情報保護學會論文誌
第11券 第6號, 2001. 12

Rijndael 암호 알고리즘을 구현한 암호 프로세서의 설계*

전 신 우**, 정 용 진***, 권 오 준****

Design of Cryptographic Processor for Rijndael Algorithm

Shin-woo Jeon**, Yong-jin Jeong***, Oh-Jun Kweon****

요 약

본 논문에서는 AES(Advanced Encryption Standard)로 채택된 Rijndael 알고리즘을 구현한 암호 프로세서를 설계하였다. 암호화와 복호화를 모두 수행할 수 있으며, 128비트의 블록과 128비트의 키 길이를 지원한다. 성능과 면적 측면을 모두 고려하여 가장 효율적인 구조로 한 라운드를 구현한 후, 라운드 수만큼 반복하여 암복호화를 수행하도록 하였다. 대부분의 다른 블록 암호 알고리즘과 달리 암복호화 시 구조가 다른 Rijndael의 특성으로 인한 면적의 증가를 최소화하기 위해 ByteSub와 InvByteSub은 알고리즘을 기반으로 구현함으로써 메모리로만 구현하는 방법에 비해 비슷한 성능을 가지면서 필요한 메모리 양은 1/2로 줄였다. 이와 같이 구현한 결과, 본 논문의 Rijndael 암호 프로세서는 0.5um CMOS 공정에서 약 15,000개의 게이트, 32K-bit ROM과 1408-bit RAM으로 구성된다. 그리고 한 라운드를 한 클럭에 수행하여 암복호화 하는데 블럭 당 총 11클럭이 걸리고, 110MHz의 동작 주파수에서 1.28Gbps의 성능을 가진다. 이는 현재 발표된 논문들과 비슷한 성능을 가지면서 면적의 가장 큰 비중을 차지하는 메모리 양은 절반 이상 감소하여, 지금까지 발표된 논문 중 가장 우수한 면적 대 성능 비를 가지는 것으로 판단된다.

ABSTRACT

This paper describes a design of cryptographic processor that implements the Rijndael cipher algorithm, the Advanced Encryption Standard algorithm. It can execute both encryption and decryption, and supports only 128-bit blocks and 128-bit keys. As the processor is implemented only one round, it must iterate 11 times to perform an encryption/decryption. We implemented the ByteSub and InvByteSub transformation using the algorithm for minimizing the increase of area which is caused by different encryption and decryption. It could reduce the memory size by half than implementing with only ROM. We estimate that the cryptographic processor consists of about 15,000 gates, 32K-bit ROM and 1408-bit RAM, and has a throughput of 1.28 Gbps at 110 MHz clock, based on Samsung 0.5um CMOS standard cell library. To our knowledge, this offers more reduced memory size compared to previously reported implementations with the same performance.

keyword : *cryptography, block cipher, AES, Rijndael*

* 본 연구는 2001년도 광운대학교 산학연 공동기술개발 컨소시엄 사업의 지원에 의해 이루어졌습니다.
** 광운대학교 전자통신공학과 실시간 구조 연구실(swjun@explore.kwangwoon.ac.kr)
*** 광운대학교 전자공학부 조교수(yjjeong@daisy.kwangwoon.ac.kr)
**** 동의대학교 전산통계학과 전임강사(ojkwon@dongeui.ac.kr)

Ⅰ. 서 론

암호 알고리즘은 크게 암복호화 키가 서로 동일한 비밀키 암호 알고리즘과 두 키가 서로 다른 공개키 알고리즘으로 나뉘어진다. 이 중 비밀키 암호 알고리즘은 다시 블록 암호 알고리즘과 스트림 암호 알고리즘으로 구분된다. 대표적인 블록 암호 알고리즘으로는 1977년 미국 연방 표준으로 채택되어 지금까지 널리 사용되고 있는 DES(Data Encryption Standard)가 있다. 그러나 DES는 56비트의 짧은 키를 사용하므로 현재 컴퓨터의 계산 속도로 짧은 시간 안에 해독이 가능하다. 이와 같은 안전성에 대한 문제로 인해 NIST(National Institute of Standards and Technology)에서는 DES의 표준 기한이 만료되는 1998년을 기점으로 더 이상 표준으로 사용하지 않기로 결정하였고, 1997년에 DES를 대체할 새로운 표준 블록 암호 알고리즘을 선정하기 위해 AES(advanced Encryption Standard)의 공모를 발표하였다. 1998년 1차 심사대상으로 15개의 후보 알고리즘을 선정하였고, 1999년 15개의 후보 알고리즘에서 2차 심사대상으로 5개의 알고리즘(Mars, RC6, Rijndael, Serpent, Twofish)으로 압축하였으며, 이후 다시 최종 심사를 거쳐 2000년 10월 Rijndael을 최종 AES 알고리즘으로 채택하였다. Rijndael 알고리즘은 알려진 모든 공격에 강하고, 그 응용에 있어서 속도나 하드웨어 구현에 뛰어난 장점을 가지고 있어 앞으로 여러 분야에서 Rijndael 알고리즘의 활용이 활발해 질 것으로 예상된다.[1]

정보통신기술의 발달로 인해 고속 네트워크 환경에서 정보의 송수신에 암호 기술을 적용하기 위해서는 고속의 암호 처리 기술이 요구되어진다. 그러나 범용 프로세서를 바탕으로 한 소프트웨어 처리 기술로는 정보보호의 실시간 처리 서비스가 어려우므로, 고속의 암호 처리 성능을 가지는 Rijndael 암호 프로세서의 개발이 필요하다. 이에 본 논문에서는 Rijndael 알고리즘의 기본적인 구조와 특징을 분석하고, 하드웨어로 구현 시 처리속도와 면적을 고려하여 최적화된 구조를 제안한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 현재까지 발표된 Rijndael 알고리즘의 구현사례에 대해 알아보고, 3장에서는 Rijndael 알고리즘에 대해 소개한다. 4장에서는 Rijndael 암호 프로세서의 설계에 대해 다루고, 5장에서는 구현한 Rijndael 암호 프로세서의 성능을 분석한다. 그리고 6장에서 결론을 맺는다.

