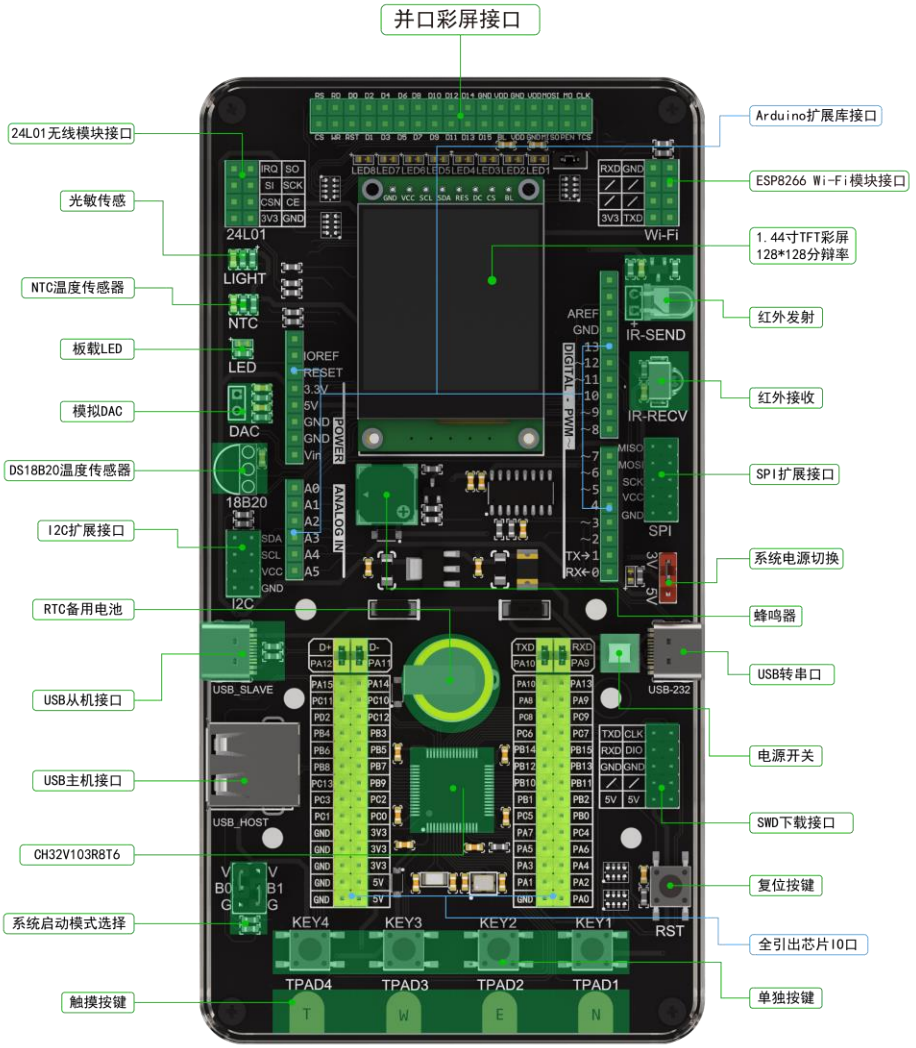
天问 CH32V 编程手册

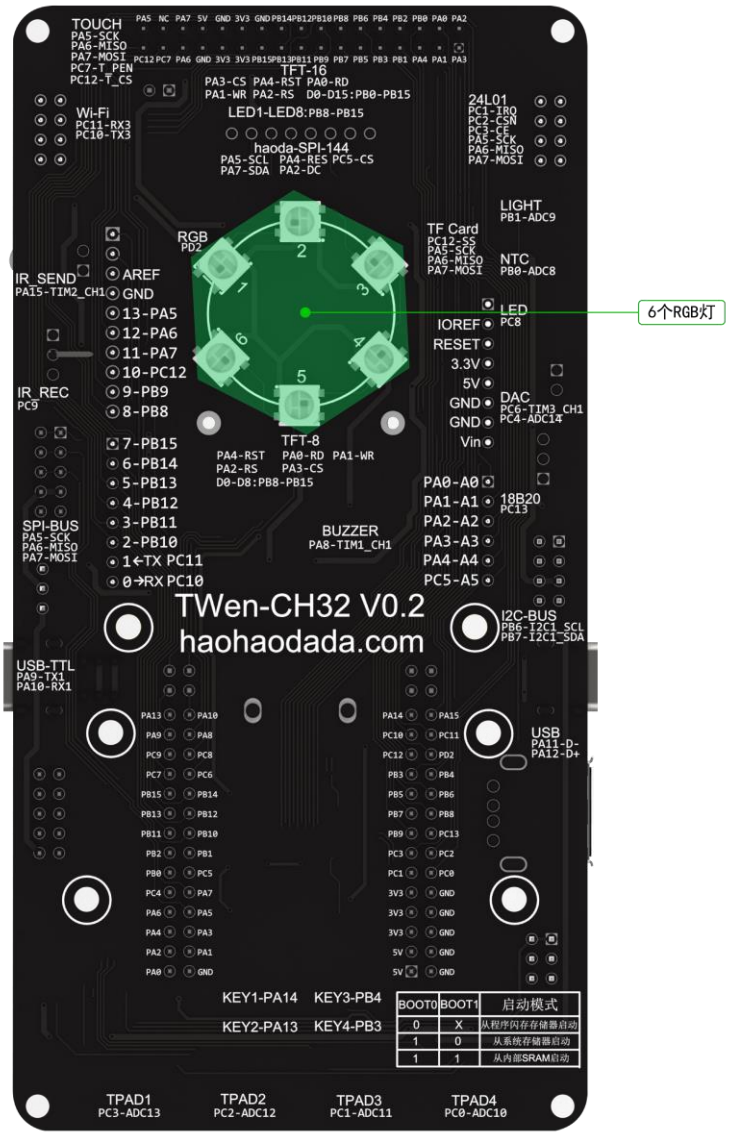
版本	更新内容
V1.0	初始版本

目录

硬件介绍

天问 CH32V 开发板是一款带 USB 的 RISC-V 32 位单片机全功能开发板，采用 CH32V103R8T6 芯片，支持 USB 主机/设备、ADC、PWM、SPI、IIC 等，板载流水灯、TFT 彩屏、红外发射、红外接收、无源蜂鸣器、4 个独立按键、4 个触摸按键、RTC 实时时钟、NTC 温度传感器、光敏传感器、6 个 RGB 彩灯等。





主板详细参数

尺寸	74*145mm
PCB 工艺	A 级 PCB，黑色油墨，沉金工艺
CPU	CH32V103R8T6 RISC-V3A 内核、64K Flash、20K SRAM、最高 80MHz 主频、GPIO*51、UART*3、USB2.0 主机/设备*1、SPI*2、I2C*2、TIMER*7、12bit ADC*16、TouchKey*16、RTC*1
电源输入	USB 5V 输入
电源输出	1117-3.3、系统电源可以通过跳线帽选择 3.3V 或者 5V
保险丝	1 个 500mA 自恢复保险丝

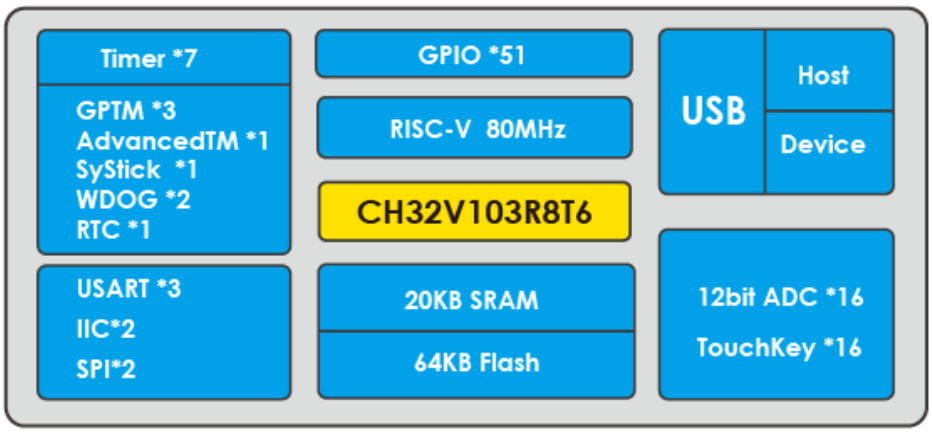
红外发射管	DY-IR333C-A
红外接收管	VS1838
光敏传感器	PT0603
热敏电阻	QN0603X103J3380HB
按键	1 个复位按键、1 个电源开关按键、4 个独立按键
LED	1 个电源 LED、八个流水灯 LED、输出比较 LED
显示屏	自带 1.44 寸 128*128 分辨率的彩屏、可以外接 2.4/4 寸触摸彩屏
蜂鸣器	1 个无源蜂鸣器
USB 转串口接口	CH340
USB 从机接口	USB2.0 协议、可模拟虚拟串口、U 盘、HID 等设备
USB 主机接口	USB2.0 协议、可外接 U 盘、鼠标等设备
Wi-Fi 扩展接口	可外接 ESP8266 Wi-Fi 模块
2.4G 扩展接口	可外 24L01 无线模块
I2C 扩展接口	可外接 2 个通用 I2C 模块
SPI 扩展接口	可外接 2 个通用 SPI 模块
Arduino 扩展接口	可外接 Arduino 扩展板

芯片介绍

概述

CH32V103 系列是以 RISC-V3A 处理器为核心的 32 位通用微控制器，该处理器是基于 RISC-V 开源指令集设计。片上集成了时钟安全机制、多级电源管理、通用 DMA 控制器。此系列具有 1 路 USB2.0 主机/设备接口、多通道 12 位 ADC 转换模块、多通道 TouchKey、多组定时器、多路 IIC/USART/SPI 接口等丰富的外设资源。

系统框图



产品特点

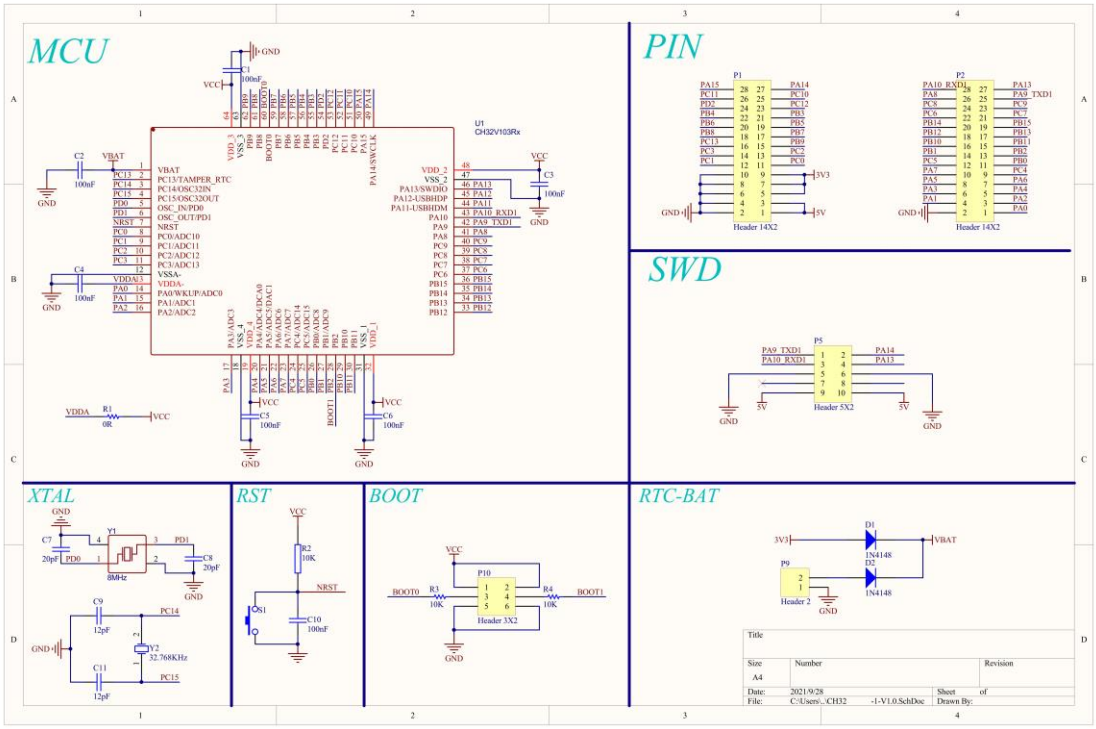
- RISC-V3A 处理器，最高 80MHz 系统主频；
- 支持单周期乘法和硬件除法；
- 20KB SRAM，64KB CodeFlash；
- 供电范围：2.7V ~ 5.5V，GPIO 同步供电电压；
- 多种低功耗模式：睡眠/停止/待机；
- 上电/断电复位（POR/PDR）；
- 可编程电压监测器（PVD）；
- 7 通道 DMA 控制器；
- 16 路 TouchKey 通道监测；
- 16 路 12 位 ADC 转换通道；
- 7 个定时器；
- 1 个 USB2.0 主机/设备接口（全速和低速）；
- 2 个 IIC 接口（支持 SMBus/PMBus）；
- 3 个 USART 接口；
- 2 个 SPI 接口（支持 Master 和 Slave 模式）；
- 51 个 I/O 口，所有的 I/O 口都可以映射到 16 个外部中断；
- CRC 计算单元，96 位芯片唯一 ID；
- 串行单线调试（SWD）接口；
- 封装形式：LQFP64M、LQFP48、QFN48。

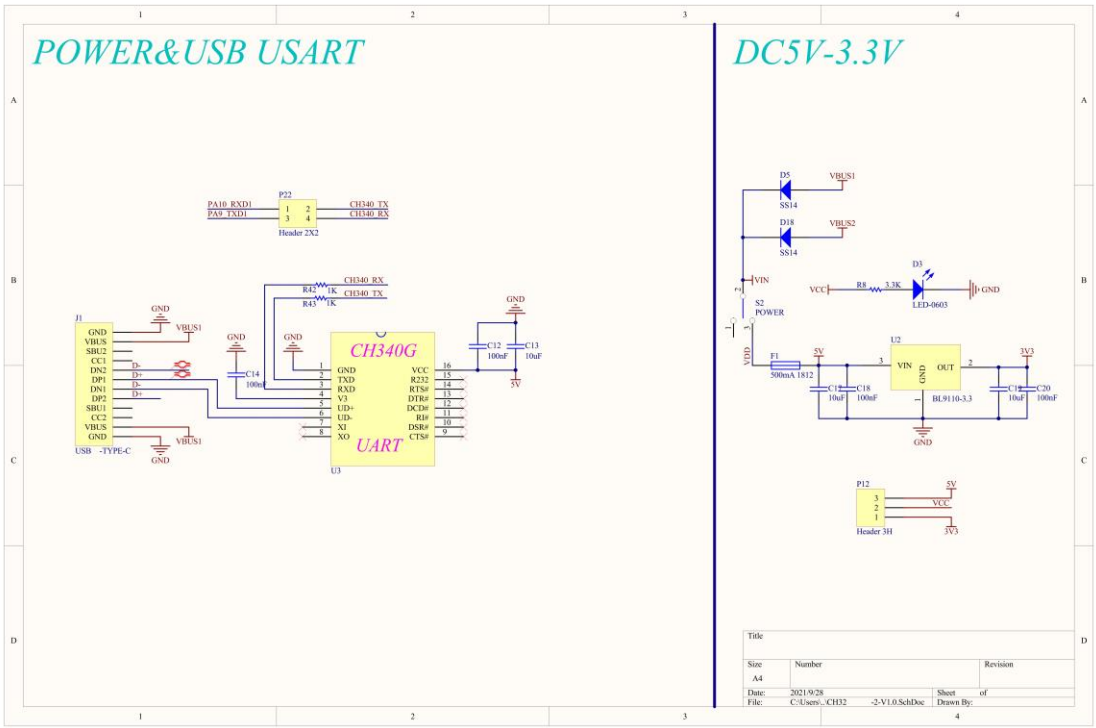
选型指南

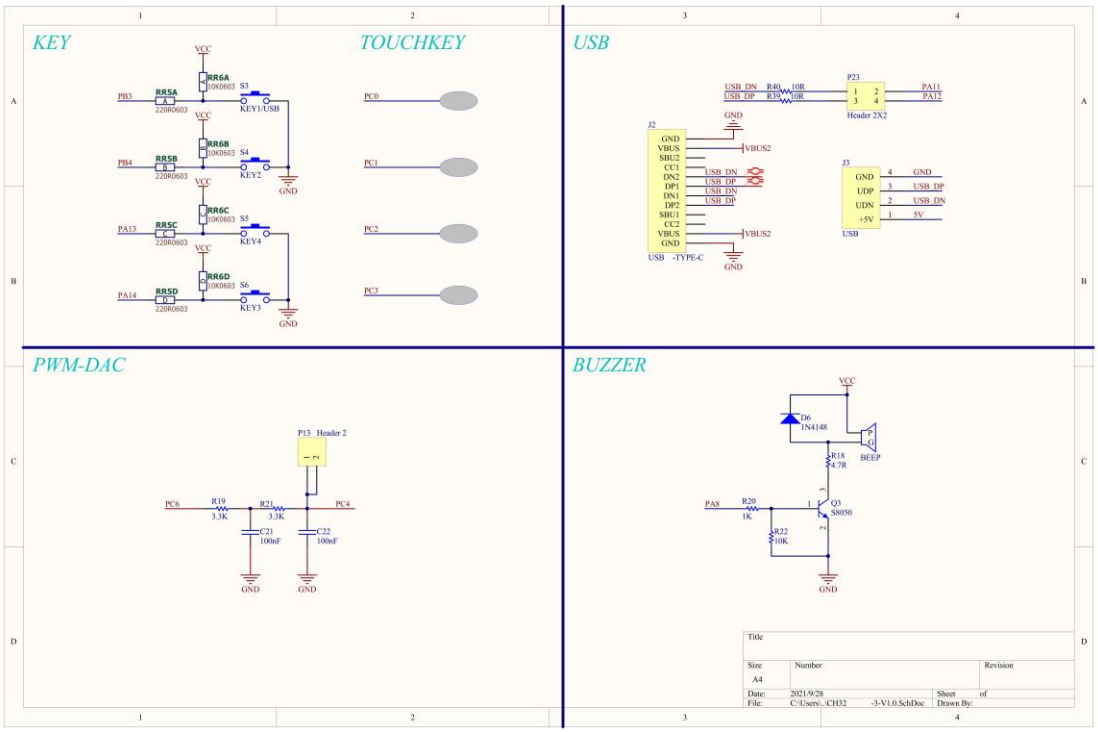
Part NO.	Flash	SRAM	Host/Device	Timer				Connectivity			ADC/ TouchKey	GPIO	VDD/V	Package
				GPTM	Advanced TM	WDOG	SysTick	SPI	I²C	USART				
CH32V103R8T6	64K	20K	1	3	1	2	1	2	2	3	16/16*12b	51	2.7~5.5	LQFP64M
CH32V103C8T6	64K	20K	1	3	1	2	1	2	2	3	10/10*12b	37	2.7~5.5	LQFP48
CH32V103C8U6	64K	20K	1	3	1	2	1	2	2	3	10/10*12b	37	2.7~5.5	QFN48
CH32V103C6T6	32K	10K	1	2	1	2	1	1	1	2	10/10*12b	37	2.7~5.5	LQFP48

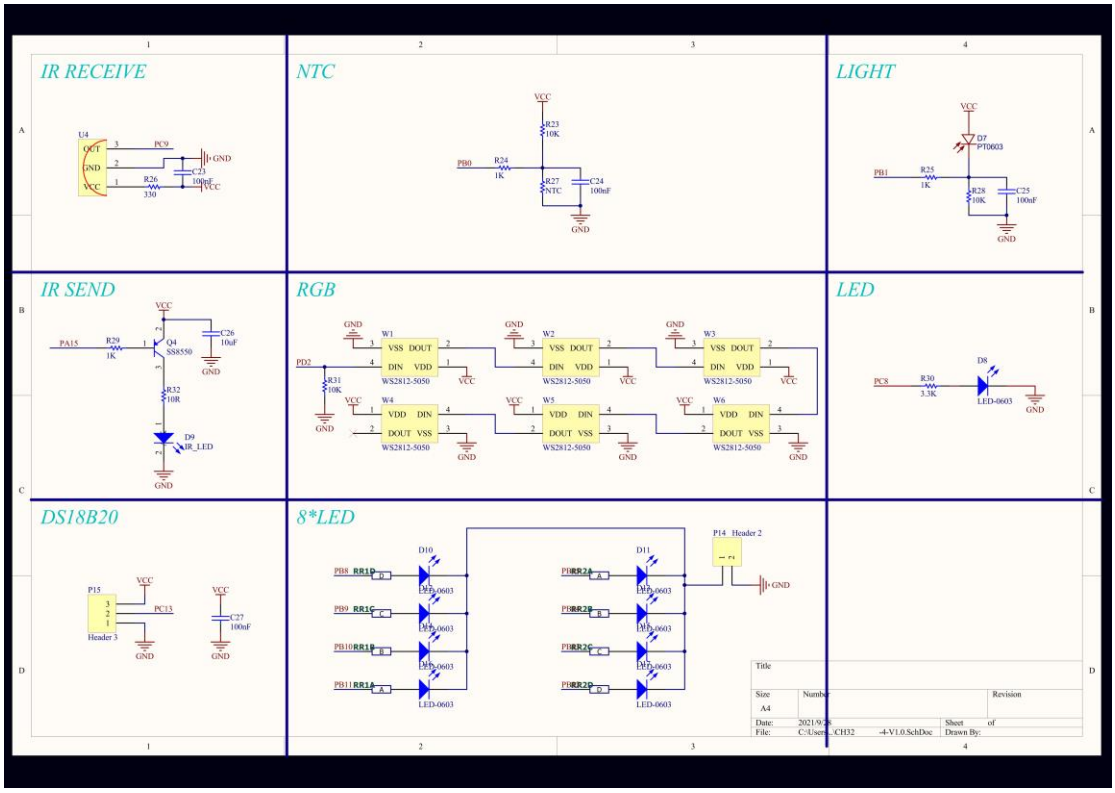
更多有关芯片的资料，请查看芯片手册和数据手册。

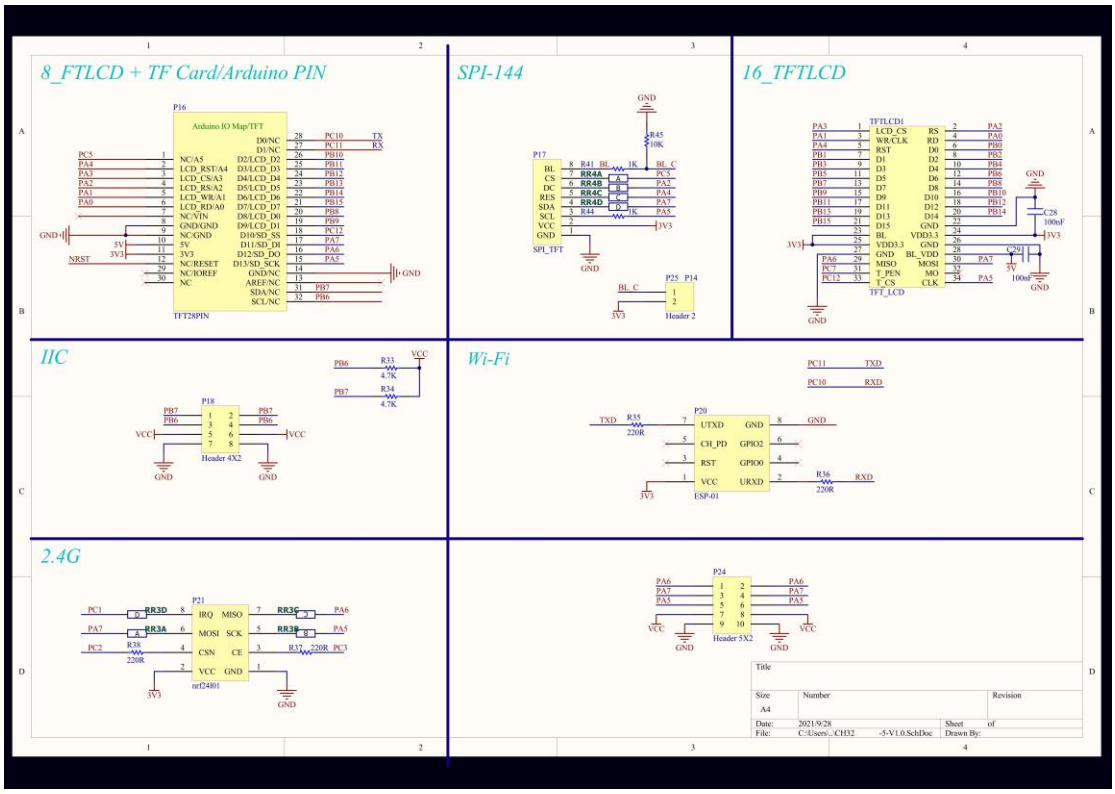
电路原理图


MCU
PIN
SWD
XTAL
RST
BOOT
RTC-BAT
U1
CH32V103Rx
P1
P2
Header 14X2
P5
Header 5X2
Y1
8MHz
Y2
32.768KHz
R1
0R
R2
10K
S1
P10
Header 3X2
D1
D2
1N4148
P9
Header 2
Title
Size
A4
Number
Revision
Date: 2021/9/28
Sheet of
File: C:\Users\...CH32 -1-V1.0.SchDoc
Drawn By:


POWER&USB USART
DC5V-3.3V
P22
Header 2X2
J1
USB -TYPE-C
CH340G
UART
U3
D5
D18
SS14
D3
LED-0603
S2
POWER
F1
500mA 1812
U2
BL9110-3.3
P12
Header 3H
Title
Size
A4
Number
Revision
Date: 2021/9/28
Sheet of
File: C:\Users\...CH32 -2-V1.0.SchDoc
Drawn By:

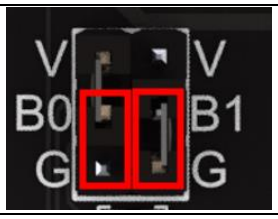
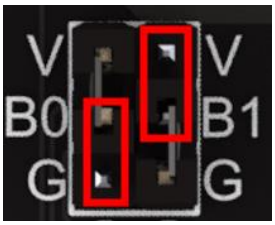

KEY
TOUCHKEY
USB
PWM-DAC
BUZZER
Title
Size
Number
Revision
A4
Date: 2021/9/28
Sheet of
File: C:\Users\...\CH32 -3-V1.0.SchDoc
Drawn By:


IR RECEIVE
NTC
LIGHT
IR SEND
RGB
LED
DS18B20
8*LED
Title
Size
Number
Revision
A4
Date: 2021/9/28
Sheet of
File: C:\Users\...\CH32 -4-V1.0.SchDoc
Drawn By:



启动模式 BOOT 引脚设置说明

由于天问出厂预置了 U 盘固件更新 Bootloader，可以通过 U 盘烧写程序。通过 BOOT 引脚的不同设置，会有两种进入方式。

BOOT0	BOOT1		U 盘模式进入方式
0	0		按住复位键插入 USB 上电，再松开复位键，进入优盘模式。 不按复位键上电直接运行 APP 区的程序。
0	1		插入 USB 进入优盘模式编程。 在线编程或天问 Block 编程，下载程序到优盘会自动运行程序，按复位键重新出现优盘。 PA14（KEY1）内置上拉，PA14（KEY1）低电平直接运行 APP 区程序。

快速上手

可以在视频学习栏目里观看开箱视频，也可以根据如下步骤操作。

第一步：开箱测试

1. 通过 USB 线连接开发板到 5V 电源，按下 KEY1 按键运行程序。
2. 彩屏会显示开机 logo 和测试菜单，根据提示依次测试各部分模块。

确认/菜单键	返回键	上键	下键
KEY4	KEY1	KEY3	KEY2

第二步：下载天问 Block 软件

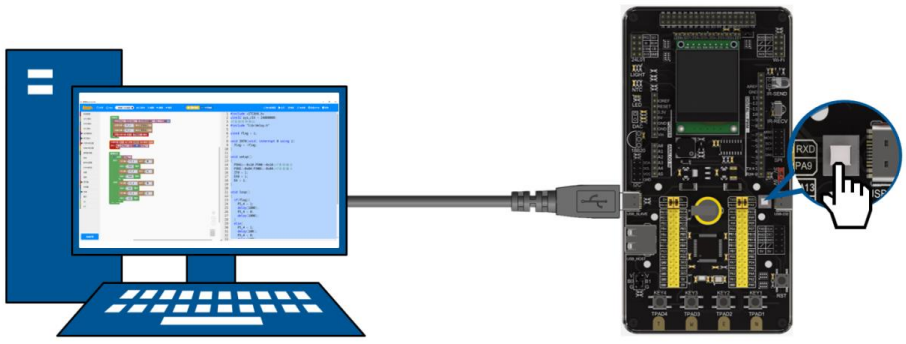
1. 浏览器打开天问官网 http://twen51.com/。
2. 点击离线软件，下载软件。

第三步：安装天问 Block 软件

根据提示默认安装，安装目录不能在“Progarm Files”和“Progarm Files (x86)”等有空格的目录，建议放磁盘根目录，默认“天问 Block”文件夹名称不能修改。

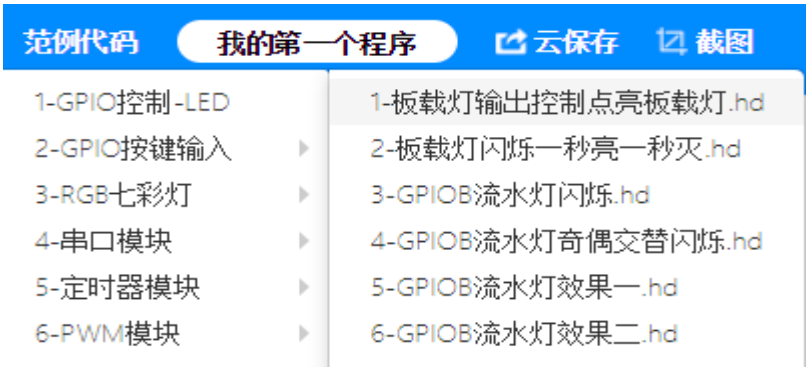
第四步：运行天问 Block 软件

1. 第一次打开软件，会让你选择主板，请选择 CH32V103。
2. 连接开发板**左侧的 USB_SLAVE** 接口到电脑，并打开电源开关。（USB-232 接口为串口）。



3. 电脑会显示一个新的 U 盘插入。（开发板出厂已经默认烧写 U 盘固件更新模式的 Bootloader 和出厂程序）。

4. 查看并打开范例程序。



5. 点击运行按钮，软件会自动编译并通过 U 盘方式下载程序到设备里。

6. 下载完成后，查看运行效果。

7. 如需要再次下载程序，需要按下 RST 复位键，U 盘会重新弹出。

8. 如需开机直接运行程序，需要按下 KEY1 按键，运行程序。

9. 如果 U 盘固件更新模式的 Bootloader 被擦除，可以点击右上角更多栏目/烧写 U 盘固件，根据提示设置 BOOT0、BOOT1 引脚来恢复固件。

更多启动模式：

设置 BOOT0 和 BOOT1 引脚全部到地，程序直接运行，如需弹出 U 盘，需要按住 RST 复位按键再开机。

软件概述

图形化编程系统是针对芯片特性进行优化，通过拖动图形块，系统会自动生成对应的 C 语言代码，关键代码都带注释说明，方便理解。生成的代码都是工程师优化过的，运行效率和直接写 C 是一致。对于没有 C 语言基础的新手快速入门，对于学过 C 语言的，只需要对照图形化生成的 C 语言代码，就能快速掌握，也可以直接用 C 语言编程。

对于 32 位单片机，天问软件框架采用 C++模式，嵌入式中的 C++，只用到了基础的类和继承，运行效率和 C 几乎一样，但由此带来了驱动层的简化和复用，编译器采用开源 GCC 编译，编译效果和商业编译器不相上下。

特点：

1. 图形化在线编程，无需记忆指令，拖动图标自动生成 C 语言完成编程;
2. 图形化编程模块中可以嵌入自定义 C 语言代码和汇编程代码，几乎可以完成所有程

序编写;

3. 图形化驱动模块，集成了常用的 GPIO、ADC、串口、PWM、定时器、外部中断、FLASH、I2C、SPI 等基本外设;

4. 除了内置的模块，可以通过扩展添加更多特定功能模块，支持个人开发自己个性化的模块库，以适应开发的需要；

5. 字符编程界面，支持内置关键字的自动补全功能，如你模糊记忆 1602 这个关键字，输入后完自动列出所有 1602 有关的函数，减少你的记忆关键字量和出错；

6. 在线版支持云编译，只要打开浏览器，就能直接编译出烧写文件，无需安装任何软件；支持云保存，文件、项目跟着网络走，也可以分享项目，方便远程交流；

7. 在线版提供健全的教学功能，支持在线教学和作业批改，编程系统和教学系统融合为一体。

对应的底层驱动库，托管在码云上，供深入学习和研究。

仓库地址：https://gitee.com/haohaodada-official/twen-51-driver-library.git

图形化界面介绍

从编程界面看，基本与通用软件一致分成**工具栏、指令区、编程区、字符代码区**四个区。

最上面一栏就是工具栏，工具栏里有最基本的文件操作、撤消、重做图标，还有串口监视器、运行、编译、字符编程等图标，每个图标对应操作的一个功能。

工具栏下面分成**指令区、编程区、字符代码区**三个并列的区。

指令区是程序指令仓库，需要编程时把指令拖动到编程区，实现编程的目的。指令区根

据指令功能可以分成单片机配置模块、C 语言程序模块、扩展模块三类。

单片机配置模块有系统配置、GPIO 模块、PWM 模块、 ADC 模块、定时器模块、串口模块、外部中断设置、所有中断设置、读写寄存器九个模块，运用这 9 个模块就可以设置单片机的所有功能，无需单片机手册就能完成配置和读写寄存器，只要读懂指令模块就可以简单方便实现单片机的各种功能。

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

扩展模块有显示器、传感器、存储、通讯、IIC 和 SPI 五个模块，这些模块是单片机基础模块的扩展，可以实现各类设备器件的图形化编程。

编程区是指令模块通过积木式编程实现程序的区域，初次打开状态里面有“初始化”和“重复执行”两个模块。程序上电后，先运行“初始化”模块中的指令，“初始化”模块中的程序只运行一次，一般是进行单片机模块初始化、配置使用。“重复执行”是单片机主要程序运行区，重复运行该模块中的各个指令，周而复始。

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

图形化介绍

图形化有四种基本类型，分别是：

1. 连接下一个块

2. 连接上一个块

3. 输出块

4. 输入块

我们可以根据凸起和凹进来判别。

基于上述基本块，演变出多种类型的模块，同一类别的模块颜色一样，可以根据颜色快速查找分类。

如下为常用的几种复合块。

1. 函数块

   里面可以放入需要执行的模块。

示例：

2. 执行块

   具体执行某一功能的模块，多个代码块，凸起和凹进地方靠近会自动吸附合并。

3. 输出块

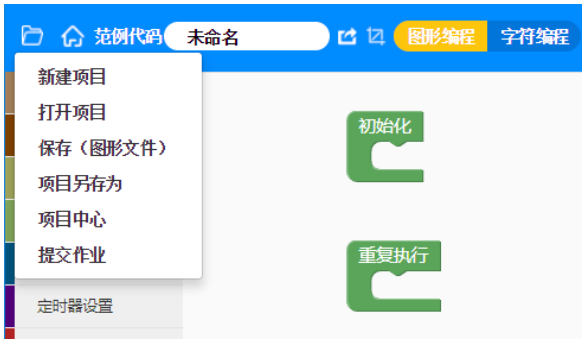
   可以是数字等常量，也可以是变量，或者是带返回值的功能块。配合输入块一起使用。

4. 输入块
   具有一个或者多个输入凹槽的块，配合输出块使用。

示例：

基本操作

一、文件操作



1. 点击新建项目按钮，会新建一个空模版，如果还有未保存的程序，会提示是否需要保存。
2. 点击打开项目按钮，会打开资源管理器，选择需要打开的文件。
3. 在输入框里可以修改文件名，点击保存按钮，会保存当前程序文件，程序文件名后缀为“.hd”。
4. 点击另存为按钮，会另存为程序文件。
5. 点击项目中心，可以看到云端的官方例程、网友共享的最新项目、自己上传到云端的我的项目。

6. 云保存，可以保存程序到云端，如果没有帐号可以免费注册，点击个人中心，可以跳转到在线版开发平台。

二、范例代码

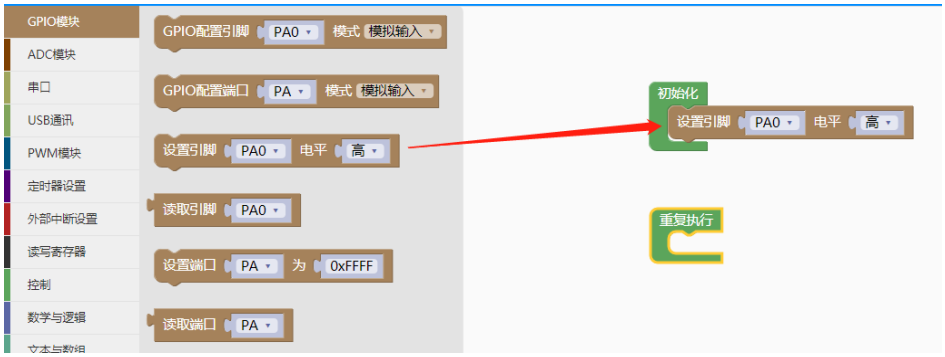
点击范例代码，选择相应的范例程序，可以查看、修改、运行范例程序。

三、图形编程和字符编程切换

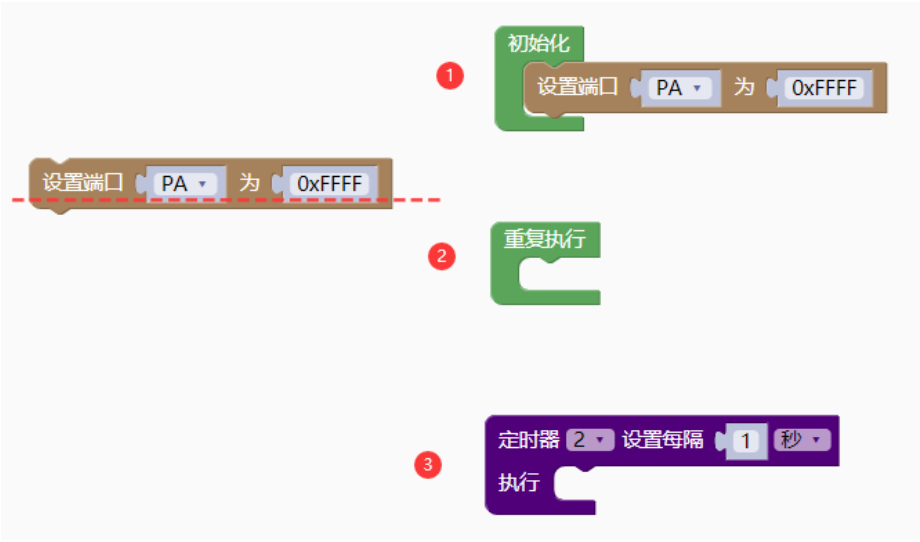
点击图形编程按钮切换到图形化编程模式，按钮变黄色；点击字符编程按钮切换到代码编程模式，按钮变黄色；

四、图形编程基本操作

1. 从指令区拖动所需图形块到编程区对应的模块下面，两个图形块靠近的时候会自动吸附。



需要注意，编程区默认有如下图所示的初始化和重复执行模块两种，外加中断函数三种图形块为第一层。其余图形化模块都只能放到这些块内部，不能放到其它空白区域，不然会导致生成的代码编译报错。



2. 复制块，可以用快捷键“Ctrl+C”，也可以右键选择复制。

3. 复制多个块，鼠标左键选中多个块的第一个块，拖动多个块移动的时候，按“Ctrl+C”，松开鼠标左键，按“Ctrl+V”粘贴块。

4. 添加注释，右键选择添加注释，点击蓝色问号，在输入框里输入注释。

5. 修改块形状，右键选择外部输入，块形状变为并列模式。

6. 缩短块的长度，右键选择折叠块。

7. 模块暂时不需要时，除了删除，还可以右键选择禁用块。

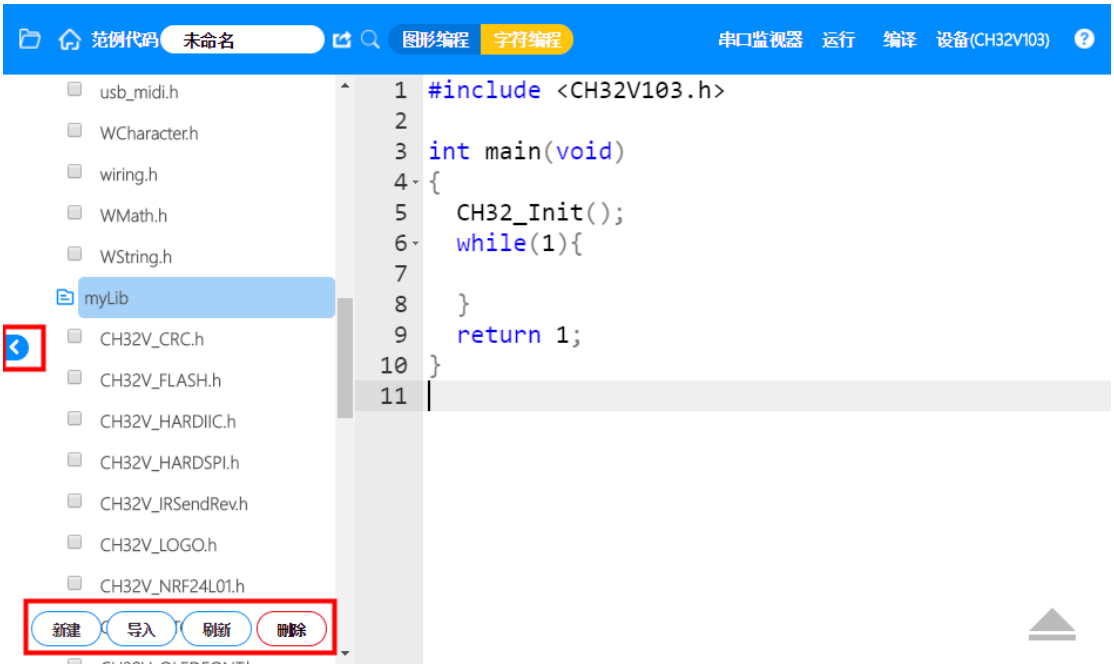
8. 删除块，可以拖动块到右下角的垃圾桶，或者左侧指令区，也可以直接按“DELETE”，也可以右键选择删除块。

9. 撤销操作
   Ctrl+Z

10. 恢复操作
    Ctrl+Shift+Z

11. 剪切操作
    Ctrl+X

五、字符编程基本操作



左侧为库文件管理区，可以通过箭头按钮，打开或者隐藏。可以新建、导入、刷新、删除对应的驱动文件。注意 lib 目录的文件不能被删除，只有 myLib 的用户文件可以删除。

六、字符编程快捷键说明

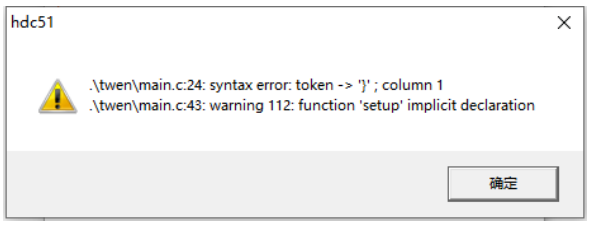
全选 selectall	Ctrl-A
查找下一个 findnext	Ctrl-K
查找上一个 findprevious	Ctrl-Shift-K
选择或查找下一个 selectOrFindNext	Alt-K
选择或查找上一个 selectOrFindPrevious	Alt-Shift-K
查找 find	Ctrl-F
选择到开头 selecttostart	Ctrl-Shift-Home
前往开头 gotostart	Ctrl-Home
向上选择 selectup	Shift-Up
向上一行 golineup	Up
选择到结尾 selecttoend	Ctrl-Shift-End
前往结尾 gotoend	Ctrl-End
向下选择 selectdown	Shift-Down
向下一行 golinedown	Down
展开/折叠 unfold/fold	Alt-Shift-L
前往这行开头 gotolinestart	Home
向左移动 gotoleft	Left
前往这行结尾 gotolineend	End
向右移动 gotoright	Right
向上滚动页面 scrollup	Ctrl-Up
向下滚动页面 scrolldown	Ctrl-Down
选择到这行开头 selectlinestar	Shift-Home

