

# 数字逻辑 project 报告


课程：数字逻辑

任课教师：刘佳琳

实验课教师：刘佳琳

成员：郑余奥泽、李子阳、张阳

贡献比：1：1：1

(视频见)  数字逻辑答辩vedio.mp4

## 一、开发计划

### 1. 小组选题

小车的全功能实现。

### 2. 成员分工

李子阳：手动、半自动、全自动模式的思路提供和设计提供，蜂鸣器的实现，答辩视频的剪辑。

张阳：手动、全自动模式的代码实现和细节更改，七段数码管的代码实现，VGA的像素补充。

郑余奥泽：手动、半自动模式的代码实现和细节更改，VGA的整体实现。

### 3. 执行记录

11月4日，实验课后，三人共同讨论选题，决定选择小车。

11月11日，实验课后，三人共同讨论整体思路，大致确定手动模式的思路。

11月18日，张阳试跑所给小车，发现连接总出问题，总是连接不上，试了一晚上总结出方法，再开一个小车 unity 窗口，才能正常运行。

11月19日，张阳提出将状态转移单独成模块实现使代码更简洁的想法，周末三人共同写出相应的真值表，以讨论不同情况。

11月26日，实验课后，晚上张阳根据真值表写出“trans”模块，输入当前状态以输出下个状态。但是调用起来与顶层 top 模块总是有冲突，导致状态根本没有发生转移。

12月5日，张阳汇报问题给组员，一同讨论，无果。

12月14日，三人共同整理设计资料，尽管“trans”模块无法正常调用，但是其能独立运行，可以满足手动驾驶需要，而且相关真值表齐全，代码也基本实现，于是将有关测试文件与现有资料整理，拍视频准备中期答辩。

