論文 95-9-9-10

# VHDL를 이용한 확장된 DES(HDES)설계

正會員 吳 幸 洙*, 韓 承 朝**

# A Design of the Extended DES Using VHDL

Haeng Soo Oh*, Seung Jo Han** Regular Members


## 要 約

컴퓨터통신망을 이용한 정보전송이 급증하고 있는 오늘날, 정보를 저장하거나 전송시에 정보가 불법적으로 노출되거나 삽입 또는 수정되는 것을 방지하기 위해 정보에 대한 암호화가 요구된다. 따라서 [1]에서 발표한 확장된 DES를 S/W로 구현하여 활용하였으나 실시간 처리에 사용할 수 없으므로, 본 논문에서는 이를 H/W로 구현하기 위해서 국제 표준언어로 채택된 시스템 설계 기술언어인 VHDL을 이용하여 탑다운 설계방식으로 설계한다. 확장된 DES는 암호처리부와 키생성부 그리고 제어부로 나누어 설계하는데, 암호처리부는 F 함수가 96비트 블럭 데이타당 16라운드를 수행하도록 하였으며, 키생성부는 128비트를 키로 입력받아 수행하도록 하였다. 그리고, 제어부는 외부 입출력에 관계한 레지스터 조작과 데이타 흐름에 대한 제어코드를 생성하도록 한다. 시뮬레이션을 통하여 암호화시의 입력된 평문과 복호화시의 출력된 평문이 동일함을 보인다.

## ABSTRACT

The information communication through the computer network has been recently increased rapidly. When sending or storing it, to prevent the information from being exposed illegaly to others and modified by others, or added to other information by an invader, the cryptosystem is needed. As in the [1] the Extended DES is used by implementing S/W, but it turns out to be not appropriate in real time.

In this paper, by implementing H/W, the top-down design methodology is tried using VHDL, the system designing technical language which is adopted as the international standard language. The Extended DES is divided into 3 parts, the encryption processing part, in which F function implements 16 rounds per 96 bits block data; the key producing part, which implements 128 bits inputted by key; the control part, which produces the control code in the resister operation and the data flow connected with the external input and output. The result of simulation shows that the inputted sentences in cryptosystem are equal to the outputted sentences in decryption.


*동신전문대학 전자과
Dept. of Electronic Eng. Dongshin College
**조선대학교 공과대학 전자공학과
Dept. of Electronic Eng. Chosun University
論文番號 : 95006-0105
接受日字 : 1995年 1月 5日

## 1. 서 론

컴퓨터 통신의 발달로 인하여 독립적이며 상대적으로 무가치한 데이타들이 개인간에 또는 국가간에 상호교류가 되어 이제는 임의의 가공 데이타가 한 기업이나 국가의 존폐를 결정할 수도 있으며, 유형적인 물질의 가치보다는 무형적인 정보의 가치를 중요시하는 정보화 시대를 맞이한 것이다. 그러나 정보전송시 정보가 고의적으로 제 3자(비 인가자)에 의해 불법적으로 노출되거나 삽입 및 수정되는것을 보호하기 위해서는 정보전송에 대한 암호화(기밀성)가 요구된다[2].

현재 가장 보편적으로 실용화되어 사용되는 암호 알고리즘은 IBM 의 Lucifer 알고리즘을 기반으로 개발한 DES(Data Encryption Standard)이다[3][4]. 그러나 DES는 학자들에 의해 여러가지 문제점들이 제기되어 왔으며, 최근에 와서 IC의 패킹밀도와 수행속도가 증가함으로 인하여 DES의 시간 복잡도와 공간 복잡도가 감소되고 있다. 이로 인하여 56비트 키 길이를 갖는 DES는 exhaustive 공격[5]과 테이블 look up 공격[6] 그리고 time-memory trade off 공격[7]에 대해서 암호강도가 감소된다. 또한 DES는 237의 복잡도로 공격이 가능한 differential cryptanalysis 공격[8][9]에 대응하기 위해서는 수행시간이 증가하지 않은 범위에서 암호알고리즘 체계를 변경하여야 하고, SAC(strict avalanche criterion)과 상관계수의 조건[10]을 만족하며 XOR (exclusive-OR)분포가 균일한 S-box를 재설계하여야 한다[2].

따라서 일본과 호주에서는 DES와 유사한 FEAL[11]과 LOKI[12][13]를 개발하여 자국에서 실용화하고 있고, 각 나라에서는 나름대로 국가표준 암호 알고리즘을 개발하려고 노력하고 있다. 그러므로 우리 나라에서도 무조건 통신량만 증가시킬 것이 아니라 이제는 정보보호 차원에서 새로운 암호 알고리즘을 개발해야 한다. 반도체 기술이 발달하여 IC의 패킹밀도가 증가하고 수행속도가 급격히 증가한 오늘날에는 상기에서 논의한 여러가지 공격방법에 의해 DES는 공격받을 가능성이 높아지므로, 암호분석의 평가지표에 따라 S/W로 구현한 확장된 DES를 [1]에서 발표한적이 있다(발표[1]에서 확장된 DES를 HDES(Hankook Data Encryption Standard)로 명명하였으므로 본논문에서도 확장된 DES를 HDES로 명칭함). 그러나 S/W로 구현한 HDES는 수행시간이 길어 실시간 처리가 어려워 실용화하기에 불편하므로 본 논문에서는 국제표준언어로 채택된 시스템 설계 기술언어인 VHDL(Very high speed integrated circuits Hardware Description Language)을 이용하여 탑다운 설계방식으로 암호시스템인 HDES를 설계하고자 한다.[14]

## 2. 확장된 DES(HDES) 알고리즘

DES는 암호화, 복호화 알고리즘이 대칭적이며 치환(Permutation)과 대치(Substitution) 그리고 S-box로 구성된 블럭암호화 시스템이다. DES는 64비트씩 암호화하여 64비트 평문을 다시 32비트씩, 좌,우 두 부분으로 나누어 전 round의 오른쪽부분이 새로운 round의 왼쪽부분이 되고 전 round의 오른쪽에 암호함수(F function)를 적용하여 전 round의 왼쪽부분과 XOR 연산을 해서 오른쪽 round 부분이 되는 동작을 16회 반복하며 반복횟수가 증가할때 마다 암호강도는 높아진다[1].

그러나 하드웨어가 발달된 최근에 와서는 DES의 비도가 떨어지므로 DES의 비도를 증가시키기 위해서 아래와 같은 조건들을 만족시키도록 HDES 알고리즘을 설계하였다.

1)현재의 56비트 키길이를 112비트 길이로 확장시켰다.

2)S-box 내의 엔트리를 일정하게 수정하여 SAC과 상관계수 조건에 맞게 S-box를 선정하였으며, S-box를 $S_1$~$S_{16}$으로 확장시켰다.

3)differential cryptanalysis에 대한 일부의 대응방안으로 N라운드 특성이 구성될 확률을 낮추기 위해 S-box수의 증가와 각 서브블럭(A,B,C)마다 16라운드 동안의 암호함수의 반복횟수를 각각 달리하도록 알고리즘을 확장하였다.

그러므로 1), 2), 3)조건을 만족시키기 위한 암호 알고리즘 설계는 DES알고리즘과 같이 입력 데이터의 한 블럭을 64 비트씩 읽어드려 이를 2개의 32비트 서브 블럭 (L, R)으로 나누어 암호 함수를 적용하는 것이 아니라 입력데이터의 한 블럭단위를 96비트로 읽어드려 이를 3개의 서브 블럭(A, B, C)이 16 라운드를 반복 수행하도록 하였다.

