FPGA를 이용한 128-비트 암호 알고리듬의 하드웨어 구현※

이 건 배†·이 병 욱††

요 약

본 논문에서는 미국 국립표준기술연구소에서 차세대 표준 암호 알고리듬으로 선정한 Rijndael 암호 알고리듬과 안정성과 성능에서 인정을 받은 Twofish 암호 알고리듬을 ALTERA FPGA를 사용하여 하드웨어로 구현한다. 두가지 알고리듬에 대해 키스케줄링과 인터페이스를 하드웨어에 포함시켜 구현한다. 알고리듬의 효율적인 동작을 위해 키스케줄링을 포함하면서도 구현된 회로의 크기가 크게 증가하지 않으며, 데이터의 암호/복호화 처리 속도가 향상됨을 알 수 있다. 주어진 128-비트 대칭키에 대하여, 구현된 Rijndael 암호 알고리듬은 11개의 클럭 만에 키스케줄링을 완료하며, 구현된 Twofish 암호 알고리듬은 21개의 클럭 만에 키스케줄링을 완료한다. 128-비트 입력 데이터가 주어졌을 때, Rijndael의 경우, 10개의 클럭 만에 주어진 데이터의 암호/복호화를 수행하고, Twofish는 16개의 클럭 만에 암호/복호화를 수행한다. 또한, Rijndael은 336.8Mbps의 데이터 처리속도를 보이고, Twofish는 121.2Mbps의 성능을 보임을 알 수 있다.

Hardware Implementation of 128-bit Cipher Algorithm Using FPGA

Keon-Bae Lee†·Byung-Wook Lee††

ABSTRACT

This paper implements two cipher algorithms, Rijndael and Twofish, with a hardware using the ALTERA FPGA. The former was selected as the AES (Advanced Encryption Standard) by NIST, and the latter was proven to have excellent stability and performance. Two algorithms are implemented in hardware with key scheduling and interface part. The size of the implemented circuit does not increase significantly even if it included the key scheduling for the efficient operation of algorithms. Thus, the throughput of encryption/decryption has been improved. The implemented Rijndael cipher algorithm has completed the key scheduling of a 128-bit symmetric key in 11 clocks, and Twofish algorithm in 21 clocks. Encryption/decryption has been performed in 10 clocks for Rijndael and in 16 clocks for Twofish with the same input data of 128-bit long. It is shown that the throughput of Rijndael is 336.8Mbps and that of Twofish is 121.2Mbps.

키워드 : 차세대 암호표준(AES), Rijndael, Twofish, 암호/복호화(encryption/decryption), 대칭키 암호 알고리듬(symmetric cipher algorithm), FPGA

1. 서 론

지난 1977년 IBM이 개발하여 미국 정부에 의하여 수정된 후 미 연방정부의 표준으로 채택되어 최근까지 사용되고 있는 64-비트 DES 암호 알고리듬은 블록 암호의 핵심기술인 S-box 설계 원리를 공개하지 않아서 구성 원리를 파악하기 어려웠으며, 최근 컴퓨터 처리 기술의 급속한 발달로 인해 암호가 해독되는 등 보안, 관리상의 문제점 및 취약성이 발견되었다. 이에, 미국 국립표준기술연구소(NIST, National Institute Standards & Technology)에서는 1997년 1월부터 DES 암호 알고리듬을 대체할 차세대 표준 암호 알고리듬(AES, Advanced Encryption Standard) 선정 프로젝트를 진행하였다. NIST는 차세대 표준 암호 알고리듬으로 128-비트 블록 암호 알고리듬, 다양한 길이의 키(128, 192, 256-비트)를 사용할 수 있는 알고리듬, 취약키를 갖지 않는 암호 알고리듬, 소프트웨어와 하드웨어 상에서 효율적인 암호 알고리듬 및 스마트카드 상에서도 동작할 수 있는 암호 알고리듬 등의 기준을 제시하였다.

AES 선정 프로젝트는 1997년 1월부터 전 세계적으로 후보 알고리듬들을 공모하기 시작하였으며 1, 2차 라운드를 거치면서 다섯 개의 후보 알고리듬(Rijndael, Twofish, MARS, RC6, Serpent)을 선정하였고, 다섯개의 후보 알고리듬들 중에서 NIST 자체평가 및 전 세계적인 공개 검증을 통하여 지난 2000년 10월 최종적으로 벨기에의 암호학자인 Joan Daemen과 Vincent Rijmen이 개발한 Rijndael 암호 알고리

※ 이 논문은 1997년 경기대학교 해외파견 연구비에 의하여 연구되었음.
† 정 회 원 : 경기대학교 전자·기계공학부 전자공학전공 교수
†† 준 회 원 : 경기대학교 대학원 전자공학과 석사과정
논문접수 : 2001년 2월 6일, 심사완료 : 2001년 5월 25일

듬을 AES로 선정하여 미 연방정부의 표준 암호 알고리듬으로 채택될 예정이다[1].

최근, 증가하는 정보 보안의 요구에 대해 소프트웨어적으로 암호/복호화를 처리하는 경우 그 한계가 존재한다. 즉, 고성능의 CPU를 사용하는 경우에는 만족할 만한 수준의 데이터 처리 속도를 얻을 수 있으나, 암호/복호화 기능이 필요한 시스템 내부에 고성능 CPU를 탑재하기 어려운 경우에는 소프트웨어적으로 처리하는데 한계성을 갖게 된다. 예로서, 세톱박스 내에서 데이터의 암호/복호화를 처리하기 위해서는 암호/복호화 기능을 전담하는 하드웨어의 사용이 필수적이라 하겠다. 암호 알고리듬을 하드웨어로 구현하는 경우, 멀티미디어와 같은 여러 분야에서 시스템 내부에 암호/복호화 하드웨어 코어와 인터페이스를 내장시킴으로써, 비디오, 오디오 데이터의 효율적인 암호/복호화 처리가 가능하다.

본 논문에서는 차세대 표준 암호 알고리듬으로 채택되어 전 세계적으로 널리 사용될 Rijndael 암호 알고리듬에 대하여 128-비트 키스케쥴링과 인터페이스를 포함한 암호/복호화 하드웨어 코어를 구현하고, AES 선정 프로젝트에서 암호의 안정성과 성능에서 인정을 받은 Twofish 암호 알고리듬에 대해서도 키스케쥴링과 인터페이스를 포함한 암호/복호화 하드웨어 코어를 ALTERA FPGA를 사용하여 구현한다. 키스케쥴링과 인터페이스를 포함하여 구현한 하드웨어의 동작이 기존의 방법들에 비해 처리 속도가 향상된 결과를 보임을 알 수 있다. 특히, 보안의 효율성을 증가시키기 위해 주기적으로 대칭키를 변경시키는 경우, 데이터의 처리 속도의 증가가 예상된다.

