//this is just a simple interface to give input signals to the fsm to let the light control system working

module top\_interface(

input logic clk,

input logic reset,

input logic start,

input logic [7:0] pass\_key,

input logic mess\_traffic\_flag,

input logic daytime\_flag,

output logic [2:0] EW\_straight\_lights\_flag,

output logic [2:0] SN\_straight\_lights\_flag,

output logic ES\_turning\_lights\_flag,

output logic WN\_turning\_lights\_flag,

output logic SW\_turning\_lights\_flag,

output logic NE\_turning\_lights\_flag,

output logic [2:0] EWS\_passenger\_flag,

output logic [2:0] EWN\_passenger\_flag,

output logic [2:0] SNE\_passenger\_flag,

output logic [2:0] SNW\_passenger\_flag

);

//internal flag signals to control the length of each state

logic white2Tminus\_flag;

logic green2yellow\_flag;

logic red\_flag;

logic need\_white2Tminus\_flag;

logic need\_green2yellow\_flag;

logic need\_red\_flag;

logic reds\_on\_flag;

// module instantiate

fsm control\_fsm(

clk,

reset,

start,

pass\_key,

mess\_traffic\_flag,

daytime\_flag,

white2Tminus\_flag,

green2yellow\_flag,

red\_flag,

need\_white2Tminus\_flag,

need\_green2yellow\_flag,

need\_red\_flag,

reds\_on\_flag,

EW\_straight\_lights\_flag,

SN\_straight\_lights\_flag,

ES\_turning\_lights\_flag,

WN\_turning\_lights\_flag,

SW\_turning\_lights\_flag,

NE\_turning\_lights\_flag,

EWS\_passenger\_flag,

EWN\_passenger\_flag,

SNE\_passenger\_flag,

SNW\_passenger\_flag

);

clock\_divider control\_delay(

clk, reds\_on\_flag,need\_white2Tminus\_flag,need\_green2yellow\_flag,

need\_red\_flag,white2Tminus\_flag,green2yellow\_flag,red\_flag

);

endmodule

module top\_tb();

reg clk;

reg reset;

reg start;

reg [7:0] pass\_key;

reg mess\_traffic\_flag;

reg daytime\_flag;

wire white2Tminus\_flag;

wire green2yellow\_flag;

wire red\_flag;

wire need\_white2Tminus\_flag;

wire need\_green2yellow\_flag;

wire need\_red\_flag;

wire reds\_on\_flag;

wire [2:0] EW\_straight\_lights\_flag;

wire [2:0] SN\_straight\_lights\_flag;

wire ES\_turning\_lights\_flag;

wire WN\_turning\_lights\_flag;

wire SW\_turning\_lights\_flag;

wire NE\_turning\_lights\_flag;

wire [2:0] EWS\_passenger\_flag;

wire [2:0] EWN\_passenger\_flag;

wire [2:0] SNE\_passenger\_flag;

wire [2:0] SNW\_passenger\_flag;

wire [28:0] state;

assign state = DUT.control\_fsm.state;

assign need\_white2Tminus\_flag = DUT.need\_white2Tminus\_flag;

assign need\_green2yellow\_flag = DUT.need\_green2yellow\_flag;

assign need\_red\_flag = DUT.need\_red\_flag;

assign reds\_on\_flag = DUT.reds\_on\_flag;

assign white2Tminus\_flag = DUT.white2Tminus\_flag;

assign green2yellow\_flag = DUT.green2yellow\_flag;

assign red\_flag = DUT.red\_flag;

top\_interface DUT (clk,

reset,

start,

pass\_key,

mess\_traffic\_flag,

daytime\_flag,

EW\_straight\_lights\_flag,

SN\_straight\_lights\_flag,

ES\_turning\_lights\_flag,

WN\_turning\_lights\_flag,

SW\_turning\_lights\_flag,

NE\_turning\_lights\_flag,

EWS\_passenger\_flag,

EWN\_passenger\_flag,

SNE\_passenger\_flag,

SNW\_passenger\_flag

);

always #5 clk = ~clk; //clk generator

initial begin

clk = 1'b1;

start = 0;

reset = 0;