选择到这行结尾 selectlineend	Shift-End
跳转至匹配的括号 jumptomatching	Ctrl-P
选择匹配的括号 selecttomatching	Ctrl-Shift-P
扩展至匹配的括号 expandToMatching	Ctrl-Shift-M
删除一行 removeline	Ctrl-D
复制一行 duplicateSelection	Ctrl-Shift-D
排列行 sortlines	Ctrl-Alt-S
注释行 togglecomment	Ctrl-/
注释块 toggleBlockComment	Ctrl-Shift-/
将选中的数字加一 modifyNumberUp	Ctrl-Shift-Up
将选中的数字减一 modifyNumberDown	Ctrl-Shift-Down
替换 replace	Ctrl-H
撤销 undo	Ctrl-Z
重做 redo	Ctrl-Shift-Z
向上复制这一行 copylinesup	Alt-Shift-Up
将这行向上移动 movelinesup	Alt-Up
向下复制这一行 copylinesdown	Alt-Shift-Down
将这一行向下移动 movelinesdown	Alt-Down
删除 del	Delete
退格 backspace	Backspace
剪切或删除 cut_or_delete	Shift-Delete
删除至行首 removetolinestart	Alt-Backspace
删除至行尾 removetolineend	Alt-Delete
删除左边的单词 removewordleft	Ctrl-Backspace
删除右边的单词 removewordright	Ctrl-Delete
向左缩进 outdent	Shift-Tab
向右缩进 indent	Tab
整行向左缩进 blockoutdent	Ctrl-[
整行向右缩进 blockindent	Ctrl-]
转为大写 touppercase	Ctrl-U
转为小写 tolowercase	Ctrl-Shift-U
选中整行 expandtoline	Ctrl-Shift-L
在上方插入指针 addCursorAbove	Ctrl-Alt-Up
在下方插入指针 addCursorBelow	Ctrl-Alt-Down
在上方插入唯一指针 addCursorAboveSkipCurrent	Ctrl-Alt-Shift-Up
在下方插入唯一指针 addCursorBelowSkipCurrent	Ctrl-Alt-Shift-Down
选择前一个 selectNextBefore	Ctrl-Alt-Shift-Left
选择后一个 selectNextAfter	Ctrl-Alt-Shift-Right
查找全部 findAll	Ctrl-Alt-K

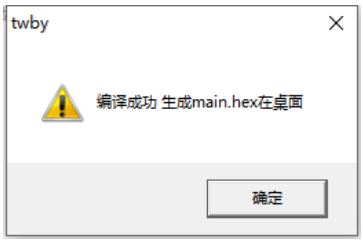
七、运行和编译

1. 程序写完后，点击运行按钮，软件会自动执行编译并下载程序到设备。

如果程序有错误，会提示错误信息。

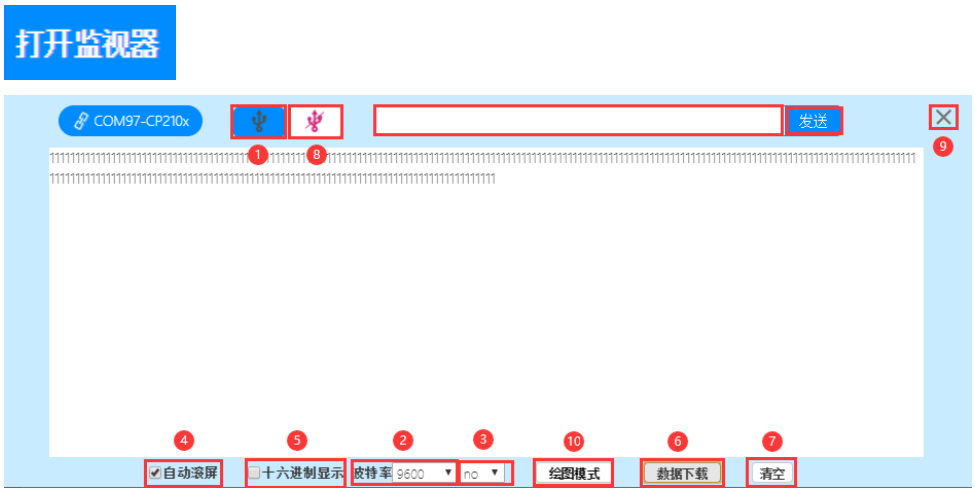


2. 如果只需要编译程序获取 HEX 文件，点击编译按钮后，会把 main.hex 文件生成在桌面。



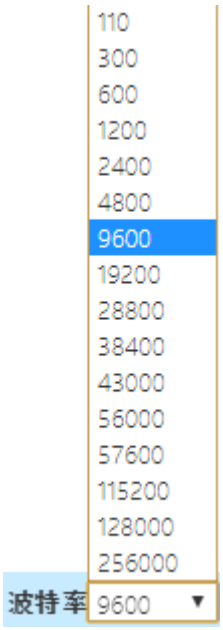
八、串口监视器

在工具栏点击打开监视器按钮，软件下放会弹出串口监视器界面。

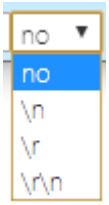


1. 启动监视器

2. 波特率选择



3. 选择发送数据时是否自动添加换行符。



4. 数据显示自动滚屏功能开关。

5. 数据显示格式切换。

6. 数据导出为 TXT 文档。

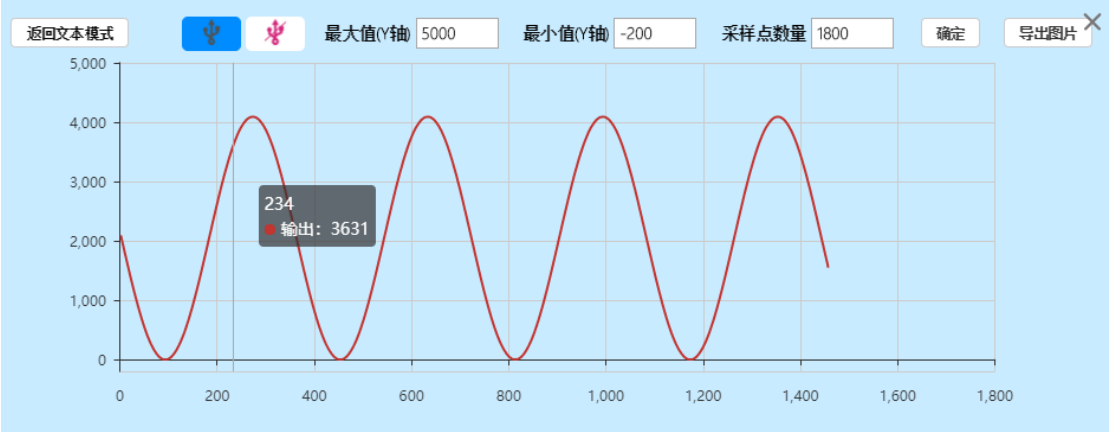
7. 清空显示数据

8. 停止串口监视器

9. 关闭串口监视器

10. 绘图模式

绘图模式下，软件根据回车换行符自动提取数据显示曲线，所以发送的数据必需添加回车换行符。



Y 坐标和采样点数量可以根据需要调整，点击确定按钮，清空数据重新开始绘图，鼠标悬停在曲线上会显示当前数据的坐标，点击导出图片按钮可以导出图片。

九、截图

在图形编程模式下，点击截图按钮，会自动把编程器的所有图形块程序保存为图片格式。

十、添加扩展

点击添加扩展按钮，弹出库管理界面。

1. 官方库

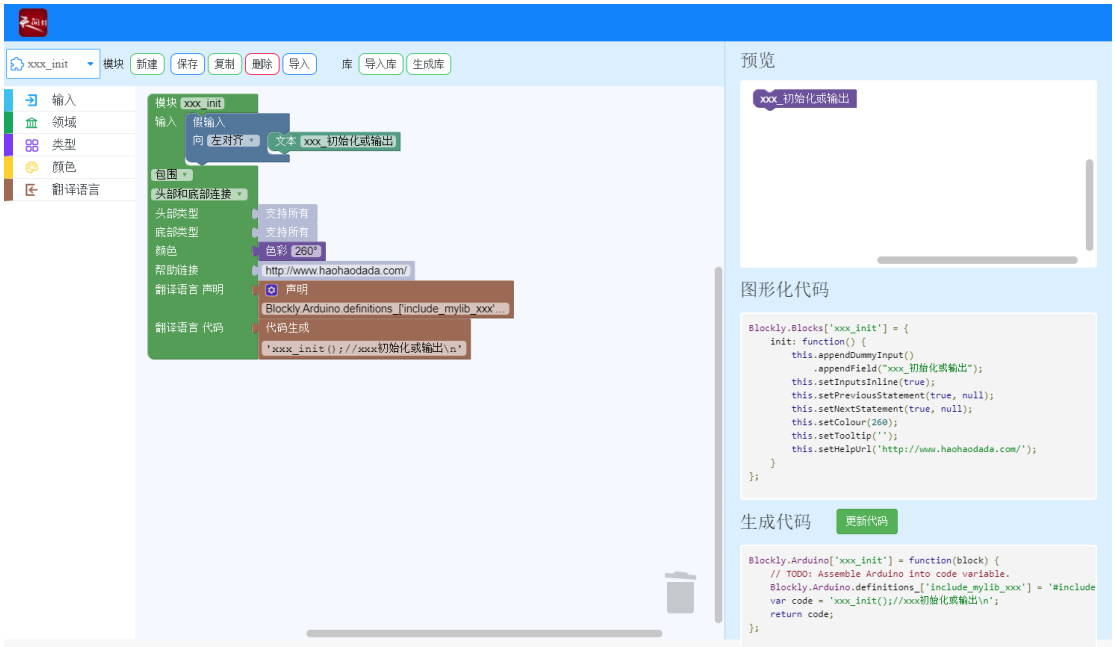
开发者开发的库提交审核后都会显示在官方库列表里。

选择需要的库，点加载图标，软件会通过网络下载库到本地，如果库版本有更新，点击更新会更新库文件，点击移除按钮可以删除库文件。点击问号按钮可以查看库的使用说明。点击星号按钮可以给库评级和反馈。

加载好的库，点击返回箭头，返回到主页面，会在指令区最下面的扩展图形化下拉列表里显示。

2. 新建库

点击新建库按钮，会进入图形化库开发工具界面，有关库的开发，请查看库开发文档和视频教程，本文不再赘述。

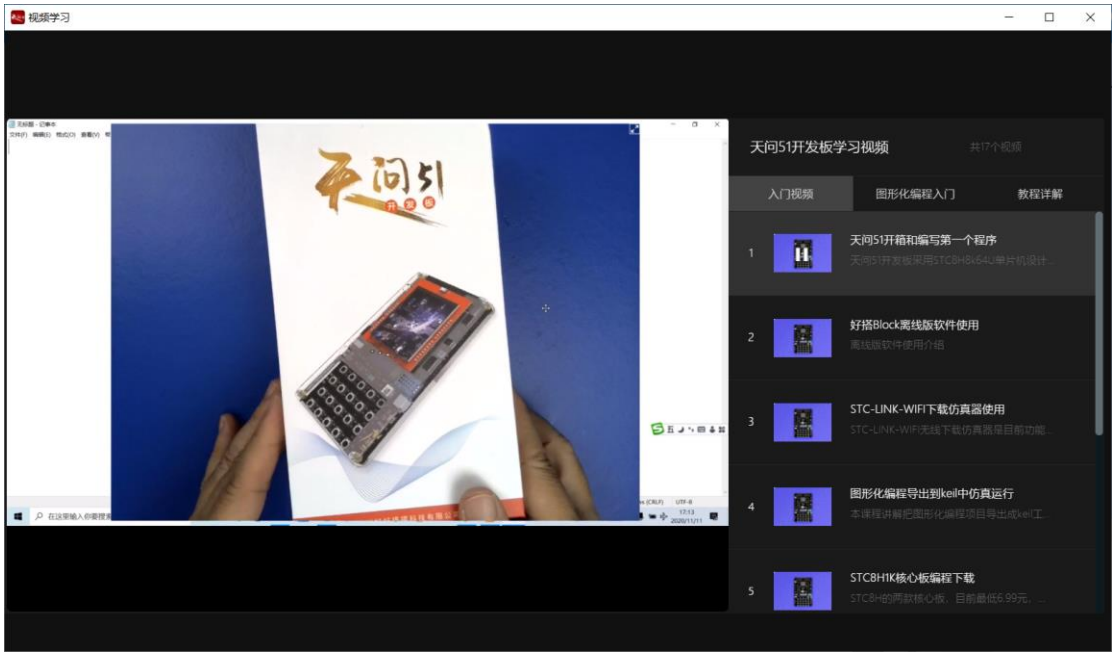


3. 用户库

用户库为提供给开发者本地测试库功能使用，测试没问题后，可以上传审核，审核通过会显示在官方库列表里。

十一、 更多功能

点击右上角更多按钮，可以查看跟多相关资料和设置。其中视频学习栏目，会自动更新配套视频教程。



基础外设模块

GPIO 模块

GPIO 口可以配置成多种输入或者是输出模式，内置可关闭的上下拉电阻，可以配置成推挽或者是开漏功能。GPIO 口还可以复用成其他功能。

端口的每个引脚都可以配置成以下的多种模式之一：

- 浮空输入
- 上拉输入
- 下拉输入
- 模拟输入
- 开漏输出
- 推挽输出

复位后，GPIO 口运行在初始状态，这时大多数 IO 口都是运行在浮空输入状态，但也有例如 JTAG 和 HSE 等外设相关的引脚是运行在外设复用的功能上。其中：

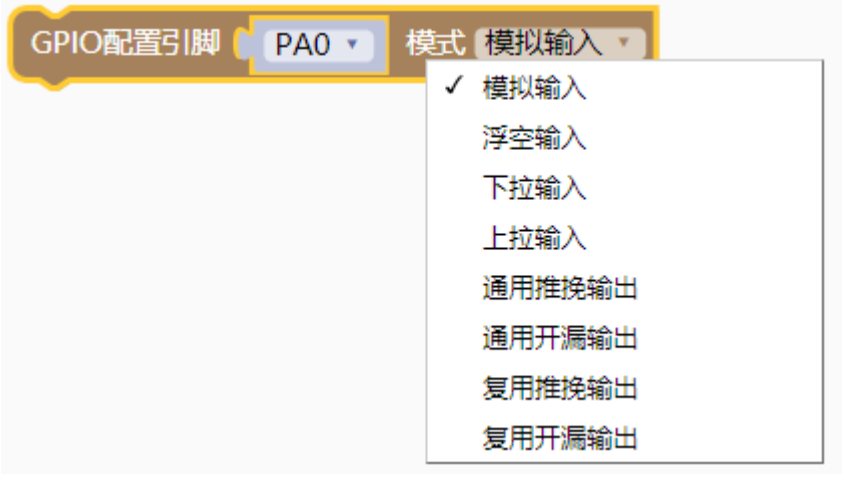
PA15 默认作为 JTDI 引脚处于上拉模式；

PA14 默认作为 JTCK 引脚处于下拉模式；

PA13 默认作为 JTMS 引脚置于上拉模式；

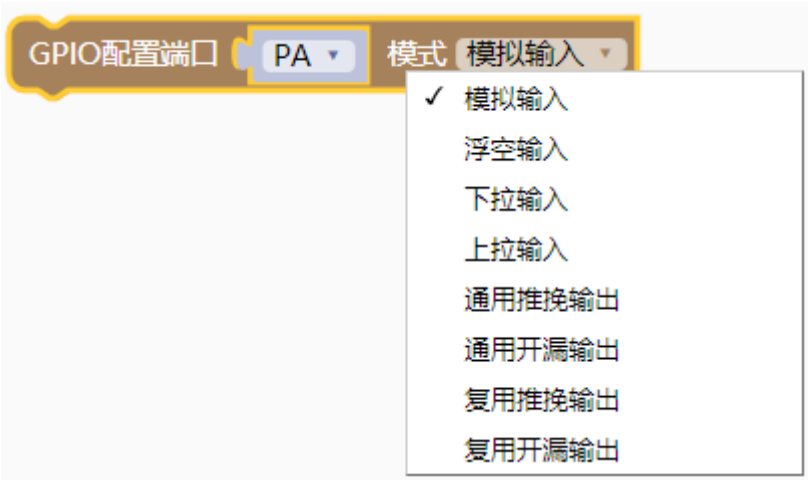
PB4 默认作为 JNTRS 引脚处于上拉模式

1. 配置引脚模式




```
pinMode(PA0, GPIO_Mode_AIN);

GPIO_Mode_AIN          0 //模拟输入
GPIO_Mode_Out_PP       1 //通用推免输出
GPIO_Mode_IPU          2 //上拉输入
GPIO_Mode_IPD          3 //下拉输入
GPIO_Mode_IN_FLOATING  4 //浮空输入
GPIO_Mode_Out_OD       5 //通用开漏输出
GPIO_Mode_AF_PP        6 //复用推免输出
GPIO_Mode_AF_OD        7 //复用开漏输出
```

2. 配置端口模式




```
CH32V_GPIOX_Init(GPIOA, GPIO_Mode_AIN, GPIO_Speed_50MHz);

GPIO_Mode_AIN          0 //模拟输入
GPIO_Mode_Out_PP       1 //通用推免输出
GPIO_Mode_IPU          2 //上拉输入
GPIO_Mode_IPD          3 //下拉输入
GPIO_Mode_IN_FLOATING  4 //浮空输入
GPIO_Mode_Out_OD       5 //通用开漏输出
GPIO_Mode_AF_PP        6 //复用推免输出
```

```
GPIO_Mode_AF_OD         7 //复用开漏输出
```

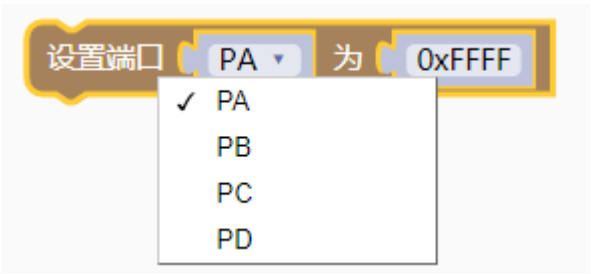
3. 设置引脚电平


```
digitalWrite(PA0, 1);//高电平：1; 低电平：0
```

4. 读取引脚电平


```
digitalRead(PA0)//高电平返回 1，低电平返回 0
```

5. 设置端口



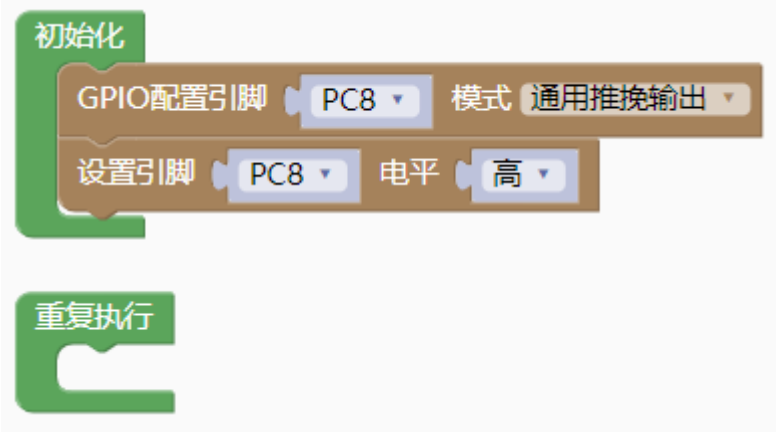

```
GPIOA->OUTDR=0xFFFF;
```

6. 读取端口


```
((uint16_t)GPIOA->INDR)
```

示例代码 1

点亮板载 LED 灯




```c
#include <CH32V103.h>

int main(void)
{
  CH32_Init();
  pinMode(PC8, GPIO_Mode_Out_PP);
  digitalWrite(PC8, 1);
  while(1){

  }
  return 1;
}
```

示例代码 2

板载 LED 灯闪烁一秒亮一秒灭


```c
#include <CH32V103.h>
```

```
int main(void)
{
  CH32_Init();
  pinMode(PC8, GPIO_Mode_Out_PP);
  while(1){
    digitalWrite(PC8, 1);
    delay(1000);
    digitalWrite(PC8, 0);
    delay(1000);
  }
  return 1;
}
```

示例代码 3

GPIOB 流水灯闪烁


```
#include <CH32V103.h>
```

```
int main(void)
{
  CH32_Init();
  pinMode(PB8, GPIO_Mode_Out_PP);
  pinMode(PB9, GPIO_Mode_Out_PP);
  pinMode(PB10, GPIO_Mode_Out_PP);
  pinMode(PB11, GPIO_Mode_Out_PP);
  pinMode(PB12, GPIO_Mode_Out_PP);
  pinMode(PB13, GPIO_Mode_Out_PP);
  pinMode(PB14, GPIO_Mode_Out_PP);
  pinMode(PB15, GPIO_Mode_Out_PP);
  while(1){
    GPIOB->OUTDR=0xFFFF;
    delay(1000);
    GPIOB->OUTDR=0x0;
    delay(1000);
  }
  return 1;
}
```

ADC 模块

ADC 模块包含一个 12 位的逐次逼近型的模拟数字转换器，最高 14MHz 的输入时钟。支持 16 个外 部通道和 2 个内部信号源采样源。可完成通道的单次转换、连续转换，通道间自动扫描模式、间断模式、外部触发模式等功能。可以通过模拟看门狗功能监测通道电压是否在阈值范围内。

主要特性

- 12 位分辨率
- 支持 16 个外部通道和 2 个内部信号源采样
- 多通道的多种采样转换方式：单次、连续、扫描、触发、间断等
- 数据对齐模式：左对齐、右对齐
- 采样时间可按通道分别编程
- 规则转换和注入转换均支持外部触发
- 模拟看门狗监测通道电压，自校准功能
- ADC 通道输入范围：0≤VIN≤VDDA

ADC 配置

1）模块上电

ADC_CTLR2 寄存器的 ADON 位为 1 表示 ADC 模块上电。当 ADC 模块从断电模式（ADON=0）下进入 上电状态（ADON=1）后，需要延迟一段时间 tSTAB 用于模块稳定时间。之后再次写入 ADON 位为 1，用于 作为软件启动 ADC 转换的启动信号。通过清除 ADON 位为 0，可以终止当前转换并将 ADC 模块置于断 电模式，这个状态下，ADC 几乎不耗电。

2）采样时钟

模块的寄存器操作基于 PCLK2（APB2 总线）时钟，其转换单元的时钟基准 ADCCLK 与 PCLK2 同步， 由 RCC_CFGR0 寄存器的 ADCPRE[1:0]域配置分频，最大不能超 14MHz。

