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

Design LD Driver

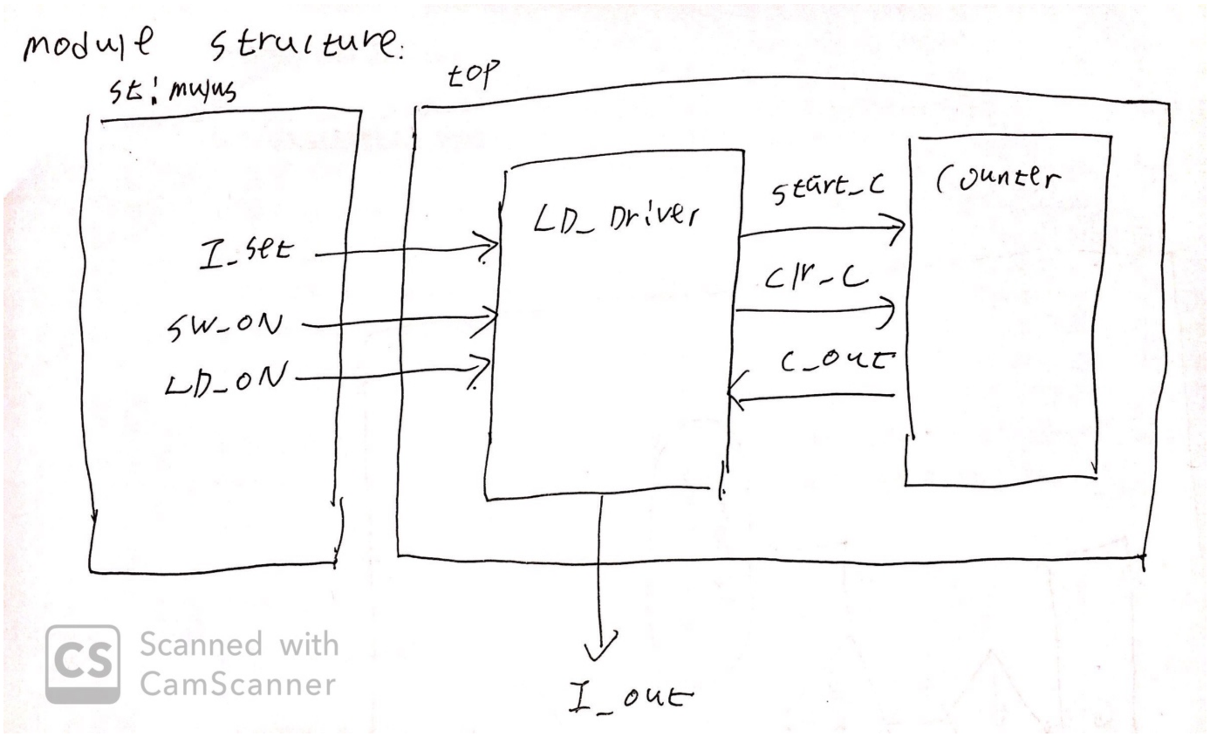
Design LD Driver with

1 12-bits binary counter I_Out (Predefined value of I_out is 2000)
1 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)