먼저 HDES의 대칭적 특징인 암호화 과정과 복호화

과정의 식은 다음과 같다.

암호화 :

$$\left.\begin{aligned} A_i &= B_{i-1} \\ B_i &= C_{i-1} \oplus f(B_{i-1}, K_{2,i}) \\ C_i &= A_{i-1} \oplus f(B_{i-1}, K_{1,i}) \end{aligned}\right\} \quad (1)$$

복호화 :

$$\left.\begin{aligned} A_{i-1} &= C_i \oplus f(A_i, K_{2,i}) \\ B_{i-1} &= A_i \\ C_{i-1} &= B_i \oplus f(A_i, K_{1,i}) \end{aligned}\right\} \quad (2)$$

S-box는 암호함수내에 존재하고 암호함수는 $K_{1,i}$와 $K_{2,i}$에 따라 2개로 양분된다.따라서 S-box를 $S_1 \sim S_8$에서 $S_1 \sim S_{16}$으로 증가시키기 위해서는 S-box도 역시 $S_1 \sim S_8$과 $S_9 \sim S_{16}$으로 양분하여 $K_{1,i}$과 $K_{2,i}$와 함께 2개의 암호함수에 다음 식과 같이 적용되도록 한다.

암호화 : $$\left.\begin{aligned} f(B_{i-1}, K_{2,i}) &= P(S_1(D_1), \cdots\cdots\cdots S_8(D_8)) \\ f(B_{i-1}, K_{1,i}) &= P(S_9(D_9), \cdots\cdots\cdots S_{16}(D_{16})) \end{aligned}\right\} (3)$$

복호화 : $$\left.\begin{aligned} f(A_i, K_{2,i}) &= P(S_1(D_1), \cdots\cdots\cdots S_8(D_8)) \\ f(A_i, K_{1,i}) &= P(S_9(D_9), \cdots\cdots\cdots S_{16}(D_{16})) \end{aligned}\right\} (4)$$

여기서

$$\left.\begin{aligned} b_1, b_2, b_3, \ldots\ldots\ldots, b_8 &= E(B_{i-1}) \oplus K_{2,i} \\ b_9, b_{10}, b_{11}, \ldots\ldots\ldots, b_{16} &= E(B_{i-1}) \oplus K_{1,i} \end{aligned}\right\} (5)$$

이며,

$f(B_{i-1}, K_{i,1})$ : 암호함수

$P(S_1, \ldots\ldots)$ : P-box

$S(D)$ : S-box

$D_i$ : 6비트 블럭으로 S-box의 입력이 됨

$E(B_{i-1})$ : extended permutation

이다.

따라서 상기에서 분석한 모든 식들을 적용하여 HDES의 암호알고리즘과 복호알고리즘을 나타내면 그림1과 같다.

그림1에서 보인 바와 같이 암호화 과정에서 마지막 라운드의 $A_{16}$과 $B_{16}$을 서로 교환하여 주어야 하며, 복호화 과정은 암호화 과정과 동일하나 $A_i$와 $B_i$를 교환하여 서브블럭에 입력하여야 한다.

키는 역순인 $K_{1,16}$, $K_{1,15}$, $K_{1,14}$, $\cdots\cdots$, $K_{1,1}$순으로 입력하되 좌우 $K_{1,i}$, $K_{2,i}$과 좌우 $S_1 \sim S_8$, $S_9 \sim S_{16}$를 서로 교환하여 주어야 한다. differential cryptanalysis공격에 대한 일부 대응방안으로 N라운드 특성이 구성될 확률를 낮추기 위해 DES와 HDES에 있어서 각 서브블럭에 대한 16라운드 동안 수행되는 암호함수의 반복횟수를 비교하면 그림 2와 같다.

그림 2에서 보인 바와 같이 DES는 서브블럭(R,L)에서 수행되는 암호함수의 반복횟수는 각각 8회이나, HDES는 $A_0$에서 $C_{16}$까지 수행되는 동안 11회 반복하고, $B_0$에서 $A_{16}$까지 수행되는 동안 10회 반복하며, $C_0$에서 $B_{16}$까지 수행되는 동안에는 11회 반복하므로 각 서브블럭(A,B,C)에 따라 암호함수 반복횟수가 각각 다르다.그러므로 DES는 L,R의 서브블럭에 대한 암호함수의 반복횟수가 같기 때문에 differential cryptanalysis에 약하지만 HDES는 각 서브블럭마다 암호함수의 반복횟수가 각각 다르기 때문에 differential cryptanalysis에 강하다고 할 수 있다. 또한 S-box를 $S_1 \sim S_8$에서 $S_1 \sim S_{16}$으로 증가시켰으며, S-box내의 각 엔트리를 SAC과 상관계수의 조건에 알맞는 S-box를 선정하였다. H/W 구현시에는 HDES는 3개의 서브블럭(96비트)이 병렬로 실행되므로 2개의 서브블럭(64비트)를 실행하는 DES보다도 오히려 훨씬 수행속도가 짧아져서 실시간 통신환경에 매우 적합하며 복호시 알고리즘이 암호시 알고리즘과 동일하기 때문에 암호화 알고리즘과 복호화 알고리즘이 각각 대칭되므로 암복호알고리즘을 따로 설계할 필요가 없다.

## 3. VHDL 모델링

HDES는 VHDL과 자동합성기법을 기본으로 탑다운 설계방식에 따라서 설계가 이루어진다. 여기서 탑다운 설계방식이란 회로 혹은 시스템의 특성을 가장 객관적인 단계로 부터 기술하기 시작하여 가장 기본적인 회로구성 요소까지 점점 세분화된 서브 블럭들을 통해 단계적으로 표현하는 기법을 의미한다. 그러므로 탑다운 설계기법은 설계 마지막에서 일어날수 있는 심각한 오류를 되도록이면 피하고 중요한 결정을 해야할 때에 이 결정들이 내려지기 전에 그에 해당하는 문제점들을 미리 파악하여 결정을 내리는 것이다.

HDES의 H/W설계를 기능별로 나누면 블럭단위(96비트)로 데이터를 암호화시키고 암호화된 데이터를 8비트 단위로 입출력하는 암호수행부, 데이터를 암호화시키는 각 라운드에 필요한 키값을 생성하는 키 생성부, 그리고 이들을 제어하는 제어부로 나눌 수있다. 그림 3는 암호기의 입출력 관계를 내부 블럭도로 도시하였다.

