VHDL 및 실습

계층구조 설계 및 디지털 시계설계

과목	VHDL 및 실습
학과	전자공학과
학번	2011144024
이름	유대성 (오전)
제출일	
담당교수	최종성 교수님

1. 주제 및 배경이론

VHDL 의 계층구조를 이해하고 디지털 시계를 설계해보자.

계층구조(Hierarchy Structure)

프로젝트의 크기가 커지고 복잡해지면 변경된 사항이 다른 사람의 작업 내역에 반영되지 않거나 내용이 너무 많아 내 수정내용을 찾는 데 오랜 시간이 걸리는 등의 문제가 발생하게 된다. 프로젝트를 세분화하여 각각의 파일을 담당자가 직접 수정하여 나중에 취합하는 방식으로 해결할 수 있는데 이를 계층구조라고 한다. 세분화, 취합하는 과정에 따라 Top-Down, Bottom-Up 으로 구분할 수 있다.

Top Entity(Top Design)

계층 구조를 최종적으로 취합하는 과정이다. 각각의 디자인이 최종적으로 합쳐지므로 Top Design 이라고도 부른다. Top Entity 에선 다음과 같은 역할을 수행한다.

- 최종 입출력 포트를 정의한다.
- 각 디자인의 입출력 포트를 연결하는 등의 취합과정을 진행한다.

Altera Quartus II 에서는 프로젝트 생성을 Schematic / HDL 로 나누어 진행할 수 있지만 Schematic 은 다른 플랫폼으로의 변경에 어려움이 있다.

디지털 시계(Digital Watch)

〈 주요 기능 〉

- 1 초마다 시간을 카운트 한다.
- 시간을 조정할 수 있는 기능이 있다.

〈 구현 방식 〉

- 1 초라는 단위를 만들기 위해 FPGA 내의 Clock 을 이용한다.
- 각 초, 분, 시의 상태를 저장할 카운터가 필요하다.
- 초, 분, 시를 표현하기 위해 FND decoder 를 사용한다.
- 초, 분, 시를 변경하기 위한 외부 signal 을 받기 위해 MUX 를 추가로 연결한다.

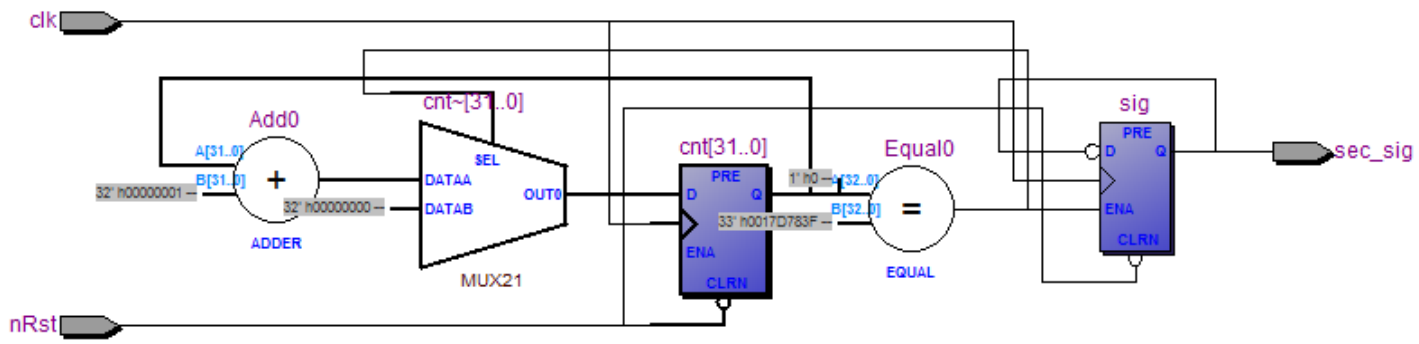
2. Component 소개

디지털 시계 설계를 위한 하위 계층 부품에 대한 설명입니다.

1 초 생성기

FPGA 내부의 클럭을 사용해 '초'라는 단위를 발생시켜야 한다. 50MHz 클럭을 사용해 이를 구현할 수 있다.

```
1  library ieee;
2    use ieee.std_logic_1164.all;
3     use ieee.std_logic_arith.all;
4     use ieee.std_logic_unsigned.all;
5
6
7  entity sec_gen is
8     port(
9        nRst        : in std_logic;
10       clk         : in std_logic;
11       sec_sig     : out std_logic
12    );
13 end sec_gen;
14
15 architecture BEH of sec_gen is
16
17    signal cnt : std_logic_vector(31 downto 0);
18    signal sig : std_logic;
19
20    begin
21    sec_sig <=sig;
22    -- begin ~ end 문장
23    -- initial, if, case, always 등을 사용할 때 블록을 지정할때 사용한다.
24    -- C 언어에서 {... }와 같은 개념이다.
25
26    process(nRst,clk)
27       begin
28          if(nRst = '0') then
29             sig <= '0';
30             cnt <=(others =>'0');
31          elsif rising_edge(clk) then
32             -- 입력클럭 50MHz 이므로 50% 듀티사이클 펄스는 아래와 같이 구할 수 있다.
33             if(cnt = 24999999) then
34                cnt <= (others =>'0');
35                sig <= not sig;
36             else
37                cnt <= cnt + 1;
38             end if;
39          end if;
40    end process;
41
42 end BEH;
43                                                    Colored by Color Scripter cs
```

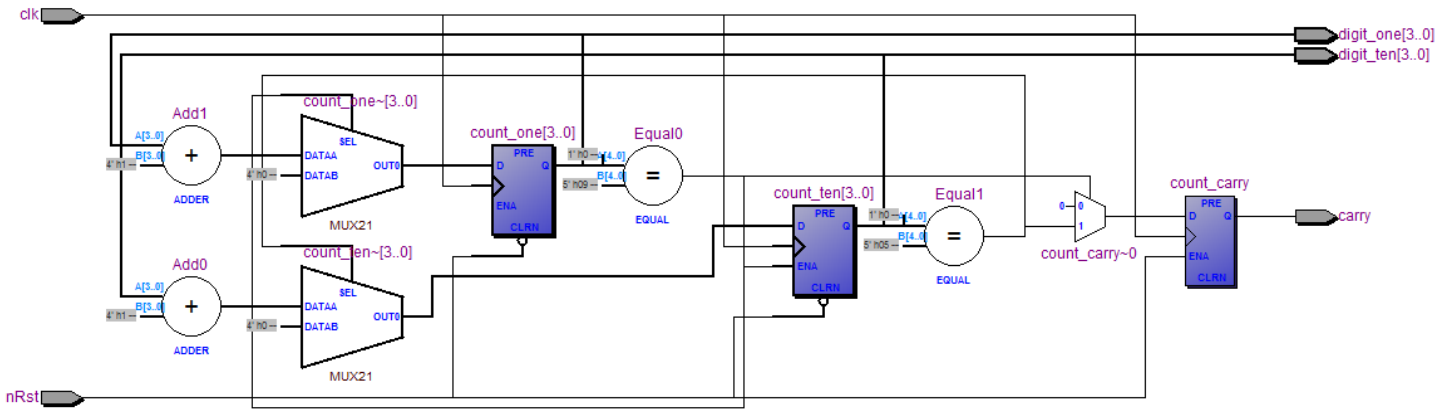


60 진 카운터

시간을 카운팅할 때 초, 분은 60 진수로 카운팅 되므로 60 진 카운터로 설계하여야 한다.

