三易串口屏

开发指南

官方网站： http://www.sany-semi.com/

淘宝店铺： https://shop149736421.m.taobao.com/

版本修订记录：

版本	日期	修订内容简要	编写	审核
V1.0	2022-11-15	初始版本		
V1.1	2023-02-12	新增扩展 IO 说明 modbus 实验说明补充 新增 can 过滤器设置详细说明		
V1.3	2023-08-22	新增系统变量说明		

目录

前言

三易串口屏是深圳市三易半导体技术有限公司自主研发的产品，具有三个原则：**简易，变易，不易**。

简易：用户上手简单，提供电脑端编辑软件，解决客户用液晶显示屏时，写驱动，做界面，调接口不方便的问题。

液晶屏有不同的尺寸，不同的驱动芯片，不同的分辨率，不同数据接口，不同显示视角，不同的玻璃面板厂家，需要把这些元素统一起来，按照不同液晶芯片的初始化序列，还要调整显示效果，用户使用液晶屏时需要研究的方面太多，造成使用不便。

串口屏是近年比较流行的使用方式，就是用 Uart 串口进行指令控制，使液晶显示简单化，但是通讯指令过多过频，使用时也会造成不便。三易串口屏可以使用电脑端编辑软件—Visual Pix 编辑客户需要的显示效果，大大减少了串口通讯次数，整个显示过程用不同的组态控件联动起来，使开发更简单，控制更简单，让需要快速开发的客户节省开发时间，节省开发费用，迅速升级产品。

变易： 三易串口屏使用了两个不同主控芯片，可以适应 1.2~10 寸的液晶屏，方便客户定制开发，如果用户需要，可以迅速“变”出各种尺寸的串口屏模块。

不易： 三易串口屏没有根据提供的功能的不同而分类，没有命令屏，所有的模块都可以使用电脑端软件，所有的模块都把能做到的功能开放给用户，没有利用功能的支持范围来分不同的版本，这是三易串口屏不易的原则。

第一章 入门指南

1. 产品收货

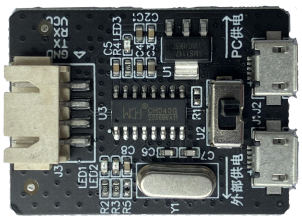
在收货后，检查模块外部硬件是否有明显损坏，所购买的产品和配套配件是否有漏发，如有疑问，请联系销售方售后。

标准产品及配件：

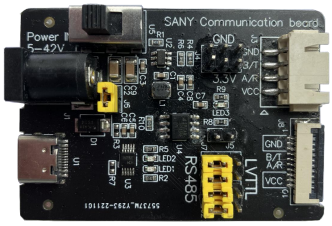
- **对应的显示模块**

- **USB 下载线及 4Pin 通信线**

- **转接小板**



或者



串口屏模块分为 G 系列和 S 系列，系列对比：

	G 系列	S 系列
包含产品	1.54 / 1.9 / 2.4 / 2.8 / 3.2 / 3.12 / 3.5	3.71 / 4.0 / 4.3 / 5.0 / 7.0 / 10.0
分辨率	240x240 / 170x320 / 240x320 / 320x480.....	480x272 / 800x480 / 1024x600.....
功能支持	音频、指针、GIF、IO、曲线.....	视频、指针、GIF、IO、曲线.....
通讯协议	RS485、LVTTL、ModbusRTU、自定义协议	RS485、LVTTL,RS232、ModbusRTU、CAN、自定义协议
外扩	蓝牙、WIFI、IO.....	

2. 串口屏模块供电

1. USB 线供电：

- 模块设计的 USB 口，连接时，一端连接串口屏模块，一端插笔记本/机箱 USB 口，此时为 5V 供电，供电后屏幕点亮，模块中有自带的出厂工程，可触摸屏幕，检查屏幕是否能正常使用。
- 此 USB 线主要用于屏幕工程下载，相对于串口下载，速度更快。
- 若使用 USB 线连接后屏幕没有正常启动（无工程画面显示，或者屏幕无法触摸），可换用手机充电头供电，或者尝试更换 USB 线，以此保证屏幕能正常使用。

2. 串口供电：

- 用 4P 线一端连接串口屏模块，一端连接转接板，usb 线一端连接转接板 PC 供电一侧，另一端接笔记本/机箱 USB 口，开关拨到 PC 供电一侧，此时为 5V 供电，供电后屏幕点亮，模块中有自带的出厂工程，可触摸屏幕，检查屏幕是否能正常使用。
- 若有单独电源供电，可自行连接串口处的 VCC、GND 供电，供电范围为 5-38V。
- 注意：老版本 USB 口为 Micro-USB 的模块，外部单独电源串口供电时，不要连接 USB 线；新版为 TYPE-C 的 USB 口，可同时连接 TYPE-C 线和外部串口供电。

3. 安装 VP 上位机软件及连接测试

1. 获取安装包后，按安装引导安装 VP 软件。
   若安装过程中有系统提示病毒木马，可先关闭杀毒软件，放心安装，此软件无任何恶意程序。
2. 使用 USB 线或者串口连接电脑后，VP 软件右下角会有连接提示：

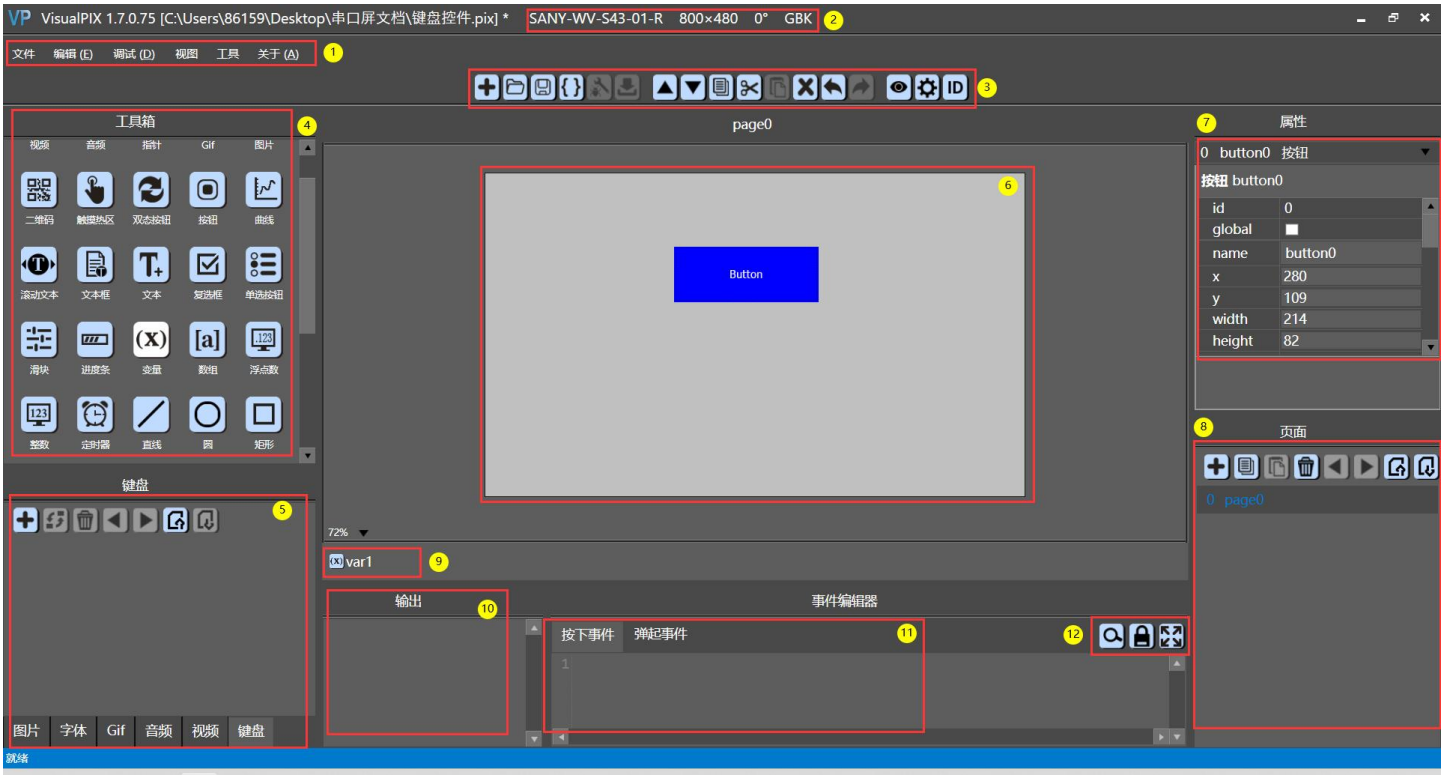
- 若提示已连接，则电脑和串口屏模块可正常通信使用；若是断开状态，可重新拔插 USB 线，或者更换 USB 口；
- 若重复尝试后，任然无法正常连接，请打开设备管理器，查看是否有端口存在，若没有端口存在，则考虑以下情况：
  串口始终无法连接模块时，系统是否有串口驱动（CH340 驱动）；
  USB 始终无法连接模块时，可去官网下载 USB 驱动。

3. 若提示已连接，则电脑和串口屏模块可正常通信使用。

4. VP 上位机简介

此上位机软件为我司自主研发，为开发串口屏模块显示界面使用，软件操作简单易上手。详细使用教程介绍请翻阅下一章。

整体界面如下：



① 菜单栏。

② 工程型号、尺寸信息、编码信息。

③ 功能快捷操作键，控件间堆叠关系变化，复制/剪切，粘贴，删除，只对面板里的控件操作有效，无法作用于资源、页面栏，可跨页面复制/粘贴控件。

④ 控件面板，控件是实现不同功能的基础，有可见控件和不可见控件之分。鼠标左键点选拖入中间画布即可。不同型号尺寸的屏幕能支持的控件不相同。

⑤ 资源面板，操作各类资源的导入、替换、删除。

⑥ 工程显示操作面板，实际效果显示区域，也是最终显示屏的显示效果。

⑦ 控件/页面属性设置面板，属性可在面板直接设置，可脚本间接控制，从而实现想要的功能和效果。

⑧ 工程页面，一个工程可以有多个页面，在此区域完成新建、删除、导出、导入页面等操作。

⑨ 不可见控件显示区域，不可见控件有：定时器控件、变量控件、数组控件、协议解析器控件、io 控件等等。

⑩ 编译结果显示面板（工程报错、工程大小等）双击报错可定位到错误控件事件。

⑪ 控件/页面事件脚本编辑面板，由代码实现功能的区域，脚本编写方式、规则和功能的实现可参考示例工程。

⑫ 脚本编辑面板的便捷操作按钮，从左到右依次为：脚本内字符查找、锁定为当前控件事件（点击其他控件，事件编辑区不会切换）、脚本事件编辑区放大。

使用上位机软件，显示各种图片，不同国家的文字，播放音频和视频，以及不同协议的支持，当然，VP 软件开发出的效果，最终会所见即所得的展示在串口屏模块上，最终成为用户产品。整个产品的功能，以控件为基础，通过脚本，操作控件属性，完成或简或繁的功能，实现智能人机交互。

5. 如何制作一个简单的工程

目标：通过 VP 软件制作，添加字库，下载到屏幕，显示“hello world 你好世界！”

需要用到的：

- 一块合适尺寸的三易串口屏
- 串口屏和电脑之间通信用的 USB 串口转换器、USB 线
- 电脑：win7 及以上，内存无要求，正常使用即可。

5.1. 安装驱动（若能正常连接，跳过这一步）

显示屏下载文件有两种方式，分别需要不同的驱动：

1.安装 USB 串口转换器驱动（例如 CH340 串口驱动），默认安装即可。

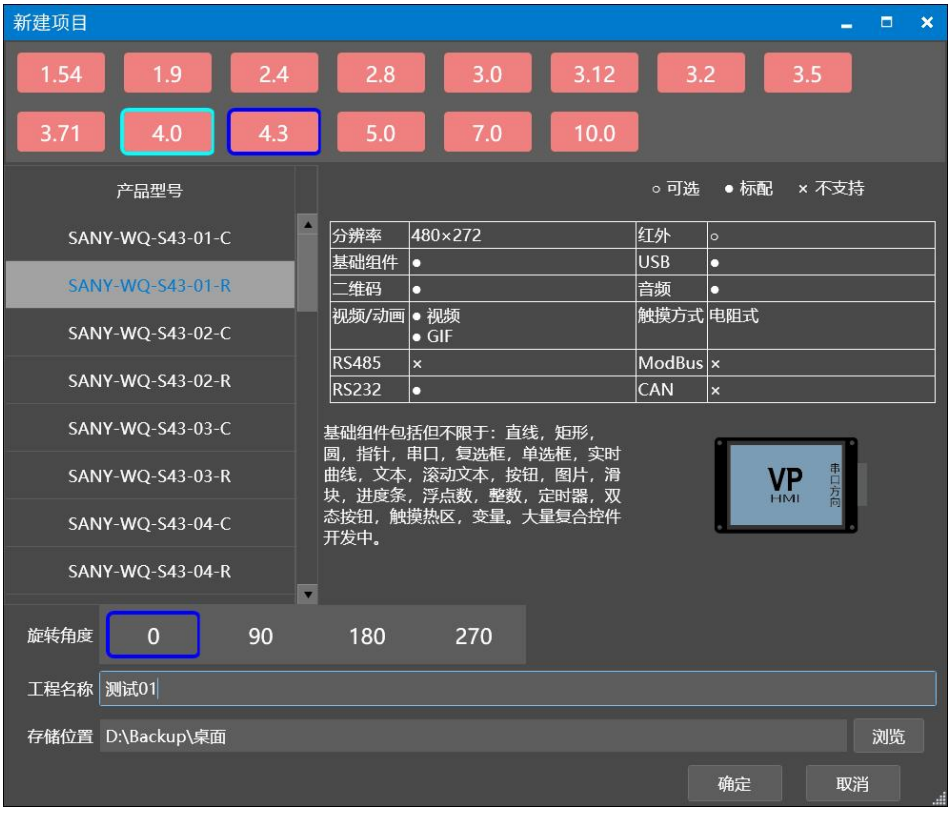
2.安装显示屏 USB 驱动。一般 Win10 系统自带，win7 系统请到 http://www.sany-semi.com/list-108-1.html 下载。

判断是否需要安装驱动：

插上 USB 或者串口后，VP 右下角有提示是否连接，或者计算机的**设备管理器**是否有端口存在，若不存在，则需要安装驱动。若能正常连接下载，可跳过这一步。

5.2. 新建工程

打开 VP 软件。文件->新建项目(或者快捷图标，快捷键 ctrl+n)



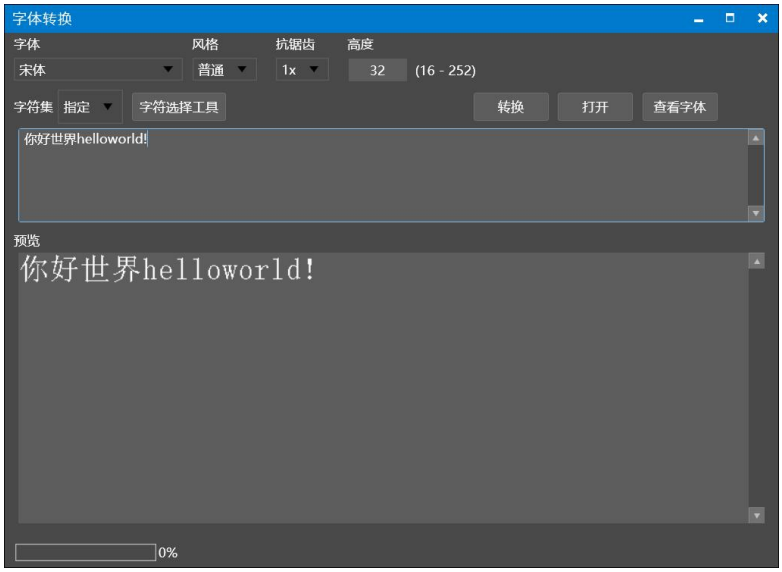
选择串口屏对应型号、选择旋转角度(显示方向)，输入工程名称，选择工程存储位置，最后点击确定，出现一个初始化空白的页面。

5.3. 添加字库

VP 左上角 工具—>取字模，进入字库生成面板。

选择字体、风格、抗锯齿和字体高度，字符集选择指定字符。

快速添加全部 ASCII 码：点击‘字符选择工具’—>勾选‘基本拉丁字母’—>确定。



输入要生成的字符“你好世界 helloworld!”。

点击转换，选择保存字体的路径和文件名，然后确定。

提示转换成功，点击“是”，然后关闭字体转换面板，返回工程编辑面板。

可以看到左下角字体资源栏增加了一个汉字字库。新建一个工程时，会自动生成一个 Tahoma_4x_ASCII 字库。

字符的风格和抗锯齿：

普通方式，增加字体丰度，消除一下字体，优点是，不增加字体存储空间；加粗，增加字体粗细，不增加字体存储空间。

1X：无抗锯齿，在串口屏上显示会有明显的锯齿感；2X，两倍抗锯齿，效果介于 1X 和 4X 之间，但空间占有率比 4X 小；

4X：4 倍抗锯齿，优点是，字体通过补偿灰度来实现字体拐弯处的流畅显示，缺点是字库大小会大 4 倍。

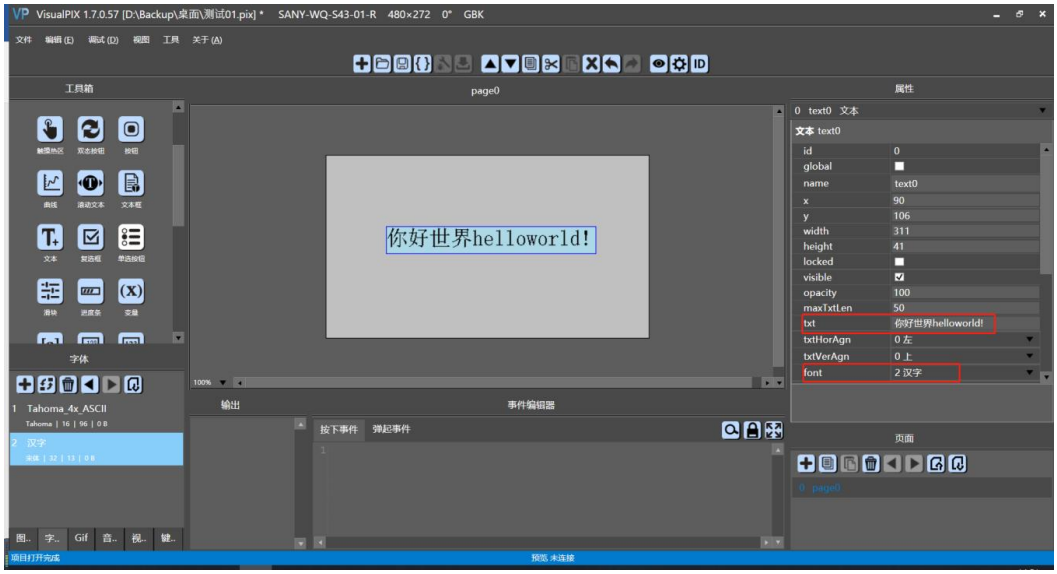
字符集选择“全部”，会根据字体来生成有哪些字符，例如宋体，就包含了几万个字符，同时内存也会占比很大。

- 字库生成导入后，也可在字体资源面板**双击字体名称**，进入取字模面板进行**二次编辑添加修改文字**。

5.4. 添加控件

在工具箱里拖动一个文本控件到中间空白的区域。选中这个文本控件，在右侧属性栏修改 txt 属性为“你好世界 helloworld!”，字体选择刚才新建的字库“汉字”

调整文本控件的宽度 width 和高度 height(可以选中控件，直接拖动或者修改右侧属性栏)使之能够完全显示



点击编译按钮，左下角输出区域提示编译成功既可。

5.5. 正确连线

下载方式有两种方式，串口下载和 USB 下载

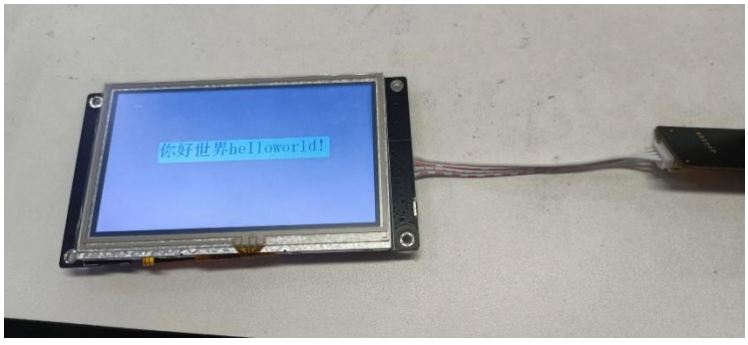
连上屏幕，若驱动正常，VP 软件右下角提示“COM 口已连接”。否则否则检查驱动是否安装成功

5.6. 下载到屏幕

编译通过后，点击下载按钮。串口下载时需选择波特率（显示屏出厂样例工程是 115200，若有修改，按选择实际波特率），USB 下载默认 115200 即可。

点击“确定”，下载成功，如下提示：

显示屏正常显示。



5.7. 下载失败常见原因

串口下载失败

原因	解决办法
供电不足。表现为黑屏、闪屏、花屏	换个供电电流大的 USB 口、换个质量好的 USB 线
未安装串口驱动。表现为电脑设备管理器未发现 COM	安装串口驱动，三易的下载小板为 CH340。一般 win10 自带该驱动。
波特率不正确	重新选择正确的波特率。出厂工程样例波特率为 115200
串口接线错误或串口不通。供电足够，也发现了 COM 口	用电脑串口助手，给屏发送查询命令 version(加换行)，若没有收到串口屏的版本信息，则串口不通。检查并调整串口接线，直到发送 version(加换行)有收到版本信息才行。

USB 口下载失败

原因	解决办法
供电不足。表现为黑屏、闪屏、花屏。	换个供电电流大的 USB 口、换个质量好的 USB 线
未安装 USB 驱动。表现为电脑设备管理器未发现 COM	安装 USB 驱动，一般 win10 自带该驱动。 驱动下载地址 http://www.sany-semi.com/list-108-1.html

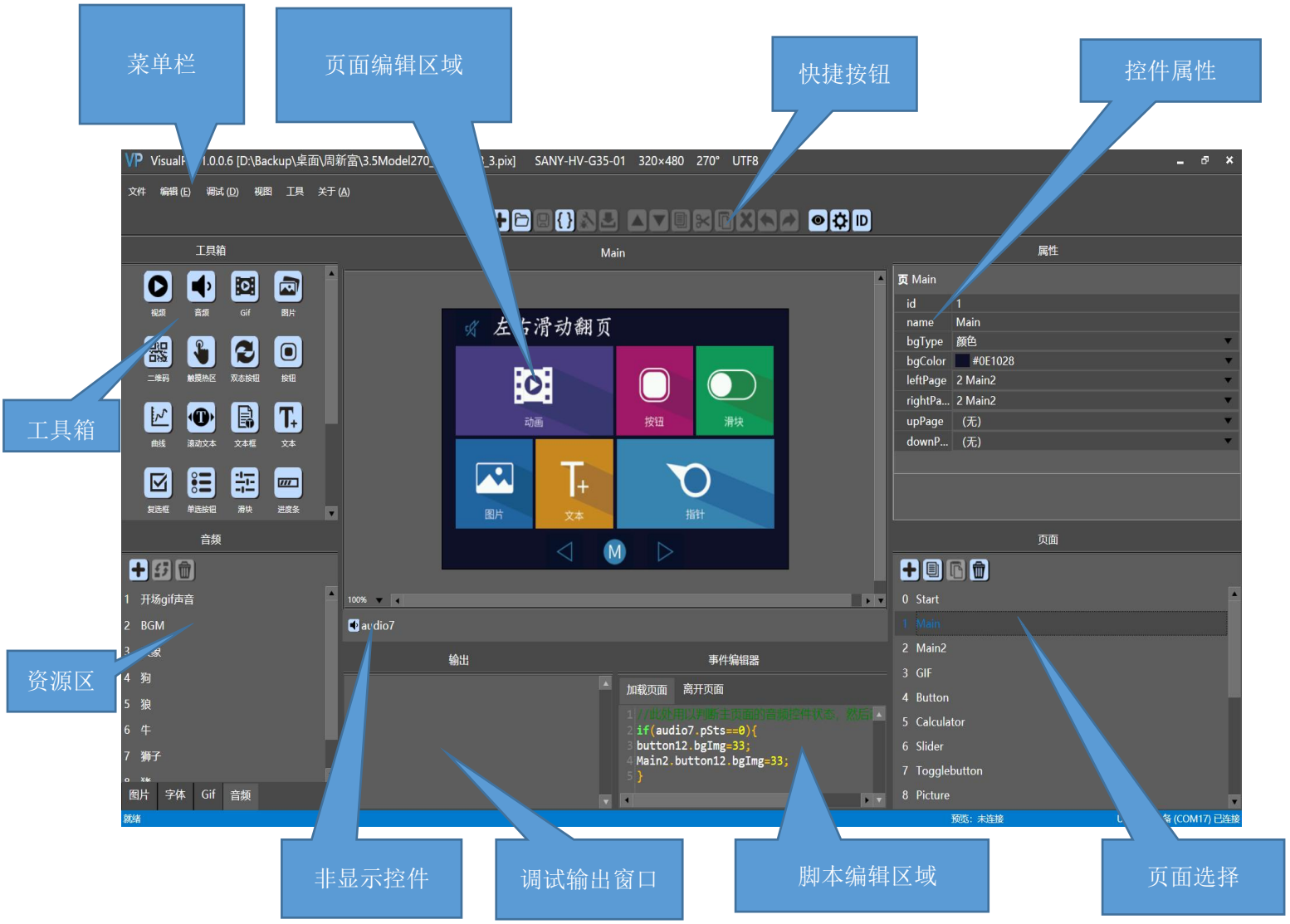
第二章 三易串口屏开发环境 VP 介绍

1. VP 界面

Visual Pix 开发环境（以下简称 VP），适用于 win7 及以上 windows 操作系统，支持 UTF-8 语言系统，默认为中文编辑界面。

VP 开发环境采用目前流行的配色方案，偏深灰色，可以更加突出用户编辑区用户使用的各种美丽的颜色和图片。常用工具和功能都放在尽量方便的地方，方便设置与取用。

以下是整体界面风格：



2. 菜单栏

2.1. 文件 - 工程文件操作

菜单项	说明
新建项目	新建一个串口屏工程项目，快捷键 Ctrl+N
打开项目	打开一个串口屏的工程，快捷键 Ctrl+O
关闭项目	关闭当前串口屏工程。
保存项目	保存当前串口屏工程。
退出	退出 VP 编辑器。

新建工程项目界面如下：

SANY-WQ-S43-01-R：分辨率为 480x272，S 系列的 4.3 寸，通讯接口为 RS232，电阻触摸

产品型号编码规则：SANY-液晶分辨率缩写-系列代码+尺寸-硬件版本号+触摸类型

名称	说明	举例
SANY	产品前缀	
液晶分辨率	液晶行业约定俗称	例如：WQ:480X272 WV:800X480 WS:1024X600 WS:1280X720
系列代码	G 或者 S	G 系列：1.54、2.4、2.8 等； S 系列：4.3、7.0、10.0 等。
尺寸	24->2.4 寸，70->7.0 寸	
硬件版本号	通讯方式，2 位编码	01：RS232/02：RS485/02M：RS485 带 Modbus /04：Ivttl 带扩展
触摸类型	N 无触摸，R 电阻触摸，C 电容触摸	

2.2. 文件 - 编辑项目（工程型号更改）

说明	**将一个型号的工程改为另一个型号，此功能可以更改已有工程的一些设置，如尺寸、产品型号、旋转角度。**
操作简述	在已创建/打开的工程中，点击文件-->项目编辑，即可根据需要调整工程设置；
注意事项	● 选择为已有工程尺寸的其它尺寸时，变化的只是操作界面，工程中的素材尺寸不会改变； ● 选择不同的旋转角度也是一样，素材的尺寸和方向不会改变； ● 整个项目编辑的结果是可逆的，设置变成其他型号看到效果后，还可以再次使用项目编辑功能，恢复原状；

界面如下：

2.3. 编辑

菜单项	说明
复制控件	复制功能内的一个控件，快捷键 Ctrl+C
剪切控件	剪切功能内的一个控件，快捷键 Ctrl+X
粘贴控件	粘贴功能内的一个控件，快捷键 Ctrl+V
删除控件	删除功能内的一个控件，快捷键 delete
撤销操作	撤销上一步操作，快捷键 Ctrl+Z
重做操作	重做功能内的一个控件，快捷键 Ctrl+Y

2.4. 调试

菜单项	说明
编译	编译当前工程，编译不通过，则输出面板输出错误，快捷键 Ctrl+B
下载	下载当前工程到串口屏模块。 插入 USB 数据线后，软件右下角显示连接，点击即可下载。 插入 USB 转串口的数据线，会弹出选择端口的窗口，选择正确的端口后，点击确定，开始下载。
调试	启动工程ＶＰ模拟器进行工程模拟运行，可以在ＶＰ软件中模拟串口屏工程的运行，方便客户调整工程中控件和脚本。 可以连接串口屏，进行上下同时运行工程，实现仿真。 可以发送串口命令到ＶＰ模拟器，可设置模拟器接收和串口接收（由于模拟客户运行状态，此时不可插入 USB 数据线，只能通过 USB 转串口设备通信）

2.5. 视图

菜单项	说明
重置窗口布局	恢复窗口布局到默认状态，开发过程中，可能把各个工作区大小拖拽得比较乱，可用此键重新恢复默认布局。

2.6. 工具

2.6.1. 函数编辑器（自定义函数）

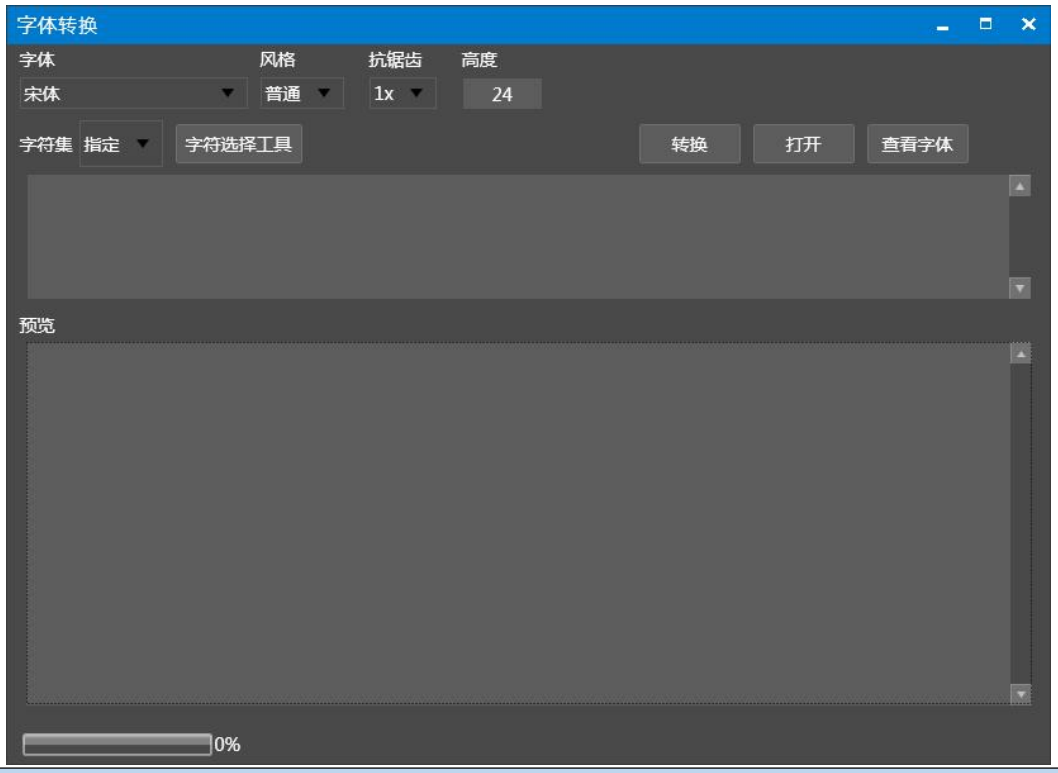
使用步骤	工具->函数编辑器，进入编辑界面，根据需求定义函数

函数的定义	**同 C 语言。**
	返回类型 函数名（类型说明 变量名,类型说明 变量名,…）
	{
	函数体；//一条或多条语句
	}
	 
	
	1.函数体内操作控件属性时，需要控件勾选 global 属性，并且加上控件所在页的页名称；
	2. 函数之间都是平行的，无主函数子函数的区别，不支持嵌套定义；
	3.定义好函数后，先点击测试，保证函数无误（即找到 n 个函数），然后点击确定退出；

函数的调用	事件编辑器
	按下事件 弹起事件 +
	1 Transform();//调用函数
	2
	3 num3.val = add(num1.val,num2.val);//函数调用，将控件值相加
	4
	1.在某个脚本事件中输入函数调用，且可在任意页面不限次数调用；
	2.可用控件属性值作实际参数，传递给函数；
	3.不支持嵌套调用；

2.6.2. 取字模（生成字库）

菜单栏：工具->取字模，打开转换工具，字体转换工具的界面如下：



操作步骤

1. **字体：**选择一个字体
2. **字符集：**选择指定字符，可输入预览自己需要的字符；选择 ASCII，生成全部 ASCII 码字符；不建议选择全部，会有过多不需要的字符占据空间
3. **字符选择工具：**里面内置了许多其他语言字体，这一步可跳过
4. **风格：**普通和加粗，按需求选择
5. **抗锯齿：**选择 2x 或 4x 抗锯齿，优点是，字体通过补偿灰度来实现字体拐弯处的流畅显示，缺点是字库大小会大 2、4 倍。

效果如下：

不抗锯齿　　4 倍抗锯齿


建议如果不是太多字，考虑用 4X。

6. **高度：**即字体的大小

7. 选择完成后，**点击转换**，给字库命名并选择字库存储位置，最后会弹窗提示是否导入到工程，选择是，再关闭字库工具面板，回到工程使用

注：

- 在 VP 软件字体资源栏中双击字体的名称，可以查看字体转换效果，修改并再次转换字库
- 为方便用户使用字体，内嵌了字体转换工具。提供 windows 字体转换成工程用的字库操作，由于各个串口屏 flash 存储有限，建议大家除了 ASCII 字符外，尽量少的做字库

2.6.3. 固件下载

简介	串口屏固件，类似串口屏的操作系统，是执行串口屏工程的平台，运行在串口屏模块的主控芯片中。串口屏模块的固件，一直在不停的升级中，修改 bug，增加功能，增加配置，都需要固件支持。
下载接口	G 系列（2.4、3.5 等）只能用串口升级，S 系列（4.3、7.0 等）可以用 USB 或者串口升级。
下载方式介绍	有三种情况： 1. 工具菜单栏->固件下载 由产品方或供应商直接提供固件给用户，用户保存到本地后，用此方式下载，属于离线更新； 2. 工具菜单栏->在线固件下载 官方有新固件需要更新时，会更新服务器上的固件，便于用户在线更新； 3. 下载串口屏工程时，若有新固件，会提示有新固件可在线更新；
串口屏显示提示	● 下载时，弹出固件升级提示窗口，选择‘是’进行升级，升级固件时，串口屏背景会绿屏并有下载提示。 ● 固件下载成功后断电重启，屏幕是全绿色，并且显示‘no file’等字样，此时 flash 中的工程是已被擦除状态，需要重新下载工程。 注意： 正在下载固件的过程中，断开连接或下载异常导致固件下载不成功，串口屏重新上电后屏幕可能会变成全白，此时屏幕是处于没有固件的状态，需重新下载固件。 白屏后：用串口升级下载固件时，波特率选择 115200 下载即可；用 USB 可直接升级下载，不用选择波特率。

2.6.4. 项目设置 - 波特率

菜单栏：工具->项目设置，或者点击快捷栏按钮，界面如下：

波特率可脚本更改，若想通过脚本设置波特率为 9600：sys_baud = 9600；

注：屏幕掉电后会波特率恢复到项目设置的初始选择值；

2.6.5. 项目设置 - Modbus 主从机设置

当工程为带 Modbus 的版本时，进入项目设置界面，会增加以下设置：

当串口屏做主机时，勾选‘主机’，不勾选则为从机；

地址为做从机时的地址，从机地址可脚本更改，如设置从机地址 100：sys_modbus_addr = 100；

注：屏幕掉电后地址会恢复到项目设置的填写值；

2.6.6. 项目设置 - Bin 文件生成和下载

- 生成 bin 文件：进入项目设置后

1. 勾选‘生成二进制文件’；
2. 点击保存，然后编译工程文件；
3. 工程文件的保存文件内会出现*.pix.bin 文件。

- 下载 bin 文件：

  此文件是烧录到串口屏模块中的文件，相当于编译完成的执行文件。

方法一：

使用 tf 卡：

在 TF 卡根目录下新建 HMI 文件夹，把*.pix.bin 文件，放在 HMI 文件夹中，文件名改成“**USER.bin**”。断电插入 tf 卡，再上电后会自动烧录到三易串口屏。bin 文件方式烧录工程能脱离 VP 软件，实现客户的大规模生产。

方法二：

工具菜单->二进制文件下载：

用转串口或 USB 线连接好屏幕后，点击选择 bin 文件即可下载。

此方式不用打开项目工程，不用更改 bin 文件的名称。

2.6.7. 项目设置 - 编码格式和 CRC16

- **编码格式：**可选择 UTF-8 和 GBK

应用：当用户在自己设备的单片机程序中，通过串口指令设置控件显示文字时，如：wset text1.txt “2022 星期一”，可能出现控件只显示 2022，而不显示汉字的情况；这时就需要确定单片机程序和 VP 工程的编码格式是否一致。

- **CRC16：**多项式：8005、1021、3D65

脚本计算，请参考后续**函数的介绍和用法**章节，CRC16 函数的相关操作函数。

2.6.8. 项目设置 - 掉电保存

做掉电保存的第一步，就需要先进入项目设置，开辟一段空间：

如要保存一个整数，在设置了用户 Flash 大小后，事件脚本如下：

```
fwInt(0,1);
fSave();//两个函数必须同时使用，才能完成存储操作
```

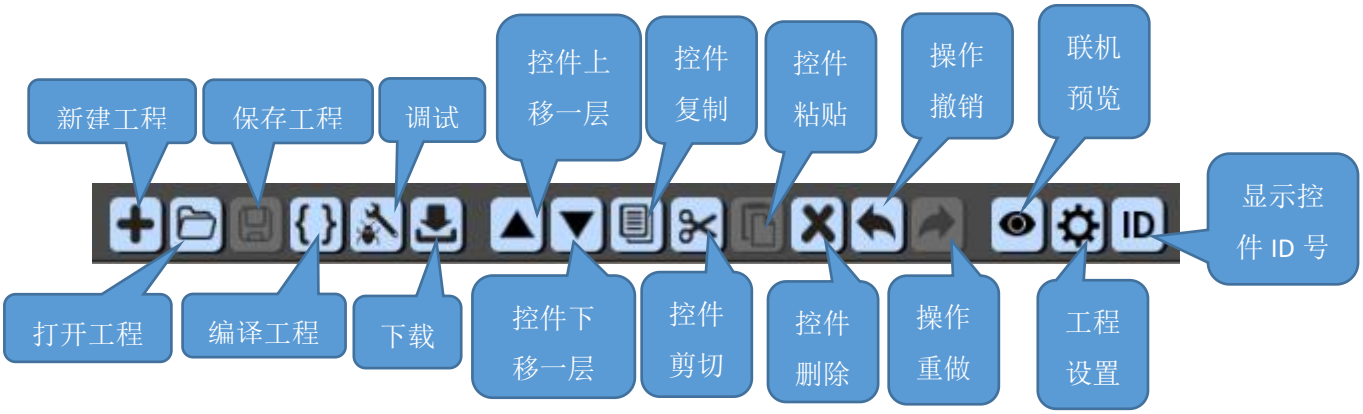
注：

Int、Float 都为 4 字节；

String 类型时，需要 **字数*2+2** 段长度；

掉电保存数据和读取数据操作，请参考后续**函数的介绍和用法**章节，Flash 的相关操作函数。

3. 快捷按钮



快捷按钮上的功能都是常用功能，如果和菜单栏重复的功能，请参考上面的介绍。

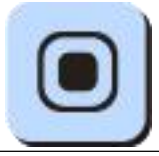
快捷键	说明
控件上移一层	为了显示效果，编辑时控件可能重合或者被遮盖，例如钟表的 3 个指针。使用这个功能可以把下层的控件移动到上层。
控件下移一层	为了显示效果，编辑时控件可能重合或者被遮盖，例如钟表的 3 个指针。使用这个功能可以把上层的控件移动到下层，使下层控件显示出来，进行设置。
联机预览	这个按钮被按下，则进入三易串口屏的特色功能。 用 usb 数据线（推荐）或 usb 转串口线，连接串口屏模块。 如果连接成功，则上位机和下位机都会显示当前的工程编辑情况。如果连接成功，VP 界面底层栏会有，“预览：已连接”字样。 用户可以在 VP 编辑区，移动控件位置，修改属性，增加减少资源，会实时同步到下位机显示。 注意，上位机此时是编辑状态，并没有运行功能，下位机是运行状态，显示当前页面真实的运行起来的显示情况，上下会有差异，这个功能主要是让初学者用户可以实时看到自己的工程在串口屏上的运行情况。
显示 ID 号	显示各个控件的名称。如果客户的页面用了大量的控件，相互遮盖交叉，还有客户书写脚本时不方便找到各个控件的名称，此时点击此按钮，就显示所有控件的名称，非常方便。

4. 工具箱

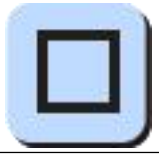
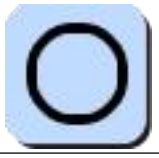
工具箱中包含了各种经常用到的控件，工具箱中的控件，点击选中拖动到编辑区的合适位置，就可以使用了。注意，非显示控件，例如定时器等，也要拖动到编辑区，才能使用，这个操作作为用户确定要使用这个控件的标定。

工具箱的控件见下图：控件会一直增加，后续会出更多的复合控件。

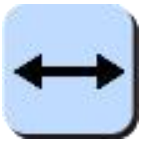
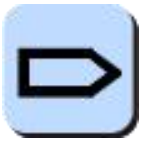
视频控件 	支持视频播放功能，默认 3.5 及以下小尺寸（下称为:G 系列）不支持，如果您需要可以定做。G 系列串口屏使用的芯片，播放速度达不到可以使用的程度，胜在性价比高。 大分辨率尺寸的串口屏（S 系列），只支持 Mp4 格式的视频播放，可以控制声音，进度，切换，可以直接作为播放器使用。 注意：视频尺寸不能超过 1280X720，声音也只支持单、双通道。 **视频功能，支持 TF 卡热插拔，直接播放。**
音频控件 	全系列支持音频播放。不支持同时播放一路以上的音频。支持停止，播放，暂停功能，可以随时获取播放进度，可以跳转到某一时刻播放，方便用其他控制和联动。 音频在 G 系列串口屏上播放时，只支持 8 位 wav 格式，全系标配 3W 功放。 音频在大尺寸（下称：S 系列）串口屏播放时，支持 MP3 格式和 16 位 wav 格式。 **音频功能，支持 TF 卡热插拔，直接播放。**

图片控件 	G 系列串口屏支持 jpeg、BMP、没有透明度的 PNG 图片，VP 会转换成二进制格式，所以图片所占空间以实际编译后为准。不支持带透明度的 PNG 图片，用户在做素材时，尽量避免用此种方式。 S 系列串口屏，支持 JPEG 解码，图片占用空间很小。支持 PNG 透明图片 G 系列图片会转换为 RGB565 格式，也就是 16 位颜色数据，支持颜色 65535 个。用户做图片时请注意，如果原图位 RGB888 格式的渐进颜色，可能会造成有水纹。这个现象是目前小尺寸液晶屏都不支持 RGB888 颜色造成，无法消除。 S 系列支持 RGB888 格式，支持 24 位颜色数据，千万级颜色。用户做图片素材时，不需要考虑颜色降低问题。
GIF 动画控件 	全系支持 GIF 动画。用户可以导入 GIF 动画到 VP 编辑界面，VP 工具会读取 GIF 动画的帧数据和时间数据，还原 GIF 动画的显示。GIF 动画目前不支持硬件解码，所以会造成编译后数据变大，用户要注意动画会占用较大 Flash 存储空间。 GIF+音频，可以完美做产品片头。
二维码控件 	全系支持动态二维码。二维码作为一个最常用的功能，支持起来比较麻烦，需要用到较大内存空间做算法。这个功能，在内存不足的情况下实现。
触摸热区控件 	类同透明的按钮，也有按下、弹起事件。触摸热区功能可以加到任意的位置，点击屏幕上的这个区域，可以触发点击事件，方便客户做一些隐形的功能。
双态按钮控件 	实现开关按钮的功能。按下后不会自动弹起，需要手动点击才能弹起。
按钮控件 	实现按钮功能。重要功能，属性多，效果也多，背景可透明（S 系列），透明后的功能大于触摸热区。
曲线控件 	曲线功能可以实现客户透传的曲线数据，为了实现快速数据传输，透传数据使用二进制格式。 默认支持 3 通道的曲线。 默认缓冲 3 通道的曲线数据，到边界后会覆盖以前数据。 曲线刷新模式有推进式、覆盖式两个可选。

滚动文本控件 	目前支持一串字符水平左右、垂直上下方向飞来飞去。经常作为提示用途。
文本控件 	用于显示文本，可选择是否自动换行，占用空间小。支持各种客户定义的字体，支持 1-100 透明度选择。
多行文本控件 	用于显示多行文本，可选择是否自动换行。如果串口屏带有触摸屏，可以拖动文本主体进行移动。
复选框控件 	未选中是空心矩形，选中为中间是黑块的矩形。特色是，复选框的触控范围可在右侧水平方向上随意调整，用户不用只点击矩形框才能选中，可以点击右侧任意有效位置选中。
单选按钮控件 	单选框为分组控件，可以设置有几个可选项，各个选项是互斥的。可以根据值改变的事件编写脚本。
滑块控件 	使用滑块可以调整进度。滑块的不同位置，会有不同的对应值，可以根据值改变编写控制脚本，例如控制音频播放，背光亮度等。
进度条控件 	进度条是被动控件，显示需要过程处理功能的进度。 进度范围值：0-100.
变量控件 	勾选 global 属性可以设置为全局变量。全局变量可以在整个工程内使用，局部变量的作用区域在本页。 变量类型分为 int、float、string 三种。 Int 为四字节整形数据；float 为四字节浮点数据；string 是 ASCII 的可见字符串。 使用变量控件，要拖动变量控件图标到编辑区域，作为有效使用标志。 变量控件可以使页面间和控件间的数据进行联动，通过脚本可以实现复杂逻辑。

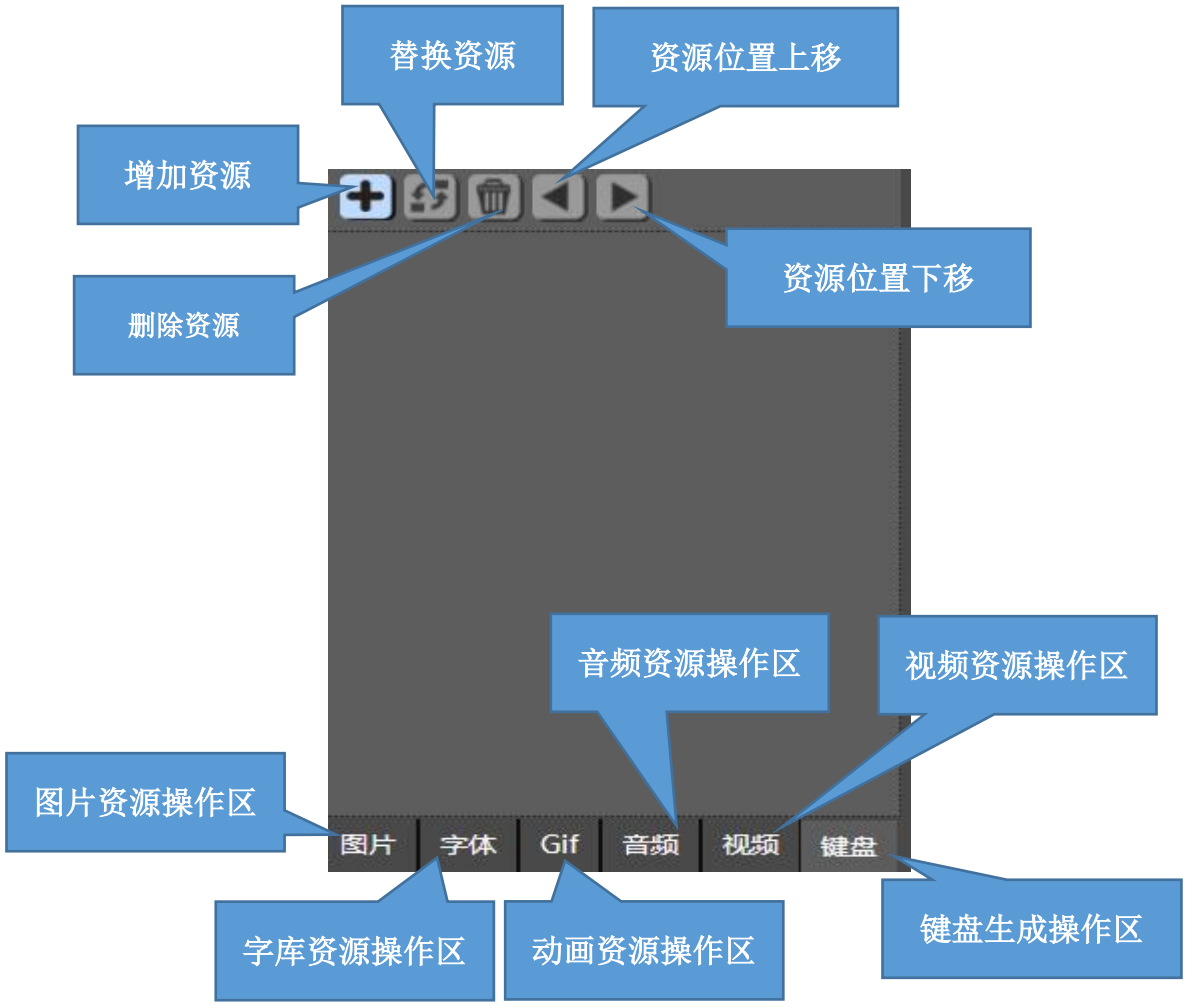
数组控件 	可以设置为全局变量和局部变量。全局变量可以在整个工程内使用，局部变量的作用区域在本页。 类型支持 Byte、Int、Float、String 四种。 使用变量控件，要拖动变量控件图标到编辑区域，作为有效使用标志。 变量控件可以使页面间和控件间的数据进行联动，通过脚本可以实现复杂逻辑。
浮点数控件 	类似于文本框，是一个浮点数的显示控件，值为 float 类型，省去把浮点数转换为文本显示的处理过程。支持正负数。
整数控件 	整数的显示控件，值为 int 类型，省去把整数转换为文本显示的处理过程。支持正负数。
定时器控件 	定时器是重要控件，页面显示的自动动作效果主要来自定时器事件。控制定时器开启、关闭和定时间隔可以完成各种自动操作。
直线控件 	在界面上画一条直线。如果是简单应用，用直线可以省去处理图片的工作，也节省存储空间。直线其实就是一个实心矩形，但是可以调整属性，使之有不同倾斜角度。
矩形控件 	在界面上画一个矩形。如果是简单应用，用矩形可以省去处理图片的工作。矩形是带有边框的矩形，线宽，颜色，填充，都可以设置。
圆 	在界面上画一个圆形。如果是简单应用，用圆形可以省去处理图片的工作。圆形是带有边框的圆形，线宽，颜色，填充都可以处理。
指针控件 	围绕中心转动的指针。目前指针的处理相当完善，中心，长端，短端的尖角角度和长度以及颜色，都可以通过配置直接生成，变化多端。并不是一条旋转的直线，不用配置样式，配置不同的参数，就可以生成几十种指针类型。配合绚丽的背景图，可以逼真实现现实中的仪表。

协议解析器控件 	用户可以对接收到的数据，自行使用脚本进行数据处理，可以接入以及处理以下的数据（包括但不限于）： 外设采集的数据，例如温度，倾角等。 用户键盘输入的键值。 红外控制信号。 用户主控芯片发来的自定义数据（用户可以做自己的 HMI 通讯协议）。 其他 HMI 的协议数据。 协议转换使用三易串口屏脚本，会使用 C 语言就可以轻松完成。 一个工程只能有一个。 作用于全局。 每接收到一次数据，控件内脚本执行一次。 **具体使用请参考后续实验章节。**
RTC 控件 	非可见控件形式，需使用系统变量在脚本内获取； **具体使用请参考后续实验章节。**
键盘控件控件 	上位机软件内置的一套键盘，包括数字、字母、符号键盘； 用户可根据需求，选择使用其中一个或全键盘； 键盘可放在任意页面，并可调整想要的尺寸； 用户可将软件自带键盘背景图替换成自己设计的图片； 各种类型键盘键可相互转换，可输入小写字母、ASCII 码全部符号、数字以及中文。
环形进度条控件 	和条形进度条的区别在于表现数据的形式上，一个水平或者垂直表示进度，而此控件是圆环进度，也可做成扇形比； 使用时需准备填充和未填充两张图片；
日志控件 	用于打印屏幕模块接收的数据，可打印显示 ASCII 码、HEX 数据；
CAN 解析器控件 	不可见控件，用法可类比协议解析器控件； 一个工程只能有一个。 作用于全局。 每接收到一次数据，控件内脚本执行一次。 **具体使用请参考后续实验章节。**
Modbus 解析器控件	不可见控件 一个工程只能有一个。 作用于全局。

	每接收到一次数据，控件内脚本执行一次。 **具体使用请参考后续实验章节。**
IO 控件 	不可见控件 **具体使用请参考后续实验章节。**
ADC 控件 	不可见控件 **具体使用请参考后续实验章节。**
PWM 控件 	不可见控件 **具体使用请参考后续实验章节。**

5. 资源区

资源区存放用户做串口屏界面使用的资源，包括 图片，字库，动画，音频，视频。资源区在 VP 编辑器的左下方。如图：



功能	介绍
增加资源按钮	点击按钮，会弹出文件夹，用户可寻找目录，选择资源。
替换资源按钮	选中需要替换的资源，点击按钮，会弹出文件夹，用户可寻找目录，替换资源。
位置上移按钮	选中资源，点击按钮，资源会上移一个位置，方便用户排列资源顺序。**注意移动位置后，资源的编号会改变，脚本和属性中的资源都是按照编号来选择。**
位置下移按钮	选中资源，点击按钮，资源会上移一个位置，方便用户排列资源顺序。**注意移动位置后，资源的编号会改变，脚本和属性中的资源都是按照编号来选择。**
图片资源操作区	点击后，可操作图片资源的增加、替换、删除、位置移动
字库资源操作区	点击后，可操作字库资源的增加、替换、删除
动画资源操作区	点击后，可操作动画资源的增加、替换、删除、位置移动
音频资源操作区	点击后，可操作音频资源的增加、替换、删除

6. 页面编辑区，输出区，事件编辑器

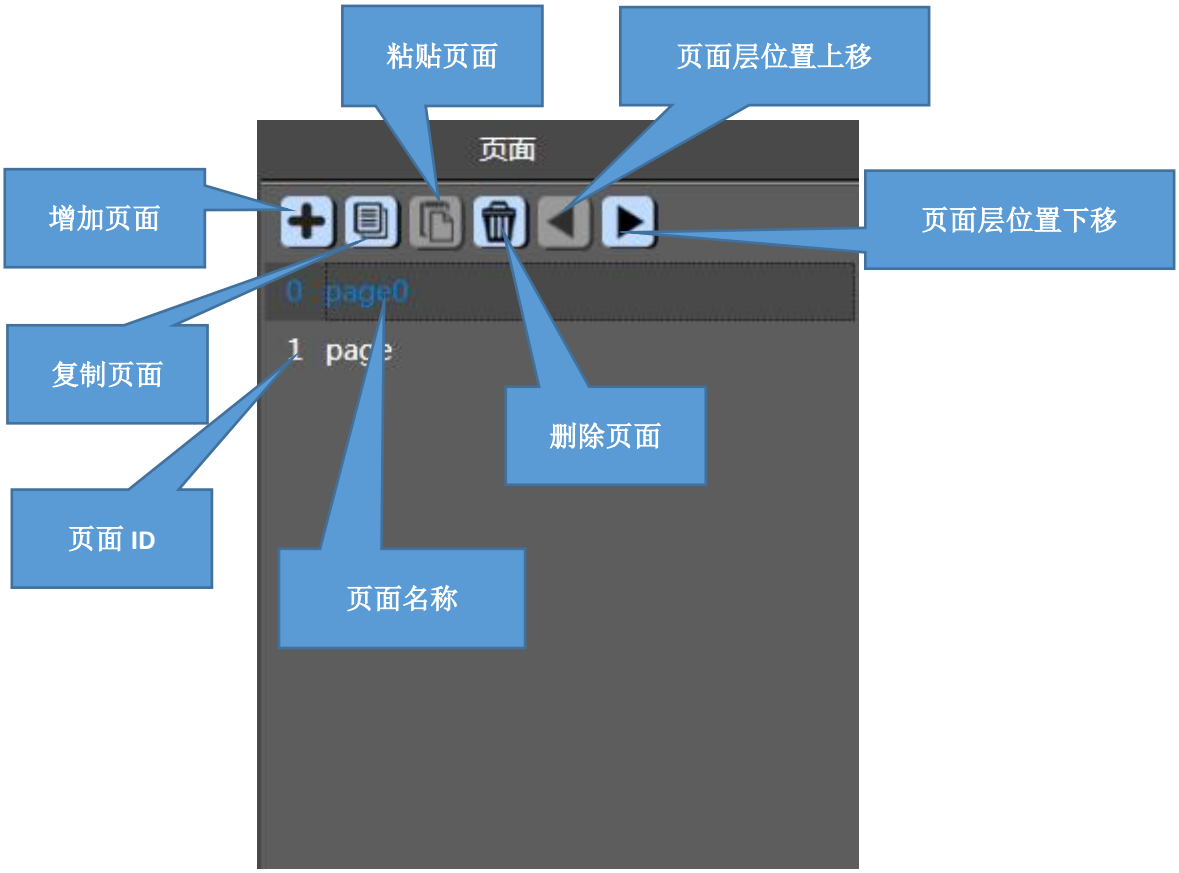
页面编辑区是页面显示主要工作区域，可以查看基本效果，位置，点击控件进行属性调整等。

输出区，主要功能是输出编译结果，提示编译错误。

事件编辑器，点击不同的控件，会出现不同的事件编辑区域。事件编辑区域中，可以使用类 C 语言编辑脚本，执行事件处理过程。

7. 页面区域

页面区域位于 VP 编辑器的右下角，执行新增页面，复制页面，粘贴页面，删除页面的操作。每个页面都有页 ID 和页名称组成，页名称只能是纯字母或字母和数字组成，**页 ID 不可修改，名称可以修改**。



功能	介绍
新建页面	点击按钮，新建一个页面，双击名称可重命名，命名只支持英文、数字和下划线组合。
复制页面	选中需要复制的页面，点击复制，然后点击后面的粘贴页面，完成页面的复制。
粘贴页面	选中需要复制的页面，点击复制，然后点击后面的粘贴页面，完成页面的复制。
删除页面	选中需要删除的页面，点击后删除页面。
页面上移	选中需要上下移动的页面，点击后改变页面层级顺序，但不会改变页面的 ID 号。
页面下移	选中需要上下移动的页面，点击后改变页面层级顺序，但不会改变页面的 ID 号。
页面 ID	不可更改，新建页面时自动生成。
页面名称	双击名称可重命名，命名只支持英文、数字和下划线组合。在脚本跨页访问控件时，需要用到完整的页面名称。

8. 脚本语法介绍

8.1. 概要

名称	功能	介绍
分隔符	语句结束符“;”	英文状态下的分号，每条语句以此为分隔符，同 C 语言。
标识符	标识变量名称	一个标识符以字母 A-Z 或 a-z 或下划线_开始，后跟零个或多个字母、下划线和数字（0-9）。标识符区分大小写。同 C 语言。
关键字	语言组成的关键字	目前支持： if,if else,for,while,do while, break ,continue,int , float,void。
数据类型	变量的数据类型定义	byte: 带符号单字节。 int：带符号 32 位整型。 float：带符号 32 位单精度浮点型。 string：字符串类型（通过变量控件设置和使用）。 其中数组可选择的类型有：byte、int、float、string。
运算符	变量常量之间的运算	目前支持：算术运算符，关系运算符，逻辑运算符 。
函数	带参数的功能处理集合	支持带变量的函数处理，方式同 C 语言，可自定义函数。
字符串	字符串常量	目前支持：字符串相加，用“+”运算符执行。
变量	可变数值的标识符	目前支持：脚本定义类型支持 int、float、byte，作用区域只在当前事件编辑区里，而变量控件可作用于全局。String 变量定义，需通过变量控件设置和使用。

8.2. 变量类型与定义

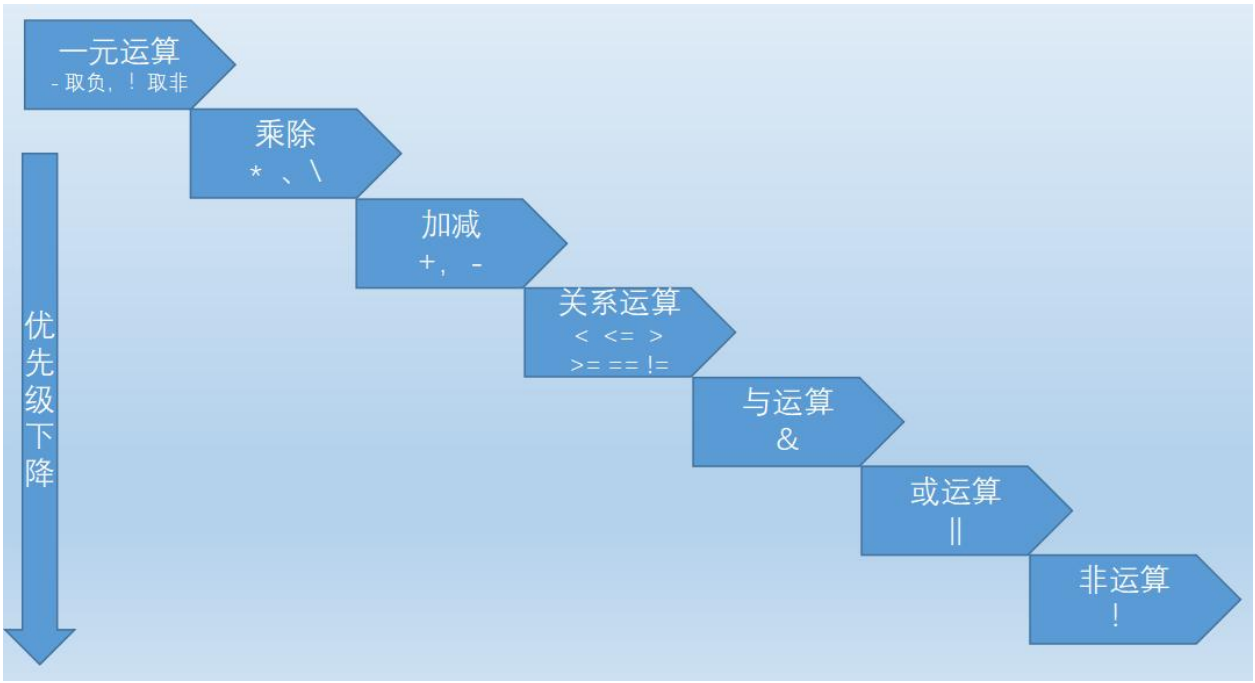
局部变量	声明一个整型变量： int a; 声明一个浮点型变量：float b; 定义一个 byte 数组：byte a[3]; 也可直接赋初值： 整型变量： int a = 100; 浮点型变量：float b = 0.01; byte 数组： byte a[3] = {0x01,0x02,0x03};

	以上定义只能作用于事件脚本编辑区，无法跨控件或跨页面使用。 **注意：脚本内变量定义的位置必须在事件脚本最前面，否则编译会报错。**

全局变量	全局变量，只能通过变量控件实现。
	拖入变量控件，首先在属性栏勾选 global 属性，勾选后才能全局使用；
	控件 type 属性选择数据类型，可选择选择 int、float、byte、string;
	变量初值为 val 属性;
	变量的值操作：
	若 type 属性选择的 **int**：var1.val = 123;//变量控件 var1 的值改为 123
	若 type 属性选择的 **float**：var1.valf = 1.23;//变量控件 var1 的值改为 1.23
	若 type 属性选择的 **string**：var1.txt = “good”;//变量控件 var1 的值改为“good”

