**实验四 综合电路设计实验报告**

**组长**：赵炫皓

**组员**：韦杰文

**组员**：陈勇斌

1. **实验题目**

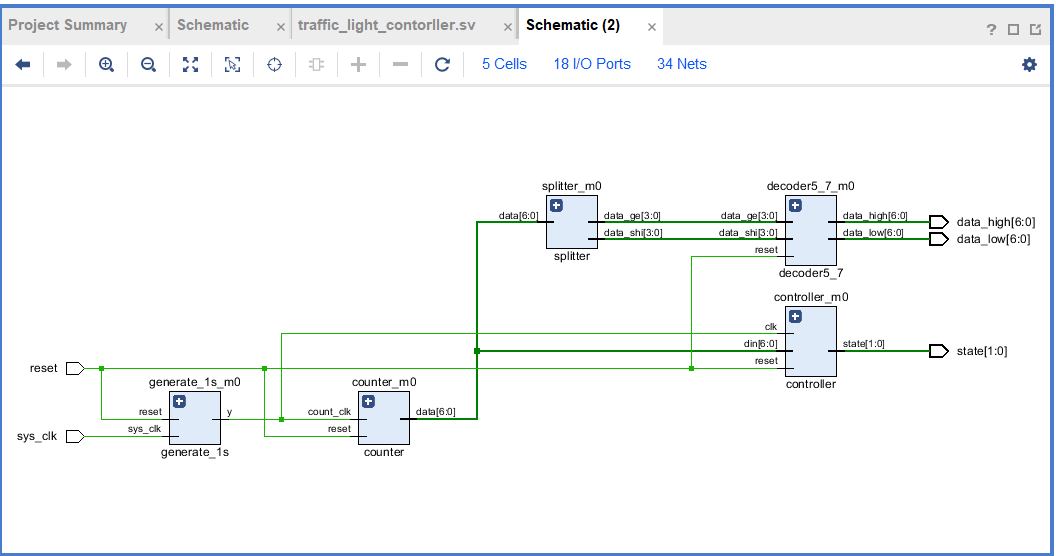
* 人行交通灯设计与实现（难度系数：0.8）

人行交通灯描述如下：

* + - “人行交通灯”用两只不同颜色的LED灯显示；
    - 红、绿两灯点亮时间比为30:20；
    - 红、绿两灯亮时，用两位数码管以“倒计时”方式显示剩余时间；
    - 最后三秒时“闪烁”LED灯，以表示临近结束
    - 开机自动运行，显示时间单位为“秒（S）”。

1. **电路设计**

**设计图：**

****

**交通灯控制器顶层框图：generate\_1s用来产生1Hz的时钟，controller输出红绿灯颜色状态，counter计数器用于输出倒计时和状态转移，splitter将分离10位和个位数，将7位二进制数转化为2个4位BCD码，decoder5\_7将4位BCD码转成7段数码管的显示数据**

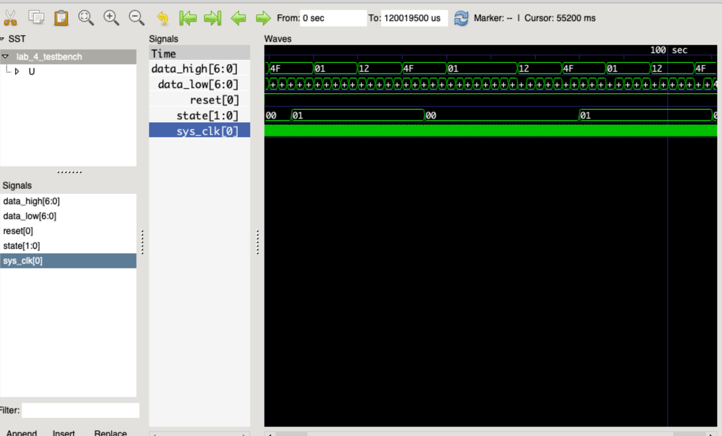
1. **电路实现**

module generate\_1s(sys\_clk, reset, y);  
 //定义输入输出端口  
 input sys\_clk, reset;  
 output y;  
   
 //内部寄存器定义  
 reg y;  
 reg [9 : 0] cnt;  
   
