8-1-5 주사위 설계하기

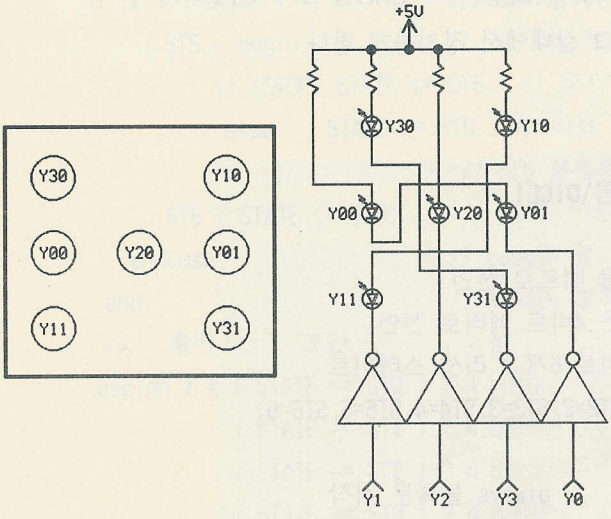
주사위(dice)도 스테이트 · 머신 설계의 가장 대표적인 예 중의 하나이며, 흔히 주사위는 흔히 우리 주의에서 많이 볼 수 있으며, 주사위를 던져서 주사위 눈의 개수를 이용하여 이기고/지는 일종의 게임이다.

"Verilog HDL로 설계하기" 부분은 책 뒤에 있는 CD에서 "Verilog 폴더"를 C 드라이브로 복사해서 사용하기 바랍니다.

1 주사위 로직과 회로

주사위를 그림 8-1-10(a)와 같이 LED를 배치해서 만들 수 있으며, 이때 LED 연결은 그림 8-1-10(b)와 같이 연결되어야 한다.

그림 8-1-10을 보면 알 수 있듯이 주사위를 동작시키는데 4선만 있으면 되며, 주사위의 숫자와 LED의 점등되는 관계는 표 8-1-2와 같다.



(a) 주사위 배치 (b) 내부 회로

그림 8-1-10. LED를 이용한 주사위 구성

표 8-1-2. 주사위의 순서 상태

주사위 상 태	Y3	Y2	Y1	Y0
	0	1	0	0
	0	0	1	0
	1	1	0	0
	1	0	1	0
	1	1	1	0
	1	0	1	1

2 설계 방법

그림 8-1-11과 같이 구성이 되어 있는 주사위 인터페이스 회로에서 **하위 부분**(그림 8-1-11의 오른쪽 부분)을 이용하여 주사위에 숫자가 나타나도록 설계해보자.

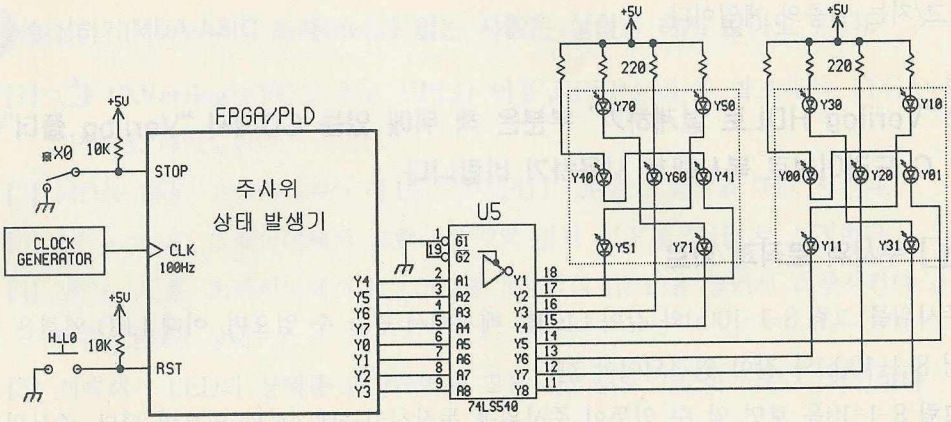


그림 8-1-11. 주사위 실험을 하기 위한 회로

(1) 클록은 100[Hz]를 사용하고. (2) 리셋일 때는 LED를 모두 OFF되게 한다.
(3) 동작 중에 STOP가 "1"이면, 그 상태에서 정지하게 한다.

A Verilog HDL로 설계하기

C\Verilog\FPGA 혹은 CPLD 이름\DICE1

```
module DICE(CLK, RST, STOP, Y);
  input CLK, RST, STOP;      // 입력을 비트로 선언
  output [3:0] Y;            // 출력은 4비트 벡터로 선언
  reg [2:0] STATE;           // 스테이트 6개 + 리셋 스테이트
    parameter [2:0] ST0=0, ST1=1, ST2=2, ST3=3, ST4=4, ST5=5, ST6=6;
  // 스테이트 머신
   always @(posedge CLK) begin      // always 블록문 시작
     if (!RST)  STATE <= ST6;       // 리셋 스테이트
     else                           // 클록에 동기되어 스테이트 이동
         case (STATE)
            ST0 : begin             // "1" 디스플레이 상태
                if (STOP) STATE <= ST0;  // 정지?
```

```
            else     STATE <= ST1;  // 다음 상태
          end                       // ST0 블록문 끝
        ST1 : begin                 // "2" 디스플레이 상태
          if (STOP) STATE <= ST1;  // 정지?
            else     STATE <= ST2;  // 다음 상태
          end                       // ST1 블록문 끝
        ST2 : begin                 // "3" 디스플레이 상태
          if (STOP) STATE <= ST2;  // 정지?
            else     STATE <= ST3;  // 다음 상태
          end                       // ST2 블록문 끝
        ST3 : begin                 // "4" 디스플레이 상태
          if (STOP) STATE <= ST3;  // 정지?
            else     STATE <= ST4;  // 다음 상태
          end                       // ST3 블록문 끝
        ST4 : begin                 // "5" 디스플레이 상태
          if (STOP) STATE <= ST4;  // 정지?
            else     STATE <= ST5;  // 다음 상태
          end                       // ST4 블록문 끝
        ST5 : begin                 // "6" 디스플레이 상태
          if (STOP) STATE <= ST5;  // 정지?
            else     STATE <= ST0;  // 다음 상태
          end                       // ST5 블록문 끝
        ST6 : STATE <= ST0;         // 리셋 스테이트
      endcase                       // case문 끝
    end                             // always 블록문 끝
    //   출력        조건            참
    assign Y = ( STATE == ST0 ) ? 4'b0100 : // 1
               ( STATE == ST1 ) ? 4'b0010 : // 2
               ( STATE == ST2 ) ? 4'b1100 : // 3
               ( STATE == ST3 ) ? 4'b1010 : // 4
               ( STATE == ST4 ) ? 4'b1110 : // 5
               ( STATE == ST5 ) ? 4'b1011 : // 6
                                  4'b0000;  // 리셋
endmodule   // 모듈 끝
```

B 시뮬레이션하기

1 테스트벤치를 만들어서 다음과 같이 수정한다(HELP.PDF 참조).

```
`timescale 1ns/1ns          // 기본 스텝 시간
module testbench;           // 모듈 이름
    reg  RST,CLK,STOP;      // 입력을 레지스터로 선언
    wire [3:0] Y;           // 출력은 와이어로 선언
    parameter STEP = 120;   // 클록 주기
// 테스트벤치에서 자동으로 만들어진 것에서 인스턴스 이름(DICE)만 변경한다.
DICE DICE (.CLK(CLK),.RST(RST),.STOP(STOP),.Y(Y));
    always #(STEP/2)  CLK = ~CLK;  // 클록 발진
   // 입력 값을 설정한다.
    initial begin                  // 블록문 시작
            CLK=1; RST=0; STOP = 0; #(STEP-10);   // 리셋
            RST=1; #(STEP*8);          // STEP×8 기간동안 동작한다.
            STOP=1; #STEP;   $stop;   // 현재 상태 정지
    end                            // 블록문 끝
endmodule                          // 모듈 끝
```

2 시뮬레이션하기(HELP.PDF 참조).

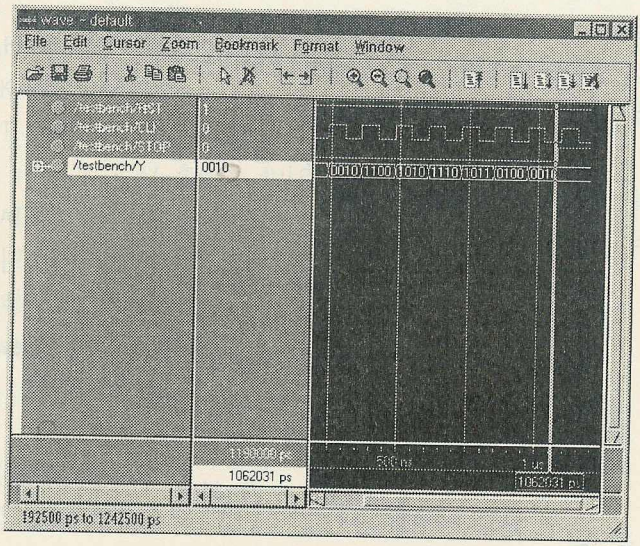


그림 8-1-12. 주사위를 시뮬레이션시킨 결과

(1) ModelSim을 이용하여 시뮬레이션시키면 그림 8-1-12와 같다.

(2) 커서를 움직이면서 주사위의 기능을 하는지 확인해보자.

C 실험하기(MDA-ASIC 트레이너가 없는 사람은 실험을 하지 않아도 된다).

【1】 C\Verilog\FPGA 혹은 CPLD 이름\HELP.PDF를 참조해서, FPGA 혹은 CPLD로 라이트한다.

【2】 MDA-ASIC 트레이너에서 "DICE OUTPUT" 토글 스위치를 "ON"시킨다.

【3】 MDA-ASIC 트레이너에서 그림 8-1-11의 입력 CLK를 "100Hz"로 설정한다.

【4】 MDA-ASIC 트레이너에서 H_L0(그림 8-1-11의 RST)을 눌러서 리셋시킨다. 처음 주사위의 상태는?

【5】 주사위가 동작 중에 MDA-ASIC 트레이너에서 X0(그림 8-1-11의 STOP)을 "H"로 하면, 그때의 주사위 상태는? 계속 반복하면서, 주사위의 기능을 하는지 확인해보자.

D 연습문제

그림 8-1-11을 참조해서 주사위를 2개로 늘려서 설계하고, 시뮬레이션해서 기능을 확인한 다음, C 실험하기 에서와 같은 방법으로 기능을 확인해보자.

☞ **정지하였을 때 상위, 하위 값(그림 8-1-11의 왼쪽, 오른쪽)이 다르게 되어야 한다.**

8-3 동적인 디스플레이

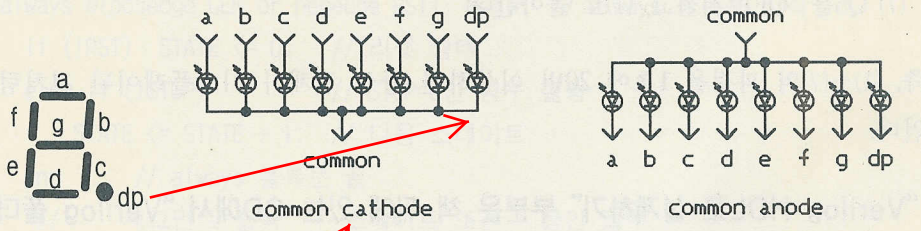
디스플레이 장치로 많이 사용되고 있는 7세그먼트(segment)를 여러 개 사용해서 임의의 숫자를 동적(dynamic)인 방법으로 디스플레이 하도록 설계해 보자.

8-3-1 동적으로 디스플레이하기

먼저 7세그먼트 동작 방법은 이미 앞장에서 설명하였으므로, 이 장에서는 7세그먼트를 여러 개 사용해서 동적(dynamic)으로 디스플레이하는 방법에 대해서 설명하기로 한다.

1 7세그먼트의 내부 구조

이미 앞장에서 설명하였지만 7세그먼트는 그림 8-3-1과 같이 CA(Common Anode), CC(Common Cathode)형이 있다.



(a) 7세그먼트 외형 (b) 7세그먼트를 펼친 그림

그림 8-3-1. 7세그먼트의 형태와 내부 회로

2 동적인 디스플레이 방법

그림 8-3-2와 같이 CC형 7세그먼트 6개를 이용하여 디스플레이하는 방법에 대해서 설명하기로 한다.

(1) Q0 디지트에 디스플레이하고자 하는 값을 주고, Q0을 일정시간 "ON"시킨다.

