2000년 한국정보처리학회 추계 학술논문발표집 제7권 제2호

SEED 암호화 알고리즘의 하드웨어 구현

송문빈, 고명관, 정연모
경희대학교 전자공학과
e-mail : asic@nms.kyunghee.ac.kr

Hardware Using of the SEED Algorithm

MoonVin Song, MyungKwan Ko and Yunmo Chung
Dept. of Electronics Engineering, Kyung Hee University

요약

국내에서는 전자상거래 진흥을 도모하고 정보사회에서의 국가 경쟁력을 확보하기 위해 128비트 블록암호알고리즘인 SEED를 발표하였다. 본 논문에서는 SEED의 하드웨어적인 응용을 위하여 외부 인터페이스를 고려한 고속의 하드웨어 구현에 대하여 연구하였다. VHDL을 이용하여 모델링 하였으며 시뮬레이션 및 합성 과정을 거쳐 수행을 검증 하였다.

1. 서론

현대 사회는 고도로 발달된 전자, 통신, 컴퓨터 등의 기술을 바탕으로 한 정보통신 및 정보처리를 근간으로 하는 정보화 사회이다. 따라서 공간 및 시간적 제약을 뛰어넘어 범세계적인 규모의 네트워크화로 진전되어 가고 있으며 정보에 대한 다양한 처리, 전송 및 저장을 요구하고 있다. 최근에는 인터넷을 통한 정보 교환 및 이동 통신 기기를 통한 개인정보 교환이 빠르게 확산되고 있으며 또한 전자 상거래, 온라인 뱅킹, 전자주민카드 등이 널리 이용되고 있다. 따라서 정보의 불법적인 유출이나 사용을 막는 정보보안이 아주 중요한 문제로 부각되고 있다. 이를 위해서 암호 및 복호화 알고리즘에 대한 연구가 활발히 이루어지고 있다.[5]

현재 미국은 AES(advanced encryption standard)를 DES(data encryption standard)의 대체할 대안으로 암호화 알고리듬의 표준화 작업을 진행 중이다. 미국 정보산업의 대외적인 영향력을 고려한다면 최종 결정과 동시에 세계적인 산업표준이 될 것이 확실시되고 있다. 이를 위한 표준화 경합 과정을 보면 미국, 유럽, 일본 등 세계 각국의 표준화 참여 열기를 알 수 있다. 표준화 작업에서 주도권을 잡는 것이 관련산업 성장의 관건이 되기 때문이다.

반면에 한국의 암호화 기술은 다른 정보통신 기술에 비해 상당히 뒤쳐져 있는 것이 사실이다.. 정보유출 량이 폭발적으로 증가하고 있는 시기에 암호화와 관련된 기반 기술의 취약은 자칫 정보산업의 주도권을 빼앗기는 결과를 초래할 수 있다.

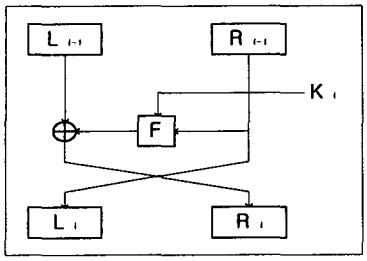
국내에서는 국내 전자상거래 진흥을 도모하고 정보사회에서의 국가 경쟁력을 확보하기 위해 128비트 블록암호알고리즘인 SEED를 발표하였다.[1]

본 논문에서는 SEED 알고리즘의 하드웨어 구현에 대해서 연구하였다. 고속동작을 위한 구조에 대해서 고려하였다. VHDL로 모델링하여 Synopsys로 합성하였다.

2. SEED 알고리듬

SEED는 대칭키 방식으로 128비트의 안전도를 제공하는 블록암호알고리즘으로 1999년 11월 TTA(한국정보통신기술협회)에서 한국 표준으로 제정되었다.[1] SEED의 암·복호화 속도는 3중 DES보다 효율적으로 설계되었으며 16라운드를 수행하는 feistel 구조를 가지고 있다.

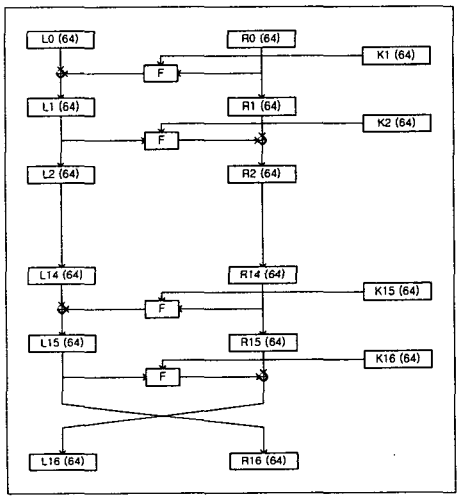
<그림 1>은 SEED의 기본 구조인 feistel 구조를 이용한 암호화 과정에 대하여 나타내었다.



