Soft Computing And Next generation Network Lab.

Hanyang Univ.

Verilog HDL
조합회로

목차

- 순차 처리문
- 데이터 ·형
- 연산자
- 설계하기

순차 처리문

- 논리회로는 입력으로부터 출력을 얻는 과정을 보통의 프로그램에서와 같이 순차적인 절차로 기술할 수 있으며 initial문 또는 always문과 같은 순차 처리문이 이러한 절차를 기술하기 위해서 사용된다.
- initial문은 처음에 한 번만 수행되는 동작을, always문은 신호가 변할 때마다 반복하여 수행되는 동작을 기술한다.
- 순차 처리문 내에서는 보통의 프로그래밍 언어에서와 같이 할당문, 조건문, 반복문등이 사용될 수 있다.
- 순차 처리문은 다른 순차 처리문과 모듈/프리미티브 인스턴스 및 연속할당문들과는 병렬로 수행된다.

순차 처리문

- initial 문

```
initial 문장; // 하나의 문장 포함

initial begin // 여러 개의 문장들을 포함
     문장;
     문장;
...
  end
```

  - 일반적으로 신호의 초기화, 모니터링을 위한 파형출력
    시뮬레이션 수행시에 한 번만 수행되는 과정을 기술하는 데 사용
  - 여러 개의 initial문이 있으면 이들은 독립적으로 실행
  - 실행하는 문장이 여러 개가 있으면 begin과 end를 사용하여 블록으로 묶어야 하며 블록 내의 문장들은 순차적으로 실행

순차 처리문

- always 문

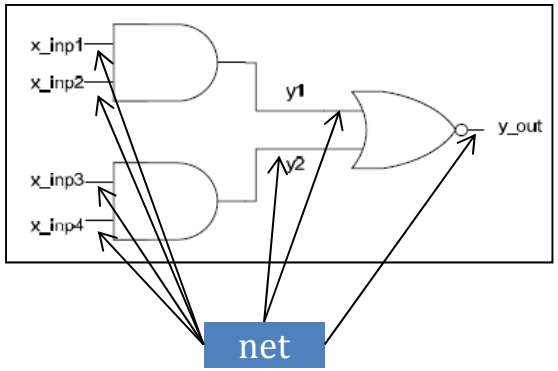
```
always 반복조건
   문장;
always 반복조건 begin // 여러 개의 문장들을 포함
     문장;
     문장;
     ...
   end
```

  - 반복조건이 만족할 때마다 문장을 반복하여 수행
  - C 언어의 무한 루프와 비슷한 개념, $finish 나 $stop에 의해서만 정지

데이터 ·형

데이터 ·형	의 미
네트(nets)	디바이스의 물리적인 연결
레지스터(registers)	추상적인 저장 장치
파라미터(parameters)	상수 선언

- 네트
  - 하드웨어 요소 사이의 연결을 나타냄
  - 네트 값이 변할 때 자동으로 네트에 새로운 값이 전달됨
  - 신호가 저장되지 않음



데이터 ·형(net)

벡터 혹은 비트로 지정

네트·형 [범위] [지연] 네트 이름, … , 네트이름;

벡터 혹은 비트로 지정

키워드	정의
wire, tri	일반적 넷에 사용
wor, trior	OR 연산을 하는 wire
wand, triand	AND 연산을 하는 wire
trireg	값을 저장하는 넷에 사용
tri1	넷에 아무것도 인가되지 않으면 1
tri0	넷에 아무것도 인가되지 않으면 0
supply1	Vdd를 나타냄
supply0	Ground를 나타냄

데이터 ·형(register)

- Registers
  - 일종의 변수로 사용, 일시적으로 데이터 저장,
    하드웨어 레지스터를 의미하지는 않음
  - 새로운 값이 할당될 때 까지는 현재의 값을 그대로 유지

벡터 혹은 비트로 지정

레지스터·형 [범위] 레지스터 이름, … , 레지스터 이름;

키워드	정의
reg	일반적으로 사용되는 레지스터
real	실수나 과학적 표기법에 사용되는 레지스터
time	시뮬레이션 시간을 저장하기 위한 레지스터
integer	정수 양을 다루기 위한 범용레지스터 **signed** 형태가 가능

회로설계에 사용

회로설계에 사용

데이터 ·형(register)

```
reg A;                                  //1비트 레지스터
reg [3:0] X;                            //4비트 레지스터 X
reg [7:0] M,N;                          //8비트 레지스터 M,N
X reg [3:0] CNT4, [7:0] CNT8;           //문법에러
```

```
reg [7:0] MEMORY[0:255];                //8비트 X 256 크기의 메모리
/*레지스터 배열은 비트 선택과 부분 선택은 할 수 없고,
 필히 워드 단위로 선택*/
MEMORY[100];                            //100번지의 내용
/*비트 선택과 부분 선택을 하려면 네트를 사용*/
wire [7:0] BYTE;                        //네트 선언
wire BIT3;
        assign BYTE = MEMORY[100];      //100번지의 내용를 복사
        assign BIT3 = BYTE[3];          //100번지 비트 번호 3을 복사
```

데이터 ·형(parameters)

- 회로의 비트 크기나 지연값 지정

상수 혹은 결과가 상수가 되는 식

parameter [범위] 이름 = 식;

```
parameter high_index = 31;                          //정수
parameter width = 32, depth = 1024;                 //정수
parameter byte_size = 8, byte_max = byte_size -1;   //정수
parameter a_real_value = 6.22;                      //실수
parameter av_delay = (min_delay + max_delay)/2;     //실수
parameterinitial_state = 8'b1001_0110;              //레지스터
```

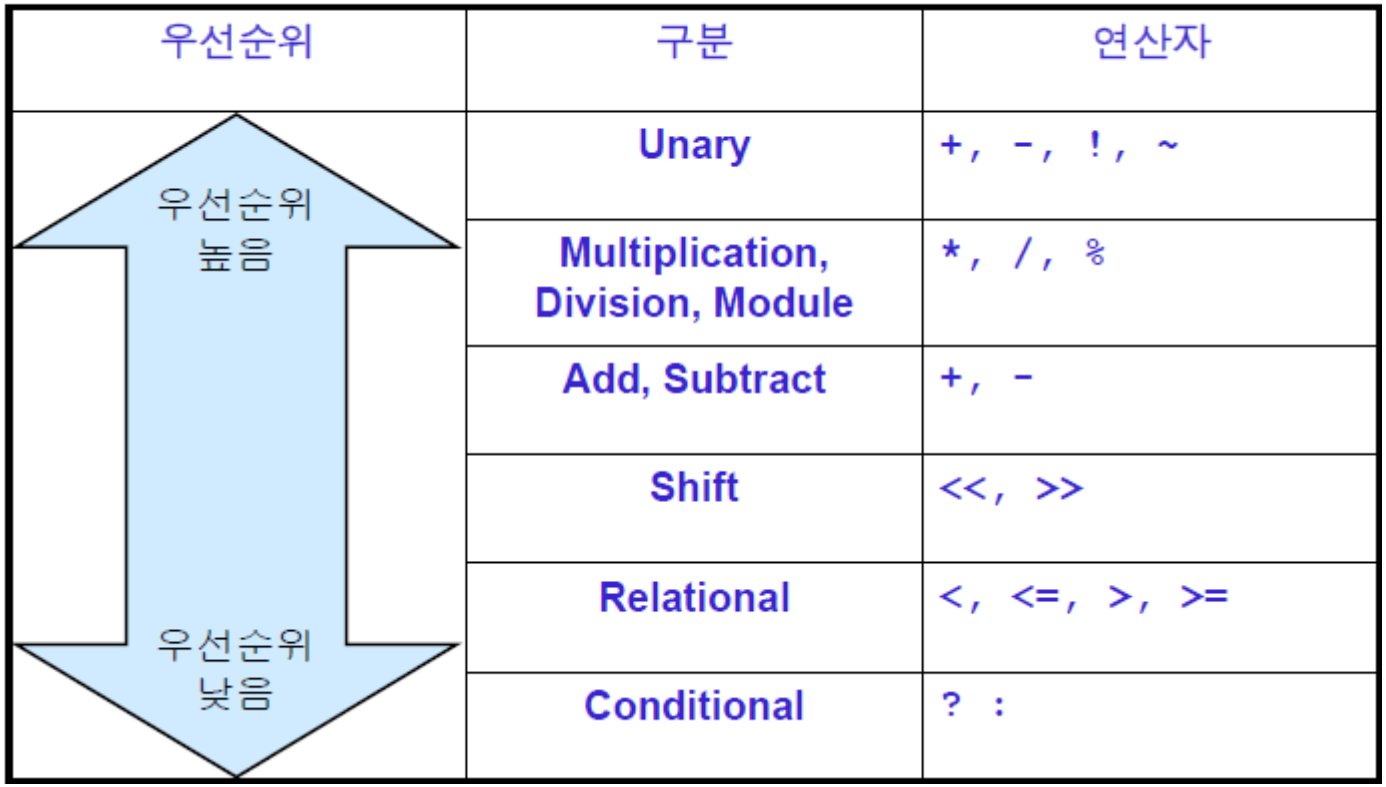
연산자

구분	연산자	의미	연산자	의미
산술 연산자	+	덧셈	-	뺄셈
	*	곱셈	/	나눗셈
	%	나머지		
관계연산자	==	같다	!=	같지않다
	<	작다	<=	작거나 같다
	>	크다	>=	크거나 같다
논리연산자	&&	논리적AND	||	논리적 OR
	!	논리적NOT		

구분	연산자	의미	연산자	의미
논리연산자	~	비트 NOT	&	비트 AND
	|	비트 OR	^	비트 XOR
	^~, ~^	비트 XNOR		
시프트연산자	>>	오른쪽 shift	>>	왼쪽 shift
그밖의 연산자	? :	조건연산자	{ }	결합 연산자

연산자

◆ 연산자 우선순위



우선순위	구분	연산자
	Unary	+, -, !, ~
	Multiplication, Division, Module	*, /, %
	Add, Subtract	+, -
	Shift	<<, >>
	Relational	<, <=, >, >=
	Conditional	? :

연산자

- 조건연산자

예 16 덧셈, 뺄셈 조건 연산자의 예

```
module ADD_OR_SUBTRACT( A, B, OP, RESULT ); // 모듈 시작
  parameter  ADD=1'b0;        // ADD는 1비트 상수("0")로 선언
  input      [7:0] A, B;      // A,B 입력은 8비트 벡터
  input      OP;              // OP는 1비트 입력
  output     [7:0] RESULT;    // 8비트 출력
  // OP가 ADD면 RESULT = A+B, OP가 ADD가 아니면 RESULT = A-B
  assign     RESULT = (OP == ADD) ? A + B : A - B;
endmodule                     // 모듈 끝
```

예 17 네스티드(nested) 조건 연산자의 예

```
module  ARITHMETIC( A, B, OP, RESULT ); // 모듈 시작
  parameter  ADD=3'h0, SUB=3'h1, AND=3'h2, OR=3'h3,  XOR=3'h4;  // 상수 선언
  input   [7:0] A,B;        // A,B 입력은 8비트 벡터
  input   [2:0] OP;         // OP는 1비트 입력
  output  [7:0] RESULT;     // 8비트 출력
  assign  RESULT = ((OP == ADD) ? A + B : (  // OP가 ADD인 경우 덧셈
                    (OP == SUB) ? A - B : (  // OP가 SUB인 경우 뺄셈
                    (OP == AND) ? A & B : (  // OP가 AND인 경우 논리 AND
                    (OP == OR) ?  A | B : (  // OP가 OR 인 경우 논리 OR

                   //OP가 XOR인 경우 논리 XOR, 그 이외의 조건에서는 A가 전달
                   (OP == XOR) ?  A ^ B :( A ))))));
endmodule              // 모듈 끝
```

설계하기

- assign 문을 이용하라.

