

# Verilog HDL을 이용한 FPGA 디지털 시계 설계

전자공학전공 김상은  
전자공학전공 차예진  
전자공학전공 최보영

# Table of Contents.

001

주 제 소 개

002

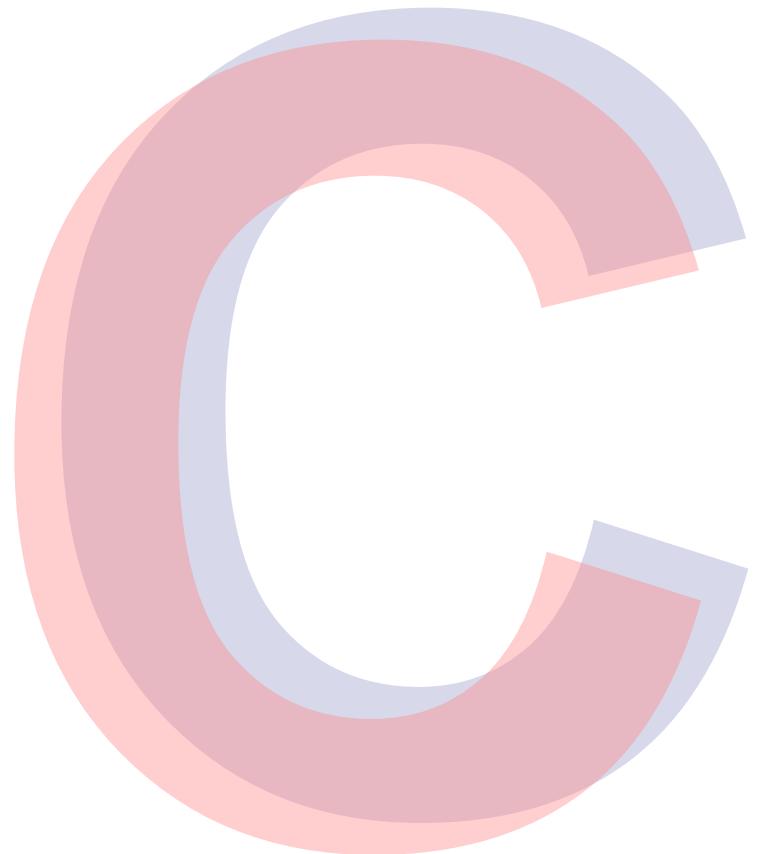
설 계 과 정 및 결 과

003

고 칠

004

Q & A



“

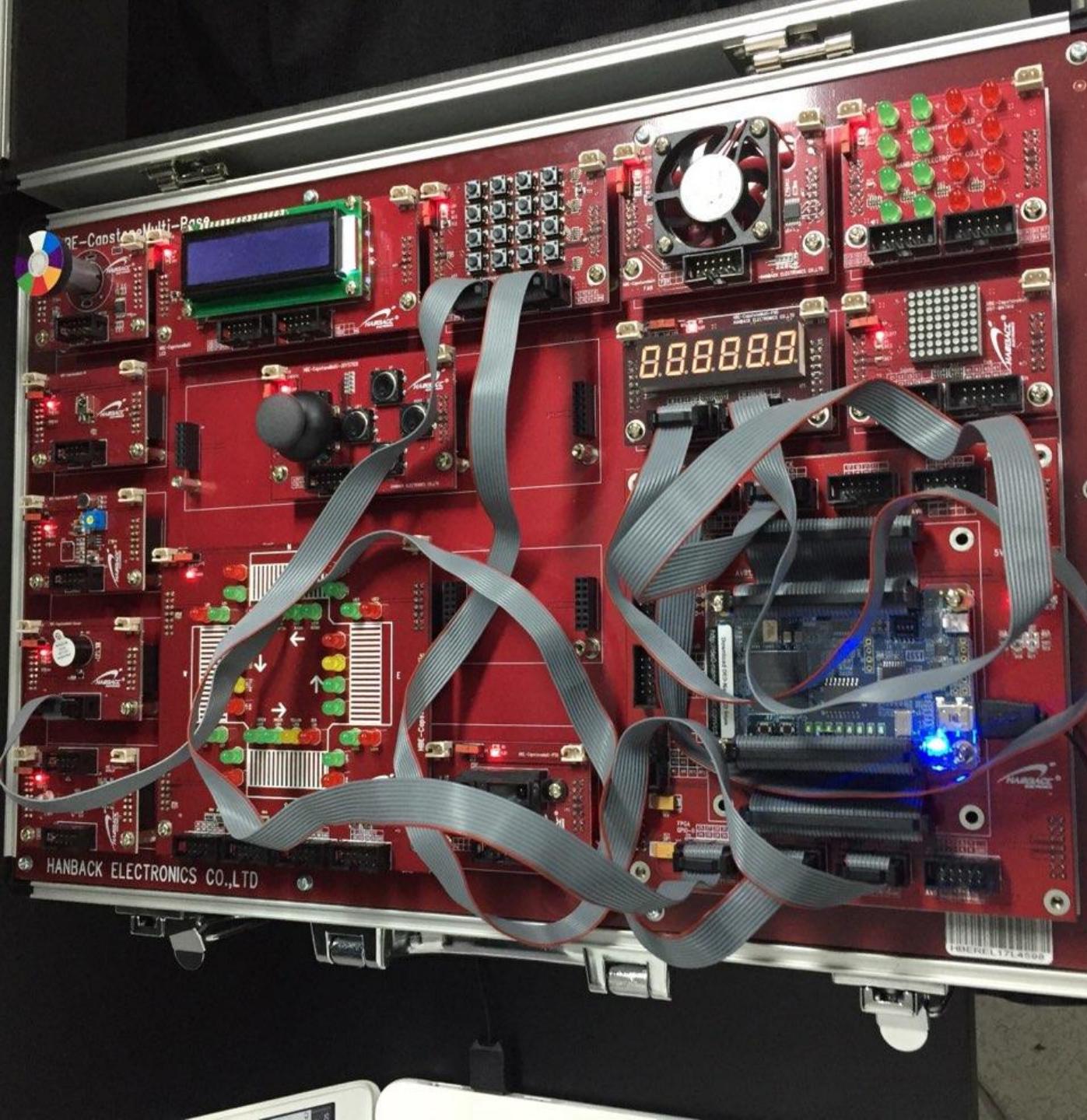
# 주제 소개

”

# 주제 소개

디지털 시계의 스위치와 동작모드

모드 선택(SW0)	SW1	SW2	SW3	SW4	SW6	SW7	
0	시계 모드	현재 시간 출력					
1	시간 설정	위치 선택	시간 증가	시계 모드 BUZZER ON OFF	스톱워치 GO STOP RESET	시계 모드 AM PM	알람 모드 BUZZER ON OFF
2	알람 설정	위치 선택	시간 증가				
3	스톱워치						
4	세계 시간 (시드니)						
5	세계 시간 (런던)	현재 시간 출력					



## 설계 목적 및 최종 설계 목표

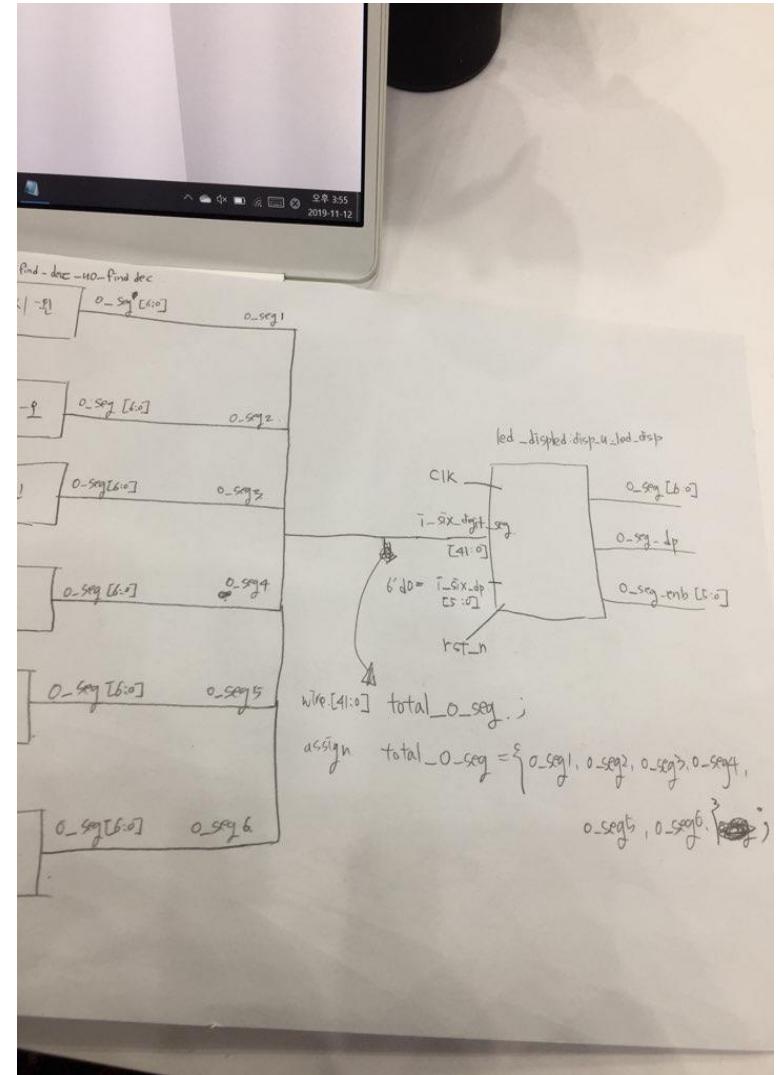
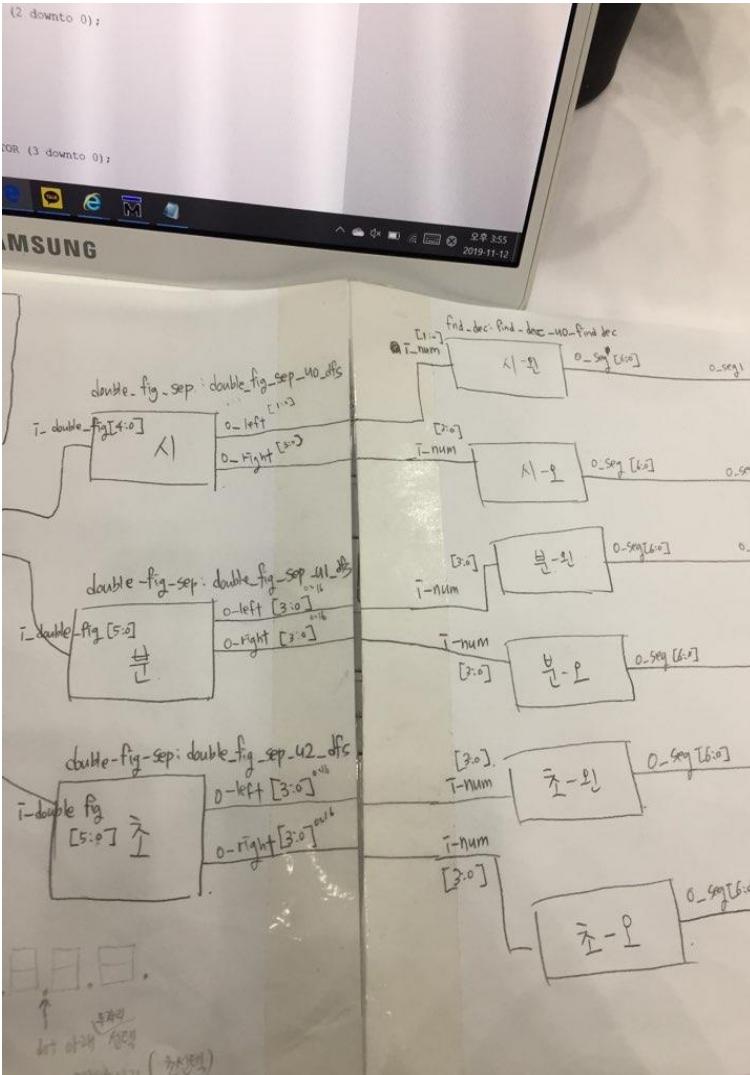
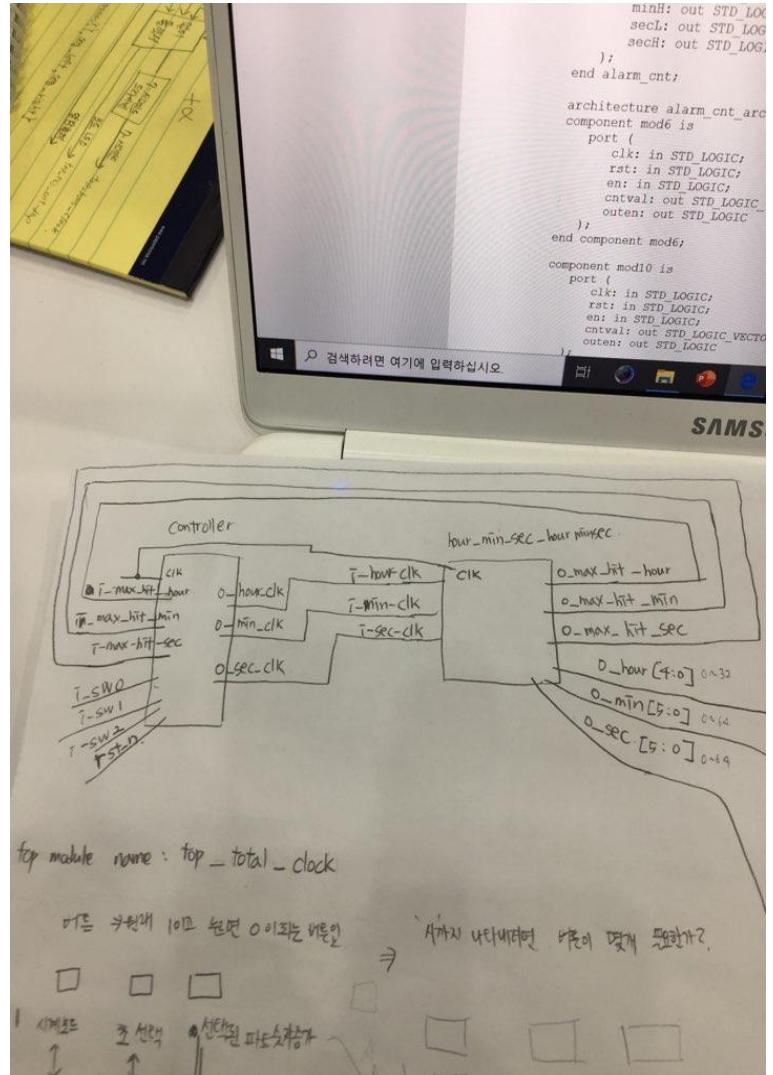
- ✓ 논리회로실험에서 배운 이론을 활용하여 실생활에서 쉽게 볼 수 있는 디지털 시계를 제작한다.
- ✓ 프로젝트 진행을 통하여 팀원과 협력을 통해 문제를 해결하는 능력을 배양한다.
- ✓ 기본적으로 시, 분, 초 단위가 표시되는 시계 기능, 알람 기능, 스톱워치 기능에 스위치, 부저, LED 등 여러 장치를 이용하여 기능을 추가한다.

