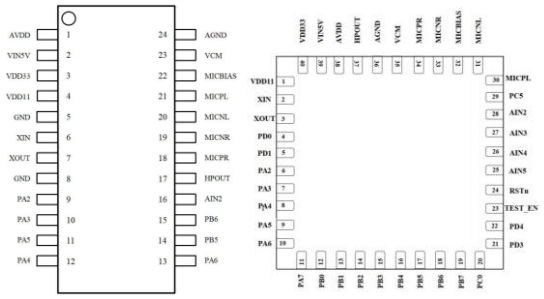
ASRPRO快速上手（专业模式）

一、概述

芯片概述

本产品是针对低成本离线语音应用方案开发的一款通用、便携、低功耗高性能的语音识别芯片，采用了第三代语音识别技术，能支持 DNN\TDNN\RNN 等神经网络及卷积运算，支持语音识别、声纹识别、语音增强、语音检测等功能，具备强劲的回声消除和环境噪声抑制能力，语音识别效果优于其它语音芯片。该芯片方案还支持汉语、英语、日语等多种全球语言，可广泛应用于家电、照明、玩具、可穿戴设备、工业、汽车等产品领域，搭配天问Block图形化编程软件，快速实现语音交互及控制和各类智能语音方案应用。



天问提供SSOP24和QFN40两种封装类型和2M、4M两种Flash容量类型。具体参数请查看对应的规格书。

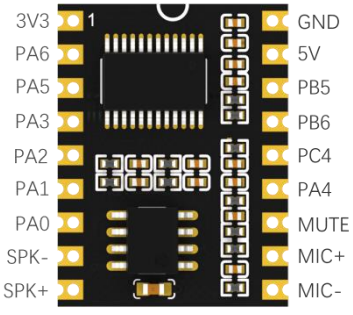
芯片特点

支持离线神经网络计算，支持单麦克风降噪增强，单麦克风回声消除，360度全方位拾音，可抑制环境噪音，保证嘈杂环境中语音识别的准确性。进行离线语音识别不依赖网络，时延小，性能高，可实现98%以上的高识别率，10米超远距离识别，响应时间小于0.1S。

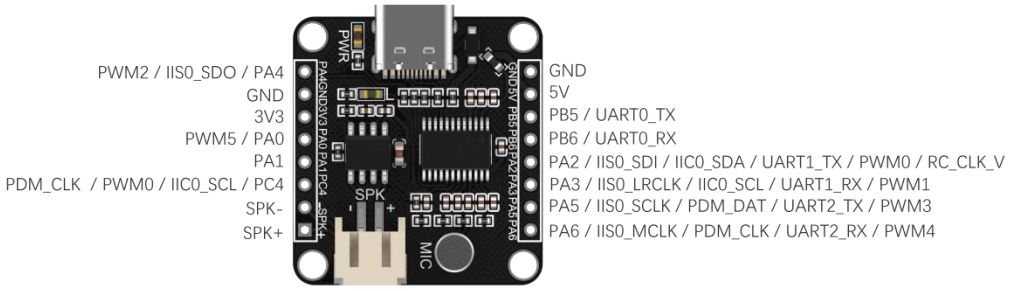
硬件概述

天问基于ASRPRO芯片目前推出了5种类型，供开发者选择。

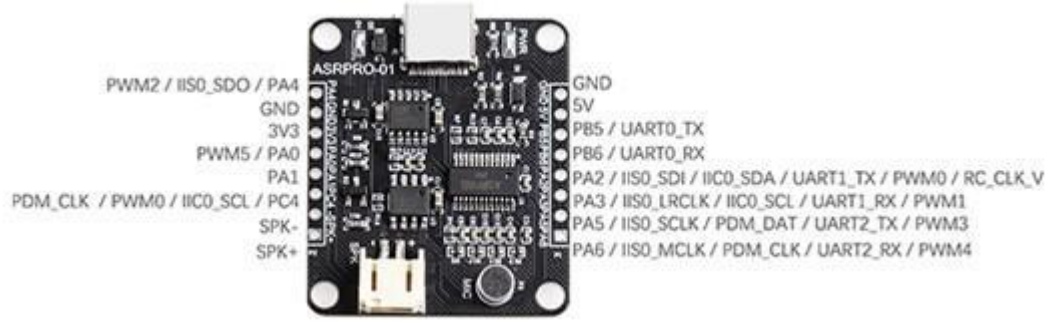
1. ASRPRO-CORE 核心板，模块体积小巧，长宽为 18x23mm，对外接口采用 2 排邮票孔和插针孔，方便采用回流贴片使用和焊接插针使用，喇叭和麦克风都需要自己外接，下载程序需要搭配 STC-LINK 下载器。



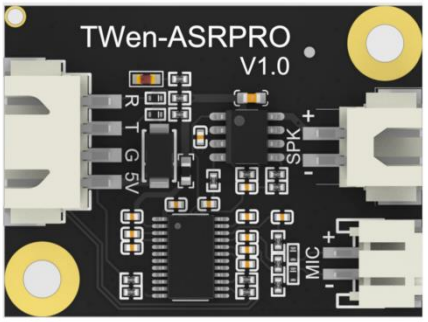
2. ASRPRO 基础开发板，长宽为 30x28mm，板载麦克风、指示灯，用户只需要外接喇叭就可以使用，下载程序需要搭配 STC-LINK 下载器。



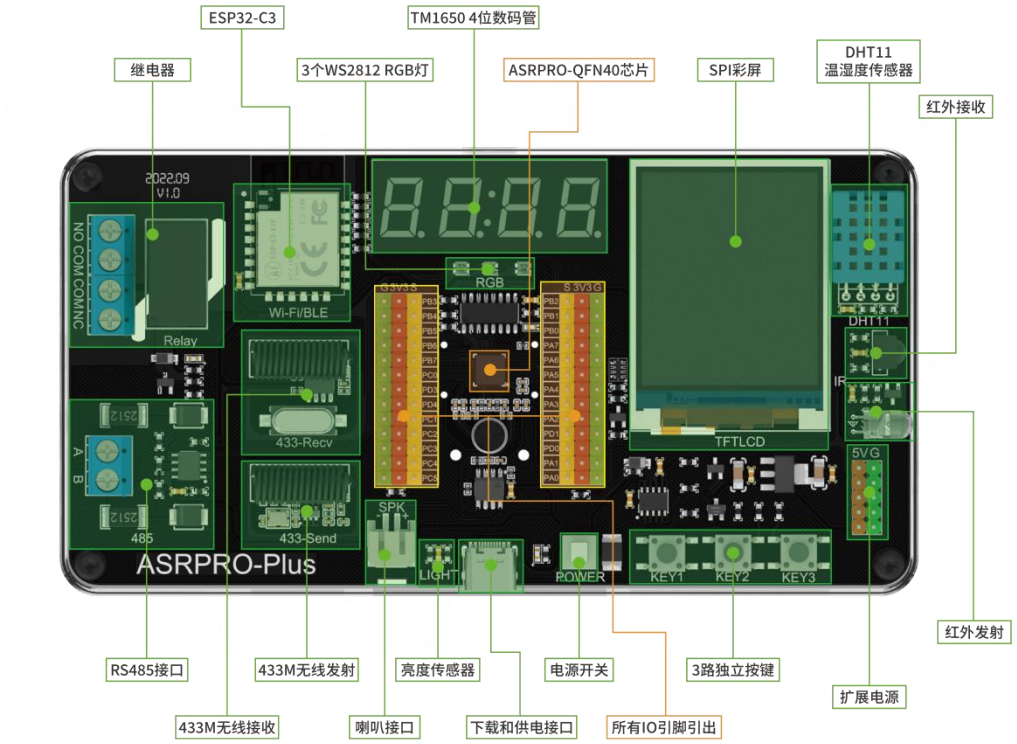
3. 鹿小班 ASRPRO 基础开发板，在 ASRPRO 基础开发板的基础上额外集成了串口下载芯片 CH340K，一根 Type-C 线就可以下载程序，并且开发板上有自动断电电路可以实现一键下载，不需要额外的 STC-LINK 下载器。



4. ASRPRO 串口模块，只引出了串口、喇叭、麦克风供用户和其它主控搭配使用。



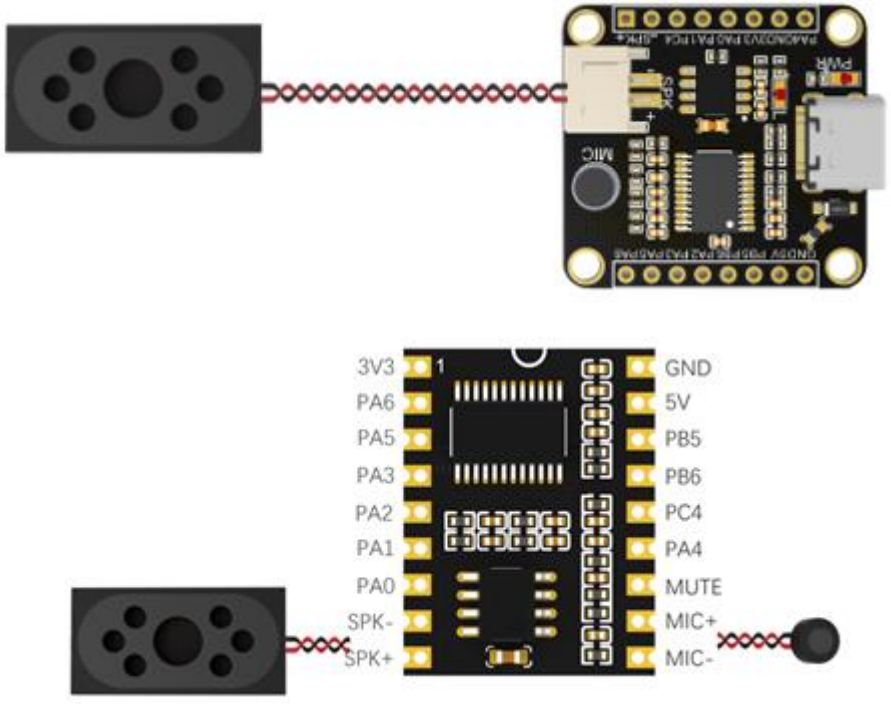
5. ASRPRO-Plus 开发板是一款带语音识别的物联网开发板，基于 32 位 RISC-V 内核，内置神经网络处理器，支持 DNN\TDN-N\RNN 等神经网络及卷积运算，支持语音识别、声纹识别、语音增强、语音检测等功能，具备强劲的回声消除和环境噪声抑制能力。板载 RS485、433M 无线收发、红外收发、ESP32-C3(2.4GHz Wi-Fi 和 Bluetooth 5LE)5 种通讯方式；SPI 彩屏、数码管、RGB 灯 3 种显示模块；光敏传感器、DHT11 温湿度传感器 2 种常用传感器；1 路继电器输出模块。搭配天问 Block 图形化编程软件，快速实现语音交互及控制和各类智能语音物联网方案应用。



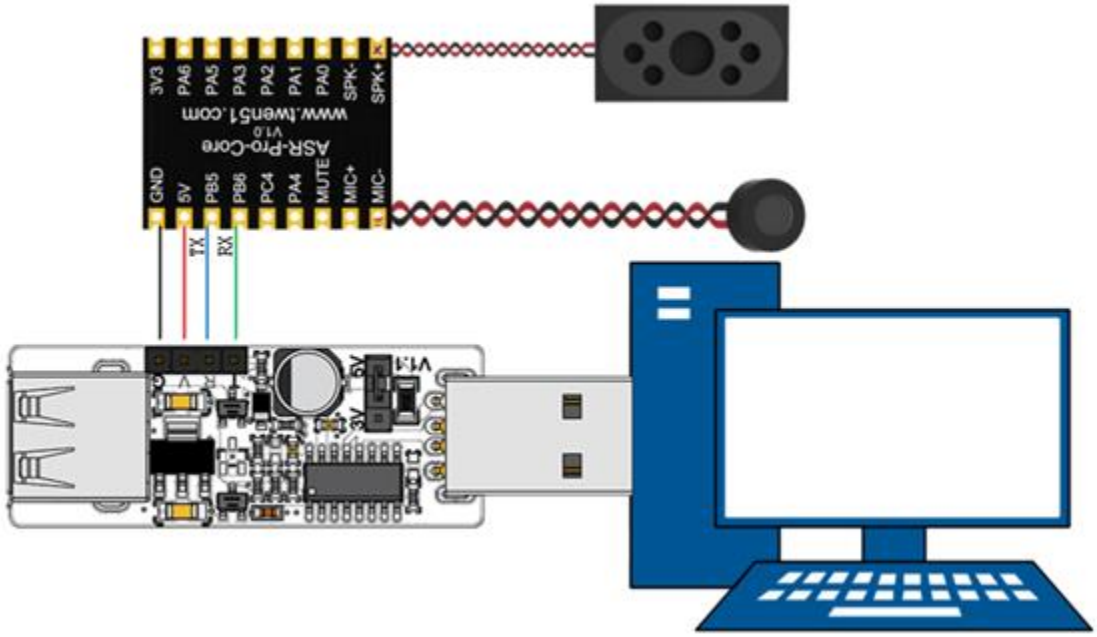
二、开机测试

ASRPRO核心板和基础版开机测试

第一步：开发板需要连接喇叭、核心板需要连接喇叭和咪头



第二步：用STC-Link连接电脑，再使用Type-C数据线将开发板与其连接。

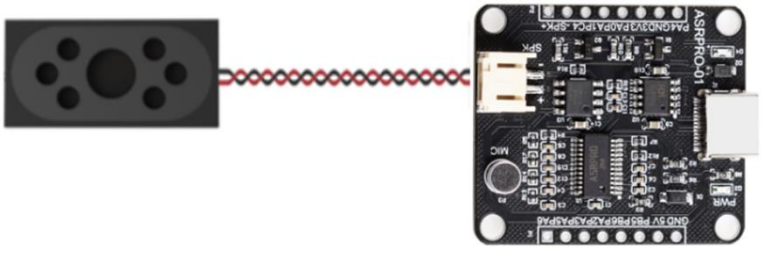


第三步：上电后，会播报欢迎词“欢迎使用智能管家，用智能管家唤醒我”，接下来你可以用“智能管家”唤醒词唤醒设备，接着用不同的命令词控制设备，详见下表。

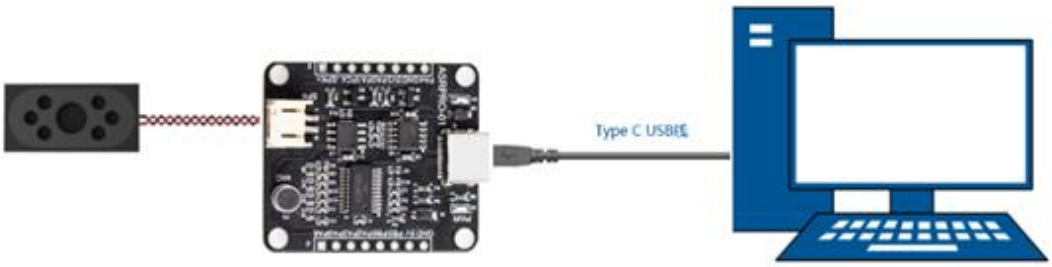
类型	识别词	回复语音
欢迎词	欢迎使用智能管家，用智能管家唤醒我	
退出语音	我退下了，用智能管家唤醒我	
唤醒词	智能管家	我在
命令词	打开灯光	好的，灯光已打开
命令词	关闭灯光	好的，灯光已关闭

ASRPRO鹿小班基础版开机测试

第一步：开发板需要外接喇叭，喇叭为PH2.0接口。下图为开发板实物图



第二步：开发板板载USB转TTL芯片，只需要一根Type-C线就可以实现一键下载。



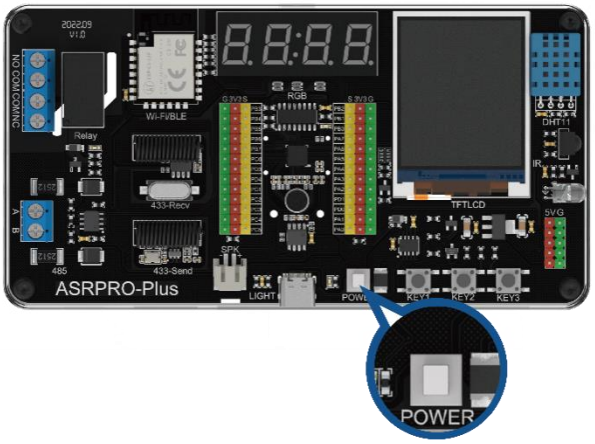
第三步：上电后，会播报欢迎词“欢迎使用智能管家，用智能管家唤醒我”，接下来你可以用“智能管家”唤醒词唤醒设备，接着用不同的命令词控制设备，详见下表。

类型	识别词	回复语音
欢迎词	欢迎使用智能管家，用智能管家唤醒我	
退出语音	我退下了，用智能管家唤醒我	
唤醒词	智能管家	我在
命令词	打开灯光	好的，灯光已打开
命令词	关闭灯光	好的，灯光已关闭

ASRPRO-Plus开机测试

第一步：使用Type-C数据线将开发板连接到电脑上

第二步：打开开发板上的电源开关POWER，此时旁边指示灯亮起



第三步：上电后，会播报欢迎词“欢迎使用语音助手，用天问五幺唤醒我”，接下来你可以用“天问五幺”唤醒词唤醒设备，接着用不同的命令词控制设备，详见下表。由于需要联网功能，所以请把热点的名称和密码修改为：名称：Twen，密码：12345678

类型	识别词	回复语音	备注
唤醒词	天问五幺	我在	
命令词	打开灯光	好的，马上打开灯光	打开板载RGB灯、继电器、发送无线开关命令1
命令词	关闭灯光	好的，马上关闭灯光	关闭板载RGB灯、继电器、发送无线开关命令2
命令词	当前天气	播报当前城市天气情况	需要ESP32模块联网，默认热点名称：Twen，密码：12345678
命令词	当前时间	播报当前网络时间	需要ESP32模块联网，默认热点名称：Twen，密码：12345678
命令词	当前温度	播报板载DHT11温度	需要完全断电复位
命令词	当前湿度	播报板载DHT11湿度	需要完全断电复位
命令词	当前亮度	播报板载光敏值	
命令词	打开电视	小米电视已打开	红外发送小米电视电源键命令
命令词	关闭电视	小米电视已关闭	红外发送小米电视电源键命令
命令词	打开一号继电器	马上执行	485控制的4路继电器模块（MODBUS协议）
命令词	关闭一号继电器	马上执行	
命令词	打开二号继电器	马上执行	
命令词	关闭二号继电器	马上执行	
命令词	打开三号继电器	马上执行	
命令词	关闭三号继电器	马上执行	
命令词	打开四号继电器	马上执行	
命令词	关闭四号继电器	马上执行	
命令词	打开所有继电器	马上执行	
命令词	关闭所有继电器	马上执行	

第四步：板载按键控制说明如下表

KEY1	KEY2	KEY3
控制板载继电器打开	控制板载继电器关闭	控制彩屏背光

三、下载与安装天问Block软件

下载软件

1.浏览器打开天问官方网站 http://twen51.com/。

2.点击天问Block 下载

安装软件

根据提示默认安装，注意安装过程中，根据提示安装CH340驱动。

四、运行天问Block软件

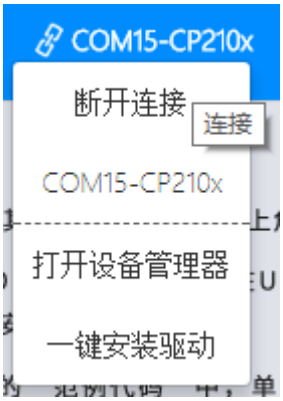
选择主板

第一次打开软件，会让你选择主板，请选择ASRPRO

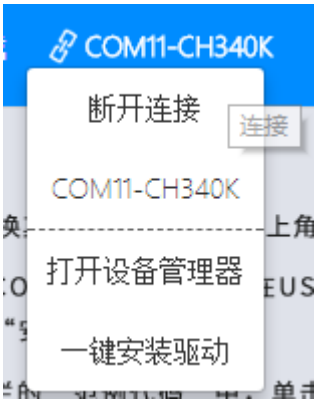
ASRPRO

检查串口连接

检查串口是否连接成功，如果未显示驱动可以一键安装驱动。



使用STC-LINK下载器



使用板载CH340下载芯片

选择开发模式

本教程是专业模式教程，因此需要切换成专业模式，此时会出现字符编程模式

界面说明

在标准模式下，页面总共分为4个部分，工具栏、指令区、编程区以及字符代码区

工具栏：有最基本的文件操作、撤消、重做图标，还可直接打开范例代码进行编译下载，还有串口监视器、生成模型、编译下载等图标，每个图标对应操作的一个功能。还可进行登录个人账号，云保存程序等操作。在更多中还可查看编程手册、原理图、学习视频、设置等功能。

指令区：是程序指令仓库，需要编程时把指令拖动到编程区，实现编程的目的。指令区根据指令功能可以分成单片机配置模块、C语言程序模块、扩展模块三类。

配置模块有GPIO模块、PWM模块、ADC模块、定时器模块、串口模块、读写寄存器、多线程模块、物联网模块等模块，运用这些模块就可以设置所有功能，无需手册就能完成配置和读写寄存器，只要读懂指令模块就可以简单方便实现各种功能。

C 语言程序模块有控制、数学与逻辑、文本与数组、变量、函数五个模块，运用这些模块就能实现程序结构、数据类型、变量设置、函数调用等功能，无需记忆 C 语言语句就能完 成基本程序编程。

扩展模块有无线遥控、DHTXX、MODBUS、舵机、DS18B20、WS2812、STT7735、SSD1306 、 TM1637 、TM1650 、IRSendRev、无线接收这些模块是单片机基础模块的扩展，可以实现各 类设备器件的图形化编程。

编程区是指令模块通过积木式编程实现程序的区域， 标准模式和专业模式中初次打开状 态里面有“初始化”模块。程序上电后， 先运行“初始化”模块中的指令，“初始化”模块中的程序 只运行一次，一般是进行单片机模块初始化、配置使用。

字符代码区有两种模式：图形化编程模式和字符编程模式。图形化编程模式时字符代码区不可编辑，代码由图形化模块编程自动生成；字符编程模式，字符代码区由用户输入编程字符，注意手动输入的字符不能自动生成图形模块，保存时只能保存字符模式。

五、运行程序

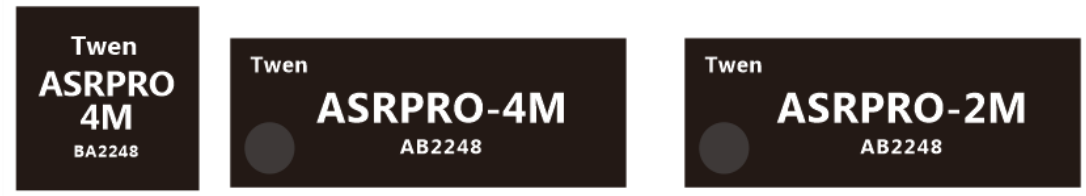
打开范例程序

打开范例代码1.智能语音对话，在跳出的对话框“是否导入并覆盖模型文件”选择确定

设置编译下载模式

ASRPRO开发板和核心板默认采用ASRPRO 2M的芯片，即芯片FLASH容量是2M，具体可查看开发板上芯片标注，需选择2M下载模式。

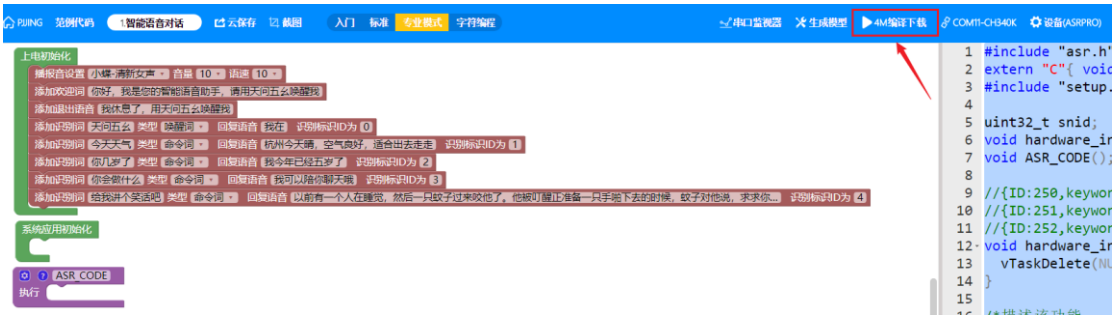
ASRPRO-Plus采用ASRPRO 4M的芯片，即芯片FLASH容量是4M，可以选择4M下载模式也可选择2M下载模式，当程序比较大时，需选择4M下载模式。



在更多-设置-编译模式中进行2M编译下载和4M编译下载切换，本教程主要以ASRPRO-Plus展开讲解，所以选择4M编译下载。

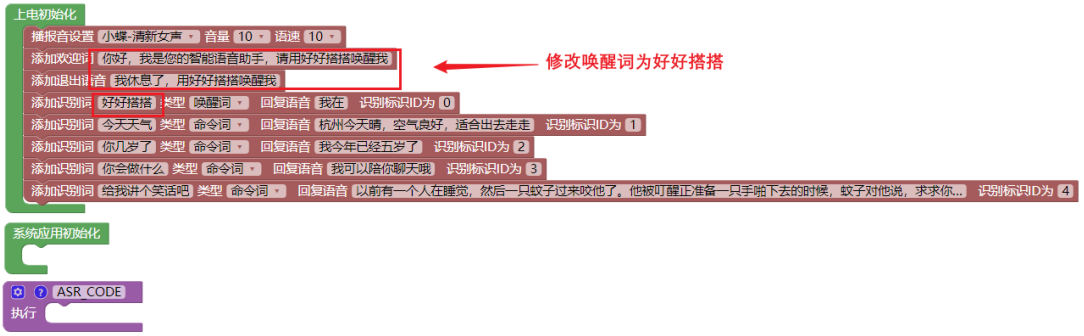
编译下载范例程序

范例代码都已经生成模型，所以直接点击编译下载即可。



修改程序

修改程序，如修改唤醒词为好好搭搭



当修改了语音相关设置时，需要重新生成模型

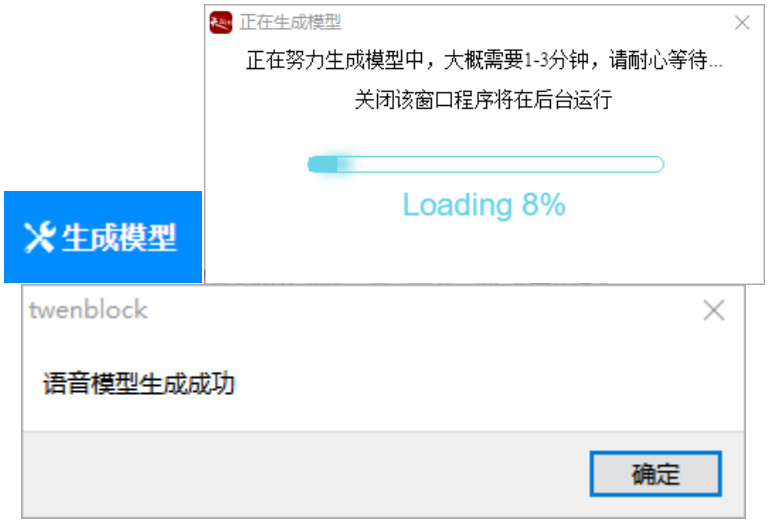
登录账号与实名认证

在使用生成模型功能时，需要登录账号（没有账号可进行免费注册）并进行实名认证

在更多中点击实名认证，输入手机号码以及验证码即可实名成功，注意一个号码只能绑定一个账号

生成模型与编译下载

点击生成模型

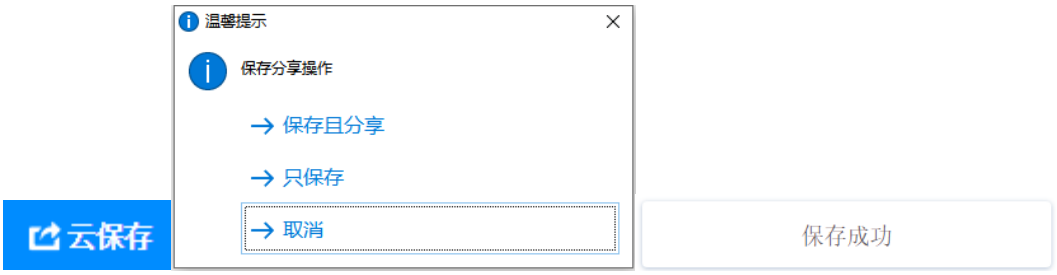


点击编译下载

六、云保存与本地保存

云保存

在登录账号的状态下点击工具栏的云保存，根据需要选择保存分享操作。



在菜单栏-项目-项目中心-我的项目中，即可查看到刚才云保存的项目，可随时打开

本地保存

1.在菜单栏-项目-保存（图形文件），选择路径进行保存，注意保存的文件下次打开编译下载前需要重新生成模型

2.在菜单栏-项目-项目保存（含模型），选择路径进行保存，保存的文件下次打开可以直接编译下载

本教程后续范例以ASRPRO-Plus开发板为例展开学习，其他开发板修改相应的IO口可以完成相同的任务。

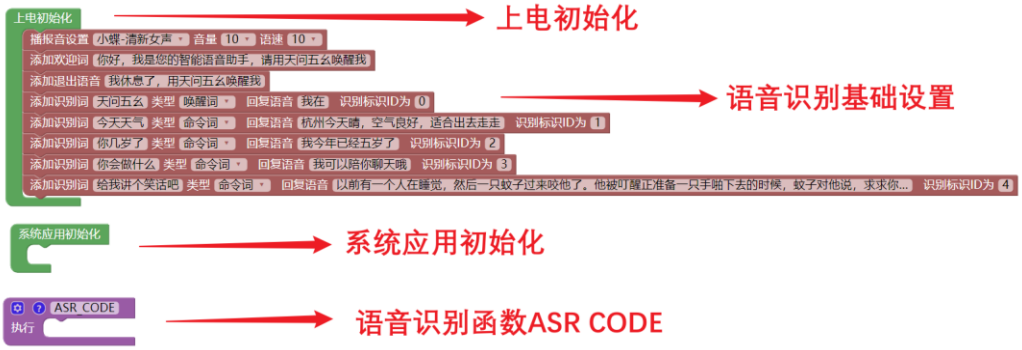
语音识别基础设置

范例1.1 智能语音对话

一、范例功能

本范例通过完成一段基于离线语音识别的人机对话，实现智能语音对话功能，达到在专业模式下进行自定义语音与设置程序结构的目的。本范例适合所有ASRPRO开发板。

二、范例分析



三、范例详解

在专业模式下，用户可以通过多线程实现语音识别控制多任务同时运行，支持字符编程，满足专业开发者的需求。用户既可以选择入门模式和标准模式的指令进行快速使用，也可以使用字符编程，随心所欲地进行创作。

在专业模式下，程序主要分为上电初始化、系统应用初始化、语音识别函数ASR_CODE、多线程、定时器、中断等多个部分。

我们首先来看一下本范例的程序结构。这个程序有三大部分，分别是上电初始化、系统应用初始化以及函数ASR_CODE。这三者在绝大多数的程序中都需要出现。

其中上电初始化指的是ASR PRO连接上电源后进行的初始化，所有的语音设置、GPIO口设置等等都放在上电初始化中。对应右边的代码是setup这个函数，其中语音部分均被注释，主要用于生成语音模型。

```
void setup()
{
  //{speak:小蝶-清新女声,vol:10,speed:10,platform:haohaodada}
  //{playid:10001,voice:你好，我是您的智能语音助手，请用天问五幺唤醒我}
  //{playid:10002,voice:我休息了，用天问五幺唤醒我}
  //{ID:0,keyword:"唤醒词",ASR:"天问五幺",ASRTO:"我在"}
  //{ID:1,keyword:"命令词",ASR:"今天天气",ASRTO:"杭州今天晴，空气良好，适合出去走走"}
  //{ID:2,keyword:"命令词",ASR:"你几岁了",ASRTO:"我今年已经五岁了"}
  //{ID:3,keyword:"命令词",ASR:"你会做什么",ASRTO:"我可以陪你聊天哦"}
  //{ID:4,keyword:"命令词",ASR:"给我讲个笑话吧",ASRTO:"以前有一个人在睡觉，然后一只蚊子过来咬他了。他被叮醒正准备一只手啪下去的时候，蚊子对他说，求求你别杀我，今天是我的生日。那个人听说后，小心翼翼把蚊子放在手心，一边拍手一边唱生日快乐歌！"}
  xTaskCreate(hardware_init,"hardware_init",256,NULL,100,NULL);
}
```

系统应用初始化中，一般放置扩展库中的一些初始化设置指令。这些会在扩展库使用中详细说明，这里不做详解。

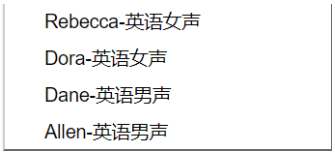
函数ASR_CODE是语音识别事件的一个回调函数。其它的外设操作都在其他线程里去完成，线程之间通讯用消息传递。所有的ID识别判断都在ASR_CODE函数里完成。

接下来我们再来学习上电初始化中的语音设置。语音基础设置分为以下几个部分，这些指令都在语音识别类别中：

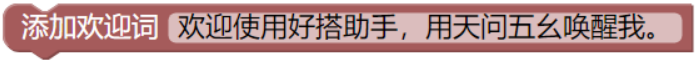
1.播报音设置

这里可以设置播报音的音色、音量和语速。音色就是谁的声音说话；音量2-20，改变的是文件的音量；语速2-20，改变的是说话的快慢。

当然ASRPRO还支持英文语音识别。此时需要注意，播报英文时要设置成英文的男声或者女声，同时不能和中文语音同时使用。



2. 欢迎词设置



欢迎词会在上电后进行播报。

3. 唤醒词设置

专业模式的唤醒词相比于其他模式，添加了识别标识ID。这个ID可以自由进行设置。在ASR_CODE函数中，我们通过对这个ID的判断可以进行语音控制。注意唤醒词和命令词的ID均不能相同。唤醒词可以同时设置多个，注意ID不能相同。

4.命令词设置

命令词的使用方法与唤醒词基本相同。

如果想要添加更多的的判断，可以如下所示添加一条新的命令词，识别标识ID

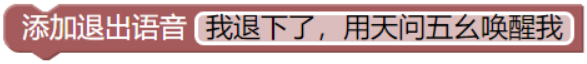
5.唤醒时间设置

唤醒时间一般由两种设置方法。

第一种为唤醒词唤醒，可以自由设置唤醒后的退出时间。例如设置时间为5s，那么唤醒后，5s内没有对其进行语音命令，则退出唤醒模式。这条指令一般不能放在上电初始化中，而需要放在语音播报的程序中。默认唤醒退出时间为15s。

第二种为永久唤醒模式。在此模式下，无需使用唤醒词进行唤醒，可以随时使用命令词进行控制。

6. 退出词设置



退出词会在默认的唤醒时间结束后进行播报。如果设置了永远唤醒，那么无需添加退出语音。

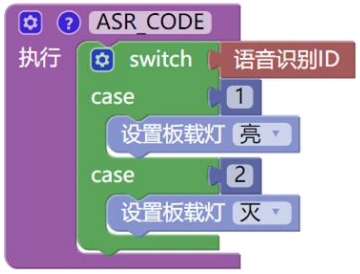
7. 退出时设置

这条指令在唤醒词唤醒模式下有效。可以在这条指令内部放入一些退出时想要做的指令。

8.音量设置

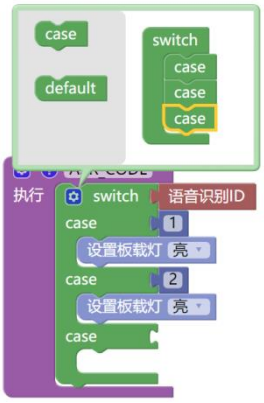
这条指令设置的音量是整体音量，不同于播报音中设置的音量。播报音设置的音量是文件本身的音量。使用这条指令可以设置整体音量，取值范围在1-7，1为最小音量，5为中等音量，7为最大音量。这个音量断电后也会保存，不会改变。

然后我们来学习语音识别的函数ASR_CODE。在这个函数中，我们可以对语音识别的ID进行判断。例如下图就对语音识别ID分别等于1和2进行判断，并根据判断情况设置了板载灯的亮灭。switch指令可以在控制类别指令中找到。语音识别ID指令则在语音识别类别指令中。查看右边的代码，我们可以发现，这个函数是对变量snid进行一个判断。




```
void ASR_CODE(){
  switch (snid) {
   case 1:
    digital_write(4,0);
    break;
   case 2:
    digital_write(4,1);
    break;
  }
```

点击switch旁边的小齿轮，将左边的case移到右边，就可以添加更多的case。



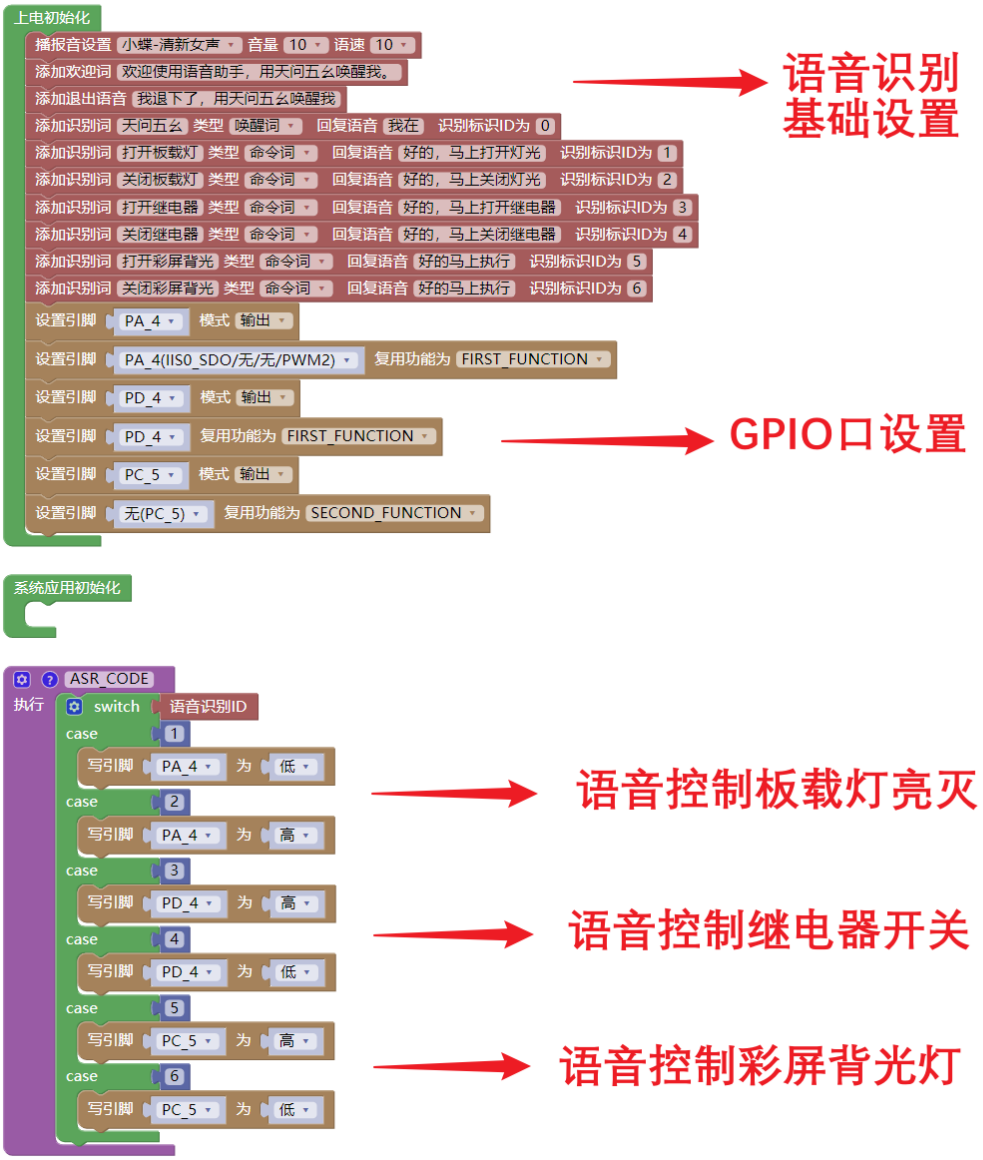
GPIO口设置

范例1.2 GPIO口输出设置

一、范例功能

本范例通过语音控制ASRPRO-Plus的板载灯PA_4、板载继电器PD_4和彩屏背光灯PC_5，实现语音控制GPIO口输出的功能，达成学习GPIO口输出设置的目的。本范例适配ASRPRO-Plus开发板，其它开发板需要修改PD_4和PC_5引脚。

二、范例分析

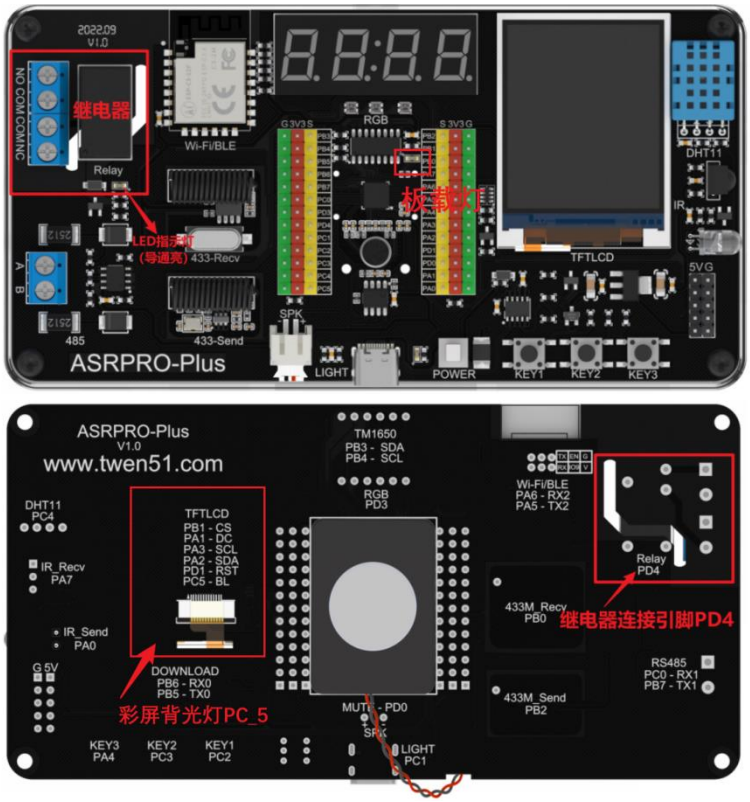


三、范例详解

GPIO口，是General-purpose input/output的缩写，指的是通用型输入输出口的简称。我们可以通过天问Block的程序编写自由控制这些引脚。

在本范例中，我们要对引脚为PA_4的板载灯、PD_4的继电器、PC_5的彩屏背光灯进行控制。

下方是ASRPRO-Plus的板载继电器、板载灯和彩屏背光灯的位置图。其中PD_4是继电器，PC_5是彩屏背光灯，PA_4是板载灯，在中间部分。

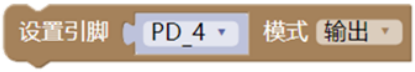


以引脚为PD_4的继电器为例，当我们使用继电器时，不能直接使用该引脚的高低电平的指令进行控制，而是需要先对该引脚进行设置。

首先我们需要使用下方的指令，设置该引脚是输出口还是输入口，以及引脚的复用功能。这些指令都在GPIO模块类别指令中。GPIO口引脚功能图如下所示。其中PA0、PA1引脚是晶振引脚，PC1、PC2、PC3、PC4的引脚默认是模拟引脚。

Pin Name	Function1	Function2	Function3	Function4	Function5	Analog Function	Specific Function
XIN	PA0	PWM5	-	-	-	XIN	-
XOUT	PA1	-	-	-	-	XOUT	-
PA2	PA2	IIS_SDI	IIC_SDA	UART1_TX	PWM0	-	-
PA3	PA3	IIS_LRCLK	IIC_SCL	UART1_RX	PWM1	-	-
PA4	PA4	IIS_SDO	-	-	PWM2	-	PG_EN
PA5	PA5	IIS_SCLK	PDM_DAT	UART2_TX	PWM3	-	-
PA6	PA6	IIS_MCLK	PDM_CLK	UART2_RX	PWM4	-	-
PA7	PA7	PWM0	UART1_TX	EXT_INT[0]	-	-	-
PB0	PB0	PWM1	UART1_RX	EXT_INT[1]	-	-	-
PB1	PB1	PWM2	UART2_TX	-	-	-	-
PB2	PB2	PWM3	UART2_RX	-	-	-	-
PB3	PB3	PWM4	IIC_SDA	-	-	-	-
PB4	PB4	PWM5	IIC_SCL	-	-	-	-
PB5	PB5	UART0_TX	IIC_SDA	PWM1	-	-	-
PB6	PB6	UART0_RX	IIC_SCL	PWM2	-	-	-
PB7	PB7	UART1_TX	IIC_SDA	PWM3	PDM_DAT	-	-
PC0	PC0	UART1_RX	IIC_SCL	PWM4	PDM_CLK	-	-
AIN5	PC1	-	UART2_TX	PWM3	PDM_DAT	AIN5	-
AIN4	PC2	-	UART2_RX	PWM2	PDM_CLK	AIN4	-
AIN3	PC3	-	IIC_SDA	PWM1	PDM_DAT	AIN3	-
AIN2	PC4	-	IIC_SCL	PWM0	PDM_CLK	AIN2	-
PC5	PC5	-	-	-	-	-	BOOT_SEL

1.引脚输出模式设置



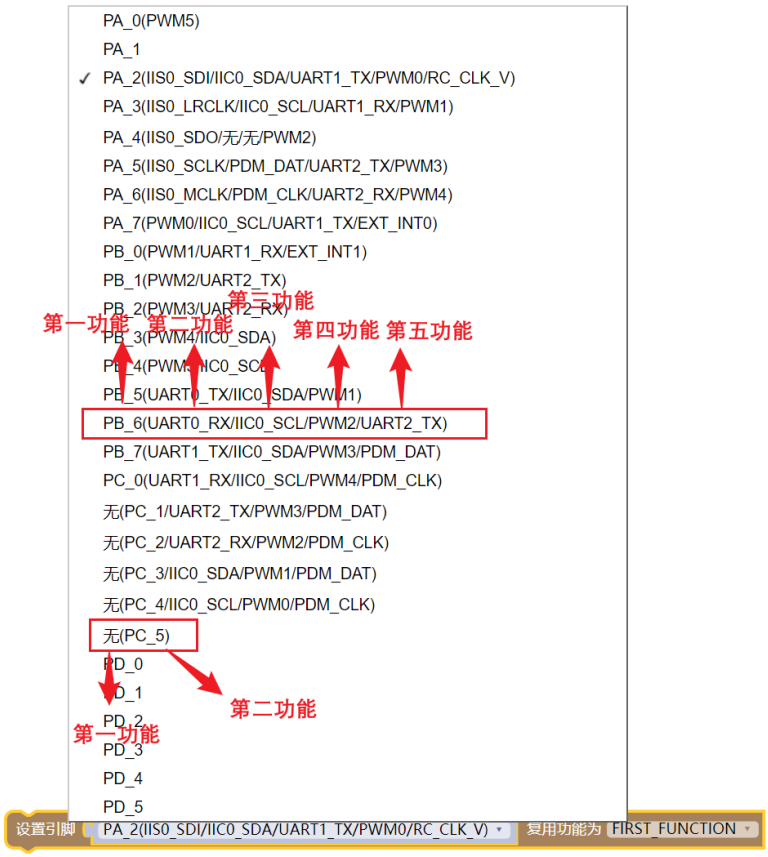
这条指令可以设置引脚是作为输出功能使用还是输入功能使用。PA0、PA1、PC1、PC2、PC3、PC4引脚默认是模拟引脚，作为IO输入输出用，还应设置成数字引脚。在下一个范例中有说明，请参照。

2.引脚复用功能设置

其次由于部分引脚会拥有多个功能，我们需要设置该引脚是第几功能。

点击引脚的下拉箭头，可以查找每个引脚都有哪些功能，也可以在芯片手册中进行查看。

我们可以通过这条指令，切换同一个引脚的不同功能。



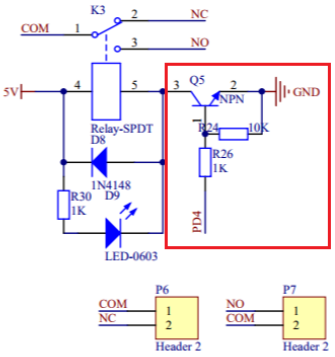
接下来我们来查看板载灯、继电器和彩屏背光灯的电路原理图，来确定设置的是高电平还是低电平。电路原理图可以点击右上角更多中的原理图进行查看。

原理图

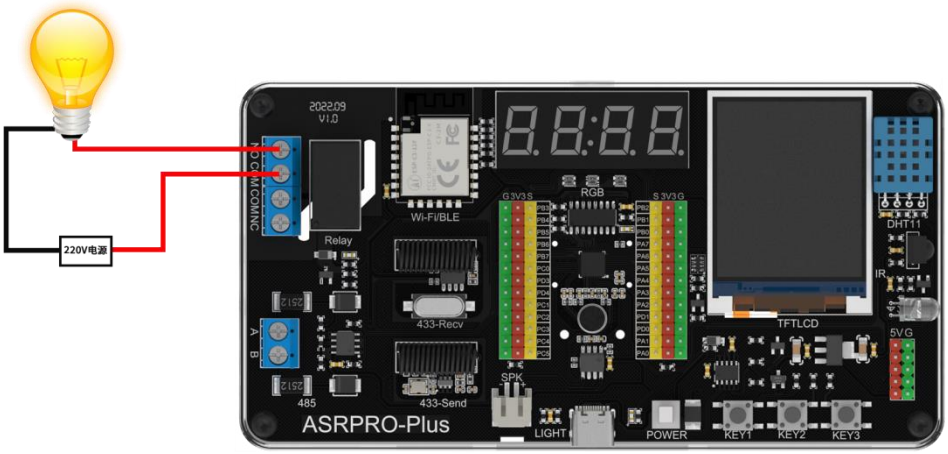
芯片手册

下图是继电器的电路原理图，当NPN三极管基极被R24下拉到电源负极，所以当信号输入端不接或者输入低电平的时候，基极的电压都是0V，此时基极处于截止状态，集电极和发射极不导通，所以继电器不导通，LED指示灯不亮；当信号输入端输入高电平，三极管基极也处于高电平，则集电极和发射极导通，继电器吸合，LED指示灯亮。

RELAY

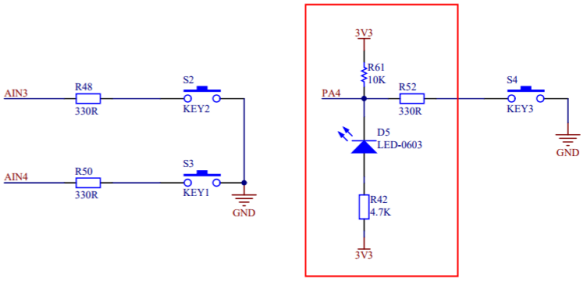


板载的继电器做过隔离设计，与家用灯泡可参考下图进行连接：

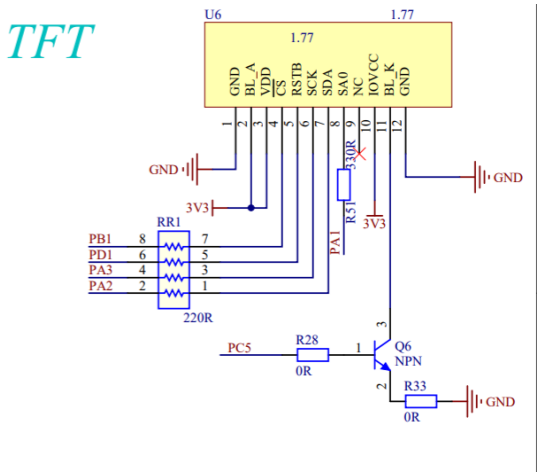


我们再来查看板载灯的原理图时我们发现，LED的正极接的是3V3的电源，只有当PA_4引脚输出低电平时，两者之间才会产生电压差，LED才能正向导通并发光；当PA_4引脚输出高电平时，没有电流通过，LED熄灭。

KEY



彩屏背光灯则是高电平点亮，低电平关闭。



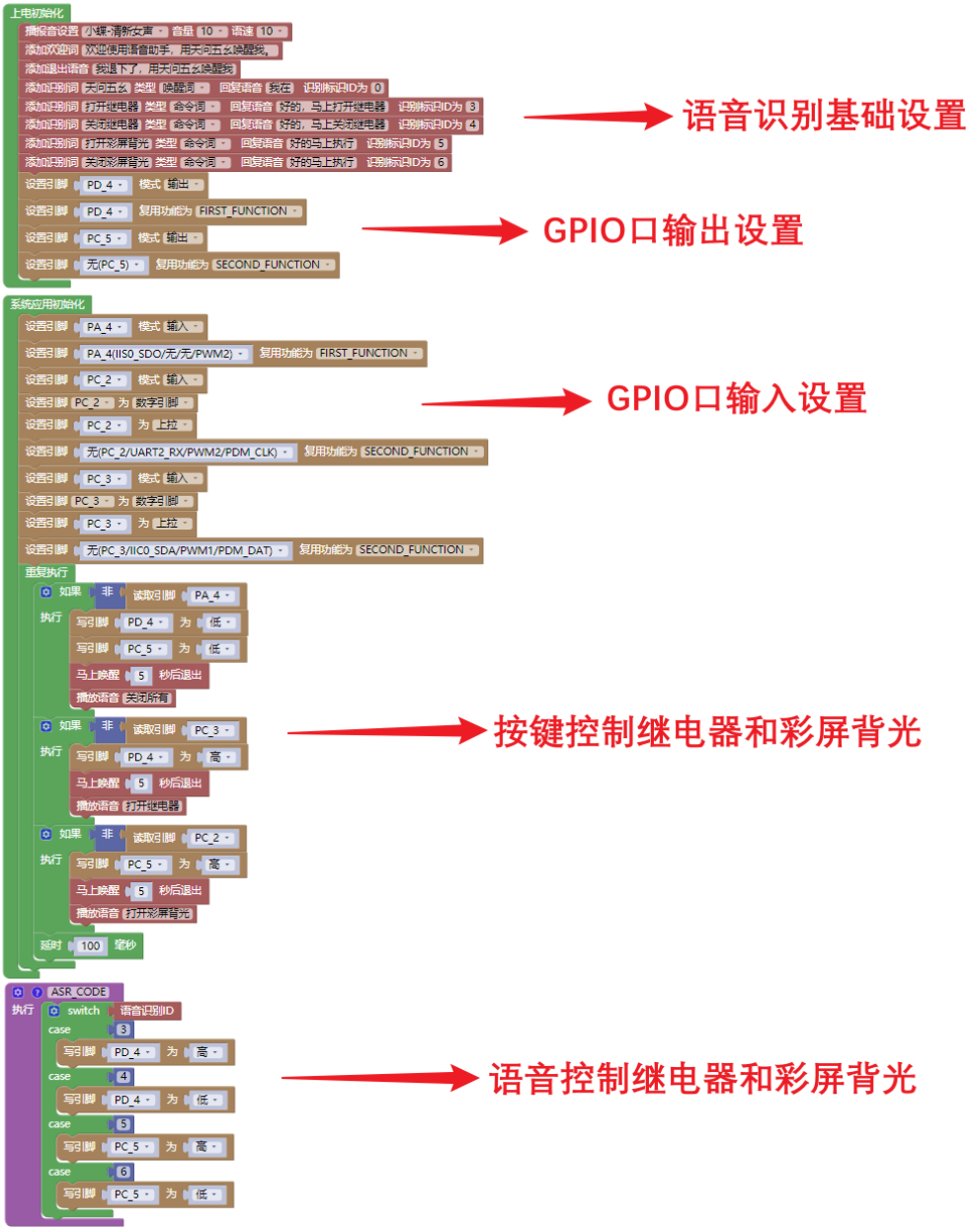
所以最终ASR_CODE部分程序设置如下图所示。

范例1.3 GPIO口输入设置

一、范例功能

本范例通过三个板载按键，控制ASRPRO-Plus的板载继电器PD_4和彩屏背光灯PC_5，实现语音和GPIO输入都能控制GPIO口输出的功能，达成学习GPIO口输入设置的目的。本范例适配ASRPRO-Plus开发板，其它开发板需修改PD_4、PC_5、PC_2、PC_3引脚为其它IO。

二、范例分析

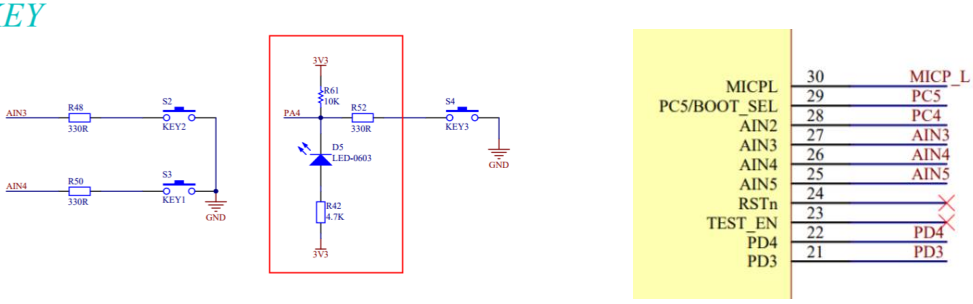


三、范例详解

ASRPRO-Plus有三个板载按键，KEY1、KEY2、KEY3，分别对应PC_2、PC_3、PA_4三个引脚。

PA_4引脚的输出设置在上一范例代码已经应用，PA_4既可以作输出功能，也可以作输入功能使用。配置为输入功能时，就是按键KEY3。见下方原理图，电路中电阻R61接在电源和IO之间，这个电阻就是上拉电阻，上拉电阻的目的是将不确定的信号通过一个电阻钳位在高电平。如果没有电阻R61，那么当按键被按下时，PA_4引脚是低电平；当按键松开时，PA_4处于悬空输入状态，IO状态不确定。接上拉电阻R61，当按键被按下时，PA_4引脚是低电平；当按键松开时，PA_4引脚不再是悬空状态，而是处于高电平状态。

KEY

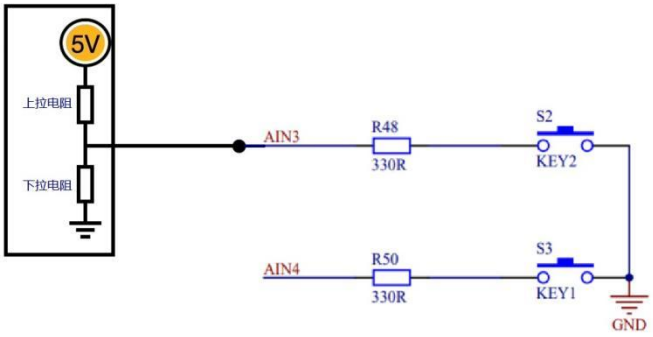


而PC_2和PC_3则比较特别。查看原理图发现，两个按键直接接到了模拟引脚AIN3和AIN4上，也没有外接上拉电阻。在这种情况下，我们发现，当按键KEY1、KEY2被按下时，AIN3、AIN4是低电平；当按键松开时，引脚处于悬空输入的状态，IO状态可能低电平或者高电平，不稳定状态。我们可以通过设置上拉模式或下拉模式完成IO的稳定状态。

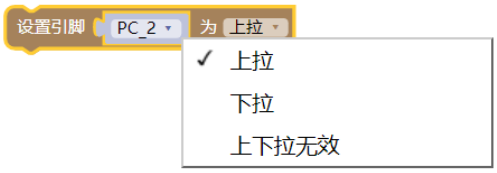
下面通过程序范例来对这三个引脚进行设置。

PA_4有外部上拉电阻，设置输入模式，设置PA_4引脚第几功能，就可以实现按键输入功能，如下所示。

PC_2和PC_3引脚，内部引脚如下图，每个引脚内部都有一个上拉电阻和下拉电阻，上拉电阻接电源（3.3V）下拉电阻接GND。我们可以通过设置引脚上下拉电阻的指令对引脚设置上拉或下拉或上下拉无效，控制了芯片内部中的上下拉电阻是否有效。

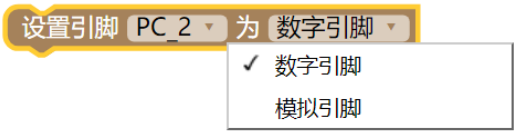


1.设置引脚上下拉电阻

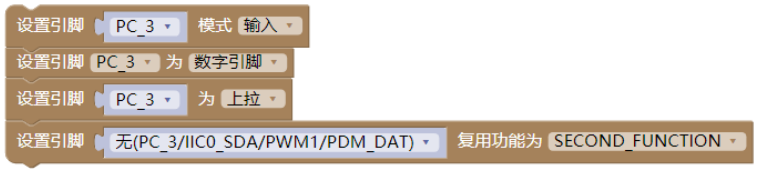


PC1-4引脚默认是模拟引脚。在本案例中，我们将两个按键作为数字引脚使用，我们还要通过下方的指令将PC_2和PC_3设置成数字引脚。

2.设置引脚数字/模拟



PC_2和PC_3的设置成IO输入按键，如下图：



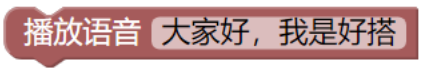
按键所对应的IO设置成按键输入模式后，下面来说明如何检测按键。

当按键被按下，按键引脚状态为低电平，可以使用“非 读取引脚PC_3”来表达真值。当按键被按下后，条件为真，执行继电器打开同时设置语音唤醒并播报“打开继电器”。

4. 马上唤醒几秒后退出

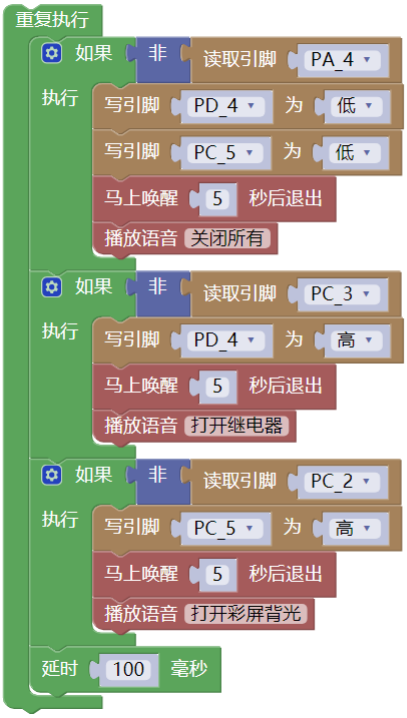
当需要进行播报语音之前，需要使用到这条指令，将ASRPRO唤醒。这条指令在语音识别类别中。如果已经设置永久唤醒模式则不需要这条指令。

5. 播报语音



这条指令非常常用，在需要进行语音播报的情况下都可以使用这条指令。这条指令在标准模式类别-执行动作区域中。

下方是按键控制板载继电器和彩屏背光灯打开和关闭的总程序。放入重复执行中后，建议添加短暂的延时100ms防抖，防止按键被短暂按下后语音重复播报。



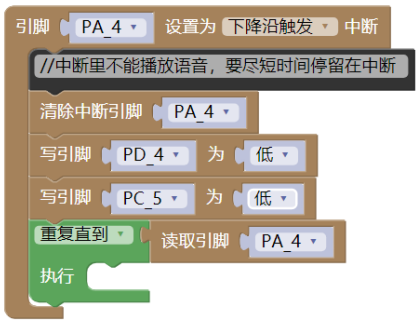
继电器、彩屏背光灯的GPIO口设置和语音识别的程序此处不再赘述，如有疑问查看上一范例GPIO口输出设置。

范例1.4 GPIO口中断

一、范例功能

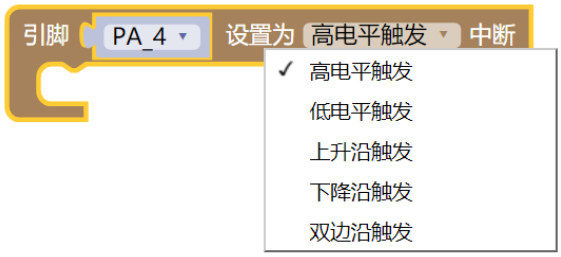
本范例通过使用PA_4中断的方法控制板载继电器PD_4和彩屏背光灯PC_5的关闭，实现中断控制GPIO输出的功能，达成学习GPIO口中断设置的目的。本范例适配ASRPRO-Plus开发板，其它开发板需修改PD_4、PC_5、PC_2、PC_3引脚为其它IO。

二、范例分析



三、范例详解

GPIO口的中断设置指令，包括了高电平触发、低电平触发、上升沿触发、下降沿触发和双边沿触发。硬件端口的中断可以使用的引脚只包含PA_0到PA_7、PB_0到PB_7，不包含PC端口。所以在本案例中，我们只以引脚PA_4为例作中断功能，另外两个按键PC_2和PC_3引脚还是在循环中作判断。

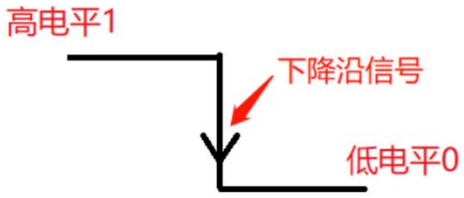


当按键松开时，处于高电平状态；当按键被按下时，处于低电平状态。

也就是说，假设将引脚PA_4设置为高电平触发中断，即表现为当引脚PA_4处于高电平状态，也就是松开状态时，会一直触发中断。

当按键被按下时，是由高电平到低电平，属于下降沿触发。

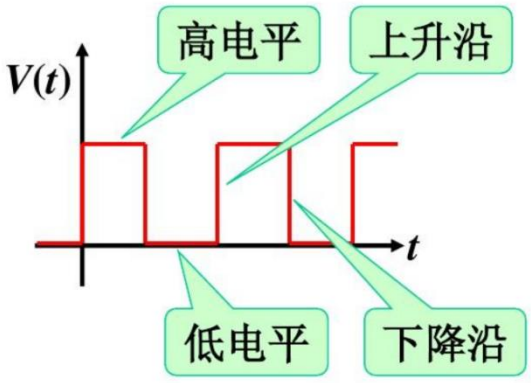
关于下降沿，可以参考下方这张图。当引脚从高电平到低电平时，就会发出一个下降沿信号。在本案例中，就是这个信号触发了中断。



进入中断后，要用清除中断引脚指令来清除中断标志，防止重复进入中断。

当按键松开时则跳出中断，程序如下所示。

如果将程序设置成上升沿触发，即表现为，当按键从被按住的状态（低电平）变成松开的状态（高电平）时，会触发一次中断。



双边沿触发则指，触发一次下降沿和一次上升沿。双边沿一般用来采集脉冲宽度。以按键为例，如果设置成双边沿触发中断，则可以用来判断按键的按下与松开。

一般来说，在实际应用中，由于边沿触发中断更好的稳定性能，高低电平触发中断已经基本被边沿触发取代。

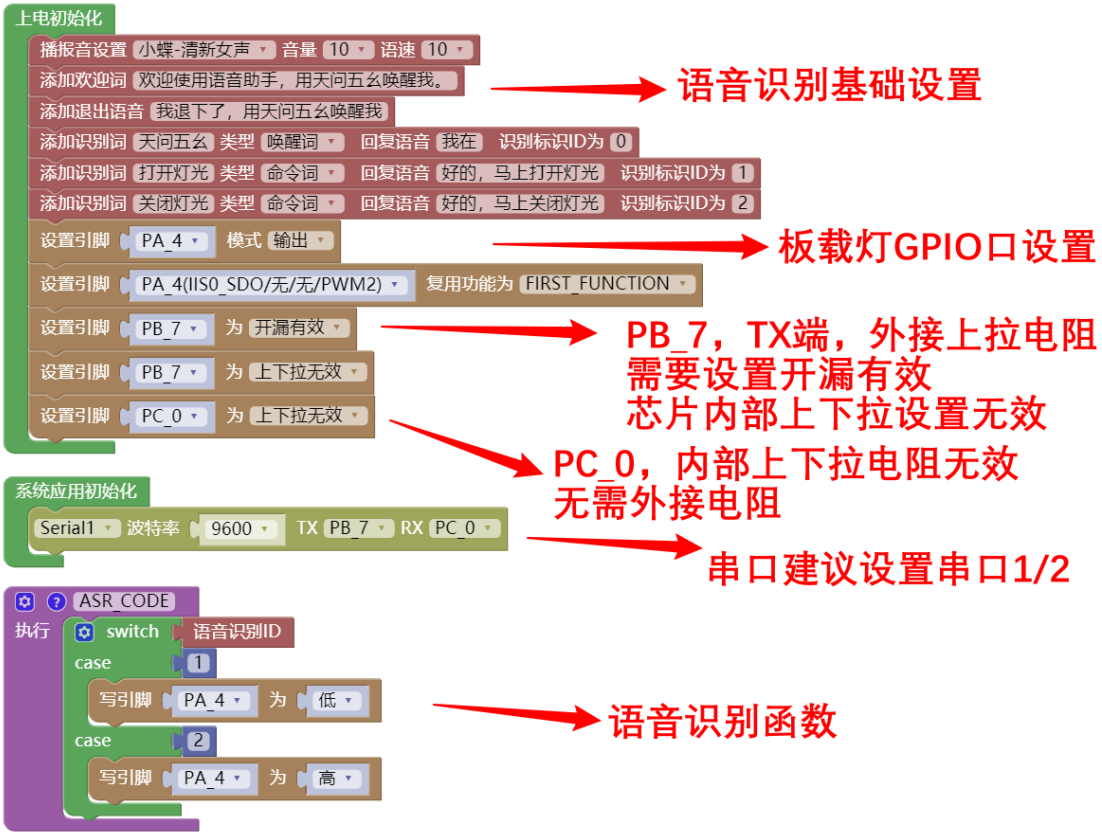
进入中断的时间尽量要短，不能放入语音播放。如果需要播放语音，可以通过发送消息来实现。后面我们可以在多线程部分学习到，这里不作详细介绍。