➢ 전가산기 설계

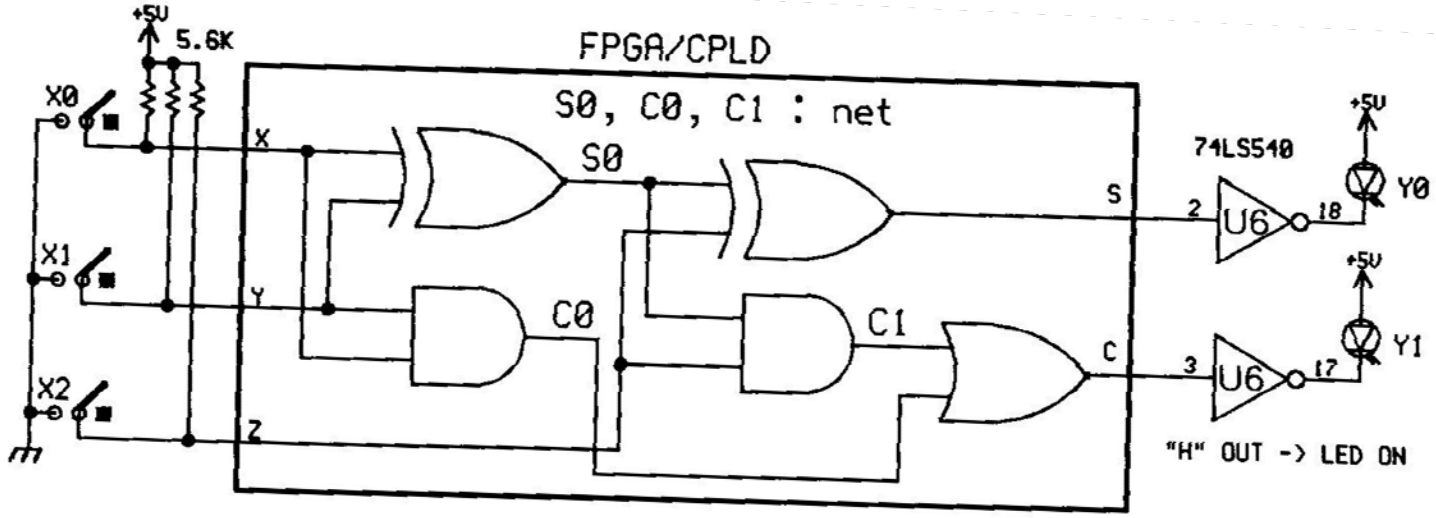


그림 3-2. 전 · 가산기를 실험하기 위한 회로도

```
module FULL_ADDER(X,Y,Z,C,S);
  input X,Y,Z;              //입력
  output C,S;               //출력
  wire S0,C0,C1;            //중간 출력을 와이어로 선언
    xor U1(S0,X,Y);         //S0 = X XOR Y
    and U2(C0,X,Y);         //C0 = X AND Y
    and U3(C1,S0,Z);        //C1 = S0 AND Z
    or U4(C,C1,C0);         //C(캐리) = C1 OR C0
    xor U5(S,S0,Z);         //S(합) = S0 XOR Z
endmodule
```

```
`timescale 1ns/1ns
`include "FULL_ADDER.v"
module testbench;
  reg X,Y,Z;
  wire C,S;

FULL_ADDER ADDER(.X(X),.Y(Y),.Z(Z),.C(C),.S(S));
  initial begin
    $dumpfile("wave.vcd");
    $dumpvars(0, testbench);
    $monitor("X=%d Y%b Z=%b, sum=%b C=%b",X,Y,Z,S,C);

    #5 X=0; Y=0; Z=0;
    #5 X=0; Y=0; Z=1;
    #5 X=0; Y=1; Z=0;
    #5 X=0; Y=1; Z=1;
    #5 X=1; Y=0; Z=0;
    #5 X=1; Y=0; Z=1;
    #5 X=1; Y=1; Z=0;
    #5 X=1; Y=1; Z=1;
  end
endmodule
```

➢ 산술 연산자 이해하기

➢ 입출력 및 기능 선택은 벡터로 선언

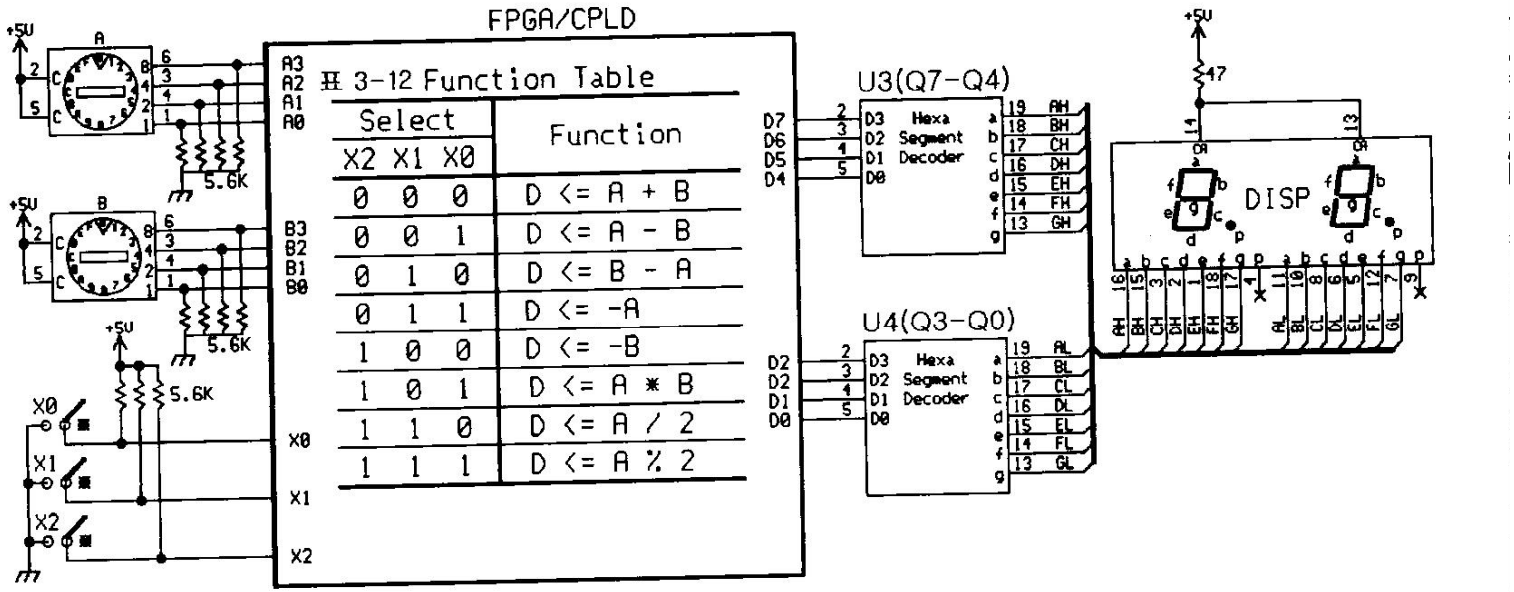


그림 3-8. 산술 연산자의 연습 회로도

```
module OPERATOR(A,B,X,D);      // 모듈 시작
  parameter  ADD=3'h0,SUB=3'h1,COMP=3'h2,NEG1=3'h3, // 상수 선언
             NEG2=3'h4,MUL=3'h5,DIV=3'h6,REM=3'h7;
  input [3:0] A,B;     // A,B 오퍼런드 4비트 입력
  input [2:0] X;       // 기능 선택 3비트 입력
  output [7:0] D;      // 출력 8비트
    assign  D = (X == ADD) ? A + B :  // X가 ADD면 A + B
                (X == SUB) ? A - B :  // X가 SUB면 A - B
                (X == COMP) ? B - A : // X가 COMP면 B - A
                (X == NEG1) ? -A :    // X가 NEG1면 -A
                (X == NEG2) ? -B :    // X가 NEG2면 -B
                (X == MUL) ? A * B :  // X가 MUL면 A 곱하기 B
                (X == DIV) ? A / 2 :  // X가 DIV면 A 나누기 2
                // X가 REM면 A/2의 나머지, 그 이외의 조건에서는 A값이 전달
                (X == REM) ? A % 2 : A ;
endmodule        // 모듈 끝
```

```
`timescale 1ns/1ns
`include "OPERATOR.v"
module testbench;
   reg[3:0] A,B;
   reg[2:0] X;
   wire[3:0] D;

OPERATOR BCD(.A(A),.B(B),.X(X),.D(D));
   initial begin
      $dumpfile("wave.vcd");
      $dumpvars(0, testbench);

      A=4'h5; B=4'h2; X=3'h0;#10;
      A=4'h5; B=4'h2; X=3'h1;#10;
      A=4'h5; B=4'h2; X=3'h2;#10;
      A=4'h5; B=4'h2; X=3'h3;#10;
      A=4'h5; B=4'h2; X=3'h4;#10;
      A=4'h5; B=4'h2; X=3'h5;#10;
      A=4'h5; B=4'h2; X=3'h6;#10;
      A=4'h5; B=4'h2; X=3'h7;#10;
   end
endmodule
```

Verilog HDL
– 조합회로 응용

논리설계 및 실험

Soft Computing And Next generation Network Lab.

