电控技术文档

（任务1、3、4、5合并在一个简易的摇控小车中，相应工程文件与展示视频见其它文件夹）

软件系统：STM32CubeIDE 1.19.0

注：1.cudeide的工程不能在keil中打开，已提交了cudeide和keil5两种版本的工程文件（因为前期都是用cudeide完成的，主体代码相同），本技术文档用的是cudeide工程，keil工程不能有中文注释，故代码移植后的keil5工程中无中文注释（乱码，变成了符号“？”）。keil工程调试时用的是ARM编译器5。实测cudeide和keil工程均可运行。

2.讲解了内置的cudemx的工程配置操作，但不对其生成的工程初始化代码作过多的解释

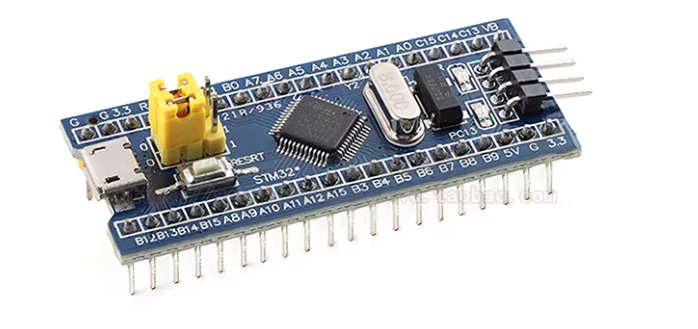
1. 串口通信。

Cudeide工程名称：serial

目标：控制三个LED灯（记为A，B，C）,串口接收两个字节的数据，并返回接收到的数据，第一个为LED灯的标志，第二个表示亮灭状态（1：亮，0：灭）。eg.A1表示A灯亮。

其中，状态为1（亮）时：A、B灯常亮，C灯闪烁（循环亮灭）

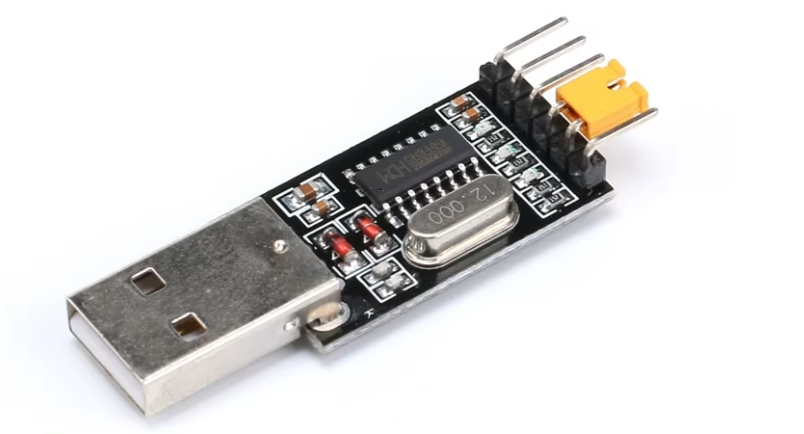
主要硬件及其图片：1、STM32f103c8t6最小系统开发板



2、stlink调试器



3、CH340模块串口



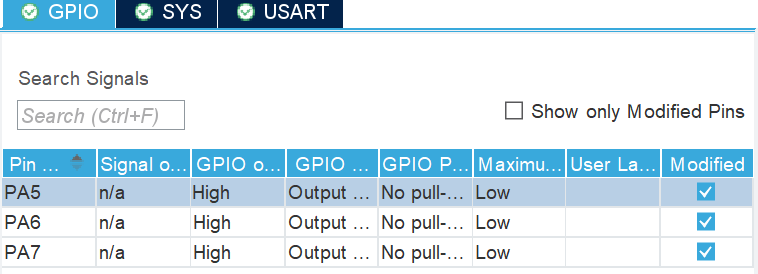
4、LED灯



主要的初始化配置：

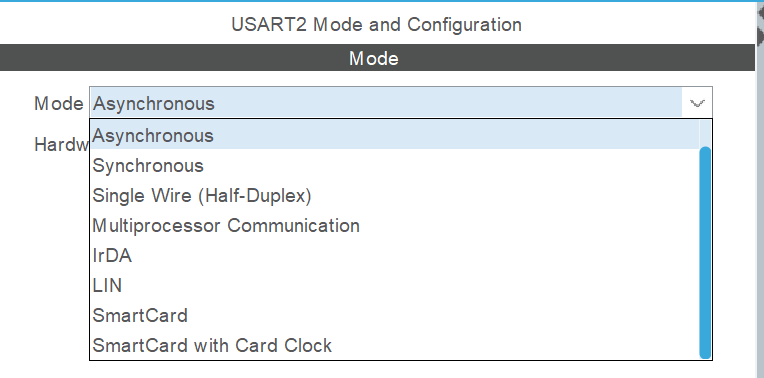
1. 设置A5、A6、A7（控制）为推挽输出模式，初始电平为高电平,其余保持默认：

分别控制灯A、B、C



1. 设置具有DMA通道的中断串口：

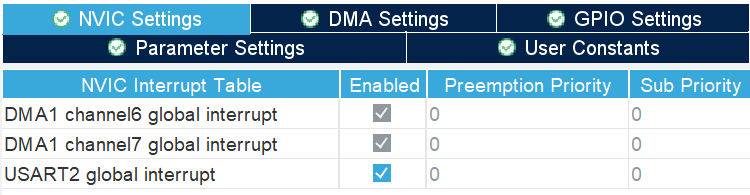
（1）设置USART2为异步模式（无需时钟线同步，不需要额外的时钟信号线，芯片对应接收RX为A3，发送TX为A2）



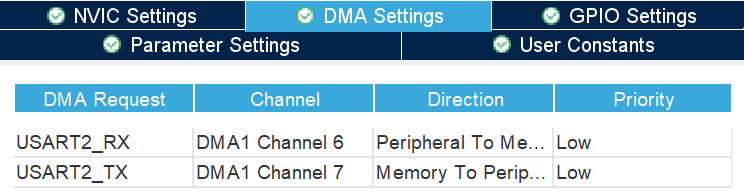
（2）比特率调为115200（蓝牙串口则换为9600）



（3）打开串口执行中断（需在while死循环中C灯执行循环亮灭时执行中断，进行串口接收）



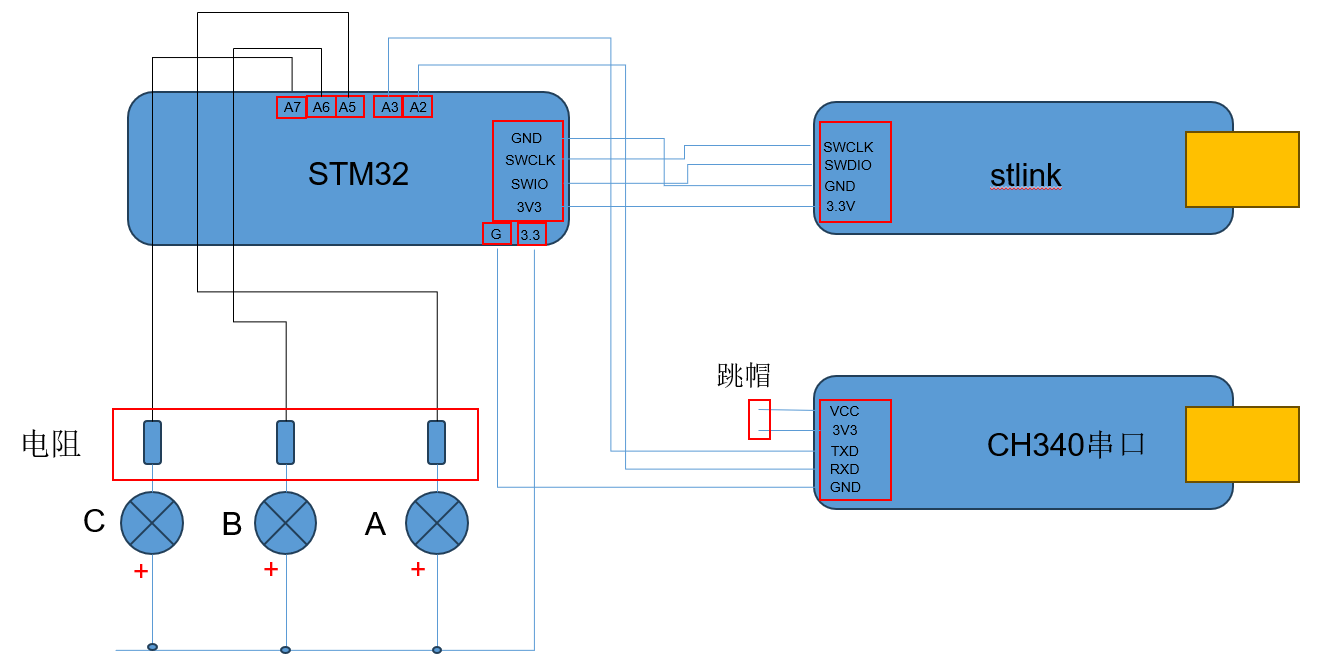
（4）为接收和发送设置DMA模式（建立接收发送DMA通道，提高效率，减少对CPU的占用）