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


entity counter_60 is
    port(
        -- 60 진 카운터에서는 sec, min 에 대한 각각의 클럭으로 카운팅이 일어나며 reset 을 통해 0 으로 리셋된다.
        -- 또한 FND 디코더에 연결된 4bit 의 출력을 내보낸다. 60 이 카운팅 되면 다음 단위로 carry 가 발생한다.
        -- 이를 통해 외부 입출력을 정의하면 다음과 같다.
        nRst            : in std_logic;
      clk             : in std_logic;
        digit_one     : out std_logic_vector(3 downto 0);
        digit_ten     : out std_logic_vector(3 downto 0);
        carry           : out std_logic
    );
    end counter_60;

architecture BEH of counter_60 is
    -- 다음은 내부 시그널이다. 위 port 는 외부 시그널을 의미하기 때문에 실질적 동작은 아래 기술하게 된다
    signal count_one        : std_logic_vector(3 downto 0);
    signal count_ten        : std_logic_vector(3 downto 0);
    signal count_carry    : std_logic;

    begin

    process(nRst, clk)
        begin
        -- reset 동작은 active low 라는 점에 유의하자.
        if(nRst = '0') then
```

```
32          count_one <= (others => '0');
33          count_ten <= (others => '0');
34
35      -- clk'EVENT AND clk='1'은 rising edge 를 clk'EVENT AND clk='0'을 falling edge 를 의미
36      -- RISING_EDGE(clk), FALLING_EDGE(clk)로 직접 작성하는 방법도 있다.
37      elsif RISING_EDGE(clk) then
38          if(count_carry = '1') then
39              count_carry <= '0';
40          end if;
41         if(count_one = 9) then
42              -- (others => '0')으로 대체하여 쓸 수 있다.
43              count_one <= "0000";
44              if(count_ten = 5 ) then
45                  -- (others => '0')으로 대체하여 쓸 수 있다.
46                  count_ten <= "0000";
47                  count_carry <= '1';
48              else
49                  count_ten <= count_ten + 1;
50              end if;
51          else
52              count_one <= count_one + 1;
53          end if;
54      end if;
55
56      carry <= count_carry;
57      digit_one <= count_one;
58      digit_ten <= count_ten;
59    end process;
60  end BEH;
61
62                                                    Colored by Color Scripter
```

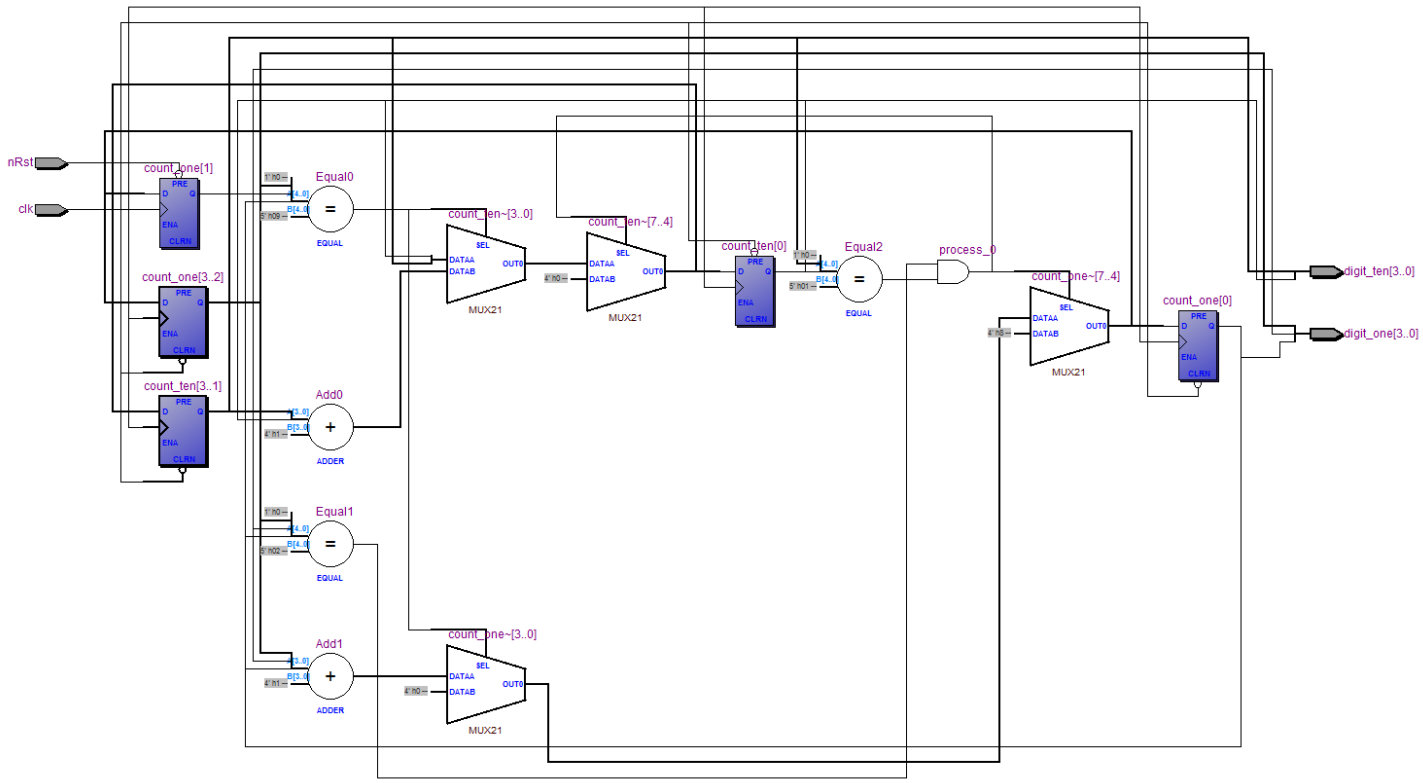


12 진 카운터

시간을 카운팅할 때 시간은 12 진수로 카운팅 되므로 12 진 카운터로 설계하여야 한다.

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

entity counter_12 is
    port(
        -- 12 진 카운터에서는 hour 에 대한 각각의 클럭으로 카운팅이 일어나며 reset 을 통해 12 로 리셋된다.
        -- 또한 FND 디코더에 연결된 4bit 의 출력을 내보낸다. 12 가 카운팅 되면 carry 없이 1 로 돌아간다.
        -- 이를 통해 외부 입출력을 정의하면 다음과 같다.
        nRst        : in std_logic;
          clk         : in std_logic;
        digit_one   : out std_logic_vector(3 downto 0);
        digit_ten   : out std_logic_vector(3 downto 0)
    );
    end counter_12;

architecture BEH of counter_12 is
    -- 다음은 내부 시그널이다. 위 port 는 외부 시그널을 의미하기 때문에 실질적 동작은 아래 기술하게 된다
    signal count_one          : std_logic_vector(3 downto 0);
    signal count_ten          : std_logic_vector(3 downto 0);

    begin

    process(nRst, clk)
        begin
        -- reset 동작은 active low 라는 점에 유의하자.
        if(nRst = '0') then
                -- 12 를 표현하기 위해서 아래 값으로 reset
            count_one <= "0010";
            count_ten <= "0001";

        -- clk'EVENT AND clk='1'은 rising edge 를 clk'EVENT AND clk='0'을 falling edge 를 의미
        -- RISING_EDGE(clk), FALLING_EDGE(clk)로 직접 작성하는 방법도 있다.
        elsif RISING_EDGE(clk) then
           if(count_one = 9) then
                -- (others => '0')으로 대체하여 쓸 수 있다.
                count_one <= "0000";
                count_ten <= count_ten + 1;
                else
                count_one <= count_one + 1;
            end if;

                if(count_one = 2 and count_ten = 1) then
                count_one <= "0001";
                     count_ten <= "0000";
                end if;
        end if;
```

```
49
50          digit_one <= count_one;
51          digit_ten <= count_ten;
52      end process;
53  end BEH;
54                                                  Colored by Color Scripter
```

