

第四届

全国大学生集成电路创新创业大赛

CICIEC

项目设计论文

参赛题目：RISC-V 挑战杯子赛题一

队伍编号： AGD103942

团队名称： light chaser

联 系 人： 林沐晗

电 话： 13600858238

邮 箱： muhanlin@link.cuhk.edu.cn

# 一、摘要

本项目通过使用 RISC-V SiFive Learn Inventor 芯片，首先通过对寄存器控制，使用红外传感器和超声波传感器实现了小车循线和避障行驶。在加入 FreeRTOS 实时多线程操作系统后，实现了芯片与小车的多线程通信，达到多任务并行的效果。 本文对该项目实施流程进行了详细介绍，并通过实验展示效果。

本项目Github地址：<https://github.com/RobertGCNiu/RISC-V_Competition>

视频地址：<https://github.com/RobertGCNiu/RISC-V_Competition>

软硬件说明

|  |  |
| --- | --- |
| 开发板 | SiFive Learn Inventor |
| 集成开发环境 | Freedom Studio v10.3.1 Linux |
| 小车 | 柴火BitCar MicroBit智能小车套件 |
| 电脑操作系统 | Ubuntu 20.04 LTS |

# 二、作品简介

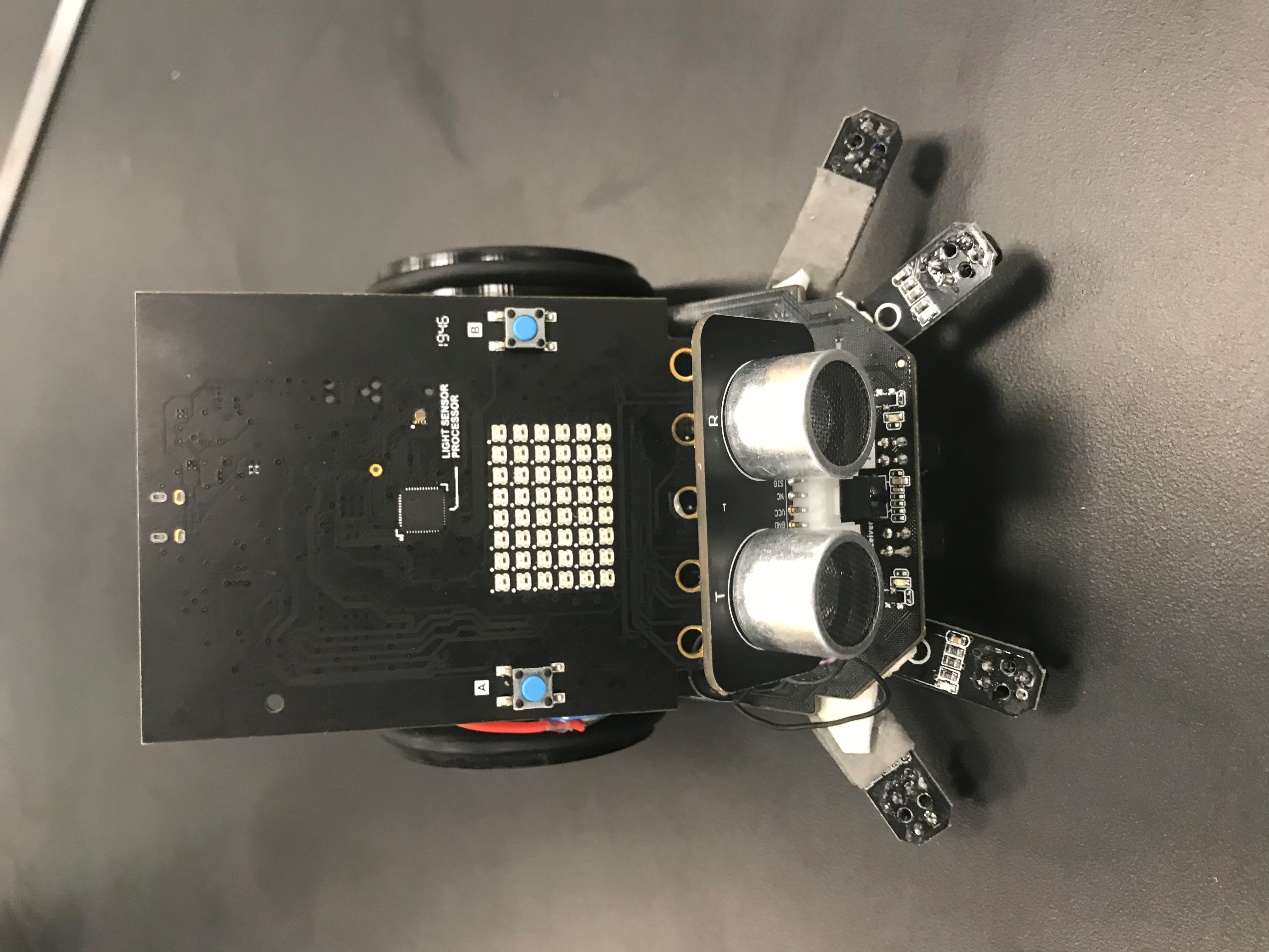


图1：智能小车

RISC-V 开源架构为物联网时代提供了新的芯片选择，近年来受到越来越多的关注。本项目使用Microbit小车搭载一枚RISC-V SiFive Learn Inventor芯片，实现了小车的循迹功能，并通过超声波模块，实现避障功能。为实现虚线避障功能，我们加装了多个红外传感器。小车控制主程序参见**附录5.1**。我们通过调用PWM函数改变占空比实现了速度控制。为了实现多线程，我们引入FreeRTOS操作系统，将多个程序的进程并列执行。在实验中，我们实现了虚线行驶、倒行、避障、循迹等功能。