12月20日，课上再次答辩，收到了老师的宝贵建议。课后，三人决定不再纠结于调用问题，李子阳先进行半自动和全自动的设计和思路开发，张阳和郑余奥泽继续当前进展。

12月25日前后，张阳和李子阳相继出现新冠症状，项目大任落到了郑余奥泽一个人头上。

1月1日，伴随着新年第一声钟声，郑余奥泽最终放弃了将状态转移模块化的想法，将状态转移写进了 top 文件，实现了小车的手动模式。病中的队友也为他感到庆幸。

1月2日，李子阳阳康，提供了半自动的设计给郑余奥泽，郑余奥泽紧接着写出了半自动的代码。

1月3日，张阳阳康，郑余奥泽转战 VGA，张阳修复了手动和半自动模式的 bug，

同时写出了七段数码管的代码。

1月4日，李子阳提供了全自动的设计想法给张阳，张阳紧接着写出了全自动模式的代码，调试之后，手动模式，半自动模式和全自动模式已经满足要求。

1月5日，郑余奥泽继续 VGA 的撰写。

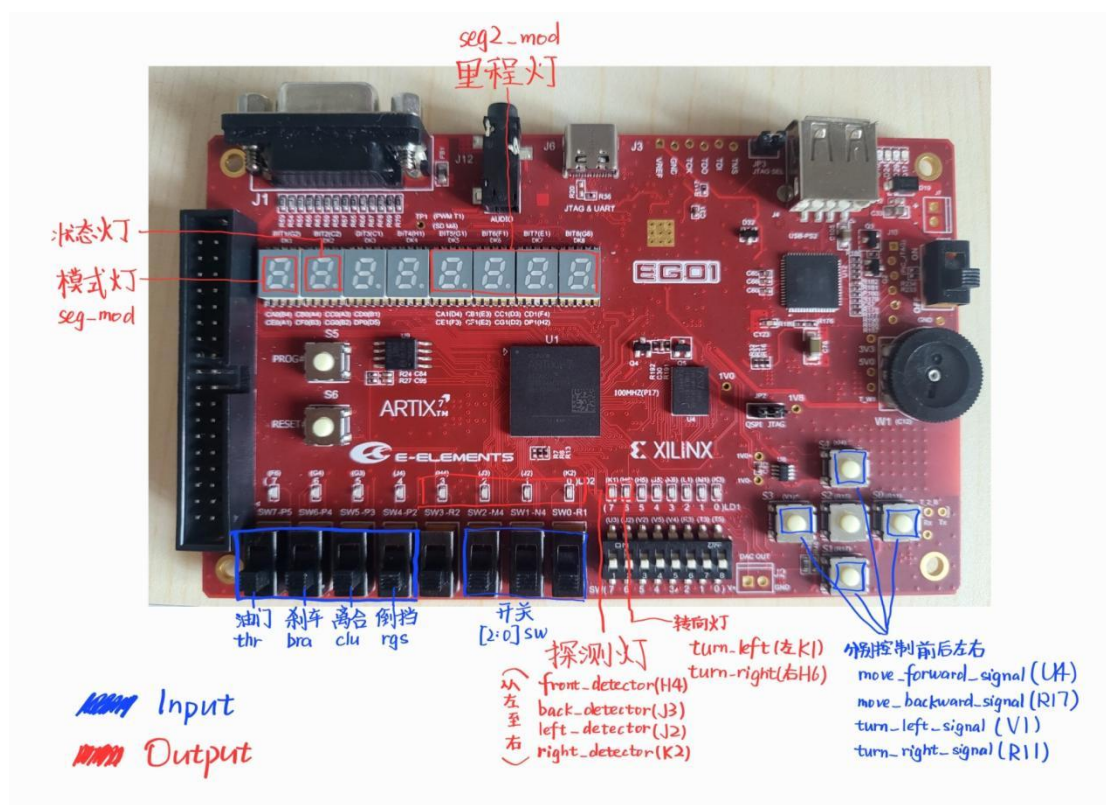
1月6日，张阳接受 VGA 的撰写，李子阳开始写蜂鸣器。

1月7日，小车全部功能实现，开始写报告和拍视频。

1月8日，李子阳剪辑完成。

## 二、使用文档

### 1. Ego1 图例



### 2. 使用说明

当前系统中：

开关 sw 是总开关，高电平有效，同时控制着小车的模式，“000”、“100”、“010”、“001”分别代表关机、手动模式、半自动模式、全自动模式，从左至右的三位分别对应 M4、N4、R1。

手动模式下的油门、刹车、离合、倒挡分别对应引脚 P5、P4、P3、P2，高电平有效；左转、右转分别对应引脚 V1、R11，高电平有效。

半自动模式下的前进指令、左转指令、右转指令、掉头指令分别对应引脚 U4、V1、R11、R17，高电平有效。

模式灯会显示当前所处模式，

关机模式下不显示；

手动模式下显示“B”意为“Basic”；

半自动模式下显示“S”，意为“Semi\_Auto”；

全自动模式下显示“A”，意为“Auto”。

状态灯会显示当前小车所处状态，

手动模式下，

关机状态下显示“O”，意为“Off”；

未起步状态下显示“n”，意为“no\_start”；

起步状态下显示“S”，意为“Start”；

前进状态下显示“F”，意为“Forward”；

后退状态下显示“B”，意为“Backward”；

半自动模式下，

关机状态下显示“O”，意为“Off”；

等待指令状态下显示“C”，意为“Command”；

前进状态下显示“F”，意为“Forward”；

左转状态下显示“L”，意为“Left\_Turning”；

右转状态下显示“r”，意为“right\_Turning”；

掉头状态下显示“o”，意为转圈；

全自动模式下，

关机状态下显示“O”，意为“Off”；

前进状态下显示“F”，意为“Forward”；

左转状态下显示“L”，意为“Left\_Turning”；


右转状态下显示“r”，意为“right\_Turning”；

掉头状态下显示“o”，意为转圈；

里程灯会显示小车当前里程，带有一位的小数位，只有小车在前进或后退时会记录，在关机时置零。

### 3. 系统结构设计

#### 1) 状态流程图

清晰大图见  小车状态转移图.png





端口：

```

) module SimulatedDevice(//统统高电平有效
    input sys_clk, //bind to P17 pin (100MHz system clock)
    input rx, //bind to N5 pin
    output tx, //bind to T4 pin
    input rst_de, //复位信号
    input turn_left_signal, //左转信号,
    input turn_right_signal, //右转信号,
    input move_forward_signal, //前进信号,
    input move_backward_signal, //后退信号
    //四个墙壁探测器
    output front_detector, //前探测器
    output back_detector, //后探测器
    output left_detector, //左探测器
    output right_detector, //右探测器

```



```

input [2:0]swi,//开关switch缩写
input thr,//油门
input clu,//离合
input bra,//刹车
input rgs,//倒车
output reg turn_left,//左转向灯
output reg turn_right,//右转向灯
output [7:0] seg_out, //模式灯和状态灯
output [3:0] seg_en,//使能端
output [7:0] seg1_out, //里程灯
output [3:0] seg1_en,//使能端

```

```

output hsync,
output vsync,
output [11:0]vga_rgb,//vga

```

```

output bu//蜂鸣器

```

```

);

```

```

parameter off = 4'b0000, no_st = 4'b0011, start = 4'b0111, movef = 4'b0110, moveb = 4'b0101;
//手动模式五个状态，分别为关机、未起步、起步、前进、后退（可控制左右方向）
parameter wait_command = 4'b1000, left_turning = 4'b1001, right_turning = 4'b1010, circle_turning = 4'b1011, keep_go = 4'b1110, semi_movef = 4'b1111;
//分别为等待指令、左转、右转、掉头、保持前进、不可控制左右方向的前进
parameter not_going = 3'b000, manual = 3'b100, semiAuto = 3'b010, auto = 3'b001;
//开关四个状态，关机、手动、半自动、全自动
parameter one_second = 27'd10000_0000, zero = 27'b0000_0000_0000_0000_0000_0000;
parameter one_second1920 = 27'd9500_0000, one_second45 = 27'd8000_0000, one_second180 = 28'd18000_0000;
//参数，在100MHz下的一秒钟、零（用于重置计数器）
reg [0:0] place_barrier_signal; //放信标
reg [0:0] destroy_barrier_signal; //摧毁信标
reg [27:0] keep_cnt = zero; //保持前进的计数器（通过它实现保持前进一段时间）
reg [26:0] cnt = zero; //开机的计数器（通过它实现开开关一秒后开机）
reg [7:0] in = 8'b00000000; //变成了reg类型，因为要后面更改in的值，实现小车各种功能，模板自带的
reg [27:0] ti; //time的缩写，用它约束保持前进的时间，它设置多久，保持前进就多久
reg [0:0] place; //用于保持前进后放不放信标，1为放，0为不放，默认不放
wire [7:0] rec; //传递结果，已经帮我们写好了，通过它获取墙壁检测器的值
reg [3:0] stat = off; //状态
reg [27:0] turn_cnt = zero; //转向所用计数器