(2) Q0을 "OFF"시키고, Q1 디지트에 디스플레이하고자 하는 값을 주고, Q1을 일정시간 "ON"시킨다.

(3) Q1을 "OFF"시키고, Q2의 디지트에 디스플레이하고자 하는 값을 주고, Q2를 일정시간 "ON"시킨다.

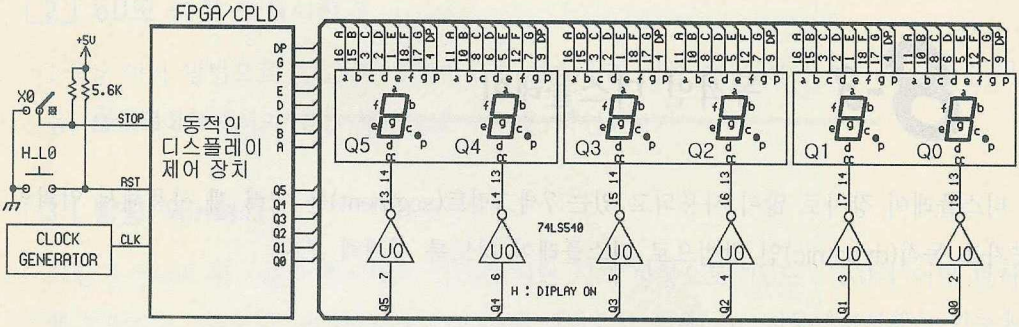


그림 8-3-2. 6개의 7세그먼트와 실험 회로도

(4) Q2를 "OFF"시키고, Q3 디지트에 디스플레이하고자 하는 값을 주고, Q3을 일정시간 "ON"시킨다.

(5) Q3을 "OFF"시키고, Q4 디지트에 디스플레이하고자 하는 값을 주고, Q4를 일정시간 "ON"시킨다.

(6) Q4를 "OFF"시키고, Q5 디지트에 디스플레이하고자 하는 값을 주고, Q5를 일정시간 "ON"시킨다.

(7) Q5를 "OFF"시키고, (1)로 돌아간다.

즉, (1)~(7)의 과정을 1초에 20번 이상하면 항상 전체가 디스플레이된 것처럼 보인다.

"Verilog HDL로 설계하기" 부분은 책 뒤에 있는 CD에서 "Verilog 폴더"를 C 드라이브로 복사해서 사용하기 바랍니다.

8-3-2 세그먼트 디스플레이

그림 8-3-2를 참조로 해서 7세그먼트 6개를 동적인(dynamic) 방법으로 그림 8-3-3과 같이 디스플레이 해보자. 기준 클록은 10[Hz]를 사용하며, 그림 8-3-2에서 "STOP=1"이면 정지되도록 설계해보자.

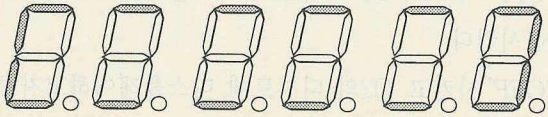


그림 8-3-3. 세그먼트 디스플레이 형태

A Verilog HDL로 설계하기

C\Verilog\FPGA 혹은 CPLD 이름\DISPLAY1

```
// 세그먼트 정의
`define  SEGMENT_a   7'b100_0000   // a 세그먼트
`define  SEGMENT_b   7'b010_0000   // b 세그먼트
`define  SEGMENT_c   7'b001_0000   // c 세그먼트
`define  SEGMENT_d   7'b000_1000   // d 세그먼트
`define  SEGMENT_e   7'b000_0100   // e 세그먼트
`define  SEGMENT_f   7'b000_0010   // f 세그먼트
// 세그먼트 모듈
module SEGMENT(RST,CLK,STOP,A,B,C,D,E,F,G,Q);
    input RST,CLK,STOP;
    output A,B,C,D,E,F,G;          // 세그먼트
    output [5:0] Q;                // 디지트 선택
    reg [3:0] STATE;               // 총 스테이트
    // 스테이트 머신
     always @(posedge CLK or negedge RST)  begin  // always 블록문 시작
       if (!RST)  STATE <= 0;    // 리셋 상태
       else if (!STOP)           // STOP=0인 경우 실행
           STATE <= STATE + 1;  // 다음 스테이트
       end      // always 블록문 끝
        //     세그먼트 출력      스테이트 조건   참인 경우
       assign {A,B,C,D,E,F,G} = (STATE == 0)  ? `SEGMENT_a :  // a 세그먼트
                                (STATE == 1)  ? `SEGMENT_a :
                                (STATE == 2)  ? `SEGMENT_a :
                                (STATE == 3)  ? `SEGMENT_a :
                                (STATE == 4)  ? `SEGMENT_a :
                                (STATE == 5)  ? `SEGMENT_a :
                                (STATE == 6)  ? `SEGMENT_f :  // f 세그먼트
                                (STATE == 7)  ? `SEGMENT_e :  // e 세그먼트
                                (STATE == 8)  ? `SEGMENT_d :  // d 세그먼트
                                (STATE == 9)  ? `SEGMENT_d :
                                (STATE == 10) ? `SEGMENT_d :
                                (STATE == 11) ? `SEGMENT_d :
```

각 segment turn on

각 7 segment turn on

Segment turn on or off bus

```
                        (STATE == 12) ? `SEGMENT_d :
                        (STATE == 13) ? `SEGMENT_d :
                        (STATE == 14) ? `SEGMENT_c :  // c 세그먼트
                                        `SEGMENT_b ;  // b 세그먼트

  // 디지트 출력,    조건          참인 경우
    assign {Q} = (STATE == 0)  ? 6'b000001 :  // Q0 디지트
                 (STATE == 1)  ? 6'b000010 :  // Q1 디지트
                 (STATE == 2)  ? 6'b000100 :  // Q2 디지트
                 (STATE == 3)  ? 6'b001000 :  // Q3 디지트
                 (STATE == 4)  ? 6'b010000 :  // Q4 디지트
                 (STATE == 5)  ? 6'b100000 :  // Q5 디지트
                 (STATE == 6)  ? 6'b100000 :
                 (STATE == 7)  ? 6'b100000 :
                 (STATE == 8)  ? 6'b100000 :
                 (STATE == 9)  ? 6'b010000 :  // Q4 디지트
                 (STATE == 10) ? 6'b001000 :  // Q3 디지트
                 (STATE == 11) ? 6'b000100 :  // Q2 디지트
                 (STATE == 12) ? 6'b000010 :  // Q1 디지트
                 (STATE == 13) ? 6'b000001 :  // Q0 디지트
                 (STATE == 14) ? 6'b000001 :
                                 6'b000001 ;
endmodule        // 모듈 끝
```

7 segment 갔다가 다시 오는것, (각 segment enable 개념)

B 시뮬레이션하기

1 테스트벤치를 만들어서 다음과 같이 수정한다(HELP.PDF 참조).

```
`timescale 1ns/1ns        // 기본 스텝 시간
 module testbench;        // 모듈 이름
  reg  RST,CLK,STOP;      // 입력을 레지스터로 선언
  wire A,B,C,D,E,F;       // 세그먼트 출력을 와이어로 선언
  wire [5:0] Q;           // 세그먼트 디지트를 와이어로 선언
  parameter STEP = 120; 클록 주기
// 테스트벤치에서 자동으로 만들어진 것에서 인스턴스 이름(SEGMENT)만 변경한다.
SEGMENT SEGMENT (.RST(RST),.CLK(CLK),.STOP(STOP),
```

```
    .A(A),.B(B),.C(C),.D(D),.E(E),.F(F),.G(G),.Q(Q));
    always #(STEP/2)  CLK = ~CLK;  // 클록 발진
    initial begin                  // 입력 값 설정
       CLK=1; RST=0; STOP = 0; #(STEP-10);  // 리셋
       RST=1; #(STEP*8);                    // STEP×8동안 동작
       STOP=1; #STEP; $stop;                // 정지
     end                    // 블록문 끝
endmodule                   // 테스트벤치 모듈 끝
```

2 시뮬레이션하기(HELP.PDF 참조).

(1) ModelSim을 이용하여 시뮬레이션시키면 그림 8-3-4와 같다.

(2) 커서를 움직이면서 세그먼트가 디스플레이 되는지 확인해보자.

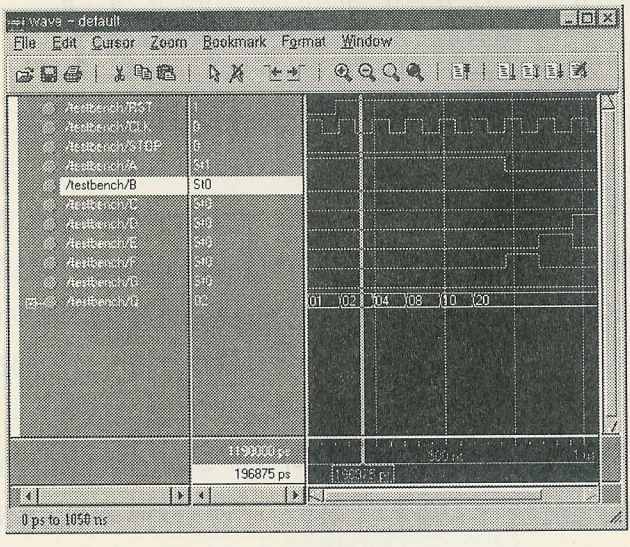


그림 8-3-4. 동적인 디스플레이를 시뮬레이션시킨 결과

C 실험하기(MDA-ASIC 트레이너가 없는 사람은 실험을 하지 않아도 된다).

【1】 C\Verilog\FPGA 혹은 CPLD 이름\HELP.PDF를 참조해서, FPGA 혹은 CPLD로 라이트한다.

【2】 MDA-ASIC 트레이너에서 "DYNAMIC DISPLAY" 토글 스위치를 "ON"시켜서 출력을 인에이블시킨다.

【3】 MDA-ASIC 트레이너에서 그림 8-3-2의 입력 CLK를 "10Hz"로 설정한다.

【4】 MDA-ASIC 트레이너에서 H_L0(그림 8-3-2의 RST)을 눌러서 리셋시킨다. 그림 8-3-2에서 처음 7세그먼트의 상태는?

【5】 7세그먼트가 디스플레이 중에, MDA-ASIC 트레이너에서 X0="H"(그림 8-3-2의 STOP)로 하였을 경우, 7세그먼트의 상태는? 계속 반복해보자.

D 연습문제

1 "8"을 디스플레이하기

그림 8-3-5와 같이 "8"이 디스플레이 되면서 왼쪽으로 이동되도록 설계하고, 시뮬레이션해서 기능을 확인한 다음, C 실험하기 에서와 같은 방법으로 기능을 확인해보자.

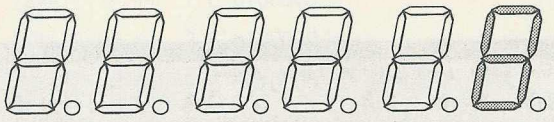


그림 8-3-5. "8" 디스플레이 형태

2 각 세그먼트를 디스플레이하기

각 디지트의 세그먼트를 "a→b→c→d→e→f→g→dp"의 순서로 디스플레이 되도록 설계하고, 시뮬레이션해서 기능을 확인한 다음, C 실험하기 에서와 같은 방법으로 기능을 확인해보자.

3 임의의 글자를 만들어서 디스플레이하기

사용자가 임의의 글자(표 4-6 참조)를 만들어서 각 디지트에 디스플레이 되도록 설계하고, 시뮬레이션해서 기능을 확인한 다음, C 실험하기 에서와 같은 방법으로 기능을 확인해보자.

8-3-3 숫자 디스플레이 하기

그림 8-3-6을 이용하여 전자 계산기의 디스플레이처럼 입력된 숫자를 최하위 디지트부터 디스플레이 한 다음, 다음 숫자가 입력되면 처음 숫자는 왼쪽으로 이동되어 디스플레이되도록 설계해보자. 클록 주파수는 1[kHz]를 사용하도록 하고, 계층 설계로 설계한다.

