Final_Project

DIGITAL DESIGN

FINAL PROJECT REPORT

Final_Project

Student Name: 杨钰城 侯芳旻 杨祎勃

小组选题: A Real Car

PART 1: 成员分工以及贡献百分比，进度安排以及执行记录

1.1成员分工以及贡献百分比

·杨钰城：手动挡模块全部完成，项目总合成。

·杨祎勃：半自动模块，全自动模块全部完成。

·侯芳旻：Global State模块，VGA模块全部完成。

（该项目完成了所有的bonus,贡献比为1：1：1）

1.2进度安排

1.项目分析&分工安排：11.19-11.27

2.手动挡模块所有功能实现：11.27-12.15

3.Global State模块实现以及顶层模块合并修改：12.15-12.25

4.半自动以及全自动模块实现：12.25-12.29

5.VGA模块实现：12.29-1.1

6.报告以及视频制作：1.1-1.8

1.3执行记录

·11/19：开始讨论Project，确定选题为A Real Car并分析框架

·11/25：实验课老师讲解状态机，对其有深入了解后准备完成手动挡

·11/27：想要分析所给demo的原理，并且在所给demo里面实例化对象导致走入误区，耽误项目完成进度

·12/2:实验课老师讲解分频器于流水灯，为实现里程显示以及转向灯闪烁提供理论基础

·12/4：完成手动挡的状态变换

·12/11：由于使用if else较多导致逻辑混乱，从而使用状态机重新完成手动挡模块的状态变换

·12/12：实现手动挡的历程记录模块

·12/14：实现手动挡的转向灯闪烁模块

·12/15：着手七段数码显示管，实现手动挡的里程动态显示，并完成中期答辩

·12/17：完成global state模块

·12/18：将global state模块添加入顶层模块，并进行手动挡与顶层模块的修改与合并

·12/23：完成半自动模块，测试成功并与顶层模块整合

·12/26：完成自动模块，测试成功并与顶层模块整合

·12/29：完成VGA模块，测试成功并与顶层模块整合

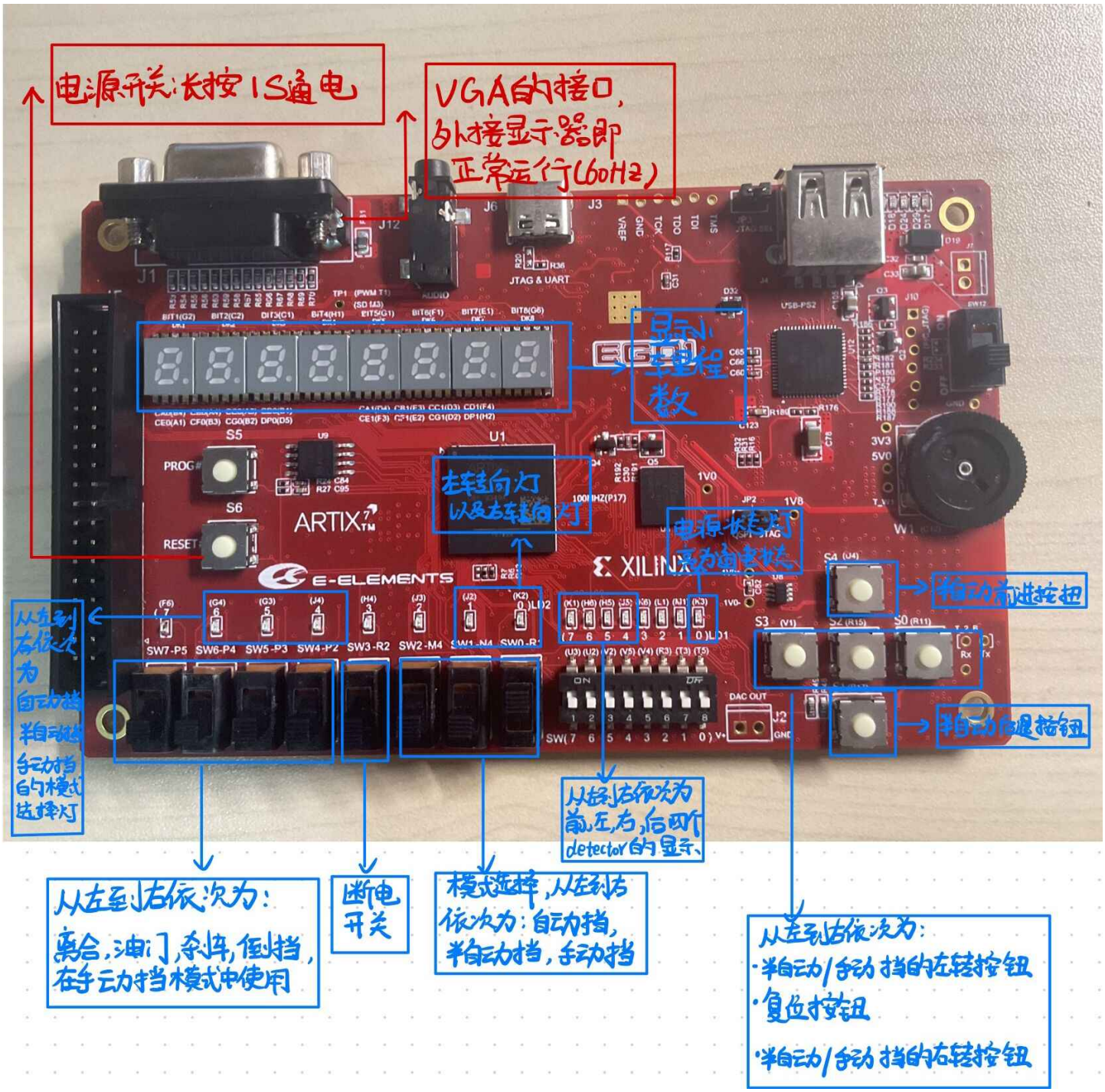
·1/3：开始着手报告

·1/6：拍摄视频并且完成报告，宣布本次项目结束

总结：在完成Project的过程中，由于开始对demo的不了解，导致浪费太多时间，以及对project难度的错误判断，出现了赶ddl的情况。得益于实验课老师对状态机，分频器以及对Project的指导；小组队员齐心协力，在后半程项目稳定推进，圆满的完成了本次Project的所有任务！

PART 2: 用户操作文档/图解

2.1用户操作图解



2.2用户使用手册

2.2.1：顶层端口规格：

输入端：系统时钟信号sys_clk，对应引脚为P17；重置开关 rst_n，对应引脚为R15；油门开关throttle，对应引脚为P4；离合开关clutch，对应引脚为P5；刹车开关brake，对应引脚为P3；倒车开关reverse，对应引脚为P2；端口绑定信号rx，对应引脚为N5；左转开关turn_left_signal，对应引脚为V1；右转开关turn_right_signal，对应引脚为R11；前进开关move_forward_signal，对应引脚为U4；后退开关move_backward_signal，对应引脚为R17；通电开关power_on，对应引脚为P15；断

电开关power_off，对应引脚为R2；模式选择开关[2:0] mode_signal，对应引脚为M4（mode_signal[2]），N4（mode_signal[1]），R1（mode_signal[0]）。

输出端：Uart模块传输信号tx，对应引脚为T4；通电/断电状态灯 power_on_led，对应引脚为K3；模式选择灯[2:0] mode_led，对应引脚为G4（mode_led[2]），G3（mode_led[1]），J4（mode_led[0]）；左转向灯left_led，对应引脚为J2；右转向灯right_led，对应引脚为K2；七段数码显示管使能信号[7:0] seg_enable，对应引脚为G2（seg_enable[7]），C2（seg_enable[6]），C1（seg_enable[5]），H1（seg_enable[4]），G1（seg_enable[3]），F1（seg_enable[2]），E1（seg_enable[1]），G6（seg_enable[0]）；前四个灯输出方式控制[7:0] seg_led1，对应引脚为B4（seg_led1[7]），A4（seg_led1[6]），A3（seg_led1[5]），B1（seg_led1[4]），A1（seg_led1[3]），B3（seg_led1[2]），B2（seg_led1[1]），D5（seg_led1[0]）；后四个灯输出方式控制[7:0] seg_led2，对应引脚为D4（seg_led2[7]），E3（seg_led2[6]），D3（seg_led2[5]），F4（seg_led2[4]），F3（seg_led2[3]），E2（seg_led2[2]），D2（seg_led2[1]），H2（seg_led2[0]）；后方障碍检测信号back_dector，对应引脚为J5；前方障碍检测信号front_dector，对应引脚为K2；左侧障碍检测信号left_dector，对应引脚为H6；右侧障碍检测信号right_dector，对应引脚为H5；VGA红色信号 [3:0] red，对应引脚为F5（red[0]），C6（red[1]），C5（red[2]），B7（red[3]）;VGA绿色信号[3:0] green，对应引脚为B6（green[0]），A6（green[1]），A5（green[2]），D8（green[3]）;VGA蓝色信号[3:0] blue，对应引脚为C7（blue[0]），E6（blue[1]），E5（blue[2]）E7（blue[3]）;显示器显像信号hsync，对应引脚为D7；vsync对应引脚为C4。

2.2.2：开发板输入设备：拨码开关，按钮

2.2.3：开发板输出设备：七段数码显示管，led灯，Uart软件

2.2.4：系统功能：

- 将比特流烧写到板子后进入断电状态（电源状态灯熄灭），在非手动挡的模式下七段数码显示管会显示“C”字样。
- 在四种模式中，VGA会在显示器的中间有不同颜色的色块显示：断电（红色），手动（黄色），半自动（绿色），自动（蓝色）。同时，在手动模式下，色块下方会有白色8位数字显示当前的里程。
- 长按通电开关（高电平有效）一秒后小车进入通电状态，开关灯将亮起。此时可根据模式选择开关（高电平有效，每次只能选择一种模式）选择手动挡，半自动挡，自动挡三个模式，选择不同模式后对应的模式灯也会亮起。也可通过断电开关使系统断电（高电平有效）
- 在手动挡的模式下，七段数码显示管会显示小车所走里程。
    - 根据汽车运行的规则，首先踩离合与油门（高电平有效）使车点火，松开离合后（此时还在踩油门）车可以前进。接着松开油门，车停止。
    - 踩离合与刹车（高电平有效）使车挂倒挡，再踩油门可实现倒车。不正确的点火与挂倒挡操作均会使车断电（就是熄火）。

- 踩刹车（高电平有效）会让车停下并进入熄火状态，需要重新点火才能通过油门实现前进或倒车。
- 在点火状态下左转右转按钮（高电平有效）会让车改变方向，转向灯会以0.5秒的频率闪烁。

- 在半自动模式下，当小车前方路线唯一时，小车会自动沿着当前道路行驶；当小车遇到岔路口时，小车会停下等待操作者指令；当小车遇到死路时会自动调头。
- 在自动模式下，小车会自动从起点走到终点，不需要来自用户的任何操作。

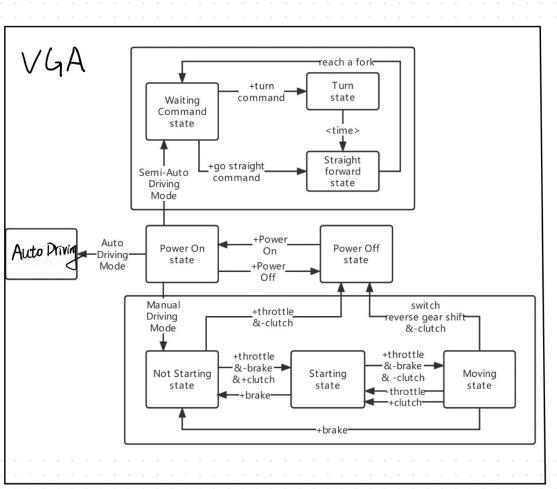
2.3视频演示

视频链接如下：【数字逻辑Project】

https://www.bilibili.com/video/BV1PG4y117ia/?share_source=copy_web&vd_source=0f39537c7767e932a502a4e6badf0663

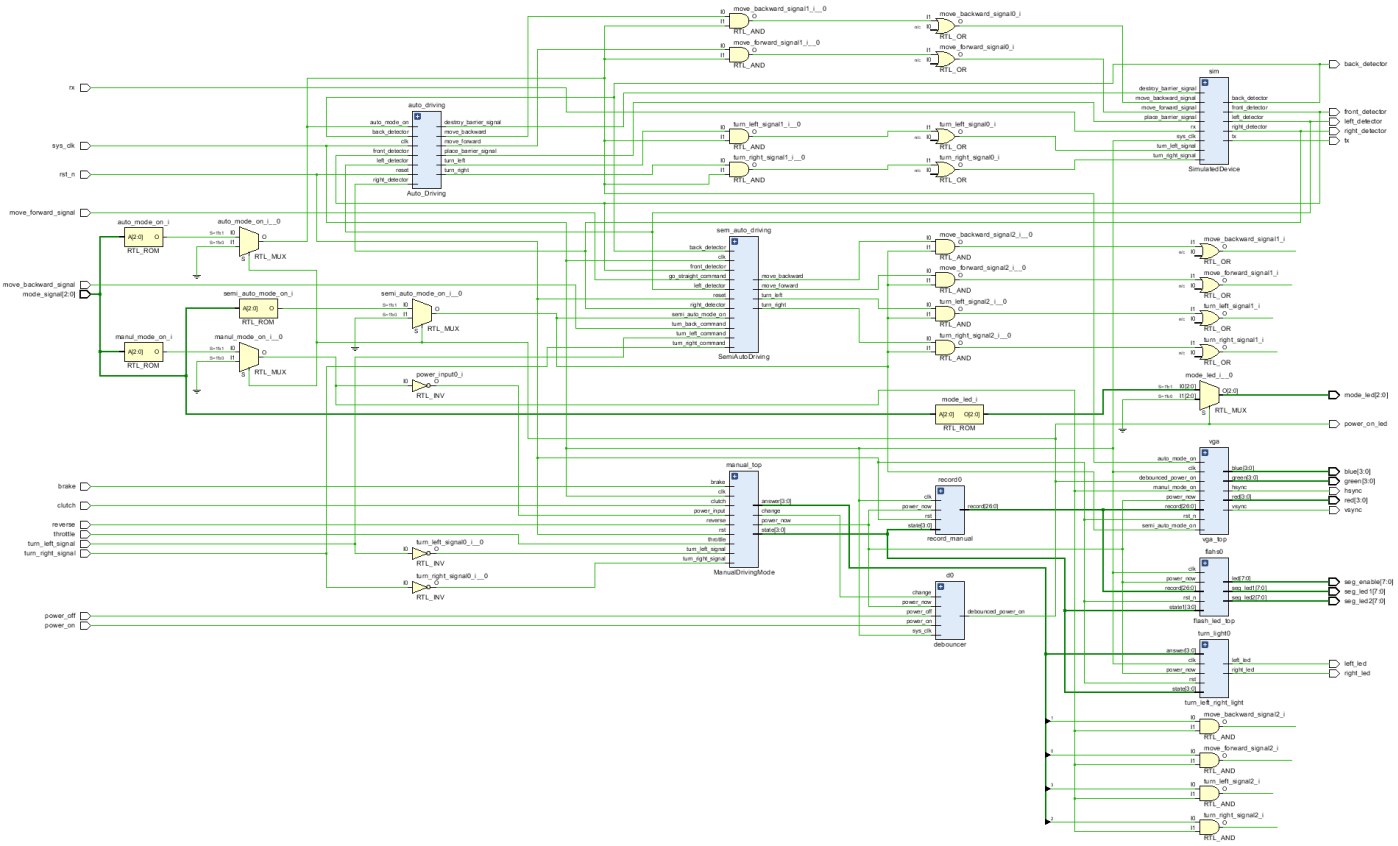
PART 3: 系统结构设计

3.1状态流程图



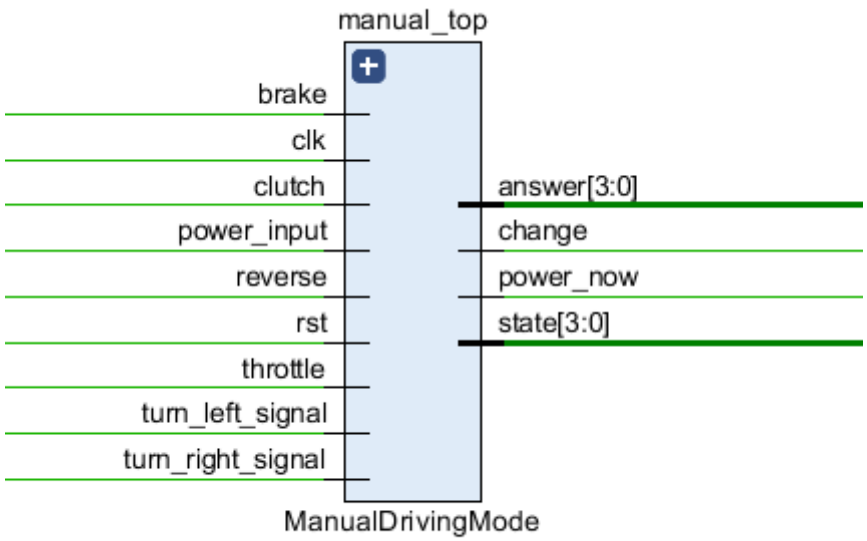
项目采用的状态迁移图如图，由于在电源开关打开后VGA便正常运行，故将VGA状态划分在最外层。