Ⅱ. 구현 사례

이 장에서는 최근 발표된 Rijndael 알고리즘의 구현 사례들에 대해 알아본다. 먼저 H. Kuo와 I. Verbauwhede[2]는 128, 192, 256비트의 블록과 키를 모두 지원하며, 암호화 동작을 수행하도록 Rijndael을 구현하였다. Rijndael은 크게 ByteSub, ShiftRow, MixColumn, 그리고 AddRoundKey의 네 개의 기능 블록으로 구성된다. ByteSub는 8×8비트의 S-box를 사용하였고, ShiftRow는 LUT(Look-up Table)을 이용하여 쉬프트 양을 결정하였으며, MixColumn은 [1]에서 제안한 Xtime 함수를 사용하여 구현하였다. 그리고 라운드 키 생성부는 온라인 계산 방식을 사용하여 라운드 키가 라운드 동작과 병행하여 계산되어지도록 하였다. 이를 0.18um 스탠다드 셀 라이브러리를 사용하여 합성 시 173,000게이트가 소요되고, 100MHz의 동작 주파수에서 1.82Gbps의 성능을 가진다.

M. McLoone과 J. V. McCanny[3]는 fully pipeline 구조를 사용하여 Xilinx Virtex FPGA에 Rijndael을 구현하였다. Pipeline 구조를 사용함으로써 성능을 향상시킬 수 있었지만, 대신 칩 면적이 증가하고 CBC 모드와 같은 feedback 모드에는 응용할 수 없는 단점이 있다. 암호화만 동작하도록 구현할 경우에는 128, 192, 256비트의 키를 모두 지원할 수 있게 하여 각각 7, 5.8, 5.1Gbps의 성능을 보인다. 그리고 암호화뿐만 아니라 복호화도 지원할 경우에는 복호화 시 사용되는 InvByteSub의 추가 구현으로 인해 암호화만 지원할 때보다 필요한 메모리의 양이 두 배로 증가한다. 그래서 [3]에서는 암호화와 복호화 동작 시 필요한 ByteSub와 InvByteSub의 값을 저장한 두 개의 ROM만을 추가로 구현하였고, ByteSub 값을 저장하고 있는 각 라운드에 배치된 모든 BRAM은 암호화 시에는 ByteSub 값을 저장한 ROM의 데이터를, 복호화 시에는 InvByteSub의 값을 저장한 ROM의 데이터를 읽어 초기화한 후 동작을 수행하도록 구현하였다. 이로 인해 면적의 증가를 최소화할 수 있으나 암복호화 과정이 바뀔 때 마다 ROM의 데이터를 읽어 BRAM을 초기화해야 하므로, 초기화할 때마다 256클럭이 소모되는 단점이 있다. 이와 같은 구조로 Xilinx Virtex-E XCV3200E-8-CG1156 FPGA

에 구현하면 128비트 키를 사용할 경우 25.3MHz의 동작 주파수를 사용하여 3.2Gbps의 성능을 보인다.

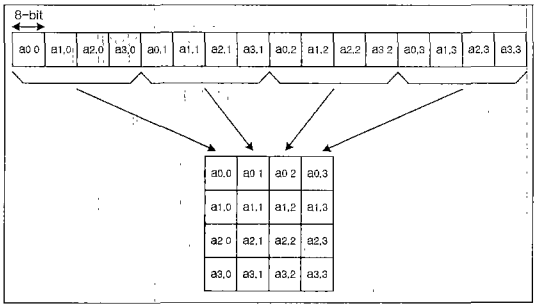
V. Fischer와 M. Drutarovsky[4]는 블록과 키의 길이는 128비트만 지원하고 암호화와 복호화를 모두 수행하는 것을 전제 조건으로 Rijndael을 구현하는 두 가지 방법을 제안하였다. 이는 8×8비트의 S-box를 이용하여 구현하는 방법과 8×32비트의 T-box를 이용하여 구현하는 방법으로 나뉜다. T-box를 이용하여 구현하면 S-box를 이용하는 방법에 비해 성능이 향상되지만 필요한 메모리의 양이 4배로 증가하게 되는 단점이 있다. 그래서 ATERA APEX FPD(Field Programmable Devices)를 타겟으로 구현 시, T-box를 이용할 경우에는 750Mbps, S-box를 이용할 경우에는 612Mbps의 성능을 보인다.

A. Rudra 등[5]은 composite field를 이용하여 Rijndael을 구현하였다. 이 경우 모든 연산이 composite field를 사용하여 계산되므로 암호화 전과 후에 추가로 데이터를 변환해 주는 연산이 필요하며, 모든 연산 과정들도 composite field상에서 계산되도록 변환하여 사용된다. 그리고 ByteSub는 메모리를 사용하지 않고 알고리즘을 이용하여 구현함으로써 면적을 최소화하였다. 그래서 128비트의 키만 사용하고 암호화만 수행하도록 구현할 경우, iterative 구조를 사용하여 4K 게이트만으로 Rijndael core 구현이 가능하다. 또, 512개의 I/O를 가지며 Rijndael core 32개가 동시에 수행되도록 구현하면, 0.5사이클마다 한 블록씩 암호화되어 32MHz의 동작 주파수에서 7.5Gbps의 성능을 가지고, 256K 게이트가 소요된다.

위와 같이 지금까지 발표된 논문들은 암호화만 구현하거나[2,5], 성능은 우수하지만 필요한 하드웨어 면적이 크거나[3,4], 필요한 하드웨어 면적은 작지만 성능이 떨어지거나[5], 또는 파이프라인 구조를 사용하여 feedback 모드에서는 사용하지 못하는[3] 단점을 가지고 있다. 이에 본 논문에서는 암호화와 복호화를 모두 구현하고, 우수한 성능을 가지는 기존의 구현 사례들과 비슷한 성능을 가지면서 면적을 최소화하는 하드웨어 구조를 제안하였다.

