명지 공학인 윤리 서약서

보고서 및 논문 윤리 서약

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.
2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.
3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.
4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁받지도 않겠습니다.

나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

시험 윤리 서약

1. 나는 대리시험을 청탁하거나 청탁받지 않겠습니다.
2. 나는 허용되지 않은 교과서, 노트 및 타학생의 답안지 등을 보고 답안지를 작성하지 않겠습니다.
3. 나는 타인에게 답안지를 보여주지 않겠습니다.
4. 나는 감독관의 지시와 명령에 따라 시험 과정에 참여하겠습니다.

나는 시험에 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

2022년 11월 06일

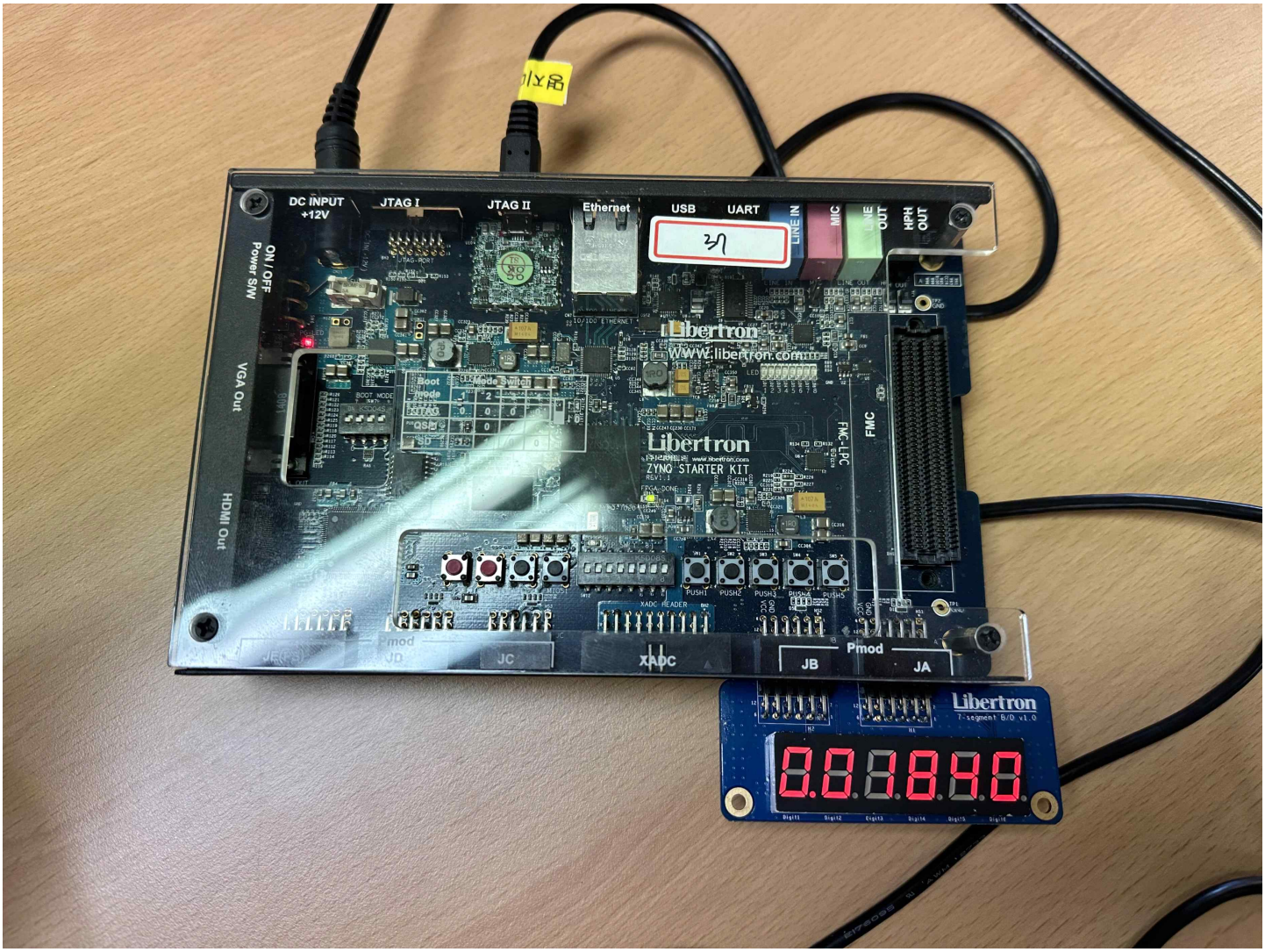
서약자

(학번) 60171878

(성명) 허무혁 **(인)**

디지털 시계 구현

60171878 허무혁



목차

1. Verilog 소스코드

top.v 코드

```
module top (
    input clk,
    input reset_poweron,
    input [3:0] btn,
    output reg [7:0] seg_data,
    output reg [5:0] seg_com
    );

    wire clk_6mhz;
    wire [6:0] sec0_out, sec1_out, min0_out, min1_out, hrs0_out, hrs1_out;
    wire [3:0] sec0, sec1, min0, min1, hrs0, hrs1;
    wire clock_en, clock_en1;
    reg [5:0] digit;
    wire left, right, up, down;
    wire [3:0] btn_pulse;
    wire locked, rst;
    //for PLL
    clk_wiz_0 clk_inst (clk_6mhz, reset_poweron, locked, clk); //for Zedboard
    //assign clk_6mhz = clk;  //for Simulation

    //for reset signal generation
    assign rst = reset_poweron | (~locked);
    //for speed control: SIZE=6000000(x1), SIZE=600000(x10), SIZE=6000(x1000)
    gen_counter_en #(.SIZE(6000000)) gen_clock_en_inst (clk_6mhz, rst, clock_en);
    clock clock_inst (clk_6mhz, rst, clock_en, digit, up, down, sec0, sec1, min0, min1, hrs0, hrs1);

    // for debouncing, use btn_pulse that is high for 1 cycle)
    debounce #(.BTN_WIDTH(4)) debounce_btn0_inst (clk_6mhz, rst, btn, ,btn_pulse);
    assign {down, up, right, left} = btn_pulse;
    //7-seg decoder
    dec7 dec_sec0_inst (sec0, sec0_out);
    dec7 dec_sec1_inst (sec1, sec1_out);
    dec7 dec_min0_inst (min0, min0_out);
    dec7 dec_min1_inst (min1, min1_out);
    dec7 dec_hrs0_inst (hrs0, hrs0_out);
    dec7 dec_hrs1_inst (hrs1, hrs1_out);
    //digit[5:0] generation code here with "left" or "right" button
    initial begin
        seg_com = 6'b100000;
        digit = 6'b100000;
    end
    always@(posedge clk_6mhz) begin
```

```
            if(left) begin
                digit[0] <= digit[1];
                digit[1] <= digit[2];
                digit[2] <= digit[3];
                digit[3] <= digit[4];
                digit[4] <= digit[5];
                digit[5] <= digit[0];
            end
            else if(right) begin
                digit[1] <= digit[0];
                digit[2] <= digit[1];
                digit[3] <= digit[2];
                digit[4] <= digit[3];
                digit[5] <= digit[4];
                digit[0] <= digit[5];
            end
        end
    //digit[5:0] = 100000,010000,001000,000100,000010,000001,100000,010000

    //seg_com[5:0] generation code here (shifts 600 times per second)
//seg_com신호를 위해 counter 600hz 신호를 하나 생성
    gen_counter_en #(.SIZE(10000)) gen_clock_en_1 (clk_6mhz, rst, clock_en1);
//seg_com신호를 600hz마다 한칸씩 shift
    always@(posedge clock_en1) begin
            seg_com[0] <= seg_com[1];
            seg_com[1] <= seg_com[2];
            seg_com[2] <= seg_com[3];
            seg_com[3] <= seg_com[4];
            seg_com[4] <= seg_com[5];
            seg_com[5] <= seg_com[0];
    end
    //seg_com[5:0] = 100000,010000,001000,000100,000010,000001,100000,010000
//case 문을 사용하여 seg_com이 shift 될 때마다 seg_data를 출력 이때 digit[]은 커서의 dot의 on/off 의미
    always @ (*) begin
        case (seg_com)
            6'b100000: seg_data = {sec0_out, digit[5]};
            6'b010000: seg_data = {sec1_out, digit[4]};
            6'b001000: seg_data = {min0_out, digit[3]};
            6'b000100: seg_data = {min1_out, digit[2]};
            6'b000010: seg_data = {hrs0_out, digit[1]};
