 bit를 Destination register의 bit와 비교해서 동일한 위치에 1이 있는 경우 Carry bit를 발생시킨다. JNC는 Carry bit가 0인 경우 분기문을 수행한다. Atmega128의 Source register의 bit를 확인하여 1인 경우 다음에 오는 명령을 수행하지 않고 0인 경우 명령을 수행한다. 따라서 SBRS 뒤에 RJMP가 있는 경우 SBRS의 결과값에 따라 분기가 일어난다. 또한 연산에 수행되는 clock cycle은 MSP430의 경우 2clock이 사용되지만 Atmega128 의 SBRS의 경우 $PC \leftarrow PC+1$인 경우 1clock, $PC \leftarrow PC+2$인 경우 2clock이 소모된다. 세 번째 명령어는 전에 계산되어진 carry값을 이용하여 덧셈연산을 수행한다. Atmega128은 source와 destination register가 모두 필요하지만 MSP430의 경우 destination register만을 사용하여 구현이 가능하다. 결과적으로 MSP430의 경우 register를 적게 사용하고도 동일한 명령을 수행할 수 있다.

### 3.2 구현환경

[그림 1]은 Atmega128, [그림 2]는 MSP430 프로세서를 탑재한 실험 보드이다. MSP430 프로세서는 USB to Serial port를 통해 컴퓨터와 연결하여 프로그램을 Install하였다. Atmega128의 경우에는 JTAG를 통한 컴퓨터와의 Serial 통신을 통해 프로그램을 Install한다. 구현환경은 [그림 3]과 같이 Vitual Machine상에 xubuntu 이미지를 생성한 후 Tinyos-2.x, nesc-1.2.8a-1 그리고 gcc 3.2.3을 설치하였다. 구현한 제안 기법은 nesc문법으로 작성하여 nesc컴파일로 컴파일하였고 결과값은 Printf문을 통해 Command 창에 출력하여 확인하였다. [그림 4]는 구현이 끝난 code의 성능 검사에 사용된 프로그램의 화면이다. windows xp 상에 IAR Simulator를 설치한 후 nesc상에서 구현된 assembly code를 넣고 clock cycle을 simulation하였다. 검증과정은 여러 차례에 걸쳐 다양한 입력값을 넣어준 후 clock cycle을 확인하는 방식으로 진행되었다.

그림 1 Sirase Algorithm을 구현한 Atmega128보드

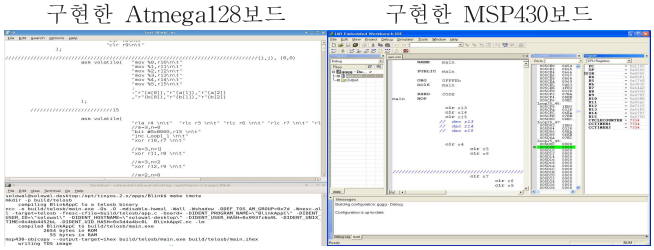

그림 3 Assembly code를 작성하여 구현한 환경 (Tinyos-2.x)

그림 2 제안된 Algorithm을 구현한 MSP430보드



그림 4 Assembly code의 Clock cycle을 검증하기 위한 환경 (IAR Embedded Workbench IDE)

