

CS207 Digital Logic

PROJECT REPORT A Real Car

- 小组成员：张天悦（学号：12112908） 罗嘉诚（学号：12112910）
- 完成日期：2023 年 01 月 08 日

1 项目开发计划

1.1 小组选题

在进行项目选题时，我们非常慎重地比较了两个选题 **A Real Car** 以及 **Self-service Washing Machine**。

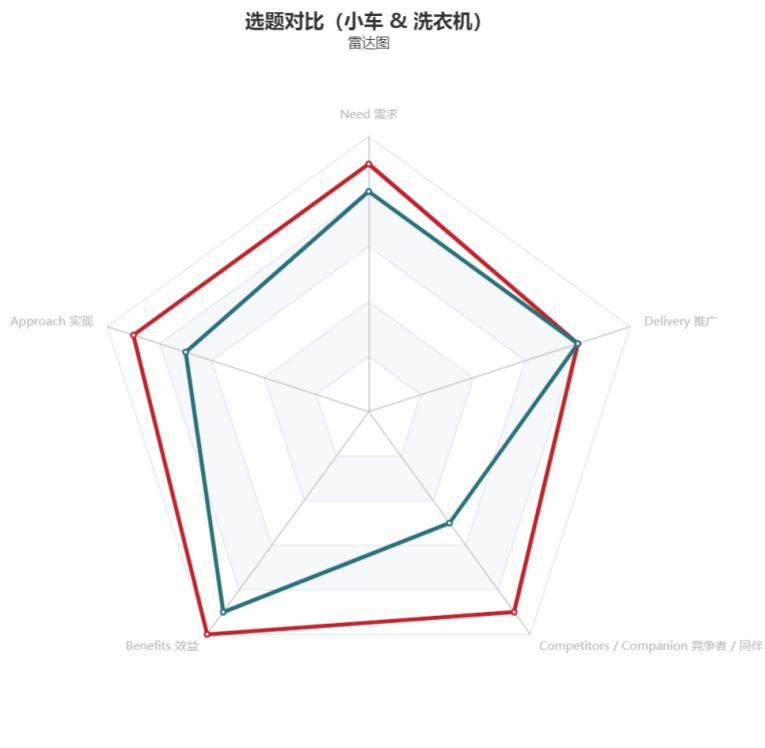
我们在选组选题时主要遵循了 **NABCD** 原则（即 Need 需求、Approach 实现、Benefits 效益、Competitors / Companion 竞争者 / 同伴、Delivery 推广）进行需求分析。

基于小组实际，选题时我们认为前者 **A Real Car** 相较于后者 **Self-service Washing Machine**，有如下优势，更加适合本小组开展工作：

- Need 需求：2022 年秋季学期，**数字逻辑**课程的期末项目，需要高效高质量地完成两个指定项目（**A Real Car** 或者 **Self-service Washing Machine**）中的一个。
- Approach 实现：小车项目具有较为完善的测试环境，并且通过 **UART 模块和相关模拟器**，可通过应用程序 **DriveCar.exe** 显示项目的**实际效果**，可以很直观地显示项目效果，很有意思。
- Benefits 效益：小车项目简洁实用，更加**贴近生活**，功能直接了然，容易理解，能让小组成员快速上手，**进行高效率工作**。这也为更好地完善小车项目留出充足的时间，获得**更高的分数**。
- Competitors / Companion 竞争者 / 同伴：根据前期了解，在选择项目时选择 **A Real Car** 项目的小组**更多**，若尝试选择人数较多的项目，便于组间的**交流和学习**，也能在期末演示中看到更多优质项目。这能更好的**提高自己的能力**。
- Delivery 推广：我们的项目在提交后，就在 **GitHub** 中开源了，由于小车项目有较为完善的测试环境，可以让**更多的朋友**看到我们的项目，并且**便于测试**，能更好的发现项目中的不足。

根据以上分析，我们绘制了两个项目（小车项目 **A Real Car**、洗衣机项目 **Self-service Washing Machine**）选题对比的雷达图，以此为依据最终确定小组项目选题。

选题对比	最大值 Maximum	小车项目		洗衣机项目	
		A Real Car	Self-service Washing Machine	A Real Car	Self-service Washing Machine
Need (需求)	10	9		8	
Approach (实现)	10	9		7	
Benefits (效益)	10	10		9	
Competitors / Companion (竞争者 / 同伴)	10	9		5	
Delivery (推广)	10	8		8	



基于以上分析，我们小组慎重地选择了 **A Real Car** 作为 2022 年 秋季学期 **数字逻辑** 课程的期末项目。

1.2 成员分工

在筹建项目小组时，我们希望小组人数较少，这能很方便地进行项目同步协作，同时也能避免摸鱼摆烂现象的发生。在同步协作方法上，我们小组主要通过 **GitHub** 中的相关仓库进行项目同步，非常方便高效。

同时，由于参与开发的小组成员较少，每位小组成员的工作压力都较大，需要完成相当数量的工作。所以更加需要科学安排每位小组成员的任务，高效率、高质量地完成本次项目。

在考虑个人贡献分配时，我们兼顾效率和公平，从如下 5 个方面，对每位小组成员在团队中的贡献进行衡量：

- 工作数量（成员在团队项目中所投入的精力）
- 工作质量（避免低效拖沓，最重要的是代码是否能正常运行，并且做到 **Bug Free**）
- 工作参考价值（工作成果对于整个团队项目进度的推进程度、对于他人是否积极提供帮助互相进步等）
- 工作投入程度（成员在项目中的用心程度和态度）
- 其他特别贡献（对整个团队有特殊贡献的部分）

根据以上原则，兼顾效率和公平，小组成员达成一致，最终的项目贡献分配比为：

小组成员	贡献比	评价
张天悦 (学号: 12112908)	50%	高效高质量地完成所有既定工作，并且工作对团队有重要参考价值，工作态度和投入程度佳，在实现项目 VGA 功能 (Bonus) 对项目有特别贡献。
罗嘉诚 (学号: 12112910)	50%	高效高质量地完成所有既定工作，并且工作对团队有重要参考价值，工作态度和投入程度佳，在实现项目 Auto Driving 功能 (Bonus) 对项目有特别贡献。

具体成员分工如下：

小组成员	具体分工
张天悦 (学号: 12112908)	<ul style="list-style-type: none">① 全局状态控制 (Global State) 20%② 手动挡位 (Manual Driving) 离合、刹车、油门、里程表控制 30%③ VGA 显示 (Bonus) 20%④ 前期设计、资料收集、项目测试、报告写作、视频录制等
罗嘉诚 (学号: 12112910)	<ul style="list-style-type: none">① 手动挡位 (Manual Driving) 倒挡、左转、右转、UART 连接 20%② 半自动挡位 (Semi-Auto Driving) 30%③ 全自动挡位 (Auto Driving, Bonus) 20%④ 前期设计、资料收集、项目测试、报告写作、视频录制等

