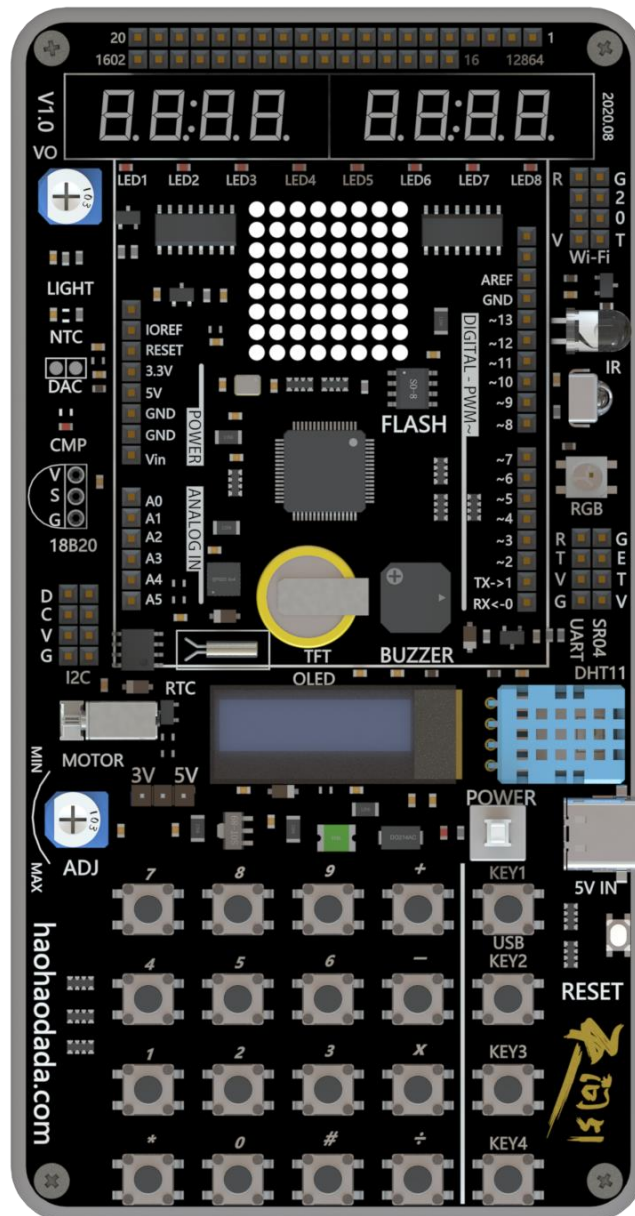


图形化编程学习 STC8H

V1.0



好好搭搭课程团队

杭州好好搭搭科技有限公司成立于 2015 年，专注于创新教育、人工智能教育、编程教育平台、硬件、软件、课程等开发，是国内最早的在线编程平台之一，致力于提供编程教育、创客教育、人工智能教育、STEAM 教育整体解决方案的互联网在线教育科技公司。五年来，为 100 多个教育局、6000 多所学校、200 多万学生提供在线平台、课程、器材、师训，主办协办各类编程及创客比赛。2018 年第一家好好搭搭旗舰店开业，揭开了好好搭搭校外培训的序幕，为全国 300 多所教育机构提供编程培训、研学旅行、营地活动方案和技术支持。好好搭搭以创新理念，做编程教育引领者。好好搭搭拥有自主知识产权的云编程平台系统，可以实现 8 位、32 位 MCU 的云编译、无线下载功能，积极和业界合作，做中国自己的云开发平台，助力中国芯腾飞。

官方网站：<http://haohaodada.com>

天问 51 资料页：<http://tw51.haohaodada.com>

天问 51 开发板技术 QQ 群一：1138055784

微信公众号：



抖音：



B 站：



目录

内容简介	1
前 言	1
学习方法	2
第一篇 硬件篇	2
天问 51 设计原则	2
天问 51 原理图	5
STC-Link 下载器使用	21
运行第一个程序	21
串口下载	24
USB 下载方式	26
无线下载方式	27
第二篇 平台篇	28
STC8 图形化设计思想	28
STC8 图形化界面介绍	28
STC8 图形化编程的基本操作	30
STC8 字符式编程基本操作	32
STC8 库管理功能	33
第三篇 初级篇（基本模块使用）	33
第一章 STC8H 单片机 IO 模块	33
第一节 单片机 IO 模块综述	33
第二节 点亮一个 LED	41
第三节 让 LED 闪烁	53
第四节 跑马灯	58
第五节 数码管显示原理	71
第六节 点阵显示原理	79
第七节 独立按键控制 LED	84
第八节 外部中断	89
第九节 74HC595 移位寄存器	93
第二章 STC8H 的定时器	99
第一节 STC8H 定时器综述	99
第二节 定时器模式	99
第三节 定时器中断	99
第四节 设计一个倒计时	99
第五节 设计一个计数器	99
第六节 万年历实现	99

	第七节	简易计算器.....	100
	第八节	定时器模拟 PWM.....	100
第三章	STC8H 的 ADC 模块.....		100
	第一节	STC8H ADC 模块综述.....	100
	第二节	电位器.....	100
	第三节	光敏传感器吧.....	101
	第四节	NTC 温度传感器 A.....	101
第四章	STC8H 的 PWM 模块.....		102
	第一节	STC8H PWM 模块综述.....	102
	第二节	震动马达.....	102
	第三节	蜂鸣器.....	102
	第四节	模拟 DAC.....	102
第五章	STC8H 串口通信.....		103
	第一节	STC8H 串口模块综述.....	103
	第二节	轮询模式.....	103
	第三节	中断模式.....	103
	第四节	上位机控制.....	103
第六章	STC8H 的 I2C 模块（主机模式）.....		104
	第一节	STC8H I2C 模块综述.....	104
	第二节	RTC 实时时钟.....	104
	第三节	加速度传感器.....	104
	第四节	OLED 显示屏.....	104
第七章	STC8H 的 SPI 模块.....		105
	第一节	STC8H SPI 模块综述.....	105
	第二节	SPI Flash.....	105
第八章	STC8H 的比较器.....		105
第九章	STC8H 的 EEPROM.....		106
	第一节	STC8H EEPROM 模块综述.....	106
	第二节	EEPROM 实验.....	106
第十章	STC8H 的看门狗.....		106
	第一节	STC8H 看门狗模块综述.....	106
	第二节	看门狗实验.....	106
第十一章	STC8H 的低功耗模式.....		106
	第一节	STC8H 看门狗模块综述.....	107
	第二节	看门狗实验.....	107
第四篇	中级篇（总线式模块和扩展模块应用）.....		107
第一章	单总线.....		107
	第一节	单总线综述.....	107
	第二节	RGB 彩灯.....	107
	第三节	DHT11 温湿度传感器.....	107