# 三、技术路线

## 1. 硬件控制

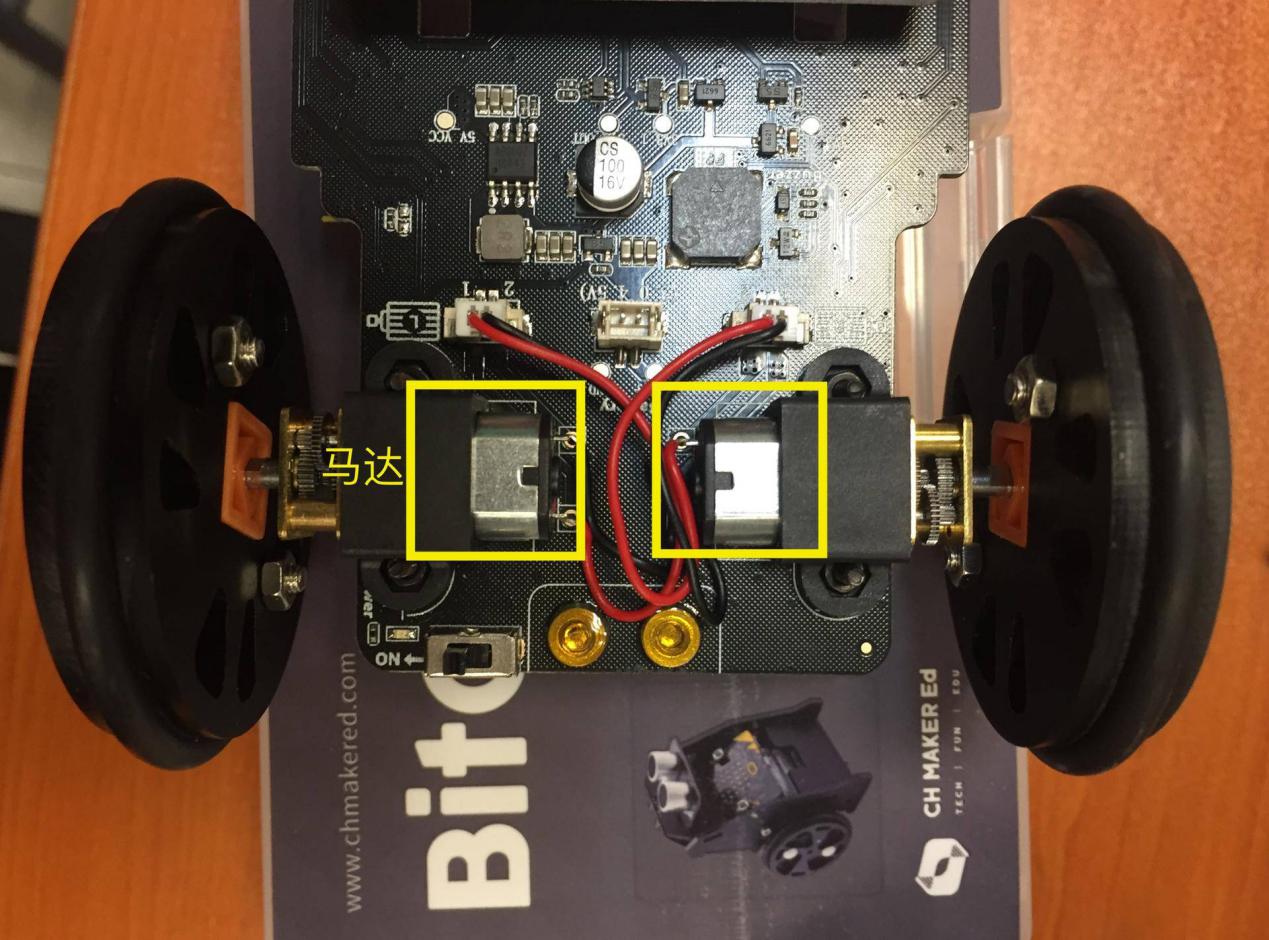


图2：小车电机

### 1.1 电机控制

小车左右轮的前转与后转分别由不同的**GPIO** 口控制，共对应4个**GPIO**引脚。按照官方文档进行地址计算，得到控制四个引脚的几个寄存器地址。通过将1写入寄存器相关位置使能四个 **GPIO**口。然后将1写入存储 **GPIO**口输出值的寄存器，便可使 GPIO口输出高电平，从而控制相应的轮子转动；若是写入0，则可使电机停止转动。