“

## 설계 과정 및 결과

”

# 직접 구상하고 그린 회로도



# Controller

- Controller은 디지털 시계, 스톱워치 및 알람 등 세 가지 기능 가운데 하나를 선택해서 출력시키는 기능을 제공한다. 이 기능에서 디지털 시계의 시간을 설정한다.
- 디지털 클럭 로직과 알람 로직은 1Hz를 사용하고, 스톱워치 로직은 1/100초 단위로 측정되므로 100Hz를 사용한다.
- Controller에서는 현재의 모드를 저장하는 신호 모드를 parameter로 선언하고, SW0 스위치가 한 번 눌릴 때마다 모드가 순환적으로 변경되도록 한다.
- 2bit mode 비트를 선언하고, 스위치를 한 번 누를 때마다 하나의 클럭으로 인식해 mode 비트를 하나씩 증가시킨다.
- 현재의 모드 정보는 저장한 다음 모드로 전달된다.

```
104 | output [2:0] o_mode ; // clock=0, move=1, alarm=2
105 | output [1:0] o_position ; // hour, min, sec position
106 |
107 | output o_sec_clk ;
108 | output o_min_clk ;
109 | output o_hour_clk ; // hour clk o => for only clock
110 |
111 | output o_alarm_en1 ;
112 | output o_alarm_en2 ;
113 |
114 | output [1:0] o_stopwatch_en ; // stop_watch state
115 | output o_am_pm_en ;
116 |
117 | output o_alarm_sec_clk ;
118 | output o_alarm_min_clk ;
119 | output o_alarm_hour_clk ; // alarm hour clk
120 |
121 | output o_sw_ssec_clk ;
122 | output o_sw_sec_clk ;
123 | output o_sw_min_clk ; // sw clk
124 |
125 | output dis_hour ;
126 | output dis_min ;
127 | output dis_sec ;
128 |
129 | input i_max_hit_sec ;
130 | input i_max_hit_min ;
```

```

131 input i_max_hit_hour ; // hit hour nessary o (max:24)
132
133 input i_sw_hit_sec ;
134 input i_sw_hit_ssec ;
135 input i_sw_hit_min ;
136
137 input i_sw0 ;
138 input i_sw1 ;
139 input i_sw2 ;
140 input i_sw3 ; // alarm button , when 0, no alarm, when 1, alarm ring
141 input i_sw4 ; // at stopwatch, stop button
142
143 input i_sw6 ;
144 input i_sw7 ;
145
146 input clk ;
147 input rst_n ;
148
149 parameter MODE_CLOCK = 3'd0 ;
150 parameter MODE_SETUP = 3'd1 ;
151 parameter MODE_ALARM = 3'd2 ; // alarm mode
152 parameter MODE_STOPWATCH= 3'd3 ;
153 parameter MODE_SYDNEY_CLOCK = 3'd4; // world clock ( sydney )
154 parameter MODE_LONDON_CLOCK = 3'd5; // world clock ( london )
155
156 parameter POS_SEC = 2'd0 ,
157

```

```

212 /// at blink using
213
214 wire dis_hour ;
215
216 nco u_nc0(
217 .o_gen_clk ( dis_hour ),
218 .i_nc0_num ( 32'd50000000 ), // 1 sec , clk change
219 .clk ( clk ),
220 .rst_n ( rst_n ));
221
222 wire dis_min ;
223
224 nco u_nc1(
225 .o_gen_clk ( dis_min ),
226 .i_nc0_num ( 32'd50000000 ), // 1 sec , clk change
227 .clk ( clk ),
228 .rst_n ( rst_n ));
229
230 wire dis_sec ;
231
232 nco u_nc2(
233 .o_gen_clk ( dis_sec ),
234 .i_nc0_num ( 32'd50000000 ), // 1 sec , clk change
235 .clk ( clk ),
236 .rst_n ( rst_n ));
237
238

```

```

258     o_position <= POS_SEC;           // pos_sec =0 start! add 1
259 end else begin
260     if(o_position >= POS_HOUR) begin
261         o_position <= POS_SEC;
262     end else begin
263         o_position <= o_position + 1'd1;
264     end
265 end
266
267 reg      o_alarm_enl          ;
268
269 always @ (posedge sw3 or negedge rst_n) begin
270     if(rst_n == 1'b0) begin
271         o_alarm_enl <= 2'd0;
272     end else begin
273         o_alarm_enl <= o_alarm_enl + 1'bl;
274     end
275 end
276
277 reg      o_alarm_en2          ;
278
279 always @ (posedge sw7 or negedge rst_n) begin
280     if(rst_n == 1'b0) begin
281         o_alarm_en2 <= 2'd0;
282     end else begin
283         o_alarm_en2 <= o_alarm_en2 + 1'bl;
284

```

```

288     //// o_stopwatch_en's mode
289
290 parameter MODE_RESET = 2'd0 ;
291 parameter MODE_GO   = 2'd1 ;
292 parameter MODE_STOP  = 2'd2 ;
293
294 reg [1:0] o_stopwatch_en ; // reset, go, stop
295
296 always @ (posedge sw4 or negedge rst_n) begin
297     if(rst_n == 1'b0) begin
298         o_stopwatch_en <= MODE_RESET;
299     end else begin
300         if(o_stopwatch_en >= MODE_STOP) begin
301             o_stopwatch_en <= MODE_RESET;
302         end else begin
303             o_stopwatch_en <= o_stopwatch_en + 1'bl;
304         end
305     end
306 end
307
308 reg      o_am_pm_en          ;
309
310 always @ (posedge sw6 or negedge rst_n) begin           // when o_am_pm_en =1, am_pm mode
311     if(rst_n == 1'b0) begin
312         o_am_pm_en <= 1'b0;
313     end else begin
314         o_am_pm_en <= o_am_pm_en + 1'bl;
315

```

```

319 | wire      clk_lhz      ;
320 |
321 | nco      ul_nco(
322 |   .o_gen_clk  ( clk_lhz      ),
323 |   .i_nco_num  ( 32'd50000000 ),
324 |   .clk        ( clk          ),
325 |   .rst_n     ( rst_n       ));
326 |
327 | reg       o_sec_clk    ;
328 | reg       o_min_clk    ;
329 | reg       o_hour_clk   ; // hour clk making
330 |
331 | reg       o_alarm_sec_clk;
332 | reg       o_alarm_min_clk;
333 | reg       o_alarm_hour_clk;
334 |
335 | reg       o_sw_ssec_clk;
336 | reg       o_sw_sec_clk  ;
337 | reg       o_sw_min_clk  ;
338 |
339 |
340 | always @(*) begin
341 |   case(o_mode)
342 |     MODE_CLOCK : begin
343 |       o_sec_clk = clk_lhz;
344 |       o_min_clk = i_max_hit_sec;
345 |       o_hour_clk = i_max_hit_min; // when max-min, hour+1
346 |

```

```

340 |     always @(*) begin
341 |       case(o_mode)
342 |         MODE_CLOCK : begin
343 |           o_sec_clk = clk_lhz;
344 |           o_min_clk = i_max_hit_sec;
345 |           o_hour_clk = i_max_hit_min; // when max-min, hour+1
346 |           o_alarm_sec_clk = 1'b0;
347 |           o_alarm_min_clk = 1'b0;
348 |           o_alarm_hour_clk = 1'b0;
349 |           o_sw_ssec_clk = 1'b0;
350 |           o_sw_sec_clk = 1'b0;
351 |           o_sw_min_clk = 1'b0;
352 |         end
353 |         MODE_SETUP : begin
354 |           case(o_position)
355 |             POS_SEC : begin
356 |               o_sec_clk = ~sw2;
357 |               o_min_clk = 1'd0;
358 |               o_hour_clk = 1'd0; // when sec select, min and hour = 0
359 |               o_alarm_sec_clk = 1'b0;
360 |               o_alarm_min_clk = 1'b0;
361 |               o_alarm_hour_clk = 1'b0;
362 |               o_sw_ssec_clk = 1'b0;
363 |               o_sw_sec_clk = 1'b0;
364 |               o_sw_min_clk = 1'b0;
365 |             end
366 |             POS_MIN : begin

