摘 要

无人驾驶领域一直是比较热门的研究领域，前人已经提出了大量的自动巡航的解决方案。并将之运用到智能飞行器，智能小车等各个领域。在此背景下，本系统主要研究有可靠标识路段的自动巡航检测的智能小车。以实现智能小车的自动巡航功能。该系统实现了，红外寻迹的逻辑检测功能模块，2.4G的通信功能模块，电机的驱动处理功能模块，命令处理功能模块，姿态获取处理模块，以及基于编码器的速度的算法处理功能模块等。系统的最终测试结果比较稳定，各个程序之间不会相互干扰，死机的可能性很小。但是由于技术支持的原因，没能实现产品的微型化，和外观的美化，无法达到商品级别的标准，只能算一个产品的功能模型。

**关键字：**智能小车，自动巡航，姿态获取

ABSTRACT

The field of unmanned driving has always been a hot research field. Predecessors have put forward a large number of automatic cruise solutions. And apply it to intelligent aircraft, intelligent car and other fields. In this context, this system mainly studies the intelligent car with automatic cruise detection with reliable identification of road sections. To realize the automatic cruise function of the intelligent car. The system realizes the logic detection function module of infrared tracing, the communication function module of 2.4G, the driving processing function module of motor, the command processing function module, the attitude acquisition processing module, and the algorithm processing function module based on the speed of encoder. The final test result of the system is relatively stable, each program will not interfere with each other, and the possibility of crash is very small. However, due to technical support, the miniaturization of products and beautification of appearance can not be realized, which can not meet the standard of commodity level. It can only be regarded as a functional model of a product.

**Key Words:** intelligent car,automatic cruise,Attitude acquisition

**目录**

[第1章 绪论 1](#_Toc25067)

[1.1 研究意义 1](#_Toc19974)

[1.2 国内外研究现状 1](#_Toc6917)

[1.2.1 国内研究现状 1](#_Toc25795)

[1.2.2 国外研究现状 2](#_Toc1861)

[1.3 本设计目标及内容安排 2](#_Toc10937)

[第2章 整体方案设计 4](#_Toc16608)

[2.1 实现功能描述 4](#_Toc22674)

[2.2 硬件用到的主要模块 4](#_Toc12282)

[2.2.1 MPU6050初始化流程图 4](#_Toc1916)

[2.2.2 DMP使用 5](#_Toc31799)

[2.2.3 寻迹逻辑程序设计 5](#_Toc12020)

[2.2.4 电机驱动部分程序设计 5](#_Toc7139)

[2.2.5 通信部分程序设计 6](#_Toc24013)

[2.2.6 Oled部分程序设计 6](#_Toc30043)

[2.2.7 遥控部分逻辑设计 7](#_Toc28801)

[2.2.8 关于freerots的简单介绍和移植 7](#_Toc27790)

[2.2.9 CubeMX对底层BSP(板级支持包)的初始化配置 8](#_Toc22295)

[2.3 本章小结 9](#_Toc17884)

[第3章 硬件设计 11](#_Toc20254)

[3.1 电源模块 11](#_Toc20635)

[3.2 最小系统开发板 11](#_Toc23175)

[3.2.1 最小系统板电路 12](#_Toc7943)

[3.3 寻迹红外模块 13](#_Toc17009)

[3.3.1 红外检测模块电路 14](#_Toc32358)

[3.4 前轮舵机 14](#_Toc11547)

[3.5 TB6612FNG模块说明 16](#_Toc14765)

[3.5.2 TB6612带稳压模块板原理图 17](#_Toc18584)

[3.6 0.96OLED显示屏 19](#_Toc16871)

[3.6.2 0.96OLED显示屏原理图 20](#_Toc22524)

[3.7 NRF24L01 无线模块简介 20](#_Toc29478)

[3.7.2 nRF24L01 电路 26](#_Toc7962)

[3.8 MPU6050简介 26](#_Toc8662)

[3.9 本章小结 30](#_Toc18701)

[3.9.1 小车实物展示和小结 30](#_Toc15254)

[3.9.2 手柄实物展示和小结 30](#_Toc2077)

[第4章 软件设计 31](#_Toc24594)

[4.1 软件工具 31](#_Toc25028)

[4.1.1 软件工具MDK5 简介 31](#_Toc14703)

[4.1.2 Oled显示屏专用取模工具 31](#_Toc28634)

[4.1.3 CubeMX 31](#_Toc22851)

[4.1.4 Git 31](#_Toc26681)

[4.1.5 串口调试工具 32](#_Toc17975)

[4.2 主程序APP部分代码 32](#_Toc1615)

[4.2.1 小车部分app 32](#_Toc1982)

[4.2.2 手柄部分app 39](#_Toc3708)

[4.3 主程序BSP部分代码 45](#_Toc10380)

[4.3.1 小车部分bsp 45](#_Toc8499)

[4.3.2 手柄部分bsp 58](#_Toc11690)

[4.4 本章小结 59](#_Toc1068)

[第5章 系统的调试及结论 60](#_Toc12133)

[5.1 调试中遇到的重点与难点和解决方案 60](#_Toc31279)

[5.1.1 MPU6050的DMP调试出现FIFO溢出问题。 60](#_Toc27233)

[5.1.2 红外检测过慢，导致小车错过检测跑道时机，而跑出跑道。 60](#_Toc17989)

[5.1.3 Pwm过低，转弯的驱动能力不足。 61](#_Toc4762)

[5.1.4 NRF24L01初始化失败 61](#_Toc7499)

[5.1.5 NRF24L01动态字节配置失败。 61](#_Toc16153)

[5.1.6 Oled初始化失败问题 62](#_Toc8506)

[5.1.7 手柄程序出现程序死机的情况 62](#_Toc15527)

[5.1.8 用定时器的模块太多，资源不够用。 63](#_Toc7079)

[5.2 本章小结 63](#_Toc18116)

[参考文献 64](#_Toc6468)

[致谢 65](#_Toc25005)

[附录 66](#_Toc22123)

[附录一：手柄的部分.h文件 66](#_Toc9525)

[附录二：小车的部分.h文件 71](#_Toc11513)

1. 绪论
   1. 研究意义

自动巡航和无人驾驶技术一直是热门的研究技术，前人已经提出了许多优秀的检测方法和算法去实现小车的智能驾驶。但是总体的解决方案的价格都是高昂的。在某些运用场景上，这些价格高昂的技术就显得十分的鸡肋。所以，本课题在于研究如何利用更低廉的单片机芯片去实现些简易，普通的自动巡航技术在简单赛道上的运用。并结合手动控制来完善自动巡航驾驶的不足，解决无人驾驶潜在的风险和威胁。

* 1. 国内外研究现状
     1. 国内研究现状

随着汽车科技技术的进步，对于智能小车的实验与设计越发重要．智能小车一致具有自动寻迹、躲避障碍物、报警等功能．其所运用的知识较为广泛，主要涉及到汽车、机械和计算机等专业．智能小车不但代表汽车技术的发展，也是学校培养机电一体化学生的学习能力与动手能力的一种主要手段。[1]

现如今国内的自动化技术发展极快，无人科技方面有极大的发展空间，自动寻迹避障车更是有极大的应用范围，小到餐饮服务业，大到太空勘探，都可将其应用其中，大大提高了工作效率，节约人工成本，在国外自动寻迹避障小车已有实际的应用，相信在不久的将来会被广泛普及。[2]

智能车( intelligent vehicle，IV) 又称轮式移动机器人，是一个集环境感知、规划决策、操作控制等功能于一体的智能体，涉及机械、运动学与动力学、电子、计算机、信息处理、控制和人工智能等科学技术领域。在智能车技术的运用中，除了能够让其自动识别各种不同形式的道路外，如何让其达到并维持较高的运行速度，是智能车控制的一个关键问题。[3]

国内在智能寻迹领域的研究论文很多是基于stc89c51和c52单片机去实现的，但是众所周知，这是一款老旧的单片机芯片，代码编写基于对寄存器的操作，并没有方便调用的硬件底层库文件。虽然主控芯片已经不值得我们学习了，但是其中对寻迹的设计是值得我们学习的，所以在红外寻迹部分，我们参考了单片机通过路径识别模块对路径信息进行采集，以实现小车寻迹的功能。系统选用 RPR220 型红外反射式光电传感器设计了路径识别模块。RPR220 是一种一体化反射型光电探测器，其发射器是一个砷化镓红外发光二极管，接收器是一个高灵敏度硅平面光电三极管。[4]红外发射管发出的红外光在遇到反光性较强的物体（表面为白色或近白色）后被折回,并被光电三极管接收到,引起光电三极管光生电流的增大，将这个变化转为电压信号,就可以被处理器接收并处理,进而实现反光性差别较大的两种颜色（如黑白两色）的识别。[5]

* + 1. 国外研究现状

国外已经研究了一种自动驾驶系统的专利技术，用于自动驾驶车辆跟随各自的目标运行轨迹。基于障碍物检测信号、车辆位置信号、道路数据和自动驾驶交通信息信号以及车对车运行信息信号设置运行指令和运行路线。在一辆车和另一辆车之间传输和接收指示行驶路线的行驶路线信号和指示设置行驶路线时一辆车的行驶位置的行驶位置设置位置信号，以便可以根据接收到的行驶路线计算适合每辆车的目标行驶轨迹，从而自动导航每辆车，使相应的车辆按照生成的目标运行轨迹运行。[6]

并且在实际运用上也有很大突破，他们可以创建一个数字路线图提供了有关道路车辆运行环境的参数数据。如果道路车辆配备了摄像机，机器视觉方法可以提供有关车辆周围实际交通环境的参数数据。在私人道路网络上自动驾驶面包车调试期间结合两种方法的经验，包括在随后的公共道路驾驶试验中收集的经验，以及硬件和软件的几个改进周期。基于这些经验，设计并调试了第二代自动驾驶汽车。目前在公共道路上对其进行评估。[7]

然后，在赛博汽车和双模车辆方面，在目前是最具创新性的车辆自动化应用试验台。定义不同自动车载系统的标准和控制架构是构建最终原型的必要任务。在移动机器人领域，已经做出了一些经典的架构定义。这些体系结构能够处理传感器输入、环境和程序知识，以管理移动机器人的不同执行器，从而完成其任务。自动车辆被认为是移动机器人技术和车辆技术领域之间的一个纽带，可以获得与移动机器人一样具有自主性，但与机器人相比，在高需求环境和不同条件下流通的汽车。[8]

并且提出了一种自动车辆通过交叉口的控制方法。如果接近交叉口的自动车辆相互通信，并通过较小的减速和加速来安排进入交叉口的时间，车辆可以通过交叉口而不停车，从而通过减少不必要的减速来实现节能。提出了一种对交叉口模式进行分类的方法，为每个交叉口模式提取实现不停车交叉口所需的条件，执行控制以实现此类交叉口，并将信息传输给进入交叉口的车辆，以创建虚拟排。通过单车道单向交通交叉口的试验验证了该方法的有效性。[9]

模糊道路的自动跟踪是视觉导航系统的重要组成部分。国外提出了一种自适应方法，该方法使用梯形形状内路面颜色的统计模型，该形状近似于道路在图像平面上的投影。该方法不执行图像的显式分割，而是横向扩展形状，直到形状和道路之间的匹配恶化，同时计算颜色统计。结果表明，该方法能够在各种情况下以所用机器人的典型行驶速度反应性地跟踪道路，同时应对道路的各种条件，如路面类型、水坑和阴影。使用大量具有地面真实性的数据集对所提出的方法进行了广泛的评估。此外，在道路跟踪的背景下评估了许多颜色空间，发现将亮度与颜色信息分离的颜色空间表现最好，尤其是在亮度信息被丢弃的情况下。[10]

* 1. 本设计目标及内容安排

用STM32f103c8t6作为微控制器，设计一款智能小车。小车能实现自动巡航，可前进后退，转弯。小车具有手动控制模式，能使用远程设备解除智能小车的自动巡航。实现手动控制小车的行进。自动巡航的跑道采用黑色线纸组成的一圈半径约为25cm环形跑道，小车在自动巡航的时候无需外部控制，可完成一圈巡航。手动控制器与小车分离，且采用合适的无线通信技术跟小车进行通信。小车可以测速，并通过速度反馈控制电机的驱动能力，使速度恒定。小车的转向模拟真实的车辆前轮左右转向。手动控制器需要屏幕显示。

1. 整体方案设计
   1. 实现功能描述

小车实现了自动沿着跑道行进，而无需人为干涉。在行进过程中，小车始终保持匀速。在转弯的过程中，舵机会改变转动角度去改变前轮的转动方向。并且随时可以通过遥控手柄，取消小车的自动驾驶，改为手动操作。小车进入手动控制模式后，遥控手柄可以通过改变MPU6050的姿态数据，给小车发送不同的操控指令完成对小车前进，后退，左转，右转的操控。

* 1. 硬件用到的主要模块

最小系统stm32C8T6模块，两个拥有编码器的的直流电机，直流电机的驱动模块TB6612模块，0.96OLED显示屏,mpu6050模块，可充放电的锂电池，NRF24L01通信模块，转向舵机G90和升压降压模块LM2596S.

* + 1. MPU6050初始化流程图



MPU6050初始化流程图

* + 1. DMP使用

MPU6050的DMP输出的是姿态解算后的四元数，采用的q30的格式，也就是放大了二的三十次方，要得到欧拉角，就得做一个转换，代码如下：q0=quat[0] / q30;q1=quat[1] / q30;q2=quat[2] / q30;q3=quat[3] / q30; pitch=asin(-2 \* q1 \* q3 + 2 \* q0\* q2)\* 57.3; roll=atan2(2 \* q2 \* q3 + 2 \* q0 \* q1, -2 \* q1 \* q1 - 2 \* q2\* q2 + 1)\* 57.3;yaw=atan2(2\*(q1\*q2+q0\*q3),q0\*q0+q1\*q1-q2\*q2-q3\*q3)\*57.3;quat[0]~quat[3]：是MPU6050的DMP解算后的四元数，q30格式。q30：是一个常量：1073741824，即2的30次方。 这些数据将显示在oled的液晶上，方便驾驶员对手柄进行操作。

* + 1. 寻迹逻辑程序设计

我们在小车前端的左侧和右侧各自安装了一个红外探测装置。初始情况下，轨道会位于这两个红外装置的中间。此时，轨道两侧的部分红外反射率高于轨道，红外探测装置便会接受到红外信号。所以，探测信号的输入为左右两个红外检测装置都检测到红外信号。单片机得到此信号的时候，会产生中断事件，中断事件会将创建的红外事件对应的标志位置1。然后，系统的红外等待任务事件会被触发，任务开始运行，进行一次电平检测任务。此时，检测到的信号状态应该为接收到红外信号。然后执行小车前进的程序。

当轨道向左偏移时，小车前端左侧的信号检测将会首先检测不到返回的红外波段。此时的信号应该为左侧检测到红外信号，右侧没有检测到红外信号。此时将运行舵机左转前进的程序代码。当轨道向右偏移的时候，同理。为了更清晰直观的展示，见下表。

红外寻迹逻辑

|  |  |  |  |
| --- | --- | --- | --- |
| 道路状态 | 左检测信号 | 右检测信号 | 小车运动状态 |
| 直线 | 有 | 有 | 前进 |
| 向左偏移 | 无 | 有 | 向左前进 |
| 向右偏移 | 有 | 无 | 向右前进 |
| （小车被拿起） | 无 | 无 | 停止 |

* + 1. 电机驱动部分程序设计

电机驱动的每一个轮子都需要两个单片机输出引脚和一个PWM输出引脚去控制。从表2-1 TB6612FNG输出真值表可以得到电机运行状态。当同时拉高拉低输出引脚时，电机会处于停止状态。当拉高其中一个引脚，拉低另一个引脚时，电机会转动起来。我们将此时的转动方向规定为正，那么对调刚才的输出，电机便会朝相反方向转动。当然此时需要输入PWM波。而PWM的频率是10khz到100khz。我们设置为10hz驱动，然后调节PWM输出引脚的输出占空比，便可以达到速度的调节。

每个电机的编码器的信号脚为两个。采用的是霍尔码盘采样，两路信号脉冲输出。而是stm32c8t6自带硬件编码器接口，并且在CubeMX中很好配置。首先，我们选择一个定时器，然后在定时器中配置Combined Chanels为Encoder Mode，在Parameter Settings中配置Encoder的各项参数。我们配置为四倍频计数，能大大增加测量的精度。

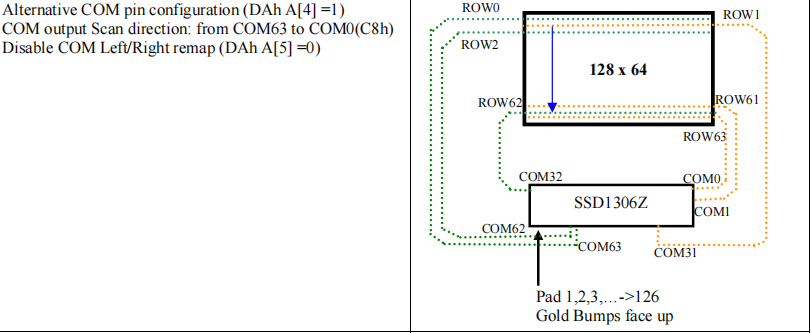
配置完毕后，只要连接对应的单片机引脚，从代码中我们便可以使用hal库函数里的api接口得到轮子的编码器计数等信息。电机减速比为30，霍尔编码器精度13，AB双相组合得到4倍频，则转1圈编码器读数为30\*13\*4=1560。所以读数间隔时间如果为0.1s。那么电机的速度就等于编码器读数除以1560乘以10圈每秒。轮子的半径为3厘米，所以小车的速度就等于电机的速度乘以2π乘以3cm每秒。

* + 1. 通信部分程序设计

NRF24L01使用的硬件spi通信，使用了DMA技术进行数据的转移存储，DMA的配置在cubemx这个软件上很好配置。配置完毕后就可以生成对应的DMA代码。为了能让小车和手柄之间能够保持双向的通信。都能进行发送和接受数据，我们将发送程序设计为，当信息发送完毕后，将自动关闭发送模式，打开接受模式。

* + 1. Oled部分程序设计

Oled使用的是软件spi通信配置。在初始化中，我们需要先拉高RST引脚100ms，再拉低RST引脚200ms后再拉高。然后对oled的进行命令写入。首先我们先设置页的寻址地址下列起始地址为0.为页面寻址模式设置跟高列的起始地址为0。然后再设置显示的起始地址0x40。设置BANK0的对比度控制为0，这个可以理解为SEG的输出电流亮度。然后设置段的重映射，扫描方向和显示器显示为正常模式。然后设置63多路模式的多重比，显示偏移为0。设置显示时钟分比为1，将时钟振荡器频率置为100帧/秒。设置预充电周期占空比为1/2。设置COM引脚硬件配置。如下图：