3）通道配置

ADC 模块提供了 18 个通道采样源，包括 16 个外部通道和 2 个内部通道。它们可以配置到两种转 换组中：规则组和注入组。以实现任意多个通道上以任意顺序进行一系列转换构成的组转换。

转换组：

- 规则组：由多达 16 个转换组成。规则通道和它们的转换顺序在 ADC_RSQRx 寄存器中设置。 规则组中转换的总数量应写入 ADC_RSQR1 寄存器的 RLEN[3:0]中。
- 注入组：由多达 4 个转换组成。注入通道和它们的转换顺序在 ADC_ISQR 寄存器中设置。注 入组里的转换总数量应写入 ADC_ISQR 寄存器的 ILEN[1:0]中。

注：如果 ADC_RSQRx 或 ADC_ISQR 寄存器在转换期间被更改，当前的转换被终止，一个新的启动信号将发送到 ADC 以转换新选择的组。

2 个内部通道：

- 温度传感器：连接 ADC_IN16 通道，用来测量器件周围的温度(TA)。
- VREFINT 内部参考电压：连接 ADC_IN17 通道。

4）校准

ADC 有一个内置自校准模式。经过校准环节可大幅减小因内部电容器组的变化而造成的精准度误 差。在校准期间，在每个电容器上都会计算出一个误差修正码，用于消除在随后的转换中每个电容器上产生的误差。

通过写 ADC_CTLR2 寄存器的 RSTCAL 位置 1 初始化校准寄存器，等待 RSTCAL 硬件清 0 表示初始化完成。置位 CAL 位，启动校准功能，一旦校准结束，硬件会自动清除 CAL 位，将校准码存储到 ADC_RDATAR 中。之后可以开始正常的转换功能。建议在 ADC 模块上电时执行一次 ADC 校准。

注：启动校准前，必须保证 ADC 模块处于上电状态(ADON=1)超过至少两个 ADC 时钟周期。

5）可编程采样时间

ADC 使用若干个 ADCCLK 周期对输入电压采样，通道的采样周期数目可以通过 ADC_SAMPTR1 和 ADC_SAMPTR2 寄存器中的 SMPx[2:0]位更改。每个通道可以分别使用不同的时间采样。

总转换时间如下计算：

TCONV = 采样时间 + 12.5TADCCLK

ADC 的规则通道转换支持 DMA 功能。规则通道转换的值储存在一个仅有的数据寄存器 ADC_RDATAR 中，为防止连续转换多个规则通道时，没有及时取走 ADC_RDATAR 寄存器中的数据，可以开启 ADC 的 DMA 功能。硬件会在规则通道的转换结束时（EOC 置位）产生 DMA 请求，并将转换的数据从 ADC_RDATAR 寄存器传输到用户指定的目的地址。

对 DMA 控制器模块的通道配置完成后，写 ADC_CTLR2 寄存器的 DMA 位置 1，开启 ADC 的 DMA 功能。

注：注入组转换不支持 DMA 功能。

6）数据对齐

ADC_CTLR2 寄存器中的 ALIGN 位选择 ADC 转换后的数据存储对齐方式。12 位数据支持左对齐和右

对齐模式。

规则组通道的数据寄存器 ADC_RDATAR 保存的是实际转换的 12 位数字值；而注入组通道的数据寄存器 ADC_IDATARx 是实际转换的数据减去 ADC_IOFRx 寄存器的定义的偏移量后写入的值，会存在正负情况，所以有符号位（SIGNB）。

数据左对齐

规则组数据寄存器

D11	D10	D9	D8	D7	D6	D5	D4	D4	D2	D1	D0	0	0	0	0

注入组数据寄存器

SIGNB	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0

数据右对齐

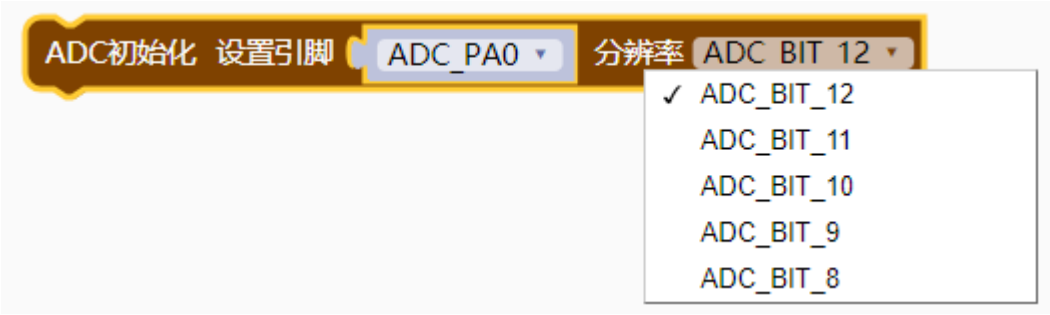
规则组数据寄存器

0	0	0	0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

注入组数据寄存器

SIGNB	SIGNB	SIGNB	SIGNB	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

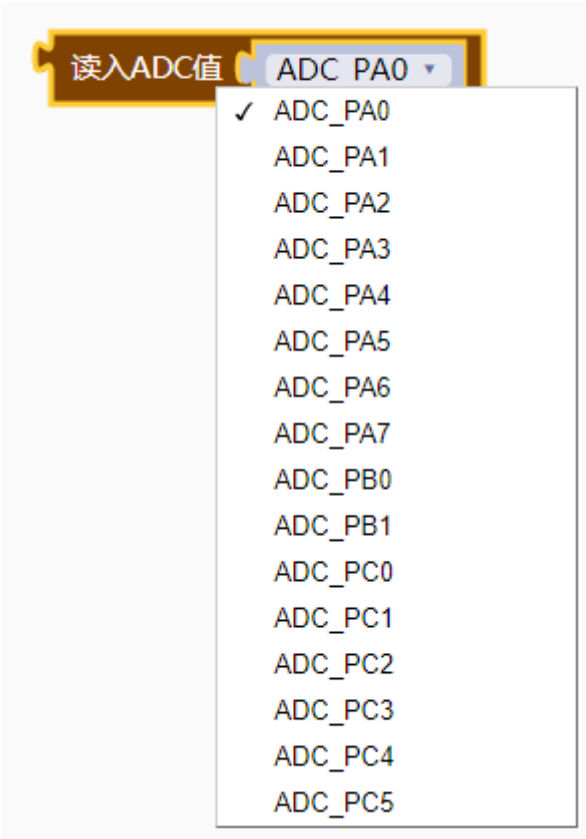
1. ADC 初始化设置引脚，分辨率。




```
CH32V_ADC_Init(ADC_PA0, ADC_SAMPLE_55_5, ADC_BIT_12);//ADC 初始化

//引脚参数范围：
ADC_PA0
ADC_PA1
ADC_PA2
ADC_PA3
```

```
ADC_PA4
ADC_PA5
ADC_PA6
ADC_PA7
ADC_PB0
ADC_PB1
ADC_PC0
ADC_PC1
ADC_PC2
ADC_PC3
ADC_PC4
ADC_PC5

//分辨率参数范围:
ADC_BIT_12          0 //12 位分辨率
ADC_BIT_11          1 //11 位分辨率
ADC_BIT_10          2 //10 位分辨率
ADC_BIT_9           3 //9 位分辨率
ADC_BIT_8           4 //8 位分辨率

//采样周期参数范围:
ADC_SAMPLE_1_5      //1.5 采样周期
ADC_SAMPLE_7_5      //7.5 采样周期
ADC_SAMPLE_13_5     //13.5 采样周期
ADC_SAMPLE_28_5     //28.5 采样周期
ADC_SAMPLE_41_5     //41.5 采样周期
ADC_SAMPLE_55_5     //55.5 采样周期
ADC_SAMPLE_71_5     //71.5 采样周期
ADC_SAMPLE_239_5    //239.5 采样周期
```

2. 读 ADC 值。




```
ADC_Read(ADC_PA0)
```

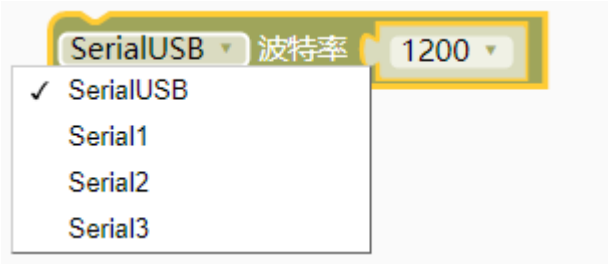
示例代码 1

USB 虚拟串口打印板载亮度传感器值

串口模块

串口模块包含两部分，一块为 USB 设备模拟的虚拟串口，另外一块为硬件串口。天问图形化端已经把两种设备简化合并在一块，方便快速开发，同时串口驱动内置环形缓存，避免数据丢失。

1. 串口初始化




```
//波特率范围 1200/4800/9600/19200/38400/57600/115200
SerialUSB.begin(9600);
Serial1.begin(9600);
Serial2.begin(9600);
Serial3.begin(9600);
```

2. 串口原始输出

写二进制数据到串口，数据是一个字节一个字节地发送的，若以字符形式发送数字请使用串口打印“print()”函数代替。

```
write()
SerialUSB.write(val)
SerialUSB.write(str)
SerialUSB.write(buf, len)
```

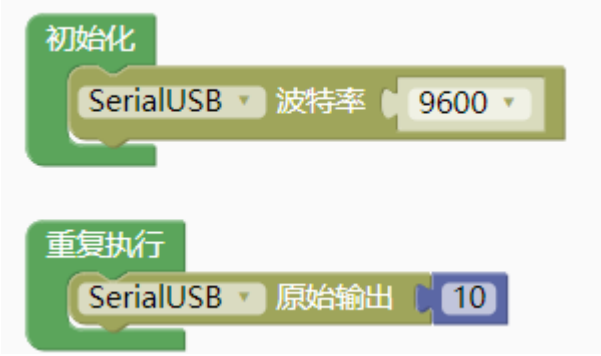
参数：
- val：作为单个字节发送的数据
- str：由一系列字节组成的字符串
- buf：同一系列字节组成的数组
- len：要发送的数组的长度

返回：
- byte

write()会返回发送的字节数，所以读取该返回值是可选的。

示例代码 1

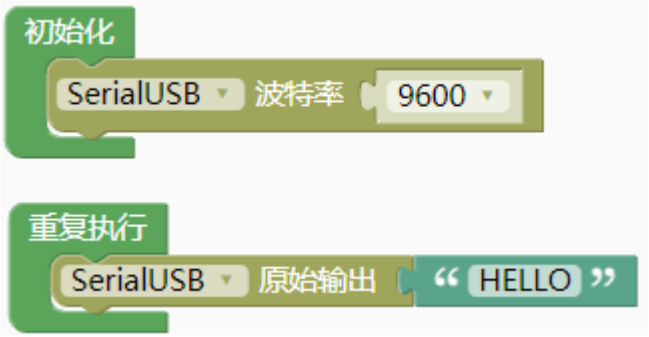
串口原始输出数字 10




```
#include <CH32V103.h>
#include "CDC.h"

int main(void)
{
  CH32_Init();
  SerialUSB.begin(9600);
  while (!SerialUSB);
  while(1){
    SerialUSB.write(10);
  }
  return 1;
}
```

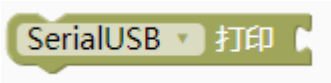
示例代码 2

串口原始输出字符串




```
#include <CH32V103.h>
#include "CDC.h"

int main(void)
{
  CH32_Init();
  SerialUSB.begin(9600);
  while (!SerialUSB);
  while(1){
    SerialUSB.write("HELLO");
  }
  return 1;
}
```

3. 串口打印



以人类可读的 ASCII 码形式向串口发送数据，无换行，该函数有多种格式。整数的每一数位将以 ASCII 码形式发送。浮点数同样以 ASCII 码形式发送，默认保留小数点后两位。字节型数据将以单个字符形式发送。字符和字符串会以其相应的形式发送。

示例代码 1

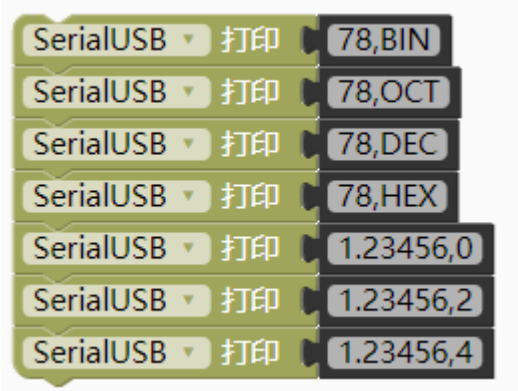
串口打印多种格式数据


```
SerialUSB.print(78) //发送 "78"
SerialUSB.print(1.23456) //发送 "1.23"
SerialUSB.print('N') //发送 "N"
SerialUSB.print("Hello world.") //发送 "Hello world."
```

可选的第二个参数用于指定数据的格式。允许的值为：BIN (binary 二进制), OCT (octal 八进制), DEC (decimal 十进制), HEX (hexadecimal 十六进制)。对于浮点数，该参数指定小数点的位数。

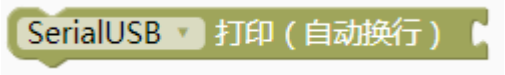
示例代码 2

串口打印指定数据的格式




```
SerialUSB.print(78, BIN) //发送 "1001110"
SerialUSB.print(78, OCT) //发送 "116"
SerialUSB.print(78, DEC) //发送 "78"
SerialUSB.print(78, HEX) //发送 "4E"
SerialUSB.print(1.23456, 0) //发送 "1"
SerialUSB.print(1.23456, 2) //发送 "1.23"
SerialUSB.print(1.23456, 4) //发送 "1.2346"
```

4. 串口打印(换行)



和串口打印一样功能，多了换行。

5. 串口打印 16 进制数(换行)

和串口打印指定格式一样功能，多了换行。

6. 串口有数据可读

获取串口上可读取的数据的字节数该数据是指已经到达并存储在接收缓存（共有 64 字节）中，如果没有数据返回-1。available()继承自 Stream 实用类。 语法：

```
SerialUSB.available()
```

7. 串口读取字符串

串口读取字符串数据

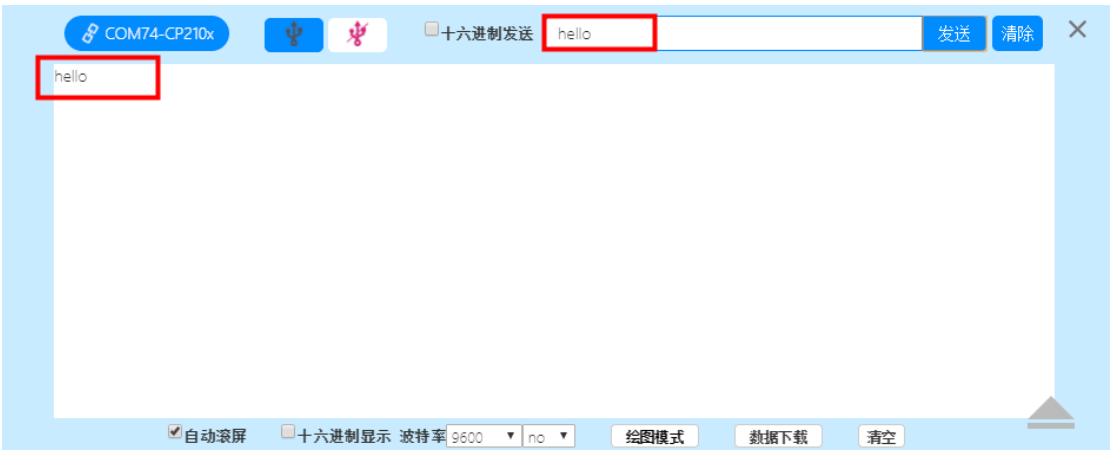
```
SerialUSB.readString()
```

示例代码 1

USB 串口输入回传


```
#include <CH32V103.h>
#include "CDC.h"

int main(void)
{
  CH32_Init();
  SerialUSB.begin(9600);
  while (!SerialUSB);
  while(1){
    if(SerialUSB.available() > 0){
      SerialUSB.println(SerialUSB.readString());
    }
```

```
  }
  return 1;
}
```



8. 串口读取字符串直到“ ”

readStringUntil 函数可用于从设备接收到的数据中读取信息。读取到的数据信息将以字符串形式返回。该函数在满足以下任一条件后都会停止函数执行并返回。

- 读取到指定终止字符
- 达到设定时间（可使用 setTimeout 函数来设置）

当函数读取到终止字符后，会立即停止函数执行。此时函数所返回的字符串为"终止字符"前的所有字符信息。

示例代码 1

USB 串口输入回传


```
#include <CH32V103.h>
#include "CDC.h"
```

```
int main(void)
{
  CH32_Init();
  SerialUSB.begin(9600);
  SerialUSB.setTimeout(5000);
  while (!SerialUSB);
  while(1){
    if(SerialUSB.available() > 0){
      SerialUSB.println(SerialUSB.readStringUntil('l'));
    }
  }
  return 1;
}
```

可以看到发送字符串“hello”后，程序检测到了‘l’字符前面的“he”，至于后面跟了一个‘o’为超时返回的数据，有关超时的说明，见 setTimeout 函数说明。

9. 串口 read 函数

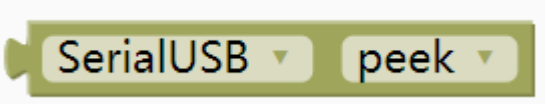
read() 函数可用于从设备接收到数据中读取一个字节的数据。一般和 write 配套使用。设备接收到数据时，返回值为接收到的数据流中的 1 个字符。

示例代码 1

USB 串口输入回传


```
#include <CH32V103.h>
#include "CDC.h"

int main(void)
{
  CH32_Init();
  SerialUSB.begin(9600);
  while (!SerialUSB);
  while(1){
    if(SerialUSB.available() > 0){
      SerialUSB.write(SerialUSB.read());
    }
  }
  return 1;
}
```

10. 串口 peek()函数



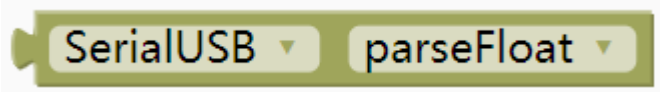
peek()函数可用于从设备接收到的数据中读取一个字节的数据。但是与 read 函数不同的是，使用 peek()函数读取数据后，被读取的数据不会从数据流中消除。这就导致每一次

调用 peek 函数，只能读取数据流中的第一个字符。然而每一次调用 read 函数读取数据时，被读取的数据都会从数据流中删除。

11. 串口 parseInt()函数

parseInt()函数可用于从设备接收到的数据中寻找整数数值。

12. 串口 parseFloat()函数



parseFloat()函数可用于从设备接收到的数据中寻找浮点数值。

13. 串口清空缓存

flush 函数可让开发板在所有待发数据发送完毕前，保持等待状态。

注意：很多人误认为 flush 函数具有清除开发板接收缓存区的功能。事实上此函数是没有此功能的。翻译名称为“清空缓存”，不是把缓存里的数据删除，而是等待缓存里的数据发送完毕。

本函数属于 Stream 类。该函数可被 Stream 类的子类所使用，如（Serial, WiFiClient, File 等）。

为了更好的理解 flush 函数的作用，我们在这里用 Serial.flush()作为示例讲解。

当我们通过 Serial.print 或 Serial.println 来发送数据时，被发送的字符数据将会存储于开发板的“发送缓存”中。这么做的原因是开发板串行通讯速率不是很高，如果发送数据较多，发送时间会比较长。

在没有使用 flush 函数的情况下，开发板不会等待所有“发送缓存”中数据都发送完毕再执行后续的程序内容。也就是说，开发板是在后台发送缓存中的数据。程序运行不受影响。

相反的，在使用了 flush 函数的情况下，开发板是会等待所有“发送缓存”中数据都发送完毕以后，再执行后续的程序内容。

14. 串口 setTimeout()函数

用于设置设备等待数据流的最大时间间隔。

当设备在接收数据时，是以字符作为单位来逐个字符执行接收任务。由于设备无法预判即将接收到的信息包含有多少字符，因此设备会设置一个等待时间。默认情况下，该等待时间是 1000 毫秒。

举例来说，假设我们要向设备发送一个字符串“ok”。那么设备在接收到第一个字符“o”以后，他会等待第二个字符的到达。假如在 1000 毫秒内，设备接收到第二个字符“k”，那么设备会重置等待时间，也就是再等待 1000 毫秒，看一看字符“k”后面还有没有字符到达。我们知道我们发给设备的字符串只有两个字符，后面没有更多字符了。但是设备并不知道这一情况。因此设备在接收到“k”以后，会等待 1000 毫秒。直到 1000 毫秒等待时间结束都没有再次接到字符。这时，设备才会很肯定地结束这一次接收工作。这里这个等待的 1000 毫秒时间就是通过 setTimeout 函数来设置的。

15. 串口 end()函数

禁用串口功能，设置串口的 TX、RX 引脚作为普通引脚使用，如需要恢复，使用 begin()函数。

16. 串口 readBytes(buffer, length)函数

readBytes(buffer, length)函数可用于从设备接收的数据中读取信息。读取到的数据信息将存放在缓存变量中。该函数在读取到指定字节数的信息或者达到设定时间后都会停止函数执行并返回。该设定时间可使用 setTimeout 来设置。

参数

buffer: 缓存变量/数组。用于存储读取到的信息。允许使用 char 或者 byte 类型的变量或数组。

length: 读取字节数量。readBytes 函数在读取到 length 所指定的字节数量后就会停止运行。允许使用 int 类型。

返回值

buffer(缓存变量)中存储的字节数。数据类型：size_t

示例代码 1

```
#include <CH32V103.h>
#include "CDC.h"
const int bufferLength = 10;     // 定义缓存大小为 10 个字节
char serialBuffer[bufferLength];// 建立字符数组用于缓存

int main(void)
```

```
{
  CH32_Init();
  SerialUSB.begin(9600);
  while (!SerialUSB);
  while(1){
    if (SerialUSB.available()){                    // 当串口接收到信息后
    SerialUSB.println("Received SerialUSB Data:");
    SerialUSB.readBytes(serialBuffer, bufferLength);// 将接收到的信息使用
readBytes 读取
    for(int i=0; i<bufferLength; i++){      // 然后通过串口监视器输出
readBytes
      SerialUSB.print(serialBuffer[i]);          // 函数所读取的信息
    }
    SerialUSB.println("");
    SerialUSB.println("Finished Printing Recevied Data.");
  }
  return 1;
}
```

