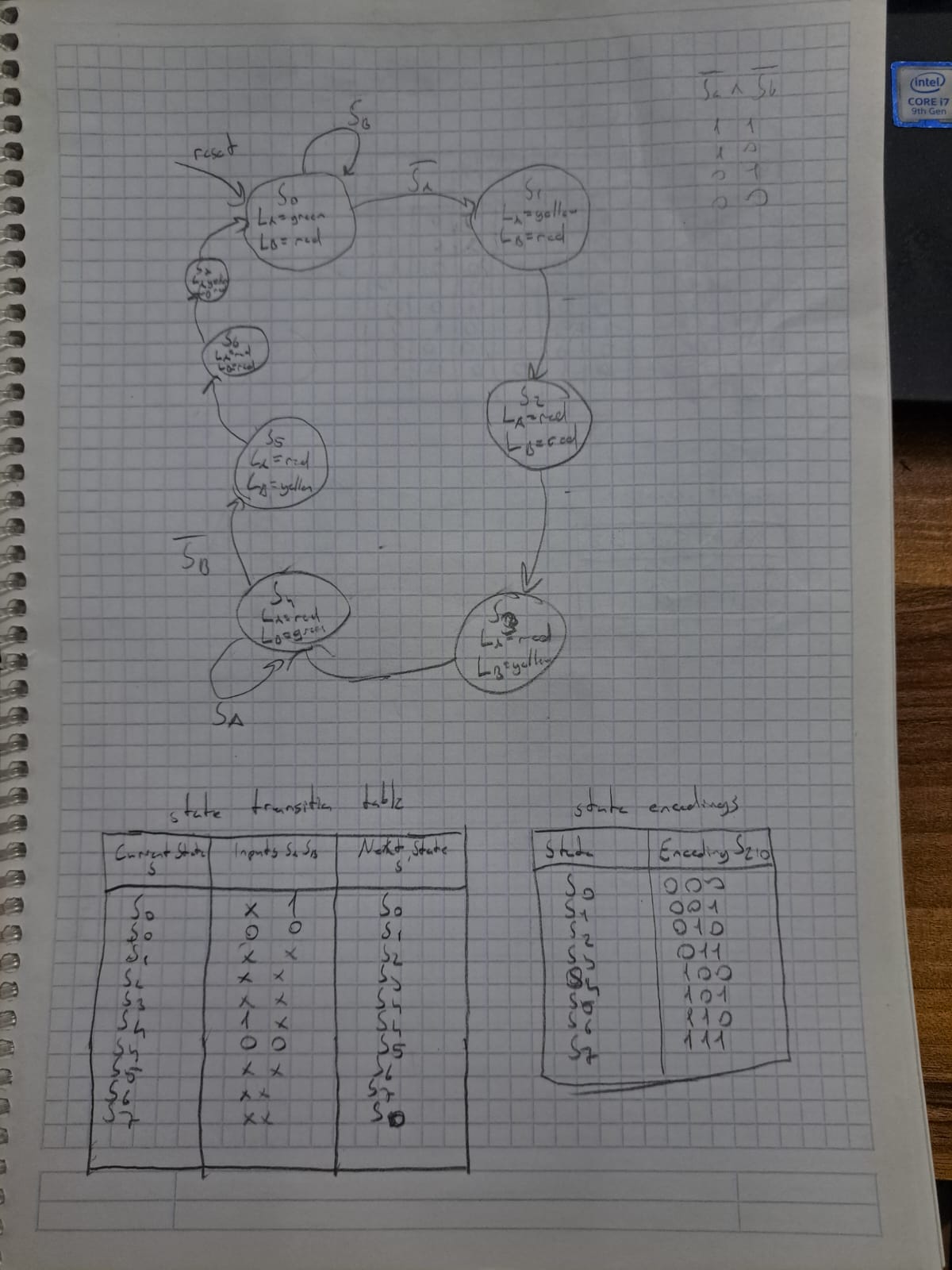
