블록 암호알고리즘 SEED의 면적 효율성을 고려한 FPGA구현

(Area Efficient FPGA Implementation of Block Cipher Algorithm SEED)

김 종 현 † 서 영 호 † 김 동 욱 ††
(Jong-Hyeon Kim) (Young-Ho Seo) (Dong-Wook Kim)

요 약 본 논문에서는 대한민국 표준 128비트 블록 암호알고리즘인 SEED를 하나의 FPGA에 사상될 수 있도록 설계한다. 이를 위해 VHDL을 이용하여 설계하고 회로는 라운드키 생성부, F함수부, G함수부, 라운드 처리부, 제어부, I/O부로 구성한다. 본 논문에서 SEED는 FPGA를 대상으로 설계하나 ASIC이나 코어(core)를 사용하는 설계 등에 응용될 수 있도록 구현대상을 정하지 않고(technology independent) 범용적으로 설계한다. SEED구조상 많은 하드웨어 자원을 필요로 하는 점 때문에 구현 시 자원제한에 의한 문제점을 최소화하기 위해 F함수부와 라운드 키 생성부에서 사용되는 G 함수를 각각 1개씩 구현하고 이를 순차적으로 사용함으로써 게이트 수를 최소화하여 부가적인 하드웨어 없이 모든 SEED알고리즘이 하나의 FPGA내에 구현되도록 한다. 설계된 SEED는 Altera FLEX10K100에서 구현할 경우 FPGA 사용률이 약 80%이고 최대 28Mhz clock에서 동작하여 14.9Mbps로 암호화를 수행할 수 있다. 설계된 SEED는 공정기술과 무관하고 공정기술의 변경에 따른 부가 하드웨어가 전혀 필요없이 하나의 FPGA로 설계되었다. 따라서 SEED의 구현이나 이를 사용하는 시스템 제작 등에 쉽게 응용할 수 있으리라 사료된다.

Abstract In this paper SEED, the Korea Standard 128-bit block cipher algorithm is implemented with VHDL and mapped into one FPGA. SEED consists of round key generation block, F function block, G function block, round processing block, control block and I/O block. The designed SEED is realized in an FPGA but we design it technology-independently so that ASIC or core-based implementation is possible. SEED requires many hardware resources which may be impossible to realize in one FPGA. So it is necessary to minimize hardware resources. In this paper only one G function is implemented and is used for both the F function block and the round key block. That is, by using one G function sequentially, we can realize all the SEED components in one FPGA. The used cell rate after synthesis is 80% in Altera FLEX10K100. The resulted design has 28Mhz clock speed and 14.9Mbps performance. The SEED hardware is technology-independent and no other external component is needed. Thus, it can be applied to other SEED implementations and cipher systems which use SEED.

1. 서 론

데이타의 비밀 전송과 인터넷을 이용한 전자상거래 등에서 데이타 전송의 안전성, 신뢰성 등을 보증하기 위해 여러 암호화 알고리즘들이 개발되었으며 몇몇 알고리즘들은 국내 및 국제 표준으로 선정되어 여러 분야에서 사용되고 있다.

암호알고리즘은 암호키의 종류에 따라 크게 관용키(conventional key) 알고리즘과 공개키(public kay) 알고리즘으로 분류된다. 관용키 알고리즘의 경우 암호화와 복호화에 사용되는 키가 같으며, 암호화 속도가 빠른 반면 암호키를 같이 전송하여야 하는 이유로 안전성이 떨

· 이 논문은 2000년도 광운대학교 교내학술연구비 지원에 의해 연구되었음.
† 비 회 원 : 광운대학교 전자재료공학과
saltlite@explore.gwu.ac.kr
axl@explore.gwu.ac.kr
†† 정 회 원 : 광운대학교 전자재료공학과 교수
dwkim@daisy.gwu.ac.kr
논문접수 : 2000년 4월 19일
심사완료 : 2001년 5월 30일

어지며 비밀키의 보호 유지수가 많다는 단점을 가지고 있다. 공개키 방식은 암호키와 복호키가 서로 다르며 암호키의 전송이 필요 없다. 따라서 관용키에 비해 보다 안전하며 비밀키의 보호 유지수가 적다. 그러나 일반적으로 암호화의 속도가 느려서 대용량 데이타의 암호화에는 적합하지 못하며 주로 전자서명, 인증, 키분배 프로토콜 등에 사용된다. 현재까지 많은 암호알고리즘들이 개발되었으며 대표적인 관용키 알고리즘으로는 DES[1], FEAL[2], LOK[3], SAFER[4], IDEA[5] 등이 있고, 공개키 알고리즘은 RSA[6], DSS[7] 등이 있다. 국내에서도 암호알고리즘의 중요성을 인식하여 1998년에 대한민국 표준 암호화 알고리즘 SEED[8]가 개발되었다. SEED는 128비트 관용키 블록 암호알고리즘이며 기존의 암호알고리즘에 비해 속도와 효율성 면에서 우수함이 입증되어 향후 국내의 여러 응용분야에서의 사용이 기대되고 있다.

암호 알고리즘들을 소프트웨어로 구현할 경우, 쉽게 구현이 가능하며 적은 양의 데이타를 암호화 할 경우에는 빠른 속도로 처리할 수 있다. 그러나 전자상거래나 무선인터넷보안시스템에서의 사용과 같은 실제 암호알고리즘을 응용할 때 경우에 따라 동시에 수십 건에서 수백 건의 암호화 작업을 수행해야 하며, 만약 이 때 소프트웨어로 암호화를 처리할 경우 시스템 과부하에 의해 처리속도가 현격히 떨어지게 된다. 이는 유·무선 인터넷을 이용한 적용에 암호알고리즘을 소프트웨어로 처리하는 것이 부적절함을 의미한다. 또한 해킹 등에 의한 암호알고리즘의 불법 접근 및 분석도 가능하여 암호알고리즘 자체의 안전 문제도 고려해야 한다. 그리고 이러한 암호시스템을 다른 시스템으로 쉽게 이식하고 설치 및 유지/보수가 용이하도록 암호시스템은 단일 칩으로 구현하는 것이 바람직하다. 따라서 빠른 속도의 암호화와 보다 안전한 암호화 처리를 위해 암호 알고리즘의 하드웨어적인 단일 칩화가 절실히 요구되고 있다. 현재 국내에서는 침입탐지시스템(IDS)이나 무선인터넷보안시스템의 고속 처리를 위해 컴퓨터에 탑재하던 소프트웨어를 하드웨어 전용장비로 개선하는 연구, 개발이 급속히 진전되고 있다. 최근 기업들의 전산시스템이 대용량 네트워크로 급전되는 환경에서 고대역 속도를 무리없이 지원하기 위해선 보안기능을 하드웨어 칩화하는 것이 필수적으로 되고 있다. 또한 ETRI 등 여러 연구 기관에서 국내 암호화 알고리즘을 고속처리를 할 수 있는 ASIC 칩을 구현하고자 연구, 개발을 활발히 하고 있다[9][10].