3.2电路结构图



由于图片不够清晰，将在以下部分详细讲解各个模块。顶层模块的接口讲解参照上文“顶层端口规格”。

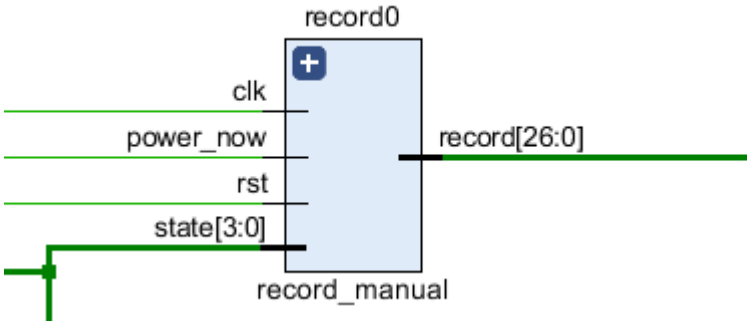
·手动挡



输入接口clk是时钟信号，power_input是手动挡的模式选择信号（使能信号），break,clutch,reverse,rst,throttle,turn_left_signal,turn_right_signal分别对应刹车、离合、倒挡开关、重置键、油门、左转信号、右转信号。

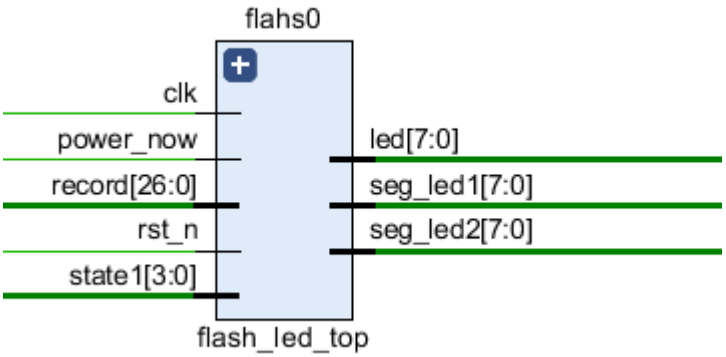
输出接口answer[3:0]是小车当前状态（answer[0]为前进，answer[1]为后退，answer[2]为右转，answer[3]为左转），change表示是外部断电还是手动挡操作失误导致小车断电，power_now表示小

车现在的状态（通电还是断电），state[3:0]表示小车现在所处状态（4'b0001为unstarting状态，4'b0010为starting状态，4'b0100为moving状态，4'b1000为power_off状态)。



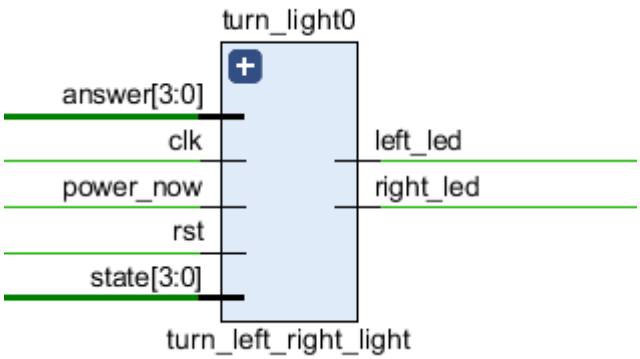
输入接口clk是时钟信号，power_now是小车现在的通电/断电状态，rst是重置信号，state[3:0]表示小车现在所处状态（4'b0001为unstarting状态，4'b0010为starting状态，4'b0100为moving状态，4'b1000为power_off状态)。

输出接口record[26：0]表示小车所走里程。



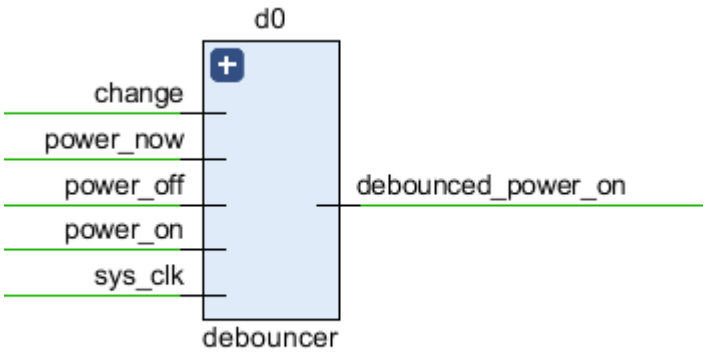
输入接口clk是时钟信号，power_now是小车现在的通电/断电状态，record[26：0]表示小车所走里程，rst_n是重置信号，state[3:0]表示小车现在所处状态（4'b0001为unstarting状态，4'b0010为starting状态，4'b0100为moving状态，4'b1000为power_off状态)。

输出接口led[7:0]控制八位数码管是否显示，seg_led1[7:0],seg_led2[7:0]分别控制前四个数码管与后四个数码管显示的内容。



输入接口answer[3:0]是小车当前状态（answer[0]为前进，answer[1]为后退，answer[2]为右转，answer[3]为左转），clk是时钟信号，power_now是小车现在的通电/断电状态，rst是重置信号，state[3:0]表示小车现在所处状态（4'b0001为unstarting状态，4'b0010为starting状态，4'b0100为moving状态，4'b1000为power_off状态)。

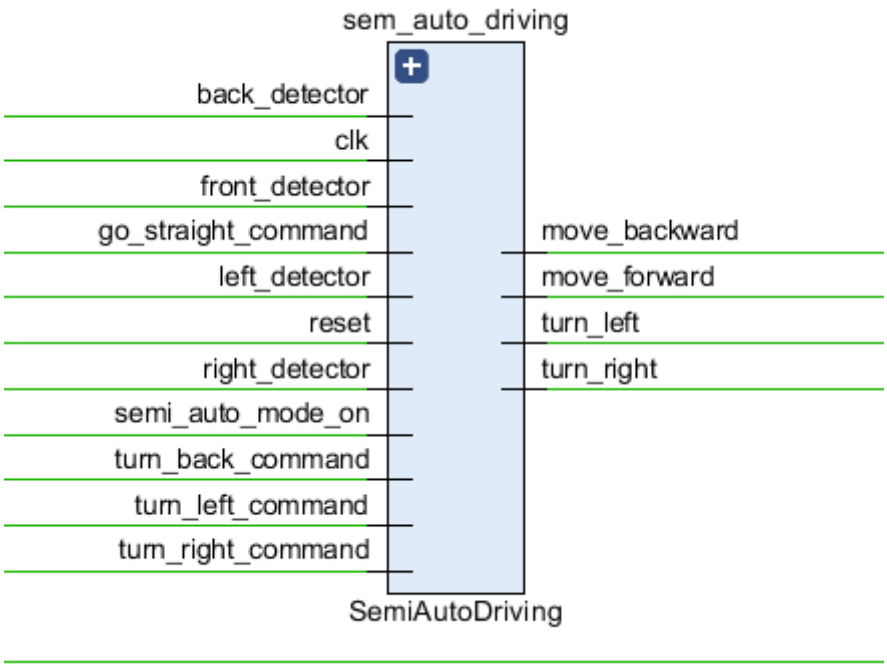
输出信号left_led是左转向灯闪烁信号。right_led是右转向灯闪烁信号。



输入接口change表示是外部断电还是手动挡操作失误导致小车断电的信号，power_now是小车现在的通电/断电状态，power_off是断电信号，power_on是通电信号，sys_clk是时钟信号。

输出信号debounced_power_on是消抖后的通电信号。

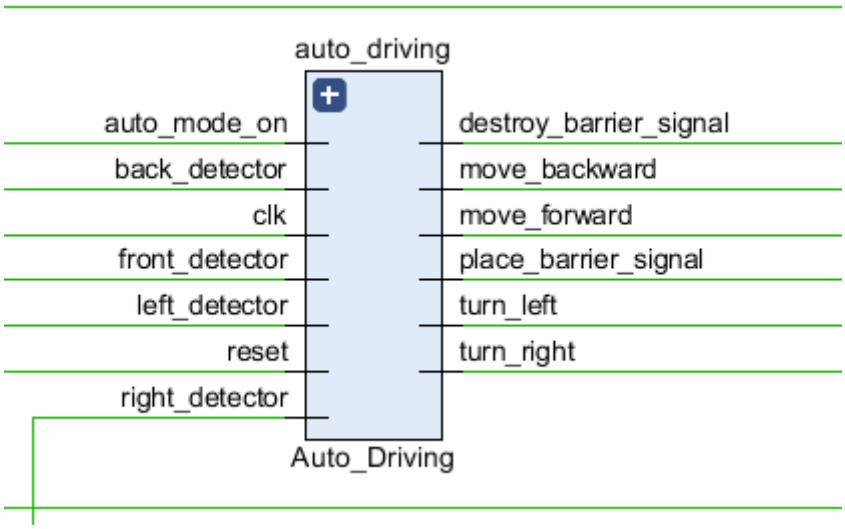
·半自动挡



输入端，clk是时钟信号；semi_auto_mode_on是半自动档的模式选择信号（使能信号）；reset是重置信号（用于重置小车当前状态）；front_detector、left_detector和right_detecor分别是前、左、右方向的障碍检测信号；go_straight_command、turn_back_command、turn_left_command、turn_right_command分别是来自用户的前进、掉头、左转、右转的指令信号。

输出端，move_forward连接SimulatedDevice模块的move_forward_signal，是小车的前进信号；move_backward连接SimulatedDevice模块的move_backward_signal，是小车的后退信号；turn_left连接SimulatedDevice模块的turn_left_signal，是小车的左转信号；turn_right连接SimulatedDevice模块的turn_right_signal，是小车的右转信号；

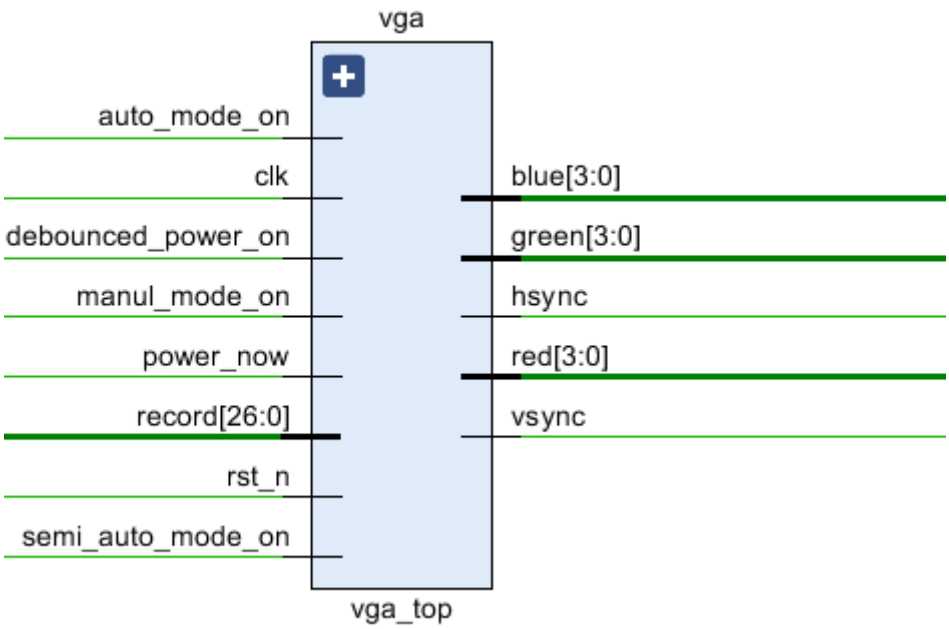
·自动挡



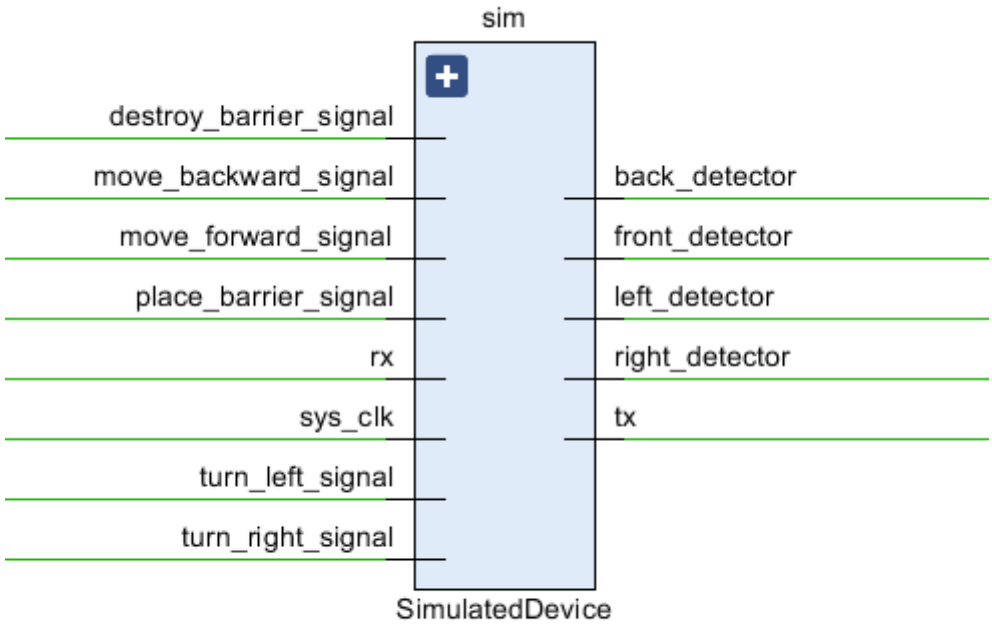
输入端，clk是时钟信号；auto_mode_on是半自动档的模式选择信号（使能信号）；reset是重置信号（用于重置小车当前状态）；front_detector、back_detector、left_detector和right_detecor分别是前、后、左、右方向的障碍检测信号；