```

```

seg_mod se1(sys_clk, swi, stat, seg_out, seg_en);
seg2_mod se2(sys_clk, stat, seg1_out, seg1_en);
uart_top md(.clk(sys_clk), .rst(0), .data_in(in), .data_rec(rec), .rx(rx), .tx(tx));
vga V_G_A(sys_clk, rst_de, state, swi, hsync, vsync, vga_rgb);
buzzer Buzzer(sys_clk, stat, bu);



```

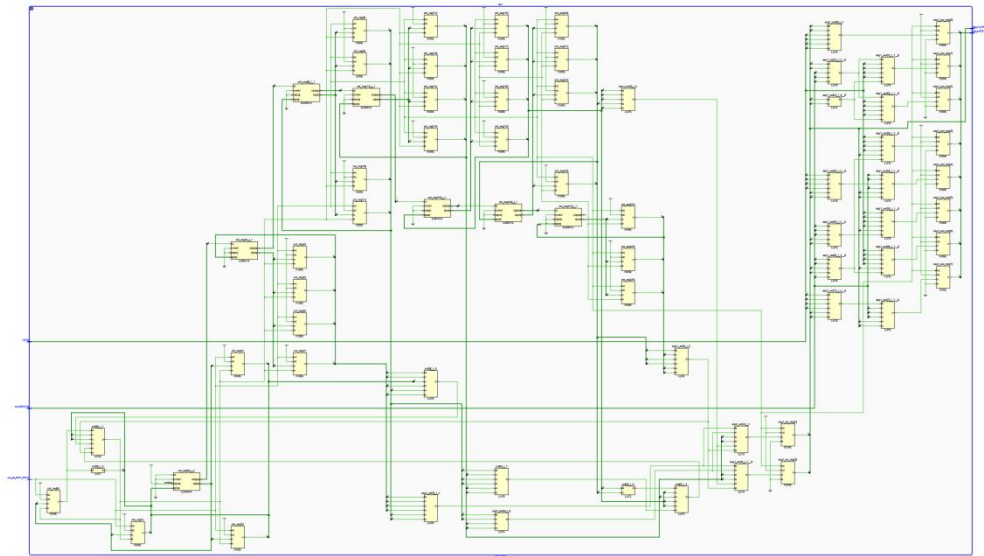
### 3）各子模块的功能、输入输出端口规格说明及结构图

#### ①seg\_mod se1

功能：七段数码管的模式显示和状态显示

设置两个时间差 c、h，cnt 计数到达不同时间差时切换使能端 seg1\_en，切换数码管，由于时间差很小，人眼分辨不出，便会觉得是两个管都在一直亮。通过 top 模块传递进来的开关 sw 和状态 state 做判断，让七段数码管 seg1\_out 显示相应状态以及模式。