본 논문의 구성으로는 제2장에서는 Rijndael, Twofish 암호 알고리듬을 소개하고, 제3장에서는 Rijndael, Twofish 암호 알고리듬의 주요 모듈 구현 방법에 대하여 설명하고, 제4장에서는 Rijndael, Twofish 암호 알고리듬의 하드웨어 구현 결과 및 성능 비교에 대하여 제시한다.

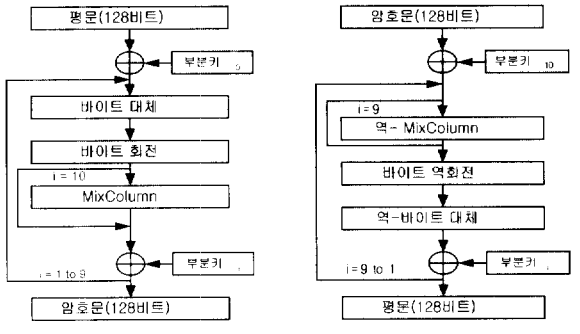
2. 암호 알고리듬

2.1 Rijndael 암호 알고리듬

Rijndael 암호 알고리듬은 대칭키 블록암호 알고리듬으로서 대칭키 길이와 암호/복호화의 기본 단위인 블록의 크기를 128, 192, 256-비트 중에서 선택할 수 있는 알고리듬이다. 이는 대칭키의 길이와 블록의 크기를 조합한 9가지의 다양한 선택을 가능하게 한다. 또한, 대칭키의 길이와 블록의 크기에 따라 10, 12, 14라운드를 갖는 Substitution-Linear Transformation 네트워크 구조이다[2].

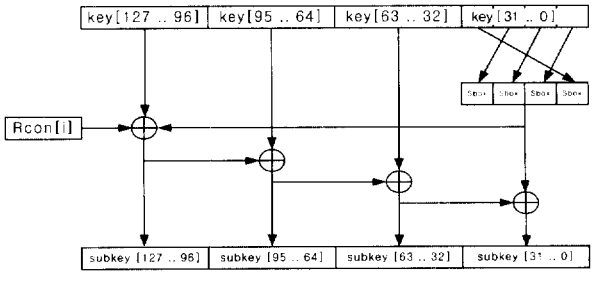
(그림 1)은 Rijndael 암호/복호화의 기본 구조를 나타낸다. Rijndael의 라운드 변환은 각각의 역변환이 가능한 바이트 대체(byte substitution)(8×8-비트 S-box), 바이트 회전(byte rotation), MixColumn, 부분키 modulo-2 덧셈(bit-wise XOR) 등의 크게 4개의 변환 모듈들로 구성되어 있으며, 각각의 연산은 Galois Field(2^8)내의 요소를 나타내는 바이트 및 4-바이트 워드 단위로 정의된다.

암호화의 과정은 128-비트 평문, 16-바이트들이 바이트 단위의 4×4-바이트 행렬로 배열되어 키스케쥴링에 의해 생성된 첫 번째 128-비트 부분키와 bit-wise XOR 연산이 되어 라운드 변환으로 넘어간다. 라운드 변환에서는 각각의 바이트들이 동일한 구조의 8×8-비트 S-box를 거쳐 바이트 대체를 수행하게 되며, 행들은 라운드의 수에 따라 고정된 바이트 수만큼 회전되어 바이트 회전 연산을 수행하게 된다. 재구성된 행렬의 열들은 GF(2^8)상의 고정된 행렬 곱셈인 MixColumn 연산을 수행하여, 128-비트의 부분키와 bit-wise XOR 연산을 수행하게 된다. 이와 같은 라운드 변환은 동일하게 9번 반복하게 되며, 맨 마지막 10번째 라운드 변환에서는 MixColumn을 제외한 라운드 변환내의 연산들을 수행하여 128-비트의 암호문을 생성하게 된다. 복호화의 과정은 암호화의 과정을 역순으로 수행함으로써 이루어지는데 복호화의 라운드 변환내의 함수들은 암호화의 라운드 변환내의 함수들의 역변환 즉, 바이트 대체는 역-바이트 대체, 바이트 회전은 바이트 역회전, MixColumn은 역-MixColumn들로 구성되며 부분키의 적용 순서 또한 역순으로 이루어진다.



(가) 암호화 (나) 복호화 알고리듬의 블록도

(그림 1) Rijndael의 (가) 암호화, (나) 복호화 알고리듬의 블록도



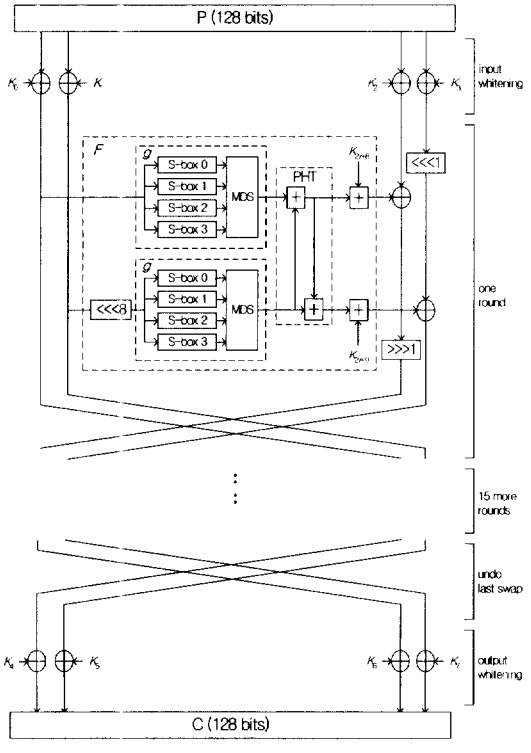
(그림 2) Rijndael 키스케쥴링 블록도

암호/복호화 과정에서 필요한 128-비트의 부분키 들은 라운드 수 보다 하나 더 많은 11개가 필요하며, 첫 번째 부분키는 주어진 128-비트의 키로 이루어지며 나머지 부분키 들은 (그림 2)와 같은 구조의 연산을 반복 수행함으로써 얻어진다. Rcon[i]는 라운드 상수로 $GF(2^8)$상의 $x^{(i-1)}$ 값이다.

2.2 Twofish 암호 알고리듬

Twofish 암호 알고리듬은 대칭키 블록 암호 알고리듬으로서, 블록의 크기는 128-비트이며 대칭키의 길이는 128, 192, 256-비트 중에서 선택할 수 있는 알고리듬이다. Twofish 알고리듬은 입력과 출력의 whitening 과정을 포함하며, 라운드 함수의 출력이 한 비트 회전된다는 점을 제외한 16라운드의 유사한 Feistel 네트워크 구조이다[3].