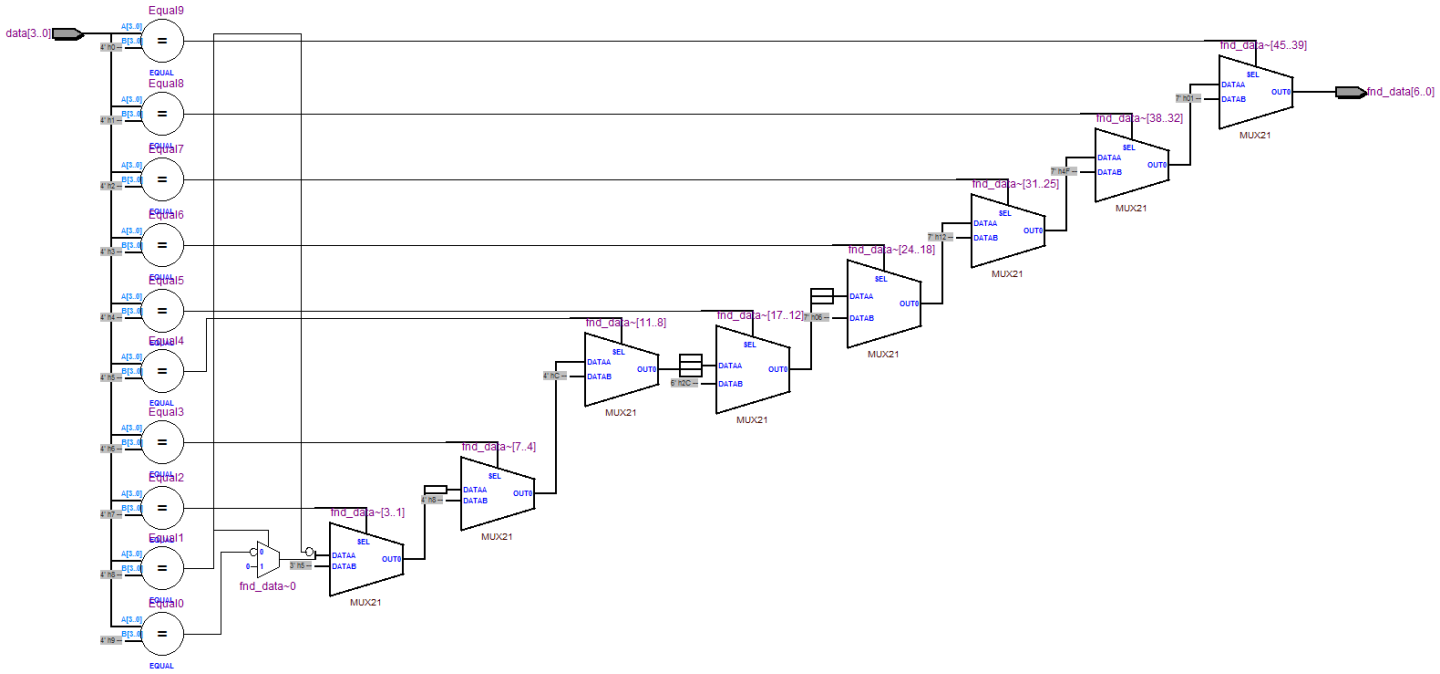


FND Decoder

시간을 출력하기 위해서는 다양한 방법이 존재할 수 있지만 직관적인 이해를 위해 FND Decoder 을 사용하기로 했다.

```
1   library ieee;
2       use ieee.std_logic_1164.all;
3       use ieee.std_logic_arith.all;
4       use ieee.std_logic_unsigned.all;
5
6   entity fnd_decoder is
7
8       port (
9           data :          in std_logic_vector(3 downto 0);
10          fnd_data :      out std_logic_vector(6 downto 0)
11      );                                                  cs
```

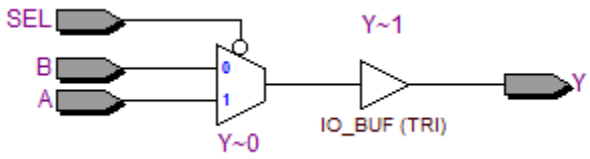
```
12 end fnd_decoder;
13
14 architecture beh of fnd_decoder is
15
16    begin
17    fnd_data <= "1000000" when data = x"0" else
18                "1111001" when data = x"1" else
19                "0100100" when data = x"2" else
20                "0110000" when data = x"3" else
21                "0011001" when data = x"4" else
22                "0010010" when data = x"5" else
23                "0000010" when data = x"6" else
24                "1011000" when data = x"7" else
25                "0000000" when data = x"8" else
26                "0010000" when data = x"9" else
27                "1111111";
28 end beh;
```

2X1 MUX

시간의 입력 방법엔 자연적으로 카운팅 되는 신호와 외부 입력으로 값을 바꿔줄 수 있는 신호가 있다. 이를 구현하기 위해 2X1 MUX 를 사용한다.

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

entity multiplexer is
    port(
        A :     in std_logic;
        B :     in std_logic;
        SEL :     in std_logic;
        Y :     out std_logic
    );
end multiplexer;

