天问 51 编程手册

版本	更新内容
V1.0	初始版本

目录

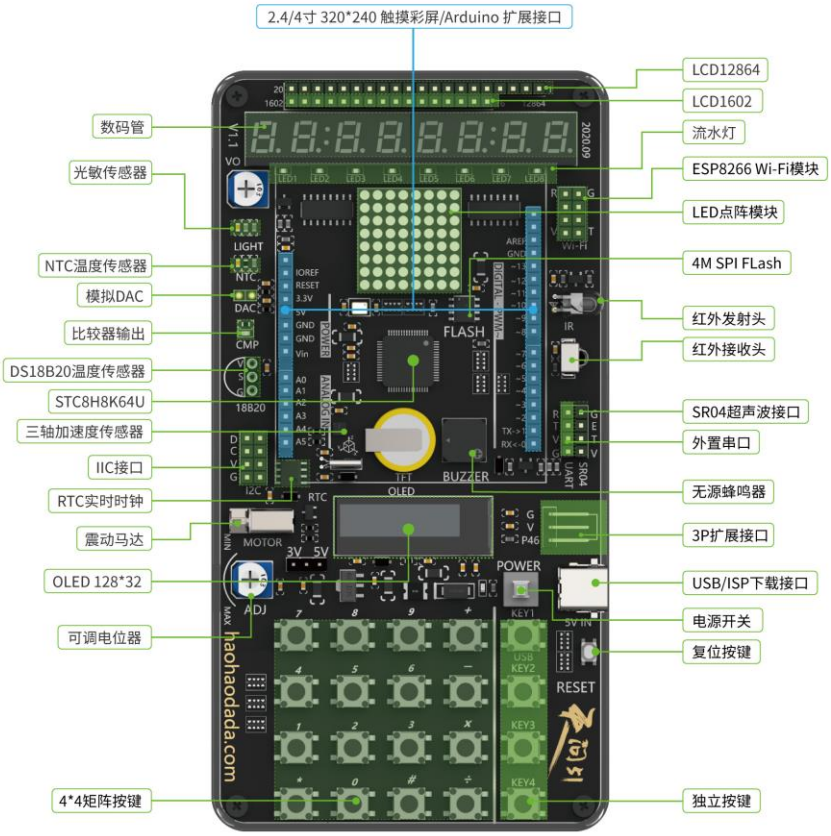
设计初衷

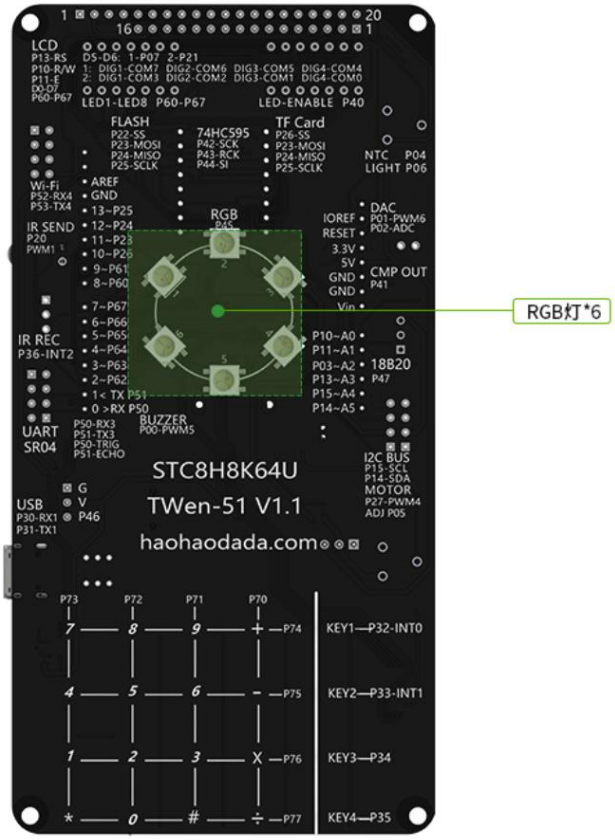
天问，屈原的大作，原意问天，但天是九五之尊，不可问，因此取名天问。天问创作于屈原被放逐后，心中忧愁憔悴之时，愤怒、彷徨而努力呼唤，天文三十问、地理四十二问、历史九十五问，凸现我们中华民族追求真理、探索求知的欲望和决心。这种处境，这种问天的勇气，非常适合我们国家当前的处境，也是我们民族当前处境写照。天问系列开发板，采用国产芯片，每一颗电阻、电容都是国产，百分之百国产，展示中华民族敢问苍天之决心，在芯片国产化上孜孜不倦、努力求真。天问系列开启一个单片机教学和单片机开发新时代，采用国产在线编程编译模式，真正做到了从设计、材料、制作、编程软件，全国产，一个真内循环产品，无惧西方国家的技术封锁。

硬件介绍

天问 51 系列共有天问 51 开发板、天问 51 实验箱、天问 51-Mini、天问 51-Nano、天问 51-Core 五种针对不同应用场合的主板和 STC-LINK、STC-LINK 无线两种下载器。

天问 51 开发板是一款带 USB 的 STC 51 全功能开发板，采用 STC8H8K64U 芯片，支持 USB、ADC、PWM、SPI、IIC 等，板载流水灯、8 位数码管、点阵模块、OLED 显示屏、SPI FLASH、红外发射、红外接收、无源蜂鸣器、4 个独立按键、4*4 矩阵按键、可调电位器、震动马达、RTC 实时时钟、三轴加速度、NTC 温度传感器、光敏传感器、6 个 RGB 彩灯等。





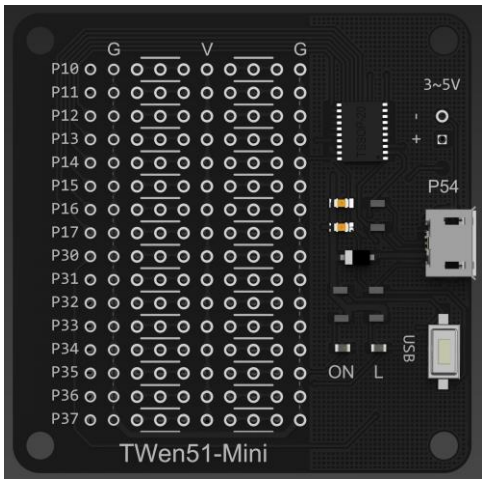
主板详细参数

尺寸	74*145mm
PCB 工艺	A 级 PCB，黑色油墨，沉金工艺
CPU	STC8H8K64U 64K Flash、256 DATA RAM、8K SRAM、UART*4、USB*1、SPI*1、I2C*1、16 位 TIM*5、2 组高级 PWM、硬件 16 位乘除法器、12 位 15 通道 ADC、比较器*1
Flash	4M Bytes SPI FLASH (W25Q32)
RTC	1 个 BM8563 芯片、一个 CR1220 电池
移位寄存器	2 个 74HC595
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
电位器	2 个 10K 可调电位器
三轴加速度传感器	QMA7891
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、16 个矩阵按键

LED	1 个电源 LED、八个流水灯 LED、6 个 RGB LED、输出比较 LED
OLED	1 个 0.91 寸 OLED 显示屏 分辨率 128*32
显示屏	可外接 LCD1602、LCD12864 和 TFT 带触摸彩屏
数码管	2 个 4 位共阳数码管
点阵	1 个 8*8 点阵
蜂鸣器	1 个无源蜂鸣器
马达	1 个震动马达
超声波	可外接 SR04 超声波模块
Wi-Fi	可外接 ESP8266 Wi-Fi 模块
3P 扩展接口	可外接通用模块
I2C 扩展板接口	可外接 2 个通用 I2C 模块
串口扩展接口	可外接通用串口模块
Arduino 扩展接口	可外接 Arduino 扩展板

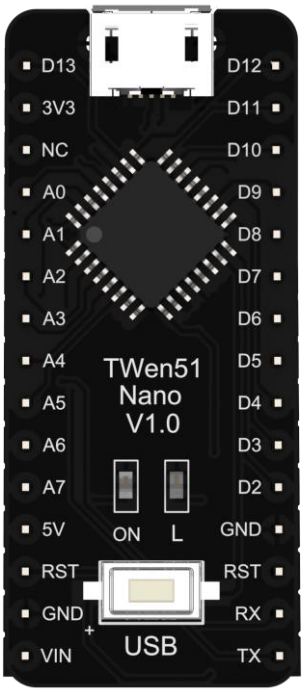
天问 51-Mini 是专为学习开发板的阶段性课程后，快速开发小项目而生，板载低成本 STC8H 芯片，接口采用面包板方式引出，方便焊接分立元器件。通过自己亲手搭建电路、焊接电路、调试电路、编写程序，才能做到真正的融会贯通，学以致用。

芯片采用 STC8H1K08-36I-TSSOP20, 8K Flash、256 字节 RAM、1K 扩展 RAM、4K EEPROM; 2 路串口、1 路 SPI、1 路 I2C、10 位 ADC、GPIO 多达 17 个，芯片价格 0.9 元。



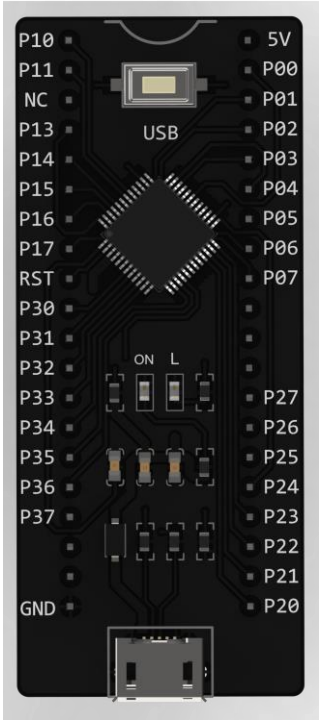
天问 51-Nano 是兼容 Arduino Nano 引脚的核心板，板载 5V、3.3V 稳压芯片，方便插到 Nano 扩展板上，配合常用的 Arduino 模块做快速的开发应用。

芯片采用 STC8H1K16-36I-LQFP32 16K Flash、256 字节 RAM、1K 扩展 RAM、12K EEPROM; 2 路串口、1 路 SPI、1 路 I2C、10 位 ADC、GPIO 多达 29 个，芯片价格 1.4 元。



天问 51-Core 是兼容 89C51/89C52 系列 DIP40 封装的核心板，可以直接替换原有 51 实验箱上的芯片。

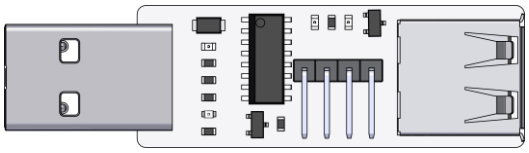
芯片采用 STC8H8K64U-48PIN 64K Flash、256 字节 RAM、8K 扩展 RAM 支持 IAP；2 路串口、1 路 SPI、1 路 I2C、12 位 ADC、GPIO 多达 44 个，芯片价格 2.9 元。



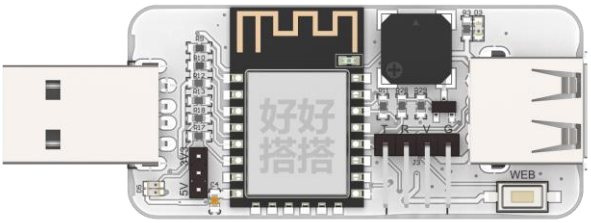
天问 51 实验箱采用核心板和外设模块独立模式，使用时，需要学生用连接线连接引脚，使用相对灵活，搭配更多基础模块和工业应用模块，从基础数字电路、模拟电路实验开始，到微机原理、嵌入式开发系统，最后到 FPGA 芯片设计，让学生对嵌入式有个全面

认识和掌握。

STC-LINK 下载器，芯片内部会自动检测 0x7F 方式实现自动断电烧写程序，非常可靠方便，支持仿真。

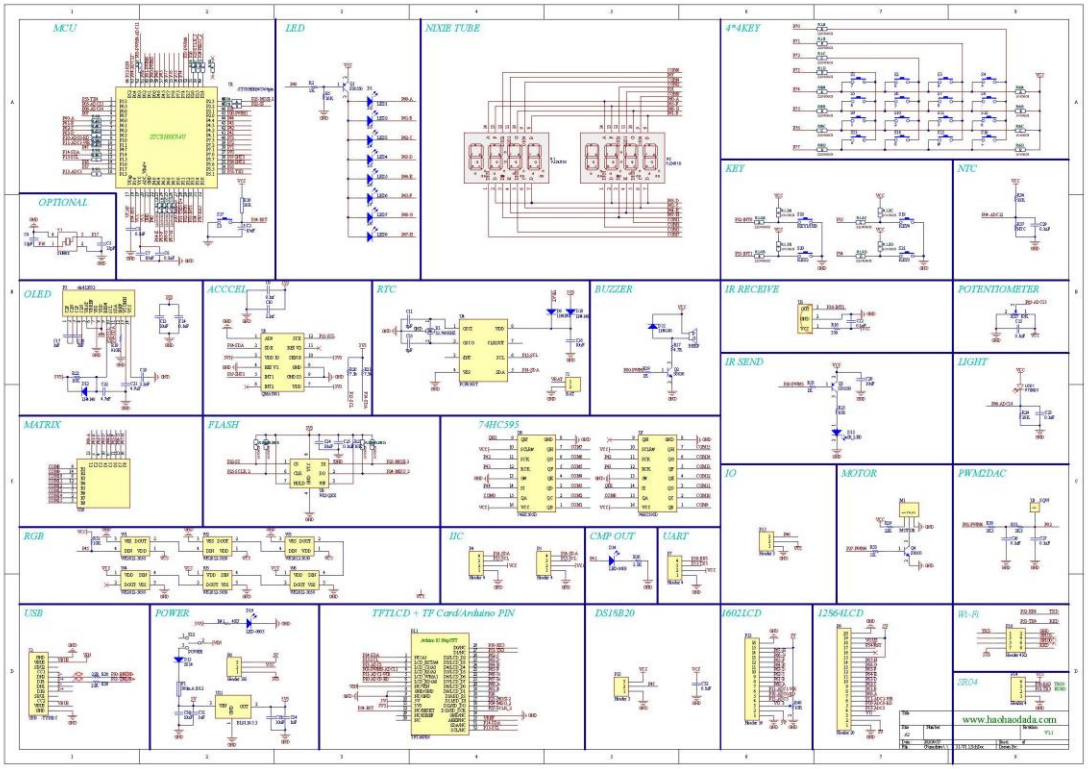


STC-LINK 无线下载器，是 STC-LINK 下载器的升级版，增加了无线下载功能。

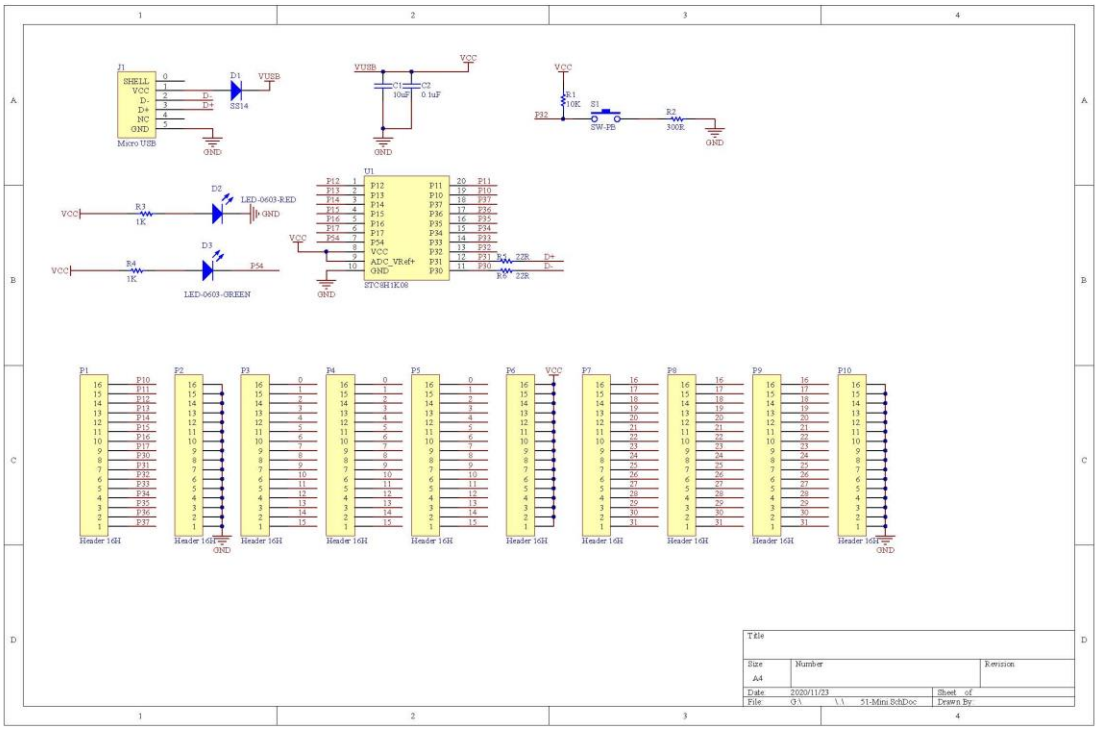


电路原理图

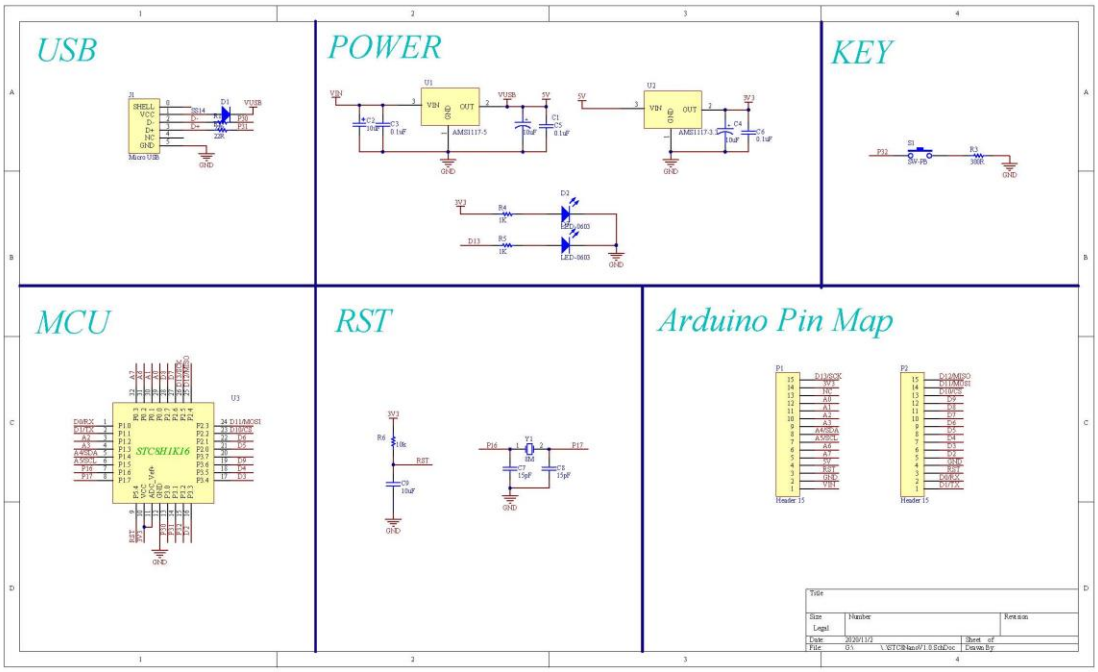
天问 51 开发板原理图



天问 51-Mini 原理图



天问 51-Nano 原理图



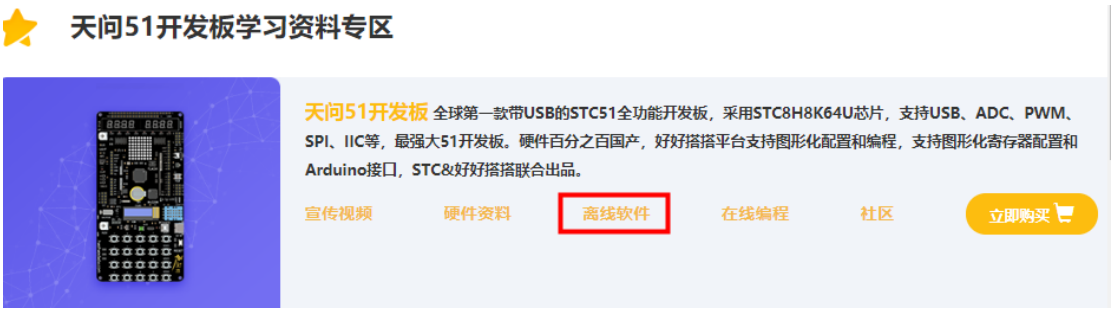
快速上手

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。

下面以天问 51 开发板为例说明，其它开发板操作类似，不再赘述。

第一步：下载好搭 Block 软件

1. 浏览器打开天问 51 资料页 http://tw51.haohaodada.com/
2. 点击离线软件，下载软件

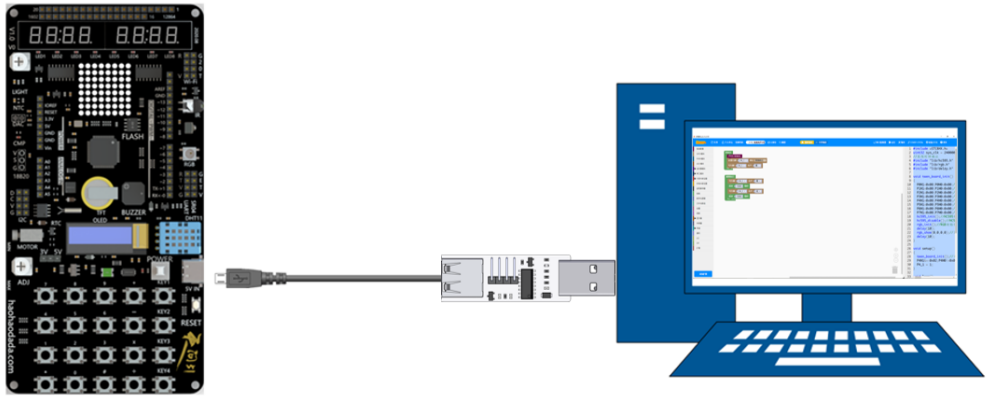


第二步：安装好搭 Block 软件

根据提示默认安装，安装过程中会自动安装 STC-LINK 下载器的 CP210x 驱动。

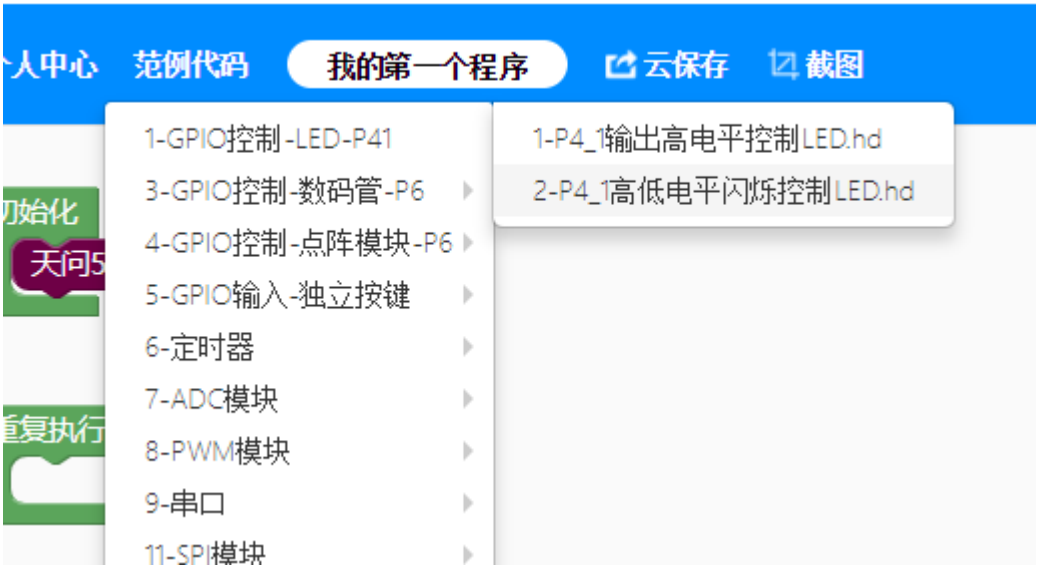
第三步：运行好搭 Block 软件

1. 第一次打开软件，会让你选择主板，请选择 STC8。
2. 连接 STC-LINK 和开发板到电脑，并打开电源开关。



软件会自动识别端口，如果没有识别到请检查驱动和连接。

3. 查看并打开范例程序



4. 点击运行按钮，软件会自动编译并下载程序到设备里。

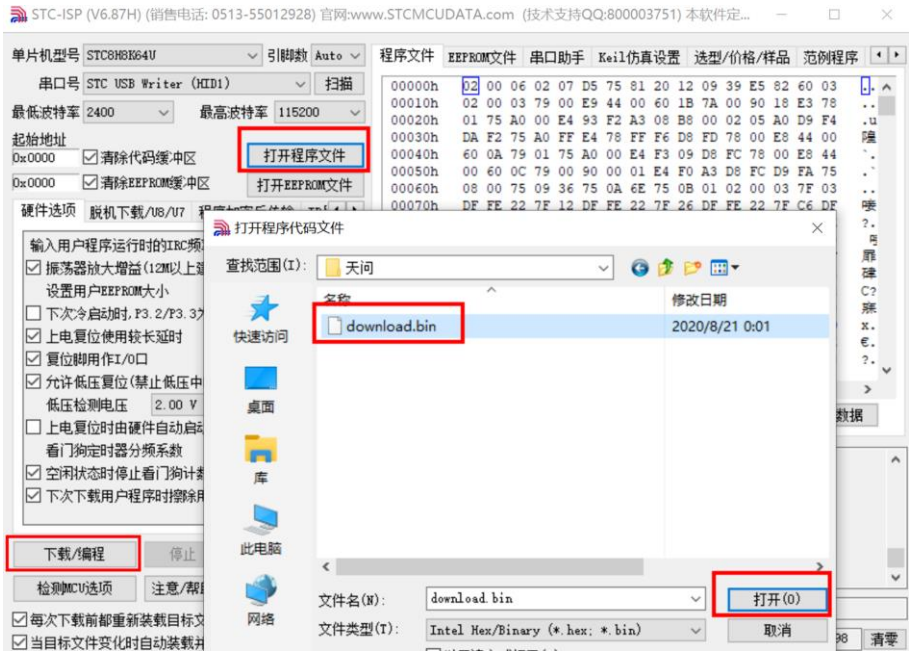
5. 下载完成后，查看运行效果。
6. 点击右上角更多栏目，可以查看文档资料、视频资料等。

其它下载方式

使用 STC-ISP 软件下载

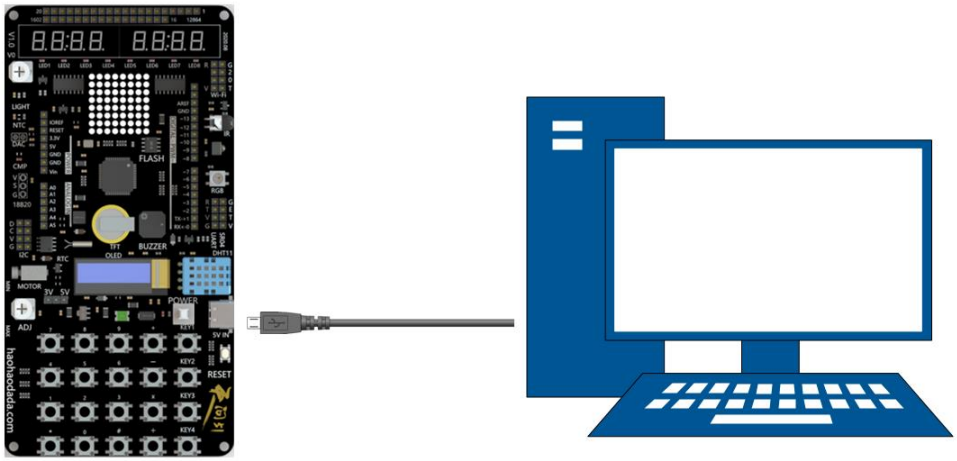
1.硬件连接参考前一节，在资料页面下载并打开 STC-ISP 软件，单片机型号选择对应型号，端口号选择刚才看到的端口号，运行频率选择 24MHz，平台程序默认都以这个频率为准，最下面的两个复选框打勾，文件有更新时会自动下载程序。

2.在 STC-ISP 软件里选择打开程序文件，打开平台编译保存的 Bin 或者 HEX 文件，点击“下载/编程”按钮，等待烧入固件完成。

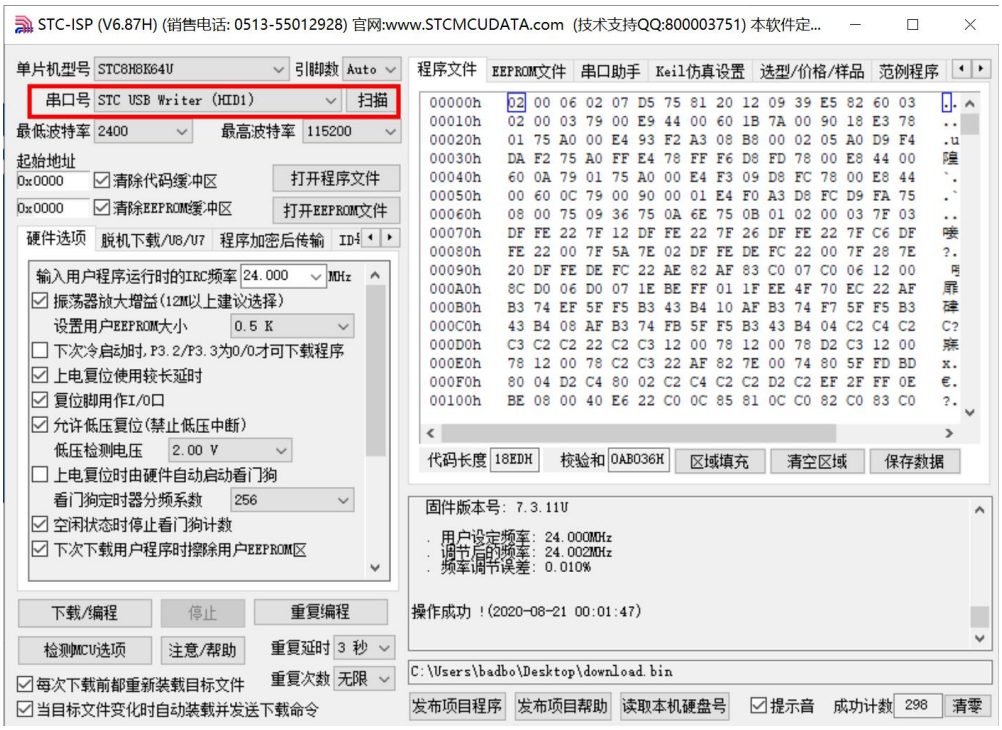


USB 下载方式

1. 用 Type C 数据线直接连接天问开发板到电脑上。



2.关闭电源按键，按住“KEY1/USB”按键，再打开电源按键，电脑会出现 HID 设备，打开 STC-ISP 软件，会看到 STC USB Writer（HID1），其它设置上同。



3.每次下载都要这种方式操作，如果不想断电，可以在程序里设置，把“KEY1/USB”按键配置中断后进入 ISP。这样后续只需要按一下“KEY1/USB”按键就能进入下载模式，不再需要开关电源按键。具体程序见下图。

软件概述

STC8H 的图形化编程系统是针对 51 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。

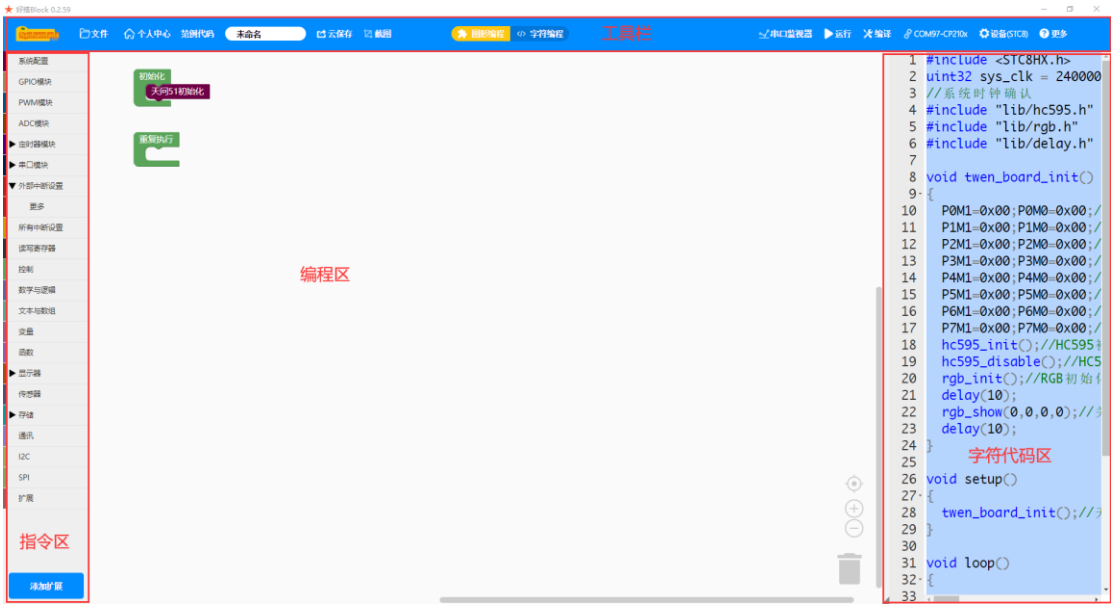
特点：

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，集成了常用的显示模块（LED、RGB、数码管、1602、12864、TFT、OLED、8*8 点阵模块等）、传感器模块（18B20、DHT11、NTC、矩阵键盘、三轴加速度、RTC8563 等模块）；
4. 除了官网内置的模块，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 bin 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;
8. 在线版支持一键导出 Keil 工程。

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

图形化系统基于 Google 的 Blockly 开源框架。

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：

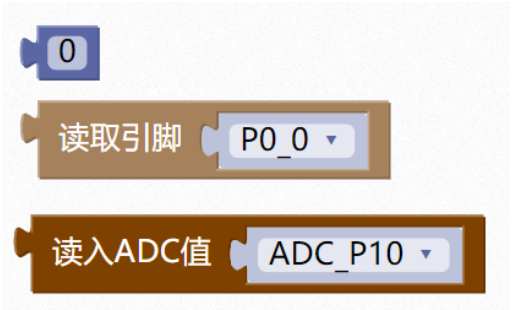
2. 执行块

   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。

示例：

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

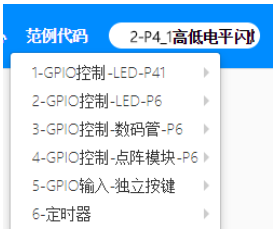
一、文件操作

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. 云保存，可以保存程序到云端，需要个人中心设置帐号信息，如果没有帐号可以免费注册。
6. 查看云端作品，可以点击个人中心。

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。

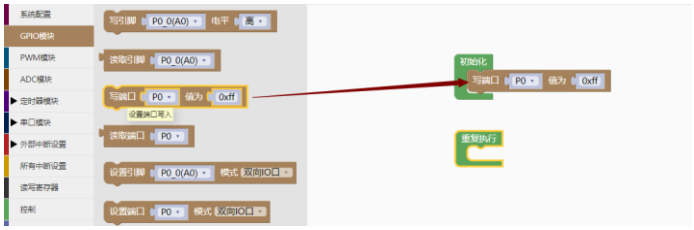


三、图形编程和字符编程切换

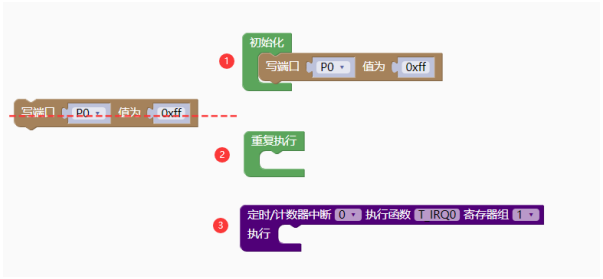
点击图形编程按钮切换到图形化编程模式，按钮变黄色；点击字符编程按钮切换到代码编程模式，按钮变黄色；

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。

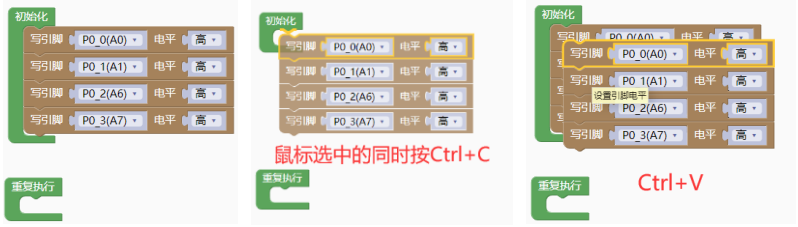


需要注意，编程区默认有如下图所示的初始化和重复执行模块两种，外加中断函数三种图形块为第一层。其余图形化模块都只能放到这些块内部，不能放到其它空白区域，不然会导致生成的代码编译报错。

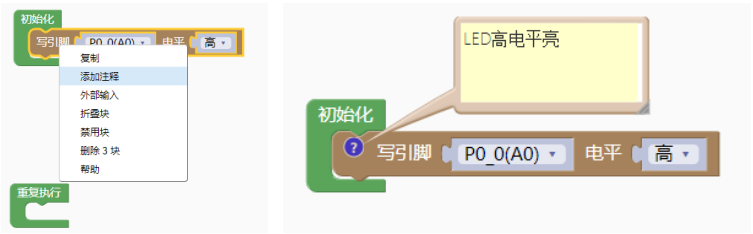


2. 复制块，可以用快捷键“Ctrl+C”，也可以右键选择复制。

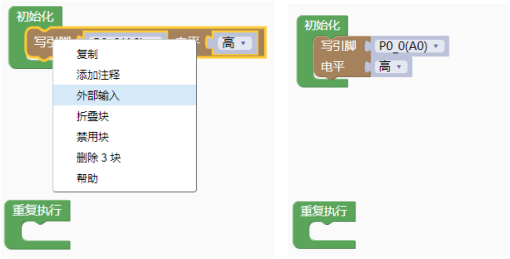
3. 复制多个块，鼠标左键选中多个块的第一个块，拖动多个块移动的时候，按“Ctrl+C”，松开鼠标左键，按“Ctrl+V”粘贴块。



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

1. 全选 selectall
   Ctrl-A

2. 查找下一个 findnext
   Ctrl-K

3. 查找上一个 findprevious
   Ctrl-Shift-K

4. 选择或查找下一个 selectOrFindNext
   Alt-K

5. 选择或查找上一个 selectOrFindPrevious
   Alt-Shift-K

6. 查找 find
   Ctrl-F

7. 选择到开头 selecttostart
   Ctrl-Shift-Home

8. 前往开头 gotostart
   Ctrl-Home

9. 向上选择 selectup
   Shift-Up

10. 向上一行 golineup
    Up

11. 选择到结尾 selecttoend
    Ctrl-Shift-End

12. 前往结尾 gotoend
    Ctrl-End

13. 向下选择 selectdown

Shift-Down

14. 向下一行 golinedown
    Down

15. 展开/折叠 unfold/fold
    Alt-Shift-L

16. 前往这行开头 gotolinestart
    Home

17. 向左移动 gotoleft
    Left

18. 前往这行结尾 gotolineend
    End

19. 向右移动 gotoright
    Right

20. 向上滚动页面 scrollup
    Ctrl-Up

21. 向下滚动页面 scrolldown
    Ctrl-Down

22. 选择到这行开头 selectlinestart
    Shift-Home

23. 选择到这行结尾 selectlineend
    Shift-End

24. 跳转至匹配的括号 jumptomatching
    Ctrl-P

25. 选择匹配的括号 selecttomatching
    Ctrl-Shift-P

26. 扩展至匹配的括号 expandToMatching
    Ctrl-Shift-M

27. 删除一行 removeline
    Ctrl-D

28. 复制一行 duplicateSelection
    Ctrl-Shift-D

29. 排列行 sortlines
    Ctrl-Alt-S

30. 注释行 togglecomment
    Ctrl-/

31. 注释块 toggleBlockComment
    Ctrl-Shift-/

32. 将选中的数字加一 modifyNumberUp
    Ctrl-Shift-Up

33. 将选中的数字减一 modifyNumberDown
    Ctrl-Shift-Down

34. 替换 replace
    Ctrl-H

35. 撤销 undo
    Ctrl-Z

36. 重做 redo
    Ctrl-Shift-Z

37. 向上复制这一行 copylinesup
    Alt-Shift-Up

38. 将这行向上移动 movelinesup
    Alt-Up

39. 向下复制这一行 copylinesdown
    Alt-Shift-Down

40. 将这一行向下移动 movelinesdown
    Alt-Down

41. 删除 del
    Delete

42. 退格 backspace
    Backspace

43. 剪切或删除 cut_or_delete
    Shift-Delete

44. 删除至行首 removetolinestart
    Alt-Backspace

45. 删除至行尾 removetolineend
    Alt-Delete

46. 删除左边的单词 removewordleft
    Ctrl-Backspace

47. 删除右边的单词 removewordright
    Ctrl-Delete

48. 向左缩进 outdent
    Shift-Tab

49. 向右缩进 indent
    Tab

50. 整行向左缩进 blockoutdent
    Ctrl-[

51. 整行向右缩进 blockindent
    Ctrl-]

52. 转为大写 touppercase
    Ctrl-U

53. 转为小写 tolowercase
    Ctrl-Shift-U

54. 选中整行 expandtoline
    Ctrl-Shift-L

55. 在上方插入指针 addCursorAbove
    Ctrl-Alt-Up

56. 在下方插入指针 addCursorBelow
    Ctrl-Alt-Down

57. 在上方插入唯一指针 addCursorAboveSkipCurrent

Ctrl-Alt-Shift-Up

58. 在下方插入唯一指针 addCursorBelowSkipCurrent
    Ctrl-Alt-Shift-Down

59. 选择前一个 selectNextBefore
    Ctrl-Alt-Shift-Left

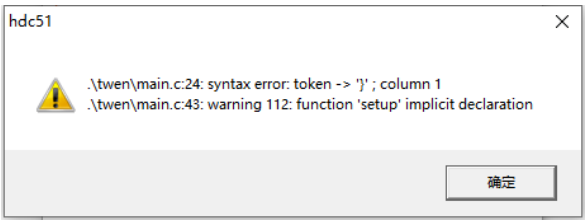
60. 选择后一个 selectNextAfter
    Ctrl-Alt-Shift-Right

