HW#14 디지털 회로 설계 및 언어 월수 9:00~10:15 2015104027 박정진

Separate code to next_state and output from ASMD

```
module Sequential_Binary_Multiplier
  #(parameter dp_width = 5 parameter BC_size = 3) (
        output [2*dp_width-1:0] Product,
        output Ready,
        input [dp_width-1:0] Multiplicand, Multiplier,
        input Start, clock, reset_b
);
  parameter S_idle = 3'b001, S_add = 3'b010, S_shift=3'b100;
  reg [2:0] state, next_state;
  reg [dp_width-1:0] A, B, Q; //Sized for datapath
  reg C;
  reg [BC_size-1:0] P;
  reg Load_regs, Decr_P, Add_regs, Shifth_regs;

  //Miscellaneous combinational logic
  assign Product = {A, Q};
  wire Zero = (P==0); // counter is zero
  wire Ready=(state == S_idle); // controller status

  //control unit
  always@(posedge clock, negedge reset_b)
        if(~reset_v) state <= S_idle;
        else state <= next_state;

  // next state logic
  always@(state, Start, Q[0], Zero) begin
        next_state <= S_idle;
        case(state)
              S_idle: if(Start) next_state <= S_add;
              S_add: next_state <= S_shift;
              S_shift:
                    begin
                      if(Zero) next_state <= S_idle;
                      else next_state <= S_add;
                    end
              default: next_state <= S_idle;
```

```
    end

    // output logic
    always@(state, Start, Q[0], Zero) begin
            Load_regs <= 0;
            Decr_P <= 0;
            Add_regs <= 0;
            Shift_regs <= 0;
            case(state)
                    S_idle: if(Start) Load_regs <= 1;
                    S_add:
                            begin
                              Decr_P <= 1;
                              if(Q[0]) Add_regs <= 1;
                            end
                    S_shift:
                            Shift_regs <= 1;
    end

    //datapath unit
    always@(posedge clock) begin
            if(Load_regs) begin
                    P <= dp_width;
                    A <= 0;
                    C <= 0;
                    B <= Multiplicand;
                    Q <= Multiplier;
            end
            if(Add_regs) {C,A} <= A+B;
            if(Shift_regs) {C,A,Q} <= ({C,A,Q} >> 1);
            if(Decr_P) P <= P-1;
    end
endmodule
```

Design LD Driver

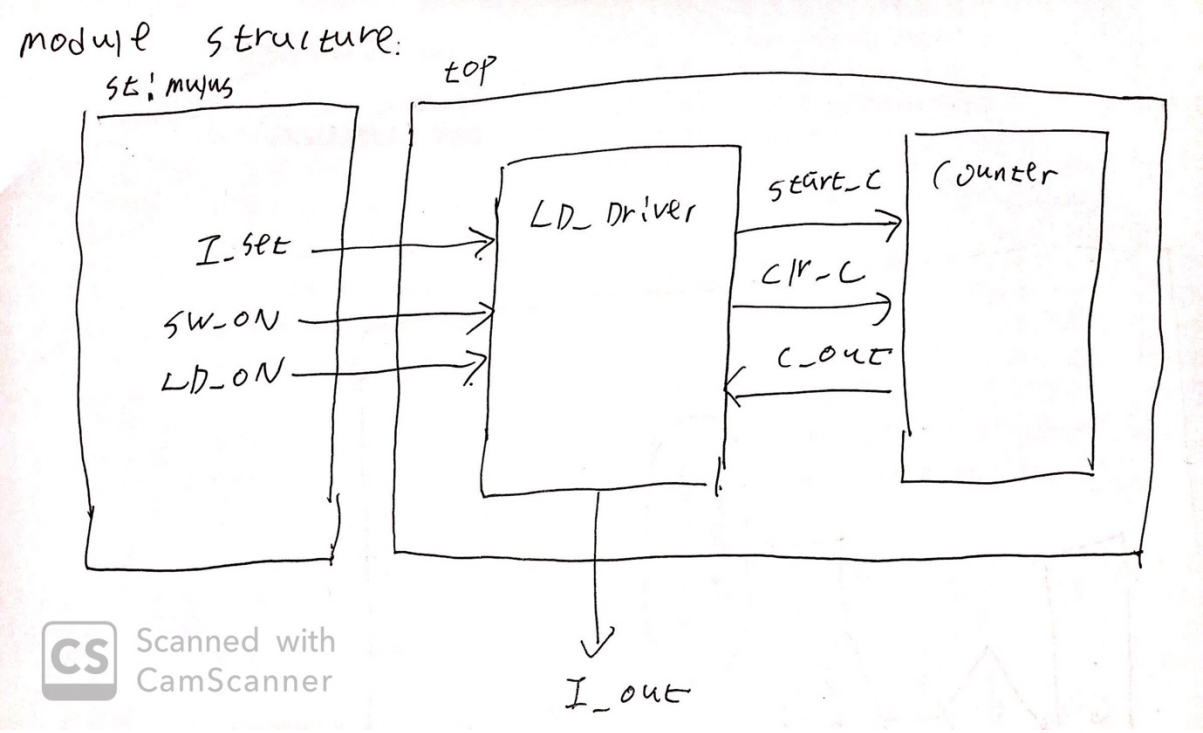
Design LD Driver with

1 12-bits binary counter I_Out (Predefined value of I_out is 2000)
1 structured module Counter_1E6
3 External input(main input) SW_ON, LD_ON, 12-bits I_set
2 12 bits Flip-Flop I_set_reg, I_incr
1 Flip-Flop LD_ON_reg
1 Master Clock 100MHZ (period = 10ns)

