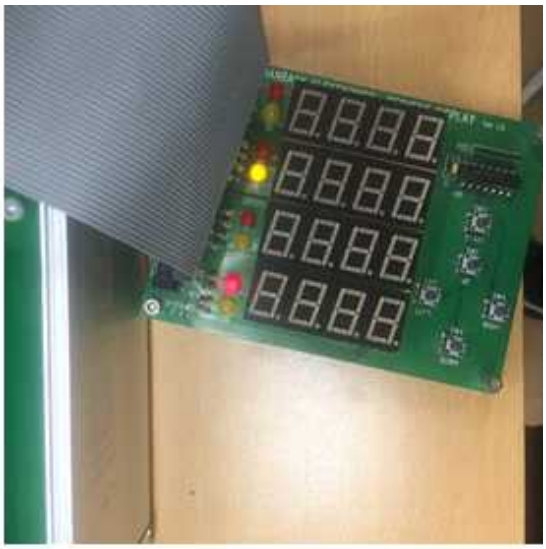
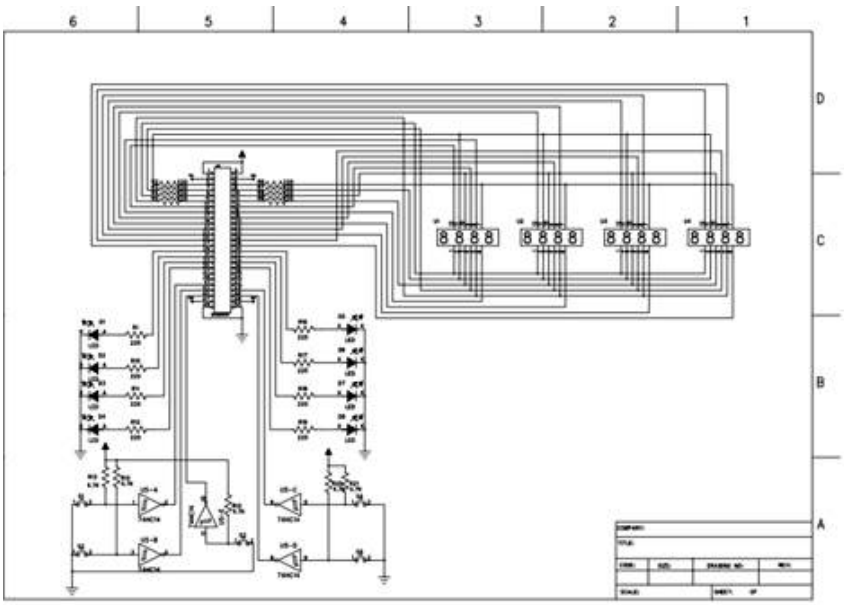
FPGA Puzzle

KangwonNational University 일반대학원 전자공학전공
박제창

■ 한백전자 Combo 장비와 Puzzle 모듈을 사용



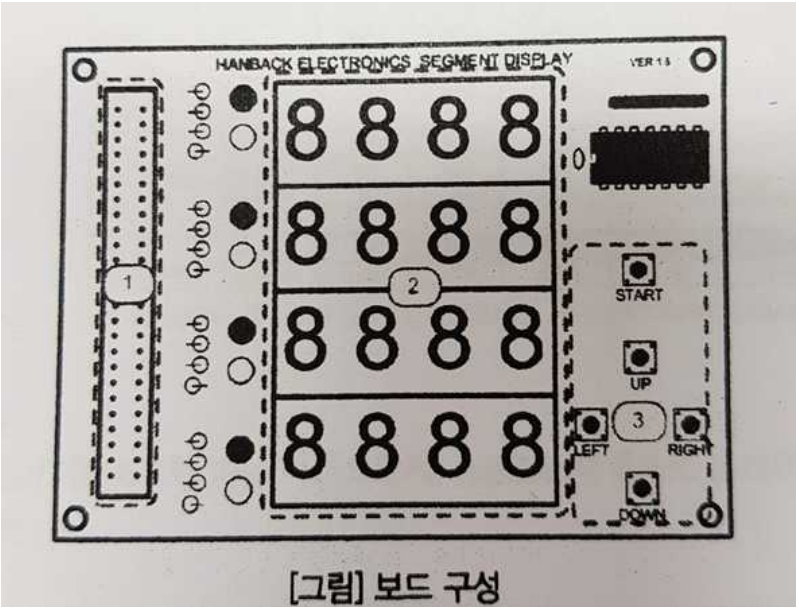


■ **퍼즐 보드 개요**

본 모듈은 4x4의 7-세그먼트 모듈을 이용한 퍼즐 게임으로 Rom에 저장되어 있는 데이터를 시작 버튼을 누르면 이중 하나의 데이터를 읽어와 세그먼트를 초기화 하고 이를 방향 버튼으로 0-E 까지 문자를 차례대로 맞추는 게임을 설계한다.

모든 모듈의 제어는 FPGA에 VHDL언어를 사용하여 설계된 로직에 의해 이루어 진다.

■ **퍼즐 보드 구성도**



[그림] 보드 구성

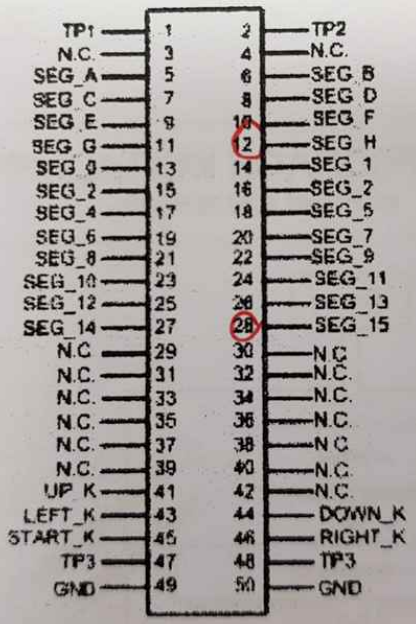
■ 퍼즐 보드 핀정보

[그림] 보드 구성

넥터(50핀)

E-COMBO 장비에 연결할 때 사용되는 커넥터

의 회로상 연결된 핀 정보이다.



[Application board 핀 정보]

1차 코드 작성 -> 2018-12-07

문제 : Rom을 사용하지 않으며 Decode 부분의 동작이 모호함.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity puzzle is
	port(
		clk			: in std_logic;
		up_k			: in std_logic;  -- up_side key in
		down_k			: in std_logic;  -- down_side key in
		left_k			: in std_logic; -- left_side key in
		right_k			: in std_logic;  -- right_side key in
		start_k			: in std_logic;	-- alone mode start, initialize puzzle, randum start

		data			: buffer std_logic_vector(7 downto 0); -- display data,
--		decode			: buffer std_logic_vector(3 downto 0);	-- each 7 segment led selector, 4x16 decoder
		verify			: buffer std_logic_vector(15 downto 0);
		pin_s			: out std_logic_vector(15 downto 0)

	);
end puzzle;

architecture sample1 of puzzle is

signal		data0		: std_logic_vector(3 downto 0);  -- 0
signal		data1		: std_logic_vector(3 downto 0);  -- 1
signal		data2		: std_logic_vector(3 downto 0);  -- 2
signal		data3		: std_logic_vector(3 downto 0);  -- 3
signal		data4		: std_logic_vector(3 downto 0);  -- 4
signal		data5		: std_logic_vector(3 downto 0);  -- 5
signal		data6		: std_logic_vector(3 downto 0);  -- 6
signal		data7		: std_logic_vector(3 downto 0);  -- 7
signal		data8		: std_logic_vector(3 downto 0);  -- 8
signal		data9		: std_logic_vector(3 downto 0);  -- 9
signal		dataA		: std_logic_vector(3 downto 0);  -- A
signal		dataB		: std_logic_vector(3 downto 0);  -- B
signal		dataC		: std_logic_vector(3 downto 0);  -- C
signal		dataD		: std_logic_vector(3 downto 0);  -- D
signal		dataE		: std_logic_vector(3 downto 0);  -- E
signal		dataF		: std_logic_vector(3 downto 0);  -- F

signal		decode		: std_logic_vector(3 downto 0);
signal displ_e		: std_logic;



begin

	process(clk)
	variable temp_up : std_logic_vector (1 downto 0);
	variable temp_dn : std_logic_vector (1 downto 0);
	variable temp_all	: std_logic_vector (3 downto 0);
	variable r_cnt		: std_logic_vector(3 downto 0);
	variable start_ran	: std_logic;
	variable veri		: std_logic;
	variable key_in : std_logic;
	variable key_in_m	: std_logic;
	variable unable : std_logic;
```

```
        variable init           : std_logic;
        variable mak_ran        : integer range 0 to 3;
        variable end_ran_m      : integer range 0 to 3;
        variable strobe_ran : std_logic;
        variable cnt            : integer range 0 to 3;
        variable cnt_ex         : integer range 0 to 4;
        variable p_cnt          : std_logic_vector(4 downto 0);

        variable ena_veri       : std_logic;
        begin
                if (clk='1' and clk'event) then

------ start start button module

                        if start_k='1' then
                                start_ran := '1';
                        else
                                start_ran := '0';
                        end if;

-------end start button module;
-------start random data generator

                        if  start_ran = '1' then
                                        key_in_m := '1';
                                if init = '1' and cnt =0 then
                                        data0 <= "0100";
                                        data1 <= "0101";
                                        data2 <= "1111";
                                        data3 <= "1001";
                                        data4 <= "1011";
                                        data5 <= "0011";
                                        data6 <= "0110";
                                        data7 <= "0111";
                                        data8 <= "0000";
                                        data9 <= "0001";
                                        dataA <= "1010";
                                        dataB <= "1110";
                                        dataC <= "1100";
                                        dataD <= "1000";
                                        dataE <= "1101";
                                        dataF <= "0010";
                                        init := '0';
                                        cnt := 1;
                                        strobe_ran  := '1';
                        elsif  cnt=1 and strobe_ran = '1' and mak_ran = 0 then
                                        data0 <= data4;
                                        data4 <= data0;
                                        data1 <= data6;
                                        data6 <= data1;
                                        data2 <= data5;
                                        data5 <= data2;
                                        data3 <= data9;
                                        data9 <= data3;
                                        data7 <= dataE;
                                        dataE <= data7;
                                        data8 <= dataB;
                                        dataB <= data8;
                                        dataA <= dataD;
                                        dataD <= dataA;
                                        dataC <= dataF;
                                        dataF <= dataC;
                                        strobe_ran := '0';
                                        cnt := 2;
                        elsif  cnt=1 and strobe_ran = '1' and mak_ran = 1 then
```