输出端，move_forward连接SimulatedDevice模块的move_forward_signal，是小车的前进信号；move_backward连接SimulatedDevice模块的move_backward_signal，是小车的后退信号；turn_left连接SimulatedDevice模块的turn_left_signal，是小车的左转信号；turn_right连接SimulatedDevice模块的turn_right_signal，是小车的右转信号；place_barrier_signal连接SimulatedDevice模块的place_barrier_signal，是用于放置信标的信号；destroy_barrier_signal连接SimulatedDevice模块的destroy_barrier_signal，是用于摧毁信标的信号。

·VGA



·SimulatedDevice



输入信号destroy_barrier_signal是摧毁信标信号，move_backward_signal是后退信号，move_forward_signal是前进信号，place_barrier_signal是放置信标信号，rx直接与端口绑定，

sys_clk是时钟信号，turn_left_signal是左转信号，turn_right_signal是右转信号。

输出信号back_dector是后方障碍检测信号，front_dector是前方障碍检测信号，left_dector是左侧障碍检测信号，right_dector是右侧障碍检测信号，tx与uart模块连接。

3.3详细设计

模块名	核心功能	备注
GTR	汇总其他模块的顶层模块	在该模块中实现消抖以及Globa
ManualDrivingMode	手动挡状态转换，信号输出模块	该模块的输出会用于有关手动挡
record_manual	实现手动挡中里程的记录，每0.5s加一	该里程将用于七段数码管的显示
flash_led_top	里程显示顶层模块，前七个灯显示小车里程	非手动挡状态或者断电状态显示
counter	里程显示模块的分频器	
flash_led_ctrl	里程显示模块控制七段数码显示管的使能以及显示内容	采用流水灯的方式实现动态变化
turn_left_right_light	实现左转右转时灯的闪烁，闪烁频率为0.5s	采用2hz的分频器
SemiAutoDriving	半自动档模块，实现小车半自动驾驶	
AutoDriving	自动挡模块，实现小车自动驾驶	
vga_top	VGA的顶层模块，实现VGA显示	
num_switch	vga_top的子模块，实现在当前位置显示不同数字时的不同VGA输出	

PART 4: 核心代码说明

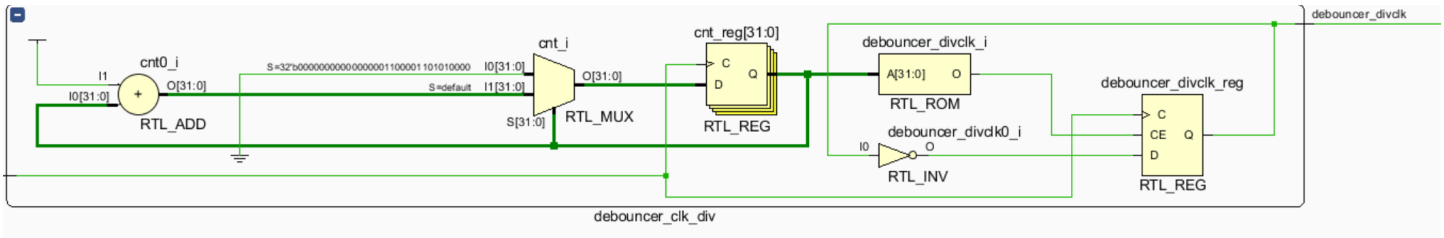
4.1顶层模块

```
module GTR (input sys_clk,
                input rst_n,
                input throttle,    // connected to the switch control throttle
                input clutch,      // connected to the switch control cluth
                input brake,       // connected to the switch control brake
                input reverse,     // connected to the switch control reverse
                input rx,  // connected to the UART RX pin. recevier information from puzzle
                output tx, // connected to the UART TX pin
                input turn_left_signal,
                input turn_right_signal,
                input move_forward_signal,
                input move_backward_signal,

                output reg power_on_led = 0,
                output reg [2:0] mode_led, // connected to the led indicating the current mode
                output left_led,   // connected to the led indicating if the car is turning left or right
                output right_led,
                input power_on,   // connected to the power on button
                input power_off, // connected to the power off button
                input [2:0] mode_signal, // connected to 3 mode switches
                output [7:0] seg_enable,    // control the output of the 7seg tubes
                output[7:0] seg_led1,
                output [7:0] seg_led2,
                output back_detector,  // get information from the puzzle of whether these direction have block
                output left_detector,
                output front_detector,
                output right_detector,
                output [3:0] red,    // information of the VGA color display
               output [3:0] green,
               output [3:0] blue,
                output hsync,  // connected to the VGA hsync and vsync pin
                output vsync
                );
```

4.1.1 分频

首先通过分频器获得防抖模块需要的合适频率

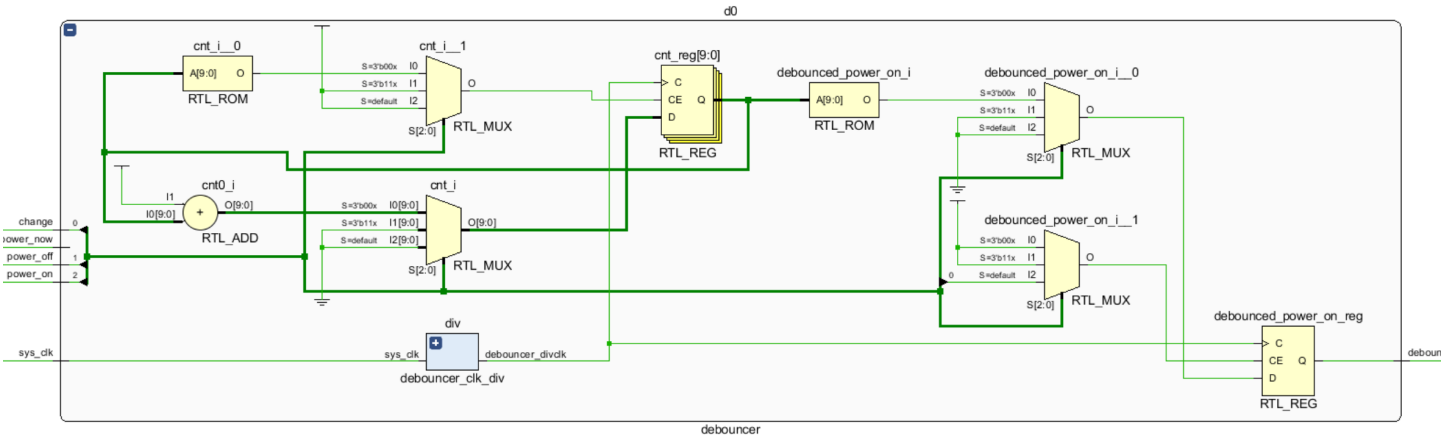
```
module debouncer_clk_div (
    input sys_clk,
    output reg debouncer_divclk
    );
    reg[31:0] cnt = 0;
    always@(posedge sys_clk)//1000HZ
    begin
        if (cnt == 32'd50000)
        begin
            debouncer_divclk <= ~debouncer_divclk;
            cnt              <= 0;
        end
        else
            cnt <= cnt+1'b1;
    end
endmodule
```



4.1.2 按键防抖

通过利用一个计数器，使得在时间到达1s后再将表示按键开关的debounced_power_on 变量置1，实现了按键的防抖。

```
always @(posedge debouncer_divclk) begin
    casex ({power_on,power_off,change})
        3'b00X:
        // press the power_on button
        begin
            case (cnt)
                10'd1000: debounced_power_on <= 1'b1;
                default:
                begin
                    cnt                <= cnt+1'b1;
                    debounced_power_on <= 1'b0;
                end
            endcase
        end
        3'b11X:
        // press the power_off button
        begin
            cnt <= 0;
            debounced_power_on <= 1'b0;
        end
        default:
        if(change==1)begin
             cnt <= 0;
            debounced_power_on <= 1'b0;
        end
        // none of the button is pressed or both of the button is pressed.
        // At this circumstance, we can NOT  declear whether it is power on or off.
        else
         cnt <= 0;
        // press 2 button at the same time
    endcase
    //   if(power_now==1'b1)begin
    //         debounced_power_on=1'b0;
    //       end

end
```

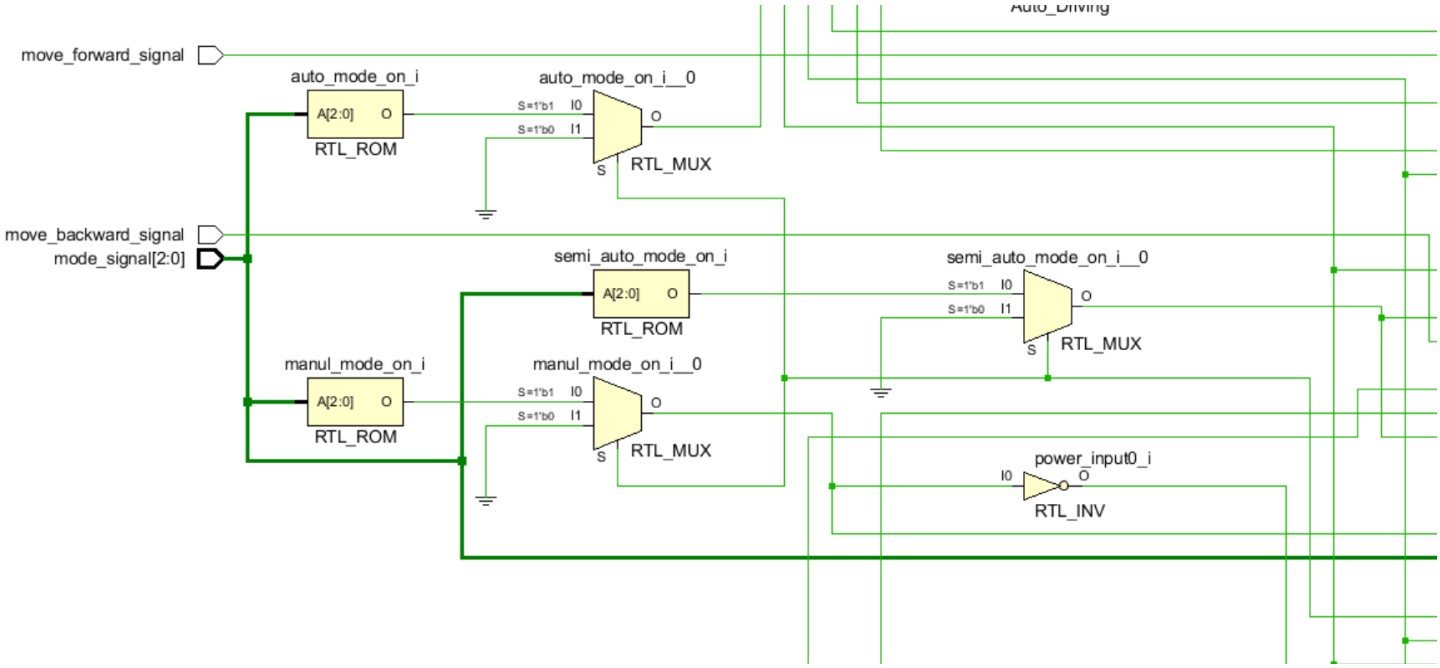


4.1.3 模式选择

在模式选择中，通过对开关按键和模式开关在不同值时的不同case中，对开关灯，模式灯等进行不同的赋值操作，实现了模式选择的功能。

```
always @(*) begin
    case ({debounced_power_on})
        1'b1:
        begin
            power_on_led = 1'b1;
            // should be one
            case (mode_signal)
                3'b001:
                // the manual mode is on.
                begin
                    mode_led       = 3'b001;
                    manul_mode_on     = 1'b1;
                    semi_auto_mode_on = 0;
                    auto_mode_on      = 0;

                end
                3'b010:
                begin
                   mode_led       = 3'b010;
                    manul_mode_on     = 0;
                    semi_auto_mode_on = 1'b1;
                    auto_mode_on      = 0;
```

```
                auto_mode_on      = 0;
            end
            3'b100:
            begin
              mode_led       = 3'b100;
                manual_mode_on     = 0;
                semi_auto_mode_on = 0;
                auto_mode_on       = 1'b1;
            end
            default:
            begin
                mode_led       = 3'b000;
                manual_mode_on     = 0;
                semi_auto_mode_on = 0;
                auto_mode_on       = 0;
            end
        endcase
    end
    // 10 is the state when power on.
    1'b0:
    begin
        power_on_led      = 0;
        mode_led       = 3'b000;
        manual_mode_on     = 0;
        semi_auto_mode_on = 0;
        auto_mode_on       = 0;

    end
  endcase
end
```



4.2手动挡部分

4.2.1手动挡状态切换

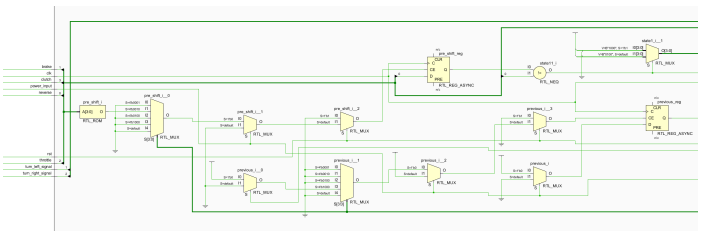
```
module ManualDrivingMode(
    input clk, //bind to P17 pin (100MHz system clock)
    input rst,//重置按钮
    input power_input,//输入通电还是断电状态
    input throttle,//油门
    input clutch,//离合
    input brake,//刹车

    input reverse,//倒车
    input turn_left_signal,//左转信号
    input turn_right_signal,//右转信号
    output reg change,//断电发生在外部还是内部
    output reg [3:0]answer,//依次输出左转，右转，后退，前进信号
    output  [3:0] state,//输出小车目前手动挡所处状态
    //4'b0001为unstarting状态，4'b0010为starting状态，4'b0100为moving状态，4'b1000为power_off状态
    output reg  power_now//输出小车当前状态0是通电，1是断电
    );
```