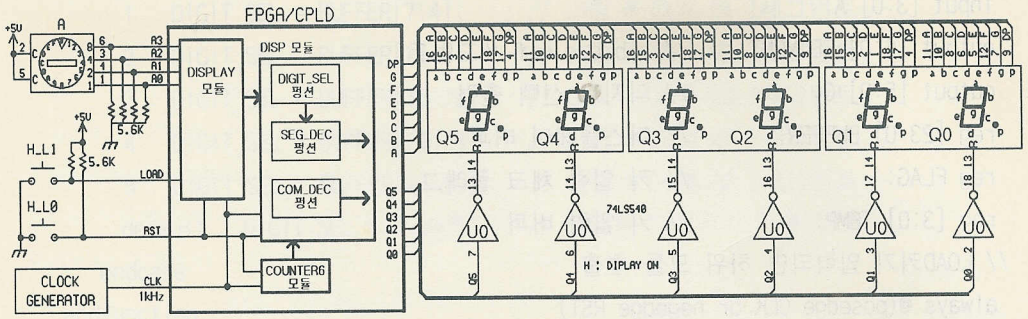


그림 8-3-6. 숫자 디스플레이 실험 회로도

☞ "FPGA/CPLD" 내부 블록도 설명

【1】 **클록** : 클록은 1[kHz]를 사용한다. 각 디지트의 점등 시간은 1[kHz]÷6≒ 167[Hz]이므로 동시에 6개가 점등된 것처럼 보인다.

【2】 **DISPLAY 모듈** : LOAD="L"일 때 A를 입력해서, 디스플레이 버퍼에 저장하고, 4비트 왼쪽으로 이동한 후, 하위 모듈 "DISP"로 "**디스플레이 버퍼**"를 넘긴다.

【3】 **COUNTER6 모듈** : 0~5까지 세며, 각 카운터의 값이 그림 8-3-6의 디지트 선택 값이 되고, 이 "**카운터 값**"을 "DISP"로 넘긴다.

【4】 **DISP 모듈** : DISPLAY 모듈에서 "**디스플레이 버퍼 값**" 및 DISP 모듈에서 "**카운터 값**"을 넘겨받아서, 6개의 7세그먼트를 동적으로 디스플레이하며, 다음과 같이 3개의 펑션과 출력 부분으로 나누어져 있다.

(1) DIGIT_SEL 펑션 : COUNTER6 모듈의 "카운터 값"을 이용하여 "디스플레이 버퍼"에서 7세그먼트에 점등할 데이터를 얻는다.

(2) SEG_DEC 펑션 : DIGIT_SEL 펑션의 값을 이용하여 7세그먼트 점등 포맷으로 변경한다.

(3) COM_DEC 펑션 : COUNTER6 모듈의 "카운터 값"을 이용하여 점등할 7세그먼트를 선택한다.

(4) 펑션을 콜해서 7세그먼트로 디스플레이할 데이터와 디지트를 출력한다.

A Verilog HDL로 설계하기

C\Verilog\FPGA 혹은 CPLD 이름\DISPLAY2

```
// 상위 모듈 정의
module DISPLAY(RST,CLK,LOAD,A,SEGMENT,Q);
  input RST,CLK,LOAD;
  input [3:0] A;
  output [6:0] SEGMENT;     // a,b,c,d,e,f 세그먼트
  output [5:0] Q;           // 디지트 선택 출력
  reg [23:0] BUFFER;        // 디스플레이 버퍼
  reg FLAG;                 // 키 입력 체크 플래그
  reg [3:0] TEMP;           // 키 입력 버퍼
 // LOAD키가 입력되면 하위 모듈 호출
  always @(posedge CLK or negedge RST)
   begin              // always 블록문 시작
      if (!RST) begin                 // 리셋
          BUFFER <= 0; FLAG <= 0;  // 버퍼 및 플래그 클리어
        end             // if 블록문 끝
      else if (!LOAD) begin      // 키 입력?
          TEMP <= A; FLAG <= 1;  // LOAD=0이면 A 값 입력, FLAG= "1"
        end             // else if 블록문 끝
     else if (FLAG) begin   // FLAG= "1"인 경우 BUFFER는 A와 함께 4비트 이동
          FLAG <= 0;  BUFFER <= {BUFFER[19:0],TEMP};     // FLAG 클리어
        end             // else if 블록문 끝
   end             // always 블록문 끝
//  디스플레이 하위 모듈 콜
DISP DISP (CLK, RST, BUFFER, SEGMENT,Q);
endmodule              // DISPLAY 모듈 끝
// 동적인 디스플레이 하위 모듈
module DISP (CLK,RST, BUFFER, SEGMENT, Q);
  input CLK,RST;
  input [23:0] BUFFER;      // 디스플레이 버퍼
  output [6:0] SEGMENT;     // 세그먼트 출력
  output [5:0] Q;           // 디지트 출력
  wire [2:0] COMCNT;        // COUNTER6의 출력
```

```
/* COUNTER6 모듈에서 COMCNT값을 받아서 출력할 디스플레이 버퍼 선택 */
function [3:0] DIGIT_SEL;
  input [2:0] COMCNT;       // COUNTER6에서 카운터 값 입력
  input [23:0] BUFFER;      // DISPLAY에서 디스플레이 버퍼 입력
   case (COMCNT)     //digit
      0 : DIGIT_SEL = BUFFER[3:0];      // 그림 8-28의 Q0 7세그먼트
      1 : DIGIT_SEL = BUFFER[7:4];      // 그림 8-28의 Q1 7세그먼트
      2 : DIGIT_SEL = BUFFER[11:8];     // 그림 8-28의 Q2 7세그먼트
      3 : DIGIT_SEL = BUFFER[15:12];    // 그림 8-28의 Q3 7세그먼트
      4 : DIGIT_SEL = BUFFER[19:16];    // 그림 8-28의 Q4 7세그먼트
      5 : DIGIT_SEL = BUFFER[23:20];    // 그림 8-28의 Q5 7세그먼트
      default : DIGIT_SEL = 4'b0000;    // 선택된 디지트가 없을 경우
    endcase
 endfunction
/* DIGIT_SEL 값을 0~F 까지의 7세그먼트 디스플레이 포맷으로 변환*/
function [6:0] SEG_DEC;
  input [3:0] DIGIT;      // 4비트 입력
   case (DIGIT)           //gfe dcba
      4'h0 : SEG_DEC = 7'b011_1111;    // "0" 디스플레이 형태
      4'h1 : SEG_DEC = 7'b000_0110;    // "1" 디스플레이 형태
      4'h2 : SEG_DEC = 7'b101_1011;    // "2" 디스플레이 형태
      4'h3 : SEG_DEC = 7'b100_1111;    // "3" 디스플레이 형태
      4'h4 : SEG_DEC = 7'b110_0110;    // "4" 디스플레이 형태
      4'h5 : SEG_DEC = 7'b110_1101;    // "5" 디스플레이 형태
      4'h6 : SEG_DEC = 7'b111_1101;    // "6" 디스플레이 형태
      4'h7 : SEG_DEC = 7'b000_0111;    // "7" 디스플레이 형태
      4'h8 : SEG_DEC = 7'b111_1111;    // "8" 디스플레이 형태
      4'h9 : SEG_DEC = 7'b110_1111;    // "9" 디스플레이 형태
      4'ha : SEG_DEC = 7'b111_0111;    // "A" 디스플레이 형태
      4'hb : SEG_DEC = 7'b111_1100;    // "B" 디스플레이 형태
      4'hc : SEG_DEC = 7'b011_1001;    // "C" 디스플레이 형태
      4'hd : SEG_DEC = 7'b101_1110;    // "D" 디스플레이 형태
      4'he : SEG_DEC = 7'b111_1001;    // "E" 디스플레이 형태
      4'hf : SEG_DEC = 7'b111_0001;    // "F" 디스플레이 형태
     endcase  // case문 끝
```

```
  endfunction      // 펑션문 끝
 /* COUNTER6 모듈에서 COMCNT값을 받아서 "ON"시킬 디지트 선택 */
  function [5:0] COM_DEC;
    input [2:0] COMCNT;  // 카운터 값 입력
     case (COMCNT)       // 점등할 7세그먼트
        0 : COM_DEC = 6'b000001;  // Q0 세그먼트 ON
        1 : COM_DEC = 6'b000010;  // Q1 세그먼트 ON
        2 : COM_DEC = 6'b000100;  // Q2 세그먼트 ON
        3 : COM_DEC = 6'b001000;  // Q3 세그먼트 ON
        4 : COM_DEC = 6'b010000;  // Q4 세그먼트 ON
        5 : COM_DEC = 6'b100000;  // Q5 세그먼트 ON
        default : COM_DEC = 6'b111111;  // 모두 OFF
     endcase          // case문 끝
   endfunction        // 펑션문 끝
   // 6개의 7세그먼트중 점등할 7세그먼트를 얻기 위하여 COUNTER6 콜
  COUNTER6 COM (CLK,RST,COMCNT); // COMCNT값이 카운터 출력(디스플레이할 7세그먼트)
   // SEG_DEC, DIGIT_SEL 펑션을 콜해서 SEMENT로 디스플레이할 데이터 출력
   assign SEGMENT = SEG_DEC(DIGIT_SEL(COMCNT,BUFFER));
   assign Q = COM_DEC(COMCNT);// COM_DEC 펑션을 콜해서 디스플레이할 디지트 출력
endmodule       // DISP 모듈 끝
 // 동적으로 디스플레이 하기 위한 카운트6
module COUNTER6 (CLK,RST,OUT);
   input RST,CLK;
   output [2:0] OUT;          // 카운터 출력
   reg [2:0] OUT;             // 카운터 출력을 레지스터로 선언
   wire OUT1;                 // 카운터 오버플로 플래그
   always @(posedge CLK)  begin  // always 블록문 시작
     if (!RST) OUT <= 0;         // 리셋
      else   // 클록에 동기되어서 동작한다.
        if ( OUT1)  OUT <= 0;    // 카운터가 오버플로면 카운터 클리어
         else   OUT <= OUT + 1;  // 카운터 증가
       end   // always 블록문 끝
        // 카운트 끝 오버플로 플래그 값 설정
       assign OUT1 = (OUT==5);   // OUT=5이면 오버플로 플래그 OUT1=1
  endmodule       // COUNTER6 모듈 끝
```

B 시뮬레이션하기

1 테스트 벤치를 만들어서 다음과 같이 수정한다(HELP.PDF 참조).

```
`timescale 1ns/1ns          // 기본 스텝 시간
module testbench;           // 모듈 이름
  reg  RST,CLK,LOAD;        // 입력을 레지스터로 선언
  reg [3:0] A;              // 입력 A를 레지스터로 선언
  wire [6:0] SEGMENT;       // 세그먼트 출력을 와이어로 선언
  wire [5:0] Q;             // 디지트 출력을 와이어로 선언
  parameter STEP = 120; // 클록 주기
// 테스트벤치에서 자동으로 만들어진 것에서 인스턴스 이름(DISPLAY)만 변경한다.
DISPLAY DISPLAY (.RST(RST),.CLK(CLK),.LOAD(LOAD),
                 .A(A),.SEGMENT(SEGMENT),.Q(Q));
  always #(STEP/2)  CLK = ~CLK;   // 클록 발진
  initial begin                   // 입력 값 설정
     CLK=1; RST=0; LOAD = 1; A= 0;#(STEP-10);    // 리셋
     RST=1; LOAD = 0; A=3;#STEP;                 // 3 로드
     LOAD=1; #STEP;
     LOAD=0; A=4; #STEP;                         // 4 로드
     LOAD=1; #(STEP*2);
     LOAD=0; A=5; #STEP;                         // 5 로드
     LOAD=1; #(STEP*2);
     LOAD=0; A=6; #STEP;                         // 6 로드
     LOAD=1; #(STEP*2);
     LOAD=0; A=7; #STEP;                         // 7 로드
     LOAD=1; #(STEP*2);
     LOAD=0; A=8; #STEP;                         // 8 로드
     LOAD=1; #(STEP*8);   $stop;
   end                     // 블록문 끝
endmodule                  // 테스트벤치 모듈 끝
```

2 시뮬레이션하기(HELP.PDF 참조).

(1) ModelSim을 이용하여 시뮬레이션시키면 그림 8-3-7과 같다.

(2) 커서를 움직이면서 기능을 확인해보자.