Ⅲ. Rijndael 알고리즘

Rijndael 알고리즘은 비밀키 블록 알고리즘으로,



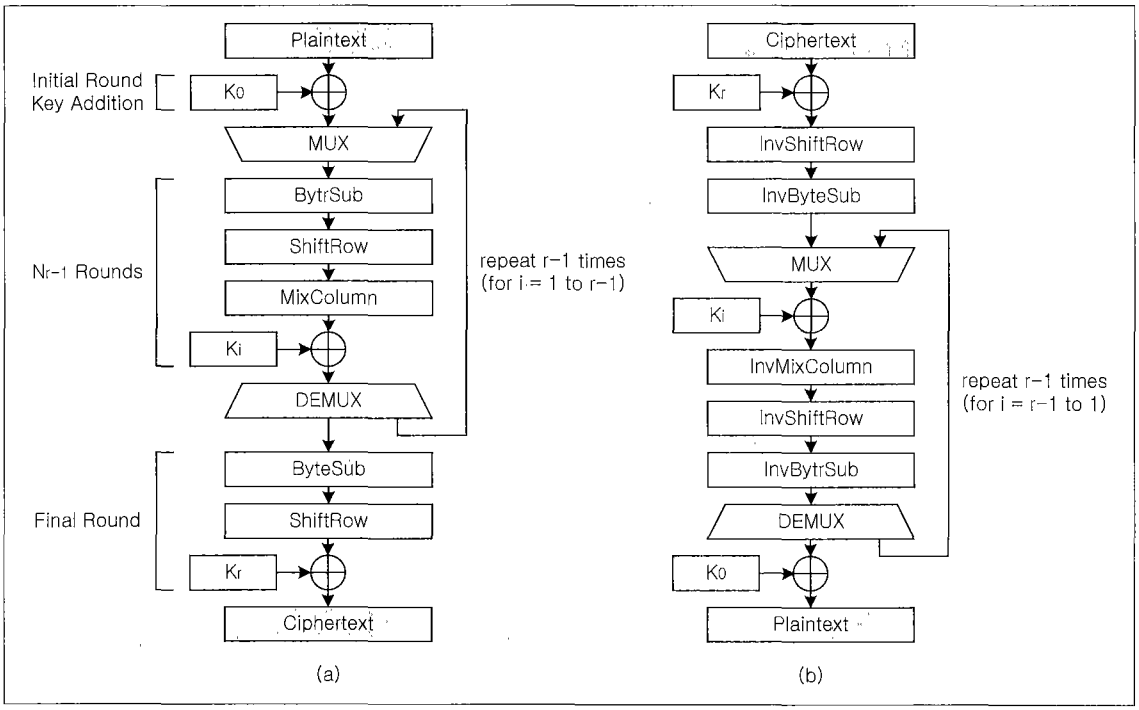
〔그림 1〕 입력 데이터의 매핑 예

AES의 조건으로 NIST에서 요구한 128비트의 블록 길이와 128, 192, 그리고 256비트의 키 길이를 지원하며, 추가로 192, 256비트의 블록 길이도 사용할 수 있다. 그리고 블록과 키 길이에 따라 라운드 수는 10, 12, 14로 가변적이다.[1]

Rijndael에서 처음 입력 데이터 값은 4행의 2차 배열로 구성한다. 배열순서는 〔그림 1〕과 같이 1행의 1열부터 2열, 3열 순서로 한 바이트씩 채워지고, 한 행이 다 채워지면 다음 행을 채운다. 그러므로 블록과 키는 4×Nb, 4×Nk의 블록으로 구성되며, 이때 Nb는 블록 길이를 32로 나눈 값이고, Nk는 키 길이를 32로 나눈 값이다.

3.1 암호화와 복호화

Rijndael은 〔그림 2(a)〕와 같이 Initial Round Key Addition, Nr-1 Rounds, 그리고 Final Round의 과정을 거쳐서 암호화를 수행한다. 하나의 라운드 함수는 ByteSub, ShiftRow, MixColumn, AddRoundKey의 4개의 변환으로 구성되어 있으며, Final Round에서는 MixColumn이 제외된다. ByteSub는 바이트 단위의 독립적인 비선형 치환이고, ShiftRow는 각 행마다 정해진 크기만큼 왼쪽으로 쉬프트 한다. MixColumn은 각 열에 대해 GF(2^8)상에서 정의된 행렬 곱셈을 연산하고, AddRoundKey는 라운드 키와 XOR 연산을 한다. 자세한 Rijndael의 내부 구조 및 이론적 근거는 〔1〕을 참조하도록 한다. 복호화의 경우에는 〔그림 2(b)〕와 같이 하나의 라운드에서뿐만 아니라 전체적으로 순서가 암호화의 반대로 된다. 라운드 키의 순서도 암호화의 반대가 되고, 라운드를 구성하는 4개의 변환들도 각 변환들의 inverse로 이루어진다. 그러므로 복호화를 수행하려면 각 변환들의 inverse를 추가로 구현해야 한다.



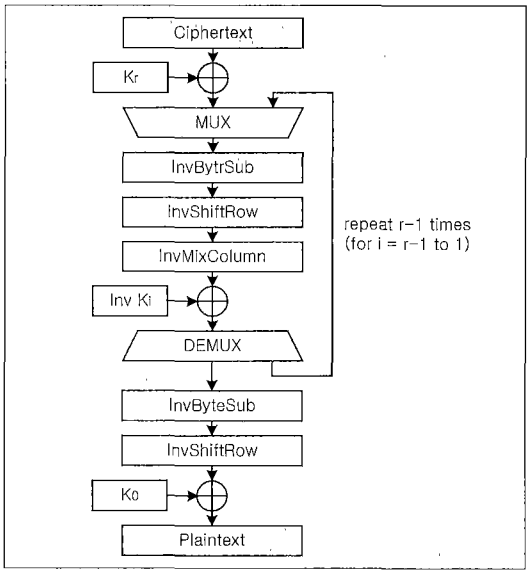
〔그림 2〕 Rijndael 알고리즘의 구조 (a) 암호화 (b) 복호화