            6'b000001: seg_data = {hrs1_out, digit[0]};
            default: seg_data = 8'b0;
        endcase
    end
```

```
endmodule
```

clk_wiz_0.v 코드

```
//clk_wiz_0는 ip를 사용하여 600mhz 신호를 만듬. ip를 이용했으므로 코드는 생략.
```

gen_counter_en.v 코드

```
module gen_counter_en (input clk, input rst, output counter_en);
//parameter SIZE와 counter를 이용하여 600mhz 클락을 변조함. default는 1hz로 바뀌는 신호.
parameter SIZE = 6000000;
reg [31:0] o;
always @(posedge clk or posedge rst)
    if (rst) o <= 0;
    else
        if (o == SIZE-1) o <= 0;
        else o <= o + 1;
assign counter_en = (o == SIZE-1)? 1'b1 : 1'b0;
endmodule
```

clock.v 테스트벤치 코드

```
module clock(clk_6mhz, rst, clock_en, digit, up, down, sec0, sec1, min0, min1, hrs0, hrs1);
//input으로 clock 신호와 reset 신호 up, down 버튼신호, digit 신호를 받음.
//output으로 digital watch의 시, 분, 초 신호를 내보냄.
//wire 내부신호를 통하여 각 시, 분, 초의 요소가 해당하는 자릿수를 넘어갈 때 clk 신호를 생성.
input clk_6mhz, rst, clock_en, up, down;
input [5:0] digit;
output [3:0] sec0, sec1, min0, min1, hrs0, hrs1;
reg [3:0] sec0, sec1, min0, min1, hrs0, hrs1;
wire trigger1, trigger2, trigger3, trigger4, trigger5, trigger6;

always@(posedge clk_6mhz or posedge rst) begin
    if(rst) sec0 <= 4'b0;
//digit 신호와 up 또는 down 신호가 1일 때 명령 수행.
    else if(digit[5] == 1'b1 && up == 1'b1) sec0 <= sec0 + 1;
    else if(digit[5] == 1'b1 && down == 1'b1) sec0 <= sec0 -1;
    else if(clock_en)
        if(sec0 == 9) begin sec0 <= 0; end
        else begin sec0 <= sec0 + 1; end
    end
//sec0가 9일 때 클럭을 하나 만듬.
assign trigger1 = (sec0 == 9) ? 1'b1 : 1'b0;

//위와 같으므로 각주 생략.
```

```
always@(posedge clk_6mhz or posedge rst) begin
    if(rst) sec1 <= 4'b0;
    else if(digit[4] == 1'b1 && up == 1'b1) sec1 <= sec1 + 1;
    else if(digit[4] == 1'b1 && down == 1'b1) sec1 <= sec1 - 1;
    else if(trigger1 && clock_en)
        if(sec1 == 5) begin sec1 <= 0; end
        else begin sec1 <= sec1 + 1; end
    end
 assign trigger2 = (sec1 == 5 && sec0==9 && clock_en) ? 1'b1 : 1'b0;

 always@(posedge clk_6mhz or posedge rst) begin
    if(rst) min0 <= 4'b0;
    else if(digit[3] == 1'b1 && up == 1'b1) min0 <= min0 + 1;
    else if(digit[3] == 1'b1 && down == 1'b1) min0 <= min0 - 1;
    else if(trigger2 && clock_en)
        if(min0 == 9) begin min0 <= 0;end
        else begin min0 <= min0 + 1; end
    end
 assign trigger3 = (min0 == 9 && sec1 == 5 && sec0==9 && clock_en) ? 1'b1 : 1'b0;

always@(posedge clk_6mhz or posedge rst) begin
    if(rst) min1 <= 4'b0;
    else if(digit[2] == 1'b1 && up == 1'b1) min1 <= min1 + 1;
    else if(digit[2] == 1'b1 && down == 1'b1) min1 <= min1 - 1;
    else if(trigger3 && clock_en)
        if(min1 == 5) begin min1 <= 0; end
        else begin min1 <= min1 + 1; end
    end
assign trigger4 = (min1 == 5 && min0 == 9 && sec1 == 5 && sec0==9 && clock_en) ? 1'b1 : 1'b0;

// “시”를 표현하기 위해서는 09, 19시 일때와 23시 일때를 구분지어 생각해야 함.
// 따라서 특별히 trigger 신호를 2개 사용해 두가지의 상황을 가정하여 코드를 작성함.
always@(posedge clk_6mhz or posedge rst) begin
    if(rst) hrs0 <= 4'b0;
    else if(digit[1] == 1'b1 && up == 1'b1) hrs0 <= hrs0 + 1;
    else if(digit[1] == 1'b1 && down == 1'b1) hrs0 <= hrs0 - 1;
    else if(trigger6 && clock_en) begin hrs0 <= 0;
    end
    else if(trigger4 && clock_en)
        if(hrs0 == 9) begin hrs0 <= 0; end
        else begin hrs0 <= hrs0 + 1; end
 end
assign trigger5 = (hrs0 == 9 && min1 == 5 && min0 == 9 && sec1 == 5 && sec0==9) ? 1'b1 : 1'b0;
assign trigger6 = (hrs1 == 2 && hrs0 == 3 && min1 == 5 && min0 == 9 && sec1 == 5 && sec0==9) ?
1'b1 : 1'b0;
```

```
always@(posedge clk_6mhz or posedge rst) begin
    if(rst) hrs1 <= 4'b0;
    else if(digit[0] == 1'b1 && up == 1'b1) hrs1 <= hrs1 + 1;
    else if(digit[0] == 1'b1 && down == 1'b1) hrs1 <= hrs1 - 1;
    else if(trigger5 && clock_en)
        if(hrs1 == 2) begin hrs1 <= 0; end
        else begin hrs1 <= hrs1 + 1; end
    else if(trigger6 && clock_en) hrs1 <= 0;
 end

endmodule
```