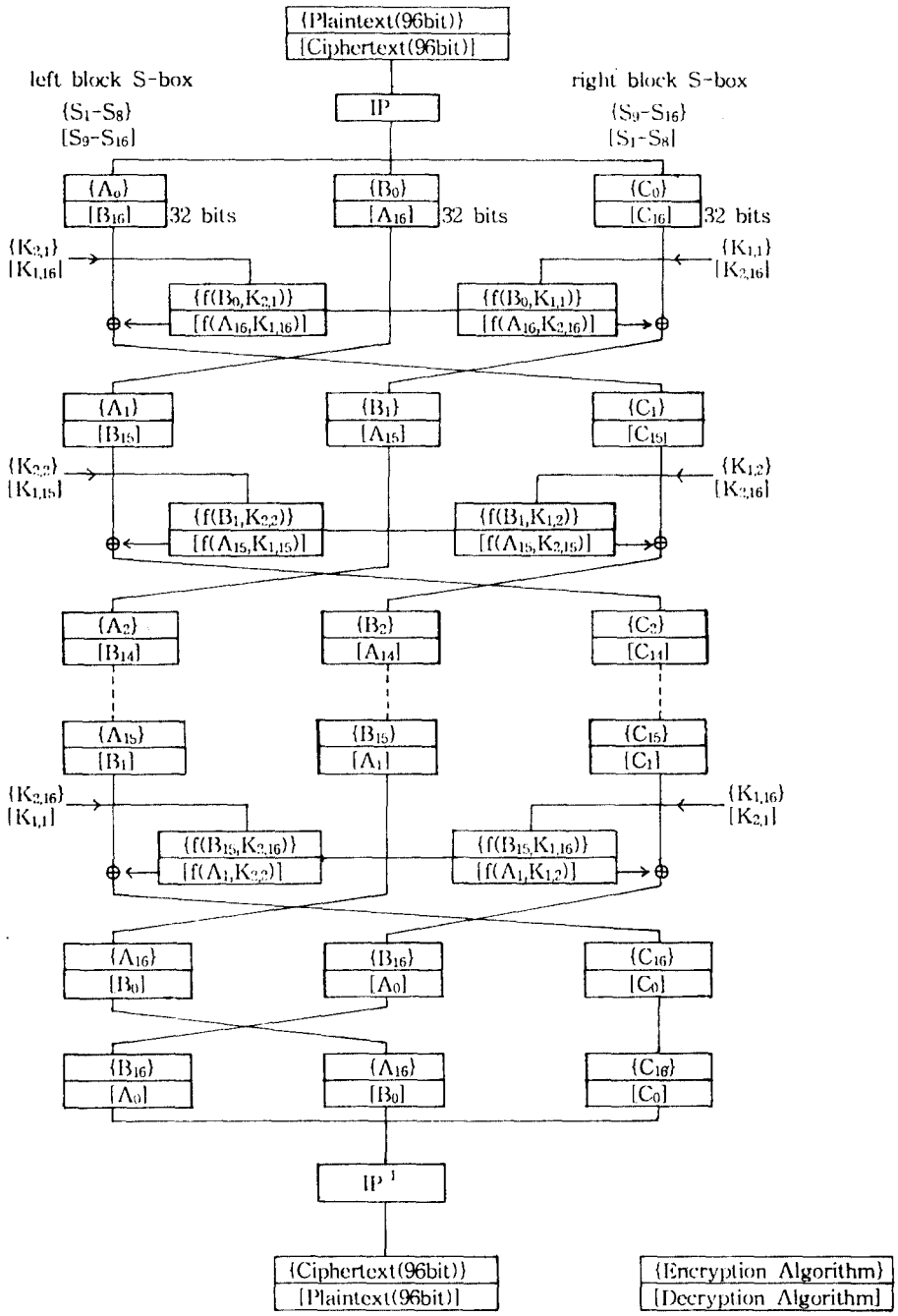



그림 1. HDES의 암복호 알고리즘
Fig. 1. Algorithm of the HDES

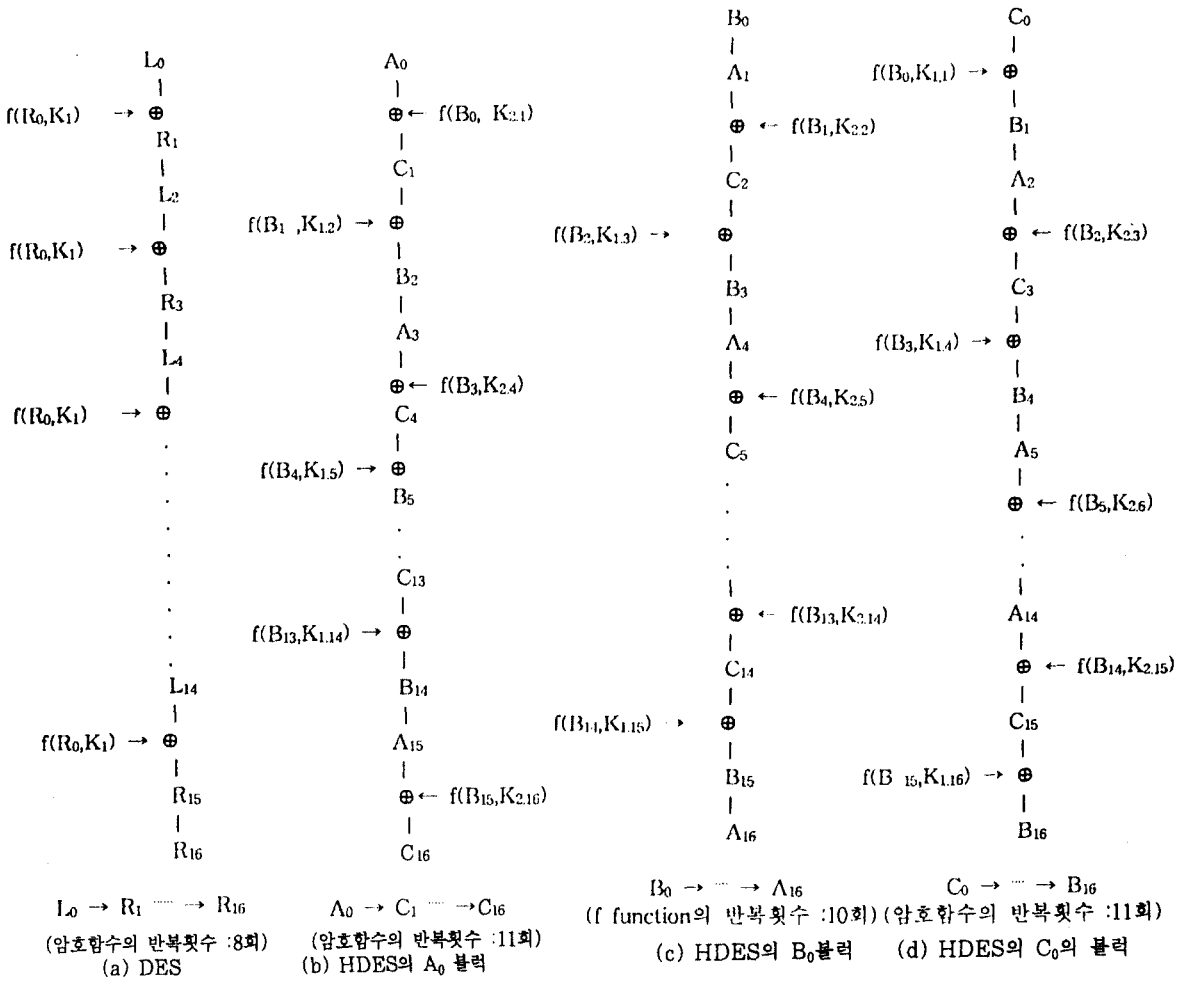



그림 2. DES와 HDES에 대한 암호함수의 반복횟수 비교
Fig. 2. Comparison in iteration number of F function between the DES and HDES

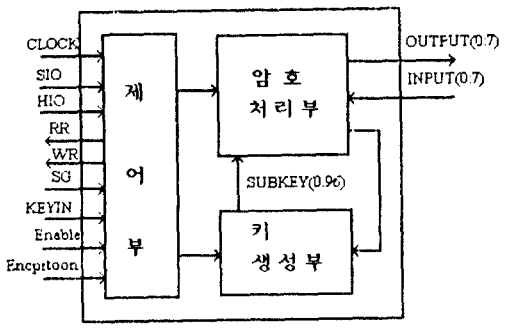



그림 3. 암호기의 내부 블럭도
Fig. 3. Internal block diagram of the HDES

### 3.1 암호처리부

C_input(7:0)와 C_output(7:0)에 의해 암복호가 수행될 데이타와 암복호화 수행시 필요한 키값을 입출력하며, 이들 입출력된 값은 암호부 레지스터에 임시 저장된다. 이중 키값은 암호부 레지스터의 출력은 IEP_box을 통해서 키생성부로 전달된다.