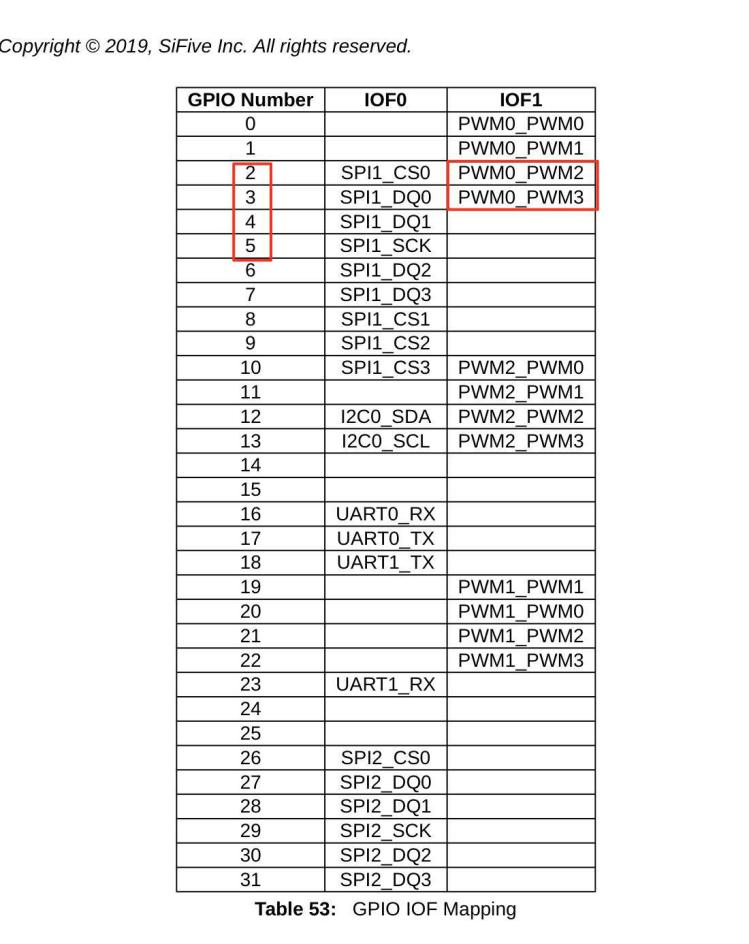


图3：PWM引脚

### 1.2 PWM调速

鉴于开发板本身 *PWM* 不完整，本队在程序中自主设计了 *PWM* 函数。函数输入量 *speed* 的大小决定了 *PWM* 的占空比的大小。占空比是指在一个脉冲循环内，通电时间相对于总时间所占的比例。占空比越大，则小车电机转动速度越快。本程序设定 *speed* 最大值为 255，最小值为 0。当输入 *speed* 为 0 时，小车马达全速运转，当输入 *speed* 为 255 时，小车马达不运转。下图一为芯片各个 GPIO 口连出的 PWM 接口，红色部分为小车电机对应的 *GPIO* 口，缺失 *PWM* 的 GPIO 口为 4 和 5。图二为芯片 GPIO 口对应的小车的 pin 口， 4、5 对应的是 *p13* 、*p14*，图三为开发板上 pin 口，可见 *p13* 、*p14* 对应小车左轮，小车左轮缺失 *PWM*。因此，本队自主设计了四个 *PWM* 函数来实现小车直行，左转，右转，倒车的控制，具体内容说明详见**附录5.2**。

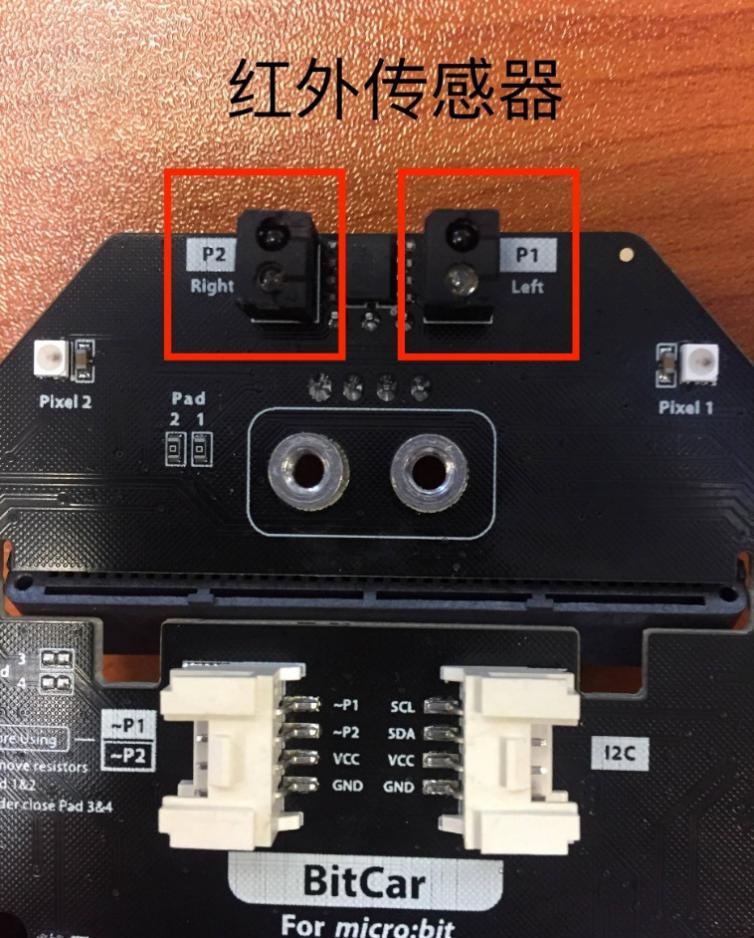