3.2 변형된 복호화

Rijndael에서 복호화를 수행할 경우 〔그림 2(b)〕와 같이 한 라운드에서뿐만 아니라 전체적으로 암호화를 수행할 때와 순서가 반대가 된다. 그러나 대수적인 특성을 이용하면 다음과 같이 구조의 변형이 가능하다. 먼저 InvShiftRow는 단순히 바이트 단위로 자리만 이동하므로 바이트의 값에는 아무런 변화를 주지 않고, InvByteSub는 바이트 단위로 독립적으로 수행되므로 두 부분의 순서를 바꾸어도 상관이 없다. 둘째, AddRoundKey와 InvMixColumn의 순서는 InvMixColumn과 AddRoundKey의 순서로 변환할 수 있다. 이는 InvMixColumn이 linear transformation이므로 식 (1)의 특성에 따라 가능하다.

$$A(x+k)=A(x)+A(k) \qquad (1)$$

대신에 AddRoundKey와 InvMixColumn의 순서를 바꾸기 위해서는 InvMixColumn이 포함되지 않은 Initial Round Key Addition과 Final Round의 AddRoundKey에서 사용하는 첫 번째와 마지막 라운드 키를 제외한 모든 라운드 키를 InvMixColumn으로 변형시켜 사용해야 한다. 위의 두 가지 특성을 이용하여 순서가 변형된 복호화 과정은 〔그림 3〕과 같으며, 암호화 과정과 순서가 같아진다.

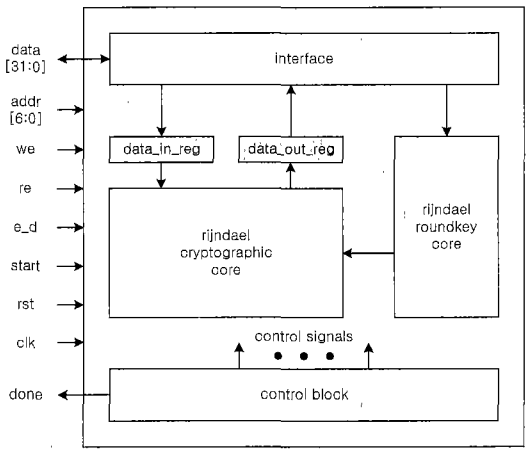


〔그림 3〕 변형된 복호화

Ⅳ. Rijndael 암호 프로세서 설계

본 논문에서는 다양한 응용분야에 적용할 수 있도록 Rijndael 알고리즘을 수행하는 암호 프로세서를 하드웨어로 구현하였다. 구현한 암호 프로세서는 암호화와 복호화 과정을 모두 지원하며, [그림 2(a)]의 암호화와 [그림 3]의 변형된 복호화 과정을 사용하였다. 블록과 키 길이는 128비트만 지원하도록 구현하였다. 키 생성부는 키 생성 과정이 매번 일어나는 것이 아니므로 포함시키지 않았으며, 대신에 외부의 키 생성부에서 읽어올 라운드 키를 저장할 메모리와 변형된 복호화 과정에 따른 InvMixColumn을 추가하였다.

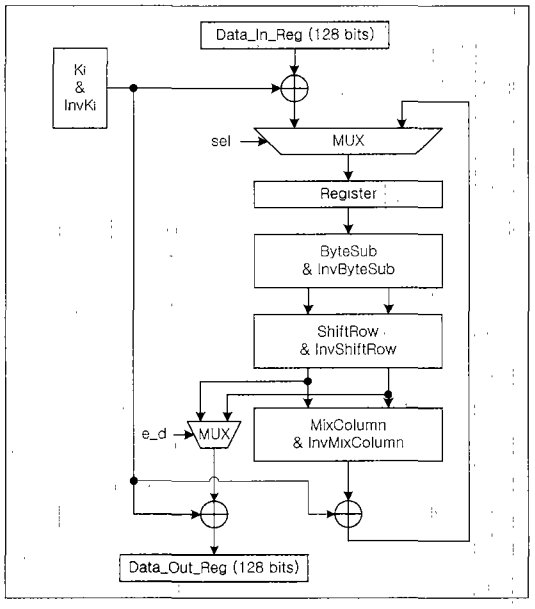
본 논문에서 구현한 Rijndael 암호 프로세서의 전체 구조는 [그림 4]와 같다. 크게 입출력 데이터의 전송을 관리하는 Interface, 입출력 데이터를 저장하는 Data In/Out Register, 데이터의 암복호화 연산을 수행하는 Rijndael Cryptographic Core, 외부의 키 생성부에서 생성한 라운드 키를 저장하는 RAM과 변형된 복호화 과정에 따른 InvMixColumn으로 이루어진 Rijndael Roundkey Core, 데이터를 암복호화 하는데 필요한 control 신호를 발생하는 Control Block으로 이루어져 있다. 임의의 데이터를 암복호화 할 경우, 먼저 데이터 버스를 통해 암복호화 할 데이터와 라운드 키를 입력하여 각각 Data_In_Register와 Rijndael Roundkey Core에 저장한다. 그리고 start 신호가 발생하면 Rijndael Cryptographic Core에서 암복호화 연산이 수행되며, 암복호화 연산이 수행하는 동안 done 신호는 '1' 상태가 된다. done 신호는 암복호화 연산이 수행중임을 알려 주는 신호로, 암복호화 연산이 끝나면 '0' 상태로 된다. 연산이 끝나면 암복호화 된 데이터는 Data_Out_Register에 저장된 후, 데이터 버스를 통해 출력된다. 그리고 입출력 핀 중 e_d 신호는 암복호화 과정을 구별하는 역할을 하고, 7비트의 addr 신호는 데이터 버스를 통해 Data_in/Out_Register와 Rijndael Roundkey Core의 RAM에 데이터를 전달할 때, 특정 위치의 데이터를 구별하는데 사용된다.



[그림 4] Rijndael 암호 프로세서 전체 구조

4.1 Rijndael 암호 코어

Rijndael Cryptographic Core는 [그림 5]와 같이 한 라운드만을 구현하고, 여기에 레지스터와 멀티플랙서를 하나씩 추가하여 라운드 수만큼 반복하여 암복호화를 수행하는 구조로 이루어져있다. 입력 데이터는 먼저 Initial Round Key Addition을 수행하고 멀티플랙서를 통해 레지스터에 저장된다. 그리고 한 클럭마다 ByteSub와 InvByteSub, ShiftRow와 InvShiftRow, MixColumn과 InvMixColumn, AddRoundKey로 구성된 한 라운드를 수행하고, 결과 값은 멀티플랙서를 통해 귀환하여 레지스터에 저장된다. 이를 라운드 수만큼 반복 수행하여 데이터를 암복호화 한다.



