**交通信号灯项目设计**

**代码**

`timescale 1ns / 1ps

////////////////////////////////////////////////

//////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/12/23 09:34:38

// Design Name:

// Module Name: traffic\_light

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module module74X163(clk,clr,LOAD,enp,ent,d,q,rco);

input clk,clr,LOAD,enp,ent;

input[3:0] d;

output [3:0] q;

output rco;

reg [3:0] q=0;

reg rco=0;

always @ (posedge clk) begin

if (clr==0) q<=0;

else if (LOAD==0) q<=d;

else if ((enp==1) && (ent==1)) q<=q+1;

else q<=q;

end

always @ (q or ent) begin

if ((ent==1) && (q==15)) rco=1;

else rco=0;

end

endmodule

//分频模块

module clk\_divider(

input wire clk\_50m, // 50MHz 输入时钟

output reg clk\_half,clk9 // 1kHz 输出时钟

);

reg [25:0] counter1; // 26位计数器

reg [25:0] counter2;

// 初始化计数器

initial begin

counter1 = 0;

counter2 = 0;

clk\_half = 0;

end

// 每当50MHz时钟上升沿到来时，计数器增加

always @(posedge clk\_50m) begin

if (counter1 == 12500000) begin // 50,000 是 50MHz 到 1kHz 的分频系数

counter1 <= 0; // 重置计数器

clk\_half <= ~clk\_half; // 反转1kHz时钟信号

end else begin

counter1 <= counter1 + 1; // 计数器递增

end

end

always @(posedge clk\_50m) begin

if (counter2 == 10000000) begin // 50,000 是 50MHz 到 1kHz 的分频系数

counter2 <= 0; // 重置计数器

clk9 <= ~clk9; // 反转1kHz时钟信号

end else begin

counter2 <= counter2 + 1; // 计数器递增

end

end

endmodule

//延迟模块

module delay\_1hz(

input wire clk\_half, // 输入的1Hz时钟信号

output reg clk1, // 延迟后的1Hz时钟信号

output reg clk2,

output reg clk3,

output reg clk4,

output reg clk5,

output reg clk6,

output reg clk7,

output reg clk8

);

parameter CLK\_FREQ = 2; // 输入时钟频率为1Hz

parameter DELAY\_SEC1 = 2; // 延迟2秒

parameter DELAY\_COUNT1 = CLK\_FREQ \* DELAY\_SEC1; // 延迟的计数次数

parameter DELAY\_SEC2 = 1; // 延迟1秒

parameter DELAY\_COUNT2 = CLK\_FREQ \* DELAY\_SEC2; // 延迟的计数次数

parameter DELAY\_SEC3 = 7; // 延迟7秒

parameter DELAY\_COUNT3 = CLK\_FREQ \* DELAY\_SEC3; // 延迟的计数次数

parameter DELAY\_SEC4 = 6; // 延迟6秒

parameter DELAY\_COUNT4 = CLK\_FREQ \* DELAY\_SEC4; // 延迟的计数次数

parameter DELAY\_SEC5 = 9; // 延迟9秒

parameter DELAY\_COUNT5 = CLK\_FREQ \* DELAY\_SEC5; // 延迟的计数次数

parameter DELAY\_SEC6 = 8; // 延迟8秒

parameter DELAY\_COUNT6 = CLK\_FREQ \* DELAY\_SEC6; // 延迟的计数次数

parameter DELAY\_SEC7 = 14; // 延迟14秒

parameter DELAY\_COUNT7 = CLK\_FREQ \* DELAY\_SEC7; // 延迟的计数次数

parameter DELAY\_SEC8 = 13; // 延迟13秒

parameter DELAY\_COUNT8 = CLK\_FREQ \* DELAY\_SEC8; // 延迟的计数次数

reg [63:0] delay\_counter1; // 延迟计数器，根据需要调整位宽

reg [63:0] delay\_counter2;

reg [63:0] delay\_counter3;

reg [63:0] delay\_counter4;

reg [63:0] delay\_counter5;

reg [63:0] delay\_counter6;

reg [63:0] delay\_counter7;

reg [63:0] delay\_counter8;