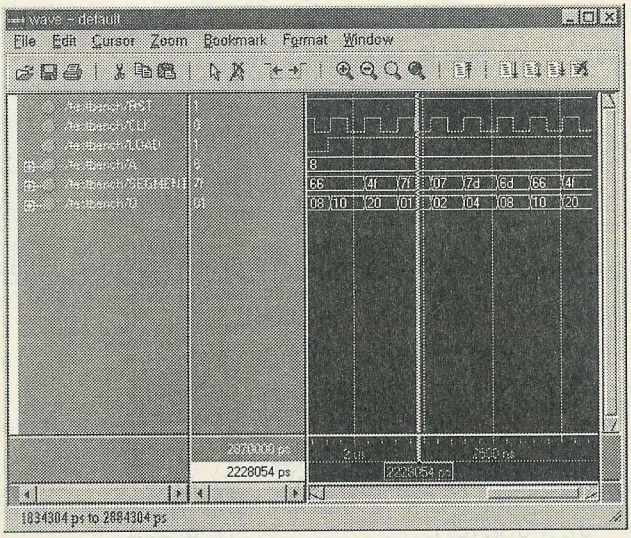


그림 8-3-7. 숫자 입력을 시뮬레이션시킨 결과

C 실험하기(MDA-ASIC 트레이너가 없는 사람은 실험을 하지 않아도 된다).

【1】 C\Verilog\FPGA 혹은 CPLD 이름\HELP.PDF를 참조해서, FPGA 혹은 CPLD로 라이트한다.

【2】 MDA-ASIC 트레이너에서 "DYNAMIC DISPLAY" 토글 스위치를 "ON"시켜서 출력을 인에이블시킨다.

【3】 MDA-ASIC 트레이너에서 그림 8-3-6의 입력 CLK를 "1kHz"로 설정한다.

【4】 MDA-ASIC 트레이너에서 H_L0(그림 8-3-6의 RST)을 눌러서 리셋시킨다. 그림 8-3-6에서 처음 7세그먼트의 상태는?

【5】 MDA-ASIC 트레이너에서 A값을 설정한 후, H_L1(그림 8-3-6의 LOAD) 키를 누르면, 7세그먼트의 상태는? 계속 반복해서 A값이 왼쪽으로 이동되면서 디스플레이 되는지 확인해보자.

D 연습문제

위의 실험과는 반대로 입력된 값이 최상위 디지트부터 디스플레이한 다음, 다음 숫자가 입력되면 처음 숫자는 오른쪽으로 이동 되도록 다시 설계하고, 시뮬레이션해서 기능을 확인한 다음, **C 실험하기**에서와 같은 방법으로 기능을 확인해보자.

8-3-4 시계 만들기

앞장에서 실험한 동적인 점등 방법을 이용하여, 그림 8-3-8과 같이 디스플레이되는 시계를 그림 8-3-9를 이용하여 계층 설계 방법으로 설계해보자.

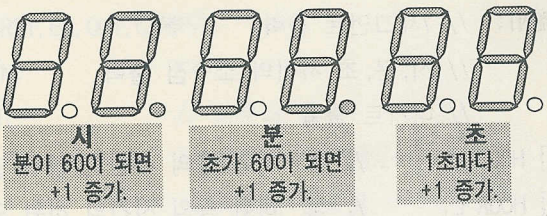


그림 8-3-8. 시계 디스플레이 형태

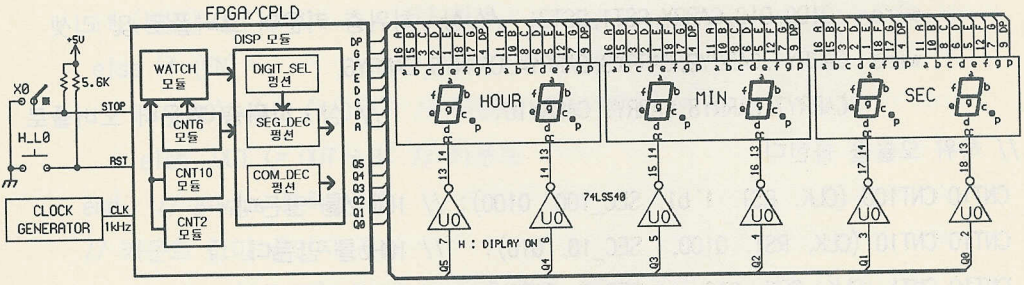


그림 8-3-9. 시계 실험 회로도

☞ "FPGA/CPLD" 내부 블록도 설명

【1】 클록 : 클록은 1[kHz]를 사용한다. 각 디지트의 점등 시간은 1[kHz]÷6≒ 167[Hz]이므로 동시에 6개가 점등된 것처럼 보인다. 그리고 1[kHz]÷1000≒ 1[Hz]를 이용하여 시계를 동작시키는 기준 클록을 만든다.

【2】 WATCH 모듈 : 하위 모듈(CNT10, CNT6, CNT2, DISP)들을 콜해서 시, 분, 초를 만들고, "DISP"를 이용하여 7세그먼트로 디스플레이한다.

【3】 CNT10 모듈 : 1[kHz]를 10, 100, 100 등으로 나누는데 사용한다.

【4】 CNT6 모듈 : 6개의 7세그먼트를 동적으로 디스플레이, 또는 분, 초의 상위 값을 만드는데 사용한다.

【5】 CNT2 모듈 : 시간의 상위 디지트를 만드는데 사용한다.

【6】 DISP 모듈 : 6개의 7세그먼트를 동적으로 디스플레이한다(앞장 참조).

A Verilog HDL로 설계하기

C\Verilog\FPGA 혹은 CPLD 이름\CLOCK

```
module WATCH(CLK, RST, SEGMENT, POINT, Q);
   input CLK, RST;
   output [6:0] SEGMENT;  // 7세그먼트 출력
   output POINT;          // 시, 분, 초 사이의 소수점 출력
   output [5:0] Q;        // 디지트 출력
        wire   [1:0] HOUR_H;      // 상·하위 층의 시간의 상위 연결
        wire   [3:0] HOUR_L;      // 상·하위 층의 시간의 하위 연결
        wire   [2:0] MIN_H, SEC_H;  // 상·하위 층의 분, 초의 상위 연결
        wire   [3:0] MIN_L, SEC_L, SEC_100, SEC_10, SEC_1; // 상·하위 층의 연결
        wire   Q100, Q10, CARRY, RST1, RST2;  // 상·하위층 카운터 오버플로 및 리셋
        wire  CARRY1, CARRY2, CARRY3, CARRY4, CARRY5, CARRY6,
              CARRY7, CARRY8, CARRY9, CARRY10;          // 상·하위층 카운터 오버플로
 // 하위 모듈을 콜한다
  CNT10 CNT100 (CLK, RST, 1'b1, SEC_100, Q100); // 100Hz를 만든다.
  CNT10 CNT10 (CLK, RST, Q100,  SEC_10, Q10);   // 10Hz를 만든다.
  CNT10 CNT1 (CLK, RST, Q10,    SEC_1, CARRY);  // 1Hz를 만든다.
  // 시계 만들기
  CNT10 CNT_S_LOW  (CLK, RST, CARRY,   SEC_L, CARRY1);  // 초 하위
  CNT6  CNT_S_HIGH (CLK, RST, CARRY1,  SEC_H, CARRY2);  // 초 상위
   assign CARRY3 = CARRY1 & CARRY2;          // 초가 "59"면 분 하위 증가
  CNT10 CNT_M_LOW  (CLK, RST, CARRY3,  MIN_L, CARRY4);  // 분 하위
   assign CARRY5 = CARRY3 & CARRY4;  // 분, 초가 "09 59"면 분 상위 증가
  CNT6  CNT_M_HIGH (CLK, RST, CARRY5,  MIN_H, CARRY6);  // 분 상위
```

```
    assign CARRY7 = CARRY5 & CARRY6;  // 분,초가 "59 59"면 분, 시간 하위 증가
   CNT10 CNT_H_LOW  (CLK, RST2, CARRY7, HOUR_L, CARRY8);  // 시간 하위
    assign CARRY9 = CARRY7 & CARRY8;  // 분,초가 "09 59 59"면 분, 시간 상위 증가
   CNT2 CNT_H_HIGH  (CLK, RST2, CARRY9, HOUR_H, CARRY10); // 시간 상위
   //  동적인 디스플레이
   DISP DISP (CLK,RST,HOUR_H,HOUR_L,MIN_H,MIN_L,SEC_H,SEC_L,SEGMENT,Q,POINT);
   // 시간이 "24"가 되었을 경우에 리셋 조건
      assign RST1 = ((HOUR_L == 4) & (HOUR_H == 2)) ? 1'b0 : 1'b1;
      assign RST2 = RST1 & RST;
endmodule
// 클록을 10으로 나눈다.
module CNT10 (CLK,RST,EN,OUT,CARRY);
    input RST,CLK,EN;
    output [3:0] OUT;              // 카운터 출력
    output CARRY;                  // 카운터 오버플로 플래그
    reg [3:0] OUT;                 // 카운터 출력을 레지스터로 선언
    wire  Q9;                      // 오버플로 플래그
    always @(posedge CLK)  begin // always 블록문 시작
       if (!RST) OUT <= 0;         // 리셋
       else if (EN)                // EN=1이면 카운트 시작
          if ( Q9)  OUT <= 0;      // 카운터 값이 "9"면 리셋
           else  OUT <= OUT + 1; // 카운트
      end   // always 블록문 끝
      // 카운트 값이 끝인지 검사
      assign Q9 = (OUT==9);   //  카운터 값이 "9"면 Q9=1
      assign CARRY = Q9 & EN;  // 캐리 출력
 endmodule  // CNT10 모듈 끝
//  클록을 6으로 나눈다.
module CNT6 (CLK,RST,EN,OUT,CARRY);
    input RST,CLK,EN;
    output [2:0] OUT;              // 카운터 출력
    output CARRY;                  // 카운터 오버플로 플래그
    reg [2:0] OUT;                 // 카운터 출력을 레지스터로 선언
```

```
    wire Q5;                        // 오버플로 플래그
    always @(posedge CLK)  begin // always 블록문 시작
      if (!RST) OUT <= 0;           // 리셋
      else if (EN)                  // EN=1이면 카운트 시작
        if (Q5)  OUT <= 0;          // 카운터 값이 "5"면 리셋
         else   OUT <= OUT + 1;  // 카운트
      end   // always 블록문 끝
      // 카운트 값이 끝인지 검사
    assign Q5 = (OUT==5);     // 카운터 값이 "5"면 Q5=1
    assign CARRY = Q5 & EN;  // 캐리 출력
endmodule  // CNT6 모듈 끝
// 클록/2
module CNT2 (CLK,RST,EN,OUT,CARRY);
    input RST,CLK,EN;
    output [1:0] OUT;               // 카운터 출력
    output CARRY;                   // 카운터 오버플로 플래그
    reg [1:0] OUT;                  // 카운터 출력을 레지스터로 선언
    wire Q2;                        // 오버플로 플래그
    always @(posedge CLK)  begin // always 블록문 시작
       if (!RST) OUT <= 0;          // 리셋
       else if (EN)                 // EN=1이면 카운트 시작
         if ( Q2) OUT <= 0;         // 카운터 값이 "2"면 리셋
         else  OUT <= OUT + 1;     // 카운트
       end   // always 블록문 끝
      // 카운트 값이 끝인지 검사
     assign Q2 = (OUT==2);     // 카운터 값이 "2"면 Q2=1
     assign CARRY = Q2 & EN;  // 캐리 출력
endmodule  // CNT2 모듈 끝
// 동적인 디스플레이
module DISP (CLK,RST,HOUR_H,HOUR_L,MIN_H,MIN_L,SEC_H,SEC_L,SEGMENT,Q,POINT);
    input CLK,RST;
    input  [1:0] HOUR_H;          // 시간의 상위 입력
    input  [3:0] HOUR_L;          // 시간의 하위 입력
```

```
  input  [2:0] MIN_H,SEC_H;  // 분, 초의 상위 입력
  input  [3:0] MIN_L,SEC_L;  // 분, 초의 하위 입력
  output [6:0] SEGMENT;      // 7세그먼트 출력
  output [5:0] Q;            // 디지트 출력
  output POINT;              // 소수점 출력
  wire CARRY;                // CNT6의 오버플로를 받기 위해서 와이어 선언
  wire [2:0] COMCNT;         // CNT6의 출력을 받기 위해서 와이어 선언
/* CNT6 모듈의 COMCNT를 받아서 디스플레이할 버퍼 선택*/
 function [3:0] DIGIT_SEL;
   input [2:0] COMCNT;    // CNT6 모듈의 COMCNT 입력
   input [3:0] SEC_L;     // 초의 하위 입력
   input [2:0] SEC_H;     // 초의 상위 입력
   input [3:0] MIN_L;     // 분의 하위 입력
   input [2:0] MIN_H;     // 분의 상위 입력
   input [3:0] HOUR_L;    // 시의 하위 입력
   input [1:0] HOUR_H;    // 시의 상위 입력
    case (COMCNT)         // COMCNT를 이용하여 디스플레이할 값 결정
      0 : DIGIT_SEL = SEC_L;   // COMCNT=0이면 초의 하위
      1 : DIGIT_SEL = SEC_H;   // COMCNT=1이면 초의 상위
      2 : DIGIT_SEL = MIN_L;   // COMCNT=2이면 분의 하위
      3 : DIGIT_SEL = MIN_H;   // COMCNT=3이면 분의 상위
      4 : DIGIT_SEL = HOUR_L;  // COMCNT=4이면 시의 하위
      5 : DIGIT_SEL = HOUR_H;  // COMCNT=5이면 시의 상위
      default : DIGIT_SEL = 4'b0000;  // 해당되는 경우가 없으면 0
    endcase       // case문 끝
 endfunction      // function문 끝
 /* 평션 DIGIT_SEL에서 얻은 값을 7세그먼트 포맷으로 변경 */
  function [6:0] SEG_DEC;
    input [3:0] DIGIT;    // 입력 4비트
     case (DIGIT)         //gfe dcba
       4'h0 : SEG_DEC = 7'b011_1111;   // "0" 디스플레이 형태
       4'h1 : SEG_DEC = 7'b000_0110;   // "1" 디스플레이 형태
       4'h2 : SEG_DEC = 7'b101_1011;   // "2" 디스플레이 형태
```

```
        4'h3 : SEG_DEC = 7'b100_1111;   // "3" 디스플레이 형태
        4'h4 : SEG_DEC = 7'b110_0110;   // "4" 디스플레이 형태
        4'h5 : SEG_DEC = 7'b110_1101;   // "5" 디스플레이 형태
        4'h6 : SEG_DEC = 7'b111_1101;   // "6" 디스플레이 형태
        4'h7 : SEG_DEC = 7'b000_0111;   // "7" 디스플레이 형태
        4'h8 : SEG_DEC = 7'b111_1111;   // "8" 디스플레이 형태
        4'h9 : SEG_DEC = 7'b110_1111;   // "9" 디스플레이 형태
        default : SEG_DEC = 7'b000_0000;   // "A~F"인 경우는 7세그먼트 OFF
      endcase       // case문 끝
    endfunction       // function문 끝
  /* CNT6 모듈의 COMCNT를 받아서 디스플레이할 7세그먼트 선택*/
    function [6:0] COM_DEC;
      input [2:0] COMCNT;  // CNT6 모듈의 COMCNT
      case (COMCNT)   //최상위 비트는 소숫점 인에이블("1"), 디스에이블("0")
        0 : COM_DEC = 7'b0000001;  // 초의 하위
        1 : COM_DEC = 7'b0000010;  // 초의 상위
        2 : COM_DEC = 7'b1000100;  // 분의 하위 및 분, 초사이의 소수점
        3 : COM_DEC = 7'b0001000;  // 분의 상위
        4 : COM_DEC = 7'b1010000;  // 시의 하위 및 시, 분사이의 소수점
        5 : COM_DEC = 7'b0100000;  // 시의 상위
        default : COM_DEC = 7'b0000000; // 없는 경우는 7세그먼트 OFF
      endcase       // case문 끝
    endfunction       // function문 끝
    // CNT6를 호출해서, 현재의 카운트 값을 COMCNT로 받는다.
    CNT6 COM (CLK,RST,1'b1,COMCNT,CARRY);
  // 시, 분, 초 세그먼트 디스플레이
  assign SEGMENT = SEG_DEC(DIGIT_SEL(COMCNT,SEC_L,SEC_H,MIN_L,MIN_H,HOUR_L,HOUR_H));
  assign {POINT,Q} = COM_DEC(COMCNT);  // 소수점 및 해당되는 7세그먼트 ON
endmodule      // DISP_모듈 끝
```