[그림 5] Rijndael 암호 코어 구조

4.1.1 ByteSub & InvByteSub 변환

ByteSub(S(x))는 바이트 단위로 독립적으로 수행되며, 식 (2)와 같이 두 개의 변환 과정으로 이루어져 있다.

$$S(x)=f(g(x)) \tag{2}$$

먼저 입력 값에 대해 g(x)를 수행한다. g(x)에서는 식 (3)과 같이 입력 값에 대해 $GF(2^8)$에서의 multiplicative inverse를 구한다.

$$g(x)=x^{-1} \text{ in } GF(2^8) \tag{3}$$

그리고 g(x)의 연산으로 구해진 결과 값을 입력으로 f(x)을 수행하여 데이터를 치환한다. f(x)에서 수행하는 연산은 식 (4)와 같다.

$$\begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \\ y4 \\ y5 \\ y6 \\ y7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{4}$$

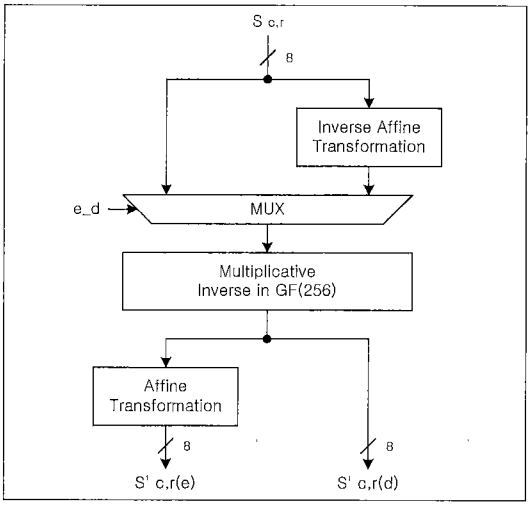
복호화 과정에서 사용하는 InvByteSub는 ByteSub의 inverse($S^{-1}(x)$)로, g(x)가 self-inverse의 성질을 가지고 있으므로 식 (5)와 같이 $f^{-1}(x)$와 g(x)의 연산으로 표현할 수 있다.

$$S^{-1}(x)=g^{-1}(f^{-1}(x))=g(f^{-1}(x)) \tag{5}$$

위의 식 (2)와 (5)에서 보이는 것과 같이 ByteSub와 InvByteSub에서 모두 g(x)를 사용하므로, 하나만 구현하여 공유할 수 있다. 그러므로 ByteSub와 InvByteSub는 단지 g(x), f(x)와 $f^{-1}(x)$로 구현할 수 있다.

이와 같은 연산으로 이루어진 ByteSub와 InvByteSub를 구현하는 방법으로는 먼저 LUT(look-up table)를 이용하는 방법이 있다. 그러나 이 방법은 본 논문과 같이 암복호화 과정을 모두 지원하는 경우, Feistel 구조인 대부분의 블록 암호 알고리즘과 달리 Rijndael은 복호화 과정을 위해 암호화 과정과 다른 LUT이 필요하다. 그러므로 LUT로 구현 시, 바이트 단위로 독립적으로 수행되는 ByteSub와 InvByteSub의 특성에 따라 16개의 4K-bit ROM이 필요하다. 그러므로 성능면에서는 우수하지만, 많은 양의 메모리를 필요로 하는 단점을 가지고 있다.

이에 본 논문에서는 식 (2)와 (5)의 함수에 기반을 두어 설계된 ByteSub와 InvByteSub의 특성을 이용하여 구현하였다. 이 경우 ByteSub와 InvByteSub의 연산은 식 (2)와 (5)에 보이는 것과 같이 둘 다 $GF(2^8)$에서의 Multiplicative Inverse(g(x)) 연산이 포함되므로, 이를 하나만 구현한 후 공유하여 사용할 수 있다. 그리고 Affine Transformation(f(x))과 Inverse Affine Transformation($f^{-1}(x)$)은 적은 양의 XOR 연산 조합으로 구현할 수 있다. 그러므로 Multiplicative Inverse 연산을 어떻게 구현하느냐가 중요한 변수가 된다. 먼저 Multiplicative Inverse 연산을 Extended Euclidean이나 Composite Field 등의 알고리즘을 이용하여 구현하면 적은 면적으로 구현할 수 있으나 우수한 성능을 기대할 수 없다. 그러므로 본 논문에서는 Multiplicative Inverse 연산은 LUT을 이용하여, [그림 6]과 같은 구조로 ByteSub와 InvByteSub를 구현하였다. 이로 인해 ByteSub와 InvByteSub의 연산 과정 전체를 LUT만으로 구현하는 방법에 비해 성능은 약간 감소하지만, 면적 측면에서 필요한 메모리 양이 절반으로 감소된다. 이렇게 구현된 본 논문의 ByteSub와 InvByteSub는 [그림 6]과 같이 암호화일 경우에는 입력 데이터가 Multiplicative Inverse 연산과 Affine Transformation 순서로, 복호화일 경우에는 Inverse Affine



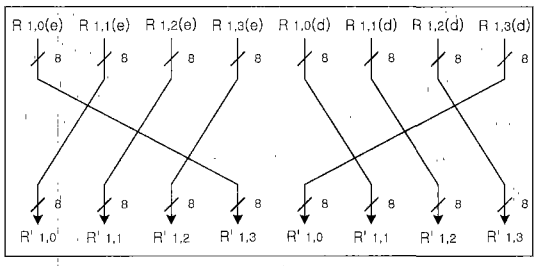
[그림 6] ByteSub & InvByteSub 변환 구조

Transformation과 Multiplicative Inverse 연산 순서로 수행되어 암복호화 경우의 데이터를 모두 출력한다.

4.1.2 ShiftRow & InvShiftRow 변환

ShiftRow와 InvShiftRow는 state를 행 단위로 Cyclic Shift한다. 암호화 과정의 경우, 첫 번째 행은 shift하지 않고, 두 번째, 세 번째, 네 번째 행은 오른쪽으로 1, 2, 3바이트만큼 shift한다. 그리고 복호화 과정의 경우는 암호화 과정과 동일한 양만큼 왼쪽으로 shift한다.

