내장형 시스템을 위한 128-비트 블록 암호화 알고리즘 SEED의 저비용 FPGA를 이용한 설계 및 구현

(Design and Implementation of a 128-bit Block Cypher Algorithm SEED Using a Low-Cost FPGA for Embedded Systems)

이 강 † 박 예 철 ††
(Kang Yi) (Ye-Chul Park)

요 약 본 논문에서는 국내 표준 128비트 블록 암호화 알고리즘인 SEED를 소형 내장형(8-bit/16-bit) 시스템에 탑재하도록 저가의 FPGA로 구현하는 방법을 제안한다. 대부분 8-bit 또는 16-bit의 소규모 내장형 시스템들의 프로세서들은 그 저장용량과 처리속도의 한계 때문에 상대적으로 계산양이 많아 부담이 되는 암호화 과정은 별도의 하드웨어 처리기를 필요로 한다. SEED 회로가 다른 논리 블록들과 함께 하나의 칩에 집적되기 위해서는 적정한 성능을 유지하면서도 면적 요구량이 최소화되는 설계가 되어야 한다. 그러나, 표준안 사양의 구조대로 그대로 구현할 경우 저가의 FPGA에 수용하기에는 면적 요구량이 지나치게 커지게 되는 문제점이 있다. 따라서, 본 논문에서는 면적이 큰 연산 모듈의 공유를 최대화하고 최근 시판되는 FPGA 칩의 특성들을 설계에 반영하여 저가의 FPGA 하나로 SEED와 주변 회로들을 구현할 수 있도록 설계하였다. 본 논문의 설계는 Xilinx 사의 저가 칩인 Spartan-II 계열의 XC2S100 시리즈 칩을 대상으로 구현하였을 때, 65%의 면적을 차지하면서 66Mpbs 이상의 throughput을 내는 결과를 얻었다. 이러한 성능은 작은 면적을 사용하면서도 목표로 하는 소형 내장형 시스템에서 사용하기에 충분한 성능이다.

키워드 : SEED, 128 비트 대칭형 블록 암호화 알고리즘, FPGA, 하드웨어 설계, 내장형 시스템, 암호화 프로세서

Abstract This paper presents an Implementation of Korean standard 128-bit block cipher SEED for the small (8 or 16-bits) embedded system using a low-cost FPGA(Field Programmable Gate Array) chip. Due to their limited computing and storage capacities most of the 8-bits/16-bits small embedded systems require a separate and dedicated cryptography processor for data encryption and decryption process which require relatively heavy computation job. So, in order to integrate the SEED with other logic circuit block in a single chip we need to invent a design which minimizes the area demand while maintaining the proper performance. But, the straight-forward mapping of the SEED specification into hardware design results in exceedingly large circuit area for a low-cost FPGA capacity. Therefore, in this paper we present a design which maximize the resource sharing and utilizing the modern FPGA features to reduce the area demand resulting in the successful implementation of the SEED plus interface logic with single low-cost FPGA. We achieved 66% area accupation by our SEED dsign for the XC2S100 (a Spartan-II series FPGA from Xilinx) and data throughput more than 66Mbps. This Performance is sufficient for the small scale embedded system while achieving tight area requirement.

Key words : SEED, 128-bit symmetric block cipher algorithm, FPGA, hardware design, embedded system, cryptography processor

† 종신회원 : 한동대학교 전산전자공학부 교수
yk@handong.edu
†† 학생회원 : 한동대학교 전산전자공학부
zangpark@hotmail.com
논문접수 : 2003년 8월 4일
심사완료 : 2004년 4월 16일

1. 서론 : FPGA를 이용한 SEED의 구현 필요성

정보화 사회로 급속히 진입하면서 정보의 보호 및 보

안에 관한 요구가 날로 증가하고 있다. 정보보호를 위한 방법들은 알고리즘이 표준화되어 공개됨으로써 그 안전성을 공증받을 수 있으며 암호화(encryption)와 복호화(decryption) 방식의 표준화를 통해서 암호화 체계의 개발과 유지에 드는 사회적 비용을 대폭 줄일 수 있는 장점이 있다. 반면에, 이러한 공개 암호화 알고리즘을 이용할 때에는 외부의 침입으로부터 암호화 시스템을 견고하게 하기 위해서 평문과 암호문의 짝으로 암호키를 찾아내지 못하도록 복잡한 연산 과정을 포함해야 하는 부담이 있다. 이러한 복잡한 연산 과정은 암호화 시스템의 부하를 증가와 속도 저하 및 소비 전력 증가로 이어진다.

일반적으로 암호화 알고리즘을 전용 하드웨어 (칩)으로 구현하면 다음과 같은 장점들이 있다. 첫째, 암호화 알고리즘을 전용 하드웨어 칩으로 구현하면 소프트웨어로 구현할 때에 비해서 정보 처리 속도의 월등한 증가와 소비 전력 소모의 감소의 효과가 있다. 마이크로프로세서를 이용한 암호화 알고리즘의 소프트웨어 구현에 방식에 비해서 하드웨어 구현은 데이타 경로상의 연산부의 구조가 특정 알고리즘에 대해 최적의 전용 구조를 가지게 할 수 있기 때문이다. 둘째로, 알고리즘 자체의 하드웨어 구현은 암호화 과정 자체의 임의 수정이나 변형이 예방할 수 있다. 소프트웨어로 구현할 경우에 크래커들의 침입이나 바이러스 프로그램 등을 통해서 암호화 시스템이 파괴되어 정보의 보안이 근본적으로 위협받을 수 있으나 하드웨어로 알고리즘을 구현함으로써 이런 위험에서 자유롭게 된다[1].

SEED[2]는 한국정보보호진흥원에서 1999년에 제정 발표한 대한민국 표준 양방향 암호화 알고리즘이다. SEED는 128 비트 단위의 블록으로 데이타를 읽어서 암호화 또는 복호화를 하는 블록암호화 알고리즘으로서 암호화와 복호화에 동일한 128 비트의 키값을 이용하는 대칭 키 방식이다. 이러한 SEED 알고리즘은 전자결제, 전자지갑, 스마트 카드, 휴대용 정보통신 단말기 등의 정보 보호를 필요로 하는 여러 내장형 시스템 등에서의 응용 범위가 넓다. 그런데, 대부분 소규모 내장형 시스템들의 8 비트 또는 16 비트 마이크로프로세서/컨트롤러들은 그 저장용량과 처리속도의 한계 때문에 상대적으로 계산 양이 많아 부담이 되는 이 암호화 과정은 별도의 하드웨어 가속기를 필요로 한다.

한편, FPGA (Field Programming Gate Array)가 과거에는 복잡한 하드웨어 검증을 목적으로 하드웨어 원형 제작(prototyping)용으로만 주로 사용되었으나, 근래에는 공정기술의 발달로 게이트 당 단가가 떨어지고, FPGA 내부에 메모리나 표준 입출력을 위한 인터페이스 회로 등 다양한 특수 기능 블록들을 내장함으로써 단일 칩으로 시스템 레벨의 설계 및 구현을 소화해낼 수 있게 되었고 이에 따라 소량 생산의 내장형 시스템 제품의 부품으로 직접 FPGA가 사용되기도 한다[3].

일반적으로, 넓은 비트 수의 피연산자를 사용하는 논리연산이 병렬적으로 요구되는 응용이나, 응용에 따른 맞춤형 데이타 경로 폭을 필요로 하는 응용 등에서 하드웨어 가속기로서의 FPGA의 사용이 더욱 효과적이다[4]. SEED의 연산 내용들은 주로 정수형 덧셈과 병렬적 수행이 가능한 비트 연산들이기 때문에 FPGA를 이용하면 범용 프로세서를 사용할 때에 비해서 계산 속도의 가속 효과가 뛰어나게 된다.

