HW#12 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
2 External input(main input) SW_ON, LD_ON
1 Flip-Flop LD_ON_reg
1 Master Clock 100MHZ (period = 10ns)

Since we can see the Counter_1E6 module is included in the list, we need to design Counter_1E6 First. Counter_1E6 acting like pulse generator go up to High Value at every 10ms

ASM approach
Design Counter_1E6 with

1 output reg C_out (1 when counter A count 1000000)
1 20-bits binary counter A
2 External input(main input) Start
1 Master Clock 100MHZ (period = 10ns)

Operation

Since, Clrn is asynchronous reset
If Clrn = 0 C_out ← 0, A ← 0 and go back or remain initial state

if Start = 1, initiate counter operation by clearing A, C_out
    Value of A determine the operation sequence
    If A=20'd999999 C_out ← 1 and clear A
    Else A counts up until the A is 20'd99999 C_out ← 0

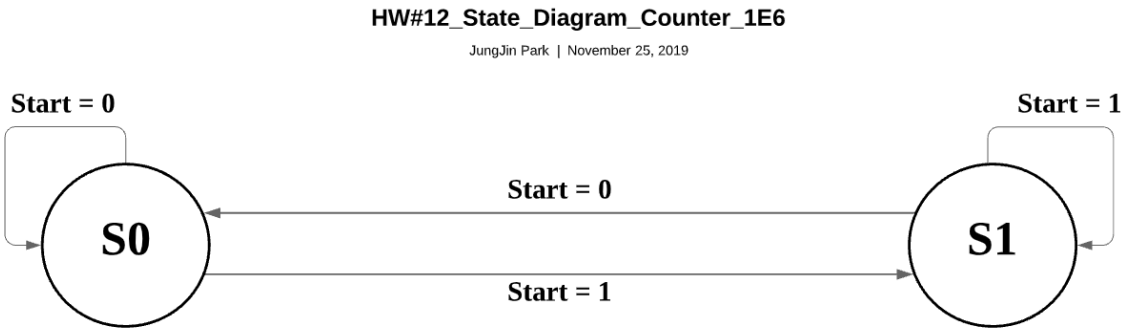
Else stop counting and go back to initial state

if Start = 0, the system remains in the initial state.