1.3 执行记录

我们以项目进行过程中实现的**重要功能**、完成的**重要事件**节点为单位，项目的执行记录，汇总如下：

时间	执行人	事件
2022.11.11	罗嘉诚、张天悦	项目组队
2022.11.16	2022 Fall 数字逻辑教学团队	项目发布
2022.11.17	张天悦	填写项目组队在线表格（确认组队）
2022.11.18	罗嘉诚、张天悦	第 1 次 Lab 课讨论 (EGO1 开发板领取、讨论课程项目选题问题、后续大致时间安排)
2022.11.25	罗嘉诚、张天悦	第 2 次 Lab 课讨论 (确定项目选题为：A Real Car，规划中期答辩前需要完成内容： 全局状态控制（Global State）、手动挡位（Manual Driving）)
2022.11.27	张天悦	完成全局状态控制（Global State）20% 模块的 Verilog 代码编写、仿真
2022.11.28	张天悦	完成全局状态控制（Global State）20% 模块的上板测试，功能正常
2022.11.29	罗嘉诚	明确 UART 连接的方法，实现一些简单测试功能、能实现与应用程序交互
2022.11.30	罗嘉诚	完成左转、右转相关功能的 Verilog 代码编写、仿真
2022.11.31	张天悦	完成离合、刹车、油门相关功能的 Verilog 代码编写、仿真
2022.12.02	罗嘉诚、张天悦	第 3 次 Lab 课讨论 (项目进度汇总、项目整合后进行上板测试，功能正常)
2022.12.04	张天悦	完成里程表相关功能的 Verilog 代码编写、仿真
2022.12.05	张天悦	完成里程表相关功能的上板测试，功能正常
2022.12.08	罗嘉诚	完成全局状态控制（Global State）、手动挡位（Manual Driving） 两部分进行功能整合
2022.12.09	罗嘉诚、张天悦	第 4 次 Lab 课讨论 (上板测试项目已经完成部分，全局状态控制（Global State）、 手动挡位（Manual Driving）功能正常，讨论中期答辩相关安排)
2022.12.13	2022 Fall 数字逻辑教学团队	更改项目期中答辩形式为视频录制，对于周五 Lab 课小组 DDL 提前
2022.12.14	罗嘉诚、张天悦	明确中期答辩形式为展示 40% 的上板功能，进行上板功能测试，测试正常
2022.12.15	罗嘉诚、张天悦	拍摄、剪辑中期答辩演示视频，并如期提交中期答辩相关材料
2022.12.16	2022 Fall 数字逻辑教学团队	第 5 次 Lab 课中期答辩 (向王薇老师进行项目中期答辩，被提出一些重要的问题和建议)
2022.12.20	罗嘉诚	针对中期答辩中出现的问题，修改完善代码，进行上板功能测试，测试正常
2022.12.23	2022 Fall 数字逻辑教学团队	中期答辩成绩出炉：5 / 5
2022.12.24	罗嘉诚	完成半自动挡位（Semi-Auto Driving）30% 的 Verilog 代码编写，仿真
2022.12.25	罗嘉诚	完成半自动挡位（Semi-Auto Driving）30% 的上板测试，功能正常
2022.12.26	2022 Fall 数字逻辑教学团队	通知项目延期约 1 周，更改最终项目展示形式为视频录制、报告撰写
2022.12.29	张天悦	完成 VGA 显示（Bonus）20% 的 Verilog 代码编写，仿真
2022.12.30	张天悦	完成 VGA 显示（Bonus）20% 的上板测试，功能正常
2022.12.31	罗嘉诚	完成全自动挡位（Auto Driving, Bonus）20% 的 Verilog 代码编写，仿真
2023.01.01	罗嘉诚	完成全自动挡位（Auto Driving, Bonus）20% 的上板测试，功能正常

时间	执行人	事件
2023.01.02	罗嘉诚	当前已经完成项目部分（全局状态控制（Global State）20%、手动挡位（Manual Driving）50%、半自动挡位（Semi-Auto Driving）30%、全自动挡位（Auto Driving, Bonus）20%）整合后进行上板测试，所有功能均正常
2023.01.03	张天悦	项目所有部分（全局状态控制（Global State）20%、手动挡位（Manual Driving）50%、半自动挡位（Semi-Auto Driving）30%、全自动挡位（Auto Driving, Bonus）20%、VGA 显示（Bonus）20%）整合后进行最终上板测试，所有功能均正常

1.4 开源

本项目已经基于 [MIT License](#) 协议在 [GitHub](#) 上开源了，您可以进入项目仓库查看我们的项目。

项目仓库地址：https://github.com/Maystern/SUSTech_DigitalLogic_Project_a-real-car。

使用方法：

1. `git clone https://github.com/Maystern/SUSTech_DigitalLogic_Project_a-real-car.git`
2. 进入获得的 `SUSTech_DigitalLogic_Project_a-real-car` 文件夹下的 `src` 目录。
3. 使用 Vivado 打开 `SUSTech_DigitalLogic_Project_a-real-car.xpr`，打开项目工程。

2 使用文档与说明

2.1 开发板与功能标注

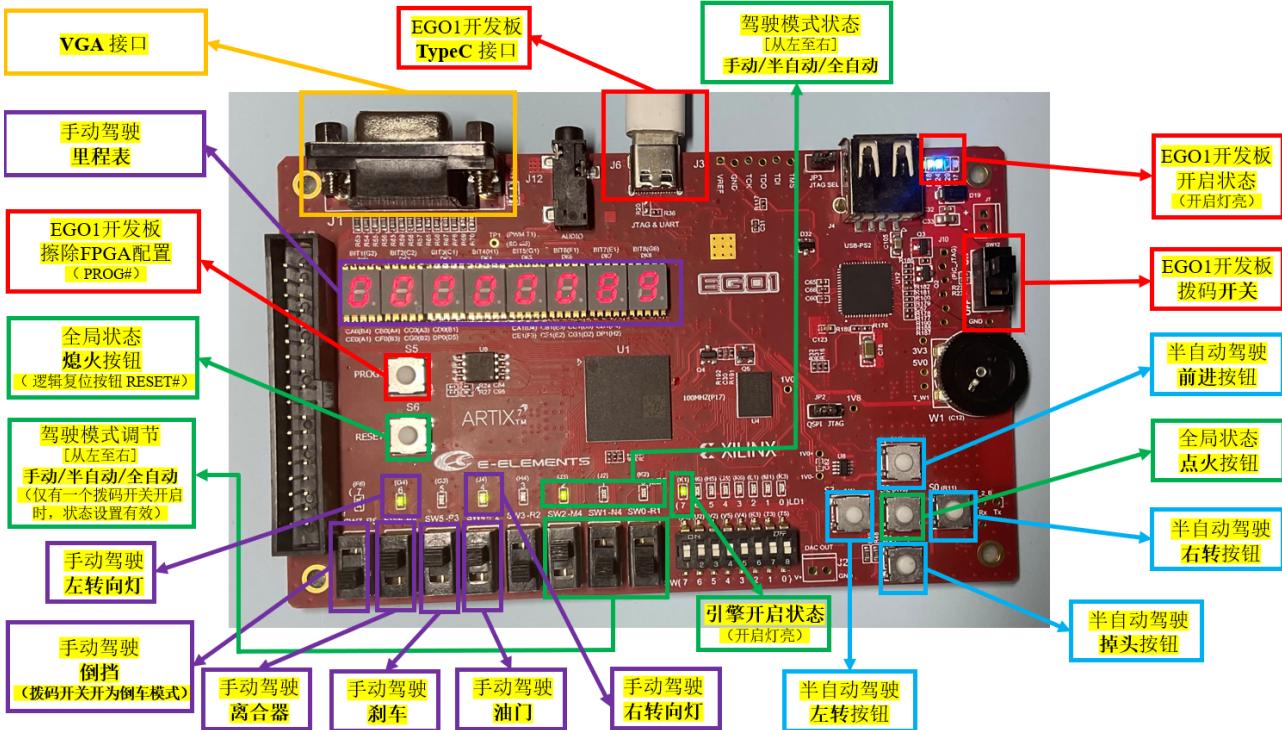
EGO1 是依元素科技基于 Xilinx Artix-7 FPGA 研发的便携式数模混合基础教学平台。

EGO1 配备的 FPGA (XC7A35T-1CSG324C) 具有大容量高性能等特点，能实现较复杂的数字逻辑设计；在 FPGA 内可以构建 MicroBlaze 处理器系统，可进行 SoC 设计。