(그림 3)은 Twofish 암호/복호 알고리듬의 기본 구조를 나타낸다. Twofish의 라운드 함수는 킷값에 종속된 8×8-비트 S-box, $GF(2^8)$상에서의 MDS(Maximum Distance separable) 행렬 곱셈기, Pseudo-Hadamard-Transform, 32-비트 modulo 2^{32} 덧셈기로 구성된다.



(그림 3) Twofish 암호/복호 알고리듬의 블록도

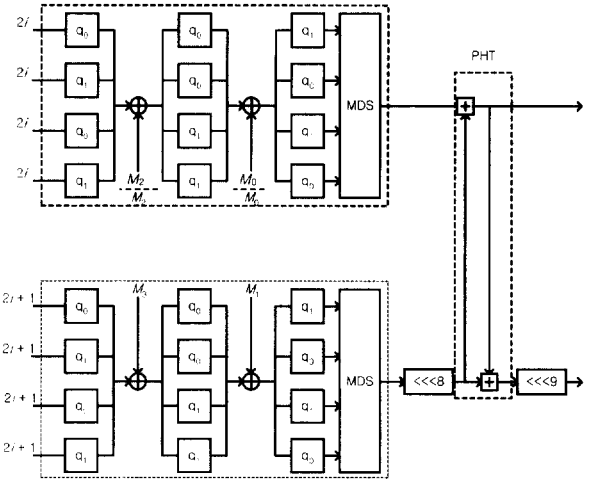
암호화의 과정은, 128-비트 평문이 little-endian convention으로 입력되어 4개의 32-비트 부분키와 bit-wise XOR 되어 입력 whitening 과정을 수행함으로써 시작된다. 각 라운드 함수에서는 좌측 두 개의 워드가 키 값에 의존하는 S-box, MDS 행렬 곱셈기를 거쳐 Pseudo-Hadamard-Transform을 이용하여 결합되고, 2개의 부분키와 32-비트 modulo 2^{32} 덧셈에 의해 더해진다. 이 라운드 함수의 두 출력은 우측 2개의 워드와 bit-wise XOR 및 1-비트 회전된다. 좌측 2개 워드와 라운드 함수의 출력과 bit-wise XOR된 우측의 두 워드는 다음 라운드를 위하여 자리바꿈 한다. 이와 같은 라운드 함수는 동일하게 16번 반복하게 되며 마지막 라운드 함수의 결과가 다시 자리바꿈 되고, 4개의 32-비트 부분키와 bit-wise XOR되어 출력 whitening을 거쳐 128-비트 평문에 적용한 little-endian convention과 같은 방법으로 128-비트의 암호문을 생성한다. 복호화 과정은 암호화 과정에서 적용한 40개의 부분키의 적용 순서와 라운드 함수의 출력이 우측 두 개의 워드와 bit-wise XOR 및 한 비트 회전되는 과정을 역으로 적용하여 이루어진다.

키스케줄링 부분은 크게 두 부분으로 나눌 수 있다. 첫 번째는 입력되는 128-비트를 사용하여 킷값에 종속된 S-box들을 만들기 위하여 S-box들에서 사용되는 S_0, S_1을 생성해 주는 부분이다. S_0, S_1은 $GF(2^8)$상에서의 Reed-Solomon 곱셈에 의해 생성되며 다음과 같은 수식으로 표현된다.

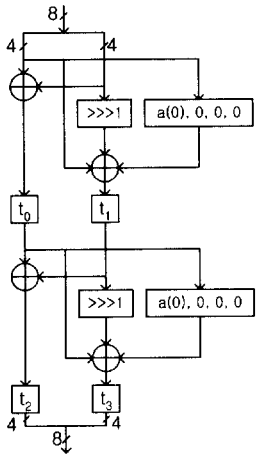
$$\text{Primitive Polynomial} : x^8 + x^6 + x^3 + x^2 + 1 ,$$

$$RS = \begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & 5A & 58 & DB & 9E & 03 \end{pmatrix}$$

두 번째 부분은 각각의 라운드 및 입력/출력 whitening 부분에서 사용될 40개의 32-비트 부분키를 생성하는 부분이다. 이 부분에서는 (그림 3)의 암호/복호화 부분의 g-함수와 PHT 구조에 추가적인 고정된 회전으로 이루어지기 때문에 암호/복호화의 g-함수와 PHT을 그대로 이용할 수 있다. (그림 4)와 (그림 5)는 Twofish 키스케줄링 및 S-box



(그림 4) Twofish 키스케줄링 블록도



(그림 5) q-치환

내의 고정된 8×8-비트 q-치환에 대하여 각각의 상세한 블록도를 나타내고 있다.

3. 암호 알고리듬의 주요 모듈 구현

본 논문에서 구현한 128-비트 Rijndael 알고리듬과 128-비트 Twofish 알고리듬 내에서 사용되는 주요 모듈들을 ALTEAR FPGA 상에서 구현하는 방법에 대하여 기술하면 다음과 같다.

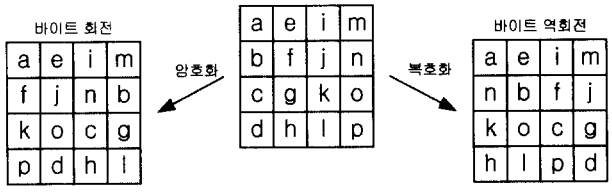
3.1 Rijndael 암호 알고리듬의 주요 모듈 구현

3.1.1 바이트 대체 / 역-바이트 대체

바이트 대체는 암호/복호화에 사용될 각각의 8 × 256-비트 두 종류의 S-box들이 필요하며, 구현되는 S-box의 개수에 따라 전체 성능이 좌우된다. 즉, 16개의 S-box들을 구현한 경우 한 라운드 1 클럭 방식으로 라운드 변환의 연산이 가능하며 8개의 S-box들을 구현하였을 때에는 한 라운드 2 클럭 방식으로 라운드 변환의 연산이 이루어져 전체 구현 면적은 줄지만 전체 성능은 반으로 줄어들게 된다. 또한, 키스케줄링을 포함한 암호/복호화 코어를 같이 구현하였을 경우에는 키스케줄링을 위한 4개의 S-box, 암호화를 위한 16개의 S-box와 복호화를 위한 16개의 S- box들이 필요하게 되어 총 36개의 S-box들이 필요하게 된다. 키스케줄링 부분에서 사용되는 S-box들은 암호화에서 사용되는 S-box들과 동일하기 때문에 암호화를 위해 사용되는 S-box들을 공유하여 구현될 수 있다. ALTERA FLEX-10KE 계열의 디바이스 내에는 4096-비트의 메모리 블록들을 갖기 때문에 메모리 블록마다 16×256-비트의 구조로 암호/복호화에 사용될 S-box 16개들을 같이 구성한다.