```
                        data0 <= dataF;
                        dataF <= data0;
                        data1 <= data6;
                        data6 <= data1;
                        data2 <= dataA;
                        dataA <= data2;
                        data3 <= data5;
                        data5 <= data3;
                        data4 <= dataD;
                        dataD <= data4;
                        data7 <= dataB;
                        dataB <= data7;
                        data8 <= dataC;
                        dataC <= data8;
                        data9 <= dataE;
                        dataE <= data9;
                        strobe_ran := '0';
                        cnt := 2;
elsif  cnt=1 and strobe_ran = '1' and mak_ran = 2 then
                        data0 <= data7;
                        data7 <= data0;
                        data1 <= data4;
                        data4 <= data1;
                        data2 <= data9;
                        data9 <= data2;
                        data3 <= dataF;
                        dataF <= data3;
                        data5 <= dataC;
                        dataC <= data5;
                        data6 <= dataD;
                        dataD <= data6;
                        data8 <= dataB;
                        dataB <= data8;
                        dataA <= dataE;
                        dataE <= dataA;
                        strobe_ran := '0';
                        cnt := 2;
                elsif cnt = 2 then
                        data0 <= data0 + 1;
                        data1 <= data1 + 1;
                        data2 <= data2 + 1;
                        data3 <= data3 + 1;
                        data4 <= data4 + 1;
                        data5 <= data5 + 1;
                        data6 <= data6 + 1;
                        data7 <= data7 + 1;
                        data8 <= data8 + 1;
                        data9 <= data9 + 1;
                        dataA <= dataA + 1;
                        dataB <= dataB + 1;
                        dataC <= dataC + 1;
                        dataD <= dataD + 1;
                        dataE <= dataE + 1;
                        dataF <= dataF + 1;
                        cnt := 1;
                        strobe_ran := '1';
                end if;
                if mak_ran = 2 then
                        mak_ran := 0;
                else
                        if end_ran_m =3 then
                                end_ran_m := 0;
                                mak_ran := mak_ran + 1;
                        else
                                end_ran_m := end_ran_m + 1;
```

```
                                        end if;
                                end if;
                        else
                                strobe_ran := '0';
                                cnt := 0;
                                init := '1';
                                key_in_m := '0';
                        end if;

------ end random data generator
------ start  key In watchdog & signal return
                if start_k ='0' then
                if up_k='1' or down_k='1' or left_k='1' or right_k = '1' then
                                        key_in := '1';
                                        displ_e <= '1';
                                else
                                        displ_e <= '1';
                                        key_in := '0';
                                end if;
                        end if;

------ end  'key In watchdog & signal return



------start 'data position exchanger from one to one
                        if key_in = '1' then
                                if cnt_ex = 0 then

                                        if up_k='1'  then
                                        if dataF(3 downto 2)="11" then
                                        unable := '1';    -- can't move anywhere
                                                else
                                                temp_up := dataF(3 downto 2);
                                                cnt_ex := cnt_ex + 1;
                                                unable := '0';
                                        end if;
                                        end if;
                                        if down_k='1' then
                                        if dataF(3 downto 2)="00" then
                                        unable := '1';    -- can't move anywhere
                                        else
                                        temp_up := dataF(3 downto 2);
                                        cnt_ex := cnt_ex + 1;
                                        unable := '0';
                                        end if;
                                        end if;

                                        if left_k='1' then
                                                if dataF(1 downto 0)="11"  then
                                        unable := '1';    -- can't move anywhere
                                                else
                                                temp_dn := dataF(1 downto 0);
                                                        cnt_ex := cnt_ex + 1;
                                                        unable := '0';
                                                end if;
                                        end if;
                                        if  right_k='1'  then
                                                if dataF(1 downto 0)="00"  then
                                        unable := '1';    -- can't move anywhere
                                        else
                                                temp_dn := dataF(1 downto 0);
                                                cnt_ex := cnt_ex + 1;
                                                unable := '0';
                                        end if;
```

```
        end if;

elsif cnt_ex=1 and unable ='0' then

        if up_k ='1' then
                temp_up := temp_up + 1;
        end if;

        if down_k ='1' then
                temp_up := temp_up - 1;
        end if;

        if left_k ='1' then
                temp_dn := temp_dn + 1;
        end if;

        if right_k ='1' then
                temp_dn := temp_dn - 1;
        end if;
        cnt_ex := cnt_ex + 1;

elsif cnt_ex=2 then

        if up_k ='1' or down_k = '1' then
temp_all(3 downto 2) := temp_up;
temp_all(1 downto 0) := dataF(1 downto 0);
        end if;

        if left_k = '1' or right_k = '1' then
temp_all(3 downto 2) := dataF(3 downto 2);
temp_all(1 downto 0) := temp_dn;
        end if;
        cnt_ex := cnt_ex + 1;

elsif cnt_ex=3  then
        if temp_all=data0 then
                dataF <= data0;
                data0 <= dataF;
        elsif temp_all = data1 then
                dataF <= data1;
                data1 <= dataF;
        elsif temp_all = data2 then
                dataF <= data2;
                data2 <= dataF;
        elsif temp_all = data3 then
                dataF <= data3;
                data3 <= dataF;
        elsif temp_all = data4 then
                dataF <= data4;
                data4 <= dataF;
        elsif temp_all = data5 then
                dataF <= data5;
                data5 <= dataF;
        elsif temp_all = data6 then
                dataF <= data6;
                data6 <= dataF;
        elsif temp_all = data7 then
                dataF <= data7;
                data7 <= dataF;
        elsif temp_all = data8 then
                dataF <= data8;
                data8 <= dataF;

        elsif temp_all = data9 then
                dataF <= data9;
```

```
                        data9 <= dataF;

                elsif temp_all = dataA then
                        dataF <= dataA;
                        dataA <= dataF;

                elsif temp_all = dataB then
                        dataF <= dataB;
                        dataB <= dataF;

                elsif temp_all = dataC then
                        dataF <= dataC;
                        dataC <= dataF;
                elsif temp_all = dataD then
                        dataF <= dataD;
                        dataD <= dataF;

                elsif temp_all = dataE then
                        dataF <= dataE;
                        dataE <= dataF;

                end if;
                cnt_ex := 4 ;
                ena_veri := '1';
        end if;

else
        ena_veri := '0';
        cnt_ex := 0;
        temp_all := "0000";
        unable := '0';
end if;

if ena_veri='1' then

        if data0="0000" then
                verify(0) <= veri;
        end if;
        if data1="0001" then
                verify(1) <= veri;
        end if;
        if data2="0010" then
                verify(2) <= veri;
        end if;
        if data3="0011" then
                verify(3) <= veri;
        end if;
        if data4="0100" then
                verify(4) <= veri;
        end if;
        if data5="0101" then
                verify(5) <= veri;
        end if;
        if data6="0110" then
                verify(6) <= veri;
        end if;
        if data7="0111" then
                verify(7) <= veri;
        end if;
        if data8="1000" then
                verify(8) <= veri;
        end if;
        if data9="1001" then
                verify(9) <= veri;
        end if;
```

```
                        if dataA="1010" then
                                verify(10) <= veri;
                        end if;
                        if dataB="1011" then
                                verify(11) <= veri;
                        end if;
                        if dataC="1100" then
                                verify(12) <= veri;
                        end if;
                        if dataD="1101" then
                                verify(13) <= veri;
                        end if;
                        if dataE="1110" then
                                verify(14) <= veri;
                        end if;
                        if dataF="1111" then
                                verify(15) <= veri;
                        end if;
                else
                        veri := '1';
                        verify <="0000000000000000";

