SHA-1과 HAS-160과 의사 난수 발생기를 구현한 해쉬 프로세서 설계

정회원 전 신 우*, 김 남 영**, 정 용 진***

Design of Hash Processor for SHA-1, HAS-160, and Pseudo-Random Number Generator

Shin-woo Jeon*, Nam-young Kim**, Yong-jin Jeong*** *Regular Members*

요 약

본 논문에서는 미국과 한국의 해쉬 함수 표준인 SHA-1과 HAS-160 해쉬 알고리즘, 그리고 SHA-1을 이용한 의사 난수 발생기를 구현한 해쉬 프로세서를 설계하였다. SHA-1과 HAS-160이 동일한 단계 연산을 가지므로, 한 단계 연산만을 구현하여 공유함으로써 하드웨어 리소스를 감소시켰다. 그리고 메시지 변수의 사전 계산과 단계 연산을 두 단계의 파이프라인 구조로 구현함으로써 한 개의 클럭으로 한 단계 연산을 수행하는 방식보다 최장지연경로는 1/2로 줄고, 총 단계 연산에 필요한 클럭 수는 하나만 증가하므로 성능은 약 2배 향상되었다. 그 결과, 설계한 해쉬 프로세서는 삼성 0.5 um CMOS 스탠다드 셀 라이브러리를 근거로 산출할 때, 100 MHz의 동작 주파수에서 약 624 Mbps의 성능을 얻을 수 있다. 그리고 의사 난수 발생기로 사용될 때는 약 195 Mbps의 난수 발생 성능을 가진다. 이러한 성능은 지금까지 상용화된 국내외의 어느 해쉬 프로세서보다 빠른 처리 시간을 가지는 것으로 판단된다.

ABSTRACT

In this paper, we present a design of a hash processor for data security systems. Two standard hash algorithms, SHA-1(American) and HAS-160(Korean), are implemented on a single hash engine to support real time processing of the algorithms. The hash processor can also be used as a PRNG(Pseudo-random number generator) by utilizing SHA-1 hash iterations, which is being used in the Intel software library. Because both SHA-1 and HAS-160 have the same step operation, we could reduce hardware complexity by sharing the computation unit. Due to precomputation of message variables and two-stage pipelined structure, the critical path of the processor was shortened and overall performance was increased. We estimate performance of the hash processor about 624 Mbps for SHA-1 and HAS-160, and 195 Mbps for pseudo-random number generation, both at 100 MHz clock, based on Samsung 0.5um CMOS standard cell library. To our knowledge, this gives the best performance for processing the hash algorithms.

Ⅰ. 서 론

국내에서 최근 빈발하고 있는 전산 시스템에서의 범죄나 보안 사고들은 정보화 사회로의 발전에 가장 큰 걸림돌이 되고 있다. 컴퓨터에 의한 전산망의 발달은 자유로운 정보의 교환이라는 장점이 있지만 이러한 컴퓨터 범죄나 프라이버시 침해 등은 중요한 정보를 처리하는 대규모 전산 센터뿐만 아니라

* 광운대학교 전자통신공학과 실시간 구조 연구실(swjun@explore.gwu.ac.kr)
** 광운대학교 전자공학부 부교수(nykim@daisy.gwu.ac.kr), *** 광운대학교 전자공학부 조교수(yjjeong@daisy.gwu.ac.kr)
논문번호 : K01186-0827, 접수일자 : 2001년 8월 27일
※ 본 연구는 2001년도 광운대학교 산학연 공동기술개발 컨소시엄 사업의 지원에 의해 이루어졌습니다.

개인의 생명도 침해할 수 있는 위협 요소로 등장하고 있다.

이러한 전산망 위협 요소에 대응하기 위해 서명과 신분의 인증이 중요한 기능으로 대두되고 있으며 이러한 기능 중 핵심 요소들 중의 하나가 바로 해쉬 함수이다. 해쉬 함수는 임의의 길이의 입력을 일정한 길이의 출력 결과로 안전하게 생성하여 메시지의 무결성을 보장해주는 함수로서 각종 정보 보호 메카니즘에 활용되는 요소 기술이다. 정보 보호에 사용되는 해쉬 함수의 기본 조건으로는 주어진 해쉬 코드에 대하여 이 해쉬 코드를 생성하는 데이터 스트링을 찾아내는 것은 계산상 실행 불가능하며, 주어진 데이터 스트링에 대하여 같은 해쉬 코드를 생성하는 또 다른 데이터 스트링을 찾아내는 것은 계산상 실행 불가능하다는 두 가지 성질을 만족하는 함수를 말한다.

해쉬 알고리즘은 크게 DES와 같은 블록 암호 알고리즘에 기초한 해쉬 알고리즘과 전용 해쉬 알고리즘으로 나눌 수 있다. 블록 암호를 이용한 해쉬 알고리즘은 이미 구현되어 사용되고 있는 블록 암호를 사용할 수 있다는 이점이 있으나, 대부분의 블록 알고리즘의 속도가 그리 빠르지 않을 뿐더러 이를 기본 함수로 이용한 해쉬 알고리즘의 경우 블록 암호보다도 훨씬 더 속도가 떨어지므로 현재는 대부분의 응용에서 전용 해쉬 알고리즘이 주로 이용된다. 전용 해쉬 알고리즘으로는 MD4, MD5, SHA-1, RIPEMD-160, HAS-160등이 있다.[1][2][4]

암호 알고리즘의 많은 응용은 대칭키 암호에서의 비밀키나 IV(Initial Vector) 생성, 공개키 암호의 비밀키/공개키 혹은 공개 파라미터 생성 등을 위해 언제나 난수를 필요로 하므로 거의 모든 암호 알고리즘 관련 규격들은 난수를 암호 시스템에서 생성하도록 하고 있다. 그래서 대부분의 암호 시스템은 난수 발생기를 포함하고 있다. 난수는 크게 순수 난수(True Random Number)와 의사 난수(Pseudo Random Number)로 구분하는데, 전자는 주로 하드웨어적인 잡음원을 증폭시켜 샘플링함으로써 난수가 얻어진다. 그리고 후자는 해쉬 함수나 CBC-MAC 같은 안전한 일방향 함수, 또는 이산대수나 소인수 분해 같은 어려운 수학적 이론을 이용하여 난수를 생성한다. 하드웨어로 구현된 암호 시스템에서는 전자 방식의 하드웨어 난수 생성기를 탑재하는 경우도 많지만, 대부분 이 하드웨어 난수 생성기로부터 발생하는 난수를 하나의 잡음원으로 간주하고, 이를 암호학적인 함수를 통해 변환시켜 사용하는 의사 난수 발생기를 주로 사용한다. 그런데 이를 소프트웨어로 구현한다면 많은 부하가 예상되므로 암호 시스템에서 난수 발생기의 하드웨어적 구현은 암호 시스템의 성능 향상에 꼭 필요한 요소라 할 수 있다.