3.1.2 바이트 회전 / 바이트 역회전

바이트 회전은 별도의 연산과정 없이 바이트들간의 재배열로 이루어지며, ALTERA FPGA내의 논리 셀을 사용하지 않고 배선패턴의 사용으로만 구현된다. (그림 6)에 암호/복호화에 따른 바이트들의 재배열을 나타낸다.



(그림 6) Rijndael 바이트 회전 / 바이트 역회전

3.1.3 MixColumn / 역-MixColumn

MixColumn과 역-MixColumn은 GF(2^8)상에서의 고정된 4×4-바이트 행렬 곱셈으로 다음과 같은 수식들로 표현되며, GF(2^8)상에서의 곱셈기를 직접 구현하는 것이 필요할 것 같으나 행렬을 잘 살펴보면 여러 개의 modulo-2 덧셈(XOR) 또는 다입력 modulo-2 덧셈(XOR)으로 간단하게 구현할 수 있다[4]. <표 1>에 GF(2^8)상에서의 곱셈기를 다입력 modulo-2 덧셈(XOR)으로 표현하였다. ALTERA FPGA 내의 논리 셀은 4입력 1출력의 LUT(Loop Up Table)을 갖고 있기 때문에 4입력 XOR 까지는 하나의 논리 셀을 이용하여 구현될 수 있다.

Primitive Polynomial : $x^8 + x^4 + x^3 + x + 1$,

$$\begin{pmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix},$$

$$\begin{pmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \cdot \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix}$$

<표 1> Rijndael 암호 알고리듬 내의 GF(2^8)상에서의 곱셈기

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Primitive Polynomial				x	x		x	x
0x02	6	5	4	73	72	1	70	7
0x03	76	65	54	743	732	21	710	70
0x09	74	763	7652	6541	7530	762	651	50
0x0B	764	7653	76542	765431	5320	7621	76510	750
0x0D	754	7643	6532	75421	765310	620	751	650
0x0E	654	7543	6432	5321	65210	610	50	765

3.2 Twofish 암호 알고리듬의 주요 모듈 구현

3.2.1 키 값에 종속된 S-boxes

(그림 3)과 (그림 4)에서 보인 각 8×8-비트 S-box는 3개의 q-치환과 XOR로 구성된다. 고정된 8×8-비트 q-치환

(q_0, q_1)들은 8개의 다른 4×4-비트 치환들 (t_0, t_1, t_2, t_3)에 의하여 구성된다. 이 4×4-비트 치환들은 단지 4-비트 입력 함수를 형성하기 때문에 논리 셀 내의 4-비트 입력, 1-비트 출력 LUT 4개를 병렬로 이용하여 구현 될 수 있으며, q-치환내의 나머지 연산(XOR)들은 16개의 논리 셀 들로 구현될 수 있어 하나의 q-치환을 구현하는데는 32개의 논리 셀 들을 사용하여 구현할 수 있다. 하나의 S-box를 구현하는데는 112개의 논리 셀 들을 사용하여 구현 될 수 있다.

3.2.2 MDS(Maximum Distance Separable) 행렬 곱셈기

Twofish 내의 MDS 행렬 곱셈기 또한 $GF(2^8)$상에서의 고정된 4×4-바이트 행렬 곱셈기로서 고정된 3개의 피연산자 만을 가지고 있으며, 다음과 같은 수식으로 표현된다. 구현은 Rijndael 암호 알고리듬의 MixColumn/역-MixColumn과 마찬가지로 여러 개의 modulo-2 덧셈(XOR) 또는 다입력 modulo -2 덧셈(XOR)으로 간단하게 구현할 수 있다[5]. <표 2>에 Towfish 암호 알고리듬에서 사용되는 $GF(2^8)$상의 곱셈기를 다입력 modulo-2 덧셈(XOR)으로 표현된다.

$$\text{Primitive Polynomial}: \; x^8 + x^6 + x^5 + x^3 + x + 1 \,,$$

$$\begin{pmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \cdot \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix}$$

〈표 2〉 Twofish 암호 알고리듬내의 $GF(2^8)$상에서의 곱셈기

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Primitive Polynomial		x	x		x		x	x
0x5B	71	60	751	6410	530	421	310	20
0xEF	710	760	76510	6541	5430	43210	3210	210

3.2.3 32-비트 modulo 2^{32} 덧셈

32-비트 가산기 구현에서는 복잡하고 많은 논리 셀 들을 필요로 하지만 속도를 향상시킬 수 있는 CLA(Carry-Lookahead Adder), CSA(Carry Select Adder) 등으로 구현할 수 있다. 그러나, ALTERA FPGA내의 인접한 논리 셀 들간에 carry chain 이라는 빠른 전용 연결선을 이용하여 32-비트 CRA (Carry Ripple Adder)를 구현함으로써 속도를 향상시킴과 동시에 논리 셀 수를 줄일 수 있다.

3.2.4 RS 곱셈기[6]

$GF(2^8)$상에서 RS 곱셈기의 구현은 semi-systolic 어레이를 사용하여 LSB-First 알고리듬, MSB-First 알고리듬 등으로 구현할 수 있다. 그러나, LSB-First 알고리듬이 MSB-First 알고리듬 보다 임계 경로상의 지연시간이 적어 더 효율적이다.

4. 128-비트 암호 알고리듬의 하드웨어 구현

4.1 128-비트 암호 알고리듬의 하드웨어 구현 방식

암호/복호화 과정에서 Rijndael 암호 알고리듬은 10라운드의 반복 과정을 수행하고, Twofish 암호 알고리듬은 16라운드의 반복 과정을 수행한다. Rijndael 알고리듬은 각 라운드에서 각각 11개의 128-비트 부분키를 사용하며, Twofish 알고리듬은 42개의 32-비트 부분키를 사용하는데, 이들 부분키들은 키스케쥴링 단계에서 생성된다.

부분키를 데이터 블록의 암호/복호화 과정에서 매번 계산하여 사용하는 방식을 on-the-fly 방식[7]이라 하며, 이 방식은 부분키를 저장하지 않기 때문에 별도의 메모리가 필요하지 않으므로, 구현된 회로의 크기가 작아지는 장점이 있으나, 대칭키가 변경되지 않더라도 매번 부분키를 반복 계산해야 하는 단점을 가지고 있다. 참고문헌 [7]은 이러한 방식을 사용하고 있다.

주어진 대칭키에 대한 부분키를 생성한 후 하드웨어 내의 메모리에 저장하여 사용하는 경우에는 별도의 메모리 공간이 필요하지만, 일단 계산된 부분키를 반복 사용함으로써 매번 재계산할 필요가 없다.

