mini project 레포트

1. Block diagram

이번 과제에서는 다음과 같이 모듈을 구성하였다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

counter 모듈은 25Mhz의 clk 신호를 1khz로 나눠 clk\_1khz 클럭을 출력한다.

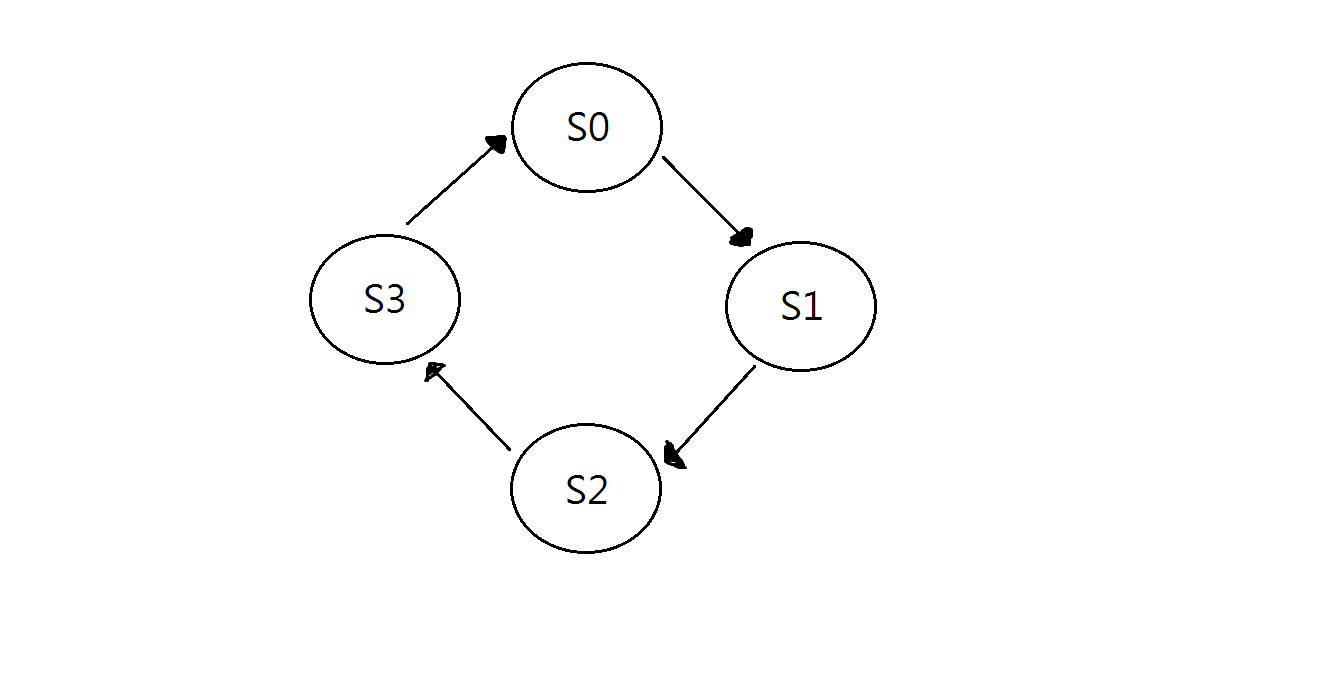
debounce 모듈은 입력의 버튼을 눌렀을 때 발생하는 chattering을 제어하기 위한 모듈이다. inc와 next 버튼 입력 신호, clk, nRst 신호를 입력으로 받아 next 모듈에 next\_reg, increase 모듈에 inc\_reg를 출력한다.

next 모듈과 increase 모듈은 nRst, clk, 그리고 각각 next와 inc 버튼 입력을 debounce 모듈에 넣어서 출력을 받아 각각 inc\_falling과 next\_falling 신호를 출력한다.

마지막으로 timer 모듈은 next와 increase 모듈에서 출력된 inc\_falling과 next\_falling 신호와 counter 모듈에서 출력된 clk\_1khz, 그리고 nRst, clk 입력을 받아서 최종적으로 seg\_sel [3:0], seg\_dat [7:0]을 출력한다.

1. State diagram

이번 과제에서 구현한 timer는 다음과 같은 state diagram에 따라 동작한다.



S0 상태는 초기 상태이며 seg\_sel[3:0]=0111을 출력한다. S1 상태는 seg\_sel[3:0]=1011을 출력하고, S2 상태는 seg\_sel[3:0]=1101을 출력하고, S3 상태는 seg\_sel[3:0]=1110을 출력한다.