COM信号的引脚配置

设置VCOMH调节器的输出为0，降低oled功耗。设置内存寻址模式为页面寻址模式。在页面寻址模式下，读写显示RAM后，列地址指针自动增加1。如果列地址指针到达列结束地址，则列地址指针将被重置为列起始地址，而页面地址指针将不会被更改。用户必须设置新的页面和列地址，才能访问下一页RAM内容。然后打开液晶屏显示就可以了。

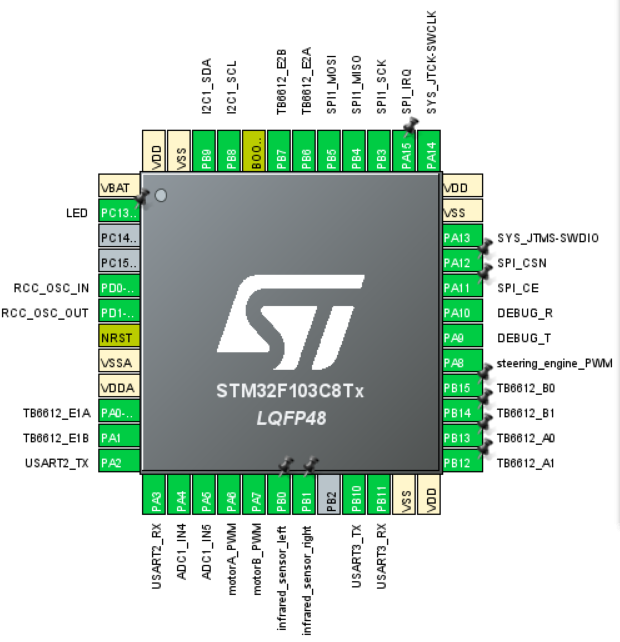
* + 1. 遥控部分逻辑设计

手柄的通过对MPU6050的三个姿态角的获取的数据，根据不同的姿态给小车发送不同的姿态命令。当手柄向前倾斜，小车便会接收到前进的命令。然后根据倾斜的角度大小设置不同的速度。当手柄向后倾斜的时候，小车接收到后退的指令，同样根据倾斜角度设置速度。当手柄向左倾斜时，舵机会向左转动。手柄向右倾斜时，舵机会向右转动。当手柄保持水平是，小车会停止移动，并且舵机的会转动到直线运动位置。但是，手柄必须先向右转动，给小车发送指令，让小车停止寻迹模式，进入手动操控模式。如果成功与小车建立连接，手柄会接收到小车的返回信息。并将小车运行状态信息返回到oled屏幕上显示。如果要取消手动模式，手柄左旋转即可。小车将再次进入自动巡航模式。

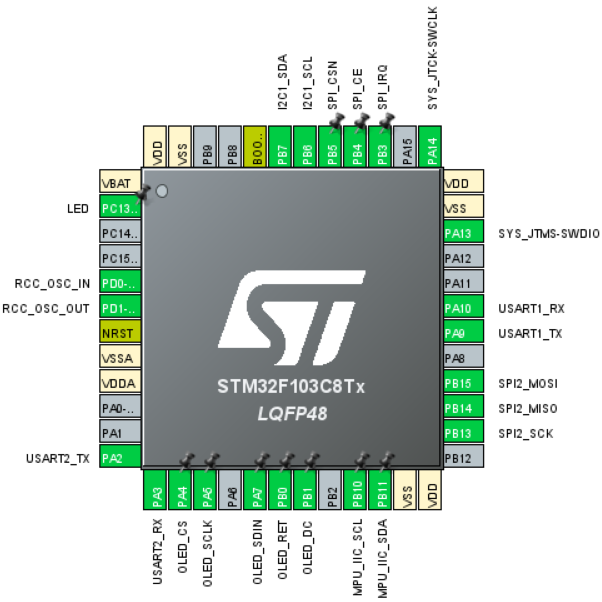
* + 1. 关于freerots的简单介绍和移植

单片机的指令执行是单线程。平时，我们学习测试时，也是将整个程序放在一个main函数里执行。但是，这样程序在应对一些需要及时响应的情况时，只能靠中断去处理。我们经常会遇见程序中，某个代码需要等待一段时间，继续执行的情况。如果是单线程，程序就会停用在这个代码位置，从而造成CPU的资源浪费。为了解决这种为题，操作系统就孕育而生了。它可以让系统在等待的情况下，去执行另外的一段代码，从而充分利用CPU资源。而对于嵌入式单片机，freerots作为一款小型的实时操作系统，就能满足那些内存空间不大的单片机，对操作系统的需求。而移植这款操作系统，我们只需要在官网去下载它的源代码，就能轻松的移植到stm32单片机上。如果运用cubemax去配置源代码，我们甚至可以直接一键生成搭载了freerots操作系统的代码模板。

* + 1. CubeMX对底层BSP(板级支持包)的初始化配置



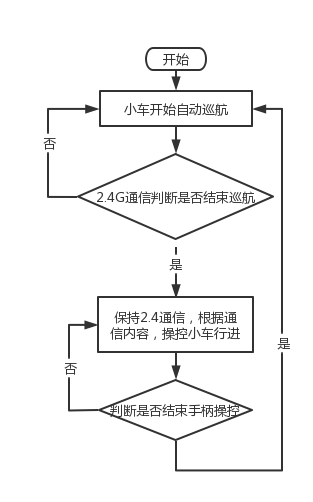
小车部分配置



手柄部分配置

* 1. 本章小结

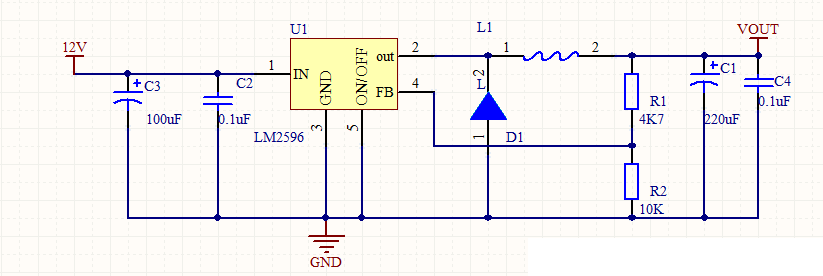
程序运行最终实现的效果是将小车放置在预定轨道上，然后将TB6602模块上的电源开关打开。因为锂电池是直接给该模块供电的。该模块还有一端电源输出引脚接降压模块输入端，然后输出3.3V电源给核心板控制板供电。程序开始运行自动寻迹程序，开始寻迹任务，小车将自动沿着跑道行进。同时小车一直在维持和手柄控制器之间的2.4G通信。当接受到手柄控制发送的手动操控信号后，小车停止自动巡航模式，进入手动操控模式。此时，手柄可以直接对小车进行操控。直到接收到自动巡航信号再次进入自动巡航模式。



小车运行的主要逻辑流程图

1. 硬件设计
   1. 电源模块

小车的电源模块，采用了可充放电的12v锂电池供电，并采用DC-DC直流可调的降压稳压电源模块板进行降压处理成3.3v。手柄的电源模块采用了可充放电的3.7v锂电池供电，同样配置了降压模块进行降压。关于LM2596S这款芯片，一共有五个引脚，一个GND引脚，一个电压输入引脚，一个电压输出引脚，一个控制芯片是否工作的引脚。和一个feedback的反馈引脚。

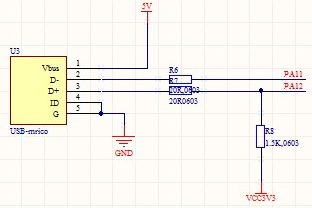
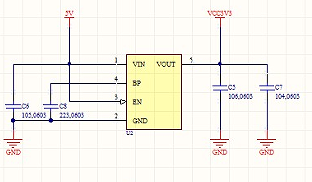
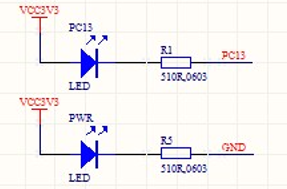
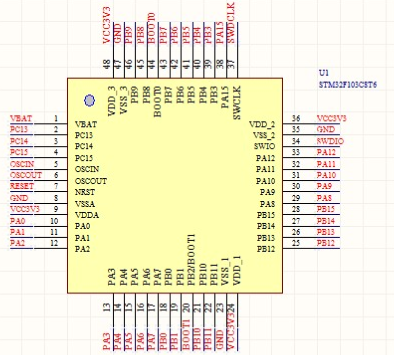


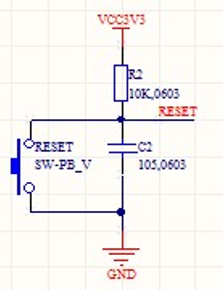
LM2596S简易运用电路示例

* 1. 最小系统开发板

小车和手柄的采用了基于stm32f103c8t6的最小系统控制板。基于ARM Cortex-M 内核STM32系列的三十二位的微控制器嵌入式操作系统，拥有从六万四千或十二万八千字节的闪存flash程序存储器，高达两万字节的SRAM。工作电压两伏到三点六伏，温度区间为负四十摄氏度到八十五摄氏度。拥有最高七十二MHz的工作频率，在存储器的等待周期访问时可达五分之四DMips/MHz。拥有两个I方C接口(支持SMBus和PMBus)。三个USART串口通信接口。两个个SPI通信接口(十八兆比特每秒)。

* + 1. 最小系统板电路





最小系统板C8T6主要部分电路图

* 1. 寻迹红外模块

在小车的前端配置了左右两个红外传感器模块。该红外传感器模块对环境光线的抗干扰能力强，拥有一对红外线发射装置与接收装置，发射装置发射出某一波段的红外线，当发射红外线方向遇到拥有反射红外线能力的反射面时，红外线被反射回来，被接收设备接收，然后通过比较器LM393电路进行信号处理之后，接受信号的LED灯会亮起。同时，输出接口OUT引脚输出一个数字低电平信号。拥有电位器旋钮调节按钮，方便调节红外检测距离，有效距离范围是二到三十厘米，工作电压区间为三点三伏到五伏。因为模块发射端会主动发射红外线去检测,所以影响红外探测距离的重要因素是探测对象的颜色红外反射率和形状。其中黑色对红外线的吸收能力最强。红外探测距离因此会减小。相反白色是反射能力最强的。面积不大的对象的探测距离小,大面积的对象的探测距离大。输出端口的OUT引脚可以直接驱动一个五伏的继电器，也能直接和MCU的IO口引脚连接，；连接方式：VCC-VCC;GND-GND;OUT-IO。电路板尺寸为3.2CM\*1.4CM。

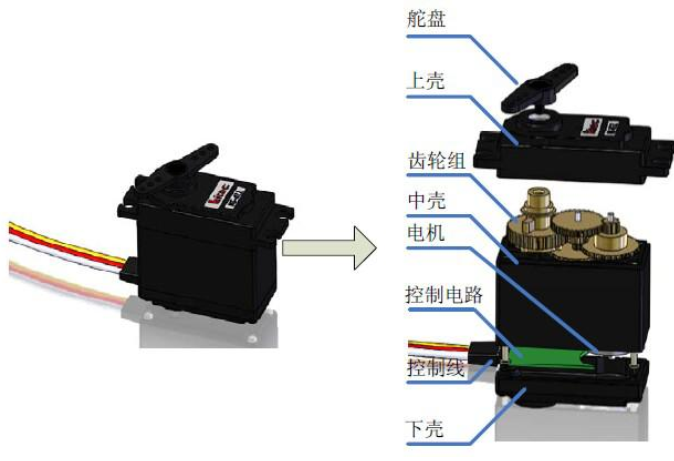
* + 1. 红外检测模块电路



红外检测模块电路图

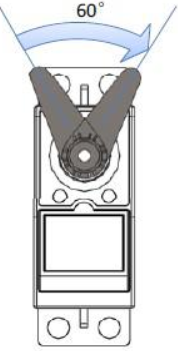
* 1. 前轮舵机

舵机根据其控制电机的方式可以称为微型伺服马达。因为它早期在模型上主要用于控制模型的舵面，所以市面上又被称为舵机。舵机能够接收一个简易的操作命令，便能够自动旋转到操作命令对应的较为精准的位置，所以十分方便在关节型结构机器产品中运用。可以较为准确的操控和维持输出轴的转动角度。直流电机这样的控制方式又叫做闭环控制，所以舵机更准确的名称为伺服马达，英文名servo。



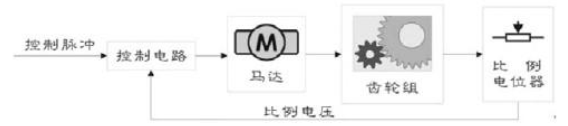
舵机的结构

舵机的主要构成如图 2-1 所示，分别为舵盘、上壳、减速齿轮组、中壳、电机、控制电路、控制线、下壳。舵机的外壳普遍是高分子聚合物材质的，特殊的可能会有金属铝合金材质的外壳。金属外壳能更牢靠的固定在目标地点。伺服马达的齿轮箱的种类有：塑料、混合、金属等差别。塑料的噪音小，成本底，但是强度一般不大；金属的强度虽然很高，但是成本也很高，如果装配的精度不行，就会产生很大的噪音。小扭矩的微舵，扭矩虽然很大，一般所有用塑料齿轮的伺服马达功率密度小，如辉盛的九克金属齿轮微舵主要用在功率密度很高的伺服马达上，比如辉盛的MG995伺服马达却能提供十三千克的扭矩。Hitec甚至用钛合金作为齿轮材料，其强度能提供二十几公斤的扭矩。混合类型的在塑料和金属之间做了平衡，在扭矩不大的部位，如电机输出减速箱的地方用塑料齿轮。

伺服马达的规格和选型：1伺服马达转速，伺服马达无负载的条件下，转速在转过六十度角时，所需时间的长短。伺服马达常见的速度一般在零点零一秒到零点二一秒每六十度之间。2伺服马达转矩，伺服马达扭矩的单位是千克每厘米。

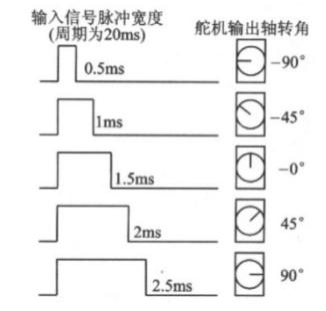
舵机的转速和转矩

舵机的工作理论：舵机是一个微型的伺服控制系统，具体的控制原理可以用下图表示。



舵机的控制原理

通过控制电路接收信号源的控制信号，并驱动电机转动是基本的运行原理；电机的速度被齿轮组成倍数的缩小，电机的输出扭矩则会放大响应的倍数，然后输出给外部；齿轮组跟电位器的最后端口会同时旋转，通过舵机轴的旋转角度的测量；电路板会进行检测，并且依照电位器去判断舵机旋转的角度，最后再操控舵机旋转到对象角度或者停留在对象角度。舵机需要一个外部的控制终端产生的脉宽调制信号去控制舵机旋转角度，舵机脉冲的宽度是控制终端必要的编码信息。舵机的控制脉冲一般周期是二十毫秒，脉宽从零点五毫秒到二点五毫秒，分别对应着旋转到负九十度和正九十度的位置。如下图所示：

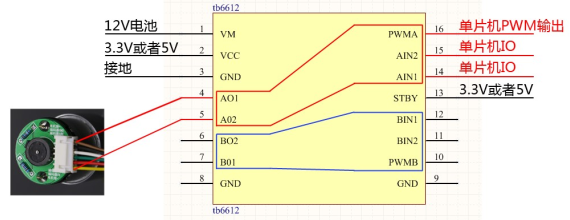
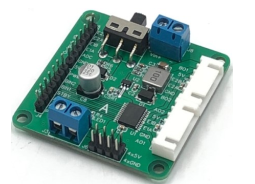


舵机信号脉冲示意图

* 1. TB6612FNG模块说明

TB6612FNG对比L298N，它无需额外的散热片，拥有更好的热耗性。外围电路比二极管续流电路更简单。增加了电源的滤波电容便能够直接驱动电机，便于降低系统的尺寸。PWM信号的输入频率高达十万赫兹，能够满足绝大部分的设计要求。

带稳压模块版说明：新增五伏稳压电路，支持五伏两安的输出。共有四个这样的引脚可以对外部供电。增加电压测量电路，通过串联一个一万欧姆和一个一千欧姆的电阻，对输入电源进行十一分之一的分压后，可以通过ADC进行采集并计算得到电源电压进行监控。引出电机标准的六个引脚接口，可以直接用排线连接，AB相编码器的信号接到单独的引脚输出。新增电源输出电路，可输出与输入的电源保持一致的电源。含有电源开关，可以对板子供电进行开启和关闭的操作。



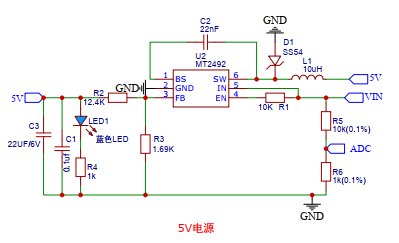
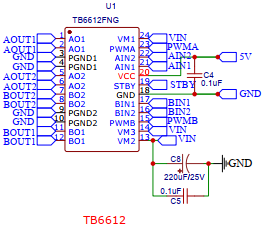
TB6612FNG实物图和引脚连接示意图

必须要有PWM波输入才有AO1和AO2的信号，只接AIN1和AIN2不会产生AO1和AO2的信号。VM引脚接电源正极，VCC引脚接内部的逻辑供电，模块的GND接电源地，一个接单片机地，STBY引脚置高，模块才能正常运行。上图中蓝色部分的五个引脚控制一路马达，红色部分的控制另外一路马达，AO1和AO2分别接到马达的正极和负极。然后通过PWMA、AIN2、AIN1控制电机。其中 PWMA接到单片机的PWM引脚，一般一万赫兹的PWM即可，并通过改变占空比来调节电机的速度。B路的使用是相同的。下面是真值表：