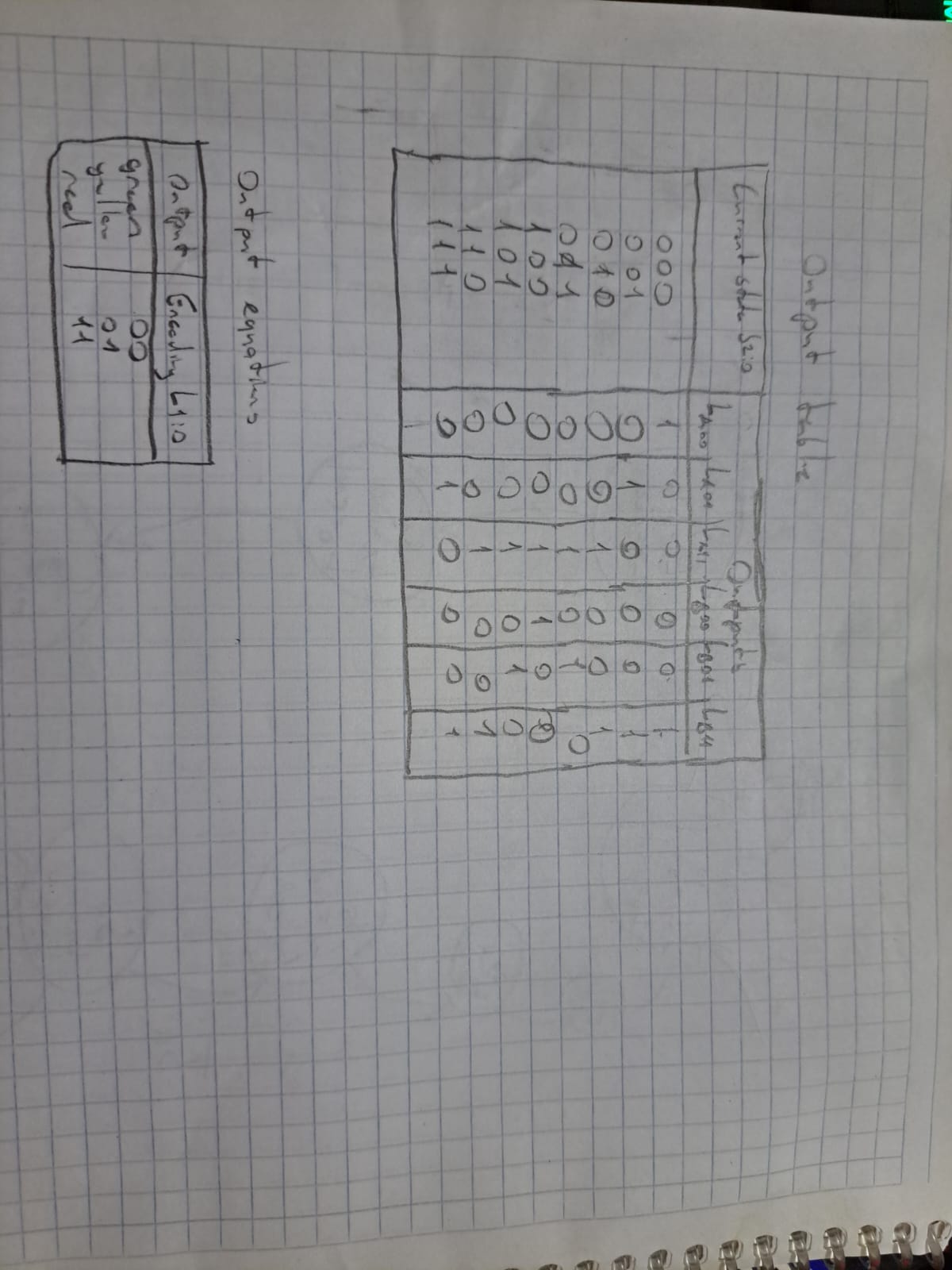
CS223 LAB5