architecture BEH of multiplexer is
begin
    Y <= A when SEL = '0' else
         B when SEL = '1' else
         'Z';    -- 예외 출력 조건을 다음과 같이 Z 로 설정, Z 는 대문자 사용
end BEH;
```

3. 소스코드 및 코드 설명(3, 4 단계 함께 기술)

소스코드를 설명과 함께 기술한다.

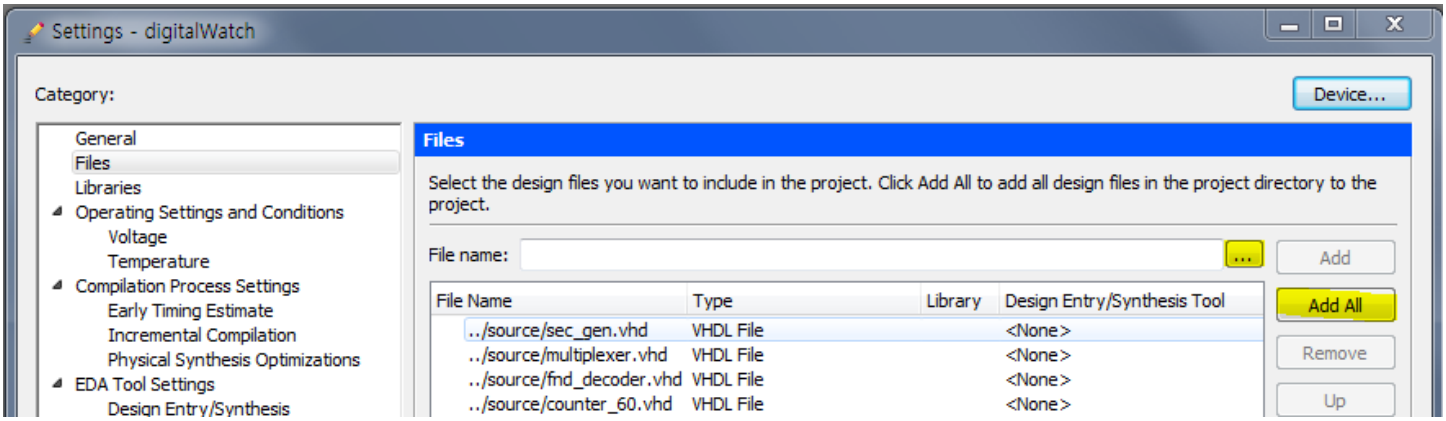
4. 시뮬레이션 결과 및 설명(3, 4 단계 함께 기술)

코드 시뮬레이션과 그 결과를 기술한다.

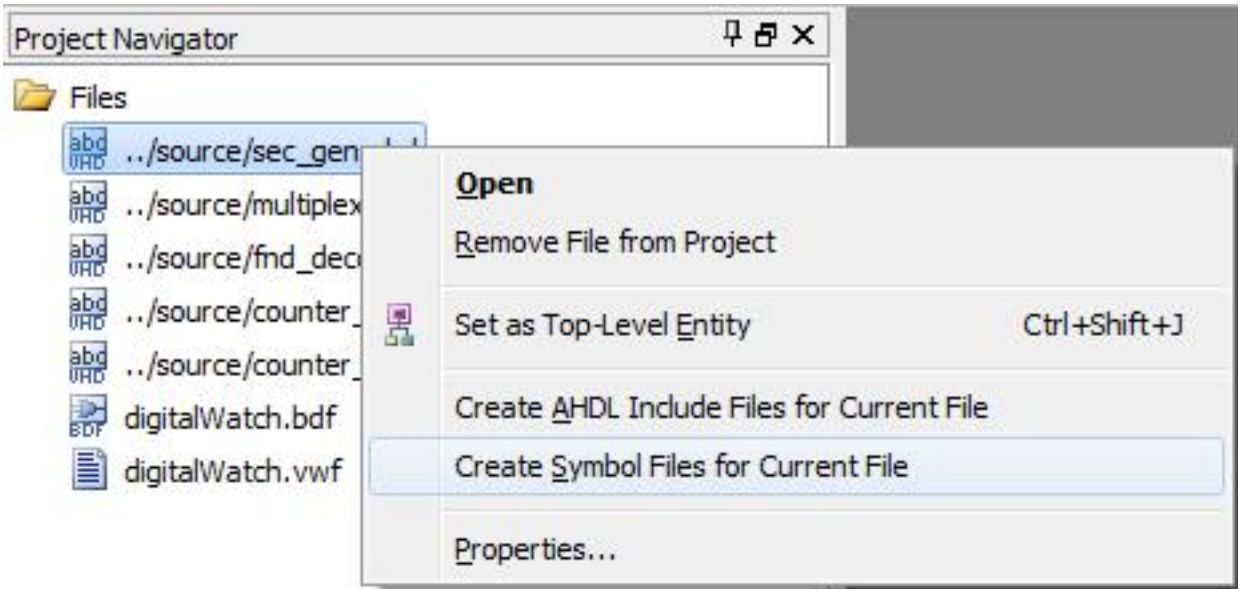
Schematic

Schematic 은 회로를 좀 더 직관적으로 이해 할 수 있기 때문에 VHDL 보다 우선적으로 시행해보는 것이 좋다.

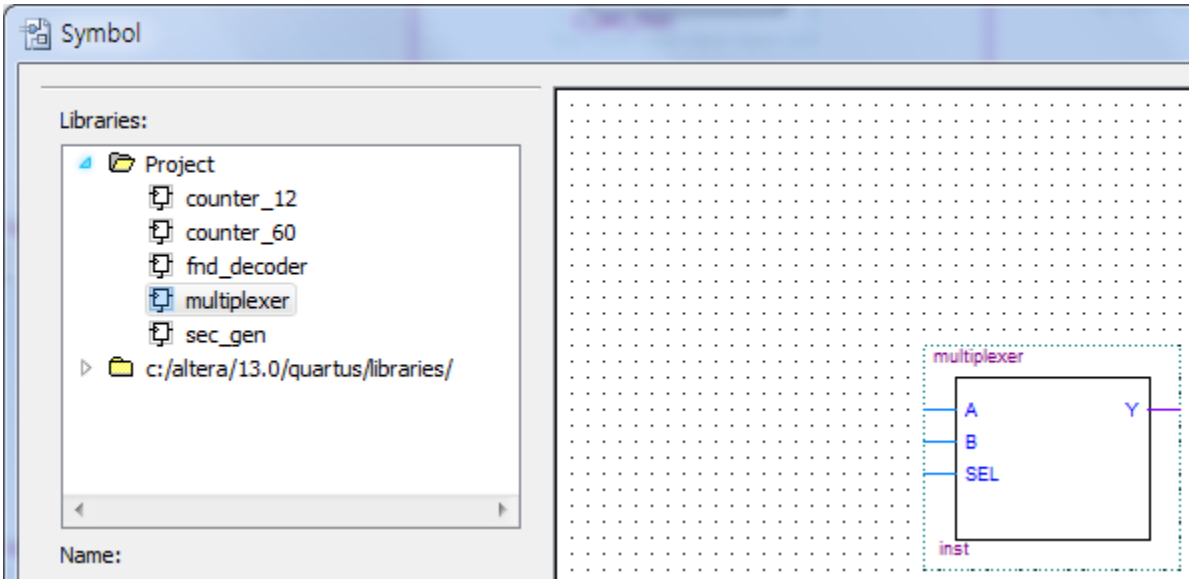
1. 프로젝트 생성 시, 하위 component 들을 연결해주어 프로젝트가 해당 파일을 인식할 수 있도록 하자.



2. Top entity 에서 각각의 컴포넌트들은 하나의 소자로 볼 수 있다. 이를 소자로 만들어 주는 기능인 ‘create symbol’을 실행하자.

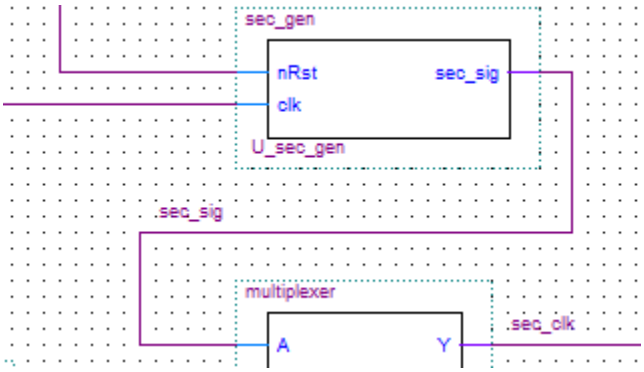


3. 각 심볼을 생성하고 도면을 더블클릭하면 해당 심볼이 생성된 것을 볼 수 있다.

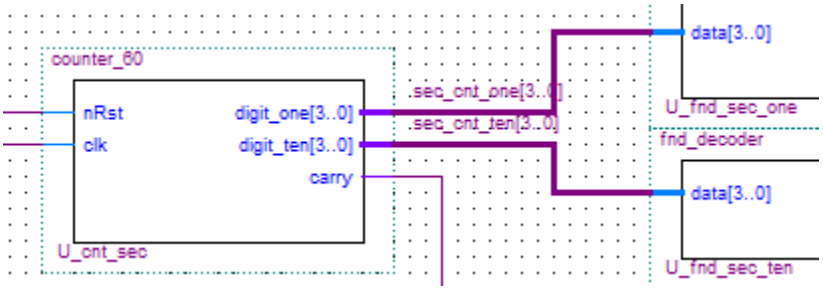


4. 이제 각 심볼을 이용하여 하위 컴포넌트들을 연결해보자.

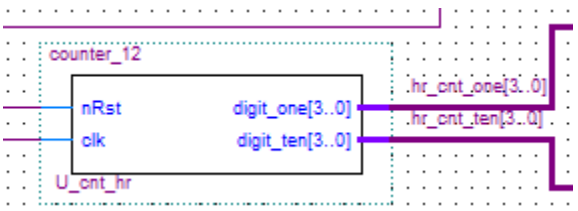
1) 1 초 생성기



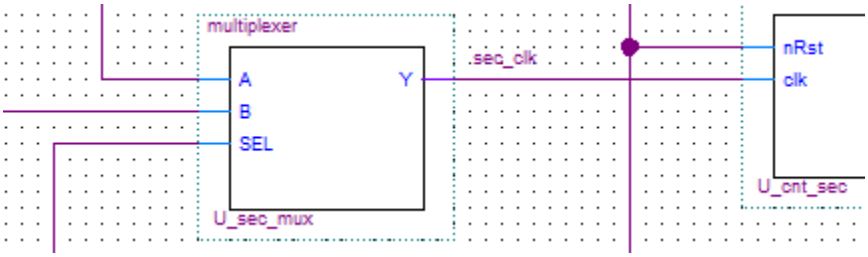
2) 60 진 카운터



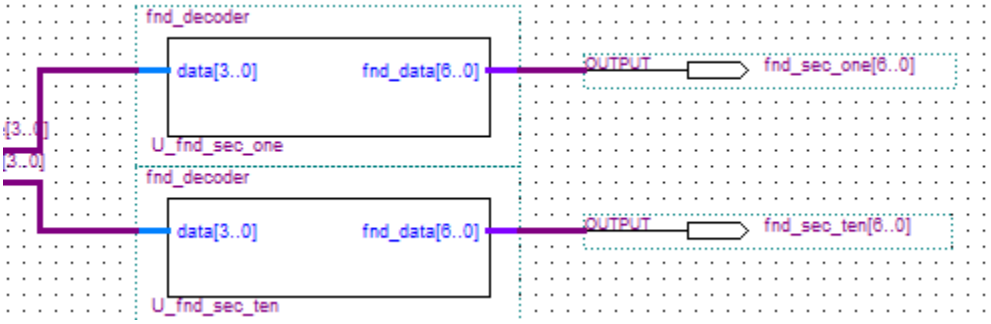
3) 12 진 카운터



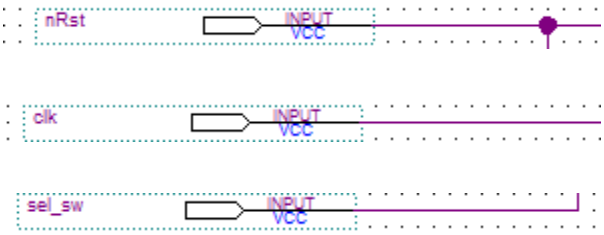
4) 멀티플렉서



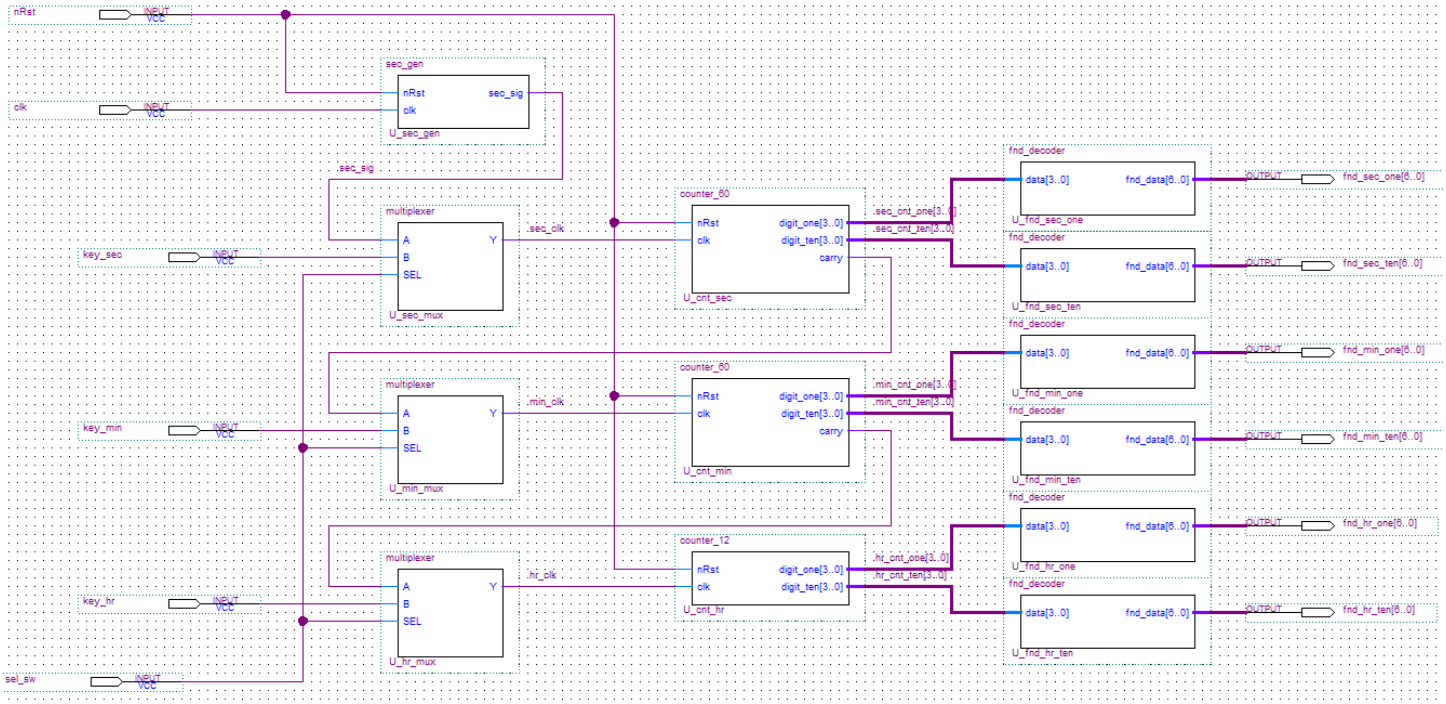
5) FND 디코더



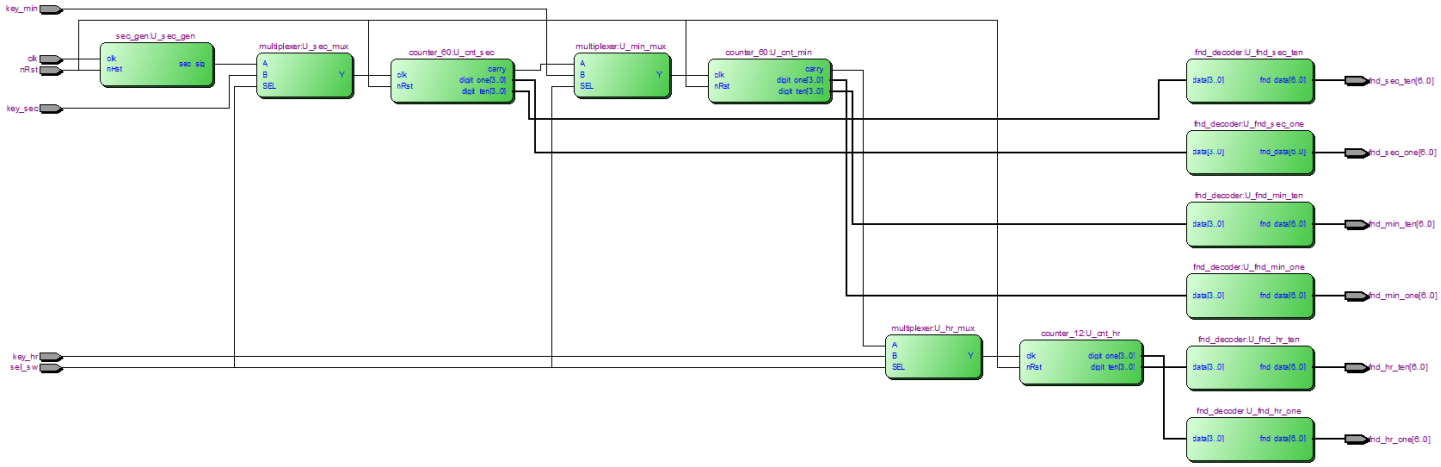
6) 입출력 포트 (특징적인 signal 만 표시)



7) 최종본



5. RTL Viewer 를 통해 어떤식으로 회로가 구성되는 지 살펴보자. (VHDL 로 작성한 파일과 동일한 구조를 가지는 지 살펴보자)



6. Pin planner 를 통해 각각의 핀을 배치해보자.

1) 1 초 생성기의 input 인 50MHz 클락

Signal Name	FPGA Pin No.	Description
CLOCK_27	PIN_D13	TV Decoder Clock Input.
CLOCK_50	PIN_N2	50 MHz clock input

Node Name	Direction	Location
clk	Input	PIN_N2

2) 푸시 버튼을 통해 reset 기능과 초, 분, 시의 각각의 key 값을 연결

KEY[0]	PIN_G26	Pushbutton[0]
KEY[1]	PIN_N23	Pushbutton[1]
KEY[2]	PIN_P23	Pushbutton[2]
KEY[3]	PIN_W26	Pushbutton[3]

3) 스위치를 통해 시간 조정 여부를 조절 할 수 있도록 한다.

Signal Name	FPGA Pin No.	Description
SW[0]	PIN_N25	Toggle Switch[0]

key_hr	Input	PIN_W26
key_min	Input	PIN_P23
key_sec	Input	PIN_N23
nRst	Input	PIN_G26
sel_sw	Input	PIN_N25

4) FND 순서를 헷갈리면 시, 분, 초의 위치가 꼬이므로 잘 보고 설정하도록 하자.

HEX6[0]	PIN_R2	HEX7[0]	PIN_L3
HEX6[1]	PIN_P4	HEX7[1]	PIN_L2
HEX6[2]	PIN_P3	HEX7[2]	PIN_L9
HEX6[3]	PIN_M2	HEX7[3]	PIN_L6
HEX6[4]	PIN_M3	HEX7[4]	PIN_L7
HEX6[5]	PIN_M5	HEX7[5]	PIN_P9
HEX6[6]	PIN_M4	HEX7[6]	PIN_N9

HEX4[0]	PIN_U9	HEX5[0]	PIN_T2
HEX4[1]	PIN_U1	HEX5[1]	PIN_P6
HEX4[2]	PIN_U2	HEX5[2]	PIN_P7
HEX4[3]	PIN_T4	HEX5[3]	PIN_T9
HEX4[4]	PIN_R7	HEX5[4]	PIN_R5
HEX4[5]	PIN_R6	HEX5[5]	PIN_R4
HEX4[6]	PIN_T3	HEX5[6]	PIN_R3

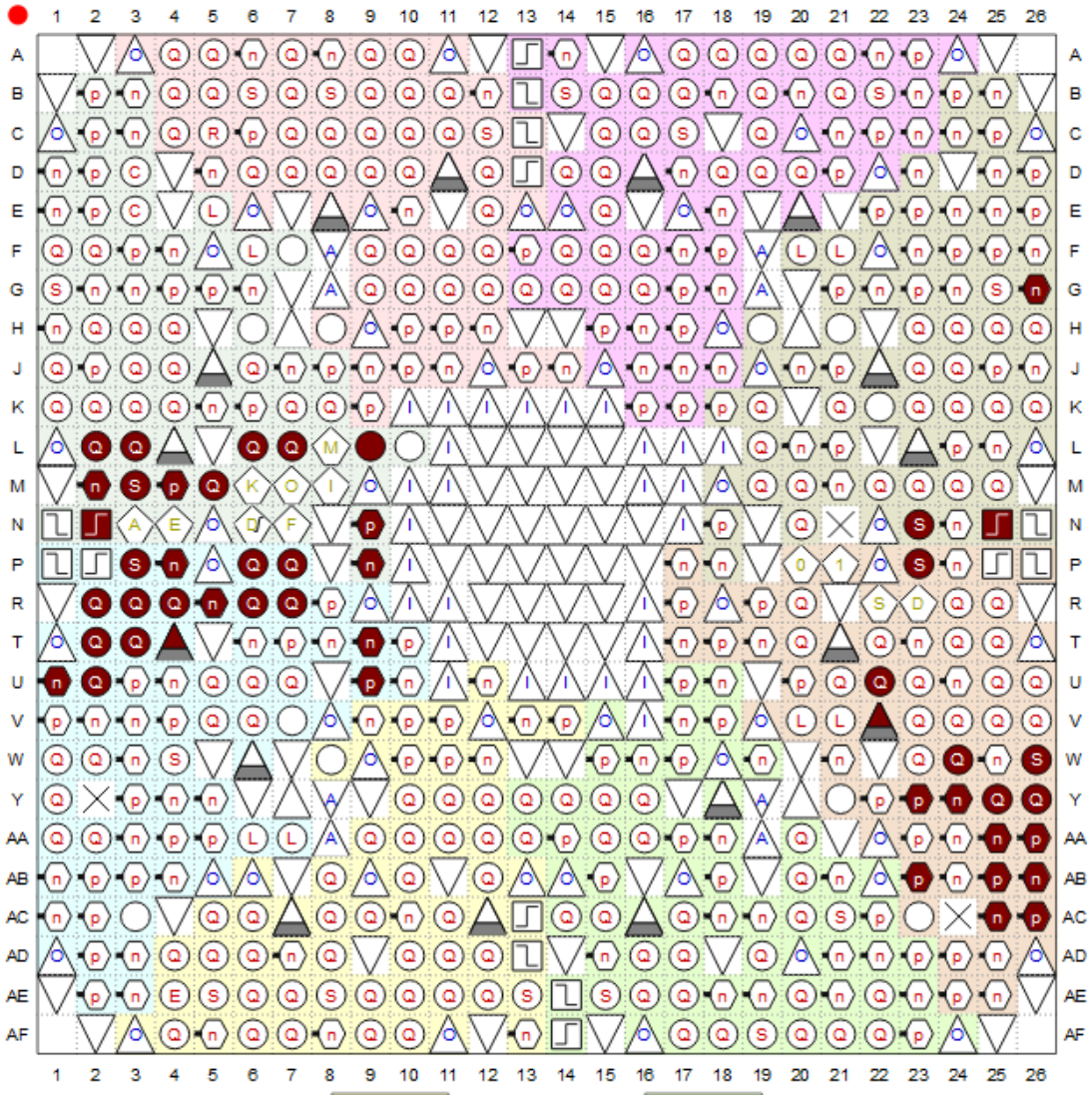
HEX2[0]	PIN_AB23	HEX3[0]	PIN_Y23
HEX2[1]	PIN_V22	HEX3[1]	PIN_AA25
HEX2[2]	PIN_AC25	HEX3[2]	PIN_AA26
HEX2[3]	PIN_AC26	HEX3[3]	PIN_Y26
HEX2[4]	PIN_AB26	HEX3[4]	PIN_Y25
HEX2[5]	PIN_AB25	HEX3[5]	PIN_U22
HEX2[6]	PIN_Y24	HEX3[6]	PIN_W24

fnd_hr_one[6]	Output	PIN_M4
fnd_hr_one[5]	Output	PIN_M5
fnd_hr_one[4]	Output	PIN_M3
fnd_hr_one[3]	Output	PIN_M2
fnd_hr_one[2]	Output	PIN_P3
fnd_hr_one[1]	Output	PIN_P4
fnd_hr_one[0]	Output	PIN_R2
fnd_hr_ten[6]	Output	PIN_N9
fnd_hr_ten[5]	Output	PIN_P9
fnd_hr_ten[4]	Output	PIN_L7
fnd_hr_ten[3]	Output	PIN_L6
fnd_hr_ten[2]	Output	PIN_L9
fnd_hr_ten[1]	Output	PIN_L2
fnd_hr_ten[0]	Output	PIN_L3

fnd_min_one[6]	Output	PIN_T3
fnd_min_one[5]	Output	PIN_R6
fnd_min_one[4]	Output	PIN_R7
fnd_min_one[3]	Output	PIN_T4