States

initial state : S_0
count : S_1

State diagram

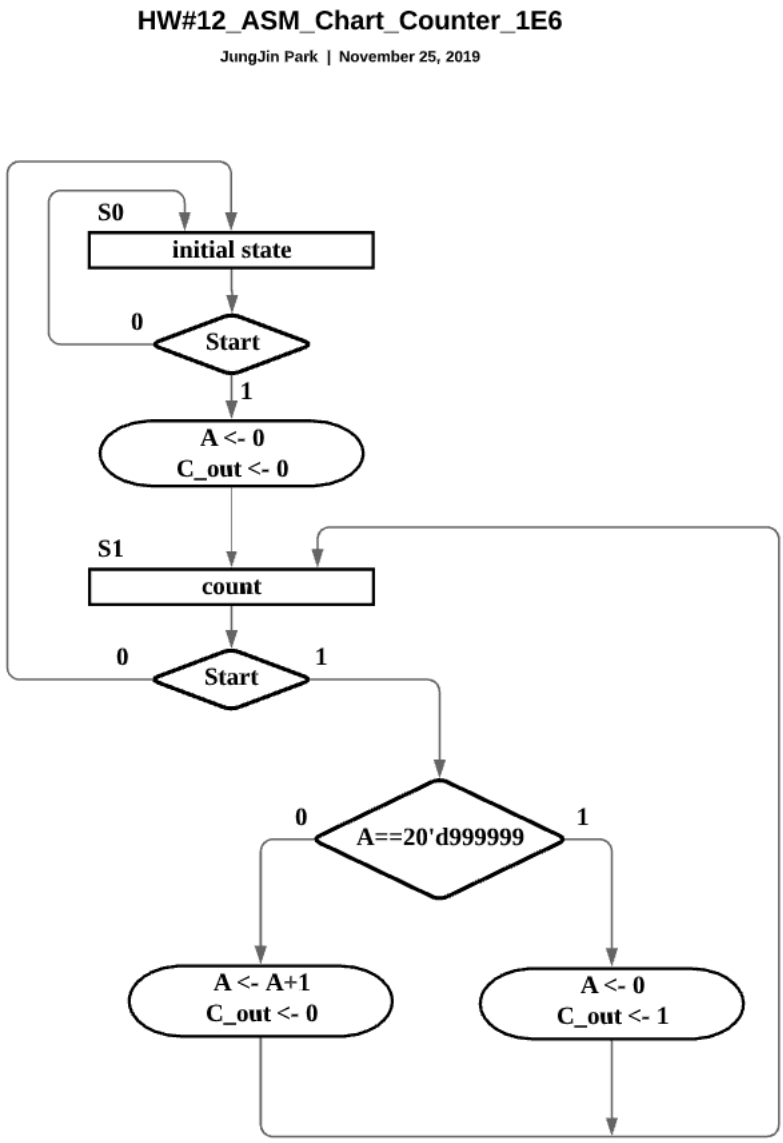


Register Transfer Operation

S0 : if (Start) then A ← 0, C_out ← 0

S1 :
if (Start)
    if (A = 20'd99999) then A ← 0, C_out ← 1
    else then A ← A+ 1, C_out ← 0

ASM Chart



Verilog

Source

```
module Counter_1E6_ASM_Test_v_jin(
   output reg C_out,
   input Start,
   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
		C_out <= 1'b0;
		A <= 20'b0;
		pstate <= S0;
	end
	else pstate <= nstate; //clocked operation
end

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

//Reigster Transfer operations
always @(posedge CLK) begin
	case(pstate)
		S0:
		begin
			if(Start) begin
			  A <= 20'b0;
			  C_out <= 1'b0;
			end
		end
```

```
                S1:
                begin
                        if(Start) 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
        endcase
  end
endmodule
```

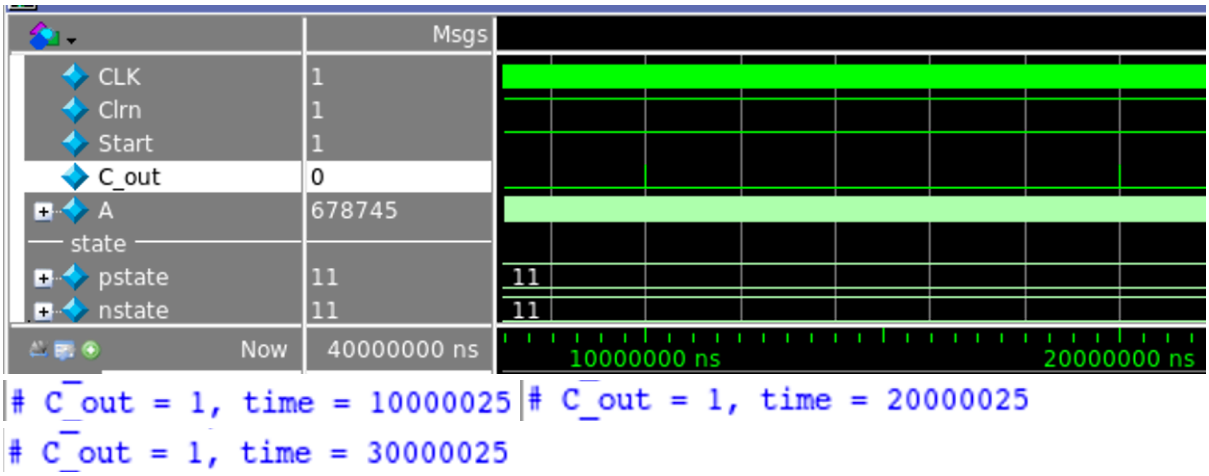
testbench

```
`timescale 1ns/1ns
module Counter_1E6_ASM_Test_v_tb_jin;

wire C_out;
reg Start;
reg CLK, Clrn;

initial begin
  #50E6 $finish;
end

initial
begin
  CLK <= 1'b0;
  Clrn <= 1'b0;
  Start <= 1'b1;
  #20 Clrn <= 1'b1; // reset during two clock edge
  #35E6 Start <= 1'b0; //start off
  #2E6 Start <= 1'b1; //again start on
end
```