Altay İlker Yiğitel

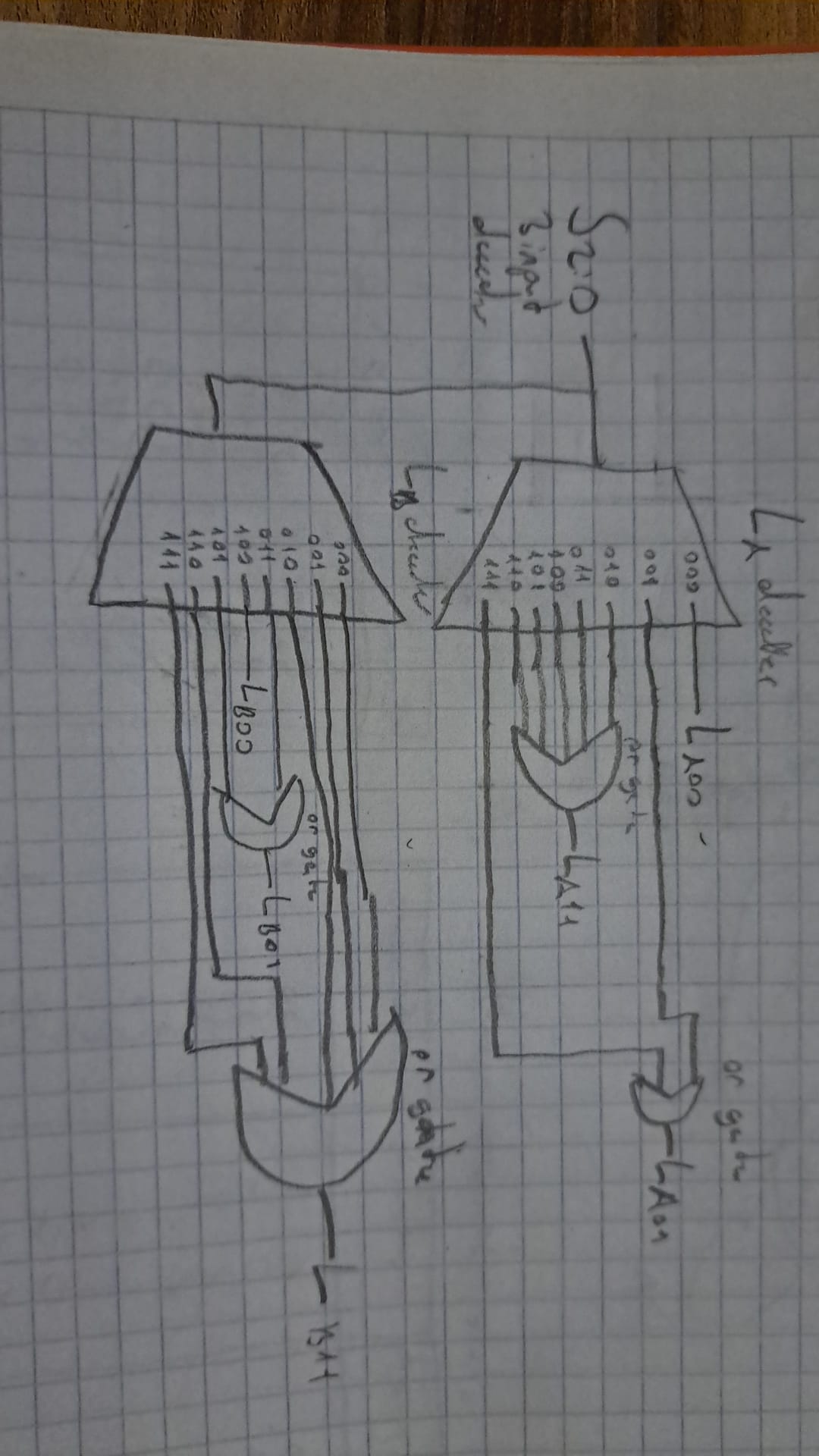
22203024

CS223 sec: 01





2 flip flops are needed as there are 2 important states La=green Lb=red and La=red and Lb=green