목 차

- 동작 레벨 기술
  - Function문
  - Block문
  - If문
  - Case문
  - For문
- 설계하기
  - Decoder
  - Encoder
  - Parity

동작레벨 기술

- “assign”문은 논리식을 그대로 나열하여 표현
  이런 방법을 데이터 흐름(Data flow)기술 이라 함

- 이와 반면 회로도와 논리식을 고려하지 않고,
  그 회로의 동작을 기술하는 방법이 있음
  이런 방법을 동작 레벨(behavioral) 기술이라 함

function 문

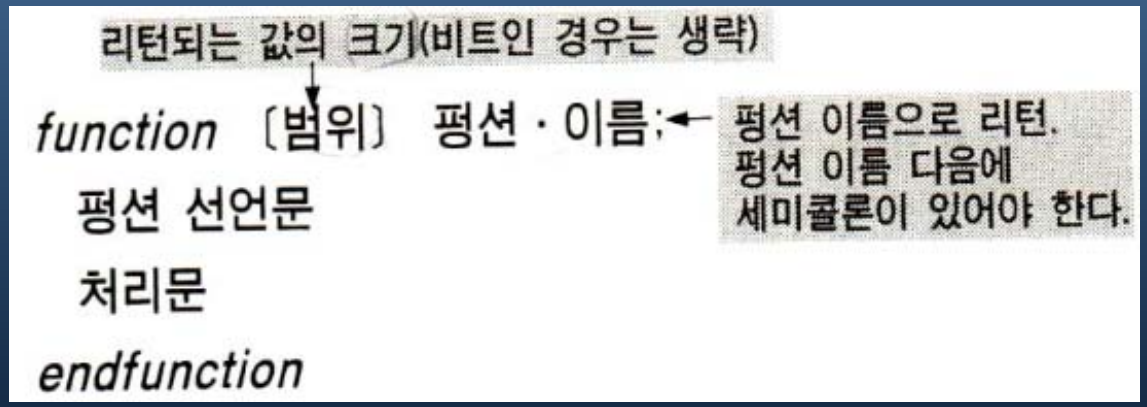
- 함수 호출을 인라인(in-line) 코드로 호출하여
  함수의 출력을 합성하면 조합회로의 일부분이 됨

예) XOR함수를 “function”문으로 사용한 경우

```
module EXOR (IN1, IN2, OUT);                    //모듈 시작
   input    IN1, IN2;                           //입력
   output   OUT;                                //출력
            assign OUT = EXOR_FUNC (IN1, IN2);  //function 호출
   function EXOR_FUNC;                          //function 이름
   input IN1, IN2;                              //입력 선언
            if(IN1 ^ IN2)                       //(IN1 XOR IN2) 결과가 "1"이면 실행
               EXOR_FUNC = 1;                   //function 이름으로 return
            else                                //(IN1 XOR IN2) 결과가 "0"이면 실행
               EXOR_FUNC = 0;                   //function 이름으로 return
   endfunction                                  //function 문 끝
endmodule                                       //모듈 끝
```

function 문

- “function”문 형식



(1) function내부에서 하나 이상의 문장을 사용할 수 있으며, 하나 이상의 문장을 사용하려면 “begin ~ end” 블록으로 둘러싸야 함

(2) function내에 선언된 지역(local) 변수는 전부 일시적인 것으로 취급되며, 이와 같은 변수는 “wire”로 합성됨

(3) function 정의에는 타이밍을 정의하는 문장을 포함하면 안됨

(4) function 정의는 module안에 있어야 함

(5) function은 순차 처리문에서 설명하는 “task”를 사용할 수 없는 반면, “task”에서는 function을 사용할 수 있음

function 문

- “function”문 호출 방법
  function은 function자체의 이름으로 단지 한 개의 값만 return할 수 있는데, 이 값을 받는 형식은 다음과 같음

  네트 = 펑션 · 이름 (인수 · 이름,…,인수 · 이름)

  – 입력 선언

    function이름 다음에 바로 입력을 선언해야 하며, 적어도 하나 이상의 “input”이 있어야 하고, “output”, “inout” 포트는 사용 불가능 함

    input **〔범위〕 입력 · 변수,…, 입력 · 변수**

  – function에서의 출력

    function에서 출력은 function이름에 할당되며, 출력은 하나만 가질 수 있음 연결 연산자 [{ }]를 사용하면, 여러 개의 출력을 하나로 묶어서 return 가능

  – 다른 선언문

    reg, parameter, integer등을 선언할 수 있음

block 문

- “begin ~ end”라는 block문은 여러 개의 문을 하나의 문으로 취급하기 위하여 사용하는 것

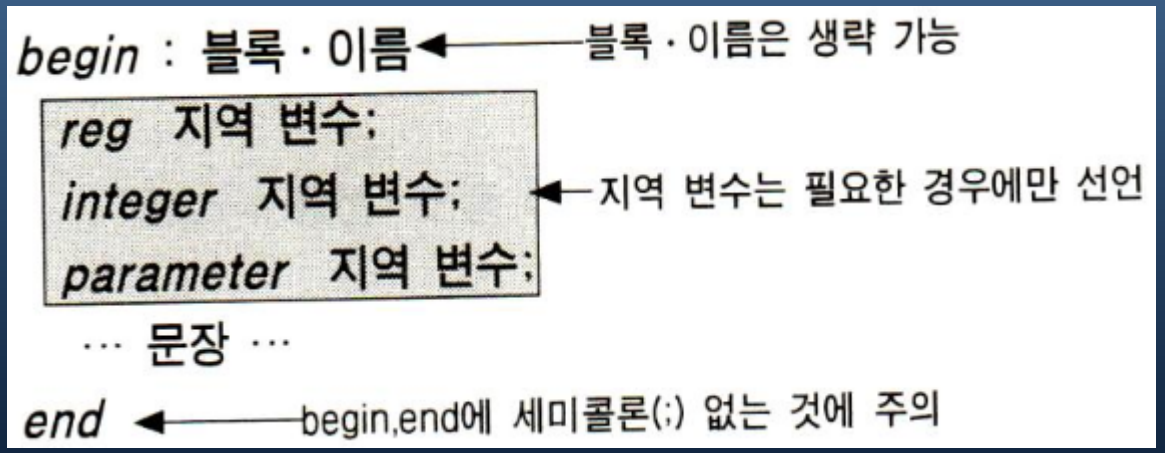
  (1) 복문을 1개의 문(statement)으로 취급함
  (2) 순차 처리 block을 명확히 나누게 함
  (3) 기술한 것을 보기가 쉬움

  – 문(statement) 이란?

    문이란 [assign A =B;]와 같이 [;]으로 구분한 것을 의미,
    if문, case문도 하나의 문으로 취급가능
    그러나, 이런 방법을 사용하면 if문에서 하나의 문밖에 쓸 수 없으므로,
    여러 개의 문을 쓰는 방법을 생각해야함

block 문

– 여러 개의 문을 하나의 문으로 취급



- 이름이 있는 블록 안에서는 지역변수들을 선언할 수 있지만, 이름이 없는 블록 안에서는 지역변수를 선언할 수 없음
- 이름 있는 블록 안에서는 “disable”문을 사용할 수 있음
- if, case문 등에서 한 문장 이상을 사용할 때는 반드시 블록문을 사용
- function, always 문에서 여러 개의 문장을 하나로 묶을 때 많이 사용, always 문에서 주로 사용하기 때문에 “순차처리 블록”이라 부름
- 병렬 처리를 위한 병렬 처리 블록인 “fork ~ join”이 있지만, 실제 설계에는 사용 불가능, 시뮬레이션에서만 사용됨

if 문

- "if ~ else"문
  (1) 블록문이 없는 "if ~ else"문의 형식

```
if (조건식)
    문장 1;   ← <조건식>이 조건이 맞으면 실행
else          ← 이 이하는 생략이 가능하지만, 조합회로에서는 생략하면 불필요한 래치가 만들어진다.
    문장 2;   ← <조건식>이 조건이 맞지 않으면 실행
```

  (2) 블록문이 있는 "if ~ else"문의 형식

```
if (조건식)
begin
    … 문장 …   ← <조건식>이 참("1")이면 블록을 실행
end
else
begin
    … 문장 …   ← <조건식>이 거짓("0")이면 블록을 실행
end
```