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

Counter_1E6_ASM_Test_v_jin ct1(
   .C_out(C_out),
   .Start(Start),
   .CLK(CLK),
   .Clrn(Clrn));

always @(C_out) begin
   $monitor("C_out = %d, time = %0d", C_out, $time);
end
endmodule
```

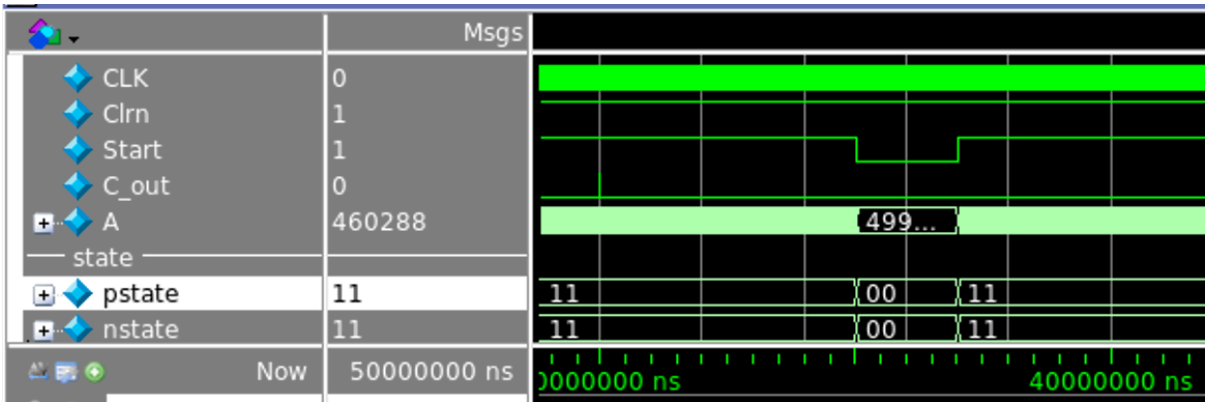
RTL Simulation

1) Count A



At 10000025ns = 10.000025ms , 20.000025ms, 30.000025ms
Start = 1
C_out = 1
State : S1 (11)