#10;

reset = 1;

#10;

reset = 0;

pass\_key = 8'b00000000;

mess\_traffic\_flag = 0;

daytime\_flag = 0; //set all initial values

#10;

start = 1;

#10;

start = 0;

mess\_traffic\_flag = 1;

#100; // here is for testing the system when it not in day time

daytime\_flag = 1;

pass\_key = 8'b00000001; //when it is daytime in rush hours, button is disabled, the system runing in a specific logic

#1000;

mess\_traffic\_flag = 0;// when it s in off-peak hours, passengers can control the system by pressing buttons.

#1000;

$stop;

end

endmodule

//this is state machine module. the only function of this module is controlling the combination of signal lights

//other matters such as how long the lights will keep on or if the traffic is heavy will be determined in other modules

//therefore all the states are about the light combination.

//when the outside module finished counting, flags will be input and allow the fsm processing.

module fsm(

input logic clk,

input logic reset,

input logic start,

input logic [7:0] pass\_key,

input logic mess\_traffic\_flag,

input logic daytime\_flag,

//i think that i can put the traffic counter and daytime conunter out side of the FSM so there will be not that many states.

input logic white2Tminus\_flag,

input logic green2yellow\_flag,

input logic red\_flag,

output logic need\_white2Tminus\_flag,

output logic need\_green2yellow\_flag,

output logic need\_red\_flag,

output logic reds\_on\_flag,

output logic [2:0] EW\_straight\_lights\_flag,

output logic [2:0] SN\_straight\_lights\_flag,

output logic ES\_turning\_lights\_flag,

output logic WN\_turning\_lights\_flag,

output logic SW\_turning\_lights\_flag,

output logic NE\_turning\_lights\_flag,

output logic [2:0] EWS\_passenger\_flag,

output logic [2:0] EWN\_passenger\_flag,

output logic [2:0] SNE\_passenger\_flag,

output logic [2:0] SNW\_passenger\_flag

);

reg [28:0] state;

reg D\_flag; //to determine the previous lights comb condition, if the previous direction is EW, then flag value is 0, otherwise is 1

assign D\_flag = start ? 1 : state[0];

assign reds\_on\_flag = (state[5]|state[6]|state[8]|state[9]) ? 1 : 0;

assign need\_white2Tminus\_flag = state[1];

assign need\_green2yellow\_flag = state[2];

assign need\_red\_flag = state[3];

assign EW\_straight\_lights\_flag = state[6:4];

assign SN\_straight\_lights\_flag = state[9:7];

assign ES\_turning\_lights\_flag = state[10];

assign WN\_turning\_lights\_flag = state[11];

assign SW\_turning\_lights\_flag = state[12];

assign NE\_turning\_lights\_flag = state[13];

assign EWS\_passenger\_flag = state[16:14];

assign EWN\_passenger\_flag = state[19:17];

assign SNE\_passenger\_flag = state[22:20];

assign SNW\_passenger\_flag = state[25:23];

//state parameters. i wanna use there states to control lights, so i arrange the array in a special way that every part of the array can carry information

//

//for parameters, i use the digits for flag information

//id/SNE\_passenger\_flag/SNE\_passenger\_flag/EWN\_passenger\_flag/EWS\_passenger\_flag/NE\_turning\_lights\_flag/SW\_turning\_lights\_flag/WN\_turning\_lights\_flag/ES\_turning\_lights\_flag/SN\_straight\_lights\_flag/EW\_straight\_lights\_flag/need\_end\_flag/need\_red\_flag/need\_green2yellow\_flag/need\_white2Tminus\_flag/previous direction

parameter initial\_waiting = 29'b000\_000\_000\_000\_000\_0000\_000\_000\_0\_0\_0\_0;

parameter check\_condition = 29'b001\_000\_000\_000\_000\_0000\_000\_000\_0\_0\_0\_0;