范例1.5 GPIO引脚接5V电平处理

一、范例功能

本范例说明GPIO引脚外接5V电平的使用方法，如何与外部MCU的GPIO为5V电平时正常与外接设备通讯。本范例适配ASRPRO-Plus开发板，其它开发板需修改PB7、PC0为PA2、PA3。

二、范例分析



三、范例详解

ASRPRO有3组串口，UART0 预留为程序升级接口，方便后期升级。和其它 MCU 通讯建议使用 UART1 或者 UART2。

ASRPRO的 IO口为3.3V电平，为了可靠通讯，建议设置TX、RX引脚内部上下电阻无效，同时设置TX为开漏模式，外接上拉电阻到5V，串联电阻，电路示意图如下所示：

软件设置如下：

更多和其它单片机串口连接示例，请查看附录二。

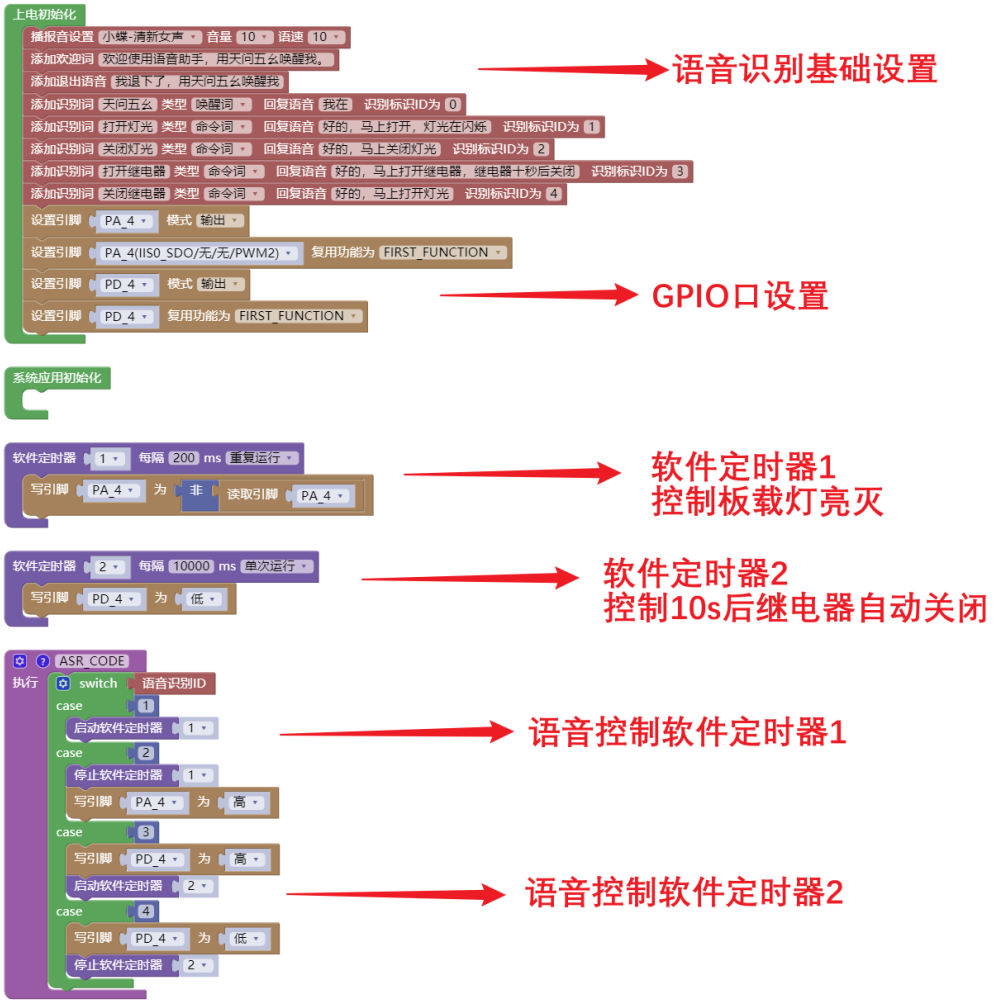
定时器的使用

范例1.6 软件定时器单次运行和重复运行

一、范例功能

本范例通过语音来控制定时器的开启与关闭，实现以此来控制ASRPRO-Plus的板载灯PA_4定时闪烁、板载继电器PD_4开关的功能，达成学习软件定时器程序编写的目的。本范例适配ASRPRO-Plus开发板，其它开发板需修改PD_4引脚为其它IO。

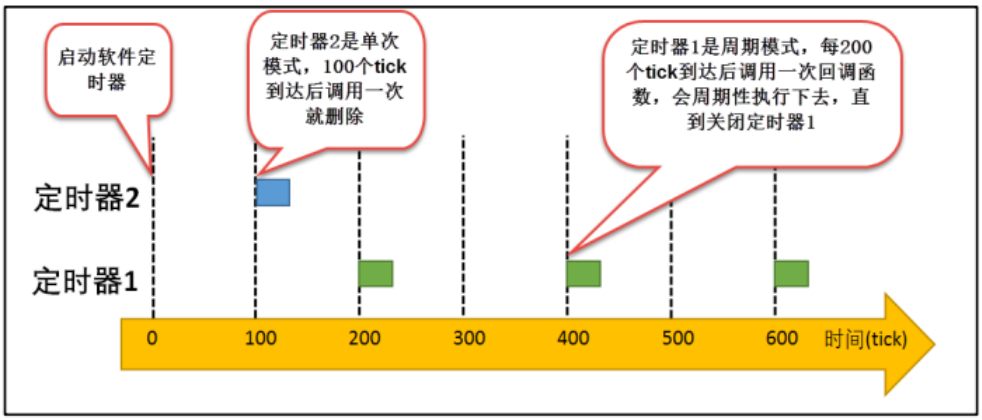
二、范例分析



三、范例详解

软件定时器是由操作系统提供，它构建在硬件定时器的基础之上，使系统能够提供不受硬件定时器资源限制的定时器服务，实现的功能与硬件定时器类似。

FreeRTOS提供的软件定时器支持单次模式和周期模式。单次模式和周期模式的定时时间到之后都会调用软件定时器的回调函数，用户可以在回调函数中加入要执行的工程代码。



如果设置为单次运行，当用户创建并启动了定时器后，定时时间到了，只执行一次回调函数之后就将该定时器设置为休眠状态；

如果设置为周期模式，那么这个定时器会按照设置的定时时间循环执行回调函数，直到用户将该定时器删除。

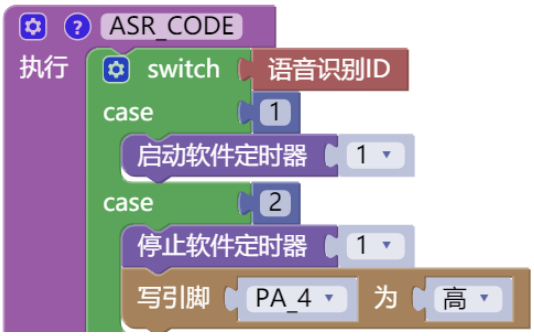
每个定时器之间互不干扰。

指令在多线程类别指令中。

指令中包含了设置软件定时器的编号、间隔时间、单次运行或重复运行、软件定时器的启用和软件定时器的停止。软件定时器的图形块指令默认有8个，只要内存足够可以设置大于8个。

在本范例中，定时器1的程序如下所示。当软件定时器1启动时，PA_4每隔200ms就会进行高低电平的切换。当PA_4为高电平时，200ms后会把PA_4从高电平设置成低电平；200ms后又把PA_4从低电平设置成高电平，而且是一直重复进行。具体表现为板载灯PA_4闪烁。

我们用语音来控制软件定时器1的启动和停止。当然停止定时器时记得设置将板载灯关闭。程序如下所示。



软件定时器也可以单次运行。例如软件定时器2，就是当定时器启动10s后，设置板载继电器PD_4关闭。当然我们也可以通过语音控制，提前对继电器进行关闭。

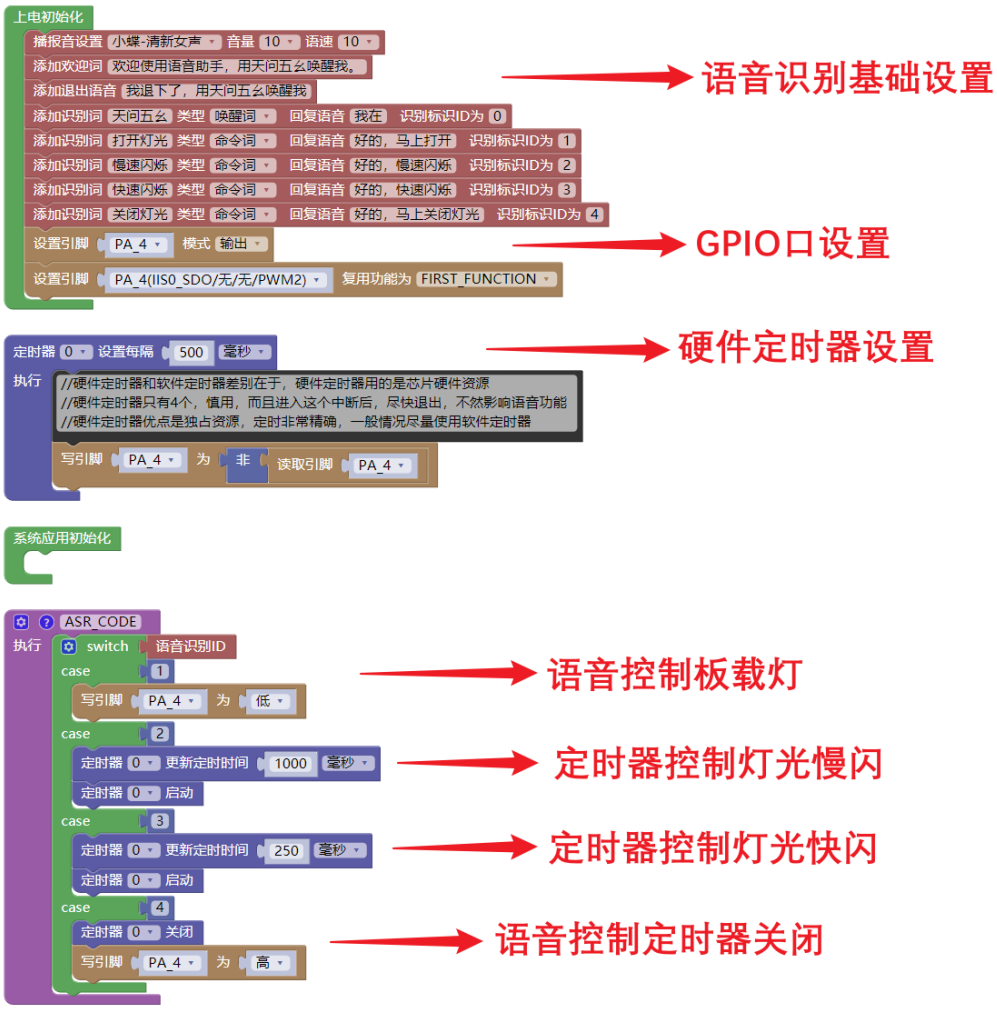
在语音识别ASR_CODE函数中少用延时，尽量使用软件定时器，可以让语音识别更加流畅。

范例1.7 硬件定时器使用

一、范例功能

本范例通过硬件定时器，实现控制ASRPRO-Plus的板载灯PA_4快速闪烁和慢速闪烁的功能，达成学习硬件定时器程序编写的目的。本范例适配所有ASRPRO开发板。

二、范例分析



三、范例详解

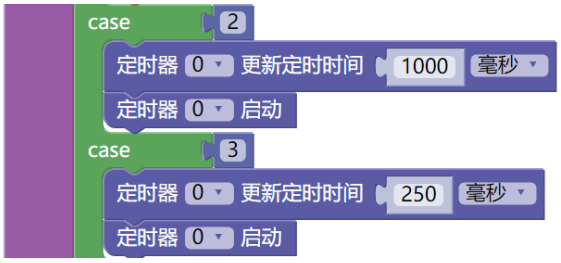
硬件定时器是芯片内置的定时器模块，占用物理硬件。一般是由外部晶振或内部RC晶振提供给输入时钟，配置寄存器，接受控制输入，设定时间值后芯片中断控制器产生时钟中断。硬件定时器的精度一般很高，可以达到纳秒级别，并且是中断触发方式。

硬件定时器和软件定时器的差别在于，硬件定时器用的是芯片硬件资源。

硬件定时器的指令和软件定时器类似，都有定时器的设置、定时器启动和定时器关闭。与软件定时器不同的是，硬件定时器默认是永远循环，但是可以随时去更新定时的间隔时间。

需要注意的是硬件定时器只有4个，进入定时中断后，事务处理结束尽快退出，不然影响语音功能。

范例程序中，默认的闪烁间隔时间是500ms，定时器不分频模式最大可设置38.8s左右，可以设置为2、4、8和16分频，最大定时时间可以再乘以2、4、8、16。当语音识别到慢速闪烁时，定时器更新定时间隔时间为1000ms，板载灯PA_4进入慢闪；当语音识别到快速闪烁时，定时器更新时间为250ms，板载灯PA_4进入快闪。注意更新定时时间后需要重新启动定时器。



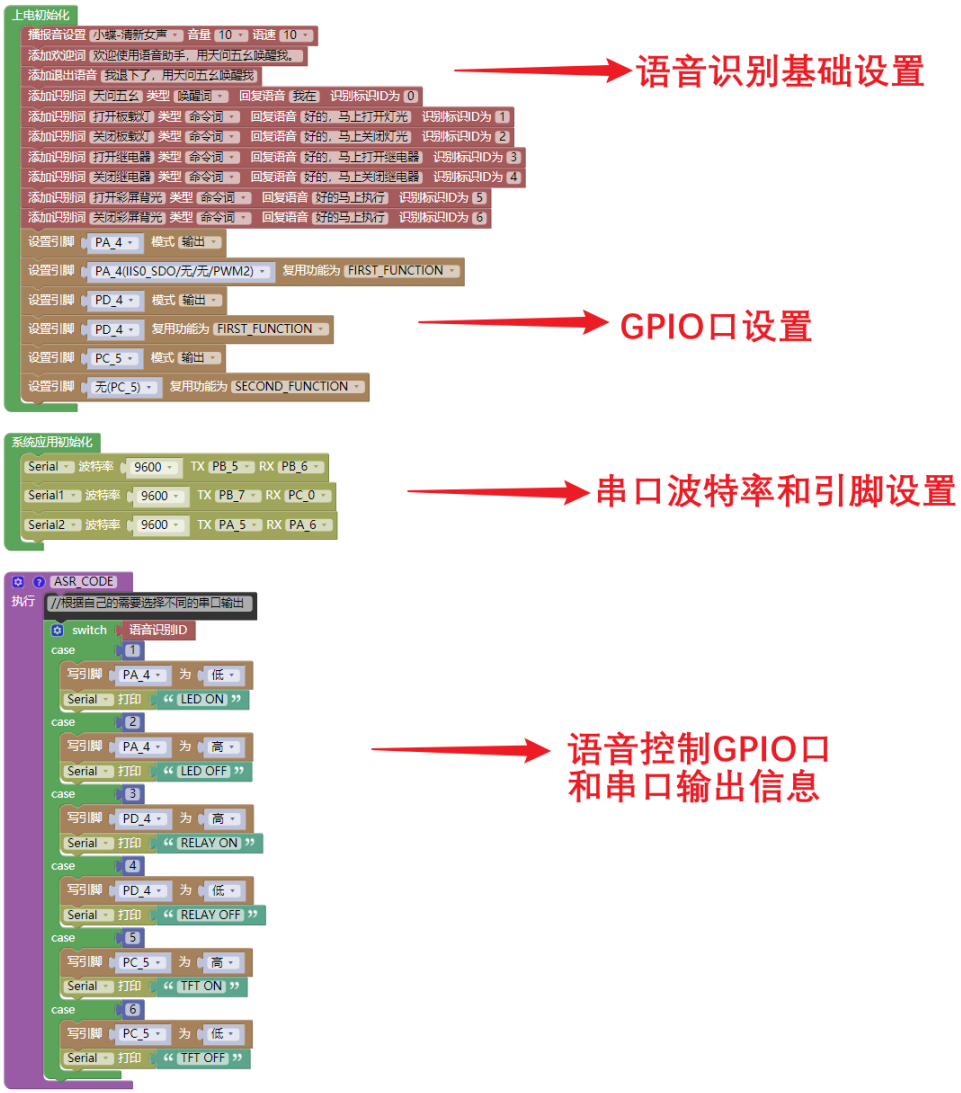
串口

范例1.8 串口设置与输出

一、范例功能

本范例通过串口设置，实现在控制ASRPRO-Plus的板载灯PA_4、板载继电器PD_4和彩屏背光灯PC_5的同时，串口同时输出相关信息，达成学习如何进行串口设置与串口输出信息的目的。本范例适配ASRPRO-Plus开发板，其它开发板需修改PD_4、PC_5、引脚为其它IO，串口1设置为PA2、PA3，串口2设置为PA5、PA6。

二、范例分析



三、范例详解

单片机通信是单片机与外部设备或其他计算机之间的信息交换，可以分为并行通信和串行通信。并行通信通常是将数据字节的各位用多条数据线同时进行传送。串行通信是指将数据字节以一位一位的形式在一条传输线上逐个地传送。并行通信控制简单、传输速度快，但是占用的端口数量多；串行通信端口占用少，但是数据的传送控制比并行通信复杂。

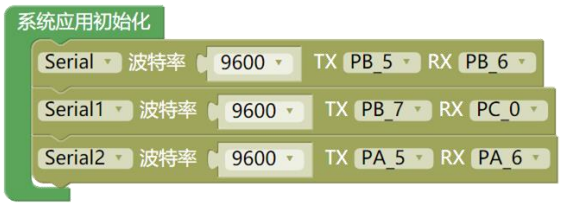
串行通信常用的有UART、IIC、SPI和单总线等几类，本节内容主要讲解UART部分。

串口的初始化设置一般放在系统应用初始化中，指令在串口类别指令中。

其中串口0 Serial，默认的TX是PB_5，RX是PB_6，无法更改。

串口1和串口2的引脚则可以相对自由地进行设置，RX和TX可以自由组合。

ASRPRO-Plus外接的RS485、WiFi、433无线等都使用了一些串口。当我们进行串口1或者串口2的设置时，很容易与这些模块发生冲突。为了数据的稳定性，建议串口1或者串口2使用RS485模块的RX1和TX1，也就是PC_0-RX1和PB_7-TX1，RS485模块不要外接。ASRPRO开发板和核心板则不受影响。

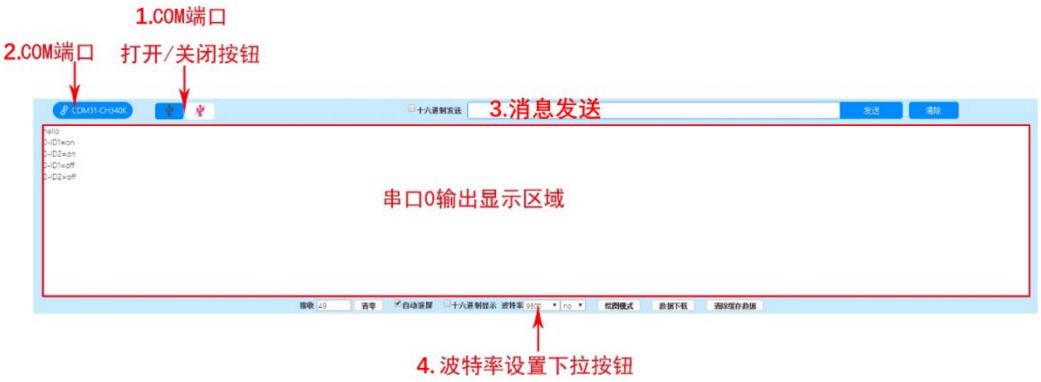


查看右侧字符代码我们也可以发现，这条指令会帮助我们设置好每个引脚具体是第几功能。所以使用这条指令设置后，无需再次设置引脚的复用功能为串口。

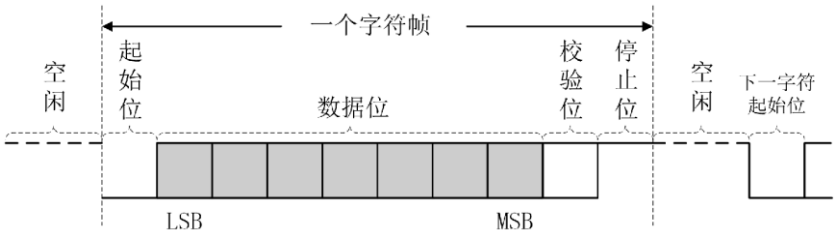
```
setPinFun(13,SECOND_FUNCTION);
setPinFun(14,SECOND_FUNCTION);
Serial.begin(9600);
setPinFun(15,SECOND_FUNCTION);
setPinFun(16,SECOND_FUNCTION);
Serial1.begin(9600);
setPinFun(5,FORTH_FUNCTION);
setPinFun(6,FORTH_FUNCTION);
Serial2.begin(9600);
```

编写好语音控制串口打印不同字符串的程序，我们可以通过串口监视器直接查看串口0的输出值。我们以语音播报打开板载灯和关闭板载灯为例进行查看。

打开左上角的串口监视器，会得到下方的界面。其中点击区域1可以打开串口；区域2部分是串口的输出信息；在区域3内可以往串口发送消息；区域4部分可以设置串口波特率，注意此处的串口波特率要与程序中相同。当我们说打开板载灯、关闭板载灯，串口监视器会显示LED ON和LED OFF。由于此处没有使用换行指令，所以会在同一行显示。



波特率（BaudRate）表示数据传送速率，即每秒钟传送的二进制位数。波特率通常单位是 bit/s，例如数据传送速率为120字符/秒，而每一个字符为10位（1个起始位，7个数据位，1个校验位，1个结束位），则其传送的波特率为10×120＝1200位/秒＝1200波特。为提高通讯速率，可以在1200基础上倍频，所以形成了2400、4800、9600、19200等标准波特率。比较常用的波特率有9600，57600，115200等等。



起始位：先发出一个逻辑“0”信号，表示传输字符的开始。

数据位：可以是5~8位逻辑“0”或“0”。如 ASCII 码（7位），扩展 BCD 码（8位）。

校验位：数据位加上这一位后，使得“1”的位数应为偶数(偶校验)或奇数(奇校验)

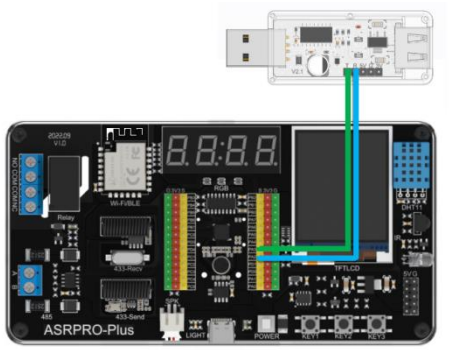
停止位：它是一个字符数据的结束标志。可以是1位、1.5位、2位的高电平。

空闲位：处于逻辑“1”状态，表示当前线路上没有资料传送。

同理我们也可以让串口1和串口2打印字符串。

我们可以通过软件自带的串口监视器来查看，点击串口监视器的左上角切换串口即可。

那么如何查看串口具体的端口号呢？我们这里以使用STC-Link为例，通过使用STC-ISP详细介绍如何查看串口1、串口2的输出信息。硬件连接如下图所示：

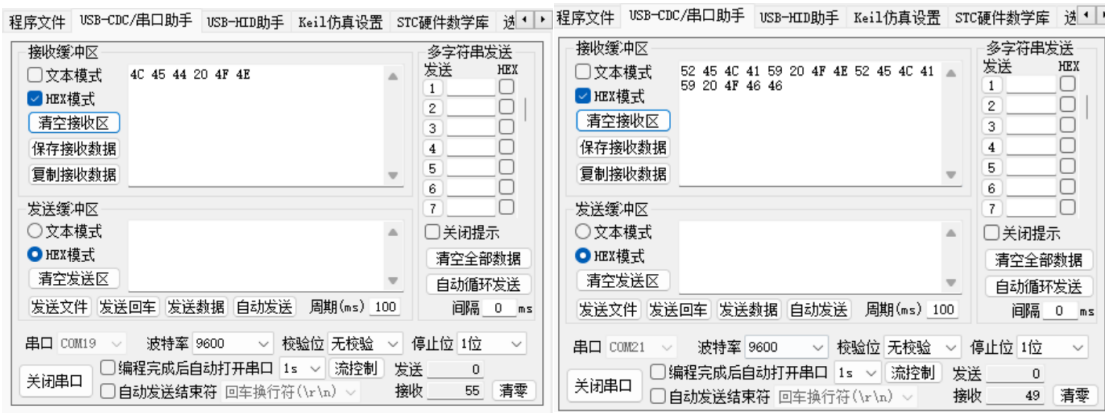


将STC-LINK连接到电脑，我们可以在设备管理器查看到串口的COM口。由下图我们可以看到端口号为COM20。

如果多设备连接，可通过插拔连接口，查看不同设备对应的端口号。当然，直接使用串口0的驱动是CH340K，用STC-Link使用串口1和串口2的驱动是CP210x。查看串口0和串口1的设备管理器如下所示，查看到串口0对应COM19，串口2对应COM21。

打开STC-ISP，在串口助手的左下角，设置好串口、波特率，打开串口，进行测试。

当ASRPRO识别到“打开板载灯”、“打开继电器”、”关闭继电器“等语音时，串口0和串口会输出对应的信息。切换文本模式则可以显示接收到的字符串



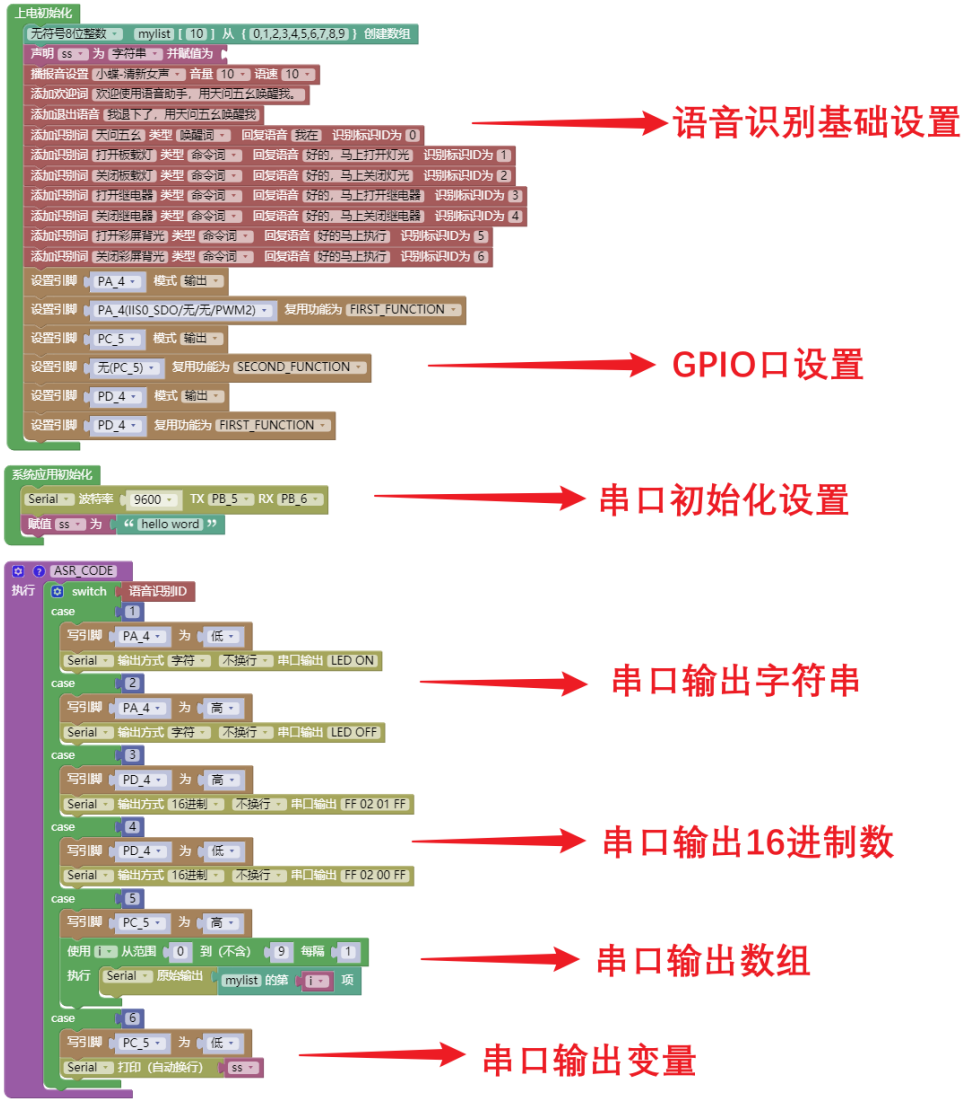
串口发送接收消息的程序编写，以及外接设备如何使用，我们会在后续多线程的范例和附录二中详细讲解。

范例1.9 串口输出十六进制与字符串

一、范例功能

本范例在控制ASRPRO-Plus的板载灯PA_4、板载继电器PD_4和彩屏背光灯PC_5的同时，实现串口分别输出字符串、十六进制、数组、变量的功能，达成学习进行串口输出不同信息的目的。本范例适配ASRPRO-Plus开发板，其它开发板需修改PD_4、PC_5引脚为其它IO。

二、范例分析



三、范例详解

串口发送16进制数或者字符串的指令在标准模式的执行动作区域。

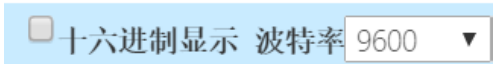
这条指令可以设置串口、设置发送16进制数/字符串、设置发送串口消息后是否换行以及串口的具体输出内容。

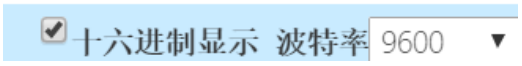

```
case 4:
 digitalWrite(28,0);
 Serial.write(0xFF);
 Serial.write(0x02);
 Serial.write(0x00);
 Serial.write(0xFF);
 break;
case 5:
 digitalWrite(21,1);
 for (int i = 0; i < 9; i = i + 1) {
   Serial.write(mylist[(int)(i)]);
 }
```

查看代码我们发现，这条指令和串口原始输出指令相同，都是Serial.write。

我们先来查看串口监视器，观察串口输出字符串和16进制数有什么不同。

通过发现，我们看到下方有个十六进制显示可以勾选。





串口输出字符串：不勾选则串口监视器显示LED ON，勾选十六进制显示则会显示该字符串对应的ASCII码，如下所示。

```
LED ON4c 45 44 20 4f 4e
```

串口输出16进制数：则一定要勾选十六进制显示，不然会出现乱码。下方是串口输出16进制数FF 02 01 FF，不勾选和勾选的两个输出结果。

```
□□□ff 02 01 ff
```

ASCII码表参考下方表格。

Bin	Oct	Dec	Hex	缩写/字符	解释	Bin	Feb	Apr	Hex	缩写/字符	解释
(二进制)	(八进制)	(十进制)	(十六进制)			(二进制)	(八进制)	(十进制)	(十六进制)		
0011 0000	60	48	0x30	0	字符0	0100 1110	116	78	0x4E	N	大写字母N
0011 0001	61	49	0x31	1	字符1	0100 1111	117	79	0x4F	O	大写字母O
0011 0010	62	50	0x32	2	字符2	0101 0000	120	80	0x50	P	大写字母P
0011 0011	63	51	0x33	3	字符3	0101 0001	121	81	0x51	Q	大写字母Q
0011 0100	64	52	0x34	4	字符4	0101 0010	122	82	0x52	R	大写字母R
0011 0101	65	53	0x35	5	字符5	0101 0011	123	83	0x53	S	大写字母S
0011 0110	66	54	0x36	6	字符6	0101 0100	124	84	0x54	T	大写字母T
0011 0111	67	55	0x37	7	字符7	0101 0101	125	85	0x55	U	大写字母U
0011 1000	70	56	0x38	8	字符8	0101 0110	126	86	0x56	V	大写字母V
0011 1001	71	57	0x39	9	字符9	0101 0111	127	87	0x57	W	大写字母W
0011 1010	72	58	0x3A	:	冒号	0101 1000	130	88	0x58	X	大写字母X
0011 1011	73	59	0x3B	;	分号	0101 1001	131	89	0x59	Y	大写字母Y
0011 1100	74	60	0x3C	<	小于	0101 1010	132	90	0x5A	Z	大写字母Z
0011 1101	75	61	0x3D	=	等号	0101 1011	133	91	0x5B	[	开方括号
0011 1110	76	62	0x3E	>	大于	0101 1100	134	92	0x5C	\	反斜杠
0011 1111	77	63	0x3F	?	问号	0101 1101	135	93	0x5D	]	闭方括号
0100 0000	100	64	0x40	@	电子邮件符号	0101 1110	136	94	0x5E	^	脱字符
0100 0001	101	65	0x41	A	大写字母A	0101 1111	137	95	0x5F	_	下划线
0100 0010	102	66	0x42	B	大写字母B	0110 0000	140	96	0x60	`	开单引号
0100 0011	103	67	0x43	C	大写字母C	0110 0001	141	97	0x61	a	小写字母a
0100 0100	104	68	0x44	D	大写字母D	0110 0010	142	98	0x62	b	小写字母b
0100 0101	105	69	0x45	E	大写字母E	0110 0011	143	99	0x63	c	小写字母c
0100 0110	106	70	0x46	F	大写字母F	0110 0100	144	100	0x64	d	小写字母d
0100 0111	107	71	0x47	G	大写字母G	0110 0101	145	101	0x65	e	小写字母e
0100 1000	110	72	0x48	H	大写字母H	0110 0110	146	102	0x66	f	小写字母f
0100 1001	111	73	0x49	I	大写字母I	0110 0111	147	103	0x67	g	小写字母g
1001010	112	74	0x4A	J	大写字母J	0110 1000	150	104	0x68	h	小写字母h
0100 1011	113	75	0x4B	K	大写字母K	0110 1001	151	105	0x69	i	小写字母i
0100 1100	114	76	0x4C	L	大写字母L	0110 1010	152	106	0x6A	j	小写字母j
0100 1101	115	77	0x4D	M	大写字母M	0110 1011	153	107	0x6B	k	小写字母k

我们要学会区分按字符模式发送和按16进制发送。

当我们不点选16进制时，按字符模式发送。这是我们输入的文本区的内容是一个个字符。比如输入06，这时06为‘0’和‘6’两个字符。发送的时候会将字符‘0’的ASCII码和字符‘6’的ASCII码发送出去，即是0x30和0x36。当我们以文本模式(ASCII)接收时就会收到06,当我们以16进制(HEX)接收时就会收到0X30，0X36,其中0x代表16进制，不会在串口调试助手上显示出来，只会显示 30 36。

当我们按16进制发送06时，这时06为一个16进制数即0x06。当我们以文本模式(ASCII)接收时就会收到的为乱码，因为16进制的0x06的ASCII码是不可显示字符，为ACK。当我们以16进制(HEX)接收时就会收到0X06，其中0x代表16进制，不会在串口调试助手上显示出来，只会显示06。

根据这两种情况，我们可以得出结论：

按字符模式发送串口消息时，既可以用字符模式显示，又可以按16进制显示；

按16进制模式发送串口消息时，建议用16进制显示。

串口输出变量类型的字符串：和直接输出字符串一致。

```
hello word
68 65 6c 6c 6f 20 77 6f 72 64 0d 0a
```

串口输出数组：数组的创建一般放在上电初始化中。

对于数组，我们既可以和范例代码中一样，使用原始输出write，此时建议用16进制数显示，结果如下所示。

00 01 02 03 04 05 06 07 08

也可以使用串口打印指令print，此时直接显示即可，结果如下所示。

012345678

查看右边代码指令，我们来看一下串口原始输出和串口打印的区别。

串口原始输出对应的函数是Serial.write()，直接调用硬件串口发送函数，支持如下三种

Serial.write(val)

Serial.write(str)

Serial.write(buf, len)

串口打印对应的函数是Serial. print ()，能根据输入的类型自动转换后再调用Serial.write()发送。支持如下多种格式：

```
size_t print(const __FlashStringHelper *);
    size_t print(const String &);
    size_t print(const char[]);
    size_t print(char);
    size_t print(unsigned char, int = DEC);
    size_t print(int, int = DEC);
    size_t print(unsigned int, int = DEC);
    size_t print(long, int = DEC);
    size_t print(unsigned long, int = DEC);
    size_t print(double, int = 2);
    size_t print(const Printable&);
```

更多有关串口的学习可以参考多线程中的部分案例。

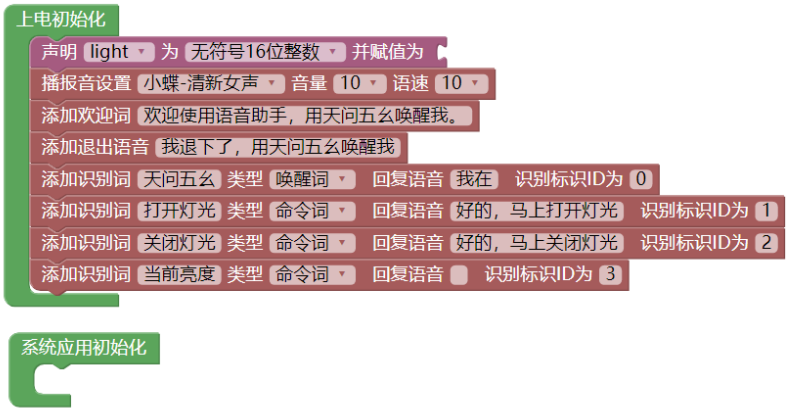
ADC

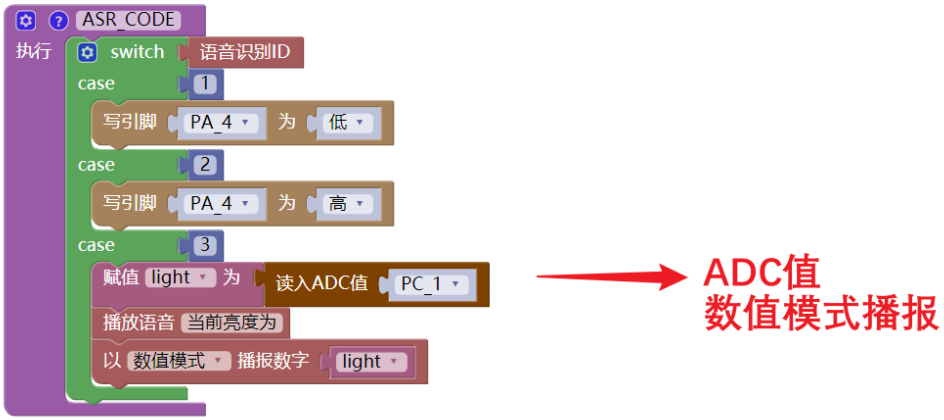
范例1.10 ADC使用

一、范例功能

本范例通过获取板载亮度传感器引脚PC_1的ADC值，实现用语音的数值模式进行播报ADC数值的功能，达成学习ADC模块程序编写的目的。本范例适配ASRPRO-Plus开发板，其它开发板需修改PC_1为PC_4。

二、范例分析





三、范例详解

本案例使用语音以数值模式播报了板载亮度传感器的ADC值。

ADC是模数转换器（Analog-to-digital converter），指的是将模拟信号转换为数字信号。由于模拟信号容易受到干扰，信号处理时容易受到其他条件的限制，且不易存储，所以在实际处理经常换成数字信号。ADC就在这个时候充当了模拟信号转化为数字信号的中转站。

ADC的位数由芯片本身决定，一般有8位，10位和12位的：如果8位AD，采样值范围为0-255；如果10位AD，采样值范围为0-1023；12 位 AD，采样值范围为0-4095。ASRPRO

中的ADC是12位AD，是12位二进制数的意思，等于10进制的0-4095，也就是采样值范围在0-4095之间。位数越高，精度也越高，进而误差越小。

接下来我们介绍以数值模式播报数字和以号码模式播报数字两条指令。这两条指令在标准模式类别-执行动作中。

1.号码模式播报数字

这条指令是以号码模式播报数字，指的是单独播报输入数字的每一个数字，例如输入123，则会播报一二三。

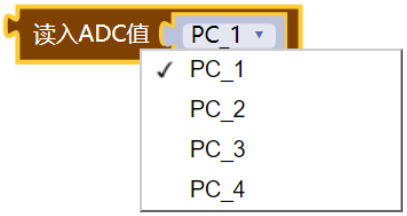
2.数值模式播报数字

这条指令是以数值模式播报数字，指的是整体播报输入的数字，例如输入123，则会播报一百二十三。

当我们需要用到语音播报数字时，就可以用到这两条指令。播报电话号码等可以使用号码模式，播报温湿度、传感器数值等可以使用数值模式。

然后我们来学习一下ASRPRO的ADC引脚。

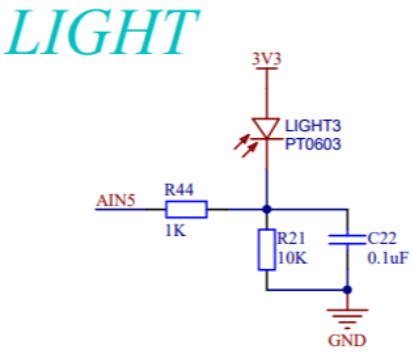
3.读取ADC值



这条指令在ADC模块类别中，用于读取ADC值。

使用这条指令不用再进行该引脚的GPIO口的设置，图形化模块对应的代码中已经对该引脚的GPIO口进行了设置。

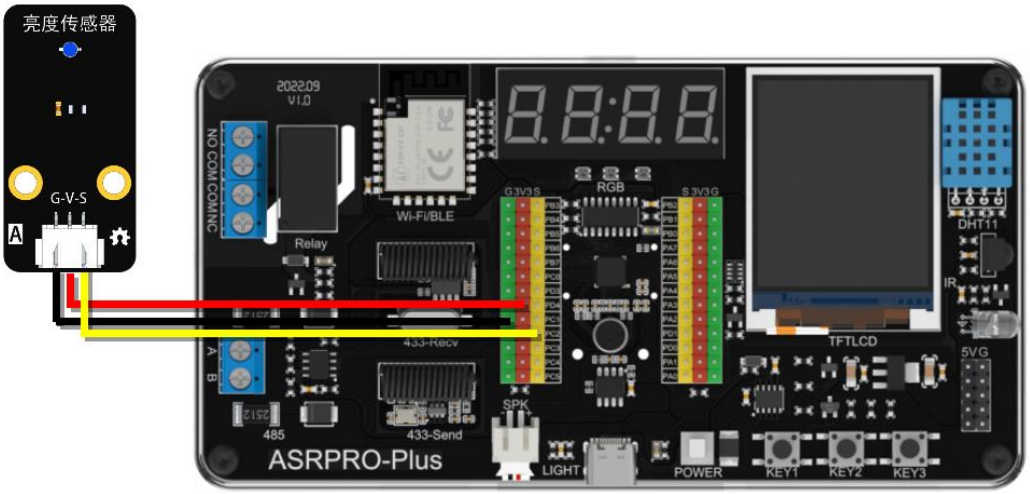
案例中使用的引脚为PC_1的板载亮度传感器。



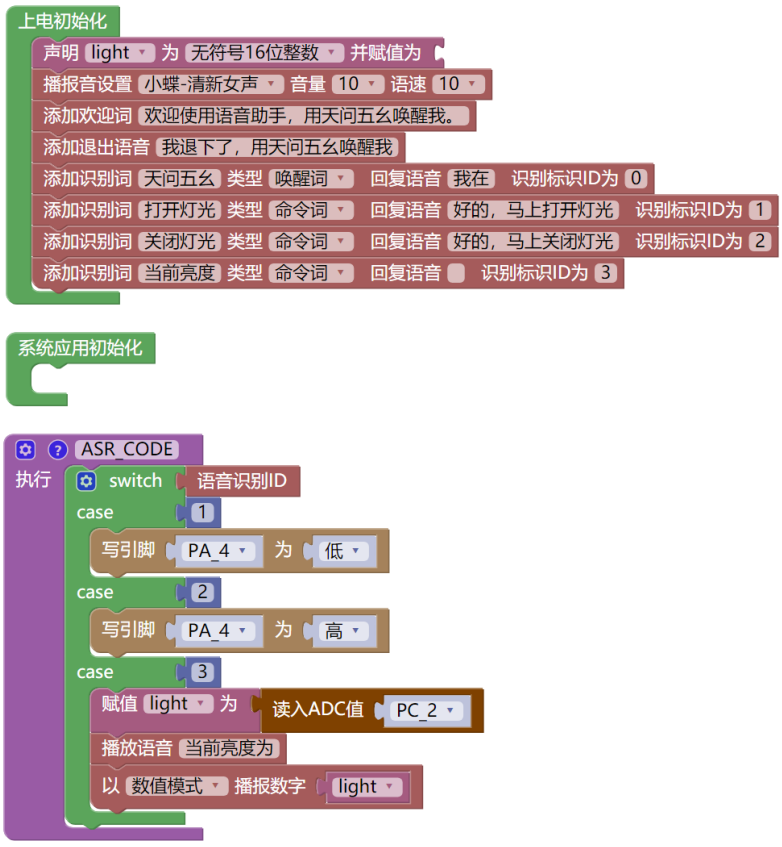
查看电路原理图我们可以发现，光线变化会引起电阻变化，从而电压跟着变化，我们可以通过ADC采样获取电压值。

这里以在PC_2引脚外接一个光敏传感器为例，介绍如何进行程序编写。PC_2引脚，我们在之前的GPIO口输入设置中使用过，当时是设置PC_2引脚为数字引脚并接上拉电阻，作按键功能使用。PC_2引脚默认是模拟引脚，在这里当作ADC引脚使用。

外接图片参考：



整体程序如下所示，可供参考。



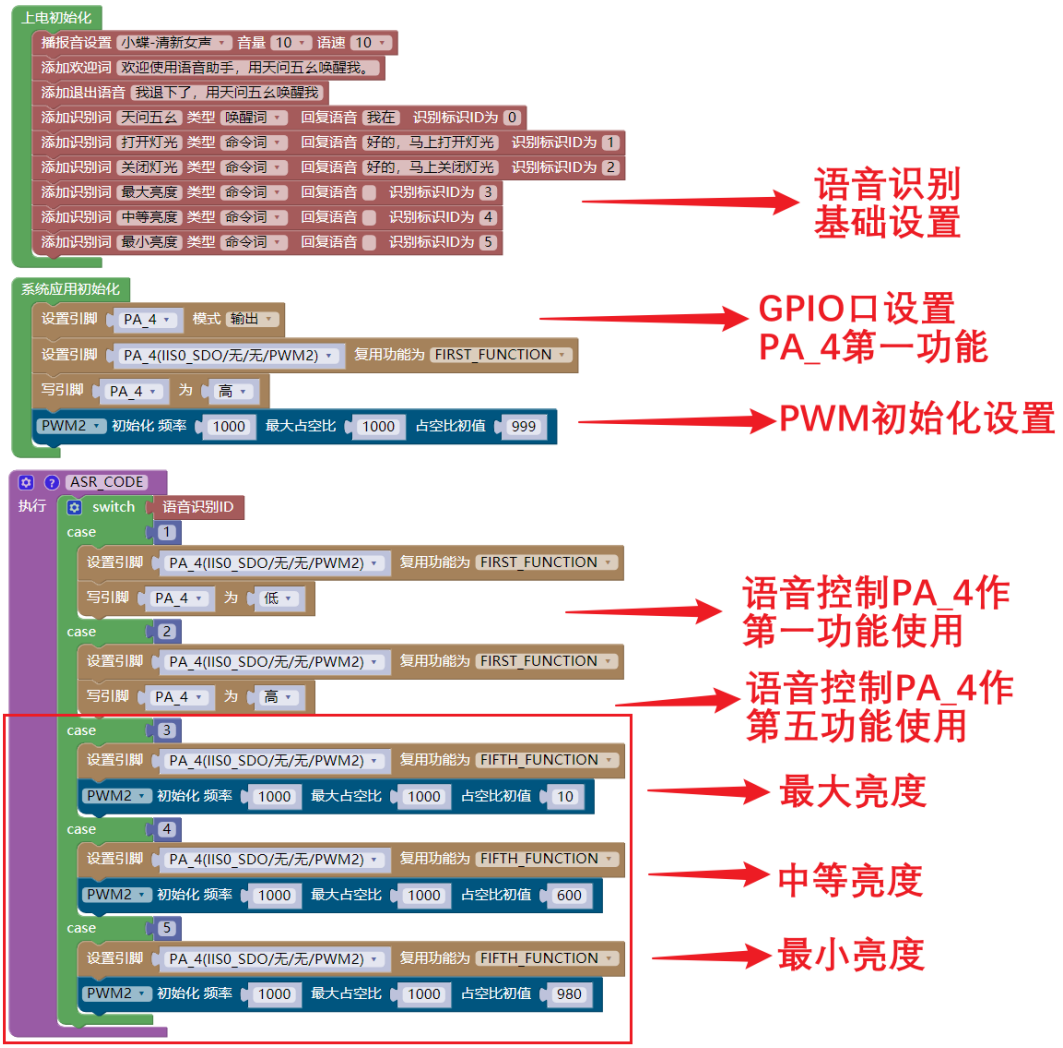
PWM

范例1.11 PWM控制原理

一、范例功能

本范例通过PWM的方式调节PA_4引脚的板载灯的亮度，实现用PWM自由调节板载灯光亮度的功能，达成学习复用功能的切换和PWM模块程序编写的目的。本范例适配所有ASRPRO开发板。

二、范例分析



三、范例详解

PA_4引脚在本范例前我们输出GPIO模式，使用它的第一功能。通过查看复用功能，PA_4引脚的第五功能是PWM2，可以设置为PWM引脚使用。

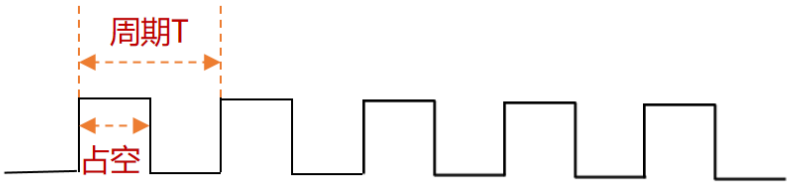
1.PWM初始化设置

该指令在PWM模块类别指令中，包含三个参数，频率、占空比和占空比初值。

PWM指脉冲宽度调制，是一种通过数字信号对模拟电路进行高效控制的方法。 PWM一般用于控制LED发光强度或电机速度。 PWM具有两个很重要的参数：频率和占空比。

频率：每秒钟信号从高电平到低电平再回到高电平的次数，也是周期的倒数。

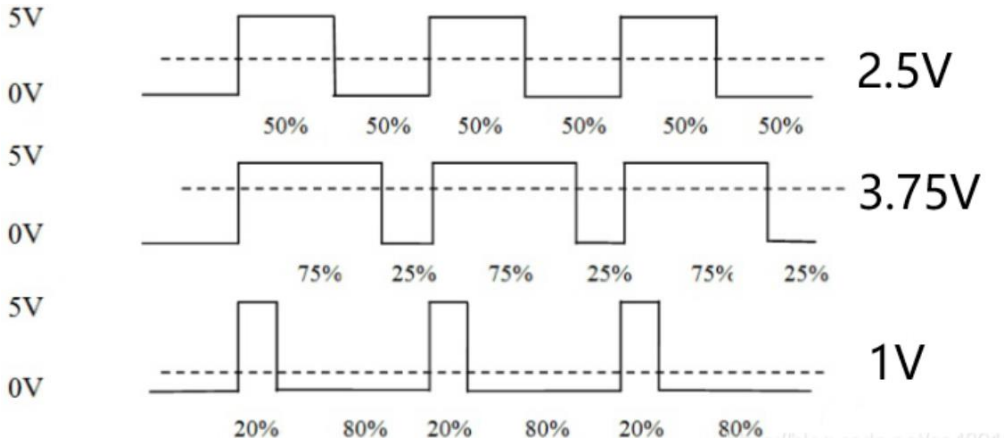
占空比：一个脉冲周期内，信号处于高电平的时间占据整个脉冲周期的百分比。通常，我们将一个脉冲周期内维持高电平的时间称为占空，通过数字设备可以改变占空值。



其中频率的单位是Hz，周期的单位是s。例如频率设置1000，则周期T为1/1000 s，也就是1ms。频率根据实际情况可以自由进行设置。初始化的设置一般用于PWM的单次设置。占空比初值/最大占空比=占空比，最大占空比配合占空比值一起设置，一般来说设置为1000。

$$占空比=\frac{占空}{脉冲周期}\times 100\%$$

下方三张图分别代表了占空比为50%、75%、20%。以占空比为20%为例，也就是说，在一个周期内，20%的时间设置的是高电平，另外80%的时间设置的是低电平。我们都知道，电压是以一种连接1或断开0的重复脉冲序列被夹到模拟负载上去的（例如LED灯，直流电机等），连接即是直流供电输出，断开即是直流供电断开。通过对连接和断开时间的控制，理论上来讲，可以输出任意不大于最大电压值的模拟电压。假设高电平5V，低电平0V。在20%的占空比下，我们得到的模拟电压就是1V。如果占空比为75%，那么得到的模拟电压就是3.75V。

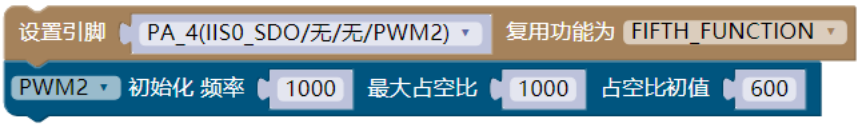


2.PWM占空比调整

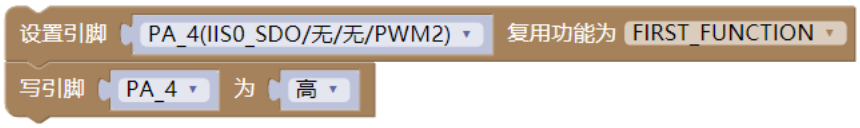
该指令在PWM模块类别指令中，一般用于动态调整PWM。当多次更改PWM时，使用这条指令。

在本案例中，PA_4有两种功能模式GPIO和PWM，设置为第一功能时，通过设置高低电平点亮和熄灭板载灯。当需要调节亮度时，就需要用到PWM功能，设置成复用功能的第五功能。

PWM功能：

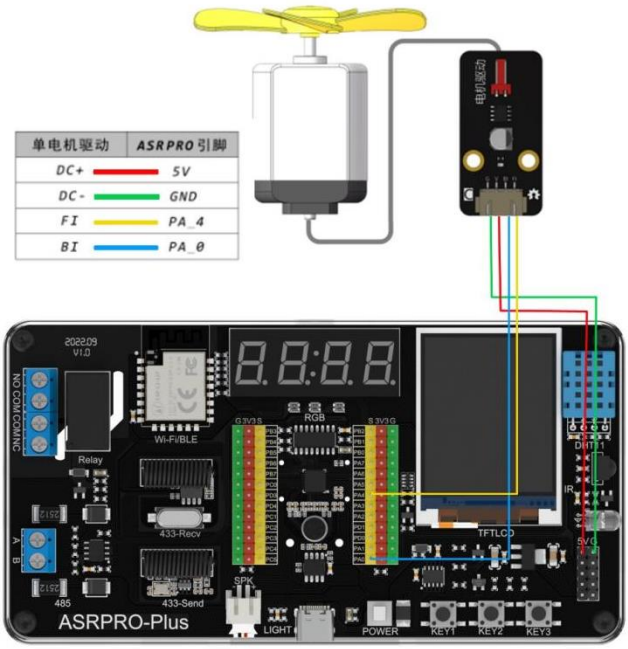


GPIO功能：

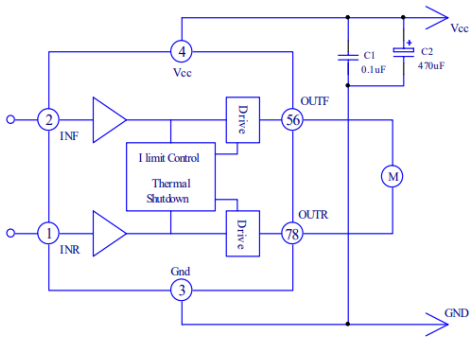


在本范例中，值得注意的是，板载灯是低电平点亮，高电平熄灭。PWM设置时占空比越小，板载灯越亮；占空比越大，板载灯越暗。

PWM除了可以调节灯光亮度外，还常常应用与控制电机转速。电机的转速由电枢电流控制，常见的高效的直流电机调速方式为PWM。单片机的引脚不能直接驱动电机，因此需要增加驱动电路，通过连接单路电机驱动可以实现电机调速的功能，具体连接可参考下图：



单路电机驱动模块，内置双向马达驱动IC，它有两个逻辑输入端，分辨用来控制电机前进/后退和转速。



IC内部电路框图如上，我们可以通过控制BI和FI两个引脚来控制马达的速度和方向，控制逻辑如下：

输入真值表

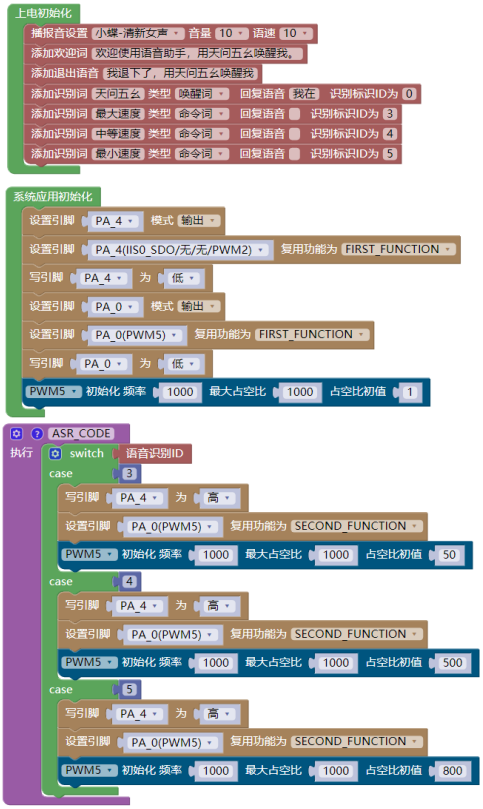
2 脚　前进输入	1 脚　后退输入	5,6 脚 前进输出	7,8 脚　后退输出
H	L	H	L
L	H	L	H
H	H	L	L
L	L	Open	Open

BI（后退输入）和FI（前进输入）管脚与单片机的管脚相连，接收单片机的逻辑高低电平信号。BI和FI引脚的电平变化有四种：高低、低高、高高、低低，从而控制电机的正转、反转、制动和悬空。要控制电机的转速就要用到PWM。

当FI、BI为同一电平时，即都为高或低电平，电机处于制动或悬空状态。

当FI、BI之间存在电压差时，就可以实现电机转动（正转或反转），结合PWM进行调速，电机转动的速度为两个端口PWM的差值，可实现电机的最小速度、中等速度和最大速度，选择FI或BI设置其引脚为PWM功能，并调整占空比即可。

当占空比越小时，电压差越大，转速越快；占空比越大，电压差越小，转速越小。


上电初始化
捕报音设置 小蝶-清新女声 音量 10 语速 10
添加欢迎词 欢迎使用语音助手，用天问五幺唤醒我。
添加退出语音 我退下了，用天问五幺唤醒我
添加识别词 天问五幺 类型 唤醒词 回复语音 我在 识别标识ID为 0
添加识别词 最大速度 类型 命令词 回复语音 识别标识ID为 3
添加识别词 中等速度 类型 命令词 回复语音 识别标识ID为 4
添加识别词 最小速度 类型 命令词 回复语音 识别标识ID为 5
系统应用初始化
设置引脚 PA_4 模式 输出
设置引脚 PA_4(IIS0_SDO/无/无/PWM2) 复用功能为 FIRST_FUNCTION
写引脚 PA_4 为 低
设置引脚 PA_0 模式 输出
设置引脚 PA_0(PWM5) 复用功能为 FIRST_FUNCTION
写引脚 PA_0 为 低
PWM5 初始化 频率 1000 最大占空比 1000 占空比初值 1
ASR_CODE
执行
switch 语音识别ID
case 3
写引脚 PA_4 为 高
设置引脚 PA_0(PWM5) 复用功能为 SECOND_FUNCTION
PWM5 初始化 频率 1000 最大占空比 1000 占空比初值 50
case 4
写引脚 PA_4 为 高
设置引脚 PA_0(PWM5) 复用功能为 SECOND_FUNCTION
PWM5 初始化 频率 1000 最大占空比 1000 占空比初值 500
case 5
写引脚 PA_4 为 高
设置引脚 PA_0(PWM5) 复用功能为 SECOND_FUNCTION
PWM5 初始化 频率 1000 最大占空比 1000 占空比初值 800

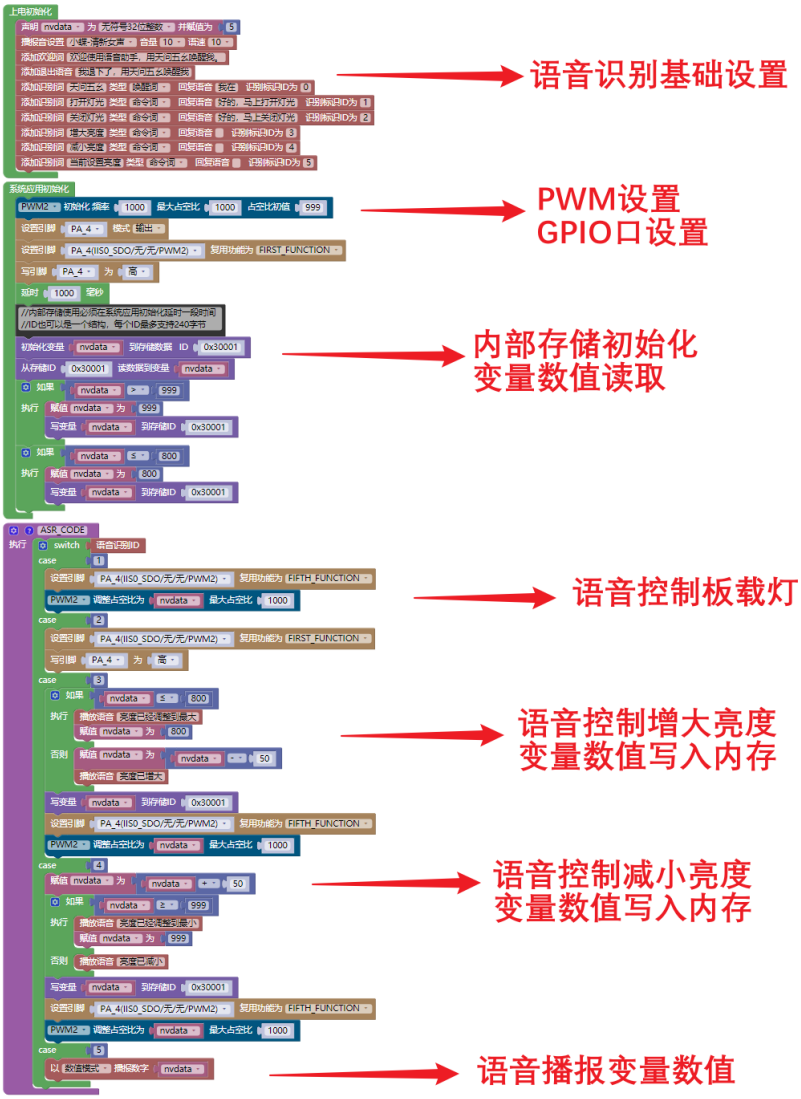
内部存储器

范例1.12 内部存储使用

一、范例功能

本范例通过使用变量控制板载亮度传感器引脚PC_1的PWM值，实现将一个变量在芯片内部存储写入和读取的功能，达成学习内部存储程序编写的目的。本范例适配ASRPRO-Plus开发板，其它开发板需修改PC_1为PC_4。

二、范例分析

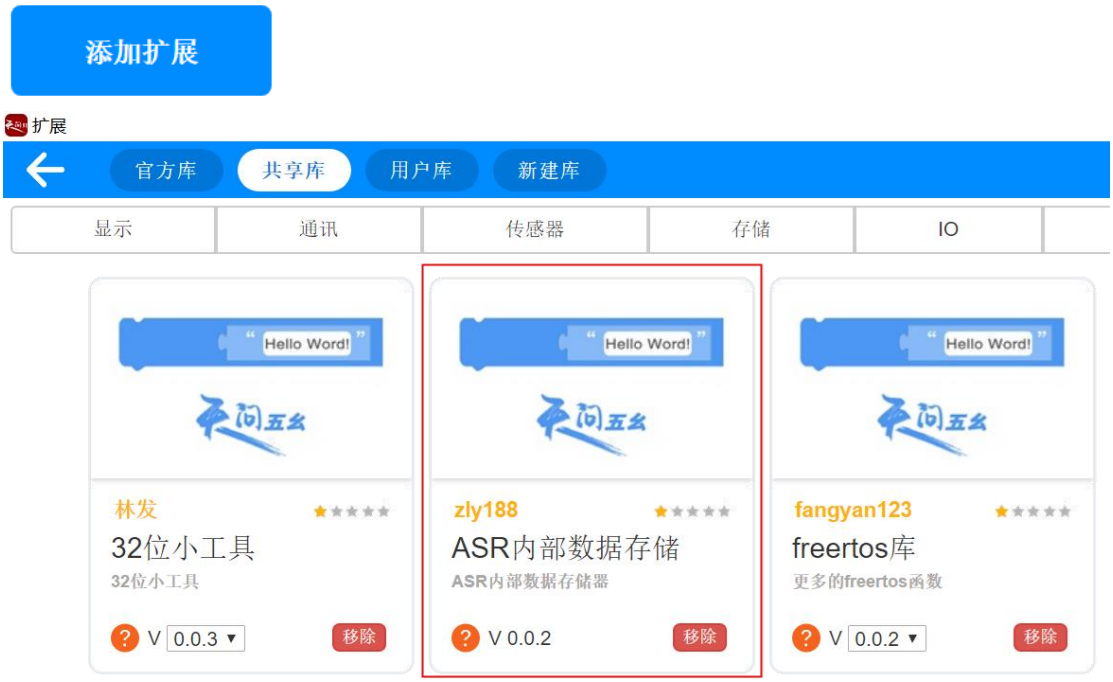


三、范例详解

本案例主要通过变量nvdata内部存储器存储PWM2的占空比值来演示断电保存数据功能。本范例适配所有ASRPRO开发板。

内部数据存储的指令需要在扩展库中添加。点击软件左下角的添加扩展，添加内部存储的扩展库。

点击添加扩展，选择共享库中的ASR内部数据存储扩展库，点击加载。



加载后，扩展类别中就会出现ASR内部数据存储的指令。

1.初始化变量到内部存储器

我们可以使用变量的形式将数据存储到内部存储中。变量的类型可以是无符号整数、数组等各种形式，在本案例中nvdata是一个无符号32位整数。在存储内部，我们以ID的形式区分位置，一般来说从0x30001到0x40000都可以使用。注意在使用这条指令前，需要增加一条延时指令，一般设置延时1000ms。在系统上电后，不能立刻使用内部存储。