// 初始化延迟计数器和输出时钟

initial begin

delay\_counter1 = 0;

delay\_counter2 = 0;

delay\_counter3 = 0;

delay\_counter4 = 0;

delay\_counter5 = 0;

delay\_counter6 = 0;

delay\_counter7 = 0;

delay\_counter8 = 0;

clk1=0;

clk2=0;

clk3=0;

clk4=0;

clk5=0;

clk6=0;

clk7=0;

clk8=0;

end

// 使用1Hz时钟信号来计数并创建延迟

always @(posedge clk\_half) begin

if (delay\_counter1 < DELAY\_COUNT1 - 1) begin

// 如果计数器未达到延迟次数，则递增计数器

delay\_counter1 <= delay\_counter1 + 1;

end else begin

// 一旦计数器达到延迟次数，开始输出延迟后的时钟信号

clk1 <= ~clk1; // 翻转时钟信号以保持1Hz频率

end

end

always @(posedge clk\_half) begin

if (delay\_counter2 < DELAY\_COUNT2 - 1) begin

// 如果计数器未达到延迟次数，则递增计数器

delay\_counter2 <= delay\_counter2 + 1;

end else begin

// 一旦计数器达到延迟次数，开始输出延迟后的时钟信号

clk2 <= ~clk2; // 翻转时钟信号以保持1Hz频率

end

end

always @(posedge clk\_half) begin

if (delay\_counter3 < DELAY\_COUNT3 - 1) begin

// 如果计数器未达到延迟次数，则递增计数器

delay\_counter3 <= delay\_counter3 + 1;

end else begin

// 一旦计数器达到延迟次数，开始输出延迟后的时钟信号

clk3 <= ~clk3; // 翻转时钟信号以保持1Hz频率

end

end

always @(posedge clk\_half) begin

if (delay\_counter4 < DELAY\_COUNT4 - 1) begin

// 如果计数器未达到延迟次数，则递增计数器

delay\_counter4 <= delay\_counter4 + 1;

end else begin

// 一旦计数器达到延迟次数，开始输出延迟后的时钟信号

clk4 <= ~clk4; // 翻转时钟信号以保持1Hz频率

end

end

always @(posedge clk\_half) begin

if (delay\_counter5 < DELAY\_COUNT5 - 1) begin

// 如果计数器未达到延迟次数，则递增计数器

delay\_counter5 <= delay\_counter5 + 1;

end else begin

// 一旦计数器达到延迟次数，开始输出延迟后的时钟信号

clk5<= ~clk5; // 翻转时钟信号以保持1Hz频率

end

end

always @(posedge clk\_half) begin

if (delay\_counter6 < DELAY\_COUNT6 - 1) begin

// 如果计数器未达到延迟次数，则递增计数器

delay\_counter6 <= delay\_counter6 + 1;

end else begin

// 一旦计数器达到延迟次数，开始输出延迟后的时钟信号

clk6 <= ~clk6; // 翻转时钟信号以保持1Hz频率

end

end

always @(posedge clk\_half) begin

if (delay\_counter7 < DELAY\_COUNT7 - 1) begin

// 如果计数器未达到延迟次数，则递增计数器

delay\_counter7 <= delay\_counter7 + 1;

end else begin

// 一旦计数器达到延迟次数，开始输出延迟后的时钟信号

clk7 <= ~clk7; // 翻转时钟信号以保持1Hz频率

end

end

always @(posedge clk\_half) begin

if (delay\_counter8 < DELAY\_COUNT8 - 1) begin

// 如果计数器未达到延迟次数，则递增计数器

delay\_counter8 <= delay\_counter8 + 1;

end else begin

// 一旦计数器达到延迟次数，开始输出延迟后的时钟信号

clk8 <= ~clk8; // 翻转时钟信号以保持1Hz频率

end

end

endmodule

module topmodule(

input clk\_50m, // 共用的时钟信号

output wire Gew,Rew,Yew,Gsn,Rsn,Ysn // 最终输出结果

);

wire clk1; // 延迟后的1Hz时钟信号

wire clk2;

wire clk3;

wire clk4;

wire clk5;

wire clk6;

wire clk7;

wire clk8;

wire clk9;

wire clk\_half;

clk\_divider uut15(

.clk\_50m(clk\_50m),

.clk\_half(clk\_half),

.clk9(clk9)

);

delay\_1hz uut16(

.clk\_half(clk\_half),

.clk1(clk1),

.clk2(clk2),

.clk3(clk3),

.clk4(clk4),

.clk5(clk5),

.clk6(clk6),

.clk7(clk7),

.clk8(clk8)

);

// 第一个74X163的信号

wire [3:0] q1;

wire rco1;

wire load1;

// 第二个74X163的信号

wire [3:0] q2;

wire rco2;

wire [3:0] q3;

wire rco3;

wire load3;

wire [3:0] q4;

wire rco4;

wire [3:0] q5;

wire rco5;

wire load5;

wire [3:0] q6;

wire rco6;

wire [3:0] q7;

wire rco7;

wire load7;

