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 :

count :

State diagram

A screenshot of a social media post

Description automatically generated

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

A close up of text on a white background

Description automatically generated

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

A screenshot of a video game

Description automatically generated



At 10000025ns = 10.000025ms , 20.000025ms, 30.000025ms

Start = 1

C\_out = 1

State : S1 (11)

2) Start off

A screenshot of a video game

Description automatically generated

Start = 0

C\_out = 0

State : S0 (00)

ASMD approach

State diagram

A screenshot of a cell phone

Description automatically generated

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

A close up of text on a white background

Description automatically generated

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

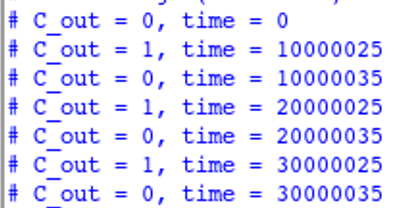
endmodule

RTL Simulation

1) Count A

A screenshot of a video game

Description automatically generated



At 10000025ns = 10.000025ms , 20.000025ms, 30.000025ms (Generate pulse)

Start = 1

C\_out = 1

State : S1 (11)

2) Start off

A screenshot of a video game

Description automatically generated

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

A screenshot of a video game

Description automatically generated

HW#12 – Separate Counter HW#10, 11 Counter inside LD\_Driver

A close up of a screen

Description automatically generatedA screen shot of a computer

Description automatically generated

(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.