부분키를 생성하는 방법으로는 키스케쥴링 과정을 하드웨어에 포함시켜 구현하는 경우와 별도의 키스케쥴링 과정에 의해 계산된 부분키를 하드웨어 내부의 메모리에 저장하여 사용하는 방식[4]이 있다. 참고문헌 [4]는 구현된 하드웨어 내부에 키스케쥴링을 포함하지 않고 별도의 과정을 거쳐 계산된 부분키를 사용한다. 즉, 하드웨어적으로 부분키 생성 과정을 처리하지 않고 외부의 별도 프로그램에 의해 부분키를 계산하였다고 가정하고, 메모리에 저장하여 사용하는 방식이다. 이 방식은 대칭키가 고정된 경우와 같이 외부에서 계산된 부분키를 계속 사용할 때 유리하지만, 대칭키가 변경될 때마다 암호/복호화 과정을 중단하고, 별도의 외부 과정에 의해 부분키를 재 계산하여 메모리에 저장하여 사용해야 한다.

데이터의 보안 향상을 위해 대칭키를 고정시키지 않고, 일정한 주기로 갱신하여 사용하는 것이 일반적이다. 대칭키가 고정된 경우에는 부분키의 재 계산이 불필요하여 한번 계산하면 영구적으로 사용 가능하지만, 보안의 효율면에서 비현실적이다. 따라서, 주기적으로 변경되는 대칭키에 대해 하드웨어 내부에서 부분키를 계산, 저장하여 사용하는 방식이 효율적이다.

본 논문에서는 부분키를 생성하는 키스케쥴링 과정을 하드웨어 내부에 포함하여 구현하고, 계산된 부분키들을 메모

리에 저장하여 사용하는 방식을 채택한다. 이는 일정한 주기로 변경되는 대칭키에 대해서 암호/복호 과정에서 부분키를 계산하여 사용하는 방식으로서, 대칭키가 자주 변경되는 경우에 하드웨어 내부에서 효율적으로 처리가 가능하다.

하드웨어 구현된 암호/복호화 알고리듬은 하드웨어 내부에서 128-비트 블록 단위로 동작한다. 즉, 128-비트 Rijndael 알고리듬과 128-비트 Twofish 알고리듬은 하드웨어 내부에서 128-비트 블록 단위로 데이터를 암호/복호화 하지만, 처리된 데이터를 PCI와 같은 인터페이스를 통해 하드웨어 외부로 전송하는 경우, 32-비트 단위로 나누어 전송하는 인터페이스가 필요하다. 이는 구현된 암호/복호화 하드웨어를 암호/복호화를 필요로 하는 응용 시스템과 결합하기 위해 인터페이스가 필수적이다.

본 논문에서는 참고문헌 [4]와 같이 32-비트 인터페이스를 암호/복호화 하드웨어 코어에 포함시켜 구현함으로써, 구현된 하드웨어 코어를 응용 시스템에서 사용 가능하게 한다.

본 논문에서는 각 라운드에서 사용되는 부분키 들을 저장하기 위한 메모리(RAM)와 Rijndael 암호 알고리듬에서 사용되는 바이트 대체/역-바이트 대체 부분을 제외한 모든 암호/복호화 부분과 키스케쥴링, 외부와의 인터페이스를 VHDL로 기술한다. 부분키를 저장하기 위한 메모리(RAM)와 바이트 대체/역-바이트 대체를 저장하기 위한 메모리(ROM)는 ALTERA FPGA내의 local memory (EAB)를 이용하여 구현하기 위해 ALTERA에서 제공하는 라이브러리(LPM : Library of Parameterized Module)를 사용한다.

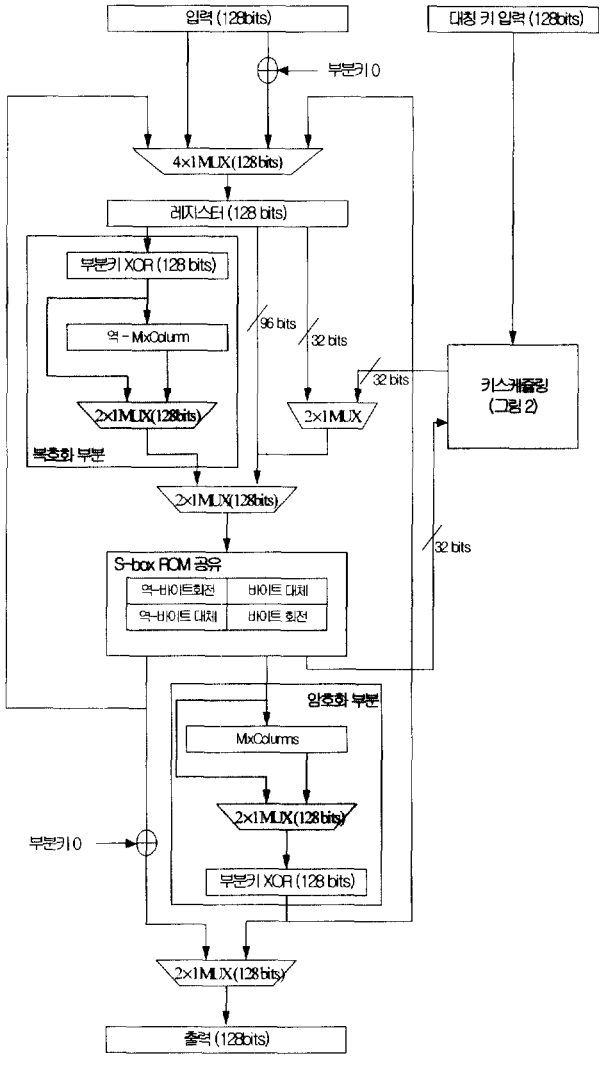
VHDL 코드는 암호 알고리듬 제안 문서에서 제공하는 테스트 벡터를 기반으로 검증되며, 검증된 VHDL 코드는 SYNOPSYS사의 FPGA EXPRESS를 사용하여 합성과 최적화를 수행한다. FPGA EXPRESS로부터 얻어진 netlist는 ALTERA MAX+PLUS-II에서 다시 컴파일 되고 배치/배선을 거쳐 구현되며, post-레이아웃 타이밍 시뮬레이션으로 구현된 알고리듬을 검증한다.

4.2 Rijndael 암호 알고리듬의 하드웨어 구현

Rijndael 암호 알고리듬의 하드웨어 구현은 (키스케쥴링 + 암호화)구조, (키스케쥴링 + 복호화)구조, (키스케쥴링 + 암호/복호화)구조 등의 세 가지 구조에 대하여 구현한다.