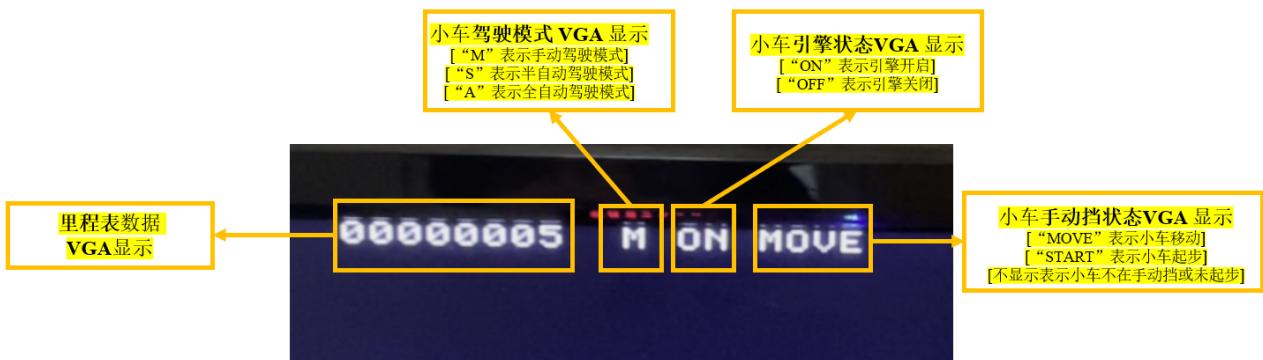
该平台拥有丰富的外设，以及灵活的通用扩展接口。

如需查阅 EGO1 开发板用户手册，请访问用户手册：https://e-elements.readthedocs.io/zh/ego1_v2.2。

本项目使用 EGO1 开发板，系统电路在开发板上实际使用的输入输出设备，在下图中给出了标注：



使用 VGA 外接显示器时显示的内容，在下图中给出了标注：



2.2 使用说明

上图中，我们使用红色线框图表示 EGO1 开发板启动和连接所需的输入输出设备及相关说明。

- TypeC 接口**: EGO1 提供的该 typeC 接口功能为 UART 和 JTAG，该接口可以为板卡供电。板卡上提供建议电压转换电路将 TypeC 输入的 5V 电压转换为板卡上各类芯片需要的工作电压。上电成功后 **开启状态** LED 灯点亮。
- 开启状态**: 该 LED 灯的亮灭表示开发板是否开启，对应引脚 D18。
- 拨码开关**: 该拨码开关控制 EGO1 开发板的启动和关闭，当拨至 **ON** 时，开发板将启动，当拨至 **OFF** 时，开发板将关闭。
- 擦除FPGA配置**: 即 **PROG#**，是专用按键，按下后用来擦除FPGA配置。

上图中，我们使用绿色线框图表示小车系统进行全局设置时，所需的相关输入输出设备及相关说明。

- 点火按钮**: 该按钮控制小车点火，若当前小车引擎熄火，则需要长按该按钮 **1s**，使得小车重新启动，对应引脚 R15。

- **熄火按钮**：该按钮控制小车熄火，若当前小车引擎打开，若按下该按钮，小车立即熄火，对应引脚 P15。
- **引擎开启状态**：该 LED 灯显示小车引擎是否开启，若当前小车熄火，LED灯不亮；若小车引擎开启，LED亮，对应引脚 K1。
- **驾驶模式调节**：这一组 3 个拨码开关，用来切换小车的模式。拨码开关从左至右，依次表示是否开启手动模式、是否开启半自动模式、是否开启全自动模式。当且仅当其中一个拨码开关打开时，将进入对应的驾驶模式。手动模式、半自动模式、全自动模式从左至右依次对应引脚 M4、N4、R1。
- **驾驶模式状态**：这一组 3 个 LED 灯，用来显示小车当前所在的驾驶模式。LED 灯从左至右，依次对应手动模式是否开启、半自动模式是否开启、全自动模式是否开启。当对应 LED 亮时，表明进入其中的对应驾驶模式。特别地，当 3 个 LED 灯均不亮时，小车未进入任何一种驾驶模式，原地静止。手动模式是否开启、半自动模式是否开启、全自动模式是否开启从左至右依次对应引脚 J3、J2、K2。

上图中，我们使用紫色线框图表示小车系统进行**自动驾驶模式**时，所需的相关输入输出设备及相关说明。

- **左转向灯**：该 LED 灯显示小车进入自动驾驶模式后的左转状态。若小车所处其他模式，该 LED 不亮；当小车进入自动驾驶模式，并且处在**未启动**状态时，该 LED 灯长亮，处在**启动中**状态时，该 LED 灯不亮，处在**移动中**并且执行左转向时，该 LED 灯闪亮。对应引脚 G4。
- **右转向灯**：该 LED 灯显示小车进入自动驾驶模式后的右转状态。若小车所处其他模式，该 LED 不亮；当小车进入自动驾驶模式，并且处在**未启动**状态时，该 LED 灯长亮，处在**启动中**状态时，该 LED 灯不亮，处在**移动中**并且执行右转向时，该 LED 灯闪亮。对应引脚 J4。
- **倒挡**：该拨码开关用来设置小车进入自动驾驶模式后，是否进入倒车状态。若该拨码开关关闭，小车受到**离合器**、**刹车**、**油门**的共同作用向前移动；若该拨码开关打开，小车受到**离合器**、**刹车**、**油门**的共同作用向后移动。对应引脚 P5。
- **离合器**：该拨码开关用来设置小车进入自动驾驶模式后的离合器状态。若该拨码开关关闭，小车离合器不踩下；若该拨码开关打开，小车离合器踩下。小车受到**离合器**、**刹车**、**油门**的共同作用移动。对应引脚 P4。
- **刹车**：该拨码开关用来设置小车进入自动驾驶模式后的刹车状态。若该拨码开关关闭，小车刹车不踩下；若该拨码开关打开，小车刹车踩下。小车受到**离合器**、**刹车**、**油门**的共同作用移动。对应引脚 P3。
- **油门**：该拨码开关用来设置小车进入自动驾驶模式后的油门状态。若该拨码开关关闭，小车油门不踩下；若该拨码开关打开，小车油门踩下。小车受到**离合器**、**刹车**、**油门**的共同作用移动。对应引脚 P2。
- **里程表**：里程表由 8 个共阴极数码管构成，用来显示当前小车自动驾驶模式时的运行里程。对应引脚较为复杂，需要使用到的引脚有：B4、A4、A3、B1、A1、B3、B2、D5、D4、E3、D3、F4、F3、E2、D2、H2（数码管的段选信号），G2、C2、C1、H1、G1、H1、G1、F1、E1、G6（数码管的片选信号）。

上图中，我们使用蓝色线框图表示小车系统进行**半自动驾驶模式**时，所需的相关输入输出设备及相关说明。

- **前进按钮**：该按钮用来接收小车进入岔路口后是否通过前进方式通过岔路口。若该按钮按下，表示小车将以前进通过岔路口。对应引脚 U4。
- **左转按钮**：该按钮用来接收小车进入岔路口后是否通过左转方式通过岔路口。若该按钮按下，表示小车将左转通过岔路口。对应引脚 V1。
- **右转按钮**：该按钮用来接收小车进入岔路口后是否通过右转方式通过岔路口。若该按钮按下，表示小车将右转通过岔路口。对应引脚 R11。
- **掉头按钮**：该按钮用来接收小车进入岔路口后是否通过掉头方式通过岔路口。若该按钮按下，表示小车将掉头通过岔路口。对应引脚 R17。