图4：红外传感器

### 1.3 循迹

**P1、P2** 两个 Pin 口一端分别连接左右两个红外传感器，另一端连接两个 GPIO 引脚。当传感器在黑色轨道上时，传感器将接收到弱红外线信号，则对应的 GPIO 引脚输出低电平，寄存器相关值显示为0。当传感器不在黑色轨道上时，传感器将接收到强红外线信号，则对应的 GPIO 引脚输出高电平，寄存器相关值显示为1。程序计算地址，获得存储*GPIO* 引脚输入值的寄存器位的对应地址，再读取该地址内的值，来判断小车运动状态，从而控制小车做出不同的反应，实现循迹功能。

### 1.4 避障

在进入障碍区之前超声波会一直工作，赛道之间的距离为60cm, 当超声波测出的距离大于30cm时程序会判断为无障碍物。本队小车将按照Z字形穿过障碍区。小车进入障碍区之前，当超声波测距小于30cm时，小车停下向左转，转至检测不到障碍物，即距离大于30cm时，小车直行进入障碍区，直至红外传感器检测到黑线，停止。在障碍区中，当小车左边红外传感器触碰到边界后，小车将向右转，转至超声波探测不到障碍物，再直行至右边边界，小车左转至超声波探测不到障碍物，重复操作直至小车驶出障碍区。

