FPGA를 이용한 AES 복호화 코어 구현에 관한 연구

김남우* · 허창우*

*목원대학교

The study of AES Decryption Core for FPGA implements.

Nam-woo Kim* · Chang-Woo Hur**

*MOKWON University

E-mail : gotree94@gmail.com

요 약

FPGA상에 AES 복호화 코어를 FIPS-197사양에 기술된 AES 알고리즘의 복호화부분을 구현하였다. 키값의 길이는 128/192/256비트를 지원하며, 별도의 코어 로직은 FPGA의 6-input lookup table의 이점을 살리도록 설계되었으며, 이결과로 2000개의 lookup table만을 이용하여 256비트 키에서 3Gbps의 처리가 가능하게 되었다. 코어는 난수 및 FIPS-197, SP-800a와 AESAVS사양의 테스트 벡터를 통해서 검증하였다.

키워드

AES, Rijndael, Decription.

I. 서론

AES는 1997년 1월 2일 미국국립표준원(NIST)에서 DES를 대체할 목적으로 암호알고리즘을 제안을 받아 DES를 대체할 암호를 공모하여 선정된 암호로 정식 명칭은 AES(Advanced Encryption Standard) 이다. 128비트 블록을 128혹은 192혹은 256비트 키 길이로 처리할 수 있는 암호 알고리즘이어야 하며, 무료로 배포 할 수 있는 알고리즘이어야 했다. (NIST의 이러한 요구안은 암호론의 Kerckhoffs′ principle 에 기반한 것이다.) 1998년 1월 15일까지 제안할 수 있었으며 21개의 암호 알고리즘이 제안되었고 그 중 15개 암호 알고리즘이 AES 후보로 선정되어 안전성을 평가 받았다. NIST는 15개의 암호 알고리즘에 대하여 두 번의 선별과정을 거친 후, 1999년 8월에는 5개의 후보 알고리즘을 최종 후보로 선정하여 많은 암호학자들로 부터 안전성 평가를 받게 하였다. 이 때, 남은 5개 알고리즘의 이름은 MARS, RC6, Rijndael, Serpent, Twofish 였다. 그리고 2000년 4월 "Third AES Candidate Conference" 이 개최되고 2000년 10월 2일 AES 알고리즘으로 Rijndael을 선정 하였다. 이어 NIST는 2001년 2월 28일에 연방 정보 처리 표준으로 AES를 공개/리뷰/배포 하면서 기밀성있는 정보에 DES를 대체하여 AES를 사용하기 시작한다.

AES 후보 알고리즘들은 다음과 같은 세가지 조건을 만족해야 했다.

- 안전성(security)
- 비용(cost)
- 알고리즘 및 구현 특성 (algorithm and implementation characteristics)

"안전성"은 절대적으로 갖춰야 하는 부분이며 특히 대칭키 암호를 분석하는 방법인 '선형 공격(linear cryptanalysis)'와 '차분 공격(Differential cryptanalysis)'에 대한 안전성 증명되어야 하며,

"비용"은 '스마트 카드, 하드웨어, 소프트웨어, 구현'을 위한 다양한 형태의 계산효율성(computational efficiency)을 참고하여 평가 되었다. 여기서 계산효율성이라고 하는 것은 속도 및 메모리 요구량(speed and memory requirement) 등을 의미 하였다.

"알고리즘 및 구현 특성"은 유연성(Flexibility)와 알고리즘의 단순성(simplicity)을 주로 평가 하였다.

Rijndael의 알고리즘이 안전성, 속도, 효율성, 구현 및 유연성이 다른 알고리즘들 보다 우수했다.