```

```
390     MODE_ALARM : begin
391         case(o_position)
392             POS_SEC : begin
393                 o_sec_clk = clk_lhz;
394                 o_min_clk = i_max_hit_sec;
395                 o_hour_clk = i_max_hit_min;
396                 o_alarm_sec_clk = ~sw2;
397                 o_alarm_min_clk= 1'b0;
398                 o_alarm_hour_clk = 1'b0;
399                 o_sw_ssec_clk = 1'b0;
400                 o_sw_sec_clk = 1'b0;
401                 o_sw_min_clk = 1'b0;
402             end
403             POS_MIN : begin
404                 o_sec_clk = clk_lhz;
405                 o_min_clk = i_max_hit_sec;
406                 o_hour_clk = i_max_hit_min;
407                 o_alarm_sec_clk = 1'b0;
408                 o_alarm_min_clk = ~sw2;
409                 o_alarm_hour_clk = 1'b0;
410                 o_sw_ssec_clk = 1'b0;
411                 o_sw_sec_clk = 1'b0;
412                 o_sw_min_clk = 1'b0;
413             end
414             POS_HOUR: begin
415                 o_sec_clk = clk_lhz;
416                 o_min_clk = i_max_hit_sec;
```

```
427     MODE_STOPWATCH : begin
428         case(o_stopwatch_en)
429             MODE_RESET: begin //reset
430                 o_sec_clk =clk_lhz ;
431                 o_min_clk = i_max_hit_sec;
432                 o_hour_clk = i_max_hit_min;
433                 o_alarm_sec_clk = 1'b0;
434                 o_alarm_min_clk = 1'b0;
435                 o_alarm_hour_clk = 1'b0;
436                 o_sw_ssec_clk = clk_lhz ;
437                 o_sw_sec_clk =clk_lhz ;
438                 o_sw_min_clk = clk_lhz ;
439             end
440             MODE_GO: begin //go
441                 o_sec_clk = clk_lhz ;
442                 o_min_clk = i_max_hit_sec;
443                 o_hour_clk = i_max_hit_min;
444                 o_alarm_sec_clk = 1'b0;
445                 o_alarm_min_clk = 1'b0;
446                 o_alarm_hour_clk = 1'b0;
447                 o_sw_ssec_clk = clk_100hz;
448                 o_sw_sec_clk = i_sw_hit_ssec;
449                 o_sw_min_clk = i_sw_hit_sec;
450             end
451             MODE_STOP:begin //stop
452                 o_sec_clk = clk_lhz ;
453                 o_min_clk = i_max_hit_sec;
```

```

465     MODE_SYDNEY_CLOCK : begin
466         o_sec_clk = clk_lhz;
467         o_min_clk = i_max_hit_sec;
468         o_hour_clk = i_max_hit_min; // when max-min, hour+1
469         o_alarm_sec_clk = 1'b0;
470         o_alarm_min_clk = 1'b0;
471         o_alarm_hour_clk = 1'b0;
472         o_sw_ssec_clk = 1'b0;
473         o_sw_sec_clk = 1'b0;
474         o_sw_min_clk = 1'b0;
475     end
476     MODE_LONDON_CLOCK : begin
477         o_sec_clk = clk_lhz;
478         o_min_clk = i_max_hit_sec;
479         o_hour_clk = i_max_hit_min; // when max-min, hour+1
480         o_alarm_sec_clk = 1'b0;
481         o_alarm_min_clk = 1'b0;
482         o_alarm_hour_clk = 1'b0;
483         o_sw_ssec_clk = 1'b0;
484         o_sw_sec_clk = 1'b0;
485         o_sw_min_clk = 1'b0;
486     end
487
488     default: begin
489         o_sec_clk = 1'b0;
490         o_min_clk = 1'b0;
491         o_hour_clk = 1'b0;

```

```

212     // at blink using
213
214     wire dis_hour ;
215
216     nco u_nco0(
217         .o_gen_clk ( dis_hour ),
218         .i_nco_num ( 32'd50000000 ), // 1 sec , clk change
219         .clk       ( clk ),
220         .rst_n    ( rst_n ) );
221
222     wire dis_min ;
223
224     nco u_ncol(
225         .o_gen_clk ( dis_min ),
226         .i_nco_num ( 32'd50000000 ), // 1 sec , clk change
227         .clk       ( clk ),
228         .rst_n    ( rst_n ) );
229
230     wire dis_sec ;
231
232     nco u_nco2(
233         .o_gen_clk ( dis_sec ),
234         .i_nco_num ( 32'd50000000 ), // 1 sec , clk change
235         .clk       ( clk ),
236         .rst_n    ( rst_n ) );
237
238

```

# 모드 0에서의 시계(Digital Clock)

- 모드 0에서 정상적으로 동작할 경우 1초 단위로 증가하면서 현재 시간을 출력한다.
- 시, 분, 초 단위로 시간을 나타내기 위해 HMS(Hour:Min:Sec) Counter에서 레지스터를 선언할 때 max 값 59임을 고려하여 7비트를 선언한다.
- 스톱워치의 ssec을 받기 위해서 7비트를 선언한다.
- 1Hz 클럭에 동기하여 초 단위 레지스터를 하나씩 늘리다가 60초가 되는 동시에 분 레지스터를 증가시키고, 초 레지스터는 0으로 초기화한다.
- 마찬가지로 분 레지스터가 60분이 되면 시간 레지스터를 하나 증가시키고 분 레지스터는 0으로 초기화된다.
- Hourminsec 모듈에서 시계 동작, 알람 동작, 스톱워치 동작을 하는 서로 다른 input을 만들었다.

```
651 output [6:0] o_sec ;  
652 output [6:0] o_min ;  
653 output [6:0] o_hour ; // hour(hour : 0-24, but o_hms_cnt is 6-bit, so o_hour need 6-bit)  
654  
655 output o_alarm1 ;  
656 output o_alarm2 ;  
657  
658 output o_max_hit_sec ;  
659 output o_max_hit_min ;  
660 output o_max_hit_hour ; // max_hit_hour  
661  
662 output o_sw_hit_ssec ;  
663 output o_sw_hit_sec ;  
664 output o_sw_hit_min ; // sw_hit  
665  
666 input [2:0] i_mode ;  
667 input [1:0] i_position ;  
668  
669  
670 input i_sec_clk ;  
671 input i_min_clk ;  
672 input i_hour_clk ; // hour_clk  
673  
674 input i_sw_ssec_clk ;  
675 input i_sw_sec_clk ;  
676 input i_sw_min_clk ; // sw_clk  
677
```

```

678 input i_alarm_sec_clk ;
679 input i_alarm_min_clk ;
680 input i_alarm_hour_clk; // i_alarm_hour_clk
681
682 input i_alarm_en1 ;
683 input i_alarm_en2 ;
684
685 input [1:0] i_stopwatch_en ; // sw-en
686
687 input i_am_pm_en ;
688
689 input clk ;
690 input rst_n ;
691
692 parameter MODE_CLOCK = 3'd0 ;
693 parameter MODE_SETUP = 3'd1 ;
694 parameter MODE_ALARM = 3'd2 ;
695 parameter MODE_STOPWATCH = 3'd3 ;
696 parameter MODE_SYDNEY_CLOCK = 3'd4; // world clock ( sydney )
697 parameter MODE_LONDON_CLOCK = 3'd5; // world clock ( london )
698
699
700 parameter POS_SEC = 2'd0 ;
701 parameter POS_MIN = 2'd1 ;
702 parameter POS_HOUR = 2'd2 ;
703
704

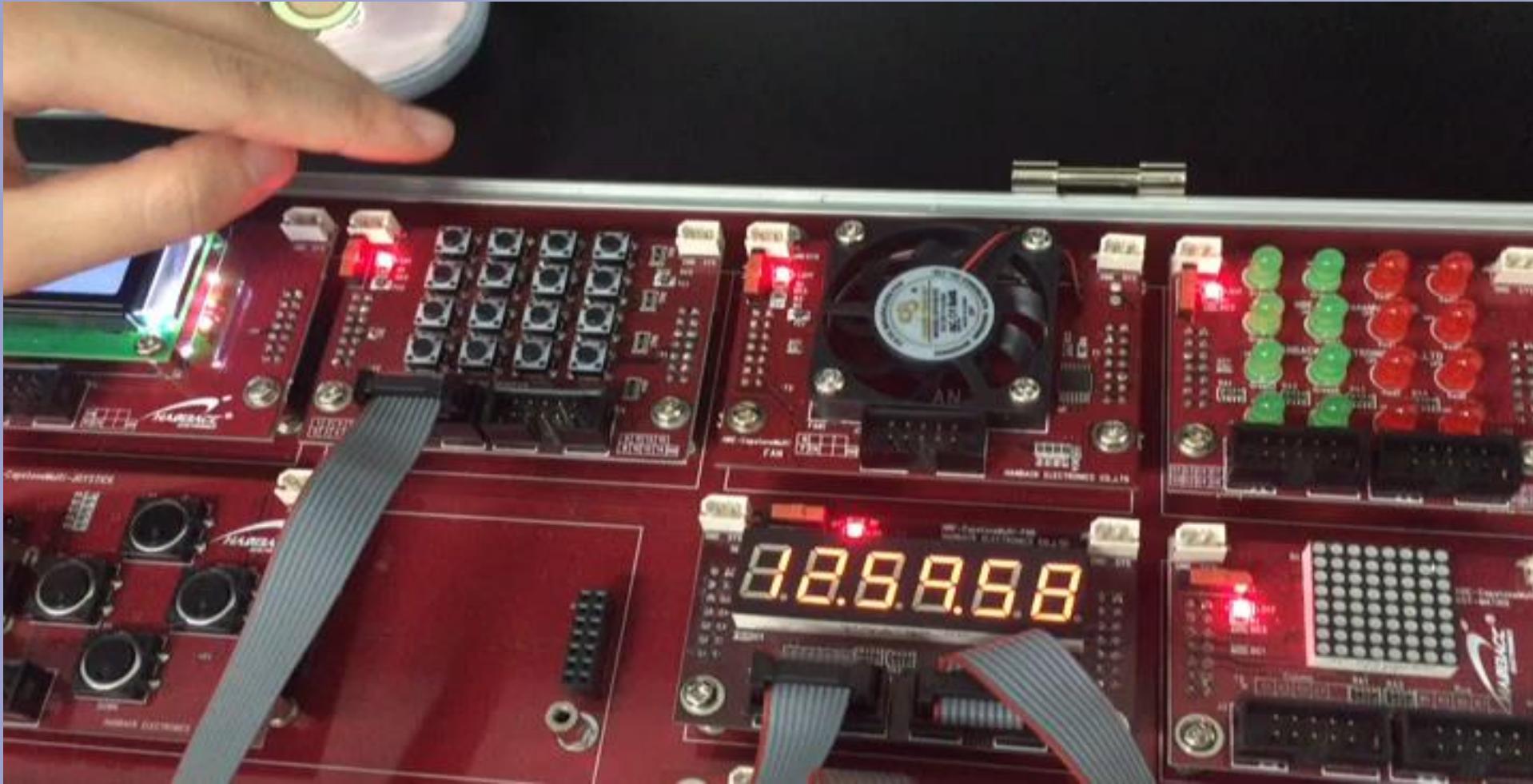
```

```

714 .clk ( i_sec_clk ), ;
715 .rst_n ( rst_n ), ;
716 .i_en ( ) ;
717 );
718
719 wire [6:0] min ;
720 wire o_max_hit_min ;
721
722 hms_cnt ul_hms_cnt( // ul : min
723 .o_hms_cnt ( min ), ;
724 .o_max_hit ( o_max_hit_min ), ;
725 .i_max_cnt ( 7'd59 ), // min(0-59)
726 .clk ( i_min_clk ), ;
727 .rst_n ( rst_n ), ;
728 .i_en ( ) ;
729 );
730
731 wire [6:0] hour ;
732 wire o_max_hit_hour ;
733
734 hms_cnt u2_hms_cnt( // u2 : hour
735 .o_hms_cnt ( hour ), ;
736 .o_max_hit ( o_max_hit_hour ), ;
737 .i_max_cnt ( 7'd23 ), // hour(0-23) |
738 .clk ( i_hour_clk ), ;
739 .rst_n ( rst_n ), ;
740 .i_en ( ) ;

