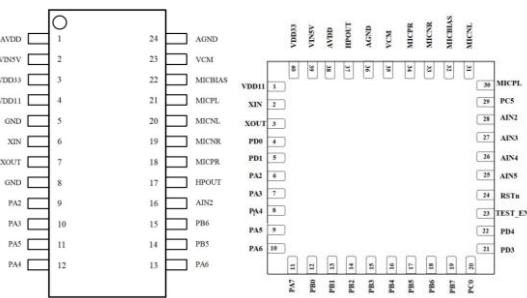


ASRPRO 快速上手（标准模式）

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。

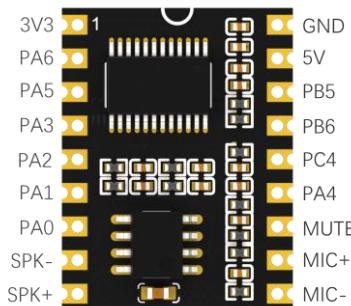
芯片特点

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。

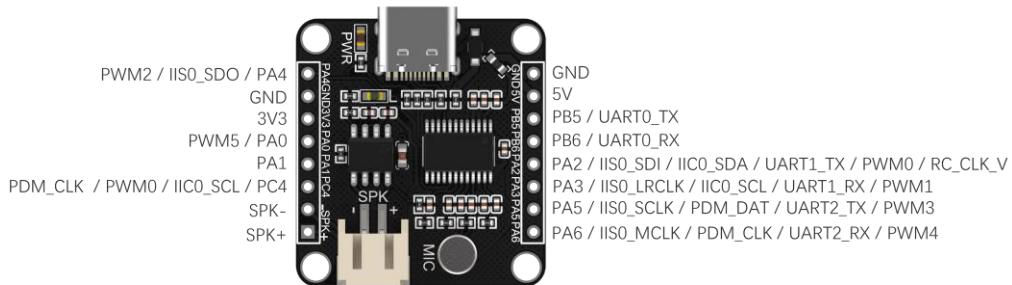
硬件概述

天问基于 ASRPRO 芯片目前推出了 5 种类型，供开发者选择。

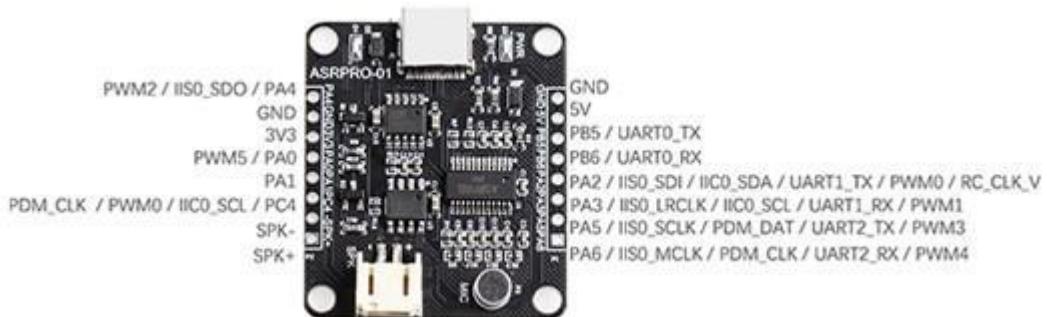
1. ASRPRO-CORE 核心板，模块体积小巧，长宽为 18x23mm，对外接口采用 2 排邮票孔和插针孔，方便采用回流贴片使用和焊接插针使用，喇叭和麦克风都需要自己外接，下载程序需要搭配 STC-LINK 下载器。



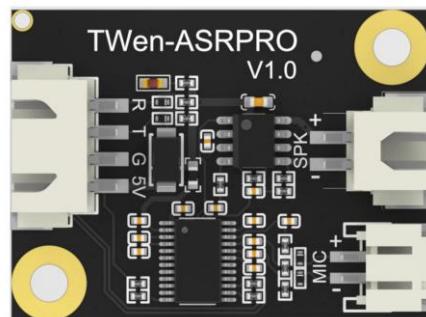
2. ASRPRO 基础开发板，长宽为 30x28mm，板载麦克风、指示灯，用户只需要外接喇叭就可以使用，下载程序需要搭配 STC-LINK 下载器。



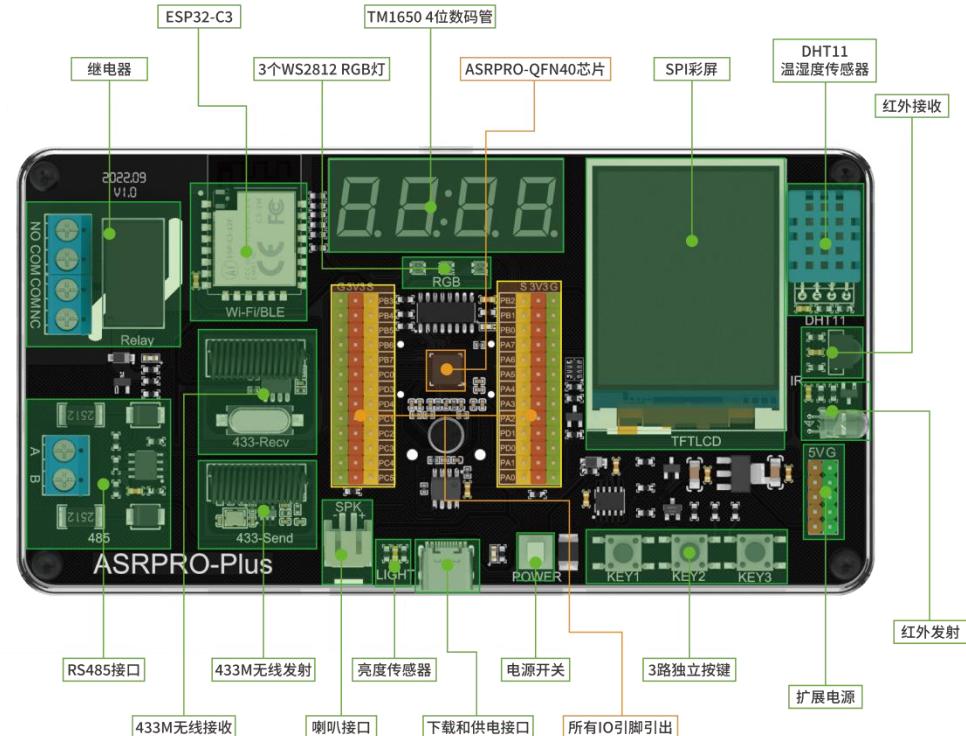
3. 鹿小班 ASRPRO 基础开发板，在 ASRPRO 基础开发板的基础上额外集成了下载芯片，一根 Type-C 线就可以下载程序，并且开发板上有自动断电电路可以实现一键下载，不需要额外的 STC-LINK 下载器。



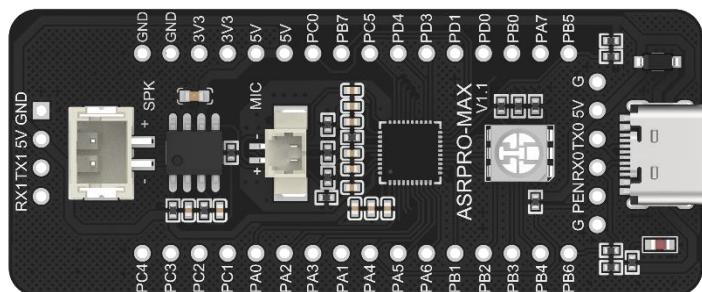
4. ASRPRO 串口模块，只引出了串口、喇叭、麦克风供用户和其它主控搭配使用。



5. ASRPRO-Plus 开发板，一款全功能带语音识别的物联网开发板，方便学习。板载 RS485、433M 无线收发、红外收发、ESP32-C3(2.4GHz Wi-Fi 和 Bluetooth 5LE)、SPI 彩屏、数码管、RGB 灯、光敏传感器、DHT11 温湿度传感器、1 路继电器输出模块。



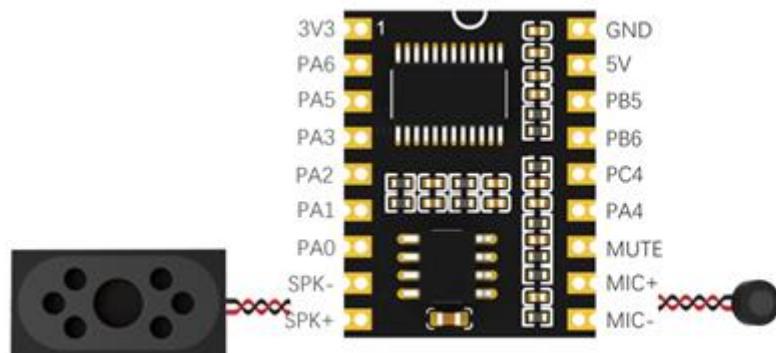
6. ASRPRO-MAX核心板，采用QFN40封装芯片，26路IO口独立引出，长宽为62.0x25.5mm，板载RGB，用户只需要外接喇叭和咪头就可以使用，下载程序需要搭配STC-LINK下载器。



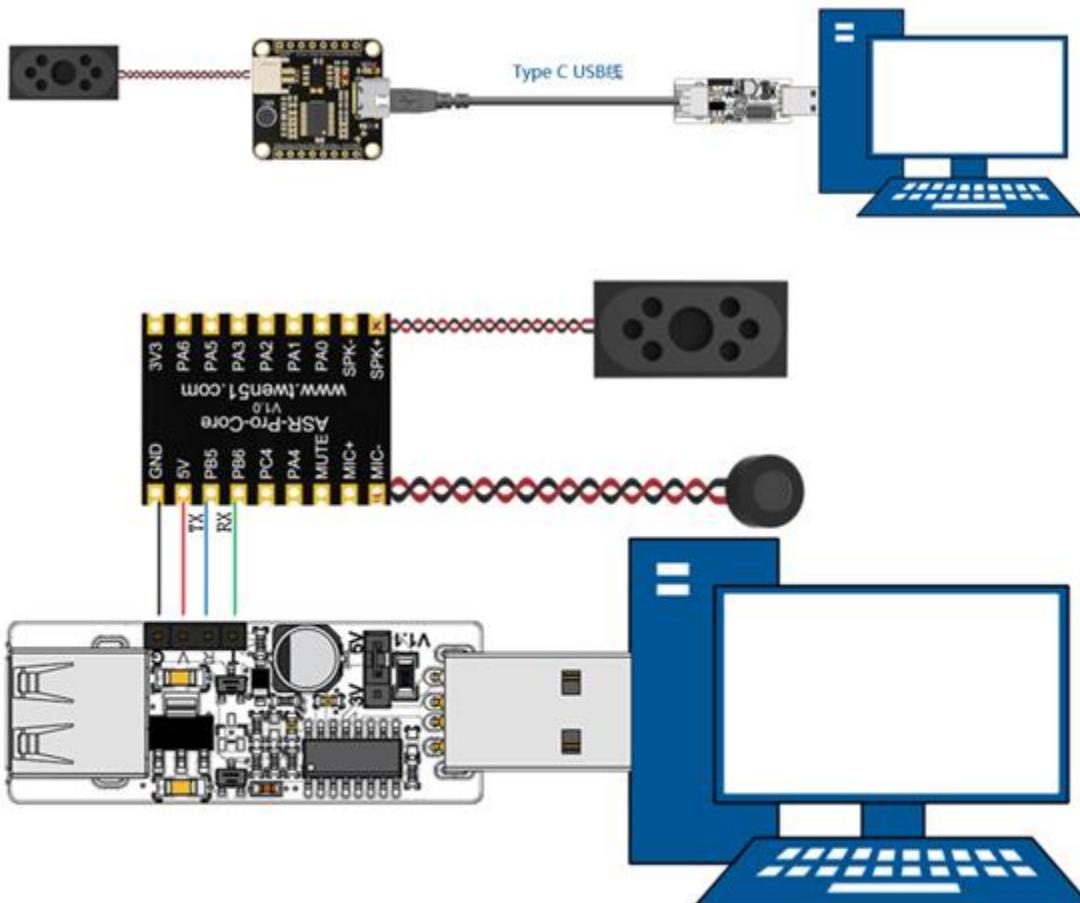
二、开机测试

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。

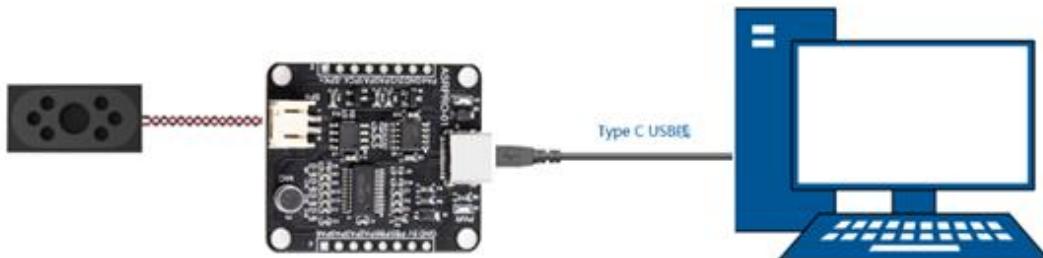
| 类型 | 识别词 | 回复语音 |
|------|-------------------|----------|
| 欢迎词 | 欢迎使用智能管家，用智能管家唤醒我 | |
| 退出语音 | 我退下了，用智能管家唤醒我 | |
| 唤醒词 | 智能管家 | 我在 |
| 命令词 | 打开灯光 | 好的，灯光已打开 |
| 命令词 | 关闭灯光 | 好的，灯光已关闭 |

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。

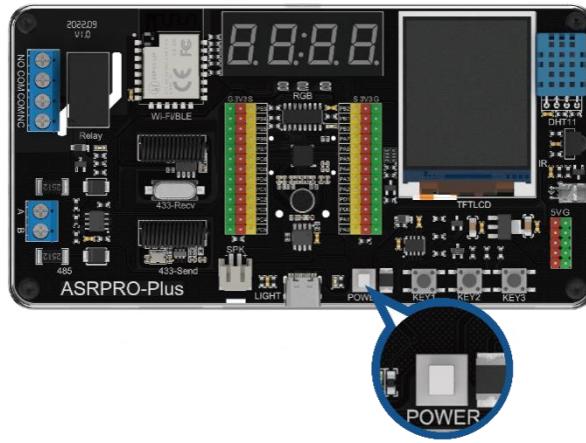
| 类型 | 识别词 | 回复语音 |
|------|-------------------|----------|
| 欢迎词 | 欢迎使用智能管家，用智能管家唤醒我 | |
| 退出语音 | 我退下了，用智能管家唤醒我 | |
| 唤醒词 | 智能管家 | 我在 |
| 命令词 | 打开灯光 | 好的，灯光已打开 |
| 命令词 | 关闭灯光 | 好的，灯光已关闭 |

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，用天问五么唤醒我”，接下来你可以用“天问五么”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 | 备注 |
|-----|---------|---------------|---------------------------------------|
| 唤醒词 | 天问五么 | 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 | 议) |
| 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 | |

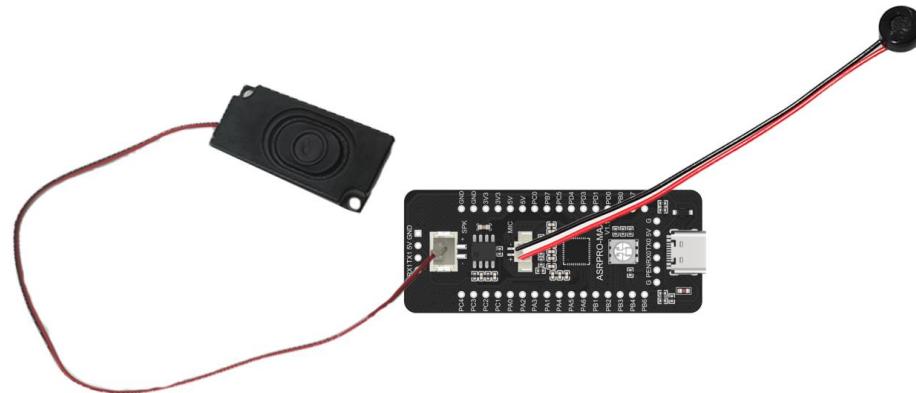
第四步：板载按键控制说明如下表

| 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 |

ASRPRO-MAX 开机测试

第一步：连接麦克风和喇叭到开发板上

注意麦克风极性，红色线为正，黑色线为负，不要插反。



第二步：用 STC-Link 连接 ASRPRO-MAX 到电脑，给开发板供电。



第三步：上电后，会播报欢迎词“欢迎使用语音助手，用天问五么唤醒我”，接下来你可以用“天问五么”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，用天问 | |

| | | |
|------|-------------------|---------|
| | 五么唤醒我 | |
| 退出语音 | 我退下了，用天问五么唤醒 我 | |
| 唤醒词 | 天问五么 | 我在 |
| 命令词 | 打开灯光 | 好的，马上执行 |
| 命令词 | 关闭灯光 | 好的，马上执行 |

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/。](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



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。

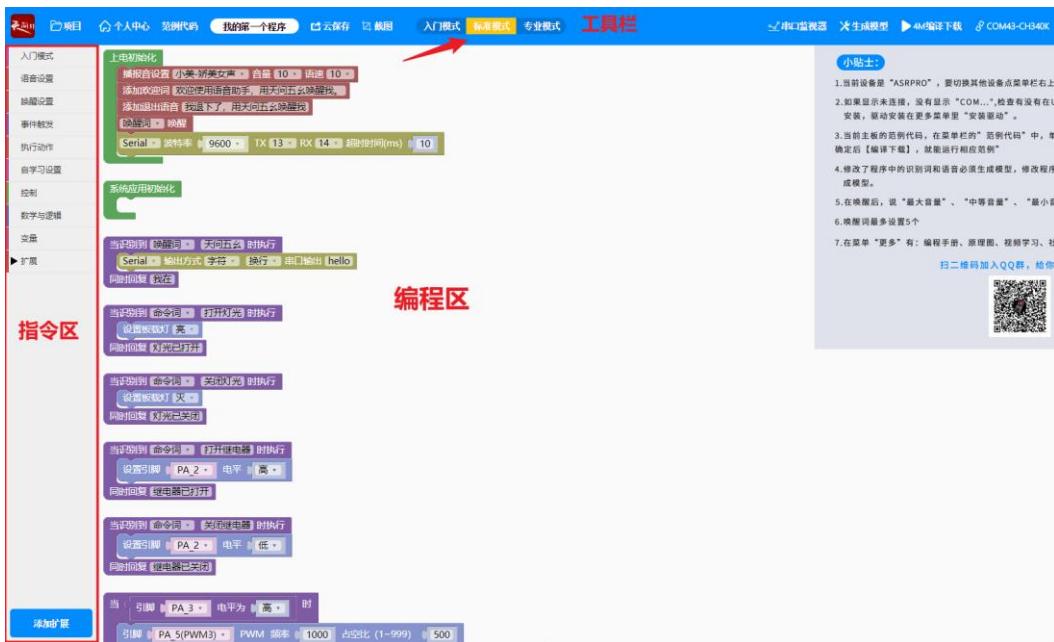


选择开发模式

本教程是标准模式教程，因此需要切换成标准模式



界面说明



在标准模式下，页面总共分为3个部分，工具栏、指令区和编程区。

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：包含了标准模式的基本指令，还可以添加扩展

编程区：将图形化指令拖拽至编程区进行合理修改组合编程

五、运行程序

打开范例程序

打开范例代码 1.1 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

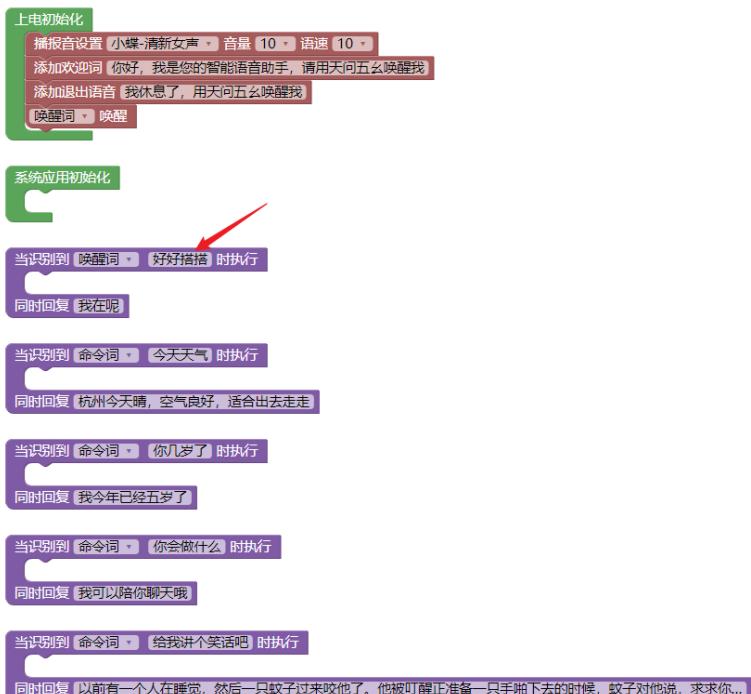
范例代码都已经生成模型，直接点击编译下载即可。





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

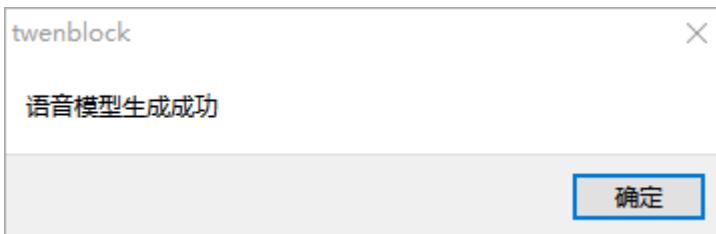


在更多中点击实名认证，输入手机号码以及验证码即可实名成功，注意一个号码只能绑定一个账号



生成模型与编译下载

点击生成模型



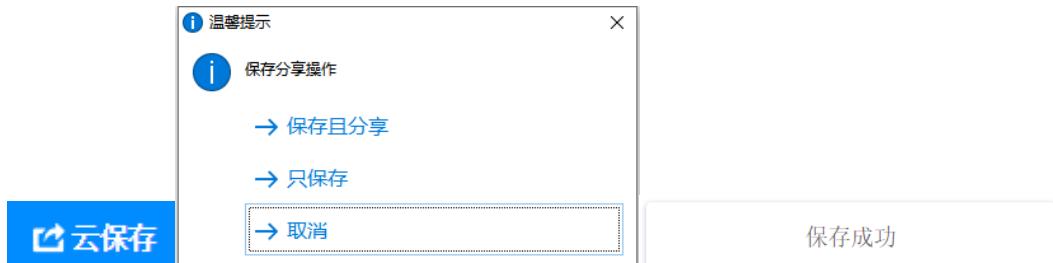
点击编译下载



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

项目中心

官方例程 最新项目 我的项目 搜索

请输入作品名称

ASRPRO 1.智能语音对话 ① 2023-01-05 分享

ASRPRO 作业 ① 2023-01-03 分享

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 开发板展开学习！

范例 1.1 智能语音对话

一、范例功能

本范例通过学习如何修改语音模型（唤醒词、命令词及回复词），实现智能的语音对话功能，达到用户可以自定义对话内容的目的。

二、范例分析

语音设置
包括音色、音量、语速、
欢迎词、退出语音、唤醒方式

语音识别触发事件设置
包括唤醒词、命令词及其回复词的设置

| 当识别到 | 时执行 |
|-------------|--|
| 唤醒词 天问五么 | 同时回复 我在呢 |
| 命令词 今天天气 | 同时回复 杭州今天晴，空气良好，适合出去走走 |
| 命令词 你几岁了 | 同时回复 我今年已经五岁了 |
| 命令词 你会做什么 | 同时回复 我可以陪你聊天哦 |
| 命令词 给我讲个笑话吧 | 同时回复 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... |

