Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.24, NO.4, Aug. 2014 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2014.24.4.593

경량화 암호의 GEZEL을 이용한 효율적인 하드웨어/소프트웨어 통합 설계 기법에 대한 연구*

김 성 곤,† 김 현 민, 홍 석 희‡
고려대학교 정보보호연구원

Research on efficient HW/SW co-design method of light-weight cryptography using GEZEL*

Sung-gon Kim,† Hyun-min Kim, Seok-hie Hong‡
Center for Information Security Technologies, Korea University

요 약

본 논문에서는 하드웨어로 경량 암호 HIGHT, PRESENT, PRINTcipher를 설계하고 소프트웨어로 암호 운영모드를 구현하여 대칭키 암호에 대한 효율적인 하드웨어/소프트웨어 통합설계 방법을 제안하였다. 우선 효과적인 통합설계 언어인 GEZEL 기반으로 대칭키 암호를 하드웨어로 구현한 후 FSMD 방식의 각 암호 모듈을 unfolding, retiming 등 하드웨어 최적화 이론을 적용하여 성능을 향상시켰다. 또한, 8051 마이크로프로세서에 대칭키 암호 운영모드를 C언어로 구현하여 서로 다른 운영모드를 지원하는 다양한 플랫폼에 적용할 수 있게 하였다. 이때 하드웨어/소프트웨어간의 신뢰성 있는 통신 확립과 통신 간 발생할 수 있는 시간 지연을 막기 위하여 하드웨어의 통신 코어와 암호 코어를 분리하여 병렬적으로 수행되어 암호화 연산 수행 중에도 메시지를 송/수신 할 수 있도록 처리하는 개선된 handshake 프로토콜을 사용하여 전체적인 성능을 향상시켰다.

ABSTRACT

In this paper, we propose the efficient HW/SW co-design method of light-weight cryptography such as HIGHT, PRESENT and PRINTcipher using GEZEL. At first the symmetric cryptographic algorithms were designed using the GEZEL language which is efficiently used for HW/SW co-design. And for the improvement of performance the HW optimization theory such as unfolding, retiming and so forth were adapted to the cryptographic HW module conducted by FSMD. Also, the operation modes of those algorithms were implemented using C language in 8051 microprocessor, it can be compatible to various platforms. For providing reliable communication between HW/SW and preventing the time delay the improved handshake protocol was chosen for enhancing the performance of the connection between HW/SW. The improved protocol can process the communication-core and cryptography-core on the HW in parallel so that the messages can be transmitted to SW after HW operation and received from SW during encryption operation.

Keywords: HIGHT, PRESENT, PRINTcipher, GEZEL, HW/SW co-design, light-weight cryptography

접수일(2014년 7월 25일), 게재확정일(2014년 8월 5일)
* 본 연구는 미래창조과학부 및 정보통신산업진흥원의 대학 IT연구센터육성 지원사업의 연구결과로 수행되었음 (NIPA-2014-H0301-14-1004)

† 주저자, sgkim8308@korea.ac.kr
‡ 교신저자, shhong@korea.ac.kr(Corresponding author)

I. 서 론

암호 시스템은 암호화 및 복호화에 공개키와 비밀키를 각각 다르게 사용하는 공개키 암호 시스템과 하나의 비밀키로 암호화 및 복호화를 수행하는 대칭키 암호 시스템으로 크게 분류할 수 있다. 특히, RFID나 USN 같은 저전력/경량화 환경에서는 안전성을 보장하면서도 효율적인 암호 알고리즘을 필요로 한다. 이러한 환경에서 효과적으로 사용하기 위해 평문을 블록 단위로 암호화를 수행하는 대칭키 방식의 경량 암호가 개발되었다.

국내에서 개발된 HIGHT는 2006년 한국정보통신기술협회(TTA) 표준으로 제정되었으며, 2010년에 국제표준화기구(ISO/IEC) 18033-3의 표준으로 제정되었다. HIGHT는 128비트 비밀키를 사용하는 64비트 대칭키 암호로서 32라운드 8-branch type generalized feistel 구조이다. PRESENT는 31라운드 SPN구조의 64비트 대칭키 암호로서 80/128비트 비밀키를 사용한다. 그리고 PRINTcipher는 SPN구조의 48/96비트 대칭키 암호로서 80/160비트 비밀키를 사용하며, 48/96라운드를 가진다.

이러한 경량 암호를 포함하여 대칭키 방식의 암호는 단독으로 사용되기 보다는 다양한 길이를 가진 메시지를 암호화하기 위해 암호 운영모드와 함께 사용된다.

또한 암호 시스템을 설계할 때 하드웨어와 소프트웨어 각각의 모듈로 따로 설계하기 보다는 각 설계 방법의 이점들을 최대한 살릴 수 있는 하드웨어/소프트웨어 통합 설계 기법이 많이 적용되어 사용되고 있다. 하지만, 하드웨어/소프트웨어의 언어적 특징이 상당히 다른 면이 있고, 최적화 방법도 상이하기 때문에 각각의 장점을 활용하여 효과적으로 설계하기는 쉽지 않다. 이러한 문제점을 해결하기 위하여 2003년 Patrick. R. Shaumont 등에 의하여 GEZEL이 개발되었고, 이후 AES, DES, SHA-1, HECC, RIPEMD-160등 다양한 암호 알고리즘과 해쉬 암호를 개발하는데 효과적으로 이용되었다[1-4].

GEZEL은 하드웨어 단독 설계뿐만 아니라 특히 하드웨어와 소프트웨어의 통합 설계에도 적합하도록 개발되었다. 통합 설계에 있어서 GEZEL은 ARM, 8051 마이크로프로세서의 IP core block을 지원하기 때문에 별도의 컴파일 없이도 GEZEL에서 지원하는 IP block 코드를 삽입하는 것만으로도 ARM, 8051 마이크로프로세서들과 port-mapped 또는 memory-mapped 방식으로 통신이 가능하다[5].

GEZEL은 다른 하드웨어 언어와는 다르게 FSMD(Finite State Machine with Datapath)를 사용하는 언어이기 때문에 각 하드웨어 서브 모듈을 데이터패스(datapath)로 정의하여 하드웨어 블록(module) 단위의 설계가 가능하다.

본 논문에서는 GEZEL의 FSMD기반의 다양한 장점들을 기반으로 최적설계 기법들을 적용하여 대칭키 암호 HIGHT, PRESENT, PRINTcipher를 설계하였다.

첫 번째로, HIGHT의 경우 알고리즘 구성상 서브키를 16개 사용하고 난 이후에 입력된 비밀키가 바뀌는 것에 착안하여 최적화를 수행하였다. 그래서 한 사이클에 4라운드씩 연산을 수행하는 unfolding 방법과 설계 과정에서 레지스터의 위치를 변경하여 critical path의 delay를 감소시키는 방법인 retiming 등의 하드웨어 최적화 기법을 적용하여 하드웨어 co-processor의 성능을 향상시켰다. 두 번째로, PRESENT의 경우 각 데이터패스들 사이에 발생할 수 있는 critical path에서의 지연시간을 최소화하기 위하여 레지스터 retiming 기법을 적용하면서 최소 delay값을 갖도록 레지스터의 위치를 변경하였다. 마지막으로, PRINTcipher의 경우 암호 이론 특성상 전처리가 가능한 부분을 미리 테이블로 설정하여 계산량을 감소시키고, 수행 사이클 감소를 위해 unfolding 방법을 적용하였다.

