한국표준 해쉬함수 HAS-160의 하드웨어 구현

서영호[0] 김종현 김왕현 김동욱
광운대학교 전자재료공학과
Intis.com Inc.
axl@explore.kwangwoon.ac.kr

Hardware Implementation of Korea Standard Hash Function HAS-160

Young-ho Seo[0] Jong-Hyeon Kim Wang-Hyun Kim Dong-Wook Kim
Dept. of Electronic Materials Engineering, Kwangwoon University
Intis.com Inc.

요 약

이 논문은 한국정보통신기술협회에 의해 1998년 11월에 제정된 HAS-160 해쉬 함수를 하드웨어로 구현하였다. SET(Secure Electronic Transaction) 혹은 SSL(Secure Socket Layer)등의 암호 프로토콜에서 하나의 구성 요소를 이루기 위해 설계되었고 초기값을 원래의 알고리즘에서 주어진 값이 아닌 사용자 혹은 프로토콜이 요구하는 값으로 입력할 수 있게 하였다. 전체적으로 회로는 VHDL top-down 설계 방법을 따랐고 IEEE 표준 라이브러리만을 사용하여 범용성을 가진다. 그리고 블록들은 내부적으로 행위적 수준(behavior level)에서 설계되었고 설계된 각각의 블록들은 구조적 수준(structure-level)에서 연결되었다. 설계된 회로는 125MHz의 클럭 주파수와 26Mbps의 성능으로 동작하며 ALTERA FLEX10K EPF10K200칩에서 6018(60%)개의 셀을 차지한다.

1. 서론

HAS(Hash Algorithm Standard)-160은 1998년 11월에 정보처리시스템 또는 정보통신망환경에서 정보보호 서비스를 제공하는 전자서명에 활용될수 있도록 한국정보통신기술회(TTA)에 의해 암호학적 해쉬함수의 표준으로 제정되었다[1]. 또한 현재 MD4[2], MD5, SNEFRU[3], 그리고 SHA-1[3] 등 여러가지 해쉬 함수들이 세계적으로 널리 쓰이고 있다. HAS-160은 기존의 SHA(Secure Hash Algorithm) 압축함수[5]를 기본으로 설계되었으며 임의 길이의 입력메시지를 받아서 고정된 160비트의 출력값인 해쉬코드로 압축시킨다. 해쉬함수는 주어진 해쉬코드에 대하여 이 해쉬코드를 생성하는 데이터 스트링을 찾아내는 것은 계산상 불가능하며, 주어진 데이터 스트링에 대하여 같은 해쉬코드를 생성하는 또다른 데이터 스트링을 찾아내는 것은 계산상 실행 불가능하다는 두 가지 성질을 만족하는 함수를 말한다.

암호학적 해쉬 알고리즘의 충돌 저항성은 전자서명에서 송신자 외의 제3자에 의한 문서위조를 방지하는 부인봉쇄를 제공하기 위한 필수적인 요건이 된다.

암호 시스템은 빠른 시간내에 소프트웨어로 개발이 가능하고 사용상의 유연성 또한 높다. 하지만 트랩도어(trap-door), 바이러스, 해킹, 그리고 운영체제에 대한 의존성등의 소프트웨어 자체 불안정성을 가진다. 그리고 대용량의 고속 데이터 처리를 실시간으로 요구받을때 한계를 가지게 된다. 따라서 하드웨어로의 구현을 위한 많은 연구가 진행되어 오고 있고 본 논문도 역시 빠른 동작 속도와 강한 안전성을 충족시키기 위한 암호 시스템 구현의 일환으로 연구되었다.

본 연구는 SET(Secure Electronic Transaction)[6] 혹은 SSL(Secure Socket Layer)등의 암호 프로토콜에서 사용될 해쉬 함수를 목적으로 설계되었고 공개키 암호알고리즘을 KCDSA로 사용할 것을 고려하여 초기값(A, B, C, D, E)들을 KCDSA 알고리즘에서 주어진 값으로 입력할 수 있게 하였다.