其余保持不变

1. 生成初始化代码（直接在cudeide中进行代码的编辑与调试，不用keil、故无需将IDE的设置改为MDK-ARM）

总接线示意图：采用低电平LED灯亮的接线



主要代码：（main.c中）

注：相关说明见代码中注释

**1、main主函数外（前）：**

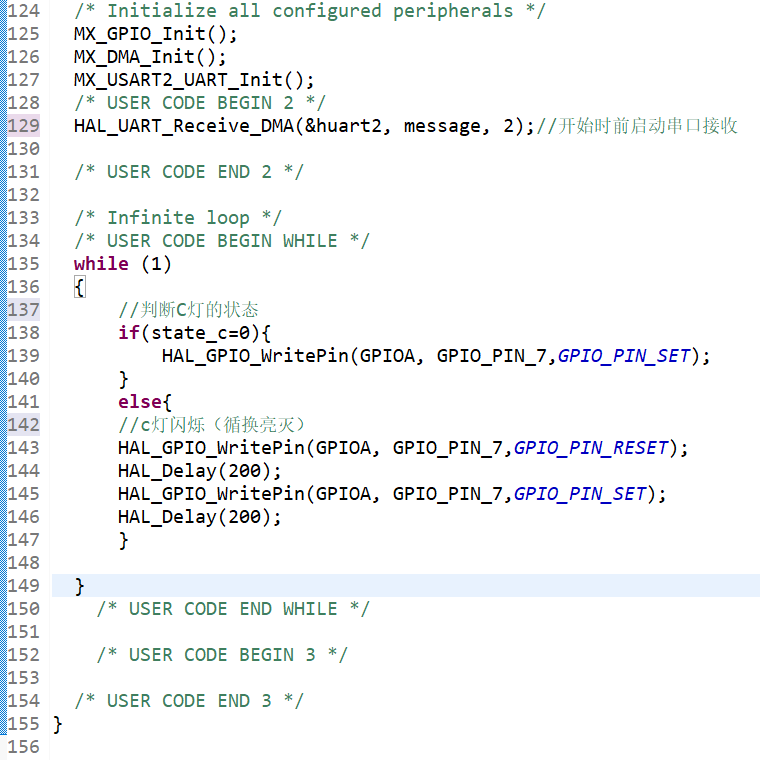
（1）设立相关变量（全局变量）



（2）撰写中断回调函数，进行数据的分析与处理，在每次中断处理完成后开启下 一次的串口接收



1. **main主函数中：**



1. **HAL库的相关函数：**
2. 延时函数： HAL\_Delay(Delay),括号内Delay为毫秒，eg：HAL\_Delay(100)表示延时100ms
3. 串口接收函数（DMA模式）：HAL\_UART\_Receive\_DMA(huart, pData, Size)
4. 串口发送函数（DMA模式）：HAL\_UART\_Transmit\_DMA(huart, pData, Size)
5. 改写GPIO口输出电平的函数：HAL\_GPIO\_WritePin(GPIOx, GPIO\_Pin, PinState)

测试：（展示视频在另一个文件夹）

1. 四驱遥控小车（激光模块的开与关，云台舵机的角度控制，底盘动⼒设备的速度控制，远程控制）。

cudeide工程名称：PS2\_case

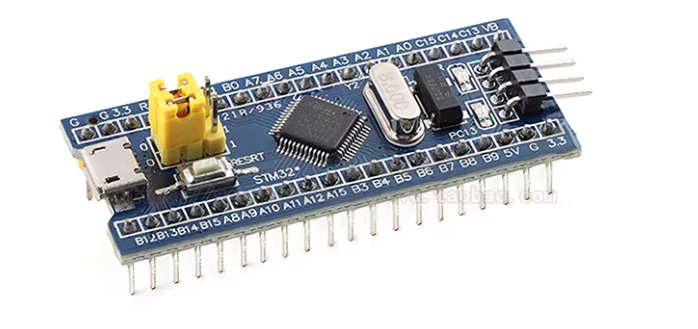
🡪cudeide和keil工程中，ps2.c和delay.c都与main.c在同一个路径，ps2.h和delay.h都与main.h在同一个路径