8.3. 运算符

算术	算术运算符包括：+ - * / %，对应 加 减 乘 除 取余，同 C 语言。 精度：4 字节的整数或者浮点数。
关系	关系运算符包括：< 小于，<= 小于等于，>大于 ，>=大于等于， ==是否相等， !=是否不等。同 C 语言。
逻辑	逻辑运算符包括：&& 与运算， || 或运算，！非运算。同 C 语言。
位运算	位运算符包括：“<<”左移运算，“>>”右移运算，“&”位与，“|”位或，“^”位异或。同 C 语言。



运算符优先级

8.4. 条件语句

介绍	目前支持条件关键字： **if，if ... else...**
	一个 if 语句由一个布尔表达式后跟一个或多个语句组成。如果是一条语句，if，else 后不用大括号{}，如果为多条，用大括号{}，同 C 语言。
		举例：
		if(b0.txt==“确定”)
		{
		b0.txt=“取消”;
		}else if(b0.txt==“取消”)
		{
		b0.txt=“等待”;
		}else
		{
		b0.txt=“确定”;
		}
		b0 是按钮控件的名字，txt 是按钮控件的属性。这个例子完成按钮控件上文字的变换。

8.5. 循环语句

指令名	功能	举例

for	固定多次执行一个语句序列，简化管理循环变量的代码。同 C 语言。	while(a>10)
		{
		……
		}
		do
		{
		……
		}while(a>10)
		for (a = 0; a < 100; a++)
		{
		……
		}

while	当给定条件为真时，重复语句或语句组。它会在执行循环主体之前测试条件。同 C 语言。	
for	除了它是在循环主体结尾测试条件外，其他与 while 语句类似，同 C 语言。	
break	终止循环语句，程序流将继续执行紧接着循环的下一条语句。同 C 语言。	
continue	告诉一个循环体立刻停止本次循环迭代，重新开始下次循环迭代。同 C 语言。	

8.6. 脚本注释

单行注释： **//**text3.txt = “123”;

多行注释： **/***text3.txt = “123”;
text3.txt = “456”;***/**

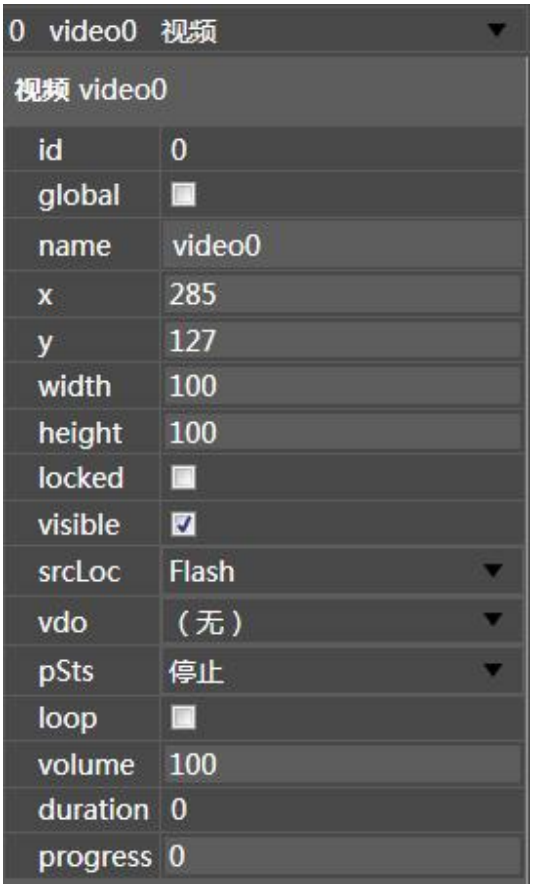
第三章 控件的介绍和用法

控件属性区，显示每个控件（包含不可见控件）的属性，通过属性配置，就可以生成丰富多彩的页面，属性配置比较细致和重要，请大家仔细阅读。

点击每一个属性，下方都有对属性的中文说明。

大部分可以改变的属性，都支持脚本和命令修改，用户在事件脚本和串口指令中，可以实时改变控件的属性，例如替换图片，替换音频，修改大小，移动位置等。例如通过定时器来控制控件的 x、y 坐标值，可以使图片等控件移动起来。

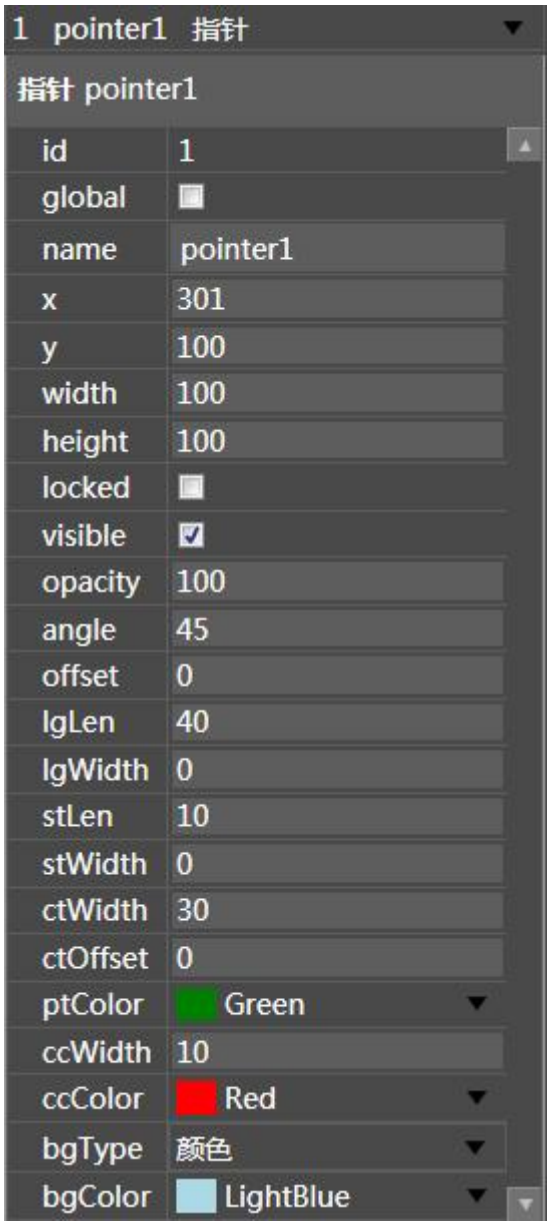
1. 视频控件

布局	属性	说明
	id	控件 ID 号，不可更改，不可脚本读写
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称 video+序号
	x/y width/height	控件的位置和宽高，脚本可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件的隐藏（未选中）与显示（选中）
	srcLoc	选择从 Sdcard/Flash 中读取视频，脚本不可读写
	path	srcLoc 属性选择 Sdcard 才有，填视频资源的路径，必须以.MP4 后缀结束（只支持 MP4 格式的视频，路径只支持英文）。脚本可读写
	vdo	srcLoc 属性选择 Flash 才有，用于指定视频源 脚本可读写
	pSts	视频状态，开始：1，暂停：2，停止：0
	loop	是否循环播放，循环：1，不循环：0
	volume	设置音量：0~100，100 为最大音量
	duration	视频总时长，只读属性，不可修改
	progress	调整视频播放进度，范围 0~100，例如：progress=50，则跳转到视频的 50%开始播放。可读写
使用详解：**http://www.sany-semi.com/list-107-1.html**		

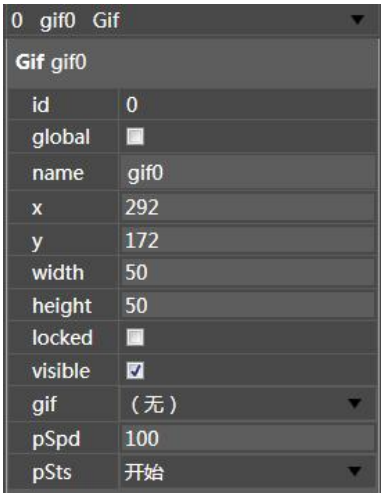
2. 音频控件

布局	属性	说明
	id	控件 ID 号，不可更改，不可脚本读写
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，audio+序号
	srcLoc	资源路径选择，不可脚本读写。
	path	srcLoc 属性选择 Sdcard 才有，填音频资源的路径，必须以”.wav/.mp3”后缀结束（支持 WAV 和 MP3 格式的音频，路径只支持英文）。
	ado	srcLoc 属性选择 Flash 才出现，直接选择已经在资源中加入的音频，下拉框选择。
	pSts	播放状态，停止：0，开始：1，暂停：2
	loop	设置 loop=1，循环，loop=0，不循环
	volume	设置范围 volume=0~100，100 为最大音量
	duration	音频总时长，只读属性，不可修改，用户可以用来做音频播放进度控制。
	progress	调整音频播放进度，范围 0~100，例如：progress=50，则跳转到音频的 50%开始播放。
使用详解：http://www.sany-semi.com/list-107-1.html		

3. 指针控件

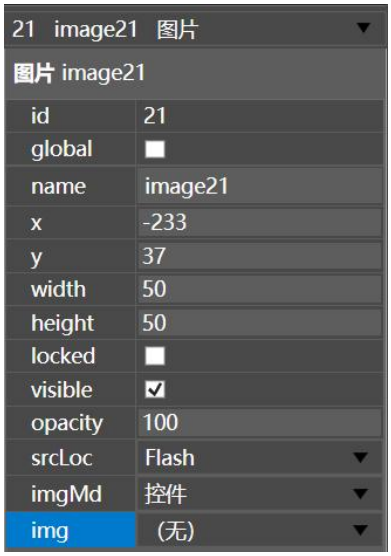
布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称， rectangle +序号
	x/y Width/height	控件的位置和宽高，脚本可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：visible=1，可见，visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	angle	指针起始角度，左边水平方向为 0 度。
	offset	指针角度的偏移量，单位度。
	lgLen	指针长端的长度。
	lgWidth	指针长端的端点宽度。默认为 0，是完全的尖角，不为 0，尖端会变成直线。
	stLen	指针短端的长度。
	stWidth	指针短端的端点宽度。默认为 0，是完全的尖角，不为 0，尖端会变成直线。
	ctWidth	指针中段宽度。
	ctOffset	指针向长端的偏移量。
	ptColor	指针的颜色。
	ccWidth	指针中心圆心的直径，为 0，则中心轴消失。形成悬浮指针。
	ccColor	指针中心轴的颜色。
	bgType	背景类型，颜色或者图片
	bgColor	如果设置为颜色，设定具体颜色
	bgImg	背景类型是图片时，选择背景图片。根据图片资源名称和顺序号选择。
	bgImgMd	背景类型是图片时，默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角

4. GIF 控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，gif+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：gif1.visible=1，可见；gif1.visible=0，不可见
	gif	选择资源区的 gif 文件
	pSpd	播放速度，调整范围：10%~1000%，脚本举例（以下出现属性值=值，则省去“脚本举例”字样）：gif1.pSpd=20；
	pSts	播放状态：gif1.pSts=0; 停止，gif1.pSts=1;开始，gif1.pSts=2；暂停。

注意：串口屏所有系列 GIF 控件都不支持画面背景有透明度的 gif；

5. 图片控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，image+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：image0.visible=1，可见，image0.visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	srcLoc	图片资源的位置，可内部 Flash 和外部 Sdcard 两种方式
	ImgMd	图片的对齐方式，可相对于控件对齐，和相对于页面对齐
	Img	srcLoc 选择 Flash 时直接选择导入到图片资源栏的图片 srcLoc 选择 Sdcard 时，则为 SD 卡中图片的路径

注意：

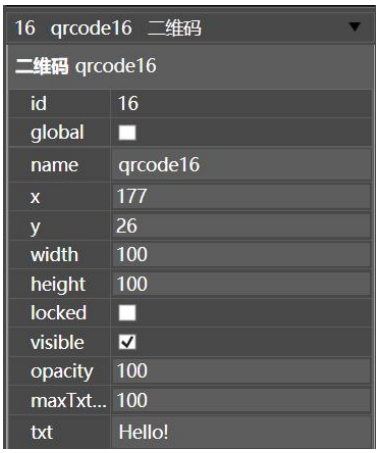
- 串口屏 G 系列（2.4、2.8、3.2、3.5 等）不支持带有透明的 png 格式图片，最好请使用 jpg 格式的图片，或者是 png 格式不带透明的图片：

例：图 1. 

图 2. 

- 这两种都是 png 格式的图片，但是图 1 不支持，图 2 支持，因为图 1 是带有透明的 png，图 2 格式虽为 PNG，但不带透明；
- 若 G 系列串口屏，VP 上位机工程导入带有透明的 PNG（图一），下载到串口屏，运行时带有透明的部分会显示为黑色，并可能会在后续的使用中出现不可预计的问题（比如花屏）；
- 串口屏 S 系列（4.3、7.0）及以上尺寸支持 jpg 和上面图一图二格式的图片；
- 图片格式的区别，也存在于其他可以使用图片的控件上，如按钮控件，指针控件，文本控件等；
- 对于大尺寸的屏幕，如 7.0、10.0，导入的图片最大不能超过 1280X720，超过后会导致图片无法显示；

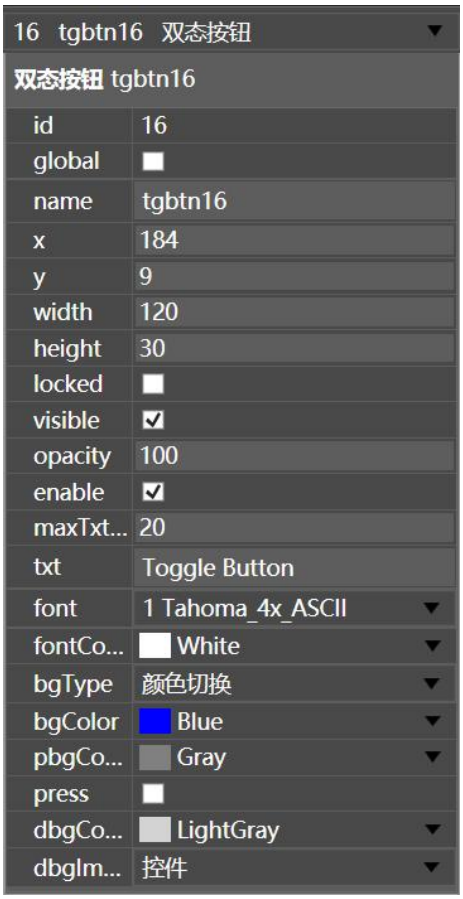
6. 二维码控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，qrcode+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可在显示区外，高度不够会显示不全。
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：qrcode1.visible=1，可见，qrcode1.visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	maxTxtLen	生成二维码的数据最大长度。默认 maxTxtLen=100 ， byte。
	txt	生成二维码的具体数据，可见 ASCII 码

7. 触摸热区控件

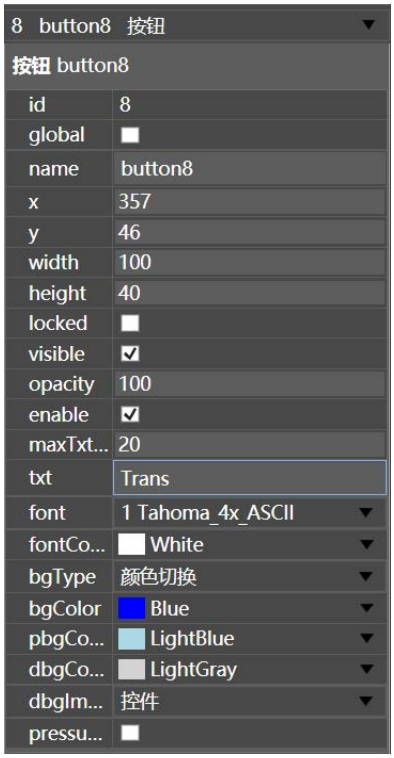
布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，hotZone+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性

8. 双态按钮控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，tgbtn+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：tgbtn3.visible=1，可见，tgbtn3.visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	enable	功能使能。控制按钮是否可以使用，按钮类支持不可用状态，方便客户有条件的启动按钮。enable=0;不可用 enable=1;可用
	maxTxtLen	按钮上文本的最大长度。
	txt	按钮上显示的文本。
	font	按钮文本使用的字库。
	fontColor	按钮文本使用的颜色。
	bgType	背景类型，包括按下时 切换背景颜色，切换背景图。
背景为**颜色**切换时，设置说明	bgColor	如果背景为颜色，此处设置默认的背景颜色。
	pbgColor	如果背景为颜色，此处设置按下时背景颜色
	dbgColor	禁用时所使用的背景颜色。
背景为**图片**切换时，设置说明	bgImg	选择背景图片。根据图片资源名称和顺序号选择。
	bgImgMd	默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角

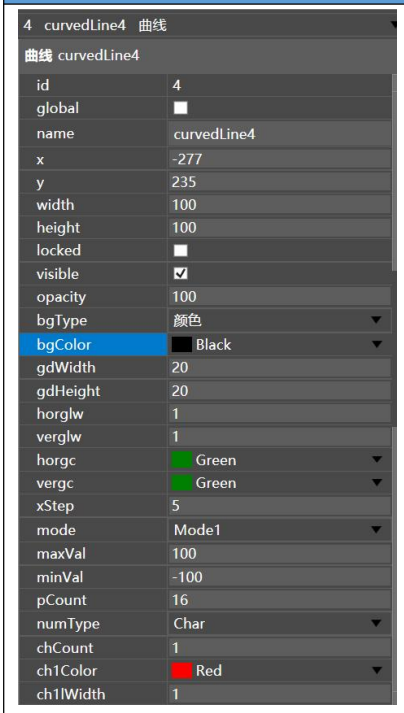
	pbgImg	选择按下时的背景图片。 根据图片资源名称和顺序号选择。
	pbgImgMd	按下时背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	dbgImg	选择禁用时的背景图片。 根据图片资源名称和顺序号选择
	dbgImgMd	禁用时的背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	press	双态按钮是否处于按下状态：1，按下；0，弹起； 此属性可脚本读写；

9. 按钮控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，button+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本:button7_3.visible=1，可见，button7_3.visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	enable	功能使能。按钮类支持不可用状态，方便客户有条件的启动按钮。enable=0;不可用　enable=1;可用
	maxTxtLen	按钮上文本的最大长度。
	txt	按钮上显示的文本。
	font	按钮文本使用的字库。
	fontColor	按钮文本使用的颜色。
	bgType	背景类型，分为四种： 颜色，选中此项，则指定只需指定一个颜色，按下时不会切换。 颜色切换，选中此项，需要指定，默认颜色，和按下去切换的颜色。 图片，选中此项，则指定只需指定一个背景图片，按下时不会切换。 图片切换，选中此项，需要指定，默认背景图片，和按下去切换的背景图片。
背景为**颜色**，设置说明	bgColor	如果背景为颜色，此处设置默认的背景颜色。
	dbgColor	禁用时所使用的背景颜色。

背景为**图片**时，设置说明	bgImg	选择背景图片。根据图片资源名称和顺序号选择。
	bgImgMd	默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	dbgImg	选择禁用时的背景图片。根据图片资源名称和顺序号选择
	dbgImgMd	禁用时的背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
背景为**颜色**切换时，设置说明	bgColor	如果背景为颜色，此处设置默认的背景颜色。
	pbgColor	如果背景为颜色，此处设置按下时背景颜色
	dbgColor	禁用时所使用的背景颜色。
背景为**图片**切换时，设置说明	bgImg	选择背景图片。根据图片资源名称和顺序号选择。
	bgImgMd	默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	pbgImg	选择按下时的背景图片。根据图片资源名称和顺序号选择。
	pbgImgMd	按下时背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	dbgImg	选择禁用时的背景图片。根据图片资源名称和顺序号选择
	dbgImgMd	禁用时的背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角

10. 曲线控件

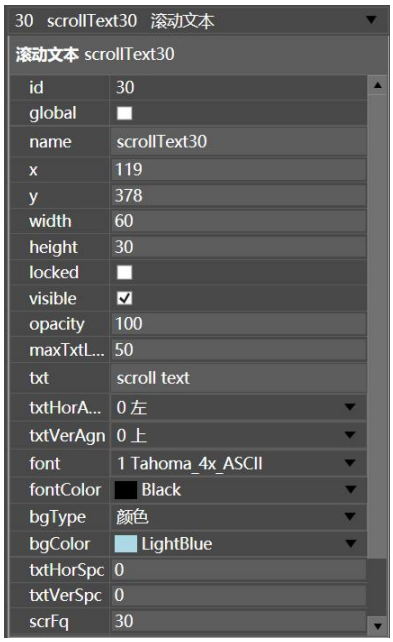
布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，curvedLine+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：curvedLine.visible=1，可见，curvedLine.visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	bgType	背景类型，可选择颜色、图片。
	bgColor	背景类型是颜色时，选择背景颜色。
	bgImg	背景类型是图片时，选择背景图片。 根据图片资源名称和顺序号选择。
	bgImgMd	背景类型是图片时，默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	gdWidth	网格宽度，如果不需要网格线，可设置为 0
	gdHeight	网格高度，如果不需要网格线，可设置为 0
	horglw	水平网格线条宽度，如果不需要网格线，可设置为 0
	werglw	垂直网格线条宽度，如果不需要网格线，可设置为 0
	horgc	设置水平网格线条的颜色
	vergc	设置垂直网格线条的颜色。
	xStep	曲线打点时水平步进的距离，单位，像素。
	mode	曲线绘制的模式 Mode1：推进式绘制 Mode2：覆盖式绘制 Mode3：折线图模式(新增)
	maxVal	数据的最大值。
	minVal	数据的最小值。
	pCount	每个通道显示的点的数量，但是超过 x 轴总像素除以步进距离的值后，无效。
	numType	1.每个数据点的值的类型：字符型，2 字节整型，4 字节整型，四字节浮点型。 2.为了方便客户采样后不用转数据，直接发送到串口屏模块，定义了四种数据类型。 3.客户选用数据类型后，定义实际需要的最大值和最小值（支持正负数）后，可以直接发送收集到的二进制数据到三易串口屏模块，模块根据具体数值算出数据应该显示的点位，自动显示。
	chCount	数据通道数量，也就是要显示几条曲线，目前支持最多 3 条。

	ch1Color	通道一的数据打点颜色。
	ch1Width	通道一的数据打点的线宽，单位：像素点。
	ch2Color	通道二的数据打点颜色。
	ch2Width	通道二的数据打点的线宽，单位：像素点。
	ch3Color	通道三的数据打点颜色。
	ch3Width	通道三的数据打点的线宽，单位：像素点。

void clear(int chId)
功能：清除指定通道的数据
参数：chId：通道号，范围 1~3
返回值：无

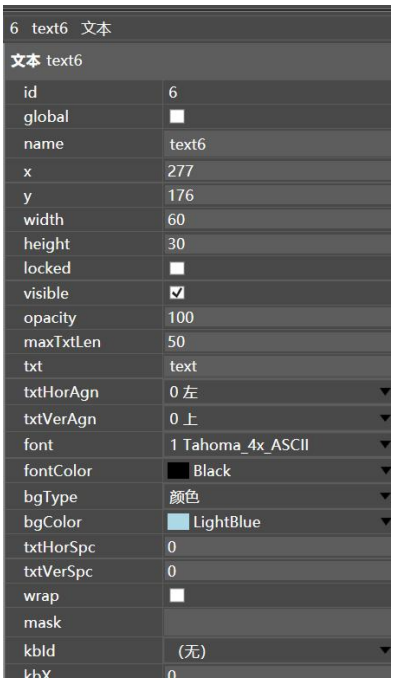
void setPoint(int chId, int pid, float x, float y)
功能：设置折线图模式(Mode3)的点数据，该函数仅在 Mode3 模式下有效
参数：chId：通道号，范围 1~3
pid：点序号，范围 0~ pCount
x：横坐标值，显示的位置与 minValX，maxValX 相关
y：纵坐标值，显示的位置与 minVal，maxVal 相关
返回值：无

11. 滚动文本控件

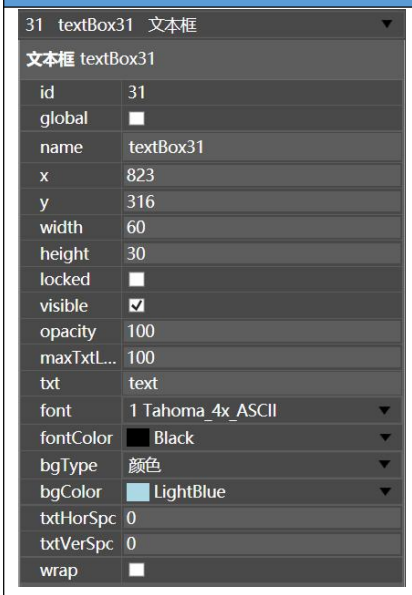
布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，scrollText+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：visible=1，可见，visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。 值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	maxTxtLen	文本内容的最大长度，单位字节。
	txt	需要显示的文本
	txthorAgn	水平对齐方式，左，中，右。
	txtVerAgn	垂直对齐方式，上，中，下。
	font	显示文本使用的字体，不设置不能显示字体
	fontColor	显示的文字颜色。
	bgType	背景类型，颜色或者图片
	bgColor	如果设置为颜色，设定具体颜色
	bgImg	背景类型是图片时，选择背景图片。根据图片资源名称和顺序号选择。
	bgImgMd	背景类型是图片时，默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。

		页面对齐：图片左上角对齐页面的左上角
	txtHorSpc	字符水平间距
	txtVerSpc	字符的行间距
	scrFq	两次滚动动作的间隔，单位毫秒。
	scrDst	单词滚动，字符移动的距离，单位像素。
	scrMd	滚动方式，左到右，右到左，上到下，下到上
	scrSts	滚动字符状态，0 停止，1 开始，2 暂停

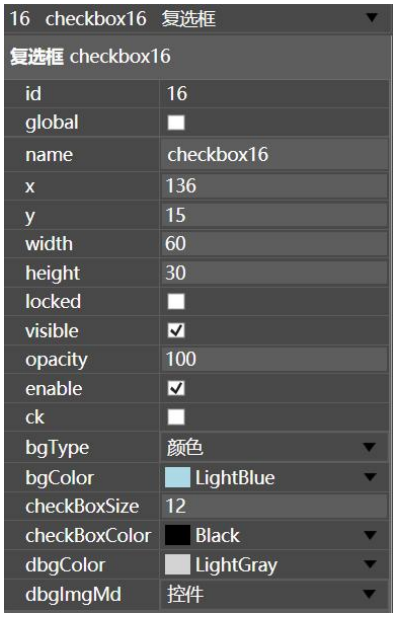
12. 文本控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，text+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，宽度不够字会显示不全。可读写
	height	控件高度，高度不够字会显示不全。可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	是否可见，脚本：visible=1，可见，visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。 值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	maxTxtLen	文本内容的最大长度，单位字节。
	txt	需要显示的文本
	txthorAgn	水平对齐方式，左，中，右。
	txtVerAgn	垂直对齐方式，上，中，下。
	font	显示文本使用的字体，不设置不能显示字体。
	fontColor	显示的文字颜色。
	bgType	背景类型，颜色或者图片
	bgColor	如果设置为颜色，设定具体颜色
	bgImg	背景类型是图片时，选择背景图片。根据图片资源名称和顺序号选择。
	bgImgMd	背景类型是图片时，默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	txtHorSpc	字符水平间距
	txtVerSpc	字符的行间距
	wrap	是否自动换行
	mask	填入一个字符，则 txt 文本所有的显示都会显示这一个字符，例如填入‘*’，则用 ‘*’ 代替字符显示，用作模拟密码输入显示。删除掉 mask 里面填入的字符，则 txt 显示应该显示的明文。
	kbId	绑定生成的键盘资源，点击文本控件时直接弹出键盘，键盘输入回车后直接隐藏。
	kbX	键盘弹出时的 X 坐标点。
	kbY	键盘弹出时的 Y 坐标点。
	KbInitVal	勾选后，每次弹出键盘自动清空上次的输入

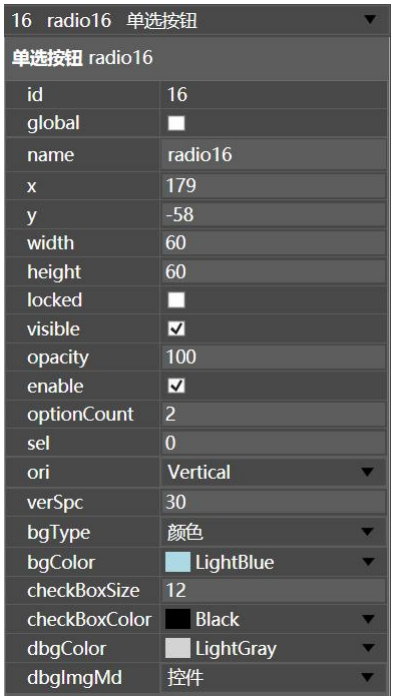
13. 文本框控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，textBox+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，宽度不够字会显示不全。可读写
	height	控件高度，高度不够字会显示不全。可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	是否可见，脚本：visible=1，可见，visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。 值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	maxTxtLen	文本内容的最大长度，单位字节。
	txt	需要显示的文本，可换行显示，多行时，支持滑动查看未显示部分
	txthorAgn	水平对齐方式，左，中，右。
	txtVerAgn	垂直对齐方式，上，中，下。
	font	显示文本使用的字体，不设置不能显示字体。
	fontColor	显示的文字颜色。
	bgType	背景类型，颜色或者图片
	bgColor	如果设置为颜色，设定具体颜色
	bgImg	背景类型是图片时，选择背景图片。根据图片资源名称和顺序号选择。
	bgImgMd	背景类型是图片时，默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	txtHorSpc	字符水平间距
	txtVerSpc	字符的行间距
	wrap	是否自动换行
	mask	填入一个字符，则 txt 文本所有的显示都会显示这一个字符，例如填入‘*’，则用 ‘*’ 代替字符显示，用作模拟密码输入显示。删除掉 mask 里面填入的字符，则 txt 显示应该显示的明文。
	kbId	绑定生成的键盘资源，点击文本控件时直接弹出键盘，键盘输入回车后直接隐藏。
	kbX	键盘弹出时的 X 坐标点。
	kbY	键盘弹出时的 Y 坐标点。
	KbInitVal	勾选后，每次弹出键盘自动清空上次的输入

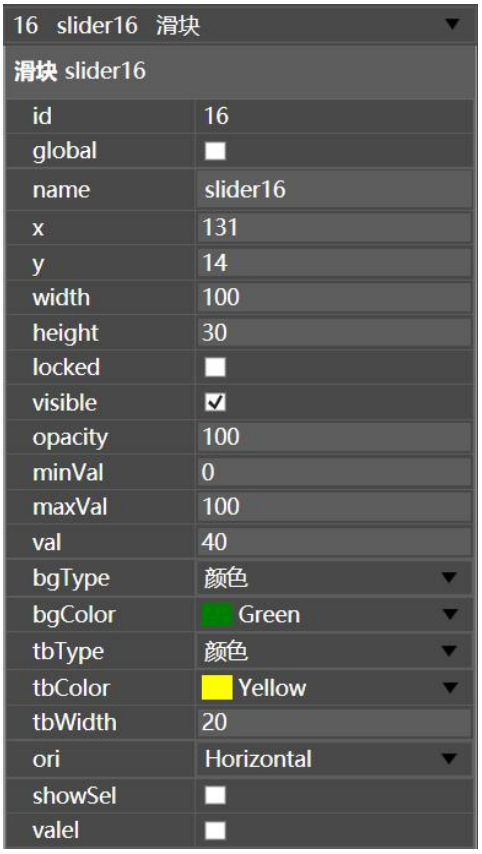
14. 复选框控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称 checkbox+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：visible=1，可见，visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	enable	控件使能，复选框控件是否起作用，有些情况下，不让使用时，客户可以设置这个控件失效。勾选为可用
	ck	此复选框是否选中
	bgType	背景类型，颜色或者图片
	bgColor	如果设置为颜色，设定具体颜色
	bgImg	背景类型是图片时，选择背景图片。根据图片资源名称和顺序号选择。
	bgImgMd	背景是图片时，默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	checkBoxSize	选择边框的大小，只是方形选择框区域
	checkBoxColor	选择边框的颜色，只是方形选择框区域。控件大小不等于选择框大小。
	dbgColor	如果不可用状态，而且是背景选择为颜色，则在这里选择不可用时的背景颜色。
	dbgImg	如果不可用状态，而且是背景选择为图片，则在这里选择不可用时的背景图片。
	dbgImgMd	背景类型是图片时，不可用状态的背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角

15. 单选按钮控件

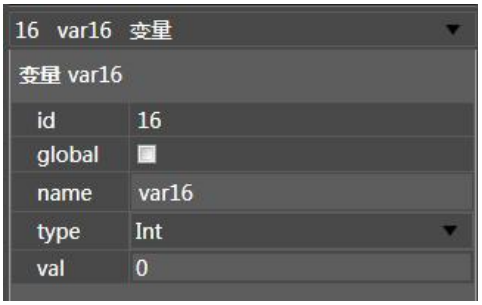
布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，redio+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：visible=1，可见，visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	enable	使能，复选框控件是否起作用，有些情况下，不满足条件时，客户可以设置这个控件不起作用。勾选为可用
	optionCount	设置一组单选，具体多少个选项
	sel	当前选择的是哪个选项，sel = 8，第 8 项选中。
	ori	选项的排列方向：Vertical ->垂直排列，Horizont-> 水平排列，
	HorSpc	选择 Horizont 时：水平排列的选项间的间距
	verSpc	选择 Vertical 时：垂直排列的选项间的间距
	checkBoxSize	设置选择框大小
	checkboxColor	设置选择框颜色
	bgType	背景类型，颜色或者图片
	bgColor	如果设置为颜色，设定具体颜色
	bgImg	背景类型是图片时，选择背景图片。根据图片资源名称和顺序号选择。
	bgImgMd	背景类型是图片时，默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	dbgColor	如果不可用状态，而且是背景选择为颜色，则在这里选择不可用时的背景颜色。
	dbgImg	如果不可用状态，而且是背景选择为图片，则在这里选择不可用时的背景图片。
	dbgImgMd	背景类型是图片时，不可用状态的背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角

16. 滑块控件

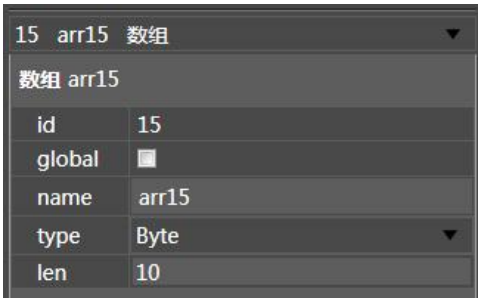
布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称 slider+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：visible=1，可见，visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	minVal	设置滑块最小值
	maxVal	设置滑块最大值
	val	滑块当前值，改变当前值，可以改变滑块的位置，val=50，滑块移动到，0~100 值范围的中间位置。
	bgType	背景类型，颜色或者图片
	bgColor	如果设置为颜色，设定具体颜色
	bgImg	背景类型是图片时，选择背景图片。根据图片资源名称和顺序号选择。
	bgImgMd	背景是图片时，默认背景的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	tbType	游标 背景类型，颜色或者图片
	tbColor	如果设置为颜色，设定具体颜色
	tbImg	背景类型是图片时，选择背景图片。根据图片资源名称和顺序号选择。
	tbWidth	游标宽度，如果游标背景是图片，图片宽度要>=游标宽度。
	ori	滑块摆放方向：横放 -> horizont,竖放 -> vertical。
	showSel	是否显示选择范围。 选中，显示颜色或者图片，必须和 bgtype 选项一致。 滑块画过的区域会显示 selBgImg 属性设置的颜色或者图片； 不选，滑块在设置的颜色或者图片上滑动，没有改变颜色或者图片的效果。
	selBgColor	划过区域，背景类型是颜色时，选择背景颜色。
	selBgImg	划过区域，背景类型是图片时，选择背景图片。根据图片资源名称和顺序号选择。
	selBgImgMd	划过区域，背景类型是图片时，默认背景图片的对齐模式。

		控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	valel	仅在触摸时，触发值改变事件。 不选中，其他控件脚本改变滑块的 val 属性值时，可以触发滑块的值改变事件。 选中，只有通过触摸屏滑动滑块，才能触发值改变的事件，运行事件的脚本。 选中后，可以控制只有触摸滑块才能使用滑块功能。 默认不选中。

17. 变量控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，var+序号
	type	数据类型，int ； float； string；
	val	变量当前值。变量的值可以通过脚本、串口命令等改变，作为变量的暂存空间。 **脚本赋值操作：** **type 属性选择 Int 时，var16.val = 123;** **type 属性选择 Float 时，var16.valf = 12.3;** **type 属性选择 String 时，var16.txt= “123”；**

18. 数组控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	不勾选，作用于当前页面；勾选后，作用于全局
	name	控件名称，可改，默认名称，arr+序号
	type	数据类型，byte；int ； float； string；
	len	数组长度

脚本中数组元素的赋值操作

type 属性选择 byte 时：

赋值：arr15.valbs[0] = 0x64; 或者 arr15.valbs[0] = 100;

读取：num1.val = arr15.valbs[0]; //读出用整数控件 num1 显示，显示值为 100

type 属性选择 int 时：

赋值：arr15.vals[0] = 0x64; 或者 arr15.vals[0] = 100;

读取：num1.val = arr15.vals[0]; //读出用整数控件 num1 显示，显示值为 100

type 属性选择 float 时：

赋值：arr15.valfs[0] = 100.0;

读取：numf1.valf = arr15.valfs[0]; //读出用浮点数控件 numf1 显示，显示值为 100.0

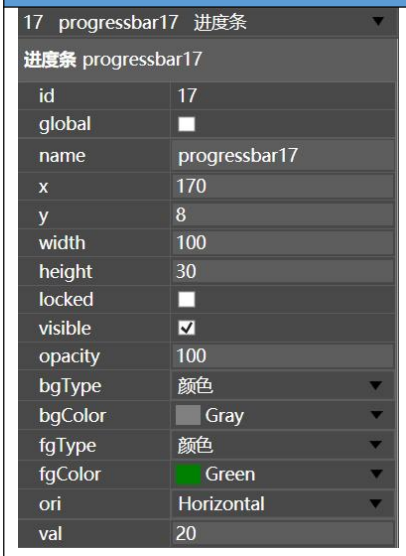
type 属性选择 string 时，需要用控件的方法函数：

```
赋值：arr15.set(0,"txt123");  //赋值字符串"txt123"
读取：text1.txt = arr15.get(0);  //读出用文本控件 text1 显示，显示值为"txt123"
清除：arr15.clean(0,10);  //清除从下标 0 开始，向后的 10 个元素
```

19. 定时器控件

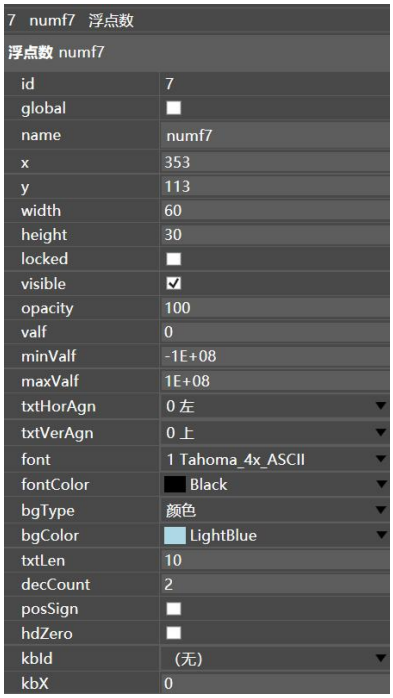
布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称 timer+序号
	en	使能，不选中，失效；选中，有效。
	interval	定时器信号发生间隔。

20. 进度条控件

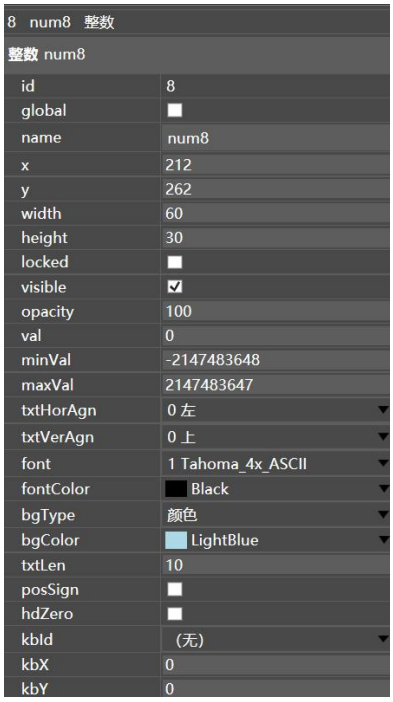
布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称 progressbar+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本： Progressbar17.visible=1，可见 Progressbar17.visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。 值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	bgType	背景类型，颜色或者图片
	bgColor	如果设置为颜色，设定具体颜色
	bgImg	背景类型是图片时，选择背景图片。根据图片资源名称和顺序号选择。
	bgImgMd	背景类型是图片时，默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	fgType	进度条前景类型，颜色或者图片
	fgColor	如果设置为颜色，设定具体颜色
	fgImg	进度条前景类型是图片时，选择前景图片。根据图片资源名称和顺序号选择。

	ori	滑块摆放方向：横放 -> horizont,竖放 -> vertical。
	val	进度条当前值。

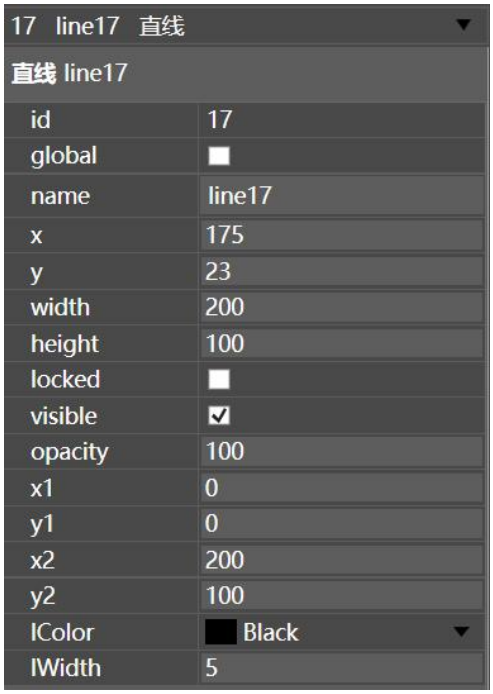
21. 浮点数控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，numf+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	是否可见，脚本：visible=1，可见，visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。 值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	valf	浮点数当前值。 **浮点数赋值时，注意：** numf10.valf = 10/4;运算值只会显示整数位，即 2.0 numf10.valf = 10/4.0;运算值才会显示成 2.5 所以**整数除以整数**时，一般把除数乘 1.0，即： numf10.valf = 10/(4*1.0); 或者添加强制转换：numf10.valf = 10/(float)4;
	minValf	此控件的浮点数最小值。
	maxValf	此控件的浮点数最大值。
	txtHorAgn	控件显示的水平对齐方式。左，中，右对齐。
	txtVerAng	控件显示的竖着对齐方式。上，中，下对齐。
	font	显示文本使用的字体，如果不设置，不能显示字体出来。
	fontColor	显示的文字颜色。
	bgType	背景类型，颜色或者图片
	bgColor	如果设置为颜色，设定具体颜色
	bgImg	背景类型是图片时，选择背景图片。
	bgImgMd	背景类型是图片时，默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	txtLen	最大显示字符的长度。
	decCount	显示的小数位数。
	posSign	是否显示符号。
	hdZero	是否填充前导“0”。
	kbId	绑定生成的键盘资源，点击文本控件时直接弹出键盘， 键盘输入回车后直接隐藏。
	kbX	键盘弹出时的 X 坐标点。
	kbY	键盘弹出时的 Y 坐标点。
	KbInitVal	勾选后，每次弹出键盘自动清空上次的输入

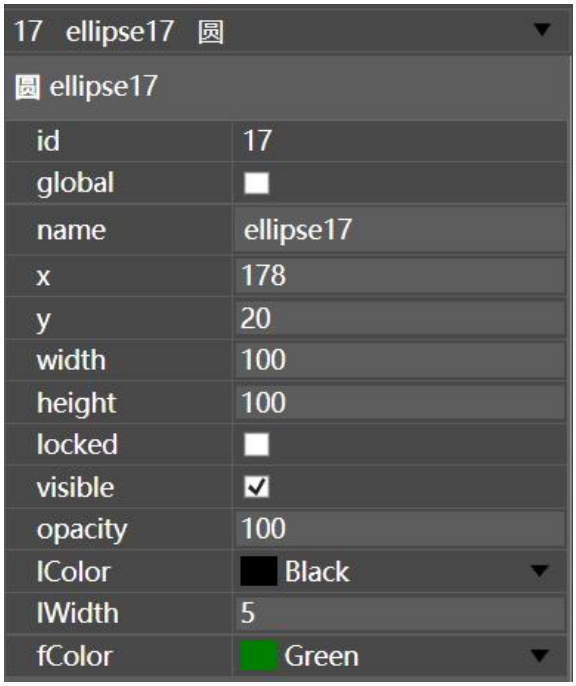
22. 整数控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页；不可脚本读写
	name	控件名称，可改，默认名称，num+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，宽度不够时字会显示不全。可读写
	height	控件高度，宽度不够时字会显示不全。可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本： num8.visible=1，可见，num8.visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	val	整数当前值。整数控件会显示当前值的 ascii 码表示，例如整数：1231323（四个字节），控件会显示“1231323”
	minVal	此控件的浮点数最小值。
	maxVal	此控件的浮点数最大值。
	txtHorAgn	控件显示的水平对齐方式。左，中，右对齐。
	txtVerAng	控件显示的竖着对齐方式。上，中，下对齐。
	font	显示文本使用的字体，如果不设置，不能显示字体出来。
	fontColor	显示的文字颜色。
	bgType	背景类型，颜色或者图片
	bgColor	如果设置为颜色，设定具体颜色
	bgImg	背景类型是图片时，选择背景图片。根据图片资源名称和顺序号选择。
	bgImgMd	背景类型是图片时，默认背景图片的对齐模式。 控件对齐：图片左上角对齐控件左上角。 页面对齐：图片左上角对齐页面的左上角
	txtLen	最大显示字符的长度。
	posSign	是否显示符号。
	hdZero	是否填充前导“0”。
	kbId	绑定生成的键盘资源，点击文本控件时直接弹出键盘，键盘输入回车后直接隐藏。
	kbX	键盘弹出时的 X 坐标点。
	kbY	键盘弹出时的 Y 坐标点。
	KbInitVal	勾选后，每次弹出键盘自动清空上次的输入

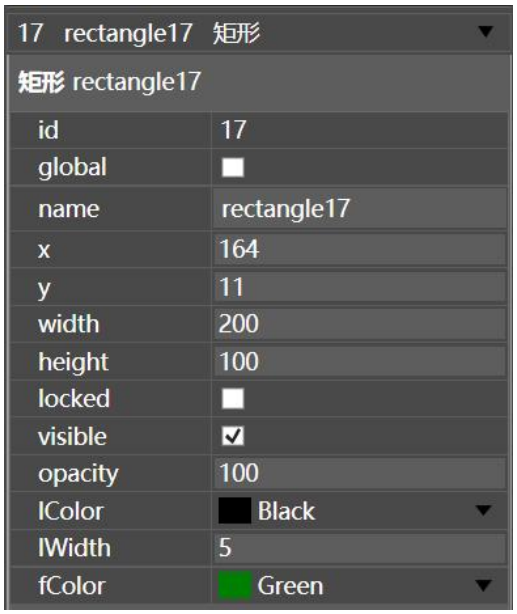
23. 直线控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认名称，line+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：visible=1，可见，visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	x1	直线起始点在控件内的相对坐 x 标值。控件的左上角作为相对坐标的零点。
	y1	直线起始点在控件内的相对坐 y 标值。
	x2	直线结束点在控件内的相对坐 x 标值。
	y2	直线结束点在控件内的相对坐 y 标值。
	IColor	直线线条的颜色。
	IWidth	直线线条的宽度。

24. 圆形控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认：ellipse+序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：visible=1，可见，visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	lColor	圆形边框线条的颜色。
	lWidth	圆形边框线条的宽度。
	fColor	圆形的填充颜色

25. 矩形控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认：rectangle +序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：visible=1，可见，visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	IColor	矩形边框线条的颜色。
	IWidth	矩形边框线条的宽度。
	fColor	矩形的填充颜色

26. 协议解析器控件

布局	属性	说明
	id	控件 ID 号，不可更改
	name	控件名称，可改，默认：protocol+序号
	rxMaxLen	用于接收数据的缓存的长度。
	txMaxLen	用于发送数据的缓存的长度。

注：一个工程只能用一个此控件，作用于全局，使用详解请参考后续第八章，典例应用。

27. RTC

非可见控件形式，需使用系统变量获取

rtc_year　　获取时间年
rtc_month　获取时间月
rtc_day　　获取时间日

rtc_hour　　获取时间时
rtc_minute　获取时间分
rtc_second　获取时间秒
rtc_week　　获取时间周

例：　num1.val = rtc_minute;　　//读取 minute 值赋给整数控件 num1 显示
　　rtc_month = 6;　　//设置月份为 6

注：以上系统变量除了 rtc_week 只能读以外，其他变量都可进行读写操作。

使用详解请参考后续第八章，典例应用。

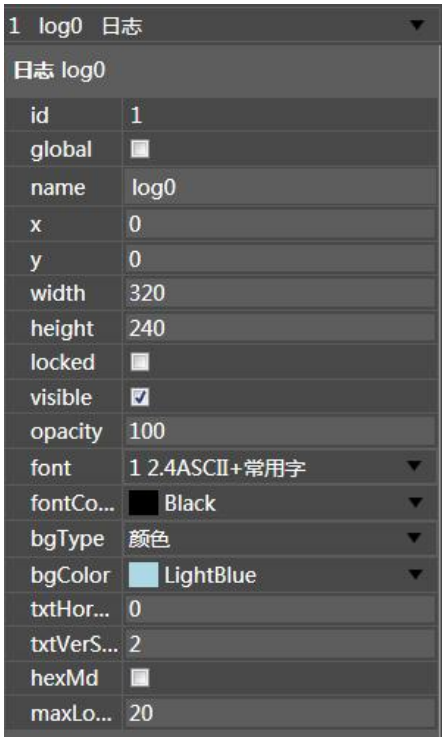
28. 环形进度条控件

布局	属性	说明
18 cprogressbar18 环形进度条 环形进度条 cprogressbar18 id 18 global ☐ name cprogressbar18 x 131 y 24 width 100 height 100 locked ☐ visible ☑ opacity 100 angle 135 stAngle 45 bgImg (无) bgImgMd 控件 fgImg (无) fgImgMd 控件	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认：cprogressbar +序号
	x	控件起始点 x 坐标
	y	控件起始点 y 坐标
	width	控件宽度，可读写
	height	控件高度，可读写
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件是否可见，脚本：visible=1，可见，visible=0，不可见
	opacity	不透明度，opacity=0，完全透明，opacity=100，完全不透明。值介于 0~100 之间。默认 100，非必要不建议修改此项。 **注：仅 S 系列支持调整透明度**
	angle	偏移角度，基于起始角度去增加
	stAngle	起始角度，范围：0-360
	bgImg	背景图片，进度未填充状态的图片
	bgImgMd	背景图片的对齐方式
	fgImg	前景图片，进度填充状态的图片
	fgImgMd	前景图片的对齐方式

29. 键盘控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认：keyboard+序号
	X/Y	键盘的位置
	Width/ Height	键盘的宽高
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	键盘的可见与隐藏 脚本：keyboard2.visible=1，可见，keyboard2.visible=0，不可见
	keyboard	选择要使用的键盘
	maxTxt	键盘字符的最大输入数量
	Mask	选择某一符号代替显示实际的输入内容，和文本控件的 mask 属性一样，此属性可用来做密码输入（填字符‘*’）
使用详解：http://www.sany-semi.com/list-107-1.html		

30. 日志控件

布局	属性	说明
	id	控件 ID 号，不可更改
	global	勾选：作用于全局，不勾选：作用于所在页； 不可脚本读写
	name	控件名称，可改，默认：log+序号
	X/Y	控件的显示坐标
	width/height	控件的宽高
	locked	勾选后锁定控件位置，但不影响脚本操作控件属性
	visible	控件的可见性 脚本：log0.visible=1，可见，log0.visible=0，不可见
	opacity	控件的透明度，默认 100，不透明，值介于 0~100 之间
	font	显示文本使用的字体，如果不设置，不能显示字体出来。
	fontColor	字体颜色
	bgType	控件背景类型。可选择纯色和图片。
	txtHorSpc	字符的水平间距

	txtVerSpc	字符的垂直间距（行间距）
	hexMd	十六进制显示模式
	MaxLogCount	存储的最大行数。设置多少行去显示接收的数据，当数据填满设置行数后，会自动清除超出的行数数据。

用法简介

1.打印显示字符串：

log0.addString("txt123"); //直接打印显示字符串"txt123"

2.打印显示字节数据，需先勾选控件的 hexMd 属性：

比如定义有数组：byte a[4]={0x01,0x02,0x03,0x04};

log0.addBytes(a,0,4); //打印数组 a，参数为数组名称 a，起始位置，长度

3.**log0.addBytes(protocol0.rxBuf,0,protocol0.rxLen);**

//用日志控件打印协议解析器 protocol0 所接收到的数据（十六进制显示，要勾选控件的 hexMd 属性）

4.**log0.addString(bytesToAscii(protocol0.rxBuf,0,protocol0.rxLen));**

//ASCII 码字符串显示（转换含有中文的数据时，使用 stringDecode( )函数）

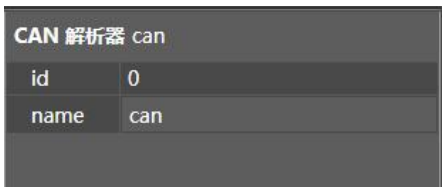
5.**log0.clear();**

//清空整个 log 控件的显示内容

addBytes()、addString()和 clear()方法函数是日志控件 log 的专用函数，输入"log0+."后会弹出对应方法。

注意：addBytes()和 addString()是两种显示模式，打印 Bytes 或者 String 数据，两种方法脚本中不可同时使用。

31. CAN 解析器控件

布局	属性	说明
	id	控件 ID 号，不可更改
	name	控件名称，可改，默认：can

控件的方法函数

void setFilter(int id,int mode,int arg1,int arg2);

此方法函数为 CAN 解析器控件专属，用于设置 CAN 的过滤器。

设置 CAN 过滤的方法有两种：一种是在项目设置中设置掩码和滤码（具体计算可参考后续第八章，实验 5）

一种就是利用此方法函数，在开机时去设置过滤（可参考后续第八章，实验 5）

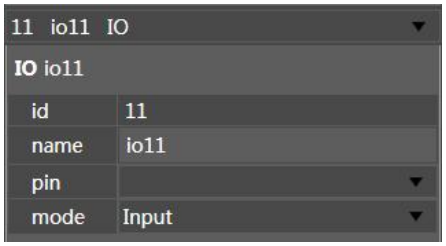
注：一个工程只能用一个 CAN 解析控件，作用于全局，使用详解请参考后续第八章，典例应用。

32. Modbus 解析器控件

布局	属性	说明
	id	控件 ID 号，不可更改
	name	控件名称，可改，默认：modprotocol+序号
	rxBufLen	用于接收数据的缓存的长度

注：一个工程只能用一个此控件，作用于全局，使用详解请参考后续第八章，典例应用。

33. IO 控件

布局	属性	说明
	id	控件 ID 号，不可更改
	name	控件名称，可改，默认：io+序号
	pin	引脚
	mode	该 IO 口是用作输入、输出或者中断

用法简介

IO 控件 mode 属性可配置 IO 引脚的模式，有输入、输出、中断输入三种模式。三种用法类似，主要是对控件的 val 属性访问。

输入模式：在脚本里读 val 属性。如 num2.val = io1.val 表示读取 IO 引脚的状态显示到整数控件上。

输出模式：在脚本里写 val 属性。如 io1.val = 1 表示设置 IO 引脚输出电平为高； io1.val = 0 表示设置 IO 引脚输出电平为低。

中断输入模式：如 num2.val = io1.val 表示读取 IO 引脚的状态显示到整数控件上。区别于普通输入模式是，中断输入模式可以在 IO 输入引脚的电平发生翻转时，可触发中断事件。

注意：

IO 引脚输出输入电平都为 3.3V。

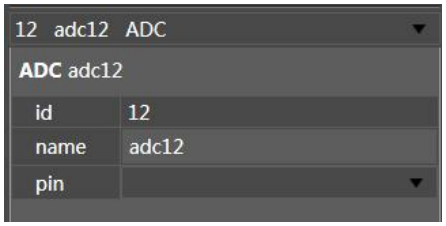
若输入电平高于 3.3，可能会损坏引脚，这种情况下需要进行分压等电路设计，保证输入电平电压为 3.3V 左右。

配置为输出时输出电压为 3.3V。若设备的驱动电压会高于 3.3V，需要搭配驱动电路，如 MOS 管、继电器等，也可以联系我司业务员购买驱动电路或者定制。

当有多个 IO 控件时，引脚号不要有互相冲突，否则编译会报错。

使用详解请参考后续第八章，典例应用。

34. ADC 控件

布局	属性	说明
	id	控件 ID 号，不可更改
	name	控件名称，可改，默认：adc+序号
	pin	引脚

用法简介

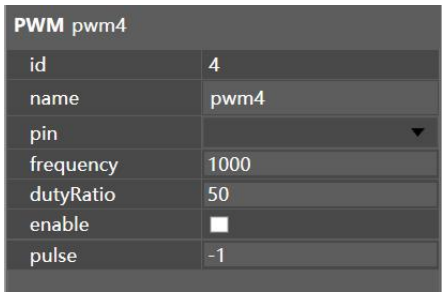
访问 val 属性可读取 ADC 引脚上的电压，如 num2.val = adc3.val 表示获取引脚的电压赋值为整数控件显示单位 mV。

注意：**输入电压不能高于 3.3V**。若需要采集的电压高于 3.3V，则需要设计电路降压处理，如电阻分压等。

注：使用详解请参考后续第八章，典例应用。

35. PWM 控件

布局	属性	说明



id	控件 ID 号，不可更改
name	控件名称，可改，默认： pwm+序号
pin	引脚号
frequency	频率。单位 Hz，可配置范围 1hz~1Mhz
dutyRatio	高电平占空比，0~100。
enable	是否使能。1 输出使能，0 输出不使能
pulse	输出脉冲数量，默认为-1，表示持续输出。当 pulse 大于 0 时，可输出指定脉冲个数，每自动输出一个脉冲个数，pulse 自动减 1，当减为 0 时停止输出

用法简介

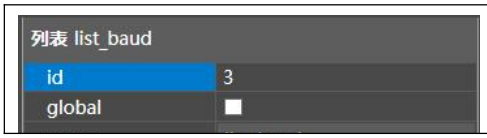
若需要持续输出 PWM 脉冲，配置 pulse 为-1，控制 enable 属性可开启或关闭 PWM 输出功能；

若需要输出指定脉冲个数，可设置 pulse 为要输出的个数，控制 enable 属性可开启或关闭 PWM 输出功能；

注：PWM 脉冲电压为 3.3V。大多数设备的驱动电压可能会高于 3.3V，因此这种情况下，需要搭配驱动电路，如 MOS 管、继电器等，也可以联系我司业务员购买驱动电路或者定制。

使用详解请参考后续第八章，典例应用。

36. 列表控件



属性	介绍
Id	只读属性只在编辑工程可改变
Global	全局属性，勾选时控件可作用于全局，全局调用时在前加页名称

Name	控件名称，只在工程编辑时可更改
x	控件 x 坐标，脚本可读写
y	控件 y 坐标，脚本可读写
width	控件宽，脚本可读写
height	控件高，脚本可读写
Locked	位置和大小锁定，只在编辑时有效，工程运行时无效
Visible	控件可视性，脚本可读写。Visible=0 控件不可见　visible=控件可见
opacity	控件透明度，脚本可读写（仅 s 系列支持透明度）
enable	控件使能，脚本可读写，enable=0 使能关闭　enable=1 使能打开
Items	列表条目，脚本可读写，编辑工程时在此处编辑列表内容
itemcount	条目数量，脚本可读写。
maxitem	最大条目数量，运行时不可读写
sel	选中的序号，从 0 开始，脚本可读写
Selex	勾选之后在选中项未改变时仍会触发“值改变事件”的脚本
Selel	勾选之后仅在触摸时会触发“值改变事件”的脚本
ori	控件方向，脚本可读写。Ori=0 横向，ori=1 纵向
itemspc	列表条目的间距，脚本可读写
bgtype	控件背景类型，脚本只读，bgtype=1 颜色切换　bgtype=3 图片切换
bgcolor	控件背景颜色，脚本可读写，十六进制颜色格式
selbgcolor	选中项背景颜色，脚本可读写
font	控件内容选择的字库，脚本可读写，读写值为字库序号
fontcolor	控件本文的颜色，脚本可读写
selfontcolor	选中项字体颜色，脚本可读写
charspc	字符间距，脚本可读写
txthoragn	文本的水平对齐方式，脚本可读写 0 左 1 中 2 右
txtveragn	文本的垂直对齐方式，脚本可读写 0 上 1 中 2 下
bdwidth	边框宽度，脚本只读，编辑工程时可更改
bdcolor	边框颜色，脚本只读，编辑工程时可更改
Bdwith1	内边框宽度，脚本只读，编辑工程时可更改
Bdcolor1	内边框颜色，脚本只读，编辑工程时可更改

用法简介

list 控件可用来做单选下拉框、菜单等。 常用属性 sel，选中项的序号，可读可写。

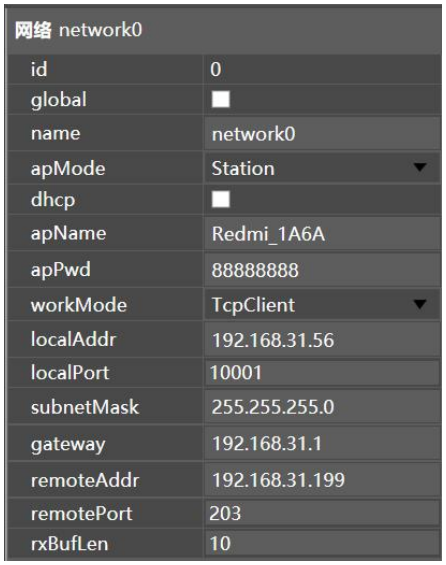
常用控件专有函数

string　getItem(int　index)获取对应项的文本

void setItme(int index, string item)设置对应项的文本

更多详细用法，可参考简易示例【简易串口助手_list 控件用法.pix】地址 http://www.sany-semi.com/list-268-1.html

37. 网络控件(wifi)

布局	属性	说明
	id	控件 ID 号，不可更改
	name	控件名称，可改
	apMode	wifi 模式。station 模式、softap 模式。默认为 station
	dhcp	是否使能 dhcp。目前该功能保留。默认不使能
	apName	wifi 名称
	apPwd	wifi 密码
	workMode	工作模式：TcpClient、TcpServer、Udp、Mqtt
	localAddr	本地地址
	localPort	本地端口
	subnetMask	本地子网掩码
	gateway	本地网关
	remoteAddr	远端地址
	remotePort	远端端口
	rxBufLen	接收数据缓存长度

网络控件方法介绍

int openAp()
功能：
1、station 模式下为 连接到 wifi
2、ap 模式下为 打开 wifi
参数：无
返回值：0:成功；1 失败

int closeAp()
功能：
1、station 模式下为 断开已连接的 wifi；
2、ap 模式下为 关闭 wifi。
参数：无
返回值：0:成功；1 失败；

int openLocalPort()
功能：打开本地端口，TCP 服务器模式下有效
参数：无
返回值：0:成功；1 失败；

int closeLocalPort()
功能：关闭本地端口，TCP 服务器模式下有效
参数：无
返回值：0:成功；1 失败；

int connectRemote()
功能：连接到远端，TCP 客户端模式下有效
参数：无
返回值：0:成功；1 失败；

int disconnectRemote()
功能：断开与远端的连接，TCP 客户端模式下有效
参数：无
返回值：0:成功；1 失败；

int send(byte[] buffer,int offset, int len)
功能：发送数据到远端
参数：buffer：要发送数据的数组名
offset：数组起始下标
len：要发送的字节长度
返回值：0:成功；1 失败；

int sendTo(byte[] buffer,int offset, int len, int connectId)
功能：向指定连接发送数据，TCP 服务器模式下有效
参数：buffer：要发送数据的数组名
offset：数组起始下标
len：要发送的字节长度
connectId：客户端连接号 0
返回值：0:成功；1 失败；

int sendString(string str)
功能：发送数据到远端
参数：str：要发送的字符串
返回值：0:成功；1 失败；

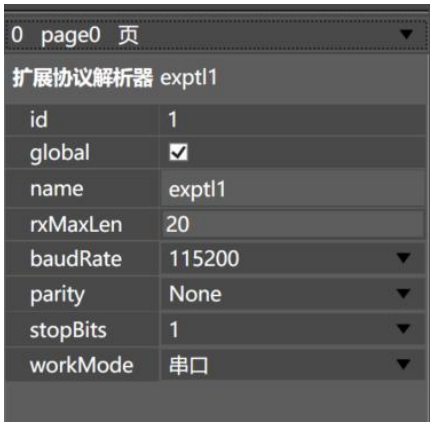
int sendStringTo(string str , int connectId)
功能：向指定连接发送数据，TCP 服务器模式下有效
参数： str：要发送的字符串
connectId：客户端连接号 0
返回值：0:成功；1 失败；

int publish(string topic,string msg)
功能：MQTT 模式下，发布主题
参数：topic：主题名称
msg：消息字符串

返回值：0:成功；1 失败；
int subscribe(string topic) 功能：MQTT 模式下，订阅主题 参数： topic：主题名称 返回值：0:成功；1 失败；