본 논문에서는 SEED를 하드웨어로 설계하고자 한다. SEED는 128비트로 데이타를 처리하고 Feistel[11]구조로 16라운드를 수행하며 S-box등을 사용하기 때문에 FPGA등 자원의 제한을 받는 경우 구현이 불가능할 수 있다. 따라서 본 논문에서는 SEED 알고리즘을 하나의 FPGA내에 모두 구현할 수 있도록 게이트 수를 줄이기 위한 구조로 설계하며 부가적인 하드웨어 없이 SEED알고리즘을 모두 구현하도록 설계하고자 한다.

2. SEED 구성 및 동작

SEED는 대부분의 대칭키 알고리즘이 사용하는 Feistel[11]구조로 이루어져 있으며 전체적인 구조는 그림 1과 같다. 128비트 크기의 평문(plain text)과 암호키를 입력받은 후 평문을 각각 64비트 블록으로 나누어 암호키에 의해 생성된 각 라운드키와 함께 F함수를 통해 암호화가 수행된다. 모두 16번의 라운드를 수행한 후 최종 128비트 암호문(cipher text)을 출력한다.

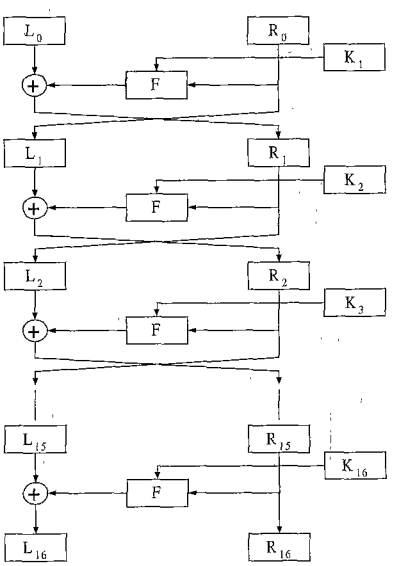

그림 1 SEED의 전체구조

2.1 F 함수

F함수는 Feistel구조 블록 알고리즘의 특성을 구분하는 가장 큰 요소로서, SEED의 F함수는 수정된 64비트 Feistel 암호알고리즘으로 구성되어 있다. F함수의 구조는 그림 2와 같다. 64비트 입력을 32비트씩 C와 D 부분으로 나눈 후 exclusive-OR 과정과 32비트 modulo 연산[12] 및 G함수 등을 통해 F함수가 수행되며 2개의 32비트 결과 값인 C′ 및 D′를 생성한다.

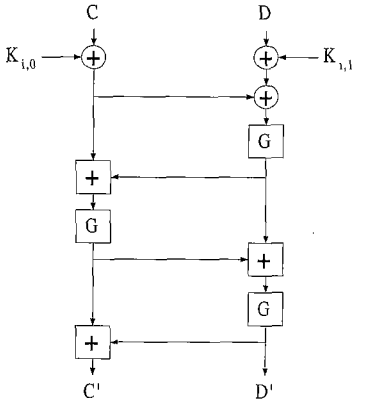

그림 2 F 함수

2.2 라운드키 생성

각 라운드에서 사용되는 라운드키는 128비트 암호키를 64비트씩 좌우로 나눈 후 교대로 8비트씩 좌/우로 회전이동(shift-round) 하고, 그 결과의 32비트에 대한 간단한 산술연산과 G함수를 적용하여 생성된다. 라운드키생성 알고리즘은 그림 3과 같다.

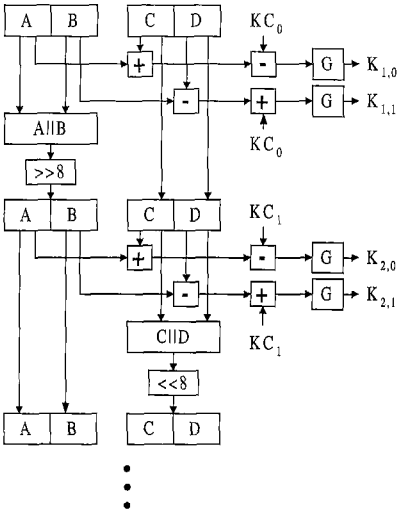

그림 3 라운드 키생성 알고리즘

2.3 G 함수

F함수 및 라운드키 생성시에 사용되는 G함수는, 그림 4에서 보는 바와 같이, 32비트 입력을 4부분으로 나눈 후 각 블록에 대해 S-box(S1-box, S2-box)를 통과시켜, m0에서 m3까지의 값들을 생성한다. 이 값들은 비트별 AND연산(&mi)을 수행한 후 그 결과를 exclusive-OR하여 최종적으로 32비트 출력을 생성한다.