B 시뮬레이션하기

1 테스트 벤치를 만들어서 다음과 같이 수정한다(HELP.PDF 참조).

```
`timescale 1us/1us          // 기본 스텝 시간(us)
module testbench;           // 모듈 이름
   reg  RST,CLK;            // 입력을 레지스터로 선언
   wire [6:0] SEGMENT;      // 7세그먼트 출력을 와이어로 선언
   wire [5:0] Q;            // 디지트 출력을 와이어로 선언
   wire  POINT;             // 7세그먼트 소수점 출력을 와이어로 선언
   parameter STEP = 1000; // 1[ms] 클록 주기
// 테스트벤치에서 자동으로 만들어진 것에서 인스턴스 이름(WATCH)만 변경한다.
WATCH WATCH (.CLK(CLK),.RST(RST),.SEGMENT(SEGMENT),.POINT(POINT),.Q(Q));
   always #(STEP/2)  CLK = ~CLK;  // 클록 주기
   initial begin                  // 입력 값 설정
           CLK=1; RST=0; #(STEP*2-10);        // 리셋
           RST=1; #(STEP*1000000); $stop;     // STEP×1000000 기간 동안 동작
    end   // initial 블록문 끝
    // 계속해서 시, 분, 초를 모니터로 디스플레이
    always #STEP $display("%0d, TIME=%d%d:%d%d:%d%d",
           $stime/STEP,WATCH.HOUR_H,WATCH.HOUR_L,
           WATCH.MIN_H,WATCH.MIN_L,
           WATCH.SEC_H,WATCH.SEC_L);
endmodule                      // 테스트벤치 모듈 끝
```

2 시뮬레이션하기(HELP.PDF 참조).

(1) ModelSim을 이용하여 시뮬레이션시키면 다음과 같이 디스플레이 된다.

```
                :
    # 60000, TIME=0 0:0 0:5 9
    # 60001, TIME=0 0:0 0:5 9
    # 60002, TIME=0 0:0 1:0 0  ← 60002 스텝에서 분 증가
    # 60003, TIME=0 0:0 1:0 0
    # 60004, TIME=0 0:0 1:0 0
    # 60005, TIME=0 0:0 1:0 0
```

```
# 60006, TIME=0 0:0 1:0 0
# 60007, TIME=0 0:0 1:0 0
        ⋮
```

(2) ModelSim의 런, 스톱 기능을 이용하여 분 및 시간의 증가를 확인해보자.

C 실험하기(MDA-ASIC 트레이너가 없는 사람은 실험을 하지 않아도 된다).

【1】 C\Verilog\FPGA 혹은 CPLD 이름\HELP.PDF를 참조해서, FPGA 혹은 CPLD로 라이트한다.

【2】 MDA-ASIC 트레이너에서 "DYNAMIC DISPLAY" 토글 스위치를 "ON"시켜서 출력을 인에이블시킨다.

【3】 MDA-ASIC 트레이너에서 그림 8-3-9의 입력 CLK를 "1kHz"로 맞춘다.

【4】 MDA-ASIC 트레이너에서 H_L0(그림 8-3-9의 RST)을 눌러서 리셋시킨다.
그림 8-3-9에서 처음 7세그먼트의 상태는? 시계가 되는지 확인해보자.

D 연습문제

1 기능 추가하기

그림 8-3-9의 "X0=H"면 시계 정지, "X0=L"이면 시계 동작 기능을 추가해서, 계층 설계 방법으로 시계를 다시 설계하고, 시뮬레이션해서 기능을 확인한 다음, C 실험하기 에서와 같은 방법으로 기능을 확인해보자.

2 초시계 설계하기

그림 8-3-9의 "X0=H"면 초시계 정지, "X0=L"이면 초시계 동작 기능을 갖고, 그림 8-3-10의 기능이 되도록, 계층 설계 방법으로 초시계를 설계하고, 시뮬레이션해서 기능을 확인한 다음, C 실험하기 에서와 같은 방법으로 기능을 확인해보자.

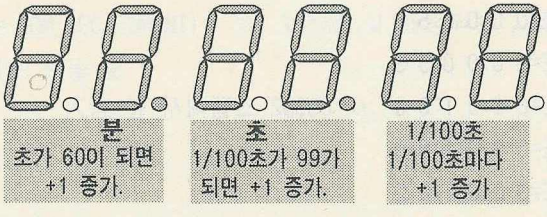


그림 8-3-10. 초시계 디스플레이 형태

8-8 데이터 통신하기

이 장에서는 직렬로 데이터 통신을 할 수 있는 UART(Universal Asynchronous Receiver Transmitter)를 FPGA/CPLD로 설계해보자.

8-8-1 UART 내부 기능

그림 8-8-1과 같은 UART를 설계하기로 하며, UART의 내부 블록도를 설명하면 다음과 같다.

1 TxCLK, RxCLK

보통 TxCLK를 송신용 시리얼 클록, RxCLK를 수신용 시리얼 클록이라 부르며, 그림 8-8-2와 같이 1비트를 전송하는데 외부에서 입력되는 CLK가 16개 필요하다.

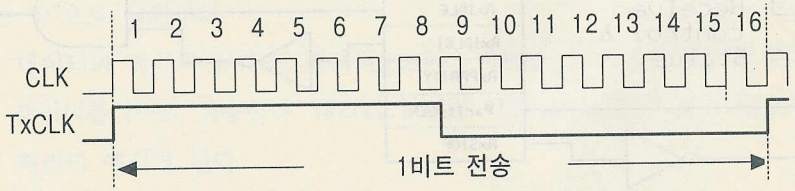


그림 8-8-2. 1비트 송신/수신에 필요한 클록

만약, 1초에 9600비트(Bit Per Second)를 전송하려면, 9600×16=153600[Hz]의 외부 클록(CLK)이 필요하게 되는 것이다.

2 TxHOLD,TxREG

$\overline{WR}$ 신호의 하강 모서리에서 송신할 데이터(D_IN(7~0))가 TxHOLD 레지스터에 저장되고, 이어서 TxREG 레지스터에 저장되고, TxREG 레지스터의 내용이 TxCLK에 동기되어 오른쪽으로 이동(최하위 비트부터 전송)하면서, TxD핀으로 직렬로 전송되게 된다. TxHOLD 레지스터가 전송이 끝나서 비게 되면, 송신 가능 플래그 TxRDY가 "1"이 된다(다음 장의 송신기 설계에서 설명).

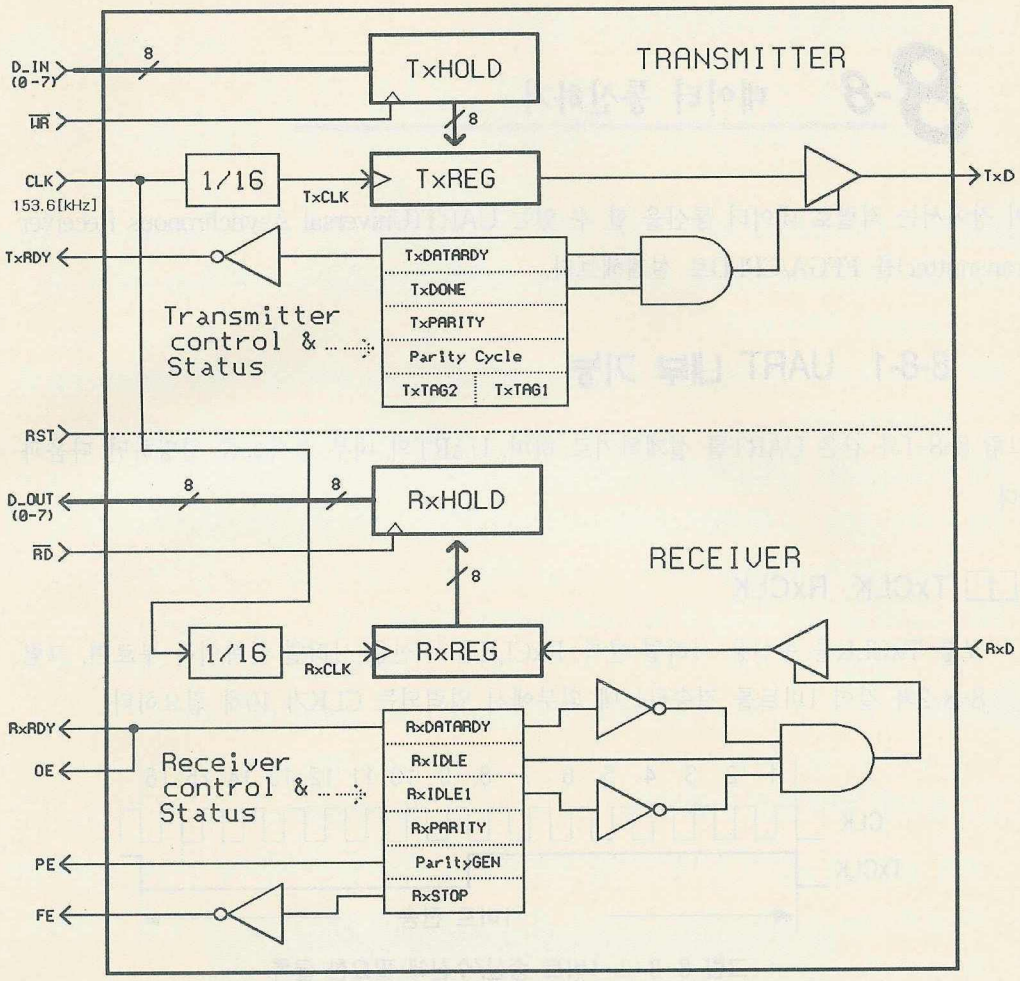


그림 8-8-1. UART 내부 블록도

3 RxHOLD, RxREG

수신 가능한 상태가 되면, RxCLK에 동기되어 RxD핀으로 직렬로 입력된 신호는 RxREG로 오른쪽으로 이동(최상위부터 입력)되면서 수신하게 된다. 수신이 끝나면 RxREG 레지스터가 RxHOLD에 저장되고, 수신 완료 플래그 RxRDY가 "1"이 되어, 수신 완료를 외부에 알려 준다. 이때 OE(Overrun Error) 플래그도 "1"이 된다. 그리고 수신된 내용에 따라 PE(Parity Error), FE(Framing Error)가 "1" 혹은 "0"으로 설정된다. $\overline{RD}$ 신호의 하강 모서리에서 수신된 데이터(RxHOLD)가 D_OUT(7~0)로 출력된다.