fnd_min_one[2]	Output	PIN_U2
fnd_min_one[1]	Output	PIN_U1
fnd_min_one[0]	Output	PIN_U9
fnd_min_ten[6]	Output	PIN_R3
fnd_min_ten[5]	Output	PIN_R4
fnd_min_ten[4]	Output	PIN_R5
fnd_min_ten[3]	Output	PIN_T9
fnd_min_ten[2]	Output	PIN_P7
fnd_min_ten[1]	Output	PIN_P6
fnd_min_ten[0]	Output	PIN_T2

fnd_sec_one[6]	Output	PIN_Y24
fnd_sec_one[5]	Output	PIN_AB25
fnd_sec_one[4]	Output	PIN_AB26
fnd_sec_one[3]	Output	PIN_AC26
fnd_sec_one[2]	Output	PIN_AC25
fnd_sec_one[1]	Output	PIN_V22
fnd_sec_one[0]	Output	PIN_AB23
fnd_sec_ten[6]	Output	PIN_W24
fnd_sec_ten[5]	Output	PIN_U22
fnd_sec_ten[4]	Output	PIN_Y25
fnd_sec_ten[3]	Output	PIN_Y26
fnd_sec_ten[2]	Output	PIN_AA26
fnd_sec_ten[1]	Output	PIN_AA25
fnd_sec_ten[0]	Output	PIN_Y23

5) 핀 배치가 완료되었다. 복잡하긴 하지만 매뉴얼을 본다면 쉽게 따라할 수 있을 것이다.



VHDL

VHDL 은 처음에 복잡할 수 있으나 다른 플랫폼으로의 전환이 용이하므로 익숙해 질 수 있도록 하자.

1. HDL 작성
    1) 최 외곽 input, output 을 선언한다.
    2) 외부 파일에 있는 포트구조를 가져온다.
    3) Port map 에 컴포넌트의 시그널들을 정의한다. Top entity 에 있는 시그널을 연결하여야 한다.
    4) 3)단계 작성 중 top entity 에 선언되지 않은 내부 시그널은 architecture 내에 선언하여 사용한다.

아래는 위와 같은 단계를 거쳐 만들어진 디지털 시계의 상세 코드이다.

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

entity digitalWatch is
    port
    (
        -- top entity 의 입출력 포트 작성
        nRst :            in std_logic;
        clk :              in std_logic;

        key_sec :        in std_logic;
        key_min :        in std_logic;
        key_hr :          in std_logic;
        sel_sw :          in std_logic;

        fnd_hr_one :    out std_logic_vector(6 downto 0);
        fnd_hr_ten :    out std_logic_vector(6 downto 0);
        fnd_min_one :    out std_logic_vector(6 downto 0);
        fnd_min_ten :  out std_logic_vector(6 downto 0);
        fnd_sec_one :  out std_logic_vector(6 downto 0);
        fnd_sec_ten :  out std_logic_vector(6 downto 0)
    );
end digitalWatch;
architecture beh of digitalWatch is
    -- 1 초 발생기
    component sec_gen
        port(
            nRst       : in std_logic;
            clk        : in std_logic;
            sec_sig    : out std_logic
        );
    end component;
    -- 60 진 카운터
    component counter_60
        port(
            -- 60 진 카운터에서는 sec, min 에 대한 각각의 클럭으로 카운팅이 일어나며 reset 을 통해 0 으로 리셋된다.
            -- 또한 FND 디코더에 연결된 4bit 의 출력을 내보낸다. 60 이 카운팅 되면 다음 단위로 carry 가 발생한다.
            -- 이를 통해 외부 입출력을 정의하면 다음과 같다.
            nRst            : in std_logic;
            clk             : in std_logic;
            digit_one     : out std_logic_vector(3 downto 0);
            digit_ten     : out std_logic_vector(3 downto 0);
            carry           : out std_logic
        );
    end component;
    -- 12 진 카운터
    component counter_12
         port(
             -- 12 진 카운터에서는 hour 에 대한 각각의 클럭으로 카운팅이 일어나며 reset 을 통해 12 로 리셋된다.
             -- 또한 FND 디코더에 연결된 4bit 의 출력을 내보낸다. 12 가 카운팅 되면 carry 없이 1 로 돌아간다.
             -- 이를 통해 외부 입출력을 정의하면 다음과 같다.
             nRst       : in std_logic;
             clk        : in std_logic;
             digit_one  : out std_logic_vector(3 downto 0);
             digit_ten  : out std_logic_vector(3 downto 0)
```