if 문

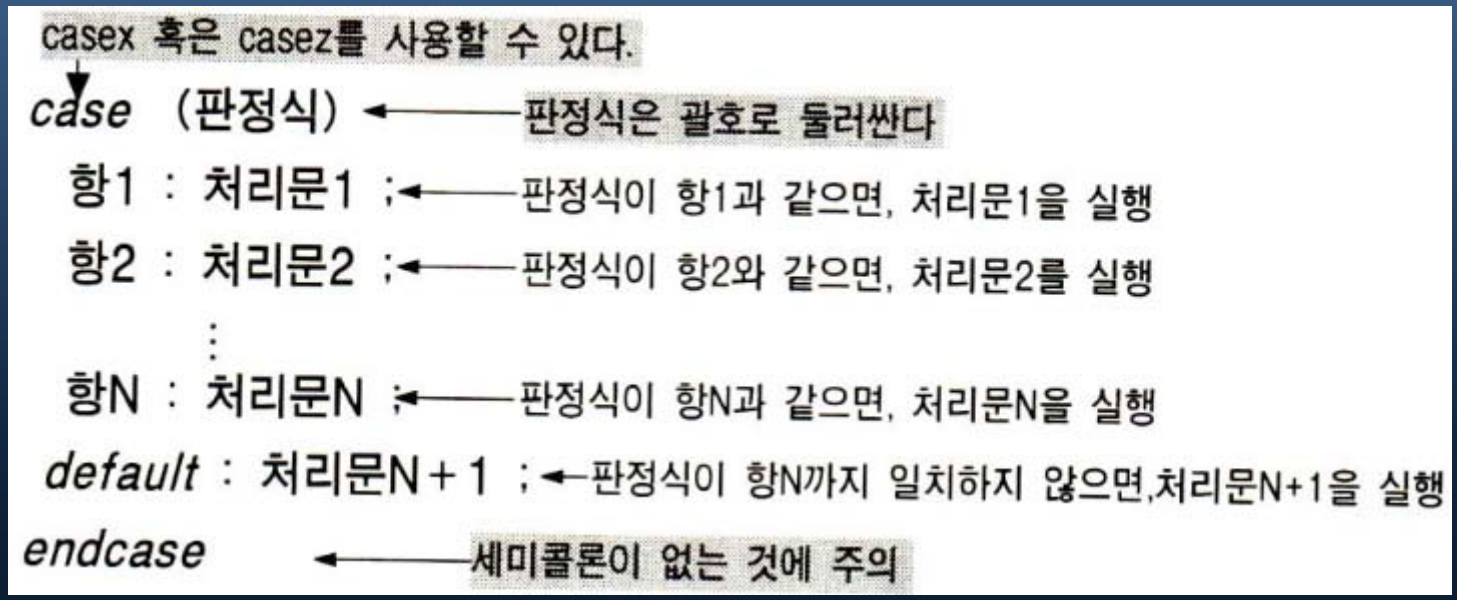
- Nested “if ~ else”문

  “if ~ else”문에서 “else” 다음에 “if”문을 또 사용하는 것
  5개를 넘지 않는 것이 하드웨어 구성상 좋음

```
if(SELECT[1])                                   //SELECT[1]=1일 경우
   begin                                        //블록문 시작
            if(SELECT[0]) OUT=IN[3];            //SELECT[0]=1일 경우
            else OUT=IN[2];                     //SELECT[0]=0일 경우
   end                                          //블록문 끝
else                                            //SELECT[1]=0일 경우
   begin                                        //블록문 시작
            if(SELECT[0]) OUT=IN[1];            //SELECT[0]=1일 경우
            else OUT=IN[0];                     //SELECT[0]=0일 경우
   end                                          //블록문 끝
```

case 문

- "case" 문장 형식



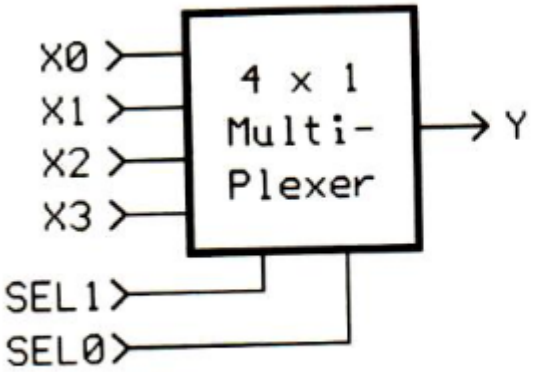
(1) "항1 ~ 항n"은 한번 사용한 것은 사용불가, 보통 상수를 사용, "판정식"에 식 혹은 여러 식을 사용할 경우 ","로 분리

(2) 처리문에서 여러 문장을 처리하려면, 블록문으로 묶어야 함

(3) 판정식에서 판정가능한 모든 값을 "항1 ~ 항n"에서 사용하지 않았을 경우 "default"문으로 사용하지 않은 "항"에 대해 처리 해야 함

case 문

- 예제



FUNCTION TABLE

SEL1	SEL0	Y
0	0	X0
0	1	X1
1	0	X2
1	1	X3

그림 4-1. 데이터 선택기

```
/* 4-1 데이터 선택기 */
module SEL (X0, X1, X2, X3, SEL, Y);                    //모듈시작
   input X0, X1, X2, X3;                                //입력을 각각 비트로 선언
   input [1:0] SEL;                                     //선택선을 벡터로 선언
   output Y;                                            //출력
          assign Y=FUNC_SEL (X0, X1, X2, X3, SEL);      //function호출, 출력은 1비트
   function FUNC_SEL;                                   //function이름
   input X0, X1, X2, X3;                                //입력 선언
   input [1:0] SEL;                                     //입력 선언
          case (SEL)                    //조건을 모두 나열했기에 "default"는 없어도 됨
            0 : FUNC_SEL = X0;          //SEL=0이면 X0을 function이름으로 return
            1 : FUNC_SEL = X1;          //SEL=1이면 X1을 function이름으로 return
            2 : FUNC_SEL = X2;          //SEL=2이면 X2을 function이름으로 return
            3 : FUNC_SEL = X3;          //SEL=3이면 X3을 function이름으로 return
          endcase                       //case문 끝(세미콜론 없음을 주의)
   endfunction                                          //function문 끝
end                                                     //모듈 끝
```

case 문

- “casex, casez” 문
  “case”문을 변형한 것, 비교하는데 무관 조건 사용 가능

  (1) “case”문에서는 “항1 ~ 항n”에 “0, 1, x, z”의 네 가지 항을 취급

  (2) “casex”문에서는 “항1 ~ 항n”에 “x, z, ?”의 값이 있으면,
  “casex (판정식)”의 대응되는 비트는 비교하지 않음

  (3) “casez”문에서는 “항1 ~ 항n”에 “z, ?”의 값이 있으면,
  “casez (판정식)”의 대응되는 비트는 비교하지 않음

  -> (주의) 위에서 제기한 문제들은 시뮬레이션할 때 발생 됨

```
"casex", "casez"문 사용 예
reg[3:0] cond;
casex (cond)                    //cond=4'1000 혹은 cond=4'b1001일때 OUT=1
   4'b100x : OUT = 1;
   default : OUT = 0;
endcase

casez (cond)                    //cond=4'1000 혹은 cond=4'b1001일때 OUT=1
   4'b110z : OUT = 1;
   default : OUT = 0;
endcase
```

for 루프문

- “for” 루프문의 형식

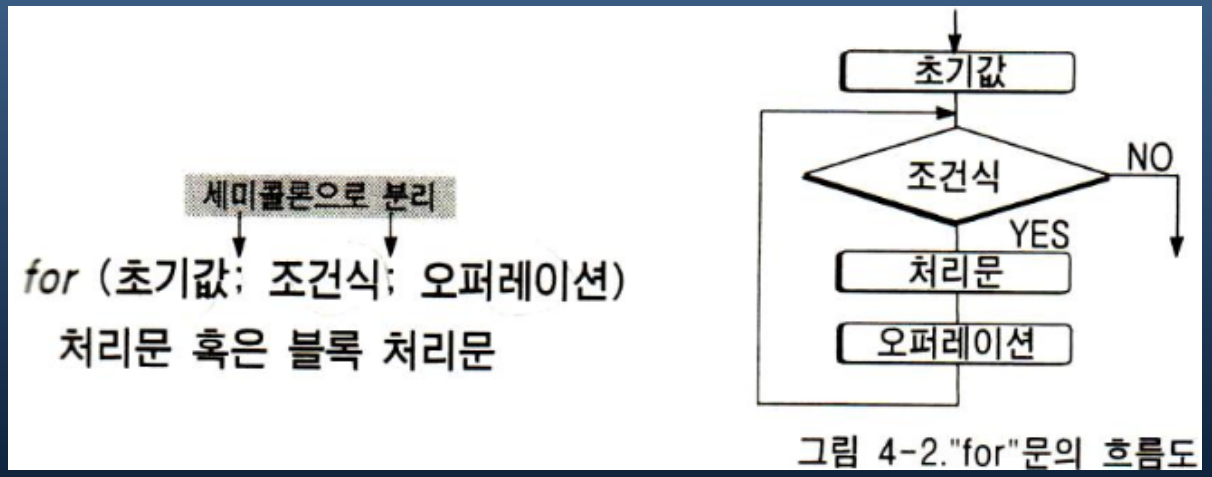


그림 4-2."for"문의 흐름도

-> “for”문에서 사용하는 인덱스(index)는 “integer”로 선언되야 함

- “for” 루프문의 사용 가능 형식

For (index = low_range; index < high_range; index = index + step)
For (index = high_range; index > low_range; index = index - step)
For (index = low_range; index <= high_range; index = index + step)
For (index = high_range; index >= low_range; index = index - step)

for 루프문

- 예제

```
(A의 하위 비트 ~ 상위 비트) AND (B의 상위 비트 ~ 하위 비트)
for (i=0; i<8; i=i+1)
   C[i] = A[i] & B[7-i];

//연산결과
            C[0] = A[0] & B[7];
            C[1] = A[1] & B[6];
            C[2] = A[2] & B[5];
            C[3] = A[3] & B[4];
            C[4] = A[4] & B[3];
            C[5] = A[5] & B[2];
            C[6] = A[6] & B[1];
            C[7] = A[7] & B[0];
```

설계하기

- 디코더 (decoder)