	第四节	DS18B20 温度传感器.....	108
第二章	并口总线.....		108
	第一节	并口总线综述.....	108
	第二节	LCD 1602.....	108
	第三节	LCD 12864.....	109
	第四节	TFT 彩屏.....	109
第三章	扩展模块.....		109
	第一节	超声波模块.....	109
	第二节	舵机模块.....	110
	第三节	继电器模块.....	110
第四章	红外编码和解码.....		110
	第一节	红外通讯协议综述.....	110
	第二节	红外接收.....	110
	第三节	红外发送.....	111
第五章	STC8H 的 PWM 高级模式.....		111
	第一节	捕获模式.....	111
	第二节	互补输出.....	111
第六章	STC8H 的 I2C 从机模式.....		112
第七章	工业总线.....		112
	第一节	工业应用总线综述.....	112
	第二节	485 总线.....	112
第五篇	高级篇（高级应用案例）.....		112
第一章	文件系统.....		112
	第一节	文件系统综述.....	112
	第二节	SD 卡读写.....	113
	第三节	FAT 文件系统实现.....	113
第二章	USB.....		113
	第一节	硬件设计.....	113
	第二节	虚拟串口.....	113
	第三节	HID 设备.....	113
	第四节	大容量设备.....	114
	第五节	MIDI 音乐.....	114
第三章	物联网.....		114
	第一节	物联网综述.....	114
	第二节	蓝牙通讯.....	114
	第三节	2.4G 无线通讯.....	114
	第四节	以太网通讯.....	115
	第五节	Wi-Fi 通讯.....	115
第六篇	综合应用（实战案例）.....		115
第一章	运动手表.....		115

第一节	功能需求分析.....	115
第二节	硬件搭建.....	115
第三节	程序框图.....	115
第四节	字符式编程.....	115
第五节	运行效果.....	116
第二章	数码相框.....	116
第一节	功能需求分析.....	116
第二节	硬件搭建.....	116
第三节	程序框图.....	116
第四节	字符式编程.....	116
第五节	运行效果.....	116
第三章	手写板.....	116
第一节	功能需求分析.....	116
第二节	硬件搭建.....	116
第三节	程序框图.....	116
第四节	字符式编程.....	116
第五节	运行效果.....	116
第四章	示波器.....	116
第一节	功能需求分析.....	116
第二节	硬件搭建.....	117
第三节	程序框图.....	117
第四节	字符式编程.....	117
第五节	运行效果.....	117
第五章	智能家居.....	117
第一节	功能需求分析.....	117
第二节	硬件搭建.....	117
第三节	程序框图.....	117
第四节	字符式编程.....	117
第五节	运行效果.....	117
附录	118
嵌入式 C 关键字	118
数据类型	118
常用编码格式和转换	118
ASSIC 码表	118
代码格式规范	118
常用算法	118

内容简介

本教程硬件基于天问 51 开发板，软件基于好好搭搭图形化编程平台，版权归好好搭搭课程团队所有。

本教程将由浅入深，带领大家学习 51 单片机及嵌入式相关的常用外设。

共分为六篇：1. 硬件篇，主要介绍天问 51 硬件平台；2. 平台篇，主要介绍 STC8 的图形化编程平台的基本使用；3. 初级篇，通过 11 章，27 个基础案例，掌握 STC8H 的常用外设；4. 中级篇，通过 7 章，15 个中级案例，掌握常用总线模块、扩展模块、PWM、I2C 等高级模式；5. 高级篇，通过 3 章，9 个高级案例，掌握文件系统、USB、物联网相关知识；6. 实战篇，通过 5 个综合案例，融会贯通前面所学知识。

因为 STC8H 芯片已经不同于传统 89C51 单片机，内部额外集成了 AD、PWM、I2C、SPI、USB，这些外设基本已经包含目前 32 位单片机的常用外设，所以“学 51=学 ARM”。

本教程适合零基础的大学、高职院校学生和电子爱好者的入门教程，通过本教程的学习，不仅学习了 51 单片机，更系统的学习并掌握嵌入式的软硬件开发、调试、学习方法，为后面学习 32 位单片机打下坚实基础。

前 言

现代社会，随着计算机的普及和微处理器的不断更新发展，给我们生活带来了翻天覆地的变化。单片机的发展先后经历了 4 位、8 位、16 位和 32 位等阶段；8 位单片机经历了不断迭代发展，特别是在 STC 宏晶公司姚永平带领的研发团队修改、优化、创新，达到 1T 时钟速度，极其丰富的外设，成为嵌入式开发入门的首选。

传统 51 单片机的教程，先讲大堆枯燥的微机原理，汇编语言、C 语言，再到嵌入式开发。往往 80% 的初学者被挡在了前面几个难啃的骨头上，学的云里雾里。本课程得益于天问 51 划时代的软硬件设计理念、系统架构，初学者可以通过图形化快速的开发应用，在开发应用过程之中嵌入微机原理、汇编语言知识、C 语言知识，让学习者清楚的认识到了单片机能做什么，有哪些组成，面对一个需求或问题的时候，知道怎么排查问题，找资料，调试，自己应该怎么学习。授之以鱼不如授之以渔。

本教程采用图形化和代码同时讲解，同时做如下约定。

关于硬件

1. 单片机发展及应用趋势
2. 单片机的基本结构
3. 单片机的最小系统
4. 基本的数电模电知识

以上硬件知识本次课程中不做系统性深入讲解，课程中遇到会做简单的介绍。建议学过本教程后，再回过头去学习微机原理。

关于软件

课程会先用图形化讲解，再结合代码，把 C 语言的知识穿插进来讲解，所以对没有 C 语言基础的也能很好的入门，如果有 C 语言基础能更快的掌握单片机的开发。建议学过本教程后，再回过头去学习 C 语言，推荐教材《C Primer Plus》。

学习方法

想要在嵌入式开发这条路上走得远，最重要的是自己的兴趣，很多搞嵌入式开发的前辈基本都是小时候就很喜欢捣鼓东西，拆解家里的一些小家电。兴趣能驱动你遇到一个问题的时候，无时无刻的都在思考，包括洗澡、上厕所、睡觉。