이에 본 논문에서는 현재 미연방정보처리규격 FIPS(Federal Information Processing Standard) 180-1의 표준안인 SHA-1[1]과 국내 표준 해쉬 알고리즘인 HAS-160[2]과 인텔 소프트웨어 라이브러리에서 사용하는 SHA-1 해쉬 알고리즘에 기반을 둔 의사 난수 발생기[3]를 고속으로 처리할 수 있는 하드웨어 구조를 제안하고, 이를 하드웨어로 구현하였다. SHA-1과 HAS-160은 동일한 단계 연산을 가지므로, 한 단계 연산만을 구현하여 공유함으로써 하드웨어의 양을 감소시켰다. 그리고 SHA-1을 이용한 의사 난수 발생기를 사용함으로써, 내부상태변수를 갱신해 주는 믹서의 구현만으로 의사 난수 발생기의 기능을 추가하였다.

성능 향상을 위해서는 메시지 변수의 사전 계산과 단계 연산의 두 단계 파이프라인 구조를 사용하였다. 메시지 변수는 사용되기 이전 클럭에 미리 계산하여 동작 주파수를 감소시키는 최장지연경로를 생성하지 않도록 하였다. 그리고 단계 연산에서는 두 단계의 파이프라인 구조를 사용하여 한 클럭으로 한 번의 단계 연산을 수행하는 방식에 비해 최장지연경로가 약 1/2로 줄어 클럭 주파수가 증가하고, 총 단계 연산에 필요한 클럭 수는 80개에서 81개로 한 개만 증가하므로 성능이 약 두 배 향상되었다.

해쉬 알고리즘을 하드웨어로 구현한 기존의 사례를 살펴보면, 대부분의 회사들이 각자 구현하여 사용하고 있겠지만, 대부분 내부 구조를 공개하지 않고 있다. 성능이 공개된 [5]의 경우에는 SHA-1 해쉬 알고리즘을 구현하였고, 한 단계 연산에 필요한 하드웨어만 갖추고, 이를 80번 반복 사용하여 해쉬 함수를 수행하는 구조를 채택하였다. 그리고 SHA-1 해쉬 알고리즘의 한 단계 연산을 한 클럭으로 구현하였다.

본 논문의 구성은 2 장에서는 본 논문에서 구현한 SHA-1, HAS-160 해쉬 알고리즘과 의사 난수 발생기 알고리즘에 대해 간단히 소개하고, 3 장에서는 의사 난수 발생기를 겸한 SHA-1과 HAS-160 해쉬 알고리즘을 수행하는 해쉬 프로세서의 설계에 대해 다룬다. 그리고 4 장에서는 구현한 해쉬 프로세서의 성능을 비교 분석하며, 5 장에서는 결론과 향후 연구 방향을 제시한다.

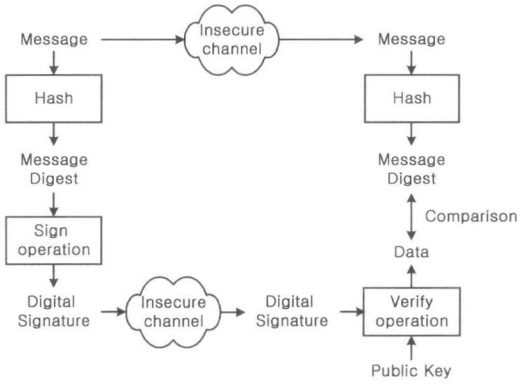


그림 1. 전자 서명의 생성 및 검증 과정

Ⅱ. 해쉬 알고리즘과 의사 난수 발생기

1. 해쉬 알고리즘의 응용 분야

해쉬 함수는 임의의 길이의 입력을 고정된 길이의 출력값인 해쉬 코드로 압축시키는 함수로, 응용 분야는 전자 서명의 효율성 증대와 중요 정보의 무결성 확인이다.

전자 서명의 경우 해쉬 함수로 메시지를 압축한 데이터에 서명함으로써 효율성을 높일 수 있다. RSA, DSA(Digital Signature Algorithm), KCDSA (Korea Certificate-based Digital Signature Algorithm)와 같이 전자 서명에 사용되는 알고리즘은 메시지 자체를 암호화하는 경우 해쉬 함수보다 처리 속도가 느린데, 서명을 메시지 해쉬 함수로 축약함으로써 전자 서명 알고리즘의 계산량을 줄일 수가 있다. 즉, 그림 3과 같이 송신자는 메시지의 해쉬값을 구하고 그 해쉬 값에 자신의 비밀키로 전자서명을 한 후, 메시지와 서명을 송신한다. 수신자는 메시지를 동일한 해쉬 함수로 축약하고 전자 서명을 공개키로 복호화해서 구해지는 해쉬값과 비교함으로써 서명을 인증하게 된다.

무결성 확인 방법은, 우선 송신자는 메시지와 함께 그 메시지의 해쉬값을 함께 보내면, 수신자는 메시지를 동일한 해쉬 함수로 압축하여 송신자로부터 받은 해쉬값과 비교하여 값이 동일하면 변조되지 않은 것으로 받아들일 수 있다.

그리고 IPsec을 비롯해 다양한 인터넷 프로토콜에서 사용하는 메시지 인증 알고리즘인 HMAC (Hash-based Message Authentication Code)에서도 사용한다. HMAC은 메시지 무결성과 함께 메시지의 출처 인증을 위해 사용된다.

