128-비트 블록 암호화 알고리즘 SEED의 저면적 고성능 하드웨어 구조를 위한 하드웨어 설계 공간 탐색

(A Hardware Design Space Exploration toward Low-Area and High-Performance Architecture for the 128-bit Block Cipher Algorithm SEED)

이 강 †
(Kang Yi)

요 약 본 논문에서는 국내 표준 128비트 블록 암호화 알고리즘인 SEED를 하드웨어로 설계할 경우 면적-성능 간의 trade-off 관계를 보여준다. 본 논문에서 다음 4가지 유형의 설계 구조를 비교한다. (1) Design 1 : 16 라운드 완전 파이프라인 방식, (2) Design 2 : 단일 라운드의 반복 사용 방식 (3) Design 3 : G 함수 공유 및 반복 사용 방식 (4) Design 4 : 단일 라운드 내부 파이프라인 방식. (1),(2),(3)의 방식은 기존의 논문들에서 제안한 각기 다른 설계 방식이며 (4)번 설계 방식이 본 논문에서 새롭게 제안한 설계 방식이다. 본 논문에서 새롭게 제안한 방식은, F 함수 내의 G 함수들을 파이프라인 방식으로 연결하여 면적 요구량을 (2)번에 비해서 늘이지 않으면서도 파이프라인과 공유블록 사용의 효과로 성능을 Design 2와 Design 3보다 높인 설계 방식이다. 본 논문에서 4가지 각기 다른 방식을 각각 실제 하드웨어로 설계하고 FPGA로 구현하여 성능 및 면적 요구량을 비교 분석한다. 실험 분석 결과, 본 논문에서 새로 제안한 F 함수 내부 3단 파이프라인 방식이 Design 1 방식을 제외하고 가장 throughput 이 높다. 제안된 Design 4 가 단위 면적당 출력성능(throughput)면에서 다른 모든 설계 방식에 비해서 최대 2.8배 우수하다. 따라서, 새로이 제안된 SEED 설계가 기존의 설계 방식들에 비해서 면적대비 성능이 가장 효율적이라고 할 수 있다.

키워드 : SEED, 128 비트 대칭형 블록 암호화 알고리즘, 면적과 성능의 trade-offs, 하드웨어 설계 공간 탐색, 내장형 시스템, 암호화 프로세서

Abstract This paper presents the trade-off relationship between area and performance in the hardware design space exploration for the Korean national standard 128-bit block cipher algorithm SEED. In this paper, we compare the following four hardware design types of SEED algorithm : (1) Design 1 that is 16 round fully pipelining approach, (2) Design 2 that is a one round looping approach, (3) Design 3 that is a G function sharing and looping approach, and (4) Design 4 that is one round with internal 3 stage pipelining approach. The Design 1, Design 2, and Design 3 are the existing design approaches while the Design 4 is the newly proposed design in this paper. Our new design employs the pipeline between three G-functions and adders consisting of a F function, which results in the less area requirement than Design 2 and achieves the higher performance than Design 2 and Design 3 due to pipelining and module sharing techniques. We design and implement all the comparing four approaches with real hardware targeting FPGA for the purpose of exact performance and area analysis. The experimental results show that Design 4 has the highest performance except Design 1 which pursues very aggressive parallelism at the expanse of area. Our proposed design (Design 4) shows the best throughput/area ratio among all the alternatives by 2.8 times. Therefore, our new design for SEED is the most efficient design comparing with the existing designs.

Key words : SEED, 128-bit symmetric block cipher algorithm, FPGA, hardware design space exploration, area-performance trade-offs, cryptography processor

· 이 논문은 산업자원부의 IDEC 지원 프로그램의 지원을 받았습니다.
† 종신회원 : 한동대학교 전산전자공학부 교수
yk@handong.edu
논문접수 : 2006년 10월 4일
심사완료 : 2007년 7월 30일

1. 서 론

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[1].

일반적으로 암호화 알고리즘을 전용 하드웨어(칩)으로 구현하면 전용 연산기를 사용하고 특정 응용에 적합한 최적의 구조로 설계되기 때문에 범용 마이크로 프로세서를 이용하는 소프트웨어 구현 방식에 비해서 고성능, 저면적, 저전력 소비의 장점을 가진다. 더욱이, 하드웨어로 구현된 암호 알고리즘은 해커들의 침입에 의해서 암호 시스템이 손실되는 문제를 근본적으로 차단되기 때문에 안정성 면에서도 장점을 가진다[2]. 이러한 장점들 때문에 표준 대칭형 암호 알고리즘인 SEED의 하드웨어 설계 및 구현 방식이 여러 연구들에 의해서 수행되고 그 결과들이 발표되었다.

본 논문에서는 기존 연구 논문들에 의해서 발표된 SEED의 설계 방식들을 세가지로 분류하고 본 논문에서 새롭게 제안된 방식과 면적 및 성능 면에서 비교 수행한다. 정확한 비교를 위해서 FPGA를 이용한 하드웨어로 실제 구현하여 면적 대비 성능의 우수성이 새롭게 제안된 방식이 가장 높다는 것을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 SEED 알고리즘을 소개하고, 3장에서는 기존 논문들의 방식들을 소개한다. 4장에서는 SEED 알고리즘을 위한 네가지 설계 방식을 보이고 이들을 개략적으로 비교한다. 5장에서는 이들 설계를 VHDL로 기술하고 실제 FPGA 칩으로 구현했을때 얻어진 면적과 성능 데이타를 근거로 정량적 비교를 시도하고 6장에서 결론 및 향후 연구 방향을 제시한다.