三、涉及指令讲解

➤ 控制相关



主要用于上电后的硬件或语音的初始化设置。

➤ 语音设置相关

播报音设置 小蝶-清新女声 音量 10 语速 10

1. 用于设置语音播报的音色、音量和语速，数值越高，音量越高、语速越快，下拉可进行选择，语音识别的相关注意事项详见附录一。

添加欢迎词 欢迎使用好搭助手，用天问五么唤醒我。

2. 用于设置上电后自动播报的欢迎词内容。

添加退出语音 我退下了，用天问五么唤醒我

3. 用于设置退出工作状态时自动播报的退出语音内容。

唤醒词 唤醒

4. 用于设置唤醒方式，支持唤醒词唤醒和永远唤醒两种模式。

- 唤醒词唤醒模式：需要用户先喊唤醒词唤醒模块，让模块进入工作状态，再喊对应的命令词，模块才会识别命令词。
- 永远唤醒模式：即表示模块一直处于工作的状态，随时喊命令词都能识别。（具体使用见范例 2.1）

➤ 事件触发相关

当识别到 唤醒词 天问五么 时执行

同时回复 我在

1. 当识别到相应唤醒词时，内部可添加执行动作指令，同时语音进行回复，并进入工作状态，可切换识别词的类型（唤醒词或命令词），修改识别词内容，以及语音回复的内容。

当识别到 命令词 打开灯光 时执行

同时回复 灯光已打开

2. 当识别到相应命令词时，内部可添加执行动作指令，同时语音进行回复，可切换识别词的类型（唤醒词或命令词），修改识别词内容，以及语音回复的内容。

四、范例程序讲解

要实现自定义的语音对话，可以在范例 1.1 的基础上进行修改，主要是修改语音识别触发事件的指令参数。

1. 修改/添加唤醒词及回复词

我们可以在范例的基础上修改“天问五么”为其他唤醒词，或者添加新的唤醒词事件指令，修改唤醒词为“智能管家”、“小爱同学”等等你喜欢的名称；同时其回复的语音内容也可以进行修改（可以为空）。

如下设置唤醒词为“智能管家”，语音回复“你好”，设备并进入工作的状态。

当识别到 唤醒词 ▾ 智能管家 时执行

同时回复 你好

注意：在上电初始化中，欢迎词、退出语音需要修改唤醒词的提示。

2. 修改/添加命令词及回复词

修改/添加命令词触发事件指令参数，如下设置命令词为“打开灯光”，当识别到该命令词时，会自动回复“好的，马上打开灯光”（在后续的范例中会讲解如何添加执行动作，实现灯光的语音控制）。

当识别到 命令词 ▾ 打开灯光 时执行

同时回复 好的，马上打开灯光

另外，你还添加更多的命令词和相应的语音回复，可以实现更多有趣的语音对话。

注意：需在使用唤醒词唤醒设备后，设备从待机状态进入工作状态，才可以使用命令词进行对话。

五、下载运行

1.直接下载范例

直接运行范例代码（已带语音模型）可直接点击“编译下载”下载程序。

2.生成模型下载

如果涉及到语音的更改，则需要重新生成模型（需提前登陆注册账号，并实名认证），再点击编译下载。

1 ✖ 生成模型 2 ➡ 4M 编译下载

范例 1.2 语音控制板载 LED

一、范例功能

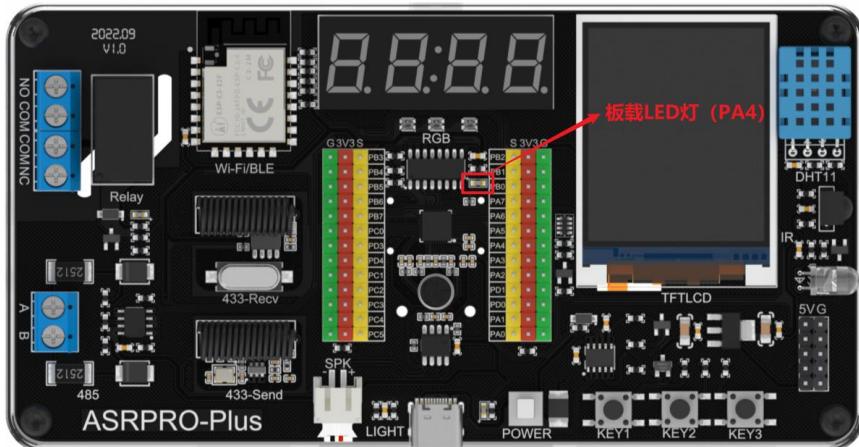
本范例通过学习如何添加命令词的执行动作，实现语音控制板载 LED 灯的功能，达到用户可以用语音同时控制单个设备的目的。

二、范例分析

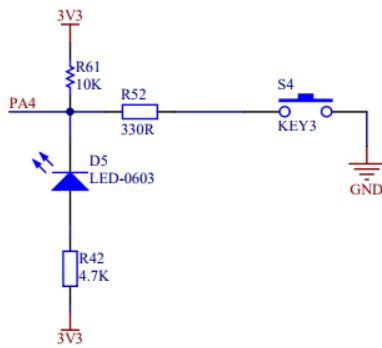


三、电路实物说明

ASRPRO-Plus 板载灯所处位置如下图所示：



其内部电路原理图如下图所示：



由上图我们可以看到 LED 的正极连接 3V3 的电源，只有当 PA4 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 PA4 引脚输出高电平时，没有电流通过，LED 熄灭。

四、涉及指令讲解

➤ 执行动作相关

设置板载灯 亮

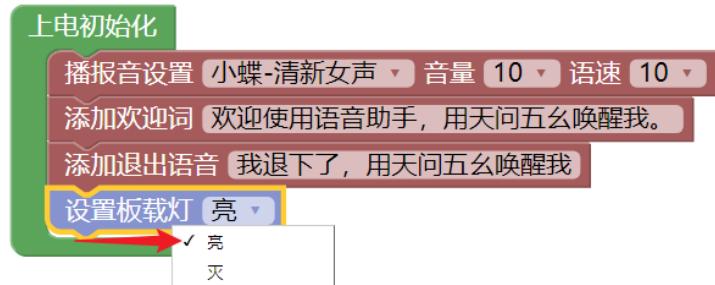
用于设置板载灯的状态为亮还是灭，下拉可进行选择。

五、范例程序讲解

要添加自定义命令词的执行动作，可以在范例 1.2 的基础上进行修改，主要要设置板载灯的初始状态，以及不同命令词触发执行的动作。

1. 修改板载灯初始状态

如果想要在开机时灯是打开的状态，在范例中可以修改板载灯的初始状态为亮；否则就设置常灭。



2. 命令词触发事件设置

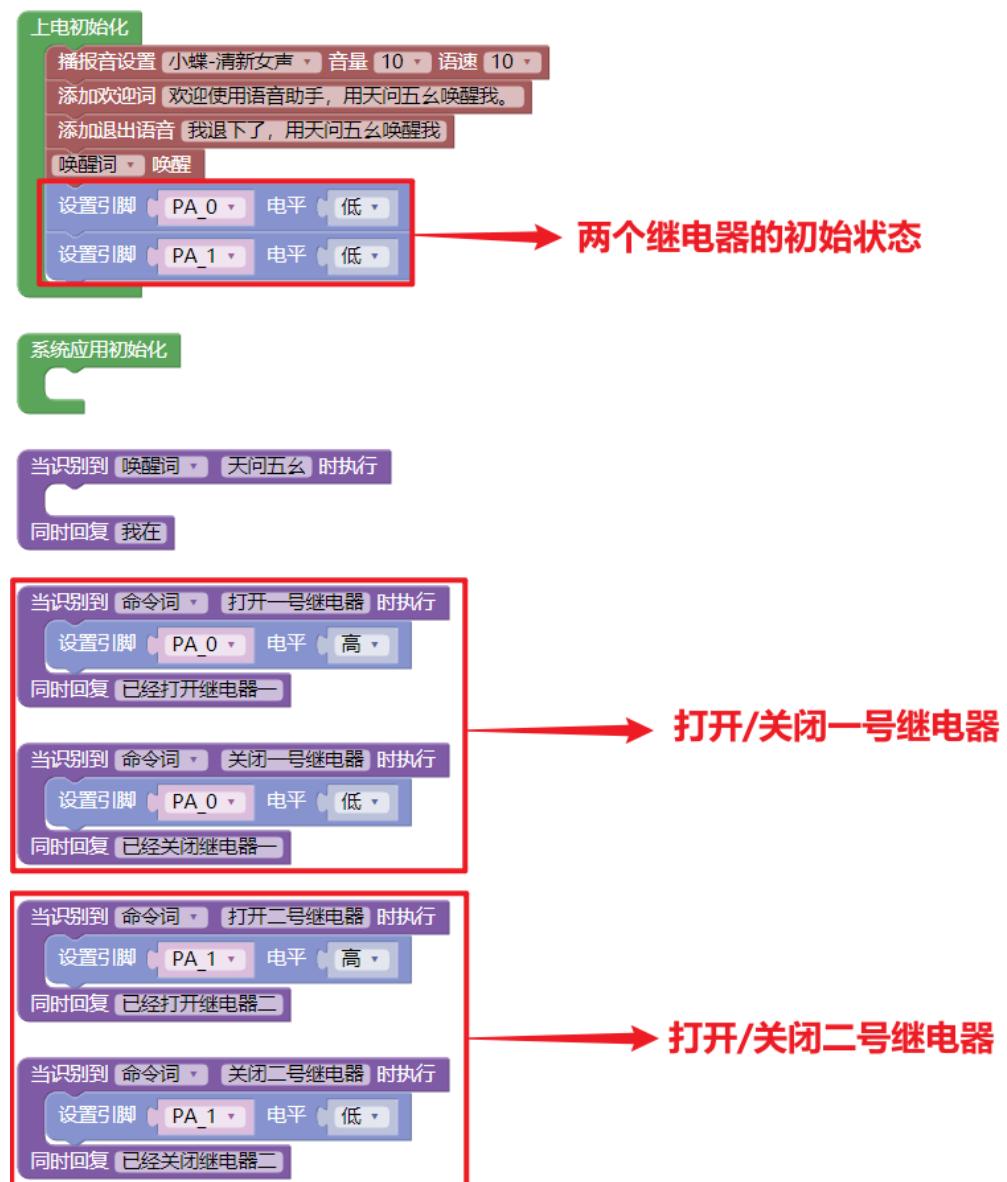
在命令词触发事件内部，可以添加相应的执行动作，即识别到该命令词时，做什么事情。本范例只是讲解如何通过语音控制板载的灯光，在接下来的范例学习中，可以了解如何添加更多的执行动作（如继电器、彩屏等等）。

范例 1.3 语音控制继电器

一、 范例功能

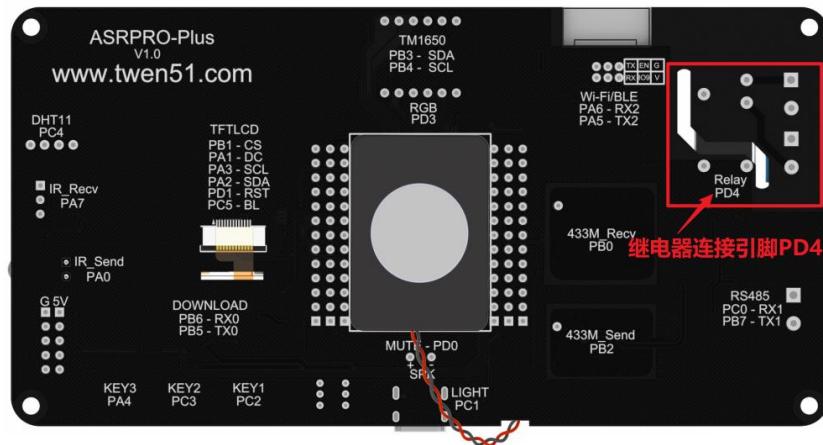
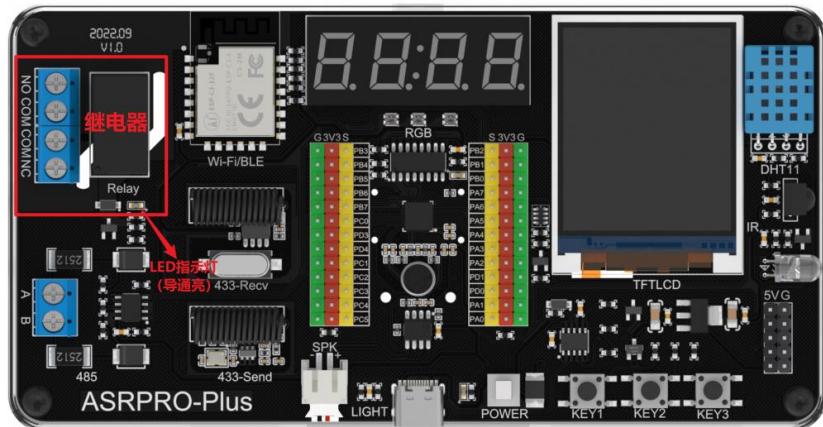
本范例通过学习如何添加继电器的执行动作，实现语音控制继电器的功能，达到用户可以用语音同时控制多个设备的目的。

二、 范例分析



三、 电路实物说明

ASRPRO-Plus 板载继电器所处位置如下图所示：



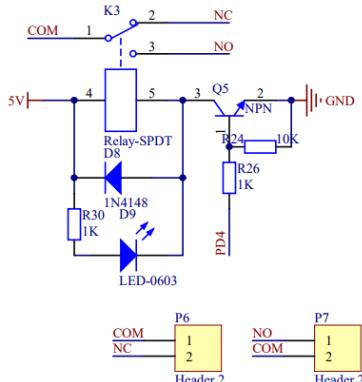
继电器与家用灯泡可参考下图进行连接：



由上图我们可以知道在 ASRPRO-Plus 开发板中有一个继电器，与其连接的引脚为 PD4。

其内部电路原理图如下图所示：

RELAY



左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。

总而言之，ASRPRO-Plus 开发板载继电器为高电平触发，输入低电平时电路断开，指示灯灭，输入高电平时电路连通，指示灯亮。在范例中，我们设置继电器的初始电平为低电平，并用语音控制继电器的开/关。

四、涉及指令讲解

➤ 执行动作相关



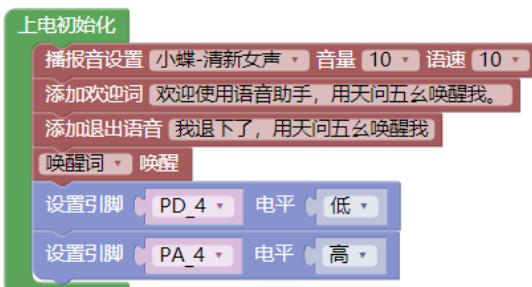
属于执行动作类别下的指令，用于设置指定引脚的高/低电平，下拉可进行选择。

五、范例程序讲解

在范例 1.3 的基础上进行修改，我们可以实现语音控制多个设备的功能（如外接另一个继电器、板载的灯光、彩屏等）。

1. 修改并添加引脚初始化设置

根据继电器和板载灯的连接引脚，修改引脚 PD4 为低电平，即开机时继电器断开；添加设置板载灯 PA4 引脚为高电平，即开机时灯是灭的状态。（你也可以根据你实际连接外设的引脚，修改其初始的电平状态）



2. 命令词触发单个执行动作

根据继电器的连接引脚，修改“打开一号继电器”命令词下执行动作的控制引脚为 PD4，同“关闭一号继电器”，就可以控制 ASRPRO-Plus 的板载继电器。

根据板载灯的连接引脚，修改“打开二号继电器”命令词及其回复词，并修改其执行动作的控制引脚和电平，同“关闭二号继电器”，就可以控制 ASRPRO-Plus 的板载灯光。



3. 命令词同时触发执行多个动作

添加新的命令词触发事件指令，在同一命令词下，实现多个设备的同时控制（如用命令词“打开所有”、“关闭所有”，实现同时打开/关闭继电器和板载灯）。

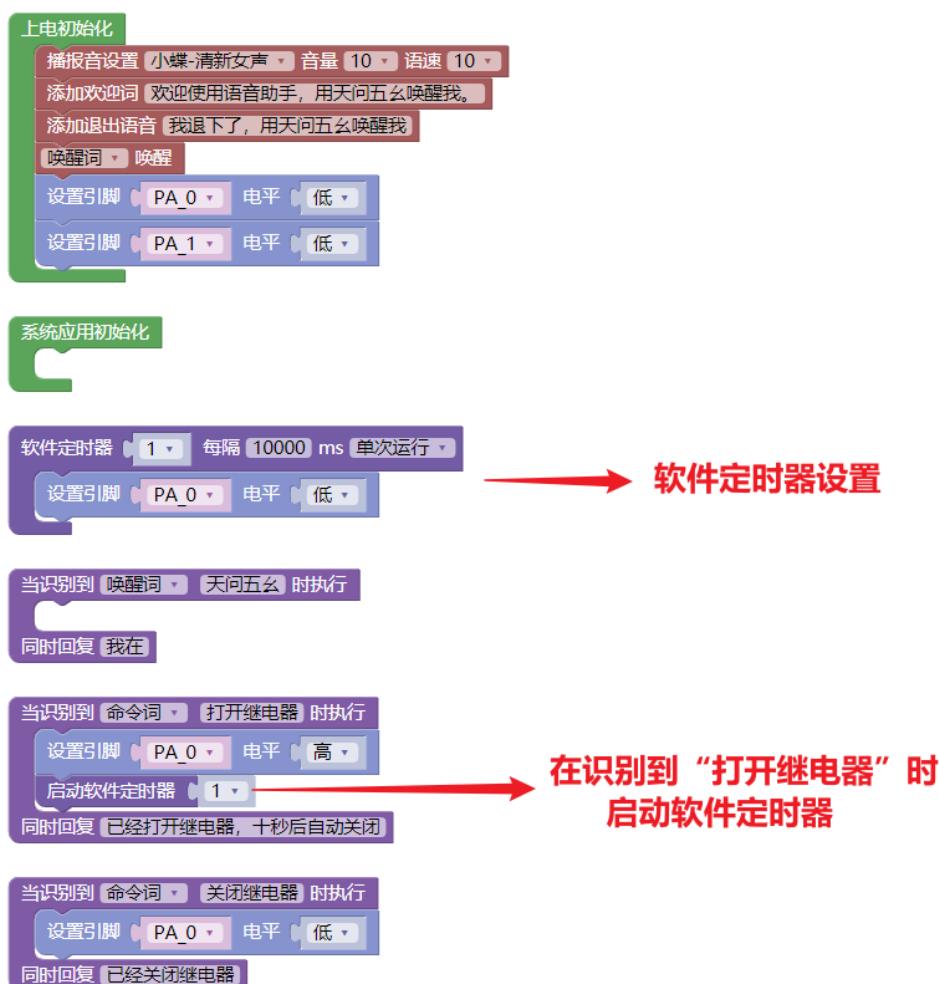


范例 1.4 语音控制继电器延时关闭

一、 范例功能

本范例通过学习如何使用软件定时器，实现语音控制继电器延时关闭的功能，达到用户可以用软件定时器实现延时控制的目的。

二、 范例分析



三、 涉及指令讲解

➤ 事件触发相关

1. 用于创建软件定时器（图形化中最多设置八个，需要设置更多可以切换字符模式编程实现）。软件定时器是基于系统 Tick（操作系统的最基本时间单位）时钟中断且由软件来

模拟的定时器。当经过设定的时间后，会触发用户自定义的事件（下拉可选择单次运行/重复运行该事件）。

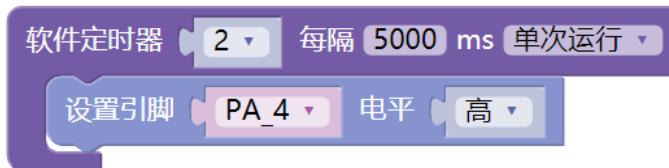
2.  用于启动软件定时器（下拉可选择定时器序号），注意先创建再启动定时器。

四、范例程序讲解

在范例 1.4 的基础上进行修改，我们还可以实现延时控制其他外设的功能（如灯光、彩屏等），下面以实现延时灯为例编写程序。

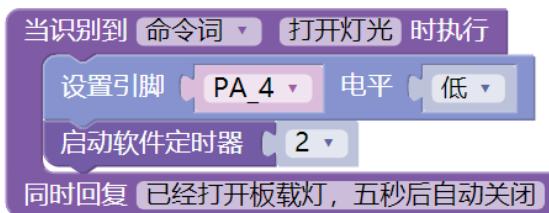
1. 创建软件定时器事件

创建新的软件定时器事件，序号修改为 2，如要实现打开灯光后 5 秒自动关闭的效果，则需要修改为每隔“5000”ms，单次运行设置引脚 PA4 电平为高，即启动后 5 秒关闭板载灯。（如需实现灯的自动闪烁，具体想要了解“重复运行”的应用，可参考专业模式的范例 1.6）



2. 根据需要启动软件定时器

创建新的命令触发事件，当识别到“打开灯光”时，设置 PA4 引脚为低，并启动软件定时器 2，就可以实现打开灯光 5 秒后自动熄灭的效果。（因为在标准模式只能设置单次运行软件定时器的事件，无需用到停止软件定时器的功能，具体想了解其使用的方法，可参照专业模式的范例 1.6）



注意：在“上电初始化”中添加设置引脚 PA4 初始电平状态为高。

范例 1.5 语音和按钮同时控制继电器

一、 范例功能

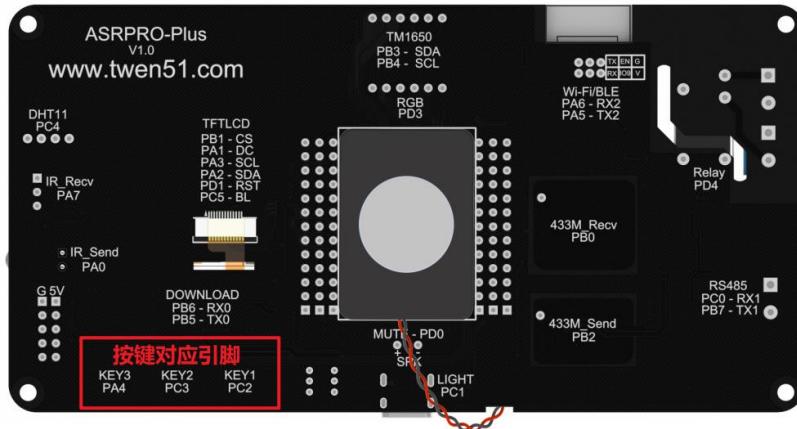
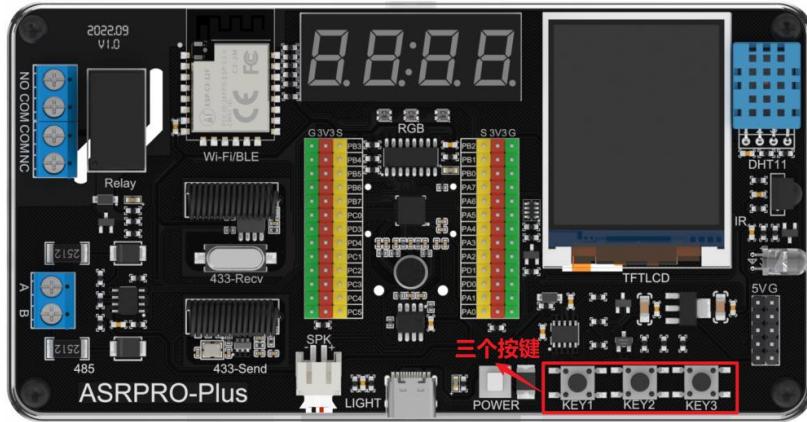
本范例通过学习如何使用按键，实现语音和按钮同时控制继电器的功能，达到用户可以综合运用多种方式控制设备的目的。