 //逻辑实现  
 always@(posedge sys\_clk or negedge reset)  
 begin  
 if(!reset)  
 begin  
 y <= 1'b0;  
 cnt <= 10'd0;  
 end  
 else  
 begin  
 if(cnt == 10'd999)  
 begin  
 y <= 1'b1;  
 cnt <= 10'd0;  
 end  
 else  
 begin  
 y <= 1'b0;  
 cnt <= cnt + 1;  
 end  
 end  
 end  
endmodule  
  
module counter(count\_clk, reset, data);  
 //输入输出端口定义  
 input count\_clk, reset;  
 output [6 : 0] data; //64~35, 34~5, 4~0循环减计数  
   
 //内部寄存器定义  
 reg [6 : 0] data;  
   
 //逻辑实现  
 always@(posedge count\_clk or negedge reset)  
 begin  
 if(!reset) data <= 7'd64; //异步复位  
 else if(data == 7'd0) data <= 7'd64;  
 else data <= data - 7'd1;  
 end  
endmodule  
  
module controller(clk, reset, din, state);  
 //定义输入输出端口  
 input clk, reset;  
 input [6 : 0] din;  
 output [1 : 0] state;  
   
 //内部寄存器定义  
 reg [1 : 0] state, current\_state, next\_state;  
   
 //状态编码  
 parameter red = 2'b00, green = 2'b01;  
   
 //时序逻辑实现状态转移  
 always@(posedge clk or negedge reset)  
 begin  
 if(!reset) current\_state <= red;  
 else current\_state <= next\_state;  
 end  
   
 //组合逻辑实现转移条件判断  
 always@(current\_state or din)  
 begin  
 case(current\_state)  
 red: next\_state = (din == 7'd50) ? green : red;  
 green: next\_state = (din == 7'd20) ? red : green;  
 endcase  
 end  
   
 //组合逻辑实现输出  
 always@(current\_state)  
 begin  
 case(current\_state)  
 red: state = 2'b00;  
 green: state = 2'b01;  
 endcase  
 end  
endmodule  
  
module splitter(data, data\_shi, data\_ge);  
 //输入输出端口定义  
 input [6 : 0] data;  
 output [3 : 0] data\_shi;  
 output [3 : 0] data\_ge;  
   
 //内部寄存器定义  
 reg [3 : 0] data\_shi, data\_ge;  
 reg [6 : 0] data\_display;  
   
 //显示数据转换  
 always@(data)  
 begin  
 if(data >= 7'd35) data\_display = data - 7'd35;  
 else if(data >= 7'd5) data\_display = data - 7'd5;  
 else data\_display = data;  
 end  
   
 //用左移加3法将7位二进制数转换为两个4位的BCD码  
 integer i;  
 always@(data)  
 begin  
 data\_shi = 4'd0;  
 data\_ge = 4'd0;  
   
 for(i = 6; i >= 0; i = i - 1)  
 begin  
 if(data\_ge >= 4'd5) data\_ge = data\_ge + 4'd3;  
 if(data\_shi >= 4'd5) data\_shi = data\_shi + 4'd3;  
   
 data\_shi = data\_shi << 1;  
 data\_shi[0] = data\_ge[3];  
 data\_ge = data\_ge << 1;  
 data\_ge[0] = data\_display[i];  
 end  
 end  
endmodule  
  
module decoder5\_7(reset, data\_shi, data\_ge, data\_high, data\_low);  
 //输入输出端口定义  
 input reset;  
 input [3 : 0] data\_shi, data\_ge;  
 output [6 : 0] data\_high, data\_low;  
   
 //内部寄存器定义  
 reg [6 : 0] data\_high, data\_low;  
   
 //译码个位数  
 always@(reset or data\_ge)  
 begin  
 if(!reset)  
 begin  
 data\_high <= 7'b0000110; //7'h06  
 data\_low <= 7'b0000001; //7'h01  
 end  
 else  
 begin  
 case(data\_ge)  
 4'd0: data\_low <= 7'b0000001; //7'h01  
 4'd1: data\_low <= 7'b1001111; //7'h4f  
 4'd2: data\_low <= 7'b0010010; //7'h12  
 4'd3: data\_low <= 7'b0000110; //7'h06  
 4'd4: data\_low <= 7'b1001100; //7'h4c  
 4'd5: data\_low <= 7'b0101100; //7'h2c  
 4'd6: data\_low <= 7'b0100000; //7'h20  
 4'd7: data\_low <= 7'b0001111; //7'h0f  
 4'd8: data\_low <= 7'b0000000; //7'h00  
 4'd9: data\_low <= 7'b0000100; //7'h04  
 default: data\_low <= 7'b0110000; //7'h48  
 endcase  
 end  
 end  
   