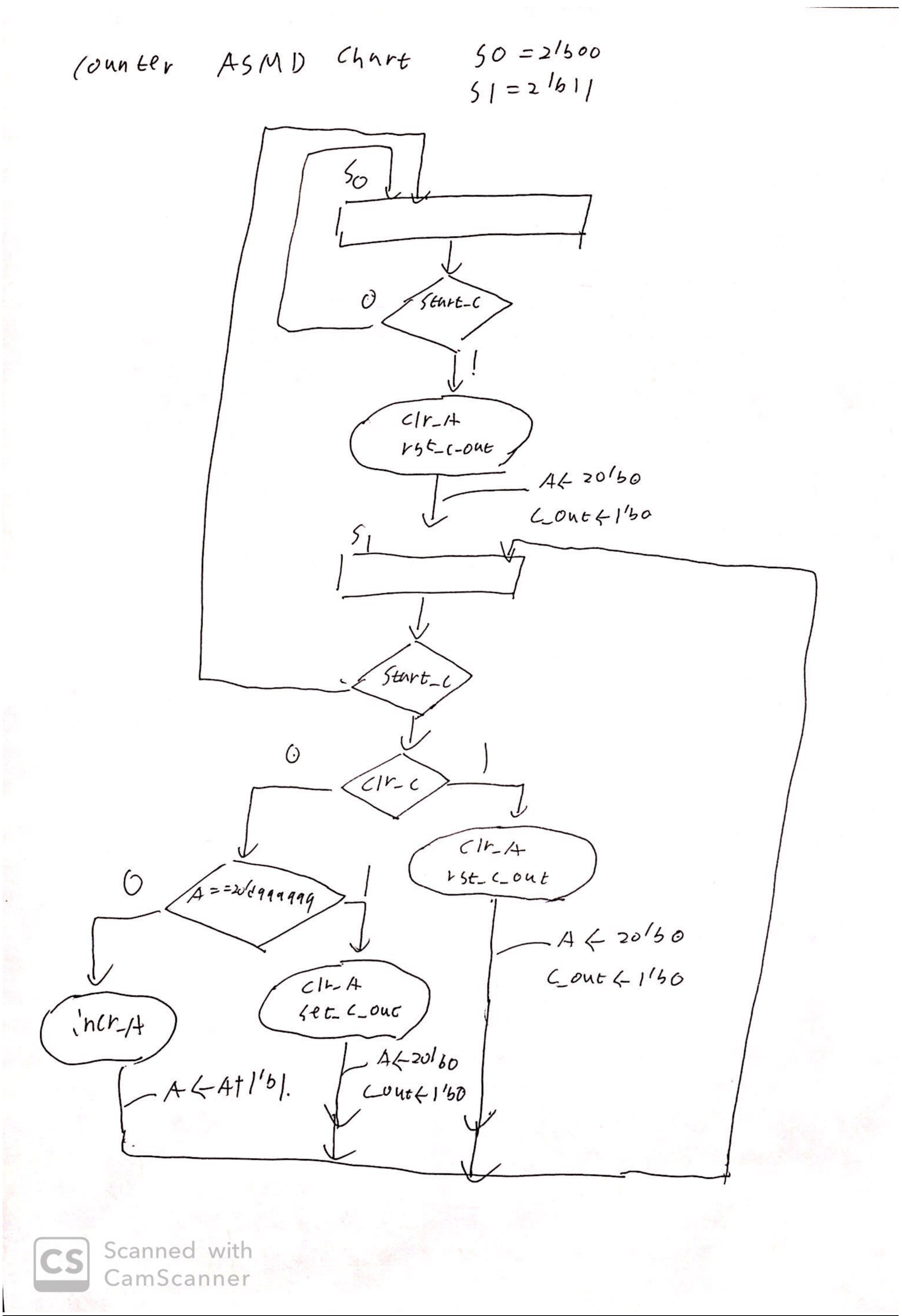
States

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

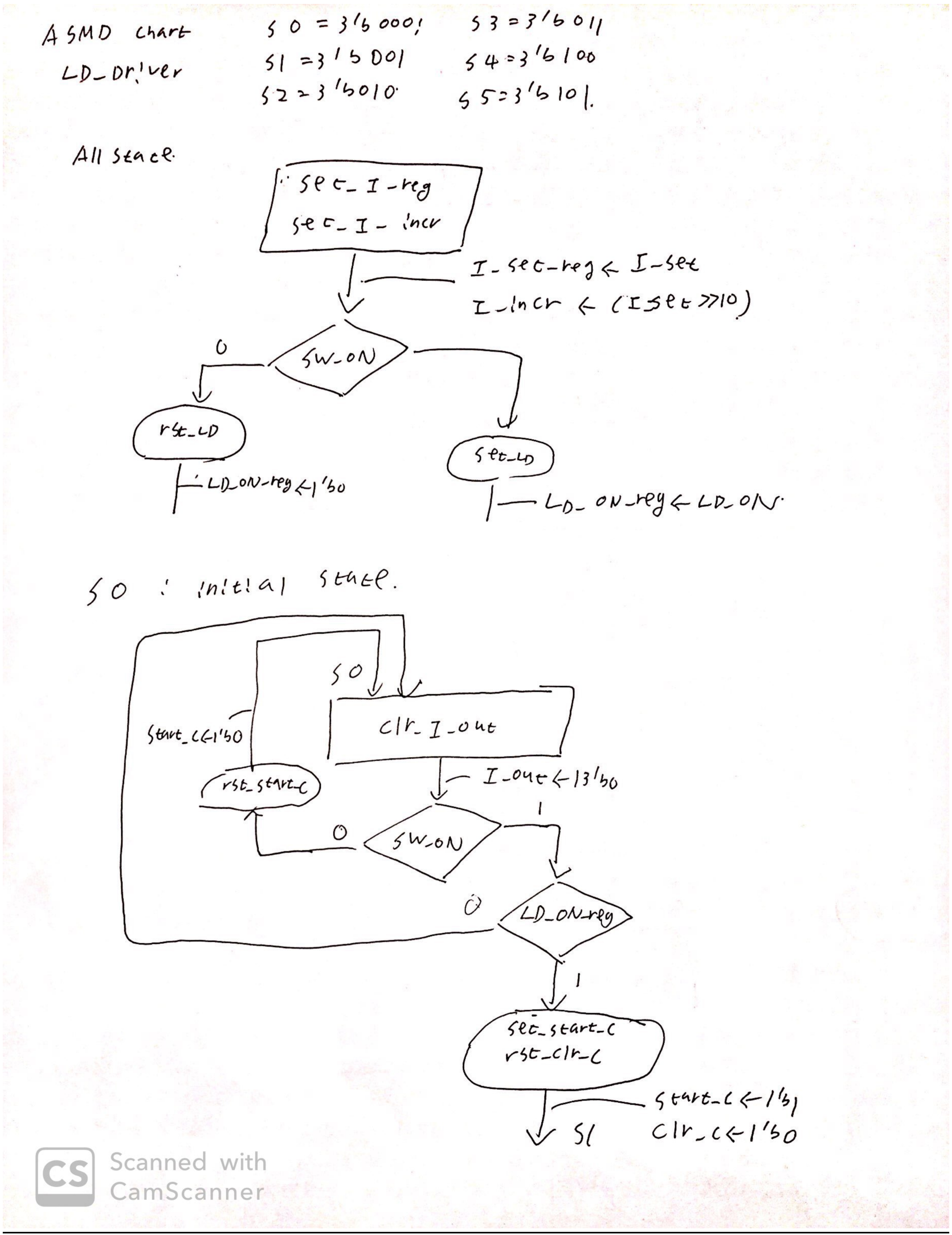
Module Structure



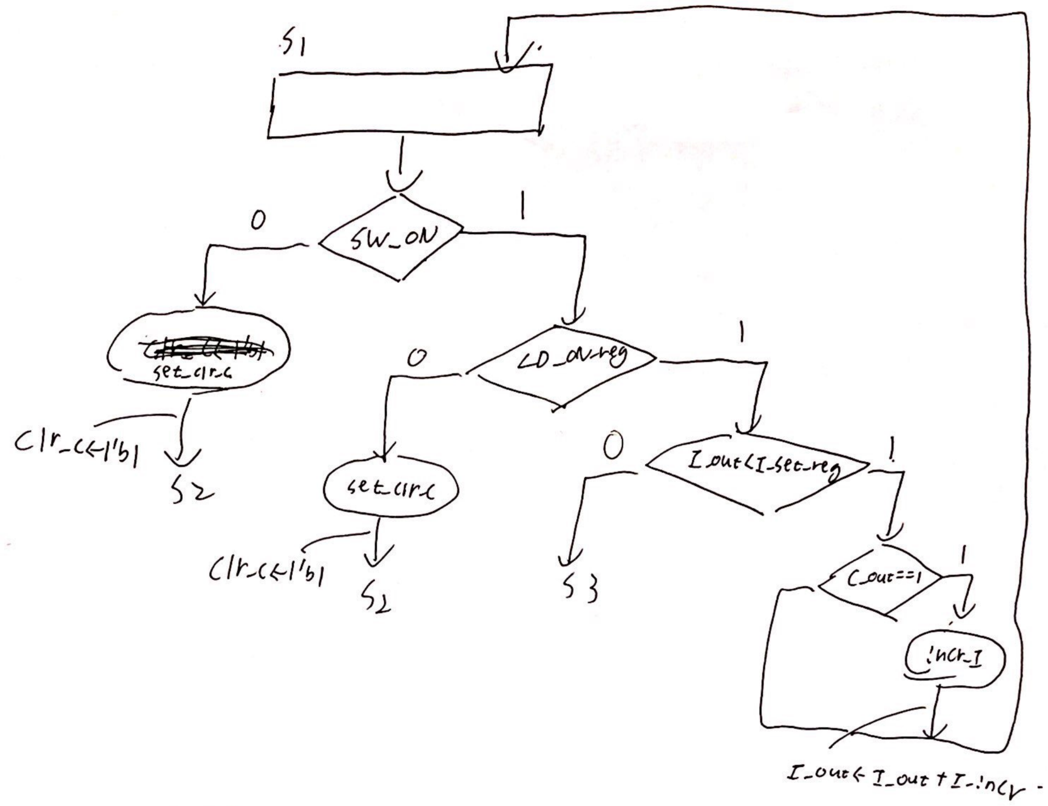
ASMD Chart of Counter_1E6



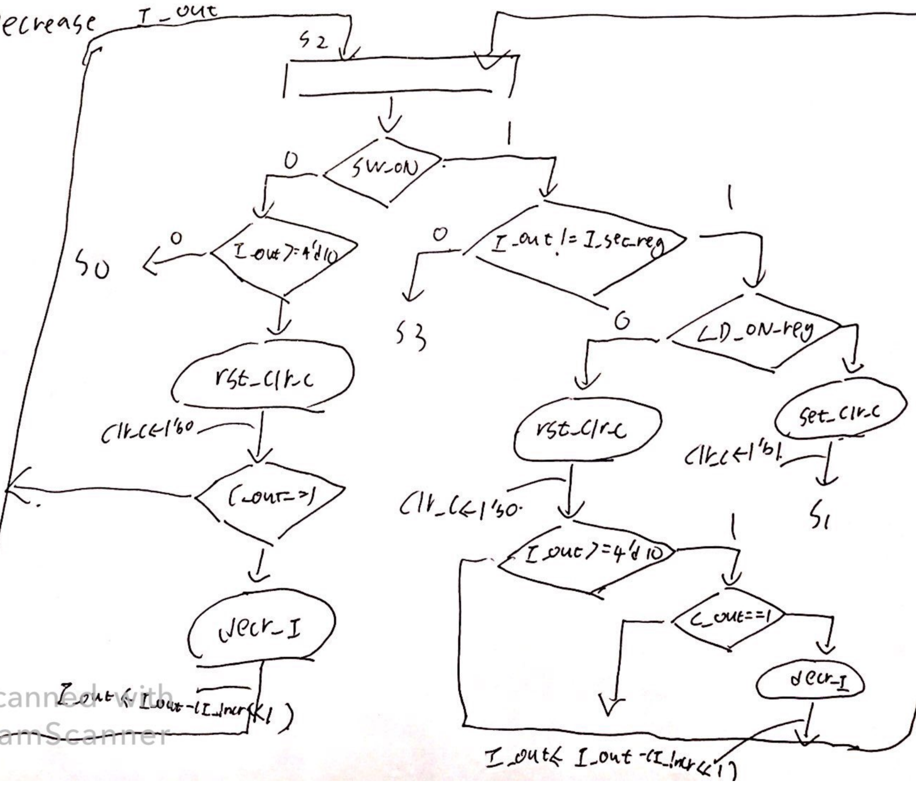
ASMD Chart of LD_Driver



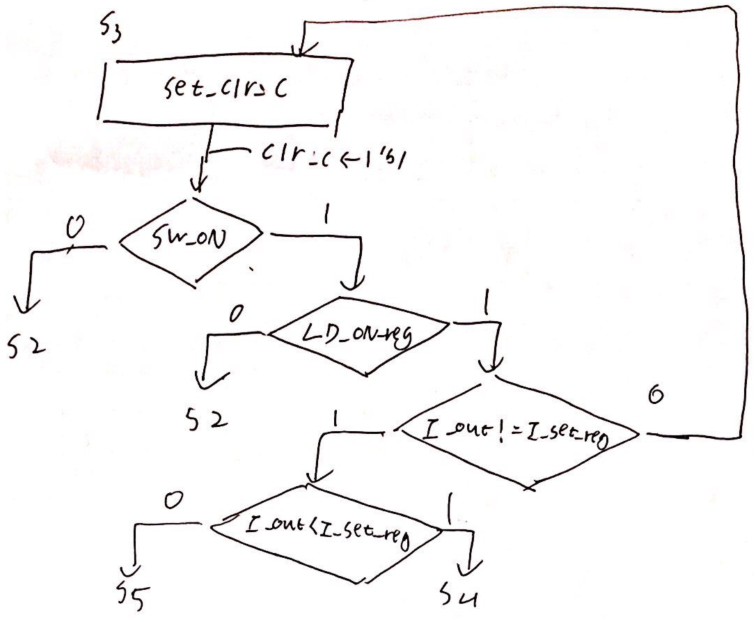
S1 Increase I_out



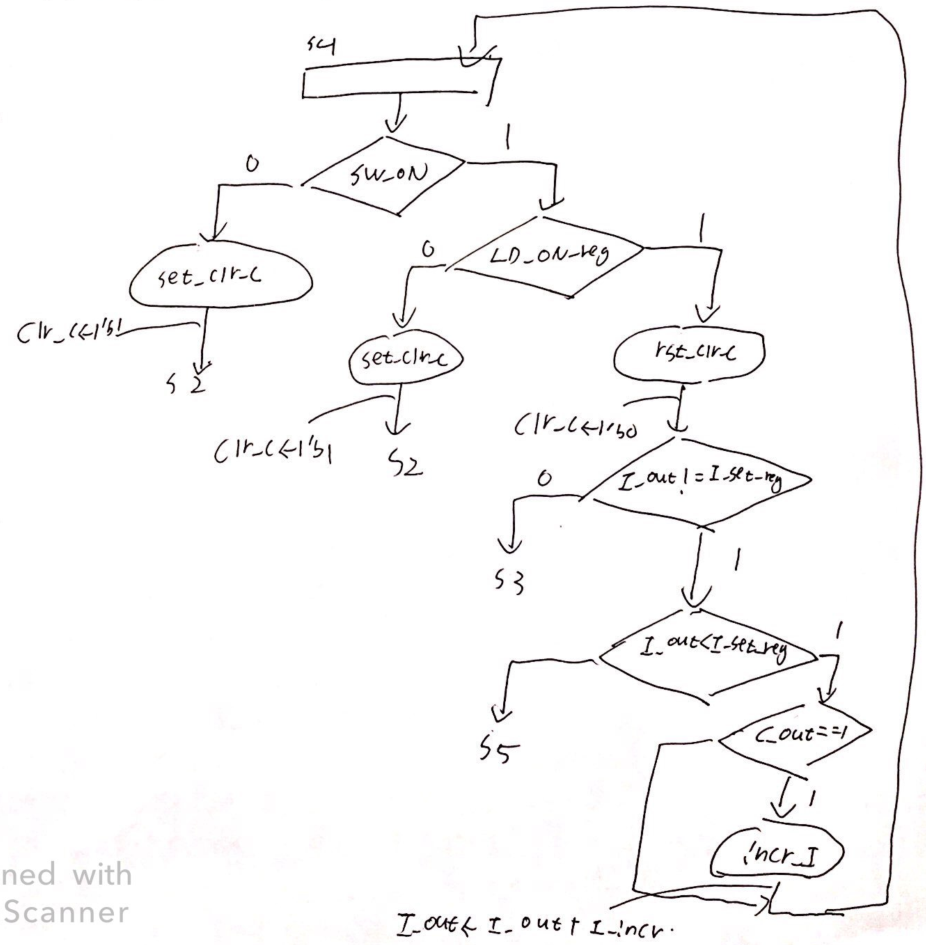
S2 Decrease I_out



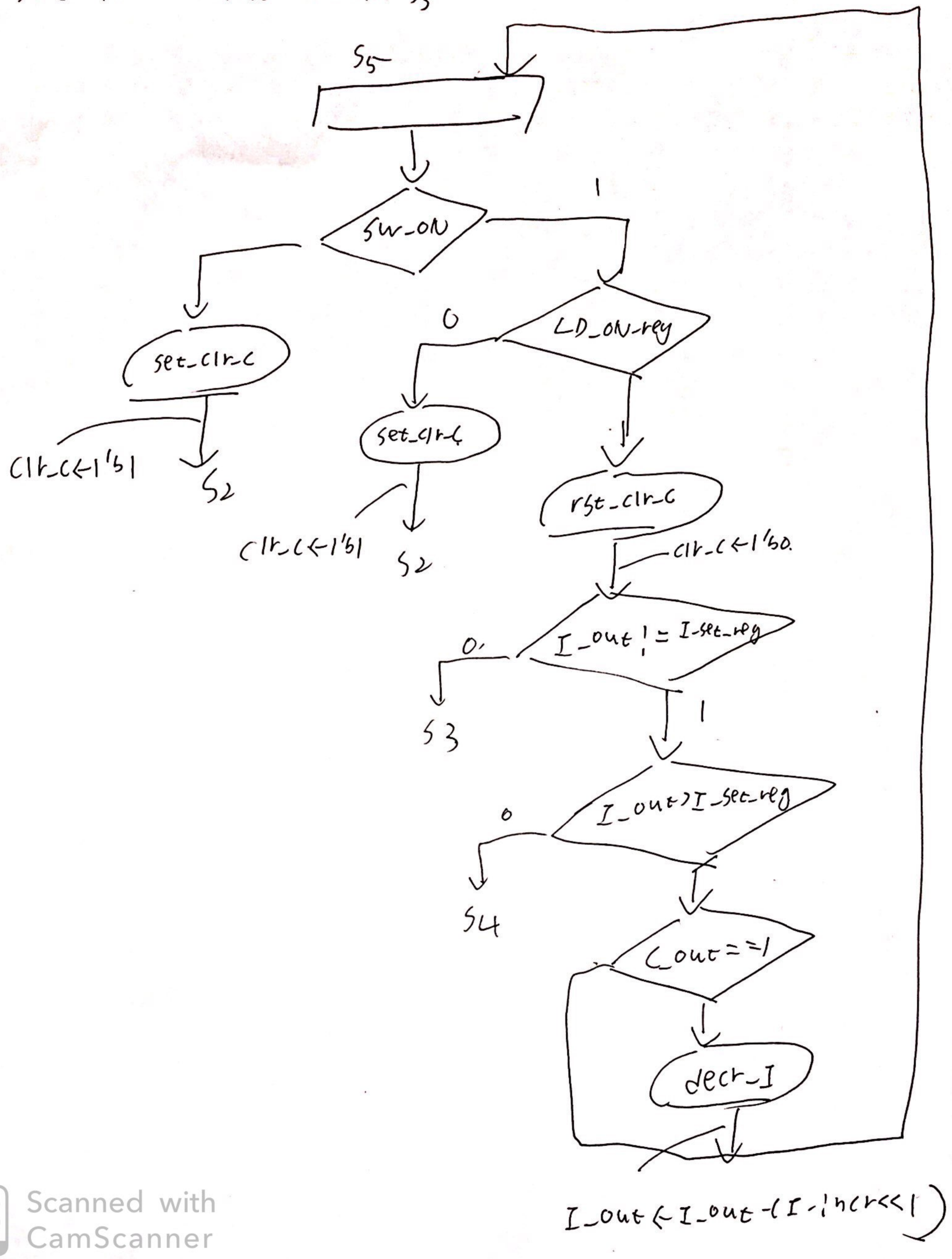
S3 keep I_out



S4 set high value from S3



S5 set low value from S3



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_ASMD_v_jin.v

```
module Counter_1E6_ASMD_v_jin(
   output reg C_out,
   input CLK, Clrn
);
```

```
reg [19:0] A; //Counter_1E6
reg [1:0] pstate, nstate; // state
reg clr_A, incr_A, rst_C_out, set_C_out;

//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) pstate <= S0;
 else pstate <= nstate; //clocked operation
end

// code next state logic
always@(pstate, A) begin
 case(pstate)
   S0:
     nstate <= S1;
   S1:
     if(A==20'd999999) nstate <= S0;
     else nstate <= S1;
 endcase
end

//code output logic
always@(pstate, A) begin
 clr_A <= 1'b0;
 incr_A <= 1'b0;
 rst_C_out <= 1'b0;
 set_C_out <= 1'b0;
 case(pstate)
   S0:
     clr_A <= 1'b1;
     rst_C_out <= 1'b1;
   S1:
     if(A==20'd999999) begin
```

```
         clr_A <= 1'b1;
         set_C_out <= 1'b1;
       end
       else begin
         incr_A <=1'b1;
         rst_C_out <= 1'b1;
       end
   endcase
  end

  //Register operation
  always @ (posedge CLK, negedge Clrn ) begin
   if(~Clrn) begin
     A <= 20'b0;