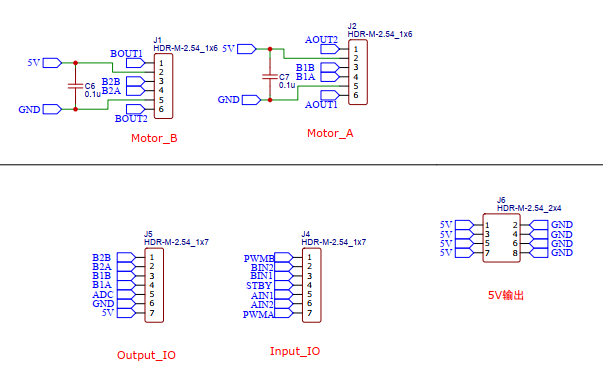
TB6612FNG输出真值表

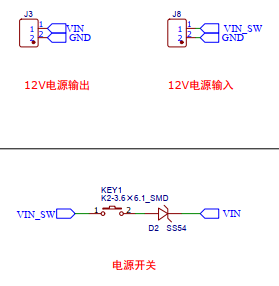
|  |  |  |
| --- | --- | --- |
| AIN1 | 0 | 0 |
| AIN2 | 0 | 1 |
|  | 停止 | 反转 |

* + 1. TB6612带稳压模块板原理图



TB6612芯片和电源部分

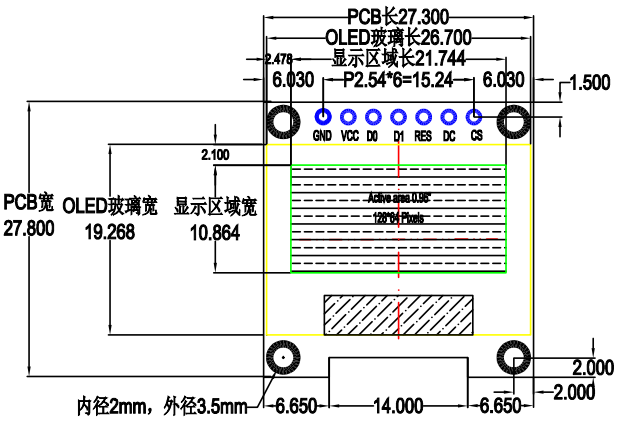




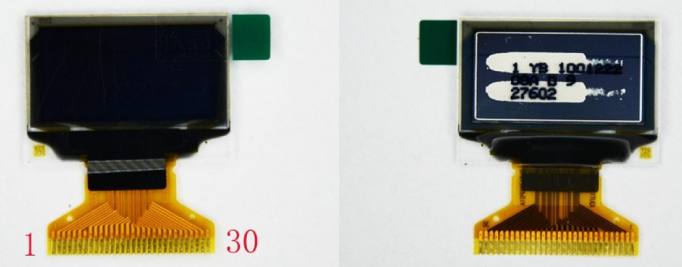
TB6612外接端口部分

* 1. 0.96OLED显示屏

OLED（Organic Light Emitting Diode），又称作有机发光二极管。LCD都需要背光，而OLED不需要，因为它是自发光的。因此同样的显示，OLED效果要好一些。在此我们使用的是中景园电子的0.96寸OLED黄蓝显示屏，该屏有以下特点：其中黄蓝是屏上1/4部分为黄光，下3/4为蓝；而且是固定区域显示固定颜色，颜色和显示区域均不能修改；白光则为纯白，也就是黑底白字；蓝色则为纯蓝，也就是黑底蓝字。分辨率为128\*64。多种接口方式，通过屏上的 BS0~BS2 来配置。

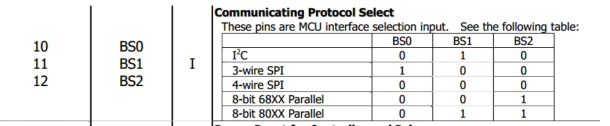


裸屏为 30pin，从屏正面看左下角为 1，右下角为 30；如下图所示：



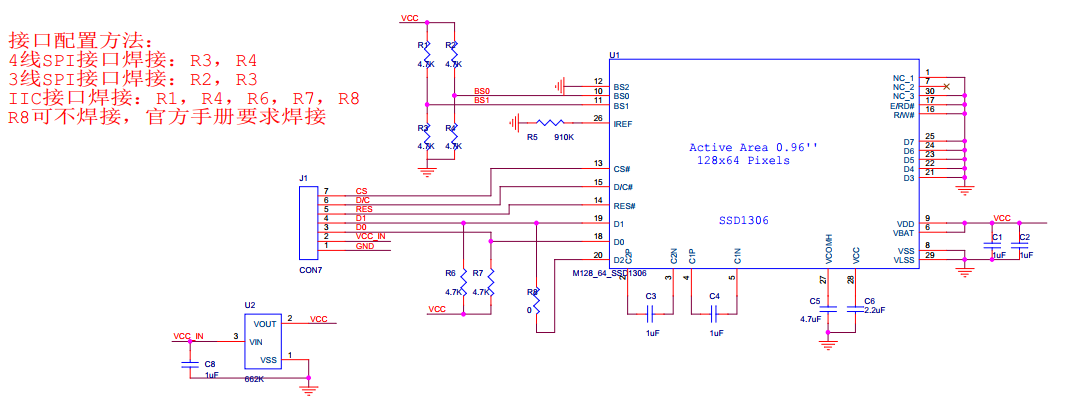
0.96寸OLED裸屏外观

0.96寸OLED驱动IC为 SSD1306；其具有内部升压功能；所以在设计的时候不需要再专一设计升压电路；SSD1306总共八页，每页拥有一百二十八个字节，刚好可以对应128\*64的点阵大小。七针SPI\_OLED有七个引脚，分别为VCC、D0、GDN、RES、DC、D1、CS。由OLED的数据手册我们可以知道 0.96 寸 OLED 裸屏是支持四种五种不同接口的。



通信接口是通过 BS0,BS1,BS2 三个管脚来配置的。

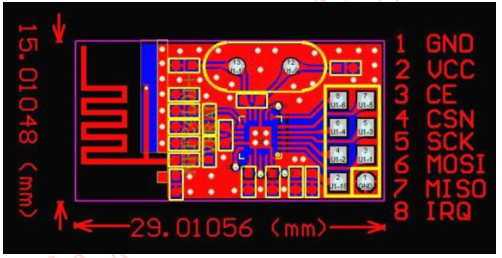
* + 1. 0.96OLED显示屏原理图



Oled显示屏引脚连接示意图

* 1. NRF24L01 无线模块简介

NRF24L01无线模块采用NRF24L01芯片，该模块拥有以下特征：一百二十五个频道可以选择，满足调频通信和多点通信的需要。免许可证使用的2.4G全球开放的ISM频段。内置CRC检错和点对多点的通信地址控制。高效的GFSK调制技术，两兆bps为最高工作速率，抗干扰能力强。低工作电压区间1.9到3.6V。可设置自动应答，确保数据可靠传输。所以在后面软件编程的时候 SPI 速度不能高于这个最大值。已经被大量使用，目前各方面表现都十分不错。该模块的外形和引脚图如图 2-8 所示：



NRF24L01 模块外观引脚图

nRF24L01可以设置为以下几种主要的模式：

nRF24L01的模式设置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 模式 | PWR\_UP | PRIM\_RX | CE | FIFO寄存器状态 |
| 接受模式 | 1 | 1 | 1 | - |
| 发送模式 | 1 | 0 | 1 | 数据在TX FIFO寄存器中 |
| 发送模式 | 1 | 0 | 1-0 | 停留在发送模式，直到数据发送完 |
| 待机模式2 | 1 | 0 | 1 | TX FIFO 为空 |
| 待机模式1 | 1 | - | 0 | 无数据传输 |
| 掉电模式 | 0 | - | - | - |

nRF24L01在不同模式下的引脚功能

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 引脚名称 | 方向 | 发送模式 | 接受模式 | 待机模式 | 掉电模式 |
| CE | 输入 | 高电平>10us | 高电平 | 低电平 | - |
| CSN | 输入 | SPI片选使能，低电平使能 | | | |
| SCK | 输入 | SPI时钟 | | | |
| MOSI | 输入 | SPI串行输入 | | | |
| MISO | 三态输出 | SPI串行输出 | | | |
| IRQ | 输出 | 中断，低电平使能 | | | |

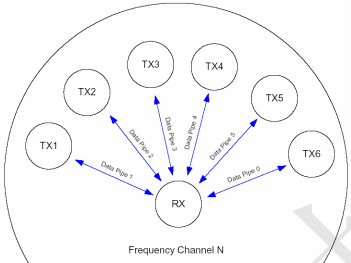
模式的介绍

|  |  |
| --- | --- |
| 待机模式 | 待机模式A可以在保证快速启动的同时并减少系统平均消耗的电流。晶振正常工作。当发送端FIFO寄存器为空并且CE为高电平时进入待机模式B。待机模式B下部分时钟缓冲器处在工作模式。寄存器配置字内容保持不变。 |
| 掉电模式 | 掉电模式由寄存器中PWR\_UP位来控制，进入后，nRF24L01停止工作，各功能关闭，电流消耗最小，但寄存器内容保持不变。 |

nRF24L01的数据包处理方式有：1.ShockBurstTM。2.增强型ShockBurstTM 模式。

ShockBurstTM模式：ShockBurst模式下能够与成本较低的低速单片机连接。该模式减小了通信的平均消耗电流，因为在与单片机低速通信的同时而在无线部分高速通信。在ShockBurstTM发送模式下，会自动生成前导码及 CRC校验。数据发送完毕后，IRQ中断会通知单片机。降低了单片机的查询时间，降低了单片机的工作量并且也降低了软件的开发时间。内部各有三个不同的 RX\_FIFO与TX\_FIFO寄存器。

增强型的ShockBurstTM模式：能够让双向链接协议运行更加简单、高效。经典的双向链接协议为：发送对象规定终端设备在得到数据信号后有应答信号，让发送对象检验是否有数据的丢失。如果数据发生丢失，便重新发送将丢失的数据恢复。该模式无需增加其它工作量便能够同时控制重发功能和应答。

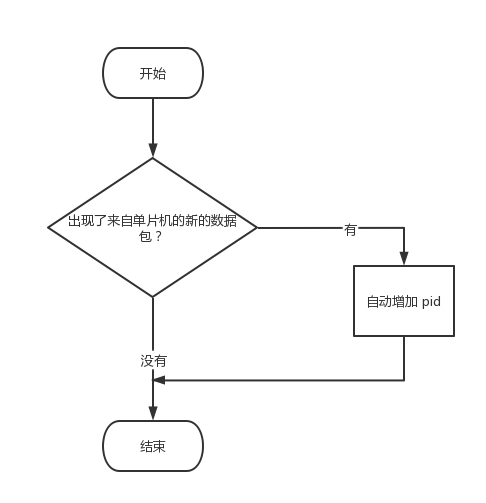


nRF2402在星形网络中的结构图

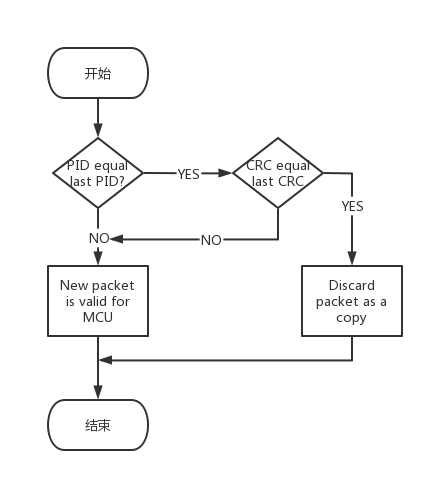
在接收模式下nRF24L01能够接收六路不同通道的数据，见图3-9。每一个通道都可以设置不同的通信地址，但是共用相同的频道。一个设置为接收模式的设备后能够与六个不同的设置为发送模式设备同时进行通讯。而设置为接收模式的设备能够对六个发射端进行身份识别。一到五的通道都拥有八位自身地址和三十二位公用地址。所有的通道都能够设置为增强型ShockBurst模式。通道零是唯一一个能够配置为 四十位自身地址的通道。在发送端，通道零被用做接收应答信号的通道。如果重发次数超过了设定值，则会产生中断。

在发送对象每次从单片机取得一包新数据后数据包识别值加一。

接收方：接收方对新接收数据包的数据包值与上一包进行比较。如果数据包值不同，则认为接收的数据包是新数据包。发送方：每发送一包新的数据则发送方的数据包值加一。

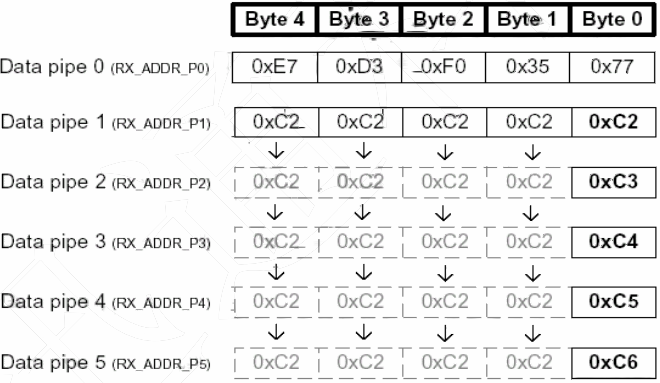


PID的生成



PID的检测

通过SPI接口进行配置CRC校验的长度。接收数据包的附加要求是CRC的校验错误，便不会接收数据包。nRF24L01能够接收六路不同的地址相同频率的数据。每个数据通道都拥有自己的地址，并能够通过寄存器来分别配置。数据通道是通过寄存器配置的，初始状态下只有通道零和数据通道一是开启状态的。一般情况下不会允许不同的通道设置一样的地址。通道0有40位可配置地址。数据通道1~5的地址为：32位共用地址+各自的地址。



通道0-5的地址设置

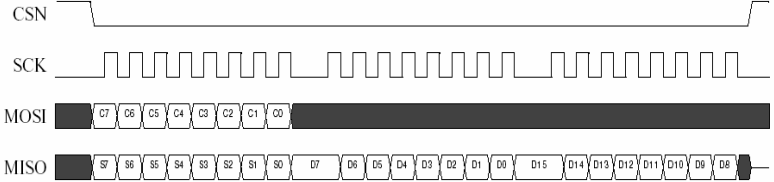
图 3-12 所示的是通道一到五的地址设置方法举例。全部通道可以设置为多达四十位，但是一到五通道的最低位必须不同。

所有寄存器都是通过SPI口进行配置的。SPI 指令格式：命令字+数据字节。

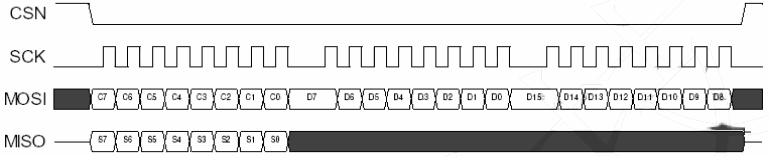
nRF24L01 SPI串行口指令设置

|  |  |  |
| --- | --- | --- |
| 指令名称 | 指令格式 | 操作 |
| R\_REGISTER | 000A AAAA | 读配置寄存器。 AAAAA 指出读操作的寄存器地址 |
| W\_REGISTER | 001A AAAA | 写配置寄存器。 AAAAA 指出写操作的寄存器地址。只有在掉电模式和待机模式下可操作。 |
| R\_RX\_PAYLOAD | 0110 0001 | 读 RX 有效数据： 1-32 字节。读操作全部从字节 0 开始。当读 RX有效数据完成后， FIFO 寄存器中有效数据被清除。应用于接收模式下。 |
| W\_RX\_PAYLOAD | 1010 0000 | 写 TX 有效数据： 1-32 字节。写操作从字节 0 开始。应用于发射模式下。 |
| FLUSH\_TX | 1110 0001 | 清除 TX FIFO 寄存器，应用于发射模式下。 |
| FLUSH\_RX | 1110 0010 | 清除 RX FIFO 寄存器，应用于接收模式下。在传输应答信号过程中不应执行此指令。也就是说，若传输应答信号过程中执行此指令的话将使得应答信号不能被完整的传输。 |
| REUSE\_TX\_PL | 1110 0011 | 重新使用上一包有效数据。当 CE 为高过程中，数据包被不断的重新发射。在发射数据包过程中必须禁止数据包重利用功能。 |
| NOP | 1111 1111 | 空操作。可以用来读状态寄存器。 |

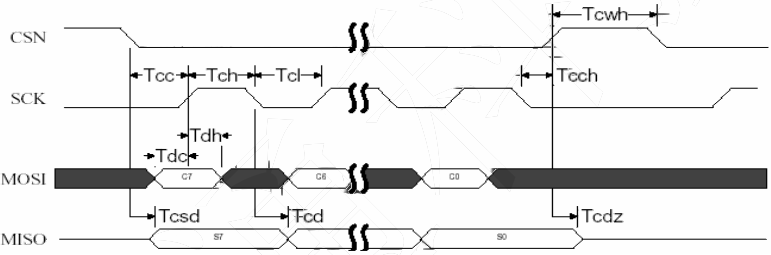
当访问多字节寄存器时首先要读/写的是最低字节的高位。



SPI读操作

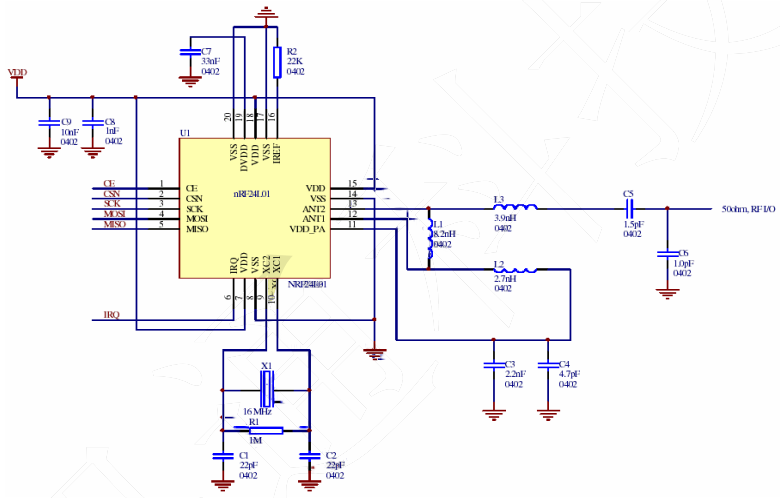


SPI写操作



SPI NOP操作时序图

* + 1. nRF24L01 电路

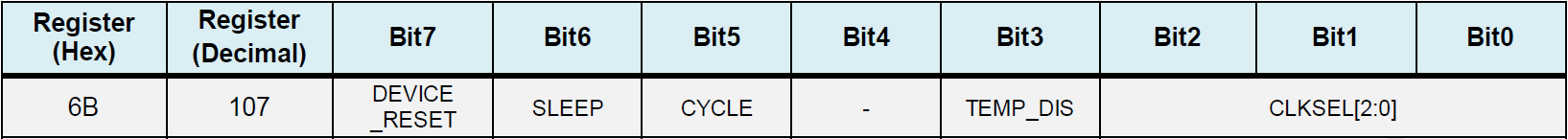


nRF24L01单端匹配网络：晶振、偏置电阻、去藕电容部分电路

* 1. MPU6050简介

MPU6050是整合性六轴运动处理组件。

AD0=0地址=0X68；AD0=1地址=0X69；



电源管理寄存器1（0X6B）

DEVICE\_RESE=1，复位MPU6050，复位完成后，自动清零。SLEEP=1，进入睡眠模式；SLEEP=0，正常工作模式。如下表所示：

系统时钟源选择表

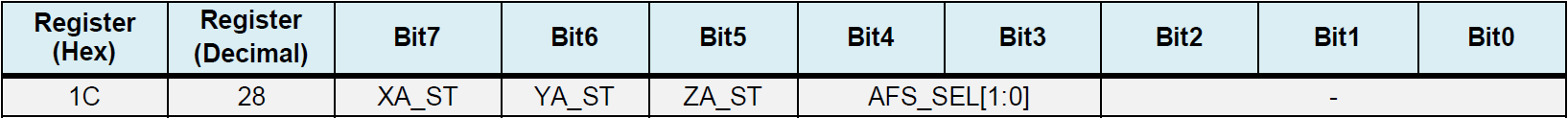
|  |  |
| --- | --- |
| CLKSEL[2:0] | 时钟源 |
| 000 | 内部8M RC晶振 |
| 001 | PLL，使用X轴陀螺作为参考 |
| 010 | PLL，使用Y轴陀螺作为参考 |
| 011 | PLL，使用Z轴陀螺作为参考 |
| 100 | PLL，使用外部32.768Khz作为参考 |
| 101 | PLL，使用外部19.2Mhz作为参考 |
| 110 | 保留 |
| 111 | 关闭时钟，保持时序产生电路复位状态 |