上图中，我们使用橙色线框图表示小车系统能**外接显示器**进行输出时，所需的相关输入输出设备及相关说明。

- **VGA接口**：该接口可以使用连接线与外接显示器相连，用来显示当前小车运行的状态（包括运行里程、运行模式），对应引脚 J1。

2.3 管脚约束

在项目中，我们使用了 EGO1 开发板的如下引脚，并将对应功能以表格形式简要列出。

您可以在项目的约束文件 `cons.xdc` 中找到如下的约束信息（名称和引脚编号）：

名称	原理图标号	FPGA IO PIN	功能简介
rx	UART_TX	N5	UART 模块与程序交互
tx	UART_RX	T4	UART 模块与程序交互
sys_clk	SYS_CLK	P7	时钟信号
stop_engine_signal	FPGA_RESET	P15	熄火键 / 复位键
enable_auto_signal	SW_0	R1	自动模式选择
enable_semauto_signal	SW_1	N4	半自动模式选择
enable_manual_signal	SW_2	M4	手动模式选择
throttle_signal	SW_4	P2	油门
brake_signal	SW_5	P3	刹车
clutch_signal	SW_6	P4	离合
reverse_signal	SW_7	P5	倒车挡
turn_right_signal	PB0	R11	右转
reverse_signal_button	PB1	R17	掉头
start_engine_signal	PB2	R15	点火键
turn_left_signal	PB3	V1	左转
go_straight_signal	PB4	U4	直行
auto_state	LED2_0	K2	全自动模式开启状态
semauto_state	LED2_1	J2	半自动模式开启状态
manual_state	LED2_2	J3	手动模式开启状态
turn_right	LED2_4	J4	右转向灯
turn_left	LED2_6	G4	左转向灯
engine_power	LED1_7	K1	引擎开启状态
seg_en[7: 0]	DN0_K1, ..., DN0_K4 DN1_K1, ..., DN1_K4	G2, C2, C1, H1 G1, F1, E1, G6	里程表片选信号
seg0[7: 0]	LED1_CA, ..., LED1_CG, LED1_DP	D4, E3, D3, F4 F3, E3, D3, H2	里程表低位段选信号
seg1[7: 0]	LED0_CA, ..., LED0_CG, LED0_DP	B4, A4, A3, B1 A1, B3, B2, D5	里程表高位段选信号
red[0: 3]	VGA_R0, ..., VGA_R3	F5, C6, C5, B7	VGA 红色信号
green[0: 3]	VGA_G0, ..., VGA_G3	B6, A6, A5, D8	VGA 绿色信号
blue[0: 3]	VGA_B0, ..., VGA_B3	C7, E6, E5, E7	VGA 蓝色信号
hs	VGA_HSYNC	D7	VGA 行同步信号
vs	V-SYNC	C4	VGA 场同步信号

2.4 小车机制

本项目中的小车运行机制，严格参照课程项目A Real Car 说明文档 [DL_2022F_project_introduction.pdf](#) 中的相关内容进行实现。

本部分内容，将简要介绍本项目中小车的相关机制，具体相关细节可参考：[项目说明文档](#)。

2.4.1 全局状态

在全局状态中，小车有 `power-on` 和 `power-off` 两种引擎状态，分别代表引擎启动和引擎关闭。

- 按下点火键 **1s** 后，小车将从 `power-off` 状态进入 `power-on` 状态，引擎启动。
- 按下熄火键后，小车将从 `power-on` 状态进入 `power-off` 状态，引擎关闭。
- 引擎启动时，`引擎开启状态` LED 灯长亮；当引擎关闭时，`引擎开启状态` LED 灯不亮。

在全局状态中，小车有 `Manual driving`、`Semi-auto driving`、`Auto driving` 三种驾驶模式，分别代表手动驾驶模式、半自动驾驶模式、全自动驾驶模式。

- 拨码开关（对应引脚从左到右依次为M4, N4, R1）状态为 `100` 时，小车切换进入手动驾驶模式。
- 拨码开关（对应引脚从左到右依次为M4, N4, R1）状态为 `010` 时，小车切换进入半自动驾驶模式。
- 拨码开关（对应引脚从左到右依次为M4, N4, R1）状态为 `001` 时，小车切换进入全自动驾驶模式。
- 小车引擎关闭，或者拨码开关的状态不为 **3 bit 独热码** 时，小车不进入任何驾驶模式。

2.4.2 手动驾驶模式

在手动驾驶模式中，小车有三种状态 `not-starting`、`starting`、`moving` 分别表示未起步状态、起步状态、移动状态。

- 当小车进入手动驾驶模式时，初始状态为未起步状态。
- 小车向 UART 模块中输出向前或者向后移动信号，当且仅当小车处于移动状态。

小车的移动由 `倒车挡`、`离合器`、`刹车`、`油门`、`左转按钮`、`右转按钮` 共同决定，具体机制如下：

1. 当小车处于未起步状态时，
 - 若开启油门、关闭离合，小车引擎关闭。
 - 若开启油门、关闭刹车、开启离合，小车进入起步状态。
 - 左转向灯和右转向灯长亮，小车静止。
2. 当小车处于起步状态时，
 - 若开启刹车，小车进入未起步状态。
 - 若开启油门、关闭刹车、关闭离合，小车进入移动状态。
 - 若仅左转按钮按下，小车将原地左转、左转向灯闪亮
 - 若仅右转按钮按下，小车将原地右转、右转向灯闪亮
 - 若左转按钮、右转按钮均按下或均不按下，小车静止、左转向灯和右转向灯均不亮。
3. 当小车进入移动状态时，
 - 若打开离合或者关闭油门，小车进入起步状态。
 - 若打开刹车，小车进入未起步状态。
 - 若关闭离合并且改变倒车档拨码开关状态，小车引擎关闭。
 - 若仅左转按钮按下，小车将向左转向行驶、左转向灯闪亮
 - 若仅右转按钮按下，小车将向右转向行驶、右转向灯闪亮

- 若左转按钮、右转按钮均按下或均不按下，小车直线行驶、左转向灯和右转向灯均不亮。
- 4. 小车具有刹车优先系统（**BOS**），当刹车和油门同时打开时，油门将失效。
- 5. 小车处于移动状态，倒车挡开关关闭，小车向前移动；倒车挡开关开启，小车向后移动。
- 6. 小车引擎关闭或者不处于自动驾驶模式，里程表不显示数据，并且重设为 **0**。
- 7. 小车引擎开启并且处于自动驾驶模式，并且处于移动状态，里程表匀速增加。

2.4.3 半自动驾驶模式

在半自动驾驶模式中，小车有三种状态 `moving`、`turning`、`wait`，分别表示移动状态、转向状态、等待指令状态。

- 当小车进入半自动驾驶模式时，初始状态为移动状态。

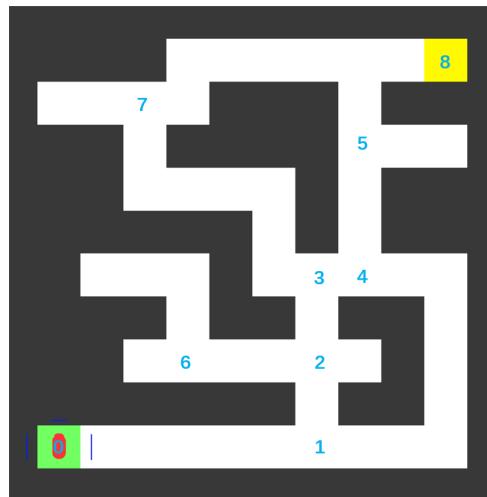
小车的状态主要由每个路口处处于的等待状态时，用户通过 `前进按钮`、`掉头按钮`、`右转按钮`、`左转按钮` 输入的命令决定，具体机制如下：

1. 当小车处于移动状态，
 - 若小车检测到当前处于路口处，小车进入等待指令状态。
 - 若小车检测到当前不处于路口处，按照当前行驶状态行驶。
 - 若小车检测到以当前行驶状态无法继续行驶时，小车将会自动掉头，按照当前行驶状态继续行驶。
2. 当小车处于等待指令状态，
 - 若用户按下 `前进按钮`，小车进入移动状态。
 - 若用户按下 `掉头按钮`，小车进入（向后）转向状态。
 - 若用户按下 `左转按钮`，小车进入（向左）转向状态。
 - 若用户按下 `右转按钮`，小车进入（向右）转向状态。
 - 若用户不输入指令，小车继续处于等待指令状态。
3. 当小车处于转向状态，
 - 小车将按既定方向进行转向，并且通过当前路口后，进入移动状态。