cs

```
59          );
60      end component;
61      -- FND decoder
62      component fnd_decoder
63          port (
64              data :        in std_logic_vector(3 downto 0);
65              fnd_data :    out std_logic_vector(6 downto 0)
66          );
67      end component;
68      -- MUX
69      component multiplexer
70          port(
71              A :     in std_logic;
72              B :     in std_logic;
73              SEL :     in std_logic;
74              Y :     out std_logic
75          );
76      end component;
77      -- 내부 시그널 작성부
78      signal sec_cnt_one :    std_logic_vector(3 downto 0);
79      signal sec_cnt_ten :    std_logic_vector(3 downto 0);
80      signal min_cnt_one :    std_logic_vector(3 downto 0);
81      signal min_cnt_ten :    std_logic_vector(3 downto 0);
82      signal hr_cnt_one :    std_logic_vector(3 downto 0);
83      signal hr_cnt_ten :    std_logic_vector(3 downto 0);
84
85      signal sec_carry :    std_logic;
86      signal min_carry :    std_logic;
87      signal sec_sig :        std_logic;
88      signal sec_clk :        std_logic;
89      signal min_sig :        std_logic;
90      signal min_clk :        std_logic;
91      signal hr_sig :        std_logic;
92      signal hr_clk :        std_logic;
93
94      begin
95          -- 각 컴포넌트의 입출력을 정의, 하위 계층의 시그널 이름을 top entity 에서 다시 정의하는 것을 볼 수 있다.
96          U_sec_gen : sec_gen
97          port map(
98              nRst         => nRst,
99              clk          => clk,
100             sec_sig      => sec_sig
101         );
102
103         U_sec_mux : multiplexer
104         port map(
105             A     => sec_sig,
106             B     => key_sec,
107             SEL => sel_sw,
108             Y     => sec_clk
109         );
110
111         U_cnt_sec : counter_60
112         port map(
113             nRst     => nRst,
114             clk     => sec_clk,
115             digit_one => sec_cnt_one,
116             digit_ten => sec_cnt_ten,
117             carry     => sec_carry
118         );
119
120         U_min_mux : multiplexer
121         port map(
122             A     => sec_carry,
123             B     => key_min,
124             SEL => sel_sw,
125             Y     => min_clk
126         );
127
128         U_cnt_min : counter_60
129         port map(
130             nRst     => nRst,
131             clk     => min_clk,
132             digit_one => min_cnt_one,
133             digit_ten => min_cnt_ten,
134             carry     => min_carry
135         );
```