설계된 회로는 125MHz의 클럭 주파수로 동작하고 512비트 입력에 대해 한번의 해쉬함수를 수행하는데 245개의 클럭수가 소요된다. 따라서 26Mbps의 성능을 가진다. 또한 ALTERA FLEX10K200 칩에서 6018개(60%)의 셀을 차지한다.

2. HAS-160 알고리즘

그림 1에 보이는 것처럼 해쉬 알고리즘 표준은 임의의 길이를 가지는 입력 메시지를 512비트의 블록 단위로 처리하며, 160비트를 출력으로 낸다.

2.1 해쉬 알고리즘의 변수

압축함수는 4라운드로 구성되고 해쉬함수 입력 블록의 크기는 512비트이며, 메시지 갱신에 필요한 연쇄변수의 개수는 5개이다. 또한 각 라운드에 적용될 메시지 변수의 개수는 입력블록 512비트로부터 생성된 16개의 변수와 이로부터 추가로 생성되는 4개의 변수를 포함하여 20개가 된다.

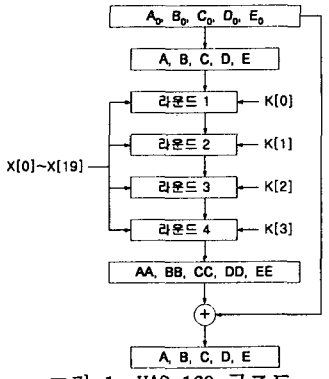

그림 1. HAS-160 구조도

2.2 입력블록 길이 및 덧붙이기

입력 메시지는 512비트 단위로 처리된다. 마지막 메시지 블록은 블록의 길이가 448비트(= 512-64)가 되도록 " 1" 다음에 필요한 개수의 " 0" 을 채운 다음, 마지막 64비트는 덧붙이기 전 메시지 길이를 2^{64}법 연산값으로 채운다.

2.3 초기값 및 상수

해쉬 알고리즘 표준의 연쇄 변수의 초기값은 표 1과 같고 각 라운드에서 사용되는 상수는 표 2와 같다.

A_0	B_0	C_0	D_0	E_0
67452301	EFCDAB89	98BADCFE	10325476	C3D2E1F0

표 1. 연쇄 변수의 초기값 (16진수)

K[0]	K[1]	K[2]	K[3]
0	5A827999 $\lfloor 2^{32}\sqrt{2} \rfloor$	6ED9EBA1 $\lfloor 2^{30}\sqrt{3} \rfloor$	8F1BBCDC $\lfloor 2^{30}\sqrt{5} \rfloor$

표 2. 라운드 상수값 (16진수)

2.4 단계 연산

단계 연산은 아래와 같이 정의 된다.

$A(i+1)=W(i)+ROL_{S1}(A(i))+f(B(i),C(i),D(i))+E(i)+K(i)$

$B(i+1)=A(i)$

$C(i+1)=ROL_{S2}(B(i))$

$D(i+1)=C(i)$

$E(i+1)=D(i)$

2.5 부울 함수(f 함수)

아래와 같은 3개의 부울 함수를 사용하여 4개의 라운드에 f_0, f_1, f_2, f_1의 순서로 적용한다.

f_0=(X AND Y) XOR (NOT(X) AND Z)

f_1=X XOR Y XOR Z

f_2=(X OR NOT(Z)) XOR Y

2.6 최종 연쇄변수 갱신과정

그림 1에 보이듯이 4라운드의 연산이 끝난 다음에는 연쇄변수를 다음과 같이 갱신시킨다. 여기서 A, B, C, D, E는 이전 메시지 블록에 대한 압축함수의 결과이며 AA, BB, CC, DD, EE는 현재의 메시지 블록에 의해 A, B, C, D, E가 갱신된 결과이다.

3. 하드웨어 설계

회로는 그림 2에 보여지는 것처럼 제어부와 데이터 패스 블록으로 나누어 진다. HAS-160은 4번의 라운드 동안 각각 20번의 연산과정을 순차적으로 거치므로 메시지 변수를 미리 구해 놓지 않고 연산이 이루어지는 하나의 클럭 전에 32비트 하나씩을 80번 생성시켜 사용한다. 따라서 회로에 사용되는 레지스터의 수를 줄인다. 제어부는 데이터패스 블록내의 레지스터들에 클럭을 제공하고 모든 기능 블록들을 제어한다.