## 2. FreeRTOS多线程控制

### 2.1 FreeRTOS简介

FreeRTOS 是一个能满足嵌入式硬实时要求的实时内核（或调度器）。它实现了应用的多线程设计要求。对于仅有一个内核的处理器来说，任意时间只有一个线程能被执行。内核根据设计者分配给各个任务的优先级高低来决定哪个线程先被执行。任务优先级能保证一个应用满足其执行时的时限要求。

### 2.2 FreeRTOS应用

仅使用小车裸机程序时，小车无法同时执行多项任务。这会导致小车执行任务的实时性和效率较低，浪费了处理时间和内存资源，能耗较高。同时，对于多项任务的调度工作完全依赖于开发者编写的程序，不仅增加了开发者的工作负担，也使得任务程序更繁杂，更容易产生错误。

本项目应用实时操作系统很好地解决了这些问题。当小车执行延时程序、电机持续输出相同电压时，处理器实际上是处于空闲状态。应用 **RTOS** 可以将该时段利用起来：延时的任务进入阻塞态，处理器另从就绪列表中选择优先级最高的任务来执行。这样就实现了“多任务同时进行”的程序执行效果，节省处理时间。**FreeRTOS** 的任务调度器可以依据优先级协调各任务的执行，并且 **FreeRTOS** 提供计时服务，这使得应用程序更简洁，软件表现更稳定，大大降低开发者负担。同时，RTOS 为各项任务提供合理的内存管理服务，使得内存利用更高效、更具可控性。

### 2.3 FreeRTOS原理

#### 2.3.1多任务实现的原理说明

  事实上，在任意时间 CPU 总是仅执行一个任务的程序。RTOS 可能在某个任务循环中阻塞或挂起该任务，然后去执行另一任务，从而实现多任务同时进行的效果。任务调度由任务调度器函数来实现。

  任务有四种状态：运行态、就绪态、阻塞态和挂起态。每个任务都拥有指向自己的状态列表项和事件列表项。这些列表项从属于某个列表，例如就绪列表、等待某个事件的列表等等。它们记录了各个任务的状态或是阻塞状态下等待的事件。任务调度器总是会从就绪列表选择优先级最高的任务来执行。当某一事件发生，等待该事件的任务 A 将会从阻塞态进入就绪态，实质是其状态列表项从一个列表中删除后进入另一个列表。如果任务 A 的优先级高于当前执行的任务 B 的优先级，且抢占式内核对应的宏被设为 1,A 将会夺取 B 的执行权转入运行态，任务 B 的执行因此被中断。若是任务函数中调用了任务挂起函数，则目标任务将被挂起，CPU 使用权亦将被转让。只有当其他任务调用函数来恢复这个被挂起的任务时，该任务会重新进入就绪态等待被执行。

  对于同一优先级的任务调度，有两种情况。若是使用时间片管理（相关宏设为 1），则 CPU 在每个时间片执行一个任务，该优先级的任务轮流被执行。若是不使用时间片管理（相关宏设为 0），则任务转换仅仅发生在更高优先级任务抢占或当前任务进入阻塞或挂起态之时。

  在这个项目中，首先被执行的是开始任务 Start\_Task，优先级为 1。开始任务首先创建出队列和优先级为 5 的 Print\_task,打印所需字符串。Print\_task 执行完成，该任务被删除，CPU 使用权转交给 Start\_task。接着优先级为 3 的红外检测任务被创建，CPU 使用权被转交给红外检测任务。红外检测任务的首个循环完成后，vTaskDelay 函数被调用，该任务进入阻塞态。此时 CPU 创建 Car\_task,优先级为 4,高于红外探测任务，优先被 CPU 运行。Car\_task 也在第一个循环末尾进入延时阻塞状态。在 Car\_task 的延时时段中，优先级低于它的 Infrared\_task 得以执行一个循环并进入延时阻塞状态。延时时间到，Car\_task 解除阻塞，获得 CPU 使用权。接下来小车控制和红外检测任务轮流使用 CPU，完成巡线目标。

#### 2.3.2 队列与通信的原理说明

在 FreeRTOS 中，队列被用于任务与任务之间或是中断与任务之间的信息传递。队列实质上是链表，每一个链表项大小相同、至多记录一条信息。消息一般遵循先进先出的规则进出列表。对于一个队列来说，任何一个任务都可以向其传递信息（其实是记录信息的地址），实质是将传递内容保存在链表项中。同样，任何一个任务都可以从该队列中读取信息。被读取过的信息根据程序设置可以被删除也可以被保留以供其他任务读取。