암복호화할 데이타의 입출력이 수행된후엔 제어신호에 따라서 IP_box에 의해 초기 치환을 수행한 후 암호화부

의 레지스터에 저장된다.

암호화 수행에 있어서는 암호부 레지스터의 출력이 세 부분으로 나뉘게된다. 암호부 레지스터 출력 중 한부분(Parall_out(63:32))은 F 함수부에서 확장치환을 거친후 키생성부로부터 입력받은 서브키(Key_sub1(47:0), key_sub2(47:0))와 XOR연산된다. 연산된 값들은 비선형치환을 거쳐 각각의 출력(F_out1, F_out2)을 내보낸다.

암호부 레지스터 출력중 나머지 두부분(Parall_out(95:64), Parall_out(31:0))은 F 합수부의 출력(F_out1, F_out2)와 XOR연산되고, 연산된 값과 암호부 레지스터 출력중 한부분(Parrall_out(63:32))이 다음 라운드의 암복호화 수행을 위해 암호부 레지스터에 임시로 저장된다. 확장된 DES에서는 이와 같은 동작이 16번의 반복 수행된다. 모든 라운드의 수행이 된후엔 제어신호에 따라서 IP_invert에 의해 최종치환을 한후 출력된다.

다음은 암호화 블럭의 VHDL 기술중 일부이다.

```
entity Cryto is
port(C_CP, C_SHLD, F_select, IP_invert : in bit;
     C_input : in bit_vector(7 downto 0);
     C_output : out bit_vector(7 downto 0);
     Key_sub1, Key_sub2 : in bit_vector(47 downto 0);
     Keyin :out bit_vector(55 downto 0));
end Cryto;

architecture Cryto_st of Cryto is
signal IP_signal, IP_invert_signal, Parallel_out,
     Load_signal : bit_vector(95 downto 0);
signal F_out1, F_out2, Xor1_signal, Xor2_signal
                          : bit_vector(31 downto 0);
     .
     .
 begin

 C0:SH_REG
     port map(CP => C_CP, SHLD => C_SHLD,
          SH_in => C_input, SH_out => C_output,
          REG_in => Load_signal, REG_out => Parallel_out);

C1:F_function
     port map(F_in => Parallel_out(63 downto 32),
          Sub_key1 => Key_sub1, Sub_key2 => Key_sub2,
          F_out1 => F_out1, F_out2 => F_out2);

C2:Vector_XOR
     generic map(N => 32);
     port map(XOR_in1 => Parallel_out(31 downto 0),
          XOR_in2 => F_out1, XOR_out => Xor1_signal);

C3:Vector_XOR
     port map(XOR_in1 => Parallel_out(95 downto 64),
          XOR_in2 => F_out2, XOR_out => Xor2_signal);

C4:IP_box
     port map(P_in => Parallel_out, P_out => IP_signal );

C5:IP_invert
     port map(P_in => Parallel_out, P_out => IP_invert_signal);

C6:IEP_box
     port map(IEP_in =>Parallel_out(63 downto 0),
          IEP_out => Keyin);

C7:SLECT_VEC
     generic map(N => 96);
     port map(S0 =>F_select, S1 =>IP_invert,
          S_in1 => Parallel_out(63 downto 32) & F_out1 &
                    F_out2,
          S_in2 => IP_signal, S_in3 => IP_invert_signal,
          S_out => Load_signal);
end Cryto_st;
```

암호화 블럭은 암호하고자 하는 블럭 데이터의 입출력을 수행하고 암호화과정에 있어서 매 라운드의 수행결과를 래치(저장)하는 레지스터부와 암호의 비도와 밀접한 관계가 있는 F함수부, 그리고 치환및 XOR로 구성된다.

3.1.1 레지스터

레지스터의 동작은 SHLD신호에 따라서 8비트 단위

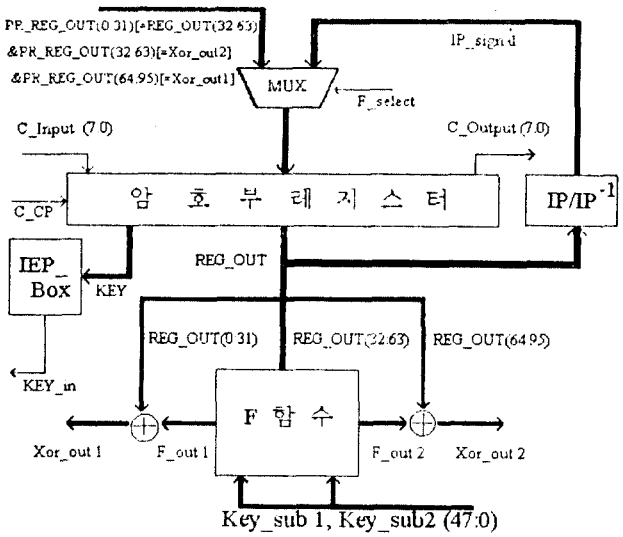



그림 4. 암호화처리의 내부블럭도
Fig. 4. Block diagram of the encryption processing

로 시프트되어 암호화할 데이터 블럭이나 키생성부의 키 값을 입출력하거나, 암호수행중 매 라운드마다 수행된 결과값을 임시로 저장하는 기능을 가진다. 여기서 SH_in(7:0)과 SH_out(7:0)은 8비트 시프트의 입력과 출력이 된다. Reg_in(95:0)과 Reg_out(95:0)은 암호 수행중에 데이터를 임시저장하고 출력하는 레지스터의 병렬입출력이다.

다음은 레지스터부의 VHDL 기술중 일부이다.

```
entity SH_REG is
      port(CP, SHLD : in bit;
      SH_in : in bit_vector(7 downto 0);
      SH_out : out bit_vector(7 downto 0);
      REG_in : in bit_vector(95 downto 0);
      REG_out : out bit_vector(95 downto 0));
end SH_REG;

architecture SR of SH_REG is
begin
      .
      .
      .
if rising_edge ( CP ) then
      .
      .
      .
if SHLD = '0' then
  PARALLEL_LOAD:
  for I in 0 to N_bit - 1 loop
   if REG_in(I) /= '0' and REG_in(I) /= '1' then
      Invalid :=1;
      exit PARALLEL_LOAD;
    elsif REG_in(I) = '1' then
    REG_value(I) := '1';
else
    REG_value(I) := '0';
   end if;
   end loop PARALLEL_LOAD;
else
   -- Shift
      .
      .
      .
  end if;
end if;
—Output
      .
```

```
end SR;
```

### 3.1.2 F_function(F함수)

F함수는 DES에서 비도을 결정하는 중요한 부분으로써 S_box, E_box, P_box와 XOR연산으로 구성되어 있다. 먼저 암호화의 블럭 데이터중 32비트를 F함수의 입력(F_in(31:0))으로 정하고 이 입력은 확장치환(E_box)을 거쳐서 32비트의 데이터를 48비트로 변환한다. 변환된 신호 48비트(E_signal)는 키 생성부에서 보내온 서브키(Sub_key1(47:0),Sub_key2(47:0))와 XOR연산되어 S_box의 입력으로 보내진다. S_box을 통과한 신호는 단순한 비트 재배열(P_box)을 거쳐 출력된다.

```
entity F_function is
  port(F_in : in bit_vector(31 downto 0);
       Sub_key1, Sub_key2 : in bit_vector(47 downto 0);
       F_out1, F_out2 : out bit_vector(31 downto 0));
end F_function;

architecture FF of F_function is

signal E_signal , K1_signal ,K2_signal : bit_vector(47 downto 0);
signal S1_signal, S2_signal : bit_vector(31 downto 0);
.
.
```