17. 串口 readBytesUntil(character, buffer, length)函数

readBytesUntil(character, buffer, length)函数可用于从设备接收到数据中读取信息。读取到的数据信息将存放在缓存变量中。该函数在满足以下任一条件后都会停止函数执行并且返回。

– 读取到指定终止字符
– 读取到指定字节数的信息
– 达到设定时间（可使用 setTimeout 来设置）

当函数读取到终止字符后，会立即停止函数执行。此时 buffer（缓存变量/数组）中所存储的信息为设备读取到终止字符前的字符内容。

参数

character: 终止字符。用于设置终止函数执行的字符信息。设备在读取数据时一旦读取到此终止字符，将会结束函数执行。允许使用 char 类型。

buffer: 缓存变量/数组。用于存储读取到的信息。允许使用 char 或者 byte 类型的变量或数组。

length: 读取字节数量。readBytes 函数在读取到 length 所指定的字节数量后就会停止运行。允许使用 int 类型。

返回值

buffer(缓存变量)中存储的字节数。数据类型：size_t

18. 串口 find(target)函数

find 函数可用于从设备接收到的数据中寻找指定字符串信息。当函数找到了指定字符串信息后将会立即结束函数执行并且返回“真”。否则将会返回“假”。

参数

target: 被查找字符串。允许使用 String 或 char 类型。

返回值

返回值类型为 bool。当函数找到了指定字符串信息后将会立即结束函数执行并且返回“真”。否则将会返回“假”。

示例代码 1

```
#include <CH32V103.h>
#include "CDC.h"

int main(void)
{
  CH32_Init();
  SerialUSB.begin(9600);
  while (!SerialUSB);
  while(1){
    if (SerialUSB.available()){                    // 当串口接收到信息
后
      SerialUSB.println("SerialUSB Data Available..."); // 通过串口监视器
通知
                                                   // 用户系统开始查找指定
信息
      SerialUSB.print("system is trying to find "); SerialUSB.println("
^_^");

      // 执行查找并通过串口监视器输出查找结果
      if(SerialUSB.find("^_^")) {
        SerialUSB.print("Great! System found "); SerialUSB.println("^_^
");
      } else {
```

```
        SerialUSB.print("Sorry System can't find "); SerialUSB.println(
"^_^");
      }
      SerialUSB.println("");
    }
  }
  return 1;
}
```

19. 串口 findUntil (target ,terminator)函数

findUntil(target ,terminator)函数可用于从设备接收到的数据中寻找指定字符串信息。当函数找到了指定字符串信息后将会立即结束函数执行并且返回“真”。否则将会返回“假”。该函数在满足以下任一条件后都会停止函数执行

- 读取到指定终止字符串
- 找到了指定字符串信息
- 达到设定时间（可使用 setTimeout 来设置）

参数

target: 被查找字符串。允许使用 String 或 char 类型。

terminator: 终止字符串。用于设置终止函数执行的字符串信息。设备在读取数据时一旦读取到此终止字符串，将会结束函数执行并返回。

返回值

返回值类型为 bool。当函数找到了指定字符串信息后将会立即结束函数执行并且返回“真”。否则将会返回“假”。

USB 通讯模块

CH32V 的 USB 支持主机模式和从机模式。主机模式可以外接 U 盘、鼠标、键盘等。从机模式可以设置为虚拟串口、HID、U 盘等设备。

因为 USB 比较复杂，天问软件框架已经封装了 USB 底层驱动，简化了开发模式。

有关 USB 的详细资料，请查看专门的对应资料，本文不做过多赘述。

1. 初始化 USB 设备为鼠标键盘 HID 设备，并等待和主机连接成功。


```
usb_keymouse_init();
  _configured = 0;
  while (bDeviceState != CONFIGURED);
  if (_configured == 0) {
    delay(2000);
```

```
    _configured = 1;
  }
```

2. USB 键盘发送数值


```
keyBoard_Number(123);
```

3. USB 键盘发送文本


```
keyBoard_String("abcd");
```

4. USB 键盘打开指定程序
   内部原理为通过发送键盘快捷键打 Windows 系统自带的应用程序


```
keyBoard_Run("calc");
// calc 计算器
// notepad 记事本
// mspaint 画图板
// write 写字板
// dvdplay Media Player
// SnippingTool 截图工具
```

5. USB 键盘发送功能按键

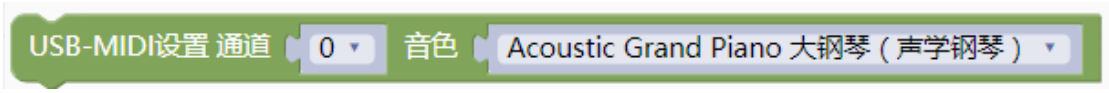

```
keyBoard_Send(0,0,0,0,0,0,0,0);
```

   按键值可以查看键盘码表，也可以对照图形化查看自动生产

6. USB 鼠标操作


```
Moue_Send(key,x,y,wheel);
//KEY 0：无按键按下，1：左键按下，2：右键按下，4：中间按下
```

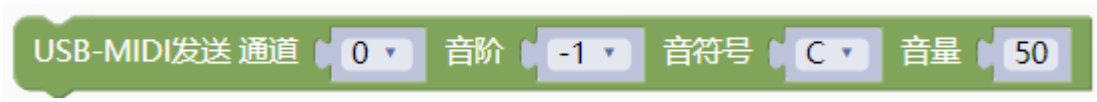
7. USB MIDI 设备设置通道和音色




```
Midi_Send(0x09,0xc0,0x0,0x0);
```

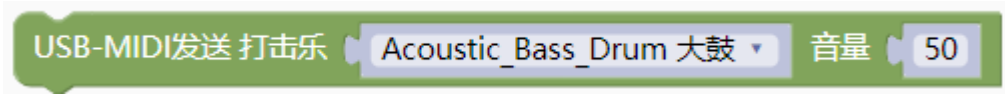
有关 MIDI 协议请查看相应资料。

8. USB MIDI 设备设置通道、音阶、音符、音量




```
Midi_Send(0x09,0x90,0x0,0x32);
```

9. USB MIDI 设备设置打击乐、音量

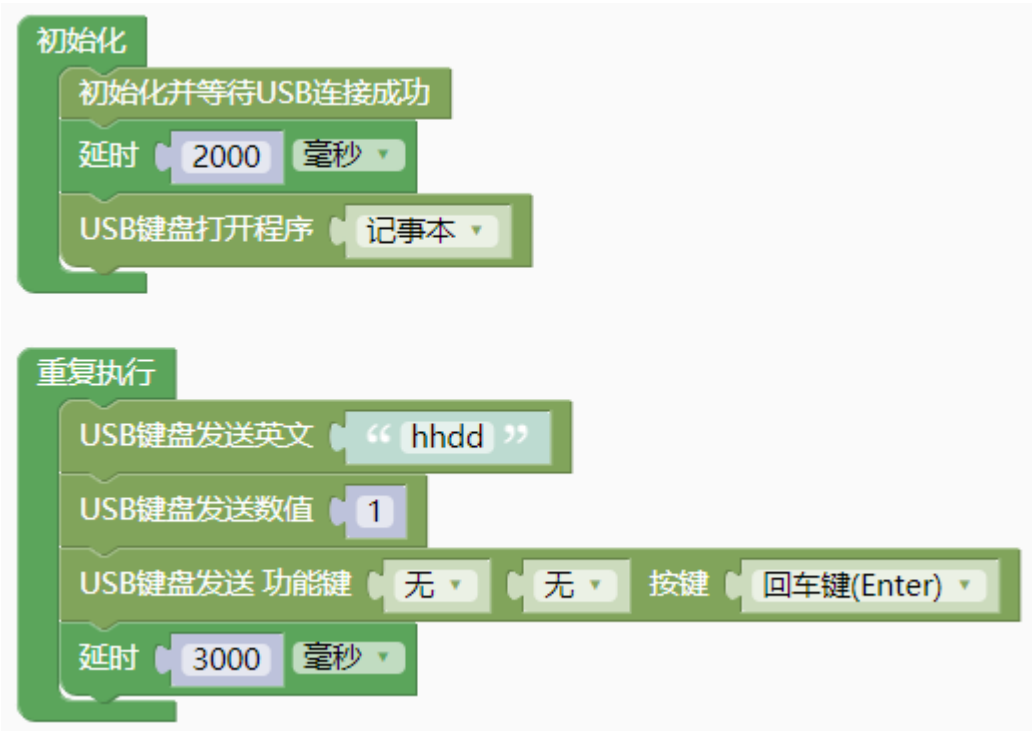



```
Midi_Send(0x09,0x90,0x0,0x32);
```

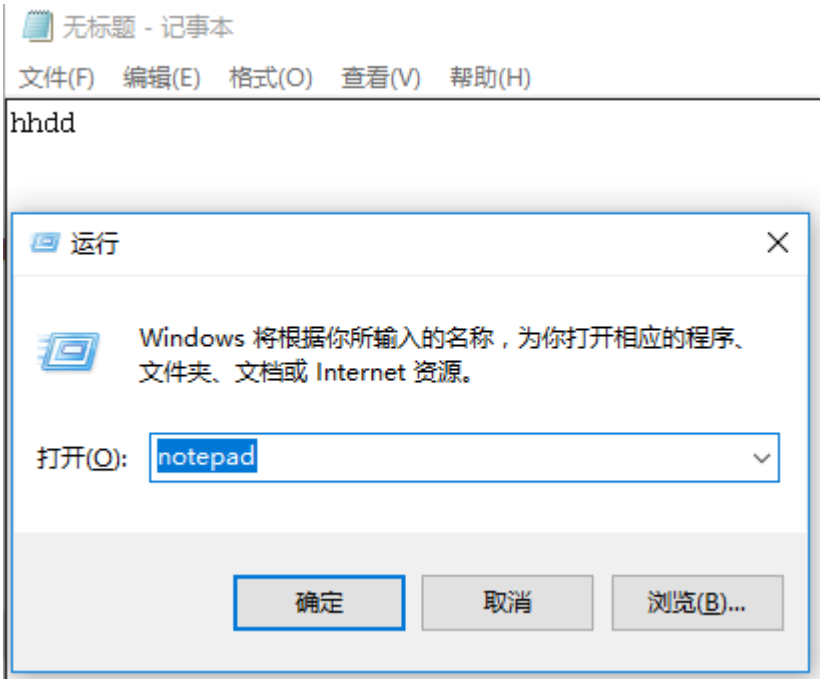
示例代码 1

模拟 USB 键盘打开记事本，自动输入文字。

注意本案例演示时，需要把输入法切换为英文状态。




```
#include <CH32V103.h>
#include "usb_keyboard.h"

boolean _configured;

int main(void)
{
  CH32_Init();
  usb_keymouse_init();
  _configured = 0;
  while (bDeviceState != CONFIGURED);
  if (_configured == 0) {
    delay(2000);
    _configured = 1;
  }
  delay(2000);
  keyBoard_Run("notepad");
  while(1){
    keyBoard_String("hhdd");
    keyBoard_Number(1);
    keyBoard_Send(0,40,0,0,0,0,0);
    delay(3000);
  }
  return 1;
}
```



示例代码 2

模拟 USB 鼠标操作

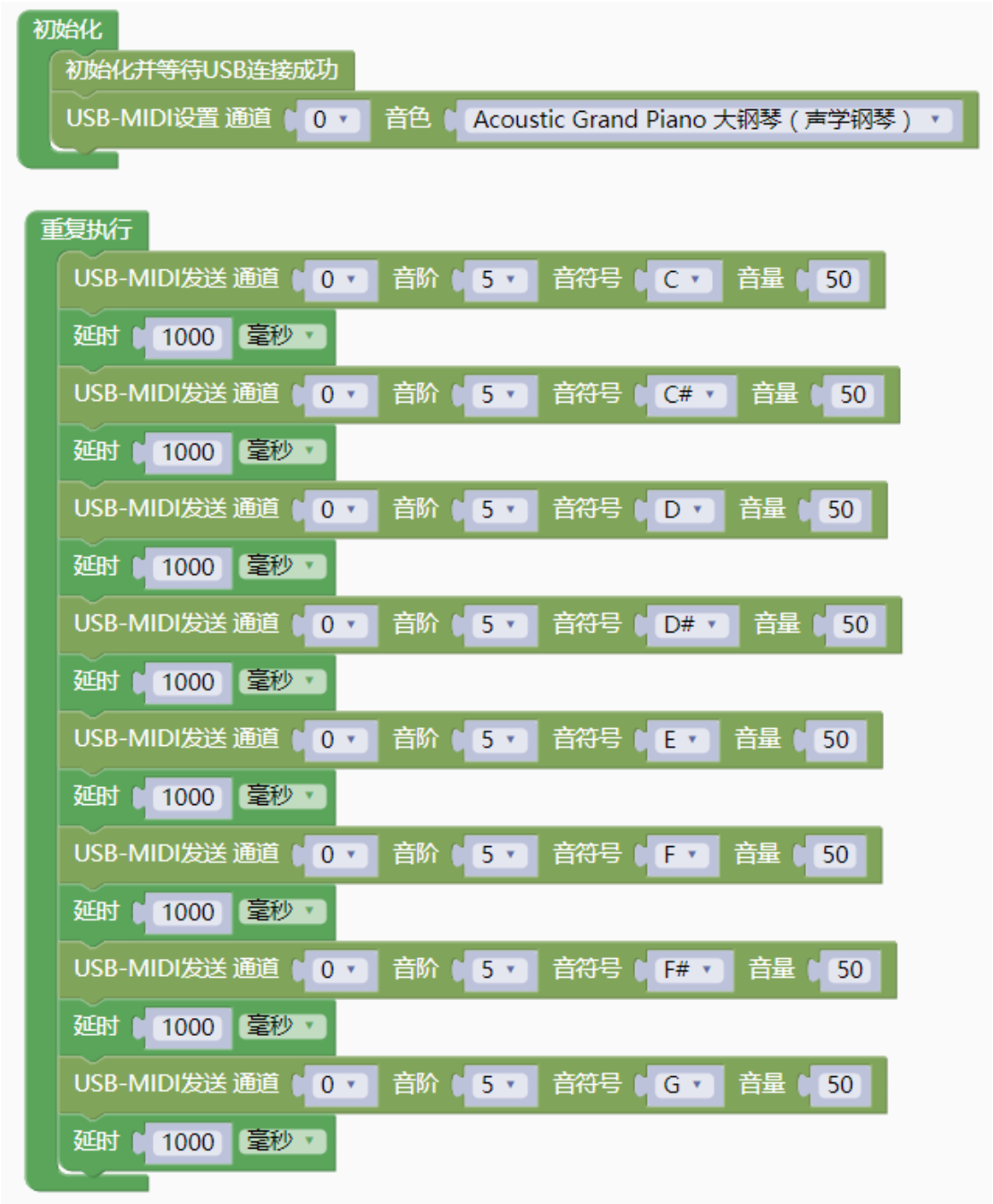

```
#include <CH32V103.h>
#include "usb_keyboard.h"

uint32_t i = 0;

//模拟鼠标功能
//程序运行后会先模拟鼠标“单击右键”
//之后鼠标会在屏幕上向右移动 200 个像素，之后向左移动 200 个像素，以此往复
boolean _configured;

int main(void)
{
  CH32_Init();
  usb_keymouse_init();
  _configured = 0;
  delay(2000);
```

```
  while (bDeviceState != CONFIGURED);
  if (_configured == 0) {
    delay(2000);
    _configured = 1;
  }
  Moue_Send(2,0,0,0);
  while(1){
    for (i = 0; i < 200; i = i + 1) {
      Moue_Send(0,1,0,0);
      delay(50);
    }
    for (i = 0; i < 200; i = i + 1) {
      Moue_Send(0,-1,0,0);
      delay(50);
    }
    delay(1000);
  }
  return 1;
}
```

示例代码 3

模拟 HID MIDI 设备发送 MIDI 音乐。

注意电脑上需要运行 MIDI 播放软件，才能正常演示。




```
#include <CH32V103.h>
#include "usb_midi.h"

boolean _configured;

int main(void)
{
  CH32_Init();
  usb_midi_init();
  _configured = 0;
  while (bDeviceState != CONFIGURED);
```

```
  if (_configured == 0) {
    delay(2000);
    _configured = 1;
  }
  Midi_Send(0x09,0xc0,0x0,0x0);
  while(1){
    Midi_Send(0x09,0x90,0x48,0x32);
    delay(1000);
    Midi_Send(0x09,0x90,0x49,0x32);
    delay(1000);
    Midi_Send(0x09,0x90,0x4a,0x32);
    delay(1000);
    Midi_Send(0x09,0x90,0x4b,0x32);
    delay(1000);
    Midi_Send(0x09,0x90,0x4c,0x32);
    delay(1000);
    Midi_Send(0x09,0x90,0x4d,0x32);
    delay(1000);
    Midi_Send(0x09,0x90,0x4e,0x32);
    delay(1000);
    Midi_Send(0x09,0x90,0x4f,0x32);
    delay(1000);
  }
  return 1;
}
```

PWM 模块

CH32V 的高级定时器 TIM1 和通用定时器 TIM2/3/4 都支持 PWM 功能，天问的软件框架把 PWM 功能单独的独立出来，支持基本的 PWM 输出功能，方便快速使用。如需要更多的 PWM 高级功能，可以直接使用 CH32V 的 SDK 开发。

引脚分布如下：
TIM1_CH1 PA8
TIM1_CH2 PA9
TIM1_CH3 PA10
TIM1_CH4 PA11

TIM2_CH1 PA0 (PA15)
TIM2_CH2 PA1 (PB3)
TIM2_CH3 PA2 (PB10)

TIM2_CH4 PA3 (PB11)

TIM3_CH1 PA6 (PB4) (PC6)
TIM3_CH2 PA7 (PB5) (PC7)
TIM3_CH3 PB0 (PC8)
TIM3_CH4 PB1 (PC9)

TIM4_CH1 PB6
TIM4_CH2 PB7
TIM4_CH3 PB8
TIM4_CH4 PB9

注意同一组 PWM 频率一样，占空比可以不一样，注意复用功能组的选择，成对使用，不能单独引脚设置。

1. 预定义 PWM 最大占空比值


```
#define PWM_DUTY_MAX 1000//PWM 最大占空比值
```

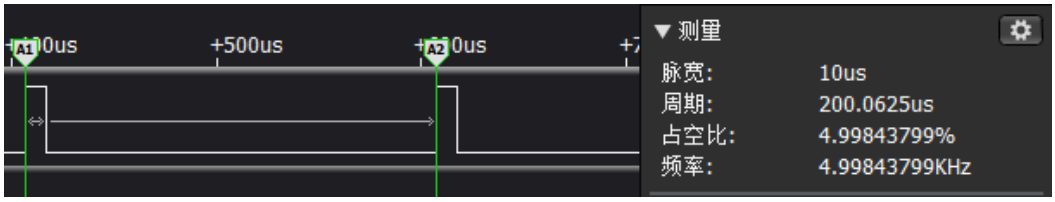
```
#ifndef PWM_DUTY_MAX
#define PWM_DUTY_MAX        100
#endif
```

如果不设置默认为 100，占空比会根据这个值来计算的。

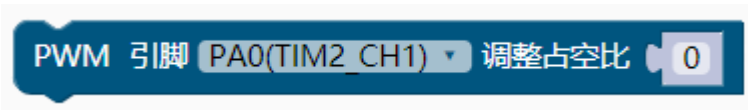
2. 设置 PWM 引脚、频率、占空比


```
PWM_Init(TIM2_CH1, PA0, 5000, 50);
```

这里需要注意占空比如果设置为 50，前面的最大占空比值为 100，则 PWM 波的占空比为 50%，如果前面的最大占空比值为 1000，则 PWM 波的占空比为 5%。



3. 单独调节占空比




```
PWM_Duty_Updata(TIM2_CH1, 0);
```

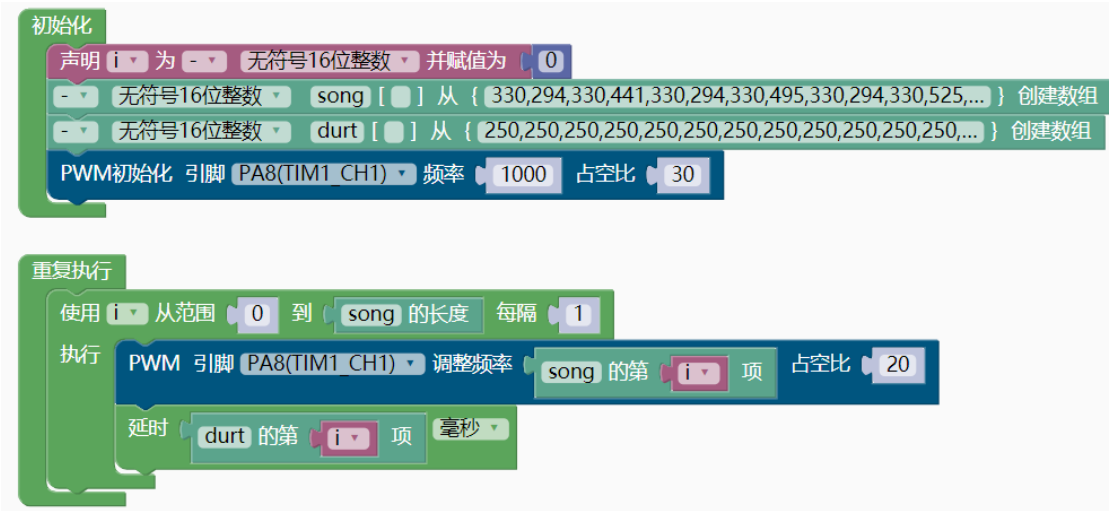
4. 单独调节 PWM


```
PWM_Frequency_Updata(TIM2_CH1, 5000, 0);
```