61. 查找全部 findAll
    Ctrl-Alt-K

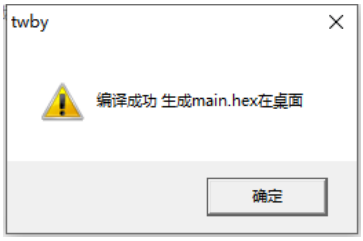
六、运行和编译

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。

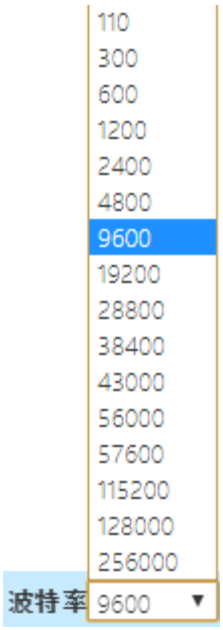


七、串口监视器

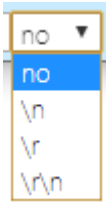
在工具栏点击打开监视器按钮，软件下放会弹出串口监视器界面。

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。

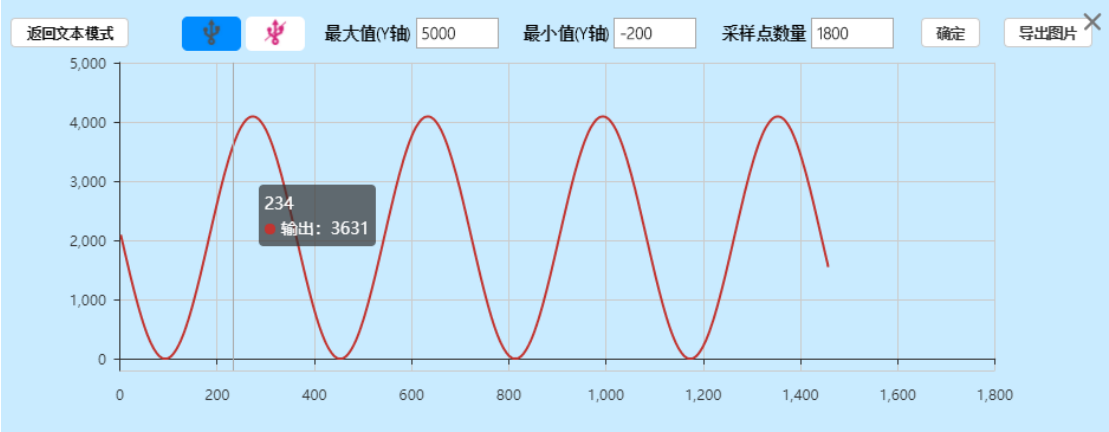
7. 清空显示数据

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。

八、截图

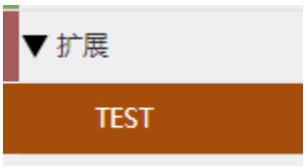
在图形编程模式下，点击截图按钮，会自动把编程器的所有图形块程序保存为图片格式。

九、添加扩展

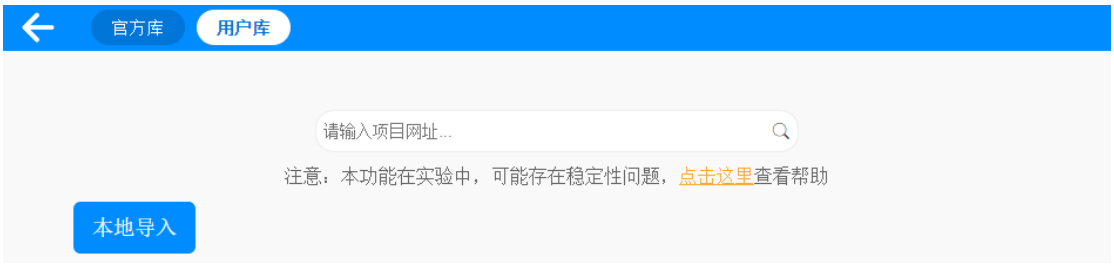
点击添加扩展按钮，弹出库管理界面。有关库的开发，请查看库开发文档，本文不再赘述。

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，点加号图标，软件会通过网络下载库到本地，如果库版本有更新，会自动更新。点击加载按钮或者移除按钮管理库。

加载好的库，会在指令区最下面的扩展图形化下拉列表里显示。



用户库为提供给开发者本地测试库功能使用。



十、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。

STC8 外设模块

系统配置模块

1. 系统频率设置，在 PWM、定时器、串口、EEPROM 模块中调用了这个参数。在图形化编程模板里已经默认设置为 24M，此参数，初学者不要随便修改。

```
uint32 sys_clk = 24000000;
```

2. 设置分频系数，此参数，初学者不要随便修改。


```
#include "lib/sysclock.h"//引入头文件
sysclock_set_hir_irc(0);//内部高速时钟分频设置
sysclock_set_32k_irc(0);//内部低速时钟分频设置
sysclock_set_xosc(0);//外部时钟分频设置
```

3. 设置 PWM 占空比的分母，系统默认 1000，会影响 PWM 占空比的输出。此参数，初学者不要随便修改。


```
#define PWM_DUTY_MAX 1000
```

示例：

设置最大占空比值为 1000，当设置占空比为 500 时，输出为 50%。
设置最大占空比值为 500，当设置占空比为 500 时，输出为 100%。

4. 设置程序自动进入 ISP 下载模式，在 USB 下载模式场合里，配合按键中断使用


```
IAP_CONTR = 0x60;
```

示例：按下按键 KEY1，进入 ISP 下载模式


```
#include <STC8HX.h>

void INT0(void) interrupt 0 using 1{
  IAP_CONTR = 0x60;
}

void setup()
{
  IT0 = 0;
  EX0 = 1;
```

```
  EA = 1;
}

void loop()
{

}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

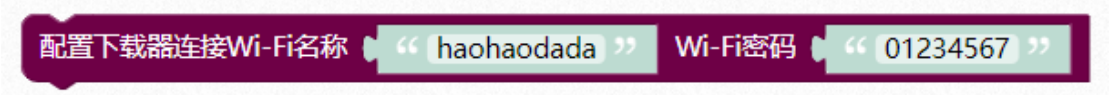
5. 当硬件设备为天问 51 开发板时，需要在初始化里添加，用来初始化天问 51 板载的 595 移位寄存器默认输出。另外关闭 RGB，因为 RGB 为单总线控制，下载时会引起 RGB 模块随机亮，影响使用体验。同时把所有端口默认设置为双向 IO 口。

天问51初始化

```
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}
```

```
void setup()
{
  twen_board_init();//天问 51 初始化
}

void loop()
{

}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

6. 配置 STC-LINK 无线下载器的网络



具体使用请查看 STC-LINK 无线下载器使用说明。

GPIO 模块

传统 51 芯片，引脚默认配置为准双向 IO 可以直接读写就可以输入或输出，STC8H 系列单片机，引脚除 P3.0 和 P3.1 外默认配置为高阻输入，只能读入不能输出，需要设置才能切换为其他模式。

1. 写引脚的电平状态，1 为高电平；0 为低电平。


```
P0_0 = 1;
```

2. 读引脚的电平状态，引脚电平如果为高电平，返回 1；低电平，返回 0。


```
P0_0;
```

3. 写端口的电平状态，用一个字节的 8 个位，对应端口的 8 个引脚。1 为高电平；0 为低电平。


```
P0 = 0xff;
```

上述程序为设置 P0 端口的所有引脚电平状态都为高。

4. 读端口的电平状态，返回一个字节，用一个字节的 8 个位，对应端口的 8 个引脚。引脚电平如果为高电平，返回 1；低电平，返回 0。


```
P0;
```

上述程序，如果 P0_0 引脚为高电平，其它引脚为低电平，则范围值等于 0x01。

5. 设置引脚工作模式：双向 IO 口、推挽输出、高阻输入、开漏输出。

PnM0 与 PnM1 的组合方式如下表所示

PnM1	PnM0	I/O 口工作模式
0	0	准双向口（传统8051端口模式，弱上拉） 灌电流可达20mA，拉电流为270～150μA（存在制造误差）
0	1	推挽输出（强上拉输出，可达20mA，要加限流电阻）
1	0	高阻输入（电流既不能流入也不能流出）
1	1	开漏输出（Open-Drain），内部上拉电阻断开 开漏模式既可读外部状态也可对外输出（高电平或低电平）。如要正确读外部状态或需要对外输出高电平，需外加上拉电阻，否则读不到外部状态，也对外输不出高电平。

注：n = 0, 1, 2, 3, 4, 5, 6, 7

主要为设置 PnM1 和 PnM0 两个寄存器的值。


```
P0M1&=~0x01;P0M0&=~0x01;//双向 IO 口
```

6. 设置端口工作模式：双向 IO 口、推挽输出、高阻输入、开漏输出。


```
P0M1=0x00;P0M0=0x00;//双向 IO 口
```

7. 设置引脚是否启用内置上拉电阻，主要设置 PnPU 寄存器。

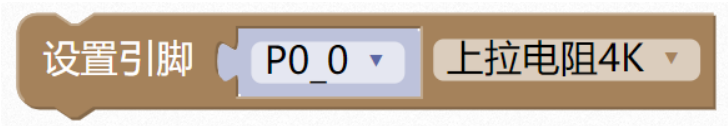
端口上拉电阻控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	FE15H	-	-	-	P54PU	P53PU	P52PU	P51PU	P50PU
P6PU	FE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU
P7PU	FE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU

端口内部4.1K上拉电阻控制位（注：P3.0和P3.1口上的上拉电阻可能会略小一些）

0：禁止端口内部的 4.1K 上拉电阻

1：使能端口内部的 4.1K 上拉电阻




```
P0PU|=0x01;//上拉电阻 4K
```

8. 设置引脚电流大小，主要设置 PnDR 寄存器。

端口驱动电流控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0DR	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR
P1DR	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR
P2DR	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR
P3DR	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR
P4DR	FE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR
P5DR	FE2DH	-	-	-	P54DR	P53DR	P52DR	P51DR	P50DR
P6DR	FE2EH	P67DR	P66DR	P65DR	P64DR	P63DR	P62DR	P61DR	P60DR
P7DR	FE2FH	P77DR	P76DR	P75DR	P74DR	P73DR	P72DR	P71DR	P70DR

控制端口的驱动能力

0：一般驱动能力

1：增强驱动能力


```
P0DR|=0x01;//强电流
```

PWM 模块

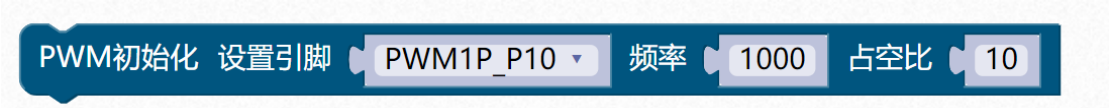
STC8H 系列的单片机内部集成了两组高级 PWM 定时器，两组 PWM 的周期可不同，可分别单独设置。第一组可配置成 4 对互补/对称/死区控制的 PWM，第二组可配置成 4 路 PWM 输出或捕捉外部信号。两组 PWM 定时器内部的计数器时钟频率的分频系数为 1～65535 之间的任意数值。

第一组 PWM 定时器有 4 个通道(PWM1P/PWM1N、PWM2P/PWM2N、PWM3P/PWM3N、PWM4P/PWM4N)，每个通道都可独立实现 PWM 输出（可设置带死区的互补对称 PWM 输出)、捕获和比较功能；第二组 PWM 定时器有 4 个通道 (PWM5、PWM6、PWM7、PWM8)，每个通道也可独立实现 PWM 输出、捕获和比较功能。两组 PWM 定时器唯一的区别是第一组可输出带死区的互补对称 PWM，而第二组只能输出单端的 PWM，其他功能完全相同。下面关于高级 PWM 定时器的介绍只以第一组为例进行说明。

当使用第一组 PWM 定时器输出 PWM 波形时，可单独使能 PWM1P/PWM2P/PWM3P/PWM4P 输出，也可单独使能 PWM1N/PWM2N/PWM3N/PWM4N 输出，若单独使能了 PWMxP 输出后，则不能再独立使能 PWMxN 输出，除非是输出互补对称输出才可以；反之，若单独使能了 PWMxN 输出后，也不能再独立使能 PWMxP 的输出。若需要使用第一组 PWM 定时器进行捕获功能或者测量脉宽时，输入信号只能从每路的正端输入，即只有 PWM1P/PWM2P/PWM3P/PWM4P 才有捕获功能和测量脉宽功能。另外，两组高级 PWM 定时器对外部信号进行捕获时，可选择上升沿捕获或者下降沿捕获，但不能同时既捕获上升沿又捕获下降沿。如果需要同时捕获上升沿和下降沿，则可将输入信号同时接入到两路 PWM，使能其中一路捕获上升沿，另外一路捕获下降沿即可，或者也可选择 STC8G 系列的 PCA/CCP/PWM 功能，则可同时捕获上升沿和下降沿。

图形化模块只提供了常用的 PWM 功能，至于输入捕获这些高级功能，需要用代码实现。

1. 初始化设置引脚 PWM 频率和占空比



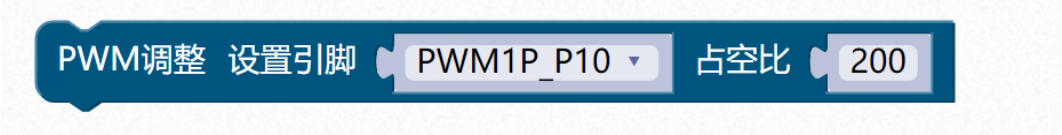

```
#include <STC8HX.h>//引入头文件
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/PWM.h"//引入头文件

pwm_init(PWM1P_P10, 1000, 10);//pwm 初始化三个参数分别是引脚、频率、占空比
10/PWM_DUTY_MAX
```

引脚可以通过下拉菜单选择

- ✓ PWM1P_P10
- PWM1N_P11
- PWM1P_P20
- PWM1N_P21
- PWM1P_P60
- PWM1N_P61
- PWM2P_P12
- PWM2N_P13
- PWM2P_P22
- PWM2N_P23
- PWM2P_P62
- PWM2N_P63
- PWM3P_P14
- PWM3N_P15
- PWM3P_P24
- PWM3N_P25
- PWM3P_P64
- PWM3N_P65
- PWM4P_P16
- PWM4N_P17
- PWM4P_P26
- PWM4N_P27
- PWM4P_P66
- PWM4N_P67
- PWM4P_P34
- PWM4N_P33
- PWM5_P20
- PWM5_P17
- PWM5_P00
- PWM5_P74
- PWM6_P21
- PWM6_P54
- PWM6_P01
- PWM6_P75
- PWM7_P22
- PWM7_P33
- PWM7_P02
- PWM7_P76
- PWM8_P23
- PWM8_P34
- PWM8_P03
- PWM8_P77

这里需要注意占空比，系统配置里有一个 PWM 最大占空比的模块，我们设置的占空比为和这个最大占空比的比值。

2. PWM 调整占空比值，一般用在程序运行过程中需要动态改变占空比输出时。




```
pwm_duty(PWM1P_P10, 200);//pwm 调整三个参数分别是引脚、频率、占空比
10/PWM_DUTY_MAX
```

3. PWM 调整频率和占空比，一般用在程序运行过程中需要动态改变频率和占空比输出时。


```
pwm_freq_duty(PWM1P_P10, 1000, 10);//pwm 调整三个参数分别是引脚、频率、占空
比 10/PWM_DUTY_MAX
```

ADC 模块

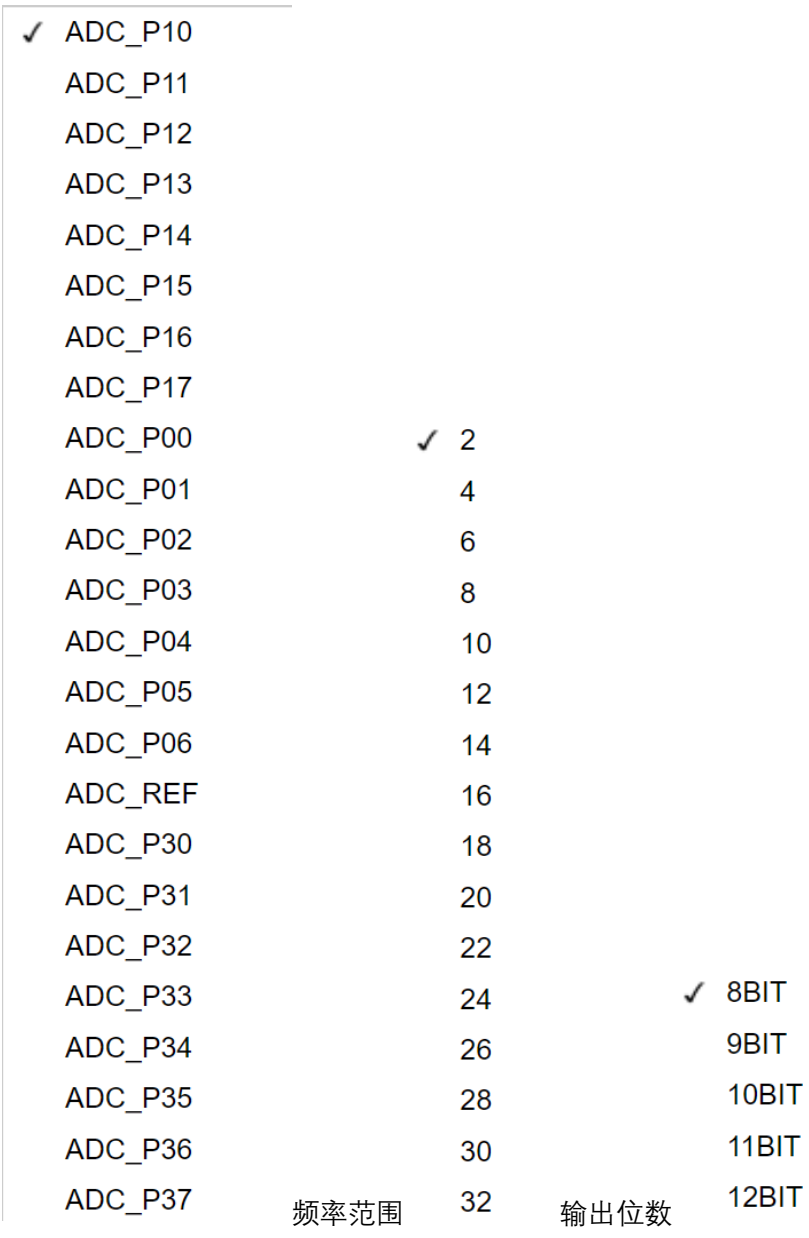
STC8H1K16 和 STC8H1K08 系列单片机内部集成了一个 10 位高速 A/D 转换器，STC8H3K64S4、STC8H3K64S2、STC8H8K64U 系列单片机内部集成了一个 12 位高速 A/D 转换器。ADC 的时钟频率为系统频率 2 分频再经过用户设置的分频系数进行再次分频（ADC 的时钟频率范围为 SYSclk/2/1 ~ SYSclk/2/16）。

ADC 转换结果的数据格式有两种：左对齐和右对齐。可方便用户程序进行读取和引用。注意：ADC 的第 15 通道只能用于检测内部参考电压，参考电压值出厂时校准为 1.19V，由于制造误差以及测量误差，导致实际的内部参考电压相比 1.19V，大约有±1%的误差。如果用户需要知道每一颗芯片的准确内部参考电压值，可外接精准参考电压，然后利用 ADC 的第 15 通道进行测量标定。

1. ADC 初始化设置引脚，频率，输出位数。
   图形化模块默认转换结果为右对齐，需要注意采用的芯片型号支持的 AD 位数，如果是 10 位的 AD，即使选择了 12 位，输出也还是 10 位。


```
#include <STC8HX.h>//引入头文件
```

```
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/ADC.h"//引入头文件

adc_init(ADC_P10, ADC_SYSclk_DIV_2, ADC_8BIT);//ADC 初始化，三个参数 ADC 引脚，时钟分频双数 2-32，输出值位数 12BIT 最大分率-12 位的 ADC 输出 12 位，10 位的输出 10 位
```



2. 读取 ADC 采样值。
   如果 10 位 AD，采样值范围为 0-1023；12 位 AD，采样值范围为 0-4095；


```
adc_read(ADC_P10)
```

定时器模块

STC8H 系列单片机内部设置了 5 个 16 位定时器/计数器。5 个 16 位定时器 T0、T1、T2、T3 和 T4 都具有计数方式和定时方式两种工作方式。对定时器/计数器 T0 和 T1，用它们在特殊功能寄存器 TMOD 中相对应的控制位 C/T 来选择 T0 或 T1 为定时器还是计数器。对定时器/计数器 T2，用特殊功能寄存器 AUXR 中的控制位 T2_C/T 来选择 T2 为定时器还是计数器。对定时器/计数器 T3，用特殊功能寄存器 T4T3M 中的控制位 T3_C/T 来选择 T3 为定时器还是计数器。对定时器/计数器 T4，用特殊功能寄存器 T4T3M 中的控制位 T4_C/T 来选择 T4 为定时器还是计数器。定时器/计数器的核心部件是一个加法计数器，其本质是对脉冲进行计数。只是计数脉冲来源不同：如果计数脉冲来自系统时钟，则为定时方式，此时定时器/计数器每 12 个时钟或者每 1 个时钟得到一个计数脉冲，计数值加 1；如果计数脉冲来自单片机外部引脚，则为计数方式，每来一个脉冲加 1。

当定时器/计数器 T0、T1 及 T2 工作在定时模式时，特殊功能寄存器 AUXR 中的 T0x12、T1x12 和 T2x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T0、T1 和 T2 进行计数。当定时器/计数器 T3 和 T4 工作在定时模式时，特殊功能寄存器 T4T3M 中的 T3x12 和 T4x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T3 和 T4 进行计数。当定时器/计数器工作在计数模式时，对外部脉冲计数不分频。

定时器/计数器 0 有 4 种工作模式：模式 0（16 位自动重装载模式），模式 1（16 位不可重装载模式），模式 2（8 位自动重装模式），模式 3（不可屏蔽中断的 16 位自动重装载模式）。定时器/计数器 1 除模式 3 外，其他工作模式与定时器/计数器 0 相同。T1 在模式 3 时无效，停止计数。定时器 T2 的工作模式固定为 16 位自动重装载模式。T2 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。定时器 3、定时器 4 与定时器 T2 一样，它们的工作模式固定为 16 位自动重装载模式。T3/T4 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。

1. 定时器初始化设置。
   图形化模块默认为模式 0（16 位自动重装载模式）


```
TMOD |= 0x00;    //模式 0
TL0 = 0x37;    //设定定时初值
TH0 = 0xff;    //设定定时初值
```

2. 启动定时器。
   设置 TRn 寄存器为 1。


```
TR0 = 1;// 启动定时器
```

3. 停止定时器。
   设置 TRn 寄存器为 0。


```
TR0 = 0;// 停止定时器
```

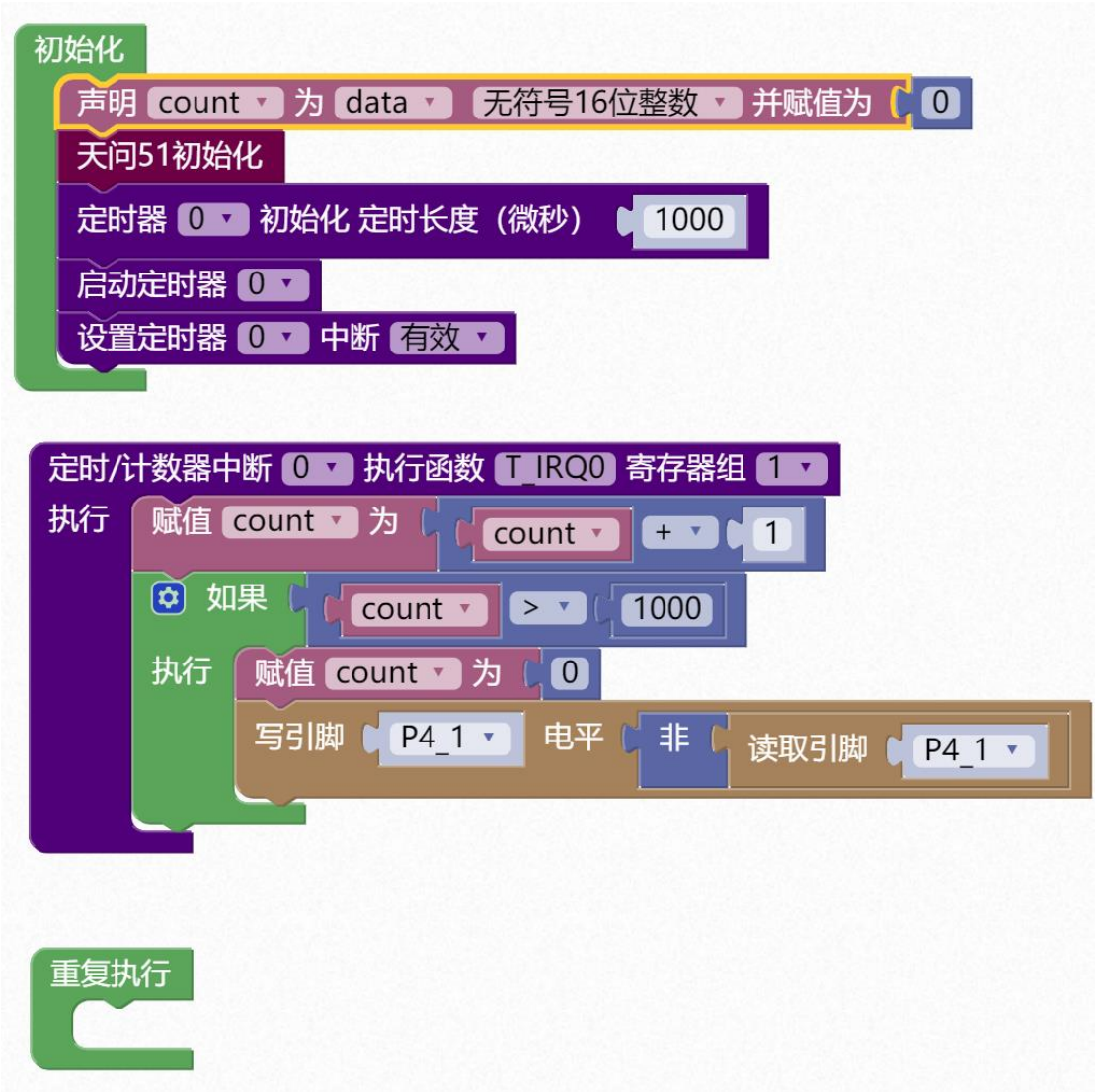
4. 设置定时器的中断开关


```
EA = 1; // 控制总中断
ET0 = 1; // 控制定时器中断
```

5. 定时器中断函数
   定时器用的中断号为 1，寄存器组默认使用 1。


```
void T_IRQ0(void) interrupt 1 using 1{

}
```

示例 1：定时器 0，配置为 16 位自动重载模式，周期为 1 毫秒，打开中断，中断里每隔 1 秒反转一次引脚状态，现象为 P41 上的 LED，一秒亮，一秒灭。




```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"

uint16 count = 0;

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
```

```
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void Timer0Init(void) //1000 微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  count = count + 1;
  if(count > 1000){
    count = 0;
    P4_1 = !P4_1;
  }
}

void setup()
{
  twen_board_init();//天问 51 初始化
  Timer0Init();
  TR0 = 1;// 启动定时器
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
}

void loop()
{

}

void main(void)
{
  setup();
  while(1){
    loop();
```

```
  }
}
```

定时器模块还有更多选项，里面是一些直接读写寄存器的模块，供高级应用。

6. 读定时器溢出标志。
   读取 TFn 寄存器数值。


```
TF0
```

7. 清楚定时器溢出标志。
   设置 TFn 寄存器数值为 0。


```
TF0 = 0;
```

8. 读定时器计数寄存器 TLn/THn 数值。


```
TL0
```

9. 设置定时器计数寄存器 TLn/THn 数值。


```
TL0 = 0xff;
```

10. 读定时器模式寄存器 TMOD/控制寄存器 TCON 数值。


```
TMOD
```

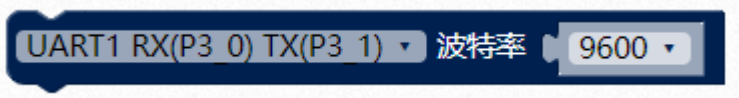
11. 设置定时器模式寄存器 TMOD/控制寄存器 TCON 数值。


```
TMOD = 0xff;
```

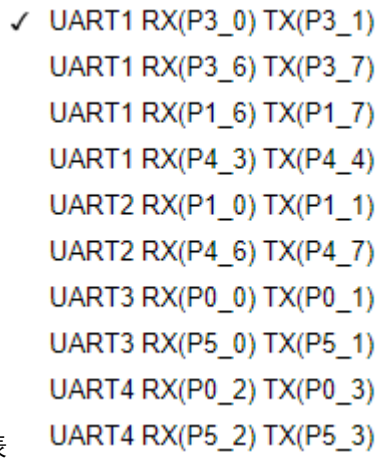
串口模块

STC8H 系列单片机具有 4 个全双工异步串行通信接口。每个串行口由 2 个数据缓冲器、一个移位 寄存器、一个串行控制寄存器和一个波特率发生器等组成。每个串行口的数据缓冲器由 2 个互相独立的接收、发送缓冲器构成，可以同时发送和接收数据。 STC8 系列单片机的串口 1 有 4 种工作方式，其中两种方式的波特率是可变的，另两种是固定的，以供不同应用场合选用。串口 2/串口 3/串口 4 都只有两种工作方式，这两种方式的波特率都是可变的。用户可用软件设置不同的波特率和选择不同的工作方式。主机可通过查询或中断方式对接收/发送进行程序处理，使用十分灵活。串口 1、串口 2、串口 3、串口 4 的通讯口均可以通过功能管脚的切换功能切换到多组端口，从而可以将一个通讯口分时复用为多个通讯口。

1. 串口引脚和波特率设置。

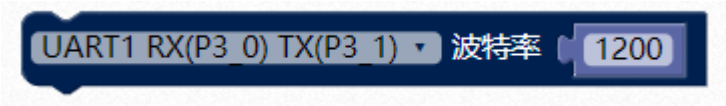



```
#include "lib/UART.h"//引用头文件
uart_init(UART_1, UART1_RX_P30, UART1_TX_P31, 9600, TIM_1);//初始化串口
```



引脚选择列表

波特率尽可能选用常用的 1200、2400、4800、9600、19200、38400、57600、115200，和外部晶振有关，特殊波特率如果不能被整除，会导致波特率不准。图形化模块下拉框没有需要的波特率，可以自己添加数字模块后修改。



图形化模块默认初始化串口为模式 1 可变波特率 8 位数据方式。
串口 1，波特率采用定时器 1 来控制；
串口 2，波特率采用定时器 2 来控制；
串口 3，波特率采用定时器 3 来控制；
串口 4，波特率采用定时器 4 来控制。
使用的时候要注意不要和定时器冲突了。

2. 串口发送数据。


```
uart_putchar(UART_1, 0x31);//串口单个字符输出
```

3. 串口发送指定长度的数组数据。


```
uart_putbuff(UART_1, buff, 12);//数组输出
```

4. 串口发送字符串。


```
uart_putstr(UART_1, "haohaodada");//字符串输出
```

5. 读串口接收/发送中断请求标志位。


```
UART1_GET_RX_FLAG
```

UART.H 文件里有如下宏定义：

```
#define UART1_GET_RX_FLAG   (SCON  & 0x01)
#define UART2_GET_RX_FLAG   (S2CON & 0x01)
#define UART3_GET_RX_FLAG   (S3CON & 0x01)
#define UART4_GET_RX_FLAG   (S4CON & 0x01)

#define UART1_GET_TX_FLAG   (SCON  & 0x02)
#define UART2_GET_TX_FLAG   (S2CON & 0x02)
#define UART3_GET_TX_FLAG   (S3CON & 0x02)
#define UART4_GET_TX_FLAG   (S4CON & 0x02)
```

RI：串口 1 接收中断请求标志位。
TI：串口 1 发送中断请求标志位。

6. 清除串口接收/发送中断请求标志位。


```
UART1_CLEAR_RX_FLAG
```

UART.H 文件里有如下宏定义：

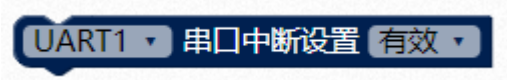
```
#define UART1_CLEAR_RX_FLAG (SCON  &= ~0x01)
#define UART2_CLEAR_RX_FLAG (S2CON &= ~0x01)
#define UART3_CLEAR_RX_FLAG (S3CON &= ~0x01)
```

```
#define UART4_CLEAR_RX_FLAG (S4CON &= ~0x01)

#define UART1_CLEAR_TX_FLAG (SCON  &= ~0x02)
#define UART2_CLEAR_TX_FLAG (S2CON &= ~0x02)
#define UART3_CLEAR_TX_FLAG (S3CON &= ~0x02)
#define UART4_CLEAR_TX_FLAG (S4CON &= ~0x02)
```

7. 获取串口接送缓存数据。


```
SBUF
```

8. 设置串口中断，同时打开总中断。



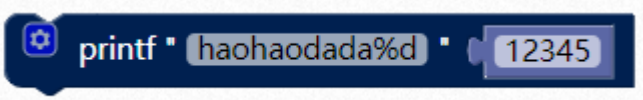

```
EA = 1;
ES = 1;
```

9. 串口接收中断函数。

串口 1 中断号为 4;
串口 2 中断号为 8;
串口 3 中断号为 17;
串口 4 中断号为 18;


```
void UART_R(void) interrupt 4 using 1{

}
```

10. 串口格式化打印输出。



如果要输出多个参数，可以点击蓝色齿轮，然后把左边的项目拖到列表里，就能多一个输入框，自己再拖入一个数字模块。

```
printf_small("haohaodada%d", 12345, );
```

因为 51 资源有限，使用的是剪裁版本的 printf，功能不全，输出数据长度有限。

printf 基本用法：

printf("Hello World!\n"); // \n 表示换行

printf("%d\n", i); /*%d 是输出控制符，d 表示十进制，后面的 i 是输出参数*/.

如果是%x 就是以十六进制的形式输出，要是%o 就是以八进制的形式输出

printf("i = %d, j = %d\n", i, j); i = 10, j = 3，输出多个数据。

常用的输出控制符主要有以下几个：

控制符	说明
%d	按十进制整型数据的实际长度输出。
%ld	输出长整型数据。
%md	m 为指定的输出字段的宽度。如果数据的位数小于 m，则左端补以空格，若大于 m，则按实际位数输出。
%u	输出无符号整型（unsigned）。输出无符号整型时也可以用 %d，这时是将无符号转换成有符号数，然后输出。但编程的时候最好不要这么写，因为这样要进行一次转换，使 CPU 多做一次无用功。
%c	用来输出一个字符。
%f	用来输出实数，包括单精度和双精度，以小数形式输出。不指定字段宽度，由系统自动指定，整数部分全部输出，小数部分输出 6 位，超过 6 位的四舍五入。
%.mf	输出实数时小数点后保留 m 位，注意 m 前面有个点。
%o	以八进制整数形式输出，这个就用得很少了，了解一下就行了。
%s	用来输出字符串。用 %s 输出字符串同前面直接输出字符串是一样的。但是此时要先定义字符数组或字符指针存储或指向字符串，这个稍后再讲。
%x（或 %X 或 %#x 或 %#X）	以十六进制形式输出整数，这个很重要。

更多用法，请查看 C 语言相关知识。

示例 1:

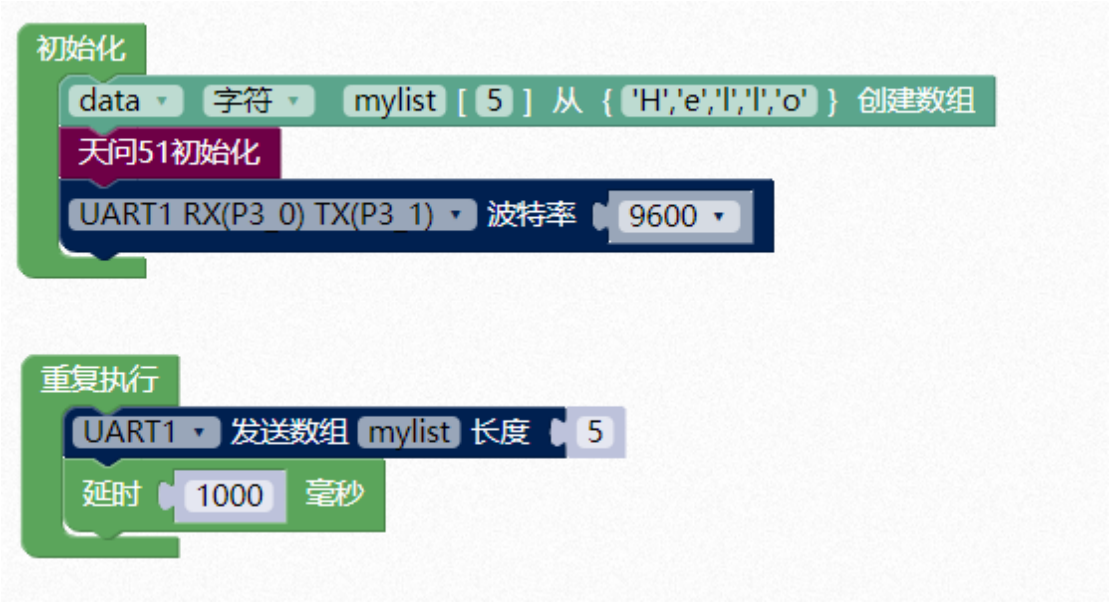
串口 1 定时发送字符'1',对应的 ASSIC 码为 0x31。


```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/UART.h"

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
```

```
{
  twen_board_init();//天问 51 初始化
  uart_init(UART_1, UART1_RX_P30, UART1_TX_P31, 9600, TIM_1);//初始化串
口
}

void loop()
{
  uart_putchar(UART_1, 0x31);//串口单个字符输出
  delay(1000);
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

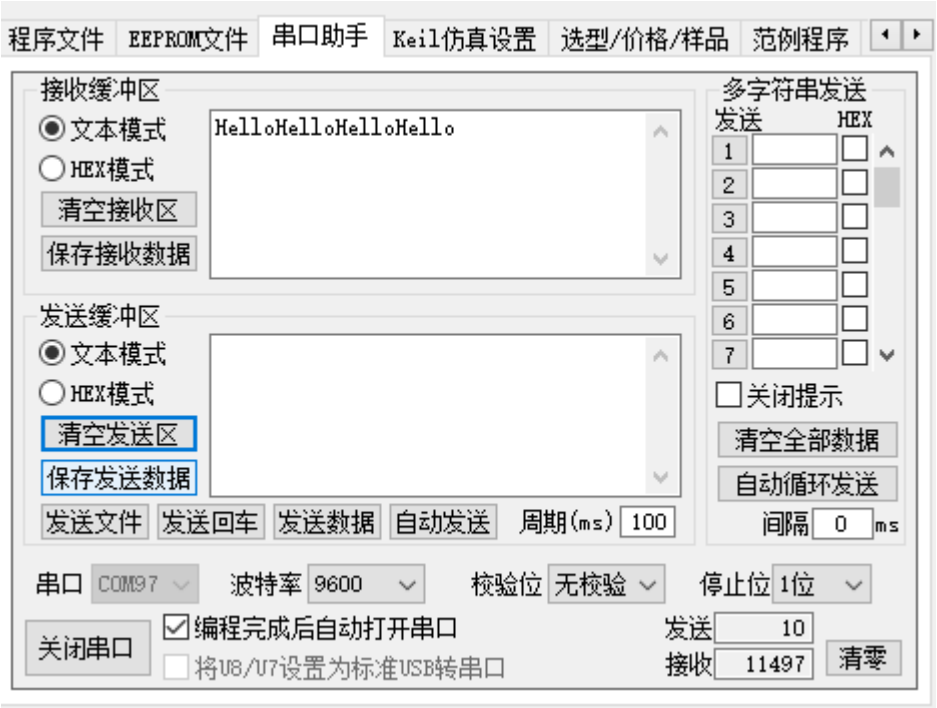
示例 2:

串口 1 定时发送数组，打开串口工具，可以看到输出字符“Hello”。




```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
```

```
#include "lib/delay.h"
#include "lib/UART.h"

char mylist[5]={'H','e','l','l','o'};
void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  uart_init(UART_1, UART1_RX_P30, UART1_TX_P31, 9600, TIM_1);//初始化串
口
}

void loop()
{
  uart_putbuff(UART_1, mylist, 5);//数组输出
  delay(1000);
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

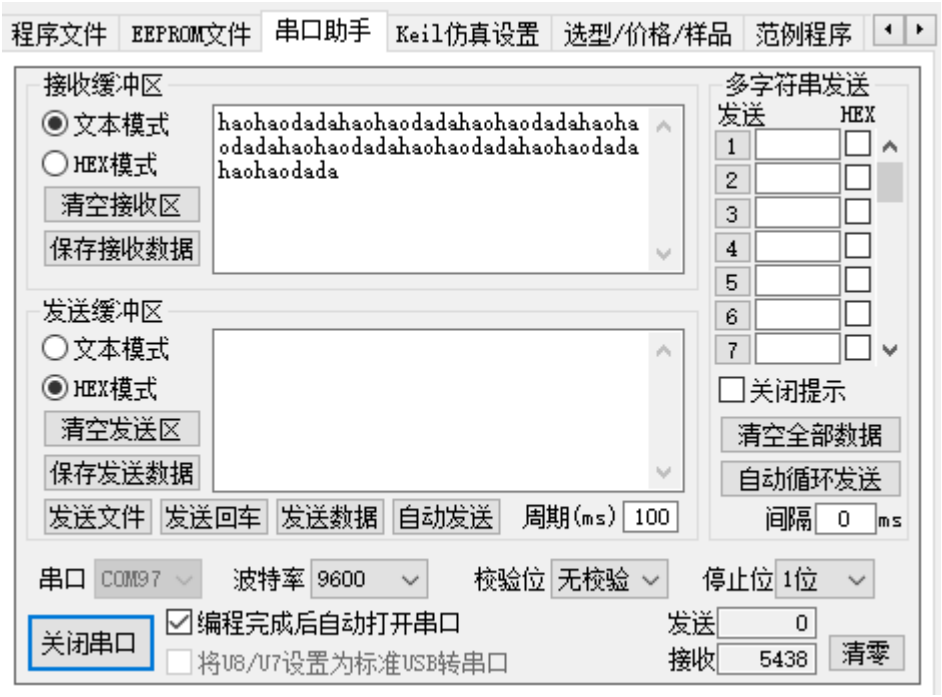


示例 3:

串口 1 定时发送字符串，打开串口工具，可以看到输出字符“haohaodada”。

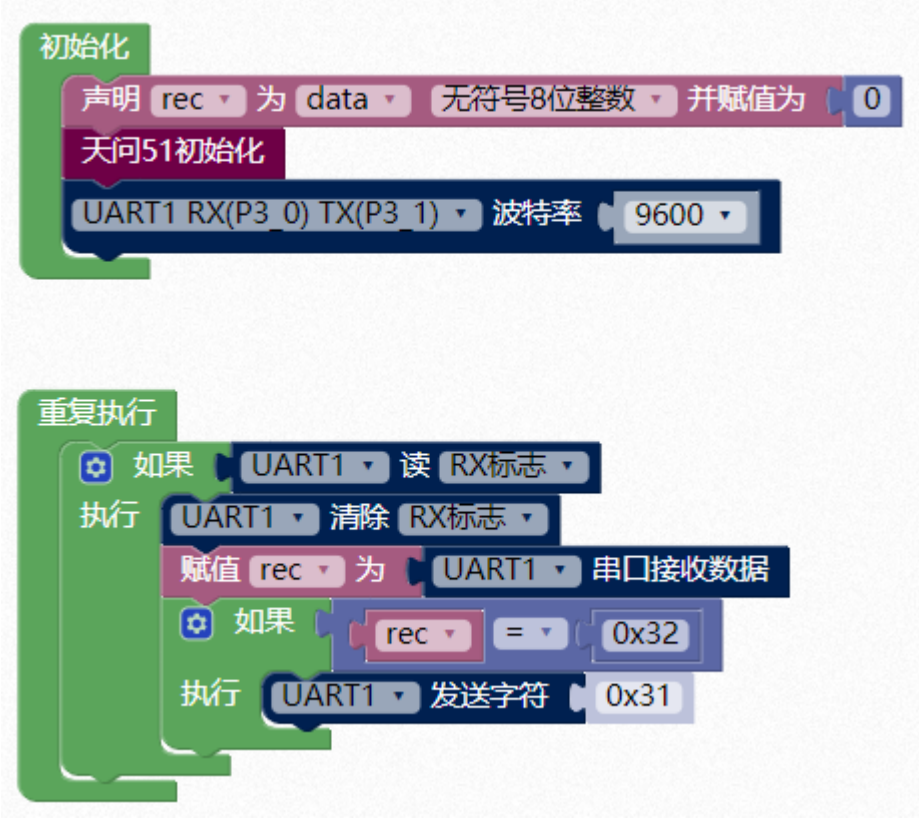

```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/UART.h"

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
```

```
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  uart_init(UART_1, UART1_RX_P30, UART1_TX_P31, 9600, TIM_1);//初始化串
口
}

void loop()
{
  uart_putstr(UART_1, "haohaodada");//字符串输出
  delay(1000);
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```



示例 4:

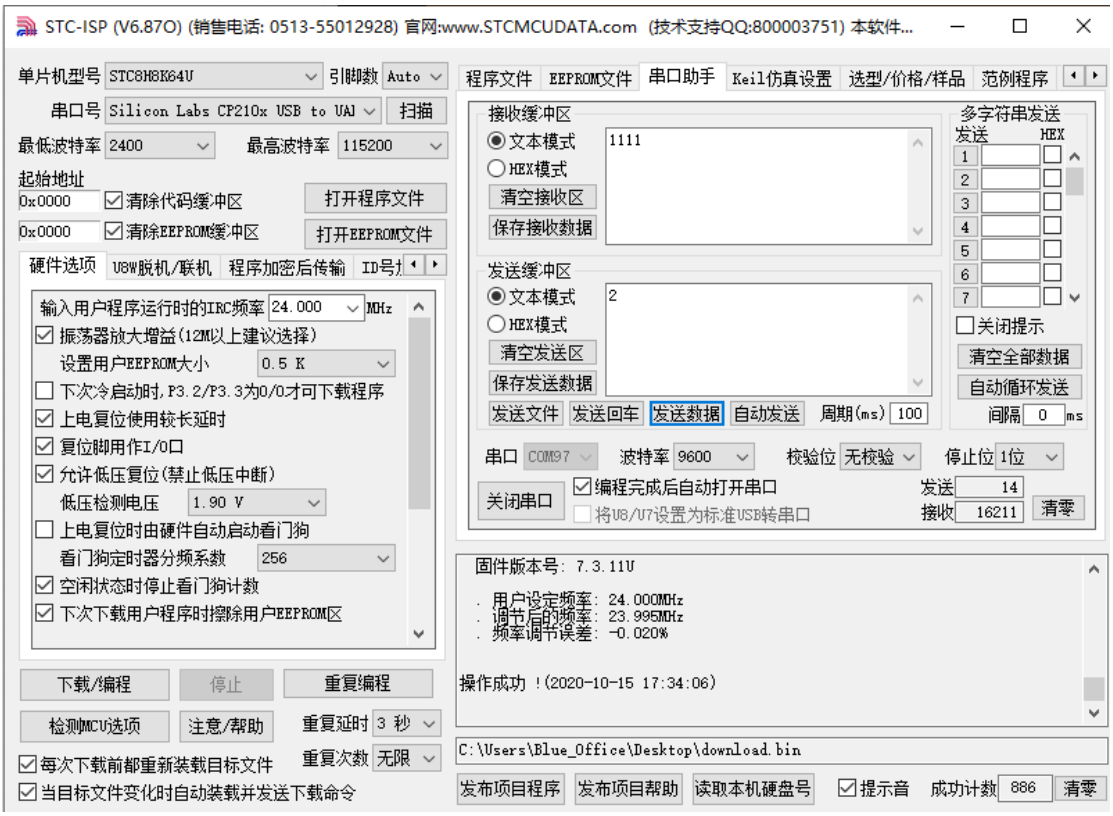
串口 1 采用查询方式，读取接收数据，这种方式，当循环里任务多的时候，容易丢失数据。打开串口工具，我们发送文本 2，会自动回复文本 1。




```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
```

```
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/UART.h"

uint8 rec = 0;

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  uart_init(UART_1, UART1_RX_P30, UART1_TX_P31, 9600, TIM_1);//初始化串
口
}

void loop()
{
  if(UART1_GET_RX_FLAG){
    UART1_CLEAR_RX_FLAG;
    rec = SBUF;
    if(rec == 0x32){
      uart_putchar(UART_1, 0x31);//串口单个字符输出
    }
  }
}

void main(void)
```

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

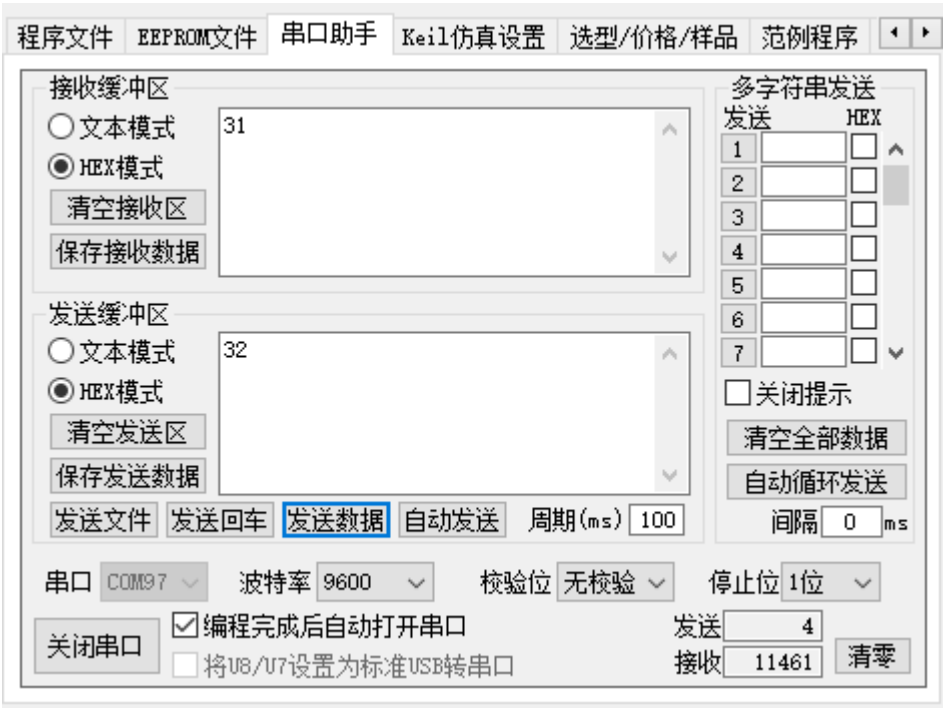


示例 5:

串口 1 采用中断方式，读取接受数据。打开串口工具，我们发送十六进制 32，会自动回复十六进制 31。


```c
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/UART.h"

uint8 rec = 0;

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
```

```
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void UART_R(void) interrupt 4 using 1{
  UART1_CLEAR_RX_FLAG;
  rec = SBUF;
  if(rec == 0x32){
    uart_putchar(UART_1, 0x31);//串口单个字符输出
  }
}

void setup()
{
  twen_board_init();//天问 51 初始化
  uart_init(UART_1, UART1_RX_P30, UART1_TX_P31, 9600, TIM_1);//初始化串
口
  EA = 1;
  ES = 1;
}

void loop()
{

}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```



在串口模块里还有更多选项，里面是一些直接读写寄存器的模块，供高级应用。

1. 读串口的 TI/RI/REN 寄存器。


```
REN //UART1
S2CON & 0x10 // UART2,不能位寻址
S3CON & 0x10 // UART3,不能位寻址
S4CON & 0x10 // UART4,不能位寻址
```

REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

```
TI //UART1
S2CON & 0x02 // UART2,不能位寻址
S3CON & 0x02 // UART3,不能位寻址
S4CON & 0x02 // UART4,不能位寻址
```

```
RI //UART1
S2CON & 0x01 // UART2,不能位寻址
S3CON & 0x01 // UART3,不能位寻址
S4CON & 0x01 // UART4,不能位寻址
```

TI：串口 1 发送中断请求标志位。在模式 0 中，当串口发送数据第 8 位结束时，由硬件自动将 TI 置 1，向主机请求中断，响应中断后 TI 必须用软件清零。在其他模式中，则在停止位开始发送时由硬件自动将 TI 置 1，向 CPU 发请求中断，响应中断后 TI 必须用软件清零。

RI：串口 1 接收中断请求标志位。在模式 0 中，当串口接收第 8 位数据结束时，由硬件自动将 RI 置 1，向主机请求中断，响应中断后 RI 必须用软件清零。在其他模式中，串行接收到停止位的中间时刻由硬件自动将 RI 置 1，向 CPU 发中断申请，响应中断后 RI 必须由软件清零。

2. 写串口的 TI/RI/REN 寄存器值。


```
REN = 1; //UART1
S2CON |= 0x10; // UART2,不能位寻址
S3CON |= 0x10; // UART3,不能位寻址
S4CON |= 0x10; // UART4,不能位寻址

TI = 1; //UART1
S2CON |= 0x02; // UART2,不能位寻址
S3CON |= 0x02; // UART3,不能位寻址
S4CON |= 0x02; // UART4,不能位寻址

RI = 1; //UART1
S2CON |= 0x01; // UART2,不能位寻址
S3CON |= 0x01; // UART3,不能位寻址
S4CON |= 0x01; // UART4,不能位寻址
```

3. 读 PCON 电源管理寄存器。


```
PCON
```

电源管理寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

SMOD：串口 1 波特率控制位

0：串口 1 的各个模式的波特率都不加倍

1：串口 1 模式 1、模式 2、模式 3 的波特率加倍

SMOD0：帧错误检测控制位

0：无帧错检测功能

1：使能帧错误检测功能。此时 SCON 的 SM0/FE 为 FE 功能，即为帧错误检测标志位。

4. 写 PCON 电源管理寄存器值。

```
PCON = 0xff;
```

5. 读 SCON 电源管理寄存器。


```
SCON   //UART1
S2CON  //UART2
S3CON  //UART3
S4CON  //UART4
```

串口 1 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE：当PCON寄存器中的SMOD0位为1时，该位为帧错误检测标志位。当UART在接收过程中检测到一个无效停止位时，通过UART接收器将该位置1，必须由软件清零。当PCON寄存器中的SMOD0位为0时，该位和SM1一起指定串口1的通信工作模式，如下表所示：

SM0	SM1	串口1工作模式	功能说明
0	0	模式0	同步移位串行方式
0	1	模式1	可变波特率8位数据方式
1	0	模式2	固定波特率9位数据方式
1	1	模式3	可变波特率9位数据方式

SM2：允许模式 2 或模式 3 多机通信控制位。当串口 1 使用模式 2 或模式 3 时，如果 SM2 位为 1 且 REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 RB8）来筛选地址帧，若 RB8=1，说明该帧是地址帧，地址信息可以进入 SBUF，并使 RI 为 1，进而在中断服务程序中再进行地址号比较；若 RB8=0，说明该帧不是地址帧，应丢掉且保持 RI=0。在模式 2 或模式 3 中，如果 SM2 位为 0 且 REN 位为 1，接收收机处于地址帧筛选被禁止状态，不论收到的 RB8 为 0 或 1，均可使接收到的信息进入 SBUF，并使 RI=1，此时 RB8 通常为校验位。模式 1 和模式 0 为非多机通信方式，在这两种方式时，SM2 应设置为 0。

REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

TB8：当串口 1 使用模式 2 或模式 3 时，TB8 为要发送的第 9 位数据，按需要由软件置位或清 0。在模式 0 和模式 1 中，该位不用。

RB8：当串口 1 使用模式 2 或模式 3 时，RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 和模式 1 中，该位不用。

TI：串口 1 发送中断请求标志位。在模式 0 中，当串口发送数据第 8 位结束时，由硬件自动将 TI 置 1，向主机请求中断，响应中断后 TI 必须用软件清零。在其他模式中，则在停止位开始发送时由硬件自动将 TI 置 1，向 CPU 发请求中断，响应中断后 TI 必须用软件清零。

RI：串口 1 接收中断请求标志位。在模式 0 中，当串口接收第 8 位数据结束时，由硬件自动将 RI 置 1，向主机请求中断，响应中断后 RI 必须用软件清零。在其他模式中，串行接收到停止位的中间时刻由硬件自动将 RI 置 1，向 CPU 发中断申请，响应中断后 RI 必须由软件清零。

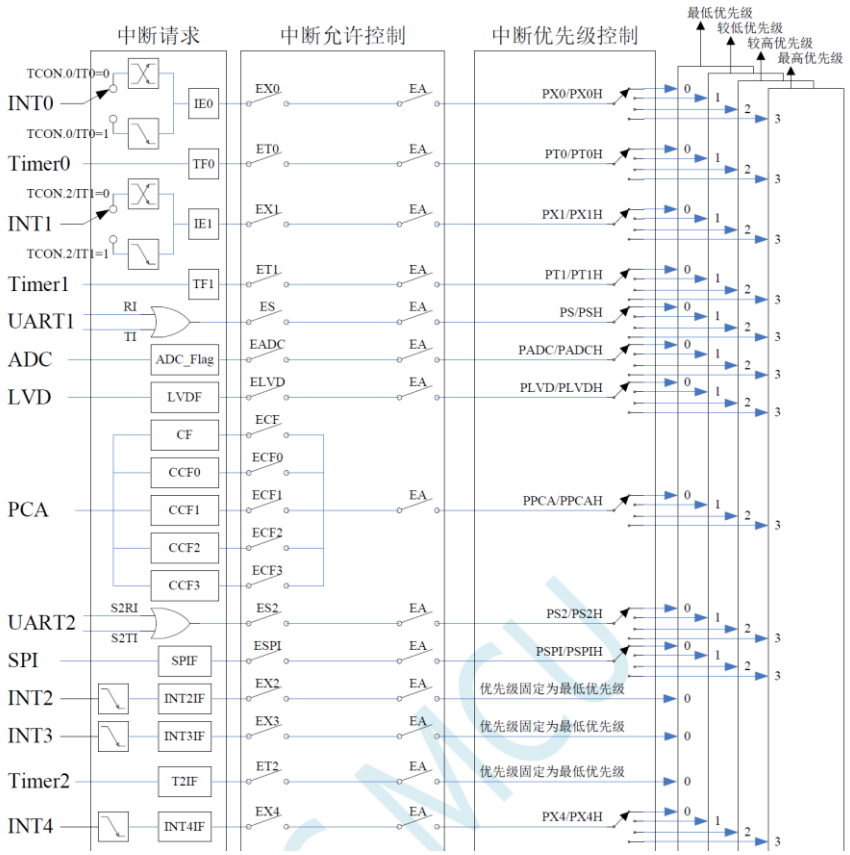
6. 写 SCON 电源管理寄存器值。


```
SCON = 0xff;  //UART1
S2CON = 0xff; //UART2
S3CON = 0xff; //UART3
```

```
S4CON = 0xff; //UART4
```

外部中断模块

STC8H 外部中断有 5 个 INT0 到 INT4，天问 51 开发板上的 INT0 为 P32 连接到了独立按键 KEY1，INT1 为 P33 连接到了独立按键 KEY2，INT2 为 P36 连接到了红外接收引脚，INT3 为 P37 连接到了加速度传感器的中断引脚，INT4 为 P30 连接到了 USB 接口的 D-。



通过手册提供的中断结构图，我们可以看到配置 INT0，先要设置 IT0 寄存器，IT0=0 为引脚上的电平从低电平变为高电平，或者从高电平变为低电平时，即电平变化就会触发中断；IT0=1 为引脚上的电平从高电平变为低电平时，即下降沿就会触发中断。然后设置 EX0 允许中断，和 EA 总中断控制。中断优先级我们暂时不设置，默认为最低优先级。

1. 设置外部中断。

这里需要注意，只有外部中断 INT0、INT1 有电平变化和下降沿两种状态，INT2、INT3、INT4 只有下降沿。

```
IT0 = 0;
EX0 = 1;
EA = 1;
```

IE（中断使能寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：总中断允许控制位。EA 的作用是使中断允许形成多级控制。即各中断源首先受 EA 控制;其次还受各中断源自己的中断允许控制位控制。

0：CPU 屏蔽所有的中断申请

1：CPU 开放中断

ELVD：低压检测中断允许位。

0：禁止低压检测中断

1：允许低压检测中断

EADC：A/D 转换中断允许位。

0：禁止 A/D 转换中断

1：允许 A/D 转换中断

ES：串行口 1 中断允许位。

0：禁止串行口 1 中断

1：允许串行口 1 中断

ET1：定时/计数器 T1 的溢出中断允许位。

0：禁止 T1 中断

1：允许 T1 中断

EX1：外部中断 1 中断允许位。

0：禁止 INT1 中断

1：允许 INT1 中断

ET0：定时/计数器 T0 的溢出中断允许位。

0：禁止 T0 中断

1：允许 T0 中断

EX0：外部中断 0 中断允许位。

0：禁止 INT0 中断

1：允许 INT0 中断

INTCLKO（外部中断与时钟输出控制寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

EX4：外部中断 4 中断允许位。

0：禁止 INT4 中断

1：允许 INT4 中断

EX3：外部中断 3 中断允许位。

0：禁止 INT3 中断

1：允许 INT3 中断

EX2：外部中断 2 中断允许位。

0：禁止 INT2 中断

1：允许 INT2 中断

13.2 定时器 0/1

定时器 0/1 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1：T1溢出中断标志。T1被允许计数以后，从初值开始加1计数。当产生溢出时由硬件将TF1位置“1”，并向CPU请求中断，一直保持到CPU响应中断时，才由硬件清“0”（也可由查询软件清“0”）。

TR1：定时器T1的运行控制位。该位由软件置位和清零。当GATE（TMOD.7）=0，TR1=1时就允许T1开始计数，TR1=0时禁止T1计数。当GATE（TMOD.7）=1，TR1=1且INT1输入高电平时，才允许T1计数。

TF0：T0溢出中断标志。T0被允许计数以后，从初值开始加1计数，当产生溢出时，由硬件置“1”TF0，向CPU请求中断，一直保持CPU响应该中断时，才由硬件清0（也可由查询软件清0）。

TR0：定时器T0的运行控制位。该位由软件置位和清零。当GATE（TMOD.3）=0，TR0=1时就允许T0开始计数，TR0=0时禁止T0计数。当GATE（TMOD.3）=1，TR0=1且INT0输入高电平时，才允许T0计数，TR0=0时禁止T0计数。

IE1：外部中断1请求源（INT1/P3.3）标志。IE1=1，外部中断向CPU请求中断，当CPU响应该中断时由硬件清“0”IE1。

IT1：外部中断源1触发控制位。IT1=0，上升沿或下降沿均可触发外部中断1。IT1=1，外部中断1程控为下降沿触发方式。

IE0：外部中断0请求源（INT0/P3.2）标志。IE0=1外部中断0向CPU请求中断，当CPU响应外部中断时，由硬件清“0”IE0（边沿触发方式）。

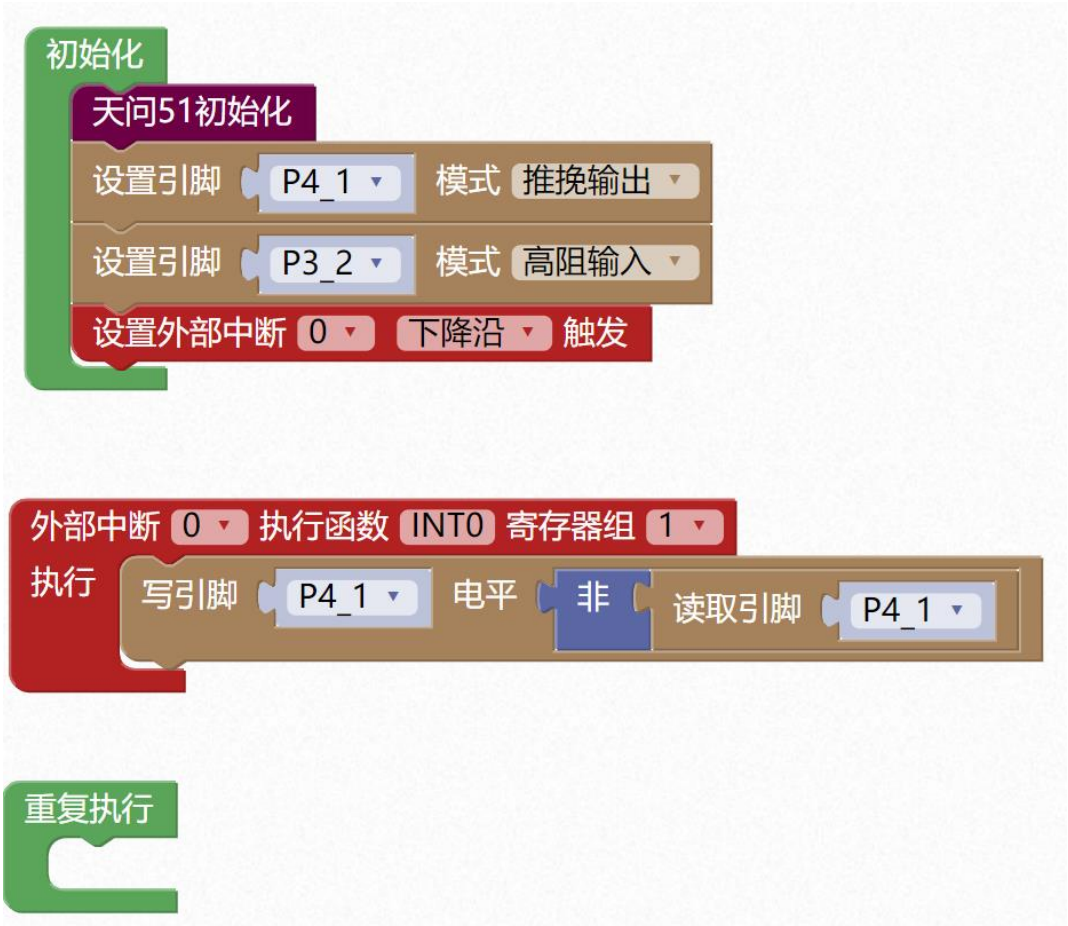
IT0：外部中断源0触发控制位。IT0=0，上升沿或下降沿均可触发外部中断0。IT0=1，外部中断0程控为下降沿触发方式。

2. 外部中断函数。


```
void INT0(void) interrupt 0 using 1{
} //

void INT1(void) interrupt 2 using 1{
}

void INT2(void) interrupt 10 using 1{
}

void INT3(void) interrupt 11 using 1{
}

void INT4(void) interrupt 16 using 1{
}
```

示例 1:

设置按键 KEY1 下降沿中断，控制 P41 LED 灯翻转。




```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
```

```
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void INT0(void) interrupt 0 using 1{
  P4_1 = !P4_1;
}

void setup()
{
  twen_board_init();//天问 51 初始化
  P4M1&=~0x02;P4M0|=0x02;//推挽输出
  P3M1|=0x04;P3M0&=~0x04;//高阻输入
  IT0 = 1;
  EX0 = 1;
  EA = 1;
}

void loop()
{

}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

所有中断设置模块

这部分使用频率比较低，相比于前面单独的外部中断、定时器中断、串口中断，还多了 USB 中断，提供给高级用户使用。

- ✓ 总中断
- 定时器0
- 定时器1
- 定时器2
- 定时器3
- 定时器4
- 串口中断1
- 串口中断2
- 串口中断3
- 串口中断4
- 外部中断0
- 外部中断1
- 外部中断2
- 外部中断3
- 外部中断4
- USB中断

1. 设置对应中断状态

2. 读对应中断状态

3. 对应的中断函数

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

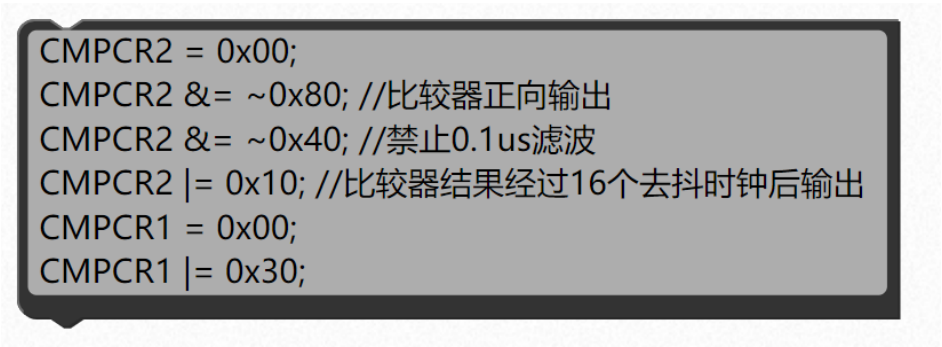
等效于

```
#define MYDEFINE P0_0
```

3. 嵌入代码

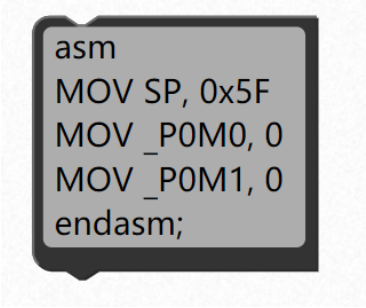
示例 1:

嵌入 C 语言代码。



示例 2:

嵌入汇编语言代码，开头需要 asm 关键字，结尾需要 endasm。



程序模块

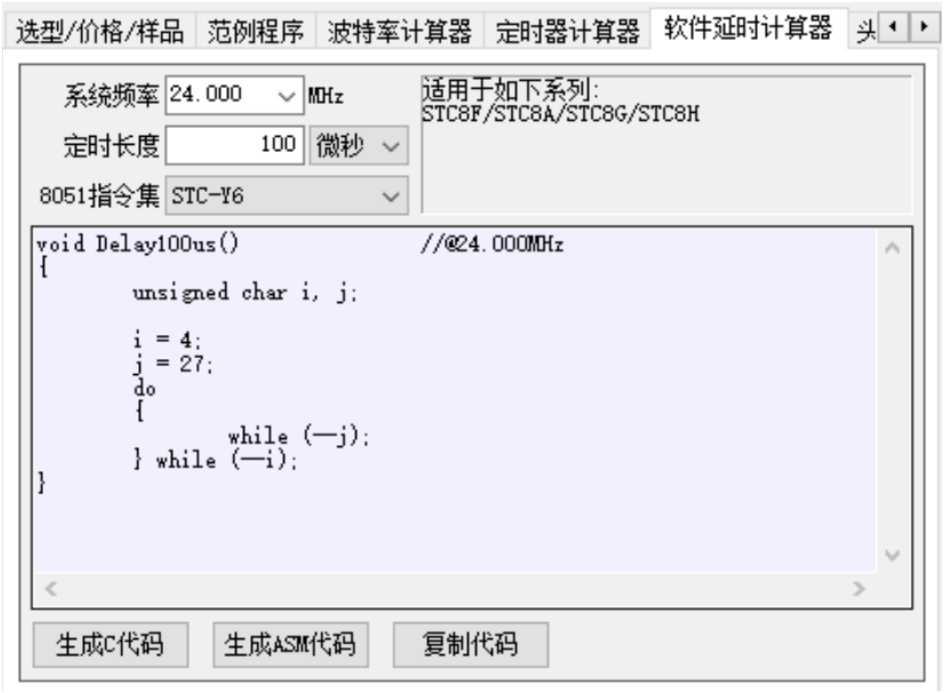
程序模块主要是和 C 语言相关的模块。

控制模块

1. 延时 1/5/10/50/100 微秒。
   图形化模块提供了常用几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。

✓ 1
5
10
50
100

```
delay1us();
delay5us();
delay10us();
delay50us();
delay100us();
```



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。


```
#include "lib/delay.h"
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.
```

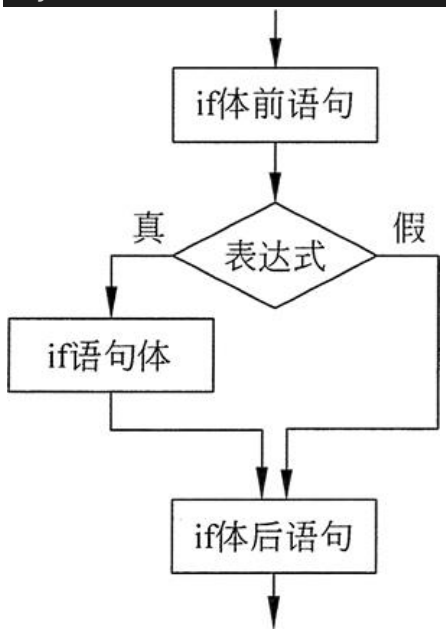
```
// 参数: 延迟时间（0-65535）.
// 返回: none.
//========================================================================
void delay(uint16 time)
{
  do { delay1ms();} while (--time);
}
```

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。

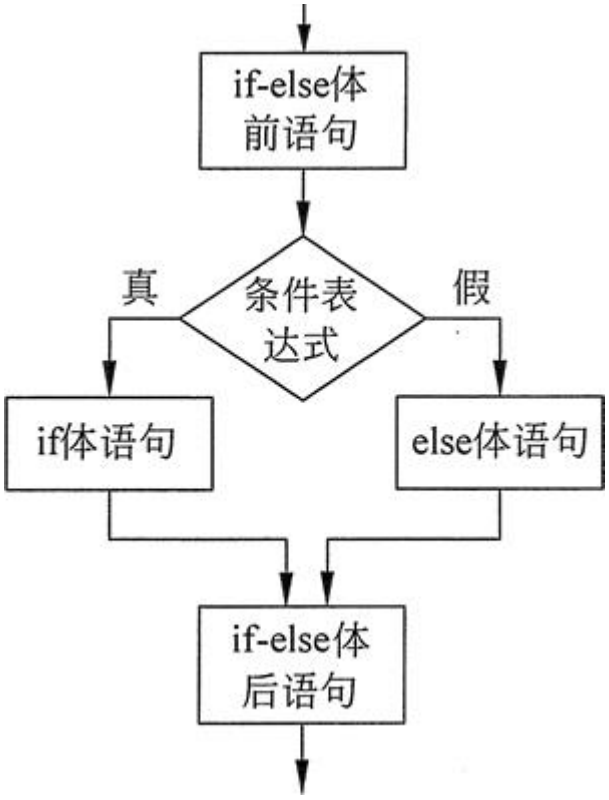

```
if(0){

  }
```



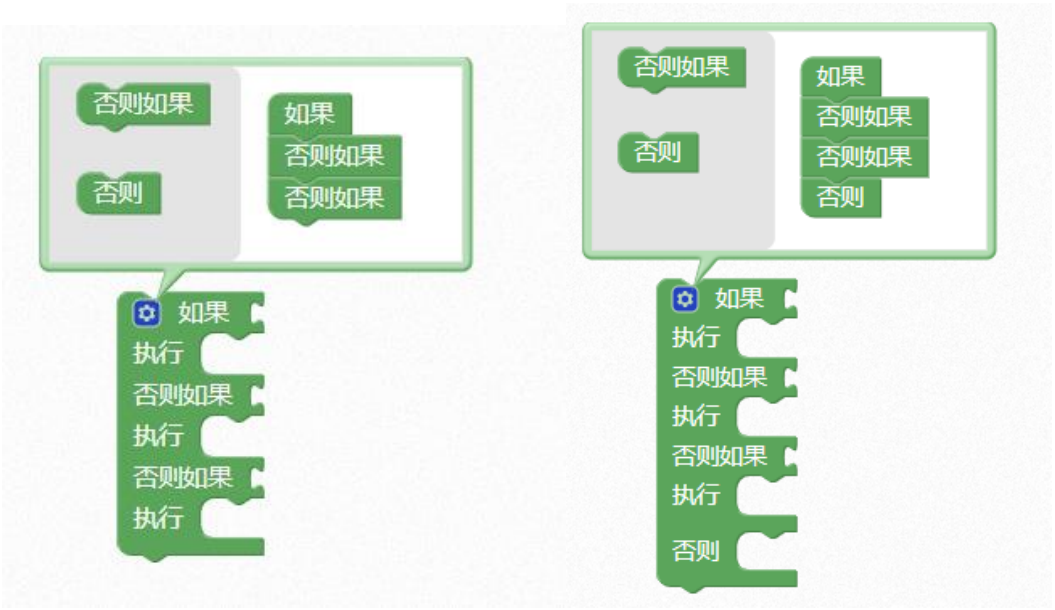
还可以通过点击蓝色小齿轮，添加多个判断语句。
如下为如果否则判断分支语句:

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。

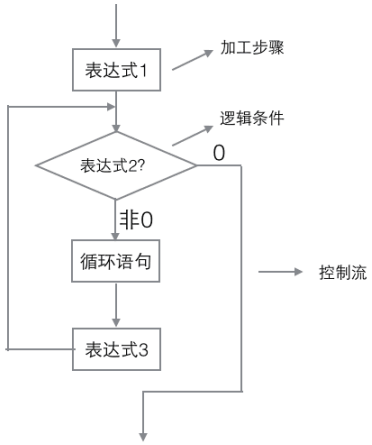

```
uint8 i;

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。

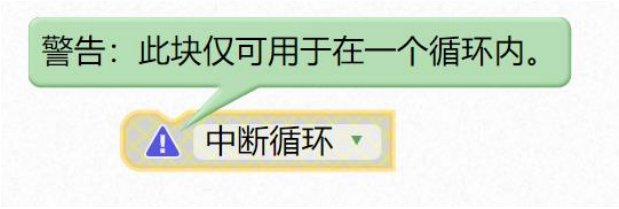
for (表达式 1;表达式 2;表达式 3)
{
  循环语句
}
表达式 1 给循环变量赋初值

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：

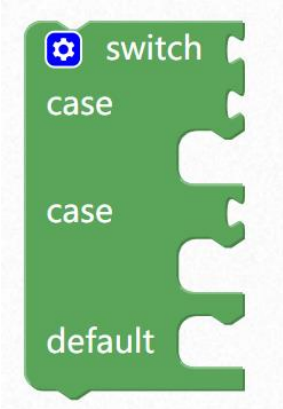


如果循环中需要中断循环，可以用中断循环模块。



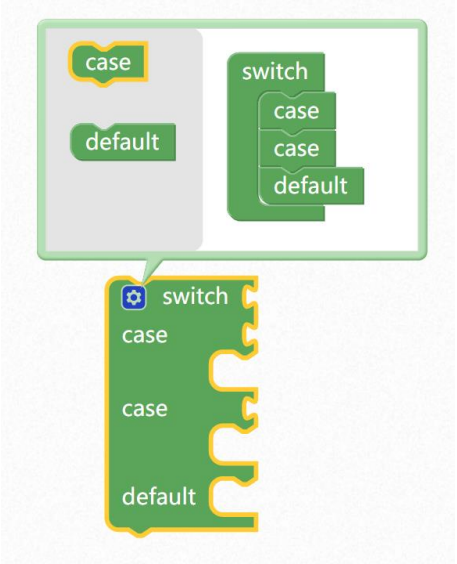

```
break;
```

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。

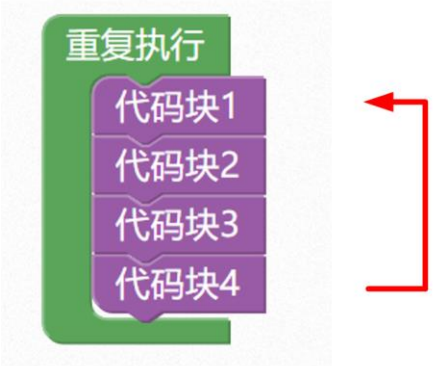
里面的代码只在上电后执行一次，因此我们通常把一些变量的声明或引脚初始化等放在初始化里。

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。

✓ &

|

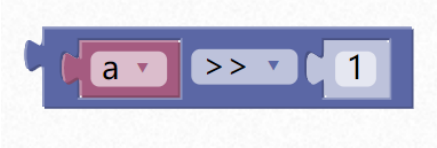
^

>>

<<

示例 1:

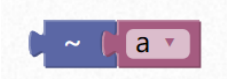
变量 a>>1 右移 1 位。



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。

✓ =

≠

<

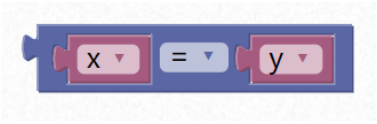
≤

>

≥

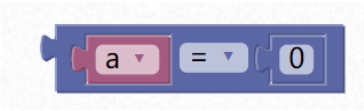
示例 1:

比较变量 x 和 y 是否相等。



示例 2:

比较变量 x 是否等于 0。

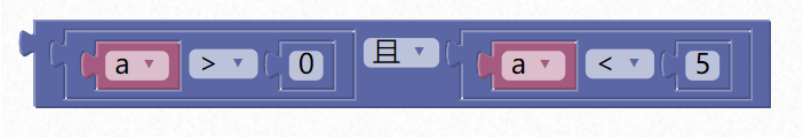


6. 逻辑比较

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。

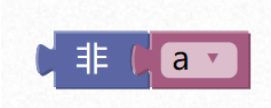



```
(a>0) && (a<5)
```

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


```
#include "lib/wmath.h" //引用头文件
//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。


```
#include "lib/wmath.h"//引用头文件
round(x);//四舍五入
```

返回四舍五入后的数。

向上取整

```
#include <math.h> //引用头文件
ceilf(float x);
```

示例 1:

ceilf(-3.14) = -3;

ceilf(4.56) = 5;

向下取整。

```
#include <math.h> //引用头文件
floorf(float x);
```

示例 1:

floorf(-3.14) = -4;

floorf(4.56) = 4;

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

```
#include <math.h> //引用头文件
sqrtf(float a);//平方根
fabsf(float x);//绝对值
- // 负数
logf(float x);//ln
log10f(float x);//log10
expf(float x);//e^
powf(float x, float y);//10^
sinf(float x);//sin
cosf(float x);//cos
tanf(float x);//tan
asinf(float x);//asin
acosf(float x);//acos
atanf(float x);//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。


```
#include "lib/wmath.h"//引用头文件
```

```
map(long x, long in_min, long in_max, long out_min, long out_max)
```

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:

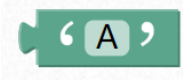
定义"hello world"字符串


```
"hello world"
```

2. 字符

示例 1:

定义'A'字符



3. 连接文本

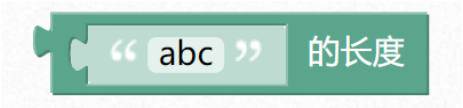
示例 1:


```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/oled.h"
#include <string.h>

char dest[50] = "hello";

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}
```

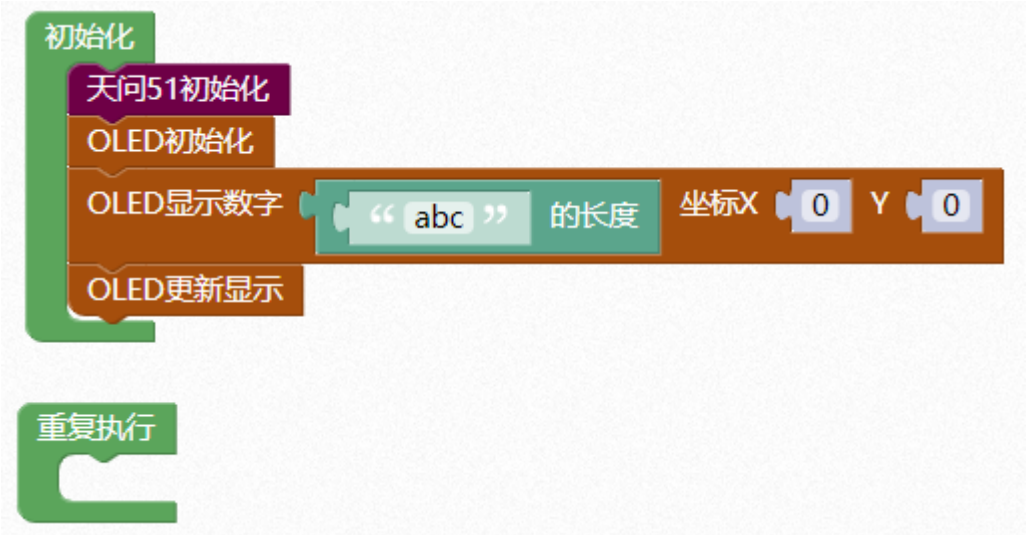
```
void setup()
{
  twen_board_init();//天问 51 初始化
  oled_init();//OLED 初始化
  oled_show_string(0,0,(strcat(dest, "world")));
  oled_display();//OLED 更新显示
}

void loop()
{

}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

OLED 显示 hello world。

4. 获取字符串长度



示例 1:




```
#include <STC8HX.h>
```

```
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/oled.h"
#include <string.h>

char str2[] = "abc";

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  oled_init();//OLED 初始化
  oled_show_num(0,0,(strlen(str2)));
  oled_display();//OLED 更新显示
}

void loop()
{

}

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

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

OLED 显示 3

5. 文本转整数

示例 1:

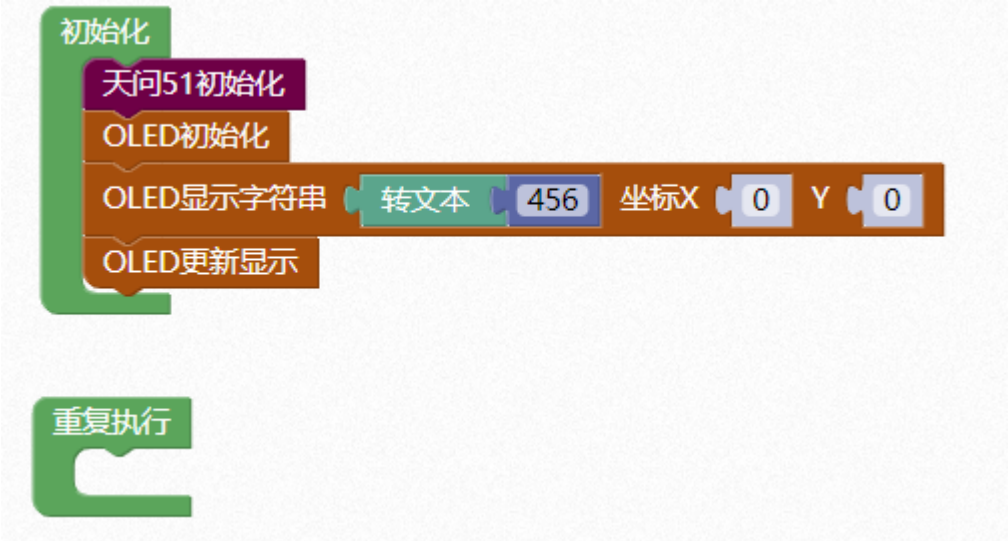

```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/oled.h"
#include <stdlib.h>

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
```

```
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  oled_init();//OLED 初始化
  oled_show_num(0,0,(atoi("123")));
  oled_display();//OLED 更新显示
}

void loop()
{

}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

OLED 显示 123

6. 转文本

示例 1:

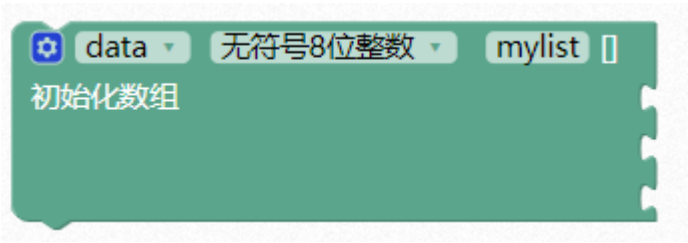



```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/oled.h"
#include <stdlib.h>

char str2[10];

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
```

```
{
  _itoa(456, str2, 10);
  twen_board_init();//天问 51 初始化
  oled_init();//OLED 初始化
  oled_show_string(0,0,str2);
  oled_display();//OLED 更新显示
}

void loop()
{

}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

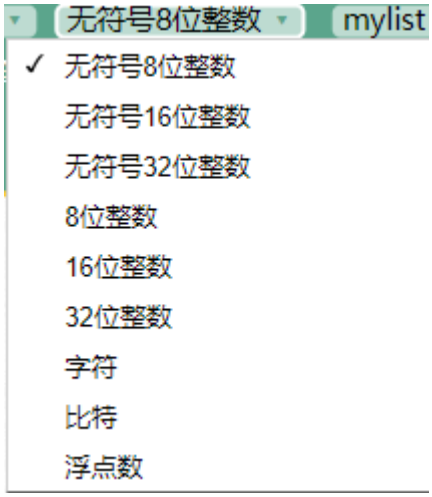
OLED 显示 456。

7. 创建数组方式 1

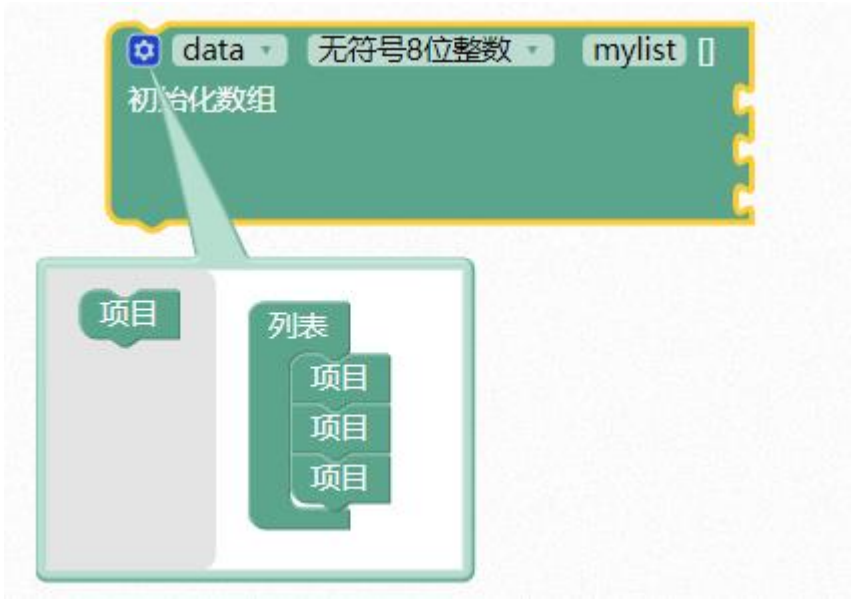


数组存储位置可以选择，默认为 data。

数组类型根据情况选择。

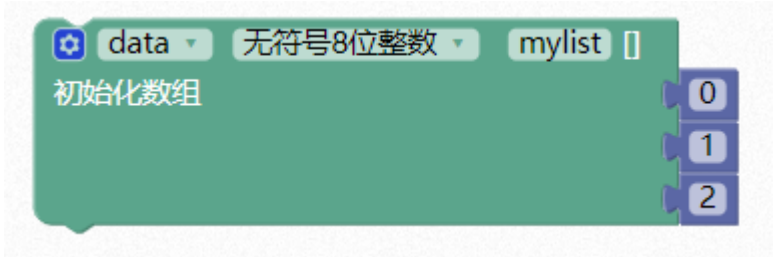


数组初始数据可以通过蓝色小齿轮增加



示例 1:

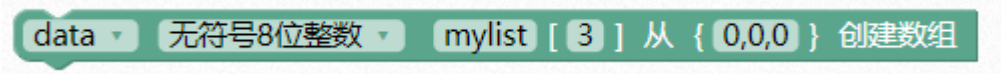
定义一个 8 位数组，数组名为 mylist，数组初始内容为 0，1，2




```
uint8 mylist[]={0, 1, 2};
```

8. 创建数组方式 2

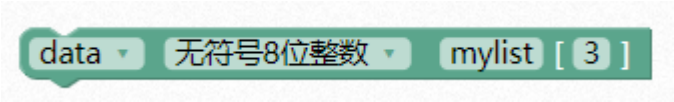
前面一种方式，拖动起来比较麻烦，可以用这种方式



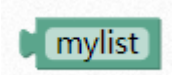
示例 1:


```
uint8 mylist[3]={0, 1, 2};
```

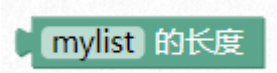
9. 创建数组方式 3
   只定义长度，内容为空。



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

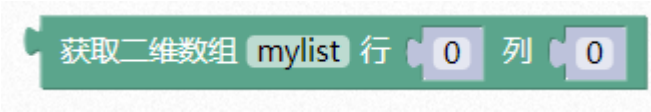
13. 给数组指定项目赋值

14. 创建二维数组方式 1

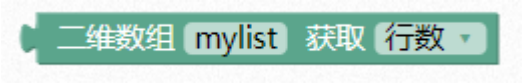
15. 创建二维数组方式 2

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



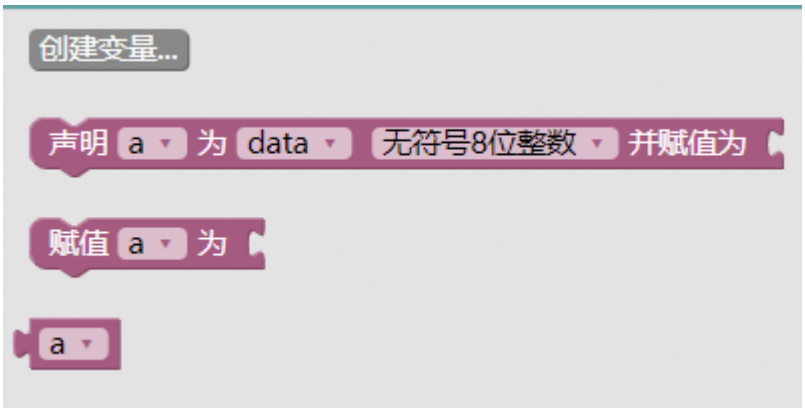
变量模块

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。

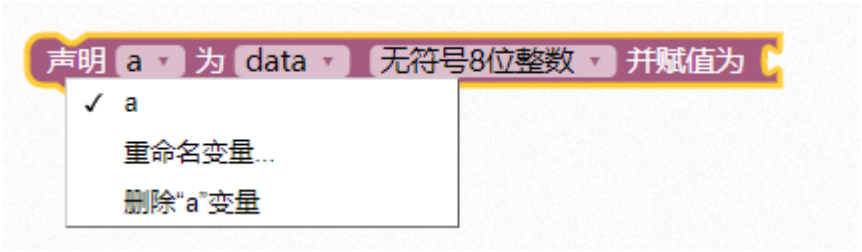
变量栏目默认没有变量，需要点击灰色按钮创建。

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

第一个选型里，可以再次重命名或者删除变量。



可以设置变量存放的 RAM 区，默认为 data。

✓ data

code

xdata

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

8位整数

16位整数

32位整数

字符

比特

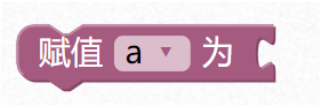
浮点数

字符串

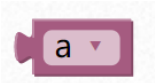
```
typedef unsigned char   uint8;  //  8 bits
typedef unsigned int    uint16; // 16 bits
```

```
typedef unsigned long   uint32; // 32 bits
typedef signed char     int8; //  8 bits
typedef signed int      int16;  // 16 bits
typedef signed long     int32;  // 32 bits
```

3. 变量赋值

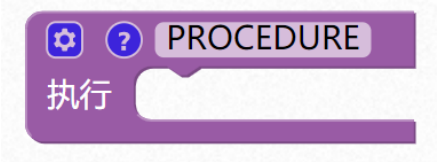


4. 获取变量值

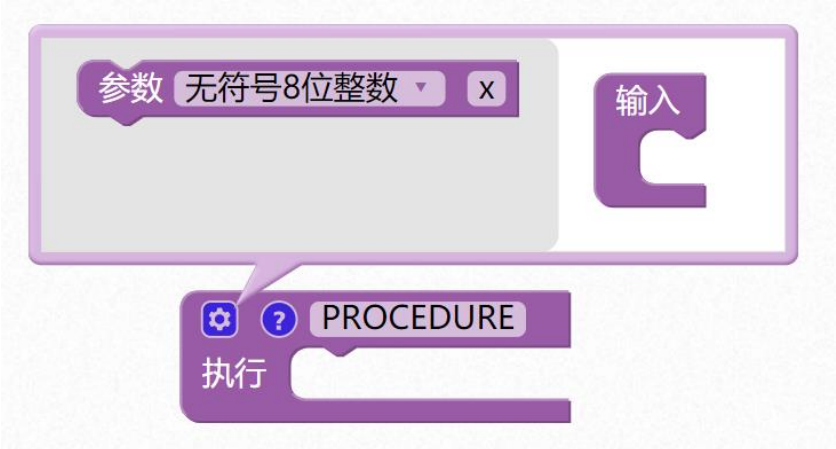


函数模块

1. 定义无返回值函数

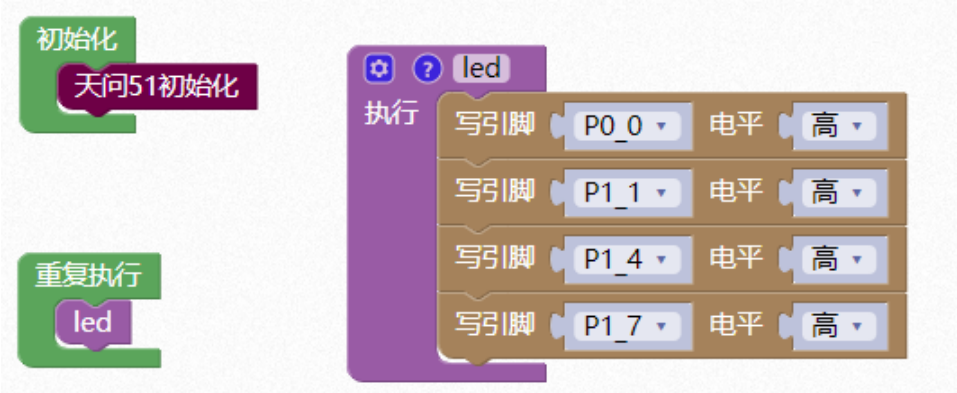


点击蓝色小齿轮，可以添加输入参数，函数名可以自己命名，建议不要用中文。

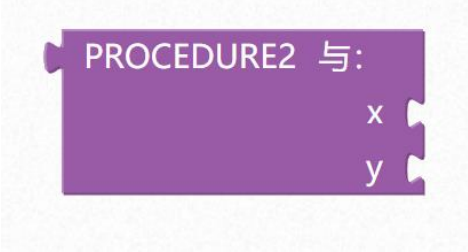


函数块使用好了后，在函数栏目里会自动出现对应的执行函数块。

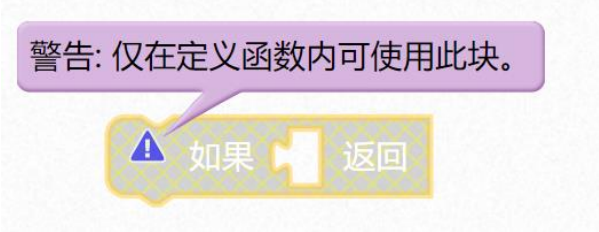
示例 1:
把多个 LED 操作归纳到 led 函数里，让主程序可读性增强。



2. 定义有返回值函数
   如需要输入参数，操作和上一节一样。

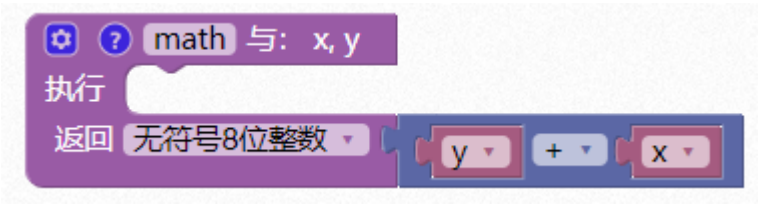


如果需要在函数执行过程中间范围，可以添加如果返回模块。



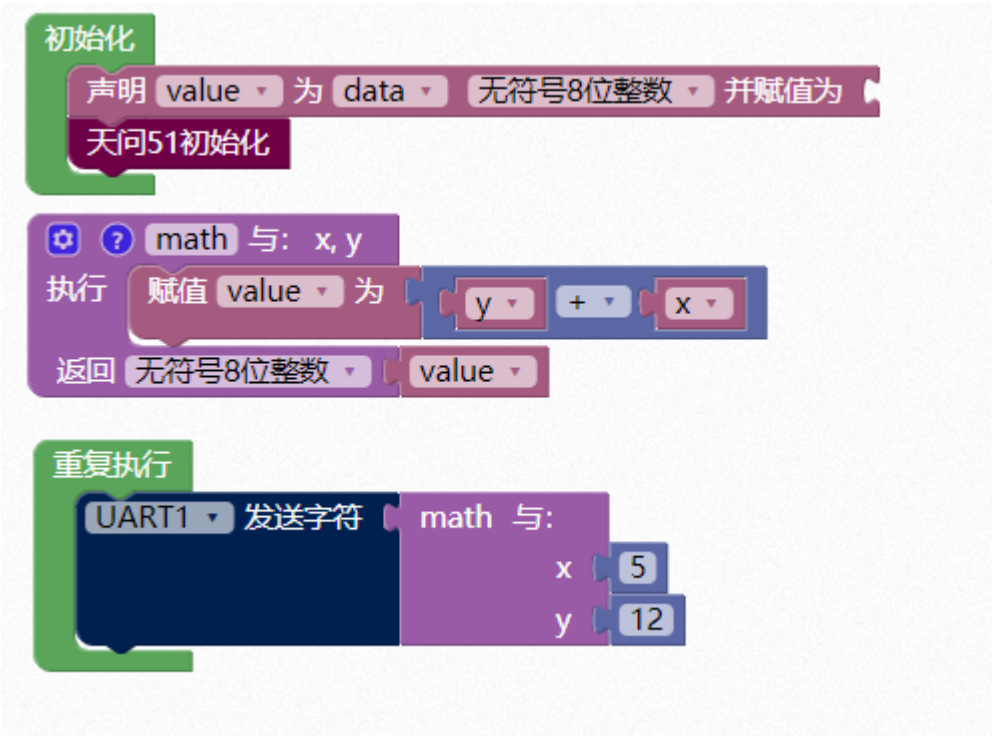
示例 1:

定义数学函数，返回 x+y 运算结果值



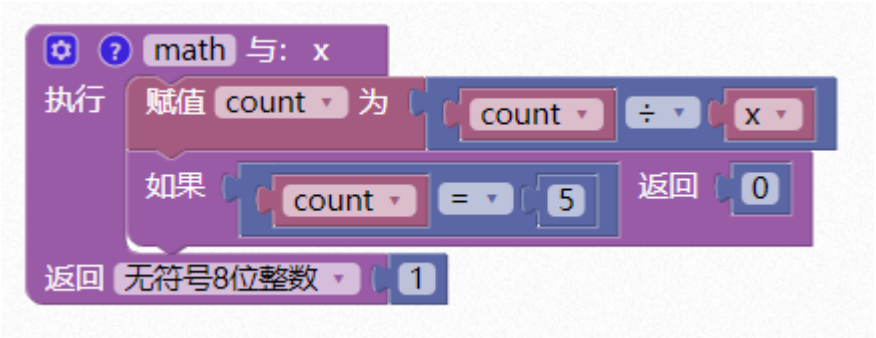
示例 2:

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。



显示模块

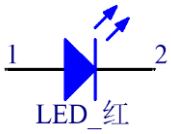
LED 流水灯

硬件概述

发光二极管是一种常用的发光器件，通过电子与空穴复合释放能量发光，它在照明领域应用广泛。发光二极管可高效地将电能转化为光能，在现代社会具有广泛的用途，如照明、平板显示、医疗器件等。

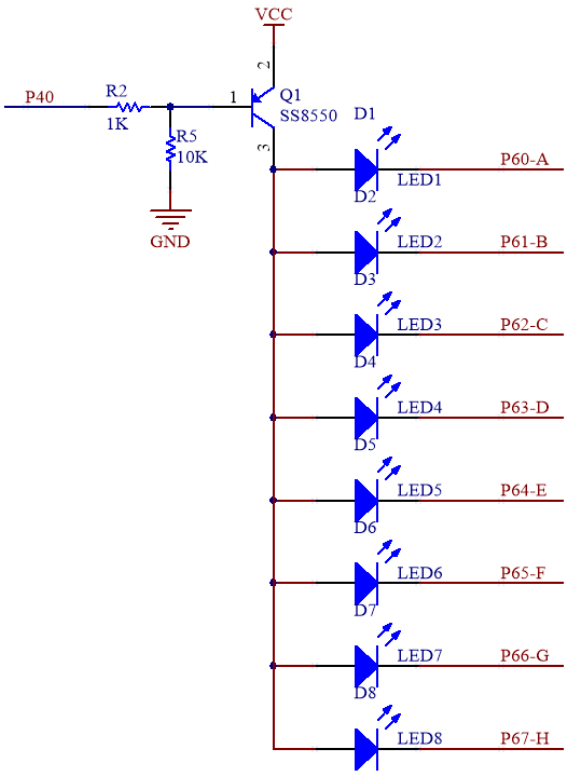
这种电子元件早在 1962 年出现，早期只能发出低光度的红光，之后发展出其他单色光的版本，时至今日能发出的光已遍及可见光、红外线及紫外线，光度也提高到相当的光度。而用途也由初时作为指示灯、显示板等；随着技术的不断进步，发光二极管已被广泛地应用于显示器和照明。

引脚定义



序号	符号	管脚名	功 能 描 述
1	1	正极	电源正极输入
2	2	负极	输出到电源负极

电路原理图



图形化模块

1. 打开 8 个 LED 流水灯电源

根据硬件原理图可知，R5 电阻下拉，流水灯电源默认为打开状态。

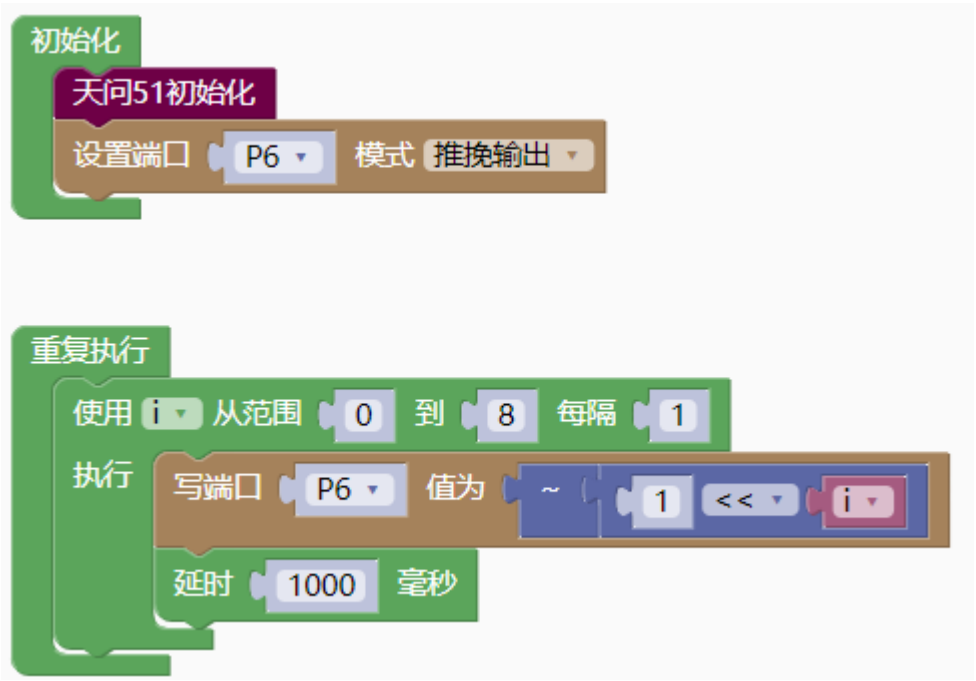
2. 关闭 8 个 LED 流水灯电源

示例代码 1

通过 P40 LED 流水灯电源控制引脚控制 8 个 LED 同时闪烁。

示例代码 2

打开八个 LED 流水灯电源，P6 端口设置为推挽输出，设置 P6 端口的每一个引脚轮流为低电平，每隔 1000ms 右移，循环 8 次。



调用函数代码

```
引入头文件
```

```
#include "lib/led8.h"
```

```
void led8_enable()//LED 使能函数，参数无
```

示例代码 1

```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  P6M1=0x00;P6M0=0xff;//推挽输出
  P6 = 0;
}

void loop()
{
  led8_enable();//打开 8 个 LED 流水灯电源
  delay(1000);
```

```
  led8_disable();//关闭 8 个 LED 流水灯电源
  delay(1000);
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

示例代码 2

```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"

uint8 i;

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
```

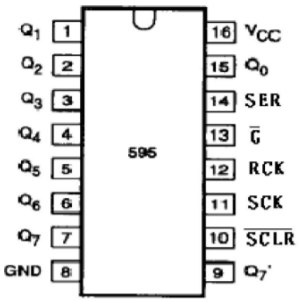
```
  P6M1=0x00;P6M0=0xff;//推挽输出
}

void loop()
{
  for (i = 0; i < 8; i = i + 1) {
    P6 = ~(1<<i);
    delay(1000);
  }
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

595 移位寄存器

硬件概述

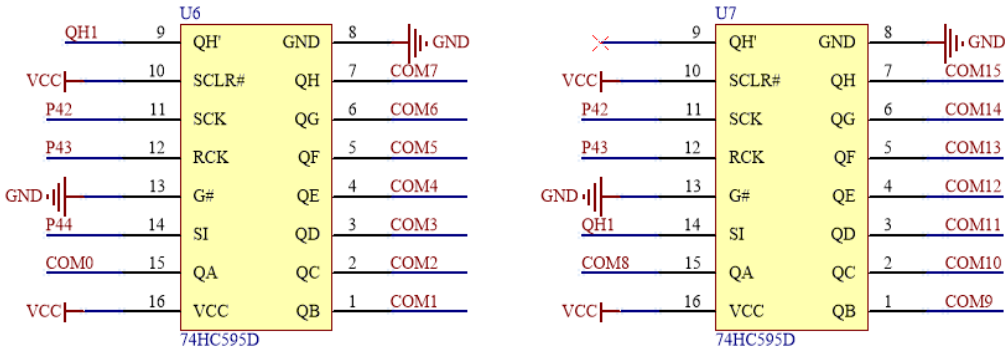
74HC595 是一个 8 位串行输入、并行输出的位移缓存器：并行输出为三态输出。在 SCK 的上升沿，串行数据由 SDL 输入到内部的 8 位位移缓存器，并由 Q7'输出，而并行输出则是在 LCK 的上升沿将在 8 位位移缓存器的数据存入到 8 位并行输出缓存器。当串行数据输入端 OE 的控制信号为低使能时，并行输出端的输出值等于并行输出缓存器所存储的值。

引脚定义



序号	符号	管脚名	功 能 描 述
1	Q0--Q7	并行输出端	8 位并行数据输出
2	Q7’	串行输出	串行数据输出
3	/SCLR	复位端	主复位（低电平有效）
4	SCK	数据输入时钟线	移位寄存器时钟，上升沿移位
5	RCK	输出存储器锁存时钟线	锁存寄存器时钟，上升沿存储
6	/G	输出有效（低电平有效）	输出使能端，为低电平使，输出选通；为高电平时，输出为 3 态
7	SER	串行数据输入	串行数据输入端
8	VCC	电源	供电管脚
9	GND	地	信号接地和电源接地

电路原理图



COM0-COM7 对应数码管、COM8-COM15 对应点阵。

图形化模块

1. 595 DS 引脚初始化，STCP 引脚初始化，SHCP 引脚初始化。

3. 595 禁止点阵和数码管输出。

4. 595 输出位选择 COM 口。

5. 595 启用数码管。

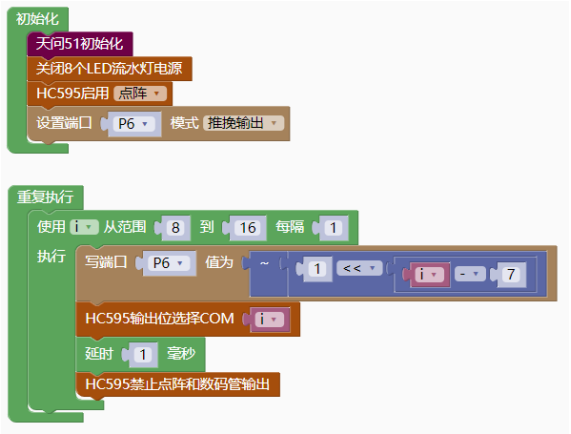
6. 595 启用点阵。

示例代码 1

初始化配置 595 的引脚，设置 P6 端口为推挽输出，595 启用数码管。
选择 COM0 为输出口，写 P6 端口为 0。

示例代码 2

初始化配置 595 的引脚，595 启用点阵。点阵显示点。



调用函数代码

```
引入头文件

#include "lib/hc595.h"
```

```
预定义 595 连接引脚，SI 的引脚，RCK 的引脚，SCK 的引脚。

#define HC595_DS   P4_4//SI 的引脚

#define HC595_STCP P4_3//RCK 的引脚

#define HC595_SHCP P4_2//SCK 的引脚
```

```
void hc595_init()//595 初始化函数，参数无
```

```
void hc595_bit_select(uint8 index) //595 发送位选函数，参数：(0~15)位
```

```
void hc595_disable();//HC595 禁止点阵和数码管输出
```

```
void hc595_enable_nix()//595 使能数码管函数，参数: 数组地址，数据长度
```

```
void hc595_enable_matrix()// 595 使能点阵函数，参数: 数组地址，数据长度
```

示例代码 1

```
#define HC595_DS    P4_4
#define HC595_DS_MODE {P4M1&=~0x10;P4M0|=0x10;}//P4_4 推挽输出
#define HC595_STCP P4_3
#define HC595_STCP_MODE {P4M1&=~0x08;P4M0|=0x08;}//P4_3 推挽输出
#define HC595_SHCP P4_2
#define HC595_SHCP_MODE {P4M1&=~0x04;P4M0|=0x04;}//P4_2 推挽输出

#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
```

```
}

void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  hc595_init();//HC595 初始化
  hc595_enable_nix();
  P6M1=0x00;P6M0=0xff;//推挽输出
  P6 = 0;
  hc595_bit_select(0);
}

void loop()
{

}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

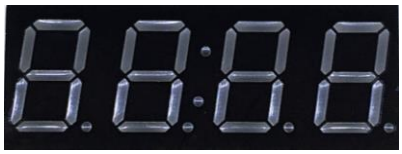
示例代码 2

```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"

uint8 i;

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
```

```
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  hc595_enable_matrix();
  P6M1=0x00;P6M0=0xff;//推挽输出
}

void loop()
{
  for (i = 8; i < 16; i = i + 1) {
    P6 = ~(1<<(i - 7));
    hc595_bit_select(i);
    delay(1);
    hc595_disable();//HC595 禁止点阵和数码管输出
  }
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

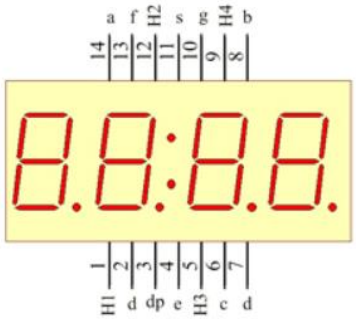
数码管模块

硬件概述



共阳数码管是指将所有发光二极管的阳极接到一起，形成公共阳极（COM）的数码管，共阳数码管在应用的时候，应该将 COM 端口接到正极，当某一段发光二极管的阴极为低电平的时候，相对应的段就点亮，当某一字段的阴极为高电平的时候，相对应段就不亮。

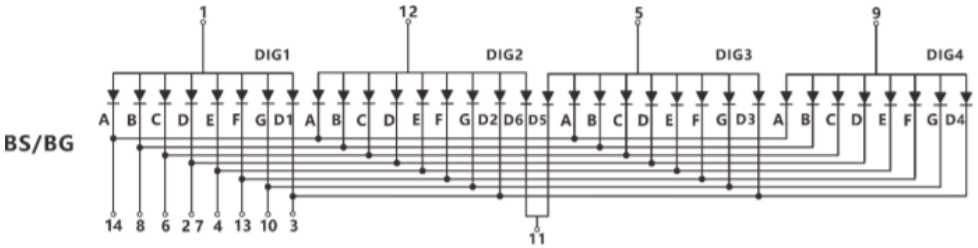
引脚定义



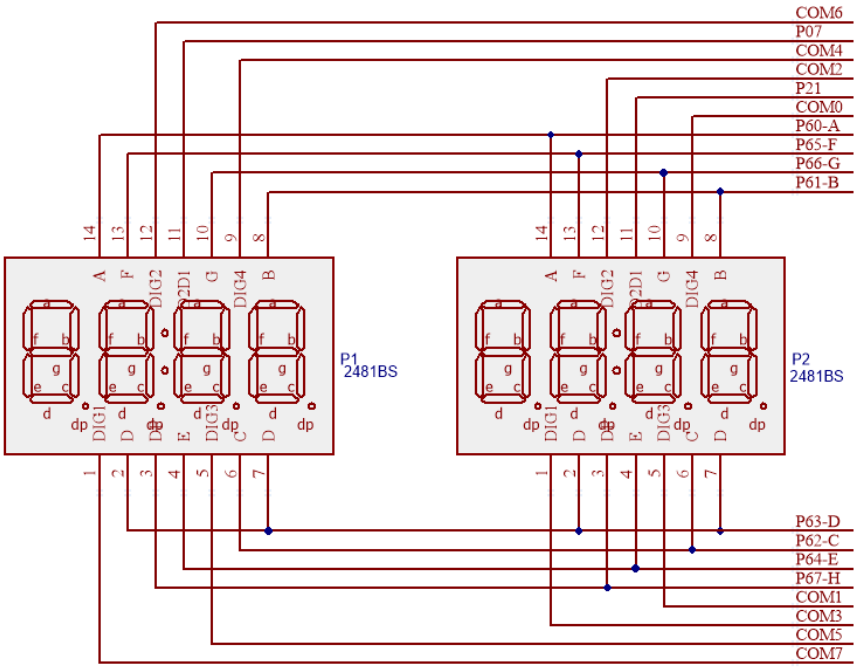
s脚为时钟点的段选端
公共端H2对应时钟下面的点
公共端H3对应时钟上面的点

序号	符号	管脚名	功 能 描 述
1、12、5，9	H1、H2、H3、H4	COM	公共端口
14、8、6、2 和 7、4、13、10、3	a、b、c、d、e、f、g、dp	段码	0-7 段数码管段选端口
11	s	段选端	时钟的段选端

数码管内部连线图



电路原理图



图形化模块

1. 初始化数码管和左右冒号相对应端口。

2. 数码管扫描回调函数

3. 数码管清屏函数

4. 数码管显示整数

5. 数码管显示浮点数

6. 数码管显示时间

7. 数码管显示时间，显示的位置

8. 数码管清除位

9. 数码管更新显示缓存

10. 数码管设置指定位小数点状态

示例代码 1

八位数码管显示整数 1~8。

示例代码 2

数码管显示浮点数 3.1415，精度设置 4 位。

示例代码 3

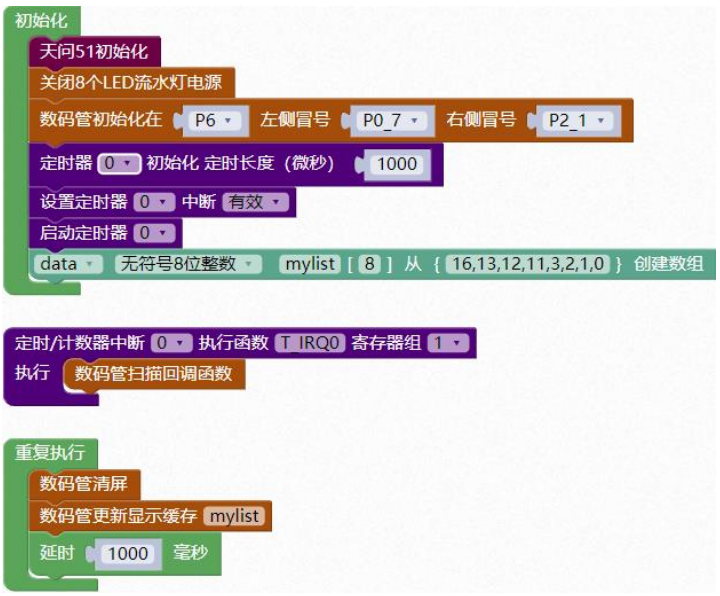
数码管设置时间为 12：30，设置在右侧显示。

示例代码 4

```
//段码
//0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,-,NONE
code uint8 _nix_seg[18]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0
x90,0x88,0x83,0xC6,0xA1,0x86,0x8E,0xBF,0xFF};
//显示缓存
xdata uint8 _nix_display_buff[8]={17,17,17,17,17,17,17,17};
```

数码管的驱动库，内部定义了 0-F 等 18 个常用码，和一个 8 字节的显示缓存。如果需要数码管显示特定数据，我们只需要更新显示缓存的显示码。

自定义缓存数据为 0，1，2，3，11，12，13，16，显示的数据为 0，1，2，3，b，c，d，-。数码管更新显示数据从 mylist 数组中调取。



调用函数代码

```
引入头文件

#include "lib/nixietube.h"
```

```
预定义数码管、左侧数码管冒号、右侧数码管冒号的连接引脚，引脚预处理输出

#define NIXIETUBE_PORT P6//数码管输出端口

#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出

#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号

#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽

输出

#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号

#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推

挽输出
```

```
void nix_init()//数码管显示初始化，参数无
```

```
void nix_scan_callback()//数码管扫描回调函数，参数无
```

```
void nix_display_num(int32 num) //数码管显示整数函数，参数：(-

9999999~99999999)
```

```
void nix_display_float(float num, uint8 precision)

//数码管显示浮点数函数，参数：浮点数，精度(1,2,3,4)
```

```
void nix_display_time(uint8 hour,uint8 minute,uint8 dir)
```

```
//数码管显示时间函数，参数：小时，分，0：左侧；1：右侧
```

```
void nix_display_clear()//数码管清屏，参数无
```

```
void nix_display_update_buf(uint8 *buf)

//数码管显示更新显示缓存数据函数，参数：缓存地址
```

```
void nix_display_clear_bit(uint8 nbit) //数码管清除指定位函数，参数（0~7）
```

```
void nix_display_time2(uint8 hour,uint8 minute,uint8 second)

//数码管显示时间， 参数 时，分，秒
```

示例代码 1

```
#define NIXIETUBE_PORT P6//数码管输出端口

#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出

#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号

#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽
输出

#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号

#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推
挽输出

#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"
```

```
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/nixietube.h"//引用 数码管 头文件

void twen_board_init()
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  nix_init();//数码管初始化
}
```

```
void loop()
{
  nix_scan_callback();//数码管扫描回调函数
  nix_display_num(12345678);//数码管显示整数 12345678
  delay(1);
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

示例代码 2

```
#define NIXIETUBE_PORT P6//数码管输出端口
#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出
#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号
#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽
输出
#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号
```

```
#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推
挽输出


#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/nixietube.h"//引用 数码管 头文件


void twen_board_init()
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}
```

```
void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  nix_init();//数码管初始化
}

void loop()
{
  nix_scan_callback();//数码管扫描回调函数
  nix_display_float(3.1415,4);//数码管显示浮点数 3.1415，4 位精度
  delay(1);
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

示例代码 3

```
#define NIXIETUBE_PORT P6//数码管输出端口

#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出

#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号

#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽
输出

#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号

#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推
挽输出


#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"

#include "lib/rgb.h"

#include "lib/delay.h"

#include "lib/led8.h"

#include "lib/nixietube.h"//引用 数码管 头文件


void twen_board_init()

{

  hc595_init();//HC595 初始化

  hc595_disable();//HC595 禁止点阵和数码管输出
```

```
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void Timer0Init(void) //1000 微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  nix_scan_callback();//数码管扫描回调函数
}

void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  nix_init();//数码管初始化
```

```
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}

void loop()
{
  nix_display_clear();//数码管清屏
  nix_display_time(12,30,1);//数码管显示时间 12：30，右侧显示
  delay(2000);
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

示例代码 4

```
#define NIXIETUBE_PORT P6//数码管输出端口
```

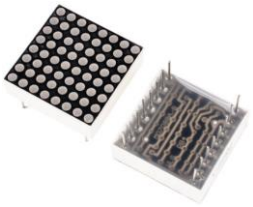
```
#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出

#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号

#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽
输出

#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号

#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推
挽输出


#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"

#include "lib/rgb.h"

#include "lib/delay.h"

#include "lib/led8.h"

#include "lib/nixietube.h"//引用 数码管 头文件


uint8 mylist[8]={16,13,12,11,3,2,1,0};//自定义数组


void twen_board_init()

{

  hc595_init();//HC595 初始化
```

```
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void Timer0Init(void) //1000 微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  nix_scan_callback();//数码管扫描回调函数
}

void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
```

```
  nix_init();//数码管初始化
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}

void loop()
{
  nix_display_clear();//数码管清屏
  nix_display_update_buf(mylist);//数码管更新显示缓存 0,1,2,a,b,c,-
  delay(1000);
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

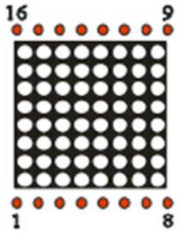
点阵模块

硬件概述



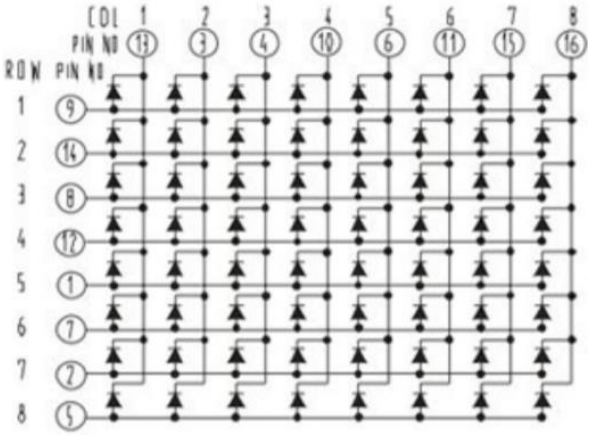
LED 点阵模块指的是利用封装 8*8 的模块组合点元板形成模块，它连接微处理器与 8 位数字的 7 段数字 LED 显示，也可以连接条线图显示器或者 64 个独立的 LED。其上包括一个片上的 B 型 BCD 编码器、多路扫描回路，段字驱动器，而且还有一个 8*8 的静态 RAM 用来存储每一个数据。只有一个外部寄存器用来设置各个 LED 的段电流。每个数据可以寻址在更新时不需要改写所有的显示。LED 点阵显示模块可显示汉字、图形、动画及英文字符等;显示方式有静态、横向滚动、垂直滚动和翻页显示等。