     clr_A <= 1'b0;
     incr_A <= 1'b0;
     rst_C_out <= 1'b0;
     set_C_out <= 1'b0;
   end
   else begin
     if(clr_A) A <= 20'b0;
     if(incr_A) A <= A+ 1'b1;
     if(rst_C_out) C_out <= 1'b0;
     if(set_C_out) C_out <= 1'b1;
   end
  end
endmodule
```

LD_Driver_ASMD_v_jin.v

```
module LD_Driver_ASMD_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;
  reg set_I_reg, set_I_incr, rst_LD, set_LD, clr_I_out, rst_Start_C, set_Start_C,
  rst_Clr_C, set_Clr_C, incr_I, decr_I;
  //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) pstate <= S0;
     else pstate <= nstate; //clocked operation
  end

  // code next state logic
  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

always @(SW_ON, LD_ON_reg, pstate, I_out, I_set_reg) begin

    // default assignment
    rst_LD <= 1'b0;
    set_LD <= 1'b0;
    clr_I_out <= 1'b0;
    rst_Start_C <= 1'b0;
    set_Start_C <= 1'b0;
    rst_Clr_C <= 1'b0;
    set_Clr_C <= 1'b0;
    incr_I <= 1'b0;
    decr_I <= 1'b0;

    // if SW_ON == 0, set LD_ON_reg 1'b0
```

```
if (SW_ON) set_LD <= 1'b1;
else rst_LD <= 1'b1;
set_I_reg <= 1'b1; // store i_set
set_I_incr <= 1'b1; // divide by 1024(approx 1000)

case (pstate)
   S0:
   begin
      clr_I_out <= 1'b1;
      if(SW_ON) begin
         set_Start_C <= 1'b1;
         rst_Clr_C <= 1'b1;
      end
      else rst_Start_C <= 1'b1;
   end

   S1:
   begin
      if(SW_ON) begin
         if(LD_ON_reg) begin
            rst_Clr_C <= 1'b1;
            if(I_out < I_set_reg) begin
               if(C_out == 1'b1) begin
                  incr_I <= 1'b1;