8-8-2 송신기 설계하기

그림 8-8-1을 참조해서 송신기를 설계해 보자.

1 일반적인 데이터 전송 형식

일반적으로 데이터 전송 형식은 그림 8-8-3과 같은 형식으로 되어 있으며, 전송 순서는 다음과 같다.

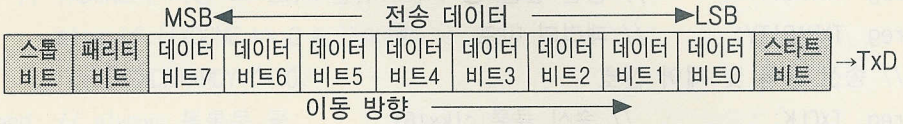


그림 8-8-3. 직렬 데이터 전송 형식

(1) 스타트 비트를 전송하기 위하여, TxCLK 1주기 동안 "0"을 전송

(2) 최하위 비트(Least Significant Bit)부터 최상위 비트(Most Significant Bit)순서로 TxCLK에 동기되어 오른쪽으로 이동하면서 전송되고, 보통 오른쪽으로 전송되면서 빈자리는 "0"으로 채워진다.

(3) 전송할 데이터의 전송이 끝나면, 패리티(parity) 비트를 홀수(odd) 패리티 혹은 짝수(even) 패리티를 1비트 전송한다. 패리티가 전송 약속(protocol)이 맞지 않으면, 수신기에서 패리티 에러가 된다.

(4) 스톱 비트를 1비트, 혹은 1½ 비트 혹은 2비트로 전송한다. 만약 스톱 비트가 없거나 스톱 비트의 비트 수가 약속과 다를 경우는 수신기에서 프레밍(framing) 에러가 발생한다.

(5) 스톱 비트까지 전송이 끝나면, 송신기 출력은 "1"로 되게 한다.

2 송신기 설계하기

"1비트 스타트 비트, 8비트 데이터, 홀수 패리티, 1스톱 비트"의 형식으로 송신기를 설계하면 다음과 같다.

```
module TRANSMITTER(RST, CLK, WR, D, TXD, TXRDY);
    input RST, CLK;          // 리셋, 시리얼 클록 입력
    input WR;                //송신 데이터 라이트 신호
```

```
input [7:0] D;          // 송신용 데이터
output TXD;             // 송신 데이터
output TXRDY;           // 송신 데이터 가능 플래그
reg TXD;
// 송신 데이터 홀딩 레지스터(Transmit data holding register)
reg  [7:0] TXHOLD;
reg  [7:0] TXREG;     // 송신 시프트 레지스터
reg  TXTAG2;          // 송신 끝을 검사하기 위한 비트
reg  TXTAG1;          // 송신 끝을 검사하기 위한 비트
reg  TXPARITY;        // 패리티 비트
// 송신 클록 및 제어 신호
reg  TXCLK;           // 송신 클록(clkx16)
wire  TXDONE;         // '1'이면, 시프트 중임을 나타낸다.
wire  PARITYCYCLE;    // '1'이면, 패리티 사이클
reg  TXDATARDY;       // '1'이면, TXHOLD 레지스터에 송신 데이터 저장 가능
// TXCLK를 만드는데 사용하는 카운터
reg  [2:0] CNT;
reg  WR1,WR2;         // WR 신호 1,2 사이클을 지연하기 위한 변수
reg  TXDONE1;         // TXDONE 신호 1사이클을 지연하기 위한 변수
// WR 신호의 하강 모서리에서 입력된 데이터를 TXHOLD 레지스터에 저장
always @(posedge CLK or negedge RST)
 begin  // always 블록문 시작
   if (!RST) begin          // 리셋이면 클리어
       WR1 <= 0 ;  WR2 <= 0 ; TXDONE1 <= 0;
      end  // if 블록문 끝
    else begin    // 클록의 상승 모서리에서 동작
   // 에지 검출을 위해서 WR, TXDONE 신호 지연
      WR2 <= WR1;
      WR1 <= WR;
      TXDONE1 <= TXDONE;
     end  // else 블록문 끝
  end     // always 블록문 끝
// WR의 하강 모서리에서 송신 데이터를 TXHOLD에 로드하고, TXDATARDY=1로 한다.
// TXDONE의 하강 모서리에서 TXDATARDY=0이 된다.
always @(posedge CLK or negedge RST)
```

```
  begin  // always 블록문 시작
     if (!RST)        // 리셋이면 클리어
        TXDATARDY <= 0;
      else     // 클록의 상승 모서리에서 동작
      // WR의 하강 모서리에서 송신 데이터가 TXHOLD로 래치
        if(WR1 == 0 & WR2 == 1) begin
              TXHOLD = D; TXDATARDY <= 1;
            end  // if 블록문 끝
      // TXDONE의 하강 모서리에서, txhold 로드
         else if (TXDONE == 0 & TXDONE1 == 1)
              TXDATARDY <= 0;
    end  // always 블록문 끝
  // 송신용 시프트 클록인 TXCLK 클록을 만든다.
  // CLK= TXCLK(16) x 9600 = 153600
  always @(posedge CLK or negedge RST)
  begin  // always 블록문 시작
// CLK16핀의 8개 입력마다 TXCLK를 토글시켜서,시프트 클록인 x16의 반주기를 만든다.
// 결과적으로 CLK 클록 주기의 곱하기 16이 된다.
    if (!RST) begin   // 리셋(RST=0)이면 송신용 클록 및 카운터 클리어
       CNT <= 0;  TXCLK <= 0;
       end  // if 블록문 끝
     else  begin     // CLK핀의 상승 모서리에서 동작
        CNT <= CNT + 1 ;          // 카운터 + 1
       if (CNT == 0)              //  카운터가 '0'이면, TXCLK 플래그 토글
           TXCLK <= ~TXCLK;       // CLK 클록 핀 8번 입력마다 토글
       end  // else 블록문 끝
  end  // always 블록문 끝
   // 송신 시프트
  always @(posedge TXCLK or negedge RST)
   begin : TXSHIFT  // TXSHIFT 블록문 시작
     if (!RST) begin          // 리셋이면 송신 신호 및 레지스터 클리어
        TXREG  <= 0 ; TXTAG1 <= 0 ; TXTAG2 <= 0 ;
        TXPARITY <= 0 ;
       end  // if 블록문 끝
     else                                  // TXCLK의 상승 모서리에서 동작
```

```
        if ((TXDONE & TXDATARDY)==1) begin // 송신 가능?
            // 레지스터 초기 설정 및 다음 데이터 로드
            TXREG    <= TXHOLD;  // 송신 데이터 저장
            TXTAG2   <= 1;       // 송신 검출을 하기 위한 태그 비트
            TXTAG1   <= 1;       // 시프트 중임을 나타낸다.
            TXPARITY <= 1;       // '1'이면 홀수 패리티('0'이면 짝수 패리티)
          end  // if 블록문 끝
        else begin
            TXREG   <= {TXTAG1,TXREG[7:1]};  // 송신 데이터 전송
            TXTAG1  <= TXTAG2;  TXTAG2  <= 0;
            // 패리티 비트를 만든다.
            TXPARITY  <= TXPARITY ^ TXREG[0];
            end  // else 블록문 끝
      end  // TXSHIFT 블록문 끝
   // 송신
   always @(posedge TXCLK or negedge RST)
   begin : TX_OUT    // TX_OUT 블록문 시작
     if (!RST)  TXD <= 1 ;       // 리셋이면 송신 신호 클리어
     // 송신 데이터 혹은 패리티 비트 혹은 스톱/아이들 비트를 tx핀으로 출력
     else if (TXDATARDY)  TXD <= 0;       // 스타트 비트
     else if (TXDONE)  TXD <= 1;          // 스톱/아이들 비트
     else if (PARITYCYCLE) TXD <= TXPARITY;     // 패리티 비트
     else     TXD <= TXREG[0];        // 송신 데이터
    end    // TX_OUT 블록문 끝
   // txtag2이 txreg(1)에 도달했을 때, paritycycle = 1
   //  패리티 비트를 tx핀으로 출력
   assign PARITYCYCLE = TXREG[1] & ~(TXTAG2 | TXTAG1 | (|TXREG[5:2]));
   // txtag2가 tx핀에 도달했을 때(시프트 중), txdone = 1
   assign TXDONE = ~(TXTAG2 | TXTAG1 | (| TXREG[7:0]));
   // 송신 가능을 나타내는 플래그
   assign TXRDY = ~TXDATARDY;
endmodule  // TRANSMITTER 모듈 끝
```

8-8-3 수신기 설계하기

그림 8-8-1을 참조해서 수신기를 설계해 보자.

1 일반적인 데이터 전송 형식

송신기에서 최하위 비트를 전송하므로 수신기에서는 그림 8-8-4와 같이 최상위 비트로 입력하면서 오른쪽으로 이동한다.

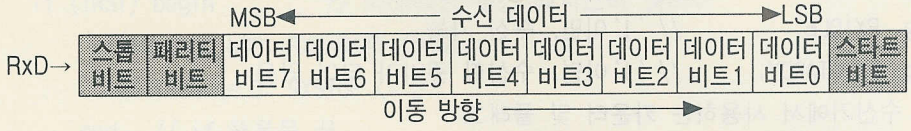


그림 8-8-4. 직렬 데이터 수신 형식

(1) 스타트 비트를 찾는다. 이 것을 헌팅(hunting) 모드라 하며, 이 스타트 비트를 찾아야 다음 동작을 하게 된다.

(2) RxCLK에 동기되어 최상위 비트(Most Significant Bit)쪽으로 수신하면서 최하위 비트(Least Significant Bit)로 이동하게 된다. 그러면 송신기에서 전송한 최하위 비트가 수신기의 최하위 비트가 된다.

(3) 수신이 끝나면, 패리티(parity) 비트를 검사해서 에러 유무를 출력하고, 스톱 비트가 없는 경우는 프레밍 에러를 출력하게 된다.

(4) 수신된 데이터가 수신 홀딩 레지스터로 전송된다.

(5) 스톱 비트까지 수신되면, 수신기 입력(RXD)은 "1"이 입력되게 된다.