二、 范例分析



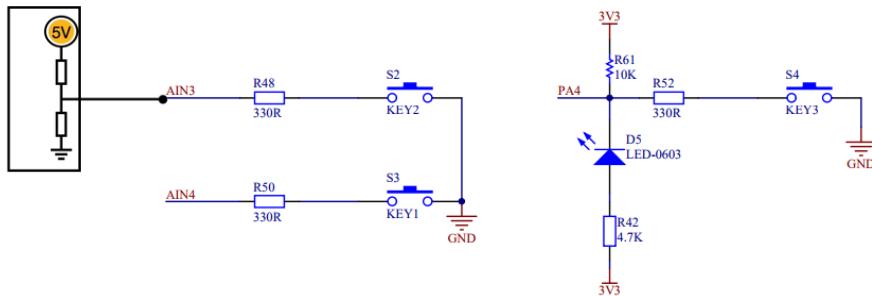
三、 电路实物说明

板载按键所处位置如下图所示：



内部电路原理图如下图所示：

KEY



由图我们可以看到 ASRPRO-Plus 板载按键 KEY1 和 KEY2 在未按下时，处于悬空输入的状态，随机输入电平，不稳定（在专业模式中可编程添加上拉电阻输入高电平），上拉电阻可以将一个不确定的信号钳位在高电平，在单片机内部的每个引脚上都独立连接上下拉电阻。

当引脚处于开漏输出时，引脚内部的 MOS 永远是断开的，相当于连接着一个无穷大的电阻，如果我们想让它输出高电平，则需要把下面的 MOS 截止，这样相当于引脚连接着两个无穷大的电阻，这两个电阻就相当于断开，所以他输出的信号是不确定的，但当我们给他加一个上拉电阻之后，他就能输出高电平了。

如果想要在标准模式下使用 KEY1 和 KEY2 实现对其他设备的控制，用户可以在按键外围自行添加上拉电阻（4.7k 到 10k）。

在本范例中想要实现按键控制继电器的功能，我们只能使用 KEY3 按键，当按键按下

时，输入低电平，未按下时会输入高电平。

四、涉及指令讲解

➤ 事件触发相关



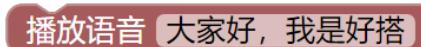
用于引脚电平触发的事件设置，下拉可选择引脚和电平，程序会自动判断对应引脚的电平，条件成立时会自动触发执行框里面的程序模块。如当按键 KEY3 按下时执行内部的程序，则需修改引脚为 PA4，电平为低。

➤ 唤醒设置相关



用于马上唤醒设备，并经过设定的时间后自动退出。一般不放在初始化中，需要放在具体的某个事件中。

➤ 执行动作相关



用于播报指定的语音内容，使用的时候要在唤醒状态才会执行，也可以在前面添加唤醒模块来实现自动唤醒播报。

五、范例程序讲解

根据继电器的引脚位置，在范例 1.5 的基础上进行修改，根据 KEY3 按键是否按下，控制继电器引脚的电平变化，同时触发语音播报。

修改引脚电平触发事件

修改引脚为 PA4，电平为低，即代表按键按下时，打开继电器，即设置引脚 PD4 为高电平，再添加马上唤醒功能，并自动播报“电平控制继电器打开”（不先唤醒的话无法进行播报）。



当按键没有被按下的时候，就是当引脚 PA4 电平为高时，设置引脚 PD4 电平为低，继电器关闭；在这里我们需要删除范例中“马上唤醒”和“播放语音”的指令，因为按键未按下是常态，如果未删除，则在上电后会一直播报“电平控制继电器关闭”的语音，不适用于正常的应用场景。



范例 1.6 串口输出字符串

一、范例功能

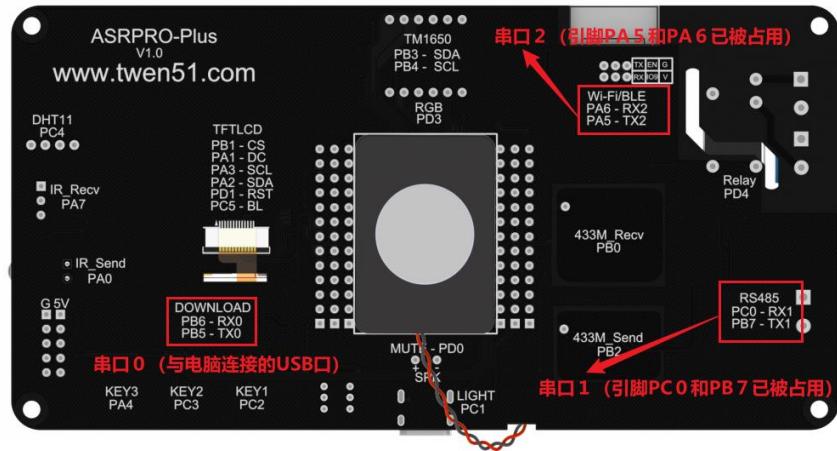
本范例通过学习如何使用串口自动发送字符串数据，实现串口输出字符串的功能，达到用户可以使用串口发送数据的目的。

二、范例分析



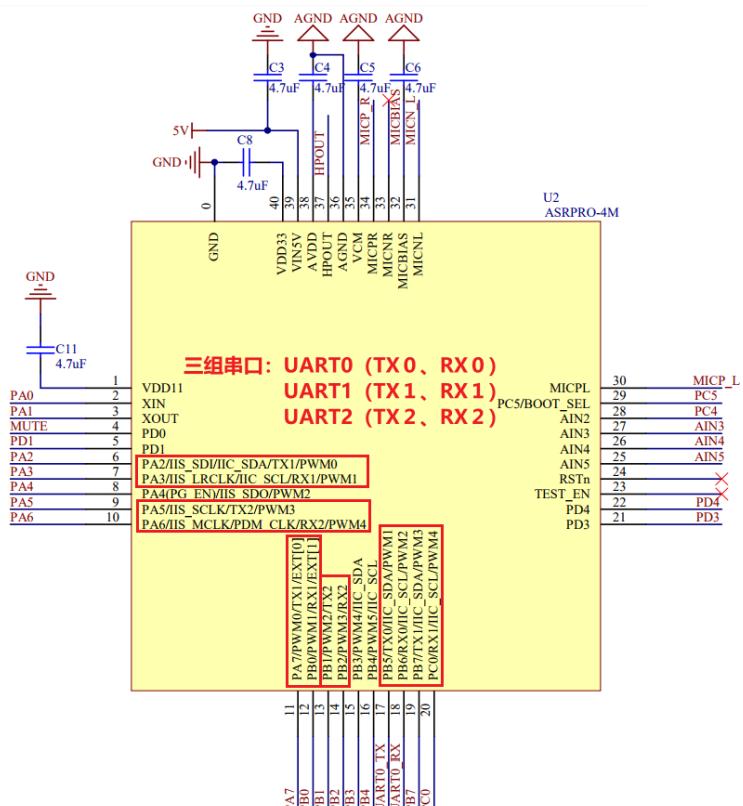
三、电路实物说明

串口所处位置如下图所示：



由上图我们可以看到具有串口通讯功能的的 PA5、PA6、PC0 和 PB7 引脚已分别与 WIFI 模块、RS485 模块连接，如果要查看串口通讯的数据，可以使用烧写器连接其他具有串口通讯功能的引脚（如果 RS485 未接设备，可使用 PC0 和 PB7 引脚进行串口通讯），并借助串口监视器或 STC-ISP 软件查看不同串口的通讯数据。

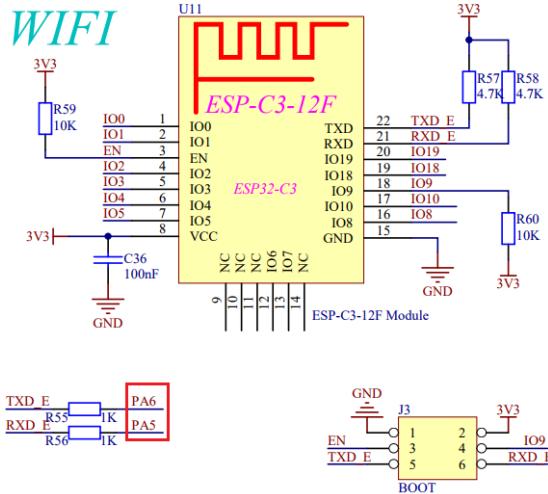
芯片内部电路原理图如下图所示：



由图我们可以看到 ASRPRO-Plus 开发板的芯片共有三组串口，分别是 UART0、UART1 和 UART2。UART 是最常见的串行通讯，广泛应用于单片机和单片机之间通讯。比如 WiFi 模块，串口液晶屏等。串口通信经过信号转换，可以进行 RS232、RS422、RS485 通信，广泛应用于设备之间远程通信。

串口通讯采用 2 条通讯线：发送数据线 TX，接收数据线 RX。要实现不同设备之间的数据收发，需要将通讯设备 1 的 TX 与通讯设备 2 的 RX 相连，将通讯设备 1 的 RX 与通讯设备 2 的 TX 相连，就可以同时进行数据的发送和接收。

WIFI 通讯模块的内部原理图如下：



根据上图我们可以看到在 ASRPRO-Plus 中，PA5、PA6 引脚已用于 WIFI 通讯，无法使用这两个引脚进行串口通讯。

另外，如果 RS485 未接设备，可使用 PC0 和 PB7 引脚进行串口通讯。

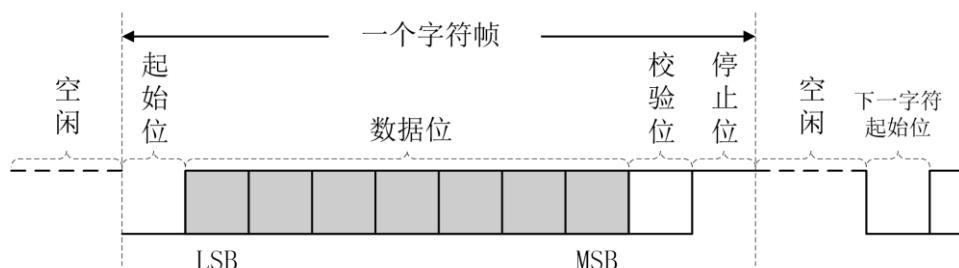
在本范例中，我们需要设置三组串口的引脚位置，串口 0 的位置已经固定（PB5 和 PB6），串口 1 和串口 2 的位置可以通过编程修改确定（避开已被占用引脚）。

四、涉及指令讲解

➤ 执行动作相关

1. **Serial ▾ 输出方式 16进制 ▾ 不换行 ▾ 串口输出 FF 1F 34 AF**
用于指定串口输出支持 16 进制和字符两种输出方式，支持换行和不换行输出。其中多个 16 进制数据，用空格隔开（在范例 1.7 会具体讲解）。
2. **Serial ▾ 波特率 9600 ▾ TX PB_5 ▾ RX PB_6 ▾ 超时时间(ms) 10**
用于设置对应串口的波特率，其中超时时间为用来判断数据是否接收完毕，默认 10ms。

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，比较常用的波特率有 9600, 57600, 115200 等等。

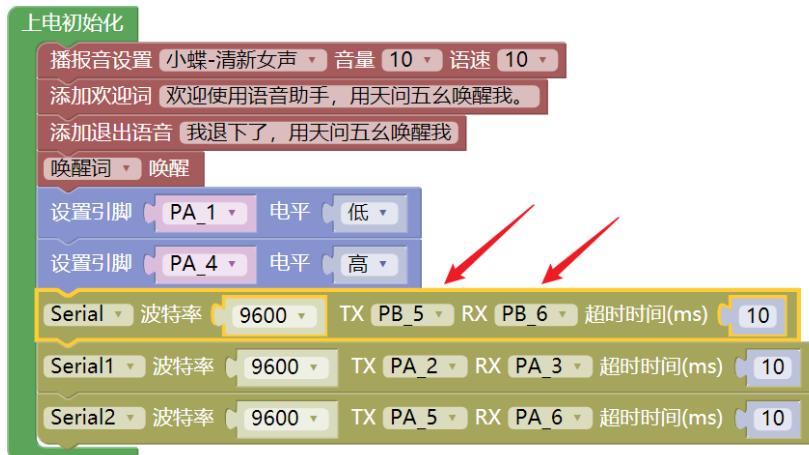


五、范例程序讲解

根据串口在 ASRPRO-Plus 的实际位置，在范例 1.6 的基础上进行修改，分别查看串口 0 和串口 1（串口 2 操作方式同串口 1）发送的数据。

1. 修改串口 0 设置并查看发送的字符串

修改 Serial 的 TX 和 RX 分别为 PB5 和 PB6（固定），波特率的设置可根据需要修改（一般常用 9600 和 115200）。



当说出“打开一号继电器”的命令词时，串口 0 会输出指定字符串（字符串中除了可以包含数字字符、字母字符或特殊字符外，还可以包含转义字符），你可自行修改串口输出的内容。



通过左上角的串口监视器来查看串口 0 输出的字符串。

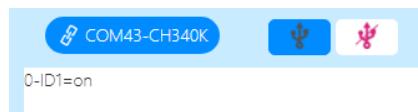
将程序下载到 ASRPRO-Plus 后，打开串口监视器（软件界面右上方）。



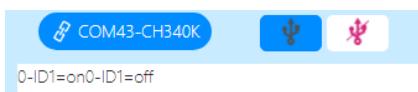


首先打开串口，修改波特率为程序中设定的波特率数值，接下来就可以用命令词触发串口发送字符串了。

如下图唤醒设备后，说出命令词“打开一号继电器”后，串口 0 接收数据显示“0-ID1=on”。

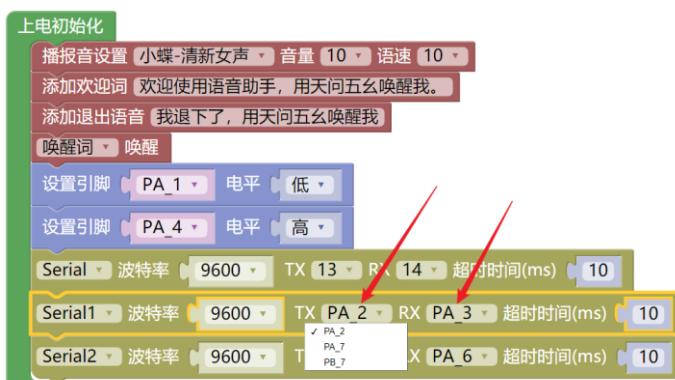


说出命令词“关闭一号继电器”后，串口 0 接收数据显示“0-ID1=off”。



2. 修改串口 1 设置并查看发送的字符串

修改 Serial1 的 TX 和 RX 为可用引脚，波特率的设置可根据需要修改（一般常用 9600 和 115200）。



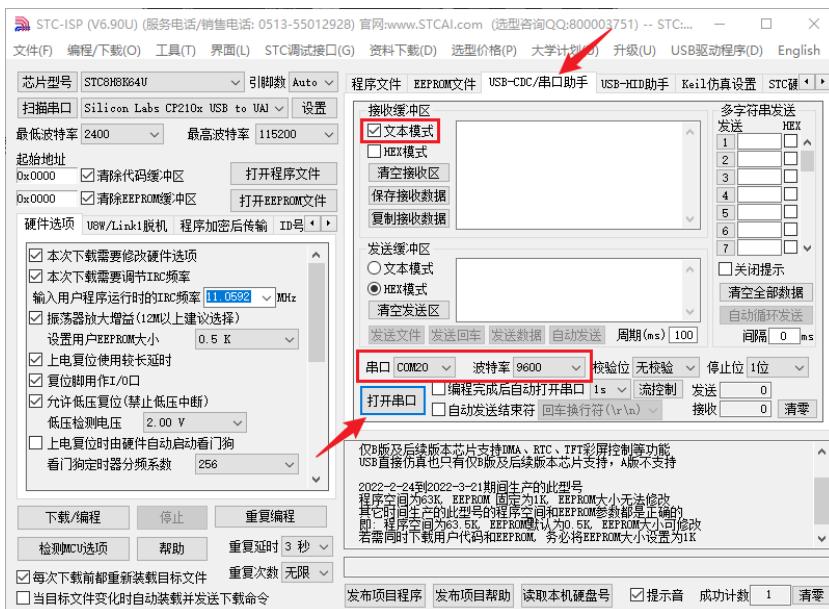
下载程序到 ASRPRO-Plus 后，你可以选择连接其他设备与 ASRPRO-Plus 进行串口通讯，以方便查看串口 1 的发送数据。下面以连接 STC-LINK 烧写器与 ASRPRO-Plus 为例，根据初始化中设定的引脚，硬件连接如下图所示：



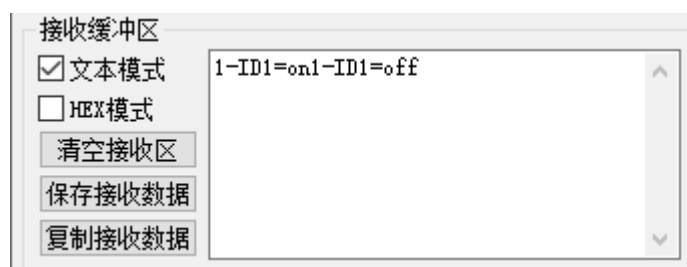
将 STC-LINK 连接到电脑，通过设备管理器查看连接的端口号是多少。由下图我们可以看到端口号为 COM20（如果多设备连接，可通过插拔连接口，查看不同设备对应的端口号）。



打开 STC-ISP 软件或打开软件的串口监视器，切换端口为 COM20（根据在设备管理器查看到的实际端口号进行修改）。下面以 STC-ISP 软件操作为例，修改波特率，选择接收数据模式为文本模式，最后点击“打开串口”。



ASRPRO-Plus 维持供电的状态（可接在电脑的另一个 USB 口），通过唤醒设备后说出命令词“打开一号继电器”、“关闭一号继电器”，在 STC-ISP 软件界面中可以看到串口 1 的接收数据。



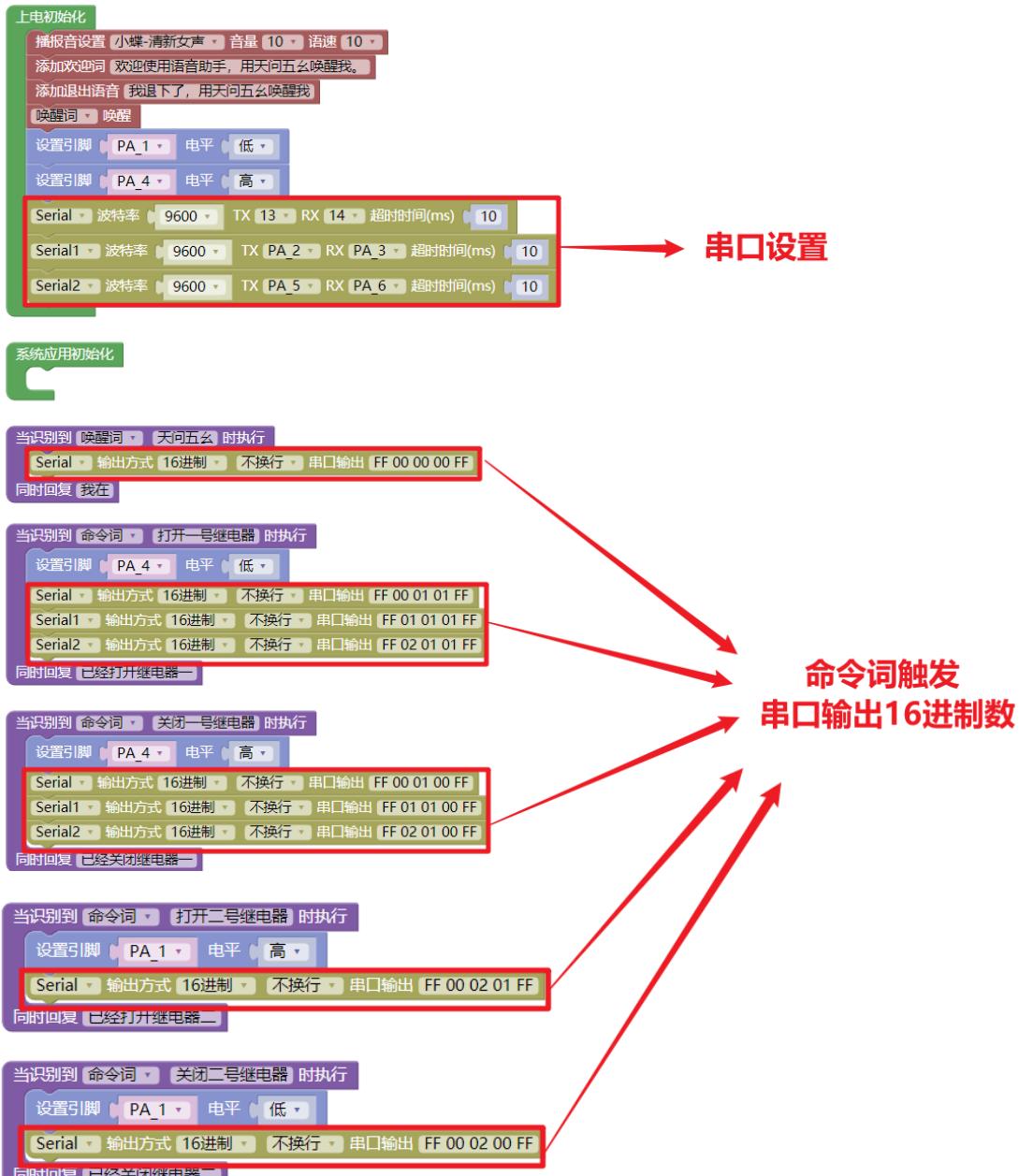
查看串口 2 发送字符串的具体操作方式同上。

范例 1.7 串口输出十六进制

一、范例功能

本范例通过学习如何使用串口自动发送十六进制数据，实现串口输出十六进制数的功能，达到用户可以使用串口发送数据的目的。

二、范例分析



三、范例程序讲解

根据串口在 ASRPRO-Plus 的实际位置，在范例 1.7 的基础上进行修改，分别查看串口 0 和串口 1（串口 2 操作方式同串口 1）发送的数据。

1. 修改串口 0 设置并查看发送的十六进制数

修改 Serial 的 TX 和 RX 分别为 PB5 和 PB6（固定），波特率的设置可根据需要修改（一般常用 9600 和 115200）。

当说出“打开一号继电器”的命令词时，串口 0 会输出指定十六进制数，你可自行修改串口输出的内容（多个十六进制数输入时，需用空格隔开）。

十六进制数的基数是 16，采用的数码是 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F。其中 A-F 分别表示十进制数字 10-15，十六进制数的技术规则是“逢十六进一”。

| 十六进制数字 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| 十进制值 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

具体可参照 ASCII 对照表（以下截取部分），输入想要发送的十六进制数。

| 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 |

注意：图形块里不用写 0x 前缀，软件自动处理好了，方便用户快速输入。

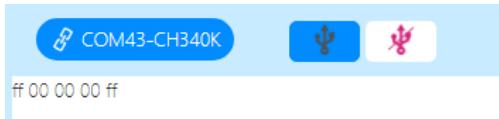
通过左上角的串口监视器来查看串口 0 输出的十六进制数。

将程序下载到 ASRPRO-Plus 后，打开串口监视器（软件界面右上方）。

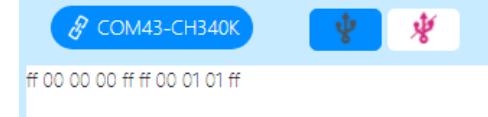


打开串口，修改波特率，并注意勾选“十六进制显示”，接下来就可以用命令词触发串口发送十六进制数了。

如下图唤醒设备后，串口 0 接收数据的显示“ff 00 00 00 ff”。



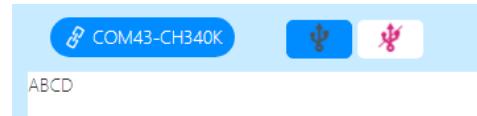
接着说出命令词“打开一号继电器”后，串口 0 接收数据显示“ff 00 01 01”，你也可以修改命令词及其触发的输出十六进制数，再来查看串口 0 接收到的数据。



修改范例 1.7 唤醒词的触发事件，参照 ASCII 码表，A、B、C、D 对应的十六进制为 0x41、0x42、0x43，设置唤醒触发串口输出“41 42 43 44”，并重新下载程序。



打开串口监视器，无需勾选“十六进制显示”，可以看到说出唤醒词时，串口输出“ABCD”。



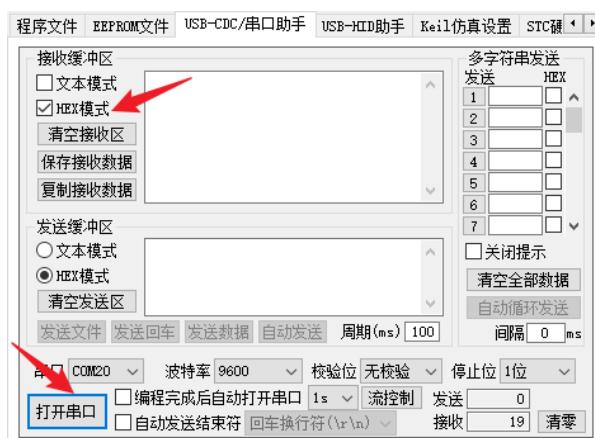
2. 修改串口 1 设置并查看发送的十六进制数

修改 Serial1 的 TX 和 RX 为可用引脚，波特率的设置可根据需要修改（一般常用 9600 和 115200）。

下载程序到 ASRPRO-Plus 后，你可以选择连接其他设备与 ASRPRO-Plus 进行串口通讯，以方便查看串口 1 的发送数据。

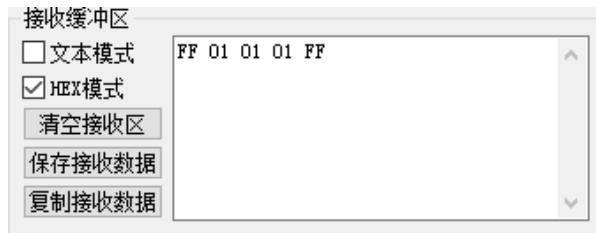
将需要通讯的设备连接到电脑，查看连接外设的端口号（设备管理器）。

借助 STC-ISP 软件（或软件右上方的串口监视器），根据在设备管理器查看到的实际端口号修改串口，并修改波特率，选择接收数据模式为 HEX 模式，最后点击“打开串口”。



以上具体操作可参考范例 1.6。

唤醒设备后，通过命令词“打开一号继电器”可查看串口 1 接收到的十六进制数。



查看串口 2 发送十六进制数的具体操作方式同上。

范例 1.8 串口和语音同时控制继电器

一、范例功能

本范例通过学习如何在串口输入并发送数据，并根据串口接收到的数据触发执行相应的事件，实现串口和语音同时控制设备的功能，达到用户可以使用串口通讯实现设备之间的通讯和控制的目的。

二、范例分析



三、涉及指令讲解

➤ 事件触发相关

当 Serial 接收到一帧数据为 “1” 时

1.