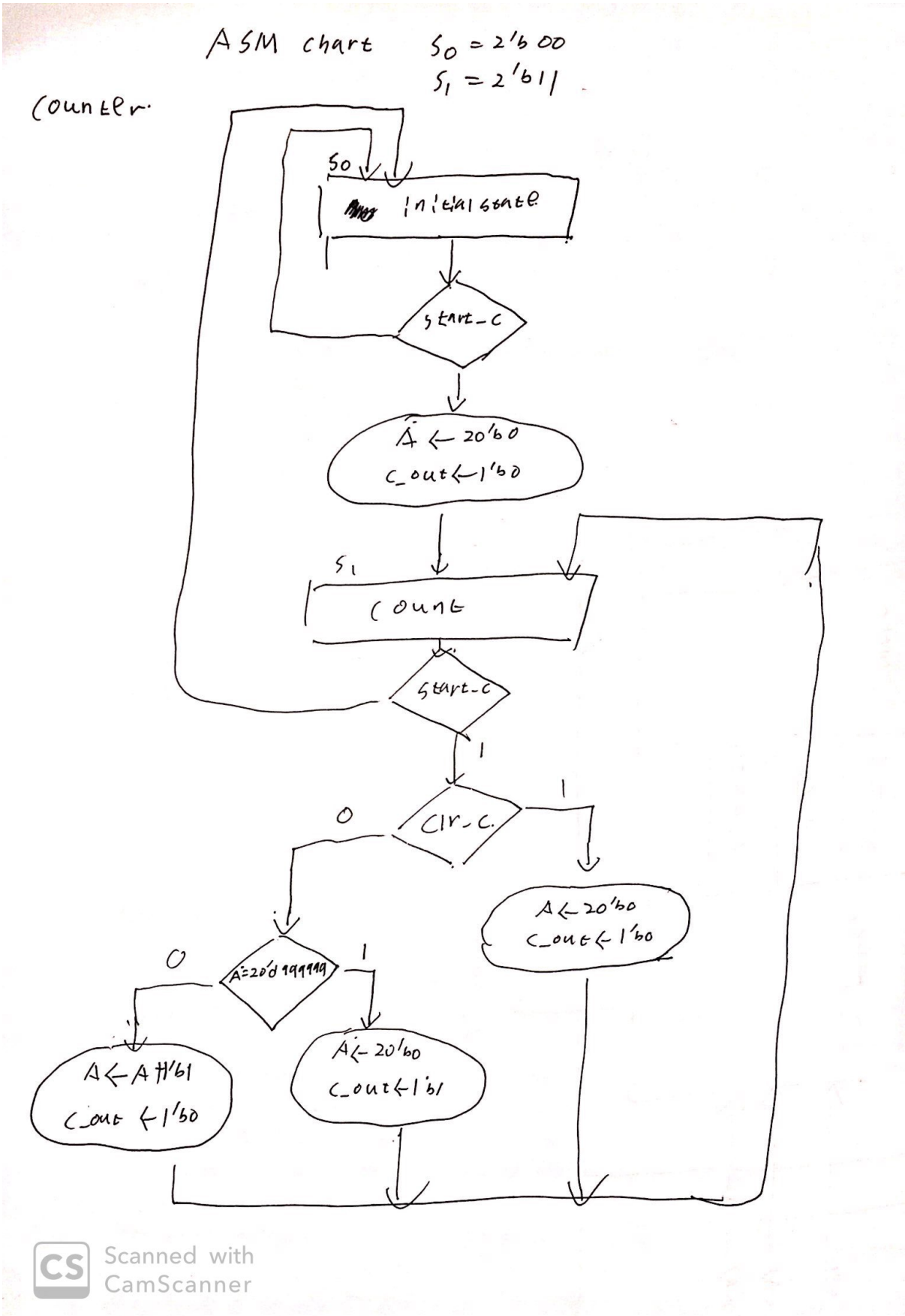
States

initial state : S_0
increase I_out : S_1
decrease I_out : S_2
keep I_out : S_3
set high value from S_3 : S_4
set low value from S_3 : S_5

Module Structure



ASM Chart of Counter_1E6



ASM Chart of LD_Driver

ASM Chart

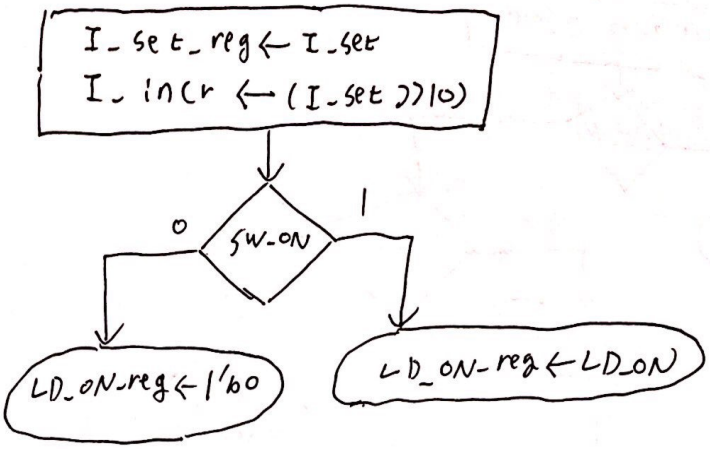
CLD - Driver.