2. SEED 암호 알고리즘 개요

본 2장은 표준안[1]에 기술된 SEED 자체 알고리즘을 요약 설명한다. SEED는 외부의 128비트 키값과 128비트 원문을 입력받아서 128비트의 암호문을 출력한다. 암호화할 원문은 128비트 단위로 잘라져서 암호문으로 바뀌는 블록 암호화 방식이다. 그림 1에 보여진 바와 같이 SEED 알고리즘은 동일한 연산 구조가 여러번 반복되는 Feistel 구조로서 16-라운드로 구성된다. SEED는 암호화와 복호화에서 같은 키값을 사용하는 대칭키 알고리즘이므로 복호화시에는 암호화와 동일한 데이타 경로를 사용하되 각 라운드의 키 값을 암호화와 반대의 순서로 적용하면 된다.

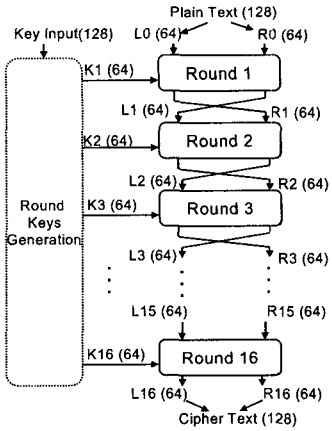

그림 1 SEED의 전반적 구조

그림 2는 전체 SEED 계산 과정 전체 16 라운드 중에서 처음 2개의 라운드 키값 생성부 구조와 라운드 구조 및 F 함수의 내부 구조를 보여주고 있다. 그림 2의 맨 좌측부는 라운드 키 생성부이고 가운데 부분은 라운드 계산부이다. 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 덧셈기(뺄셈기도 덧셈기로 구현한다고 가정)이다. 에서의 모든 덧셈은 모듈로 2^{32} 연산을 의미한다. SEED 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 64 비트 값(L_i, R_i)이다. 각 라운드의 주요 연산자는 1개의 F함수 블록이다. 라운드의 출력 계산식은 L_{i+1}, = R_i, R_{i+1} = $F(K_{i0}, K_{i1}, R_{i-1})$ XOR L_{i-1}이다. F함수에서 요구하는 주요 연산자는 G함수와 32비트 덧셈기이다. F함수와 라운드 키 계산에 사용되는 G 함수의 내부 구조는 그림 3에 제시된 바와 같다. G 함수 모듈에는 2가지 종류의 상수값이 저장된 표(S_1, S_2 박스)가 각 종류별로 2개가 있고 2-입력 AND 게이트들과 4-입력 XOR 게이트들이 포함된다. 여기서 S1과 S2 박스는 8비트의 입력을 8비트의 값으로 변환시키는 표로서 256*8비트의 ROM으로 구현가능하다. 그림 3에서 점선으로 표시된 부분인 8비트 입력과 32 비트 출력으로 구성된 블록을 (256 * 32 비트 크기의 ROM)을 SS박스라고 한다. SS박스를 이용한 방식은 G 함수를 구성하는 조합회로의 지연시간을 더 줄여주고 하드웨어 구현을 더욱 간결하게 하는 장점이 있다.

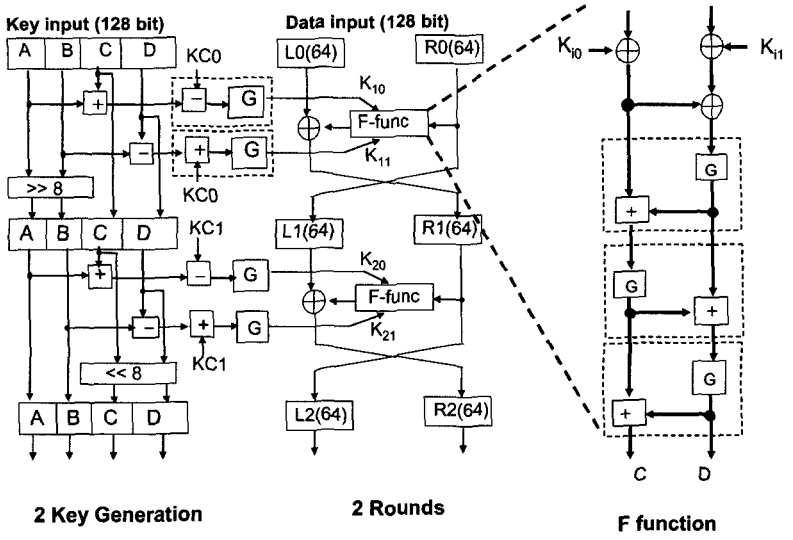

그림 2 첫 2개의 키생성부와 라운드 계산 및 F 함수 내부 구조

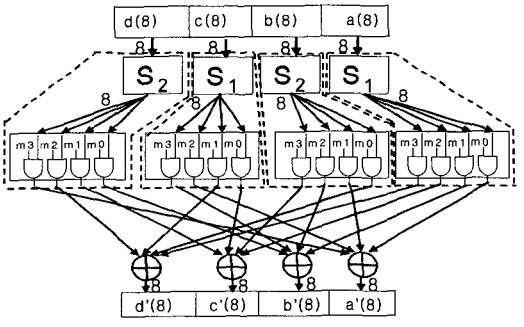

그림 3 G 함수 내부 구조 (점선은 SS 박스)