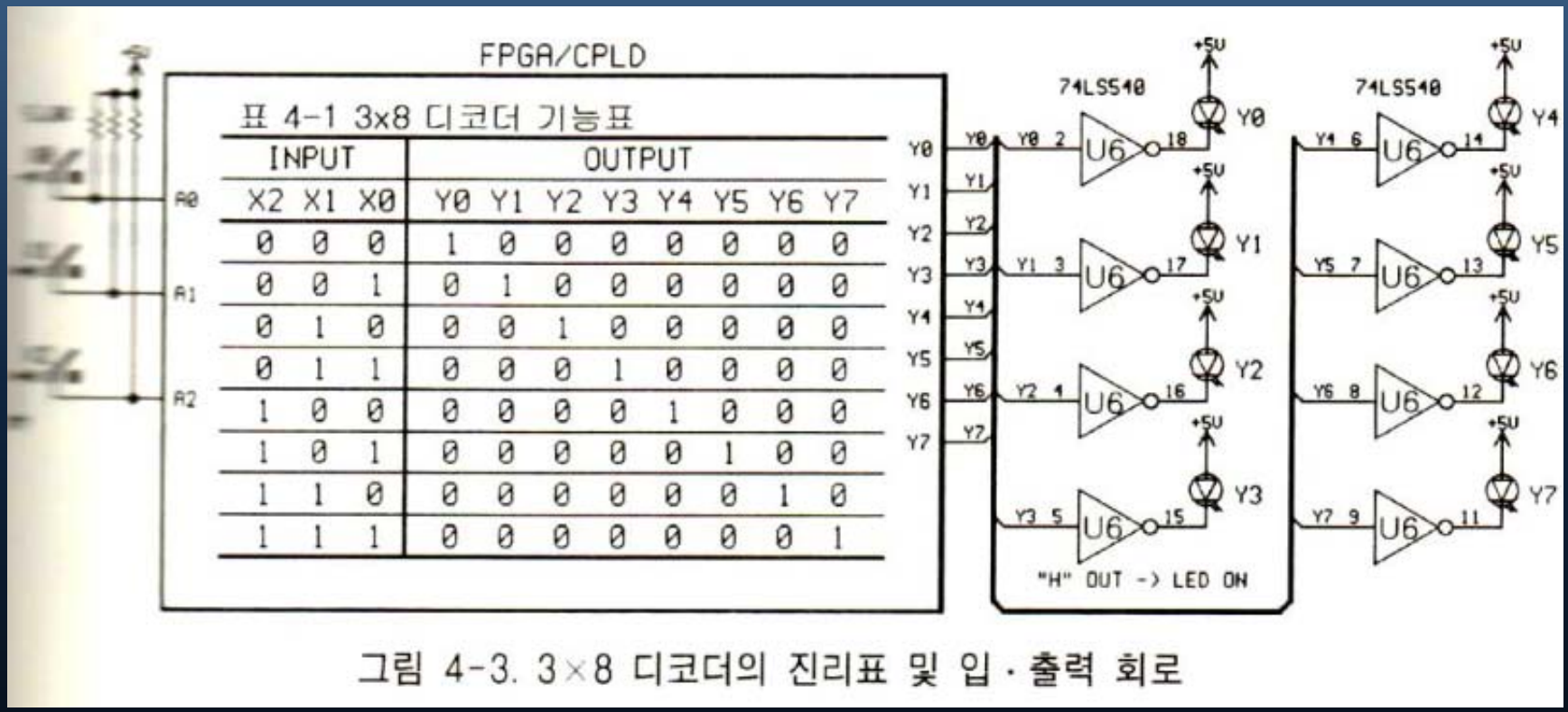


그림 4-3. 3×8 디코더의 진리표 및 입·출력 회로

설계하기

```
//시프트 연산자 이용하기
module DECODER (A, Y);
   input [2:0] A;                    //입력 3비트
   output [7:0] Y;                   //출력 8비트
            assign Y = 1'b1 << A;          //입력값만큼 "1"을 왼쪽으로 이동해서 Y에 대입
endmodule
```

```
//관계 연산자 이용하기
module DECODER (A, Y);
   input [2:0] A;                    //입력 3비트
   output [7:0] Y;                   //출력 8비트
          //관계 연산자의 비교 결과가 참이면 1이 됨.
          assign Y[0] = (A == 3'h0);       //A가 "000"이면 Y[0]=1
          assign Y[1] = (A == 3'h1);       //A가 "001"이면 Y[1]=1
          assign Y[2] = (A == 3'h2);       //A가 "010"이면 Y[2]=1
          assign Y[3] = (A == 3'h3);       //A가 "011"이면 Y[3]=1
          assign Y[4] = (A == 3'h4);       //A가 "100"이면 Y[4]=1
          assign Y[5] = (A == 3'h5);       //A가 "101"이면 Y[5]=1
          assign Y[6] = (A == 3'h6);       //A가 "110"이면 Y[6]=1
          assign Y[7] = (A == 3'h7);       //A가 "111"이면 Y[7]=1
endmodule
```

설계하기

```
//"case"문 이용하기
module DECODER (A, Y);
   input [2:0] A;                    //입력 3비트
   output [7:0] Y;                   //출력 8비트
            assign Y = FUNC_DECODER (A);          //function을 콜한다. return된 값을 Y로 출력
            // 디코더 function. retrun된 값은 FUNC_DECODER 8비트
   function [7:0] FUNC_DECODER;
   input [2:0] A;                    //입력 3비트
            case (A)
               0 : FUNC_DECODER = 8'b0000_0001;  //A가 "000"인 경우
               1 : FUNC_DECODER = 8'b0000_0010;  //A가 "001"인 경우
               2 : FUNC_DECODER = 8'b0000_0100;  //A가 "010"인 경우
               3 : FUNC_DECODER = 8'b0000_1000;  //A가 "011"인 경우
               4 : FUNC_DECODER = 8'b0001_0000;  //A가 "100"인 경우
               5 : FUNC_DECODER = 8'b0010_0000;  //A가 "101"인 경우
               6 : FUNC_DECODER = 8'b0100_0000;  //A가 "110"인 경우
               7 : FUNC_DECODER = 8'b1000_0000;  //A가 "111"인 경우
               //조건을 모두 나열했기 때문에, "default"항은 없어도 됨.
               default : FUNC_DECODER = 8'b0000_0000;       //A가 일치되는 값이 없을 경우
            endcase
   endfunction
endmodule
```

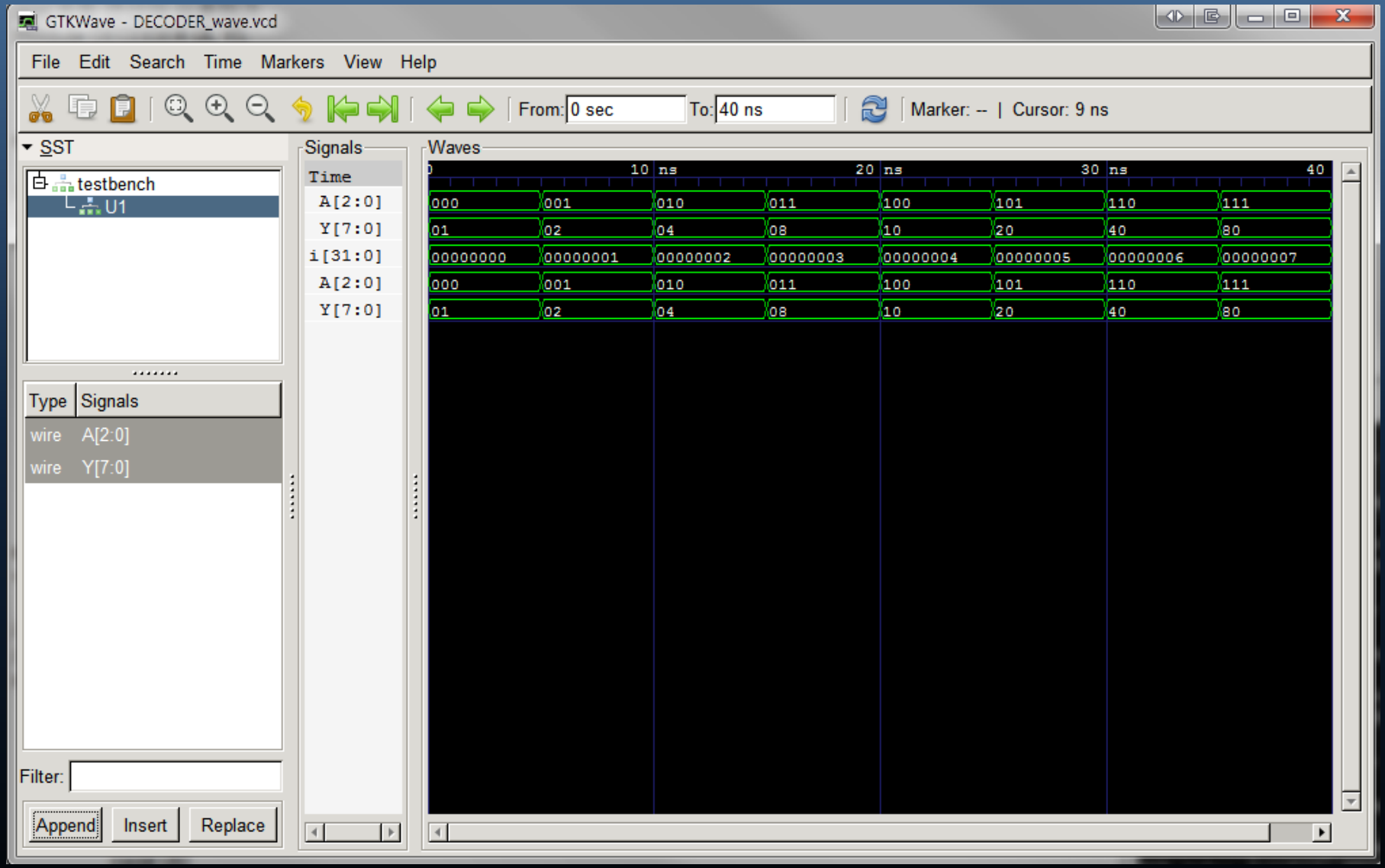
설계하기

```
//"if"문 이용하기
module DECODER (A, Y);
   input [2:0] A;                    //입력 3비트
   output [7:0] Y;                   //출력 8비트
            assign Y = FUNC_DEC (A);      //function을 콜한다. return된 값을 Y로 출력
            // 디코더 function. retrun된 값은 FUNC_DEC 8비트
   function [7:0] FUNC_DEC;
   input [2:0] A;                    //입력 3비트
            if (A == 3'b000)         //A가 "000"인 경우
                     FUNC_DEC = 8'b0000_0001;
            else                     //A가 "000"이 아닌 경우
              if (A == 3'b001)                  //A가 "001"인 경우
                        FUNC_DEC = 8'b0000_0010;
              else if (A == 3'b010)                     //A가 "010"인 경우
                        FUNC_DEC = 8'b0000_0100;
              else if (A == 3'b011)                     //A가 "011"인 경우
                        FUNC_DEC = 8'b0000_1000;
              else if (A == 3'b100)                     //A가 "100"인 경우
                        FUNC_DEC = 8'b0001_0000;
              else if (A == 3'b101)                     //A가 "101"인 경우
                        FUNC_DEC = 8'b0010_0000;
              else if (A == 3'b110)                     //A가 "110"인 경우
                        FUNC_DEC = 8'b0100_0000;
              else if (A == 3'b111)                     //A가 "111"인 경우
                        FUNC_DEC = 8'b1000_0000;
              else                  //조건이 맞는 경우가 없을 경우
                        FUNC_DEC = 8'b0000_0000;
   endfunction
endmodule
*/
```