实际使用，请参考第八章 实验 13 WIFI 通信实验。

38. 扩展协议解析器控件

布局	属性	说明
	id	控件 ID 号，不可更改
	name	控件名称，可改
	rxMaxLen	用于接收数据的缓存的长度。
	baudRate	波特率。
	parity	校验位
	stopBits	停止位
	workMode	工作模式：普通串口、modbus

扩展协议解析器控件方法介绍

仅有扩展模块的串口屏，即型号名称后缀是 04 或者 05 结尾的，支持扩展串口。扩展串口的使用主要是对扩展协议解析器操作。

扩展串口与主串口操作类似，扩展串口调用相关串口发送函数的时候需要在函数前面加上控件名称，如下：

主串口发送字符串方式为　uartSend(“hello”);
扩展串口发送字符串方式为 exptl1.uartSend(“hello”); //这里控件名称为 exptl1，以实际为准

更多的串口发送函数，参考第四章的串口相关函数介绍。

第四章 函数的介绍和用法

介绍：串口屏脚本中的函数为内置函数，类似系统提供的工具，完成一些脚本不能实现的功能。

使用：使用方式同 C 语言。

1. Flash 操作

1.1. Flash 操作简介

某些时候，用户需要将一些数据保存在 Flash 中，用以掉电保存。在对 Flash 进行读写之前，需要在项目设置中，配置使用的 Flash 大小，默认情况下用户可以使用的 Flash 空间为 0 个字节。

用户 Flash 空间的设置方法为：VP 软件左上菜单栏->工具->项目设置->用户 Flash 大小，最大空间是 4096 字节，设置一个合适的大小->保存；空间的使用则是通过相关函数完成。

由于 Flash 擦除次数有限（通常为 10 万次），因此需要频繁写入的数据不建议保存到 Flash 中。

为了降低擦除 Flash 的次数，在调用写入相关函数之后，数据不会直接写入 Flash，需要调用保存函数才会实际写入。

1.2.用户 flash 写入整数 - fwInt

介绍	向 Flash 中写入整数，每个整数占用 4 个字节的存储空间。
定义	void fwInt(int offset , int val);
参数	offset：flash 中的偏移（可理解为空间存储数据的起始位置） val：要写入的整数 备注：如果 offset 小于 0 或大于 flash 的空间范围，则不执行写入
使用	fwInt(0，100);　　// 写入第一个整数 100，占用 4 个字节 fwInt(4，200);　　// 写入第二个整数 200，占用 4 个字节 fSave();　　// 将数据保存到 flash 中 // 将数据写入到 flash 中，这两个函数必须同时使用，才能将数据写入到 flash 空间中并保存；

1.3.用户 flash 读取整数 - frInt

介绍	从 Flash 中读取已存入的整数。
定义	int frInt(int offset); 返回值：所存储的值
参数	offset：flash 中的偏移（可理解为空间存储数据的起始位置） 备注：如果 offset 小于 0 或大于 flash 的范围空间，那么返回 0

使用	// 读取 flash 空间中的整数并赋给整数控件 num1 显示（偏移数需和写入的偏移一致）
	num1.val = frInt(0);
	// 读取 flash 空间中的整数并赋给整数控件 num2 显示（偏移数需和写入的偏移一致）
	num2.val = frInt(4);
	// 和写入函数配合使用，即可完成数据读写的整个操作；

1.4.用户 flash 写入浮点数 – fwFloat

介绍	向 Flash 中写入浮点数，每个浮点数占用 4 个字节的存储空间。
定义	void fwFloat(int offset, float val);
参数	offset：flash 中的偏移（可理解为空间存储数据的起始位置） val：要写入的浮点数 备注：如果 offset 小于 0 或大于 flash 的范围空间，那么不执行写入

使用	fwFloat(0，10.0)； // 写入浮点数 10.0，占用 4 个字节
	fwFloat(4，20.0)； // 写入浮点数 20.0，占用 4 个字节
	fSave()； // 将数据保存到 flash 中
	// 将数据写入到 flash 中，这两个函数必须同时使用，才能将数据写入到 flash 空间中并保存；

1.5.用户 flash 读取浮点数 – frFloat

介绍	从 Flash 中读取已存入的浮点数。
定义	float frFloat(int offset); 返回值：所存储的浮点数
参数	offset：flash 中的偏移（可理解为空间存储数据的起始位置） 备注：如果 offset 小于 0 或大于 flash 的范围空间，那么返回 0

使用	// 读取 flash 空间中的整数并赋给整数控件 num1 显示（偏移数需和写入的偏移一致）
	numf1.valf = frFloat(0);
	// 读取 flash 空间中的整数并赋给整数控件 num2 显示（偏移数需和写入的偏移一致）
	numf2.valf = frFloat(4);

	// 此函数和上一页的写入函数配合使用，即可完成数据读写的整个操作；

1.6.用户 flash 写入字符串 - fwString

介绍	向 Flash 中写入字符串。如果字符串的长度为 n，则占用 2n+2 个字节的存储空间。
定义	void fwString(int offset,string val);
参数	offset：flash 中的偏移（可理解为空间存储数据的起始位置） val：要写入的字符串 备注：如果 offset 小于 0 或大于 flash 的范围空间，那么不执行写入

使用	fwString(0，” Good”)； // 写入字符串 Good，占用 10 个字节的存储空间
	fwString(10，” 12345”)； // 写入字符串 12345，占用 12 个字节的存储空间
	fSave()； // 将数据保存到 flash 中
	// 将数据写入到 flash 中，这两个函数必须同时使用，才能将数据写入到 flash 空间中并保存；

1.7.用户 flash 读取字符串 - frString

介绍	从 Flash 中读取已存入的字符串。从起始位置一直读取，直到读取到两个连续的 0 为止。
定义	string frString(int offset); 返回值：所存储的字符/字符串
参数	offset：flash 中的偏移（可理解为空间存储数据的起始位置） 备注：如果 offset 小于 0 或大于 flash 的范围空间，那么返回一个空字符串

使用	// 读取 flash 空间中的字符串并赋给文本控件 text1 显示（偏移数需和写入的偏移一致）
	text1.txt=frString(0);
	// 读取 flash 空间中的字符串并赋给文本控件 text2 显示（偏移数需和写入的偏移一致）
	text2.txt=frString(10);
	// 此函数和上一页的写入函数配合使用，即可完成数据读写的整个操作；

1.8.用户 flash 写入字节数组 - fwBytes

介绍	向 Flash 中写入字字节数组。
定义	void fwBytes(offset, byte array[ ], int start, int len);
参数	offset：flash 中的偏移 array：写入数据的字节数组 start：需要保存的数据的起始位置 len：要保存的数据长度 备注：如果 offset 小于 0 或大于 flash 的范围空间，那么不执行写入

使用	byte a[5]={0x01,0x02,0x03,0x04,0x05};
	// 写入偏移为 0，名称为 a，起始位置为 0，长度为 5 的字节数组
	fwBytes(0,a,0,5);
	// 将数据保存到 flash 中
	fSave();
	// 将数据写入到 flash 中，这两个函数必须同时使用，才能将数据写入到 flash 空间中并保存；

1.9.用户 flash 读取字节数组 - frBytes

介绍	从 Flash 中读取已存入的字节数组。
定义	byte frBytes(offset, byte array[ ], int start, int len);
参数	offset：flash 中的偏移 array：保存读取数据的字节数组 start：读取的数据在 array 中的起始位置 len：要读取的数据长度 备注：如果 offset 小于 0 或大于 flash 的范围空间，那么不读取。

使用	byte a[5]={ };
	// 读取 flash 空间中的字节数组（偏移数需和写入的偏移一致）
	frBytes(0,a,0,5);
	// 此函数和上一页的写入函数配合使用，即可完成数据读写的整个操作；

1.10. 用户 flash 写入保存 - fSave

介绍	保存用户写入的数据到 flash 中。
定义	void fSave();
参数	无

使用	// 在不同空间段存储不同数据
	fwInt(0，100);
	fwFloat(100，10.0);
	fwString(200，Good);

	fSave(); // 需要写入保存多条不同类型数据时，只需在最后写一条保存函数（考虑到 flash 的擦写次数有限）

2. SD 卡操作

2.1.读取 SD 卡中文件的字符串行数 - fileReadLineCount

介绍	读取文件的字符串行数（文件过大时不推荐使用，读取速度会变慢）。
定义	int fileReadLineCount(string filePath); 返回值：文件的行数；-1 表示读取失败
参数	filePath：文件路径

使用	// 读取 SD 卡中 Data.txt 文件的字符串行数,并用整数控件 num1.val 显示
	num1.val = fileReadLineCount(“Data.txt”);

2.2.读取 SD 卡中文件的某一行字符串 - fileReadLine

介绍	读取一行字符串
定义	string fileReadLine(string filePath, int row, int maxLen); 返回值：读取的字符串
参数	filePath：文件路径 row：要读取的字符串的行号，从 0 开始 maxLen：读取的最大字符数。

使用	// 读取 SD 卡中 Data.txt 文件的第二行字符串,并用文本控件 text1 显示
	text1.txt =fileReadLine(“Data.txt”,2,10);

2.3.读取 SD 卡文件的字节数 - fileReadLength

介绍	读取文件的字节数
定义	int fileReadLength(string filePath); 返回值：文件的字节数；-1 表示读取失败。
参数	filePath：文件路径

使用	// 读取 SD_WR 文件夹下 test.bin 文件的字节数，并由整数控件 num2.val 显示
	num2.val = fileReadLength("SD_WR/test.bin");

2.4.读取 SD 卡文件的字节数组 - fileReadBytes

介绍	读取字节数组
定义	int fileReadBytes(string filePath , int offset , byte array[ ] , int start , int len); 返回值：0 表示成功。
参数	filePath：文件路径 offset：文件中的偏移。 array：保存读取数据的字节数组 start：保存读取的数据的起始位置 len：要读取的数据长度 备注：如果 offset 小于 0，那么不执行读取；

使用	// 读取 SD_WR 文件夹下 test.bin 中的字节数组，偏移为 0，数组名 a，起始位置 0，长度为 1 个字节
	fileReadBytes("SD_WR/test.bin",0,a,0,1);
	// 在 num3.val 上显示读取的数据
	num3.val = a[0];

2.5.删除 SD 卡中的文件 - fileDelete

介绍	删除不需要的文件
定义	int fileDelete(string filePath); 返回值：0 表示成功。
参数	filePath：文件路径

使用	fileDelete("SD_WR/test.txt");
	fileDelete("SD_WR/test.bin");
	// 删除 SD_WR 文件夹下的 test.txt 、test.bin 文件
	备注：删除是删除 txt/bin 文件，文件夹被删除，删除后，可以不用新建 test.txt/test.bin 文件，下一次写入的时候会根据函数中的文件名自动新建。

2.6.SD 卡文件末尾写入一行字符串 - fileWriteLine

介绍	在文件末尾写入一行字符串
定义	int fileWriteLine(string filePath, string txt); 返回值：0 表示成功。
参数	filePath：文件路径 txt：要写入的字符串

使用	// 在 SD 卡的 txt 文件 Data 末尾行写入字符串“a12”,字符串可替换为文本控件的 txt 属性。
	fileWriteLine(“Data.txt”,” a12”);

2.7.SD 卡文件写入字节数组 - fileWriteBytes

介绍	在文件中指定位置写入字节数组
定义	int fileWriteBytes(string filePath,int offset,byte[] array, int start, int len); 返回值：0 表示成功。
参数	filePath：文件路径 offset：文件中的偏移。 array：写入数据的字节数组 start：数据的起始位置 len：数据长度 备注：如果 offset 小于 0，那么不执行写入；
使用	byte a[7] = {0x30,0x31,0x32,0x33,0x34,0x35,0x36}; // 将数组 a 中，从 0 开始的 7 个字节，保存到 SD 卡的 SD_WR 文件夹下 test.bin 文件中，起始位置 0。

	fileWriteBytes("SD_WR/test.bin",0,a,0,7);

2.8.重新挂载 SD 卡 - sdRemount

介绍	重新挂载 SD 卡。一般情况下，不需要调用此函数。只有在特殊情况下，，如使用 TF 卡延长卡带时，当插入 SD 卡后，系统不会主动检测到 SD 卡已插入，这个时候可考虑使用
定义	int sdRemount() 返回值：0 表示成功。
参数	
使用	

2.9.打开文件夹 - dirOpen

介绍	打开文件夹。配合 dirEnumItem 函数使用。
定义	int dirOpen(string path) 返回值：0 表示成功。
参数	path: 文件夹路径
使用	num0.val = dirOpen("Admin1");//打开 SD 卡文件夹 Admin1 返回为 0 表示打开成功

2.10.枚举文件夹内的条目(文件和文件夹) - dirEnumItem

介绍	枚举文件夹内的条目(文件和文件夹)。配合 dirOpen 函数和系统变量 sys_dir_item_type 使用。 调用 dirOpen 函数时，会从第一个条目开始枚举
定义	string dirEnumItem() 返回值：文件或文件夹的名称，枚举完成时返回空字符串。
参数	无

使用	text3.txt = dirEnumItem();//读取一个条目的名称
	num6.val = sys_dir_item_type;//条目的类型，缓存在系统变量 sys_dir_item_type 中

2.11.关闭文件夹 - dirClose

介绍	关闭文件夹。配合 dirOpen 和 dirEnumItem 函数使用。 调用 dirOpen 函数后，枚举结束后，需要调用 dirClose 函数来关闭文件夹
定义	int dirClose() 返回值：0 表示成功。
参数	无
使用	

3. 发送数据

3.1.外发字符串函数 – uartSend

介绍	串口发送字符串。 注意：发送的字符串没有‘\0’结尾，只有字符串内容。 如果发送的字符串内包含非 ASCII 字符，那么会根据项目设置的字符编码，进行编码后发送，因此不同的编码发送的数据不同。工程编码格式在项目设置中设置。
定义	void uartSend(string a)
参数	a：需要发送的字符串

使用	// 发送文本控件 text1 的字符
	uartSend（text1.txt）；
	// 直接发送“ABcd123”字符串
	uartSend（“ABcd123”）；

字符串中用到的转义符定义		
字符	**ASCII**	**含义**
\n	010	换行(LF)，将当前位置移到下一行开头
\\	092	代表一个反斜线字符''\'
\"	034	代表一个双引号字符

3.2.字节数组发送函数 –uartSendBytes

介绍	串口发送字节数组
定义	void uartSendBytes(byte[] array, int start, int len);
参数	array：发送字节数组 start：起始位置 len：长度

使用	// 定义一个数组 array
	byte array[5] = { 0x0F, 0xF0, 0xFF, 0x33, 0x03};
	// 通过串口发送数组 array，从下标 0 开始，长度为 5。
	uartSendBytes(array, 0, 5);
	// 具体用法可参考协议解析器说明文档，或者下载官网的参考工程。

3.3.CAN 口发送字节数组函数 – canSendBytes

介绍	CAN 版本屏幕中，CAN 口发送字节数组函数
定义	void canSendBytes(int id，byte[] array, int len，int fmd);
参数	id：报文 id array：保存发送数据的字节数组 len：发送的数据长度 fmd：帧模式。1：标准数据帧；2：扩展数据帧；3：标准远程帧；4：扩展远程帧

使用	byte a[8]={0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c};
	// 发送 id 为 0x12345678，名称为 a,长度为 7，帧模式为 2（扩展数据帧）的字节数组
	canSendBytes(0x12345678,a,7,2);

3.4.在 modbus 上发送数据函数 – modbusSendBytes

介绍	带 Modbus 版本中，在 modbus 上发送数据
定义	int modbusSendBytes(int addr, int fcode, byte[] data, int len, int timeout); 返回值：0 成功，-1 超时
参数	addr：从站地址 fcode：功能码 data：保存发送数据的字节数组 len：发送的数据 data 长度 timeout：超时时间，单位毫秒

使用	// 从机按照 modbus 协议发送数据
	modbusSendBytes(mod.addr, mod.fcode, buf, 1+num*2, 0);
	// 此函数是串口屏作为从机使用时使用，因篇幅过大，请下载 modbus 从机例程参考，工程中有详细的使用说明

3.5.在 modbus 上发送读取帧 - modbusRead

介绍	带 Modbus 版本中，在 modbus 上发送读取帧
定义	int modbusRead(int addr, int fcode, int startaddr, int count, int timeout); 返回值：0 成功，-1 超时
参数	addr：地址 fcode：功能码 startaddr：起始地址 count：数量 timeout：超时时间，单位毫秒

使用	// 从 Modbus 总线读数据，设备地址 0x01,功能码 0x03,起始地址 0x0000,读寄存器数量 2，超时 50mS
	modbusRead(0x01,0x03,0x0000,2,50);
	// 可在官网下载相应工程参考。

3.6.在 modbus 上发送写入帧 - modbusWrite

介绍	带 Modbus 版本中，在 modbus 上发送写入帧
定义	int modbusWrite(int addr, int fcode, int outputaddr, int val, int timeout); 返回值：0 成功，-1 超时
参数	addr：地址 fcode：功能码 outputaddr：输出地址 val：输出值

	timeout：超时时间，单位毫秒

使用	// 在 Modbus 总线上发送写入帧，设备地址 0x01,功能码 0x06,输出地址 0x0064,输出值 0x02，超时 50mS
	modbusWrite(0x01,0x06,0x0064,0x02,50);
	// 可在官网下载相应工程参考。

3.7.在 modbus 上发送写入帧（数组） - modbusWrites

介绍	带 Modbus 版本中，在 modbus 上发送写入帧
定义	int modbusWrites (int addr, int fcode, int outputaddr, int count, byte[] val, int len, int timeout); 返回值：0 成功，-1 超时
参数	addr：地址 fcode：功能码 outputaddr：输出地址 count：输出值数量 val：值数组 len：值数组长度 timeout：超时时间，单位毫秒

使用	// 在 Modbus 总线上发送写入帧数组，设备地址 0x01,功能码 0x06,输出地址 0x0064,输出值数量 1，值数组 a,数组长度 8，超时 50mS
	modbusWrites(0x01,0x06,0x0064,1,a,8,50);

4. 字符串处理

4.1.前退格函数 - stringTrimStart

介绍	移除字符串首部的部分字符，返回新字符串。
定义	string stringTrimStart(string s,int count)
参数	s：字符串 count：移除的字符数量

使用	var1.txt = “123456”；
	// 移除字符串的前 3 个字符，变为“456”。
	var1.txt =stringTrimStart(var1.txt, 3);

4.2.后退格函数 - stringTrimEnd

介绍	移除字符串尾部的部分字符，返回新字符串。
定义	string stringTrimEnd(string s,int count)
参数	s：字符串 count：移除的字符数量

使用	var1.txt = “123456”；
	// 移除字符串的前 3 个字符，变为“123”。
	var1.txt = stringTrimEnd(var1.txt, 3);

4.3.在字符串中查找某一字符串的出现位置 - stringIndex

介绍	查找指定字符串在目标字符串中的位置。从 0 开始计算，即第一个字符的位置为 0
定义	int stringIndex(string str1, int start, string str2); 返回值：要查找的字符串的起始位置。如果没找到，返回 -1。
参数	str1：要检索的字符串 start：起始位置，从 0 开始。 str2：被检索的字符串

使用	// 在字符串“text123”中查找“x”的出现位置，并由整数控件 num4 显示，位置会返回 2
	num4.val = stringIndex(“text123”,0,"x");
	// 备注：如果找到，返回位置；否则返回 -1。

4.4.返回某一字符串的字符个数 - stringLength

介绍	返回字符串的长度，不包括空结束字符。
定义	int stringLength(string str); 返回值：字符串的长度，不包含空结束字符
参数	str：字符串

使用	// 返回字符串“text123”中的字符个数，并由整数控件 num5 显示，个数会返回 7
	num5.val = stringLength("text123");
	// 备注：结束符不计算，但会将空格计算在内。

4.5.返回某一字符串的子字符串 - stringSub

介绍	返回指定字符串的子字符串。
定义	string stringSub(string str, int start, int len); 返回值：子字符串
参数	str：字符串 start：起始位置 len：长度

使用	// 返回字符串“text123”中的某段字符，位置从第一个字符开始，向后取 4 个字符，并由文本控件 text2 显示，字符串会返回“text”
	text2.txt = stringSub("text 123", 0, 4);

4.6.在字符串的指定位置插入字符串 - stringInsert

介绍	在字符串的指定位置插入字符串。
定义	string stringInsert(string str, int offset, string str1) 返回值：新的字符串
参数	str：字符串 offset：偏移 str1：要插入的字符串 offset 从 0 开始，如果小于 0 或者大于字符串的长度，返回原字符串。
使用	// text2.txt 显示的文本为“ABCtext 123”

	text2.txt = stringInsert("text 123", 0, "ABC");
	// text2.txt 显示的文本为“text 123ABC”
	text2.txt = stringInsert("text 123", 8, "ABC");

4.7.移除字符串中的部分字符 - stringRemove

介绍	在字符串的指定位置插入字符串。
定义	string stringRemove(string str, int offset, string len) 返回值：新的字符串
参数	str：字符串 offset：偏移 len：长度 offset 从 0 开始，如果小于 0 或者大于字符串的长度，返回原字符串。

使用	// text2.txt 显示的文本为“cd1234”
	text2.txt = stringRemove("abcd1234", 0, 2);
	// text2.txt 显示的文本为“abcd123”。第 7 个字符是“4”，只有一个字符了，所以只删除一个。
	text2.txt = stringRemove("abcd1234", 7, 2);

4.8.替换字符串中的字符 - stringReplace

介绍	替换字符串中的部分字符。
定义	string stringReplace(string str, string str1，string str2) 返回值：新的字符串

参数	str：初始字符串 str1：要替换字符串 str2：新的字符串 str1 和 str2 的长度可以不一样。如果 str1 的长度为 0，不执行替换，以免字符串终结符被破坏。如果 str2 的长度为 0，移除所有 str1。 空字符串的长度为 0。

使用	// text2.txt 显示的文本为“aeecd1234”
	text2.txt = stringReplace ("abcd1234", "b", "ee");
	// text2.txt 显示的文本为“acd123”。“b”被删除。
	text2.txt = stringReplace ("abcd1234", "b", "");

4.9.提取包含单个字符的字符串 - stringGet

介绍	提取包含单个字符的字符串。
定义	string stringGet(string str,int offset) 返回值：包含单个字符的字符串
参数	str：字符串 offset：偏移 如果 offset 小于 0 或者 offset 超过了 str 的长度，那么返回空字符串。

使用	// text2.txt 显示的文本为“c”
	text2.txt = stringGet ("abcd1234", 2);
	// text2.txt 显示的文本为“”。
	text2.txt = stringGet ("abcd1234", -1);

4.10. 修改字符串的单个字符 - stringSet

介绍	修改字符串的单个字符。
定义	string stringSet(string str,int offset, string str1) 返回值：修改了该字符的新字符串
参数	str：字符串 offset：偏移 str1：新的字符 如果 offset 小于 0 或者 offset 超过了 str 的长度，那么该函数不做任何操作。如果 str1 长度为 0，不执行任何操作。如果 str1 的长度大于 1，只取第一个字符。

使用	// text2.txt 显示的文本为“ab4d1234”。只取第一个字符。
	text2.txt = stringSet ("abcd1234", 2, "4");
	// text2.txt 显示的文本为“abcd1234”。
	text2.txt = stringSet ("abcd1234", 2, "");
	// 空字符的情况下不做任何操作

4.11. 将字符转换为小写形式 - stringLower

介绍	将字符串中的 ASCII 英文大写字母转换为小写。
定义	string stringLower(string str); 返回值：小写形式的字符串
参数	str：字符串

使用	text4.txt = stringLower("ABCD"); // 返回字母 ABCD 的小写，并由文本控件 text4 显示
	text4.txt = stringLower("abc"); // 返回 abc

4.12. 将字符转换为大写形式 - stringUpper

介绍	将字符串中的 ASCII 英文小写字母转换为大写。
定义	string stringUpper(string str); 返回值：大写形式的字符串
参数	str：字符串

使用	text4.txt = stringUpper("abcd"); // 返回字母 abcd 的大写，并由文本控件 text4 显示
	text4.txt = stringUpper("ABC"); // 返回 ABC
	text4.txt = stringUpper("123"); // 返回 123

5. 数学函数

5.1. 返回弧度角 x 的正弦：float sin(float x)

5.2. 返回弧度角 x 的余弦：float cos(float x)

5.3. 返回弧度角 x 的正切：float tan(float x)

5.4. 返回以弧度表示的 x 的反正弦：float asin(float x)

5.5. 返回以弧度表示的 x 的反余弦：float acos(float x)

5.6. 返回以弧度表示的 x 的反正切：float atan(float x)

5.7. 四舍五入：float round(float x)

5.8. 返回大于或等于 x 的最小的整数值：int ceil(float x)

5.9. 返回整数 x 的绝对值：int abs(int x)

5.10. 返回浮点数 x 的绝对值：float fabs(float x)

5.11. 求平方根：float sqrt(float x)

5.12. 返回小于或等于 x 的最大整数：float floor(float x)

5.13. 返回 x 的 y 次幂：float pow(float x,float y)

5.14. 返回 x 以底数为 10 时的对数：float log10(float x)

6. 类型转换

6.1.整型转字符串型 - intToString

介绍	将整数转换为字符串。
定义	string intToString(int i); 返回值：整数的字符串形式
参数	i：要转换的整数

使用	int a;
	a=2020;
	txt1.txt=intToString(a); // 文本控件 txt1 显示文本“2020”

6.2.浮点型转字符串型 - floatToString

介绍	将浮点数转换为字符串。
定义	string floatToString(float f,int num); 返回值：浮点数的字符串形式
参数	f：要转换的浮点数 num：要保留的小数位数

使用	float a;
	a=2020.0101;
	txt1.txt =floatToString (a，2);// 文本控件 txt1 显示文本“2020.01”

6.3.浮点型转字符串型 - floatToInt

介绍	将浮点数转换为整数。

定义	int floatToInt(float f); 返回值：浮点数的整数位
参数	f：要转换的浮点数

使用	float a;
	a=2020.0101;
	num1.val = floatToInt (a);// 整数控件 num1 显示整数 2020

6.4.字符串型转浮点型 - stringToFloat

介绍	将字符串转换为浮点数。 若字符串参数 s 不是浮点数，上位机模拟运行时会提示类型错误
定义	float stringToFloat(string s); 返回值：浮点数
参数	s：字符串形式的浮点数
使用	float a; a=stringToFloat（“2020.0101”); // a 的值变为浮点数 2020.0101

6.5.字符串型转整型 - stringToInt

介绍	将字符串转换为整数。 若字符串不是整数，上位机模拟运行时会提示类型错误
定义	int stringToInt(string s); 返回值：整数

参数	s：字符串形式的整数
使用	int a; a=stringToInt（“2020”）; // a 的值变为整数 2020

6.6.提取 Ascii 字符串 - bytesToAscii

介绍	从字节数组中提取 Ascii 字符串，一个字节代表一个字符。配合上位机协议解析器控件使用。 要处理非 ASCII 字符串，使用 bytesToString 函数。
定义	string bytesToAscii(byte[] array, int start, int len); 返回值：提取出来的字符串
参数	array：字节数组 start：起始位置 len：提取多少个字节
使用	byte a[5] = {0x31,0x32,0x33,0x34,0x35}; text3.txt = bytesToAscii(a,0,5);//text3 文本控件显示"12345"

6.7.提取标准字符串 - bytesToString

介绍	从字节数组中提取标准字符串。2 个字节表示一个字。通常配合上位机协议解析器控件使用。
定义	string bytesToString(byte[] array, int start, int len); 返回值：提取出来的字符串
参数	array：字节数组 start：起始位置

	len：提取多少个字节，必须是 2 的整数倍
使用	byte a[10] = {0x31,0x00,0x32,0x00,0x33,0x00,0x34,0x00,0x35,0x00}; text3.txt = bytesToString(a,0,10);//text3 文本控件显示"12345"

6.8.提取整数 - bytesToInt

介绍	从字节数组中提取整数。通常配合上位机协议解析器控件使用。 将从起始位置开始的 4 个字节，转换成整数
定义	int bytesToInt(byte[] array, int start, int mode); 返回值：提取出来的整数
参数	array：字节数组 start：起始位置 mode：大小端。0 为小端模式，1 为大端模式
使用	byte a[4] = {0x64,0x01,0x00,0x00}; num0.val = bytesToInt(a,0,0);//整数控件 num0 显示为 356 (即 0x00000164)

6.9.提取浮点数 - bytesToFloat

介绍	从字节数组中提取整数。通常配合上位机协议解析器控件使用。
定义	float bytesToFloat(byte[] array, int start, int mode); 返回值：提取出来的整数
参数	array：字节数组 start：起始位置

	mode：大小端。0 为小端模式，1 为大端模式
使用	byte a[4] = {0xC3,0xF5,0x48,0x40}; numf9.valf = bytesToFloat(a,0,0); //浮点数控件 numf9 显示 3.14

6.10. 提取短整数 –bytesToShort

介绍	从字节数组中提取短整数。通常配合上位机协议解析器控件使用。
定义	int bytesToShort(byte[] array, int start, int mode); 返回值：提取出来的短整数
参数	array：字节数组 start：起始位置 mode：大小端。0 为小端模式，1 为大端模式

使用	byte a[4] = {0xC3,0xF5};
	byte b[4] = {0x10,0x00};
	num6.val = bytesToShort(a,0,0); //整数控件 num6 显示 -2621
	num7.val = bytesToShort(b,0,1); //整数控件 num7 显示 4096

6.11. 提取无符号短整数 –bytesToUshort

介绍	从字节数组中提取无符号短整数。通常配合上位机协议解析器控件使用。
定义	int bytesToUshort(byte[] array, int start, int mode); 返回值：提取出来的无符号短整数
参数	array：字节数组

	start：起始位置 mode：大小端。0 为小端模式，1 为大端模式

使用	byte a[4] = {0xC3,0xF5};
	byte b[4] = {0x10,0x00};
	num6.val = bytesToUshort(a,0,0);//整数控件 num6 显示 62915
	num7.val = bytesToUshort(b,0,1);//整数控件 num7 显示 4096

6.12. 返回某一字节的十六进制字符串 - byteToHex

介绍	字节转换为十六进制字符串
定义	string byteToHex(byte val); 返回值：字节的十六进制字符串形式
参数	val：字节类型数据

使用	byte a[3] = {0x01,0x02,0x03};
	text3.txt = byteToHex(a[0]);//文本控件 text3 显示字符串“01”
	text4.txt = byteToHex(a[1]);//文本控件 text4 显示字符串“02”
	// 备注：只能返回数组中某一字节的字符串。

6.13. 整型转字节数组 - intToBytes

介绍	将整数保存到字节数组
定义	void intToBytes(int i , byte a[ ] ， int start , int mode); 返回值：无

参数	int i：要转换的整数 byte a[ ]：缓存转换后的数组 int start：缓存在 byte a[ ]中的起始位置 int mode：字节端序 0：小端 1：大端

使用	int i=123;
	byte a[4]；//**此函数转换后，占用四字节，所以数组长度为 4**
	//将整数 i 转换为 byte 型，缓存在数组 a 中，大端序
	intToBytes(i , a , 0 , 1);//转换结果为：00 00 00 7B

6.14. 浮点型转字节数组 - floatToBytes

介绍	将浮点数保存到字节数组。
定义	void floatToBytes(float f , byte a[ ] ， int start , int mode); 返回值：无
参数	float f：要转换的浮点数 byte a[ ]：缓存转换后的数组 int start：缓存在 byte a[ ]中的起始位置 int mode：字节端序 0：小端 1：大端

使用	//将浮点数 f 转换为 byte 型，缓存在数组 a 中，大端序
	byte a[4];//此函数转换后，占用四字节，所以数组长度为 4
	float b = 3.14;
	floatToBytes(b,a,0,0);//转换结果为：C3 F5 48 40

6.15. 返回某整数的十六进制字符串 - intToHex

介绍	整数转换为十六进制字符串
定义	string intToHex(int val); 返回值：字节的十六进制字符串形式
参数	val：整数类型数据

使用	// 返回整数 10 的十六进制字符串，并由文本控件 text4 显示，字符串会返回“0000000A”
	
	text4.txt = intToHex(10);

6.16. ascii 字符串转字节数组 - asciiToBytes

介绍	字符串转字节数组。一个字符转换为一个字节。 只能正确转换纯 ASCII 字符，要处理非 ASCII 字符串，请使用 stringToBytes 函数。
定义	void asciiToBytes(byte[] array, int offset, string str)；
参数	array：保存结果数据的字节数组 offset：保存数据的起始索引 str：要转化的字符串 如果 offset 小于 0，那么不做任何操作。
使用	byte arr[10]; asciiToBytes(arr , 2 , "123");// 数组 arr 中的内容 {00 00 31 32 33 00 00 00 00 00}

6.17. 字符串转字节数组 - stringToBytes

介绍	字符串转字节数组。一个字符转换为两个字节。
定义	void stringToBytes (byte[] array, int offset, string str);
参数	array：保存结果数据的字节数组 offset：保存数据的起始索引 str：要转化的字符串 如果 offset 小于 0，那么不做任何操作。
使用	byte arr[10]; StringToBytes(arr, 2,"12 上山");// 数组 arr 中的内容 {00 00 00 31 00 32 C9 CF C9 BD}

6.18. 将十六进制字符串转换为字节 - hexToByte

介绍	将十六进制字符串转换为字节。
定义	byte hexToByte(string str); 返回值：字节
参数	str：要转换的字符串
使用	示例： “a” -> 0x0a “aa” -> 0xaa “s” -> 0x0 //非 16 进制字符

6.19. 将十六进制字符串转换为整数 - hexToInt

介绍	将十六进制字符串转换为整数。
定义	int hexToInt(string str); 返回值：整数
参数	str：要转换的字符串
使用	示例： “1234” -> 0x1234 “123” -> 0x123 “123S4” -> 0x0 //非 16 进制字符

6.20. 将十六进制字符串转换为字节数组 - hexToBytes

介绍	将十六进制字符串转换为字节数组。 每两个字符表示一个字节。不足两个字符的前面补 0。
定义	int hexToBytes(byte[] array, int offset, string str); 返回：成功转换的字节数量。
参数	array：保存转换后数据的字节数组 offset：保存的起始索引 str：要转换的字符串 如果 offset 小于 0，那么不做任何操作。
使用	示例： “1234” -> 0x12 0x34

	“12345” -> 0x12 0x34 0x05 “123s5” -> 0x12 //非 16 进制字符串

6.21. 从字节数组中解码字符串 - stringDecode

介绍	从字节数组中解码字符串。支持非 ASCII 字符。 对于非 ASCII 字符，其编码结果受项目设置中的字符编码选项影响，函数按照设置的编码方式进行编码。
定义	string stringDecode(byte[] array, int start, int len); 返回值：解码的字符串。
参数	array：保存已编码字符串数据的字节数组 start：解码的起始索引 len：解码的字节数量

使用	// 数组
	byte sub[4] = {0x61,0x62,0x63,0x64};
	// 函数解码为字符串，返回字符由文本控件 text4 显示，此处返回 ‘abcd’
	text4.txt = stringDecode(sub,0,4);
	// 对照 ASCII 码表，0x61 十进制 97 查表对应小写字母 ‘a’ 。
	// 也可解码为其他字符，如 0x2A,0x2B，解码转换则为：* 和 +。

6.22. 将字符串编码为字节数组 - stringEncode

介绍	将字符串编码为字节数组。支持非 ASCII 字符。 对于非 ASCII 字符，其解码结果受项目设置中的字符编码选项影响，函数按照设置的编码方式进行解码。

定义	`int stringEncode(byte[] array, int offset, string str);` 返回值：编码后的字节长度
参数	array：保存编码后数据的字节数组 offset：编码后数据保存的起始索引 str：需要编码的字符串 注意检查 offset 的范围。如果 offset 小于 0，那么返回 0。

使用	// 定义数组，用于存储转换后的字节数据
	
	byte a[10];
	
	// 将字符串转换为字节数组，起始位置为 0，并缓存于数组 a 中，函数返回值为 10，由整数控件 num1 显示
	
	num1.val = stringEncode(a,0,”0123456789”);
	
	// 若想要查看编码转换后的字节数据，可以用日志控件打印显示出来：
	
	// 1. 拖一个日志控件，拉大宽度；
	
	// 2. 勾选控件的 hexMd 属性；
	
	// 3. 在上面转换函数脚本后面加上打印脚本：
	
	// log6 为日志控件名，a 为上面的缓存数组，0 为从 a[0]开始显示，打印 10 个字节
	
	log6.addBytes(a,0,10);
	
	// 最终日志控件十六进制显示：
	
	// 30 31 32 33 34 35 36 37 38 39
	
	// 转换为十进制分别为：
	
	// 48 49 50 51 52 53 54 55 56 57
	
	// 查 ASCII 表，则对应 0-9；
	
	// 其他字符，如+-*/，也可编码转换。

7. 其他函数

7.1.翻页 - showPage

介绍	跳转到某一页。
定义	void showPage(int a)
参数	a：页的 id 号

使用	// 跳转到 id 为 0 的页
	showPage（0）；
	// 跳转到名称为 main 的页，使用 id 属性获取页 id
	showPage(main.id);

7.2.亮度调节 - setBright

介绍	设置背光亮度，100 级可调。
定义	void setBright（int a）
参数	a：亮度。范围 0-100

使用	// 设置背光亮度为最大值
	setBright（100）；
	// 关掉背光
	setBright（0）；

7.3.随机数 - getRandom

介绍	取得一个随机数。

定义	int getRandom(int a, int b) 返回值：a~b 之间的一个随机数。包含 a 和 b
参数	a：下界值 b：上界值
使用	int ran; ran = getRandom（1,9999）;

7.4.串口屏校准 - screenAdjust

介绍	进入屏幕触摸校准模式
定义	void screenAdjust();
参数	无

使用	// 进入校准模式。
	screenAdjust();

7.5.CRC16 - crc16

介绍	计算字节数组的 CRC16。该值的结果受到项目设置中的“CRC 多项式”影响。
定义	int crc16(byte array[ ], int start, int len); 返回值：计算得出的 crc16 值
参数	array：需要计算的字节数组 start：计算的起始位置 len：计算的字节数量

使用	// 定义一个数组 array
	byte array[5] = { 0x0F, 0xF0, 0xFF, 0x33, 0x03};
	// 计算 array 中从 0 开始的 5 个字节的 CRC
	crc = crc16(array,0,5);
	// 具体用法可参考协议解析器说明文档，或者下载官网的参考工程。

7.6.计算 CRC16/MODBUS - crc16Modbus

介绍	计算字节数组的 CRC16/MODBUS。
定义	int crc16Modbus(byte[] array, int start, int len);
参数	array：需要计算的字节数组 start：计算的起始位置 len：计算的字节数量

使用	CRC_data = (com.rxBuf[23] << 8) | com.rxBuf[22];
	// 计算数组 com.rxBuf 中从 0 开始的 22 个字节的 crc
	CRC_value= crc16Modbus(com.rxBuf,0,22);
	if(CRC_data == CRC_value)
	{.....}

7.7.延时函数 - delay

介绍	脚本等待一段时间之后继续执行，不推荐使用。
定义	void delay(int x)
参数	x：延时的毫秒数

使用	// 延时 10ms
	delay(10);
	// 在脚本中使用延时函数，屏幕程序会阻塞在延时处，此时做不了其他操作，延时结束恢复正常，所以在使用时，应避免延时过长。若需要长延时，可使用定时器控件来实现

7.8.获取控件属性的 ID 号 - idof (宏定义)

介绍	控件属性的 ID 号为串口屏的固有属性，不会随用户的工程变化，可以使用该宏定义获取。一般用来开发调试时查询，配套的函数是 setPropInt、 getPropInt、 setPropFloat、 getPropFloat、 setPropStr、 getPropStr
定义	int idof(控件属性)
参数	

使用	// 获取整数控件 num0.val 的属性 ID 号为 48
	num2.val = idof(num0.val);
	//常见的属性 ID 列举如下
	整数控件的值 val---(48)
	浮点数控件的值 valf---(51)
	文本控件的文本 txt---(22)
	按钮控件的背景颜色 pbgColor---(101)
	进度条控件的值 val---(48)

7.9. 根据 ID 号设置控件属性(整数类型) - setPropInt

介绍	根据 ID 号，设置控件属性，限定控件属性为整数的情况。
定义	`void setPropInt(int pageId, int ctrlId, int propId, int val)`
参数	`pageId`：控件所在的页的 Id `ctrlId`：控件的 ID `propId`：控件属性的 Id `val`：新的值

使用	// 设置整数控件(所在页面 id 为 0，控件 id 为 0) 的值为 100
	
	setPropInt(0, 0, 48, 100);
	
	// 这里控件属性 id 为 48 为整数控件 val 的固有属性 id，可以 idof 查看

7.10. 据 ID 号获取控件属性(整数类型) - getPropInt

介绍	根据 ID 号，设置控件属性，限定控件属性为整数的情况。
定义	`int getPropInt(int pageId, int ctrlId, int propId)`
参数	`pageId`：控件所在的页的 Id `ctrlId`：控件的 ID `propId`：控件属性的 Id

使用	// 获取整数控件(所在页面 id 为 0，控件 id 为 0) 的值
	
	num2.val = getPropInt(0, 0, 48);
	
	// 这里控件属性 id 为 48 为整数控件 val 的固有属性 id，可以 idof 查看

7.11. 根据 ID 号设置控件属性(小数类型) - setPropFloat

介绍	根据 ID 号，设置控件属性，限定控件属性为小数的情况。
定义	`void setPropFloat(int pageId, int ctrlId, int propId, float  val)`
参数	`pageId`：控件所在的页的 Id `ctrlId`：控件的 ID `propId`：控件属性的 Id

	val：新的值

使用	// 设置整数控件(所在页面 id 为 0，控件 id 为 51) 的值为 3.12
	
	setPropFloat (0, 0, 51, 3.12);
	
	// 这里控件属性 id 为 51 为浮点数控件 valf 固有属性的 id，可以 idof 查看

7.12. 根据 ID 号获取控件属性(小数类型) - getPropFloat

介绍	根据 ID 号，设置控件属性，限定控件属性为小数的情况。
定义	float getPropFloat(int pageId, int ctrlId, int propId)
参数	pageId：控件所在的页的 Id ctrlId：控件的 ID propId：控件属性的 Id

使用	// 获取浮点数控件(所在页面 id 为 0，控件 id 为 3) 的值
	
	num2.valf = getPropFloat (0, 3, 51);
	
	// 这里控件属性 id 为 51 为浮点数控件 valf 固有属性的 id，可以 idof 查看

7.13. 根据 ID 号设置控件属性(字符串类型) - setPropString

介绍	根据 ID 号，设置控件属性，限定控件属性为字符串类型的情况。
定义	void setPropString(int pageId, int ctrlId, int propId, string val)
参数	pageId：控件所在的页的 Id ctrlId：控件的 ID propId：控件属性的 Id val：新的字符串

使用	// 设置文本控件(所在页面 id 为 0，控件 id 为 4) 的字符串为 hello
	
	setPropString (0, 4, 22, “hello”);
	
	// 这里控件属性 id 为 22 为文本控件 txt 固有属性的 id，可以 idof 查看

7.14. 根据 ID 号获取控件属性(字符串类型) - getPropString

介绍	根据 ID 号，获取控件属性，限定控件属性为字符串类型的情况。
定义	string getPropString (int pageId, int ctrlId, int proId)
参数	pageId：控件所在的页的 Id ctrlId：控件的 ID propId：控件属性的 Id

使用	// 获取文本控件(所在页面 id 为 0，控件 id 为 4) 的字符串
	
	text6.txt = getPropString(0, 4, 22,);
	
	// 这里控件属性 id 为 22 为文本控件 txt 固有属性的 id，可以 idof 查看

7.15. 软件复位 - reboot

介绍	重启串口屏，软复位。
定义	void reboot()
参数	无

使用	// 软件复位。
	
	reboot();

7.16. crc8 算法初始化参数 - crc8init

介绍	初始化 crc8 的参数，包括多项式、初始值、异或值、是否输入数据反转、是否输出数据反转。
定义	void crc8init(byte poly, byte init, byte xorout, int refin, int refout);
参数	poly：多项式，默认为 x8+x2+x+1,即 0x07 init：初始值，默认为 0x00

	xorout：异或值 refin：是否输入数据反转 refout： 是否输出数据反转

使用	crc8init(0x07,0,0,0,0);
	//系统默认 crc8 算法初始化参数为 多项式 poly=0x07，初始值 Init=0，异或值 xorout=0，refin=0，
	refout=0，如需使用其他类型的 CRC8 算法，需要调用该函数进行修改

7.17. crc8 算法 - crc8

介绍	计算 crc8。
定义	byte crc8(byte[] data,, int start , int int len);
参数	data ：要计算的 byte 类型数组 start：数组下标起始位置 len：计算的字节长度

使用	byte a[5] = {0x01,0x02,0x03,0x04,0x05};
	num6.val = crc8(a,0,5);
	//计算数组 a 的 5 个字节的 crc8，计算结果 num6 控件显示为 188，即 0xBC
	//网页在线计算工具 http://www.ip33.com/crc.html

第五章 系统变量介绍

编号	变量名	说明	取值范围
1	sys_pid	当前页面 ID（只读）	0~255
2	sys_baud	当前波特率（只读）	设备支持的波特率：4800，9600，14400，19200，38400，56000，57600，115200，256000，460800，921600
3	sys_light	亮度（可读写）	0~100
4	sys_sleep	是否休眠（可读写）	0 不休眠，1 休眠
5	sys_slp_nu	无串口数据自动睡眠时间（可读写）	单位:秒 0~255 0 表示不睡眠
6	sys_slp_nt	无触摸操作自动睡眠时间（可读写）	单位:秒 0~255 0 表示不睡眠
7	sys_wup_bt	睡眠模式下触摸唤醒开关（可读写）	0 睡眠后触摸不会唤醒 1 睡眠后触摸唤醒
8	sys_wup_bu	睡眠模式下串口数据唤醒开关（可读写）	0 睡眠后串口不会唤醒 1 睡眠后串口唤醒
9	备用		

10	备用		
11	sys_bkcmd	设置指令执行结果的返回 （只读）	0x00 不返回结果 0x01 只返回成功的结果 0x02 只返回失败的结果 0x03 成功或者失败都返回结果
12	rtc_year	使用 RTC 时获取时间年（可读写）	0~99，使用时需要加上偏移 2000
13	rtc_month	使用 RTC 时获取时间月（可读写）	1-12
14	rtc_day	使用 RTC 时获取时间日（可读写）	1-28/29/30/31
15	rtc_week	使用 RTC 时获取时间周（只读）	周一到周六为：1-6，周日：0
16	rtc_hour	使用 RTC 时获取时间时（可读写）	00:00-23:00
17	rtc_minute	使用 RTC 时获取时间分（可读写）	00-59
18	rtc_second	使用 RTC 时获取时间秒（可读写）	00-59
19	sys_slp_cls	休眠等级（可读写）	0 为普通休眠（熄灭背光），1 为深度休眠
20	sys_run_time	系统运行时间（只读）	0~n，单位秒 系统运行时，该变量从 0 累加
21	sys_nt_time	系统无触摸时间（只读）	0~n，单位秒 无触摸时，该变量从 0 累加；有触摸时，自动置零

22	sys_nu_time	系统无串口数据时间（只读）	0~n，单位秒 无接收到串口数据时，该变量从 0 累加；有接收到串口数据时，自动置零
23	sys_modbus_addr	主串口 modbus 从机地址（可读写）	0~255
24	sys_can_baud	can 通信的波特率（可读写，仅适用于支持 can 通信的串口屏型号）	100K~1M
25	sys_uid	串口屏唯一 ID（只读）	4 字节的 int 类型，0~0xFFFFFFFF
26	sys_stopbits	主串口通信 停止位（可读写）	1: 1 位; 2: 2 位; 3: 1.5 位
27	sys_parity	主串口通信 校验位（可读写）	0: 无; 1: 奇校验; 2: 偶校验
28	sys_dis_angle	显示角度（可读写）	0（0 度）， 1（90 度）， 2（180 度）， 3（270 度），
29	sys_mem_per1	系统运行内存使用情况（只读）	0-100%
30	sys_mem_per2	系统 GUI 内存使用情况（只读）	0-100%

31	sys_dir_item_type	SD 卡文件条目类型，只能配合 readdir() 函数使用（只读）	0 文件， 1 文件夹
32	备用		
33	sys_buzzer_run	蜂鸣器控制	取值范围{-1，0~n}，单位 ms， -1 常开， 0 关闭， 其他值(不小于100ms)表示运行n毫秒 后，自动关闭
说明：以上 RTC 操作在硬件支持的屏幕上有效。RTC 系统变量以外的变量，本次上电事件内修改有效，掉电后改为串口屏工程内设定值。			

第六章 串口指令的简介和用法

通讯指令：通讯指令，是用户向串口屏模块发送的指令。

脚本指令：脚本指令，是串口屏模块内部脚本，计算数据，控制页面控件，以及发送数据给用户。

这里介绍的是通信指令。

功能	用户通过通讯指令控制串口屏模块。 通讯指令通过串口发送，发送方式为 ASCII 码方式
基本格式	命令码标识 + 参数 1（+参数 2） + 结束符[回车换行，即\r\n，十六机制为 0x0D,0x0A]
特点	通过精简的方式控制串口屏模块，命令极少。 串口数据为 ascii 码方式，更加人性化，也方便初学者上手，曲线透传数据除外。 串口命令以回车换行作为帧尾，没有数据帧头，减少客户组合数据时的操作。

1. 串口命令的发送格式举例

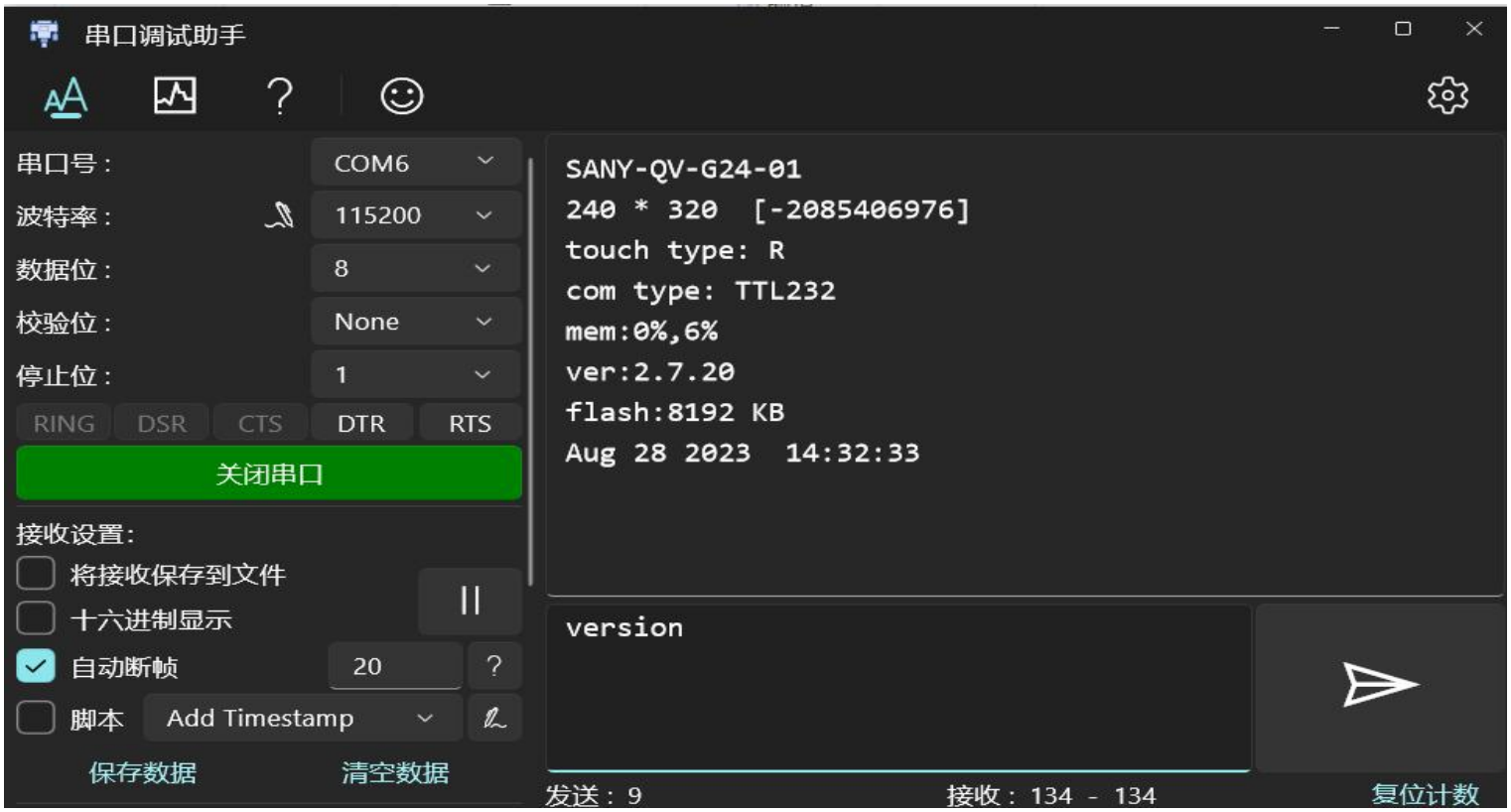
用 USB 转串口(TTL 或 485)工具、连接串口屏的 4pin 座子。**注意不是 typeC 口，typeC 口只能用于下载，不能用于通信。**

示例 1

打开串口助手，设置正确的波特率，打开串口。输入命令 version，勾选发送新行，点击发送，串口助手即可收到串口屏返回的版本信息。有的串口助手没有【发送新行】选项，需要手动输入回车换行，参考示例 2

示例 2

打开串口助手，设置正确的波特率，打开串口。输入命令 version 和[Enter 键]，点击发送，串口助手即可收到串口屏返回的版本信息



示例 3

用单片机程序发送。

假设用户发送串口数据的函数定义为： void uart_send(char *ptr)；

那么页跳转的方式为：uart_send(“page 8\r\n”);，串口数据为 page 8(回车换行)

2. 查看所有指令 ：help

指令名	功能	参数及举例
help	回传所有可用 串口指令	无参数。 串口数据举例：**help(+回车换行)** 回传数据： help - help page - page name/page id wset - wset name value wget - wget name sset - sset name value sget - sget name event - event name val click - click name adjust - adjust factory - return to factory,only valide 100ms before power on addt - HEX format pisx - HEX format version - get information reset - reset 1

3. 页面跳转指令 ： page

指令名	功能	参数及举例
page	页面跳转	page name/id(+回车换行) 参数: 页面名称/页面 ID 举例: 跳转到页面“main”，页面”main”的 id 为 0， 则命令为：page main 或者 page 0 回传数据：参看后面通讯指令返回值介绍 注：只有此命令可以使用 ID 号来跳转。

4. 控件属性设置指令 ：wset

指令名	功能	参数及举例

wset	设置页面以及控件的属性值	wset name value(+回车换行)
		参数：页面\控件名称·属性名称　属性值。
		举例：
		wset text1.txt　“明天会更好”
		//设置当前显示页面的文本控件 text1 的 txt 属性为“明天会更好”
		wset main.text1.txt　“明天会更好”
		//设置页面‘main’中文本控件“text1”的“txt”为“明天会更好”
		wset num1.val 100
		//设置当前页面数字控件 num1 的 val 属性为 100
		回传数据: 执行结果。参看后面通讯指令返回值介绍。

说明：

页面\控件名称是带有可查找路径的名称，要设置页面的属性，则用“wset 页面名称.属性名称”，如：

wset main.bgColor 0xFFFF0000　//设置页面 main 的背景颜色为红色

wset main.num1.val 123　//设置页面 main 中整数控件 num1 的当前值为 123

属性值有三种类型，整数，小数，字符串。注意字符串类型需要用双引号括起来。中英文字符串，注意制作串口屏工程时选择的字符编码（工具->项目设置），和发送命令的用户编译环境选择的字符编码一致。如：VP 中字符编码选择 GBK，那么用户发送的字符串编码也得是 GBK。

如果设置页面的属性，则必须写明页面路径。

5. 控件属性值获取指令 ： wget

指令名	功能	参数及举例

wget	取得页面以及控件的属性值	wget name (+回车换行)
		参数:
		页面\控件名称·属性名称。
		举例:
		wget text1.txt
		//读取当前页面文本控件 text1 的 txt 属性
		wget main.text1.txt
		//读取页面 main 中的文本控件 text1 的 txt 属性
		回传: 属性值。参看后面通讯指令返回值介绍。

说明:

页面\控件名称是带有可查找路径的名称，如果要读取页面的属性，则用“wget 页面名称.属性名称”， 如:

wget main.name //获取 main 页面的页名称（读出来是字符串形类型，可赋给文本控件打印显示）

wget page1.numf1.valf //获取 page1 页面的浮点数控件 numf1 的浮点数值

返回的属性值有三种类型，整数，小数，字符串，整数和小数都用十六进制码表示，小端在前；中英文字符串，返回中英文的 ascii 码值或者区位码值，十六进制表示。注意制作串口屏工程时选择的字符编码（工具->项目设置），和发送命令的用户编译环境选择的字符编码一致。

如果获取页面的属性，则必须写明页面路径。

6. 设置系统变量 ： sset

指令名	功能	参数及举例
sset	设置系统变量	sset name value(+回车换行) 参数: 系统变量名称　系统变量值（全部为整数类型）。 举例: 设置系统亮度为 100 。 命令: sset　sys_light 100 回传数据: 执行结果。参看后面通讯指令返回值介绍。 说明: sys_light ，是系统变量–背光亮度的名称，100 是亮度级别。系统变量会在后面页面列举介绍。

7. 设置系统变量 ： sget

指令名	功能	参数及举例
sget	读取系统变量的值	sget name (+回车换行) 参数: 系统变量名称。 举例: 取得系统亮度为 100 。 命令: sget sys_light 回传数据: 系统变量值。参看后面通讯指令返回值介绍。 说明: sys_light ，是系统变量– 背光亮度的名称，亮度级别 1-100。系统变量会在后面页面列举介绍。

8. 事件触发 ：event

指令名	功能	参数及举例
event	事件触发	event name val(+回车换行) 参数: name ： 控件名称，支持 按钮，触摸热区，双态按钮，定时器。 val：触发类型值：1 按下事件，2 弹起事件。 举例: event button0 1 //触发按钮 button0 的按下事件。
说明: 此指令可以触发按钮控件，触摸热区控件，双态按钮控件的按下、弹起的处理脚本。 event timer 1 //触发定时器脚本，会立即执行定时器脚本，定时器的 interval 设置（触发事件的间隔时间）将无效。		

9. 模拟点击 ：click

指令名	功能	参数及举例
click	模拟点击事件	click name (+回车换行) 参数: name ： 控件名称，支持 按钮，触摸热区，双态按钮。 举例: click button0 //模拟按钮 button0 被按下。 发送此指令按下按钮后，就等于模拟做了按下弹起动作，按钮会执行按下和弹起里面的脚本事件。
说明: 此指令可以模拟按下按钮控件，触摸热区控件，双态按钮控件。 方便没有触摸屏时，用户使用物理按键操作串口屏模组。		

10. 触摸校准&恢复出厂 ：adjust & factory

指令名	功能	参数及举例
adjust	触发触屏校准功能	adjust(+回车换行) 参数：无 举例：adjust 注意：进入触屏校准页面后，需手动触屏校准完才退出。 此指令对 VP 编辑软件无效。
		factory(+回车换行) 参数：无 举例：factory 注意：此命令会擦除用户工程。对固件无影响。 如果需要重新下载串口屏工程，请重新上电下载。 此指令对 VP 编辑软件无效。

11. 曲线数据透传 ： addt

指令名	功能	参数及举例
addt	曲线控件数据透传指令 该指令参数使用 16 进制数据，不需要结束符，回车换行 \r\n(0x0D 0x0A)	命令格式： [0~3] 指令名 addt，即 0x61 0x64 0x64 0x74 [4] 页 ID [5] 控件 ID [6] 通道号 0~2，一个曲线控件最多 3 个通道 [7~8] 数据字节字节数 N，小端模式。例如 100 个 char 型数据，十六进制为 0x0064，命令中为 64 00。注意曲线控件的属性 numType，char 类型：1 个字节表示一个点；short 类型：2 个字节表示一个点；int、float 类型：4 个字节表示一个点。 [9~11] 备用。填 00 00 00 即可。 [12~12+N] 数据。可以分为 4 个类型：int（4 字节) short(2 字节） char（1 字节） float（4 字节），在串口屏工程中曲线控件属性中设置，组织传输数据时，要和属性设置一致。

举例：

设置 0 通道 byte 类型 10 个点：61 64 64 74 00 04 00 0A 00 00 00 00 01 02 04 06 08 0A 0C 0E 10 12

byte 类型，一个字节表示一个点

设置 0 通道 int 类型 3 个点：61 64 64 74 00 04 00 0C 00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00

int 类型，4 个字节表示一个点，小端模式，这里三个点分别是 0x00000001,0x00000002,0x00000003

设置 0 通道 float 类型 3 个点：61 64 64 74 00 04 00 0C 00 00 00 00 9a 99 99 3F 9A 99 19 40 66 66 66 40

float 类型，4 个字节表示一个点，小端模式，这里三个点分别是 1.2 ， 2.4 ， 3.6

12. 显示点阵 ：pisx（弃用）

指令名	功能	参数及举例
pisx	显示点阵数据	参数介绍： [0~3]指令名 pisx，即 0x70 0x69 0x78 0x73 [4-5] 起始横坐标 [6-7] 起始竖坐标 [8]单个字模宽度 [9]单个字模高度 [10-12]颜色，小端模式，如红色 0xFF0000，[11]00，[12]00，[13]FF [13-14]字模数据字节数 [15-N]字模数据，1bit 对应一个像素点 取模说明：逐行式，从第一行开始向右每取 8 个点作为一个字节，如果最后不足 8 个点就补满 8 位。取模顺序是从高到低，即第一个点作为最高位

说明：此功能实现了客户想显示一些字库中没有特殊字符或者单色图片，显示时需要用户实时发送，例如 笑脸等用户自定义有个性的图标。

举例：

70 69 78 73 \\命令码 | 00 00 \\起始横坐标 | 0A 00 \\起始竖坐标 | 10 \\字模宽度 16 | 10\\字模高度 16

00 00 FF \\颜色 0xFF0000，红色 |20 00 \\字模字节数 32

08 80 08 80 08 80 11 FE 11 02 32 04 34 20 50 20 91 28 11 24 12 24 12 22 14 22 10 20 10 A0 10 40

\\字模数据”你”，16*16，宋体，不加粗，不斜体

注意：该命令已弃用，建议使用图片或者其他控件，显示特殊字符。

13. 查询版本信息 ：version

指令名	功能	参数及举例
version	查询版本信息	version (+回车换行) 参数：无。 举例： version 查询串口屏硬件相关信息。
说明： 此指令可查询串口屏的型号，flash 大小，固件的时间版本。 在 VP 编辑软件或其它串口助手上发送，会返回上述信息。 此指令需用串口连接串口屏后查询才有返回信息。		

14. 复位屏幕 ：reset

指令名	功能	参数及举例
reset	复位屏幕	reset 1 (+回车换行) 参数：1 举例： reset 1
说明： 此指令可以模拟屏幕断电重启。 通过串口发送指令复位，无需断电也可重启复位串口屏。 此指令对 VP 编辑软件无效。		

15. 通讯指令返回值介绍 1

条件	只有当系统变量 bkcmd 为非 0 的时候才会返回指令执行成功或者失败数据。
格式	返回指令格式为 命令码(1 字节)+参数(长度随命令码而不同)+结束符(2 字节，\r\n)
数据	所有指令名以及参数全部 16 进制数据，非 ASCII 数据，便于软件解析。

16. 通讯指令返回值介绍 2

命令码	含义	格式	参数说明
0x00	指令执行结果	执行结果代码 +返回数据 +结束符（\r\n）	执行结果代码：1 字节 0x00 指令执行成功 0x01 命令码无效（指令第一个字符串无效） 0x02 指令参数 1 无效（指令第二个字符串无效） 0x03 指令参数 2 无效（指令第三个字符串无效） 0x04 指令参数 3 无效（指令第四个字符串无效） 0x05 指令参数 4 无效（指令第五个字符串无效） 0x06 指令参数 5 无效（指令第六个字符串无效） 0x07 指令参数个数错误 0x08 串口接收超时