debounce.v 코드

```
module debounce(clk, rst, btn_in, btn_out, btn_out_pulse);
parameter SIZE = 16; //if pressed for 1/clk*2^(SIZE-1)sec, it can be debounced. 5.46ms for 6MHz
parameter BTN_WIDTH = 5;
input clk, rst;
input [BTN_WIDTH-1:0] btn_in;
output [BTN_WIDTH-1:0] btn_out, btn_out_pulse;
reg [BTN_WIDTH-1:0] btn_in_d [1:4]; //btn delayed
wire set; //sync reset to zero
reg [SIZE-1:0] o = {SIZE{1'b0}}; //counter is initialized to 0
always @(posedge clk or posedge rst)
begin
if (rst) begin
btn_in_d[1] <= 0;
btn_in_d[2] <= 0;
btn_in_d[3] <= 0;
o <= 0;
end
else begin
btn_in_d[1] <= btn_in;
btn_in_d[2] <= btn_in_d[1];
if (set == 1) o <= 0; //reset counter when input is changing
else if (o[SIZE-1] == 0) o<=o+1; //stable input time is not yet met
else btn_in_d[3] <= btn_in_d[2]; //stable input time is met, catch the btn and retain.
end
end
assign btn_out = btn_in_d[3]; //debounced button stable out
assign set = (btn_in_d[1] != btn_in_d[2])? 1 : 0; //determine when to reset counter
always @(posedge clk or posedge rst) begin
if (rst) btn_in_d[4] <= 0;
else btn_in_d[4] <= btn_in_d[3];
end
assign btn_out_pulse = btn_in_d[3] & (~btn_in_d[4]); //debounced button pulse out
```

```
endmodule
```

dec.v 테스트벤치 코드

```
module dec7(
    input [3:0] dec_in,
    output reg [6:0] dec_out
    );

always @ (dec_in) begin
        case (dec_in)
            4'b0000: dec_out = 7'b1111110;
            4'b0001: dec_out = 7'b0110000;
            4'b0010: dec_out = 7'b1101101;
            4'b0011: dec_out = 7'b1111001;
            4'b0100: dec_out = 7'b0110011;
            4'b0101: dec_out = 7'b1011011;
            4'b0110: dec_out = 7'b1011111;
            4'b0111: dec_out = 7'b1110010;
            4'b1000: dec_out = 7'b1111111;
            4'b1001: dec_out = 7'b1111011;
            default: dec_out = 7'b0000000;
        endcase
end

endmodule
```

테스트벤치 코드

```
// 테스트 벤치 코드를 작성할 때 중요요소들을 검증하기 위하여 top 모듈과 clock모듈의 dut를 생성하여 해당 모듈들의 출력을 찍어 봄.
module tb_digitalwatch;
reg clk, reset_poweron, clk_6mhz;
wire counter_en;
reg clock_en;
reg [3:0] btn;
reg rst, up, down;
reg [5:0] digit;
wire [3:0] sec0, sec1, min0, min1, hrs0, hrs1;
wire [7:0] seg_data;
wire [5:0] seg_com;
// digit 신호는 100000와 같은 모습을 보여야 하지만 검증의 편의를 위해 101010으로 세팅하고 진행함.
// 버튼도 본래 한번 누를 때 한번만 증가해야 하지만 검증의 편의를 위해 긴 펄스를 생성하여 검증을 진행함.
initial begin
    clk = 0; clock_en = 0; reset_poweron = 1; btn = 0; rst = 1; digit = 6'b101010; up = 0; down = 0;
    #1 rst = 0; reset_poweron = 0;
    #100 btn = 4'b0010; up = 1; down = 0;
    #25 btn = 4'b0000; up = 0; down = 0;
```

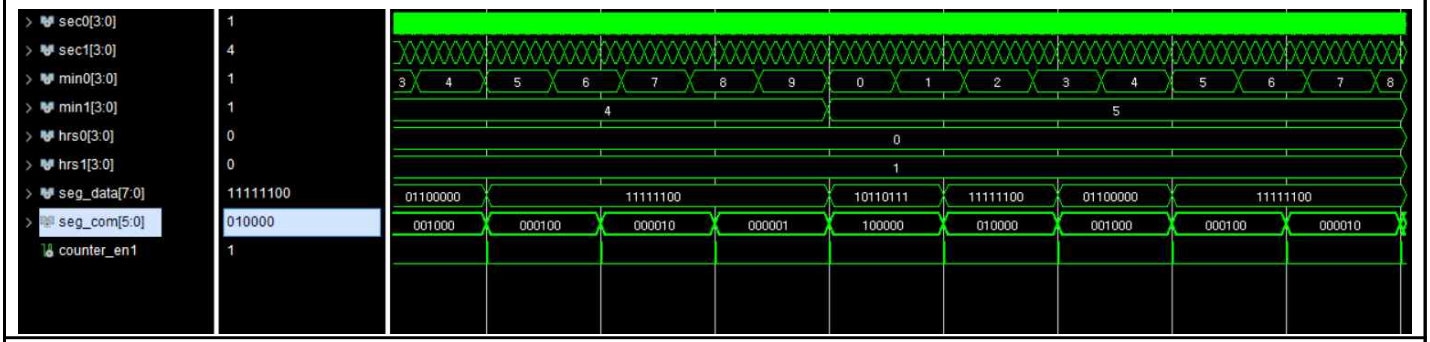
```
    #100 btn = 4'b0010; up = 1; down = 0;
    #25 btn = 4'b0000; up = 0; down = 0;
    #100 btn = 4'b0001; up = 0; down = 1;
    #25 btn = 4'b0000; up = 0; down = 0;
    #100 btn = 4'b0001; up = 0; down = 1;
    #25 btn = 4'b0000; up = 0; down = 0;
end

always begin
    #10; clk = ~clk; clk_6mhz = clk; clock_en = counter_en;

end

top u1(clk, reset_poweron, btn, seg_data, seg_com);
clock u2(clk_6mhz, rst, clock_en, digit, up, down, sec0, sec1, min0, min1, hrs0, hrs1);
gen_counter_en #(.SIZE(10)) u3(clk, rst, counter_en);

endmodule
```