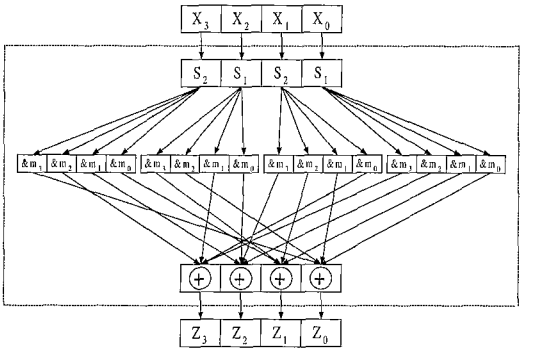

그림 4 G 함수

2.4 S-box

S-Box는 8비트(0에서 255사이의 값)를 받아 8비트 출력값을 만들어내는 함수로서, 내부적으로 비선형 함수를 사용하고 S1-box와 S2-box로 구성이 된다. 그러나 일반적으로는 이미 비선형 함수를 수행한 결과를 표 형식으로 제공한다[8]. 표 1에 S1-box중 일부를 나타내었다. 예를 들면, S1-box의 경우 입력(i)이 0이면 출력(S1(i))은 169가 된다.

표 1 S1-box의 비선형 함수 수행결과

i	S1(i)	i	S1(i)	i	S1(i)	i	S1(i)	i	S1(i)	i	S1(i)	i	S1(i)	i	S1(i)
0	169	1	133	2	214	3	211	4	84	5	29	6	172	7	37
8	93	9	67	10	24	11	30	12	81	13	252	14	202	15	99
16	40	17	68	18	32	19	157	20	224	21	226	22	200	23	23
24	165	25	143	26	3	27	123	28	187	29	19	30	210	31	238
32	112	33	140	34	63	35	168	36	50	37	221	38	246	39	116
40	236	41	149	42	11	43	87	44	92	45	91	46	189	47	1
48	36	49	28	50	115	51	152	52	16	53	204	54	242	55	217
56	44	57	231	58	114	59	131	60	155	61	209	62	134	63	201
64	96	65	80	66	163	67	235	68	13	69	182	70	158	71	79
72	183	73	90	74	198	75	120	76	166	77	18	78	175	79	213

3. SEED의 설계

3.1 전체구조

본 논문에서 설계한 SEED의 전체적인 하드웨어 구조는 그림 5와 같다. 'I/O Interface'는 SEED의 입출력이 128비트이나 실제 응용 시 I/O 핀 수에 의한 문제점을 고려하여 16비트 혹은 32비트로 데이타를 처리한다.

'SEED' 블록은 설계될 하드웨어의 데이타 패스부를 나타내고 있고 F함수부, 라운드키 생성부, G함수부로 구성된다. 입출력에 사용되는 32비트 데이타 블록은 'I/O Interface' 블록을 거친 후 128비트로 확장되어 'SEED' 블록이 암호화 과정을 수행하는데 필요한 입력 데이타가 된다. 'Controller'에서는 'SEED' 블록의 암호화 과정을 수행하기 위해 필요한 제어신호를 생성하게 된다.

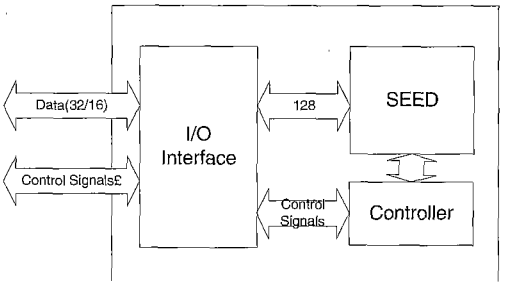

그림 5 전체 SEED 구조

3.2 SEED의 내부 구조 및 특징

SEED 알고리즘을 하드웨어로 구현하기 위한 블록 다이어그램을 그림 6에 나타내었다. SEED 알고리즘 수행 중 라운드 키 생성부에서는 각 라운드 키를 생성하기 위해 두 번의 G함수가 사용되며, F 함수부에서는 각 라운드마다 3번의 G함수가 사용된다. G함수는 구조상 G함수 수행에 필요한 S-box로 인해 많은 자원을 필요로 하며, 하드웨어의 크기가 매우 증가하게 된다. 본 논문에서는 구현 대상을 FPGA로 선택하였으므로 SEED 알고리즘을 모두 구현하기 위해 하드웨어 자원의 양을 줄이는 방법이 필요하다. 본 논문에서는 이를 위해 우선 라운드키 생성부에서 64비트 라운드키를 생성하기 위해 32비트씩 병렬로 G함수로 입력되던 부분을 32비트씩 순차적으로 G함수로 입력되게 함으로써 G함수부의 수를 1개로 줄였다. F함수부에서는 3개의 G함수부를 1개의 G함수로 처리하기 위해 결과를 귀환시켜 처리함으로써 F함수부에서 필요한 G함수의 수를 각각 1개로 줄였다. 그리고 그림 6에 보듯이 G함수를 라운드키 생성부와 F함수가 공동으로 사용하도록 하였다. 따라서 전체적으로 필요한 G함수의 수를 1개로 줄였다.

G함수 구현 시에도 하드웨어 양을 줄이기 위해 G함수 동작에 필요한 S1-box, S2-box를 각각 한 개씩만 구현하고 이들을 순차적으로 사용함으로써 G함수 구현에 필요한 S-box의 수를 반으로 줄였다.

입력으로 들어온 128비트의 암호키는 라운드키 생성을 위한 과정을 Key 생성부 내에서 거친다. Key생성부의 최종 출력값은 MUX를 통해 G함수부로 입력되고 G함수 동작을 거친 후 한 라운드의 암호화를 위한 라운드키가 되어 라운드키 레지스터에 저장된다. 라운드키 생성 과정을 16번 반복하면 라운드키 레지스터에 16 라운드 처리를 위한 16개의 라운드키가 저장된다. 이렇게 저장된 라운드키는 F함수부의 동작시 맨처음 호출되어 데이타 레지스터로부터 입력되는 평문과 연산과정을 거친다. 또한 F함수부 역시 16 라운드과정 동안 동일한 G함수부를 반복적으로 사용하고 G함수부를 거친 데이타 값은 다시 DEMUX를 통해 F함수부로 귀환한다. 그림에서 보이듯이 라운드키 생성부와 F함수부는 G함수부를 공유하고 있다. G함수부는 라운드키 생성부와 F함수부로부터의 입력 데이타를 MUX로 선택하고 DEMUX를 통해서 G함수부의 출력 데이타를 제어하게 된다. 이때 라운드키를 생성하는 과정과 F함수를 수행하는 동작 순서는 중첩되지 않으므로 G함수부의 입력과 출력의 제어에 의한 전체 동작시간의 손실은 없다.