3. 관련 연구

SEED를 하드웨어로의 구현을 제안한 다수의 논문들이 있다. 다수의 논문들은 성능 보다는 면적을 줄이는데 초점을 맞추고 있다. 왜냐하면 SEED의 제안서에 있는 구조도는 칩면적을 많이 차지하기 때문이다. 기존의 논문들은 3가지로 분류될 수 있다. 있다. 첫째는, 표준안 문서에 나온 SEED의 구조(그림 1)을 그대로 본따서 SEED의 16개의 라운드를 모두 파이프라인으로 연결하여 구현하는 방식이 있다[5]. 그러나, 이 방식은 파이프라인으로 인한 최대의 성능을 낼 수는 있으나, 지나치게 큰 면적을 요구하는 단점이 있다. 둘째는, 면적을 줄이기 위해서 파이프라인 방식을 사용하지 않고 한 라운드만 만들어서 반복적으로 루프를 라운드만 파이프라인 방식으로 설계하고 여러 클럭에 걸쳐서 반복 실행시켜서 한 블록씩 데이타를 순차적으로 처리하는 방식이다[8-10]. 즉, 하나의 라운드 계산 블록과 한 개의 라운드 키 생성블록을 매 평문마다 16번씩 반복해서 사용하는 방식이 있다. 이 방식의 변형으로는 여러 라운드를 파이프 라인으로 만들어서 면적을 늘이고 성능을 조금 더 높이는 방식이 있다. 세째는, 면적을 더욱 줄이기 위해서 라운드 계산 블록과 라운드 키 생성블록에서 공통으로 사용되는 G 함수 모듈을 하나만 사용하고 이를 양 블록에서 공유하는 방식 등이다[2,9,10]. 한편, 세번째 방법보다 면적을 더욱 줄이기 위해서 G 함수 내의 덧셈기를 하나만 사용하는 변형들이 존재한다[11].

본 논문의 다음 장에서는 기존의 세가지 유형의 방식들의 장점을 모두 결합한 새로운 구조를 제안하고 기존 방식들과의 성능 및 면적을 비교한다. 새로운 방식은 두번째 방식과 같이 하나의 라운드를 구현하여 반복 사용하는 방식을 사용면서도 세번째 방식과 같이 키생성과 라운드 블록을 별도로 만들지 않고 자원 공유를 통해서 하나의 연산 블록으로 구현했으며, F 함수 내를 파이프라인을 구축하여 면적 대비 성능을 최대로 높일 수 있는 방식이다.

4. 네 가지 다른 SEED 설계 방식

본 장에서는 여러 방식들간의 객관적 비교를 목적으로 기존 방식들을 최대한 면적 효율적으로 재설계한다. 단, 성능을 떨어뜨리지 않는 한도 내에서 최대한 면적을 줄이는 변형을 시도한다.

4.1 *Design 1* : 16 라운드 완전 파이프라인 방식

그림 4는 16 라운드를 완전한 파이프 라인으로 연결하여 설계한 구조도이다. SEED 알고리즘 자체 특성이 동일한 라운드 구조가 16번 반복 사용되면서 연산이 이

루어지기 때문에 라운드 간을 파이프라인을 하는 것은 매우 자연스럽다. 이 방식은 고성능만을 목적으로 한 설계 방식으로서, 면적 요구량이 비현실적으로 큰 것이 단점이다. 논문[5]에서는 키생성부와 라운드 계산부를 모두 16단 파이프라인으로 설계했으나, 그림 4에서는 라운드부만 파이프라인 설계 방식을 따르고 있다. 키 생성부는 하나의 128비트 키값이 정해지면 처음 한번만 실행하여 16개의 라운드 키값 쌍을 생성한 뒤에 이 라운드 키값들을 별도의 레지스터에 저장해서 차후의 데이타 블록들에 대해서 몇번이고 반복 사용하면 된다.

이 완전 파이프라인 구조를 위해서 필요한 주요 연산기의 개수는 다음과 같이 계산된다.

# of G-functions
= (# of Rounds × # of G-functions/round) + (# of G-functions in Key Generator)
= 16 × 3 + 2 = 50,
# of 32 bits modular adders =
(# of Rounds × # of 32 bits modular adders/round) +
(# of 32 bits modular adders in a key generator)
= 16 * 3 + 4 = 52

Design 1의 구조를 사용한다면, 처음 16 클럭 사이클 이후로는 매 클럭마다 한 128비트 데이타 블록씩을 암호화할 수 있다. 즉, 한 클럭당 128비트를 처리하는 성능을 가진다고 볼 수 있다.

