HW#10 디지털 회로 설계 및 언어 월수 9:00~10:15 2015104027 박정진

Design LD Driver

Design LD Driver with

1 12-bits binary counter I_Out (Predefined value of I_out is 2000)
1 11-bits binary counter Counter_1000
2 External input(main input) SW_ON, LD_ON
1 Flip-Flop LD_ON_reg
1 Master Clock 100MHZ (period = 10ns)

Operation

/*
If design the circuit which satisfy the condition that take 10 seconds to fully increase, it takes too long time to simulate it, so I scale it down to 10ms. If you want to design the circuit acting like original assignment condition, just count 10e^6. It means change the condition that on state S1, if Counter == 999999 then increase I_out, and on state S2, if Counter == 499999 then decrease I_out.
However it is not efficient that storing 10e^6 decimal because it requires a lot of bits. Another way to count 10e^6 more efficient is design asynchronous counter. But in condition all register operates by master clock edge, it is not proper with given condition.
*/

If SW_ON=1, initiate the system operation by clearing I_out and Counter_1000 then
If LD_ON=1, start increasing I_out by counting Counter_1000.

```
    If Counter_1000 == 11'd999, I_Out ← I_out+ 2
    Else Counter_1000 ← Counter_1000+ 1'b1
```

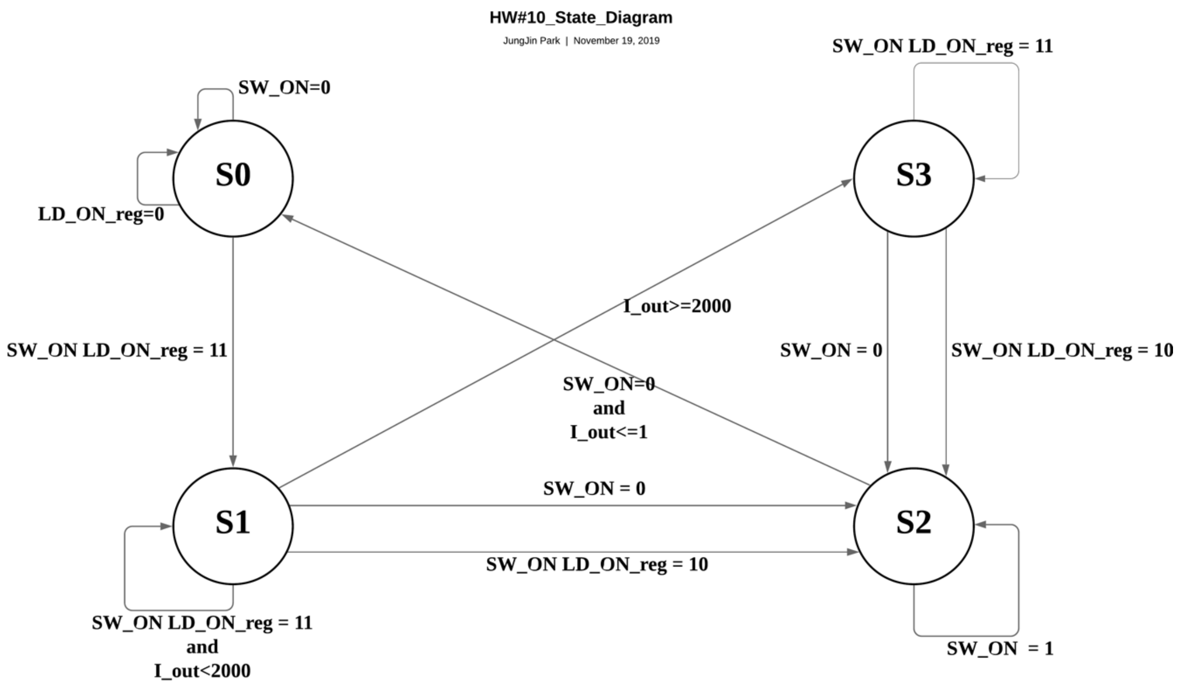
I_out increase linearly and time to fully increase is 10ms. when I_out reaches 2000, stop increasing , keep its value before SW_ON or LD_ON change to 0.
If LD_ON = 0, start decreasing I_out value to twice about the rate of increase, and stop when I_out becomes less equal than 1. (maintain the system switch on which means that remain the state on S2)
During increasing or decreasing, if LD_ON switches to opposite value, increase or decrease value from that time and the rate of increase or decrease is same as before that we designated.
If SW_ON = 0 after on any state except initial, change value of LD_ON to 0 (For this reason, we need to declare reg type LD_ON_reg to store LD_ON) and decrease I_out

value to twice the rate of increase, finally stop the system. On this state, operation is independent of LD_ON.
If SW_ON=1, LD_ON=0 on initial state, the operation cycle repeats by clearing I_out and Counter_1000.
If SW_ON = 0 on initial state, the system remains in the initial state