$S0 = 3'b000$ $S3 = 3'b011$

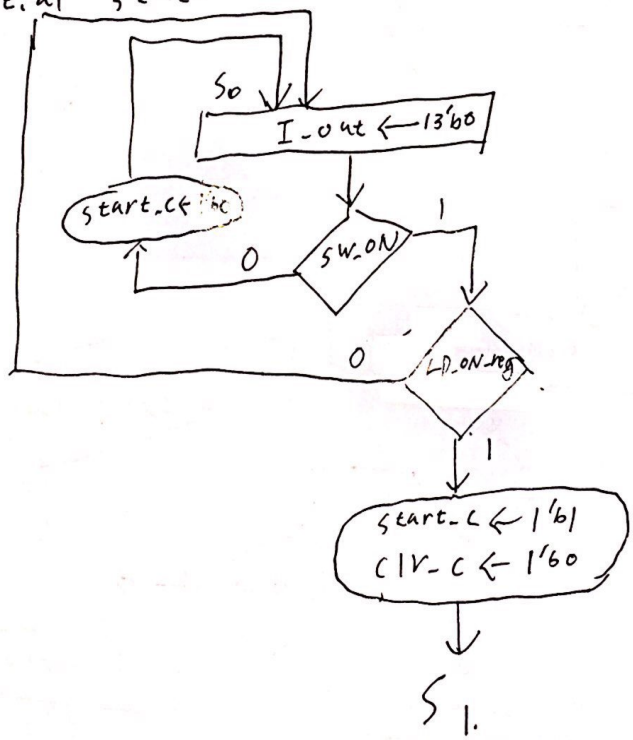
$S1 = 3'b001$ $S4 = 3'b100$

$S2 = 3'b010$ $S5 = 3'b101$

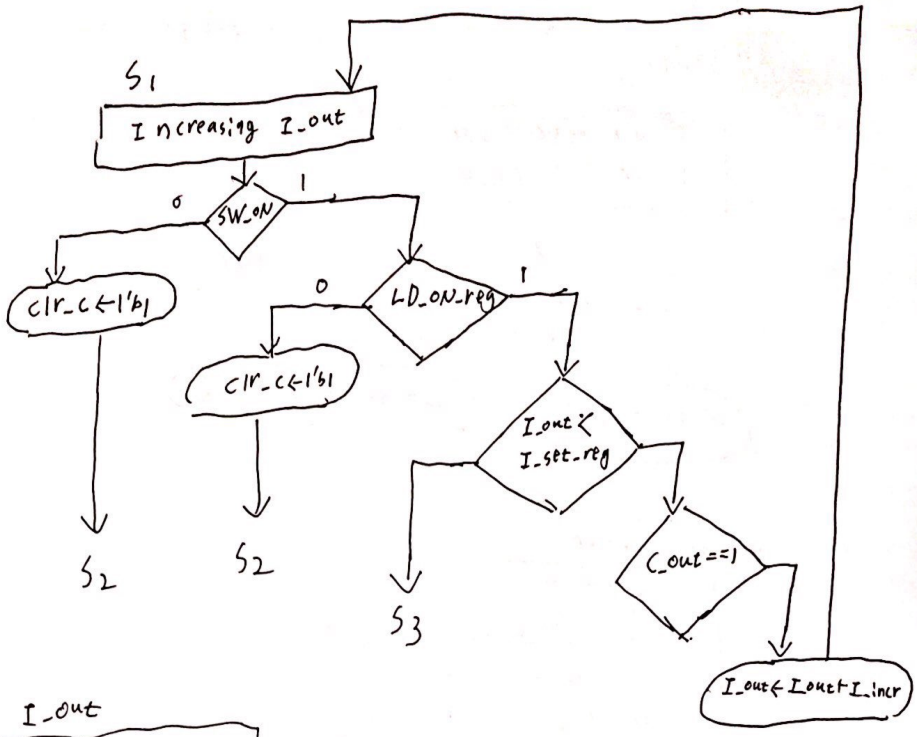
All state



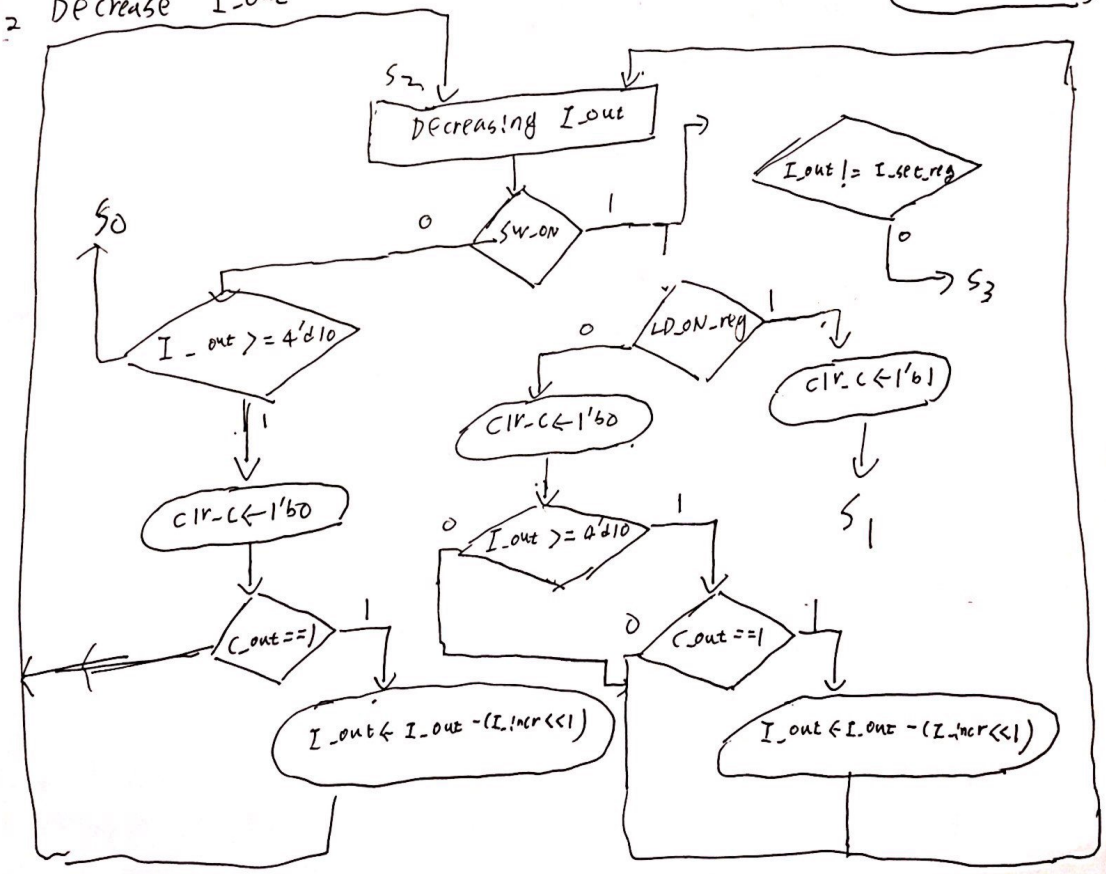
$S0$ initial state



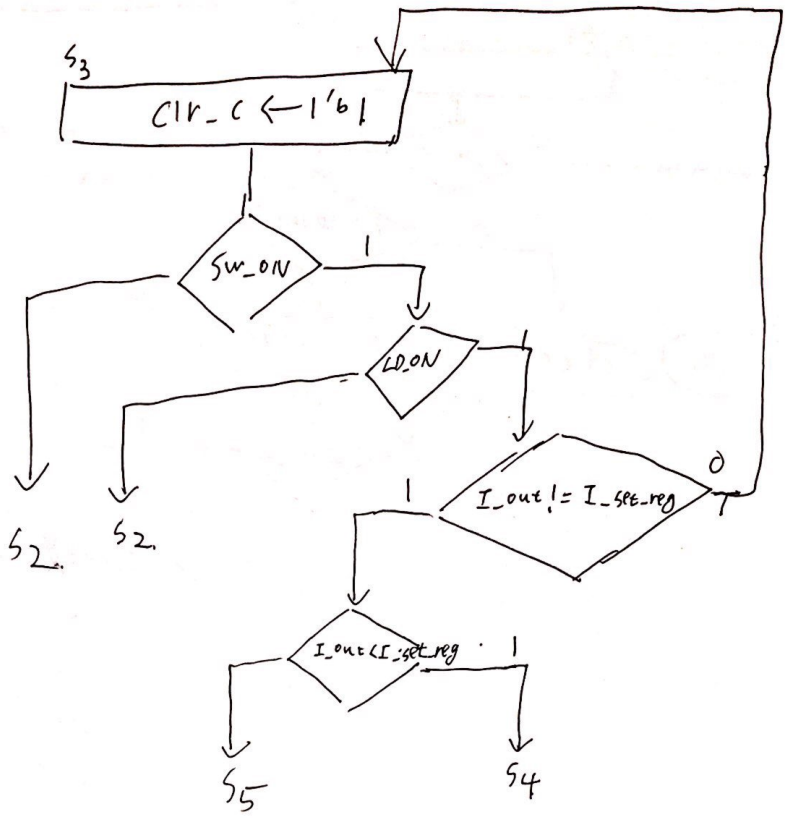
S1 Increase I_out



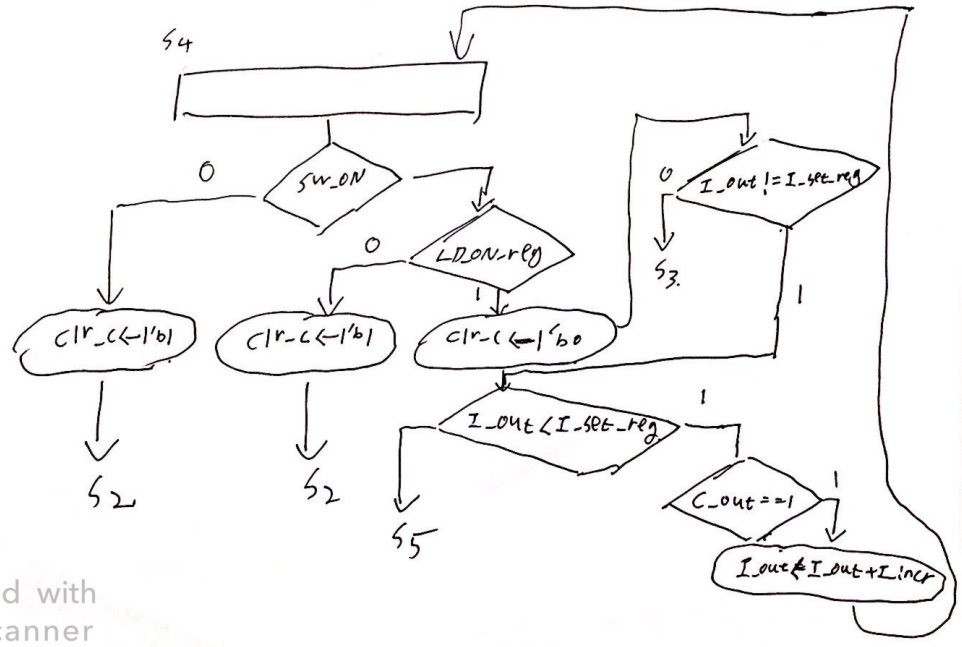
S2 Decrease I_out



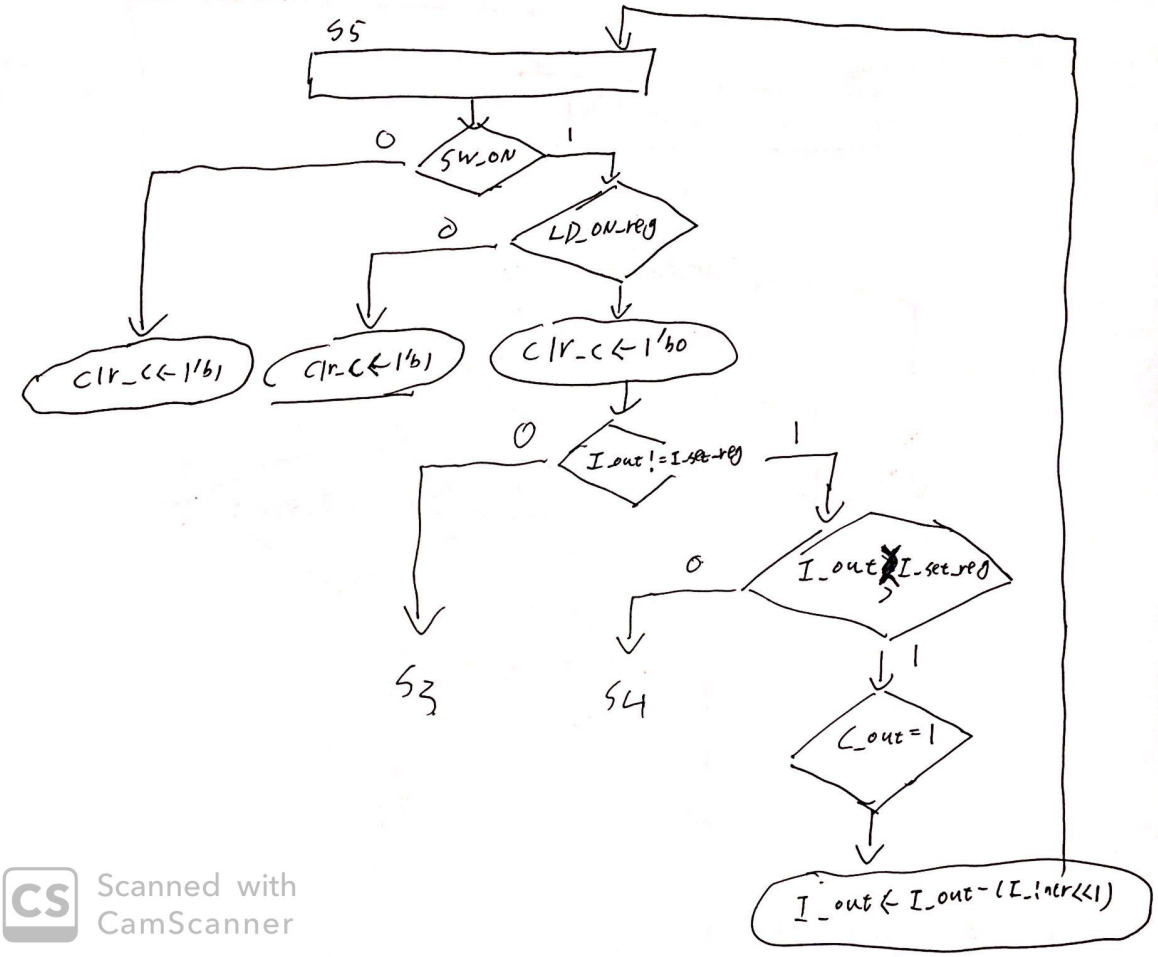
S_3 keeping I_out



S_4 set high value from S_3



S_5 get low value from S_3



Verilog

top.v

```
module top(
   output [12:0] I_out,
   input [12:0] I_set,
   input SW_ON, LD_ON,
   input CLK, Clrn
);
   wire Start_C, Clr_C, C_out;

   Counter_1E6_ASM_v_jin C1 (
         .C_out(C_out),
         .Start_C(Start_C),
         .Clr_C(Clr_C),
         .CLK(CLK),
         .Clrn(Clrn));

   LD_Driver_ASM_v_jin LD1 (
         .I_out(I_out),
         .Start_C(Start_C),
         .Clr_C(Clr_C),
         .I_set(I_set),
         .SW_ON(SW_ON),
         .LD_ON(LD_ON),
         .C_out(C_out),
         .CLK(CLK),
         .Clrn(Clrn));
endmodule
```

Counter_1E6_ASM_v_jin.v

```
module Counter_1E6_ASM_v_jin(
   output reg C_out,
   input Start_C, Clr_C,
   input CLK, Clrn
);

   reg [19:0] A; //Counter_1E6
   reg [1:0] pstate, nstate; // state
   //S0 : initial state, S1 : count
   parameter S0 = 2'b00, S1=2'b11;

   //state transition for control logic
   always @(posedge CLK, negedge Clrn) begin
         if(~Clrn) begin
               pstate <= S0;
               A <= 20'b0;
               C_out <= 1'b0;
         end
         else begin
               pstate <= nstate; //clocked operation
               //Register Transfer operation controlled by external input
               case(pstate)
                     S0:
                     begin
                       if(Start_C) begin
                             A <= 20'b0;
                             C_out <= 1'b0;
                       end
                     end
                     S1:
                     begin
                       if(Start_C) begin
                             if(Clr_C) begin
                                   A <= 20'b0;
                                   C_out <= 1'b0;
```

```
                        end
                        else begin
                                if(A==20'd999999) begin
                                        A <= 20'b0;
                                        C_out <= 1'b1;
                                end
                                else begin
                                        A <= A +  1'b1;
                                        C_out <= 1'b0;
                                end
                        end
                    end
                end
            endcase
        end
    end

    // decide next state
    always@(pstate, Start_C) begin
        case(pstate)
            S0:
            begin
                if(Start_C) nstate <= S1;
                else nstate <= S0;
            end
            S1:
            begin
                if(Start_C) nstate <= S1;
                else nstate <= S0;
            end
        endcase
    end
endmodule
```

LD_Driver_ASM_v_jin.v

```
module LD_Driver_ASM_v_jin(
   output reg [12:0] I_out,
   output reg Start_C,
   output reg Clr_C,
   input [12:0] I_set,
   input SW_ON, LD_ON, C_out,
   input CLK, Clrn
);
   reg LD_ON_reg; // store LD_ON
   reg [12:0] I_set_reg; //store i_set value
   reg [12:0] I_incr; // setting incremental
   reg [2:0] pstate, nstate;

   //Encode the states
   parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100,
S5=3'b101;

   //state transition
   always @(posedge CLK, negedge Clrn) begin
         if(~Clrn) begin
               pstate <= S0;
               I_out <= 13'b0;
               Start_C <= 1'b0;
               Clr_C <= 1'b0;
               LD_ON_reg <= 1'b0;
               I_set_reg <= 13'b0;
               I_incr <= 13'b0;