## 4. Our implementation

### 4.1 Binary field상의 Eta-T Pairing연산

Supersingular elliptic curve $E: Y^2=X^3+X+b, b\in(0,1)$은 binary field상의 홀수인 extension degree m을 가지는 $GF(2^m)$상에서 정의된다. [표 7]에 정의된 Eta-T pairing의 연산은 $\eta_T : E(GF(2^m))[l]\times E(GF(2^m))[l]\rightarrow GF(2^{4m})^*$로 정의하며 $GF(2^m)$상의 값들을 Eta-T Pairing연산을 통해 $GF(2^{4m})$상으로 mapping시킨다. l은 $l|E(GF(2^m))$을 성립하는 가장 큰 홀수인 동시에 소수이며 $(2^{km}-1)$을 나눌 수 있는 가장 작은 양의 정수 l은 embedding degree라고 한다. 연산의 결과값은 $GF(2^m)$상의 값 $a_0, a_1, a_2, a_3$을 $s^2+s+1=0$과 $t^2+t+s=0$의 특성을 만족하는 $GF(2^{4m})$상의 값 $a_0+a_1s+a_{2t}+a_3st$로 mapping한다. [표 7]에 나타난 바와 같이 Eta-T Pairing연산 중 multiplication연산은 "Algorithm 1"의 Step 6과 Step 8 그리고 Step 12에서 사용된다. 이 중에서도 Step 6과 Step 8은 반복문인 Step 3안에서 동작하므로 m값에 따라 $\frac{m-1}{2}$번 반복수행되므로 성능에 큰 영향을 미친다.

표 7 [Algorithm 1] Computing the $\eta_T$ Pairing[11]

*Input* : $P=(\alpha,\beta), Q=(x,y)\in E(GF(2^m))$
*Output* : $\eta_T(P,Q)\in E(GF(2^m))$
1 : $C\leftarrow 1$
2 : $\alpha\leftarrow\alpha^2+1$, $\beta\leftarrow\beta^2+1$, $u\leftarrow y+b+1$, $v\leftarrow x+1$, $\theta\leftarrow\alpha v$
3 : for $i\leftarrow 0$ *to* $\frac{m-1}{2}$
4 : $\quad A\leftarrow\beta+\theta+u+(\alpha+v+1)s+t$
5 : $\quad C\leftarrow C^2$
6 : $\quad C\leftarrow C\bullet A$
7 : $\quad$ if $i<\frac{m-1}{2}$ *then*
8 : $\quad\quad \alpha\leftarrow\alpha^4$, $\beta\leftarrow\beta^4$, $u\leftarrow u+v+1$, $v\leftarrow v+1$, $\theta\leftarrow\alpha v$
9 : $\quad$ *end* if
10 : *end* for
11 : $A\leftarrow A+(\alpha^2+v+1)+s$
12 : $C\leftarrow C\bullet A$
13 : $C\leftarrow C^M$
14 : *return* $C$

Algorithm 2는 Algorithm 1의 Step 13에 해당하는 final exponentiation이다. m과 b의 값에 따라 수행되는 연산이 달라지며, 계산된 결과는 $GF(2^m)$상의 값을 기저로 하는 $GF(2^{4m})$상으로 확장된다. 본 논문에서는 eta-T Pairing의 m값이 239일 경우를 기준으로 곱셈기법을 제안한다.

표 8 [Algorithm 2] Final exponentiation

| |
|---|
| if $(m \bmod 8, b)=(1,0),(3,1),(5,1),(7,0)$ |
| $(2^{2m}-1)(2^m-2^{(m+1)/2}+1)2^m$ |
| if $(m \bmod 8, b)=(1,1),(3,0),(5,0),(7,1)$ |
| $(2^{2m}-1)(2^m+2^{(m+1)/2}+1)2^m$ |

### 4.2 이전 구현사례

센서 상에서의 연산은 register의 사용효율성을 최대로 하여 memory access를 줄이는 것이 중요하다. 2009년 Shirase에 의해 제안된 Block comb method 곱셈기법은 센서의 register를 효율적으로 사용하여 곱셈동안 memory access를 최소화한다.[3]