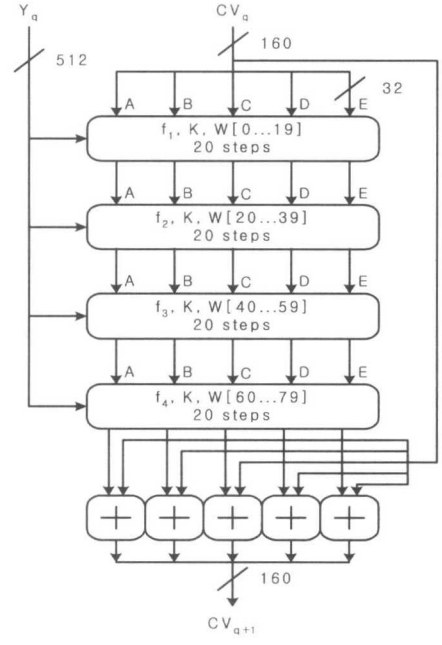


그림 2. SHA-1과 HAS-160의 전체 구조도

2. SHA-1과 HAS-160 해쉬 알고리즘

SHA-1과 HAS-160은 대부분의 전용 해쉬 알고리즘과 같이 임의의 길이의 메시지 X를 입력단위의 배수가 되도록 덧붙이기를 하여 t개의 입력 블록 (X1,...,Xt)으로 분할한다. 해쉬 코드는 각 블록 Xi에 대해 연쇄 변수를 주어진 초기값(IV)으로 초기화한 후 압축 함수를 반복적으로 적용하여 계산되며, 그 처리 과정은 식 1과 같이 기술할 수 있다. 여기서 f는 h의 압축 함수이며, Hi는 단계 i의 중간 계산 값이다.

$$H_0 = IV,$$
$$H_i = f(H_{i-1}, X_i),\ 1 \le i \le t,$$
$$h(X) = H_t \qquad (1)$$

SHA-1과 HAS-160은 MD4[4]의 설계 원리에 기반을 두고 만든 해쉬 알고리즘으로 2^{64}비트보다 작은 메시지에 대해 512비트 단위의 메시지 블록으로 분할한 후, 각 메시지 블록에 대해 20 스탭으로 이루어진 4 라운드의 총 80 단계의 연산 수행과 최종 연쇄 변수 갱신 과정을 거쳐서 160비트의 해쉬 코드를 생성하는 알고리즘으로, 그림 2와 같이 동일한 길이의 입출력 및 전체 구조를 가진다. 그리고 식 2와 같이 32비트인 5개의 연쇄 변수(A, B, C, D, E)와 메시지 변수(Wt)와 상수(Kt)를 입력으로 받아서

모듈러 덧셈 연산(+)과 데이터의 순환 이동(<<), 부울 함수 연산(Ft)을 통해 5개의 32비트 출력을 생성하는 동일한 단계 연산을 가진다. 이로 인해 하나의 단계 연산만을 구현하여 공유함으로써 하드웨어의 양을 감소시킬 수 있다.

$$\begin{aligned} A(t) &= A(t-1)^{<<S1(t)} + Ft(B(t-1), C(t-1), D(t-1)) \\ &\quad + E(t-1) + W_t + K_t; \\ B(t) &= A(t-1); \\ C(t) &= B(t-1)^{<<S2(t)}; \\ D(t) &= C(t-1); \\ E(t) &= D(t-1); \end{aligned} \quad (2)$$

SHA-1과 HAS-160은 위와 같은 공통점이 있는 반면에 메시지 변수의 생성 과정과 연쇄 변수의 초기값, 단계 연산에서 A와 B의 순환 이동의 양, 3라운드에서의 부울 함수는 다르다.

SHA-1의 메시지 변수 생성 과정의 경우, 처음 16 단계는 입력 메시지 512비트를 32비트씩 나누어 MSB(Most Significant Bit)부터 차례대로 사용하고, 그 이후부터는 식 3과 같이 이전에 사용된 네 개의 메시지 변수의 XOR 연산과 한 비트의 데이터 순환 과정을 거쳐 생성된다.

$$W_t = S^1(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}) \quad (3)$$

HAS-160은 메시지 변수는 입력 메시지 512비트(16개의 워드,X[0], X[1],..., X[15])와 이로부터 각 라운드마다 네 개의 메시지 워드를 XOR하여 추가로 네 개의 메시지 변수(X[16], X[17], X[18], X[19])를 생성하며, 그 생성방법은 식 4와 같이 각 라운드의 메시지 변수의 적용순서에 따라 다르다.

$$\begin{aligned} W_i[16] &= W_i[l(t+1)] \oplus W_i[l(t+2)] \oplus W_i[l(t+3)] \\ &\quad \oplus W_i[l(t+4)] \\ W_i[17] &= W_i[l(t+6)] \oplus W_i[l(t+7)] \oplus W_i[l(t+8)] \\ &\quad \oplus Wi[l(t+9)] \\ W_i[18] &= W_i[l(t+11)] \oplus W_i[l(t+12)] \oplus W_i[l(t+13)] \\ &\quad \oplus W_i[l(t+14)] \\ W_i[19] &= W_i[l(t+16)] \oplus W_i[l(t+17)] \oplus W_i[l(t+18)] \\ &\quad \oplus W_i[l(t+19)] \end{aligned} \quad (4)$$

두 해쉬 알고리즘의 부울 함수와 상수, A와 B의 순환 이동의 양은 표 1과 같이 각 라운드에 따라 다르게 적용된다. 여기서 ⊕, v, ^와 ~은 각각 XOR, OR, AND와 비트별 NOT 연산을 나타낸 것이다. 그리고 ^{0}은 $0 \le t \le 19$, ^{1}은 $20 \le t \le 39$, ^{2}는 $40 \le t \le 59$, ^{3}은 $60 \le t \le 79$의 단계에서 적용된다.