嵌入式开发学习路线：

从操作系统层面可以分为裸机开发、RTOS 实时操作系统开发、Linux 系统开发。

从硬件层面可以分为简单系统的电路设计，商业应用产品开发中的选型、稳定性、可靠性设计，芯片设计。

从程序层面可以分为程序基础、程序逻辑、算法。

至于网络、蓝牙等各种协议和外围芯片，用到什么补什么。

嵌入式开发这条路需要学到的知识非常多而杂，作为初学者切记盲目的求新，博而不专，而应该打好基础，掌握学习的方法，做到融会贯通，举一反三。

嵌入式学习的三重境界：

第一重、初识境界

小荷才露尖尖角，感觉一下子很多东西要学，每天不知疲倦的学习，在各种论坛和群里请教问题或者查找资料，经常遇到各种最后能被自己气死的低级错误。

第二重、熟知境界

磨拳霍霍，技术到了一定阶段，常用的基本都会，开始不屑于小白的提问，追求高技术，挑战自我。

第三重、忘我境界

蓦然回首，对技术有了系统性的认识，对新技术能快速的消化吸收，能不断的基于现有技术进行应用创新，为人也开始变得谦逊，对茫茫宇宙还有很多的未知，感到自己的渺小。

第一篇 硬件篇

天问 51 设计原则

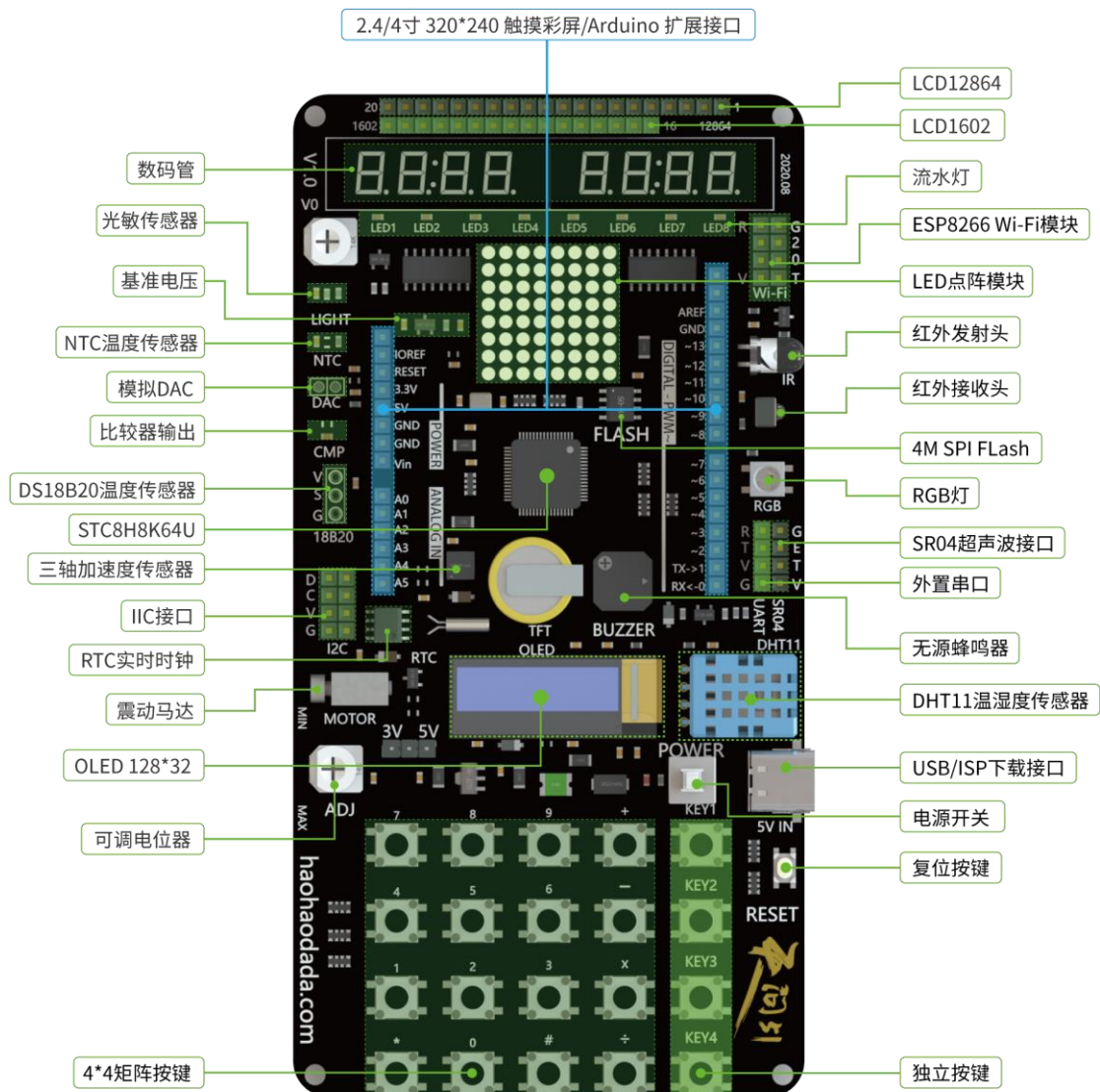
设计初衷

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。

主板功能

天问 51 开发板采用 STC8H8K64US4-64 芯片，该芯片性能强劲、功能强大是目前 STC 最先进的 51 芯片。主频达到 48Mhz 的 1T8051 内核，外设有 ADC、PWM、IIC、SPI、UART 等，更为可贵的是在 STC51 首次加入了 USB 模块。由于这些模块的加入，51 就非常强大了，可以学习更多的嵌入式知识。天问 51 开发板充分利用这些外设，巧妙设计了多种电路，在板子上集成了更多设备，学习了天问 51 就可以很方便地学习 ARM，只是内核不同而已。



IO 模块: LED、8 位数码管、8*8 点阵、4 位独立按键、4*4 矩阵按键

ADC 模块: 模拟量 NTC、光敏、电位器 (AD 兼比较器)

PWM 模块: PWM 马达、蜂鸣器、DAC (PWM+RC)、红外接收+发射 (外部中断+PWM)

I2C 总线: 3 轴加速度、RTC、OLED

SPI 总线: SPIflash、TFT 卡、触摸屏

单线总线: 18B20、DHT11、WS2812RGB

UART 总线: WIFI、蓝牙、超声波

外扩并口总线: 1602、12864、TFT

USB 总线: U 盘、虚拟串口、键盘、鼠标、MIDI

Arduino 扩展模块: 连接 Arduino 产品线的所有设备

从这些设备中, 可以看到基本涵盖了市面上的所有设备。在学习这些设备过程中, 大家可以学习到或掌握各类总线知识、TFT 卡、文件系统、网络原理、USB、显示原理、电机驱动等等。可以说, 学好天问 51 再学习其他知识就比较容易了。学 51=学 ARM, 就是基于这个解释, 我们的学习不是仅仅学习一种 MCU, 而可以学到更多的各类协议和原理。

详细参数

尺寸: 74*145mm

PCB 工艺: A 级 PCB, 哑黑油墨, 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

基准电压: TL431 基准芯片

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

温湿度: DHT11

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、1 个 RGBLED、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