         end
         else begin
               pstate <= nstate; //clocked operation
               // if SW_ON == 0, set LD_ON_reg 1'b0
               if(SW_ON) LD_ON_reg <= LD_ON;
               else LD_ON_reg <= 1'b0;
               // store i_set
               I_set_reg <= I_set;
```

```
// make incr
I_incr <= (I_set >> 10); // divide by 1024(approx 1000)

//Register Transter operation
case(pstate)
        S0:
        begin
          I_out <= 13'b0;
          if(SW_ON) begin
                  Start_C <= 1'b1;
                  Clr_C <= 1'b0;
          end
          else Start_C <= 1'b0;

        end
        S1:
        begin
          if(SW_ON) begin
                  if(LD_ON_reg) begin
                          Clr_C <= 1'b0;
                          if(I_out < I_set_reg) begin
                                  if(C_out == 1'b1) begin
                                     I_out <= I_out+ I_incr;
                                  end
                          end
                  end
                  else Clr_C <= 1'b1;
          end
          else Clr_C <= 1'b1;
        end

        S2:
        begin
          if(SW_ON) begin
                  if(!LD_ON_reg) begin
                          Clr_C <= 1'b0;
                          if(I_out >= 4'b1010) begin
```

```
                    if(C_out == 1'b1) begin
                      //multiple by 2
                      I_out <= I_out-(I_incr << 1);
                    end
                end
            end
            else Clr_C <= 1'b1;
        end
        else begin
            if(I_out >= 4'b1010) begin
                Clr_C <= 1'b0;
                if(C_out == 1'b1) begin
                    I_out <= I_out-(I_incr << 1);
                end
            end
            else Clr_C <= 1'b1;
        end
    end
    S3:
    begin
        Clr_C <= 1'b1;
    end

    S4:
    begin
        if(SW_ON) begin
            if(LD_ON_reg) begin
                Clr_C <= 1'b0;
                if(I_out < I_set_reg) begin
                    if(C_out == 1'b1) begin
                      I_out <= I_out+ I_incr;
                    end
                end
            end
            else Clr_C <= 1'b1;
        end
        else Clr_C <= 1'b1;
```

```
                end

                S5:
                begin
                  if(SW_ON) begin
                        if(LD_ON_reg) begin
                              Clr_C <= 1'b0;
                              if(I_out > I_set_reg) begin
                                    if(C_out == 1'b1) begin
                                      I_out <= I_out-(I_incr<<1);
                                    end
                              end
                        end
                        else Clr_C <= 1'b1;
                  end
                  else Clr_C <= 1'b1;
                end

            endcase
      end
end

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

            S1:
            begin
                  if(SW_ON & LD_ON_reg) begin
                        if(I_out >= I_set_reg) nstate <= S3;
                        else nstate <= S1;
                  end
                  else nstate <= S2;
```

```
            end

            S2:
            begin
                if(SW_ON) begin

                  if(LD_ON_reg) nstate <= S1;
                  else nstate <= S2;
                end
                else
                  if(I_out <= 4'b1010) nstate <= S0;
                  else nstate <= S2;
            end

            default:
            begin
                if(SW_ON & LD_ON_reg) begin
                  if (I_out != I_set_reg) begin
                        if(I_out < I_set_reg) nstate <= S4; //i_set incr
                        else nstate <= S5;
                  end
                  else nstate <= S3;
                end
                else nstate <= S2;
            end
        endcase
  end
endmodule
```