2.从内部存储器读取变量

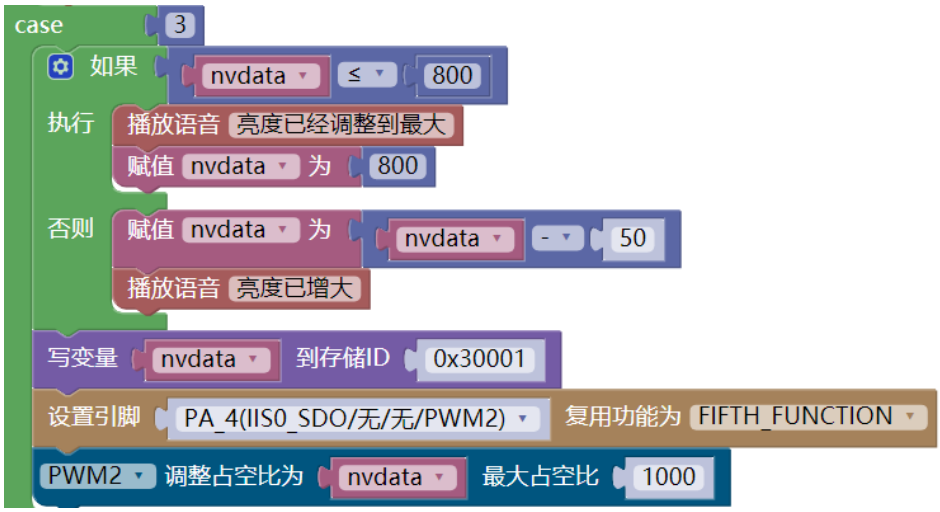
这条指令可以从内部存储ID为0x30001读取数据，用变量nvdata保存。查看右边的代码我们可以发现，这条指令是包括了读取的位置、变量的长度和变量指针。

```
asrpro_nvdata_read(0x30001,sizeof(nvdata),&nvdata);
```

3.写入变量到内部存储器

这条指令就是往0x30001中写入数据。

了解这些之后，我们来看PWM部分。PA_4板载灯，低电平点亮，高电平熄灭。当PA_4引脚作为PWM使用时，占空比越高，高电平比例越高，那么板载灯就会越暗。所以本案例以PWM占空比为999/1000和800/1000为例。当占空比值为999时，亮度调整到最小；当占空比值为800时，亮度调整到最大。



语音控制增大亮度时，如果亮度还没达到最大，就将变量nvdata减少50，以此来增大亮度，同时将这个变量写入到存储器中，设置PWM占空比；如果亮度是800已经达到最大，那么就播报亮度已经调整到最大。

这个内部存储器是将变量映射到FLASH中。断电后，重新上电，可以从存储中读取到该变量的值。例如亮度调整到850，断电后，重新上电开机，变量依旧为850，数据不会丢失。


从存储ID
0x30001
读数据到变量
nvdata

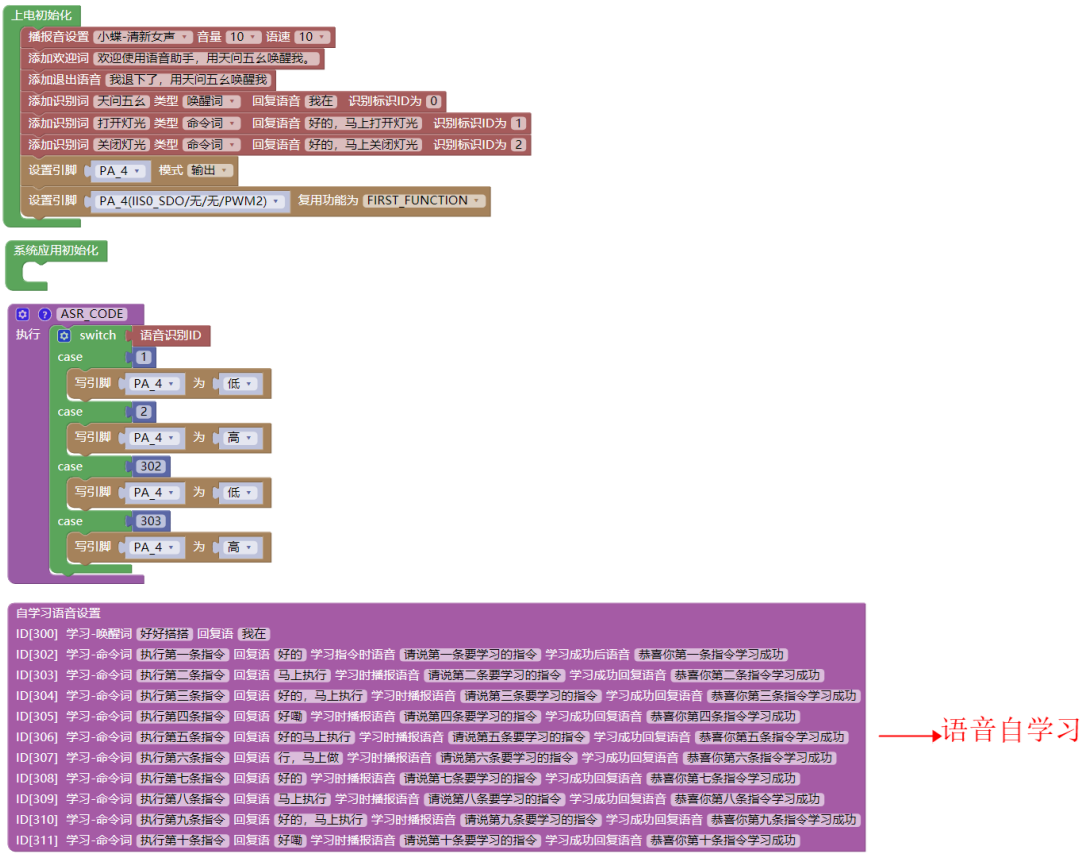
语音自学习

范例1.13 自学习范例

一、范例功能

本范例通过进行自定义学习的语音设置，实现自定义语音自学习的模板，达成制作语音自学习项目的目的。本范例适配所有ASRPRO开发板。

二、范例分析



三、范例详解

语音自学习功能，可以通过学习方式修改唤醒词和命令词，学习的语言可以是中文、方言和其它外语。如我们进行家中客厅灯的控制，我们既可以直接通过程序设置好命令词，例如打开灯光、关闭灯光，也可以通过语音自学习，通过方言来控制。尤其是家中的老人，普通话经常说不标准，就可以通过语音自学习的方式来对灯光进行控制。

我们先以范例代码为例，生成模型并编译下载程序。下载完成后ASRPRO就拥有了语音自学习功能。

（一）学习唤醒词

首先用默认的唤醒词唤醒语音助手，然后说“学习唤醒词”，根据提示来学习新的唤醒词。

提示：学习状态中，保持安静

说“小智小智”

提示：请再说一次！

再次说“小智小智”

提示：指令学习成功

就可以使用学习过的唤醒词“小智小智”来唤醒语音助手！

这样学习的唤醒词可以是普通话，也可以是方言，但是注意学习过程中说的两次唤醒词需要保持一致。

（二）学习命令词

用唤醒词（默认或已学习的）唤醒语音助手，然后说“学习命令词”，根据提示来学习新的命令词。

在自学习状态下学习指令，语音会进行提示：

学习状态中，保持安静，请说第一条要学习的指令。

说“打开客厅灯”

提示：请再说一次！

再次说“打开客厅灯”

提示：指令学习成功，请说出第二条要学习的指令

…… (继续学习即可)

或者使用“退出学习”来退出当前的学习状态。

（三）删除唤醒词和命令词

用唤醒词（默认或已学习的）唤醒语音助手，然后说出“我要删除”，根据提示来删除新学习的唤醒词/命令词。

提示：删除唤醒词还是命令词

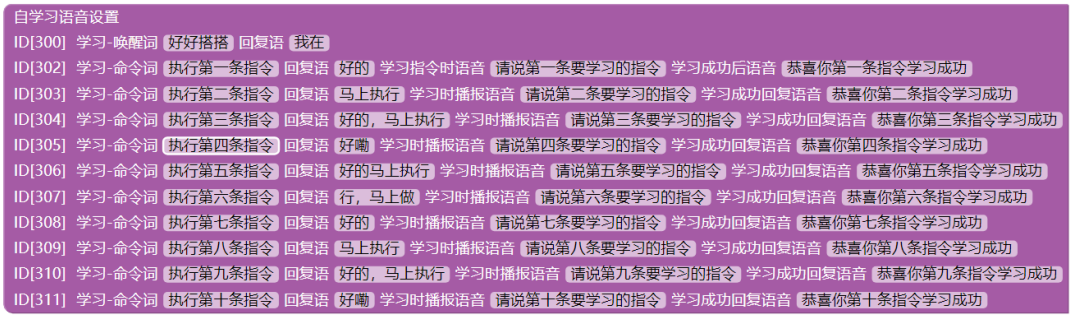
删除命令词：删除学习过的命令词

删除唤醒词：删除学习过的唤醒词

全部删除：删除学习过的唤醒词和命令词

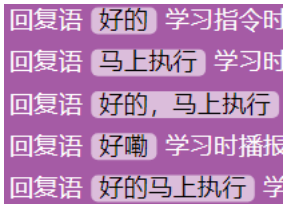
删除后会提示删除成功。

接下来我们来学习如何对程序的语音内容进行修改。



如果你想修改学习过程中的提示语音，请修改下方的这两部分指令。

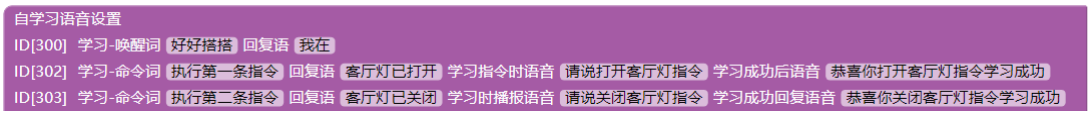
如果你想修改语音识别后的回复语，可以修改这一部分。



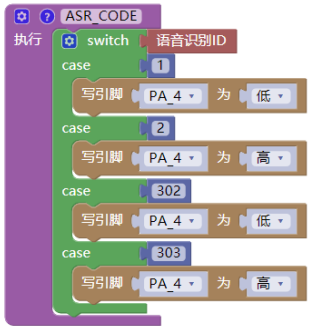
这里我以改造家中的客厅灯为例，修改自学习语音设置指令。

我想让语音ID302，语音ID303，分别代表打开客厅灯、关闭客厅灯，但我又想用方言来控制，同时我希望在学习过程中语音能够进行提示，现在学习的语音控制的是什么部分。

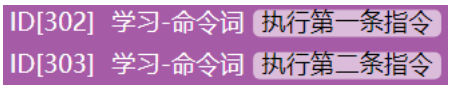
我对自学习语音设置指令进行了如下修改。



修改程序，重新生成语音模型并编译下载后，发现自学习的语音对话已经改变。结合ASR_CODE函数中的程序，我们就可以实现通过自学习的方言来对灯光进行控制。



当然我们也可以发现，在图形化指令，学习-命令词的位置，还有一处位置可以修改。例如ID302和303的“执行第一条指令”和“执行第二条指令”。当你用普通话说这两句命令词时，同样可以控制客厅灯的打开和关闭。修改这一部分指令，可以让常规的语音识别和语音自学习新添加的命令词同时进行控制。



如果你希望能够修改整个语音自学习的内容，你也可以直接修改代码部分。在语音识别ID400-421之间，你可以直接修改语音自学习的提示语。例如我对学习唤醒词和学习命令词的提示语进行修改，通过字符编程，将保持安静修改为大家请保持安静，重新生成模型，进行编译下载。修改后，语音自学习的提示语就会改变。

```
//{ID:400,keyword:"命令词",ASR:"学习唤醒词",ASRTO:"学习状态中，大家请保持安静"}
//{ID:401,keyword:"命令词",ASR:"学习命令词",ASRTO:"学习状态中，大家请保持安静"}
```

详细的语音提示语如下所示，可以按照个人喜好进行修改。

```
//{ID:400,keyword:"命令词",ASR:"学习唤醒词",ASRTO:"学习状态中，保持安静"}
//{ID:401,keyword:"命令词",ASR:"学习命令词",ASRTO:"学习状态中，保持安静"}
//{ID:402,keyword:"命令词",ASR:"重新学习",ASRTO:"学习状态中，保持安静"}
//{ID:403,keyword:"命令词",ASR:"退出学习",ASRTO:"好的"}
//{ID:404,keyword:"命令词",ASR:"我要删除",ASRTO:"删除唤醒词还是命令词"}
//{ID:405,keyword:"命令词",ASR:"删除唤醒词",ASRTO:"删除成功"}
//{ID:406,keyword:"命令词",ASR:"删除命令词",ASRTO:"删除成功"}
//{ID:407,keyword:"命令词",ASR:"退出删除",ASRTO:"马上退出"}
//{ID:408,keyword:"命令词",ASR:"全部删除",ASRTO:"好的"}
//{ID:409,keyword:"命令词",ASR:"指令学习成功",ASRTO:"学习成功，请再说一次"}
//{ID:410,keyword:"命令词",ASR:"学习失败",ASRTO:"学习失败，再说一次"}
//{ID:411,keyword:"命令词",ASR:"注册成功",ASRTO:"指令学习成功"}
//{ID:412,keyword:"命令词",ASR:"超上限",ASRTO:"学习数量超上限"}
//{ID:413,keyword:"命令词",ASR:"删除成功",ASRTO:"删除成功"}
//{ID:414,keyword:"命令词",ASR:"删除失败",ASRTO:"删除失败"}
//{ID:415,keyword:"命令词",ASR:"正在删除",ASRTO:"正在删除"}
//{ID:416,keyword:"命令词",ASR:"未找到命令词",ASRTO:"找不到要删除的命令词"}
//{ID:417,keyword:"命令词",ASR:"学习成功",ASRTO:"学习完成"}
//{ID:418,keyword:"命令词",ASR:"失败",ASRTO:"学习失败，退出学习模式"}
//{ID:419,keyword:"命令词",ASR:"请再说一次",ASRTO:"请再说一次"}
//{ID:420,keyword:"命令词",ASR:"语音太短",ASRTO:"语音太短了"}
//{ID:421,keyword:"命令词",ASR:"指令重复",ASRTO:"命令词和默认有相同，请换命令词"}
```

注意命令词学习后，断电后也不会删除。如果需要修改请删除后重新学习。目前最多支持32条自定义语音。如果需要可以通过字符编程对源码进行修改，这里不作详细说明。

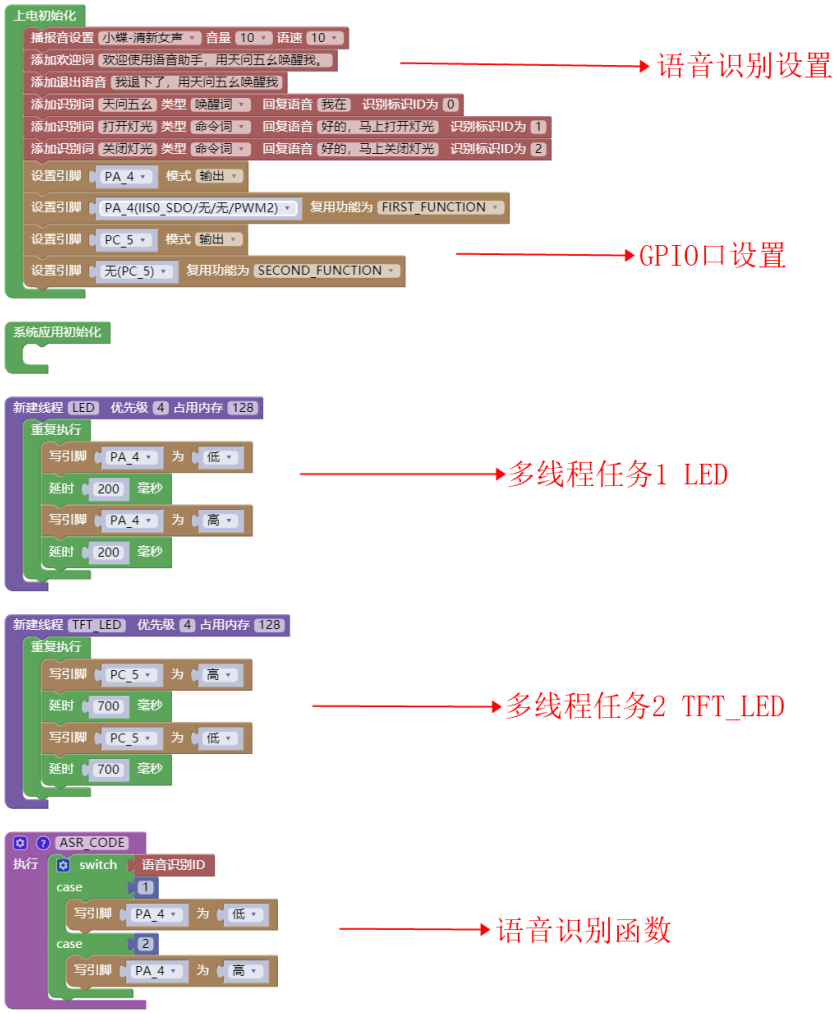
多线程

范例2.1 第一个多线程程序

一、范例功能

本范例通过对为什么使用多线程、多线程任务的定义与切换进行详解，学习多线程的基础知识，实现两个线程同时运行的功能，学习专业模式下多线程程序的基础编写。

二、范例分析



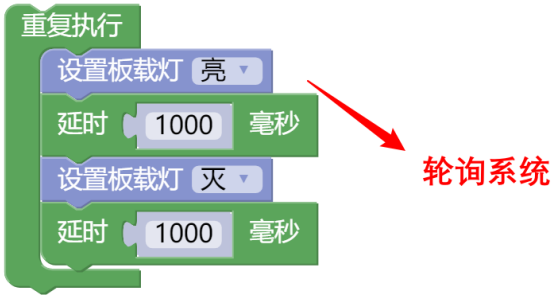
三、范例详解

ASRPRO底层框架是基于FreeRTOS实现的。所谓RTOS，指的是Real Time Operating System，也就是实时操作系统。当我们刚接触到嵌入式开发中时，往往先接触的是单片机编程。这里面多是裸机编程，没有加入任何RTOS。我们之前编写的程序也大多如此。在学习编写多线程的程序前，了解我们为什么要使用多线程，多线程和之前的编程模式有什么区别，是相当重要的一件事。

为了了解多线程的使用场景和意义所在，我们先来讲解下单片机编程中的裸机系统和多任务系统这几种软件结构的区别。

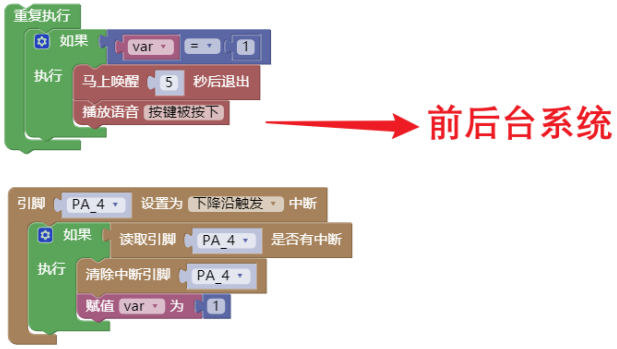
裸机系统通常分成轮询系统和前后台系统。

轮询系统即是在裸机编程的时候，先初始化好相关的硬件，然后让主程序在一个死循环里面不断循环，顺序地做各种事情。轮询系统是一种非常简单的软件结构，通常只适用于那些只需要顺序执行代码且不需要外部事件来驱动的就能完成的事情。如果只是实现LED电平翻转、串口输出、液晶显示等这些操作，那么使用轮询系统将会非常完美。但是，如果加入了按键操作等需要检测外部信号的事件，用来模拟紧急报警，那么整个系统的实时响应能力就不会那么好了。当外部按键被按下，相当于一个警报，这个时候，需要立马响应，并做紧急处理，而这个时候程序刚好执行到一部分，这部分内容需要执行的时间比较久，久到按键释放之后都没有执行完毕。这就相当于丢失了一次事件。足见，轮询系统只适合顺序执行的功能代码，当有外部事件驱动时，实时性就会降低。



相比轮询系统，前后台系统是在轮询系统的基础上加入了中断。外部事件的响应在中断里面完成，事件的处理还是回到轮询系统中完成，中断在这里我们称为前台，main函数里面的无限循环我们称为后台。在顺序执行后台程序的时候，如果有中断来临，那么中断会打断后台程序的正常执行流，转而去执行中断服务程序，在中断服务程序里面标记事件，如果事件要处理的事情很简短，则可在中断服务程序里面处理，如果事件要处理的事情比较多，则返回到后台程序里面处理。相比于轮询系统，程序的实时响应能力会高很多。

例如我们可以在中断里置标志位，在大循环里根据标志位状态进行处理。



我们之前编写的程序多是按照轮询系统或者前后台系统。

相比前后台系统，多任务系统的事件响应也是在中断中完成的，但是事件的处理是在任务中完成的。在多任务系统中，任务跟中断一样，也具有优先级，优先级高的任务会被优先执行。当一个紧急的事件在中断被标记之后，如果事件对应的任务的优先级足够高，就会立马得到响应。相比前后台系统，多任务系统的实时性又被提高了。

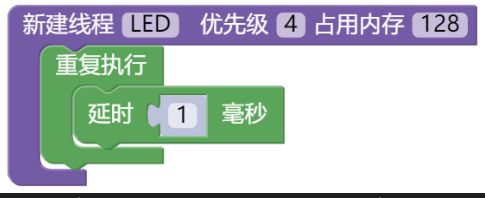
相比前后台系统中后台顺序执行的程序主体，在多任务系统中，根据程序的功能，我们把这个程序主体分割成一个个独立的，无限循环且不能返回的小程序，这个小程序我们称之为任务。每个任务都是独立的，互不干扰的，且具备自身的优先级，它由操作系统调度管理。加入操作系统后，我们在编程的时候不需要精心地去设计程序的执行流，不用担心每个功能模块之间是否存在干扰。加入了操作系统，我们的编程反而变得简单了。整个系统随之带来的额外开销就是操作系统占据的那一丁点的FLASH和RAM。现如今，单片机的FLASH和RAM是越来越大，完全足以抵挡RTOS那点开销。

对于这三种系统，我们不能一锤子吹定孰优孰劣。它们是不同时代的产物，在各自的领域有应用。但是按照下方三种系统的特点来看，多任务系统，显然更符合ASRPRO离线语音识别的开发与应用。我们要逐渐适应这种软件结构的程序编写。

模型	事件响应	事件处理	特点
轮询系统	主程序	主程序	轮询响应事件，轮询处理事件
前后台系统	中断	主程序	实时响应事件，轮询处理事件
多任务系统	中断	任务	实时响应事件，实时处理事件

而在一个程序中，这些独立运行的程序片段叫作“线程”。线程是操作系统能够进行运算调度的最小单位。从软件或者硬件上实现多个线程并发执行的技术则叫做多线程。而在我们ASRPRO的开发应用中，我们也可以把线程称作任务。

新建一个名为LED的线程，我们查看代码可以发现，这实际上是一个Task的新建。

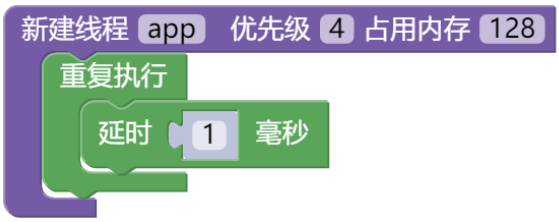



```
xTaskCreate(LED,"LED",128,NULL,4,NULL);
```

掌握以上基础，了解为什么使用多线程后，我们通过案例2-1来了解如何进行任务的创建和切换。

任务是一个独立的函数，函数主体无限循环且不能返回。

找到多线程类别区域的指令。使用这条指令可以新建一个任务。



这条指令可以修改任务的名称、优先级、占用内存以及函数主体。

其中名称推荐使用英文，优先级的范围默认是0-6，实际可通过configMAX_PRIORITIES进行配置，数值越大优先级越高，占用内存最高512*2，需要根据程序 整体合理分配。建议先设置最大，再慢慢减小，查看运行情况是否正常来确定最终大小。重复执行内部的延时建议大于2ms。如果不加延时，线程很容易堵塞，影响其他线程的运行。

在多任务系统中，每个任务都是独立的，互不干扰的，所以要为每个任务都分配独立的栈空间。这个栈空间通常是一个预先定义好的全局数组，也可以是动态分配的一段内存空间，但它们都存在于RAM中。系统为了顺利的调度任务，为每个任务都额外定义了一个任务控制块。任务的栈，任务的函数实体，任务的控制块最终需要联系起来才能由系统进行统一调度。而这个联系的工作就由我们刚刚创建的这个函数xTaskCreate()来实现。

在main函数中将硬件和RTOS系统先初始化后，会创建一个启动任务后就启动调度器，然后在启动任务里面创建各种应用任务，当所有任务都创建成功后，启动任务把自己删除。这对应了右边代码中的vTaskDelete。

```
  xTaskCreate(LED,"LED",128,NULL,4,NULL);
  xTaskCreate(TFT_LED,"TFT_LED",128,NULL,4,NULL);
  vTaskDelete(NULL);
```

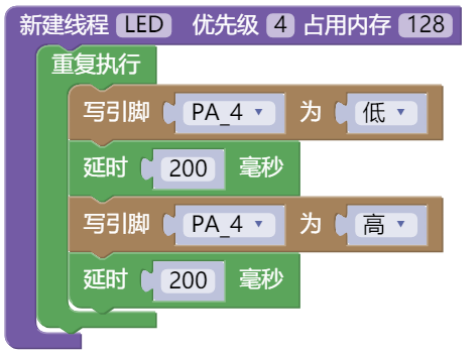
系统应用初始化，在操作系统运行之后，初始化相应的外围硬件，完成后会把新建线程以及将启动任务删除。

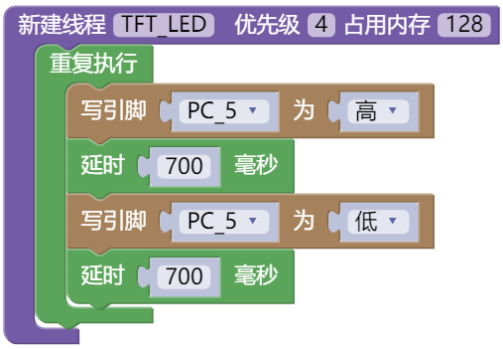
```
void hardware_init(){
  xTaskCreate(LED,"LED",128,NULL,4,NULL);
  xTaskCreate(TFT_LED,"TFT_LED",128,NULL,4,NULL);
  vTaskDelete(NULL);
}
```

不建议在系统应用初始化开始位置用重复执行，一直while循环中，对线程新建和启动任务删除产生影响。所以我们一般不在系统应用初始化开始位置增加重复执行。

```
void hardware_init(){
  while (1) {

  }
  xTaskCreate(LED,"LED",128,NULL,4,NULL);
  xTaskCreate(TFT_LED,"TFT_LED",128,NULL,4,NULL);
  vTaskDelete(NULL);
}
```

在本范例中，一共有两个任务，分别是LED和TFT_LED。





这两个任务会被先添加到就绪列表中，表示任务以及就绪，系统可以随时调度。然后我们通过调度器，实现任务的切换，即从就绪列表里面找到优先级最高的任务。

接着我们又发现，这些任务函数的重复执行中，都添加了延时指令。为什么要添加延时呢？能不能不添加？

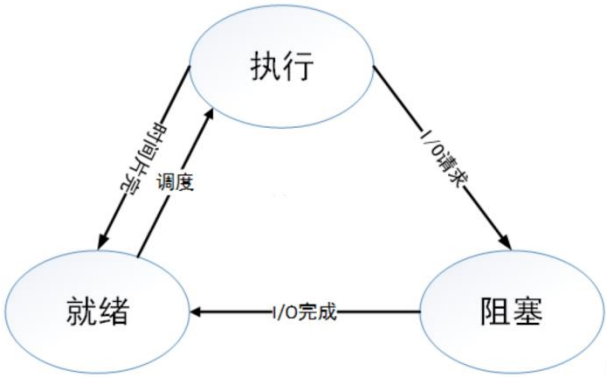
接下来我们通过对阻塞延时和空闲任务进行了解，进一步了解多线程的工作模式。

RTOS的有一个很大的优势，就是它充分利用了CPU的资源。我们一般把任务函数中的延时叫做阻塞延时。阻塞延时指的是，当一个任务进入阻塞延时后，任务会放弃CPU的使用权，CPU可以去做其他的事情，例如找其他的任务做；当任务延时时间到，这个任务会重新获取CPU使用权，任务继续运行。这样可以把CPU的资源都利用起来，而不是干等着。

那么当所有任务都进入阻塞延时怎么办？如果没有其他任务可以运行，RTOS都会为CPU创建一个空闲任务，这个时候CPU就运行空闲任务。在FreeRTOS中，空闲任务是系统在【启动调度器】的时候创建的优先级最低的任务。

一般来说，任务有三个状态，即就绪状态，运行状态，阻塞状态，具体如下图所示。

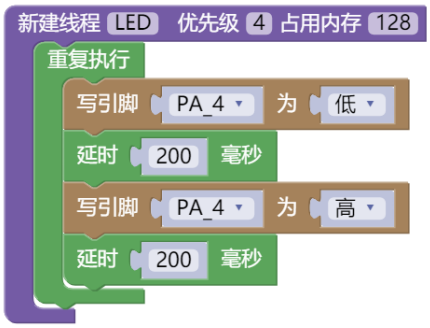
运行态：任务占用CPU，并在CPU上运行；
就绪态：任务已经具备运行条件，但是CPU还没有分配过来；
阻塞态：任务因等待某件事发生而暂时不能运行；

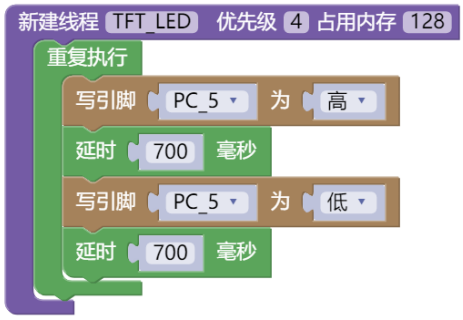


对于任务的切换，实际上是在一个中断函数中进行的，在ASRPRO中这个中断间隔为2ms，也就是每隔2ms就进行一次判断。如果当前判断的任务处于就绪状态，就切换到该任务；如果不在就绪状态中，就切换到下一个任务。关于就绪状态的判断，实际上是阻塞延时有一个delay的函数，阻塞延时结束了，任务就从阻塞状态变成了就绪状态。

有这些知识的基础后，我们重新回到当前案例中。创建了两个任务后，任务调度器就开始工作，在2ms的中断中进行判断，开始调度这些任务。LED和TFT_LED两个任务先后被调度（同优先级支持时间片，这里不做详解，对人的感观基本没有影响），板载灯和彩屏背

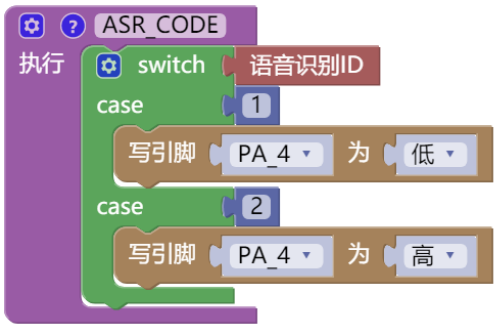
光灯被打开，接着两个任务都进入到阻塞延时。任务调度器发现任务列表中的这些任务都进入了阻塞模式，就开始调度空闲任务。任务LED的延时时间较短，只有200ms，200ms结束后重新进入就绪状态。此时任务调度器在中断中检测到任务LED处于就绪状态，就重新对他进行调用，任务LED的主函数继续执行，板载灯被关闭，任务LED重新进入阻塞延时。而此时彩屏背光灯还是处于打开，TFT_LED任务处于阻塞延时。





多线程的运作模式，让这两个任务，能够互不干扰地运行下去。而在我们人眼中，板载灯和彩屏背光以不同的频率一直闪烁。

语音识别的函数ASR_CODE，实际上也在一个线程之中。这个线程包括了音频的采集、语音的识别等各种功能。有兴趣的用户可以自行查看源码。

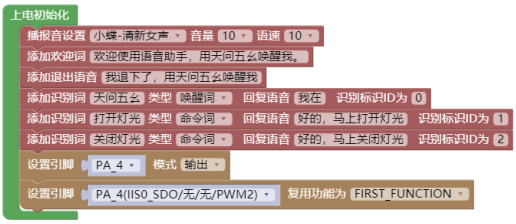


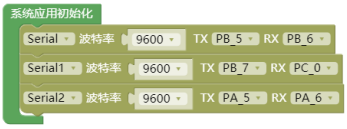
范例2.2 串口多线程发送程序

一、范例功能

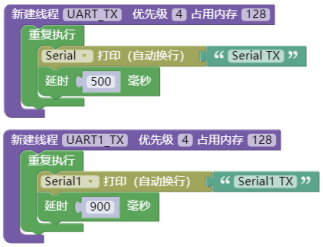
本范例通过使用多线程来发送串口消息，实现在多线程模式下串口输出消息的功能，达成ASRPRO通过串口与外部进行通讯的目的。

二、范例分析

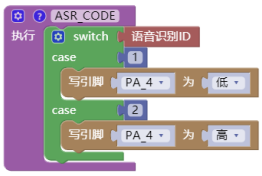




串口初始化设置



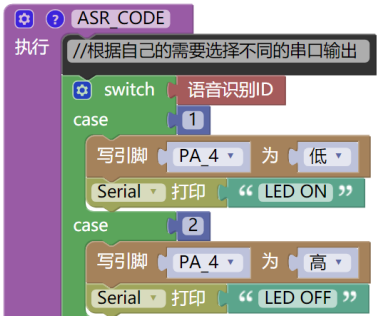
多线程发送串口消息



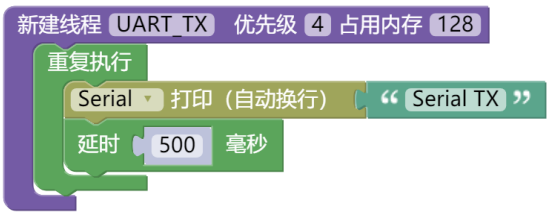
三、范例详解

我们在之前的范例1.8、范例1.9和范例2.1中已经简单学习过串口和多线程的基础知识。

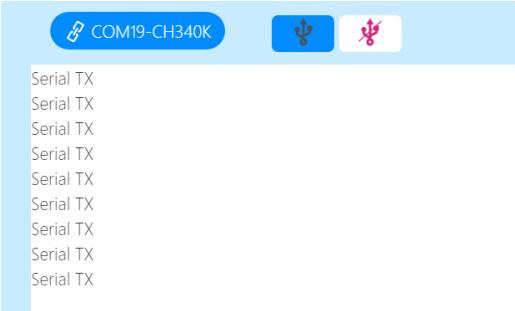
在以往的程序编写中，我们发送串口消息，通常在语音识别的函数ASR_CODE中。



我们在范例2.1第一个多线程程序的中，也简单介绍了，ASR_CODE这个函数，本身也是在一个线程之中。但是为了应对各种的开发应用，我们还是需要学习，如何新建一个线程，在线程中使用串口发送消息。



我们这里以新建一个线程UART_TX为例，优先级为4，占用内存128字节，来向串口0持续发送消息“Serial TX”。查看串口监视器，输出消息如下。我们可以看到，这个多线程任务会每隔500毫秒向串口0发送消息，不会受到其他线程的影响。你可以写一个新的线程，让板载灯间隔1秒进行闪烁。两个线程之间不会互相干扰，串口仍然可以顺利发送消息。



注意，如果需要使用另外两个串口，建议每个串口都使用一个单独的线程来处理。

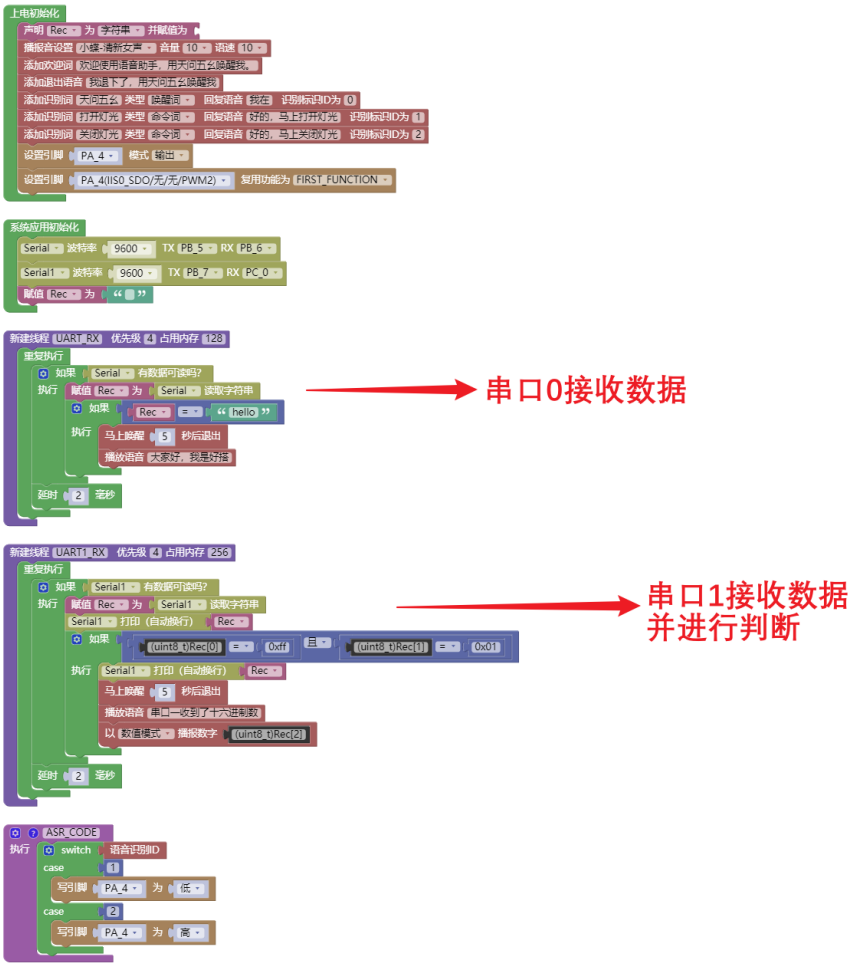
ASRPRO与其他外接设备进行串口通讯请参考附录二。

范例2.3 串口多线程接收程序

一、范例功能

本范例通过使用多线程来接收串口消息，实现ASRPRO通过串口接收其他设备发送的消息并进行判断的功能，达成ASRPRO通过串口与外部进行通讯的目的。

二、范例分析



三、范例详解

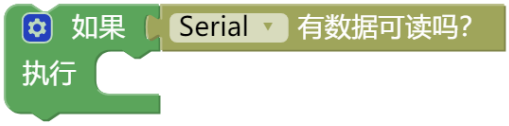
本范例是ASRPRO的串口通过多线程的方式接收消息，并对串口消息进行判断。我们在之前的范例中，学习了如何通过串口发送字符串和16进制数。本范例也以这两种类型的串口消息为基础，来对接收到的串口消息进行处理。

我们从学习串口类别区域的指令来剖析如何对串口消息进行处理。

1.判断串口是否有数据


```
if(Serial.available() > 0)
```

这条指令可以判断串口是否有数据可读，一般配合判断语句使用，例如：

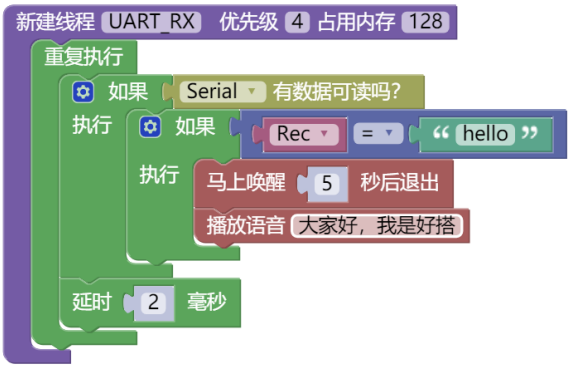


2.串口读取字符串


```
Serial.readString()
```

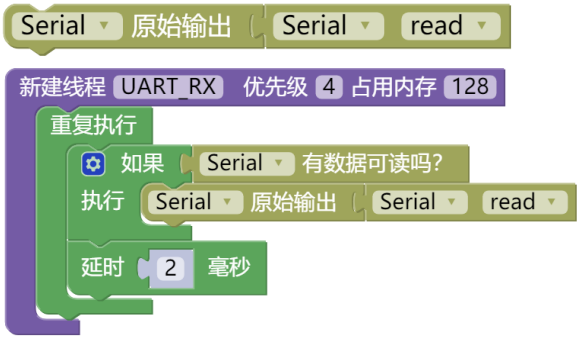
这条指令可以读取一帧数据。。一般用一个变量存储，然后对变量进行判断。

当确定串口接收到的消息是字符串时，就可以用范例中的写法。当串口0接收到消息后进行判断，如果接收的字符串是“hello”，那么就唤醒并播报语音。

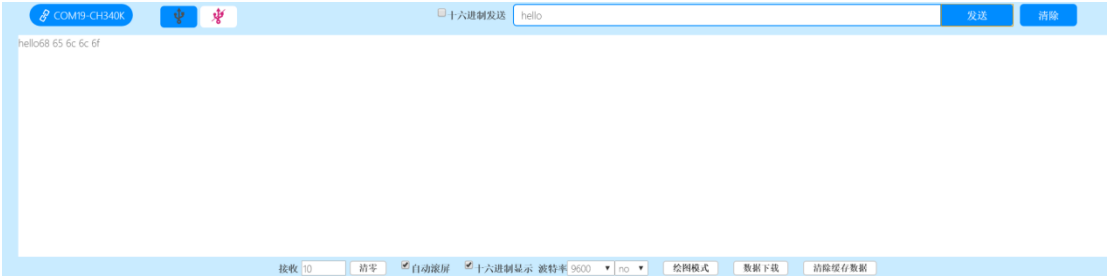


3.串口原始读取

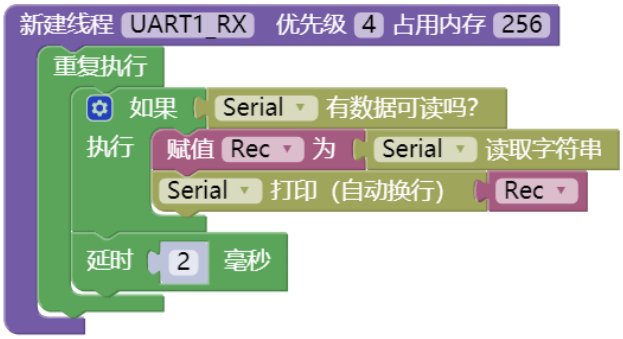

```
Serial.read()
```



当我们通过串口监视器输入字符串“hello”时，指令2和指令3这两种形式都可以输出hello。如果勾选16进制显示，那么就会输出该字符串对应的16进制数，结果如下所示。



串口如何接收16进制数类型的消息，并进行处理。对于16进制数，串口的接收依旧可以和字符串类型一样进行处理。我们新建一个字符串类型的变量Rec，16进制数据仅是整数的一种表现形式，我们把十六进制数据赋到字符串的内存，就是相当于把一个整数写到内存地址中。但在实际应用中16进制数据一般建议使用Serial.read()来一个个数据接收处理，因为Serial.readString()对于特殊的十六进制数可能会被截取调。



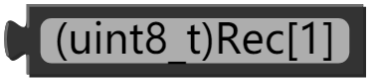
接下来我们对这些16进制数进行判断。例如ff 00 01。这里有三个16进制数，分别是0xff，0x00，0x01。每个16进制字符是一个4位整数，而0xff，这种两个连续的16进制数字，则是一个字节。在串口监视器中0x一般可以不显示，但是在判断中，我们需要填写完整的16进制数。

4.字符串变量读取

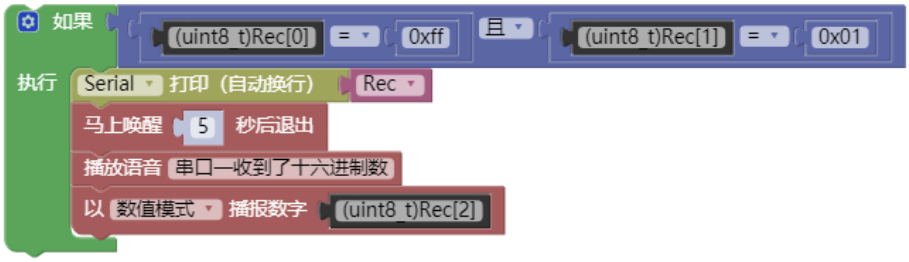
这条指令在读写寄存器中。 

这条指令代表着从寄存器中读取字符串类型变量Rec存储的第一个字节，并把他转换位uint8_t数组，也就是无符号一个字节的整型。

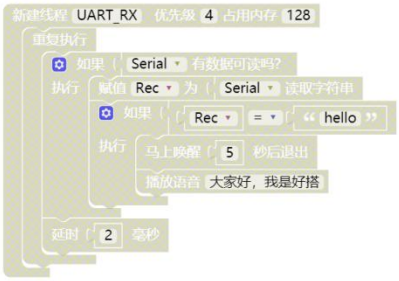
如果要获取这个变量的第二个字节，则可以将指令改成如下所示。



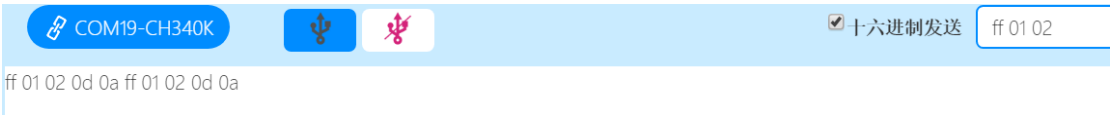
通过这样的形式，我们可以对串口接收到的一串16进制数进行读取和判断。



我们将范例代码中的UART1_RX进行修改，使用串口0进行测试。注意，每个串口要单独用一个线程进行处理。这里我们选择将带有串口0指令的另一个线程禁用。



勾选16进制发送和16进制显示，当我们输入 ff 01 02 时，串口会输出ff 01 02 0d 0a，其中0d 0a是自动换行。

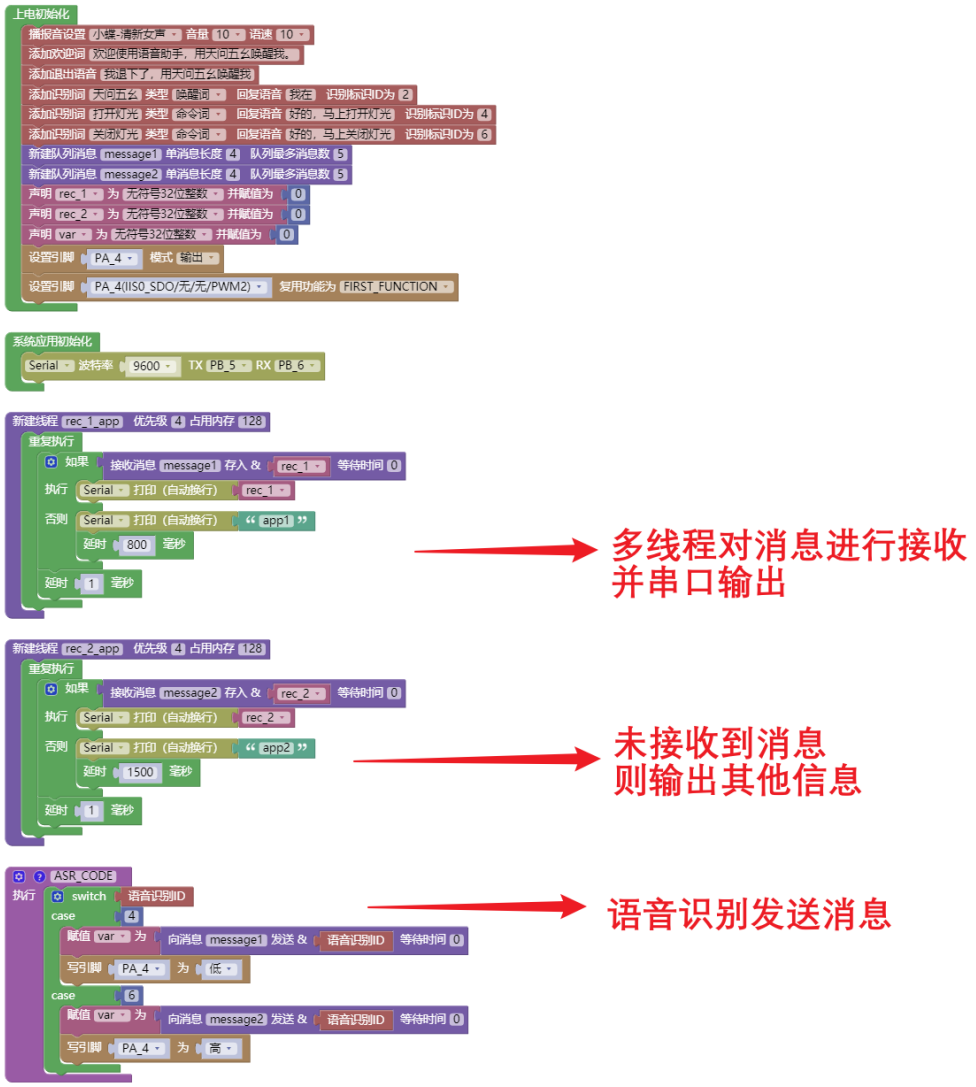


范例2.4 多线程使用--消息队列

一、范例功能

本范例通过使用消息队列，实现多个线程之间互相通信的功能，达成学习消息队列在多线程中应用的目的。

二、范例分析

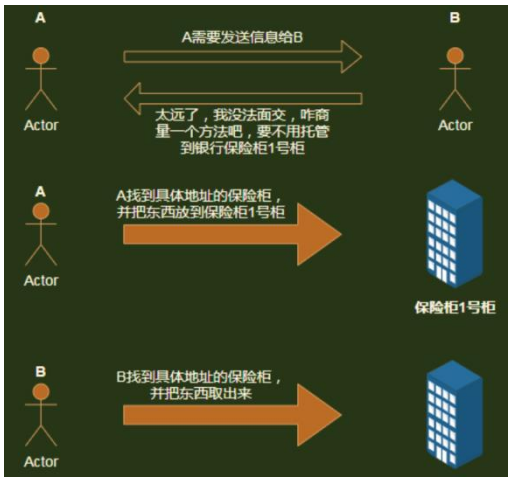


三、范例详解

消息队列，又称作队列，是一种常用于任务间通信的数据结构。队列可以在任务与任务间、中断和任务间传递信息，实现了任务接收来自其他任务或中断的不固定长度的消息。

我们可以通过下图比较直观的了解消息队列的作用。A要把信息发送个B，因为某些原因，当时A不能把东西面交给B，这时候他们商量，你把东西放到一个地方先寄存起来，我到时候去那个地方去取。A按照协议商定的，就把信息放到协议约定好的地方，并约定好了

提货方式。对于B而言，要想成功拿到A的东西，就必须要保证去同一个地址，并且只有1号柜才是给自己的信息。这个地址就是消息队列，保险1号柜中的东西就是消息。



通过消息队列服务，任务或中断可以将一条或多条消息放入消息队列中。同样，一个或多个任务可以从消息队列中获得消息。当有多个消息发送到消息队列时，通常是将先进入消息队列的消息先传给任务，也就是说，任务先得到的是最先进入消息队列的消息，即先进先出原则（FIFO）。

对与获取消息来说，一个任务能够从任意一个消息队列接收和发送消息。多个任务也能够从同一个消息队列接收和发送消息，但是能不能多个消息都发给同一个任务呢？例如我同时把消息1和消息2都发给了任务1，那么任务1接收的到底是哪一个消息呢？

我们使用的消息队列一般不是属于某个任务的队列，在很多时候，我们创建的队列，是每个任务都可以去对他进行读写操作的，但是为了保护每个任务对它进行读写操作的过程，我们必须有阻塞机制，在某个任务对它读写操作的时候，必须保证该任务能正常完成读写操作，而不受后来的任务干扰。如上图所示，如果任务B想要从消息队列获取消息，必须一条消息一条消息的来获取，而不能一次性全部拿完。

也就是说，当任务B先接到了消息1，此时任务会进入阻塞态，只有当任务B接收了消息1/判断没接收到不等了/一直等到接收到了消息1，才会进入就绪态，重新接收新的消息。通过这张图我们也能进一步了解先进先出原则。消息1是先进入队列的，当读取消息时，就先接收消息1，随后才是消息2。如果对这里有不理解的可以回顾多线程范例2.1中的三种状态。

假如有多个任务阻塞在一个消息队列中，那么这些阻塞的任务将按照任务优先级进行排序，优先级高的任务将优先获得队列的访问权。

1.创建消息队列

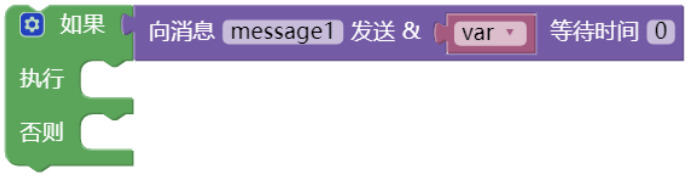

```
message1=xQueueCreate(5,4);
```

这条指令可以新建一个消息队列。其中message1代表着队列的名称。我们需要为消息队列分配足够消息存储空间，空间的大小为队列长度*单个消息大小。如上图的指令，单消息的长度为4，代表着单消息4个字节，也就是32位，例如你要发送的消息是32位的，就设置单消息长度位4。队列长度，也就是队列最多消息数。一般最大为10。单消息长度和队列最多消息数，要看内存和实际存储的消息来进行设置

2.向消息队列发送消息

这条指令可以向消息队列message1发送消息。发送的消息可以是一个变量，可以是一个数组。注意发送消息使用的是拷贝的形式，也就是数据的复制，而不是传递的数据地址。也就是一旦消息发送完成，这个变量可以再次被使用。

一般使用方式如下所示，可以判断向消息message1是否发送消息成功，发送成功或未成功输出不同信息。当然我们也可以使用范例中的方法，使用一个变量，但是最好对变量的值进行判断，确认是否发送成功。

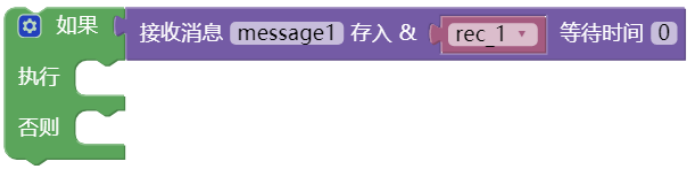


等待时间指，当接收消息的这个队列message1满了的时候，最多可以阻塞的时间，也就是可以等待的时间。如果等待时间设置为0，如果队列已满，那么调用会马上返回。如果成功发送，返回pdTRUE，发送失败返回pdFALSE。

```
    if (rst != pdTRUE)
    {
        mprintf("send msg err[%d]\n",codec_index);
    }
    return 0;
```

3.从消息队列接收消息

这条指令可以从消息队列message1接收消息。消息队列中的消息遵循先进先出的原则。当使用这条指令时，需要一个变量，用来存放这些消息。这个变量可以是局部变量，在线程中定义。我们可以通过对这条指令进行判断，来判断是否接收到消息。



和发送消息类似，接收消息也有一个等待时间，但两者并不相同。当队列中的消息是空时，读取消息的任务将被阻塞，用户可以指定阻塞的任务时间xTicksToWait，这个变量就是我们要填入的等待时间。在这段时间中，如果队列为空，该任务将保持阻塞状态以等待队列

数据有效。当队列中有新消息时，被阻塞的任务会被唤醒并处理新消息；当等待的时间超过了指定的阻塞时间，即使队列中尚无有效数据，任务也会自动从阻塞态转为就绪态。

这条指令会按照顺序将队列中的所有消息都读取一次。当消息被读取后队列就会被删除。

注意使用这条指令时，如果是多个消息队列，建议每个消息队列单独放到一个线程中，接收消息的处理在一个线程中处理，不要在同一个线程中放两个消息队列的处理。例如本范例，就将两个消息队列放在了两个线程中，线程rec_1_app处理消息队列message1，线程rec_2_app处理消息队列message2。

运行程序。当读取消息队列，发现都没有消息存入时，两个线程会向串口0发送app1和app2，间隔时间分别为800毫秒和1500毫秒。当说打开灯光时，语音识别后就向消息队列message1发送snid的数值，也就是4。线程rec_1_app读取了消息队列message1的消息，并将消息的值存放到变量rec_1中，随后消息删除，查看下一个消息，直到队列中没有消息就会出队，判断接收消息完成之后串口输出信息。另一个线程同理。

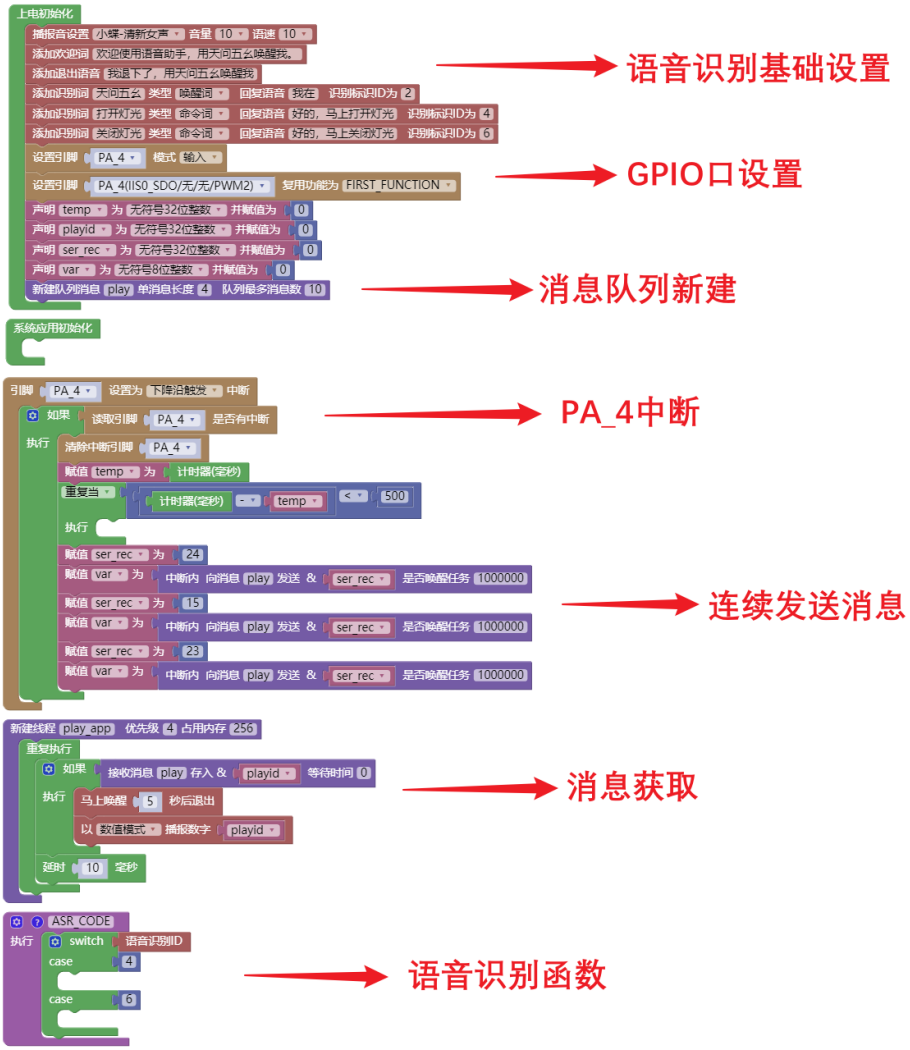
```
app1
app2
4
app1
app1
app2
app1
app1
6
app2
app1
app1
app2
```

范例2.5 PA_4中断发送消息

一、范例功能

本范例通过使用中断向多线程发送消息，实现线程和中断之间互相通信的功能，达成学习中断在多线程中应用的目的。

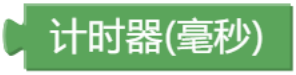
二、范例分析



三、范例详解

我们先来查看中断部分发送消息的代码。当PA_4引脚被按下，从高电平变成低电平，下降沿触发中断后，设置500毫秒后向消息队列连续发送三个消息。注意中断中不建议加延时，所以这里使用计时器来处理。

1.计时器



这条指令在控制类别指令中。计时器在ASRPRO上电后开始计时。通常可以通过变量进行存储，通过调用来计算两次计时的差值，接着对差值进行判断。

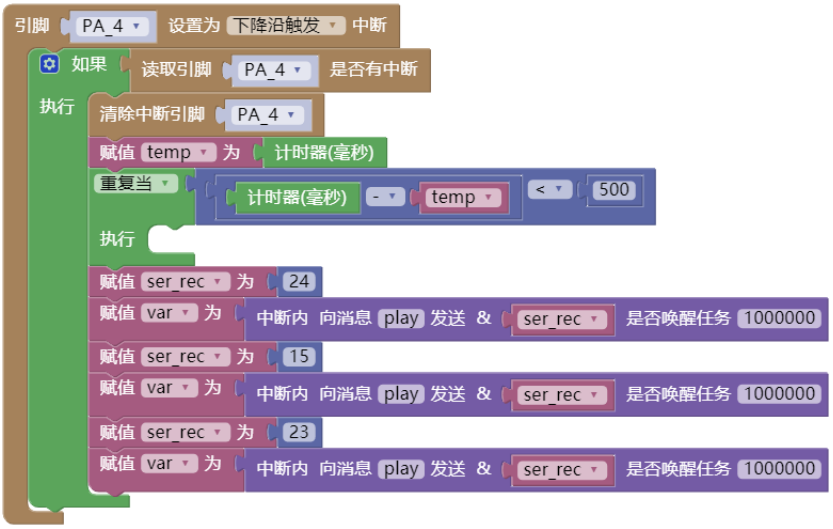
2.中断发送消息

这条指令在多线程类别指令中，用与中断和任务之间进行通信。任务在工作的时候，如果此时发生了一个中断，无论中断的优先级是多大，都会打断当前任务的执行。这个指令的用法和之前发送消息的指令类似，一般新建一个变量用于判断是否发送成功，第二个参数用于修改消息队列名称，第三个参数是消息。

中断中发送消息不允许带有阻塞机制的，因为发送消息的上下文环境是在中断中，不允许有阻塞的情况。

第四个参数，是否唤醒任务，在这里其实是PxHigherPriorityTaskWoken参数。这个参数可以获取是否要进行任务的切换。如果将PxHigherPriorityTaskWoken设置为pdTRUE，那么如果消息发送到队列后导致任务解除阻塞，且解除阻塞的任务的优先级高于当前运行的任务，就会在中断结束前请求任务切换。这里输入0设置的是pdFALSE，输入值非0设置为pdTRUE。

在本范例中，我们在中断中连续发送了多个消息。



我们通过中断，连续发送了三个消息到消息队列play中。接着我们可以用一个新的线程来接收消息。当接收消息后，就用语音把消息的值播报出来。由于消息遵循先进先出的原则，所以语音会先后播报24，15，23。

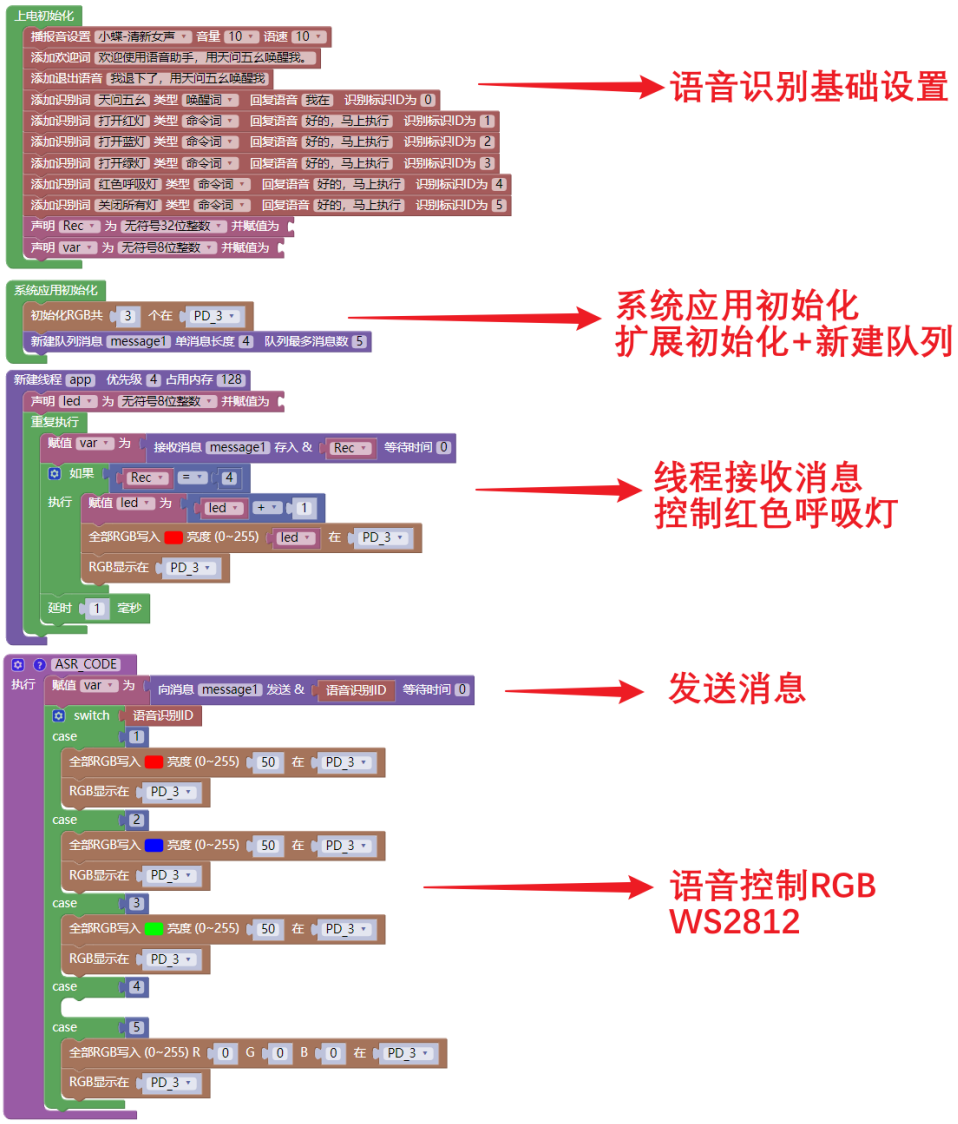
扩展库使用

范例3.1 语音控制WS2812彩灯

一、范例功能

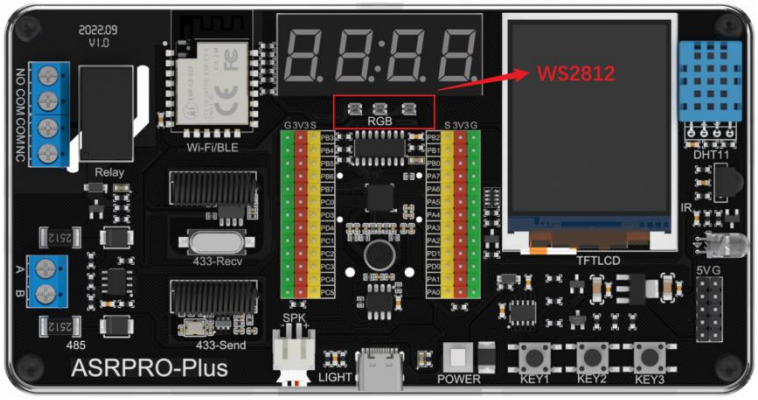
本范例通过使用语音控制和多线程，实现语音控制RGB显示各种颜色、呼吸灯效果和关闭的功能，达成学习编写WS2812程序的目的。

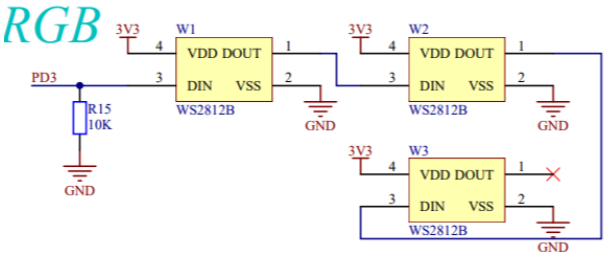
二、范例分析



三、范例详解

本范例介绍如何编写WS2812模块的程序，使用语音识别来控制RGB的亮灭、呼吸灯效果、彩虹循环效果等。ASRPRO-Plus的中央部分就外接了WS2812模块。下方是RGB的实物图和电路原理图。





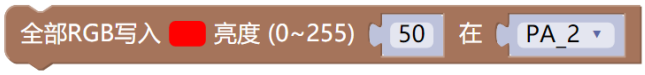
我们在编写WS2812的程序前，需要先添加扩展库。点击添加扩展，找到官方扩展库中的WS2812，版本选择最新，点击加载。注意，编译下载带扩展库的程序时，默认使用最新版本！如果用了之前的扩展库版本保存了程序，扩展库会切换到之前的版本；如果扩展库版本有更新，此时需要重新更改扩展库版本，然后点击保存。