States

initial state : S_0
increase I_out : S_1
decrease I_out : S_2
keep I_out : S_3

State diagram

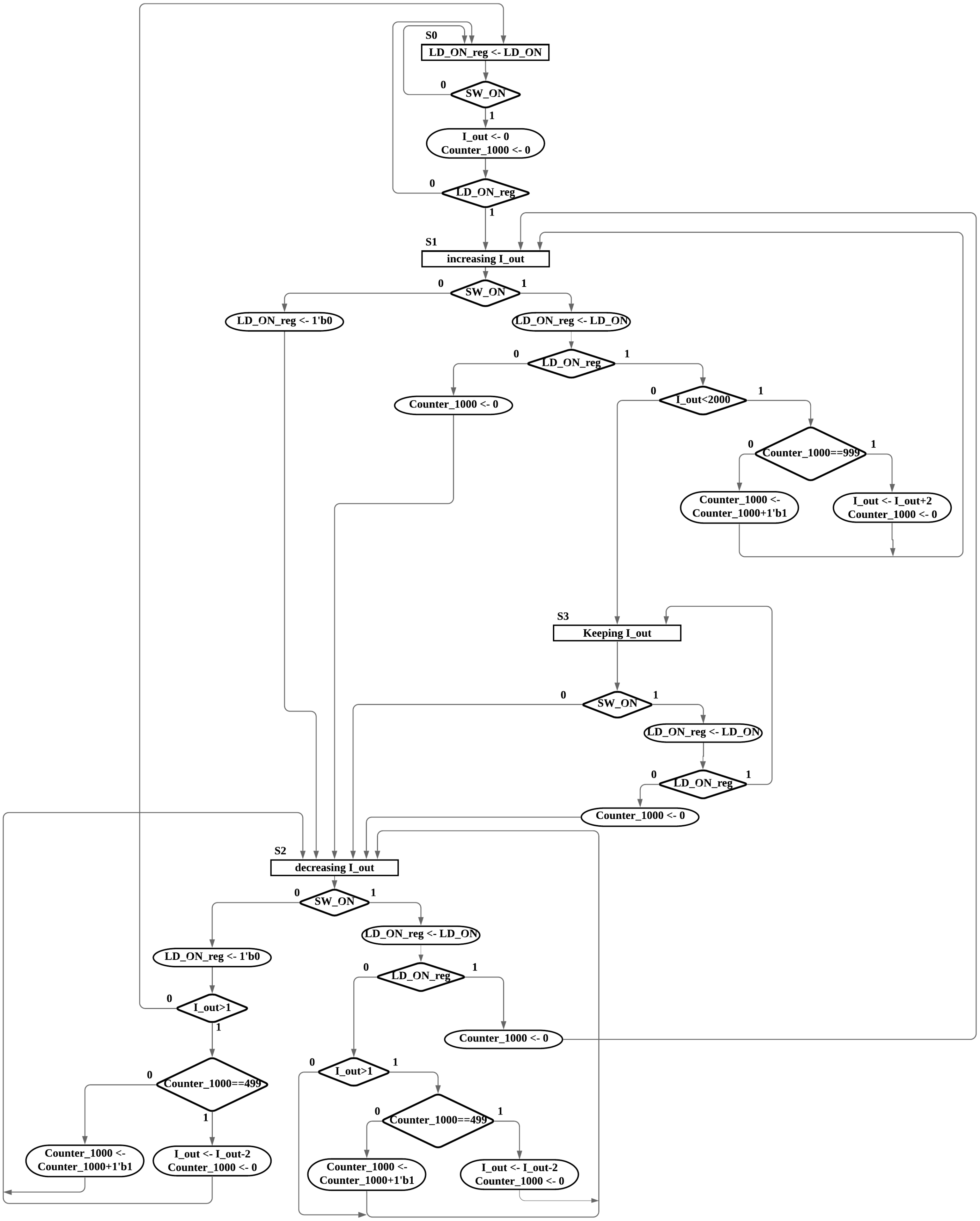


Register Transfer Operation

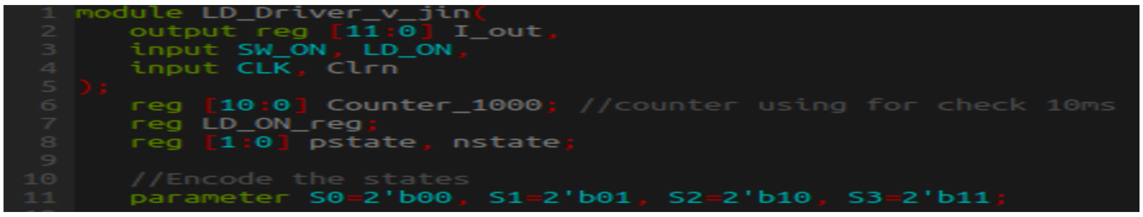
```
S0 : LD_ON_reg ← LD_ON
     if (SW_ON) then I_out ← 0, Counter_1000 ← 0

S1 : if (SW_ON)
         then LD_ON_reg ← LD_ON
         if (LD_ON_reg)
            if(I_out < 2000)
               if(Counter_1000 == 999) then I_Out ← I_out+ 2, Counter_1000 ← 0
               else then Counter_1000 ← Counter_1000+ 1'b1
         else then Counter_1000 ← 0
     else then LD_ON_reg ← 1'b0

S2 : if (SW_ON)
         then LD_ON_reg ← LD_ON
         if (LD_ON_reg) then Counter_1000 ← 0
         else
            if(I_out > 1)
               if(Counter_1000 == 499) then I_Out ← I_out-2, Counter_1000 ← 0
               else then Counter_1000 ← Counter_1000+ 1'b1
      else then LD_ON_reg ← 0
         if(I_out > 1)
            if(Counter_1000 == 499) then I_Out ← I_out-2, Counter_1000 ← 0
            else then Counter_1000 ← Counter_1000+ 1'b1

S3 : if (SW_ON)
         then LD_ON_reg ← LD_ON
         if (LD_ON_reg = 0) then Counter_1000 ← 0
```