따라서, 내장형 시스템에 사용될 SEED의 구현대상 기술로 FPGA가 적합하며 FPGA에 가장 효율적인 SEED연산 하드웨어의 설계가 필요하다. 한편, FPGA를 제품의 부품으로 사용하기 위해서는 저가의 FPGA이어야만 의미가 있다. 본 논문에서는 이러한 필요를 염두에 두고 저가의 FPGA를 이용하여 SEED 알고리즘을 최대한 효율적으로 구현하였다. 아울러, 8-비트의 마이크로컨트롤러 또는 UART 등의 주변장치와의 인터페이스를 위하여 8비트와 128비트 간의 데이타 형식 전환을 위한 모듈도 포함시켜서 하나의 칩에 구현이 가능하도록 면적 효율성을 중시한 설계를 제안하였다. 나아가, 제안된 SEED를 개당 3만원 미만의 소비자 가격인 Xilinx사의 Spartan-II FPGA 칩 하나의 65% 정도의 면적만 사용하여 구현하여 하드웨어적 실험으로 동작을 검증함하여 본 제안의 우수성과 정확성을 보였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 착안점 및 기존연구와의 차이점을 제시한다. 3장에서는 SEED 알고리즘을 소개하고 이의 구현에 직접 필요한 하드웨어 구조를 소개하고, 3장에서는 FPGA로 구현시 면적을 최소화하기 위한 하드웨어 구조를 제안한다. 4장에서는 제안된 설계를 실제 FPGA로 구현한 결과 면적과 성능 등에 대한 분석 자료를 제시한다. 5장에서는 결론 및 향후 연구할 사항을 제안하고 마무리한다.

2. 관련 연구

SEED를 하드웨어로의 구현을 제안한 다수의 논문들이 있다. 이들은 다음 몇가지 부류로 나뉠 수 있다. 첫째, SEED의 16개의 라운드를 모두 파이프라인으로 연결하여 구현하는 방식이 있다[5,6]. 그러나, 이 방식은 파이프라인으로 인한 성능 증대의 효과는 있으나, 지나친 면적 요구량으로 FPGA로 구현하기에는 적합하지 않은 방식이다. 둘째는, 이 단점을 보완하기 위해서 파이프라인 방식을 사용하지 않고 한 라운드만 만들거나, 몇 개의 라운드만 파이프라인 방식으로 설계하고 이들을 반복시켜서 여러 클럭에 걸쳐서 한 블록의 데이타를

처리하는 방식이 제안되었다[7-9]. 즉, 하나의 라운드 계산 블록과 한 개의 라운드 키 생성블록을 16번 반복해서 사용하는 방식이 있다. 세째는, 면적 효율성을 더욱 높이기 위해서 라운드 계산 블록과 라운드 키 생성 블록에서 공통으로 사용되는 G 함수 모듈을 하나만 사용하고 이를 양 블록에서 공유하는 방식 등이다[1,10,11].

그러나, 이러한 기존 설계 방식들도 Xilinx의 저가 FPGA에 들어가기에는 여전히 면적이 크다. 본 연구에 의하면, 공유 구조 부분을 확대하고 FPGA의 특성을 잘 활용하면 추가로 면적을 줄일 수 있는 여지가 많이 있다. 본 논문은 기존 설계들보다 더욱 면적을 줄이기 위해서 F함수와 덧셈기에 공히 사용되는 G함수 블록뿐만 아니라 32비트 덧셈기도 전체적으로 단 1개만 사용하여 그 공유도를 더욱 높였다. 이러한 구조는 각 연산에 소요되는 총 클럭의 수를 증가시키지만, 모듈의 형태가 간소하여 클럭의 주기를 짧게함으로써 (주기 x 클럭수)는 여전히 고속의 응용에 충분한 성능을 내도록 조절이 가능하다. 실제 구현 결과 본 설계는 Xilinx Spartan-II 시리즈를 대상으로 구현할 경우에 66Mbps이상의 처리 속도를 낼 수 있음이 분석결과로 증명되었다.

3. 128-bit 블록 암호화 알고리즘 SEED

3.1 SEED의 전체 구조

SEED는 동일한 연산 구조가 여러번 반복되는 Feistel 구조로 이루어져 있으며, 전체가 16-라운드로 구성된다. 그림 1은 SEED의 개괄적 구조를 보여주고 있다. 좌측의 라운드키 생성 부분과 우측의 라운드 계산 부분으로 크게 나누어진다. 각 라운드 계산 부분은 동일한 구조를 가진 데이타 연산부를 가지고 있으며, 각 라운드는 외부로부터의 입력된 128비트 평문 또는 라운드로부터의 출력값 128비트 값과 라운드 키생성부에서 제공되는 64비트 라운드 키값을 입력으로 사용한다. SEED는 대칭키 알고리즘이므로 복호화시에는 동일한 데이타 경로를 사용하되 각 라운드의 키 값을 반대 순서로 적용하면 된다.

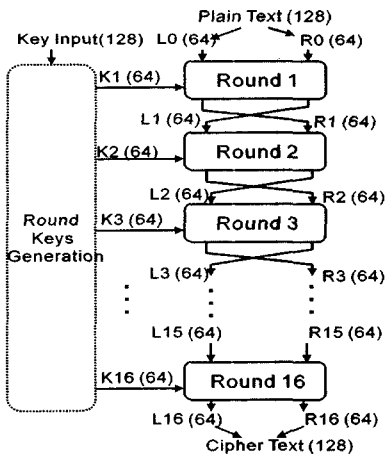


그림 1 SEED의 개괄적 구조

3.2 SEED의 각 라운드 구조

그림 1은 전체 SEED 계산 과정 중에서 처음 2개의 라운드와 라운드 키값 생성부를 한꺼번에 자세히 보여주고 있다. 그림 2의 우측부분은 라운드 계산부이고 좌측부는 라운드 키 생성부이다. 이 그림에서 ⊕는 Exclusive-OR 연산을 의미하고 ⊞는 두 32비트 피연산자 a 와 b에 대한 모듈로 덧셈이다. 즉, a⊞b = (a+b) mod 2^{32}을 의미한다. 각 라운드 i의 입력은 이전 라운드의 2개의 64 비트값(L_{i-1}, R_{i-1})과 2개의 32비트의 라운드 키값(K_{i0}, K_{i1})이며 각 라운드의 출력은 2개의 64비트 값(L_i, R_i)이다. 각 라운드는 1개의 F함수와 64개의 2비트 XOR 게이트로 이루어져 있다. 라운드의 출력 계산식은 $(L_i, R_i) = F(K_{i0}, K_{i1}, R_{i-1}) \oplus L_i$ 이다.

라운드 키값 (K_{i0}, K_{i1})은 하나의 128비트 외부 키값과 라운드별 상수값 KCi로부터 합성해서 만들어진다. 하나의 키값 합성에 사용되는 연산자는 2개의 G함수와 4개의 32비트 mod-2^{32} 덧셈기(뺄셈기도 덧셈기로 구현한다고 가정)이다. 외부 128비트 키값이 정해지면 16개의 라운드 키값이 이에 따라 계산된다.

각 라운드 계산의 핵심연산부인 F 함수는 그림 3에 제시된 바와 같이 라운드 키 값 (K_{i0}, K_{i1})과 이전 라운드로부터의 출력 값(C, D)을 입력받아서 64비트 값(C′, D′)을 출력한다. F함수에서 요구하는 연산자는 XOR와 G함수와 32비트 덧셈기(⊞)이다. G 함수의 구조는 그림 4에 제시된 바와 같다. 이는 32비트 값 입력 (d,c,b,a)를 받아서 32비트 값(d′,c′,b′,a′)을 출력한다. G 함수 모듈에는 2가지 종류의 상수값이 저장된 표 (S_1, S_2 박스)가 각 종류별로 2개가 있고 2-입력 AND 게이트들과 4-입