                end if;
--- display module block
                if displ_e = '1' then
                                if p_cnt = 31 then
                                        r_cnt := r_cnt + 1;
                                        p_cnt := "00000";
                                elsif r_cnt=data0 then
                                        decode <= r_cnt;
                                        data <= "00111111";
                                        p_cnt := p_cnt +1;
                                elsif r_cnt=data1 then
                                        decode <= r_cnt;
                                        data <= "00000110";
                                        p_cnt := p_cnt + 1;
                                elsif r_cnt=data2 then
                                        decode <= r_cnt;
                                        data <= "01011011";
                                        p_cnt := p_cnt +1;
                                elsif r_cnt=data3 then
                                        decode <= r_cnt;
                                        data <= "01001111";
                                        p_cnt := p_cnt +1;
                                elsif r_cnt=data4 then
                                        decode <= r_cnt;
                                        data <= "01100110";
                                        p_cnt := p_cnt +1;
                                elsif r_cnt=data5 then
                                        decode <= r_cnt;
                                        data <= "01101101";
                                        p_cnt := p_cnt +1;
                                elsif r_cnt=data6 then
                                        decode <= r_cnt;
                                        data <= "01111101";
                                        p_cnt := p_cnt +1;
                                elsif r_cnt=data7 then
                                        decode <= r_cnt;
                                        data <= "00100111";
                                        p_cnt := p_cnt +1;
                                elsif r_cnt=data8 then
                                        decode <= r_cnt;
                                        data <= "01111111";
                                        p_cnt := p_cnt +1;
                                elsif r_cnt=data9 then
```

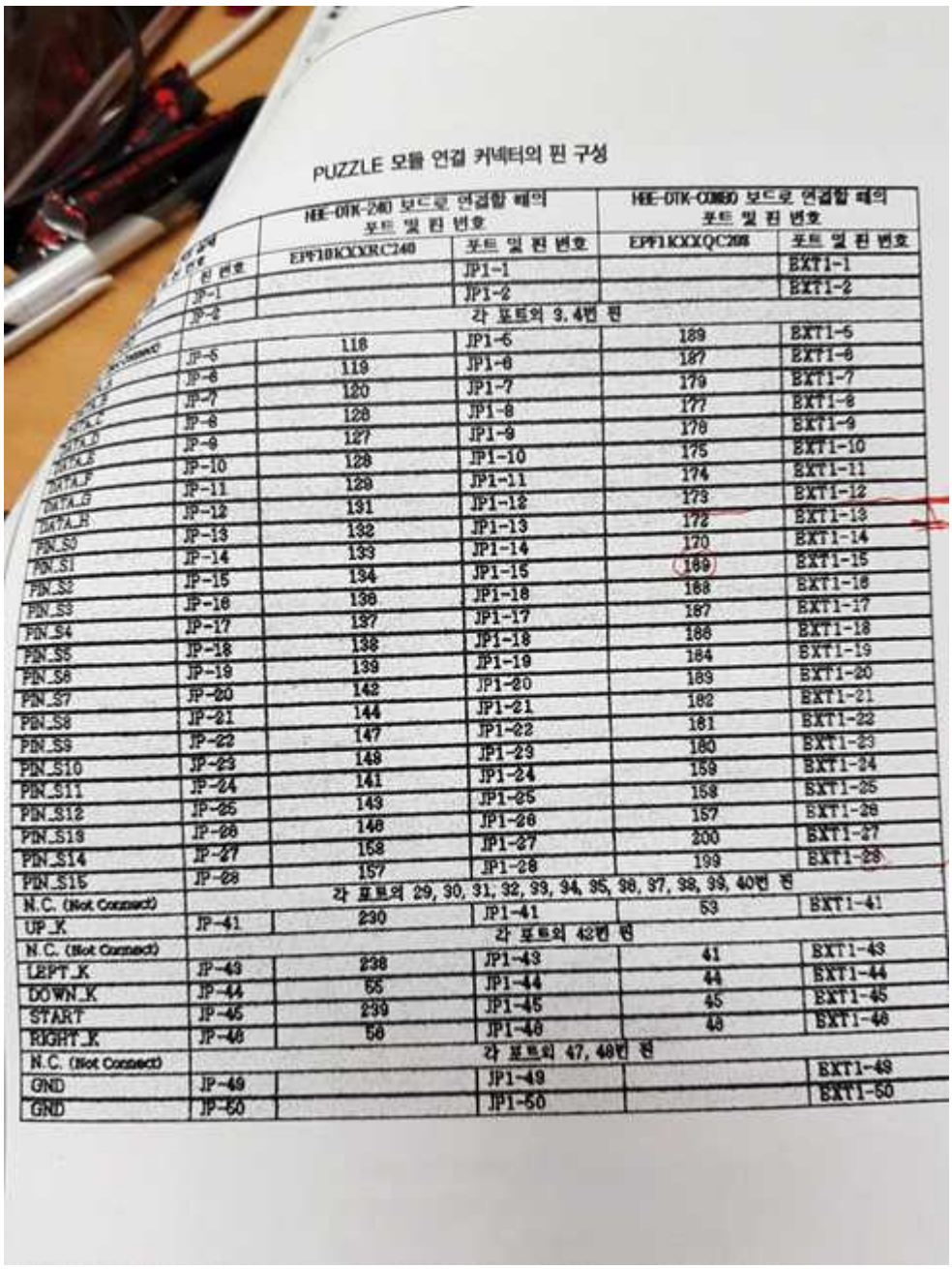
```
                                        decode <= r_cnt;
                                        data <= "01101111";
                                        p_cnt := p_cnt + 1;
                                elsif r_cnt = dataA then
                                        decode <= r_cnt;
                                        data <= "01110111";
                                        p_cnt := p_cnt + 1;
                                elsif r_cnt = dataB then
                                        decode <= r_cnt;
                                        data <= "01111100";
                                        p_cnt := p_cnt + 1;
                                elsif r_cnt = dataC then
                                        decode <= r_cnt;
                                        data <= "00111001";
                                        p_cnt := p_cnt + 1;
                                elsif r_cnt = dataD then
                                        decode <= r_cnt;
                                        data <= "01011110";
                                        p_cnt := p_cnt + 1;
                                elsif r_cnt=dataE then
                                        decode <= r_cnt;
                                        data <= "01111001";
                                        p_cnt := p_cnt + 1;
                                elsif r_cnt = dataF then
                                        decode <= r_cnt;
                                        data <= "00000000";
                                        p_cnt := p_cnt + 1;
                                end if;
                        else
                                p_cnt := "00000";
                                r_cnt := "0000";
                        end if;

------ end display module
                end if;
        end process;
        process(clk)
        begin

        if clk'event and clk ='1' then
                case decode is
                        when "0000" =>
                                pin_s <= "0111111111111111";
                        when "0001" =>
                                pin_s <= "1011111111111111";
                        when "0010" =>
                                pin_s <= "1101111111111111";
                        when "0011" =>
                                pin_s <= "1110111111111111";
                        when "0100" =>
                                pin_s <= "1111011111111111";
                        when "0101" =>
                                pin_s <= "1111101111111111";
                        when "0110" =>
                                pin_s <= "1111110111111111";
                        when "0111" =>
                                pin_s <= "1111111011111111";
                        when "1000" =>
                                pin_s <= "1111111101111111";
                        when "1001" =>
                                pin_s <= "1111111110111111";
                        when "1010" =>
                                pin_s <= "1111111111011111";
                        when "1011" =>
                                pin_s <= "1111111111101111";
```

```
                        when "1100" =>
                                pin_s <= "1111111111110111";
                        when "1101" =>
                                pin_s <= "1111111111111011";
                        when "1110" =>
                                pin_s <= "1111111111111101";
                        when "1111" =>
                                pin_s <= "1111111111111110";
                        when others => null;
                end case;
        end if;
end process;
end sample1;
```

■ 핀 설정



PUZZLE 모듈 연결 커넥터의 핀 구성

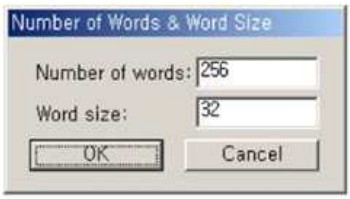
		NEE-01K-240 보드로 연결할 때의 포트 및 핀 번호		NEE-01K-03060 보드로 연결할 때의 포트 및 핀 번호	
	핀 번호	EPF10KXXRC240	포트 및 핀 번호	EPF1KXXQC208	포트 및 핀 번호
	JP-1		JP1-1		EXT1-1
	JP-2		JP1-2		EXT1-2
		각 포트의 3, 4번 핀			
	JP-5	118	JP1-5	189	EXT1-5
	JP-6	119	JP1-6	187	EXT1-6
	JP-7	120	JP1-7	179	EXT1-7
	JP-8	126	JP1-8	177	EXT1-8
	JP-9	127	JP1-9	176	EXT1-9
	JP-10	128	JP1-10	175	EXT1-10
DATA_F	JP-11	129	JP1-11	174	EXT1-11
DATA_G	JP-12	131	JP1-12	173	EXT1-12
DATA_H	JP-13	132	JP1-13	172	EXT1-13
PIN_S0	JP-14	133	JP1-14	170	EXT1-14
PIN_S1	JP-15	134	JP1-15	169	EXT1-15
PIN_S2	JP-16	136	JP1-16	168	EXT1-16
PIN_S3	JP-17	137	JP1-17	167	EXT1-17
PIN_S4	JP-18	138	JP1-18	166	EXT1-18
PIN_S5	JP-19	139	JP1-19	164	EXT1-19
PIN_S6	JP-20	142	JP1-20	163	EXT1-20
PIN_S7	JP-21	144	JP1-21	162	EXT1-21
PIN_S8	JP-22	147	JP1-22	161	EXT1-22
PIN_S9	JP-23	148	JP1-23	160	EXT1-23
PIN_S10	JP-24	141	JP1-24	159	EXT1-24
PIN_S11	JP-25	143	JP1-25	158	EXT1-25
PIN_S12	JP-26	146	JP1-26	157	EXT1-26
PIN_S13	JP-27	158	JP1-27	200	EXT1-27
PIN_S14	JP-28	157	JP1-28	199	EXT1-28
PIN_S15		각 포트의 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40번 핀			
N.C. (Not Connect)	JP-41	230	JP1-41	53	EXT1-41
UP_K		각 포트의 42번 핀			
N.C. (Not Connect)	JP-43	238	JP1-43	41	EXT1-43
LEFT_K	JP-44	55	JP1-44	44	EXT1-44
DOWN_K	JP-45	239	JP1-45	45	EXT1-45
START	JP-46	56	JP1-46	48	EXT1-46
RIGHT_K		각 포트의 47, 48번 핀			
N.C. (Not Connect)	JP-49		JP1-49		EXT1-49
GND	JP-50		JP1-50		EXT1-50
GND					

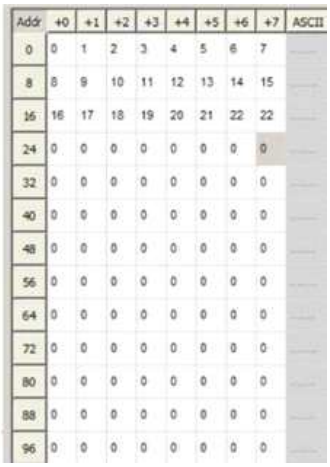
■ 2차 코드 작성

-Rom 데이터를 활용

■ 롬 Mif 파일 생성하는 방법.

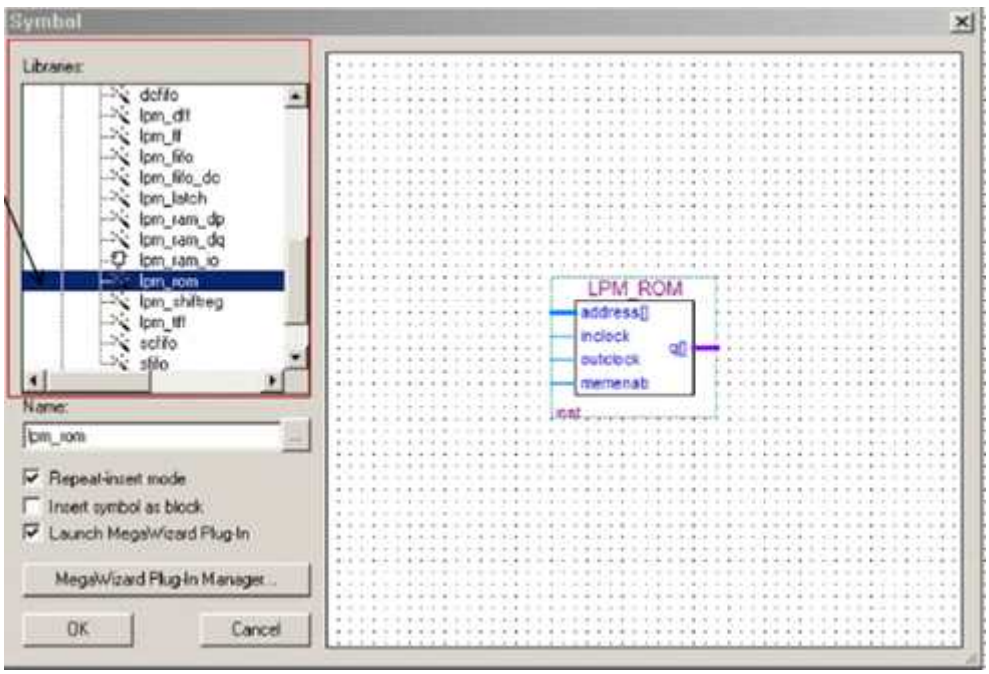
■ 파일을 새로 생성한 후 원하는 사이즈와 워드 크기를 설정하여 메모리에 데이터를 넣는다.



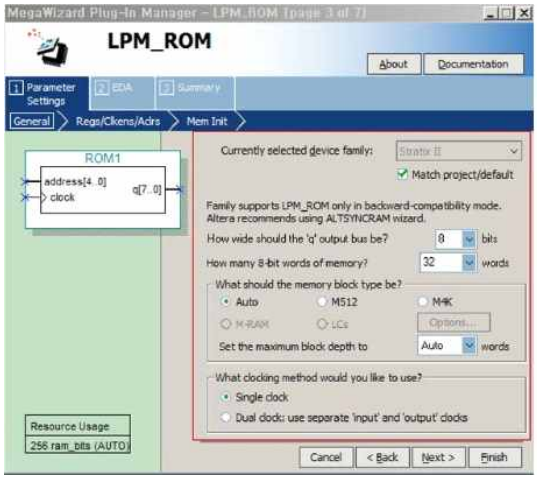


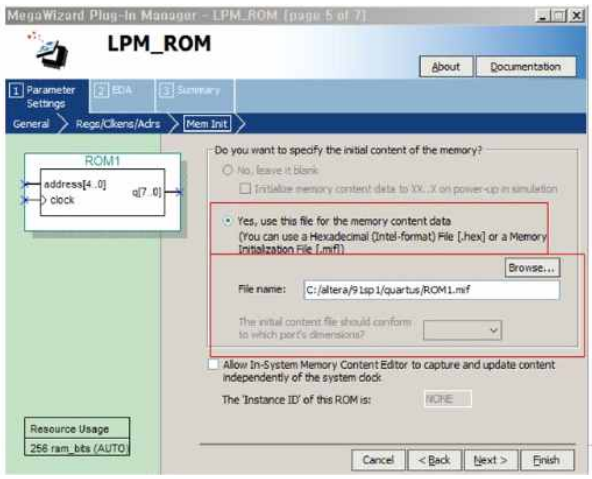
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	0	1	2	3	4	5	6	7	
8	8	9	10	11	12	13	14	15	
16	16	17	18	19	20	21	22	22	
24	0	0	0	0	0	0	0	0	
32	0	0	0	0	0	0	0	0	
40	0	0	0	0	0	0	0	0	
48	0	0	0	0	0	0	0	0	
56	0	0	0	0	0	0	0	0	
64	0	0	0	0	0	0	0	0	
72	0	0	0	0	0	0	0	0	
80	0	0	0	0	0	0	0	0	
88	0	0	0	0	0	0	0	0	
96	0	0	0	0	0	0	0	0	

■ 보드 파일에서 lpm_rom 심볼 파일을 생성한다.



■ MegaWizrd에서 rom을 설정한다.





앞에서 생성한 mif 파일 또는 hex파일을 Rom을 생성할 때 위저드에서 Import 한뒤에 사용해야 한다.

1. 롬 데이터 파일