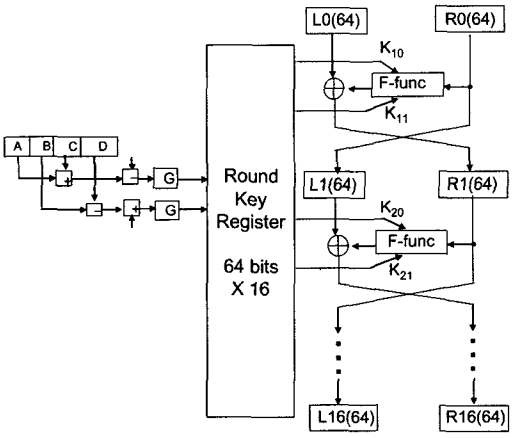

그림 4 16라운드 완전 파이프 라인 방식 구조

4.2 Design 2 : 단일 라운드의 반복 사용 방식

Design 2는 하드웨어 구현에 필요한 면적을 대폭 줄이기 위해서 Design 1과는 달리 라운드 연산부를 16개를 사용하지 않고 하나의 라운드 연산부만 구현한 뒤 이 블록을 16회 반복 사용함으로써 하나의 입력에 대한 연산을 수행한다. 키값 생성부는 Design 1과 동일하게 하나의 라운드키만 생성하는 단순 구조를 취한다고 가정한다. 그림 5는 Design 2의 개략적 구조를 보여주고 있다. 단일의 라운드 연산부만 존재하므로 하나의 F 함수만을 사용하는 구조라고 할 수도 있다.

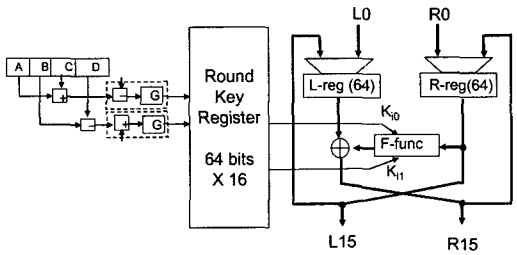

그림 5 단일 라운드 연산부의 반복 사용 구조

Design 2 방식을 사용할 경우 필요한 주요 연산기의 개수는 다음식과 같이 계산된다.

# of G-functions = (# of G-functions per round)
+ (# of G-functions in the key generator)
= 3 + 2 = 5
# of 32 bits modular adders = (# of the 32-bits modular adder in the one round)
+ (# of the 32-bits modular adder in Key Generator)
= 3 + 4 = 7

결국, Design 1에 비해서 G함수 갯수는 10%로, 32비트 덧셈기 갯수는 13.4% 이하로 줄었다. 한편, 면적이 줄어든 반면 성능은 떨어진다. 키 생성부분을 제외하더라도 16클럭 사이클이 한 데이타 블록의 암호화에 필요하다. 즉, Design 2의 평균 출력 성능은 한클럭 사이클당 1/16 * 128 비트 = 8 비트 이하의 성능이 된다. 즉 Design 2의 성능은 Design1 에 비해서 6.25% 수준으로 떨어져서, 결과적으로 Design 2는 면적의 이득에 비해서 성능 손실이 더 심한 것을 볼 수 있다.

4.3 Design 3 : G함수 공유 및 반복 사용 방식

Design 3은 공유할 수 있는 연산 모듈을 더 많이 찾아내어 Design 2의 면적을 더욱 줄인 설계이다. 그림 6에서 보인 바와 같이 면적을 줄이기 위해서 F 함수내의 여러 개 존재하는 G 함수와 키생성부에 있는 G 함수를 하나만 사용하고 두 양쪽에서 공유하는 방식이 제안되었다[9]. 이에 한발 더 나아가서 F 함수와 키생성기 내에 각각 존재하는 32비트 덧셈기도 하나만 사용하여 공유하는 설계가 제안되었다[11]. 아래 그림 6은 모든 연

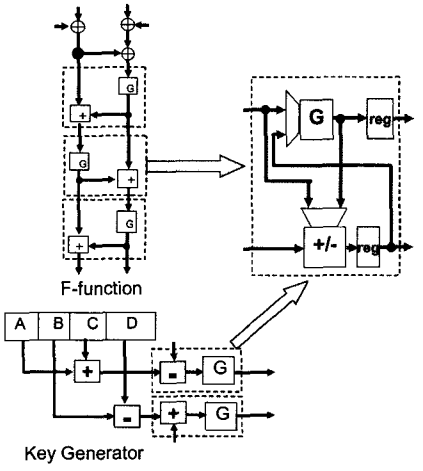

그림 6 F 함수와 키생성기의 공유블록 추출 과정 (G함수와 32비트 덧셈기)

산블록에서 덧셈기와 G함수를 공유하기 위해서 공유 구조를 고안하는 과정을 보여주고 있다. 그림 6의 공유 블록을 이용하여 SEED를 설계한 도면이 그림 7에 나타내 보인다. 이 방식을 이용하면 G 함수와 덧셈기를 각각 1개씩만 사용하면 되므로, Design 2에 비해서 G함수 갯수는 20%, 덧셈기 갯수는 14.2%만 필요로 하게 된다.

그림 7의 Design 3의 SEED 구조는 상,중,하 연산 블록으로 나뉘어진다. Design 3은 키를 생성할 때는 상위 연산 블록과 가운데 연산 블록을 사용하여 계산을 수행하고 그 결과를 상위 블록의 라운드키 레지스터에 저장한다. 데이타 암호화시에는 가운데 연산 블록과 하위 연산 블록을 16회 반복 사용하여 한 입력 데이타 블록의 처리를 마친다.

Design 3을 사용할 경우 [11]에 따르면, 라운드 키생성을 제외하고 128비트의 한 블록을 처리하는데 49 클럭 사이클이 필요하다. 즉, Design 3은 한 클럭당 128비트/49 클럭 = 2.6 비트 정도를 처리하는 성능을 가지고 있다. 이 방식은 면적은 최소한으로 줄이는 장점이 있으나, 성능이 너무 떨어지는 약점이 있다.