<table>
<tr><td></td><td colspan="3">说明：
命令执行结果返回值，主要用于客户取得串口屏控件的属性值，响应客户发送的命令，如果客户发送的指令有错，例如命令码拼写错误，属性名称拼写错误，属性名称拼写错误等，造成串口屏不能解析，将返回不同的错误码，方便客户查错。
只有当系统变量 bkcmd 为非 0 的时候才会返回指令执行成功或者失败数据。
这个命令提供了一种客户的调试命令的方式，大部分情况下，在调试完毕后，都会把系统变量 bkcmd 置 0，不会收到此命令。
举例：如果客户发送指令的命令码拼写错误，返回：“00010d0a”。解释：从左到右，00 表示返回的命令码，01 表示命令码无效，0d 表示 \r，0a 表示 \n。</td></tr>
<tr><td>0x01</td><td>整数返回</td><td>0x01+4 字节整数+结束符</td><td>4 字节整数：4 字节小端模式存储，低字节在前</td></tr>
<tr><td>0x02</td><td>小数返回</td><td>0x02+4 字节小数+结束符</td><td>4 字节小数：4 字节小端模式存储，低字节在前</td></tr>
<tr><td>0x03</td><td>字符串返回</td><td>0x03+字符串+结束符</td><td>字符串：GBK 编码</td></tr>
<tr><td></td><td colspan="3">返回结果包括客户发起的命令执行情况和需要返回的数据。执行情况即执行结果，需要返回的数据主要是取串口屏控件属性的值。值的类型为整数，小数，字符串。
举例：获取按钮 1 的 x 坐标。发送指令“wget button1 x\r\n”，返回为：“01880000000d0a”
解释：01 表示返回值为整数；88000000 表示位 buttton1 的 x 坐标，低字节在前的格式；0D0A 是命令结束符。
如果系统变量 bkcmd 没有设置为 0，这个指令会接收到两个返回指令，“00000d0a”，和“01880000000d0a”，第一个指令表示命令执行成功，第二个指令是用户需要的结果。如果系统变量 bkcmd 设置为 0，则只返回第二个指令（默认不返回执行成功或失败）。</td></tr>
</table>

第七章 扩展 IO 口介绍

为适应更多灵活的应用场景，三易串口屏预留了扩展 IO。扩展 IO 的功能有普通输入、普通输出、中断输入、AD 输入、PWM 输出和捕获输入的功能。用户可以使用扩展 IO 功能来控制一些简单的设备。

G 系列串口屏可预留 6 个 IO，S 系列串口屏最多可预留 25 个 IO。

1、G 系列串口屏 IO 功能介绍

G 系列串口屏 2.4 寸、2.8 寸、3.2 寸可以支持扩展 IO 功能。PCB 板上是预留的 6 个焊接孔，如下图所示：

每个 IO 的功能如下表：

端口号	普通输入输出	中断输入	AD 输入	PWM 输出
IO0	√	√		
IO1	√	√	√	√
IO2	√		√	√
IO3	√	√	√	√
IO4	√	√		
IO5	√	√		

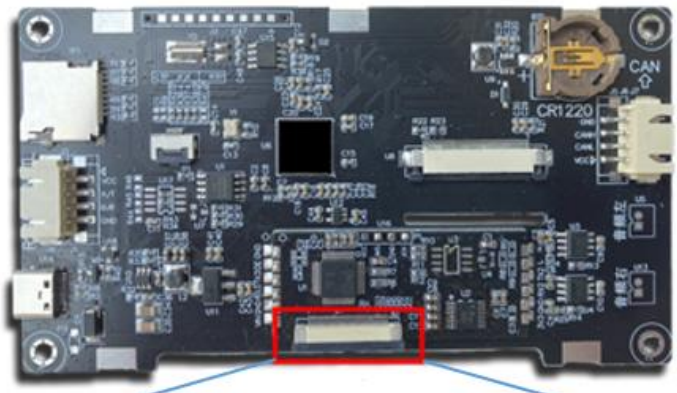
1、所有引脚都可以作为普通输入和输出
2、除开 IO2，其余 IO 都可作为中断输入
3、可同时选择三路 AD 输入
4、可同时输出三路频率一样、占空比不同的 PWM 波形
5、由于 RTC 功能和 IO0、IO1 冲突了。所以如果使用 RTC 功能，IO0 和 IO1 将不允许使用

2、S 系列串口屏 IO 功能介绍

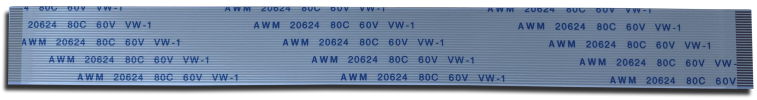
S 系列串口屏所有尺寸 4.3 寸、7 寸、10 寸都可以支持扩展 IO 功能，支持 IO 功能需要串口屏背面搭载 can 模块或者 IO 模块。

2.1、串口屏搭载 IO 模块

需要搭配 30Pin 的软排线使用。



pin1~pin25	pin26、pin27、pin28	pin29、pin30
IO0~IO24	GND	VCC3.3



30Pin 软排线

每个 IO 的功能如下表：

端口号	芯片引脚	输入、输出、中断输入	AD 输入通道	扩展串口	PWM 输出	输入捕获
IO0	PA1	支持	AD_CH1		TIM2_CH2	TIM2_CH2，频率、占空比、脉冲计数
IO1	PA2	支持	AD_CH2	UART_TX	TIM2_CH3	TIM2_CH3，频率、脉冲计数
IO2	PA3	支持	AD_CH3	UART_RX	TIM2_CH4	TIM2_CH4，频率、脉冲计数
IO3	PA6	支持	AD_CH6		TIM3_CH1	TIM3_CH1，频率、占空比、脉冲计数
IO4	PA7	支持	AD_CH7		TIM3_CH2	TIM3_CH2，频率、脉冲计数
IO5	PB0	支持	AD_CH8		TIM3_CH3	TIM3_CH3，频率、脉冲计数
IO6	PB1	支持	AD_CH9		TIM3_CH4	TIM3_CH4，频率、脉冲计数
IO7	PA8	支持				
IO8	PA9	支持			TIM1_CH2	TIM1_CH2，频率、占空比、脉冲计数
IO9	PA10	支持			TIM1_CH3	TIM1_CH3，频率、脉冲计数
IO10	PA0	支持	AD_CH0		TIM2_CH1	TIM2_CH1，频率、脉冲计数

IO11	PA4	支持	AD_CH4			
IO12	PA5	支持	AD_CH5			
IO13	PA11	支持			TIM1_CH4	TIM1_CH4，频率、脉冲计数
IO14	PA12	支持				
IO15	PA15(上电默认高电平)	支持			TIM2_CH1 重映射	TIM2_CH1，重映射，频率、脉冲计数
IO16	PB3 （上电默认高电平）	支持			TIM2_CH2 重映射	TIM2_CH2，重映射，频率、占空比、脉冲计数
IO17	PB4	支持			TIM3_CH1 重映射	TIM3_CH1，重映射，频率、占空比、脉冲计数
IO18	PB5	支持			TIM3_CH2 重映射	TIM3_CH2，重映射，频率、脉冲计数
IO19	PB8	支持			TIM4_CH3	TIM4_CH3，频率、占空比、脉冲计数
IO20	PB9	支持			TIM4_CH4	TIM4_CH4，频率、脉冲计数
IO21	PB13	支持				
IO22	PB14	支持				
IO23	PB2	支持				
IO24	PB15	支持				

IO 引脚 0~24，来源于 STM32F103 芯片，引脚的详细介绍可参考 STM32 相关文档。

1、所有引脚都可以作为普通输入和输出。作为中断输入的时候需要注意，引脚序号相同的 io 口，不能同时作为中断输入。如已经选择了 IO0(PA1)作为中断输入，那么 IO6(PB1)不能再作为中断输入。

2、一共 10 个通道的 AD 可同时选择。

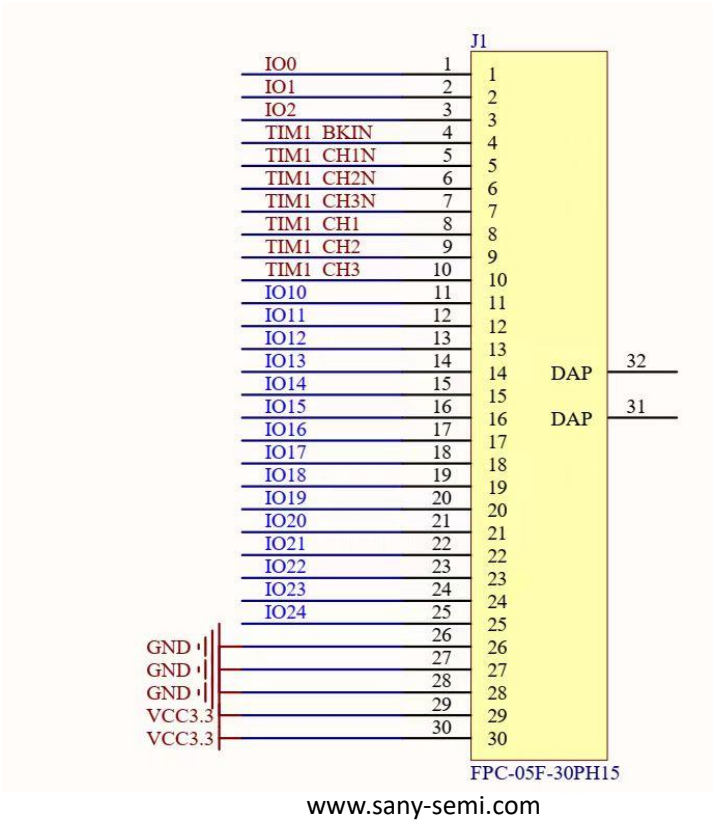
3、一个扩展串口，固定引脚在 IO1 和 IO2

4、有 4 个内部定时器，每个定时器可单独设置周期和占空比，同一个定时器下的不同通道，频率只能设置同一个值，能设置不同的占空比。如 IO0 和 IO1 输出的 PWM 频率一定是一样的。

5、输入捕获功能可以获取频率、占空比(部分 io)、脉冲计数。与 PWM 功能共用内部的 4 个定时器，同一个定时器不能同时作为 PWM 功能和输入捕获，如 TIM2_CH2 通道作为了 PWM 输出，那么 TIM2_CH3 和 TIM2_CH4 通道不能作为输入捕获；同一个定时器只有一个固定的通道有获取占空比的功能。

6、IO15 和 IO16 是 TIM2_CH1 和 TIM2_CH2 的重映射；IO17 和 IO18 是 TIM3_CH1 和 TIM3_CH2 的重映射。如选择了 IO15(TIM2_CH1)作为 PWM 输出，那么 IO10(TIM2_CH1)和 IO0(TIM2_CH2)将不能作为 PWM 功能，此时 TIM2_CH1 和 TIM2_CH2 重映射到了 IO15 和 IO16。

7，IO25-29 为 GND 和 VCC 引脚



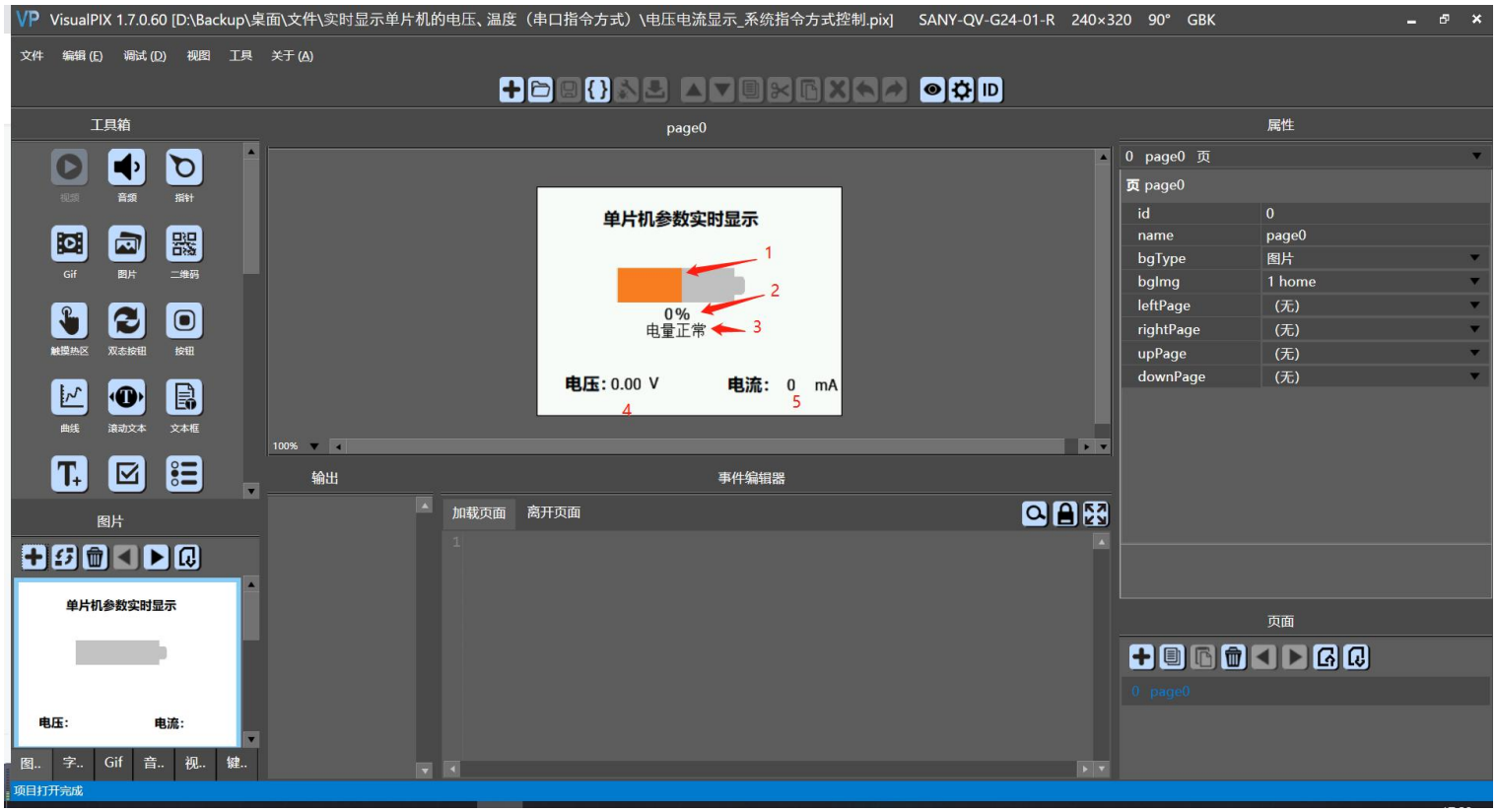
第八章 典型应用

实验 1 实时显示单片机的参数（整数、小数、中文 系统指令方式）

1.实验目的

单片机按照串口屏的系统指令格式发送控制命令，串口屏实时显示单片机的参数，包括整数、小数、和中文字符。

2.页面设计



(1)进度条控件，以进度条的方式显示出 SOC 值
(2)整数控件，显示具体 SOC 值
(3)文本控件，显示电量状态
(4)浮点数控件，显示电压值
(5)整数控件，显示电流值
选中控件，可以在右侧属性栏查看控件的各个属性。详细属性说明可参考 第四章 控件的介绍和用法。

资源区：
一共使用 4 张图片，home 作为页背景图片，另外三张图片为进度条不同状态的颜色。
字体字体除了工程自带的字库 Tahoma_4x_ASCII（包括英文和数字），另外创建了一个中文的字库，双击这个字库，可以看到详细的信息。字库详细制作方法可以参考第三章的添加字库小节。

3.串口屏协议处理

由于该例程使用串口屏自带的命令(详细命令 参考第六章 串口指令的简介和用法)，无需设计其他通信协议。单片机按照指令格式即可控制显示屏。

4.下载验证

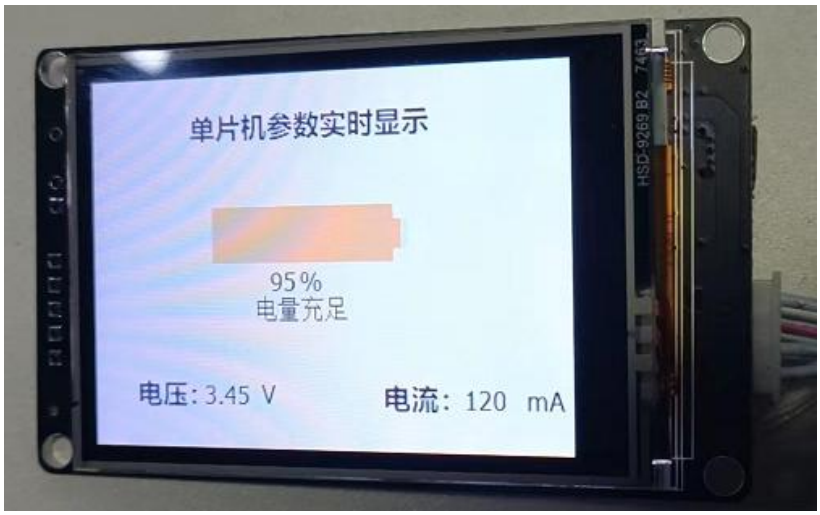
编译成功后，点击下载按钮，选择正确的端口号和波特率，下载到串口屏。可以看到显示如下显示：

测试时，可以先通过电脑串口助手给屏发指令来调试。

注意：串口屏系统命令是以\r\n 结尾的，所以发送数据时需要勾选”发送新行”，或者输入是手动敲入换行

```
wset numVf.valf 3.45
wset numA.val 120
wset psoc.val 95
wset nsoc.val 95
wset text.txt "电量充足"
```

串口屏显示如下，表示串口屏程序设计没问题。



实际应用时，是 MCU 和串口屏通信。本实验使用的单片机型号是 STM32F103TB,改例程仅作供参考，开发环境 keil V5.24。其他 MCU 的串口发送方式也类似。

部分代码

```
int main(void)
{
    Sys_SetRcc();                                    //设置系统使用内部时钟
    delay_init(64);                                  //延时函数初始化
    usart1_init();                                   //初始化串口 波特率 115200

    while(1)
    {
        //设置电压值
        sprintf(page_str,"wset numVf.valf %.2f\r\n",V);//①
        USART_OUT(USART1, (uint8_t*)page_str,strlen(page_str)); //②
        delay_ms(1); //③命令之间需要加上小延时分开

        //设置电流值
        sprintf(page_str,"wset numA.val %d\r\n",I);
        USART_OUT(USART1, (uint8_t*)page_str,strlen(page_str));
        delay_ms(1); //命令之间需要加上小延时分开

        //设置进度条 SOC
        sprintf(page_str,"wset psoc.val %d\r\n",SOC);
        USART_OUT(USART1, (uint8_t*)page_str,strlen(page_str));
        delay_ms(1); //命令之间需要加上小延时分开

        //设置 SOC 百分比
        sprintf(page_str,"wset nsoc.val %d\r\n",SOC);
        USART_OUT(USART1, (uint8_t*)page_str,strlen(page_str));
        delay_ms(1); //命令之间需要加上小延时分开



        if(SOC > 80)
```

```
        {
            sprintf(page_str,"wset psoc.fgImg 2\r\n");//④绿色图片
            USART_OUT(USART1, (uint8_t*)page_str,strlen(page_str));
            delay_ms(1); //命令之间需要加上小延时分开
            sprintf(page_str,"wset text.txt \"电量充足\"\r\n"); //⑤
            USART_OUT(USART1, (uint8_t*)page_str,strlen(page_str));
            delay_ms(1); //命令之间需要加上小延时分开
        }
        else if(SOC > 20)
        {
            sprintf(page_str,"wset psoc.fgImg 4\r\n");//黄色图片
            USART_OUT(USART1, (uint8_t*)page_str,strlen(page_str));
            delay_ms(1); //命令之间需要加上小延时分开

            sprintf(page_str,"wset text.txt \"电量正常\"\r\n");
            USART_OUT(USART1, (uint8_t*)page_str,strlen(page_str));
            delay_ms(1); //命令之间需要加上小延时分开
        }
        else
        {
            sprintf(page_str,"wset psoc.fgImg 3\r\n");//红色图片
            USART_OUT(USART1, (uint8_t*)page_str,strlen(page_str));
            delay_ms(1); //命令之间需要加上小延时分开

            sprintf(page_str,"wset text.txt \"电量过低\"\r\n");
            USART_OUT(USART1, (uint8_t*)page_str,strlen(page_str));
            delay_ms(1); //命令之间需要加上小延时分开
        }
        //模拟数据变化
        V += 0.08;
        if(V > 5) V = 3.0;
        if(++I > 100)I = 0;
        if(++SOC > 100)SOC = 0;

        delay_ms(200);
    }
}
```

①sprintf 处理字符串，浮点数控件的赋值方式
②串口发送函数，以自己单片机工程里面的发送函数为准
③命令之间需要加上小延时分开，保证串口屏收到的命令之间没有粘连
④设置图片，这里的 2 表示图片的编号，在 VP 软件的图片资源里可以看到图片的具体编号
⑤字符串的发送，在 C 语言里双引号需要转义，这里发送的是 wset text.txt “电量充足”\r\n 。发送中文时，注意 VP 工程里面的编码方式和开发环境代码的编码方式要一致，例程里面是 GBK 编码。否则汉字显示可能会有问题。

查看 VP 软件里的编码：项目设置  ，里面的字符编码，默认使用 GBK 编码

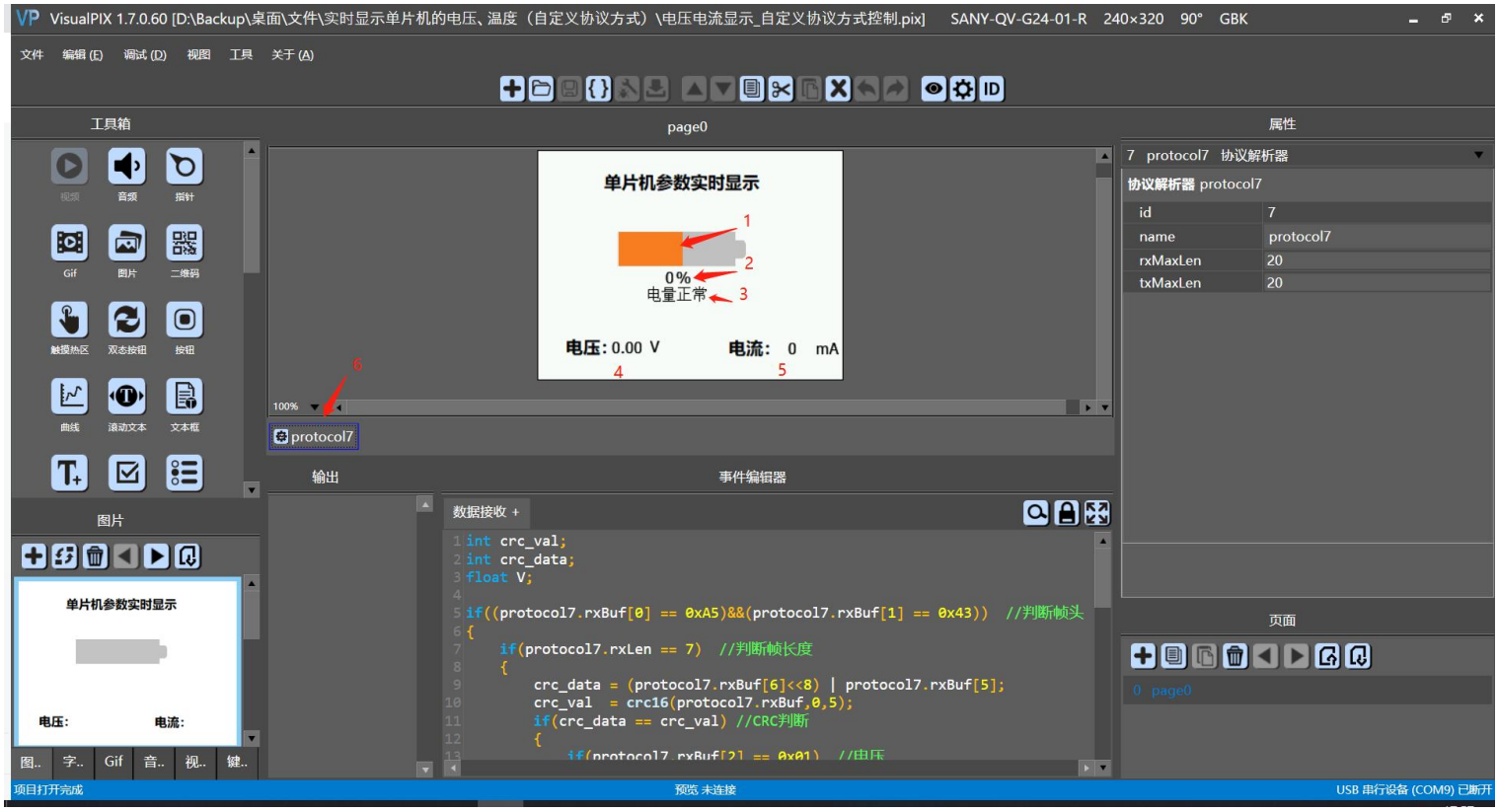
查看 keil 工程的编码：Eidt->Configruation->Eidtor,Encoding 选项，keil 的 GB2312 对应 VP 软件里面的 GBK；keil 的 Encode in ANSI 对应 VP 软件里面的 UTF-8
若使用其他的开发环境，类似的设置。注意编码方式一致即可。

实验 2　实时显示单片机的参数（整数、小数、中文　自定义协议方式）

1.实验目的

单片机按照自定义协议格式发送控制命令，串口屏实时显示单片机的参数，包括整数、小数、和中文字符。实现效果和实验一相同。

2.页面设计



(1)进度条控件，以进度条的方式显示出 SOC 值
(2)整数控件，显示具体 SOC 值
(3)文本控件，显示电量状态
(4)浮点数控件，显示电压值
(5)整数控件，显示电流值
(6)协议解析器控件，编写脚本代码，解析自定义的协议
选中控件，可以在右侧属性栏查看控件的各个属性。详细属性说明可参考 第四章 控件的介绍和用法。

资源区：
一共使用 4 张图片，home 作为页背景图片，另外三张图片为进度条不同状态的颜色。
字体字体除了工程自带的字库 Tahoma_4x_ASCII（包括英文和数字），另外创建了一个中文的字库，双击这个字库，可以看到详细的信息。字库详细制作方法可以参考第三章的添加字库小节。

3.串口屏协议处理

该例程使用了协议解析器控件protocol7。串口屏每次收到一帧数据，都会触发一次协议解析器的脚本，和单片机的空闲中断函数类似。

协议定义（仅仅是示例，仅供参考）
帧头(2 字节) 命令码(1 字节) 数据(2 字节) 校验(2 字节)

A5 43	01	2byte	2byte	//设置电压 使用时除 100，如 3.14V，传输时是 314
A5 43	02	2byte	2byte	//设置电流
A5 43	03	2byte	2byte	//设置 SOC

数据和校验 使用小端模式，即低字节在前，高字节在后
校验方式为 CRC16，多项式 8005

选中控件，在事件编辑器串口可以看到数据的解析过程，代码如下：

```
int crc_val;
int crc_data;
float V;

if((protocol7.rxBuf[0] == 0xA5)&&(protocol7.rxBuf[1] == 0x43))  //①判断帧头
{
  if(protocol7.rxLen == 7)  //②判断帧长度
  {
    crc_data = (protocol7.rxBuf[6]<<8) | protocol7.rxBuf[5];
    crc_val  = crc16(protocol7.rxBuf,0,5);//③
    if(crc_data == crc_val) //CRC 判断
    {
      if(protocol7.rxBuf[2] == 0x01)  //电压
      {
        V = (protocol7.rxBuf[4]<<8) | protocol7.rxBuf[3];
        numVf.valf = V/100;
      }
      else if(protocol7.rxBuf[2] == 0x02)  //电流
      {
        numA.val = (protocol7.rxBuf[4]<<8) | protocol7.rxBuf[3];
      }
      else if(protocol7.rxBuf[2] == 0x03)  //SOC
      {
        psoc.val = (protocol7.rxBuf[4]<<8) | protocol7.rxBuf[3];
        nsoc.val = psoc.val;

        if(nsoc.val > 80)
        {
          psoc.fgImg = user.image.blue;  ④
          text.txt = "电量充足";
        }
        else if(nsoc.val > 20)
```

```
        {
            psoc.fgImg = user.image.yellow;
            text.txt = "电量正常";
        }
        else
        {
            psoc.fgImg = user.image.red;
            text.txt = "电量过低";
        }
      }
    }
  }
}
```

①接收到串口数据缓存在协议解析器控件的 rxBuf 里，是一个数组，数组长度是协议解析器的一个属性，可配置。决定接收一帧数据的最大长度，最大可设置 512 字节。

② 属性 rxLen 是本次接收到数据长度。

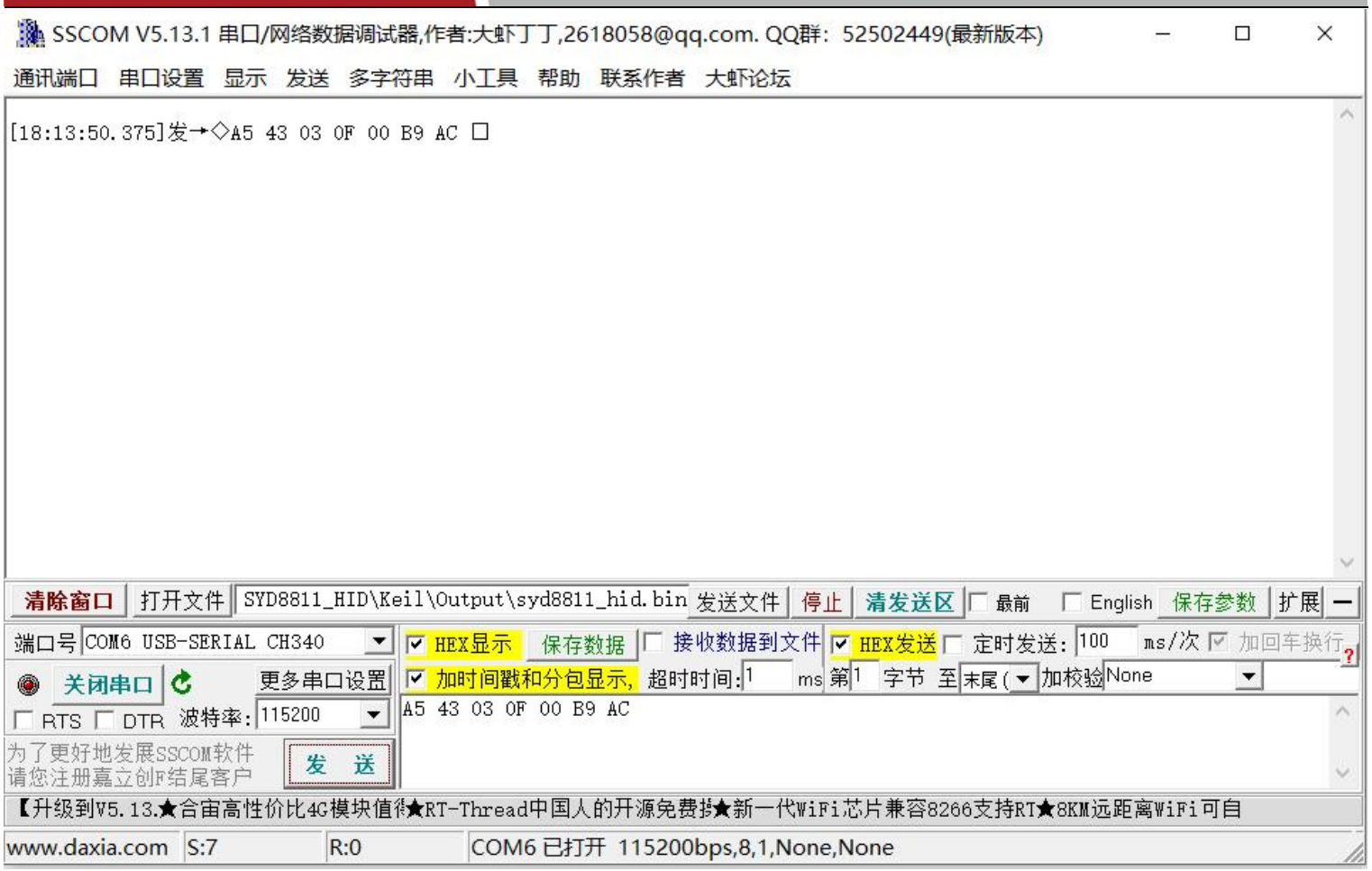
③ crc16 是一个内部函数，用于计算 CRC 校验值。具体使用方法 请参考 第五章 函数的介绍和用法。具体项目中，可选择使用或者不使用校验。

④ psoc.fgImg = user.image.blue;这里是修改进度条的图片，也可以写成这样 psoc.fgImg = 2;效果是一样的。

4.下载验证

编译成功后，点击下载按钮，选择正确的端口号和波特率，下载到串口屏。可以看到显示如下显示：

测试时，可以先通过电脑串口助手给屏发指令来调试。



注意：发送时勾选 hex 发送

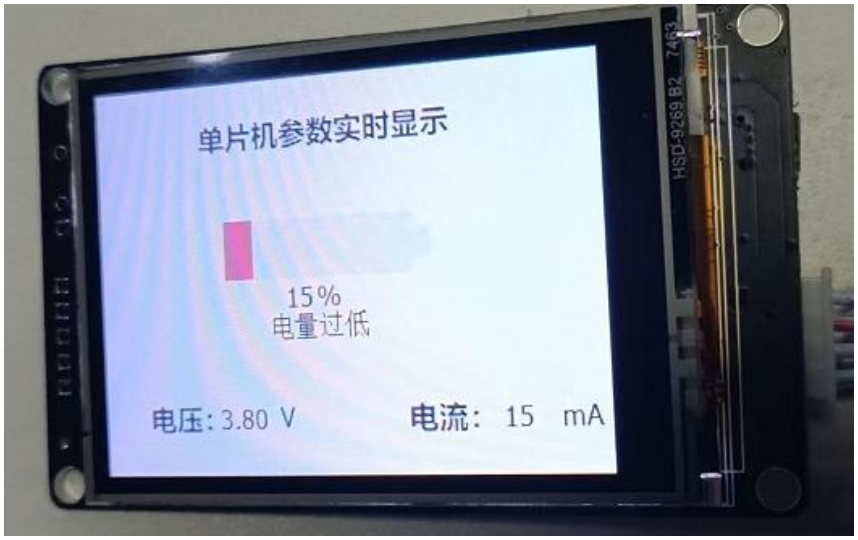
测试的命令

A5 43 01 7C 01 91 86 //设置电压值 3.80V

A5 43 02 0F 00 AE 2C//设置电流值 15mA

A5 43 03 0F 00 B9 AC//设置 SOC 值 15%

串口屏显示如下，表示串口屏收到数据并正确处理。



实际应用时，是 MCU 和串口屏通信。本实验使用的单片机型号是 STM32F103TB,改例程仅作供参考，开发环境 keil V5.24。其他 MCU 的串口发送方式也类似。

部分代码

```
int main(void)
{
    u16 crc_val;


    Sys_SetRcc();                                    //设置系统使用内部时钟
    delay_init(64);                                 //延时函数初始化
    usart1_init();                                   //初始化串口 波特率 115200

    while(1)
    {
        //设置电压值
        str[0] = 0xA5;
        str[1] = 0x43;
        str[2] = 0x01;
        str[3] = (V>>0) & 0xFF;
        str[4] = (V>>8) & 0xFF;
        crc_val = Get_CRC16(5,str);
        str[5] = (crc_val>>0) & 0xFF;
        str[6] = (crc_val>>8) & 0xFF;

        USART_OUT(USART1, (uint8_t*)str,7);
        delay_ms(1); //命令之间需要加上小延时分开



        //设置电流值
        str[0] = 0xA5;
        str[1] = 0x43;
        str[2] = 0x02;
        str[3] = (I>>0) & 0xFF;
        str[4] = (I>>8) & 0xFF;
        crc_val = Get_CRC16(5,str);
        str[5] = (crc_val>>0) & 0xFF;
        str[6] = (crc_val>>8) & 0xFF;


        USART_OUT(USART1, (uint8_t*)str,7);
        delay_ms(1); //命令之间需要加上小延时分开

        //设置 SOC
        str[0] = 0xA5;
        str[1] = 0x43;
        str[2] = 0x03;
        str[3] = (SOC>>0) & 0xFF;
        str[4] = (SOC>>8) & 0xFF;
        crc_val = Get_CRC16(5,str);
        str[5] = (crc_val>>0) & 0xFF;
```

```
        str[6] = (crc_val>>8) & 0xFF;

        USART_OUT(USART1, (uint8_t*)str,7);
        delay_ms(1); //命令之间需要加上小延时分开

        //模拟数据变化
        V+=8;
        if(V > 500)V = 300; //电压值范围 3~5V
        if(++I > 100)I = 0;
        if(++SOC > 100)SOC = 0;

        delay_ms(200);

    }

}
```

单片机部分代码很简单，按照协议格式组帧发送即可。注意每条命令之间需要加一点点延时用于串口屏区分。

总结：

实验二和实验一，最后的显示结果都是一样的。系统指令的方式适合界面不复杂的情况，无需设计通信协议，按照串口屏内置的系统指令格式来发送即可；自定义协议方式体现为更加灵活，C 语言编写脚本，通信效率更高，一帧数据可给多个控件赋值，适用于大多数项目。推荐使用自定义协议方式。两种方式混合使用也可以。

实验 3　MODBUS 主站实验

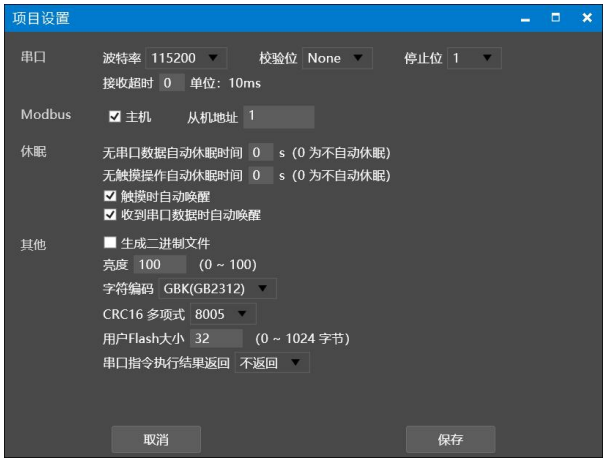
1.实验目的

串口屏作为 Modbus 主站，读取和设置温湿度模块的参数。

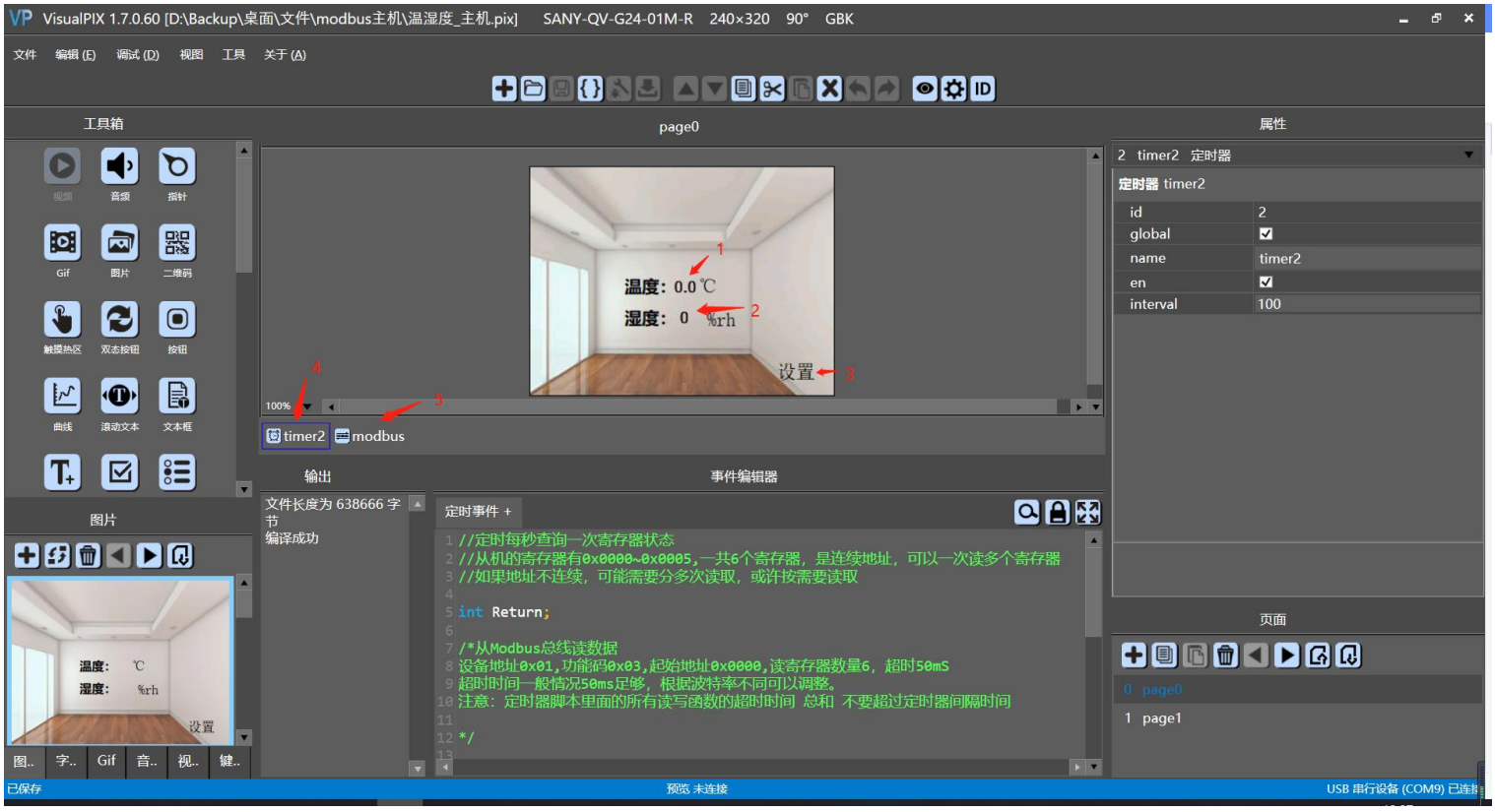
说明:型号名称中带有字母 M 的，表示支持 modbus 协议。

2.页面设计

项目设置里面，勾选 Modbus 主机，表示显示屏作为 modbus 主站。

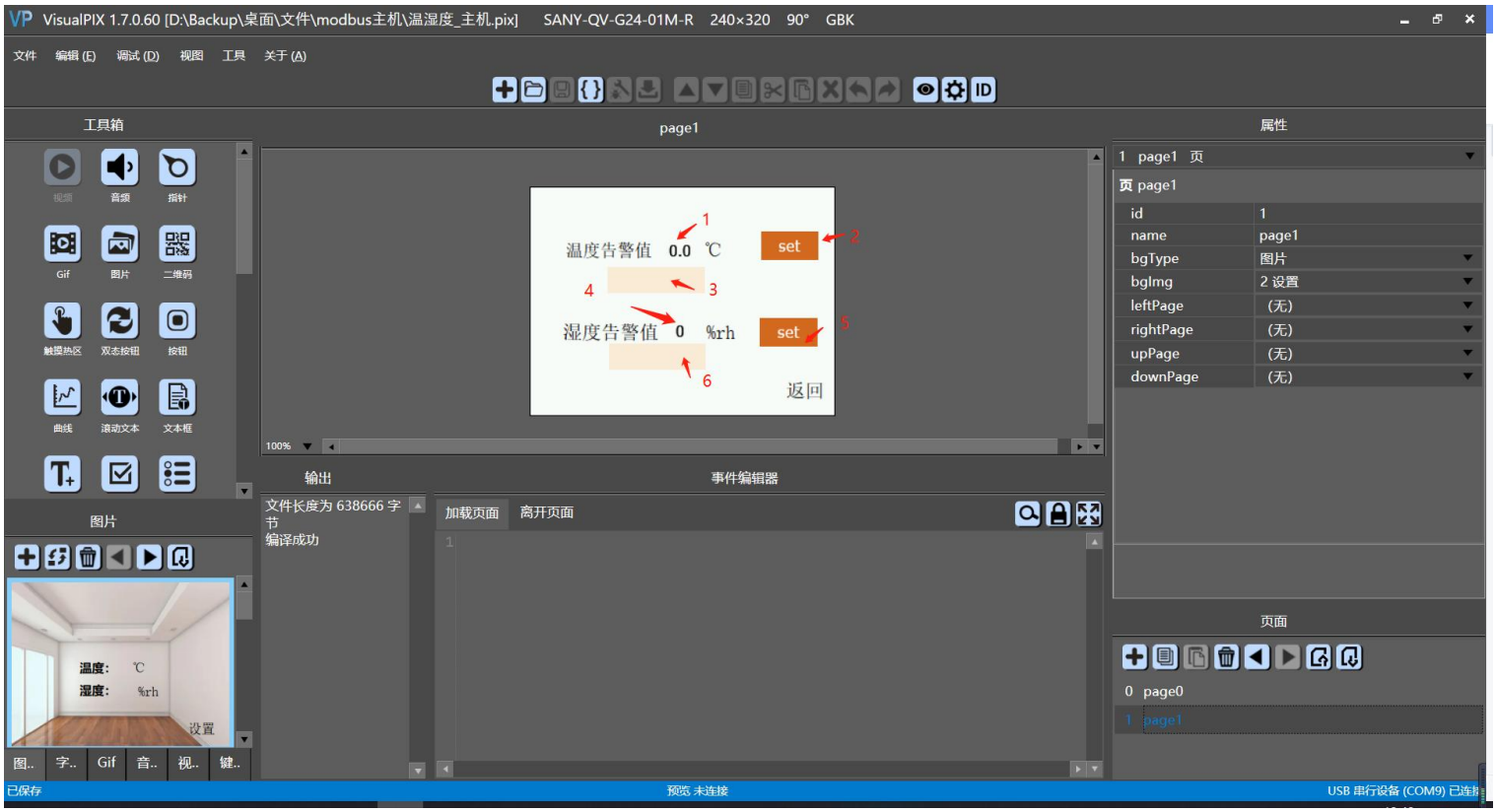


Page0 页面控件



(1)浮点数控件，显示温度值
(2)整数控件，显示湿度值
(3)按钮控件，点击按钮切到第二页
(4)定时控件，定时每秒发送读取帧，获取从机寄存器值，并显示。选中可以查看事件脚本代码。
(5)modbus 控件

page1 页面控件

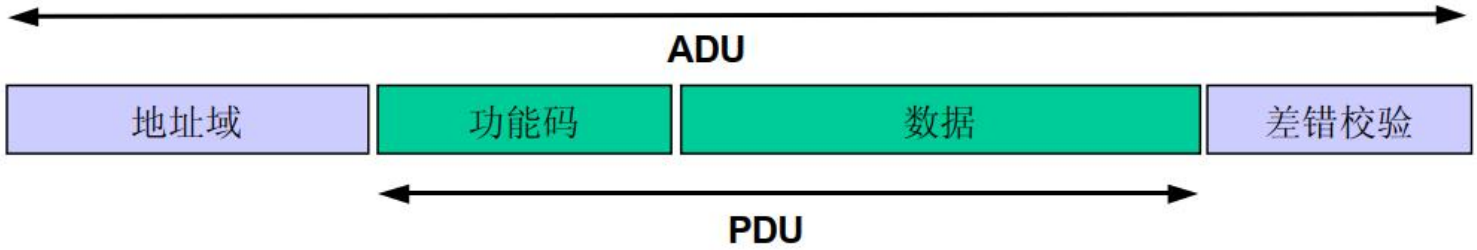


(1)浮点数控件，显示温度告警值
(2)按钮控件，调用 modbus 写函数，设置温度告警值
(3)文本控件，显示温度告警设置是否成功
(4)整数控件，显示湿度告警值
(5)按钮控件，调用 modbus 写函数，设置湿度告警值
(6)文本控件，显示湿度告警设置是否成功

3.串口屏协议处理

modbus 协议简单介绍
注：modbus 介绍取自于《MODBUS 协议中文版》部分，详细介绍请查阅 modbus 相关资料。

MODBUS 协议定义了一个与基础通信层无关的 P 简单协议数据单元（DU）。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



modbus 有主站和从站的区分，一个完整的总线上有一个主站和若干从站，每个从站有自己独特的地址，区分哪个是从站设备。

地址域：即从站地址，1 字节，用一个字节编码 MODBUS 数据单元的功能码域。有效的码字范围是十进制 1-255（128-255 为异常响应保留）。

功能码：1 字节，当从客户机向服务器设备发送报文时，功能码通知服务器执行哪种操作。

数据：根据不同功能码，有着不同的数据格式和长度

差错校验：循环冗余校验 (CRC)

有两种串行传输模式被定义: RTU 模式 和 ASCII 模式。

它定义了报文域的位内容在线路上串行的传送。它确定了信息如何打包为报文和解码。

Modbus 串行链路上所有设备的传输模式 (和串行口参数) 必须相同。

尽管在特定的领域 ASCII 模式是要求的，但达到 Modbus 设备之间的互操作性只有每个设备都有

相同的模式： 所有设备必须必须实现 RTU 模式。 ASCII 传输模式是选项。

设备应该由用户设成期望的模式， RTU 或 ASCII。 默认设置必须为 RTU 模式。

RTU 传输模式是常用的方式，报文帧格式如下：

子节点地址	功能代码	数据	CRC
1 字节	1 字节	0 到 252 字节	2 字节 CRC 低 | CRC 高

Modbus RTU 帧最大为 256 字节。

本实验，显示屏作为主站，通过 modbus 协议标准获取温度模块的参数。

温度模块 modbus 参数如下：

设备地址	寄存器地址			
1	0x0000	温度 H	只读	float 类型，两个寄存器表示温度，单位 0.1℃
	0x0001	温度 L		
	0x0002	温度报警值 H	读写	float 类型，两个寄存器表示温度，单位 0.1℃
	0x0003	温度报警值 L		
	0x0004	湿度	只读	16 位整数类型，单位%rh
	0x0005	湿度报警值	读写	
读功能码 0x03,写功能码 0x10				

modbus 为控件控件名称(也可以改为其他名字)，与 modbus 协议帧对应的控件属性如下：

从站地址	功能码	数据	校验
modbus.addr	Modbus.fcode	modbus.rxBuf	

定时器控件用于定时查询寄存器值，脚本代码如下：

```
//定时每秒查询一次寄存器状态
//从机的寄存器有 0x0000~0x0005,一共 6 个寄存器，是连续地址，可以一次读多个寄存器
//如果地址不连续，可能需要分多次读取，或许按需要读取
/**********************本实验寄存器列表**********************************
  0x0000  温度 H  只读  float 类型，两个寄存器表示温度，单位 0.1℃
  0x0001  温度 L
  0x0002  温度报警值 H  读写  float 类型，两个寄存器表示温度，单位 0.1℃
```

```
  0x0003   温度报警值 L
  0x0004   湿度   只读   16 位整数类型，单位%rh
  0x0005   湿度报警值   读写
**************************************************************************/
int Return;


/*********modbus 帧与控件属性的对应关系***********

从站地址       功能码            数据             校验
modbus.addr    Modbus.fcode   modbus.rxBuf
*****************************************/


/*从 Modbus 总线读数据
设备地址 0x01,功能码 0x03,起始地址 0x0000,读寄存器数量 6，超时 50mS
超时时间一般情况 50ms 足够，根据波特率不同可以调整。
注意：定时器脚本里面的所有读写函数的超时时间 总和 不要超过定时器间隔时间
*/

//读寄存器 参数（从机地址 功能码 寄存器起始地址 寄存器数量 超时时间）
Return = modbusRead(0x01,0x03,0x0000,6,50);
if(Return == 0) //一定要检测返回值，返回值为 0，表示读取成功
{
  //获取温度值
  temp.valf = bytesToFloat(modbus.rxBuf,1,1); //将 4 个字节转换成一个浮点数。modbus.rxBuf[1~4]
  //读寄存器值时，返回数据缓存在 modbus 控件的 rxBuf 属性中，其中 modbus.rxBuf[0]为属性 modbus.rxLen 的值，
  //所以数据从 modbus.rxBuf[1]开始取

    //获取温度告警值
    page1.temp_alarm.valf = bytesToFloat(modbus.rxBuf,5,1);//将 4 个字节转换成一个浮点数。modbus.rxBuf[5~8]

    //获取湿度值
    hum.val = (modbus.rxBuf[9]<<8) | modbus.rxBuf[10];

    //获取湿度告警值
    page1.hum_alarm.val = (modbus.rxBuf[11]<<8) | modbus.rxBuf[12];
  }
```

Page1 的设置按钮，弹起事件主要是寄存器写操作，脚本代码如下

```
int res;
byte a[4];


floatToBytes(temp_alarm.valf,a,0,1);
//写操作函数：参数(从机地址 功能码 寄存器地址 寄存器数量 数组 数组长度 超时时间)
res = modbusWrites (0x01, 0x10, 0x0002, 2, a, 4, 50);
```

```
if(res == 0)
{
   text5.txt = "Set OK";
}
else
{
   text5.txt = "Set Fail";
}
page0.timer2.en = 1;
```

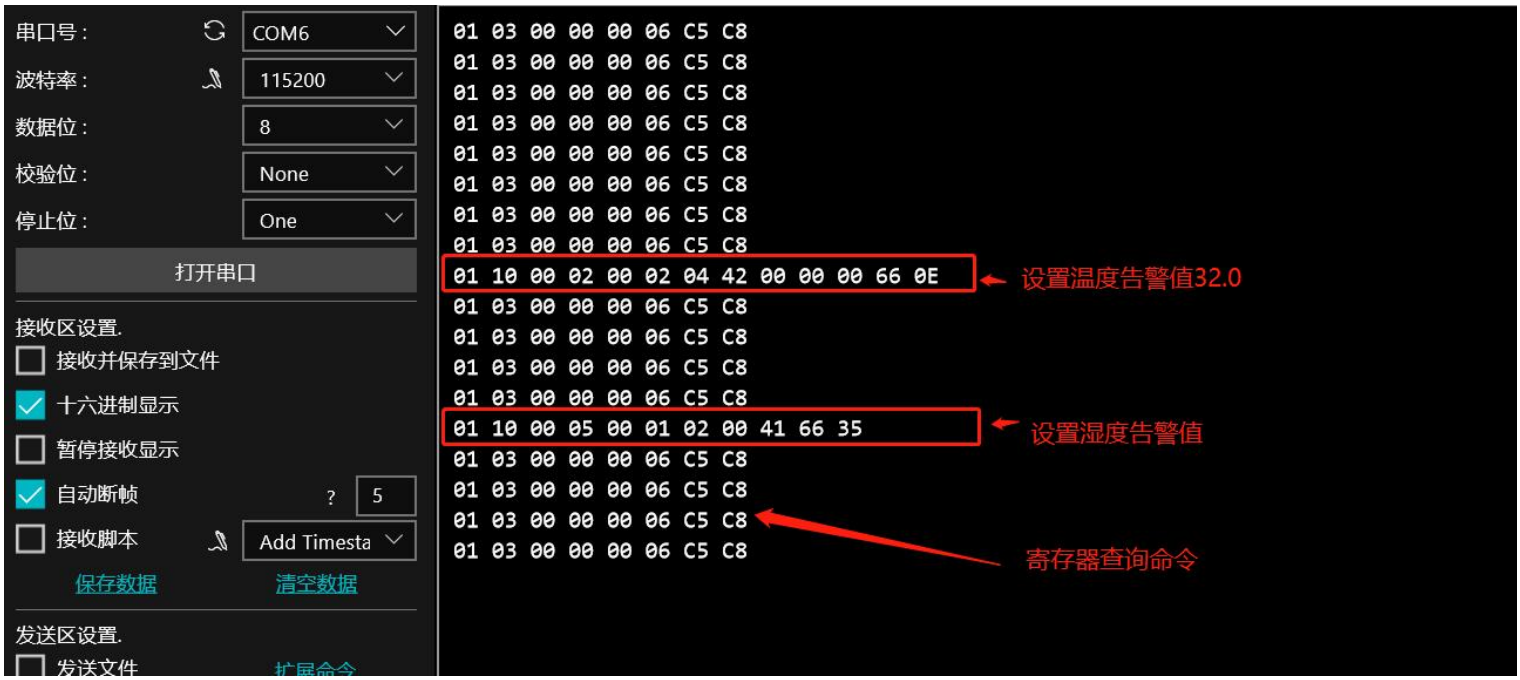
解析的主要过程，是对 modbus.rxBuf 字节数组的处理。上面读操作，使用了移位+按位或操作得到一个无符号的整数并赋值为控件，用于显示；还使用字节转浮点数的函数 bytesToFloat，将 4 个字节转换成一个浮点数。具体函数相关的介绍，可以参考第五章函数的介绍。

4.下载验证

编译成功后，点击下载按钮，选择正确的端口号和波特率，下载到串口屏。可以看到显示如下显示：

用 VP 软件将例程下载到串口屏，连接温湿度模块，即可看到温湿度模块的参数。

调试时，将显示屏的串口数据发送到电脑的串口助手上，查看读命令、写命令是否正确：


串口号：
COM6
波特率：
115200
数据位：
8
校验位：
None
停止位：
One
打开串口
接收区设置.
接收并保存到文件
十六进制显示
暂停接收显示
自动断帧
5
接收脚本
Add Timesta
保存数据
清空数据
发送区设置.
发送文件
01 03 00 00 00 06 C5 C8
01 03 00 00 00 06 C5 C8
01 03 00 00 00 06 C5 C8
01 03 00 00 00 06 C5 C8
01 03 00 00 00 06 C5 C8
01 03 00 00 00 06 C5 C8
01 03 00 00 00 06 C5 C8
01 03 00 00 00 06 C5 C8
01 10 00 02 00 02 04 42 00 00 00 66 0E
设置温度告警值32.0
01 03 00 00 00 06 C5 C8
01 03 00 00 00 06 C5 C8
01 03 00 00 00 06 C5 C8
01 03 00 00 00 06 C5 C8
01 10 00 05 00 01 02 00 41 66 35
设置湿度告警值
01 03 00 00 00 06 C5 C8
01 03 00 00 00 06 C5 C8
01 03 00 00 00 06 C5 C8
寄存器查询命令
01 03 00 00 00 06 C5 C8

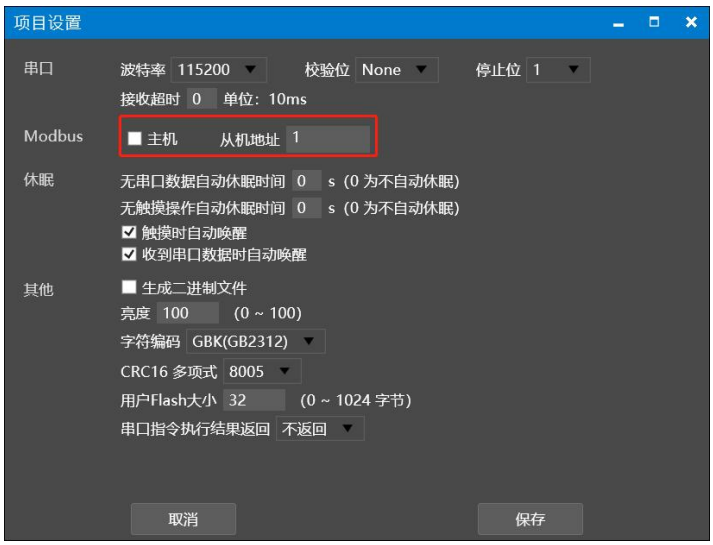
实验 4　MODBUS 从站实验

1.实验目的

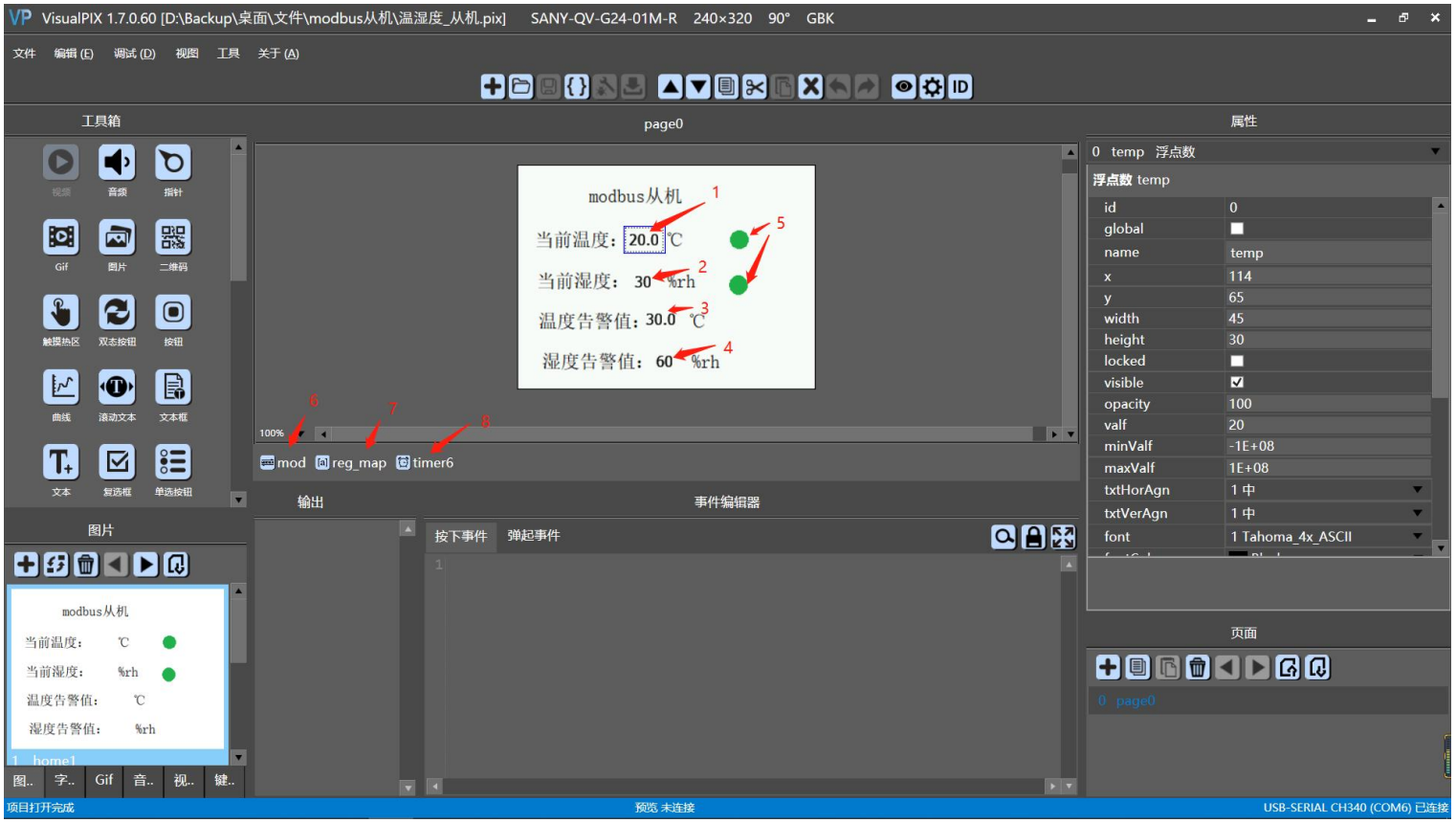
本实验模拟实验四 用到的温湿度模块，显示屏作为 modbus 从站。

2.页面设计

项目设置里面，modbus 不勾选主机，填写从机(从站)地址，默认为 1.



page0 界面



(1)浮点数控件，用于显示温度值，一位小数点
(2)整数控件，显示湿度
(3)浮点数控件，显示温度告警值
(4)整数控件，显示湿度告警值
(5)图片控件，用于显示告警状态，绿色正常，红色异常
(6)Modbus 控件，协议处理主要在控件的事件编辑器里
(7)数组控件，用于存放寄存器值的表
(8)定时器，定时 1 秒模拟温湿度的变化，并判断是否告警