引脚定义

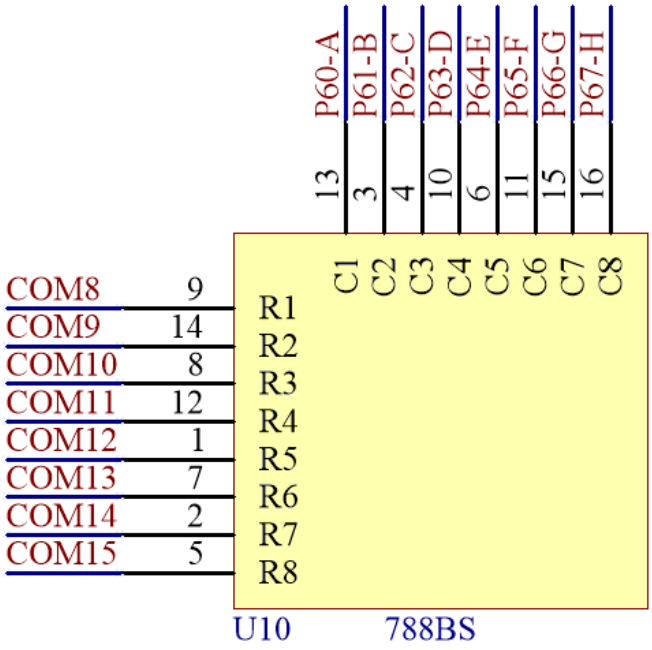


序号	符号	管脚名	功 能 描 述
1	1、2、5、7、8、9、12、14	COM	公共正极
2	3、4、6、10、11、13、15、16	负极	电源负极

点阵内部连线图



电路原理图



图形化模块

1. 点阵初始化在 P6 端口。

2. 点阵扫描回调函数。

3. 点阵清屏函数。

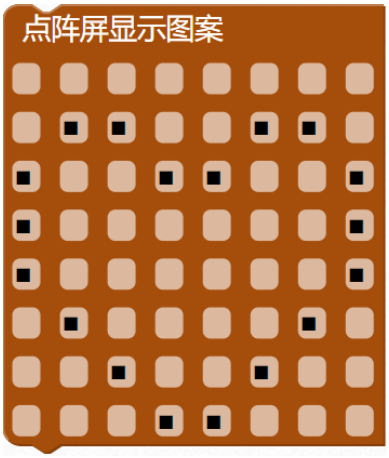
4. 点阵显示数字。

5. 点阵显示字符串。

6. 设置点阵在第几行，第几列显示点。

7. 设置点阵在第几行，第几列清除点。

8. 设置点阵显示图案。



9. 点阵更新显示缓存

10. 设置点阵显示自带图案

示例代码 1

设置点阵在第 0 行，第 0 列显示一个点。

示例代码 2

设置点阵显示数字 123。

示例代码 3

设置点阵显示字符串 abcd。

示例代码 4

设置点阵显示大爱心，300 毫秒后切换小爱心，依次循环。

示例代码 5

显示自定义图案，设置数组数据 0xe3,0xc1,0x81,0x03,0x03,0x81,0xc1,0xe3，点阵更新显示数据从 mylist 数组中调用数据。

调用函数代码

```
引入头文件

#include "lib/matrix.h"
```

```
预定义点阵连接引脚，引脚预处理输出

#define MATRIX_PORT P6//点阵的引脚

#define MATRIX_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出
```

```
void matrix_init()//点阵屏初始化，参数无
```

```
void matrix_scan_callback()

//点阵屏扫描回调函数，(需要在 1ms 定时器中断函数里调用总周期 16ms<视觉 20ms)，

参数无
```

```
matrix_set_pixel(uint8 x,uint8 y,uint8 state)

//点阵屏设置指定点亮灭状态

// 参数: x 坐标（0-7），y 坐标（0-7），1：亮；0：灭.

// (0,7)-------(7,7)
```

```
//   |                |

//   |                |

//   |                |

// (0,0)-------(7,0)
```

```
void matrix_clear()//点阵屏清屏，参数无
```

```
void matrix_update_buf(uint8 *from) //点阵屏显示缓存更新数据，参数:子模数组指针
```

```
void matrix_display_string(uint8 *chr);// 点阵屏显示字符，参数: 字符
```

```
void matrix_display_num(int16 value);//点阵显示数字，参数：数字
```

示例代码 1

```
#define MATRIX_PORT P6//点阵的引脚

#define MATRIX_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出



#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"

#include "lib/rgb.h"

#include "lib/delay.h"

#include "lib/matrix.h"//引用 点阵 头文件

#include "lib/led8.h"
```

```
void twen_board_init()
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  matrix_init();//点阵初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
}

void loop()
{
  matrix_scan_callback();//点阵扫描回调函数
  matrix_set_pixel(0,0,1); //点阵屏设置第一个点状态亮
```

```
}


void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

示例代码 2

```
#define MATRIX_PORT P6//点阵的引脚
#define MATRIX_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出


#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/matrix.h"//调用 点阵 头文件
#include "lib/led8.h"
#include "lib/oled.h"
```

```
void twen_board_init()
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void Timer0Init(void) //1000 微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  matrix_scan_callback();//点阵扫描回调函数
}
```

```
void setup()
{
  twen_board_init();//天问 51 初始化
  matrix_init();//点阵初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}

void loop()
{
  matrix_display_num(123); //点阵显示数字 123
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
```

```
}
```

示例代码 3

```
#define MATRIX_PORT P6//点阵的引脚

#define MATRIX_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出



#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"

#include "lib/rgb.h"

#include "lib/delay.h"

#include "lib/matrix.h"//引用 点阵 头文件

#include "lib/led8.h"

#include "lib/oled.h"



void twen_board_init()

{

  hc595_init();//HC595 初始化

  hc595_disable();//HC595 禁止点阵和数码管输出

  rgb_init();//RGB 初始化

  delay(10);

  rgb_show(0,0,0,0);//关闭 RGB
```

```
  delay(10);
}

void Timer0Init(void) //1000 微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  matrix_scan_callback();//点阵扫描回调函数
}

void setup()
{
  twen_board_init();//天问 51 初始化
  matrix_init();//点阵初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
```

```
  TR0 = 1;// 启动定时器
}


void loop()
{
  matrix_display_string("abcd");//点阵显示字符 abcd
}


void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

示例代码 4

```
#define MATRIX_PORT P6//点阵的引脚
#define MATRIX_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出


#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
```

```
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/matrix.h"//引用 点阵 头文件
#include "lib/led8.h"

void twen_board_init()
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void Timer0Init(void) //1000 微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}
```

```
void T_IRQ0(void) interrupt 1 using 1{
  matrix_scan_callback();//点阵扫描回调函数
}

uint8_t matrix[8];

void setup()
{
  twen_board_init();//天问 51 初始化
  matrix_init();//点阵初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}

void loop()
{
  matrix[0] = 0xe3;matrix[1] = 0xc1;
  matrix[2] = 0x81;matrix[3] = 0x03;
```

```
  matrix[4] = 0x03;matrix[5] = 0x81;
  matrix[6] = 0xc1;matrix[7] = 0xe3;
  matrix_update_buf(matrix); //点阵屏显示缓存更新数据
  delay(300);
  matrix[0] = 0xff;matrix[1] = 0xe7;
  matrix[2] = 0xc3;matrix[3] = 0x87;
  matrix[4] = 0x87;matrix[5] = 0xc3;
  matrix[6] = 0xe7;matrix[7] = 0xff;
  matrix_update_buf(matrix);  //点阵屏显示缓存更新数据
  delay(300);
}


void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

示例代码 5

```
#define MATRIX_PORT P6//点阵的引脚
#define MATRIX_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出
```

```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/matrix.h"//引用 点阵 头文件
#include "lib/led8.h"

uint8 mylist[8]={0xe3,0xc1,0x81,0x03,0x03,0x81,0xc1,0xe3，};//自定义数组

void twen_board_init()
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}
```

```
void Timer0Init(void) //1000 微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  matrix_scan_callback();//点阵扫描回调函数
}

void setup()
{
  twen_board_init();//天问 51 初始化
  matrix_init();//点阵初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}
```

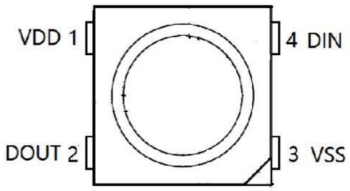
```
void loop()
{
  matrix_update_buf(mylist);//点阵更新显示自定义数组缓存数据
}
void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

RGB 彩灯模块

硬件概述

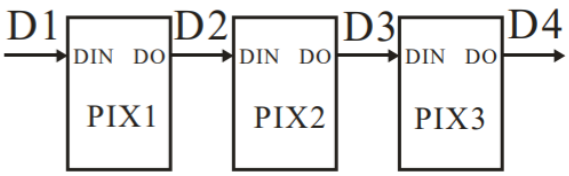
WS2812 是一个集控制电路与发光电路于一体的智能外控 LED 光源。其外型与一个 5050LED 灯珠相同，每个元件即为一个像素点。像素点内部包含了智能数字接口数据锁存信号整形放大驱动电路，还包含有高精度的内部 振荡器和可编程定电流控制部分，有效保证了像素点光的颜色高度一致。 数据协议采用单线归零码的通讯方式，像素点在上电复位以后，DIN 端接受从控制器传输过来的数据，首先送过来的 24bit 数据被第一个像素点提取后，送到像素点内部的数据锁存器，剩余的数据经过内部整形处理电路整 形放大后通过 DO 端口开始转发输出给下一个级联的像素点，每经过一个像素点的传输，信号减少 24bit。像素点 采用自动整形转发技术，使得该像素点的级联个数不受信号传送的限制，仅受限信号传输速度要求。

引脚定义

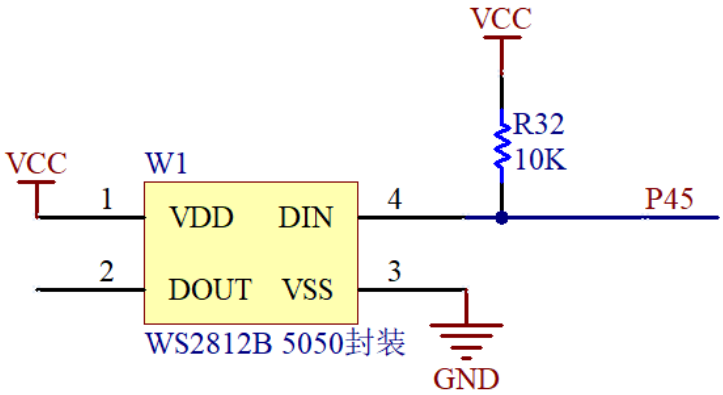


序号	符号	管脚名	功 能 描 述
1	VDD	电源	供电管脚
2	DOUT	数据输出	控制数据信号输出
3	VSS	地	信号接地和电源接地
4	DIN	数据输入	控制数据信号输入

串接方式



电路原理图



图形化模块

1. 初始化 RGB 的控制引脚和总共 RGB 灯数量

2. 设置第几个灯显示指定 RGB 颜色值

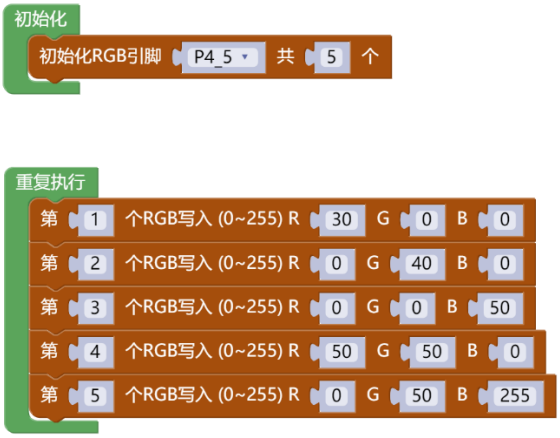
3. 设置第几个灯显示下拉框内常用颜色和亮度

示例代码 1

设置 1 个 RGB 灯，设置灯的 RGB 颜色为红色，亮度为 50。

示例代码 2

设置 5 个 RGB 灯，依次设置 5 个灯为指定的 RGB 颜色。



调用函数代码

```
引入头文件

#include "lib/rgb.h"
```

```
预定义 RGB 灯连接引脚，RGB 灯的数量，引脚预处理输出

#define RGB_PIN P4_5//RGB 灯的引脚

#define RGB_NUMLEDS 5 //RGB 灯的个数

#define RGB_PIN_MODE {P4M1&=~0x20;P4M0|=0x20;}//推挽输出
```

```
//=========================================================================

void rgb_init()//RGB 初始化函数，参数无
```

```
void rgb_show(uint8 num, uint8 r, uint8 g, uint8 b)

//RGB 显示函数，参数 num 第几个 RGB,参数 r 红色值，参数 g 绿色值，参数 b 蓝色值
```

示例代码 1

```
#define RGB_PIN P4_5//RGB 灯的引脚

#define RGB_NUMLEDS 5 //RGB 灯的个数

#define RGB_PIN_MODE {P4M1&=~0x20;P4M0|=0x20;}//推挽输出



#include <STC8HX.h>
```

```
uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/rgb.h"//引入 RGB 头文件

void setup()

{

   rgb_init();//RGB 初始化函数

}

void loop()

{

   rgb_show(0,30,0,0);//第 0 个灯显示 RGB

   rgb_show(1,0,40,0);//第 1 个灯显示 RGB

   rgb_show(2,0,0,50);//第 2 个灯显示 RGB

   rgb_show(3,50,50,0);//第 3 个灯显示 RGB

   rgb_show(4,0,50,255);//第 4 个灯显示 RGB

}


void main(void)

{

   setup();

   while(1){

      loop();

   }
```

```
}
```

OLED 显示模块

硬件概述

OLED，即有机发光二极管（Organic Light-Emitting Diode），又称为有机电激光显示(Organic Electroluminesence Display，OELD)。OLED 同时具备自发光，不需背光源、对比度高、厚度薄、视角广、反应速度快、可用于挠曲性面板、使用温度范围广、构造及制程较简单等优异之特性。模块具有一下特点：

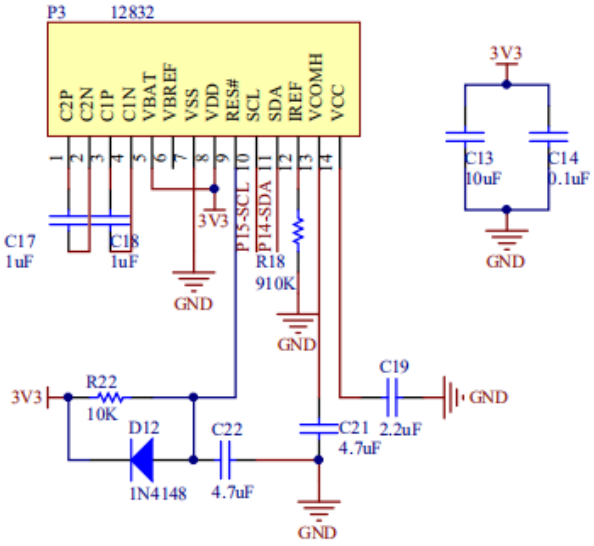
(1) 尺寸小，显示尺寸为 0.91 寸,屏幕尺寸 30mm*11.5mm。
(2) 高分辨率，分辨率为 128*32。
(3) 使用 IIC 通信，只需 2 根线即可控制 OLED。

引脚定义

标号	符号	引脚说明
1	C2P	电容 2 正极
2	C2N	电容 2 负极
3	C1P	电容 1 正极
4	C1N	电容 1 负极
5	VBAT	DC/DC 转换电路电源
6	VBREF	保留引脚
7	VSS	电源地
8	VDD	电源正极
9	RES#	控制器和驱动器的电源复位
10	SCL	IIC 的时钟脚
11	SDA	IIC 的数据脚
12	IREF	电流参考亮度调整

13	VCOMH	电压输出 COM 高电平
14	VCC	OLE 面板电源

电路原理图



图形化模块

1. 初始化 OLED

2. OLED 关闭显示

3. OLED 更新显示

4. OLED 清屏

5. OLED 在坐标 X,Y 显示一个像素点

6. OLED 在坐标 X,Y 清除像素点

7. OLED 在坐标 X,Y 显示一个字符

8. OLED 在坐标 X,Y 显示一个字符串

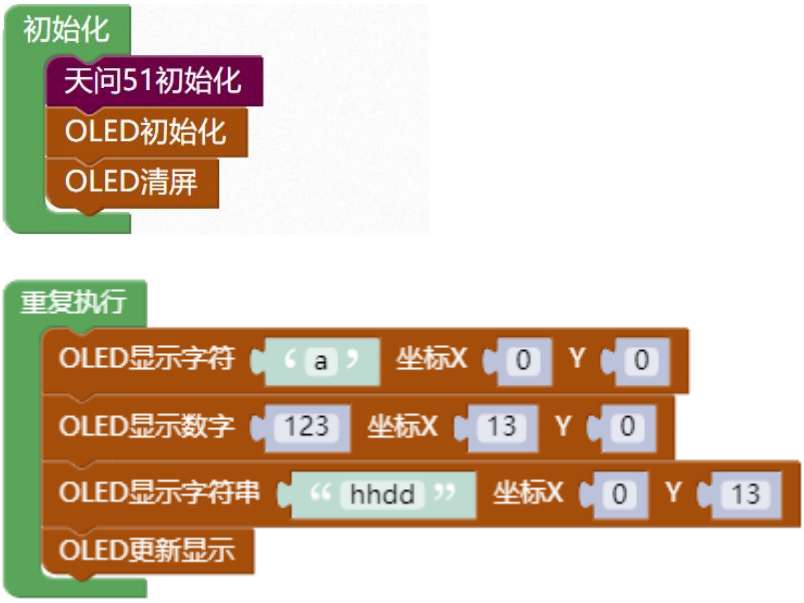
9. OLED 在坐标 X,Y 显示一个数值

10. OLED 在坐标 X,Y 显示一个指定字体大小的汉字

11. OLED 在一个范围内显示图片

示例代码 1

OLED 显示字符“a”在坐标(0,0)，显示数字“123”在坐标(13,0)，显示字符串在坐标(0,13)。

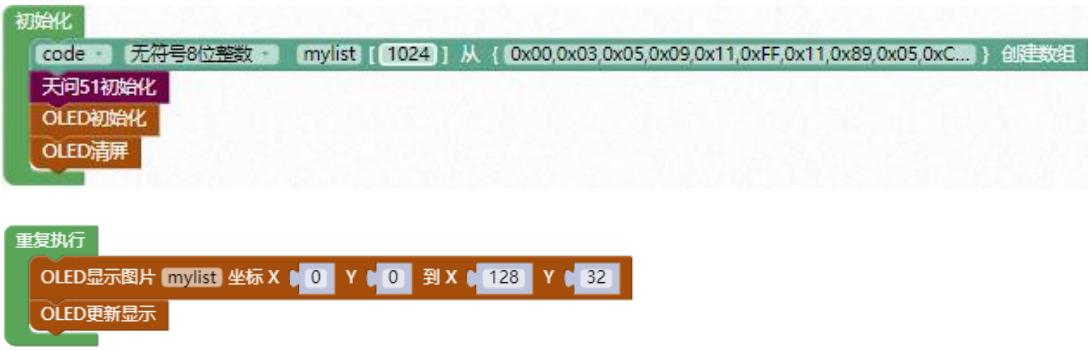


示例代码 2

OLED 显示 12*12 字体大小的汉字“好好”在坐标(0,0)，显示 16*16 字体大小的汉字“搭”在坐标(24,0)，显示 24*24 字体大小的汉字“搭”在坐标(40,0)。

示例代码 3

OLED 显示转换过的 BMP 图片。



调用函数代码

```
引入头文件

#include "lib/oled.h"
```

```
void oled_init()//OLED 初始化函数，参数无
```

```
void oled_display_off()//关闭 OLED 函数，参数无
```

```
void oled_display()//OLED 更新显示函数，参数无
```

```
void oled_clear()//OLED 清屏函数，参数无
```

```
void oled_set_pixel(uint8 x, uint8 y,uint8 pixel)

//OLED 设置点,参数 x,y 为要显示的坐标,参数 pixel 写 1 该点亮，写 0 灭
```

```
void oled_show_char(int8 x,int8 y,uint8 chr)

//OLED 显示单个字符(字符高 8 宽 5),参数 x,y 为要显示的坐标,参数 y 为显示的字符
```

```
void oled_show_string(int8 x,int8 y,uint8 *chr)

//OLED 在指定位置显示字符串(字符高 8，间距 8),参数 x,y 为要显示的坐标,参数 chr 为//

显示的字符串
```

```
void oled_show_font12(const uint8* hz,int x,int y)

//OLED 在指定位置显示 12x12 汉字,参数 hz 为要显示的汉字，参数 x,y 为要显示的坐标
```

```
void oled_show_font16(const uint8* hz,int x,int y)

//OLED 在指定位置显示 16x16 汉字,参数 hz 为要显示的汉字，参数 x,y 为要显示的坐标
```

```
void oled_show_font24(const uint8* hz,int x,int y)

//OLED 在指定位置显示 24x24 汉字,参数 hz 为要显示的汉字，参数 x,y 为要显示的坐标
```

```
void oled_show_bmp(uint8 x0, uint8 y0,uint8 x1, uint8 y1,uint8* BMP)
//显示转换过的 BMP 图片，参数 x0,y0 为起点坐标，参数 x1,y1 为终点坐标，参数 BMP
//为要显示的图片
```

示例代码 1

```
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/oled.h"


void twen_board_init()
{
  hc595_init();
  hc595_disable();
  rgb_init();
  delay(100);
  rgb_show(0,0,0,0);//熄灭 RGB
  delay(100);
}
```

```
void setup()
{
  twen_board_init();
  oled_init();//OLED 初始化
  oled_clear();//OLED 清屏
}

void loop()
{
  oled_show_char(0,0,'a');
  oled_show_num(13,0,123);
  oled_show_string(0,13,"hhdd");
  oled_display();//OLED 更新显示
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
```

```
}
```

示例代码 2

```
#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"

#include "lib/rgb.h"

#include "lib/delay.h"

#include "lib/oled.h"


void twen_board_init()

{

  hc595_init();

  hc595_disable();

  rgb_init();

  delay(100);

  rgb_show(0,0,0,0);//熄灭 RGB

  delay(100);

}


void setup()
```

```
{
  twen_board_init();
  oled_init();//OLED 初始化
  oled_clear();//OLED 清屏
}

void loop()
{
  oled_show_font12("好好",0,0);
  oled_show_font16("搭",24,0);
  oled_show_font24("搭",40,0);
  oled_display();//OLED 更新显示
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

示例代码 3

```
#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"

#include "lib/rgb.h"

#include "lib/delay.h"

#include "lib/oled.h"


code uint8 mylist[1024]={

 0x00,0x03,0x05,0x09,0x11,0xFF,0x11,0x89,0x05,0xC3,0x00,0xE0,0x00,0xF0,0x00,0xF
8,

 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x44,0x28,0xFF,0x11,0xAA,0x44,0x00,0x00,0x
00,

 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,

 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,

 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,

 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x83,0x01,0x38,0x44,0x82,0x
92,
```

```
0x92,0x74,0x01,0x83,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x7C,0x44,0xFF,0x01,0x7
D,
0x7D,0x7D,0x01,0x7D,0x7D,0x7D,0x7D,0x01,0x7D,0x7D,0x7D,0x7D,0x7D,0x01,0xFF
,0x00,
0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x
01,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x01,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x01,0x00,0x00,0x00,0x
00,
0x00,0x00,0x01,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x01,0x
01,
0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3F,0x3F,0x03,0x0
3,
0xF3,0x13,0x11,0x11,0x11,0x11,0x11,0x11,0x01,0xF1,0x11,0x61,0x81,0x01,0x01,0x0
1,
0x81,0x61,0x11,0xF1,0x01,0x01,0x01,0x01,0x41,0x41,0xF1,0x01,0x01,0x01,0x01,0x0
1,
0xC1,0x21,0x11,0x11,0x11,0x11,0x21,0xC1,0x01,0x01,0x01,0x01,0x41,0x41,0xF1,0x
01,
0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x11,0x11,0x11,0x11,0x11,0xD3,0x
33,
0x03,0x03,0x3F,0x3F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xE0,0xE0,0x00,0x0
0,
0x7F,0x01,0x01,0x01,0x01,0x01,0x01,0x00,0x00,0x7F,0x00,0x00,0x01,0x06,0x18,0x0
6,
0x01,0x00,0x00,0x7F,0x00,0x00,0x00,0x00,0x40,0x40,0x7F,0x40,0x40,0x00,0x00,0x0
0,
```

```
0x1F,0x20,0x40,0x40,0x40,0x40,0x20,0x1F,0x00,0x00,0x00,0x00,0x40,0x40,0x7F,0x4
0,
0x40,0x00,0x00,0x00,0x00,0x60,0x00,0x00,0x00,0x00,0x40,0x30,0x0C,0x03,0x00,0x
00,
0x00,0x00,0xE0,0xE0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x07,0x07,0x06,0x
06,
0x06,0x06,0x04,0x04,0x04,0x84,0x44,0x44,0x44,0x84,0x04,0x04,0x84,0x44,0x44,0x
44,
0x84,0x04,0x04,0x04,0x84,0xC4,0x04,0x04,0x04,0x04,0x84,0x44,0x44,0x44,0x84,0x
04,
0x04,0x04,0x04,0x04,0x84,0x44,0x44,0x44,0x84,0x04,0x04,0x04,0x04,0x04,0x84,0x
44,
0x44,0x44,0x84,0x04,0x04,0x84,0x44,0x44,0x44,0x84,0x04,0x04,0x04,0x04,0x06,0x
06,
0x06,0x06,0x07,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x10,0x18,0x14,0x12,0x11,0x00,0x00,0x0F,0x10,0x10,0x1
0,
0x0F,0x00,0x00,0x00,0x10,0x1F,0x10,0x00,0x00,0x00,0x08,0x10,0x12,0x12,0x0D,0x
00,
0x00,0x18,0x00,0x00,0x0D,0x12,0x12,0x12,0x0D,0x00,0x00,0x18,0x00,0x00,0x10,0x
18,
0x14,0x12,0x11,0x00,0x00,0x10,0x18,0x14,0x12,0x11,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0x80,0x80,0x
80,
0x80,0x80,0x80,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x7F,0x03,0x0C,0x30,0x0C,0x03,0x7F,0x00,0x00,0x38,0x54,0x54,0x58,0x00,0x
00,
0x7C,0x04,0x04,0x78,0x00,0x00,0x3C,0x40,0x40,0x7C,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xAA,0xAA,0x
AA,
0x28,0x08,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0,
```

```
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x03,0x0C,0x30,0x0C,0x03,0x
7F,
 0x00,0x00,0x26,0x49,0x49,0x49,0x32,0x00,0x00,0x7F,0x02,0x04,0x08,0x10,0x7F,0x0
0,
};


void twen_board_init()
{
  hc595_init();
  hc595_disable();
  rgb_init();
  delay(100);
  rgb_show(0,0,0,0);//熄灭 RGB
  delay(100);
}


void setup()
{
  twen_board_init();
  oled_init();//OLED 初始化
```

```
  oled_clear();//OLED 清屏
}

void loop()
{
  oled_show_bmp(0,0,128,32,mylist);
  oled_display();//OLED 更新显示
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

LCD1602 显示模块

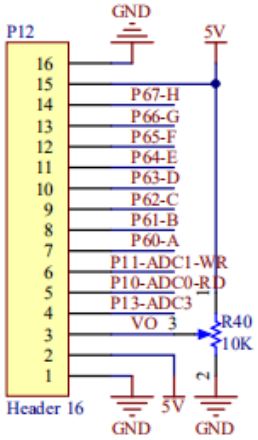
硬件概述

1602A 字符型液晶显示模块是专门用于显示字母、数字、符号等的点阵型液晶显示模块。分 4 位和 8 位数据传输方式。提供 5×7 点阵 + 游标的显示模式。提供显示数据缓冲区 DDRAM、字符发生器 CGROM 和字符发生器 CGRAM，可以使用 CGRAM 来存储自己定义的最多 8 个 5×8 点阵的图形字符的字模数据。提供了丰富的指令设置：清显示；游标回原点；显示开/关;游标开/关;显示字符闪烁;游标移位;显示移位等。

引脚定义

标号	符号	引脚说明	标号	符号	引脚说明
1	VSS	电源地	9	D2	数据
2	VDD	电源正极	10	D3	数据
3	VL	液晶显示偏压	11	D4	数据
4	RS	数据/命令选择	12	D5	数据
5	R/W	读/写选择	13	D6	数据
6	E	使能信号	14	D7	数据
7	D0	数据	15	BLA	背光源正极
8	D1	数据	16	BLK	背光源负极

电路原理图

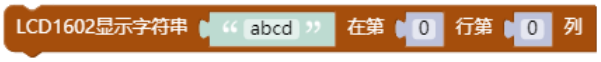


图形化模块

1. 初始化 LCD1602 的控制引脚

2. LCD1602 显示单个字符在第几行第几列

3. LCD1602 显示字符串在第几行第几列

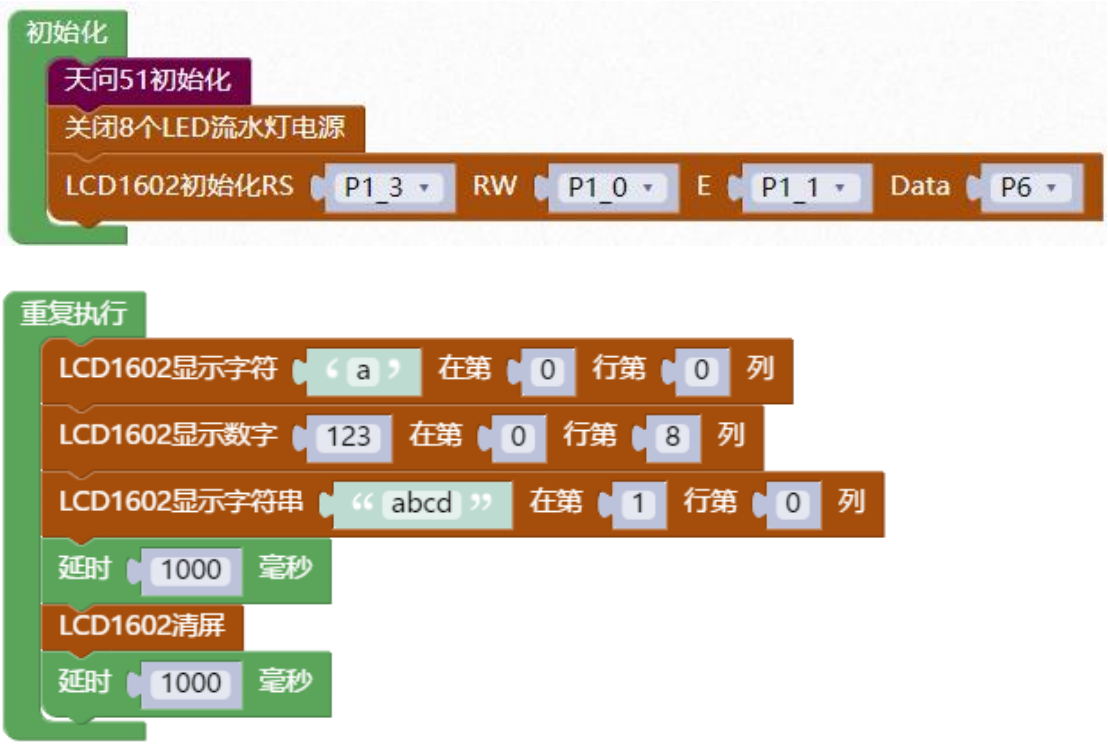


4. LCD1602 显示数字在第几行第几列

5. LCD1602 清屏

示例代码 1

LCD1602 每隔 1 秒显示单个字符“a”在第 0 行第 0 列，显示字符串在第 1 行第 0 列，显示数字在第 0 行第 8 列。



调用函数代码

```
引入头文件

#include "lib/lcd1602.h"
```

```
预定义 LCD1602 连接引脚，引脚预处理输出

#define LCD1602_RS P1_3

#define LCD1602_RS_OUT {P1M1&=~0x08;P1M0|=0x08;}//推挽输出

#define LCD1602_RW P1_0

#define LCD1602_RW_OUT {P1M1&=~0x01;P1M0|=0x01;}//推挽输出

#define LCD1602_E P1_1

#define LCD1602_E_OUT {P1M1&=~0x02;P1M0|=0x02;}//推挽输出

#define LCD1602_Data P6

#define LCD1602_Data_OUT {P6M1=0x00;P6M0=0xff;}//推挽输出
```

```
void lcd1602_init()//LCD1602 初始化函数，参数无
```

```
void lcd1602_show_char(uint8 x, uint8 y, char c)
//LCD1602 显示一个字符,参数 x 显示在第几行,参数 y 显示在第几列,参数 c 显示的字符
```

```
void lcd1602_show_string(uint8 x, uint8 y, uint8 *str)
//LCD1602 显示字符串,参数 x 显示在第几行,参数 y 显示在第几列,参数 str 显示的字符//
串
```

```
void lcd1602_show_num(uint8 x,uint8 y,int num)
//LCD1602 显示数字,参数 x 显示在第几行,参数 y 显示在第几列,参数 num 显示的数字
```

```
void lcd1602_clear();//LCD1602 清屏函数，参数无
```

示例代码 1

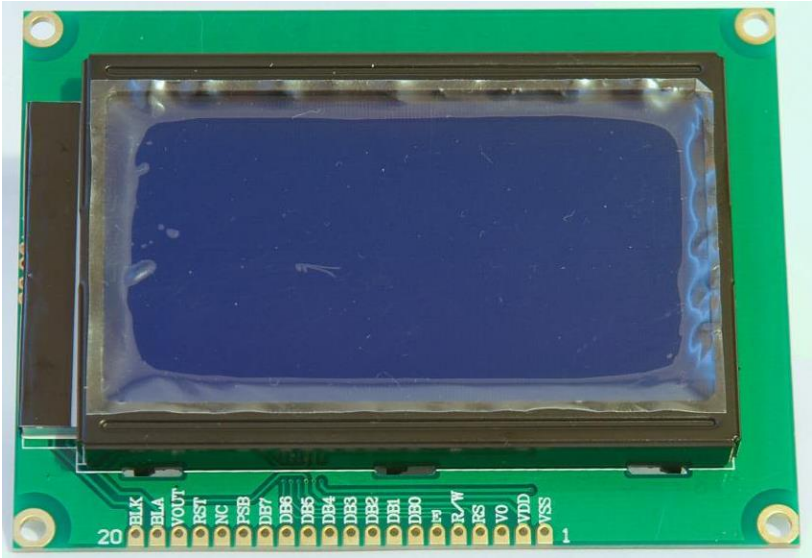
```
#define LCD1602_RS P1_3
#define LCD1602_RS_OUT {P1M1&=~0x08;P1M0|=0x08;}//推挽输出
#define LCD1602_RW P1_0
#define LCD1602_RW_OUT {P1M1&=~0x01;P1M0|=0x01;}//推挽输出
#define LCD1602_E P1_1
#define LCD1602_E_OUT {P1M1&=~0x02;P1M0|=0x02;}//推挽输出
#define LCD1602_Data P6
```

```
#define LCD1602_Data_OUT {P6M1=0x00;P6M0=0xff;}//推挽输出
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/lcd1602.h"
void twen_board_init()
{
  hc595_init();
  hc595_disable();
  rgb_init();
  delay(100);
  rgb_show(0,0,0,0);//熄灭 RGB
  delay(100);
}
void setup()
{
  twen_board_init();
  led8_disable();//关闭 8 个 LED 流水灯电源
```

```
  lcd1602_init();//LCD1602 初始化
}
void loop()
{
  lcd1602_show_char(0,0,'a');
  lcd1602_show_num(8,0,123);
  lcd1602_show_string(0,1,"abcd");
  delay(1000);
  lcd1602_clear();//LCD1602 清屏
  delay(1000);
}
void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

LCD12864 显示模块

硬件概述

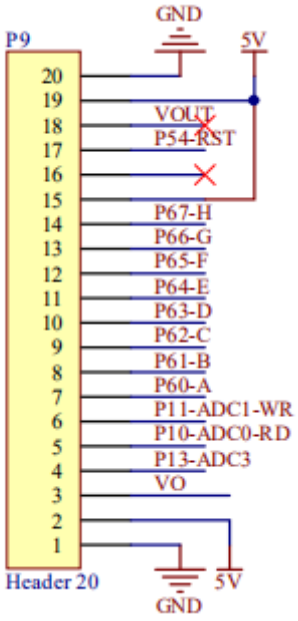


LCD12864 是一种具有 4 位/8 位并行、2 线或 3 线串行多种接口方式，内部含有国标一级、二级简体中文字库的点阵图形液晶显示模块；其显示分辨率为 128×64，内置 8192 个 16*16 点汉字，和 128 个 16*8 点 ASCII 字符集。利用该模块灵活的接口方式和简单、方便的操作指令，可构成全中文人机交互图形界面。可以显示 8×4 行 16×16 点阵的汉字. 也可完成图形显示.低电压低功耗是其又一显著特点。由该模块构成的液晶显示方案与同类型的图形点阵液晶显示模块相比，不论硬件电路结构或显示程序都要简洁得多，且该模块的价格也略低于相同点阵的图形液晶模块。

引脚定义

标号	符号	引脚说明	标号	符号	引脚说明
1	VSS	电源地	11	D4	数据
2	VDD	电源正极	12	D5	数据
3	VL	液晶显示偏压	13	D6	数据
4	RS	数据/命令选择	14	D7	数据
5	R/W	读/写选择	15	PSD	H:并口方式；L:串口方式
6	E	使能信号	16	NC	空脚
7	D0	数据	17	/RESET	复位端
8	D1	数据	18	VOUT	LCD 驱动电压输出端
9	D2	数据	19	BLA	背光源正极
10	D3	数据	20	BLK	背光源负极

电路原理图



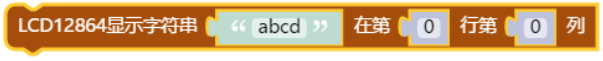
图形化模块

1. 初始化 LCD12864 的控制引脚

2. LCD12864 清除整个屏幕

3. LCD12864 显示单个字符在第几行第几列

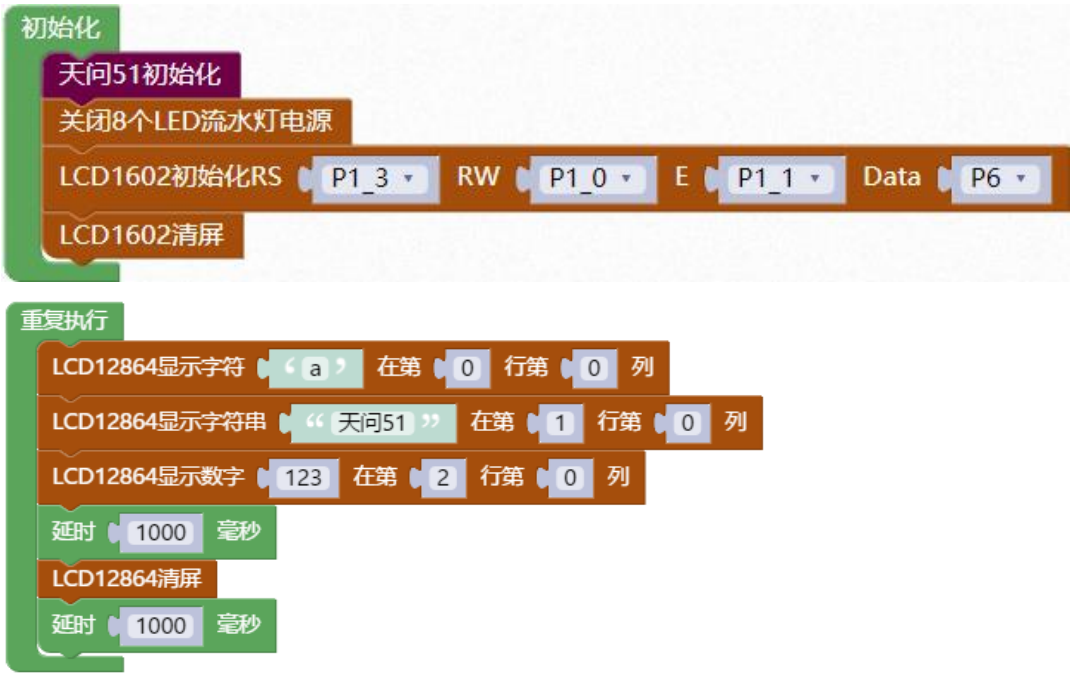
4. LCD12864 显示字符串在第几行第几列



5. LCD12864 显示数字在第几行第几列

示例代码 1

LCD12864 每隔 1s 在第 0 行第 0 列显示单个字符“a”，在第 1 行第 0 列显示字符串“天问 51”，在第 2 行第 0 列显示数字 123。



调用函数代码

```
引入头文件

#include "lib/lcd12864.h"
```

```
预定义 LCD12864 连接引脚，引脚预处理输出

#define LCD12864_RS P1_3

#define LCD12864_RS_OUT {P1M1&=~0x08;P1M0&=~0x08;} //双向 IO 口

#define LCD12864_RW P1_0

#define LCD12864_RW_OUT {P1M1&=~0x01;P1M0&=~0x01;} //双向 IO 口

#define LCD12864_E P1_1

#define LCD12864_E_OUT {P1M1&=~0x02;P1M0&=~0x02;} //双向 IO 口

#define LCD12864_RST P1_3

#define LCD12864_RST_OUT {P5M1&=~0x10;P5M0&=~0x10;} //双向 IO 口

#define LCD12864_Data P6
```

```
#define LCD12864_Data_OUT {P6M1=0x00;P6M0=0x00;} //双向 IO 口
```

```
void lcd12864_init()//LCD12864 初始化函数，参数无
```

```
void lcd12864_clear()//LCD12864 清屏函数，参数无
```

```
void lcd12864_show_char(uint8 X,uint8 Y,uint8 sig)
//LCD12864 显示一个字符,参数 X 显示在第几行,参数 Y 显示在第几列,参数 sig 要显示的
//字符
```

```
void lcd12864_show_string(uint8 X, uint8 Y, uint8 *str)
//LCD12864 显示字符串,参数 X 显示在第几行,参数 Y 显示在第几列,参数 str 要显示的字//
符串
```

```
void lcd12864_show_num(uint8 x,uint8 y,int num)
//LCD12864 显示数字,参数 x 显示在第几行,参数 y 显示在第几列,参数 num 显示的数字
```

示例代码 1

```
#define LCD12864_RS P1_3
#define LCD12864_RS_OUT {P1M1&=~0x08;P1M0&=~0x08;} //双向 IO 口
#define LCD12864_RW P1_0
#define LCD12864_RW_OUT {P1M1&=~0x01;P1M0&=~0x01;} //双向 IO 口
#define LCD12864_E P1_1
```

```
#define LCD12864_E_OUT {P1M1&=~0x02;P1M0&=~0x02;} //双向 IO 口
#define LCD12864_RST P1_3
#define LCD12864_RST_OUT {P5M1&=~0x10;P5M0&=~0x10;} //双向 IO 口
#define LCD12864_Data P6
#define LCD12864_Data_OUT {P6M1=0x00;P6M0=0x00;} //双向 IO 口
#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/lcd12864.h"


void twen_board_init()
{
  hc595_init();
  hc595_disable();
  rgb_init();
  delay(100);
  rgb_show(0,0,0,0);//熄灭 RGB
  delay(100);
```