用于指定串口接收字符串并比对，如果一致，则自动触发内部的程序模块。

当 Serial 接收到一帧16进制数据为 FF 1F 34 AF 时

2.

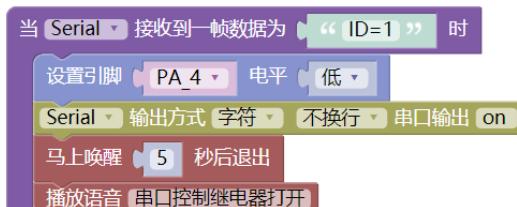
用于指定串口接收十六进制格式的数据并比对，如果一致，则自动触发内部的程序模块（支持多个数据同时比对，用空格隔开）。

四、范例程序讲解

根据串口在 ASRPRO-Plus 的实际位置，在范例 1.8 的基础上进行修改，修改 Serial 的 TX 和 RX 分别为 PB5 和 PB6（固定），波特率的设置可根据需要修改（一般常用 9600 和 115200）。

1. 串口输入并发送字符串

在范例中我们可以看到当串口接收到“ID=1”的字符串数据时，会自动输出“on”并控制相应的外设；当串口接收到“ID=0”的字符串数据时，会自动输出“off”并控制相应的外设。



下载程序到 ASRPRO-Plus 后，打开串口监视器，打开串口并设置波特率，注意不要勾选“十六进制发送”和“十六进制显示”。



在输入框输入“ID=0”，点击“发送”，串口接收会显示“off”，同时会触发相应的执行动作，语音唤醒并播报指定内容。



在输入框输入“ID=1”，串口接收会显示“on”，同时会触发相应的执行动作，语音唤醒并播报指定内容。



2. 串口输入并发送十六进制数

在范例中我们可以看到当串口接收到“FF 01 FF”的16进制数据时，会自动输出“01”并控制相应的外设；当串口接收到“FF 00 FF”的16进制数据时，会自动输出“00”并控制相应的外设。



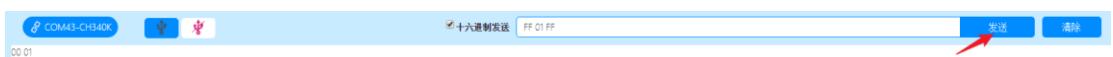
下载程序到 ASRPRO-Plus 后，打开串口监视器，打开串口并设置波特率，注意勾选“十六进制发送”和“十六进制显示”。



在输入框输入“FF 00 FF”，点击“发送”，串口接收会显示“00”，同时会触发相应的执行动作，语音唤醒并播报指定内容。



在输入框输入“FF 01 FF”，点击“发送”，串口接收会显示“01”，同时会触发相应的执行动作，语音唤醒并播报指定内容。



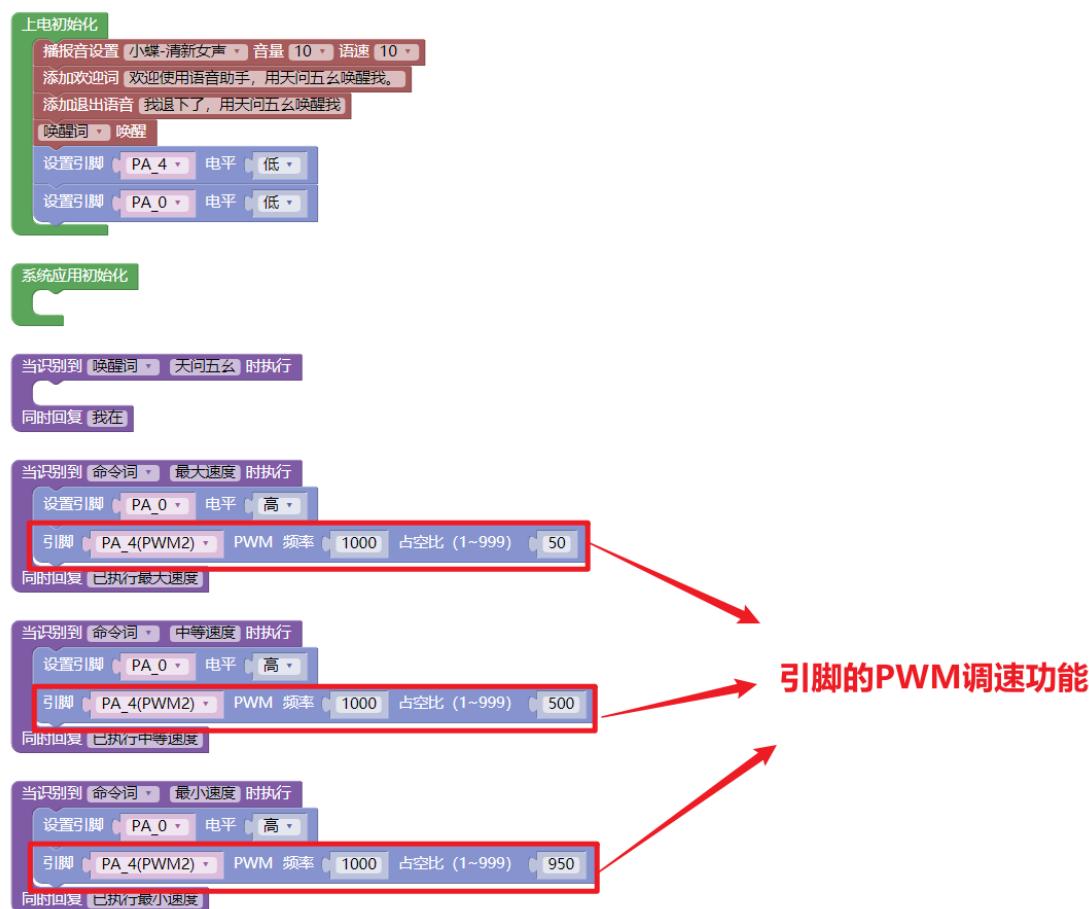
如想要使用其他串口外接设备，接收并处理另一个设备传送过来的数据，可以修改程序内的串口相关设置，并使用串口监视器或其他软件来查看串口通讯的具体数据（更多串口通讯案例见附录二）。

范例 1.9 语音控制单路电机驱动

一、范例功能

本范例通过学习如何使用引脚的 PWM 功能，实现指定引脚的脉冲宽度调制的功能，达到用户可以实现对设备的调速、调光等目的。

二、范例分析



三、电路及实物说明

ASRPRO-Plus 开发板芯片引脚及其功能如下：

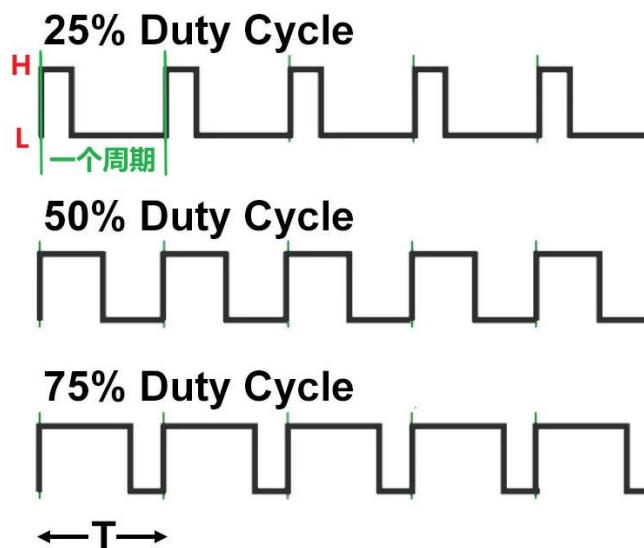
| Pin Name | Function1 | Function2 | Function3 | Function4 | Function5 | Analog Function | Specific Function |
|----------|-----------|-----------|-----------|------------|-----------|-----------------|-------------------|
| XIN | PA0 | PWM5 | - | - | - | XIN | - |
| XOUT | PA1 | - | - | - | - | XOUT | - |
| 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 |

由图我们可以看到具有 PWM 功能的引脚为 PA0, PA2 到 PA7, PB0 到 PB7, PC0 到 PC4, 共 6 路 PWM 通道 (PWM0 到 PWM5)。

PWM 就是数字对模拟信号高效的控制，可以实现电压大小的控制，被广泛应用于设备调速、调光等。

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，就是周期（单位为 s）的倒数，单位为 Hz (赫兹)。
- 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。
占空比=占空/脉冲周期×100%



举一个简单的例子去理解 PWM，如我们实行的是 8 小时工作制，24 小时就是周期，那么 8 小时就是脉宽，24 小时的倒数就是 PWM 的频率，占空比 1/3。

四、涉及指令讲解

➤ 执行动作相关



用于引脚输出 PWM，一般应用建议设置在 2M 以下，2M 以上需要修改底层驱动，不然会不准。占空比范围为 1 到 999。要注意芯片总共有 6 路 PWM 通道，可以分配到不同引脚。

五、范例程序讲解

以实现 PA4 板载灯的 PWM 调光功能为例，在范例 1.9 的基础上进行修改。

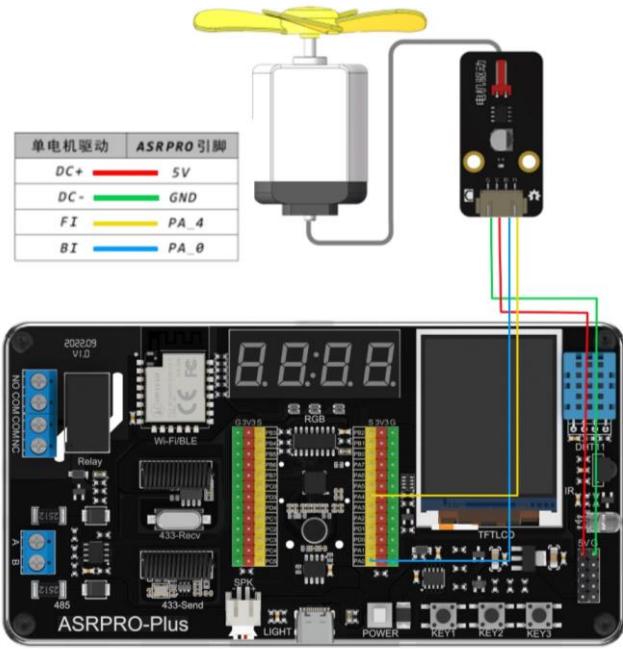
修改引脚的初始电平为高电平（高电平灭，低电平亮），即使其开机保持熄灭的状态。

要实现灯光亮度的三级变化，也就是修改占空比的数值，占空比的数值越大，高电平的时间占比越长，板载灯的亮度就越暗；反之，则越亮。

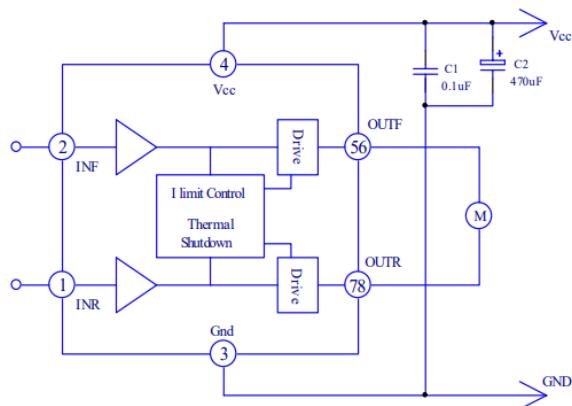
PWM 频率越大（赫兹为单位），响应越快。



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子用来控制电机前进、后退及制动。



IC 内部电路框图如上，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：高高、低低、高低、低高，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。

如下，在范例 1.9 的基础上进行修改，实现电机往相反的方向转动，即交换引脚的位置：

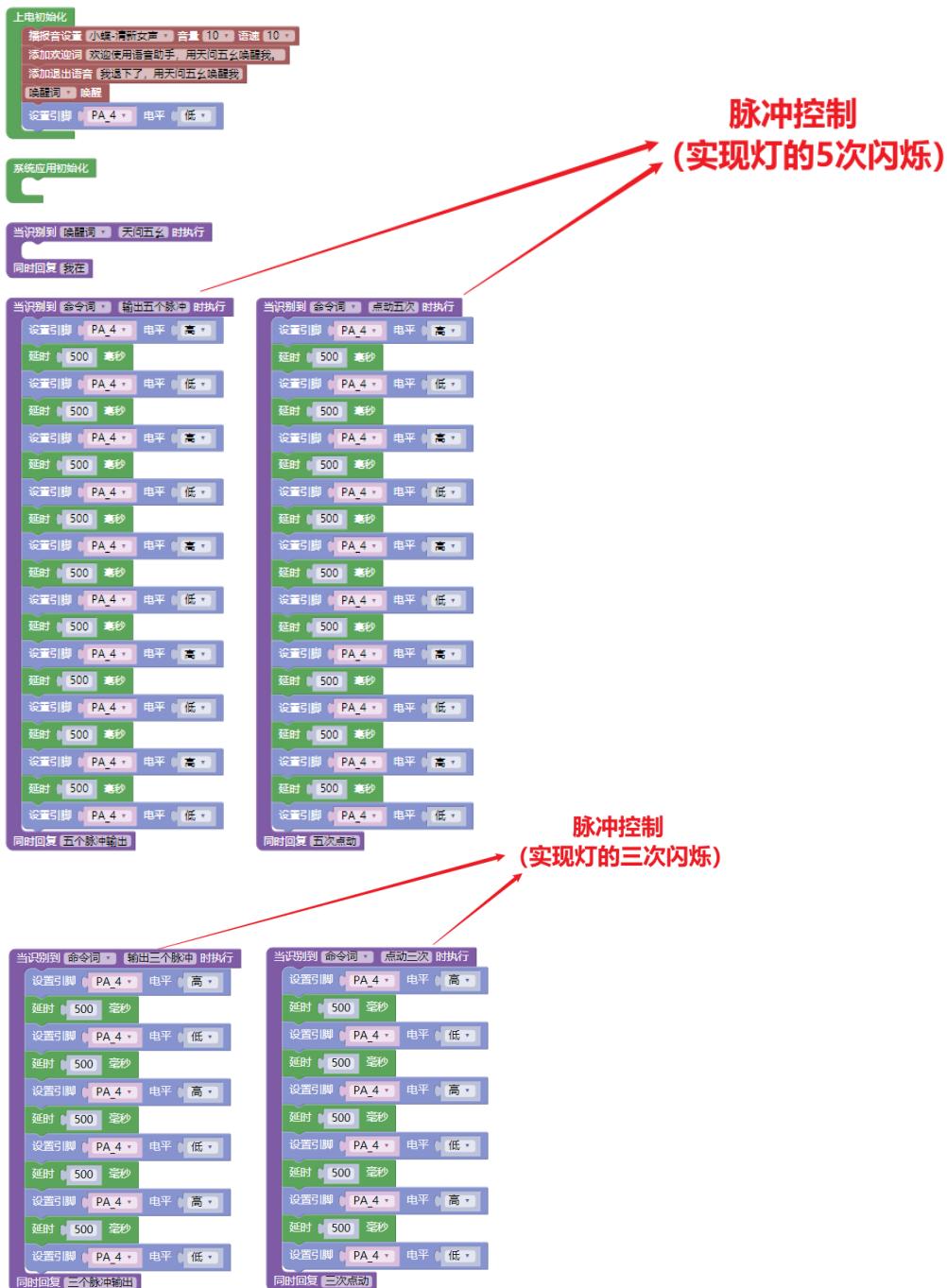


范例 1.10 语音控制引脚（脉冲）点动

一、范例功能

本范例通过学习如何控制引脚的脉冲，实现指定引脚的高低电平延时控制的功能，达到用户可以控制设备脉冲的目的。

二、范例分析

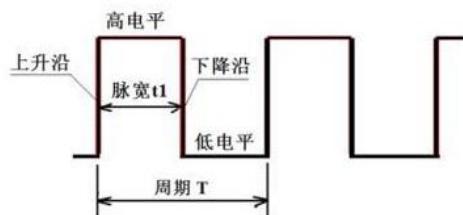


三、范例程序讲解

1. 脉冲信号

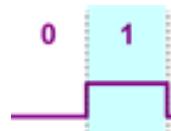
脉冲信号,也就是像脉搏跳动这样的信号,相对于直流,断续的信号,如果用水流形容,直流就是把龙头一直开着淌水,脉冲就是不停的开关龙头形成水脉冲。你把手电打开灯亮,这是直流,你不停的开关灯亮、熄,就形成了脉冲,开关速度的快慢就是脉冲频率的高低。

标准脉冲信号如下图所示：



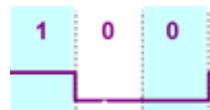
我们把脉冲信号从低电压到高电压的沿称为上升沿,从高电压到低电压的沿称为下降沿,有些数据也称为前沿和后沿。低电压叫低电平,高电压叫高电平。

高脉冲：即从逻辑 0 变化到逻辑 1 再变化到逻辑 0,如此便是一个高脉冲。



在单片机中定义高脉冲就是让某个 I/O 先输出逻辑 0,接着保持一定的时间(延时),再输出逻辑 1,同样保持一定的时间(延时),最后再转变输出为逻辑 0+延时。

低脉冲则反之,即从逻辑 1 变化到逻辑 0 再变化到逻辑 1。



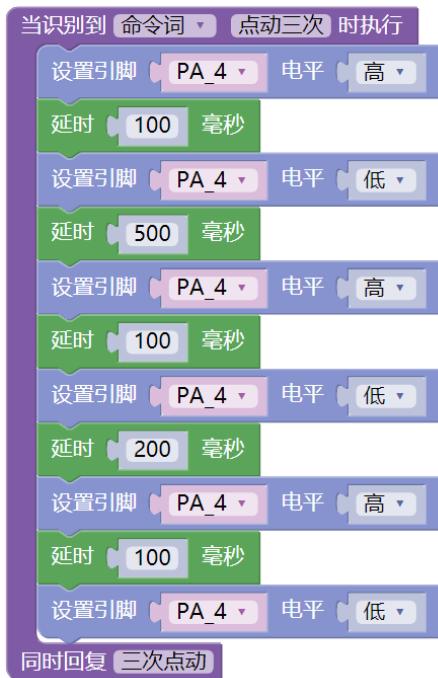
在范例 1.10 中,要注意引脚的初始电平,与命令词触发脉冲点动的起始电平,需设置不同的电平。如果开始设置高电平,在听到命令词“输出三个脉冲”时,开始的实际的电平变化为高→低→高,即会先拉低再拉高,运行效果与命令词不符。



2. 修改延时时间

在入门模式只能设置每个脉冲为相同的时间（高电平和低电平维持的时间一致），在标准模式中你可以设置脉冲的不同时间，通过修改延时的时间，来设置相应电平的维持时间，根据实际应用场景，实现不同速度的电平变化。

如下修改“点动三次”触发事件的延时时间，实现灯光灭 0.1 秒→亮 0.5 秒→灭 0.1 秒→亮 0.2 秒→灭 0.1 秒→亮的闪烁变化。

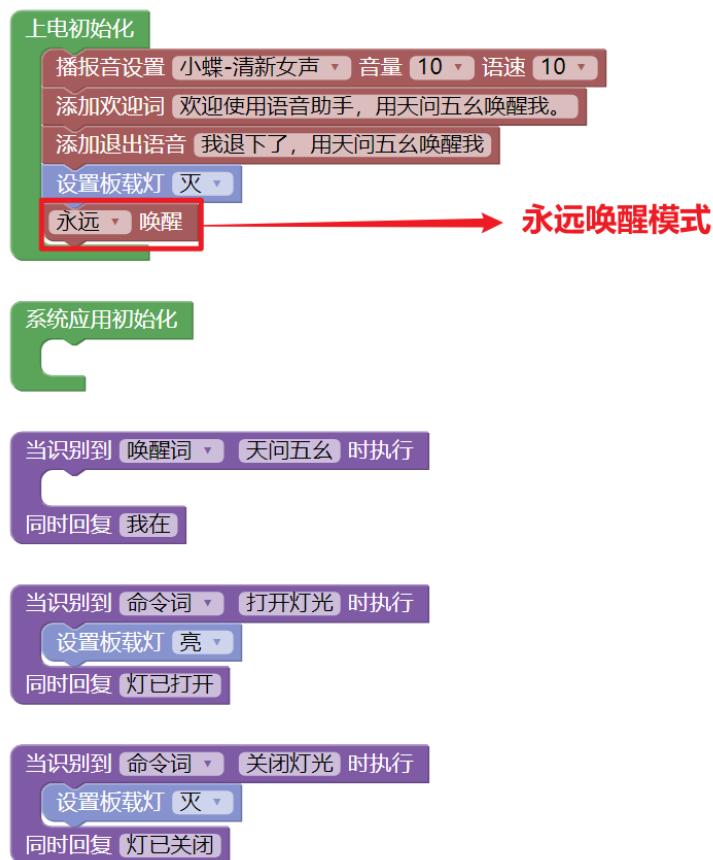


范例 2.1 永远唤醒模式

一、范例功能

本范例通过学习如何修改唤醒模式，实现直接使用命令词控制设备的功能，达到用户可以根据实际应用场景调整唤醒模式的目的。

二、范例分析



三、范例程序讲解

范例 2.1 中设置了永远唤醒的模式，而非唤醒词唤醒的模式，使其一直处于工作状态，永远不会退出（供电的情况下），因此可删除对于退出语音的设置，我们可以直接使用命令词控制板载的 LED 灯。

在制作一些具有简单的语音控制功能的作品/产品时，使用永远唤醒的模式，用户随时喊命令词都可以识别，优化了用户的使用体验，操作也更加便捷。

范例 2.2 调整音量

一、范例功能

本范例通过学习如何设置整体播报音量，实现语音调节整体音量逐步增大或减小的功能，达到用户可以轻松调节整体音量大小的目的。

二、范例分析



三、涉及指令讲解

➤ 语音设置相关



用于调节整体语音播报的音量大小，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，7 为最大音量。播报音量的设置，和设置唤醒时间一样，一般不放在初始化中，需要放在某一条具体的语音中。