HW#10_ASM_Chart

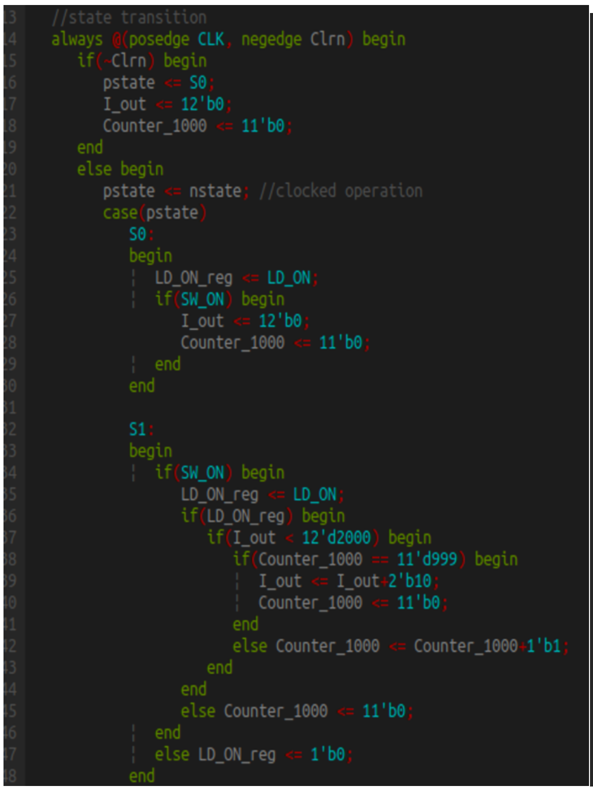
JungJin Park | November 19, 2019



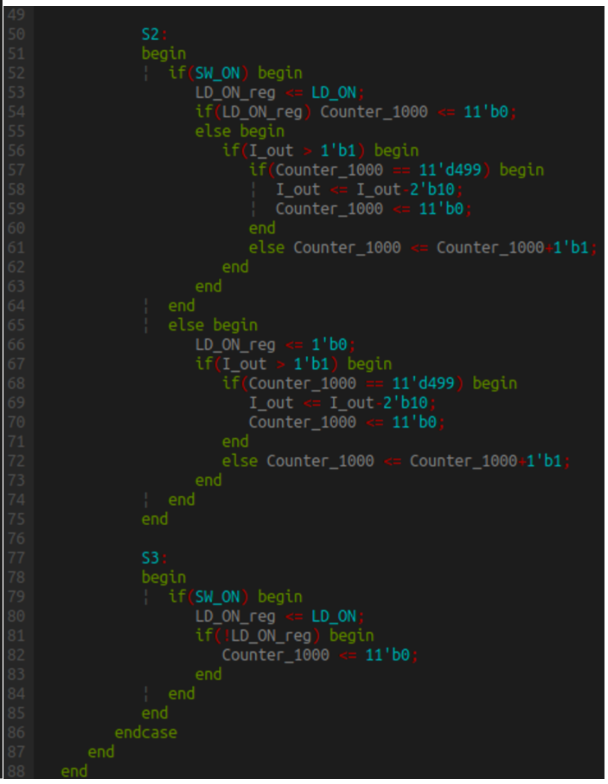
Verilog



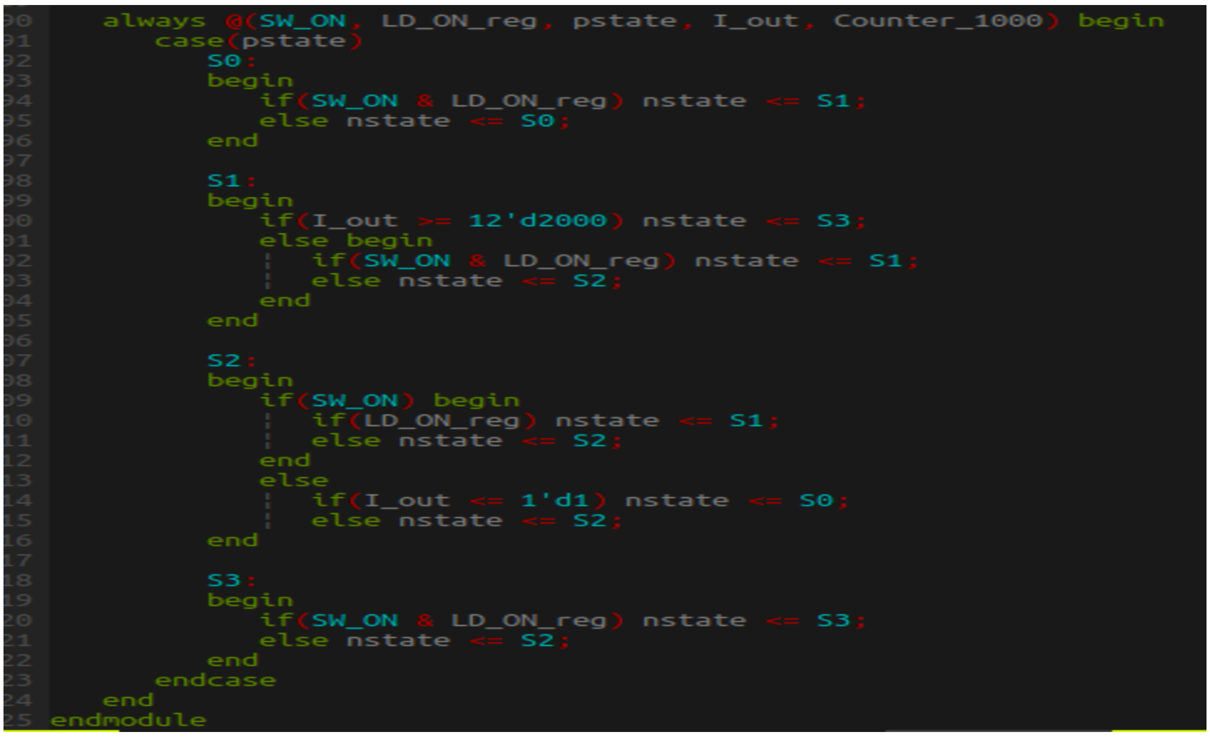
```verilog
module LD_Driver_v_jin(
   output reg [11:0] I_out,
   input SW_ON, LD_ON,
   input CLK, Clrn
);
   reg [10:0] Counter_1000; //counter using for check 10ms
   reg LD_ON_reg;
   reg [1:0] pstate, nstate;

   //Encode the states
   parameter S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;
```



```verilog
   //state transition
   always @(posedge CLK, negedge Clrn) begin
      if(~Clrn) begin
         pstate <= S0;
         I_out <= 12'b0;
         Counter_1000 <= 11'b0;
      end
      else begin
         pstate <= nstate; //clocked operation
         case(pstate)
            S0:
            begin
               LD_ON_reg <= LD_ON;
               if(SW_ON) begin
                  I_out <= 12'b0;
                  Counter_1000 <= 11'b0;
               end
            end

            S1:
            begin
               if(SW_ON) begin
                  LD_ON_reg <= LD_ON;
                  if(LD_ON_reg) begin
                     if(I_out < 12'd2000) begin
                        if(Counter_1000 == 11'd999) begin
                           I_out <= I_out+2'b10;
                           Counter_1000 <= 11'b0;
                        end
                        else Counter_1000 <= Counter_1000+1'b1;
                     end
                  end
                  else Counter_1000 <= 11'b0;
               end
               else LD_ON_reg <= 1'b0;
            end
```