표 1. SHA-1과 HAS-160의 부울 함수와 상수와 순환 이동의 양

		SHA-1	HAS-160
부울 함수 (Ft())		$(B\hat{}C)v(\sim B\hat{}D)^0$ $B\oplus C\oplus D^{1,3}$ $(B\hat{}C)v(B\hat{}D)v(C\hat{}D)^2$	$(B\hat{}C)v(\sim B\hat{}D)^0$ $B\oplus C\oplus D^{1,3}$ $(Bv\sim D)\oplus C^2$
상수 (K_t)		$5a827999^0$ $6ed9eba1^1$ $8f1bbcdc^2$ $ca62c1d6^3$	00000000^0 $5a827999^1$ $6ed9eba1^2$ $8f1bbcdc^3$
순환 이동의 양	A (s_1)	5	5, 11, 7, 13, 15, 6, 13, 9, 5, 11, 7, 12, 8, 15, 13, 8, 15, 6, 7, 14
	B (s_2)	30	10^0, 17^1, 25^2, 30^3

3. SHA-1을 이용한 의사 난수 발생기

본 논문에서 구현한 의사 난수 발생기 알고리즘은 인텔 소프트웨어 라이브러리에서 사용하는 SHA-1 해쉬 알고리즘에 기반을 둔 난수 발생 알고리즘을 사용하였다. SHA-1을 이용한 의사 난수 발생기의 전체 구조는 그림 3과 같으며, 내부상태변수를 입력으로 SHA-1 해쉬 알고리즘을 수행하여 출력되는 160비트의 해쉬 코드를 난수로 사용한다. 그리고 내부상태변수는 노출되더라도 그 영향을 최소화할 수 있도록 주기적으로 식 5의 초기화 과정, 또는 잡음추가 과정을 수행하여 내부상태변수를 갱신한다.

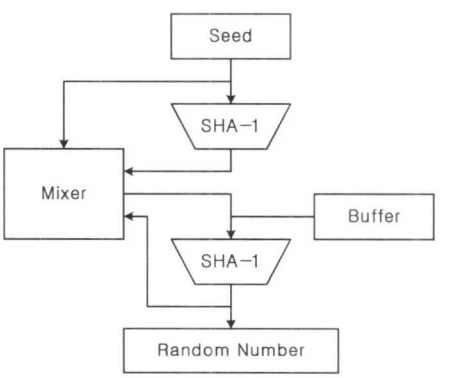


그림 3. 의사 난수 발생기의 전체 구조

내부상태변수는 그림 4와 같이 내부상태변수의 사용 후에는 SHA-1 해쉬 알고리즘을 수행하여 생성된 해쉬 코드(T)와 새로운 입력된 seed(Seed), 이

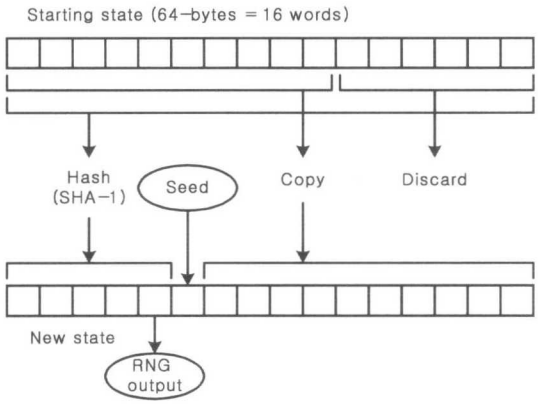


그림 4. 의사 난수 발생기의 믹서 구조

전 내부상태변수(S)의 일부를 이용하여 내부상태변수를 갱신한다.

$$
\begin{aligned}
&\text{초기화과정 : } T^* = f(\text{Seed}^*)\\
&\qquad S = T^* \parallel \text{Seed} \parallel S^*[512:192]\\
&\text{잡음추가과정 : } T^* = f(S^*)\\
&\qquad S = T^* \parallel \text{Seed} \parallel S^*[512:192]\\
&\text{난수생성과정 : } T = f(S) \qquad (5)
\end{aligned}
$$

Seed : 32-bit의 외부 입력 (*는 512-bit)
S : 내부상태변수 (*는 이전의 내부상태변수)
T : 난수 (*는 이전에 생성된 난수)
f( ) : SHA-1 해쉬 알고리즘

Ⅲ. 해쉬 프로세서의 ASIC 설계

본 장에서는 2 장에서 기술한 SHA-1, HAS-160 해쉬 알고리즘과 의사 난수 발생기를 구현한 해쉬 프로세서의 설계 사양과 전체 구조, 메시지 변수 처리 블록과 단계 연산 블록과 제어 블록의 설계, 전체 수행 과정에 대해 기술한다.

1. 해쉬 프로세서의 설계 사양

본 논문에서 구현한 해쉬 프로세서는 외부 호스트 프로세서에 대한 보조 프로세서 형태로 설계되었으며, SHA-1과 HAS-160 해쉬 알고리즘과 의사 난수 발생기를 지원하도록 설계하였다. 외부 인터페이스는 32 비트 PCI 인터페이스 방식을 지원한다. 그리고 입출력 버퍼 레지스터와 해쉬 프로세서의 입출력 레지스터를 분리하여 해쉬 프로세서가 동작 중에 데이터의 입출력 과정이 동시에 수행 가능하게 함으로써 입출력 시간에 따른 성능 저하를 없앴다.

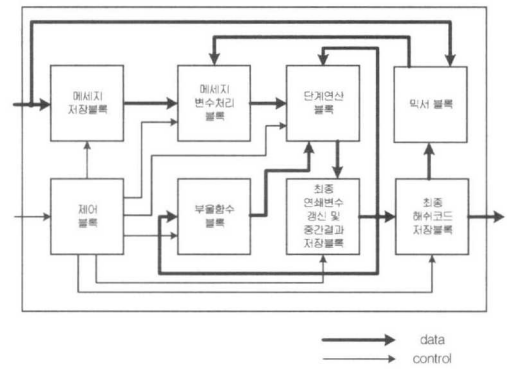


그림 5. 해쉬 프로세서의 전체 구조도

2. 해쉬 프로세서의 전체 구조

본 논문에서 제안한 해쉬 프로세서의 전체 구조는 그림 5와 같이 크게 8개의 블록(메시지 저장 블록, 메시지 변수 처리 블록, 부울 함수 연산 블록, 단계 연산 블록, 최종 연쇄 변수 갱신 및 중간 결과 저장 블록, 최종 해쉬 코드 저장 블록, 제어 블록, 믹서 블록)으로 나누어져 있다.