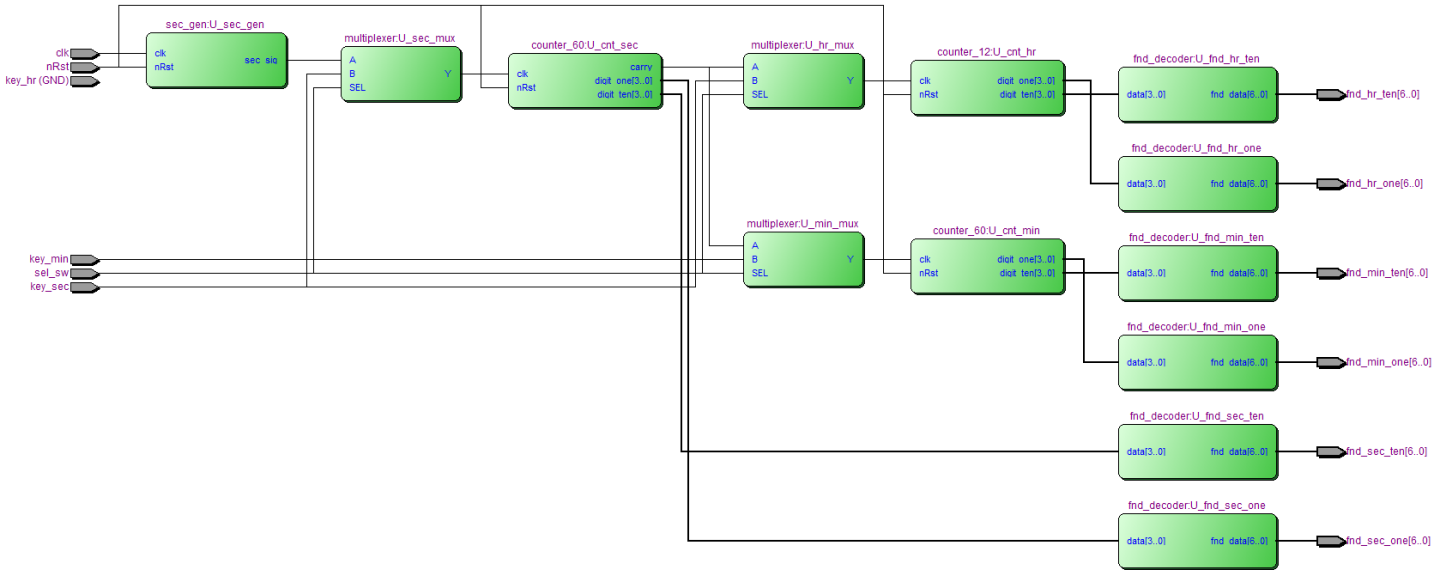
```
136
137     U_hr_mux : multiplexer
138     port map(
139         A   => sec_carry,
140         B   => key_sec,
141         SEL => sel_sw,
142         Y   => hr_clk
143     );
144
145     U_cnt_hr : counter_12
146     port map(
147         nRst    => nRst,
148         clk     => hr_clk,
149         digit_one => hr_cnt_one,
150         digit_ten => hr_cnt_ten
151     );
152
153     U_fnd_sec_one : fnd_decoder
154     port map(
155         data        => sec_cnt_one,
156         fnd_data    => fnd_sec_one
157     );
158
159     U_fnd_sec_ten : fnd_decoder
160     port map(
161         data        => sec_cnt_ten,
162         fnd_data    => fnd_sec_ten
163     );
164
165     U_fnd_min_one : fnd_decoder
166     port map(
167         data        => min_cnt_one,
168         fnd_data    => fnd_min_one
169     );
170
171     U_fnd_min_ten : fnd_decoder
172     port map(
173         data        => min_cnt_ten,
174         fnd_data    => fnd_min_ten
175     );
176
177     U_fnd_hr_one : fnd_decoder
178     port map(
179         data        => hr_cnt_one,
180         fnd_data    => fnd_hr_one
181     );
182
183     U_fnd_hr_ten : fnd_decoder
184     port map(
185         data        => hr_cnt_ten,
186         fnd_data    => fnd_hr_ten
187     );
188
189
end beh;
```