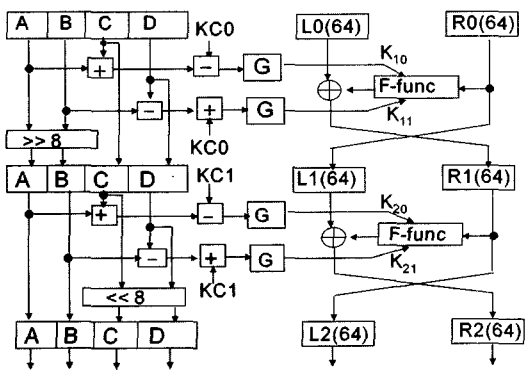


그림 2 SEED의 첫 2 라운드와 키 생성부 상세도

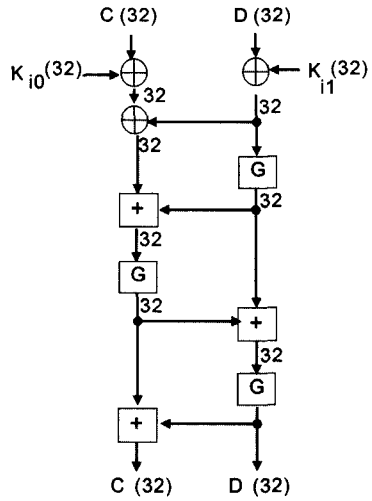


그림 3 F 함수의 구조 (Ki,0과 Ki,1은 라운드키)

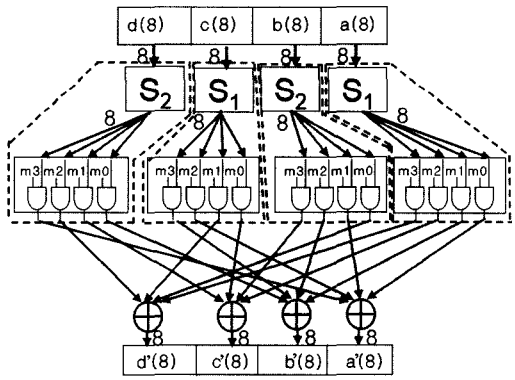


그림 4 G함수의 내부 구조(점선 부분: SS박스)

력 XOR 게이트들이 포함된다. 여기서 S_1과 S_2 박스는 8비트의 입력을 8비트의 값으로 변환시키는 표로서 256*8비트의 ROM으로 구현가능하다. 그림 4에서 점선으로 표시된 부분인 S-박스와 2입력 AND 게이트 4*8개들로 구성된 부분을 합하여 하나의 ROM으로 구현할 수도 있다. 이 256 * 32 비트 크기의 ROM을 SS박스라고 한다. SS박스를 이용한 방식은 G 함수를 구성하는 조합회로의 지연시간을 더 줄여주고 하드웨어 구현을 더욱 간결하게 하는 장점이 있다. 단, 256 * 8비트의 ROM 4개 대신에 256 * 32비트의 ROM 4개가 필요하다는 단점이 있다. 본 논문에서는 SS-ROM 방식을 채택하였다.

4. FPGA를 위한 SEED의 최소 면적 구현을 위한 구조 설계

4.1 하드웨어 면적 요구량 분석

SEED의 하드웨어 구현시의 문제는, G함수와 덧셈기가 차지하는 면적이 매우 크다는 것이다. 하나의 F함수에 G함수가 3개 필요하고 하나의 라운드키 생성에 2개의 G함수가 필요하므로 전체적으로 (3+2) * 16 개의 G함수가 필요하다. 이와 아울러 32비트 modulo-2^{32} 덧셈기가 각 라운드마다 7개씩(빨셈도 덧셈기를 이용하여 구현하므로 각 라운드 키생성 블록마다 4개씩이고 각 라운드마다 3개씩)필요하다. 덧셈기와 ROM 블록을 제외하고는 XOR가 데이타 경로를 이루는 연산자인데, 상대적으로 논리연산자는 면적 부담이 적은 편이다. 따라서, 덧셈기와 ROM 블록의 사용량을 줄이는 것이 결국 전체 칩 면적을 줄이는 관건이다. 특히, FPGA 구조의 특성상, 대용량의 메모리 요구는 곧바로 내장 메모리가 큰 고가의 FPGA 칩의 사용을 요구하게 된다. 아래 표 1은 SEED 구성 요소별로 필요로 하는 주요 하드웨어의 면적 요구량을 구현 방식별로 ROM 블록과 32비트 덧셈기 위주로 비교하여 보여주고 있다.

표 1에 따르면, 전체를 [5,6]과 같이 병렬구조 또는 파이프라인 구조로 구현할 경우에는 적어도 2.5Mbits 이상의 내부 ROM이 필요하며 112개의 32비트 덧셈기가 필요하다. 이는 저가 FPGA로는 구현이 불가능한 면적 요구량이다. 따라서, 본 논문에서는 면적 요구량이

표 1 SEED의 구성 요소별/설계방식별 하드웨어 요구량 비교

	G함수1개 구현에 필요한 양	키생성부에 1단계에 필요한 양	F함수 1개 구현에 필요한 양	전체 병렬구현에 필요한 양 (16라운드+키생성부 16단계)
32비트 덧셈기 갯수	-	4개	3개	112개 ((4+3)*16)
ROM의 크기 (S박스 2개 사용시)	4Kbits (8 * 256 * 2개)	8Kbits (4Kbits * 2)	12Kbits (4Kbits * 3)	320Kbits (4Kbits * 5 * 16)
ROM의 크기 (S박스 4개 사용시)	8Kbits (8*256 * 4개)	16Kbits (8Kbits * 2)	24Kbits (8Kbits * 3)	640Kbits (8Kbits * 5 * 16)
ROM 크기 (SS박스 2개 사용시)	16Kbits (32 * 256 * 2개)	32Kbits (16Kbits * 2)	48Kbits (16Kbits * 3)	1,280Kbits (4Kbits * 5 * 16)
ROM크기 (SS박스 4개사용시)	32Kbits (32 * 256 * 4개)	64Kbits (32Kbits * 2)	96Kbits (32Kbits * 3)	2,560Kbits (4Kbits * 5 * 16)

가장 큰 블록인 덧셈기와 G 함수를 최소한으로 사용하는 방식으로 설계한다.

4.2 면적과 성능 간의 Trade-Off 관계

일반적으로 면적과 성능은 반비례 관계에 있다. 성능과 면적 간의 trade-off 관계를 탐색하는데 고려할 수 있는 선택 사항은 다음 3가지가 있다.

- 선택 1 : 동일한 라운드 계산부를 16개 만들어서 파이프라인 방식으로 설계하는 방식과 하나의 라운드 블록을 설계한 뒤에 이를 16번 반복 실행하는 방식이 있다.
- 선택 2 : 라운드키 생성부와 라운드 계산부를 따로 설계하는 방식과 양자에서 필요로하는 G함수를 공유하는 방식이 있다.
- 선택 3 : 하나의 G함수 설계시에 S박스를 S1과 S2의 종류별로 하나씩만 사용하는 방식(모두 2개의 S박스)과 종류별로 2개씩 사용하는 방식(모두 4개의 S박스)이 있다.
- 선택 4 : F함수에서 사용할 덧셈기와 라운드 키계산에서 사용할 32비트 덧셈기를 따로 사용하는 방식과 하나의 덧셈시를 공유하는 방식이 있다.

위 4가지 선택 사항 외에도 라운드 키 16개를 먼저 생성하고 라운드 계산은 뒤에 하는 방식과 키생성과 라운드 계산을 병렬적으로 수행하는 방식이 있으나, 면적 최소화를 위해서는 키생성과 라운드 계산은 순차적 실행을 항상 선택한다고 가정하자. 위에 열거된 1번째부터 4번째의 선택 사항의 모든 가능한 조합들 중에서 의미있는 조합은 다음 6가지이다. 이들 각 경우에 대해서 키생성을 제외한 암호화/복호화 연산의 throughput과 SEED 핵심부의 면적을 비교하였다. 가장 큰 면적을 요구하는 32비트 모듈러 덧셈기 하나와 S박스 하나 면적을 각각 A와 S로 나타내어 면적의 상대적 크기를 표시하였다.