4.4 Design 4 : 단일 라운드 내부 파이프라인 방식

Design 4는 Design 2와 Design 3의 혼합 절충형 설계이다. Design 2와 같이 하나의 라운드를 16회 반복 사용하는 방식을 취하지만, Design 3과 같이 키생성 블록과 라운드 연산부를 공유하는 방식을 취하기 때문에 면적 면에서 Design 2 보다는 작으나 Design 3보다는 크다. 성능을 높이기 위한 방안으로 그림 8에서 보인 바와 같이 라운드 연산부의 F 함수 내부를 파이프 라인으로 설계한다. 그림 8에서 파이프라인을 위해서 추가된 레지스터는 파이프라인 단계 간을 분리시키는 3개의 64비트 파이프라인 레지터스 외에 라운드의 L(i+1)와 R(i+1) 출력값을 저장하는 레지스터도 각각 2개씩 더 추가되어있다. 3 단 쉬프트 레지스터로 만들어진 L(i+1)와 R(i+1) 출력 저장용 레지스터는, i-번째 라운드에서 계산된 L(i+1)과 R(i+1) 값이 정확히 3클럭 뒤에 i+1 번째 라운드에서 각각 L(i)와 R(i)로 입력으로 공급되게 한다. 이로써 3쌍의 128비트 데이타 블록의 계산이 한

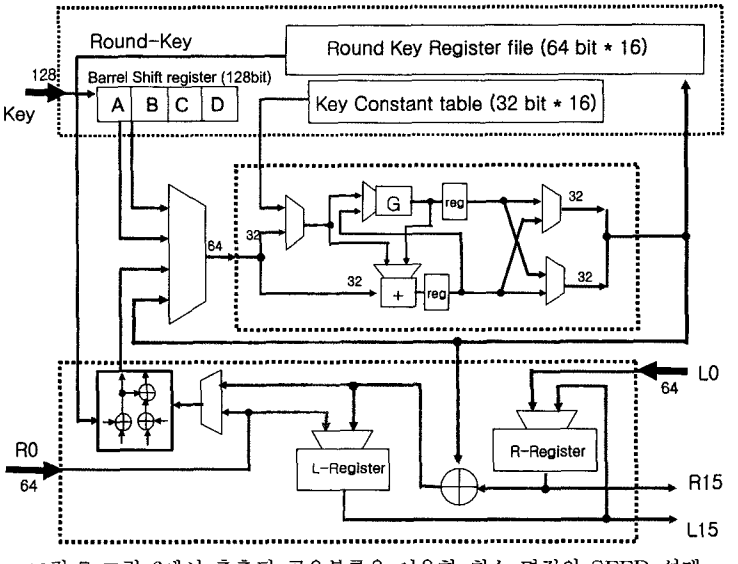

그림 7 그림 6에서 추출된 공유블록을 이용한 최소 면적의 SEED 설계

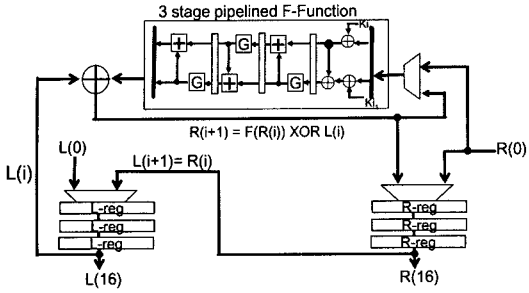

그림 8 라운드 계산부의 F함수 내의 3단 파이프라인 레지스터 삽입

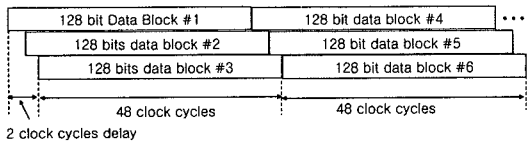

그림 9 Design 4의 데이타 처리 타이밍도 (평균 출력 : 16클럭/블록)

라운드에서 다음 라운드로 지연 시간 없이 원활히 진행될 수 있도록 한다. 라운드 계산 내부가 3단 파이프라인이므로 하나의 데이타 블록 입력에 대해서 한 라운드를 계산하는데 3 클럭 사이클이 필요하다. 그러나, 한번에 3 개의 데이타 블록을 F 함수 파이프 라인에서 동시에 처리할 수 있기 때문에 Design 2와 클럭 사이클 수 면에서 동등하게 된다. 이 타이밍에 대한 설명이 그림 9에서 제시되고 있다. 이 파형의 요점은, 3개의 데이타 블록이 48클럭에 계산이 완료됨으로써 실효적으로 16클럭에 한 블록이 계산되는 효과를 낼 수 있다는 것이다.

라운드 계산 뿐만이 아니라 키생성부까지 하나의 구조로 통합하기 위하여 그림 8의 파이프라인 모양을 수정하여 그림 10에 제시한다. 이 수정된 파이프라인 구조의 이해를 돕기 위해서 그림 12에 변형된 F 함수 구조의 각부분이 어떻게 사용되는지를 보여주고 있다. 그림 11과 그림 12는 그림 10의 구조에서 키생성과 라운드 계산시에 활성화 되는 데이타 경로를 각각 짙은 색으로 표기하여 보여주고 있다.