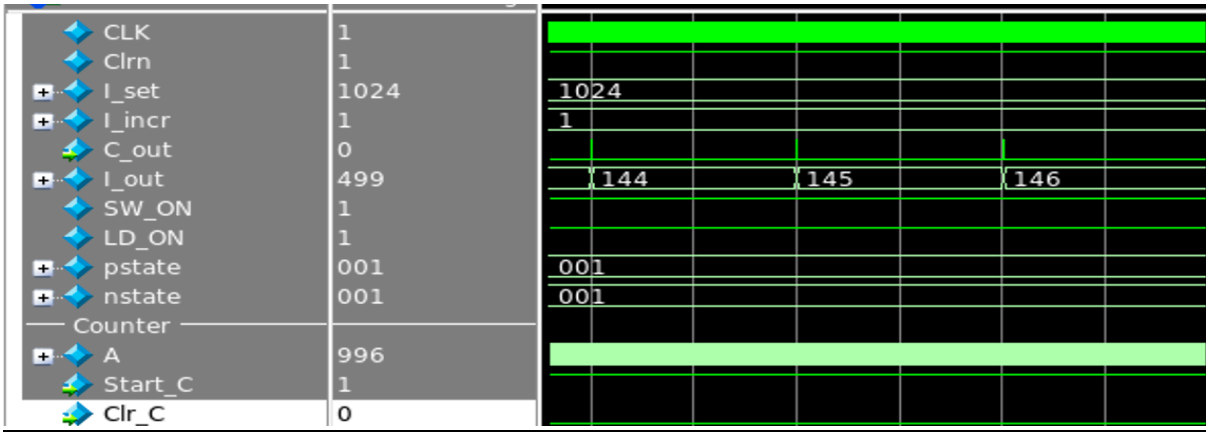
sti.v

```
`timescale 1ns/1ns
module sti;

  reg CLK, Clrn;
  reg [12:0] I_set;
  wire [12:0] I_out;
  reg SW_ON, LD_ON;

  //set end time
  initial begin
        #40E6 $finish;
  end

  initial
  begin
        CLK <= 1'b0;
        Clrn <= 1'b0;
        SW_ON <= 1'b1;
        LD_ON <= 1'b1;
        I_set <= 13'd1024;
        #20 Clrn <= 1'b1; // reset two clock edge
        #5E6 I_set <= 13'd2048;
        #2E6 I_set <= 13'd1024;
        #4E6 I_set <= 13'd2048;
        #8E6 I_set <= 13'd1024;
        #6E6 LD_ON <= 1'b0; //start decreasing
        #3E6 I_set <= 13'd2048;
        #2E6 I_set <= 13'd1024;
        #2E6 LD_ON <= 1'b1; //again start increasing
        #2E6 SW_ON <= 1'b0; //turn off the switch
        #1E6 SW_ON <= 1'b1; //again swithch on
  end

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