1. 라운드 키 생성과 라운드 계산을 동시에 하는 방식(16단 파이프 라인식 설계) throughput : 1 클럭 사이클 / 면적 : 16*(7A+12S)
2. 라운드 키 생성과 라운드 계산을 16번 반복하는 실행하는 방식 throughput : 16 클럭 사이클 / 면적 : 7A+12S
3. 라운드 키 생성과 라운드 계산이 하나의 G 함수부를 공유하고 S박스은 4개 사용 throughput : 3*16클럭 사이클 / 면적 : 5A+4S
4. 라운드 키 생성과 라운드 계산이 하나의 G 함수부를 공유하고 S박스은 2개 사용 throughput : 3*2*16클럭 사이클 / 면적 : 5A+2S
5. 라운드 키 생성과 라운드 계산이 G 함수와 덧셈기를 공유하고 S박스은 4개 사용 throughput : 3*16클럭 사이클 / 면적 : A+4S
6. 라운드 키 생성과 라운드 계산이 G 함수와 덧셈기를 공유하는 S박스은 2개 사용 throughput : 3*2*16클럭 사이클 / 면적 : A+2S

대개의 FPGA의 경우, ROM으로 설정 가능한 내장 메모리가 40Kbit이상 제공되는데, 이들은 FPGA 내의 고정된 면적을 차지하는 부분으로서 이를 이용한다고 칩 논리블록 면적 사용량이 추가로 늘어나는 것은 아니다. S박스 하나에 4Kbit를 차지하므로(내장 ROM이 40Kbits인 경우) 10개 이하의 S박스를 사용하는 것은 면적 오버헤드가 없다. 한편, 확장된 SS 박스를 이용할 경우에는 SS박스 하나에 8Kbit가 필요하므로 5개까지는 면적 overhead가 없다. 만일, S박스가 블록 ROM이 아닌 논리블록을 이용하여 구현될 경우에는 S박스 하나당 16K 게이트에 해당하는 면적 요구량을 가진다(SS박스는 32K 게이트 사용). 32비트 덧셈기는 637 게이트에 해당하는 면적을 사용한다. 이 점을 감안하여 각 설계 선택 조합별로 덧셈기와 S박스로 인하여 소요되는 FPGA 논리블록 면적 요구량을 그래프로 표시하면 다음 그림 5와 같다. 그림 5에서 얻을 수는 결론은 5번과 6번 안이 가장 작은 면적을 차지하고, 이 중에서 5번이 더 나은 성능(전체 암호화에 3*16 클럭 사이클 소요)을 가진다는 것이다. 따라서, 본 논문에서는 하나의 덧셈기와 4개의 SS박스를 가지는 G함수를 하나를 사용하는 5번 방식에 의해서 설계를 한다.

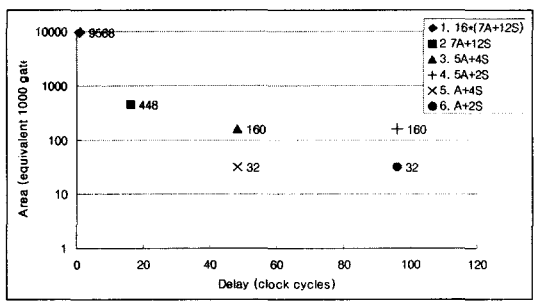


그림 5 여러 가지 설계 방식에 따른 성능과 면적의 관계

4.3 공유블록의 추출 및 공유 구조 설계

면적을 줄이기 위해서 덧셈기와 G함수를 하나씩만 사용하여 라운드 키 계산과 라운드 계산에 공유하는 것이 바람직하다는 결론을 앞서 얻었다. G함수와 덧셈기 공유하는 방식도 여러 가지 구조가 가능하므로 어떤 구조가 SEED 구현에서 가장 효율적인지를 결정하여야 한다. 그림 6은 SEED의 여러 부분에서 덧셈기와 G함수가 사용되는 상황을 보여주고 있다(여기서는 뺄셈기도 덧셈기의 일종으로 간주한다). 그림 6에서 G함수와 덧

셈기의 연결 구조는 유형을 2가지로 분류할 수 있다. F 함수의 경우는 G함수의 출력이 덧셈기의 입력으로 사용되고, 키 생성 블록에서는 역으로 덧셈기의 출력이 G함수의 입력으로 사용되는 구조이다.

F함수와 키 생성 블록의 G함수와 덧셈기의 연결 유형을 모두 한꺼번에 구현할 수 있도록 한 것이 그림 7의 연산부 구조이다. MUX를 이용해서 두가지 유형의 데이타 경로를 구현하도록 한다. 그림 7에서 제어 신호 S가 1이면, 외부 입력 Y가 G함수의 입력 값이 되어 G 함수의 출력 Y'을 만들고 Y'과 외부입력 X가 덧셈기의 입력이 되어 출력 X'이 만들어진다. 제어신호 S가 0이면 외부입력 Y와 X가 덧셈기의 입력값이 되어 덧셈기 출력 X'을 만들고, X'이 G함수의 입력이 되어 출력 Y'이 만들어진다. G함수와 덧셈기 출력부에 있는 플립플롭은 조합형 회로의 순환를 제거하기 위함이다.

그림 7의 블록을 제어신호 S값을 적절히 설정함으로써 라운드 계산과 라운드 키 생성에 모두 이용할 수 있다. 라운드 계산에서는 S=1로 설정한 상태의 데이타 경로(G함수의 출력이 덧셈기의 입력이 되는 경로)를 만들고 3번 반복해서 통과시키면 F함수에서 XOR 게이트를 제외한 나머지 부분의 기능을 수행하게 된다. 라운드 키 계산 시에는 S=0으로 설정한 상태의 데이타 경로를 만들고, 먼저 덧셈기만 이용하여 A+C(또는 B+D)를 수행하여 덧셈기 출력 X'을 저장한다. 다음 클럭 사이클에서 X'과 KCi를 각각 X와 Y에 입력하면 Ki0(또는 Ki1) 출력을 얻게 된다. 이 통합 모듈의 설계시에 덧셈기와 G 함수 사이에 래치를 사용하여 설계를 한다면 Ki0의 계산 시에 클럭 사이클 수를 하나 더 줄일 수 있지만 안정적인 고속 동작을 보장하기 위해서 래치 대신 플립플롭을 메모리 소자로 선택하였다.

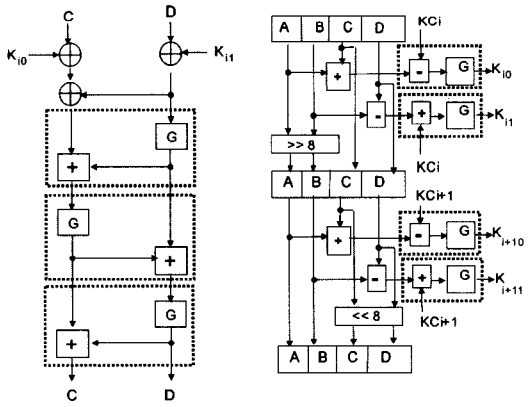


그림 6 F 함수와 키 생성 블록에서 G함수와 덧셈기의 반복 구조 추출

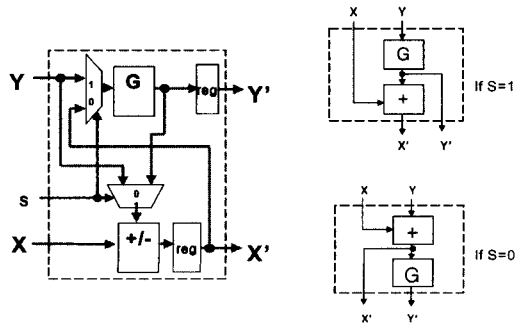


그림 7 공유 블록 구조 : G함수와 덧셈기

4.4 클럭 사이클 수 감소를 위한 레지스터의 조정

라운드 계산부의 레지스터의 위치를 재조정함으로써 사용되는 G함수와 덧셈기의 공유블록을 연결하는 효율적인 구조를 찾을 수 있다. 레지스터의 입출력을 어떻게 배선하느냐에 따라서 연산에 사용되는 클럭 사이클 수가 달라진다. 그림 2의 SEED 라운드 계산부를 직관적인 반복 구조로 변경하여 설계하면 그림 8과 같다. 그림 8의 구조는 매 라운드의 계산 직후에 결과를 R-reg에 저장하도록 되어있기 때문에 각 라운드 계산에 최소 4 클럭 사이클이 필요하다. 구해진 그러나, 굳이 라운드 계산 결과를 R-reg를 경유하지 않고 바로 R(i)로 연결한다면 다음 라운드 계산을 지체하지 않고 바로 시작함으로써 매 라운드마다 한클럭 사이클을 절약할 수 있다. 그림 9는 이러한 점에 착안한 라운드 계산부의 개선된 데이타 경로를 보여주고 있다. 이 개선된 구조는 각 라운드 계산에서 전체적으로 16 클럭 사이클 수를 절약할 수 있다.