3.串口屏协议处理

寄存器列表

设备地址	寄存器地址			
1	0x0000	温度 H	只读	float 类型，两个寄存器表示温度，单位 0.1℃
	0x0001	温度 L		
	0x0002	温度报警值 H	读写	float 类型，两个寄存器表示温度，单位 0.1℃
	0x0003	温度报警值 L		
	0x0004	湿度	只读	16 位整数类型，单位%rh
	0x0005	湿度报警值	读写	
读功能码 0x03,写功能码 0x10				

modbus 为控件控件名称(也可以改为其他名字)，与 modbus 协议帧对应的控件属性如下：

从站地址	功能码	数据	校验
modbus.addr	modbus.fcode	modbus.rxBuf	

串口屏作为从站时，当收到一个主站的读/写命令后，显示屏自己判断地址和校验是否匹配，若无误则会触发一个 modbus 控件的事件，客户需要做的是完善这个 modbus 控件的事件。modbus 的相关协议处理，在 modbus 控件数据接收里面，脚本代码如下：

```
/*
modbus 从机接收处理  脚本
响应主机的读命令  和  写命令，并发送响应命令帧
从机地址  可在  项目设置里配置
*/

/*************modbus 协议帧对应的控件属性******
从站地址      功能码        数据           校验
modbus.addr   Modbus.fcode   modbus.rxBuf
*******************************************/

int reg=0; //寄存器地址
int index;
int num; //寄存器个数
int Byte_num;
byte buf[16];

int j;
```

```
int i;

if(mod.fcode == 0x03)  //功能码为 0x03,表示主站来读寄存器
{
  reg = (mod.rxBuf[0]<<8) | mod.rxBuf[1];  //取出要读的寄存器地址
  num = (mod.rxBuf[2]<<8) | mod.rxBuf[3];  //取出要读的寄存器个数
  if((reg <= 0x0005)&&(num <= 6))   //检测参数范围是否有效

  index = reg*2;
  {
    buf[0] = (byte)(num*2);              //字节数
    j = 1;
    for(i=0;i<num;i++)              //根据要读的寄存器个数，取出寄存器值，放到 buf 里面
    {
      buf[j++] = reg_map.valbs[index++];
      buf[j++] = reg_map.valbs[index++];
    }
    modbusSendBytes(mod.addr, mod.fcode, buf, 1+num*2, 0); //发送一帧 modbus 响应数据，从机发送无需无超时
  }
}
else if(mod.fcode == 0x10)  //功能码为 0x10,表示主站写寄存器
{
  reg = (mod.rxBuf[0]<<8) | mod.rxBuf[1];  //取出要写的寄存器地址(起始地址)
  num = (mod.rxBuf[2]<<8) | mod.rxBuf[3];  //取出要写的寄存器数量
  Byte_num = mod.rxBuf[4]; //取出需要写的字节数

  if(num*2 < Byte_num)   //处理寄存器数量和字节数不相等的情况
    num = Byte_num / 2;

  if((reg <= 0x0005)&&(num <= 6))   //检测参数范围是否有效
  {
    index = reg*2;
    for(i=0;i<num;i++)
    {
      reg_map.valbs[index]   = mod.rxBuf[j+5];
      reg_map.valbs[index+1] = mod.rxBuf[j+6];
      index+=2;
      j+=2;
    }


    for(i=0;i<num;i++)
    {
      if(reg == 0) //温度 H
      {
        //只读
      }
      else if(reg == 1) //温度 L
      {
```

```
        //只读
      }
      else if(reg == 2) //温度告警值 H
      {
        //放到 reg == 3 的时候，解析数据
      }
      else if(reg == 3) //温度告警值 L
      {
        temp_alarm.valf = bytesToFloat(reg_map.valbs,(reg-1)*2,1); //将 4 字节转换成 float 类型，并赋值给控件
      }
      else if(reg == 4) //湿度
      {
        //只读
      }
      else if(reg == 5) //湿度告警值
      {
        hum_alarm.val = (reg_map.valbs[reg*2]<<8) | reg_map.valbs[reg*2+1];
      }
      reg++;
    }
    modbusSendBytes(mod.addr, mod.fcode, mod.rxBuf, 4, 0);  //发送一帧 modbus 响应数据，从机发送无需无超时
  }
}
```

串口屏每次收到主机的读操作、写操作都会执行 modbus 数据接收的脚本，因此在脚本里面，需要检测功能码，根据功能码(读操作或写操作)做处理，并发送响应帧。作为从机时，建议用一个数组控件 reg_map 来缓存寄存器的值，便于操作。

定时器控件用于模拟温度和湿度值的变化，并检测是否需要报警，脚本代码如下：

```
/***模拟数据变化**********************/
temp.valf += 0.2;
if(temp.valf > 40)temp.valf = 20;
hum.val += 1;
if(hum.val > 70)hum.val = 30;
/***********************************/


//将数据放到寄存器列表中
floatToBytes(temp.valf,reg_map.valbs,0,1);
reg_map.valbs[8] = (byte)(hum.val>>8);
reg_map.valbs[9] = (byte)(hum.val>>0);


//检测是否要告警
if(temp.valf > temp_alarm.valf)
{
  sta1.img = user.image.home2;
}
else
{
```

```
    sta1.img = user.image.home1;
}
if(hum.val > hum_alarm.val)
{
    sta2.img = user.image.home2;
}
else
{
    sta2.img = user.image.home1;
}
```

4.下载验证

编译成功后，点击下载按钮，选择正确的端口号和波特率，下载到串口屏。可以看到显示如下显示：

用串口助手按照 modbus 协议帧，给串口屏发送查询和设置命令。测试命令如下：

获取温度：
助手发送　　01 03 00 00 00 02 C4 0B
串口屏回复　01 03 04 41 E4 41 E4 9E 23　　//28.5 度

设置温度告警值
助手发送　　01 10 00 02 00 02 04 41 F1 99 9A DD 82 //30.2 度
串口屏回复　01 10 00 02 00 02 E0 08

获取湿度
助手发送　　01 03 00 04 00 01 C5 CB
串口屏回复　01 03 02 00 2B F8 5B

设置湿度告警值
助手发送　　01 10 00 05 00 01 02 00 3C A6 14
串口屏回复　01 10 00 05 00 01 11 C8

注意：这里温度是小数表示，4 个字节表示一个小数，大端模式，如 41 E4 41 E4，表示 28.5。

将本实验的从机 接到实验三 的主机串口上，主机可以看到从机的温湿度参数，并且可以设置温湿度告警值。

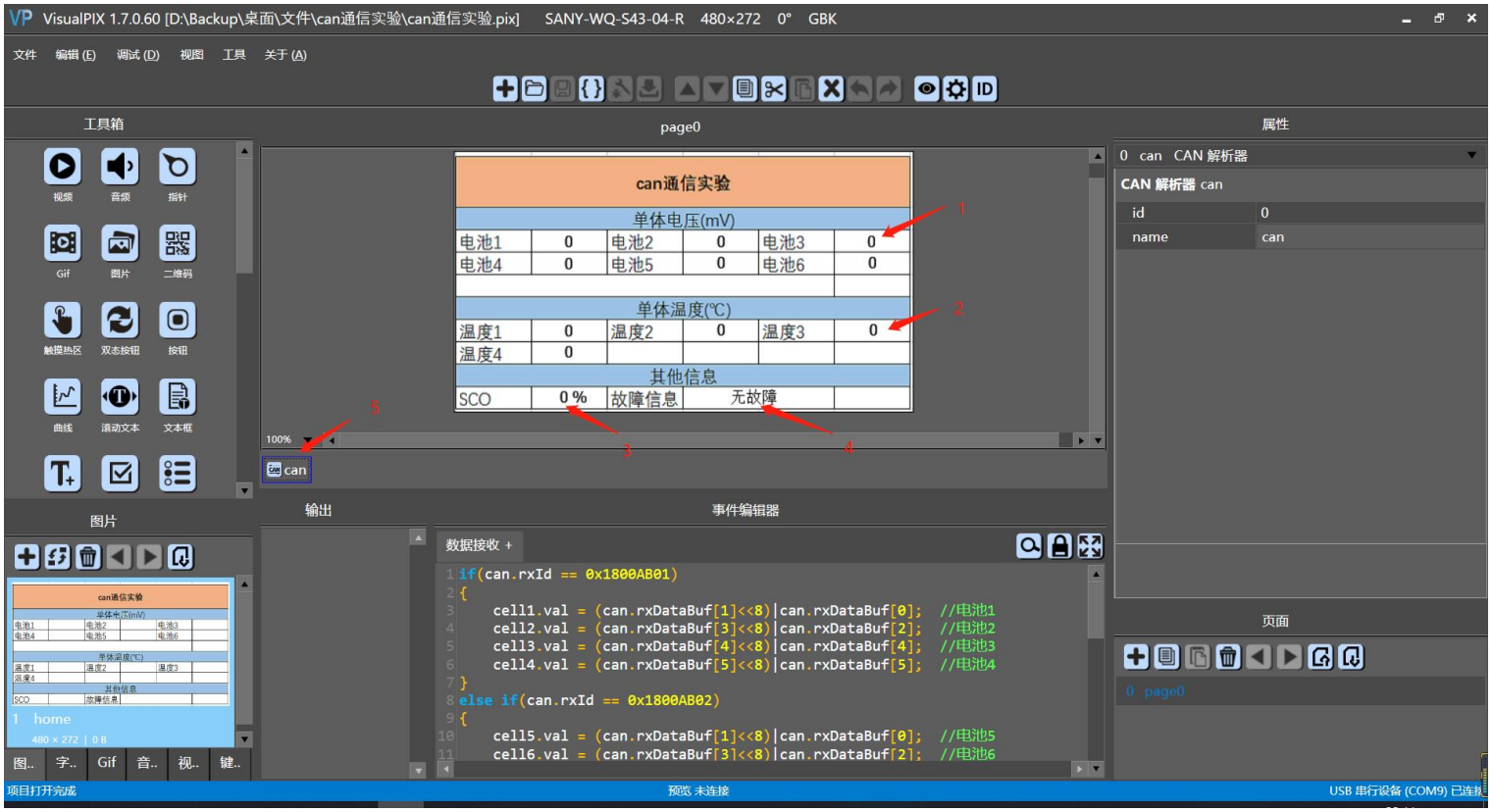
实验 5　can 通信实验

1.实验目的

按照指定协议，解析 can 报文，显示电压温度等信息。

说明：4.3 寸及以上带 can 版本的串口屏，支持 can 通信。

2.页面设计



(1)整数控件，显示电压值
(2)整数控件，显示温度值
(3)整数控件，显示 SOC
(4)文本控件，显示故障信息
(5)can 协议解析器控件，可以编写脚本代码，解析 can 报文。

图片资源，使用了一个 480*272 分辨率的背景图片。页面的背景图片额可以使用专业的绘图软件更加丰富、绚烂的背景图片。图片素材决定了显示的整体外观，让界面的设计更加容易。
字体字体除了工程自带的字库 Tahoma_4x_ASCII（包括英文和数字），另外创建了一个中文的字库，双击这个字库，可以看到详细的信息。字库详细制作方法可以参考第三章的添加字库小节。

3.串口屏协议处理

该例程使用了 can 解析器。串口屏每次收到一帧 can 数据，都会触发一次 can 解析器的脚本，和单片机的 can 中断函数类似。

协议定义（仅供参考）

扩展数据帧，波特率 250K		
ExdId	data[0~7]	
0x1800AB01	[0~1]电池 1 电压值 [2~3]电池 2 电压值 [4~5]电池 3 电压值 [6~7]电池 4 电压值	小端模式，单位 mV
0x1800AB02	[0~1]电池 5 电压值 [2~3]电池 6 电压值	小端模式，单位 mV
0x1800AB10	[0] 温度 1 [1] 温度 2	小端模式，单位℃
0x1800AB11	[1] 温度 3 [2] 温度 4	小端模式，单位℃
0x1800AB12	[0]SOC	范围 0~100
0x1800AB13	[0]故障信息 00 无故障 01 电池故障 02 温度故障 03 系统故障	

选中 can 解析器，可看到事件编辑窗口，事件代码如下：

```
if(can.rxId == 0x1800AB01)
{
  cell1.val = (can.rxDataBuf[1]<<8)|can.rxDataBuf[0]; //电池 1
  cell2.val = (can.rxDataBuf[3]<<8)|can.rxDataBuf[2]; //电池 2
  cell3.val = (can.rxDataBuf[4]<<8)|can.rxDataBuf[4]; //电池 3
  cell4.val = (can.rxDataBuf[5]<<8)|can.rxDataBuf[5]; //电池 4
}
else if(can.rxId == 0x1800AB02)
{
  cell5.val = (can.rxDataBuf[1]<<8)|can.rxDataBuf[0]; //电池 5
  cell6.val = (can.rxDataBuf[3]<<8)|can.rxDataBuf[2]; //电池 6
}
else if(can.rxId == 0x1800AB11)
{
  t1.val = can.rxDataBuf[0];//温度 1
  t2.val = can.rxDataBuf[1];//温度 2
  t3.val = can.rxDataBuf[2];//温度 3
```

```
  t4.val = can.rxDataBuf[3];//温度 4
}
else if(can.rxId == 0x1800AB12)
{
  soc.val = can.rxDataBuf[0];//SOC
}
else if(can.rxId == 0x1800AB13)  //故障信息
{
  if(can.rxDataBuf[0] == 0x00)
    fault.txt = "无故障";
  else if(can.rxDataBuf[0] == 0x01)
    fault.txt = "电池故障";
  else if(can.rxDataBuf[0] == 0x02)
    fault.txt = "温度故障";
  else if(can.rxDataBuf[0] == 0x03)
    fault.txt = "系统故障";
}
```

事件解析器的代码很简单，can.rxId 存放报文 id, can.rxDataBuf 存放数据，数组长度 8 字节。按照协议解析数据即可。

4.硬件过滤器的配置

显示屏接入 can 总线时，可以设置硬件过滤器。硬件上过滤掉不需要接收的报文，减少显示屏处理过多无用报文的压力，提高显示屏刷屏流畅度。设计 can 程序时，配置过滤器很有必要。

显示屏 can 收发调试过程中，可以不配置过滤器，方便调试。工程调试完成后，再将掩码和滤码设置成合适的值。

显示屏过滤器最多有 14 个，编号 0~13，每个过滤器有两种过滤模式：屏蔽位模式、标识符列表模式

屏蔽位模式：

需要设置过滤标识符 filter 和过滤掩码 mask。硬件收到一帧 can 报文 id 时，按照 mask 为 1 的 bit,与过滤标识符对比，全部 mask 位相同时，则接收该报文，否则拒收。换个说法：如果(id & mask) == filter，接收此报文;否则不符合过滤要求， 不接收此报文。 适用于要接收 can 报文比较多且报文 id 有一定规律。

标识符列表模式：

硬件收到一帧 can 报文 id 时，与过滤器列表一一对应，如果有匹配的报文 id 则接收该报文，否则拒收。每个过滤器可以设置两个 ID，所以最多可设置 28 个。适用于要接收的 can 报文比较少的情况，可以实现 can 报文的精准过滤。

串口屏的过滤器设置可以通过项目设置界面配置(只能配置过滤器 0 的屏蔽位模式)，也能在脚本里调用 can 控件的方法 setFilter 来配置(通过参数来配置过滤器的参数，可以灵活的设置的工作模式)。

1、项目设置界面；

打开 ”can 通信实验.pix”工程，点击项目设置图标，可以查看到 can 的波特率、过滤报文的掩码和滤码。

这种配置方式，只能设置过滤器 0 的屏蔽位模式。适用于过滤器简单的设置，优点是配置简单

计算掩码 mask 和滤码 filter 的时候，考虑所有要接收的报文 id，找出相同的 bit 和有差异的 bit。该例程的所有要接收的报文有 6 个 id，如下：

0x1800AB01

0x1800AB02

0x1800AB10

0x1800AB11

0x1800AB12

0x1800AB13

有差异的 bit 有[1:0]、[4]，其余 bit 都是相同的。则 mask 的[1:0]、[4]为 0，其他为 1，即 mask=0xFFFFFFEC。相同的 bit 即为 filter，即 filter=0x1800AB00。最后可将上面 5 个验证 是否(id & mask) == filter .

2、调用 can 控件的方法 setFilter

当过滤的规则比较复杂时，建议使用 setFilter 方法来配置过滤器。

打开 ”can 通信实验_setFilter.pix”工程，点击 page0 空白处，可以看到加载页面事件，脚本代码如下

加载页面 + 离开页面

```
//用startup标记，上电只执行一次
if(startup.val == 1)
{
    startup.val = 0;

    //配置过滤器


    //标识符列表模式
    can.setFilter(0,1,(0x1800AB01<<3)|0x04,(0x1800AB02<<3)|0x04);
    can.setFilter(1,1,(0x1800AB10<<3)|0x04,(0x1800AB11<<3)|0x04);
    can.setFilter(2,1,(0x1800AB12<<3)|0x04,(0x1800AB13<<3)|0x04);




    //屏蔽位模式
    //can.setFilter(0,0,(0xFFFFFFEB<<3)|0x04,(0x1800AB00<<3)|0x04);

}
```

这里需要注意的是，该脚本是首页的加载页面事件，用了一个变量 startup 来标记该脚本只执行一次，因为配置过滤器只需上电配置一次就可以了。

```
void    setFilter(u8 filterid,u8 mode,u32 param1, u32 param2);
/*
函数功能：配置 can 接收过滤器
参数 1：filterid,过滤器编号，0~13
参数 2：mode,过滤器模式，0 屏蔽位模式，1 标识符列表模式
参数 3：param1,过滤器参数 1，参数位定义见下表
参数 4：param2,过滤器参数 2，参数位定义见下表
*/
```

过滤器 param1 和 param2 位定义：

位 31:21	STID[10:0]/EXID[28:18]:标准标识符或扩展标识符 依据 IDE 位的内容，这些位　或是标准标识符(11bit)，或是扩展标识符的高 11bit
位 20:3	EXID[17:0]扩展标识符低 18bit
位 2	IDE:标识符类型 0：标准帧 1：扩展帧
位 1	RTR:远程发送请求 0：数据帧 1：远程帧
位 0	保留位

标识符列表模式

```
can.setFilter(0,1,(0x1800AB01<<3)|0x04,(0x1800AB02<<3)|0x04); //过滤器 0
can.setFilter(1,1,(0x1800AB10<<3)|0x04,(0x1800AB11<<3)|0x04); //过滤器 1
can.setFilter(2,1,(0x1800AB12<<3)|0x04,(0x1800AB13<<3)|0x04); //过滤器 2
```

屏蔽位模式

```
can.setFilter(0,0,(0xFFFFFFEC<<3)|0x04,(0x1800AB00<<3)|0x04); //过滤器 0
```

方法 setFilter 的参数 patam1 和 param2 都与 0x04 按位或运算，表示只接收扩展帧。

注：

假如要接收标准帧，如 0x6A1(标准帧)，0x6A2(标准帧)，可以这样设置：

```
can.setFilter(0,1,0x6A1<<21,0x6A2<<21); 过滤器 0，标识符列模式
```

或者：

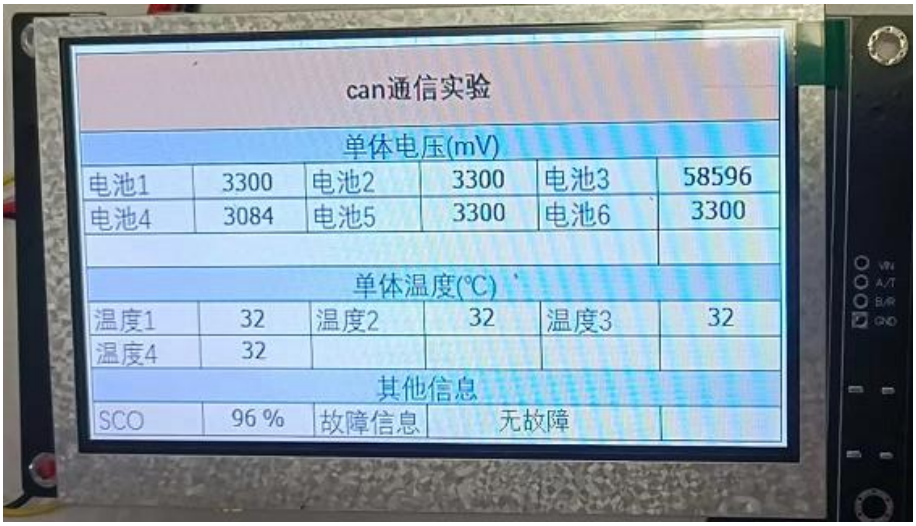
```
can.setFilter(0,0,0xFFC<<21,0x6A0<<21); 过滤器 0，屏蔽位模式
```

至于这里为什么是左移 21 位，可以参考上表 param1 和 param2 位定义。

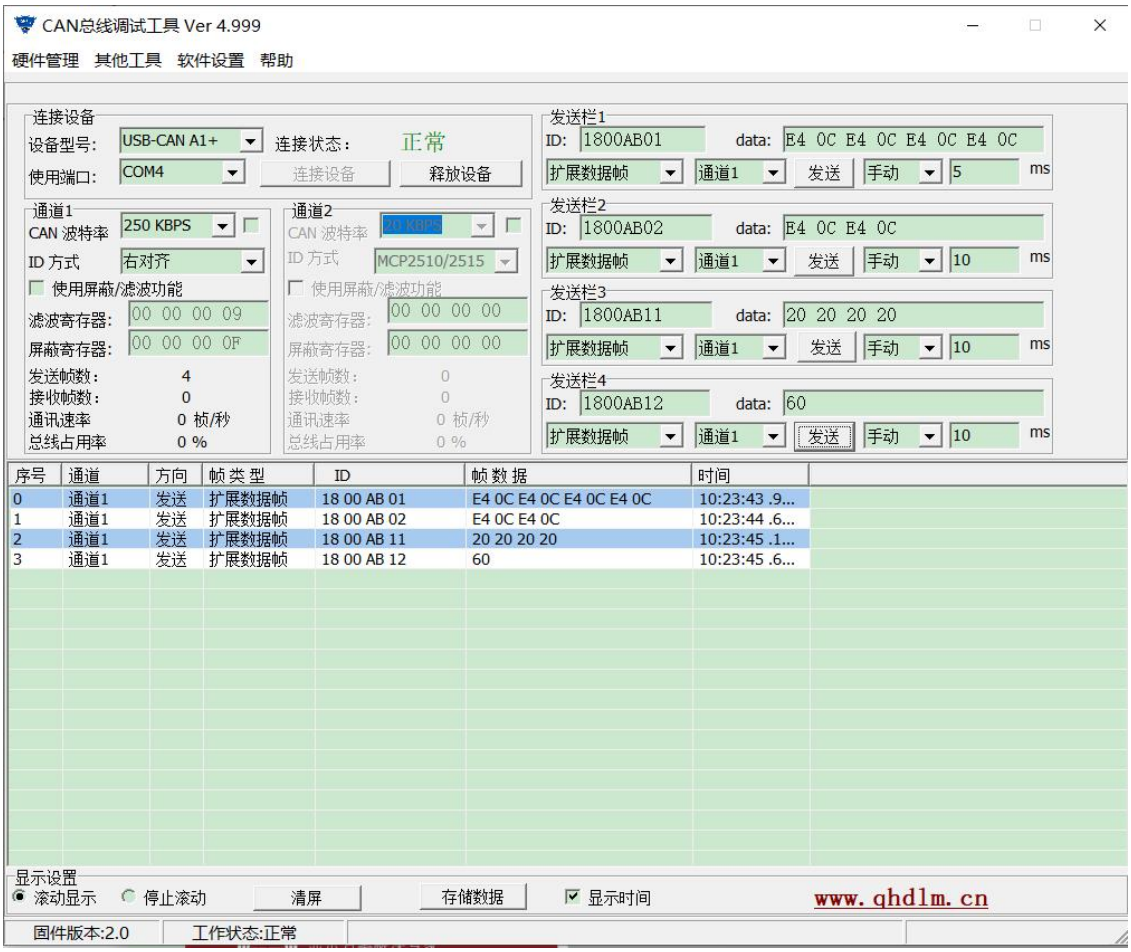
5.下载验证

编译成功后，点击下载按钮，选择正确的端口号和波特率，下载到串口屏。将串口屏接到 can 总线上，串口屏收到正确报文时，会解析并显示。

调试时，用任意 can 收发器的助手，给显示屏发报文测试即可。界面如下：



can 调试工具发送数据



测试命令如下：

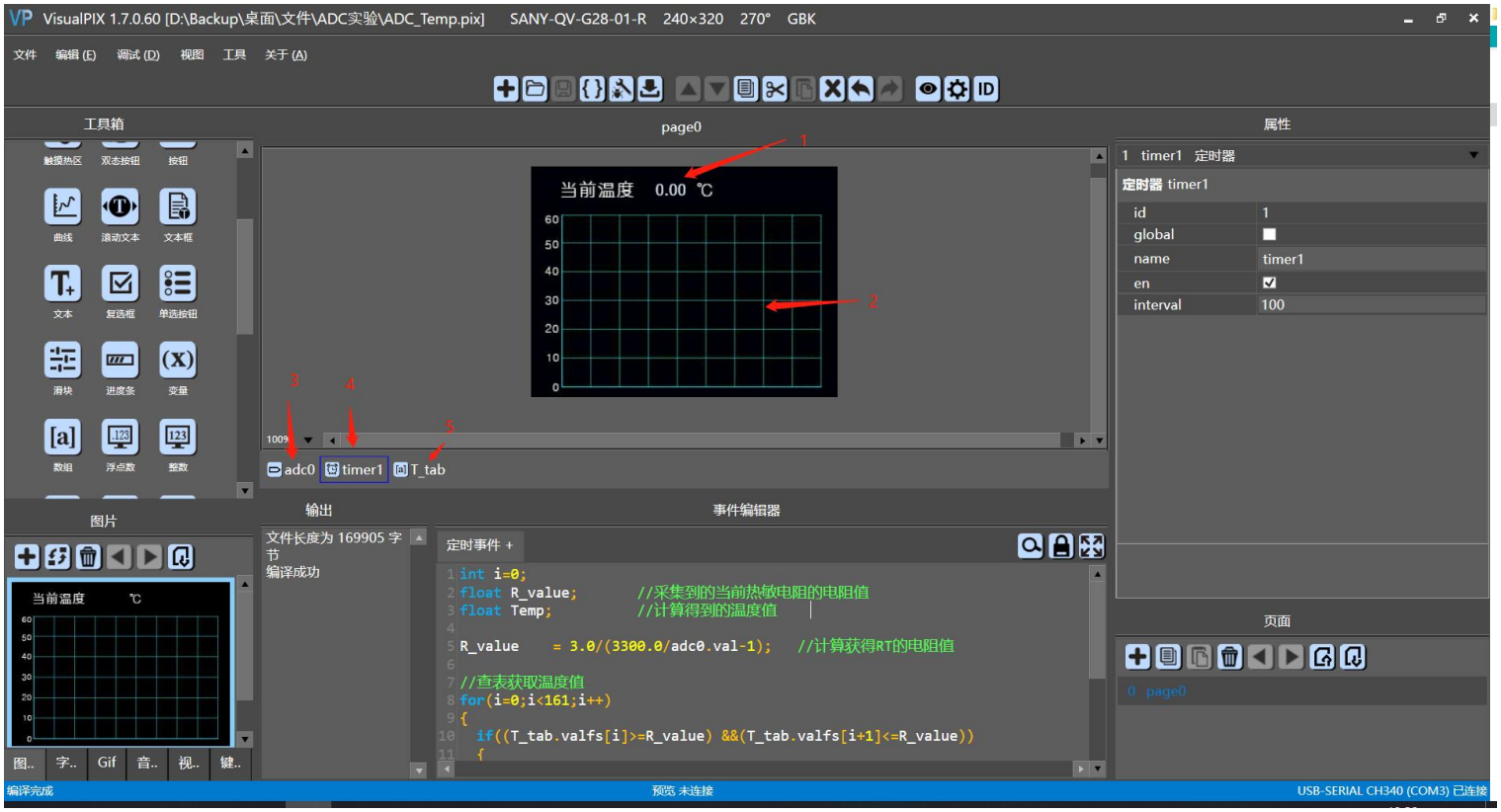
报文 ID	数据	说明
1800AB01	E4 0C E4 0C E4 0C E4 0C	设置电池 1~4
1800AB02	E4 0C E4 0C	设置电池 5~6
1800AB10	20 20	设置温度 1~2
1800AB11	20 20	设置温度 3~4
1800AB12	60	设置 SOC
1800AB13	00	故障信息

实验 6　ADC 和曲线控件实验

1.实验目的

使用串口屏自带的 AD 输入引脚采样温度模块的电压，计算出温度，显示当前温度并绘制温度曲线。

2.页面设计



(1)浮点数控件，显示当前温度值
(2)曲线控件，显示曲线，温度的变化
(3)adc 控件，可以获取 IO 电压值
(4)定时器，间隔一秒，读取 AD 电压并计算温度，显示曲线
(5)数组控件，存放 电阻-温度对应表，计算温度时使用
选中控件，可以在右侧属性栏查看控件的各个属性。详细属性说明可参考 第四章 控件的介绍和用法。

G 系列串口屏支持 3 路 AD 输入。本实验串口屏型号为 SANY-QV-G28-01-R，只用到了 1 路 AD。S 系列串口屏可以支持更多路的 AD 输入。
曲线控件属性配置时，注意通道点数 pCount 的正确设置。Width >= xStep * pCount，否则曲线显示可能达不到预期的效果。

3.串口屏协议处理

本实验演示 ADC 控件和曲线控件的用法。无通信协议。部分脚本代码如下：

页面的加载事件

```
//进入这个页面的时候，会执行一次加载页面事件
//温度表初始化

int i;
float RT_Tab[161] =
{
    366.4286,344.5753,324.1796,305.1344,287.3413,270.7097,255.1561,240.6035,226.9810,214.2231,   // -50℃~-41℃
    202.2693,191.0637,180.5546,170.6944,161.4387,152.7468,144.5807,136.9052,129.6879,122.8985,   // -40℃~-31℃
    116.5089,110.4932,104.8272, 99.4883, 94.4558, 89.7101, 85.2332, 81.0082, 77.0194, 73.2523,   // -30℃~-21℃
     69.6931, 66.3291, 63.1485, 60.1402, 57.2939, 54.5998, 52.0490, 49.6330, 47.3439, 45.1743,   // -20℃~-11℃
     43.1172, 41.1663, 39.3153, 37.5587, 35.8910, 34.3074, 32.8029, 31.3734, 30.0145, 28.7225,   // -10℃~-1℃
     27.4936, 26.3245, 25.2119, 24.1527, 23.1442, 22.1835, 21.2682, 20.3959, 19.5644, 18.7714,   //  0℃~ 9℃
     18.0151, 17.2935, 16.6048, 15.9475, 15.3198, 14.7203, 14.1475, 13.6003, 13.0772, 12.5771,   //  10℃~ 19℃
     12.0988, 11.6413, 11.2037, 10.7848, 10.3839, 10.0000,  9.6324,  9.2802,  8.9428,  8.6195,   //  20℃~ 29℃
      8.3096,  8.0124,  7.7275,  7.4541,  7.1919,  6.9403,  6.6987,  6.4669,  6.2442,  6.0304,   //  30℃~ 39℃
      5.8250,  5.6276,  5.4380,  5.2557,  5.0804,  4.9119,  4.7498,  4.5939,  4.4439,  4.2995,   //  40℃~ 49℃
      4.1605,  4.0268,  3.8980,  3.7739,  3.6544,  3.5393,  3.4284,  3.3215,  3.2185,  3.1191,   //  50℃~ 59℃
      3.0234,  2.9310,  2.8419,  2.7559,  2.6729,  2.5929,  2.5156,  2.4410,  2.3690,  2.2994,   //  60℃~ 69℃
      2.2322,  2.1673,  2.1046,  2.0440,  1.9854,  1.9288,  1.8740,  1.8211,  1.7699,  1.7204,   //  70℃~ 79℃
      1.6725,  1.6262,  1.5813,  1.5379,  1.4959,  1.4553,  1.4159,  1.3778,  1.3408,  1.3051,   //  80℃~ 89℃
      1.2704,  1.2368,  1.2043,  1.1728,  1.1422,  1.1126,  1.0839,  1.0560,  1.0290,  1.0028,   //  90℃~ 99℃
      0.9774,  0.9527,  0.9288,  0.9055,  0.8830,  0.8611,  0.8399,  0.8193,  0.7992,  0.7798,   // 100℃~109℃
      0.7609                                                                                      // 110℃
};

//温度对应表，初始化
for(i=0;i<161;i++)
{
  T_tab.valfs[i] = RT_Tab[i];
}
```

页面加载事件，仅仅在进入页面的时候会执行脚本代码。本实验只有 1 页，页面加载事件只在上电的时候执行一次，所以可以在这里做一些初始化，这里是初始化 电阻-温度表的数组。

定时器事件

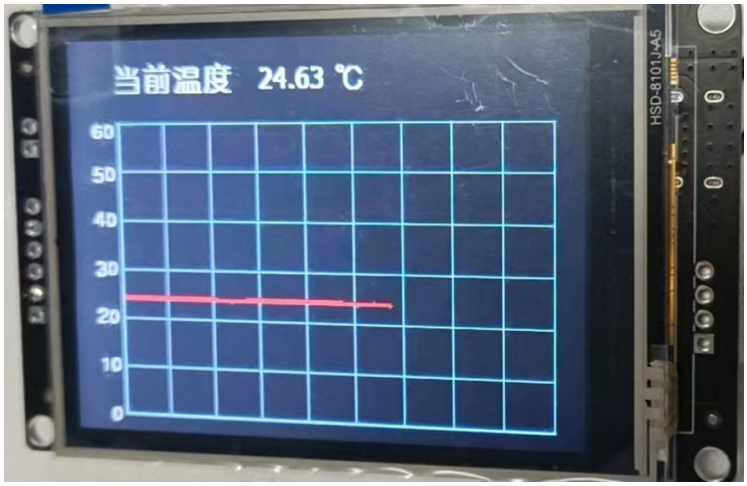
```
int i=0;
float R_value;     //采集到的当前热敏电阻的电阻值
float Temp;        //计算得到的温度值

R_value   = 3.0/(3300.0/adc0.val-1);  //计算获得 RT 的电阻值①

//查表获取温度值
for(i=0;i<161;i++)
{
 if((T_tab.valfs[i]>=R_value) &&(T_tab.valfs[i+1]<=R_value))
 {
  Temp = i + (R_value - T_tab.valfs[i+1])/(T_tab.valfs[i] - T_tab.valfs[i-1]) - 50;
 }
```

```
}

numf3.valf = Temp;  //显示当前温度
curvedLine2.ch1Val = Temp; //曲线控件添加一个点②
```

①adc0.val 是 AD 引脚的电压值，单位 mV。这里是计算 RT 电阻值
②曲线控件添加一个点，注意曲线控件的数值类型和赋值的类型应该一致。

4.下载验证

编译成功后，点击下载按钮，选择正确的端口号和波特率，下载到串口屏。可以看到显示如下显示：



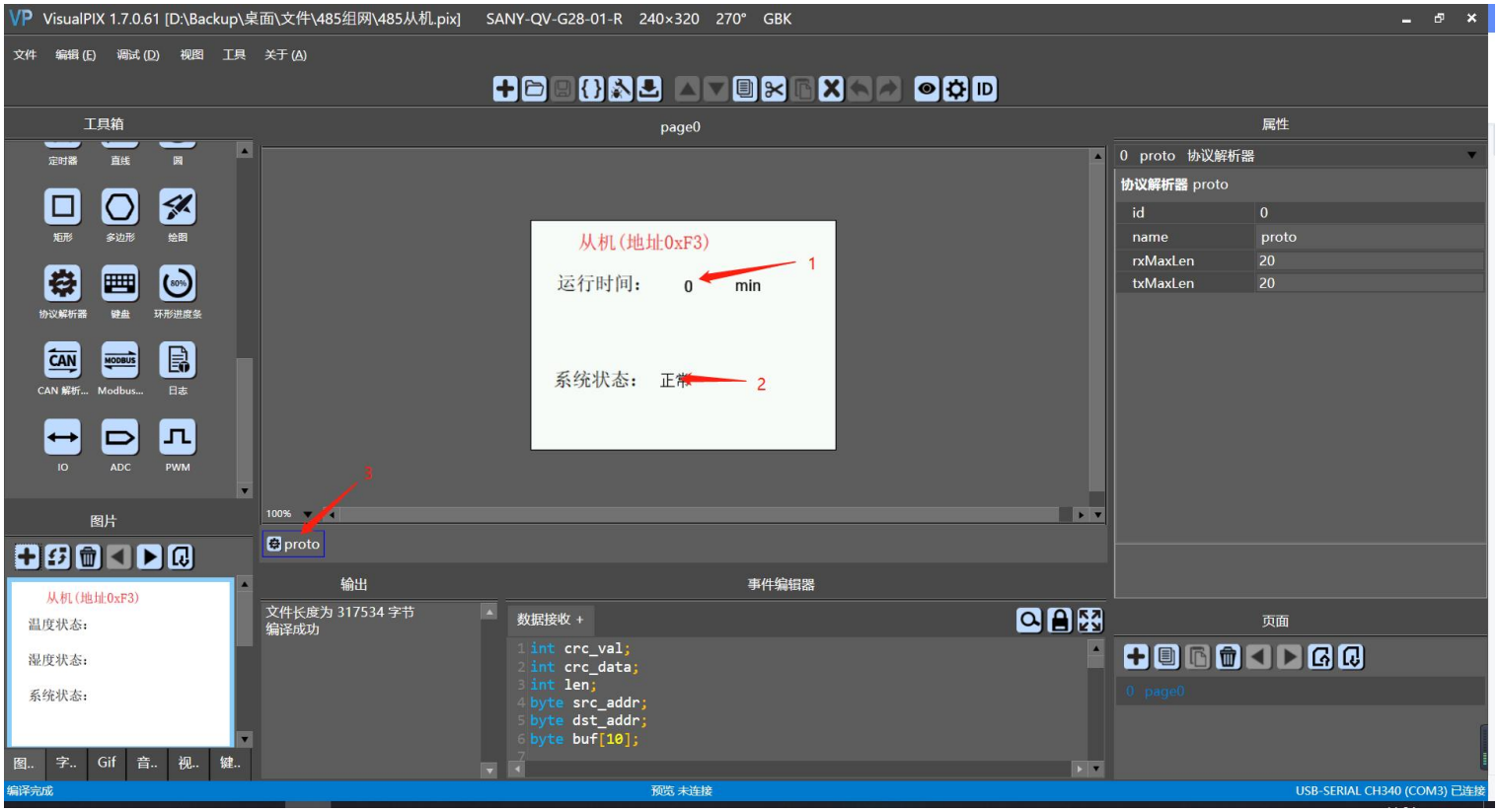
实验 7　485 组网实验_从机

1.实验目的

本实验为非 mobus 方式，实现 485 组网。modbus 组网参考实验三和实验四。

带 485 接口的串口屏，也可以进行 485 组网。要进行 485 组网，需要定义一套比较完整的协议。主机是谁，从机是哪些，主机和从机的地址需要定义好，不要冲突。
比较常见的，是 485 总线上一主机多从机的模式，主机可以主动发起查询和设置，从机只能被动响应主机的命令。
本实验和实验二类似，区别是解析的协议不同。本实验的协议仅仅简单的体现 485 怎么实现组网，仅供参考。

2.页面设计



(1)整数控件，显示系统运行时间
(2)文本控件，中文显示系统运行状态
(3)协议解析器控件，解析串口协议

3.串口屏协议处理

本实验通信协议定义如下：
报文帧格式

帧头	原地址	目的地址	命令码	参数	校验
0xA5	1Byte	1Byte	1Byte	N	2Byte

主机：STM32F103 实验板　主机地址 0x01

从机地址	命令码	参数长度	参数	校验
0xF1 温湿度模块	0x01 主机发送 查询温湿度值	0 字节		
	0xF1 从机响应	6 字节	[0~3]温度值 [4~5]湿度值	
	0x02 主机发送 查询告警温度值	0 字节		
	0xF2 从机响应	6 字节	[0~3]温度告警值 [4~5]湿度告警值	
	0x03 主机发送 设置告警温度值	6 字节	[0~3]温度告警值 [4~5]湿度告警值	
	0xF3 从机响应	1 字节	[0]是否成功	0 成功，1 失败
0xF3 从机显示屏	0x01 主机发送 系统运行时间	2 字节	[0~1]运行时间	单位，分钟
	0xF1 从机响应	1 字节		
	0x02 主机发送 系统运行状态	1 字节	[0]运行状态	0 系统正常 1 系统故障
	0xF2 从机响应	0 字节		

本实验是从机串口屏，地址为 0xF3，显示主机的系统参数。

串口屏处理串口数据，在协议解析器控件 proto 的接收事件里面。包括帧头，地址，命令码和校验的判断，参数获取然后赋值给控件，显示出来。

协议解析器接收事件的脚本代码

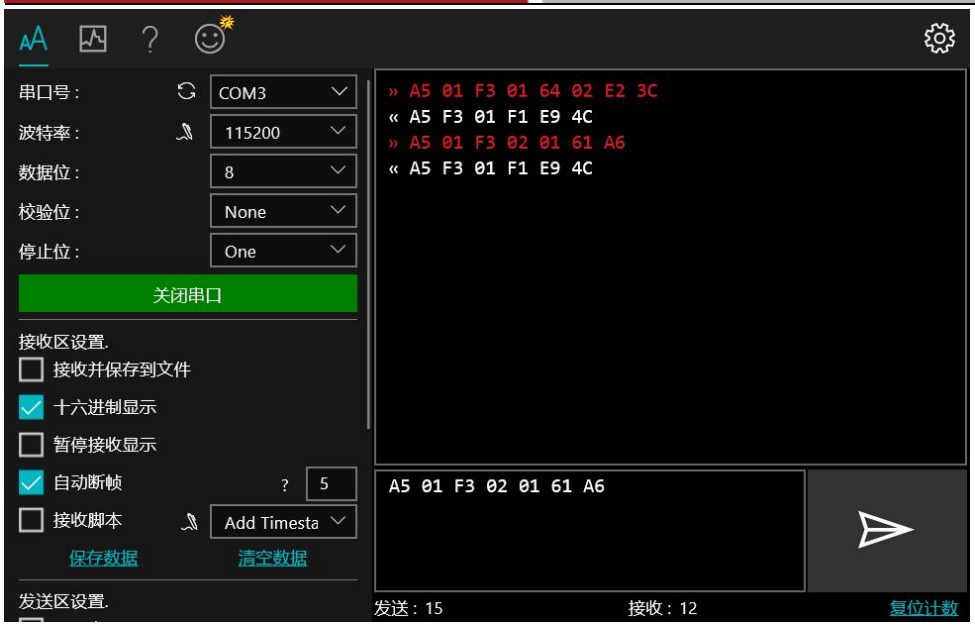
```
int crc_val;
int crc_data;
int len;
byte src_addr;
byte dst_addr;
byte buf[10];

if(proto.rxBuf[0] == 0xA5)
{
  src_addr = proto.rxBuf[1];
  dst_addr = proto.rxBuf[2];

  if(dst_addr == 0xF3)  //检测该命令，是否是本机的命令
  {
    len = proto.rxLen;

    crc_data = (proto.rxBuf[len-1]<<8) | proto.rxBuf[len-2];
    crc_val  = crc16(proto.rxBuf,0,len-2);
    if(crc_data == crc_val) //CRC 判断
    {
```

```
        if(proto.rxBuf[3] == 0x01)  //主机设置温湿度故障信息
        {
          time.val = (proto.rxBuf[5]<<8) | proto.rxBuf[4];

          //命令响应
          buf[0] = 0xA5;
          buf[1] = dst_addr;
          buf[2] = src_addr;
          buf[3] = 0xF1;
          crc_val= crc16(buf,0,4); //计算 CRC 值
          buf[4] = (byte)crc_val;
          buf[5] = (byte)(crc_val>>8);
          uartSendBytes(buf, 0, 6);//发送数组的数据
        }
        else if(proto.rxBuf[3] == 0x02) //主机设置系统运行状态
        {
          if(proto.rxBuf[4] == 0)
            sta.txt = "正常";
          else
            sta.txt = "故障";

          //命令响应
          buf[0] = 0xA5;
          buf[1] = dst_addr;
          buf[2] = src_addr;
          buf[3] = 0xF1;
          crc_val= crc16(buf,0,4); //计算 CRC 值
          buf[4] = (byte)crc_val;
          buf[5] = (byte)(crc_val>>8);
          uartSendBytes(buf, 0, 6);//发送数组的数据

        }
      }
    }
}
```

每收到一次正确命令。解析完成后，需要组帧，然后调用 uartSendBytes，响应主机的命令。

4.下载验证

编译成功后，点击下载按钮，选择正确的端口号和波特率，下载到串口屏。
调试时，用电脑的串口助手给串口屏发命令



测试命令

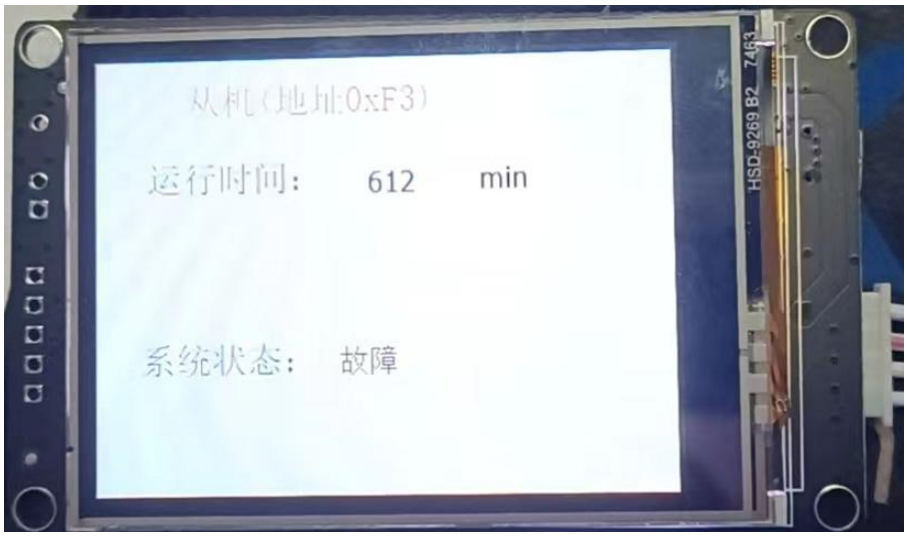
» A5 01 F3 01 64 02 E2 3C　//运行时间 612

« A5 F3 01 F1 E9 4C　　//响应

» A5 01 F3 02 01 61 A6　　//系统状态 故障

« A5 F3 01 F1 E9 4C　　//响应

显示屏如下图显示，则正确。



实验 8　485 组网实验_主机

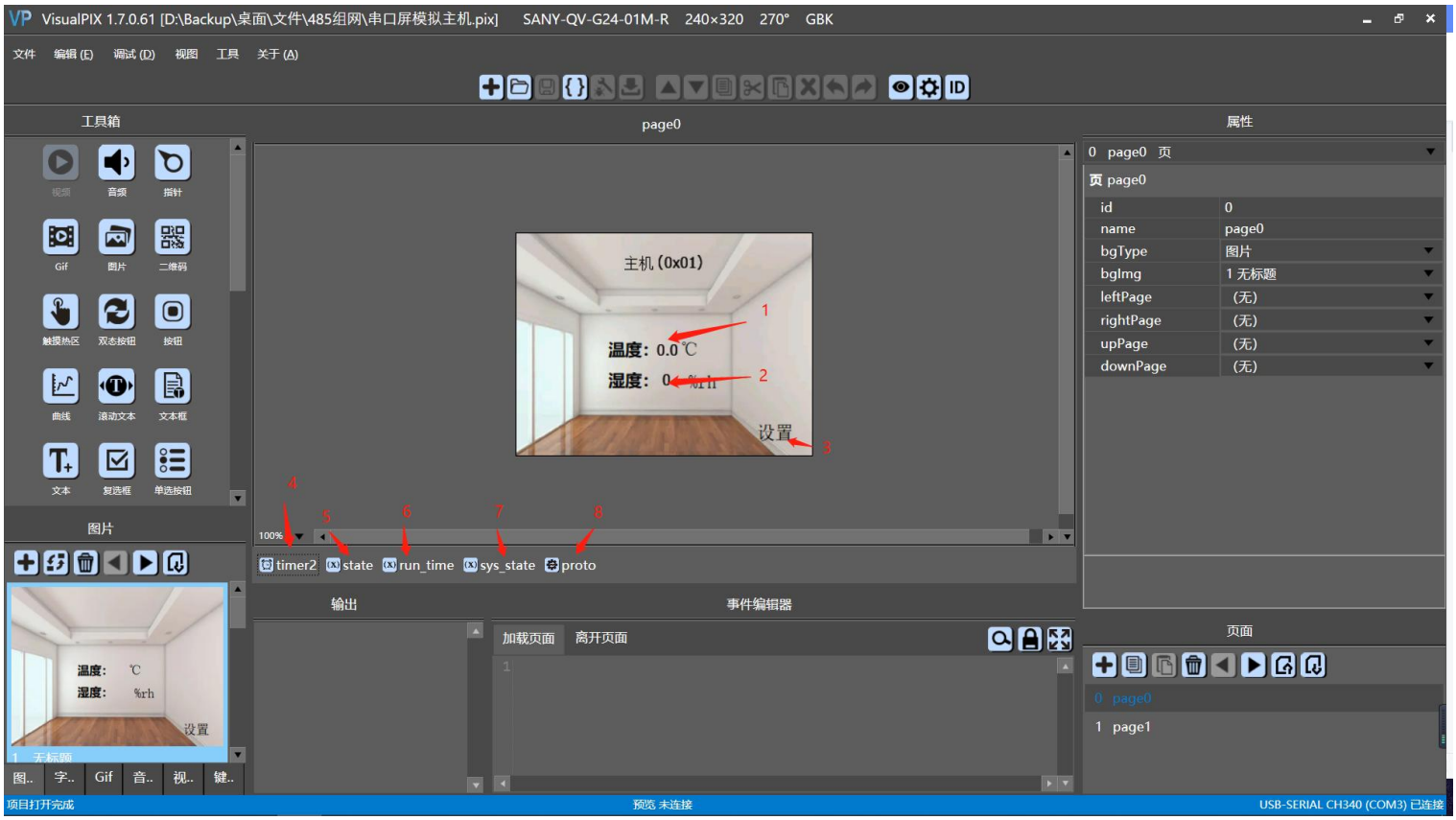
1.实验目的

本实验为非 mobus 方式，实现 485 组网。modbus 组网参考实验三和实验四。

带 485 接口的串口屏，也可以进行 485 组网。要进行 485 组网，需要定义一套比较完整的协议。主机是谁，从机是哪些，主机和从机的地址需要定义好，不要冲突。
比较常见的，是 485 总线上一主机多从机的模式，主机可以主动发起查询和设置，从机只能被动响应主机的命令。
本实验的协议仅仅简单的体现 485 怎么实现组网，仅供参考。

本实验模拟实验八中的主机。

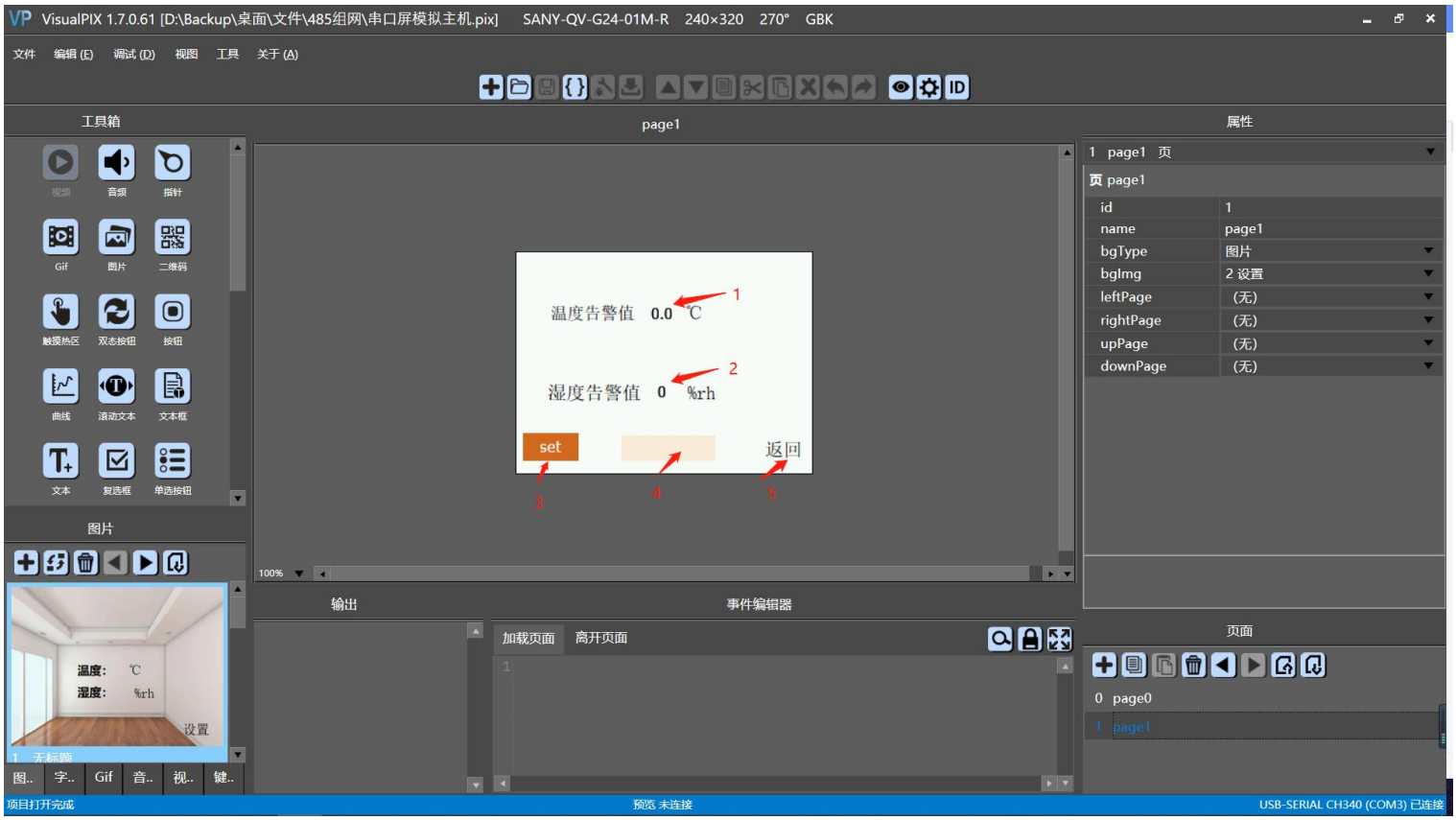
2.页面设计



(1)浮点数控件，显示温度值
(2)整数控件，显示湿度
(3)按钮，翻页
(4)定时器，定时器间隔 100ms，定时时间到会执行定时器脚本。在定时器里面发送查询和设置命令
(5)变量控件，存放命令发送状态
(6)变量控件，存放系统运行时间
(7)变量控件，存放系统故障状态
(8)协议解析器控件，处理接收到的串口命令

选中控件，可以在右侧属性栏查看控件的各个属性和事件代码。详细属性说明可参考 第四章 控件的介绍和用法。

Page1



1、浮点数控件，显示温度告警值
2、整数控件，显示湿度告警值
3、按钮，点击按钮时会发送设置命令
4、文本控件，显示设置成功或失败
5、按钮，返回 page0
选中控件，可以在右侧属性栏查看控件的各个属性和事件代码。详细属性说明可参考 第四章 控件的介绍和用法。

3.串口屏协议处理

本实验通信协议定义如下：
报文帧格式

帧头	原地址	目的地址	命令码	参数	校验
0xA5	1Byte	1Byte	1Byte	N	2Byte

主机：STM32F103 实验板 主机地址 0x01

从机地址	命令码	参数长度	参数	校验
0xF1 温湿度模块	0x01 主机发送 查询温度值	0 字节		
	0xF1 从机响应	6 字节	[0~3]温度值 [4~5]湿度值	
	0x02 主机发送 查询告警温度值	0 字节		

	0xF2 从机响应	6 字节	[0~3]温度告警值 [4~5]湿度告警值	
	0x03 主机发送 设置告警温度值	6 字节	[0~3]温度告警值 [4~5]湿度告警值	
	0xF3 从机响应	1 字节	[0]是否成功	0 成功，1 失败
0xF3 从机显示屏	0x01 主机发送 系统运行时间	2 字节	[0~1]运行时间	单位，分钟
	0xF1 从机响应	1 字节		
	0x02 系统运行状态	1 字节	[0]运行状态	0 系统正常 1 系统故障
	0xF2 从机响应	0 字节		

本实验是串口屏模拟 STM32F103 实验板，地址为 0x01.

定时器用于间隔发送查询和设置命令，脚本代码如下

```
byte buf[10];
int crc_val;

if(state.val == 0) //查询温度值
{
  buf[0] = 0xA5;
  buf[1] = 0x01;
  buf[2] = 0xF1;
  buf[3] = 0x01;
  crc_val = crc16(buf,0,4);
  buf[4] = (byte)crc_val;
  buf[5] = (byte)(crc_val>>8);
  uartSendBytes(buf, 0, 6);//发送数组的数据
}
else if(state.val == 1) //查询温度告警
{
  buf[0] = 0xA5;
  buf[1] = 0x01;
  buf[2] = 0xF1;
  buf[3] = 0x02;
  crc_val = crc16(buf,0,4);
  buf[4] = (byte)crc_val;
  buf[5] = (byte)(crc_val>>8);
  uartSendBytes(buf, 0, 6);//发送数组的数据
}

else if(state.val == 2) //发送系统运行时间
{
  run_time.val++;
```

```
  buf[0] = 0xA5;
  buf[1] = 0x01;
  buf[2] = 0xF3;
  buf[3] = 0x01;
  buf[4] = (byte)run_time.val;
  buf[5] = (byte)(run_time.val>>8);
  crc_val = crc16(buf,0,6);
  buf[6] = (byte)crc_val;
  buf[7] = (byte)(crc_val>>8);
  uartSendBytes(buf, 0, 8);//发送数组的数据
}

else if(state.val == 3) //发送系统状态
{
  buf[0] = 0xA5;
  buf[1] = 0x01;
  buf[2] = 0xF3;
  buf[3] = 0x02;
  buf[4] = (byte)sys_state.val;
  crc_val = crc16(buf,0,5);
  buf[5] = (byte)crc_val;
  buf[6] = (byte)(crc_val>>8);
  uartSendBytes(buf, 0, 7);//发送数组的数据
}

state.val++;
if(state.val > 3)
  state.val = 0;

delay(20); //延时 20ms,等待从机响应
```

协议解析器处理串口接收的命令，脚本代码如下

```
int src_addr;
int dst_addr;
int len;
int crc_data;
int crc_val;

if(proto.rxBuf[0] == 0xA5)
{
  src_addr = proto.rxBuf[1];
  dst_addr = proto.rxBuf[2];

  if(dst_addr == 0x01)  //检测该命令，是否是本机的命令
  {
    len = proto.rxLen;
    crc_data = (proto.rxBuf[len-1]<<8) | proto.rxBuf[len-2];
    crc_val  = crc16(proto.rxBuf,0,len-2);
```

```
    if(crc_data == crc_val) //CRC 判断
    {
      if(src_addr == 0xF1) //温度模块的数据
      {
        if(proto.rxBuf[3] == 0xF1)
        {
          //获取温度值
          temp.valf = bytesToFloat(proto.rxBuf,4,0);

          //获取湿度值
          hum.val = (proto.rxBuf[9]<<8) | proto.rxBuf[8];
        }
        else if(proto.rxBuf[3] == 0xF2)
        {
          //获取温度告警值
          page1.temp_alarm.valf = bytesToFloat(proto.rxBuf,4,0);

          //获取湿度告警值
          page1.hum_alarm.val = (proto.rxBuf[9]<<8) | proto.rxBuf[8];
        }
        else if(proto.rxBuf[3] == 0xF3)
        {
          if(proto.rxBuf[4] == 0)
            page1.text5.txt = "设置成功";
          else
            page1.text5.txt = "设置失败";
        }
      }
      else if(src_addr == 0xF3) //从机显示屏的数据
      {
        //不用处理
      }

    }
  }
}
```

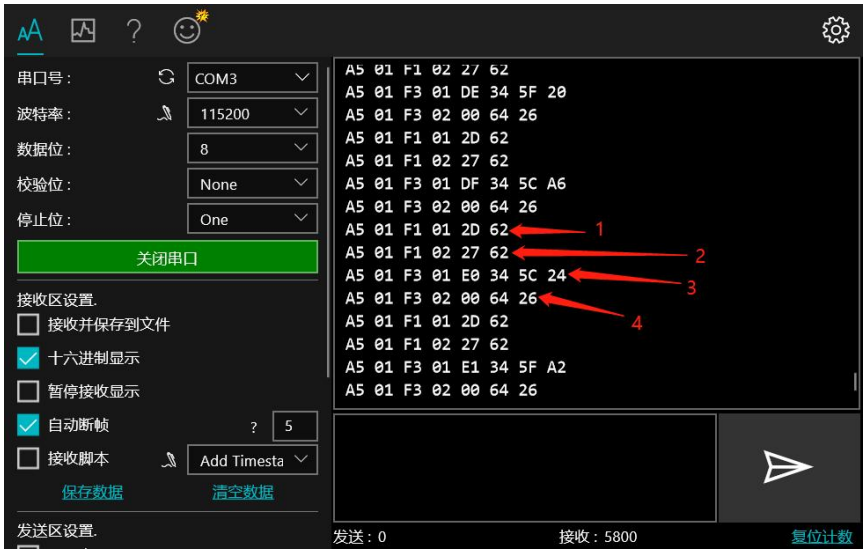
Page1 的设置按钮用于发送设置命令，脚本代码如下

```
byte buf[16];
int crc_val;



buf[0] = 0xA5;
buf[1] = 0x01;
buf[2] = 0xF1;
buf[3] = 0x03;

floatToBytes(temp_alarm.valf,buf,4,0);  //温度告警值
```

```
buf[8] = (byte)(hum_alarm.val>>0);    //湿度告警值 L
buf[9] = (byte)(hum_alarm.val>>8);     //湿度告警值 H
crc_val = crc16(buf,0,10);
buf[10] = (byte)crc_val;
buf[11] = (byte)(crc_val>>8);
uartSendBytes(buf, 0, 12);//发送数组的数据

page0.timer2.en = 1;
```

4.下载验证

编译成功后，点击下载按钮，选择正确的端口号和波特率，下载到串口屏。
调试时，用电脑的串口助手监控串口屏的数据。



可以看到串口屏不停的发出命令
(1)查询温度湿度值
(2)查询温度湿度告警值
(3)发送系统运行时间
(4)发送系统状态

发送测试命令

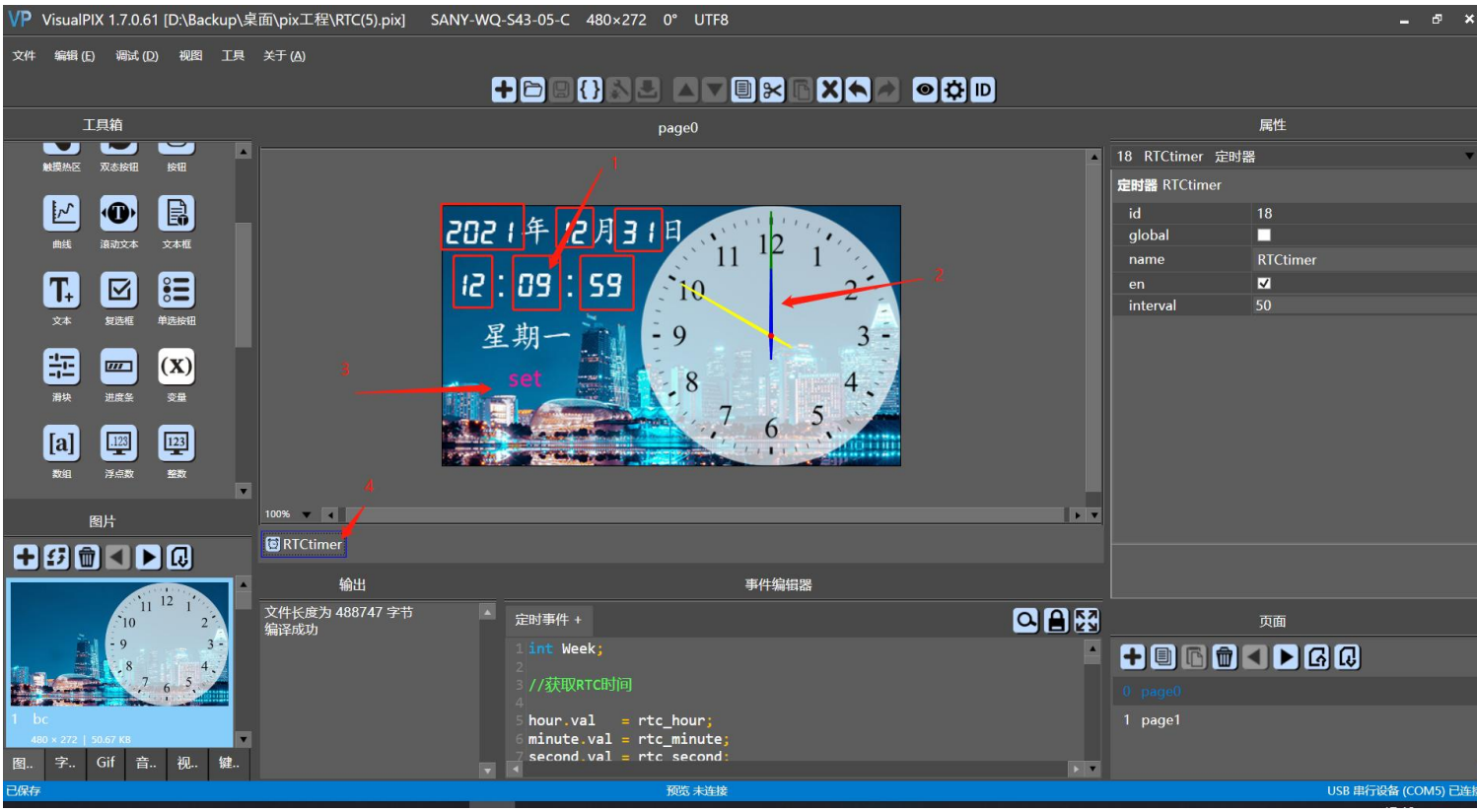
```
//模拟从机 0xF1 的响应
A5 F1 01 F1 33 33 BB 41 3A 00 29 47   //温度 23.4，湿度  58
A5 F1 01 F2 00 00 20 42 46 00 22 10   //温度告警 40，湿度告警 70
A5 F1 01 F3 00 AB 4C                  //设置成功
```