```
}


void setup()
{
  twen_board_init();
  led8_disable();//关闭 8 个 LED 流水灯电源
  lcd12864_init();//LCD12864 初始化
  lcd12864_clear();//LCD12864 清屏
}
void loop()
{
  lcd12864_show_char(0,0,'a');
  lcd12864_show_string(1,0,"天问 51");
  lcd12864_show_num(2,0,123);
  delay(1000);
  lcd12864_clear();//LCD12864 清屏
  delay(1000);
}


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

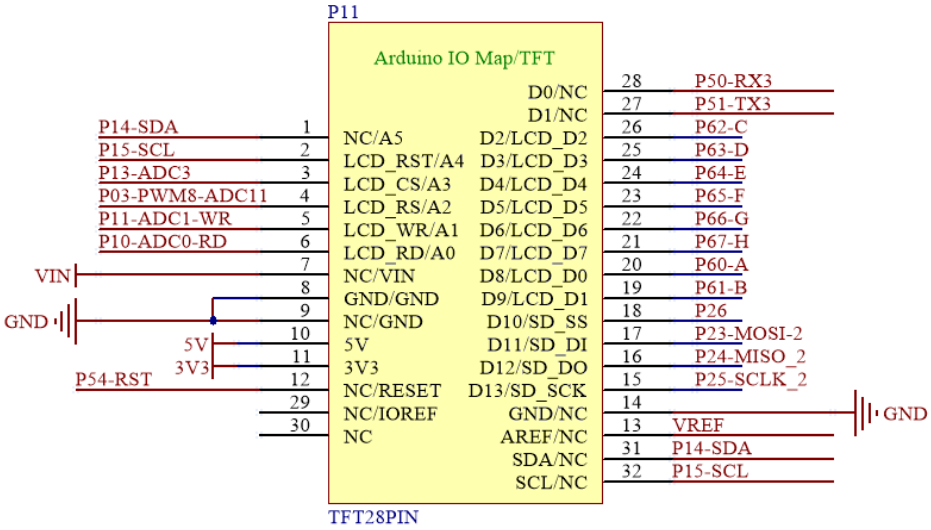
```
 while(1){

   loop();

 }
}
```

TFT 彩屏模块

硬件概述

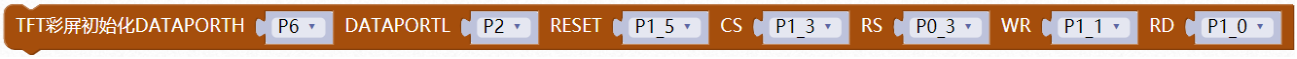
TFT（Thin Film Transistor）即薄膜场效应晶体管，它可以“主动地”对屏幕上的各个独立的像素进行控制，这样可以大大提高反应时间。一般 TFT 的反应时间比较快，约 80 毫秒，而且可视角度大，一般可达到 130 度左右，主要运用在高端产品。从而可以做到高速度、高亮度、高对比度显示屏幕信息。TFT 属于有源矩阵液晶显示器，在技术上采用了“主动式矩阵”的方式来驱动，方法是利用薄膜技术所作成的电晶体电极，利用扫描的方法“主动拉”控制任意一个显示点的开与关，光源照射时先通过下偏光板向上透出，借助液晶分子传导光线，通过遮光和透光来达到显示的目的。

电路原理图



图形化模块

1. 彩屏各引脚初始化。



2. TFT 彩屏清屏并设置背景颜色。

3. 彩屏显示数字，坐标、字体颜色、背景颜色、字体大小、有无叠加设置。

4. 彩屏显示字符串，坐标、字体颜色、背景颜色、字体大小、有无叠加设置。

5. 彩屏显示汉字，坐标、字体颜色、背景颜色、字体大小、有无叠加设置。

6. 颜色选择。

示例代码 1

TFT 屏幕显示字符 a b c d，和汉字好好搭搭。

调用函数代码

```
引入头文件

#include "lib/tftlcd.h"
```

```
预定义 TFT 连接引脚，引脚预处理输入输出模式

#define TFT_LCD_DATAPORTH P6//高 8 位数据口,8 位模式下只使用高 8 位

#define TFT_LCD_DATAPORTH_IN {P6M1=0xff;P6M0=0x00;}//P6 口高阻输入

#define TFT_LCD_DATAPORTH_OUT {P6M1=0x00;P6M0=0xff;}//P6 口推挽输出

#define TFT_LCD_DATAPORTL P2//低 8 位数据口,8 位模式下只使用高 8 位

#define TFT_LCD_RESET P1_5

#define TFT_LCD_RESET_OUT {P1M1&=~0x20;P1M0|=0x20;}//推挽输出

#define TFT_LCD_CS P1_3

#define TFT_LCD_CS_OUT {P1M1&=~0x08;P1M0|=0x08;}//推挽输出

#define TFT_LCD_RS P0_3

#define TFT_LCD_RS_OUT {P0M1&=~0x08;P0M0|=0x08;}//推挽输出

#define TFT_LCD_WR P1_1

#define TFT_LCD_WR_OUT {P1M1&=~0x02;P1M0|=0x02;}//推挽输出

#define TFT_LCD_RD P1_0

#define TFT_LCD_RD_OUT {P1M1&=~0x01;P1M0|=0x01;}//推挽输出
```

```
void tft_lcd_init();    //TFT 初始化，参数无
```

```
void tft_lcd_clear(uint16 color)//LCD 清屏函数，参数: color:清屏的颜色
```

```
uint16 tft_lcd_read_id();   //读取 ID 号
```

```
void tft_lcd_show_string(int16 x,int16 y,uint8 *p,uint16 font_color, uint16 backgrou
nd_color,uint8 size,uint8 mode) //在指定位置显示字符串.参数: x:起始 x 坐标; y:起始 y
坐标; p:要显示的字符串; font_color:字符串的颜色值; background_color:背景色 size:显
示字符的大小（12 或 16）；mode:0-无叠加，1-叠加.
```

```
void tft_lcd_show_font12(uint8 lenth, uint8 *hz, int16 x, int16 y,uint16 font_color, u
int16 background_color, uint8 mode) //描述: 在指定位置显示 12*12 字体汉字.参
数: hz:汉字的指针; x:起始 x 坐标; y:起始 y 坐标; lenth: 字体的总长度 font_color:显示字符
的颜色值; background_color:显示字符的背景色;mode:0-无叠加，1-叠加.
```

```

```

```
void tft_lcd_show_font16(uint8 lenth, uint8 *hz, int16 x, int16 y,uint16 font_color, u
int16 background_color, uint8 mode) // 在指定位置显示 16*16 字体汉字.参数: hz:汉
字的指针; x:起始 x 坐标; y:起始 y 坐标; lenth: 字体的总长度 font_color:显示字符的颜色
值; background_color:显示字符的背景色;mode:0-无叠加，1-叠加.
```

```
void tft_lcd_show_font24(uint8 lenth, uint8 *hz, int16 x, int16 y,uint16 font_color, u
int16 background_color, uint8 mode) //在指定位置显示 24*24 字体汉字.参数: hz:汉字
的指针; x:起始 x 坐标; y:起始 y 坐标; lenth: 字体的总长度 font_color:显示字符的颜色
值; background_color:显示字符的背景色;mode:0-无叠加，1-叠加.
```

```
void tft_lcd_show_font32(uint8 lenth, uint8 *hz, int16 x, int16 y,uint16 font_color, u
int16 background_color, uint8 mode)
// 在指定位置显示 32*32 字体汉字，参数: hz:汉字的指针; x:起始 x 坐标; y:起始 y 坐
标; lenth: 字体的总长度 font_color:显示字符的颜色值; background_color:显示字符的背
景色;mode:0-无叠加，1-叠加.
```

```
void tft_lcd_draw_bmp16(uint16 x,uint16 y,uint16 w,uint16 h,const uint8*p) //显示
16 位的 BMP 图像，参数: x0:起始 x 坐标; y0:起始 y 坐标;w:图片的宽度; h:图片的高度; p:
图像数组的起始地址
```

示例代码 1

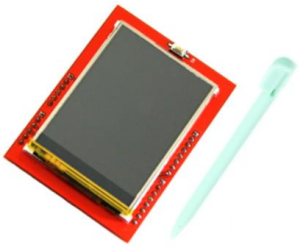
```
#define TFT_LCD_DATAPORTH P6//高 8 位数据口,8 位模式下只使用高 8 位
#define TFT_LCD_DATAPORTH_IN {P6M1=0xff;P6M0=0x00;}//P6 口高阻输入
#define TFT_LCD_DATAPORTH_OUT {P6M1=0x00;P6M0=0xff;}//P6 口推挽输出
#define TFT_LCD_DATAPORTL P2//低 8 位数据口,8 位模式下只使用高 8 位
#define TFT_LCD_RESET P1_5
#define TFT_LCD_RESET_OUT {P1M1&=~0x20;P1M0|=0x20;}//推挽输出
#define TFT_LCD_CS P1_3
#define TFT_LCD_CS_OUT {P1M1&=~0x08;P1M0|=0x08;}//推挽输出
#define TFT_LCD_RS P0_3
#define TFT_LCD_RS_OUT {P0M1&=~0x08;P0M0|=0x08;}//推挽输出
#define TFT_LCD_WR P1_1
#define TFT_LCD_WR_OUT {P1M1&=~0x02;P1M0|=0x02;}//推挽输出
#define TFT_LCD_RD P1_0
#define TFT_LCD_RD_OUT {P1M1&=~0x01;P1M0|=0x01;}//推挽输出


#include <STC8HX.h>
uint32 sys_clk = 24000000;
```

```
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/tftlcd.h"//引入 tftlcd 头文件

void twen_board_init()
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  tft_lcd_init();
```

```
}


void loop()
{
  tft_lcd_show_font12("好好搭搭",100,100,(TFT_LCD_BLACK),(TFT_LCD_WHITE),0);
//显示汉字  好好搭搭
  tft_lcd_show_string(100,50,"a b c d",(TFT_LCD_BLACK),(TFT_LCD_WHITE),12,0);
//显示字符串  a b c d
}


void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

彩屏触摸

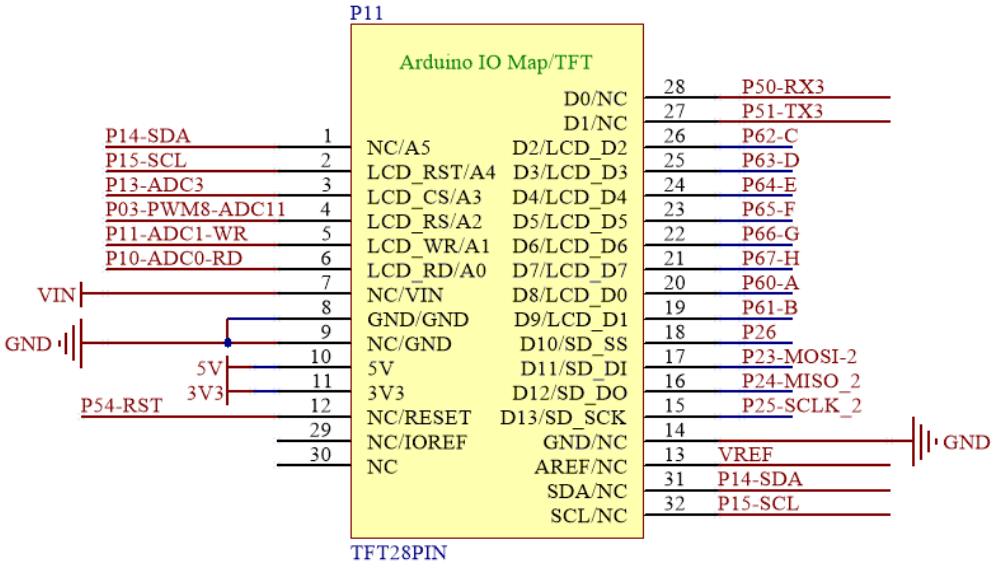
概述



触摸屏（Touch Panel）又称为“触控屏”、“触控面板”，是一种可接收触头等输入讯号的感应式液晶显示装置，当接触了屏幕上的图形按钮时，屏幕上的触觉反馈系统可根据预先编程的程式驱动各种连结装置，可用以取代机械式的按钮面板，并借由液晶显示画面制造出生动的影音效果。

触摸屏作为一种最新的电脑输入设备，它是简单、方便、自然的一种人机交互方式。它赋予了多媒体以崭新的面貌，是极富吸引力的全新多媒体交互设备。主要应用于公共信息的查询、工业控制、军事指挥、电子游戏、多媒体教学等

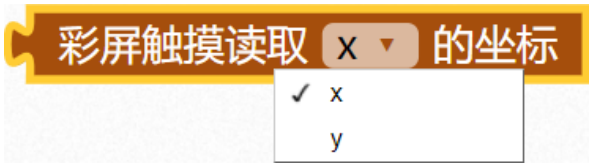
电路原理图



图形化模块

1. 彩屏触摸初始化程序，选择是否需要校准。

2. 彩屏触摸读取 X or Y 的坐标。



3. 彩屏触摸读取压力值。

示例代码 1

彩屏初始化校准，然后可以进行触摸画点。

调用函数代码

```
引入头文件

#include "lib/touch.h"
```

```
void touch_init(uint8 t);

//触摸初始化程序，参数: t:0,不进入校准程序;其他值:进入校准程序
```

```
uint16 touch_read_x();  //读取 x 校准过的坐标
```

```
uint16 touch_read_y();  //读取 y 校准过的坐标
```

```
uint16 touch_read_pressure(); //读取触摸的压力值.
```

示例代码 1

```
#define TFT_LCD_DATAPORTH P6//高 8 位数据口,8 位模式下只使用高 8 位

#define TFT_LCD_DATAPORTH_IN {P6M1=0xff;P6M0=0x00;}//P6 口高阻输入

#define TFT_LCD_DATAPORTH_OUT {P6M1=0x00;P6M0=0xff;}//P6 口推挽输出

#define TFT_LCD_DATAPORTL P2//低 8 位数据口,8 位模式下只使用高 8 位

#define TFT_LCD_RESET P1_5

#define TFT_LCD_RESET_OUT {P1M1&=~0x20;P1M0|=0x20;}//推挽输出

#define TFT_LCD_CS P1_3

#define TFT_LCD_CS_OUT {P1M1&=~0x08;P1M0|=0x08;}//推挽输出

#define TFT_LCD_RS P0_3

#define TFT_LCD_RS_OUT {P0M1&=~0x08;P0M0|=0x08;}//推挽输出

#define TFT_LCD_WR P1_1

#define TFT_LCD_WR_OUT {P1M1&=~0x02;P1M0|=0x02;}//推挽输出

#define TFT_LCD_RD P1_0

#define TFT_LCD_RD_OUT {P1M1&=~0x01;P1M0|=0x01;}//推挽输出



#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"
```

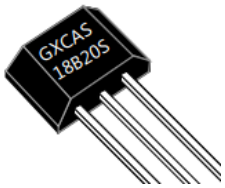
```
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/tftlcd.h"
#include "lib/touch.h"

void twen_board_init()
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  tft_lcd_init();
  touch_init(1);  //彩屏触摸初始化程序
```

```
}


void loop()
{
  tft_lcd_draw_point((touch_read_x()),(touch_read_y()));//画点
}


void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

传感器模块

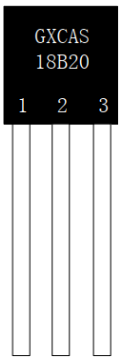
18B20 模块

硬件概述



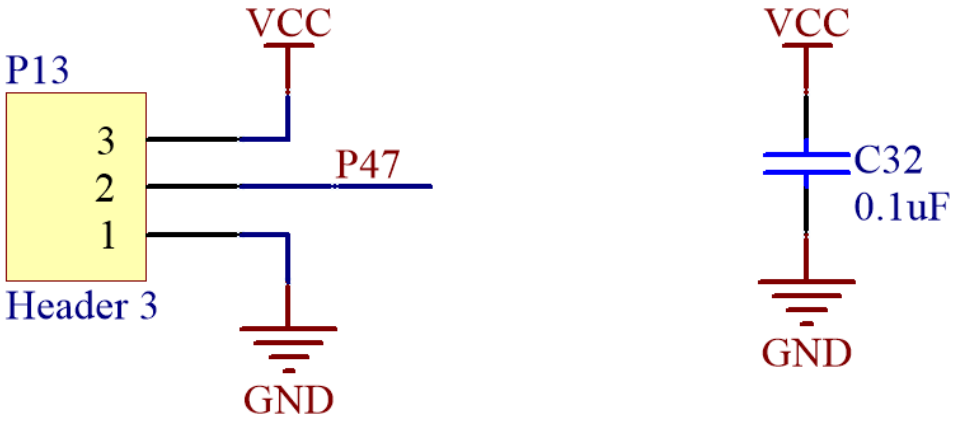
18B20 数字温度计提供 9 到 12bit 分辨率的温度测量，可以通过可编程非易失性存储单元实现温度的下限和上限报警。18B20 采用单总线协议与上位机进行通信，只需要一根信号线和一根地线。它的温度测量范围为-55℃~+125℃。在-10℃~+70℃范围内的测试精度可以达到± 0.4℃。此外它还可以工作在寄生模式下，直接通过信号线对芯片供电，从而不需要额外的供电电源。每个 18B20 都有一个全球唯一的 64 位序列号，可以将多个 18B20 串联在同一跟单总线上进行组网，只需要一个处理器就可以控制分布在大面积区域中的多颗 18B20。这种组网方式特别适合 HVAC 环境控制，建筑、设备、粮情测温和工业测温以及过程监测控制等应用领域。

引脚定义



序号	符号	管脚名	功 能 描 述
1	GND	接地	信号接地和电源接地
2	DQ	数据传输	数据输入输出管脚，当寄生供电模式下，该管脚给芯片供电
3	VDD	电源	供电管脚，在寄生供电模式下 VDD 管脚必须连接到地

电路原理图



图形化模块

1. 初始化 18B20 的控制引脚

2. 设置读取温度

示例代码 1

设置 18B20 读取温度，并用数码管显示。

调用函数代码

```
引入头文件

#include "lib/ds18b20.h"
```

```
预定义 18B20 连接引脚，引脚预处理双向 IO

#define DS18B20_DQ P4_7//18B20 的引脚

#define DS18B20_DQ_MODE {P4M1&=~0x80;P4M0&=~0x80;} //双向 IO 口
```

```
void ds18b20_init()//18B20 初始化函数，参数无
```

```
float ds18b20_read_temperature()//18B20 初始化函数，参数无
```

示例代码 1

```
#define DS18B20_DQ P4_7//18B20 的引脚

#define DS18B20_DQ_MODE {P4M1&=~0x80;P4M0&=~0x80;} //双向 IO 口



#include <STC8HX.h>
```

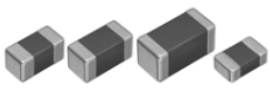
```
uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"

#include "lib/rgb.h"

#include "lib/delay.h"

#include "lib/nixietube.h"

#include "lib/led8.h"

#include "lib/ds18b20.h"//引入 18B20 头文件


void twen_board_init()

{

  hc595_init();

  hc595_disable();

  rgb_init();

  delay(100);

  rgb_show(0,0,0,0);//熄灭 RGB

  delay(100);

}


void Timer0Init(void) //1000 微秒@24.000MHz

{

  TMOD |= 0x00;   //模式 0
```

```
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  nix_scan_callback();//数码管扫描回调函数
}

void setup()
{
  twen_board_init();
  nix_init();//数码管初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  ds18b20_init();//18B20 初始化
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}

void loop()
{
```

```
  nix_display_clear();//数码管清屏
  nix_display_num((ds18b20_read_temperature()));//数码管显示温度值
  delay(1000);
}


void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

NTC 模块

硬件概述



热敏电阻器是敏感元件的一类，按照温度系数不同分为正温度系数热敏电阻器（PTC）和负温度系数热敏电阻器（NTC）。热敏电阻器的典型特点是对温度敏感，不同的温度下表现出不同的电阻值。正温度系数热敏电阻器（PTC）在温度越高时电阻值越大，负温度系数热敏电阻器（NTC）在温度越高时电阻值越低，它们同属于半导体器件。

热敏电阻将长期处于不动作状态；当环境温度和电流处于 c 区时，热敏电阻的散热功率与发热功率接近，因而可能动作也可能不动作。热敏电阻在环境温度相同时，动作时间随着电流的增加而急剧缩短；热敏电阻在环境温度相对较高时具有更短的动作时间和较小的维持

电流及动作电流。

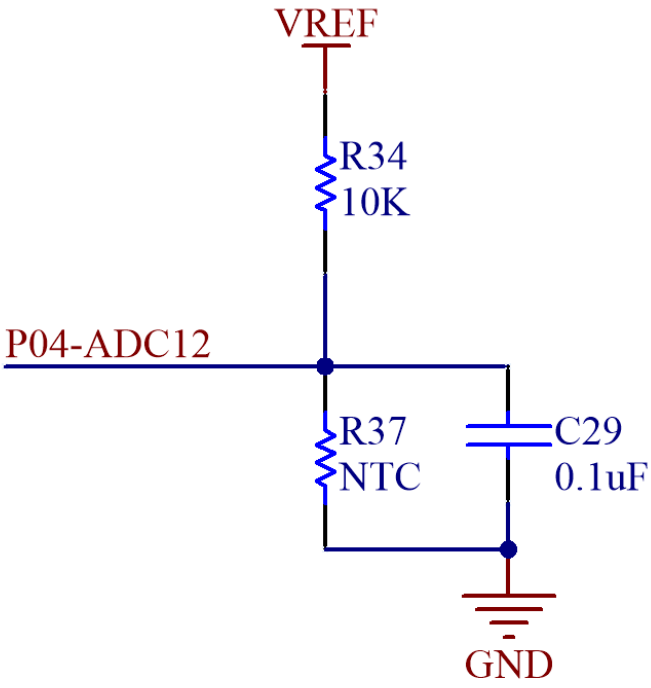
引脚定义

序号	符号	管脚名	功 能 描 述
1	①	端电极	焊接固定
2	②	端电极	焊接固定

温度与电阻值对照表

温度 Temp. (℃)	R 最小值 R_Min (Kohm)	R 中心值 R_Cent (Kohm)	R 最大值 R_Max (Kohm)	阻值公差 Res TOL.	温度公差 Temp. TOL.(℃)
2	23.265	25.177	27.178	7.95%	1.83
3	22.320	24.124	26.008	7.81%	1.81
4	21.420	23.121	24.895	7.67%	1.79
5	20.560	22.165	23.836	7.54%	1.77
6	19.739	21.253	22.826	7.40%	1.75
7	18.955	20.384	21.865	7.27%	1.73
8	18.207	19.555	20.950	7.13%	1.71
9	17.492	18.764	20.078	7.00%	1.68
10	16.810	18.010	19.247	6.87%	1.66
11	16.158	17.290	18.455	6.74%	1.64
12	15.534	16.602	17.699	6.61%	1.62
13	14.938	15.946	16.979	6.48%	1.60
14	14.368	15.319	16.292	6.35%	1.58
15	13.823	14.720	15.636	6.22%	1.56
16	13.301	14.148	15.011	6.10%	1.53
17	12.802	13.601	14.413	5.97%	1.51
18	12.324	13.078	13.843	5.85%	1.49
19	11.867	12.578	13.298	5.72%	1.47
20	11.429	12.099	12.777	5.60%	1.45
21	11.010	11.642	12.280	5.48%	1.42
22	10.608	11.204	11.804	5.36%	1.40
23	10.223	10.785	11.350	5.24%	1.38
24	9.854	10.384	10.916	5.12%	1.35
25	9.500	10.000	10.500	5.00%	1.33
26	9.140	9.632	10.125	5.12%	1.37
27	8.796	9.280	9.766	5.24%	1.41
28	8.467	8.943	9.421	5.35%	1.45
29	8.152	8.619	9.090	5.47%	1.49
30	7.850	8.309	8.773	5.59%	1.53
31	7.561	8.012	8.468	5.70%	1.57
32	7.284	7.727	8.176	5.82%	1.62
33	7.018	7.453	7.895	5.93%	1.66
34	6.764	7.191	7.626	6.04%	1.70
35	6.520	6.939	7.366	6.16%	1.74
36	6.287	6.698	7.118	6.27%	1.78
37	6.063	6.466	6.878	6.38%	1.83
38	5.848	6.243	6.649	6.49%	1.87
39	5.642	6.029	6.428	6.61%	1.91
40	5.444	5.824	6.215	6.72%	1.96
41	5.254	5.627	6.011	6.83%	2.00
42	5.072	5.437	5.814	6.94%	2.05
43	4.897	5.255	5.625	7.05%	2.09
44	4.729	5.080	5.443	7.16%	2.14
45	4.567	4.911	5.268	7.26%	2.18

电路原理图



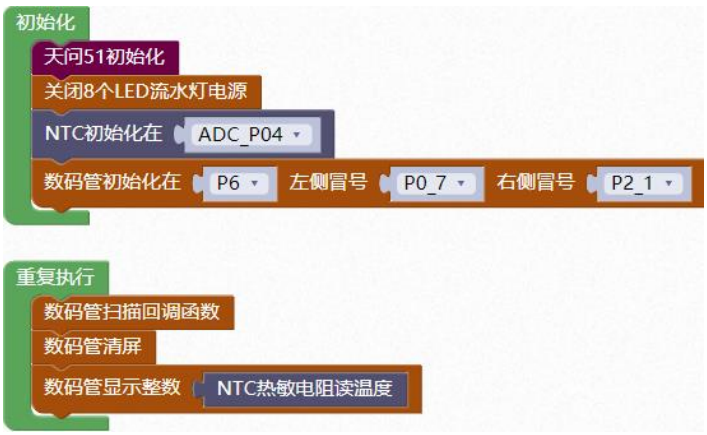
图形化模块

1. 初始化 NTC 的控制引脚

2. 读取 NTC 温度

示例代码 1

设置 NTC 读取温度，并用数码管显示。



调用函数代码

```
引入头文件

#include "lib/ntc.h"
```

```
预定义 RGB 灯连接引脚

#define NTC_ADC_PIN ADC_P04
```

```
void ntc_init()//NTC 初始化函数，参数无
```

```
float ntc_read_temp()//NTC 读取温度函数，参数无
```

示例代码 1

```
#define NTC_ADC_PIN ADC_P04//NTC 的引脚

#define NIXIETUBE_PORT P6

#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出

#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号

#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽
输出

#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号

#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推
挽输出



#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认
```

```
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/ntc.h"//引用 ntc 头文件
#include "lib/nixietube.h"

void twen_board_init()
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  ntc_init();//NTC 热敏电阻测量初始化
```

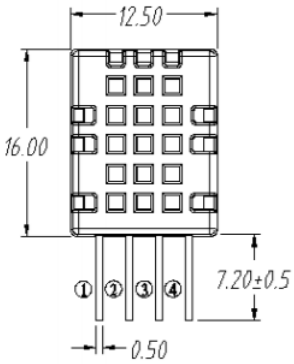
```
  nix_init();//数码管初始化
}

void loop()
{
  nix_scan_callback();//数码管扫描回调函数
  nix_display_clear();//数码管清屏
  nix_display_num((ntc_read_temp()));//数码管显示 ntc  温度
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

DHT11 模块

硬件概述

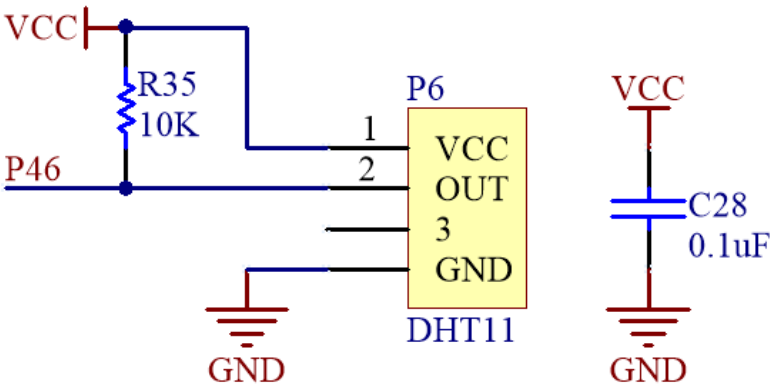
DHT11 数字温湿度传感器，是一款含有已校准数字信号输出的温湿度复合传感器，它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的可靠性和卓越的长期稳定性。传感器包括一个电阻式感湿元件和一个 NTC 测温元件，并与一个高性能 8 位单片机相连接。因此该产品具有品质卓越、超快响应、抗干扰能力强、性价比极高等优点。每个 DHT11 传感器都在极为精确的湿度校验室中进行校准。校准系数以程序的形式存在 OTP 内存中，传感器内部在检测信号的处理过程中要调用这些校准系数。单线制串行接口，使系统集成变得简易快捷。超小的体积、极低的功耗，使其成为该类应用中，在苛刻应用场合的最佳选择。产品为 4 针单排引脚封装，连接方便。

引脚定义



序号	符号	管脚名	功 能 描 述
1	VDD	电源	供电管脚
2	DATA	数据传输	串行数据，单总线
3	NC	悬空	无
4	GND	接地	信号接地和电源接地

电路原理图



图形化模块

1. 初始化 DHT11 的控制引脚

2. 设置读取温度

3. 设置读取湿度

示例代码 1

设置 DHT11 读取温度，并用数码管显示。

示例代码 2

设置 DHT11 读取湿度，并用数码管显示。

调用函数代码

```
引入头文件

#include "lib/dht11.h"
```

```
预定义 DHT11 连接引脚，引脚预处理双向 IO

#define DHT11_DQ P4_6//DHT11 的引脚

#define DHT11_DQ_MODE {P4M1&=~0x40;P4M0&=~0x40;}//双向 IO 口
```

```
uint8 dht11_init()//DHT11 初始化函数，参数无
```

```
uint8 dht11_read_humidity()//DHT11 读取湿度函数，参数无
```

```
float dht11_read_temp()//DHT11 读取温度函数，参数无
```

示例代码 1

```
#define DHT11_DQ P4_6//DHT11 的引脚

#define DHT11_DQ_MODE {P4M1&=~0x40;P4M0&=~0x40;}//P4_6 双向 IO 口
```

```
#define NIXIETUBE_PORT P6

#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出

#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号

#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽
输出

#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号

#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推
挽输出


#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"

#include "lib/rgb.h"

#include "lib/delay.h"

#include "lib/led8.h"

#include "lib/dht11.h"//引入 DHT11 头文件

#include "lib/nixietube.h"


void twen_board_init()

{

  hc595_init();//HC595 初始化
```

```
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void Timer0Init(void) //1000 微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  nix_scan_callback();//数码管扫描回调函数
}

void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
```

```
  dht11_init();
  nix_init();//数码管初始化
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}

void loop()
{
  nix_display_num((dht11_read_temp()));//数码管显示 DHT11 温度
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

示例代码 2

```
#define DHT11_DQ P4_6//DHT11 的引脚
#define DHT11_DQ_MODE {P4M1&=~0x40;P4M0&=~0x40;}//P4_6 双向 IO 口
#define NIXIETUBE_PORT P6
#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出
#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号
#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽
输出
#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号
#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推
挽输出


#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/dht11.h"//引入 DHT11 头文件
#include "lib/nixietube.h"


void twen_board_init()
```

```
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void Timer0Init(void) //1000 微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  nix_scan_callback();//数码管扫描回调函数
}

void setup()
{
```

```
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  dht11_init();
  nix_init();//数码管初始化
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}

void loop()
{
  nix_display_num((dht11_read_humidity()));//数码管显示 DHT11 湿度
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

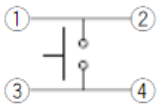
矩阵键盘模块

硬件概述

矩阵键盘是单片机外部设备中所使用的排布类似于矩阵的键盘组。矩阵式结构的键盘显然比直接法要复杂一些，识别也要复杂一些，列线通过电阻接正电源，并将行线所接的单片机的 I/O 口作为输出端，而列线所接的 I/O 口则作为输入。

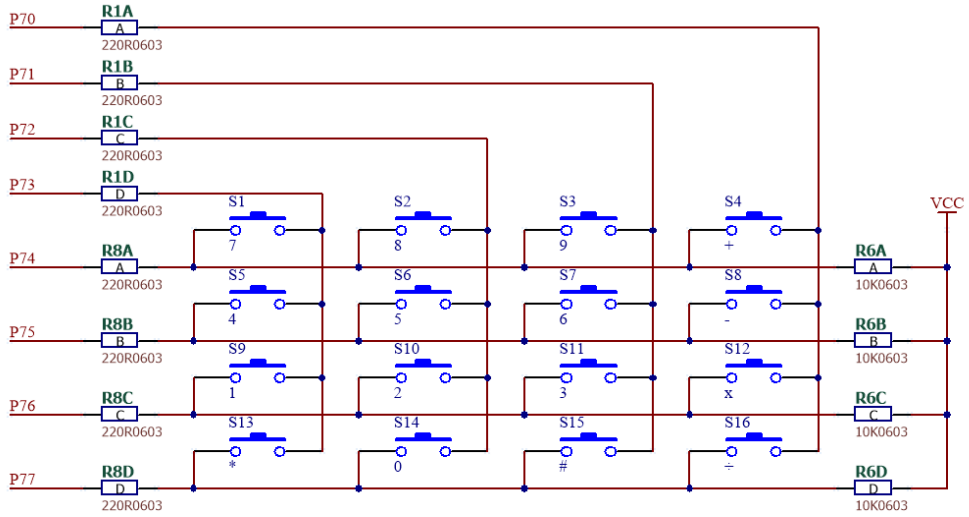
在键盘中按键数量较多时，为了减少 I/O 口的占用，通常将按键排列成矩阵形式。在矩阵式键盘中，每条水平线和垂直线在交叉处不直接连通，而是通过一个按键加以连接。这样，一个端口（如 P7 口）就可以构成 4*4=16 个按键，比之直接将端口线用于键盘多出了一倍，而且线数越多，区别越明显，比如再多加一条线就可以构成 20 键的键盘，而直接用端口线则只能多出一键（9 键）。由此可见，在需要的键数比较多时，采用矩阵法来做键盘是合理的。

引脚定义



序号	符号	功 能 描 述
1	①	与②脚相连
2	②	与①脚相连
3	③	与④脚相连
4	④	与③脚相连

电路原理图

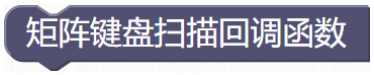


图形化模块

1. 矩阵键盘初始化

2. 矩阵键盘获取按键值

3. 矩阵键盘扫描回调函数



示例代码 1

数码管显示矩阵键盘按键值。

调用函数代码

```
引入头文件

#include "lib/keypad.h"
```

```
void keypad_init()//矩阵键盘初始化函数，参数无
```

```
int8 keypad_get_value();//获取按键值，参数无
```

```
void io_key_scan();  //按键扫描函数，参数无
```

示例代码 1

```
#define NIXIETUBE_PORT P6

#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出

#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号

#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽
输出

#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号

#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推
挽输出


#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"
```

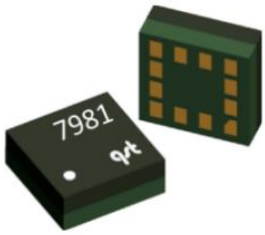
```
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/keypad.h"//引用 矩阵按键 头文件
#include "lib/led8.h"
#include "lib/nixietube.h"

uint32 systick = 0;

void twen_board_init()
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void Timer0Init(void) //1000 微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
```

```
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  systick = systick + 1;
  nix_scan_callback();//数码管扫描回调函数
  if(systick % 20 == 0){
    io_key_scan();//矩阵键盘扫描回调函数
  }
}

void setup()
{
  twen_board_init();//天问 51 初始化
  keypad_init();
  led8_disable();//关闭 8 个 LED 流水灯电源
  nix_init();//数码管初始化
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}
```

```
void loop()
{
  nix_display_clear();//数码管清屏
  nix_display_num((keypad_get_value()));//数码管显示按键值
  delay(100);
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

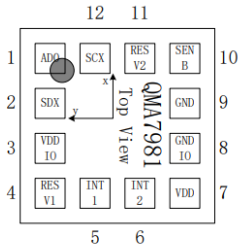
QMA7981 加速度模块

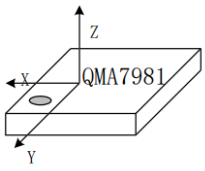
硬件概述



QMA7981 是一个单芯片三轴加速度传感器，14-Bit ADC 采样精度，内置常用运动算法，提供标准 I2C/SPI 接口，支持低功耗模式，广泛应用与手机、运动手表等设备。

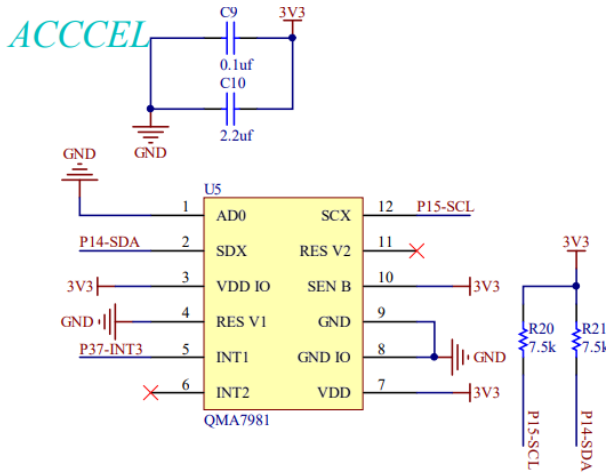
引脚定义





序号	符号	功 能 描 述
1	AD0	I2C 地址设置引脚
2	SDX	传输数据
3	VDDIO	数字电路电源
4	RESV1	保留
5	INT1	中断引脚
6	INT2	中断引脚
7	VDD	电源
8	GNDIO	数字电路地
9	GND	地
10	SENB	通讯协议选择
11	RESV2	保留
12	SCL	时钟信号

电路原理图



图形化模块

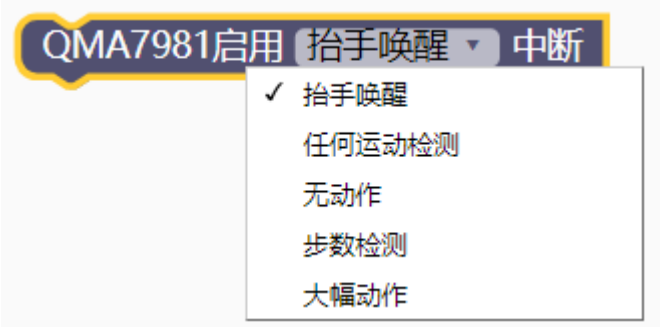
1. QMA7981 初始化。

2. QMA7981 刷新数据

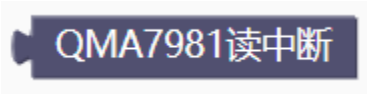
3. QMA7981 读 X/Y/Z 轴加速度

4. QMA7981 获取步数

5. QMA7981 启用运动检测算法中断



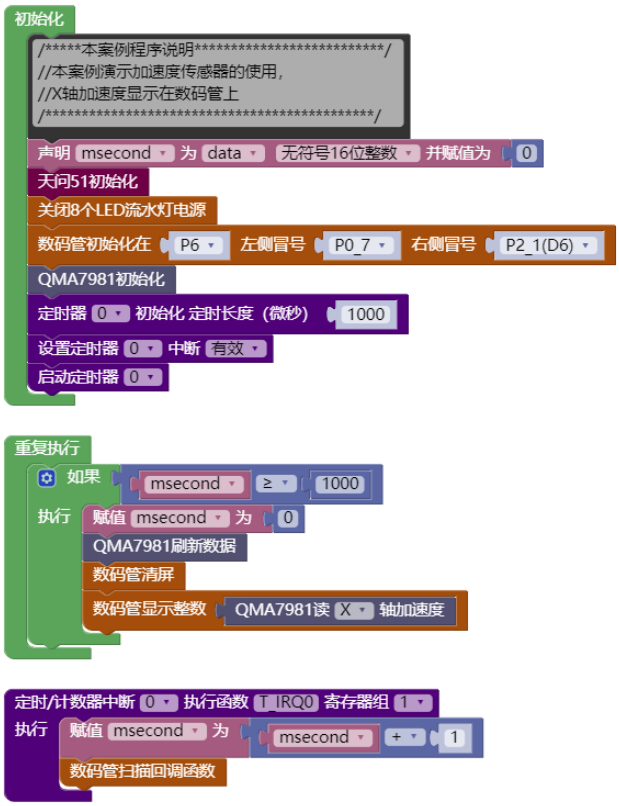
6. QMA7981 读中断



7. QMA7981 判断中断类型

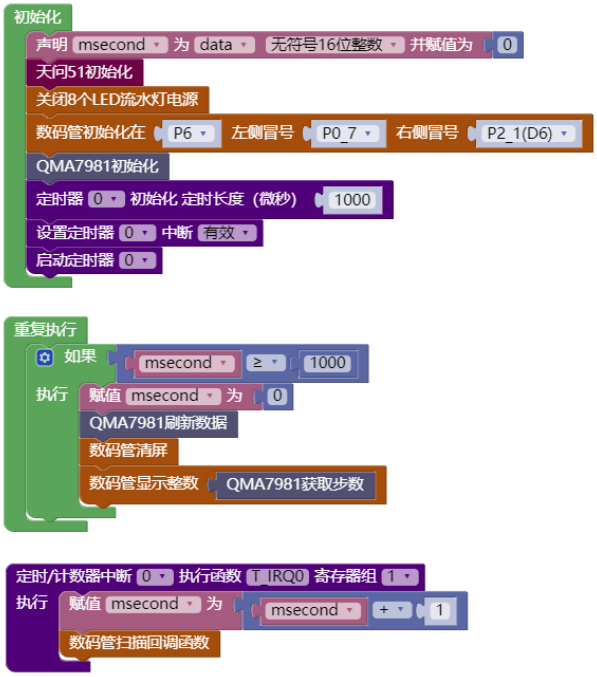
示例代码 1

在数码管上显示 X 轴加速度



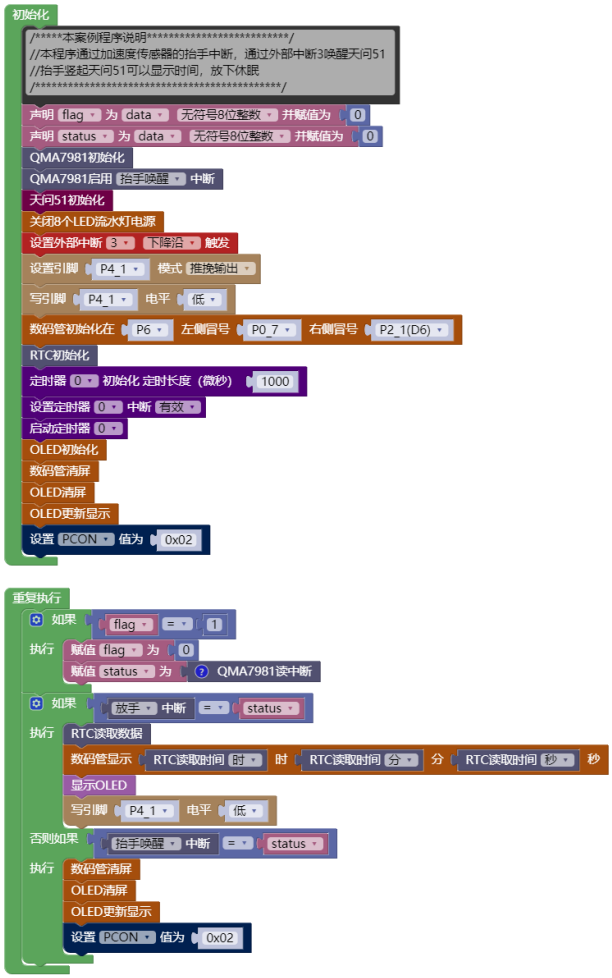
示例代码 2

在数码管上显示步数



示例代码 3

运动手表，抬手唤醒功能。