```
    top t1 (
            .I_out(I_out),
            .I_set(I_set),
            .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);
    end
endmodule
```

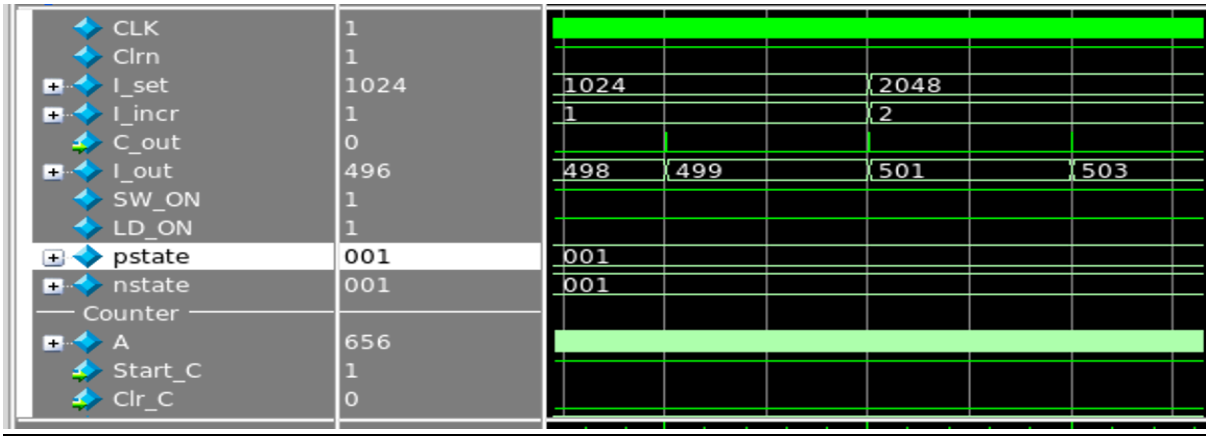
RTL Simulation

1) Increasing I_out



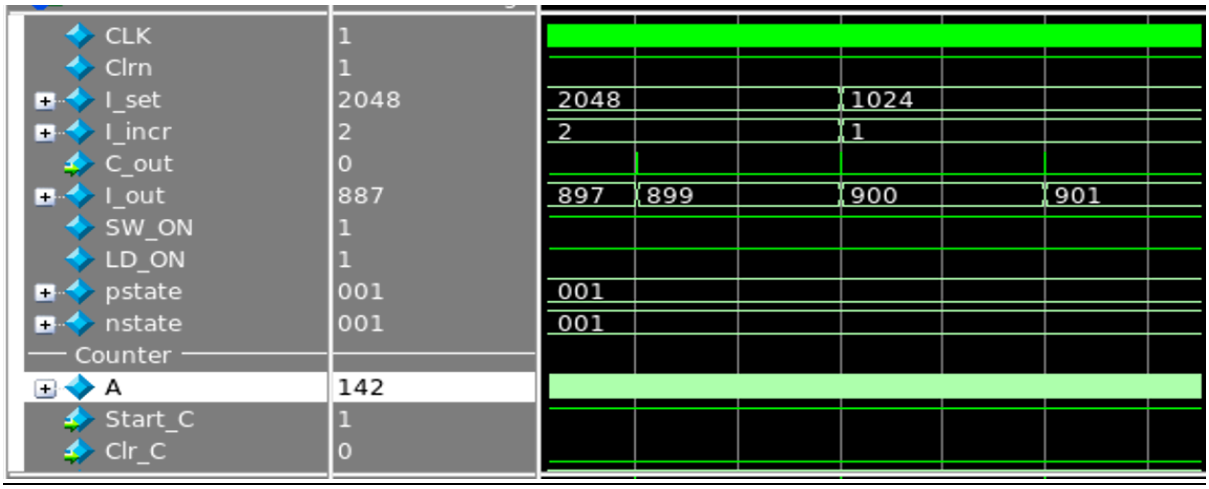
SW_ON = 1
LD_ON = 1
LD_ON_reg = 1
I_set = 1024
I_incr = 1
State : S1 (001)

2) change I_set during increasing I_out



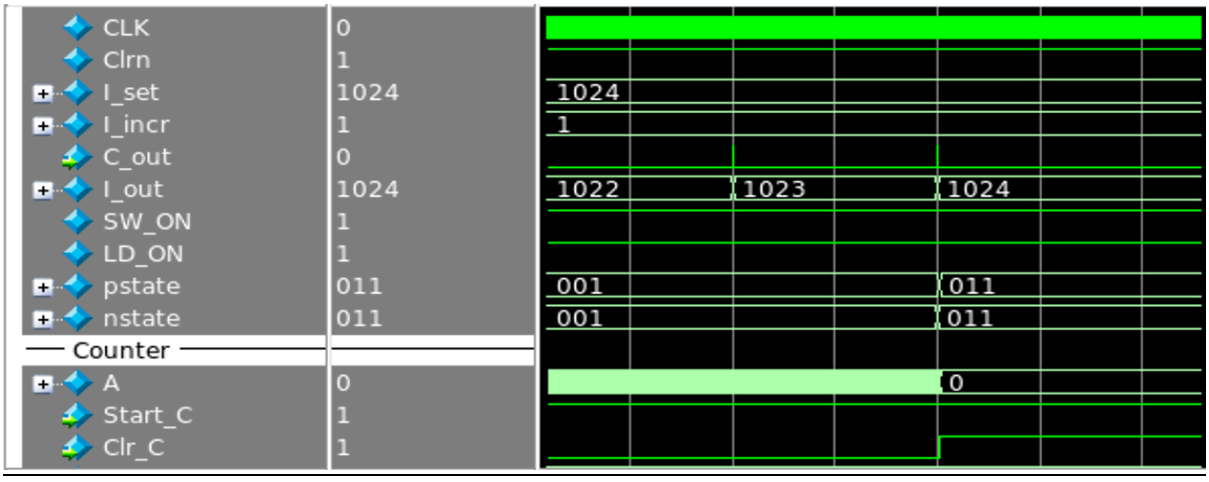
SW_ON = 1
LD_ON = 1
LD_ON_reg = 1
I_set = 2048
I_incr = 2
State : S1 (001)

3) change again I_set during increasing I_out



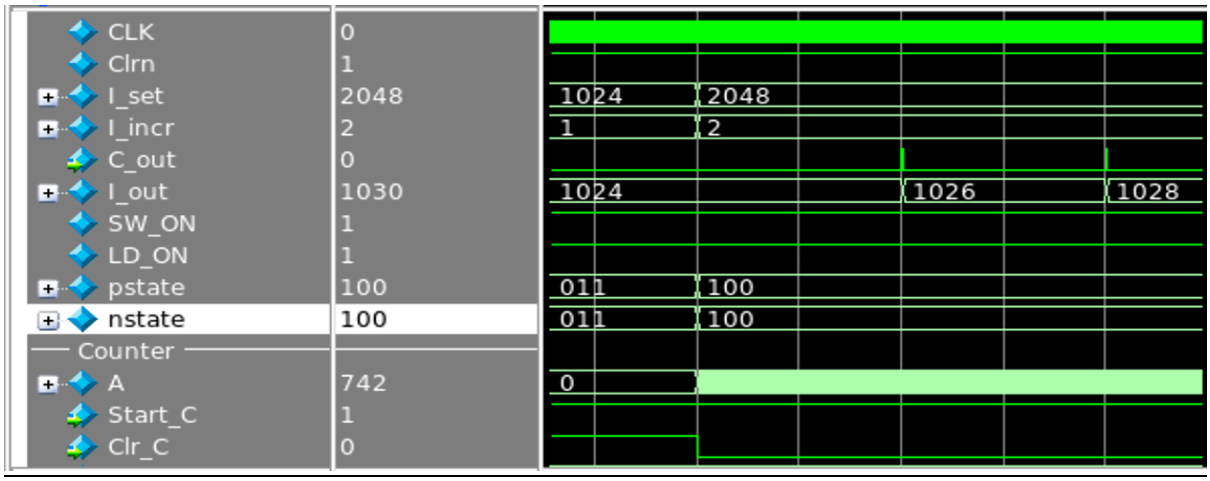
SW_ON = 1
LD_ON = 1
LD_ON_reg = 1
I_set = 1024
I_incr = 1
State : S1 (001)

4) reach I_set value



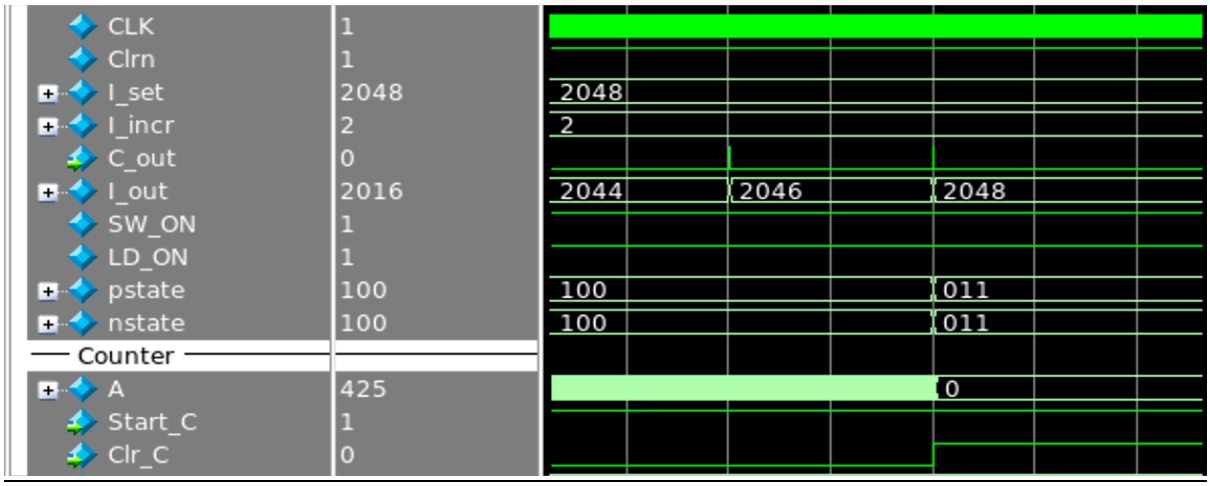
SW_ON = 1
LD_ON = 1
LD_ON_reg = 1
I_set = 1024
I_incr = 1