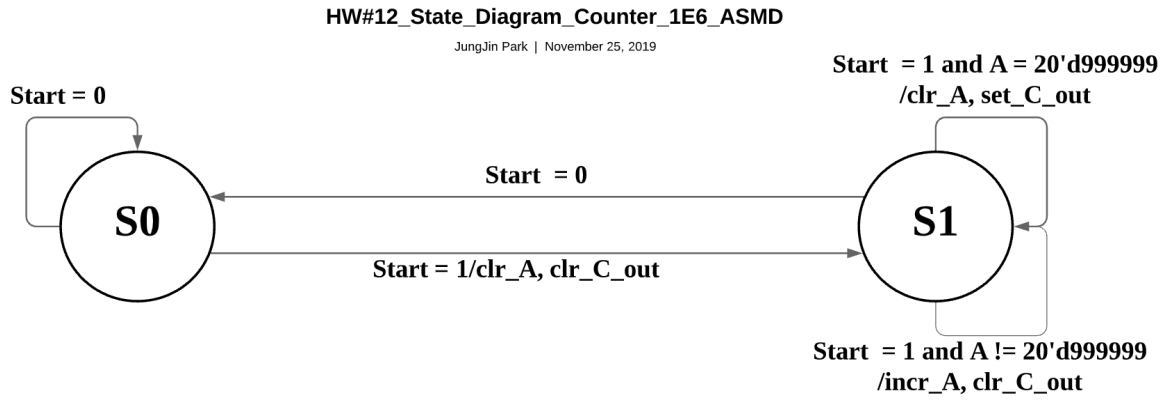
2) Start off



Start = 0
C_out = 0
State : S0 (00)

ASMD approach

State diagram

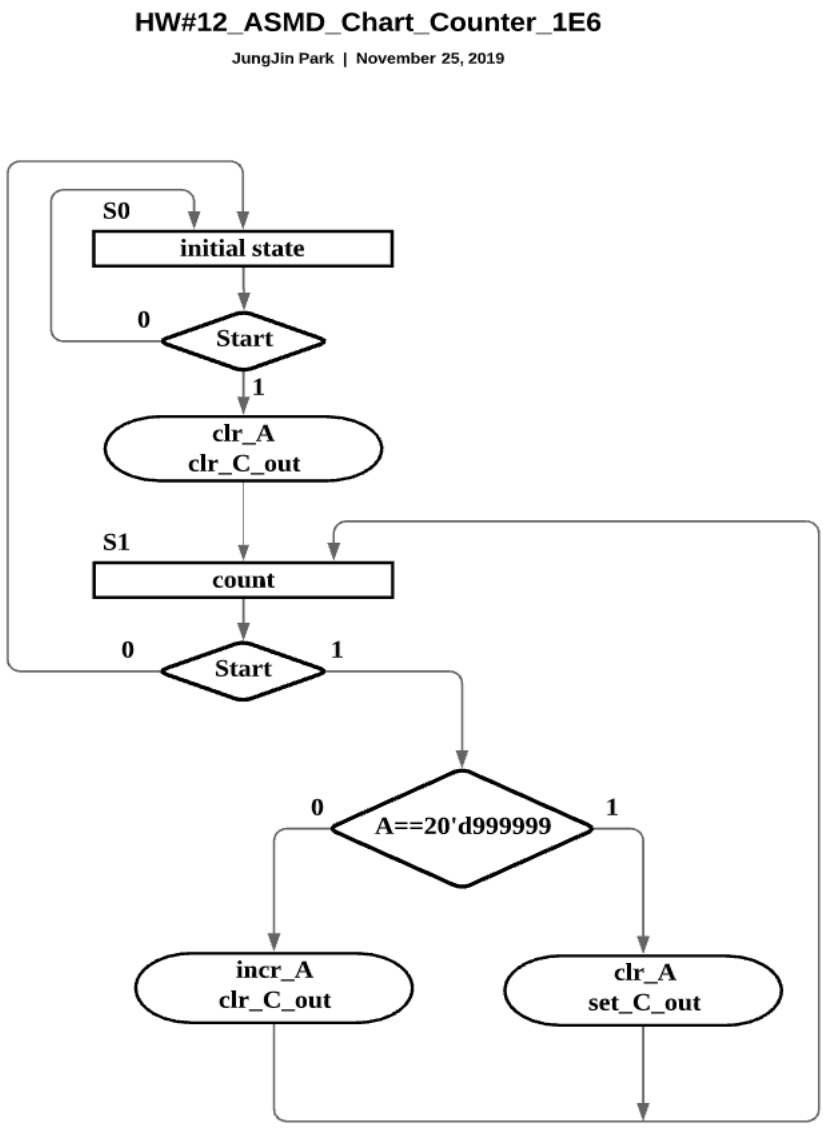


Register Transfer Operation

S0 :
if (Start) S0 → S1, clr_A , clr_C_out : A ← 0, C_out ← 0

S1 :
if (Start)
S1 → S1
if (A = 20'd99999) clr_A, set_C_out: A ← 0, C_out ← 1
else incr_A, clr_C_out: A ← A+ 1, C_out ← 0
else S1 → S0

ASM Chart



Verilog

Source