이와 같이 각 암호시스템의 하드웨어 co processor의 성능을 향상시키고, 소프트웨어로 8051 마이크로프로세서에 대칭키 암호의 운용모드를 구현 후 탑재하였다. 그리고 평문과 비밀키 값은 지속적인 업데이트를 통한 안전성과 유연성을 확보하기 위하여 마이크로프로세서의 내부 레지스터에 저장하여 하드웨어 co-processor로 암호 연산 시 전송하도록 설계하였다. 또한 이러한 통합설계 기법은 하드웨어의 메모리 사용으로 인한 면적을 추가적으로 줄일 수 있어서 구현상 효율성도 높일 수 있었다.

전체 시스템의 동작에서 latency가 가장 많이 발생하는 하드웨어/소프트웨어 간의 통신에서는 개선된 handshake 프로토콜을 사용하여 하드웨어와 소프트웨어간의 신뢰성 있는 통신을 확립하였고, 첫 연산 사이클 이후 소프트웨어 연산 시 하드웨어의 동작 대기 시간을 최소화하고 소프트웨어와 병렬적으로 같이 수행하게 하여 하드웨어가 암호화 연산 수행 중에도 소프트웨어 모듈과 메시지를 주고 받을 수 있도록 상호간의 인터페이스를 최적화하여 전체적인 latency

를 감소시키고, 속도를 향상시켰다.

본 논문에서는 통합설계 기법을 적용하여 운용모드가 포함된 대칭키 암호 HIGHT, PRESENT, PRINTcipher를 Altera의 Quartus Ⅱ 13.0 버전을 사용하여 합성하였다. 그리고 모든 하드웨어 모듈은 Cyclone Ⅲ family의 EP3C16F484C6 FPGA 칩에 구현한 결과, HIGHT는 logic element 1199, 최대동작주파수 113.35MHz, 총 동작 시간은 185.26ns로 동작하였고, PRESENT는 logic elements 337, 최대동작주파수 379.79MHz, 총 동작 시간 89.52ns로 동작하였다. 마지막으로 PRINTcipher는 logic element 648, 최대동작주파수 184.54MHz, 그리고 총 동작시간은 157.14ns으로 동작하였다.

본 논문의 구성은 다음과 같다. 2장에서는 대칭키 암호 HIGHT, PRESENT, PRINTcipher에 대한 소개와 각 암호모듈에 적용한 하드웨어 최적화 기법을 설명하였고, 3장에서는 하드웨어/소프트웨어간의 인터페이스 및 통신에 있어서의 기본적인 동작 방식에 대한 설명을 하고 본 논문에서 적용한 최적화 기법을 설명하였다. 그리고 4장에서는 실험 결과를 기술하고, 마지막으로 5장에서 결론을 맺는다.

II. 암호 소개 및 하드웨어 구현 최적화

본 장에서는 대칭키 암호 HIGHT, PRESENT, PRINTcipher에 대한 소개를 하고 각 암호에 대한 하드웨어 구현 최적화 기법에 대해 설명한다.

2.1 대칭키 경량 암호

2.1.1. HIGHT

대칭키 암호 HIGHT는 RFID, USN 등과 같이 저전력, 경량화를 요구하는 컴퓨팅 환경에서 기밀성을 제공하기 위해 개발된 암호이다[6]. HIGHT는 초경량 암호 알고리즘으로 128비트 마스터키, 64비트 평문으로부터 64비트 암호문을 출력하는 32라운드 8-branch type generalized feistel 구조를 가진다. 또한 제한적 자원을 갖는 환경에서 구현될 수 있도록 8비트 단위의 기본적인 산술 연산들인 법덧셈(⊞), XOR(⊕), 좌측순환이동(≪)만으로 설계되었다.

HIGHT 암호에서 사용되는 평문 P와 암호문 C는 다음과 같이 8비트 단위로 나뉘어 표기된다.

- $P = P_7 \| P_6 \| P_5 \| P_4 \| P_3 \| P_2 \| P_1 \| P_0$
- $C = C_7 \| C_6 \| C_5 \| C_4 \| C_3 \| C_2 \| C_1 \| C_0$

각 라운드의 출력값 X_i는 다음과 같이 8비트 단위로 나뉘어 표기된다.

- $X_i = X_{i,7} \| X_{i,6} \| X_{i,5} \| X_{i,4} \| X_{i,3} \| X_{i,2} \| X_{i,1} \| X_{i,0}$

$WK_{0 \le i \le 7}$은 화이트닝키, $SK_{0 \le i \le 127}$은 라운드키이고, 내부 함수 $F_0(x)$와 $F_1(x)$는 다음과 같다.

- $F_0(x) = (x \lll 1) \oplus (x \lll 2) \oplus (x \lll 7)$
- $F_1(x) = (x \lll 3) \oplus (x \lll 4) \oplus (x \lll 6)$

Fig.1. 은 HIGHT 암호 알고리즘의 전체 암호화 과정을 나타내었다.

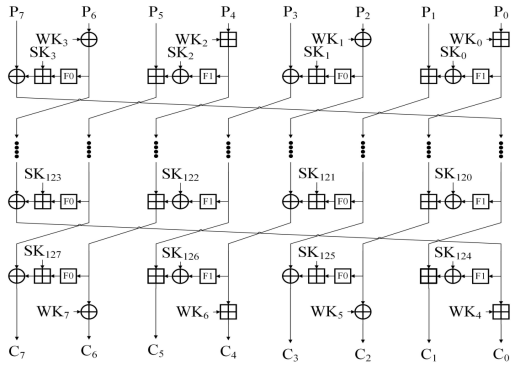

Fig. 1. HIGHT encryption process

HIGHT의 암호화 과정은 초기 화이트닝키 삽입. 라운드 함수, 최종 화이트닝키 삽입과정으로 구성되며, 각 암호화 연산 과정은 다음과 같다.

■ 초기/최종 화이트닝키 삽입

Fig.1. 에서 확인할 수 있듯이 초기 라운드 진입전과 최종 라운드 이후에 화이트닝키를 삽입한다. 화이트닝키($WK_{0 \le i \le 7}$)는 비밀키(MK)를 이용하여 다음과 같은 방법으로 생성한다.

- $WK_i = MK_{i+12}, 0 \le i \le 3$
- $WK_i = MK_{i-4}, 4 \le i \le 7$

■ 라운드 함수

① 라운드 1~31

Fig.2. 는 HIGHT의 라운드 1 ~ 31의 라운드 함수를 나타낸다.

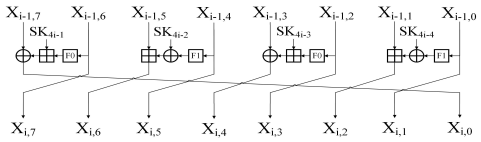

Fig. 2. HIGHT's 1~31 round function

② 라운드 32

Fig.3. 은 HIGHT의 라운드 32의 라운드 함수를 나타낸다. 마지막 라운드는 라운드 1~31와 다르게 좌우 블록의 교차가 없다.