注意：设置的音量断电后会自动保存，重新上电后不会改变。

四、范例程序讲解

范例 2.2 中设置了音量调节的范围为 1 到 7 级，即音量最小为 1 级，音量最大为 7 级，我们可以通过唤醒设备后，使用命令词增大/减小音量。

用户可根据实际应用场景来调节好适合的音量，注意“设置播报音量”指令需放置在事件触发中，而非初始化中，断电后会自动保存。

范例 2.3 自学习控制板载灯

一、范例功能

本范例通过学习如何自学习新的命令词/唤醒词，实现自学习控制板载灯的功能，达到用户可以通过语音学习新的识别词，并控制外设的目的。

二、范例分析



三、涉及指令讲解

➤ 自学习设置相关

- 自学习语音设置
- | |
|--|
| ID[300] 学习-唤醒词 好好搭搭 回复语 我在 |
| ID[302] 学习-命令词 执行第一条指令 回复语 好的 学习指令时语音 请说第一条要学习的指令 学习成功后语音 恭喜你第一条指令学习成功 |
| ID[303] 学习-命令词 执行第二条指令 回复语 马上执行 学习时播报语音 请说第二条要学习的指令 学习成功回复语音 恭喜你第二条指令学习成功 |
| ID[304] 学习-命令词 执行第三条指令 回复语 好的，马上执行 学习时播报语音 请说第三条要学习的指令 学习成功回复语音 恭喜你第三条指令学习成功 |
| ID[305] 学习-命令词 执行第四条指令 回复语 好嘞 学习时播报语音 请说第四条要学习的指令 学习成功回复语音 恭喜你第四条指令学习成功 |
| ID[306] 学习-命令词 执行第五条指令 回复语 好的马上执行 学习时播报语音 请说第五条要学习的指令 学习成功回复语音 恭喜你第五条指令学习成功 |
| ID[307] 学习-命令词 执行第六条指令 回复语 行，马上做 学习时播报语音 请说第六条要学习的指令 学习成功回复语音 恭喜你第六条指令学习成功 |
| ID[308] 学习-命令词 执行第七条指令 回复语 好的 学习时播报语音 请说第七条要学习的指令 学习成功回复语音 恭喜你第七条指令学习成功 |
| ID[309] 学习-命令词 执行第八条指令 回复语 马上执行 学习时播报语音 请说第八条要学习的指令 学习成功回复语音 恭喜你第八条指令学习成功 |
| ID[310] 学习-命令词 执行第九条指令 回复语 好的，马上执行 学习时播报语音 请说第九条要学习的指令 学习成功回复语音 恭喜你第九条指令学习成功 |
| ID[311] 学习-命令词 执行第十条指令 回复语 好嘞 学习时播报语音 请说第十条要学习的指令 学习成功回复语音 恭喜你第十条指令学习成功 |
1. 可以自定义唤醒词和命令词，自行添加回复播报的语音，通过多个命令词或唤醒词来执行指令不需要再次编译下载程序。自学习的 ID 从 300 开始到 311，总共可以自定义 1 个唤醒词，10 个命令词。注意使用自学习语音时要注意在每个输入框都要有文字（执行内容或回复词可以为空），否则模型生成会处理失败。自学习的操作流程，具体可参考范例程序讲解。
- 当识别到ID为 302 时执行
2. 学习成功后，再次说出该识别词，就会识别并触发相应的事件。

四、范例程序讲解

语音的自学习功能，在实际的开发应用中会经常用到。我们可以使用程序定义新的命令词内容（以普通话的形式识别），也可以通过自学习来学习方言形式的命令词。

1. 自学习学习流程

(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！

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。

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。

2. 修改自学习命令词

根据实际应用场景，在范例 2.3 的基础上进行修改。

如自学习“打开客厅灯”、“关闭客厅灯”命令词，回复语分别为“客厅灯已打开”、“客厅灯已关闭”，并修改学习指令时的语音、学习成功后的语音，更加符合应用场景。

自学习语音设置
ID[300] 学习-唤醒词 好好搭搭 回复语 我在
ID[302] 学习-命令词 执行第一条指令 回复语 客厅灯已打开 学习指令时语音 请说打开客厅灯指令 学习成功后语音 恭喜你打开客厅灯指令学习成功
ID[303] 学习-命令词 执行第二条指令 回复语 客厅灯已关闭 学习时播报语音 请说关闭客厅灯指令 学习成功回复语音 恭喜你关闭客厅灯指令学习成功

我们可以也通过“执行第一条指令”和“执行第二条指令”这两条命令词，来分别实现和命令词“打开客厅灯”和“关闭客厅灯”相同的效果。

你可以选择语音自学习方言形式的命令词，在程序中设定具有相同意义的普通话形式的命令词，这样就可以实现普通话和方言的两种控制方式。

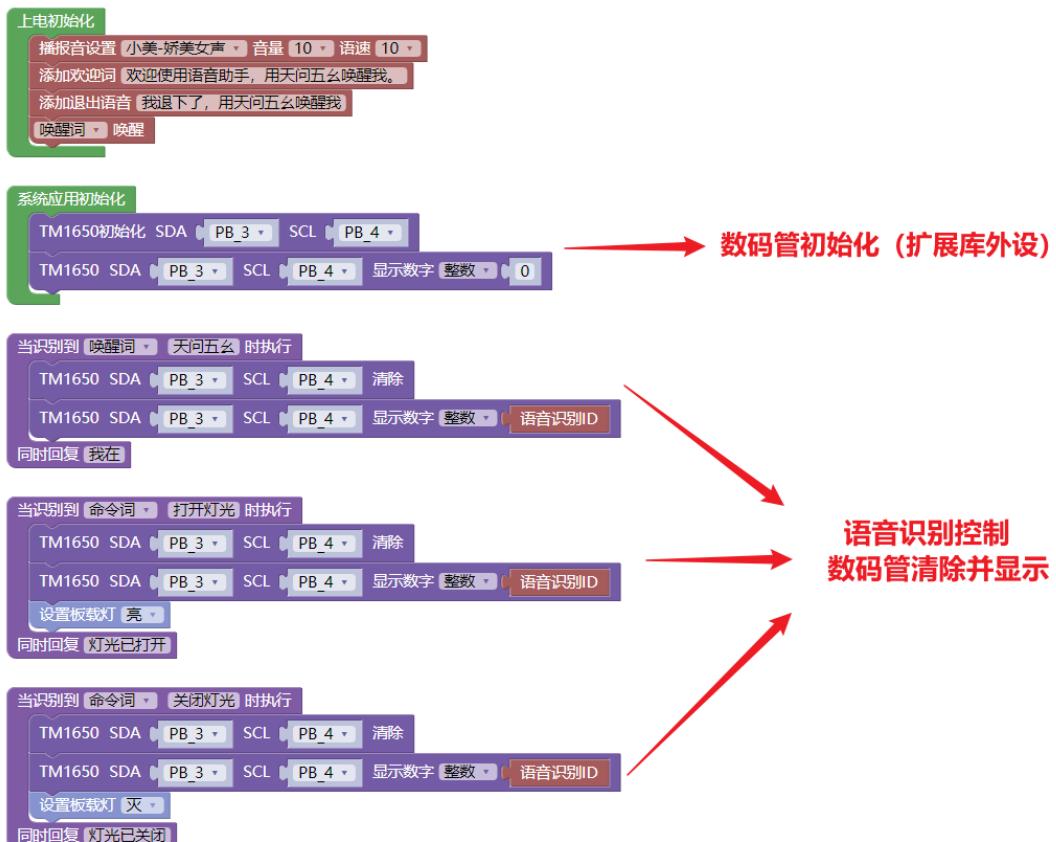
注意：学习完新的命令词后，断电后重新上电不会删除。如果需要修改请删除后重新学习。目前最多支持 32 条自定义语音。

范例 2.4 数码管（TM1650）使用

一、范例功能

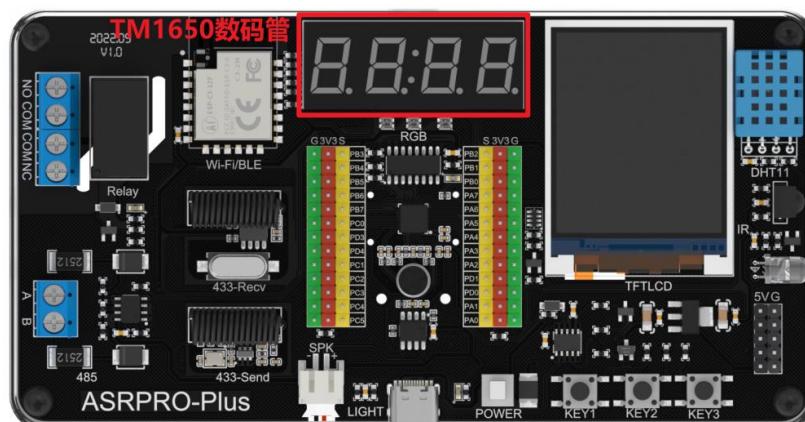
本范例通过学习如何使用 TM1650 数码管，实现数码管显示语音识别 ID 的功能，达到用户可以根据实际应用场景让数码管显示相关数值的目的。

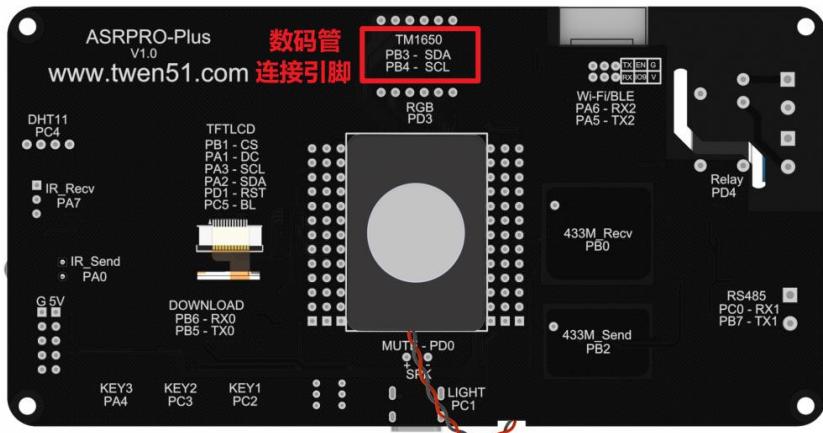
二、范例分析



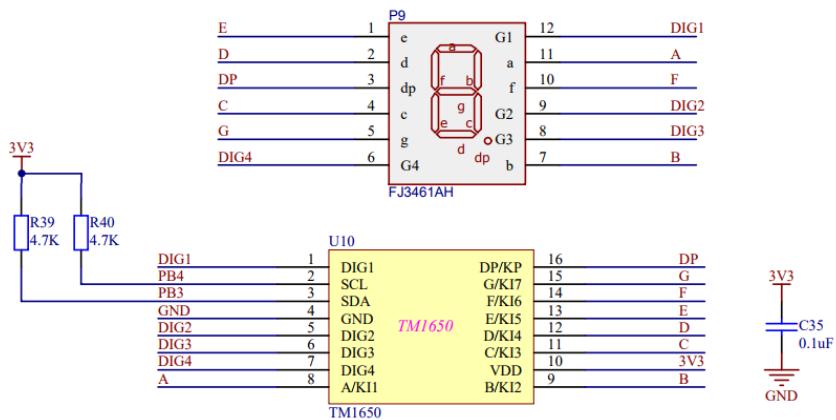
三、电路及实物说明

ASRPRO-Plus 板载数码管所处位置如下图所示：



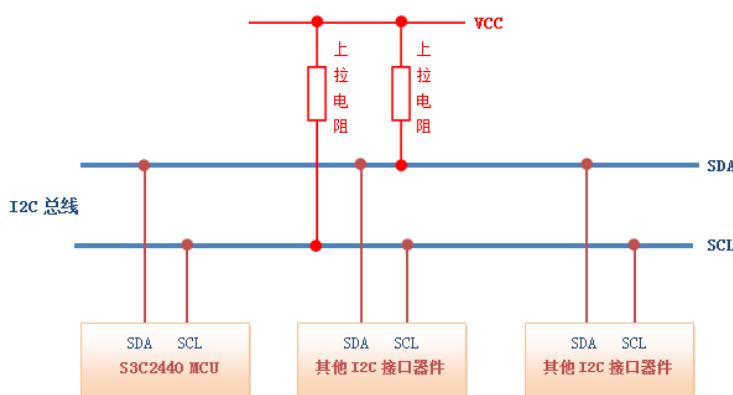


内部电路原理图如下图所示：



由上图得知 ASRPRO-Plus 板载 TM1650 数码管连接于 PB4 (SCL)、PB3 (SDA) 引脚。

TM1650 数码管可以显示数字和其他信息。采用 I2C 通讯，I2C 串行总线一般有两根信号线，一根是双向的数据线 SDA，另一根是时钟线 SCL，其时钟信号是由主控器件产生。所有接到 I2C 总线设备上的串行数据 SDA 都接到总线的 SDA 上，各设备的时钟线 SCL 接到总线的 SCL 上。对于并联在一条总线上的每个 IC 都有唯一的地址。



I2C 总线物理拓扑图

一般情况下，数据线 SDA 和时钟线 SCL 都是处于上拉电阻状态。因为：在总线空闲状态时，这两根线一般被上面所接的上拉电阻拉高，保持着高电平。

ASRPRO-Plus 开发板芯片引脚及其功能如下：

| Pin Name | Functional | 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 |

由图我们可以看到具有相同功能的引脚还有很多，注意引脚 PB3-PC0 已分别被数码管、串口 0 和 RS485 占用，你可以外接其他 TM1650 数码管到 PA2、PA3 或 PC3、PC4 引脚进行使用。

四、涉及指令讲解

添加扩展 点击 **添加 TM1650 扩展库**（显示类别），点加载图标，软件会通过网络下载库到本地，如果库版本有更新，点击更新会更新库文件，点击移除 **移除** 按钮可以删除库文件。点击问号 **?** 按钮可以查看库的使用说明。点击星号 **★** 可以给库评级和反馈。

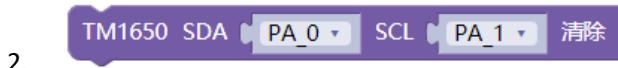


添加好扩展库后，点击左上角 **返回**，在 **扩展** 中可找到相关库指令。

➤ TM1650 相关

1. **TM1650初始化 SDA PA_0 SCL PA_1**

用于初始化 TM1650 数码管在指定的引脚，属于外围设备初始化的指令应放在“系统应用初始化”内。



2. 用于清除 TM1650 数码管显示的内容。



3. 用于在 TM1650 数码管上显示指定数字，下拉可选择类型为整数/小数，可以显示某个变量的数值。

➤ 语音设置相关

语音识别ID

可调用语音识别的 ID，是一个变量，表示识别词在内部存储的 ID 号，不可单独使用，应结合显示或判断的指令使用。

➤ 控制相关

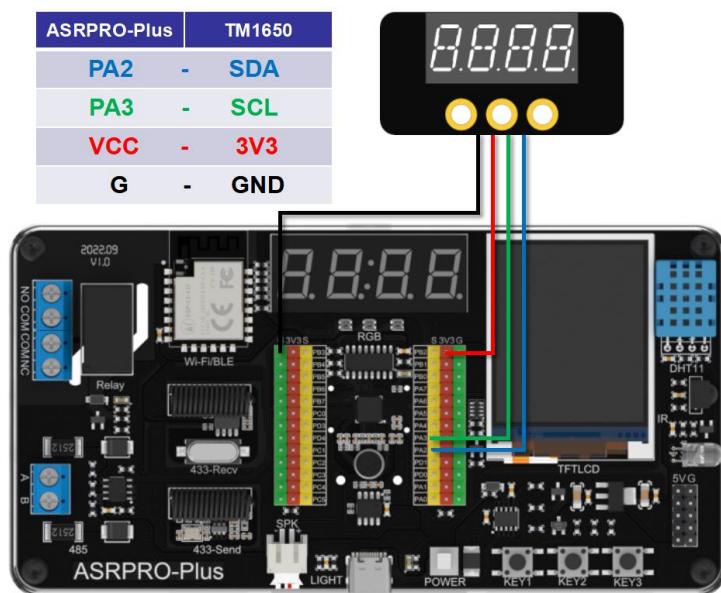
系统应用初始化

主要用于连接的外围设备进行初始化（扩展内添加的外设）。

五、范例程序讲解

以外接 TM1650 数码管为例，在范例 2.4 的基础上进行修改。

具体硬件连接可参考下图：



你也可以选择接在 PC3、PC4。

根据引脚连接，修改范例中的数码管的引脚设置。



下载程序后，数码管显示 0，喊出不同的识别词时，会显示相应的 ID。当然你也可以将传感器获取的数值显示在数码管上。

范例 2.5 亮度 (ADC) 测试仪

一、范例功能

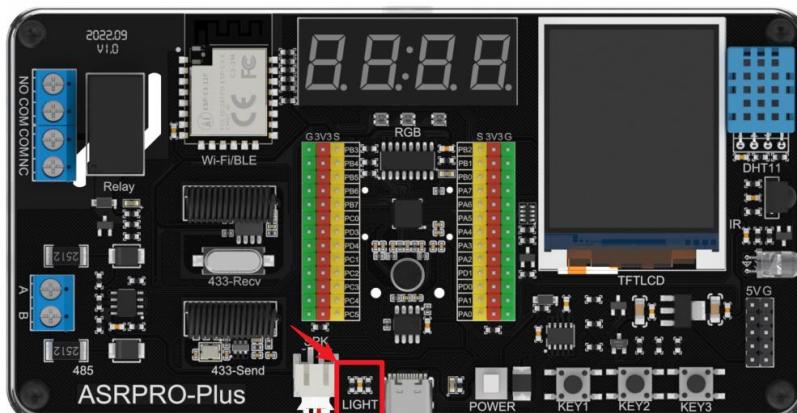
本范例通过学习如何使用亮度传感器，了解什么是 ADC，实现亮度测试仪的功能，达到用户可以使用 ADC 采集传感器数值并播报的目的。

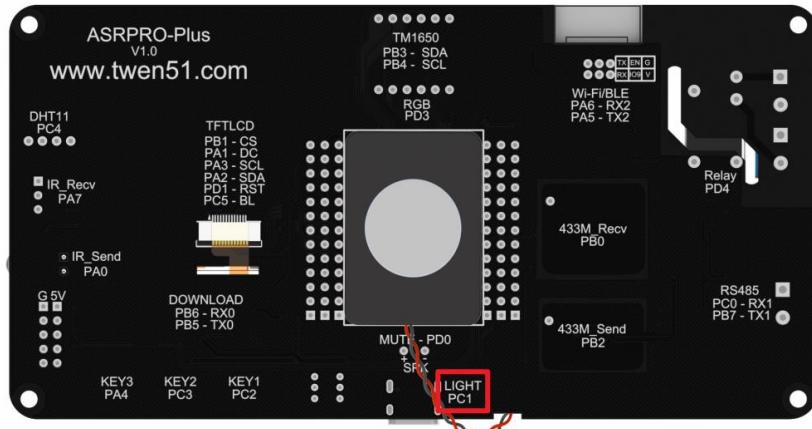
二、范例分析



三、电路及实物说明

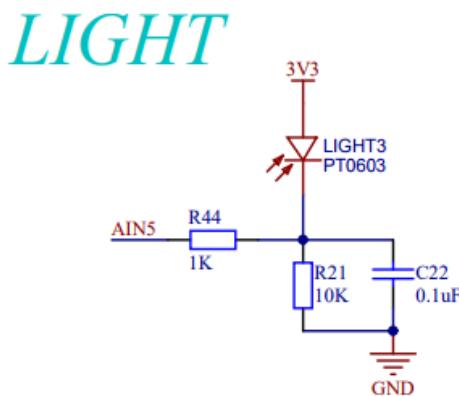
ASRPRO-Plus 板载亮度传感器所处位置如下图所示：





由上图我们可以看到亮度传感器连接在 PC1 引脚。

其内部原理图如下图所示：



亮度传感器类似于三极管，其管芯是一个具有光敏特征的 PN 结，光线控制光敏传感器的开关，利用光敏元件将光信号转换成电信号输出，内部光敏电阻可以感应光线的明暗变化，输出电信号给主控制器，普遍适用于光控灯、路灯、照相机。

像亮度、温度等物理量都是随时间变化而变化的，是没有规律的，在控制领域我们把这些随时间变化而发生连续变化的物理量称为模拟量。

单片机只能处理数字信号，如果要显示模拟量，需要将其转换成数字量，比如说电压表就是把电压的模拟信号转换成数字信号显示的，这就要用到 ADC (analog to digital converter) 模拟数字转换器了。

在 ASRPRO-Plus 开发板芯片集成了一个 12 位的 ADC，12 位 ADC 就是 12 位二进制数的意思，即 000000000000-111111111111 等于 10 进制的 0-4095。12 代表的是其分辨率位 $1/\{ (2^{\text{n}})^{\text{次方}} \}$ ，位数也高，精度也高，进而误差越小。

ADC 的位数可以根据具体需求进行设置：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。图形化 ADC 指令设置默认是 12 位 AD，也就是采样值范围在 0-4095 之间。

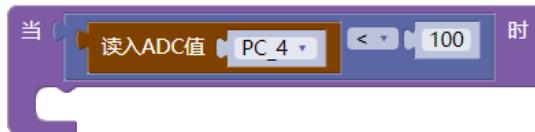
AIN 是内部 ADC 的模拟输入引脚，ASRPRO-Plus 开发板芯片引脚及其功能如下：

| 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 |

由上图我们可以看到 AIN2-AIN5，也就是 PC1-PC4 都是 ADC 的模拟输入引脚，可用于模数转换。连接亮度传感器使用时，光线越强 AD 采样值越大，反之越小，取值范围为 (0, 4095)。

四、涉及指令讲解

➤ 事件触发相关



程序会自动判断对应引脚的 ADC 的模拟量读取值，范围为 0-4095，条件成立时会自动触发执行框里面的程序模块。可将“读入 ADC 值”指令拖出来使用，引脚下拉可选择 PC1-PC4。

➤ 执行动作相关



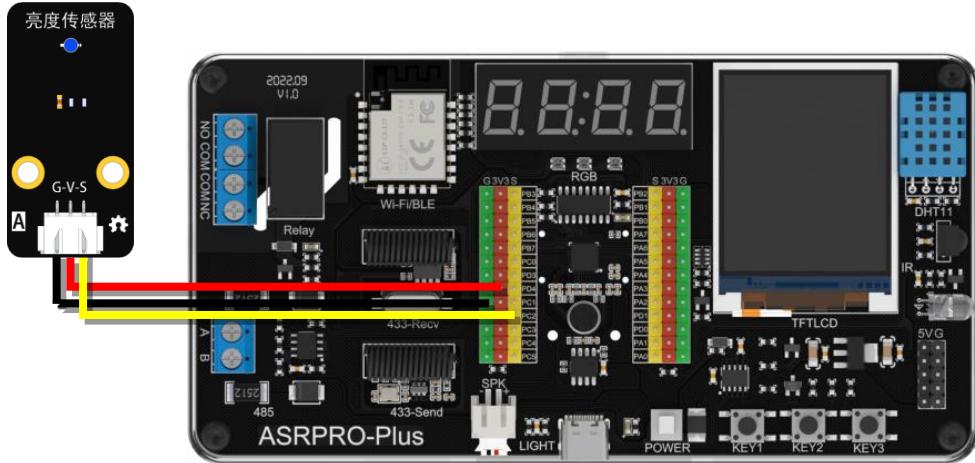
支持号码和数字两种模式，支持 2 位小数，支持最大整数为 9 千亿，支持负数。号码