```
//引入头文件
#include "lib/qma7981.h"
```

```
//使能抬手唤醒中断
#define QMA7981_HAND_UP_DOWN
//使能任何运动检测中断
#define QMA7981_ANY_MOTION
//使能无动作中断
#define QMA7981_NO_MOTION
//使能步数检测中断
#define QMA7981_STEP_INT
//使能重大中断（用户位置变化而产生的运动）
#define QMA7981_SIGNIFICANT_MOTION
```

```
uint8 qma7981_init();//加速度初始化
void qma7981_set_range(uint8 range);//设置加速度范
围 QMA7981_RANGE_2G/4G/8G/16G/32G.
void qma7981_read_acc(int32 *accData);//读加速度三轴值
uint32 qma7981_read_stepcounter();//获取步数
unsigned char qma7981_irq_hdlr();//读中断
void qma7981_raise_config(uint8 wake_sum, uint8 diff, uint16 period, ui
nt16 timeout, uint8 hd_z, uint8 hd_x);    //抬手唤醒配置函数
```

示例代码 1

在数码管上显示 X 轴加速度

```
#define NIXIETUBE_PORT P6
#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出
#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号
#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽输
出
#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号
#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推挽输
出

#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/nixietube.h"
#include "lib/qma7981.h"

uint16 msecond = 0;
int32 _accData[3];

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void Timer0Init(void) //1000 微秒@24.000MHz
{
```

```
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  msecond = msecond + 1;
  nix_scan_callback();//数码管扫描回调函数
}

void setup()
{
  /*****本案例程序说明*************************/
  //本案例演示加速度传感器的使用，
  //X 轴加速度显示在数码管上
  /*******************************************/
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  nix_init();//数码管初始化
  qma7981_init();
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}

void loop()
{
  if(msecond >= 1000){
    msecond = 0;
    qma7981_read_acc(_accData);//QMA7981 读三轴加速度加速度值
    nix_display_clear();//数码管清屏
    nix_display_num((_accData[0]));//数码管显示整数
  }
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

示例代码 2
在数码管上显示步数

```
#define NIXIETUBE_PORT P6
#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出
#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号
#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽输
出
#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号
#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推挽输
出
#define QMA7981_STEPCOUNTER 1

#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/nixietube.h"
#include "lib/qma7981.h"

uint16 msecond = 0;
int32 _accData[3];

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}
```

```
void Timer0Init(void) //1000微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  msecond = msecond + 1;
  nix_scan_callback();//数码管扫描回调函数
}

void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  nix_init();//数码管初始化
  qma7981_init();
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}

void loop()
{
  if(msecond >= 1000){
    msecond = 0;
    qma7981_read_acc(_accData);//QMA7981 读三轴加速度加速度值
    nix_display_clear();//数码管清屏
    nix_display_num((qma7981_read_stepcounter()));//数码管显示整数
  }
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

示例代码 3

运动手表，抬手唤醒功能。

```
#define QMA7981_HAND_UP_DOWN
#define NIXIETUBE_PORT P6
#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出
#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号
#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽输
出
#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号
#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推挽输
出

#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/qma7981.h"
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/nixietube.h"
#include "lib/pcf8563.h"
#include "lib/oled.h"

uint8 flag = 0;
uint8 status = 0;
struct pcf8563_Time _mytime;
void _E6_98_BE_E7_A4_BAOLED();

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
```

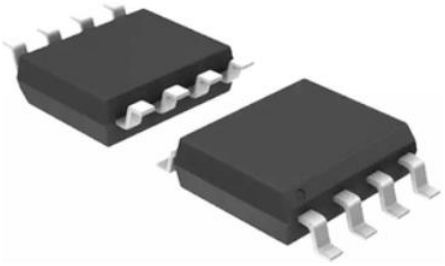
```
  delay(10);
}

void Timer0Init(void) //1000微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

/*描述该功能...
*/
void _E6_98_BE_E7_A4_BAOLED(){
  oled_clear();//OLED清屏
  oled_show_num(12,4,_mytime.year);
  oled_show_font16("年",42,0);
  oled_show_num(58,4,_mytime.month);
  oled_show_font16("月",74,0);
  oled_show_num(90,4,_mytime.day);
  oled_show_font16("日",106,0);
  oled_show_num(20,22,_mytime.hour);
  oled_show_font12("时",36,20);
  oled_show_num(50,22,_mytime.minute);
  oled_show_font12("分",66,20);
  oled_show_num(80,22,_mytime.second);
  oled_show_font12("秒",96,20);
  oled_display();//OLED更新显示
}

void T_IRQ0(void) interrupt 1 using 1{
  nix_scan_callback();//数码管扫描回调函数
}

void INT3(void) interrupt 11 using 1{
  AUXINTIF &= ~0x20;
  P4_1 = 0;
  flag = 1;
}

void setup()
{
  /*****本案例程序说明*************************/
  //本程序通过加速度传感器的抬手中断，通过外部中断3唤醒天问51
  //抬手竖起天问51可以显示时间，放下休眠
```

```
  /*****************************************/
  qma7981_init();
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  INTCLKO |= 0x20;
  EA = 1;
  P4M1&=~0x02;P4M0|=0x02;//推挽输出
  P4_1 = 0;
  nix_init();//数码管初始化
  pcf8563_init();    //RTC 初始化
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
  oled_init();//OLED 初始化
  nix_display_clear();//数码管清屏
  oled_clear();//OLED 清屏
  oled_display();//OLED 更新显示
  PCON = 0x02;
}

void loop()
{
  if(flag == 1){
    flag = 0;
    // 读中断，清除中断标志位
    status = qma7981_irq_hdlr();
  }
  if(7 == status){
    pcf8563_read_rtc(&_mytime);
    nix_display_time2(_mytime.hour,_mytime.minute,_mytime.second);//数码
管显示时间
    _E6_98_BE_E7_A4_BAOLED();
    P4_1 = 0;
  }
  else if(6 == status){
    nix_display_clear();//数码管清屏
    oled_clear();//OLED 清屏
    oled_display();//OLED 更新显示
    PCON = 0x02;
  }
}

void main(void)
```

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

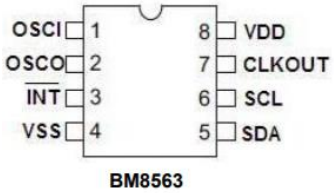
RTC 实时时钟

硬件概述



BM8563 是一款低功耗 CMOS 实时时钟/日历芯片，它提供一个可编程的时钟输出，一个中断输出和一个掉电检测器，所有的地址和数据都通过 I2C 总线接口串行传递。最大总线速度为 400Kbits/s，每次读写数据后，内嵌的字地址寄存器会自动递增。

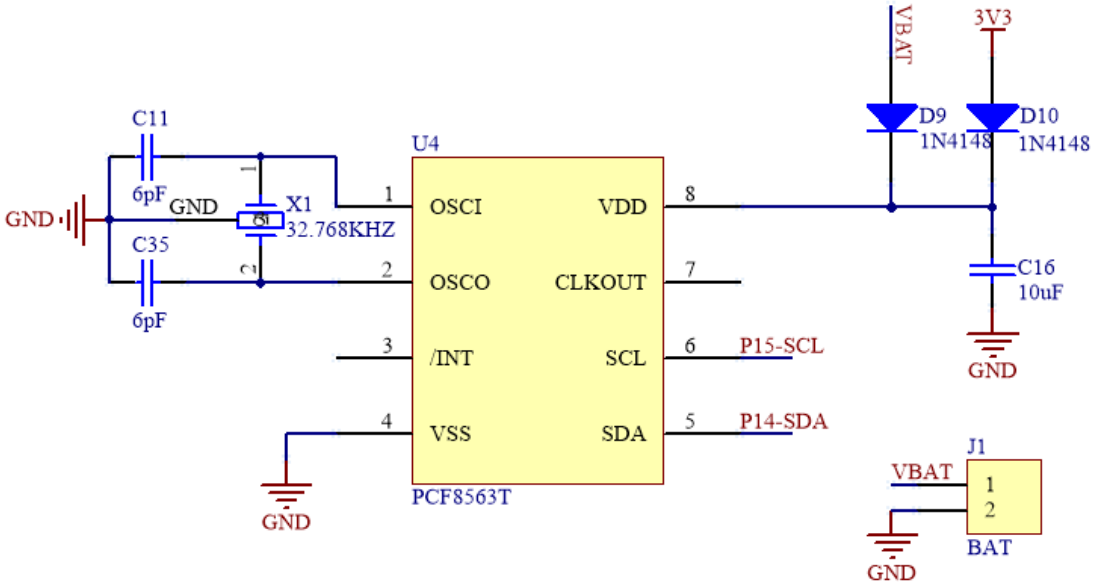
引脚定义



序号	符号	管脚名	功 能 描 述
1	OSCI	输入	振荡器输入
2	OSCO	输出	振荡器输出
3	/INT	中断输出	中断开漏输出
4	VSS	地	信号接地和电源接地
5	SDA	串行数据 I/O	串行数据 I/O（开漏）
6	SCL	串行时钟输入	串行时钟输入
7	CLKOUT	时钟输出	时钟开漏输出

8	VDD	正电源	供电管脚

电路原理图



图形化模块

1. RTC 初始化。

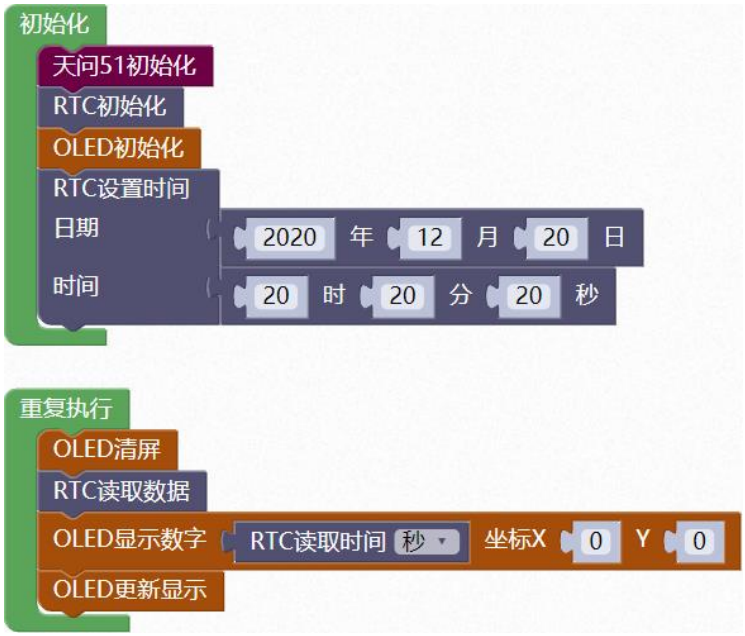
2. RTC 读取数据。

3. RTC 读取时间选取，可选择年、月、日、周、时、分、秒。

4. RTC 设置时间和日期。

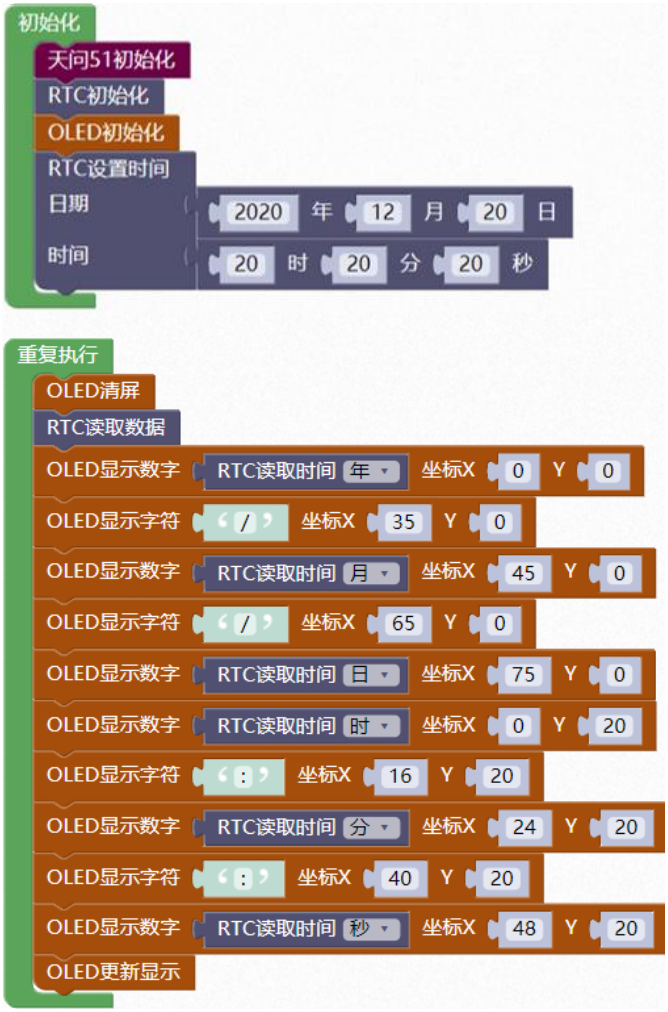
示例代码 1

初始化 OLED 和 RTC，设置好日期和时间。用 OLED 显示读取到的秒。



示例代码 2

初始化 OLED 和 RTC，设置好日期和时间。用 OLED 显示读取到的日期和时间。



调用函数代码

```
引入头文件

#include "lib/pcf8563.h"
```

```
void pcf8563_init();     //RTC 初始化
void pcf8563_read_rtc(struct pcf8563_Time *tim);// 设置时间
void pcf8563_write_rtc(struct pcf8563_Time *tim);// 读取时间
```

示例代码 1

```
#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"
```

```
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/oled.h"
#include "lib/pcf8563.h"//引入 pcf8563 头文件

struct pcf8563_Time _mytime;

void twen_board_init()
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  oled_init();//OLED 初始化
  pcf8563_init();    //RTC 初始化
```

```
  _mytime.year = 2020;
  _mytime.month = 12;
  _mytime.day = 20;
  _mytime.hour = 20;
  _mytime.minute = 20;
  _mytime.second = 20;
  pcf8563_write_rtc(&_mytime);   //RTC 设置时间
}

void loop()
{
  oled_clear();//OLED 清屏
  pcf8563_read_rtc(&_mytime);
  oled_show_num(0,0,_mytime.second);
  oled_display();//OLED 更新显示
}

void main(void)
{
  setup();
  while(1){
    loop();
```

```
  }

}
```

示例代码 2

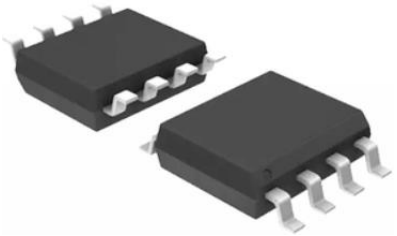
```
#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/hc595.h"

#include "lib/rgb.h"

#include "lib/delay.h"

#include "lib/oled.h"

#include "lib/pcf8563.h"



struct pcf8563_Time _mytime;



void twen_board_init()

{

  hc595_init();//HC595 初始化

  hc595_disable();//HC595 禁止点阵和数码管输出

  rgb_init();//RGB 初始化

  delay(10);

  rgb_show(0,0,0,0);//关闭 RGB

  delay(10);

}
```

```
void setup()
{
  twen_board_init();//天问 51 初始化
  oled_init();//OLED 初始化
  pcf8563_init();    //RTC 初始化
  _mytime.year = 2020;
  _mytime.month = 12;
  _mytime.day = 20;
  _mytime.hour = 20;
  _mytime.minute = 20;
  _mytime.second = 20;
  pcf8563_write_rtc(&_mytime);   //RTC 设置时间
}

void loop()
{
  oled_clear();//OLED 清屏
  pcf8563_read_rtc(&_mytime);
  oled_show_num(0,0,_mytime.year);
  oled_show_char(35,0,'/');
  oled_show_num(45,0,_mytime.month);
```

```
  oled_show_char(65,0,'/');
  oled_show_num(75,0,_mytime.day);
  oled_show_num(0,20,_mytime.hour);
  oled_show_char(16,20,':');
  oled_show_num(24,20,_mytime.minute);
  oled_show_char(40,20,':');
  oled_show_num(48,20,_mytime.second);
  oled_display();//OLED 更新显示
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

FLASH 模块

硬件概述

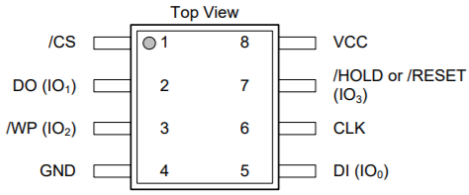


W25Q32JV 有 16384 个可编程页，每个页有 256 个字节。最多 256 字节可以一次被编程。页可以按 16 组(4KB 扇区擦除)、128 组擦除(32KB 块擦除)、256 组(64KB 块擦除)或整个芯片(芯片擦除)。W25Q32JV 已经分别有 1024 个可擦扇区和 64 个可擦块。小的 4KB 部门允许更大的需要数据和参数存储的应用程序的灵活性。

W25Q32JV 支持标准串行外设接口(SPI)和高性能双/四输出，以及双/四 I/O SPI：串行时钟，芯片选择，串行数据 I/O0 (DI)，I/O1 (DO)，I/O2，和 I/O3。SPI 时钟频率高达 133MHz 支持允许等效时钟率 266MHz(133MHz x 2)用于双 I/O 和 532MHz (133MHz x 4)用于四次 I/O 时，使用快速读双/四次 I / O 指令。这些传输速率可以超过标准的异步 8 位和 16 位并行 Flash 记忆。

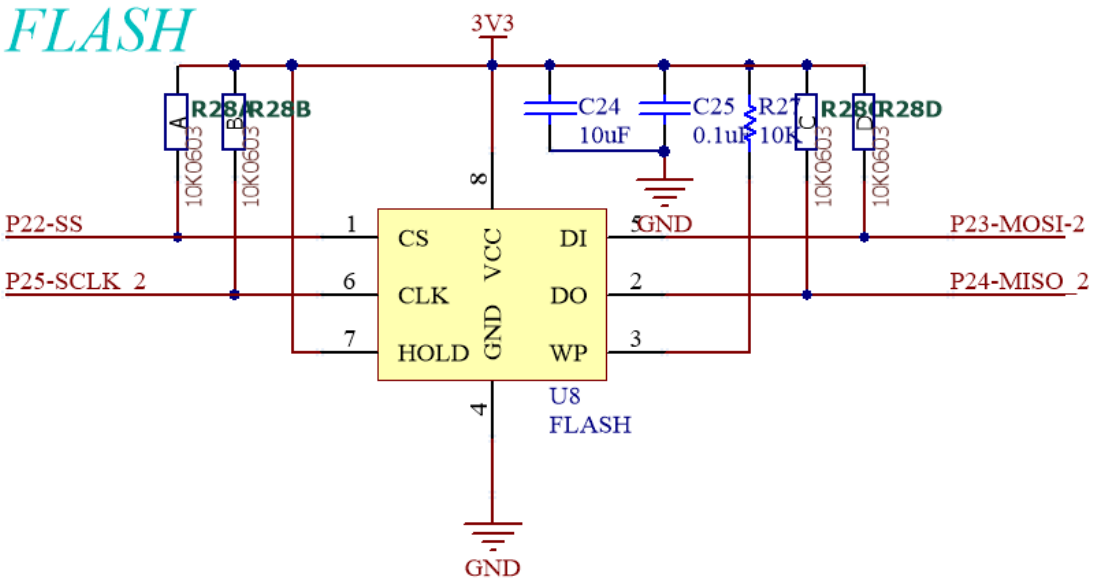
此外，该设备支持 JEDEC 标准制造商和设备 ID 和 SFDP 寄存器，一个 64 位唯一序列号和三个 256 字节安全寄存器。

引脚定义



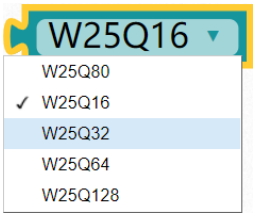
序号	符号	管脚名	功 能 描 述
1	/CS	输入	芯片选择输入
2	DO (IO_1)	输出	数据输出(数据输入输出 1)
3	/WP(IO_2)	写保护	写入保护输入(数据输入输出 2)
4	GND	接地	信号接地和电源接地
5	DI(IO_0)	输入	数据输入(数据输入输出 0)
6	CLK	时钟输入	串行时钟输入
7	/HOLD OR RESET(IO_3)	保持或复位	保持或复位输入(数据输入输出 3)
8	VCC	电源	供电管脚

电路原理图



图形化模块

1. 芯片型号，配合读取芯片 ID 模块一起使用。



2. Flash 初始化引脚。

3. Flash 读取芯片 ID。

4. 把 buf 的数据写入到 Flash 中。

5. 读取 Flash 数据到 buf。

6. Flash 擦除指定扇区。

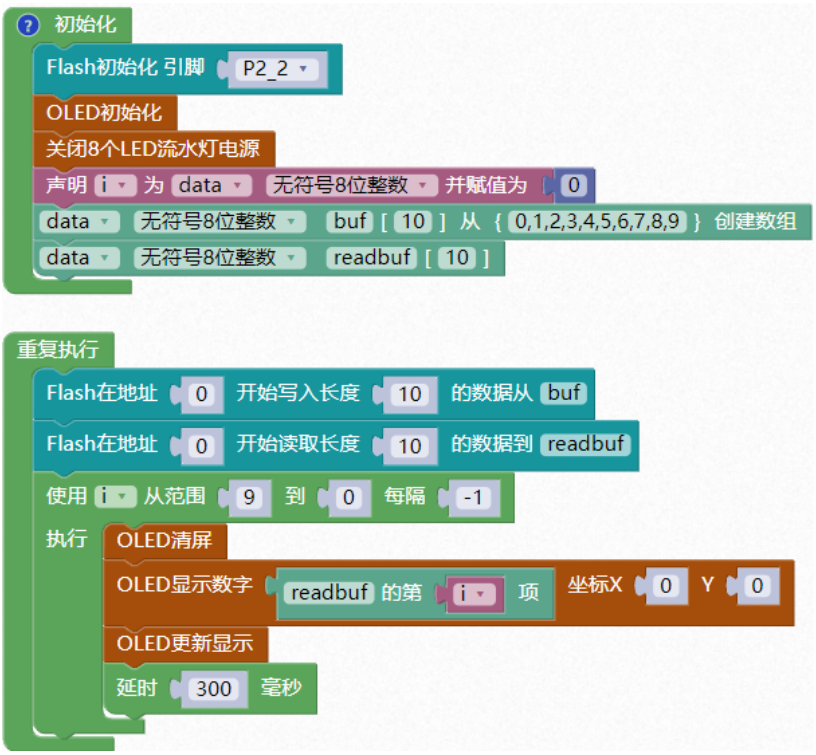
7. Flash 全片擦除。

8. Flash 进入掉电模式。

9. Flash 唤醒。

示例代码 1

将 buf 的数据写入到 Flash 当中，然后再将写入的数据读取到 readbuf 当中，并且用 oled 显示。



调用函数代码

```
引入头文件

#include "lib/w25qxx.h"
```

```
预定义 W25QXX 的片选信号灯连接引脚，引脚预处理输出

#define W25QXX_CS P2_2//W25QXX 的片选信号
```

```
#define W25QXX_CS_MODE {P2M1&=~0x04;P2M0|=0x04;}//P2_2 推挽输出
```

```
void w25qxx_init(); //w25qxx 初始化函数，参数无
```

```
uint16 w25qxx_read_id();    //读取芯片 ID  函数，参数无
```

```
void w25qxx_read(uint8* pBuffer,uint32 ReadAddr,uint16 NumByteToRead);
//在指定地址开始读取指定长度的数据，参数：数据地址，长度，读取数据
```

```
uint16 w25qxx_read_id()// 描述: 读取芯片 ID，参数无
```

```
void w25QXX_Erase_Chip()// 描述: 全片擦除，需等待很长时间，参数无
```

```
void w25qxx_erase_sector(uint32 Dst_Addr)
// 描述: 擦除一个扇区，参数: Dst_Addr:扇区地址(4K 一个扇区，2M FLASH 扇区最大为
512)。
```

```
void w25qxx_power_down()   //   w25qxx 进入掉电模式.
```

```
void w25qxx_wake_up()   // 描述: 唤醒 w25qxx.
```

```
void w25qxx_write_page(uint8* pBuffer,uint32 WriteAddr,uint16 NumByteToWrite)
```

```
// 在指定地址开始写入最大 256 字节的数据. 参数：数据地址，长度，读取数据
```

```
void w25qxx_write_nocheck(uint8* pBuffer,uint32 WriteAddr,uint16 NumByteToWr
ite)   // 描述: 无检验写 spiflash.必须确保所写的地址范围内的数据全部为 0XFF.确保地址
不越界。
参数: pBuffer:数据存储区; WriteAddr:开始写入的地址(24bit)；NumByteToWrite:要写入
的字节数(最大 65535) .
```

```
void w25qxx_write(uint8* pBuffer,uint32 WriteAddr,uint16 NumByteToWrite
// 在指定地址开始写入指定长度的数据，参数: pBuffer:数据存储区; WriteAddr:开始写入
的地址(24bit); NumByteToWrite:要写入的字节数(最大 65535) 。
```

示例代码 1

```
#define W25QXX_CS P2_2//W25QXX 的片选信号
#define W25QXX_CS_MODE {P2M1&=~0x04;P2M0|=0x04;}//P2_2 推挽输出


#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/w25qxx.h"//引入  w25qxx.h  头文件
#include "lib/oled.h"
#include "lib/led8.h"
```

```
#include "lib/delay.h"


uint8 i = 0;
uint8 buf[10]={0,1,2,3,4,5,6,7,8,9};//自定义数组


uint8 readbuf[10]; //自定义数组


void setup()
{
  w25qxx_init();  //w25qxx 初始化
  oled_init();//OLED 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
}


void loop()
{
  w25qxx_write(buf,0,10);  //在指定地址开始写入指定长度的数据
  w25qxx_read(readbuf,0,10);  //在指定地址开始读取指定长度的数据
  for (i = 9; i > 0; i = i + (-1)) {
    oled_clear();//OLED 清屏
    oled_show_num(0,0,readbuf[(int)(i)]);
    oled_display();//OLED 更新显示
```

```
    delay(300);
  }
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

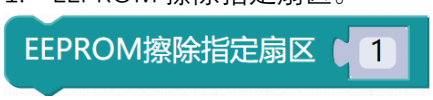
EEPROM

概述

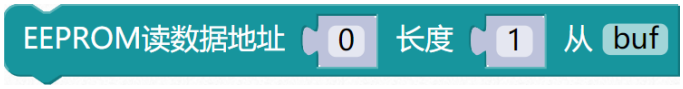
STC8H 系列单片机内部集成了大容量的 EEPROM。利用 ISP/IAP 技术可将内部 Data Flash 当 EEPROM，擦写次数在 10 万次以上。 EEPROM 可分为若干个扇区，每个扇区包含 512 字节。使用时，建议同一次修改的数据放在同一个扇区，不是同一次修改的数据放在不同的扇区，不一定要用满。数据存储器的擦除操作是按扇区进行的。EEPROM 可用于保存一些需要在应用过程中修改并且掉电不丢失的参数数据。在用户程序中，可以对 EEPROM 进行字节读/字节编程/扇区擦除操作。在工作电压偏低时，建议不要进行 EEPROM 操作，以免发送数据丢失的情况

图形化模块

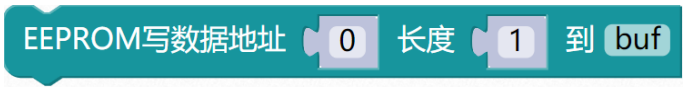
1. EEPROM 擦除指定扇区。



2. EEPROM 从 buf 中读取数据。

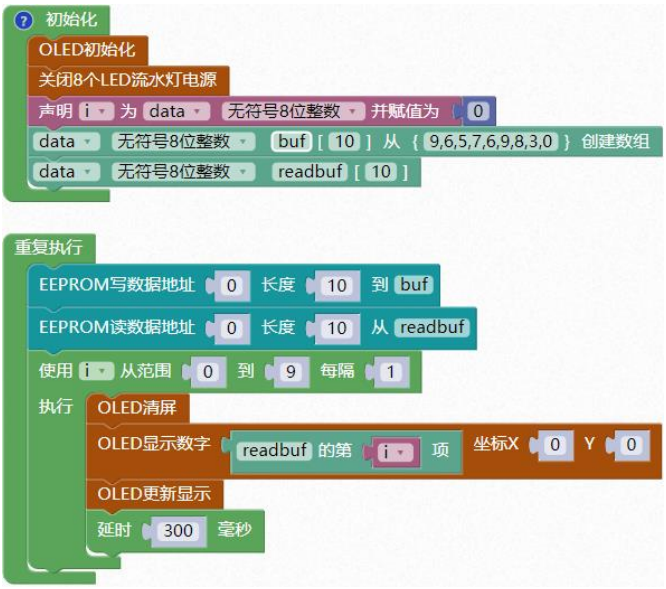


3. EEPROM 写入数据到 buf。



示例代码 1

将 buf 的数据写入到 EEPROM 当中，然后再将写入的数据读取到 readbuf 当中，并且用 oled 显示。



调用函数代码

```
引入头文件

#include "lib/eeprom.h"
```

```
void eeprom_sector_erase(uint16 EE_address)

//擦除一个扇区函数

参数: EE_address:要擦除的 EEPROM 的扇区中的一个字节地址.
```

```
void eeprom_read(uint16 EE_address,uint8 *DataAddress,uint8 length)

//读 N 个字节函数
```

```
参数: EE_address:  要读出的 EEPROM 的首地址 DataAddress: 要读出数据的指

针.length:      要读出的长度
```

```
uint8 eeprom_write(uint16 EE_address,uint8 *DataAddress,uint8 length)

// 写 N 个字节函数

参数: EE_address:  要写入的 EEPROM 的首地址。DataAddress: 要写入数据的指

针. length: 要写入的长度
```

示例代码 1

```
#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/oled.h"

#include "lib/led8.h"

#include "lib/eeprom.h"//引入  EEPROM  头文件

#include "lib/delay.h"



uint8 i = 0;

uint8 buf[10]={9,6,5,7,6,9,8,3,0};//自定义数组



uint8 readbuf[10]; //自定义数组



void setup()
```

```
{
  oled_init();//OLED 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
}

void loop()
{
  eeprom_write(0,buf,10);  //EEPROM 写数据
  eeprom_read(0,readbuf,10);  //EEPROM 读数据
  for (i = 0; i < 9; i = i + (1)) {
    oled_clear();//OLED 清屏
    oled_show_num(0,0,readbuf[(int)(i)]);
    oled_display();//OLED 更新显示
    delay(300);
  }
}

void main(void)
{
  setup();
  while(1){
    loop();
```

```
 }

}
```

SD 储存卡

硬件概述

SD 存储卡(Secure Digital Memory Card)是一种基于半导体快闪存储器的新一代高速存储设备。SD 存储卡的技术是从 MMC 卡(MultiMedia Card 格式上发展而来，在兼容 SD 存储卡基础上发展了 SDIO(SD Input/ Output)卡，此兼容性包括机械，电子，电力，信号和软件，通常将 SD、SDIO 卡俗称 SD 存储卡。

SD 卡具有高记忆容量、快速数据传输率、极大的移动灵活性以及很好的安全性，它被广泛地应用于便携式装置上，例如数码相机、平板电脑和多媒体播放器等。

目前一般都用 Micro SD Card 替代 SD 卡，原名 Trans-flash Card（TF 卡），因为体积小巧。

图形化模块

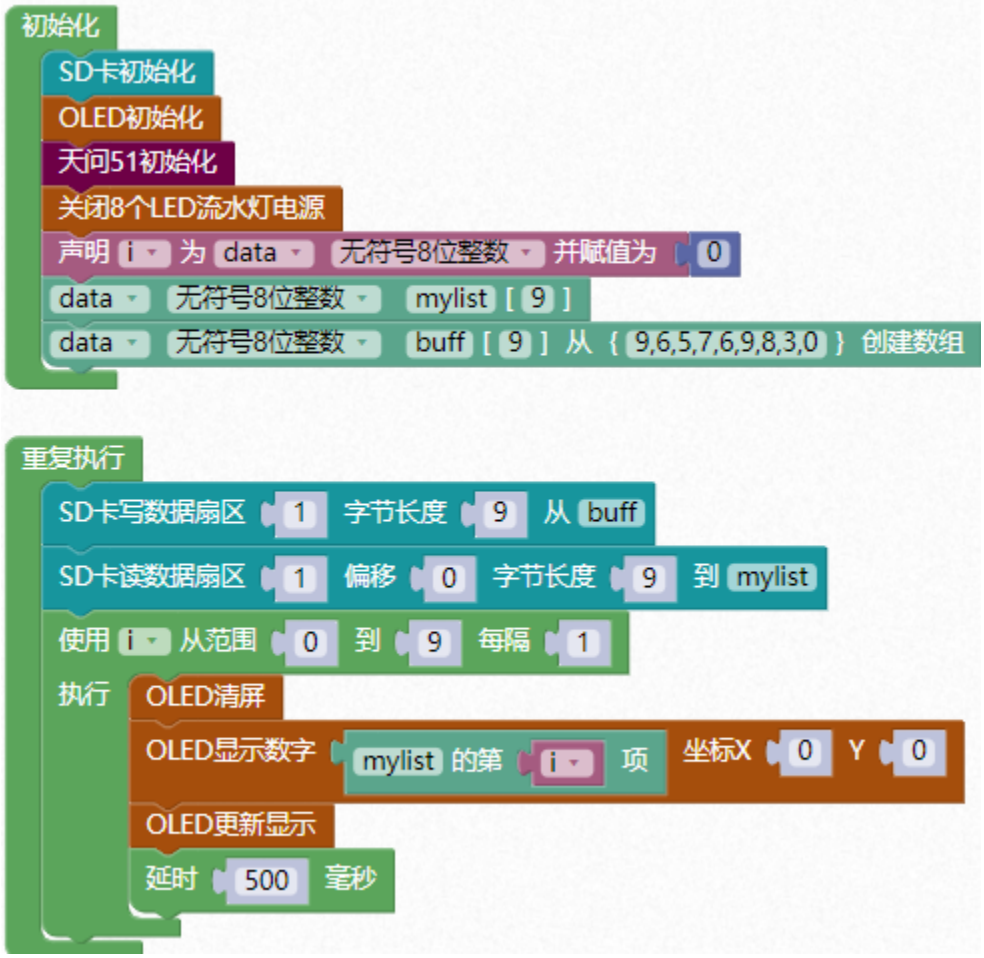
1. SD 卡初始化

2. 从 buff 中，写入字节长度，到第几个扇区。

3. 从 SD 卡中的第几个扇区，偏移量是多少，读取多少个字节到 buff 中。

示例代码 1

将 buff 中的 9 个字节数据写入到 SD 卡的第一个扇区当中，然后再将 SD 卡的第一个扇区中读取 9 个字节长度的数据到 mylist 的这个数组中。用 OLED 来显示读取到的 9 个数据。



示例代码 1

```
#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/pff.h"//引入文件系统头文件

#include "lib/oled.h"

#include "lib/hc595.h"
```

```
#include "lib/rgb.h"

#include "lib/delay.h"

#include "lib/led8.h"


uint8 i = 0;

uint8 buff[9]={9,6,5,7,6,9,8,3,0};//自定义数组


void twen_board_init()

{

  hc595_init();//HC595 初始化

  hc595_disable();//HC595 禁止点阵和数码管输出

  rgb_init();//RGB 初始化

  delay(10);

  rgb_show(0,0,0,0);//关闭 RGB

  delay(10);

}


uint8 mylist[9]; //自定义数组


void setup()

{

  disk_initialize();
```

```
  oled_init();//OLED 初始化
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
}

void loop()
{
  disk_writep(0,1);//0 代表启动完成扇区写入，1 代表写入扇区 1
  disk_writep(buff,9);//buff 为写入的数据地址，9 代表写入的字节数 9
  disk_writep(0,0);//0，0 代表写入完成
  disk_readp(mylist,1,0,9); //读取第一个扇区的 9 个字节数据，缓存到 mylist 的数组当中
  for (i = 0; i < 9; i = i + (1)) {
    oled_clear();//OLED 清屏
    oled_show_num(0,0,mylist[(int)(i)]); //OLED 显示 mylist 数组内容
    oled_display();//OLED 更新显示
    delay(500);
  }
}

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

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

文件系统

图形化模块

1. 文件操作返回状态。

2. 返回值选取。

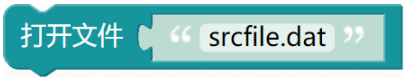
3. 文件系统挂载物理设备选择。

4. 打开目录。

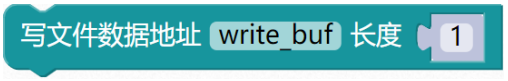
5. 读取目录信息。

6. 获取文件名。

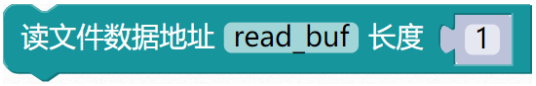
7. 打开文件。



8. 写文件数据长度到 write_buf。



9. 读取文件数据长度到 read_buf。

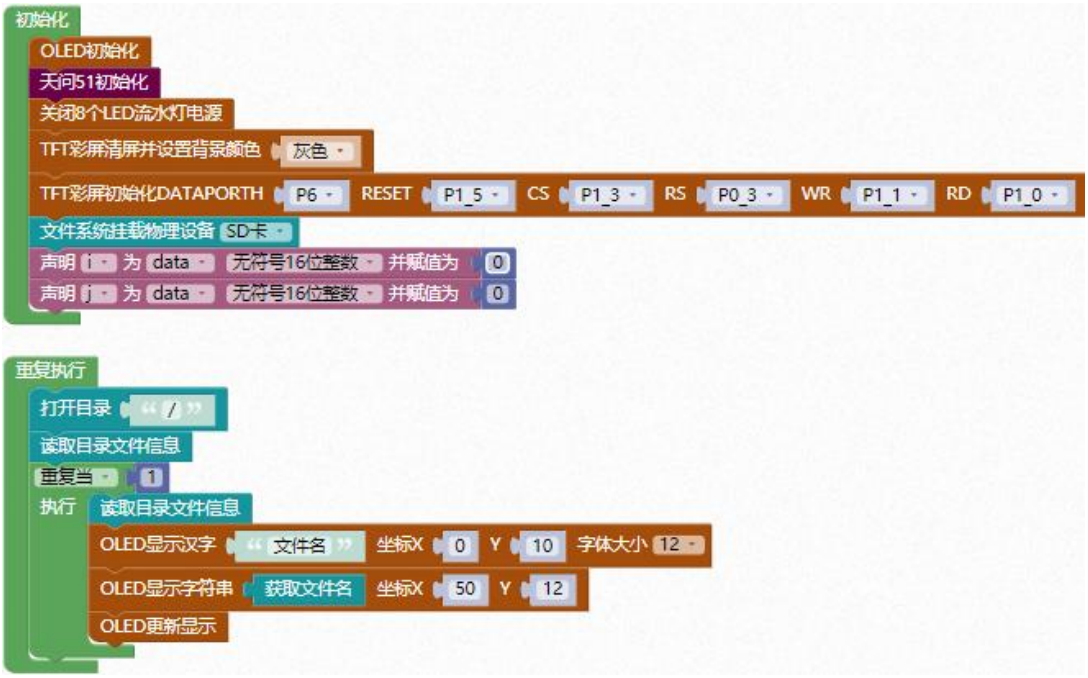


10. 获取实际读取/写入数据长度。

11. 设置读取文件指针偏移量。

示例代码 1

读取 SD 卡文件系统，并用 OLED 显示文件名字（注：需要先在 SD 卡内放置文件才能够读取，否则 SD 卡内文件为空，无法读取）。



示例代码 1

```
#define TFT_LCD_DATAPORTH P6//高 8 位数据口,8 位模式下只使用高 8 位

#define TFT_LCD_DATAPORTH_IN {P6M1=0xff;P6M0=0x00;}//P6 口高阻输入

#define TFT_LCD_DATAPORTH_OUT {P6M1=0x00;P6M0=0xff;}//P6 口推挽输出

#define TFT_LCD_DATAPORTL P2//低 8 位数据口,8 位模式下只使用高 8 位

#define TFT_LCD_RESET P1_5

#define TFT_LCD_RESET_OUT {P1M1&=~0x20;P1M0|=0x20;}//推挽输出

#define TFT_LCD_CS P1_3

#define TFT_LCD_CS_OUT {P1M1&=~0x08;P1M0|=0x08;}//推挽输出
```

```
#define TFT_LCD_RS P0_3

#define TFT_LCD_RS_OUT {P0M1&=~0x08;P0M0|=0x08;}//推挽输出

#define TFT_LCD_WR P1_1

#define TFT_LCD_WR_OUT {P1M1&=~0x02;P1M0|=0x02;}//推挽输出

#define TFT_LCD_RD P1_0

#define TFT_LCD_RD_OUT {P1M1&=~0x01;P1M0|=0x01;}//推挽输出


#include <STC8HX.h>

uint32 sys_clk = 24000000;

//系统时钟确认

#include "lib/oled.h"

#include "lib/hc595.h"

#include "lib/rgb.h"

#include "lib/delay.h"

#include "lib/led8.h"

#include "lib/tftlcd.h"

#include "lib/pff.h"//引入 文件系统 头文件


FRESULT res;

FATFS xdata g_fatfs;

uint16 i = 0;

uint16 j = 0;
```