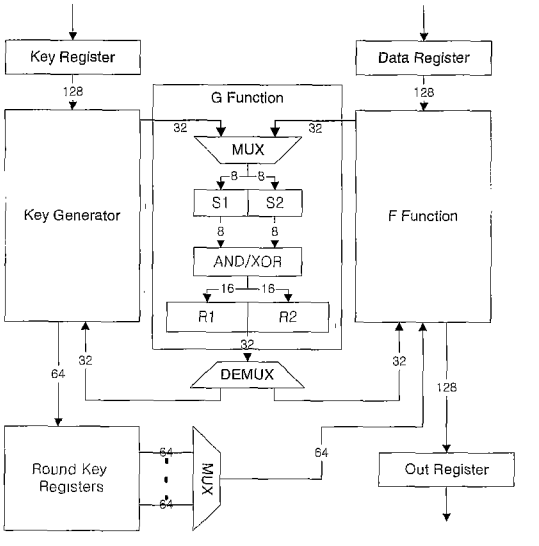

그림 6 SEED 블록 다이어그램

4. 주요 기능 및 동작 설명

4.1 라운드 처리부

라운드 처리부에서는 F함수를 이용하여 16라운드 동안 데이타를 처리함으로써 암호화를 수행하며 내부 구조는 그림 7과 같다. 라운드 처리부에서 사용된 F함수부는 G함수를 라운드키 생성부와 공유하기 위하여 F함

수 내에 G함수부를 포함시키지 않고 G함수부를 독립적인 블록으로 설계한다. F함수부에서는 F함수를 수행하기 위해 한 라운드 동안 모두 3번의 G함수를 사용해야 하며 이를 위해 G함수쪽으로 32비트 출력을 보내고 다시 32비트 결과를 받는 과정을 3번 반복하게 된다. 따라서 G함수 결과를 귀환시켜 F 함수를 처리함으로써 처리속도는 떨어지나 F함수를 수행하기 위해 G함수를 한 번만 구현하면 되므로 F함수 구현에 필요한 하드웨어 양을 줄일 수 있다.

평문 128비트는 64비트 두 데이타 블록으로 나누어져서 레지스터1과 레지스터2에 64비트씩 입력된다. 레지스터1에 저장된 64비트 데이타는 MUX를 거쳐 F함수부의 동작을 수행하고 F함수부의 출력은 레지스터2에 저장된 값과 XOR연산을 거친 후 레지스터2에 저장된다. 또한 레지스터2에 저장되어 있던 64비트 데이타는 그대로 레지스터1로 저장되어 전체 하드웨어가 Feistel구조로 동작하게 된다.

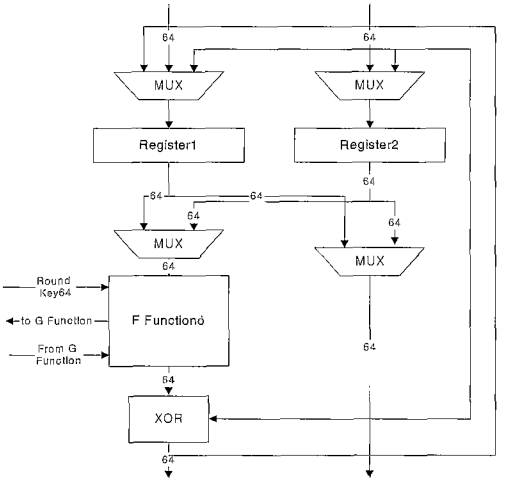

그림 7 Round 처리부

4.2 KEY 생성부

그림 8은, F함수부와 마찬가지로, 라운드키 생성부에서 G함수부분가 제외된 나머지 부분을 나타내고있다. SEED 알고리즘의 라운드키 생성부에서는 라운드 상수와 이전에 입력받은 128비트 암호키 중 64비트를 이용하여 간단한 연산을 수행하고, 그 결과 두개의 32비트 결과를 각각 G함수를 통해 새로운 라운드키를 생성한다. 그러나 본 논문에서는 G함수의 수를 줄이기 위해 연산까지는 병렬로 처리 하지만, G함수 처리를 위해서는 두 개의 32비트 결과를 순차적으로 G함수로 보내도록 하였다. 그 결과 라운드키 생성에 필요한 G함수의 수를 반으로 줄일 수 있었다. 그리고 하나의 라운드키를 생성한 후 각 64비트 암호키 두 블록은 교대로 회전이동 과정을 거치면서 다음 라운드키를 생성하기 위한 입력 데이타가 된다. 즉, 키 레지스터1과 키 레지스터2에 저장된 데이터는 연산블록으로 출력되어 라운드키를 생성하고 동일한 데이타를 회전이동 블록으로 귀환시켜 데이타를 회전이동 시킨 다음 라운드키를 생성하기 위한 준비를 한다. 따라서 16 라운드를 위한 라운드키들은 하나의 라운드키 생성부로부터 반복적으로 생성된다.

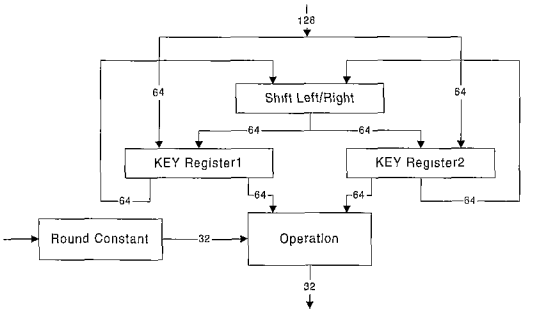

그림 8 KEY 생성부

4.3 F함수부

F함수 블록은 그림 9에 나타낸 것처럼 데이타의 흐름과 타이밍을 제어하기 위한 레지스터, MUX, 연산을 위한 XOR 블록, 그리고 덧셈기로 구성된다. F 함수 블록 내에서 사용되는 32비트 모듈라 덧셈기(modular adder)는 덧셈 속도를 높이기 위해 Carry Look-Ahead(CLA)