5. FPGA를 위한 SEED의 구현

5.1 상위 단계 회로 구조의 개요

그림 10은 본 논문의 핵심인 SEED 모듈의 상위 단

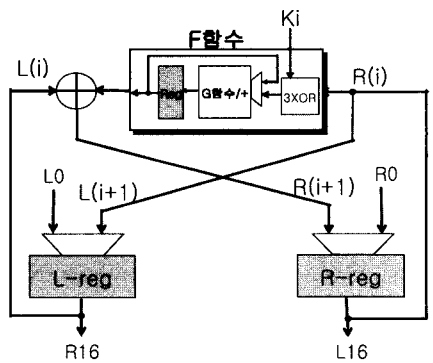


그림 8 직관적 방식의 기존 라운드 계산 데이타 경로

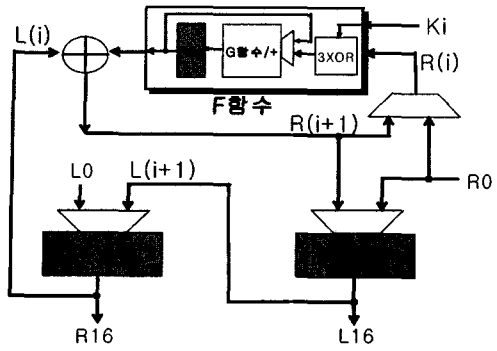


그림 9 클럭 사이클 수를 줄이는 새로운 방식의 라운드 데이타 경로

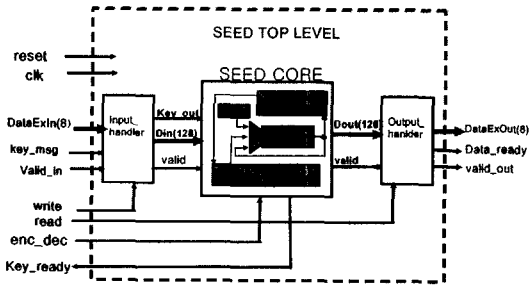


그림 10 상위레벨 구조의 개요

계 구조로서 핵심부와 입출력 변환기 모듈 간의 연결 구조를 보여주고 있다. 본 논문에서 소규모 내장형 시스템을 목표로 하기 때문에 8비트 외부 데이타버스와 연결되는 것을 전제로 하고 있다. 따라서, 8비트 데이타 입출력을 하기 위하여 데이타 폭의 변환을 위한 인터페이스 회로가 필요하다. 그림 10의 좌측에 있는 input_handler 모듈과 우측의 output_handler 모듈이 출력 128비트 데이타와 8비트 데이타 간의 변환부이다. 입력 제어 신호 key_msg에 의해서 입력되는 키/데이타를 구분하고 enc_dec 신호에 의해서 암호화/복호화 동작을 결정한다. SEED 코어 자체는 라운드 키 생성부와 라운드 계산부로 나뉘어지고, 양쪽 모두가 공유하는 G함수와 덧셈기 블록이 있다.

5.2 SEED 핵심부의 내부 데이타 경로 설계

그림 11에서는 제안된 SEED의 핵심부의 데이타 경로 구조를 보여주고 있다. 위쪽 점선 블록은 라운드 키를 계산하는데 사용되는 블록이고, 아래쪽 점선 부분은 라운드 계산시 필요한 블록이다. 가운데 점선 블록은 양쪽에 공통적으로 사용되는 블록이다. 이 그림의 어두운 부분들은 레지스터나 메모리 블록을 의미한다.

공유되는 부분은 그림 7에서 설계한 G함수와 덧셈기

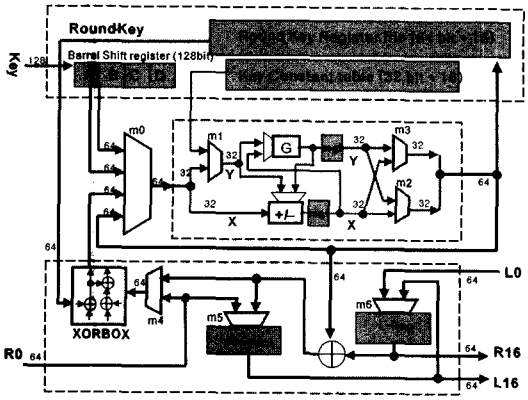


그림 11 제안된 SEED CORE의 내부 구조

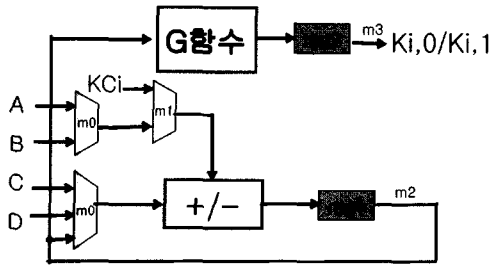


그림 12 라운드 키 값 계산을 위해 활성화되는 경로

의 공유블록과 그 출력과 입력에 멀티플렉서를 부착하여 설계하였다. 연산자의 공유를 위한 멀티플렉서들을 사용하고 있다. 그림에는 직접 나타나 보이지는 않지만 공유블록의 G함수를 위한 폭이 32비트이고 깊이가 256인 SS박스 4개가 사용되었다. 라운드키 계산 전용 부분인 위쪽 점선 부위에는 RAM으로 워드의 폭이 64비트이고 깊이가 16인 라운드키 레지스터 파일이 있고, ROM으로 32비트이고 깊이가 16인 키상수값 표가 있다. 또한, 키값 입력 128비트 레지스터인 ABCD는 AB 혹은 CD 단위로 독립적인 쉬프트 기능을 가진다. 라운드 계산 전용 부분인 아래쪽 점선 분은 2입력 XOR 64개와 2입력 XOR 32개가 3쌍과 각 라운드의 입력값을

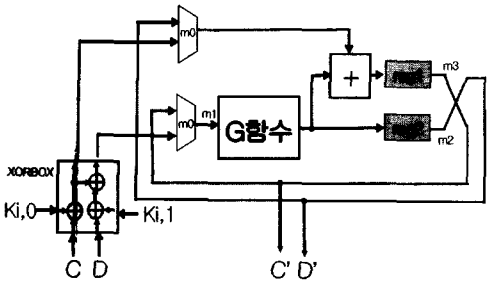


그림 13 라운드 계산을 위하여 활성화되는 경로

저장하는 64비트 레지스터인 L-reg와 R-reg가 있다. 이 R-Reg와 L-Reg레지스터들과 공유블록의 연결 구조는 앞의 그림 9의 구조와 같다. 그림 11의 회로에서 라운드 키 값 계산을 위하여 위쪽 점선과 가운데 공유블록의 활성화되는 데이타 경로만 따로 나타낸 것이 그림 12이다. 그림 13은 그림 11의 회로에서 라운드 계산을 위하여 활성화되는 데이타 경로만을 나타낸 것이다.

5.3 제어부의 설계

그림 10과 그림 11의 데이타 경로를 이용하여 전체 연산 과정을 시간에 따라 나타내면 그림 14와 같다. 그림 14는 하나의 키(128비트)와 데이타 블록(128비트)를 입력받아서 암호화/복화화하는데 필요한 제어스텝을 RTL 단계로 보여주고 있다. 이 그림에서의 굵은 화살표로 표시된 부분은 레지스터에 값을 저장하는 동작을 의미한다. 하나의 제어스텝에 있는 여러 개의 레지스터 할당문들은 병렬적으로 동작한다.