               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
             rst_Clr_C <= 1'b1;
      if(I_out >= 4'b1010) begin
        if(C_out == 1'b1) begin
          //multiple by 2
          decr_I <= 1'b1;
        end
      end
    end
    else Clr_C <= 1'b1;
  end
  else begin
    if(I_out >= 4'b1010) begin
      rst_Clr_C <= 1'b1;
      if(C_out == 1'b1) begin
        decr_I <= 1'b1;
      end
    end
    else Clr_C <= 1'b1;
  end
end

S3:
  set_Clr_C <= 1'b1;

S4:
begin
  if(SW_ON) begin
    if(LD_ON_reg) begin
      rst_Clr_C <= 1'b1;
      if(I_out < I_set_reg) begin
        if(C_out == 1'b1) begin
          incr_I <= 1'b1;
        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
                 rst_Clr_C <= 1'b1;
                 if(I_out > I_set_reg) begin
                    if(C_out == 1'b1) begin
                       decr_I <= 1'b1;
                    end
                 end
              end
              else Clr_C <= 1'b1;
           end
           else Clr_C <= 1'b1;
        end
    endcase
end

//Register Transfer operation
always @ ( posedge CLK, negedge Clrn ) begin
    if(~Clrn) begin
       I_out <= 13'b0;
       LD_ON_reg <= 1'b0;
       I_set_reg <= 13'b0;
       I_incr <= 13'b0;
    end
    else begin
       if(set_I_reg) I_set_reg <= I_set; //store i value
       if(set_I_incr) I_incr <= (I_set >> 10);  // make incr
       if(rst_LD) LD_ON_reg <= 1'b0;
       if(set_LD) LD_ON_reg <= LD_ON;
       if(clr_I_out) I_out <= 13'b0;
       if(rst_Start_C) Start_C <= 1'b0;
       if(set_Start_C) Start_C <= 1'b1;
       if(rst_Clr_C) Clr_C <= 1'b0;
```