먼저 메시지 저장 블록은 임의의 길이의 메시지를 512비트의 배수가 되도록 덧붙이기 한 후, 분할되어 외부의 데이터 버스를 통해 입력되는 512비트의 메시지를 저장한다. 본 논문에서 구현한 해쉬 프로세서는 메시지의 덧붙이기와 분할 과정은 소프트웨어로 처리되는 것으로 가정한다.

메시지 변수 처리 블록은 2장에서 설명한 메시지 변수 생성 방법으로 메시지 저장 블록에 저장된 512비트의 메시지로부터 각 단계에 적용될 메시지 변수를 생성한다.

부울 함수 연산 블록은 표 1에서와 같이 해쉬 함수와 라운드에 따라 다른 부울 함수를 적용하여 사용되는 라운드의 단계 연산 시 부울 함수의 연산을 수행한다.

단계 연산 블록은 32비트인 5개의 연쇄 변수와 메시지 변수와 상수와 부울 함수 연산 블록으로부터 계산된 값을 입력으로 받아서 모듈러 덧셈 연산과 데이터의 순환 이동을 통해 5개의 32비트 출력을 생성한다.

최종 연쇄 변수 갱신 및 중간 결과 저장 블록은 단계 연산이 수행될 때마다 생성되는 연쇄 변수의 중간 결과를 저장하고, 4 라운드까지의 단계 연산이 끝난 후에는 연쇄 변수와 이전 메시지 블록에 대한 해쉬 알고리즘의 연산 결과를 더하는 최종 연쇄 변수 갱신 과정을 수행한다.

최종 해쉬 코드 저장 블록은 입력 메시지에 대해 해쉬 함수의 연산을 수행하여 최종 계산된 160비트의 해쉬 코드를 저장한다.

제어 블록은 SHA-1과 HAS-160 해쉬 알고리즘에 따라 서로 다른 연산 부분을 구분해주고, 512비트의 메시지가 4 라운드까지의 단계 연산 과정을 끝낸 후 최종 연쇄 변수 갱신 과정을 통해 해쉬코드를 생성할 수 있도록 제어한다.

믹서 블록은 의사 난수 발생기에서 사용되는 블록으로, 잡음 추가를 위한 의사 난수 발생기의 내부 상태변수의 갱신 작용을 한다.

3. 메시지 변수 처리 블록 설계

메시지 변수 처리 블록은 입력된 512비트의 메시지로부터 각 단계에 적용될 메시지 변수를 생성하는 블록으로, SHA-1과 HAS-160 해쉬 알고리즘의 메시지 변수의 처리 과정이 다르므로 따로 구현하였다. 그리고 80개의 메시지 변수를 해쉬함수의 수행 전에 모두 계산하는 오프라인 방식을 사용하면 레지스터의 수와 메시지 변수 계산으로 인해 연산 시간이 증가하는 문제가 발생하므로, 본 논문에서는 메시지 변수의 생성을 단계 연산과 병행하여 계산하는 온라인 방식을 사용하였다.

먼저 SHA-1의 메시지 생성 블록에서 처음 16 단계의 메시지 변수는 따로 연산 과정이 필요 없고, 그 이후 단계에서의 메시지 변수는 식 3과 같이 이전의 메시지 변수를 입력으로 연산이 이루어지므로 단계 연산에서 사용되기 전에 미리 계산이 가능하다. 이에 본 논문에서는 그림 6과 같이 SHA-1 해쉬 알고리즘의 메시지 변수 생성 블록을 설계하였다. 구현한 SHA-1 해쉬 알고리즘의 메시지 변수 생성 블록은 첫 번째 클럭에서 32비트의 레지스터 16개에 512비트의 입력 메시지를 받아 저장한다. 그리고 다음 클럭부터는 0번째 레지스터에서 14번째 레지스터까지는 뒤의 레지스터 값이 이동하여

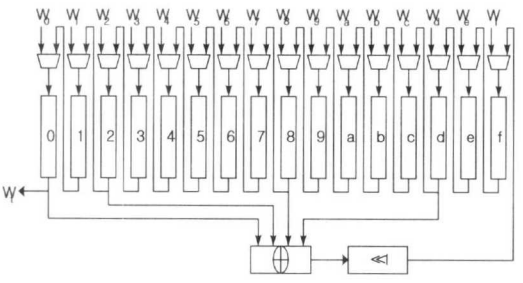


그림 6. SHA-1의 메시지 변수 처리 블록도

저장되고, 마지막 15번째 레지스터는 그림 6처럼 이전 단계에 사용될 네 개의 워드를 XOR와 쉬프트 연산을 하여 저장한다. 이렇게 연산 과정이 필요한 17 단계부터의 메시지 변수는 미리 사용되기 15클럭 전에 계산되고, 그 이후부터는 다음 레지스터로 이동하는 동작만 하므로 동작 주파수를 감소시키는 최장지연경로를 생성하지 않는다.

HAS-160은 입력 메시지 512비트(16개의 메시지 변수)와 이로부터 각 라운드마다 서로 다른 네 개의 메시지 변수를 XOR하여 네 개의 메시지 변수를 추가로 생성하며, 그 생성 방법은 각 라운드의 메시지 변수의 적용 순서에 따라 다르다. 그러므로 입력 메시지와 라운드마다 추가로 생성된 네 개의 메시지 변수를 레지스터에 저장하고, 이를 멀티플랙서를 통해 각 라운드의 스텝에 맞게 적용하도록 구현하였다. 그래서 HAS-160도 SHA-1과 마찬가지로 메시지 변수의 계산이 사용될 단계 전에 미리 계산된다. 그러므로 HAS-160의 메시지 변수 생성 과정도 해쉬 프로세서의 수행 시 최장지연경로에서 제외되고, 이로 인해 성능을 향상시킬 수 있었다.