模式一般为个位数的播放，如数字“0-9”，数值模式则为有计数单位的数值，比如 100 播放为一百。

五、范例程序讲解

范例 2.5 通过 ADC 采集了环境亮度，并使用数值模式播报；当亮度低于 100 时自动触发唤醒并播报“亮度太低了”。

这里以在 PC2 引脚外接一个光敏传感器为例，硬件连接如下图所示：



在范例 2.5 的基础上进行修改。



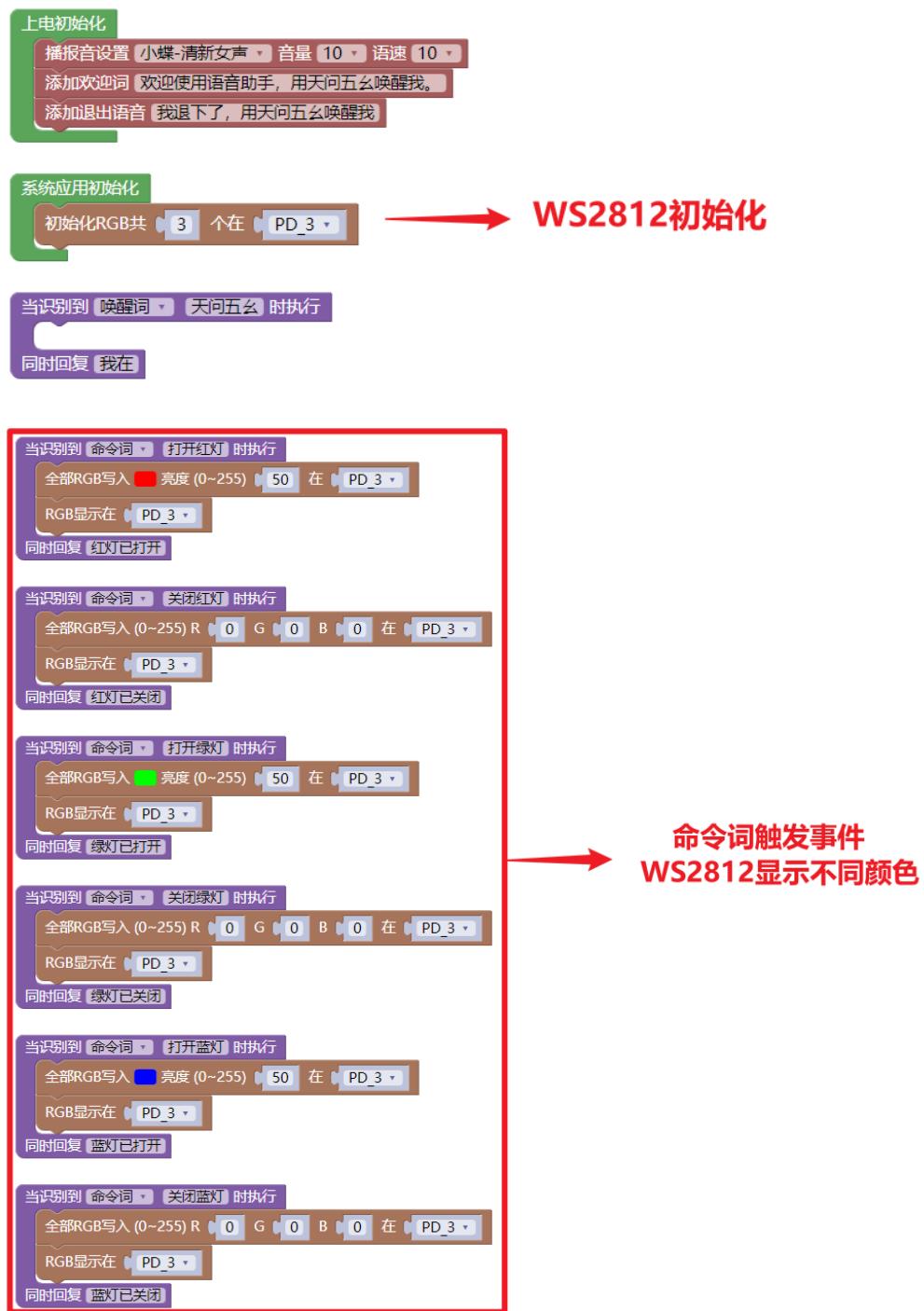
我们还可以通过 ADC 采集其他模拟量，比如说温度、湿度等等。

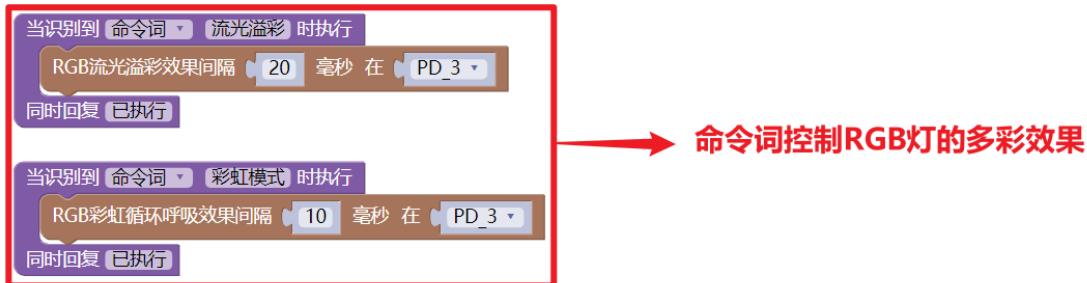
范例 2.6 语音控制 WS2812 彩灯

一、范例功能

本范例通过学习如何使用 WS2812 彩灯，实现语音控制 WS2812 彩灯的功能，达到用户可以根据实际应用场景使用 WS2812 呈现不同灯光效果的目的。

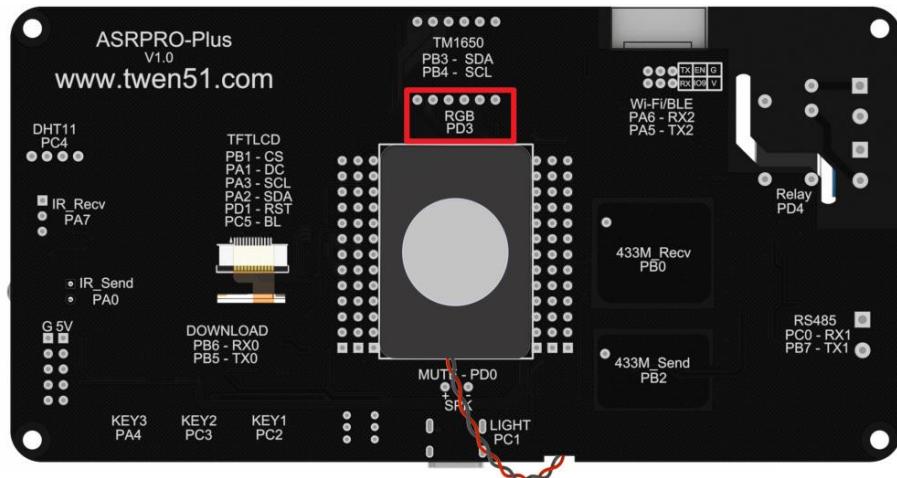
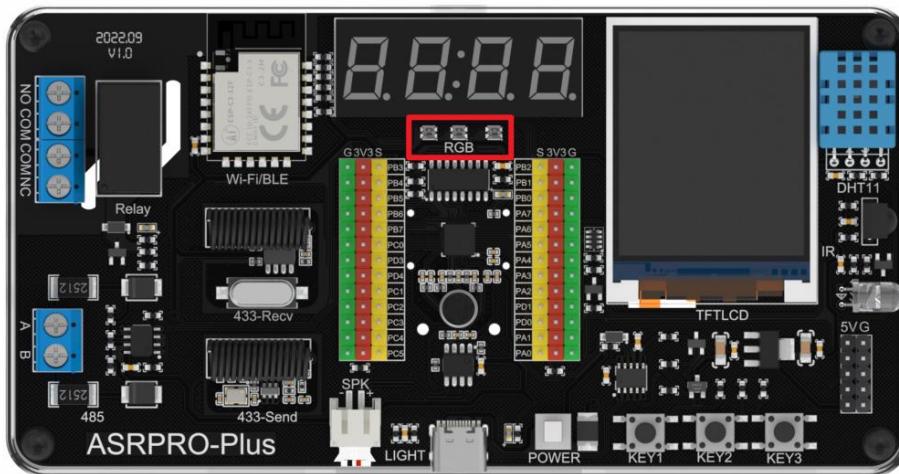
二、范例分析





三、电路及实物说明

ASRPRO-Plus 板载 WS2812 彩灯 (RGB) 所处位置如下图所示：



由上图我们可以看到 RGB 连接在 PD3 引脚。

四、涉及指令讲解

添加扩展
点击 **添加** WS2812 扩展库 (显示类别), 点加载图标, 软件会通过网络
下载库到本地, 如果库版本有更新, 点击更新会更新库文件, 点击移除 **移除** 按钮可以删除

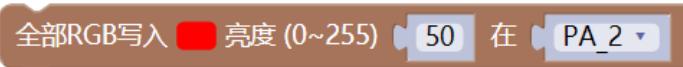
库文件。点击问号  按钮可以查看库的使用说明。点击星号  可以给库评级和反馈。



添加好扩展库后，点击左上角  返回，在  中可找到相关库指令。

➤ WS2812 相关

1. 

用于初始化 WS2812 彩灯在指定的引脚，设置灯珠的数量，属于外围设备初始化的指令应放在“系统应用初始化”内。
2. 

用于设置指定引脚的 RGB 灯显示不同颜色（点击色块可选择指定颜色），并设置亮度，不可单独使用，需配合“RGB 显示”指令使用。
3. 

用于设置指定引脚的 RGB 灯显示不同的颜色（颜色可自定义），不可单独使用，需配合“RGB 显示”指令使用。
4. 

用于将设置好的 RGB 灯颜色显示出来，需配合“RGB 写入”指令使用，相当于显示生效。
5. 

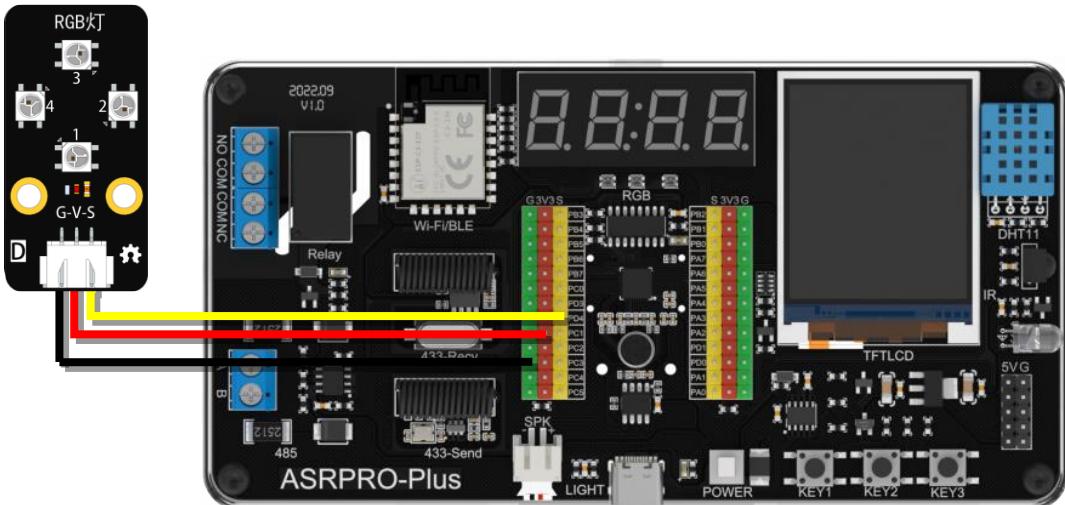
用于设置指定引脚的每一颗 RGB 灯珠呈现不同颜色的循环呼吸灯效果，可设置间隔时间，即颜色变换的速度。
6. 

用于设置指定引脚的 RGB 灯珠呈现同色的循环呼吸灯效果，可设置间隔时间，即颜色变换的速度。

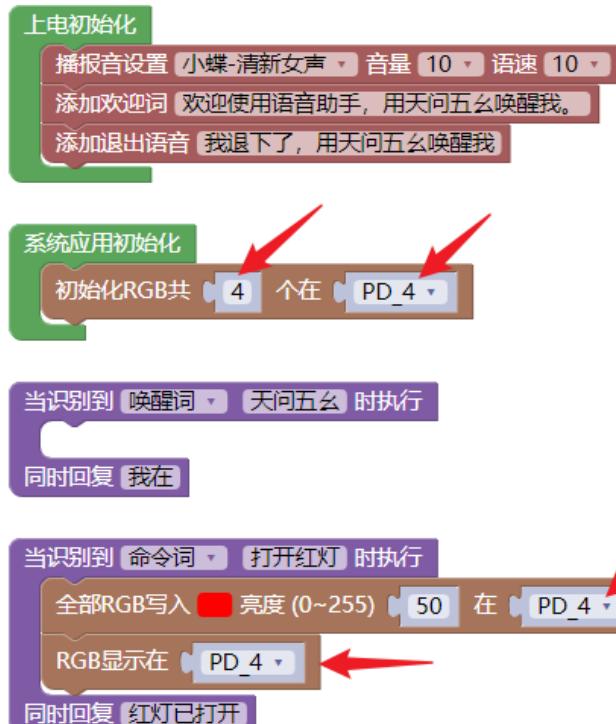
五、范例程序讲解

范例 2.6 通过不同的命令词控制 RGB 灯显示不同颜色以及各种灯光效果。

这里以在 PD4 引脚外接 4 颗灯珠的 RGB 模块为例，硬件连接如下图所示：



在范例 2.6 的基础上修改初始化中的灯珠数量，以及设置的引脚。



当说出“打开红灯”时，4 颗 RGB 灯珠全亮红色，你还可以添加更多命令词，实现不同的灯光效果。

范例 2.7 语音播报当前温湿度

一、范例功能

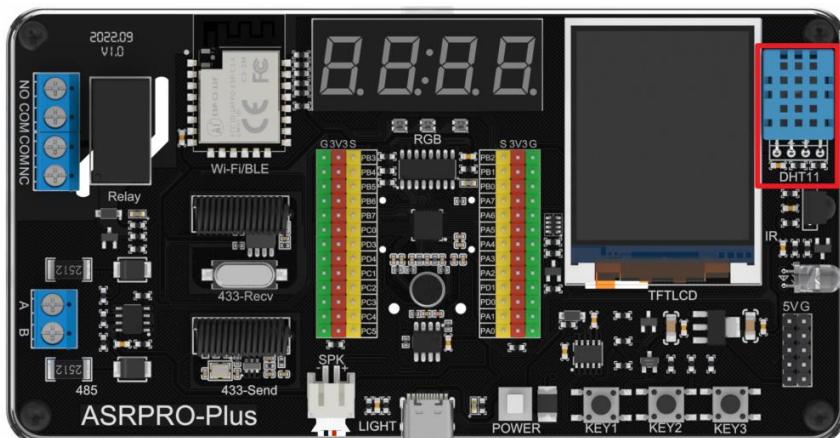
本范例通过学习如何使用 DHT11 温湿度传感器，实现语音播报当前温湿度的功能，达到用户可以语音获取当前环境的温湿度的目的。

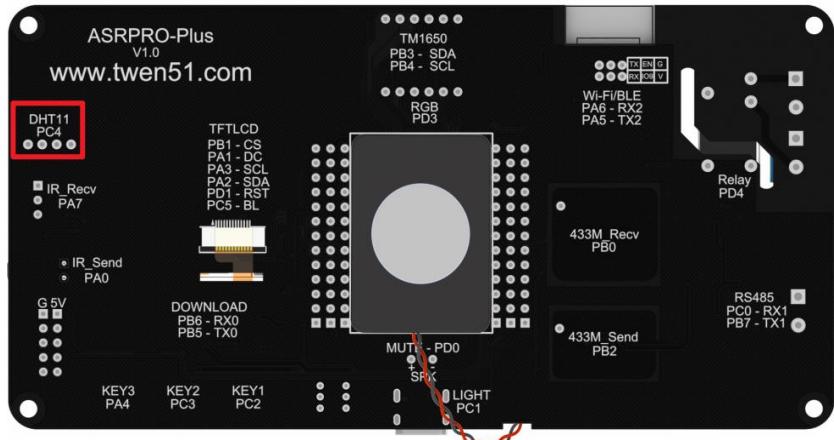
二、范例分析



三、电路及实物说明

ASRPRO-Plus 板载 DHT11 温湿度传感器所处位置如下图所示：





由上图我们可以看到 DHT11 温湿度传感器连接在 PC4 引脚。

四、涉及指令讲解

添加扩展 点击 **添加 DHTxx 扩展库 (传感器类别)**，点加载图标，软件会通过网络下载库到本地，如果库版本有更新，点击更新会更新库文件，点击移除 **移除** 按钮可以删除库文件。点击问号 **?** 按钮可以查看库的使用说明。点击星号 **★** 可以给库评级和反馈。



添加好扩展库后，点击左上角 **返回**，在 **扩展** 中可找到相关库指令。

➤ DHTxx 相关

1. **DHTXX 初始化类型 DHT11 在 PA_0**
用于初始化指定引脚连接的 DHT 传感器（下拉可选择 DHT11、DHT21、DHT22），属于外围设备初始化的指令应放在“系统应用初始化”内。
2. **DHTXX 读取温度 °C 在 PA_0**
用于获取指定引脚连接的 DHT 传感器采集的温度，下拉可选择单位为 C°（摄氏）或 F°（华氏）。

3. DHTXX读取湿度在 PA_0

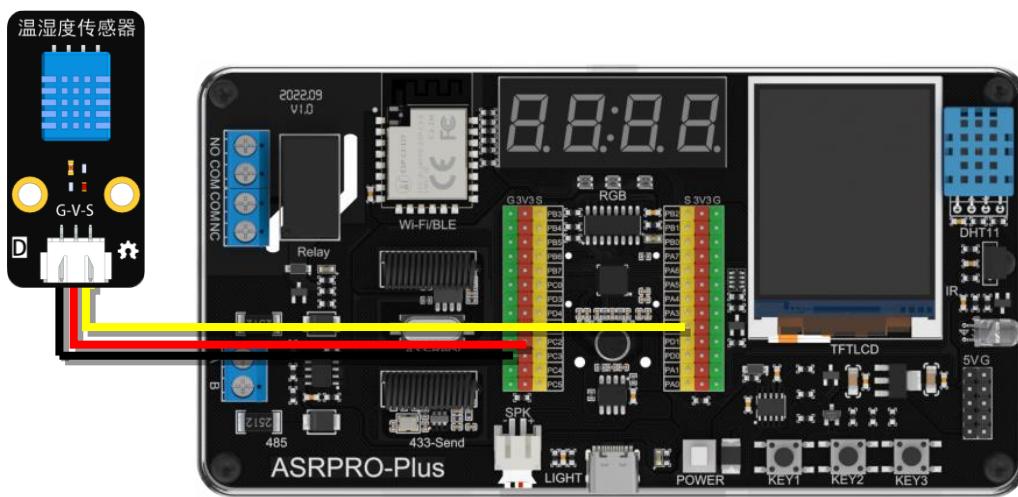
用于获取指定引脚连接的 DHT 传感器采集的相对湿度，单位是%RH。。

五、范例程序讲解

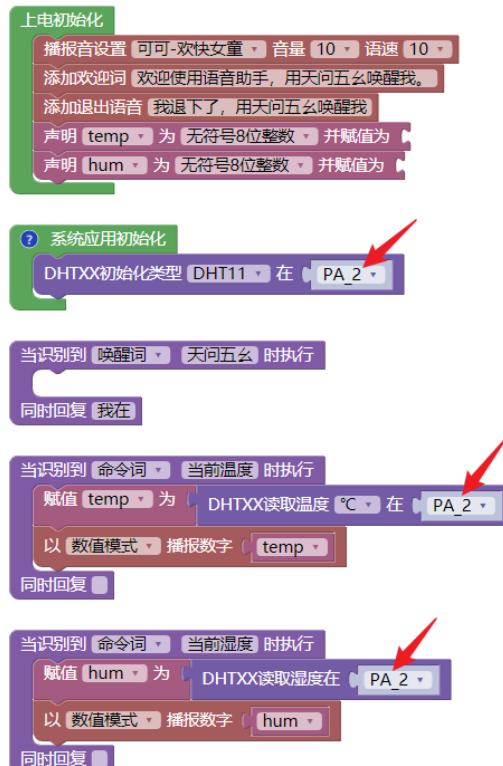
范例 2.7 通过不同的命令词播报当前环境的温度或湿度。

注意：使用板载的 DHT11 需要下载程序关闭电源，过一会儿再开机才能正常工作。使用时注意。如果 DHT11 接 5V 电源工作就正常。

这里以在 PA2 引脚外接 DHT11 为例，硬件连接如下图所示：



在范例 2.7 的基础上修改初始化以及触发事件中有关于 DHT11 引脚的设置。



唤醒后说出“当前湿度”时播报当前相对湿度值；说出“当前温度”时播报当前温度值。

范例 2.8 无线发射（433M）控制无线面板

一、范例功能

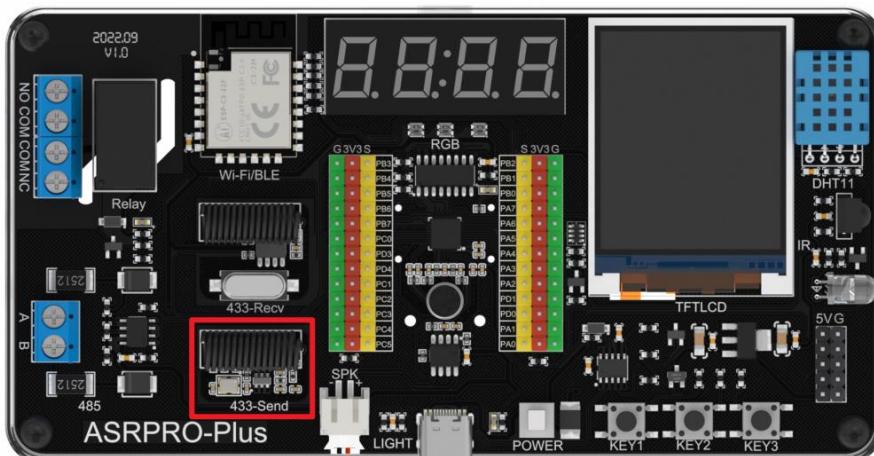
本范例通过学习如何使用无线发射数据，实现无线发射 1527 编码数据的功能，达到用户可以使用无线进行设备之间通信的目的。

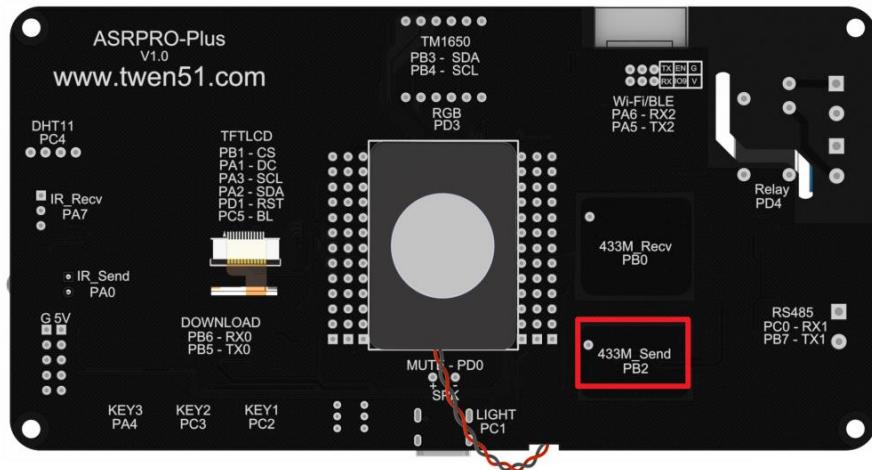
二、范例分析



三、电路及实物说明

ASRPRO-Plus 板载 433M 无线模块（发射部分）所处位置如下图所示：



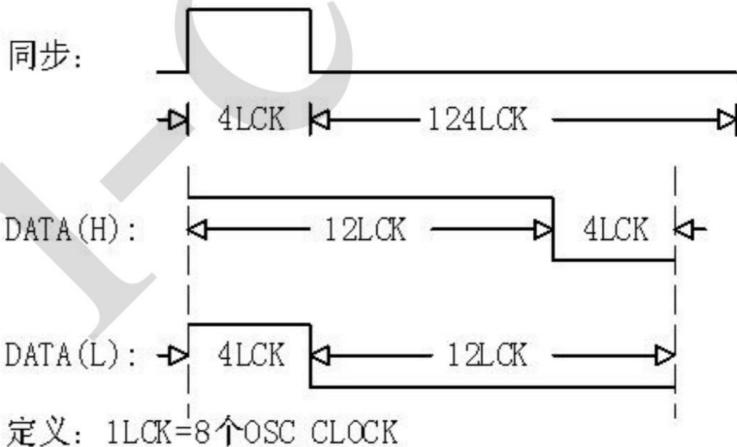


由上图我们可以看到 433M 无线模块（发射部分）连接在 PB2 引脚。

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有编码格式有 2262、1527 等，还有一些私有协议，比如宁波杜亚电机协议。

1527 编码输出的格式：

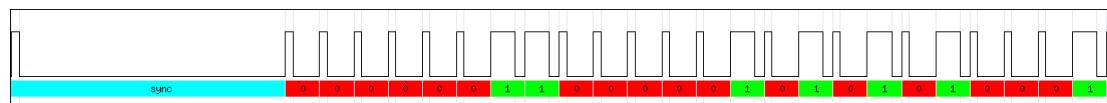


如上图，包含一个同步头，20 位地址码，4 位开关码。

同步头高电平和低电平脉宽比为 1:31；

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 300-500us 之间，常用的有 350us，公牛面板为 400us。

无线信号容易受到干扰，尤其第一帧，所以数据至少需要连续发送 3 次以上。库里默认为重复发送 10 次。

四、涉及指令讲解

点击 **添加扩展** 按钮 添加无线遥控扩展库（通讯类别），点加载图标，软件会通过网络下载库到本地，如果库版本有更新，点击更新会更新库文件，点击移除 **移除** 按钮可以删除库文件。点击问号 **?** 按钮可以查看库的使用说明。点击星号 **★** 可以给库评级和反馈。



添加好扩展库后，点击左上角 **返回**，在 **扩展** 中可找到相关库指令。

➤ 无线遥控相关

1.