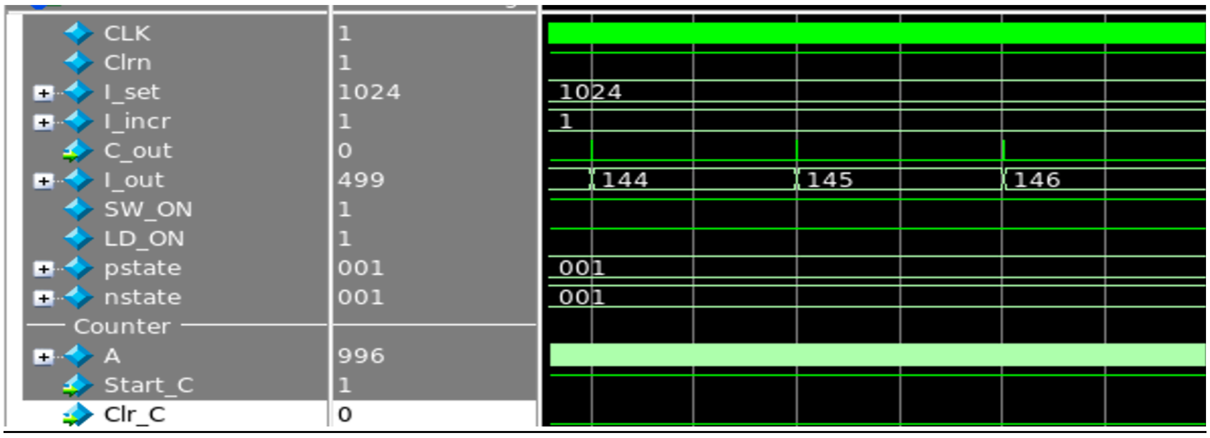
```
        if(set_Clr_C) Clr_C <= 1'b1;
        if(incr_I) I_out <= I_out + I_incr;
        if(decr_I) I_out <= I_out - (I_incr << 1);
      end
  end
endmodule
```

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



```
CLK        1
Clrn       1
I_set      1024
I_incr     1
C_out      0
I_out      499
SW_ON      1
LD_ON      1
pstate     001
nstate     001
Counter
A          996
Start_C    1
Clr_C      0
```

SW_ON = 1

LD_ON = 1

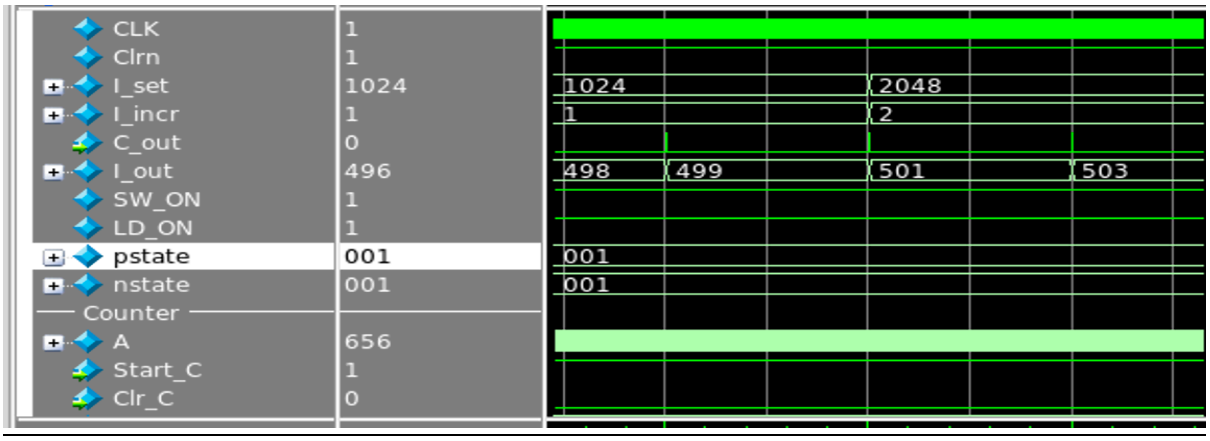
LD_ON_reg = 1

I_set = 1024

I_incr = 1

State : S1 (001)

2) change I_set during increasing I_out



```
CLK        1
Clrn       1
I_set      1024
I_incr     1
C_out      0
I_out      496
SW_ON      1
LD_ON      1
pstate     001
nstate     001
Counter
A          656
Start_C    1
Clr_C      0
```

SW_ON = 1

LD_ON = 1

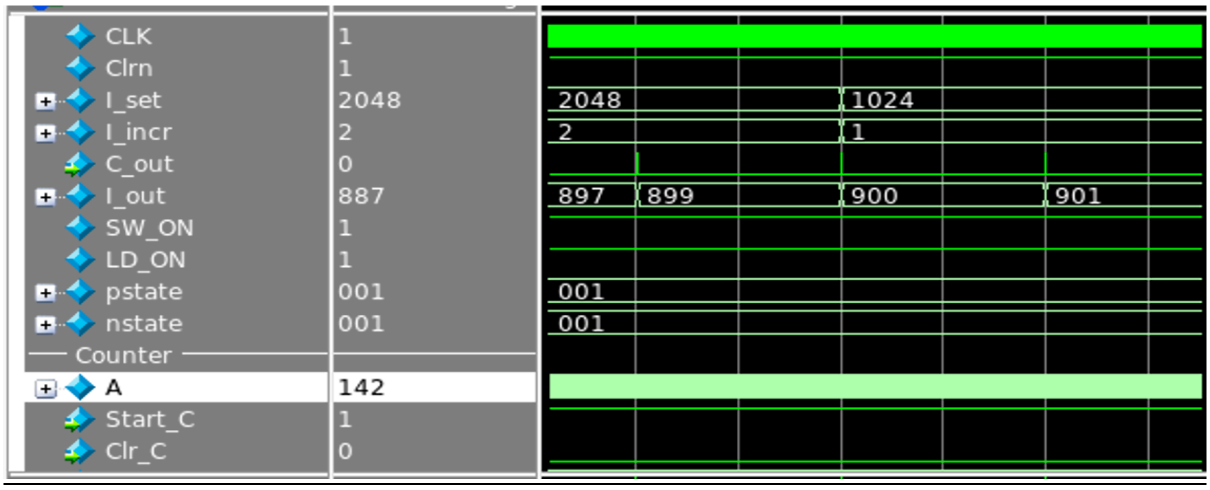
LD_ON_reg = 1

I_set = 2048

I_incr = 2

State : S1 (001)

3) change again I_set during increasing I_out



SW_ON = 1

LD_ON = 1

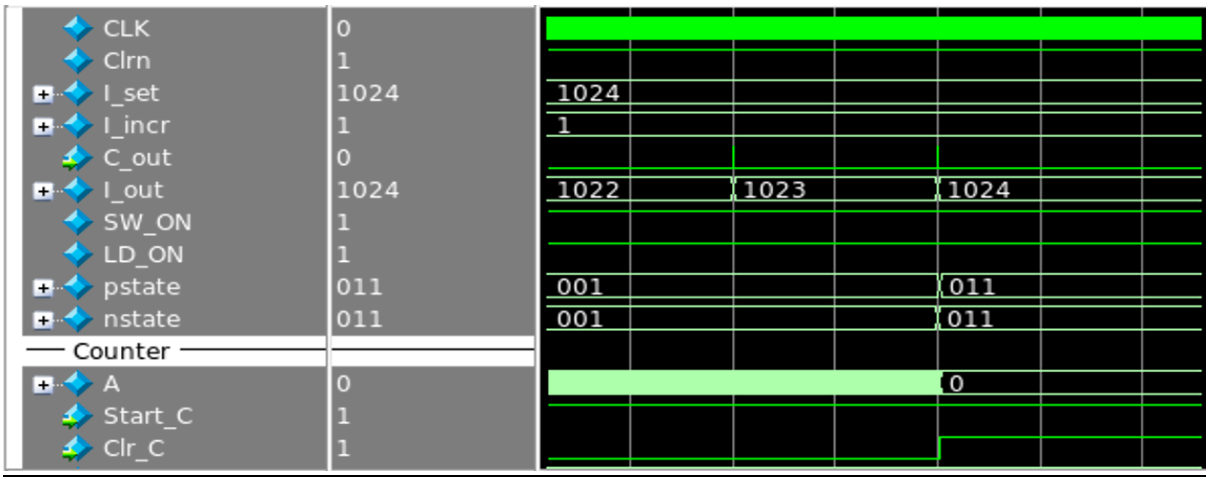
LD_ON_reg = 1

I_set = 1024

I_incr = 1

State : S1 (001)

