HW#13 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 :

increase I\_out :

decrease I\_out :

keep I\_out :

set high value from :

set low value from :

Module Structure

A picture containing text, whiteboard

Description automatically generated

ASM Chart of Counter\_1E6

A close up of text on a white background

Description automatically generated

ASM Chart of LD\_Driver

A picture containing text, whiteboard

Description automatically generated

A picture containing text, map

Description automatically generatedA picture containing text

Description automatically generatedA picture containing text, whiteboard

Description automatically generated

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

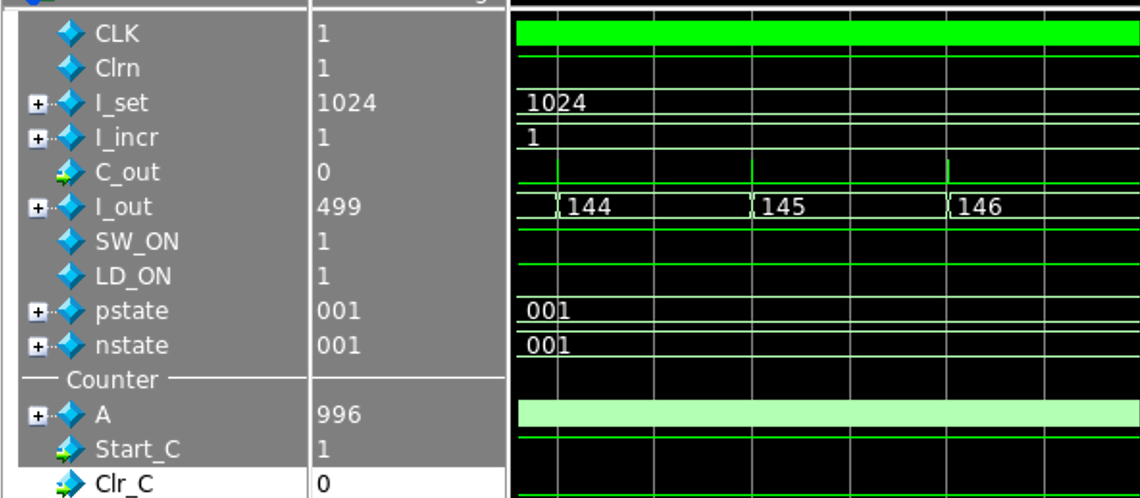
$monitor("I\_out = %d, time = %0d", I\_out, $time);