陀螺仪配置寄存器，FS\_SEL[1:0]用于设置陀螺仪的满量程范围。



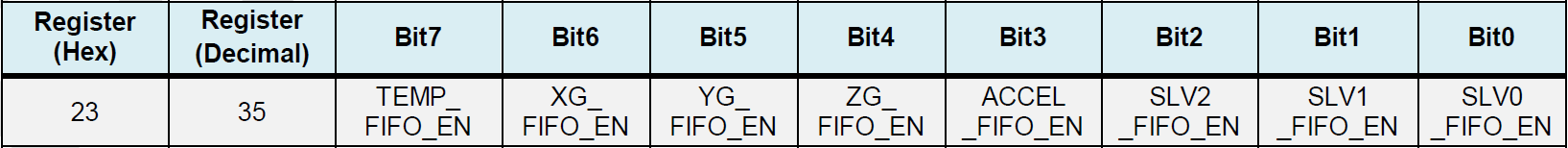
陀螺仪配置寄存器（0X1B）

加速度传感器配置寄存器，AFS\_SEL[1:0]用于设置加速度传感器的满量程范围。



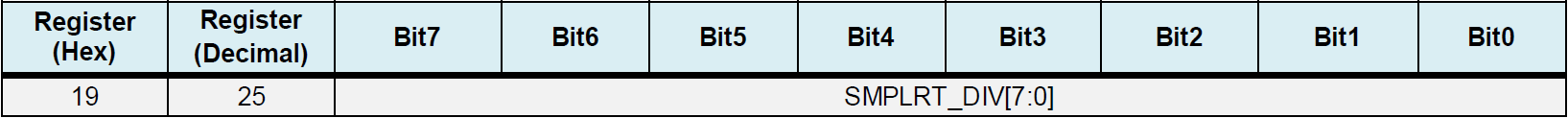
加速度传感器配置寄存器（0X1C）

FIFO使能寄存器用于控制FIFO使能，在简单读取传感器数据的时候，可以不用FIFO，设置对应位为：0，即可禁止FIFO，设置为1，则使能FIFO。注意：加速度传感器的3个轴，全由1个位（ACCEL\_FIFO\_EN）控制，只要该位置1，则加速度传感器的三个通道都开启FIFO了。

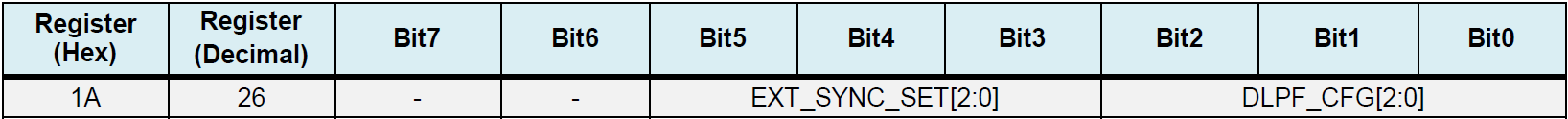


FIFO使能寄存器（0X23）

陀螺仪采样频率等于陀螺仪输出频率除以SMPLRT\_DIV寄存器的值加一，DLPF滤波频率一般设置为采样率的一半。采样率，我们假定设置为50Hz，那么：SMPLRT\_DIV=1000/50-1=19。



陀螺仪采样率分频寄存器（0X19）



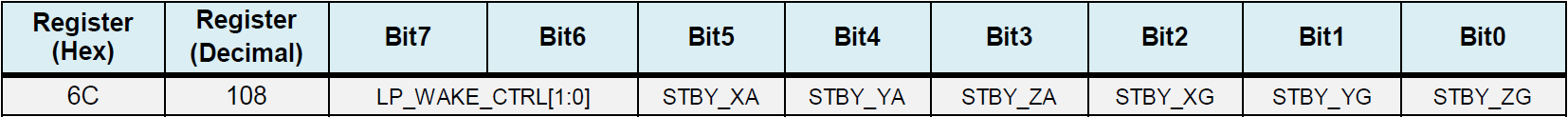
配置寄存器（0X1A）

数字低通滤波器（DLPF）的设置位，如下表：

数字低通滤波器（DLPF）的设置位

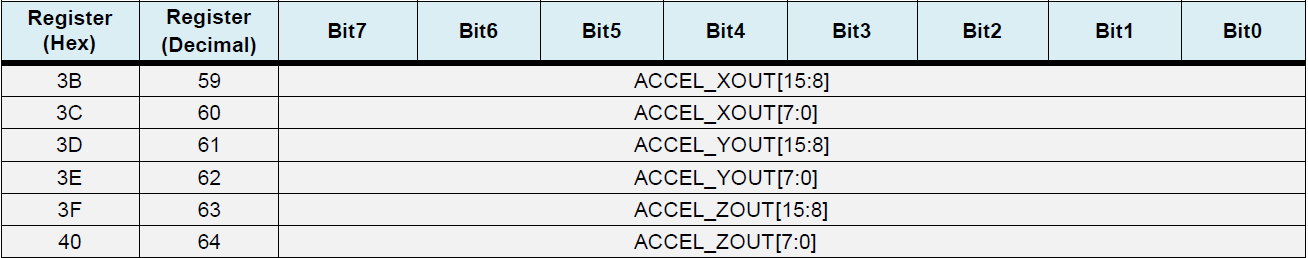
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DLPF\_CFG[2:0] | 加速度传感器Fs=1Khz | | 角速度传感器（陀螺仪） | | |
| 带宽(Hz) | 延迟（ms） | 带宽(Hz) | 延迟（ms） | Fs(Khz) |
| 000 | 260 | 0 | 256 | 0.98 | 8 |
| 001 | 184 | 2.0 | 188 | 1.9 | 1 |
| 010 | 94 | 3.0 | 98 | 2.8 | 1 |
| 011 | 44 | 4.9 | 42 | 4.8 | 1 |
| 100 | 21 | 8.5 | 20 | 8.3 | 1 |
| 101 | 10 | 13.8 | 10 | 13.4 | 1 |
| 110 | 5 | 19.0 | 5 | 18.6 | 1 |
| 111 | 保留 | | 保留 | | 8 |

该寄存器的LP\_WAKE\_CTRL用于控制低功耗时的唤醒。剩下的分别控制加速度和陀螺仪是否进入待机模式。



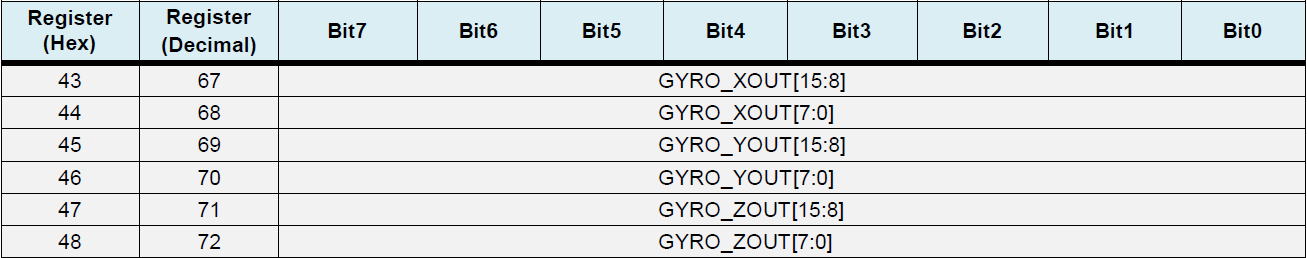
电源管理寄存器2（0X6C）

加速度传感器数据输出寄存器输出X/Y/Z三个轴的加速度传感器值，高字节在前，低字节在后。



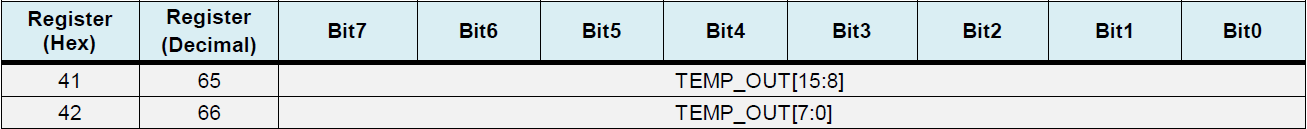
加速度传感器数据输出寄存器（0X3B~0X40）

陀螺仪数据输出寄存器输出X/Y/Z三个轴的陀螺仪传感器数据，高字节在前，低字节在后。



陀螺仪数据输出寄存器（0X43~0X48）

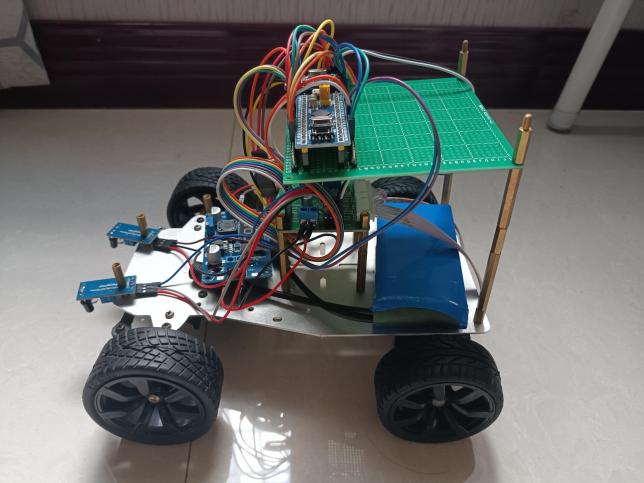
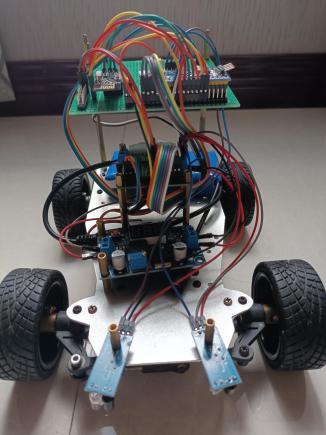
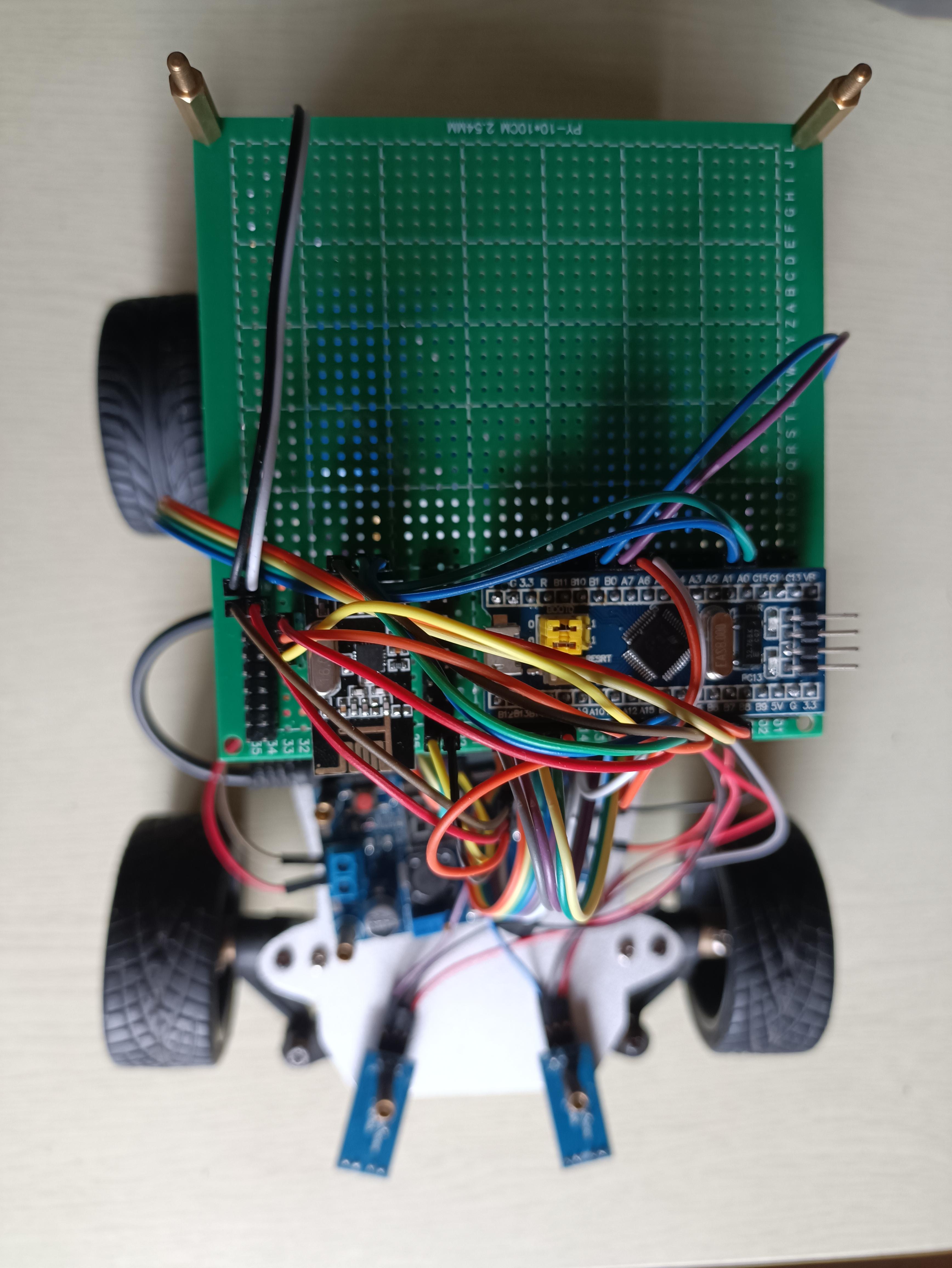
温度换算公式为：Temperature = 36.53 + regval/340单位为℃，regval为从0X41和0X42读到的温度传感器值。



温度传感器数据输出寄存器（0X41~0X42）

通过前面配置，便可以读取MPU6050的温度、加速度、陀螺仪的数据参数，但是我们在实际的使用过程中希望得到的数据是姿态数据欧拉角：yaw、roll和pitch。利用原始数据，进行姿态融合解算比较复杂，不易掌握，且会浪费大量单片机的计算资源。但是MPU6050本身拥有DMP数字运动处理器，并且InvenSense提供了对应的驱动库，官方DMP驱动库移植，主要是实现这4个函数：i2c\_write、i2c\_read、delay\_ms和get\_ms。