无线发送初始化 引脚 PA_0 占用定时器 TIMERO 编码 1527

无线发送初始化参数，第一个为发送的引脚；第二个为调用的定时器：有 TIMERO-3 4 个定时器可以调用，注意不要和其他库同时调用同一个定时器，造成冲突；编码格式下拉可选择 1527、2262 或杜亚电机。属于外围设备初始化的指令应放在“系统应用初始化”内。
2.

无线发送-1527 地址(20位) 0x12345 数据(4位) 1

无线发送参数，第一个为 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；第二个为 D0-D3 的数据，包含 (0~15) 16 种开关；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。

五、范例程序讲解

本范例使用无线遥控扩展库，可通过单片机来实现 1527 的编码，驱动无线模块发送无线信号来控制对应的设备。

要实现无线控制其他设备，需要编写无线接收端的程序（范例 2.9）并下载到另一设备中（搭载 433M 无线模块），接收无线信号并在串口打印出来（可参照专业模式范例 1.8 串口设置与输出）。

添加无线控制继电器功能（发送端）

在范例 2.8 的基础上，我们可以添加无线控制继电器的功能，说出“打开继电器”时，则无线发送数据“2”；说出“关闭继电器”时，无线发送数据“3”。（无线接收端的范例修改见范例 2.9）

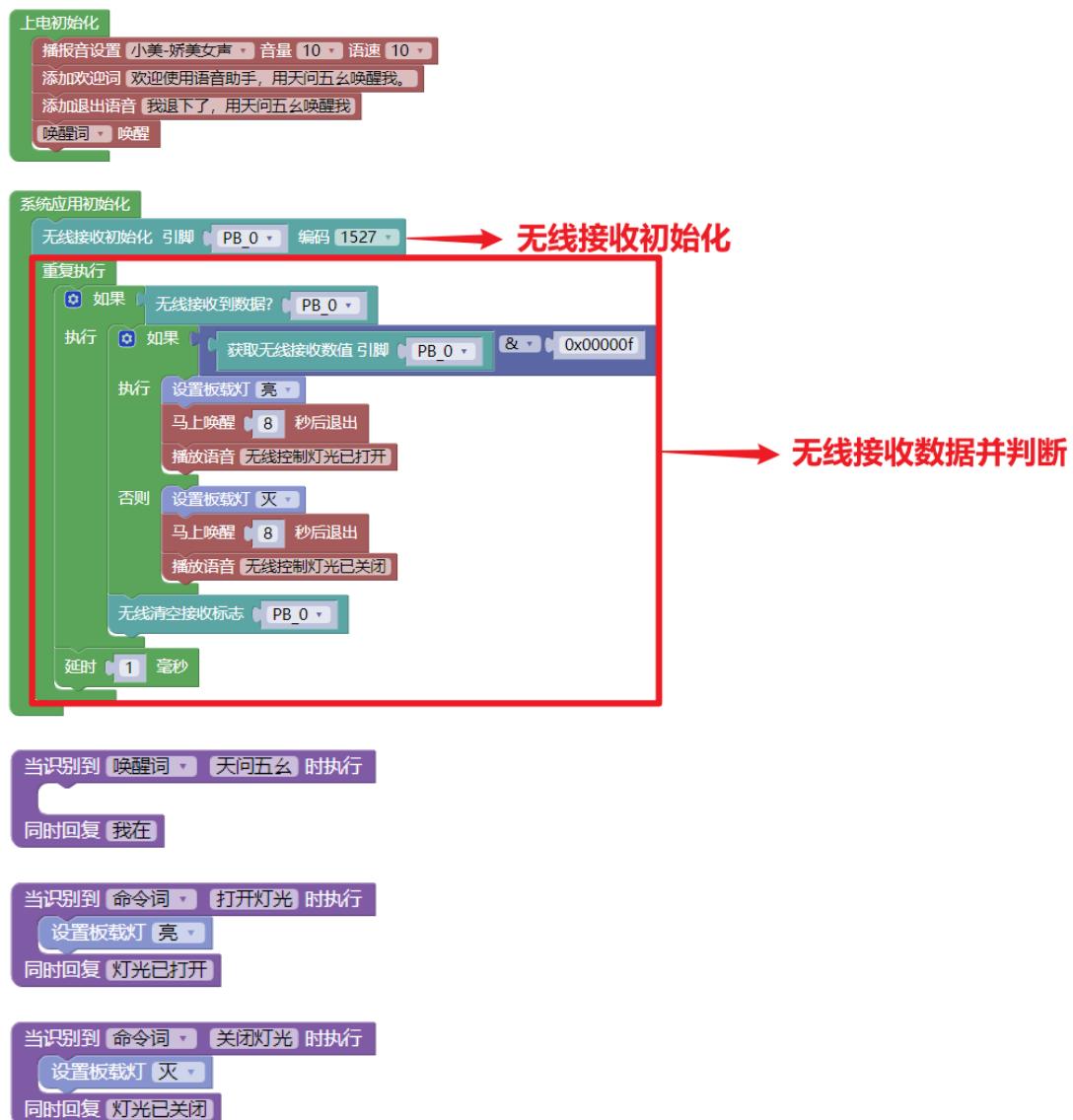


范例 2.9 无线接收（433M）控制板载灯

一、范例功能

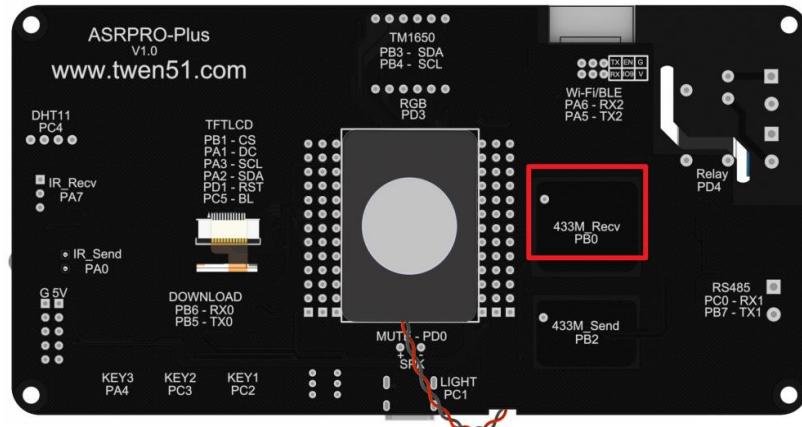
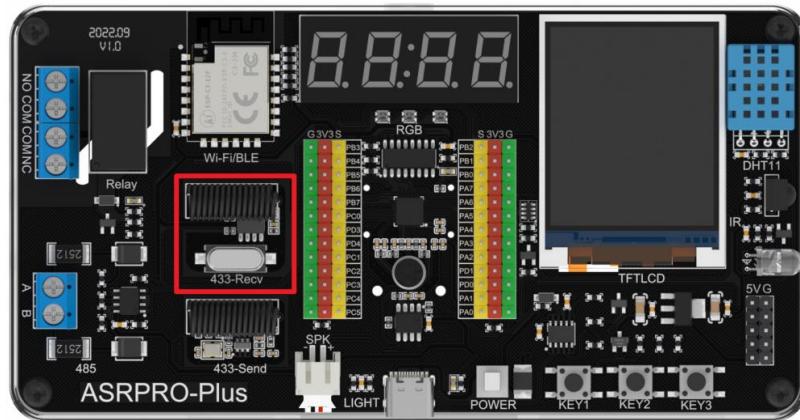
本范例通过学习如何使用无线接收数据，实现无线控制灯光的功能，达到用户可以使用无线进行设备之间通信的目的。

二、范例分析



三、电路及实物说明

ASRPRO-Plus 板载 433M 无线模块（接收部分）所处位置如下图所示：



由上图我们可以看到 433M 无线模块（接收部分）连接在 PB0 引脚。

四、涉及指令讲解

添加扩展
点击 **添加扩展** 按钮添加无线接收扩展库（其它类别），点加载图标，软件会通过网络下载库到本地，如果库版本有更新，点击更新会更新库文件，点击移除 **移除** 按钮可以删除库文件。点击问号 **?** 按钮可以查看库的使用说明。点击星号 **★** 可以给库评级和反馈。



添加好扩展库后，点击左上角 **返回**，在 **扩展** 中可找到相关库指令。

➤ 无线接收相关

1. 无线接收初始化 引脚 PA_0 编码 1527

无线接收初始化参数，第一个为接收的引脚；编码格式目前仅支持 1527 协议。属于外围设备初始化的指令应放在“系统应用初始化”内。

- 2. 
用于判断无线接收引脚是否接收到数据，返回 true（接收到数据）或 false（没有接收到数据）。
- 3. 
用于获取无线接收的数据（十进制显示）。
- 4. 
用于清空无线接收标志，即接收到数据后，标志需清零一次。

五、范例程序讲解

本范例实现 1527 编码和解码，1 号设备驱动无线模块发送无线信号（范例 2.8），2 号设备实现接收无线信号并控制灯光。

1. 与运算的用法

参加运算的两个数据，按二进制位进行“与”运算。

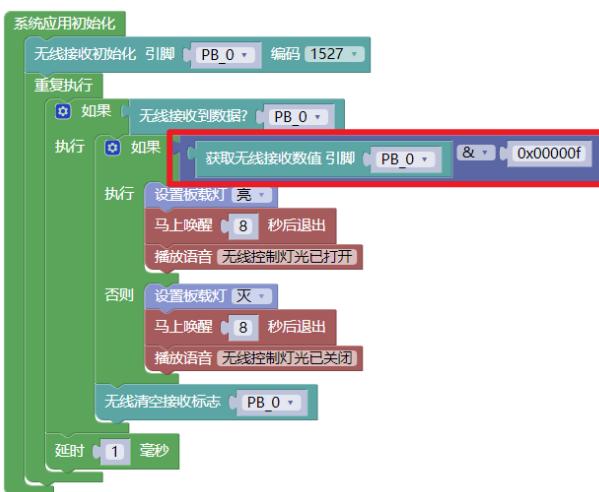
运算规则： $0\&0=0; 0\&1=0; 1\&0=0; 1\&1=1;$

即：两位同时为“1”，结果才为“1”，否则为 0

例如： $3\&5$ 即 $0000\ 0011 \& 0000\ 0101 = 0000\ 0001$ 因此， $3\&5$ 的值得 1。

例：设 $X=10101110$ ，取 X 的第 4 位，用 $X \& 0000\ 1111 = 0000\ 1110$ 即可得到；
还可用来取 X 的 2、4、6 位。

在本范例中，判断无线接收的数据的部分就是用到了与运算。

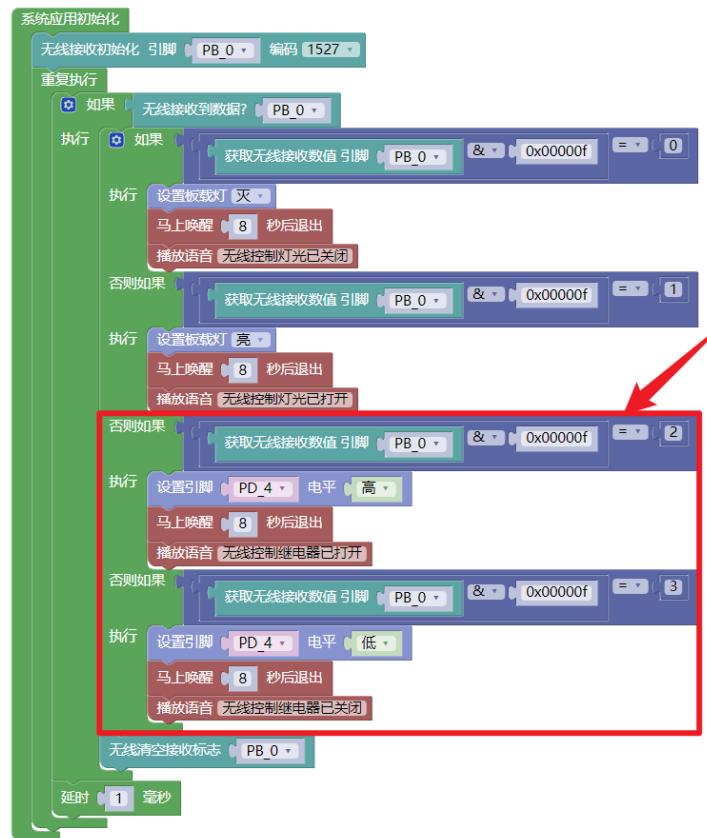


判断的条件就是取无线接收数值的第 4 位数据，如果是“1”，则返回 True，即控制灯亮；如果是“0”，则返回 False，即控制灯光熄灭。对应发送端发送的数据（见下图）。



2. 添加无线控制继电器功能（接收端）

在范例 2.9 的基础上，我们可以添加无线控制继电器的功能，当接收到数据“2”时，控制板载继电器打开；当接收到数据“3”时，控制板载继电器关闭。（无线发送端的范例修改见范例 2.8 的第五部分）



范例 2.10 彩屏使用

一、范例功能

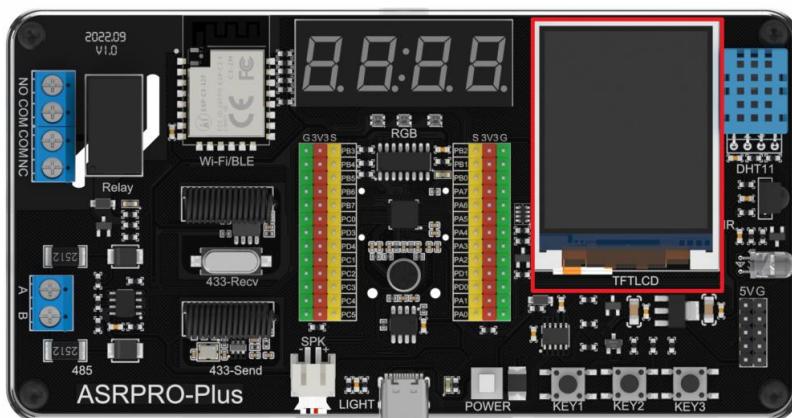
本范例通过学习如何使用彩屏，实现语音控制彩屏的功能，达到用户可以根据实际应用场景使用彩屏显示图片或文字的目的。

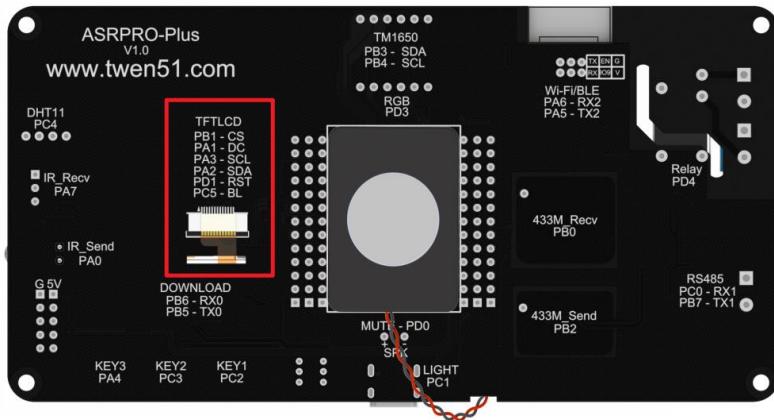
二、范例分析



三、电路及实物说明

ASRPRO-Plus 板载彩屏所处位置如下图所示：

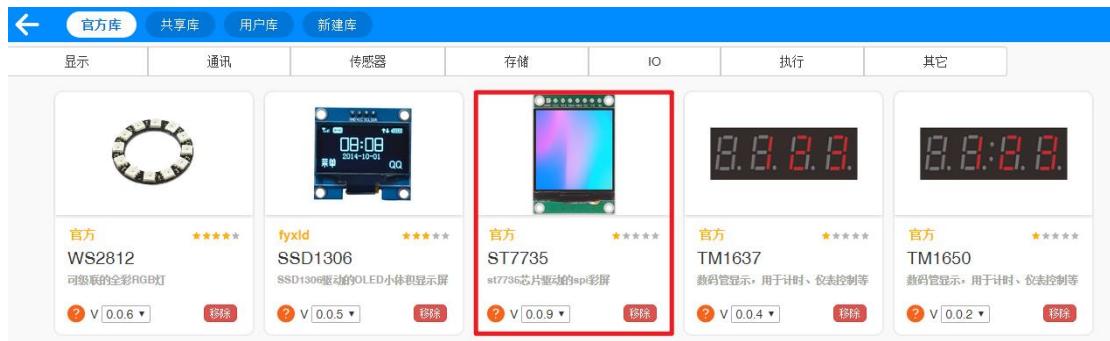




由上图我们可以看到彩屏连接在 PB1 (片选/使能信号)、PA1 (数据 (1) 和命令 (0) 控制管脚)、PA3 (SPI 时钟信号)、PA2 (SPI 的主设备输出, 从设备输入信号 (即 MOSI))、PD1 (屏幕复位信号) 和 PC5 (背光控制 (高电平亮)) 引脚。

四、涉及指令讲解

添加扩展 点击 **添加 ST7735 扩展库 (显示类别)**, 选择最新版本, 点加载图标, 软件会通过网络下载库到本地, 如果库版本有更新, 点击更新会更新库文件, 点击移除按钮可以删除库文件。点击问号 **?** 按钮可以查看库的使用说明。点击星号 **★** 可以给库评级和反馈。



添加好扩展库后, 点击左上角 **返回**, 在 **扩展** 中可找到相关库指令。

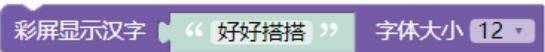
➤ ST7735 相关

1. 彩屏初始化 (模拟SPI) 分辨率 128*128 CS PA_4 SCL PA_5 SDA PA_6 DC PA_7 RES PB_0

用于彩屏初始化设置。下拉可选择分辨率有 128*128, 128*160, 80*160 三种, 默认分辨率 128*128 ; 使用模拟 SPI 可以任意选择引脚 (没有已经用作其它用途) 连接, 只要保证实际和程序设置的引脚一致即可。属于外围设备初始化的指令应放在“系统应用初始化”内。

2. 彩屏显示颜色模式切换(RGB/BGR) 0

用于设置彩屏显示的颜色模式为 RGB 还是 BGR (BGR 与 RGB 基本相同，除了区域顺序颠倒。红色占据最不重要的区域，绿色占第二位（静止），蓝色占第三位)，0 就是 RGB 模式，1 是 BGR 模式。

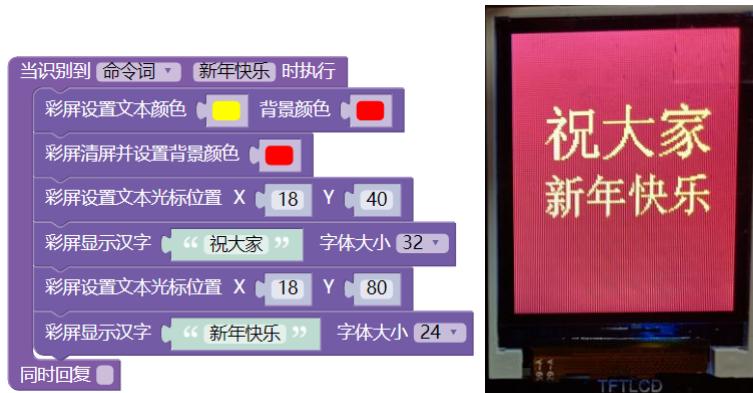
3.  彩屏设置显示方向 **0°**
4.  彩屏清屏并设置背景颜色
5.  彩屏设置文本颜色 **背景颜色**
6.  彩屏设置文本光标位置 **X 0 Y 0**
7.  彩屏显示汉字 **“好好搭搭” 字体大小 12**

五、范例程序讲解

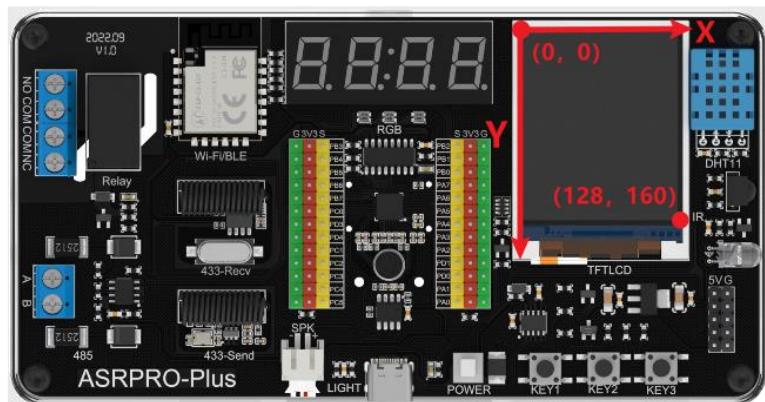
在范例 2.11 的基础上添加新的命令词触发事件，如显示黄色背景：



在彩屏上显示“祝大家新年快乐”，显示效果如右图所示：



注意：如要显示多行汉字，需先设置文本光标位置，再设置显示的汉字内容和字体大小，彩屏的屏幕坐标系如下：



附录一：语音识别的注意事项

关于中英文语音识别还有一些注意事项，这里给出以下使用建议：

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

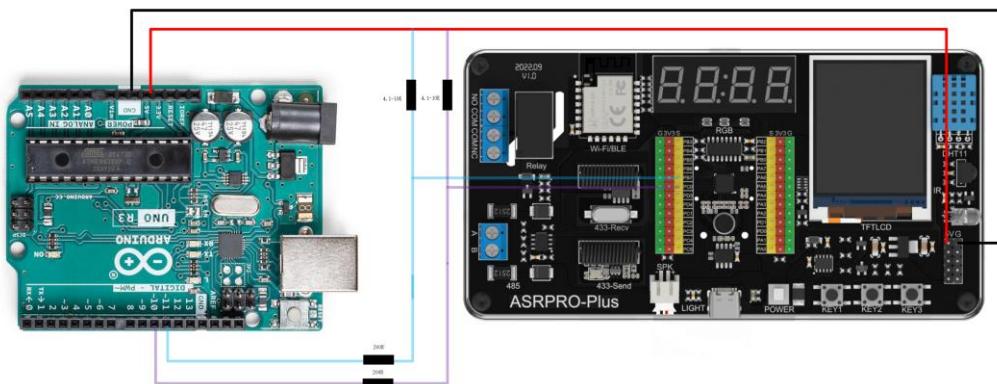
ASRPRO 有 3 组串口，UART0 预留为程序升级接口，方便后期升级。如需和其它 MCU 通讯建议使用 UART1 或者 UART2。