加载后我们就可以在扩展类别中，找到WS2812对应的指令。

1.RGB初始化

这条指令可以设置RGB灯珠的个数和引脚。以Plus外接的WS2812为例，一共有三颗灯珠，引脚在PD_3，则指令要设置成如下所示。扩展库的初始化一般放系统应用初始化中。

2.RGB写入



RGB写入一共有四条指令。其中前两条可以设置任意一个RGB灯珠的RGB值，后两条则是设置全部RGB灯珠的颜色和亮度。颜色可以点击红色方块自由选择一些基础颜色，也可以自己设置具体的RGB数值，亮度与设置的RGB值有关。最后要更改的引脚设置注意与初始化、实际连接相同即可。RGB写入要配合RGB显示一起使用。

```
ASR_WS2812_27.setBrightness(50);
ASR_WS2812_27.pixel_set_all_color(255,0,0);
```

3.RGB清除

RGB清除指令。这条指令可以将所有RGB灯熄灭，要配合RGB显示一起使用。

```
ASR_WS2812_27.pixel_clear();
```

4.RGB显示

RGB显示生效指令。必须使用这条指令，RGB的写入和清除才会生效。查看代码我们可以发现，写入是set，清除是clear，都是属于内部的设置。只有当使用show时，才会体现到硬件上，让之前的指令都生效。RGB显示生效一般都放在所有RGB设置指令的下方。

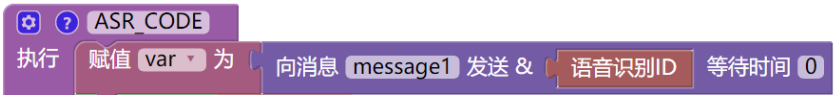
```
ASR_WS2812_27.pixel_show();
```

学习基础指令后，我们再来仔细学习范例代码。整个范例包括了语音控制灯光显示红色、绿色、蓝色、关闭，以及使用消息队列的方式控制红色呼吸灯。

语音控制部分较为简单，对语音识别ID进行判断，设置RGB颜色即可。

这里主要对如何使用消息队列的方式进行控制作分析。

首先是消息的发送。我们在范例2.1第一个多线程程序中曾介绍，函数ASR CODE其实也是在一个线程中。这个线程包括了相当多的内容，其中就有对音频的解析。也就是当函数ASR CODE前，我们就以及获取到了语音识别ID，ASR CODE函数只是对snid进行一个判断并进行相应的处理。我们将消息的发送指令放在ASR CODE函数的一开始，也就是代表着，每当识别到语音，就将语音识别ID发送到消息队列1。



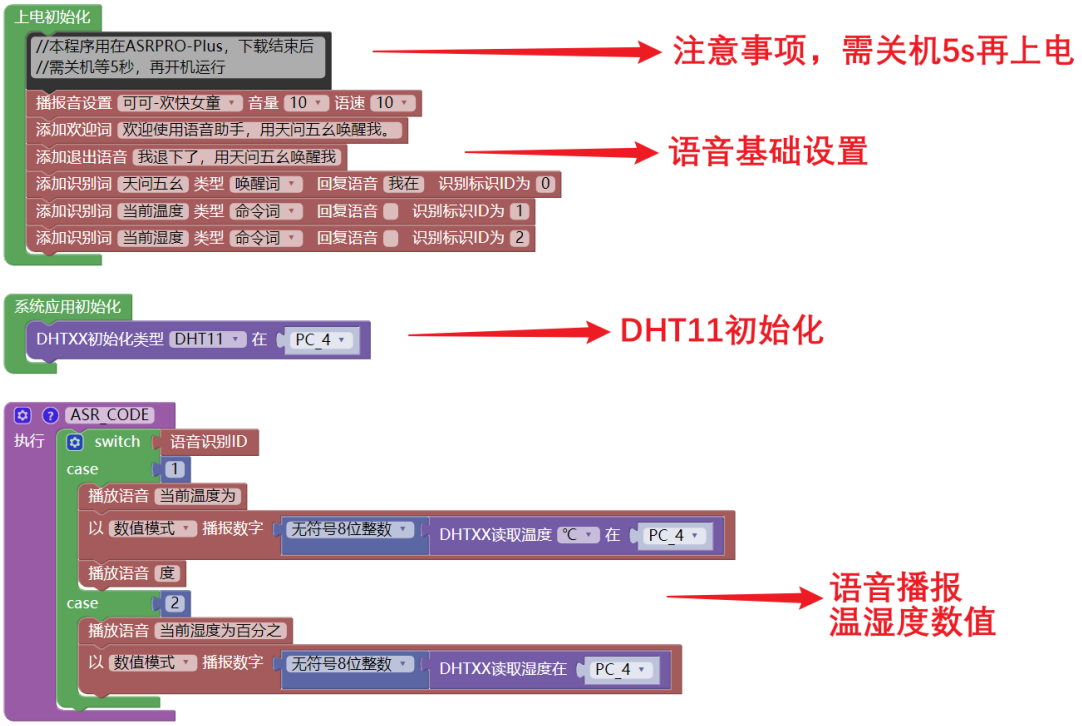
然后我们在线程app中接收消息。当我们识别到消息的值为4时，就开始红色呼吸灯的程序。我们可以直接在线程的一开始声明一个变量led，把RGB设为红色，亮度的值设为led，让每隔1ms累加1。注意led是一个无符号8位整数，最大是255，也就是二进制的11111111。此时再累加是，led会变成00000000并高位溢出。所以这个红色呼吸灯的效果是从最暗到最亮，然后直接变到最暗，又从最暗到最亮。

范例3.2 语音播报DHTXX温湿度

一、范例功能

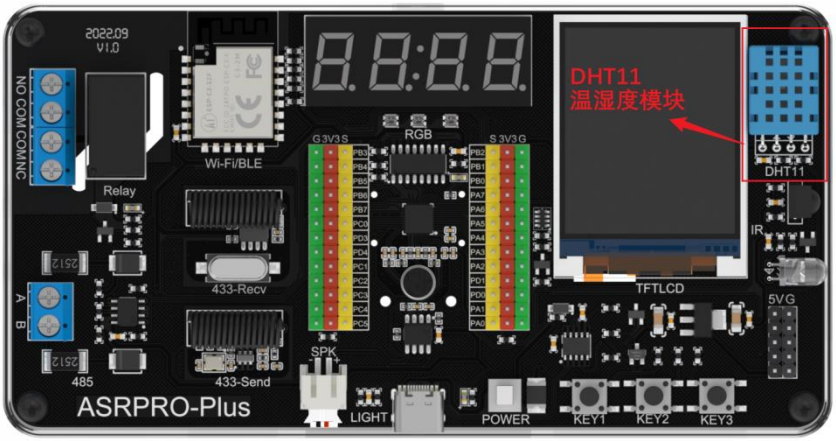
本范例通过语音播报DHT11温湿度传感器的温度值和湿度值，达成学习DHTXX模块程序编写的目的。

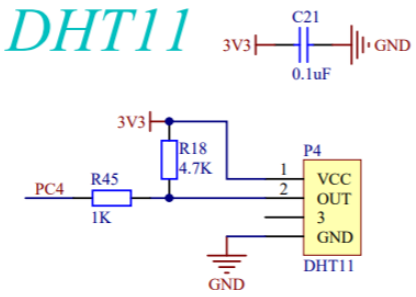
二、范例分析



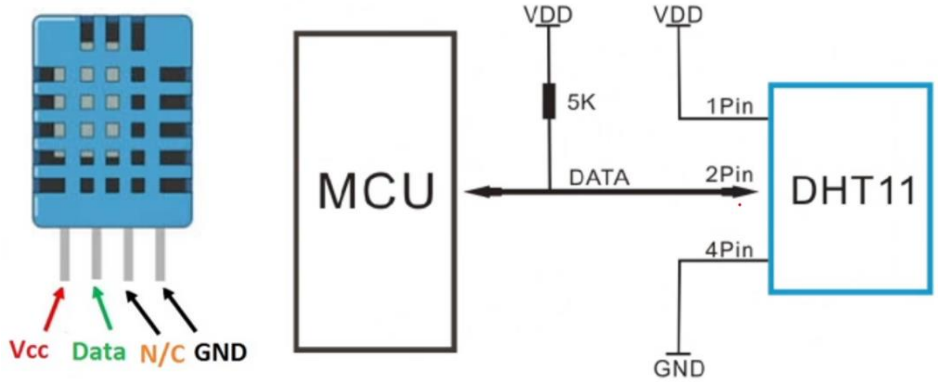
三、范例详解

本范例介绍如何编写DHTXX模块的程序，使用语音播报的方式来得知温度值和湿度值。ASRPRO-Plus的右上角部分就外接了一个DHT11模块。下方是DHT11温湿度模块的实物图和电路原理图。





DHT11温湿度模块内部有四个引脚，如果外接，连接方式如下，注意需要接一个5k电阻。



我们在编写DHTXX温湿度传感器的程序前，需要先添加扩展库。点击添加扩展，找到官方扩展库中的DHTXX，版本选择最新，点击加载。

加载后我们就可以在扩展类别中，找到WS2812对应的指令。

1.DHTXX初始化

这条指令可以设置DHTXX模块的类型和引脚。通过查找DHTXX模块类型的引用，我们可以发现，DHTXX扩展库可以支持DHT11、DHT22、DHT21（AM2301）这几种类型。

```
#define DHT11 11
#define DHT22 22
#define DHT21 21
#define AM2301 21
```

2.DHTXX读取温度

这条指令可以读取DHTXX模块的温度值，单位是摄氏度（C°）或者华氏度（F°）

```
read_temperature(0)
```

查看代码，输入0表示FALSE，代表摄氏度；输入1表示TRUE，代表华氏度。

```
float DHTxx::convertCtoF(float c) {
    return c * 9 / 5 + 32;
}
```

华氏度根据摄氏度c按照公式进行转化。

3.DHTXX读取湿度

这条指令可以读取DHTXX模块的湿度值，单位是%RH。这个湿度代表着是相对湿度，表示空气中的绝对湿度与同温度和气压下的饱和绝对湿度的比值，也就是指某湿空气中所含水蒸气的质量与同温度和气压下饱和空气中所含水蒸气的质量之比。

当一个地方没有明显的干空气或湿空气进来时，实际含水量是相对稳定少变的，但是理论最大含水量，这个分母项是会变的。例如清晨，气温低，就会导致分母降低，相对湿度变大，会形成大雾；放晴后分母逐渐变大，相对湿度降低，雾就散了。

下方是温度值和相对湿度值的性能表。例如相对湿度值范围在5-95%RH，在25C°下精度为±5；温度范围在-20-60C°，精度是±2C°。

表 1 相对湿度性能表

参数	条件	min	type	max	单位
量程范围		5		95	%RH
精度[1]	25℃		±5		%RH
重复性			±1		%RH
互换性		完全互换			
响应时间[2]	1/e(63%)		<6		S
迟滞			±0.3		%RH
漂移[3]	典型值		<±0.5		%RH/年

表 2 温度性能表

参数	条件	min	type	max	单位
量程范围		−20		60	℃
精度[1]	25℃		±2		℃
重复性			±1		℃
互换性		完全互换			
响应时间[2]	1/e(63%)		<10		S
迟滞			±0.3		℃
漂移[3]	典型值		<±0.5		℃/年

DHT11温湿度传感器采用的是典型的单总线通信。

名称	单总线格式定义
起始信号	微处理器把数据总线（SDA）拉低一段时间至少18ms（不超过30ms），通知传感器准备数据
响应信号	传感器把数据总线（SDA）拉低83us，再接高87us以响应主机的起始信号
数据格式	收到主机起始信号后，传感器一次性从从数据总线（SDA）串出40位数据，高位先出
湿度	湿度高位是湿度整数部分数据，湿度低位是湿度小数部分数据
温度	温度高位是温度整数部分数据，温度低位是温度小数部分数据，且温度低位Bit8为1则表示负温度，否则为正温度。
校验位	校验位=湿度高位+湿度低位+温度高位+温度低位

示例一：接收到的40位数据为：

0011 0101　0000 0000　0001 1000　0000 0100　0101 0001

湿度高8位　湿度低8位　温度高8位　温度低8位　校验位

计算：

0011 0101+0000 0000+0000 1000+0000 0100=0101 0001

接收数据正确：

湿度：0011 0101（整数）=53%RH　0000 0000（小数）=0.0%RH　合计53.0%RH

温度：0001 1000（整数）=24℃　0000 0100（小数）=0.4摄氏度　合计24.4℃

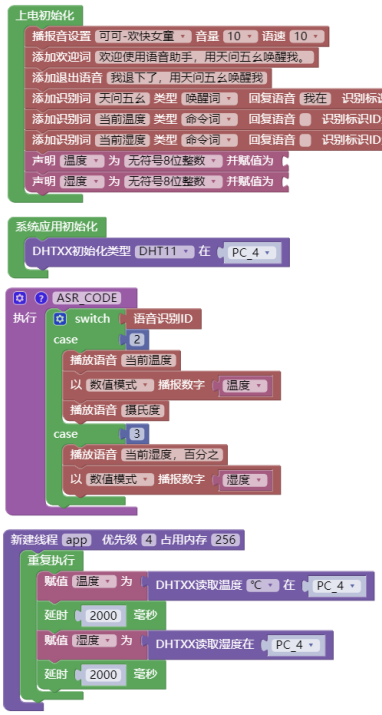
◎特殊说明：

当温度低于0℃时，温度数据的低8位的最高位置为1。

示例：-10.1℃表示为 0000 1010 1000 0001

整数部分为10，小数部分为0.1，低8位的最高位置为1，代表是负数，所以是-10.1℃

每当我们使用读取温度或者读取湿度的指令时，都会去对温度或湿度进行一次数据的读取。例如DHT11温湿度传感器，建议每隔2s获取一次温度值和湿度值，以此来确保数据的准确性。我们也可以通过多线程的方式，在线程中获取温湿度传感器的数值。程序如下所示。



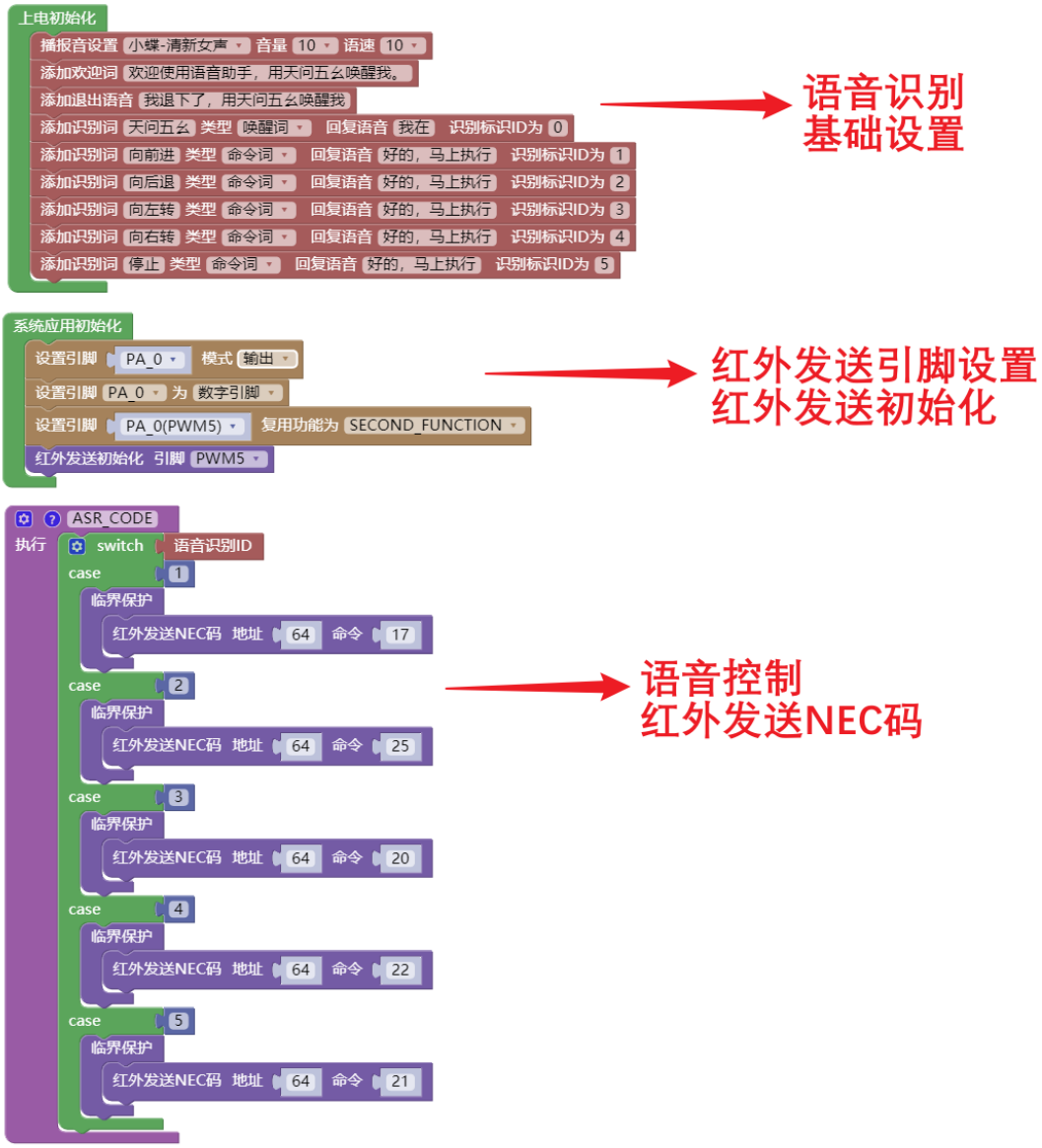
无论用哪种写法，都建议编译下载后，关机等待5s，然后再重新上电，不然会出现温度值和湿度值都变为0的情况。

范例3.3 红外发射NEC码

一、范例功能

本范例通过红外发射模块，实现语音控制红外发送NEC码的功能，达成学习红外发射模块和NEC协议的目的。

二、范例分析

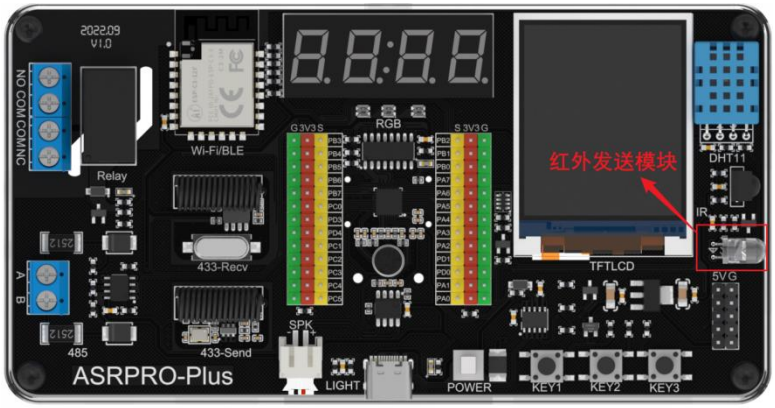


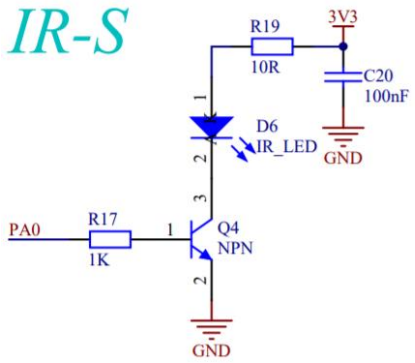
三、范例详解

红外线的应用，从日常生活到军工产品都有。如：红外线开关、医疗保健、遥控器、红外接口、防盗装置、红外遥感以及红外侦察等。日常生活中接触最多的是红外线遥控器，被广泛使用在各种类型的家电产品上（如遥控开关、智能开关等）。

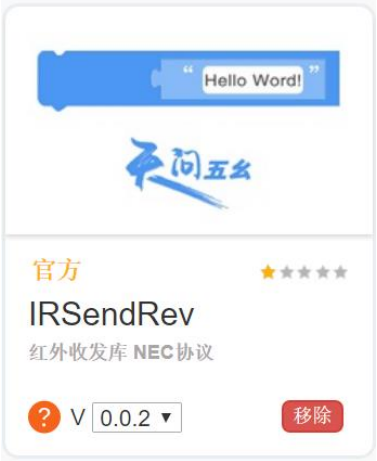
本范例介绍如何编写红外发射模块的程序，使用语音来控制红外发送NEC码。

ASRPRO-Plus的右侧部分就外接了一个红外发射模块。下方是红外发射模块的实物图和电路原理图。





我们在编写红外发送的程序前，需要先添加扩展库。点击添加扩展，找到官方扩展库中的IRSendRev，版本选择最新，点击加载。



加载后我们就可以在扩展类别中，找到红外发射对应的指令。

1.红外发送初始化

目前红外遥控器广泛使用的两种遥控码格式，一种是NEC Protocol 的PWM(脉冲宽度调制) 标准，一种是Philips RC-5 Protocol 的PPM(脉冲位置调制) 标准。

ASRPRO的红外扩展库通过NEC协议来实现红外发射和接收，驱动红外模块发送红外信号来控制对应的设备。

红外发送模块是一款38KHz红外线发射传感器，可发射标准38KHz的调制信号。

这条指令可以设置红外发送使用的是哪一个PWM。PWM的频率默认设置为38kHz。

```
    // 1. 载波配置
    enableIROut(38000);
```

在实际设置中，查看代码发现，这个初始化并没有自动设置启用GPIO口功能，所以我们要先启用PA_0引脚的PWM功能，再将红外发送初始化设置为PWM5。

```
void IRSendRev::begin(pwm_base_t channel)
{
    ir_send_channel = channel;
}
```

2.红外发送初始化

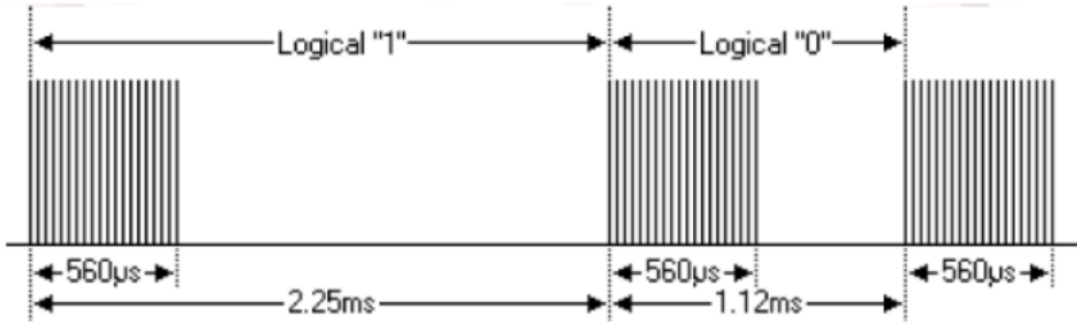
这条指令可以让红外发射模块发送NEC码，包括地址与命令，两者均是一个字节数据，也就是8位。

为了能够深刻理解地址address和命令command，我们来了解一下NEC协议。

NEC协议是众多红外遥控协议的其中一种，市面上大部分的红外遥控器都集成了一种或多种编码。

对于红外发射来说，它用发射红外载波的占空比来代表“0”和“1”。

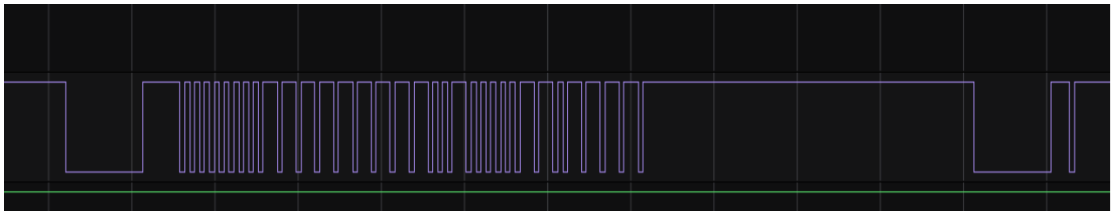
下方是NEC协议逻辑1和逻辑0的表示图：



其中逻辑1为2.25ms，脉冲时间560us；逻辑0为1.12ms，脉冲时间560us。

我们可以根据脉冲时间的长短来进行解码。

下方是我截取的一段红外信号。



NEC编码的一帧（通常按一下遥控器按钮所发送的数据）由引导码、地址码及数据码组成，如下图所示。把地址码及数据码取反的作用是加强数据的正确性。

在NEC协议中，首先出现的是引导码，包括9ms的高电平脉冲和4.5ms的低电平脉冲，接下来就是8bit的地址码，而后是8bit的地址码的反码（刚刚说过这是为了校验数据是否出错），最后是8bit的命令码以及命令码的反码。

我们也可以查询irsendrev.send_nec()中去查看红外发送的这一过程。

```
// 1. 载波配置
enableIROut(38000);

// 2. 发送引导码
mark(NEC_HDR_MARK);
space(NEC_HDR_SPACE);
// 3. 发送地址
for (i=0;i<8;i++)
{
    if (address & 0x01)
    {
        mark(NEC_BIT_MARK);
        space(NEC_ONE_SPACE);
    }
    else
    {
        mark(NEC_BIT_MARK);
        space(NEC_ZERO_SPACE);
    }
    address >>= 1;
}
// 4. 发送地址反码
for (i=0;i<8;i++)
{
    if (inverse_address & 0x01)
    {
        mark(NEC_BIT_MARK);
        space(NEC_ONE_SPACE);
    }
    else
    {
        mark(NEC_BIT_MARK);
        space(NEC_ZERO_SPACE);
    }
    inverse_address >>= 1;
}
```

```
// 5. 发送命令
for (i=0;i<8;i++)
{
    if (command & 0x01)
    {
        mark(NEC_BIT_MARK);
        space(NEC_ONE_SPACE);
    }
    else
    {
        mark(NEC_BIT_MARK);
        space(NEC_ZERO_SPACE);
    }
    command >>= 1;
}

// 6. 发送命令反码
for (i=0;i<8;i++)
{
    if ((inverse_command) & 0x01)
    {
        mark(NEC_BIT_MARK);
        space(NEC_ONE_SPACE);
    }
    else
    {
        mark(NEC_BIT_MARK);
        space(NEC_ZERO_SPACE);
    }
    inverse_command >>= 1;
}
mark(NEC_BIT_MARK);
space(1);
```

3.临界保护

这条指令在多线程类别指令中，可以保护被执行的指令顺利执行，不受中断影响。当发送红外信号时，红外发射的引导码有13.5ms，超过了2ms的系统定时器切换时间，为了保护数据的顺利发送，务必使用这条指令。

红外发送的地址和命令由于是8位整数，所以取值范围是0-255。

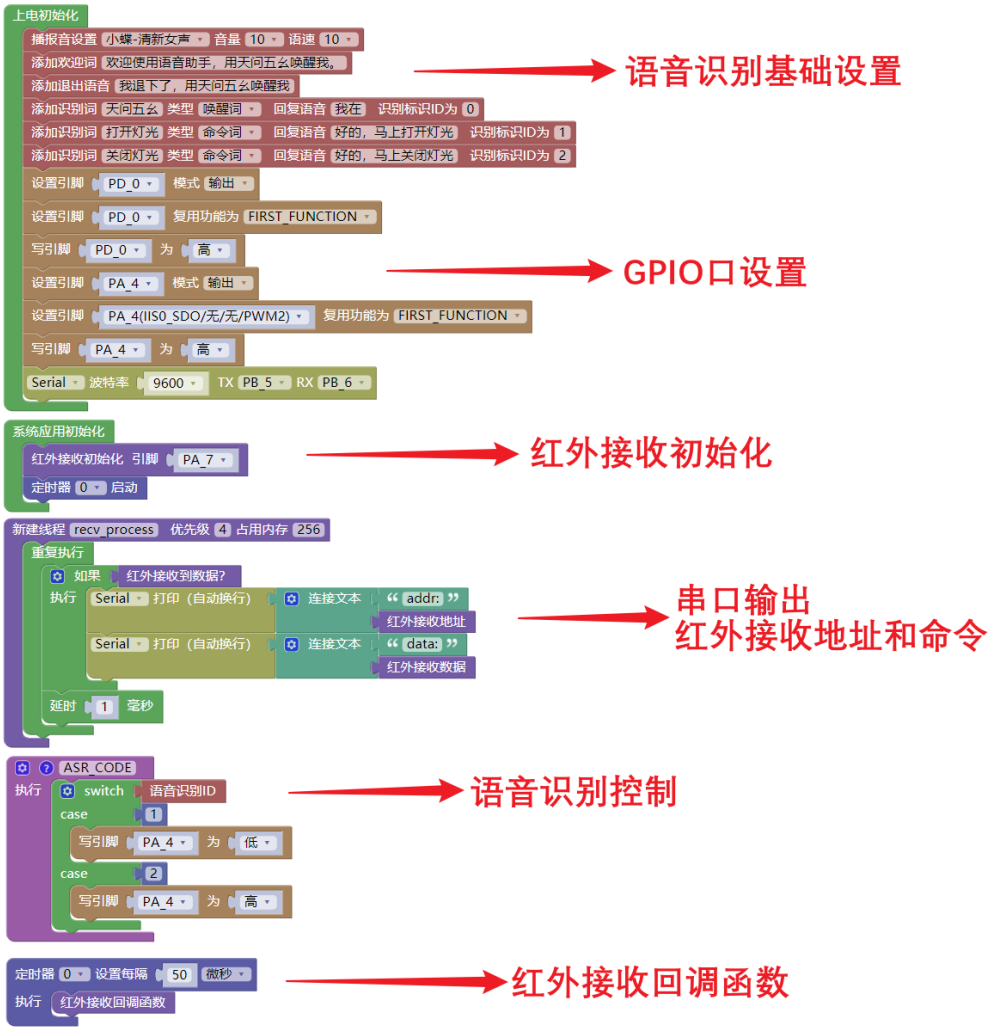
当然我们既可以发送十进制数，也可以发送16进制数。例如下方两种写法，红外接收的结果是一致的，都是地址address=101，命令command=1。

范例3.4 红外接收NEC码

一、范例功能

本范例通过红外接收模块，实现串口输出红外接收NEC码的功能，达成学习红外接收模块的目的。

二、范例分析

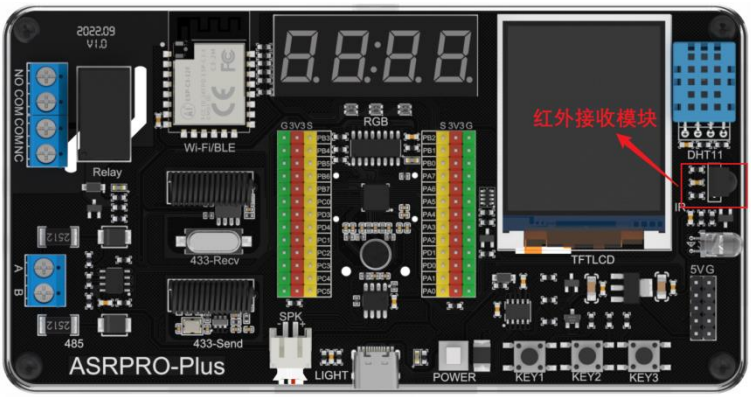


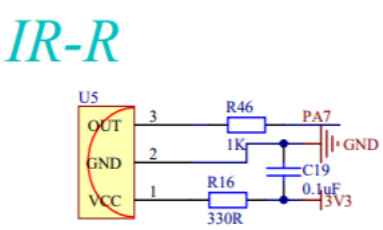
三、范例详解

本范例介绍如何编写红外接收模块的程序，并用串口0输出红外接收的NEC码。

ASRPRO-Plus的右侧部分就外接了一个红外接收模块。下方是红外接收模块的实物图和电路原理图。

ASRRPO-Plus还配套了一个红外发射遥控器，作配套的开发测试使用。





我们之前已经添加了IRSendRev扩展库。在指令区中找到红外接收对应的指令。

1.红外发送初始化

这条指令可以设置红外接收模块的引脚，Plus板默认接在PA_7引脚。查看代码可以发现，这条指令已经设置了启用GPIO口功能，所以不用添加其他指令。

```
// 描述: 红外接收初始化
// 参数: pin:引脚.
// 返回: none.
//=============================================================
void IRSendRev::begin(uint8_t pin)
{
    irparams.recvpin = pin;
    irparams.rcvstate = STATE_IDLE;
    irparams.rawlen = 0;
    pinMode(pin, input);
#if defined(TW_ASR_PRO)
    dpmu_set_io_pull((PinPad_Name)pinToFun[pin],DPMU_IO_PULL_UP);
    set_pin_to_gpio_mode(pin);
#else
    if((pin<=8) && (pin >=5))
    {
        setPinFun(pin,SECOND_FUNCTION);
    }
    else
    {
        setPinFun(pin,FIRST_FUNCTION);
    }
#endif //
```

2.是否接收到数据

这条指令对是否接收到了红外数据进行判断，返回0表示没有接收到数据。

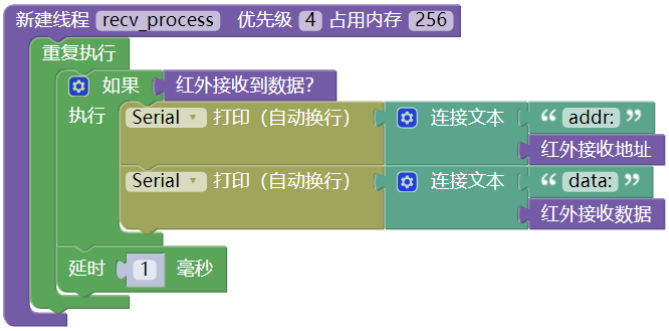
3.红外接收地址和数据

这两条指令可以接收红外发射的地址address和命令command。

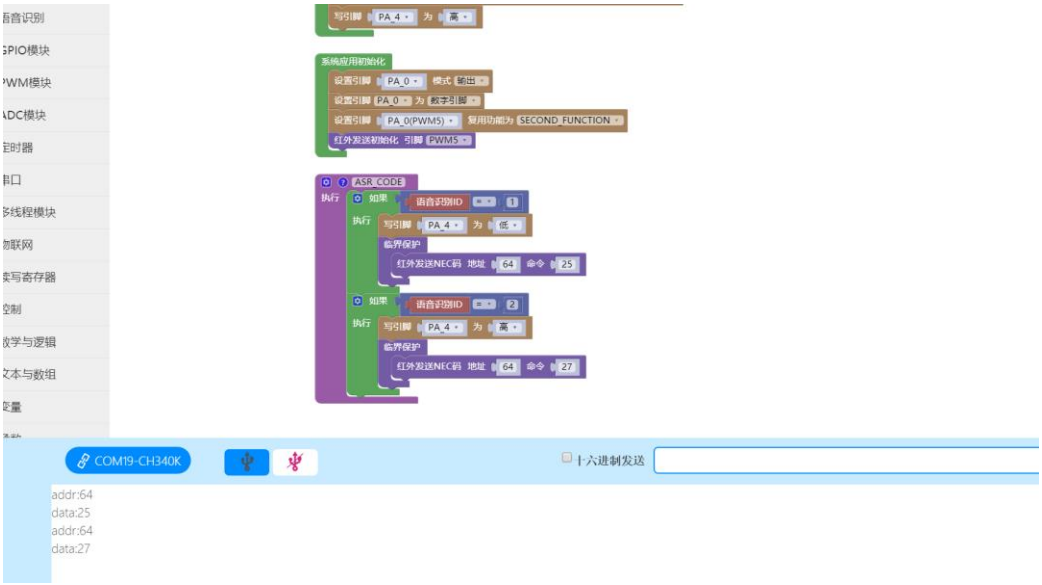
4.红外接收回调函数

这条指令指系统当符合你设定的条件后，对红外接收中断回调函数进行调用。回调函数相当于一个中断处理函数。在本案例中，我们通过硬件定时器，每隔50us来调度一次这个红外接收回调函数。

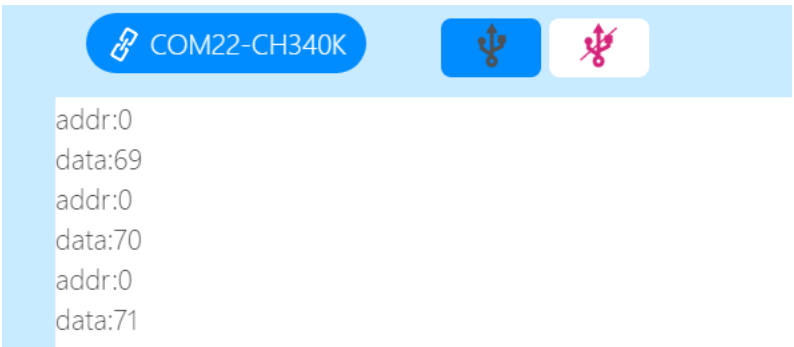
红外接收回调函数会接收外界发送的红外信号。我们在线程recv_process中进行判断，如果判定接收到了一帧完整数据，就用串口输出address和data。



查看下方，当我们语音控制发送NEC码时，串口就会输出地址为64，数据，也就是接收到的命令，分别为25和27。注意不能同时用同一块ASRPRO-Plus作发送和接收使用。



当然我们可以使用配套的遥控器发送红外信号，作红外接收的应用和测试。分别按下1、2、3，串口收到的信息如下显示：

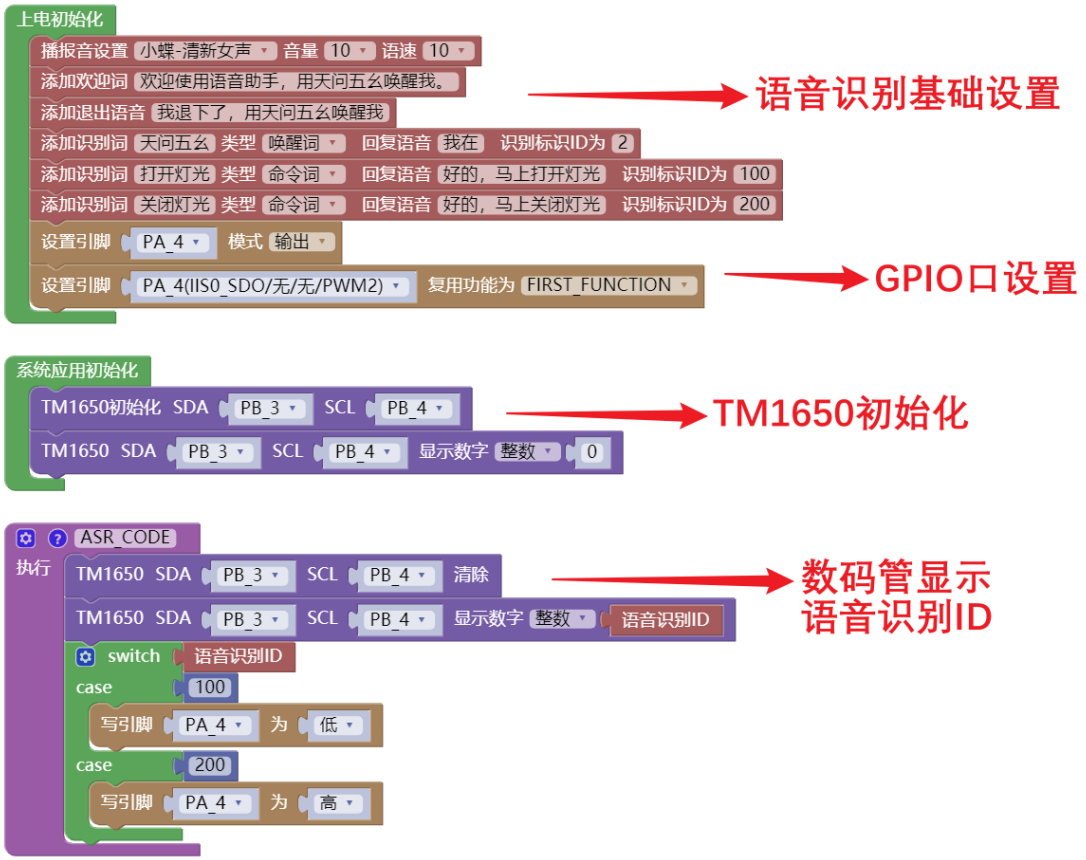


范例3.5 TM1650数码管使用

一、范例功能

本范例通过TM1650数码管显示语音识别ID，达成学习TM1650数码管模块程序编写的目的。

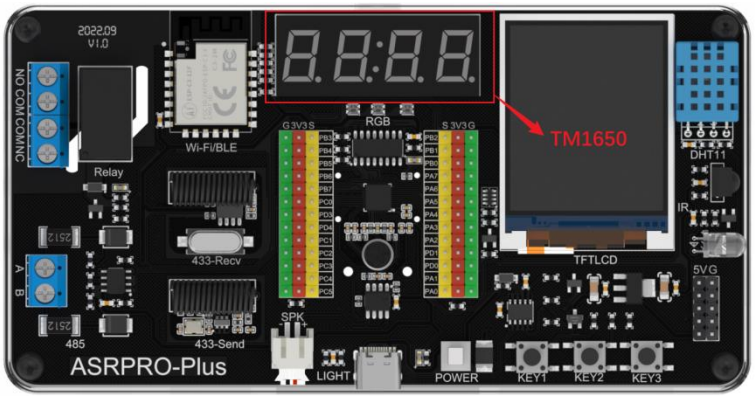
二、范例分析

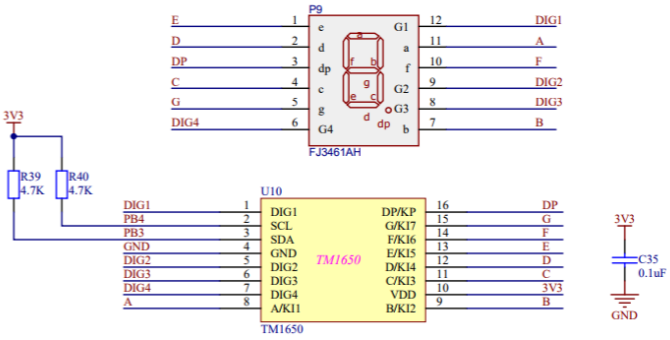


三、范例详解

TM1650是一种带键盘扫描接口的LED(发光二极管显示器)驱动控制专用电路。内部集成有MCU输入输出控制数字接口、数据锁存器、LED 驱动、键盘扫描、辉度调节等电路。TM1650性能稳定、质量可靠、抗干扰能力强，可适用于24小时长期连续工作的应用场合。

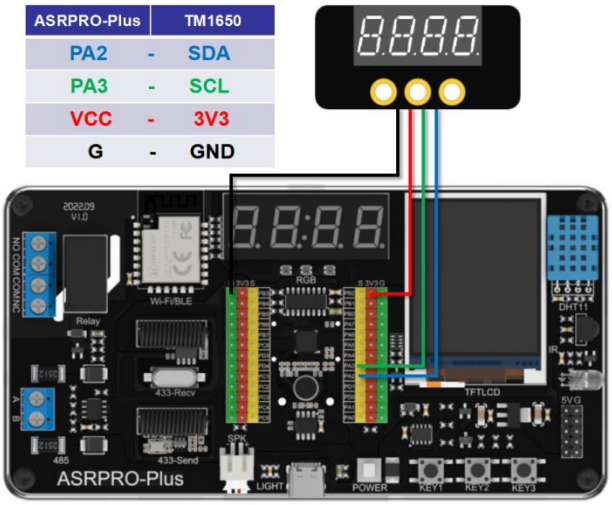
本范例介绍如何编写TM1650数码管模块的程序，使用数码管显示语音识别的ID。ASRPRO-Plus的上方就外接了一个TM1650数码管模块。下方是TM1650数码管模块的实物图和电路原理图。





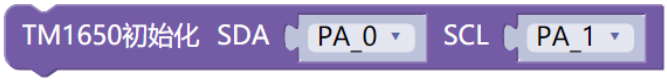
外接TM1650数码管硬件连接可参考下图：

ASRPRO-Plus		TM1650
PA2	-	SDA
PA3	-	SCL
VCC	-	3V3
G	-	GND



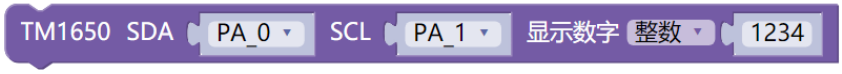
我们在编写TM1650数码管的程序前，需要先添加扩展库。点击添加扩展，找到官方扩展库中的DHTXX，版本选择最新，点击加载。

1.TM1650初始化



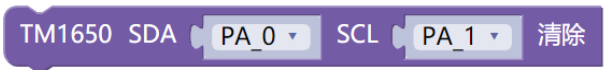
这条指令可以设置TM1650模块的初始引脚。ASRPRO-Plus外接的数码管的引脚为PB_3 - SDA和PB_4 - SCL。TM1650使用的是类IIC接口，只是不带从机地址机制，其中SCL指的是串行通信时钟线，SDA指的是串行通信数据线。

2.数码管显示数字



这条指令可以让数码管显示整数或者小数，显示的数字在-999到9999之间，小数点会根据具体的数值在下方的四个位置显示。

3.数码管清除

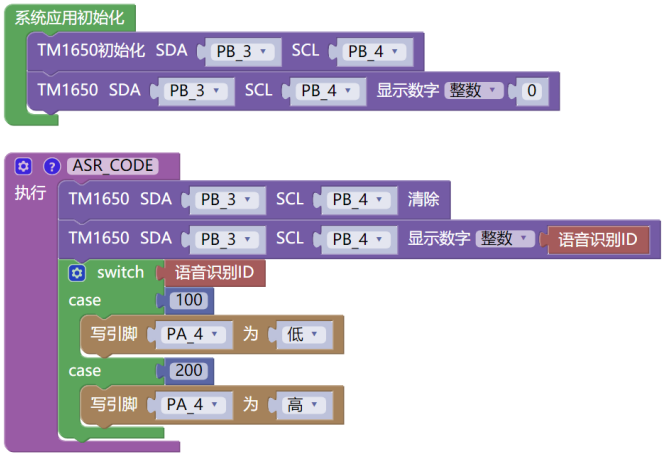


这条指令可以让数码管的内容清除。

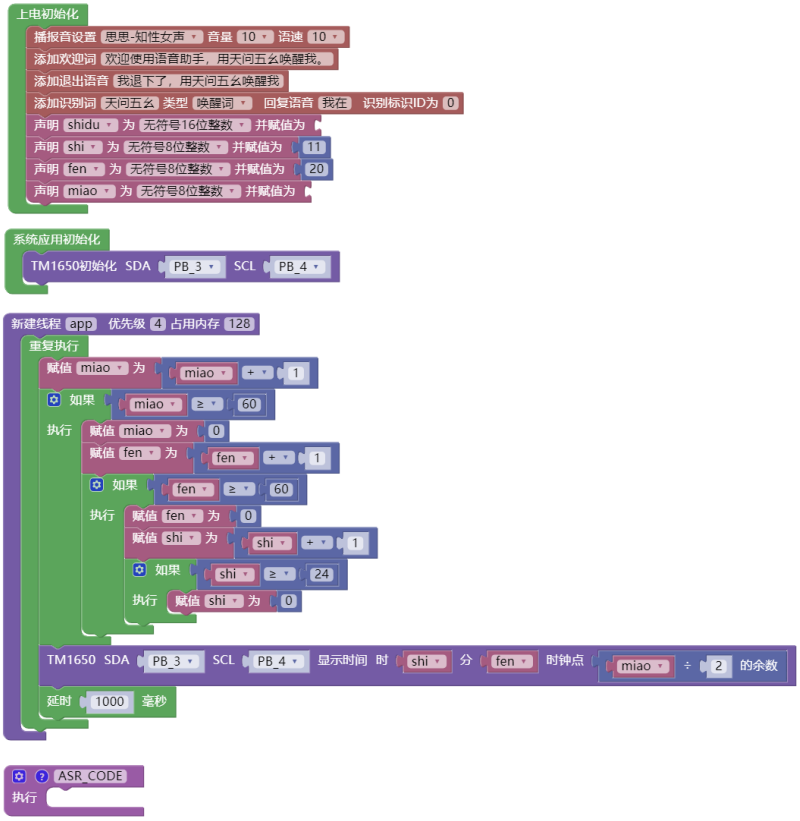
4.数码管显示时间

这条指令可以让数码管以时分秒的形式显示出来，时钟点指的是数码管中间的两个点。我们可以通过指令控制这两个点的亮灭。

范例程序较为简单，数码管在上电后显示数字0。接着显示语音识别ID。当听到打开灯光时显示数字100，听到关闭灯光时显示数字200。



这里提供一个数码管显示时间的范例，设定从11：20开始计时。数码管会实时显示时间，中间的时钟点会根据秒数/2的余数，轮流输入0和1来点亮和熄灭，达成闪烁的效果。

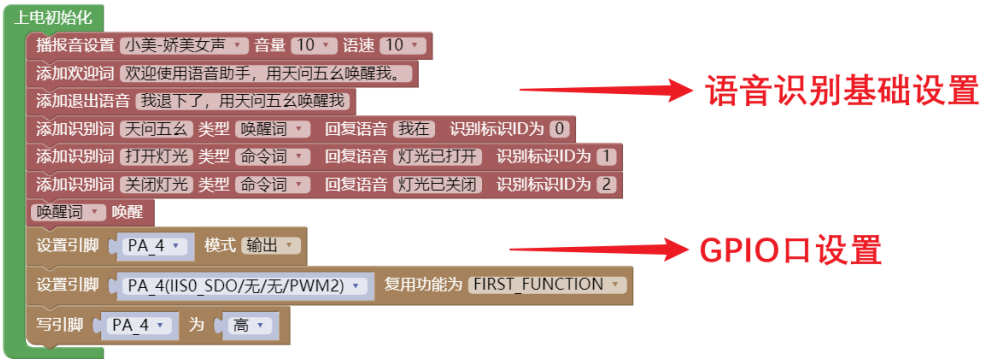

上电初始化
播报音设置 思思-知性女声 音量 10 语速 10
添加欢迎词 欢迎使用语音助手，用天问五幺唤醒我。
添加退出语音 我退下了，用天问五幺唤醒我
添加识别词 天问五幺 类型 唤醒词 回复语音 我在 识别标识ID为 0
声明 shidu 为 无符号16位整数 并赋值为
声明 shi 为 无符号8位整数 并赋值为 11
声明 fen 为 无符号8位整数 并赋值为 20
声明 miao 为 无符号8位整数 并赋值为
系统应用初始化
TM1650初始化 SDA PB_3 SCL PB_4
新建线程 app 优先级 4 占用内存 128
重复执行
赋值 miao 为 miao + 1
如果 miao ≥ 60
执行 赋值 miao 为 0
赋值 fen 为 fen + 1
如果 fen ≥ 60
执行 赋值 fen 为 0
赋值 shi 为 shi + 1
如果 shi ≥ 24
执行 赋值 shi 为 0
TM1650 SDA PB_3 SCL PB_4 显示时间 时 shi 分 fen 时钟点 miao ÷ 2 的余数
延时 1000 毫秒
ASR_CODE
执行

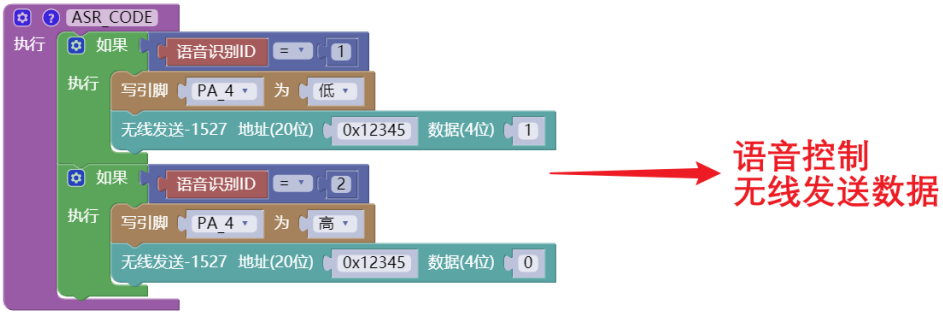
范例3.7 无线发送（433M）

一、范例功能

本范例通过语音控制无线发送数据，支持1527、2262、杜亚电机等编码，驱动无线模块发送无线信号，达成学习433M无线发送程序编写的目的。

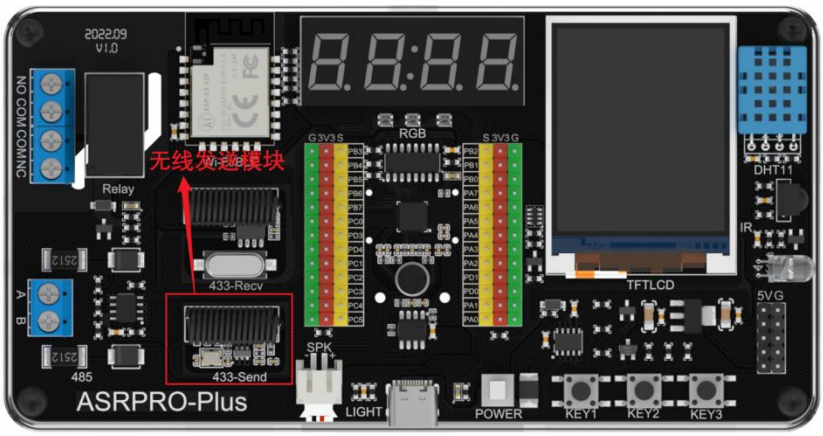
二、范例分析



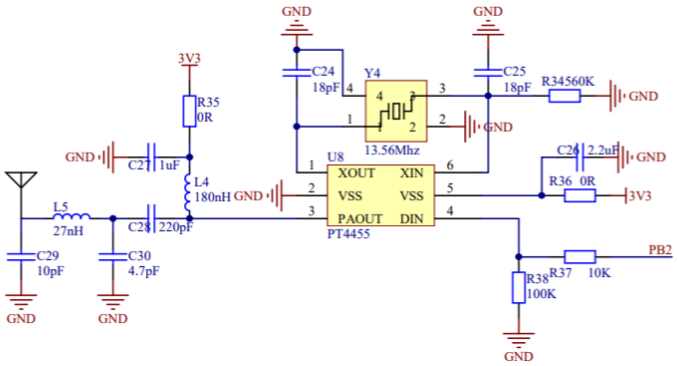


三、范例详解

本范例介绍如何编写无线发送模块的程序，以1527协议为例，通过语音识别控制无线发送数据。ASRPRO-Plus的左下部分就外接了433-Send无线发送模块。下方是无线发送模块的实物图和电路原理图。



433-S



433M/315M的无线在现实生活中使用很广泛，尤其是一些小家电里的无线遥控，比如遥控车库门、遥控晾衣架、无线开关、无线窗帘电机等。

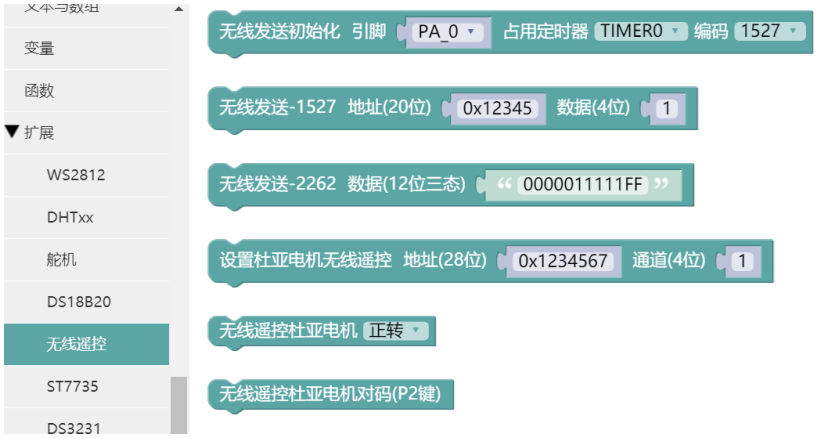
无线遥控器常用的编码方式有两种类型，即固定码与滚动码两种，滚动码是固定码的升级换代产品，目前凡有保密性要求的场合，都使用滚动编码方式。而固定码目前常用的编码格式有2262、1527等，还有一些私有协议，比如宁波杜亚电机协议。这些编码的键值没有加密，传送的都是明文，保密性不高，通常用于遥控开关、家电等。

查看上方433-Send的电路原理图，我们发现内部晶振是13.56Mhz，芯片用的是PT4455。晶振使得工作频率能够稳定在250Mhz-450Mhz。外接的433-Send无线发送模块参数如下。

工作频率	315 (E43) /433 (E40) MHz	谐波抑制	>40dBc
功　　率	15dBm(3V) 16dBm (5V) 18dBm(12V)	工作电流	18-20mA (9V 供电, 40%调制占空比)
调制方式	ASK/OOK	工作电压	2-12V

我们在编写无线发送模块的程序前，需要先添加扩展库。点击添加扩展，找到官方扩展库中的无线遥控，版本选择最新，点击加载。

加载后我们就可以在扩展类别中，找到无线发送对应的指令。



1.无线发送初始化

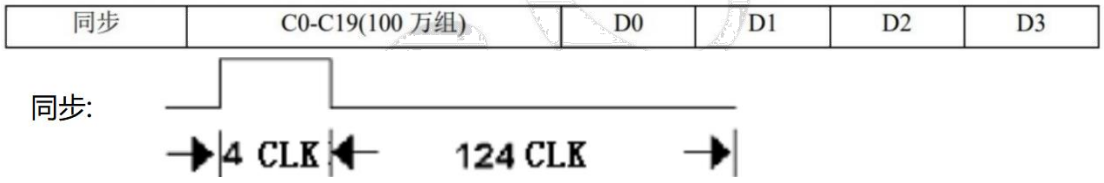
这条指令可以设置无线发送模块的引脚、占用哪个硬件定时器以及编码。ASRPRO-Plus外接的433-Send接PB_2引脚。

系统会调用硬件定时器中断去进行发送，我们在这里选择TIMER0。注意不要和其他库同时调用一个定时器，会产生冲突。

接下来我们来详细介绍一下1527、2262、杜亚电机三种编码/协议。

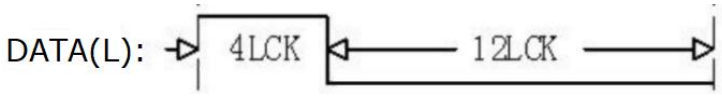
1527编码：

1527编码包含一个同步码，20位地址码C0-C19，4位开关码D0-D3。

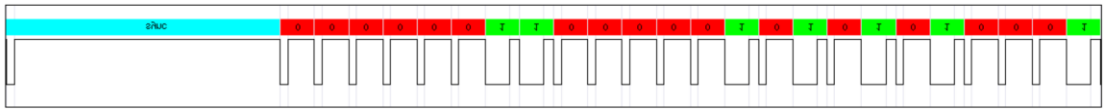


同步头高电平和低电平脉宽比为1:31，其中高电平至少为四个时钟，定义 :1CLK=8OSC CLOCK。

数据1高电平和低电平脉宽比为3:1

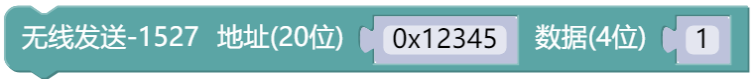


数据0高电平和低电平脉宽比为1:3



4LCK脉宽范围在300-500us之间，常用的有350us，公牛面板为400us。

2.无线发送-1527



这条指令可以通过1527编码发送无线消息，可以设置两个参数。第一个参数是C0-C19的20位地址，一般为出厂统一设置，程序上可以用EEPROM或者FLASH来存储；第二个参

数是D0-D3的数据，包含（0~15）16种开关。例如该指令，地址位0x12345，数据位1，会转化成二进制数据000100100011010001010001，共24位，也就是对应了16进制的123451，其中前五位是地址位，后一位是数据位。注意：有些设备会有特殊的设置，比如官方提供的无线插座，数据位需要大于1。

```
 * @brief 1527协议应用层数据收发
 * 1527编码
 * 同步+地址（C0-C19）+开关（D0 D1 D2 D3）
 * 同步    _|—|____________________|
 * 时间    _|1T|________31T_________|
 * DATA(H)_|————————|__|
 *         _|——3T——|1T|
 * DATA(L)_|—|————————|
 *         _|1T|——3T——|
 * 16LCK=4T
 */
class rf1527:public BlueRFLink
{
  public:
    rf1527(uint8_t tx_pin, timer_base_t timer):BlueRFLink(tx_pin,timer)
    {
      setProtocol(13);//默认使用公牛的协议，这样公牛面板和其他厂家的面板都能兼容，公牛面板只支持这个。
    };
    void send(uint32_t address, uint8_t data);
};

void rf1527::send(uint32_t address, uint8_t data)
{
  sendRaw(((address<<4) | (data&0x0f)), 24);
}
```

无线信号容易受到扰乱，尤其第一帧，所以数据至少需要连续发送3次以上。扩展库里默认为重复发送10次。如需修改，可以查看BlueRFLink基类。

```
BlueRFLink::BlueRFLink(uint8_t tx_pin, timer_base_t timer)
{
    rfparams.tx_pin = tx_pin;
    rfparams.len = 0;
    this->setRepeatTransmit(10);//默认重复发送10次
    this->setProtocol(1);//默认使用协议1
    rfparams.timer = timer;
    rfparams.invertedSignal = protocol.invertedSignal;

    if(rfparams.timer == TIMER0)
    {
      _timerx_irq = TIMER0_IRQn;
    }
    else if(rfparams.timer == TIMER1)
    {
      _timerx_irq = TIMER1_IRQn;
    }
    else if(rfparams.timer == TIMER2)
    {
      _timerx_irq = TIMER2_IRQn;
    }
    else if(rfparams.timer == TIMER3)
    {
      _timerx_irq = TIMER3_IRQn;
    }
}
```

2262编码：

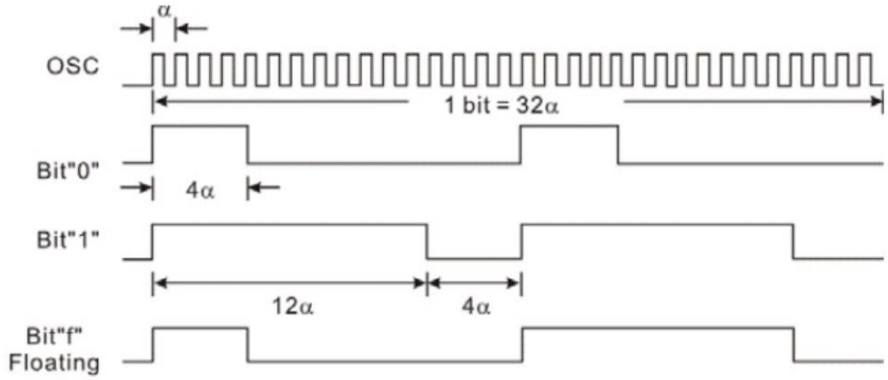
2262编码包含一个SYNC位（同步码）和12个AD位（地址/数据）。同步码是时间较长的低电平间隔，高低电平比为1:31，地址码和数据码则用宽度不同的脉冲来表示。两个窄脉

冲表示“0”，两个宽脉冲表示“1”，一个窄脉冲一个宽脉冲表示“F”，也就是地址码的“悬空”。实际脉冲高低电平比例如下：

数据0: 1 Pulse High + 3 Pulses Low + 1 Pulse High + 3 Pulses Low

数据1: 3 Pulses High + 1 Pulse Low + 3 Pulses High + 1 Pulse Low

数据F: 1 Pulse High + 3 Pulses Low + 3 Pulses High + 1 Pulse Low



3.无线发送-2262

这条指令通过2262编码发送无线消息。其中12位AD位中，包含了6个数据地址共用位，最大12位地址。

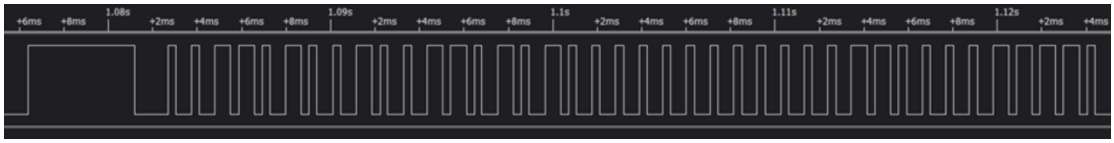
A0	A1	A2	A3	A4	A5	A6/D5	A7/D4	A8/D3	A9/D2	A10/D1	A11/D0

例如该指令，输入的数据是0000011111FF，假设我在接收应用处理时，设置数据位是6位，数据位是4位，那么其中000001是地址位，1111是数据位，FF是悬空。在数据发送时，和1527一样转换为24位长度发送。

```
 * @brief 2262协议应用层数据收发
 * 脉冲宽度300-500us,默认350
 * 同步    _|—|_____________________|
 * 时间    _|1T|________31T_________|
 * DATA(0)_|—|________|—|________| 数据0
 * DATA(1)_|————————|__|————————|__| 数据1
 * DATA(F)_|—|________|————————|__| 浮空
 * A0 A1 A2 A3 A4 A5 A6/D5 A7/D4 A8/D3 A9/D2 A10/D1 A11/D0
 * 最大12位地址，包含了6位数据地址共用位。
 * 底层数据发送还是和1527协议一样转换为24位长度发送
 * 不同厂商对地址的分配不同，需要根据实际情况做应用层数据解析
 */
class rf2262:public BlueRFLink
{
  public:
    rf2262(uint8_t tx_pin, timer_base_t timer):BlueRFLink(tx_pin,timer){setProtocol(1);};
    void sendTriState(const char* sCodeWord);
};
```

杜亚电机协议：

宁波杜亚是全球最大的一家门窗电机制造商，目前市面上的智能窗帘电机大部分都是出自他家；如果不是他家的，里面的通讯协议也会兼容他的，所以杜亚电机私有协议，已经默认为行业协议。



输出编码包括了同步码、28位的地址码（ID）、4位的通道（CH）和8位的命令（CMD）

其中同步码高低电平脉宽比为11.95:3.9，数据1高低电平比脉宽比为1.75:1，数据0高低电平比脉宽比为1:1.75。

CH=0代表群控，通道最大为15。

CMD=0x11代表电机顺时针转；CMD=0x55代表停；CMD=0x33代表电机逆时针转；

CMD=0xCC代表对码；CMD=0x1E代表电机顺时针点动；CMD=0x3C代表逆时针点动；

```
 * @brief DOOYA电机协议应用层数据收发
 * 1527编码
 * 同步+28 Bit ID + 4 Bit CH + 8 Bit CMD
 * 同步    _|————————————|_______|
 * 时间    _|———11.95T———|__3.9T__|
 * DATA(H)_|————————|__|
 *         _|—1.75T—|1T|
 * DATA(L)_|—|————————|
 *         _|1T|—1.75T—|
 * CH=0 群控，最大15
 * CMD=0x11 开；CMD=0x55 停；CCMD=0x33 关；MD=0xCC 对码；CMD=0x1E 固定点动开；CMD=0x3C 固定点动关；
 */
class dooya:public BlueRFLink
{
  public:
    dooya(uint8_t tx_pin, timer_base_t timer):BlueRFLink(tx_pin,timer)
    {
      setProtocol(14);
      setChannel(1);
      setAddress(0xfffffff);
    };
    void motorCW();
    void motorCCW();
    void motorStop();
    void motorInchingCW();
    void motorInchingCCW();
    void matchCode();
    void setAddress(uint32_t addr);
    void setChannel(uint8_t ch);
    void sendCmd(uint8_t cmd);
  private:
    uint32_t _address;
    uint8_t _ch;
};
```

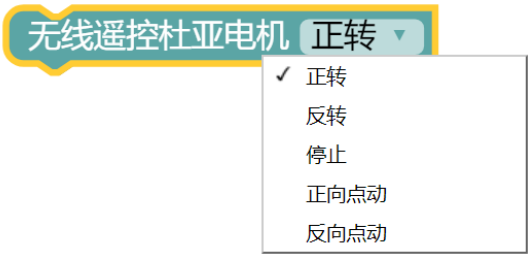
3.杜亚电机-地址通道设置

这条指令可以根据杜亚电机协议设置地址和通道，其中地址为28位，通道位是4位，一般为出厂统一设置，程序上可以用EEPROM或者FLASH来存储。查看上方代码，默认address是0xfffffff，默认ch是1。