여기에서는 구현된 암호 해독 AES 코어는 FPGA에서 FIPS-197 규격에 설명 된 AES (일명 Rijndael) 알고리즘의 복호화 부분을 구현한다. 128/192/256 비트의 키 길이를 지원하며, 별도의 인스턴스 래퍼로 분리되어있다. 코어 로직은 6-input 룩업 테이블 (LUT6) 기반의 FPGA 아키텍처를 활용하도록 설계 되어 그 결과, 2000개의 LUT만 차지 하면서도 256비트 키의 3Gbps 이상의 피크 스루풋 성능이 가능하다. 코어는 렌덤 테스트 벡터, FIPS-197, SP-800A 및 AESAVS 사양에 제안된 테스트 벡터를 통해서 검증 하였다.

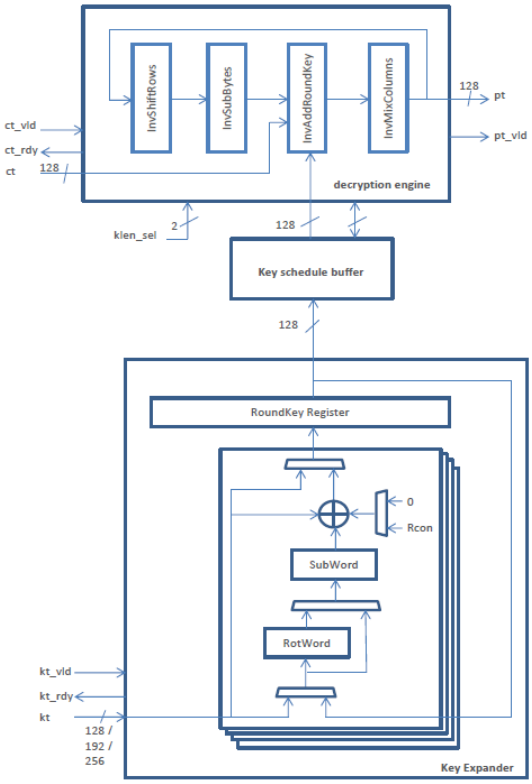

그림 1. AES Core Architecture

해독 AES 코어의 블록도는 [그림1]과 같다. 코어는 각 인터페이스로부터 암호문과 키 텍스트를 받아들이고, FIPS-197 규격에 기재된 AES 암호 해독 알고리즘을 수행하여, pt 인터페이스에서 평문을 출력한다. 암호 해독 엔진 과 키 스케줄 버퍼는 같은 길이의 키를 사용하며, 각각의 키값의 길이를 지원하기 위해서 key expander를 별도로 가지고 있다. 키 길이마다 별도의 래퍼를 두어 적절한 key expander 및 다른 모듈의 인스턴스를 용이하게 하였다. 런타임 중에 키 길이의 동적 전환은 지원되지 않는다.

암호 해독 엔진은 FIPS-197 규격의 [그림 12]에 있는 역 암호화 알고리즘을 구현하였다.

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word
w[Nb*(Nr+1)]) begin byte state[4,Nb]

        state = in

        AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) //
        See Sec. 5.1.4

        for round = Nr-1 step -1 downto 1
             InvShiftRows(state) // See Sec. 5.3.1
             InvSubBytes(state) // See Sec. 5.3.2
             AddRoundKey(state, w[round*Nb,
             (round+1)*Nb-1]) InvMixColumns(state) //
             See Sec. 5.3.3

        end for

        InvShiftRows(state) InvSubBytes(state)
        AddRoundKey(state, w[0, Nb-1])

        out = state
   end
```

그림 2. 역 암호화 알고리즘

키 확장기는 FIPS-197 규격의 [그림 11]의 키 확장 알고리즘을 구현하였다.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
word temp
i = 0
while (i < Nk)
w[i] = word(key[4*i], key[4*i+1], key[4*i+2],
key[4*i+3])
i = i+1
end while
i = Nk
while (i < Nb * (Nr+1)]
temp = w[i-1]
if (i mod Nk = 0)
temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
else if (Nk > 6 and i mod Nk = 4)
temp = SubWord(temp)
end if
w[i] = w[i-Nk] xor temp
i = i + 1
end while
end
Note that Nk=4, 6, and 8 do not all have to be
implemented;
they are all included in the conditional statement
above for
conciseness. Specific implementation requirements
for the
Cipher Key are presented in Sec. 6.1.
```