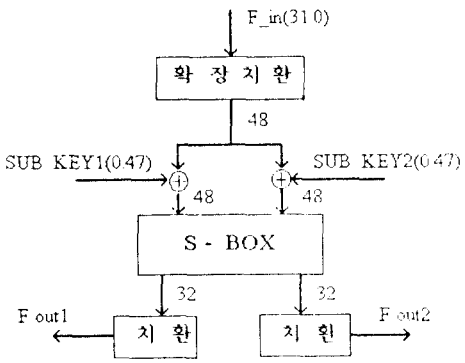



그림 5. F함수부
Fig. 5. Part of the F_Function

```
begin
  F0: E_box
      port map(E_in => F_in, E_out => E_signal);

  F1: Vector_XOR
      generic map(N => 48);
  port map(XOR_in1 => E_signal , XOR_in2 => Sub_key1,
      XOR_out => K1_signal);

  F2: Vector_XOR
      generic map(N => 48);
  port map(XOR_in1 => E_signal , XOR_in2 => Sub_key2,
       XOR_out => K2_signal);

  F3: S_box
      port map(S_in1 => K1_signal , S_in2 => K2_signal,
          S_out1 => S1_signal , S_out2 => S2_signal);

  F4: P_box
      port map(P_in => S1_signal, P_out => F_out1);

  F5: P_box
      port map(P_in => S2_signal, P_out => F_out2);
end FF;
```

### 3.1.3 E_box와 P_box

E_box는 32비트 입력을 받아서 48비트로 출력하는 확장치환을 행하는 부분이다. 우선 비트의 배열을 보면 첫번째비트와 네번째비트는 각각 두비트씩 출력과 연결된다. 두번째와 세번째는 단지 1비트의 출력과 연결된다. 그림 6에는 이와 같은 E_box의 패턴을 나타내고있

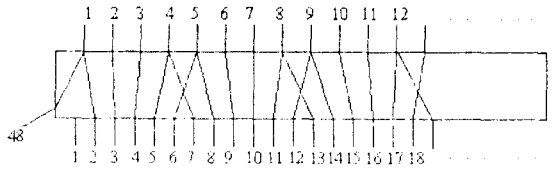



그림 6. E_box의 패턴
Fig. 6. Pattern of the E_box

다.

```
entity E_box is
   port(E_in : in bit_vector(31 downto 0);
        E_out : out bit_vector(47 downto 0));
end E_box;

architecture E_B of E_box is
begin
   process
        begin
          .
          .
          .

   E_PATTEN:
   for I in 0 to 7 loop
        E_out(I*6 + 2) <= E_in(I*4 +1);
        E_out(I*6 + 3) <= E_in(I*4 +2);
        E_out(I*6 + 4) <= E_in(I*4 +3);
        E_out(I*6 + 5) <= E_in(*4 +4);
        E_out(I*6 + 6) <= E_in(I*4 +3);
        E_out(I*6 + 7) <= E_in(I*4 +4);
end loop E_patten;
          .
          .
          .
   end process;
end E_B;
```

P_box는 32비트의 입력에 32비트의 출력을 갖는 1:1 치환이다. 다음은 P_box에 관한 기술이다.

```
entity P_box is
   port( P_in : in bit_vector(31 downto 0);
        P_out : out bit_vector(31 downto 0));
end P_box;
architecture P_B of P_box is

   Type Table_array_32 is array(0 to 31) of Positive;
   constant P_table : Table_array_32 := (15, 6, 19, 20, ..... , 24);
begin
   process
        begin
   P0:
   for A in 0 to 31 loop
        P_out(A) <= P_in(P_table(A));
   end loop P0;
  end process;
end P_B;
```

3.1.4 S_box

HDES에서는 S_box가 16개 사용되며, 에서 6비트의 입력을 4비트로 비선형치환하여 출력한다. S_box에 입력되는 6비트를 $S_1$, $S_2$, $S_3$, $S_4$, $S_5$, $S_6$로 표현했을 때 S1_box테이블에서 보인 바와같이 S1과 S6에 의한 값은 테이블의 행으로 사용되고, $S_2$, $S_3$, $S_4$, $S_5$에 의한 값은 열로 사용해서 행과 열에 해당하는 값이 4비트로 결정된다.

표 1. S1_box table
Table 1. S1_box table

| 행 | 열 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 1 | 14 | 7 | 2 | 11 | 12 | 3 | 10 | 9 | 4 | 6 | 13 | 8 | 0 | 5 | 15 |
| 1 | 13 | 15 | 4 | 5 | 12 | 6 | 1 | 11 | 7 | 2 | 14 | 3 | 0 | 10 | 8 | 9 |
| 2 | 1 | 7 | 6 | 8 | 0 | 12 | 5 | 13 | 3 | 14 | 2 | 10 | 9 | 15 | 11 | 4 |
| 3 | 10 | 12 | 14 | 9 | 7 | 6 | 1 | 2 | 11 | 15 | 3 | 8 | 4 | 13 | 0 | 5 |

```
entity S_box is
  port( S_in1 , S_in2 : in bit_vector(47 downto 0);
     S_out1 , S_out2 : out bit_vector(31 downto 0));
end S_box;
architecture S_B of S_box is
  begin
     S1:process(S_in1(0 to 5))
        begin
     if S_in1(0) = '0' then
        if S_in1(5) = '0' then
              with S_in1(4 downto 1) select
                S_out1(3 downto 0) <=
                " " when "0000",
```

```
            " " when "0010",
              .
              .
              .
                " " when "1111";
        end seleted;
  else
              .
              .
              .
    end if;
  else
              .
              .
              .
    end process;

    S2:process(S_in1(11 downto 6))
              .
              .
              .
    end process;
  end S_B;
```

3.1.5 IP_box및 IP_invert

IP_box와 IP_invert는 암호화 알고리즘중에서 처음과 끝에 동작하는 부분으로써 96비트의 입력에 96비트의 출력을 가지고 있는 일대일 치환기능을 가진다. 다음은 IP_box에 대한 VHDL기술이다.

```
  entity IP_box is
    port( P_in : in bit_vector(95 downto 0);
        P_out : out bit_vector(95 downto 0));
  end P_box;

  architecture IP_B of IP_box is

    Type Table_array_96 is array(95 downto 0) of Positive;
    constant P_table : Table_array_96 := ( , , , .... , );
  begin
    process
        begin
        P0:
        for A in 0 to 95 loop
            P_out(A) <= P_in(P_table(A));
        end loop P0;
    end process;
  end IP_B;
```

3.2.1 키생성부

키생성부는 128비트중 패리티 비트을 제거한 112비트을 키 값으로 입력 받아서 28비트씩 4블럭으로 나누어진 시프트 레지스터에 저장하며, 암호수행중 매 라운드마다 1비트 또는 2비트씩 시프트하여 그출력을 선택치환하여 두개의 서브키 Sub_key1 (47:0), Sub_key2(47:0)를 암호수행부로 전달한다.

```
  entity KEY_gen is
  port(K_CP, K_PL1, K_PL2, K_LEFT, K_SH_2bit : bit;
    K_in : in bit_vector(55 downto 0);
    Sub_key1,Sub_key2 : out bit_vector(47 downto 0));
  end KEY_gen;

  architecture K_GEN of KEY_gen is

  signal REG_Pin, REG_Pout : bit_vector(111 downto 0);
              .
              .
              .
  begin
```