4.杜亚电机-地址通道设置

当与电机进行配对时，需要先发送这条指令，对应的cmd=0xCC。

5.杜亚电机-电机正反转



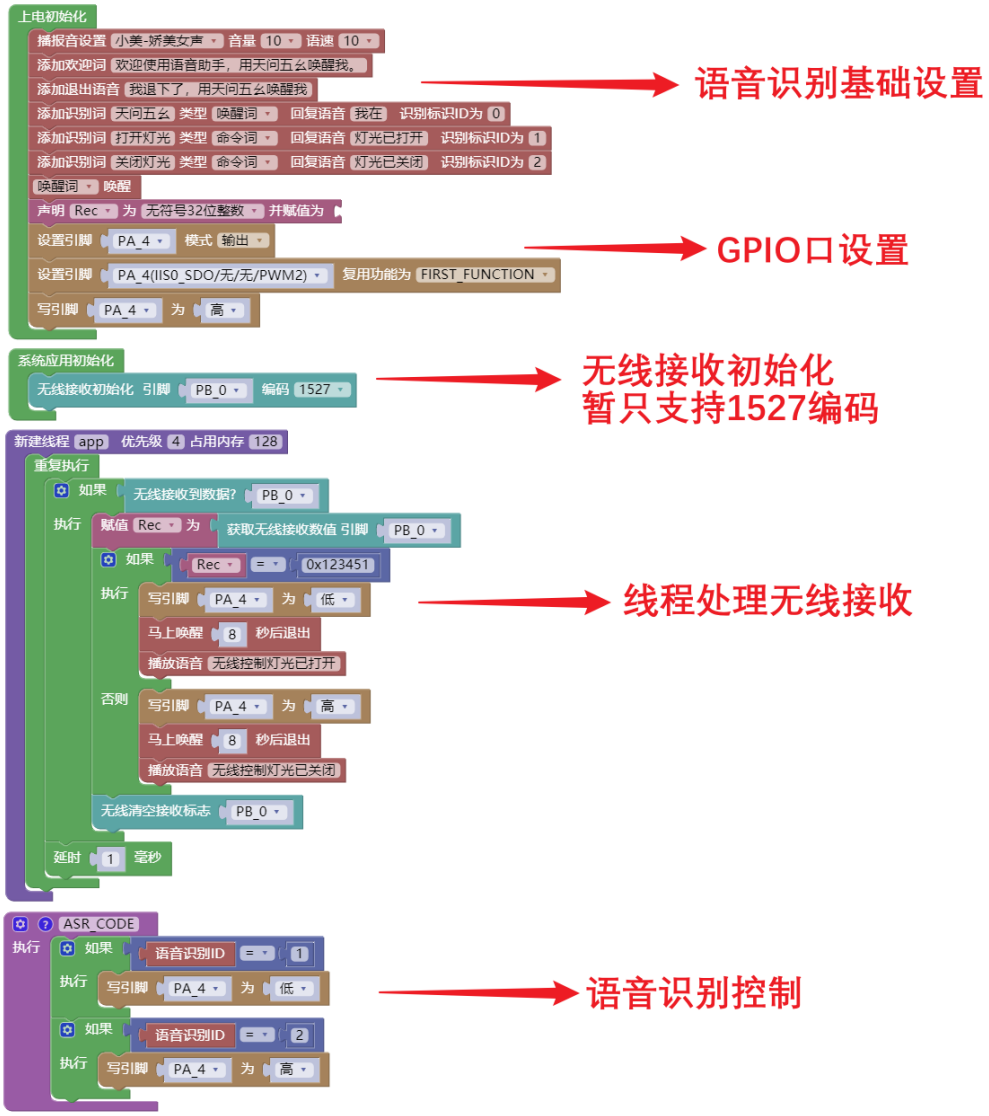
这条指令可以发送无线信号控制杜亚电机正转、反转、停止、正向点动和反向点动，发送的cmd参考上方代码。

范例3.8 无线接收（433M）

一、范例功能

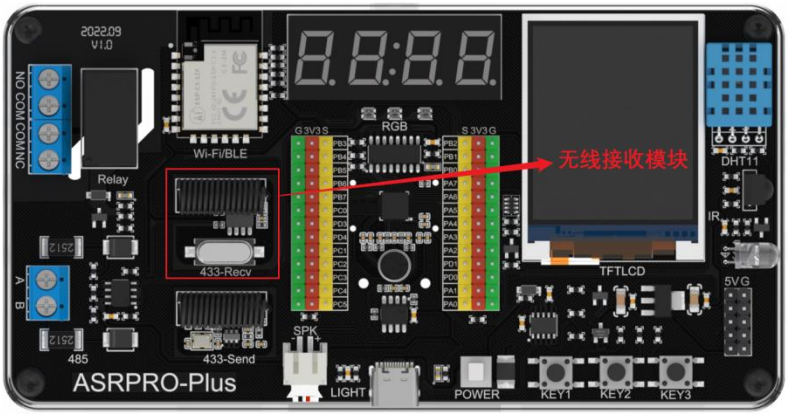
本范例通过多线程的方式处理无线接收数据，实现通过1527编码，433-Recv接收无线消息控制板载灯光亮灭的功能，达成学习无线接收模块程序编写的目的。无线接收引脚需要GPIO引脚中断功能，只能是PA0-PA7、PB0-PB7中的一个。

二、范例分析

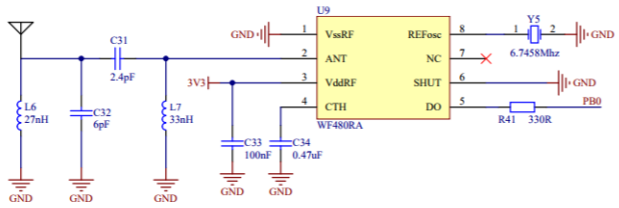


三、范例详解

本范例介绍如何编写无线接收模块的程序。无线接收扩展库目前只支持1527协议。我们通过多线程的方式处理无线接收数据并进行判断，以此控制板载灯光亮灭。ASRPRO-Plus的左下部分就外接了433-Recv无线接收模块。下方是无线接收模块的实物图和电路原理图。



433-R

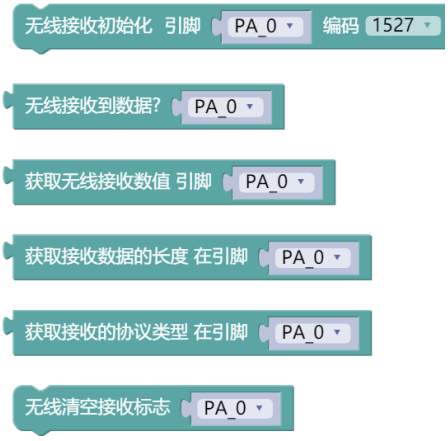


查看上方433-Recv的电路原理图，我们发现内部晶振是6.7458Mhz，芯片用的是WF480RA。晶振使得工作频率能够稳定在300Mhz-440Mhz。外接的433-Recv无线接收模块参数如下。

- 低功耗
  - 5.0mA/3.3V @ 315MHz
  - 5.0mA/3.3V @ 433.92MHz
  - 0.01uA/3.3V @ Shut Down Mode
- 低启动时间：3 ms
- 数据速率：≤ 10kbps
- 宽工作电压：DC 2.0V~ 5.5V

我们在编写无线接收模块的程序前，需要先添加扩展库。点击添加扩展，找到官方扩展库中的无线接收，版本选择最新，点击加载。

加载后我们就可以在扩展类别中，找到无线接收对应的指令。



1.无线接收初始化

这条指令可以设置无线接收的引脚，ASRPRO-Plus的无线接收模块连接在PB_0引脚。

2.无线接收是否收到数据

如果接收到的数据不为0，就判断收到数据，一般配合如果使用

```
bool RCSwitch::available()
{
  return RCSwitch::nReceivedValue != 0;
}
```

3.无线接收数据数值

这条指令可以获得无线接收到的数据，数据类型为32位整数。

```
uint32_t RCSwitch::getReceivedValue()
{
  return RCSwitch::nReceivedValue;
}
```

4.无线接收数据长度

这条指令可以获得无线接收到数据的长度。

```
unsigned int RCSwitch::getReceivedBitlength()
{
  return RCSwitch::nReceivedBitlength;
}
```

5.无线接收数据协议

这条指令可以获得无线接收到数据的协议。

```
unsigned int RCSwitch::getReceivedProtocol()
{
  return RCSwitch::nReceivedProtocol;
}
```

对指令3-5有疑问的，可以查看范例3.8无线发送1527编码部分。

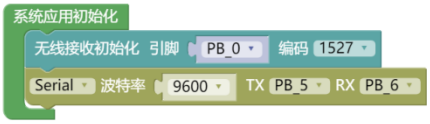
6.无线接收标志清空

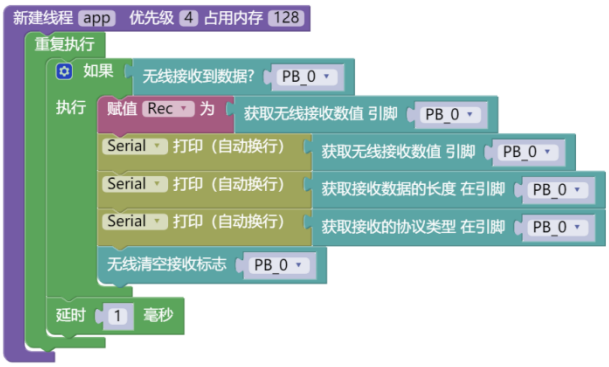
每次接收无线数据后，都要使用这条指令清空无线接收标志，保证下次接收顺利进行。

学习这些指令后，我们使用串口来查看一下数值、协议类型、数据长度。

无线发送指令可以直接使用范例代码3.8，通过语音控制，根据1527发送消息。注意不能在同一块ASRPRO-Plus上进行发送和接收。

无线接收指令可以对范例代码3.9进行简单修改，通过串口0输出信息。



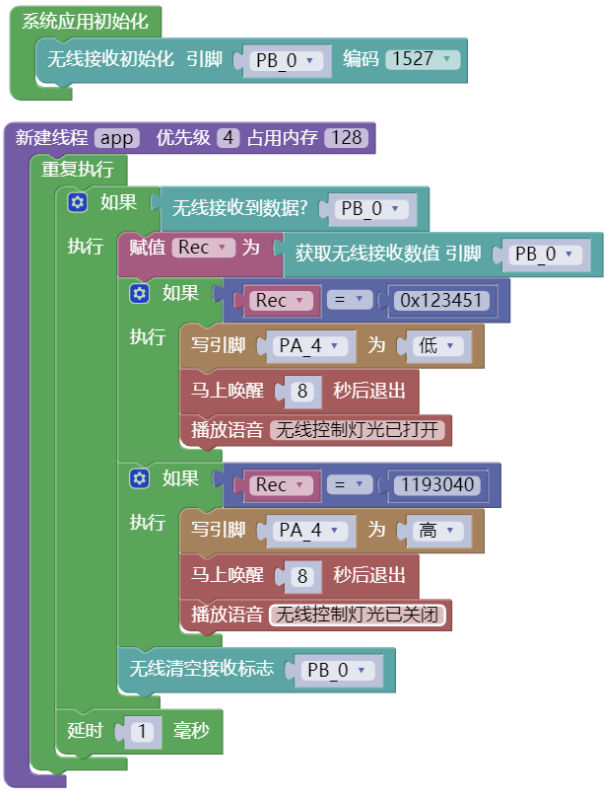


说“打开灯光”，发送无线信号，然后查看串口信息。

其中1193041是十进制数，转化为16进制数是123451，其中12345是地址码，1是数据。

数据长度为24，也就是24位，即20位地址和4位数据，协议类型为1。

对于本范例来说，我们可以用以下结构对数据进行判断。判断的数值既可以使用十六进制数，也可以使用10进制数。



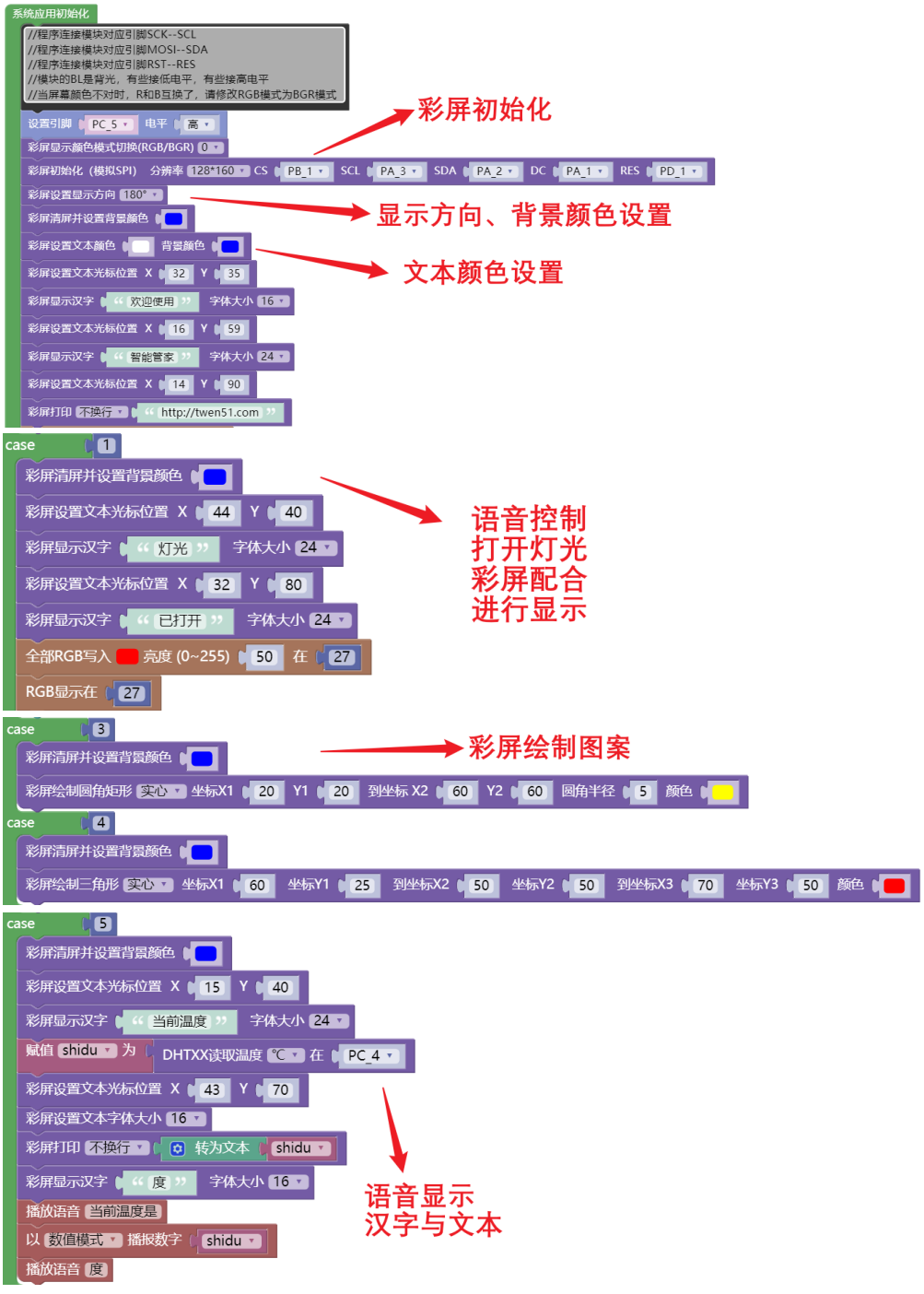
范例3.9 彩屏（ST7735）综合范例

一、范例功能

本范例通过语音控制彩屏，实现彩屏显示文本、汉字、图案，同时可以设置背景颜色和文字颜色的功能，达成学习ST7735彩屏程序编写的目的。

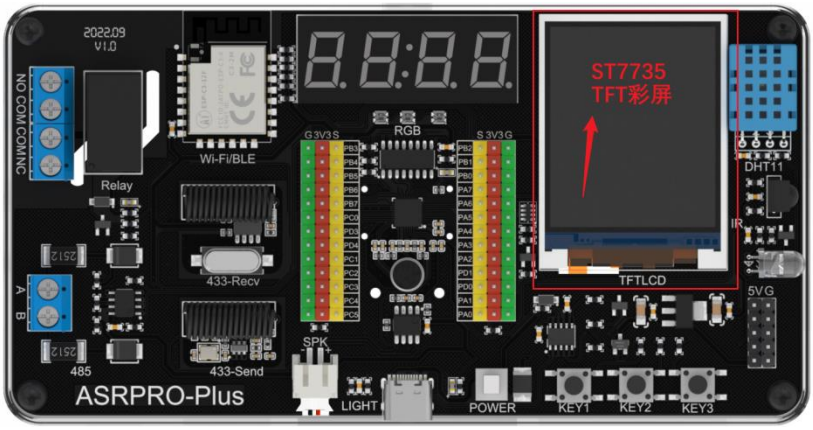
二、范例分析

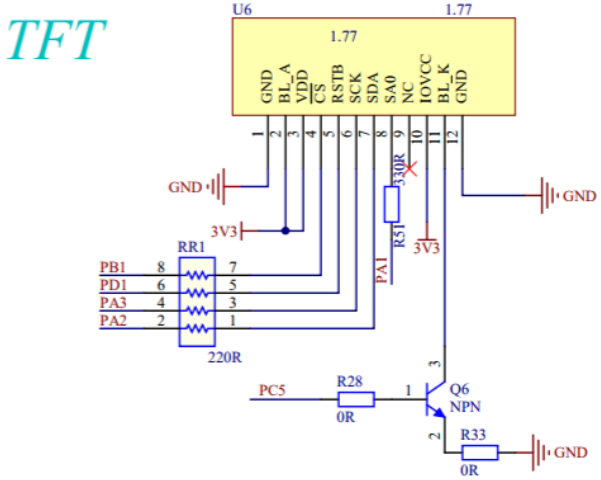
此范例包含了彩屏、数码管、WS2812、无线发送、温湿度传感器等多个部分，是一个综合案例。这里只对彩屏部分作分析。



三、范例详解

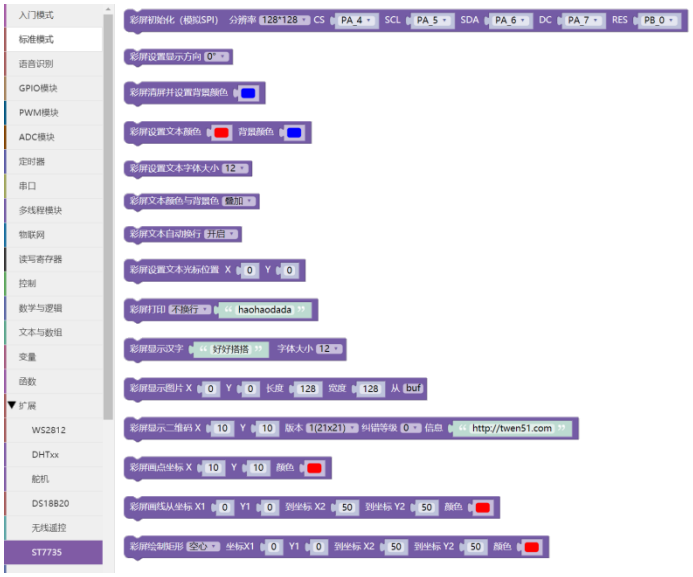
本范例介绍如何编写ST7735 TFT彩屏的程序。ASRPRO-Plus的右上部分就外接了一块彩屏。下方是彩屏的实物图和电路原理图。





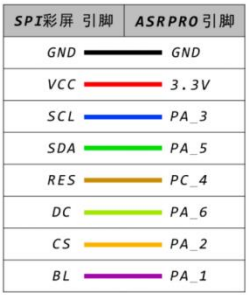
我们在编写彩屏的程序前，需要先添加扩展库。点击添加扩展，找到官方扩展库中的ST7735，版本选择最新，点击加载；注意扩展库版本用最新版，最新版优化最完善。

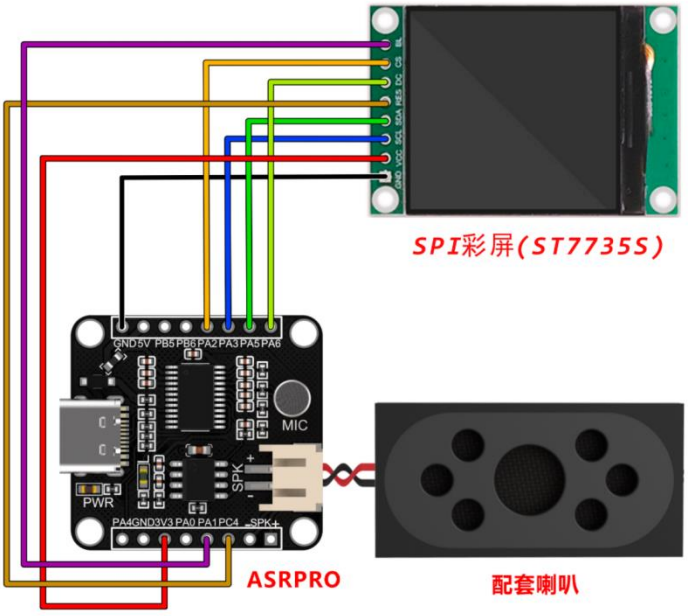
加载后我们就可以在扩展类别中，找到彩屏对应的指令。



ST7735芯片驱动的SPI彩屏，分辨率可以自由设置，最大为128*160。

这里附带一张ASRPRO与彩屏相连的连接图。





而当彩屏与ASRPRO-Plus连接时，引脚连接说明如下：

ASRPRO-Plus引脚	彩屏引脚	接口定义
GND	GND	接地端
3.3V	VCC	3.3V
PA_4	CS	LCD片选信号
PA_5	SCL	串行SPI时钟信号
PA_6	SDA	串行SPI数据输入端
PA_7	DC	命令/数据选择端
PD_1	RES	LCD复位信号
PC_5	BL	彩屏背光开关，BL=1背光亮，BL=0背光灭

1.打开彩屏背光

请一定记得打开彩屏背光灯，不然显示黑屏。

2.彩屏初始化

这条指令可以初始化彩屏，放在系统上电初始化中，直接从指令区拖出，对应硬件修改程序相应引脚。设置彩屏的分辨率为128*160。

3.彩屏方向、颜色设置

第一条指令可以设置彩屏方向0度、90度、180度、270度，一般设置180度。

第二条指令可以设置颜色模式，一般切换RGB模式。ST7735彩屏采用16位数据显示彩色像素，每个像素数据是R(5)G(6)B(5)，个别彩屏内部会采用B(5)G(6)R(5)。

4.彩屏清屏

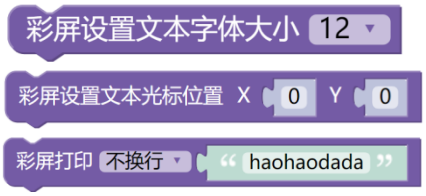
这条指令可以让彩屏清屏并设置背景颜色，主要作用是清除屏幕。

5.彩屏文本颜色设置

这条指令可以设置文本颜色和背景颜色。这条指令可以和指 

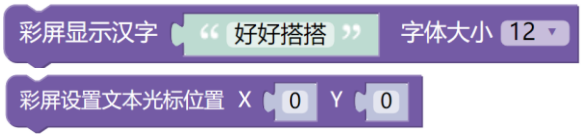
配合使用，可以设置两者颜色叠加或不叠加。

6.彩屏显示文本



彩屏显示文本一般使用上面三条指令。这三条指令可以设置文本的字体大小、文本的位置和文本内容。其中彩屏的分辨率以128*160为例，x的范围是0-127，y的范围是0-159；彩屏打印的内容，常规的ASCII码均可以填入，例如数字、英文、常用符号。

7.彩屏显示中文



这条指令可以设置显示中文的内容和字体大小，一般也配合彩屏设置文本光标位置使用。汉字的字体大小可以设置为12，16，24，32。假如设置为12，那么一个汉字大小就是12*12。

8.彩屏显示点、线、图形

这些指令可以绘制点、线和各种图形。其中画线用的是两点连成线段；绘制矩形的方法是通过定义矩形左上角和右下角的坐标来确定矩形；绘制圆形需要填入的参数是圆心坐标和半径；绘制三角形需要填入三个点的坐标；绘制圆角矩形则是在矩形的基础上添加圆角半径参数；绘制图形均可以设置是空心还是实心，也就是内部是否充满。

9.彩屏显示图片

这条指令可以显示图片，我们需要将图像取模数据存放到数组中，通过这条指令去读取。

这里提供一个显示图片的范例、最终结果以及取模软件地址。

http://www.haohaodada.com/new/learning_show.php?id=413

这里以下方图片为例，详细说明如何使用取模软件并进行程序编写。

首先先对图片大小进行修改。如果超过128*160的，先修改大小。

打开取模软件，并修改最大宽度和高度为128和160，设置16位真彩，其他为默认。

点击左上角的打开，选择图片，打开图片后如下所示：

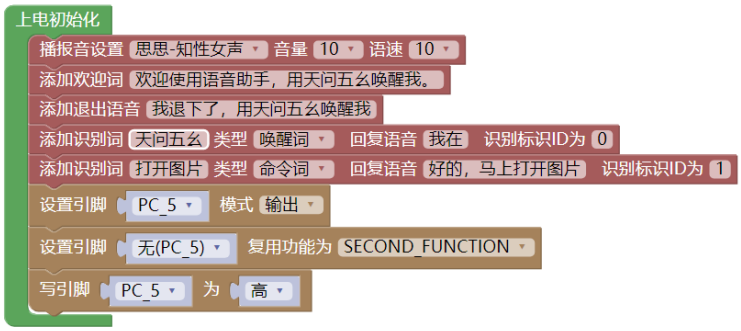
点击左上角的保存，此时会将图片的数据保存为一个.C的文件，用记事本打开，将里面的内容全部复制。

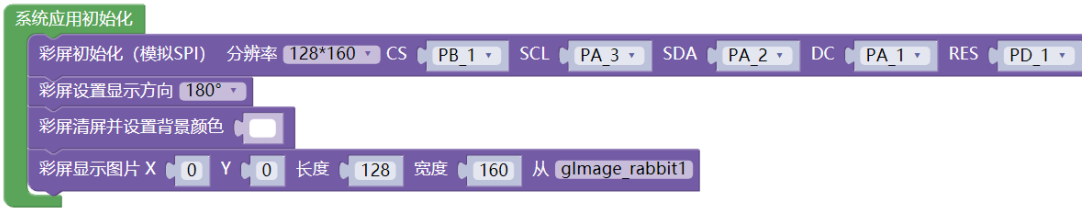
查看该文件，我们之后要将这些数据存放到数组中，数组的名称就用开头的gImage_rabbit1。

```
const unsigned char gImage_rabbit1[40968] = { 0X00,0X10,0X80,0X
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XF
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XF
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XF
```

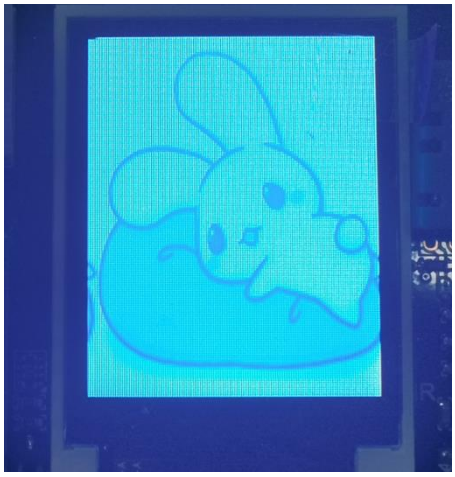
找到读写寄存器的代码框，将这些复制并存放，然后将代码框折叠起来。

具体程序代码如下：





实际展示效果：



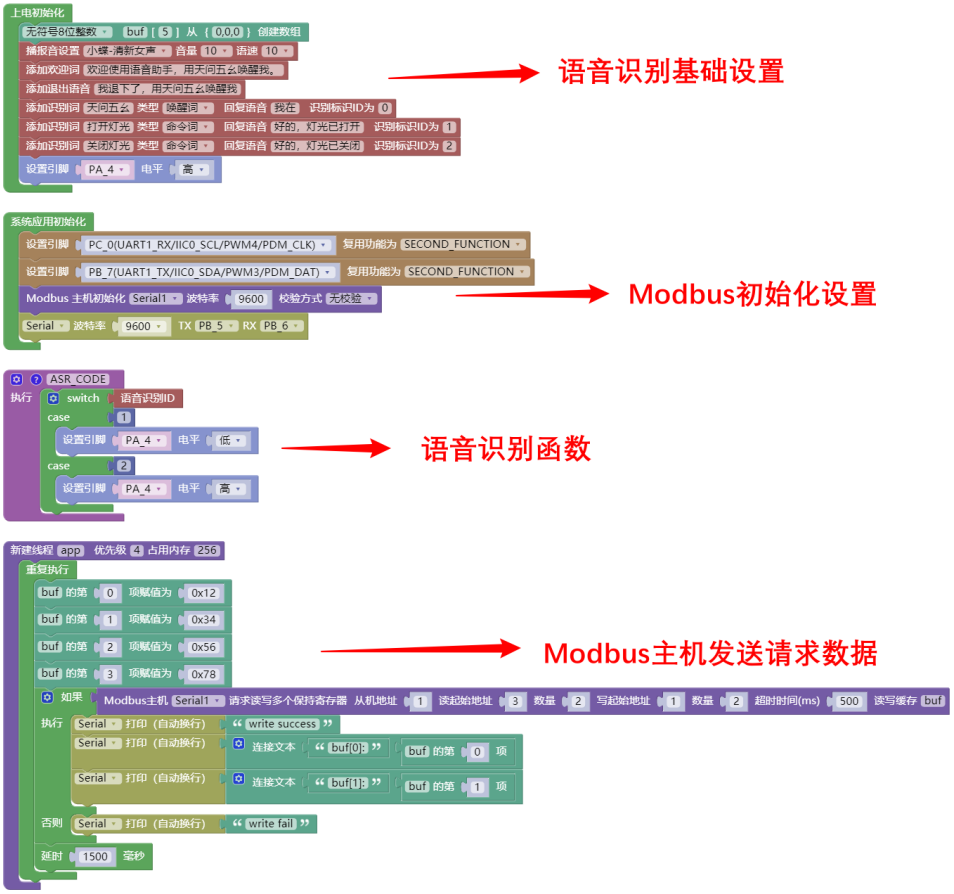
注意这里测试图片是128*160，程序中显示图片大小应该根据实际情况调整，不然会出现花屏的现象。

范例3.10 MODBUS主机案例

一、范例功能

本范例通过学习Modbus协议，实现主机读写数据的功能，达成学习MODBUS主机程序编写的目的。

二、范例分析



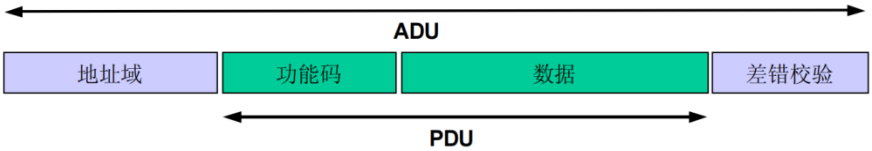
三、范例详解

Modbus是一种串行通信协议，是Modicon公司（现在的施耐德电气Schneider Electric）于1979年为使用可编程逻辑控制器（PLC）通信而发表。它在连接到不同类型总线或网络的设备之间提供客户机/服务器通信。目前Modbus已经成为工业领域通信协议的业界标准（De facto），并且是工业电子设备之间常用的连接方式。

这里Modbus主要指Modbus-RTU协议，下面的说明均以Modbus-RTU协议进行说明。

Modbus协议定义了一个与基础通信层无关的简单协议数据单元（PDU）。特定总线或网络上的Modbus协议映射能够在应用数据单元（ADU）上引入一些附加域。

这是一个通用的Modbus帧，包含了地址码、功能码、数据码和校验码。



地址码：就是指从机的地址；范围是0-255，0是广播，一般默认为1，具体按实际情况。

功能码：指明主机要执行的动作，有效范围是1-255（128-255为异常响应保留）。功能码包括公共功能码和用户定义功能码。

公共功能码：

				功能码		
				码	子码	（十六进制）
数据访问	比特访问	物理离散量输入	读输入离散量	02		02
		内部比特或物理线圈	读线圈	01		01
			写单个线圈	05		05
			写多个线圈	15		0F
	16 比特访问	输入存储器	读输入寄存器	04		04
		内部存储器或物理输出存储器	读多个寄存器	03		03
			写单个寄存器	06		06
			写多个寄存器	16		10
			读/写多个寄存器	23		17
			屏蔽写寄存器	22		16
	文件记录访问		读文件记录	20	6	14
			写文件记录	21	6	15
	封装接口		读设备识别码	43	14	2B

数据：传输的数据内容

校验码：验证收、发的数据是否正确

我们以主机通过Modbus协议读取从机的温湿度值为例，来进行详细介绍。

地址码为0x01；

功能码为0x03，也就是指读多个寄存器；

起始地址和数据长度，我们来看具体情况。

以扁卡轨壳485型温湿度变送器为例，其主机问询帧结构和寄存器地址如下所示。

主机问询帧结构：

地址码	功能码	寄存器起始地址	寄存器长度	校验码低位	校验码高位
1 字节	1 字节	2 字节	2 字节	1 字节	1 字节

4.3 寄存器地址

寄存器地址	PLC或组态地址	内容	操作	说明
0000 H	40001	湿度	只读	湿度实时值（扩大10倍）
0001 H	40002	温度	只读	温度实时值（扩大10倍）
0050H	40081	温度校准值	读写	整数（扩大10倍）
0051H	40082	湿度校准值	读写	整数（扩大10倍）
07D0 H	42001	设备地址	读写	1~254（出厂默认1）
07D1 H	42002	波特率	读写	0代表2400 1代表4800

假设我要同时读取湿度值和温度值，那么我们从寄存器0000H开始读，起始地址用两个字节表示，也就是0x00，0x00；数据长度为2，也就是0002，用0x00，0x02表示。最后是

校验码0BC4。由于温度值放大了10倍，所以最后读取的数值处理时要除以10。

举例：读取设备地址 0x01 的温湿度值

问询帧（16 进制）：

地址码	功能码	起始地址	数据长度	校验码低位	校验码高位
0x01	0x03	0x00 0x00	0x00 0x02	0xC4	0x0B

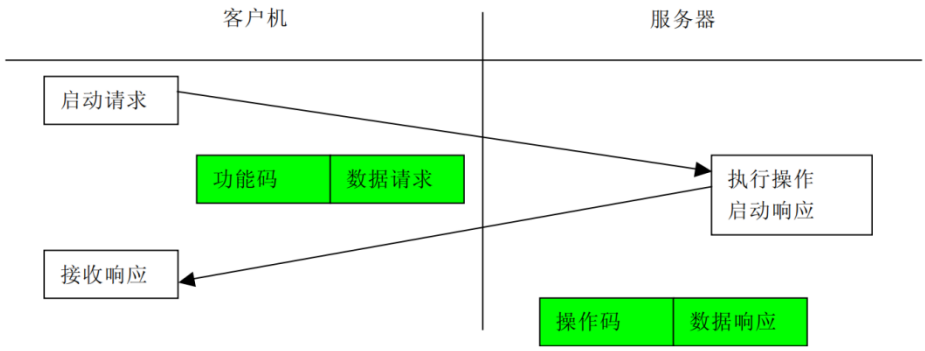
假设我只想读取温度值，查看上方参数，那么一帧的数据是

地址码:0x01　功能码:0x03 起始地址:0x00 0x00　数据长度:0x00 0x01

假设我只想读取温度值，查看上方参数，那么一帧的数据是

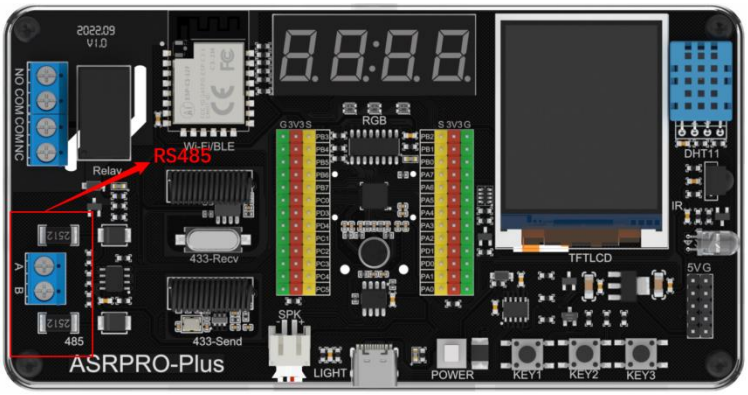
地址码:0x01　功能码:0x03 起始地址:0x00 0x01　数据长度:0x00 0x01

启动请求发送后，当主机对客户机响应时，它使用功能码来指示正常（无差错）响应或者出现某种差错（称为异常响应）。

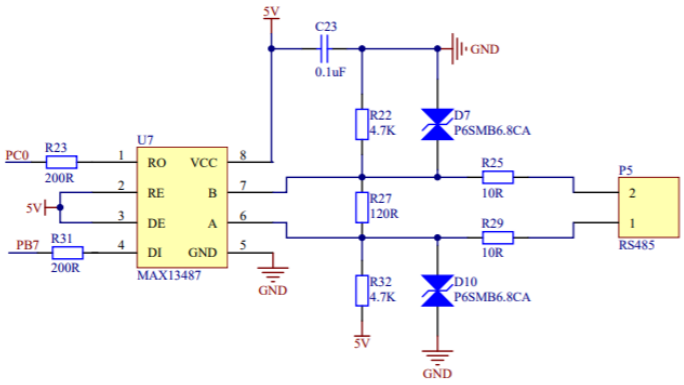


Modbus协议在本开发板基于RS485物理层。RS-485是一个仪表通信接口，由于拥有联网通信的功能，所以逐渐取代原来的RS232。在RS485通信网络中一般采用的是主从通信方式，即一个主机带多个从机。RS-485接口采用差分方式传输信号方式，可以减少干扰的影响。目前多采用的是两线制接线方式，有A、B两个端口。

ASRPRO-Plus的左侧部分就外接了一个RS485模块。下方是RS485模块的实物图和电路原理图。



RS485



ASRPRO的RS485模块通过串口PB_7和PC_0与外部从机进行通讯。

我们在编写RS485的程序前，需要先添加扩展库。点击添加扩展，找到官方扩展库中的ModbusMaster和ModbusSlave，版本选择最新，点击加载。

加载后我们就可以在扩展类别中，找到Modbus主机库和从机库对应的指令。在本范例中，我们先学习Modbus主机库指令。

1.Modbus主机初始化

这条指令可以让Modbus主机初始化，并设置串口引脚，485默认的串口引脚是PC_0 - RX1、PB_7 - TX1，波特率根据从机设备，统一设置，共用一个波特率，默认为9600。校验方式有无校验、奇校验、偶校验，一般选择无校验。ASRPRO-Plus开发板板载的485芯片支持收发自动切换。

如果485芯片的收发切换控制需要单独引脚设置的，需要设置控制引脚的有效电平，其中1代表该引脚高电平有效，0代表低电平有效。

2.Modbus主机发送请求

这些指令都属于Modbus请求读取指令，包括了读写线圈（开关输出信号）、读离散量输入、读输入寄存器、读写保存寄存器等。结合我们之前说的Modbus数据访问图和字符代码，可以更深一步地了解这些指令。

				功能码		
				码	子码	（十六进制）
数据访问	比特访问	物理离散量输入	读输入离散量	02		02
		内部比特或物理线圈	读线圈	01		01
			写单个线圈	05		05
			写多个线圈	15		0F
	16 比特访问	输入存储器	读输入寄存器	04		04
		内部存储器或物理输出存储器	读多个寄存器	03		03
			写单个寄存器	06		06
			写多个寄存器	16		10
			读/写多个寄存器	23		17
			屏蔽写寄存器	22		16
	文件记录访问		读文件记录	20	6	14
			写文件记录	21	6	15
封装接口			读设备识别码	43	14	2B

我们以指令读保存寄存器为例，介绍要输入的参数。

当我们需要获取从机的一些I/O口数据时，我们就可以使用这条指令。这条指令可以读取一个或多个寄存器的数据，对应的功能码是0x03。在这里我们不需要填写功能码，图像块指令已经帮我们写好了，我们只需要填写从机地址、起始地址和数量。

举例：读取设备地址 0x01 的温湿度值

问询帧（16 进制）：

地址码	功能码	起始地址	数据长度	校验码低位	校验码高位
0x01	0x03	0x00 0x00	0x00 0x02	0xC4	0x0B

搬运之前我们举过的读取温湿度值的例子，我们只需要输入从机地址1，起始地址0，数量2即可。超时时间指的是从启动请求到接收响应的最大时间。如果超时就响应异常。

```
// 描述: modbus读保持寄存器函数
// 参数: MB_id:从站ID; addr:地址; uslen:长度; _data:要写入的寄存器值; timeout:超时时间.
// 返回: none.
//=============================================================================
int8_t MBMaster::ReqReadHoldingRegister(uint8_t MB_id, uint16_t addr, uint16_t uslen, uint8_t
*databuf, uint16_t timeout)
{
```

```
    uint16_t temp;
    uint8_t j;

    mb_m_command[0] = MB_id;//站号
    mb_m_command[1] = 0x03;//功能号
    temp = addr;
    mb_m_command[2] = (uint8_t)(temp >> 8);//地址高
    mb_m_command[3] = addr;//地址低
    temp = uslen;
    mb_m_command[4] = (uint8_t)(temp >> 8);//寄存器高
    mb_m_command[5] = uslen;//寄存器低
    Send(mb_m_command, 6);//发送
    if (rev_process(timeout)) //在超时时间内接收到正确的数据
    {
        if (mb_m_rec_temp[0] == MB_id && mb_m_rec_temp[1] == 0x03)//返回正确数据
        {
            for (j = 0; j < uslen*2; j++)
                databuf[j] = mb_m_rec_temp[3 + j];
            return 1;
        }else{
            return -1;//不正确类型
        }
    }else{
        return 0;
    }
```

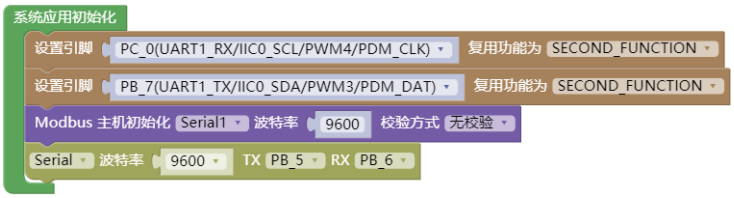
除了读寄存器外，还有写寄存器，比起之前的指令多了一个输入的数值。我们可以通过这条指令控制一些类似于温度的输出。写单个寄存器对应的功能码是0x06。

```
// 描述: modbus写单个寄存器函数
// 参数: MB_id:从站ID; addr:地址; _data:要写入的寄存器值; timeout:超时时间.
// 返回: none.
//=============================================================================
int8_t MBMaster::ReqWriteHoldingRegister(uint8_t MB_id, uint16_t addr, uint16_t  _data, uint16_t
timeout)
{
    uint16_t temp;

    mb_m_command[0] = MB_id;//站号
    mb_m_command[1] = 0x06;//功能号
    temp = addr;
    mb_m_command[2] = (uint8_t)(temp >> 8);//地址高
    mb_m_command[3] = addr;//地址低
    temp =_data;
    mb_m_command[4] = (uint8_t)(temp >> 8);//字节高
    mb_m_command[5] = _data;//字节低
    Send(mb_m_command, 6);//发送
    if (rev_process(timeout)) //在超时时间内接收到正确的数据
    {
        if (mb_m_rec_temp[0] == MB_id && mb_m_rec_temp[1] == 0x06)//返回正确数据
        {
            return 1;
        }else{
            return -1;//不正确类型
        }
    }else{
        return 0;
    }
}
```

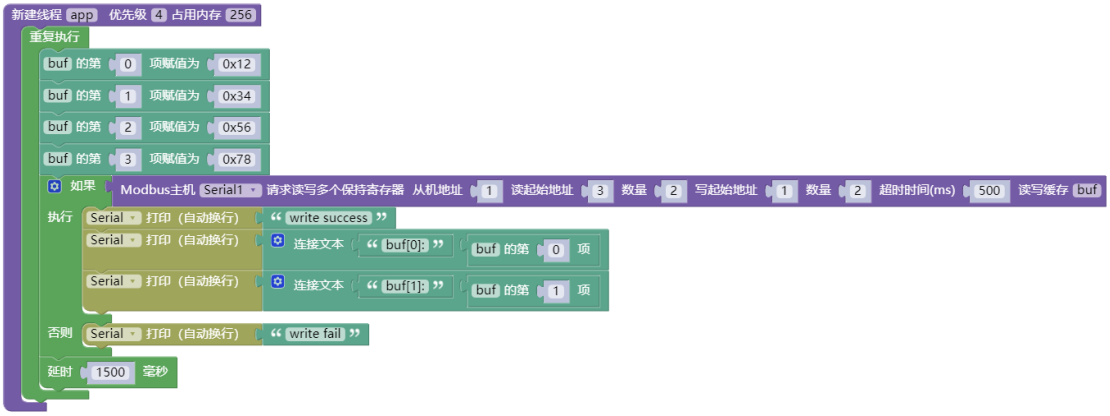
如果需要控制继电器等，可以用写线圈指令。

当需要控制按键等时，用读离散量输入指令。

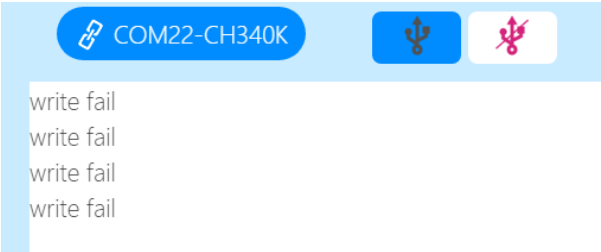
接下来我们来看一看范例3.11Modbus主机案例：



我们先对串口进行设置，Modbus主机在串口1初始化，输出信息在串口0显示。



如果485没有外接任何设备，那么请求超时500ms就会失败，串口会输出write fail。

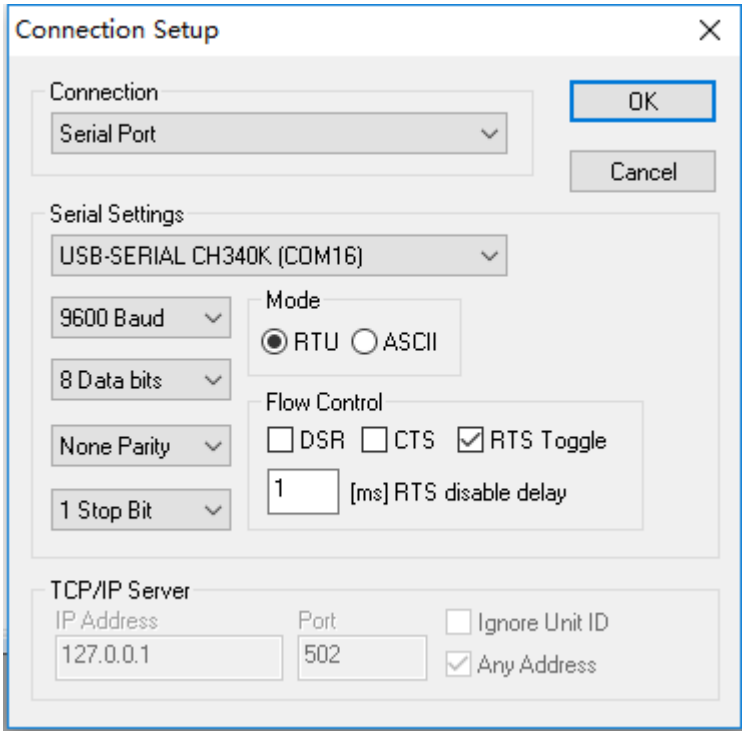


如果请求被顺利响应，那么串口会输出write success和12，34。

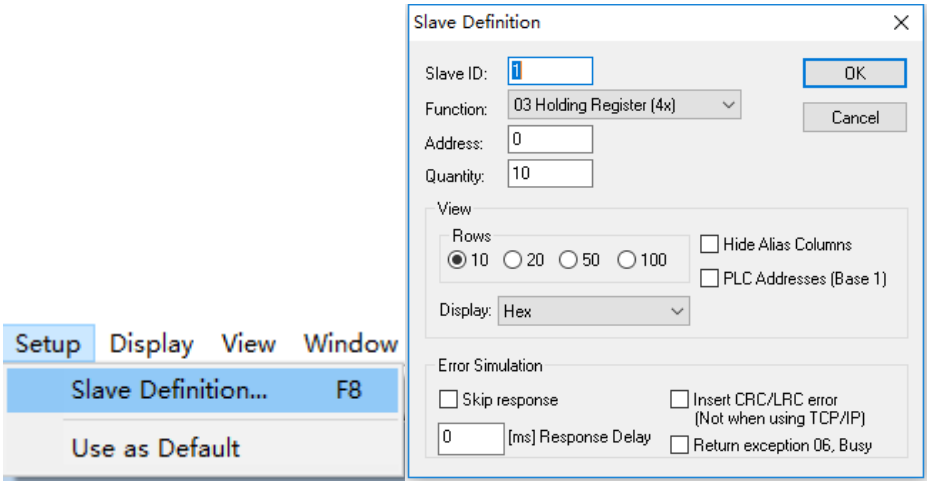
具体的连接控制测试我们在范例3.13Modbus控制继电器中说明。

如果我们没有Modbus从设备，可以用软件来模拟，常用的软件有Modbus Slave从机模拟软件。当我们使用从机模拟软件来测试时，程序要做简单修改，应该使用串口0进行测试，修改如下所示：

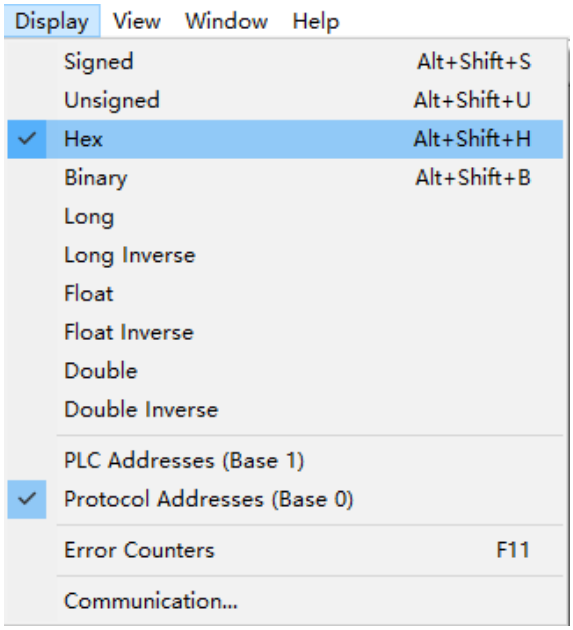
1. 需要先设置串口连接参数，保持和程序中设置的一致。（注意校验方式，我们一般为无校验，软件默认是奇校验，需要修改）



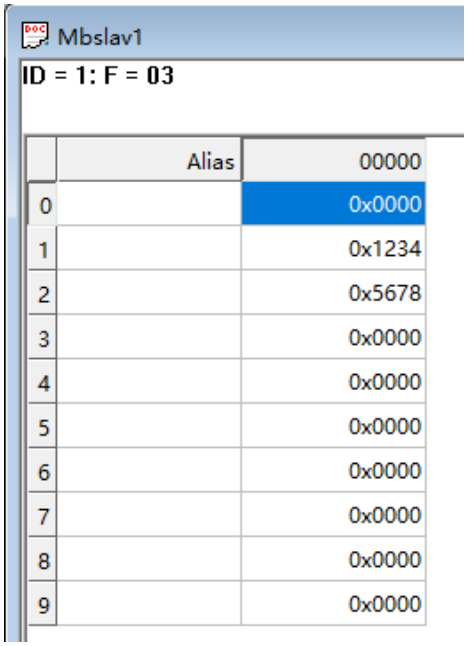
2. 设置从机属性



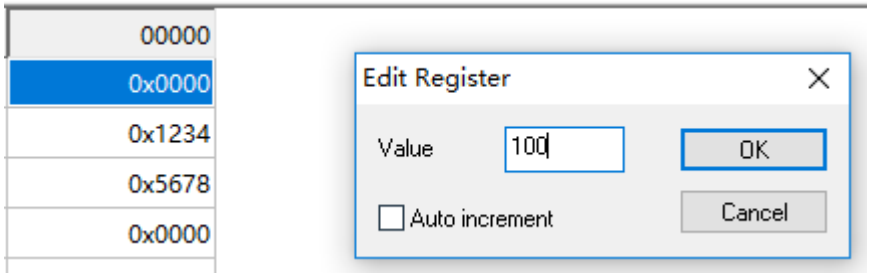
3. 显示格式可以根据需求设置



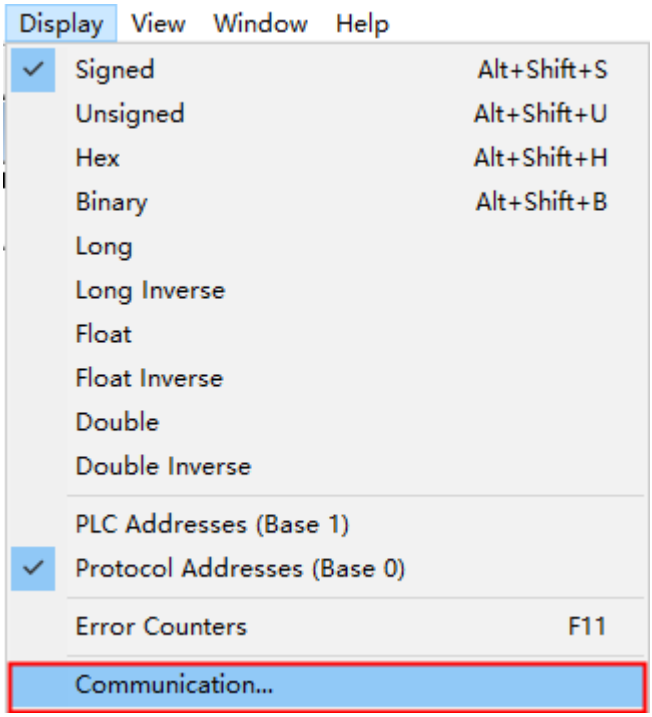
4. 数据显示窗口就会显示案例中对应寄存器的数值。

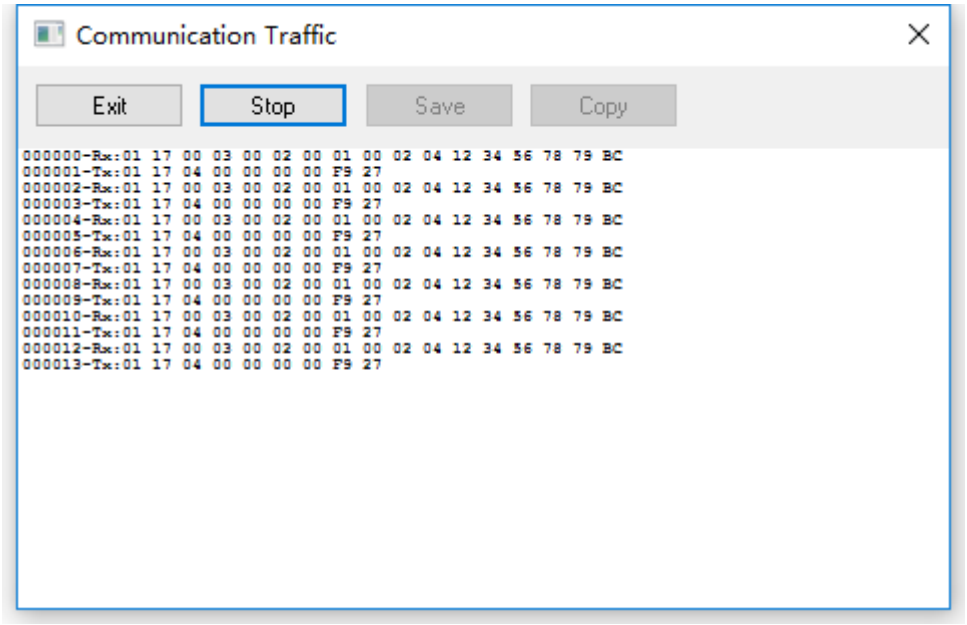


5. 双击数据表，可以手动修改数值。



6. 我们可以查看通讯的原始数据


Display
View
Window
Help
Signed Alt+Shift+S
Unsigned Alt+Shift+U
Hex Alt+Shift+H
Binary Alt+Shift+B
Long
Long Inverse
Float
Float Inverse
Double
Double Inverse
PLC Addresses (Base 1)
Protocol Addresses (Base 0)
Error Counters F11
Communication...

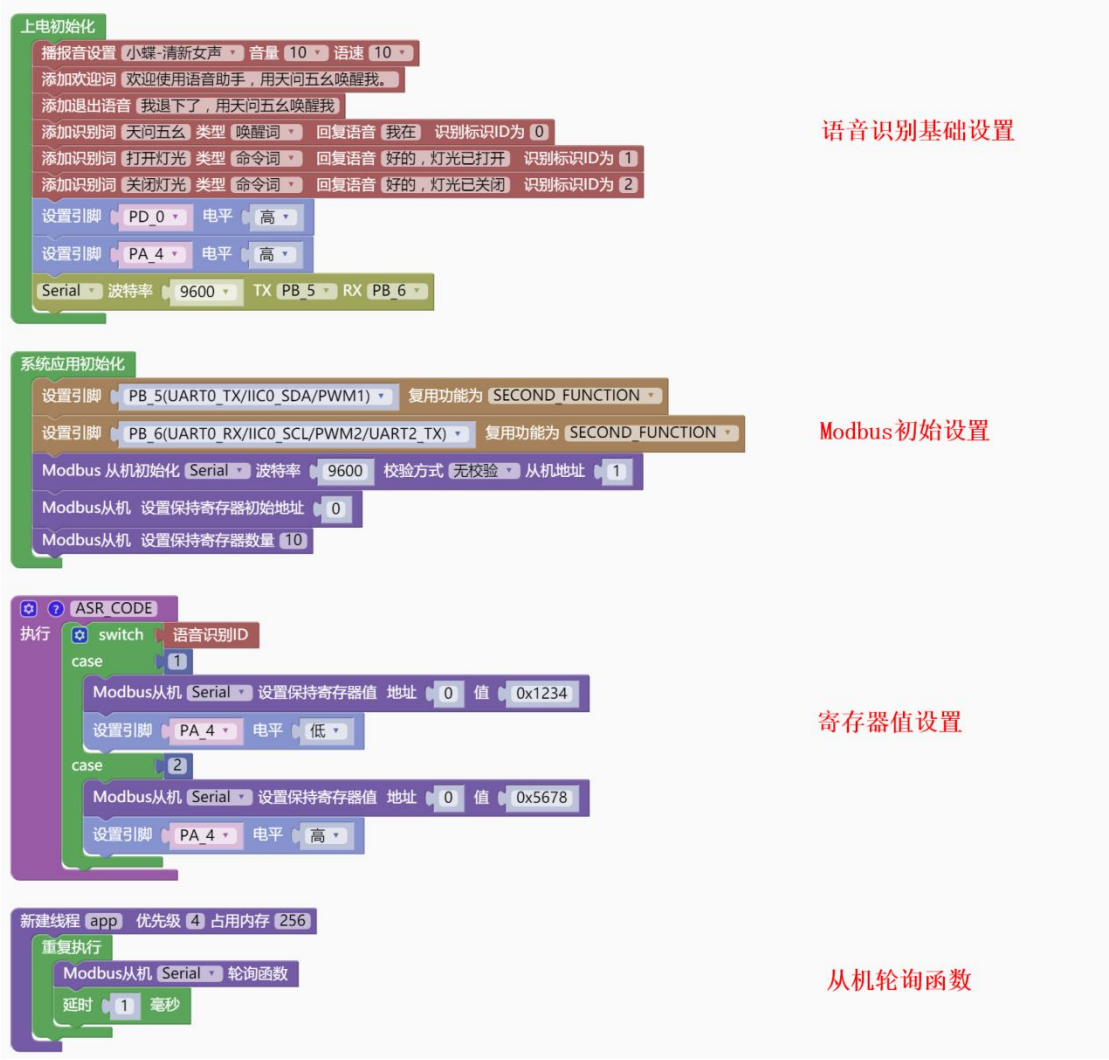

Communication Traffic
Exit
Stop
Save
Copy
000000-Rx:01 17 00 03 00 02 00 01 00 02 04 12 34 56 78 79 BC
000001-Tx:01 17 04 00 00 00 00 F9 27
000002-Rx:01 17 00 03 00 02 00 01 00 02 04 12 34 56 78 79 BC
000003-Tx:01 17 04 00 00 00 00 F9 27
000004-Rx:01 17 00 03 00 02 00 01 00 02 04 12 34 56 78 79 BC
000005-Tx:01 17 04 00 00 00 00 F9 27
000006-Rx:01 17 00 03 00 02 00 01 00 02 04 12 34 56 78 79 BC
000007-Tx:01 17 04 00 00 00 00 F9 27
000008-Rx:01 17 00 03 00 02 00 01 00 02 04 12 34 56 78 79 BC
000009-Tx:01 17 04 00 00 00 00 F9 27
000010-Rx:01 17 00 03 00 02 00 01 00 02 04 12 34 56 78 79 BC
000011-Tx:01 17 04 00 00 00 00 F9 27
000012-Rx:01 17 00 03 00 02 00 01 00 02 04 12 34 56 78 79 BC
000013-Tx:01 17 04 00 00 00 00 F9 27

范例3.11 MODBUS从机案例

一、范例功能

本范例通过深化学习Modbus协议，实现Modbus从机功能，达成学习Modbus从机应用案例的使用目的。

二、范例分析



三、范例详解

本范例主要介绍当ASRPRO作为Modbus从机时，如何编写程序。

我们在编写Modbus从机的程序前，需要先添加扩展库。点击添加扩展，找到官方扩展库中的ModbusSlave，版本选择最新，点击加载。我们在Modbus主机范例中已经成功添加，这里不再赘述。

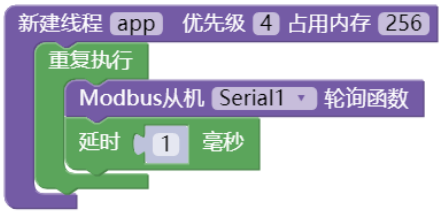
1.Modbus从机初始化

这条指令可以设置当ASRPRO作为Modbus从机时的串口和波特率，同时设置校验方式，一般设置无校验。从机地址一般设置为1，与主机程序对应。

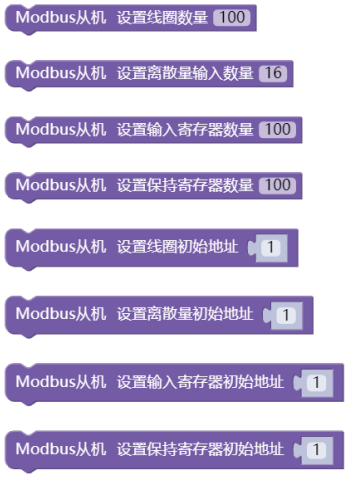
如果485芯片的收发切换控制需要单独引脚设置的，需要设置控制引脚的有效电平，其中1代表该引脚高电平有效，0代表低电平有效。

2.从机轮询函数

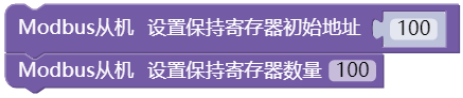
这条指令处理主机的请求。一般放到线程中，每1毫秒判断一次。



3.线圈、离散量、寄存器初始设置



这些指令可以对线圈、离散量输入、输入寄存器、保持寄存器的初始地址和数量进行设置。以设置保存寄存器为例，我们可以进行如下设置，对于主机来说，初始地址就是100，最多可以有100个保持寄存器。注意区分从机地址和寄存器初始地址。

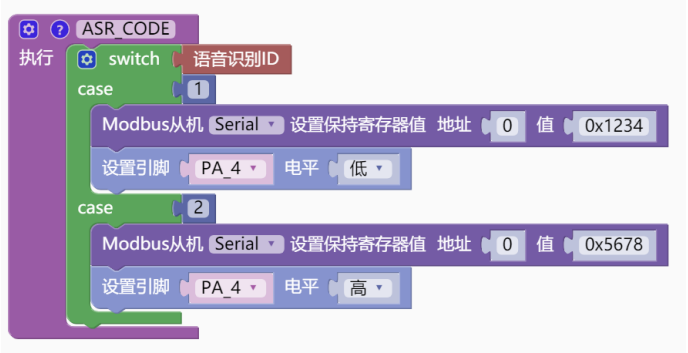


4.线圈、离散量、寄存器地址和数值设置

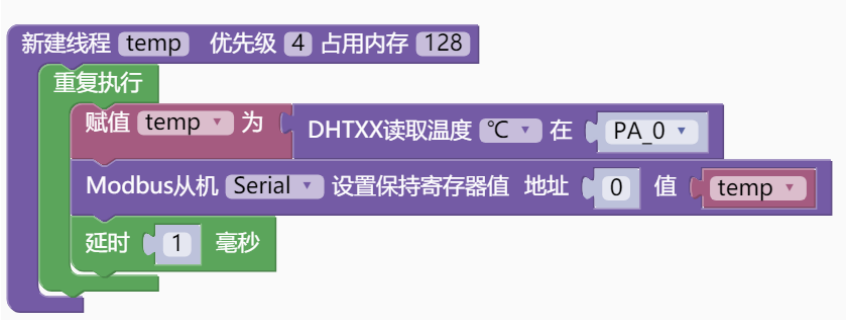
这些指令可以对线圈、离散量、输入寄存器、保持寄存器的地址和数值进行获取或设置。

以设置保持寄存器的地址和数值为例，我们在上方设置了初始地址为100，数量为100，那么这里的地址的范围就是100-199，后面是输入的数值。

在本范例中，我们是用语音进行测试，并设置了保持寄存器的值。

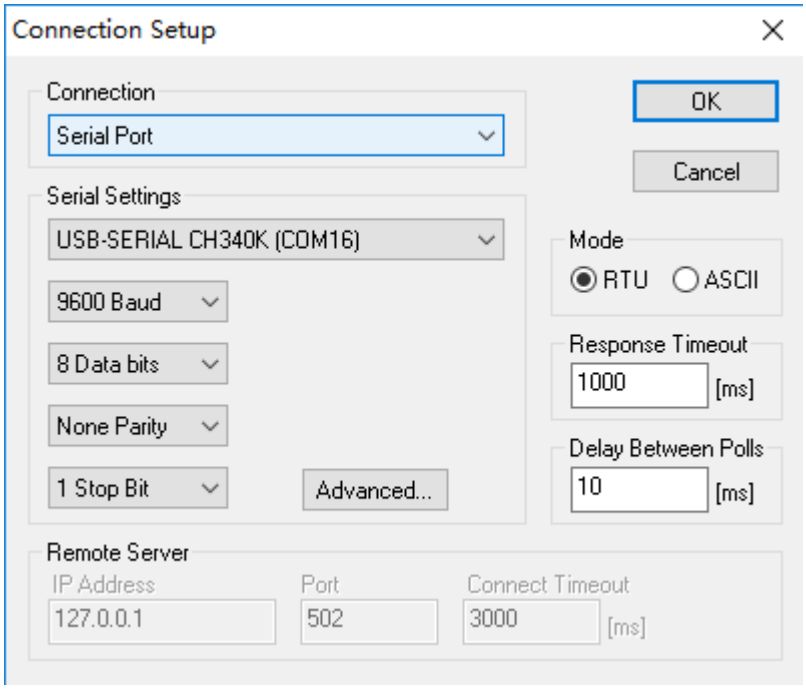


当然我们也可以新建一个线程，去设置保持寄存器的值，如下所示。

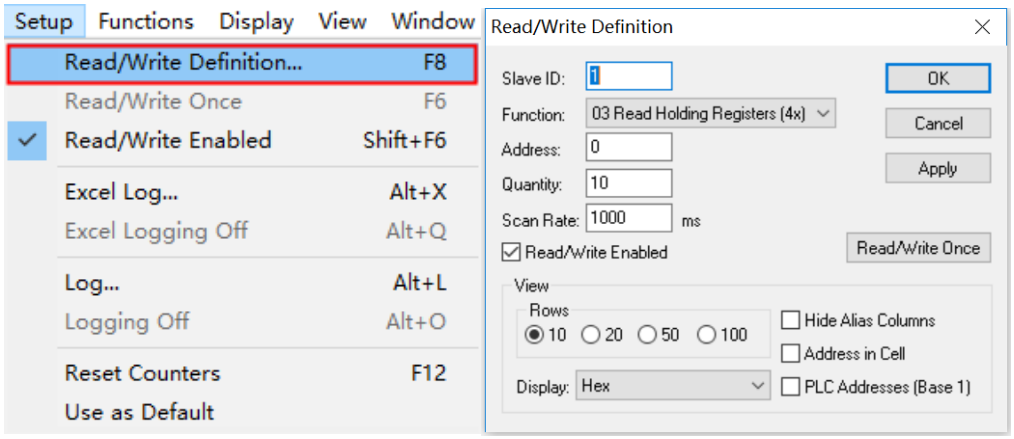


我们可以用另外一个主机设备来读写这个从机范例的寄存器数值，也可以用软件来模拟，常用的软件有Modbus Poll主机模拟软件。

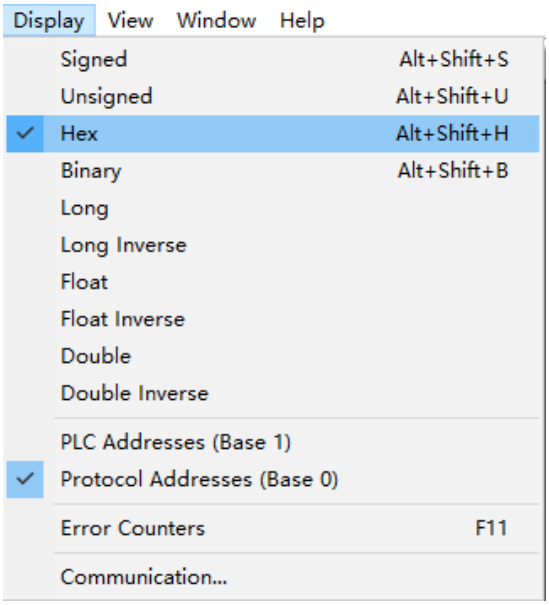
1.需要先设置串口连接参数，保持和程序中设置的一致。（注意校验方式，我们一般为无校验，软件默认是奇校验，需要修改）



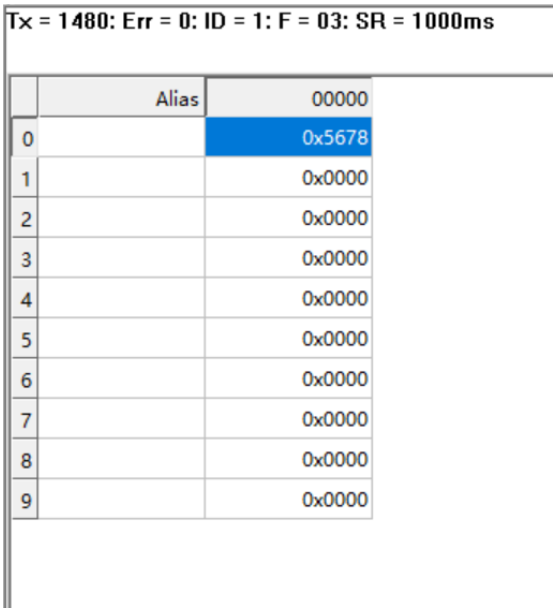
2.设置读写定义，设置从机地址、功能码、对应的寄存器地址、数量等。注意上位机读保持寄存器最多设置50个，因为驱动程序里收发数据缓存最大为50，如果设置过多会让下位机死机。



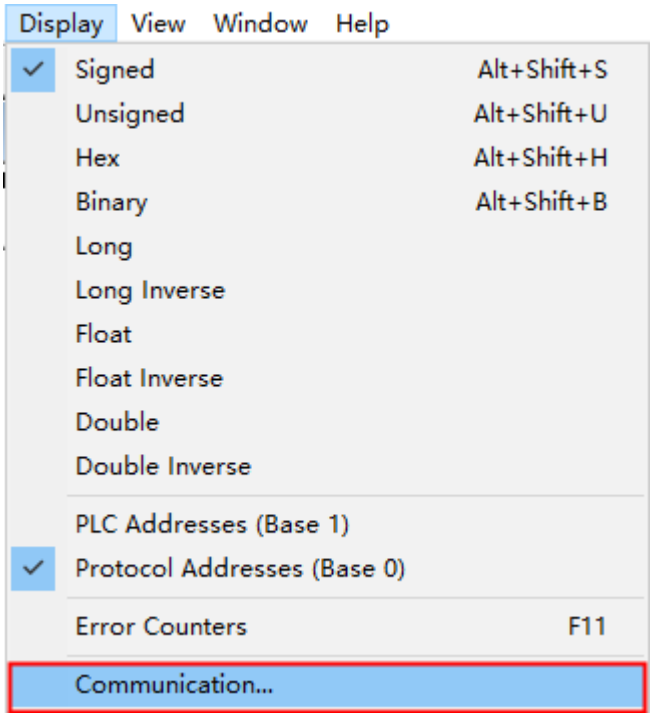
3.显示格式可以根据需求设置

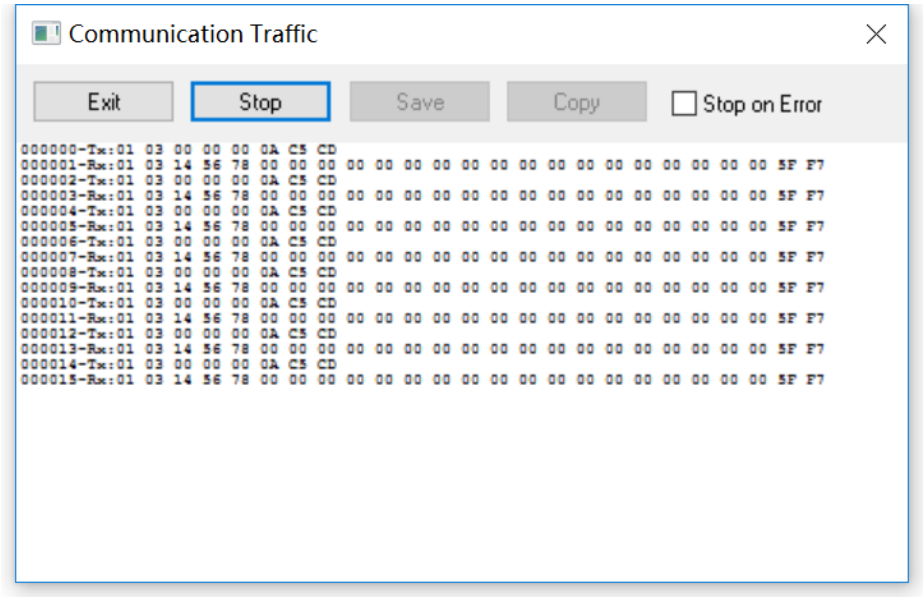


4.数据显示窗口就会显示案例中寄存器0的数值。



5.我们可以查看通讯的原始数据


Display
View
Window
Help
Signed Alt+Shift+S
Unsigned Alt+Shift+U
Hex Alt+Shift+H
Binary Alt+Shift+B
Long
Long Inverse
Float
Float Inverse
Double
Double Inverse
PLC Addresses (Base 1)
Protocol Addresses (Base 0)
Error Counters F11
Communication...