표 9 Block multiplication of size 6 [3]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | $A_{24}^6$<br>$B_{24}^6$ | $A_{18}^6$<br>$B_{18}^6$ | $A_{12}^6$<br>$B_{12}^6$ | $A_6^6$<br>$B_6^6$ | $A_0^6$<br>$B_0^6$ |
| | | | | $A_{24}^6$<br>$B_0^6$ | $A_{18}^6$<br>$B_0^6$ | $A_{12}^6$<br>$B_0^6$ | $A_6^6$<br>$B_0^6$ | $A_0^6$<br>$B_0^6$ |
| | | | $A_{24}^6$<br>$B_6^6$ | $A_{18}^6$<br>$B_6^6$ | $A_{12}^6$<br>$B_6^6$ | $A_6^6$<br>$B_6^6$ | $A_0^6$<br>$B_6^6$ | |
| | | $A_{24}^6$<br>$B_{12}^6$ | $A_{18}^6$<br>$B_{12}^6$ | $A_{12}^6$<br>$B_{12}^6$ | $A_6^6$<br>$B_{12}^6$ | $A_0^6$<br>$B_{12}^6$ | | |
| | $A_{24}^6$<br>$B_{18}^6$ | $A_{18}^6$<br>$B_{18}^6$ | $A_{12}^6$<br>$B_{18}^6$ | $A_6^6$<br>$B_{18}^6$ | $A_0^6$<br>$B_{18}^6$ | | | |
| $A_{24}^6$<br>$B_{24}^6$ | $A_{18}^6$<br>$B_{24}^6$ | $A_{12}^6$<br>$B_{24}^6$ | $A_6^6$<br>$B_{24}^6$ | $A_0^6$<br>$B_{24}^6$ | | | | |
| $C_{54:48}^6$ | $C_{48:42}^6$ | $C_{42:36}^6$ | $C_{36:30}^6$ | $C_{30:24}^6$ | $C_{24:18}^6$ | $C_{18:12}^6$ | $C_{12:6}^6$ | $C_{6:0}^6$ |

표 9는 Atmega128상에서 Shirase에 의해 제안된 Block comb method의 수행 모습이다. Block comb method는 register들을 block단위로 묶어서 연산자 A와 피연산자 B를 곱한 뒤 같은 자리의 결과값을 한번에 저장한다. 예를들어 오른쪽에서 두 번째 줄의 $A_6^6\times B_0^6$과 $A_0^6\times B_6^6$의 결과값의 자리는 register 6~11로 동일하기 때문에 계산된 결과는 $C_{12:6}^6$상에 저장된다. 사용된 인자들의 표기법은 밑수는 register의 시작위치를 나타내며 승수자리의 값은 register의 개수를 의미한다. 예를들어 $A_{24}^6$은 6개의 register로 구성되며 범위는 24~29까지이다. 실질적으로 Atmega128의 register는 8bit를 저장하므로 실제로는 192bit부터 240bit까지를 나타낸다. 본 논문에서는 Shirase의 Block comb method를 MSP430에 적용함으로써 성능을 향상시켰다.

### 4.3 MSP430상에서의 Block comb method 곱셈

표 10에 나타난 Algorithm 3은 Shirase에 의해 제안된 Block comb method를 MSP430 환경에 맞게 개선 및 수정한 Algorithm이다. Step 1~8에서는 사용하게 될 register를 초기화시켜준다. Step 10~15에서는 연산자와 피연산자를 register에 load하며 Step 16~25까지는 Block단위의 곱셈이 수행되게 된다. Step 17은 $R_4$에서 $R_9$까지 1bit left shift연산을 수행한다. $R_9$의 16번째 bit는 $R_4$의 1번째 bit로 rotate되어 left shift된다. Step 26에서는 c값을 for문이 돌때마다 증가시키며 register가 rotate 된 이후의 위치 계산에 사용된다. 예를 들어 Step 30의 $R_{(k+c)\%6+4}$에서 c값은 rotate가 수행된 횟수를 기록하

여 register가 rotate된 경우 인자에 대한 %(modular)연산을 통해 rotate된 범위를 일정한 범위로 한정한다. Step 29~34에서는 계산이 끝난 결과값을 memory에 저장한다.