그림 13은 Design 4의 전체 SEED 설계의 데이타 경로 구조를 보여주고 있다. Design 3과 유사한 구조로 보이는데, 주요한 차이점은 라운드 연산 및 키 생성을 위한 공유연산부가 파이프 라인 구조로 되어 있고, G 함수와 덧셈기를 3개씩 사용하여 3개의 데이타 블록이 동시에 처리될 수 있도록 한 점이다. 아울러, 이 파이프라인 구조를 원활히 사용하여 매 라운드별 계산이 지연시간없이 이어지도록 라운드 레지스터가 2쌍의 64비트 * 3단 쉬프트 레지트터로 구현되어 제공되는 점이 Desig3

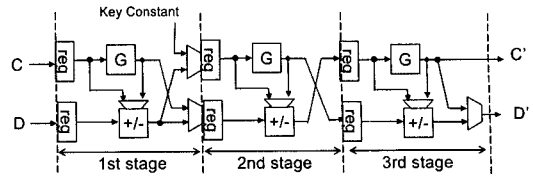

그림 10 라운드 키생성 블록과 라운드 연산블록을 공유한 파이프라인 상세 구조

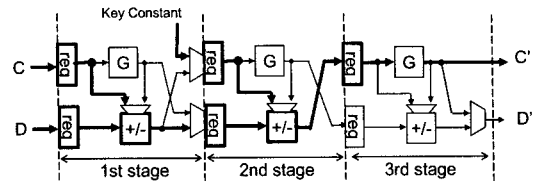

그림 11 키 생성기로 사용될 때 활성화되는 공유 연산기 내의 데이타 경로

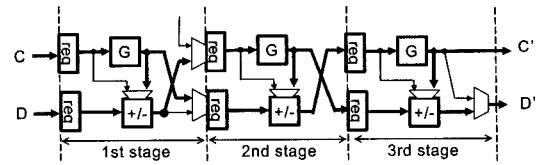

그림 12 라운드 연산기로 사용될 때 활성화되는 공유 연산기 내의 데이타 경로

과의 주요 차이점이다. 그림 13의 맨 위의 점선으로 둘러싸인 부분은 라운드키 생성을 위한 상수값 저장 ROM과 16쌍의 라운드 키값이 저장되는 레지스터 파일을 위한 부분이다. 그림 13의 가운데 부분은 공유 연산 블록이고, 맨아래 부분은 라운드 계산시에만 사용되는 3개의 라운드 계산 결과를 저장할 수 있는 레지스터 등의 데이타경로 부분이다.

그림 14는 위의 파이프라인 구조에서 라운드키 값 계산이 이루어지는 최적의 타이밍을 보여준다. 그림 11의 데이타 경로를 따라서 라운드 1과 라운드 2를 위한 키값이 계산되는 과정을 보여주고 있다. 아래 그림에서 어두운 색으로 보이는 부분은 Ki0을 계산하는 부분이고 밝은 부분은 Ki1을 계산하는 부분이다. 처음 2 클럭 사이클 지연 후에 매 클럭마다 라운드 키값인 Ki0 또는 Ki1이 출력되어 라운드키 저장 레지스터 파일에 보관된다.따라서, 전체 16쌍의 라운드 키값을 계산하는데는 다음식에 의해서 34클럭 사이클이면 된다.