2 수신기 설계하기

"1비트 스타트 비트, 8비트 데이터, 홀수 패리티, 1스톱 비트"의 형식으로 수신기를 설계하면 다음과 같다.

```
module RECEIVER(RST, CLK, RD, RXD, D, PE, FE, OE, RXRDY);
    input RST, CLK, RD, RXD; // 리셋, 시리얼 클록, 수신 데이터 리드, 수신 데이터
    output [7:0] D;          // 수신용 데이터
    output PE, FE, OE;       // 패리티 에러, 프레밍 에러, 오버런 에러
    output RXRDY;            // 수신 가능 플래그
    reg [7:0] D;             // 수신용 데이터
```

```
reg PE,FE,OE;          // 에러 출력을 레지스터로 선언
// 수신 시프트 레지스터
reg  [7:0] RXHOLD;    // 수신 레지스터
reg  [7:0] RXREG;     // 수신 시프트 레지스터
reg  RXPARITY;        // 수신한 비트의 패리티 비트 저장
reg  PARITYGEN;       // 수신한 데이터의 패리티 발생 비트 저장
reg  RXSTOP;          // 수신 데이터의 스톱 비트 저장
// 수신 클록 및 수신용 제어 신호
reg  RXCLK;           // 수신을 하기 위한 시프트 클록
reg  RXIDLE;          // '1'이면, 수신 가능
reg  RXDATARDY;       // '1'이면, 수신할 준비가 되어 있음
// 수신기에서 사용하는 카운터 및 플래그
reg [3:0] RXCNT;  // 카운터로 사용하기 위하여 변수 선언
reg RX1;          // RXD 수신 핀의 1 사이클 지연 플래그
reg HUNT;         // 스타트 비트를 찾기 위한 헌팅 플래그
reg RD1, RD2;          // 리드 입력 1,2 사이클 지연 변수
reg RXIDLE1;           // RXIDLE 신호 1사이클 지연
// 수신용 시프트 클록인 RXCLK를 만든다.
always @(posedge CLK or negedge RST)
begin  // always 블록문 시작
  if (!RST) begin          // 리셋이면 클리어
     HUNT <= 0 ;  RXCNT <= 0 ;  RX1 <= 0 ; RXCLK <= 0 ;
    end  // if 블록문 끝
  else begin                    // CLK핀의 상승 모서리에서 동작
   // RXCLK = CLK핀 주파수/16
    RXCLK <= RXCNT[3];
    // 에지 검출을 위해서 1클록 동안 지연
    RX1 <= RXD;
       // 스타트 비트를 찾고 있는 중이면 HUNT=1
       // 스타트 비트는 클록의 8번째 하강 모서리에서 RXD=0을 찾는 것이다.
       // 아이들 모드이고, 하강 모서리에서 헌팅 시작
      if(RXIDLE == 1 & RXD == 0 & RX1 == 1)  HUNT <= 1;
        // RXD ='1'이거나 데이터가 시프트될 때 헌팅 모드 정지
      if (RXIDLE == 0 | RXD == 1)     HUNT <= 0;
        // 아이들, 헌팅 모드면 카운터 증가
```

```
      if (RXIDLE == 0 | HUNT == 1)  RXCNT <= RXCNT + 1;
         // 아이들 모드일 때는 '1'을 홀드하고, 에지를 기다린다.
        else      RXCNT <= 4'b0001;
    end  // else 블록문 끝
  end    // always 블록문 끝
  // 아이들이 아닐 때, 매 RXCLK에서 수신 시프트
always @(posedge RXCLK or negedge RST)
  begin  // always 블록문 시작
   if (!RST) begin          // 리셋이면 수신 레지스터 클리어
       RXREG <= 8'h00; RXPARITY <= 0;
       PARITYGEN <= 0; RXSTOP <= 0 ;
      end   // if 블록문 끝
    else     // RXCLK의 상승 모서리에서 동작
      if (RXIDLE) begin  // RXIDLE=1이면 실행
       // 아이들 상태에서는 모두 '1'을 로드한다.
        RXREG <= 8'hff;
        RXPARITY <= 1;
        PARITYGEN <= 1;// 홀수 패리티 지정
        RXSTOP <= 0;
       end   // if 블록문 끝
      else begin
         // 아이들 상태가 아닐 때는 데이터 시프트 라이트
          // rxreg (0 TO 6) <= rxreg (1 TO 7), rxreg(7) <= rxparity;
          RXREG <={ RXPARITY,RXREG[7:1]};
           RXPARITY <= RXSTOP;                    // 패리티 비트
           PARITYGEN <= PARITYGEN ^ RXSTOP;       // 스톱 비트와 패리티 발생
           RXSTOP <= RXD;                         // 스톱 비트 저장
     end   // else 블록문 끝
  end      // always 블록문 끝
  //  rxidle는 rxclk에 동기 되기 때문에 비동기 프리셋이 필요하다.
  // 그래서 그 값을 이용하여 rxclk를 발생할 것인지, 아닌지를 결정하여야 한다.
always @(posedge RXCLK or negedge RST)
  begin      // always 블록문 시작
   if (!RST) RXIDLE <= 0;          // 리셋이면 RXIDLE 클리어
   else                           // RXCLK의 상승 모서리에서 동작
```

```
        RXIDLE <= ~RXIDLE & ~RXREG[0];
    end       // always 블록문 끝
  // RD의 하강 모서리를 만든다.
  always @(posedge CLK or negedge RST)
    begin       // always 블록문 시작
     if (!RST) begin          // 리셋이면 클리어
           RD1 <= 0;  RD2 <= 0; RXIDLE1 <= 0;
       end       // if 블록문 끝
     else begin     // 클록의 상승 모서리에서 동작
       // 에지를 검출하기 위하여 rxidle 1사이클 지연
       RXIDLE1 <= RXIDLE;
       RD2 <= RD1;
       RD1 <= RD;
      end       // else 블록문 끝
    end         // always 블록문 끝
 // 수신된 데이터를 읽기 및 에러 검사
  always @(posedge CLK or negedge RST)
   begin       // always 블록문 시작
     if (!RST) begin          // 리셋이면 에러 플래그 및 수신 레지스터 클리어
        OE <= 0; PE <= 0 ; FE <= 0;
        RXHOLD <= 0;    RXDATARDY <= 0;
       end       // if 블록문 끝
     else begin
       // 데이터가 리드되면 데이터 레지스터 및 에러 클리어
       if (RD1 == 0 & RD2 == 1) begin
           D <= RXHOLD;
           RXDATARDY  <= 0;
           PE <= 0; FE <= 0; OE <= 0;
         end       // if 블록문 끝
       // RXIDLE의 상승 모서리를 찾아서, 출력 레지스터를 업데이트시킨다.
       if( RXIDLE == 1 & RXIDLE1 == 0) begin
         // 앞의 데이터가 아직 있으면 오버런 에러
          if (RXDATARDY) OE <= 1;
           else begin
             // 홀딩 레지스터가 비어 있으면 오버런 에러가 아니다.
```

```
                OE <= 0;     // 홀딩 레지스터 업데이트
               RXHOLD <= RXREG;    // 패리티 에러면 paritygen = 1
               PE <= PARITYGEN;    // 스톱 비트가 없으면 프레밍 에러
               FE <= ~RXSTOP;
               // 데이터가 읽을 준비가 되었음을 알리는 신호
               RXDATARDY <= 1;
            end  // else 블록문 끝
         end     //if 블록문 끝
      end          // else 블록문 끝
   end             // always 블록문 끝
   // 수신 가능을 나타내는 플래그
   assign RXRDY = RXDATARDY;
endmodule  // RECEIVER 모듈 끝
```

8-8-4 UART 설계하기

그림 8-8-5과 같은 회로를 이용하여, 지금까지 설명한 송·수신기를 UART를 Verilog HDL로 설계해 보자. 송신할 데이터는 B(상위 4비트), A(하위 4비트)의 16진 숫자를 이용하여 ASCII 코드로 송신하고, 수신한 데이터는 7세그먼트를 이용하여 ASCII 코드로 디스플레이하게 한다.

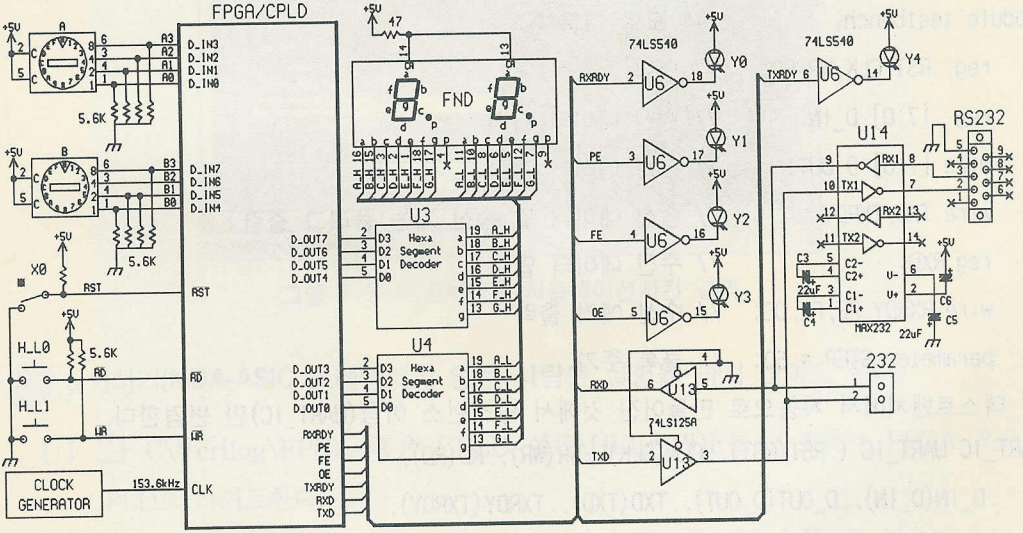


그림 8-8-5. UART 인터페이스

A Verilog HDL 설계하기

C\Verilog\FPGA 혹은 CPLD 이름\UART

```
module UART_IC(RST, CLK, WR, RD, D_IN, D_OUT, TXD, TXRDY, RXD, RXRDY, PE, FE, OE);
    input RST, CLK, WR, RD;      // 리셋, 클록, 라이트, 리드 신호
    input [7:0] D_IN;            // 송신 데이터 입력
    output [7:0] D_OUT;          // 수신 데이터 출력
    output TXD, TXRDY;           // 송신 데이터 및 송신 가능 플래그 출력
    input RXD;                   // 수신 데이터 입력
    output RXRDY, PE, FE, OE;    // 수신 에러 출력
    // 송신기, 수신기 모듈 콜
    TRANSMITTER TRANSMITTER(RST, CLK, WR, D_IN, TXD, TXRDY);
    RECEIVER RECEIVER(RST, CLK, RD, RXD, D_OUT, PE, FE, OE, RXRDY);
endmodule // UART_IC 모듈 끝
```

// 송신기, 수신기는 앞장에 있는 것을 그대로 사용한다.

B 시뮬레이션하기

1 테스트 벤치를 만들어서 다음과 같이 수정한다.

```
`timescale 1ns/1ns          //  기본 스텝 시간
module testbench;           //  모듈 이름
    reg  RST, CLK, WR, RD;  // 입력
    reg  [7:0] D_IN;        // 송신 데이터 입력
    wire [7:0] D_OUT;       // 수신 데이터 출력
    wire TXD, TXRDY;        // 송신 데이터 및 송신 가능 플래그 출력
    reg RXD;                // 수신 데이터 입력
    wire RXRDY, PE, FE, OE; // 수신 에러 출력
    parameter STEP = 50;    // 클록 주기
// 테스트벤치에서 자동으로 만들어진 것에서 인스턴스 이름(UART_IC)만 변경한다.
UART_IC UART_IC (.RST(RST), .CLK(CLK), .WR(WR), .RD(RD),
    .D_IN(D_IN), .D_OUT(D_OUT), .TXD(TXD), .TXRDY(TXRDY),
    .RXD(RXD), .RXRDY(RXRDY), .PE(PE), .FE(FE), .OE(OE));
```

```
always #(STEP/2)  CLK = ~CLK;  // 클록 발진
initial begin                  // 입력 값 설정
        CLK=1; RST=0; WR =1; RD=1; RXD =1; D_IN =0 ; // 리셋
        force D_OUT =0 ; #(STEP-10);        // 출력을 강제로 0으로 만든다.
        RST=1; D_IN = 8'h37; WR= 0; #STEP;  // 데이터 라이트
        WR= 1; #(STEP*32*10);               // 320주기 동안 동작
        $stop;                              // 정지
 end   // initial 블록문 끝
endmodule  // 테스트벤치 모듈 끝
```

2 시뮬레이션시킨다.

(1) ModelSim을 이용하여 시뮬레이션시키면 그림 8-8-6와 같다.

(2) 커서를 움직이면서 데이터가 전송되는지 확인해보자.