4. 단계 연산 블록 설계

단계 연산 블록은 식 2와 같이 네 번의 덧셈과 두 번의 데이터 순환 과정으로 이루어진 해쉬 알고리즘의 단계 연산을 구현한 블록이다. 본 논문에서 단계 연산 블록 설계는 한 단계 연산만 구현하고, 이를 80번 반복 수행하는 구조를 채택하였다. 그런데 한 단계 연산을 한 클럭으로 구현할 경우 최장지연경로가 2-입력 멀티플랙서와 플립플롭과 네 번의 32비트 모듈러 덧셈 연산으로 이루어지므로 낮은 동작 주파수를 갖게 된다. 그러므로 해쉬 프로세서의 단계 연산 블록을 그림 7과 같이 두 개의 부분 라운드로 나눈 후, 각 부분 라운드가 단일 클럭에 수행되도록 구현하였다. 부분 라운드 1은 E 값과 메시지 변수 값(W)과 부울 함수부(Ft) 연산 결과의 모듈러 덧셈 연산과 데이터 순환 과정을 통한 B, C, D ,E의 연산으로 이루어져 있고, 부분 라운드 2는 데이터 순환 과정을 거친 A 값과 라운드 상수(K)와 부분 라운드 1에서 계산된 결과의 모듈러 덧셈 연산으로 이루어져 있다.

그런데 그림 7에서 보면, 부분 라운드 1의 수행으로 다음 단계의 부분 라운드 1의 동작에 필요한 입력 값이 A 값만 제외하고 모두 계산되어 진다. Wt 값은 메시지 생성 블록에 의해 미리 계산되고, B와 C와 D와 E 값은 그림 7에서 보이는 것과 같

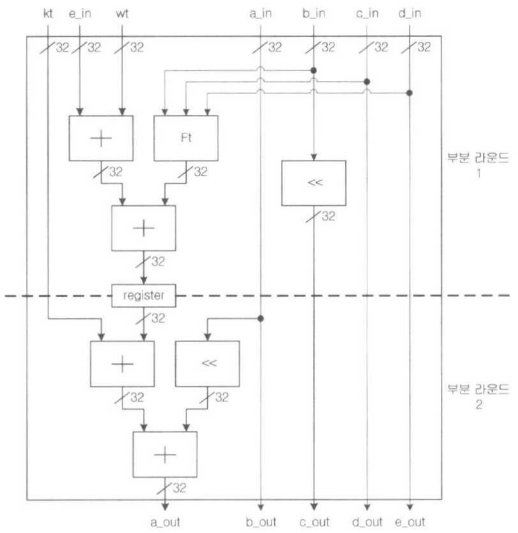


그림 7. 해쉬 프로세서의 단계 연산 블록도

이 부분 라운드 1의 수행으로 계산된다. 그러나 다음 단계의 B 값 계산을 위해 필요한 A 값은 동일 클럭에서 부분 라운드 2의 수행에 의해 생성되는 A 값으로 사용할 수 있다. 이는 다음 단계의 B 값은 연산 과정 없이 그림 7에서와 같이 단순히 A 값을 받아 사용하기 때문에 두 부분 라운드가 동시에 연산 수행이 가능하다. 그러므로 부분 라운드 2가 동작하는 동안 다음 단계의 부분 라운드 1이 동시에 동작할 수 있으므로, 두 단계의 파이프라인 구조를 사용하여 구현할 수 있다. 이렇게 두 단계의 파이프라인 구조로 구현하면 한 번의 부분 라운드 1의 동작을 위한 한 클럭만 추가함으로써 전체 80번의 단계 연산이 가능하고, 한 단계 연산을 두 개의 부분 라운드로 나눔으로써 동작 주파수가 약 두 배로 증가한다. 그러므로 한 클럭으로 한 단계 연산을 수행하는 방식에 비해, 본 논문에서 제안한 두 단계의 파이프라인 방식은 최장지연경로가 약 1/2로 줄어들어 클럭 주파수가 약 두 배 증가한데 비해, 총 단계 연산에 필요한 클럭 수는 80개에서 81개로 한 개만 증가하므로 성능을 약 두 배 향상시킬 수 있다. 이와 같은 방식으로 본 논문에서 구현한 단계 연산 블록에서 가장 주요한 연산인 32비트 모듈러 덧셈기는 고속 처리를 위해 4비트씩 두 단계의 CLA(Carry Lookahead Adder)를 사용하여 구현하였다.

5. 제어 블록의 설계

본 논문에서 설계한 해쉬 프로세서에서 SHA-1과 HAS-160과 의사 난수 발생기는 mode 신호에 의해 선택되고, 시작(start) 신호가 1이 되면 해쉬 함수의 연산이 시작된다. 해쉬 함수의 각 단계마다의 연산 수행은 두 개의 카운터(s_cnt와 r_cnt)에 의해 제어된다. s_cnt는 한 라운드의 20 스탭동안의 연산을 제어하기 위해 사용되는 카운터로, 그림 8과 같이 매 클럭마다 하나씩 증가하며 20이 되면 0으로 돌아가게 된다. 그리고 r_cnt는 s_cnt가 20이 될 때마다 하나씩 증가하여 4까지 카운트하며, 네 번의 라운드 동작을 제어하기 위한 카운터이다.

해쉬 프로세서의 수행은 idle 상태에서 시작 신호가 1이 되면 round 상태가 되어 두 개의 카운터에 의해 메시지 생성과 부울 함수 연산과 80 번의 단계 연산을 수행한다. 그리고 r_cnt와 s_cnt가 각각 0과 1E가 되면 final_add 상태가 된다. final_add 상태에서는 최종 연쇄 변수 갱신 과정을 수행하며, 의사 난수 발생기로 사용 시에는 내부상태변수의 갱신 과정도 이 상태에서 일어난다. final_add 상태에서 end 신호가 1이 되면 idle 상태로 돌아가고, 0이 되면 round 상태가 되어 해쉬 함수를 수행하게 된다. 이와 같은 제어 블록의 FSM(Finite State Machine)은 그림 9에 보였다.