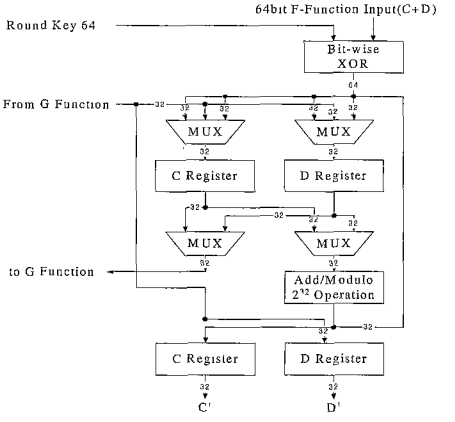

그림 9 F함수부

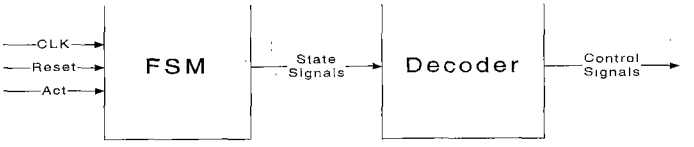

그림 10 제어부

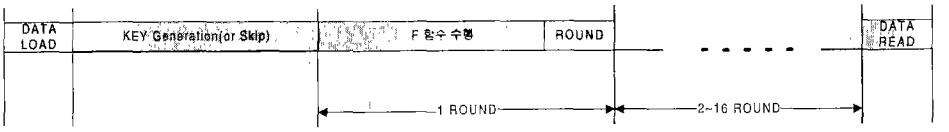

그림 11 SEED 전체 동작 순서

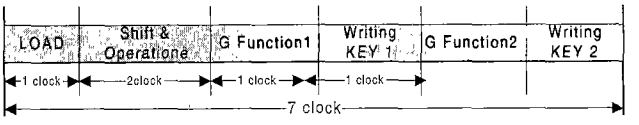

그림 12 Round Key 생성 순서

덧셈기[13]를 사용하였다. 입력으로 들어오는 64비트 데이타는 라운드키와 XOR연산을 거치고 C 레지스터와 D 레지스터로 저장된다. C 레지스터에 저장된 데이타는 MUX를 거쳐 G함수부로 출력된다. 이 데이타는 G함수 동작을 거친 후 귀환되어 D 레지스터의 데이타와 모듈라 덧셈 과정을 수행한다. 이 결과 데이타는 다시 귀환되여 C 레지스터에 저장이 되고 D 레지스터에는 G함수부의 출력값이 들어가게 된다.

4.4 제어부

제어부의 구조는 그림 10과 같이 FSM(Finite State Machine)부와 그 상태에 필요한 제어신호를 생성하는 디코더부로 구성하였으며, 클럭에 의한 동작은 그림 11과 같다. FSM부는 16 라운드를 위한 FSM, 라운드키 생성 및 F함수 처리를 위한 FSM, 그리고 G함수의 동작을 위한 FSM으로 구성된다. 16 라운드 처리를 위한 FSM은 "act" 신호에 의해 활성화되고 동작 순서에 따라서 F함수부와 라운드키 생성부를 위한 FSM을 활성화시킨다. F함수부와 라운드키 생성부는 둘다 G함수부의 동작을 요구한다. 그러나 동시에 G함수부의 동작을 요구하지는 않으므로 G함수부를 위한 FSM은 독립적으로 제어될 수 있다. 세 개의 FSM은 각각 F함수부, 라운드키 생성부, 그리고 G함수부를 위한 제어신호를 디코더를 통해서 생성한다.

28비트 평문과 128비트 암호키를 레지스터에 쓰기 위한 'DATA LOAD' 동작을 거친 후 새로운 암호키를 생성해야 하는 경우 'Key Generation' 동안 라운드키를 생성한다. 만약 기존의 암호키를 사용할 경우 'Key Generation' 과정은 생략된다.

'Key Generation'이 끝난 후 생성된 라운드키와 입력받은 데이타를 이용하여 F함수를 수행하게 되며, 이후 다음 라운드를 수행하기 위한 'ROUND'과정을 거친다. 이러한 동작을 16라운드 동안 반복하여 수행하게 된다. 16라운드 후 암호문을 출력하기 위해 'DATA READ'동작을 수행한다. 제어부에서는 이와 같은 일련의 알고리즘을 수행하기 위해 여러 블록에서 필요한 제어 신호를 생성한다. 16라운드 동안 각각의 동작에 필요한 제어신호들을 생성하게 되는데, 각 동작에 대한 제어신호의 생성은 다음과 같다.

4.4.1 라운드키 생성

라운드키 생성 순서는 그림 12와 같다. 먼저 'LOAD' 동작에서는 라운드키를 생성하기 위해 입력받은 128비트 암호키(1라운드) 혹은 왼쪽/오른쪽으로 이동된 값을 레지스터에 저장한다. 이 후 'Shift & Operation'에서는 G함수 처리하기 전 단계인 연산처리 과정을 거치고 동시에 좌/우 이동을 하게 된다. 연산이 끝난 후 'G함수1'에서는 64비트 데이타 중 상위 32비트가 먼저 G함수로 보내져 처리되고, 그 결과의 상위 32비트 라운드 키는 'Writing KEY1'에서 라운드키 레지스터에 저장된다. 'G함수2'에서는 하위 32비트가 G함수 처리되며 'Writing KEY2'에서 나머지 하위 32비트 라운드키가 저장된다. 이러한 과정이 각 라운드동안 반복되면서 라운드 키를 생성한다.