2. 타이밍도 캡쳐

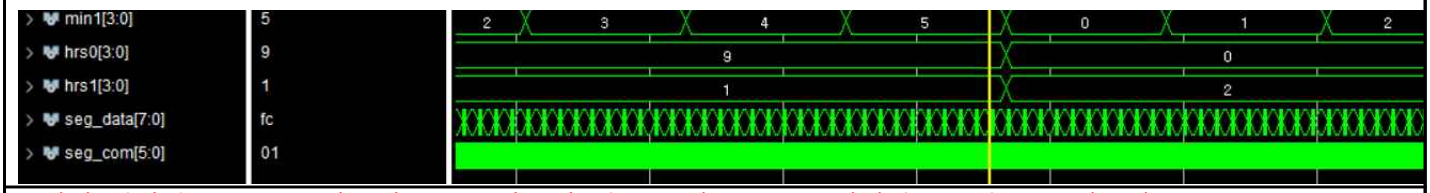
타이밍도

// 해당 타이밍도는 sec0, sec1 min0, min1의 trigger에 따라 다음 sec1, min0.. 등이 증가하는 모습을 보인다.
// seg_com은 clk의 조건에 따라 shift하는 모습을 보여준다.
// seg_data는 digit과 sec, min, hrs의 신호에 따라 변화한다. 다만 여기서는 타이밍도 검증의 편의를 위해 clk과 몇몇 조건들을 조정하였기 때문에 변화가 사진에서 관찰되지는 않지만 실제 kit에서는 문제 없이 작동한다.
seg_data와 seg_com이 잘 작동하는 모습은 다음페이지에 보여진다.

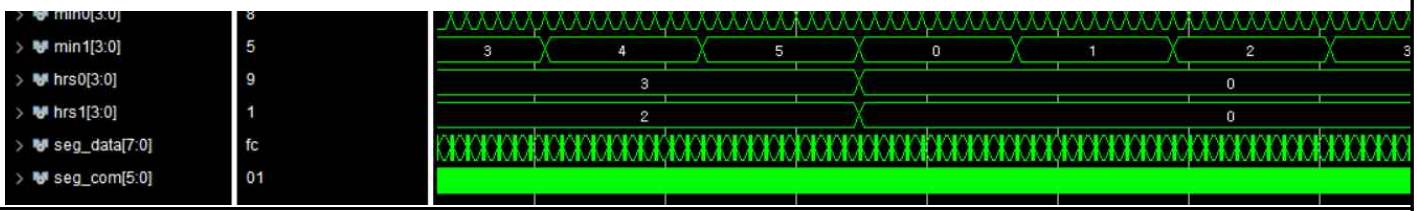
// seg_data와 seg_com이 제대로 작동하는 모습



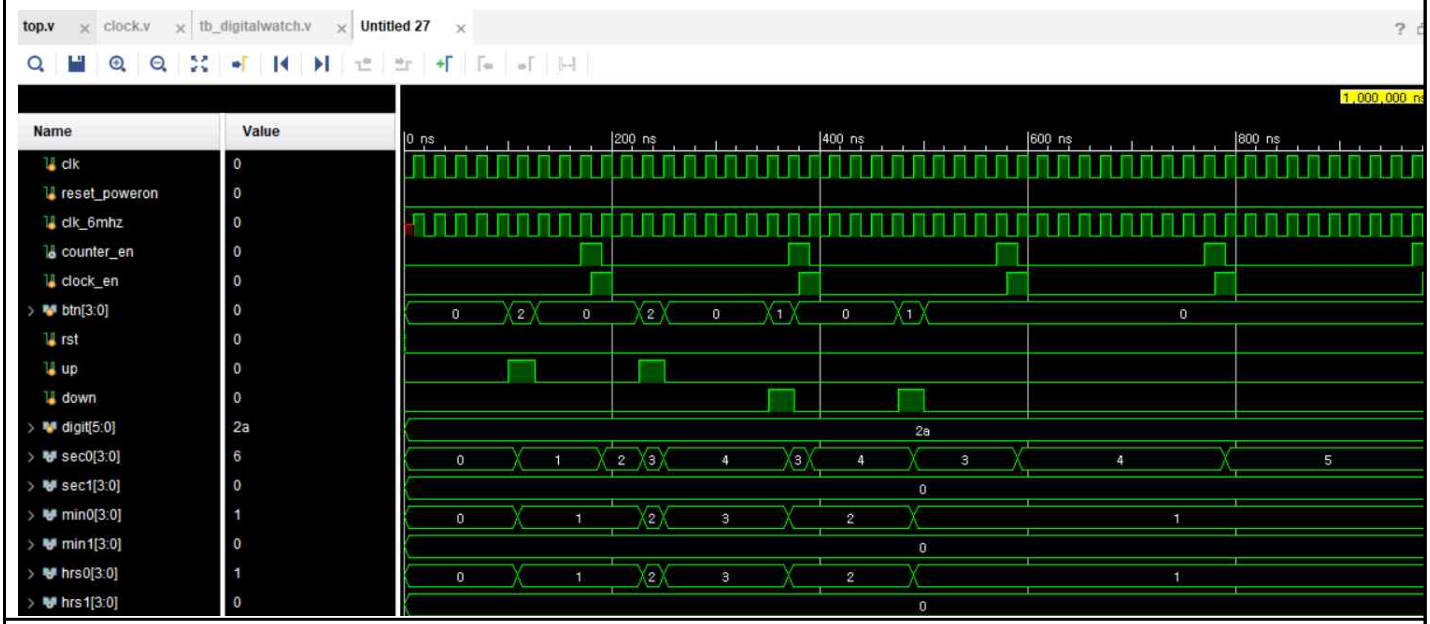
//해당 타이밍 도는 hrs1이 1이고 hrs0가 9일 때 hrs2가 2로 증가하는 모습을 보여준다.