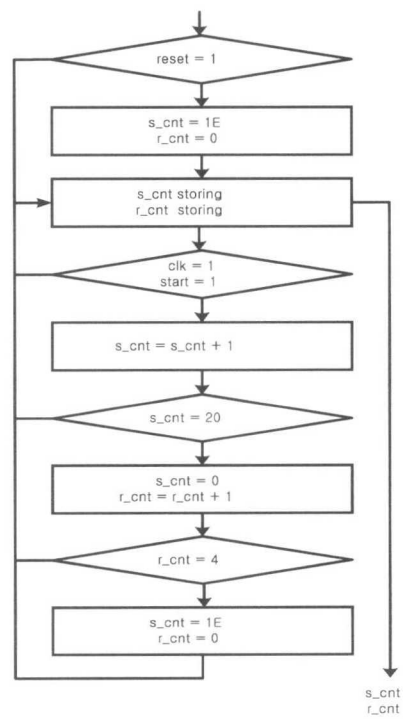


그림 8. s_cnt와 r_cnt의 상태도

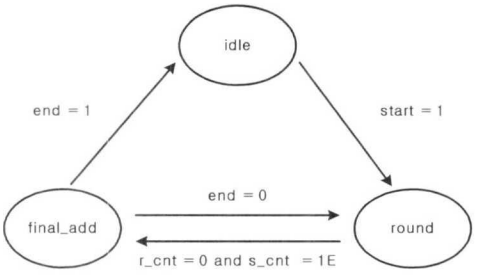


그림 9. 제어 블록의 FSM

6. 전체 수행 과정

본 논문에서 설계한 해쉬 프로세서의 전체 수행 과정은 다음과 같다. 먼저 외부로부터 소프트웨어로 메시지의 덧붙이기와 분할 과정을 거쳐 생성된 512 비트의 입력 메시지를 32비트 PCI 인터페이스를 통해 메시지 저장 블록에 저장한다. 그리고 시작 신호가 발생 시 데이터 저장 블록에 저장된 데이터는 메시지 생성 블록에 입력되어 메시지 변수를 생성하고, 단계 연산 블록의 부분 라운드 1이 수행함으로써 해쉬 함수의 동작이 시작된다. 81클럭동안 단계 연산 블록을 반복 수행한 후, 다음 클럭에서 초기 값을 더해주는 최종 연쇄 변수 갱신 과정을 거쳐서 최종 160비트의 해쉬 코드를 출력하여 해쉬 코드 저장 블록에 저장된다. 그러나 입력 메시지가 512비트보다 클 경우에는 메시지의 덧붙이기와 분할 과정을 거쳐 생성된 512비트의 메시지 개수만큼 위의 과정을 반복 수행하여 해쉬 코드를 생성한다. 이 경우 두 번째 512비트의 메시지부터는 최종 연쇄 변수 갱신과정에서 초기 값이 아닌 이전 메시지

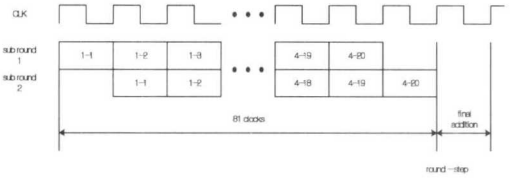


그림 10. 해쉬 프로세서의 전체 타이밍 분석도

블록에 대한 해쉬 함수의 결과를 더해준다. 이와 같이 512비트의 메시지에서 160비트의 해쉬 코드를 생성하는데 총 82클럭이 걸린다. 81클럭동안 단계 함수 블록을 수행하고, 82번째 클럭에서 최종 연쇄 변수 갱신 과정을 거쳐서 해쉬 코드를 생성하도록 구현한 해쉬 프로세서의 전체 타이밍도는 그림 10에 보였다.

Ⅳ. 성능 분석

본 논문에서 설계한 해쉬 프로세서는 Verilog HDL을 사용하여 설계 및 검증을 하였고, 검증한 회로는 삼성 0.5 um CMOS 스탠다드 셀 라이브러리를 근거로 성능을 산출하였다. 이 때 총 게이트 수는 약 30,000이었다. 클럭 스피드를 결정짓는 요소인 최장지연경로는 2-입력 멀티플랙서 한 개와 플립플롭 한 개, 32 비트 덧셈기 두 개로 약 10 ns가 걸렸다. 그래서 100 MHz의 최대 동작 주파수에서 512비트의 메시지로 160비트의 해쉬 코드를 생성하는데 총 82클럭이 소요되어 총 수행 시간은 820 ns 이며, 약 624 Mbps의 성능을 보였다. 그리고 의사 난수 발생기로 사용 시에는 한 번의 해쉬 함수의

표 2. 해쉬 프로세서의 성능 비교

	[5]	[6]	[7]	본 논문
지원 알고리즘	SHA-1	SHA-1	SHA-1 MD5	SHA-1 HAS-160 PRNG
구조적 특징	1 step / 1 clock	1 step / 1 clock	1 step / 1 clock	2-stage pipelining
한 프레임 (512-bit)당 클럭 수	81 clock cycles	81 clock cycles	81 clock cycles (SHA-1) 65 clock cycles (MD5)	82 clock cycles
성능	310 Mbps	404 Mbps	417 Mbps (SHA-1) 519 Mbps (MD5)	624 Mbps (Hash) 195 Mbps (PRNG)
동작 주파수	50 MHz	64 MHz	75 MHz	100 MHz
게이트 수	20,000 gates	1413 LCs	-	30,000 gates
사용 공정	0.5 um	Altera APEX 20K	0.25 um	0.5 um

수행으로 160비트의 난수를 생성하므로 195 Mbps의 성능을 가진다.