설계하기

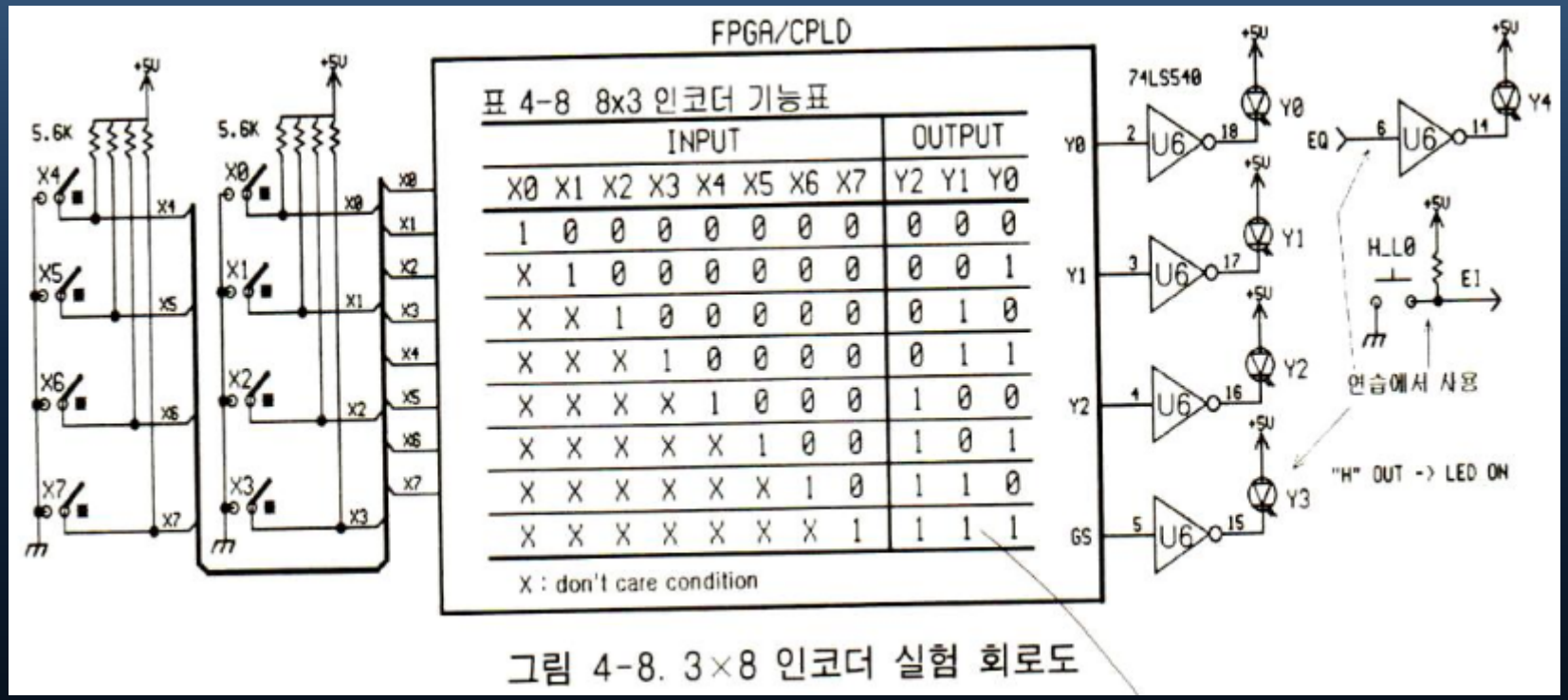
```
`include "DECODER.v"
`timescale 1ns/1ns
module testbench;
   reg [2:0] A;                    //입력을 레지스터로 선언
   wire [7:0] Y;                   //출력을 와이어로 선언
   integer i;                      //루프에 사용하는 인덱스
   //테스트벤치에서 자동으로 만들어진 것에 인스턴스 이름(U1)만 변경함
DECODER U1 (.A(A), .Y(Y));
   initial begin                   //블록문 시작
            $dumpfile ("DECODER_wave.vcd");
            $dumpvars (0, testbench);
            A = 0;                 //입력 초기값 설정
              for (i=0; i<=7; i=i+1)                    //i=0부터 7까지 루프
              //앞의 신호를 100[ns] 동안 유지하고 난 후, A+1
                       #5 A = A + 1;
   end
endmodule
```

설계하기


GTKWave - DECODER_wave.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 40 ns
Marker: -- | Cursor: 9 ns
SST
testbench
U1
Type Signals
wire A[2:0]
wire Y[7:0]
Filter:
Append Insert Replace
Signals
Time
A[2:0]
Y[7:0]
i[31:0]
A[2:0]
Y[7:0]
Waves
10 ns 20 ns 30 ns 40
000 001 010 011 100 101 110 111
01 02 04 08 10 20 40 80
00000000 00000001 00000002 00000003 00000004 00000005 00000006 00000007
000 001 010 011 100 101 110 111
01 02 04 08 10 20 40 80

설계하기

- 인코더 (encoder)



설계하기

```
//조건 연산자 이용하기
module ENCODER (X0, X1, X2, X3, X4, X5, X6, X7, Y);
   input X0, X1, X2, X3, X4, X5, X6, X7;          //입력을 각각 비트로 선언
   output [2:0] Y;                                //출력을 벡터로 선언
          assign Y = X7 ? 3'b111 :                //X7이 "1"이면 Y=111
                     X6 ? 3'b110 :                //X6이 "1"이면 Y=110
                     X5 ? 3'b101 :                //X5이 "1"이면 Y=101
                     X4 ? 3'b100 :                //X4이 "1"이면 Y=100
                     X3 ? 3'b011 :                //X3이 "1"이면 Y=011
                     X2 ? 3'b010 :                //X2이 "1"이면 Y=010
                     X1 ? 3'b001 :                //X1이 "1"이면 Y=001
                     X0 ? 3'b000 : 3'bzzz;        //X0이 "1"이면 Y=000 아니면 하이 임피던스
endmodule
```

설계하기

```
//"if"문 이용하기
module ENCODER (X, Y);
   input [7:0] X;                    //입력 8비트
   output [2:0] Y;                   //출력 3비트
              assign Y = FUNC_ENC (X);     //입력 X에 대한 출력 Y를 출력
             //encoder function
             function [2:0] FUNC_ENC;
             input [7:0] X;
                //else문과 if문 합쳐서 사용
                if (X[7]) FUNC_ENC = 3'h7;   //X7이 "1"이면 Y=111
                else if (X[6]) FUNC_ENC = 3'h6;          //X6가 "1"이면 Y=110
                else if (X[5])  FUNC_ENC = 3'h5;         //X5가 "1"이면 Y=101
                else if (X[4])  FUNC_ENC = 3'h4;         //X4가 "1"이면 Y=100
                else if (X[3])  FUNC_ENC = 3'h3;         //X3가 "1"이면 Y=011
                else if (X[2])  FUNC_ENC = 3'h2;         //X2가 "1"이면 Y=010
                else if (X[1])  FUNC_ENC = 3'h1;         //X1가 "1"이면 Y=001
                else if (X[0])  FUNC_ENC = 3'h0;         //X0가 "1"이면 Y=000
                else FUNC_ENC = 3'hz;                    //아니면 Y=zzz
             endfunction
endmodule
```

설계하기

```
//"case"문 이용하기
module ENCODER (X, Y);
   input [7:0] X;                    //입력 8비트
   output [2:0] Y;                   //출력 3비트
            assign Y = FUNC_ENC (X);     //입력 X에 대한 출력 Y를 출력
           //encoder function
           function [2:0] FUNC_ENC;
           input [7:0] X;
                    casex (X)
                       8'b1xxx_xxxx : FUNC_ENC = 3'h7;      //X가 "1xxx_xxxx"이면 FUNC_ENC=111
                       8'b01xx_xxxx : FUNC_ENC = 3'h6;      //X가 "01xx_xxxx"이면 FUNC_ENC=110
                       8'b001x_xxxx : FUNC_ENC = 3'h5;      //X가 "001x_xxxx"이면 FUNC_ENC=101
                       8'b0001_xxxx : FUNC_ENC = 3'h4;      //X가 "0001_xxxx"이면 FUNC_ENC=100
                       8'b0000_1xxx : FUNC_ENC = 3'h3;      //X가 "0000_1xxx"이면 FUNC_ENC=011
                       8'b0000_01xx : FUNC_ENC = 3'h2;      //X가 "0000_01xx"이면 FUNC_ENC=010
                       8'b0000_001x : FUNC_ENC = 3'h1;      //X가 "0000_001x"이면 FUNC_ENC=001
                       8'b0000_0001 : FUNC_ENC = 3'h0;      //X가 "0000_0001"이면 FUNC_ENC=000
                       default : FUNC_ENC = 3'h0;           //조건이 없는 경우 FUNC_ENC=000
                    endcase
           endfunction
endmodule
```