结构：清晰大图见  schematic\_seg\_mod.pdf ，代码见  seg\_mod.v



端口：



```
module seg_mod(
input clk,//连接时钟
input [2:0]sw,//连接开关
input [3:0]state,//当前状态
output reg [7:0] seg1_out,//七段数码管
output reg [3:0] seg1_en//使能端
);
//与top文件里的状态一致
parameter off = 4'b0000, no_st = 4'b0011, start = 4'b0111, movef = 4'b0110, moveb = 4'b0101;
parameter wait_command = 4'b1000, left_turning = 4'b1001, right_turning = 4'b1010, circle_turning = 4'b1011, keep_go = 4'b1110, semi_movef = 4'b1111;

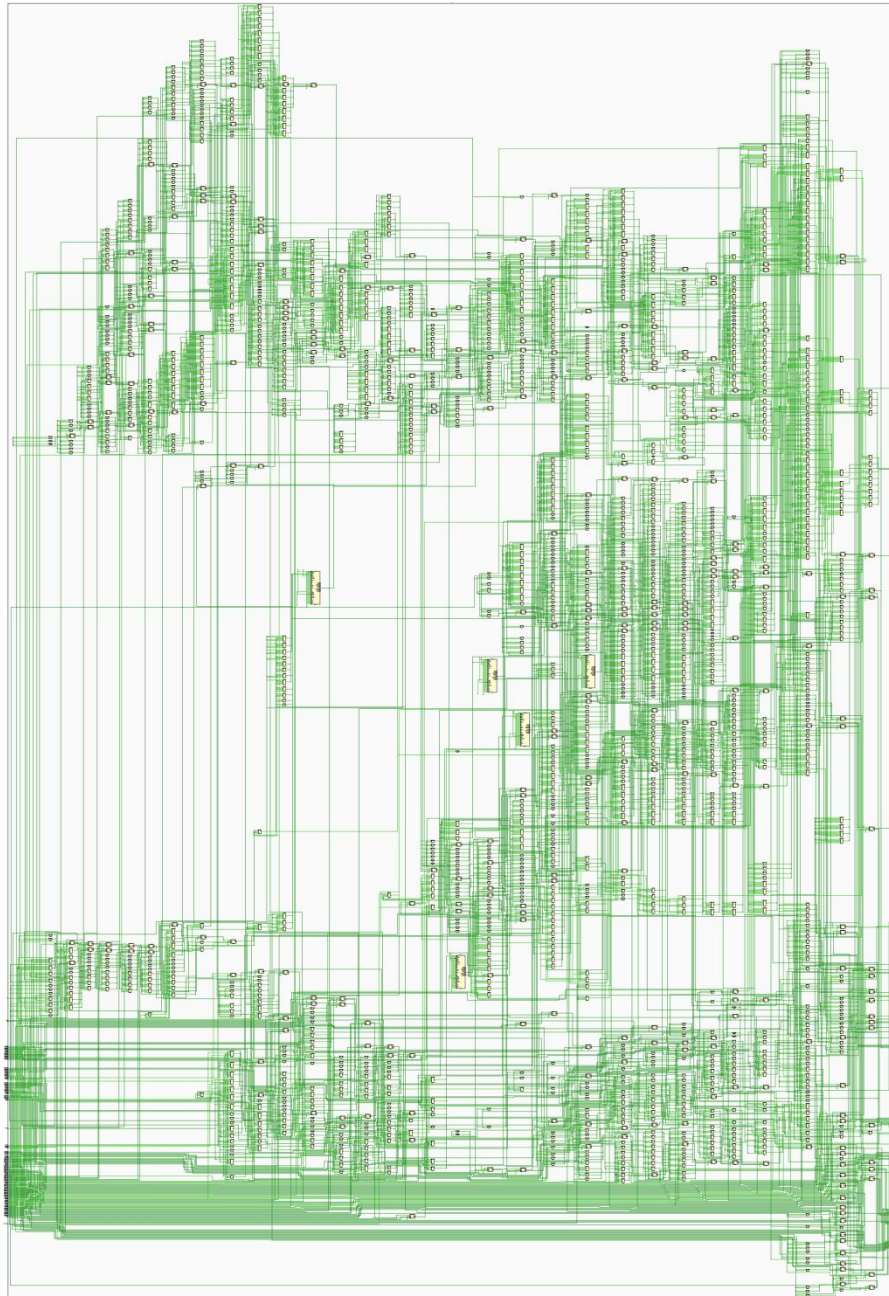
//切换显示灯的时间
parameter c = 27'd10000_0, h = 27'd5_0000, zero = 27'b0;
reg [26:0] cnt = zero;
```

## ②seg2\_mod se2

功能：七段数码管的里程显示

设置四个时间差 f、o、t、tr，换管计数器 sh 计数到达不同时间差时切换使能端 seg1\_en，切换数码管，由于时间差很小，人眼分辨不出，便会觉得是四个管都在一直亮。在小车移动状态时，里程计数器 cnt 会在每个时钟上升沿加 1，不同使能端时，通过公式计算个、十、百、小数位的呈现数字，让七段数码管 seg1\_out 显示相应数字，实现在小车移动时，将路程转化为时间，记得里程。

结构：清晰大图见  schematic\_seg1\_mod.pdf ，代码见  seg2\_mod.v



端口：

```

module seg2_mod(
input clk,//连接时钟
input [3:0]state,//当前状态
output reg [7:0] seg1_out,//七段数码管
output reg [3:0] seg1_en//使能端
);
//与top中状态一致
parameter off = 4'b0000, no_st = 4'b0011, start = 4'b0111, movef = 4'b0110, moveb = 4'b0101, keep_go=4'b1110, semi_movef=4'b1111;

//四个灯切换的时间
parameter f = 30'd13200_00, o = 30'd9900_00, t = 30'd6600_00, tr = 30'd3300_00, zero = 30'b0;

//一秒、十秒、一百秒、0.1秒
parameter one = 37'd10000_0000, ten = 37'd10000_0000_0, s = 37'd10000_0000_00, dian = 37'd10000_000;
reg [36:0] cnt = zero;//里程计数器
reg [29:0] sh = zero;//换管计数器