회로는 VHDL로 설계되었고 top-down 설계 방법을 따랐다. 그리고 특정 라이브러리에 제한을 두지 않기 위해 오직 IEEE 표준 라이브러리만을 사용하였다

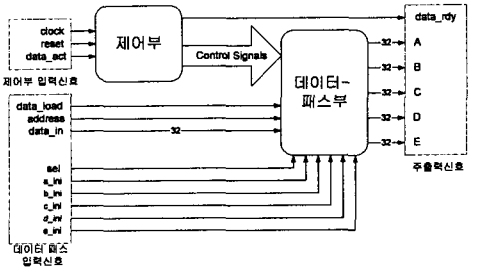

그림 2. HAS-160 하드웨어 전체 블록 다이어그램

KCDSA의 경우 시스템 변수 생성시 160비트의 해쉬함수를 요구하고 이때 KCDSA에 의해 제공되는 초기값을 가지고 해쉬함수는 압축동작을 수행한다. 따라서 HAS-160의 자체 알고리즘에 초기값의 조건적 선택이라는 부가적 기능을 첨가하였다.

3.1 데이터 패스 블록 설계

데이터 패스 블록의 내부 구조는 그림 3과 같다. 데이 터 패스 블록은 초기값을 선택하기 위한 입력단의 MUX, 초기값과 데이터의 귀환값을 선택하는 MUX, 그리고 라운드와 라운드내 20번의 순차적 동작에 따라 쉬프트 동작을 하는 a_shift와 b_shift, adder, 그리고 다수의 레지스터들로 구성되어있다. 초기값과 라운드 상수값을 저장하는 ROM은 범용적 설계를 위해 랜덤 로직으로 설계되었다. 각 라운드에서 20번의 연산 중 한번의 연산을 처리하기 위해 데이터 패스 블록은 3개의 클럭을 사용한다. 첫번째 클럭은 라운드당 20번의 연산과정을 나타내는 주소 신호를 생성 및 변화 시키는데 소요되고 두번째 클럭은 메시지 변수를 생성하는데 쓰이며 세번째 클럭은 레지스터 클럭으로 초기값을 받아들이거나 연쇄 변수를 귀환시킬 때 사용한다. 여기서 쉬프트(Shift) 동작을 위한 클록은 메시지 변수 생성시 쓰이는 클록에 동기한다.

3.2 제어부의 설계

그림 4에서 보여지는 것과 같이 제어부는 2개의 FSM(Finite State Machine)과 인코더(Encoder)로 구성되어 있다. 먼저, 라운드를 제어하는 FSM은 초기상태를 포함하여 10개의 상태로 구성되어 있다. 하나의 라운드는 2개의 상태로 처리되는데 이는 라운드 처리를 위한 FSM이 각 라운드 당 20번의 연산과정을 제어하는

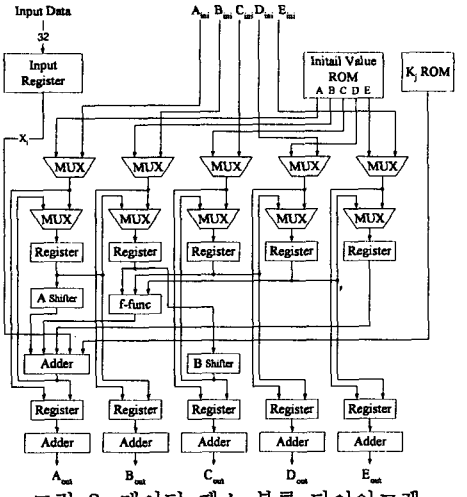

그림 3. 데이터 패스 블록 다이어그램

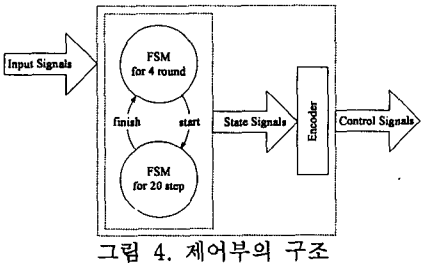

그림 4. 제어부의 구조

FSM을 안정하게 구동하기 위해서이다. FSM들의 동작으로 인해 발생된 상태 신호들은 인코더(Encoder)를 거쳐 제어 신호가 되고 이 제어 신호들은 그림 3에 보이는 데이터 패스내의 여러 블록들을 동작 순서에 따라서 구동시킨다. 이때 제어 신호들은 명확한 동작의 확립을 위해 버퍼링(buffering)을 시킨후 안정된 신호들로써 출력한다.