```
WIDTH = 8;
DEPTH = 1024;

ADDRESS_RADIX = DEC;
DATA_RADIX = HEX;

CONTENT BEGIN

0       :       10      ;
1       :       00      ;
2       :       00      ;
3       :       00      ;
4       :       08      ;
5       :       06      ;
6       :       0C      ;
7       :       04      ;
8       :       00      ;
9       :       0F      ;
10      :       0B      ;
11      :       09      ;
12      :       0D      ;
13      :       05      ;
14      :       01      ;
15      :       02      ;
16      :       03      ;
17      :       0A      ;
18      :       07      ;
19      :       0E      ;
20      :       78      ;
21      :       10      ;
22      :       00      ;
23      :       10      ;
24      :       00      ;
25      :       0B      ;
26      :       03      ;
27      :       04      ;
28      :       06      ;
29      :       0A      ;
30      :       09      ;
31      :       01      ;
32      :       07      ;
33      :       00      ;
34      :       02      ;
35      :       0C      ;
36      :       05      ;
37      :       0F      ;
38      :       0D      ;
39      :       0E      ;
40      :       08      ;
41      :       68      ;
42      :       10      ;
43      :       00      ;
44      :       20      ;
45      :       00      ;
46      :       0E      ;
47      :       0F      ;
48      :       0C      ;
49      :       03      ;
50      :       05      ;
51      :       0D      ;
52      :       0B      ;
53      :       08      ;
54      :       02      ;
```

```
55      :       0A      ;
56      :       01      ;
57      :       04      ;
58      :       00      ;
59      :       09      ;
60      :       06      ;
61      :       07      ;
62      :       58      ;
63      :       10      ;
64      :       00      ;
65      :       30      ;
66      :       00      ;
67      :       01      ;
68      :       0C      ;
69      :       05      ;
70      :       02      ;
71      :       00      ;
72      :       04      ;
73      :       08      ;
74      :       0E      ;
75      :       09      ;
76      :       06      ;
77      :       0A      ;
78      :       0B      ;
79      :       0F      ;
80      :       0D      ;
81      :       07      ;
82      :       03      ;
83      :       48      ;
84      :       10      ;
85      :       00      ;
86      :       40      ;
87      :       00      ;
88      :       05      ;
89      :       08      ;
90      :       0D      ;
91      :       02      ;
92      :       0A      ;
93      :       0C      ;
94      :       0B      ;
95      :       0E      ;
96      :       01      ;
97      :       07      ;
98      :       00      ;
99      :       03      ;
100     :       04      ;
101     :       06      ;
102     :       0F      ;
103     :       09      ;
104     :       38      ;
105     :       10      ;
106     :       00      ;
107     :       50      ;
108     :       00      ;
109     :       08      ;
110     :       05      ;
111     :       01      ;
112     :       06      ;
113     :       0D      ;
114     :       07      ;
115     :       0A      ;
116     :       09      ;
117     :       00      ;
118     :       0B      ;
119     :       0F      ;
```

120 : 03 ;
121 : 04 ;
122 : 0E ;
123 : 0C ;
124 : 02 ;
125 : 28 ;
126 : 10 ;
127 : 00 ;
128 : 60 ;
129 : 00 ;
130 : 0B ;
131 : 01 ;
132 : 0D ;
133 : 00 ;
134 : 0F ;
135 : 08 ;
136 : 0E ;
137 : 07 ;
138 : 02 ;
139 : 06 ;
140 : 03 ;
141 : 0C ;
142 : 0A ;
143 : 04 ;
144 : 05 ;
145 : 09 ;
146 : 18 ;
147 : 10 ;
148 : 00 ;
149 : 70 ;
150 : 00 ;
151 : 0E ;
152 : 0D ;
153 : 05 ;
154 : 00 ;
155 : 0A ;
156 : 06 ;
157 : 0F ;
158 : 09 ;
159 : 02 ;
160 : 0B ;
161 : 0C ;
162 : 03 ;
163 : 01 ;
164 : 08 ;
165 : 04 ;
166 : 07 ;
167 : 08 ;
168 : 10 ;
169 : 00 ;
170 : 80 ;
171 : 00 ;
172 : 02 ;
173 : 0A ;
174 : 0D ;
175 : 0F ;
176 : 05 ;
177 : 06 ;
178 : 0E ;
179 : 08 ;
180 : 00 ;
181 : 0C ;
182 : 0B ;
183 : 01 ;
184 : 04 ;

185 : 09 ;
186 : 03 ;
187 : 07 ;
188 : F8 ;
189 : 10 ;
190 : 00 ;
191 : 90 ;
192 : 00 ;
193 : 05 ;
194 : 06 ;
195 : 0E ;
196 : 0F ;
197 : 0C ;
198 : 07 ;
199 : 00 ;
200 : 01 ;
201 : 0A ;
202 : 02 ;
203 : 08 ;
204 : 03 ;
205 : 0D ;
206 : 0B ;
207 : 09 ;
208 : 04 ;
209 : E8 ;
210 : 10 ;
211 : 00 ;
212 : A0 ;
213 : 00 ;
214 : 08 ;
215 : 03 ;
216 : 0E ;
217 : 0A ;
218 : 00 ;
219 : 01 ;
220 : 05 ;
221 : 02 ;
222 : 0D ;
223 : 06 ;
224 : 0C ;
225 : 0B ;
226 : 04 ;
227 : 0F ;
228 : 09 ;
229 : 07 ;
230 : D8 ;
231 : 10 ;
232 : 00 ;
233 : B0 ;
234 : 00 ;
235 : 0C ;
236 : 0F ;
237 : 06 ;
238 : 0D ;
239 : 04 ;
240 : 09 ;
241 : 08 ;
242 : 01 ;
243 : 05 ;
244 : 0E ;
245 : 0A ;
246 : 02 ;
247 : 07 ;
248 : 03 ;
249 : 00 ;

250 : 0B ;
251 : C8 ;
252 : 10 ;
253 : 00 ;
254 : C0 ;
255 : 00 ;
256 : 0F ;
257 : 0B ;
258 : 0E ;
259 : 0C ;
260 : 00 ;
261 : 01 ;
262 : 02 ;
263 : 04 ;
264 : 0A ;
265 : 06 ;
266 : 03 ;
267 : 08 ;
268 : 09 ;
269 : 0D ;
270 : 07 ;
271 : 05 ;
272 : B8 ;
273 : 10 ;
274 : 00 ;
275 : D0 ;
276 : 00 ;
277 : 02 ;
278 : 08 ;
279 : 06 ;
280 : 0C ;
281 : 09 ;
282 : 07 ;
283 : 03 ;
284 : 04 ;
285 : 0A ;
286 : 0D ;
287 : 05 ;
288 : 0E ;
289 : 0F ;
290 : 0B ;
291 : 00 ;
292 : 01 ;
293 : A8 ;
294 : 10 ;
295 : 00 ;
296 : E0 ;
297 : 00 ;
298 : 05 ;
299 : 04 ;
300 : 0E ;
301 : 0B ;
302 : 02 ;
303 : 06 ;
304 : 07 ;
305 : 03 ;
306 : 0C ;
307 : 0D ;
308 : 0A ;
309 : 01 ;
310 : 00 ;
311 : 0F ;
312 : 08 ;
313 : 09 ;
314 : 98 ;

315 : 10 ;
316 : 00 ;
317 : F0 ;
318 : 00 ;
319 : 09 ;
320 : 01 ;
321 : 07 ;
322 : 0A ;
323 : 0F ;
324 : 04 ;
325 : 03 ;
326 : 0E ;
327 : 02 ;
328 : 08 ;
329 : 0D ;
330 : 06 ;
331 : 0B ;
332 : 00 ;
333 : 05 ;
334 : 0C ;
335 : 88 ;
336 : 10 ;
337 : 01 ;
338 : 00 ;
339 : 00 ;
340 : 0C ;
341 : 0D ;
342 : 0F ;
343 : 09 ;
344 : 0B ;
345 : 02 ;
346 : 05 ;
347 : 03 ;
348 : 06 ;
349 : 0E ;
350 : 0A ;
351 : 07 ;
352 : 00 ;
353 : 01 ;
354 : 08 ;
355 : 04 ;
356 : 77 ;
357 : 10 ;
358 : 01 ;
359 : 10 ;
360 : 00 ;
361 : 0F ;
362 : 09 ;
363 : 07 ;
364 : 04 ;
365 : 02 ;
366 : 0B ;
367 : 06 ;
368 : 05 ;
369 : 0A ;
370 : 08 ;
371 : 0D ;
372 : 03 ;
373 : 01 ;
374 : 00 ;
375 : 0C ;
376 : 0E ;
377 : 67 ;
378 : 10 ;
379 : 01 ;