```

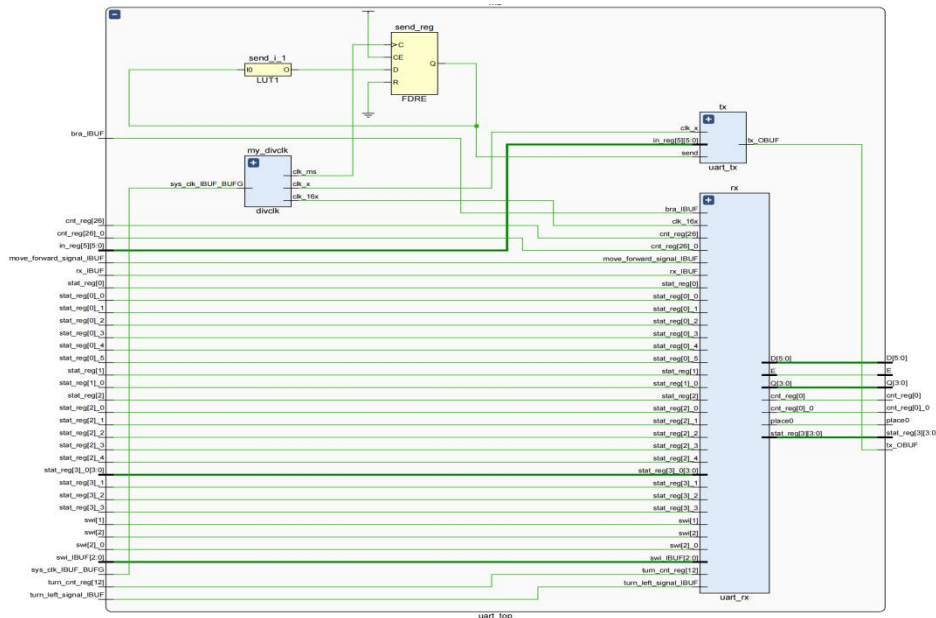


### ③uart\_top md

自带 module，未更改

结构：清晰大图见

 schematic\_uart.pdf



### ④vga V\_G\_A

功能：vga 模块，用于模式显示、状态显示和里程显示

该模块会接收顶层模块中的时钟信号、复位信号与表示小车当前状态的 state 信号与 swi 信号。：

该模块会将时钟信号传入至 Get25Clk 模块中以获得 25MHz 的时钟信号。之后的 vga\_draw 与 vga\_control 模块都将调用该 25MHz 的时钟信号,以实现相应的时序逻辑电路；

State 信号与 swi 信号会被 vga 模块传入到 vga\_draw 模块中，以确定当前所应当绘制的模式显示图像与状态显示图像。

vga 子模块：vga\_control 模块

在 vga\_control 模块中，将接收 vga 模块传入的时钟信号，复位信号，pix\_data，pix\_x,pix\_y,hsync,vsync,vga\_rgb 信号。通过在 vga\_control 中依据对显示屏幕的扫描确定出当前所扫描的像素点的坐标，并依据该像素点是否处于合法显示范围内，将存储于 pix\_data 中的绘制信息传入到 vga\_rgb 中和确定 hsync 与 vsync 是否被激活。据此，实现了连接 vga 的显示器的功能。

vga 子模块:Get25Clk 模块

该模块将接收 vga 传入的 sys\_clk(100MHz)时钟信号，然后输出 rgb\_clk(25MHz) 时钟信号。

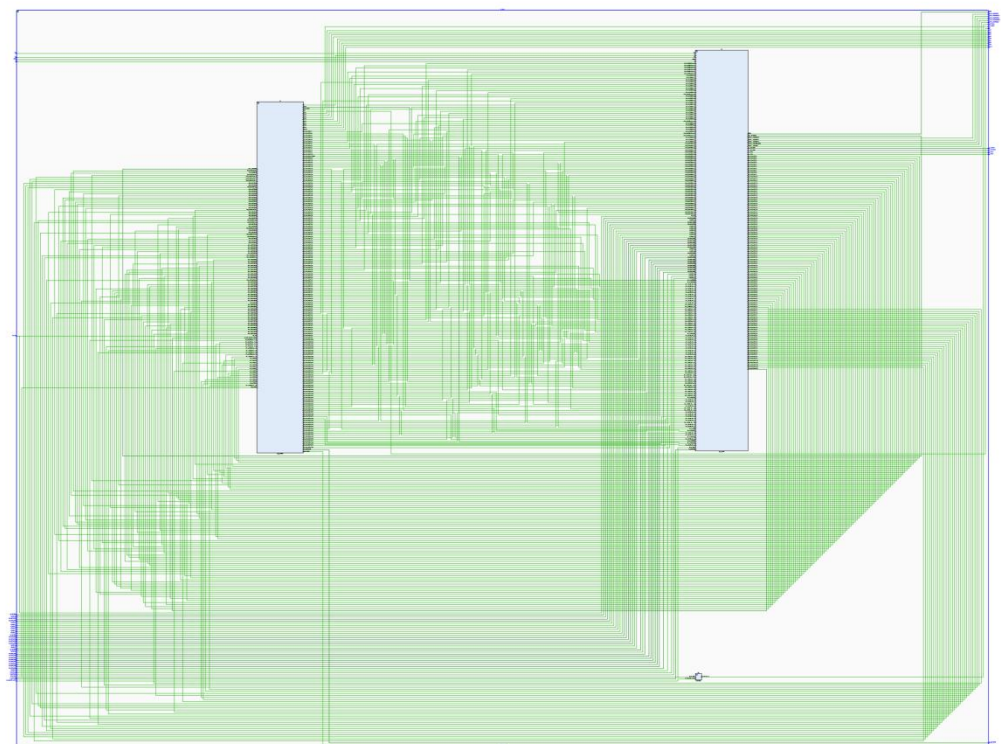
vga 子模块：vga\_draw 模块

该模块将接收 vga 模块中传入的 pix\_x , pix\_y 坐标信号，根据模块中的 reg

[]char[]确定出相应的像素点的 `pix_data` 信号，然后将该信号传至 `vga` 模块中，再由 `vga` 模块传入 `vga_control` 模块中将绘制信息传入 `vga_rgb` 中，完成像素点的实际绘制。