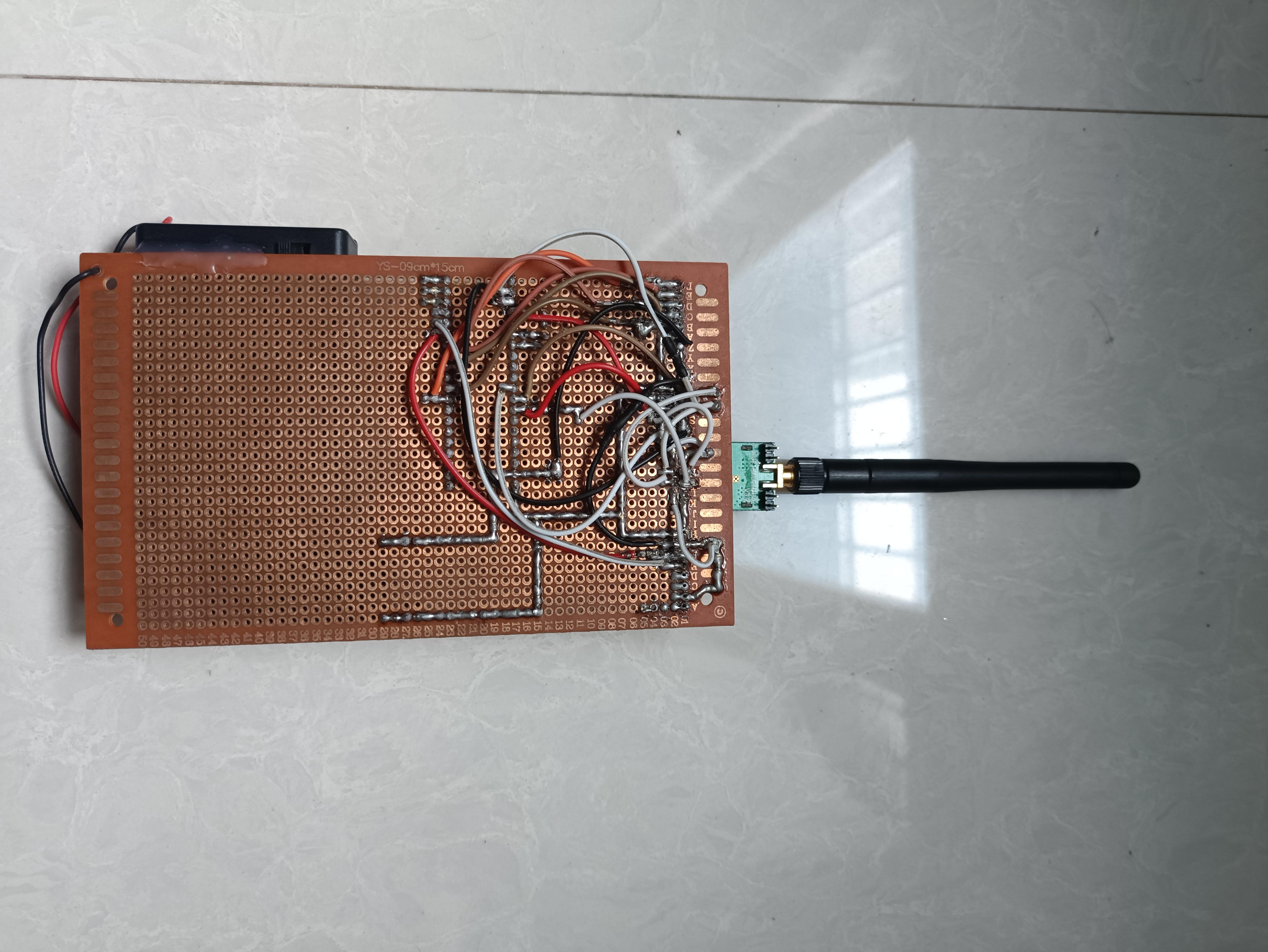
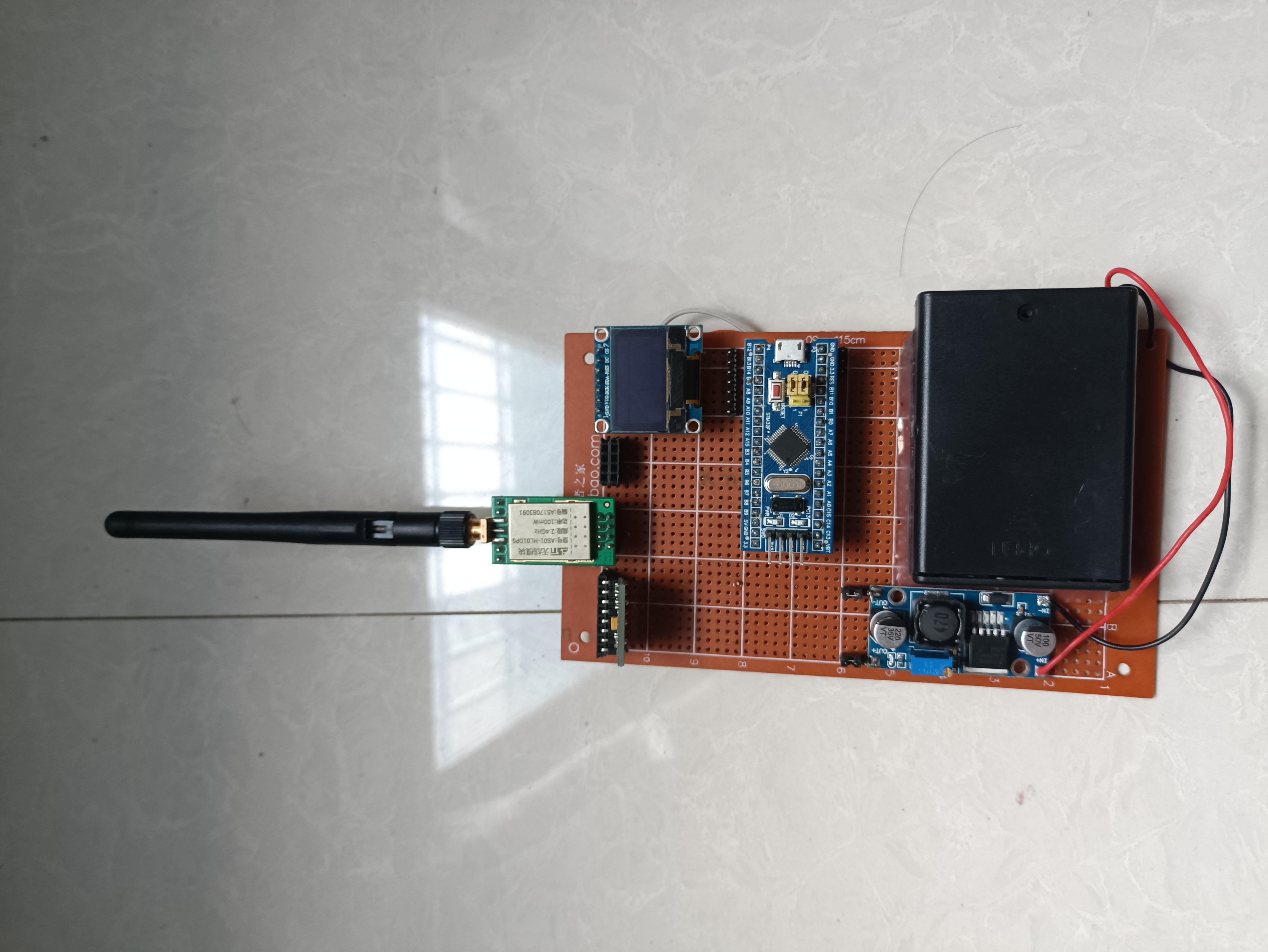
* 1. 本章小结
     1. 小车实物展示和小结



实物小车三视图

小车底盘由车轮，电机，舵机，红外传感器，稳压模块，锂电池模块组成。电机驱动模块放在了中部，由4根铜柱固定。顶部放置由stm32最小系统板和2.4G通信模块构成。上部预留了较大空间，方便后续模块加入。

* + 1. 手柄实物展示和小结



手柄实物图

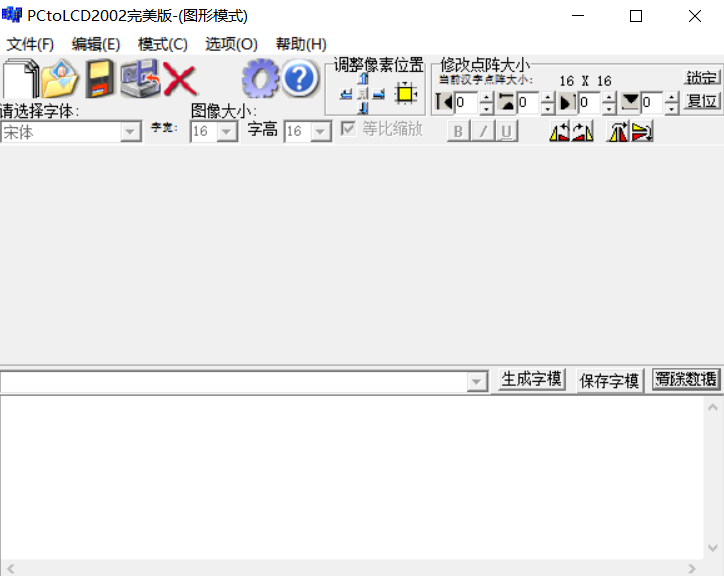
手柄部分由oled屏，MPU6050模块，最小系统板，2.4G通信模块，锂电池和降压模块组成。电池盒拥有一个电源开关，可以控制手柄是否工作。手柄部分的该硬件设计只是一个功能演示和验证。如果要产品化，应该自己设计pcb去微型化手柄，使之能够装载到手套上。

1. 软件设计
   1. 软件工具
      1. 软件工具MDK5 简介

MDK源自德国的KEIL公司，是RealView MDK的简称。MDK5由两个部分组成：MDK内核和软件包。其中，软件包可以独立于工具链进行新芯片支持和中间库的升级。本设计主要依靠这个软件进行代码编写，软件支撑debug调试，只需要连接下载器stlink便可以进行程序的下载和debug调试。

* + 1. Oled显示屏专用取模工具

该软件可生成中英文数字混合的字符串的字模数据。各种旋转，翻转文字功能。任意调整输出点阵大小，并任意调整字符在点阵中的位置。每行输出数据个数可调。支持四种取模方式：逐行，逐列，行列，列行支持阴码，阳码取模。支持纵向取模。输出数制可选16进制或10进制。动态液晶面板彷真，可调节彷真面板象素点大小和颜色。图形模式下可用鼠标作画，左键画图，右键擦图。



软件界面展示

* + 1. CubeMX

在Cube工具还没出来之前，在ST的MCU开发都是用标准固件库，标准库自推出以来受到ST的使用者的推崇，现在很多公司也都在使用。但是ST官方在2013年后就没有更新版本，ST官方也全力推HAL（Hardware Abstraction Layer）库。它的存在是为了确保 STM32 系列最大的移植性。CubeMX软件可以直接根据开发人员所需的功能勾选，以HAL库文件基础自动产生代码。开发者就可以轻松应用每一个外设。在 ST 官方的声明中，HAL 库是大势所趋，在 ST 公司最新开发的部分芯片中，只有 HAL 库而没有标准库，从这点便可以说明，以后的战略目标是逐渐的转向HAL 库 。相对于标准库来说，在使用 CubeMX 生成代码后，工程项目和初始化代码已经完成。cubeMX直接生产的工程支持keil。可以在生成代码后，直接在keil的编译环境中编写代码。

* + 1. Git

Git是用于Linux内核开发的版本控制工具。但是在PC端也可以用，本项目的代码管理，采用 Git 来管理项目开发，进行版本管理。它的操作十分方便，主需要简单操作指令就可以完成代码的上传，版本的回退。并且结合gitee这个代码管理网站，可以实现云端管理代码。可以便捷的把代码上传给gitee，或者把gitee上保存的代码clone到本地。

* + 1. 串口调试工具

只需要一个串口转USB连接器，就可以完成单片机和电脑端的数据交换。而在编程的时候，我们需要一些打印信息来帮助我们来定位系统的状态和bug。而电脑端需要的软件支持就是串口调试工具。

* 1. 主程序APP部分代码
     1. 小车部分app

#include "FreeRTOS.h"

#include "task.h"

#include "cmsis\_os.h"

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "App\_Main.h"

#include "bsp\_led.h"

#include "bsp\_infra\_red.h"

#include "bsp\_debug.h"

#include "bsp\_nrf24l01.h"

#include "bsp\_SG90.h"

#include "bsp\_encoder.h"

#include "bsp\_motor.h"

void Task01(void \* argument);

osThreadId Task01\_TaskHandle;

const osThreadAttr\_t osID\_Task01 = {

.name = "osID\_Task01",

.stack\_size = 128 \* 4,

.priority = (osPriority\_t) osPriorityNormal,

};

void Task02(void \* argument);

osThreadId Task02\_TaskHandle;

const osThreadAttr\_t osID\_Task02 = {

.name = "osID\_Task02",

.stack\_size = 128 \* 4,

.priority = (osPriority\_t) osPriorityHigh7,

};

void Task03(void \* argument);

osThreadId Task03\_TaskHandle;

const osThreadAttr\_t osID\_Task03 = {

.name = "osID\_Task03",

.stack\_size = 128 \* 4,

.priority = (osPriority\_t) osPriorityNormal7,

};

void Task04(void \* argument);

osThreadId Task04\_TaskHandle;

const osThreadAttr\_t osID\_Task04 = {

.name = "osID\_Task04",

.stack\_size = 128 \* 4,

.priority = (osPriority\_t) osPriorityNormal6,

};

void Task05(void \* argument);

osThreadId Task05\_TaskHandle;

const osThreadAttr\_t osID\_Task05 = {

.name = "osID\_Task05",

.stack\_size = 128 \* 4,

.priority = (osPriority\_t) osPriorityHigh6,

};

void APP\_Main(void)

{

Task01\_TaskHandle = osThreadNew(Task01, NULL, &osID\_Task01);

Task02\_TaskHandle = osThreadNew(Task02, NULL, &osID\_Task02);

Task03\_TaskHandle = osThreadNew(Task03, NULL, &osID\_Task03);

Task04\_TaskHandle = osThreadNew(Task04, NULL, &osID\_Task04);

Task05\_TaskHandle = osThreadNew(Task05, NULL, &osID\_Task05);

}

void Task01(void \* argument)

{

steering\_engine\_PWM(950);

for(;;)

{

led\_twinkle(100);

}

}

int car\_mode = 0;

void Task02(void \* argument)

{

EventBits\_t r\_event;

infra\_red\_init();

for(;;)

{

check\_app();

while (1)

{

r\_event = xEventGroupWaitBits(infra\_red\_Event\_Handle,

LeftCheck\_EVENT|RightCheck\_EVENT,

pdTRUE,

pdFALSE,

portMAX\_DELAY);

if(car\_mode==0)

{

check\_app();

}

else

{

osDelay(1000);

}

}

}

}

void Task03(void \* argument)

{

char \*maintain\_addr;

int maintain\_value;

char car\_value[8]="";

u8 i;

u8 rx\_buf[33]="";

while(NRF24L01\_Check())

{

printf("Error \r\n");

osDelay(1000);

}

printf("Success \r\n");

NRF24L01\_Init();

for(;;)

{

if(NRF24L01\_RxPacket(rx\_buf)==0)

{

if(car\_mode==0)

{

if(strstr((const char \*)(rx\_buf), "AT+CMD=HANDLE") != NULL)

{NRF24L01\_TxPacket("HANDLE MODE");car\_mode=1; printf("%s\r\n",rx\_buf);}

}

else

{

if(strstr((const char \*)(rx\_buf), "AT+CMD=SMART") != NULL)

{NRF24L01\_TxPacket("SMART MODE"); car\_mode=0; printf("%s\r\n",rx\_buf);}

}

if(car\_mode==1)

{

if(strstr((const char \*)(rx\_buf), "AT+CMD=STOP") != NULL)

{

printf("%s\r\n",rx\_buf);

car\_stop();

}

if(strstr((const char \*)(rx\_buf), "AT+CMD=B") != NULL)

{

printf("%s\r\n",rx\_buf);

maintain\_addr=strstr((const char \*)(rx\_buf), "=");

maintain\_addr+=3;

for(i=0;i<8;i++)

{

car\_value[i]=\*maintain\_addr;

maintain\_addr+=1;

}

printf("car\_value:%d \r\n",atoi(car\_value));

back\_run(atoi(car\_value)-20,100);

}

if(strstr((const char \*)(rx\_buf), "AT+CMD=F") != NULL)

{

printf("%s\r\n",rx\_buf);

maintain\_addr=strstr((const char \*)(rx\_buf), "=");

maintain\_addr+=3;

for(i=0;i<8;i++)

{

car\_value[i]=\*maintain\_addr;

maintain\_addr+=1;

}

printf("car\_value:%d \r\n",atoi(car\_value));

forward\_run(-atoi(car\_value)-20,100);

}

if(strstr((const char \*)(rx\_buf), "AT+CMD=L") != NULL)

{

printf("%s\r\n",rx\_buf);

maintain\_addr=strstr((const char \*)(rx\_buf), "=");

maintain\_addr+=3;

for(i=0;i<8;i++)

{

car\_value[i]=\*maintain\_addr;

maintain\_addr+=1;

}

printf("car\_value:%d \r\n",atoi(car\_value));

steering\_engine\_PWM((1600-950)\*atoi(car\_value)/60+950);

}

if(strstr((const char \*)(rx\_buf), "AT+CMD=R") != NULL)

{

printf("%s\r\n",rx\_buf);

maintain\_addr=strstr((const char \*)(rx\_buf), "=");

maintain\_addr+=3;

for(i=0;i<8;i++)

{

car\_value[i]=\*maintain\_addr;

maintain\_addr+=1;

}

printf("car\_value:%d \r\n",atoi(car\_value));

steering\_engine\_PWM(950-(950-600)\*(-atoi(car\_value))/60);

}

}

}

osDelay(100);

}

}

void Task04(void \* argument)

{

Encoder\_Start();

for(;;)

{

osDelay(1000);

}

}

void Task05(void \* argument)

{

uint32\_t tick;

for(;;)

{

tick = osKernelGetTickCount();

tick += 10U;

osDelayUntil(tick);

PID\_REFLASH(0);

PID\_REFLASH(1);

}

}

* + 1. 手柄部分app

#include "FreeRTOS.h"

#include "task.h"

#include "cmsis\_os.h"

#include <stdio.h>

#include <string.h>

#include "App\_Main.h"

#include "bsp\_led.h"

#include "bsp\_debug.h"

#include "bsp\_oled096\_spi.h"

#include "bsp\_oled096\_spi\_BMP.h"

#include "bsp\_mpu6050.h"

#include "inv\_mpu.h"

#include "inv\_mpu\_dmp\_motion\_driver.h"

#include "bsp\_nrf24l01.h"

void Task01(void \* argument);

osThreadId Task01\_TaskHandle;

const osThreadAttr\_t osID\_Task01 = {

.name = "osID\_Task01",

.stack\_size = 128 \* 4,

.priority = (osPriority\_t) osPriorityNormal,

};

void Task02(void \* argument);

osThreadId Task02\_TaskHandle;

const osThreadAttr\_t osID\_Task02 = {

.name = "osID\_Task02",

.stack\_size = 128 \* 8,

.priority = (osPriority\_t) osPriorityAboveNormal,

};

void Task03(void \* argument);

osThreadId Task03\_TaskHandle;

const osThreadAttr\_t osID\_Task03 = {

.name = "osID\_Task03",

.stack\_size = 128 \* 8,

.priority = (osPriority\_t) osPriorityNormal6,

};

void Task04(void \* argument);

osThreadId Task04\_TaskHandle;

const osThreadAttr\_t osID\_Task04 = {

.name = "osID\_Task04",

.stack\_size = 128 \* 8,

.priority = (osPriority\_t) osPriorityNormal7,

};

void APP\_Main(void) {

Task01\_TaskHandle = osThreadNew(Task01, NULL, &osID\_Task01);

Task02\_TaskHandle = osThreadNew(Task02, NULL, &osID\_Task02);

Task03\_TaskHandle = osThreadNew(Task03, NULL, &osID\_Task03);

Task04\_TaskHandle = osThreadNew(Task04, NULL, &osID\_Task04);

}

void Task01(void \* argument)

{

printf("任务1启动成功。\r\n");

for(;;)

{

led\_twinkle(100);

}

}

void Task02(void \* argument)

{

uint8\_t t=' ';

char str[32]="";

OLED\_Init();

OLED\_Clear();

OLED\_DrawBMP(0,2,128,8,BMP1);

osDelay(1000);

OLED\_Clear();

OLED\_ShowString(0,0,"temp: ",12);

OLED\_ShowString(0,1,"pitch: ",12);

OLED\_ShowString(0,2,"roll: ",12);

OLED\_ShowString(0,3,"yaw: ",12);

OLED\_ShowString(0,4,"r: ",12);

OLED\_ShowString(0,5,"mode: ",12);

for(;;)

{

sprintf(str,"%0.2f ",temp);

OLED\_ShowString(40,0,(u8\*)str,12);

sprintf(str,"%0.1f ",pitch);

OLED\_ShowString(48,1,(u8\*)str,12);

sprintf(str,"%0.1f ",roll);