//for normal amount of traffic, straight pass first then left truning

parameter EW\_strat\_green\_SNwhite = 29'b000\_001\_001\_100\_100\_0000\_001\_100\_0\_1\_1\_0;

parameter EW\_strat\_green\_Swhite\_Nred = 29'b000\_001\_001\_001\_100\_0000\_001\_100\_0\_1\_1\_0;

parameter EW\_strat\_green\_Sred\_Nwhite = 29'b000\_001\_001\_100\_001\_0000\_001\_100\_0\_1\_1\_0;

parameter EW\_strat\_green\_SNred = 29'b000\_001\_001\_001\_001\_0000\_001\_100\_0\_1\_0\_0;

parameter EW\_strat\_green\_SNTminus = 29'b000\_001\_001\_010\_010\_0000\_001\_100\_0\_1\_0\_0;

parameter EW\_strat\_green\_STminus\_Nred = 29'b000\_001\_001\_001\_010\_0000\_001\_100\_0\_1\_0\_0;

parameter EW\_strat\_green\_Sred\_NTminus = 29'b000\_001\_001\_010\_001\_0000\_001\_100\_0\_1\_0\_0;

parameter EW\_strat\_yellow\_SNred = 29'b000\_001\_001\_001\_001\_0000\_001\_010\_1\_0\_0\_0;

parameter EW\_strat\_yellow\_SNTminus = 29'b000\_001\_001\_010\_010\_0000\_001\_010\_1\_0\_0\_0;

parameter EW\_strat\_yellow\_STminus\_Nred = 29'b000\_001\_001\_001\_010\_0000\_001\_010\_1\_0\_0\_0;

parameter EW\_strat\_yellow\_Sred\_NTminus = 29'b000\_001\_001\_010\_001\_0000\_001\_010\_1\_0\_0\_0;

parameter EW\_strat\_red\_red = 29'b000\_001\_001\_001\_001\_0000\_001\_001\_0\_0\_0\_0;

parameter SN\_strat\_green\_EWwhite = 29'b000\_100\_100\_001\_001\_0000\_100\_001\_0\_1\_1\_1;

parameter SN\_strat\_green\_Ewhite\_Wred = 29'b000\_001\_100\_001\_001\_0000\_100\_001\_0\_1\_1\_1;

parameter SN\_strat\_green\_Ered\_Wwhite = 29'b000\_100\_001\_001\_001\_0000\_100\_001\_0\_1\_1\_1;

parameter SN\_strat\_green\_EWred = 29'b000\_001\_001\_001\_001\_0000\_100\_001\_0\_1\_1\_1;

parameter SN\_strat\_green\_EWTminus = 29'b000\_010\_010\_001\_001\_0000\_100\_001\_0\_1\_0\_1;

parameter SN\_strat\_green\_ETminus\_Wred = 29'b000\_001\_010\_001\_001\_0000\_100\_001\_0\_1\_0\_1;

parameter SN\_strat\_green\_Ered\_WTminus = 29'b000\_010\_001\_001\_001\_0000\_100\_001\_0\_1\_0\_1;

parameter SN\_strat\_yellow\_EWred = 29'b000\_001\_001\_001\_001\_0000\_010\_001\_1\_0\_0\_1;

parameter SN\_strat\_yellow\_EWTminus = 29'b000\_010\_010\_001\_001\_0000\_010\_001\_1\_0\_0\_1;

parameter SN\_strat\_yellow\_ETminus\_Wred = 29'b000\_001\_010\_001\_001\_0000\_010\_001\_1\_0\_0\_1;

parameter SN\_strat\_yellow\_Ered\_WTminus = 29'b000\_010\_001\_001\_001\_0000\_010\_001\_1\_0\_0\_1;

parameter SN\_strat\_red\_red = 29'b000\_001\_001\_001\_001\_0000\_001\_001\_0\_0\_0\_1;