end

endmodule

RTL Simulation

1) Increasing I\_out



SW\_ON = 1

LD\_ON = 1

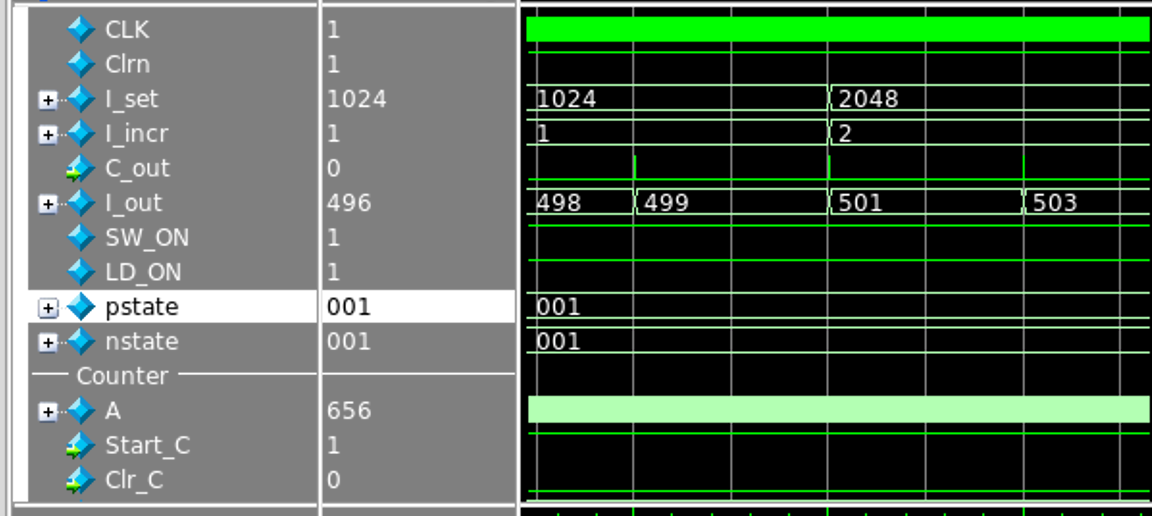
LD\_ON\_reg = 1

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

A screenshot of a video game

Description automatically generated

SW\_ON = 1

LD\_ON = 1

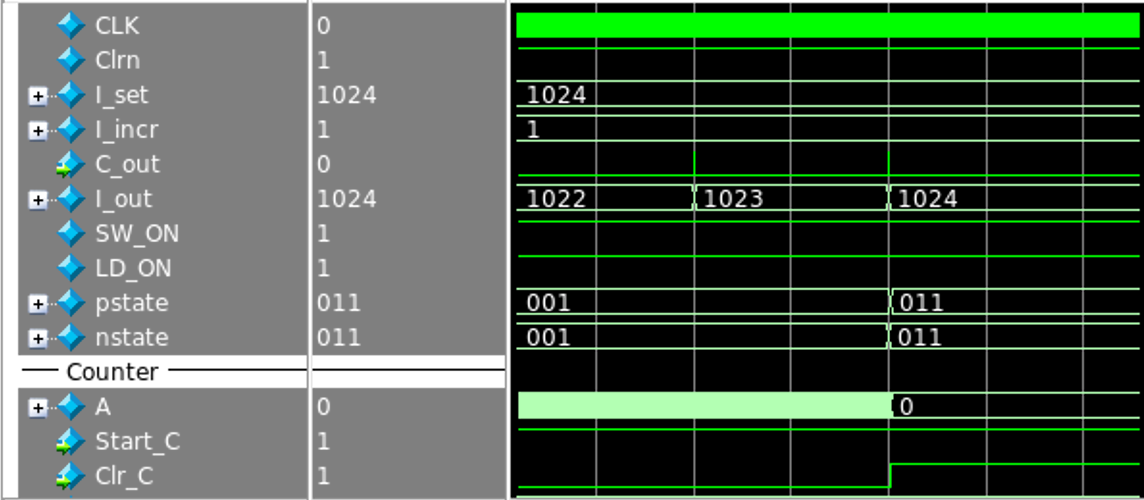
LD\_ON\_reg = 1

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

A screenshot of a cell phone

Description automatically generated

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

A screenshot of a video game

Description automatically generated

SW\_ON = 1

LD\_ON = 1

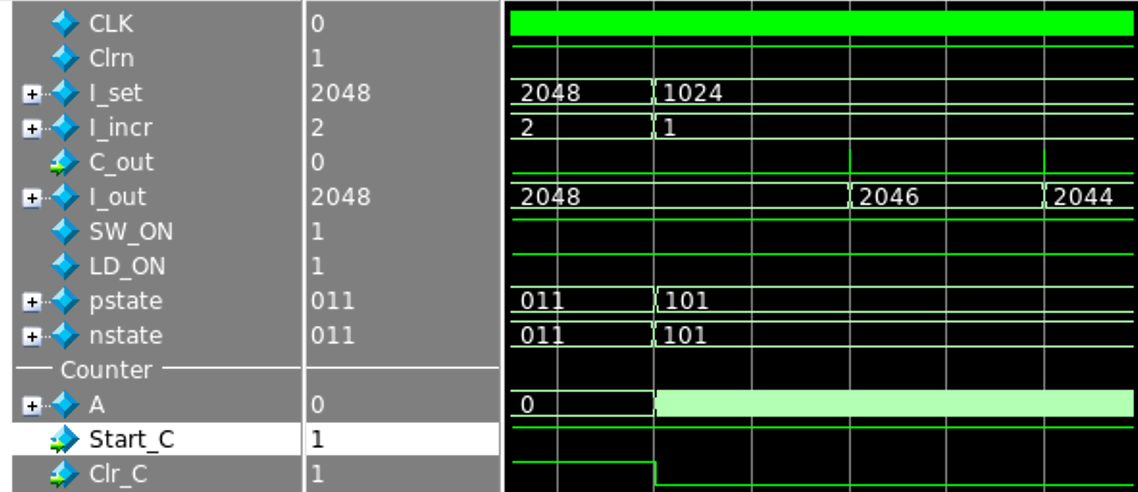
LD\_ON\_reg = 1

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

A screenshot of a cell phone