라운드수 * 키갯수/라운드 * 클럭사이클/키 + 최초 파이프라인 지연시간
= 16*2*1 클럭사이클 + 2 클럭사이클 = 34 클럭 사이클

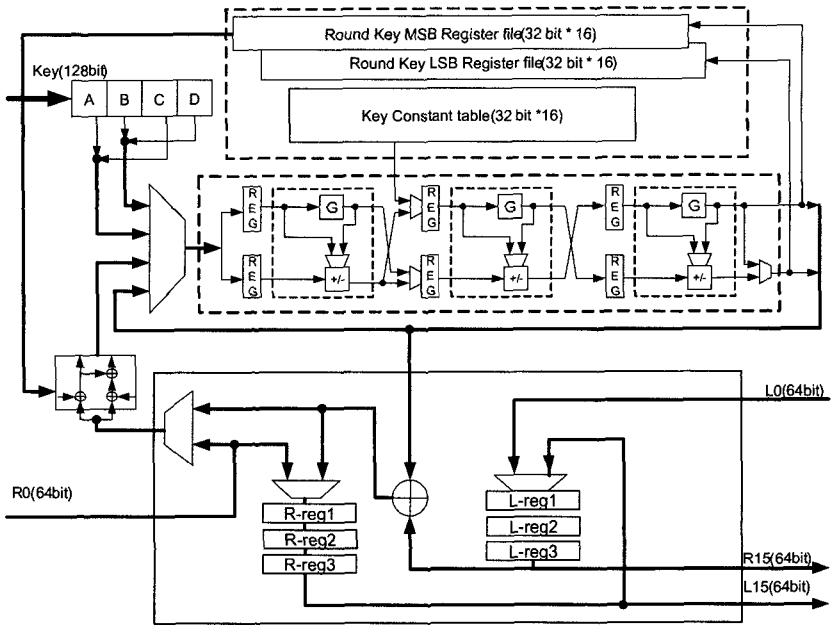

그림 13 라운드 내에 파이프라인 구조를 가지고 키생성부와 라운드 연산부가 공유된 Design 4의 상세 설계

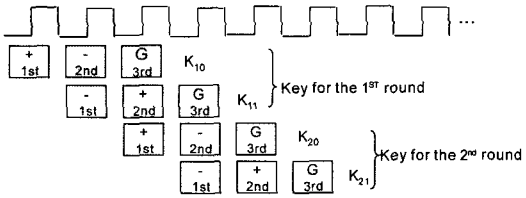

그림 14 파이프라인 구조에서 라운드키 계산의 타이밍도

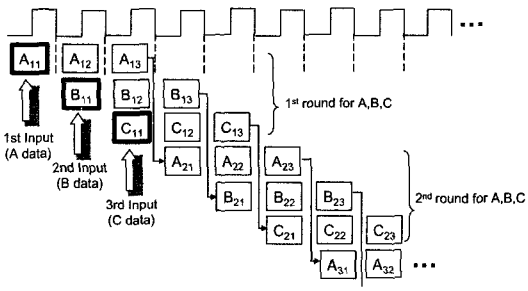

그림 15 라운드 내부 파이프 라인 구조에서 3개의 데이타블록 라운드1과 라운드2 계산의 타이밍도 (Aij, Bij, Cij는 각각 A, B, C 데이타 블록의 i번째 라운드의 j번째 파이프라인 단의 계산을 의미함)

그림 15는 3개의 128비트 데이타 블록 A,B,C의 암호화를 위한 라운드1과 라운드 2의 계산 과정을 보여주고 있다. 이 타이밍도는 그림 9의 개략적 타이밍도의 일부를 상세히 나타낸 것이다. 이 그림에서 Aij, Bij, Cij는 I 라운드의 j 번째 파이프라인 단에서 A,B,C 데이타 블록이 처리되고 있음을 의미한다. 파이프라인이 3단이고 16 라운드까지 계산해야 하므로, j는 1에서 3까지 범위를 가지고 i는 1에서 16까지 범위를 가진다. 개별 데이타 블록의 전체 처리 시간은 16라운드 * 3 클럭 = 48 클럭 사이클이 필요하지만, 3개 데이타 블록이 파이프라인을 통해서 동시에 암호화가 진행되므로 충분히 많은 수의 데이타 블록을 처리한다고 가정하면 평균 데이타 블록당 16 클럭이 걸린다. 즉, Design2와 같이 클럭 사이클당 8비트의 처리 성능을 가진다. 한편, 덧셈기와 G 함수의 요구양은 Design2에 비해서 60%와 23% 정도만 필요하다.

4.4 설계방식들 간의 구조적 비교

표 1에는 4가지 방식에 대한 요약이 제시되어있다. 이 표에서 각 라운드 계산에 필요한 사이클의 수는 하나의 키에 대해서 충분히 많은 양의 데이타의 암호화가 동일 키에 대해서 수행된다고 가정하에서 키생성 시간을 제외한 라운드 계산만의 평균시간이다. 덧셈기 개수와 G 함수 개수, 레지스터 개수 데이타 난의 괄호 안의 숫자는 Design1의 개수를 1.0으로 기준으로 하였을 때 각 설계 방식에서의 상대적인 수치이다.

여기서 모든 경우에 클럭의 주기가 같다고 가정하고 성능을 비교하였으나 실제는 파이프라인 등의 효과로 클럭 사이클의 주기가 설계방식마다 차이를 보일 것이므로 정확한 성능비교는 못된다. 이 표의 면적과 관련된

표 1 여러 설계 방식의 성능 및 면적 비교

설계명	주요 특징	# of clock cycles per Round	Throughput (round key 생성부 제외)	# of adders	# of G function.	# of 64bit data register
Design 1	Full 16 stage pipeline	1	128 bits/cycle (1.00)	52 (1.00)	50 (1.00)	52 (1.00)
Design 2	single round loop	16	8 bits/cycle (0.06)	7 (0.13)	5 (0.10)	20 (0.38)
Design 3	G function sharing	49	2.6 bits/cycle (0.02)	1 (0.02)	1 (0.02)	21 (0.40)
Design 4	Single round with pipeline	16	8 bits/cycle (0.06)	3 (0.06)	3 (0.06)	27 (0.52)

데이타는 제어부를 제외한 데이타 경로만 비교한 데이타이므로 상대적인 비교 값이 실제 합성결과의 전체 면적 비율과는 조금 다를 수 있다. 따라서, 다음 장에서 실제 FPGA 칩을 대상으로 구현 및 검증한 결과를 토대로 실제 성능과 면적을 비교한다.

5. FPGA 칩에 의한 구현 결과 비교

정확한 성능 및 면적 비교를 위해서 FPGA를 대상으로 앞서 제시된 모든 설계 방식들을 실제 칩으로 구현하였다. 대상 칩은 Xilinx 사의 Virtex2 계열인 xc2v2000(Design 1은 xc2c10000)으로 한다. 합성툴은 Xilinx ISE v5.2에서 제공하는 XST 합성기를 사용하였다. 합성결과 얻어진 성능 및 면적에 대한 비교가 그림 16에 제시되어 있다. 아래 그림에서 Throughput은 다음 식에 의해서 계산된 값이다.