//for heavy traffic, one direction each time with straight and left truning

parameter ES\_turn\_green\_Nwhite\_Sred = 29'b000\_001\_001\_100\_001\_0001\_001\_100\_0\_1\_1\_0;

parameter WN\_turn\_green\_Swhite\_Nred = 29'b000\_001\_001\_001\_100\_0010\_001\_100\_0\_1\_1\_0;

parameter SW\_turn\_green\_Ewhite\_Wred = 29'b000\_001\_100\_001\_001\_0100\_100\_001\_0\_1\_1\_0;

parameter NE\_turn\_green\_Wwhite\_Ered = 29'b000\_100\_001\_001\_001\_1000\_100\_001\_0\_1\_1\_0;

parameter ES\_turn\_green\_NTminus\_Sred = 29'b000\_001\_001\_010\_001\_0001\_001\_100\_0\_1\_0\_0;

parameter WN\_turn\_green\_STminus\_Nred = 29'b000\_001\_001\_001\_010\_0010\_001\_100\_0\_1\_0\_0;

parameter SW\_turn\_green\_ETminus\_Wred = 29'b000\_001\_010\_001\_001\_0100\_100\_001\_0\_1\_0\_0;

parameter NE\_turn\_green\_WTminus\_Ered = 29'b000\_010\_001\_001\_001\_1000\_100\_001\_0\_1\_0\_0;

parameter ES\_turn\_yellow\_NTminus\_Sred = 29'b001\_001\_001\_010\_001\_0000\_001\_010\_1\_0\_0\_0;

parameter WN\_turn\_yellow\_STminus\_Nred = 29'b001\_001\_001\_001\_010\_0000\_001\_010\_1\_0\_0\_0;

parameter SW\_turn\_yellow\_ETminus\_Wred = 29'b001\_001\_010\_001\_001\_0000\_010\_001\_1\_0\_0\_0;

parameter NE\_turn\_yellow\_WTminus\_Ered = 29'b001\_010\_001\_001\_001\_0000\_010\_001\_1\_0\_0\_0;

parameter ES\_turn\_red\_red = 29'b010\_001\_001\_001\_001\_0000\_001\_001\_0\_0\_0\_0;

parameter WN\_turn\_red\_red = 29'b011\_001\_001\_001\_001\_0000\_001\_001\_0\_0\_0\_0;

parameter SW\_turn\_red\_red = 29'b100\_001\_001\_001\_001\_0000\_001\_001\_0\_0\_0\_0;

parameter NE\_turn\_red\_red = 29'b101\_001\_001\_001\_001\_0000\_001\_001\_0\_0\_0\_0;

parameter night\_all\_red = 29'b110\_001\_001\_001\_001\_0000\_001\_001\_0\_0\_0\_0;

always @(posedge clk)begin

if(reset) state <= initial\_waiting;

else begin

case(state)

initial\_waiting: begin

if(start) begin

state <= check\_condition;

end

else begin

state <= initial\_waiting;

end

end

check\_condition: begin

if (daytime\_flag)begin

if(mess\_traffic\_flag) begin

state <= ES\_turn\_green\_Nwhite\_Sred; end //if it's weekday day time in rash hour, then ignore the pass key, directly follow the order: ES, WN, SW, NE