```
380 : 20 ;
381 : 00 ;
382 : 02 ;
383 : 06 ;
384 : 0F ;
385 : 08 ;
386 : 0E ;
387 : 09 ;
388 : 03 ;
389 : 07 ;
390 : 01 ;
391 : 0B ;
392 : 04 ;
393 : 05 ;
394 : 0A ;
395 : 0C ;
396 : 00 ;
397 : 0D ;
398 : 57 ;
399 : 10 ;
400 : 01 ;
401 : 30 ;
402 : 00 ;
403 : 06 ;
404 : 02 ;
405 : 07 ;
406 : 09 ;
407 : 0F ;
408 : 0C ;
409 : 01 ;
410 : 0D ;
411 : 0A ;
412 : 0E ;
413 : 04 ;
414 : 05 ;
415 : 0B ;
416 : 03 ;
417 : 00 ;
418 : 08 ;
419 : 47 ;
420 : 10 ;
421 : 01 ;
422 : 40 ;
423 : 00 ;
424 : 09 ;
425 : 0F ;
426 : 07 ;
427 : 03 ;
428 : 06 ;
429 : 04 ;
430 : 08 ;
431 : 01 ;
432 : 0E ;
433 : 0B ;
434 : 0C ;
435 : 00 ;
436 : 0A ;
437 : 02 ;
438 : 05 ;
439 : 0D ;
440 : 37 ;
441 : 10 ;
442 : 01 ;
443 : 50 ;
444 : 00 ;
```

```
445     :       0C      ;
446     :       0B      ;
447     :       08      ;
448     :       06      ;
449     :       0E      ;
450     :       0D      ;
451     :       09      ;
452     :       01      ;
453     :       05      ;
454     :       02      ;
455     :       0A      ;
456     :       03      ;
457     :       04      ;
458     :       0F      ;
459     :       00      ;
460     :       07      ;
461     :       27      ;
462     :       10      ;
463     :       01      ;
464     :       60      ;
465     :       00      ;
466     :       0F      ;
467     :       07      ;
468     :       00      ;
469     :       05      ;
470     :       09      ;
471     :       03      ;
472     :       01      ;
473     :       06      ;
474     :       0E      ;
475     :       08      ;
476     :       04      ;
477     :       0C      ;
478     :       0D      ;
479     :       0A      ;
480     :       0B      ;
481     :       02      ;
482     :       17      ;
483     :       10      ;
484     :       01      ;
485     :       70      ;
486     :       00      ;
487     :       03      ;
488     :       04      ;
489     :       08      ;
490     :       05      ;
491     :       0A      ;
492     :       0E      ;
493     :       00      ;
494     :       0D      ;
495     :       0B      ;
496     :       02      ;
497     :       09      ;
498     :       07      ;
499     :       0C      ;
500     :       06      ;
501     :       01      ;
502     :       0F      ;
503     :       07      ;
504     :       10      ;
505     :       01      ;
506     :       80      ;
507     :       00      ;
508     :       06      ;
509     :       00      ;
```

510 : 04 ;
511 : 0E ;
512 : 01 ;
513 : 0B ;
514 : 0F ;
515 : 09 ;
516 : 07 ;
517 : 0D ;
518 : 08 ;
519 : 0C ;
520 : 03 ;
521 : 05 ;
522 : 0A ;
523 : 02 ;
524 : F7 ;
525 : 10 ;
526 : 01 ;
527 : 90 ;
528 : 00 ;
529 : 09 ;
530 : 0D ;
531 : 08 ;
532 : 03 ;
533 : 0F ;
534 : 0C ;
535 : 05 ;
536 : 0A ;
537 : 02 ;
538 : 0B ;
539 : 06 ;
540 : 00 ;
541 : 0E ;
542 : 07 ;
543 : 04 ;
544 : 01 ;
545 : E7 ;
546 : 10 ;
547 : 01 ;
548 : A0 ;
549 : 00 ;
550 : 0C ;
551 : 09 ;
552 : 01 ;
553 : 03 ;
554 : 0E ;
555 : 07 ;
556 : 0F ;
557 : 06 ;
558 : 0A ;
559 : 02 ;
560 : 05 ;
561 : 04 ;
562 : 08 ;
563 : 0B ;
564 : 00 ;
565 : 0D ;
566 : D7 ;
567 : 10 ;
568 : 01 ;
569 : B0 ;
570 : 00 ;
571 : 00 ;
572 : 05 ;
573 : 09 ;
574 : 02 ;

```
575 : 0D ;
576 : 0F ;
577 : 0E ;
578 : 0B ;
579 : 03 ;
580 : 04 ;
581 : 06 ;
582 : 01 ;
583 : 0A ;
584 : 08 ;
585 : 0C ;
586 : 07 ;
587 : C7 ;
588 : 10 ;
589 : 01 ;
590 : C0 ;
591 : 00 ;
592 : 03 ;
593 : 02 ;
594 : 01 ;
595 : 08 ;
596 : 0C ;
597 : 0E ;
598 : 09 ;
599 : 07 ;
600 : 0F ;
601 : 00 ;
602 : 06 ;
603 : 05 ;
604 : 0A ;
605 : 0B ;
606 : 0D ;
607 : 04 ;
608 : B7 ;
609 : 10 ;
610 : 01 ;
611 : D0 ;
612 : 00 ;
613 : 06 ;
614 : 0E ;
615 : 09 ;
616 : 01 ;
617 : 03 ;
618 : 00 ;
619 : 0D ;
620 : 05 ;
621 : 0B ;
622 : 0A ;
623 : 08 ;
624 : 07 ;
625 : 04 ;
626 : 02 ;
627 : 0C ;
628 : 0F ;
629 : A7 ;
630 : 10 ;
631 : 01 ;
632 : E0 ;
633 : 00 ;
634 : 0A ;
635 : 0B ;
636 : 01 ;
637 : 00 ;
638 : 0D ;
639 : 09 ;
```

640 : 0F ;
641 : 07 ;
642 : 05 ;
643 : 04 ;
644 : 0C ;
645 : 0E ;
646 : 02 ;
647 : 08 ;
648 : 03 ;
649 : 06 ;
650 : 97 ;
651 : 10 ;
652 : 01 ;
653 : F0 ;
654 : 00 ;
655 : 0D ;
656 : 07 ;
657 : 09 ;
658 : 0F ;
659 : 08 ;
660 : 00 ;
661 : 01 ;
662 : 0C ;
663 : 0B ;
664 : 03 ;
665 : 06 ;
666 : 0E ;
667 : 05 ;
668 : 04 ;
669 : 02 ;
670 : 0A ;
671 : 87 ;
672 : 10 ;
673 : 02 ;
674 : 00 ;
675 : 00 ;
676 : 00 ;
677 : 03 ;
678 : 02 ;
679 : 0E ;
680 : 07 ;
681 : 0A ;
682 : 01 ;
683 : 0C ;
684 : 08 ;
685 : 0F ;
686 : 0B ;
687 : 06 ;
688 : 09 ;
689 : 05 ;
690 : 0D ;
691 : 04 ;
692 : 76 ;
693 : 10 ;
694 : 02 ;
695 : 10 ;
696 : 00 ;
697 : 03 ;
698 : 00 ;
699 : 0A ;
700 : 0E ;
701 : 0D ;
702 : 02 ;
703 : 01 ;
704 : 0C ;

705 : 04 ;
706 : 0B ;
707 : 06 ;
708 : 07 ;
709 : 08 ;
710 : 0F ;
711 : 09 ;
712 : 05 ;
713 : 66 ;
714 : 10 ;
715 : 02 ;
716 : 20 ;
717 : 00 ;
718 : 07 ;
719 : 0C ;
720 : 02 ;
721 : 0D ;
722 : 08 ;
723 : 04 ;
724 : 0B ;
725 : 00 ;
726 : 01 ;
727 : 06 ;
728 : 0A ;
729 : 0F ;
730 : 03 ;
731 : 0E ;
732 : 09 ;
733 : 05 ;
734 : 56 ;
735 : 10 ;
736 : 02 ;
737 : 30 ;
738 : 00 ;
739 : 0A ;
740 : 09 ;
741 : 0C ;
742 : 02 ;
743 : 0B ;
744 : 03 ;
745 : 07 ;
746 : 06 ;
747 : 05 ;
748 : 01 ;
749 : 04 ;
750 : 0F ;
751 : 00 ;
752 : 08 ;
753 : 0D ;
754 : 0E ;
755 : 46 ;
756 : 10 ;
757 : 02 ;
758 : 40 ;
759 : 00 ;
760 : 0D ;
761 : 05 ;
762 : 02 ;
763 : 0C ;
764 : 0B ;
765 : 04 ;
766 : 08 ;
767 : 0F ;
768 : 07 ;
769 : 09 ;