实验 9　RTC 显示和设置

1.实验目的

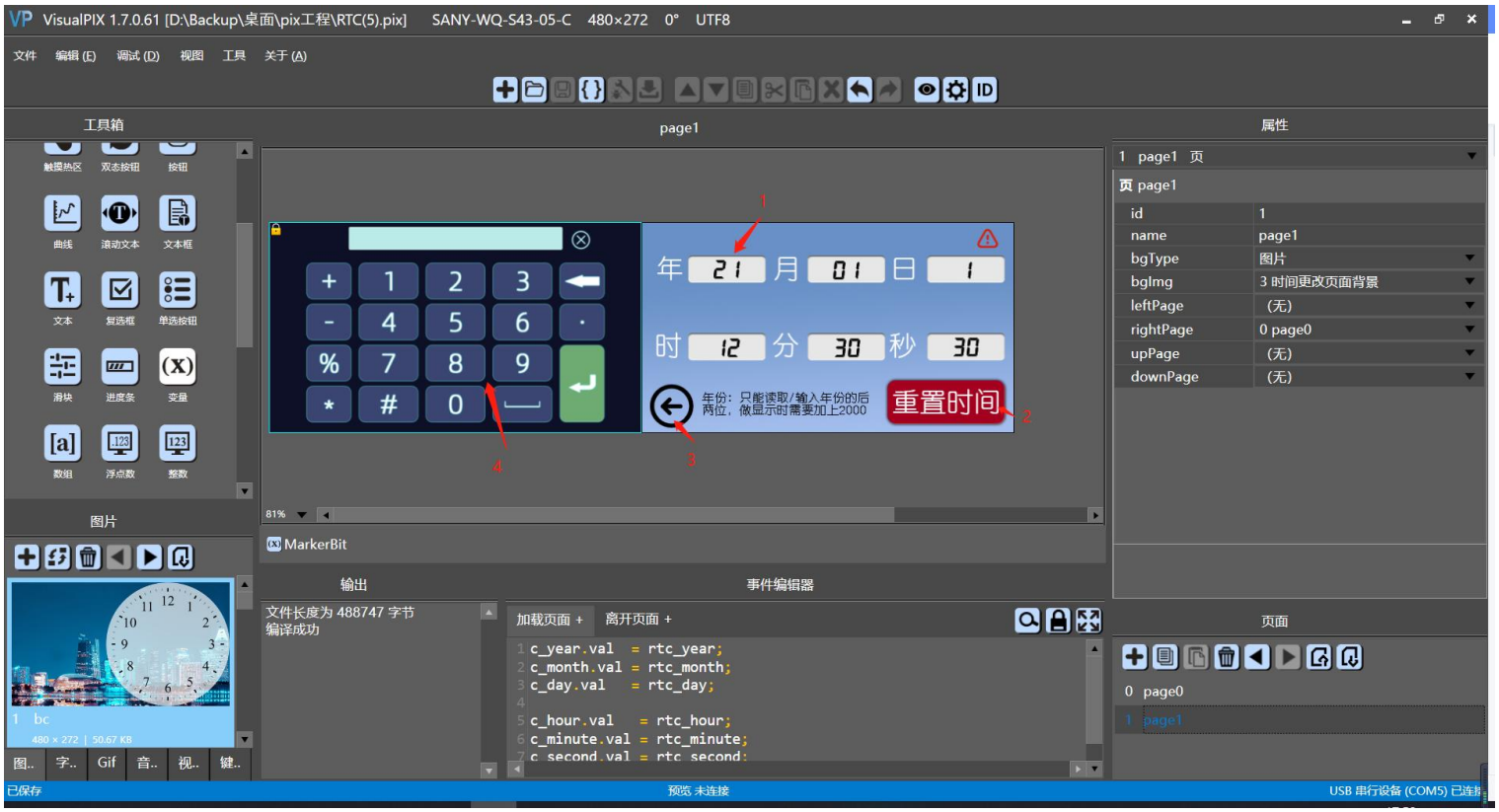
使用串口自带的 RTC 功能，显示时间，设置时间。

2.页面设计



(1)整数控件，显示年月日时分秒
(2)指针控件，显示时间
(3)按钮，翻页到 page1
(4)定时器控件，间隔 1 秒去读取 RTC 值，并赋值给控件显示
选中控件，可以在右侧属性栏查看控件的各个属性和事件代码。详细属性说明可参考 第四章 控件的介绍和用法。

Page1



1、整数控件，显示年月日时分秒

2、按钮，设置 RTC 参数，返回 page0

3、按钮，返回 page0

4、键盘控件，用于设置时间

选中控件，可以在右侧属性栏查看控件的各个属性和事件代码。详细属性说明可参考 第四章 控件的介绍和用法。

3.串口屏脚本处理

Page0 的定时器 RTCtimer 用于间隔 1 秒获取 RTC 的值，并赋值给控件显示。实际使用时，可能只需要简单的显示时间数字即可，这里为了演示，通过 RTC 时间值，计算指针的角度，实现表盘显示时间。

选中定时器控件，可以看到脚本代码如下：

```
int Week;



//获取 RTC 时间

hour.val  = rtc_hour;

minute.val = rtc_minute;

second.val = rtc_second;

year.val  = 2000 + rtc_year; //年份偏移是 2000，显示时需要加上 2000

month.val = rtc_month;

date.val  = rtc_day;
```

```
Week =rtc_week;



//根据时间，计算指针的位置
if(hour.val == 0 || hour.val == 12)
{
   Phour.angle = 0 + 90 + minute.val/10*6-6;
}else if(hour.val == 1 || hour.val == 13)
{
   Phour.angle = 30 + 90 + minute.val/10*6-6;
}else if(hour.val == 2 || hour.val == 14)
{
   Phour.angle = 60 + 90 + minute.val/10*6-6;
}else if(hour.val == 3 || hour.val == 15)
{
   Phour.angle = 90 + 90 + minute.val/10*6-6;
}else if(hour.val == 4 || hour.val == 16)
{
   Phour.angle = 120 + 90 + minute.val/10*6-6;
}else if(hour.val == 5 || hour.val == 17)
{
   Phour.angle = 150 + 90 + minute.val/10*6-6;
}else if(hour.val == 6 || hour.val == 18)
{
   Phour.angle = 180 + 90 + minute.val/10*6-6;
}else if(hour.val == 7 || hour.val == 19)
{
   Phour.angle = 210 + 90 + minute.val/10*6-6;
```

```
}else if(hour.val == 8 || hour.val == 20)
{
   Phour.angle = 240 + 90 + minute.val/10*6-6;
}else if(hour.val == 9 || hour.val == 21)
{
   Phour.angle = 270 + 90 + minute.val/10*6-6;
}else if(hour.val == 10 || hour.val == 22)
{
   Phour.angle = 300 + 90 + minute.val/10*6-6;
}else if(hour.val == 11 || hour.val == 23)
{
   Phour.angle = 330 + 90 + minute.val/10*6-6;
}
//时针的角度计算，角度 = 整点角度 +90 度偏移 + 分钟数在一小时刻度中的细分

Pminute.angle = minute.val*6+90;
Psecond.angle = second.val*6+90;
//分针和秒针的计算



//显示星期
if(Week == 1)
{
   week.txt = "星期一";
}else if(Week == 2)
{
   week.txt = "星期二";
}else if(Week == 3)
```

```
    {
      week.txt = "星期三";
    }else if(Week == 4)
    {
      week.txt = "星期四";
    }else if(Week == 5)
    {
      week.txt = "星期五";
    }else if(Week == 6)
    {
      week.txt = "星期六";
    }else if(Week == 0)
    {
      week.txt = "星期日";
    }
```

Page2 的键盘控件，用于手动输入时间，输入完成事件中，会将键盘的输入赋值给整数控件。选择键盘控件，可以看到脚本代码。

```
if(MarkerBit.val == 1)      //年
{
  c_year.val = stringToInt(keyboard1.txt);
}else if(MarkerBit.val == 2)      //月
{
  c_month.val = stringToInt(keyboard1.txt);
}else if(MarkerBit.val == 3)      //日
{
```

```
   c_day.val = stringToInt(keyboard1.txt);
}else if(MarkerBit.val == 4)      //时
{
   c_hour.val = stringToInt(keyboard1.txt);
}else if(MarkerBit.val == 5)      //分
{
   c_minute.val = stringToInt(keyboard1.txt);
}else if(MarkerBit.val == 6)      //分
{
   c_second.val = stringToInt(keyboard1.txt);
}


keyboard1.txt = "";


keyboard1.x = -480;
keyboard1.y = 0;
```

Page2 的重置时间按钮，主要是设置 RTC 的值，脚本代码如下

```
rtc_year  = c_year.val;     //RTC 只能读出后两位数，设置也只能设置后面两位
rtc_month = c_month.val;
rtc_day   = c_day.val;


rtc_hour   = c_hour.val;
rtc_minute = c_minute.val;
rtc_second = c_second.val;


showPage(0);
```

说明：RTC 可以读取和设置 年、月、日、时、分、秒。星期不用设置，设置好年月日后，系统会自己调整是星期几。年 rtc_year 的范围是 0~99，显示的时候需要加上 2000 才是正确的年份。

4.下载验证

编译成功后，点击下载按钮，选择正确的端口号和波特率，下载到串口屏。可以看到如下界面

点击 set 按钮,切换到 page1。点击数字，可以手动设置时间

修改 RTC 时间也能通过串口修改。指令格式可以串口屏系统指令 sset，系统指令用法可以参考实验 1。也可以通过自定义协议，在脚本代码里修改 RTC 系统变量的值，自定义协议用法可以参考实验 2。

实验 10　物理按键实验

1. 实验目的

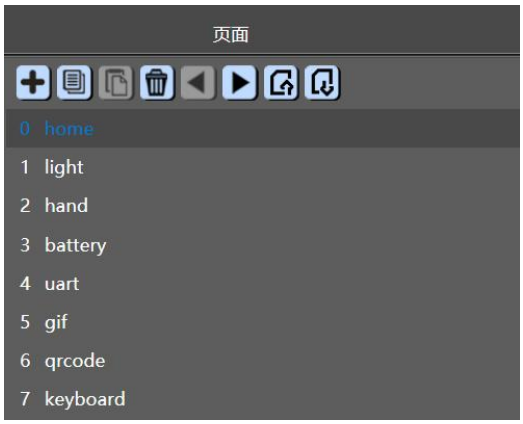
本实验演示了，不用屏幕自带的触摸板时，使用外接的串口按键板来控制串口屏的显示。如使用按键板的上、下、左、右，确认键实现切页、参数配置、参数输入等功能。

本实验配套的例程有两个工程，【物理按键_三个键值.pix】演示了按键板使用左、右和确认三个按键的实现方式；【物理按键_五个键值.pix】演示了按键板使用左、右、确认、上、下五个按键的实现方式，本实验以【物理按键_三个键值.pix】详细说明。本实验的这种物理按键实现方式比较灵活，弄清楚本实验的实现方式后，用户可根据这种方式自行调整以达到实现用户自己项目的预期效果。

本实验基于 2.4 寸串口屏，若要验证其他尺寸的串口屏，可在 VP 软件里修改项目型号（文件->编辑项目）。

2. 页面设计

本实验为演示物理按键的控制功能，一共有 8 个页面可供切换，如下：



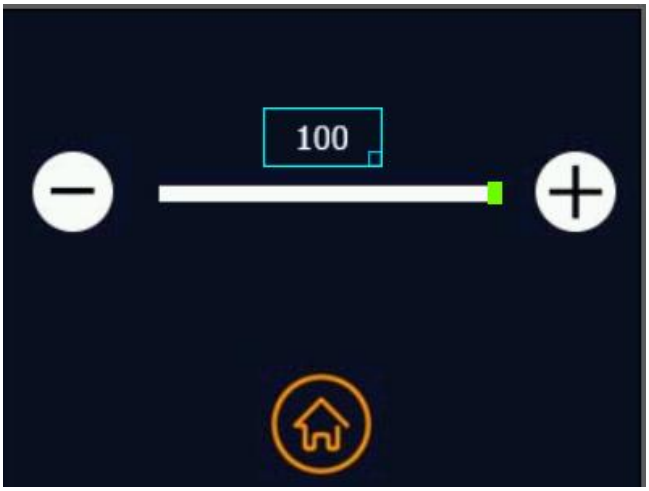
(1) home：主页面。一共 6 个按钮，点击每个按钮可切换到相应的页面。

(2) light:亮度控制页面。

3 个按钮，”-”按钮(控制系统亮度-10) ，”+”按钮(控制系统亮度+10)，“回到主页面”按钮(点击时返回主界面)。

1 个整数控件(显示具体的亮度值)。

1 个滑块控件(用滑块的位置来指示亮度值)。



(3) hand:汽车速度指针页面。

3 个按钮，”打开”按钮(启动定时器 2，指针开始顺时针转动)，”关闭”按钮(指针指向零刻度)，“回到主页面”按钮(点击时返回主界面)。

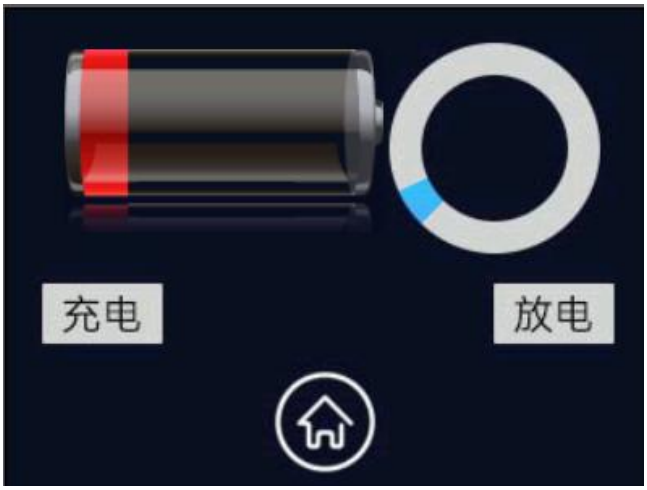
3 个指针控件，分别用于指示转速、速度、油表。

(4) battery:电池充放电模拟页面。

3 个按钮，”充电”按钮(启动定时器，电池和环形充电池的电量值增长)，”放电”按钮(启动定时器，电池和环形充电池的电量值减少)，“回到主页面”按钮(点击时返回主界面)。

1 个进度条控件，指示电池电量。

1 个环形进度条控件，指示环形充电池电量。



(5) uart：串口发送页面。

3 个按钮，”上发送”按钮(发送整数)，”下发送”按钮(发送浮点数)，“回到主页面”按钮(点击时返回主界面)。

1 个整数控件。

1 个浮点数控件。

(6) gif:动图显示页面。

3 个按钮，”播放”按钮(播放动图)，”暂停”按钮(暂停动图)，“回到主页面”按钮(点击时返回主界面)。

1 个 GIF 控件，显示动图。

(7) qrcode:二维码显示页面。

3 个按钮，”支付宝”按钮(设置二维码为 zhifubao)，”微信”按钮(设置二维码为 weixin)，“回到主页面”按钮(点击时返回主界面)。

1 个二维码控件。

(8) keyboard:键盘页面。

15 个按钮，0~9 和小数点为普通功能按钮，”←”为退格功能，”清除”为清空输入框，”退出”为退出键盘，”确认”按键将输入框的内容赋值到指定控件。

1 个文本控件，表现为输入框。

3. 串口屏脚本处理

工程【物理按键_三个键值.pix】一共需要处理三个键值 右(AA 01 BB),左(AA 02 BB),确认(AA 03 BB)。
工程【物理按键_五个键值.pix】一共需要处理五个键值 右(AA 01 BB),左(AA 02 BB),确认(AA 03 BB),下(AA 04 BB),上(AA 05 BB)。

脚本代码主要在协议解析器控件的数据接收事件里，选中 home 页面的协议解析器控件 proc，可以看到处理的脚本。
【物理按键_三个键值.pix】如下：

```
/*
接收处理按键板发过来的键值，一共三个键值 右(AA 01 BB),左(AA 02 BB),确认(AA 03 BB)
cursor 变量表示当前光标的位置，通过判断光标的位置，来设置按钮的背景图片，实现左右选中的效果。
注意:跨页访问控件属性时，需要指明页名称，并将要访问的控件设置成全局。
*/

if((proc.rxBuf[0] == 0xAA)&&(proc.rxBuf[2] == 0xBB)) //检测键值 帧头 帧尾
{
  if(sys_pid == home.id) //当前处于 home 页面
  {

    if((proc.rxBuf[1] == 1)||(proc.rxBuf[1] == 2))
    {
        //根据旧光标位置，设置按钮的背景图片 为 常态背景图片
        if(cursor.val == 1)    button1.bgImg = user.image.Main;
        else if(cursor.val == 2)button2.bgImg = user.image.Main;
        else if(cursor.val == 3)button3.bgImg = user.image.Main;
        else if(cursor.val == 4)button4.bgImg = user.image.Main;
        else if(cursor.val == 5)button5.bgImg = user.image.Main;
        else if(cursor.val == 6)button6.bgImg = user.image.Main;

        //根据键值改变光标的位置
        if(proc.rxBuf[1]== 1) //右
        {
```

```
            if((++cursor.val) > 6)cursor.val = 1;
        }
        else if(proc.rxBuf[1]== 2)  //左
        {
            if((--cursor.val) < 1)cursor.val = 6;
        }

        //根据新光标的位置，设置按钮的背景图片为深色类型，实现选中的效果
        if(cursor.val == 1)button1.bgImg = user.image.Main2;
        else if(cursor.val == 2)button2.bgImg = user.image.Main2;
        else if(cursor.val == 3)button3.bgImg = user.image.Main2;
        else if(cursor.val == 4)button4.bgImg = user.image.Main2;
        else if(cursor.val == 5)button5.bgImg = user.image.Main2;
        else if(cursor.val == 6)button6.bgImg = user.image.Main2;
    }
    else if(proc.rxBuf[1] == 3)
    {
        //确认键 执行按钮的点击事件
        if(cursor.val == 1)click(button1.x,button1.y);
        else if(cursor.val == 2)click(button2.x,button2.y);
        else if(cursor.val == 3)click(button3.x,button3.y);
        else if(cursor.val == 4)click(button4.x,button4.y);
        else if(cursor.val == 5)click(button5.x,button5.y);
        else if(cursor.val == 6)click(button6.x,button6.y);
    }
}
else if(sys_pid == light.id)
{
    if((proc.rxBuf[1] == 1)||(proc.rxBuf[1] == 2))
    {
        //根据旧光标位置，设置按钮的背景图片 为 常态背景图片
        if(cursor.val == 1)    light.button1.bgImg = user.image.light;
        else if(cursor.val == 2)light.button2.bgImg = user.image.light;
        else if(cursor.val == 3)light.button3.bgImg = user.image.light;

        //根据键值改变光标的位置
        if(proc.rxBuf[1]== 1)  //右
        {
            if((++cursor.val) > 3)cursor.val = 1;
        }
        else if(proc.rxBuf[1]== 2)  //左
        {
            if((--cursor.val) < 1)cursor.val = 3;
        }

        //根据新光标的位置，设置按钮的背景图片为深色类型，实现选中的效果
        if(cursor.val == 1)light.button1.bgImg = user.image.light2;
        else if(cursor.val == 2)light.button2.bgImg = user.image.light2;
        else if(cursor.val == 3)light.button3.bgImg = user.image.light2;
```

```
  }
  else if(proc.rxBuf[1] == 3)
  {
      //确认键 执行按钮的点击事件
      if(cursor.val == 1)click(light.button1.x,light.button1.y);
      else if(cursor.val == 2)click(light.button2.x,light.button2.y);
      else if(cursor.val == 3)click(light.button3.x,light.button3.y);
  }


}


else if(sys_pid == hand.id)
{
  if((proc.rxBuf[1] == 1)||(proc.rxBuf[1] == 2))
  {
      //根据旧光标位置，设置按钮的背景图片 为 常态背景图片
      if(cursor.val == 1)    hand.button1.bgImg = user.image.hand;
      else if(cursor.val == 2)hand.button2.bgImg = user.image.hand;
      else if(cursor.val == 3)hand.button3.bgImg = user.image.hand;

      //根据键值改变光标的位置
      if(proc.rxBuf[1]== 1)  //右
      {
        if((++cursor.val) > 3)cursor.val = 1;
      }
      else if(proc.rxBuf[1]== 2)  //左
      {
        if((--cursor.val) < 1)cursor.val = 3;
      }

      //根据新光标的位置，设置按钮的背景图片为深色类型，实现选中的效果
      if(cursor.val == 1)hand.button1.bgImg = user.image.hand2;
      else if(cursor.val == 2)hand.button2.bgImg = user.image.hand2;
      else if(cursor.val == 3)hand.button3.bgImg = user.image.hand2;
  }
  else if(proc.rxBuf[1] == 3)
  {
      //确认键 执行按钮的点击事件
      if(cursor.val == 1)click(hand.button1.x,hand.button1.y);
      else if(cursor.val == 2)click(hand.button2.x,hand.button2.y);
      else if(cursor.val == 3)click(hand.button3.x,hand.button3.y);
  }
}


else if(sys_pid == battery.id)
{
```

```
  if((proc.rxBuf[1] == 1)||(proc.rxBuf[1] == 2))
  {
      //根据旧光标位置，设置按钮的背景图片 为 常态背景图片
      if(cursor.val == 1)    battery.button1.bgImg = user.image.cell;
      else if(cursor.val == 2)battery.button2.bgImg = user.image.cell;
      else if(cursor.val == 3)battery.button3.bgImg = user.image.cell;

      //根据键值改变光标的位置
      if(proc.rxBuf[1]== 1)  //右
      {
        if((++cursor.val) > 3)cursor.val = 1;
      }
      else if(proc.rxBuf[1]== 2)  //左
      {
        if((--cursor.val) < 1)cursor.val = 3;
      }

      //根据新光标的位置，设置按钮的背景图片为深色类型，实现选中的效果
      if(cursor.val == 1)battery.button1.bgImg = user.image.cell2;
      else if(cursor.val == 2)battery.button2.bgImg = user.image.cell2;
      else if(cursor.val == 3)battery.button3.bgImg = user.image.cell2;
  }
  else if(proc.rxBuf[1] == 3)
  {
      //确认键 执行按钮的点击事件
      if(cursor.val == 1)click(battery.button1.x,battery.button1.y);
      else if(cursor.val == 2)click(battery.button2.x,battery.button2.y);
      else if(cursor.val == 3)click(battery.button3.x,battery.button3.y);
  }
}



else if(sys_pid == uart.id)
{
  if((proc.rxBuf[1] == 1)||(proc.rxBuf[1] == 2))
  {
      //根据旧光标位置，设置按钮的背景图片 为 常态背景图片
      if(cursor.val == 1)    uart.num1.bgImg = user.image.uart;
      else if(cursor.val == 2)uart.button2.bgImg = user.image.uart;
      else if(cursor.val == 3)uart.numf3.bgImg = user.image.uart;
      else if(cursor.val == 4)uart.button4.bgImg = user.image.uart;
      else if(cursor.val == 5)uart.button5.bgImg = user.image.uart;

      //根据键值改变光标的位置
      if(proc.rxBuf[1]== 1)  //右
      {
        if((++cursor.val) > 5)cursor.val = 1;
      }
```

```
        else if(proc.rxBuf[1]== 2)  //左
        {
           if((--cursor.val) < 1)cursor.val = 5;
        }

        //根据新光标的位置，设置按钮的背景图片为深色类型，实现选中的效果
        if(cursor.val == 1)uart.num1.bgImg = user.image.uart2;
        else if(cursor.val == 2)uart.button2.bgImg = user.image.uart2;
        else if(cursor.val == 3)uart.numf3.bgImg = user.image.uart2;
        else if(cursor.val == 4)uart.button4.bgImg = user.image.uart2;
        else if(cursor.val == 5)uart.button5.bgImg = user.image.uart2;
    }
    else if(proc.rxBuf[1] == 3)
    {
        //确认键 执行按钮的点击事件
        if(cursor.val == 1)click(uart.num1.x,uart.num1.y);
        else if(cursor.val == 2)click(uart.button2.x,uart.button2.y);
        else if(cursor.val == 3)click(uart.numf3.x,uart.numf3.y);
        else if(cursor.val == 4)click(uart.button4.x,uart.button4.y);
        else if(cursor.val == 5)click(uart.button5.x,uart.button5.y);
    }
}



else if(sys_pid == gif.id)
{
    if((proc.rxBuf[1] == 1)||(proc.rxBuf[1] == 2))
    {
        //根据旧光标位置，设置按钮的背景图片 为 常态背景图片
        if(cursor.val == 1)    gif.button1.bgImg = user.image.GIF;
        else if(cursor.val == 2)gif.button2.bgImg = user.image.GIF;
        else if(cursor.val == 3)gif.button3.bgImg = user.image.GIF;

        //根据键值改变光标的位置
        if(proc.rxBuf[1]== 1)  //右
        {
           if((++cursor.val) > 3)cursor.val = 1;
        }
        else if(proc.rxBuf[1]== 2)  //左
        {
           if((--cursor.val) < 1)cursor.val = 3;
        }

        //根据新光标的位置，设置按钮的背景图片为深色类型，实现选中的效果
        if(cursor.val == 1)gif.button1.bgImg = user.image.GIF2;
        else if(cursor.val == 2)gif.button2.bgImg = user.image.GIF2;
        else if(cursor.val == 3)gif.button3.bgImg = user.image.GIF2;
    }
```

```
  else if(proc.rxBuf[1] == 3)
  {
      //确认键 执行按钮的点击事件
      if(cursor.val == 1)click(gif.button1.x,gif.button1.y);
      else if(cursor.val == 2)click(gif.button2.x,gif.button2.y);
      else if(cursor.val == 3)click(gif.button3.x,gif.button3.y);
  }
}



else if(sys_pid == qrcode.id)
{
  if((proc.rxBuf[1] == 1)||(proc.rxBuf[1] == 2))
  {
      //根据旧光标位置，设置按钮的背景图片 为 常态背景图片
      if(cursor.val == 1)qrcode.button1.bgImg = user.image.qrcode;
      else if(cursor.val == 2)qrcode.button2.bgImg = user.image.qrcode;
      else if(cursor.val == 3)qrcode.button3.bgImg = user.image.qrcode;

      //根据键值改变光标的位置
      if(proc.rxBuf[1]== 1)  //右
      {
        if((++cursor.val) > 3)cursor.val = 1;
      }
      else if(proc.rxBuf[1]== 2)  //左
      {
        if((--cursor.val) < 1)cursor.val = 3;
      }

      //根据新光标的位置，设置按钮的背景图片为深色类型，实现选中的效果
      if(cursor.val == 1)    qrcode.button1.bgImg = user.image.qrcode2;
      else if(cursor.val == 2)qrcode.button2.bgImg = user.image.qrcode2;
      else if(cursor.val == 3)qrcode.button3.bgImg = user.image.qrcode2;
  }
  else if(proc.rxBuf[1] == 3)
  {
      //确认键 执行按钮的点击事件
      if(cursor.val == 1)click(qrcode.button1.x,qrcode.button1.y);
      else if(cursor.val == 2)click(qrcode.button2.x,qrcode.button2.y);
      else if(cursor.val == 3)click(qrcode.button3.x,qrcode.button3.y);
  }
}



else if(sys_pid == keyboard.id)
{
  if((proc.rxBuf[1] == 1)||(proc.rxBuf[1] == 2))
```

```
{
    //根据旧光标位置，设置按钮的背景图片 为 常态背景图片
    if(cursor.val == 1)    keyboard.button1.bgImg = user.image.key;
    else if(cursor.val == 2)keyboard.button2.bgImg = user.image.key;
    else if(cursor.val == 3)keyboard.button3.bgImg = user.image.key;
    else if(cursor.val == 4)keyboard.button4.bgImg = user.image.key;
    else if(cursor.val == 5)keyboard.button5.bgImg = user.image.key;
    else if(cursor.val == 6)keyboard.button6.bgImg = user.image.key;
    else if(cursor.val == 7)keyboard.button7.bgImg = user.image.key;
    else if(cursor.val == 8)keyboard.button8.bgImg = user.image.key;
    else if(cursor.val == 9)keyboard.button9.bgImg = user.image.key;
    else if(cursor.val == 10)keyboard.button10.bgImg = user.image.key;
    else if(cursor.val == 11)keyboard.button11.bgImg = user.image.key;
    else if(cursor.val == 12)keyboard.button12.bgImg = user.image.key;
    else if(cursor.val == 13)keyboard.button13.bgImg = user.image.key;
    else if(cursor.val == 14)keyboard.button14.bgImg = user.image.key;
    else if(cursor.val == 15)keyboard.button15.bgImg = user.image.key;

    //根据键值改变光标的位置
    if(proc.rxBuf[1]== 1)  //右
    {
      if((++cursor.val) > 15)cursor.val = 1;
    }
    else if(proc.rxBuf[1]== 2)  //左
    {
      if((--cursor.val) < 1)cursor.val = 15;
    }

    //根据新光标的位置，设置按钮的背景图片为深色类型，实现选中的效果
    if(cursor.val == 1)keyboard.button1.bgImg = user.image.key2;
    else if(cursor.val == 2)keyboard.button2.bgImg = user.image.key2;
    else if(cursor.val == 3)keyboard.button3.bgImg = user.image.key2;
    else if(cursor.val == 4)keyboard.button4.bgImg = user.image.key2;
    else if(cursor.val == 5)keyboard.button5.bgImg = user.image.key2;
    else if(cursor.val == 6)keyboard.button6.bgImg = user.image.key2;
    else if(cursor.val == 7)keyboard.button7.bgImg = user.image.key2;
    else if(cursor.val == 8)keyboard.button8.bgImg = user.image.key2;
    else if(cursor.val == 9)keyboard.button9.bgImg = user.image.key2;
    else if(cursor.val == 10)keyboard.button10.bgImg = user.image.key2;
    else if(cursor.val == 11)keyboard.button11.bgImg = user.image.key2;
    else if(cursor.val == 12)keyboard.button12.bgImg = user.image.key2;
    else if(cursor.val == 13)keyboard.button13.bgImg = user.image.key2;
    else if(cursor.val == 14)keyboard.button14.bgImg = user.image.key2;
    else if(cursor.val == 15)keyboard.button15.bgImg = user.image.key2;
}
else if(proc.rxBuf[1] == 3)
{
    //确认键 执行按钮的点击事件
    if(cursor.val == 1)click(keyboard.button1.x,keyboard.button1.y);
```

```
        else if(cursor.val == 2)click(keyboard.button2.x,keyboard.button2.y);
        else if(cursor.val == 3)click(keyboard.button3.x,keyboard.button3.y);
        else if(cursor.val == 4)click(keyboard.button4.x,keyboard.button4.y);
        else if(cursor.val == 5)click(keyboard.button5.x,keyboard.button5.y);
        else if(cursor.val == 6)click(keyboard.button6.x,keyboard.button6.y);
        else if(cursor.val == 7)click(keyboard.button7.x,keyboard.button7.y);
        else if(cursor.val == 8)click(keyboard.button8.x,keyboard.button8.y);
        else if(cursor.val == 9)click(keyboard.button9.x,keyboard.button9.y);
        else if(cursor.val == 10)click(keyboard.button10.x,keyboard.button10.y);
        else if(cursor.val == 11)click(keyboard.button11.x,keyboard.button11.y);
        else if(cursor.val == 12)click(keyboard.button12.x,keyboard.button12.y);
    }
            }
```

【物理按键_三个键值.pix】处理 右(AA 01 BB),左(AA 02 BB)两个键值时，home 页面上的变量控件 cursor 用于指示光标的位置。通过判断光标的位置，来设置按钮的背景图片为深色类型，实现左右移动的效果。如 home 页面的光标位置有

0(不显示光标)，

1(光标在 button1 按钮上)

2(光标在 button2 按钮上)

3(光标在 button3 按钮上)

4(光标在 button4 按钮上)

5(光标在 button5 按钮上)

6(光标在 button6 按钮上)

每一页光标的值默认是 0，光标值随键值而变化，如：

home 页面

循环按右键(AA 01 BB)时，光标值的变化为

0→1→2→3→4→5→6→1→2→3→4→5→6→1→2→3→4→5→.....

循环按左键(AA 02 BB)时，光标值的变化为

0→6→5→4→3→2→1→6→5→4→3→2→1→6→5→4→3→2→.....

light 页面

循环按左键(AA 01 BB)时，光标值的变化为

0→1→2→3→1→2→3→1→2→3→1→2→.....

循环按左键(AA 02 BB)时，光标值的变化为

0→3→2→1→3→2→1→3→2→.....

【物理按键_五个键值.pix】与【物理按键_三个键值.pix】的差异主要是体现在光标值的变化上。

【物理按键_三个键值.pix】只处理左右两个键值，因此键值的变化比较简单，cursor 加或者减即可。

【物理按键_五个键值.pix】同时需要考虑左右两个键值，还要考虑上下两个键值。上下两个键值的变化不是简单的加减，需要自行定义，实验中每一页定义了两个数组 arrh，arrv。其中 arrh 用于保存左右键值变化的规律，arrv 用于保存上下键值变化的规律。数组的初始化，写在 home 页面的加载页面事件里，并用 startflag 变量标记，只需初始化一次即可。如后面页面的数组初始化如下：

```
 //home 页
arrh.valbs[0] = 0;
arrh.valbs[1] = 1;
arrh.valbs[2] = 2;
arrh.valbs[3] = 3;
arrh.valbs[4] = 4;
arrh.valbs[5] = 5;
```

```
  arrh.valbs[6] = 6;

  arrv.valbs[0] = 0;
  arrv.valbs[1] = 1;
  arrv.valbs[2] = 3;
  arrv.valbs[3] = 5;
  arrv.valbs[4] = 2;
  arrv.valbs[5] = 4;
  arrv.valbs[6] = 6;
```

每一页光标的值默认是 0，根据页面上的按钮，光标值随键值而变化，如：

home 页面

循环按右键(AA 01 BB)时，光标值的变化为

0→1→2→3→4→5→6→1→2→3→4→5→6→1→2→3→4→5→.....

循环按左键(AA 02 BB)时，光标值的变化为

0→6→5→4→3→2→1→6→5→4→3→2→1→6→5→4→3→2→.....

循环按下键(AA 01 BB)时，光标值的变化为

0→1→3→5→2→4→6→1→3→5→2→4→6→1→3→5→2→4→.....

循环按上键(AA 02 BB)时，光标值的变化为

0→6→4→2→5→3→1→6→4→2→5→3→1→6→4→2→5→3→.....

【物理按键_五个键值.pix】部分脚本代码如下

```
if((proc.rxBuf[0] == 0xAA)&&(proc.rxBuf[1] <= 5)&&(proc.rxBuf[2] == 0xBB))  //检测键值  帧头  帧尾
{
  if(sys_pid == home.id) //当前处于 home 页面
  {
    if(proc.rxBuf[1] != 3)
    {
        //根据旧光标位置，设置按钮的背景图片 为 常态背景图片
        if(cursor.val == 1)    button1.bgImg = user.image.Main;
        else if(cursor.val == 2)button2.bgImg = user.image.Main;
        else if(cursor.val == 3)button3.bgImg = user.image.Main;
        else if(cursor.val == 4)button4.bgImg = user.image.Main;
        else if(cursor.val == 5)button5.bgImg = user.image.Main;
        else if(cursor.val == 6)button6.bgImg = user.image.Main;

        //根据键值改变光标的位置
        get_new_cursor(); //自定义函数，工具->函数编辑器 里可以查看和修改


        //根据新光标的位置，设置按钮的背景图片为深色类型，实现选中的效果
        if(cursor.val == 1)button1.bgImg = user.image.Main2;
        else if(cursor.val == 2)button2.bgImg = user.image.Main2;
        else if(cursor.val == 3)button3.bgImg = user.image.Main2;
        else if(cursor.val == 4)button4.bgImg = user.image.Main2;
        else if(cursor.val == 5)button5.bgImg = user.image.Main2;
        else if(cursor.val == 6)button6.bgImg = user.image.Main2;
    }
    else
```

```
    {
        //确认键 执行按钮的点击事件
        if(cursor.val == 1)click(button1.x,button1.y);
        else if(cursor.val == 2)click(button2.x,button2.y);
        else if(cursor.val == 3)click(button3.x,button3.y);
        else if(cursor.val == 4)click(button4.x,button4.y);
        else if(cursor.val == 5)click(button5.x,button5.y);
        else if(cursor.val == 6)click(button6.x,button6.y);
    }
}
```

可以看到改变光标位置的时候，调用了一个函数 get_new_cursor()，这个函数是自定义的，可在工具->函数编辑器 里可以查看和修改，主要是处理五个键值时，光标位置的变化相对复杂一些，这时候自定义一个函数，可使脚本代码更加清晰。

get_new_cursor()功能大概是，根据页面和键值的类型，找到当前光标的位置；再根据 arrh 或者 arrv 数组中定义的变化规则，改变光标的位置。

处理 确认(AA 03 BB)键值时，根据光标的所在位置，调用 click 函数，执行对应按钮的脚本，实现切页、控制功能。

注意：访问其他页面的控件时，需要勾选访问的控件 global 属性，并在访问时加上页名称前缀，如 hand.button2.bgImg

本实验由于演示的页面比较多，为了使脚本代码更清晰，脚本要访问控件的名字务必定义清晰，增加代码的易读性，可按照下面的规则：

1、名称简短

2、最好能体现实际意义

3、相同属性的控件，最好带有一定顺序的编号

4. 物理按键板单片机程序

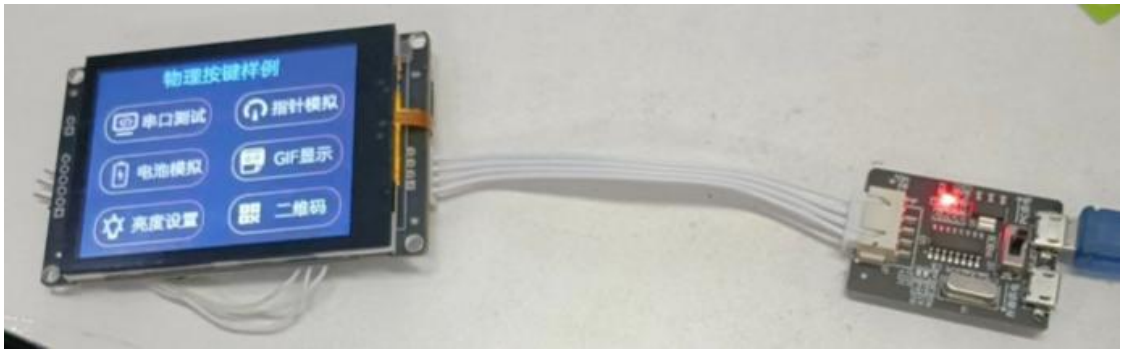
单片机的程序很简单，检测到按钮按下并发送键值即可。

```
char uart_buf[3] = {0xAA,0x00,0xBB};
int main(void)
{
    u8 key_value;

    Sys_SetRcc();                                    //设置系统使用内部时钟
    delay_init(64);                                  //延时函数初始化
    usart1_init();                                   //初始化串口
    KEY_Init();                                      //按键初始化
    while(1)
    {
        key_value = KEY_Scan(0);                     //得到键值
        if(key_value)   //上 7   左 8   右 6   下 11   中 5
        {
```

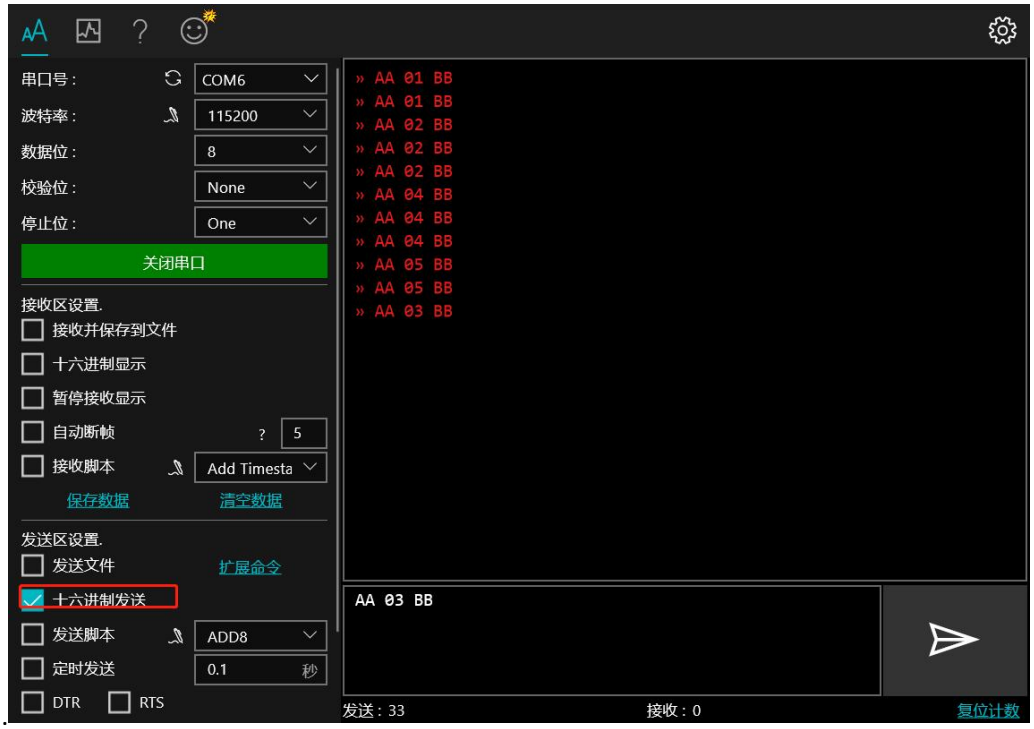
```
            if(key_value == J6_PRES)   //按键 右
            {
                uart_buf[1] = 0x01;
                USART_OUT(USART1, (uint8_t*)uart_buf,3);
            }
            else if(key_value == J8_PRES)   //按键 左
            {
                uart_buf[1] = 0x02;
                USART_OUT(USART1, (uint8_t*)uart_buf,3);
            }
            else if(key_value == J5_PRES)   //按键 OK
            {
                uart_buf[1] = 0x03;
                USART_OUT(USART1, (uint8_t*)uart_buf,3);
            }
            else if(key_value == J11_PRES)   //按键 下
            {
                uart_buf[1] = 0x04;
                USART_OUT(USART1, (uint8_t*)uart_buf,3);
            }
            else if(key_value == J7_PRES)   //按键 上
            {
                uart_buf[1] = 0x05;
                USART_OUT(USART1, (uint8_t*)uart_buf,3);
            }
        }
    }

}
```

5.下载验证

将【物理按键_三个键值.pix】或者【物理按键_五个键值.pix】编译后下载到 2.4 寸显示屏，即可看到如下界面：



(1)调试时先将屏的串口连接到电脑，用电脑端的串口助手模拟给屏发键值，确认屏能正确处理键值：



(2)再将 STM32 单片机的键值输出到电脑端的串口助手上，确认 STM32 单片机发出的键值是正确的。

(3)最后将 STM32 单片机的串口接到屏的串口上，可以验证到，上、下、左、右、确认键都是正常的

实验 11　扩展 IO 功能示例

1. 实验目的

本实验演示了扩展 IO 的相关功能及使用方法。扩展 IO 的功能有普通输入、普通输出、中断输入、AD 输入、PWM 输出和捕获输入的功能。

G 系列串口屏(3.5 寸及以下)支持 6 个扩展 IO，S 系列串口屏(4.3 寸及以上)最多可支持 25 个扩展 IO。

每个 IO 的功能不一样，项目设计的时候，需要提前了解到要用到的 IO 是否支持所需功能，比如如有的 IO 能支持 AD 输入，有的能支持 PWM 输出，具体的 IO 功能介绍参考第七章。

本实验基于 4.3 寸 480*272 分辨率的串口屏，若要验证其他尺寸的串口屏，可在 VP 软件里修改项目型号（文件->编辑项目），或者参考本实验重新设计工程。

2.页面设计

本实验一共 6 个页面，分页演示了不同 IO 功能的用法。

(1)home 页面，一共 5 个按钮，点击对应的按钮可切页到其他页面。

(2) GPIO 页面。

3 个 IO 控件，对应的引脚号是 io7,io8,io9

1 个定时器控件，定时 1 秒去读取 io8 引脚的电平状态
3 个文本控件，分别显示输出、输入、中断输入的 IO 引脚号
4 个整数控件，分别显示输出值、输入值、中断输入值和中断次数
1 个双态按钮控件，控制输入 IO 的电平，弹起是 1
1 个按钮控件，点击返回 home 页面

(3) AD 页面
1 个 ADC 控件，引脚号 io6
1 个定时器控件，定时 1 秒去读取 adc 控件的采样值
1 个浮点数控件，用于显示电压值
1 个按钮控件，点击返回 home 页面

(4) PWM 页面
1 个 PWM 控件，引脚号 io0
1 个键盘控件，用于输入频率和占空比
2 个整数控件，分别显示频率和占空比
1 个双态按钮控件，控制输出使能，按下使能，弹起不使能
1 个按钮控件，点击返回 home 页面

(5) CAPTURE 页面

1 个 Capture 控件，引脚号 io3

1 个定时器控件，定时 1 秒去读取 Capture 控件的频率、占空比和脉冲个数

3 个整数控件，分别显示频率、占空比、脉冲个数

1 个按钮控件，点击返回 home 页面

(6) ExUART 页面

1 个扩展协议解析器控件，默认引脚号 io1,io2

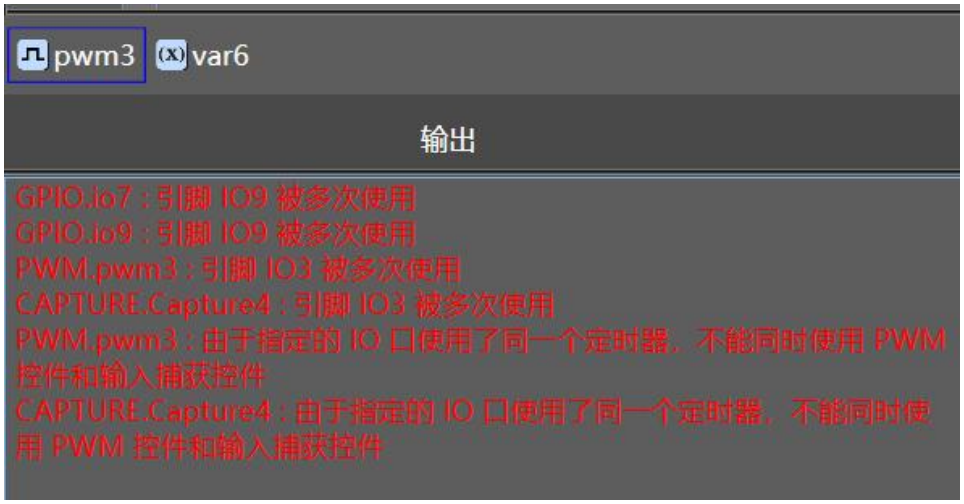
2 个文本控件，分别显示接收区和发送区的文本

1 个按钮控件，点击”发送”后，扩展串口将发送区的字符串发出去

1 个按钮控件，点击返回 home 页面

注意：在页面设计时，提前分配好 IO 的功能，如果同一个 IO 引脚被多次绑定，编译的时候将会报错。

错误提示类型举例：



3.串口屏脚本处理

本实验的脚本处理很简单，选中控件可以查看相关的脚本。IO 相关控件属性的读和写，列举几个关键的说明。

(1) GPIO 页面

输出电平控制

```
io7.val = 0;
io7.val = 1;
```

输入电平读取

```
io8num.val = io8.val; //定时 1 秒，读取 io8 的电平状态，0 或者 1
```

IO 中断电平读取和中断处理(num9 显示累加)

```
io9num.val = io9.val;
num9.val++;
```

(2) AD 页面

读取电压值

```
num1.val = adc.val; //定时 1 秒，读取 IO6 引脚上的模拟电压值
```

(3) PWM 页面

根据键盘输入值设置频率和占空比

```
int val;
val = stringToInt(keyboard4.txt); //将键盘的字符串转换成整数
if(var6.val == 0)
{
        if((val >=1)&&(val <= 1000000)) //频率范围 1Hz~1MHz
        {
                num_freq.val = val;
                pwm3.frequency = num_freq.val; //设置 PWM 输出频率
        }
}
else
```

```
{
        if((val >=0)&&(val <= 100)) //占空比范围 0~100
        {
                num_duty.val = stringToInt(keyboard4.txt);
                pwm3.dutyRatio = num_duty.val;          //设置 PWM 输出占空比
        }
}
keyboard4.x = -500; //隐藏键盘
```

设置 PWM 输出使能

```
pwm3.enable = 1;     //PWM 输出使能
tgbtn1.txt = "enable";
```

设置 PWM 输出不使能

```
pwm3.enable = 0;       //PWM 输出不使能
tgbtn1.txt = "disable";
```

(4) CAPTURE 页面

获取捕获参数

```
//定时器去读输入捕获控件的属性，频率、占空比、脉冲个数
num_freq.val = Capture4.frequency;
num_duty.val = Capture4.dutyRatio;
num_cnt.val    = Capture4.pulseCount;
```

(5) ExUART 页面

字符串发送

```
//扩展串口发送函数;
//更多的操作函数访问方式为，输入控件名称，然后输入'.', 即 exptl4.fun
exptl4.uartSend(text4.txt);
```

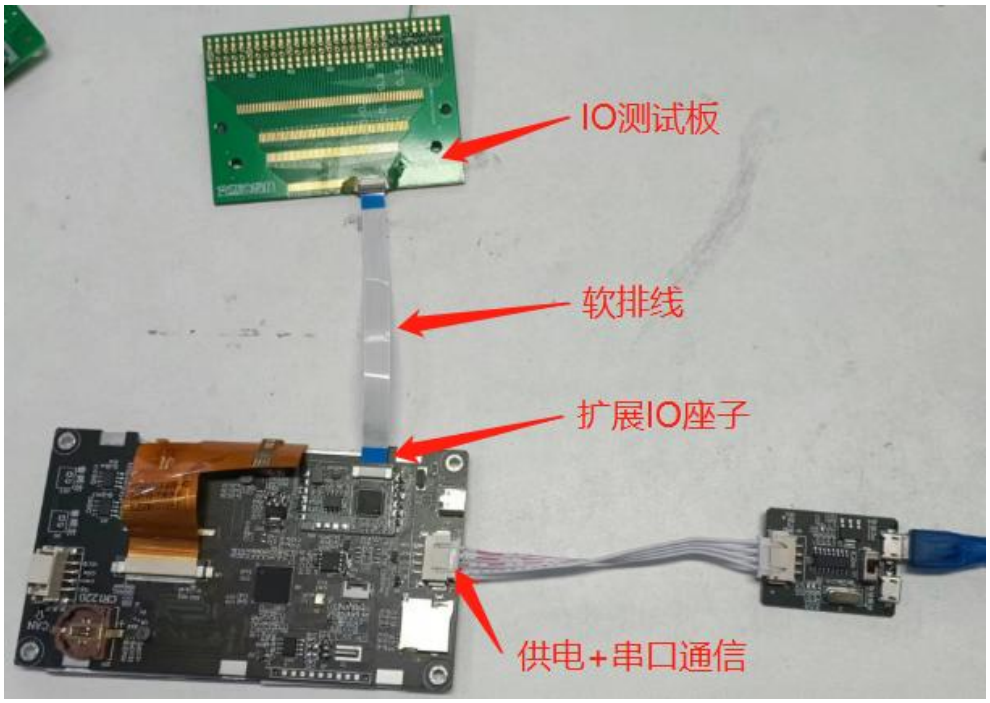
接收并显示

```
//将收到的串口数组，转换为字符串并赋值给文本显示
text3.txt = bytesToAscii(exptl4.rxBuf,0,exptl4.rxLen);
```

4.下载验证

将工程编译后下载到 4.3 寸带显示屏(带扩展 IO 模块)，即可看到如下界面：

为了测试扩展 IO 的功能，我们需要将扩展 IO 通过软排线接到 IO 测试板上，如下图：



用万用表、示波器检测 IO 测试板即可。

(1) GPIO 测试。

触摸控制 GPIO 页面双态按钮的状态，用万用表测试 io7 的输出电压变化，0V 或者 3.3V；

给 io8 接入 3.3V 或者 GND，观察显示屏上 io8 的电平值

给 io8 接入 3.3V 或者 GND，观察显示屏上 io9 的电平值和中断次数

(2) AD 测试。

给 i06 引脚接入 0~3.3V 模拟电压。观察显示屏上 AD 采用出来的电压

(3) PWM 测试。

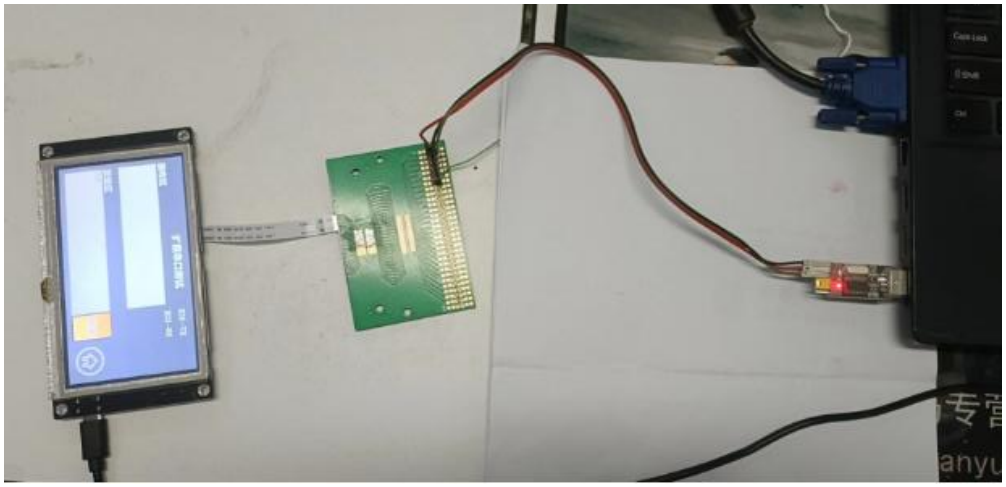
点击输出使能，设置 pwm 输出，可选修改频率和占空比，用过示波器检测 io0 引脚上的波形。

(4) CAPTURE 测试

将 io0 输出的 pwm 波形接入 io3，观察显示屏上捕获到的频率、占空比和脉冲个数

(5) ExUART 测试

将 io1 接入 TTL 工具的 rx，io2 接入 TTL 工具的 tx，然后将 TTL 工具接入电脑，用串口助手收发测试。测试接线如下：



实验 12　SD 卡操作实验

1. 实验目的

此实验将对 SD 卡存在的操作全部做出说明，包括 SD 卡写入/读取数据，SD 卡资源的操作。

此实验基于 4.3 寸 480*272 分辨率的串口屏，与屏幕通信方式无关。若要验证其他尺寸的串口屏，可在 VP 软件里修改项目型号（文件->编辑项目），或者参考本实验重新设计制作工程。

实验准备：一个读卡器，或者 SD 卡卡套；一张合适大小的 SD 卡，大小一般 8G 以内；文件系统常用 FAT32，若后续操作过程中无法正常读写 SD 卡，可尝试将卡格式化为 FAT32 文件系统；

2. 实验操作

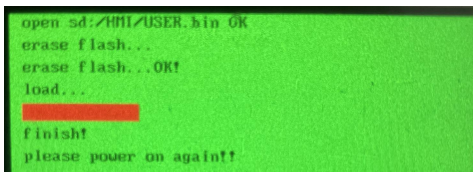
（1）SD 卡下载工程、更新固件

(1) 工程下载，操作步骤：

① VP 打开工程，工具菜单->项目设置，勾选‘生成二进制文件’，点击保存退出，重新编译工程后，工程所在的文件夹内，会生成一个 bin 格式的文件；

② 计算机上打开 SD 卡，在根目录内新建 HMI 文件夹，将第一步中生成的 bin 格式文件，拷贝到 HMI 文件夹内，并将 bin 文件改名为“**USER**”；

③ 断电串口屏，插入 SD 卡，重新上电后，屏幕会绿底黑字提示正在下载，当出现以下画面，则表示下载成功，成功后断电取下 SD 卡，重新上电，完成操作。



注意：生成 bin 文件的工程型号，注意与串口屏型号一致，否则会型号不匹配，导致不能下载。

(2) 固件下载，操作步骤：

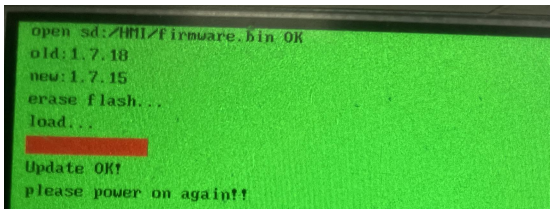
① 当需要对多个串口屏更新固件时，若使用 VP 上位机更新，效率受限，可用此方法快速批量更新固件；

② 准备好串口屏型号对应的固件；

③ 计算机上打开 SD 卡，在根目录内新建 HMI 文件夹，将准备好的固件，拷贝到 HMI 文件夹内，并将文件改名为“**firmware**”；

④ 断电串口屏，插入 SD 卡，重新上电后，屏幕会绿底黑字提示正在下载，当出现以下画面，则表示下载成功，

成功后断电取下 SD 卡，重新上电。



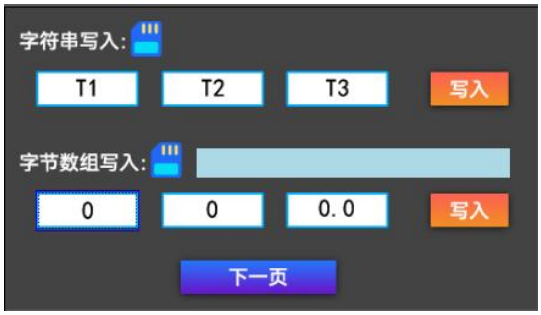
⑤ 下载固件成功重新上电后，屏幕会提示没有工程文件，此时需下载工程 pix 文件，工程下载后既可正常使用。

注意：用于工程下载的 HM I 文件夹和固件更新的 HMI 文件夹不可同时存在。

（2）SD 卡数据写入、读取

(1) 数据写入：

① 页面设计



② 字符串写入功能说明

字符串写入：SD 卡图标用于判断是否写入成功，若写入成功，则图标显示，写入失败则隐藏；

T1、T2、T3 为输入框，点击输入框，可用键盘对应输入字符串；

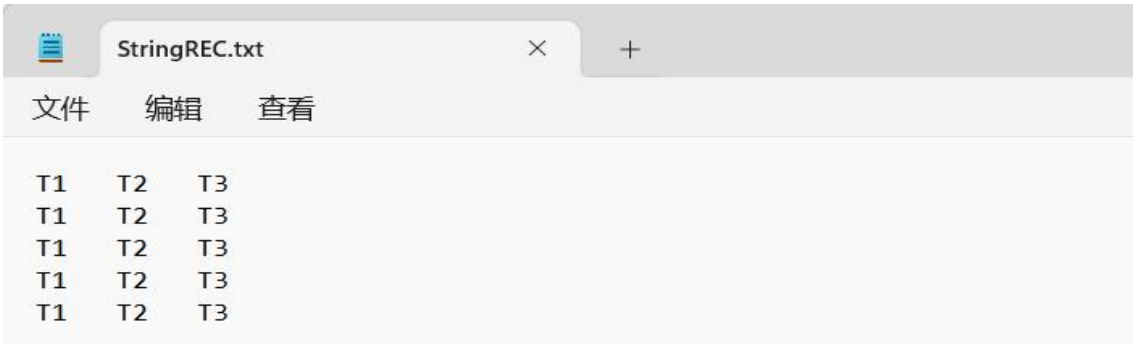
写入按钮，点击后将 T1、T2、T3 的字符串拼接起来并用函数写入 SD 卡；

写入按钮事件脚本：**串口屏支持将字符串写入 txt 格式和 csv 格式文件内**

- **写入 txt 文件内：**

```
/*****************************格式1：数据存储在txt格式文件内*****************************/

//将三个文本字符串拼接起来数据，数据之间用空格隔开一下
SDdate.txt = T1.txt + " "+ T2.txt + " "+ T3.txt;

//根据函数返回值来控制SD卡图标的显示和隐藏（返回0：写入成功    返回-1：写入失败）
if(fileWriteLine("StringREC.txt",SDdate.txt) == 0)//存入StringREC.txt文件中
{
    image38.visible=1;
}else
{
    image38.visible=0;
}
```

写入按钮部分脚本

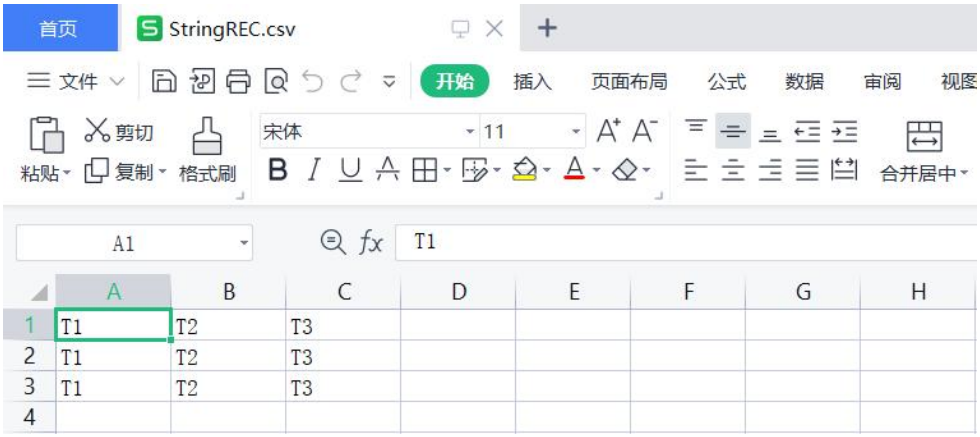


写入结果

- 写入 csv 文件内：数据之间必须用英文逗号(,)隔开

```
/*****************************格式2：数据存储在csv格式文件内*****************************/

//将三个文本字符串拼接起来数据，存储为CSV数据，数据之间必须用英文逗号(,)隔开
SDdate.txt = T1.txt + ","+ T2.txt + ","+ T3.txt;

//根据函数返回值来控制SD卡图标的显示和隐藏（返回0：写入成功    返回-1：写入失败）
if(fileWriteLine("StringREC.csv",SDdate.txt) == 0)//StringREC.csv文件中
{
    image38.visible=1;
}else
{
    image38.visible=0;
}
```

写入按钮部分脚本



写入结果

注：上位机模拟写入，写入的结果可在菜单栏：文件->打开 sd 卡目录 中查看。

③ **字节数组写入功能说明**

字节数组写入：SD 卡图标用于判断是否写入成功，若写入成功，则图标显示，写入失败则隐藏；

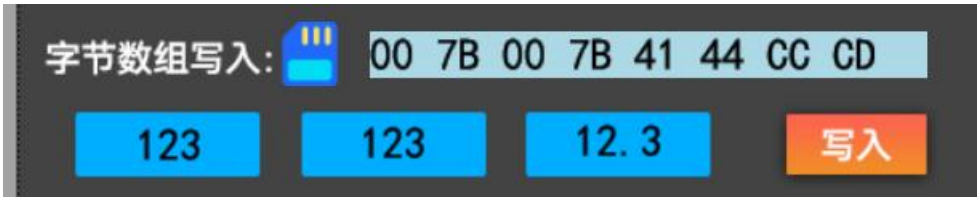
数据显示 log，用于显示转换完成的数组数据；

三个整数输入框，点击输入框，可用键盘对应输入整数和浮点数；

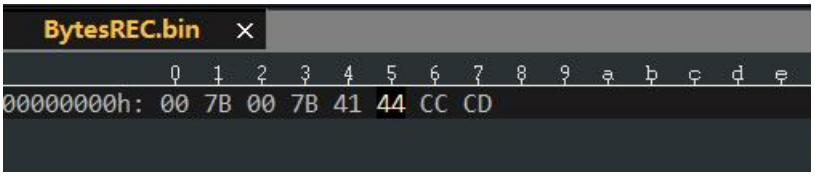
写入按钮，点击后将两个整数和浮点数转存入 arr 数组中，并用函数写入 SD 卡；

```
//按下写入按钮，将输入的数字转换拼接为数组形式，并写入SD卡内

byte arr[8];//定义一个数组，长度：2字节整数 + 2字节整数 + 4字节浮点数

//byte 0-1
arr[0] = (byte)((num1.val >> 8) & 0xff);//数据拼接
arr[1] = (byte)((num1.val >> 0) & 0xff);

//byte 2-3
arr[2] = (byte)((num2.val >> 8) & 0xff);//数据拼接
arr[3] = (byte)((num2.val >> 0) & 0xff);

//byte 4-7
floatToBytes(numf1.valf,arr,4,1);//浮点数转字节数组。
//4个参数:需要转换的浮点数 缓存转换结果的数组 起始位置 大小端

//根据函数返回值来控制SD卡图标的显示和隐藏（返回0：写入成功    返回-1：写入失败）
if(fileWriteBytes("BytesREC.bin",0,arr,0,8) == 0)//写入到BytesREC.bin文件中
{
    image39.visible=1;
}else
{
    image39.visible=0;
}

//利用log控件显示数组arr的数据,注意显示hex数据时，要勾选控件的hexMd属性
log9.clear();//清空
log9.addBytes(arr,0,8);//显示
```