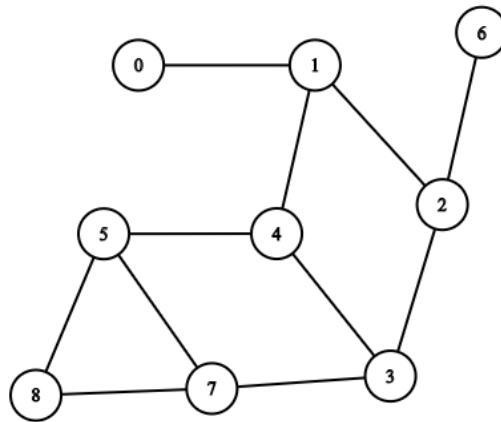
2.4.4 全自动驾驶模式

小车在全自动驾驶模式中，小车会自动驾驶，自行从当前位置开始寻找并前往目的地。

如果我们将迷宫的每个路口、起点与终点都看作图中的一个节点，路口与路口之间的唯一通道看作是图中节点间的边。如果我们能实现图的深度优先搜索 **DFS**，那么就能找到一条从起点到终点的路径。



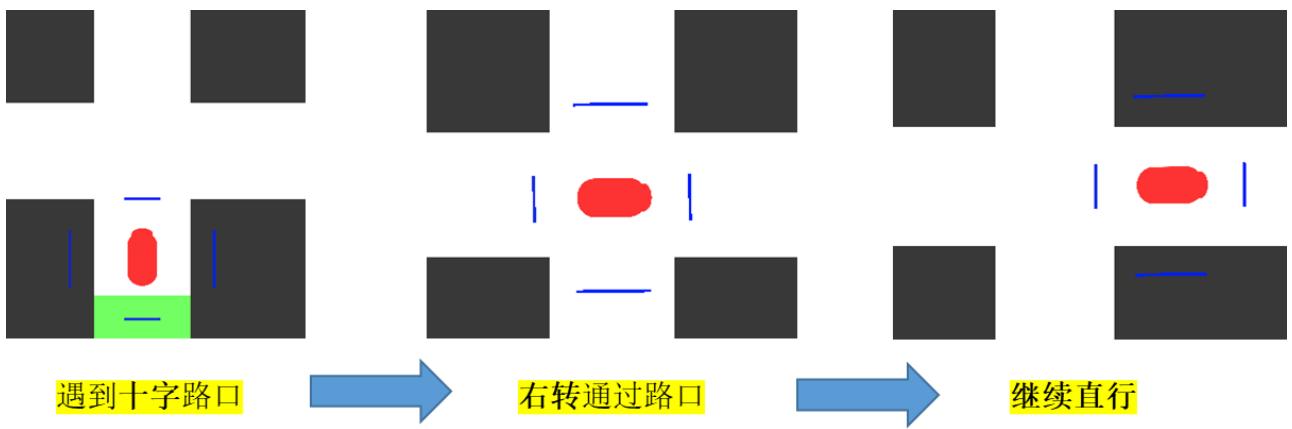
例如上面的迷宫，能抽象成如下的图，其中起点是 0，终点是 8。



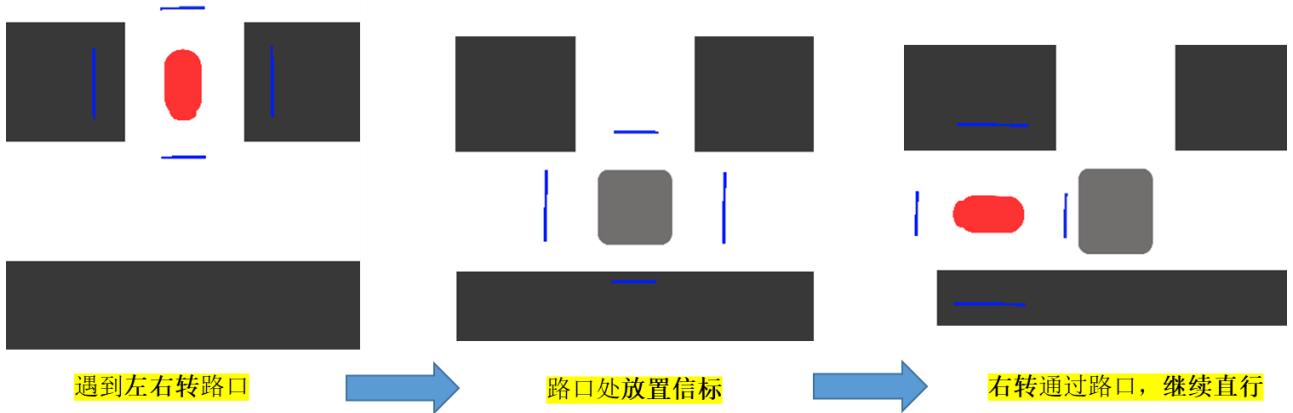
我们采用**右转优先**，即每次遇到路口处，若能右转则直接右转，其次直行，最后再左转。