```
module Counter_1E6_ASMD_Test_v_jin(
   output reg C_out,
   input Start,
   input CLK, Clrn
);

   reg [19:0] A; //Counter_1E6
```

```
reg [1:0] pstate, nstate; // state
reg clr_A, incr_A, clr_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) begin
		C_out <= 1'b0;
		A <= 20'b0;
		pstate <= S0;
	end
	else begin
		pstate <= nstate; //clocked operation
		if(clr_A) A <= 20'b0;
		if(incr_A) A <= A +  1'b1;
		if(clr_C_out) C_out <= 1'b0;
		if(set_C_out) C_out <= 1'b1;
	end
end

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

//decide control
always @(pstate, A, Start) begin
	//default assignmet (recommend)
```

```
        clr_A <= 1'b0;
        incr_A <= 1'b0;
        clr_C_out <= 1'b0;
        set_C_out <= 1'b0;

        //Register transfer operation
        case(pstate)
            S0:
            begin
                if(Start) begin
                  clr_A <= 1'b1;
                  clr_C_out <= 1'b1;

                end
            end

            S1:
            begin
                if(Start) begin
                  if(A==20'd999999) begin
                        clr_A <= 1'b1;
                        set_C_out <= 1'b1;
                  end
                  else begin
                        incr_A <= 1'b1;
                        clr_C_out <= 1'b1;
                  end
                end
            end
        endcase
  end
endmodule
```

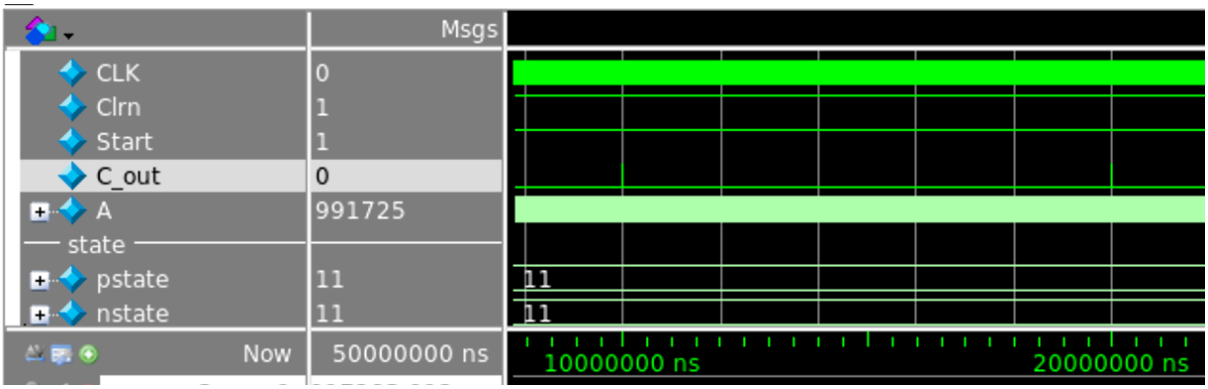
testbench

```
`timescale 1ns/1ns
module Counter_1E6_ASMD_Test_v_tb_jin;

wire C_out;
reg Start;
```

```
reg CLK, Clrn;

initial begin
   #50E6 $finish;
end

initial
begin
   CLK <= 1'b0;
   Clrn <= 1'b0;
   Start <= 1'b1;
   #20 Clrn <= 1'b1; // reset two clock edge
   #35E6 Start <= 1'b0; //start decreasing
   #2E6 Start <= 1'b1; //again start increasing
end

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

Counter_1E6_ASMD_Test_v_jin ct1(
   .C_out(C_out),
   .Start(Start),
   .CLK(CLK),
   .Clrn(Clrn));

always @(C_out) begin
   $monitor("C_out = %d, time = %0d", C_out, $time);
end
endmodule
```

RTL Simulation

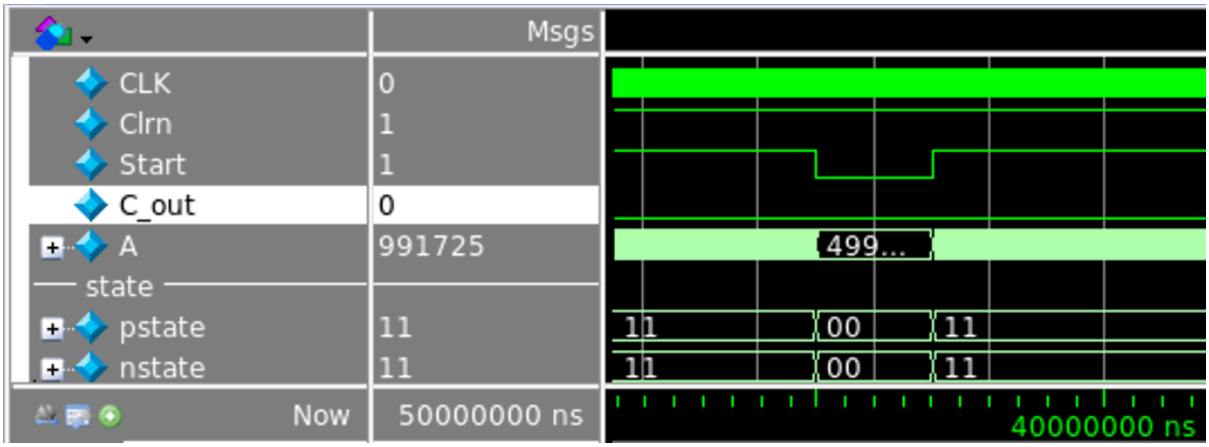
1) Count A