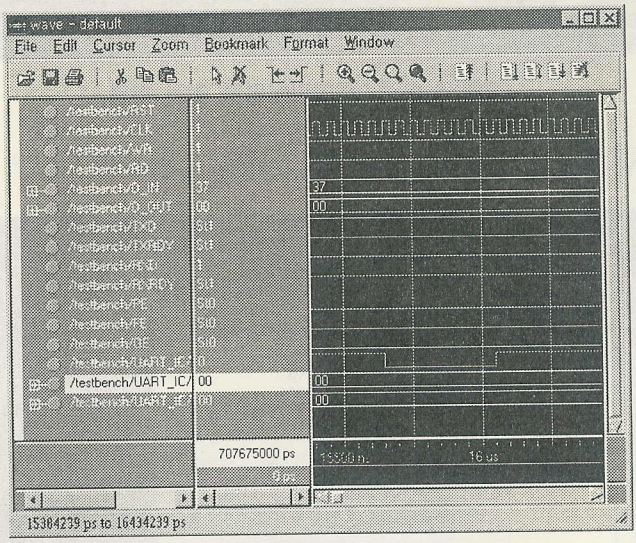


그림 8-8-6. UART를 시뮬레이션시킨 결과

D 실험하기(MDA-ASIC 트레이너가 없는 사람은 실험을 하지 않아도 된다).

【1】 C\Verilog\FPGA 혹은 CPLD 이름\HELP.PDF를 참조해서, FPGA 혹은 CPLD로 라이트한다.

【2】 MDA-ASIC 트레이너에서 그림 8-8-5의 입력 CLK를 "153.6kHz"로 맞춘다.

【3】 다음과 같이 윈도우 통신 프로그램을 셋업한다.

(1) 그림 8-8-7과 같이 "**보조 프로그램→통신→하이퍼터미널**"을 실행시킨다.

그림 8-8-7. 하이퍼터미널 찾기 화면

(2) 하이퍼터미널을 실행시킨 화면은 그림 8-8-8과 같다.

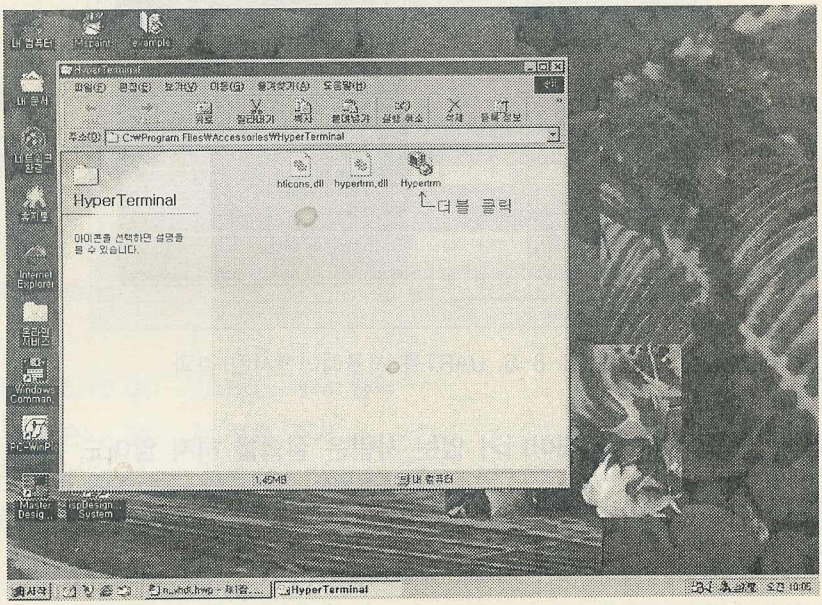


그림 8-8-8. 하이퍼터미널 실행 화면

(3) 그림 8-8-8에서 하이퍼터미널을 실행시키면 그림 8-8-9와 같이 초기 폴더 설정 화면이 디스플레이 된다. "**CPLD**"를 써넣고, 폴더를 결정해서 "**확인**"을 클릭한다.

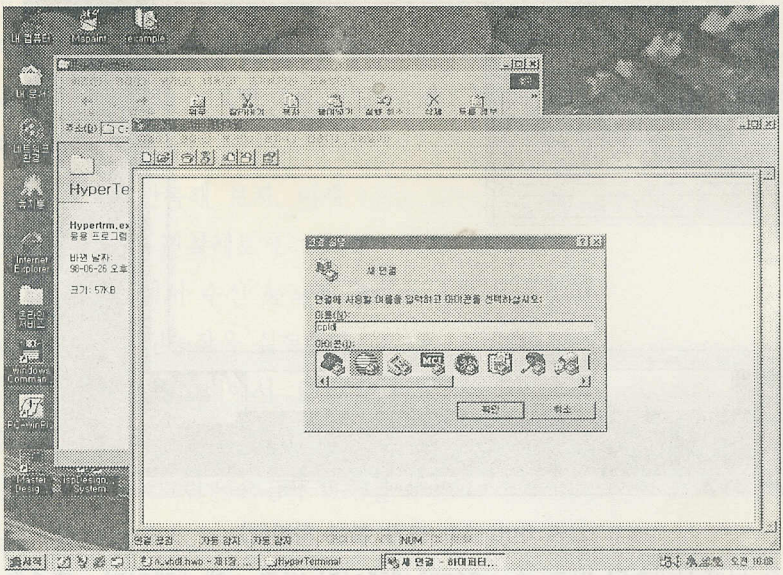


그림 8-8-9. 새 연결 폴더 화면

(4) 그림 8-8-9에서 "확인"으로 끝나면 그림 8-8-10과 같은 화면이 디스플레이 된다. "com1에 직접 연결"을 선택한다.

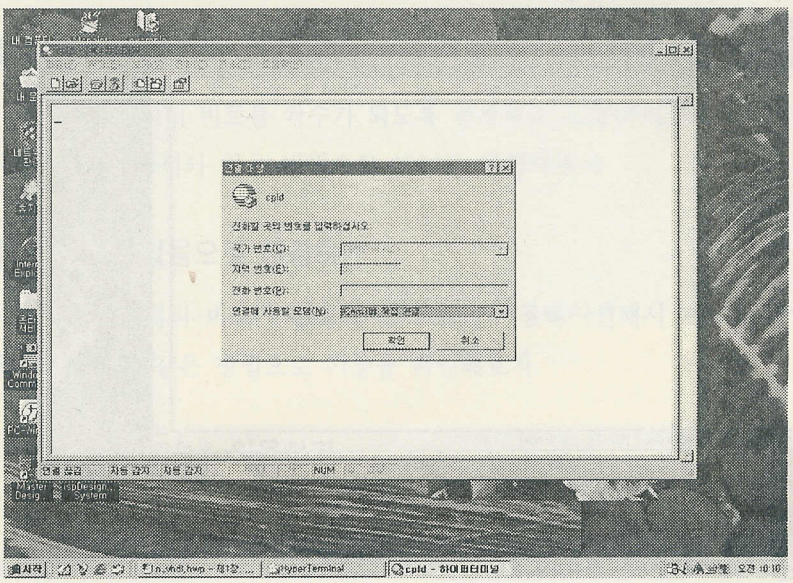


그림 8-8-10. 초기 설정 화면(1)

(5) 그림 8-8-10에서 "확인"으로 끝나면 그림 8-8-11와 같은 화면이 디스플레이된다.

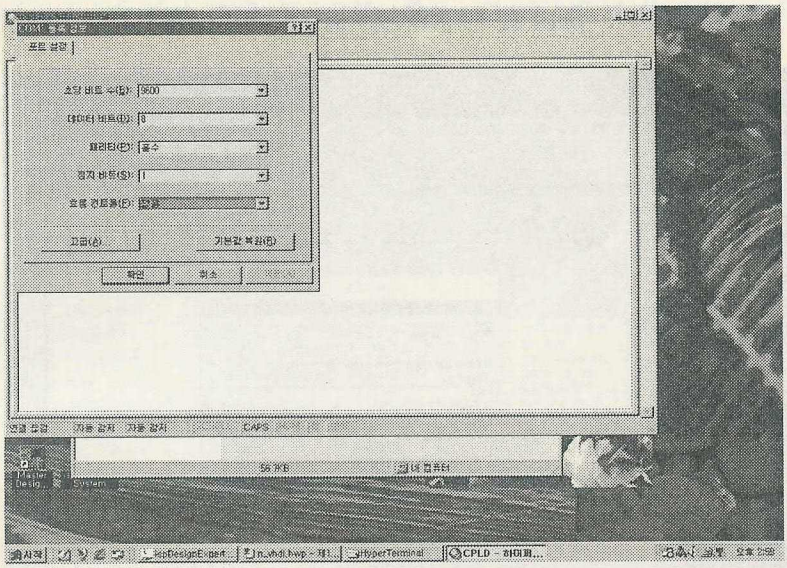


그림 8-8-11. 초기 설정 화면(2)

그림 8-8-11에서 "**초당 비트 수 : 9600**", "**데이터 비트 : 8**", "**패리티 : 홀수**", "**정지 비트 : 1**", "**흐름 컨트롤 : 하드웨어**"를 선택하고 "확인"으로 빠져 나오면 그림 8-8-12와 같이 데이터 통신이 가능한 화면이 디스플레이된다.

(6) 그림 8-8-12가 MDA-ASIC 트레이너와 데이터 통신이 가능한 화면이다.

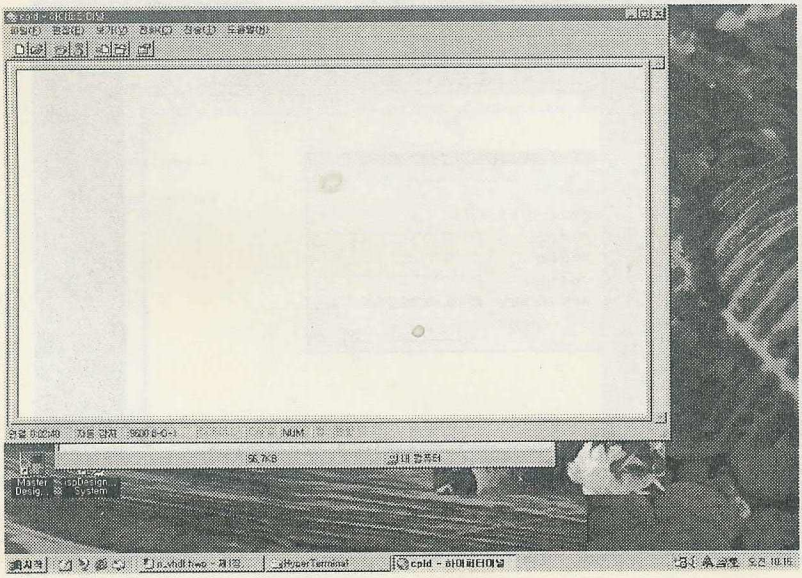


그림 8-8-12. 데이터 통신이 가능한 화면

【4】 MDA-ASIC 트레이너에서 X0(그림 8-8-5의 RST)을 "H"→"L"→"H"로 해서 리셋 시킨다.

【5】 다음과 같이 수신동작을 실험한다.

(1) PC에서 숫자 키를 누른다. 그리고 MDA-ASIC 트레이너에서 H_L0 스위치(그림 8-8-5의 $\overline{RD}$)를 누르면, 7세그먼트에 PC에서 송신된 값이 ASCII 값으로 디스플레이된다.

(2) (1)의 동작을 반복해 보자. 이때 LED Y0~Y3(그림 8-8-5의 RXRDY, PE, FE, OE)의 상태를 관찰해보자.

(3) (1), (2)의 동작에서 수신 동작이 제대로 되는지 확인한다.

【6】 다음과 같이 송신동작을 실험한다.

(1) MDA-ASIC 트레이너에서 B(상위 숫자), A(하위 숫자) 스위치를 숫자 "0~9" 중, 임의의 숫자의 ASCII 코드 값을 만든다.

(2) "H_L1" 스위치(그림 8-8-5의 $\overline{WR}$)를 누른다. 이때 그림 8-8-12의 PC 화면에 디스플레이되는 값은?

LED Y4(그림 8-8-8의 TXRDY)의 상태를 관찰해보자.

(3) (1), (2)의 동작을 반복해서 송신기 동작이 제대로 되는지 확인?

D 연습문제

1 패리티 비트 짝수로 변경하기

위의 실험에서 패리티 비트를 짝수가 되도록 설계하고, 시뮬레이션해서 기능을 확인한 다음, C 실험하기 에서와 같은 방법으로 기능을 확인해보자.

2 패리티 비트 없음으로 변경하기

위의 실험에서 패리티 비트가 없도록 설계하고, 시뮬레이션해서 기능을 확인한 다음, C 실험하기 에서와 같은 방법으로 기능을 확인해보자.

3 RXRDY, TXRDY 이용하기

위의 실험에서 "RXRDY, TXRDY"가 "1"이 되는 것을 검사해서, 수신 및 송신을 하도록 설계하고, 시뮬레이션해서 기능을 확인한 다음, C 실험하기 에서와 같은 방법으로 기능을 확인해보자.