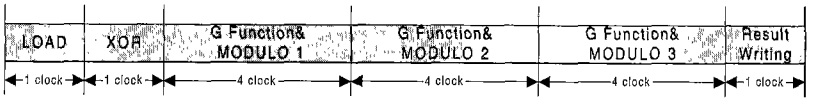

그림 13 F함수 동작 순서

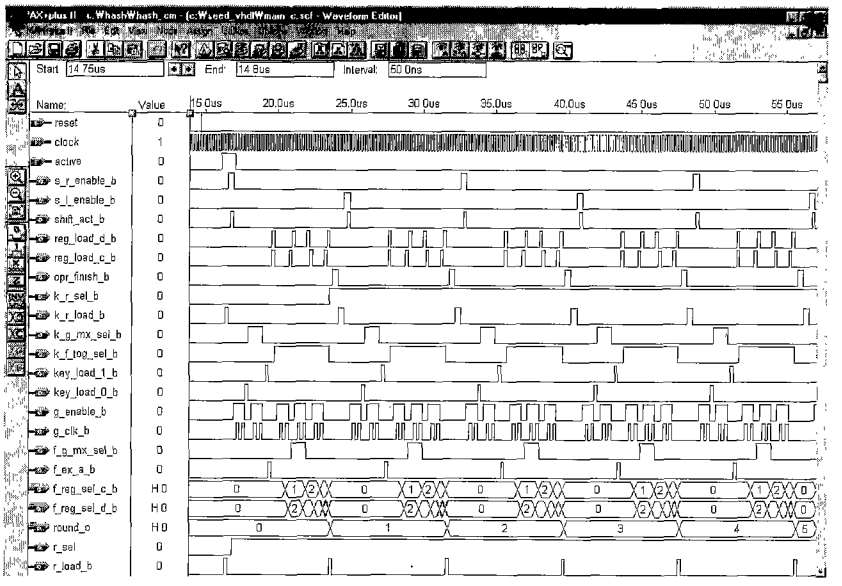

그림 14 제어부 시뮬레이션 수행결과

4.4.2 F함수 수행

F함수 수행순서는 그림 13과 같다. 'LOAD'에서 F함수 수행을 위해 64비트 데이타를 입력받은 후 'XOR'에서 32비트 exclusive-OR을 수행하고 G함수를 수행하게 된다. 이때 본 논문에서는 3번의 G함수 수행을 1개의 G함수로 수행하기 위해 'G Function & Modulo'과정을 3번 거치게 된다. 이 과정에서는 라운드키와 공유하고 있는 G함수로 데이타를 보내고 그 결과를 받는 과정이 포함된다. 이런 과정을 거친 후 'Result Writing'에서 F함수 결과를 레지스터에 저장하게 되며, 이로써 F함수 동작이 완료된다.

그림 14에 제어부에서 생성되는 제어 신호들을 시뮬레이션한 결과를 나타내었다. 'active' 신호에 '1'이 인가됨에 따라 16라운드 동안 필요한 제어신호들이 순차적으로 생성됨을 알 수 있다. 그림에서 보이듯이 제어신호들은 똑같은 신호들이 반복적으로 출력된다. 예를 들면, 약 17us에서 24us까지의 신호들이 1 라운드를 수행시키기 위한 제어신호들이 되고 이러한 주기를 가지고 다음 라운드들을 위한 제어신호들이 규칙적으로 생성된다.

그림에서 순서대로 's_r_enable_b'신호부터 'opr_finish_b' 신호까지는 F함수부를 위한 제어신호이고 'k_r_sel_b'신호에서 'key_load_0_b'신호까지는 라운드키 생성부를 위한 제어신호이다. 그리고 'g_enable_b'신호에서 'g_clk_b'신호까지는 G함수부를 위한 제어신호이고 'f_g_mx_sel_ b'신호에서 'f_reg_sel_d_b'신호까지도 F함수부를 위한 제어신호이다. 마지막으로 'round_o'신호에서 'r_load_b'신호까지는 라운드 처리를 위한 제어신호이다.

4.5 I/O 부

SEED에서는 128비트 크기로 데이타가 처리되지만 PC 혹은 다른 시스템과의 인터페이스를 위해서는 입/출력의 크기를 조절할 필요가 있다. 본 논문에서는 PC와의 인터페이스[14]를 고려하여 평문 및 암호문이 32비트로 양방향(bidirectional) 입/출력이 가능하도록 설계하였다. 이를 위해 I/O 부에서는 32비트 입출력 신호를 이용하여 128비트 평문 레지스터와 암호키 레지스터에 연결시키고 암호화가 수행된 128비트 암호문을 16비트로 출력하도록 하였다. 입출력 비트의 크기를 조절할 필요가 있을 경우 I/O모듈만 수정함으로써 변경이 가능하다.

5. 결과 및 고찰

본 논문에서 제안한 각각의 모듈을 하양식(top-down) 설계방식으로 VHDL[15]을 사용하여 설계하였다. 각 모듈의 최하위 블록은 RTL 수준으로 설계하였으며 설계된 블록을 이용하여 구조적 수준(structural level)으로 회로를 완성하였다[16]. 설계 시 특정 구현대상과 무관한 범용적인 설계를 위해 100% VHDL을 사용하여 설계하였으며 IEEE 표준 라이브러리만을 사용하였다. 설계된 회로의 검증을 위해 본 논문에서는 FPGA로 합성하였으며 합성에 사용된 툴은 SYNOPSYS의 design compiler™ [17]였다. 합성된 결과를 이용하여 Altera MAX+PLUS II에서 시뮬레이션을 수행하여 결과를 검증하였다.

FPGA합성시 Altera 10K 라이브러리를 사용하여 합성하였으며 Altera 10K100에 구현한 결과 약 80%의 FPGA 사용율을 보였다. 이때 본 설계의 특징인 범용성을 고려하여 10K100의 내부 메모리는 전혀 사용하지 않았다. 구현된 FPGA는 최대 28Mhz로 동작이 가능하며 14.9Mbps로 암호화를 수행할 수 있다. 설계시 동작속도보다는 자원의 재사용으로 구현에 필요한 자원을 줄이는데 초점을 맞춘 관계로 암호화 속도가 줄어든 대신 SEED 알고리즘을 부가적인 하드웨어 없이 FPGA로 모두 구현할 수 있었다.

설계된 회로를 FPGA가 아닌 ASIC으로 구현할 경우 FPGA구현을 고려하여 하드웨어의 양을 줄이기 위해 G함수 부분에서의 순차적인 처리와 귀환시켜 처리했던 부분을 병렬로 처리하여 처리 속도를 상당히 높일 수 있을 것으로 예상된다. 즉, 라운드키 생성부에서 G함수 부분을 병렬로 처리하고 F 함수부의 G함수 처리에서 결과를 귀환시키지 않고 병렬 처리할 경우 클럭수를 상당수 줄일 수 있다[18]. 현재 SEED알고리즘을 100% 내부적으로 처리하도록 설계된 결과가 발표되지 않은 관계로 다른 논문과의 절대적인 비교는 어려움이 있다.