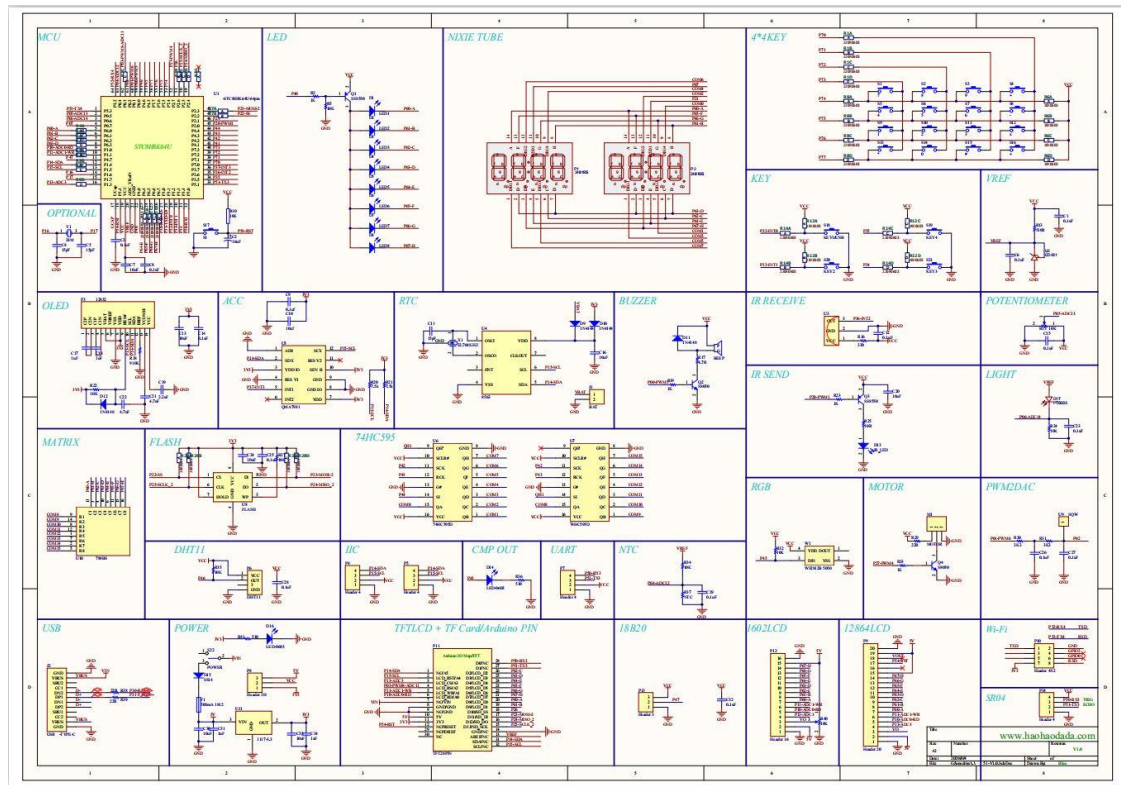
数码管: 2 个 4 位共阳数码管

点阵: 1 个 8*8 点阵

蜂鸣器: 1 个无源蜂鸣器

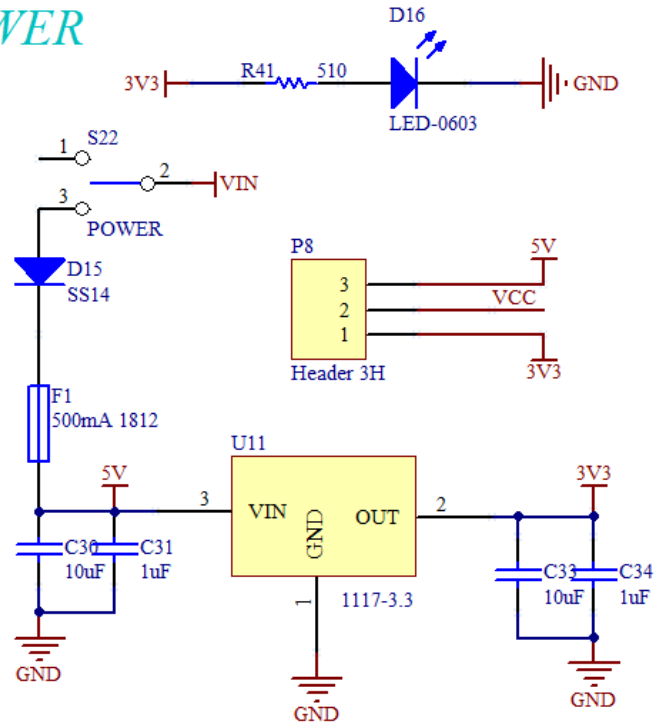
马达：1 个震动马达
超声波：可外接 SR04 超声波模块
Wi-Fi：可外接 ESP8266 Wi-Fi 模块

天问 51 原理图



电源模块

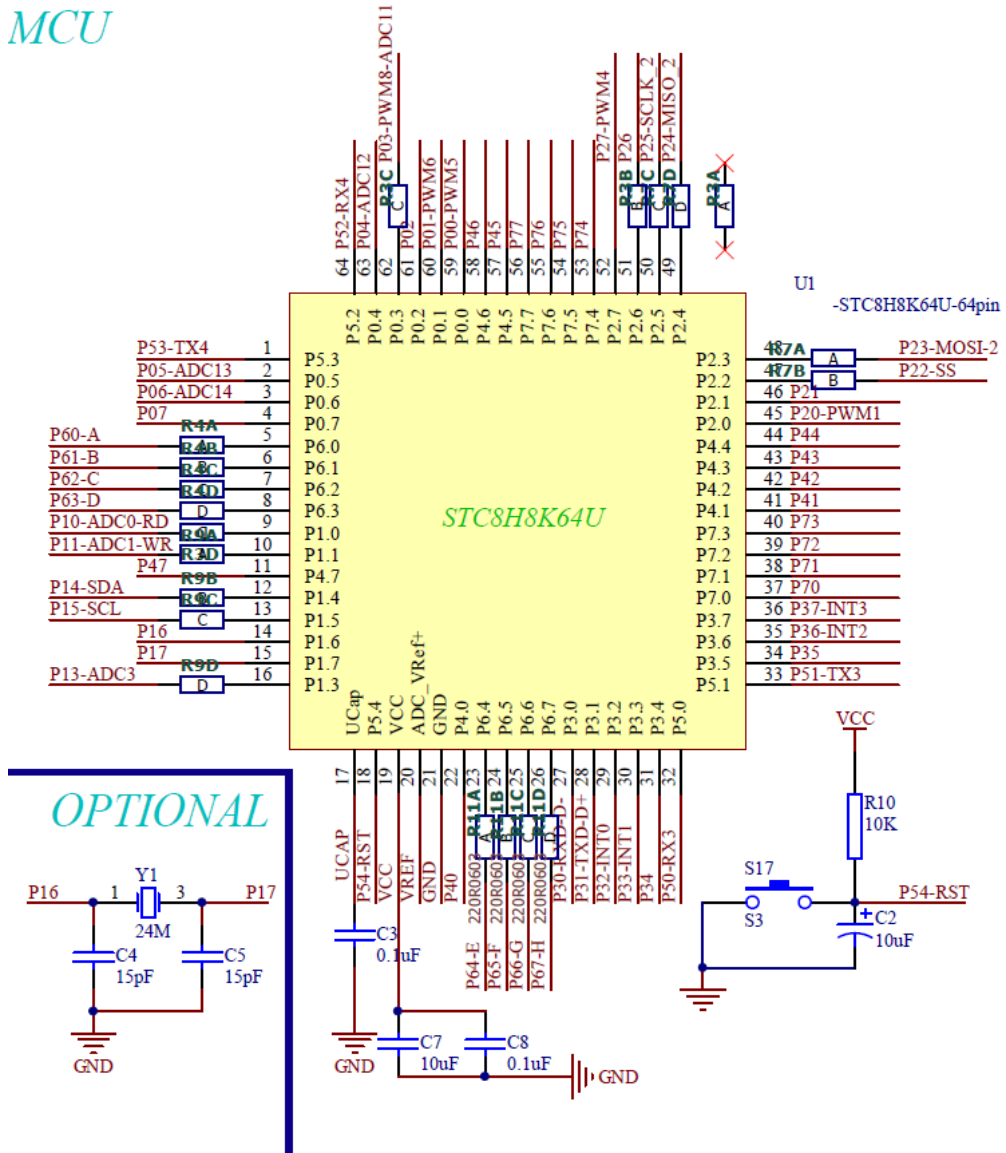
POWER



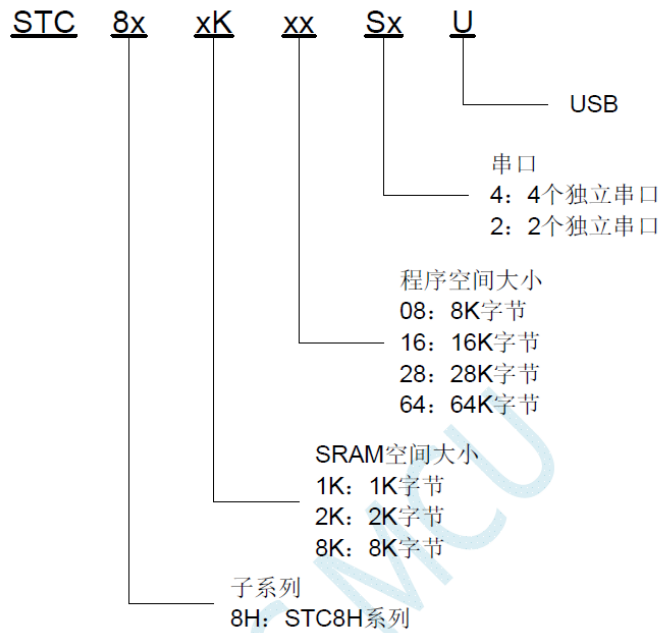
天问 51 的电源芯片采用 1117-3.3 作为 3.3V 电压的稳压，5V 电压直接采用 USB 口供电，所以 USB 口的供电不能接大于 5V 的电源，不然会烧坏后级电路。同时为了防止系统短路造成损坏，VIN 输入后级加了 500mA 自恢复保险丝。图中的 D15 二极管是为了防止电源反接。D16 的红色 LED 作为系统电源指示灯。系统有些设备的供电可以通过跳线帽来选择 3.3V 还是 5V 电源。