<그림 1> feistel 구조의
암호화 과정

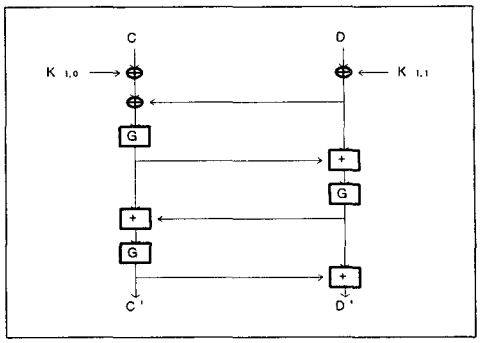
feistel 구조는 간단하게 디자인된 동일한 구조를 여러 번 반복함으로써 구현이 용이하고 복잡도를 증가시킬 수 있는 장점을 가지고 있기 때문에 블록 암호 알고리즘의 기본 구조로서 이용된다. R_{i-1}는 L_i의 입력이 되고, R_{i-1} 과 K_i를 F블록을 거쳐 L_{i-1}과 xor 연산을 한뒤에 R_i로 사용된다.[2]

<그림 2>는 festel 구조를 16번 반복한 SEED의 전체 구조를 나타낸 구성도 이다.



<그림 2> SEED의 전체 구성도

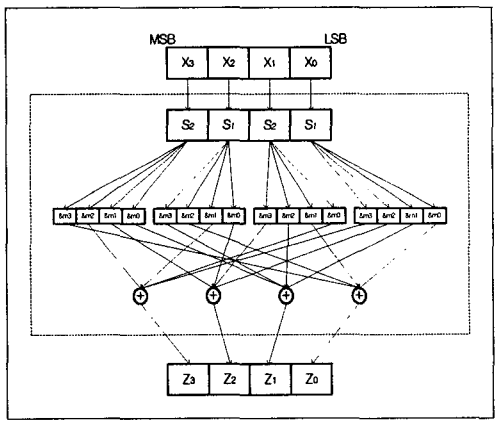
SEED 알고리즘에서는 128비트의 평문 블록을 각 64 비트의 두 개의 블록으로 나누고 128비트 키로부터 생성된 64비트 키를 입력으로 사용하여 F 함수를 거친다. <그림 2>에서와 같이 각 라운드의 결과를 다시 이용하는 16라운드를 반복하여 128비트 암호문 결과를 만든다. <그림 2>에서 Ki는 64비트로 구성되어 있으며 각 라운드마다 해당 라운드에 맞게 생성된 라운드키이다. <그림 3>은 F 함수의 구조도이다.



<그림 3> F 함수의 구조도

F 함수는 32비트의 입력으로 C, D를 받아 32비트의 출력으로 C', D'를 출력한다.

<그림 4> 는 SEED를 구성하는 G함수의 구조를 나타낸다.



<그림 4> G 함수의 구조

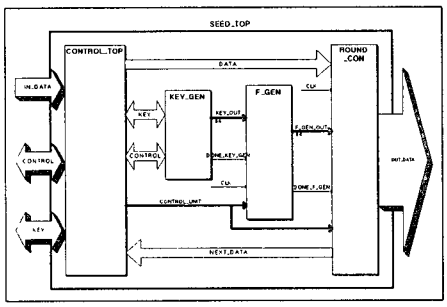
G 함수에서 S1 및 S2 블록은 룩업 테이블로 주어져 있으며 이를 이용하여 입력된 8비트의 Xi는 선형변환된다.[1]

3. 설계 연구 및 구현

SEED는 암호화나 복호화를 수행할 때 동일한 구조를 16라운드 반복하여야 하는 방식이다. 이것을 하드웨어로 구현할 때에는 한 라운드를 설계한 후 설계된 블록을 16번을 재 사용하는 방법이 있다. 이 방법은 하드웨어적인 낭비가 적은 반면 각 라운드의 상태를 제어해야 하며 각 라운드의 결과를 저장해야 한다. 다