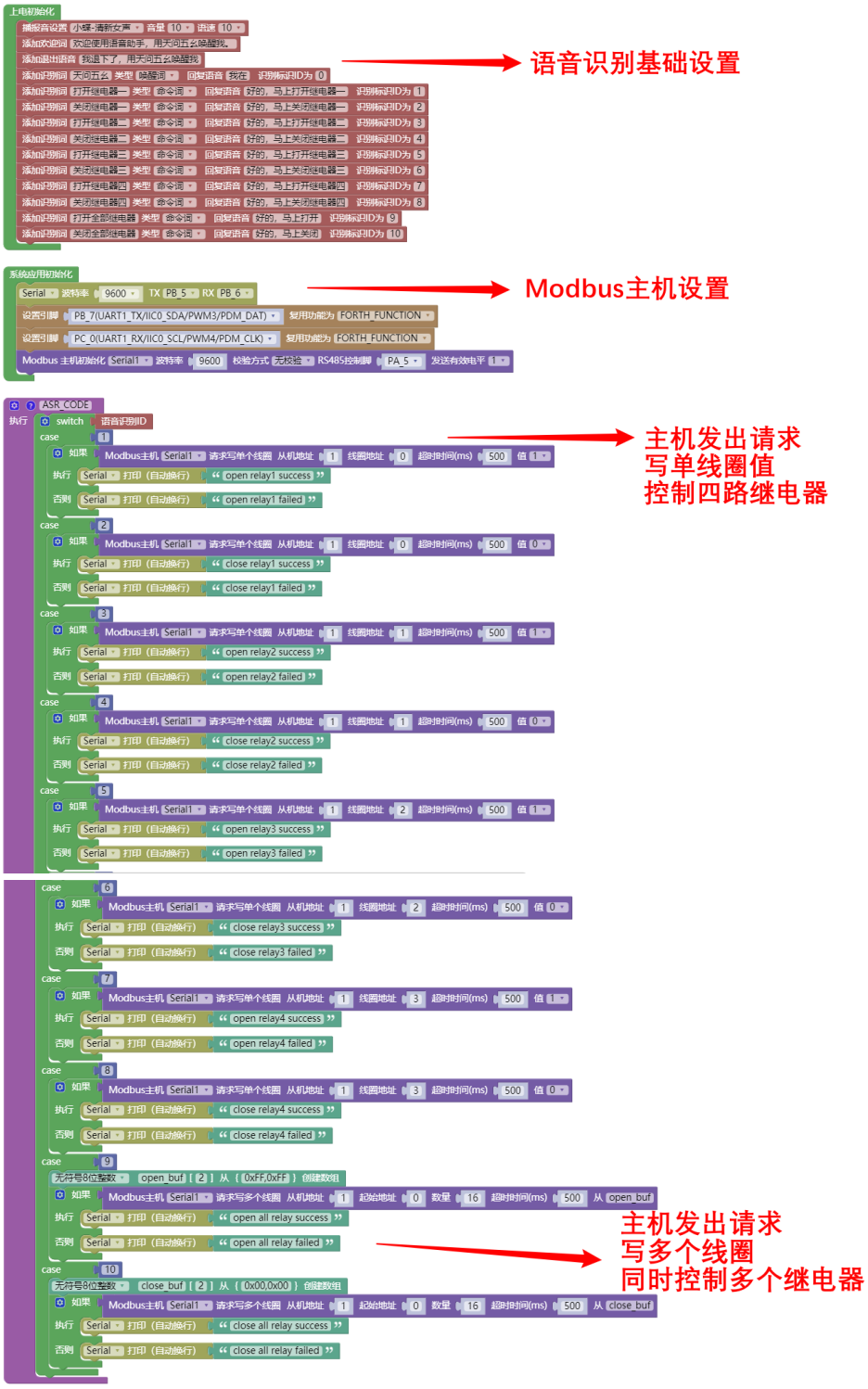

Communication Traffic
Exit
Stop
Save
Copy
Stop on Error
000000-Tx:01 03 00 00 00 0A C5 CD
000001-Rx:01 03 14 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5F F7
000002-Tx:01 03 00 00 00 0A C5 CD
000003-Rx:01 03 14 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5F F7
000004-Tx:01 03 00 00 00 0A C5 CD
000005-Rx:01 03 14 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5F F7
000006-Tx:01 03 00 00 00 0A C5 CD
000007-Rx:01 03 14 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5F F7
000008-Tx:01 03 00 00 00 0A C5 CD
000009-Rx:01 03 14 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5F F7
000010-Tx:01 03 00 00 00 0A C5 CD
000011-Rx:01 03 14 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5F F7
000012-Tx:01 03 00 00 00 0A C5 CD
000013-Rx:01 03 14 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5F F7
000014-Tx:01 03 00 00 00 0A C5 CD
000015-Rx:01 03 14 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5F F7

范例3.12 MODBUS控制继电器

一、范例功能

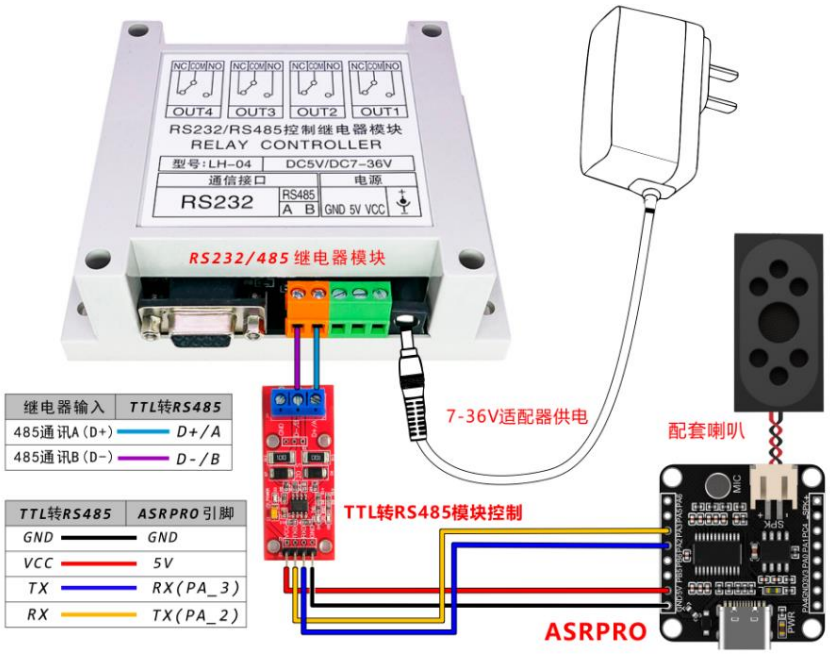
本范例通过485模块，实现使用ASRPRO控制MODBUS LH-04串口继电器模块的功能，达成深化学习Modbus主从机控制程序编写的目的。

二、范例分析



三、范例详解

本范例介绍如何使用ASRPRO控制MODBUS LH-04串口继电器模块。下方是实物连接图。如果用的是ASRPRO-Plus，可以直接将485模块的AB引脚与485继电器模块相连。



想要进行Modbus实际案例的应用，必须查看使用模块的Modbus通信协议，一般厂商都会在资料中附带。

本范例中使用的是LH-04 继电器模块。

第一部分是关于通信参数的设置。波特率的设置影响着发送数据相邻字节的时间。默认是设置9600。当接多个设备通讯时，就要注意在不同通讯速度下间隔时间的设置。

通信参数：

1、波特率：默认是 9600 支持 300/600/1200/2400/4800/9600/14400/19200/38400/56000/57600/115200bps，
2、其他参数：校验位：N，数据位：8，停止位：1
3、一条命令中相邻字节发送允许间隔时间如下：

通讯速度	发送字符间允许间隔时间
9600bps 以下	约 10ms
19200bps	约 6ms
19200bps 以上	约 4ms

第二部分是关于模块的寄存器地址、线圈地址等。在本范例中，我们要对四路继电器的线圈地址进行设置。例如我们要对OUT3继电器进行设置，就可以和下方指令一样进行参数修改。从机地址为1，线圈地址为2，超时时间500ms。

模块寄存器地址

寄存器地址	定义	备注
0000	继电器控制数据	低字节的低 4 位有效
0100	模块设备地址	低字节有效，01-FF
0200	模块通讯速度	低字节有效，数值与速度如下对应 00:300bps 01:600bps 02:1200bps 03:2400bps 04:4800bps 05:9600bps 06:14400bps 07:19200bps 08:38400bps 09:56000bps 0A:57600bps 0B:115200bps
0000	OUT1 单线圈地址	对应功能码 05
0001	OUT2 单线圈地址	对应功能码 05
0002	OUT3 单线圈地址	对应功能码 05
0003	OUT4 单线圈地址	对应功能码 05

注意到这条指令，最后还可以设置值为0或1。我们打开VSCODE查看源码，找到ReqWriteCoil这个函数，也就是Modbus写线圈函数，发现线圈值，也就是_databuf，当设置为1时，写入的是0xff 0x00；当设置为0时，写入的是0x00 0x00。

```
// 描述: modbus写线圈函数
// 参数: MB_id:从站ID; addr:地址; _databuf:线圈值; timeout:超时时间.
// 返回: none.
//=============================================================================
int8_t MBMaster::ReqWriteCoil(uint8_t MB_id, uint16_t addr,  uint8_t _data, uint16_t timeout)
{
    uint16_t temp;

    mb_m_command[0] = MB_id;//站号
    mb_m_command[1] = 0x05;//功能号
    temp = addr;
    mb_m_command[2] = (uint8_t)(temp >> 8);//地址高
    mb_m_command[3] = addr;//地址低
    mb_m_command[4] = (_data?0xff:0x00);
    mb_m_command[5] = 0x00;
    Send(mb_m_command, 6);//发送
    if (rev_process(timeout)) //在超时时间内接收到正确的数据
    {
        if (mb_m_rec_temp[0] == MB_id && mb_m_rec_temp[1] == 0x05)//返回正确数据
        {
            return 1;
        }else{
            return -1;//不正确类型
        }
    }else{
        return 0;
    }
}
```

我们再次查看继电器的Modbus通信协议，发现继电器吸合是FF00，继电器断开是0000。由于写入的值是两个字节的数据，对于FF00来说，FF是高字节，00是低字节，所以我们要分别写入0xff和0x00。当然也有部分模块会从低字节开始写入，具体开手册来决定。

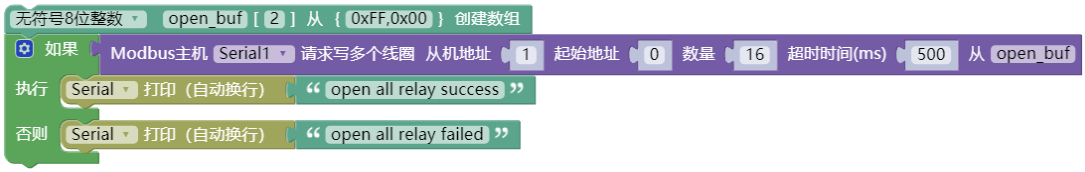
字节顺序号	内容	说明
1	模块地址码	模块的地址， 01 ：继电器 1 地址 02 ：继电器 2 地址 …… 0E ：继电器 14 地址 0F ：继电器 15 地址
2	功能码 05	固定
3	单继电器地址高字节 00	固定
4	单继电器地址低字节	OUT1:00;OUT2:01;OUT3:02;OUT4:03
5	输出数据 WORD 的高字节	继电器吸合：FF00
6	输出数据 WORD 的低字节	继电器断开：0000
7	CRC 校验码的低字节	CR 校验码 （前面所有数据的 CRC 校验码）
8	CRC 校验码的高字节	

当我们想一次性写入所有继电器的值时，我们就可以用写入多线圈指令。

查看该继电器手册发现，地址码从00到0F，一共16个，输入0表示断开，输入1表示吸合。此时我们就可以使用上方这条指令，一次性控制所有继电器。

控制所有继电器（功能码 0F，地址 0，长度 16，数值 0 对应断开，数值 1 对应吸合）

输入的数值是一个数组，如果我们想打开继电器，数组里就写入两个字节，0xff和0x00。



范例3.13 SSD1306 OLED屏案例

一、范例功能

本范例通过语音控制OLED屏，实现SSD1306驱动的OLED显示中英文、绘制图案的功能，达成学习OLED屏程序编写的目的。

二、范例分析

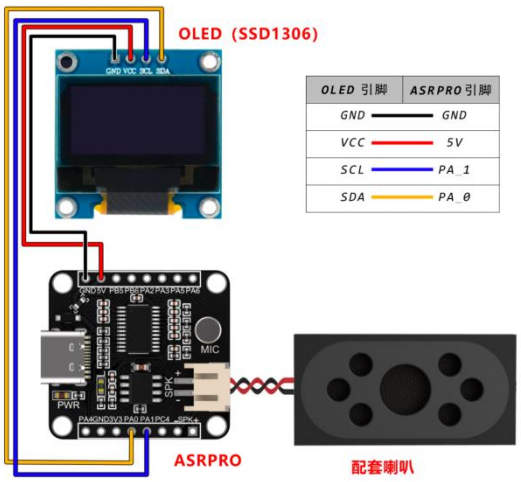
三、范例详解

本范例介绍如何编写SSD1306 OLED屏的程序。

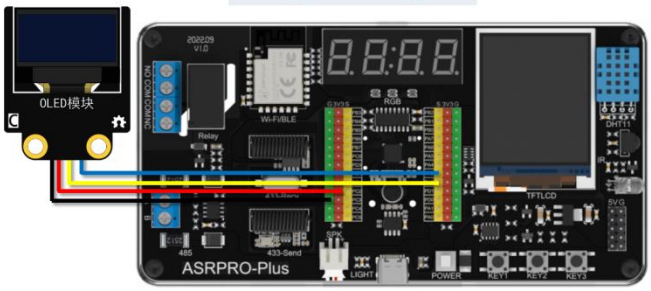
OLED显示，主要是通过电场驱动，有机半导体材料和发光材料通过过载流子注入和复合后实现发光。从本质上来说，就是通过ITO玻璃透明电极作为器件阳极，金属电极作为阴极，通过电源驱动，将电子从阴极传输到电子传输层，空穴从阳极注入到空穴传输层，之后分迁移到发光层，二者相遇后产生激子，让发光分子激发，经过辐射后产生光源。简单来说，一块OLED屏幕，就是由百千万个“小灯泡”组成。

相比于LCD的背发光，OLED属于自发光，且每个像素点都可以自由关闭和开启，因此色彩显示更为鲜艳明显。

这里提供128*64像素的SSD1306屏与ASRPRO核心板/ASRPRO Plus的硬件连接图：

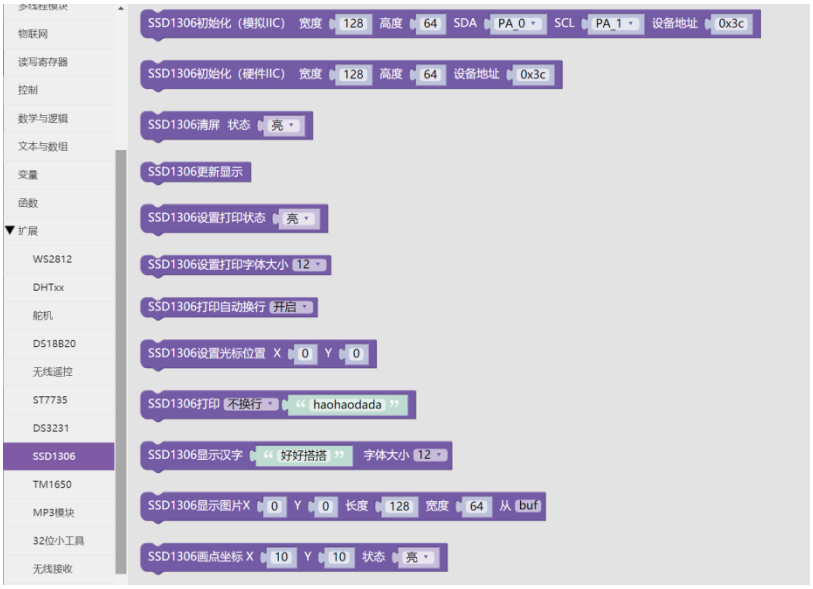


ASRPRO-Plus		OLED
PA2	-	SDA
PA3	-	SCL
VCC	-	3V3
GND	-	GND



我们在编写OLED屏的程序前，需要先添加扩展库。点击添加扩展，找到官方扩展库中的SSD1306，版本选择最新，点击加载；注意扩展库版本要最新，不然无法下载。

加载后我们就可以在扩展类别中，找到OLED屏对应的指令。



这是12864 OLED屏的具体参数。

驱动芯片	SSD1306
支持接口	I2C
分辨率	128×64
显示尺寸	0.96英寸
显示颜色	白光/蓝光
外形尺寸	27.5×27.8（mm）
玻璃尺寸	26.7×19.26×1.4（mm）
显示区域	21.74(W)×10.864（mm）
点间距	0.17×0.17（mm）
点大小	0.15×0.15（mm）
管脚数	4针
功耗	正常显示功耗为 21mA-28MAX
工作电流	正常工作时电流在20ma左右休眠时在ua级电流
视角	全视角
工作温度	-20℃~70℃
存储温度	-30℃~80℃
工作电压	5V / 3.3V

OLED屏与ASRPRO-Plus连接时，引脚连接说明如下：

PIN	SYMBOL	Descriptions
1	GND	Ground of Logic Circuit（逻辑电路接地）
2	VDD	Power Supply for Logic（逻辑电源）
3	SCK	Serial clock input（串行时钟输入）
4	SDA	Serial data inpu（串行数据）

1.OLED屏初始化

这条指令可以初始化OLED屏，默认分辨率为宽度128*高度64，通过模拟IIC传输数据，可以设置SDA和SCL的引脚，设备地址设置规则请参考下方从机地址数据格式。

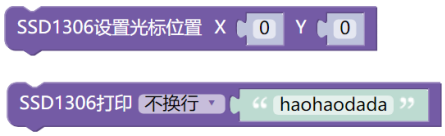
b7	b6	b5	b4	b3	b2	b1	b0
0	1	1	1	1	0	SA0	R/W#

“SA0”位为从机地址提供扩展位对应OLED屏的地址选择引脚，相应的地址“0111100”（即0x3c）或“0111101”（0x3d）可供选择，在有些驱动中对应的8位地址是0x78或0x7A。

2.OLED屏清屏

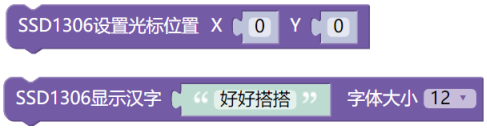
这条指令可以让OLED屏清屏，可以设置所有像素点亮或者所有像素点灭。

3.OLED屏显示文本



OLED显示文本一般使用上面两条指令。这两条指令可以设置文本的起始位置和文本内容。其中OLED屏的分辨率以128*64为例，x的范围是0-127，y的范围是0-63；OLED屏打印的内容，常规的ASCII码均可以填入，例如数字、英文、常用符号。

4.OLED屏显示中文



OLED显示文本一般使用上面两条指令。第二条指令可以设置显示中文的内容和字体大小。汉字的字体大小可以自由设置。假如设置为12，那么一个汉字大小就是12*12。

8.OLED屏显示点、线、图形

这些指令可以绘制点、线和各种图形。

9.OLED屏显示图片

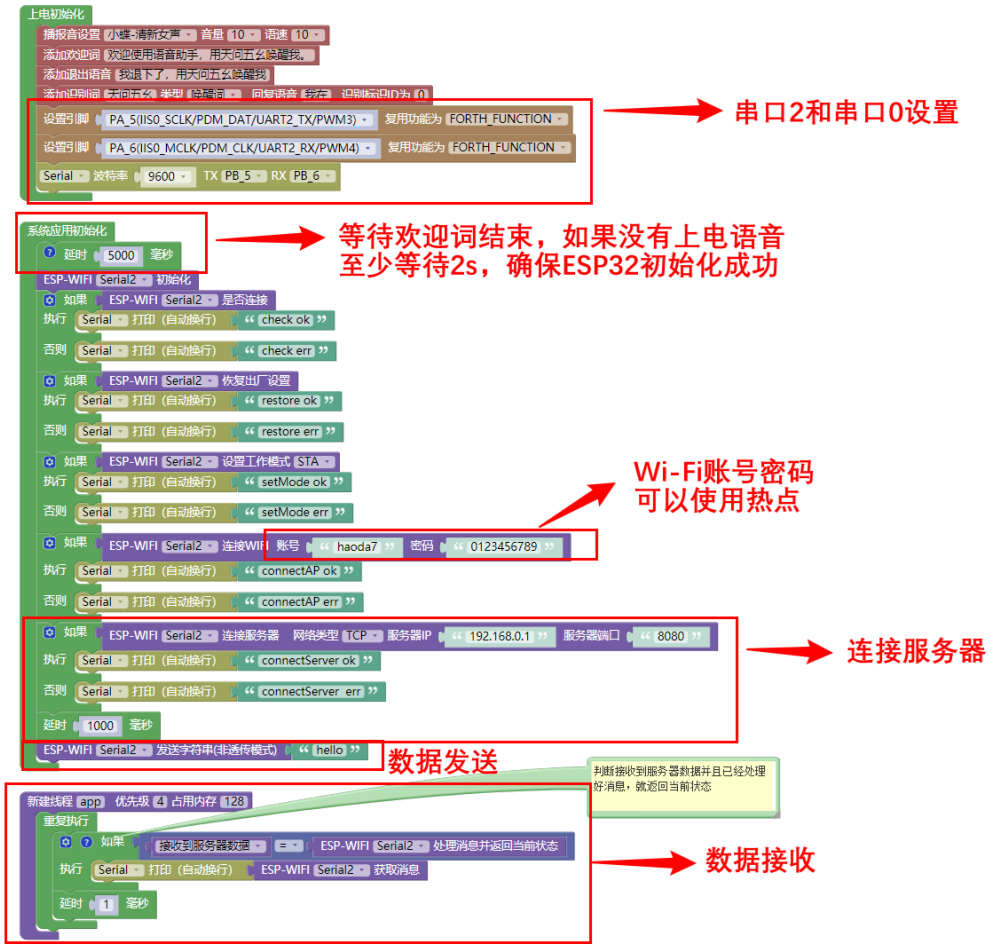
OLED显示图片的使用方法与彩屏基本相同，可参考彩屏案例，注意大小要小于128*64

范例3.14 Wi-Fi TCP UDP通讯

一、范例功能

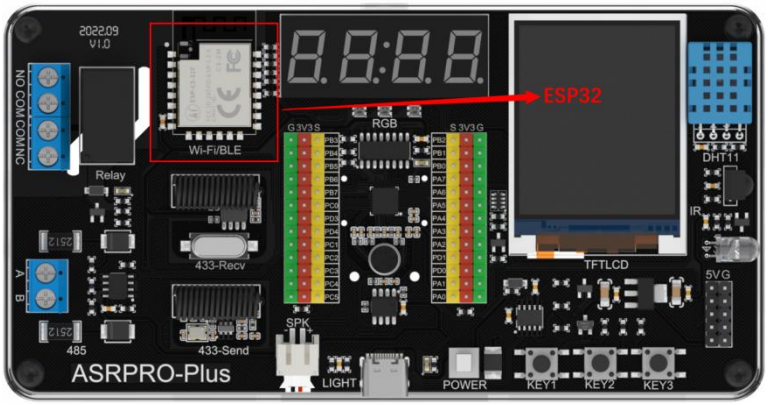
本范例通过学习ESP32 Wi-Fi扩展库，测试TCP、UDP的基本通讯，掌握Wi-Fi库的基本使用。

二、范例说明



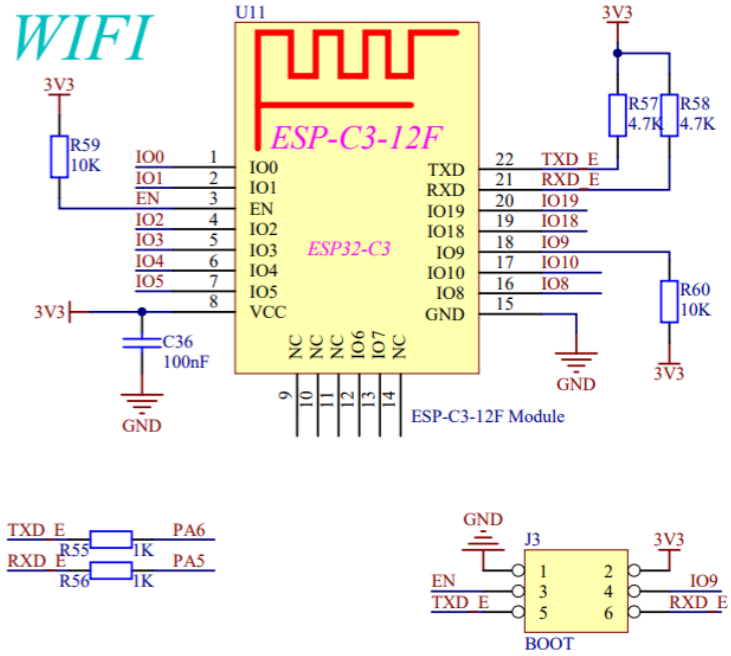
三、硬件详解

ASRPRO PLus外接了一块ESP32-C3芯片，同时支持Wi-Fi和蓝牙功能。

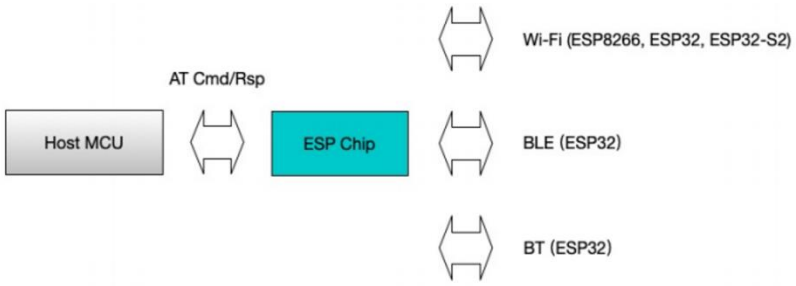


ESP32-C3是一款安全稳定、低功耗、低成本的物联网芯片，搭配RISC-V 32位单核处理器，支持2.4GHz Wi-Fi和Bluetooth5（LE）。对这两者的双重支持降低了设备配网难度，适用于广泛的物联网应用场景。

下方是WiFi/BLE模块的原理图，可以得知，实际是通过PA5-TX2、PA6-RX2进行通讯。



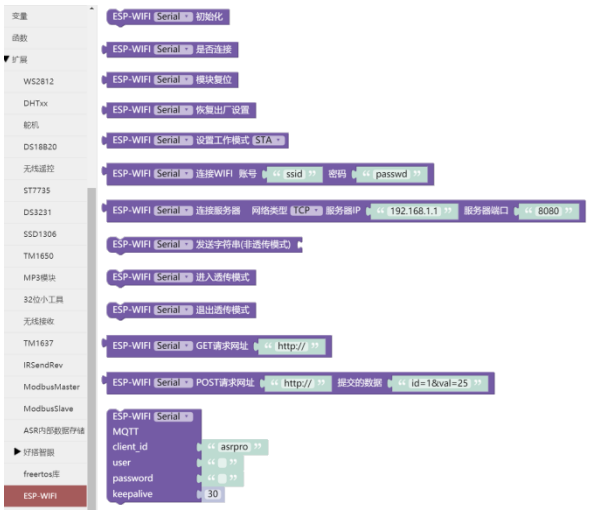
ESP32通过AT指令进行串口通讯。由于AT指令较多，不易记忆，所以我们可以直接使用ESP32 的扩展库，AT指令基于乐鑫AT_Bin_V2.3，有关对应的AT指令细节可以查看乐鑫对应的在线文档资料。



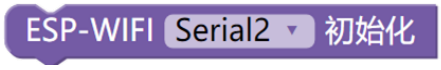
四、指令学习

我们在编写ESP32模块的程序前，需要先添加扩展库。点击添加扩展，找到官方扩展库中的ESP-WIFI，点击加载。

加载后我们就可以在扩展类别中，找到ESP32模块Wi-Fi的AT库对应的指令。用VSCode打开查看，我们也可以发现，实际上每一条指令也是包含了发送AT指令，例如ESP-WiFi是否复位，就是发了一个AT指令“AT+RST”。

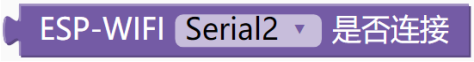


1.ESP-WIFI初始化



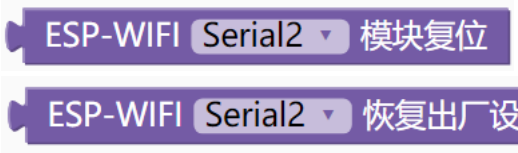
这条指令可以初始化ESP模块，上电初始化的过程需要一段时间，所以在程序中一般先等待2s。ESP32模块一般设置在串口2，默认波特率为115200。

2.ESP-WIFI模块连接确认



这条指令可以判断ESP模块是否已连接，即发送测试指令“AT”，如果正常连接则返回“OK\r\n”。

3.ESP-WIFI模块重置



第一条指令可以让ESP模块复位并返回，发送的AT指令是“AT+RST”；

第二条指令可以让ESP模块恢复出厂设置并复位，然后返回，发送的AT指令是“AT+RESTORE”，注意此条指令需要等待5s，确保重启完成。

4.ESP-WIFI模块工作模式设置

这条指令可以设置ESP模块是STA模式/AP模式/STA+AP共存模式。

所谓AP，也就是无线接入点，是一个无线网络的创建者，是网络的中心节点。一般家庭或办公室使用的无线路由器就是一个AP。AP模式下，你可以把ESP模块看成一个又不能接网线、又不能接入网络的路由器，只能等待其他设备的链接。手机、PAD、电脑等设备可以直接连上模块，可以很方便对用户设备进行控制。

STA，也就是站点（station）。每一个连接到无线AP的终端(如笔记本电脑、PDA及其它可以联网的用户设备)都可称为一个站点。站点在无线局域网中一般为客户端，可以是装有无线网卡的计算机，也可以是有Wi-Fi模块的智能手机，可以是移动的，也可以是固定的。当作为STA模式时，模块只能接入到路由器或者上位服务器进行数据上传。

而STA+AP模式下，Wi-Fi模块既可以作为AP，让客户的手机或者计算机接入，同时该模块又可以作为一个STA接入到路由器或者上位服务器进行数据上传。

需要注意的是，模块在AP和模块做STA时的MAC地址是不同的，所以在模块内部看到模块做AP时的MAC地址与在路由器里面去看到的模块作为STA时的MAC地址不同。

5.ESP模块连接WIFI

这条指令让ESP模块连接Wi-Fi。默认的账号和密码都需要是英文，不能使用中文。可以使用手机热点进行连接，但注意暂不支持5G。

6.ESP模块连接服务器

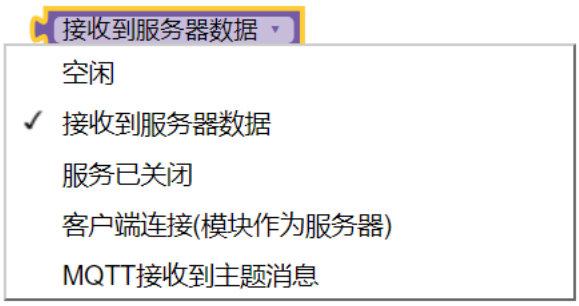
我们可以通过下方指令与服务器进行连接，可以选择TCP或者UDP协议。我们可以选择网络协议类型、是否开启多连接、填写服务器IP（severIP）和服务器端口（serverPort），然后通过AT指令“AT+CIPSTART”，也就是建立TCP连接、UDP传输。

TCP和UDP协议如果作简单区分，就是TCP在连接前会先发请求是否能发数据，就像打电话先拨号再建立连接，且只能是一对一，优点是服务比较可靠；而UDP则是发数据不需要建立连接，可以一对一、一对多的发送，与此同时不能保证数据可靠的交付。

7.ESP模块向服务器发送数据

与服务器连接后，我们就可以通过下方指令向服务器发送数据。这条指令对应的AT指令是“AT+CIPSEND”，也就是在普通传输模式或Wi-Fi 透传模式下发送数据，该指令默认是在非透传模式下发送。

8.判断接收到服务器数据

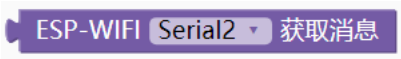


这条指令可以判断是否接收到服务器的一帧数据。

9.判断从服务器接收的数据是否已经处理完成并返回

这条指令可以确认接收消息完成并返回消息类型，不能用于透传模式。

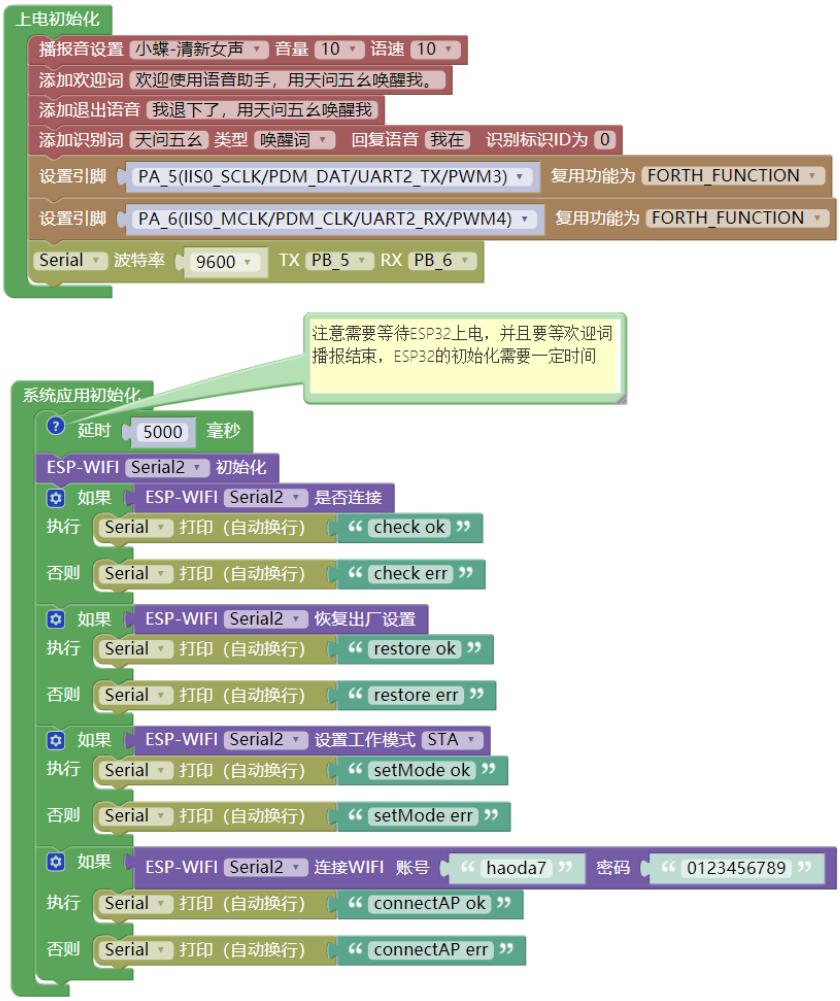
10.从服务器接收的消息



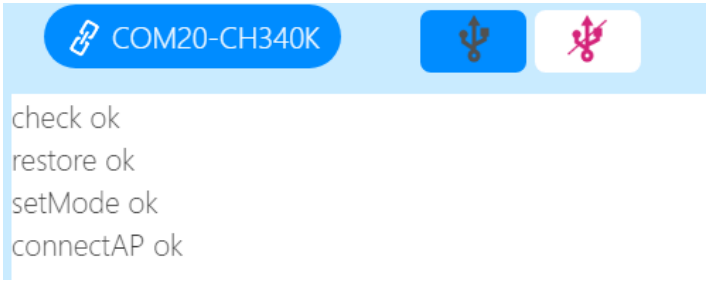
这条指令是从服务器接收到的消息。

五、程序详解

我们编写一个简单的程序，对ESP32 AT扩展库的这些指令进行初步测试。ESP32上电初始化需要一段时间，并且由于是单核，ESP-WIFI初始化运行和欢迎词播报无法同时进行。为了防止上电后欢迎词的播报出现卡顿现象，建议等待一段时间后再进行ESP-WIFI初始化。

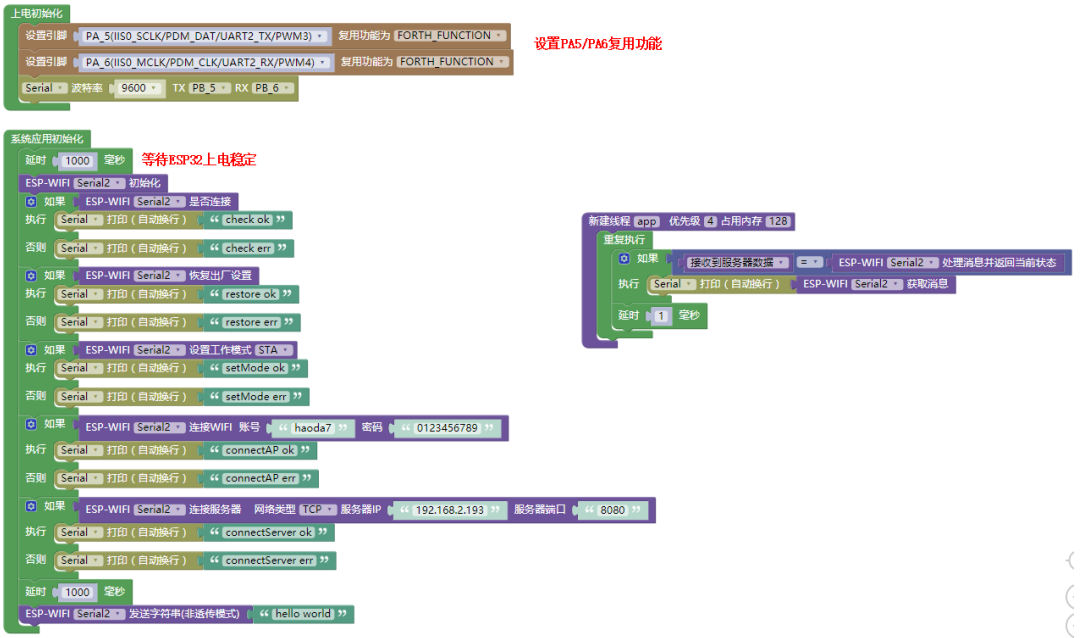


整个过程需要一定时间，根据返回数据可以判断ESP模块正常连接或在哪个步骤没有顺利进行，串口监视器显示如下：



接着我们连接上服务器，向服务器发送消息，并在线程中对接收消息进行程序编写，ESP模块接收服务器的消息的程序如下：

向服务器发送数据和从服务器接收数据总程序如下：



我们可以通过网络调试助手NetAssist进行数据的查看和消息的发送，注意更改协议类型、本地主机地址、本地主机端口即可；上方是服务器接收到的消息，下方是发送消息。

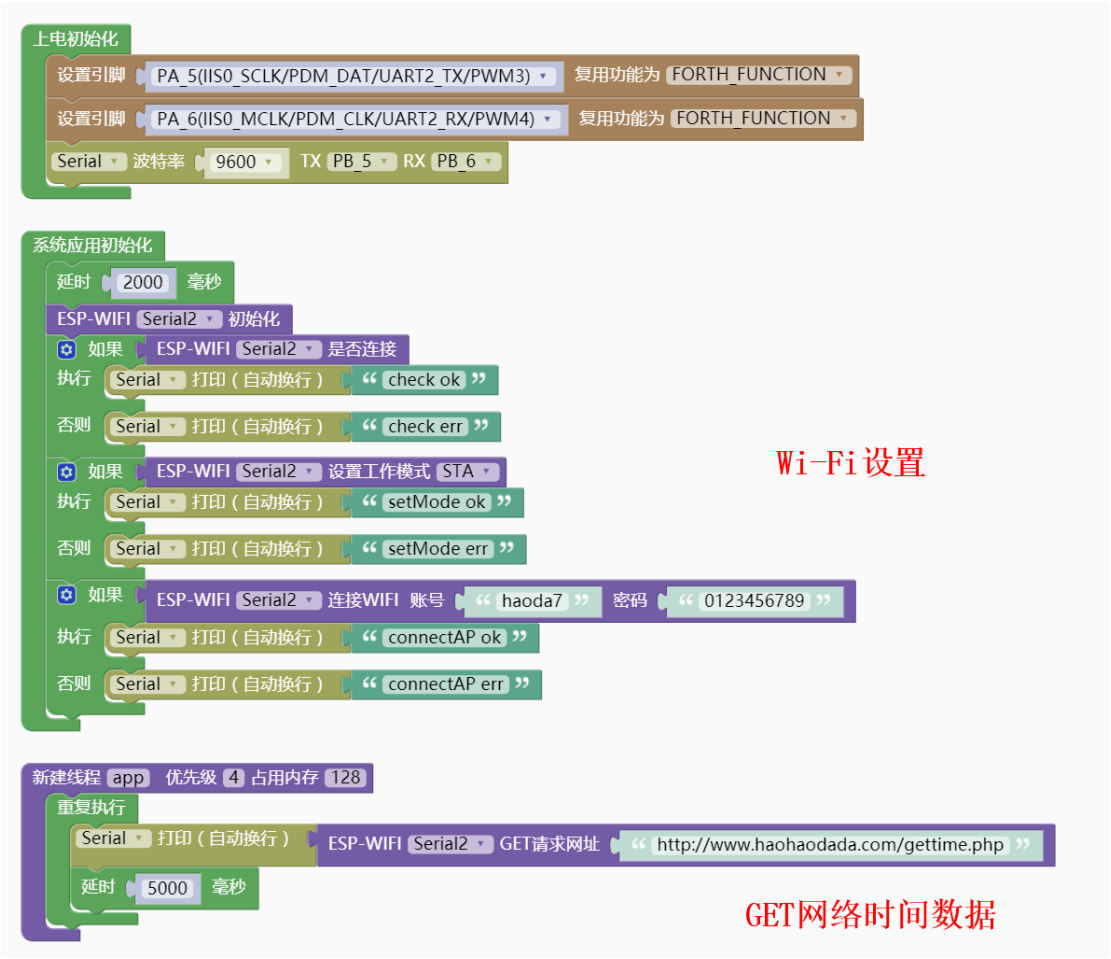
我们也可以修改程序改为UDP通讯，自己实际测试下。

范例3.15 HTTP GET网络时间

一、范例功能

本范例通过学习ESP32扩展库，实现HTTP-GET功能，成功联网获取网络时间并进行处理，达成学习ESP32模块Wi-Fi功能程序编写的目的。

二、范例说明

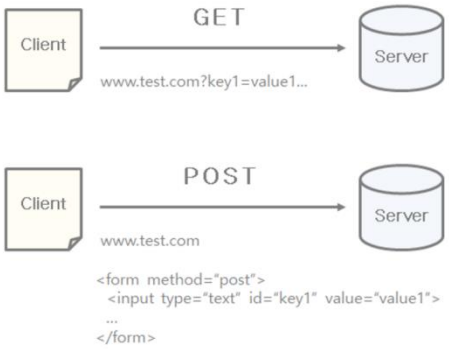


三、知识概念

HTTP是指超文本传输协议，是一个简单的请求/响应协议，通常运行在TCP之上，简单来说是关于数据如何在万维网中如何通信的协议。在客户机与服务器之间进行请求-响应时，两种最常用到的方法是GET和POST。当我们需要从网络上获取数据时，就可以使用这两种方法。

GET和POST本质上就是TCP链接，并无差别。但是由于HTTP的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。 两者最直观的区别就是GET把参数包含在URL中，POST通过request body传递参数，通过表单提交不会显示在url上，隐蔽性更强。

GET是从指定的资源请求数据，而POST是向指定的资源提交被处理的数据。



四、指令学习

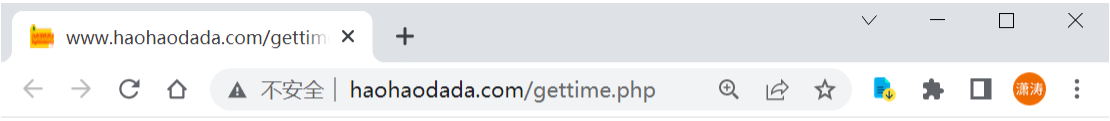
使用HTTP GET方式请求数据。

五、程序详解

本案例主要对GET部分进行学习。

天问已经在自己的服务器上做好了一个网络时间请求接口，我们可以通过浏览器输入API地址查看返回的数据。

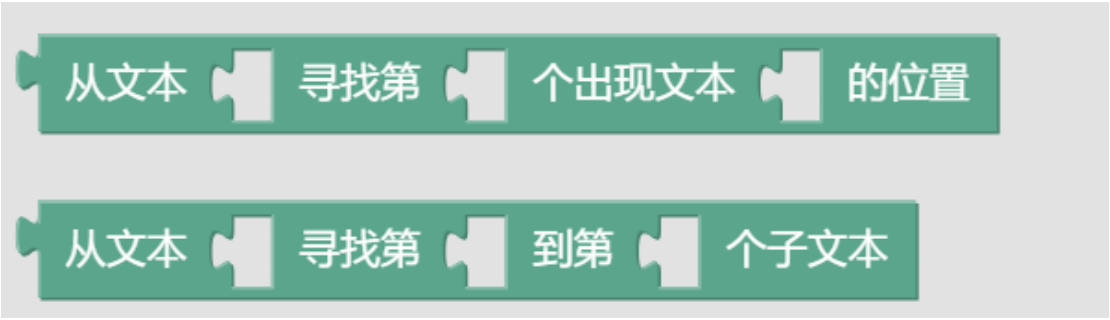
API接口：http://www.haohaodada.com/gettime.php



我可以发现返回的内容是年月日时分秒组成的一段数字。

所以我们的Wi-Fi模块请求这个API接口也能获得对应的时间。

后续我们可以通过如下文本模块



来处理提取相应的年、月、日、时、分、秒，更多的处理方式可以参考出厂程序范例程序。

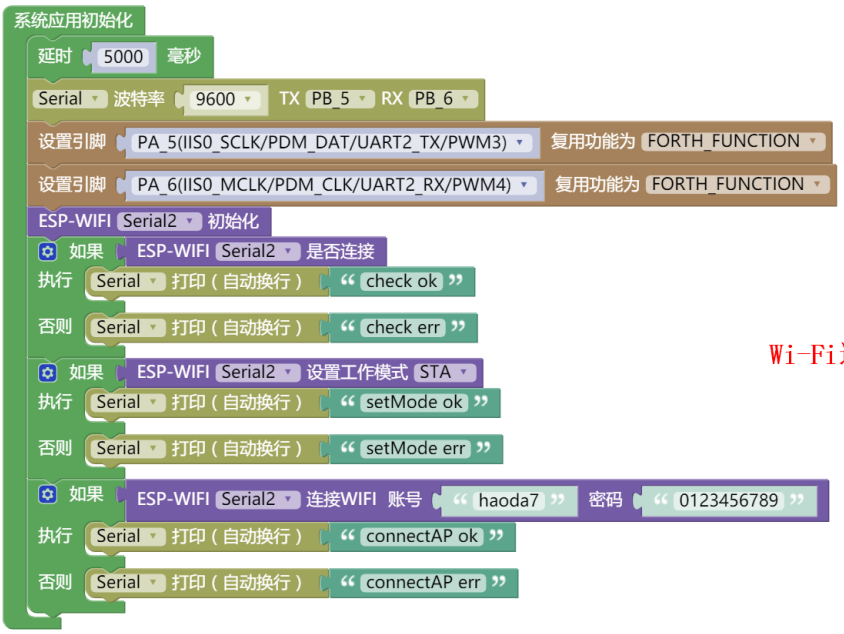
范例3.16 HTTP GET网络天气

一、范例功能

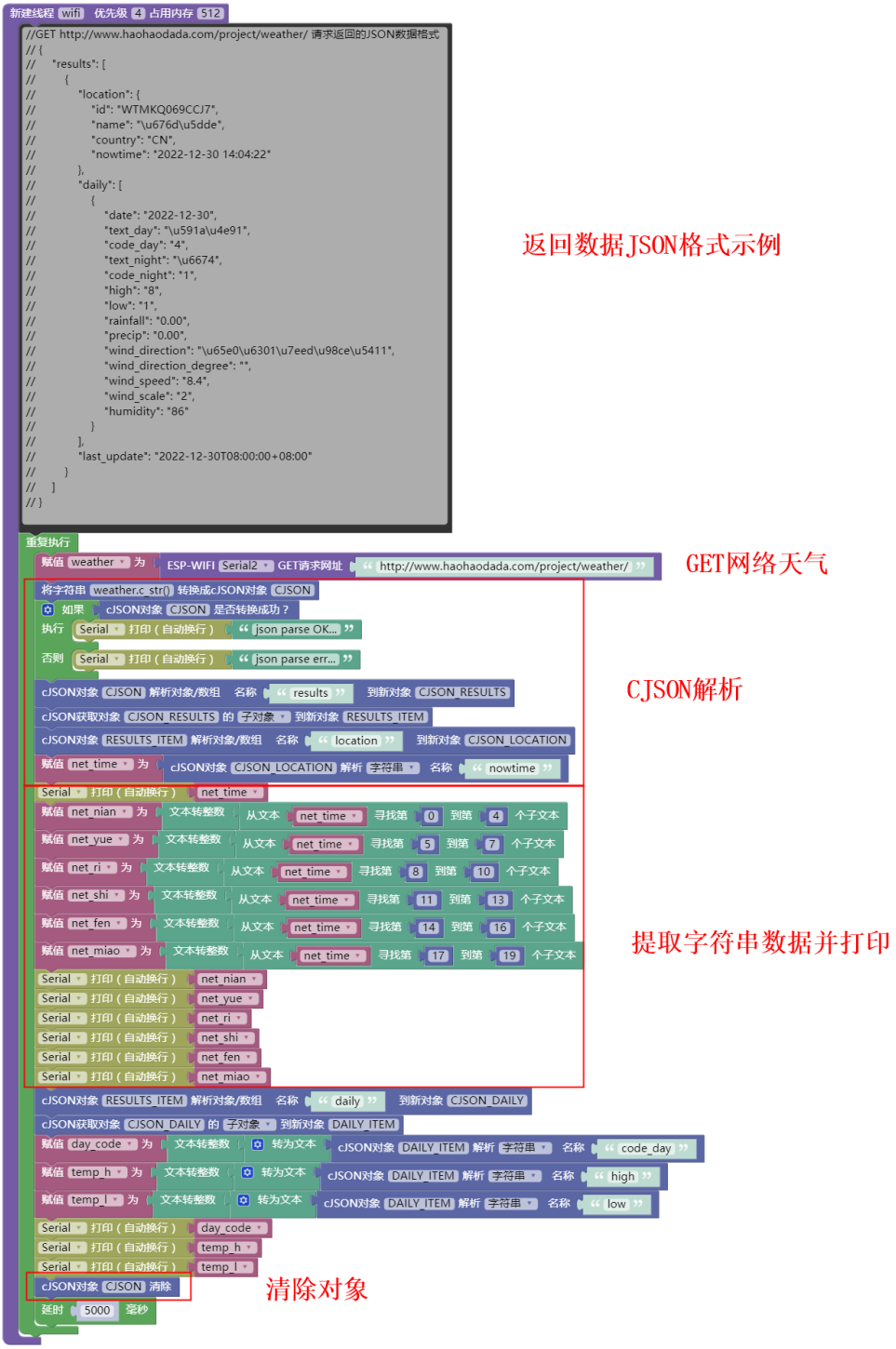
上一个范例我们已经初步了解了GET请求方式，但是网络上大部分返回的数据格式都是JSON，本案例我们通过GET网络天气，掌握cJSON库对数据进行解析。

二、范例说明

变量初始化和cJSON初始化



Wi-Fi连接



三、知识概念

JSON —— 轻量级的数据格式

JSON 全称 JavaScript Object Notation，即 JS对象简谱，是一种轻量级的数据格式。

它采用完全独立于编程语言的文本格式来存储和表示数据，语法简洁、层次结构清晰，易于人阅读和编写，同时也易于机器解析和生成，有效的提升了网络传输效率。

JSON语法规则
JSON对象是一个无序的"名称/值"键值对的集合：

以"{“开始，以”}"结束，允许嵌套使用；
每个名称和值成对出现，名称和值之间使用":"分隔；
键值对之间用","分隔
在这些字符前后允许存在无意义的空白符；
对于键值，可以有如下值：

一个新的json对象
数组：使用“[”和“]”表示
数字：直接表示，可以是整数，也可以是浮点数
字符串：使用引号"表示
字面值：false、null、true中的一个(必须是小写)
示例如下：

```
//GET http://www.haohaodada.com/project/weather/ 请求返回的 JSON 数据格式
{
    "results": [
        {
            "location": {
                "id": "WTMKQ069CCJ7",
                "name": "\u676d\u5dde",
                "country": "CN",
                "nowtime": "2022-12-30 14:04:22"
            },
            "daily": [
                {
                    "date": "2022-12-30",
                    "text_day": "\u591a\u4e91",
                    "code_day": "4",
                    "text_night": "\u6674",
                    "code_night": "1",
                    "high": "8",
                    "low": "1",
                    "rainfall": "0.00",
                    "precip": "0.00",
                    "wind_direction": "\u65e0\u6301\u7eed\u98ce\u5411",
                    "wind_direction_degree": "",
                    "wind_speed": "8.4",
                    "wind_scale": "2",
                    "humidity": "86"
                }
            ],
```

```
            "last_update": "2022-12-30T08:00:00+08:00"
        }
    ]
}
```

cJSON是一个使用C语言编写的JSON数据解析器，具有超轻便，可移植，单文件的特点，使用MIT开源协议。cJSON内部运用列表和结构体来实现，封装JSON数据的过程，其实就是创建链表和向链表中添加节点的过程，解析的过程其实就是剥离一个一个链表节点(键值对)的过程。

cJSON的设计思想从其数据结构上就能反映出来。

cJSON使用cJSON结构体来表示一个JSON数据，定义在cJSON.h中，源码如下：

```
/* The cJSON structure: */
typedef struct cJSON
{
    /* next/prev allow you to walk array/object chains. Alternatively, use
GetArraySize/GetArrayItem/GetObjectItem */
    struct cJSON *next;
    struct cJSON *prev;
    /* An array or object item will have a child pointer pointing to a chain
of the items in the array/object. */
    struct cJSON *child;

    /* The type of the item, as above. */
    int type;

    /* The item's string, if type==cJSON_String  and type == cJSON_Raw */
    char *valuestring;
    /* writing to valueint is DEPRECATED, use cJSON_SetNumberValue instead
*/
    int valueint;
    /* The item's number, if type==cJSON_Number */
    double valuedouble;

    /* The item's name string, if this item is the child of, or is in the
list of subitems of an object. */
    char *string;
} cJSON;
```

cJSON的设计很巧妙。

首先，它不是将一整段JSON数据抽象出来，而是将其中的一条JSON数据抽象出来，也就是一个键值对，用上面的结构体 strcut cJSON 来表示，其中用来存放值的成员列表如下：

String：用于表示该键值对的名称；

type：用于表示该键值对中值的类型；

valuestring：如果键值类型(type)是字符串，则将该指针指向键值；

valueint：如果键值类型(type)是整数，则将该指针指向键值；

valuedouble：如果键值类型(type)是浮点数，则将该指针指向键值；

其次，一段完整的JSON数据中由很多键值对组成，并且涉及到键值对的查找、删除、添加，所以使用链表来存储整段JSON数据，如上面的代码所示：

next指针：指向下一个键值对

prev指针指向上一个键值对

最后，因为JSON数据支持嵌套，所以一个键值对的值会是一个新的JSON数据对象（一条新的链表），也有可能是一个数组，方便起见，在cJSON中，数组也表示为一个数组对象，用链表存储，所以：

在键值对结构体中，当该键值对的值是一个嵌套的JSON数据或者一个数组时，由child指针指向该条新链表。

cJSON项目托管在Github上，仓库地址如下：

https://github.com/DaveGamble/cJSON

天问官方已经移植库到扩展里了，找到共享库中的cJSON库，点击加载。

加载后我们就可以在扩展类别中，找到cJSON的指令。

四、指令学习

下面我们重点介绍下如何通过cJSON解析数据。

1.cJSON初始化

如果需要对JSON数据解析，需要使用这条指令进行库的初始化，主要对内存进行分配。

2.把字符串转换成cJSON对象

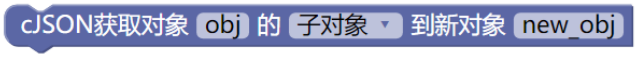
这条指令可以将字符串数据转换成JSON结构体。

3.cJSON解析到新对象


```
(cJSON *) cJSON_GetObjectItem(const cJSON * const object, const char * const string);
```

根据键值对的名称从链表中取出对应的值，返回该键值对(链表节点)的地址。

4.cJSON获取子对象



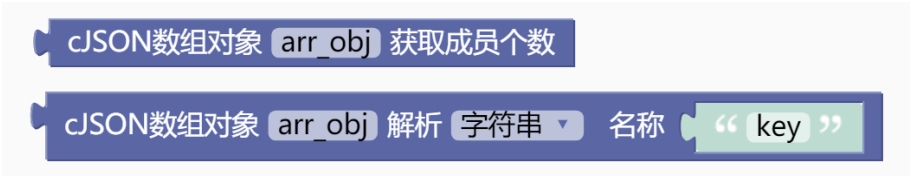

```
cJSON *new_obj=obj->child;
```

配合解析对象模块一起使用来层层剥离嵌套数据出来。

5. cJSON获取对应KEY的数据


```
cJSON_GetObjectItem(obj,"key")->valuestring;
```

根据KEY值类型，选择对应的字符串/整形/双浮点型/逻辑型。如果是数组类型，请使用如下语句来处理；




```
cJSON_GetArraySize(arr_obj)
cJSON_GetArrayItem(arr_obj,"key")->valuestring
```

6. cJSON对象清除


```
cJSON_Delete(obj);
```

内存及时释放，cJSON的所有操作都是基于链表的，所以cJSON在使用过程中大量的使用malloc从堆中分配动态内存的，所以在使用完之后，应当及时清空cJSON指针所指向的内存，该函数也可用于删除某一条数据，该函数删除一条JSON数据时，如果有嵌套，会连带删除。

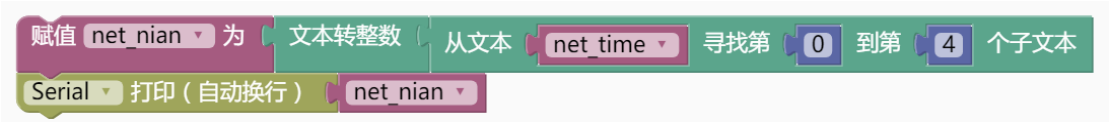
五、程序详解

在网址http://www.haohaodada.com/project/weather/上有实时的时间和天气数据。这里以一段2022-12-30的JSON数据为例，详细说明是如何对数据进行处理获得天气和实时时间的。

```
//GET http://www.haohaodada.com/project/weather/ 请求返回的JSON数据格式
// {
//     "results": [
//         {
//             "location": {
//                 "id": "WTMKQ069CCJ7",
//                 "name": "\u676d\u5dde",
//                 "country": "CN",
//                 "nowtime": "2022-12-30 14:04:22"
//             },
//             "daily": [
//                 {
//                     "date": "2022-12-30",
//                     "text_day": "\u591a\u4e91",
//                     "code_day": "4",
//                     "text_night": "\u6674",
//                     "code_night": "1",
//                     "high": "8",
//                     "low": "1",
//                     "rainfall": "0.00",
//                     "precip": "0.00",
//                     "wind_direction": "\u65e0\u6301\u7eed\u98ce\u5411",
//                     "wind_direction_degree": "",
//                     "wind_speed": "8.4",
//                     "wind_scale": "2",
//                     "humidity": "86"
//                 }
//             ],
//             "last_update": "2022-12-30T08:00:00+08:00"
//         }
//     ]
// }
```

核心代码如下：

1. 通过GET请求获取String类型的字符串数据。
2. weather属于String类型，不是C语言通用的字符串类型，所以先通过weather.c_str()转换，再转换成CJSON结构体对象。
3. 先获取最外面的"results": []对象。
4. 获取"results": []下面的子对象："location"、"daily"、"last_update"。
5. 在获取第二层的" location ": []对象。
6. 根据键值"nowtime"字符串获取对应的数据"2022-12-30 14:04:22"字符串。
7. 通过串口打印字符串数据