```

# “ 모드0에서의 DigitalClock ”



## 모드 2에서의 알람(Alarm)

- 알람 기능을 이용하기 위해서는 알람 기능을 ON/OFF시키는 동작이 필요하므로 SW3을 이용하여 알람을 ON/OFF 할 수 있도록 한다.
- 알람 기능을 사용하려면 알람 시간을 설정해야 하는데 모드 2에서 해당 시간 위치의 알람 설정이 인에블되면 SW2를 눌러 알람 시간 설정을 할 수 있다.
- SW2를 한 번 누를 때마다 시, 분, 초 값이 하나씩 증가한다.
- 모드가 알람 모드일 때, 원하는 시간을 설정해야 하므로 알람 설정이 원하는 시간의 위치여야 한다.
- 초의 경우, SW2를 누를 때마다 1초 씩 증가하며, 59 다음에는 다시 0으로 초기화된다. 분과 시도 동일하다.
- 현재 시간과 설정된 알람 시간이 일치할 경우 알람 신호를 출력한다.
- 결과적으로 알람의 출력은 buzzer의 melody이다.

## 정각&알람 출력의 부저(Buzzer)

- Hourminsec 모듈 부분에서 코드를 수정하여 정각에 buzzer가 울리도록, 알람 맞춘 시각에 buzzer 울리도록 한다.
- 정각에는 사이렌 소리가 울리고, 알람 맞춘 시각에는 ‘엘리제’를 위하여 ‘ ’ 가 울린다.
- 알람을 정각으로 맞추었을 때에는 정각 buzzer보다 알람 buzzer가 우선시되도록 한다.
- alarm\_en\_1=정각 모드, alarm\_en\_2=알람 모드로 2가지를 만들어 출력이 다르게 나오도록 한다.
- alarm\_en\_1과 alarm\_en\_2의 시그널이 ON인지 OFF인지 구분하기 위하여 AM, PM 스위치 누르지 않았을 때에 한하여 dp로 표현한다.  
○ ○ ○ ○ ● ●에서 왼쪽 dp는 alarm\_en\_1이고, 오른쪽 dp는 alarm\_en\_2를 나타낸다.

```

918     endcase
919 end
920
921 reg      o_alarm1      ;
922 reg      o_alarm2      ;
923
924
925 always @ (posedge clk or negedge rst_n) begin
926   if (rst_n == 1'b0) begin
927     o_alarm1 <= 1'b0;
928     o_alarm2 <= 1'b0;
929   end else begin
930     if( (sec == 7'd0) && (min == 7'd0) && (hour != 7'd0) ) begin
931       o_alarm1 <= 1'bl & i_alarm_en1;
932     end else begin
933       o_alarm1 <= o_alarm1 & i_alarm_en1;
934     end
935
936     if( (sec == alarm_sec) && (min == alarm_min) && (hour == alarm_hour)) begin
937       o_alarm2 <= 1'bl & i_alarm_en2;
938     end else begin
939       o_alarm2 <= o_alarm2 & i_alarm_en2;
940     end
941   end
942 end
943
944 endmodule

```

```

957 input      i_buzz_en1      ;
958 input      i_buzz_en2      ;
959
960 input      clk            ;
961 input      rst_n          ;
962
963 parameter HIGH_DO = 23889 ;
964 parameter HIGH_RE = 21285 ;
965 parameter HIGH_RE_SHARP = 20088 ;
966 parameter HIGH_MI = 18960 ;
967 parameter SI = 25316 ;
968 parameter RA = 28409 ;
969 parameter DO = 47801 ;
970 parameter MI = 37936 ;
971 parameter REST = 250 ;
972
973 wire       clk_bit        ;
974 nco       u_nco_bit(
975           .o_gen_clk    ( clk_bit      ),
976           .i_nco_num   ( 25000000 ),
977           .clk          ( clk         ),
978           .rst_n        ( rst_n      ));
979
980 reg      [4:0]  cnt          ;
981 always @ (posedge clk_bit or negedge rst_n) begin
982   if(rst_n == 1'b0) begin
983     cnt <= 5'd0;

```

```
981     always @ (posedge clk_bit or negedge rst_n) begin
982         if(rst_n == 1'b0) begin
983             cnt <= 5'd0;
984         end else begin
985             if(cnt >= 5'd24) begin
986                 cnt <= 5'd0;
987             end else begin
988                 cnt <= i_buzz_en2 ;
989                 cnt <= cnt + 1'd1;
990             end
991         end
992     end
993
994     reg [4:0] h_cnt ;
995
996     always @ (posedge clk_bit or negedge rst_n) begin
997         if(rst_n == 1'b0) begin
998             h_cnt <= 5'd0;
999         end else begin
1000            if(h_cnt >= 5'd24) begin
1001                h_cnt <= 5'd0;
1002            end else begin
1003                h_cnt <= i_buzz_enl ;
1004                h_cnt <= h_cnt + 1'd1;
1005            end
1006        end
1007    end
```

```
1012     always @ (*) begin
1013         case(h_cnt)
1014             5'd00: h_nco_num = MI;
1015             5'd01: h_nco_num = REST;
1016             5'd02: h_nco_num = MI;
1017             5'd03: h_nco_num = REST;
1018             5'd04: h_nco_num = MI;
1019             5'd05: h_nco_num = REST;
1020             5'd06: h_nco_num = MI;
1021             5'd07: h_nco_num = REST;
1022             5'd08: h_nco_num = MI;
1023             5'd09: h_nco_num = REST;
1024             5'd10: h_nco_num = MI;
1025             5'd11: h_nco_num = REST;
1026             5'd12: h_nco_num = MI;
1027             5'd13: h_nco_num = REST;
1028             5'd14: h_nco_num = MI;
1029             5'd15: h_nco_num = REST;
1030             5'd16: h_nco_num = MI;
1031             5'd17: h_nco_num = REST;
1032             5'd18: h_nco_num = MI;
1033             5'd19: h_nco_num = REST;
1034             5'd20: h_nco_num = MI;
1035             5'd21: h_nco_num = REST;
1036             5'd22: h_nco_num = MI;
1037             5'd23: h_nco_num = REST;
1038             5'd24: h_nco_num = MI;
```

```

1044
1045     reg [31:0] nco_num ;
1046
1047     always @ (*) begin
1048         case(cnt)
1049             5'd00: nco_num = HIGH_MI;
1050             5'd01: nco_num = HIGH_RE_SHARP;
1051             5'd02: nco_num = HIGH_MI;
1052             5'd03: nco_num = HIGH_RE_SHARP;
1053             5'd04: nco_num = HIGH_MI;
1054             5'd05: nco_num = SI;
1055             5'd06: nco_num = HIGH_RE;
1056             5'd07: nco_num = HIGH_DO;
1057             5'd08: nco_num = RA;
1058             5'd09: nco_num = REST;
1059             5'd10: nco_num = DO;
1060             5'd11: nco_num = MI;
1061             5'd12: nco_num = RA;
1062             5'd13: nco_num = SI;
1063             5'd14: nco_num = MI;
1064             5'd15: nco_num = HIGH_DO;
1065             5'd16: nco_num = SI;
1066             5'd17: nco_num = RA;
1067             5'd18: nco_num = REST;
1068         endcase
1069     end
1070

```

```

1071     wire buzzl ; // n'o clock ringing
1072
1073     nco u_nco_buzzl(
1074         .o_gen_clk ( buzzl ),
1075         .i_nco_num ( h_nco_num ),
1076         .clk ( clk ),
1077         .rst_n ( rst_n ));
1078
1079     wire buzz2 ;
1080
1081     nco u_nco_buzz2(
1082         .o_gen_clk ( buzz2 ),
1083         .i_nco_num ( nco_num ),
1084         .clk ( clk ),
1085         .rst_n ( rst_n ));
1086
1087     wire [1:0] i_buzz_en ;
1088     assign i_buzz_en = { i_buzz_en1, i_buzz_en2 };
1089
1090     reg o_buzz ;
1091
1092     always @ (*) begin
1093         case(i_buzz_en)
1094             2'b00 : o_buzz = 1'b0 ;
1095             2'b01 : o_buzz = buzz2 & i_buzz_en2;
1096             2'b10 : o_buzz = buzz1 & i_buzz_en1;
1097             2'b11 : o_buzz = buzz2 & i_buzz_en2; // alarm >>>> n'o clock
1098
1099

```

“ 정각 출력 ”



```
1076     .clk      ( clk      ),
1077     .rst_n    ( rst_n    ));
1078
1079 wire      buzz2      ;
1080
1081 nco      u_nco_buzz2(
1082     .o_gen_clk  ( buzz2      ),
1083     .i_nco_num  ( nco_num    ),
1084     .clk        ( clk      ),
1085     .rst_n    ( rst_n    ));
1086
1087 wire [1:0]  i_buzz_en      ;
1088 assign      i_buzz_en = { i_buzz_en1, i_buzz_en2 };
1089
1090 reg       o_buzz      ;
1091
1092 always @ (*) begin
1093   case(i_buzz_en)
1094     2'b00 : o_buzz = 1'b0 ;
1095     2'b01 : o_buzz = buzz2 & i_buzz_en2;
1096     2'b10 : o_buzz = buzz1 & i_buzz_en1;
1097     2'b11 : o_buzz = buzz2 & i_buzz_en2;
1098   default : ;
1099   endcase
1100 end
1101 endmodule
1102
```

“ 알람 출력 ”