본 논문에서는 [그림 7]과 같이 ByteSub와 InvByteSub에서 ByteSub와 InvByteSub을 수행한 데이터가 둘 다 출력되도록 구현하여 ShiftRow는 ByteSub의 출력 데이터를, InvShiftRow는 InvByteSub의 출력 데이터를 입력으로 사용한다. 그리고 ShiftRow와 InvShiftRow을 수행한 두 개의 데이터로 출력된다. 그러므로 ShiftRow와 InvShiftRow는 하드웨어 리소스를 사용하지 않고 단순히 hard-wire로 구현이 가능하다. ShiftRow와 InvShiftRow의 두 번째 행을 [그림 7]에 예로 보였다.



[그림 7] ShiftRow & InvShiftRow 변환 구조

4.1.3 Mixcolumn & InvMixColumn 변환

MixColumn은 식 (6)처럼 한 열의 4바이트를 입력으로 $f(x)=x^8+x^4+x^3+x+1$을 primitive polynomial로 사용하는 $GF(2^8)$에서의 행렬 곱셈으로 이루어져 있다.

$$\begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \bullet \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \end{bmatrix} \quad (6)$$

InvMixColumn은 MixColumn의 inverse 연산으로 식 (7)과 같이 표현된다.

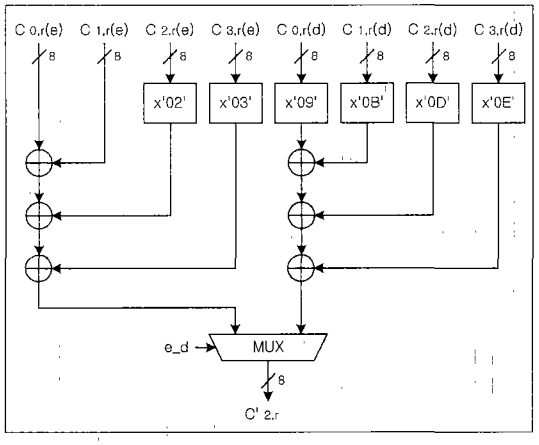
$$\begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \end{bmatrix} = \begin{bmatrix} 0E\ 0B\ 0D\ 09 \\ 09\ 0E\ 0B\ 0D \\ 0D\ 09\ 0E\ 0B \\ 0B\ 0D\ 09\ 0E \end{bmatrix} \bullet \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \end{bmatrix} \quad (7)$$

식 (6)과 (7)의 연산으로 이루어진 MixColumn과 InvMixColumn은 '02', '03', '09', '0B', '0D', '0E'로 구성된 상수와의 곱셈을 어떻게 구현하느냐가 중요한 이슈가 된다. 이를 위해 [1]에서는 Xtime 함수를 이용한 방법을 제안하였다. 본 논문에서는 이를 여러 개의 XOR 연산을 이용하여 구현하였다. 이를 이용한 상수와의 곱셈 연산 예를 식 (8)에 보였다.

$$Y=09 \bullet X, \quad \text{for } X, Y \in GF(2^8)$$

$$\begin{aligned} y_0 &= x_0 \oplus x_5 \\ y_1 &= x_1 \oplus x_5 \oplus x_6 \\ y_2 &= x_2 \oplus x_6 \oplus x_7 \\ y_3 &= x_0 \oplus x_3 \oplus x_5 \oplus x_7 \\ y_4 &= x_1 \oplus x_4 \oplus x_5 \oplus x_6 \\ y_5 &= x_2 \oplus x_5 \oplus x_6 \oplus x_7 \\ y_6 &= x_3 \oplus x_6 \oplus x_7 \\ y_7 &= x_4 \oplus x_7 \end{aligned} \quad (8)$$

이와 같이 여러 개의 XOR 연산을 이용하여 행렬 곱셈을 구현한 MixColumn과 InvMixColumn에서 한 바이트의 연산 블럭도는 [그림 8]과 같다. 암호화 동작일 경우에는 ShiftRow의 출력을 입력으로 식 (6)의 연산을 수행한 결과를, 복호화 동작일 경우에는 InvShiftRow의 출력을 입력으로 식 (7)의

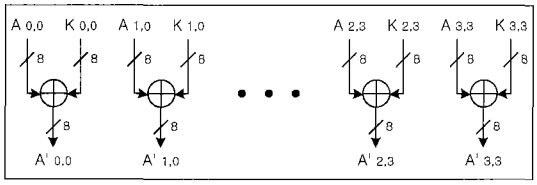


[그림 8] MixColumn & InvMixColumn 변환 구조

연산을 수행한 결과를 멀티플랙서를 통해 선택하여 출력으로 내보낸다.

4.1.4 AddRoundkey 변환

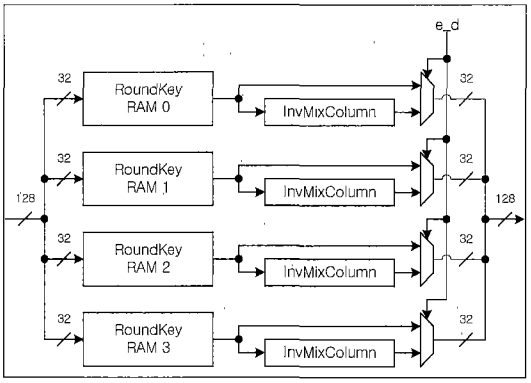
AddRoundKey는 [그림 9]와 같이 128비트 데이터와 128비트 라운드 키의 bitwise-XOR 연산으로 이루어져 있다. 복호화를 위한 AddRoundKey의 inverse는 AddRoundKey와 동일하므로 따로 구현할 필요가 없다.