目标：

实现PS2手柄与STM32的通信，通过PS2手柄远程控制激光模块的亮灭、云台舵机的角度，以及底盘动力设备。

主要硬件及其图片：

1. STM32f103c8t6最小系统开发板



1. stlink调试器



1. 18650锂电池组



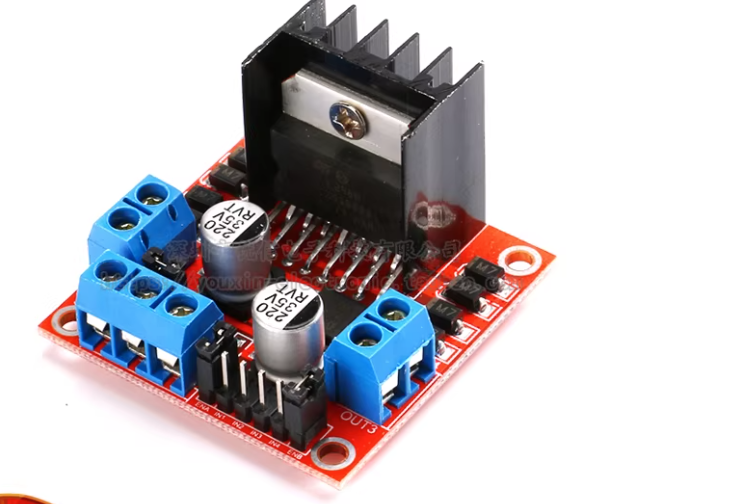
1. 激光模组



1. PS2无线手柄+直插转接板



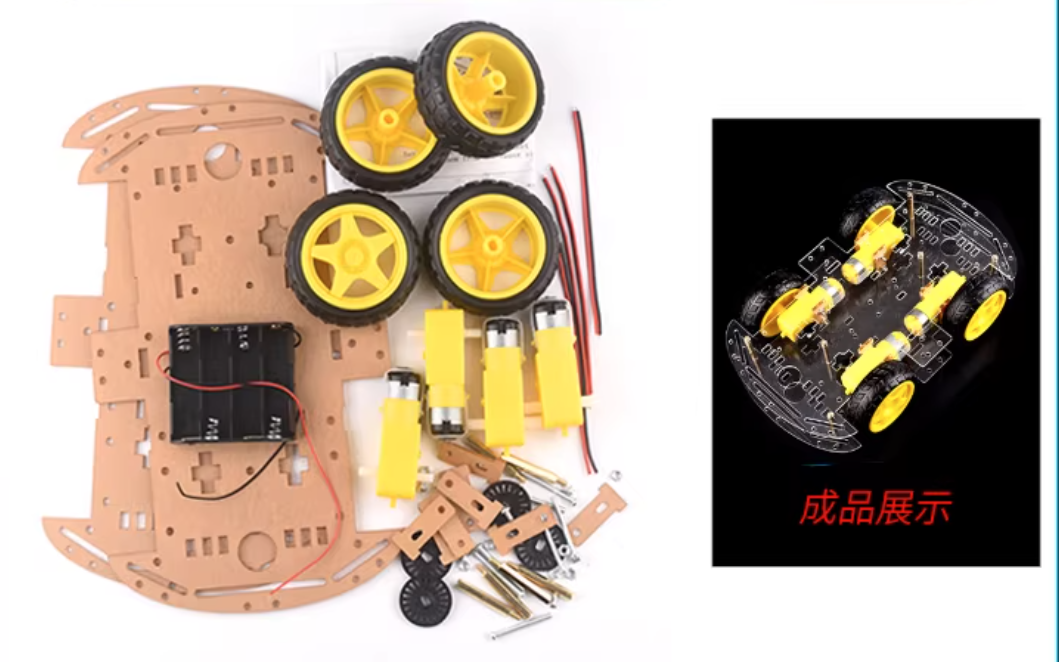
1. L298N电机驱动模块\*2



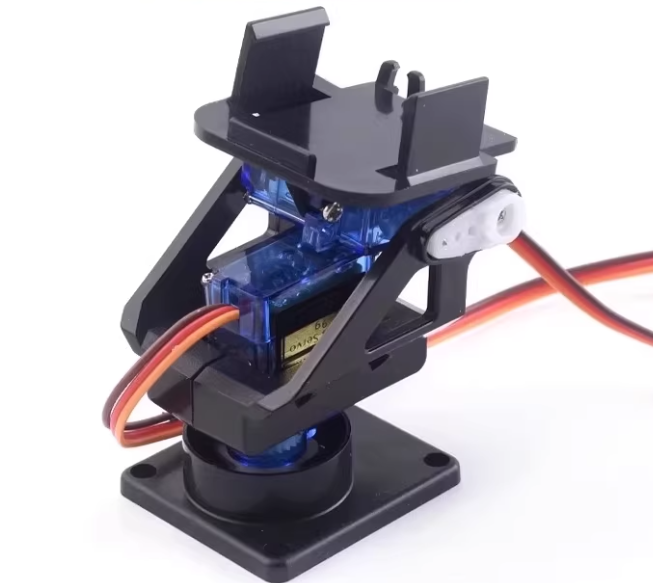
1. 船型开关



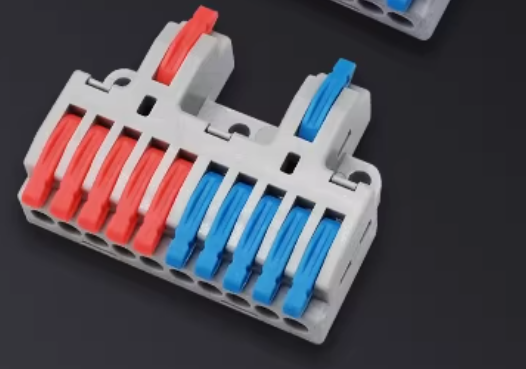
1. 4WD亚力克板小车套件：直流减速电机\*4，轮子\*4，亚力克板\*2



1. 二自由度舵机云台塑料支架+SG90舵机180度\*2



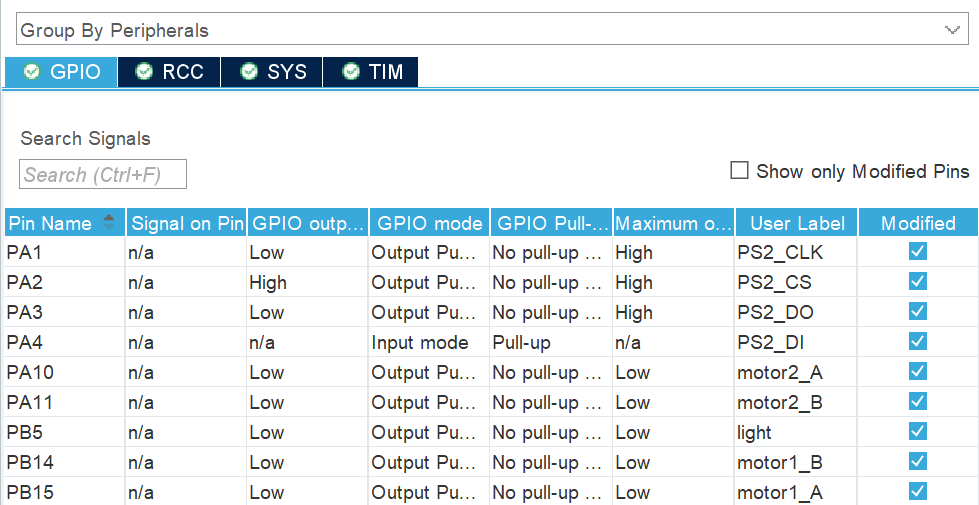
10、分线板



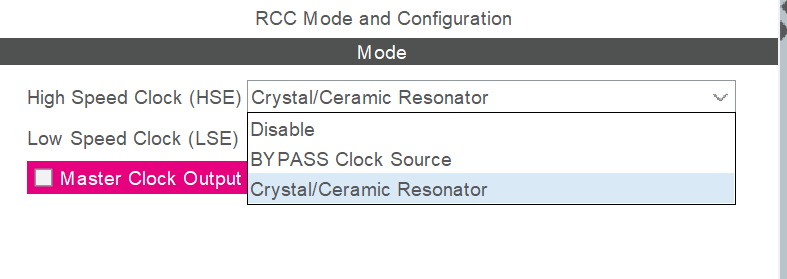
11、LM2596S DC-DC直流可调降压模块



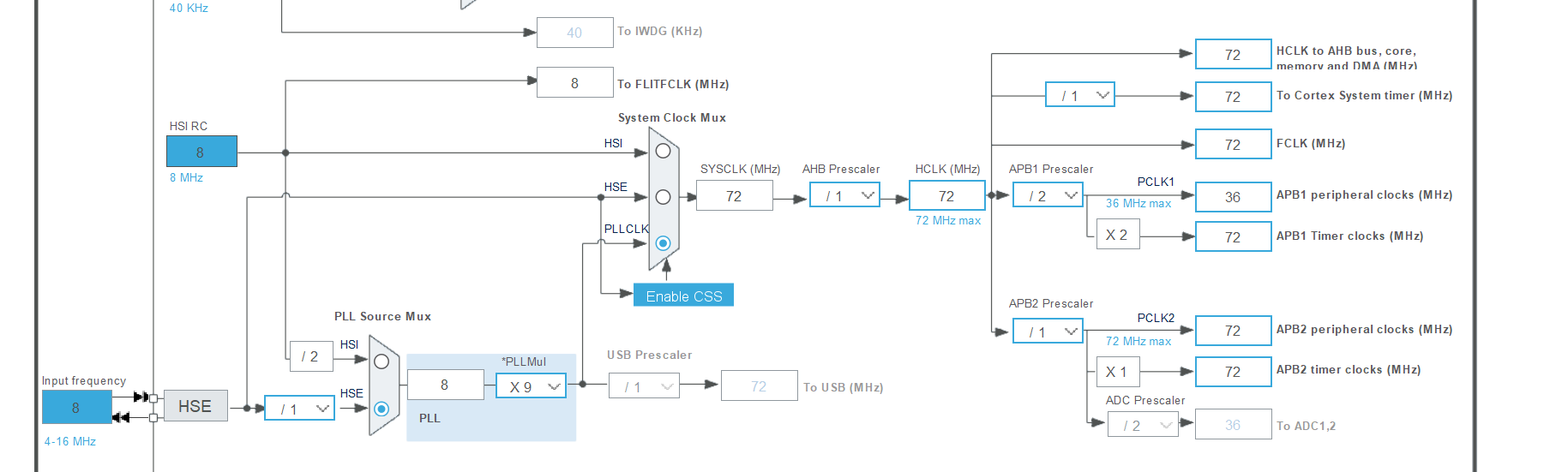
主要的初始化配置：

1、GPIO口设置：（均加上了相应的标签，便于使用和阅读）

1. PA1~4用于与PS2的通信：PA4用于接收数据，设置为上拉的输入模式；PA1~3均为推挽输出模式，将最大输出速度设置为高，GPIO口的初始电平不重要，后续可调。其中PA1为时钟线，用于同步PS2手柄的时钟。PA2为触发线，用于触发通信。PA3为输出线，用于向PS2手柄发送数据。
2. PB5用于控制激光的发射，设置为推挽输出的模式，输出高电平时发射激光，输出低电平时关闭激光。
3. PA10、PA11和PB14、PB15分别控制电机组2（右侧两个车轮）和电机组1（左侧两个车轮）的转向:输出电平相同时不转，一高一低时正转/反转。
4. 设置外部高速时钟源为外部晶振，提高精度，提供更精确、稳定的时钟信号

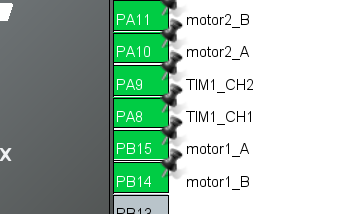
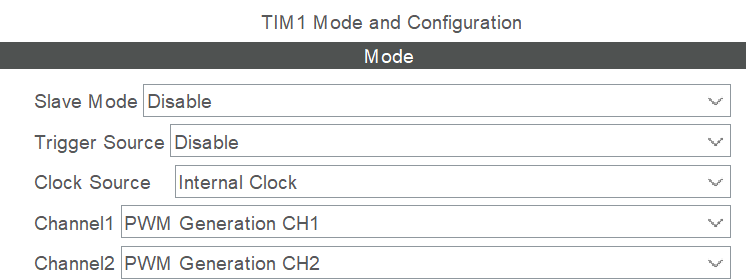


3、在时钟树配置中，将 AHB 总线时钟（HCLK）设置为 **72 MHz**，为定时器TIM 提供 72MHz 的基准时钟，再“回车”（“回车”可让系统自动完成分频配置）。

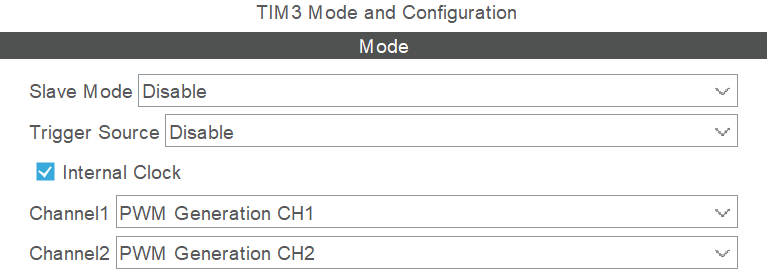
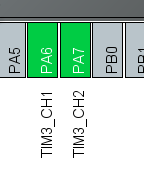