// 第二个74X163的信号

wire [3:0] q8;

wire rco8;

wire [3:0] q9;

wire rco9;

wire load9;

wire [3:0] q10;

wire rco10;

wire [3:0] q11;

wire rco11;

wire load11;

wire [3:0] q12;

wire rco12;

// 实例化第一个74X163

module74X163 uut1 (

.clk(clk1),

.clr(1'b1),

.LOAD(load1),

.enp(1'b1),

.ent(1'b1),

.d(4'b0000),

.q(q1),

.rco(rco1)

);

// 计算第一个74X163的LOAD信号

assign load1 = ~(q1[0] & q1[2] & q1[3]);

module74X163 uut3 (

.clk(clk1),

.clr(1'b1),

.LOAD(load3),

.enp(1'b1),

.ent(1'b1),

.d(4'b0000),

.q(q3),

.rco(rco3)

);

// 计算第一个74X163的LOAD信号

assign load3 = ~(q3[0] & q3[2] & q3[3]);

module74X163 uut5 (

.clk(clk3),

.clr(1'b1),

.LOAD(load5),

.enp(1'b1),

.ent(1'b1),

.d(4'b0000),

.q(q5),

.rco(rco5)

);

// 计算第一个74X163的LOAD信号

assign load5 = ~(q5[0] & q5[2] & q5[3]);

module74X163 uut7 (

.clk(clk5),

.clr(1'b1),

.LOAD(load7),

.enp(1'b1),

.ent(1'b1),

.d(4'b0000),

.q(q7),

.rco(rco7)

);

// 计算第一个74X163的LOAD信号

assign load7 = ~(q7[0] & q7[2] & q7[3]);

module74X163 uut9 (

.clk(clk5),

.clr(1'b1),

.LOAD(load9),

.enp(1'b1),

.ent(1'b1),

.d(4'b0000),

.q(q9),

.rco(rco9)

);

// 计算第一个74X163的LOAD信号

assign load9 = ~(q9[0] & q9[2] & q9[3]);

module74X163 uut11 (

.clk(clk7),

.clr(1'b1),

.LOAD(load11),

.enp(1'b1),

.ent(1'b1),

.d(4'b0000),

.q(q11),

.rco(rco11)

);

// 计算第一个74X163的LOAD信号

assign load11 = ~(q11[0] & q11[2] & q11[3]);

// 实例化第二个74X163

module74X163 uut2 (

.clk(clk2),

.clr(1'b1),

.LOAD(1'b0),

.enp(1'b1),

.ent(1'b1),

.d(4'b1111),

.q(q2),

.rco(rco2)

);

module74X163 uut4 (

.clk(clk2),

.clr(1'b1),

.LOAD(1'b0),

.enp(1'b1),

.ent(1'b1),

.d(4'b1111),

.q(q4),

.rco(rco4)

);

module74X163 uut6 (

.clk(clk4),

.clr(1'b1),

.LOAD(1'b0),

.enp(1'b1),

.ent(1'b1),

.d(4'b1111),

.q(q6),

.rco(rco6)

);

module74X163 uut8 (

.clk(clk6),

.clr(1'b1),

.LOAD(1'b0),

.enp(1'b1),

.ent(1'b1),

.d(4'b1111),

.q(q8),

.rco(rco8)

);

module74X163 uut10 (

.clk(clk6),

.clr(1'b1),

.LOAD(1'b0),

.enp(1'b1),

.ent(1'b1),

.d(4'b1111),

.q(q10),

.rco(rco10)

);

module74X163 uut12 (

.clk(clk8),

.clr(1'b1),

.LOAD(1'b0),

.enp(1'b1),

.ent(1'b1),

.d(4'b1111),

.q(q12),

.rco(rco12)

);

// 计算最终输出G

assign Gew = ((~q1[3] & ~q1[2]) | (~q1[3] & ~q1[1] & ~q1[0])) & q2[0];

assign Rew = (~q3[3]&q3[2]&q3[1]&q3[0])|(q3[3]&~q3[1])|(q3[3]&~q3[2])|(~q4[2]);

assign Yew = ~((q5[1])|(q5[2])|(q5[3]))&q6[1]&clk9;

assign Gsn = ((~q7[3] & ~q7[2]) | (~q7[3] & ~q7[1] & ~q7[0])) & q8[0];

assign Rsn = (~q9[3]&q9[2]&q9[1]&q9[0])|(q9[3]&~q9[1])|(q9[3]&~q9[2])|(~q10[2]);

assign Ysn = ~((q11[1])|(q11[2])|(q11[3]))&q12[1]&clk9;

endmodule