最小系统

MCU



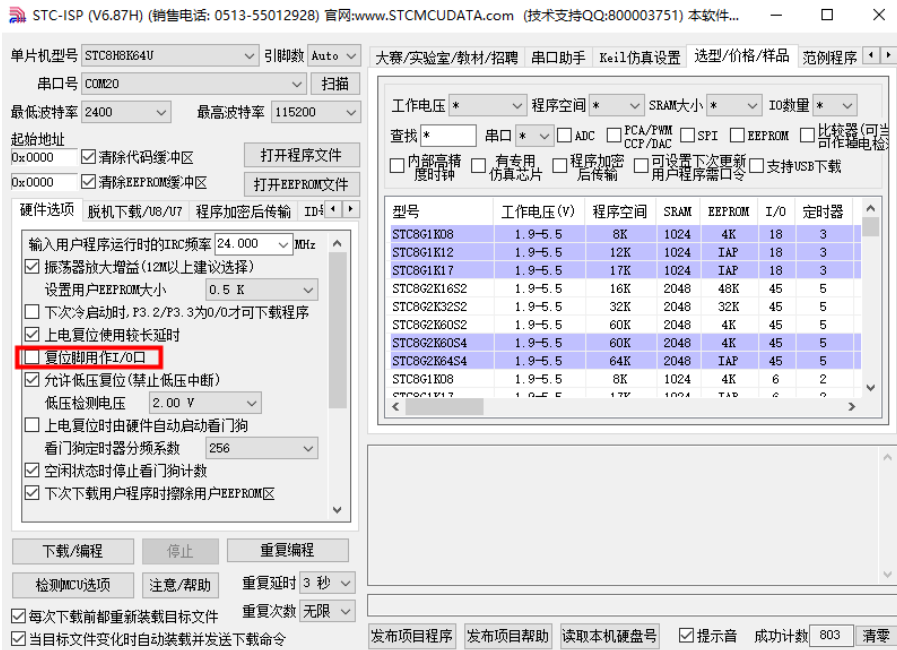
上图是 STC8H 的最小系统，单片机型号采用 STC8H8K64U，我们看下芯片的命名规则



从上图所知，天问 51 的 STC8H 芯片 SRAM 为 8K 字节，ROM 为 64K 字节，带硬件 USB。

STC8H 系列单片机是不需要外部晶振和外部复位的单片机，是以超强抗干扰/超低价/高速/低功耗为目标的 8051 单片机，在相同的工作频率下，STC8H 系列单片机比传统的 8051 约快 12 倍（速度快 11.2-13.2 倍），依次按顺序执行全部的 111 条指令，STC8H 系列单片机仅需 147 个时钟，而传统 8051 则需要 1944 个时钟。STC8H 系列单片机是 STC 生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/强抗静电/较强抗干扰的新一代 8051 单片机，超级加密。指令代码完全兼容传统 8051。

天问 51 开发板配置外部 24M 晶振，用户可以通过程序切换时钟源。



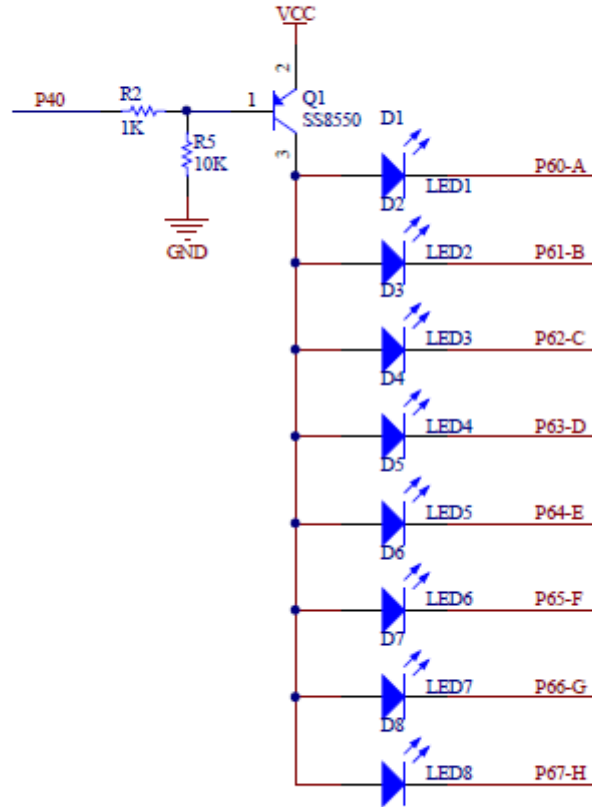
复位电路默认不启用，需要外部复位的时候需要在 ISP 下载软件里配置。

显示模块

原理图中，最复杂的总线就是 P6 口，在 P6 口上集结了 8 个 LED、8 个数码管、8*8 点阵、lcd1602 并口总线、lcd12864 并口总线、TFT 并口总线、Arduino 并口总线（对应 D2-D9）。从原理图获知 P6 口只能分时工作，也就是说这些设备同一时间只有一个可以正常工作，对于学习开发板来说，主要学习各设备的工作原理，影响不会很大。lcd1602、lcd12864、TFT 液晶屏、Arduino 并口总线由于采用插拔模式，分时工作比较容易理解，不要在同一时间接两种设备就可以分时工作了。8 颗 LED、8 个数码管、8*8 点阵的分时工作，采用了非常巧妙的电路和程序配合来解决。