```
770     :       06      ;
771     :       01      ;
772     :       0E      ;
773     :       00      ;
774     :       03      ;
775     :       0A      ;
776     :       36      ;
777     :       10      ;
778     :       02      ;
779     :       50      ;
780     :       00      ;
781     :       00      ;
782     :       02      ;
783     :       0A      ;
784     :       0B      ;
785     :       07      ;
786     :       08      ;
787     :       04      ;
788     :       01      ;
789     :       0E      ;
790     :       0D      ;
791     :       09      ;
792     :       0C      ;
793     :       0F      ;
794     :       05      ;
795     :       06      ;
796     :       03      ;
797     :       26      ;
798     :       10      ;
799     :       02      ;
800     :       60      ;
801     :       00      ;
802     :       04      ;
803     :       0E      ;
804     :       03      ;
805     :       0A      ;
806     :       02      ;
807     :       0F      ;
808     :       0C      ;
809     :       05      ;
810     :       00      ;
811     :       07      ;
812     :       01      ;
813     :       09      ;
814     :       06      ;
815     :       0D      ;
816     :       08      ;
817     :       0B      ;
818     :       16      ;
819     :       10      ;
820     :       02      ;
821     :       70      ;
822     :       00      ;
823     :       07      ;
824     :       0A      ;
825     :       0B      ;
826     :       0C      ;
827     :       06      ;
828     :       05      ;
829     :       09      ;
830     :       03      ;
831     :       00      ;
832     :       0F      ;
833     :       0E      ;
834     :       08      ;
```

```
835     :       0D      ;
836     :       01      ;
837     :       02      ;
838     :       04      ;
839     :       06      ;
840     :       10      ;
841     :       02      ;
842     :       80      ;
843     :       00      ;
844     :       0A      ;
845     :       07      ;
846     :       03      ;
847     :       09      ;
848     :       0D      ;
849     :       06      ;
850     :       08      ;
851     :       0F      ;
852     :       02      ;
853     :       00      ;
854     :       0B      ;
855     :       0C      ;
856     :       05      ;
857     :       0E      ;
858     :       01      ;
859     :       04      ;
860     :       F6      ;
861     :       10      ;
862     :       02      ;
863     :       90      ;
864     :       00      ;
865     :       0D      ;
866     :       03      ;
867     :       0B      ;
868     :       08      ;
869     :       02      ;
870     :       06      ;
871     :       07      ;
872     :       04      ;
873     :       05      ;
874     :       09      ;
875     :       00      ;
876     :       0E      ;
877     :       0C      ;
878     :       0F      ;
879     :       0A      ;
880     :       01      ;
881     :       E6      ;
882     :       10      ;
883     :       02      ;
884     :       A0      ;
885     :       00      ;
886     :       01      ;
887     :       00      ;
888     :       03      ;
889     :       08      ;
890     :       0C      ;
891     :       0A      ;
892     :       0E      ;
893     :       07      ;
894     :       0D      ;
895     :       04      ;
896     :       06      ;
897     :       05      ;
898     :       09      ;
899     :       0F      ;
```

900 : 0B ;
901 : 02 ;
902 : D6 ;
903 : 10 ;
904 : 02 ;
905 : B0 ;
906 : 00 ;
907 : 04 ;
908 : 0C ;
909 : 0B ;
910 : 07 ;
911 : 01 ;
912 : 09 ;
913 : 08 ;
914 : 0D ;
915 : 03 ;
916 : 06 ;
917 : 0F ;
918 : 00 ;
919 : 0E ;
920 : 02 ;
921 : 0A ;
922 : 05 ;
923 : C6 ;
924 : 10 ;
925 : 02 ;
926 : C0 ;
927 : 00 ;
928 : 07 ;
929 : 08 ;
930 : 04 ;
931 : 06 ;
932 : 01 ;
933 : 0F ;
934 : 09 ;
935 : 0E ;
936 : 0B ;
937 : 02 ;
938 : 0D ;
939 : 0A ;
940 : 0C ;
941 : 05 ;
942 : 03 ;
943 : 00 ;
944 : B6 ;
945 : 10 ;
946 : 02 ;
947 : D0 ;
948 : 00 ;
949 : 0B ;
950 : 05 ;
951 : 0C ;
952 : 06 ;
953 : 0E ;
954 : 08 ;
955 : 0A ;
956 : 07 ;
957 : 04 ;
958 : 09 ;
959 : 02 ;
960 : 03 ;
961 : 01 ;
962 : 0D ;
963 : 00 ;
964 : 0F ;

```
965     :       A6      ;
966     :       10      ;
967     :       02      ;
968     :       E0      ;
969     :       00      ;
970     :       0E      ;
971     :       01      ;
972     :       04      ;
973     :       05      ;
974     :       06      ;
975     :       00      ;
976     :       0B      ;
977     :       0F      ;
978     :       07      ;
979     :       03      ;
980     :       08      ;
981     :       0C      ;
982     :       02      ;
983     :       0A      ;
984     :       0D      ;
985     :       09      ;
986     :       96      ;
987     :       10      ;
988     :       02      ;
989     :       F0      ;
990     :       00      ;
991     :       01      ;
992     :       0E      ;
993     :       0C      ;
994     :       04      ;
995     :       08      ;
996     :       06      ;
997     :       0B      ;
998     :       0D      ;
999     :       03      ;
1000    :       02      ;
1001    :       00      ;
1002    :       0A      ;
1003    :       05      ;
1004    :       09      ;
1005    :       0F      ;
1006    :       07      ;
1007    :       86      ;
1008    :       10      ;
1009    :       03      ;
1010    :       00      ;
1011    :       00      ;
1012    :       04      ;
1013    :       0A      ;
1014    :       0C      ;
1015    :       02      ;
1016    :       01      ;
1017    :       06      ;
1018    :       07      ;
1019    :       09      ;
1020    :       00      ;
1021    :       05      ;
1022    :       08      ;
1023    :       0D      ;