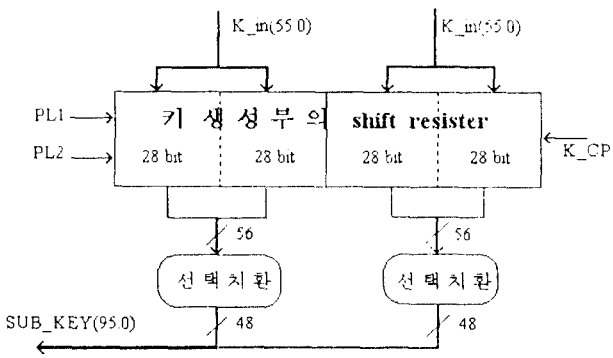



그림 7. 키생성부의 내부 블럭도
Fig. 7. Internal block diagram of the key producing part

```
KG0: Key_reg
  port map(CP => K_CP, PL1 => K_PL1, PL2 => K_PL2,
      Left => K_LEFT, SH_2bit => K_SH_2bit,
      K_reg_SH_in => K_in, K_reg_Pin => REG_Pin,
      K_reg_Pout => REG_Pout);

KG1: EP_box
  port map(EP_in => REG_Pout(55 downto 0),
      EP_out => Sub_key1);

KG2: EP_box
  port map(EP_in => REG_Pout(111 downto 56),
      EP_out => Sub_key2);

end K_GEN;
```

3.2.2 키생성부의 레지스터

암호 및 입출력부에서 키를 입력받아 패리티 비트를 제거하는 초기 선택치환을 거쳐서 키생성부의 레지스터 입력 K_in(55:0)이 된다. 이러한 입력을 두번을 반복하여 키값으로 112비트가 저장되고 저장된값은 28비트씩 4 부분의 시프트 레지스터로 나누어지며, 매 라운드마다 제어 신호 Left에 따라서 좌우로 1 또는 2비트씩 시프트 된다. 여기서 시프트 시킬때 몇 비트을 시프시킬 것인가를 위한 제어신호 SH_2bit가 사용되고, 제어신호 PL1과 PL2는 각각 키값의 전반부와 후반부를 병렬로 입력을 받기 위한 신호이다. 다음은 키생성부의 레지스터를 VHDL로 일부만 기술하였다.

```
entity Key_reg is
port(CP, PL1, PL2, Left, SH_2bit : in bit;
  K_reg_Pin : in bit_vector(55 downto 0);
  K_reg_Pout : out bit_vector(111 downto 0));
end Key_reg;

architecture K_reg of Key_reg is
begin
  process( CP )
    begin
        .
        .
        .
  if rising_edge ( CP ) then
        .
        .
        .
    LEFT_1BIT_SHIFT:
    for I in 0 to 3 loop
      Reg_buffer:= Key_reg_value(28*I+27);
      LEFT1:
      for J in 26 downto 0 loop
        Key_reg_value(28*I+J+1) :=
            Key_reg_value(28*I+J);
     end loop LEFT1;
    Key_reg_value(28*I):=Reg_buffer;
    end loop LEFT_1BIT_SHIFT;
        .
        .
        .
end K_reg;
```

3.3.1 제어부

제어부에서는 암호수행부와 키생성부의 동작을 위한 제어신호을 생성하는곳으로 HDES의 제어신호는 크게 4부분으로 나눌 수 있는데, 키 입력에 따르는 제어신호, 암호할 블럭데이터(96 비트)을 입출력하기위한 제어신호, 그리고 암호화 수행을 위한 제어신호이다. 이들은 먼저 HDES의 구동시작을 알리는 SIO(start I/O)가 ON 되면 Key_in 신호가 ON인지을 확인하여, ON이면 키생성부로 키값(128 비트)을 전달하는 동작에 필요한 제어신호를 생성한다. 키생성부로 키값의 전달이 끝나면, 암복호화 할 데이터 입출력을 위한 제어신호와 암호화수행에 필요한 제어신호을 생성한다. Key_in신호가 OFF이면, 바로 입출력과 암호화 수행에 필요한 제어신호을 생성한다.

제어부의 VHDL 기술은 네개의 process 을 사용한 다중 process구조을 가진다. 이들 네 process들은 내부 제어신호에 의해 기동과 초기화 시점을 알게된다. 제어부의 제어절차는 그림 8의 제어부 순서도에 나타나 있다.

```
entity MAIN_CONTROL is
```

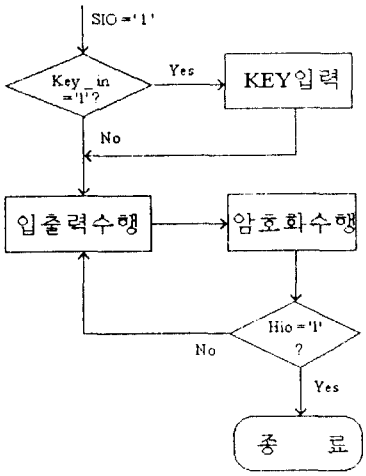



그림 8. 제어의 순서도
Fig. 8. Flowchart of control part

```
  port(K_sh_2bit, F_control, L_CP, IP_inv, WR, RR,
          CP_select, LD1, LD2 : out bit;
       CP, SG, Key_in, SIO, HIO : in bit);
end MAIN_CONTROL;

architecture M_CON of MAIN_CONTROL is

signal IO_con, CRPTO, Key_con : bit;

 begin
   INI_CON:process(SIO,HIO)
   begin
       if rising_edge ( SIO ) then
       --Initial
       CPPTO <= '0';
       if Key_in = '1' then
            Key_con <= '1';
            IO_con <= '0';
       else
            Key_con <= '0';
            IO_con <= '1';
       end if;
   end if;
   end process INI_CON;

 IO_CONTROL : process(IO_con)
              .
              .
              .
 end process IO_CONTROL;

 CRPTO_PROCESS : process(CP, CRPTO)
              .
              .
              .
  end process CRPTO_PROCESS;
  KEY_INPUT : process(IO_con)
              .
              .
              .
       end process KEY_INPUT;

 end M_CON;
```

### 3.3.2 입출력을 위한 제어신호

암호화할 블럭 데이터의 입출력을 시작한다는 IO_con 신호가 '1'이되면, 다음과 같은 제어신호을 생성한다. 먼저 WR(Write Request)신호를 '1'로 하여 데이타의 출력 요구를 한후 SG 신호가 '1'될 때까지 기다린다. SG 신호가 '1'되면 8비트 데이타를 출력하고, RR(Read Request)신호를 '1'로 하여 데이타의 입력을 요구하면서 SG신호가 '1'될때까지 기다린다. SG 신호가 '1'되면 8비트 데이터을 입력받는다. 이와 같은 동작을 12번 수행하여 96비트의 데이터을 모두 입출력하면 IO_con 신호을 '0'으로하면서 암호화 수행을 알리는 CRPTO 신호을 '1'로 하고 입출력 제어을 끝낸다. 다음은 입출력제어의 동작적 기술중 일부이다.

```
IO_CONTROL : process( IO_con, SG)
begin
```

```
  if IO_con'event and IO_con = '1' then
     WR <= '0';
     count = 0;
          .
          .
          .
end if;

if IO_con = '1' and SG'event and SG = '0' then
     if WR = '0' then
        WR <= '1' after Delay;
        CP <= '0' after Delay;
        count = count + 1;
     else
       RR <= '1' after Delay;
     end if;

      if count = 12 then
        IO_con <= '0' after Delay;
        CRPTO <= '1' after Delay;
     end if;
end if;

if IO_con = '1' and SG'event and SG = '1' then
     if WR = '1' then
        WR <= '0'after Delay;
     else
        CP <= '1';
        RR <= '0' after Delay;
     end if;
end if;

end process IO_CONTROL;
```

### 3.3.3 암호화 수행을 위한 제어신호

입출력을 위한 제어가 끝나면, 암호화 수행의 시작을 알리는 CRPTO 신호가 '1'된다. CRPTO가 '1'로되면 암호화 수행을 위한 모든신호는 초기화되고, 초기화된 후에는 모든 동작은 CP에 의해 동기화된다. 다음은 제어신호에 따른 암호화부의 동작이다.