  在 Start\_task 中，消息队列 Message\_task 被创建。只要队列未满，CPU 每执行一次 Infrared\_task，就会有一则红外探测消息被发送至队列中。若队列已满，Infrared\_task 就放弃发送这则消息，将 CPU 使用权交给 Car\_task。只要队列未空，CPU 每执行一次 Car\_task，队列开头的消息就会被该任务取出（读取后不再保留在队列中）。若是队列为空，则 Car\_task 进入阻塞态，直至有消息进入队列中。代码内容参考**附录5.3.**

## 2.4 FreeRTOS 移植说明

(1) 下载 ***freedom-e-sdk*** 程序包，在 **ubuntu** 终端设置编译所需路径:

1. export RISCV\_OPENOCD\_PATH=~/riscvtool/riscv-openocd-0.10.0-2019.08.2-x86\_64-linux-ubuntu14
2. export PATH=$PATH:~/riscvtool/riscv-qemu-4.1.0-2019.08.0-x86\_64-linux-ubuntu14/bin
3. export RISCV\_PATH=~/riscvtool/riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86\_64-linux-ubuntu14/

(2)在 **ubuntu** 终端使用如下命令编译 **SiFive learn inventor** 对应的 **FreeRTOS** 示例程序 ***example-freertos-minimal***:

1. make PROGRAM=example-freertos-minimal TARGET=sifive-hifive1-revb LINK\_TARGET=freertos software

编译得到一个 ***elf*** 文件，该文件自动保存在该示例程序文件夹下的 ***debug*** 文件夹内。

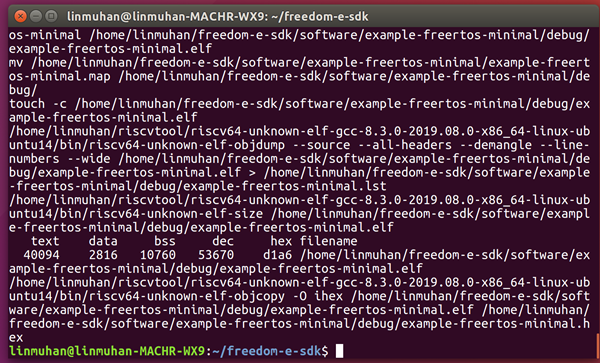


图 5：FreeRTOS移植

（3）打开 **FreedomStudio**，以“*C/C++ Executable*”的形式导入 ***elf*** 文件，生成一个单独的工程。将该工程烧录到开发板上，串口打印如下信息，则说明 **FreeRTOS** 移植成功。

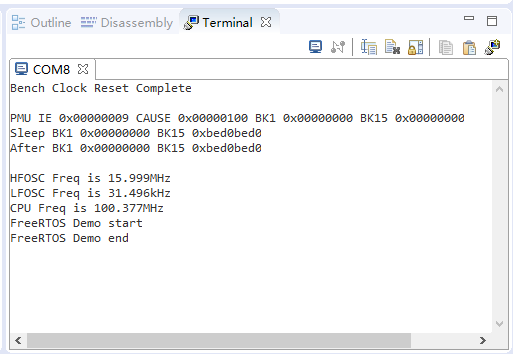


图 6：移植成功界面

# 三、总结与展望

## 3.1 总结

本项目使用**Microbit**小车搭载**RISC-V** **SiFive Learn Inventor** 芯片，实现了小车的循迹功能，通过调用**PWM**函数实现对小车速度的控制。为了实现多线程控制，我们引入了**FreeRTOS**，最终达到了多程序并行执行的效果**。**在本项目报告中，我们详细介绍了项目实施的过程并展示了实验结果。

## 3.2 展望

通过本项目，我们对RISC-V芯片架构有了充分的了解，我们将在后续的工作中进一步完成智能小车的循迹避障功能，并尝试使用其他传感器，如激光雷达，深度摄像头等，实现对小车的控制。