1. 设置TIM1和TIM3的Channel1、Channel为PWM generation模式，并选用内部时钟源作为时钟源（internal clock），内部时钟由系统主时钟（HCLK）经 APB 总线分频得到，稳定性高。

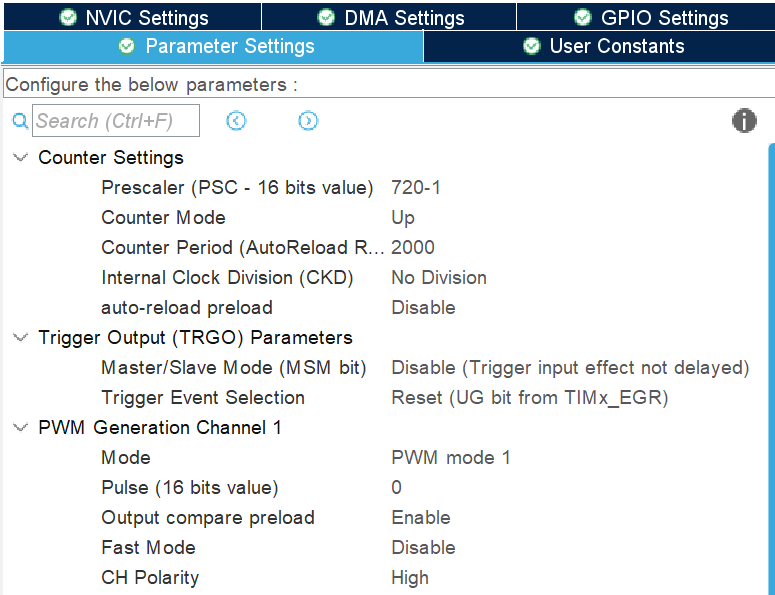
（1）TIM1的Channel1和2（PA8、PA9）用于控制左右车轮的转速：



（2）TIM3的Channel1和2（PA6、PA7）用于控制上下舵机的角度：

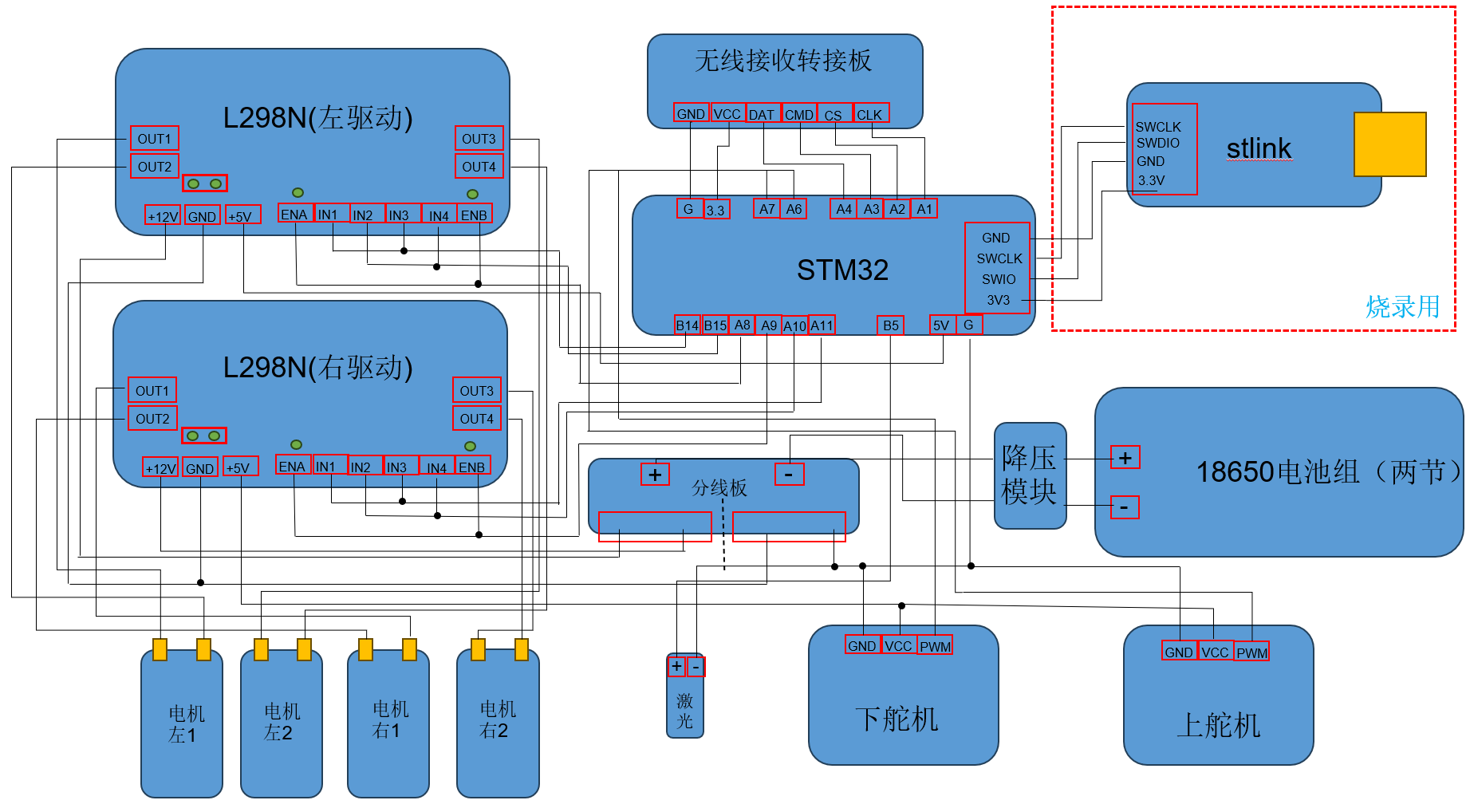
 

（3）定时器TIM（PWM模式）设置：四个均相同



预分频器设置为720-1，为720分频，自动重装载值设为2000，得到PWM频率约为50Hz。其余的均保持不变：向上计数，采用PWM mode1（计数器低于比较寄存器时为高电平，高于时为低电平）

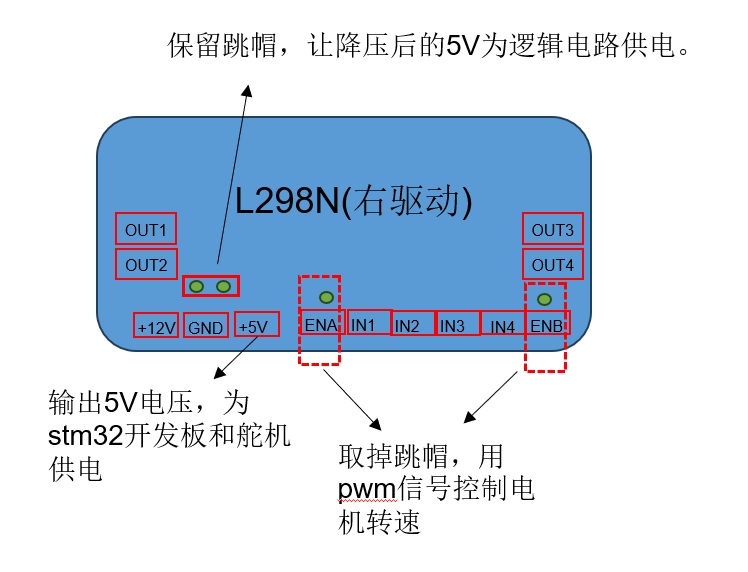
总接线示意图：



硬件说明：

1. L298N：

内部有降压功能，能将输入电压降低为5V为逻辑电路供电（跳帽接通时），并对外输出5V电压，直接将这输出的5V电压接给stm32开发板和舵机，无需再额外准备降压模块。