 //译码十位数  
 always@(reset or data\_shi)  
 begin  
 if(!reset)  
 begin  
 data\_high <= 7'b0000110; //7'h06  
 data\_low <= 7'b0000001; //7'h01  
 end  
 else  
 begin  
 case(data\_shi)  
 4'd0: data\_high <= 7'b0000001; //7'h01  
 4'd1: data\_high <= 7'b1001111; //7'h4f  
 4'd2: data\_high <= 7'b0010010; //7'h12  
 4'd3: data\_high <= 7'b0000110; //7'h06  
 default: data\_high <= 7'b0110000; //7'h48  
 endcase  
 end  
 end  
endmodule  
  
module traffic\_light\_controller(sys\_clk, reset, data\_high, data\_low, state);  
 //输入输出端口定义  
 input sys\_clk, reset;  
 output [1 : 0] state;  
 output [6 : 0] data\_high, data\_low;  
   
 //内部寄存器及连线定义  
 wire count\_clk;  
 wire [6 : 0] data;  
 wire [3 : 0] data\_shi, data\_ge;  
   
 //逻辑实现  
 generate\_1s generate\_1s\_m0(.sys\_clk(sys\_clk), .reset(reset), .y(count\_clk));  
 counter counter\_m0(.count\_clk(count\_clk), .reset(reset), .data(data));  
 controller controller\_m0(.clk(count\_clk), .reset(reset), .din(data), .state(state));  
 splitter splitter\_m0(.data(data), .data\_shi(data\_shi), .data\_ge(data\_ge));  
 decoder5\_7 decoder5\_7\_m0(.reset(reset), .data\_shi(data\_shi), .data\_ge(data\_ge), .data\_high(data\_high), .data\_low(data\_low));  
endmodule

1. **电路验证**
   1. **TestBench**

`timescale 1ms/100us  
`define sys\_clk\_half\_period 0.5 //系统时钟频率为1kHz，其半周期为0.5ms  
module lab\_4\_testbench();  
 // 输入输出端口定义  
 output[1:0] state;  
 output[6:0] data\_high, data\_low;  
  
 // 内部寄存器及连线定义  
 reg sys\_clk, reset;  
 wire count\_clk;  
 wire[6:0] data;  
 wire[3:0] data\_shi, data\_ge;  
 traffic\_light\_controller U(sys\_clk, reset, data\_high, data\_low, state);  
 /\*iverilog\*/  
 initial  
 begin  
 $dumpfile("wave.vcd");// 生成vcd文件名称  
 $dumpvars(0, lab\_4\_testbench);// tb模块名称  
 end  
 /\*iverlog\*/  
  
 // 产生测试信号  
 initial  
 begin  
 sys\_clk = 1'b0;  
 #5;  
 reset = 1'b1;  
 #5;  
 reset = 1'b0;  
 #9.5;  
 reset = 1'b1;  
 #120000 $stop;// 仿真120s停止  
 end  
  
 // 产生1kHz的系统时钟sys\_clk  
 always #`sys\_clk\_half\_period sys\_clk = ~sys\_clk;  
endmodule

* 1. **仿真结果**

****

**data\_high和data\_low是两个7位2进制数，表示数码管的十位和个位**

**sys\_clk是时钟信号，reset是复位.state是两位二进制数，其中红灯是00状态，绿灯是01状态，00与01之比应该是3:2。**

1. **电路上板**
2. **实验心得**

**进一步加深理解verilog硬件语言，初步掌握了模块化编程的思想。其中遇到用两位数码管以“倒计时”方式显示剩余时间的问题，一时无法下手。最终采用以下方法解决：首先，用七位循环计数器循环50s到20s，20s到0s再回到50s的过程。得到这样的七位二进制数。其次，将七位二进制数使用左移加3法得到2个4位BCD编码。最后，参照书本106页将BCD码转成7段码点亮数码管对应的段。这三个步骤分别对应电路设计中的三个模块。**