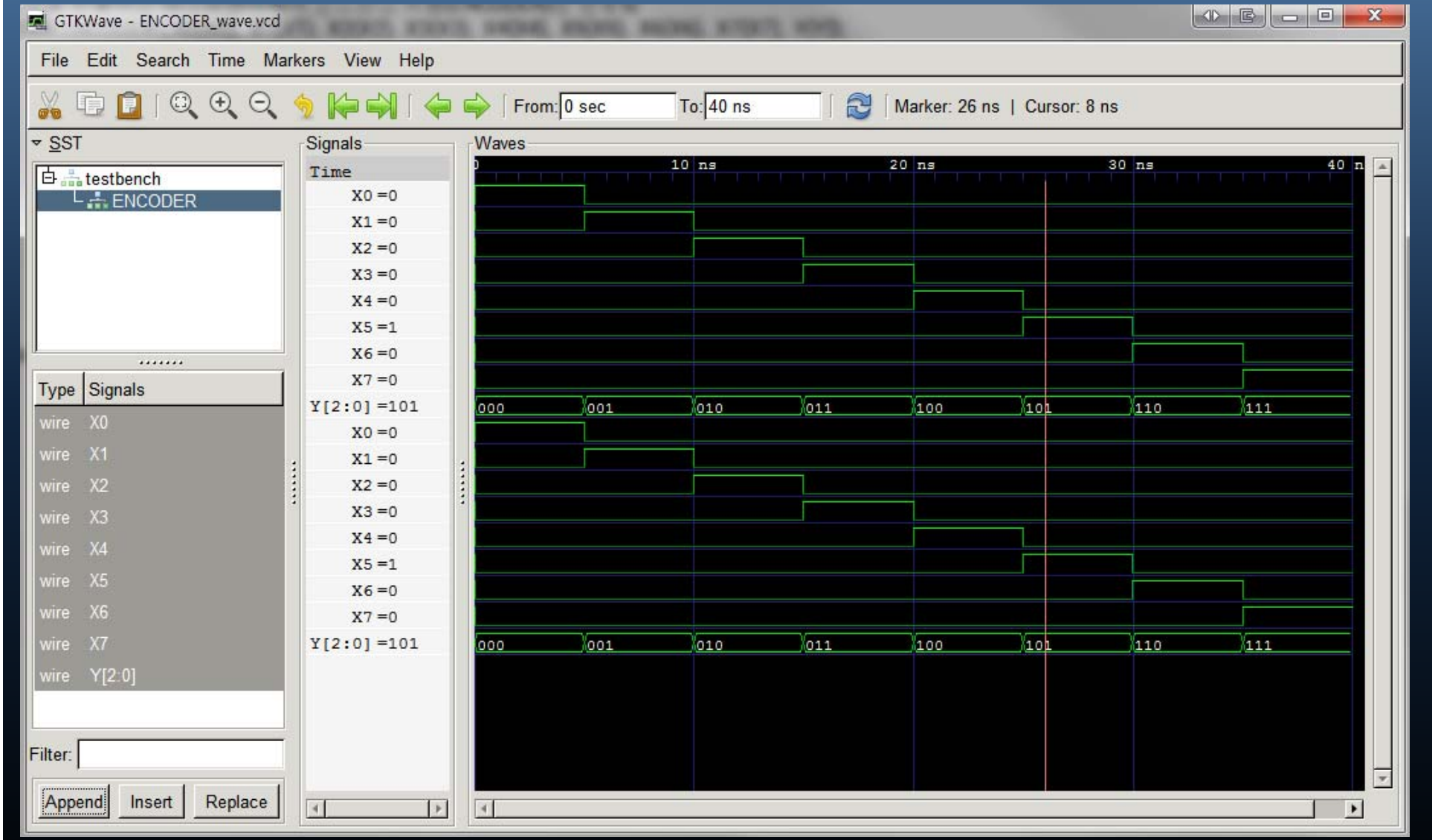
설계하기

```
`include "ENCODER.v"
`timescale 1ns/1ns
module testbench;
   reg X0, X1, X2, X3, X4, X5, X6, X7;
   wire [2:0] Y;
//자동으로 만들어진 테스트벤치에서 인스턴스 이름(ENCODER)만 변경함
ENCODER ENCODER (.X0(X0), .X1(X1), .X2(X2), .X3(X3), .X4(X4), .X5(X5), .X6(X6), .X7(X7), .Y(Y));
            initial begin            //입력 값을 5[ns] 동안 유지
                        $dumpfile("ENCODER_wave.vcd");
                        $dumpvars(0, testbench);
                        X0=1; X1=0; X2=0; X3=0; X4=0; X5=0; X6=0; X7=0; #5;
                        X0=0; X1=1; X2=0; X3=0; X4=0; X5=0; X6=0; X7=0; #5;
                        X0=0; X1=0; X2=1; X3=0; X4=0; X5=0; X6=0; X7=0; #5;
                        X0=0; X1=0; X2=0; X3=1; X4=0; X5=0; X6=0; X7=0; #5;
                        X0=0; X1=0; X2=0; X3=0; X4=1; X5=0; X6=0; X7=0; #5;
                        X0=0; X1=0; X2=0; X3=0; X4=0; X5=1; X6=0; X7=0; #5;
                        X0=0; X1=0; X2=0; X3=0; X4=0; X5=0; X6=1; X7=0; #5;
                        X0=0; X1=0; X2=0; X3=0; X4=0; X5=0; X6=0; X7=1; #5;
            end
endmodule
```

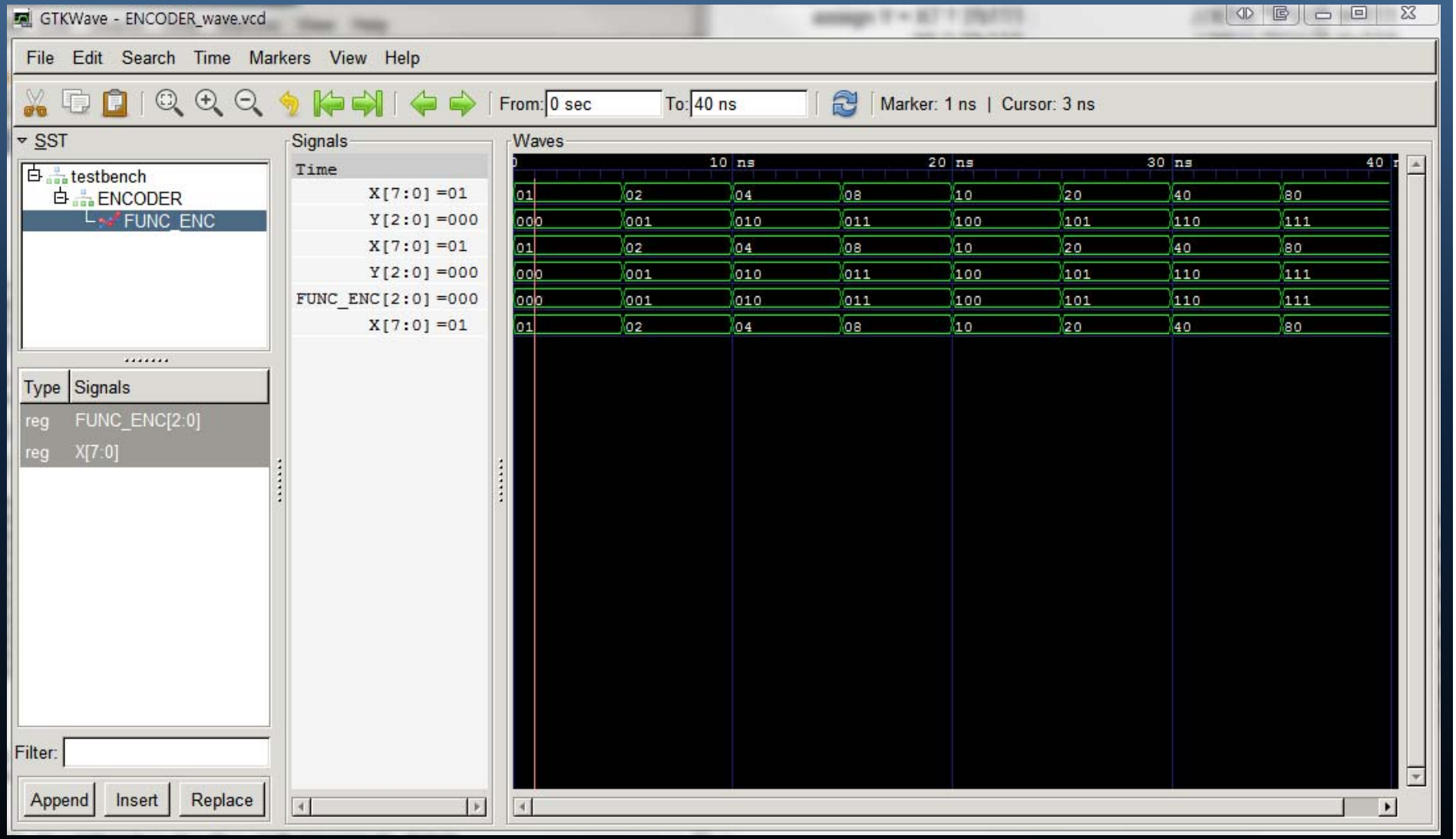
설계하기

```
`include "ENCODER.v"
`timescale 1ns/1ns
module testbench;
    reg [7:0] X;
    wire [2:0] Y;
//자동으로 만들어진 테스트벤치에서 인스턴스 이름(ENCODER)만 변경함
ENCODER ENCODER (.X(X), .Y(Y));
        initial begin   //입력 값을 5[ns] 동안 유지
                $dumpfile("ENCODER_wave.vcd");
                $dumpvars(0, testbench);
                X[0]=1; X[1]=0; X[2]=0; X[3]=0; X[4]=0; X[5]=0; X[6]=0; X[7]=0; #5;
                X[0]=0; X[1]=1; X[2]=0; X[3]=0; X[4]=0; X[5]=0; X[6]=0; X[7]=0; #5;
                X[0]=0; X[1]=0; X[2]=1; X[3]=0; X[4]=0; X[5]=0; X[6]=0; X[7]=0; #5;
                X[0]=0; X[1]=0; X[2]=0; X[3]=1; X[4]=0; X[5]=0; X[6]=0; X[7]=0; #5;
                X[0]=0; X[1]=0; X[2]=0; X[3]=0; X[4]=1; X[5]=0; X[6]=0; X[7]=0; #5;
                X[0]=0; X[1]=0; X[2]=0; X[3]=0; X[4]=0; X[5]=1; X[6]=0; X[7]=0; #5;
                X[0]=0; X[1]=0; X[2]=0; X[3]=0; X[4]=0; X[5]=0; X[6]=1; X[7]=0; #5;
                X[0]=0; X[1]=0; X[2]=0; X[3]=0; X[4]=0; X[5]=0; X[6]=0; X[7]=1; #5;
        end
endmodule
```

설계하기


GTKWave - ENCODER_wave.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 40 ns
Marker: 26 ns | Cursor: 8 ns
SST
testbench
ENCODER
Type Signals
wire X0
wire X1
wire X2
wire X3
wire X4
wire X5
wire X6
wire X7
wire Y[2:0]
Filter:
Append Insert Replace
Signals
Time
X0 =0
X1 =0
X2 =0
X3 =0
X4 =0
X5 =1
X6 =0
X7 =0
Y[2:0] =101
X0 =0
X1 =0
X2 =0
X3 =0
X4 =0
X5 =1
X6 =0
X7 =0
Y[2:0] =101
Waves
10 ns
20 ns
30 ns
40 n
000
001
010
011
100
101
110
111

설계하기


GTKWave - ENCODER_wave.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 40 ns
Marker: 1 ns | Cursor: 3 ns
SST
testbench
ENCODER
FUNC_ENC
Type Signals
reg FUNC_ENC[2:0]
reg X[7:0]
Filter:
Append Insert Replace
Signals
Time
X[7:0] =01
Y[2:0] =000
X[7:0] =01
Y[2:0] =000
FUNC_ENC[2:0] =000
X[7:0] =01
Waves
10 ns
20 ns
30 ns
40 ns
01 02 04 08 10 20 40 80
000 001 010 011 100 101 110 111
01 02 04 08 10 20 40 80
000 001 010 011 100 101 110 111
000 001 010 011 100 101 110 111
01 02 04 08 10 20 40 80