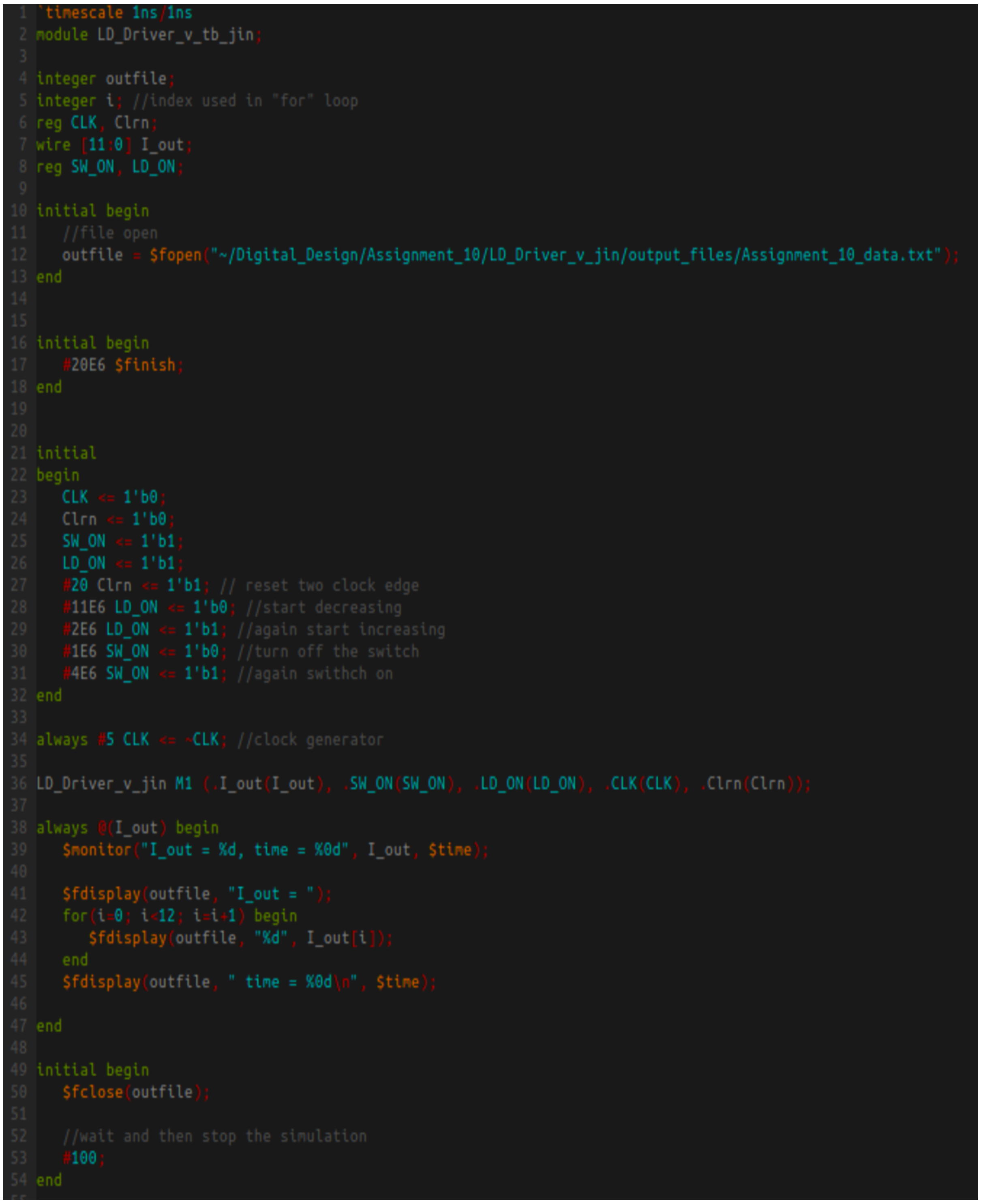


```verilog

            S2:
            begin
               if(SW_ON) begin
                  LD_ON_reg <= LD_ON;
                  if(LD_ON_reg) Counter_1000 <= 11'b0;
                  else begin
                     if(I_out > 1'b1) begin
                        if(Counter_1000 == 11'd499) begin
                           I_out <= I_out-2'b10;
                           Counter_1000 <= 11'b0;
                        end
                        else Counter_1000 <= Counter_1000+1'b1;
                     end
                  end
               end
               else begin
                  LD_ON_reg <= 1'b0;
                  if(I_out > 1'b1) begin
                     if(Counter_1000 == 11'd499) begin
                        I_out <= I_out-2'b10;
                        Counter_1000 <= 11'b0;
                     end
                     else Counter_1000 <= Counter_1000+1'b1;
                  end
               end
            end

            S3:
            begin
               if(SW_ON) begin
                  LD_ON_reg <= LD_ON;
                  if(!LD_ON_reg) begin
                     Counter_1000 <= 11'b0;
                  end
               end
            end
         endcase
      end
   end
```



```verilog
   always @(SW_ON, LD_ON_reg, pstate, I_out, Counter_1000) begin
      case(pstate)
         S0:
         begin
            if(SW_ON & LD_ON_reg) nstate <= S1;
            else nstate <= S0;
         end

         S1:
         begin
            if(I_out >= 12'd2000) nstate <= S3;
            else begin
               if(SW_ON & LD_ON_reg) nstate <= S1;
               else nstate <= S2;
            end
         end

         S2:
         begin
            if(SW_ON) begin
               if(LD_ON_reg) nstate <= S1;
               else nstate <= S2;
            end
            else
               if(I_out <= 1'd1) nstate <= S0;
               else nstate <= S2;
         end

         S3:
         begin
            if(SW_ON & LD_ON_reg) nstate <= S3;
            else nstate <= S2;
         end
      endcase
   end
endmodule
```