1. F_control = '1' : F 함수를 동작시킨다.

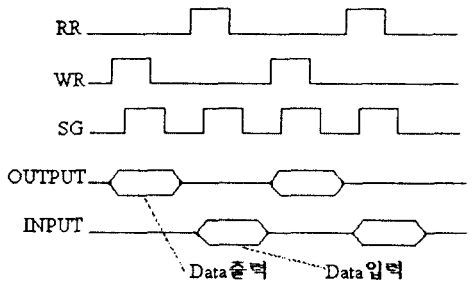



그림 9. 입출력 제어신호의 타이밍도
Fig. 9. Timming chart of the input/output control signal

2. F_control = '0' and IP_inv = '0' : IP_box를 동작시킨다.

3. F_control = '0' and IP_inv = '1' : IP_invert를 동작시킨다.

( K_sh_2bit는 F함수에 전달할 서브키 생성에 있어서 시프트될 비트수을 결정하기 위한 제어신호이다.)

다음은 암호화수행을 위한 제어의 동작적 기술이다.

```
 CRPTO_PROCESS : process( CP , CRPTO)
 variable count : integer;
  begin
 if CRPTO'event and CRPTO = '1' then
       count := 0;
       CP_select <= '1';
       F_control <= '0';
       IP_inv <= '0';
 end if;

 if CP'event and CP = '1' then
       case count is
            when 0 =>
              F_control <= '1' after Delay;
              K_sh_2bit <= '0' after Delay;
            when 1 - 15 =>
              if table_array (count) = 1 then
                K_sh_2bit <= '1' after Delay;
              else
                K_sh_2bit <= '0' after Delay;
              end if;
```

```
        when 16 =>
          F_control <= '0' after Delay;
          IP_inv <= '1' after Delay;
        when 17 =>
          IP_inv <= '0' after Delay;
          CRPTO <= '0' after Delay;
          IO_con <= '1' after Delay;
          CP_select <= '0' after Delay;
      end case;
      count := count + 1;
    end if;
  end process CRPTO_PROCESS;
```

### 3.3.4 키값 입력을 위한 제어신호

키생성부의 키값을 입력 받기위해서 암호부의 입출력 기능을 사용하여 64비트 씩 두번에 걸쳐 키값을 입력받게 된다. 키값을 입력받을 시기는 암호화 수행이 되기전에 즉 SIO가 ON되었을 때 Key_in이 ON 이면 키값을 입력받게된다. 키값을 입력받는 절차는 다음과 같다.

1. RR(read request)신호을 '1'로하고 SG가 '1'이 될 때까지 기다린다.

2. SG가 '1' 이되면, 8비트을 읽어 들이고(CP <= '1') RR을 '0'으로 한다.

3. 1.과 2.와 같은 동작을 8번 반복수행 후 IEP_box을 통해 패리티 비트을 제거하고 키값을 키 생성부 레지스터상반부에 입력한다.

4. 1., 2., 와같은 동작을 8번 반복수행 후 IEP_box을 통해 패리티비트을 제거하고 키 값을 키 생성부 레지스터 후반부에 입력한다.

다음은 키입력을 위한 제어신호 생성을 위한 동작적기술중 일부이다.

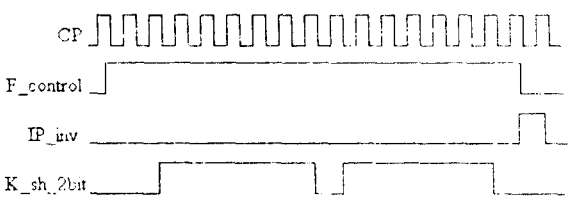



그림 10. 암호화 제어신호 타이밍도
Fig. 10. Timming chart of the encryption control signal

```
KEY_INPUT process(Key_con,SG)
        .
        .
        .
begin
  if Key_con'event and Key_con = '1' then
    RR <= '1' after Delay;
    LD1 <= '0';
    LD2 <= '0';
    CP_count = 0;
        .
        .
        .
end if;

if Key_con = '1' and SG'event and SG ='1' then
    RR <= '0' after Delay;
    CP <= '1' after Delay;
end if;

if Key_con = '1' and SG'event and SG ='1' then
    CP <= '0' after Delay;
    CP_count = CP_count + 1 ;
    if CP_count = 8 then
       LD1 <= '1' after Delay;
       L_CP <= '1' after Delay;
    end if;
    if CP_count = 16 then
       LD2 <= '1' after Delay;
       L_CP <= '1' after Delay;
  IO_con <= '1' after Delay;
  Key_con <= '0' after Delay;
    end if;
  end if;
end process KEY_INPUT;
```

## 4. 시뮬레이션 및 검토

본 확장된 DES(HDES)는 Intergraph사의 EDA 툴을 사용하여 워크스테이션에서 시뮬레이션을 수행하였다. 시뮬레이션은 평문을 입력받아 암호화후 비문을 입

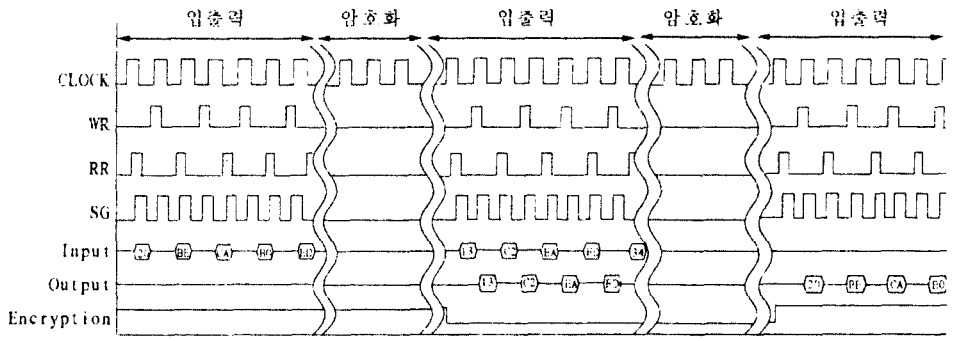
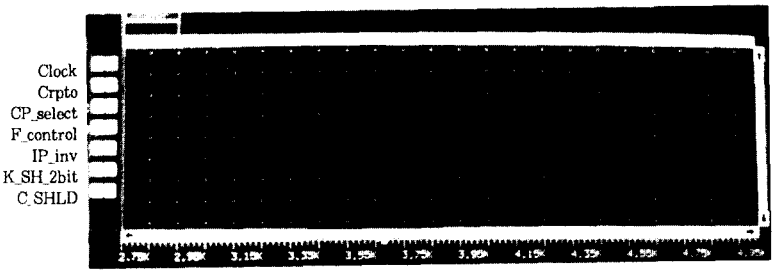
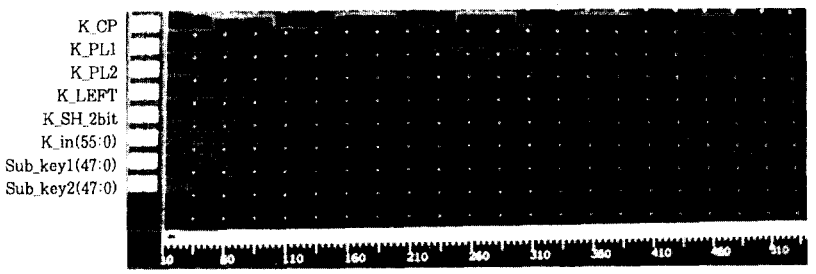