표 10 [Algorithm 3] Block comb method for computing C=AB with block size 3(A,B in $GF(2^{239})$) over MSP430[3]

*Input*: *Binary polynomials* $A=(A_{14},\ldots,A_0), B=(B_{14},\ldots,B_0)$ *(in memory), where* $A,B\in GF(2^{239})$
*Output*: $(C_{29},\ldots,C_0)=AB$*(to memory)*
1: for $i=0$ *to* 2 *do*
2: $R_{i+4}\leftarrow 0$
3: *end* for
4: $c=0$;
5: for $i=0$
6: for $j=0$
7: $R_{(j+c+3)\%6+4}\leftarrow 0$
8: *end* for
9: for $j=0$
10: $k=i-j$;
11: if $0\leq k$ and $k\leq 4$ *then*
12: for $l=0$
13: *load* $R_{10+l}\leftarrow A_{3j+l}$
14: *load* $R_{13+l}\leftarrow B_{3k+l}$
15: *end* for
16: for $l=15$ *downto* 0 *do*
17: *rotate*$[(R_9,..,R_4)\leftarrow(R_9,..,R_4)]$
18: for $m=2$ *downto* 0 *do*
19: for $n=0$ *to* 2 *do*
20: if$(the\ l-th\ bit\ of\ R_{13+m})=1$ *then*
21: $R_{(m+n+c+1)\%6+4}\leftarrow R_{(m+n+c+1)\%6+4}\oplus R_{10+n}$
22: *end*if
23: *end* for
24: *end* for
25: *end* for
26: $c=c+1$;
27: *end*if
28: *end* for
29: for $k=0$ *to* 2 *do*
30: *store* $C_{3i+k}\leftarrow R_{(k+c)\%6+4}$
31: *end* for
32: *end* for
33: for $i=0$ *to* 2 *do*
34: *store* $C_{27+i}\leftarrow R_{(i+c+3)\%6+4}$
35: *end* for

표 11 Algorithm 3에 사용된 명령어

| | |
|---|---|
| ⊕ | exclusive-or연산을 수행한다 |
| load | memory에 있는 내용을 register로 저장한다. |
| store | register에 있는 내용을 memory로 저장한다. |
| move | register의 내용을 다른 register에 저장한다. |
| rotate | register의 내용을 왼쪽으로 rotate shift시킨다. |
| % | 인자에 modular연산을 수행한다. 실제로 assembly 언어로 작성 시에는 미리 계산된 register 위치를 직접 할당해 준다. 따라서 추가적인 clock cost가 발생하지 않는다. |

### 4.4 곱셈구현기법 및 성능향상

기존의 Atmega128에 구현된 Block comb method를 MSP430에 구현하는 것은 불가능하다.

표 12 연산에 사용되어지는 general register 비교

| | 연산자 | 피연산자 | 결과값 | Shift Register | 총 |
|---|---|---|---|---|---|
| Atmega128 | 6 | 6 | 12 | 1 | 25 |
| MSP430 | 3 | 3 | 6 | 1 | 13 |

MSP430에서는 총 13개의 register가 필요하다. 하지만 MSP430에서 제공하는 general register는 12개이다. 이는 rotate shift 기법과 MSP430의 Instruction set의 특성을 이용하면 해결가능하다. Atmega128상에서는 Instruction "Adc"에 사용되는 register가 2개이므로 carry만을 결과값에 더하고 싶을 경우 register가 추가적으로 필요하다. 하지만 MSP430의 "Adc"에 사용되는 register는 1개이므로 추가적인 register가 필요하지 않다.

#### 4.4.1 memory access 최소화 기법

MSP430은 register의 크기가 16bit이므로 8bit 프로세서인 Atmega128에 비해 memory access가 절반으로 줄어든다. 센서 상에서 memory access는 clock cycle을 2clock 소비하므로 전체 성능은 표 13과 같이 Atmega128에 비해 memory access가 180회 줄어들고 clock cycle은 360clock 개선된다. 또한 Algorithm 3에서 Step 1~3과 Step 4~6에서 수행하게 되는 register clear는 Atmega128의 경우 총 60번의 clear연산을 수행해야 하지만 MSP430의 경우 30번의 clear연산만 수행된다.