OLED\_ShowString(40,2,(u8\*)str,12);

sprintf(str,"%0.1f ",yaw);

OLED\_ShowString(32,3,(u8\*)str,12);

snprintf(str,10,"%s ",rf\_rxbuf);

OLED\_ShowString(16,4,(u8\*)str,12);

osDelay(100);

}

}

int NRF24L01\_Init\_sucflag=0;

void Task03(void \* argument)

{

MPU\_Init();

char str[32]="";

while(mpu\_dmp\_init())

{

printf ("MPU6050 Error\r\n");

delay\_ms(1000);

}

printf ("MPU6050 OK \r\n");

int error = 0;

for(;;)

{

error = mpu\_dmp\_get\_data(&pitch,&roll,&yaw);

if(error==0)

{

temp=MPU\_Get\_Temperature();

}

else

{

mpu\_dmp\_get\_data(&pitch,&roll,&yaw);

}

if(NRF24L01\_Init\_sucflag==1)

error=NRF24L01\_RxPacket(rf\_rxbuf);

if(strstr((const char \*)(rf\_rxbuf), "HANDLE MODE") != NULL)

{

snprintf(str,11,"HANDLE MODE ");

OLED\_ShowString(40,5,(u8\*)str,12);

}

if(strstr((const char \*)(rf\_rxbuf), "SMART MODE") != NULL)

{

snprintf(str,11,"SMART MODE ");

OLED\_ShowString(40,5,(u8\*)str,12);

}

osDelay(100);

}

}

void Task04(void \* argument)

{

int mode = 1;

while(NRF24L01\_Check())

{

printf("NRF24L01\_Check erro \r\n");

osDelay(1000);

}

printf("NRF24L01\_Check suc \r\n");

NRF24L01\_Init();

NRF24L01\_Init\_sucflag=1;

char rf\_txbuf[TX\_PLOAD\_WIDTH+1];

int error = 0;

for(;;)

{

if(-20<pitch && pitch<20 && roll<20 && roll>-20)

{

snprintf(rf\_txbuf,32,"AT+CMD=STOP");

NRF24L01\_TxPacket((u8\*)rf\_txbuf);

printf("%s \r\n",rf\_txbuf);

osDelay(100);

}

if(pitch>20)

{

snprintf(rf\_txbuf,32,"AT+CMD=B,%0.1f",pitch);

NRF24L01\_TxPacket((u8\*)rf\_txbuf);

printf("%s \r\n",rf\_txbuf);

osDelay(100);

}

if(pitch<-20)

{

snprintf(rf\_txbuf,32,"AT+CMD=F,%0.1f",pitch);

NRF24L01\_TxPacket((u8\*)rf\_txbuf);

printf("%s \r\n",rf\_txbuf);

osDelay(100);

}

if(roll>20)

{

snprintf(rf\_txbuf,32,"AT+CMD=L,%0.1f",roll);

NRF24L01\_TxPacket((u8\*)rf\_txbuf);

printf("%s \r\n",rf\_txbuf);

osDelay(100);

}

if(roll<-20)

{

snprintf(rf\_txbuf,32,"AT+CMD=R,%0.1f",roll);

NRF24L01\_TxPacket((u8\*)rf\_txbuf);

printf("%s \r\n",rf\_txbuf);

osDelay(100);

}

if(yaw>40 && mode == 1)

{

mode = 0;

snprintf(rf\_txbuf,32,"AT+CMD=HANDLE");

NRF24L01\_TxPacket((u8\*)rf\_txbuf);

printf("%s \r\n",rf\_txbuf);

osDelay(100);

}

if(yaw<-40 && mode == 0)

{

mode = 1;

snprintf(rf\_txbuf,32,"AT+CMD=SMART");

NRF24L01\_TxPacket((u8\*)rf\_txbuf);

printf("%s \r\n",rf\_txbuf);

osDelay(100);

}

osDelay(100);

}

}

* 1. 主程序BSP部分代码
     1. 小车部分bsp

#include "bsp\_motor.h"

#include "bsp\_SG90.h"

#include "bsp\_encoder.h"

int base\_speed = 1000;

#define left\_fix\_speed 0

#define right\_fix\_speed 0

#define turn\_fix\_speed 60

void forward\_run(int TargetVelocity,int TargetCircle)

{

F\_RUN;

steering\_engine\_PWM(950);

PID\_SET\_SPEED(TargetVelocity,TargetCircle);

left\_wheel\_speed(base\_speed+left\_fix\_speed);

right\_wheel\_speed(base\_speed+right\_fix\_speed);

}

void back\_run(int TargetVelocity,int TargetCircle)

{

B\_RUN;

steering\_engine\_PWM(950);

PID\_SET\_SPEED(TargetVelocity,TargetCircle);

left\_wheel\_speed(base\_speed+left\_fix\_speed);

right\_wheel\_speed(base\_speed+right\_fix\_speed);

}

void left\_run(int TargetVelocity,int TargetCircle)

{

LB\_RUN;

RF\_RUN;

steering\_engine\_PWM(600);

PID\_SET\_SPEED(TargetVelocity,TargetCircle);

left\_wheel\_speed(base\_speed+left\_fix\_speed);

right\_wheel\_speed(base\_speed+right\_fix\_speed);

}

void right\_run(int TargetVelocity,int TargetCircle)

{

LF\_RUN;

RB\_RUN;

steering\_engine\_PWM(1600);

PID\_SET\_SPEED(TargetVelocity,TargetCircle);

left\_wheel\_speed(base\_speed+left\_fix\_speed);

right\_wheel\_speed(base\_speed+right\_fix\_speed);

}

void car\_stop()

{

CAR\_STOP;

steering\_engine\_PWM(950);

PID\_SET\_SPEED(0,0);

left\_wheel\_speed(0);

right\_wheel\_speed(0);

}

#include "bsp\_infra\_red.h"

#include <stdio.h>

#include "bsp\_motor.h"

EventGroupHandle\_t infra\_red\_Event\_Handle =NULL;

void infra\_red\_init(void )

{

infra\_red\_Event\_Handle = xEventGroupCreate();

if(NULL != infra\_red\_Event\_Handle)

printf("infra\_red\_Event\_Handle 事件创建成功!\r\n");

else

printf("infra\_red\_Event\_Handle 事件创建失败!\r\n");

}

void check\_app(void)

{

if(left\_state==state\_on && right\_state== state\_on)

{

car\_stop();

}

if(left\_state==state\_off && right\_state== state\_on)

{

right\_run(10,100);

}

if(left\_state==state\_on && right\_state== state\_off)

{

left\_run(10,100);

}

if(left\_state==state\_off && right\_state== state\_off)

{

forward\_run(10,100);

}

}

BaseType\_t xHigherPriorityTaskWoken = pdFALSE;

void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)

{

if(infra\_red\_Event\_Handle!=NULL)

{

switch(GPIO\_Pin)

{

case infrared\_sensor\_left\_Pin:

xEventGroupSetBitsFromISR(infra\_red\_Event\_Handle,LeftCheck\_EVENT,&xHigherPriorityTaskWoken);

break ;

case infrared\_sensor\_right\_Pin:

xEventGroupSetBitsFromISR(infra\_red\_Event\_Handle,RightCheck\_EVENT,&xHigherPriorityTaskWoken);

break ;

}

}

}

#include "bsp\_nrf24l01.h"

#include "spi.h"

static u8 rf\_read\_modeflag=0;

u8 rf\_len;

#ifdef RX\_PLOAD\_WIDTH

u8 rf\_rxbuf[RX\_PLOAD\_WIDTH];

#else

u8 rf\_rxbuf[100];

#endif

#ifdef TX\_PLOAD\_WIDTH

char rf\_txbuf[TX\_PLOAD\_WIDTH];

#else

u8 rf\_rxbuf[100];

#endif

u8 rf\_flag;

const char TX\_ADDRESS[TX\_ADR\_WIDTH]={0x19,0x95,0x03,0x26,0x13};

const char RX\_ADDRESS[RX\_ADR\_WIDTH]={0x19,0x95,0x03,0x26,0x13};

void NRF24L01\_Init(void)

{

NRF24L01\_CE(0);

NRF24L01\_Write\_Buf(NRF\_WRITE\_REG+RX\_ADDR\_P0,(u8\*)RX\_ADDRESS,RX\_ADR\_WIDTH);

NRF24L01\_Write\_Buf(NRF\_WRITE\_REG+TX\_ADDR,(u8\*)TX\_ADDRESS,TX\_ADR\_WIDTH);

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+EN\_AA,0x01);

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+EN\_RXADDR,0x01);

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+SETUP\_RETR,0x1a);

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+SETUP\_AW,0x03);

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+RF\_SETUP,0x0f);

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+RF\_CH,40);

#ifdef RX\_PLOAD\_WIDTH

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+RX\_PW\_P0,RX\_PLOAD\_WIDTH);

#else

{ NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+DYNPD,0X3F);}

#endif

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+STATUS,0xff);

NRF24L01\_RX\_Mode();

NRF24L01\_CE (1);

}

u8 NRF24L01\_Check(void)

{

u8 buf[5]={0XA5,0XA5,0XA5,0XA5,0XA5};

u8 i;

NRF24L01\_CE(0);

NRF24L01\_CSN(1);

NRF24L01\_Write\_Buf(NRF\_WRITE\_REG+TX\_ADDR,buf,5);

NRF24L01\_Read\_Buf(TX\_ADDR,buf,5);

for(i=0;i<5;i++)if(buf[i]!=0XA5)break;

if(i!=5)return 1;

return 0;

}

uint8\_t SPI1\_ReadWriteByte(uint8\_t TxData)

{

uint8\_t Rxdata;

HAL\_SPI\_TransmitReceive(&hspi1, &TxData, &Rxdata, 1, 1000);

return Rxdata;

}

u8 NRF24L01\_Write\_Reg(u8 reg,u8 value)

{

u8 status;

NRF24L01\_CSN(0);

status =SPI1\_ReadWriteByte(reg);

SPI1\_ReadWriteByte(value);

NRF24L01\_CSN(1);

return(status);

}

u8 NRF24L01\_Read\_Reg(u8 reg)

{

u8 reg\_val;

NRF24L01\_CSN(0);

SPI1\_ReadWriteByte(reg);

reg\_val=SPI1\_ReadWriteByte(0XFF);

NRF24L01\_CSN(1);

return(reg\_val);

}

u8 NRF24L01\_Read\_Buf(u8 reg,u8 \*pBuf,u8 len)

{

u8 status,u8\_ctr;

NRF24L01\_CSN(0);

status=SPI1\_ReadWriteByte(reg);

for(u8\_ctr=0;u8\_ctr<len;u8\_ctr++)pBuf[u8\_ctr]=SPI1\_ReadWriteByte(0XFF);

NRF24L01\_CSN(1);

return status;

}

u8 NRF24L01\_Write\_Buf(u8 reg, u8 \*pBuf, u8 len)

{

u8 status,u8\_ctr;

NRF24L01\_CSN (0);

status = SPI1\_ReadWriteByte(reg);

for(u8\_ctr=0; u8\_ctr<len; u8\_ctr++)SPI1\_ReadWriteByte(\*pBuf++);

NRF24L01\_CSN (1);

return status;

}

#ifdef RX\_PLOAD\_WIDTH

u8 NRF24L01\_TxPacket(u8 \*txbuf)

{

u8 sta;

if(rf\_read\_modeflag == 1 )NRF24L01\_TX\_Mode();

NRF24L01\_CE(0);

NRF24L01\_Write\_Buf(WR\_TX\_PLOAD,txbuf,TX\_PLOAD\_WIDTH);

NRF24L01\_CE(1);

while(NRF24L01\_IRQ!=0);

sta=NRF24L01\_Read\_Reg(STATUS);

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+STATUS,sta);

if(sta&MAX\_TX)

{

NRF24L01\_Write\_Reg(FLUSH\_TX,0xff);

return MAX\_TX;

}

if(sta&TX\_OK)

{

NRF24L01\_RX\_Mode();

return TX\_OK;

}

return 0xff;

}

#else

u8 NRF24L01\_TxPacket(u8 \*txbuf,u8 TX\_PLOAD\_WIDTH)

{

u8 sta;

if(rf\_read\_modeflag == 1 )NRF24L01\_TX\_Mode();

NRF24L01\_CE(0);

NRF24L01\_Write\_Buf(WR\_TX\_PLOAD,txbuf,TX\_PLOAD\_WIDTH);

NRF24L01\_CE(1);

while(NRF24L01\_IRQ!=0);

sta=NRF24L01\_Read\_Reg(STATUS);

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+STATUS,sta);

if(sta&MAX\_TX)

{

NRF24L01\_Write\_Reg(FLUSH\_TX,0xff);

return MAX\_TX;

}

if(sta&TX\_OK)

{

NRF24L01\_RX\_Mode();

return TX\_OK;

}

return 0xff;

}

#endif

u8 NRF24L01\_RxPacket(u8 \*rxbuf)

{

u8 sta;

if(rf\_read\_modeflag == 0 )NRF24L01\_RX\_Mode();

sta=NRF24L01\_Read\_Reg(STATUS);

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+STATUS,sta);

if(sta&RX\_OK)

{

#ifdef RX\_PLOAD\_WIDTH

NRF24L01\_Read\_Buf(RD\_RX\_PLOAD,rxbuf,RX\_PLOAD\_WIDTH);

#else

rf\_len = NRF24L01\_Read\_Reg(R\_RX\_PL\_WID);

NRF24L01\_Read\_Buf(RD\_RX\_PLOAD, rf\_rxbuf, rf\_len);

#endif

NRF24L01\_Write\_Reg(FLUSH\_RX,0xff);

return 0;

}

return 1;

}

void NRF24L01\_RX\_Mode(void)

{

NRF24L01\_CE(0);

NRF24L01\_Write\_Reg(FLUSH\_RX,0xff);

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+CONFIG, 0x0f);

NRF24L01\_CE (1);

rf\_read\_modeflag=1;

}

void NRF24L01\_TX\_Mode(void)

{

NRF24L01\_CE(0);

NRF24L01\_Write\_Buf(NRF\_WRITE\_REG+TX\_ADDR,(u8\*)TX\_ADDRESS,TX\_ADR\_WIDTH);

NRF24L01\_Write\_Reg(FLUSH\_RX,0xff);

NRF24L01\_Write\_Reg(NRF\_WRITE\_REG+CONFIG,0x0e);

NRF24L01\_CE(1);

rf\_read\_modeflag=0;

}

#include "bsp\_encoder.h"

#include "FreeRTOS.h"

#include "task.h"

#include "cmsis\_os.h"

#include "bsp\_motor.h"

int TargetVelocity=5, TargetCircle=100, CurrentPosition , Encoder, PWM;

float Velcity\_Kp=80, Velcity\_Ki=10, Velcity\_Kd;

float Position\_Kp=120, Position\_Ki=0.1, Position\_Kd=400;

void Encoder\_Start()

{

right\_encoder\_start;

left\_encoder\_start;

}

int Read\_Encoder(int choose)

{

int Encoder\_number;

if(choose == 1)

{

Encoder\_number=get\_left\_encoder;

if(get\_left\_direction == dirt\_back)Encoder\_number=Encoder\_number-0xffff;

set\_left\_encoder(0);

}

if(choose == 0)

{

Encoder\_number=get\_right\_encoder;

if(get\_right\_direction == dirt\_back)Encoder\_number=Encoder\_number-0xffff;

set\_right\_encoder(0);

}

return Encoder\_number;

}

int Velocity\_FeedbackControl(int TargetVelocity, int CurrentVelocity)

{

int Bias;

static int ControlVelocity, Last\_bias;

Bias=TargetVelocity-CurrentVelocity;

ControlVelocity+=Velcity\_Kp\*(Bias-Last\_bias)+Velcity\_Ki\*Bias;

Last\_bias=Bias;

return ControlVelocity;

}

int Position\_FeedbackControl(float Circle, int CurrentPosition)

{

float TargetPosition,Bias, ControlVelocity;

static float Last\_bias, Integral\_Bias;

TargetPosition=Circle\*1560\*1.01;

Bias=TargetPosition-CurrentPosition;

Integral\_Bias+=Bias;

if(Integral\_Bias> 970) Integral\_Bias= 970;

if(Integral\_Bias<-970) Integral\_Bias=-970;

ControlVelocity=Position\_Kp\*Bias+Position\_Ki\*Integral\_Bias+Position\_Kd\*(Bias-Last\_bias);

Last\_bias=Bias;

return ControlVelocity;

}

int PWM\_Restrict(int PWM\_P, int TargetVelocity)

{

if (PWM\_P>+TargetVelocity\*76) PWM\_P=+TargetVelocity\*76;

else if(PWM\_P<-TargetVelocity\*76) PWM\_P=-TargetVelocity\*76;

else PWM\_P=PWM\_P;

return PWM\_P;

}

void PID\_REFLASH(int choose)

{

static int PWM\_P, PWM\_V;

Encoder=Read\_Encoder(choose);

if(Encoder<0)Encoder=-Encoder;

CurrentPosition+=Encoder;

PWM\_P=Position\_FeedbackControl(TargetCircle, CurrentPosition);

PWM\_P=PWM\_Restrict(PWM\_P,TargetVelocity);

PWM\_V=PWM\_P/76;

PWM=Velocity\_FeedbackControl(PWM\_V, Encoder);

if(PWM<0)PWM=-PWM;

if(PWM>7200-1)PWM=7200-1;

base\_speed=PWM;

}

void PID\_SET\_SPEED(int x,int y)

{

TargetVelocity=x;

TargetCircle=y;

}

#include "APP\_delay.h"

#include "tim.h"

void delay\_us(uint16\_t us)

{

uint16\_t differ = 0xffff-5-us;

\_\_HAL\_TIM\_SET\_COUNTER(&htim2,differ);

HAL\_TIM\_Base\_Start(&htim2);

while(differ < 0xffff-5)

{

differ = \_\_HAL\_TIM\_GET\_COUNTER(&htim2);

}

HAL\_TIM\_Base\_Stop(&htim2);

}

* + 1. 手柄部分bsp

#include "bsp\_debug.h"

#define SENDBUFF\_SIZE 10

uint8\_t SendBuff[SENDBUFF\_SIZE];

int fputc(int ch, FILE \*f)

{

HAL\_UART\_Transmit(&huart1, (uint8\_t \*)&ch, 1, 1000);

return (ch);

}

int fgetc(FILE \*f)

{

int ch;

HAL\_UART\_Receive(&huart1, (uint8\_t \*)&ch, 1, 1000);

return (ch);

}

#include "bsp\_led.h"

#include "stdint.h"

#include "cmsis\_os.h"

led\_err\_t led\_twinkle(int time)