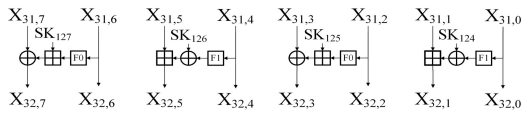

Fig. 3. HIGHT's the last round(32nd) function

HIGHT의 키스케줄은 128비트 비밀키를 입력받아 8비트 화이트닝키($WK_{0 \le i \le 7}$) 8개와 8비트 라운드키($SK_{0 \le i \le 127}$) 128개를 fig. 4. 과정을 통해 생성한다. 또한 LFSR을 이용하여 델타값 ($\delta_{0 \le i \le 127}$)을 만들어 라운드키 생성에 사용한다.

```
WhiteningKeyGeneration(K, WK)
for i = 0 to 7
    if 0 ≤ i ≤ 3, then WK_i = K_{i + 12}
    else, WK_i = K_{i - 4}

SubkeyGeneration(K, SK)
for i = 0 to 7
    for j = 0 to 7
        SK_{(16 · i +j)}    ← K_{(j - i mod 8)} ⊞ delta_{(16 · i + j)}
        SK_{(16 · i +j +8)} ← K_{(j - i mod 8) + 8} ⊞ delta_{(16 · i + j + 8)}
```

Fig. 4. HIGHT's key schedule algorithm

2.1.2 PRESENT

PRESENT-80/128은 31-라운드 64비트 SPN 구조의 대칭키 암호로서 80/128비트 비밀키를 사용한다[7]. 본 논문에서는 80비트 비밀키를 가지는 PRESENT-80을 구현하였다.

PRESENT-80 암호에서 사용되는 주요 표기는 다음과 같다.

- A_i: i번째 라운드의 S-box layer 입력값
- B_i: i번째 라운드의 S-box layer 출력값
- C_i: Permutation Layer의 출력값
- rk_i: i라운드의 Round Key

Fig.5. 는 PRESENT-80 암호 알고리즘의 전체 구조이다. PRESENT-80은 31라운드로 이루어져 있으며, 각 라운드는 다음 세 과정을 거치게 된다.

- AddRoundKey: 64-비트 입력값과 라운드키를 XOR한 64-비트 값을 출력
- S-box Layer(SL): 64-비트 입력값을 16개의 S-box로 연산한 후 연접한 64-비트 값을 출력
- Permutation Layer(PL): 64-비트 입력값을 단순 비트 치환하여 64-비트 값을 출력

PRESENT-80의 키스케줄은 80-비트 비밀키를 입력 받아 32개의 64-비트 라운드 키를 생성한다. 먼저 입력된 비밀키의 상위 64비트를 라운드 1의 라운드키 rk_1으로 추출한 후, 키스케줄에 따라 키를 갱신한다. 그리고 갱신된 키의 상위 64 비트를 라운드 2의 라운드키 rk_2로 추출한다. 이러한 방법을 반복하여 나머지 라운드키를 생성한다. 비밀키를 갱신 하는 키스케줄은 아래 수식과 같다. 여기서 ctr은 라운드 상수를 의미한다.

- $K^i = [k_{79}k_{78} \cdots k_1k_0]$
- $[k_{79}k_{78} \cdots k_1k_0] = [k_{18}k_{17} \cdots k_0k_{79} \cdots k_{20}k_{19}]$
- $[k_{79}k_{78}k_{77}k_{76}] = [S(k_{79}k_{78}k_{77}k_{76})]$
- $[k_{19}k_{18}k_{17}k_{16}k_{15}] = [k_{19}k_{18}k_{17}k_{16}k_{15} \oplus ctr]$
- $K^{i+1} = [k_{79}k_{78} \cdots k_1k_0]$

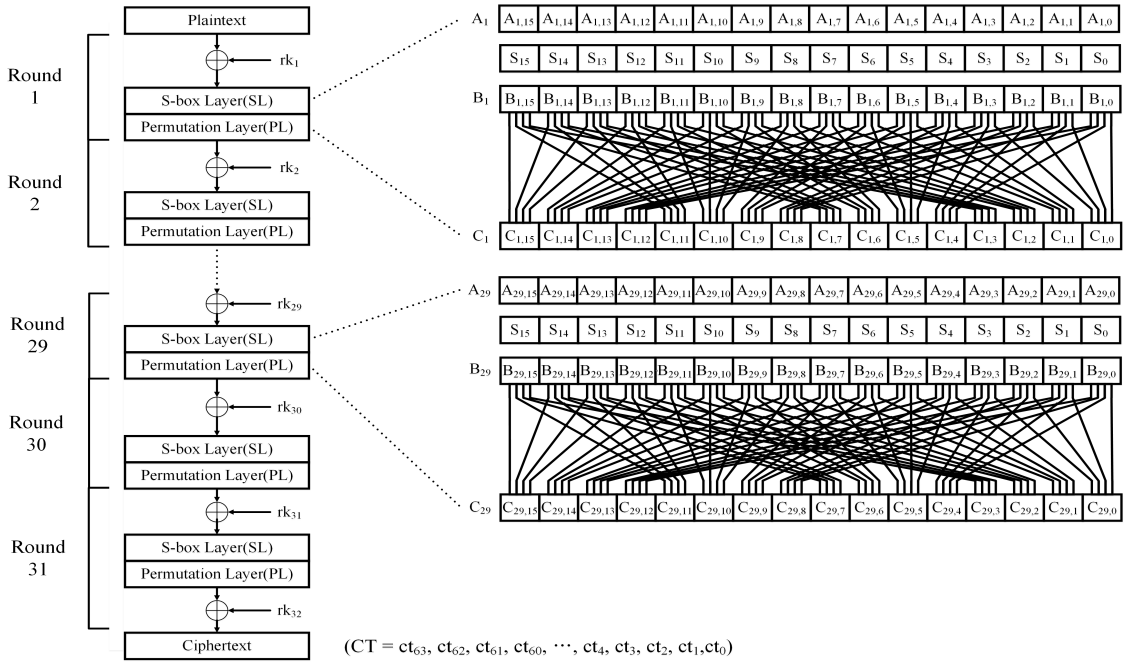

Fig. 5. PRESENT encryption process

2.1.3 PRINTcipher

PRINTcipher-48/96은 SPN 구조의 48/96비트 블록암호로서 48/96라운드로 이루어졌으며, 80/160 비트의 비밀키를 사용한다[8]. 현재까지 하드웨어로 PRINT cipher를 구현한 사례는 없으며, 본 논문에서 처음으로 PRINTcipher-48를 하드웨어로 구현하였다.

Fig.6. 은 PRINTcipher-48 암호 알고리즘의 전체 구조이며, 라운드는 Key XOR, Linear diffusion layer, RC XOR 과정으로 구성되며 각 과정은 다음과 같다.