写入按钮事件脚本



模拟写入数据



bin 文件写入结果

注：上位机模拟写入，写入的结果可在菜单栏：文件->打开 sd 卡目录中查看。
打开 bin 文件需要一些工具，例如 Notepad++、UltraEdit 等，可自行网站搜索下载。

(2) 数据读取：

① 页面设计

② **字符串读取功能说明**

字符串读取：点击读取按钮，将已写入的字符串文本全部读出来，用文本框控件显示；

- **读取 txt 文件：**

```
int value;
int i;

//读取文件的行数,函数参数为文件路径
value = fileReadLineCount("StringREC.txt");

textBox4.txt = "";//读出显示前，先清空显示区
for(i=0;i<value;i++)
{
    //利用函数读取每行所有的字符，每行后面添加一个换行，给文本框控件显示
    textBox4.txt += fileReadLine("StringREC.txt",i,20)+"\r\n";
    //参数为：文本路径，第几行，读取的字符数
}
```

读取按钮部分脚本

读取结果

- **读取 csv 文件：**

```
//读取文件的行数,函数参数为文件路径
value = fileReadLineCount("StringREC.csv");

textBox4.txt = "";//读出显示前，先清空显示区
for(i=0;i<value;i++)
{
    //利用函数读取每行所有的字符，每行后面添加一个换行，给文本框控件显示
    textBox4.txt += fileReadLine("StringREC.csv",i,20)+"\r\n";
    //参数为：文本路径，第几行，读取的字符数
}
```

读取按钮部分脚本

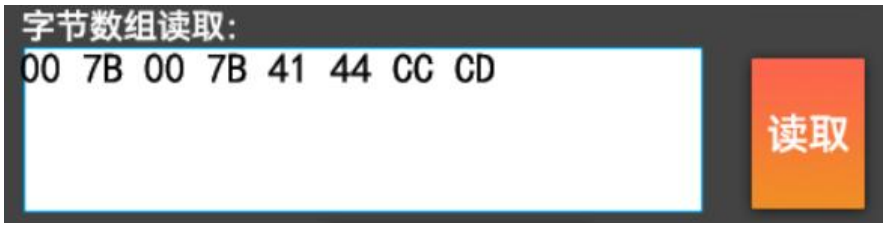
读取结果

③ 字节数组读取功能说明

字节数组读取：点击读取按钮，将已写入的 bin 文件数据全部读出来，用 log 控件显示；

```
byte arr[8];

//读取文件的字节数据
fileReadBytes("BytesREC.bin",0,arr,0,8);
//参数为：文件路径 起始位置 用于缓存所读数据的数组 数组的缓存起始位置 字节数

//利用log控件显示数组arr的数据,注意显示hex数据时，要勾选控件的hexMd属性
log6.clear();//清空
log6.addBytes(arr,0,8);//显示
```

读取按钮部分脚本

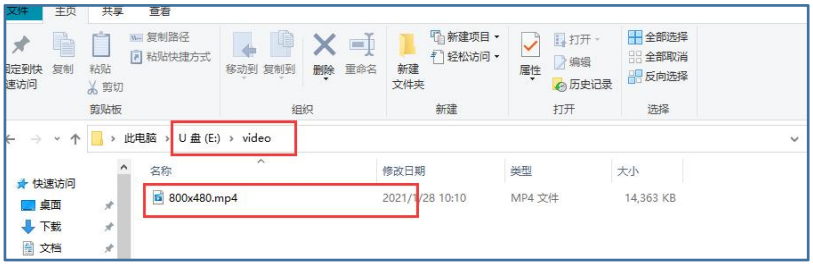


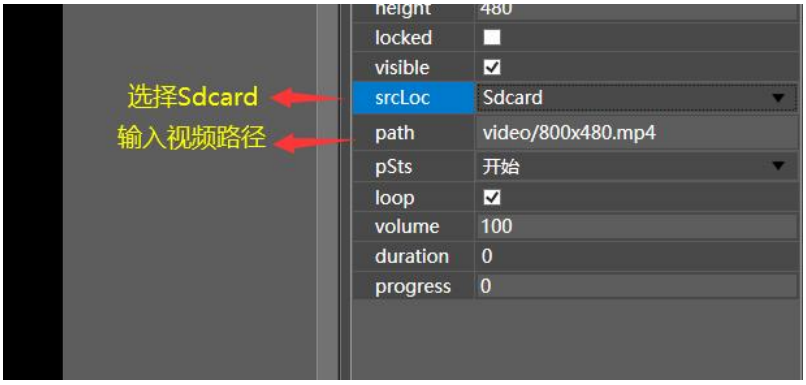
读取结果

（3）SD 卡存放资源素材

当串口屏自带 Flash 空间不足以存放更多资源素材时，SD 卡可用作为外扩存储空间使用。
能 SD 卡存储素材资源的控件：视频控件的视频、音频控件的音频、图片控件的图片。

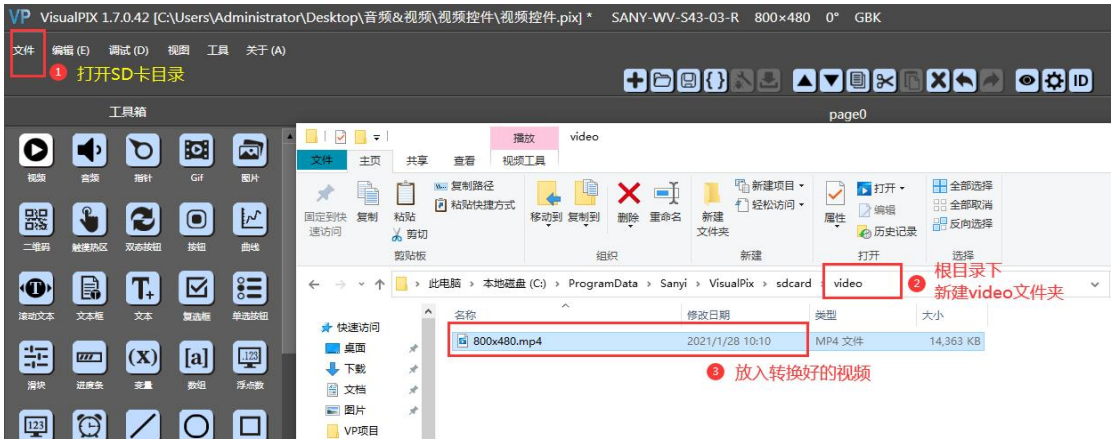
- **视频控件：**
  播放 Sdcard 中的视频时，在 SD 卡根目录下新建 video 文件夹，放入转换好的视频，名称改为非中文，并在控件属性栏选择 Sdcard，输入视频路径：





srcLoc：视频资源的所在空间，选择为外部 Sdcard；
path：视频资源的位置路径，此处解释为：放在 SD 卡内，video 文件夹下，名称为 800x480.mp4 的视频；

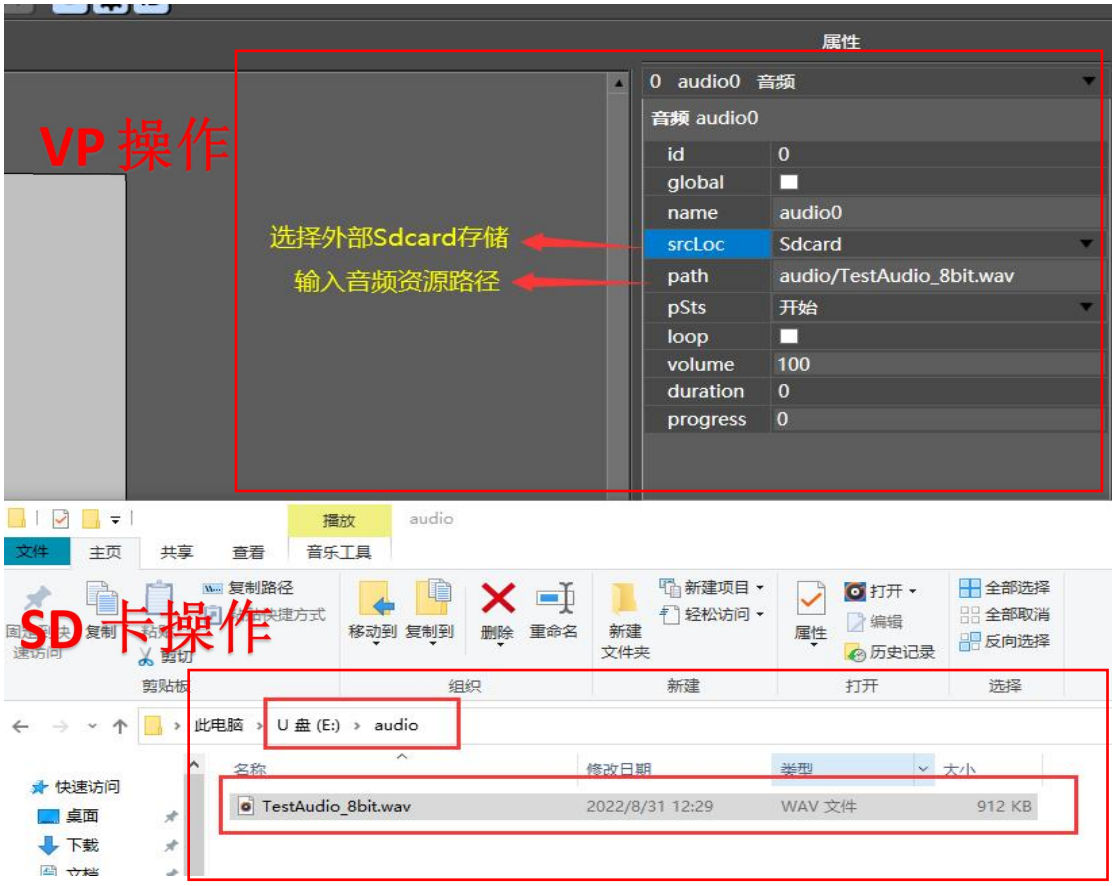
注：VP 上位机模拟 SD 卡存储播放视频时
在选择外部 Sdcard 存储资源时，VP 软件需要调试模拟，此时资源存储也要相应处理，将视频文件放入 VP 模拟 SD 卡的文件夹中。



关于视频控件更详细的使用介绍，请到官网下载‘视频控件使用教程’压缩包。

- 音频控件：

选择外部 Sdcard 存储音频资源时，在 SD 卡根目录中新建文件夹 audio，然后将转换好的音频放入，改好音频名称（路径只支持英文）；输入路径时注意路径字符输入正确，具体操作如下图：

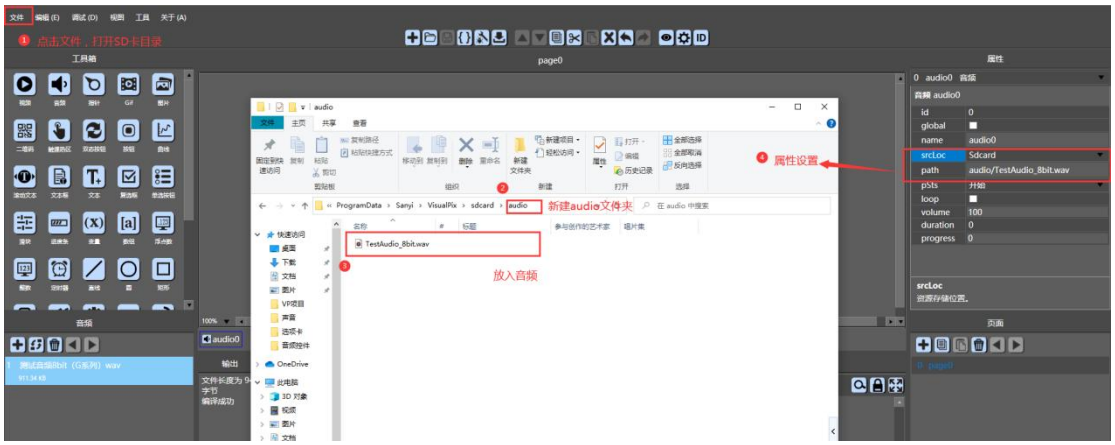


srcLoc：音频资源的所在空间，选择为外部 Sdcard；

path：音频资源的位置路径，此处解释为：放在 SD 卡内，audio 文件夹下，名称为 TestAudio_8bit.wav 的音频；

注：VP 上位机模拟 SD 卡存储播放音频时

在选择外部 Sdcard 存储资源时，VP 软件需要调试模拟，此时资源存储也要相应处 理，将音频文件放入 VP 模拟 SD 卡的文件夹中。

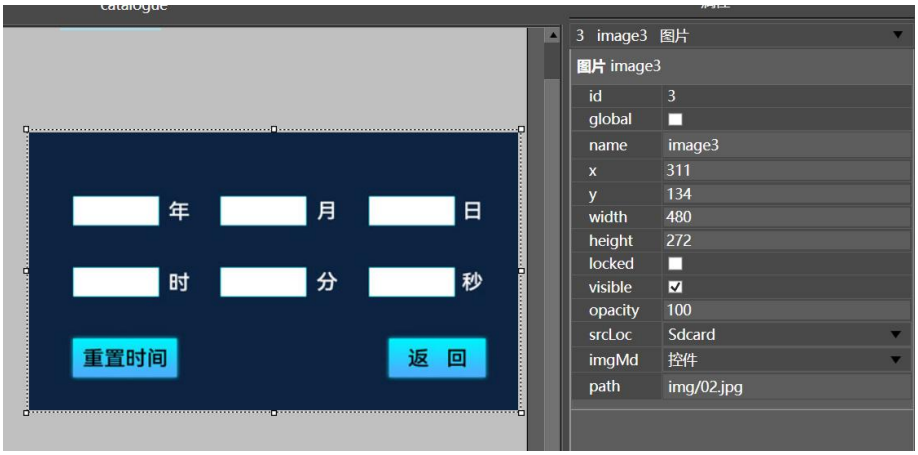


关于视频控件更详细的使用介绍，请到官网下载‘视频控件使用教程’压缩包。

- 图片控件：

① SD 卡根目录内新建 img 文件夹，将图片改名为非中文，放入 img 文件夹内；

② 拖入图片控件，设置属性，srcLoc 属性选择 Sdcard，path 属性填入路径 img/02.jpg;



实验 13　WIFI 通信实验

1.实验目的

带有 WIFI 扩展模块型号的串口屏，可以实现 WIFI 通信。目前带 WIFI 模块的串口屏可以实现 TCP 服务器、TCP 客户端、UDP 客户端、MQTT 客户端。WIFI 模块可配置为 station 或 ap 模式。

本实验以常用的 UDP 客户端作为讲解。其他模式也大同小异，也会在本实验最后一一展示测试步骤。

注：

使用 wifi 通信，需要带 wifi 模块的串口屏，并且固件版本为 1.7.27 及以上。

本实验使用 800*480 分辨率型号的屏，如使用低分辨率屏，如 480*272，可手动调整控件位置至显示区域。

2.页面设计

(1)、wifi 的连接状态。

通过定时器 5 秒的间隔读取 wifi 的连接状态，并显示。选中 timer8 控件，可以看到读取脚本代码

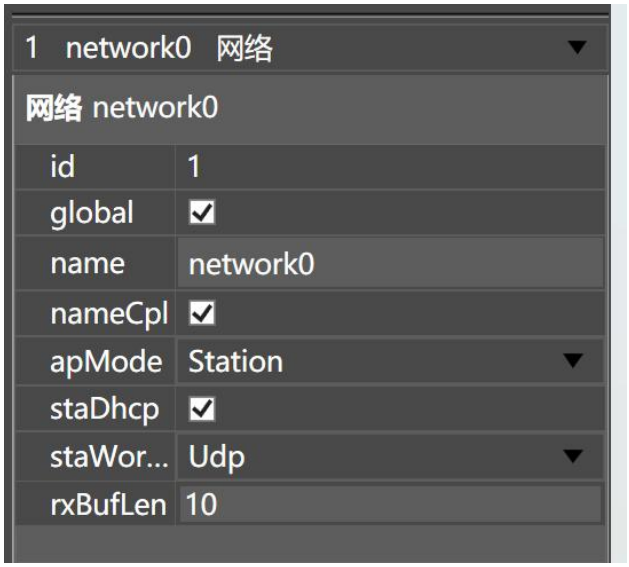
```
if(network0.staWifiConn==1)
{
  text9.txt = "已连接";
}
else
{
  text9.txt = "未连接";
```

```
}
```

(2)、按钮控件。点击后，切换到网络配置页面
(3)、文本控件。用于显示接收 UDP 报文的地址
(4)、整数控件。用于显示接收 UDP 报文的端口
(5)、整数控件。用于显示接收 UDP 报文的长度
(6)、文本控件。显示接收 UDP 报文的内容
(7)、文本控件。用于显示要发送的字符串，点击可触发键盘编辑
(8)、按钮控件。发送 UDP 报文，选中可以看到脚本

```
//发送字符串
network0.staSendString(text1.txt); //发送 text1 控件的文本，默认为 text
```

(9)、网络控件。选中可以查看配置的属性和接收脚本。

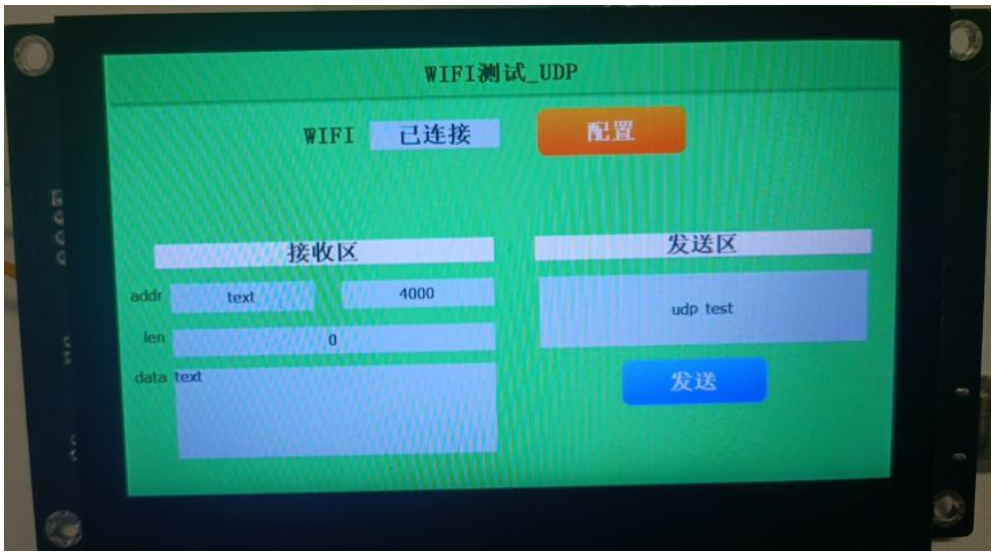


数据接收的脚本为

```
num62.val = network0.rxPort; //接收到远端端口
text22.txt= network0.rxAddr; //接收到远端地址
num24.val = network0.rxLen;    //接收到的数据长度
text13.txt= stringDecode(network0.rxBuf,0,network0.rxLen);//接收到的数据。将接收到的字节数组 rxBuf 转换为字符串，赋值给 text13 的 txt 属性
```

3.下载验证（UDP 客户端 station 模式）

1、编译后，下载到屏幕，可看到如下初始化界面。

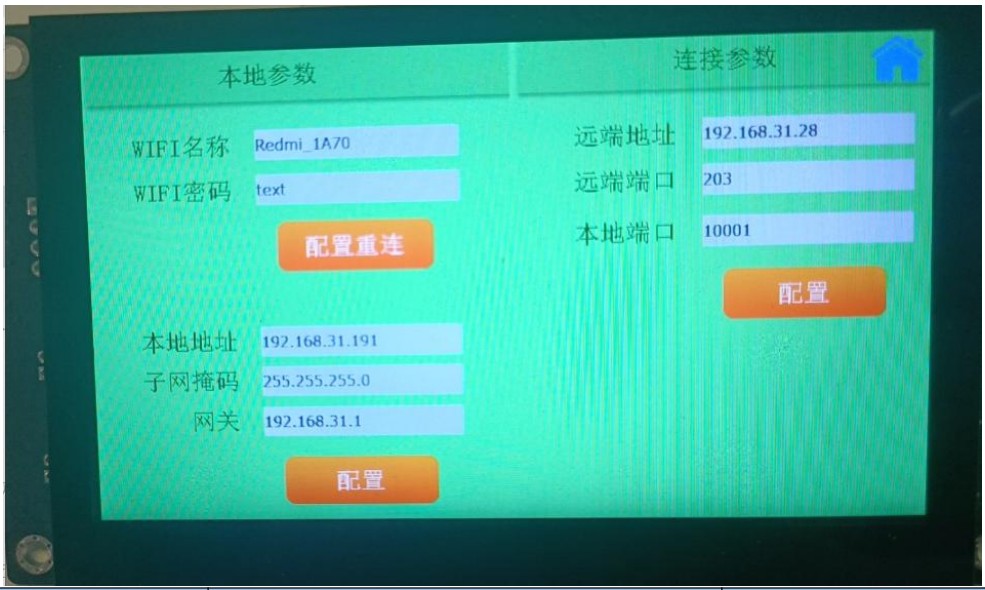


2、串口屏上电后，系统会系统自动连接 WIFI（系统保存上次的 WIFI 连接信息，会在下次上电的时候自动连接 WIFI）。如果显示未连接，需要点击配置按钮，输入 WIFI 名称和密码（一般首次使用时，需要配置），然后点击配置重连按钮。

若连接不成功，检查 WiFi 名称和密码是否填写正常，大小写是否区分，是否外接 WIFI 天线。或者先用手机、电脑连一连这个 wifi，看能否正常连接。

3、在配置界面，如果本地地址、子网掩码、网关没有显示，则需要手动输入配置（staDHCP 禁用时配置才有效），这三个参数会断电保存。

4、在配置界面，连接参数有 远端地址、远端端口、本地端口，这三个参数为配置临时有效，断电不保存。



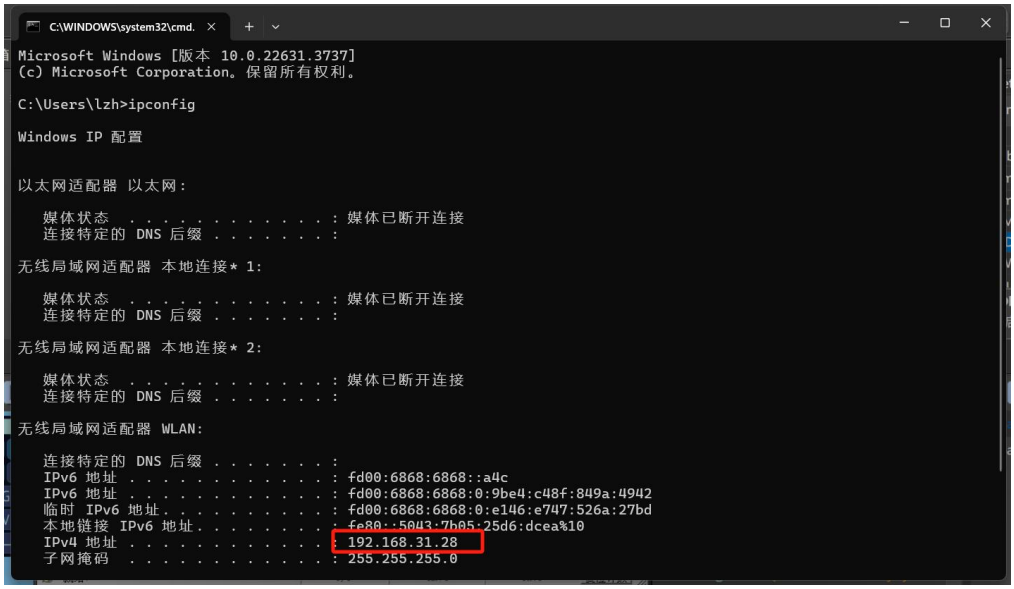
参数名称	能否配置	是掉电保存
WIFI 名称	可配置	是
WIFI 密码	可配置	是
本地地址	staDHCP 禁用时可配置	是
子网掩码	staDHCP 禁用时可配置	是
网关	staDHCP 禁用时可配置	是
远端地址	可配置	否
远端端口	可配置	否

本地端口	可配置	否

5、配置完成后，点击右上角返回按钮，可返回到主界面。

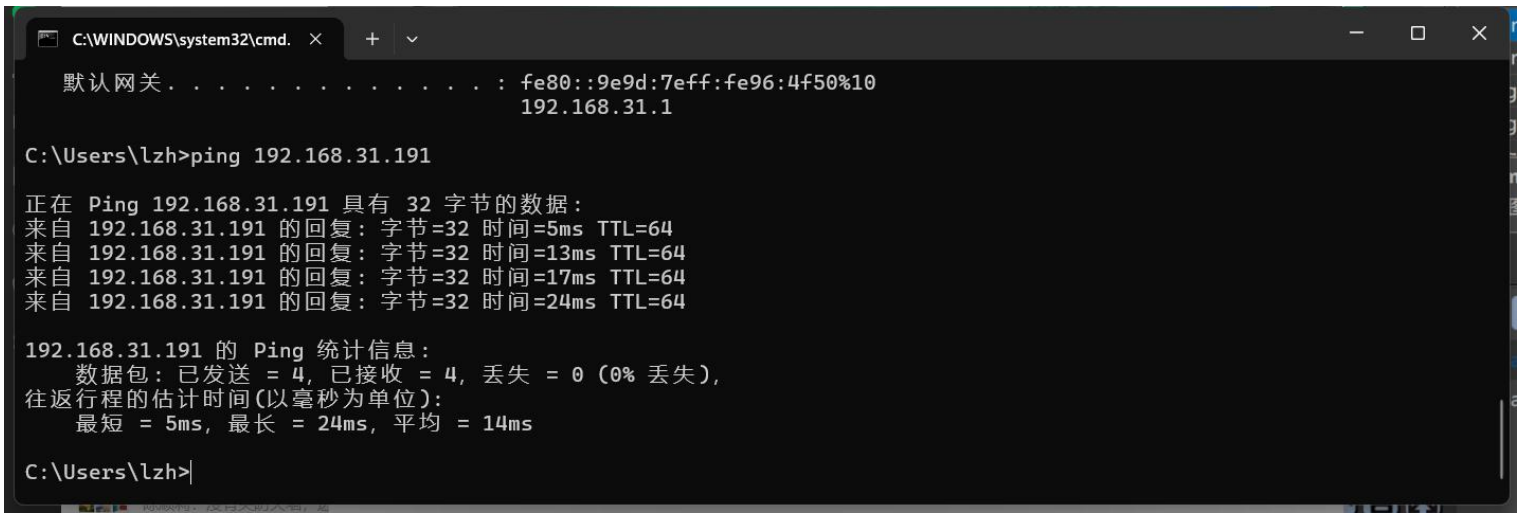
6、由于我们需要测试 UDP 通信，屏作为了一个 UDP 客户端，那我们至少还需要一个 UDP 客户端，两个 UDP 客户端互相通信。这里为了便于测试，我们使用电脑的网络调试助手作为另一个 UDP 客户端。用电脑连上同一个 WIFI，名称为 Redmi_1A70，如下图。

7、查询电脑本地 IP 地址。 win+R 弹窗输入 cmd 然后点击确定，输入 ipconfig 回车，可以看到本地的 IP 地址（连接网络的情况下才能查询到 IP 地址）。如下图



可以看到测试的电脑连上 wifi 后的 IP 地址为 192.168.31.28。所以我们需要在 第二页的配置界面 设置远端地址 remoteAddr 为 192.168.31.28(客户在验证时需要修改为自己的 IP 地址，以实际为准)

8、可以在电脑的 cmd 窗口，ping 显示屏的 IP 地址，这里为 ping 192.168.31.191，测试电脑和显示屏网络是否正常。

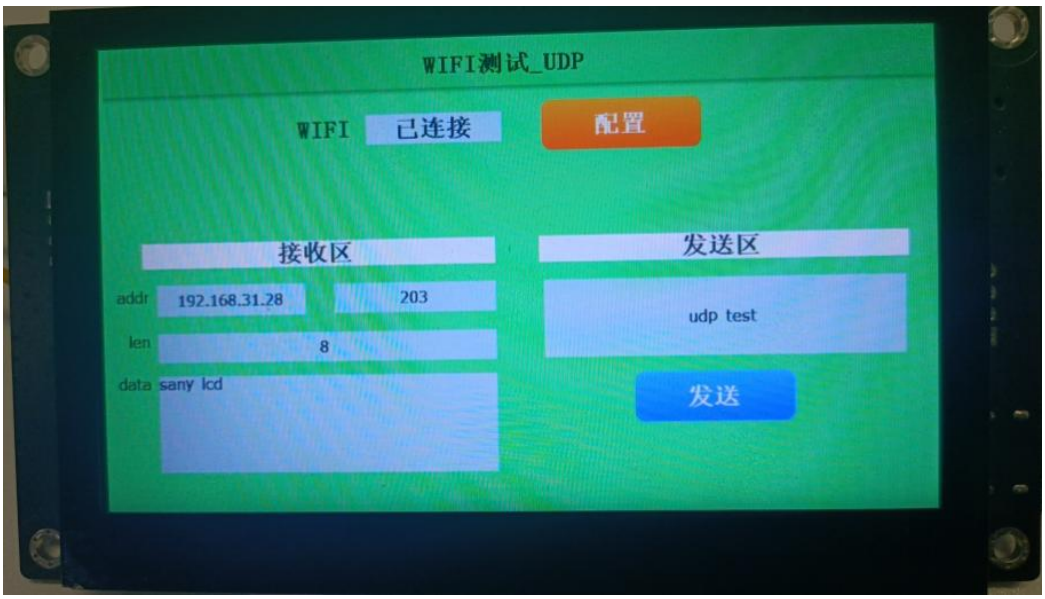


9、打开网络调试助手。协议类型选中 UDP，本地主机地址选择 192.168.31.28，端口号输入 203，点击连接。远程主机地址设置为 192.168.31.191: 10001（这是屏的 IP 地址和端口，以实际为准）

10、网络调试助手发送字符串 sany lcd。串口屏则会收到 UDP 报文，显示收到报文的地址为 192.168.31.28，端口号 203，数据长度 8，数据类容为 sany lcd；

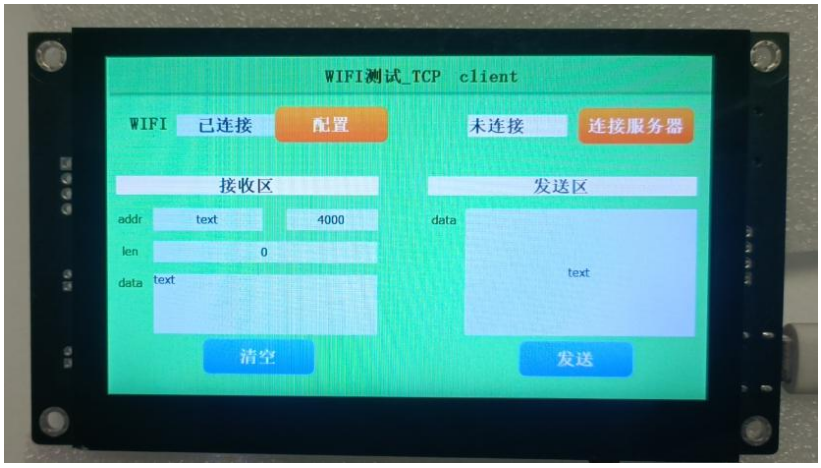
串口屏上点击发送按钮，网络调试将会收到对应的数据，这里收到的数据为字符串 udp test。

测试结果如下：


WIFI测试_UDP
WIFI
已连接
配置
接收区
发送区
addr
192.168.31.28
203
len
8
data
sany lcd
udp test
发送

4.下载验证（TCP 客户端 station 模式）

1、编译后，下载到屏幕，可看到如下初始化界面。

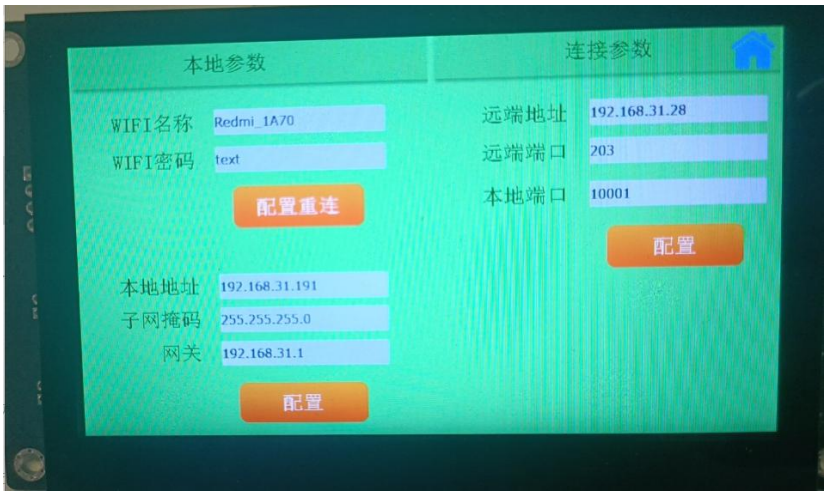


2、串口屏上电后，系统会系统连接 WIFI（系统保存上次的 WIFI 连接信息，会在下次上电的时候自动连接 WIFI）。如果显示未连接，需要点击配置按钮，输入 WIFI 名称和密码（一般首次使用时，需要配置），然后点击配置重连按钮。

若连接不成功，检查 WiFi 名称和密码是否填写正常，大小写是否区分，是否外接 WIFI 天线。或者先用手机、电脑连一连这个 wifi，看能否正常连接。

3、在配置界面，如果本地地址、子网掩码、网关没有显示，则需要手动输入配置（staDHCP 禁用时可配置），这三个参数会断电保存。

4、在配置界面，连接参数有 远端地址、远端端口、本地端口，这三个参数为配置临时有效，断电不保存。

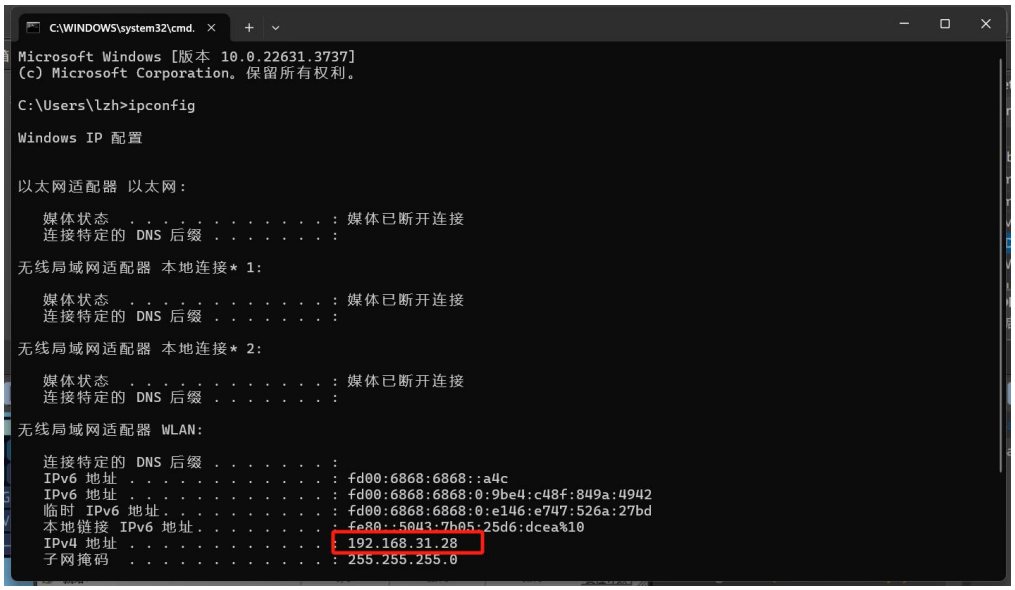


参数名称	能否配置	是掉电保存
WIFI 名称	可配置	是
WIFI 密码	可配置	是
本地地址	staDHCP 禁用时可配置	是
子网掩码	staDHCP 禁用时可配置	是
网关	staDHCP 禁用时可配置	是
远端地址	可配置	否
远端端口	可配置	否
本地端口	可配置	否

5、配置完成后，点击右上角返回按钮，可返回到主界面。

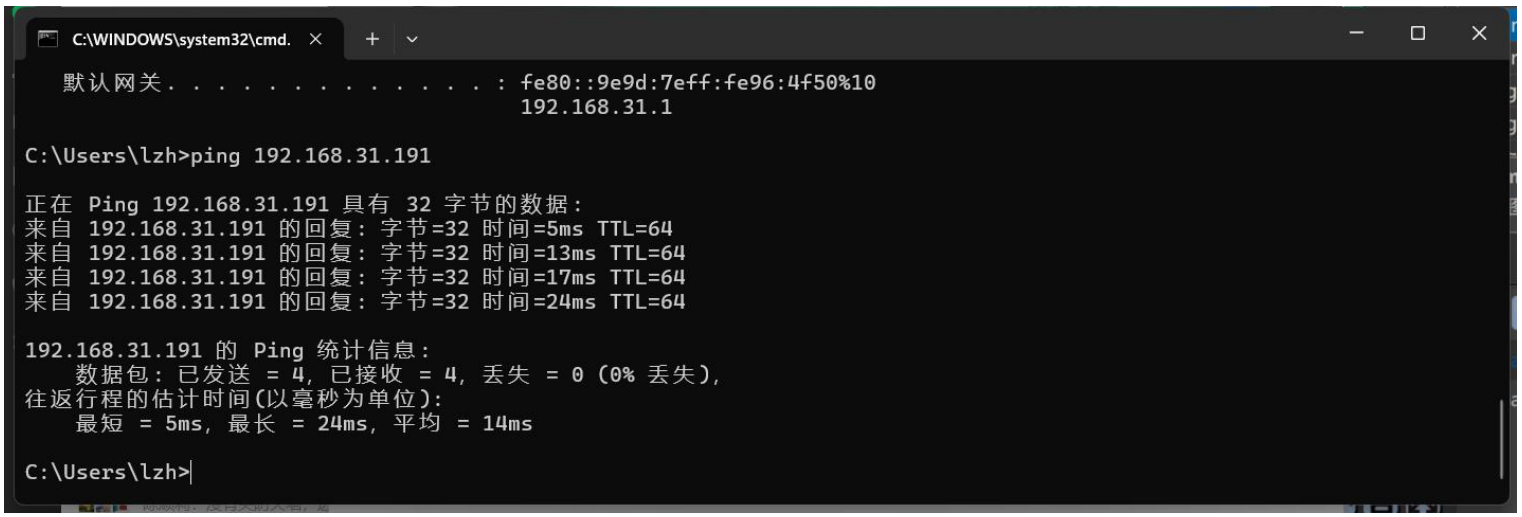
6、屏作为了一个 TCPclient 端，那我们至少还需要一个 TCP 服务器。这里为了便于测试，我们使用电脑的网络调试助手作为 TCP 服务器。用电脑连上同一个 WIFI。用电脑连上同一个 WIFI，名称为 Redmi_1A70，如下图。

7、询电脑本地 IP 地址。 win+R 弹窗输入 cmd 然后点击确定，可以看到本地的 IP 地址。如下图



可以看到测试的电脑连上 wifi 后的 IP 地址为 192.168.31.28。所以我们需要在 第二页的配置界面 设置远端地址 remoteAddr 为 192.168.31.28(客户在验证时需要修改为自己的 IP 地址，以实际为准)

8、可以在电脑的 cmd 窗口，ping 显示屏的 IP 地址，这里为 ping 192.168.31.191，测试电脑和显示屏网络是否正常。



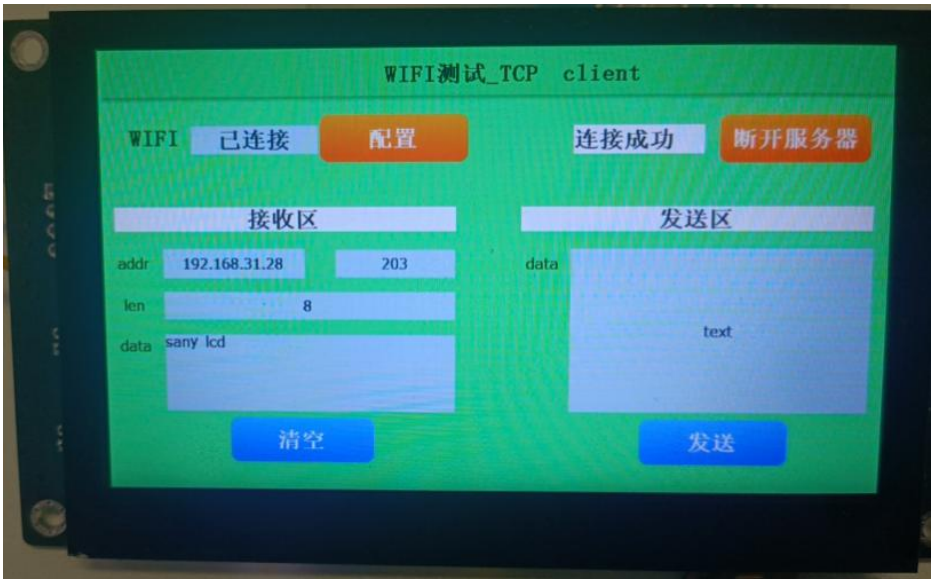
9、打开网络调试助手。协议类型选择 TCP Server，本地主机地址选择 192.168.31.28(客户在验证时需要修改为自己的 IP 地址)，端口号输入 203，点击连接即打开了 TCP 服务器端口。

9、屏上点击”连接服务器”按钮。

11、串口屏上点击发送按钮，网络调试将会收到对应的数据，这里收到的数据为字符串 text。

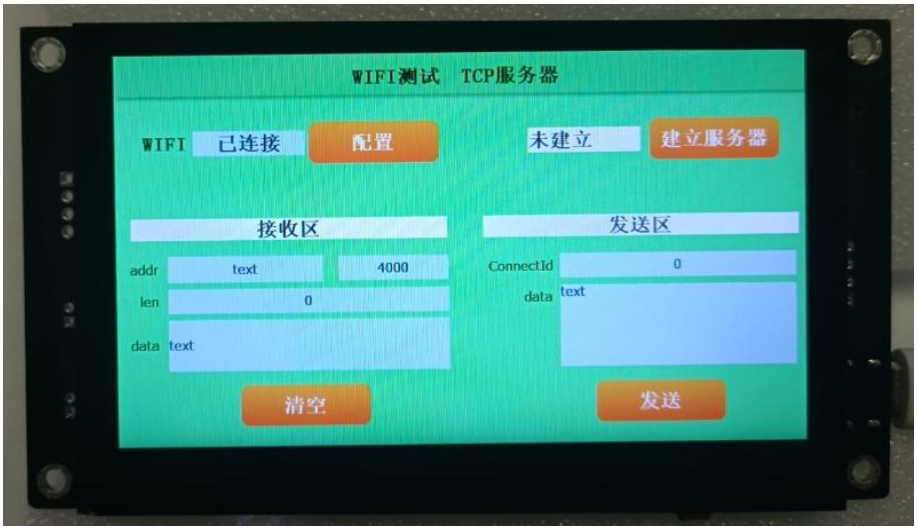
12、网络调试助手发送字符串 sany lcd。串口屏则会收到 UDP 报文，显示收到报文的地址为 192.168.31.28，端口号 203，数据长度 8，数据类容为 sany lcd；

测试结果如下：



5.下载验证（TCP 服务器 station 模式）

1、编译后，下载到屏幕，可看到如下初始化界面。

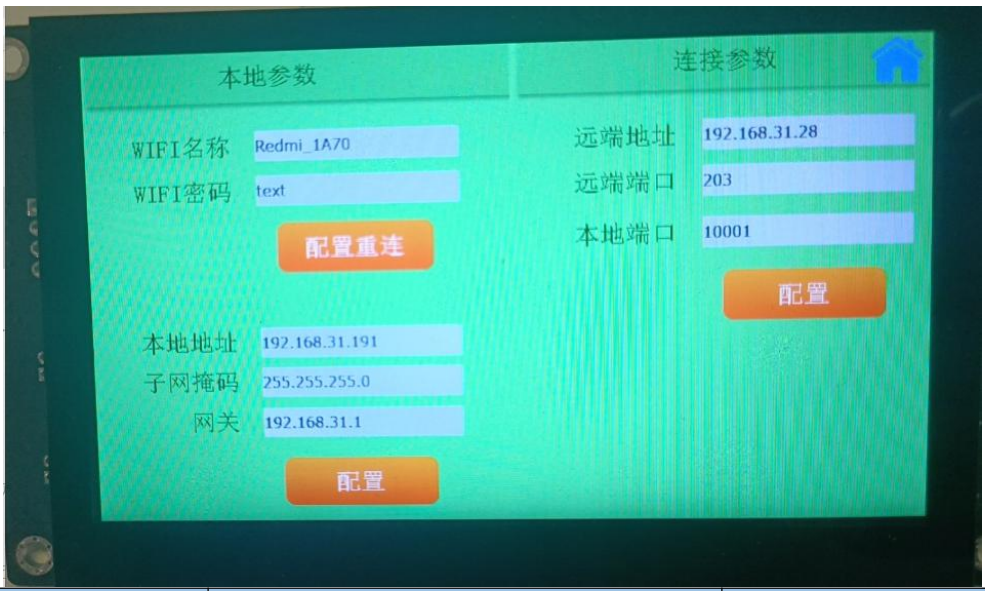


2、串口屏上电后，系统会系统连接 WIFI（系统保存上次的 WIFI 连接信息，会在下次上电的时候自动连接 WIFI）。如果显示未连接，需要点击配置按钮，输入 WIFI 名称和密码（一般首次使用时，需要配置），然后点击配置重连按钮。

若连接不成功，检查 WiFi 名称和密码是否填写正常，大小写是否区分，是否外接 WIFI 天线。或者先用手机、电脑连一连这个 wifi，看能否正常连接。

3、在配置界面，如果本地地址、子网掩码、网关没有显示，则需要手动输入配置（staDHCP 禁用时可配置），这三个参数会断电保存。

4、在配置界面，连接参数有 远端地址、远端端口、本地端口，这三个参数为配置临时有效，断电不保存。显示屏作为 TCP 服务器时，远端地址和远端端口无效。



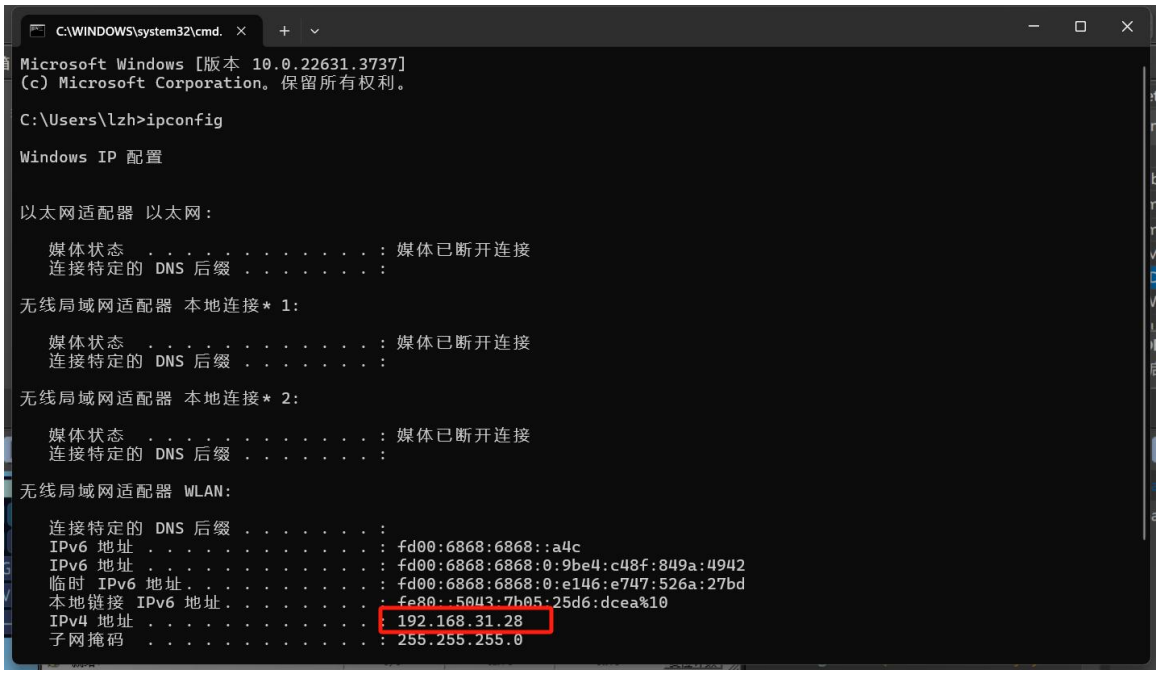
参数名称	能否配置	是掉电保存
WIFI 名称	可配置	是
WIFI 密码	可配置	是
本地地址	staDHCP 禁用时可配置	是
子网掩码	staDHCP 禁用时可配置	是
网关	staDHCP 禁用时可配置	是

远端地址	无效	否
远端端口	无效	否
本地端口	可配置	否

5、配置完成后，点击右上角返回按钮，可返回到主界面。

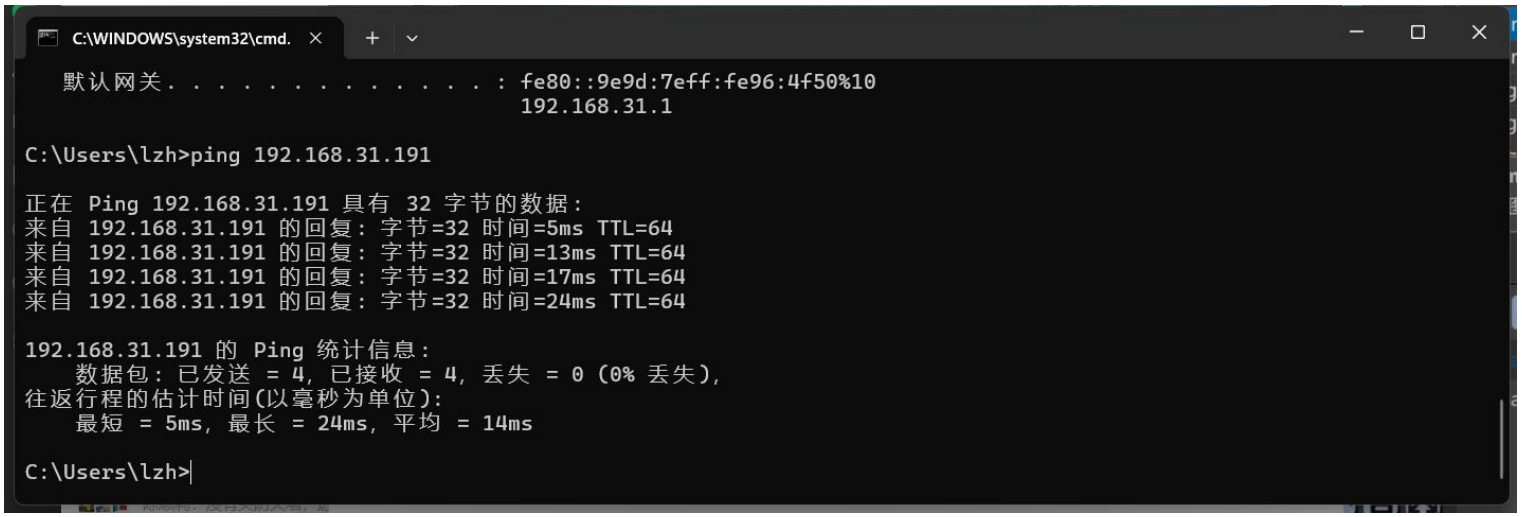
6、由于我们需要测试 TCP 通信，屏作为了一个 TCP 服务器端，那我们至少还需要一个 TCP 客户端。这里为了便于测试，我们使用电脑的网络调试助手作为一个 TCP 客户端。用电脑连上同一个 WIFI，如下图。

7、查询电脑本地 IP 地址。 win+R 弹窗输入 cmd 然后点击确定，可以看到本地的 IP 地址。如下图



可以看到测试的电脑连上 wifi 后的 IP 地址为 192.168.31.28。

8、可以在电脑的 cmd 窗口，ping 显示屏的 IP 地址，这里为 ping 192.168.31.191，测试电脑和显示屏网络是否正常。



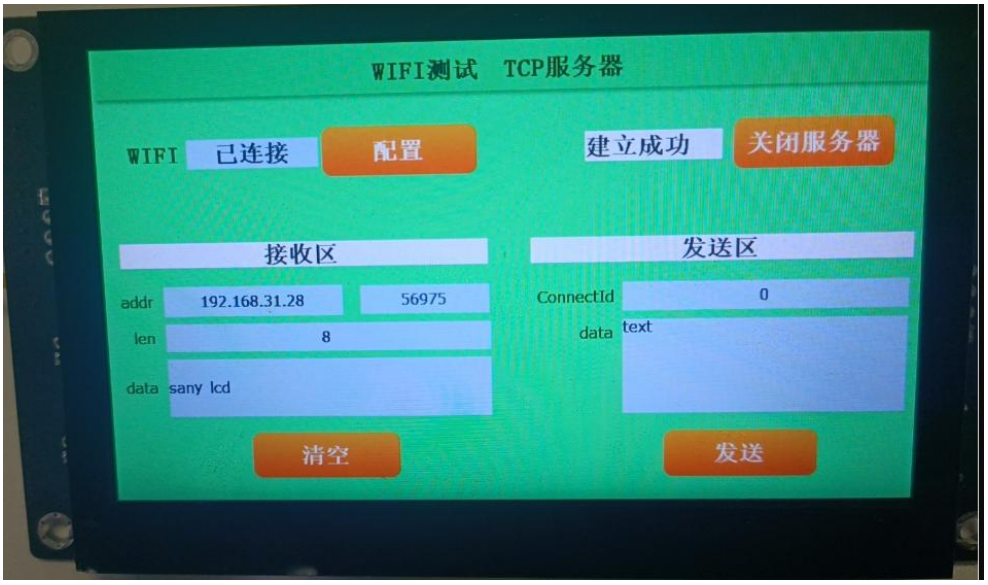
9、在主界面上，点击按钮，建立一个 TCP 服务器

10、打开网络调试助手。协议类型选中 TCP client，远程主机地址选择 192.168.31.191，远程端口号选择 10001（以配置界面，显示屏的 IP 地址和端口为准）。点击连接。

11、网络调试助手发送字符串 sany lcd。串口屏则会收到 TCP 报文，显示收到报文的地址为 192.168.31.28，端口号 56975，数据长度 8，数据类容为 sany lcd；

12、串口屏上点击发送按钮，网络调试将会收到对应的数据，这里收到的数据为字符串 text。

测试结果如下：




网络调试助手
网络设置
(1) 协议类型
TCP Client
(2) 远程主机地址
192.168.31.191
(3) 远程主机端口
10001
断开
接收设置
ASCII HEX
按日志模式显示
接收区自动换行
接收数据不显示
接收保存到文件...
自动滚屏 清除接收
发送设置
ASCII HEX
转义符指令解析
自动发送附加位
打开文件数据源...
循环周期 1000 ms
快捷指令 历史发送
数据日志
NetAssist V5.0.3
[2024-06-24 11:15:21.330]# The server is connected from local 192.168.31.27:63499
[2024-06-24 11:16:29.420]# SEND ASCII>
sany lcd
[2024-06-24 11:16:31.801]# RECV ASCII>
text
数据发送
清除 清除
sany lcd
发送
就绪!
1/1
RX:4
TX:8
复位计数

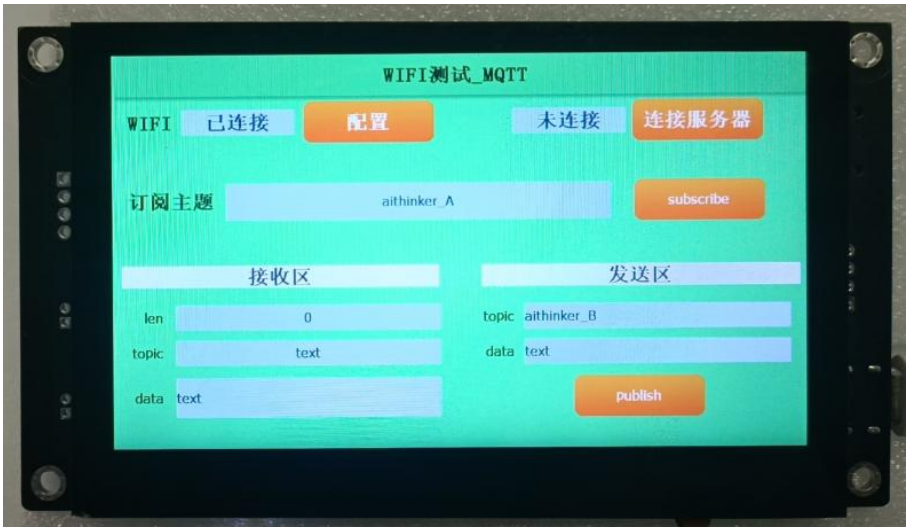
6.下载验证（MQTT 客户端 station 模式）

由于我们需要测试 MQTT 通信，屏作为了一个 MQTT 客户端。显示屏使用的是安信可公司的 WIFI 模组，这里我们使用安信可提供的 MQTT 服务器作为测试

MQTT 服务器 ：wx.ai-thinker.com 连接端口：1883

MQTT 服务器控制台 ：wx.ai-thinker.com 连接端口：18083

1、 编译后，下载到屏幕，可看到如下初始化界面。



2、串口屏上电后，系统会系统连接 WIFI（系统保存上次的 WIFI 连接信息，会在下次上电的时候自动连接 WIFI）。如果显示未连接，需要点击配置按钮，输入 WIFI 名称和密码（一般首次使用时，需要配置），然后点击配置重连按钮。

若连接不成功，检查 WiFi 名称和密码是否填写正常，大小写是否区分，是否外接 WIFI 天线。或者先用手机、电脑连一连这个 wifi，看能否正常连接。

3、在配置界面，如果本地地址、子网掩码、网关没有显示，则需要手动输入配置（staDHCP 禁用时可配置），这三个参数会断电保存。

4、在配置界面，连接参数有 远端地址、远端端口、本地端口，这三个参数为配置临时有效，断电不保存。

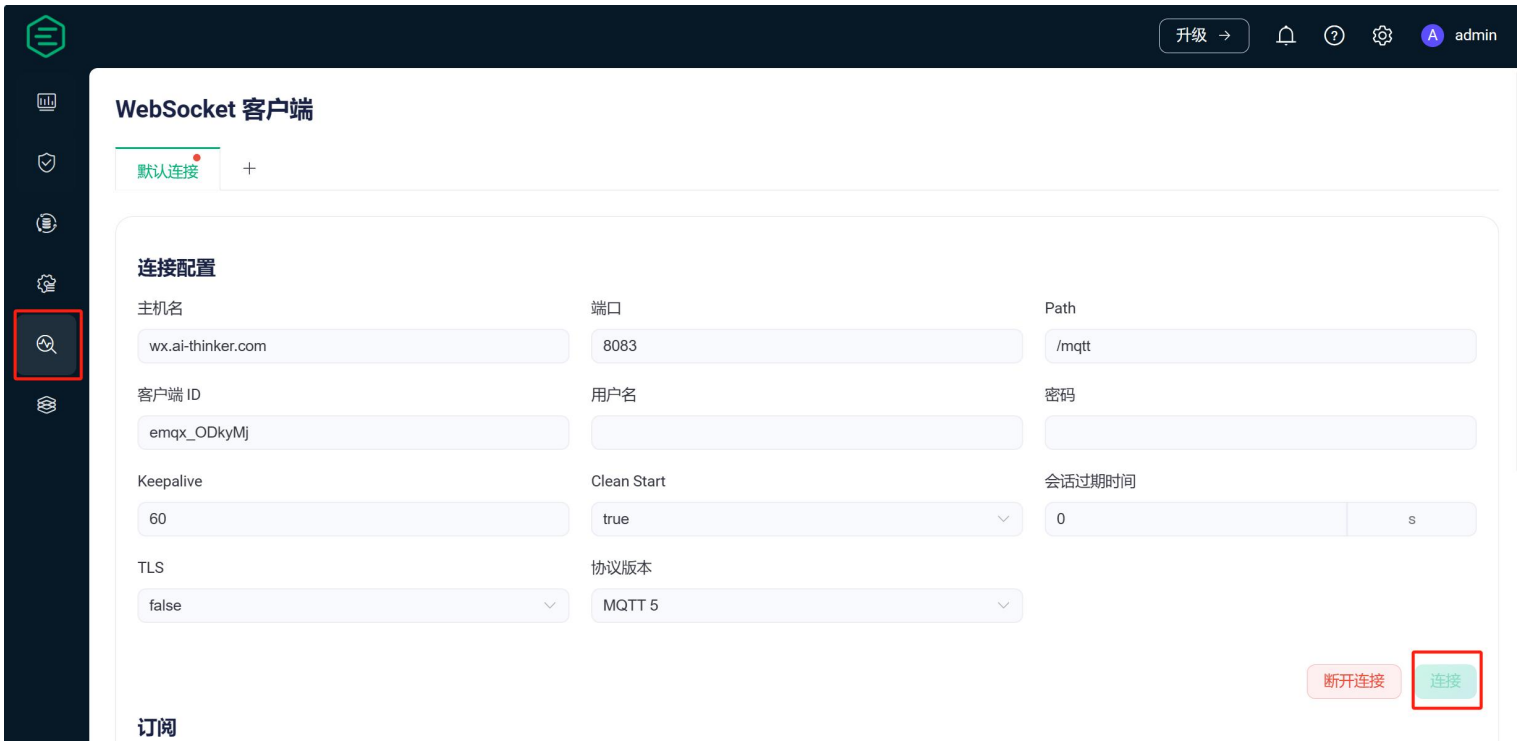
参数名称	能否配置	是掉电保存
WIFI 名称	可配置	是
WIFI 密码	可配置	是
本地地址	staDHCP 禁用时可配置	是
子网掩码	staDHCP 禁用时可配置	是
网关	staDHCP 禁用时可配置	是
远端地址	可配置	否
远端端口	可配置	否
本地端口	可配置	否
MQTT 客户端 id	可配置	否
MQTT 登陆用户名	可配置	否
MQTT 登陆密码	可配置	否

5、配置完成后，点击右上角返回按钮，可返回到主界面。
6、点击界面上的“连接服务器”按钮。系统会按照网络控件的配置连接 MQTT 服务器：wx.ai-thinker.com 连接端口：1883

7、点击界面上的“subscribe”按钮，将会订阅主题 aithinker_A.