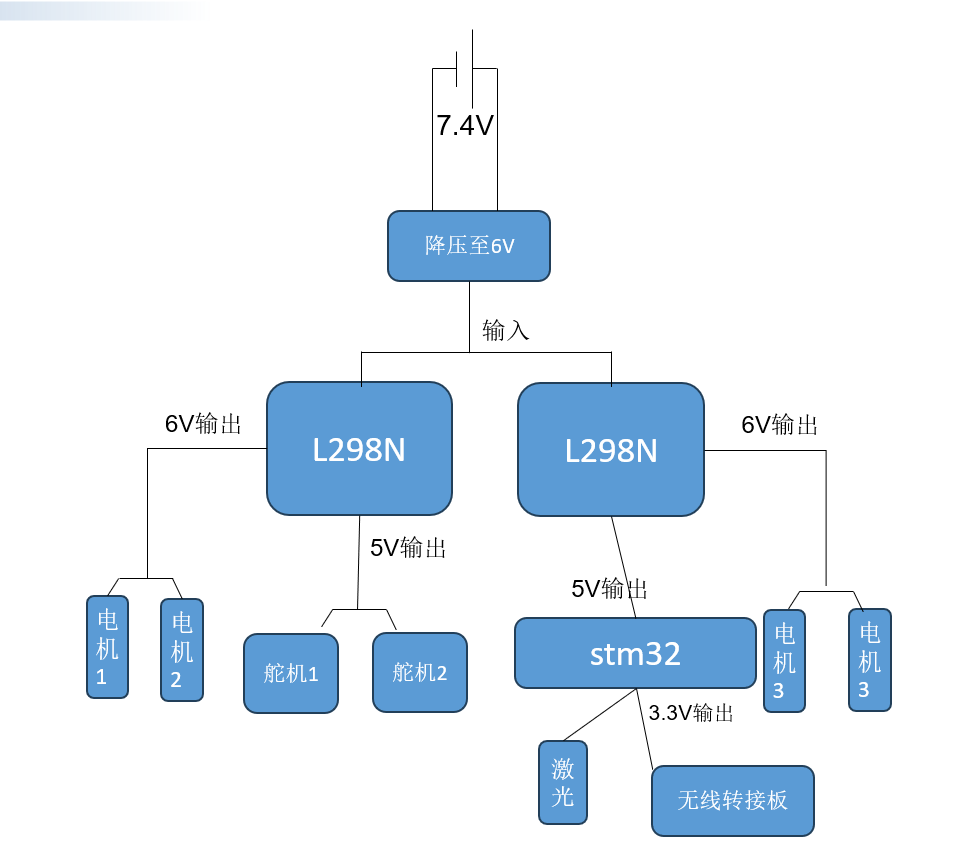
1. 分线板、面包板：

不用焊接，有利于接线、调试与修改

1. 降压模块：

将18650电池组的7.4V电压（万用表实测>7.4V）降低为6V，来接入电机驱动模块，以保护直流减速电机（3V~6V）

1. 供电说明：



电池所能承受的最大电流为6A，功率为45W左右。

单个电机6v时电流为0.2A,激光3.3V时电流<35mA，单个SG90 180度舵运行电流大约<100mA,开发板运行电流<100mA，无线转接板运行电流约为30mA,总电流小于6A，供电合适

主要代码：

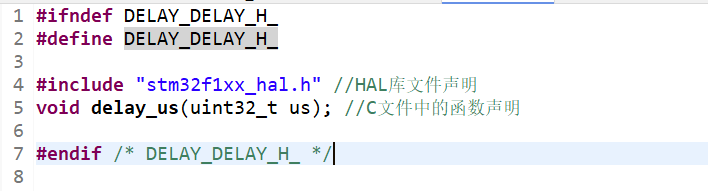
1. PS2驱动代码：

根据PS2手柄通信协议（基于 SPI-like 的半双工同步串行协议），建构与PS2通信的驱动代码。

PS2 手柄通信的数据读取操作对时序要求较高，需要微秒级延时

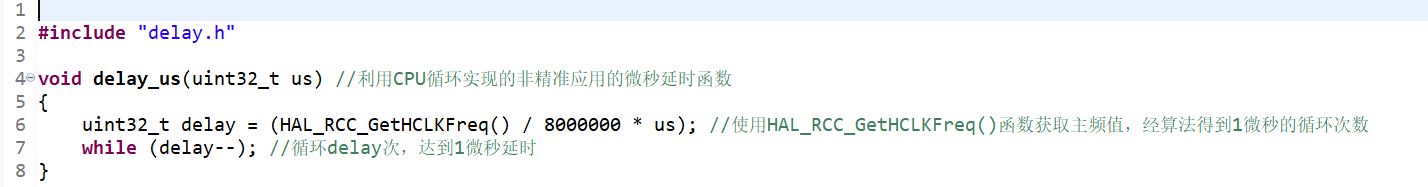
1. delay.h中:

声明微秒级延时函数 delay\_us()的函数原型，并引用 HAL 库，确保与系统时钟配置兼容



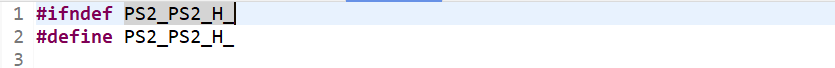
1. delay.c中:

定义（实现）微秒级延时函数 delay\_us()，利用 CPU 主频进行循环计数来实现us延时效果



1. ps2.h中:

a)防止重复引用同文件（delay.h中也一样）。#ifndef检查是否定义宏 PS2\_PS2\_H\_，若未定义，则执行后续代码并定义该宏，若已定义，则跳过整个文件内容（跳过#ifndef与#endif之间的内容），避免重复

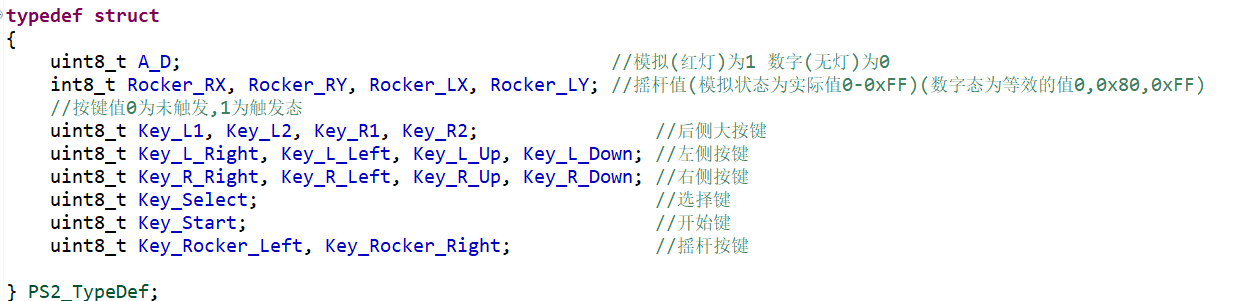




b)宏定义四个GPIO口，增加可读性的同时便于对代码的修改



c)设立一个结构体类型来存储PS2手柄的相关变量，别名为PS2\_TypeDef



d)使用 extern 声明全局变量，声明一个类型为PS2\_TypeDef的结构体PS2\_Data,用于存储PS2手柄的相关变量



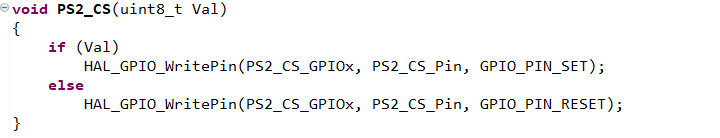
e)先声明**PS2\_Read\_Data函数的函数原型**



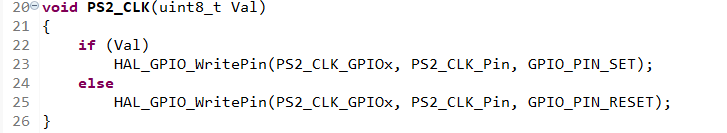
1. ps2.c中:
2. 定义全局变量并进行初始化



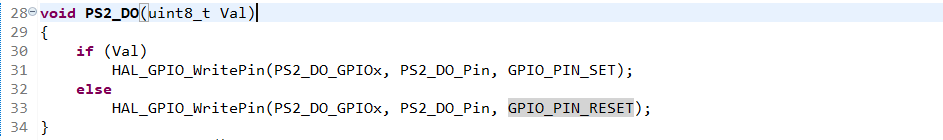
1. 定义片选信号控制函数PS2\_CS（），用来控制与PS2设备通信的时序和片选信号，拉低CS时启动通信，拉高结束通信，



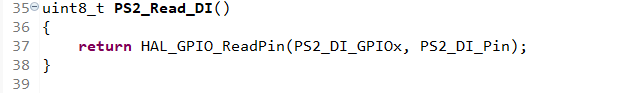
1. 定义时钟线控制函数PS2\_CLK（），用于手动模拟时钟信号，控制数据同步



1. 定义数据输出函数PS2\_DO（），主机通过 DO 线向手柄发送命令



1. 定义数据输入函数PS2\_IN（），通过读取 DI 引脚电平来获取手柄返回的数据位

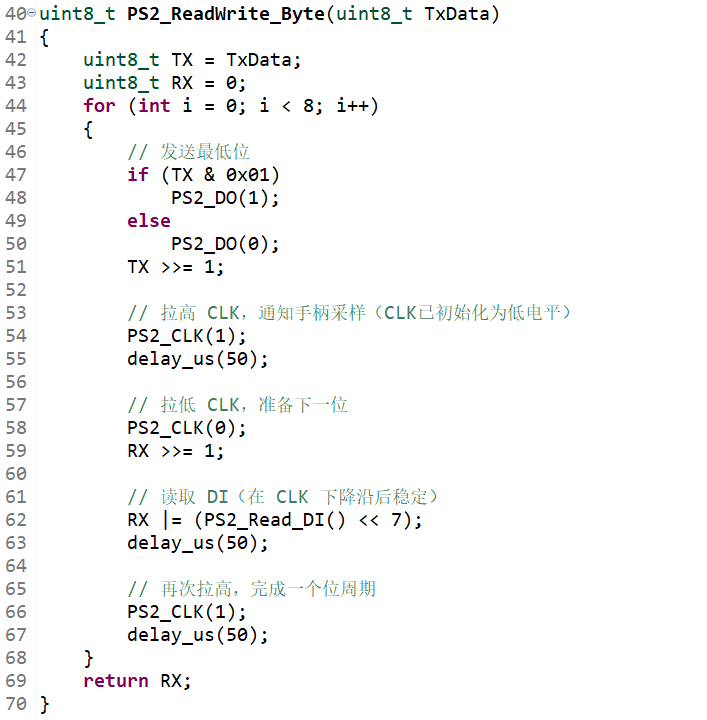


1. 定义核心通信函数PS2\_ReadWrite\_Byte（），向 PS2 手柄发送 1 个字节（8 位）命令，同时接收手柄返回的 1 个字节数据，通过位交错实现“同时”通信。