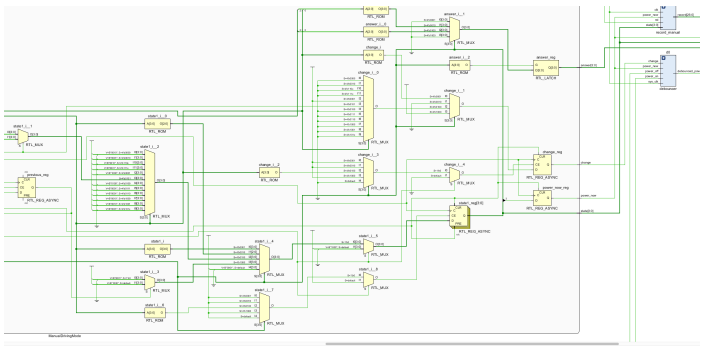
```
    //状态机实现各个状态的转移
      reg previous;
      reg pre_shift;
       reg [3:0] state1= 4'b0001;
    parameter unstarting=4'b0001, starting=4'b0010, moving=4'b0100,power_off=4'b1000;
always @(posedge clk ,negedge rst) begin
        if(rst)begin
            state1<= 4'b0001;
            power_now<=0;
            change<=0;
            end
       else if(power_input==1'b0)begin
         case(state1)
        4'b0001:casex({clutch,throttle,brake,reverse})//未启动
                        4'b000X :begin state1<=unstarting;change<=0;
                        end
                        4'b001X :begin state1<=unstarting;change<=0;
                        end
                        4'b010X : begin state1<=power_off;change<=1;
                        end
                        4'b011X : begin state1<=unstarting;change<=0;
                        end
                        4'b10XX : begin state1<=unstarting;change<=0;
                        end
                        4'b110X : begin state1<=starting;change<=0;
                        end
                        4'b111X : begin state1<=unstarting;change<=0;
                        end
                endcase
         4'b0010:casex({clutch,throttle,brake,reverse})//启动
                        4'b000X :begin
                                 state1<=starting;pre_shift<=reverse;change<=0;
                                 end
                        4'b001X : begin state1<=unstarting;change<=0;
                        end
                        4'b010X : begin state1<=moving;pre_shift<=reverse;change<=0;
                                  end
                        4'b011X : begin state1<=unstarting;change<=0;
                        end
                        4'b1X0X : begin state1<=starting;change<=0;
                        end
                        4'b1X1X : begin state1<=unstarting;change<=0;
                        end
                 endcase

       4'b0100:casex({clutch,throttle,brake,reverse})
                        4'b0000 : begin state1<=starting;change<=0;
                        end
                        4'b0010 : begin state1<=unstarting;change<=0;
                        end
                        4'b0001 : begin state1<=power_off;change<=1;
                        end
                        4'b0101 : begin if(pre_shift!=reverse)begin
                                   state1<=power_off;change<=1;
                                    end
```

```
                                    end
                                    else begin
                                    state1<=moving;change<=0;
                                    end
                                end
                        4'b0100 : begin state1<=moving;change<=0;
                        end
                        4'b0111 :begin state1<=unstarting;change<=0;
                        end
                        4'b0110 :begin state1<=unstarting;change<=0;
                        end
                        4'b1000 : begin state1<=starting;change<=0;
                        end
                        4'b1001 : begin state1<=starting;change<=0;
                        end
                        4'b101X : begin state1<=unstarting;change<=0;
                        end
                        4'b110X : begin state1<=starting;change<=0;
                        end
                        4'b111X :begin state1<=unstarting;change<=0;
                        end
                endcase
        4'b1000:casex({clutch,throttle,brake,reverse})
                        4'bXXXX :begin if(previous==0)begin
                                        state1<=unstarting;
                                        previous<=1;
                                        change<=0;
                        end         else
                                        state1<=power_off;
                                        change<=1;
                                        end
                endcase

         default :
                        state1<=unstarting;
          endcase
          power_now<=state1[3];
          end
        else begin
          previous<=0;
          state1<=power_off;
          power_now<=state1[3];
        end

end //判断状态
```

```
//在各个状态下对输入信号进行处理得到最终的输出信号
always @(*)begin
        case(state1)
        unstarting:casex({turn_right_signal,turn_left_signal,reverse})
                                3'bXXX :answer=4'b0000;
                            endcase
        starting:casex({turn_right_signal,turn_left_signal,reverse})
                                3'b00X :answer=4'b0000;
                                3'b10X :answer=4'b1000;
                                3'b01X :answer=4'b0100;
                                3'b11X :answer=4'b0000;
        endcase
        moving:case({turn_right_signal,turn_left_signal,reverse})
                                3'b000 :answer=4'b0001;
                                3'b001 :answer=4'b0010;
                                3'b010 :answer=4'b0101;
                                3'b011 :answer=4'b0110;
                                3'b100 :answer=4'b1001;
                                3'b101 :answer=4'b1010;
                                3'b110 :answer=4'b0001;
                                3'b111 :answer=4'b0010;
        endcase
        power_off:casex({turn_right_signal,turn_left_signal,reverse})
                                3'bXXX :answer=4'b0000;
       default :
                        answer=4'b0000;
        endcase
        endcase
    end
assign state=state1;
endmodule
```

结构图如下

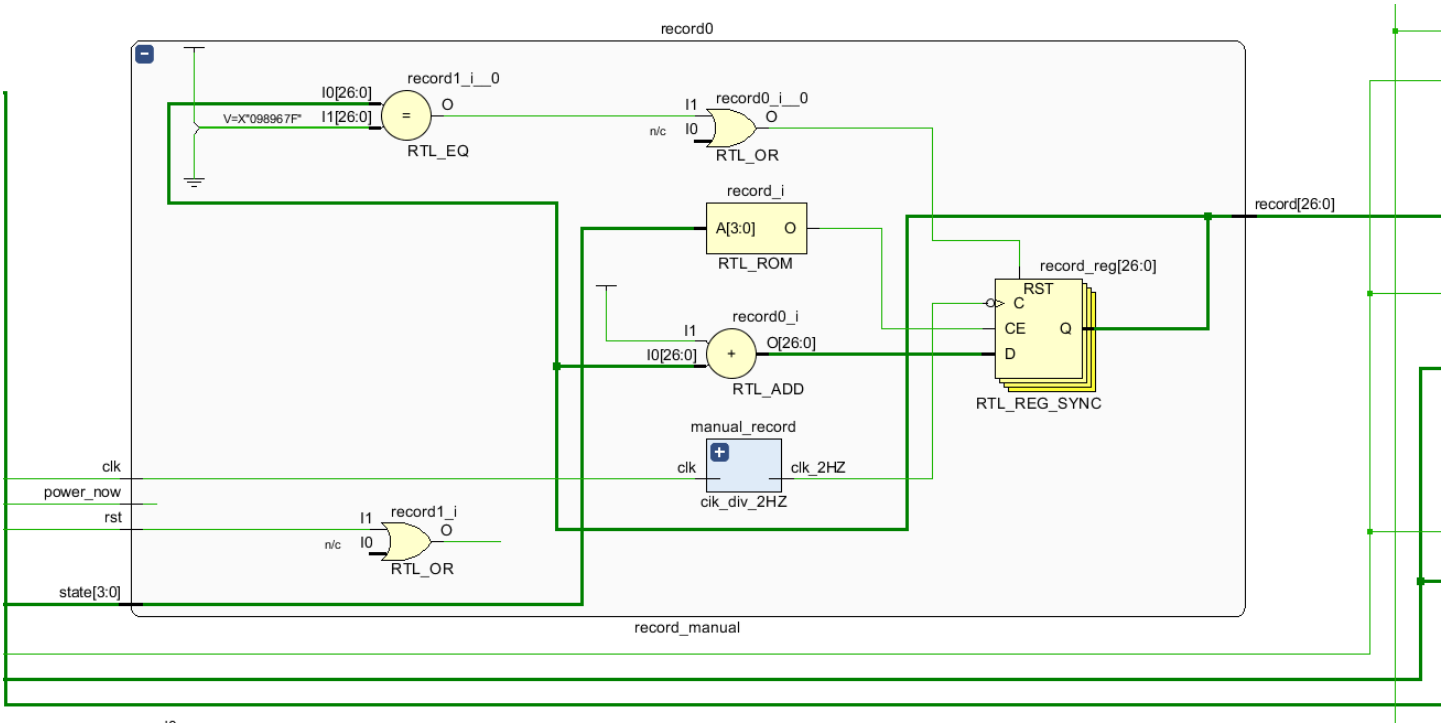




4.2.2里程记录

```
module record_manual(//我设计的是每0.5s里程加一

                    input clk,
                    input rst,
                    input power_now,//必须是通电状态才有效

                    input [3:0] state,//输入手动挡的状态
                    output reg[26:0] record  );
    wire clk_2hz;
    cik_div_2HZ manual_record(
                            .clk(clk),
                            .clk_2HZ(clk_2hz)
    );

  always@(negedge clk_2hz )begin
      if(power_now||rst||record==27'd999_9999)begin
          record<=0;
       end
       else if(state==4'b0100) begin
          record<=record+1;
       end
    end


endmodule


 module cik_div_2HZ(input clk,output reg clk_2HZ  );//2hz分频器
parameter  period = 50_000000;
reg [31:0] div2hz_cnt=0;
always@(posedge clk )
begin
    if( div2hz_cnt==12500000)
    begin
        clk_2HZ=~clk_2HZ;
        div2hz_cnt=0;
    end
    else
        div2hz_cnt=div2hz_cnt+1'b1;
end


endmodule
```

结构图如下



4.2.3转向灯闪烁

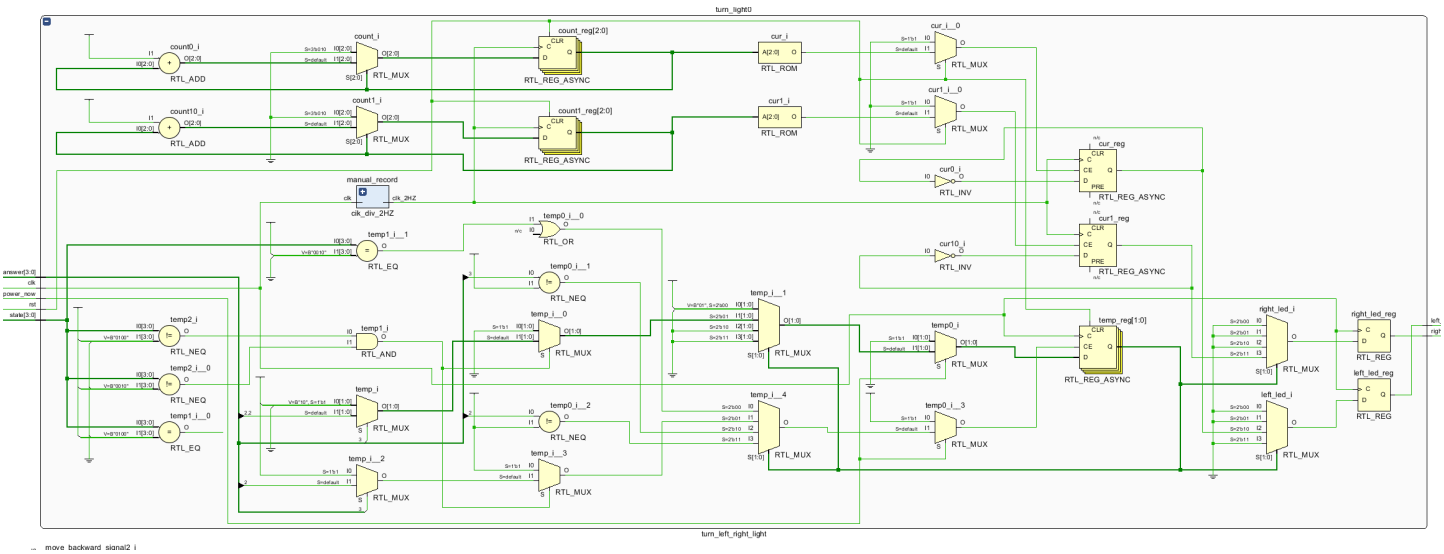
```
module turn_left_right_light(
                        input rst,
                        input clk,
                        input power_now,//小车现在是通电还是断电状态
                        input [3:0] state,//4'b0001为unstarting状态，4'b0010为starting状态，4'b0100为moving状态，4'b1000为power_off状态
                        input [3:0]answer,//左转，右转，后退，前进信号
                        output reg left_led,
                        output reg right_led

                            );
  wire clk_2hz;
  reg cur=0;
  reg cur1=0;
  reg[2:0] count;
  reg[2:0] count1;
  reg [1:0] temp;
        cik_div_2HZ manual_record(//和里程记录使用的分频器一样
                                .clk(clk),
                                .clk_2HZ(clk_2hz)
       );
```

```
//调整闪烁频率
      always @(posedge clk_2hz or negedge rst) begin
            if(rst)begin
            count=3'd00;
            end
      else if( count==2)
      begin
            cur=~cur;
            count=0;
      end
      else
            count=count+1'b1;

      end
      always @(posedge clk_2hz or negedge rst) begin
            if(rst)begin
            count1=3'd00;
            end
      else if( count1==2)
      begin
            cur1=~cur1;
            count1=0;
      end
      else
            count1=count1+1'b1;
      end
```

```
//使用状态机划分状态
        always @(posedge clk or negedge rst) begin
                if(rst||power_now) begin
                        temp<=2'b00;
                end
                else begin
                case (temp)
                2'b00:begin
                        if(state==4'b0100||state==4'b0010)begin
                                temp<=2'b01;
                        end

                        end
                2'b01:begin
                        if(state!=4'b0100&&state!=4'b0010)begin
                                temp<=2'b00;
                        end
                         else if(answer[3]==1)begin
                                 temp<=2'b10;
                        end
                         else if(answer[2]==1)begin
                                 temp<=2'b11;
                        end
                end
                 2'b10:begin
                        if(answer[3]!=1)begin
                                temp<=2'b00;
                        end
                end
                2'b11:begin
                        if(answer[2]!=1)begin
                                temp<=2'b00;
                        end
                end
                endcase
        end
        end
```

```
        //对不同状态进行赋值
    always @(posedge clk) begin
        case(temp)
                2'b00:begin
                        left_led<=0;
                        right_led<=0;
                end
                2'b01:begin
                        left_led<=0;
                        right_led<=0;
                end
                2'b10:begin
                        left_led<=cur;
                        right_led<=0;
                end
                2'b11:begin
                        left_led<=0;
                        right_led<=cur1;
                end
        endcase
    end
```

结构图如下



4.2.4里程显示

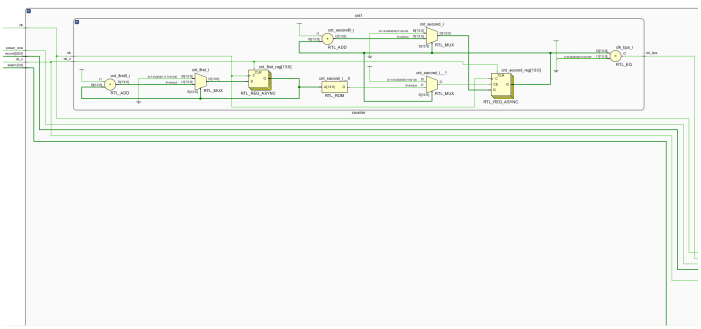
```
module flash_led_top(//里程显示
    input clk,
    input rst_n,
    input power_now,
    input [3:0]state1,
    input [26:0] record,
    output  [7:0] led,//八个灯的使能，用流水灯
    output [7:0] seg_led1,//前四个灯输出控制
    output [7:0] seg_led2//后四个灯输出控制

    );
    counter cnt1(.clk(clk),.rst_n(rst_n),.clk_bps(clk_bps));//分频产生视觉暂留的效果
    flash_led_ctrl fled_ctrl(.clk(clk),
                             .rst_n(rst_n),
                             .clk_bps(clk_bps),
                             .power_now(power_now),
                             .state1(state1),
                             .record(record),
                             .led(led),
                             .seg_led1(seg_led1),
                             .seg_led2(seg_led2));



endmodule


//////////////////////////////////////////////////////////////////////////////////
module flash_led_ctrl(//流水灯模块
    input clk,
    input rst_n,
    input clk_bps,
    input power_now,
    input [3:0] state1,
    input [26:0] record,
    output reg [7:0] led,//八个灯的使能，用流水灯
    output [7:0] seg_led1,
    output [7:0] seg_led2
    );
    reg [3:0] led1,led2,led3,led4,led5,led6,led7;//led8是最高位，将记录的里程划分成七位
    reg [3:0] in1,in2;
    reg[3:0] cur1,cur2;
    light_7seg_ego ego1(.sw(cur1),.rst(rst_n),.seg_out(seg_led1));
    light_7seg_ego ego2(.sw(cur2),.rst(rst_n),.seg_out(seg_led2));
    always@(posedge clk or negedge rst_n)
        if( rst_n||power_now )
            led <= 8'h80;
        else
                if(clk_bps)
                    if( led != 8'h01 )
                        led <= led >>1'b1;  //shift right
                    else
                        led <= 8'h80;
```