상태는 1khz 클럭이 positive edge 가 될 때마다 다음 상태로 넘어간다.

이를 통해 dynamic display를 구현했다.

게임, 남자이(가) 표시된 사진

자동 생성된 설명

그리고 next 버튼의 입력이 들어오는 것을 카운트를 해주는 변수를 만들어서, 그 횟수에 따라 상태가 달라지게 구현했다.

우선 초기 상태는 next\_cnt가 0일 때이다. 이 상태에서 inc 버튼을 누르면 digit1의 숫자가 1 상승하도록 출력한다.. 다음 상태는 next\_cnt가 1일 때로, 이 상태에서 inc 버튼을 누르면 digit2의 숫자가 1 상승하도록 출력한다. 다음 상태는 next\_cnt가 2일 때로, 이 상태에서 inc 버튼을 누르면 digit3의 숫자가 1 상승하도록 출력한다. 다음 상태는 next\_cnt가 3일 때로, 이 상태에서 inc 버튼을 누르면 digit4의 숫자가 1 상승하도록 출력한다. 위 상황에서 digit1~4의 숫자가 9를 넘어가게 되면 0으로 돌아가도록 설계했다. next\_cnt가 0,1,2,3인 경우는 next 버튼을 누르는 것이 다음 상태로 넘어가는 조건이지만, 마지막 상태인 next\_cnt가 4일 때는, inc 버튼과 next 버튼의 입력을 무시하도록 설계했다. 그리고 digit1~4가 각각 천의 자리, 백의 자리, 십의 자리, 일의 자리에 들어가는 숫자로 해서, 그 수가 0이 될 때까지 0.1초마다 1씩 감소하도록 구현했다. 그리고 0이 되면 다음 상태, next\_cnt가 0인 상태로 넘어가서 이 과정을 반복하도록 설계했다. 그리고 nRst 버튼을 누르면 초기상태로 돌아가고, digit1~4에 설정된 숫자도 모두 0이 되도록 했다.

1. 구현

모듈은 다음과 같이 구현되었다.

module timer(clk, nRst, inc, next, seg\_dat, seg\_sel);

input clk, nRst, inc, next;

output reg [7:0] seg\_dat;

output reg [3:0] seg\_sel;

parameter [1:0] S0 = 2'd0, S1 = 2'd1, S2 = 2'd2, S3 = 2'd3;

reg [1:0] state=0, nxt\_state;

reg [3:0] seg;

reg [22:0] cnt1 = 0;

reg [22:0] cnt2 = 0;

reg [3:0] offvalue = 10;

reg [3:0] value1=0, value2=0, value3=0, value4=0, value5=0, value6=0, value7=0, value8=0;

reg [3:0] onvalue;

wire inc\_falling, next\_falling;

reg inc\_a, next\_a;

reg [3:0] dig1, dig2, dig3, dig4;

wire clk\_1khz;

reg [2:0] next\_cnt=0, next\_cntv=0;

counter c0(.clk(clk), .clk\_1khz(clk\_1khz));

increase in0(.clk(clk), .nRst(nRst), .inc(inc), .inc\_falling(inc\_falling));

next ne0(.clk(clk), .nRst(nRst), .next(next), .next\_falling(next\_falling));

always @(\*) begin

if (next\_cnt == 4) begin

inc\_a=0;

next\_a=0;

end else begin

inc\_a=inc\_falling;

next\_a=next\_falling;

end

end

always @(\*) begin

case (seg)

0 : seg\_dat = 8'b00111111;

1 : seg\_dat = 8'b00000110;

2 : seg\_dat = 8'b01011011;

3 : seg\_dat = 8'b01001111;

4 : seg\_dat = 8'b01100110;

5 : seg\_dat = 8'b01101101;

6 : seg\_dat = 8'b01111101;

7 : seg\_dat = 8'b00000111;

8 : seg\_dat = 8'b01111111;

9 : seg\_dat = 8'b01100111;

default : seg\_dat = 8'b00000000;

endcase

end

always @(posedge clk\_1khz)

if (~nRst)

state <= S0;

else

state <= nxt\_state;

always @(posedge clk\_1khz) begin

if (~nRst) begin

seg = 0;

end else if (next\_cnt == 4) begin

seg = onvalue;

end else begin

if (cnt1<50) begin

cnt1 = cnt1+1;

seg = onvalue;

end else if (cnt1<100) begin

cnt1 = cnt1+1;

seg = offvalue;

end else begin

cnt1= 0;

end

end

end

always @(posedge clk\_1khz) begin

case (state)