除此之外，还需要使用信标（可以通行的墙）来记录当前路口是否被访问过，以判断图中的环。

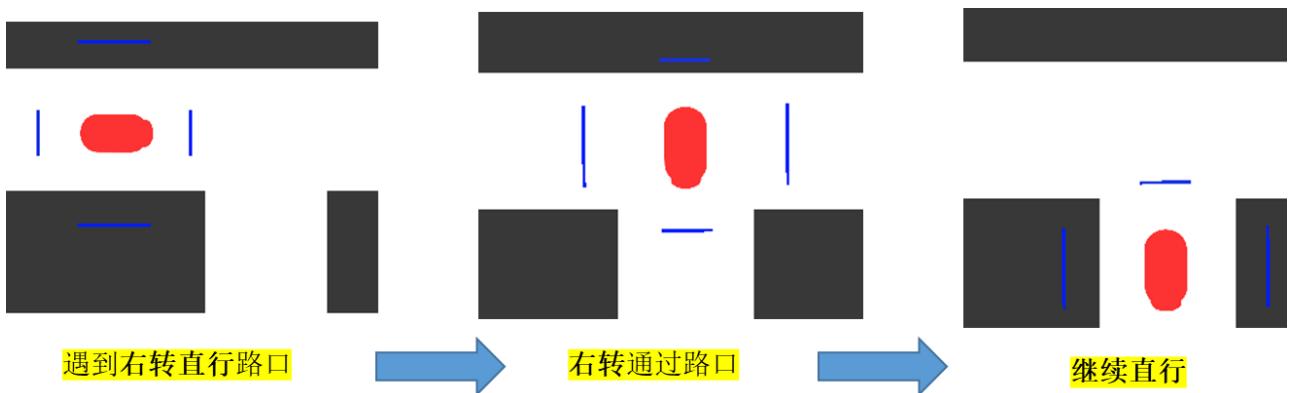
- 当遇到十字路口时，右转通过路口，不放置信标。



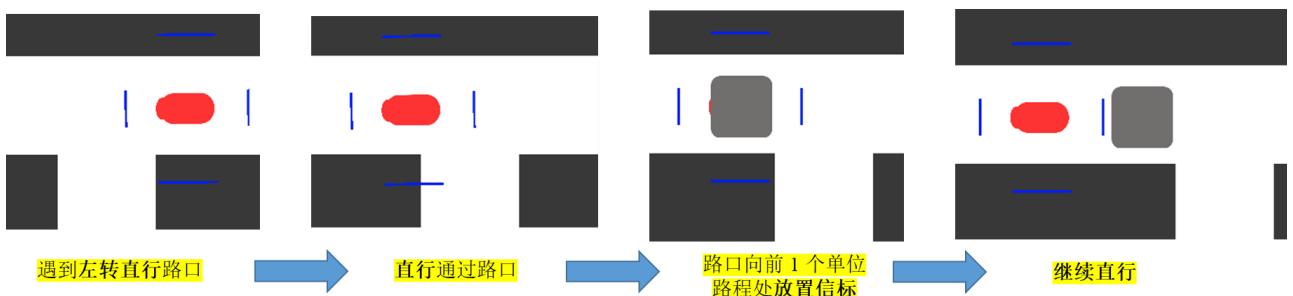
- 当遇到左右转路口时，在路口处放置信标，并右转通过路口。



- 当遇到右转直行路口时，右转通过路口，不放置信标。

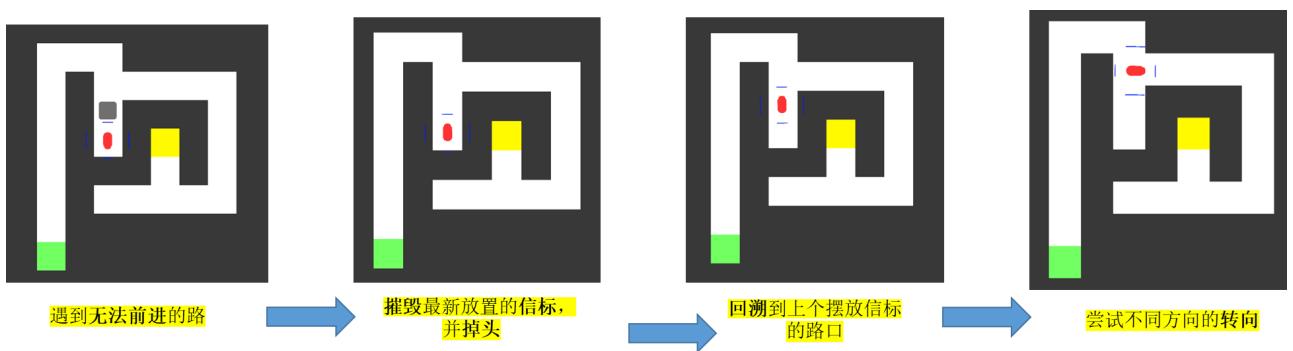


- 当遇到左转直行路口时，前进通过路口，并在路口向前 1 个单位路程处放置信标。



除此之外，当遇到无法前进的路时，需要摧毁最近的信标，进行回溯。

- 当无法前进时，摧毁最近的信标，并掉头（随后小车将回溯到上个路口，进行不同转向的尝试）。



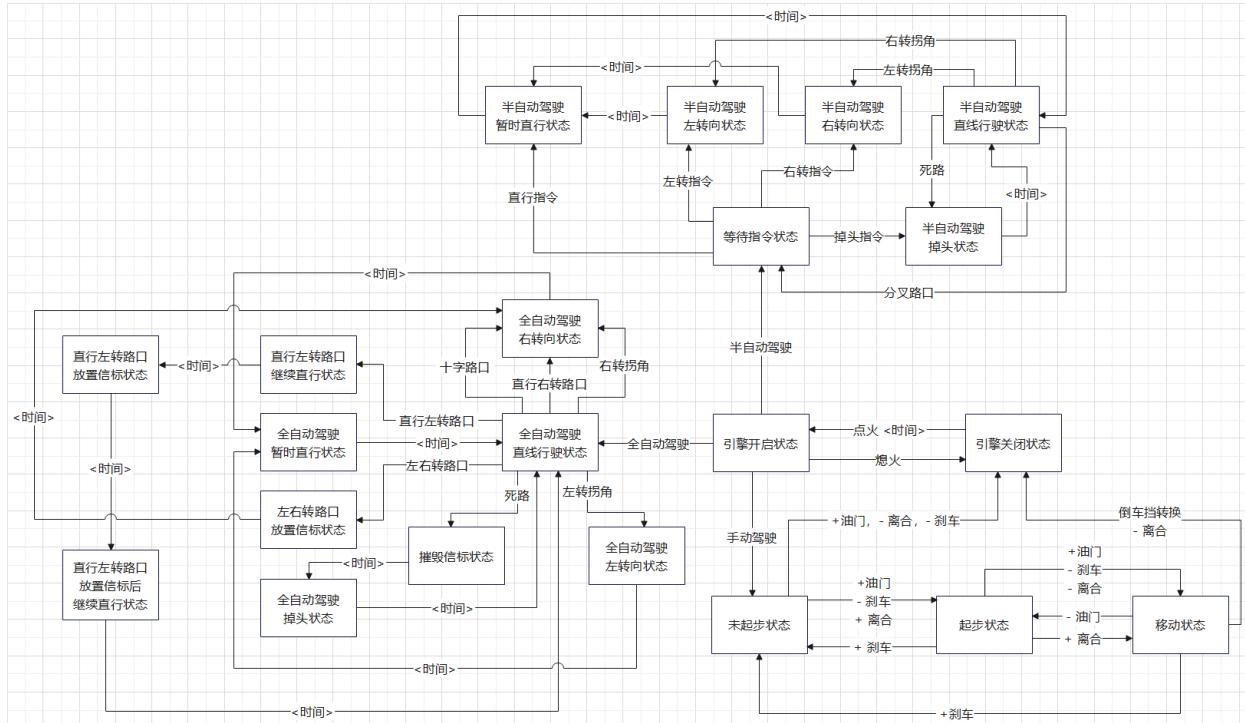
3 系统结构设计

本项目可以分为五个模块：全局设置模块、手动驾驶模块、半自动驾驶模块、全自动驾驶模块、VGA模块。

其中 VGA 模块相对独立，进行实时显示。而其他四个模块中存在不同状态，依据电路的外部输入进行状态转换，并将电路相关信号进行输出，以控制模拟测试程序 **DriveCar.exe** 中小车的运动状态和 EGO1 开发板上相关输入输出设备。

3.1 状态流程图与模块工作机制

这是本项目的状态流程图，使用 **Edrawmax** 综合绘图软件 绘图：



3.2 电路结构框图

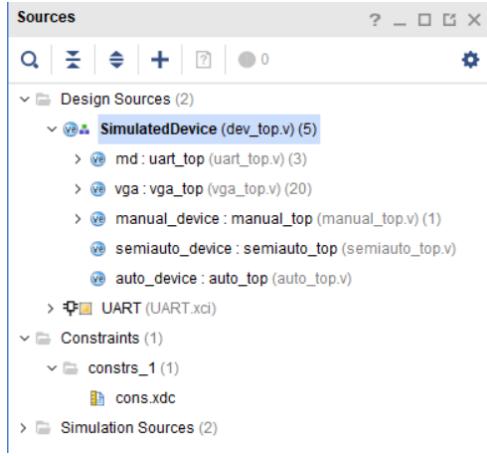
3.2.1 项目结构

在我们的 **vivado** 工程中，顶层模块是 **dev_top** 模块，该模块主要实现小车各模块之间的切换，以及引擎开启关闭的探测并控制小车引擎。

子模块包括：

- **uart_top** 模块，该模块主要用于实现 **UART** 连接，并与模拟测试程序 **DriveCar.exe** 交互。
- **vga_top** 模块，该模块主要用于实现外接显示器显示，显示小车当前运动状态。
- **manual_top** 模块，该模块主要用于实现小车的手动驾驶模式。
- **semiauto_top** 模块，该模块主要用于实现小车的半自动驾驶模式。
- **auto_top** 模块，该模块主要用于实现小车的全自动驾驶模式。