본 논문에서 구현한 해쉬 프로세서의 객관적인 성능 평가를 위해 해쉬 알고리즘을 구현하여 상용화되어 사용중인 [5], [6]. [7]과 성능을 비교하여 표 2에 보였다. [5]와 [6]은 SHA-1만 수행하고, [7]은 SHA-1과 MD5를 수행한다. 그리고 [5], [6], [7]은 모두 한 클럭에 한 단계 연산을 수행하도록 구현하여, SHA-1을 한 번 수행하는데 총 81 클럭이 소요된다. 사용 공정이 모두 달라서 동작 주파수는 50 ~ 75 MHz를 사용하여 310 ~ 417 Mbps의 성능을 보인다. 이에 비해, 본 논문에서 구현한 해쉬 프로세서는 SHA-1과 HAS-160과 의사 난수 발생기를 모두 수행할 수 있다. 그리고 단계 연산을 두 개의 부분 라운드로 나누고, 이를 파이프라인 시키는 구조를 사용하여 [5], [6], [7]에 비해 클럭 수는 82개로 한 개 더 늘어났지만 최장지연경로가 약 1/2로 줄어 약 두 배정도 향상된 성능을 가진다. 그래서 본 논문의 해쉬 프로세서의 성능이 100 MHz의 동작 주파수에서 624 Mbps로, 동일한 0.5 um 공정을 사용한 [5]의 성능보다 약 두 배의 빠른 성능을 보였다. 이런 결과를 고려해 볼 때 본 논문에서 구현한 해쉬 프로세서가 고속의 연산을 요구하는 인터넷 보안 시스템에 적절하게 사용될 수 있을 것으로 판단된다.

V. 결 론

본 논문에서는 SHA-1, HAS-160 해쉬 알고리즘과 SHA-1 알고리즘을 이용한 의사 난수 발생기를 모두 구현한 해쉬 프로세서를 설계하였다. 먼저 면적 측면에서, SHA-1과 HAS-160이 동일한 단계 연산 구조를 가지므로 한 단계 연산만을 구현하여 공유함으로써 하드웨어 양을 감소시켰다. 그리고 메시지 변수의 사전 계산으로 인해 최장지연경로를 생성하지 않게 하였고, 단계 연산을 두 단계의 파이프라인 구조로 설계하여 한 클럭으로 한 단계 연산을 수행하는 방식보다 클럭 수는 한 개 증가했지만 최장지연경로를 약 1/2로 줄여 성능을 약 두 배 향상시켰다. 그리고 외부 호스트 프로세서와 해쉬 프로세서간의 입출력 과정과 해쉬 프로세서의 연산 과정을 병렬로 수행하여, 입출력 시간에 의해 생기는 성능 저하를 제거하였다.

또한 이 해쉬 프로세서는 32 비트의 PCI 인터페이스를 지원하며, 외부 호스트 프로세서에 대한 해쉬 보조 프로세서 형태로 설계하여 가상 사설망, 침입 차단 시스템 등에서 해쉬 함수 가속기로 사용될 수 있을 것으로 판단된다. 그리고 본 논문에서 구현한 해쉬 프로세서의 성능은 0.5 um CMOS 공정을 사용할 경우 최대 100 MHZ의 동작 주파수가 예상되며, 이때의 성능은 624 Mbps로 현재 상용화된 국내외의 어느 해쉬 프로세서보다도 빠른 처리 시간이다.

향후에는 현재 본 논문에서 구현한 해쉬 프로세서에 공개키 암호 알고리즘과 비밀키 암호 알고리즘 연산 모듈을 추가하여 단일 칩에 구현하는 통합 암호 프로세서를 구현할 예정이다.

참 고 문 헌

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

[2] 한국정보통신기술협회, "해쉬함수 표준 - 제2부 :해쉬 함수 알고리즘(HAS-160)", 1998년 11월.

[3] Benjamin Jun and Paul Kocher, "The Intel Random Number Generator", CRYPTOGRAPHY RESEARCH, INC., 22 April 1999.

[4] Ronald L. Rivest, "The MD4 Message Digest Algorithm", *Advances in Cryptology - Crypto '90*, pp. 303-311, 1991.

[5] SCI-WORX, "High Speed SHA-1 Hash Engine", http://www.sci-worx.com/internet/designobjects/do_list/handouts/cryptography/sha-1_ho.pdf.

[6] ALATEK, "ALATEK ALSHA IP Core Application Note", http://www.alatek.com/files/ALSHA.pdf

[7] Tality, "Hashing Core (HASH)", http://www.alatek.com/solutions/ip/docs/hash_flyer.pdf

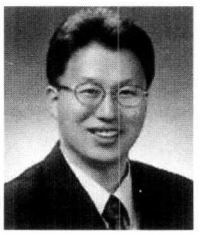
전 신 우(Shin-woo Jeon) 준회원

1996년 2월 : 광운대학교 전자공학부 졸업

2000년 2월~현재 : 광운대학교 전자공학과 석사과정

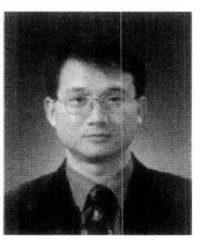
<주관심 분야> 통신용 칩 설계, 무선통신 ,정보보호

김 남 영(Nam-young Kim) 정회원



1987년 2월 : 광운대학교 전자공학과졸업
1991년 2월 : 미국뉴욕주립대학 (SUNY at Buffalo) 전자공학과 석사
1994년 2월 : 미국뉴욕주립대학 (SUNY at Buffalo) 전자공학과 박사
1994년 9월~현재 : 광운대학교 전자공학과 부교수
1998년 5월~현재 : 광운대학교 RFIC 교육센터 센터장
2000년 1월~현재 : 한국통신학회 학술지 편집위원
<주관심 분야> RFIC, MMIC, ASIC

정 용 진(Yong-jin Jeong) 정회원



1983년 2월 : 서울대학교 제어계측공학과 졸업
1983년 3월~1989년 8월 : 한국전자통신연구원
1991년 5월 : 미국 UMASS 전자전산공학과 석사
1995년 2월 : 미국 UMASS 전자전산공학과 박사
1995년 4월~1999년 2월 : 삼성전자 반도체 수석 연구원
1999년 3월~현재 : 광운대학교 전자공학부 조교수
<주관심 분야> 컴퓨터 연산 알고리즘, ASIC 설계, 무선 통신, 정보보호