END;
```

■ Puzzle code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity puzzle is
port(
        clk      : in std_logic;  -- 100hz 클럭
        bt_start : in std_logic;  -- 퍼즐을 시작하거나 게임도중 다시 시작하도록 하는 버
튼
        bt_up : in std_logic;  -- 공백을 위로 이동시키는 버튼
        bt_down : in std_logic;  -- 공백을 아래로 이동시키는 버튼
        bt_left : in std_logic; -- 공백을 왼쪽으로 이동시키는 버튼
        bt_right : in std_logic;  -- 공백을 오른쪽으로 이동시키는 버튼

        rom_data : in std_logic_vector (3 downto 0);  -- 외부 롬으로 부터 읽는 숫자
데이터
        rom_address : out std_logic_vector (11 downto 0);  -- 외부 롬에 대한 어드레
스
        rom_cs : out std_logic;  -- 외부 롬의 chip enable 신호

-- 각 7segment에 대한 데이터 값, 단 공백의 표시는 "1111"을 사용한다.
        dig_00, dig_01, dig_02, dig_03 : out std_logic_vector (3 downto 0);
        dig_04, dig_05, dig_06, dig_07 : out std_logic_vector (3 downto 0);
        dig_08, dig_09, dig_10, dig_11 : out std_logic_vector (3 downto 0);
        dig_12, dig_13, dig_14, dig_15 : out std_logic_vector (3 downto 0)
);
end puzzle;

architecture a of puzzle is
-- 각 7segment의 값을 처리하기 위한 내부 변수
signal  data_00 : std_logic_vector(3 downto 0);
signal  data_01 : std_logic_vector(3 downto 0);
signal  data_02 : std_logic_vector(3 downto 0);
signal  data_03 : std_logic_vector(3 downto 0);
signal  data_04 : std_logic_vector(3 downto 0);
signal  data_05 : std_logic_vector(3 downto 0);
signal  data_06 : std_logic_vector(3 downto 0);
signal  data_07 : std_logic_vector(3 downto 0);
signal  data_08 : std_logic_vector(3 downto 0);
signal  data_09 : std_logic_vector(3 downto 0);
signal  data_10 : std_logic_vector(3 downto 0);
signal  data_11 : std_logic_vector(3 downto 0);
signal  data_12 : std_logic_vector(3 downto 0);
signal  data_13 : std_logic_vector(3 downto 0);
signal  data_14 : std_logic_vector(3 downto 0);
signal  data_15 : std_logic_vector(3 downto 0);

-- puzzle의 동작 상태를 나타내기 위한 state (idle : 초기상태 또는 종료상태,
-- address_scan : 롬에서 데이터를 읽기 위한 address를 생성하는 상태,
-- load_data : address_scan state에서 만들어진 address를 가지고 롬으로 부터 데이터
로드,
-- play : 실제 동작 상태
type status is (idle, address_scan, load_data, play);
signal puzzle_status : status;

-- 퍼즐의 모든 숫자가 맞추어 질 경우 '1'의 값을 가진다.
signal play_done : std_logic;

-- 롬에 저장된 256가지의 데이터 어드레스
signal rom_address_msb : integer range 255 downto 0;
-- 256가지의 데이터에 대한 각각의 숫자 어드레스
signal rom_address_lsb : integer range 15 downto 0;
```

```
-- 버튼 누름 감지를 위한 변수, 값이 "01"일 경우 누름으로 감지
signal start, up, down, left, right : std_logic_vector (1 downto 0);

-- 공백의 위치에 대한 변수
signal position : integer range 15 downto 0;

-- 256가지의 데이터에 대한 각각의 숫자 어드레스
signal address : integer range 15 downto 0;

-- 버튼의 누름을 감지하기 위한 변수값을 생성하는 구문
-- 예를 들어 start의 값이 "01"인경우 키가 눌려짐을 의미하고
-- "10"인 경우 눌러진 상태에서 떼어진 상태로 바뀜을 의미한다.
begin
process (clk)
begin
	if clk'event and clk = '1' then
		start(0) <= bt_start;
		start(1) <= start(0);
		up(0) <= bt_up;
		up(1) <= up(0);
		down(0) <= bt_down;
		down(1) <= down(0);
		left(0) <= bt_left;
		left(1) <= left(0);
		right(0) <= bt_right;
		right(1) <= right(0);
	end if;
end process;

-- 퍼즐의 동작 상태를 제어하기 위한 구문
process (clk)
begin
	if clk'event and clk = '1' then
		case puzzle_status is
			when idle =>
-- start 버튼이 눌러지면
				if start = "01" then
					puzzle_status <= address_scan;
				end if;
			when address_scan =>
-- start 버튼이 떼어지면
				if start = "10" then
					puzzle_status <= load_data;
				end if;
			when load_data =>
-- address가 15가 되면(rom으로 부터 data load가 완료되면)
				if address = 15 then
					puzzle_status <= play;
				end if;
			when play =>
-- play 도중 start가 눌러지면 게임 종료
				if start = "01" then
					puzzle_status <= idle;
-- 모든 숫자가 맞추어 지면 게임 종료
				elsif play_done = '1' then
					puzzle_status <= idle;
				end if;
			when others =>
				puzzle_status <= idle;
		end case;
	end if;
end process;

-- 롬으로 부터 데이터가 읽어지는 load_data 상태에서
```

```
rom_cs <= '0' when puzzle_status = load_data else '1';

-- address_scan state는 start키가 눌려지는 동안 상태가 유지되는데
이때 address를 무한 반복으로 카운트 하고 state가 load_data로 전환
될때 카운트된 address를 넘겨주게 된다. 이렇게 하여 준비된 256개의
데이터중에서 임의의 데이터가 선택되도록 한다.
process (clk)
begin
        if clk'event and clk = '1' then
                if puzzle_status = address_scan then
                        if rom_address_msb = 255 then
                                rom_address_msb <= 0;
                        else
                                rom_address_msb <= rom_address_msb + 1;
                        end if;
                elsif puzzle_status = load_data then
                        rom_address(11          downto          4)          <=
conv_std_logic_vector(rom_address_msb, 8);
                end if;
        end if;
end process;

-- 롬으로 선택된 숫자 데이터를 읽기위한 하위 어드레스를 생성하기 위한 구문
process (clk)
begin
        if clk'event and clk = '1' then
                if puzzle_status = load_data then
                        rom_address_lsb <= rom_address_lsb + 1;
                else
                        rom_address_lsb <= 0;
                end if;
                address <= rom_address_lsb;
        end if;
-- conv_std_logic_vector는 integer의 형태를 std_logic_vector로 변환하기 위한 function
        rom_address(3 downto 0) <= conv_std_logic_vector(rom_address_lsb,4);

end process;

process (clk)
begin
        if clk'event and clk = '1' then
-- 롬으로 부터 읽어진 데이터에서 공백("1111")위치를 찾는다.
                if puzzle_status = load_data then
                                if rom_data = "1111" then
                                        position <= address;
                                end if;
                elsif puzzle_status = play then
-- 방향키가 눌러짐에 따라 공백의 위치를 변경한다.
                        if up = "01" then
-- 공백의 위치가 가장 위의 줄에 위치할 경우 더이상 위로의 이동이 불가 하므로
-- 위치 이동이 없으며 이외의 조건에서는 이전위치에서 4를 빼서 다음 위치를 지정한다.
                                if position >= 4 then
                                        position <= position - 4;
                                end if;
                        elsif down = "01" then
-- 공백의 위치가 가장 아래의 줄에 위치할 경우 더이상 아래로의 이동이 불가 하므로
-- 위치 이동이 없으며 이외의 조건에서는 이전위치에서 4를 더해서 다음 위치를 지정한다.
                                if position <= 11 then
                                        position <= position + 4;
                                end if;
                        elsif left = "01" then
-- 공백의 위치가 가장 왼쪽의 줄에 위치할 경우 더이상 왼쪽으로의 이동이 불가 하므로
-- 위치 이동이 없으며 이외의 조건에서는 이전위치에서 1를 빼서 다음 위치를 지정한다.
                                if position /= 0 and position /= 4 and position /= 8
and position /= 12 then
```

```
                                        position <= position - 1;
                                end if;
                        elsif right = "01" then
-- 공백의 위치가 가장 오른쪽의 줄에 위치할 경우 더이상 오른쪽으로의 이동이 불가 하므로
-- 위치 이동이 없으며 이외의 조건에서는 이전위치에서 1를 더해서 다음 위치를 지정한다.
                                if position /= 3 and position /= 7 and position /=
11 and position /= 15 then
                                        position <= position + 1;
                                end if;
                        end if;
                end if;
        end if;
end process;

process (clk)
begin
        if clk'event and clk = '1' then
-- 롬으로 부터 숫자 데이터를 읽어오는 구문
                if puzzle_status = load_data then
                        case address is
                                when 00 => data_00 <= rom_data;
                                when 01 => data_01 <= rom_data;
                                when 02 => data_02 <= rom_data;
                                when 03 => data_03 <= rom_data;
                                when 04 => data_04 <= rom_data;
                                when 05 => data_05 <= rom_data;
                                when 06 => data_06 <= rom_data;
                                when 07 => data_07 <= rom_data;
                                when 08 => data_08 <= rom_data;
                                when 09 => data_09 <= rom_data;
                                when 10 => data_10 <= rom_data;
                                when 11 => data_11 <= rom_data;
                                when 12 => data_12 <= rom_data;
                                when 13 => data_13 <= rom_data;
                                when 14 => data_14 <= rom_data;
                                when 15 => data_15 <= rom_data;
                                when others => null;
                        end case;
-- 각 키의 눌러짐과 현재 공백의 위치를 조합하여 다음 데이터의 값을 변경한다.
                elsif puzzle_status = play then
-- 현재 공백의 위치가 두번째 줄의 첫번째에 위치해 있을때 up 버튼이 눌러진 경우나
-- 공백의 위치가 첫번째 줄의 첫번째에 위치한 상태에서 down 버튼이 눌러진 경우
-- 첫째눌의 첫번째 값과 두번째 줄의 첫번째의 값을 교환한다.
                        if (up = "01" and position = 04) or (down = "01" and position
= 00) then
                                data_00 <= data_04;
                                data_04 <= data_00;
                        elsif (left = "01" and position = 01) or (right = "01" and
position = 00) then
                                data_00 <= data_01;
                                data_01 <= data_00;
                        end if;

                        if (up = "01" and position = 05) or (down = "01" and position
= 01) then
                                data_01 <= data_05;
                                data_05 <= data_01;
                        elsif (left = "01" and position = 02) or (right = "01" and
position = 01) then
                                data_01 <= data_02;
                                data_02 <= data_01;
                        end if;

                        if (up = "01" and position = 06) or (down = "01" and position
= 02) then
```

```
                data_02 <= data_06;
                data_06 <= data_02;
            elsif (left = "01" and position = 03) or (right = "01" and
position = 02) then
                data_02 <= data_03;
                data_03 <= data_02;
            end if;

            if (up = "01" and position = 07) or (down = "01" and position
= 03) then
                data_03 <= data_07;
                data_07 <= data_03;
            end if;

            if (up = "01" and position = 08) or (down = "01" and position
= 04) then
                data_04 <= data_08;
                data_08 <= data_04;
            elsif (left = "01" and position = 05) or (right = "01" and
position = 04) then
                data_04 <= data_05;
                data_05 <= data_04;
            end if;

            if (up = "01" and position = 09) or (down = "01" and position
= 05) then
                data_05 <= data_09;
                data_09 <= data_05;
            elsif (left = "01" and position = 06) or (right = "01" and
position = 05) then
                data_05 <= data_06;
                data_06 <= data_05;
            end if;

            if (up = "01" and position = 10) or (down = "01" and position
= 06) then
                data_06 <= data_10;
                data_10 <= data_06;
            elsif (left = "01" and position = 07) or (right = "01" and
position = 06) then
                data_06 <= data_07;
                data_07 <= data_06;
            end if;

            if (up = "01" and position = 11) or (down = "01" and position
= 07) then
                data_07 <= data_11;
                data_11 <= data_07;
            end if;

            if (up = "01" and position = 12) or (down = "01" and position
= 08) then
                data_08 <= data_12;
                data_12 <= data_08;
            elsif (left = "01" and position = 09) or (right = "01" and
position = 08) then
                data_08 <= data_09;
                data_09 <= data_08;
            end if;

            if (up = "01" and position = 13) or (down = "01" and position
= 09) then
                data_09 <= data_13;
                data_13 <= data_09;
            elsif (left = "01" and position = 10) or (right = "01" and
```

```
position = 09) then
                                data_09 <= data_10;
                                data_10 <= data_09;
                        end if;

                        if (up = "01" and position = 14) or (down = "01" and position
= 10) then
                                data_10 <= data_14;
                                data_14 <= data_10;
                        elsif (left = "01" and position = 11) or (right = "01" and
position = 10) then
                                data_10 <= data_11;
                                data_11 <= data_10;
                        end if;

                        if (up = "01" and position = 15) or (down = "01" and position
= 11) then
                                data_11 <= data_15;
                                data_15 <= data_11;
                        end if;

                        if (left = "01" and position = 13) or (right = "01" and
position = 12) then
                                data_12 <= data_13;
                                data_13 <= data_12;
                        end if;

                        if (left = "01" and position = 14) or (right = "01" and
position = 13) then
                                data_13 <= data_14;
                                data_14 <= data_13;
                        end if;

                        if (left = "01" and position = 15) or (right = "01" and
position = 14) then
                                data_14 <= data_15;
                                data_15 <= data_14;
                        end if;
                end if;
        end if;
end process;

process (clk)
begin
-- 숫자들이 정확한 자리에 위치된 것을 체크하기 위한 구문
        if clk'event and clk = '1' then
                if data_00 = "0000" and data_01 = "0001" and data_02 = "0010" and
                data_03 = "0011" and data_04 = "0100" and data_05 = "0101" and
                data_06 = "0110" and data_07 = "0111" and data_08 = "1000" and
                data_09 = "1001" and data_10 = "1010" and data_11 = "1011" and
                data_12 = "1100" and data_13 = "1101" and data_14 = "1110" and
                data_15 = "1111"  then
                        play_done <= '1';
                else
                        play_done <= '0';
                end if;
        end if;
end process;

dig_00 <= data_00;
dig_01 <= data_01;
dig_02 <= data_02;
dig_03 <= data_03;
dig_04 <= data_04;
dig_05 <= data_05;
```

```
dig_06 <= data_06;
dig_07 <= data_07;
dig_08 <= data_08;
dig_09 <= data_09;
dig_10 <= data_10;
dig_11 <= data_11;
dig_12 <= data_12;
dig_13 <= data_13;
dig_14 <= data_14;
dig_15 <= data_15;
end a;
```