8 颗 LED 的电路如下图：

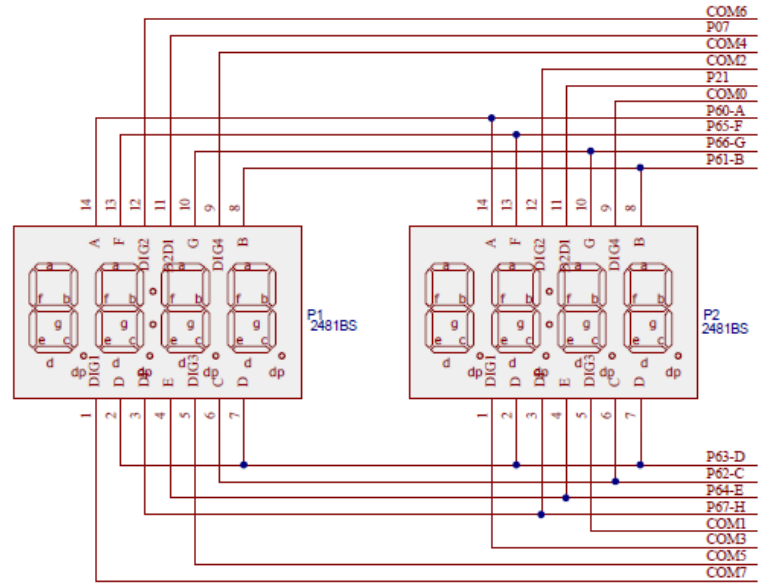
LED



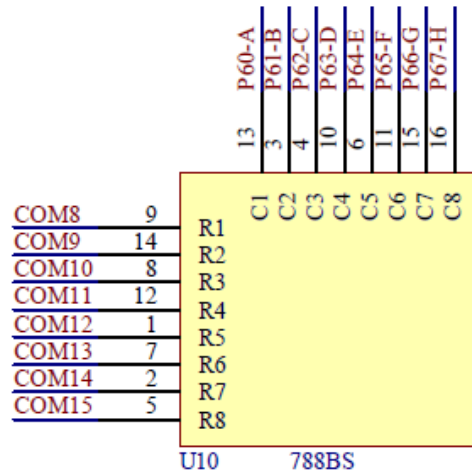
8 个 LED 采用共阳方式，连接到一公共 PNP 三极管驱动端，默认开机状态 P40 是高阻输入模式，PNP 三极管通过 R5 基极接入地，三极管工作于导通状态。P6 口只要写入 0，就能让 LED 发光。如果程序控制把 P40 设置成输出状态，输出高电平 1，PNP 三极管基极高电平，三极管截止就关断了 8 个 LED 的电源。从电路中可以看出，我们开机上电，可以直接控制 P6 的亮灭，通过 P40 可以随时关闭 LED 而不影响其他电路工作。

8 位数码管和 8*8 点阵模块采用共阳模块，共阳端口采用两个 HC595 串转并电路来驱动，一个 HC595 的输出为 COM0-COM7 控制 8 个数码管的公共端，另一个 HC595 的输出为 COM8-COM15 控制点阵模块的公共端。原理图如下：

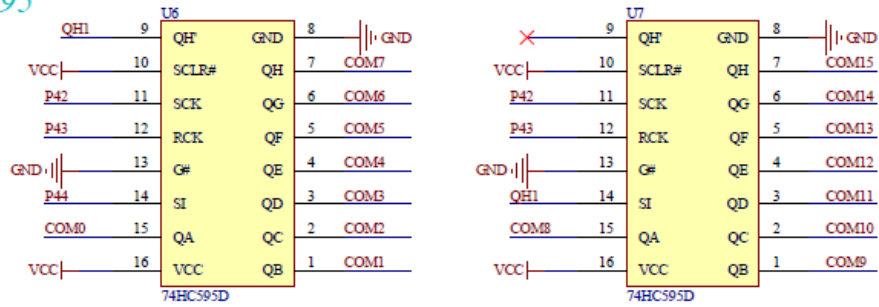
NIXIE TUBE



MATRIX



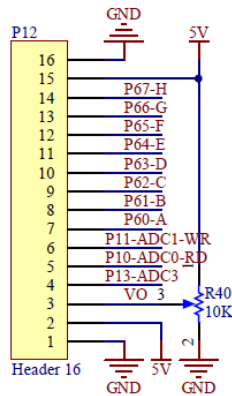
74HC595



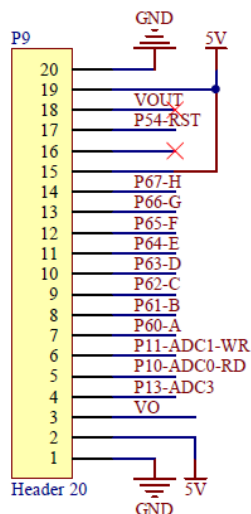
HC595 采用三线控制级联模式，三线就可以输出 COM0-COM15 每位的值，所有值输出为 0 时，由于模块采用共阳模块，数码管和点阵模块不管 P6 口高低电平都不会发光，达到关断数码管和点阵模块的作用。通过 HC595 输出不同的值可以随时打开数码管和点阵模块。

当 LED*8 公共端控制引脚 P40 输出高电平，HC595 所有端口输出低电平，就关闭了板载的 LED、数码管、点阵模块。这时就可以通过分时接插 lcd1602、lcd12864、TFT 液晶、Arduino 模块，由 P6 口随心所欲输出数据。

1602LCD



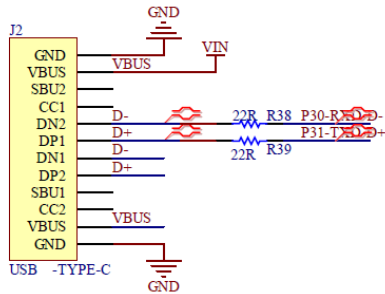
12864LCD



1602 和 12864 液晶屏的背光共用一个电位器来调节。

USB 模块

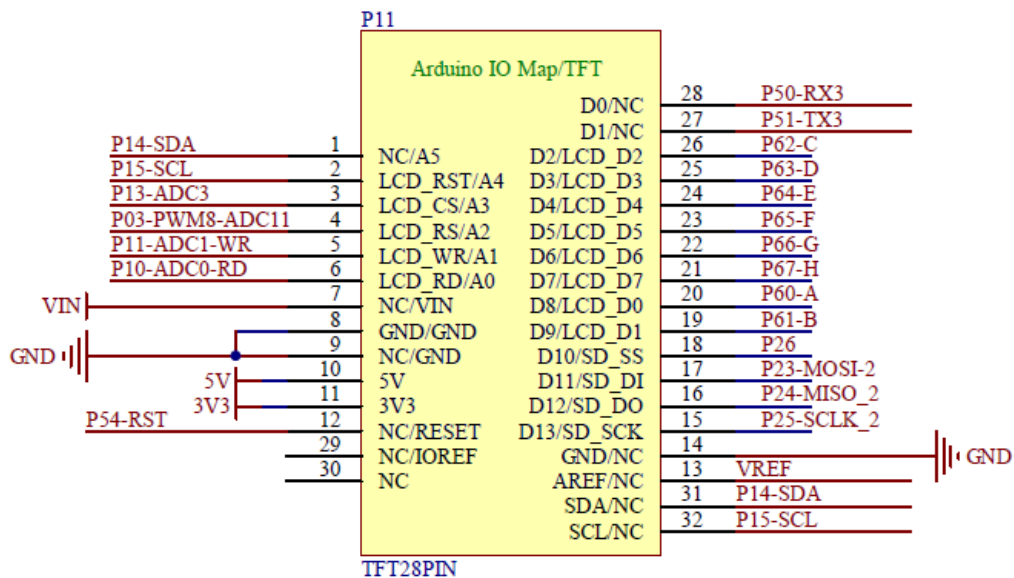
USB



STC8H 的 USB 接口和串口 1 用的是 P30、P31 两个相同引脚，所以天问 51 开发板上的程序下载电路没集成在开发板上，而是采用外接下载器来使用，这样 USB 的功能不会因为下载电路的影响而导致没法使用。