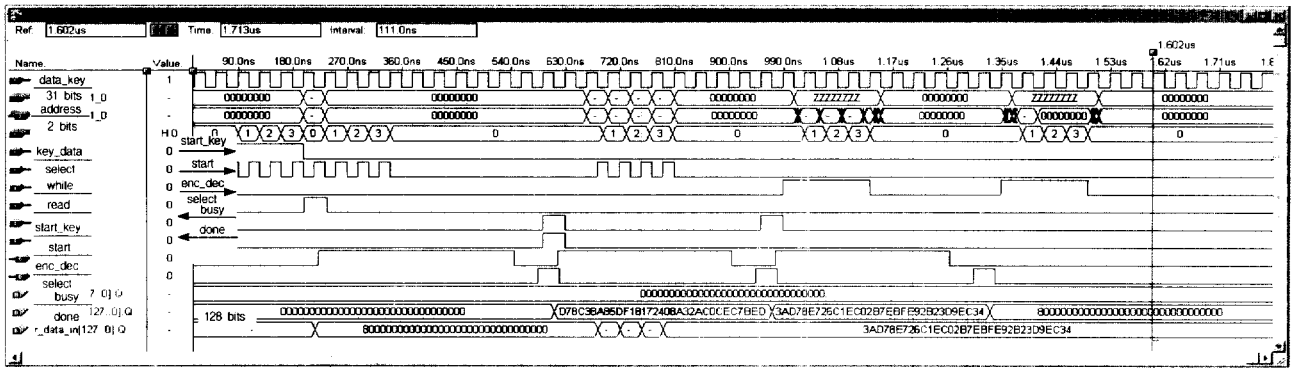
본 논문에서 구현한 Rijndael 알고리듬의 세가지 하드웨어 구조 중 키스케쥴링을 포함하는 암호/복호화 구조 즉, 하드웨어 코어의 블록도는 (그림 7)과 같다.

Rijndael 알고리듬은 암호화와 복호화 과정이 상이하여, Twofish와 같은 다른 암호 알고리듬에 비해 공통된 부분이 적으므로, 암호/복호화를 하드웨어로 구현하는 경우 회로 크기가 커지는 단점이 있다. (그림 7)과 같이, 키스케쥴링을 포함하는 암호/복호화 구조는 구현하는 과정에서 키스케쥴링에서 필요한 4개의 S-box를 암호화 부분에서 구현한 S-box를 같이 공유하여 사용함으로써 처리속도의 저하 없이 구현된 회로 크기의 증가를 억제하도록 구현한다.

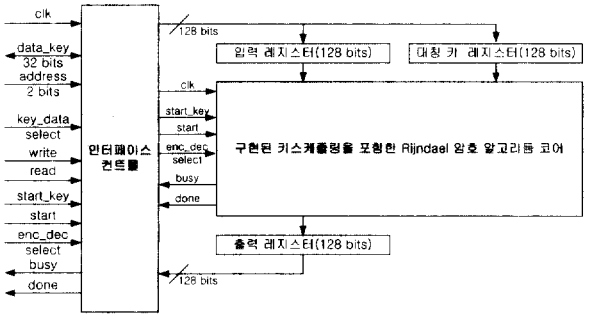


(그림 7) Rijndael 알고리듬 코어 블럭도

본 논문에서 구현한 Rijndael 암호 알고리듬의 전체 블록도는 (그림 8)과 같으며, 키스케쥴링을 포함하는 암호/복호화 구조를 코어라 하고, 코어와 외부의 데이터 교환을 위한 32-비트 인터페이스 회로가 포함되어 있다. 구현된 코어 내부에서는 128-비트 단위로 암호/복호 데이터가 처리되며, 인터페이스를 통해 코어 외부와 32-비트 단위로 데이터가 전송된다.



(그림 9) Rijndael 연속적인 키스케쥴링, 암호화, 복호화 타이밍 시뮬레이션



(그림 8) 인터페이스를 포함하는 Rijndael의 전체 블록도

세 가지 구조에 대한 암호/복호화 코어의 동작에 대한 설명은 다음과 같다.

구현된 Rijndael 암호 알고리듬 코어 세 가지의 구조에 대하여, 한 라운드 당 하나의 클럭을 사용하는 방식의 구현에 있어 라운드 연산에 10개의 클럭이 소요되고, 입력에 1개의 클럭, 출력에 1개의 클럭이 소요되므로, 총 12개의 클럭이 필요하다. 그러나, 입력 데이터에 대하여 부분키 덧셈을 적용함과 동시에 첫 번째 라운드 변환을 수행하고, 마지막 라운드 변환을 수행함과 동시에 출력을 발생시키면, 입력을 받아들이는데 필요한 하나의 클럭과 출력을 발생시키는데 필요한 하나의 클럭을 절약할 수 있다. (그림 9)는 이러한 방법을 사용하여 구현한 Rijndael 암호 알고리듬 코어의 타이밍도를 나타내고 있으며, 알고리듬 제안 문서와 함께 제공된 테스트 벡터를 기반으로 38nsec의 클럭 주기에 주어진 키와 평문, 암호문에 대하여 연속적인 키스케쥴링, 암호화, 복호화 타이밍 시뮬레이션을 한 결과를 보이고 있다. 주어진 키에 대하여 11개 클럭 만에 키스케쥴링을 완료하며 주어진 평문/암호문에 대하여 10개 클럭 만에 암호/복호화를 완료할 수 있다. "start" 및 "start_key" 신호가 "HIGH"일 때 입력 신호들(데이터 및 비밀키)은 샘플링되어 암호/복호화 또는 키스케쥴링을 시작한다. "enc_dec" 신호가 "HIGH"일 때 암호화, "LOW" 일 때 복호화를 수행한다. 암호/복호화 또는 키스케쥴링을 수행하는 중에는 "busy" 신호를 출력하며, 완료되면 "done" 신호를 출력한다. "done" 신호가 "HIGH"일 때 "start", "start_key" 신호를 인가하면, 결과를 출력함과 동시에 새로운 입력 또는 키를 받아들여 새로운 암호/복호화 또는 키스케쥴링을 수행한다.

본 논문에서 키스케쥴링과 인터페이스를 포함하는 세가지 구조로 하드웨어 구현한 Rijndael 암호 알고리듬의 회로 크기와 처리속도는 <표 3>과 같다. 이는 ALTERA FPGA 중에서 FLEX-10KE 계열의 24개의 메모리 블록을 갖는

〈표 3〉 Rijndael 암호 알고리듬의 하드웨어 구현 결과

구조	논리 셀	최대 동작주파수	처리속도
키스케쥴링 + 암호	1484	41.6 MHz	533.3 Mbps
키스케쥴링 + 복호	1918	27.7 MHz	355.5 Mbps
키스케쥴링 + 암호/복호	2819	26.3 MHz	336.8 Mbps