```
# C_out = 0, time = 0
# C_out = 1, time = 10000025
# C_out = 0, time = 10000035
# C_out = 1, time = 20000025
# C_out = 0, time = 20000035
# C_out = 1, time = 30000025
# C_out = 0, time = 30000035
```

At 10000025ns = 10.000025ms , 20.000025ms, 30.000025ms (Generate pulse)
Start = 1
C_out = 1
State : S1 (11)

2) Start off



Start = 0
C_out = 0
State : S0 (00)

Counclusion

I tried to separate Counter module and LD_Driver, but it failed. Because there are many things to consider if we design to separate it. First thing to consider is if using counter that is separated and controlled by an external input named Start as designed above, There is one clock delay to initialize counter and restart counting compared to original design . This is a serious problem because the increasing time 10s is delayed by one clock every changed the state of LD_Driver. Second, if design Counter which do a simple operation that just count, counter cannot be controlled meticulously. In addition, the counting itself already do proper operation in the LD_Driver module implemented in the previous HW#10, 11. Last if control the counter more meticulously, the counter's output itself can become I_out, and the counter itself become the LD_Driver we want to make. For these reasons, I gave up to design LD_Driver which is separated from Counter

Test when separate Counter and LD_Driver

Folllowing my counter design, it need to give external input named Start, but we should keep external input only has SW_ON and LD_ON. For this reason, I removed Start input, and designed it go back to S0(initial state) when finish to count 1E6

Verilog

Top level design (hierarchy)

top.v

```
module top(
   output [11:0] I_out,
   input SW_ON, LD_ON,
   input CLK, Clrn
);

   wire C_out;

   Counter_1E6_ASM_v_jin C1 (
         .C_out(C_out),
         .CLK(CLK),
```

```
        .Clrn(Clrn));

    LD_Driver_ASM_v_jin LD1 (
        .I_out(I_out),
        .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 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;

    always @(posedge CLK, negedge Clrn) begin
        if(~Clrn) begin
            C_out <= 1'b0;
            A <= 20'b0;
            pstate <= S0;
        end
        else begin
            pstate <= nstate; //clocked operation

            //Register transfer operation
            case(pstate)
                S0:
                begin
                    A <= 20'b0;
```

```
                C_out <= 1'b0;
            end

            S1:
            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
        endcase
    end
end

// state trnasition for control logic
always@(pstate, A) begin
    case(pstate)
        S0:
            nstate <= S1;
        S1:
            if(A==20'd999999) nstate <= S0;
            else nstate <= S1;
    endcase
end
endmodule
```

LD_Driver_ASM_v_jin.v

```
module LD_Driver_ASM_v_jin(
    output reg [11:0] I_out,
    input SW_ON, LD_ON, C_out,
    input CLK, Clrn
);
    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;

//state transition
always @(posedge CLK, negedge Clrn) begin
	if(~Clrn) begin
		pstate <= S0;
		I_out <= 12'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;
			  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(C_out == 1'b1) begin
						  I_out <= I_out+ 2'b10;
						end
					end
				end
			  end
			  else LD_ON_reg <= 1'b0;
			end

			S2:
			begin
			  if(SW_ON) begin
```

```
                        LD_ON_reg <= LD_ON;
                        if(!LD_ON_reg) begin
                                if(I_out > 1'b1) begin
                                        if(C_out == 1'b1) begin
                                          I_out <= I_out-3'b100;
                                        end
                                end
                        end
                  end
                  else begin
                        LD_ON_reg <= 1'b0;
                        if(I_out > 1'b1) begin
                                if(C_out == 1'b1) begin
                                        I_out <= I_out-3'b100;
                                end
                        end
                  end
                end