由于 `char` 遍布整个显示区域，虽然“模式：”“状态：”“里程：”几个字不会变，但是显示的模式、状态、里程会变，所以多创建了几个小一些的 `reg`，将 `char` 模块化，再用这些 `reg` 给 `char` 相应区域赋值，实现不同区域在不同模式、不同状态、不同里程下显示不同东西，实时变化。需要显示的像素字模，由 `pctolcd2002` 工具生成，需要大量撰写的 Verilog 语句，由自己编写的一个 `java` 程序生成，大大减少了写代码时间。

结构：清晰大图见  `vga.pdf`，代码见  `Vga.v`



端口：

```

module vga(//clk 是25MHZ
input wire sys_clk,//100Mhz时钟
input wire sys_rst,//reset
input wire[3:0] stat,//当前状态
input wire [2:0]swi,//开关

output wire hsync,
output wire vsync,
output wire [11:0]vga_rgb
);
wire [9:0] pix_x;
wire [9:0] pix_y;
wire [11:0] pix_data;
wire clk_25m;
vga_control vc(
clk_25m , sys_rst , pix_data , pix_x , pix_y , hsync, vsync , vga_rgb
);
vga_draw vd(clk_25m , sys_rst , stat , swi , pix_x , pix_y , pix_data );
Get25Clk Gc(sys_clk ,sys_rst ,clk_25m);
endmodule

```

vga\_control 模块:

```

22 module vga_control(
23   input wire vga_clk,//所用的时钟信号
24   input wire sys_rst,//rst复位信号
25   input wire [11:0] pix_data,//存某点的像素信息
26   output reg [9:0] pix_x,//所扫描的像素的x坐标
27   output reg [9:0] pix_y,//所扫描像素的y坐标
28   output wire[0:0] hsync,//行同步信号，相关端口会绑在vga的端口上。
29   output wire[0:0] vsync,//场同步信号，相关端口会绑在vga的端口上。
30   output reg [11:0] vga_rgb//图像的绘制色彩信息，相关端口会绑在vga的端口上。
31 );
32
33 parameter H_SYNC = 10'd96; //行同步周期
34 parameter H_BA=10'd48;//行后延
35 parameter H_VA = 10'd640; //合法显示部分
36 parameter H_TO=10'd800;//行扫描周期
37
38 parameter V_SYNC = 10'd2; //场同步周期
39 parameter V_BA=10'd33;//场后延
40 parameter V_VA = 10'd480; //合法显示部分
41 parameter V_TO=10'd525;//场扫描周期
42
43 reg[9:0] cnt_h=10'b00000_00001;//用于记录行扫描位点
44 reg[9:0] cnt_v=10'b00000_00001;//用于记录场扫描位点
45
46
47

```

```

always@(posedge vga_clk,posedge sys_rst)begin//探测目前的扫描状况。行扫描每完成一个周期，场扫描位点前进一位
if(sys_rst==1'b1)begin
    cnt_v<=10'b00000_00001;
end
else if( (cnt_h==H_TO)&&(cnt_v < V_TO ) ) begin
    cnt_v<=cnt_v+10'b00000_00001;
end
else if( (cnt_h ==H_TO) &&(cnt_v == V_TO ) ) begin
    cnt_v<=10'b00000_00001;
end
else begin
    cnt_v <=cnt_v;
end
end

always@(posedge vga_clk,posedge sys_rst)begin
    if(sys_rst==1'b1)begin
        cnt_h<=0;
    end
    else if(cnt_h==(H_TO) ) begin
        cnt_h<=10'b00000_00001;
    end
    else begin
        cnt_h<=cnt_h+10'b00000_00001;
    end
end
end

```

Get25Clk 模块:

```

module Get25Clk(input sys_clk,input rst,output reg clk_25m);//分频器
//将顶层模块中所用的100MHz分频为25MHz

parameter period =4;
reg[3:0] cnt;
always@(posedge sys_clk,posedge rst)begin
    if(rst)begin
        cnt<=0;
        clk_25m<=0;
    end
    else
        if(cnt==((period>>1)-1)) begin
            clk_25m <=~clk_25m;
            cnt<= 0;
        end
        else begin
            cnt<=cnt+1;
        end
    end
endmodule