因为占空比和频率关联，所以调节过频率后，要同时设置占空比。

示例代码 1

PWM 控制板载蜂鸣器演奏音乐




```
#include <CH32V103.h>
#include "CH32V_PWM.h"

uint16_t i = 0;
uint16_t song[]={330,294,330,441,330,294,330,495,330,294,330,525,495,39
3,330,294,330,441,330,294,330,495,393,294,248,330,294,330,441,330,294,3
30,495,330,294,330,525,495,393,294,330,221,294,330,221,196,221,262,248}
;
uint16_t durt[]={250,250,250,250,250,250,250,250,250,250,250,250,500,50
0, 250,250,250,250,250,250,250,250, 500,500,1000, 250,250,250,250,250,2
50,250,250, 250,250,250,250,500,500, 250,250,500,250,250,250,250,500,50
0,1000,   250,250,250,250,250,250,250,125,125, 750,250,1000, 250,250,25
0,250,250,250,500,500,1500, 250,250,250,250,250,250,250,125,125, 750,25
0,1000, 250,250,500,250,250,250,250,1500,  250,250,750,250,500,250,250,
 750,250,500,250,250,500,250,250,250,250,500,500,1000, 250,250,875,125,
500,250,250,500,500,1000, 250,250,500,250,250,250,250,1500,  250,250,75
0,250,500,250,250, 500,250,250,500,250,250,500,250,250,250,250,250,250,
1500, 250,250,750,250,500,250,250, 500,250,250,1000,250,250,500,250,250
,250,250,2000};

int main(void)
{
```

```
  CH32_Init();
  PWM_Init(TIM1_CH1, PA8, 1000, 30);
  while(1){
    for (i = (0); i < (sizeof(song)/sizeof(song[0])); i = i + 1) {
      PWM_Frequency_Updata(TIM1_CH1, song[(int)(i)], 20);
      delay(durt[(int)(i)]);
    }
  }
  return 1;
}
```

定时器模块

CH32V 有四个 16 位定时器 TIM1/2/3/4，其中 TIM1 属于高级定时器，包含很多高级功能，天问软件框架集成 TIM2/3/4 基本的定时功能，后续通过扩展库添加包括 TIM1 的输入捕获、输出比较、编码器等功能。

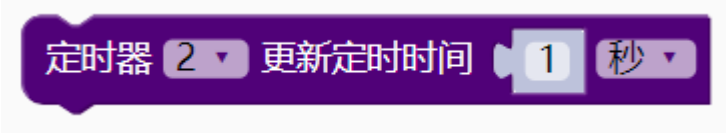
1. 定时器设置和回调函数


```
TIM_attachInterrupt(TIM2, 1000000, TIM_attachInterrupt_2);
```

```
void TIM_attachInterrupt_2() {
}
```

定时器 2，3，4 最大定时时间为 6553 毫秒。
使用本语句，初始化完成后，默认自动启动定时器。

2. 更新定时时间




```
TIM_Duty_Updata(TIM2, 1000000);
```

3. 关闭定时器


```
TIM_Disable(TIM2);
```

4. 启动定时器


```
TIM_Enable(TIM2);
```

示例代码 1

200 毫秒翻转 IO


```
#include <CH32V103.h>
#include "CH32V_TIM.h"

void TIM_attachInterrupt_2() {
  digitalWrite(PC8, !(digitalRead(PC8)));
}

int main(void)
{
  CH32_Init();
  pinMode(PC8, GPIO_Mode_Out_PP);
  TIM_attachInterrupt(TIM2, 200000, TIM_attachInterrupt_2);
  while(1){

  }
  return 1;
}
```

外部中断模块

CH32V103 系列内置快速可编程中断控制器（PFIC - Programmable Fast Interrupt Controller），最多支持 255 个中断向量。当前系统管理了 44 个外设中断通道和 5 个内核中断通道，其他保留。

CH32V103 的所有 IO 口支持上升沿、下降沿、电平变化三种中断方式。

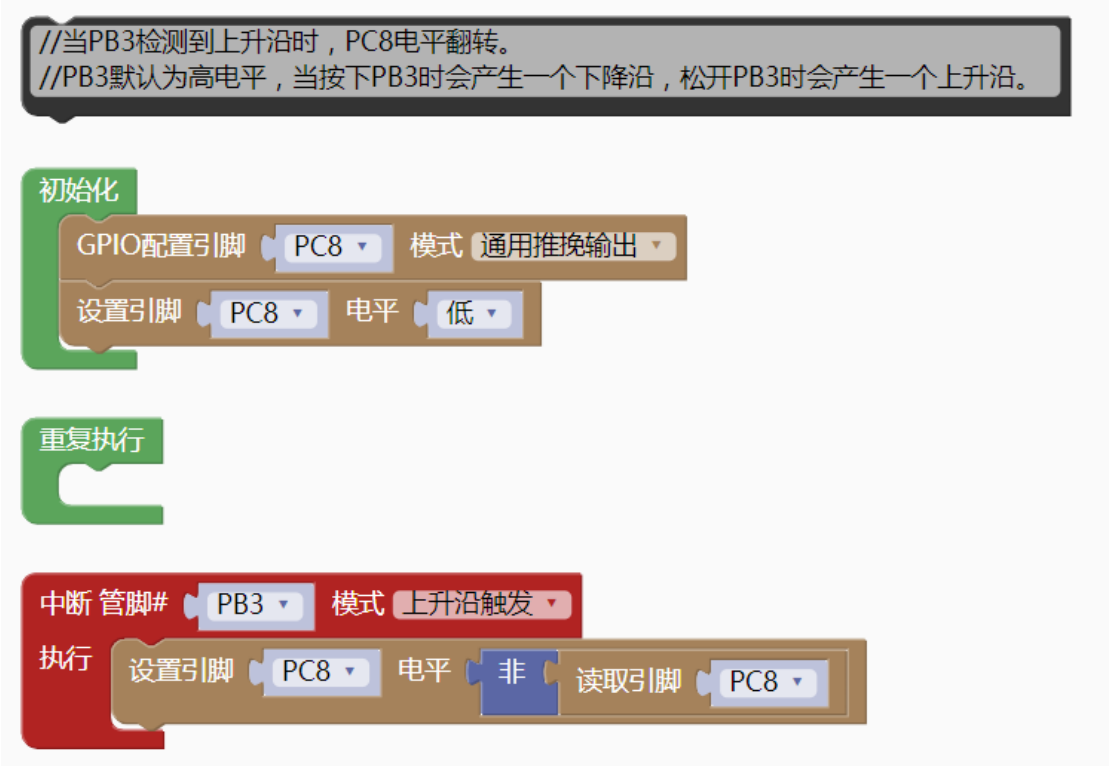
1. 中断管脚设置，触发模式设置，回调函数


```
 Pin_attachInterrupt(PB3,EXTI_Trigger_Rising,pin_attachInterrupt_fun_PB
3);
```

```
void pin_attachInterrupt_fun_PB3() {
}
```

示例代码 1

PB3 上升沿触发控制 LED




```
#include <CH32V103.h>
```

```
#include "CH32V_EXTI.h"

//当 PB3 检测到上升沿时，PC8 电平翻转。
//PB3 默认为高电平，当按下 PB3 时会产生一个下降沿，松开 PB3 时会产生一个上升沿。
void pin_attachInterrupt_fun_PB3() {
  digitalWrite(PC8, !(digitalRead(PC8)));
}

int main(void)
{
  CH32_Init();
  pinMode(PC8, GPIO_Mode_Out_PP);
  digitalWrite(PC8, 0);
  Pin_attachInterrupt(PB3,EXTI_Trigger_Rising,pin_attachInterrupt_fun_P
B3);
  while(1){

  }
  return 1;
}
```

Flash 模块

芯片内部闪存组织结构如下（以 xR8T6 为例）：

表 24-1 闪存组织结构

块	名称	地址范围	大小（字节）
主存储器	页 0	0x08000000 - 0x0800007F	128
	页 1	0x08000080 - 0x080000FF	128
	页 2	0x08000100 - 0x0800017F	128
	页 3	0x08000180 - 0x080001FF	128
	页 4	0x08000200 - 0x0800027F	128
	页 5	0x08000280 - 0x080002FF	128
	页 6	0x08000300 - 0x0800037F	128
	页 7	0x08000380 - 0x080003FF	128
	…	…	…
	页 511	0x0800FF80 - 0x0800FFFF	128
信息块	系统引导代码存储 1	0x1FFFF000 - 0x1FFFF7FF	2K
	用户选择字	0x1FFFF800 - 0x1FFFF87F	128
	厂商配置字	0x1FFFF880 - 0x1FFFF8FF	128
	系统引导代码存储 2	0x1FFFF900 - 0x1FFFFFFF	1792

上述主存储器区域用于用户的应用程序存储，以 4K 字节（32 页）单位进行写保护划分；除了“厂商配置字”区域出厂锁定，用户不可访问，其他区域在一定条件下用户可操作。

如上图所示，CH32V103 的闪存地址从 0x08000000 开始，一共 64K 字节数据，其中又把每 128 个字节分为一页，有 0-511（共 512）页，天问 CH32V 开发板出厂前面已经内置 8K 的 bootloader，为了防止 bootload 被擦除，该 Flash 的前 8K 字节的数据不允许写入，如果需要写入，请自行修改 Flash 库。

闪存编程及安全性

1）2 种编程/擦除方式

- 标准编程：此方式是默认编程方式（兼容方式）。这种模式下 CPU 以单次 2 字节方式执行编程， 单次 1K 字节执行擦除及整片擦除操作。
- 快速编程：此方式采用页操作方式（推荐）。经过特定序列解锁后，执行单次 128 字节的编程及 128 字节擦除。

2）安全性-防止非法访问（读、写、擦）

- 页写入保护
- 读保护

读保护状态下：

1）主存储器 0-31 页（4K 字节）自动写保护状态，不受 FLASH_WPR 寄存器控制；解除读保护状态，所有主存储页都由 FLASH_WPR 寄存器控制。

2）系统引导代码区、SWD 模式、RAM 区域都不可对主存储器进行擦除或编程，整片擦除除外。可擦除 或编程用户选择字区域。如果试图解除读保护（编程用户字），

芯片将自动擦除整片用户区。

注：进行闪存的编程/擦除操作时，必须打开内部 RC 振荡器（HSI）。

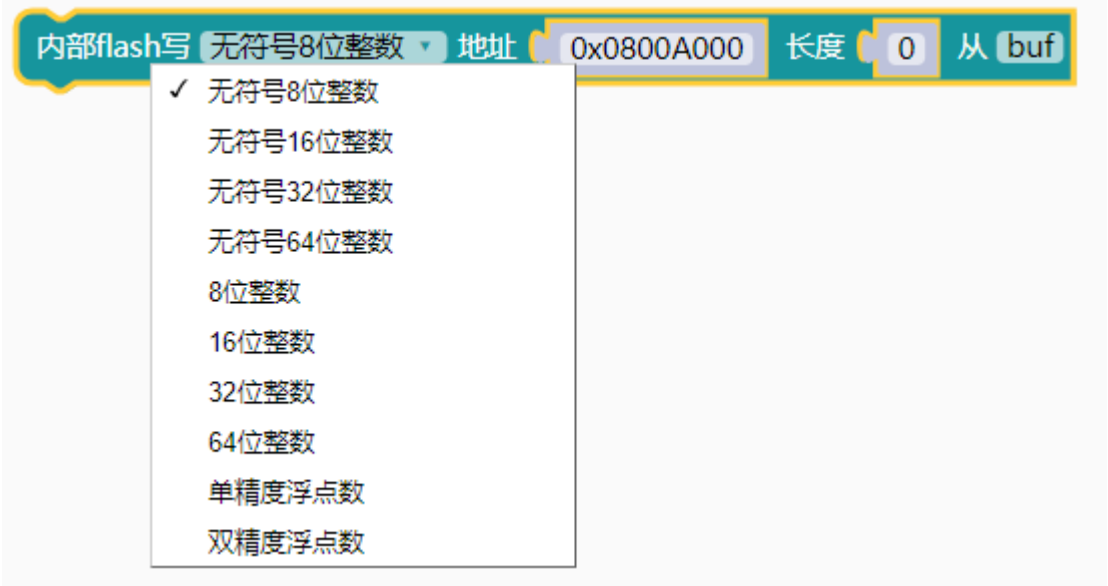
天问软件框架把 Flash 的底层操作封装好，方便快速使用。

1. FLASH 读数据


```
int8_t read(uint32_t data_addr, uint8_t *buf, uint16_t num_bytes);
int8_t readChar(uint32_t data_addr);
uint8_t readUChar(uint32_t data_addr);
int16_t readShort(uint32_t data_addr);
uint16_t readUShort(uint32_t data_addr);
int32_t readInt(uint32_t data_addr);
uint32_t readUInt(uint32_t data_addr);
int64_t readLong64(uint32_t data_addr);
uint64_t readULong64(uint32_t data_addr);
float readFloat(uint32_t data_addr);
double readDouble(uint32_t data_addr);
```

支持多种数据格式的读取，注意地址范围，不要和 bootloader 冲突。

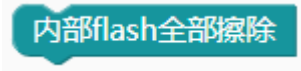
2. FLASH 写数据




```
FLASH_Status writeChar(uint32_t data_addr,char *Data,uint32_t count);
FLASH_Status writeUChar(uint32_t data_addr,uint8_t *Data,uint32_t count
);
FLASH_Status writeShort(uint32_t data_addr,int16_t *Data,uint32_t count
);
FLASH_Status writeUShort(uint32_t data_addr,uint16_t *Data,uint32_t cou
nt);
FLASH_Status writeInt(uint32_t data_addr,int32_t *Data,uint32_t count);
FLASH_Status writeUInt(uint32_t data_addr,uint32_t *Data,uint32_t count
);
FLASH_Status writeLong64(uint32_t data_addr,int64_t *Data,uint32_t coun
t);
FLASH_Status writeULong64(uint32_t data_addr,uint64_t *Data,uint32_t co
unt);
FLASH_Status writeFloat(uint32_t data_addr,float *Data,uint32_t count);
FLASH_Status writeDouble(uint32_t data_addr,double *Data,uint32_t count
);
```

支持多种数据格式的写入，注意地址范围，不要和 bootloader 冲突。

3. FLASH 全部擦除



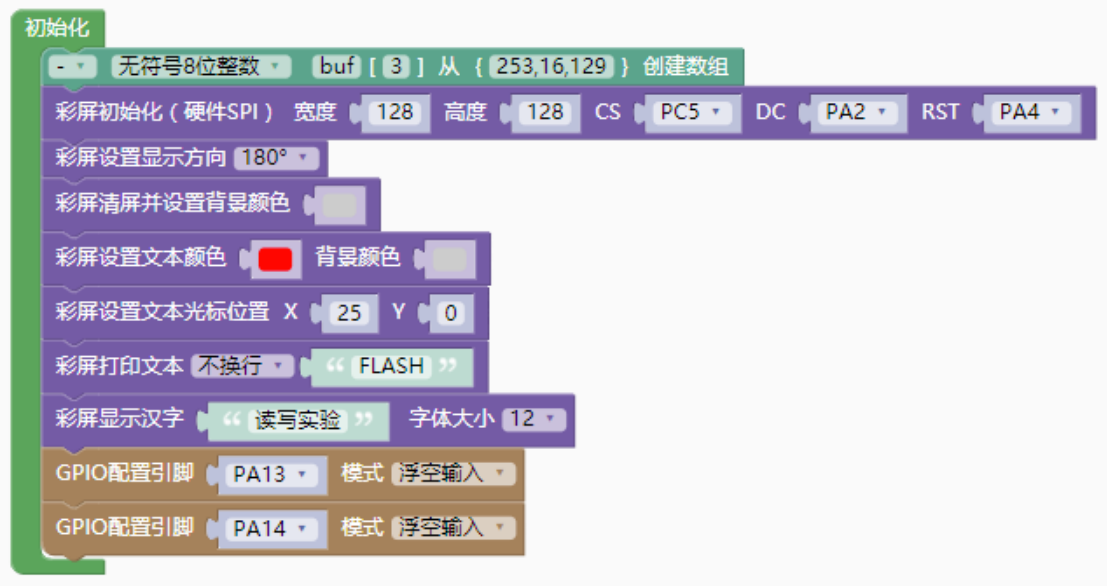

```
FLASH_Status FLASH_erase_page_all(void);  //擦除 flash 全部数据
```

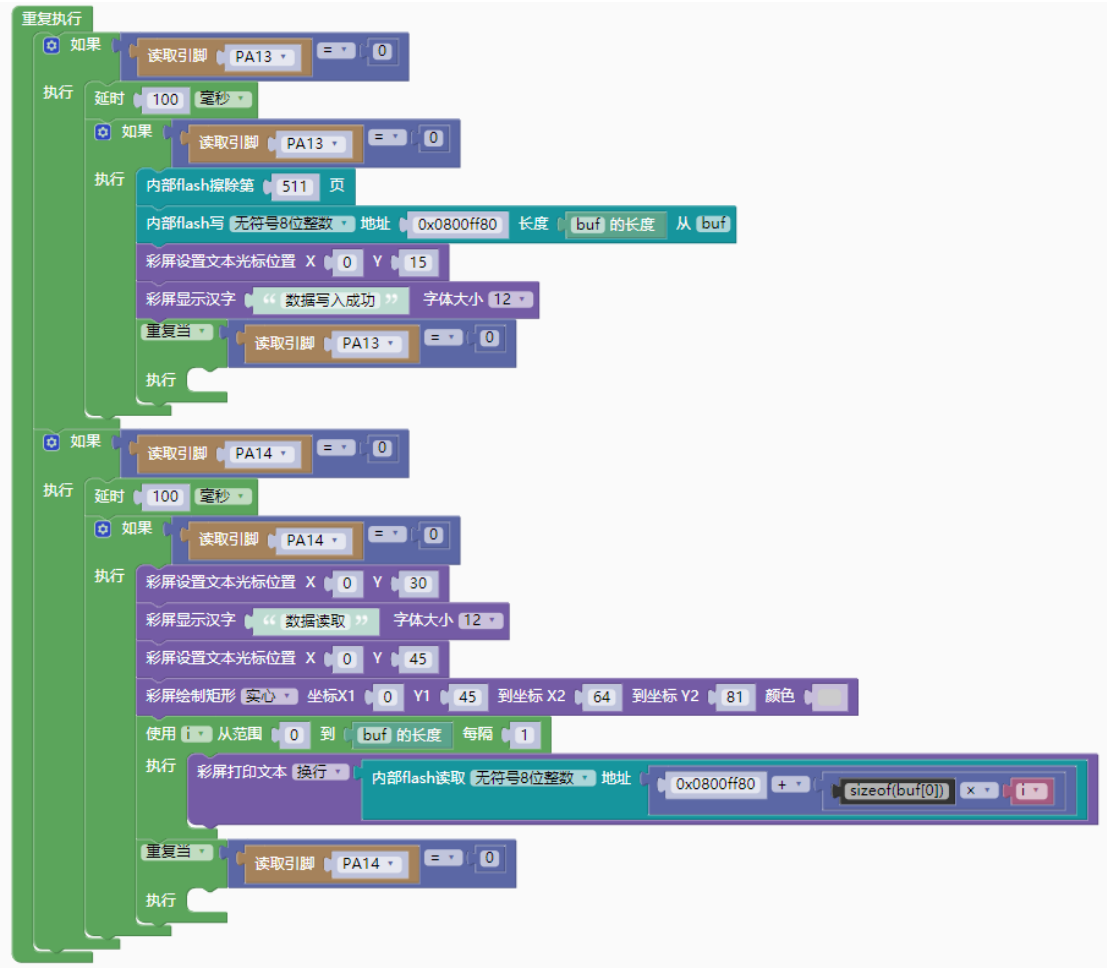
4. FLASH 擦除指定页


```
FLASH_Status erase_page(uint32_t page);
```

示例代码 1

内部 flash 读写无符号 8 位整数






```
#include <CH32V103.h>
#include "myLib/CH32V_ST7735S.h"
#include "CH32V_FLASH.h"
```

```
uint8_t buf[3]={253,16,129};
unsigned char i;

//按下 KEY 键往 flash 写入数据，按下 KEY2 键读出数据
SPITFT spi_tft(128,128,PC5,PA2,PA4);
INTERNAL_FLASH flash;

int main(void)
{
  CH32_Init();
  spi_tft.init();
  spi_tft.set_direction(2);
  spi_tft.clear((0xCE79));
  spi_tft.set_text_color((0xF800),(0xCE79));
  spi_tft.set_cursor(25,0);
  spi_tft.print("FLASH");
  spi_tft.draw_hanzi_12("读写实验");
  pinMode(PA13, GPIO_Mode_IN_FLOATING);
  pinMode(PA14, GPIO_Mode_IN_FLOATING);
  while(1){
    if(digitalRead(PA13) == 0){
      delay(100);
      if(digitalRead(PA13) == 0){
        flash.erase_page(511);
        flash.writeUChar(0x0800ff80,buf,(sizeof(buf)/sizeof(buf[0])));
        spi_tft.set_cursor(0,15);
        spi_tft.draw_hanzi_12("数据写入成功");
        while (digitalRead(PA13) == 0) {
        }
      }
    }
    if(digitalRead(PA14) == 0){
      delay(100);
      if(digitalRead(PA14) == 0){
        spi_tft.set_cursor(0,30);
        spi_tft.draw_hanzi_12("数据读取");
        spi_tft.set_cursor(0,45);
        spi_tft.fill_rectangle(0,45,64,81,(0xCE79));
        for (i = (0); i < (sizeof(buf)/sizeof(buf[0])); i = i + 1) {
          spi_tft.println((flash.readUChar((0x0800ff80 + sizeof(buf[0])
 * i))));
        }
        while (digitalRead(PA14) == 0) {
```

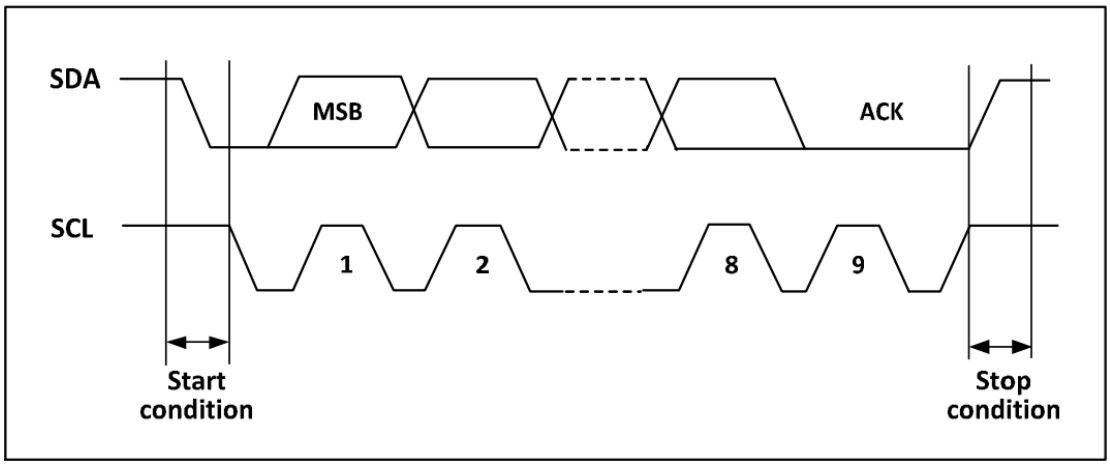
```
        }
      }
    }
  }
  return 1;
}
```