그림 3. 키 확장 코드

키 스케줄 버퍼는 키 확장 및 해독 엔진 사이에 있다. 128비트의 16개의 어드레스를 가지는 듀얼 포트 메모리로 키 확장기와 복호화 엔진 하시에 데이터를 주고 받을 수 있도록 독립적으로 읽기/쓰기가 가능하다. 역 암호 알고리즘들이 키 확장 알고리즘에 의해 생성되는 것보다 역순 라운드 키를 소비하기 때문에 키 스케줄 버퍼가 필요하다.

표 1. 입/출력

Ports	길이	방향	설명
CLK	1	입력	코어 클럭 . 상승 에지.
rst	1	입력	코어 리셋 . 액티브 하이.
KT[0:N]	128/ 192/ 256	입력	키 입력
kt_vld	1	입력	키 유효. 액티브 하이.
kt_rdy	1	출력	인터페이스를 준비.
CT[0:127]	128	입력	암호문 입력.
ct_rdy	1	출력	인터페이스를 준비. 새로운 암호문을 받아들임.
PT[0:127]	128	출력	평문 출력.
pt_vld	1	출력	평문 유효.

Ⅱ. 본론

2.1 AES Core 동작의 이해

기본 암호 해독사이클은 3 단계로 구성되는데

1. 암호화 키를 로드
2. 암호문을 로드
3. 평문 읽기

일반 평문이 pt 인터페이스에서 사용할 수 있게 되면, 다음 복호화 사이클은 다른 키나 암호문을 입력할 수 있게 된다. 암호문을 사용하는 경우 이전과 동일한 암호키를 사용하며, 키 스케줄러 버퍼에 이미 이전 키 스케줄이 저장되어있기 때문에 다시 키를 로드하지 않아도 된다.

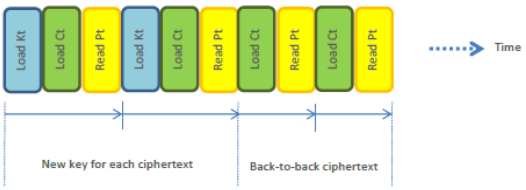

그림 4. 기본 암호 해독사이클

평문이 읽혀지기 전에 새 암호문이 입력된 경우, 코어는 즉시 새로운 암호문을 해독하고 이전 평문을 덮어 쓰도록 되어 있다.

새로운 키를 로딩은 key expension 사이클 일 때만 시작하며 이전의 평문은 변경하지 않는다.

2.2 Simulation

코어는 FIPS-197, AESAVS 및 SP-800A에서 선택된 테스트 벡터를 통해서 검증하였다. 또한 랜덤 테스트 벡터와 AES 동적 모델에 대해 테스트 하였다.

수행되는 시험은 아래에 나열되어 있다.

1. FIPS-197 sample vector test
2. Back-to-back ciphertext test
3. ECB-AES128/192/256.Decrypt sample vector test. SP800-38a appendix F
4. GFSbox Known Answer Test. AESAVS appendix B
5. KeySbox Known Answer Test. AESAVS appendix C
6. VarTxt Known Answer Test. AESAVS appendix D
7. VarKey Known Answer Test. AESAVS appendix E
8. Random vector test

2.3 Verification

알려진 좋은 값 또는 golden 모델의 출력 중 하나에 코어의 출력을 비교한다. 불일치의 경우에는, 에러 메시지를 출력하고 시뮬레이션을 계속한다. 일단 모든 시험은 "OK"또는 "Fail"을 표시하고 완료된다.

기본적인 128 비트 복호주기의 타이밍도는 [그림 5]와 같다.