표 13 Atmega128과 MSP430 memory access비교

| | 연산자 A 불러오기(회) | 피연산자 B 불러오기(회) | 결과값 C 저장하기(회) |
|---|---|---|---|
| Atmega128 | 150 | 150 | 60 |
| MSP430 | 75 | 75 | 30 |

다른 구현기법으로 block comb method는 이전 연산에 사용되었던 (피)연산자를 재사용하면 memory access를 24번 줄일수 있다. 표 14의 회색으로 색칠된 부분은 이전에 사용되었던 값을 재사용하여 memory access가 줄어든 부분이다.

표 14 register에 저장되는 (피)연산자의 순서

| 곱셈순서 | 연산자 A의 register | | | 피연산자 B의 register | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 0 | 2 | 1 | 0 |
| 2 | 5 | 4 | 3 | 2 | 1 | 0 |
| 3 | 2 | 1 | 0 | 5 | 4 | 3 |
| 4 | 2 | 1 | 0 | 8 | 7 | 6 |
| . . . | | | | | | |
| 24 | 11 | 10 | 9 | 14 | 13 | 12 |
| 25 | 14 | 13 | 12 | 14 | 13 | 12 |

#### 4.4.2 MSP430상에서의 left구현 기법

표 15에는 Atmega128과 MSP430상에서의 left rotate shift기법의 차이점을 비교해서 나타내고 있다. Atmega128에서는 하나의 register를 더 사용하여 shift되어 올라오는 값을 저장한다. 하지만 MSP430에서는 추가적으로 사용할 수 있는 register가 부족하다. 따라서 $R_9$에서 left shift를 수행하고 나서 발생하는 carry값을 다시 $R_4$에 addition해 주는 방식으로 구현하였다. 해당 기법이 가능한 이유는 가장 하위 register인 $R_4$의 하위 bit들은 left가 수행되고 나면 남는 공간이 된다. 따라서 전체 수행에 영향을 미치지 않는 공간을 활용하여 left shift시 발생하는 carry값을 저장한다.

표 15 ATmega128과 MSP430의 left연산비교

| ATmega128L | MSP430 |
|---|---|
| $lsl\ R_0$ | $rla\ R_4$ |
| $rol\ R_1$ | $rlc\ R_5$ |
| $rol\ R_2$ | $rlc\ R_6$ |
| $rol\ R_3$ | · |
| · | $rlc\ R_9$ |
| $rol\ R_{12}$ | $adc\ R_4$ |

표 16 [표 15]에 사용되어진 파라미터 정리

| 명령어 | 인자 | 수행되는 연산 |
|---|---|---|
| lsl | $R_d$(destionation) | left shift연산을 수행한다. |
| rol | $R_d$(destionation) | Rotate left를 수행한다. |
| rla | $R_d$(destionation) | left shift연산을 수행한다. |
| rlc | $R_d$(destionation) | Rotate left를 수행한다. |
| adc | $R_d$(destionation) | carry와 register를 합한다. |

#### 4.4.3 IF-ELSE 분기문 구현

IF-ELSE문 작성 시 Instruction은 프로세서에 따라 크게 차이가 난다. 만약 MSP430에서 한 bit씩 검사를 한다면 Atmega128에 비해 약 3000 clock이 더 소모되게 된다. 따라서 BIT연산 수행 시 연산을 Top-Down방식으로 검사하면 속도 개선이 된다. [표 17]에서와 같이 16bit가 있다면 4bit씩 한번에 검사한다. [표 17]에서와 같이 [Step 1]에서 bit연산 결과로 도출된 carry값이 0인 경우 4bit를 한번에 뛰어 넘을 수 있고 아닌 경우 [Step3], [Step6], [Step9], [Step12]에서 1, 2, 4, 8자리에 해당하는 bit를 모두 검사한다. 이는 Top-Down방식이 아닌 기존방식보다 4bit마다 4clock이 늘어나 1200 clock이 증가하게 되지만 모든 값이 0인 경우 3600 clock이 줄어든다.