그림 15는 암호화를 수행하기 위해 128비트 암호키(0x00010203 04050607 08090A0B 0C0D0E0F)를 입력하는 과정을 나타내고 있다. data_ready신호가 '1'인 것을 확인하여 현재 데이타를 받을 수 있는 초기화 상태임을 확인한 후 reg_add[0..1]를 증가시키며, 32비트 데이타를 모두 4번 입력하여 128비트 암호키를 입력으로 받는다. 입력된 암호키를 입력한 후 그림 16와 같이 128비트 평문(0x00000000 00000000 00000000 00000000)을

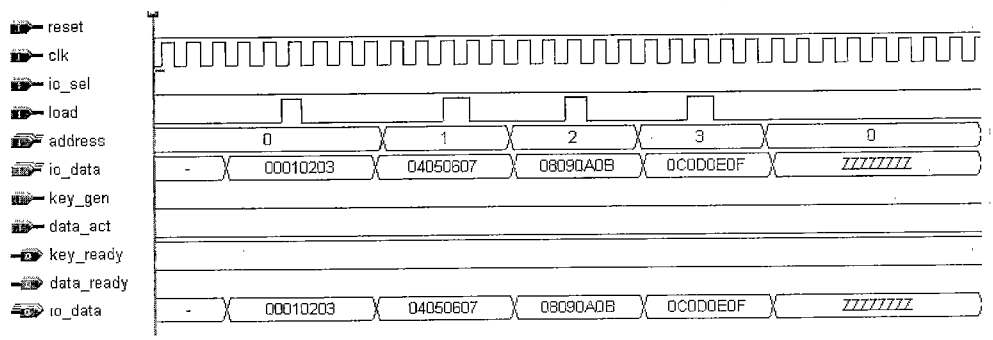

그림 15 암호키 입력

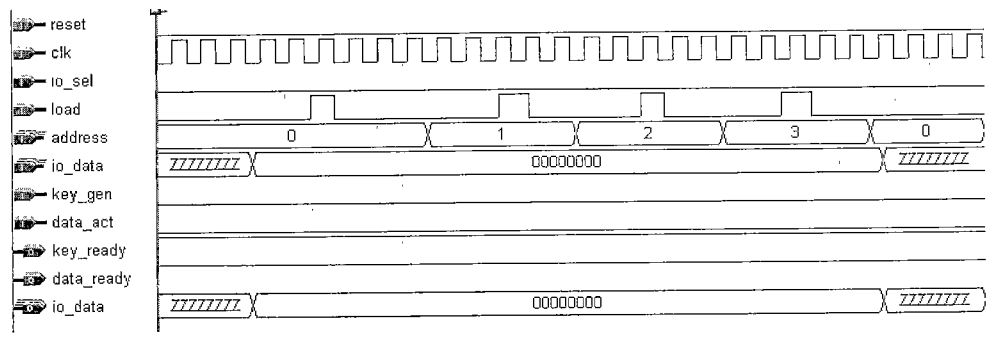

그림 16 평문 입력

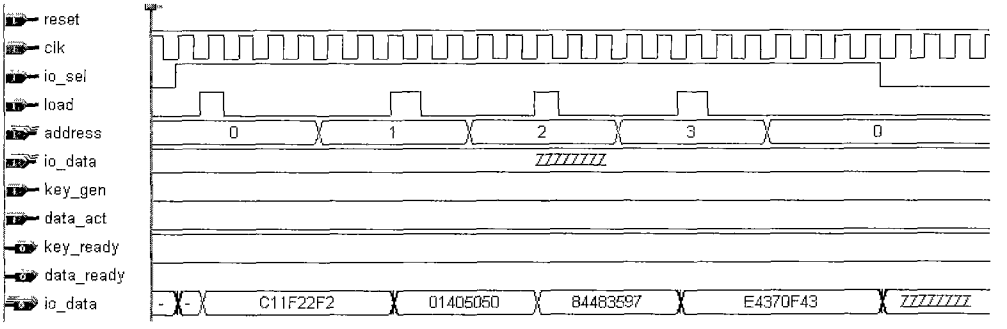

그림 17 암호문 출력

입력한다. 암호키와 평문입력이 끝난 후 data_act신호에 '1'을 인가하여 암호화 과정을 수행한다. 이때, data_ready신호는 '0'이 되어 현재 회로가 암호화 동작중임을 나타내고 다른 데이타가 입력될 수 없는 상태가 된다. 그림 17은 암호화가 끝난 후 암호문(0xC11F22F2 01405050 84483597 E4370F43)을 출력하는 동작을 나타낸다. 그림에서 볼 수 있듯이 생성된 암호문의 결과가 설계된 소프트웨어 혹은 하드웨어의 검증을 위해 한국정보보호센터에서 제공하는 참조 구현값[8]과 같음을 알 수 있어 SEED암호화 동작이 올바르게 이루어짐을 알 수 있다.

SEED는 128비트의 입력과 출력 데이타 블록 크기를 가진다. 그러나 실제적인 상용 칩이나 시스템의 경우 128비트를 수용할 수 있는 데이타 버스를 가진 것은 드물기 때문에 데이타 입력과 출력의 데이타 블록 크기를 나누어야한다. 또한 PCI 버스와의 인터페이스[14]를 고려하여 암호키, 평문, 그리고 암호문의 입력과 출력은 32비트 단위를 가지는 양방향 핀(bidirectional pin)으로 이루어진다.