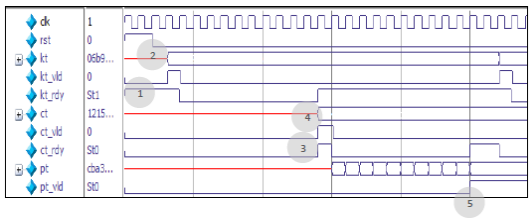

그림 5. 기본적인 128 비트 복호주기의 타이밍도

192 및 256 비트 키의 암호 해독 사이클은 대기시간이 다른것을 제외하고 비슷합니다.

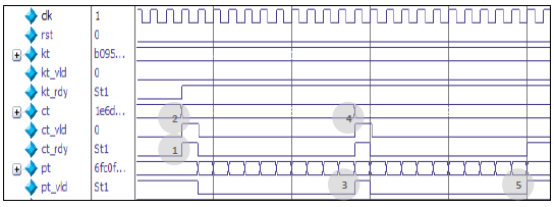

그림 6. 백투백 암호문에 대한 복호화 사이클

[그림6]은 백투백 암호문에 대한 복호화 사이클을 나타낸다.

1. 새로운 암호문을 받아 들일 준비가되었을 때의 코어는 ct_rdy를 '1'로 설정한다.
2. CT에 암호문을 구동하고 유효한 암호문이 존재한다고 ct_vld를 '1'로 설정한다.
3. 유효한 일반 평문을 사용할 수 있을 때 코어는 pt_vld를 '1'로 설정한다. 동시에 새로운 암호문을 받아들일 준비가 되었음을 나타 내기 위해 다시 ct_rdy을 '1'로 설정한다.
4. CT로 다음 암호문을 구동하고 새로운 암호문이 존재하면 코어는 ct_vld를 '1'로 설정한다.
5. 두 번째 일반 텍스트가 PT에 사용할 수 있을 때 pt_vld를 다시 '1'로 설정한다.

Ⅲ. 결론

코어는 현대 LUT6 기반의 FPGA에 구현시 성능 및 리소스 활용을 극대화 할 목적으로 설계하였다. 가능하면 LUT6s에 잘 맞게 할 수 있도록 대부분의 6 입력 신호에 사용가능한 로직으로 제한된 소스 코드를 통해서 구현 하였다. 그 외에는 다른 회사의 아키텍터와 무관하도록 소스코드를 구현하다.

자일링스 Vivado 합성 속성은 특정 논리 자원의 추론을 암시하고 FPGA의 조각으로 잘 포장의 설계 계층 구조를 유지하기 위해 소스 코드 전체에서 사용됩니다. 이러한 속성은 대상 FPGA에 대한 최대 성능을 달성하기 위해 필수적입니다.

[표2]는 다른 FPGA 기술로 타겟 변경 또는 다른 합성 툴을 사용할 때 교체해야 Vivado 합성 특성을 나타낸다.

벤치마킹을 위해, I/O 핀 수를 줄이기 위해 시프트 레지스터와 같은 구조로 코어를 레핑 하였고, 자일링스 Vivado 2013.4의 "Performance _Explore"로 합성 및 구현 하였다. 대상 디바이스는 자일링스의 KINTEX군의 xc7k325tffg900-3를 사용하였다.

지연시간은 키 텍스트와 암호문이 도착한 때로부터 평문의 출력이 출력될 때 까지의 클럭 개수로 측전하는데 키 확장 지연시간 및 복호화 엔진의 지연시간의 합으로 구성된다. 후속 암호문 블록은 이전과 동일한 키를 사용하게 된다. 따라서 이전에 계산 된 키 스케줄을 다시 사용되기 때문에 키 확장 대기 시간은 단지 복호화 엔진 지연시간과 같아진다.

표 2. 벤치마크 결과

	128-bit	192-bit	256-bit
LUT	1909	2126	2029
FF	299	426	439
BRAM	0	0	0
Latency w/ key switching w/o key switching	22clk 11clk	26clk 13clk	30clk 15clk
Fmax (MHz)	364	360	357
Peak throughput (Gbps)	4.235	3.544	3.046