//해당 타이밍 도는 hrs1이 2이고 hrs0이 3일 때 hrs2가 0으로 초기화되는 모습을 보여준다.



//해당 타이밍 도는 버튼(up, down)신호가 들어왔을 때 각 시간 요소들이 증가/감소하는 모습을 보여준다.
//검증의 편의를 위해 digit은 101010으로 초기 세팅해두었다. 실제 키트에서는 digit이 제대로 동작한다.



3. Discussion

설계방법

1. top모듈의 원형은 교수님이 먼저 작성한 코드를 그대로 사용하였다. 해당 top 모듈의 계층구조에서 추가해야 모듈은 다음과 같다. IP로 생성한 clk_wiz_0, clk_wiz_0에서 생성한 600mhz의 신호를 특정 주파수로 변조하는 gen_counter_en, gen_counter_en에서 생성한 클락을 이용하여 시간을 계산하는 clock, 버튼의 채터링을 보조하는 debounce, 7segment 디코더의 출력을 위한 dec. 다음은 위의 하위 계층의 모듈을 설계한 방법이다.

2. clk_wiz_0의 설계는 vivado에서 자체 제공하는 clk_wiz ip를 이용하였다. 해당 ip를 생성하여 600mhz를 세팅하면 완성된 코드가 튀어나온다. 이번 설계에서는 ip가 설계한 클럭을 한번 더 변조한다.

3. gen_counter_en의 설계는 clk_wiz_0가 설계한 600mhz의 신호를 입력받아 1hz의 신호로 변조한다. 따라서 600m -1번 count가 되었을 때 assign counter_en = (o == SIZE-1)? 1'b1 : 1'b0;를 통하여 1hz의 counter_en를 생성하였다.

4. counter 신호는 sec0, sec1, min0, min1, hrs0, hrs1신호를 표현하기 위하여 6개의 always 구문을 이용하였다. always 구문 하나에는 하나의 신호만을 제어한다. always의 sensitivity list 안에는 clk_6mhz와 rst을 조건으로 받는다.

```
always@(posedge clk_6mhz or posedge rst) begin
    if(rst) sec0 <= 4'b0;
    else if(digit[5] == 1'b1 && up == 1'b1) sec0 <= sec0 + 1;
    else if(digit[5] == 1'b1 && down == 1'b1) sec0 <= sec0 -1;
    else if(clock_en)
        if(sec0 == 9) begin sec0 <= 0; end
        else begin sec0 <= sec0 + 1; end
end
assign trigger1 = (sec0 == 9) ? 1'b1 : 1'b0;
```

위의 코드를 보면 rst신호가 들어오면 sec0을 초기화한다. up, down 신호와 digit[5]의 신호가 1로 일치할 때 sec0을 증가하거나 감소시킨다. 1초에 한번 clock_en 클럭이 들어오면 sec0을 증가시키고 9에 도달하면 초기화 한다. assign 문으로 조합회로를 통하여 sec0이 9에 도달하면 펄스 신호를 하나 생성한다.

```
always@(posedge clk_6mhz or posedge rst) begin
    if(rst) sec1 <= 4'b0;
    else if(digit[4] == 1'b1 && up == 1'b1) sec1 <= sec1 + 1;
    else if(digit[4] == 1'b1 && down == 1'b1) sec1 <= sec1 - 1;
    else if(trigger1 && clock_en)
        if(sec1 == 5) begin sec1 <= 0; end
        else begin sec1 <= sec1 + 1; end
    end
 assign trigger2 = (sec1 == 5 && sec0==9 && clock_en) ? 1'b1 : 1'b0;
```

위의 코드는 sec1을 만드는 신호이다. 기본구조는 sec0과 동일하지만 sec0은 clock_en신호가 도착할 때 count되었다. 그러나 sec1은 trigger1과 count_en가 같이 도달할 때만 count 된다. trigger1이 도착했을 때 count 되는 것은 상식적으로 당연하지만 count가 같이 도착해야하는 이유는 clk_6mhz를 sensitivity 리스트에 넣었기 때문이다. 따라서 count_en 신호가 도착해야지만 10초에 한번 카운트 된다. 해당 내용에 대한 이야기는 뒤의 문제해결 파트에서 자세하게 다룬다.

trigger2 신호는 sec1 sec0 clock_en을 모두 조건안에 넣어주어야 정확한 타이밍에 짧은 펄스를 생성할 수 있다. 이유는 clk_6mhz가 도착했을 때 여러번 카운트 되면 안되고 한번만 카운트 되어야하기 때문이다. 해당내용도 뒤의 문제해결 파트에서 자세하게 다룬다.

min0, min1은 위와 같은 알고리즘을 쓰기 때문에 생략한다.

```
always@(posedge clk_6mhz or posedge rst) begin
    if(rst) hrs0 <= 4'b0;
    else if(digit[1] == 1'b1 && up == 1'b1) hrs0 <= hrs0 + 1;
    else if(digit[1] == 1'b1 && down == 1'b1) hrs0 <= hrs0 - 1;
    else if(trigger6 && clock_en) begin hrs0 <= 0;
    end
    else if(trigger4 && clock_en)
        if(hrs0 == 9) begin hrs0 <= 0; end
        else begin hrs0 <= hrs0 + 1; end
 end
assign trigger5 = (hrs0 == 9 && min1 == 5 && min0 == 9 && sec1 == 5 && sec0==9) ? 1'b1 : 1'b0;
assign trigger6 = (hrs1 == 2 && hrs0 == 3 && min1 == 5 && min0 == 9 && sec1 == 5 && sec0==9) ? 1'b1
: 1'b0;

always@(posedge clk_6mhz or posedge rst) begin
    if(rst) hrs1 <= 4'b0;
    else if(digit[0] == 1'b1 && up == 1'b1) hrs1 <= hrs1 + 1;
    else if(digit[0] == 1'b1 && down == 1'b1) hrs1 <= hrs1 - 1;
    else if(trigger5 && clock_en)
        if(hrs1 == 2) begin hrs1 <= 0; end
        else begin hrs1 <= hrs1 + 1; end
    else if(trigger6 && clock_en) hrs1 <= 0;
 end
```

위의 신호는 hrs0과 hrs1에 대한 코드 설계이다. 중요한 점은 24시에 도달하면 rst되어야 한다는 점이 sec와 min과는 다른점이다. 해당내용을 구현하기 위하여 trigger5와 trigger6를 구현하였으며 trigger5는 sec와 min과 같은 알고리즘을 사용한다. trigger6는 24시가 되면 rst시키는 trigger이다.

5. debounce 모듈이 필요한 이유는 버튼을 누를 때 생기는 채터링 때문이다.