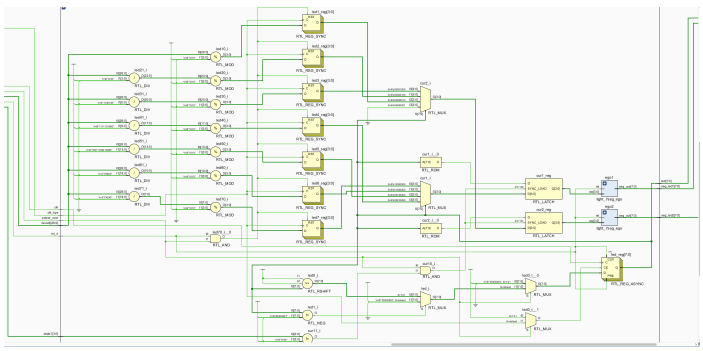
```
module light_7seg_ego(input [3:0]sw,output reg [7:0] seg_out, input rst);

    always @ *

            if(rst)begin
            seg_out = 8'b1111_1100;
            end
            else begin
                case(sw)
            4'h0: seg_out = 8'b1111_1100; //0
            4'h1: seg_out = 8'b0110_0000; //1
            4'h2: seg_out = 8'b1101_1010; //2
            4'h3: seg_out = 8'b1111_0010; //3
            4'h4: seg_out = 8'b0110_0110; //4
            4'h5: seg_out = 8'b1011_0110; //5
            4'h6: seg_out = 8'b1011_1110; //6
            4'h7: seg_out = 8'b1110_0000; //7
            4'h8: seg_out = 8'b1111_1110; //8
            4'h9: seg_out = 8'b1110_0110; //9
            4'ha: seg_out = 8'b1110_1110; //A
            4'hb: seg_out = 8'b0011_1110; //B
            4'hc: seg_out = 8'b1001_1100; //C
            4'hd: seg_out = 8'b0111_1010; //D
            4'he: seg_out = 8'b1001_1110; //E
            4'hf: seg_out = 8'b1000_1110; //F
            default: seg_out = 8'b0000_0001;
        endcase
            end
endmodule
```

```
//里程划分
always @(posedge clk_bps ) begin//复位和断电状态就让板子灯熄灭
        if (rst_n&&power_now) begin
          led7 <=0; //0
          led6 <=0; //0
          led5 <=0; //0
          led4 <=0; //0
          led3 <=0; //0
          led2 <=0; //0
          led1 <=0; //0


        end
        else begin
                        led7 <= (record/100_0000)%10;
                        led6 <= (record/10_0000)%10;
                        led5 <= (record/1_0000)%10;
                        led4 <= (record/1_000)%10;
                        led3 <= (record/100)%10;
                        led2 <= (record/10)%10;
                        led1 <= record%10;
        end
        end
```

```
always @(*) begin//流水赋值，形成视觉暂留
        if(power_now&&state1!=4'b0100) begin
          cur1<=4'hc;
          cur2<= 4'hd;
        end
        else begin
          case(led)
          8'b1000_0000:begin
                       cur1<=led7;

          end
          8'b0100_0000:begin
                       cur1<=led6;

          end
          8'b0010_0000:begin
                       cur1<=led5;

          end
          8'b0001_0000:begin
                       cur1<=led4;

          end
          8'b0000_1000:begin
                       cur2<=led3;

          end
          8'b0000_0100:begin
                       cur2<=led2;
          end
          8'b0000_0010:begin
                       cur2<=led1;
          end
          8'b0000_0001:begin
                       cur2<=0;
          end
        endcase
        end
end
```

结构图如下





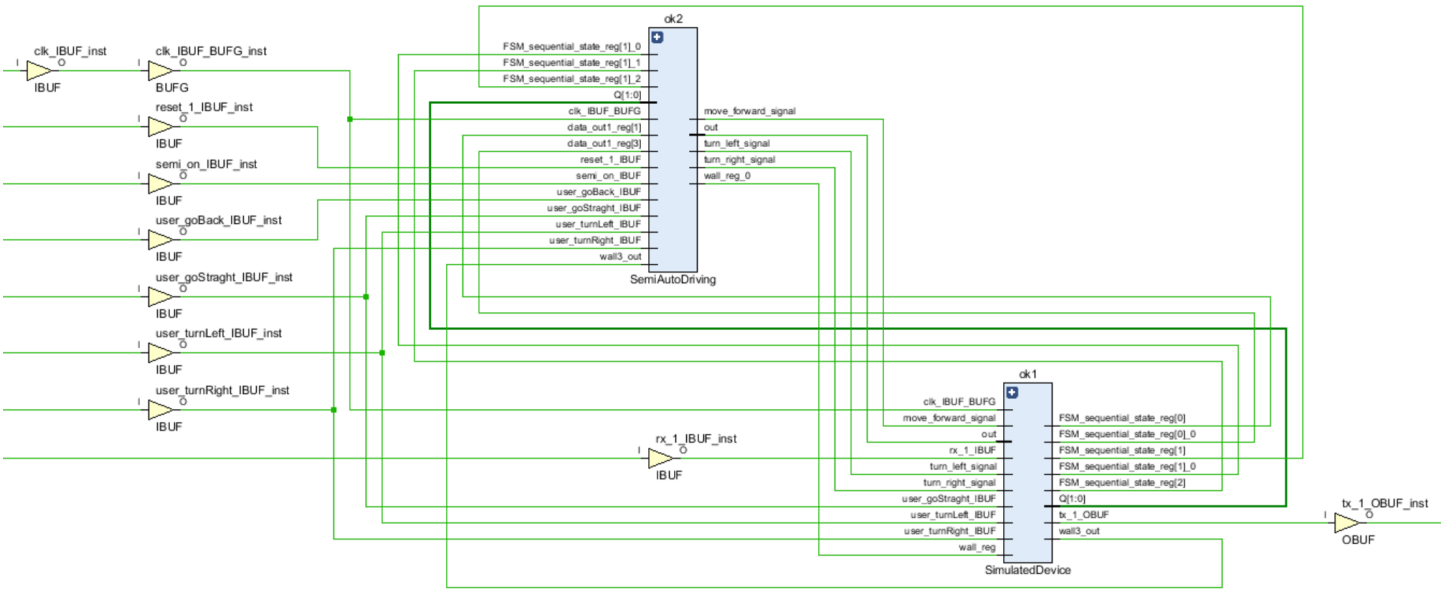
4.3 半自动挡

```
//整个模块只有一个时钟上升沿触发的时序逻辑电路
always @(posedge clk_100hz) begin
    if (reset) begin //重置
        state<=3'b000;
        wall<=0;
        counter<=8'b00000000;
        {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0000;
    end
    else if(semi_auto_mode_on==1'b1)begin //半自动驾驶模式开启
        counter<=counter+1'b1; //计数器
        case(state)
            3'b000:begin //当前状态为等待指令
                if (go_straight_command) begin
                    state <= 3'b011; //接收到前进指令，切换到前进状态
                    counter<=8'b00000000;
                wall<=0;
                end
                else if (turn_right_command) begin
                    state <= 3'b001; //接收到右转指令，切换到右转状态
                    counter<=8'b00000000;
                wall<=0;
                end
                else if (turn_left_command) begin
                    state <= 3'b010; //接收到左转指令，切换到左转状态
                    counter<=8'b00000000;
                wall<=0;
                end
                else if(turn_back_command) begin
                    state<=3'b100; //接收到调头指令，切换到调头状态
                    counter<=8'b00000000;
                wall<=0;
                end
            end
            3'b001:begin //当前状态为向右转
                if (counter>=8'd90) begin
                        state <= 3'b011; //右转完成后，切换到前进状态
                        counter<=8'b00000000;
                wall<=0;
                end
            end
            3'b010:begin //当前状态为向左转
                if (counter>=8'd90) begin
                        state <= 3'b011; //左转完成后，切换到前进状态
                        counter<=8'b00000000;
                wall<=0;
                end
            end
```

```
        3'b011:begin //当前状态为前进
            if (!wall) begin //寄存器wall值为0时表示前方探测器未发现障碍
                if(counter>=8'd35) begin
                    if (detectors==4'b0100||detectors==4'b0010||detectors==4'b0001||detectors==4'b0000) begin
                        state<=3'b000; //遇到岔路口，切换到等待指令状态
                        counter<=8'b00000000;
                    end
                end
                if (detectors==4'b0110) begin
                    counter<=0;
                    wall<=1;
                end
                else if (detectors==4'b0101) begin
                    counter<=0;
                    wall<=1;
                end
                else if (detectors==4'b0111) begin
                    counter<=0;
                    wall<=1;
                end
                else if(detectors==4'b0100) begin
                    counter<=0;
                    wall<=1;
                end
            end
            else if(wall) begin //寄存器wall值为1时表示前方探测器发现了障碍
                if(counter>=8'd6) begin ///前方发现障碍后，经过了6个周期，再做出反应（防止前方探测器刚碰到墙就做出反应，小车走不到路中间）
                    if (detectors==4'b0110) begin
                        state <= 3'b001; //只有右方有路，切换到右转状态
                        counter<=8'b00000000;
                    end
                    else if (detectors==4'b0101) begin
                        state <= 3'b010; //只有左方有路，切换到左转状态
                        counter<=8'b00000000;
                    end
                    else if (detectors==4'b0111) begin
                        state<=3'b100; //只有后方有路，切换到调头状态
                        counter<=8'b00000000;
                    end
                    else if(detectors==4'b0100) begin
                        state<=3'b000; //遇到左右都有路的岔路口，切换到等待命令状态
                        counter<=8'b00000000;
                    end
                end
            end
        end
        3'b100:begin //当前状态为调头
            if (counter>=8'd180) begin
                state <= 3'b011; //调头完成后，切换到前进状态
                counter<=8'b00000000;
                wall<=0;
            end
        end
```

```
    case (state) //不同状态时对应的输出信号
        3'b000:  {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0000;
        3'b001:  {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0001;
        3'b010:  {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0010;
        3'b011:  {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0100;
        3'b100:  {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0001;
        default: {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<={move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal};
    endcase
```

结构图如下：



4.4 自动挡

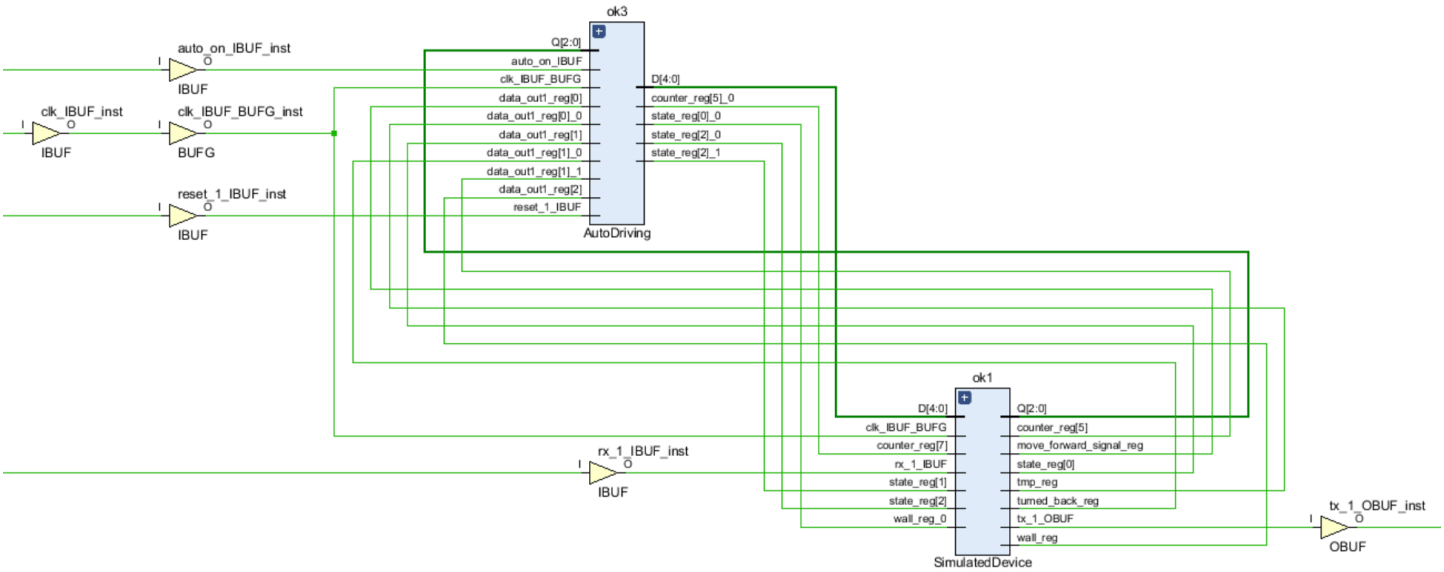
```
//整个模块只有一个时钟上升沿触发的时序逻辑电路
always @(posedge clk_100hz) begin
    if (reset) begin //重置
        state<=3'b000;
        wall<=0;
        counter<=8'd50;
        another_cnt<=0;
        turned_right<=0; //turned_right用于表示小车刚刚右转过
        turned_back<=0; //turned_back用于表示小车刚刚调头过
        {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0000;
        tmp<=0;
    end
    else if(auto_mode_on==1'b1)begin
        counter<=counter+1'b1;
        another_cnt<=another_cnt+1'b1;
        case(state)
            3'b110: begin //此状态表示，正在销毁最近放置的一个障碍
                if(another_cnt==8'd5)begin
                    destroy_barrier_signal<=1;
                end
                if(another_cnt==8'd10) begin
                    destroy_barrier_signal<=0; //destroy_barrier_signal信号为高天平持续5个周期
                end
                if(another_cnt>=8'd50) begin
                    wall<=0;
                    counter<=8'd50;
                    state<=3'b011;
                    another_cnt<=8'b0;
                end
            end
            3'b000:begin //初始状态
                if(detectors==4'b0111) begin
                    state<=3'b111; //起点处只有后方有路，先调头
                    counter<=8'b00000000;
                end
                else begin
                    state=3'b011; //进入行进状态
                    counter<=8'd50;
                end
            end
            3'b001:begin //当前状态为向右转
                if (counter>=8'd90) begin
                        state <= 3'b011; //右转完成后，切换到前进状态
                        counter<=8'b00000000;
                        wall<=0;
                        turned_right<=1;
                end
            end
            3'b010:begin //当前状态为向左转
                if (counter>=8'd90) begin
                        state <= 3'b011; //左转完成后，切换到前进状态
                        counter<=8'b00000000;
                        wall<=0;
                end
            end
```

```
3'b011:begin //当前状态为前进
    if(turned_right) begin //如果刚刚完成过右转，直行一段时间后放置一个信标
        if(counter==8'd75) begin //直行一段时间后，使place_barrier_signal为高电平
            place_barrier_signal<=1;
        end
        else if(counter==8'd80) begin //5个周期后，使place_barrier_signal恢复低电平，放置一个信标
            place_barrier_signal<=0;
            turned_right<=0;
        end
    end

    if (!wall) begin //寄存器wall值为0时表示前方探测器未发现障碍
            if(counter>=8'd40) begin //走过一个路口，再判断是否进入新的岔路口
                if (detectors==4'b0100||detectors==4'b0010||detectors==4'b0000||
                    detectors==4'b1100||detectors==4'b1010||detectors==4'b1000) begin
                    state<=3'b001; //遇到岔路口，切换到等待指令状态
                    counter<=8'b00000000;
                    if(turned_back) begin
                        turned_back<=0;
                    end
                end
                else if(detectors==4'b0001||detectors==4'b1001) begin
                    state<=3'b011; //左方和前方都有路时，继续向前走
                    counter<=8'b00000000;
                    if(turned_back) begin
                        turned_back<=0;
                    end
                end
            end
            //前方探测器检测到障碍
            if (detectors==4'b0110||detectors==4'b1110) begin
                counter<=0;
                wall<=1;
            end
            else if (detectors==4'b0101||detectors==4'b1101) begin
                counter<=0;
                wall<=1;
            end
            else if (detectors==4'b0111||detectors==4'b1111) begin
                counter<=0;
                wall<=1;
            end
            else if(detectors==4'b0100||detectors==4'b1100) begin
                counter<=0;
                wall<=1;
                if(turned_back) begin
                    turned_back<=0;
                end
            end
    end
```

```
            else if(wall) begin //寄存器wall值为1时表示前方探测器发现了障碍
                if(counter>=8'd8) begin //前方发现障碍后，经过了8个周期，再做出反应（防止前方探测器刚碰到墙就做出反应，小车走不到路中间）
                    if (detectors==4'b0110||detectors==4'b1110) begin
                        state <= 3'b001; //只有右方有路，切换到右转状态
                        counter<=8'b00000000;
                    end
                    else if (detectors==4'b0101||detectors==4'b1101) begin
                        state <= 3'b010; //只有左方有路，切换到左转状态
                        counter<=8'b00000000;
                    end
                    else if(detectors==4'b0100||detectors==4'b1100) begin
                        state<=3'b001; //右方和前方有路，切换到右转状态
                        counter<=8'b00000000;
                        if(turned_back) begin
                            turned_back<=0;
                        end
                    end
                    else if (detectors==4'b0111||detectors==4'b1111) begin //前方左方右方都没路
                        if(turned_back && tmp==0) begin //如果已经掉过头，说明进入了死胡同，前方障碍一定是信标，摧毁前方的信标
                            state<=3'b110; //进入摧毁信标状态
                            counter<=8'b00000000;
                            another_cnt<=8'b0;
                            wall<=0;
                            tmp<=1;
                        end
                        else begin
                            state<=3'b100; //正常进入调头状态
                            counter<=8'b00000000;
                        end
                    end
                    else begin
                        wall<=0;
                    end
                end
            end
            3'b100:begin //当前状态为调头
                if (counter>=8'd180) begin
                    state <= 3'b011; //调头完成后，进入前进状态
                    counter<=8'b00000000;
                    turned_back<=1;
                    wall<=0;
                end
            end
            3'b111:begin //起点处如果只有后方有路，那么先调头
                if (counter>=8'd180) begin
                    state <= 3'b011;
                    counter<=8'b00000000;
                    wall<=0;
                end
            end
        endcase
```

```
        case (state) //不同状态时对应的输出信号
            3'b000:  {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0000;
            3'b001:  {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0001;
            3'b010:  {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0010;
            3'b011:  {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0100;
            3'b100:  {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0001;
            3'b111:  {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0001;
            // default: {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<={move_backward_signal,move_forward_
            default:  {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b000;
        endcase
    end
    else begin
        {move_backward_signal,move_forward_signal,turn_left_signal,turn_right_signal}<=4'b0000;
    end
```