S0 : begin

seg\_sel <= 4'b0111;

end

S1 : begin

seg\_sel <= 4'b1011;

end

S2 : begin

seg\_sel <= 4'b1101;

end

S3 : begin

seg\_sel <= 4'b1110;

end

endcase

end

always @(\*) begin

case (state)

S0 : onvalue <= dig1;

S1 : onvalue <= dig2;

S2 : onvalue <= dig3;

S3 : onvalue <= dig4;

endcase

end

always @(\*) begin

nxt\_state = 2'bx;

case (state)

S0 : begin

nxt\_state = S1;

end

S1 : begin

nxt\_state = S2;

end

S2 : begin

nxt\_state = S3;

end

S3 : begin

nxt\_state = S0;

end

endcase

end

always @(posedge clk) begin

if (~nRst) begin

next\_cnt <= #1 0;

end else if (next\_cnt == 4) begin

if (value5==0 && value6==0 && value7==0 && value8==0) begin

next\_cnt <= #1 0;

end

end else

next\_cnt <= #1 next\_cntv;

end

always @(\*) begin

if (next\_a) begin

next\_cntv <= next\_cnt+1;

end else

next\_cntv <= next\_cnt;

end

always @(\*) begin

if (~nRst) begin

dig1 <= #1 0;

dig2 <= #1 0;

dig3 <= #1 0;

dig4 <= #1 0;

end else begin

if (next\_cnt == 4) begin

dig1 <= #1 value5;

dig2 <= #1 value6;

dig3 <= #1 value7;

dig4 <= #1 value8;

end else begin

dig1 <= #1 value1;

dig2 <= #1 value2;

dig3 <= #1 value3;

dig4 <= #1 value4;

end

end

end

always @(posedge clk) begin

if (inc\_a) begin

if (next\_cnt == 0) begin

value1 = dig1>=9 ? 0 : dig1+1;

end else if (next\_cnt == 1) begin

value2 = dig2>=9 ? 0 : dig2+1;

end else if (next\_cnt == 2) begin

value3 = dig3>=9 ? 0 : dig3+1;

end else if (next\_cnt == 3) begin

value4 = dig4>=9 ? 0 : dig4+1;

end

end else begin

value1=dig1;

value2=dig2;

value3=dig3;

value4=dig4;

end

end

always @(posedge clk\_1khz) begin

if (~nRst) begin

value5=0;

value6=0;

value7=0;

value8=0;

end else if (next\_cnt == 4) begin

if (cnt2<=100) begin

cnt2=cnt2+1;

end else begin

cnt2=0;

value8=dig4-1;

if (value8>9) begin

value8=9;

value7=dig3-1;

end

if (value7>9) begin

value7=9;

value6=dig2-1;

end

if (value6>9) begin

value6=9;

value5=dig1-1;

end

end

end else begin

value5 = value1;

value6 = value2;

value7 = value3;

value8 = value4;

end

end

endmodule

////////////////////////////////////////////////////////////////

module counter(input clk, output clk\_1khz);

reg [14:0] cnt=0;

reg clk\_1khz=0;

always @(posedge clk) begin

if (cnt<12500) begin

cnt <= cnt + 1;

end else begin

cnt <= 0;

clk\_1khz <= ~clk\_1khz;

end

end

endmodule

//////////////////////////////////////////////////////////////////////////////////////////

module debounce(input clk, nRst, button\_in, output reg DB\_out);

parameter N = 11;

reg [N-1:0] q\_reg;

reg [N-1:0] q\_next;

reg DFF1, DFF2;

wire q\_add;

wire q\_reset;

assign q\_reset = (DFF1^DFF2);

assign q\_add = ~(q\_reg[N-1]);

always @ (\*) begin

case( {q\_reset , q\_add})

2'b00: q\_next = q\_reg;

2'b01: q\_next = q\_reg + 1;

default : q\_next = { N {1'b0} };

endcase

end

always @(posedge clk) begin

if(nRst == 1'b0) begin

DFF1 <= 1'b0;

DFF2 <= 1'b0;

q\_reg <= { N {1'b0} };

end else begin

DFF1 <= button\_in;

DFF2 <= DFF1;

q\_reg <= q\_next;

end

end

always @(posedge clk) begin

if(q\_reg[N-1] == 1'b1)