제어 스텝은 크게 키 입력 부분 (8비트를 128비트로 변환), 16 라운드키 생성부분 및 저장부분, 데이타 입력 부분(데이타 8비트를 128비트로 변환), 16 라운드계산 부분 부분, 128비트 데이타를 8비트로 나누어 출력하는 부분으로 나누어진다. 각 라운드 키의 계산 과정의 첫 번째 제어 스텝은 이전 라운드의 키값을 저장하는 동작과 병렬적으로 일어나게 되며 암호화/복호화를 위한 각 라운드 계산부에서도 2번째 라운드부터는 각 라운드의 첫 번째 제어스텝은 이전 라운드의 마지막 연산과 병행해서 일어나기 때문에 필요한 클럭수를 줄일 수 있다. 이 제어부의 설계에 따르면 라운드 키 계산에 필요한 제어 스텝은 98 스텝이며, 라운드 계산에 필요한 제어 스텝의 수는 49 스텝이다.

5.4 FPGA의 특성을 고려한 설계

요즘의 FPGA는 어느 회사의 제품이건 공통적으로 칩 내부에 내장 메모리 영역을 별도로 가지고 있다. 이 내장 메모리는 수십Kbits이상의 값을 저장할 수 있는 RAM이나 ROM으로 사용되어 로직 영역의 자원을 사용하지 않고 표나 레지스터 파일등의 구현을 효과적으로 하도록 지원한다. 따라서, FPGA를 구현 목표로 한 설계라면 이 영역을 충분히 활용하는 것이 면적과 성능 효율면에서 나은 선택이라고 할 수 있다. 본 설계에서도 G함수에 필요한 S박스와 라운드 킷값을 저장하는 레지스터 파일용으로 이 내장 메모리 영역을 활용하여 구현하였기 때문에 저가의 FPGA로 구현이 가능하였다.

MUX의 구현에는 2가지 옵션이 있다. FPGA의 일반 논리블록의 LUT를 이용한 구현방법 외에 내부 tri-state 버퍼를 이용하는 방법이 존재한다. tri-state 버퍼를 사용하는 방식은 논리블록을 사용하지 않기 때문에 역시 면적을 줄일 수있는 반면에 tri-state 버퍼는 칩 전체에 고루 분포하기 때문에 비트수가 큰 mux를 이방식으로 구현할 경우에 데이타 경로가 칩 전체로 분산되

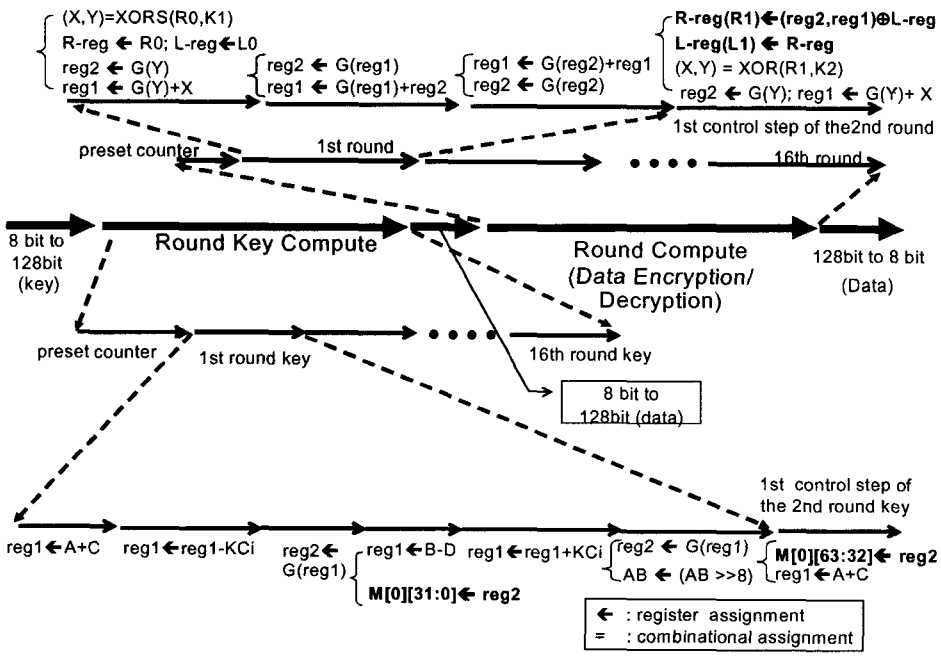


그림 14 SEED 제어부의 설계

어 배선 길이의 증가로 회로 전체의 동작 지연시간이 증가할 수가 있다. FPGA에서 로직 영역을 구현하는 기본 단위인 LUT는 4개의 입력을 가지는 임의의 논리함수를 구현하도록 되어 있기 때문에 4:1 MUX나 2:1 MUX 모두 같은 크기의 면적을 사용한다. 따라서, 본 설계에서는 비교적 비트수가 작은 2:1 MUX들은 tri-state buffer를 이용하였고, 4:1 MUX는 로직 블록을 이용하여 구현하였다.

6. FPGA를 이용한 구현 결과

6.1 기능 시뮬레이션에 의한 검증

제안된 SEED 회로를 기능 레벨에서의 동작을 검증하기 위한 ModelSim-II XE를 이용한 VHDL 시뮬레이션 결과는 아래 그림 15와 그림 16과 같다. 그림 15는 키 생성과정을 보여주고 그림 16은 암호화 과정을 보여준다. 암호키 값을 8비트 단위로 16번에 나누어 입력하고 키생성이 끝났을때 key_ready가 활성화된다. 이 신호에 의해서 평문 128비트를 8비트 단위로 나누어서 읽어들이고 라운드계산을 마치면 data_ready가 활성화된다. 출력 암호문을 외부에서 읽어갈 때 valid_out이 활성화되는 것을 볼 수 있다. 사용된 클럭 수는 라운드 키 생성에 98클럭이 사용되었고, 16 라운드 전체 계산과 결과 저장에 97클럭이 사용되었다. 라운드 계산이 이론적인 계산으로는 3*16+1=49 제어 스텝이 필요하지만, G 함수를 구현한 내장 메모리 블록의 특성상 읽기에 한클럭이 소요되기 때문에 실제로는 (3*2)*16+1=97 클럭이 사용된다. 라운드키 생성과 데이타 변환까지 포함하면, 98(키생성)+97(데이타 생성)+16(입력 변환)+18(출력 변환)=245 클럭이 된다.

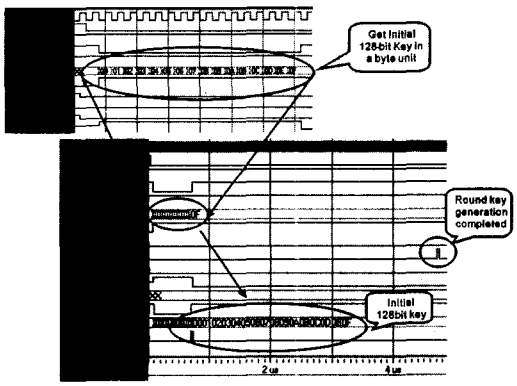


그림 15 시뮬레이션 결과 : 라운드키 입력 및 생성 과정

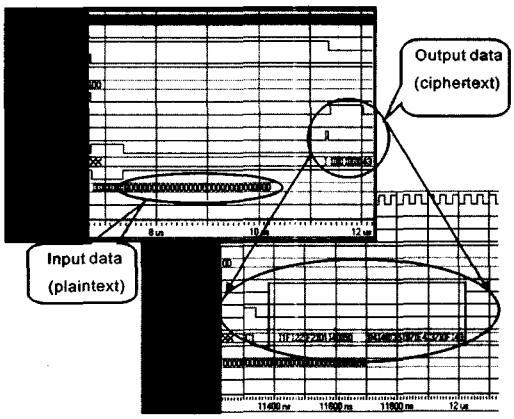


그림 16 시뮬레이션 결과 : 암호화 과정
(128비트 평문 -> 128비트 암호문)

6.2 합성 및 구현 결과

본 논문에서는 제안된 설계를 VHDL로 기술하고 Xilinx ISE 4.2i 환경에서 XST 합성기를 사용하여 FPGA를 대상으로 회로를 구현하였다. 구현 대상 FPGA는 Xilinx의 SPARTAN-II 시리즈의 XC2S100-6PQ208 칩이다.

다음은 Place&Route 과정을 마친 후의 Map Report 파일의 내용의 일부이다. 총 787개의 Slice를 사용하여 65%의 FPGA 논리 블록 면적을 사용하였고, 최대 동작 가능 클럭 속도는 50.569MHz까지 가능하다고 타이밍 분석 결과가 나왔다.