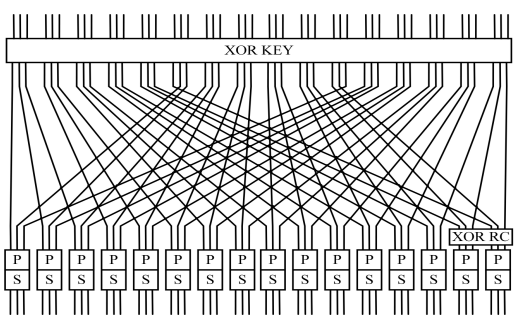

Fig. 6. PRINTcipher-48 encryption process

■ Key XOR

PRINTcipher-48은 모든 라운드에 대해서 동일한 Key가 사용되며, Key의 상위 48비트(k_1)와 XOR을 수행한다.

■ Linear diffusion layer

다음과 같은 diffusion 함수에 의해서 각 비트들을 섞어준다.

- $P(i) = 3 \times i \bmod (b-1)$ for $0 \le i \le b-2$
- $P(i) = b-1$ for $i = b-1$

■ RC(Round Constant) XOR

라운드 상수는 현재 상태의 가장 오른쪽 비트들의 XOR 값으로 만들어진다. 레지스터의 상태를 $x_i (0 \le i \le n-1)$ 라고 할 때 n-bit shift register는 다음과 같은 방식으로 갱신된다.

- $t = 1 + x_{n-1} + x_{n-2}$
- $x_i = x_{i-1}$ for $1 \le i \le n-1$

- $x_0 = t$

위와 같이 생성된 RC_i는 fig.8. 과 같이 Linear diffusion layer 상태의 마지막 6비트와 XOR 과정을 수행한다.

■ Three bit Permutation & S-box Substitution

PRINTcipher-48의 Permutation과 Substit -ution 과정은 비밀키의 하위 32비트(k_2)에 의해 결정된다. 3비트 단위의 입력을 16개 받아서 출력되는 값은 Permutation과 Substitution 과정을 거쳐서 출력되는 값은 table.1. 과 같다.

예를 들어 k_2의 2비트가 11이고 Permutation & S-box Layer의 입력값이 4라면 그 출력값은 1이 된다. 위와 같은 방식으로 나머지 Permutation & S-box를 처리한다.

Table 1. 3 bit Permutation & S-box Substitution

$a_1 \| a_0$		x	0	1	2	3	4	5	6	7
00	$c_2 \| c_1 \| c_0$	$V_0(x)$	0	1	3	6	7	4	5	2
01	$c_1 \| c_2 \| c_0$	$V_1(x)$	0	1	7	4	3	6	5	2
10	$c_2 \| c_0 \| c_1$	$V_2(x)$	0	3	1	6	7	5	4	2
11	$c_0 \| c_1 \| c_2$	$V_3(x)$	0	7	3	5	1	4	6	2

2.2 하드웨어 최적 설계 기법

2.2.1 unfolding

unfolding은 프로그램의 루프 로직을 풀어서 한 번에 여러 연산을 수행하는 방법이다. 이때 공간의 소비가 있으므로, 속도적 측면과 공간의 효율을 고려해야한다[9].

```
for i = 1 to N
  read a[i];       //3cycle
  b[i] = a[i]+1; //1cycle
  write b[i];      //2cycle
end;
```

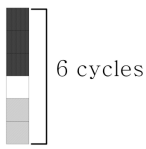

Fig. 7. example of 6 cycles program

Fig.7.을 보면 레지스터 a[i]를 읽는데 3cycle, 계산된 값을 레지스터 b[i]에 할당하는데 1cycle, 할당된 값을 레지스터 b[i]에 쓰는데 2cycle이 소요되고 이를 N번 반복 수행하므로 총 수행 사이클은 6N이 필요하다. 이를 fig.8. 구조로 변경하면 총 8cycle만에 계산이 완료되고 이를 N/3번 반복하므로 총 수행 사이클은 6N에서 8×N/3로 감소되는 효과를 얻을 수 있다.

```
for i = 1 to N/3
  read a[i];          //3cycle
  read a[i+1];        //3cycle
  read a[i+2];        //3cycle
  b[i] = a[i]+1;     //1cycle
  b[i] = a[i+1]+1; //1cycle
  b[i] = a[i+2]+1; //1cycle
  write b[i];         //2cycle
  write b[i+1];       //2cycle
  write b[i+2];       //2cycle
end;
```

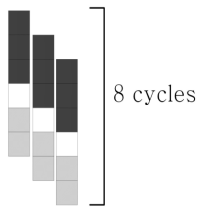

Fig. 8. Adapting unfolding method

2.2.2 retiming

retiming 기법은 레지스터의 위치를 변경하여 clock period/critical path의 감소를 통해 clock rate를 증가시키는 방법이다[9].

예를 들어 fig.9. 과 fig.10. 은 IIR filter에 대한 데이터 플로우 그래프를 나타내었다. 그리고 그 식은 아래와 같이 표현될 수 있다.

- $w(n) = ay(n-1) + by(n-2)$
- $y(n) = w(n-1) + x(n)$
  $= ay(n-2) + by(n-3) + x(n)$

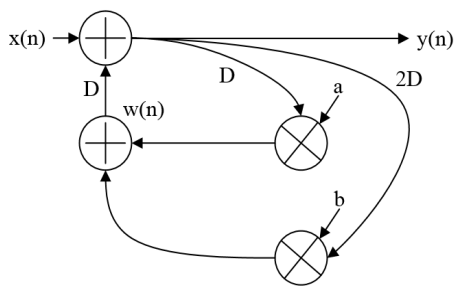

Fig. 9. 4 delay version of an IIR filter

- $w_1(n) = ay(n-1)$
- $w_2(n) = by(n-2)$
- $y(n) = w_1(n-1) + w_2(n-1) + x(n)$
  $= ay(n-2) + by(n-3) + x(n)$

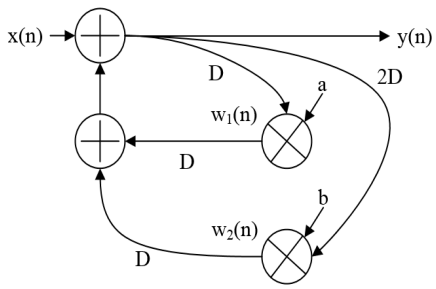

Fig. 10. 5 delay version of an IIR filter

최종적으로 fig.9. 과 fig.10. 의 두 필터는 다른 위치에 딜레이 엘리먼트를 갖고 있으나, 같은 인풋/아웃풋을 갖는다. 이때 곱셈 연산 수행시간을 2u.t(unit time)과 덧셈 연산 수행시간을 1u.t으로 가정하면 fig.9. 의 경우 critical path는 1개의 곱셈과 1개의 덧셈이 되며, 최소 clock period는 3u.t 이 된다. 이에 반해 fig.10. 의 경우 critical path는 2개의 덧셈이 되며, 최소 clock period는 2u.t로 fig.9. 에 비해 약 33%의 감소효과를 얻을 수 있다. 단, retiming 기법은 레지스터 수에도 영향을 주기 때문에 fig.9. 의 경우에는 레지스터를 4개 사용하지만 fig.10.의 경우에는 레지스터가 5개로 증가하게 된다. 따라서 retiming 기법을 적용 시 clock period와 레지스터의 트레이드 오프적 측면을 고려해서 설계를 해야 한다.