# Dot\_place 표시

모드	시계	0	●	○	●	○	○	○
1	세팅	○	●	○	●	○	●	
2	알람	○	●	○	●	○	●	
3	스톱워치	○	○	○	●	○	○	

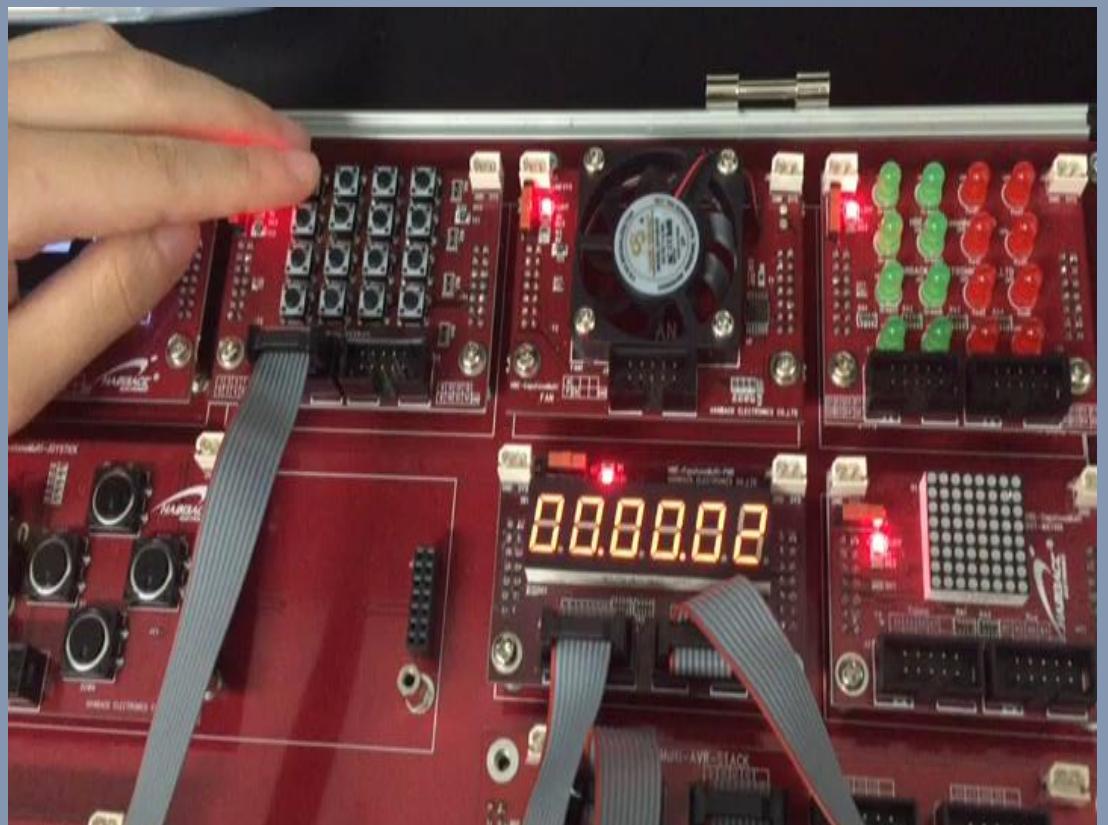
**모드 0인 경우** : 시, 분, 초 보기 좋게 나누어 놓음

**모드 1인 경우** : 세팅 시 선택된 부분 dp 켜짐

**모드 2인 경우** : 세팅 시 선택된 부분 dp 켜짐

**모드 3인 경우** : msec 앞에 dp 켜짐

“ Dot\_place ”



```
1103 module dot(
1104     mode,
1105     position,
1106     o_six_dp,
1107     blink_clk,
1108     rst_n,
1109     alarm_en1,
1110     alarm_en2,
1111     ampm_en
1112 );
1113 input [2:0] mode;
1114 input [1:0] position;
1115 input blink_clk;
1116 input rst_n;
1117 input alarm_en1;
1118 input alarm_en2;
1119 input ampm_en;
1120 output [5:0] o_six_dp;
1121 reg [5:0] o_six_dp;
1122 wire [1:0] alarm_en;
```

```
1130 assign alarm_en = { alarm_en1, alarm_en2 };
1131
1132 always @(posedge blink_clk or negedge rst_n) begin
1133 if(rst_n == 1'b0) begin
1134     o_six_dp <= 6'b00_00_00;
1135 end else begin
1136     case(mode)
1137         3'd0 : begin
1138             if(ampm_en == 1'b0) begin
1139                 case(alarm_en)
1140                     2'b00 : o_six_dp = 6'b01_01_00;
1141                     2'b01 : o_six_dp = 6'b01_01_01;
1142                     2'b10 : o_six_dp = 6'b01_01_10;
1143                     2'b11 : o_six_dp = 6'b01_01_11;
1144             endcase
1145         end else begin
1146             o_six_dp = 6'b11_11_11;
1147         end
1148     end
1149     3'd1 : begin //setting_mode
1150         case(position)
1151             2'b00 : o_six_dp = 6'b00_00_01;
1152             2'b01 : o_six_dp = 6'b00_01_00;
1153             2'b10 : o_six_dp = 6'b01_00_00;
1154             default : o_six_dp = 6'b00_00_00;
1155         endcase
1156     end
1157 end
```

```
54
55     end
56     3'd2: begin                                // alarm mode
57         if(alarm_en2 == 1'b0) begin
58             case(position)
59                 2'b00 : o_six_dp = 6'b00_00_01 ;
60                 2'b01 : o_six_dp = 6'b00_01_00 ;
61                 2'b10 : o_six_dp = 6'b01_00_00 ;
62                 default : o_six_dp = 6'b00_00_00 ;
63             endcase
64         end else begin
65             case(position)                         // when alarm_en =1, alarm ringing dp[5]=on
66                 2'b00 : o_six_dp = 6'b10_00_01 ;
67                 2'b01 : o_six_dp = 6'b10_01_00 ;
68                 2'b10 : o_six_dp = 6'b11_00_00 ;
69                 default : o_six_dp = 6'b00_00_00 ;
70             endcase
71         end
72     end
73     3'd3 : o_six_dp = 6'b00_01_00;
74     3'd4 : o_six_dp = 6'b01_01_00 ;
75     3'd5 : o_six_dp = 6'b01_01_00 ;
76     default : o_six_dp <= 6'b00_00_00 ;
77 endcase
78 end
79
```

# 블링크(Blink) 동작

모드 1,2에서 시간 설정 시, 설정 시간 위치의 블링크(Blink) 동작

```

1382  input      clk          ;
1383  input      rst_n       ;
1384
1385  input      i_dis_hour  ;
1386  input      i_dis_min   ;
1387  input      i_dis_sec   ;
1388
1389  output     o_dis_hour;
1390  output     o_dis_min;
1391  output     o_dis_sec ;
1392
1393
1394  output     blink        ;
1395  output     blink_clk    ;
1396
1397  input [2:0] setting_mode;
1398  input [1:0] setting_position;
1399
1400  wire      blink_clk    ;
1401
1402  nco       blink_clk_ul(
1403    .o_gen_clk  ( blink_clk      ),
1404    .i_nco_num  ( 32'd25000000 ),
1405    .clk         ( clk           ),
1406    .rst_n       ( rst_n         ));
1407
1408  reg       blink        ;

```

## WHY 블링크(Blink) ? HOW 블링크(Blink) !

- ✓ 이 기능은 7-세그먼트 FND 출력에서 현재 모드, 시간 설정 위치, 설정 스위치가 눌린 상태를 감지하여 해당하는 위치의 FND를 일정한 간격으로 ON/OFF하는 것이다.
- ✓ 블링크가 '1'인 동시에 모드가 1 또는 2일 때만 해당 출력 장치가 블링크 동작하도록 한다.
- ✓ 시간 설정 모드(모드 1) 또는 알람 설정 모드(모드 2)가 되면 해당 설정 시간 단위의 세그먼트는 블링크를 시작 한다.
- ✓ 블링크 모듈을 불러와 LED 모듈에서 출력하는 부분을 수정해야 하는데, 블링크 ON 조건과 일치하지 않는 경우 세그먼트는 항상 ON 상태인 것이다.

```

1412    always @ (posedge blink_clk or negedge rst_n) begin
1413        if(rst_n == 1'b0) begin                                // start 0
1414            blink     <= 1'b0;
1415        end else begin
1416            if((setting_mode == 3'd1)|| (setting_mode == 3'd2)) begin
1417                blink <= 1'bl;
1418            end else begin
1419                blink <= 1'b0;
1420            end
1421        end
1422    end
1423
1424    reg      o_dis_hour;
1425    reg      o_dis_min;
1426    reg      o_dis_sec ;
1427
1428    always @ ( posedge clk) begin
1429        if( (setting_mode == 3'd1)|| (setting_mode == 3'd2)) begin
1430            if(blink == 1'bl) begin
1431                case(setting_position)
1432                    2'b00 :begin                                // sec
1433                        o_dis_hour<= 1'bl;
1434                        o_dis_min <= 1'bl;
1435                        o_dis_sec <= ~i_dis_sec;
1436                    end
1437                    2'b01 :begin                                // min
1438                        o_dis_hour<= 1'bl;

```

```

1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
end
2'b01 :begin                                // min
o_dis_hour<= 1'bl;
o_dis_min <= ~i_dis_min;
o_dis_sec <= 1'bl;
end
2'b10 :begin                                // hour
o_dis_hour<= ~i_dis_hour;
o_dis_min <= 1'bl;
o_dis_sec <= 1'bl;
end
default : ;
endcase
end else begin
o_dis_hour<= 1'bl;
o_dis_min <= 1'bl;
o_dis_sec <= 1'bl;
end
end else begin
o_dis_hour<= 1'bl;
o_dis_min <= 1'bl;
o_dis_sec <= 1'bl;
end
end
endmodule

```

```

1 // -----
2 //      Flexible Numerical Display Decoder
3 // -----
4 `timescale 1ns/1ns
5
6 module fnd_dec(
7     clk,
8     hour10,
9     hour0,
10    min10,
11    min0,
12    sec10,
13    sec0,
14    blink,
15    blink_clk,
16    dis_hour,
17    dis_min,
18    dis_sec,
19    i_six_digit_seg
20 );
21
22 output [41:0] i_six_digit_seg ;
23
24 input   clk      ;
25
26 input [3:0] hour10 ;
27 input [3:0] hour0  ;
28 input [3:0] min10  ;
29 input [3:0] min0   ;
30 input [3:0] sec10  ;
31 input [3:0] sec0   ;
32
33 input   dis_hour ;
34 input   dis_min  ;
35 input   dis_sec  ;
36
37 input   blink    ;
38