State : S3 (011)

5) change High value of I_set when keep I_out



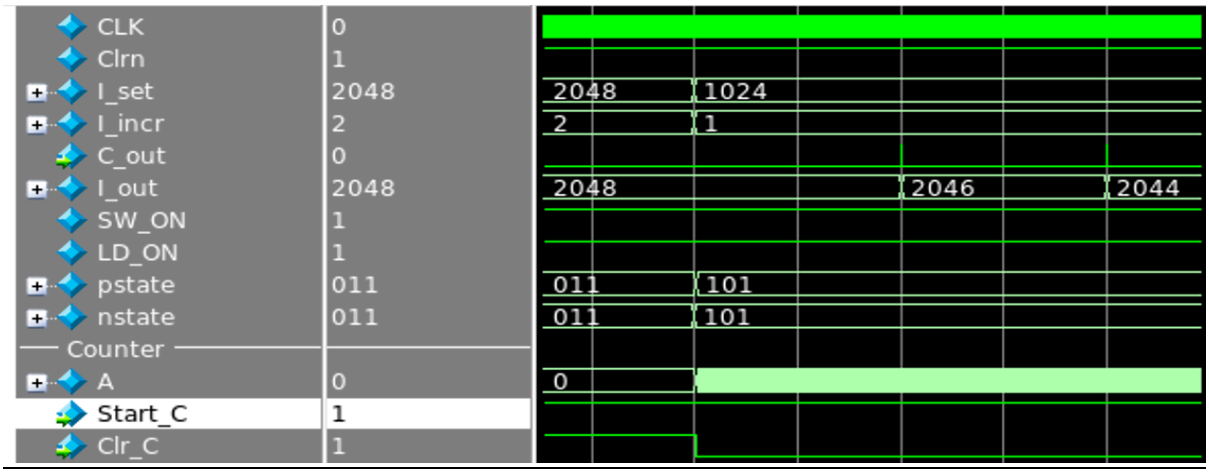
SW_ON = 1
LD_ON = 1
LD_ON_reg = 1
I_set = 2048
I_incr = 2
State : S4 (100)

6) reach High value of I_set



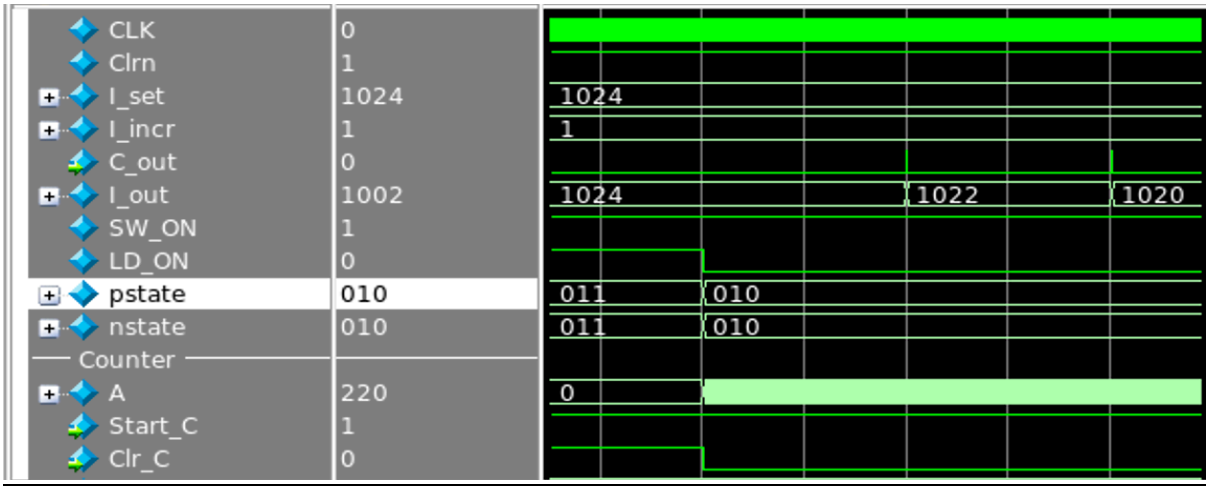
SW_ON = 1
LD_ON = 1
LD_ON_reg = 1
I_set = 2048
I_incr = 2
State : S3 (011)

7) change low value of I_set when keep I_out



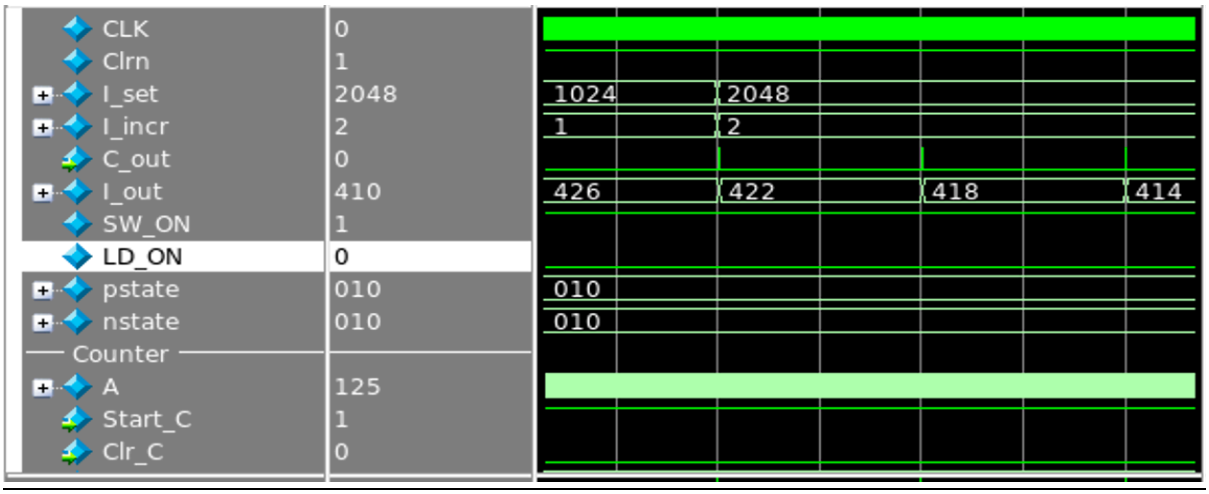
SW_ON = 1
LD_ON = 1
LD_ON_reg = 1
I_set = 1024
I_incr = 1 (I_decr : 2)
State : S5 (101)

8) Decreasing I_out when LD_ON is 0



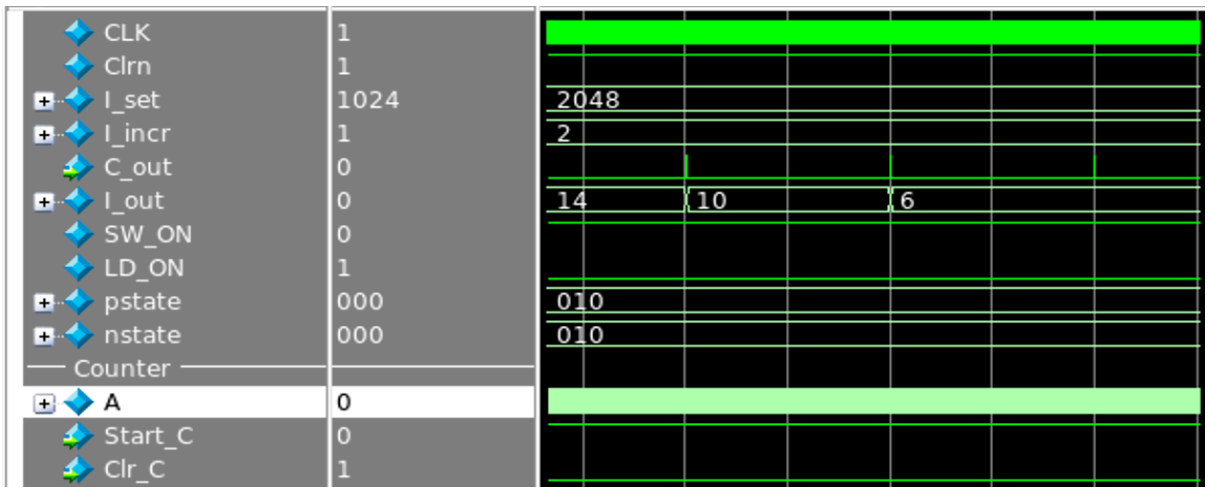
SW_ON = 1
LD_ON = 0
LD_ON_reg = 0
I_set = 1024
I_incr = 1 (I_decr : 2)
State : S2 (010)

9) change I_set during decreasing I_out



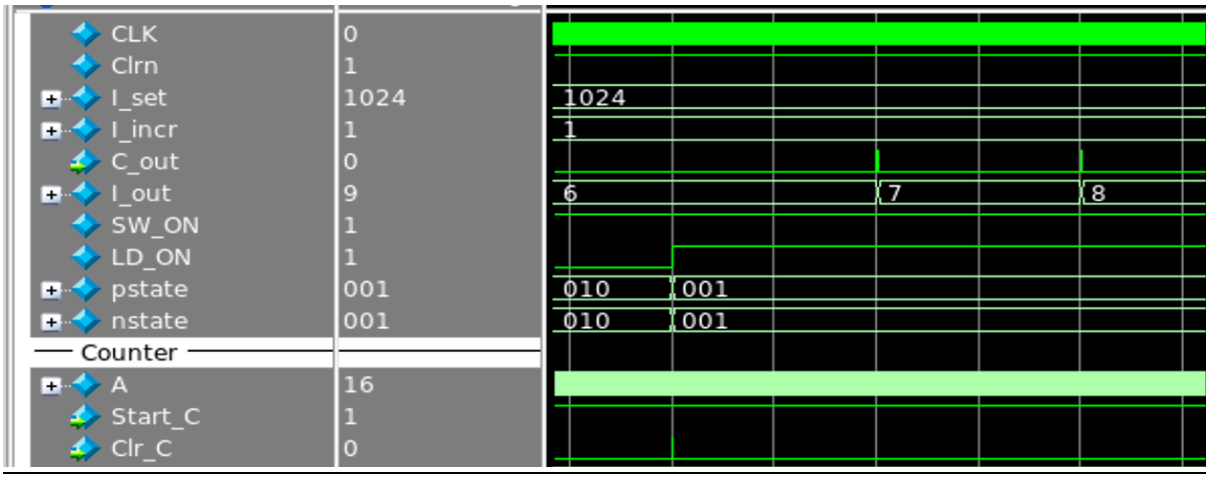
SW_ON = 1
LD_ON = 0
LD_ON_reg = 0