Number of Slices: 787 out of 1,200 65%
Minimum period: 19.775ns (Maximum frequency: 50.569MHz)

암호화시에 대개 하나의 키를 이용해서 대량의 데이타를 암호화와 복호화하는 것이 일반적이므로 키 생성에 걸리는 시간은 제외하고 성능을 따지는 것이 적절한 평가이다. 아울러, 데이타의 변환에 걸리는 시간은 성능 평가시 제외하는 것이 적절하다. 왜냐하면, 여기서의 비교 대상은 암호화 프로세서 엔진 자체의 성능에 관심이 있기 때문이다. 따라서, 성능 계산은 하나의 128비트 블록을 처리하는데 걸리는 시간인 97클럭을 기준으로 하는 것이 타당하다. 즉, 처리 능력은 다음과 같다.

Throughput = 처리양 / 처리시간 = 처리데이타 비트 수/ (클럭주기 * 클럭수)
= 128bit / (19.775 ns * 97) = 66.73Mbps

키 생성에 필요한 클럭수가 98클럭이므로, 이것까지 고려한다면, 최악의 경우 출력은 128비트/(19.775 ns * (97+98)) = 33.19Mbps이다. 그러나, 대개 키 생성은 매우 드물게 이루어진다고 가정할 때, 대체적으로 66.73 Mbps 성능이라고 주장할 수 있다. 이 성능은 소규모 내장형 시스템용으로는 여유가 있는 데이타 전송속도이다. 예컨대, 이 성능은 만약에 SEED를 UART와 통합할 경우에 하나의 SEED 엔진으로 시분할 처리 방식에

의해서 UART의 Rx와 Tx쪽을 모두 암호화/복호화 할 수 있는 성능이다. 우리가 구현에 사용한 FPGA는 Xilinx에서 저가용 FPGA칩으로 판매하고 있는 Spartan-II 시리즈의 XC2S100 칩인데, pipeline 버전의 논문[5]에서 필요로 하는 Xilinx VirtexE 칩의 가격의 1/100 정도에 해당하는 매우 저렴한 가격이다. 이런 저가로 구현하여야만 SEED를 내장형 시스템의 하드웨어 가속기로 쓸 수가 있다.

6.3 기존 방식과의 구현 결과 비교

6.3.1 설계의 구조적 비교

다음 표 2에서는 제안된 방식과 기존의 발표된 방법들 중에서 FPGA를 구현대상으로 한 경우와 비교하였다. [5]는 파이프라인 방식으로 설계하였기 때문에 매 클럭사이클마다 결과가 출력되는 좋은 성능을 보이지만 면적 요구량이 지나치게 커서 저용량 저가의 FPGA로 구현이 불가능하다. [10]은 FPGA 구현을 목표로 하여 면적 최소화를 시도하였다. [10]은 면적을 줄이기 위해서 G함수부분만을 공유하였으나, 본 논문에서는 G 함수뿐만이 아니라 32비트 덧셈기도 하나만을 사용하는 구조를 사용하여 면적을 절약하였다. [10]에서는 S박스의 구현에서 FPGA의 내장 메모리 블록을 사용하지 않았으나, 본 구현은 FPGA 구현이 목표이므로 이 부분을 활용하여 면적을 대폭 줄였다. [10]에서는 S박스를 각 종류별로 1개씩만 사용하여 반복계산을 하였으나 본 논문은 사용되지 않는 칩의 내장 메모리를 충분히 활용하여 S박스를 여러개 구현함으로써 추가 논리 블록의 면적 오버헤드 없이 한 클럭에 G함수의 값을 얻어낼 수 있도록 설계하였다. 더구나, 본 논문에서 제안한 seed 프로세서의 구조는 라운드 계산을 위한 데이타 경로의 레지스터 연결을 효과적으로 함으로써 클럭수를 16 클럭 더 줄일 수 있었다. 결과적으로, 본 구현에서는 면적 오버헤드 없이 [10]에 비해서 연산에 필요한 클럭 사이클의 수를 2.5배 줄일 수 있게 되어 더 나은 성능을 달성할 수 있게 된다.

6.3.2 FPGA 구현 결과 면적과 성능 비교

다음 표에서는 FPGA로 구현한 결과 면적과 성능에 관하여 기존의 2개의 논문에서의 결과와 비교한다. 이 비교표에 의하면 새로운 방식이 [10]의 기존 방식에 비해서 성능이 약 4.4배 우수한 것으로 나왔다. 두 FPGA 간의 특성의 차이 때문에 클럭 속도 차이가 생길 수 있으므로 양자 간의 같은 빠르기의 클럭을 사용하였다고 가정하여도 새로운 방식이 2.4배 가량 우수한 성능임을 알 수 있다. 한편, 파이프 라인 버전으로 구현한 [5]의 방법에서 사용한 FPGA 칩은 본 논문에서 사용한 FPGA에 비해서 100배이상 비싼 칩으로서 성능대비 비용면에서 새로운 방식이 파이프라인 방식에 배해서 나은 것을 알 수 있다.

6.3.3 소프트웨어 구현과의 비교

본 논문에서 목표로 하는 소규모 내장형 시스템에 사용될 수 있는 16비트 마이크로 프로세서인 80196C를 이용하여 SEED 암호화 알고리즘을 구현한 결과와 비교하였다. 한국정보보호진흥원 [2]에서 표준으로 배포되는 SEED 소스코드를 80196C를 대상으로 컴파일한 결과 실행 명령어 크기가 약 32KB 크기가 되었다. S박스를 위한 표를 저장하는 4KB를 여기에 포함하면 SEED 알고리즘을 위해서 최소한 36KB 크기의 메모리가 필요로 하게 된다. 따라서, 코드의 크기가 너무 크기 때문에 다른 작업을 위한 코드에 필요한 기억장소가 확보되지 않을 수 있으므로 사실상 16비트나 8비트 프로세서로는 SEED를 처리하면서 다른 작업을 하기에 문제가 있음을

표 2 기존의 논문과의 설계 방식 비교

	기존방법[10]	기존방법 [5]	제안된 새방식
특징	면적 최소화	성능 최대화 (파이프라인 구현)	면적 최소화
G함수 갯수	1개	80=5*16	1개
덧셈기 갯수	5개	112=7*16	1개
라운드 계산부의 throughput	16*15 clock cycles	1 clock cycles	16*6 clock cycles (제어 스탭은 16*3)

표 3 기존의 논문과의 구현 결과 비교

	기존방법[10]	기존방법 [5]	제안된 새방식
구현 목적	면적 최소화	성능 최대화 (파이프라인 구현)	면적 최소화
사용한 칩	Altera 10K100 (10만 게이트 급)	Xilinx Virtex-EXCV2600E (320만 게이트 급)	Xilinx Spartan-II XC2S100 (10만게이트 급)
면적	칩 면적의 80%	-	칩 면적의 65%
성능	14.9Mbps(28MHz)	-	66.7Mbps(50.5MHz)
클럭 속도를 5.5MHz로 가정했을 경우 성능	26.8Mbps	-	66.7Mbps

알 수 있다.

처리 속도면에서는 16비트 프로세서에서의 컴파일 결과가 32KB이므로 코드가 1 명령어당 1클럭이 소요된다고 해도 최소 16,000 클럭 사이클이 필요하다. 이는, 제안된 FPGA 구현에서에(라운드 계산+라운드키 생성+데이타 길이 변환에 총 245 클럭 사이클)비해서 65배 이상의 성능 차이가 난다. 더구나, FPGA에서 사용하는 클럭의 주파수가 일반적으로 16비트 마이크로프로세서용으로 사용되는 클럭보다 보다 2배이상 빠를 것이므로 그 성능차이는 200배 가량이 될 것이다.