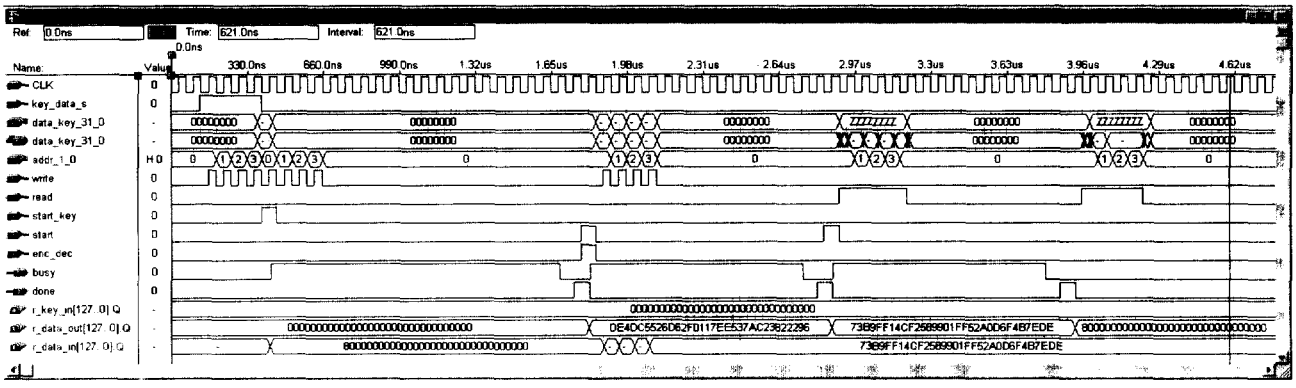
EPF10K200SRC240-1 디바이스에서 측정한 결과이다.

4.3 Twofish 암호 알고리듬의 하드웨어 구현

Twofish 암호 알고리듬의 구현에서는 키스케쥴링과 인터페이스를 포함한 구조를 구현한다. 부분키를 생성하는 키스케쥴링 부분은 암호/복호화 구조에서 사용하는 g-함수와 PHT을 공유하고 있으며, 암호/복호화가 동일한 라운드 함수를 갖기 때문에 암호/복호화 부분을 분리하여 구현하지 않는다.

구현한 Twofish 알고리듬에 대하여 입/출력 방법은 Rijndael과 동일한 방법으로 구현하여 키스케쥴링에 21개 클럭, 16라운드 암호/복호화에 16개의 클럭이 소요된다. (그림 10)은 Twofish 암호 알고리듬 제안 문서와 함께 제공된 테스트 벡터를 기반으로 연속적인 키스케쥴링, 암호화, 복호화의 동작을 66nsec의 클럭 주기로 타이밍 시뮬레이션 한 결과를 나타낸다.

본 논문에서 구현한 키스케쥴링과 인터페이스를 포함하는



(그림 10) Twofish 연속적인 키스케쥴링, 암호화, 복호화 타이밍 시뮬레이션

128-비트 Twofish 암호 알고리듬의 회로 크기와 처리속도는 <표 4>와 같다. 이는 ALTERA FPGA 중에서 FLEX-10KE 계열의 EPF10K130EQC240-1 디바이스에서 측정한 결과이다.

〈표 4〉 Twofish 암호 알고리듬의 하드웨어 구현 결과

구조	논리 셀	최대 동작주파수	처리 속도
키스케쥴링 + 암호/복호	3601	15.15 MHz	121.21 Mbps

4.4 성능 비교

〈표 5〉 Rijndael : 다양한 구조에 따른 처리속도 비교

논문	구 조	메모리 블록		논리 셀	처리 속도	FPGA 디바이스
참고문헌 [7]	키스케쥴링 + 암호 (인터페이스 제외)	키스케쥴링	4	1032	268.0 Mbps	EPF10K250AGC599-1
		암호(S-box)	16			
	키스케쥴링 + 복호 (인터페이스 제외)	키스케쥴링	4	2885	248.0 Mbps	
		복호(S-box)	16			
참고문헌 [4]	암호 (인터페이스 포함)	부분키 저장	8	1585	232.7 Mbps	FLEX10KE family (EPF10K130EQC240-1) SPEED GRADE : -1
		암호(S-box)	16			
	복호 (인터페이스 포함)	부분키 저장	8	2145	211.5 Mbps	
		복호(S-box)	16			
	암호/복호 (인터페이스 포함)	부분키 저장	8	3348	179.0 Mbps	
		암호/복호(S-box)	16			
본논문	키스케쥴링 + 암호 (인터페이스 포함)	부분키 저장	8	1484	533.3 Mbps	FLEX10KE family (EPF10K200SRC240-1) SPEED GRADE : -1
		암호(S-box)	16			
	키스케쥴링 + 복호 (인터페이스 포함)	부분키 저장	8	1918	355.5 Mbps	
		복호(S-box)	16			
	키스케쥴링 + 암호/복호 (인터페이스 포함)	부분키 저장	8	2819	336.8 Mbps	
		암호/복호(S-box)	16			

128-비트 Rijndael 암호 알고리듬에 대해 여러 가지 구조 및 구현 방법에 따른 처리속도의 비교는 <표 5>와 같다.

참고문헌 [7]의 방법은 암호/복호화 과정에서 매번 부분키를 생성하여 사용하는 방식으로 부분키를 저장하기 위한 메모리 블록을 사용하지 않는 장점은 있으나, 암호/복호화 처리속도의 효율이 저하되고 암호/복호부의 회로 크기가 커지는 단점을 가지고 있다. 일반적으로 대칭키는 매번 변경되는 것이 아니라, 한번 정해지면 일정한 주기를 갖고 사용되므로, 매번 키스케쥴링에 의한 부분키의 생성을 암호/복호화 과정에 포함시킬 필요가 없다. 따라서, 대칭키가 변경되는 경우, 필요 시 키스케쥴링 과정을 사용한 것이 바람직하다. 참고문헌 [7]의 방법은 외부와의 인터페이스가 구현되지 않은 구조이다.

참고문헌 [4]의 방법은 키스케쥴링을 구현하지 않고, 암호/복호화 과정에서 사용되는 부분키의 생성을 소프트웨어적으로 처리하거나, 별도의 과정을 통해 계산하여 미리 메모리에 저장하였다고 가정하는 방식을 사용하고 있으며, 이때, 별도로 계산된 부분키와 S-box를 저장하기 위해 24개의 메모리 블록을 사용하는 구조이다. 이는 별도의 키스케쥴링 과정이 있다고 가정하고 구현한 결과이므로, 대칭키의 변경시 하드웨어 내에서 키스케쥴링이 수행하지 못하는 단점을 가지고 있다. 또한, 보안의 효율을 증가시키기 위해 대칭키를 주기적으로 변경하는 경우, 반드시 부분키를 계산하여 메모리에 저장하는 과정이 별도로 수반되어야 한다. 참고문헌 [4]의 방법은 인터페이스가 구현된 구조이다.

본 논문에서는 매번 대칭키가 변경되지 않는 한, 암호/복호화 과정에 매번 키스케쥴링을 적용할 필요가 없으므로, 대칭키의 변경 시에만 키스케쥴링 부분에 의해 부분키를 계산하여 메모리에 저장하는 방식을 채택한다. 이는 키스케쥴링 과정에서 생성된 부분키와 S-box를 저장하기 위해 24개의 메모리를 사용하는 방식이며, 24개 이상의 메모리 블록을

포함하는 FPGA 디바이스를 선택하여 사용한다. 본 논문은 참고문헌 [4]와 같이 24개의 메모리 블록을 사용하여 부분키와 S-box를 저장하여 사용하는 방식이나, 키스케줄링 과정을 하드웨어 내에 포함시켜 구현하고, 인터페이스를 포함한 구조로 구현한다.