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：



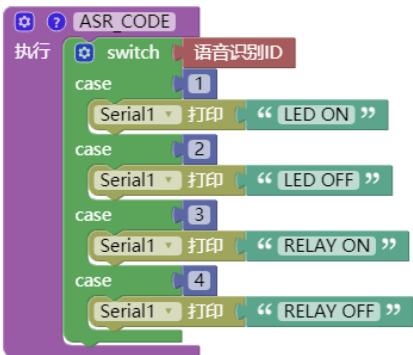
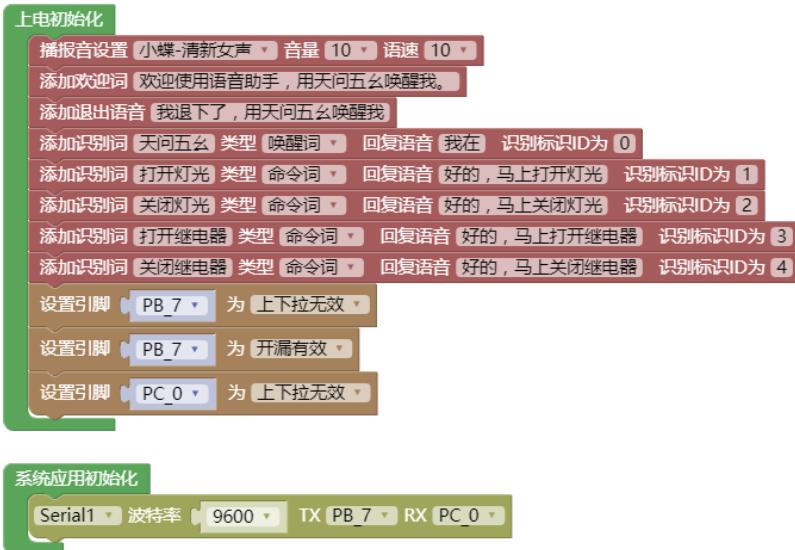
二、Arduino UNO (5V 单片机)

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
    }

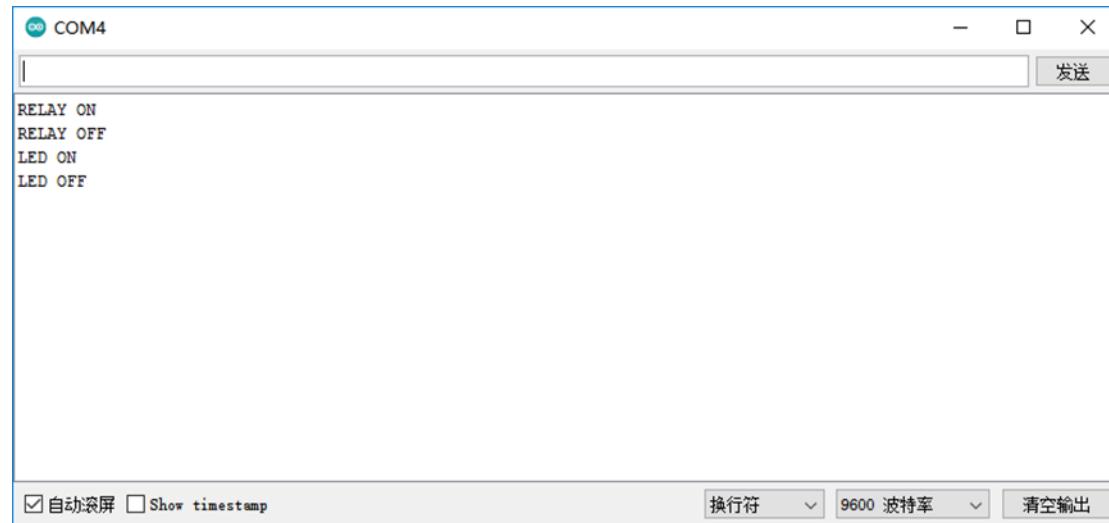
    mySerial.begin(9600);
    pinMode(LED_PIN, OUTPUT);
    pinMode(RELAY_PIN, OUTPUT);
}
  
```

```
}

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);
        }
    }
}
```

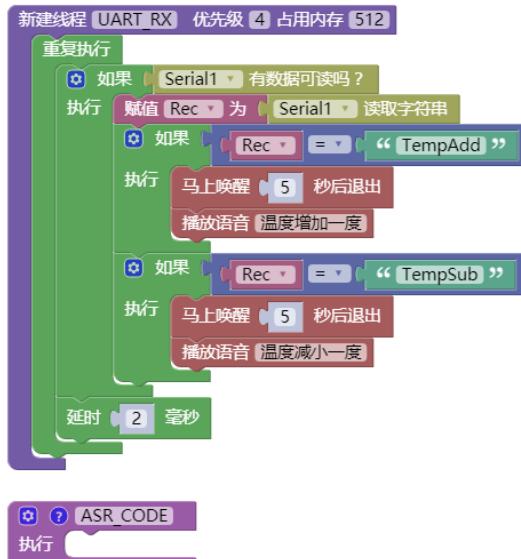
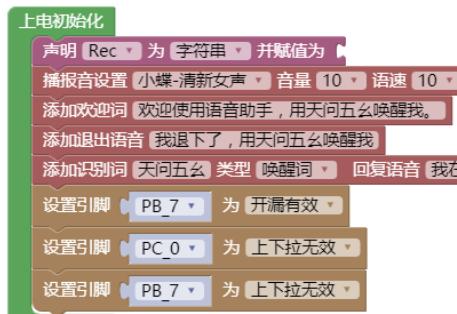
3) 程序效果

通过用“天问五么”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);
}

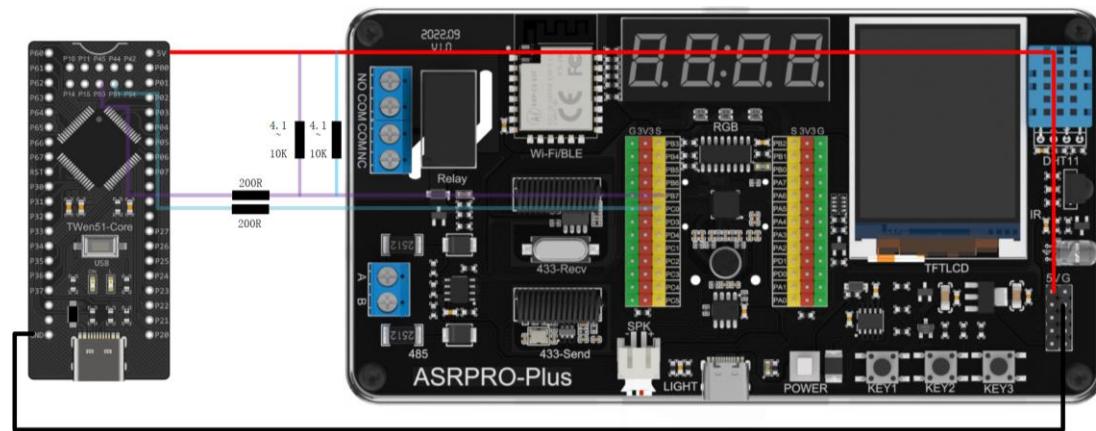
void loop() { // run over and over
    mySerial.print("TempAdd");
    delay(5000);
    mySerial.print("TempSub");
    delay(5000);
}
```

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)

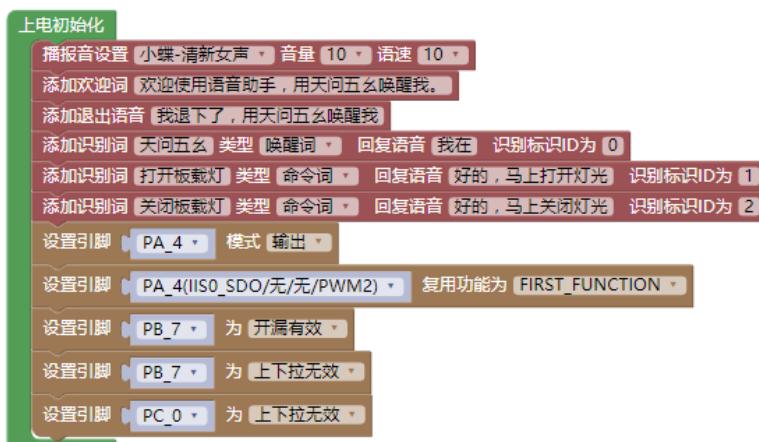
1. 电路连接

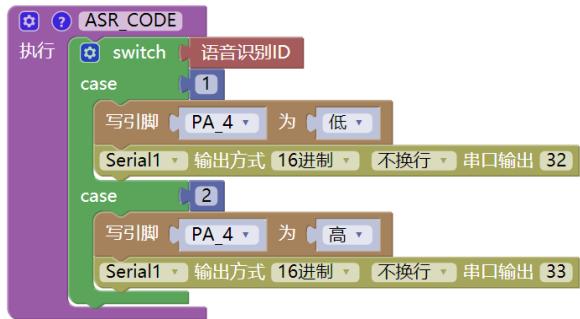


本案例以 P5_0 为 RX，P5_1 为 TX 举例。P5_0 与 ASRPRO 的 TX 连接，也就是接在 PB7，P5_1 与 ASRPRO 的 RX 连接，也就是接在 PC0，注意 STC8 的电源插脚接到 5V，GND 引脚互相连接。

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



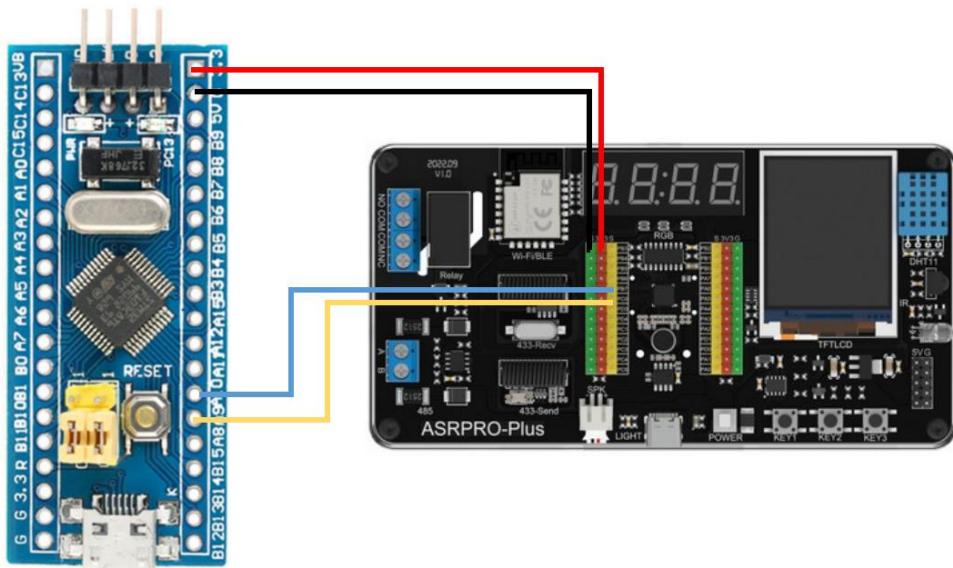


2) STC8 端程序



四、STM32 (3V 单片机)

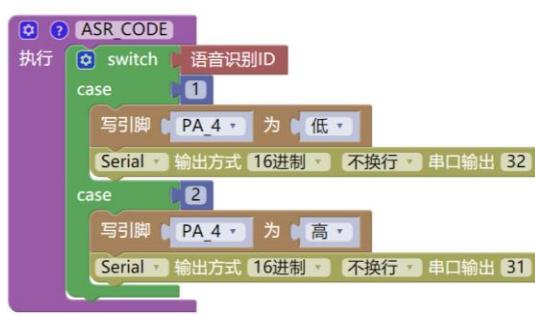
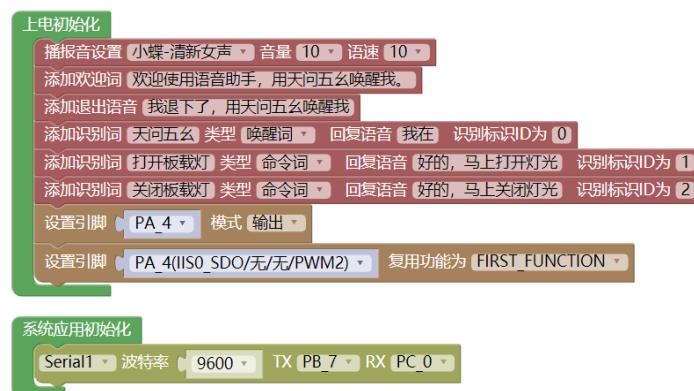
1. 电路连接



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

```
#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)
    {
    }
}
```

uart.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_RXE)==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; // 波特率设为 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);
}

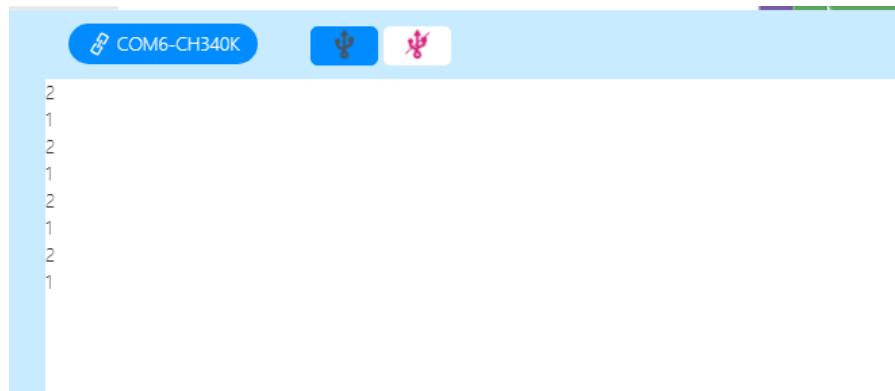
void LED_OFF(void)
{
    GPIO_SetBits(GPIOB,GPIO_Pin_10);
}

void LED_ON(void)
{
    GPIO_ResetBits(GPIOB,GPIO_Pin_10);
}

```

3) 程序效果

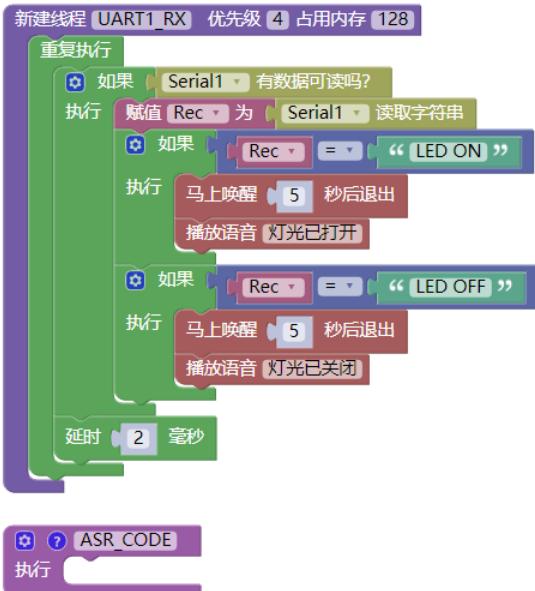
通过用“天问五么”语音唤醒后，分别说测试语音“打开灯光”、“关闭灯光”，STM32 端接收到串口命令后会执行对应引脚的控制和串口打印，“2”代表打开，“1”代表关闭。



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);
    }
}

```

```
}
```

```
uart.c
```

```
#include "uart.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_RXE)==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');

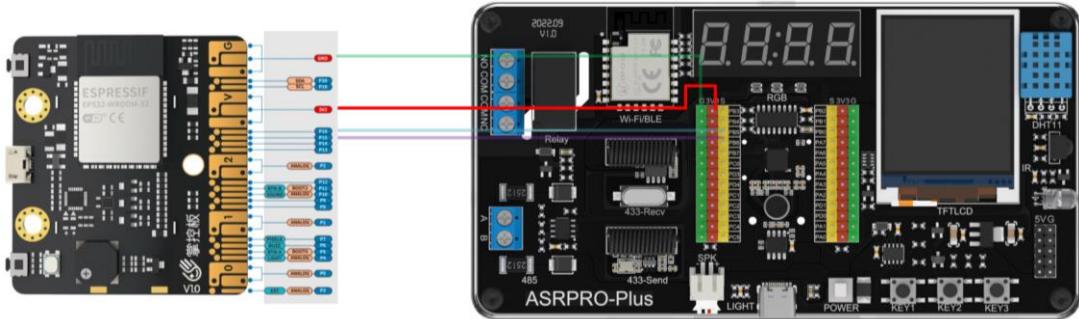
    /* 等待发送完成 */
    while(USART_GetFlagStatus(pUSARTx,USART_FLAG_TC)==RESET)
    {}
}
```

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

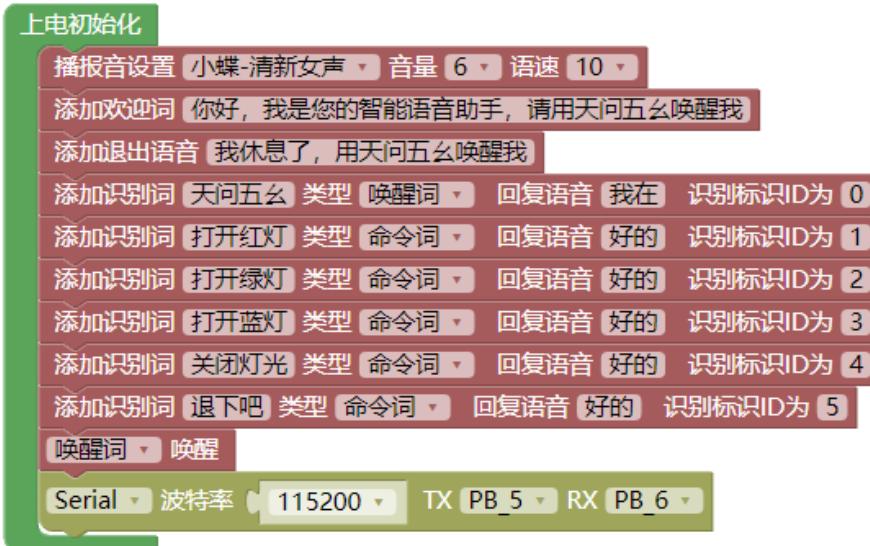


掌控行板的 P15 引脚的 TX，接 ASRPRO 的 RX，也就是接在 PB6；P16 引脚接到 ASRPRO RX 引脚（PB5），两者的 3V 引脚互相连接，GND 引脚互相连接。

2. 范例：ASRPRO 与掌控行板进行串口通讯

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

1) ASRPRO 端程序



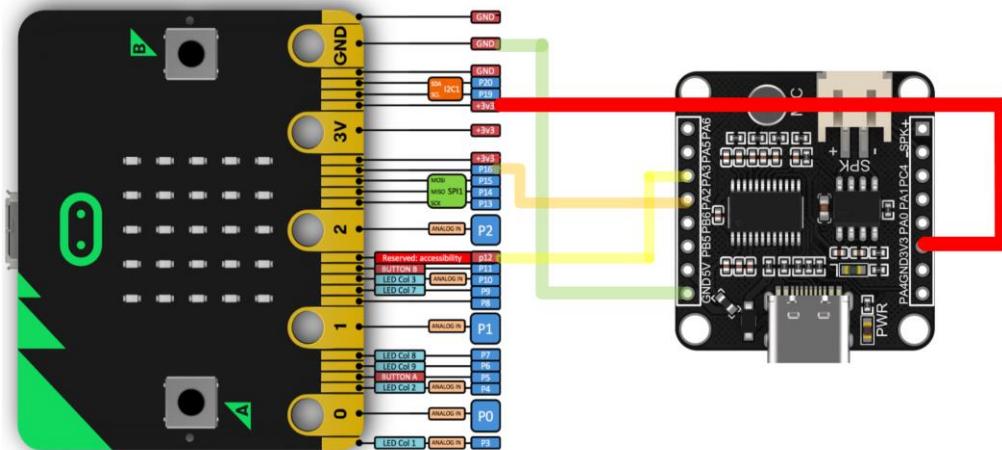
```
执行
switch [语音识别ID]
case 1
    Serial 打印 (自动换行) “id=1”
case 2
    Serial 打印 (自动换行) “id=2”
case 3
    Serial 打印 (自动换行) “id=3”
case 4
    Serial 打印 (自动换行) “id=4”
case 5
    马上退出
```

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 v] 串口 uart1 读取数据 转字符串
    将变量 id 设定为 [从文本 ck 取得一段字符串自# 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 v] RGB 灯颜色为 [红色]
    如果 [id = “2”]
        执行 设置 [所有 v] RGB 灯颜色为 [绿色]
    如果 [id = “3”]
        执行 设置 [所有 v] RGB 灯颜色为 [蓝色]
    如果 [id = “4”]
        执行 关闭 [所有 v] RGB 灯
    等待 [100 毫秒]
```

六、micro:bit (3V 单片机)

1. 电路连接



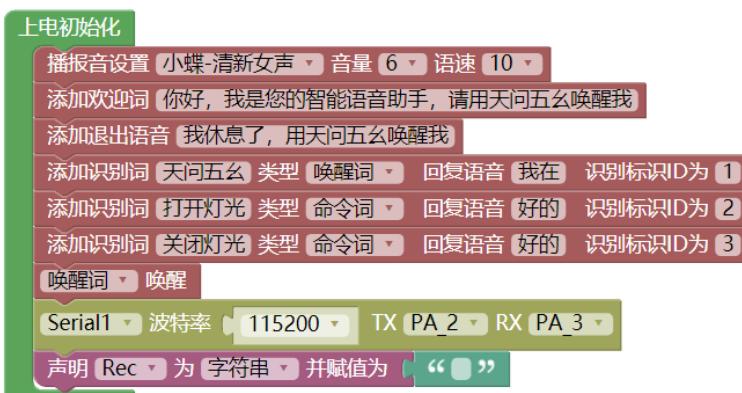
在进行串口通信时，两个设备进行双向通信，此时两个设备的 RX 和 TX 要交错连接。例如 micro:bit，定义 P16 为 RX 口，则要接到 ASRPRO 的 TX 上，也就是 PA2。

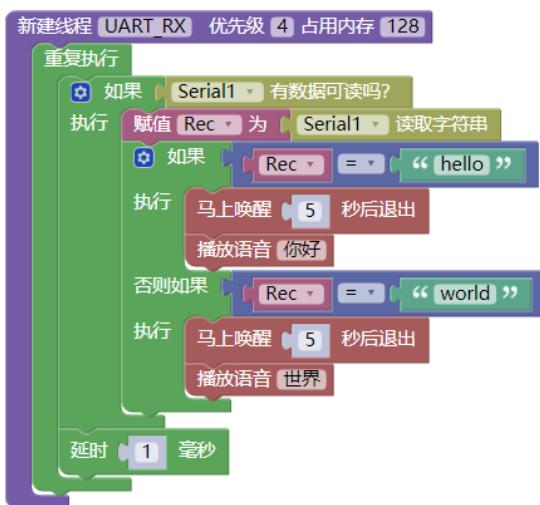
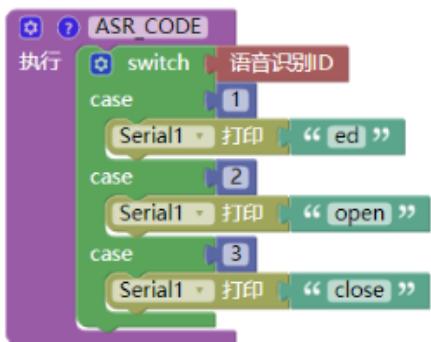
micro:bit 的 P16 引脚的 RX，接 ASRPRO 的 TX，也就是接在 PA2；P12 引脚接到 ASRPRO 的 RX 引脚（PA3），两者的 3V 引脚互相连接，GND 引脚互相连接。

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