else begin

if(!D\_flag) begin

if(pass\_key[0]|pass\_key[7]) begin

if(pass\_key[3]|pass\_key[4]) begin

state <= SN\_strat\_green\_EWwhite; end

else begin

state <= SN\_strat\_green\_Ered\_Wwhite; end

end

else if(pass\_key[3]|pass\_key[4]) begin

state <= SN\_strat\_green\_Ewhite\_Wred; end

else begin

state <= SN\_strat\_green\_EWred; end

end

else begin

if(pass\_key[1]|pass\_key[2]) begin

if(pass\_key[5]|pass\_key[6]) begin

state <= EW\_strat\_green\_SNwhite; end

else begin

state <= EW\_strat\_green\_Sred\_Nwhite; end

end

else if(pass\_key[5]|pass\_key[6]) begin

state <= EW\_strat\_green\_Swhite\_Nred; end

else begin

state <= EW\_strat\_green\_SNred; end

end

end

end

else begin//night

state <= night\_all\_red;

end

end

night\_all\_red: begin

if(daytime\_flag) state <= check\_condition;

else state <= night\_all\_red;

end

/\* else begin//weekend no rush hours

if (daytime\_flag)begin

if(pass\_key[0]|pass\_key[7]) begin

if(pass\_key[3]|pass\_key[4]) begin

state <= SN\_strat\_green\_EWwhite; end

else begin

state <= SN\_strat\_green\_Ered\_Wwhite; end

end

else if(pass\_key[3]|pass\_key[4]) begin

if(pass\_key[0]|pass\_key[7]) begin

state <= SN\_strat\_green\_EWwhite; end

else begin

state <= SN\_strat\_green\_Ewhite\_Wred; end

end

else if(pass\_key[1]|pass\_key[2]) begin

if(pass\_key[5]|pass\_key[6]) begin

state <= EW\_strat\_green\_SNwhite; end

else begin

state <= EW\_strat\_green\_Sred\_Nwhite; end

end

else if(pass\_key[5]|pass\_key[6]) begin

if(pass\_key[1]|pass\_key[2]) begin

state <= EW\_strat\_green\_SNwhite; end

else begin

state <= EW\_strat\_green\_Swhite\_Nred; end

end

else begin

state <= EW\_strat\_green\_SNred; end

end

else begin//night

state <= night\_all\_red; end

end

end\*/

ES\_turn\_green\_Nwhite\_Sred: begin

if(white2Tminus\_flag) begin

state <= ES\_turn\_green\_NTminus\_Sred; end

else begin

state <= ES\_turn\_green\_Nwhite\_Sred; end

end

ES\_turn\_green\_NTminus\_Sred: begin

if(green2yellow\_flag) begin

state <= ES\_turn\_yellow\_NTminus\_Sred; end

else begin

state <= ES\_turn\_green\_NTminus\_Sred; end

end

ES\_turn\_yellow\_NTminus\_Sred: begin

if(red\_flag) begin

state <= ES\_turn\_red\_red; end

else begin

state <= ES\_turn\_yellow\_NTminus\_Sred; end

end

ES\_turn\_red\_red: begin

state <= WN\_turn\_green\_Swhite\_Nred;