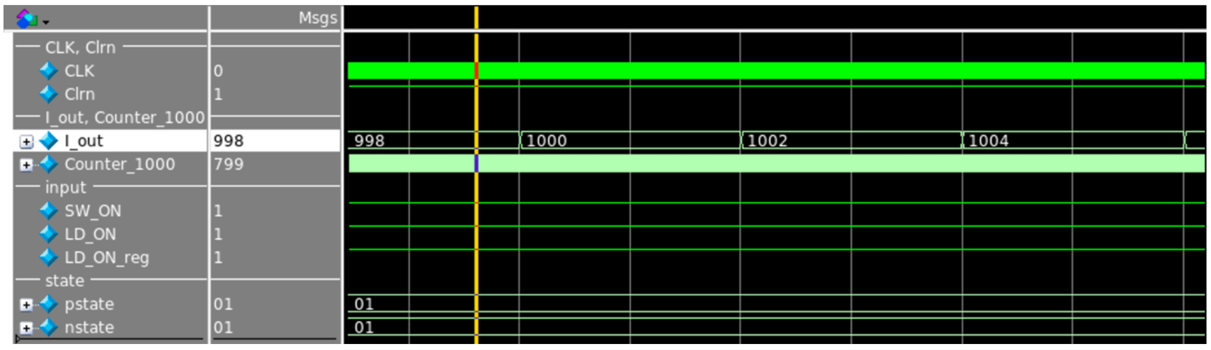
Testbench



```verilog
`timescale 1ns/1ns
module LD_Driver_v_tb_jin;

integer outfile;
integer i; //index used in "for" loop
reg CLK, Clrn;
wire [11:0] I_out;
reg SW_ON, LD_ON;

initial begin
   //file open
   outfile = $fopen("~/Digital_Design/Assignment_10/LD_Driver_v_jin/output_files/Assignment_10_data.txt");
end


initial begin
   #20E6 $finish;
end


initial
begin
   CLK <= 1'b0;
   Clrn <= 1'b0;
   SW_ON <= 1'b1;
   LD_ON <= 1'b1;
   #20 Clrn <= 1'b1; // reset two clock edge
   #11E6 LD_ON <= 1'b0; //start decreasing
   #2E6 LD_ON <= 1'b1; //again start increasing
   #1E6 SW_ON <= 1'b0; //turn off the switch
   #4E6 SW_ON <= 1'b1; //again swithch on
end

always #5 CLK <= ~CLK; //clock generator

LD_Driver_v_jin M1 (.I_out(I_out), .SW_ON(SW_ON), .LD_ON(LD_ON), .CLK(CLK), .Clrn(Clrn));

always @(I_out) begin
   $monitor("I_out = %d, time = %0d", I_out, $time);

   $fdisplay(outfile, "I_out = ");
   for(i=0; i<12; i=i+1) begin
      $fdisplay(outfile, "%d", I_out[i]);
   end
   $fdisplay(outfile, " time = %0d\n", $time);

end

initial begin
   $fclose(outfile);

   //wait and then stop the simulation
   #100;
end
```