[그림 9] AddRoundKey 구조

4.2 Rijndael 라운드 키 코어

본 논문에서 RoundKey Core는 [그림 10]과 같이 외부 키 생성부에서 생성한 라운드 키를 저장하는 RAM과 복호화 시 [그림 3]의 변형된 복호화 과정을 사용함으로써 추가로 필요한 InvMixColumn으로 구성되어 있다. 본 논문에서 구현한 암호 프로세서는 외부와 32비트 인터페이스를 사용하므로, 한 라운드마다 필요한 128비트의 라운드 키는 32비트의 입출력을 사용하는 4개의 RAM에 나누어 저장한다. 그리고 암호화 시에는 이를 그대로 사용하고, 복호화 시에는 첫 번째와 마지막 라운드 키를 제외한 모든 라운드 키를 InvMixColumn을 수행하여 사용한다.



[그림 10] Rijndael 라운드 키 코어 구조

V. 성능 분석

본 논문에서 구현한 Rijndael 암호 프로세서는 VerilogHDL을 이용하여 설계하였고, ALTERA FPGA를 사용하여 합성하고 성능을 검증하였다. 시뮬레이션을 통해 나온 결과 값은 NIST에서 주어진 테스트 벡터 값과 일치함을 확인하였다.

먼저 이론적으로 분석한 Rijndael 암호 프로세서의 성능을 [표 1]에 보였다. 하드웨어 리소스는 [표 1]에 보이는 것과 같이 대부분 메모리와 XOR 연산의 조합으로 이루어져 있다. Rijndael의 암복호화를 수행하는 Cryptographic Core의 경우 ShiftRow와 InvShiftRow에서는 하드웨어 리소스가 사용되지 않았고, 대부분의 하드웨어 리소스가 ByteSub와 InvByteSub, MixColumn과 InvMixColumn에서 소모되었다. 클럭 주기를 결정하는 최장지연경로는 [표 1]에 보인 것과 같이 ROM 1개, 2-input

[표 1] Rijndael 암호 프로세서의 하드웨어 복잡도

	Cryptographic Core				RoundKey Core	Extra
	ByteSub & InvByteSub	ShiftRow & InvShiftRow	MixColumn & InvMixColumn	AddRoundKey & InitAddRoundKey		
Hardware Resource	8*8 ROM 16개 XOR2 288개 XOR3 288개 MUX2 128개	-	XOR2 1696개 XOR3 400개 MUX2 128개	XOR2 384개	352-bit RAM 4개 XOR2 1184개 XOR3 352개 MUX2 128개	MUX2 256개 REG128 3개
	➡ 32 Kbits ROM, 1408-bit RAM, XOR2 3552개, XOR3 1040개, MUX2 640개, REG128 3개					
Critical Path	ROM 1개 XOR2 1개 XOR3 1개 MUX2 1개	-	XOR2 3개 XOR3 1개 MUX2 1개	XOR2 1개	-	MUX2 1개 Flip-Flop 1개
	➡ ROM 1개, XOR2 5개, XOR3 2개, MUX2 3개, Flip-Flop 1개					
# of clock	11 clock cycle					

XOR 5개, 3-input XOR 2개, 2-input MUX 3개, Flip-Flop 1개로, ShiftRow와 InvShiftRow, RoundKey Core에서는 전혀 생성되지 않고 대부분 ByteSub와 InvByteSub, MixColumn과 InvMixColumn에 의해 결정되었다. 그리고 한 라운드를 한 클럭에 수행하여 암복호화하는데 블럭 당 총 11 클럭이 걸린다.

본 논문에서 구현한 Rijndael 암호 프로세서는 ALTERA FLEX 10KE200 디바이스를 타겟으로 합성하였으며, 합성한 회로의 성능은 [표 2]와 같다. 구현한 Rijndael 암호 프로세서는 34816비트의 메모리와 3422개의 로직 셀을 사용하였으며, 이는 각각 EPF10K200SRC240-1 디바이스의 메모리와 로직 셀의 35, 34%에 해당한다. 그리고 최대 지연경로의 수행 시간은 약 33.9ns로, 29.49MHz의 최대 동작 주파수를 가진다. 그러므로 128비트의 블록을 암호화/복호화 하는데 총 수행 시간은 373ns가 걸리고, 343.26Mbps의 성능을 가진다.

본 논문의 Rijndael 암호 프로세서의 객관적인 성능 평가를 위해 최근 발표된 논문 중 128비트의 블록과 128비트의 키 길이를 지원하며, 암복호화를 모두 수행할 수 있도록 구현한 [4]와 비교 분석하였다. 그런데 [4]의 논문에서 제시한 성능과 면적은 FPGA로 구현했을 경우의 수치이므로, 최대한 공정한 비교를 위하여 삼성 0.5um CMOS 스탠다드 셀 라이브러리[6]를 근거로 예측하여 면적과 성능, 그리고 AT-product를 계산하여 [표 3]에 보였다. AT-product는 면적 대 성능 비를 비교하기 위한 수치로, A(면적)는 메모리를 게이트 수로 계산하여 random logic의 게이트 수와 더한 결과이며 T(수행시간)는 128비트의 블록을 암호화/복호화 하는데 걸리는 시간이다. 그리고 본 논문에서 제안한 방식으로 구현한 결과의 A와 T를 곱한 값을 1이라 가정하여 AT-product를 계산하였다. 본 논문에서 구현한 Rijndael 암호 프로세서는 최장지연경로의 수행시간은 약 9ns가 걸렸고, 한 라운드만을 구현하여 라운드 수만큼 반복하는 구조이므로 128비트의 블록을 암복호화 하는데 총 11클럭이 소요된다. 그러므로 110MHz의 동작 주파수에서 총 수행시간은 100ns가 걸려서 1.28Gbps의 성능을 보인다. 그리고 32K-bit ROM, 1408-bit RAM과 약 15,000게이트의 하드웨어 리소스가 필요하다. [4]는 본 논문과 같이 한 라운드만을 구현하여 반복하는 구조로, 두 가지 방법을 사용하여 구현하였다. S-box를 이용한 방법과 T-box를 이용한 방법으로, 이는 본 논문과 같이 128비트의 블록을 암복호화 하는데 총 11클럭이 소요되어 본 논문과 비슷한 성능을 가지지만 면적의 상당 부분을 차지하는 메모리의 크기가 본 논문에 비해 각각 2, 4배 크다. 그 결과 본 논문은 AT-product가 1인 반면에 [4]의 S-box와 T-box를 이용하여 구현한 결과는 각각 1.26과 1.89로, 본 논문에서 제안한 방식으로 구현한 결과의 AT-product가 가장 작음을 알 수 있다. 그러므로 암복호화를 모두 수행하도록 Rijndael을 구현할 경우, 본 논문에서 제시한 방법이 성능과 면적