2.3 경량 암호의 하드웨어 구현 최적화

2.3.1 HIGHT 하드웨어 구현 최적화

HIGHT는 암호 알고리즘 전체를 수행하는 HIGHT_round_dp 데이터패스를 최상위 개체로 설정하고 키 스케줄을 수행하는 KEY expand 데이터패스를 설계하였다. 또한, look-up table 형태로 내부함수 $F_0(x)$와 $F_1(x)$의 출력 값을 갖고 있는 F_0 데이터패스, F_1 데이터패스, 마지막으로 라운드의 서브키 조합에 사용되는 델타값을 저장해둔 delta 데이터패스의 총 5개의 데이터패스로 나누어 설계를 하였다.

본 논문에서 구현한 HIGHT 하드웨어 모듈은 최상위 개체인 HIGHT_round_dp 데이터패스를 FSM으로 컨트롤하는 FSM 컨트롤러를 설계하여 각 데이터패스들을 이용한 연산이 가능하게 하였다.

HIGHT는 알고리즘 구성상 서브키를 16개 사용하고 난 이 후에 입력된 비밀키의 상위 64비트와 하위 64비트가 각각 8비트씩 순환 왼쪽 시프트를 하는 것에 착안하여 최적화를 수행하였다. 즉, 서브키를 16개 사용하는 4라운드 수행 후 키스케줄을 수행할 수 있게 구현하고, 4라운드 수행 시 발행하는 critical path의 증가문제를 해결하기 위하여 retiming 기법을 적용하여 critical path의 delay를 감소시켜 주었다. HIGHT의 FSM 설계는 fig.11. 과 같다.

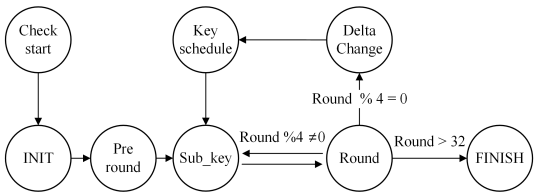

Fig. 11. FSM of HIGHT

■ Check_start 상태

통신 모듈과 소프트웨어의 통신이 연결되었는지 확인하여, 연결이 되지 않았다면 암호화를 시작하지 않는다. 정상적으로 연결이 되고 통신 모듈로부터 평문과 키를 입력 받았다면 다음 단계인 Init 상태로 넘어간다.

■ Init 상태

외부 통신 모듈로부터 평문과 키를 입력 받은 후 HIGHT 내부 레지스터에 저장한다. 그리고 HIGHT 암호화를 수행하는 동안 사용될 레지스터를 초기화하는 상태이다.

■ Pre_round 상태

라운드 시작 전에 입력 받은 평문에 대해서 화이트닝키를 삽입하는 과정을 수행한다.

■ Sub_key 상태

HIGHT의 라운드키로부터 2라운드에 사용될 8개

의 서브키를 생성한다. delta 데이터패스의 출력값과 현재 키를 조합하여 서브키를 생성한다.

■ Round 수행 상태

앞서 생성된 8개의 서브키를 이용하여 2라운드를 수행하고 그 값을 레지스터에 저장하는 상태이다. 이때 HIGHT는 마지막 라운드가 기존 라운드와 다르기 때문에 마지막 라운드는 32라운드 알고리즘을 수행하고 최종 화이트닝키도 삽입한 값을 출력한다.

이 상태에서는 키스케줄을 시작하기 전에 4라운드 중 처음 2라운드만 수행했을 때는 다시 서브키를 생성하는 sub_key 상태로 회귀되고, 4라운드가 모두 수행되었으면 키스케줄을 수행하는 상태로 이행된다.

■ Delta_change 상태

라운드의 서브키 조합에 사용되는 델타 데이터패스의 입력값인 delta_constant를 갱신한 후 레지스터에 저장한다.

■ Key_schedule 상태

키를 갱신하는 키스케줄을 수행하는 상태이다. 현재 키의 상위 64비트와 하위 64비트 각각 8비트씩 좌측 순환 이동(left shift rotation)시킨 값을 키 레지스터에 저장한다.

■ Finish 상태

최종 라운드까지 수행하였다면, 암호문을 외부 통신 모듈로 전송한다. 그리고 HIGHT의 시작 상태를 알려주는 레지스터를 다시 초기화 시킨다.

2.3.2 PRESENT-80 하드웨어 구현 최적화

PRESENT-80은 암호 알고리즘 전체를 수행하는 PRESENT_round_dp 데이터패스를 최상위 개체로 설정하고 Look-up table 형태로 S-box0~3 입/출력을 처리하는 4 개의 S-box 데이터패스, 키스케줄 시 사용하는 Key-box 데이터패스로 총 6 개의 데이터패스로 설계를 하였다.

또한, PRESENT_round_dp 데이터패스를 FSM으로 컨트롤하는 FSM 컨트롤러를 설계하여 암호화를 수행하였다. PRESENT-80의 암호 알고리즘의 라운드 함수는 AddRoundKey, S-box Layer, Permutation Layer의 세 가지 과정으로 구성된다. 본 논문에서는 S-box를 입력과 출력이 필요한 모듈인 Look-up table로 하나의 데이터패스로 구현하였기 때문에 S-box Layer의 입력값들을 레지스터로 선언하여 PRESENT_round_dp 데이터패스와 S-box 데이터패스 간에 발생할 수 있는 지연을 최소화하였다. 따라서 한 사이클마다 S-box Layer, Permutation Layer, 다음 라운드의 AddRoundKey를 수행하도록 설계하였다. 그 이유는 레지스터에 할당된 값은 다음 라운드에 사용할 수 있기 때문에 S-box 입력을 중심으로 사이클을 나누는 것이 효율적이기 때문이다. 그리고 레지스터를 8비트 단위로 분할한 이후에 비밀키와 XOR을 수행하기 때문에 64비트 단위 XOR을 수행할 때보다 효율을 높일 수 있었다. PRESENT-80의 FSM 설계는 fig.12. 와 같다.

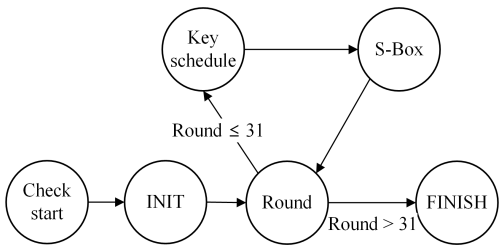

Fig. 12. FSM of PRESENT-80

■ Check_start 상태

통신 모듈과 소프트웨어의 통신이 연결되었는지 확인하여, 연결이 되지 않았다면 암호화를 시작하지 않는다. 정상적으로 연결이 되고 통신 모듈로부터 평문과 키값을 입력 받았다면 다음 단계인 Init 상태로 넘어간다.

■ Init 상태

외부 통신 모듈로부터 평문과 키값을 입력 받은 후 PRESENT_round_dp 내부 레지스터에 저장한다. 그리고 암호화를 수행하는 동안 사용될 레지스터를 초기화하는 상태이다.