System verilog code

Traffic Light System

module traffic(

input SA,

input SB,

output LA\_green,

output LA\_yellow,

output LA\_red,

output LB\_green,

output LB\_yellow,

output LB\_red,

input clk

);

typedef enum logic [3:0] {

BOTH\_RED1,

BOTH\_RED2,

LA\_GREEN,

LB\_GREEN,

LA\_YELLOW1,

LB\_YELLOW1,

LA\_YELLOW2,

LB\_YELLOW2

} state\_t;

state\_t current\_state, next\_state;

logic [1:0] counter;

initial begin

current\_state = LA\_GREEN;

counter = 0;

LA\_green = 1; LA\_yellow = 0; LA\_red = 0;

LB\_green = 0; LB\_yellow = 0; LB\_red = 1;

end

always @(posedge clk) begin

case (current\_state)

LA\_GREEN: begin

if (!(SA && SB)) begin

next\_state = LA\_GREEN;

end else begin

next\_state = LA\_YELLOW1;

end

end

LB\_GREEN: begin

if (!(SA && SB)) begin

next\_state = LB\_GREEN;

end else begin

next\_state = LB\_YELLOW1;

end

end

BOTH\_RED1: begin

if (counter == 3) begin

next\_state = LA\_YELLOW2;

end else begin

next\_state = BOTH\_RED1;

end

end

BOTH\_RED2: begin

if (counter == 3) begin

next\_state = LB\_YELLOW2;

end else begin

next\_state = BOTH\_RED2;

end

end

LA\_YELLOW1: begin

if (counter == 3) begin

next\_state = BOTH\_RED1;

end else begin

next\_state = LA\_YELLOW1;

end

end

LB\_YELLOW1: begin

if (counter == 3) begin

next\_state = BOTH\_RED2;

end else begin

next\_state = LB\_YELLOW1;

end

end

LA\_YELLOW2: begin

if (counter == 3) begin

next\_state = LB\_GREEN;

end else begin

next\_state = LA\_YELLOW2;

end

end

LB\_YELLOW2: begin

if (counter == 3) begin

next\_state = LA\_GREEN;

end else begin

next\_state = LB\_YELLOW2;

end

end

endcase

end

always @(posedge clk) begin

if (counter == 3) begin

counter <= 0;

end else begin

counter <= counter + 1;

end

case (next\_state)

BOTH\_RED: begin

LA\_green = 0; LA\_yellow = 0; LA\_red = 1;

LB\_green = 0; LB\_yellow = 0; LB\_red = 1;

end

BOTH\_YELLOW: begin

LA\_green = 0; LA\_yellow = 1; LA\_red = 0;

LB\_green = 0; LB\_yellow = 1; LB\_red = 0;

end

LA\_GREEN: begin

LA\_green = 1; LA\_yellow = 0; LA\_red = 0;

LB\_green = 0; LB\_yellow = 0; LB\_red = 1;

end

LB\_GREEN: begin

LA\_green = 0; LA\_yellow = 0; LA\_red = 1;

LB\_green = 1; LB\_yellow = 0; LB\_red = 0;

end

LA\_YELLOW: begin

LA\_green = 0; LA\_yellow = 1; LA\_red = 0;

LB\_green = 0; LB\_yellow = 0; LB\_red = 1;

end

LB\_YELLOW: begin

LA\_green = 0; LA\_yellow = 0; LA\_red = 1;

LB\_green = 0; LB\_yellow = 1; LB\_red = 0;

end

endcase

current\_state <= next\_state;

end

endmodule

Testbench

module traffic\_tb;

input SA;

input SB;

input clk;

output LA\_green;

output LA\_yellow;

output LA\_red;

output LB\_green;

output LB\_yellow;

output LB\_red;

traffic dut (

.SA(SA),

.SB(SB),

.LA\_green(LA\_green),

.LA\_yellow(LA\_yellow),

.LA\_red(LA\_red),

.LB\_green(LB\_green),

.LB\_yellow(LB\_yellow),

.LB\_red(LB\_red),

.clk(clk)