{

led\_err\_t err=led\_ok;

if(time<=0)

{

err=led\_time\_err;

}

int i=0;

for(i=0;i <= time ;i++)

{

led\_on;

osDelay(1000);

led\_off;

osDelay(1000);

}

return err;

}

* 1. 本章小结

操作系统的使用可以极大优化程序的编写，cubemx软件，又可以极大简化对于底层硬件初始化的操作。方便程序员花更多的时间专注于应用app上的编写。程序员应该有良好的代码管理习惯，积极运用Git之类的代码管理工具，可以做好代码的版本管理。程序中遇见bug，要学会根据现象分析问题，对比代码，回退代码，屏蔽代码等方式，都可以帮助我们去分析bug出现的原因。并且，像示波器，电压表等测量仪器，也能帮助我们找到问题原因，甚至帮助我们判断是硬件问题还是软件问题。

1. 系统的调试及结论
   1. 调试中遇到的重点与难点和解决方案
      1. MPU6050的DMP调试出现FIFO溢出问题。

源代码：

while(1)

{

delay\_ms(1000);

if(mpu\_dmp\_get\_data(&pitch,&roll,&yaw)==0)

{

temp=MPU\_Read\_Temp();

MPU\_Read\_Accel(&ax,&ay,&az);

MPU\_Read\_Gyro(&gx,&gy,&gz);

mpu6050\_send\_data(ax,ay,az,gx,gy,gz);

}

}

进入debug模式，进行断点分析。发现程序并没有跑进if条件语句成立的部分代码，于是通过断点检测进入条件函数中，发现在运行dmp\_read\_fifo函数代码的时候被返回1了，正确情况下是不会返回的。于是继续在dmp\_read\_fifo函数里面打断点。发现在执行mpu\_read\_fifo\_stream函数的时候，程序出现了异常的返回值。最终，发生异常的语句定位位置如下为如下：if (fifo\_count > (st.hw->max\_fifo >> 1)) { /\* FIFO is 50% full, better check overflow bit. \*/ if (i2c\_read(st.hw->addr, st.reg->int\_status, 1, tmp)) return -1; // printf("0x%02x, ",tmp[0]); if (tmp[0] & BIT\_FIFO\_OVERFLOW) { mpu\_reset\_fifo(); return -2; } }断点显示int\_status的值为0x13,发生了FIFO寄存器的溢出中断。从数据手册中得知，当FIFO的存储容量被使用50%以上便会进入中断检测状态，读取FIFO数据的速度如果太慢，便会导致计数寄存器的数据过多。于是检查函数代码，发现在读取完数据后，距离下次获取数据的延迟时间太长，也就是说，采样的速率不够快，于是我们采用了单独的任务去单独执行采样程序，并将采集到的数据存储为全局变量。

* + 1. 红外检测过慢，导致小车错过检测跑道时机，而跑出跑道。

在实际的测试中，我们发现，小车在转弯的时候，容易跑出跑道。降低速度后，情况虽然能够得到改善，但是，依旧会发生检测不到的情况。于是我们对代码进行分析后发现，检测中断虽然能及时响应检测信号，但是信号的处理确实放在任务中的。如果任务的优先级不够。小车便不能够及时响应红外检测信号，以至于错过检测点，跑出跑道。于是我们上调了检测程序的优先级。问题的到了解决。

* + 1. Pwm过低，转弯的驱动能力不足。

在小车运行过程中，我们发现小车前进和转弯的驱动能力是不一样的。比如，占空比10的PWM波虽然能够驱动小车前进，但是完全无法驱动小车的左转和右转。特别是运用了舵机的左右转向。得出的结论便是，小车在转弯的时候需要更大的驱动力。于是我们改进了程序代码，在小车转弯的时候，给了小车更大的pwm波驱动能力。但是实际测试后，发现虽能解决问题，但是效果不理想。因为小车电源的电池电能是在不停被消耗的。所以，同样的PWM波占空比，会因为马达电源电压的不一样，而拥有不同的驱动能力。研究分析问题后，我们发现静态的pwm设置，无法解决动态的电压变化情况。于是在查阅了大量资料后，我们引入pid算法，进行对小车电机的动态pwm设置。根据编码器对小车速度的信息采集，将其带入到pid算法中去。便能够实现pwm波的动态设置。实现的效果为，小车会维持一个匀速前进，当小车负重增加，或者电源供能下降的时候，程序会自动增加电机的pwm波的占空输出。以实现速度的动态平衡。

* + 1. NRF24L01初始化失败

在调试2.4G通信的时候，之前测试通过的代码程序突然，无法完成初始化验证。于是我们对代码进行分析，发现程序一直无法完成模块的初始化程序。于是首先需要排除硬件故障，因为连线是重新焊接过的。所以我们使用万能表依次对每个引脚的电路进行通断检测，发现每条电路都是导通的。说明硬件连接没有问题。再检测一次连线引脚是否连接正确，发现没有问题。硬件故障排除。说明问题出在代码上。于是我们再次将测试手柄程序上NRF24L01的初始化部分代码，看是否出现相同的问题。发现手柄的程序初始化代码是可以通过的。然后我们将小车上的NRF24L01替换在了手柄上。发现同样能够正常初始化。说明，NRF24L01的硬件本身不存在问题。于是我们用git版本管理对小车部分代码进行了版本回退后，对比代码发现，引脚只有IRQ不一样。而这个引脚并不影响初始化。于是我们将回退版本的代码烧入小车程序里，发现初始化依旧失败。但是之前验证是通过的。程序没有问题。于是我们将NRF24L01从小车上取下，用另一块stm32C8T6最小系统去测试。发现初始化成功了。程序正常运行。然后再次测试最新的的程序，程序依旧能够成功运行。说明小车的C8T6模块可能发生了损坏。于是我们将小车上的C8T6模块取下，烧入程序，进行验证。测试结果是stm32C8T6最小系统板发生损坏。

* + 1. NRF24L01动态字节配置失败。

去掉动态字节配置部分代码，发现程序依旧无法进行无线通信。于是我们只有回退到上一个成功运行的版本进行测试。发现程序能够成功通信。说明程序出在后续的代码改动上。因为之前的程序代码，只能运行接收模式或者发送模式，而接收和发送模式里面对寄存器的配置有重复部分。所以我们将相同的部分提取到了模块初始化中进行配置。这一部分移植经过测试是成功的。然后继续对比代码后发现，当初始化NRF24L01\_Write\_Reg (NRF\_WRITE\_REG+SETUP\_AW，2) ; 设置地址宽度(所有数据通道)02,5字节后，通信便会发生异常。于是我们查找数据手册对该寄存器进行深入的解析。该寄存器的高6位是默认为0，且不做修改的。我们只关注最后两位，最后两位对应着4种模式，分别为：00，无效。01，3字节宽度。10,4字节宽度。11,5字节宽度。而我们在配置这个寄存器的时候配置为了2，代表4字节的地址宽度。而实际上我们使用的是5字节的地址宽度。所以我们将值改为3。然后重新编译代码并烧入，结果通信恢复正常，问题解决。

在将优化后的代码移植到手柄遥控的主程序上去时，通信又出现了失败问题。最后发现是因为屏蔽掉了配置数据位宽的寄存器写入。将屏蔽取消后，程序恢复正常。通信可以保持发送和接受数据。

* + 1. Oled初始化失败问题

在最初版本的初始化中，我们发现oled的指令虽然成功发送了，但是oled完全没有反应。这时，我们需要一个能够成功运行的代码来对比分析。于是我们找到了卖我们oled屏幕的店家要来了oled的点亮例程。将例程烧入芯片中后，我们发现店家的程序能够成功运行。说明不是硬件问题。于是我们对比分析两者的代码，发现在写寄存器这个操作，例程有两种模式。一种是写数据，一种是写命令。为什么会有这种差别呢？于是我们查看了数据手册分析代码。发现当DC引脚被拉高的时候，代表是写入命令。当DC命令拉低时，代表是写入数据。于是我们对，写入操作进行了优化，在写指令里加入了模式判断。但是，依旧没有解决问题。于是继续分析代码，最终发现是因为使用的数据手册里缺失了设置电源是否打开的指令。加上这句指令后，程序运行成功。

* + 1. 手柄程序出现程序死机的情况

在修改了手柄的通信发送逻辑后，手柄出现了死机的情况。且死机后，oled的第5行对2.4G接收消息的显示会出现乱码。用debug定位程序问题发现，程序在任务3里发生了错误。对比回退版本的代码，发现新增代码是2.4G通信的接受数据函数。与oled死机后，显示乱码的现象极度相关。于是我们试着屏蔽掉这句话，来看看是否能解决问题。结果屏蔽后，代码正常运行。于是我们分析这句代码可能存在的问题发现，任务3早于任务4创建，且任务3的运行周期是10ms，任务4的运行周期是100ms。这很大可能发生，当执行接收函数的时候，2.4G的通信还没有初始化。于是我们在此基础上，对程序进行修改，让接收数据函数程序只能在初始化后运行。结果问题依旧没有解决。现象依旧是程序运行一段时间后，接收显示乱码，程序死机。说明思路错误，于是我们把之前能够运行的程序代码下载进入手柄，测试发现程序依旧出现死机现象。于是我们对程序debug。最后定位到程序死在了一个对中断io口检测的while循环里面。说明硬件也许发生了问题。于是我们检测硬件电路。发现IRQ引脚发生了松动。重新焊接后，问题解决。但是最新版的代码依旧有问题。我们继续屏蔽接收，oled程序。发现单独运行接收程序是没有问题的。但是oled运行时，程序会出现问题。我们试着增加任务的优先级，问题解决。但是打开接受任务后，程序依旧发生了死机现象。说明接受程序和发送程序存在冲突。然后我们单独运行接收程序时，有趣的现象发生了。随MPU的姿态角获取，接收程序会出现接收到数据的情况。但是这是不可能出现的情况。于是我们尝试着更换通信模块。程序所有异常消失。说明硬件损坏。但是手头已经没有多于的通信模块了。于是我们只好选择了ASC1-ML01DP5这款新的2.4G通信模块。在查看模块的数据手册后发现完全与当前模块兼容，可以直接换上。更换后，问题解决。

* + 1. 用定时器的模块太多，资源不够用。

因为最开始没有设计编码器的的程序设计，定时器是够用的。但是后面发现，两个车轮就要用两个定时器。加上舵机需要一个定时器。本身电机驱动需要一个定时器，就足以使用了完了四个定时器。已经没有多于的资源增加一个超声波了。

* 1. 本章小结

智能寻迹小车这种课题已经被大量的验证过了。前人也给出了大量可行的方案，本论文中只能在前人的研究基础上给出一种简易的，可行的，便宜的，能够快速构架的嵌入式产品方案。但是由于时间原因。只实现了简易的系统，只完成了最基础的寻迹功能。但是在最初的程序设计架构上，我提前将各种通信接口预留了出来，以便后期能快速的进行功能上的扩展。比如手柄的控制部分，最终的方案是要设计电路板，将设计微型化嵌入在手套上的，然后通过手势的操作去控制小车的行动。至于小车部分，红外探测的寻迹方案是远远不能满足复杂地形下的寻迹的。如果采用摄像头进行道路情况的实时采集和搜集，通过图像分析的智能算法，便能实现更加可靠的智能寻迹系统。并且可以外加各种信息采集系统，比如，温度湿度，气体检测，气压检测，距离检测等。并可以通过网络系统实时将数据上传。由于时间和成本关系，产品并没有完成外观设计，和电路板的设计。无法达到商品级别的产品设计，只停留在了功能实现和验证的基础上。

参考文献

* + - 1. 陈明妙, 吴汶, 李曼. 基于STC89C52单片机的智能寻迹小车的设计[J]. 湛江师范学院学报, 2013, 34(6):7.
      2. 李万义, 谢林汐, 肖锋,等. 基于STM32的智能小车寻迹避障系统硬件设计[J]. 电子世界, 2019(7):2.
      3. 李鹏博. 基于视频的自动巡航智能小车的设计与实现[J]. 河北工程大学学报(自然科学版), 2012, 29(4):74-78.
      4. 胡媛媛, 邓世建, 王书婧. 基于红外光电传感器的智能寻迹小车设计[J]. 电子设计工程, 2011, 19(007):141-143.
      5. 刘群, 郑丹莹, 刘鸣. 红外反射式传感器的自寻迹小车的设计[J]. 实验科学与技术, 2007, 5(5):3.
      6. Jitsukata E , Kobayashi S , Tamura K . Automatic driving system: US, US6169940 B1[P]. 2001.
      7. Nagel H H , Enkelmann W , Struck G . FhG-Co-driver: From map-guided automatic driving by machine vision to a cooperative driver support[J]. Mathematical & Computer Modelling, 1995, 22(4-7):185-212.
      8. Naranjo J E , Gonzalez C , Pedro T D , et al. AUTOPIA architecture for automatic driving and maneuvering[C]// Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE. IEEE, 2006.
      9. Omae M , Ogitsu T , Honma N , et al. Automatic Driving Control for Passing through Intersection without Stopping[J]. International Journal of Intelligent Transportation Systems Research, 2010, 8(3):201-210.
      10. Ososinski M , Labrosse F . Automatic Driving on Ill‐defined Roads: An Adaptive, Shape‐constrained, Color‐based Method[J]. Journal of Field Robotics, 2015, 32(4):504-533.

致谢

在课题实现过程中，指导老师们给了我极大的帮助，母校电子科技大学成都学院也提供了大量资料，文献的支持。并且成立了相关工作组实时的掌握我们的论文进度，以便根据我们的论文完成情况，给与我们及时的帮助。

附录

附录一：手柄的部分.h文件

#ifndef \_\_bsp\_oled096\_spi\_H

#define \_\_bsp\_oled096\_spi\_H

#include "main.h"

#include "stdlib.h"

//OLED模式设置

//0:4线串行模式

//1:并行8080模式

#define OLED\_MODE 0

#define SIZE 16

#define XLevelL 0x00

#define XLevelH 0x10

#define Max\_Column 128

#define Max\_Row 64

#define Brightness 0xFF

#define X\_WIDTH 128

#define Y\_WIDTH 64

//-----------------OLED端口定义----------------

#define OLED\_SCLK\_Clr() HAL\_GPIO\_WritePin(OLED\_SCLK\_GPIO\_Port,OLED\_SCLK\_Pin,GPIO\_PIN\_RESET)//CLK

#define OLED\_SCLK\_Set() HAL\_GPIO\_WritePin(OLED\_SCLK\_GPIO\_Port,OLED\_SCLK\_Pin,GPIO\_PIN\_SET)

#define OLED\_SDIN\_Clr() HAL\_GPIO\_WritePin(OLED\_SDIN\_GPIO\_Port,OLED\_SDIN\_Pin,GPIO\_PIN\_RESET)//DIN

#define OLED\_SDIN\_Set() HAL\_GPIO\_WritePin(OLED\_SDIN\_GPIO\_Port,OLED\_SDIN\_Pin,GPIO\_PIN\_SET)

#define OLED\_RST\_Clr() HAL\_GPIO\_WritePin(OLED\_RET\_GPIO\_Port,OLED\_RET\_Pin,GPIO\_PIN\_RESET)//RES

#define OLED\_RST\_Set() HAL\_GPIO\_WritePin(OLED\_RET\_GPIO\_Port,OLED\_RET\_Pin,GPIO\_PIN\_SET)

#define OLED\_DC\_Clr() HAL\_GPIO\_WritePin(OLED\_DC\_GPIO\_Port,OLED\_DC\_Pin,GPIO\_PIN\_RESET)//DC

#define OLED\_DC\_Set() HAL\_GPIO\_WritePin(OLED\_DC\_GPIO\_Port,OLED\_DC\_Pin,GPIO\_PIN\_SET)

#define OLED\_CS\_Clr() HAL\_GPIO\_WritePin(OLED\_CS\_GPIO\_Port,OLED\_CS\_Pin,GPIO\_PIN\_RESET)//CS

#define OLED\_CS\_Set() HAL\_GPIO\_WritePin(OLED\_CS\_GPIO\_Port,OLED\_CS\_Pin,GPIO\_PIN\_SET)

#define OLED\_CMD 0 //写命令

#define OLED\_DATA 1 //写数据

//-----------------OLEDcmd指令定义----------------

#define SetColumnAddress (0x21)//设置列地址

#define SetPageAddress (0X22)//设置页面地址

//OLED控制用函数

void OLED\_WR\_Byte(u8 dat,u8 cmd);

void OLED\_Display\_On(void);

void OLED\_Display\_Off(void);

void OLED\_Init(void);

void OLED\_Clear(void);

void OLED\_DrawPoint(u8 x,u8 y,u8 t);

void OLED\_Fill(u8 x1,u8 y1,u8 x2,u8 y2,u8 dot);

void OLED\_ShowChar(u8 x,u8 y,u8 chr, u8 size);

void OLED\_ShowNum(u8 x,u8 y,u32 num,u8 len,u8 size);

void OLED\_ShowString(u8 x,u8 y,u8 \*chr,u8 size);

void OLED\_Set\_Pos(unsigned char x, unsigned char y);

void OLED\_ShowCHinese(u8 x,u8 y,u8 no);

void OLED\_DrawBMP(unsigned char x0, unsigned char y0,unsigned char x1, unsigned char y1,unsigned char BMP[]);

#endif

#ifndef \_BSP\_MPU6050\_H

#define \_BSP\_MPU6050\_H

#include "i2c.h"

#include "mpuiic.h"