[표 2] ALTERA FLEX 10KE200-1에서 Rijndael 구현 결과

Device	EPF10K200SRC240-1
Memory bits	34816/98304(35%)
Logic Cells	3422/9984(34%)
Frequency(MHz)	29.49
Speed(Mbits/s)	343.26

[표 3] 성능 비교

Feature		Fischer et al[4] Iterative S-box	Fischer et al[4] Iterative T-box	Ours Iterative
Block/Key Length		128/128		
# Cycles/Block		11		
Performance		1.45 Gbits/s (125MHz)	1.75 Gbits/s (150MHz)	1.28 Gbits/s (110MHz)
Area	Random Logic	13788.8 gates	9520.8 gates	15449.6 gates
	Memory	64K-bit ROM 1408-bit RAM	256K-bit ROM 1408-bit RAM	32K-bit ROM 1408-bit RAM
	Total	63629.5 gates	114115.7 gates	44268.2 gates
AT-product		1.26	1.89	1

측면에서 가장 효율적인 구현 방법임을 알 수 있다.

Ⅵ. 결 론

본 논문에서는 성능과 면적 측면에서 가장 효율적인 구조로 Rijndael 암호 프로세서를 FPGA에 구현하였다. 이는 암복호화를 모두 수행하며, 블록과 키 길이는 128비트만 지원하도록 하였다. 한 라운드에서뿐만 아니라 전체적으로 암호화와 순서가 반대로 바뀌는 복호화의 경우 효율적인 구현을 위해 암호화와 순서가 동일한 그림 3의 변형된 복호화 구조를 사용하여 구현하였다. 그리고 키 생성부는 키 생성 과정이 매번 일어나는 것이 아니므로 구현하지 않고, 외부의 키 생성부에서 읽어올 라운드 키를 저장할 메모리와 변형된 복호화 과정에 따른 InvMixColumn으로 구현하였다. 암복호화 시 구조가 다른 Rijndael의 특성으로 인한 면적의 증가를 최소화하기 위해 ByteSub와 InvByteSub는 알고리즘을 기반으로 구현함으로써 메모리로만 구현하는 방법에 비해 비슷한 성능을 가지면서 필요한 메모리 양은 1/2로 줄였다. 이로 인해 본 논문에서 구현한 Rijndael 암호 프로세서는 지금까지 발표된 논문 중 가장 우수한 면적 대 성능 비를 가진다.

본 논문의 Rijndael 암호 프로세서는 삼성 0.5um CMOS 스탠다드 셀 라이브러리를 근거로 예측 시, 약 15,000개의 게이트, 32K-bit ROM과 1408-bit RAM로 구성되며, 약 110MHz의 동작 주파수에서 1.28Gbps의 성능을 가진다. 이러한 성능은 0.25나 0.18um의 기술을 적용하면 성능을 더욱 향상시킬 수 있을 것이다. 그러므로 본 논문의 Rijndael 암호 프로세서는 전자 상거래나 네트워크 등 여러 분야의 보안 모듈로써 적용할 수 있을 것으로 여겨진다.

앞으로 128비트 이외에 192, 256비트의 블록과 키 길이를 모두 지원하도록 Rijndael 암호 프로세서를 구현 시, 동일한 구조를 가져 확장이 용이한 다른 부분과 달리 블록과 키 길이에 따라 shift 양이 변하는 ShiftRow와 InvShiftRow을 어떻게 구현하느냐가 중요한 이슈가 될 것이다. 그러므로 이로 인해 생기는 성능 저하와 면적의 증가를 최소화하는 연구가 필요하다.

참 고 문 헌

[1] J. Daemen and V. Rijimen, "AES Proposal : Rijndael, AES Algorithm Submission", September 1999, available at http://csrc.nist.gov/encryption/aes/rijn dael/Rijndael.pdf

[2] Henry Kuo and Ingrid Verbauwhede, "Architectural Optimization for a 1.82Gbits/sec VLSI Implementation of the AES Rijndael Algorithm", *Worksho p on Cryptographic Hardware and Em bedded Systems 2001(CHES 2001)*, May 2001.

[3] Maire McLoone and J. V. McCanny, High Performance Single-Chip FPGA Rijndael Algorithm Implementations, *Workshop on Cryptographic Hardware and Embedded Systems 2001(CHES 2001)*, May 2001.

[4] Viktor Fischer and Milos Drutarovsky, Two Methods of Rijndael Implementation in Reconfigurable Hardware, *Workshop on Cryptographic Hardware and Embedded Systems 2001(CHES 2001)*, May 2001.

[5] Atri Rudra, Pradeep K. Dubey, Charanjit S. Jutla, Vijay Kumar, Josyula R. Rao, and Pankaj Rohatgi, Efficient Implementation of Rijndael Encryption with Composite Field Arithmetic, *Workshop on Cryptographic Hardware and Embedded Systems 2001 (CHES 2001)*, May 2001.

[6] Samsung Electronics, Samsung STD85/STDM85 0.5um High Density CMOS Standard Cell Library, September 1997.

〈著者紹介〉

전 신 우 (Shin-woo Jeon) 학생회원
2000년 2월 : 광운대학교 전자공학부 졸업
2000년 3월~현재 : 광운대학교 전자공학과 석사과정
〈관심분야〉 통신용 칩 설계, 무선 통신, 정보보호

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
〈관심분야〉 컴퓨터 연산 알고리즘, ASIC 설계, 무선 통신, 정보보호

권 오 준 (Oh-jun Kweon) 정회원
1986년 2월 : 경북대학교 전자공학과 졸업
1992년 2월 : 충남대학교 전산학과 석사
1998년 2월 : 포항공과대학교 전자계산학과 박사
1986년 1월~2000년 2월 : 한국전자통신연구원 선임연구원
2000년 3월~현재 : 동의대학교 전산통계학과 전임강사
〈관심분야〉 신경망 응용, 패턴 인식, 지능정보처리, 정보통신 서비스 및 정보보호