따라서, 본 논문에서 구현한 Rijndael 알고리듬의 하드웨어는 암호/복호화 및 키스케줄링, 인터페이스까지를 모두 포함하면서 참고문헌 [7](on-the-fly 방식, 키스케줄링 + 인터페이스 제외)에 비해 회로의 크기가 다소 증가하지만, 데이터의 처리 속도 면에서는 향상된 결과를 얻을 수 있고, 참고문헌[4](키스케줄링 제외 + 인터페이스)에 비해 회로 크기가 감소하면서, 데이터의 처리 속도 면에서도 향상된 결과를 보임을 알 수 있다. 이때, FPGA로 구현된 회로의 크기가 감소하면, 추후 ASIC으로 구현 시 컴팩트한 구현이 가능할 것으로 기대된다.

Twofish 암호 알고리듬에 대해 기존의 구현 방법과 비교한 결과는 <표 6>과 같다.

〈표 6〉 Twofish : 구현된 구조에 따른 처리속도 비교

논문	구 조	메모리 블록		논리 셀	처리 속도	FPGA 디바이스
참고문헌 [4]	암호 (인터페이스 포함)	부분키 저장	8	1950	81.5 Mbps	FLEX10KE family (EPF10K130EQC240-1) SPEED GRADE : -1
	복호 (인터페이스 포함)	부분키 저장	8	1935	81.5 Mbps	
	암호/복호 (인터페이스 포함)	부분키 저장	8	2104	80.3 Mbps	
본 논문	키스케줄링+ 암호/복호 (인터페이스 포함)	부분키 저장	8	3601	121.2 Mbps	

참고문헌 [4]의 방법은 Rijndael 구현과 마찬가지로, 보안 능력의 향상을 위해 대칭키가 주기적으로 변경될 때마다 별도의 키스케줄링 과정에 의한 처리가 요구된다.

본 논문에서는 키스케줄링과 인터페이스를 모두 포함한 구조로 하드웨어를 구현한다. 구현된 하드웨어에 키스케줄링 처리부가 포함됨으로써, 회로의 크기가 다소 증가하지만, 암호/복호화 처리속도가 50% 정도 향상됨을 알 수 있다. 또한, 대칭키가 주기적으로 변경되는 경우에는 키스케줄링을 하드웨어 내부에서 처리하므로 더 효율적인 동작이 예상된다.

5. 결 론

본 논문에서는 128-비트 Rijndael 암호 알고리듬과 Twofish 암호 알고리듬에 대해서 키스케줄링과 인터페이스를 포함하여 하드웨어 코어를 설계, 구현하고 검증하였으며, ALTERA FPGA 상에서 구현하여 성능을 비교하였다.

본 논문에서는 128-비트 Rijndael과 128-비트 Twofish 암호 알고리듬을 구현하는 과정에서 키스케줄링을 하드웨어에 포함시켜 구현함으로써, 대칭키의 변경 시에만 키스케줄링에 의해 부분키를 계산하여 메모리에 저장하는 방식을 채택하였다. 이는, 매 라운드마다 부분키를 계산하여 사용하는 on-the-fly 방식에 비하여 데이터 암호/복호화의 처리 속도가 향상이 되었으며, 보안 능률을 향상하기 위해 주기적으로 변경되는 대칭키에 대해서도 부분키를 계산하는 과정이 하드웨어 내부에서 처리되므로, 별도의 과정이 필요하지 않다.

128-비트 Rijndael 알고리듬의 구현 결과, 하드웨어 내부에 키스케줄링과 인터페이스를 모두 포함하면서도 구현된 회로의 크기가 크게 증가하지 않았으며, 데이터의 암호/복호화 처리 속도가 향상됨을 알 수 있다.

128-비트 Twofish 알고리듬의 구현 결과, 키스케줄링의 포함하면서 회로의 크기가 증가하였으나, 데이터의 암호/복호화 처리 속도가 50% 정도 향상되었으며, 특히, 대칭키가 주기적으로 변경되는 경우, 더 큰 동작상의 효율이 예상된다.

본 논문에서 구현한 인터페이스를 포함하는 128-비트 Rijndael, Twofish 암호 알고리듬의 하드웨어는 대칭키 암호 기능을 필요로 하는 ASIC, 임베디드 시스템, 보안 기능의 디지털 카메라 시스템 등에 널리 사용될 수 있다.

앞으로, 대칭키 블록 암호 알고리듬의 운영 방식인 CBC, CFB, OFB 방식으로 하드웨어를 구현해야 할 것이다.

참 고 문 헌

[1] National Institute Standards & Technology,(http : //csrc.nist.gov/encryption/aes/).

[2] J. Daemen, V. Rijmen, "AES Proposal : Rijndael," (http : //csrc.nist.gov/encryption/aes/).

[3] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels, "Twofish : A 128 bit Block Cipher,"(http : //www.counterpane.com/twofish.html).

[4] Viktor Fischer, "Realization of the Round 2 AES Candidates using ALTERA FPGA," (http : //csrc.nist.gov/encryption/aes/).

[5] Pawel Chodowiec, Kris Gaj, "Implementation of the Twofish Cipher Using FPGA Devices," (http : //csrc.nist.gov/encryption/aes/).

[6] Surendar K. Jain, Leilei Song, Keshab K. Parhi, "Efficient Semisystolic Architectures for Finite-Field Arithmetic," IEEE Trans. VLSI Syst., Vol.6, pp.101-113, Mar. 1998.

[7] Piotr Mroczkowski, "Implementation of the block cipher Rijndael using Altera FPGA," (http : //csrc.nist.gov/encryption/aes/round2/pubcmnts.htm).

이 건 배

e-mail : kblee@kuic.kyonggi.ac.kr

1982년 한양대학교 전자공학과 졸업(학사)

1984년 한양대학교 대학원 전자공학과 졸업 (공학석사)

1989년 한양대학교 대학원 전자공학과 졸업 (공학박사)

1998년~1999년 미국 UCLA 방문연구교수

1991년~현재 경기대학교 전자공학전공 부교수

관심분야 : 암호보안, ASIC설계, CAD 및 VLSI 설계, GIS

이 병 욱

e-mail : bwlee@kuic.kyonggi.ac.kr

1999년 경기대학교 전자공학과 졸업

2000년~현재 경기대학교 대학원 전자공학과 석사과정 재학 중

관심분야 : ASIC 설계, 암호보안, IP, 하드웨어 설계