#include "main.h"

extern float pitch,roll,yaw; //欧拉角

extern short aacx,aacy,aacz; //加速度传感器原始数据

extern short gyrox,gyroy,gyroz; //陀螺仪原始数据

extern float temp; //温度

//MPU6050 AD0控制脚

//#define MPU\_AD0\_CTRL PAout(15) //控制AD0电平,从而控制MPU地址

//#define MPU\_ACCEL\_OFFS\_REG 0X06 //accel\_offs寄存器,可读取版本号,寄存器手册未提到

//#define MPU\_PROD\_ID\_REG 0X0C //prod id寄存器,在寄存器手册未提到

#define MPU\_SELF\_TESTX\_REG 0X0D //自检寄存器X

#define MPU\_SELF\_TESTY\_REG 0X0E //自检寄存器Y

#define MPU\_SELF\_TESTZ\_REG 0X0F //自检寄存器Z

#define MPU\_SELF\_TESTA\_REG 0X10 //自检寄存器A

#define MPU\_SAMPLE\_RATE\_REG 0X19 //采样频率分频器

#define MPU\_CFG\_REG 0X1A //配置寄存器

#define MPU\_GYRO\_CFG\_REG 0X1B //陀螺仪配置寄存器

#define MPU\_ACCEL\_CFG\_REG 0X1C //加速度计配置寄存器

#define MPU\_MOTION\_DET\_REG 0X1F //运动检测阀值设置寄存器

#define MPU\_FIFO\_EN\_REG 0X23 //FIFO使能寄存器

#define MPU\_I2CMST\_CTRL\_REG 0X24 //IIC主机控制寄存器

#define MPU\_I2CSLV0\_ADDR\_REG 0X25 //IIC从机0器件地址寄存器

#define MPU\_I2CSLV0\_REG 0X26 //IIC从机0数据地址寄存器

#define MPU\_I2CSLV0\_CTRL\_REG 0X27 //IIC从机0控制寄存器

#define MPU\_I2CSLV1\_ADDR\_REG 0X28 //IIC从机1器件地址寄存器

#define MPU\_I2CSLV1\_REG 0X29 //IIC从机1数据地址寄存器

#define MPU\_I2CSLV1\_CTRL\_REG 0X2A //IIC从机1控制寄存器

#define MPU\_I2CSLV2\_ADDR\_REG 0X2B //IIC从机2器件地址寄存器

#define MPU\_I2CSLV2\_REG 0X2C //IIC从机2数据地址寄存器

#define MPU\_I2CSLV2\_CTRL\_REG 0X2D //IIC从机2控制寄存器

#define MPU\_I2CSLV3\_ADDR\_REG 0X2E //IIC从机3器件地址寄存器

#define MPU\_I2CSLV3\_REG 0X2F //IIC从机3数据地址寄存器

#define MPU\_I2CSLV3\_CTRL\_REG 0X30 //IIC从机3控制寄存器

#define MPU\_I2CSLV4\_ADDR\_REG 0X31 //IIC从机4器件地址寄存器

#define MPU\_I2CSLV4\_REG 0X32 //IIC从机4数据地址寄存器

#define MPU\_I2CSLV4\_DO\_REG 0X33 //IIC从机4写数据寄存器

#define MPU\_I2CSLV4\_CTRL\_REG 0X34 //IIC从机4控制寄存器

#define MPU\_I2CSLV4\_DI\_REG 0X35 //IIC从机4读数据寄存器

#define MPU\_I2CMST\_STA\_REG 0X36 //IIC主机状态寄存器

#define MPU\_INTBP\_CFG\_REG 0X37 //中断/旁路设置寄存器

#define MPU\_INT\_EN\_REG 0X38 //中断使能寄存器

#define MPU\_INT\_STA\_REG 0X3A //中断状态寄存器

#define MPU\_ACCEL\_XOUTH\_REG 0X3B //加速度值,X轴高8位寄存器

#define MPU\_ACCEL\_XOUTL\_REG 0X3C //加速度值,X轴低8位寄存器

#define MPU\_ACCEL\_YOUTH\_REG 0X3D //加速度值,Y轴高8位寄存器

#define MPU\_ACCEL\_YOUTL\_REG 0X3E //加速度值,Y轴低8位寄存器

#define MPU\_ACCEL\_ZOUTH\_REG 0X3F //加速度值,Z轴高8位寄存器

#define MPU\_ACCEL\_ZOUTL\_REG 0X40 //加速度值,Z轴低8位寄存器

#define MPU\_TEMP\_OUTH\_REG 0X41 //温度值高八位寄存器

#define MPU\_TEMP\_OUTL\_REG 0X42 //温度值低8位寄存器

#define MPU\_GYRO\_XOUTH\_REG 0X43 //陀螺仪值,X轴高8位寄存器

#define MPU\_GYRO\_XOUTL\_REG 0X44 //陀螺仪值,X轴低8位寄存器

#define MPU\_GYRO\_YOUTH\_REG 0X45 //陀螺仪值,Y轴高8位寄存器

#define MPU\_GYRO\_YOUTL\_REG 0X46 //陀螺仪值,Y轴低8位寄存器

#define MPU\_GYRO\_ZOUTH\_REG 0X47 //陀螺仪值,Z轴高8位寄存器

#define MPU\_GYRO\_ZOUTL\_REG 0X48 //陀螺仪值,Z轴低8位寄存器

#define MPU\_I2CSLV0\_DO\_REG 0X63 //IIC从机0数据寄存器

#define MPU\_I2CSLV1\_DO\_REG 0X64 //IIC从机1数据寄存器

#define MPU\_I2CSLV2\_DO\_REG 0X65 //IIC从机2数据寄存器

#define MPU\_I2CSLV3\_DO\_REG 0X66 //IIC从机3数据寄存器

#define MPU\_I2CMST\_DELAY\_REG 0X67 //IIC主机延时管理寄存器

#define MPU\_SIGPATH\_RST\_REG 0X68 //信号通道复位寄存器

#define MPU\_MDETECT\_CTRL\_REG 0X69 //运动检测控制寄存器

#define MPU\_USER\_CTRL\_REG 0X6A //用户控制寄存器

#define MPU\_PWR\_MGMT1\_REG 0X6B //电源管理寄存器1

#define MPU\_PWR\_MGMT2\_REG 0X6C //电源管理寄存器2

#define MPU\_FIFO\_CNTH\_REG 0X72 //FIFO计数寄存器高八位

#define MPU\_FIFO\_CNTL\_REG 0X73 //FIFO计数寄存器低八位

#define MPU\_FIFO\_RW\_REG 0X74 //FIFO读写寄存器

#define MPU\_DEVICE\_ID\_REG 0X75 //器件ID寄存器

//如果AD0脚(9脚)接地,IIC地址为0X68(不包含最低位).

//如果接V3.3,则IIC地址为0X69(不包含最低位).

#define MPU\_ADDR 0X68

////因为模块AD0默认接GND,所以转为读写地址后,为0XD1和0XD0(如果接VCC,则为0XD3和0XD2)

//#define MPU\_READ 0XD1

//#define MPU\_WRITE 0XD0

u8 MPU\_Init(void); //初始化MPU6050

u8 MPU\_Write\_Len(u8 addr,u8 reg,u8 len,u8 \*buf);//IIC连续写

u8 MPU\_Read\_Len(u8 addr,u8 reg,u8 len,u8 \*buf); //IIC连续读

u8 MPU\_Write\_Byte(u8 reg,u8 data); //IIC写一个字节

u8 MPU\_Read\_Byte(u8 reg); //IIC读一个字节

u8 MPU\_Set\_Gyro\_Fsr(u8 fsr);

u8 MPU\_Set\_Accel\_Fsr(u8 fsr);

u8 MPU\_Set\_LPF(u16 lpf);

u8 MPU\_Set\_Rate(u16 rate);

u8 MPU\_Set\_Fifo(u8 sens);

float MPU\_Get\_Temperature(void);

u8 MPU\_Get\_Gyroscope(short \*gx,short \*gy,short \*gz);

u8 MPU\_Get\_Accelerometer(short \*ax,short \*ay,short \*az);

#endif

附录二：小车的部分.h文件

#ifndef \_BSP\_LED\_H

#define \_BSP\_LED\_H

#include "main.h"

#define led\_on HAL\_GPIO\_WritePin(GPIOC, GPIO\_PIN\_13, GPIO\_PIN\_RESET);//引脚电平拉低，点亮

#define led\_off HAL\_GPIO\_WritePin(GPIOC, GPIO\_PIN\_13, GPIO\_PIN\_SET);//引脚电平拉高，熄灭

typedef enum

{

led\_ok = 1u,

led\_time\_err,

} led\_err\_t;

led\_err\_t led\_twinkle(int time);

#endif /\* \_\_LED\_H \*/

#ifndef \_BSP\_MOTOR\_H

#define \_BSP\_MOTOR\_H

#include "main.h"

#include <stdio.h>

extern int base\_speed;

//轮子的速度 tim3的通道1或通道2

extern TIM\_HandleTypeDef htim3;

#define left\_wheel\_speed(x) if(x>=0&&x<=7200) \

{\_\_HAL\_TIM\_SET\_COMPARE(&htim3, TIM\_CHANNEL\_1,x);HAL\_TIM\_PWM\_Start(&htim3,TIM\_CHANNEL\_1);} \

else \

{printf("left\_wheel\_speed 失败!\r\n");}

#define right\_wheel\_speed(x) if(x>=0&&x<=7200) \

{\_\_HAL\_TIM\_SET\_COMPARE(&htim3, TIM\_CHANNEL\_2,x);HAL\_TIM\_PWM\_Start(&htim3,TIM\_CHANNEL\_2);} \

else \

{printf("right\_wheel\_speed 失败!\r\n");}

/\*

//轮子的定义

转向不对时，只需修改AB接口对应的引脚就好

\*/

#define L\_wheel\_B TB6612\_A1\_Pin

#define L\_wheel\_A TB6612\_A0\_Pin

#define R\_wheel\_B TB6612\_B1\_Pin

#define R\_wheel\_A TB6612\_B0\_Pin

//轮子前进方式的定义

#define LF\_RUN {HAL\_GPIO\_WritePin(GPIOB, L\_wheel\_A, GPIO\_PIN\_RESET);HAL\_GPIO\_WritePin(GPIOB, L\_wheel\_B, GPIO\_PIN\_SET); }

#define RF\_RUN {HAL\_GPIO\_WritePin(GPIOB, R\_wheel\_A, GPIO\_PIN\_RESET);HAL\_GPIO\_WritePin(GPIOB, R\_wheel\_B, GPIO\_PIN\_SET); }

#define LB\_RUN {HAL\_GPIO\_WritePin(GPIOB, L\_wheel\_A, GPIO\_PIN\_SET);HAL\_GPIO\_WritePin(GPIOB, L\_wheel\_B, GPIO\_PIN\_RESET); }

#define RB\_RUN {HAL\_GPIO\_WritePin(GPIOB, R\_wheel\_A, GPIO\_PIN\_SET);HAL\_GPIO\_WritePin(GPIOB, R\_wheel\_B, GPIO\_PIN\_RESET); }

#define F\_RUN {LF\_RUN;RF\_RUN}

#define B\_RUN {LB\_RUN;RB\_RUN}

#define CAR\_STOP {HAL\_GPIO\_WritePin(GPIOB, L\_wheel\_A|L\_wheel\_B|R\_wheel\_A|R\_wheel\_B, GPIO\_PIN\_RESET);}

void forward\_run(int TargetVelocity,int TargetCircle);

void back\_run(int TargetVelocity,int TargetCircle);

void left\_run(int TargetVelocity,int TargetCircle);

void right\_run(int TargetVelocity,int TargetCircle);

void car\_stop(void);

#endif /\* \_BSP\_MOTOR\_H \*/

#ifndef \_BSP\_INFRA\_RED\_H

#define \_BSP\_INFRA\_RED\_H

#include "FreeRTOS.h"

#include "event\_groups.h"

extern EventGroupHandle\_t infra\_red\_Event\_Handle;

#define left\_check infrared\_sensor\_left\_Pin

#define right\_check infrared\_sensor\_right\_Pin

#define LeftCheck\_EVENT (0x01 << 0)//设置事件掩码的位0

#define RightCheck\_EVENT (0x01 << 1)//设置事件掩码的位1

#define left\_state HAL\_GPIO\_ReadPin(GPIOB, left\_check)

#define right\_state HAL\_GPIO\_ReadPin(GPIOB, right\_check)

/\*红外接受到产生低电平，没接收到红外，产生高电平。\*/

#define state\_on GPIO\_PIN\_SET

#define state\_off GPIO\_PIN\_RESET

void infra\_red\_init(void );

void check\_app(void);

#endif /\* \_BSP\_INFRA\_RED\_H \*/

#ifndef bsp\_nrf24l01\_H

#define bsp\_nrf24l01\_H

#include "main.h"

#define u8 unsigned char

//NRF24L01寄存器操作命令

#define NRF\_READ\_REG 0x00 //读配置寄存器,低5位为寄存器地址

#define NRF\_WRITE\_REG 0x20 //写配置寄存器,低5位为寄存器地址

#define R\_RX\_PL\_WID 0x60 //读取接收数据长度

#define RD\_RX\_PLOAD 0x61 //读RX有效数据,1~32字节

#define WR\_TX\_PLOAD 0xA0 //写TX有效数据,1~32字节

#define FLUSH\_TX 0xE1 //清除TX FIFO寄存器.发射模式下用

#define FLUSH\_RX 0xE2 //清除RX FIFO寄存器.接收模式下用

#define REUSE\_TX\_PL 0xE3 //重新使用上一包数据,CE为高,数据包被不断发送.

#define NOP 0xFF //空操作,可以用来读状态寄存器

//SPI(NRF24L01)寄存器地址

#define CONFIG 0x00 //配置寄存器地址;bit0:1接收模式,0发射模式;bit1:电选择;bit2:CRC模式;bit3:CRC使能;

//bit4:中断MAX\_RT(达到最大重发次数中断)使能;bit5:中断TX\_DS使能;bit6:中断RX\_DR使能

#define EN\_AA 0x01 //使能自动应答功能 bit0~5,对应通道0~5

#define EN\_RXADDR 0x02 //接收地址允许,bit0~5,对应通道0~5

#define SETUP\_AW 0x03 //设置地址宽度(所有数据通道):bit1,0:00,3字节;01,4字节;02,5字节;

#define SETUP\_RETR 0x04 //建立自动重发;bit3:0,自动重发计数器;bit7:4,自动重发延时 250\*x+86us

#define RF\_CH 0x05 //RF通道,bit6:0,工作通道频率;

#define RF\_SETUP 0x06 //RF寄存器;bit3:传输速率(0:1Mbps,1:2Mbps);bit2:1,发射功率;bit0:低噪声放大器增益

#define STATUS 0x07 //状态寄存器;bit0:TX FIFO满标志;bit3:1,接收数据通道号(最大:6);bit4,达到最多次重发

//bit5:数据发送完成中断;bit6:接收数据中断;

#define MAX\_TX 0x10 //达到最大发送次数中断

#define TX\_OK 0x20 //TX发送完成中断

#define RX\_OK 0x40 //接收到数据中断

#define OBSERVE\_TX 0x08 //发送检测寄存器,bit7:4,数据包丢失计数器;bit3:0,重发计数器

#define CD 0x09 //载波检测寄存器,bit0,载波检测;

#define RX\_ADDR\_P0 0x0A //数据通道0接收地址,最大长度5个字节,低字节在前

#define RX\_ADDR\_P1 0x0B //数据通道1接收地址,最大长度5个字节,低字节在前

#define RX\_ADDR\_P2 0x0C //数据通道2接收地址,最低字节可设置,高字节,必须同RX\_ADDR\_P1[39:8]相等;

#define RX\_ADDR\_P3 0x0D //数据通道3接收地址,最低字节可设置,高字节,必须同RX\_ADDR\_P1[39:8]相等;

#define RX\_ADDR\_P4 0x0E //数据通道4接收地址,最低字节可设置,高字节,必须同RX\_ADDR\_P1[39:8]相等;

#define RX\_ADDR\_P5 0x0F //数据通道5接收地址,最低字节可设置,高字节,必须同RX\_ADDR\_P1[39:8]相等;

#define TX\_ADDR 0x10 //发送地址(低字节在前),ShockBurstTM模式下,RX\_ADDR\_P0与此地址相等

#define RX\_PW\_P0 0x11 //接收数据通道0有效数据宽度(1~32字节),设置为0则非法

#define RX\_PW\_P1 0x12 //接收数据通道1有效数据宽度(1~32字节),设置为0则非法

#define RX\_PW\_P2 0x13 //接收数据通道2有效数据宽度(1~32字节),设置为0则非法

#define RX\_PW\_P3 0x14 //接收数据通道3有效数据宽度(1~32字节),设置为0则非法

#define RX\_PW\_P4 0x15 //接收数据通道4有效数据宽度(1~32字节),设置为0则非法

#define RX\_PW\_P5 0x16 //接收数据通道5有效数据宽度(1~32字节),设置为0则非法

#define NRF\_FIFO\_STATUS 0x17 //FIFO状态寄存器;bit0,RX FIFO寄存器空标志;bit1,RX FIFO满标志;bit2,3,保留

//bit4,TX FIFO空标志;bit5,TX FIFO满标志;bit6,1,循环发送上一数据包.0,不循环;

#define DYNPD 0x1C //动态长度地址;bit7~6,只允许'00';bit5~0,分别对应使能动态有效数据长度数据通道5~0

#define FEATURE 0x1D //功能寄存器地址;bit7~3,只允许'0000';bit2,使能动态有效数据长度;

//bit1,使能ACK有效数据;bit0,使能'W\_TX\_PAYLOAD\_NOACK'命令

#define NRF24L01\_CE(x) HAL\_GPIO\_WritePin(SPI\_CE\_GPIO\_Port, SPI\_CE\_Pin, x);//24L01片选信号

#define NRF24L01\_CSN(x) HAL\_GPIO\_WritePin(SPI\_CSN\_GPIO\_Port, SPI\_CSN\_Pin, x);//SPI片选信号

#define NRF24L01\_IRQ HAL\_GPIO\_ReadPin(SPI\_IRQ\_GPIO\_Port,SPI\_IRQ\_Pin) //IRQ主机数据输入

//24L01发送接收数据宽度定义

#define TX\_ADR\_WIDTH 5 //5字节的地址宽度

#define RX\_ADR\_WIDTH 5 //5字节的地址宽度

#define DYNPD\_enable 0 //动态数据宽度是否开启。1：开启，0：关闭

#define TX\_PLOAD\_WIDTH 32 //32字节的用户数据宽度

#define RX\_PLOAD\_WIDTH 32 //32字节的用户数据宽度

#ifdef RX\_PLOAD\_WIDTH

u8 NRF24L01\_TxPacket(u8 \*txbuf); //发送一个包的数据

#else

u8 NRF24L01\_TxPacket(u8 \*txbuf,u8 TX\_PLOAD\_WIDTH);

#endif

void NRF24L01\_Init(void); //初始化

void NRF24L01\_RX\_Mode(void); //配置为接收模式

void NRF24L01\_TX\_Mode(void); //配置为发送模式

u8 NRF24L01\_Write\_Buf(u8 reg, u8 \*pBuf, u8 u8s);//写数据区

u8 NRF24L01\_Read\_Buf(u8 reg, u8 \*pBuf, u8 u8s); //读数据区

u8 NRF24L01\_Read\_Reg(u8 reg); //读寄存器

u8 NRF24L01\_Write\_Reg(u8 reg, u8 value); //写寄存器

u8 NRF24L01\_Check(void); //检查24L01是否存在

u8 NRF24L01\_RxPacket(u8 \*rxbuf); //接收一个包的数据

extern u8 rf\_rxbuf[RX\_PLOAD\_WIDTH];

extern char rf\_txbuf[TX\_PLOAD\_WIDTH];

#endif