);

always #5 clk = ~clk;

initial begin

SA = 0;

SB = 0;

#10;

assert(LA\_green && LB\_red);

SA = 1;

#10;

assert(LA\_yellow && LB\_red);

SA = 0;

SB = 1;

#10;

assert(LB\_green && LA\_red);

SA = 1;

#10;

assert(LA\_red && LB\_red);

$finish;

end

endmodule

Counter

module BCD\_Counter(

input clk,

input reset,

input enable,

input load,

input [3:0] D,

output [3:0] Q

);

always @(posedge clk) begin

if (reset) begin

Q <= 4'b0000;

end

if (enable) begin

if (load) begin

Q <= D;

end else begin

if (Q == 4'b1001) begin

Q <= 4'b0000;

end else begin

Q <= Q + 1;

end

end

end

end

endmodule

testbench

module BCD\_Counter\_TB;

input clk;

input reset;

input enable;

input load;

input [3:0] D;

output [3:0] Q;

BCD\_Counter counter(

.clk(clk),

.reset(reset),

.enable(enable),

.load(load),

.D(D),

.Q(Q)

);

always #((CLK\_PERIOD)/2) clk = ~clk;

initial begin

clk = 0;

reset = 0;

enable = 0;

load = 0;

D = 0;

#20 reset = 1;

#20 reset = 0;

#10 enable = 1;

#20;

assert(Q == 4'b0001) else ;

#30;

assert(Q == 4'b0000) ;

D = 4'b1100;

load = 1;

#20;

load =0;

assert(Q == 4'b1100) ;

#20;

assert(Q == 4'b1101) ;

$finish;

end

endmodule

BCD Counter

module BCD\_Counter(

input clk,

input reset,

input enable,

output [3:0] BCD

);

always @(posedge clk) begin

if (reset) begin

BCD <= 4'b0000;

end else if (enable) begin

if (BCD == 4'b1001) begin

BCD <= 4'b0000;

end else begin

BCD <= BCD+1;

end

end

end

endmodule

module BCD\_Counter\_Reset\_TB;

input clk;

inpt reset;

input enable;

output [3:0] BCD;

BCD\_Counter counter(

.clk(clk),

.reset(reset),

.enable(enable),

.BCD(BCD)

);

always #((CLK\_PERIOD)/2) clk = ~clk;

initial begin

clk = 0;

reset = 0;

enable = 0;

#20 reset = 1;

#20 reset = 0;

#10 enable = 1;

#20;

assert(BCD == 4'b0001) ;

#30;

assert(BCD == 4'b0000) ;

#30;

assert(BCD == 4'b0001) ;

#200;

assert(BCD == 4'b1001) ;

#30;

assert(BCD == 4'b0000) ;

#30;

assert(BCD == 4'b0001);

$finish;

end

endmodule

Two digit

module TwoDigit\_BCD\_Counter(

input clk,

input reset,

output [3:0] ones,

output [3:0] tens

);

always @(posedge clk) begin

if(ones ==9) begin

ones <=0;

if(tens ==9) begin

tens <=0;

end

else begin

tens <= tens+1;

end

end

else begin

ones <=ones+1;

end

endmodule

Testbench

module TwoDigit\_BCD\_Counter\_TB;

reg clk;

reg reset;

wire [3:0] ones;

wire [3:0] tens;

TwoDigit\_BCD\_Counter counter(

.clk(clk),

.reset(reset),

.ones(ones),

.tens(tens),

);

always #((CLK\_PERIOD)/2) clk = ~clk;

initial begin

clk = 0;

reset = 0;

#20 reset = 1;

#20 reset = 0;

#10;

assert(ones == 4'b0000 && tens 4'b0000);

$finish;

end

endmodule