4) reach I_set value



SW_ON = 1

LD_ON = 1

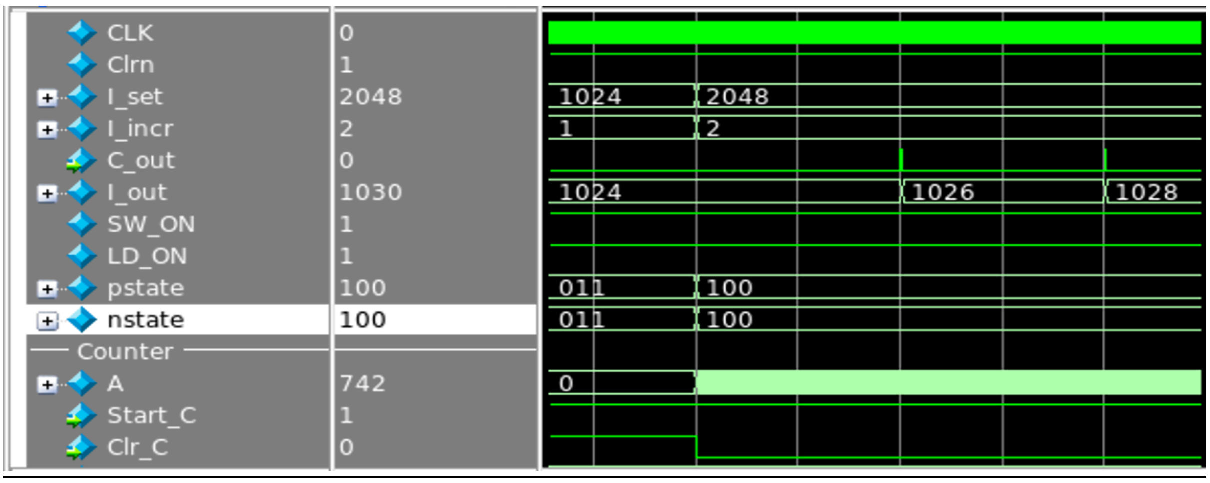
LD_ON_reg = 1

I_set = 1024

I_incr = 1

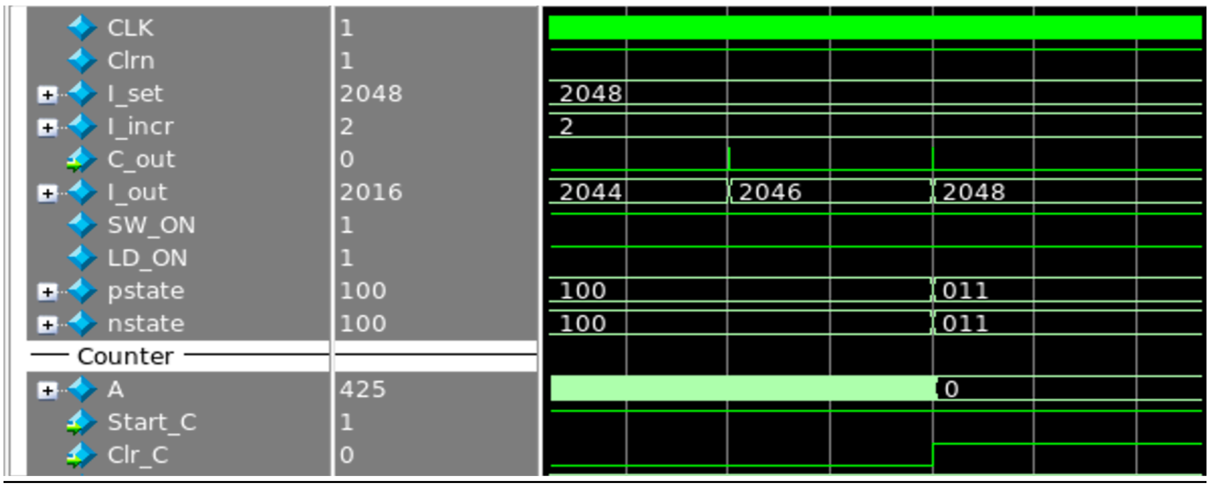
State : S3 (011)

5) change High value of I_set when keep I_out



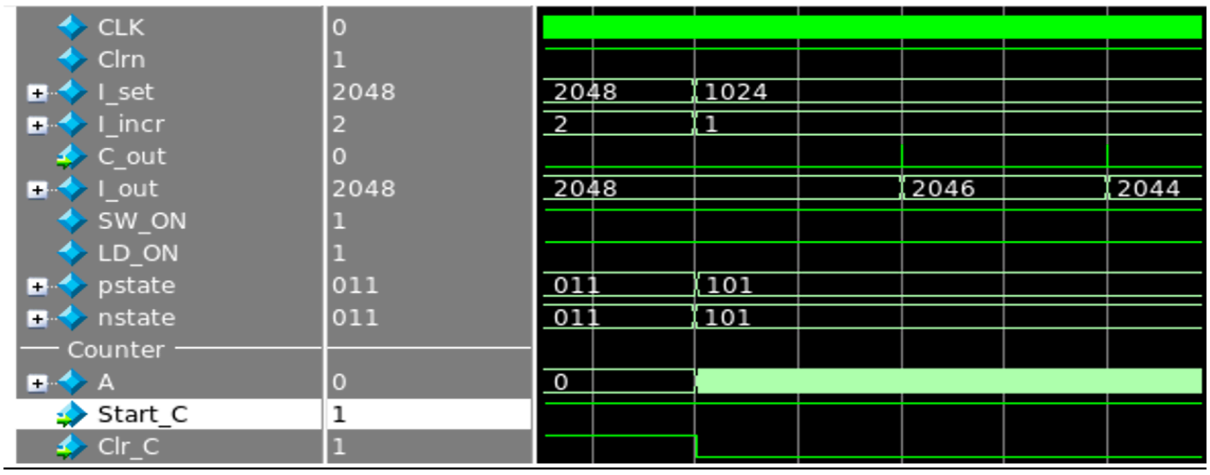
SW_ON = 1
LD_ON = 1
LD_ON_reg = 1
I_set = 2048
I_incr = 2
State : S4 (100)