Colored by Color Scripter

빠른 작성을 위한 팁을 적자면

- 외부 파일에서 component 시그널을 가져와 port 오버라이딩
- 위쪽에 선언된 컴포넌트 정의를 가져와 portmap 오버라이딩
- 문법적인 에러가 아닌 경우 선언된 시그널의 이름이 없거나 잘못 매치되었을 확률이 높다. 에러메시지를 더블클릭해 해당 코드를 살펴보도록 하자.

2. RTL Viewer 를 통해 어떤식으로 회로가 구성되는 지 살펴보자. (Schematic 으로 작성한 파일과 동일한 구조를 가지는 지 살펴보자)



5. 실습보드 적용 결과

소스를 보드에 다운로드하여 예상한 결과에 맞게 동작하는 지 테스트하시오

실습 시간 중 에러가 너무 많이 발생해 당황하여 보드를 통해 구현하지 못했습니다. 이후 실습시간에 시간이 있다면 다운로드를 진행하여 결과를 확인할 수 있도록 하겠습니다.

6. 실습소감

실습을 통해 경험한 것을 자유롭게 서술하시오.

이번 실습 시 준비해 가야 하는 항목들이 있었는데 이전에 만들어 놓은 MUX 의 포트명이 실습 동영상과 달라 자꾸만 에러가 발생했었다. 때문에 수업 중 실습시간에 다운로드를 진행하지 못한 점이 아쉬웠다. 이번 실습에 자잘한 문제로 컴필레이션 시에 많은 에러가 발생는데 보드 테스트에서는 한번에 원하는 결과가 나올 수 있도록 메뉴얼을 잘 살펴보았다. 다음 수업 시간에 다운로딩 해 볼 여유가 있다면 보드 테스트를 진행해볼 생각이다. 또 회로를 직접 만지는 것이 아니라 익숙한 코딩 형태로 하드웨어를 조작할 수 있어 직관적이고 편리했다. 이전엔 브레드보드에 신호등을 구현하거나 PCB 를 구매해 스피커를 만들었는데 원하는 결과가 나오지 않으면 보드를 새로 구매하거나 회로를 뜯어내고 다시 조립하는 등의 어려움이 있었다. HDL 로 구현하는 것이 훨씬 편리하다는 것을 느끼니 훨씬 유익하고 즐겁게 느껴졌다. 직접 구현하는 회로의 한계를 벗어나 Quartus II 를 사용해 극복할 수 있다면 머릿속으로 그린 상상을 구현해 낼 수 있을 것 같다.
이번 실습에서 가장 어려웠던 점은 1 초 발생기의 시뮬레이션 중 시간 폭을 바꾸어 테스트하는 것이였다. Top entity 에서도 동일하게 시간 폭을 바꾸어보았지만 function simulation 을 클릭하면 나오는 화면에서 계속 멈춰있는 문제가 발생해 진행할 수 없었다. 아무런 에러메시지 없이 멈춰버리니 어찌할 방도가 없었다.

7. 문의 사항

실습하다 겪은 어려웠던 점을 기술하시오.

이번 실습 시, 1 초 생성기와 같은 방식으로 top entity 에도 시간 폭을 변경하여 시뮬레이션 해보려고 했으나 function simulation 버튼을 클릭 시 다음 화면에서 멈춰져 결과 파형이 나타나지 않은 문제가 발생하였습니다. 방법을 몰라 대처하지 못했는데 이 점 질문 드립니다. 다음은 멈춤 현상이 발생한 부분의 캡쳐화면입니다.