# 四、团队成员及指导老师介绍

## 4.1指导老师介绍

潘文安教授先后毕业于香港中文大学，日本筑波大学和美国南加州大学，并分别获得工学学士，硕士以及哲学博士学位。潘教授于2015年加入香港中文大学（深圳）前，长期在世界500强公司科研机构包括华为（美国），三菱（波士顿）和 索尼（东京）从事前沿科研工作。潘教授现在的研究方向包括智慧物联网以及机器学习在通信和卫星遥感领域的应用。他担任港中大（深圳）-中科辰新卫星通信与遥感技术联合实验室主任(2020-2025)以及深圳市物联网智能系统与无线网络技术重点实验室副主任(2018-2020)。

## 4.2项目成员介绍

方文欣，林沐晗均为2019级本科生，就读于香港中文大学（深圳）电子信息工程专业计算机工程方向。在本项目中，方文欣主要负责电子电路方面的硬件设计与调试，林沐晗主要负责FreeRTOS多线程控制。二人共同完成所需算法设计及程序编写。

# 五、附录

## 5.1小车控制主程序



## 5.2 PWM函数





* ***PWM\_run***：控制小车以一定速度直行运动。
* ***PWM\_left***：控制小车以一定速度右转：左轮以一定速度运转，右轮不运转。
* ***PWM\_right***：控制小车以一定速度左转：右轮以一定速度运转，左轮不运转。
* ***PWM\_back***: 控制小车以一定速度倒车。
* 函数输入值 *speed* ：控制低电平输出时间。
* ***GPIO\_SET（uint32\_t pin\_num, uint32\_t pin\_val, uint32\_t pin\_model)***:

使能相应 **GPIO** 引脚，并设置其为输入或输出以及其输入或输出的值（均通过计算得到所需寄存器地址后将1或0写入寄存器相应位实现）

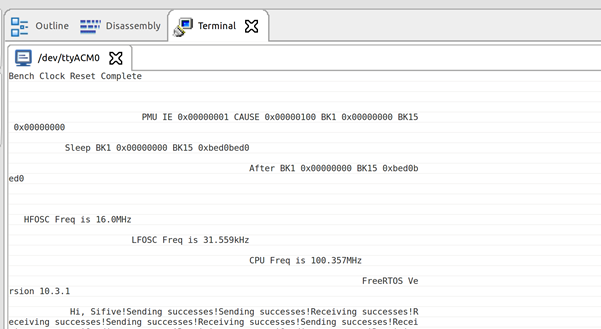
1. 参数 *pin\_num*: 设置的 **GPIO** 引脚的序号。*car\_motor\_left\_forward、car\_motor\_left\_backward、car\_motor\_right\_forward、car\_motor\_right\_backward* 均为存储电机对应的 GPIO 引脚序号的常量。
2. 参数 *pin\_val*: 该引脚的输出/输入值。1表示高电平，0表示低电平。
3. 参数 *pin\_model*:该引脚为输入或输出。可填入的值为 *input* 和 *output*（在相关文件中已将 *output* 定义为1，*input* 定义为2）。

* **PWM** 函数控制原理：在 **PWM** 函数中，只有当 *i* 大于 *speed* 时，驱动电机的 **GPIO** 引脚才会输出高电压。因此 *speed* 越大，输出高电压的时间越短。通过使用 *for* 循环，能够模拟一定占空比的交变电流，从而实现 **PWM** 控制小车速度。

5.3 FreeRTOS发送信息代码及输出



* 程序运行输出



# 六、参考文献

【1】周亦敏, and 魏洪兴. "嵌入式RISC处理器体系结构并行技术的研究." *计算机科学 34.001(2007):262-263,277*.

【2】东野长磊. "基于现场可编程门阵列的RISC处理器设计." *计算机工程* 37.11(2011):242-244.

【3】杨树长, and 魏平. "基于RISC处理器的实时交通计测系统." *北京理工大学学报* (1999).

【4】梁平. "怎样做大RISC-V的蛋糕?." *单片机与嵌入式系统应用* 019.006(2019):3-5.

【5】刘滨, 王琦, and 刘丽丽. "嵌入式操作系统FreeRTOS的原理与实现." *单片机与嵌入式系统应用* 000.007(2005):8-11.

【6】张龙彪等. "嵌入式操作系统FreeRTOS的原理与移植实现." *信息技术* 11(2012):31-34.

【7】余晓光. 基于实时操作系统FreeRTOS的Lwip协议的移植研究. Diss. *昆明理工大学*, 2013.