RTL Simulation

1) Increasing I_out



SW_ON = 1

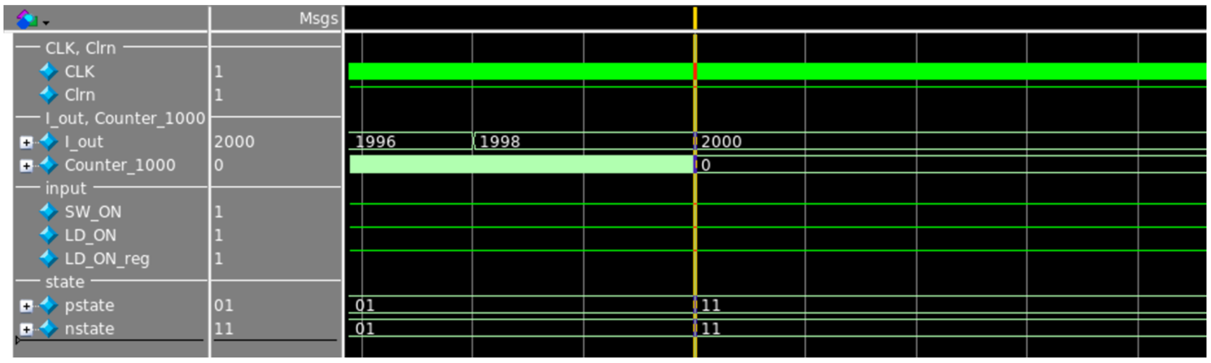
LD_ON = 1

LD_ON_reg = 1

State : S1 (01)

I_out increase in 2 unit time blocks

2) Keeping I_out



SW_ON = 1

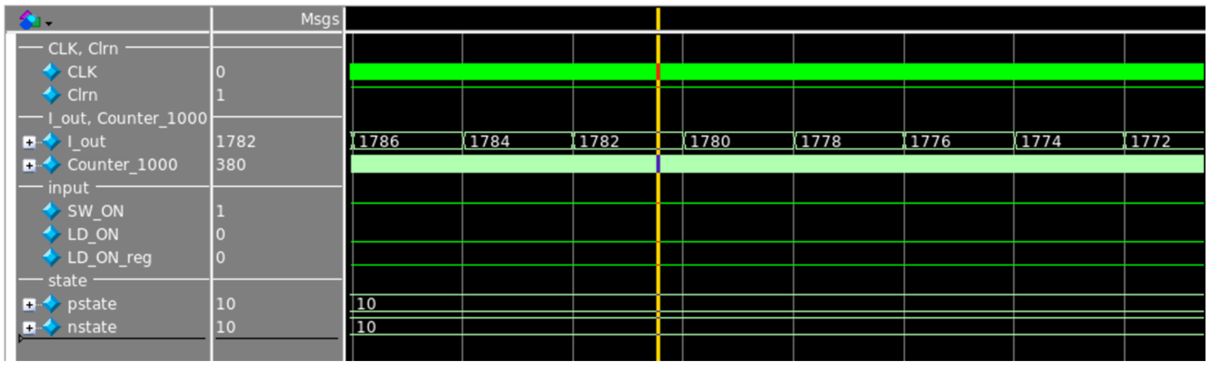
LD_ON = 1

LD_ON_reg = 1

State : S3 (11)

Keeping I_out (from 10ms)

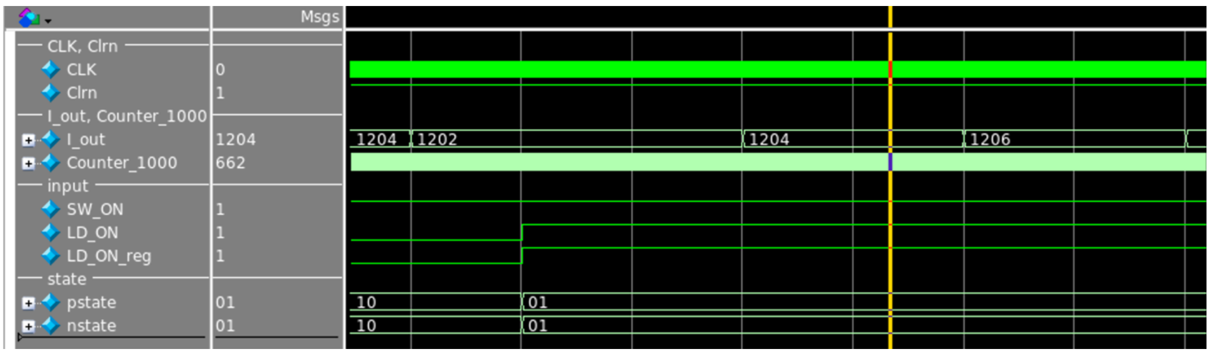
3) Decreasing I_out



SW_ON = 1
LD_ON = 0
LD_ON_reg = 0
State : S2 (10)