DB\_out <= DFF2;

else

DB\_out <= DB\_out;

end

endmodule

//////////////////////////////////////

module increase(input clk, input nRst, input inc, output inc\_falling);

wire inc\_reg, inc\_falling;

reg inc\_reg2;

debounce db0(.clk(clk), .nRst(nRst), .button\_in(inc), .DB\_out(inc\_reg) );

assign inc\_falling = ~inc\_reg && inc\_reg2;

always @(posedge clk) begin

if (~nRst) begin;

inc\_reg2 <= #1 1'b0;

end else begin

inc\_reg2 <= #1 inc\_reg;

end

end

endmodule

//////////////////////////////////////////////////

module next(input clk, input nRst, input next, output next\_falling);

wire next\_reg, next\_falling;

reg next\_reg2;

debounce db1(.clk(clk), .nRst(nRst), .button\_in(next), .DB\_out(next\_reg) );

assign next\_falling = ~next\_reg && next\_reg2;

always @(posedge clk) begin

if (~nRst) begin;

next\_reg2 <= #1 1'b0;

end else begin

next\_reg2 <= #1 next\_reg;

end

end

endmodule

1. 구현

timer 모듈은 다음과 같은 동작 시나리오에 의해 검증하였다.

module tb\_timer();

reg tb\_clk, tb\_inc, tb\_next, tb\_nRst;

wire [7:0] tb\_seg\_dat;

wire [3:0] tb\_seg\_sel;

timer t0 (.clk(tb\_clk), .nRst(tb\_nRst), .inc(tb\_inc), .next(tb\_next),

.seg\_dat(tb\_seg\_dat), .seg\_sel(tb\_seg\_sel));

always

#20 tb\_clk = ~tb\_clk;

initial begin

tb\_clk = 0;

tb\_nRst = 0;

tb\_inc = 0;

tb\_next = 0;

#200 tb\_nRst = 1;

#2000000 tb\_inc = 1;

#2000000 tb\_inc = 0;

#2000000 tb\_inc = 1;

#2000000 tb\_inc = 0;

#2000000 tb\_inc = 1;

#2000000 tb\_inc = 0;

#2000000 tb\_inc = 1;

#2000000 tb\_inc = 0;

#2000000 tb\_inc = 1;

#2000000 tb\_inc = 0;

#2000000 tb\_inc = 1;

#2000000 tb\_inc = 0;

#2000000 tb\_inc = 1;

#2000000 tb\_inc = 0;

#2000000 tb\_inc = 1;

#2000000 tb\_inc = 0;

#2000000 tb\_inc = 1;

#2000000 tb\_inc = 0;

#2000000 tb\_inc = 1;

#2000000 tb\_inc = 0;

#2000000 tb\_inc = 1;

#2000000 tb\_inc = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_next = 1;

#1000000 tb\_next = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_next = 1;

#1000000 tb\_next = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_next = 1;

#1000000 tb\_next = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_next = 1;

#1000000 tb\_next = 0;

#5000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_next = 1;

#1000000 tb\_next = 0;

#1000000000 tb\_nRst = 0;

#100000 tb\_nRst = 1;

#1000000 tb\_next = 1;

#1000000 tb\_next = 0;

#1000000 tb\_next = 1;

#1000000 tb\_next = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_next = 1;

#1000000 tb\_next = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_inc = 1;

#1000000 tb\_inc = 0;

#1000000 tb\_next = 1;

#1000000 tb\_next = 0;

end

endmodule

처음에는 inc에 2ms가 지날 때마다 0과 1을 번갈아 가며 줘서, 숫자가 9가 넘어가면 0으로 넘어가는지를 검증했다. 그리고 다음으로 next에 1ms 신호를 주고, 또 inc에 1ms 신호를 여러 번 주는 걸 반복해서 digit2, digit3, digit4에 해당하는 숫자도 정상적으로 증가하는지 검증했다.

그 다음에 네번째로 next 신호를 줬을 때, 숫자가 감소하는 동안 next와 inc 신호를 한번씩 줌으로써, 숫자가 감소하는 동안에는 inc와 next신호를 무시하는 것을 검증했다.

그리고 1s 정도의 시간동안 숫자가 감소하는 것을 확인하고, nRst 신호를 0으로 해서, 초기화 되는 것을 확인했다.

그리고 숫자가 모두 감소했을 때의 상태를 확인하기 위해 digit3, digit4에 적은 수를 주고, 이를 검증했다.