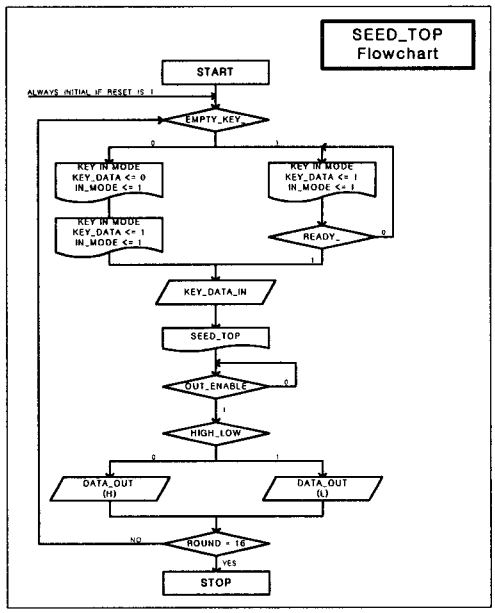
른 방법으로는 16번의 라운드 전체를 일련의 동작으로 고려하여 하나의 하드웨어로 구현하는 방법이 있다. 이 방법은 제어가 쉽고 구조상 16 클럭만큼의 시간을 줄일 수 있지만 동일한 구조를 16번 직렬로 사용해야 하므로 하드웨어 적인 낭비가 제일 심한 구조이다. 또한 위의 두 가지 방식을 절충한 방식으로 2번의 라운드를 하나의 블록으로 설계해 이를 8번 반복하는 구조로 설계를 할 수도 있다. 본 논문에서는 하나의 라운드를 설계해 이를 재 사용하는 방식으로 SEED를 구현하는 방법에 대해 연구하였다.

고속으로 동작하는 SEED를 구현하기 위해서는 동작 주파수를 높이는 방법과 동일한 주파수 상에서 수행 클럭의 숫자를 줄이는 방법이 있는데, 본 연구에서는 좀더 빠른 속도의 암호화 및 복호화를 위해서 한 라운드를 수행하는 클럭의 수를 줄이는 방법을 고려하였다. 즉, 킷값을 생성하는 블록과 F 함수에서의 고속 연산을 위한 구조 개발에 적용하여 연구하였다. <그림 5>는 설계된 SEED의 전체 블록 다이아 그램이다.



<그림 5> 전체 블록 다이아 그램

<그림 5>에서 CONTROL_TOP 블록은 전체 블록 다이아그램인 SEED_TOP을 제어하는 제어 블록으로서, 외부에서 암호화할 데이터 및 초기킷값을 입력받고 내부의 상태와 SEED_TOP 외부의 제어 신호를 판단해 내부 블록들을 제어하고, 외부와의 인터페이스의 일부분을 담당한다. <그림 6>은 SEED_TOP 블록의 동작을 정의한 플로우 차트이다.



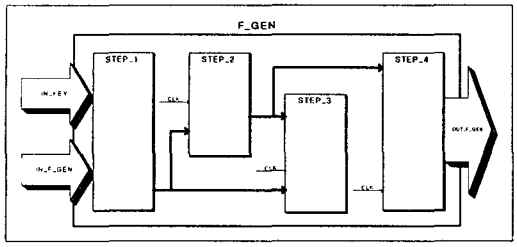
<그림 6> SEED_TOP 플로우 차트

SEED_TOP을 제어하는 외부 제어신호들의 조건에 의해 SEED_TOP 블록의 동작을 정의하였다.

KEY_GEN 키 블록은 키를 생성하고, 암호화 및 복호화 동작을 수행할 때 해당 라운드의 조건에 의해 생성되어있는 킷값을 주변 블록에 제공하는 블록이다. 초기 전원을 인가하거나 시스템을 리셋 한 초기 상태 또는 암호화나 복호화하는 킷값을 변경한 경우에만 KEY_GEN 블록은 각 라운드에 해당하는 킷값을 생성하여 저장한다. 저장된 킷값은 초기 상태가 아닌 다음의 라운드부터는 킷값을 생성하는 동작은 수행하지 않는다. 그리고 생성하여 저장되어 있는 해당 킷값을 F_GEN 블록으로 바로 출력하는 알고리즘으로 구성되어 있다. 따라서 초기 상태의 첫 라운드를 제외한 모든 라운드 동작 시에 저장된 킷값을 사용하므로 그만큼 빠르게 다음 동작을 수행 할 수 있다. 매 라운드마다 바뀌는 킷값을 생성하는 구조와 비교해 같은 동작 주파수에서 더 빠른 수행 결과를 얻을 수 있다. 즉, 전체적인 SEED의 동작 속도를 높일 수 있다. 복호화 시에도 암호화 때와 같은 원리로 동작하여 빠른 수행결과를 얻을 수 있다.