🡪循环拉高拉低CLK电平和延时，为PS2手柄提供时钟

🡪 通过按位与、按位或、移位拼接来实现LSB 先传（数据从最低位开始发送/接收），进而实现位交错

🡪手柄在 CLK 上升沿采样 DO 线，在 CLK 下降沿之后更新 DI 线上的输出位（返回给主机的数据）



1. 根据PS2的通信协议规范，定义数据解析函数PS2\_Decode（）

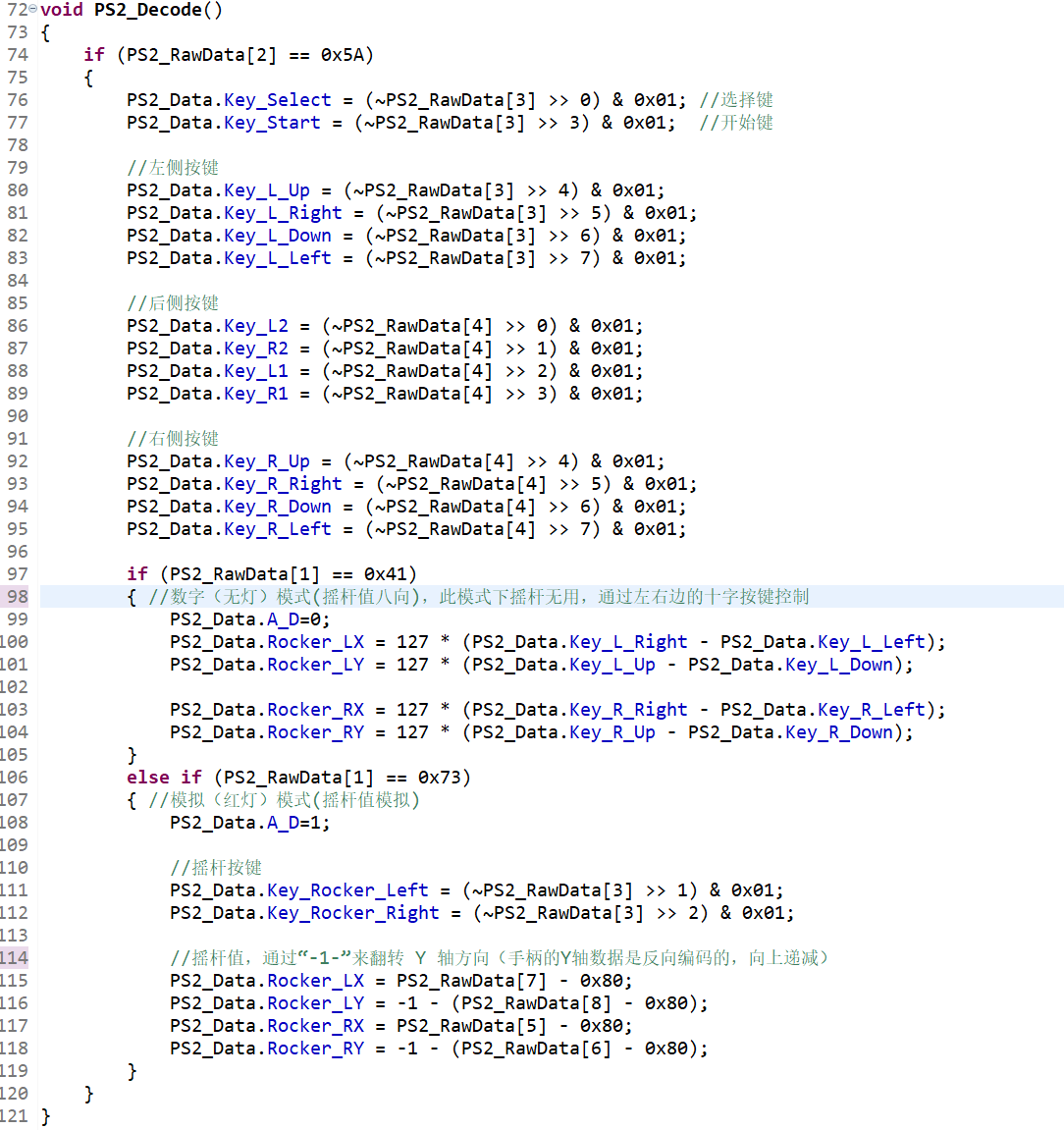
🡪0x5A表示“数据就绪”,根据 PS2\_RawData[2] == 0x5A 判断是否为有效数据包。

🡪PS2低电平表示“按下”（返回0），通过按位取反、按位与和位移操作获取所有按键(除了模式mode键)的状态，因为预设了按键值为1时才表示触发态。

🡪根据PS2\_RawData[1]判断手柄的模式（0x41表示数字模式，0x73表示模拟模式），并通过A\_D按键标识手柄的模式状态：

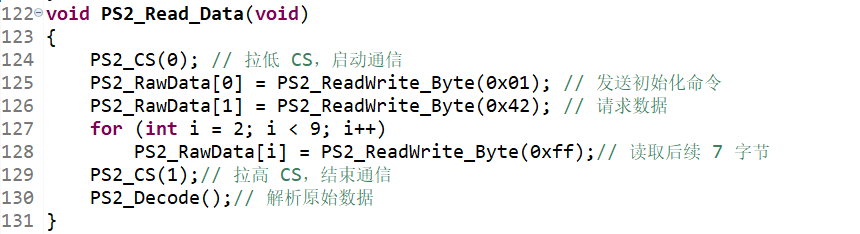
数字模式下摇杆无用，通过左右边的十字按键控制摇杆变量的状态值；

模拟模式下，通过按位取反、按位与和位移操作获取摇杆按键的状态。而摇杆提供 的是256 级模拟量（0~255），需减去中心值0x80（128）来转换为有符号数 [-128, 127]。这里R\_Y和L\_Y加上了“-1-”，是因为PS2手柄的 Y 轴数据是“反向编码”的，向上的值减小。为了实现从左到右、从下往上是数值递增方向，需要通过“-1-”来翻转 Y 轴方向。



1. 集成前面的函数，定义主读取函数PS2\_Read\_Data（），通过CS触发线启动和结束通信

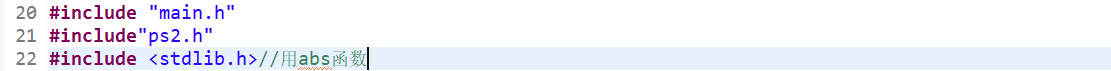
🡪通过0x01初始化通信，0x42请求当前手柄状态（PS2 手柄通信协议命令），接着读取数据，并在通信结束后解析数据