그림 11. 확장된 DES의 암·복호 시뮬레이션
Fig. 11. Encryption/decryption simulation of the HDES




a. 제어부




b. 키생성부

그림 12. 제어부와 키생성부에 대한 시뮬레이션
Fig. 12. Simulation of the control part and key producing part

표 2. 시뮬레이션시 입출력된 평문과 비문
Table 2. Plaintext and Ciphertext inputted and outputted in simulation

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 평문 | 20 | BE | CA | B0 | ED | 20 | C0 | D6 | B4 | D9 | 2E | 0D | 0A | BB | E7 | C8 |
| | EA | 20 | B5 | BF | BE | C8 | 20 | B0 | E8 | BC | D3 | 20 | B7 | D0 | B0 | AD |
| | C0 | BB | 20 | C7 | E0 | C7 | CF | B8 | E9 | BC | AD | 20 | B3 | AA | B4 | C2 |
| 비문 | 13 | C2 | EA | FD | 34 | 97 | D8 | 0F | 78 | 27 | 3C | 2B | B5 | 0E | 8B | C1 |
| | 54 | 0F | 7D | FB | 9D | 6B | E1 | 35 | 57 | 99 | 26 | 3F | 9A | D7 | 6E | 3B |
| | 4A | C8 | 36 | B5 | 81 | DD | E7 | E2 | DB | 86 | 2E | 07 | B1 | E9 | C6 | D4 |

출력하고, 입출력된 비문을 복호화 하여 평문으로 출력하였으며, 이때 입력된 평문과 출력된 평문은 동일함을 그림 11에서 알수 있다.

그림12는 제어부와 키생성부의 시뮬레이션이며, 표2에는 키값을 CHOSUNBAEYUNGSUN$_{(ASCII)}$으로 하였을 때 평문 입력에 대한 출력을 나타내었다.

## 5. 결 론

세계적으로 가장 많이 사용하고 있는 암호알고리즘인 DES의 비도를 증가시키기 위해 암호분석의 평가지표에 따라 DES를 확장한 암호기인 HDES를 H/W로 구현하기 위해서 VHDL를 이용하여 탑다운 방식으로 설계하였다.

암호기는 정상적인 동작을 위해 내부클럭암호처리부와 키 생성부, 그리고 제어부로 나누어 설계하였다. 암호처리부는 F함수을 이용한 암호수행과 96비트 블럭 데이타를 8비트 단위로 입출력을 수행하도록 하였고, F함수는 비도를 설정하는 중요한 부분으로 키 생성부에서 생성된 두 서브키(48비트)와 32비트의 입력 데이타를 받아 S-box를 이용하여 암호화시켰다. 키 생성부는 128비트를 키로 입력을 받아 선택치환을 거쳐 패리티 비트를 삭제하고 선택 재배열된 112비트는 28비트씩 네 블럭으로 나누어 시프트 처리했다. 제어부에서는 외부 입출력에 관계한 레지스터 조작과 암호 수행과정의 데이타 흐름에 대한 제어코드를 생성하도록 하였다

본 논문에서 구현한 암호기는 기밀성이 요구된 화일 저장시에 활용할 수 있으며, 컴퓨터 통신망에 적용하여 정보보호시스템개발에 효과적으로 활용할 수 있을 것이다.

## 참고문헌


1. 한승조, "실시간 통신환경에 있어서 인증성을 갖는 압축과 암호의 결합시스템", 충북대학교 박사졸업 논문, 1994.
2. M. Abrams, and H. Podell, "Computer & Network Security-Tutorial," IEEE computer soc. Press. 1987, pp.245-258.
3. NBS, "Data Encryption Standard," FIPS Pub. 46, U.S, National Bureau of Standard, Washington DC, Jan. 1977.
4. Data Encryption Algorithm, American National Standard X3, 92, ANSI, NY, 1981.
5. W. Diffie and M. E. Hellman, "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," IEEE, Vol.10, No.6, 1977, pp.74-84.
6. D. E. Denning, Cryptography and Data Security, Addison-Wesley, 1983.
7. M. E. Hellman, "A Cryptanalytic Time Memory Trade off," IEEE Trans. of Info. Theory, Vol. IT-26, No.4, July. 1980, pp.401-406.
8. E. Biham and A. Shamir, "Differential Cryptanalsis of DES-like Cryptosystem," Weizmann Institute of Science, Technical Report, Rehovot, Israel, 19 July. 1990.
9. E. Biham and A. Shamir, "Differential Cryptanalysis of Feal and N-Hash," Eurocrypt'91 Astracts, A J Clark, Ed.,IACR,Brighton, UK, April. 1991, pp.8-11.
10. R. Forre, "The Strict Avalanche Criterion: Special Properties of Boolean Function and an Extended Definition," Proc. of Crypto'88, Springer-Verlag, 1988.
11. B. D. Boer, "Cryptanalysis of FEAL," Advances in Cryptology, Proc. of EURCRYPT'87, Springer-Verlag, 1989, pp.167-173.
12. L. Brown, J. Pieprzyk and J. Seberry, "LOKI-A Cryptographic Primitive for Authentication and Secrecy," Proc. of AUSCRYPT'90, Jan. 1990, pp.222-228.
13. L. R. Knudsen, "Cryptanalysis of LOKI," Abstracts of ASCIACRYPT'91, Nov. 1991, pp.19-23.
14. Z. Navadi, "Using VHDL for modeling and design of processing unit," Proceedings of the 1992 ASIC Conf. and Ex., pp.315-326, 1992.

**吳 幸 洙**(Haeng Soo Oh) 정회원

1958년 10월 5일생
1980년 2월 : 조선대학교 전자공학과 졸업
1982년 2월 : 조선대학교 대학원 전자공학석사
1992년 9월~현재 : 조선대학교 대학원 전기공학과 박사과정
1984년 1월~1993년 2월 : 금성알프스전자 근무
1993년 3월~현재 : 동신전문대학 전자과 전임강사
※주관심 분야 : 정보보호 및 암호학

**韓 承 朝**(Seung Jo Han) 정회원

1956년 7월 2일생
1980년 2월 : 조선대학교 전자공학과 졸업
1982년 2월 : 조선대학교 대학원 전자공학석사
1994년 2월 : 충북대학교 대학원 전자계산학과 이학박사
1984년~현재 : 조선대학교 전자공학과 부교수
1986년 6월~1987년 3월 : Univ. of New Oleans 객원교수
1995년 2월~현재 : Texas 주립대학 객원교수
※주관심 분야 : 암호학 및 마이크로프로세서