설계하기

- 패리티 발생기 및 검사기

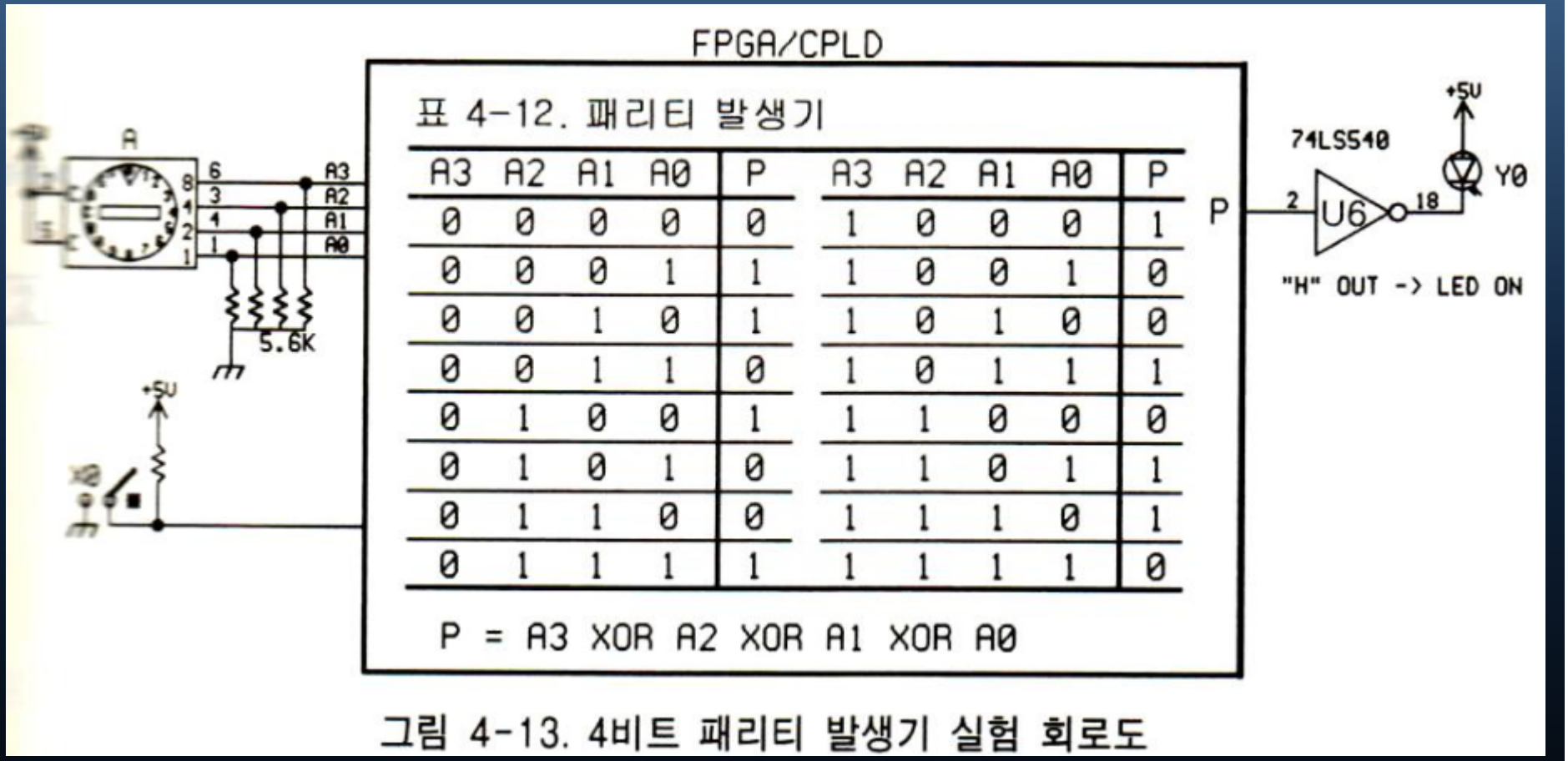


표 4-12. 패리티 발생기

A3	A2	A1	A0	P	A3	A2	A1	A0	P
0	0	0	0	0	1	0	0	0	1
0	0	0	1	1	1	0	0	1	0
0	0	1	0	1	1	0	1	0	0
0	0	1	1	0	1	0	1	1	1
0	1	0	0	1	1	1	0	0	0
0	1	0	1	0	1	1	0	1	1
0	1	1	0	0	1	1	1	0	1
0	1	1	1	1	1	1	1	1	0

P = A3 XOR A2 XOR A1 XOR A0

그림 4-13. 4비트 패리티 발생기 실험 회로도

설계하기

```
//비트 연산자 이용하기
module PARITY (A, P);
   input [3:0] A;          //입력 4비트
   output P;               //출력 1비트
            //A[3] xor A[2] xor A[1] xor A[0]
            assign P = A[3] ^ A[2] ^ A [1] ^ A[0];
endmodule
```

```
//단항 비트 처리(reduction) 연산자 이용하기
module PARITY (A, P);
   input [3:0] A;          //입력 4비트
   output P;               //출력 1비트
            //A3 xor A2 xor A1 xor A0
            assign P = ^ (A);
endmodule
```

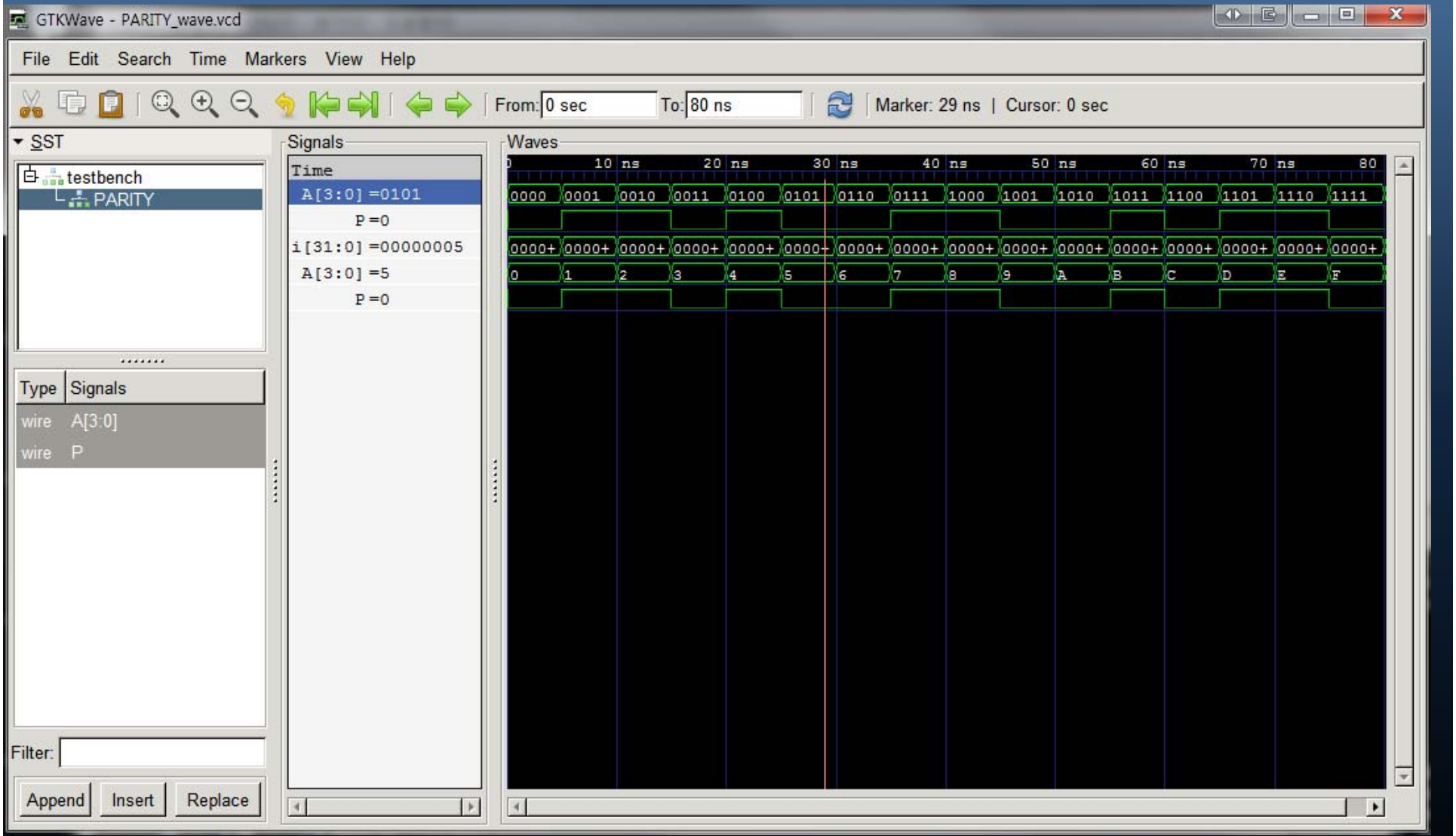
설계하기

```
//"for"문 이용하기
module PARITY (A, P);
   input [3:0] A;          //입력 4비트
   output P;               //출력 1비트
            assign P = FUNC_P (A);          //입력 A에 대해서 출력 P를 얻어서 출력
            //패리티 발생기 함수
            function FUNC_P;
               input [3:0] A;
               integer i;                   //루프 인덱스
                        begin               //블록문
                           FUNC_P = 0;      //초기값 0
                                    for (i=0; i<=3; i=i+1)
                                       FUNC_P = FUNC_P ^ A[i];     //A0 xor A1 xor A2 xor A3
                        end
            endfunction
endmodule
```

설계하기

```
`include "PARITY.v"
`timescale 1ns/1ns
module testbench;
    reg [3:0] A;                    //입력을 레지스터로 선언
    wire P;                         //출력을 와이어로 선언
    integer i;                      //루프 인덱스
//자동으로 만들어진 테스트 벤치에서 인스턴스 이름(PARITY)만 변경함
PARITY PARITY (.A(A), .P(P));
            initial begin
                        $dumpfile("PARITY_wave.vcd");
                        $dumpvars(0, testbench);
                        A=0;                    //초기 값
                for (i=0; i<=15; i=i+1)
                        #5 A = A + 1;           //5[ns]동안 유지한 후, A = A + 1
            end
endmodule
```

설계하기


GTKWave - PARITY_wave.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 80 ns
Marker: 29 ns | Cursor: 0 sec
SST
testbench
PARITY
Type Signals
wire A[3:0]
wire P
Filter:
Append Insert Replace
Signals
Time
A[3:0] =0101
P =0
i[31:0] =00000005
A[3:0] =5
P =0
Waves
10 ns 20 ns 30 ns 40 ns 50 ns 60 ns 70 ns 80
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
0000+ 0000+ 0000+ 0000+ 0000+ 0000+ 0000+ 0000+ 0000+ 0000+ 0000+ 0000+ 0000+ 0000+ 0000+ 0000+
0 1 2 3 4 5 6 7 8 9 A B C D E F