3.3 HAS-160 회로의 동작 순서

설계된 회로의 동작 순서는 그림 5에 보여지는 것과 같다. 먼저 HAS-160의 알고리즘에서 주어진 것 외의 다른 초기값을 부여할 것인지를 결정한다. 다음으로 512비트의 해쉬함수 입력값을 외부 회로와의 상호 인터페이스를 고려하여 32비트 단위로 16번 입력한다. 다음으로 20번의 연산 동작을 4번의 라운드 동안 수행한다. 두개(A shift, B shift)의 쉬프트(shift)동작은 동기되어 있고 f함수는 클럭에 무관하게 긴 동작 시간을 가지고 수행을 한다. 하나의 20번 연산동작을 마치면 하나의 클럭 시간동안 연산 동작 블록과 FSM이 초기화 상태로 가게 된다. 즉 4개의 부가적 클럭이 각 라운드당 요구되어 245(=80×3+4+1)개의 클럭이 필요하게 된다.

4. 실험 결과

본 논문에서 설계된 HAS-160 회로는 top-down 설계 방법을 이용하여 구현되었고 VHDL을 이용하여 기술하

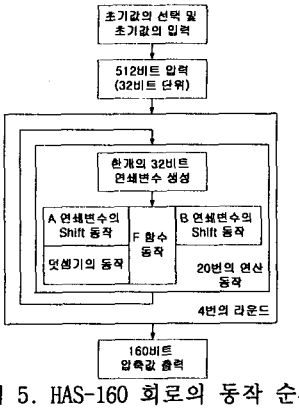

그림 5. HAS-160 회로의 동작 순서

였다. 각각의 모듈은 행위적 수준(behavior-level)에서 설계되었고 설계된 각각의 모듈은 구조적 수준(structural-level)에서 연결되었다. 또한 FPGA이외의 ASIC으로도 수정없이 구현될 수 있다. 그리고 SET등과 같은 암호 프로토콜의 코어(core)를 목적으로 설계하였으므로 원래의 알고리즘에 초기값의 선택적 입력이라는 부가적인 기능을 첨부하였다.

설계된 회로는 125MHz의 클럭 주파수로 동작하고 512비트 입력에 대해 한번의 해쉬함수를 수행하는데 245개의 클럭수가 소요된다. 따라서 26Mbps의 성능을 가진다. 또한 ALTERA FLEX10K EPF10K200칩에서 6018(60%)개의 셀을 차지한다. 이번 연구에서 설계된 HAS-160은 1998년 11월 발표된 문서를 기준으로 구현되었다.

5. 결론

본 논문에서는 표준 해쉬 함수인 HAS-160을 하드웨어로 설계하였다. 설계된 해쉬 함수는 범용적(technology-independent)으로 설계되어 이식성이 강하고 고속 동작을 수행한다. 그리고 암호 프로토콜 구현시 거의 수정이 없이 하나의 코어(core)로 사용이 가능하다.

6. 참고문헌

[1] 해쉬함수 표준- 제2부 : 해쉬함수 알고리즘(HAS-160), 한국정보통신기술협회, 1998년 11월.

[2] Rivest, R. " The MD4 Message Digest Algorithm." Proceedings, Crypto' 90, August 1990; published by Springer-Verlag.

[3] R.C.Merkel, " A Fast Software One-Way Hash Function," J. of Cryptology, vol. 3, no.1, pp. 43-58, 1990.

[4] NIST, " Secure hash standard" , FIPS 180-1, US Department of Commerce, Washington D.C., April 1995

[5] NIST, " Secure hash standard" , FIPS 180, US Department of Commerce, Washington D.C., 1993

[6] Visa, MasterCard, The SET Standard Technical Specifications, 1997, 5