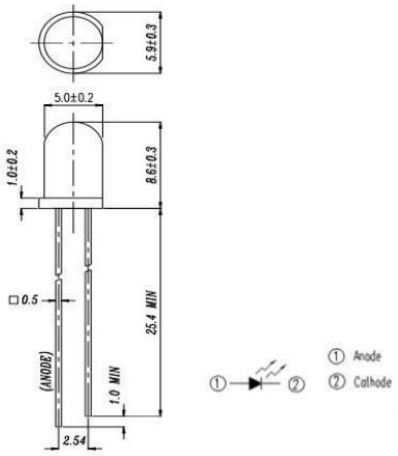
```
DIR dir;
FILINFO fileinfo;


void twen_board_init()
{
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}


void setup()
{
  oled_init();//OLED 初始化
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  tft_lcd_clear((TFT_LCD_GRAY));
  tft_lcd_init();
  res = pf_mount(&g_fatfs);
}
```

```
void loop()
{
  res = pf_opendir(&dir,"/");
  res = pf_readdir(&dir,&fileinfo);
  while (1) {
    res = pf_readdir(&dir,&fileinfo);
    oled_show_font12("文件名",0,10);
    oled_show_string(50,12,(fileinfo.fname)); //显示文件名字
    oled_display();//OLED 更新显示
  }
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

通讯模块

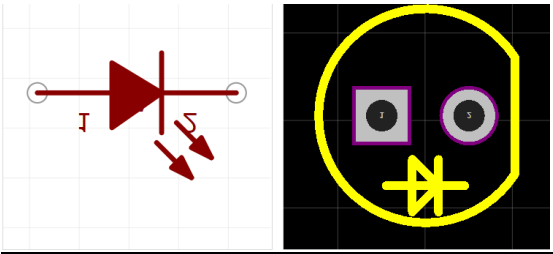
红外发送

硬件概述



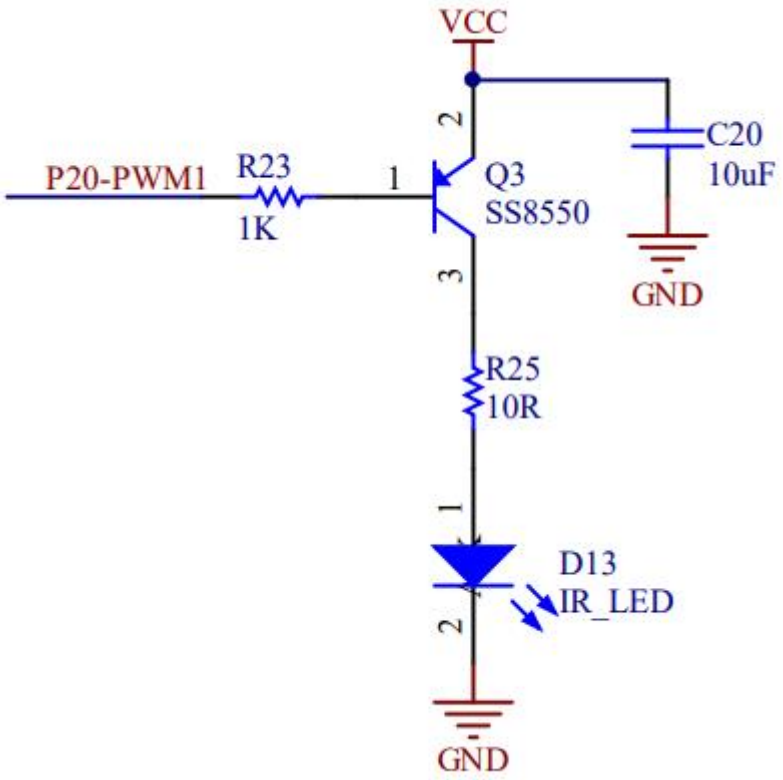
红外发光二极管（IR333C-A）是一种高辐射强度的二极管，模制在一个透明的塑料封装中，峰值波长λp=940nm。

引脚定义



序号	符号	管脚名	功 能 描 述
1	1	长脚	正极
2	2	短脚	负极

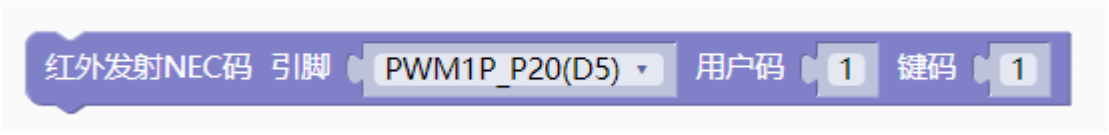
电路原理图



红外通讯一般采用 38KHz 载波，所以连在有 PWM 功能的引脚上。

图形化模块

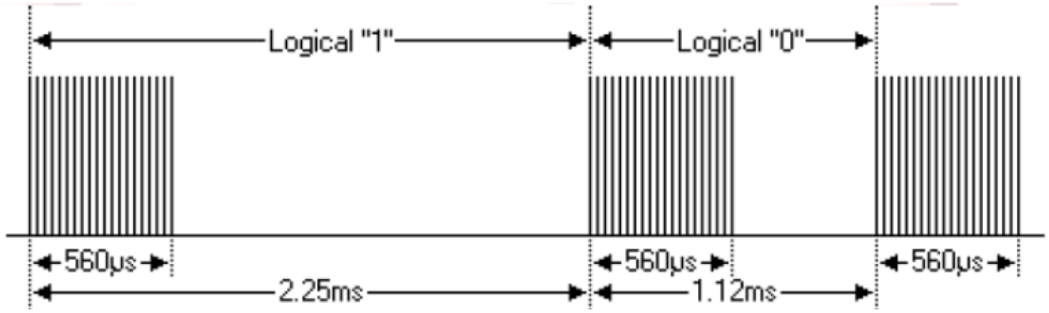
1. 红外发送 NEC 协议的数据



其中用户码和键码对应如下 NEC 协议的 Address 和 Command。

NEC协议载波：38khz

其逻辑1与逻辑0的表示如图所示：

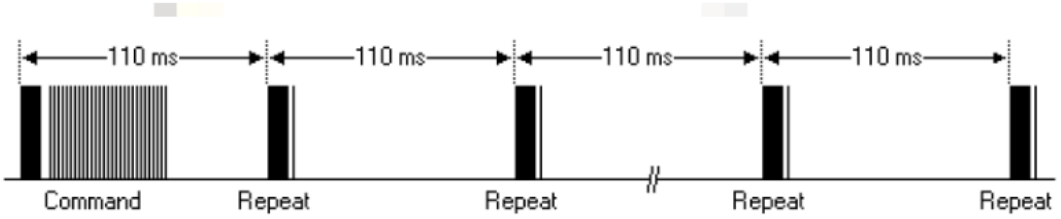


逻辑1为2.25ms，脉冲时间560us；逻辑0为1.12ms，脉冲时间560us。所以我们根据脉冲时间长短来解码。推荐载波占空比为1/3至1/4。

NEC协议格式：

首次发送的是9ms的高电平脉冲，其后是4.5ms的低电平，接下来就是8bit的地址码（从低有效位开始发），而后是8bit的地址码的反码（主要是用于校验是否出错）。然后是8bit 的命令码（也是从低有效位开始发），而后也是8bit 的命令码的反码。

以上是一个正常的序列，但可能存在一种情况：你一直按着1个键，这样的话发送的是以110ms为周期的重复码，如下图：



就是说,发了一次命令码之后，不会再发送命令码，而是每隔110ms时间，发送一段重复码。

示例代码 1

独立按键 KEY1 按下发送红外数据。

调用函数代码

```
//引入头文件
#include "lib/ir.h"
```

```
//引脚定义
#define IR_SEND_PIN          P2_0  //红外发射引脚
#define IR_SEND_PIN_OUT      {P2M1&=~0x01;P2M0|=0x01;}  //P20 推挽输出
#define IR_SEND_PIN_INIT     {P2M1|=0x01;P2M0&=~0x01;}   //P20 高阻输入
#define IR_SEND_PWM          PWM1P_P20
```

```
void ir_send_nec(uint8 address, uint8 command);      //红外发射
```

示例代码 1

```
#define PWM_DUTY_MAX 1000
//PWM 最大占空比值
#define IR_SEND_PIN P2_0
#define IR_SEND_PIN_OUT {P2M1&=~0x01;P2M0|=0x01;}//P2_0x01 推挽输出
#define IR_SEND_PIN_INIT {P2M1|=0x01;P2M0&=~0x01;}//P2_0x01 高阻输入
#define IR_SEND_PWM PWM1P_P20

#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/ir.h"

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
```

```
  twen_board_init();//天问 51 初始化
  P3M1|=0x04;P3M0&=~0x04;//高阻输入
}

void loop()
{
  if(P3_2 == 0){
    ir_send_nec(1,1);//红外发射 NEC 码
  }
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

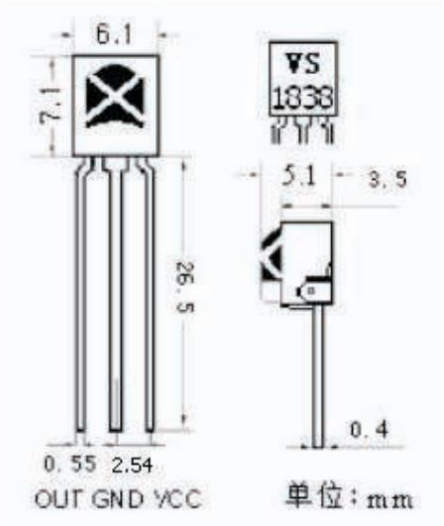
红外接收

硬件概述

内置专用 IC，自动过滤 38KHz 载波，输出红外信号。

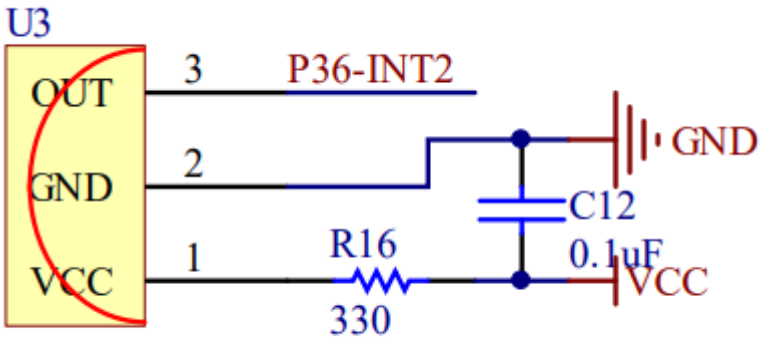
引脚定义

外型尺寸及引脚排列图



序号	符号	管脚名	功 能 描 述
1	VCC	电源脚	供电
2	GNG	接地脚	供电
3	OUT	信号脚	输出红外信号

电路原理图



图形化模块

红外库默认使用 NEC 红外协议。

1. 红外接收初始化

2. 红外接收回调函数

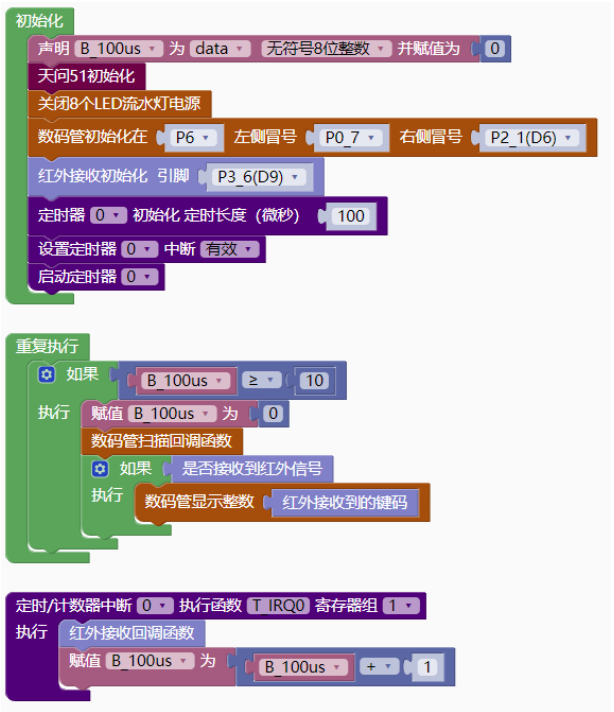
3. 判断是否接收到红外信号

4. 获取红外协议的用户码

5. 获取红外协议的键码

示例代码 1

数码管显示接收到的红外键码



调用函数代码

```
//引入头文件
#include "lib/ir.h"
```

```
//引脚定义
#define IR_REC_PIN          P3_6
```

```
#define IR_REC_PIN_MODE     {P3M1|=0x40;P3M0&=~0x40;}  //P36 输入
```

示例代码 1

```
#define NIXIETUBE_PORT P6
#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出
#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号
#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽输
出
#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号
#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推挽输
出
#define PWM_DUTY_MAX 1000
//PWM 最大占空比值
#define IR_REC_PIN P3_6
#define IR_REC_PIN_MODE {P3M1|=0x40;P3M0&=~0x40;}//P3_6 高阻输入

#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/nixietube.h"
#include "lib/ir.h"

uint8 B_100us = 0;

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}
```

```
void Timer0Init(void) //100 微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x37;   //设定定时初值
  TH0 = 0xff;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  ir_rec_callback();//红外接收回调函数
  B_100us = B_100us + 1;
}

void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  nix_init();//数码管初始化
  ir_rx_init();//红外接收初始化
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}

void loop()
{
  if(B_100us >= 10){
    B_100us = 0;
    nix_scan_callback();//数码管扫描回调函数
    if(ir_rx_available()){
      nix_display_num((ir_rx_ircode()));//数码管显示整数
    }
  }
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

I2C 模块

硬件 I2C

硬件概述

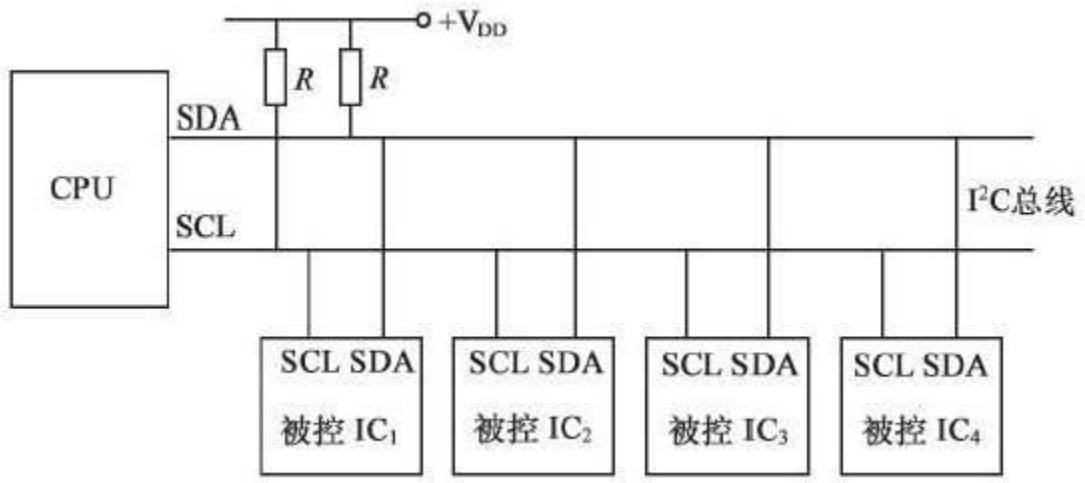
I2C 总线是由 Philips 公司开发的一种简单、双向二线制同步串行总线。它只需要两根线即可在连接于总线上的器件之间传送信息。
主器件用于启动总线传送数据，并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件．在总线上主和从、发和收的关系不是恒定的，而取决于此时数据传送方向。如果主机要发送数据给从器件，则主机首先寻址从器件，然后主动发送数据至从器件，最后由主机终止数据传送；如果主机要接收从器件的数据，首先由主器件寻址从器件．然后主机接收从器件发送的数据，最后由主机终止接收过程。在这种情况下．主机负责产生定时时钟和终止数据传送。

引脚定义

图形化库默认使用 P14、P15 两个硬件 I2C 引脚，使用代码编程可以自定义选择引脚。

序号	符号	管脚名	功 能 描 述
1	SDA	P14	串行数据线
2	SCL	P15	串行时钟线

电路原理图



图形化模块

I2C 库目前只支持主机模式。

1. I2C 总线初始化

2. I2C 读数据

3. I2C 写数据

示例代码 1

用硬件 I2C 读取和设置 RTC 时钟芯片寄存器数据。

表 1.寄存器概况

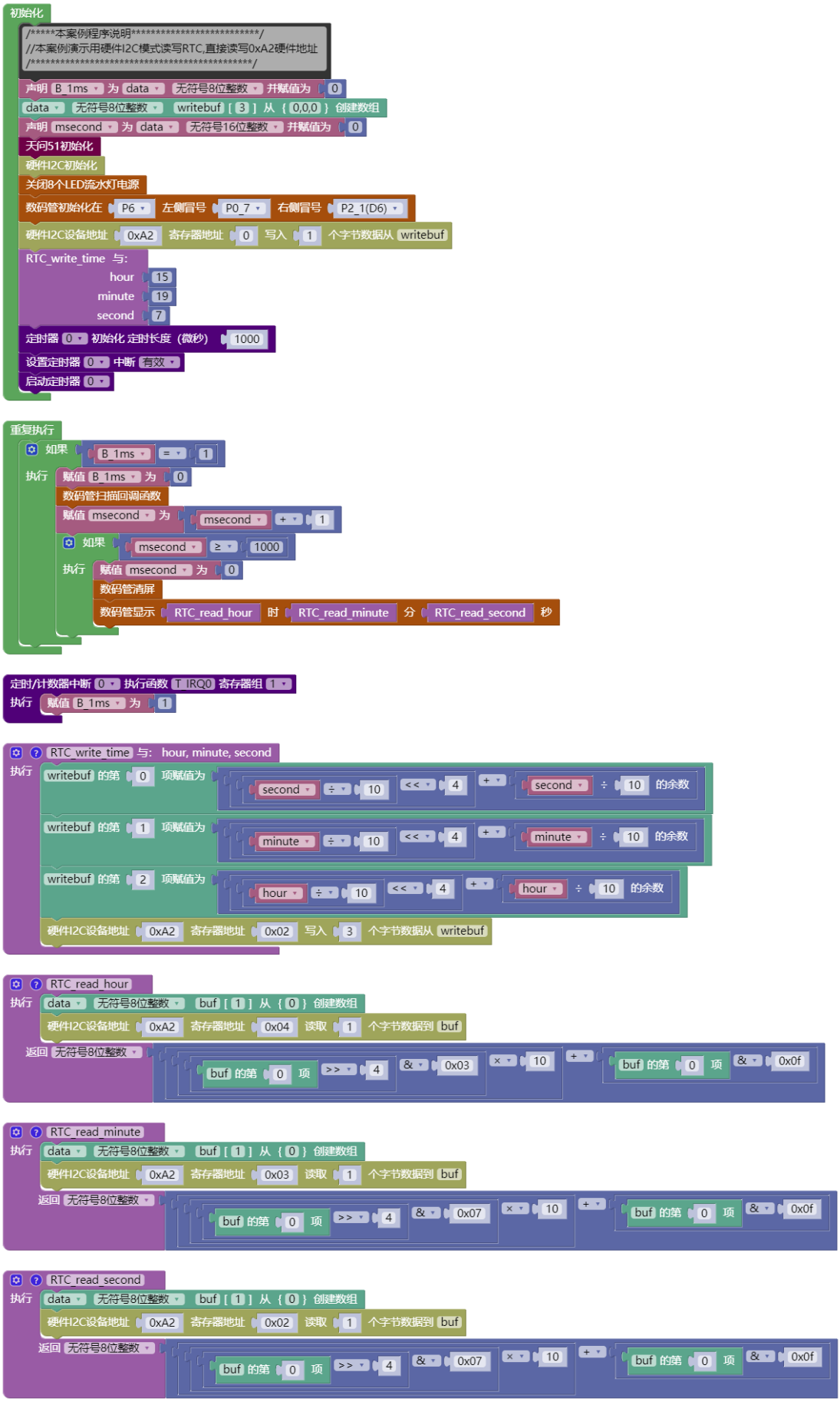
标明“—”的位无效，标明“0”的位应置为逻辑 0。

地址	寄存器名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00H	控制/状态寄存器 1	TEST	0	STOP	0	TESTC	0	0	0
01H	控制/状态寄存器 2	0	0	0	TI/TP	AF	TF	AIE	TIE
0DH	CLKOUT 频率寄存器	FE	—	—	—	—	—	FD1	FD0
0EH	定时器控制寄存器	TE	—	—	—	—	—	TD1	TD0
0FH	定时器倒计数寄存器	定时器倒计数数值							

表 2.BCD 格式寄存器概况

标明“—”的位无效

地址	寄存器名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
02H	秒	VL	00～59BCD 码格式数						
03H	分钟	—	00～59BCD 码格式数						
04H	小时	—	—	00～23BCD 码格式数					
05H	日	—	—	01～31BCD 码格式数					
06H	星期	—	—	—	—	—	0～6		
07H	月/世纪	C	—	—	01～12BCD 码格式数				
08H	年	00～99BCD 码格式数							
09H	分钟报警	AE	00～59BCD 码格式数						
0AH	小时报警	AE	—	00～23BCD 码格式数					
0BH	日报警	AE	—	01～31BCD 码格式数					
0CH	星期报警	AE	—	—	—	—	0～6		


初始化
/*****本案例程序说明**********/
//本案例演示用硬件I2C模式读写RTC,直接读写0xA2硬件地址
/*********************************/
声明 B_1ms 为 data 无符号8位整数 并赋值为 0
data 无符号8位整数 writebuf [3] 从 { 0,0,0 } 创建数组
声明 msecond 为 data 无符号16位整数 并赋值为 0
天问51初始化
硬件I2C初始化
关闭8个LED流水灯电源
数码管初始化在 P6 左侧冒号 P0_7 右侧冒号 P2_1(D6)
硬件I2C设备地址 0xA2 寄存器地址 0 写入 1 个字节数据从 writebuf
RTC_write_time 与:
hour 15
minute 19
second 7
定时器 0 初始化 定时长度（微秒） 1000
设置定时器 0 中断 有效
启动定时器 0
重复执行
如果 B_1ms = 1
执行 赋值 B_1ms 为 0
数码管扫描回调函数
赋值 msecond 为 msecond + 1
如果 msecond ≥ 1000
执行 赋值 msecond 为 0
数码管清屏
数码管显示 RTC_read_hour 时 RTC_read_minute 分 RTC_read_second 秒
定时/计数器中断 0 执行函数 T_IRQ0 寄存器组 1
执行 赋值 B_1ms 为 1
RTC_write_time 与: hour, minute, second
执行
writebuf 的第 0 项赋值为 second ÷ 10 << 4 + second ÷ 10 的余数
writebuf 的第 1 项赋值为 minute ÷ 10 << 4 + minute ÷ 10 的余数
writebuf 的第 2 项赋值为 hour ÷ 10 << 4 + hour ÷ 10 的余数
硬件I2C设备地址 0xA2 寄存器地址 0x02 写入 3 个字节数据从 writebuf
RTC_read_hour
执行
data 无符号8位整数 buf [1] 从 { 0 } 创建数组
硬件I2C设备地址 0xA2 寄存器地址 0x04 读取 1 个字节数据到 buf
返回 无符号8位整数
buf 的第 0 项 >> 4 & 0x03 × 10 + buf 的第 0 项 & 0x0f
RTC_read_minute
执行
data 无符号8位整数 buf [1] 从 { 0 } 创建数组
硬件I2C设备地址 0xA2 寄存器地址 0x03 读取 1 个字节数据到 buf
返回 无符号8位整数
buf 的第 0 项 >> 4 & 0x07 × 10 + buf 的第 0 项 & 0x0f
RTC_read_second
执行
data 无符号8位整数 buf [1] 从 { 0 } 创建数组
硬件I2C设备地址 0xA2 寄存器地址 0x02 读取 1 个字节数据到 buf
返回 无符号8位整数
buf 的第 0 项 >> 4 & 0x07 × 10 + buf 的第 0 项 & 0x0f

调用函数代码

```
//引入头文件
#include "lib/hardiic.h"
```

```
//定义引脚
#define HARDIIC_IICX 0x80 //将 IIC 设置为 P1_5,P1_4
#define HARDIIC_SCL_OUT {P1M1|=0x20;P1M0|=0x20;} //开漏输出
#define HARDIIC_SDA_OUT {P1M1|=0x10;P1M0|=0x10;} //开漏输出
```

```
void hardiic_init();//引脚 I2C 初始化
void hardiic_read_nbyte(uint8 device_addr, uint8 reg_addr, uint8 *p, ui
nt8 number);//硬件 I2C 读字节数据
void hardiic_write_nbyte(uint8 device_addr, uint8 reg_addr, uint8 *p, u
int8 number);//硬件 I2C 写字节数据
```

示例代码 1

```
#define HARDIIC_IICX 0x80 //将 IIC 设置为 P1_5,P1_4
#define HARDIIC_SCL_OUT {P1M1|=0x20;P1M0|=0x20;} //开漏输出
#define HARDIIC_SDA_OUT {P1M1|=0x10;P1M0|=0x10;} //开漏输出
#define NIXIETUBE_PORT P6
#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出
#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号
#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽输
出
#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号
#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推挽输
出

#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/hardiic.h"
#include "lib/led8.h"
#include "lib/nixietube.h"

uint8 B_1ms = 0;
uint16 msecond = 0;
void RTC_write_time(uint8 hour, uint8 minute, uint8 second);
uint8 RTC_read_hour();
uint8 RTC_read_minute();
uint8 RTC_read_second();
```

```
uint8 writebuf[3]={0,0,0};
void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void Timer0Init(void) //1000 微秒@24.000MHz
{
  TMOD |= 0x00;   //模式 0
  TL0 = 0x2f;   //设定定时初值
  TH0 = 0xf8;   //设定定时初值
}

void T_IRQ0(void) interrupt 1 using 1{
  B_1ms = 1;
}

/*描述该功能...
*/
void RTC_write_time(uint8 hour, uint8 minute, uint8 second){
  writebuf[(int)(0)] = ((second / 10)<<4) + second % 10;
  writebuf[(int)(1)] = ((minute / 10)<<4) + minute % 10;
  writebuf[(int)(2)] = ((hour / 10)<<4) + hour % 10;
  hardiic_write_nbyte(0xA2,0x02,writebuf,3);//I2C 写入 n 个字节数据
}

uint8 buf[1]={0};
/*描述该功能...
*/
uint8 RTC_read_hour(){
```

```
  hardiic_read_nbyte(0xA2,0x04,buf,1);//I2C 读取 n 个字节数据
  return ((buf[(int)(0)]>>4)&0x03) * 10 + (buf[(int)(0)]&0x0f);
}

/*描述该功能...
*/
uint8 RTC_read_minute(){
  hardiic_read_nbyte(0xA2,0x03,buf,1);//I2C 读取 n 个字节数据
  return ((buf[(int)(0)]>>4)&0x07) * 10 + (buf[(int)(0)]&0x0f);
}

/*描述该功能...
*/
uint8 RTC_read_second(){
  hardiic_read_nbyte(0xA2,0x02,buf,1);//I2C 读取 n 个字节数据
  return ((buf[(int)(0)]>>4)&0x07) * 10 + (buf[(int)(0)]&0x0f);
}

void setup()
{
  /*****本案例程序说明*************************/
  //本案例演示用硬件 I2C 模式读写 RTC,直接读写 0xA2 硬件地址
  /*******************************************/
  twen_board_init();//天问 51 初始化
  hardiic_init();
  led8_disable();//关闭 8 个 LED 流水灯电源
  nix_init();//数码管初始化
  hardiic_write_nbyte(0xA2,0,writebuf,1);//I2C 写入 n 个字节数据
  RTC_write_time(15, 19, 7);
  Timer0Init();
  EA = 1; // 控制总中断
  ET0 = 1; // 控制定时器中断
  TR0 = 1;// 启动定时器
}

void loop()
{
  if(B_1ms == 1){
    B_1ms = 0;
    nix_scan_callback();//数码管扫描回调函数
    msecond = msecond + 1;
    if(msecond >= 1000){
      msecond = 0;
      nix_display_clear();//数码管清屏
```

```
      nix_display_time2((RTC_read_hour()),(RTC_read_minute()),(RTC_read
_second()));//数码管显示时间
    }
  }
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

软件 I2C

硬件概述

引脚定义

序号	符号	管脚名	功 能 描 述
1			
2			

电路原理图

图形化模块

示例代码 1

调用函数代码

示例代码 1

SPI 模块

硬件 SPI

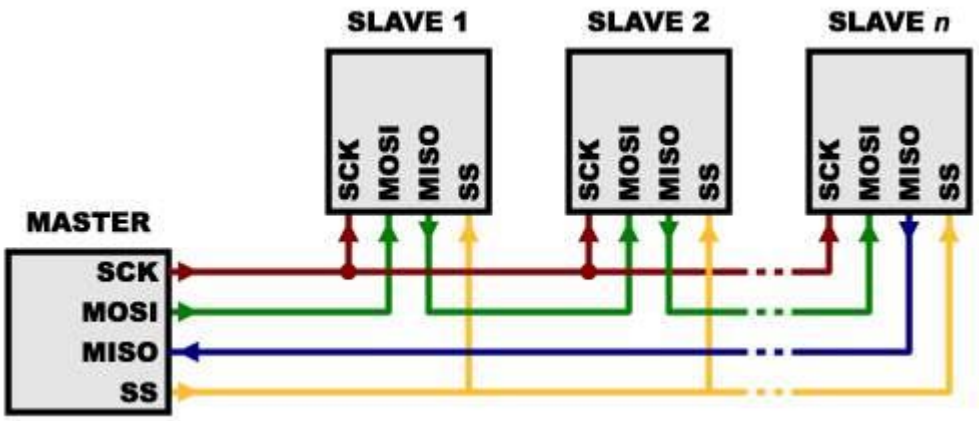
硬件概述

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 SS。

引脚定义

序号	符号	管脚名	功 能 描 述
1	SCK	P25	串行时钟线 SCK
2	MOSI	P23	主机输出/从机输人数据线
3	MISO	P24	主机输人/从机输出数据线
4	SS		低电平有效的从机选择线

电路原理图



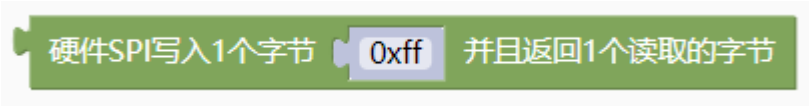
图形化模块

1. 硬件 SPI 初始化

2. 硬件 SPI 写数据

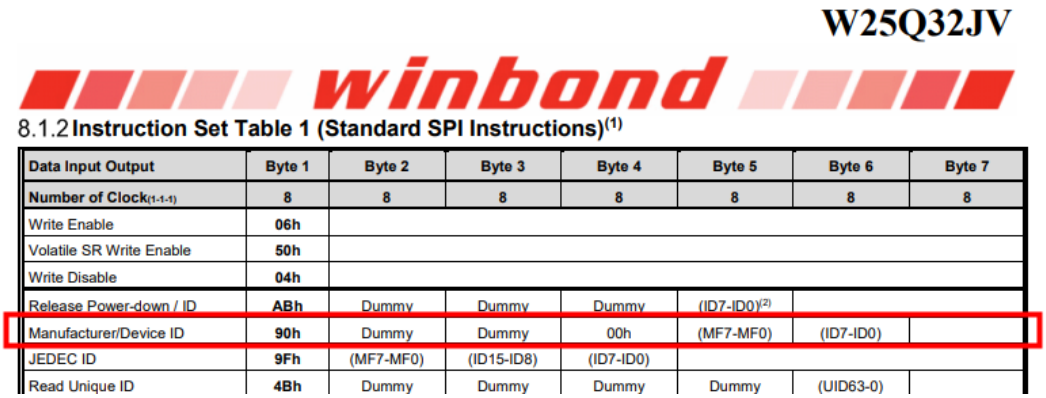
3. 硬件 SPI 读数据

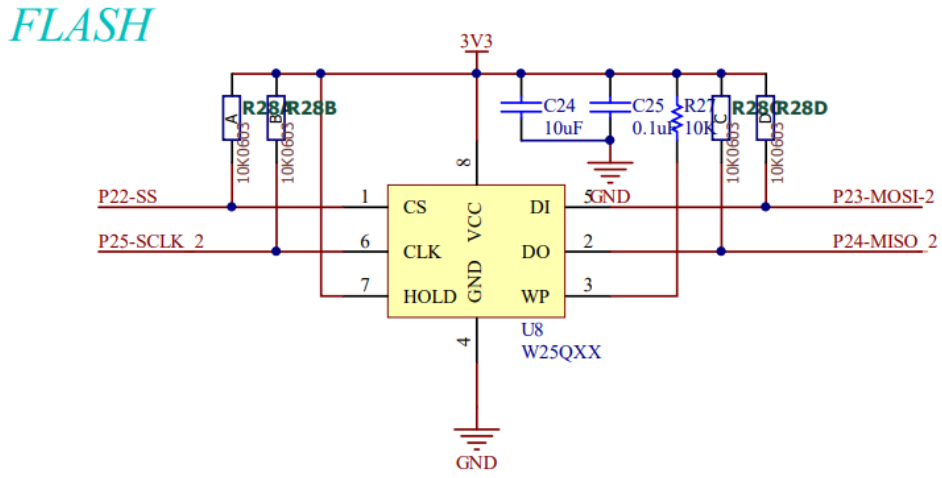
4. 硬件 SPI 写数据同时返回读取数据

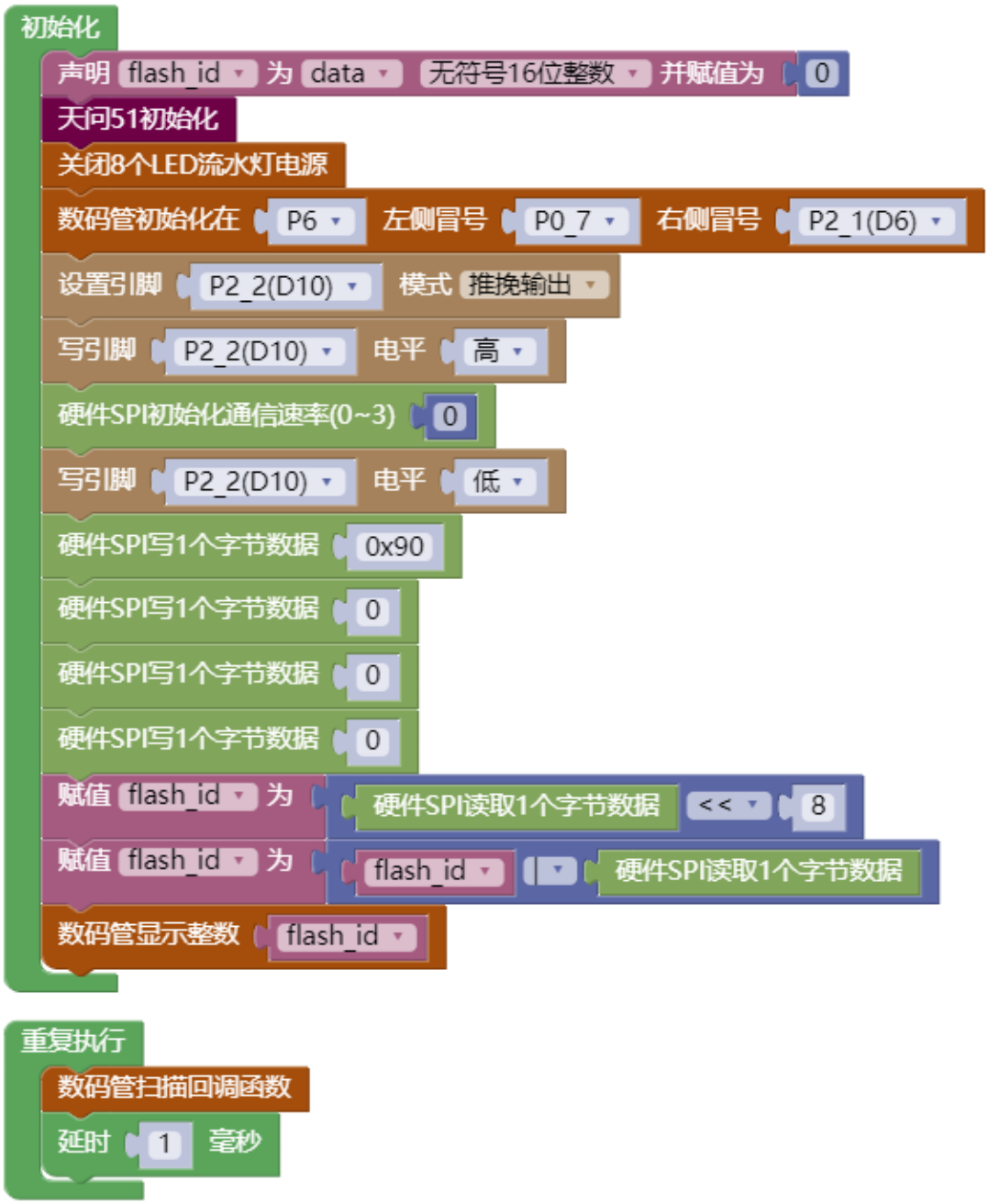


示例代码 1

用硬件 SPI 读取 FLASH ID。天问 51 开发板上的 FLASH SS 引脚连到 P22，FLASH 芯片型号为 W25Q32，ID 为 EF15H，对应的十进制数值为 61205。







调用函数代码

```
//引入头文件
#include "lib/hardspi.h"
```

```
//定义硬件 SPI 引脚
#define HARDSPI_SPI_SCK_PIN P2_5
#define HARDSPI_SCK_PIN_MODE {P2M1&=~0x20;P2M0&=~0x20;} //双向 IO 口
#define HARDSPI_SPI_MISO_PIN P2_4
#define HARDSPI_MISO_PIN_MODE {P2M1&=~0x10;P2M0&=~0x10;} //双向 IO 口
#define HARDSPI_SPI_MOSI_PIN P2_3
```

```
#define HARDSPI_MOSI_PIN_MODE {P2M1&=~0x08;P2M0&=~0x08;} //双向 IO 口
```

```
void hardspi_init(uint8 speed);          //spi 初始化
void hardspi_write_byte(uint8 out);      //spi 写一个字节
uint8 hardspi_read_byte();               //spi 读取一个字节
uint8 hardspi_wr_data(uint8 data);       //spi 写入一个字节并且返回一个读取
的字节
```

示例代码 1

```
#define NIXIETUBE_PORT P6
#define NIXIETUBE_PORT_MODE {P6M1=0x00;P6M0=0xff;}//推挽输出
#define NIXIETUBE_LEFT_COLON_PIN P0_7//左侧数码管冒号
#define NIXIETUBE_LEFT_COLON_PIN_MODE {P0M1&=~0x80;P0M0|=0x80;}//推挽输
出
#define NIXIETUBE_RIGHT_COLON_PIN P2_1//右侧数码管冒号
#define NIXIETUBE_RIGHT_COLON_PIN_MODE {P2M1&=~0x02;P2M0|=0x02;}//推挽输
出
#define HARDSPI_SPI_SCK_PIN P2_5
#define HARDSPI_SCK_PIN_MODE {P2M1&=~0x20;P2M0&=~0x20;} //双向 IO 口
#define HARDSPI_SPI_MISO_PIN P2_4
#define HARDSPI_MISO_PIN_MODE {P2M1&=~0x10;P2M0&=~0x10;} //双向 IO 口
#define HARDSPI_SPI_MOSI_PIN P2_3
#define HARDSPI_MOSI_PIN_MODE {P2M1&=~0x08;P2M0&=~0x08;} //双向 IO 口

#include <STC8HX.h>
uint32 sys_clk = 24000000;
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
#include "lib/nixietube.h"
#include "lib/hardspi.h"

uint16 flash_id = 0;

void twen_board_init()
{
  P0M1=0x00;P0M0=0x00;//双向 IO 口
  P1M1=0x00;P1M0=0x00;//双向 IO 口
  P2M1=0x00;P2M0=0x00;//双向 IO 口
  P3M1=0x00;P3M0=0x00;//双向 IO 口
  P4M1=0x00;P4M0=0x00;//双向 IO 口
```

```
  P5M1=0x00;P5M0=0x00;//双向 IO 口
  P6M1=0x00;P6M0=0x00;//双向 IO 口
  P7M1=0x00;P7M0=0x00;//双向 IO 口
  hc595_init();//HC595 初始化
  hc595_disable();//HC595 禁止点阵和数码管输出
  rgb_init();//RGB 初始化
  delay(10);
  rgb_show(0,0,0,0);//关闭 RGB
  delay(10);
}

void setup()
{
  twen_board_init();//天问 51 初始化
  led8_disable();//关闭 8 个 LED 流水灯电源
  nix_init();//数码管初始化
  P2M1&=~0x04;P2M0|=0x04;//推挽输出
  P2_2 = 1;
  hardspi_init(0);//硬件 SPI 初始化
  P2_2 = 0;
  hardspi_write_byte(0x90);//硬件 SPI 写 1 个字节数据
  hardspi_write_byte(0);//硬件 SPI 写 1 个字节数据
  hardspi_write_byte(0);//硬件 SPI 写 1 个字节数据
  hardspi_write_byte(0);//硬件 SPI 写 1 个字节数据
  flash_id = ((hardspi_read_byte())<<8);
  flash_id = (flash_id|(hardspi_read_byte()));
  nix_display_num(flash_id);//数码管显示整数
}

void loop()
{
  nix_scan_callback();//数码管扫描回调函数
  delay(1);
}

void main(void)
{
  setup();
  while(1){
    loop();
  }
}
```

软件 SPI

硬件概述

引脚定义

序号	符号	管脚名	功 能 描 述
1			
2			

电路原理图

图形化模块

示例代码 1

调用函数代码

示例代码 1

常见问题

1. 是否支持 Keil 编程
   官方资料已提供 50 多 Keil 工程案例， 所有图形化编程案例都可从网
   页版导出 Keil 工程。
2. 驱动安装不上
   STC-LINK 的驱动为 CP210x 驱动，如安装不成功请尝试重启电脑、换 USB 口。
3. 程序无法下载
   请确认 USB 线为配套的 USB 线，STC-LINK 下载器是否插上，驱动是否安装正常，设
   备管理器能是否看到串口号，电源键是否打开，电源 3V/5V 跳线帽是否插上。一般不

建议用 USB 延长线。

4. 图形化模块里没有的模块是否支持扩展
   支持自定义模块，编程软件可添加扩展。
5. 图形化的程序效率是不是很低
   图形化会自动生成专业的 C 代码，效率和直接写代码一样。
6. 是否支持仿真
   可在平台一键导出 Keil 工程，可以用 Keil 软件仿真。好搭 Block 软件已经把大部分外设和驱动自动生成好，用户只需要关心应用层的逻辑处理，推荐使用串口打印输出变量值来调试。