I_set = 2048
I_incr = 2 (I_decr : 4)
State : S2 (010)

10) keep I_out below 10 and stay in switch on



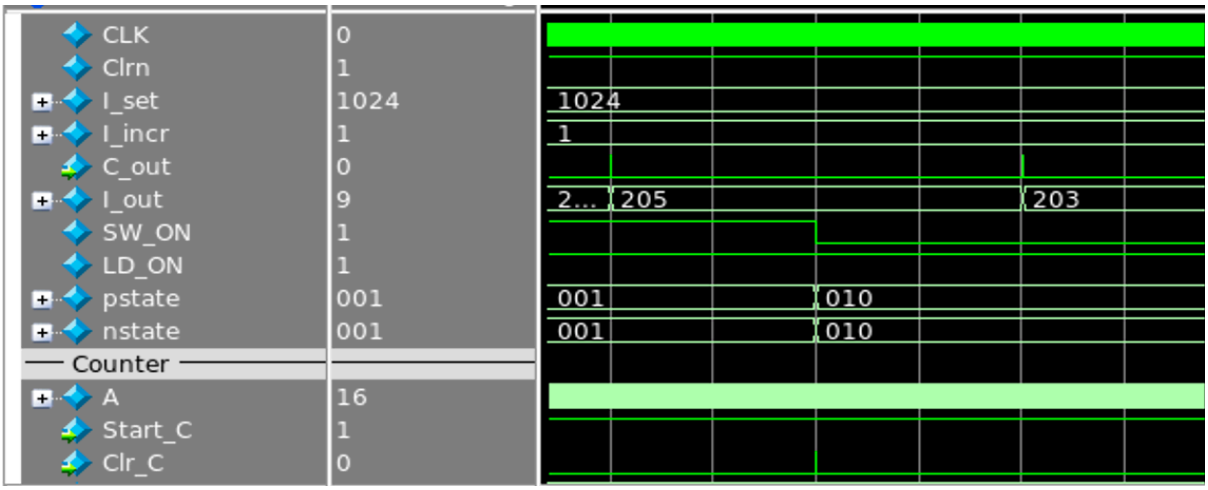
SW_ON = 1
LD_ON = 0
LD_ON_reg = 0
I_set = 2048
I_incr = 2 (I_decr : 4)
State : S2 (010)

11) change LD_ON to 1



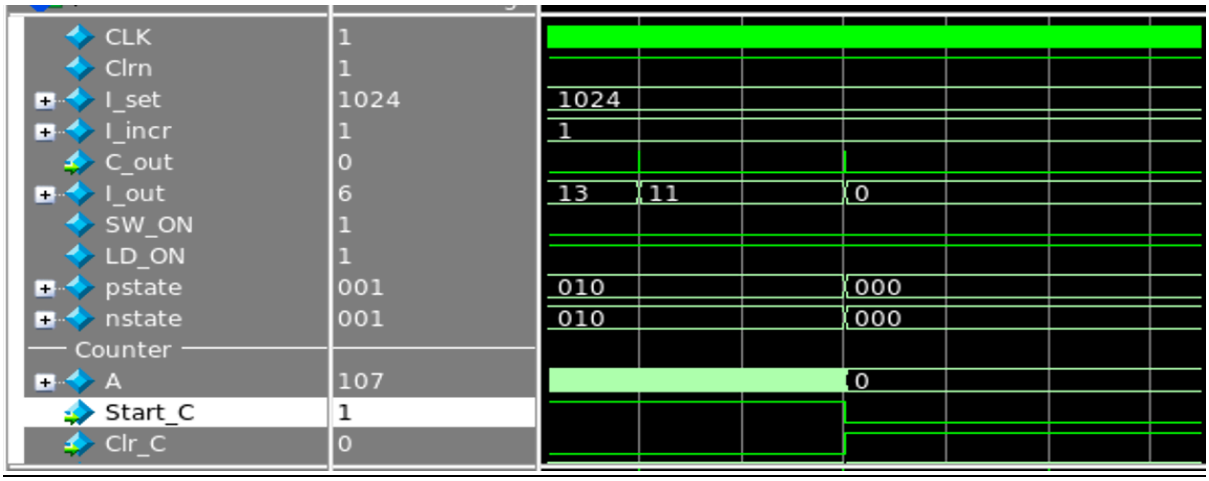
SW_ON = 1
LD_ON = 1
LD_ON_reg = 1
I_set = 1024
I_incr = 1 (I_decr : 2)
State : S1 (001)

12) Switch off



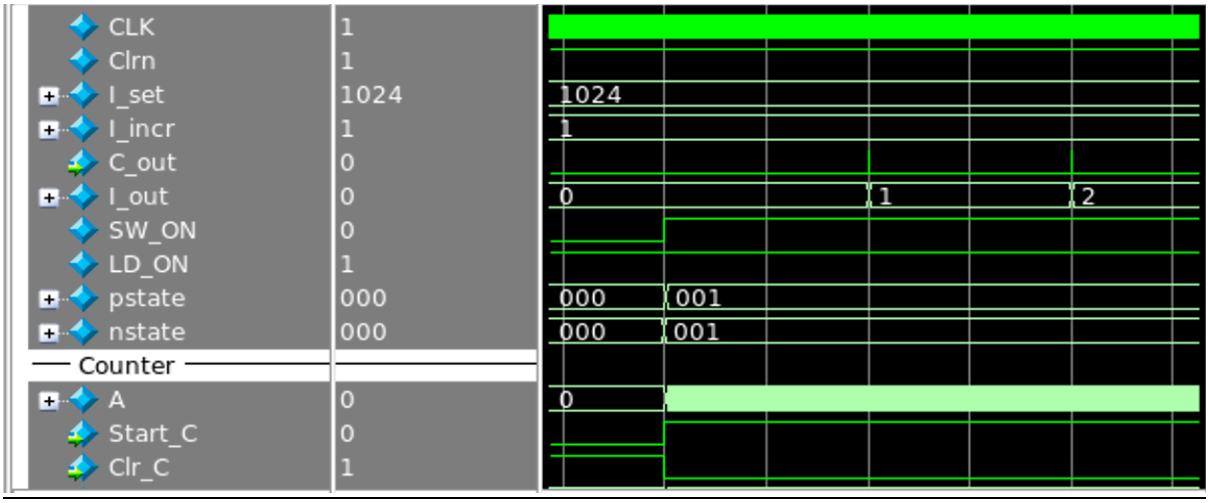
SW_ON = 0
LD_ON = 1
LD_ON_reg = 0
I_set = 1024
I_incr = 1 (I_decr : 2)
State : S2 (010)

13) When I_out reaches below 10, initialize I_out to 0 and go to initial state



SW_ON = 0
LD_ON = 1
LD_ON_reg = 0
I_set = 1024
I_incr = 1 (I_decr : 2)
State : S0 (000)

14) Switch on again



SW_ON = 1
LD_ON = 1
LD_ON_reg = 1
I_set = 1024
I_incr = 1
State : S1 (001)