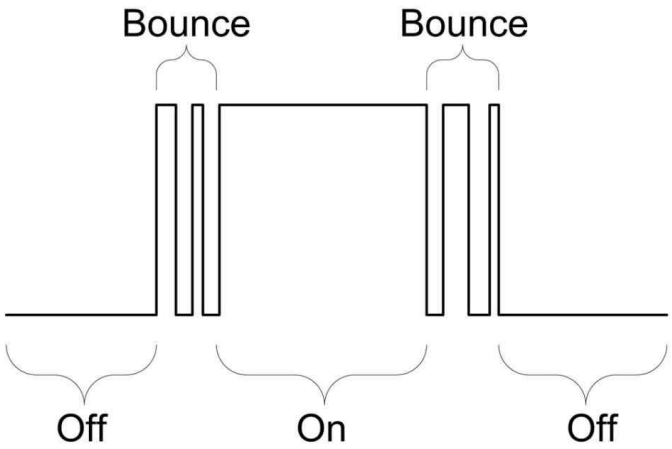


그림 8 출처 : 네이버 블로그 https://blog.naver.com/robochol84/220808821127

사진과 같이 사람은 버튼을 한번 누르지만 스프링의 기계적인 진동에 의해 여러번 펄스가 발생한다. 해당 문제를 막기 위한 모듈이며 코드는 교재의 코드를 그대로 사용하여 설계하였다.

6. dec 모듈은 7-segment디코더의 숫자 출력을 하기위해 작성한 코드이다.

```
always @ (dec_in) begin
        case (dec_in)
            4'b0000: dec_out = 7'b1111110;
            4'b0001: dec_out = 7'b0110000;
            4'b0010: dec_out = 7'b1101101;
            4'b0011: dec_out = 7'b1111001;
            4'b0100: dec_out = 7'b0110011;
            4'b0101: dec_out = 7'b1011011;
            4'b0110: dec_out = 7'b1011111;
            4'b0111: dec_out = 7'b1110010;
            4'b1000: dec_out = 7'b1111111;
            4'b1001: dec_out = 7'b1111011;
            default: dec_out = 7'b0000000;
        endcase
end
```

다음과 같이 입력 dec_in을 case의 조건으로 사용하여 7segment의 led를 키기위한 신호를 내보낸다. 해당 출력은 top모듈의 seg_data에서 digit신호와 합성되어 dot까지 표기하고 출력되게 된다.

7. 다시 top 모듈의 digit, seg_com과 seg_data의 설계이다. digit은 left, right 신호에 맞추어 변화한다. 해당 digit은 clock 모듈의 입력으로 들어가 앞서 설명한 clock모듈의 up, down에 맞추어 시분초를 증가시킨다. seg_com은 현재 출력하고싶은 6개의 led중 하나를 키게 된다. 이 과정을 초당 600번 반복하면 사람눈에는 항상 켜져있는 것처럼 보인다. 이 원리를 설계에 사용한다. 또한 seg_data는 dec_out과 digit 신호와 합성하고 신호에 맞추어 led에 시간을 출력하고 수정하고 싶은 포인트는 dot으로 출력한다.

```
    initial begin
       seg_com = 6'b100000;
       digit = 6'b100000;
    end
```

seg_com과 digit 신호를 initial 구문을 통해 1번 설정해주는 과정을 거치도록 하였다.

```
    always@(posedge clk_6mhz) begin
            if(left) begin
                digit[0] <= digit[1];
                digit[1] <= digit[2];
                digit[2] <= digit[3];
                digit[3] <= digit[4];
                digit[4] <= digit[5];
                digit[5] <= digit[0];
            end
            else if(right) begin
                digit[1] <= digit[0];
                digit[2] <= digit[1];
                digit[3] <= digit[2];
```

```
                digit[4] <= digit[3];
                digit[5] <= digit[4];
                digit[0] <= digit[5];
            end
        end
    //digit[5:0] = 100000,010000,001000,000100,000010,000001,100000,010000

//seg_com신호를 600hz마다 한칸씩 shift
    always@(posedge clock_en1) begin
            seg_com[0] <= seg_com[1];
            seg_com[1] <= seg_com[2];
            seg_com[2] <= seg_com[3];
            seg_com[3] <= seg_com[4];
            seg_com[4] <= seg_com[5];
            seg_com[5] <= seg_com[0];
    end
    //seg_com[5:0] = 100000,010000,001000,000100,000010,000001,100000,010000
//case 문을 사용하여 seg_com이 shift 될 때마다 seg_data를 출력 이때 digit[]은 커서의 dot의 on/off 의미
    always @ (*) begin
        case (seg_com)
            6'b100000: seg_data = {sec0_out, digit[5]};
            6'b010000: seg_data = {sec1_out, digit[4]};
            6'b001000: seg_data = {min0_out, digit[3]};
            6'b000100: seg_data = {min1_out, digit[2]};
            6'b000010: seg_data = {hrs0_out, digit[1]};
            6'b000001: seg_data = {hrs1_out, digit[0]};
            default: seg_data = 8'b0;
        endcase
```

digit 신호는 clk_6mhz를 이용하고 if문의 조건으로 left, right 신호가 들어오면 왼쪽, 오른쪽으로 shift 시켜준다.
seg_com 신호는 600hz로 세팅된 clock_en1을 이용하고 항상 shift상태를 반복한다.
seg_data는 case의 조건으로 seg_com을 받아 seg_data신호를 선택한다.

오류와 해결방법

이번 과제를 설계하며 100가지가 넘는 오류가 발생하였지만 모든 오류를 보고서에서 다루기에는 내용이 길어지므로 가장 치명적이고, 해결이 복잡했었던 대표적인 오류들만 다루고자 한다.

1. ambiguous clock in event control 오류이다. 인터넷 포럼에서 검색해보았지만, 명쾌한 해결방법은 찾을 수 없었다. 정황상 always@()구문의 리스트로 clk을 받지만 해당 always 구문안에서 리스트의 영향을 받지 않는 reg들이 존재할 때 발생하는 오류인 것으로 추측된다. 이 오류를 해결하기 위해서 코드를 재작성하였고 재작성시에는 if등의 조건안에는 펄스 신호만 조건으로 사용하도록 하였다.

2. redeclaration of ansi port 오류이다. 직역하자면 포트 상태의 재선언이 허용되지 않음 정도가 된다. 이역시 검색을 해보아도 명쾌한 해결방법은 찾을 수 없었다. 정황상 문법적인 오류가 발생했을 때 뜨는 것 같다. 신호가 여러번 재선언이 되었던 것 같다. 다른 오류를 고치기 위해 틀린 코드를 모두 지우고 재작성하는 과정에서 사라졌다.