I_out decrease in 1 unit time block

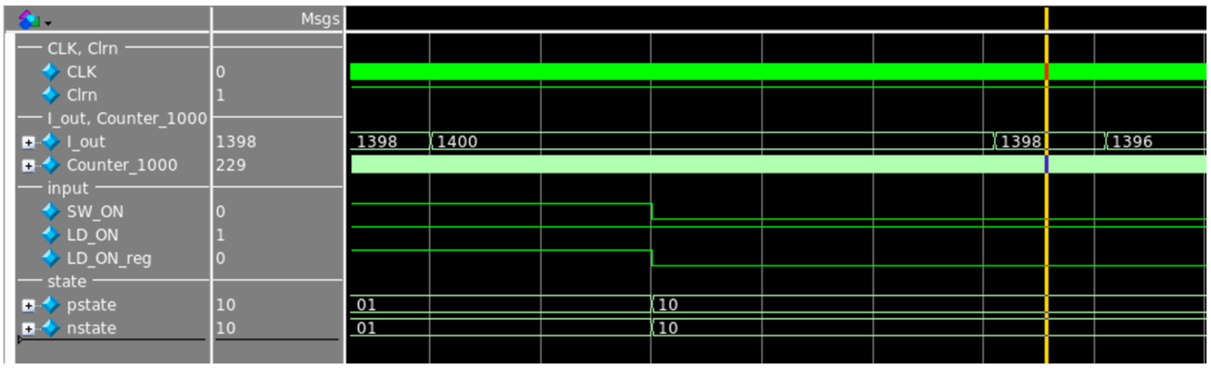
4) Increasing I_out again



SW_ON = 1
LD_ON = 1
LD_ON_reg = 1
State : S1 (01)

I_out increase in 2 unit time blocks again

5) Switching Off



SW_ON = 0

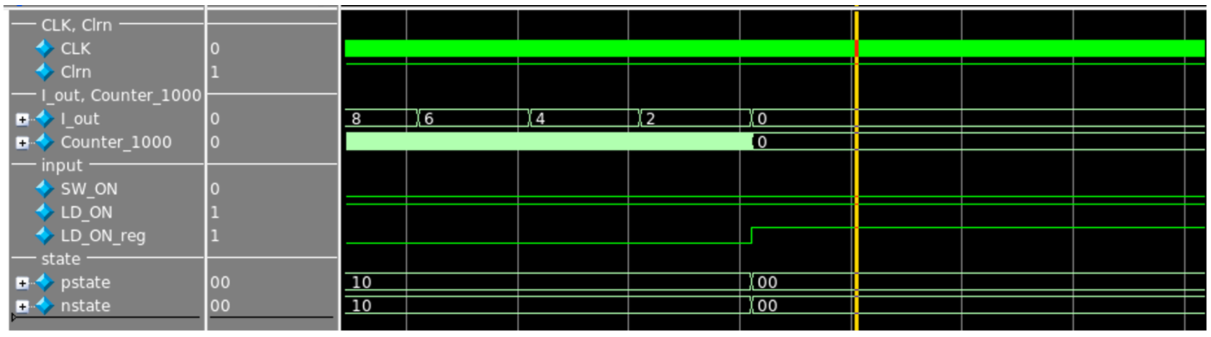
LD_ON = 1

LD_ON_reg = 0

State : S2 (10) (decreasing I_out first)

I_out decrease in 1 unit time block

6) After end of decreasing, system is on initial state



SW_ON = 0

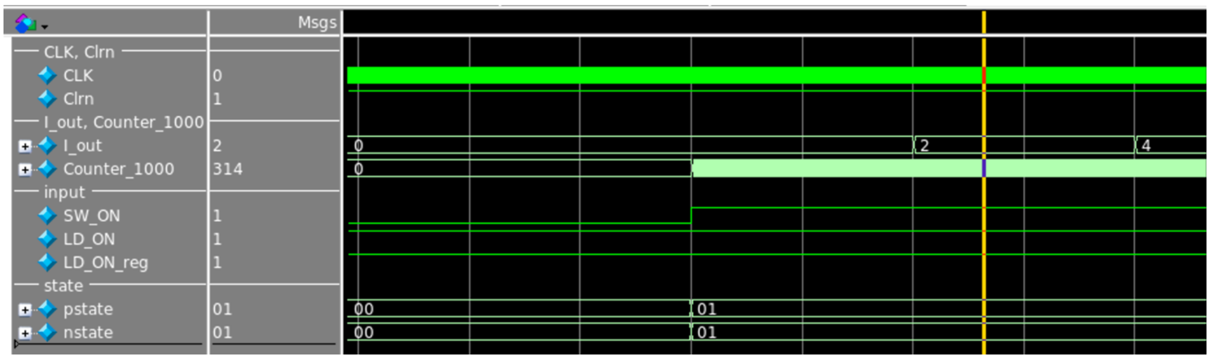
LD_ON = 1

LD_ON_reg = 1

State : S0 (00) (initial state)

Stay on initial state

7) Restart



SW_ON = 1

LD_ON = 1

LD_ON_reg = 1

State : S1 (01)

I_out increase in 2 unit time blocks