结构图如下：



4.5 VGA

```
module vga_top (input clk,  // input of system clock
                input debounced_power_on,    // obtained from the top module, store the current state of the car controlled by the button
                input manul_mode_on,         // obtained from the top module, store the current mode
                input semi_auto_mode_on,
                input auto_mode_on,
                input [26:0]record,          // obtained from the top module, store the record of the manual mode
                input power_now,         // obtained from the top module, store the present state of the car(power on/off)
                input rst_n,             // connected to the rst button on the board
                output reg hsync,        // hsync and vsync are connected to the port on board, output information for the VGA display
                output reg vsync,
                output reg [3:0] red,        // color information of the VGA display
                output reg [3:0] green,
                output reg [3:0] blue);
```

4.5.1 分频得到VGA所需要的频率

```
    // get the vga clock
    always@(posedge clk)
    begin
        begin
        clk_counter <= clk_counter+1'b1;
    end
    end
    assign clk_vga = clk_counter[1];
```

4.5.2 VGA输出hsyc, vsyc及当前坐标信息hc, vc获取

```
// get the current scan coordinate hc,vc and the output hsyc,vsyc connected to the pins on EGo1 board
always @(posedge clk_vga) begin
    if (hc == 10'd799)
    begin
        hc <= 10'd0;
    end
    else
    begin
        hc <= hc +1'b1;
    end
end

always @(posedge clk_vga) begin
    if (hc < 10'd96)  //同锯斤拷为96 96 is the sync time
        hsync <= 1'b0;
    else
        hsync <= 1'b1;
end
// update the index of current vertical line when the scan reach the end of a horizontal line
always @(posedge clk_vga) begin
    if (hc == 799) // when the scan reach the end of a horizontal line
    begin
        if (vc == 10'd524) // revised from 520
            vc <= 10'd0;
        else
            vc <= vc + 1'b1;
    end
    else
        vc <= vc;
end
always @(posedge clk_vga) begin
    if (vc < 10'd2)
        vsync <= 1'b0;
    else
        vsync <= 1'b1;
end
```

4.5.3 VGA色彩显示

```
// get the information of whrther the current scan coordinate position is possible to print color
always @(posedge clk_vga )
    begin
       if(vc==10'd32)
            valid_yr<=1'b1;
        else if(vc==10'd511)
            valid_yr<=1'b0;
    end
wire valid_y=valid_yr;
reg valid_r;
always @(posedge clk_vga )
    begin
        if((hc==10'd141)&&valid_y)
            valid_r<=1'b1;
        else if((hc==10'd781)&&valid_y)
            valid_r<=1'b0;
    end
wire valid=valid_r; // valid store if the current coordintate is valid to display color
```

```
// store the color of the current scaned pixel to show if it is possible to show the color
reg [2:0]color_index; // store the current color: 000:black 111:white
always @(posedge clk_vga) begin
    case (debounced_power_on)
        0: begin
            has_record = 1'b0;
            if ((hc >= hbp+state_xs)&&(hc < hbp+state_xe)&&(vc >= vbp+state_ys)&&(vc < vbp+state_ye))
            begin
                color_index <= 3'b100;
            end
            else
            begin
                color_index <= 3'b000;
            end
        end
        default: begin
            case ({manul_mode_on,semi_auto_mode_on,auto_mode_on})
                3'b100:
                begin
                    if ((hc >= hbp+state_xs)&&(hc < hbp+state_xe)&&(vc >= vbp+state_ys)&&(vc < vbp+state_ye))
                    begin
                        has_record = 1'b0;
                        color_index <= 3'b110;
                    end
                    else
                    begin
                        if (rst_n&&power_now) begin
                            has_record = 1'b0;
                            color_index <= 3'b000;
                        end
                        else begin
                            has_record = 1'b1;
                            led7 <= (record/100_0000)%10;
                            led6 <= (record/10_0000)%10;
                            led5 <= (record/1_0000)%10;
                            led4 <= (record/1_000)%10;
                            led3 <= (record/100)%10;
                            led2 <= (record/10)%10;
                            led1 <= record%10;

                            if ((hc >= hbp+nums_xs1)&&(hc < hbp+nums_xe1)&&(vc >= vbp+nums_ys)&&(vc < vbp+nums_ye)) begin
                                digit   <= led7;
                                x       <= hc-hbp-nums_xs1;
                                y       <= vc-vbp-nums_ys;
                                color_index <= 3'b111;
                            end
                            else if ((hc >= hbp+nums_xs2)&&(hc < hbp+nums_xe2)&&(vc >= vbp+nums_ys)&&(vc < vbp+nums_ye)) begin
                                digit   <= led6;
                                x       <= hc-hbp-nums_xs2;
                                y       <= vc-vbp-nums_ys;
                                color_index <= 3'b111;
                            end
                                else if ((hc >= hbp+nums_xs3)&&(hc < hbp+nums_xe3)&&(vc >= vbp+nums_ys)&&(vc < vbp+nums_ye)) begin
                                digit   <= led5;
```

```
                        digit   <= led3;
                        x       <= hc-hbp-nums_xs3;
                        y       <= vc-vbp-nums_ys;
                        color_index <= 3'b111;
                        end
                        else if ((hc >= hbp+nums_xs4)&&(hc < hbp+nums_xe4)&&(vc >= vbp+nums_ys)&&(vc < vbp+nums_ye)) begin
                        digit   <= led4;
                        x       <= hc-hbp-nums_xs4;
                        y       <= vc-vbp-nums_ys;
                        color_index <= 3'b111;
                        end
                        else if ((hc >= hbp+nums_xs5)&&(hc < hbp+nums_xe5)&&(vc >= vbp+nums_ys)&&(vc < vbp+nums_ye)) begin
                        digit   <= led3;
                        x       <= hc-hbp-nums_xs5;
                        y       <= vc-vbp-nums_ys;
                        color_index <= 3'b111;
                        end
                        else if ((hc >= hbp+nums_xs6)&&(hc < hbp+nums_xe6)&&(vc >= vbp+nums_ys)&&(vc < vbp+nums_ye)) begin
                        digit   <= led2;
                        x       <= hc-hbp-nums_xs6;
                        y       <= vc-vbp-nums_ys;
                        color_index <= 3'b111;
                        end
                        else if ((hc >= hbp+nums_xs7)&&(hc < hbp+nums_xe7)&&(vc >= vbp+nums_ys)&&(vc < vbp+nums_ye)) begin
                        digit   <= led1;
                        x       <= hc-hbp-nums_xs7;
                        y       <= vc-vbp-nums_ys;
                        color_index <= 3'b111;
                        end
                        else if ((hc >= hbp+nums_xs8)&&(hc < hbp+nums_xe8)&&(vc >= vbp+nums_ys)&&(vc < vbp+nums_ye)) begin
                        digit   <= 4'b0;
                        x       <= hc-hbp-nums_xs8;
                        y       <= vc-vbp-nums_ys;
                        color_index <= 3'b111;
                        end
                    else begin
                        color_index <= 3'b000;
                    end

                end
            end
        3'b010:
        begin
            has_record = 0;
            if ((hc >= hbp+state_xs)&&(hc < hbp+state_xe)&&(vc >= vbp+state_ys)&&(vc < vbp+state_ye))
            begin
                color_index <= 3'b010;
            end
            else
            begin
                color_index <= 3'b000;
            end
        end
        3'b001:
        begin
            has_record = 0;
            if ((hc >= hbp+state_xs)&&(hc < hbp+state_xe)&&(vc >= vbp+state_ys)&&(vc < vbp+state_ye))
            begin
                color_index <= 3'b001;
            end
            else
            begin
                color_index <= 3'b000;
            end
        end
        default:begin
            has_record = 1'b0;
            color_index <= 3'b000;
        end
    endcase
  end
endcase
end
```

4.5.4 数字信号输入

```
module num_switch (
    input clk_vga,  // get from the vga_top module, is the clk used for VGA
    input has_record, // get from the vga_top module, store whether the present scan position has record.
    input [3:0] digit, // get from the vga_top module, store the digit to display on VGA
    input [9:0]y,       // get from the vga_top module, store the current scan position
    input [9:0]x,
    output reg has_color    // give information to the vga_top module of whether the current scan position has a color
);
```