3. no constraint will be written out 오류이다. xdc 파일의 pin 매핑과 차이가 났을 때 발생하는 것으로 추측된다. xdc 파일을 재작성하고 코드 재작성후 사라짐을 확인했다.

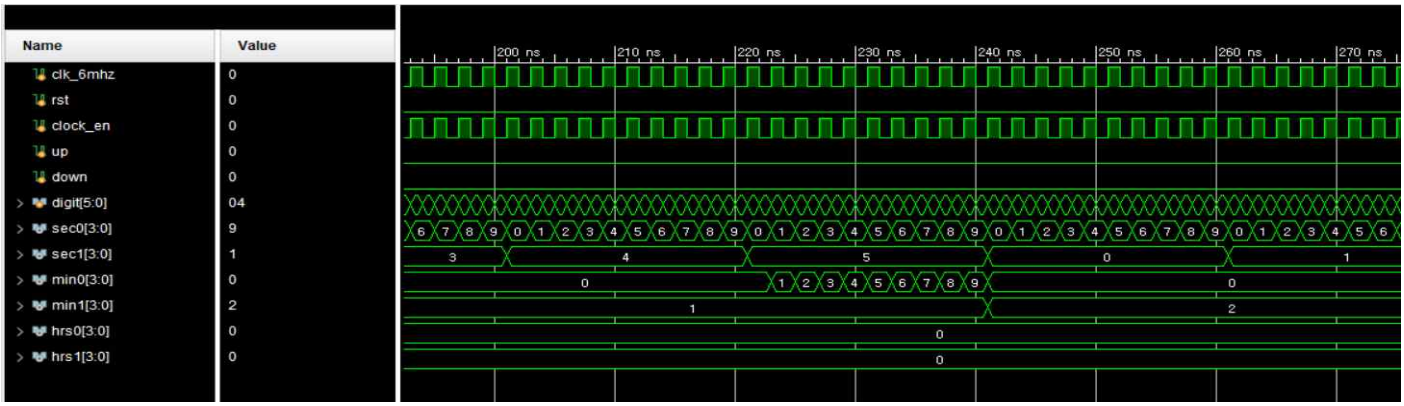
4. verilog design clock uncunnect... 오류이다. 해당 오류는 clk_wiz ip를 합성할 때 발생한 오류로 추측되는데 경고메세지를 보고 clk_wiz ip를 모두 삭제 후 다시 불러와 합성하였더지 사라짐을 확인했다.

5. multi driven net Q 오류는 합성시 회로의 reg에 여러 신호가 들어와 발생하는 오류이다. 주로 여러 always 문에서 하나의 신호를 바꾸려 할 때 발생한다. 다음 오류가 발생한 경로는 이렇다.

```
always@(posedge clk_6mhz) begin
    if(up) begin
        case(digit)
            6'b100000 : sec0 = sec0 + 1;
            6'b010000 : sec1 = sec1 + 1;
            6'b001000 : min0 = min0 + 1;
            6'b000100 : min1 = min1 + 1;
            6'b000010 : hrs0 = hrs0 + 1;
            6'b000001 : hrs1 = hrs1 + 1;
        endcase
     end
     else if(down) begin
         case(digit)
            6'b100000 : sec0 = sec0 - 1;
            6'b010000 : sec1 = sec1 - 1;
            6'b001000 : min0 = min0 - 1;
            6'b000100 : min1 = min1 - 1;
            6'b000010 : hrs0 = hrs0 - 1;
            6'b000001 : hrs1 = hrs1 - 1;
     endcase
     end
end

always@(posedge clock_en or posedge rst) begin
    if(rst) sec0 <= 4'b0;
    else
        if(sec0 == 9) begin sec0 <= 0; trigger1 = (sec0 == 9) ? 1'b1 : 1'b0; end
        else if(up) sec0 <= (digit[5] == 1'b1) ? sec0 + 1 : sec0;
        else if(down) sec0 <= (digit[5] == 1'b1) ? sec0 - 1 : sec0;
        else begin sec0 <= sec0 + 1; trigger1 = (sec0 == 9) ? 1'b1 : 1'b0; end
    end
```

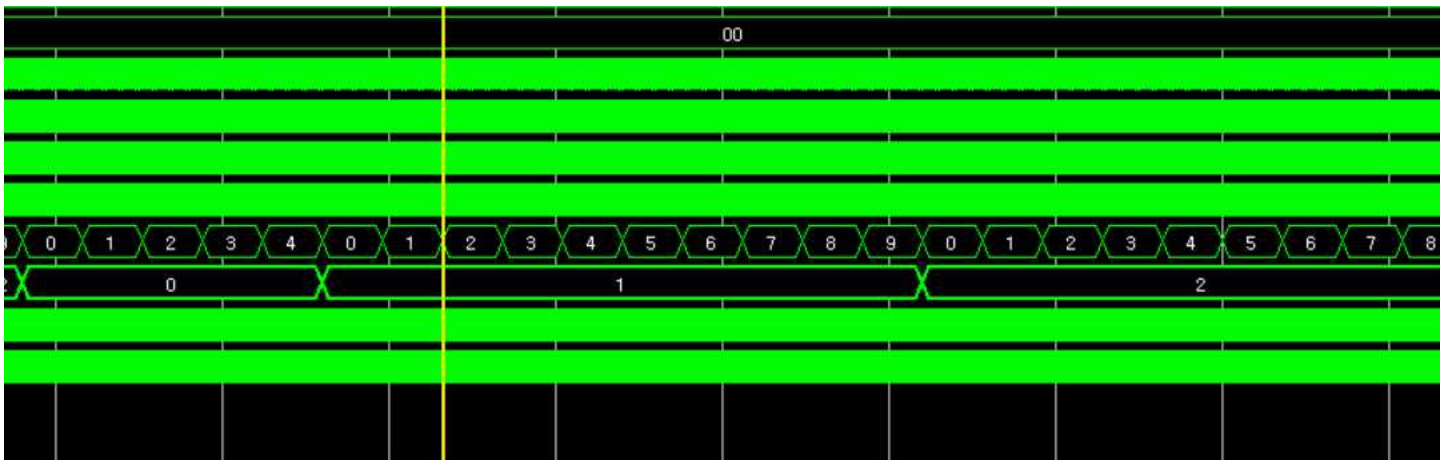
코드 작성시 always 문 하나안에서 up down 신호를 모두 제어하고 싶었으나 오류의 설명과 같이 reg하나에 여러 always 문에서 신호가 중복하여 들어오기에 합성시 문제가 된다. 문제 해결을 위하여 코드를 모두 지우고 다시 설계하는 과정을 거쳤다.