본 논문에서 구현한 PRESENT-80의 라운드 수행 알고리즘이 첫 라운드 AddRoundKey 과정을 수행하지 않도록 구성되어있기 때문에 초기 레지스터에

미리 AddRoundKey를 수행하여 저장한다.

■ Round 상태

PRESENT-80의 라운드 함수를 수행하는 상태이다. 라운드 수행 알고리즘은 앞서 설명한대로 현 라운드의 S-box Layer, Permutation Layer, 다음 라운드의 AddRoundKey를 수행한 이후에 8비트 레지스터 8개에 나누어 저장한다. 이 상태를 최종라운드까지 수행한다.

■ Key schedule 상태

키스케줄을 수행하여 키를 갱신하고 라운드 상수 ctr을 증가시킨다. 이때 키는 키값을 저장하는 레지스터에 저장된다.

■ Finish 상태

최종 라운드까지 수행하였다면, 암호문을 외부 통신 모듈로 전송한다. 그리고 PRESENT-80의 시작 상태를 알려주는 레지스터를 초기화 시킨다.

2.3.3 PRINTcipher-48 하드웨어 구현 최적화

PRINTcipher-48는 암호 알고리즘 전체를 수행하는 PRINT cipher_round_dp 데이터 패스를 최상위 모듈로 설정하고 3비트 단위로 Permutation과 S-box Substitution을 수행하는 SP-box 데이터패스 4개를 look-up table 형태로 구현하여 총 5개의 데이터패스로 나누어 설계하였다. 또한. PRINT_round_dp 데이터패스를 FSM으로 컨트롤하는 FSM 컨트롤러를 설계하여 암호화를 수행하였다.

PRINTcipher-48의 암호 설계는 구조 자체가 단순하기 때문에 수행 사이클의 감소에 목적을 두었다. 그래서 라운드 함수는 한 사이클 당 2라운드를 수행하게 unfolding 기법을 적용하여 총 사이클을 감소시켰다. 또한 PRINTcipher-48의 암호화 과정중 Permutation과 S-box Substitution을 따로 처리하지 않고 하나의 테이블로 미리 설정하여 연산량을 감소시켰다. 따라서 PRINTcipher-48의 기존 구조인 Key XOR – Permutation – S-box Substitution를 Key XOR – SP-box 구조로 설계를 하였고 라운드 연산을 수행할 때 SP-box 이전에 레지스터를 두어 현재 라운드의 SP-box, 다음 라운드의 Key XOR을 하나의 프로세스로 묶어서 구현하였다. 그리고 위 프로세스를 한 사이클 당 2번 수행하게 구성하였다.

또한 SP-box 데이터패스 입력값을 3비트 레지스터 16개로 선언하고, Key XOR 과정을 SP-box 데이터패스 내부에서 수행하도록 설계하였다. 그에 따라 48비트의 Key XOR 과정보다 3비트 단위로 나누어져 Key XOR 과정을 수행하여 속도 측면에서의 효율성이 있었다.

PRINTcipher-48의 FSM 설계는 fig.13.과 같다.

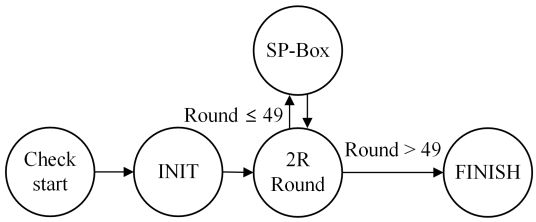

Fig. 13. FSM of PRINTcipher-48

■ Check_start 상태

다른 암호와 동일하게 통신 모듈과 소프트웨어의 통신이 연결되었는지 확인하여, 연결이 되지 않았다면 암호화를 시작하지 않는다. 정상적으로 연결이 되고 통신 모듈로부터 평문과 키값을 입력 받았다면 다음 단계인 Init 상태로 넘어간다.

■ Init 상태

외부 통신 모듈로부터 평문과 키값을 입력 받은 후 PRINTcipher_round_dp 내부 레지스터에 저장한다. 그리고 본 논문에서 구현한 PRINTcipher-48의 라운드 수행 알고리즘이 첫 라운드 Key XOR 과정을 수행하지 않도록 구성되어 있기 때문에 초기 레지스터에 저장할 때 미리 Key XOR을 수행하여 저장한다.

■ 2R-Round 상태

PRINTcipher-48의 2라운드 함수를 수행한다. Permutation – Sbox Substitution 과정을 미리 테이블로 설정한 3비트 SP-box 데이터패스를 설계하

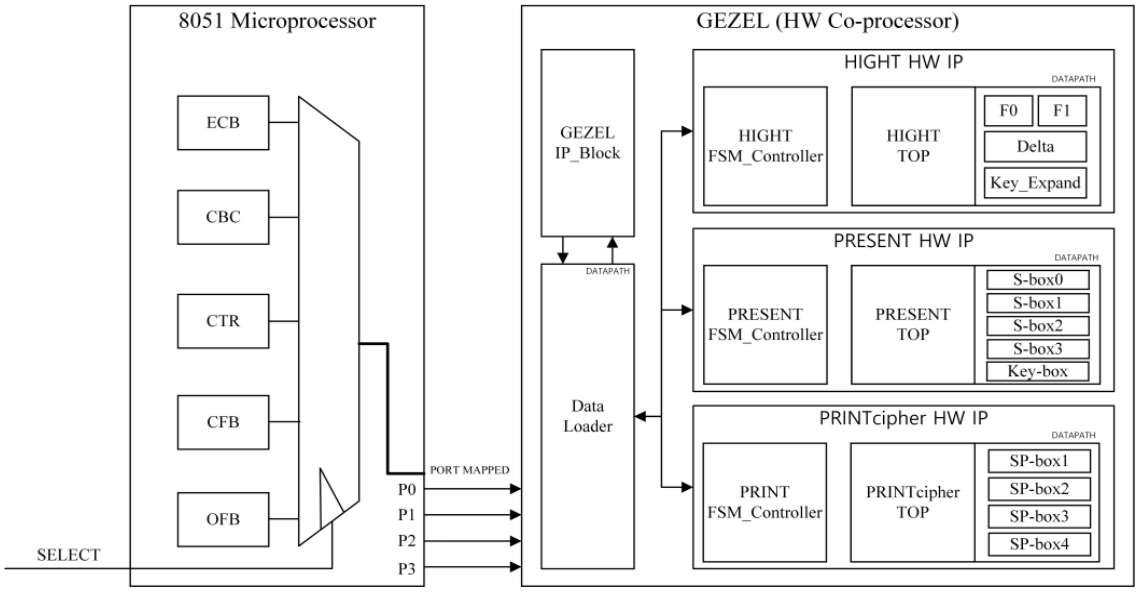

Fig. 14. Hardware/Software Co-design model between GEZEL and 8051 processor

고 SP-box 데이터패스 내부에서 Key XOR을 수행한다.

PRINTcipher_round_dp에서는 SP-box 데이터패스의 출력값을 입력받고 위 과정을 한 사이클 당 두 번 수행하도록 구현하여 한 사이클에 두 개의 라운드가 수행된다.