控制代码：

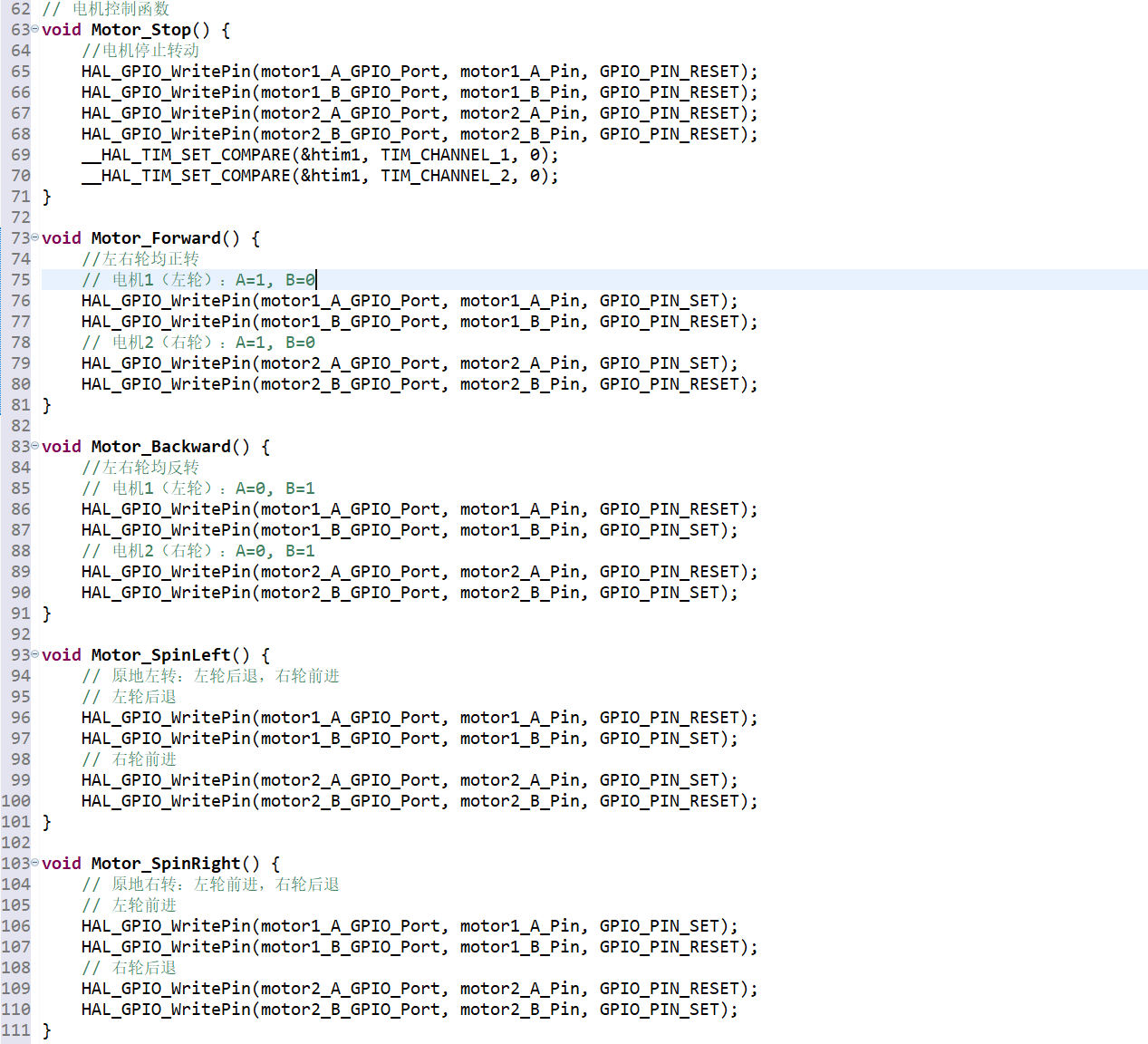
main.c中：

(1)先引入ps2.h和stdlib.h头文件



(2)在main主函数前，先写下驱动电机的函数。由于使用的是L298N电机驱动模块，控制电机的两个GPIO口一个输出高电平，一个输出低电平时正/反转，输出相同电平时不转（此处用同低），而通过设置PWM信号的比较寄存器的值可控制电机转速（由于采用了PWM mode1，计数器<比较寄存器时为高电平，比较寄存器越大，占空比越大，转速越大）。

通过改写GPIO口输出电平的函数即可实现，电机停止转动函数中设置两边的比较寄存器为0可删无影响，留下能体现停止转动（转速为0）



(3)在main主函数前，再写下数据处理和外设控制的函数**PS2\_Control\_Handler**的定义: **int** **PS2\_Control\_Handler**(){……}

设定该函数有返回值的原因是，用返回值来进行部分的条件判断，可以简化代码

a)本次工程只计划采用PS2的模拟模式，不用数字模式，故将摇控开关与模式转换按钮（按下实现模拟模式和数字模式的切换）结合在一起，切换为数字模式时无操作，兼摇控关闭功能，不额外写摇控开关按钮的代码。

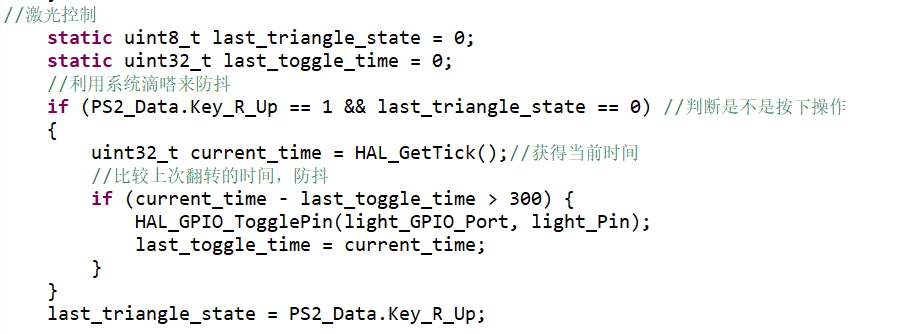


b)激光控制：

🡪用手柄右侧的按键R\_Up来控制激光的发射

🡪直接将激光模块的正极连相应的GPIO口，GPIO口输出高电平时启动激光

🡪通过HAL\_GetTick()函数获得系统启动以来的毫秒数，进而利用系统滴答来实现按键防抖(规定两次翻转的时间>300ms时)，因为用delay延时会干扰摇控的流畅性，实际测试中不进行防抖会出现按键失灵的状况

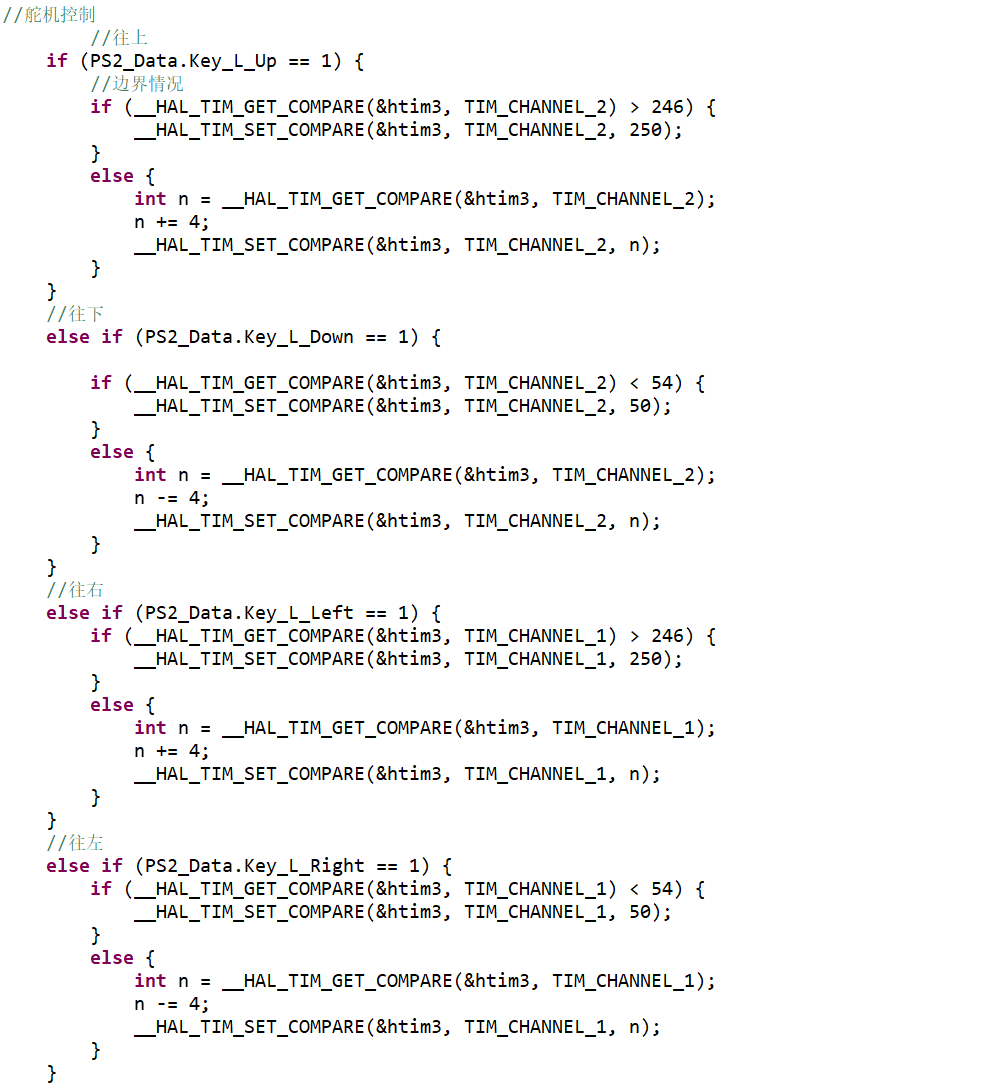


c)舵机控制：

🡪通过设置PWM信号的占空比，调节舵机旋转的角度，00~1800对应占空比2.5%~12.5%，而PWM信号的自动重装载值为2000，则对应的比较寄存器的值为50~250。则在这个范围内调节比较寄存器的值，进而控制舵机旋转