6.



다음은 trigger 가 발생했을 때 값이 폭팔적으로 증가하는 오류이다. 해당 오류가 발생한 이유 다음과 같다. 트리거가 발생했을 때 assign trigger3 = (min0 == 9) ? 1'b1 : 1'b0; 다음과 같이 조건을 구성했다면 10초 혹은 1분 혹은 1시간 동안 트리거가 1인 상태가 유지가 된다. 이때 리스트안에서 높은 주파수의 clk을 입력으로 받기에 trigger가 발생했을 때 1번만 증가하는 것이 아닌 높은 주파수에 맞추어 폭팔적으로 값이 증가한다. 해당 오류를 해결하기 위해서는 다음과 같이 트리거 조건을 구성해야 한다. assign trigger5 = (hrs0 == 9 && min1 == 5 && min0 == 9 && sec1 == 5 && sec0==9) ? 1'b1 : 1'b0; 또한 always 문안의 if 문 조건에도 (trigger && count_en) 이렇게 조건을 구성해야 문제가 생기지 않는다. 해당 조건에 맞추어 구현한 코드는 위에서 제출한 소스코드에 모두 나와있다.

7.



다음 오류는 hrs1이 1이 되었음에도 hrs0가 계속 증가하는 경우이다. 해당 오류는 hrs0의 조건에 hrs1에 대한 가정이 없었기 때문에 생긴 문제로 위 문제를 해결하기 위하여 trigger의 조건을 추가하였다.

```
always@(posedge clk_6mhz or posedge rst) begin
    if(rst) hrs0 <= 4'b0;
    else if(digit[1] == 1'b1 && up == 1'b1) hrs0 <= hrs0 + 1;
    else if(digit[1] == 1'b1 && down == 1'b1) hrs0 <= hrs0 - 1;
    else if(trigger6 && clock_en) begin hrs0 <= 0;
    end
    else if(trigger4 && clock_en)
        if(hrs0 == 9) begin hrs0 <= 0; end
        else begin hrs0 <= hrs0 + 1; end
 end
assign trigger5 = (hrs0 == 9 && min1 == 5 && min0 == 9 && sec1 == 5 && sec0==9) ? 1'b1 : 1'b0;
assign trigger6 = (hrs1 == 2 && hrs0 == 3 && min1 == 5 && min0 == 9 && sec1 == 5 && sec0==9) ? 1'b1 : 1'b0;
```

8.

```
always@(posedge clk_6mhz or posedge rst) begin
    if(rst) min0 <= 4'b0;
    else if(digit[3] == 1'b1 && up == 1'b1) min0 = min0 + 1;
    else if(digit[3] == 1'b1 && down == 1'b1) min0 = min0 - 1;
    else if(trigger2 && clock_en)
        if(min0 == 9) begin min0 <= 0;end
        else begin min0 <= min0 + 1; end
    end
assign trigger3 = (min0 == 9 && sec1 == 5 && sec0==9 && clock_en) ? 1'b1 : 1'b0;

always@(posedge clk_6mhz or posedge rst) begin
```

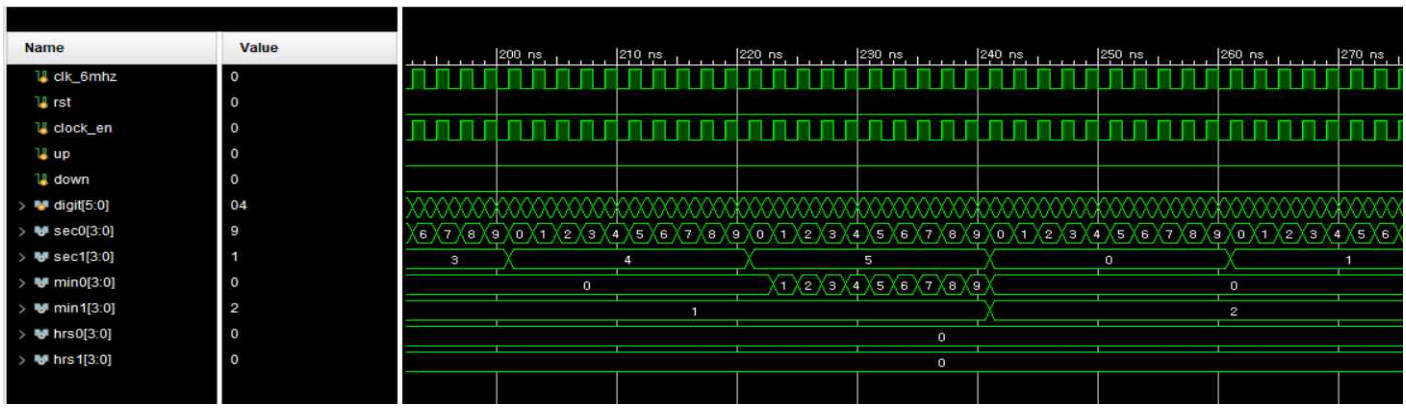
non blocking과 blocking을 구분하지 않고 적었을 때는 문제가 생기니
else if(digit[3] ==1'b1 && up == 1'b1) min0 <= min0+1; 과 같이 적어 주어야 한다.

8. 다음은 키트 구동시에 발생했던 문제들이다. led에 sec0, min0, hrs0만 띄어지는 경우는 seg_com의 주파수 문제를 고려해보아야 한다. 너무 빠를시에는 숫자가 희미하게 보이거나 안보이는 경우가 종종 생긴다.

9. led에 숫자가 아닌 다른 문자가 표기되는 경우에는 코드를 바꾸어 보는 것 보다, led에 하자가 있어 제대로 표기되지 않는 문제일 수 있다. led를 바꾸니 문제 없기 표기되었다.

10. digit의 left right 버튼은 무리없이 작동하지만 up, down이 작동하지 않는다면 kit의 버튼문제가 아니라 clock 모듈안의 always 구문의 알고리즘 오류일 가능성이 매우크다. sensitivity리스트를 clk_6mhz로 설정하고 버튼 작동에 대한 구조식을 else if (up)과 같이 구성하여 해결해야 한다.

11. 숫자가 너무 빠르게 올라간다면 clock의 알고리즘에 문제가 있다. 보통 sec0이 1초에 한번 증가하지 않는 경우이거나 위에서 설명한 것과 같이



트리거가 발생 할 때 값이 폭팔적으로 증가하는 경우이다. kit에서 작동하기 전에 testbench를 통하여 모든 출력값들을 충분히 검증하여 보아야 한다.