I2C 模块

I2C 总线是由 Philips 公司开发的一种简单、双向二线制同步串行总线。它只需要两根线即可在连接于总线上的器件之间传送信息。

主器件用于启动总线传送数据，并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件．在总线上主和从、发和收的关系不是恒定的，而取决于此时数据传送方向。如果主机要发送数据给从器件，则主机首先寻址从器件，然后主动发送数据至从器件，最后由主机终止数据传送；如果主机要接收从器件的数据，首先由主器件寻址从器件．然后主机接收从器件发送的数据，最后由主机终止接收过程。在这种情况下．主机负责产生定时时钟和终止数据传送。

I2C 是个半双工的总线，它同时只能运行在下列四种模式中之一：主设备发送模式、主设备接收 模式、从设备发送模式和从设备接收模式。I2C 模块默认工作在从模式，在产生起始条件后，会自动地切换到主模式，当仲裁丢失或者产生停止信号后，会切换到从模式。I2C 模块支持多主机功能。工作在主模式时，I2C 模块会主动发出数据和地址。数据和地址都以 8 位为单位进行传输，高位在前，低位在后，在起始事件后的是一个字节（7 位地址模式下）或两个字节（10 位地址模式下）地址，主机每发送 8 位数据或地址，从机需要回复一个应答 ACK，即把 SDA 总线拉低，如下图所示。



在嵌入式开发中经常用到硬件 I2C 和软件 I2C。硬件 I2C 的总线的信号处理由单片机硬件处理，不占用 CPU；软件 I2C，则需要 CPU 来处理。

两则的区别为：

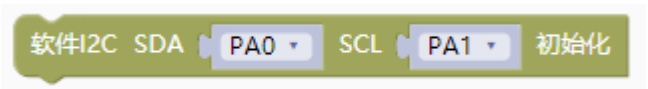
- 硬件 I2C 引脚固定，路数少，速度高，支持主从两种模式。

- 软件 I2C 所有 IO 都可以，数量多，速度受限于 CPU，只支持主机。

CH32V 包含 2 路硬件 I2C。支持多主多从模式，仅仅使用两根线（SDA 和 SCL）就能以 100KHz（标准）和 400KHz（快速）两种速度通讯。I2C 总线还兼容 SMBus 协议，不仅支持 I2C 的时序，还支持仲裁、定时和 DMA，拥有 CRC 校验功能。

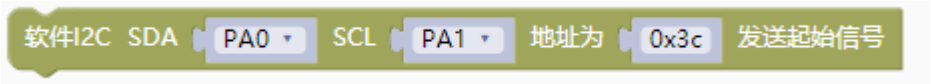
天问软件框架中目前已经集成软件主机 I2C 模块，硬件 I2C 后续更新。

1. 软件 I2C 初始化



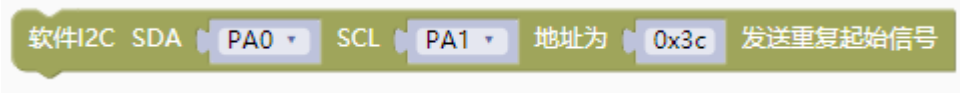

```
void begin(uint8_t sdapin, uint8_t sclpin); //SOFTIIC 初始化
```

2. 软件 I2C 发送起始信号



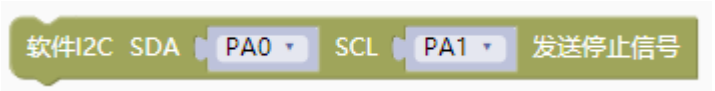

```
uint8_t read(uint8_t last); //读函数
```

3. 软件 I2C 发送重复起始信号



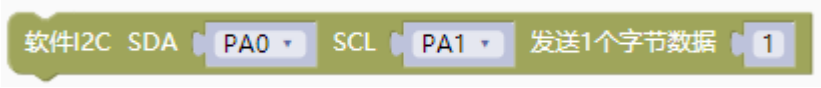

```
bool restart(uint8_t addr); //重复起始信号
```

4. 软件 I2C 发送停止信号



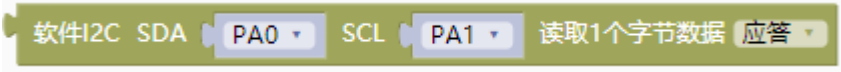

```
bool start(uint8_t addr); //起始信号
```

5. 软件 I2C 发送 1 个字节数据




```
void stop(void);  //停止信号
```

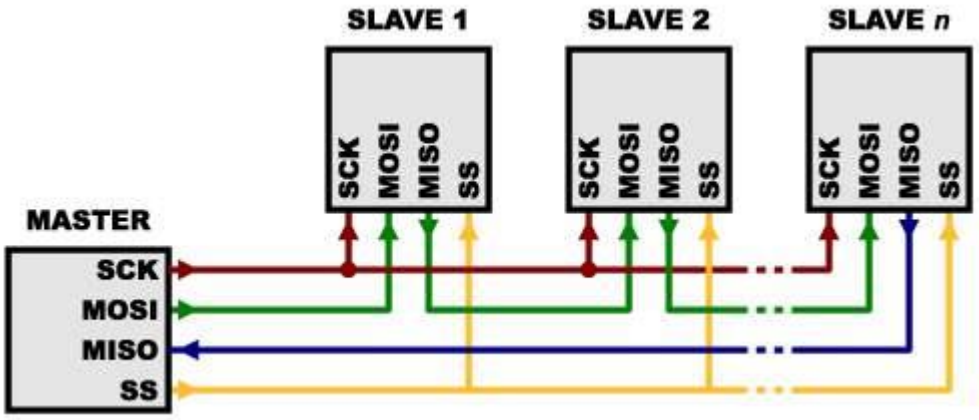
6. 软件 I2C 读取 1 个字节数据应答/非应答




```
bool write(uint8_t b);  //写函数
```

SPI 模块

SPI 是由摩托罗拉(Motorola)公司开发的全双工同步串行总线，是微处理控制单元(MCU)和外围设备之间进行通信的同步串行端口。主要应用在 EEPROM、Flash、实时时钟(RTC)、数模转换器(ADC)、网络控制器、MCU、数字信号处理器(DSP)以及数字信号解码器之间。SPI 系统可直接与各个厂家生产的多种标准外围器件直接接口，一般使用 4 条线:串行时钟线 SCK、主机输人/从机输出数据线 MISO、主机输出/从机输人数据线 MOSI 和低电平有效的从机选择线。



在嵌入式开发中经常用到硬件 SPI 和软件 SPI。硬件 SPI 的总线的信号处理由单片机硬件处理，不占用 CPU；软件 SPI，则需要 CPU 来处理。

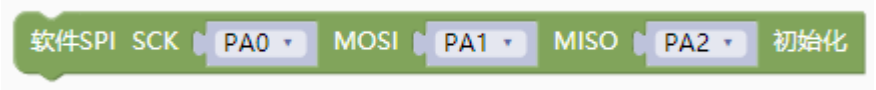
两则的区别为：

- 硬件 SPI 引脚固定，速度高，支持主从两种模式。
- 软件 SPI 所有 IO 都可以，数量多，速度受限于 CPU，只支持主机。

CH32V 包含 2 路硬件 SPI。支持多主多从模式，最高速度为系统时钟的一半。

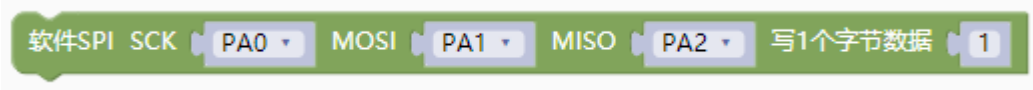
天问软件框架中目前已经集成软件主机 SPI 模块，硬件 SPI 后续更新。

1. 软件 SPI 初始化



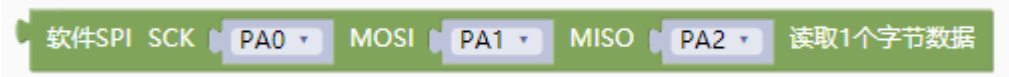

```
void begin(uint8_t sckpin, uint8_t mosipin, uint8_t misopin);
```

2. 软件 SPI 写数据




```
void write_data(uint8_t data); //写函数
```

3. 软件 SPI 写数据同时返回读取数据

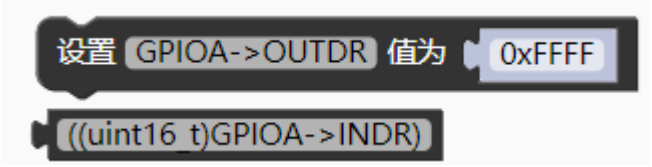



```
uint8_t read_data(); //读函数
```

读写寄存器模块

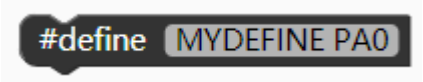
平台图形化模块只是提供了常用的功能，一些特殊外设和寄存器没有提供，如果还是想用图形化编程，可以使用如下模块，可以自己设置寄存器，也可以嵌入 C 语言代码或者直接嵌入汇编。

1. 读写寄存器



一些特殊寄存器，没做对应的图形块，可以用这个模块手动添加。

2. 宏定义



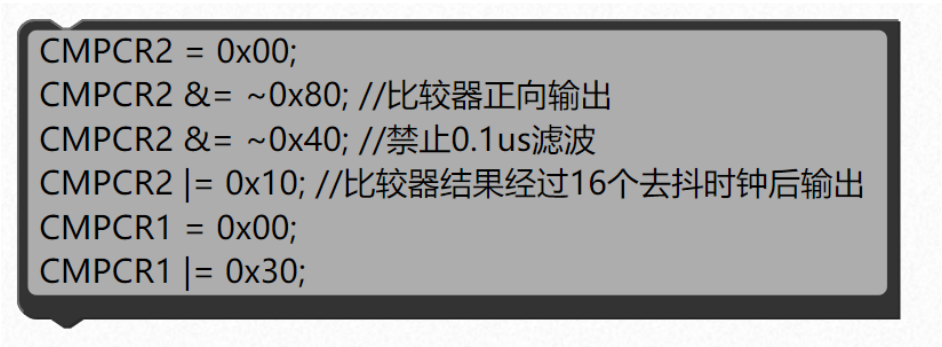
等效于

```
#define MYDEFINE PA0
```

3. 嵌入代码

示例 1：

嵌入 C 语言代码。



程序模块

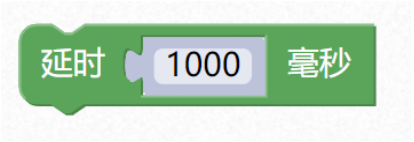
程序模块主要是和 C 语言相关的模块。

控制模块

1. 延时 1000 微秒。
   图形化模块提供了常用 1 种微秒级的延时函数，每个函数都是在频率为 24M 下，利用 STC-ISP 工具计算出来的函数。如需要其它微秒级的延时函数，请自己使用工具计算。


```
delayMicroseconds(1000);
```

2. 毫秒级延迟函数。
   以 24M 频率下的 1 毫秒延时函数为最小单位。




```
delay(1000);
```

内部实现代码

```
//-----------------------------------------------------------------------
// 描述: 延迟 1 毫秒.
// 参数: none.
// 返回: none.
//-----------------------------------------------------------------------
void delay1ms() //1 毫秒@24.000MHz
{
  uint8 i, j;
  _nop_();
  i = 32;
  j = 40;
  do { while (--j);} while (--i);
}

//-----------------------------------------------------------------------
// 描述: 延迟指定毫秒.
// 参数: 延迟时间（0-65535）.
// 返回: none.
```

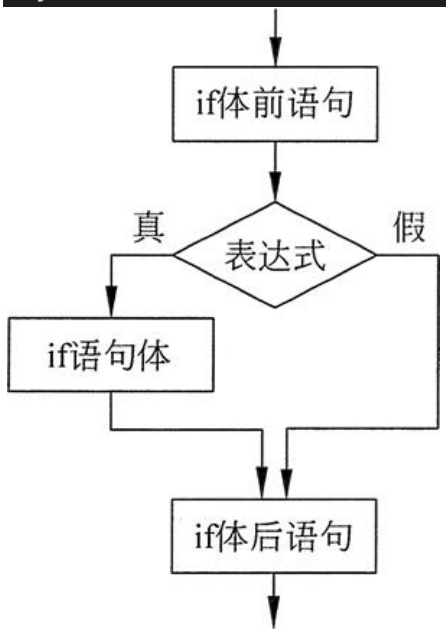
```
//====================================================================
void delay(uint16 time)
{
  do { delay1ms();} while (--time);
}
```

3. 空指令。
   执行一个指令需要的时间，由系统频率确定，用在需要精确时间的场合里，比如前面的微秒级的延时函数内部，就是由 nop 组成。


```
_nop_();
```

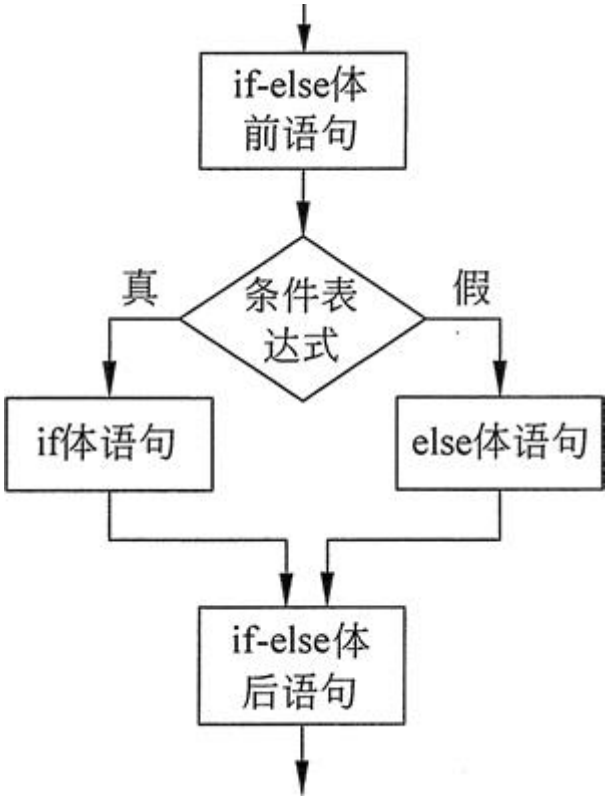
4. 如果判断分支语句。
   如果条件判断成立，则执行里面的代码，否则不执行。


```
if(0){

  }
```

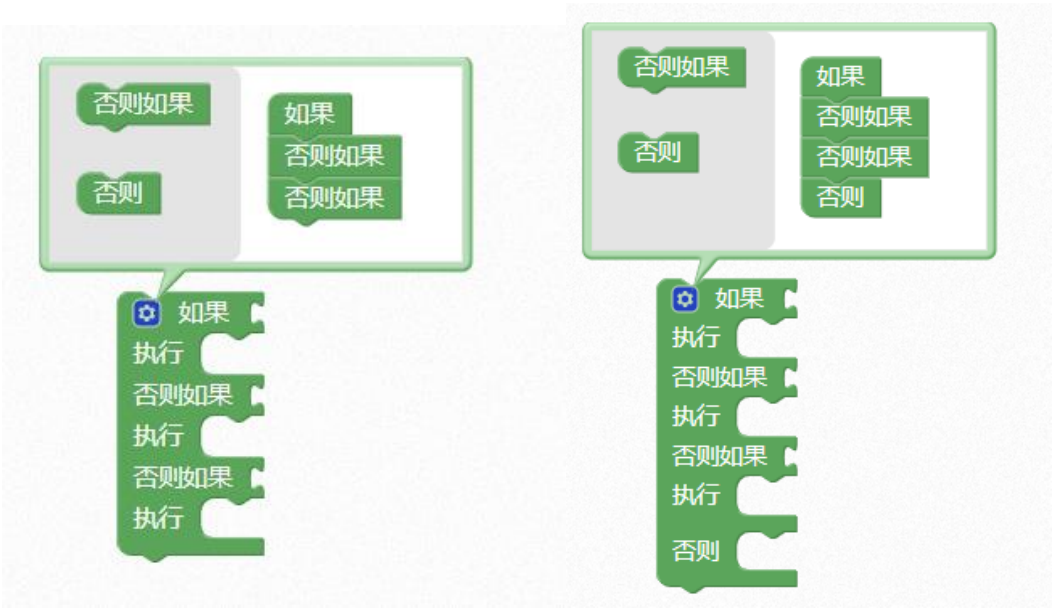


还可以通过点击蓝色小齿轮，添加多个判断语句。
如下为如果否则判断分支语句：
如果条件判断成立，则执行如果里面的代码，否则执行否则里的代码。


```
if(0)
{
  /* code */
}
else
{
  /* code */
}
```



还可以添加多个否则如果，执行多个判断，如果对应的条件成立，则执行里面的代码。




```
if(0){

}
else if(0){

}
else if(0){

}
```

5. 重复循环语句。

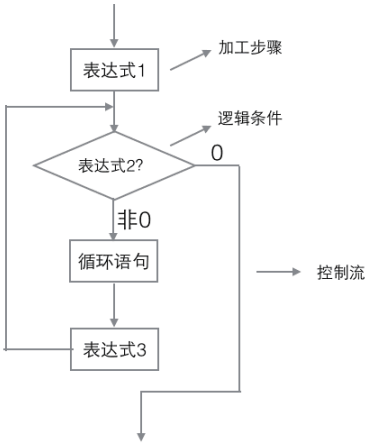

```
while(1){
  for (i = 0; i < 9; i = i + 1) {
  }
}
```

i 默认为 8 位无符号，最大值为 255，如果循环次数大于 255，请自己添加变量声明模块，修改变量类型。

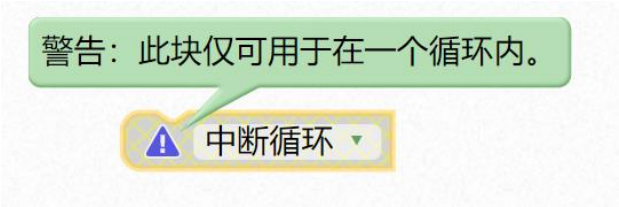
for (表达式 1;表达式 2;表达式 3)
{
  循环语句
}
表达式 1 给循环变量赋初值

表达式 2 为循环条件
表达式 3 用来修改循环变量的值，称为循环步长。

for 语句的执行流程：



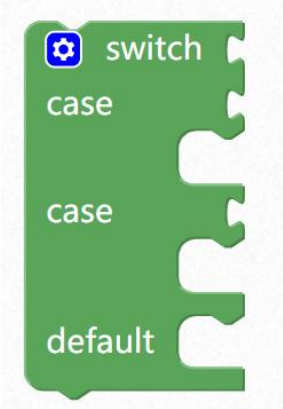
如果循环中需要中断循环，可以用中断循环模块。




```
break;
```

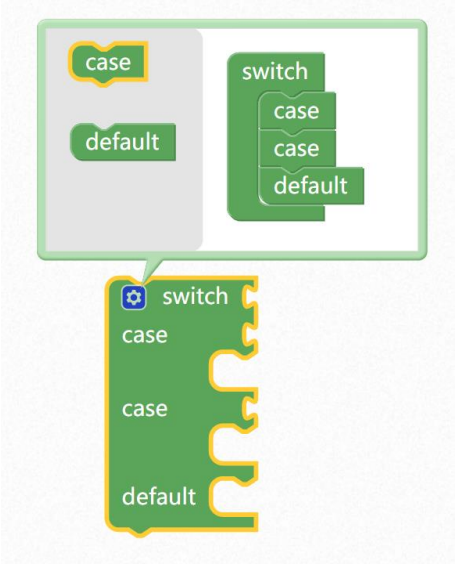
注意 :变量 i 为图形化模块自动生成, 默认为 8 位, 即范围最大 255, 如果循环间隔大于 255, 需要修改变量类型，请查看变量的相关使用。

6. 多个条件判断模块。
   功能和对个如果否则判断模块一样，有一些微小区别，我们一般用 switch case 用来判断多个常量时使用，语法简洁明了，执行效率比较高。




```
switch (NULL) {
   case NULL:
    break;
   case NULL:
    break;
   default:
    break;
```

可以通过点击蓝色小齿轮，添加多个判断语句。



示例 1：

判断学号，匹配姓名

7. 初始化模块。

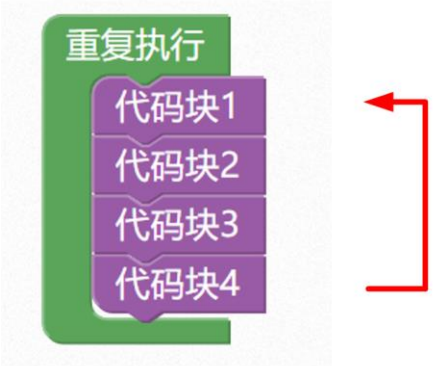
里面的代码只在上电后执行一次，因此我们通常把一些变量的声明或引脚初始化等放在初始化里。

8. 重复执行模块。

里面的代码一直在循环往复的执行。最后一条代码执行完后回过来执行第一条代码。


```
void setup()
{

}

void loop()
{

}

void main(void)
{
  setup();
```

```
  while(1){
    loop();
  }
}
```



数学与逻辑模块

1. 数字模块
   数值大小里面可以自己填写。

2. 常用数学运算
   包含加、减、乘、除、幂，两个输入框放入变量输出块或者直接修改数字，运算结果会返回给输入块。

✓ +
- -
- ×
- ÷
- ^ (幂)