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

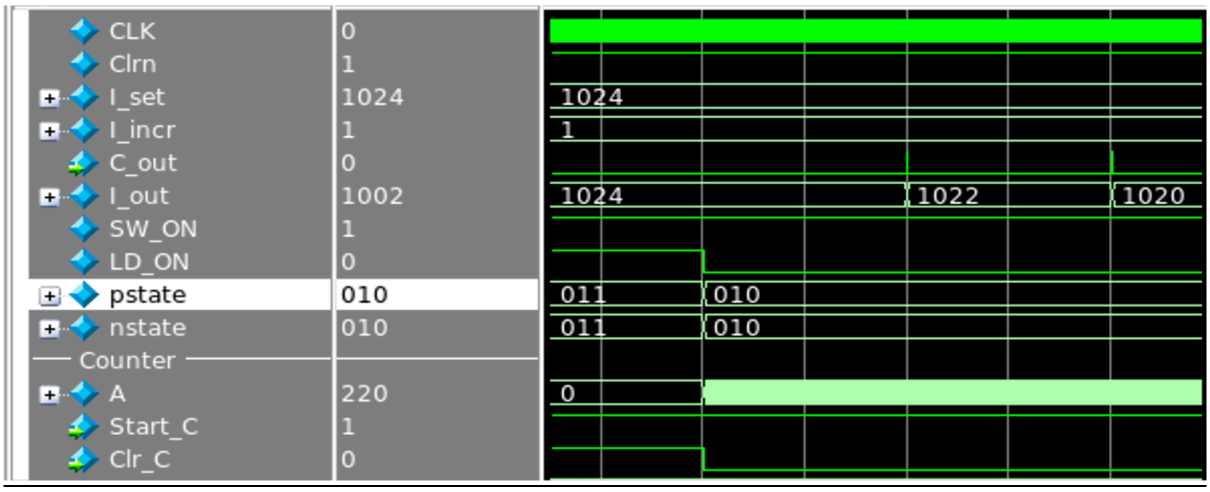
LD_ON_reg = 1

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

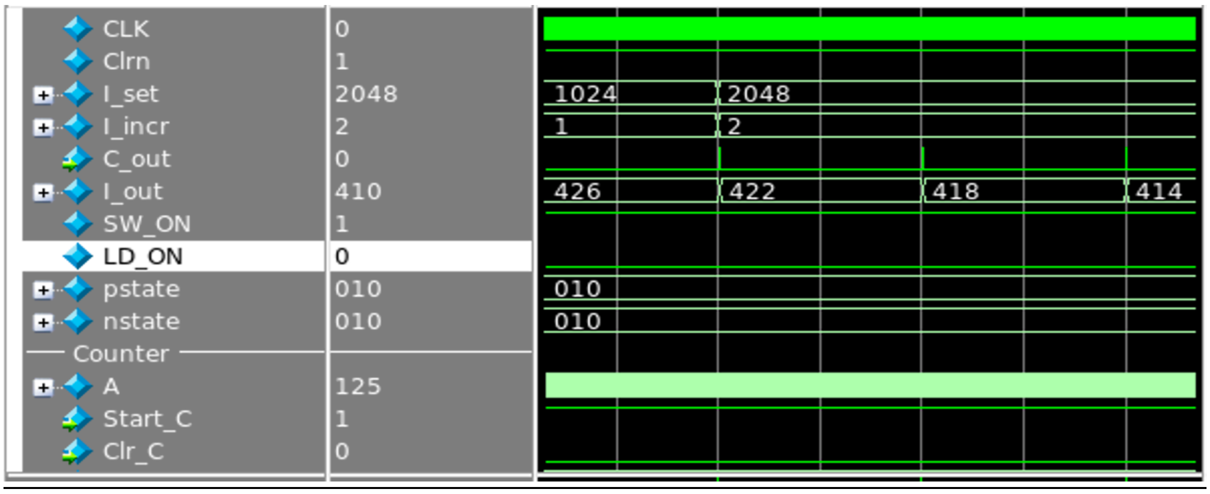
LD_ON_reg = 0

I_set = 1024

I_incr = 1 (I_decr : 2)

State : S2 (010)

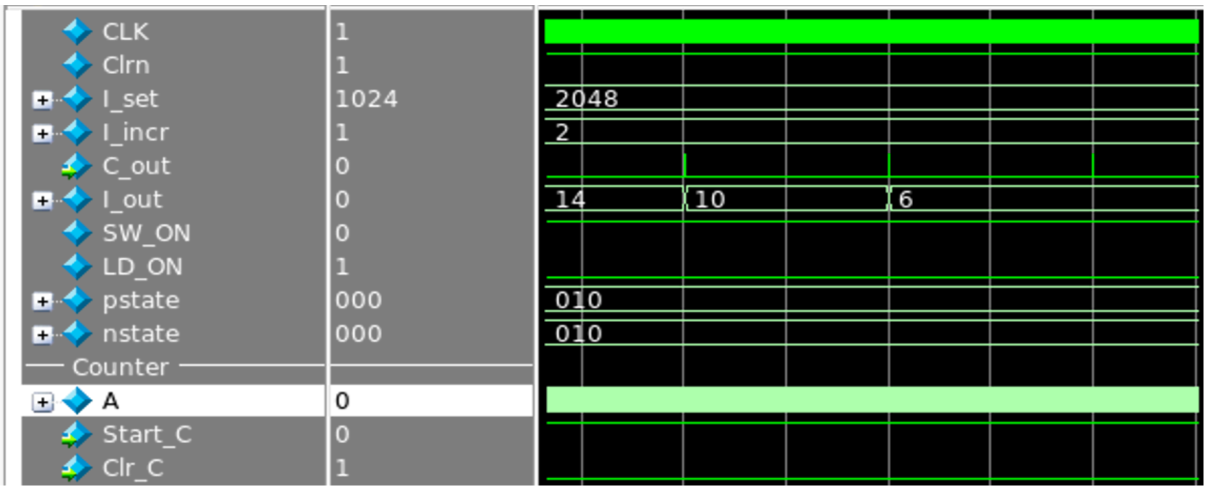
9) change I_set during decreasing I_out



CLK	0	
Clrn	1	
I_set	2048	1024 2048
I_incr	2	1 2
C_out	0	
I_out	410	426 422 418 414
SW_ON	1	
LD_ON	0	
pstate	010	010
nstate	010	010
Counter		
A	125	
Start_C	1	
Clr_C	0	

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

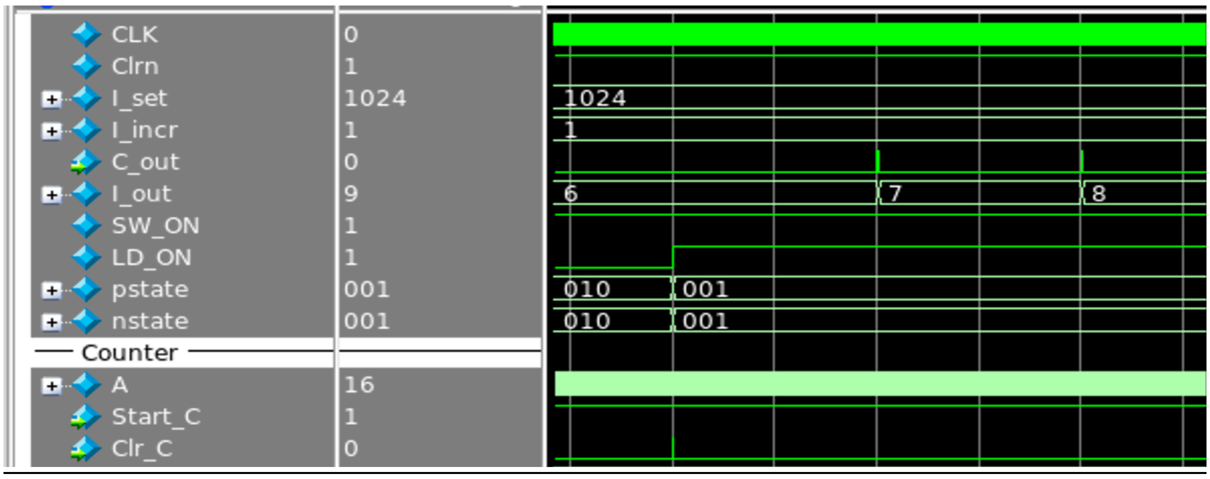
10) keep I_out below 10 and stay in switch on



CLK	1	
Clrn	1	
I_set	1024	2048
I_incr	1	2
C_out	0	
I_out	0	14 10 6
SW_ON	0	
LD_ON	1	
pstate	000	010
nstate	000	010
Counter		
A	0	
Start_C	0	
Clr_C	1	