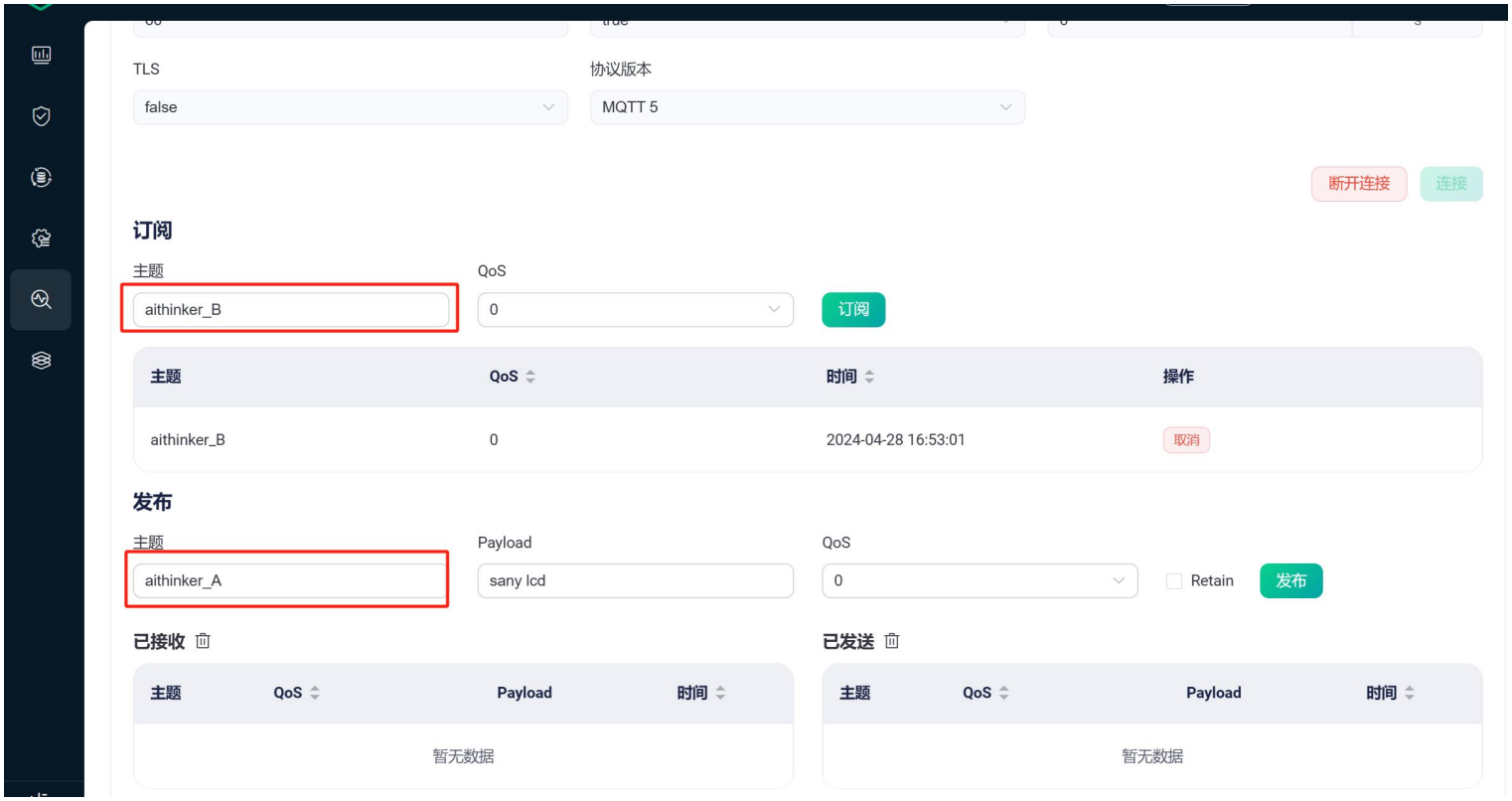
8、登录 MQTT 服务器控制台 wx.ai-thinker.com 连接端口：18083
打开浏览器输入 http://wx.ai-thinker.com:18083
username: admin
password: fae12345678

9、打开 Websocket 客户端，连接配置，默认即可，点击连接



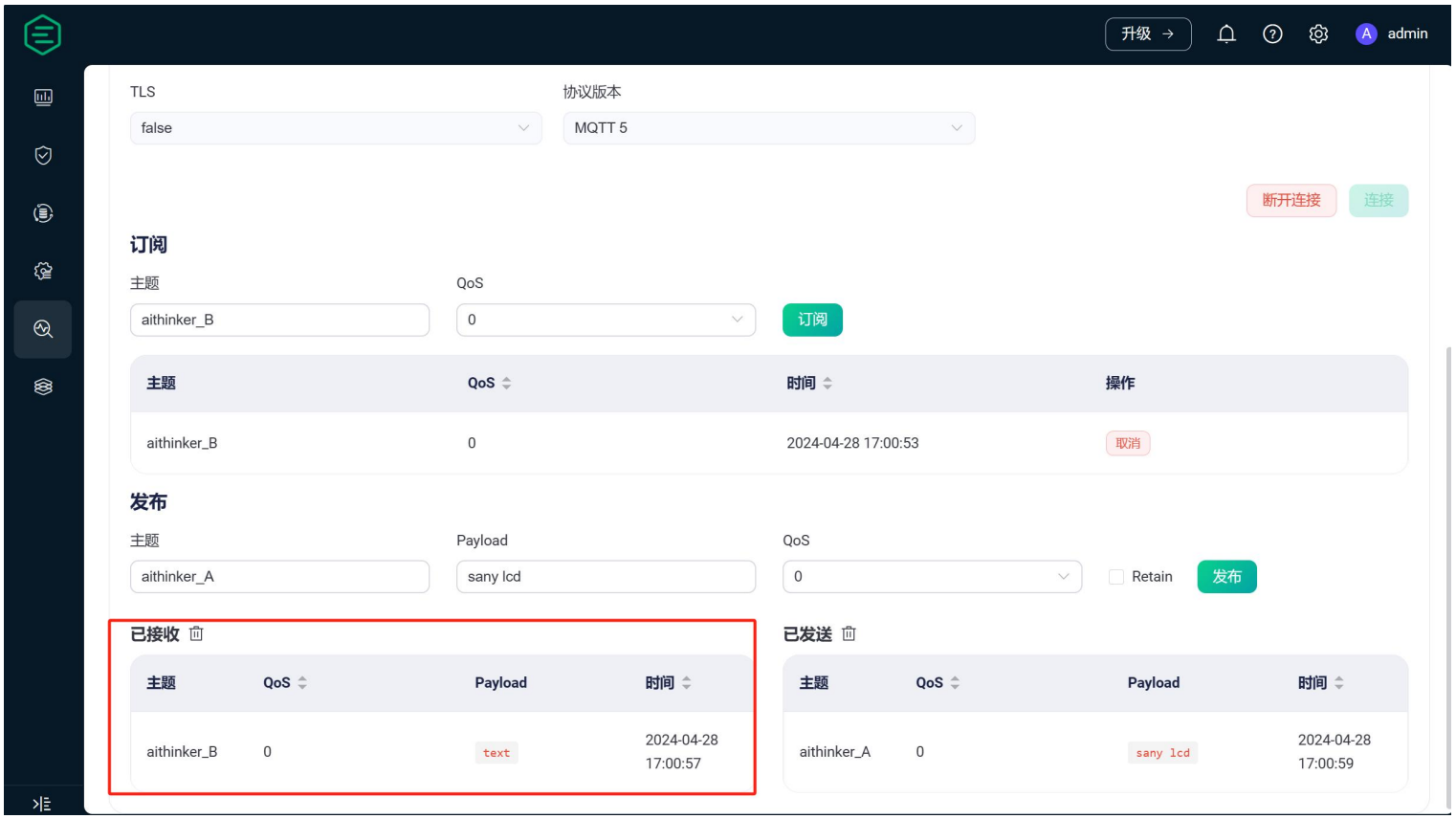
10、Websocket 客户端订阅的主题设置为 aithinker_B，点击点阅；发布的主题设置为 aithinker_A，发布的消息设置输入”sany

lcd”,,



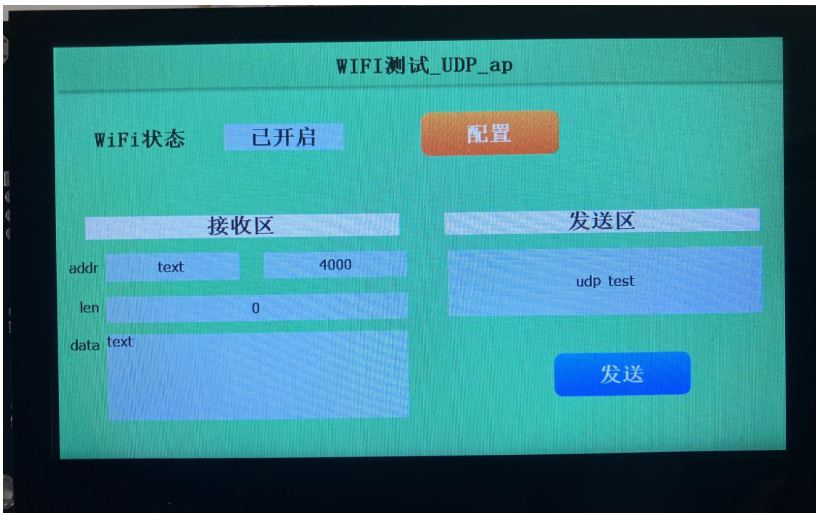
11、Websocket 客户端上点击发布按钮，显示屏会收到消息

12、显示屏上点击 publish 按钮，Websocket 客户端则会收到消息

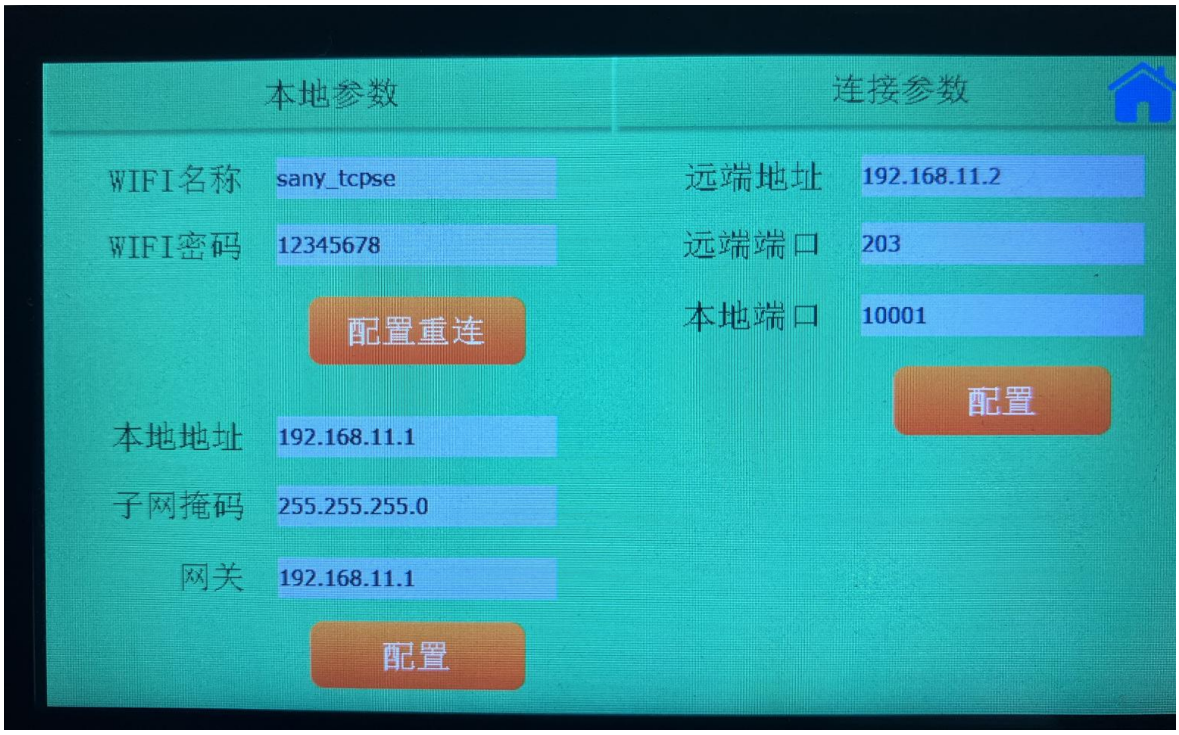


7.下载验证（UDP 客户端 ap 模式）

1、编译后，下载到屏幕，可看到如下初始化界面。



2、串口屏上电后，系统会系统自动打开热点（系统设置的热点信息，包括热点名称和密码，会在下次上电的时候以该参数打开热点）。如果显示未开启，需要点击配置按钮，输入 WIFI 名称和密码（一般首次使用时，需要配置），然后点击配置重连按钮。
3、在配置界面，如果本地地址、子网掩码、网关没有显示，则需要手动输入配置，这三个参数会断电保存。
4、在配置界面，连接参数有 远端地址、远端端口、本地端口，这三个参数为配置临时有效，断电不保存。



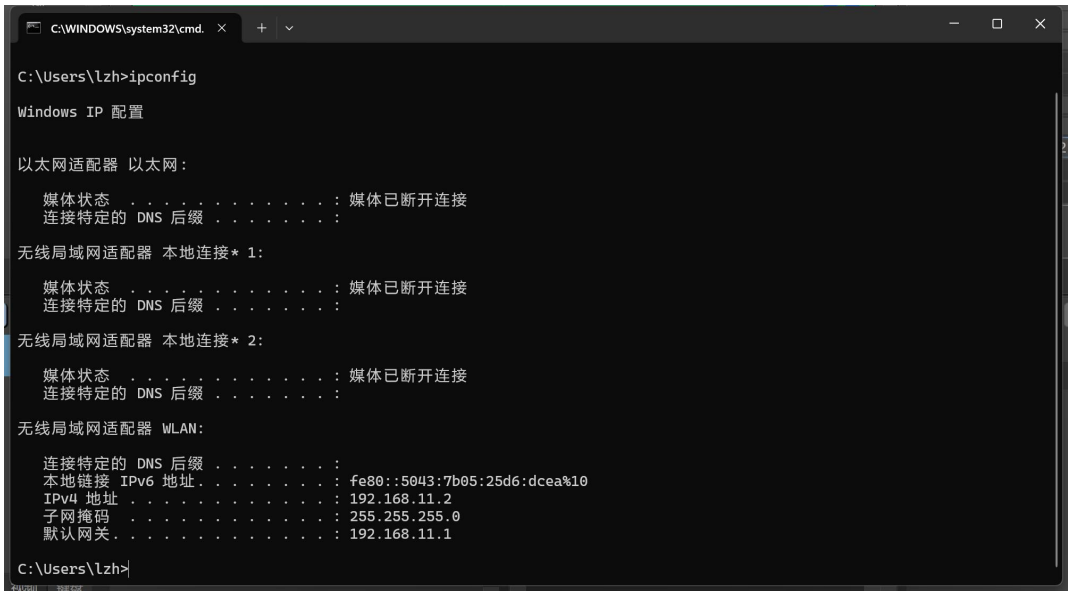
参数名称	能否配置	是掉电保存
WIFI 名称	可配置	是
WIFI 密码	可配置	是
本地地址	可配置	是
子网掩码	可配置	是
网关	可配置	是
远端地址	可配置	否

远端端口	可配置	否
本地端口	可配置	否

5、配置完成后，点击右上角返回按钮，可返回到主界面。

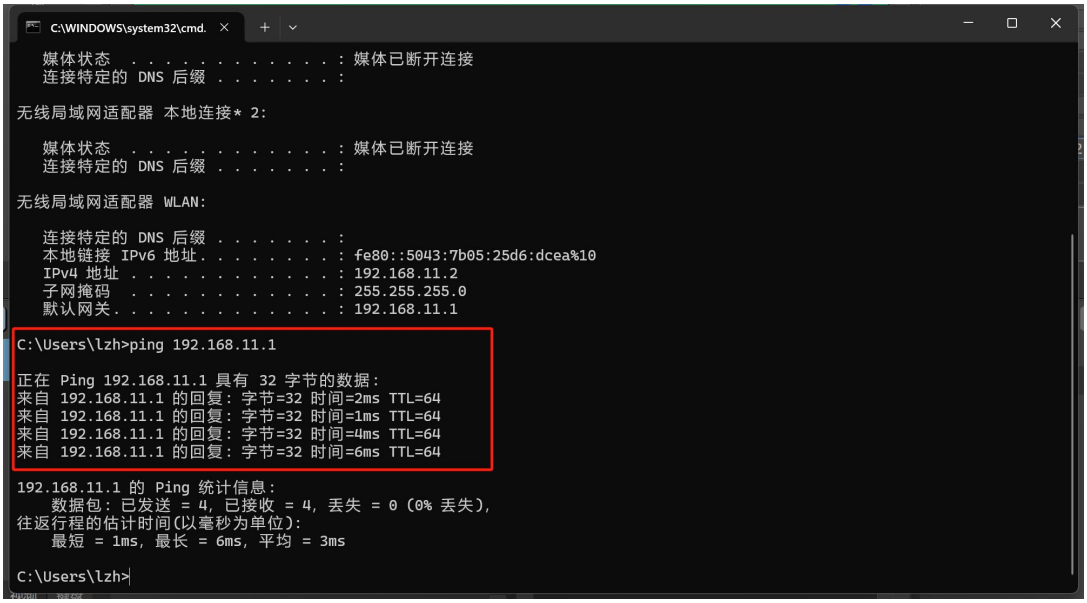
6、由于我们需要测试 UDP 通信，屏作为了一个 UDP 客户端，那我们至少还需要一个 UDP 客户端，两个 UDP 客户端互相通信。这里为了便于测试，我们使用电脑的网络调试助手作为另一个 UDP 客户端。用电脑连上串口屏发出的热点。

7、查询电脑本地 IP 地址。 win+R 弹窗输入 cmd 然后点击确定，输入 ipconfig 回车，可以看到本地的 IP 地址（连接网络的情况下才能查询到 IP 地址）。如下图

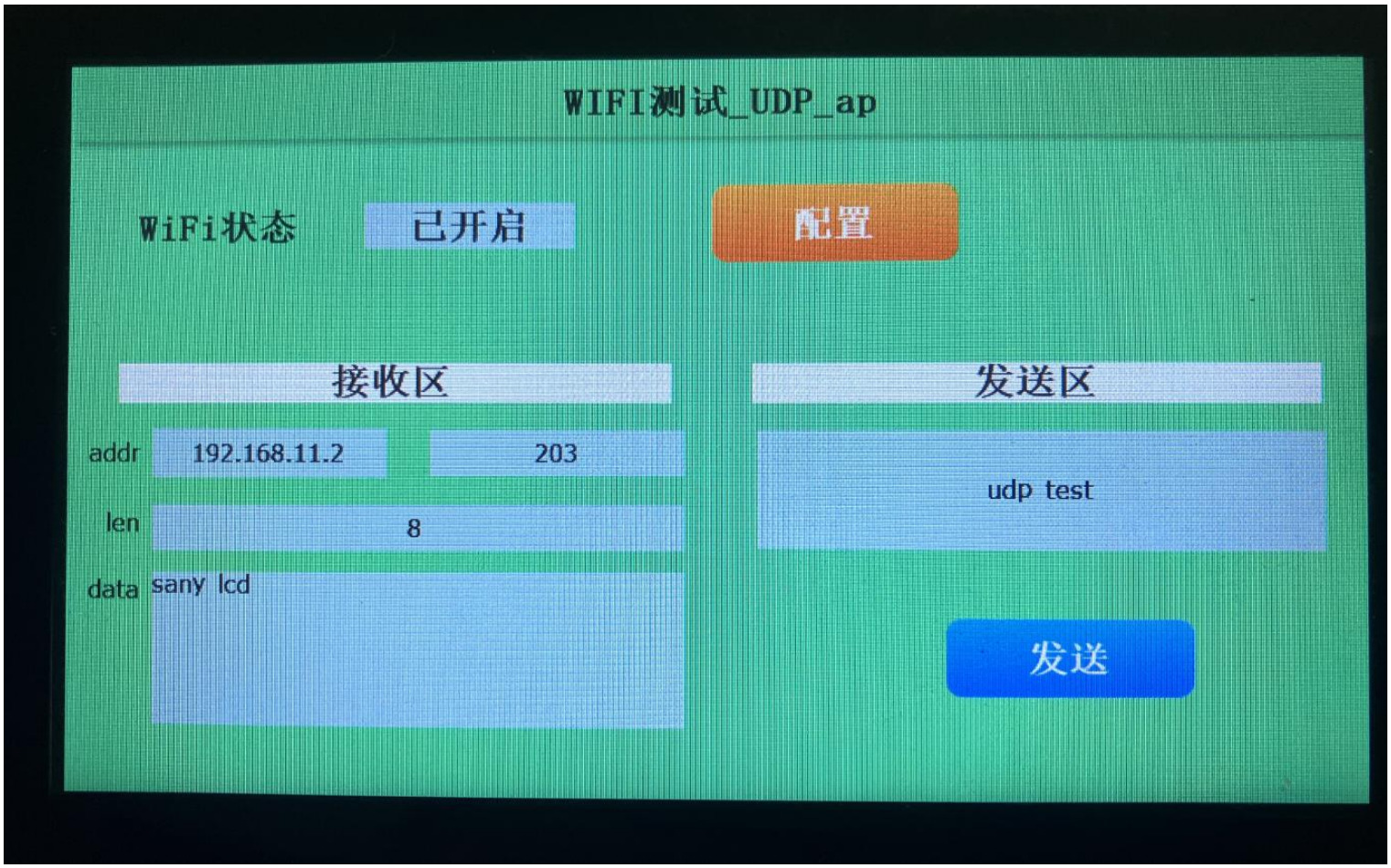


可以看到测试的电脑连上热点后的 IP 地址为 192.168.11.2。所以我们需要在 第二页的配置界面 设置远端地址 remoteAddr 为 192.168.11.2(客户在验证时需要修改为自己的 IP 地址，以实际为准)

8、可以在电脑的 cmd 窗口，ping 显示屏的 IP 地址，这里为 ping 192.168.11.1，测试电脑和显示屏网络是否正常。

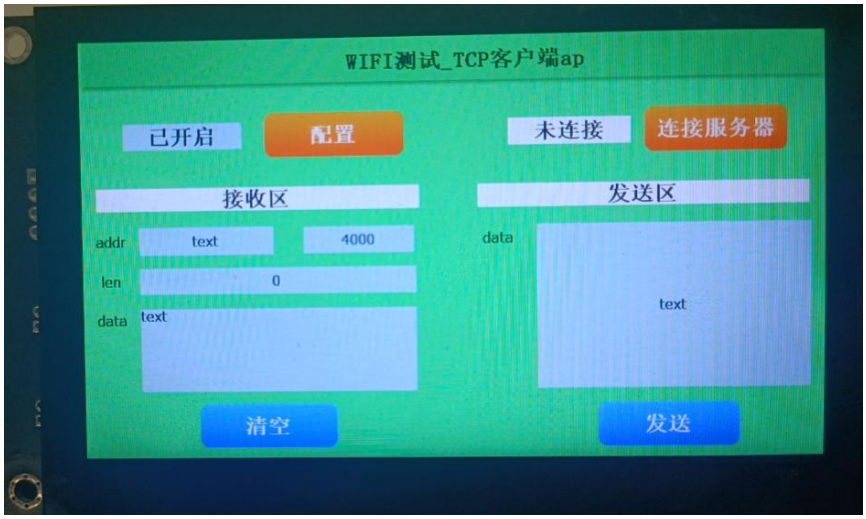


9、打开网络调试助手。协议类型选中 UDP，本地主机地址选择 192.168.11.2，端口号输入 203，点击连接。
远程主机地址设置为 192.168.11.1: 10001（这是屏的 IP 地址和端口，以实际为准）
网络调试助手发送字符串 sany lcd。串口屏则会收到 UDP 报文，显示收到报文的地址为 192.168.11.2，端口号 203，数据长度 8，数据类容为 sany lcd；
串口屏上点击发送按钮，网络调试将会收到对应的数据，这里收到的数据为字符串 udp test。
测试结果如下：


WIFI测试_UDP_ap
WiFi状态
已开启
配置
接收区
发送区
addr
192.168.11.2
203
udp test
len
8
data
sany lcd
发送

8.下载验证（TCP 客户端 ap 模式）

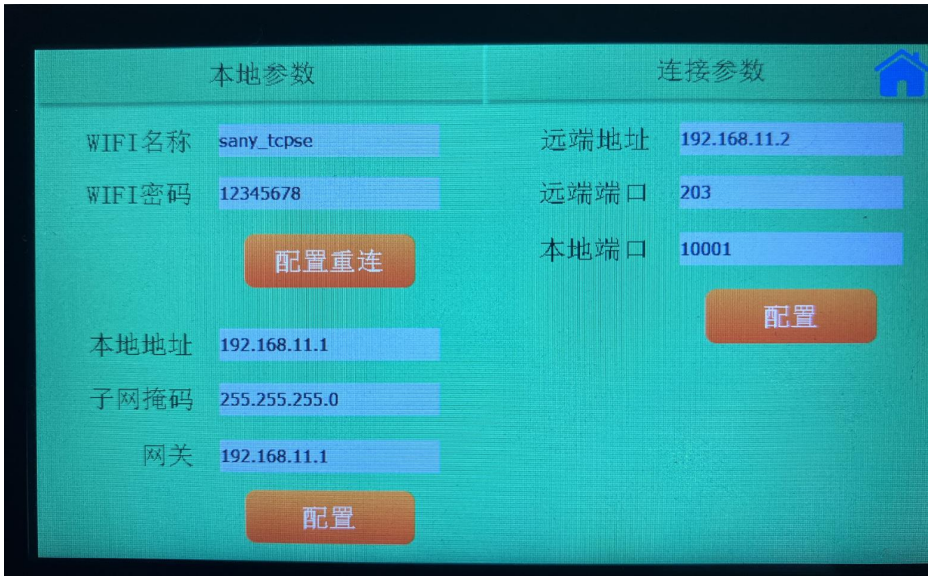
1、编译后，下载到屏幕，可看到如下初始化界面。



2、串口屏上电后，系统会系统自动打开热点（系统设置的热点信息，包括热点名称和密码，会在下次上电的时候以该参数打开热点）。如果显示未开启，需要点击配置按钮，输入 WIFI 名称和密码（一般首次使用时，需要配置），然后点击配置重连按钮。

3、在配置界面，如果本地地址、子网掩码、网关没有显示，则需要手动输入配置，这三个参数会断电保存。

4、在配置界面，连接参数有 远端地址、远端端口、本地端口，这三个参数为配置临时有效，断电不保存。

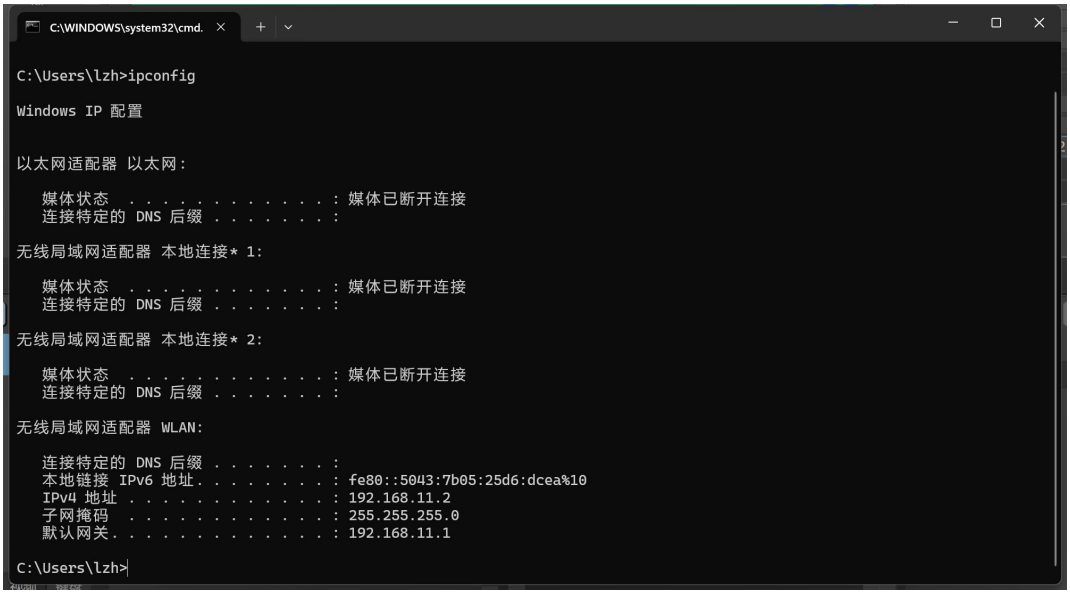


参数名称	能否配置	是掉电保存
WIFI 名称	可配置	是
WIFI 密码	可配置	是
本地地址	可配置	是
子网掩码	可配置	是
网关	可配置	是
远端地址	可配置	否
远端端口	可配置	否
本地端口	可配置	否

5、配置完成后，点击右上角返回按钮，可返回到主界面。

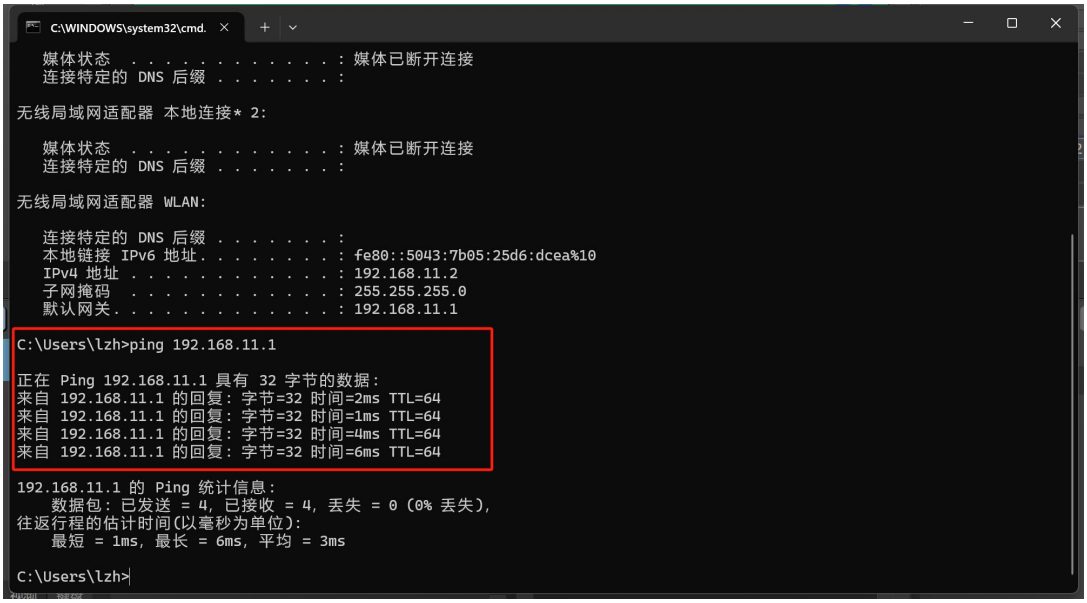
6、屏作为了一个 TCPclient 端，那我们至少还需要一个 TCP 服务器。这里为了便于测试，我们使用电脑的网络调试助手作为 TCP 服务器。用电脑连上串口屏发出的热点。

7、查询电脑本地 IP 地址。 win+R 弹窗输入 cmd 然后点击确定，输入 ipconfig 回车，可以看到本地的 IP 地址（连接网络的情况下才能查询到 IP 地址）。如下图



可以看到测试的电脑连上热点后的 IP 地址为 192.168.11.2。所以我们需要在 第二页的配置界面 设置远端地址 remoteAddr 为 192.168.11.2(客户在验证时需要修改为自己的 IP 地址，以实际为准)

8、可以在电脑的 cmd 窗口，ping 显示屏的 IP 地址，这里为 ping 192.168.11.1，测试电脑和显示屏网络是否正常。



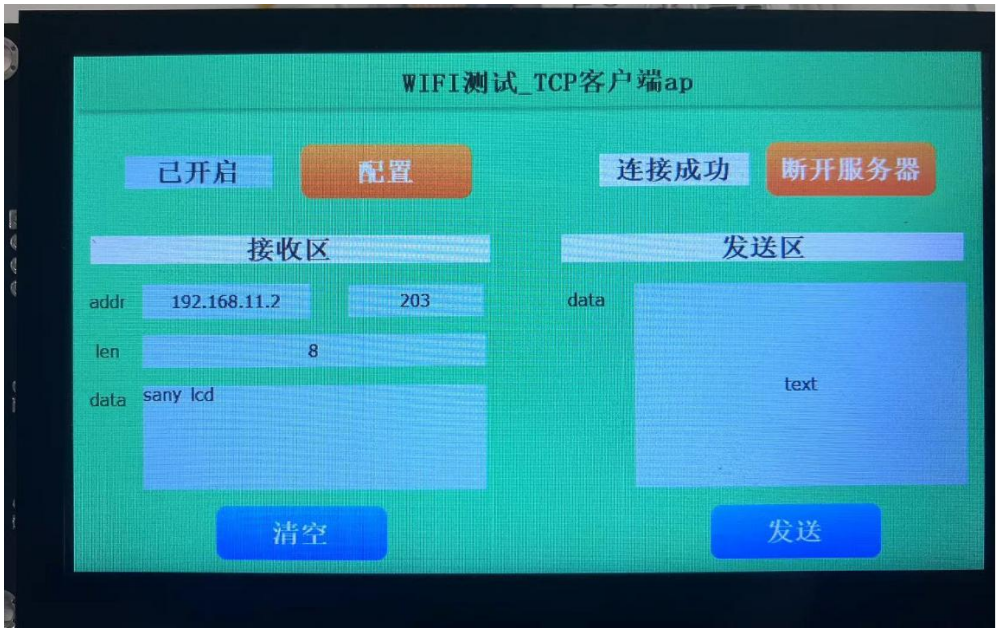
9、打开网络调试助手。协议类型选中 TCP Server，本地主机地址选择 192.168.11.2，端口号输入 203，点击打开，打开服务器端口。

10、屏上点击”连接服务器”按钮，网络调试助手上有一个客户端已经连上。

11、串口屏上点击发送按钮，网络调试将会收到对应的数据，这里收到的数据为字符串 text。

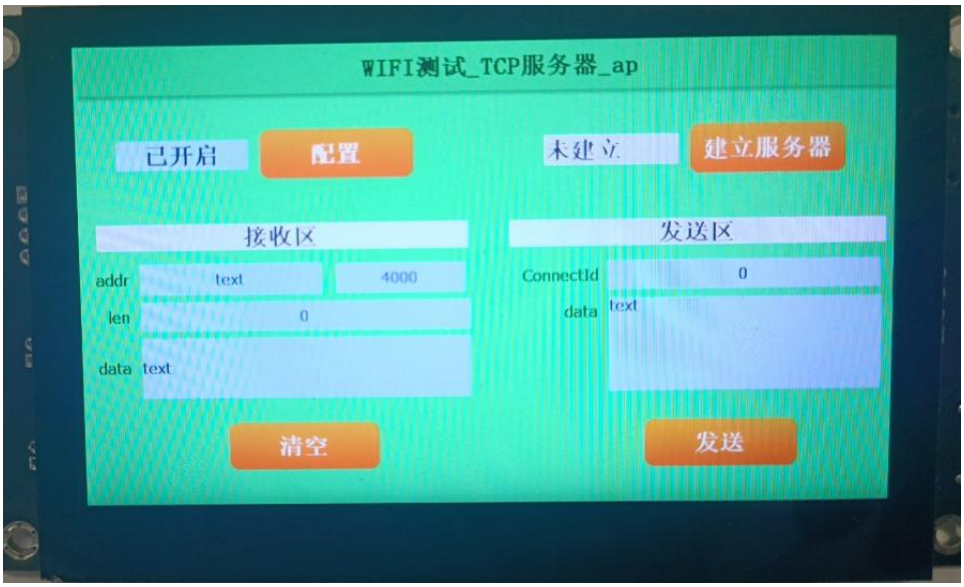
12、网络调试助手发送字符串 sany lcd。串口屏则会收到 TCP 报文，显示收到报文的地址为 192.168.11.2，端口号 203，数据长度 8，数据类容为 sany lcd；

13、测试结果如下：



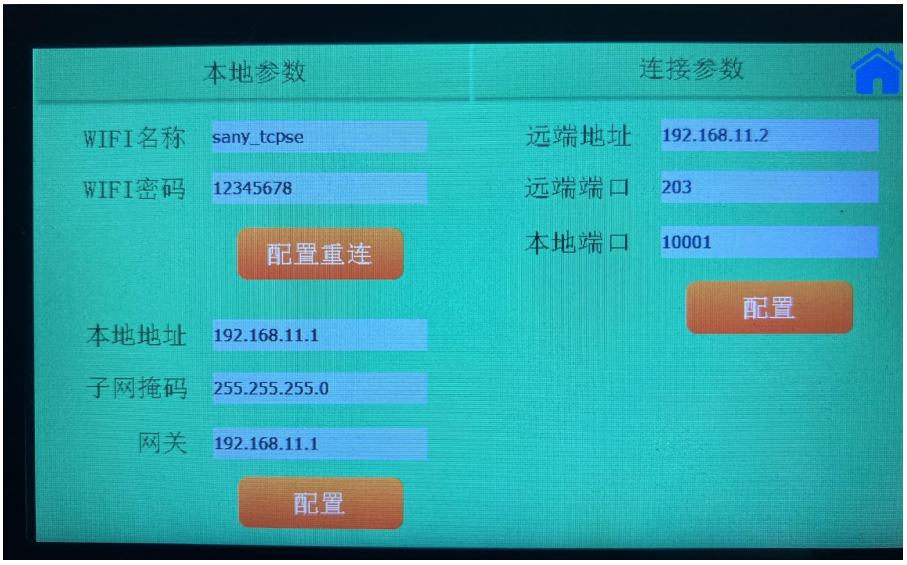
9.下载验证（TCP 服务器 ap 模式）

1、编译后，下载到屏幕，可看到如下初始化界面。



2、串口屏上电后，系统会系统自动打开热点（系统设置的热点信息，包括热点名称和密码，会在下次上电的时候以该参数打开热点）。如果显示未开启，需要点击配置按钮，输入 WIFI 名称和密码（一般首次使用时，需要配置），然后点击配置重连按钮。
3、在配置界面，如果本地地址、子网掩码、网关没有显示，则需要手动输入配置，这三个参数会断电保存。

4、在配置界面，连接参数有 远端地址、远端端口、本地端口，这三个参数为配置临时有效，断电不保存。显示屏作为 TCP 服务器时，远端地址和远端端口无效



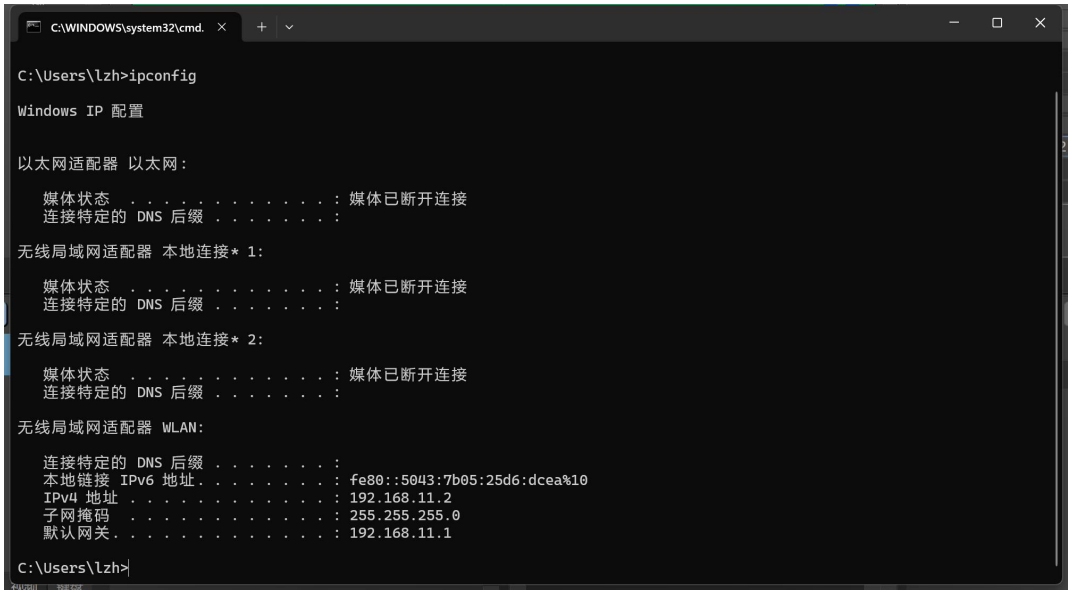
参数名称	能否配置	是掉电保存
WIFI 名称	可配置	是
WIFI 密码	可配置	是
本地地址	可配置	是
子网掩码	可配置	是
网关	可配置	是
远端地址	无效	否
远端端口	无效	否

本地端口	可配置	否

5、配置完成后，点击右上角返回按钮，可返回到主界面。

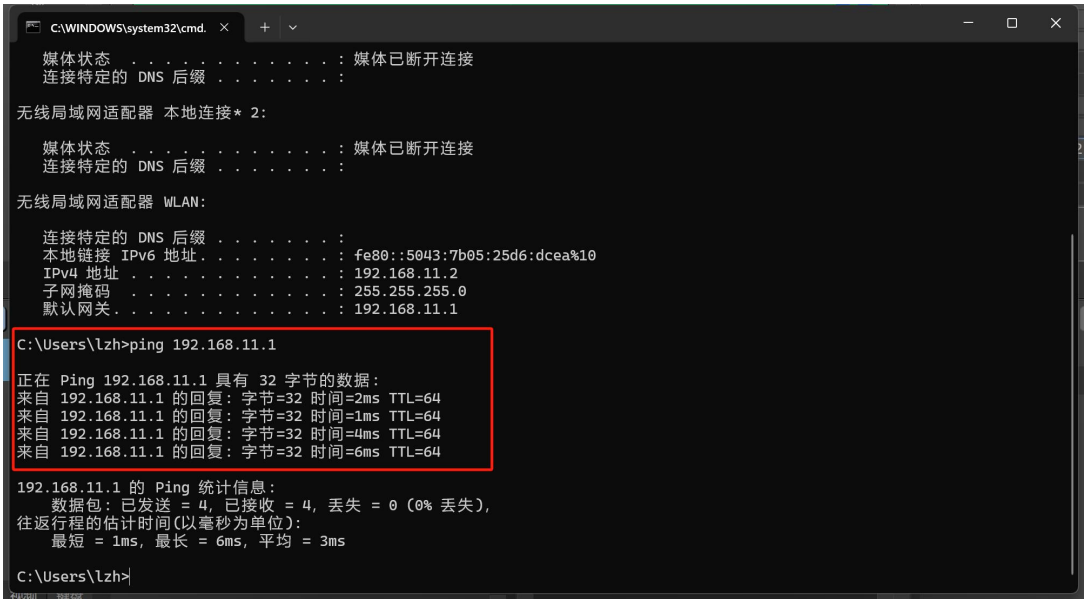
6、由于我们需要测试 TCP 通信，屏作为了一个 TCP 服务器端，那我们至少还需要一个 TCP 客户端。这里为了便于测试，我们使用电脑的网络调试助手作为一个 TCP 客户端。用电脑连上串口屏发出的热点。

7、查询电脑本地 IP 地址。 win+R 弹窗输入 cmd 然后点击确定，输入 ipconfig 回车，可以看到本地的 IP 地址（连接网络的情况下才能查询到 IP 地址）。如下图



可以看到测试的电脑连上热点后的 IP 地址为 192.168.11.2。所以我们需要在 第二页的配置界面 设置远端地址 remoteAddr 为 192.168.11.2(客户在验证时需要修改为自己的 IP 地址，以实际为准)

8、可以在电脑的 cmd 窗口，ping 显示屏的 IP 地址，这里为 ping 192.168.11.1，测试电脑和显示屏网络是否正常。



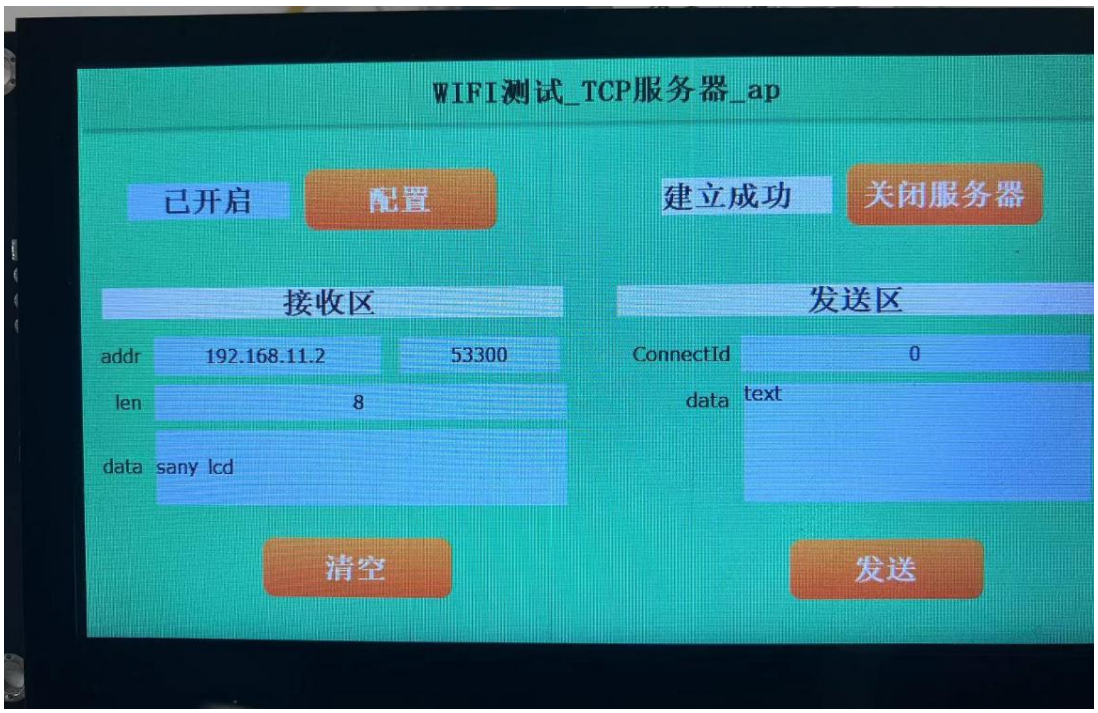
9、在主界面上，点击按钮，建立一个 TCP 服务器

10、打开网络调试助手。协议类型选中 TCP client，远程主机地址选择 192.168.11.1，远程端口号选择 10001（以配置界面，显示屏的 IP 地址和端口为准）。点击连接。

11、网络调试助手发送字符串 sany lcd。串口屏则会收到 TCP 报文，显示收到报文的地址为 192.168.11.2，端口号 53300，数据长度 8，数据类容为 sany lcd；

12、串口屏上点击发送按钮，网络调试将会收到对应的数据，这里收到的数据为字符串 text。

测试结果如下：



实验 14　蓝牙 BLE 通信

1.实验目的

带有蓝牙扩展模块的串口屏，可以实现蓝牙 BLE 通信。目前仅支持从机模式。
本实验演示了串口屏作为蓝牙从机，演示了以下功能

1、通过 notify 和 indicate，从机向主机发送数据
2、接收主机的发送 characteristic 类型的数据
3、设置本地服务特征值

注：

使用蓝牙 BLE 通信，需固件版本为 1.7.27 及以上。
本实验使用 800*480 分辨率型号的屏，如使用低分辨率屏，如 480*272，可手动调整控件位置至显示区域。

2.服务、服务特征介绍

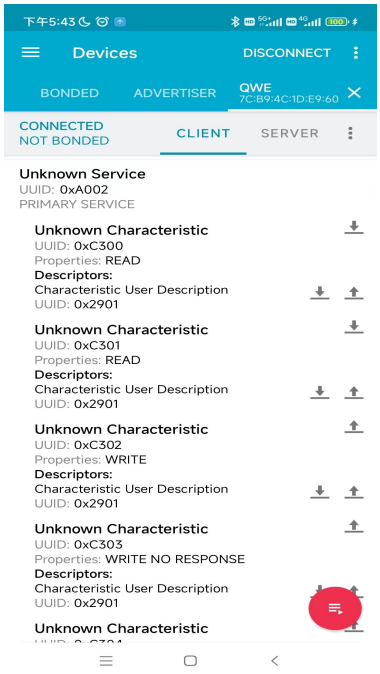
串口屏自带的蓝牙模块包含的服务、以及服务下面服务特征和描述符都是出厂时以及固化好的，用户没有权限更改，只能在现有的服务下实现通信。

每个服务以及服务特征都有固定的序号，编写脚本时，都是以序号作为操作对象。所以在确定使用某个服务和服务特征时，需要查询得到对应的序号。

服务特征及序号说明

序号	服务 uuid	服务特征信息		描述符信息	
		序号	UUID	序号	UUID
1	0xA002	1	0xc300	1	0x2901
		2	0xc301	2	0x2901
		3	0xc302	3	0x2901
		4	0xc303	4	0x2901
		5	0xc304		
		6	0xc305	6	0x2902
		7	0xc306	7	0x2902
		8	0xc307	8	0x2901
2	0xA003	1	0xc400	1	0x2901
		2	0xc401	2	0x2901

详细的信息可以通过手机软件连上蓝牙后查看，如使用蓝牙助手 nRF Connect.



3.页面设计

说明：

(1) 文本控件，显示蓝牙 BLE 名称

(2) 3 个整数控件，分别显示接收时候的 服务序号、服务特征的序号、接收数据长度

(3) 文本控件，以 ascii 方式显示接收内容

(4) 按钮控件，点击时通过 notify 从机主动向主机发送数据。这里测试的服务特征是 0xC305,服务序号 1，服务特征序号 6

```
//从机执行 notify 操作
byte a[5] = {0x31,0x32,0x33,0x34,0x35};
ble0.sendNotify(0,1,6,5,a);//连接序号 0 服务序号 1 服务特征序号 6 长度 5 数组
```

(5) 按钮控件，点击时通过 indicate，从机主动向主机发送数据。这里测试的服务特征是 0xC306,服务序号 1，服务特征序号 7

```
//从机执行 indicate 操作
byte a[5] = {0x31,0x31,0x33,0x33,0x33};
ble0.sendIndicate(0,1,7,5,a);//连接序号 0 服务序号 1 服务特征序号 7 长度 5 数组
```

(6) 按钮控件，点击时设置本地服务特征值。这里测试的服务特征是 0xC307,服务序号 1，服务特征序号 8

```
//从机设置服务特征值
byte a[1] = {0x31};
//连接序号 0 服务序号 1 服务特征序号 8  描述符序号 0 表示设置特征值 长度 1 数组
ble0.setCharacteristic(1,8,0,1,a);
```

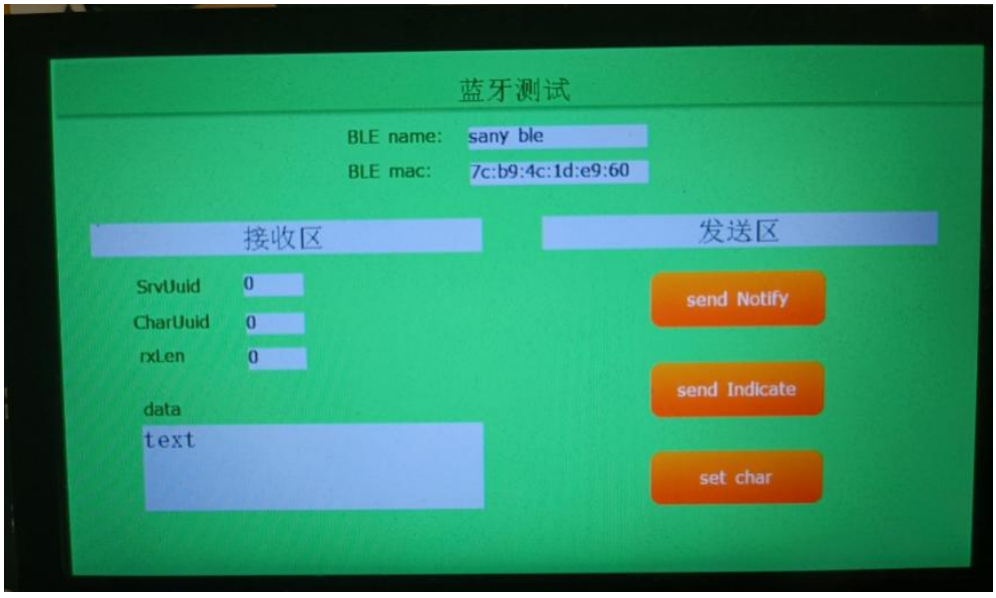
(7) 蓝牙控件，右侧可以编辑基本参数，选中蓝牙控件，可以在事件编辑器里看到接收脚本。

```
num10.val= ble0.rxSrvIndex;  //服务序号
num11.val= ble0.rxCharIndex; //服务特征的序号
num13.val= ble0.rxLen;     //接收数据的长度

//接收数据 将字节类型转换为字符串并赋值到文本控件上 显示。
text12.txt= stringDecode(ble0.rxBuf,0,ble0.rxLen);
```

4.下载验证

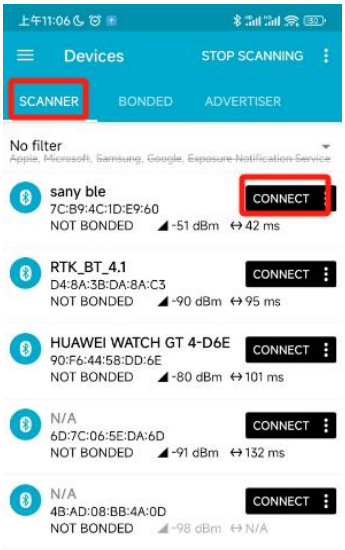
1、将工程编译后下载到 4.3 寸带显示屏(带扩展蓝牙模块)，即可看到如下界面：

注：这里工程是以 800*480 分辨率的屏设计，如果是 480*272 低分辨率，需要手动调整一下控件的位置使其在显示区域。

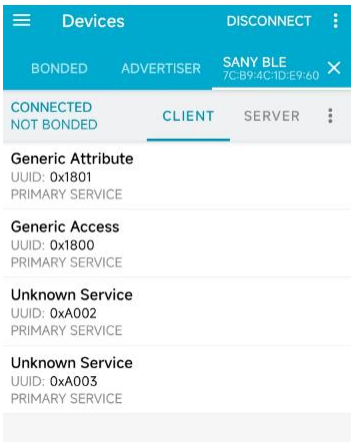


2、打开手机 APP 上的蓝牙助手 nRF Connect，点击 SCAN 开始扫描，找到 BLE 服务端的 MAC 地址或者对应的蓝牙的名称，点击 CONNECT 进行连接。

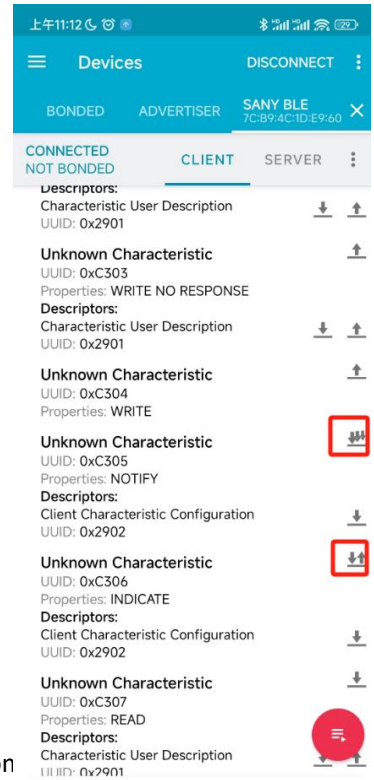
蓝牙调试助手下载链接: https://pan.baidu.com/s/1jCrL2nXxlBRERuZgy6GrJg 提取码: 3rsg



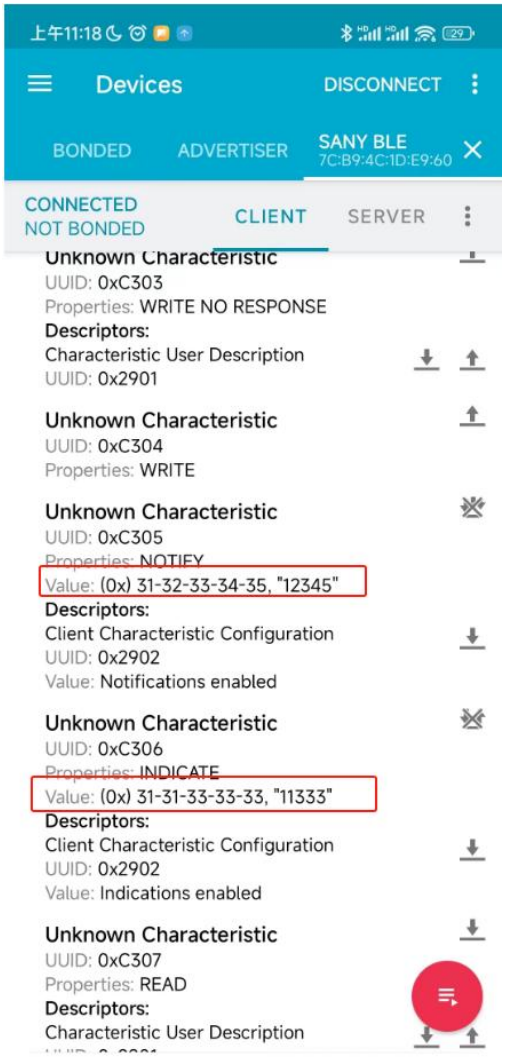
连接成功示意图如下



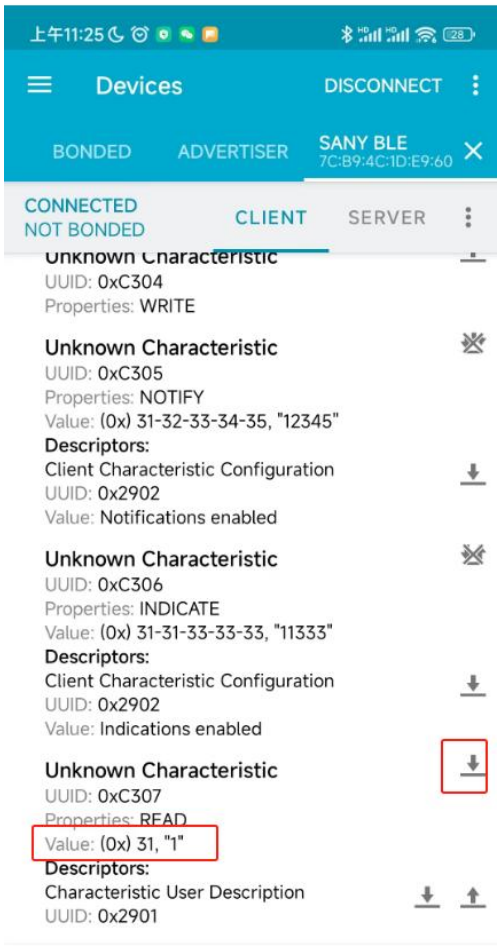
3、此时在手机 nRF 调试助手客户端的 UUID:0xA002 的 UnKnown Service 服务下一级选项中选择点击 Properties 为 NOTIFY 或者 INDICATE 的服务特征的右侧按钮，开始侦听 Properties 为 NOTIFY 或者 INDICATE 的服务特征，如右图



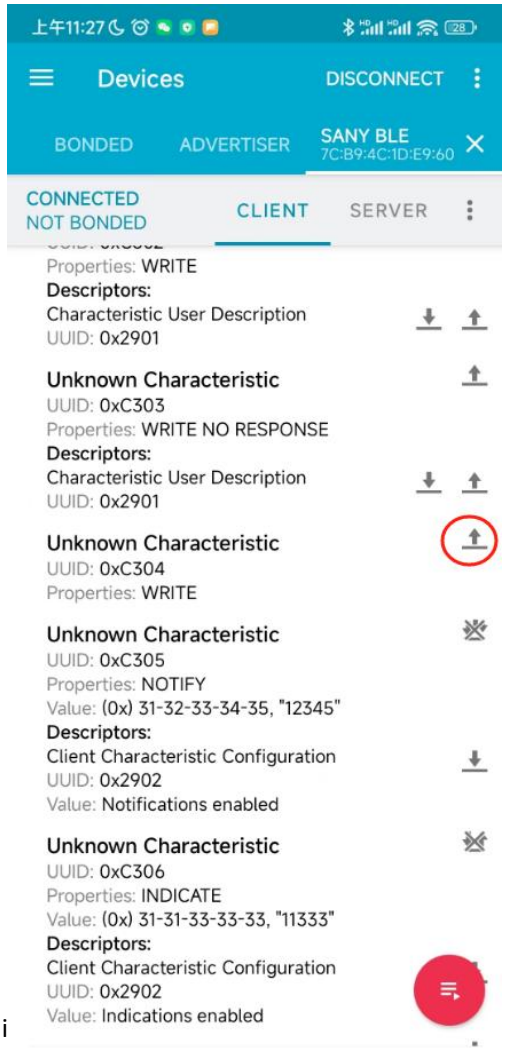
4、在屏上点击按钮【send Notify】 和【send Indicate】，既可在 nRF 蓝牙助手上收到对应的测试数据，如右图



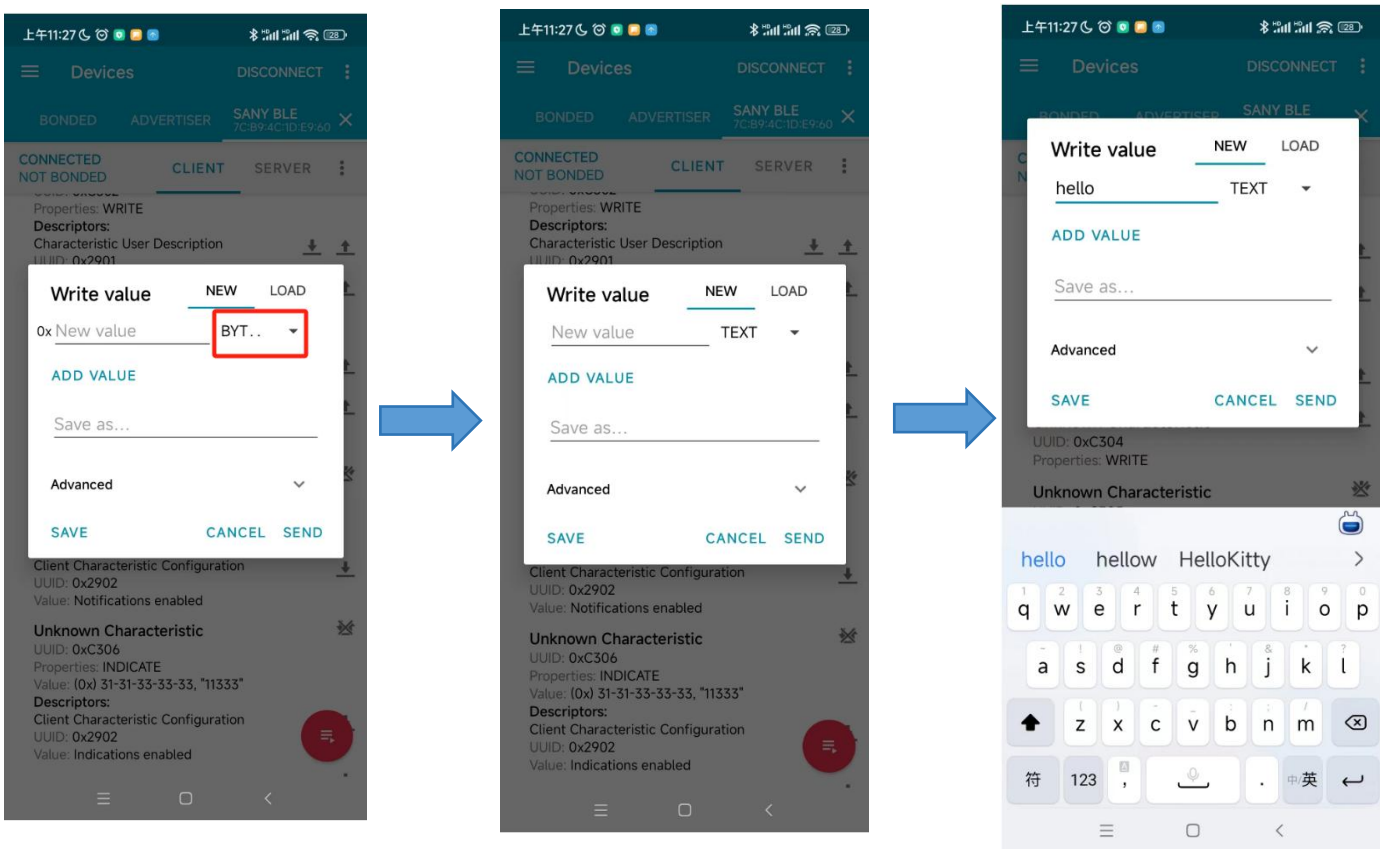
5、在 nRF 蓝牙助手上点击 UUID 为 0xC307 服务特征的读按钮，可以读到默认值为”0”，此时点击屏上的【set char】按钮设置值为”1”,nRF 蓝牙助手上重新读，则会读到值为”1”.



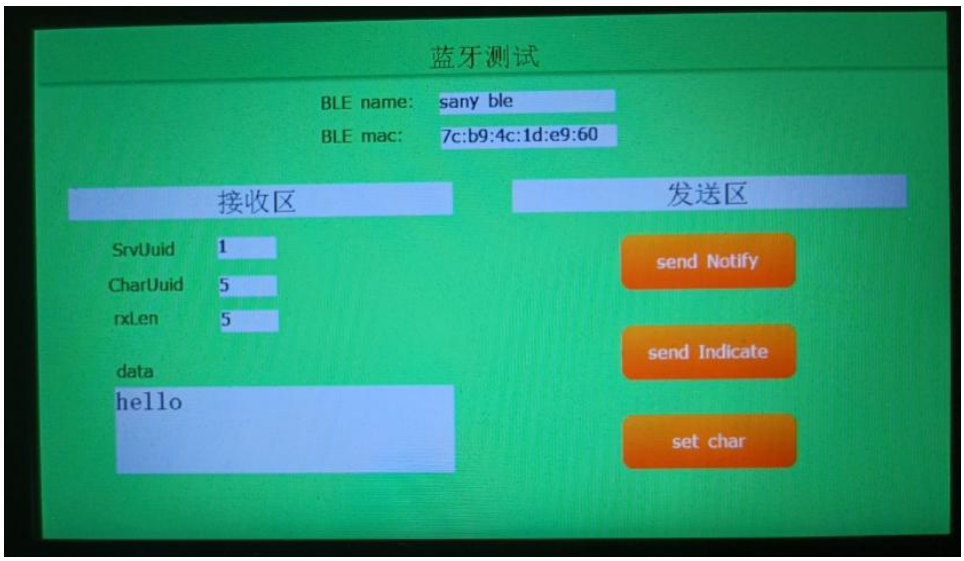
6、点击在 nRF 蓝牙助手上点击 UUID 为 0xC304 服务特征的写按钮



这里选择发送字符串类型，输入测试字符串”hello”,点击 send 发送



在显示屏上即可收到主机发送测试字符串”hello”,其中服务序号为 1，服务特征序号为 5，即服务特征为 0xC304.



谢谢您看到这里，
请愉快设计您的工程吧
资料下载：www.sany-semi.com