■ Finish 상태

최종 라운드까지 수행하였다면, 암호문을 외부 통신 모듈로 전송한다. 그리고 PRINTcipher-48의 시작 상태를 알려주는 레지스터를 초기화 시킨다.

III. GEZEL을 이용한 효과적인 하드웨어/소프트웨어 통합 설계 기법

GEZEL은 소프트웨어와 하드웨어 통합설계가 용이하도록 개발된 언어이자 설계 플랫폼이다. 또한 GEZEL의 문법은 직관적이라서 소프트웨어 설계에 익숙한 설계자들이 하드웨어를 쉽게 설계할 수 있다[5].

사이클 기반의 하드웨어 기술 언어인 GEZEL은 데이터패스를 이용한 유한상태기계(FSMD)방식으로로 되어있다. 다른 하드웨어 설계언어는 유한상태기계(FSM) 기반인데 반해 데이터패스가 결합된 유한상태기계를 구성하기 때문에 하드웨어 구현 최적화 이론을 적용하기 쉽다. 또한 데이터패스 기반이라 경쟁효과(race condition)로부터 자유로워서 신호 timing catch가 쉽고, 교착상태에 빠질 우려가 없다. 그리고 8051, ARM, AVR등과의 통합설계를 지원하기 때문에 ipblock 코드를 삽입하는 것만으로 쉽게 하드웨어와 소프트웨어 간의 통신을 구축할 수 있다. 이외에도 별도의 툴 및 컴파일러 없이도 VHDL로의 자동변환이 가능하기 때문에 기존 언어들과의 호환성이 좋고, 보편적으로 많이 사용하는 하드웨어 합성툴을 이용하여 다른 하드웨어 언어로 설계된 기존의 설계 결과와 객관적인 성능 비교가 가능한 장점을 가지고 있다.

3.1 안전성과 유연성 향상을 위한 대칭키 암호 운영 모드 설계

블록 단위로 암호화 되는 대칭키 암호는 다양한 길이의 입력과 출력을 보장하기 위해 운영 모드를 이용한다.

본 논문에서 대칭키 암호의 운용모드는 8051 마이크로프로세서를 사용하여 C언어로 구현하였다. GEZEL은 ARM, 8051 마이크로프로세서와의 통합설계를 지원하기 때문에 다른 추가적인 구현 및 컴파일 없이 GEZEL에서 제공하는 IP block만 코드에 삽입 후 합성하면 8051 마이크로프로세서와의 통신을 통해 평문과 암호문을 전송할 수 있다.

본 논문에서는 GEZEL로 구현된 암호 알고리즘의 코어와 8051 프로세서에서 구현된 ECB (Electronic Code Block), CBC(Cipher Block Chaining), CTR(Counter), CFB(Cipher FeedBack), OFB(Output FeedBack)의 5개의 운영모드를 통해 하드웨어/소프트웨어의 통합설계를 구축하였다.

8051 프로세서에서는 사용자의 입력을 받아서 ECB, CBC, CTR, CFB, OFB의 5가지 운영모드 중 하나를 선택할 수 있다. 즉, 사용 플랫폼과 상황에 맞추어 운영모드와 암호를 선택할 수 있는 유연성을 제공한다. 또한 마스터키가 하드웨어에 저장된 것이 아닌 소프트웨어로부터 입력받는 것이기 때문에 추후 키가 유출되었을 경우 쉽게 수정 및 업데이트가 가능하다는 안전성 측면에서의 유연성을 갖는다.

설계 측면에서도 하드웨어와 소프트웨어의 통합설계 시 각 대칭키 암호의 알고리즘 중 키스케줄 부분을 소프트웨어에서 구현하여 미리 각 라운드키 값을 사전 계산하여 라운드키를 하드웨어로 보내는 방식으로도 구현할 수 있다. 그에 따라 하드웨어에서는 키스케줄 연산을 처리하지 않아도 되기 때문에 면적 감소와 속도 증대 효과를 기대할 수 있다.

3.2 효율적인 하드웨어/소프트웨어 간의 통신 기법

GEZEL을 이용한 통합설계에서 8051 마이크로프로세서가 하드웨어 입출력 장치와 통신을 할 때, 사용하는 방식은 메모리 맵 입출력(memory mapped I/O)방식과 포트 맵 입출력 (port mapped I/O)방식이 있다. 메모리 맵 입출력 방식은 통신을 할 때 입출력과 메모리의 주소 공간을 분리하지 않는 방식이고, 포트 맵 입출력 방식은 입출력 회로와 메모리 주소 공간을 분리하는 방식이다. 입출력 회로가 메모리에 비해 속도가 느린 경우 이 두 공간을 분리하면 속도와 공간의 효율성이 있다. 본 논문에서는 IP block을 연결할 때 포트 맵 입출력 방식을 사용하여 속도와 공간의 효율성을 향상시켰다.

하드웨어와 소프트웨어간의 통신 시 신뢰성을 높이기 위해 8051 마이크로프로세서와 하드웨어 co-processor 간에 개선된 handshake 프로토콜을 사용하였다. 서로 통신이 확립되면 완료가 되었다는 완료 시그널(done signal)을 보내어 데이터를 주고받을 수 있는 상태가 되었는지 확인한다. 본 논문에서는 암호화와 메시지를 주고 받는 부분이 다른 모듈로 설계되어 있기 때문에 동시에 병렬적으로 수행하도록 handshake 프로토콜을 개선하였다. 따라서 하드웨어 co-pressor에서 암호화 수행 중에도 8051 마이크로프로세서와 메시지를 주고 받을 수 있게 하였다.

IV. 실험 결과

본 장에서는 앞장에서 소개한 대칭키 암호 HIGHT, PRESENT, PRINTcipher를 하드웨어와 8051 마이크로프로세서를 사용한 소프트웨어의 통합설계 방식으로 구현한 결과를 보인다. GEZEL은 ARM, 8051 마이크로프로세서와의 통합설계를 지원하기 때문에 IP block 코드를 삽입하면 별도의 컴파일 없이 8051과의 통신이 이루어지면서 결과 값을 출력한다[5].

GEZEL로 대칭키 암호 알고리즘의 코어를 구현하고 8051 마이크로프로세서에는 C언어로 대칭키 암호의 운영모드를 구현한 후 하드웨어와 소프트웨어의 상호 통신을 확립하였다. 하드웨어와 소프트웨어의 통신

Table 2. Simulation Results

Category	Register	Logic Elements	Maximum Frequency (MHz)	Operating Cycle	Operating time (ns)
HIGHT	270	930	100.19	36	359.53
HIGHT(Ver.Optimized)	400	1199	113.35	21	185.26
PRESENT	161	376	265.96	34	127.83
PRESENT(Ver.Optimized)	160	377	379.79	34	89.52
PRINTcipher	145	366	263.85	51	193.29
PRINTcipher(Ver.Optimized)	200	648	184.54	29	157.14

에는 통신 간 발생할 수 있는 지연을 방지하고자 메시지를 주고 받는 부분과 암호화 부분을 다른 모듈로 구현하여 병렬 처리가 가능하도록 개선된 handshake 프로토콜을 사용하였다.