Arduino 扩展模块

TFTLCD + TF Card/Arduino PIN



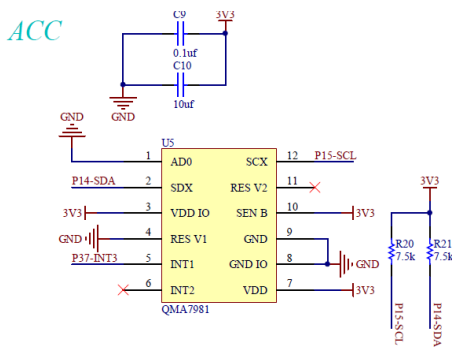
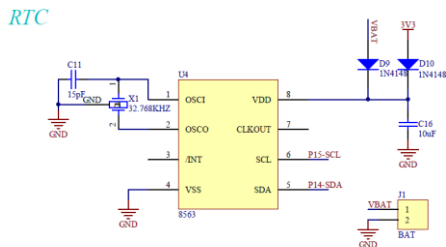
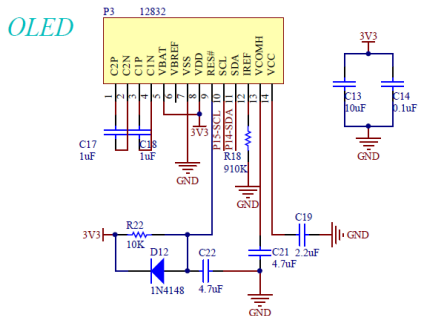
天问 51 的 Arduino 扩展接口，按照标准 Arduino 接口规范来布局。这样就可以用各种兼容 Arduino 的扩展板，包括天问 51 使用的彩屏模块。在使用的时候需要注意几个地方，P14-SDA、P15-SCL 的 I2C 功能同时还连着板载的 OLED、RTC、加速度模块，使用的时候需要注意 I2C 的地址不要冲突。同时这两个引脚不能再作为 Arduino 的模拟引脚 ADC 使用。D11、D12、D13 的 SPI 口连接内部的 SPI Flash 设备，建议作为 SPI 口来用，最好不要作为普通 IO 口来操作。P60-P67 和板载的 LED、数码管、点阵共用总线，避免冲突，使用时根据上节的原理关闭 LED、数码管和点阵模块。

I2C 模块

天问 51 的 I2C 模块全部连接在 P1.4 和 P1.5 引脚上，MCU 读写这些设备采用不同地址来进行操作，使用时注意设备的地址不要冲突，板载设备地址以下表为准。

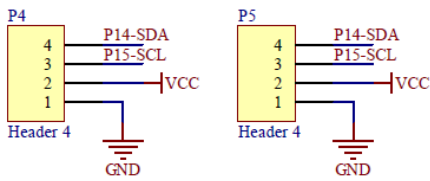
天问 51 板载 I2C 设备地址表

设备	地址
OLED	0x78
RTC	0xA2
加速度	0x12



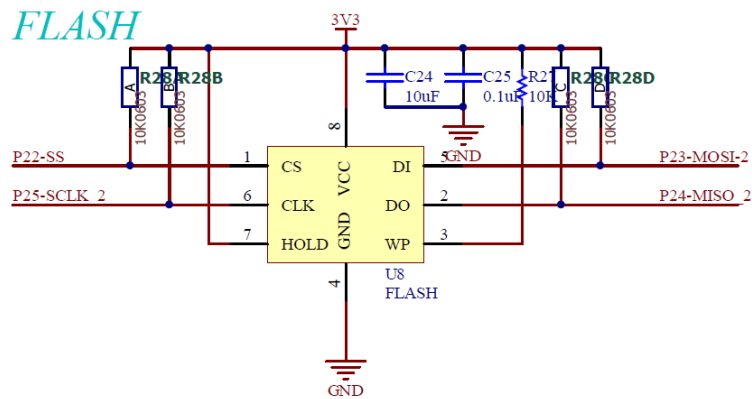
加速度模块可以设置为中断模式，中断引脚连接在 P3.7 引脚上，可以用外部中断 INT3 唤醒 MCU。

IIC



P4\P5 四针插座采用并联模式，方便外接 I2C 设备，外接 I2C 设备注意不要与板载 I2C 设备地址冲突。

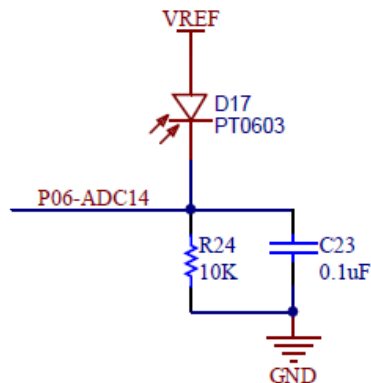
FLASH 模块



天问 51 内部的 Flash 模块采用 SPI 接口，使用了 P23-MOSI、P24-MISO、P25-SCLK、P22-SS 引脚。Flash 模块的 SPI 接口和 TFT 彩屏上 TF 卡 SPI 接口共用，其中 MOSI、MISO、SCLK 共用引脚，片选引脚 TF 卡是 P26 引脚，程序上采用片选不同来操作不同设备。