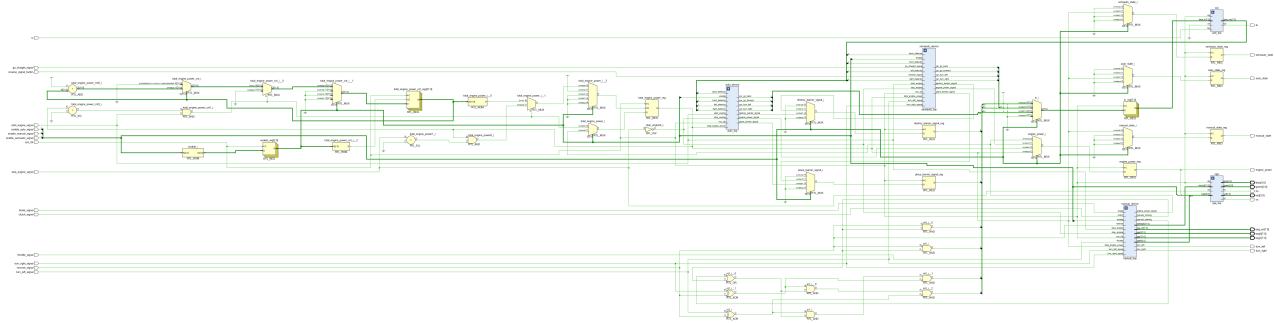
vivado 工程的项目结构图如下：



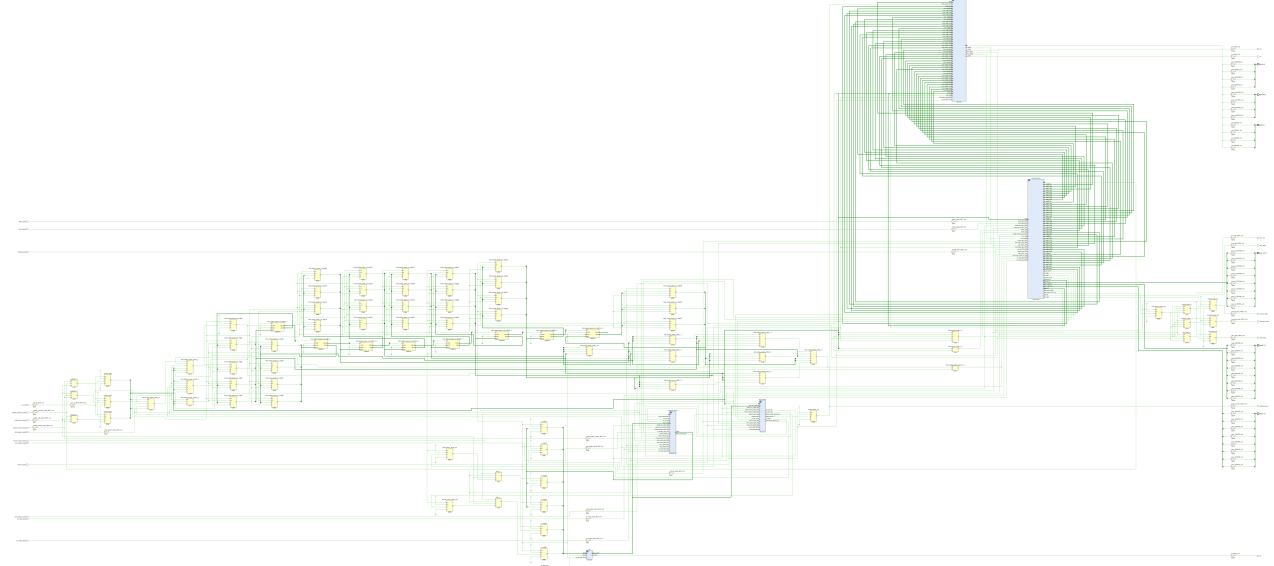
3.2.2 电路结构框图

我们使用 **vivado** 的 **RTL** 分析与综合后生成的电路图，展示顶层模块与各子模块、各子模块之间的关系。

- 这是使用 **RTL** 分析后生成的电路图。（图片清晰度足够高，可以按住 **Ctrl + 鼠标滚轮** 放大查看）



- 这是使用 **综合** 后生成的电路图。（图片清晰度足够高，可以按住 **Ctrl + 鼠标滚轮** 放大查看）



本项目**A Real Car**的顶层模块 `dev_top` 使用了 `uart_top`、`vga_top`、`manual_top`、`semiauto_top`、`auto_top` 五个子模块。

其中：

- `uart_top` 模块实现 **UART** 连接，并与模拟测试程序 **DriveCar.exe** 交互功能，使用顶层模块中的时钟信号、复位信号、数据输入、数据输出信号，并将处理后的结果段码信号、数据发送信号传递回顶层模块，以供其他子模块使用。

- `vga_top` 模块实现 **VGA** 显示小车运动状态，使用顶层模块中的时钟信号、复位信号、引擎状态、小车的模式与手动模式的状态拼接成的状态值、手动模式记录的公里数转换为十进制后的值，并将处理后的红色信号、绿色信号、蓝色信号、行同步信号、场同步信号传递回顶层模块，用于实现 **VGA** 显示。
- `manual_top` 模块实现小车的手动驾驶模式，使用顶层模块中的复位信号、时钟信号、关闭引擎信号、开启引擎信号、倒车档信号、离合信号、刹车信号、油门信号、左转向信号、右转向信号、引擎状态信号，并将处理后的新引擎状态信号、未起步信号、起步信号、移动信号、左转灯信号、右转灯信号、里程表信号、小车的模式与手动模式的状态拼接成的状态值传递回顶层模块，用于实现转向灯、里程表、引擎指示灯等功能，并将相关信号供其他子模块使用。
- `semiauto_top` 模块实现小车的半自动驾驶模式，使用顶层模块中的复位信号、时钟信号、关闭引擎信号、开启引擎信号、前探测器信号、后探测器信号、左探测性信号、右探测器信号、前进指令信号、右转指令信号、左转指令信号、掉头指令信号、引擎状态信号，并将处理后的左转向信号、右转向信号、移动信号、倒车信号、新引擎状态信号、摧毁信标信号、放置信标信号传递回顶层模块，用于实现半自动驾驶功能，并将相关信号供其他子模块使用。
- `auto_top` 模块实现小车的全自动驾驶模式，使用顶层模块中的复位信号、时钟信号、关闭引擎信号、开启引擎信号、前探测器信号、后探测器信号、左探测性信号、右探测器信号、引擎状态信号，并将处理后的左转向信号、右转向信号、移动信号、倒车信号、新引擎状态信号、摧毁信标信号、放置信标信号传递回顶层模块，用于实现全自动驾驶功能，并将相关信号供其他子模块使用。

3.2.3 各子模块功能、输入输出端口规格说明及结构图

3.2.3.1 各子模块功能

模块名称	功能
uart_top	实现 UART 连接，并与模拟测试程序 DriveCar.exe 交互
vga_top	实现外接显示器显示，显示小车当前运动状态
manual_top	实现小车的手动驾驶模式
semiauto_top	实现小车的半自动驾驶模式
auto_top	实现小车的全自动驾驶模式

3.2.3.2 输入输出端口规格说明

- `uart_top` 模块（顶层模块）

```

1 module SimulatedDevice (
2     input sys_clk, //系统时钟
3     input rx, // UART 模块与程序交互
4     output tx, // UART 模块与程序交互
5     input turn_left_signal, // 左转信号
6     input turn_right_signal, // 右转信号
7     input stop_engine_signal, // 熄火信号、复位信号
8     input start_engine_signal, // 点火信号
9     input reverse_signal, // 倒车挡信号
10    input clutch_signal, // 离合器信号
11    input brake_signal, // 刹车信号
12    input throttle_signal, // 油门信号
13    output reg engine_power, // 引擎状态指示信号
14    output turn_left, // 左转向灯信号
15    output turn_right, // 右转向灯信号
16    output [7: 0] seg1, // 里程表高位段选信号