end

WN\_turn\_green\_Swhite\_Nred: begin

if(white2Tminus\_flag) begin

state <= WN\_turn\_green\_STminus\_Nred; end

else begin

state <= WN\_turn\_green\_Swhite\_Nred; end

end

WN\_turn\_green\_STminus\_Nred: begin

if(green2yellow\_flag) begin

state <= WN\_turn\_yellow\_STminus\_Nred; end

else begin

state <= WN\_turn\_green\_STminus\_Nred; end

end

WN\_turn\_yellow\_STminus\_Nred: begin

if(red\_flag) begin

state <= WN\_turn\_red\_red; end

else begin

state <= WN\_turn\_yellow\_STminus\_Nred; end

end

WN\_turn\_red\_red: begin

state <= SW\_turn\_green\_Ewhite\_Wred; end

SW\_turn\_green\_Ewhite\_Wred: begin

if(white2Tminus\_flag) begin

state <= SW\_turn\_green\_ETminus\_Wred; end

else begin

state <= SW\_turn\_green\_Ewhite\_Wred; end

end

SW\_turn\_green\_ETminus\_Wred: begin

if(green2yellow\_flag) begin

state <= SW\_turn\_yellow\_ETminus\_Wred; end

else begin

state <= SW\_turn\_green\_ETminus\_Wred; end

end

SW\_turn\_yellow\_ETminus\_Wred: begin

if(red\_flag) begin

state <= SW\_turn\_red\_red; end

else begin

state <= SW\_turn\_yellow\_ETminus\_Wred; end

end

SW\_turn\_red\_red: begin

state <= NE\_turn\_green\_Wwhite\_Ered; end

NE\_turn\_green\_Wwhite\_Ered: begin

if(white2Tminus\_flag) begin

state <= NE\_turn\_green\_WTminus\_Ered; end

else begin

state <= NE\_turn\_green\_Wwhite\_Ered; end

end

NE\_turn\_green\_WTminus\_Ered: begin

if(green2yellow\_flag) begin

state <= NE\_turn\_yellow\_WTminus\_Ered; end

else begin

state <= NE\_turn\_green\_WTminus\_Ered; end

end

NE\_turn\_yellow\_WTminus\_Ered: begin

if(red\_flag) begin

state <= NE\_turn\_red\_red; end

else begin

state <= NE\_turn\_yellow\_WTminus\_Ered; end

end

NE\_turn\_red\_red: begin

state <= check\_condition; end

SN\_strat\_green\_EWwhite: begin

if(white2Tminus\_flag) begin

state <= SN\_strat\_green\_EWTminus; end

else begin

state <= SN\_strat\_green\_EWwhite; end

end

SN\_strat\_green\_EWTminus: begin

if(green2yellow\_flag) begin

state <= SN\_strat\_yellow\_EWTminus; end

else begin

state <= SN\_strat\_green\_EWTminus; end

end

SN\_strat\_yellow\_EWTminus: begin

if(red\_flag) begin

state <= SN\_strat\_red\_red; end

else begin

state <= SN\_strat\_yellow\_EWTminus; end

end

SN\_strat\_red\_red: begin

state <= check\_condition; end

SN\_strat\_green\_Ewhite\_Wred: begin

if(white2Tminus\_flag) begin

state <= SN\_strat\_green\_ETminus\_Wred; end

else begin

state <= SN\_strat\_green\_Ewhite\_Wred; end

end

SN\_strat\_green\_ETminus\_Wred: begin

if(green2yellow\_flag) begin

state <= SN\_strat\_yellow\_ETminus\_Wred; end

else begin

state <= SN\_strat\_green\_ETminus\_Wred; end

end

SN\_strat\_yellow\_ETminus\_Wred: begin

if(red\_flag) begin

state <= SN\_strat\_red\_red; end

else begin

state <= SN\_strat\_yellow\_ETminus\_Wred; end

end

SN\_strat\_green\_Ered\_Wwhite: begin

if(white2Tminus\_flag) begin

state <= SN\_strat\_green\_Ered\_WTminus; end

else begin

state <= SN\_strat\_green\_Ered\_Wwhite; end

end

SN\_strat\_green\_Ered\_WTminus: begin

if(green2yellow\_flag) begin

state <= SN\_strat\_yellow\_Ered\_WTminus; end

else begin

state <= SN\_strat\_green\_Ered\_WTminus; end

end

SN\_strat\_yellow\_Ered\_WTminus: begin

if(red\_flag) begin

state <= SN\_strat\_red\_red; end

else begin

state <= SN\_strat\_yellow\_Ered\_WTminus; end