통합설계 기법을 적용하여 운용모드가 포함된 대칭키 암호 HIGHT, PRESENT, PRINTcipher를 VHDL로 변환 후 Altera의 Quartus Ⅱ 13.0 버전을 사용하여 합성하였다. 그리고 모든 하드웨어 모듈은 Cyclone Ⅲ family의 EP3C16F484C6 FPGA 칩에 구현한 결과는 table.2.와 같다.

Ⅴ. 결론

본 논문에서는 GEZEL을 이용한 효율적인 대칭키 암호 HIGHT, PRESENT, PRINTcipher에 대한 하드웨어/소프트웨어 통합설계기법에 대해 연구하였다.

Table 2. 에서 확인 결과 HIGHT에 최적화 이론을 적용한 경우에는 하드웨어 최적화 이론을 적용하기 전보다 레지스터 1.48배, Logic Elements 1.28배 증가하였으나, 속도적 측면에서 Maximum Frequency 1.13배 증가, 총 동작 사이클 0.58배 감소, 총 동작시간은 0.51배 감소하는 효과를 볼 수 있었다.

PRESENT에 최적화 이론을 적용한 경우에는 하드웨어 최적화 이론을 적용하기 전보다 레지스터와 Logic Elements, 총 동작 사이클의 변화는 없으나, Maximum Frequency 1.42배 증가, 총 동작시간은 0.7배 감소하는 효과를 볼 수 있었다.

PRINTcipher에 최적화 이론을 적용한 경우에는 하드웨어 최적화 이론을 적용하기 전보다 레지스터 1.37배 증가, Logic Elements 1.77배 증가하고, Maximum Frequency는 0.69배 감소하였으나 총 동작 사이클 0.56배 감소, 총 동작시간은 0.81배 감소하는 효과를 볼 수 있었다.

본 논문에서 HIGHT의 경우에는 키가 4라운드 단위로 바뀌는 점에 착안하여 unfolding을 적용하여 한 사이클 당 4라운드 단위로 암호화를 수행하였고, 레지스터 위치를 변경하면서 속도의 최적화를 꾀하였다.

PRESENT의 최적설계 기법에 있어서는 암호 알고리즘 과정 중 본래 AddRoundKey, S-box Layer, Permutation Layer의 순서가 아닌 S-box Layer, Permutation Layer, 다음 라운드의 AddRoundKey 순서로 계산을 수행하여 S-box 데이터패스 입력 전에 레지스터를 두어서 최적화를 시켰다.

또한 PRINTcipher의 경우에는 암호화 중 미리 테이블로 설정할 수 있는 부분을 파악하여 테이블로 만들어 연산량을 감소시키고, 한 번에 여러 라운드를 수행하도록 unfolding기법을 적용시켜 사이클을 감소시켰다.

향후 다른 여러 암호에 대해서도 다양한 하드웨어 최적화 이론을 시키는 연구가 가능하며, 하드웨어/소프트웨어 통합 설계의 특성상 키스케줄 과정을 소프트웨어에서 처리하고 암호화 알고리즘을 하드웨어에서 구현하는 등의 최적화가 가능할 것이라 사료된다.

References

〔1〕 Patrick R. Schaumont, “A Practical Introduction to Hardware/Software Codesign,” Springer, Dec. 2012.

〔2〕 L. Batina, D. Hwang, A. Hodjat, B.Preneel, and I. Verbauwhede, “Hardware/Software Co-design for Hyperelliptic Curve Cryptography (HECC) on the 8051 μP,” Proc. CHES-LNCS, no. 3659, pp. 106 - 118, Springer-Verlag, 2005.

〔3〕 Y.K. Lee, H. Chan, and I. Verbauwhede, ″Throughput Optimized SHA-1 Architecture Using Unfolding Transformation,″ ASAP 2006, pp. 354-359, 2006.

〔4〕 M. Knezevic, K. Sakiyama, Y.K. Lee, and I. Verbauwhede, ″On the High-Throughput Implementation of RIPEMD-160 Hash Algorithm,″ Proc. ASAP 2008, pp. 6, 2008.

〔5〕 T. Kwon, H. Kim, and S. Hong, “SEED and ARIA algorithm design methods using GEZEL,” Journal of The Korea Institute of Information Security & Cryptology, VOL.24, No.1, Feb. 2014.

〔6〕 D. Hong, J. Sung, S. Hong, J. Lim, S.Lee, B.S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim and S. Chee, “HIGHT : a new block cipher suitable for low-resource device,” CHES 2006, LNCS 4249, pp. 46059. Springer-Verlag, 2006.

〔7〕 A. Bogdanov, L. Knudsen, G. Leander, C. Paar, "PRESENT: an ultra-light-weight block cipher," Workshop on Cryptographic Hardware and Embedded Systems, CHES'07, LNCS 4727, pp. 450-466, Springer-Verlag, 2007.
〔8〕 Lars Knudsen, Gregor Leander, Axel Poschmann, and Matthew J.B. Robshaw, "PRINT CIPHER : A Block Ciipher for IC-Printing," Workshop on Cryptographic Hardware and Embedded Systems, CHES'10, pp.16-32, Springer-Verlag, 2010.
〔9〕 Andrew B. Kahng, Jens Lienig, Igor L. Markov, Jin Hu, "VLSI Physical Design : From Graph Partitioning to Timing Closure," Springer, vol. 2, pp. 91-124, 2011.

〈저자소개〉

김 성 곤 (Sung-gon Kim) 학생회원
2009년 8월: 아주대학교 미디어학부 학사 졸업
2013년 2월~현재: 고려대학교 정보보호대학원 정보보호학과 석사과정
〈관심분야〉 암호알고리즘 하드웨어/소프트웨어 통합설계, 고속화 구현, 대칭키 암호 알고리즘

김 현 민 (Hyun-min Kim) 학생회원
2006년 2월: 동국대학교 전자공학과 학사 졸업
2005년 12월~2008년 12월: 삼성전자 반도체 총괄 연구원
2009년 3월~2011년 8월: 고려대학교 정보보호대학원 석사
2010년 9월~2011년 8월: COSIC, KULeuven, Belgium, International Scholer
2011년 9월~2012년 8월: COSIC, KULeuven, Belgium, Pre-doctoral Scholar
2013년 3월~현재: COSIC, KULeuven, Belgium, Ph.D candidate
2013년 3월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 부채널 공격, 초경량 암호모듈 설계, Secure Logic Design, 보안칩 설계, 암호 시스템 안전성 분석 및 고속화 구현

홍 석 희 (Seok-hie Hong) 종신회원
1995년 2월: 고려대학교 수학과 학사 졸업
1997년 2월: 고려대학교 수학과 석사 졸업
2001년 8월: 고려대학교 수학과 박사 졸업
1999년 8월~2004년 2월: (주)시큐리티 테크놀로지스 선임연구원
2003년 8월~2004년 2월: 고려대학교 정보보호기술연구센터 선임연구원
2004년 4월~2005년 2월: K.U.Leuven. ESAT/SCD-COSIC 박사후연구원
2005년 3월~2013년 8월: 고려대학교 정보보호대학원 부교수
2013년 9월~현재: 고려대학교 정보보호대학원 정교수
〈관심분야〉 대칭키·공개키 암호 분석 및 설계, 컴퓨터 포렌식