```

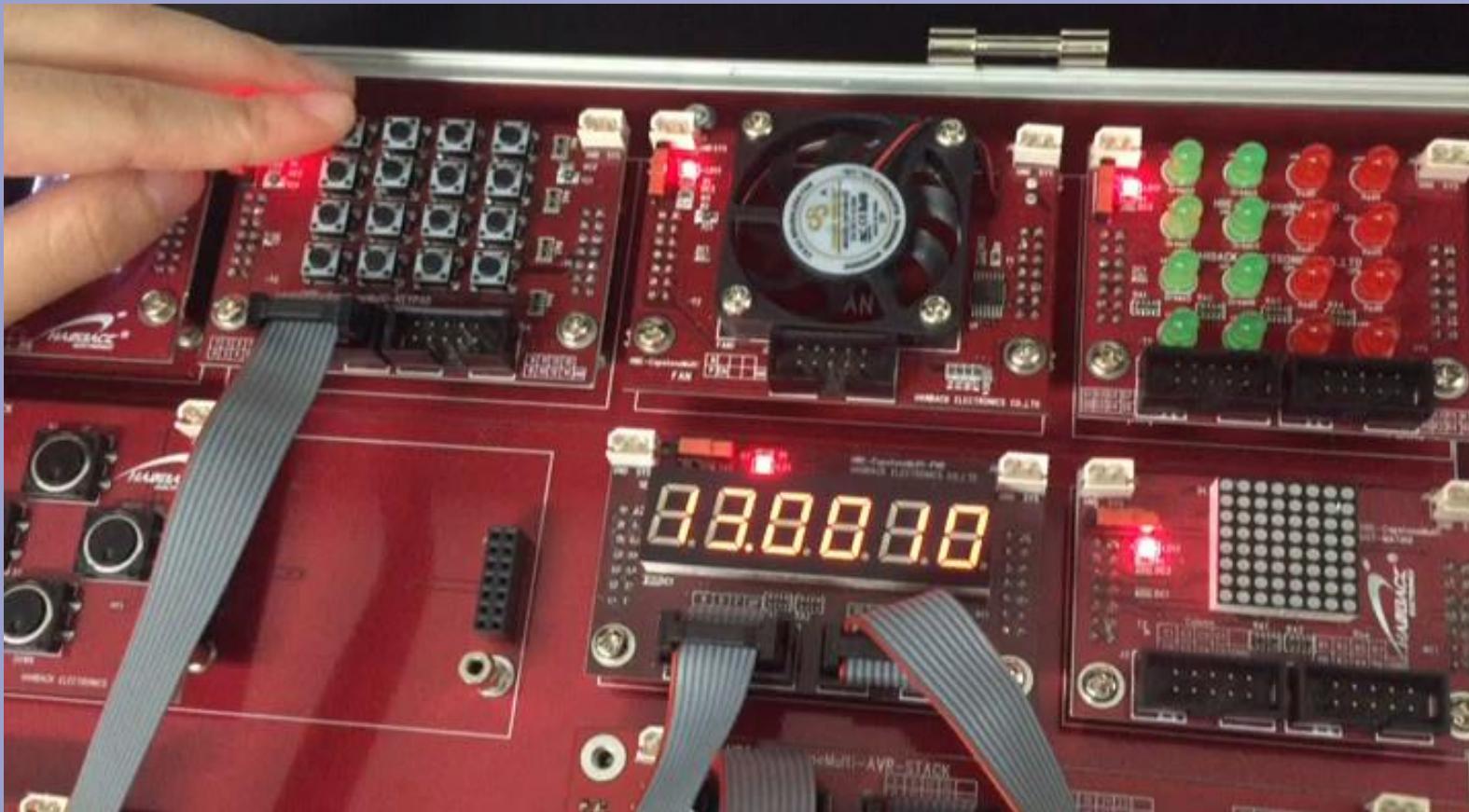
```

55
56 always @ (posedge blink_clk) begin
57   if(dis_hour == 1'b1) begin
58     case(hour10)
59       4'd0 : i_six_digit_seg[41:35] <= 7'b111_1110 ;
60       4'd1 : i_six_digit_seg[41:35] <= 7'b011_0000 ;
61       4'd2 : i_six_digit_seg[41:35] <= 7'b110_1101 ;
62       4'd3 : i_six_digit_seg[41:35] <= 7'b111_1001 ;
63       4'd4 : i_six_digit_seg[41:35] <= 7'b011_0011 ;
64       4'd5 : i_six_digit_seg[41:35] <= 7'b101_1011 ;
65       4'd6 : i_six_digit_seg[41:35] <= 7'b101_1111 ;
66       4'd7 : i_six_digit_seg[41:35] <= 7'b111_0000 ;
67       4'd8 : i_six_digit_seg[41:35] <= 7'b111_1111 ;
68       4'd9 : i_six_digit_seg[41:35] <= 7'b111_0011 ;
69     default : ;
70   endcase
71
72   case(hour0)
73     4'd0 : i_six_digit_seg[34:28] <= 7'b111_1110 ;
74     4'd1 : i_six_digit_seg[34:28] <= 7'b011_0000 ;
75     4'd2 : i_six_digit_seg[34:28] <= 7'b110_1101 ;
76     4'd3 : i_six_digit_seg[34:28] <= 7'b111_1001 ;
77     4'd4 : i_six_digit_seg[34:28] <= 7'b011_0011 ;
78     4'd5 : i_six_digit_seg[34:28] <= 7'b101_1011 ;
79     4'd6 : i_six_digit_seg[34:28] <= 7'b101_1111 ;
80     4'd7 : i_six_digit_seg[34:28] <= 7'b111_0000 ;
81     4'd8 : i_six_digit_seg[34:28] <= 7'b111_1111 ;
82     4'd9 : i_six_digit_seg[34:28] <= 7'b111_0011 ;
83   default : ;
84   endcase
85
86 end else begin
87   i_six_digit_seg[41:35] <= 7'b000_0000 ;
88   i_six_digit_seg[34:28] <= 7'b000_0000 ;
89
90

```

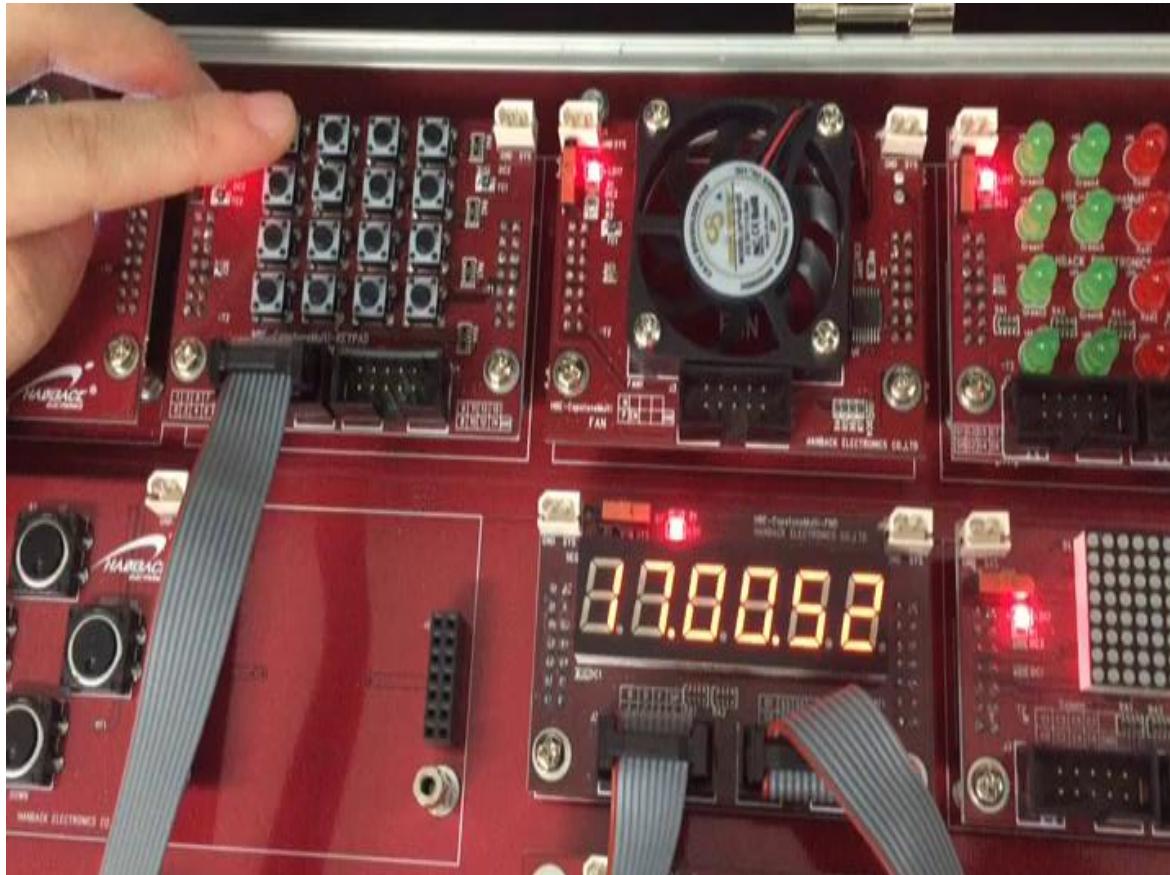
```
90
91     if(dis_min == 1'b1) begin
92         case(min10)
93             4'd0 : i_six_digit_seg[27:21] <= 7'b111_1110 ;
94             4'd1 : i_six_digit_seg[27:21] <= 7'b011_0000 ;
95             4'd2 : i_six_digit_seg[27:21] <= 7'b110_1101 ;
96             4'd3 : i_six_digit_seg[27:21] <= 7'b111_1001 ;
97             4'd4 : i_six_digit_seg[27:21] <= 7'b011_0011 ;
98             4'd5 : i_six_digit_seg[27:21] <= 7'b101_1011 ;
99             4'd6 : i_six_digit_seg[27:21] <= 7'b101_1111 ;
100            4'd7 : i_six_digit_seg[27:21] <= 7'b111_0000 ;
101            4'd8 : i_six_digit_seg[27:21] <= 7'b111_1111 ;
102            4'd9 : i_six_digit_seg[27:21] <= 7'b111_0011 ;
103        default : ;
104    endcase
105
106    case(min0)
107        4'd0 : i_six_digit_seg[20:14] <= 7'b111_1110 ;
108        4'd1 : i_six_digit_seg[20:14] <= 7'b011_0000 ;
109        4'd2 : i_six_digit_seg[20:14] <= 7'b110_1101 ;
110        4'd3 : i_six_digit_seg[20:14] <= 7'b111_1001 ;
111        4'd4 : i_six_digit_seg[20:14] <= 7'b011_0011 ;
112        4'd5 : i_six_digit_seg[20:14] <= 7'b101_1011 ;
113        4'd6 : i_six_digit_seg[20:14] <= 7'b101_1111 ;
114        4'd7 : i_six_digit_seg[20:14] <= 7'b111_0000 ;
115        4'd8 : i_six_digit_seg[20:14] <= 7'b111_1111 ;
116        4'd9 : i_six_digit_seg[20:14] <= 7'b111_0011 ;
117        default : ;
118    endcase
119 end else begin
120     i_six_digit_seg[27:21] <= 7'b000_0000 ;
121     i_six_digit_seg[20:14] <= 7'b000_0000 ;
122 end
123
124     if(dis_sec == 1'b1) begin
125         case(sec10)
126             4'd0 : i_six_digit_seg[13:7] <= 7'b111_1110 ;
127             4'd1 : i_six_digit_seg[13:7] <= 7'b011_0000 ;
128             4'd2 : i_six_digit_seg[13:7] <= 7'b110_1101 ;
129             4'd3 : i_six_digit_seg[13:7] <= 7'b111_1001 ;
130             4'd4 : i_six_digit_seg[13:7] <= 7'b011_0011 ;
131             4'd5 : i_six_digit_seg[13:7] <= 7'b101_1011 ;
132             4'd6 : i_six_digit_seg[13:7] <= 7'b101_1111 ;
133             4'd7 : i_six_digit_seg[13:7] <= 7'b111_0000 ;
134             4'd8 : i_six_digit_seg[13:7] <= 7'b111_1111 ;
135             4'd9 : i_six_digit_seg[13:7] <= 7'b111_0011 ;
136         default : ;
137     endcase
138
139     case(sec0)
140         4'd0 : i_six_digit_seg[6:0] <= 7'b111_1110 ;
141         4'd1 : i_six_digit_seg[6:0] <= 7'b011_0000 ;
142         4'd2 : i_six_digit_seg[6:0] <= 7'b110_1101 ;
143         4'd3 : i_six_digit_seg[6:0] <= 7'b111_1001 ;
144         4'd4 : i_six_digit_seg[6:0] <= 7'b011_0011 ;
145         4'd5 : i_six_digit_seg[6:0] <= 7'b101_1011 ;
146         4'd6 : i_six_digit_seg[6:0] <= 7'b101_1111 ;
147         4'd7 : i_six_digit_seg[6:0] <= 7'b111_0000 ;
148         4'd8 : i_six_digit_seg[6:0] <= 7'b111_1111 ;
149         4'd9 : i_six_digit_seg[6:0] <= 7'b111_0011 ;
150     default : ;
151     endcase
152
153 end else begin
154     i_six_digit_seg[13:7] <= 7'b000_0000 ;
155     i_six_digit_seg[6:0] <= 7'b000_0000 ;
156 end
157
158
```

“ 모드1 & 모드2에서의 블링크 ”