표 17 MSP430상에서의 IF-ELSE문 구현기법

```
1:  bit #0x000F, r15
2:  jnc zero
3:  bit #0x0001, r15
4:  jnc next_1
5:  "Algorithm 3 : STEP21"
6:  next_1 : bit #0x0002, r15
7:  jnc next_2
8:  "Algorithm 3 : STEP21"
9:  next_2 : bit #0x0004, r15
10: jnc next_3
11: "Algorithm 3 : STEP21"
12: next_2 : bit #0x0008, r15
13: jnc zero
14: "Algorithm 3 : STEP21"
15: zero : "다음 연산"
```

### 5. 결론

본 논문은 Atmega128상에 구현된 Block comb method Multiplication을 MSP430적용기법을 제안함으로써 성능개선점을 제시한다. [표 18]에 나타난 것은 MSP430과 Atmega128의 성능비교이다. 표에서 알 수 있듯이 Test Vector에 따라 성능이 차이가 많이 나는 것을 볼 수 있다. 그 이유는 Algorithm 3의 Step 16~25에서 연산은 bit값이 '1'인지 '0'인지에 따라 분기가 일어난다. Test Set 1은 인자값을 모두 0으로 한 경우이며 Test Set 2는 모두 1로 설정한 경우이다. 이를 통해 MSP430프로세서상에 제안된 Algorithm3을 통한 Pairing구현이 Sirase에 의해 Atmega128상에 구현된 결과인 1.93 sec보다 들어오게 되는 인자값에 따라 개선이 가능하다는 것을 알 수 있다.

표 18 Atmega128과 MSP430의 "Algorithm 3" 수행성능비교

| | Test Set1 (Clock / msec) | Test Set2 (Clock / msec) |
|---|---|---|
| Atmega128 | 6521 / 4.075 | 8921 / 5.575 |
| MSP430 | 3731 / 2.331 | 12131 / 7.581 |

참고 문헌


[1] 이창열, "사물통신망을 위한 식별 및 관리 체계", 사물통신망 기술 및 전망 세미나, 방송통신위원회, 2009. 6

[2] 이일우, 한동원, "IT기반의 스마트그리드 기술", 한국정보기술학회지 제7권 제1호 pp. 25 - 30, 2009. 12

[3] Masaaki Shirase, Yukinori Miyazaki, Tsuyoshi Takagi, Dong-Guk Han, Dooho Choi, "Efficient Implementation of Pairing-Based Cryptography on a sensor Node", IEICE TRANSACTIONS on Information and Systems Vol.E92-D No.5 pp. 909-917, 2009. 5

[4] Shamir, "Identity-Based Cryptosystems and Signature Schemes.", In CRYPTO'84: on Advances in cryptology, pp47-53 ,1984

[5] Moisés Salinas R., Gina Gallegos G., Gonzalo Duchén S., "An Authentication Protocol for Sensor Networks using Pairings", 2009 International Conference on Eletrical, Communications, and Computers, pp. 168-172, 2009

[6] A. Joux, "A one round protocol for tripartite diffie-hellman", In ANTS-IV: the 4th Int'l Symposium on Algorithmic Number Theory, pp. 385-394, 2000

[7] R. Sakai, K. Ohgishi, M. Kasahara, "Cryptosystems based on pairing", In Symposium on Cryptography and Information Security (SCIS2000), pp. 26-28, 2000

[8] D. Boneh, M. Franklin, "Identity-based encryption from the weil pairing", In CRYPTO 2001, number 2139 in LNCS., pp. 213-229, 2001

[9] D.Boneh, B. Lynn, H. Schcham, "Short signatures from the Weil pairing", journal of Cryptology, pp. 297-319, 2004

[10'] C. Shu, S. Kwon, K. Gaj, "FPGA accelerated Tate pairing based cryptosystems over binary fields," Cyptology ePring Archive, Report 2006/179, 2006