```

vga\_draw 模块: (此模块内未展示的 always 模块中的内容为分配各个像素点的色彩信息的代码, 长度有 4000+行且重复度高, 此处不详细展示)

```

1 module vga_draw(//25MHz,该部分用于绘制显示的图像; 会将x,y对应的坐标点的像素绘制信息传入pix_data中
2
3 input wire vga_clk,
4 input wire sys_rst,
5 input wire [3:0]state,
6 input wire [2:0]swi,
7
8
9 input wire [9:0]pix_x,
10 input wire [9:0]pix_y,
11 output reg [11:0] pix_data
12 );
13
14 parameter off=4'b0000, no_st=4'b0011, start=4'b0111, movef=4'b0110, moveb=4'b0101;
15 //手动模式五个状态, 分别为关机、未起步、起步、前进、后退 (可控制左右方向)
16 parameter wait_command=4'b1000, left_turning=4'b1001, right_turning=4'b1010, circle_turning=4'b1011, keep_go=4'b1110, semi_movef=4'b1111;
17 //分别为等待指令、左转、右转、掉头、保持前进、不可控制左右方向的前进
18 parameter black=12'h000, blue=12'h00f, white=12'hfff;
19 reg [255:0] char0 [63:0];
20 reg [255:0] char1 [63:0];
21 reg [255:0] char2 [63:0];
22 reg [255:0] char00 [63:0]; //手动
23 reg [255:0] char00_1 [63:0]; //半自动
24 reg [255:0] char00_2 [63:0]; //全自动
25 reg [255:0] char11 [63:0]; //关机
26 reg [255:0] char11_1 [63:0]; //未起步
27 reg [255:0] char11_2 [63:0]; //起步
28 reg [255:0] char11_3 [63:0]; //移动
29 reg [255:0] char11_4 [63:0]; //等待指令
30 reg [255:0] char11_5 [63:0]; //转向
31 reg [255:0] char11_6 [63:0]; //掉头
32 reg [31:0] char22_0 [63:0]; //0
33 reg [31:0] char22_1 [63:0]; //1
34 reg [31:0] char22_2 [63:0]; //2
35 reg [31:0] char22_3 [63:0]; //3
36 reg [127:0] char22_4 [63:0]; //4
37 //reg [95:0] char22_5 [63:0]; //5
38
39 reg [511:0] char [479:0];
40 parameter onekm=37'd2500_0000, tenkm=37'd25000_0000, hkm=37'd250000_0000, diankm=37'd250_0000;
41 reg [36:0] li;
42 always@(posedge vga_clk) begin
43 case(state)
44 off: li<=0;
45 movef: li<=li+1'b1;
46 moveb: li<=li+1'b1;
47 semi_movef: li<=li+1'b1;
48 keep_go: li<=li+1'b1;
49 default: li<=li;
50 endcase
51 end
52
53 always@(posedge vga_clk) begin...
54
55 always @(posedge vga_clk, posedge sys_rst) begin
56 if(sys_rst)begin
57 pix_data<=black;
58 end
59 else if (pix_x<512 && char[pix_y][10'd255-pix_x]==1'b1)
60 //if (pix_x>=416 && pix_y>=128) pix_data<=white;
61 else *pix_data<=blue;
62 else
63 pix_data<=white;
64 end
65 endmodule

```

## ⑤buzzer Buzzer

功能：蜂鸣器的实现，模拟小车起步音

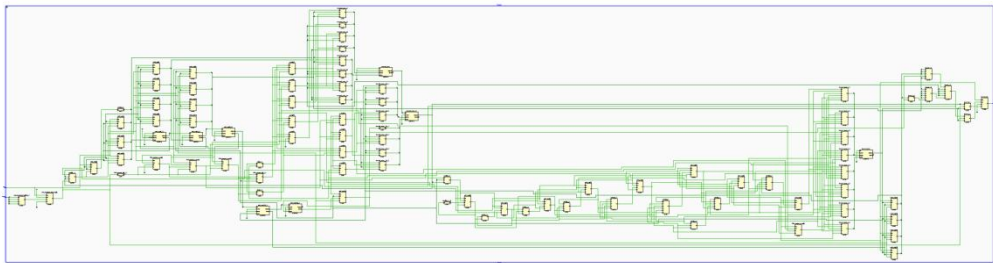
根据状态决定蜂鸣器的响声。蜂鸣器能被人听到首先需要一段特定的旋律，并且旋律需要反复重复一段时间。这里的旋律由一个旋律数字决定，y0、y1、y2、y3是四段旋律数字的一半部分，在数字前半段蜂鸣器输出1，后半段输出0，重复这个过程特定次数，四段旋律的重复次数由c0、c1、c2、c3决定，能够模拟小车发动声音。

模块内用一个 always 根据当前状态决定蜂鸣器是否开始奏响将 state\_transform\_front 赋值为1或0，另一个 always 内在确定要奏响时，旋律计数器 cnt0 计数加1，到达旋律数字 y 一半之前蜂鸣器输出1，后一半输出0，走完一次旋律，另一个旋律重复次数的计数器 cnt1 会加一，同时旋律数字的计数器 cnt0 会重新置零，开始下一次旋律重复。当这段旋律重复次数到达 c 完成时，y 切换下