```

```

17     output [7: 0] seg0, // 里程表低位段选信号
18     output [7: 0] seg_en, // 里程表片选信号
19     input go_straight_signal, // 直行信号
20     input reverse_signal_button, // 掉头信号
21     input enable_manual_signal, // 开启手动驾驶模式信号
22     input enable_semauto_signal, // 开启半自动驾驶信号
23     input enable_auto_signal, // 开启全自动驾驶信号
24     output reg manual_state, // 手动驾驶模式指示信号
25     output reg semiauto_state, // 半驾驶模式指示信号
26     output reg auto_state, // 全自动驾驶模式指示信号
27     output[3:0] red, // VGA 红色信号
28     output[3:0] green, // VGA 绿色信号
29     output[3:0] blue, // VGA 蓝色信号
30     output hs, // VGA 行同步信号
31     output vs // VGA 场同步信号
32 );
33 // ...
34 endmodule

```

- `vga_top` 模块（实现VGA显示功能的子模块）

```

1 module vga_top (
2     input clk, // 系统时钟
3     input rst, // 复位信号
4     input [31:0] record, // 手动模式记录的公里数转换为十进制后的值
5     input[6:0] state, // 小车的模式与手动模式的状态拼接成的状态值
6     output reg[3:0] red, // VGA 红色信号
7     output reg[3:0] green, // VGA 绿色信号
8     output reg[3:0] blue, // VGA 蓝色信号
9     output hs, // VGA 行同步信号
10    output vs // VGA 场同步信号
11 );
12 // ...
13 endmodule

```

- `manual_top` 模块（实现手动驾驶模式功能的子模块）

```

1 module manual_top(
2     input enable, // 复位信号
3     input sys_clk, // 系统时钟
4     input stop_engine, // 关闭引擎信号
5     input start_engine, // 开启引擎信号
6     input reverse, // 倒车挡信号
7     input clutch, // 离合器信号
8     input brake, // 刹车信号
9     input throttle, // 油门信号
10    input turn_left_signal, // 左转信号

```

```

11    input turn_right_signal, // 右转信号
12    output reg engine_power_signal, // 新引擎状态信号
13    output reg manual_not_starting, // 手动挡未起步信号
14    output reg manual_starting, // 手动挡起步信号
15    output reg manual_moving, // 手动挡移动信号
16    output reg turn_left, // 左转信号
17    output reg turn_right, // 右转信号
18    output [7: 0] seg1, // 里程表高位段选信号
19    output [7: 0] seg0, // 里程表低位段选信号
20    output [7: 0] seg_en, // 里程表片选信号
21    input total_engine_power, // 当前引擎状态信号
22    output [31:0] mileage, // 里程表数值信号
23    output reg[2:0] state // 手动模式的状态拼接成的状态值
24 );
25 // ...
26 endmodule

```

- `semiauto_top` 模块（实现半自动驾驶模式功能的子模块）

```

1 module semiauto_top(
2     input enable, // 复位信号
3     input sys_clk, // 系统时钟
4     input stop_engine, // 关闭引擎信号
5     input start_engine, // 开启引擎信号
6     output reg car_turn_right, // 小车右转信号
7     output reg car_turn_left, // 小车左转信号
8     output reg car_go_forward, // 小车前进信号
9     output reg car_go_back, // 小车后退信号
10    output reg engine_power_signal, // 新引擎状态信号
11    output reg destroy_barrier_signal, // 摧毁信标信号
12    output reg place_barrier_signal, // 放置信标信号
13    input front_detector, // 前探测器障碍物信号
14    input back_detector, // 后探测器障碍物信号
15    input left_detector, // 左探测器障碍物信号
16    input right_detector, // 右探测器障碍物信号
17    input go_straight_signal, // 直行命令信号
18    input turn_right_signal, // 右转命令信号
19    input turn_left_signal, // 左转命令信号
20    input reverse_signal, // 掉头命令信号
21    input total_engine_power // 当前引擎状态信号
22 );
23 // ...
24 endmodule

```

- `auto_top` 模块（实现自动驾驶模式的子模块）

```

1 module auto_top(

```

```

2   input enable, // 复位信号
3   input sys_clk, // 系统时钟
4   input stop_engine, // 关闭引擎信号
5   input start_engine, // 开启引擎信号
6   output reg car_turn_right, // 小车右转信号
7   output reg car_turn_left, // 小车左转信号
8   output reg car_go_forward, // 小车前进信号
9   output reg car_go_back, // 小车后退信号
10  output reg engine_power_signal, // 新引擎状态信号
11  output reg destroy_barrier_signal, // 摧毁信标信号
12  output reg place_barrier_signal, // 放置信标信号
13  input front_detector, // 前探测器障碍物信号
14  input back_detector, // 后探测器障碍物信号
15  input left_detector, // 左探测器障碍物信号
16  input right_detector, // 右探测器障碍物信号
17  input total_engine_power // 当前引擎状态信号
18 );
19 // ...
20 endmodule

```

3.2.3.3 结构图

请见 [3.2.2 电路结构框图](#)，我们使用 **vivado** 的 **RTL** 分析和综合，分别生成相关的电路图。

如需下载和查看 **PDF** 版本的电路结构图，请访问：

- **RTL** 分析后生成的电路图：[RTL ANALYSIS Schematic.pdf](#)。
- 综合 分析后生成的电路图：[SYNTHESIS Schematic.pdf](#)。

4 开发过程中间的经验总结及优化

- 张天悦（学号：12112908）

- 罗嘉诚（学号：12112910）

首先从项目开发进行的全局来看，首先就将整个项目考虑的事无巨细几乎是不可能的。所以我们小组从项目开始，就稳扎稳打，每完成一个功能就进行仿真和上板测试。项目的完成就好像搭积木一样，从简单的实现到逐渐完善趋于完美，一步步走来很有成就感。

其次，我认为 **verilog** 编程一个比较重要的技巧是明确当前时钟周期要干什么，下个时钟周期即要干什么，将较为整体的事情原子化、碎片化、状态化，便于硬件进行处理。如在项目中实现的小车转弯经过路口，我们将这个较为连续的事件转化为四个过程——直走探测路口、停止转向、直走通过路口、继续直行。将事件进行原子化、碎片化、状态化有助于提高电路的稳定度和编程的难度。

然后，硬件描述语言的编程和一般的高级语言，在编程方面感觉非常不一样。后者更加强调编程时以自顶向下的顺序原则进行思考，前者更加强调以互相平行的并行原则思考。想要熟练掌握相关的技巧和编程方式，还是有一定难度。

还有，在代码调试方面，硬件描述语言颇有难度。对于习惯一般高级语言的**输出调试法**或者**Debug调试法**的我而言，逐渐习惯使用 **verilog** 中的 **Test Bench**，使用模拟方法调试代码，不是一件容易的事情，但好在我也逐渐在克服。

然后，在实现 **bonus** 功能的自动驾驶模块时，我着实遇到了困难。我不怎么会玩迷宫，所谓的“右手原则”之前都没有听说过。通过查找互联网相关资料、询问队友和其他组的同学，我发现“右手原则”就是将右手始终碰着墙壁，保持直行。并且将这件事和 **DSAA** 课程中图的相关知识结合起来，我发现迷宫问题可以抽象为图中的寻找指定起点和终点的问题，可以使用 **DFS** 相关知识解决。并且根据放置信标，巧妙地记录了每个节点是否访问过的状态，从而解决了这个问题。所以，在遇到不熟悉的问题时，可以适当的寻求他人和其他资料的帮助，这有助于快速转化为熟悉擅长的问题，简化遇到的问题，这利于问题的解决。

最后，报告写作也并不是一件容易的事情，如何将我们在项目的成果用简洁精美的报告展示出来不容易。说实话，经过这次的报告撰写，确实提高了我的报告写作能力。

Thank You !