利用vga_num_switch 子模块，获得了在不同坐标时，不同数字所需要的输出（黑屏或者有显示）

```
// For every digit, the display is consist of 32 lines.
// In this always block, depends on different number and the current scan position of the VGA display ,
// we assign different values to the variable num_line
always @(posedge clk_vga) begin
    case (has_record)
        1'b1:
        begin
        case (digit)
            4'b0000:
            begin
                case (y)
                     0: num_line<=20'b00000000000000000000;
                     1: num_line<=20'b00000000000000000000;
                     2: num_line<=20'b00000000000000000000;
                     3: num_line<=20'b00000000000000000000;
                     4: num_line<=20'b00000000000000000000;
                     5: num_line<=20'b00000000000000000000;
                     6: num_line<=20'b00000000000000000000;
                     7: num_line<=20'b00000000000000000000;
                     8: num_line<=20'b00000000000000000000;
                     9: num_line<=20'b00001100000000000000;
                    10: num_line<=20'b00011100000000000000;
                    11: num_line<=20'b00111111000000000000;
                    12: num_line<=20'b01111111000000000000;
                    13: num_line<=20'b01111111000000000000;
                    14: num_line<=20'b01110111100000000000;
                    15: num_line<=20'b01100011100000000000;
                    16: num_line<=20'b01100011100000000000;
                    17: num_line<=20'b01100011100000000000;
                    18: num_line<=20'b01100011100000000000;
                    19: num_line<=20'b01100001100000000000;
                    20: num_line<=20'b01100001100000000000;
                    21: num_line<=20'b01100001100000000000;
                    22: num_line<=20'b01100001100000000000;
                    23: num_line<=20'b01100001100000000000;
                    24: num_line<=20'b01100011100000000000;
                    25: num_line<=20'b01100011100000000000;
                    26: num_line<=20'b01100011100000000000;
                    27: num_line<=20'b01110011100000000000;
                    28: num_line<=20'b01110111100000000000;
                    29: num_line<=20'b01110111100000000000;
                    30: num_line<=20'b01111111000000000000;
                    31: num_line<=20'b00111110000000000000;
                    default: num_line<=20'b00000000000000000000;
                endcase
            end
```

```
4'b0001: //1
begin
    case (y)
        0: num_line<=20'b00000000000000000000;
        1: num_line<=20'b00000000000000000000;
        2: num_line<=20'b00000000000000000000;
        3: num_line<=20'b00000000000000000000;
        4: num_line<=20'b00000000000000000000;
        5: num_line<=20'b00000000000000000000;
        6: num_line<=20'b00000000000000000000;
        7: num_line<=20'b00000000000000000000;
        8: num_line<=20'b00000000000000000000;
        9: num_line<=20'b00001100000000000000;
       10: num_line<=20'b00001100000000000000;
       11: num_line<=20'b00011100000000000000;
       12: num_line<=20'b00111100000000000000;
       13: num_line<=20'b00111100000000000000;
       14: num_line<=20'b01111100000000000000;
       15: num_line<=20'b01111100000000000000;
       16: num_line<=20'b01111100000000000000;
       17: num_line<=20'b01111100000000000000;
       18: num_line<=20'b00011100000000000000;
       19: num_line<=20'b00011100000000000000;
       20: num_line<=20'b00011100000000000000;
       21: num_line<=20'b00011100000000000000;
       22: num_line<=20'b00011100000000000000;
       23: num_line<=20'b00011100000000000000;
       24: num_line<=20'b00011100000000000000;
       25: num_line<=20'b00011100000000000000;
       26: num_line<=20'b00011100000000000000;
       27: num_line<=20'b00011100000000000000;
       28: num_line<=20'b00011110000000000000;
       29: num_line<=20'b00011110000000000000;
       30: num_line<=20'b00111111000000000000;
       31: num_line<=20'b00111111000000000000;
       default: num_line<=20'b00000000000000000000;
    endcase
end
4'd2: begin //2
    case (y)
        0: num_line<=20'b00000000000000000000;
        1: num_line<=20'b00000000000000000000;
        2: num_line<=20'b00000000000000000000;
        3: num_line<=20'b00000000000000000000;
        4: num_line<=20'b00000000000000000000;
        5: num_line<=20'b00000000000000000000;
        6: num_line<=20'b00000000000000000000;
        7: num_line<=20'b00000000000000000000;
        8: num_line<=20'b00000000000000000000;
        9: num_line<=20'b00011100000000000000;
       10: num_line<=20'b00011110000000000000;
       11: num_line<=20'b00111111000000000000;
       12: num_line<=20'b01111111100000000000;
       13: num_line<=20'b01111111100000000000;
       14: num_line<=20'b01111111100000000000;
```

```
            15: num_line<=20'b01100011100000000000;
            16: num_line<=20'b01100011100000000000;
            17: num_line<=20'b01100011100000000000;
            18: num_line<=20'b00000011100000000000;
            19: num_line<=20'b00000111100000000000;
            20: num_line<=20'b00001111100000000000;
            21: num_line<=20'b00001111000000000000;
            22: num_line<=20'b00001111000000000000;
            23: num_line<=20'b00011110000000000000;
            24: num_line<=20'b00011110000000000000;
            25: num_line<=20'b00111100100000000000;
            26: num_line<=20'b00111000100000000000;
            27: num_line<=20'b01111001100000000000;
            28: num_line<=20'b01111111100000000000;
            29: num_line<=20'b01111111100000000000;
            30: num_line<=20'b01111111100000000000;
            31: num_line<=20'b01111111100000000000;
            default: num_line<=20'b00000000000000000000;
        endcase
    end
    4'b0011:begin
        case (y)
             0: num_line<=20'b00000000000000000000;
             1: num_line<=20'b00000000000000000000;
             2: num_line<=20'b00000000000000000000;
             3: num_line<=20'b00000000000000000000;
             4: num_line<=20'b00000000000000000000;
             5: num_line<=20'b00000000000000000000;
             6: num_line<=20'b00000000000000000000;
             7: num_line<=20'b00000000000000000000;
             8: num_line<=20'b00000000000000000000;
             9: num_line<=20'b00001110000000000000;
            10: num_line<=20'b00001110000000000000;
            11: num_line<=20'b00111111000000000000;
            12: num_line<=20'b01111111000000000000;
            13: num_line<=20'b01111111000000000000;
            14: num_line<=20'b01111111000000000000;
            15: num_line<=20'b01100111000000000000;
            16: num_line<=20'b01100111000000000000;
            17: num_line<=20'b01001111000000000000;
            18: num_line<=20'b00001110000000000000;
            19: num_line<=20'b00011111100000000000;
            20: num_line<=20'b00011111100000000000;
            21: num_line<=20'b00011111100000000000;
            22: num_line<=20'b00011111100000000000;
            23: num_line<=20'b00000011100000000000;
            24: num_line<=20'b00000011100000000000;
            25: num_line<=20'b00000011100000000000;
            26: num_line<=20'b00000011100000000000;
            27: num_line<=20'b00000011100000000000;
            28: num_line<=20'b01110111100000000000;
            29: num_line<=20'b01110111100000000000;
            30: num_line<=20'b01111111000000000000;
            31: num_line<=20'b01111110000000000000;
            default: num_line<=20'b00000000000000000000;
```

```
    endcase
end
4'b0100: begin
    case (y)
        0: num_line<=20'b00000000000000000000;
        1: num_line<=20'b00000000000000000000;
        2: num_line<=20'b00000000000000000000;
        3: num_line<=20'b00000000000000000000;
        4: num_line<=20'b00000000000000000000;
        5: num_line<=20'b00000000000000000000;
        6: num_line<=20'b00000000000000000000;
        7: num_line<=20'b00000000000000000000;
        8: num_line<=20'b00000000000000000000;
        9: num_line<=20'b00000011000000000000;
       10: num_line<=20'b00000011000000000000;
       11: num_line<=20'b00000111000000000000;
       12: num_line<=20'b00001111000000000000;
       13: num_line<=20'b00001111000000000000;
       14: num_line<=20'b00001111000000000000;
       15: num_line<=20'b00011111000000000000;
       16: num_line<=20'b00011111000000000000;
       17: num_line<=20'b00111111000000000000;
       18: num_line<=20'b00111111000000000000;
       19: num_line<=20'b00111111000000000000;
       20: num_line<=20'b01110111000000000000;
       21: num_line<=20'b01110111000000000000;
       22: num_line<=20'b01110111000000000000;
       23: num_line<=20'b01111111110000000000;
       24: num_line<=20'b01111111110000000000;
       25: num_line<=20'b01111111110000000000;
       26: num_line<=20'b01111111110000000000;
       27: num_line<=20'b01111111110000000000;
       28: num_line<=20'b00000111000000000000;
       29: num_line<=20'b00000111000000000000;
       30: num_line<=20'b00000111000000000000;
       31: num_line<=20'b00000111000000000000;
       default: num_line<=20'b00000000000000000000;
    endcase
end
4'b0101: begin
    case (y)
        0: num_line<=20'b00000000000000000000;
        1: num_line<=20'b00000000000000000000;
        2: num_line<=20'b00000000000000000000;
        3: num_line<=20'b00000000000000000000;
        4: num_line<=20'b00000000000000000000;
        5: num_line<=20'b00000000000000000000;
        6: num_line<=20'b00000000000000000000;
        7: num_line<=20'b00000000000000000000;
        8: num_line<=20'b00000000000000000000;
        9: num_line<=20'b00111111000000000000;
       10: num_line<=20'b00111111000000000000;
       11: num_line<=20'b00111111000000000000;
       12: num_line<=20'b00111111000000000000;
       13: num_line<=20'b00111111000000000000;
```

```
            14: num_line<=20'b00111111000000000000;
            15: num_line<=20'b00110000000000000000;
            16: num_line<=20'b00110000000000000000;
            17: num_line<=20'b00111110000000000000;
            18: num_line<=20'b00111110000000000000;
            19: num_line<=20'b00111111000000000000;
            20: num_line<=20'b00111111100000000000;
            21: num_line<=20'b00111111100000000000;
            22: num_line<=20'b00111111100000000000;
            23: num_line<=20'b00000011100000000000;
            24: num_line<=20'b00000011100000000000;
            25: num_line<=20'b00000011100000000000;
            26: num_line<=20'b00000011100000000000;
            27: num_line<=20'b00000011100000000000;
            28: num_line<=20'b01110111100000000000;
            29: num_line<=20'b01110111100000000000;
            30: num_line<=20'b01111111000000000000;
            31: num_line<=20'b01111111000000000000;
            default: num_line<=20'b00000000000000000000;
        endcase
    end
    4'b0110:begin
        case (y)
             0: num_line<=20'b00000000000000000000;
             1: num_line<=20'b00000000000000000000;
             2: num_line<=20'b00000000000000000000;
             3: num_line<=20'b00000000000000000000;
             4: num_line<=20'b00000000000000000000;
             5: num_line<=20'b00000000000000000000;
             6: num_line<=20'b00000000000000000000;
             7: num_line<=20'b00000000000000000000;
             8: num_line<=20'b00000000000000000000;
             9: num_line<=20'b00000001000000000000;
            10: num_line<=20'b00000011000000000000;
            11: num_line<=20'b00001111000000000000;
            12: num_line<=20'b00011111000000000000;
            13: num_line<=20'b00011111000000000000;
            14: num_line<=20'b00111110000000000000;
            15: num_line<=20'b00111100000000000000;
            16: num_line<=20'b01111100000000000000;
            17: num_line<=20'b01111000000000000000;
            18: num_line<=20'b01110000000000000000;
            19: num_line<=20'b01111111000000000000;
            20: num_line<=20'b01111111100000000000;
            21: num_line<=20'b01111111100000000000;
            22: num_line<=20'b01101111100000000000;
            23: num_line<=20'b01100011100000000000;
            24: num_line<=20'b01100011100000000000;
            25: num_line<=20'b01100011100000000000;
            26: num_line<=20'b01100011100000000000;
            27: num_line<=20'b01110011100000000000;
            28: num_line<=20'b01110111100000000000;
            29: num_line<=20'b01110111100000000000;
            30: num_line<=20'b01111111100000000000;
            31: num_line<=20'b00111111000000000000;
            default: num_line<=20'b00000000000000000000;
```

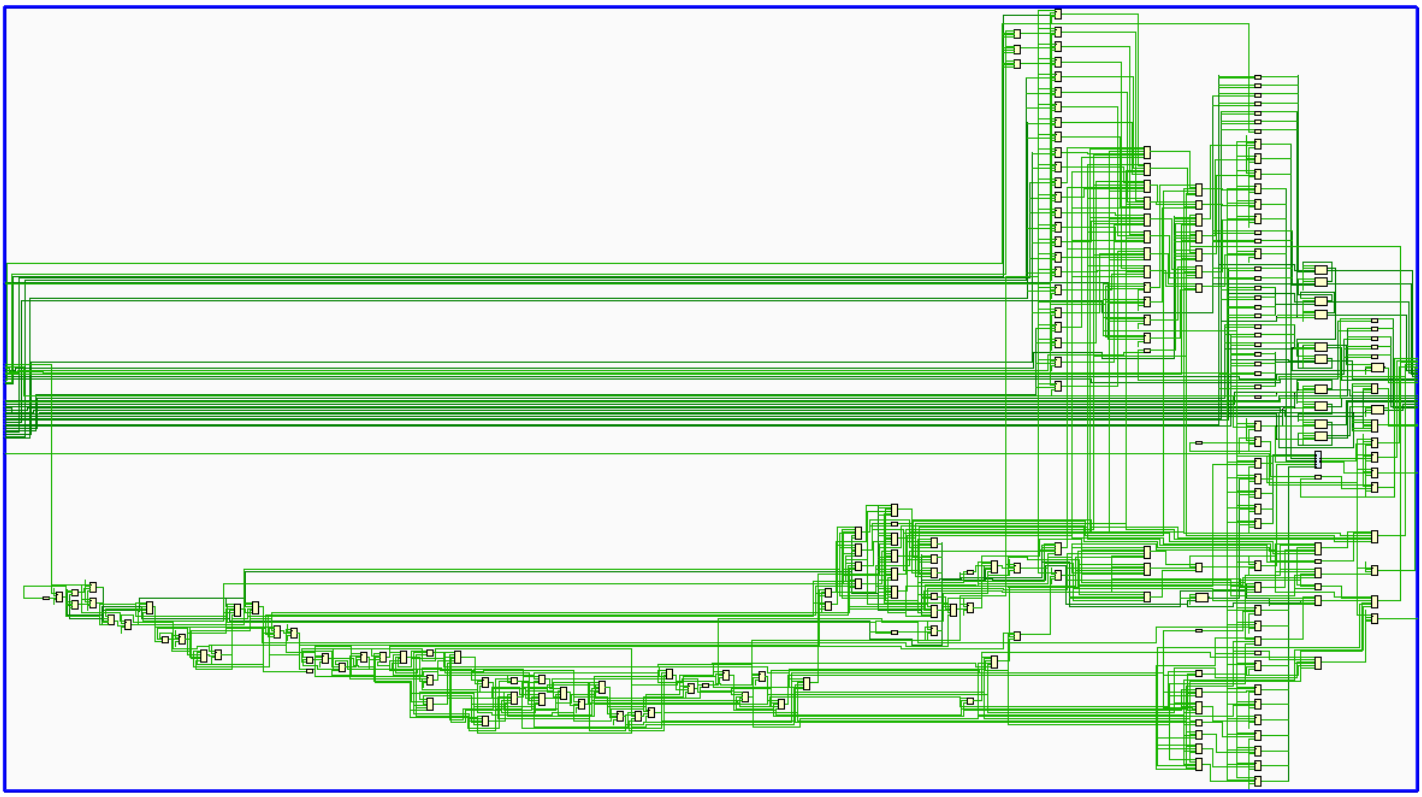
```
        default: num_line<=20'b00000000000000000000;
    endcase
end
4'b0111: begin
    case (y)
        0: num_line<=20'b00000000000000000000;
        1: num_line<=20'b00000000000000000000;
        2: num_line<=20'b00000000000000000000;
        3: num_line<=20'b00000000000000000000;
        4: num_line<=20'b00000000000000000000;
        5: num_line<=20'b00000000000000000000;
        6: num_line<=20'b00000000000000000000;
        7: num_line<=20'b00000000000000000000;
        8: num_line<=20'b00000000000000000000;
        9: num_line<=20'b01111111100000000000;
       10: num_line<=20'b01111111100000000000;
       11: num_line<=20'b01111111100000000000;
       12: num_line<=20'b01111111100000000000;
       13: num_line<=20'b01111111100000000000;
       14: num_line<=20'b01111111100000000000;
       15: num_line<=20'b01000011100000000000;
       16: num_line<=20'b01000011100000000000;
       17: num_line<=20'b00000111000000000000;
       18: num_line<=20'b00000111000000000000;
       19: num_line<=20'b00000111000000000000;
       20: num_line<=20'b00000110000000000000;
       21: num_line<=20'b00000110000000000000;
       22: num_line<=20'b00001110000000000000;
       23: num_line<=20'b00001110000000000000;
       24: num_line<=20'b00001110000000000000;
       25: num_line<=20'b00001100000000000000;
       26: num_line<=20'b00001100000000000000;
       27: num_line<=20'b00011100000000000000;
       28: num_line<=20'b00011100000000000000;
       29: num_line<=20'b00011100000000000000;
       30: num_line<=20'b00011000000000000000;
       31: num_line<=20'b00011000000000000000;
        default: num_line<=20'b00000000000000000000;
    endcase
end
4'b1000: begin
    case (y)
        0: num_line<=20'b00000000000000000000;
        1: num_line<=20'b00000000000000000000;
        2: num_line<=20'b00000000000000000000;
        3: num_line<=20'b00000000000000000000;
        4: num_line<=20'b00000000000000000000;
        5: num_line<=20'b00000000000000000000;
        6: num_line<=20'b00000000000000000000;
        7: num_line<=20'b00000000000000000000;
        8: num_line<=20'b00000000000000000000;
        9: num_line<=20'b00011110000000000000;
       10: num_line<=20'b00011110000000000000;
       11: num_line<=20'b00111111000000000000;
       12: num_line<=20'b01111111100000000000;
       13: num_line<=20'b01111111100000000000;
```

```
            14: num_line<=20'b01110011100000000000;
            15: num_line<=20'b01110011100000000000;
            16: num_line<=20'b01110011100000000000;
            17: num_line<=20'b01111111100000000000;
            18: num_line<=20'b01111111100000000000;
            19: num_line<=20'b01111111100000000000;
            20: num_line<=20'b01111111000000000000;
            21: num_line<=20'b00111111100000000000;
            22: num_line<=20'b01111111100000000000;
            23: num_line<=20'b01111111100000000000;
            24: num_line<=20'b01111111100000000000;
            25: num_line<=20'b01110011100000000000;
            26: num_line<=20'b01100001100000000000;
            27: num_line<=20'b01110011100000000000;
            28: num_line<=20'b01111111100000000000;
            29: num_line<=20'b01111111100000000000;
            30: num_line<=20'b01111111100000000000;
            31: num_line<=20'b00111111000000000000;
            default: num_line<=20'b00000000000000000000;
        endcase
    end
    4'b1001: begin
        case (y)
             0: num_line<=20'b00000000000000000000;
             1: num_line<=20'b00000000000000000000;
             2: num_line<=20'b00000000000000000000;
             3: num_line<=20'b00000000000000000000;
             4: num_line<=20'b00000000000000000000;
             5: num_line<=20'b00000000000000000000;
             6: num_line<=20'b00000000000000000000;
             7: num_line<=20'b00000000000000000000;
             8: num_line<=20'b00000000000000000000;
             9: num_line<=20'b00011100000000000000;
            10: num_line<=20'b00011100000000000000;
            11: num_line<=20'b01111111000000000000;
            12: num_line<=20'b01111111000000000000;
            13: num_line<=20'b01111111000000000000;
            14: num_line<=20'b01110111100000000000;
            15: num_line<=20'b01100011100000000000;
            16: num_line<=20'b01100011100000000000;
            17: num_line<=20'b01100011100000000000;
            18: num_line<=20'b01100011100000000000;
            19: num_line<=20'b01110011100000000000;
            20: num_line<=20'b01111111100000000000;
            21: num_line<=20'b01111111100000000000;
            22: num_line<=20'b01111111100000000000;
            23: num_line<=20'b01111111100000000000;
            24: num_line<=20'b00111111100000000000;
            25: num_line<=20'b00000111100000000000;
            26: num_line<=20'b00000111000000000000;
            27: num_line<=20'b00001111000000000000;
            28: num_line<=20'b00111110000000000000;
            29: num_line<=20'b00111110000000000000;
            30: num_line<=20'b01111100000000000000;
            31: num_line<=20'b01111000000000000000;
```