# 세계 시간 출력

우리 나라 기준으로 +2시간 시차가 나는 시드니와  
-9시간 시차가 나는 런던 시간을 출력



## 세계 시간 출력

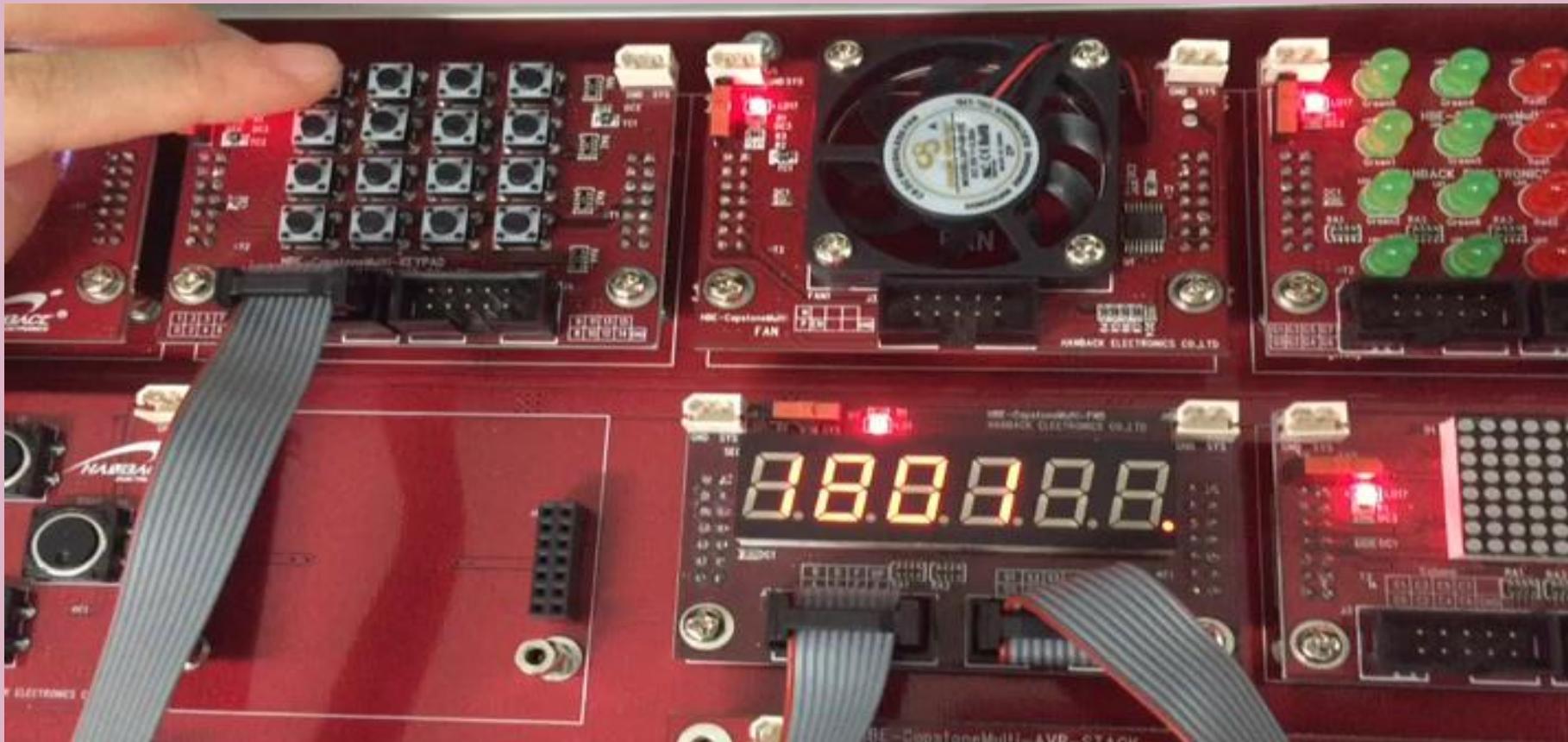
- ✓ 우리 나라 기준으로 시각이 (+)인 나라, 시각이 (-)인 두 나라(시드니와 런던)를 설정한다.
- ✓ 시드니는 우리 나라 기준으로 +2시간, 런던은 우리 나라 기준으로 -9시간이다.
- ✓ 원래 clock에 우리 나라 시간을 입력하면 그 시간을 기준으로 시드니 시간(mode 4)과 런던 시간(mode 5)을 나타낸다.
- ✓ Hms 모듈에서 시드니 시간을 mode 4로, 런던 시간을 mode 5로 나타낸다.

```
840 //      MODE_SYDNEY_CLOCK
841
842 reg [6:0] S_sec ;
843 reg [6:0] S_min ;
844 reg [6:0] S_hour ;
845
846 begin
847 if( hour >= 7'd22) begin
848     S_sec <= sec ;
849     S_min <= min ;
850     S_hour <= hour -7'd22;
851 end else begin
852     S_sec <= sec ;
853     S_min <= min ;
854     S_hour <= hour +7'd2;
855 end
856 end
857
858 //      MODE_LONDON_CLOCK
859
860 reg [6:0] L_sec ;
861 reg [6:0] L_min ;
862 reg [6:0] L_hour ;
863
864 begin
865 if( hour >= 9) begin
866     L_sec <= sec ;

```

```
854 S_hour <= hour +7'd2;
855 end
856 end
857
858 //      MODE_LONDON_CLOCK
859
860 reg [6:0] L_sec ;
861 reg [6:0] L_min ;
862 reg [6:0] L_hour ;
863
864 begin
865 if( hour >= 9) begin
866     L_sec <= sec ;
867     L_min <= min ;
868     L_hour <= hour -7'd9;
869 end else begin
870     L_sec <= sec ;
871     L_min <= min ;
872     L_hour <= hour +7'd15;
873 end
874 end
875
876 reg [6:0] o_sec ;
877 reg [6:0] o_min ;
878 reg [6:0] o_hour ;
879
```

# “모드3에서의 StopWatch”



```

507 //      RESET MODULE
508 module hms_cnt_cntstop(
509             o_hms_cnt,
510             o_max_hit,
511             i_max_cnt,
512             find_cnt, // [1:0] i_stopwatch_en -> [1:0] find_cnt
513             clk,
514             rst_n
515         );
516
517     output [6:0] o_hms_cnt;
518     output      o_max_hit;
519
520     input [6:0] i_max_cnt;
521     input [1:0] find_cnt;
522
523     input      clk;
524     input      rst_n;
525
526     reg [6:0] o_hms_cnt;
527     reg      o_max_hit;
528
529 always @(posedge clk or negedge rst_n) begin
530     if(rst_n == 1'b0) begin
531         o_hms_cnt <= 7'd0;
532         o_max_hit <= 1'b0;
533     end else begin

```

## 모드 3에서의 스톱워치 (StopWatch)

- 스톱워치의 가장 기본적인 동작은 1/100초 단위로 시간을 증가시키는 것이다. 따라서 100Hz 클럭을 받아 하나의 클럭이 들어올 때마다 1/100초 씩 증가시키고 그 값을 출력한다.
- 모드 3에서 현재 1/100초 단위로 카운트 하는 동작 상태이면 SW4을 눌렀을 때 카운트 증가, 카운트 일시 정지, 리셋(초기화) 상태로 바뀐다.
- 100Hz의 클럭에 동기하여 10msec씩 증가시키고 99를 카운트하면 다시 0으로 초기화되며 1초가 증가한다.

```

531     |     o_hms_cnt <= 7'd0;
532     |     o_max_hit <= 1'b0;
533 end else begin
534     |         case(find_cnt)
535     |             2'd0 : begin //reset
536                 |                 o_hms_cnt <= 7'd0 ;
537                 |                 o_max_hit <= 1'b0;
538             end
539             |             2'd1 : begin //go
540                 |                 if(o_hms_cnt >= i_max_cnt) begin
541                     |                     o_hms_cnt <= 7'd0;
542                     |                     o_max_hit <= 1'bl;
543                 end else begin
544                     |                     o_hms_cnt <= o_hms_cnt + 1'bl;
545                     |                     o_max_hit <= 1'b0;
546                 end
547             end
548             |             2'd2 : begin //stop
549                 |                 o_hms_cnt <= o_hms_cnt ;
550                 |                 o_max_hit <= 1'b0;
551             end
552         endcase
553     end
554 end
555
556 endmodule

```

```

803 // MODE_STOPWATCH
804
805 wire [6:0] sw_mmsec ;
806 wire     o_sw_hit_ssec ;
807
808 hms_cnt_cntstop u0_hms_cnt_stopwatch(          // u0 : ssec
809     .o_hms_cnt    ( sw_mmsec      ),
810     .o_max_hit    ( o_sw_hit_ssec ),
811     .i_max_cnt    ( 7'd99        ), // sec(0-59)
812     .clk          ( i_sw_ssec_clk ),
813     .find_cnt    ( i_stopwatch_en ),
814     .rst_n        ( rst_n       ));
815
816
817 wire [6:0] sw_sec ;
818 wire     o_sw_hit_sec ;
819
820 hms_cnt_cntstop ul_hms_cnt_stopwatch(          // ul :sec
821     .o_hms_cnt    ( sw_sec       ),
822     .o_max_hit    ( o_sw_hit_sec ),
823     .i_max_cnt    ( 7'd59        ),
824     .clk          ( i_sw_sec_clk ),
825     .find_cnt    ( i_stopwatch_en ),
826     .rst_n        ( rst_n       ));
827
828 wire [6:0] sw_min ;
829 wire     o_sw_hit_min ;

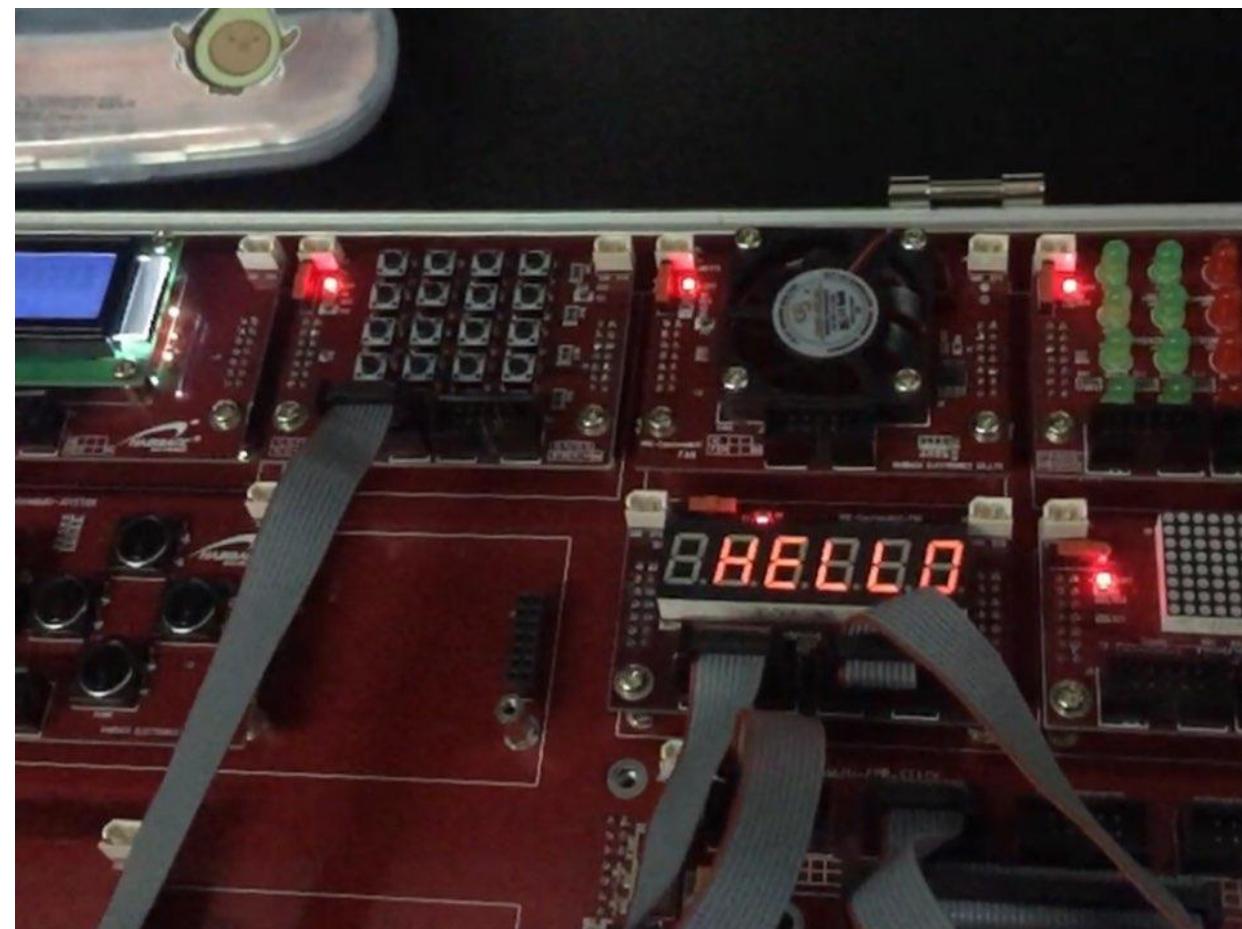
```

# 시계 모드 첫 화면에 HELLO 띄우기

- fnd\_dec 모듈 부분에 initial 코드 추가하여 만든다.