6.4 FPGA 보드를 이용한 검증과 인터페이스 회로 추가

실제로 FPGA를 이용하여 하드웨어적으로 검증을 하기 위해서 그림 17과 같이 FPGA 칩을 이용해서 보드상에서 seed의 동작을 검증하였다. 보드에 연결된 PC로부터 적재된 외부 데이타 메모리에서 암호키와 데이타 블록을 읽어들여서 seed에서 암호화를 실시한 뒤에 다른 외부 데이타 메모리에 값을 쓰는 동작을 한다. 이 외부 데이타 메모리의 값을 보드로부터 읽어들여서 암호화된 결과를 얻을 수 있다. 이 실험을 위해서 FPGA를 통해서 외부 메모리에 데이타를 쓰고 읽을 수 있도록 하는 FPGA 상의 가상 JTAG회로와 병렬포트를 통해서 PC와 통신하는 사용자 인터페이스용 프로그램을 별도로 작성해야 한다. 즉 실험 절차는 SRAM1에 데이타 저장 -> FPGA에 SEED 회로 적재 -> FPGA구동하여 결과를 SRAM2에 저장 -> SRAM2의 데이타를 PC로 읽어들여 결과 확인의 순으로 진행된다.

실제 소규모 내장형 시스템에서 사용하기 편리하도록 설계한 SEED에 8비트 통신모듈로서 대표적인 추가하고 UART와 집적하여 하나의 FPGA 칩으로 만들었다. 최소 면적으로 설계하였기 때문에 저가의 10게이트급 FPGA(XC2S100)를 사용하여도 UART 등의 추가 회로를 포함하고도 칩의 면적이 20%가량 여전히 남았다.

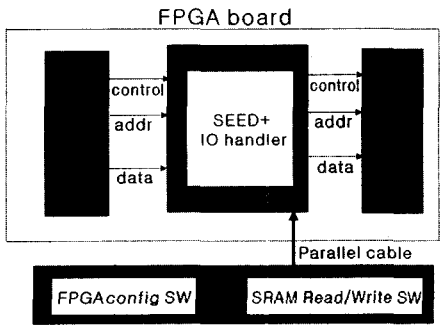


그림 17 FPGA 보드를 이용한 SEED 구현 검증

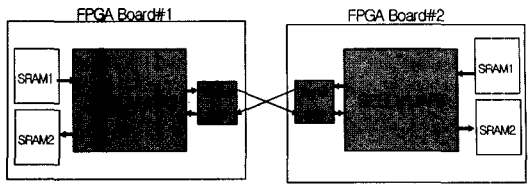


그림 18 FPGA와 UART를 한 칩에 통합하여 동작 검증

UART를 포함한 SEED는 실제 동작 속도에서의 검증을 위하여 사용되었다. 그림 18에서 보이고 있는 실험에서 2개의 보드를 이용하여 직렬 통신으로 양 SEED 간에 암호화와 복호화를 상호 실시간으로 검증하였다.

7. 결 론

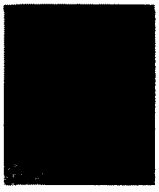
본 논문에서는 저가의 FPGA를 이용한 SEED 블록 암호화 알고리즘의 구현 방식을 제시하였다. 구현 결과는 최대 66Mbps의 처리속도로서 성능 면에서 목표로한 8비트/16비트 환경에서 사용하기에 부족함이 없는 결과를 나타내었고 면적 요구량에 있어서도 8비트와 128비트 변환모듈을 포함해서 10만 시스템 게이트급의 저가의 FPGA 면적의 65%만을 차지하였다. 나머지 면적을 이용하여 UART 등의 다른 회로도 함께 하나의 FPGA 칩으로 구현할 수 있다. 이러한 결과는 [5]의 완전 파이프 라인으로 구현했을 때에 비해서 칩 가격을 100분의 1로 줄일 수 있는 결과가 되었으며 [10]의 구현 결과에 비해서도 더 작은 면적을 사용하면서도 성능은 4.4배 정도 좋다. 본 논문에서 제안된 설계는 실제 저가의 FPGA 보드로 구현하여 회로의 동작성도 하드웨어적으로 검증을 마쳤다. 아울러, 실제적으로 내장형 시스템에 사용하기 쉽도록 UART와 동일 칩에 집적하는 데에도 성공하였다.

이러한 성공적 설계의 원인은 다음 3가지를 꼽을 수 있다. 첫째는, 면적을 많이 차지하는 공통 연산자를 단 한개의 인스턴스만 사용하고 여러 작업에서 최대한 공유하도록 하였고, 둘째는, 핵심부의 데이타 경로에서 레지스터 간의 연결을 조정하여 클럭 사이클 수를 최소화하였다. 마지막으로는, FPGA를 구현 목표로 하기 때문에 FPGA의 특성을 설계에 잘 반영하여 면적을 절약하면서도 성능 저하의 문제를 극복하였다. 이로써 최대한 논리블록의 사용을 억제하여 구조를 단순화함으로써 여러 클럭을 사용하더라도 클럭 주기를 빠르게 함으로써 문제 해결에 필요한 적정 연산 성능을 유지하면서도 면적 요구량을 최소화시킬 수 있었다.

참 고 문 헌

[1] Young-Ho Seo, Jong-Hyeon Kim, Yong-Jin Jung,

and Dong-Wook Kim, "An Area Efficient Implementation of 128-bit Block Cipher, SEED," ITC-CSCC 2000, Korea, 2000. (r).
[2] 한국정보보호진흥원(KISA), "128비트 블록 암호알고리즘 표준", <http://www.kisa.or.kr>
[3] Rob A. Rutenbar, Max Baron, Thomas Daniel, Rajeev Jayaraman, Zvi Or-Bach, Jonathan Rose, Carl Sechen "(When) Will FPGAs Kill ASICs," Proceedings of the 38th conference on Design automation June 2001, pp.321-322.
[4] Miron Abramovici, Jose T. de Sousa, Daniel Saab, "A massively-parallel easily-scalable satisfiability solver using reconfigurable hardware," Proceedings of the 36th ACM/IEEE conference on Design automation conference June 1999.
[5] 엄성용, 이규원, 박선화, "SEED 블록 암호 알고리즘의 파이프라인 하드웨어 설계", 정보과학회 논문지, 시스템 및 이론 제 30권 제 3호, pp.149-159, 2003년 4월.
[6] 엄성용, 이규원, "SEED 블록 암호 알고리즘의 파이프라인 하드웨어 설계에 관한 연구", 한국정보과학회 가을학술발표논문집(III), 2001, 10 pp.43-45.
[7] 채수붕, 김기용, 조용범, "Pipeline 구조의 SEED 암호화 프로세서 구현 및 설계", 대한전자공학회 02 하계종합학술대회 논문집(2) 2002.06, pp.125-128.
[8] 신종호, 강준우, "SEED 블록 암호 알고리즘의 단일칩 연구", 대한전자공학회 하계종합학술대회 논문집, 제23권, 제1호, 한국외국어대학교 전자제어공, pp.165-168, 2000.
[9] 전신우, 정용진, "128비트 SEED 암호화 알고리즘의 고속처리를 위한 하드웨어 구현", 통신정보보호학회논문집, 제 11권, 제 1호, 2001, 2.
[10] 김종현, 서영호, 김동욱, "블록 암호 알고리즘 SEED의 면적 효율성을 고려한 FPGA 구현", 정보과학회논문지, 컴퓨팅의 실제 제7권, 제4호, pp.372-381, 2001년 8월.
[11] 정진욱, 최병윤, "SEED와 TDES 암호 알고리즘을 구현하는 암호 프로세서의 VLSI 설계", 대한전자공학회 하계 종합 학술대회 논문집, 제23권, 제1호, 2000. 6 pp.166-172.



이 강

서울대학교 컴퓨터공학과 학사, 석사, 및 박사(1997.8). 1997년 9월~1998년 2월 서울대학교 컴퓨터신기술 연구소 특별연구원. 1998년 3월~1999년 2월 인제대학교 정보통신공학과 전임강사. 1999년 3월~현재 한동대학교 전산전자공학부 조교수. 관심분야는 VLSI/CAD, Reconfigurable Computing, SOC Design and Verification, Embedded Systems Design

박 예 철

2002년 7월 한동대학교 기계제어시스템공학부 학사. 2002년 8월~2004년 8월 한동대학교 정보통신학과 대학원 석사. 2004년 8월~현재 (주)타오네트웍스 연구원. 관심분야는 VLSI Design, 암호화 알고리즘, 디지털 하드웨어 설계