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

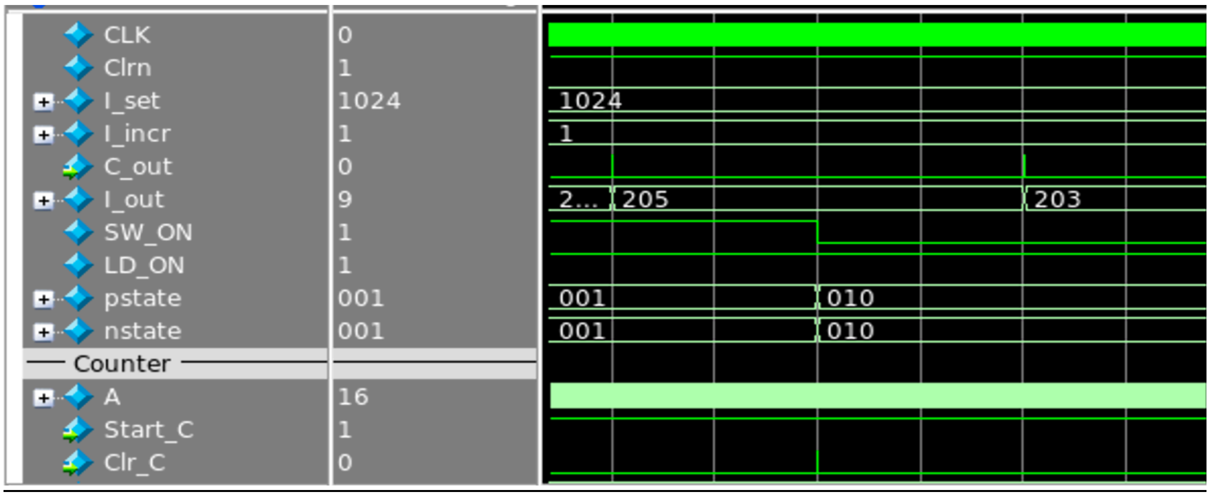
11) change LD_ON to 1



CLK	0
Clrn	1
I_set	1024
I_incr	1
C_out	0
I_out	9
SW_ON	1
LD_ON	1
pstate	001
nstate	001
Counter	
A	16
Start_C	1
Clr_C	0

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

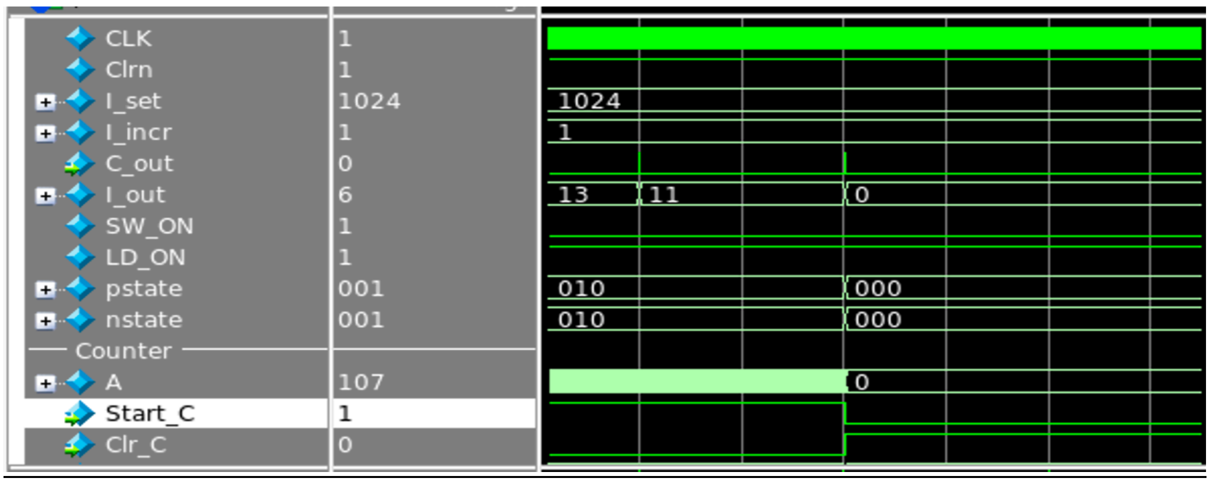
12) Switch off



CLK	0
Clrn	1
I_set	1024
I_incr	1
C_out	0
I_out	9
SW_ON	1
LD_ON	1
pstate	001
nstate	001
Counter	
A	16
Start_C	1
Clr_C	0

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

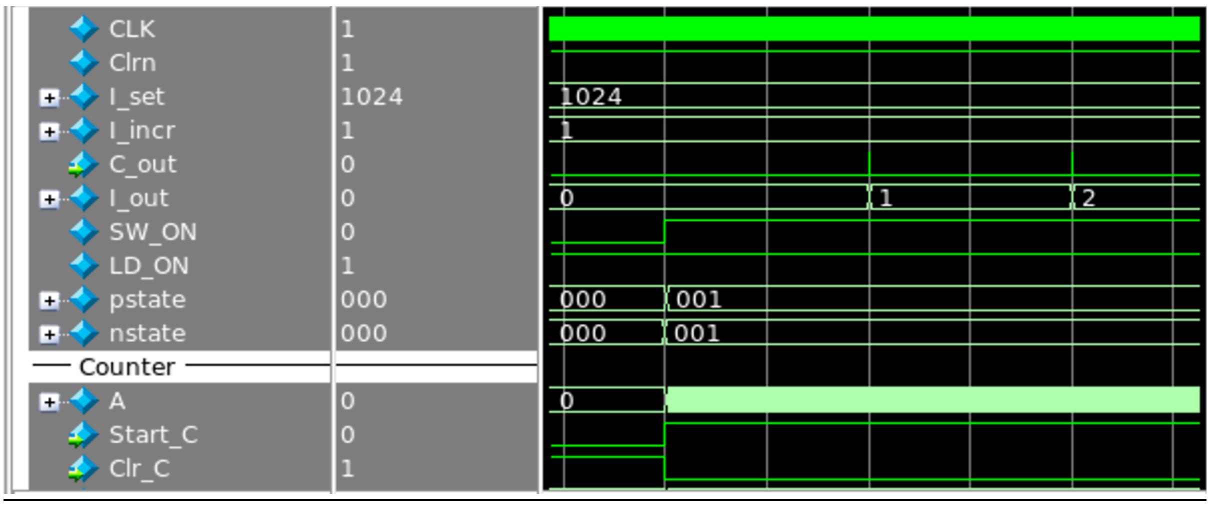
13) When I_out reaches below 10, initialize I_out to 0 and go to initial state



CLK	1
Clrn	1
I_set	1024
I_incr	1
C_out	0
I_out	6
SW_ON	1
LD_ON	1
pstate	001
nstate	001
Counter	
A	107
Start_C	1
Clr_C	0

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



CLK	1
Clrn	1
I_set	1024
I_incr	1
C_out	0
I_out	0
SW_ON	0
LD_ON	1
pstate	000
nstate	000
Counter	
A	0
Start_C	0
Clr_C	1

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

자기 평가:
수업시간에 배운 ASM, ASMD 차트로 실제 생활에 이용할 법한 것을 제작 및 설계를 해보니 코딩 실력 뿐만 아니라 어떻게 설계 해야하는지에 대해 많이 배운 것 같다. 이전에는 디지털 시계를 만드는 간단한 것만 설계할 수 있었는데 지금은 원하는 디자인 어떠한 것을 요구해도 수업에 배운 Design Process 를 따라가면 무엇이든지 만들 수 있을 것 같다는 자신감이 든다. 또한 쿼터스라는 툴을 직접 이용해보고 여러가지 결과를 확인하며 코드를 디버깅하는 능력도 향상된 것 같아서 매우 좋은 수업이었고, 수업을 떠나 이후 베릴로그 코딩 실력이 매우 높아진 것 같다.