■ 7 세그먼트를 컨트롤 하기 위한 코드

-- 일반적인 방법으로 7segment를 디스플레이 할 경우 7segment의 숫자 만큼의 출력이 필요하게 된다. 예를 들어 8개의 7segment를 사용할 경우 8bit 출력 x 8개 = 64 출력이 필요하게 된다. 이를 변형하여 출력의 수를 줄이기 위한 방법이 scanning을 이용한 방법이다. 7segment의 값은 공통으로 사용하고 출력할 위치를 선택하는 common이라는 출력을 만들어 이를 10khz 정도의 클럭을 이용하여 빠르게 디스플레이하면 전체가 디스플레이 된 것으로 보이게 된다.

```
library ieee;
use ieee.std_logic_1164.all;

entity seg_module is
port (
        clk : in std_logic;  -- 10khz 클럭
-- 7segment 데이터 입력
        dig_00 : in std_logic_vector (3 downto 0);
        dig_01 : in std_logic_vector (3 downto 0);
        dig_02 : in std_logic_vector (3 downto 0);
        dig_03 : in std_logic_vector (3 downto 0);
        dig_04 : in std_logic_vector (3 downto 0);
        dig_05 : in std_logic_vector (3 downto 0);
        dig_06 : in std_logic_vector (3 downto 0);
        dig_07 : in std_logic_vector (3 downto 0);
        dig_08 : in std_logic_vector (3 downto 0);
        dig_09 : in std_logic_vector (3 downto 0);
        dig_10 : in std_logic_vector (3 downto 0);
        dig_11 : in std_logic_vector (3 downto 0);
        dig_12 : in std_logic_vector (3 downto 0);
        dig_13 : in std_logic_vector (3 downto 0);
        dig_14 : in std_logic_vector (3 downto 0);
        dig_15 : in std_logic_vector (3 downto 0);

-- 7segment 출력
        seg_out : out std_logic_vector (7 downto 0);
-- 7segment 디스플레이 위치 지정 출력
        common : out std_logic_vector (15 downto 0)
);
end seg_module;

architecture a of seg_module is

component hex2seg
port (
        hex : in std_logic_vector (3 downto 0);
        segment : out std_logic_vector (7 downto 0)
);
end component;
-- 16개의 7segment를 사용하므로 0 ~ 15까지 카운트 하기 위해 사용.
signal cnt : integer range 15 downto 0;
signal hexa : std_logic_vector (3 downto 0);
begin

u0 : hex2seg
port map ( hexa, seg_out);

process( clk )
begin
        if clk'event and clk = '1' then
                cnt <= cnt + 1;
                case cnt is
-- common이 '1'일 경우 7segment를 디스플레이 하고 '0'일 경우
-- 디스플레이를 하지 않게 된다.
                        when 15 =>
                                common <= "0111111111111111";
                                hexa <= dig_15;
```

```
                        when 14 =>
                                common <= "1011111111111111";
                                hexa <= dig_14;
                        when 13 =>
                                common <= "1101111111111111";
                                hexa <= dig_13;
                        when 12 =>
                                common <= "1110111111111111";
                                hexa <= dig_12;
                        when 11 =>
                                common <= "1111011111111111";
                                hexa <= dig_11;
                        when 10 =>
                                common <= "1111101111111111";
                                hexa <= dig_10;
                        when 09 =>
                                common <= "1111110111111111";
                                hexa <= dig_09;
                        when 08 =>
                                common <= "1111111011111111";
                                hexa <= dig_08;
                        when 07 =>
                                common <= "1111111101111111";
                                hexa <= dig_07;
                        when 06 =>
                                common <= "1111111110111111";
                                hexa <= dig_06;
                        when 05 =>
                                common <= "1111111111011111";
                                hexa <= dig_05;
                        when 04 =>
                                common <= "1111111111101111";
                                hexa <= dig_04;
                        when 03 =>
                                common <= "1111111111110111";
                                hexa <= dig_03;
                        when 02 =>
                                common <= "1111111111111011";
                                hexa <= dig_02;
                        when 01 =>
                                common <= "1111111111111101";
                                hexa <= dig_01;
                        when 00 =>
                                common <= "1111111111111110";
                                hexa <= dig_00;
                        when others =>
                                null;
                end case;
        end if;
end process;
end a;
```

■ seg_puzzle 은 시스템을 동작시키는 핵심 코드다
따라서 architecture 내의 puzzle 코드와 seg_module 코드가 필수 적으로 프로젝트 안에 포함되어야 하며 ROM을 생성해야 컴파일에 문제가 없다.

```
library ieee;
use ieee.std_logic_1164.all;

entity seg_puzzle is
port (
        clk      : in std_logic; -- 10KHz 클럭
        bt_start : in std_logic; -- 게임 시작및 종료 버튼

-- 방향 버튼
        bt_up : in std_logic;
        bt_down : in std_logic;
        bt_left : in std_logic;
        bt_right : in std_logic;

-- 외부 롬에 대한 데이터 입력
        rom_data : in std_logic_vector (3 downto 0);
-- 외부 롬에 대한 어드레스 출력
        rom_address : out std_logic_vector (11 downto 0);
-- 외부 롬에 대한 chip select 출력
        rom_cs : out std_logic;

-- 7segment 데이터
        seg_out : out std_logic_vector (7 downto 0);
-- 7segment 선택 출력
        common : out std_logic_vector (15 downto 0)
);
end seg_puzzle;

architecture a of seg_puzzle is

component puzzle
port(
        clk      : in std_logic;
        bt_start : in std_logic;
        bt_up : in std_logic;
        bt_down : in std_logic;
        bt_left : in std_logic;
        bt_right : in std_logic;

        rom_data : in std_logic_vector (3 downto 0);
        rom_address : out std_logic_vector (11 downto 0);
        rom_cs : out std_logic;

        dig_00, dig_01, dig_02, dig_03 : out std_logic_vector (3 downto 0);
        dig_04, dig_05, dig_06, dig_07 : out std_logic_vector (3 downto 0);
        dig_08, dig_09, dig_10, dig_11 : out std_logic_vector (3 downto 0);
        dig_12, dig_13, dig_14, dig_15 : out std_logic_vector (3 downto 0)
);
end component;

component seg_module
port (
        clk : in std_logic;
        dig_00 : in std_logic_vector (3 downto 0);
        dig_01 : in std_logic_vector (3 downto 0);
        dig_02 : in std_logic_vector (3 downto 0);
        dig_03 : in std_logic_vector (3 downto 0);
        dig_04 : in std_logic_vector (3 downto 0);
        dig_05 : in std_logic_vector (3 downto 0);
        dig_06 : in std_logic_vector (3 downto 0);
```

```
        dig_07 : in std_logic_vector (3 downto 0);
        dig_08 : in std_logic_vector (3 downto 0);
        dig_09 : in std_logic_vector (3 downto 0);
        dig_10 : in std_logic_vector (3 downto 0);
        dig_11 : in std_logic_vector (3 downto 0);
        dig_12 : in std_logic_vector (3 downto 0);
        dig_13 : in std_logic_vector (3 downto 0);
        dig_14 : in std_logic_vector (3 downto 0);
        dig_15 : in std_logic_vector (3 downto 0);

        seg_out : out std_logic_vector (7 downto 0);
        common : out std_logic_vector (15 downto 0)
);
end component;

signal  dig_00, dig_01, dig_02, dig_03 : std_logic_vector (3 downto 0);
signal  dig_04, dig_05, dig_06, dig_07 : std_logic_vector (3 downto 0);
signal  dig_08, dig_09, dig_10, dig_11 : std_logic_vector (3 downto 0);
signal  dig_12, dig_13, dig_14, dig_15 : std_logic_vector (3 downto 0);

begin

u0 : puzzle
port map( clk, bt_start, bt_up, bt_down, bt_left, bt_right,
                         rom_data, rom_address, rom_cs,
                         dig_00, dig_01, dig_02, dig_03,
                         dig_04, dig_05, dig_06, dig_07,
                         dig_08, dig_09, dig_10, dig_11,
                         dig_12, dig_13, dig_14, dig_15
);

u1 : seg_module
port map(
        clk,
        dig_00, dig_01, dig_02, dig_03,
        dig_04, dig_05, dig_06, dig_07,
        dig_08, dig_09, dig_10, dig_11,
        dig_12, dig_13, dig_14, dig_15,

        seg_out,
        common
);

end a;
```