🡪用4为单位来控制旋转，既提高了转动速度，也保证了较好的精度

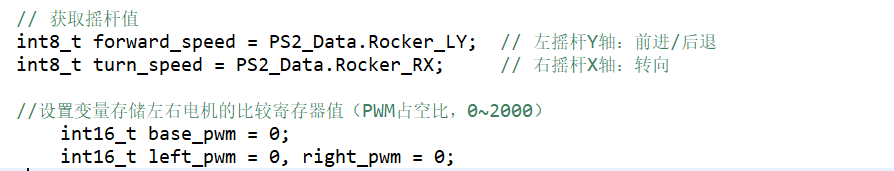
🡪通过左侧的十字按键控制二自由度舵机云台的旋转（此处默认在四个按键中同一时间只按一个，若同时按多个无效）



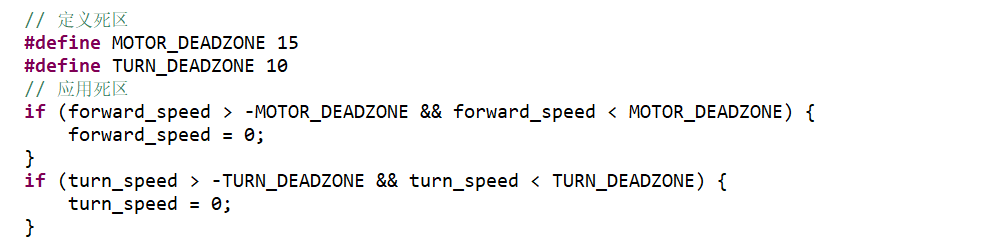
d)电机控制：运用前面的电机控制函数，并通过设置PWM信号的比较寄存器的值可控制电机转速（由于采用了PWM mode1，计数器<比较寄存器时为高电平，比较寄存器越大，占空比越大，转速越大）

🡪另设等价变量来存储摇杆值，增加代码的可读性的同时，便于后续代码的运用

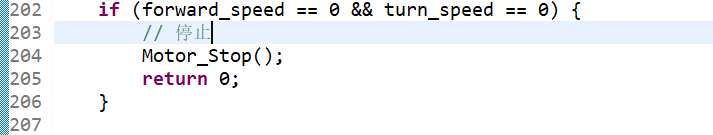
🡪设置变量来分别存储左右电机的比较寄存器的值



🡪定义并应用死区，更严谨，主要体现在：防止在按摇杆自身按键时误操作，这次未用到，但写了可便于以后的使用



🡪先判断是否是停止状态，可在无操作时结束运行（电机控制为整体控制函数的最后处理层）



🡪前进/后退控制（前后速度不为0时，左摇杆发生>10的前后偏移）：

🡪先通过比例计算前进的速度大小，取决绝对值，需处理-128的情况，避免出现比较寄存器的值为负数的情况

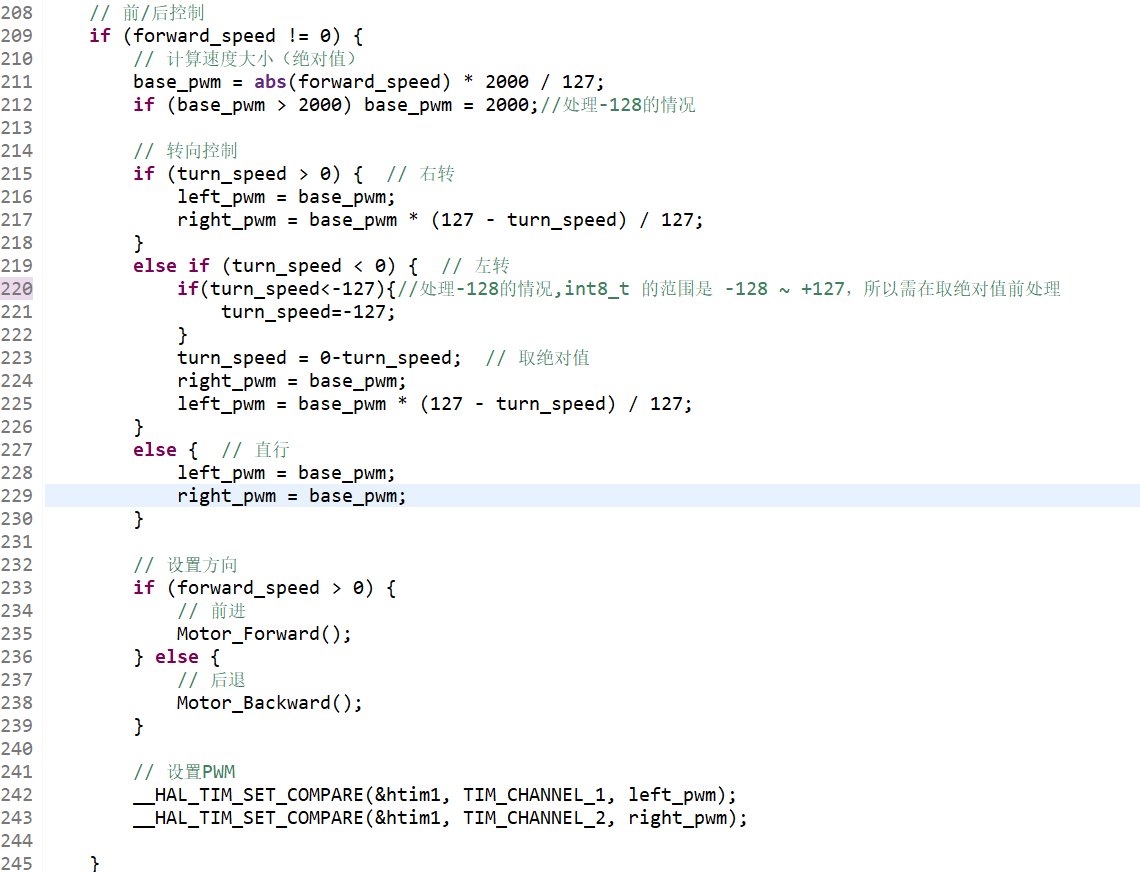
🡪再判断是否有转向，前进时有转向则对左右轮进行差速处理，右转时

右轮减速（根据摇杆值进行比例计算，按转动的幅度递减），左转时左轮减速，否则直行（左右轮速度一样）（前进和后退的情况一致）。

🡪 int8\_t 的范围是 -128 ~ +127，所以在前进时左转判定时，需在取绝对值前处理-128的情况（提前取绝对值得到128会溢出卡bug）

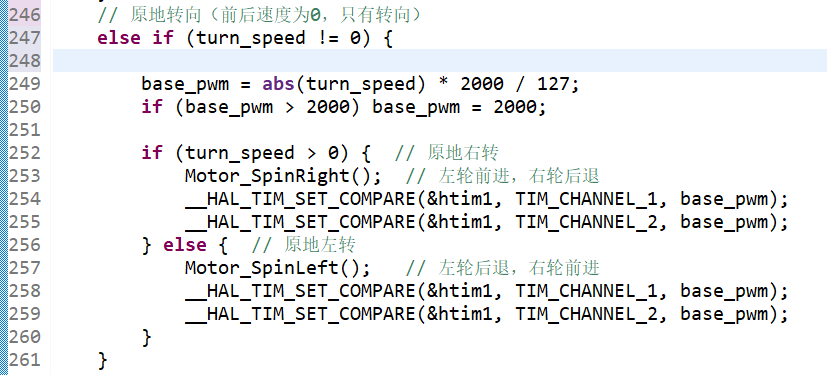
🡪最后再设置方向，设置左右轮PWM信号的比较寄存器的值（占空

比）

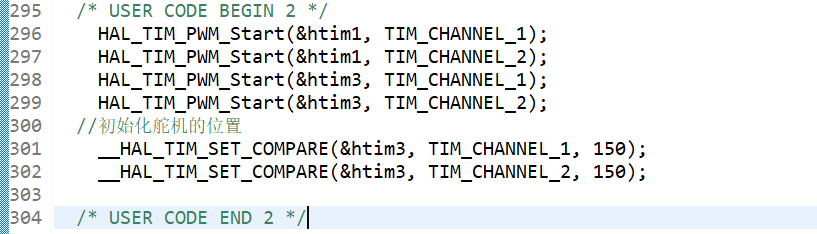


🡪原地转向：采用一边前进，一边后退，速度大小一致的方案，

原地右转时左轮前进，右轮后退，原地左转时左轮后退，右轮前进。这样可以实现快速旋转



(4)main主函数中，先启动四个PWM模式的定时器，并初始化两个舵机的角度，便于后续的控制和操作



while死循环中，通过使用前面的相应函数来运行（更简洁，且便于修改），运行逻辑为读取数据，处理数据并控制外设。然后再加个延时来控制循环效率，避免程序循环过快导致系统负担过重，同时还能让外设的控制更加稳定和流畅



测试：（展示视频在另一个文件夹）

成品图如下：