6. 결 론

본 논문에서는 대한민국 표준 블록 암호화 알고리즘인 SEED를 하나의 FPGA로 구현하는 것을 목적으로 설계하였다. 설계된 회로는 FPGA뿐만 아니라 다른 대상으로도 구현 및 응용이 쉽도록 특정 하드웨어와 무관하게 VHDL을 이용하여 설계하였다. FPGA 구현 시 필요한 자원을 최소로 줄여 SEED의 모든 알고리즘이 구현되도록 하기 위해 F함수 및 라운드키 생성 시에 사용되는 G함수 부분을 한 개만 구현하였으며, F함수 및 라운드키 생성부분에서 이를 공동으로 사용함으로써 필요한 자원의 수를 줄였다. 그 결과 FPGA로 구현 시 부가적인 하드웨어나 외부의 소프트웨어 처리없이 모든 SEED알고리즘을 하나의 FPGA내에 구현할 수 있음을 확인하였다. 또한 PC 및 여러 시스템에서의 응용이 쉽도록 32/16비트 양방향(bidirectional I/O pin)입출력 핀을 사용하였으며, 사용될 시스템 및 버스특성에 따라 입출력 포트의 비트수를 쉽게 변화시킬 수 있도록 입출력 모듈을 따로 설계하였다.

본 논문에서 설계된 SEED는 부가적인 외부 하드웨어나 알고리즘의 일부분을 소프트웨어로 처리하지 않고 독립적으로 SEED 알고리즘을 완벽히 수행할 수 있다. 또한 특정 구현대상을 고려하지 않고 여러 구현대상에 적용이 가능하도록 범용적으로 설계하였다. 따라서 SEED 암호 알고리즘이 사용될 여러 분야에 응용이 가능하며 특히 코어-기반 설계 등에 응용이 가능할 것으로 사료된다.

참 고 문 헌

[1] National Bureau of Standards. FIPS PUB 46 : Data Encryption Standard, January 1997.

[2] Akihiro Shimizu and Shoji Miyaguchi. "Fast data encipherment algorithm FEAL." In David Chaum and Wyn L. Price, editors, Advances in Cryptology-Eurocrypt'87, vol.304 of Lecture Notes in Computer Science, pp.267-280, Springer-Verlag, Berlin, 1998.

[3] Lawewnce Brown, Josef Pieprzyk, and Jennifer Seberry. "LOKI-a cryptographic primitive for authentication and secrecy applications". In Jennifer Seberry and Josef Pieprzyk, editors, Advances in Cryptology Auscrypt'90, vol.453 of Lecture Notes in Computer Science, pp.229-236, Springer-Verlag, Berlin, 1990.

[4] Xuejia Lai and James L. Massey." A proposal for a now block encryption standard. in Ivan bjerre Damgard," editor, Advances in Cryptology Eurocrypt'90, vol.473 of Lecture Notes in Computer

Science, pp.389-404, Springer-Verla, Berlin, 1990.
[5] Lai, X., and Massey, J. "A Proposal for a New Block encryption Standard." Proceedings, EUROCRYPT'90, 1990
[6] Rivest. Shamir, and Adleman, "A Method for Obtainng Digital signature and Public Key Cryptosystems." Communications of the ACM, 2, 1978.
[7] "Proposed Federal Information Proceeding Standard for Digital Signature Standard(DSS)," Federal Register, Vol.56 No.169 30, 1991
[8] 한국정보보호센터, *128비트 블록 암호알고리즘(SEED) 개발 및 분석 보고서*, 12. 1998.
[9] 한승조, "개선된 DES 암호기 설계", Computer Systems and Theory v. 24, n. 1 pp.57-67, 1, 1997.
[10] 한효정, 장형석, 전덕수, 강문식, "LFSR Key Stream Generator를 이용한 DES의 FPGA구현" IDEC MPW 발표회 논문집, pp354-357, 10. 1999.
[11] H. Feistel, Block Cipher Cryptographic System, U.S. Patent #3,798,359,19, 3, 1974.
[12] N.A. Koblitz. *A Course in Numter Teory and Cryptography*. Springer-Verlag, 1987.
[13] Behrooz Parhami, "Computer Arithmetic Algorithms and Hardware Designs," Oxpord University Press, 2000.
[14] Tom Shanley and Don Anderson, *PCI System Architecture*, Mindshare Inc., 1995
[15] I.S 1076-1993, *IEEE Standard VHDL Language Reference Manual*, IEEE, 1993.
[16] L. R. Knudsen, "Block Ciphers-Analysis, Design and Applications," Ph.D Thesis, Computer Science department, Aarhus University, 1994.
[17] Pran Kurup and Taher Abbasi, *Logic Synthesis Using Synopsys*, Kluwer Academic Publishers, 1997.
[18] Michael Keating and Pierre Bricaud, *Reuse Methodology Manual*, Kluwer Academic Publishers, 1999.

김 종 현
1973년 4월 22일생. 1997년 2월 광운대학교 전자재료공학과 졸업(공학사). 1999년 2월 광운대학교 전자재료공학과 졸업(공학석사). 2001년 2월 ~ 현재 광운대학교 전자재료공학과 박사과정. 2000년 3월 ~ 2001년 2월 인티스닷컴(주) 연구원. 2001년 3월 ~ 현재 펜타마이크로(주) 연구원. 관심분야는 VHDL, DSP, Design verification & Test, Cryptosystem

서 영 호
1975년 5월 30일생. 1999년 2월 광운대학교 전자재료공학과 졸업(공학사). 2001년 2월 광운대학교 전자재료공학과 대학원 졸업(공학석사). 2000년 3월 ~ 현재 인티스닷컴(주) 연구원. 2001년 3월 ~ 현재 광운대학교 전자재료공학과 박사과정. 관심분야는 FPGA/ASIC 설계, DSP, Cryptosystem

김 동 욱
1960년 8월 23일생. 1983년 2월 한양대학교 전자공학과 졸업(공학사). 1985년 2월 한양대학교 대학원 졸업(공학석사). 1991년 9월 Georgia 공과대학 전기공학과 졸업(공학박사). 1992년 3월 ~ 현재 광운대학교 전자재료공학과 정교수. 광운대학교 신기술 연구소 연구원. 1997년 12월 ~ 현재 광운대학교 IDEC 운영위원. 2000년 3월 ~ 현재 인티스닷컴(주) 연구원. 관심분야는 디지털 VLSI Testability, VLSI CAD, DSP 설계