再通过文本截取函数获取对应的年月日时分秒数据，为了便于数据处理，上面又把文本类型转为了整数来处理。

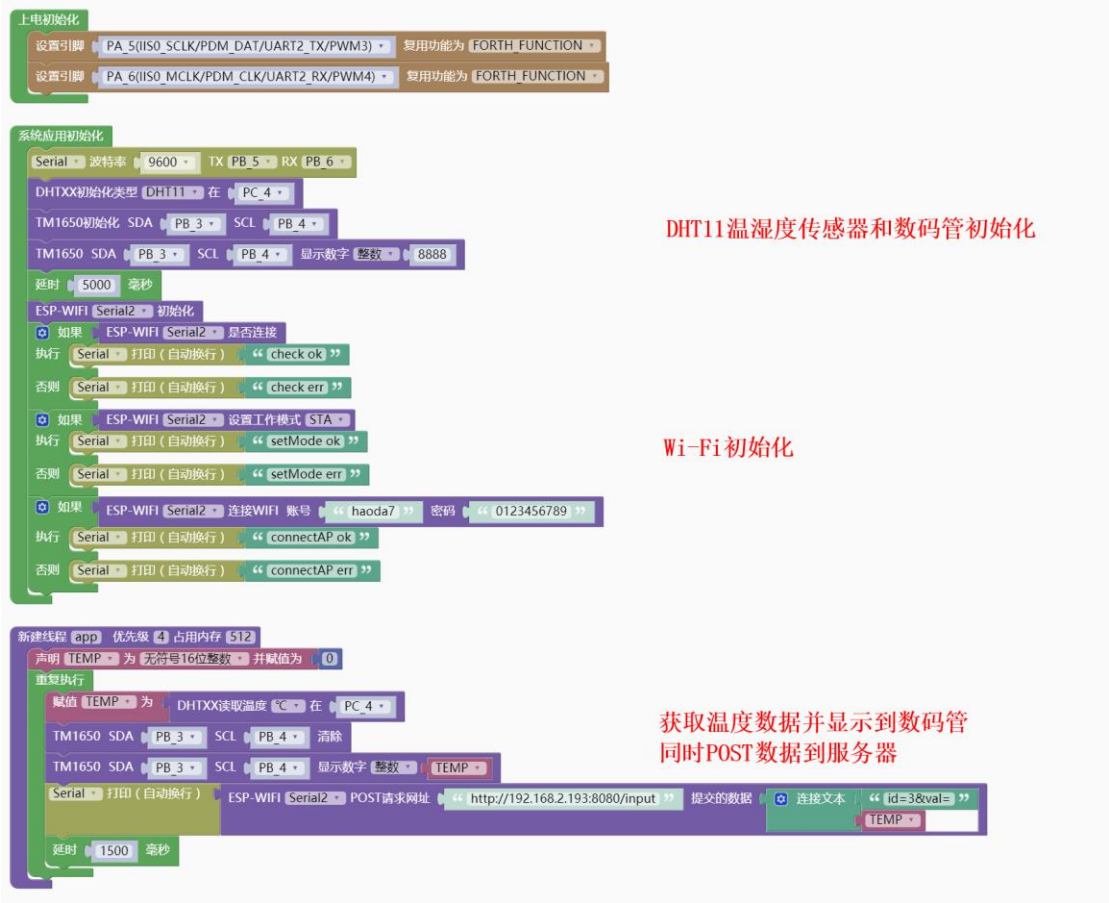
通过串口监视器的打印数据，可以看到对应的数据显示。

范例3.17 HTTP POST温度到服务器

一、范例功能

本范例使用Python搭建基于Flask框架的WEB服务器来显示设备POST过来的温度数据。

二、范例说明



三、指令学习

1.HTTO-POST

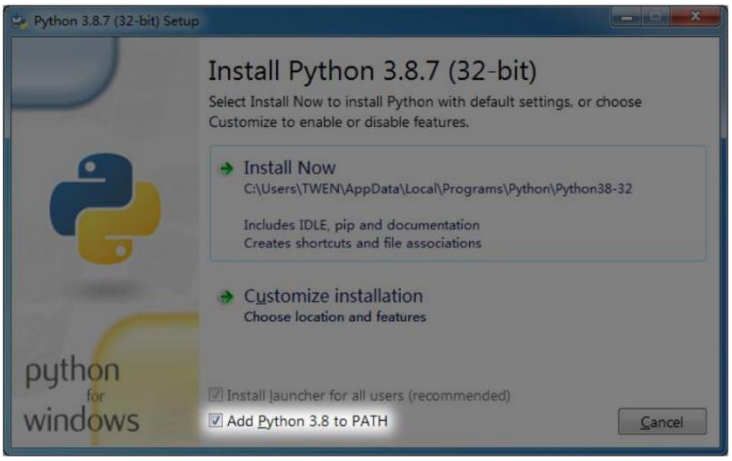
POST的程序写法与GET类似，数据可以一般用name1=value1这样的键值对来表示，如果有多个，用“&”来分隔，服务器会自动根据键值名称提取对应的数据。

四、程序详解

本案例为了方便演示，采用Python来搭建本地WEB服务器。Flask是一个使用 Python 编写的轻量级 Web 应用程序框架。实际项目中还会根据需求采用不同的框架，本文不再展开，可以自行搜索了解。

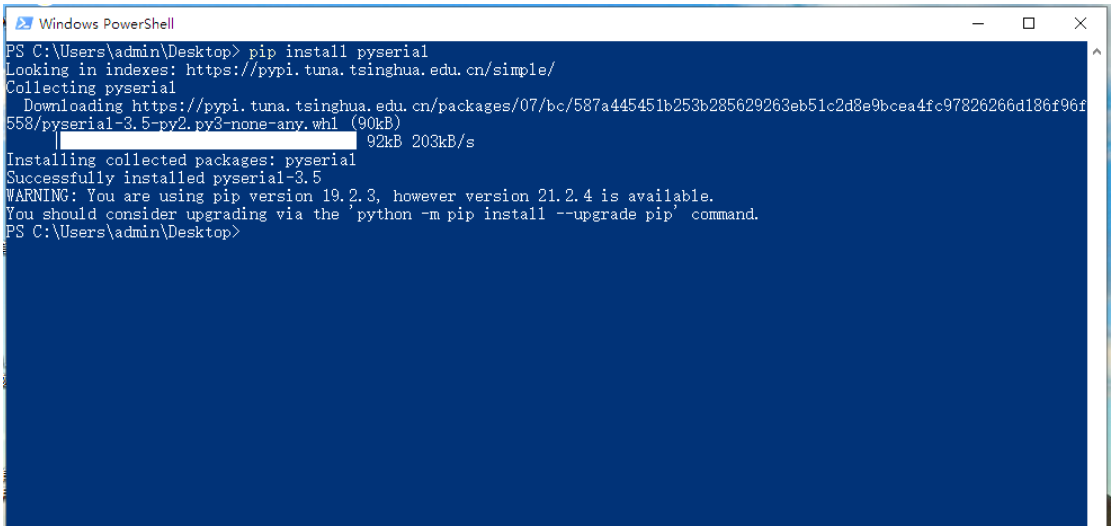
1. 安装Python IDLE，使用3.6以上版本。

Windows 版本安装时在选项中勾选添加 Python 到环境变量，点击下一步直到安装完成。



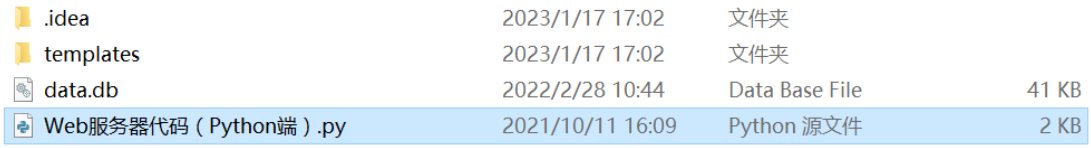
2. 安装Flask模块

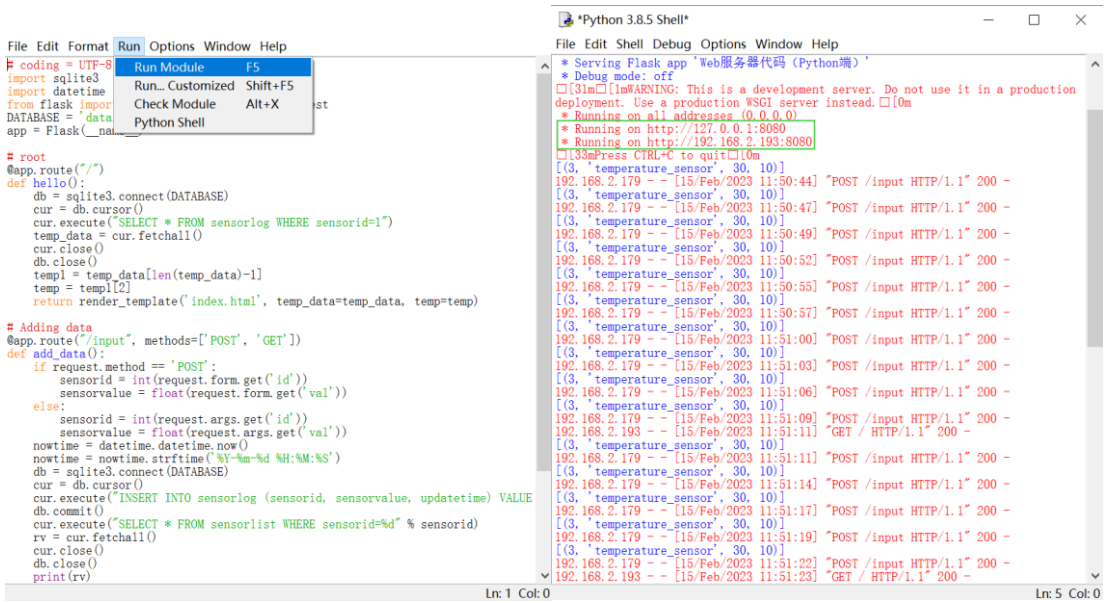
在空白处 Shift+右键，选择“在此处打开 Powershell 窗口，安装Flask模块。在 Windows PowerShell 输入”pip install flask”，按下回车。



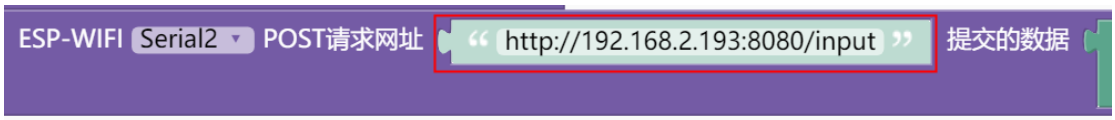
3. 运行WEB服务器

之后使用Python IDLE打开提供的室内环境实时监控系统-Web服务器代码，点击运行。



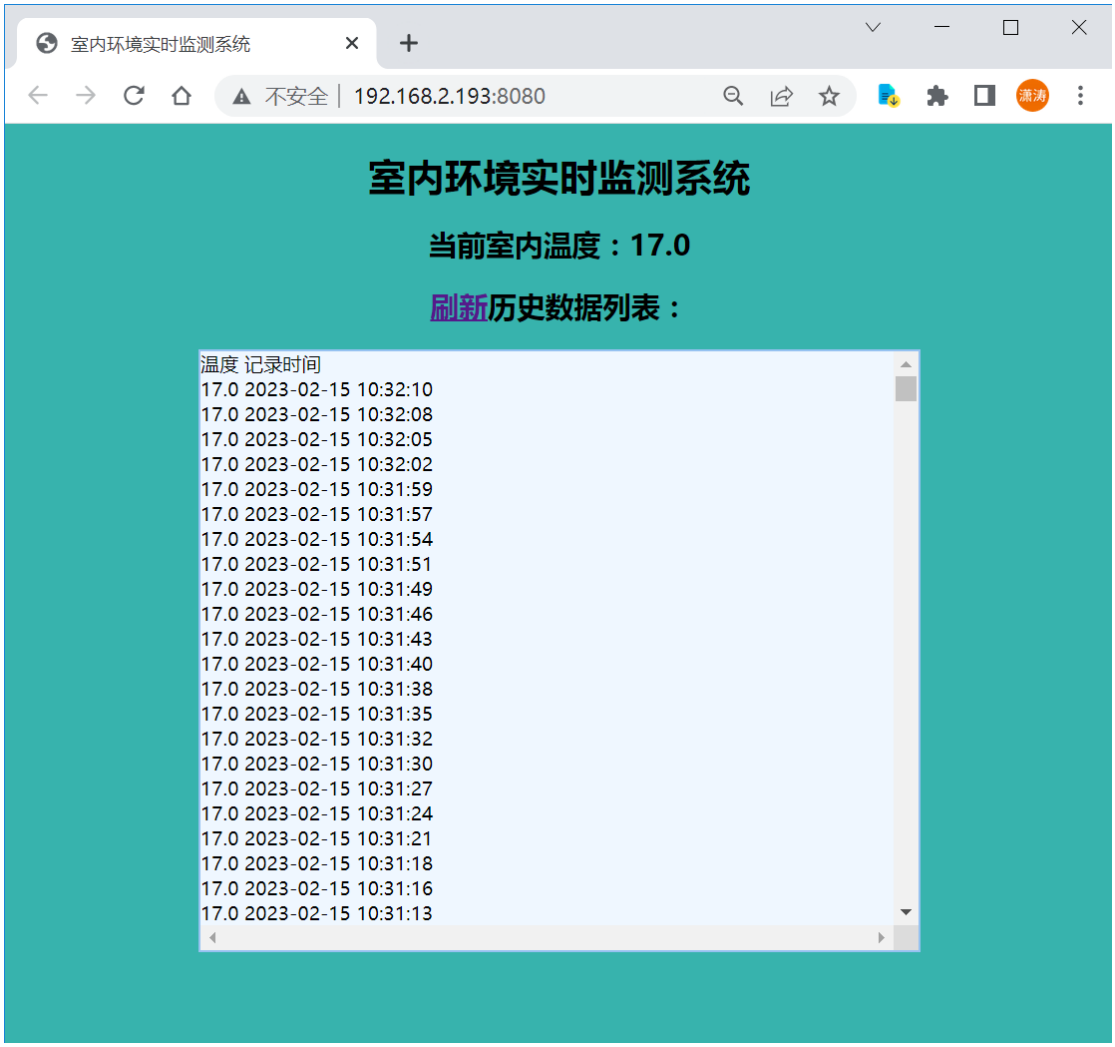


运行后会显示服务器的IP地址和端口号，注意电脑要和设置在同一网段上。笔者电脑上显示的是192.168.2.193:8080（每个人都不一样，请对应修改），在我们的程序中输入对应的IP地址，要根据实际情况修改。



4. 打开浏览器，输入http://192.168.2.193:8080/

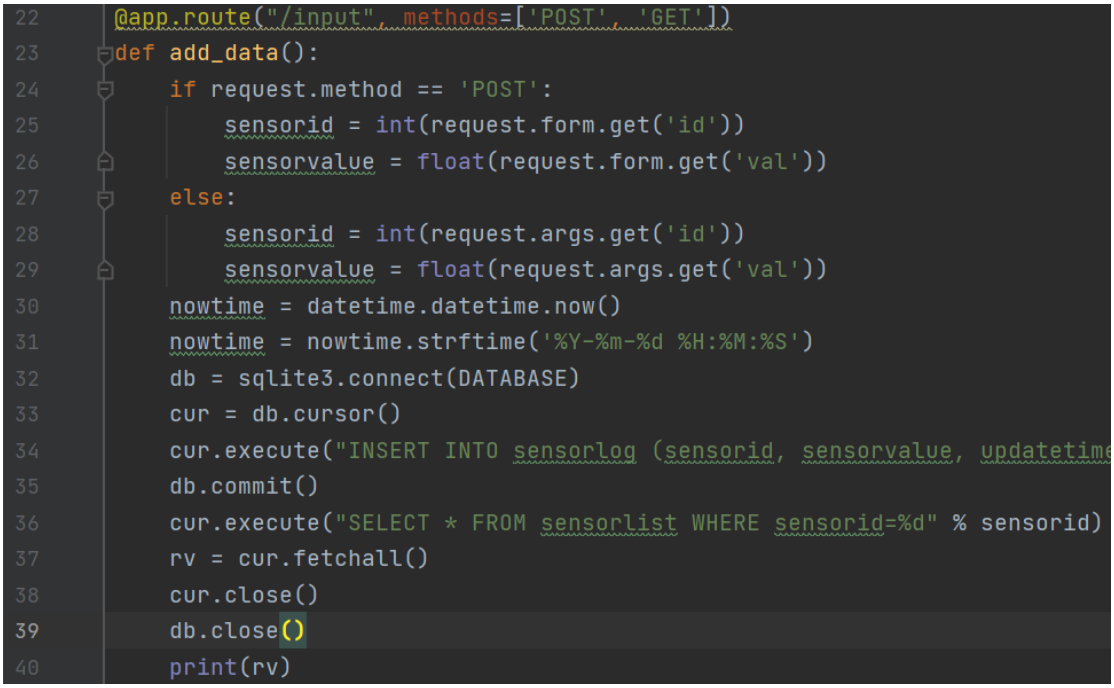
打开网址后我们可以看到以下界面，可以显示POST的温度数据。



WEB服务器关键代码主要有两部分：

```
9   @app.route("/")
10  def hello():
11      db = sqlite3.connect(DATABASE)
12      cur = db.cursor()
13      cur.execute("SELECT * FROM sensorlog WHERE sensorid=1")
14      temp_data = cur.fetchall()
15      cur.close()
16      db.close()
17      temp1 = temp_data[len(temp_data)-1]
18      temp = temp1[2]
19      return render_template('index.html', temp_data=temp_data, temp=temp)
```

第9行就是Flask框架的路由，“/”表示根目录，也就是我们浏览器打开的地址 http://192.168.2.193:8080/，浏览器请求这个地址，服务器就会调用 hello()来执行，下面的代码主要是从本地数据库中读取数据刷新到网页。



```
22  @app.route("/input", methods=['POST', 'GET'])
23  def add_data():
24      if request.method == 'POST':
25          sensorid = int(request.form.get('id'))
26          sensorvalue = float(request.form.get('val'))
27      else:
28          sensorid = int(request.args.get('id'))
29          sensorvalue = float(request.args.get('val'))
30      nowtime = datetime.datetime.now()
31      nowtime = nowtime.strftime('%Y-%m-%d %H:%M:%S')
32      db = sqlite3.connect(DATABASE)
33      cur = db.cursor()
34      cur.execute("INSERT INTO sensorlog (sensorid, sensorvalue, updatetime
35      db.commit()
36      cur.execute("SELECT * FROM sensorlist WHERE sensorid=%d" % sensorid)
37      rv = cur.fetchall()
38      cur.close()
39      db.close()
40      print(rv)
```

第22行，我们看到路径为 /input，也就是我们POST的路径http://192.168.2.193:8080/input，Flask框架会根据POST还是GET方法用对应的数据处理函数来提取数据。余下的代码就是把提取的数据随系统时间一起保存到数据库里，并打印相应的信息。

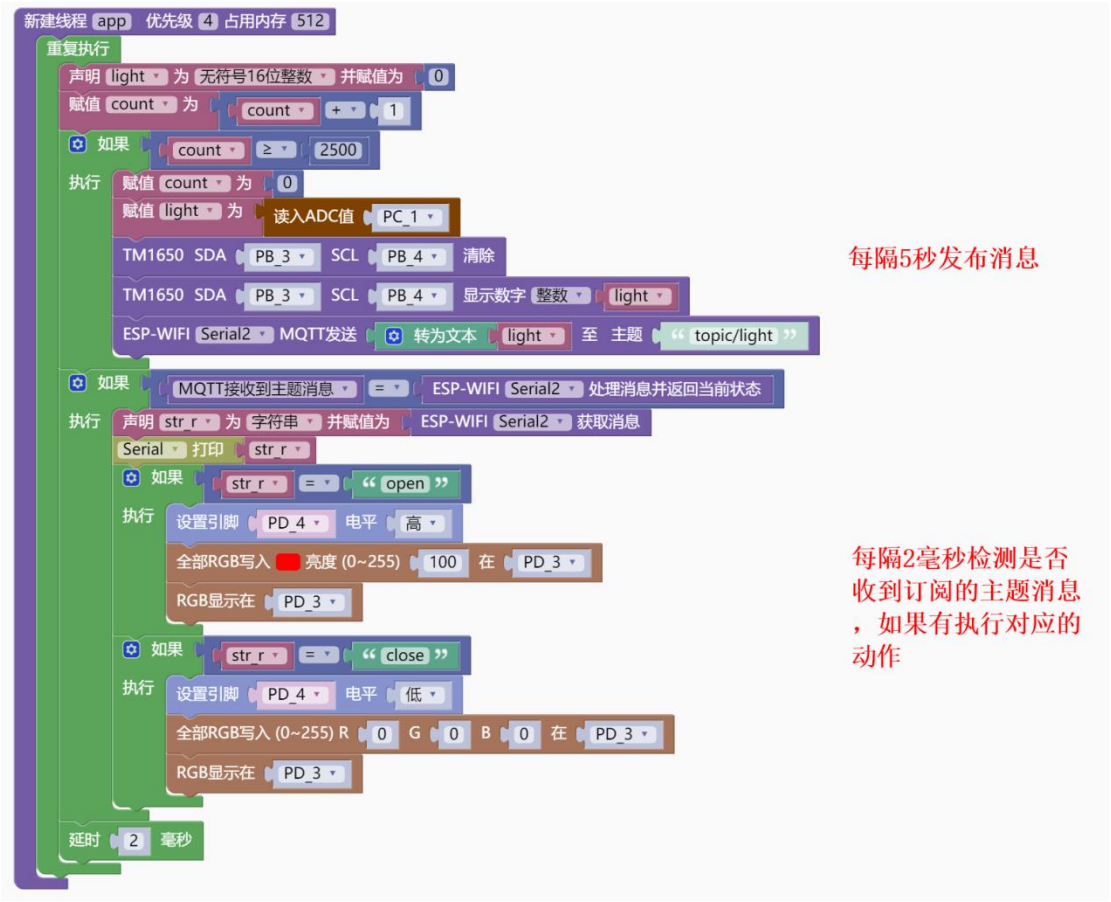
本案例基于本地WEB服务器实现，要实现真正的应用，只需要购买相应的云服务器，设置好Python运行环境，托管上述代码后，配置对应的IP和端口，就可以实现，还可以自己购买相应的域名，做好域名解析到服务器的IP地址。

范例3.18 MQTT

一、范例功能

本范例使用Go搭建的MQTT服务器，来掌握MQTT物联网协议相关知识和使用。

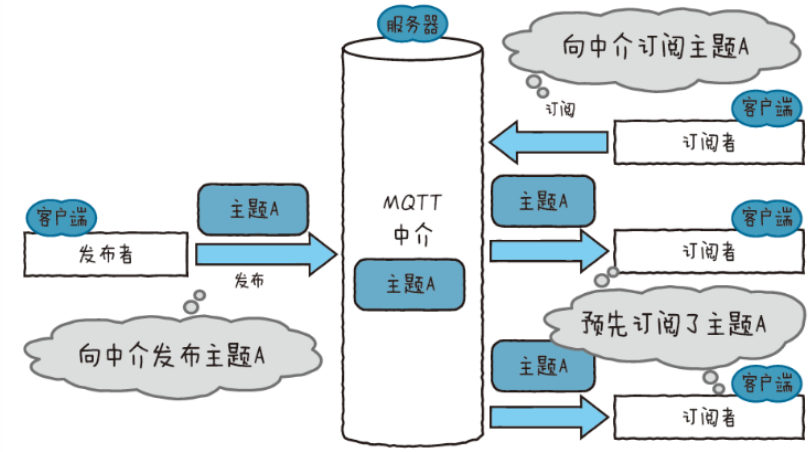
二、范例说明



三、知识概念

MQTT（Message Queue Telemetry Transport）是一种基于客户端服务端架构的发布/订阅模式的消息传输协议。它的设计思想是轻巧、开放、 简单、规范，易于实现。这些特点使得它对很多场景来说都是很好的选择，特别是对于受限的环境（带宽低、网络延迟高、网络通信不稳定），例如物联网环境（IoT）

MQTT是一种基于发布 - 订阅的“轻量级”消息传递协议，用于在TCP / IP协议之上使用，它适用于需要“小代码占用”或网络带宽有限的远程位置的连接，能实现一对多通信（人们称之为发布或订阅型）的协议。它由3 种功能构成，分别是服务器/中介（broker）、发布者（publisher）和订阅者（subscriber）



在MQTT协议通信中，有两个最为重要的角色，它们便是服务端和客户端。

服务端：

MQTT服务端通常是一台服务器（broker），它是MQTT信息传输的枢纽，负责将MQTT客户端发送来的信息传递给MQTT客户端；MQTT服务端还负责管理MQTT客户端，以确保客户端之间的通讯顺畅，保证MQTT信息得以正确接收和准确投递。

客户端：

MQTT客户端可以向服务端发布信息，也可以从服务端收取信息；我们把客户端发送信息的行为称为“发布”信息。而客户端要想从服务端收取信息，则首先要向服务端“订阅”信息。“订阅”信息这一操作很像我们在使用微信时“关注”了某个公众号，当公众号的作者发布新的文章时，微信官方会向关注了该公众号的所有用户发送信息，告诉他们有新文章更新了，以便用户查看。

MQTT主题：

上面我们讲到了，客户端想要从服务器获取信息，首先需要订阅信息，那客户端如何订阅信息呢？这里我们要引入“主题（Topic）”的概念，“主题”在 MQTT 通信中是一个非常重要的概念，客户端发布信息、订阅信息、管理消息都是围绕“主题”来进行的。MQTT交换的消息都附带“主题”地址，各个客户端把这个“主题”视为收信地址，对其执行传输消息的操作。

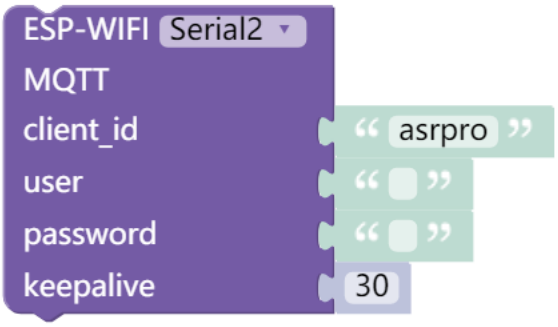
客户端发布消息时需要为消息指定一个“主题”，表示将消息发布到该主题；而对于订阅消息的客户端来说，可通过订阅“主题”来订阅消息，这样当其它客户端或当前客户端向该主题发布消息时，MQTT服务端就会将该主题的信息发送给该主题的订阅者。

MQTT服务器：

为了方便学习，我们用Go语言开发了BIOT本地MQTT物联网服务器，只需要双击运行即可，同时在网页端可以查看数据和历时曲线。最新版天问Block软件已经自带了BIOT软件。

四、指令学习

1.MQTT设置



构建对象，设置MQTT相关参数。

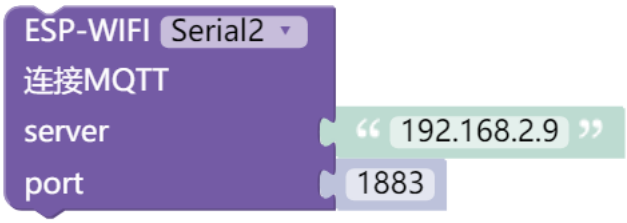
client_id：MQTT客户端的唯一的id，可不填，服务器会自动生成client_id。

user：用于连接MQTT服务器的用户名，根据服务器要求设置，BIOT默认不需要。

Password：用于连接MQTT服务器的密码，根据服务器要求设置，BIOT默认不需要。

keepalive：为保活时间，可以理解为服务器没有收到设备消息传输后设备自动断线的倒计时。如果需要一直连接，设置为0。

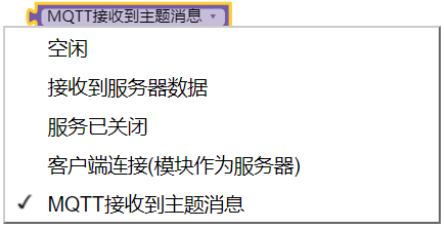
2. MQTT连接



server：MQTT代理服务器的IP地址。

port： MQTT的服务器访问的端口号,一般为1883,不同平台端口会有所不一样。

3. 判断MQTT接收消息



这条指令用于判断MQTT是否接收到topic的消息

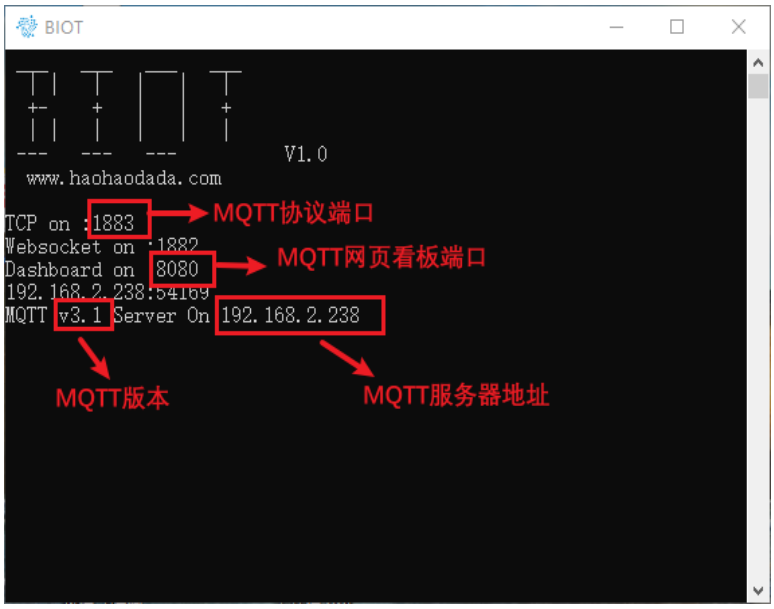
4. **MQTT发送消息**

这条指令用于给主题发布消息，常用MQTT服务器主题会采用“/”符号来管理设备，例如：气象站/温度、气象站/湿度。

五、程序详解

1. 启动BIOT

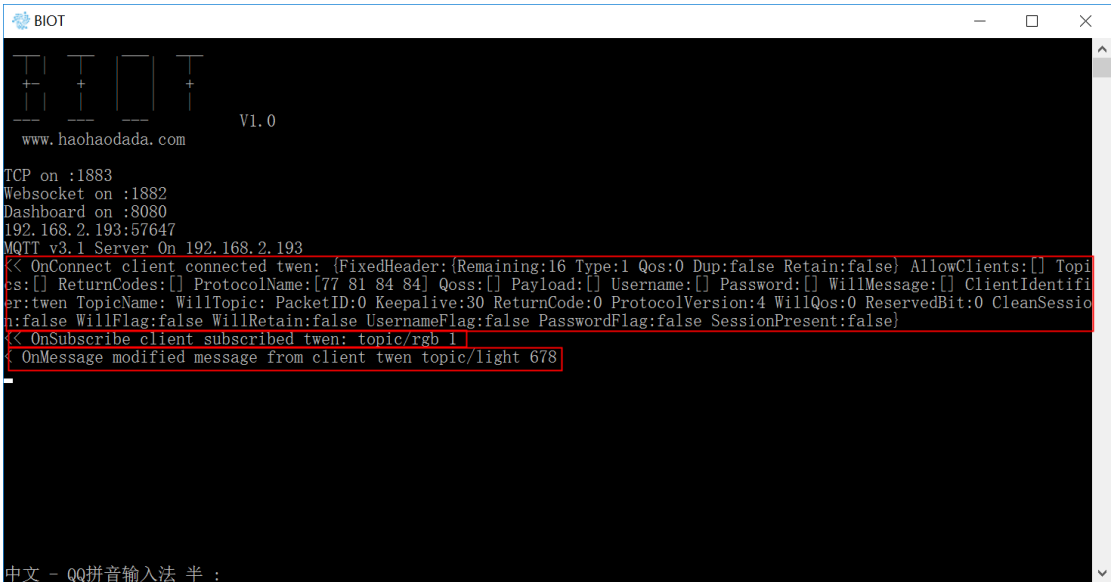
BIOT启动后，一个标准的MQTT服务器就在本地计算机搭建完成了，可以使用MQTT客户端程序与服务器进行通信。其中服务器地址是计算机局域网IP地址，MQTT的端口是1883，8080是网页查看的端口。



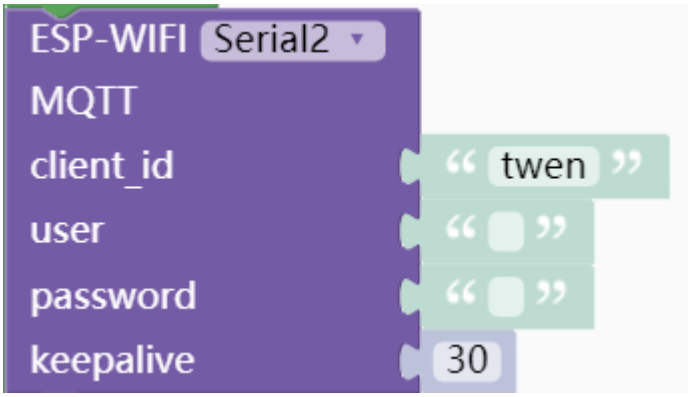
BIOT启动后会自动启动浏览器中打开此地址：http://127.0.0.1:8080

2. 修改程序MQTT服务器地址，后下载程序

3. 查看连接和发送信息



上图中第一个红框里的内容为，客户端连接上来后的配置信息，对应我们下图这些



第二个红框里显示的是订阅信息，对应我们程序里的如下图所示

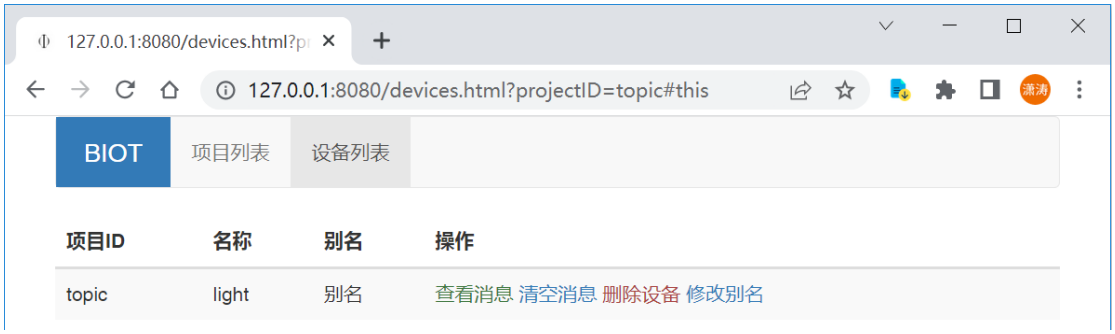
第三个红框里显示的是发布信息，对应我们程序里的如下图所示

5. 查看网页端数据

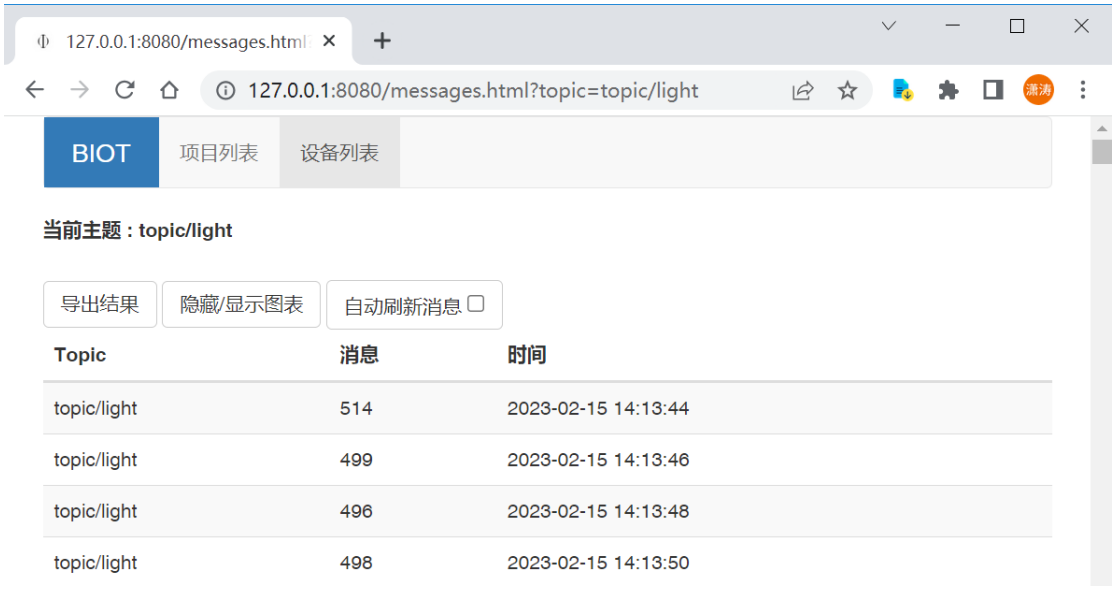
BIOT启动后会自动启动浏览器中打开http://127.0.0.1:8080

当我们向服务器的某个主题和消息时，网页会自动生成该主题。

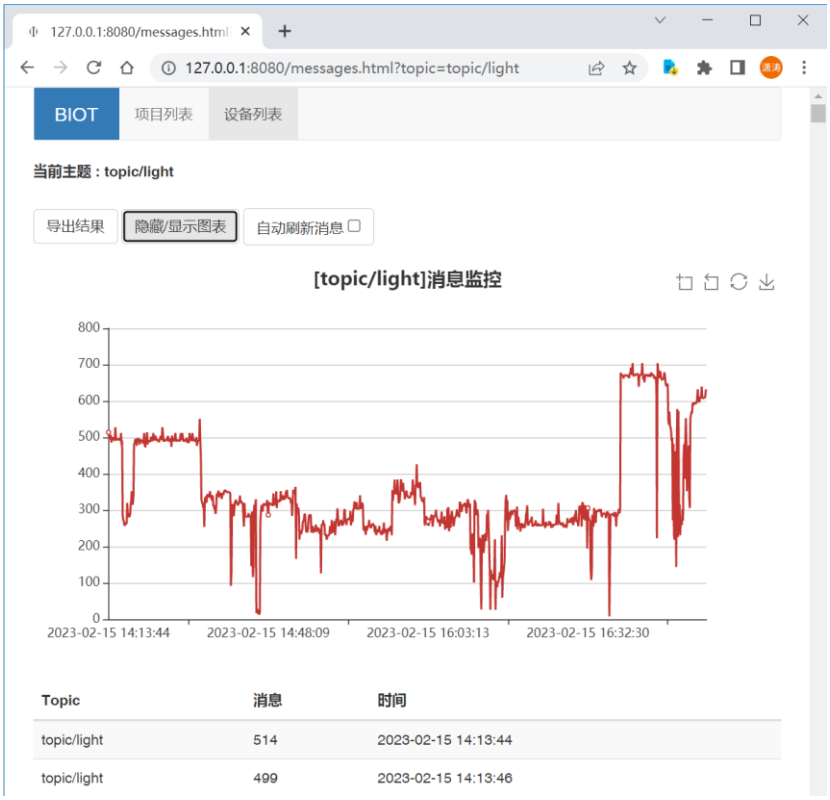
点击查看设备列表，选择查看消息。



打开后即可显示该主题接收到的消息，例如下图，是topic/light接收到的消息。



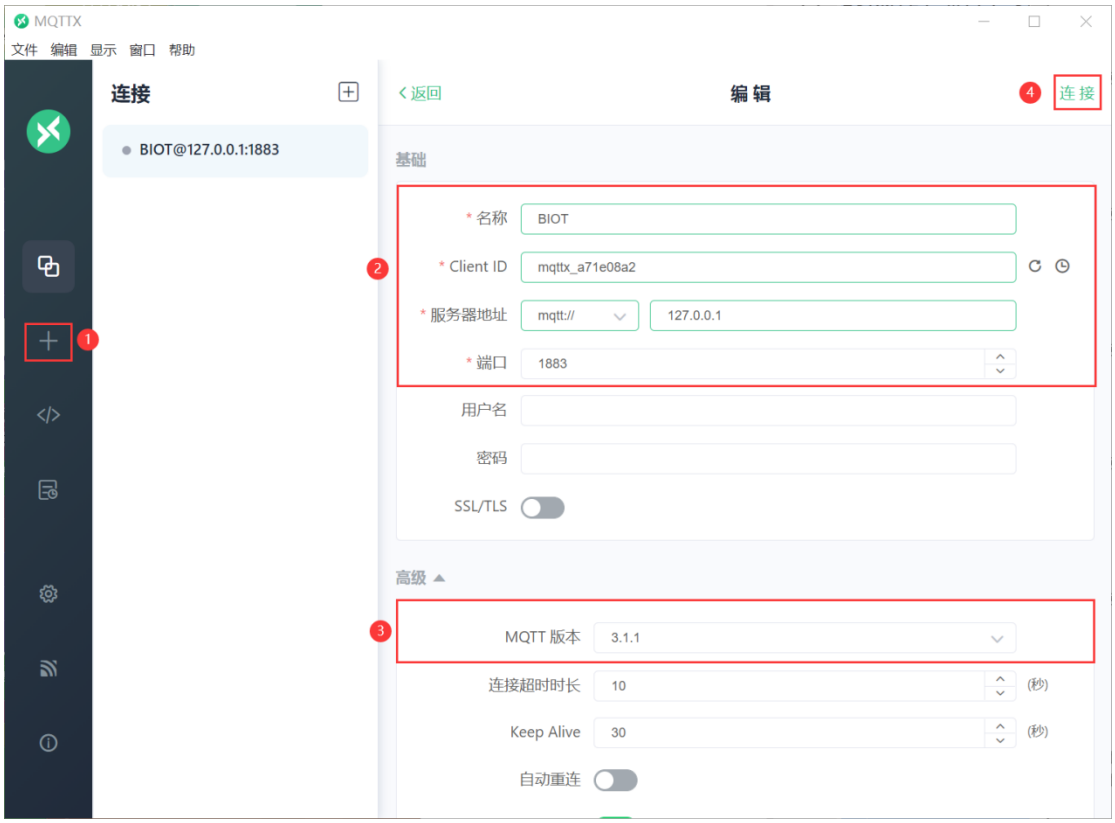
可以选择查看图表并导出结果



6. 通过其他客户端软件互联

我们可以使用软件MQTT X来发送MQTT消息，测试MQTT中的订阅功能。

安装并启动MQTT X客户端后，点击主程序页面左侧菜单栏中的“+”图标。

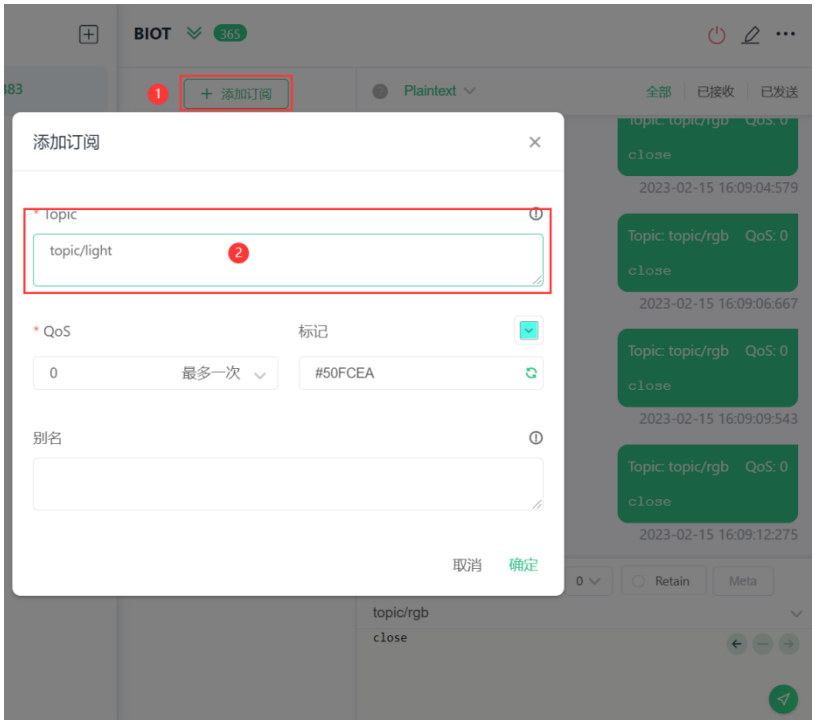


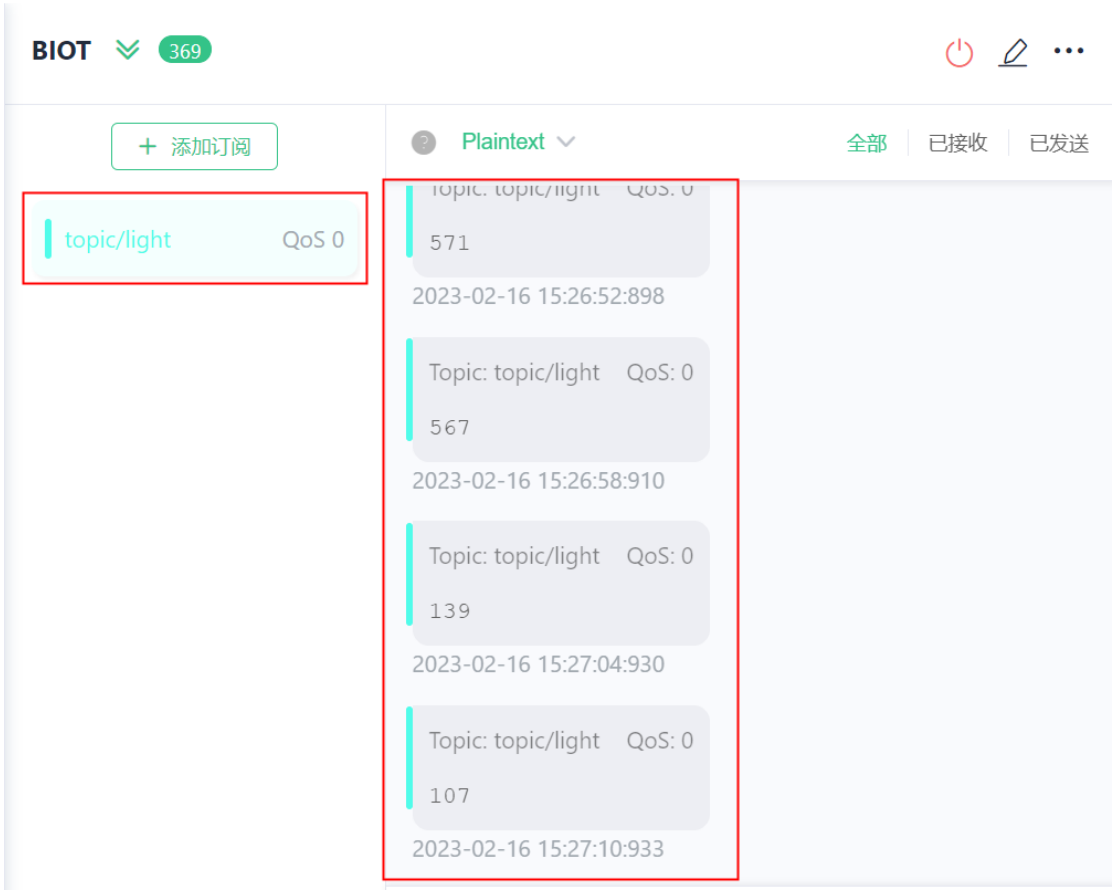
进入到创建页面后，在配置界面设置Name，Client ID，Host, Port, Username, Password等基础配置信息，然后点击MQTT X右上角的“Connect”按钮，完成MQTT客户端和MQTT服务端的连接。可以先点击左下角的设置按钮，选择切换语言。

修改名称，注意Client ID不要与其他设备重复，修改服务器地址，修改MQTT版本为3.1，然后点击连接。

7. 订阅主题

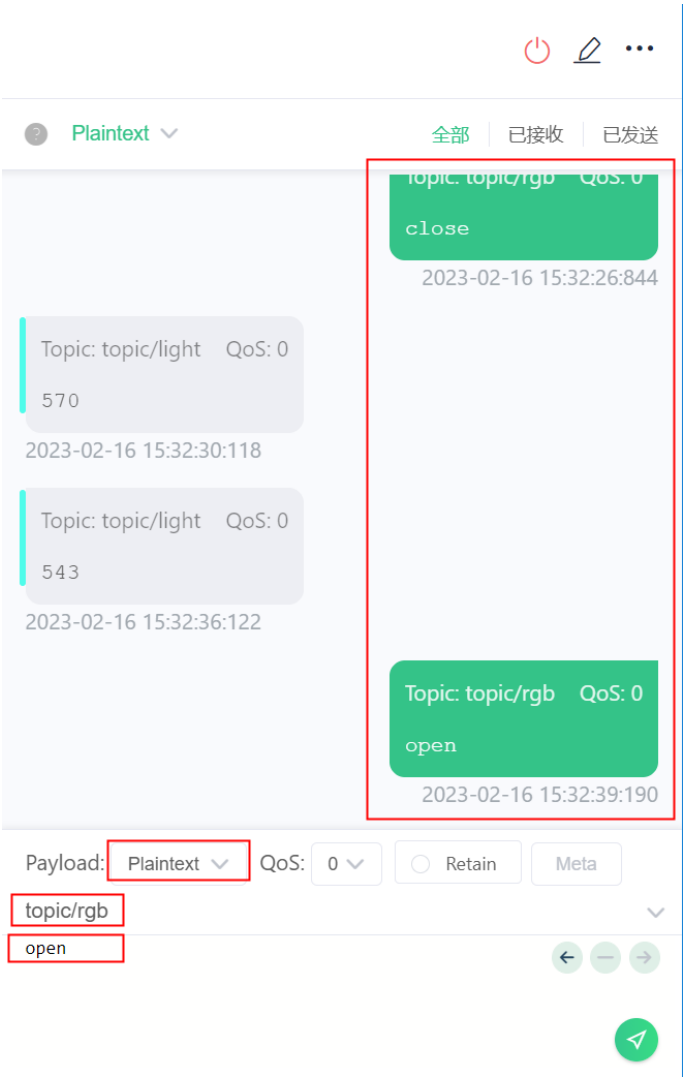
点击添加订阅按钮后，设置好和程序里对应的主题名称后点击确定，就能接收到消息了。





2. **发布主题控制RGB灯和继电器**

可以在连接主页面的下方的输入框内，输入主题(Topic) 和消息体(Payload) 后，点击右下角发送图标按钮，就可以想MQTT服务器发送测试消息了。发送“open”和“close”即可控制板载RGB点亮或熄灭。



范例3.19 BLE蓝牙控制

一、范例功能

本范例通过学习ESP32扩展库BLE部分，了解并学习蓝牙，通过蓝牙实现手机APP控制继电器开关、控制板载RGB、控制彩屏显示文本、湿度值上传等功能，达成学习ESP32模块蓝牙功能程序编写的目的。

二、范例说明

三、知识概念

蓝牙起源：

Bluetooth®

蓝牙（Bluetooth）是一种短距离、低功耗的无线通信技术。该技术最初于1994年由爱立信公司提出，当时是作为RS232的替代方案。作为有线传输的无线替代方案，其理念是使用无线电传输（即无线传输）来交换数据。

蓝牙技术目前由蓝牙技术联盟（Bluetooth Special Interest Group，缩写为 SIG）负责维护其技术标准，联盟成员已超过三万，分布在电信、电脑、网络与消费性电子产品等领域。IEEE曾经将蓝牙技术标准化为IEEE 802.15.1，但是这个标准已经不再继续使用。

蓝牙这个名字来自十世纪的一位丹麦国王哈拉尔德·戈姆森（丹麦语：Harald Blåtand Gormsen），因为Blåtand翻译成英语是Bluetooth，所以哈拉尔国王被称为“蓝牙王”。后来，蓝牙王统一了四分五裂的交战派，即现在的挪威、瑞典和丹麦。同样地，蓝牙技术作为一种开放式标准，目的是让离散的产品和行业可以建立无线连接和协同工作，因此采用了Bluetooth作为该技术的代号。

蓝牙版本：

蓝牙自从4.0版本支持两种无线技术，一是蓝牙基本速率/增强数据速率，通常称为经典蓝牙（BR/EDR），二就是低功耗蓝牙，也就是我们俗称的BLE（Bluetooth Low Energy ）。

低功耗蓝牙协议的创建旨在于一次传输非常小的数据包，因此与经典蓝牙相比功耗大大下降。

可支持经典蓝牙和低功耗蓝牙的设备称之为双模设备，如移动手机。仅支持低功耗蓝牙的设备称之为单模设备。这些设备主要用于低功耗的应用，如使用纽扣电池供电的应用。

ESP32-C3属于单模BLE低功耗蓝牙4.2版本。

蓝牙版本	发布时间	最大传输速度	传输距离
蓝牙1.0	1998	723.1 Kbit/s	10米
蓝牙1.1	2002	810 Kbit/s	10米
蓝牙1.2	2003	1 Mbit/s	10米
蓝牙2.0+EDR	2004	2.1 Mbit/s	10米
蓝牙2.1+EDR	2007	3 Mbit/s	10米
蓝牙3.0+HS	2009	24 Mbit/s	10米
蓝牙4.0	2010	24 Mbit/s	50米
蓝牙4.1	2013	24 Mbit/s	50米
蓝牙4.2	2014	24 Mbit/s	50米
蓝牙5.0	2016	48 Mbit/s	300米
蓝牙5.1	2019	48 Mbit/s	300米
蓝牙5.2	2020	48 Mbit/s	300米

服务端与客户端：

蓝牙客户端（也叫主机/中心设备/Central）：

客户端扫描附近的设备，当它找到它正在寻找的服务器时，它会建立连接并监听传入的数据。比如手机一般作为客户端。

蓝牙服务端（也叫从机/外围设备/peripheral）：

服务器宣传它的存在，因此它可以被其他设备发现并包含客户端可以读取的数据。比如蓝牙手环，又例如本案例中ASRPRO Plus的蓝牙模块。

广播：

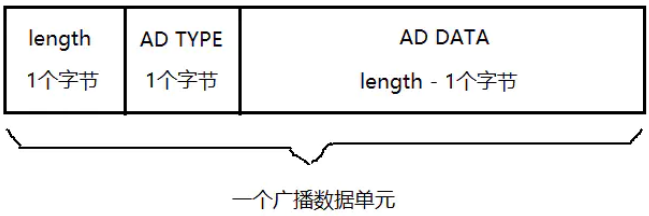
广播的目的就是让别人能够发现自己，也就是发现蓝牙，从而建立连接。

广播包中包含若干个广播数据单元，广播数据单元也称为AD Structure。

以程序中的指令 

为例

上面这条指令的广播数据是020106090948616F64615F4254



LE	TYPE	VALUE
0x02	0x01	0x06
0x09	0x09	48616F64615F4254

第一个广播数据单元，LE代表长度是2，包含了0106，其中Type=0x01代表设备LE物理连接，06是value值，基本固定。

第二个广播数据单元，LE代表长度是9，Type=0x09代表设备的全名称，我们将48616F64615F4254转换为ASCII码，为Haoda_BT，这个名称也就是我们后续可以搜索到的蓝牙名称。

举例，如果设置蓝牙名称为ASRPRO，那么查看ASCII码，最后六个字节是41535250524F，整个广播数据为0201060709415352505 24F。如果长度超出10，那么10个字节用0x0A，11个用0x0B，以此类推。

服务序号、特征序号和UUID：

我们将从机具有的数据或者属性特征叫做Profile，而Profile包含一个或者多个服务序号Service，一个Service包含多个特征序号Characteristic，一个特征包含多个描述符。

而Service、Characteristic、Descriptor，这三部分都由UUID作为唯一标示符。

如果使用ESP-C3-12F官方提供的默认固件，它的服务序号、特征值以及对应的UUID都是相同的。

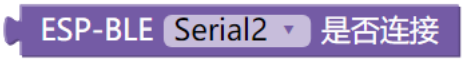
四、指令学习

我们在学习蓝牙的指令前，需要先添加扩展库。点击添加扩展，找到官方扩展库中的ESP-BLE，版本选择最新，点击加载；

1.ESP模块初始化

主要初始化对应的串口。

2.ESP-BLE是否已连接

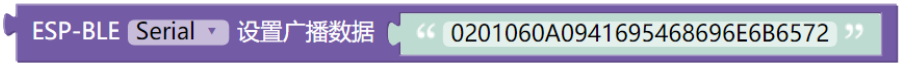


这条指令是判断蓝牙模块硬件是否已连接，而不是蓝牙是否成功连接，注意区分。

3.蓝牙设备名称设置

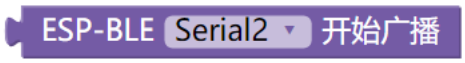
这条指令可以设置蓝牙设备的名称，注意这个名称是服务内的名称，而不是搜索列表中的名称，两者不要混淆。

4.广播数据设置



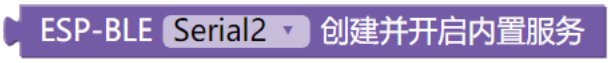
建立蓝牙连接的初始设置之一，名称会显示在手机搜索列表中，配合开始广播一起使用。

2. 开始广播



建立蓝牙连接的初始设置之一，开始广播，确保蓝牙设备能够被搜索到。

6.内置服务开启



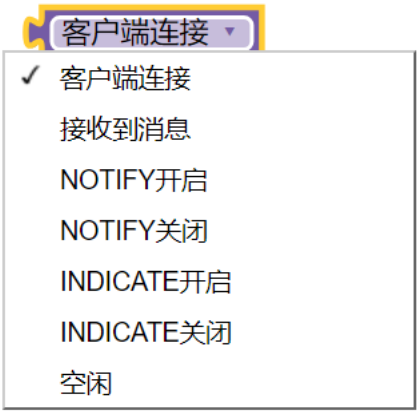
建立蓝牙连接的初始设置之一

7.蓝牙数据发送

notify与indicate的区别在于,indicate需要有回复才能发送下一个数据包。

在本案例中发送温度值使用的是notify。本案例使用的蓝牙特征属性和notify和write属性可以详见参考资料ESPBLE-APP INVENTOR蓝牙控制。

8.客户端连接/消息接收判断



这条指令可以判断蓝牙客户端是否连接/蓝牙是否接收到消息等。

9.服务序号和特征序号

我们在知识概念中已经介绍过服务序号和特征序号，本案例中，我们在发送数据后，串口会打印，打印具体内容如下，服务序号为1，特征序号为3。

五、程序详解

GPIO设置、语音设置、彩屏初始化设置等这里不再赘述。本部分主要讲解蓝牙部分和APP inventor部分。

蓝牙连接：

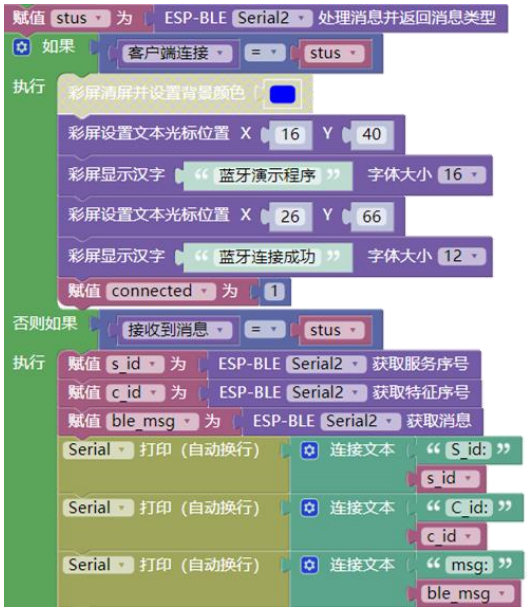
蓝牙设备连接，显示在搜索列表的名称为Haoda_BT。

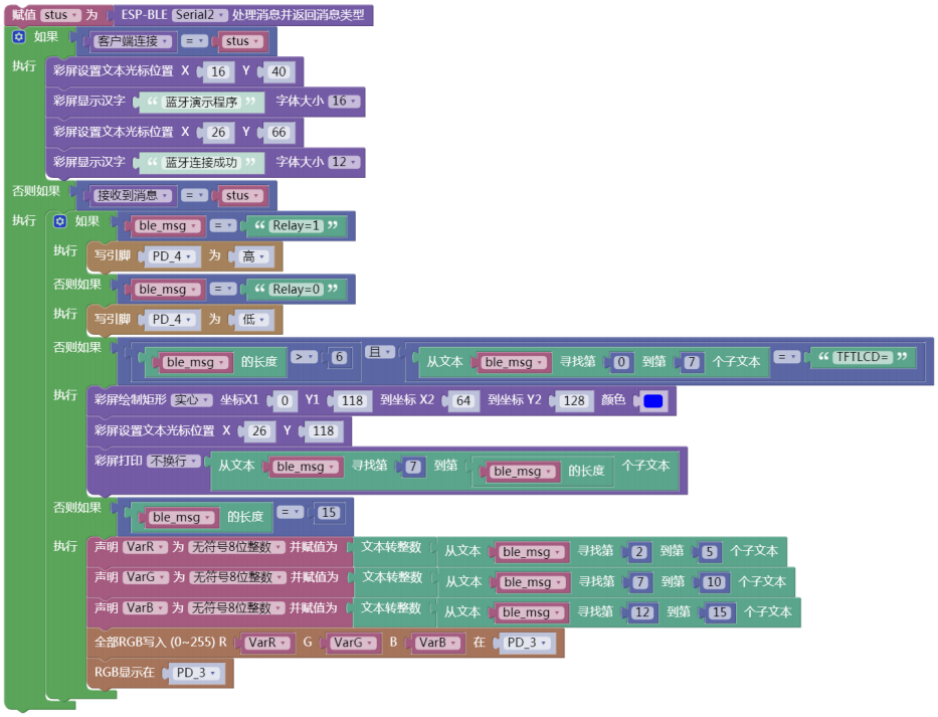
蓝牙数据发送：

每隔5s发送一次湿度值，并在数码管上显示温度值。如果显示数值为0，重启5s后开机。

蓝牙数据接收：

蓝牙数据接收包括三个部分：判断ESP模块是否接收到消息、判断客户端是否已连接、判断是否接收到蓝牙消息。接着对接收的消息进行判断。下方程序包括了通过对接收的数据进行处理，开关继电器、彩屏显示文本和RGB显示。





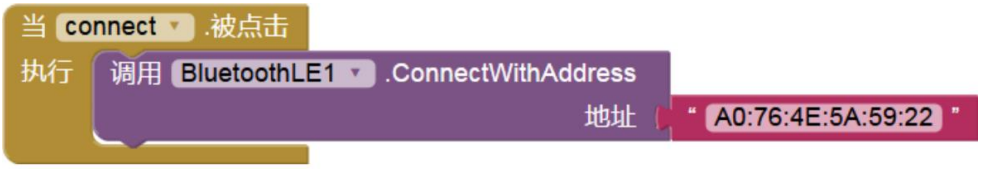
APP inventor：

APP inventor包含组件设计与逻辑设计，组件设计包含可视组件与非可视组件，逻辑设计使用图形块编写功能。推荐使用广州电教馆版本：http://app.gzjkw.net

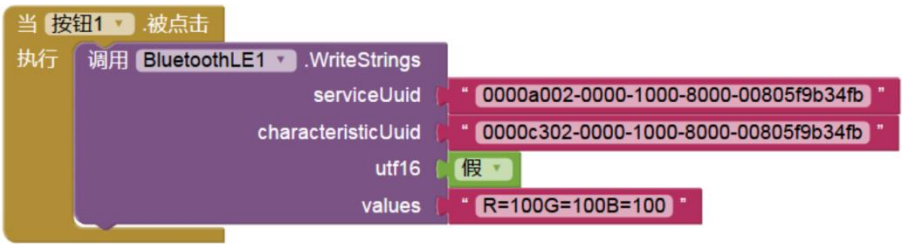
App inventor的正式版是不带BLE功能的，只有基础的蓝牙功能，需要使用BLE就要自己下载aix包后导入。

使用BLE需要把它拖进屏幕内，这样就能在逻辑面板使用BLE的各个模块了。接着拖两个按钮和一个文本输入框到屏幕内。BLE蓝牙直接按MAC地址连接，如需选择设备请查看完整项目，借鉴其中的代码。

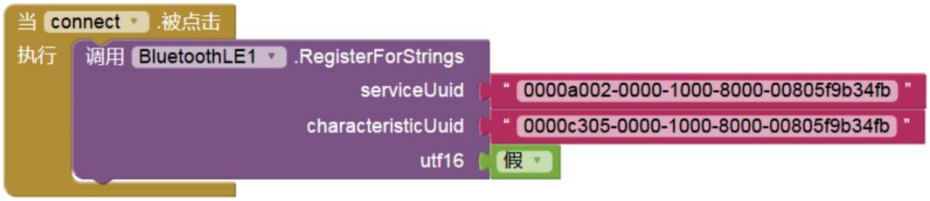
BLE蓝牙连接：



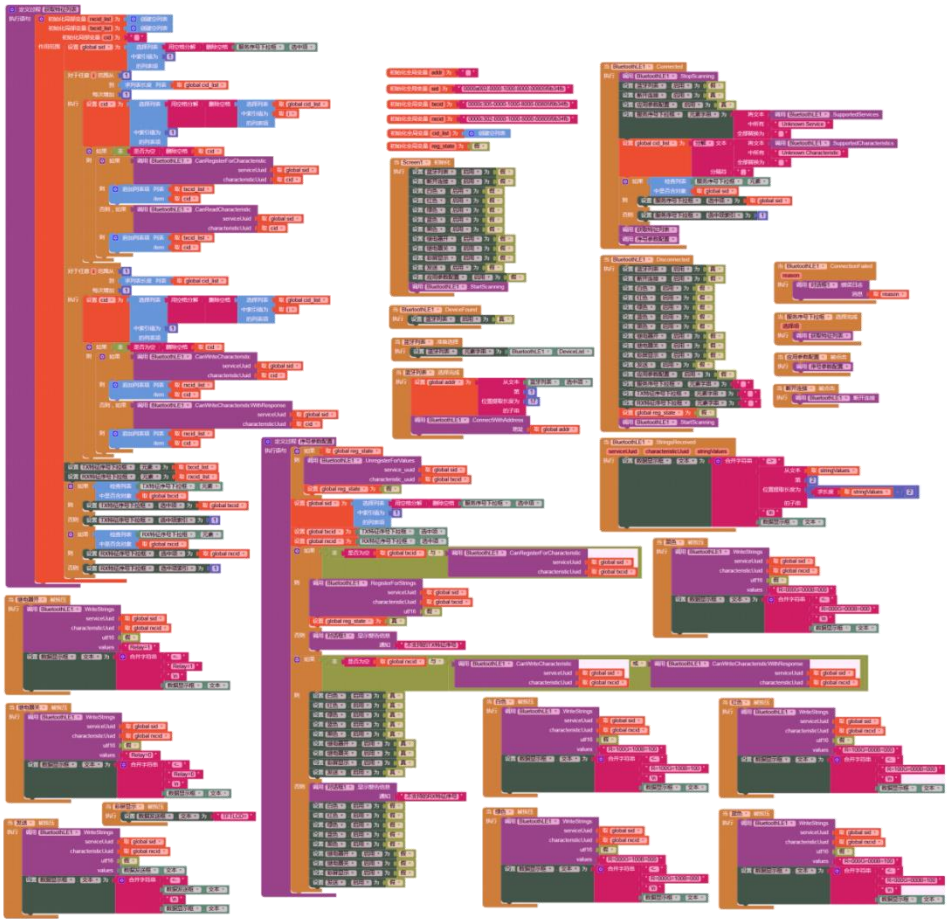
发送数据：



接收数据：



以上部分为蓝牙BLE基础功能的实现，完整项目的逻辑设计图和组件设计图如下所示。下载参考附件中ESPBLE.apk，如果导入则是.aia文件。



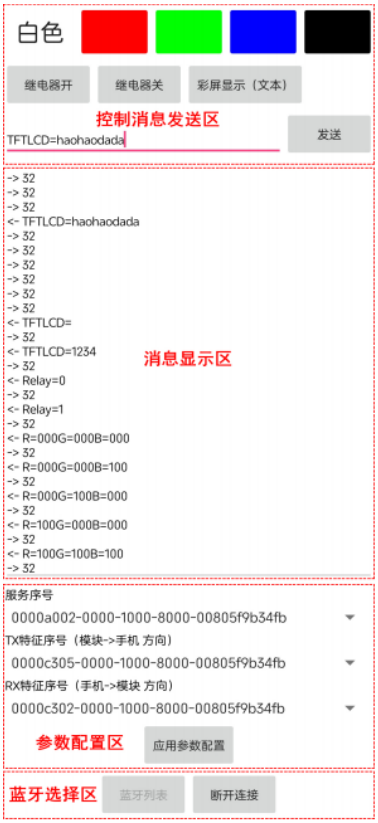
逻辑设计

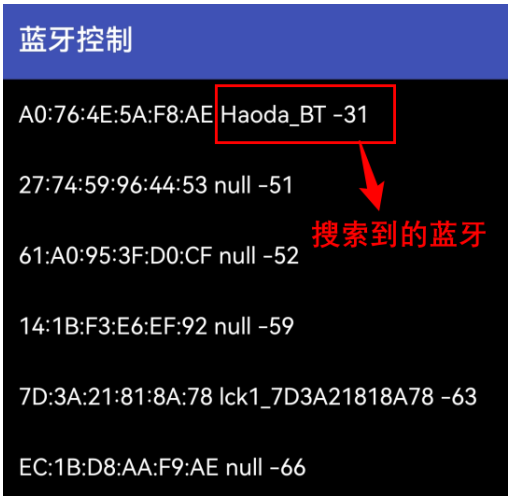
实际使用：

打开安装好的APP，会提示蓝牙控制想要开启蓝牙，点击允许。

开启蓝牙后，灰色的蓝牙列表会变成黑色。（如果没有直接重启app）

接着打开ASRPRO Plus，运行程序，等待一段时间，点击蓝牙列表，选择Haoda_BT。



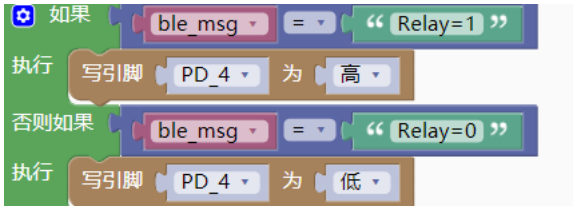


查看上方界面，整个蓝牙界面包括消息发送区、消息显示区、参数配置区和蓝牙选择区。

我们可以通过蓝牙对各种设备进行控制。

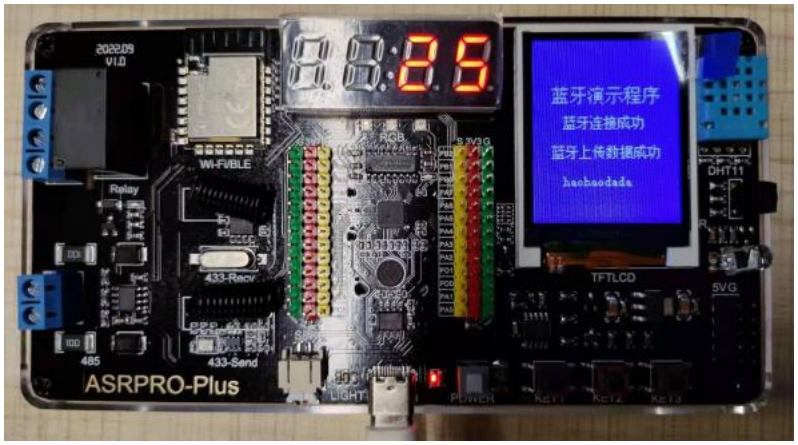
首先蓝牙每隔5s会收到一个数字，这个数字代表通过蓝牙发送的DHT11的湿度值。

接着我们可以按下上方的继电器开和继电器关按钮来控制继电器，实际上就是通过对蓝牙接收到的消息进行判断，收到Relay=1，继电器打开，消息显示区也会显示<-Realy=1。接收到Relay=0，继电器关闭。