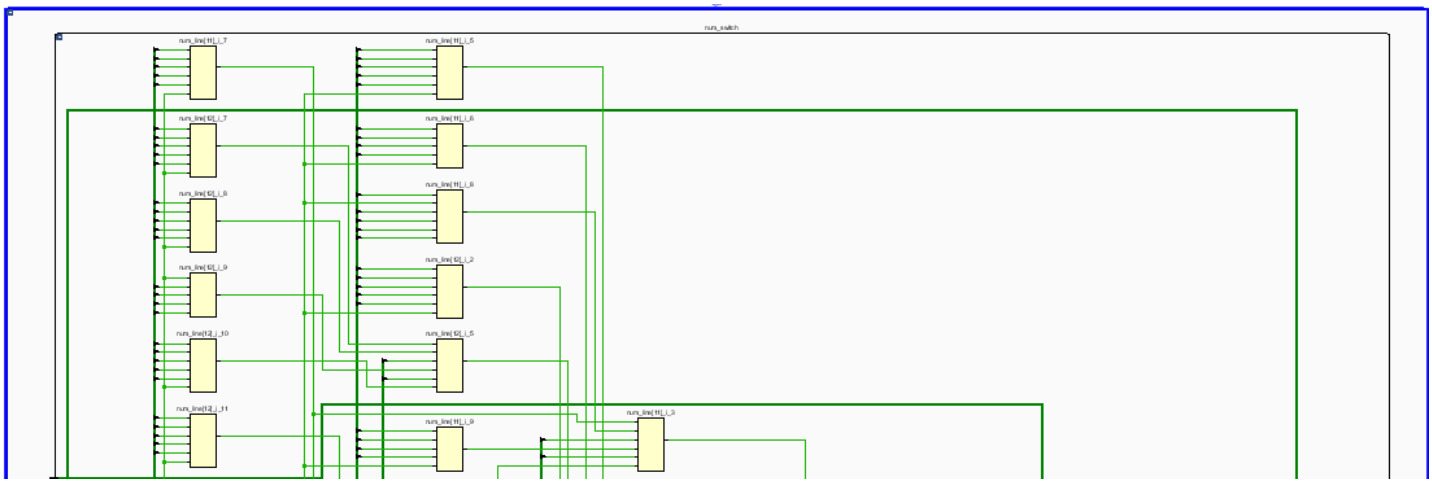
```
                default: num_line<=20'b00000000000000000000;
            endcase
        end
        default: num_line<=20'b00000000000000000000;
    endcase
    end
    default: num_line<=20'b00000000000000000000;
endcase
```

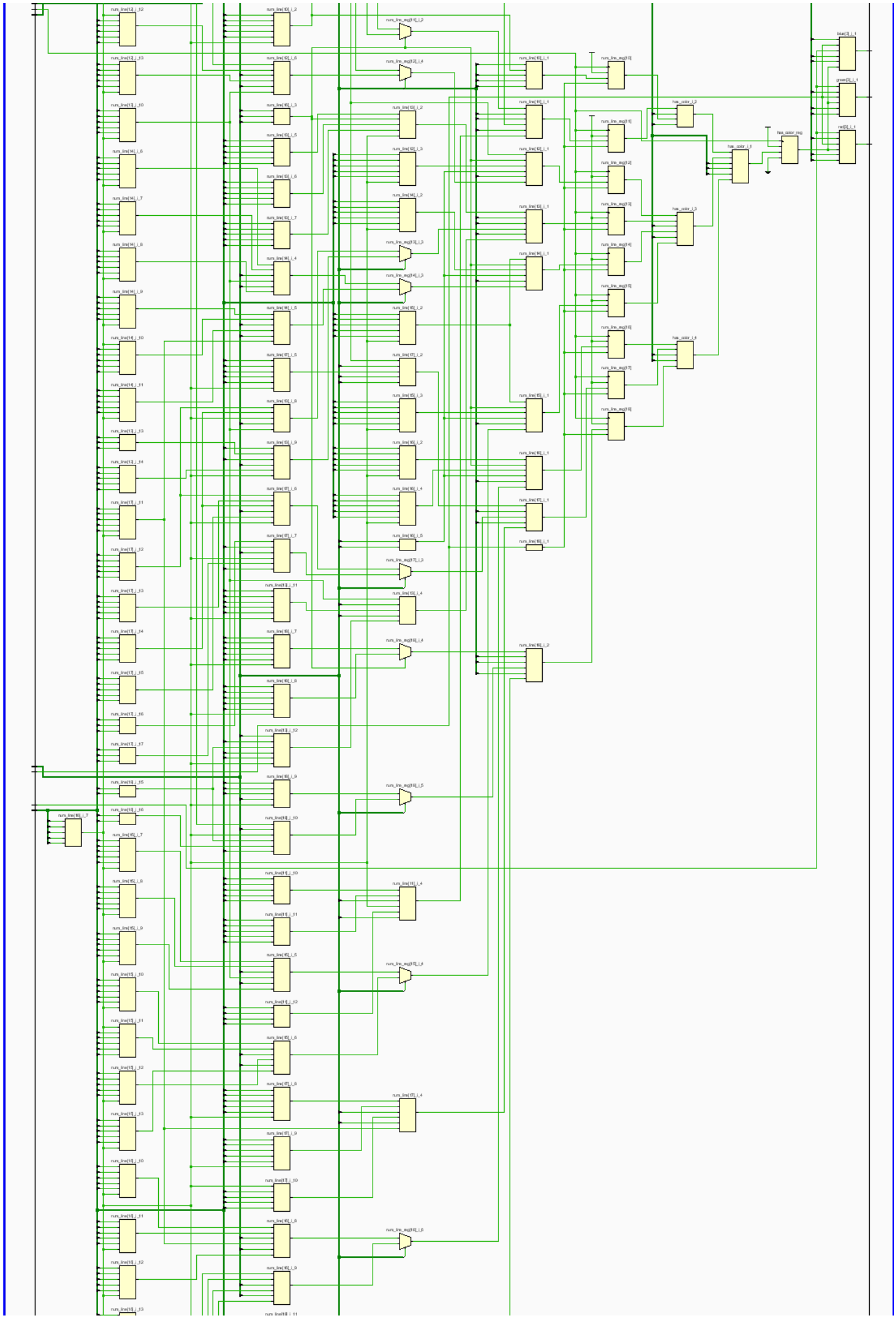
```
// use num_lin to store the current line to be print and store the color of current position in the parameter has_color
always @(posedge clk_vga) begin
    has_color <= num_line[10'd19-x];
end
```

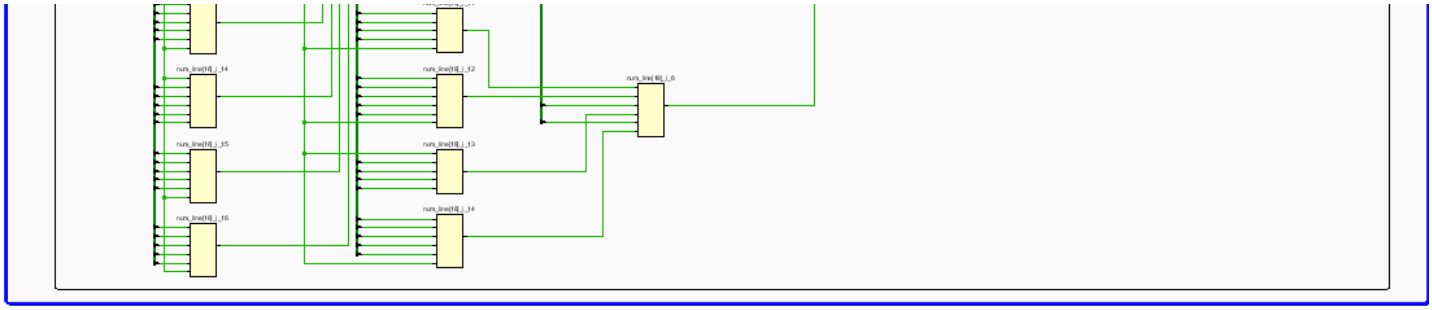
VGA模块整体结构图



子模块vga_num_switch







PART 5: 开发经验总结及优化

5.1 经验总结

- 在总体开始设计之前应该先详细规定好各个子模块的输入输出端口的名称，有效电频，以及与顶层模块的连接方式。这样可以使得后续在顶层模块实例化各个子模块时结构更加清晰，同时不容易出错。尤其要注意关机状态不止可以通过按键实现，在手动模式中也会有可能关机。（侯芳旻）
- 在开始设计之前，要先规定好开发板上各个输入输出端口的使用功能，尽量避免后续的多次修改。同时对于开发板上的按键，要确定好按下是是高电平还是低电平。（侯芳旻）
- 在设计VGA模块时，在开始之前一定要详细了解VGA工作的具体原理，尤其是对于不同的分辨率和刷新率，VGA工作时的Pixel Clock，Hor Porch，V Porch 等参数也会有所不同，如果在一开始就把这些数值没有确定好，显示器就不会有信号输入。在设计的时候将这些定值都定义为一个单独的常数将会比较便于后续的修改。（侯芳旻）
- 在设计VGA部分代码的时候，由于VGA扫描的过程中扫描每个pixel的时间是固定的，必须注意VGA的两个输入信号hc, vc是一种时序逻辑这一特点。因而输入的RGB色彩的变化也应当由时序逻辑控制而非使用组合逻辑，否则会出现许多bug。同时要注意对于时序逻辑，在赋值时要使用nonblocking assignment.（侯芳旻）
- 在设计VGA中RGB色彩的输出时，发现一定要对数值的位数，进制及实际值做出严格的规定，不能有忽略，否则会出问题。（侯芳旻）
- 注意对于数值，MSB是位于最左侧的，所以在使用VGA进行数字的显示的时候，需要使显示器从左到右，从MSB到LSB依次显示对于的像素值。（侯芳旻）
- 在实例化模块的时候，要注意输入输出的位数是否与规定的一致，有时可能忘记规定位数而只进行了名称的定义，这种情况下虽然不会报错，但是会有bug出现。（侯芳旻）
- 在设计手动挡模块时，由于输入数据较多，节省时间想采用if-else 语句约束部分输入数据，结果导致逻辑较为紊乱，查找Bug也变得困难。用CaseX语句后逻辑较为清晰，代码正确性也大大提高（杨钰城）
- 在设计转向灯闪烁模块时，由于初步设计时没有采用状态机，对输入的数据直接进行处理，导致左灯与右灯只有一个灯有效，改用状态机后才解决这个问题。由此可见在编写verilog代码时要明白状态机的重要性与泛用性。（杨钰城）

- 由于断电操作可以发生在外部，也可以发生在手动挡内部，故通电与断电模块不能只考虑顶层。在设计时由于协同问题导致手动挡断电后外部依然是通电状态，处理这个问题耗费了大量时间，这是可以避免的，由此可见在设计前确认设计模块的功能是非常重要的。（杨钰城）
- 在里程显示模块中，使用了流水灯。这是实验课老师在课堂上讲解过的模块，由于没有重视复习，导致在编写该模块时遗忘了大量内容，导致重新学习。应该提高课堂效率。（杨钰城）
- 在模块设计时也会出现multiple driver等问题，但是在仿真等操作时系统又警告提醒，根据提醒能明确自己在哪个地方产生multiple driver，通过系统来de bug也是一个很好的选择。（杨钰城）
- 在项目完成初期时间安排较为不合理，在中期答辩前几天才开始项目的整体推进，压力很大。应该制定具体的时间表稳步推进项目。（杨钰城）
- 在设计半自动模块和全自动模块时，要尽量保持小车行驶在路中间，每当到达岔路口中间时再来判断状态是否应该改变，不能在检测器刚刚碰到墙时就做出反应，否则小车会越走越偏，同时产生各种bug。（杨祎勃）
- 为了保持小车行驶在路中间，时钟周期频率、转向时长、探测器延时时长、走过一个路口所需时长等数据都需要反复试验。（杨祎勃）
- 小车的自动行驶模式需要充分考虑各种情况，不能简单地依赖右手优先原则，比如遇到环形地图和死胡同的情况；小车放置信标的位置也需要充分考虑，怎样能起到标记作用，又不会堵住还没走过的路。

5.2 系统特色

- 本项目使用模块化编程，每个子模块都单独书写并在顶层实例化，结构非常清晰。我们的项目可以在通电状态下实现自动挡/手动挡/半自动挡的切换。并且在修改时可以直接修改对应功能的子模块，大大提高了开发效率。（杨钰城）
- 在完成该项目时我们加入了防抖功能，便于用户操作，用户体验较佳。同时每个模块都加入了重置按钮，使用户在操作上有更多的选择。（杨钰城）
- 在VGA显示中可以显示小车当前的4种状态，同时可以实现里程的记录，便于用户使用中获取输出信息。（侯芳旻）
- 小车的半自动模式增加了调头操作，半自动模式下操作者如果不慎将小车驶入死胡同，可以调头返回；此外，半自动模式操作者如果不慎操作失误，下达撞墙的指令，小车会自动掉头或者在墙之前停下，防止小车撞墙。（杨祎勃）
- 通过试验各种关键时间点的时长，小车在半自动模式和全自动模式下都可以保持一直走在道路中间。（杨祎勃）

5.3 优化方向

- VGA可以考虑实现更复杂的图像显示，如对于小车不同的状态有不同的文字显示等。目前的实现还比较基础。（侯芳旻）

- 在某些位置使用了if-else 语句，如果可能的话可以改为case。（侯芳旻）
- 本项目书写了大量分频器模块，这些模块其实可以用一个模块完成，通过更改参数就可以实现不同的分频输出。（杨钰城）
- 全自动模式小车驶出终点后，地图中小车曾经留下的障碍物可以自动清除掉，方便重新生成地图进行再次测试，也可以增加美观度。（杨祎勃）