end

SN\_strat\_green\_EWred: begin

if(green2yellow\_flag) begin

state <= SN\_strat\_yellow\_EWred; end

else begin

state <= SN\_strat\_green\_EWred; end

end

SN\_strat\_yellow\_EWred: begin

if(red\_flag) begin

state <= SN\_strat\_red\_red; end

else begin

state <= SN\_strat\_yellow\_EWred; end

end

EW\_strat\_green\_SNwhite: begin

if(white2Tminus\_flag) begin

state <= EW\_strat\_green\_SNTminus; end

else begin

state <= EW\_strat\_green\_SNwhite; end

end

EW\_strat\_green\_SNTminus: begin

if(green2yellow\_flag) begin

state <= EW\_strat\_yellow\_SNTminus; end

else begin

state <= EW\_strat\_green\_SNTminus; end

end

EW\_strat\_yellow\_SNTminus: begin

if(red\_flag) begin

state <= EW\_strat\_red\_red; end

else begin

state <= EW\_strat\_yellow\_SNTminus; end

end

EW\_strat\_red\_red: begin

state <= check\_condition; end

EW\_strat\_green\_Swhite\_Nred: begin

if(white2Tminus\_flag) begin

state <= EW\_strat\_green\_STminus\_Nred; end

else begin

state <= EW\_strat\_green\_Swhite\_Nred; end

end

EW\_strat\_green\_STminus\_Nred: begin

if(green2yellow\_flag) begin

state <= EW\_strat\_yellow\_STminus\_Nred; end

else begin

state <= EW\_strat\_green\_STminus\_Nred; end

end

EW\_strat\_yellow\_STminus\_Nred: begin

if(red\_flag) begin

state <= EW\_strat\_red\_red; end

else begin

state <= EW\_strat\_yellow\_STminus\_Nred; end

end

EW\_strat\_green\_Sred\_Nwhite: begin

if(white2Tminus\_flag) begin

state <= EW\_strat\_green\_Sred\_NTminus; end

else begin

state <= EW\_strat\_green\_Sred\_Nwhite; end

end

EW\_strat\_green\_Sred\_NTminus: begin

if(green2yellow\_flag) begin

state <= EW\_strat\_yellow\_Sred\_NTminus; end

else begin

state <= EW\_strat\_green\_Sred\_NTminus; end

end

EW\_strat\_yellow\_Sred\_NTminus: begin

if(red\_flag) begin

state <= EW\_strat\_red\_red; end

else begin

state <= EW\_strat\_yellow\_Sred\_NTminus; end

end

EW\_strat\_green\_SNred: begin

if(green2yellow\_flag) begin

state <= EW\_strat\_yellow\_SNred; end

else begin

state <= EW\_strat\_green\_SNred; end

end

EW\_strat\_yellow\_SNred: begin

if(red\_flag) begin

state <= EW\_strat\_red\_red; end

else begin

state <= EW\_strat\_yellow\_SNred; end

end

endcase

end

end

endmodule

module clock\_divider(

input logic clk,

input logic reds\_on\_flag,

input logic need\_white2Tminus\_flag,

input logic need\_green2yellow\_flag,

input logic need\_red\_flag,

output logic white2Tminus\_flag,

output logic green2yellow\_flag,

output logic red\_flag

);

parameter cyc\_10 = 4'b1010;

parameter cyc\_30 = 5'b11110;

logic [3:0] counter1 = 0; //this delay is for white2Tminus and red\_flag

logic [4:0] counter2 = 0; // this delay is for green2yellow

logic [4:0] counter = 0;

always\_ff @(posedge clk) begin

if(need\_white2Tminus\_flag) begin

if(counter1 <= cyc\_10)begin

counter1 = counter1 + 1;

end

else begin

counter1 = 0;

white2Tminus\_flag = 1;

red\_flag = 0;

end

end

else if(need\_green2yellow\_flag) begin

if(counter2 <= cyc\_30)begin

counter2 = counter2 + 1;

end

else begin

counter2 = 0;

green2yellow\_flag = 1;

end

end

else if(need\_red\_flag) begin

if(counter <= cyc\_10)begin

counter = counter + 1;

end

else begin

counter = 0;

red\_flag = 1;

white2Tminus\_flag = 0;

green2yellow\_flag = 0;

end

end

end

endmodule