按下上方的红绿蓝黑，可以控制板载RGB的颜色，当然我们也可以输入长度为15的消息，例如R=050G=100B=000，即RGB的值分别为50，100，0，蓝牙模块接收到消息后可以控制板载RGB灯显示对应的颜色。

我们还可以发送消息，控制彩屏显示文本。例如可以发送TFTLCD=haohaodada，就可以在（26，118）的位置显示haohaodada。点击彩屏显示(文本)可以自动写入TFTLCD= 。



这是蓝牙控制彩屏显示文本的效果图。

附录

附录一：语音识别设置注意事项

学习完整体的程序结构后，关于语音设置还有一些注意事项，这里给出一些中文语音和英文语音使用时的建议。

1. 一般为 4-6 个字，4 个字最佳，过短容误识高，过长不便用户呼叫和记忆；
2. 命令词中相邻汉字的声韵母区分度越大越好；
3. 符合用户的语言习惯，尽量采用常用说法，内容具体直接；
4. 应避免使用日常用语，如：“吃饭啦”;
5. 生僻字和零声母字应尽量避免，如“语音识别”中“语音”两个字均为零声母字;
6. 命令词中的字最好不要有语气词，如“啊”、“呢”等;
7. 应避免使用叠词，如：“你好你好”;
8. 中文命令词中只能由汉字组成,不允许有空格,逗号等其他字符;
9. 命令词中的数字需要以汉字表示，如“调高一度”;
10. 若您还未确定命令词，建议您从平台的“命令词推荐”中选择。
11. 英文建议由 2-4 个单词(4-6 个音节)组成，过短容误识高，过长不便用户记忆；
12. 英文命令词间音节区分度越大越好；
13. 英文的语音符合用户的语言习惯，尽量采用常用说法，内容具体直接；
14. 英文的唤醒词、命令词应避免使用日常用语，如：“HI、HELLO”；
15. 避免使用相似音节，词的发音清晰响度要大，如避免同时使用 TURN-ON 和 TURN-OFF ；
16. 应避免使用叠词，如：“HELLO -HELLO；

附录二：ASRPRO与其他单片机串口通讯的范例说明

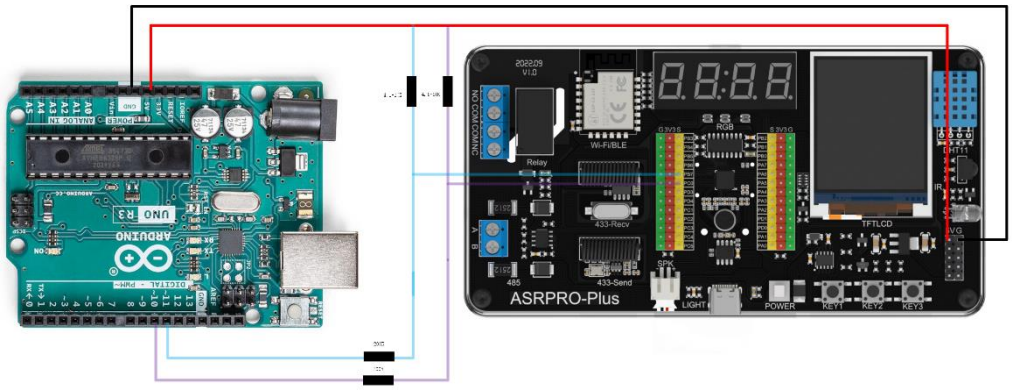
一、概述

ASRPRO有3组串口，UART0 预留为程序升级接口，方便后期升级。如需和其它 MCU 通讯建议使用 UART1 或者 UART2。

ASRPRO IO口为3.3V电平，为了可靠性，建议设置TX、RX引脚内部上下电阻无效，同时设置TX为开漏模式，外接上拉电阻到5V，串联电阻，电路示意图如下所示：

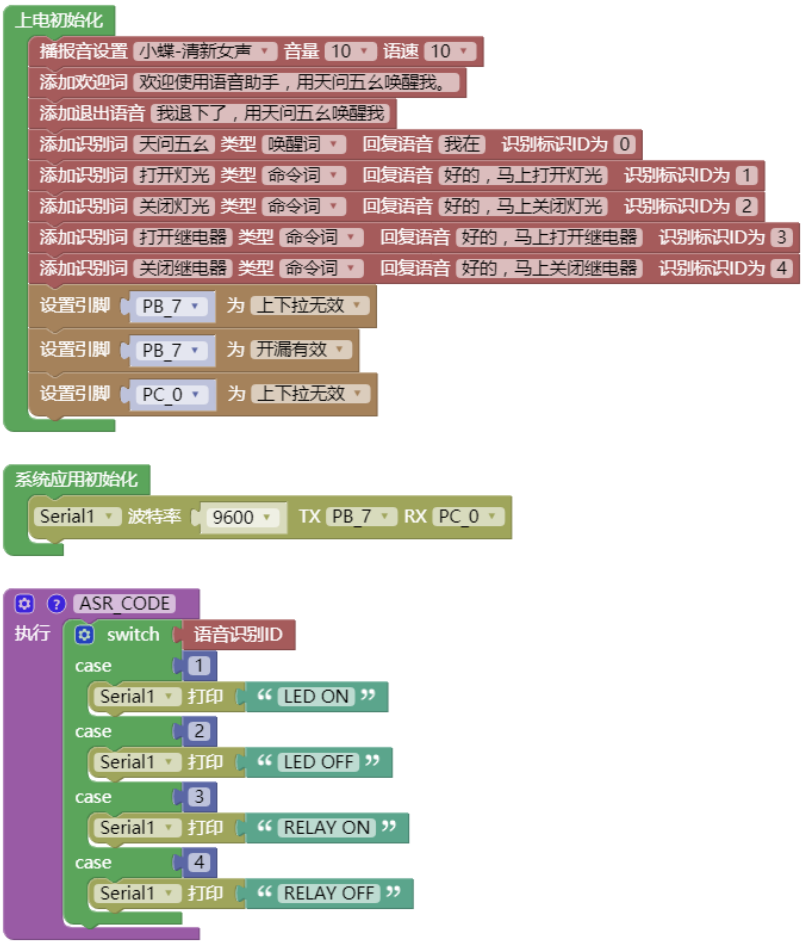
二、Arduino UNO（5V单片机）

1. 电路连接



2. 范例1：ASRPRO语音发送串口控制Arduino执行动作

1) ASRPRO 端程序



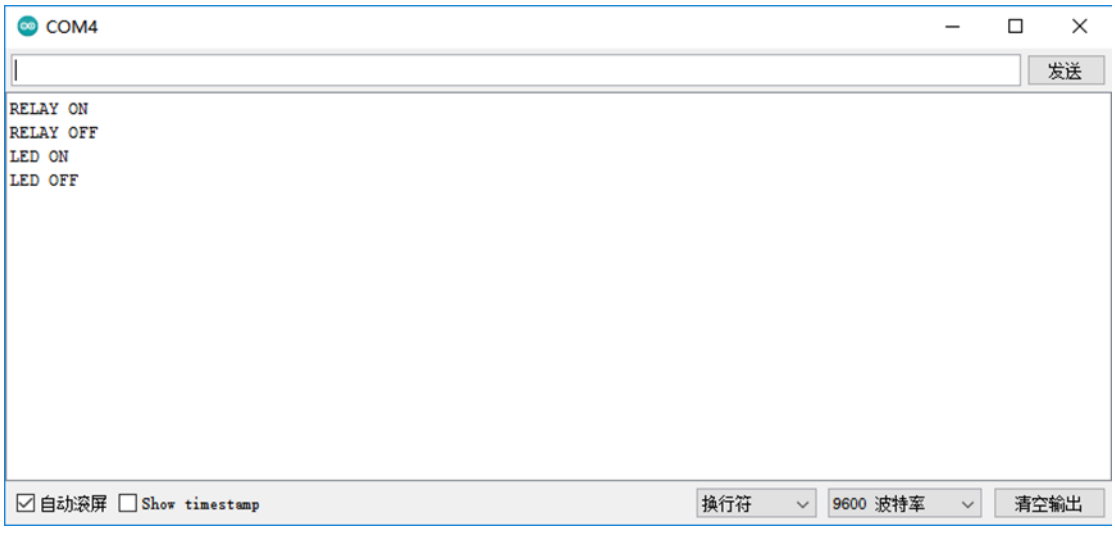
2) Arduino 端程序

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX, TX

String value;

#define LED_PIN   13
#define RELAY_PIN 12

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  mySerial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
  pinMode(RELAY_PIN, OUTPUT);
}

void loop() { // run over and over
```

```
  if (mySerial.available()) {
    value = (mySerial.readString());
    Serial.println(value);
    if (value == "LED ON")
    {
      digitalWrite(LED_PIN,HIGH);
    }
    else if (value == "LED OFF")
    {
      digitalWrite(LED_PIN,LOW);
    }
    else if (value == "RELAY ON")
    {
      digitalWrite(RELAY_PIN,HIGH);
    }
    else if (value == "RELAY OFF")
    {
      digitalWrite(RELAY_PIN,LOW);
    }
  }
}
```

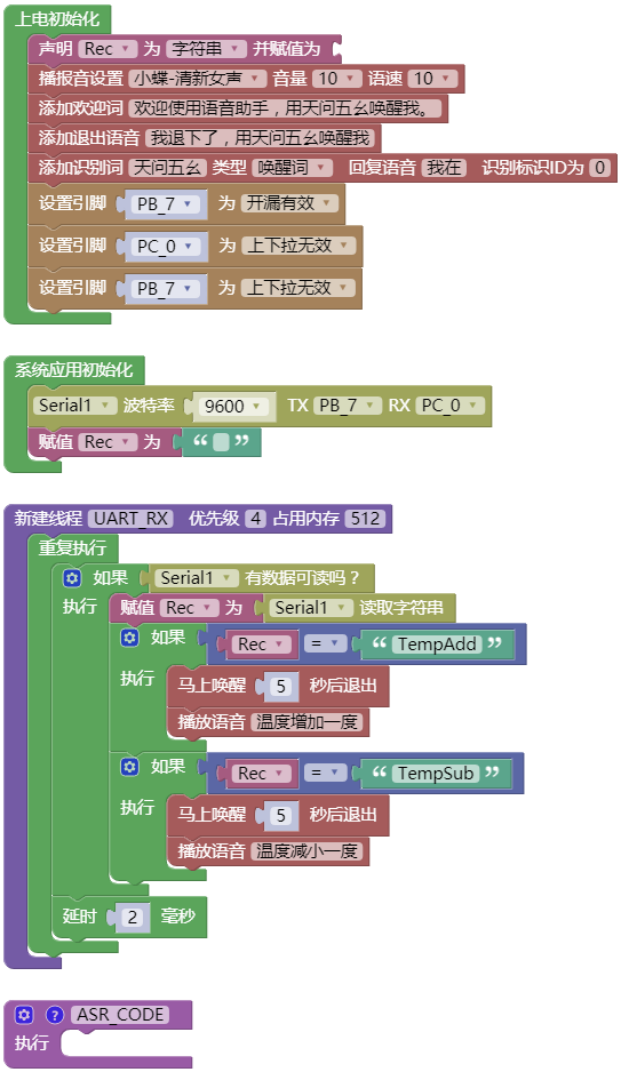
3) 程序效果

通过用“天问五幺”语音唤醒后，分别说测试语音“打开灯光”、“关闭灯光”、“打开继电器”、“关闭继电器”，Arduino端接收到串口命令后会执行对应引脚的控制和串口打印。



3. 范例2：Arduino发送串口控制ASRPRO播放语音

1) ASRPRO 端程序



2) Arduino 端程序

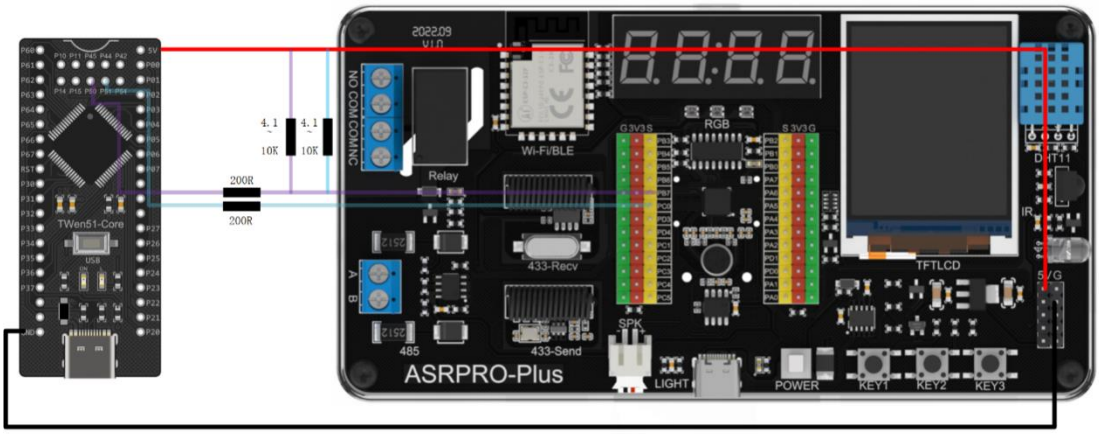
```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX, TX
void setup() {
  mySerial.begin(9600);
}
void loop() { // run over and over
  mySerial.print("TempAdd");
  delay(5000);
  mySerial.print("TempSub");
  delay(5000);
}
```

3) 程序效果

Arduino端间隔5秒串口发送“TempAdd”、"TempSub"，ASRPRO接收到串口命令后会马上唤醒自动播报语音“温度增加一度”、“温度减小一度”。

三、STC（5V单片机）

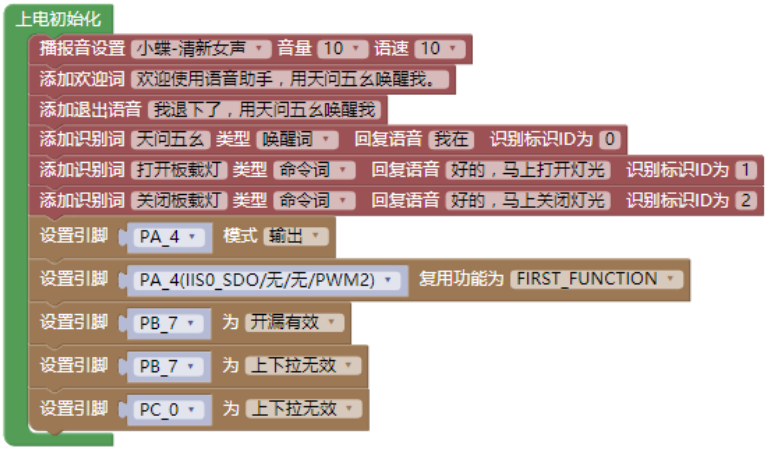
1. 电路连接

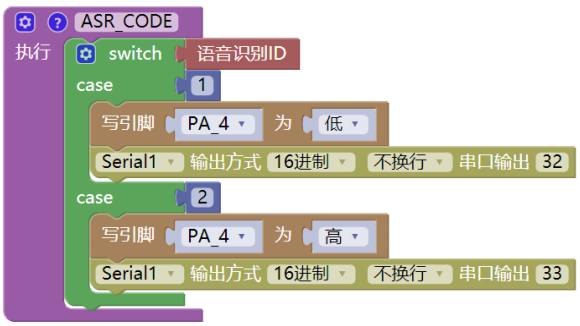


本案例以P5_0为RX，P5_1为TX举例。P5_0与ASRPRO的TX连接，也就是接在PB7，P5_1与ASRPRO的RX连接，也就是接在PC0，注意STC8的电源插脚接到5V，GND引脚互相连接。

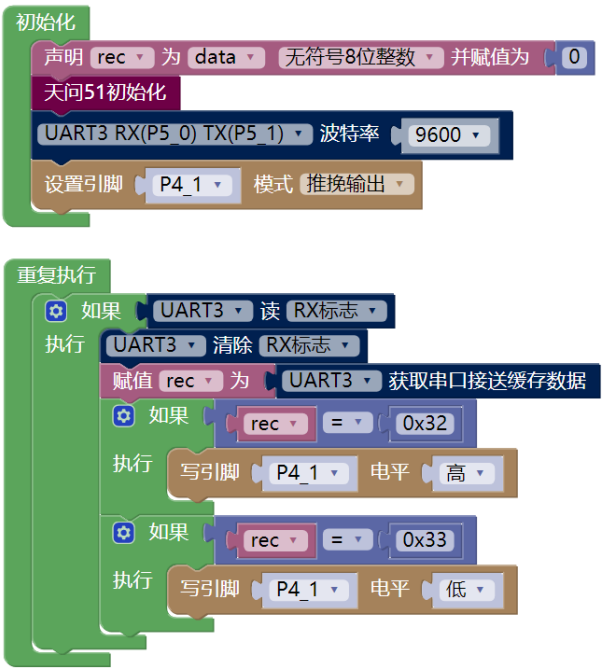
2. 范例：ASRPRO与STC8进行串口通讯语音控制STC8板载灯

1) ASRPRO 端程序



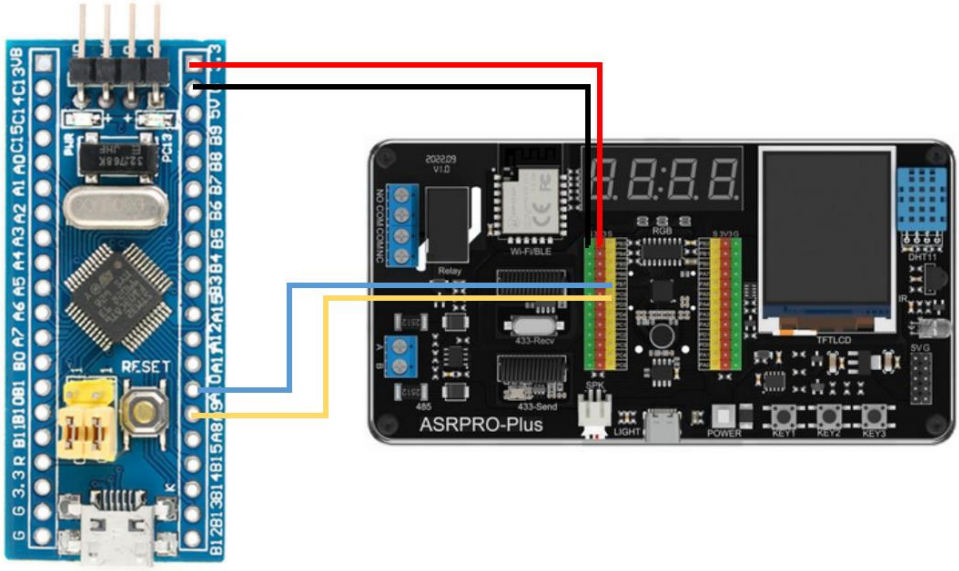


2) STC8 端程序



四、STM32（3V单片机）

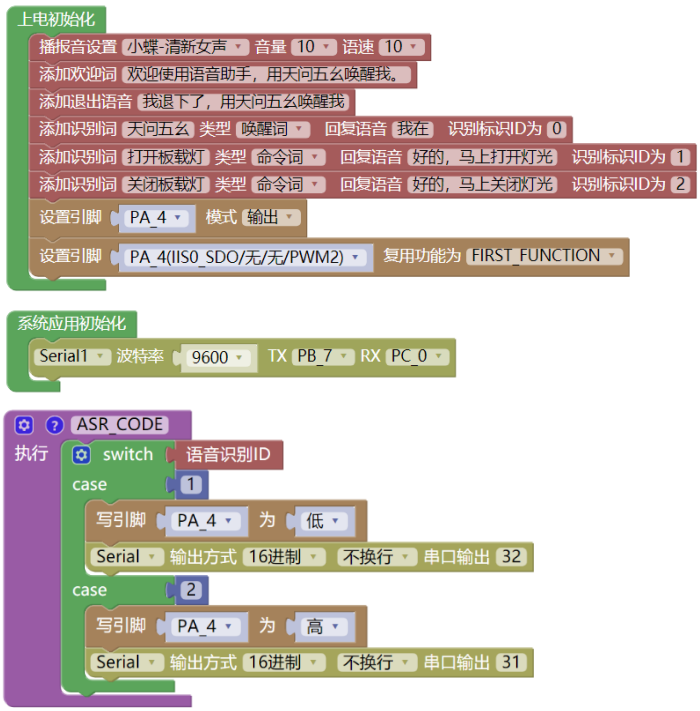
1. 电路连接



ASRPRO的PB7（TX）引脚接STM32的A10（RX）引脚，PC0（RX）引脚接STM32的A9（TX）引脚。

2. 范例1：ASRPRO语音发送串口控制STM32执行动作

1) ASRPRO 端程序



2) STM32 部分程序

main.c

```c
#include "stm32f10x.h"
#include "usart.h"
#include "led.h"
#include "delay.h"
u16 USART_RX_STA=0;                         //接收状态标记
static u16 fac_ms = 0;
int main(void)
{
    LED_Init();
    MyUSART_Init();
    delay_init();
    while(1)
    {
    }
}
```

usart.c

```c
#include "usart.h"
#include "led.h"

//重定向C库函数printf到串口，重定向后可使用printf函数
int fputc(int ch,FILE *f)
```

```
{
    /* 发送一个字节数据到串口 */
    USART_SendData(USART1,(uint8_t) ch);
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    return (ch);
}
//重定向C库函数scanf到串口,重写向后可使用scanf、getchar等函数
int fgetc(FILE *f)
{
    /* 等待串口输入数据 */
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    return (int)USART_ReceiveData(USART1);
}
void MyUSART_Init()
{
    /* 定义GPIO、NVIC和USART初始化的结构体 */
    GPIO_InitTypeDef GPIO_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    /* 使能GPIO和USART的时钟 */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
    /* 将USART TX（A9）的GPIO设置为推挽复用模式 */
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Init(GPIOA,&GPIO_InitStructure);
    /* 将USART RX（A10）的GPIO设置为浮空输入模式 */
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;
    GPIO_Init(GPIOA,&GPIO_InitStructure);

    /* 配置串口 */
    USART_InitStructure.USART_BaudRate=9600;                                    /
/波特率了设置为9600
    USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_None;
//不使用硬件流控制
```

```
    USART_InitStructure.USART_Mode=USART_Mode_Tx|USART_Mode_Rx;
//使能接收和发送
    USART_InitStructure.USART_Parity=USART_Parity_No;                                /
/不使用奇偶校验位
    USART_InitStructure.USART_StopBits=USART_StopBits_1;                             /
/1位停止位
    USART_InitStructure.USART_WordLength=USART_WordLength_8b;                        /
/字长设置为8位
    USART_Init(USART1, &USART_InitStructure);

    /* Usart1 NVIC配置 */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);                 //设置NVIC中断分组
2
    NVIC_InitStructure.NVIC_IRQChannel=USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelCmd=ENABLE;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority=0;
    NVIC_Init(&NVIC_InitStructure);

    /*初始化串口，开启串口接收中断 */
    USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
    /* 使能串口1 */
    USART_Cmd(USART1,ENABLE);

}
/* USART1中断函数 */
void USART1_IRQHandler(void)
{
    uint8_t ucTemp;                                          //接收数据
    if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET)
    {
        ucTemp = USART_ReceiveData(USART1);
        USART_SendData(USART1,ucTemp);
        if(ucTemp == 0x32)
        {
            LED_ON();
        }
        if(ucTemp == 0x31)
```

```
        {
           LED_OFF();
        }
    }
}
/* 发送一个字节 */
void Usart_SendByte( USART_TypeDef * pUSARTx, uint8_t ch)
{
    /* 发送一个字节数据到USART */
    USART_SendData(pUSARTx,ch);

    /* 等待发送数据寄存器为空 */
    while (USART_GetFlagStatus(pUSARTx, USART_FLAG_TXE) == RESET);
}
/* 发送字符串 */
void Usart_SendString( USART_TypeDef * pUSARTx, char *str)
{
    unsigned int k=0;
  do
  {
      Usart_SendByte( pUSARTx, *(str + k) );
      k++;
  } while(*(str + k)!='\0');

  /* 等待发送完成 */
  while(USART_GetFlagStatus(pUSARTx,USART_FLAG_TC)==RESET)
  {}
}


}
```

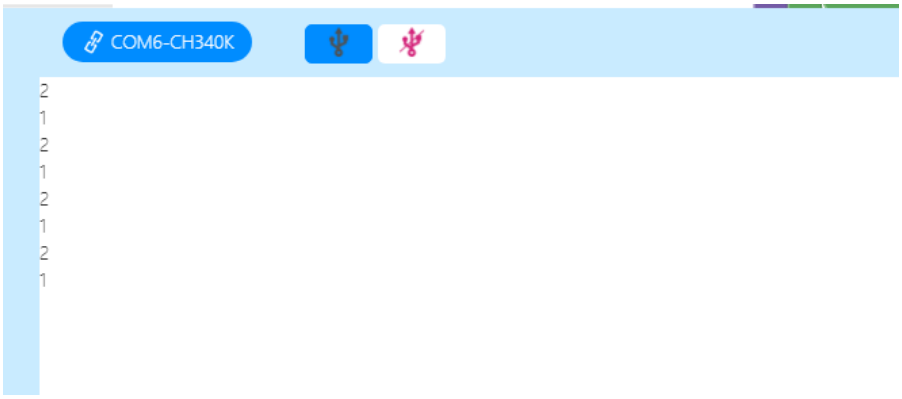
led.c

```
#include "led.h"
void LED_Init(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph_AFIO, ENABLE);    //
使能B端口时钟
    GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
```

```
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;         //推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度50MHz
    GPIO_Init(GPIOB, &GPIO_InitStructure);     //初始化GPIOB
    GPIO_SetBits(GPIOB,GPIO_Pin_10);
//  GPIO_SetBits(GPIOA,GPIO_Pin_8);
}
void LED_OFF(void)
{
    GPIO_SetBits(GPIOB,GPIO_Pin_10);
}
void LED_ON(void)
{
    GPIO_ResetBits(GPIOB,GPIO_Pin_10);
}
```

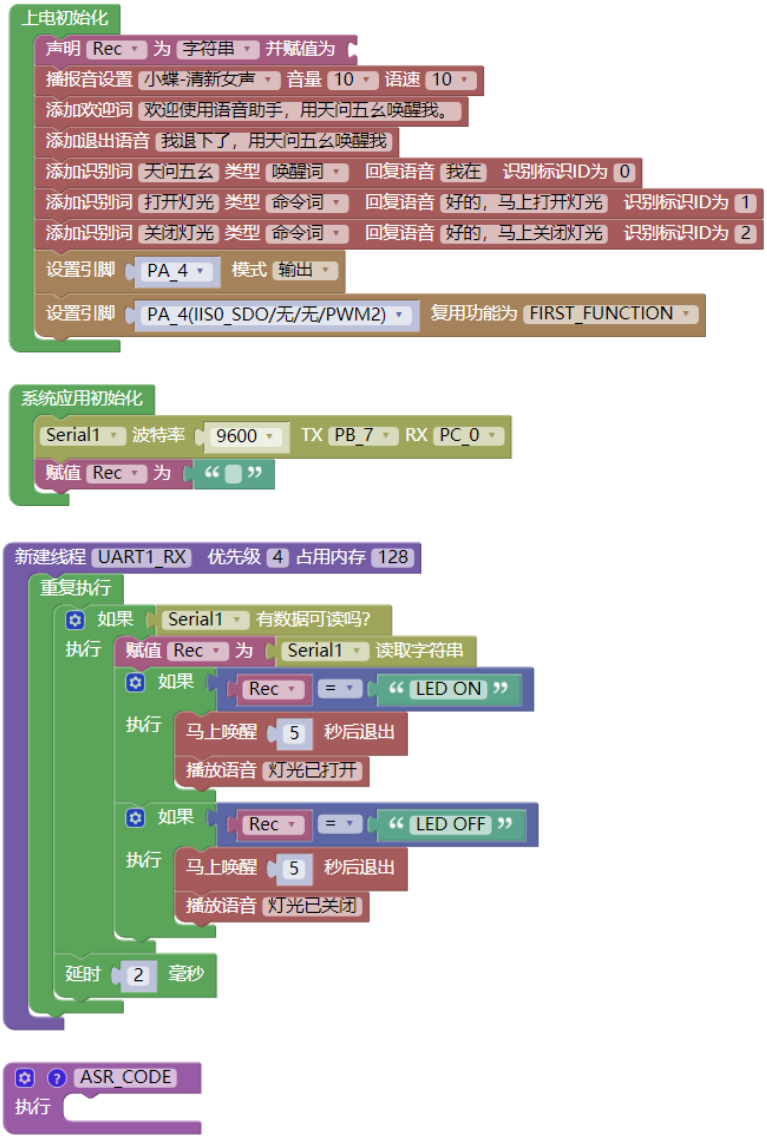
3) 程序效果

通过用“天问五幺”语音唤醒后，分别说测试语音“打开灯光”、“关闭灯光”，STM32端接收到串口命令后会执行对应引脚的控制和串口打印，“2”代表打开，“1”代表关闭。



3. 范例2：STM32串口发送控制ASRPRO播报语音

1) ASRPRO 端程序



2) STM32 部分程序

main.c

```
#include "stm32f10x.h"
#include "usart.h"
#include "led.h"
#include "delay.h"

u16 USART_RX_STA=0;                         //接收状态标记
static u16 fac_ms = 0;
//void delay_init(void);
//void delay_ms(u16 nms);
int main(void)
{
```

```
    LED_Init();
    MyUSART_Init();
    delay_init();
    while(1)
    {
        Usart_SendString( USART1,"LED ON");
        LED_ON();
        delay_ms(5000);
        Usart_SendString( USART1,"LED OFF");
        LED_OFF();
        delay_ms(5000);
    }
}
```

usart.c

```
#include "usart.h"
#include "led.h"

//重定向C库函数printf到串口，重定向后可使用printf函数
int fputc(int ch,FILE *f)
{
    /* 发送一个字节数据到串口 */
    USART_SendData(USART1,(uint8_t) ch);
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    return (ch);
}
//重定向C库函数scanf到串口,重写向后可使用scanf、getchar等函数
int fgetc(FILE *f)
{
    /* 等待串口输入数据 */
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    return (int)USART_ReceiveData(USART1);
}
void MyUSART_Init()
{
    /* 定义GPIO、NVIC和USART初始化的结构体 */
    GPIO_InitTypeDef GPIO_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    USART_InitTypeDef USART_InitStructure;
```

```
    /* 使能GPIO和USART的时钟 */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
    /* 将USART TX（A9）的GPIO设置为推挽复用模式 */
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Init(GPIOA,&GPIO_InitStructure);
    /* 将USART RX（A10）的GPIO设置为浮空输入模式 */
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;
    GPIO_Init(GPIOA,&GPIO_InitStructure);

    /* 配置串口 */
    USART_InitStructure.USART_BaudRate=9600;                                    /
/波特率了设置为9600
    USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_None;
//不使用硬件流控制
    USART_InitStructure.USART_Mode=USART_Mode_Tx|USART_Mode_Rx;                  /
/使能接收和发送
    USART_InitStructure.USART_Parity=USART_Parity_No;                           /
/不使用奇偶校验位
    USART_InitStructure.USART_StopBits=USART_StopBits_1;                        /
/1位停止位
    USART_InitStructure.USART_WordLength=USART_WordLength_8b;                   /
/字长设置为8位
    USART_Init(USART1, &USART_InitStructure);

    /* Usart1 NVIC配置 */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);                 //设置NVIC中断分组
2
    NVIC_InitStructure.NVIC_IRQChannel=USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelCmd=ENABLE;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority=0;
    NVIC_Init(&NVIC_InitStructure);

    /*初始化串口，开启串口接收中断 */
```

```
    USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
    /* 使能串口1 */
    USART_Cmd(USART1,ENABLE);

}
/* USART1中断函数 */
void USART1_IRQHandler(void)
{
    uint8_t ucTemp;                                          //接收数据
    if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET)
    {
        ucTemp = USART_ReceiveData(USART1);
        USART_SendData(USART1,ucTemp);
        if(ucTemp == 0x32)
        {
            LED_ON();
        }
        if(ucTemp == 0x31)
        {
            LED_OFF();
        }
    }
}
```

```
/* 发送一个字节 */
void Usart_SendByte( USART_TypeDef * pUSARTx, uint8_t ch)
{
    /* 发送一个字节数据到USART */
    USART_SendData(pUSARTx,ch);

    /* 等待发送数据寄存器为空 */
    while (USART_GetFlagStatus(pUSARTx, USART_FLAG_TXE) == RESET);
}
/* 发送字符串 */
void Usart_SendString( USART_TypeDef * pUSARTx, char *str)
{
    unsigned int k=0;
  do
```

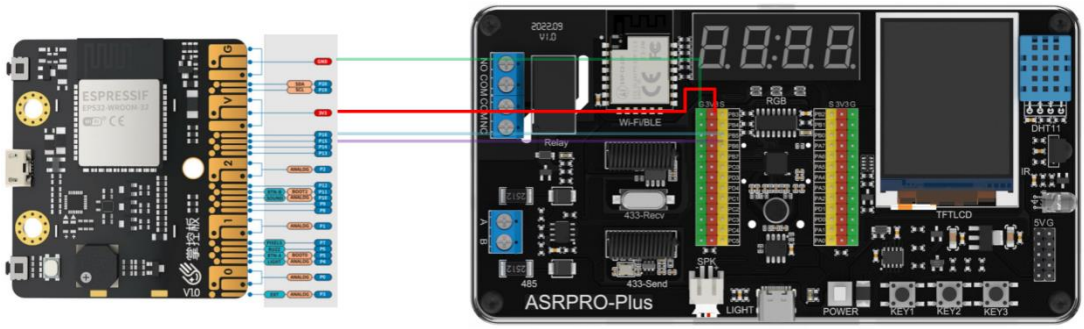
```
  {
      Usart_SendByte( pUSARTx, *(str + k) );
      k++;
  } while(*(str + k)!='\0');

  /* 等待发送完成 */
  while(USART_GetFlagStatus(pUSARTx,USART_FLAG_TC)==RESET)
  {}
}
```

3) 程序效果

STM32端间隔一段时间串口发送“LED ON”、"LED OFF"，ASRPRO接收到串口命令后会马上唤醒自动播报语音“灯光已打开”、“灯光已关闭”。

五、ESP32（3V单片机）

1. 电路连接

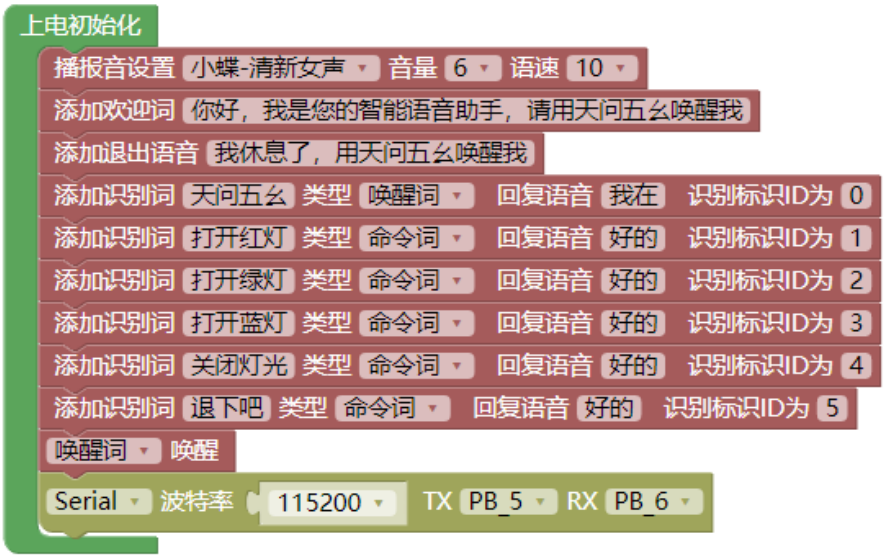


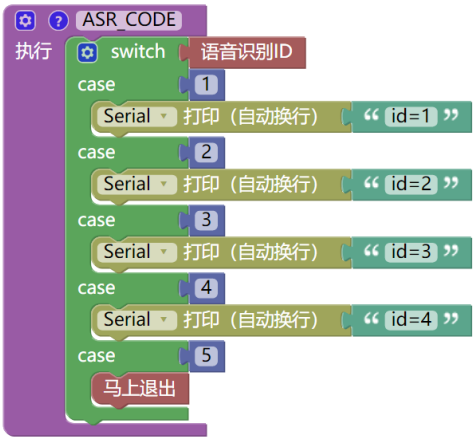
掌控板的P15引脚的TX，接ASRPRO的RX，也就是接在PB6；P16引脚接到ASRPRO RX引脚（PB5），两者的3V引脚互相连接，GND引脚互相连接。

2. 范例：ASRPRO与掌控板进行串口通讯

语音控制ASRPRO发送串口数据，并控制掌控板的板载RGB灯显示不同的颜色。

1) ASRPRO 端程序


上电初始化
播报音设置 小蝶-清新女声 音量 6 语速 10
添加欢迎词 你好，我是您的智能语音助手，请用天问五幺唤醒我
添加退出语音 我休息了，用天问五幺唤醒我
添加识别词 天问五幺 类型 唤醒词 回复语音 我在 识别标识ID为 0
添加识别词 打开红灯 类型 命令词 回复语音 好的 识别标识ID为 1
添加识别词 打开绿灯 类型 命令词 回复语音 好的 识别标识ID为 2
添加识别词 打开蓝灯 类型 命令词 回复语音 好的 识别标识ID为 3
添加识别词 关闭灯光 类型 命令词 回复语音 好的 识别标识ID为 4
添加识别词 退下吧 类型 命令词 回复语音 好的 识别标识ID为 5
唤醒词 唤醒
Serial 波特率 115200 TX PB_5 RX PB_6

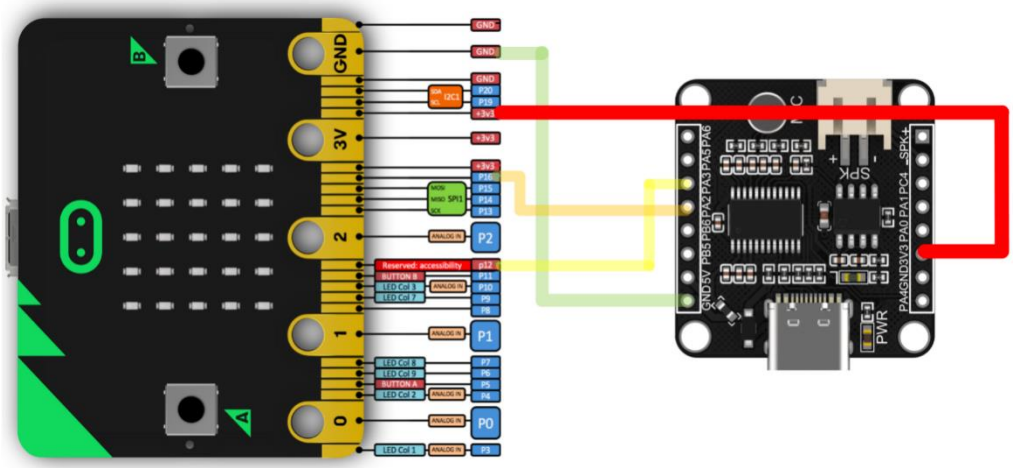

ASR_CODE
执行
switch 语音识别ID
case 1
Serial 打印（自动换行） “ id=1 ”
case 2
Serial 打印（自动换行） “ id=2 ”
case 3
Serial 打印（自动换行） “ id=3 ”
case 4
Serial 打印（自动换行） “ id=4 ”
case 5
马上退出

2) 掌控板端程序


串口 uart1 初始化 波特率 115200 tx P15 rx P16
重复执行
如果 串口 uart1 有可读数据
执行
将变量 ck 设定为 字节 串口 uart1 读取数据 转 字符串
将变量 id 设定为 从文本 ck 取得一段字串自# 4 到字符# 4
OLED 显示 清空
OLED 第 1 行显示 转为文本 ck 模式 普通
OLED 第 2 行显示 转为文本 id 模式 普通
OLED 显示生效
如果 id = “ 1 ”
执行 设置 所有 RGB 灯颜色为
如果 id = “ 2 ”
执行 设置 所有 RGB 灯颜色为
如果 id = “ 3 ”
执行 设置 所有 RGB 灯颜色为
如果 id = “ 4 ”
执行 关闭 所有 RGB 灯
等待 100 毫秒

六、micro:bit（3V 单片机）

1. 电路连接



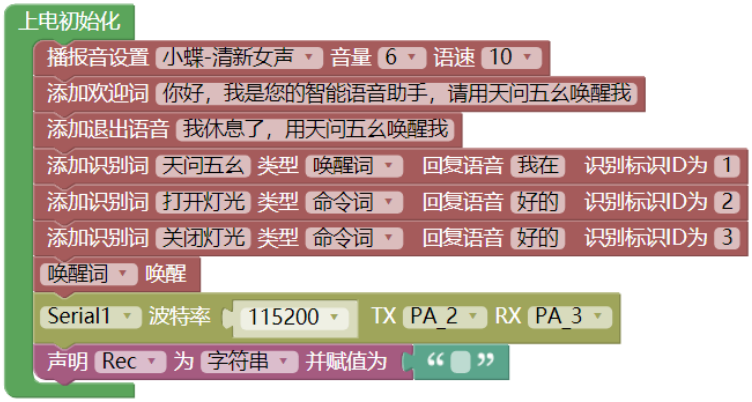
在进行串口通信时，两个设备进行双向通信，此时两个设备的RX和TX要交错连接。例如micro:bit，定义P16为RX口，则要接到ASRPRO的TX上，也就是PA2。

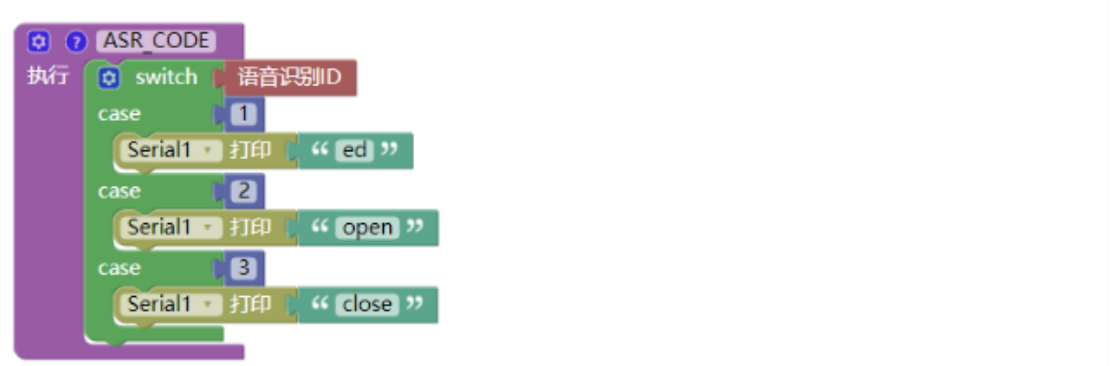
micro:bit的P16引脚的RX，接ASRPRO的TX，也就是接在PA2；P12引脚接到ASRPRO的RX引脚（PA3），两者的3V引脚互相连接，GND引脚互相连接。

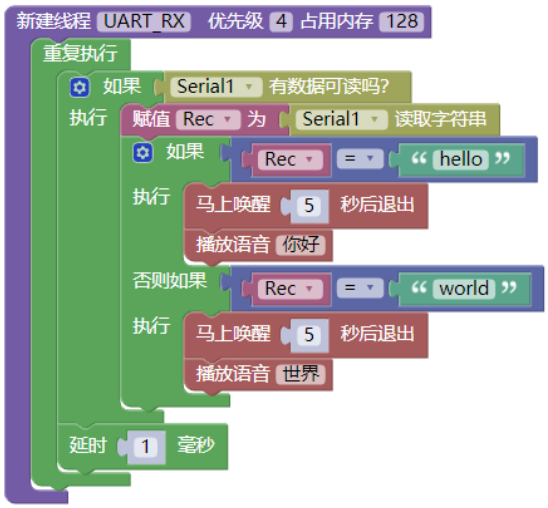
2. 范例：ASRPRO与micro:bit进行串口通讯

按下micro:bit的AB按键，通过串口可以给ASRPRO发送字符串hello和world；当ASRPRO接收到后，就会回复对应的语音；当对ASRPRO说出“打开灯光、关闭灯光”时，ASRPRO和micro:bit的串口信息会显示在micro:bit的点阵屏上。

1) ASRPRO 程序







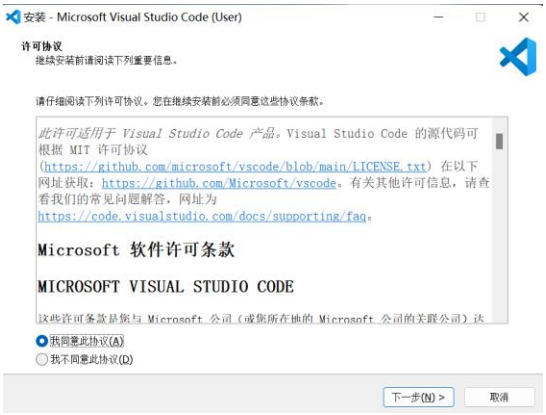
2) micro:bit 程序

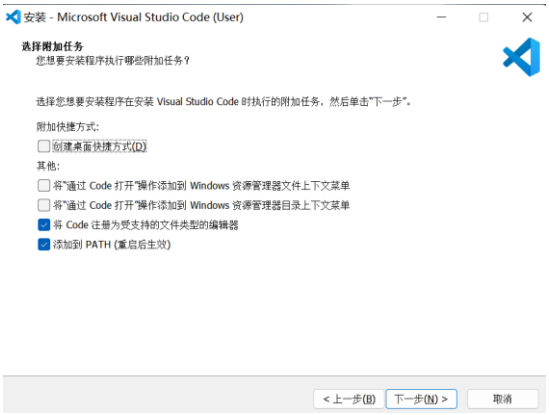
附录三：VS Code使用与全编译下载

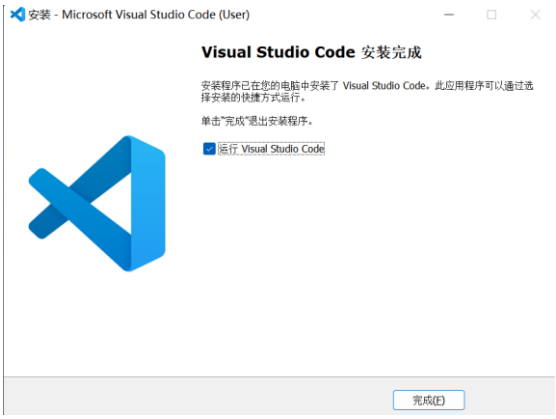
一、VS Code安装

ASRPRO的字符编程已经全面和VS Code打通。我们可以使用VSC对代码进行查看和修改，并进行编译下载。

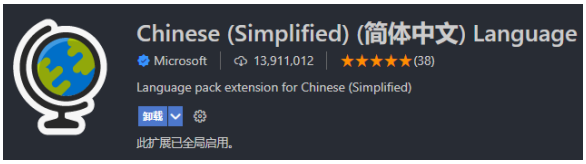
首先安装VS Code，并安装部分常规插件。注意第二幅图必须勾选添加到PATH，即勾选配置环境变量，否则无法通过天问Block的字符编程模式右键打开。







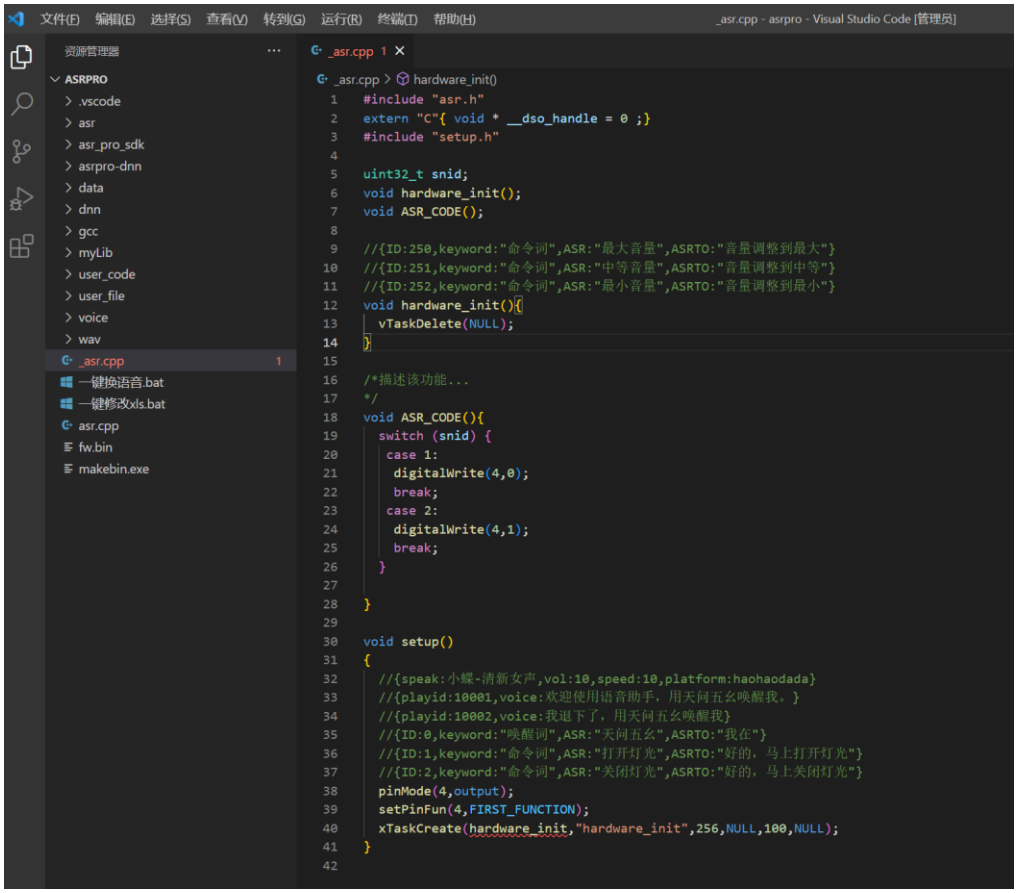
为了方便后续使用，建议安装简体中文插件。按下crtl+Shift+X，开始搜索。



安装后，我们就可以直接通过天问Block的字符编程，右键，选择VS Code打开文件。

```
#include "asr.h"
extern "C"{ void * __dso_handle = 0 ;}
#include "setup.h"

uint32_t snid;
void hardware_init();
void ASR_CODE();

//{ID:250,keyword:"命令词",ASR:"最大音量",ASRTO:"音量调整到最大"}
//{ID:251,keyword:"命令词",ASR:"中等音量",ASRTO:"音量调整到中等"}
//{ID:252,keyword:"命令词",ASR:"最小音量",ASRTO:"音量调整到最小"}
void hardware_init(){
  vTaskDelete(NULL);
}

/*描述该功能...
*/
void ASR_CODE(){
  switch (snid) {
   case 1:
    digitalWrite(4,0);
    break;
   case 2:
    digitalWrite(4,1);
    break;
  }

}

void setup()
{
```

这里以范例代码1.1智能语音对话为例，打开VSC后，我们可以在左边看到ASRPRO文件的目录，右边是我们打开的代码，名称叫做_asr.app。



_asr.app这段可编辑的代码，与我们天问Block的字符编程区的代码是同步的。也就是说，当我们在这里修改并点击左上角文件-保存时，天问Block区域的字符代码也会改变。

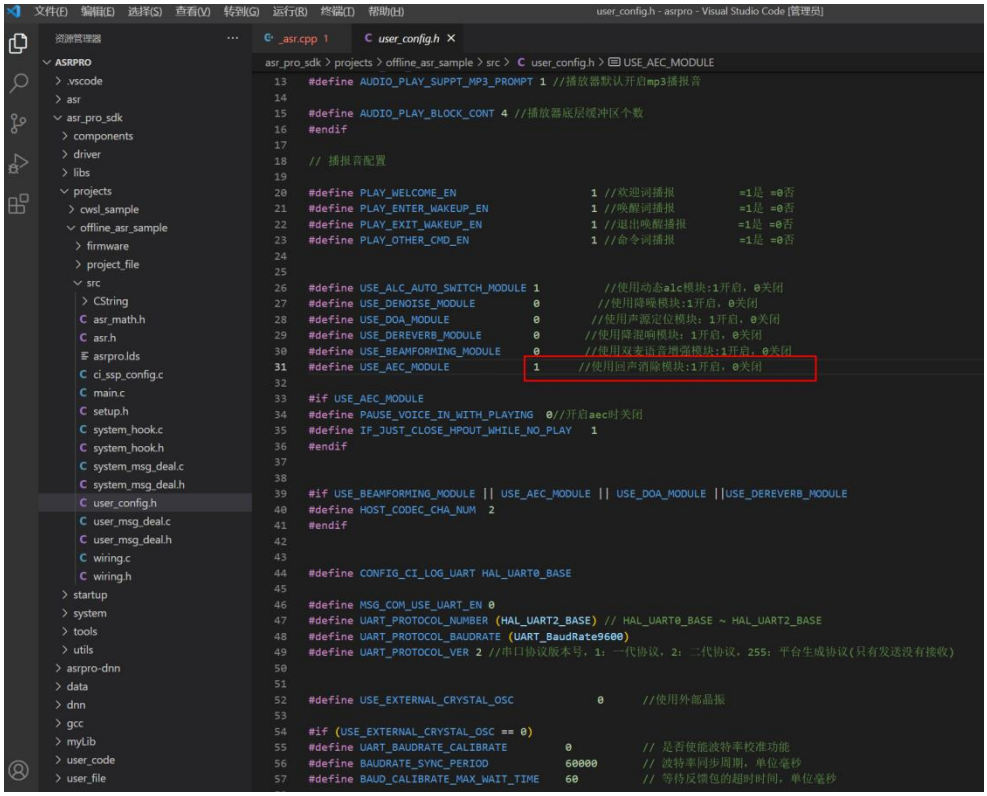
当然在实际过程中，_asr.app并不参与编译下载，这段代码只是为了将代码同步到天问Block中，实际我们用天问Block进行编译下载时，真正参与的是下方的asr.app。

二、全编译下载

这里以修改打断功能为例，介绍如何进行在VS Code中进行全部重新编译下载。

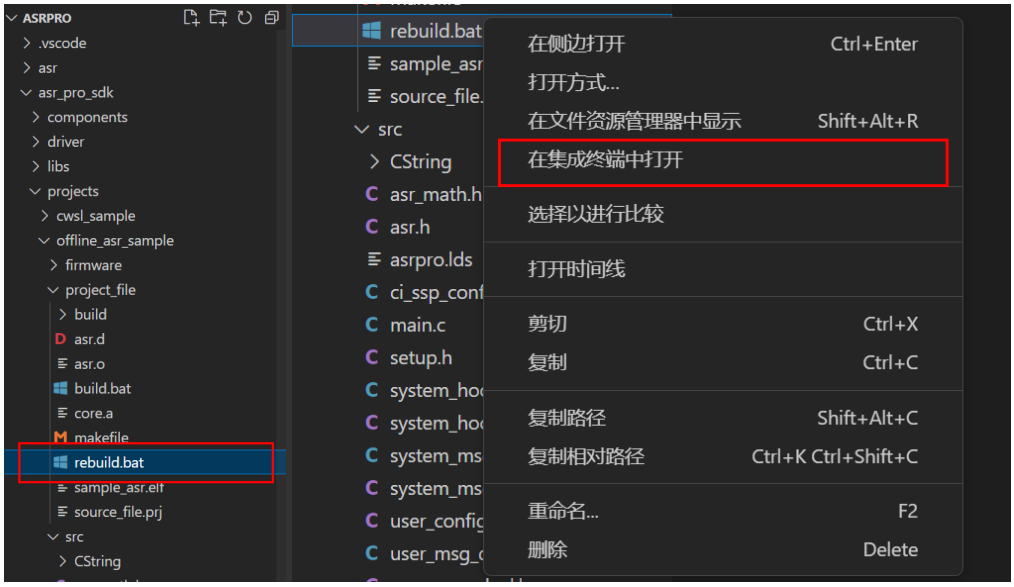
所谓打断功能，就是我们可以使用唤醒词对正在播放的语音进行打断，不会受到正在播放的声音的影响。如果不启用打断功能，那么一长条的语音播报是无法通过语音控制中断的。

打断功能的修改在asr_pro_sdk→projects→offline_asr_sample→src→user_config.h文件中，其中使用回声消除模块就是打断功能，1开启0关闭，默认是关闭，如下图所示。

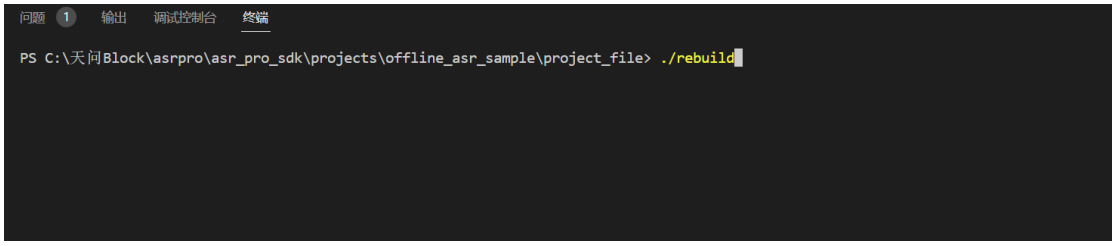


修改后为了使打断功能生效，我们需要重新编译。

重新编译rebuild功能的修改asr_pro_sdk→projects→offline_asr_sample→project_file→Rebuild.bat文件中。右键点击该文件，选择在集成终端中打开。



打开后输入./rebuild，按下回车，就会开始全部重新编译下载。



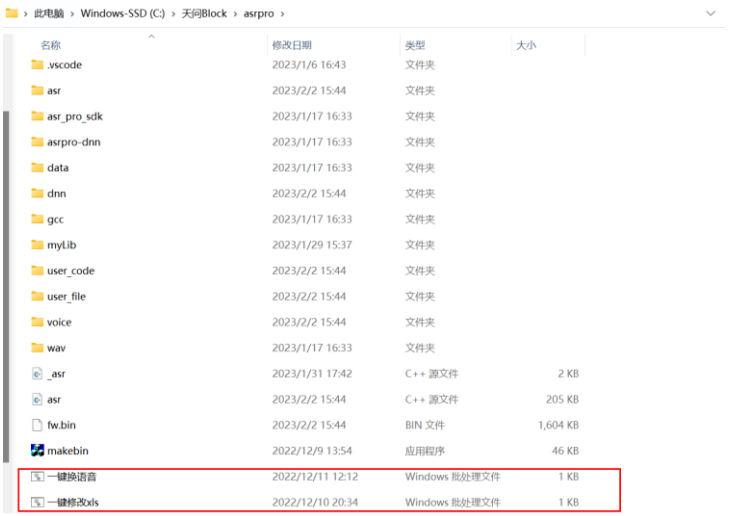
编译完成后会跳出烧写工具。连接ASRPRO Plus，点击烧写即可。

烧写后ASRPRO Plus就拥有打断功能，可以用唤醒词打断正在播报的语音。

附录四：一键语音替换和修改xls

一、前言

当我们需要修改替换、新增语音但是无法直接通过天问Block实现时，可以使用天问Block-asrpro文件夹中的一键换语音.bat和一键换xls.bat。



二、语音替换

当我们需要替换语音识别的回复词时，可以选择将天问Block-asrpro-voice-mp3目录下的语音文件进行替换。这些语音文件都是生成模型后，唤醒词和命令词对应的回复语和一些基础用语。例如下方图片，0是唤醒后的回复语，1-6是程序中添加的命令词的回复语，文件名称全部采用[1]命令词，如[1]打开板载灯，250-252是固定的音量回复语，10001是开机语音，10002是退出语音。

我们以替换ID=1的语音为例，虽然名称是[1]打开板载灯，实际内容实际上是“好的，马上打开板载灯”，这也是我们程序中生成模型后的结果。我们用一个mp3文件替换它，将原来的文件删除，然后将名称修改为[1]打开板载灯。

修改后，点击天问Block-asrpro文件夹中的一键换语音即可。

之后点击编译下载，无需重新生成模型。

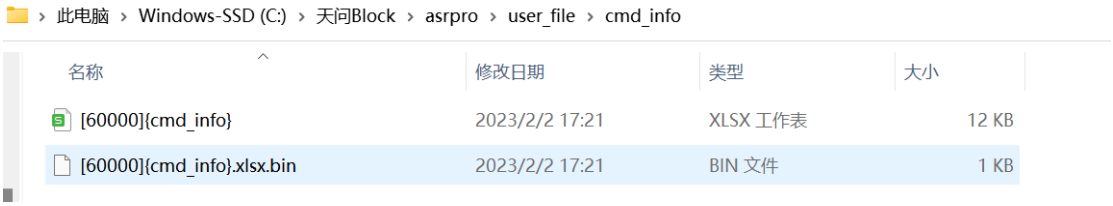
三、语音新增

当我们需要新增语音时，就需要同时对以下两部分内容进行修改。

首先是在mp3文件夹中新增语音文件。注意语音名称要是[ID]命令词的形式，这里设置语音ID=7，命令词为你好。

接着为了让程序能够顺利找到这个语音文件，我们需要修改

天问Block-asrpro-user_file-cmd_info文件夹中的[6000]{cmd_info}.xlsx文件



打开表格，复制一行命令词，并修改命令词的名称、对应的ID，如下所示。

	A	B	C	D	E	F	G	H	I	J	K	L	M
	模型名	模型ID	命令词	命令词ID	命令词语义ID	置信度	唤醒词	组合词	期望词	不期望词	特殊词计数	播报音类型	播报音0ID
DNN	ID	0	天问五幺	0	0x00	35	YES	NO	NO	NO	0	自定义	0
ASR	ID	0	最大音量	250	0x1E419C5	35	NO	NO	NO	NO	12	自定义	250
			中等音量	251	0x1E41A01	35	NO	NO	NO	NO	0	自定义	251
			最小音量	252	0x1E41A45	35	NO	NO	NO	NO	0	自定义	252
			打开灯光	1	0x00	32	NO	NO	NO	NO	0	自定义	1
			关闭板载灯	2	0x00	32	NO	NO	NO	NO	0	自定义	2
			打开继电器	3	0x00	32	NO	NO	NO	NO	0	自定义	3
			关闭继电器	4	0x00	32	NO	NO	NO	NO	0	自定义	4
			打开彩屏背光	5	0x00	32	NO	NO	NO	NO	0	自定义	6
			关闭彩屏背光	6	0x00	32	NO	NO	NO	NO	0	自定义	6
			<welcome>	10001	0x00	0	NO	NO	NO	NO	0	自定义	10001
			<inactivate>	10002	0x00	0	NO	NO	NO	NO	0	自定义	10002
			<beep>	10003	0x00	0	NO	NO	NO	NO	0	自定义	10003
			你好	7	0x00	32	NO	NO	NO	NO	0	自定义	7

修改后点击asrpro文件夹中的一键修改xls.bat。

当然需要注意的是，语音ID=5和6的播报音0ID都是6，这是因为它们两个的回复语相同，最后调用的mp3文件也相同。

打开彩屏背光	5	0x00	32	NO	NO	NO	NO	0	自定义	6
关闭彩屏背光	6	0x00	32	NO	NO	NO	NO	0	自定义	6

四、随机回复

当我们需要语音随机回复时，需要修改表格内容。

即对天问Block-asrpro-user_file-cmd_info文件夹中的[6000]{cmd_info}.xlsx文件部分内容进行修改。

打开表格，找到“播报音类型”，并下拉选择“随机”，如下所示。

模型名	模型ID	命令词	命令词ID	命令词语义ID	置信度	唤醒词	组合词	期望词	不期望词	特殊词计数	播报音类型	播报音0ID	播报音1ID	...
DNN ID	0	天问五幺	0	0x00	35	YES	NO	NO	NO	0	自定义	0		
ASR ID	0	最大音量	250	0x1E419C5	35	NO	NO	NO	NO	12	自定义	250		
		中等音量	251	0x1E41A01	35	NO	NO	NO	NO	0	随机	251		
		最小音量	252	0x1E41A45	35	NO	NO	NO	NO	0		252		
		打开板载灯	1	0x00	32	NO	NO	NO	NO	0	自定义	1		
		关闭板载灯	2	0x00	32	NO	NO	NO	NO	0	自定义	2		
		打开继电器	3	0x00	32	NO	NO	NO	NO	0	自定义	3		
		关闭继电器	4	0x00	32	NO	NO	NO	NO	0	自定义	4		
		打开彩屏背光	5	0x00	32	NO	NO	NO	NO	0	自定义	5		
		关闭彩屏背光	6	0x00	32	NO	NO	NO	NO	0	自定义	6		
		<welcome>	10001	0x00	0	NO	NO	NO	NO	0	自定义	10001		
		<inactivate>	10002	0x00	0	NO	NO	NO	NO	0	自定义	10002		
		<beep>	10003	0x00	0	NO	NO	NO	NO	0	自定义	10003		

可根据自己需求复制并插入多列播报音0ID栏，分别修改为“播报音0ID”、“播报音1ID”、“播报音2ID”...，如下所示。

模型名	模型ID	命令词	命令词ID	命令词语义ID	置信度	唤醒词	组合词	期望词	不期望词	特殊词计数	播报音类型	播报音0ID	播报音1ID	播报音2ID	...
DNN ID	0	天问五幺	0	0x00	35	YES	NO	NO	NO	0	随机	0			
ASR ID	0	最大音量	250	0x1E419C5	35	NO	NO	NO	NO	12	自定义	250			
		中等音量	251	0x1E41A01	35	NO	NO	NO	NO	0	自定义	251			
		最小音量	252	0x1E41A45	35	NO	NO	NO	NO	0	自定义	252			
		打开板载灯	1	0x00	32	NO	NO	NO	NO	0	自定义	1			
		关闭板载灯	2	0x00	32	NO	NO	NO	NO	0	自定义	2			
		打开继电器	3	0x00	32	NO	NO	NO	NO	0	自定义	3			
		关闭继电器	4	0x00	32	NO	NO	NO	NO	0	自定义	4			
		打开彩屏背光	5	0x00	32	NO	NO	NO	NO	0	自定义	5			
		关闭彩屏背光	6	0x00	32	NO	NO	NO	NO	0	自定义	6			
		<welcome>	10001	0x00	0	NO	NO	NO	NO	0	自定义	10001			
		<inactivate>	10002	0x00	0	NO	NO	NO	NO	0	自定义	10002			
		<beep>	10003	0x00	0	NO	NO	NO	NO	0	自定义	10003			

在需要随机播报的命令词栏目分别填入已生成的命令词ID并保存表格修改，如下所示。

模型名	模型ID	命令词	命令词ID	命令词语义ID	置信度	唤醒词	组合词	期望词	不期望词	特殊词计数	播报音类型	播报音0ID	播报音1ID	播报音2ID	...
DNN ID	0	天问五幺	0	0x00	35	YES	NO	NO	NO	0	随机	0	3	5	
ASR ID	0	最大音量	250	0x1E419C5	35	NO	NO	NO	NO	12	自定义	250			
		中等音量	251	0x1E41A01	35	NO	NO	NO	NO	0	自定义	251			
		最小音量	252	0x1E41A45	35	NO	NO	NO	NO	0	自定义	252			
		打开板载灯	1	0x00	32	NO	NO	NO	NO	0	自定义	1			
		关闭板载灯	2	0x00	32	NO	NO	NO	NO	0	自定义	2			
		打开继电器	3	0x00	32	NO	NO	NO	NO	0	自定义	3			
		关闭继电器	4	0x00	32	NO	NO	NO	NO	0	自定义	4			
		打开彩屏背光	5	0x00	32	NO	NO	NO	NO	0	自定义	5			
		关闭彩屏背光	6	0x00	32	NO	NO	NO	NO	0	自定义	6			
		<welcome>	10001	0x00	0	NO	NO	NO	NO	0	自定义	10001			
		<inactivate>	10002	0x00	0	NO	NO	NO	NO	0	自定义	10002			
		<beep>	10003	0x00	0	NO	NO	NO	NO	0	自定义	10003			

修改后点击asrpro文件夹中的一键修改xls.bat，然后编译下载即可，无需再重新生成模型。