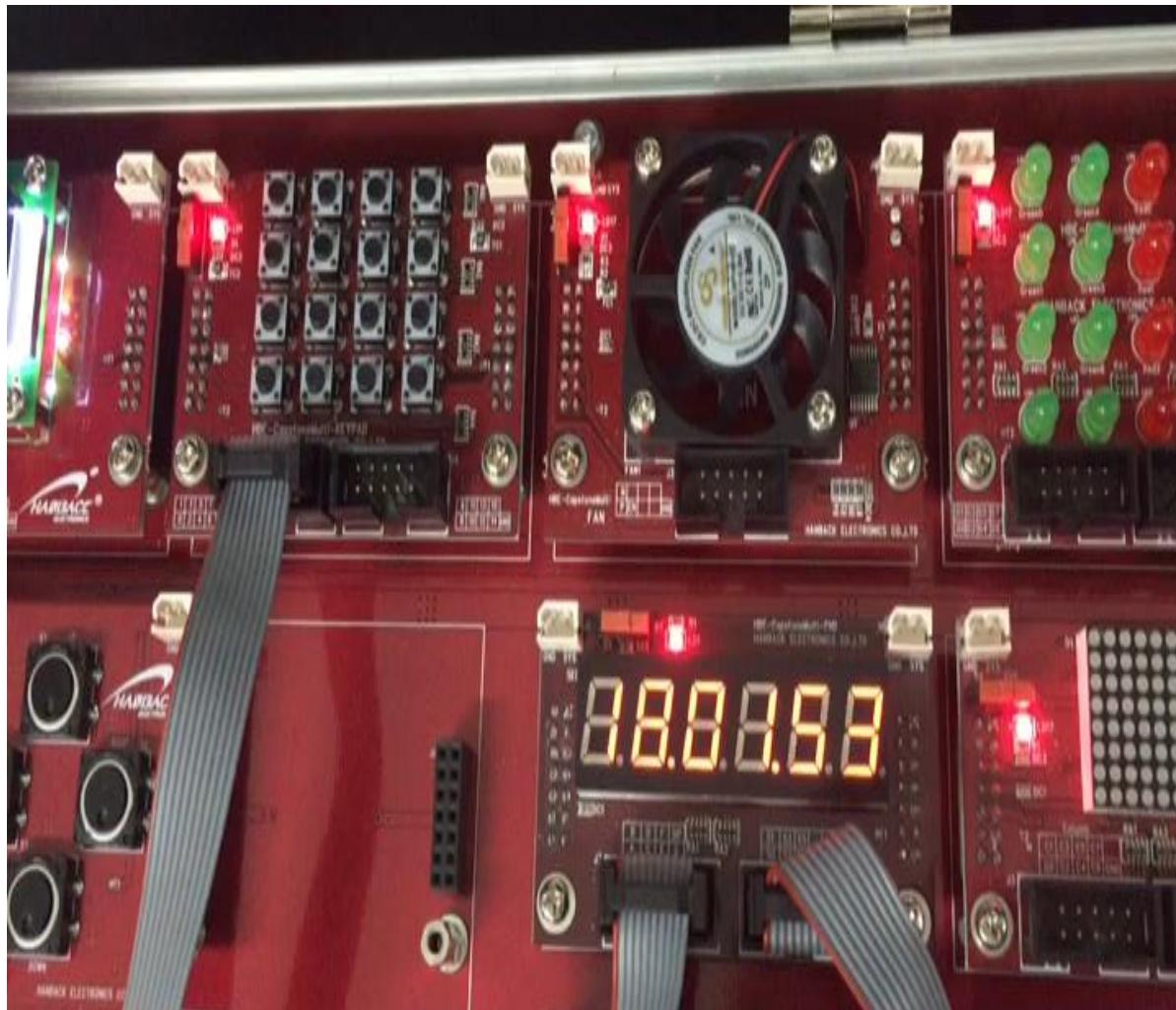
```
1226 parameter tck = 1000/50      ;
1227
1228 initial begin
1229 #(0*tck)      i_six_digit_seg[41:0]  = 42'd0      ;
1230 #(10*tck) begin
1231     i_six_digit_seg[41:35] = 7'b000_0000      ;      // H
1232     i_six_digit_seg[34:28] = 7'b011_0111      ;      // E
1233     i_six_digit_seg[27:21] = 7'b100_1111      ;      // L
1234     i_six_digit_seg[20:14] = 7'b000_1110      ;      // L
1235     i_six_digit_seg[13:7]  = 7'b000_1110      ;      // O
1236     i_six_digit_seg[6:0]   = 7'b111_1110      ;      // O
1237 end
1238 end
1239
1240 always @ (posedge blink_clk) begin
1241     if(dis_hour == 1'b1) begin
1242         case(hour10)
1243             4'd0 : i_six_digit_seg[41:35] <= 7'b111_1110      ;
1244             4'd1 : i_six_digit_seg[41:35] <= 7'b011_0000      ;
1245             4'd2 : i_six_digit_seg[41:35] <= 7'b110_1101      ;
1246             4'd3 : i_six_digit_seg[41:35] <= 7'b111_1001      ;
1247             4'd4 : i_six_digit_seg[41:35] <= 7'b011_0011      ;
```

DigitalClock을 시작할 때 HELLO



# CLOCK 모드일 때, AM or PM 출력

- 12시 이후에 SW 6을 누를 경우, dp가 다 켜짐으로써 PM으로 나타낼 수 있다.

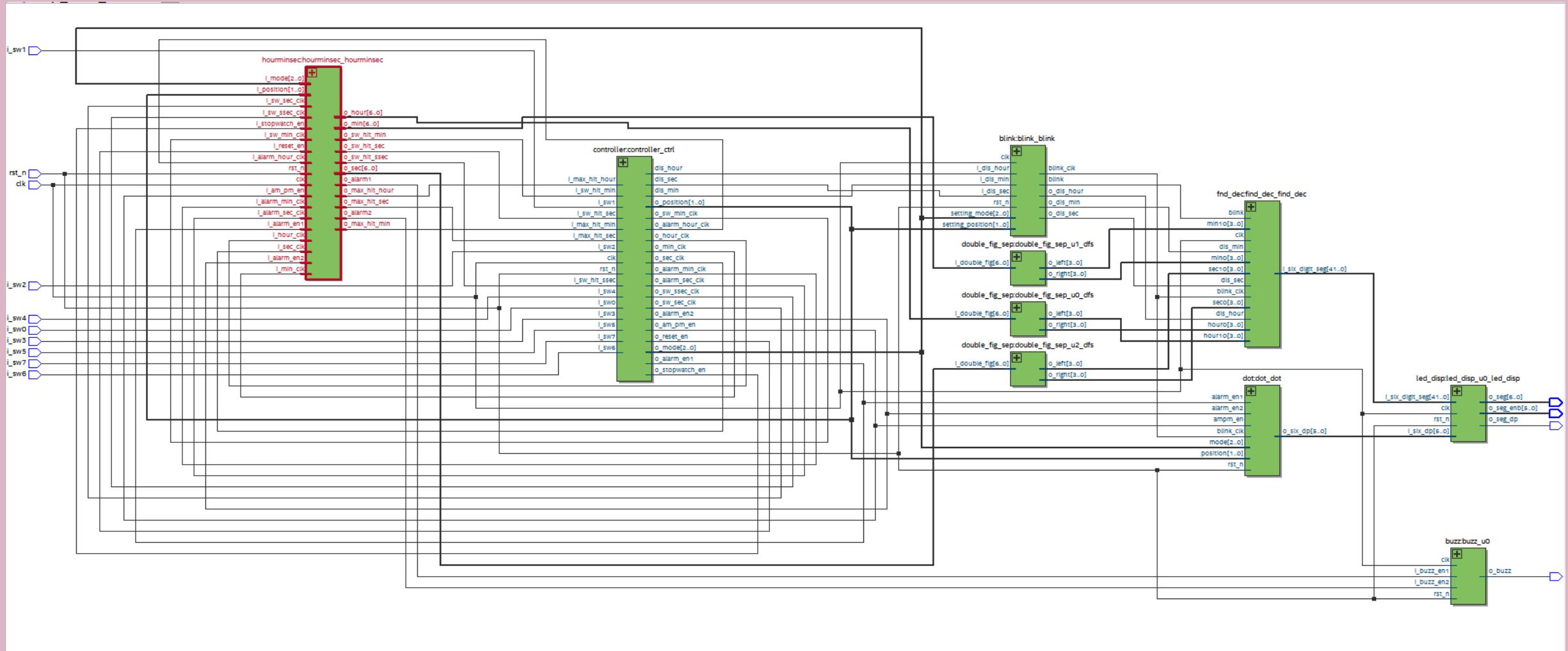


```
737     .i_max_cnt  ( 7'd23      ), // hour(0-23)
738     .clk        ( i_hour_clk ),
739     .rst_n     ( rst_n      ),
740     .i_en       (          )
741   );
742
743   reg [6:0]  apm_sec    ;
744   reg [6:0]  apm_min    ;
745   reg [6:0]  apm_hour   ;
746
747   always @ (*) begin
748     if( i_am_pm_en == 1'b0) begin
749       apm_sec <= sec  ;
750       apm_min <= min  ;
751       apm_hour<= hour ;
752     end else begin
753       if( hour >= 7'd12) begin
754         apm_sec <= sec  ;
755         apm_min <= min  ;
756         apm_hour<= hour - 7'd12 ;
757       end else begin
758         apm_sec <= sec  ;
759         apm_min <= min  ;
760         apm_hour<= hour ;
761     end
762   end
763 end
```

# 실험 과정 및 결과

Quartus RTL Viewer로 확인한 회로도

- ✓ 모듈 간 연결이 잘 되었는가?
- ✓ 수정해야 할 부분은 없는가?
- ✓ 구상한 회로도와 일치하는가?



“

고 찰

”

# 고찰

최종 설계 사양 및 프로젝트 완료 일정



Verilog HDL을 이용하여 설계한  
FPGA 디지털 시계 최종 설계 사양

- ✓ 시, 분, 초가 나타나는 DigitalClock
- ✓ Buzzer에서 멜로디가 울리는 Alarm
- ✓ 정각에 울리는 Buzzer
- ✓ 모드마다 Dot\_place 표시
- ✓ 처음 DigitalClock 화면에 'HELLO'
- ✓ 1/100초씩 시간이 증가하는 StopWatch
- ✓ 세계 시간 출력(시드니 & 런던)
- ✓ StopWatch에서의 start, stop, reset 기능
- ✓ DigitalClock에서 AM, PM 표현
- ✓ 모드 1과 모드 2에서의 Blink 기능

“

# Q & A

”

Thank You 😊