Description automatically generated

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

A screenshot of a cell phone

Description automatically generated

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

A screenshot of a cell phone

Description automatically generated

SW\_ON = 1

LD\_ON = 0

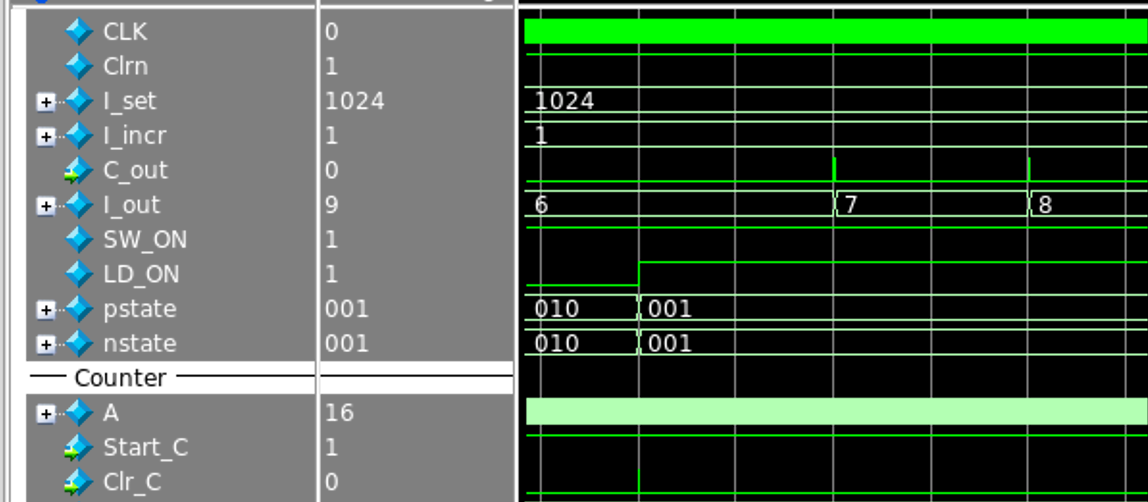
LD\_ON\_reg = 0

I\_set = 2048

I\_incr = 2 (I\_decr : 4)

State : S2 (010)

11) change LD\_ON to 1



SW\_ON = 1

LD\_ON = 1

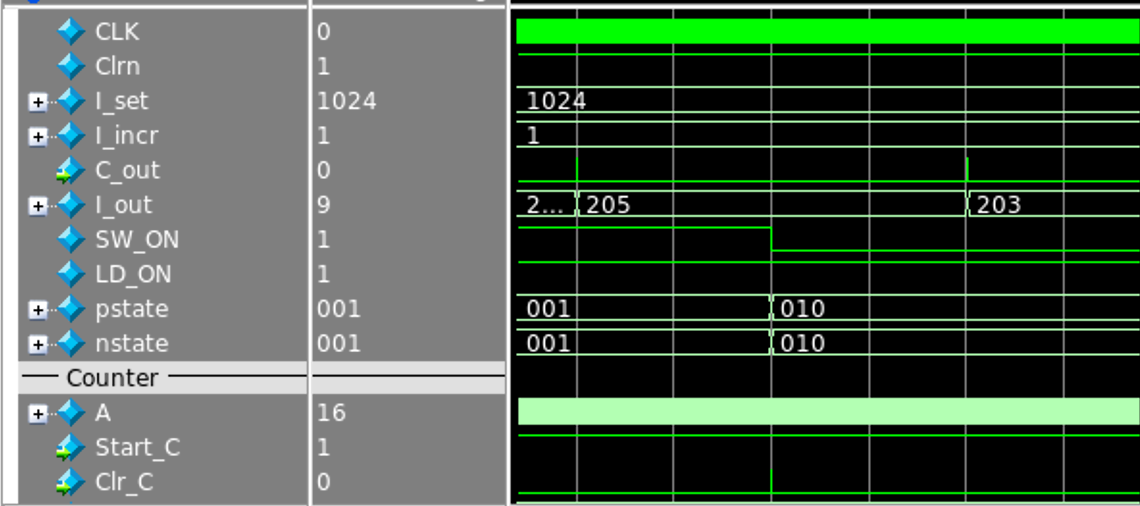
LD\_ON\_reg = 1

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

A screenshot of a video game

Description automatically generated

SW\_ON = 0

LD\_ON = 1

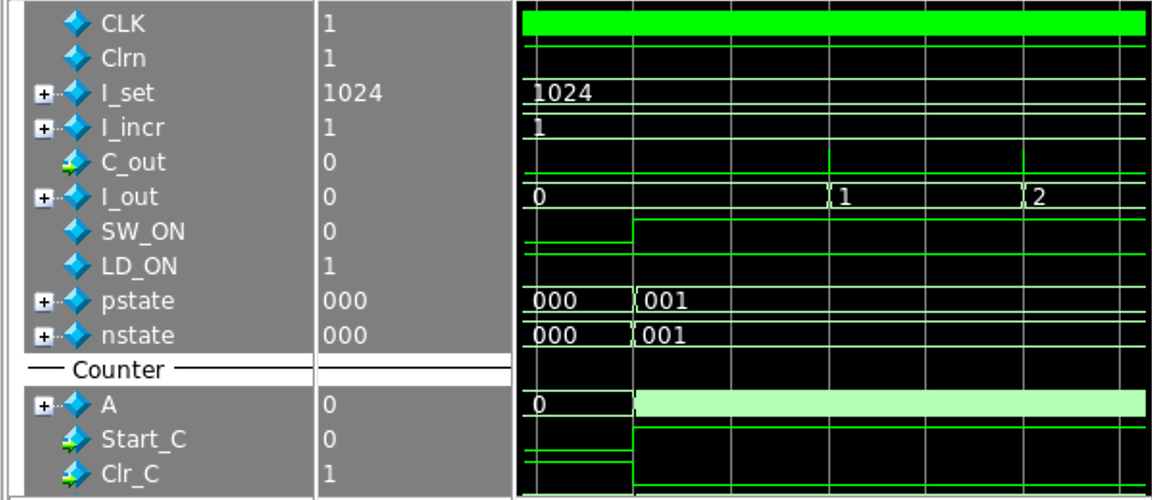
LD\_ON\_reg = 0

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)