一段旋律数字，c 会切换下一段旋律的重复次数，开始下一段旋律的重复。第四段旋律重复完成为一秒钟，蜂鸣器能发出一秒的起步音。

结构：清晰大图见  buzzer.pdf ，代码见  buzzer.v






端口：

```
//以下为小车用蜂鸣器
module buzzer(
    input clk, //时钟输入
    input [3:0] state, //当前状态
    output reg buzzer //驱动蜂鸣器
);

parameter off = 4'b0000, no_st = 4'b0011, start = 4'b0111, movef = 4'b0110, moveb = 4'b0101;
//手动模式五个状态，分别为关机、未起步、起步、前进、后退（可控制左右方向）
parameter wait_command = 4'b1000, left_turning = 4'b1001, right_turning = 4'b1010, circle_turning = 4'b1011, keep_go = 4'b1110, semi_movef = 4'b1111;
//分别为等待指令、左转、右转、掉头、保持前进、不可控制左右方向的前进
parameter y0 = 16'b1011011110011000, y1 = 16'b1010010000010000, y2 = 16'b0111100100011000, y3 = 16'b0110000110101000;
//四段旋律
parameter c0 = 8'b01111001, c1 = 8'b0111101, c2 = 8'b10110100, c3 = 8'b11100110;
//四段旋律分别重复次数
reg [15:0] y; //旋律
reg [7:0] c; //次数
reg [0:0] stat_transform_front; //状态决定1/0
reg [16:0] cnt0; //计数每个音符对应的时序周期
reg [7:0] cnt1; //计数每个音符重复次数
```

## ⑥相关工具

字模生成工具：  pctolcd  PCtoLCD2002.exe

生产语句的 java 代码：  trans.java  give.java

## 4. 开发过程中间的经验总结及优化

郑余奥泽：

在此次数字逻辑的 project 开发中，我主要负责了手动档、半自动档与 vga 的连接设置等方面的开发。我的手动档与半自动档的开发是建立在我们团队中期答辩前，对手动档驾驶编写的失败经验之上的。我们初期开发时，由于对 verilog 的时序电路和各模块间关系理解不够深入，在编写过程中仍带有明显的软件开发思维，而引发了小车无法正常开机启动与多驱动问题等严重 bug。

在理清各模块间的时序关系后，手动档驾驶开发成功。我认为，verilog 的多模块开发应当着重关注时序问题与多模块间对某一数值的操作关系。此外，在使用 always 语句时，需要避免在不同的 always 模块中对同一个 reg、wire 类型的数据做赋值，以避免多驱动 bug。

在 vga 的连接配置上，我认为理解行、场间的周期变化是至关重要的。在理解行、场等

对应的理论知识后，还需要注意分频器的编写，要严格依据 `vga` 的相关参数处理好分辨率、刷新率与时钟信号频率间的关系。同时，在将 `vga` 模块接入 `global` 模块时，还需注意两个模块间由于所用时钟信号频率不同而易引起其它 `reg`、`wire` 等数据值发生变化。我们在开发中，由于未考虑不同模块使用的时钟信号频率不同而直接将 `global` 模块中的 `input reg stat`（用于表示小车当前状态）传入到了 `vga` 模块中，导致小车在启动后，四个方向的障碍物探测器 `detector` 信号持续为“1”。之后，通过增加中间量，规避不同时钟信号的模块调用同一个数值才解决了这一 `bug`。

总结：我认为 `verilog` 开发中最重要的是摒弃软件开发思维，时时刻刻关注各模块间的时序逻辑与相互影响作用。

李子阳：

我在 `project` 中的分工主要是前期的真值表的设计与填写，以及蜂鸣器的设计与代码构造。做 `project` 给我最直观的感受就是，即使在理论层面上对于设计已经做到天衣无缝，用代码和软件实现起来也会遇到意想不到的阻碍。反复确认了自己的状态设计图，以及半自动和自动的感应器设计后，队友在后续代码实现中也依旧需要不断的调试与改进。我在做蜂鸣器时理解并设计出蜂鸣器的流程后，后续也并不是一帆风顺的。无论是调试音符的周期还是时钟信号的个数，都是很复杂的。而为了让蜂鸣器可以发出连续的从低到高的音调，也是实在令人煞费苦心。但是通过这次 `project`，可以让我更深刻的理解数字电路设计。用程序模拟小车尚且有这么多问题，要是做真小车，真不能预料会有多少麻烦。

张阳：

`Project` 不能一蹴而就，要付出足够的努力才可能有收获。在我们学习理论的同时，实际应用起来却存在和理论不符的许多地方，这时不能局限于已有圈子，而要想办法去解决。最典型的例子就是全自动模式下小车遇到岔路口时，由于小车游戏的一些小缺陷，总是把岔路口识别成为死胡同，引起难以预料的反应。但是通过多加 `keep_go` 状态，使小车前进一段距离后再进行岔路口判断，便能够成功识别。这启示我们，在现有条件下，利用好已有的东西去想办法解决问题才是关键。同时，对于 `keep_go` 状态的应用，有了新优化。通过一个时间变量和一个“放置”变量，动态控制 `keep_go` 的持续时间和放置信标与否，能够很好的应用这个状态，具有很好的适用性。

在测试转弯时间以及放置信标时间时，要有足够耐心，不断调试，找到最佳时间；在 `vga` 撰写时，要通过不断测试，找到字符的排列方式，才能最终实现，撰写字符字模和大量代码时，要学会通过 `java` 生成，活学活用，大大提高速度。我从中获得的耐性，也让我颇为感慨。

项目做到如火如荼时我恰巧感染新冠，多谢老师们的体谅和队友的强有力支持，才能做成现在这个样子。再次感谢！