Performance(Throughput) =
128 bits * (clock frequency / the # of clock cycles per round computation)

그림 16에는 4가지 설계 방식을 각기 칩으로 구현했을 경우 성능과 면적 요구량을 분포도로 보여주고 있다. 이 분표도 그래프에서 알 수 있듯이, Design 1은 면적 요구량과 성능이 모두 높고, Design 2와 Design 3은 면적 요구량이 줄어드는데 비례해서 성능이 줄어든다. 그러나, 본 논문에서 새롭게 제안한 Design 4는 면적 요구량은 Design2와 Design 3의 중간치이지만 성능은 양자의 중간이 아닌 양자보다 뛰어난 특성을 보인다. 그림 17은 Design 1을 1.0으로 기준으로 하여 모든 설계 방식들의 성능(throughput), 면적 (FPGA slice 개수), 단일 면적당 성능(throughput/slice 개수)를 표시하고 있다. Design 4의 면적 당 성능비가 최대 2.8배(Design 2과 비교)에서 최소 2.4배(Design 1과 비교)까지 우수함을 알 수 있다.

6. 결 론

본 논문에서는 대한민국 표준 블록 암호화 알고리즘인 SEED의 다양한 설계 방식들을 비교하였다. 비교 대상 설계는 기존의 극단적인 설계 방식과 절충형의 설계 방식을 변형한 3가지의 설계와 본 논문에서 제안한 한 가지 설계로서 모두 다음의 4가지이다. (1) Design 1: 16개의 라운드를 모두 파이프라인으로 연결한 최대 성능과 최대 면적요구 방식, (2) Design 2: 1개의 라운드를 반복해서 사용하는 방식 (3) Design 3: 라운드 계산부와 키 생성부를 통합하여 하나의 G함수와 하나의 덧셈기만 반복해서 사용하는 방식, (4) Design 4: 본 논문의 새로운 설계로서 라운드 계산부와 키 생성부를 통합하고 G 함수와 덧셈기를 각각 3개씩 사용하여 F 함수의 내부 파이프라인으로 연결한 방식이다. 구조적 분석에 의하면 Design 4가 면적 요구량은 Design 2와

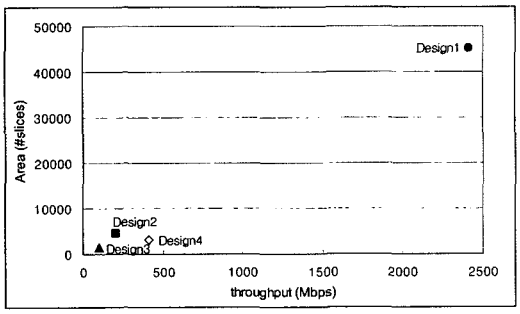

그림 16 4가지 설계의 합성 결과들의 면적과 성능비교

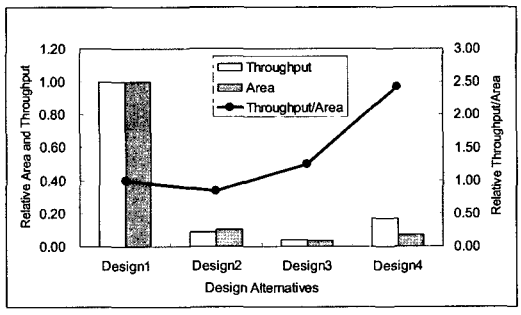

그림 17 Design 1을 기준으로 한 성능과 면적 및 성능/면적 비교

Design 3의 중간쯤이고 성능은 Design 2,3보다 나을 것으로 예측되었다. FPGA칩으로 회로를 합성한 결과는 이와 같은 구조적 분석을 뒷받침한다. Design 4는 네가지 설계 방식 중에서 성능/면적 비가 가장 우수하다.

참 고 문 헌

[1] 한국정보보호진흥원(KISA), "128비트 블록 암호알고리즘 표준", <http://www.kisa.or.kr>

[2] 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.

[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 제30권 제3호, pp. 149-159, 2003년 4월.

[6] 채수봉, 김기용, 조용범, "Pipeline 구조의 SEED 암호화 프로세서 구현 및 설계", 대한전자공학회 02 하계종합학술대회 논문집(2), 2002.06 pp. 125-128.

[7] 신종호, 강준우, "SEED 블록 암호 알고리즘의 단일칩 연구", 대한전자공학회 하계종합학술대회 논문집, 제23권, 제1호, pp. 165-168, 2000년.

[8] 전신우, 정용진, "128비트 SEED 암호화 알고리즘의 고속처리를 위한 하드웨어 구현", 통신정보보호학회논문집, 제11권, 제1호, 2001년, 2월.

[9] 김종현, 서영호, 김동욱 "블록 암호 알고리즘 SEED의 면적 효율성을 고려한 FPGA 구현", 정보과학회논문지, 컴퓨팅의 실제 제7권, 제4호, pp. 372-381, 2001년 8월.

[10] 정진욱, 최병윤, "SEED와 TDES 암호 알고리즘을 구현하는 암호 프로세서의 VLSI 설계", 대한전자공학회 하계 종합 학술대회 논문집, 제23권, 제1호, pp. 166-172, 2000년, 6월.

[11] 이강, 박예철, "내장형 시스템을 위한 128비트 블록 암호화 알고리즘의 저비용 FPGA를 이용한 설계 및 구현", 한국정보과학회 논문지 : 시스템 및 이론(Journal of KISS : Computer Systems and Theory), Vol. 31, No.7.8, pp. 402-413, 2004년, 8월.

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