<그림 5> 에서 ROUND_CON 블록은 각 라운드의 결과를 현재의 상태와 비교 판단하여 다음 라운드 연산을 위해 CONTROL_TOP 블록으로 돌려주거나 최종 출력을 위해 각 라운드의 연산 결과를 제어한다.

<그림 7> 은 F_GEN 블록으로서 F 함수에 해당하는 동작을 수행하는 블록 다이아그램이다.



<그림 7> F_GEN 블록 다이아 그램

SEED에서 한 라운드마다 F 블록은 아래의 수식과 같이 총 10번의 G 함수 연산을 수행한다.

$$C' = G[G[G\{(C \oplus K_{i0}) \oplus (D \oplus K_{i1})\} \boxplus (C \oplus K_{i0})] \boxplus G\{(C \oplus K_{i0}) \oplus (D \oplus K_{i1})\}] \boxplus G[G[G\{(C \oplus K_{i0}) \oplus (D \oplus K_{i1})\} \boxplus (C \oplus K_{i0})]$$

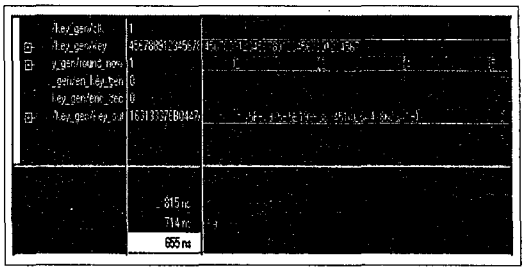
$$D' = G[G[G\{(C \oplus K_{i0}) \oplus (D \oplus K_{i1})\} \boxplus (C \oplus K_{i0})] \boxplus G\{(C \oplus K_{i0}) \oplus (D \oplus K_{i1})\}]$$

그러나 <그림 7>의 F_GEN 블록에서는 10번을 사용하는 G 함수를 4개의 스텝으로 나누어 설계하였으며, 앞의 연산결과를 재 사용하는 방식으로 G 함수를 사용하는 횟수를 원래의 반인 5회로 줄였다. 따라서 하드웨어 적인 측면에서의 게이트 적인 이득과 함께 전체적인 SEED_TOP 블록의 동작 속도를 향상 시켰다.

4. 결과 및 결론

설계된 SEED는 <그림 5>의 SEED_TOP과 같이 외부와의 인터페이스를 고려하여 시스템 레벨에서부터 TOP-DOWN 방식으로 디자인되었다. 각 블록은 VHDL로 모델링 하였으며 각 단계의 시뮬레이션은 modelsim을 통해 검증하였다. 합성은 Synopsys로 합성하였다.

<그림 8>은 KEY_GEN 블록을 시뮬레이션 한 결과이다.



<그림 8> KEY_GEN 블록 시뮬레이션 결과

시뮬레이션 결과는 킷값이 초기상태에서 입력된 후 1 라운드에서 16 라운드에 해당하는 킷값을 생성해 저장하고 있다가 암호화나 복호화 동작시 해당 라운드의 킷값을 현재의 라운드 상태와 제어 신호에 의해 출력하는 결과를 확인 할 수 있다.

본 논문에서는 G함수를 구현하기 위해 룩업테이블을 사용하였다. 향후 연구과제로 룩업테이블을 사용하지 않고 G함수를 로직으로 구현하는 것에 대해 연구하려 한다.

5. 참고문헌

[1] 한국 정보 보호 센터, 128 비트 블록 암호알고리즘(SEED) 개발 및 분석보고서, 1999.

[2] Feistel, "Cryptography and Computer Privacy", Scientific American, May 1973.

[3] "Data Encryption Standard," National Bureau of Standard, Federal Information Processing Standards Publication 46, Jan. 1977.

[4] T. A. Berson, "Long key variants of DES," proc. crypto'82.

[5] D. R. Stinson, Cryptography, CRC Press, 1995.

[6] "Data Encryption Algorithm," X3.92-1981 American National Standards Institute, New York, 1980.

[7] 조주연, 이필중, "Differential Cryptanalysis에 관한 고찰," 한국통신정보보호학회 학회지 제 3 권 1 호, 1993.