                S3:
                begin
                  if(SW_ON) begin
                        LD_ON_reg <= LD_ON;
                  end
                end
            endcase
        end
end

always @(SW_ON, LD_ON_reg, pstate, I_out) 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;
```

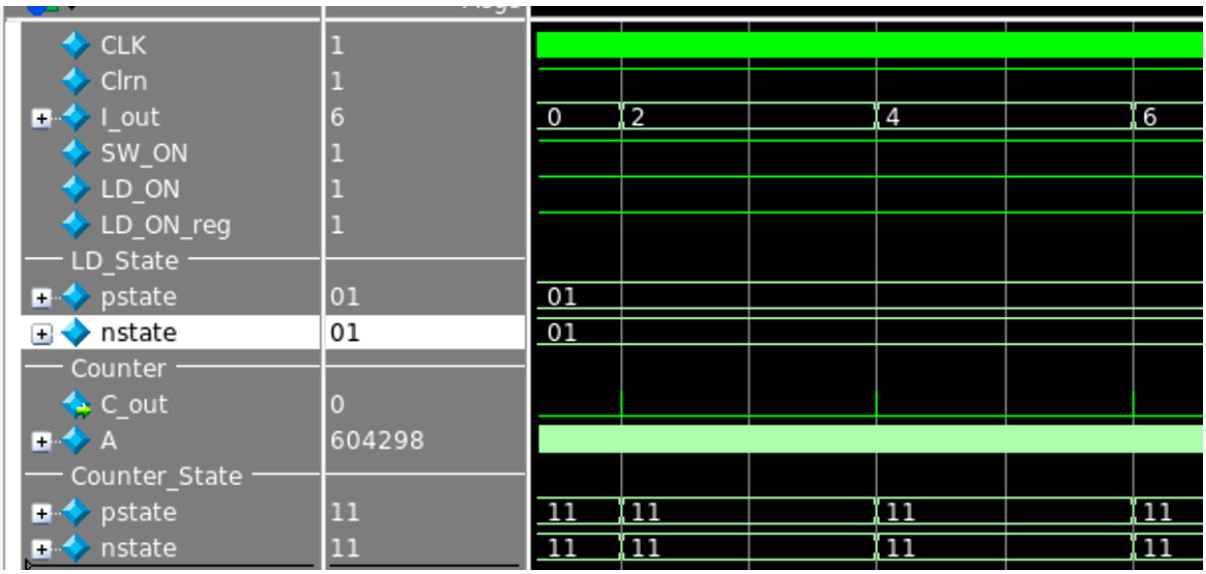
```
                else begin
                  if(SW_ON & LD_ON_reg) nstate <= S1;
                  else nstate <= S2;
                end
            end

            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
```

sti.v (testbency)

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

reg CLK, Clrn;
wire [11:0] I_out;
reg SW_ON, LD_ON;

initial begin
  #30E9 $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
   #11E9 LD_ON <= 1'b0; //start decreasing
   #2E9 LD_ON <= 1'b1; //again start increasing
   #1E9 SW_ON <= 1'b0; //turn off the switch
   #4E9 SW_ON <= 1'b1; //again swithch on
end

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

top t1 (.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);
end
endmodule
```

RTL Simulation



HW#12 – Separate Counter

```
# I_out =    16, time = 80000105
# I_out =    18, time = 90000115
# I_out =    20, time = 100000125
# I_out =    22, time = 110000135
# I_out =    24, time = 120000145
# I_out =    26, time = 130000155
# I_out =    28, time = 140000165
# I_out =    30, time = 150000175
```

HW#10, 11 Counter inside LD_Driver

```
# I_out =    16, time = 80035
# I_out =    18, time = 90035
# I_out =    20, time = 100035
# I_out =    22, time = 110035
# I_out =    24, time = 120035
# I_out =    26, time = 130035
# I_out =    28, time = 140035
# I_out =    30, time = 150035
```

(Since When I did HW#10, 11, I scale down increasing time to 10ms cause too long to simulate)

In the figure of RTL Simulation, It seems module operate properly that I_out is increasing when C_out(pulse) is come out after counting 1E6.

As you can see result of transcript in modelsim when separate Counter from LD_Driver, There is one clock delay to initialize counter and restart counting. In this situation, when I_out arrive at the predefined value, Increasing Time is delayed by 10us. It is serious problem when design LD_Driver. On the other hand when counter is inside LD_Driver, it has only Flip-Flop delay 35ns. More Serious Problem is come out when LD_Driver state change. Since Counter only can reset on master Clrn, it doesn't reset Counting when state changes. on this reason, I_out can be change suddenly. This is not proper operation what we want to design. If Counter acts more meticulous operation, Counter is not a "counter" what it is called. Therefore, In my opinion, Separating counter from LD_Driver is not a good way to design Driver that we want.