示例 1：
变量 a+1。

示例 2：
变量 x+y。

3. 位操作

包含与、或、异或、右移、左移。

✓ &

|

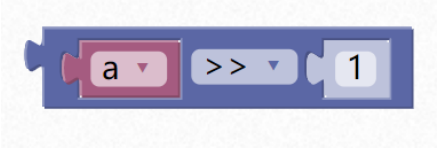
^

>>

<<

示例 1：

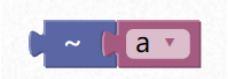
变量 a>>1 右移 1 位。



4. 位取反

示例 1：

变量 a 按位取反。



5. 数字比较

包含等于、不等于、小于、小于等于、大于、大于等于。

✓ =

≠

<

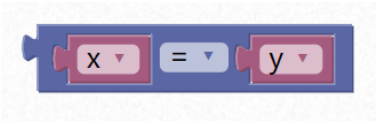
≤

>

≥

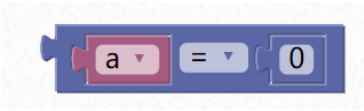
示例 1：

比较变量 x 和 y 是否相等。



示例 2：

比较变量 x 是否等于 0。

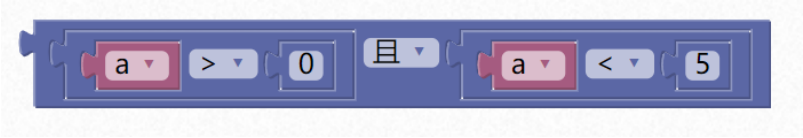


6. 逻辑比较

包含逻辑且（&&）、或（||）。

示例 1：

当 a>0 并且 a<5 时，条件才成立返回真，否则返回假。

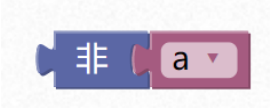



```
(a>0) && (a<5)
```

7. 逻辑非

示例 1：

变量 a，逻辑取反。




```
!a
```

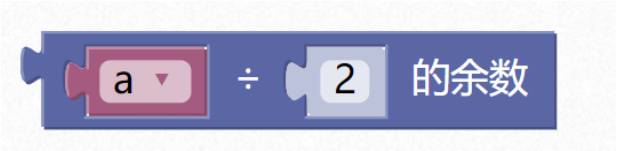
8. 获取指定区间内的随机数


```
//返回指定区间内的随机数，不包含区间最大值。
random(1, 100+1); //返回 1——100 之间的随机数。
```

9. 取余数（%）

示例 1：

返回变量除于 2 的余数，只有 0，1 两种，可以用来判断奇偶数。



10. 取舍取整函数

包含四舍五入、向上取整，向下取整。


```
round(3.1)
```

返回四舍五入后的数。

11. 复杂数学运算

包含平方根、绝对值、负数、对数、幂、三角函数。

✓ 平方根

绝对

-

ln

log10

e^

10^

sin

cos

tan

asin

acos

atan

```
sqrt(9)     //平方根
abs(9)      //绝对值
-           // 负数
log(9)     //ln
log10(9)   //log10
exp(9)     //e^
pow(10,9) //10^9
sin(9 / 180.0 * PI) //sin
cos(9 / 180.0 * PI) //cos
tan(9 / 180.0 * PI) //tan
asin(9) / PI * 180 //asin
acos(9) / PI * 180 //acos
atan(9) / PI * 180 //atan
```

12. 映射

返回指定比例系数和范围的数据。常用在给数据的范围等比例放大或者缩小。


```
map(,1,100,1,1000)
```

内部实现代码

```
long map(long x, long in_min, long in_max, long out_min, long out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_m
in;
}
```

示例 1：

变量 a 的初始范围为 1 到 100，等比例放大 10 倍。

即 a=1，返回 1；

a=50，返回 500；

a=100，返回 1000；

文本与数组模块

1. 字符串

示例 1：

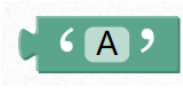
定义"hello world"字符串


```
"hello world"
```

2. 字符

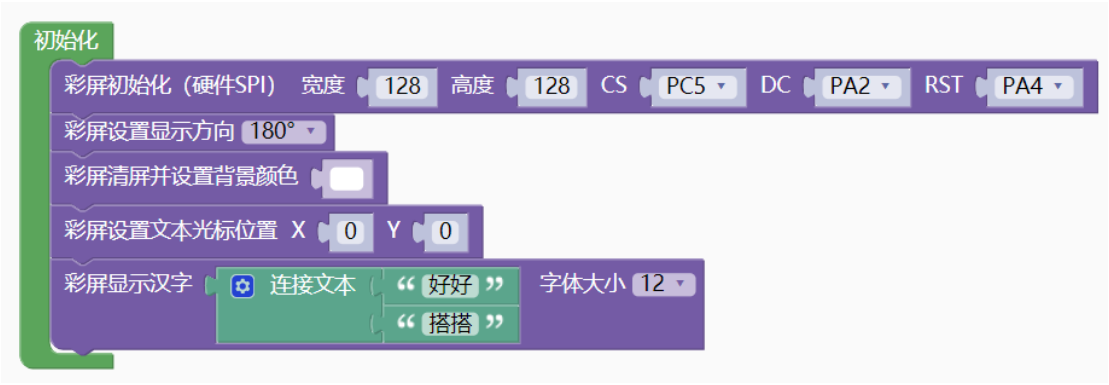
示例 1：

定义'A'字符



3. 连接文本

示例 1：

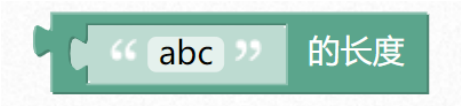



```
#include <CH32V103.h>
#include "myLib/CH32V_ST7735S.h"

SPITFT spi_tft(128,128,PC5,PA2,PA4);

int main(void)
{
  CH32_Init();
  spi_tft.init();
  spi_tft.set_direction(2);
  spi_tft.clear((0xFFFF));
```

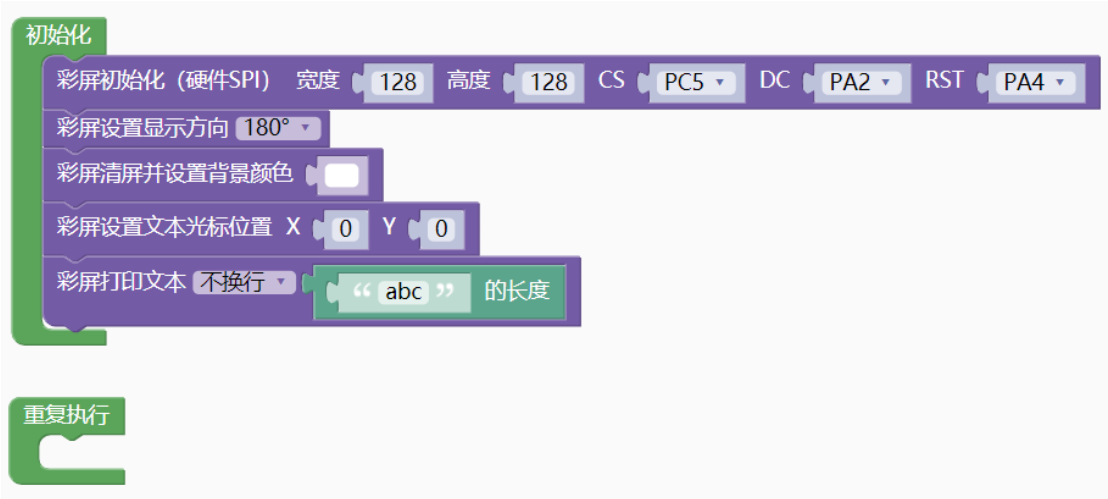
```
  spi_tft.set_cursor(0,0);
  spi_tft.draw_hanzi_12((String("好好") + String("搭搭")));
  while(1){

  }
  return 1;
}
```

彩屏显示“好好搭搭”。

4. 获取字符串长度



示例 1：



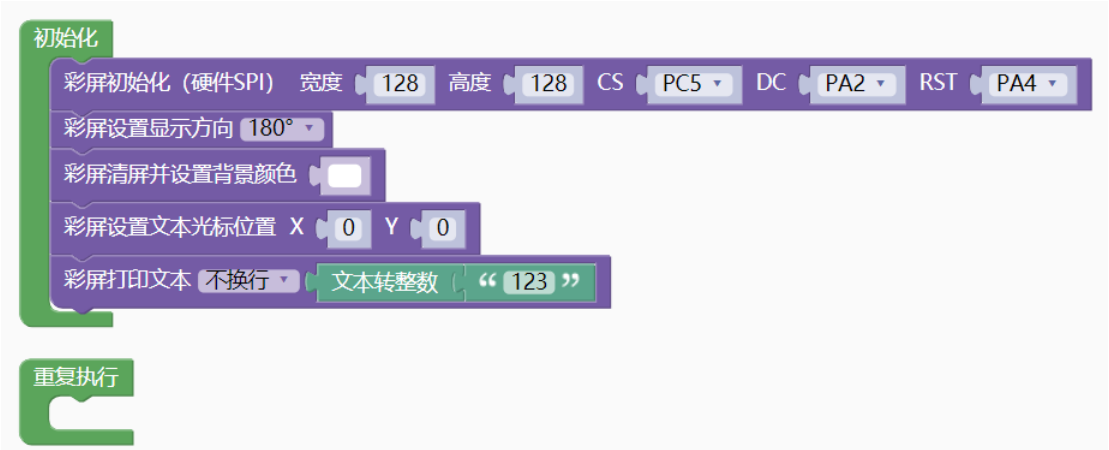

```
#include <CH32V103.h>
#include "myLib/CH32V_ST7735S.h"
#include <string.h>

SPITFT spi_tft(128,128,PC5,PA2,PA4);

int main(void)
{
  CH32_Init();
  spi_tft.init();
  spi_tft.set_direction(2);
  spi_tft.clear((0xFFFF));
  spi_tft.set_cursor(0,0);
  spi_tft.print((strlen("abc")));
  while(1){
```

```
  }
  return 1;
}
```

彩屏显示 3

5. 文本转整数

示例 1：

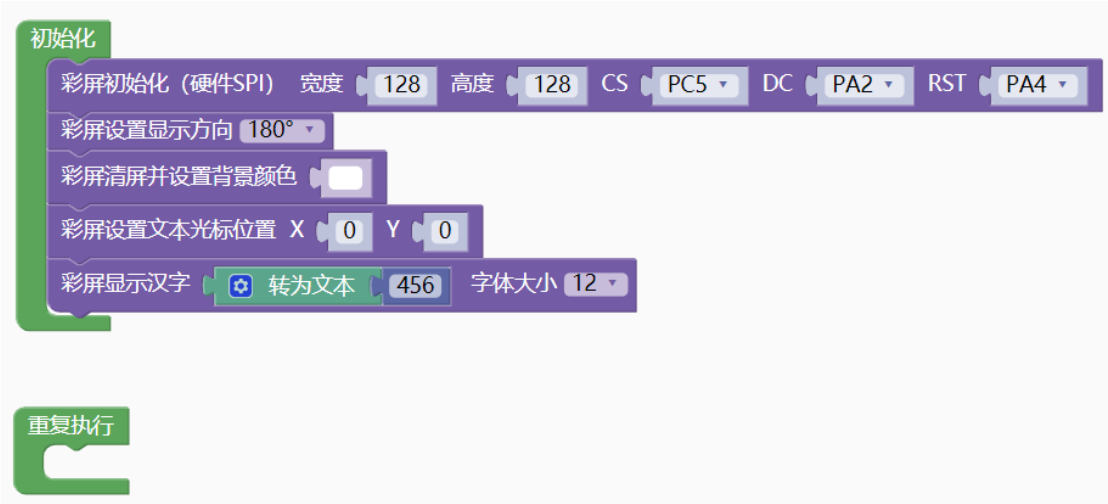



```
#include <CH32V103.h>
#include "myLib/CH32V_ST7735S.h"

SPITFT spi_tft(128,128,PC5,PA2,PA4);

int main(void)
{
  CH32_Init();
  spi_tft.init();
  spi_tft.set_direction(2);
  spi_tft.clear((0xFFFF));
  spi_tft.set_cursor(0,0);
  spi_tft.print(("123".toInt()));
  while(1){

  }
  return 1;
}
```

彩屏显示 123

6. 转文本

示例 1：




```
#include <CH32V103.h>
#include "myLib/CH32V_ST7735S.h"

SPITFT spi_tft(128,128,PC5,PA2,PA4);

int main(void)
{
  CH32_Init();
  spi_tft.init();
  spi_tft.set_direction(2);
  spi_tft.clear((0xFFFF));
  spi_tft.set_cursor(0,0);
  spi_tft.draw_hanzi_12((String(456)));
  while(1){

  }
  return 1;
}
```

彩屏显示 456。

7. 创建数组方式 1

数组类型根据情况选择。

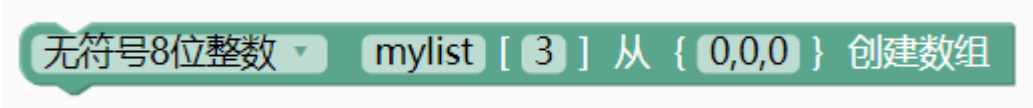
数组初始数据可以通过蓝色小齿轮增加

示例 1：

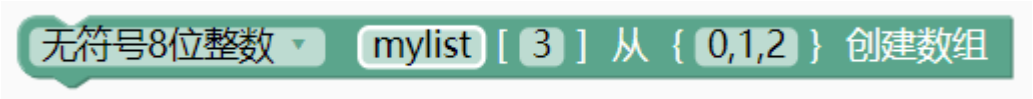
定义一个 8 位数组，数组名为 mylist，数组初始内容为 0，1，2


```
uint8_t mylist[]={0, 1, 2};
```

8. 创建数组方式 2
   前面一种方式，拖动起来比较麻烦，可以用这种方式



示例 1：




```
uint8_t mylist[3]={0,1,2};
```

9. 创建数组方式 3
   只定义长度，内容为空。

10. 获取数组地址

11. 获取数组长度

12. 获取数组指定项目数据

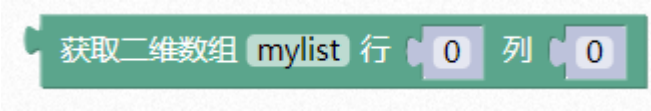
13. 给数组指定项目赋值

14. 创建二维数组方式 1

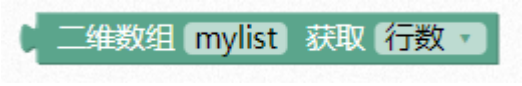
15. 创建二维数组方式 2

16. 给二维数组的指定行列赋值

17. 获取二维数组的指定行列数据



18.获取行/列数



变量模块

1. 创建变量

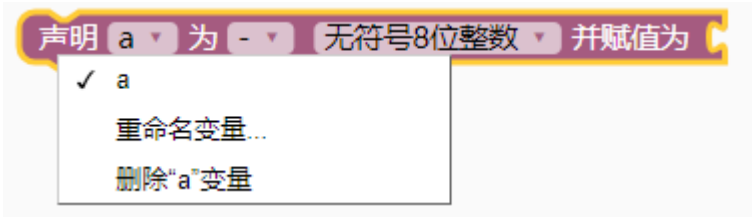
图形化模块支持变量名为中文，系统会自动转义为英文，但是可读性差，一般不建议用中文。

变量栏目默认没有变量，需要点击灰色按钮创建。

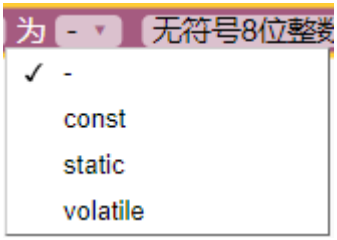
在弹出框中输入变量名，点击确定。再次打开变量栏目出现如下图形块：

2. 变量声明

第一个选项里，可以再次重命名或者删除变量。



第二个选项里，可以设置变量前缀

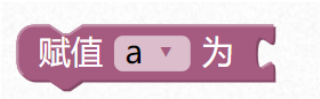


第三个选项里，选择变量的类型

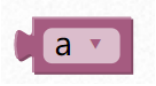
- ✓ 无符号8位整数
- 无符号16位整数
- 无符号32位整数
- 无符号64位整数
- 8位整数
- 16位整数
- 32位整数
- 64位整数
- 长整数
- 单精度浮点数
- 双精度浮点数
- 布尔
- 字符串

```
uint8_t;  //  8 bits
uint16_t; // 16 bits
uint32_t; // 32 bits
Uint64_t; // 64 bits
int8_t; //  8 bits
int16_t;  // 16 bits
int32_t;  // 32 bits
int64_t;  // 64 bits
long;
float;
double;
bool;
String;
```

3. 变量赋值

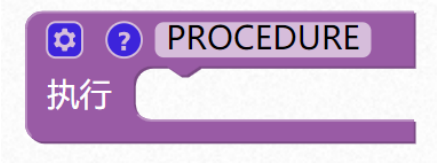


4. 获取变量值



函数模块

1. 定义无返回值函数



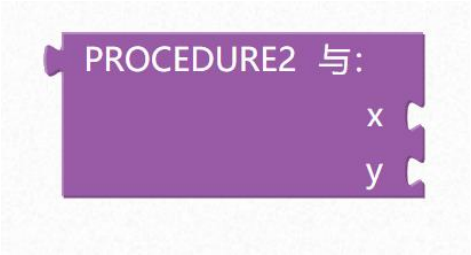
点击蓝色小齿轮，可以添加输入参数，函数名可以自己命名，建议不要用中文。

函数块使用好了后，在函数栏目里会自动出现对应的执行函数块。

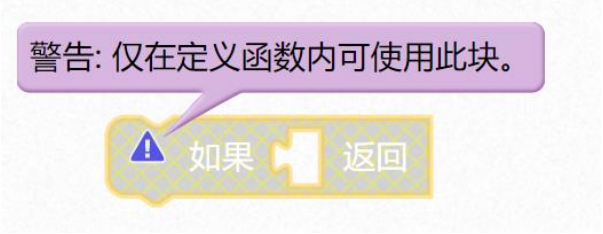
示例 1：
把多个 LED 操作归纳到 led 函数里，让主程序可读性增强。

2. 定义有返回值函数

如需要输入参数，操作和上一节一样。

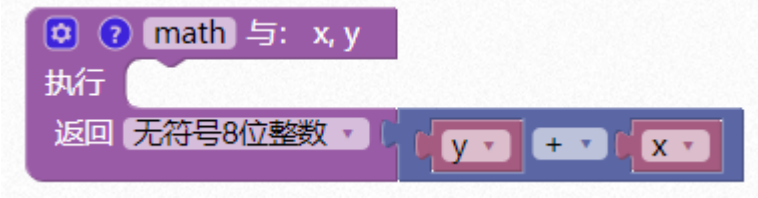


如果需要在函数执行过程中间范围，可以添加如果返回模块。



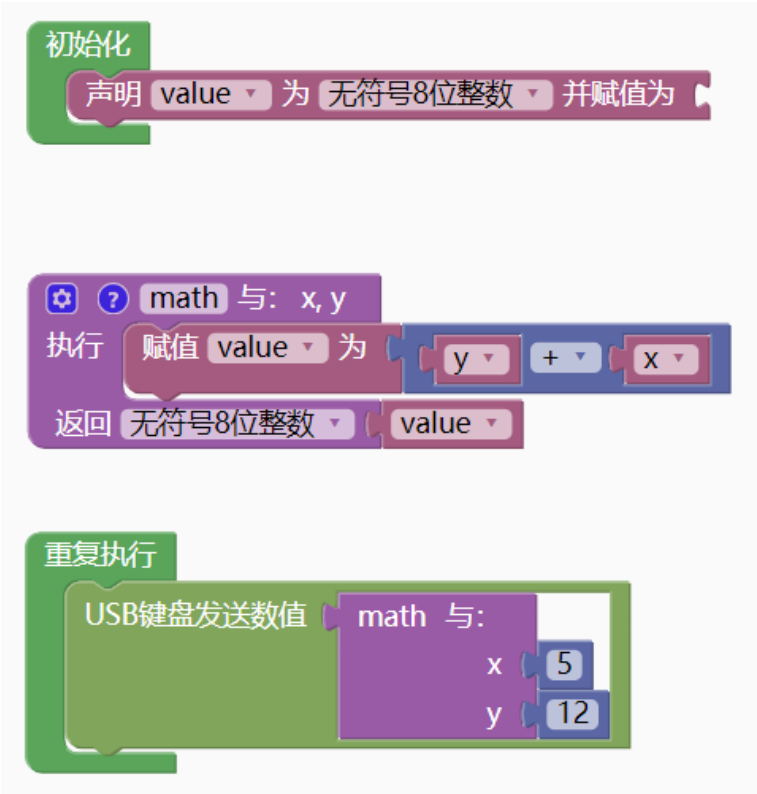
示例 1：

定义数学函数，返回 x+y 运算结果值



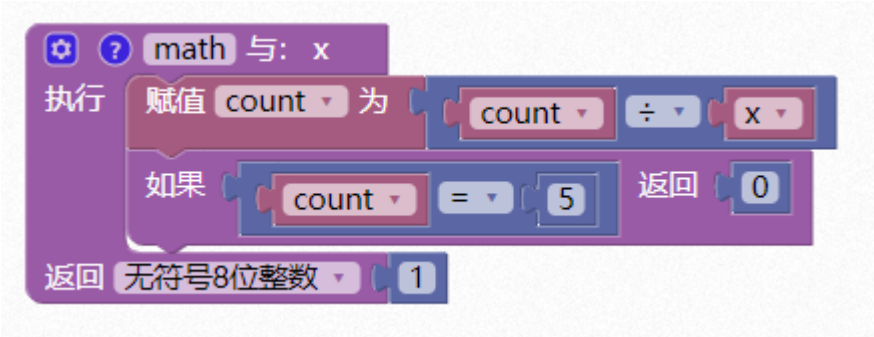
示例 2：

上述程序也可以写成这样



示例 3：

在函数内部执行过程中判断，条件成立直接返回，不用等全部执行完后再返回。



常见问题

1. USB 接入计算机没有 U 盘出现

请确认 USB 的 TpyeC 是否接在左侧的 USB_SLAVE 口。跳线帽 B0 接 G，B1 接 V，USB 接开发板左侧的 USB_SLAVE 口，重新插拔 USB。如果还没有出现 U 盘，请参考第 7 条。

2. 双击天问 Block 软件图标没反应

天问 Block 只支持 64 位系统，32 位系统不能正常运行。如是 Win7 的 64 位系统请右击软件图标设置运行兼容模式为 Win7。

3. 点击运行按键没反应

80%用户为下载工具被杀毒软件删除了，请关闭杀毒软件，重新安装。20%用户为更改了安装路径和目录名称，请安装在磁盘根目录，默认安装。

4. 屏幕显示白屏

烧写过不带屏幕显示的程序，按下 KEY1 键运行程序，就会显示白屏。需要彩屏显示，写彩屏程序并下载运行才能显示。

5. 能否关闭彩屏背光

彩屏右上角有 BL 丝印的跳线帽为彩屏背光电源。

6. 能否用 Keil 开发

Keil 不支持 RISC-V 内核，可以使用沁恒的 MounRiver Studio IDE 开发。

7. 如何重烧 U 盘固件

打开天问 Block 软件，右上角/更多/安装固件驱动程序后，再烧写 U 盘固件，烧写时按提示进行。