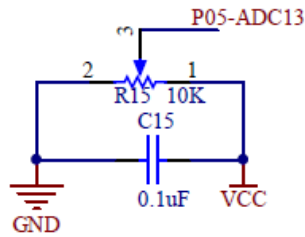
ADC 模块

LIGHT



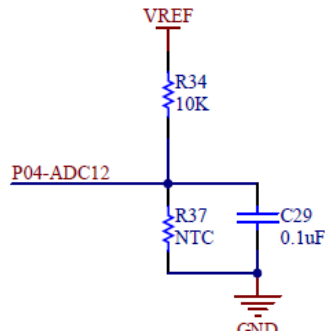
光敏模块，通过 AD 采样获取环境亮度。

POTENTIOMETER



电位器模块，通过 AD 采用获取电阻分压大小。

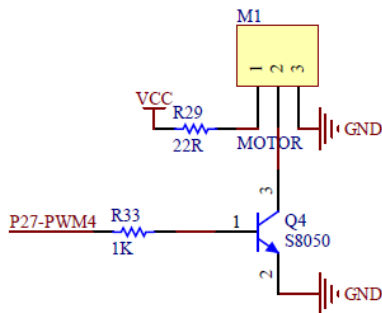
NTC



NTC 热敏电阻模块，通过 AD 采用来计算 NTC 当前电阻大小，从而计算出当前温度。

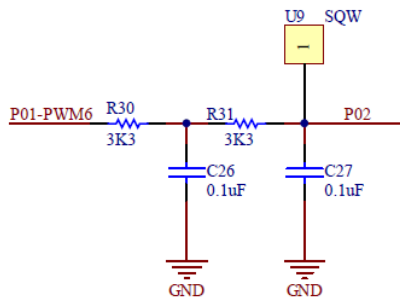
PWM 模块

MOTOR



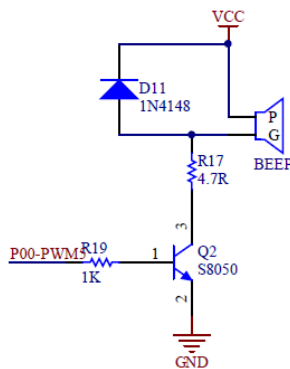
可以通过 PWM 来控制电机速度。

PWM2DAC



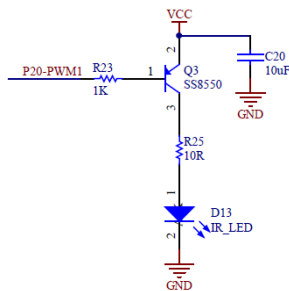
通过 PWM 和外部 RC 电路模型 DAC 输出。

BUZZER



通过 PWM 控制无源蜂鸣器发出不同频率的音调。

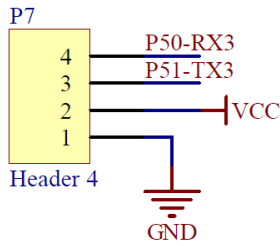
IR SEND



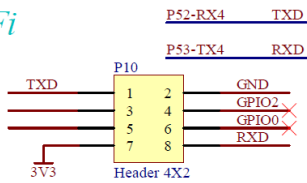
通过 PWM 来发送频率为 38KHz 的红外脉冲。

串口模块

UART



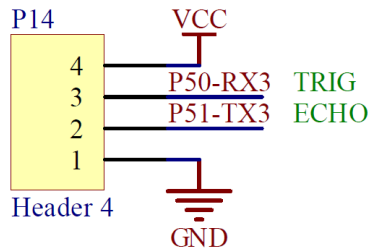
Wi-Fi



STC8H 包含 4 路串口，其中串口 1 和 USB 引脚共用，天问 51 额外引出了串口 3 作为扩展接口，串口 4 作为 Wi-Fi 接口。

超声波模块

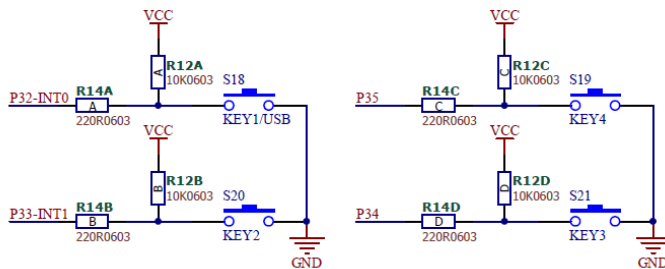
SR04



超声波模块的接口和串口 3 共用，使用的时候需要注意。

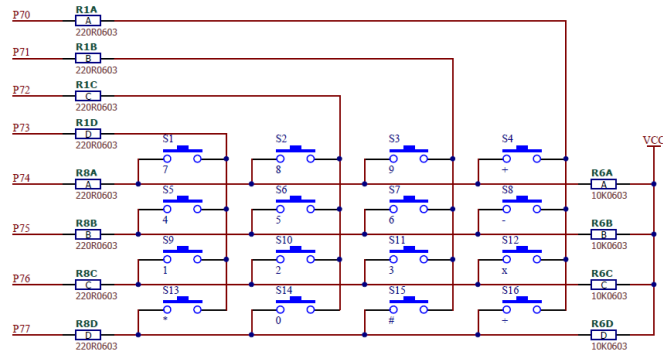
按键模块

KEY



4 路独立按键，其中 KEY1 和 KEY2 为外部中断 0 和 1。

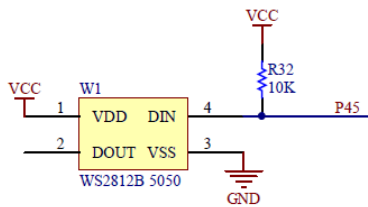
4*4KEY



矩阵键盘连接在 P7 端口，通过扫描方式获取按键值。

RGB 模块

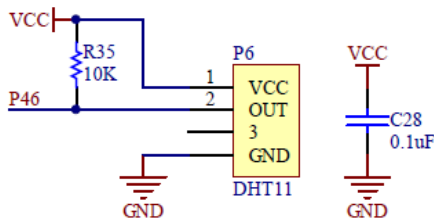
RGB



RGB 采用 WS2812 芯片，可以串联多个灯珠级联。

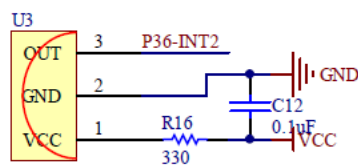
DHT11 温湿度模块

DHT11

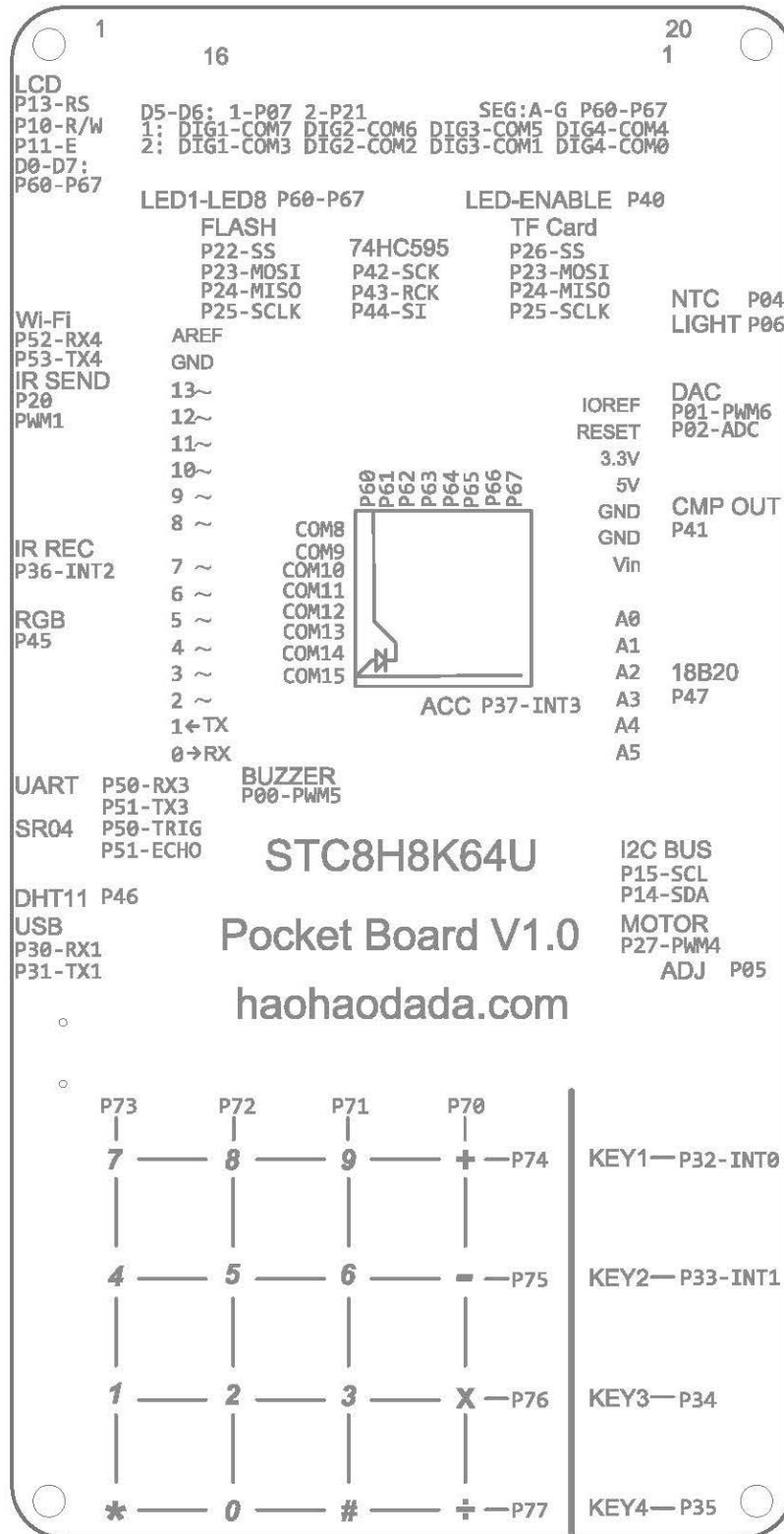


红外接收模块

IR RECEIVE

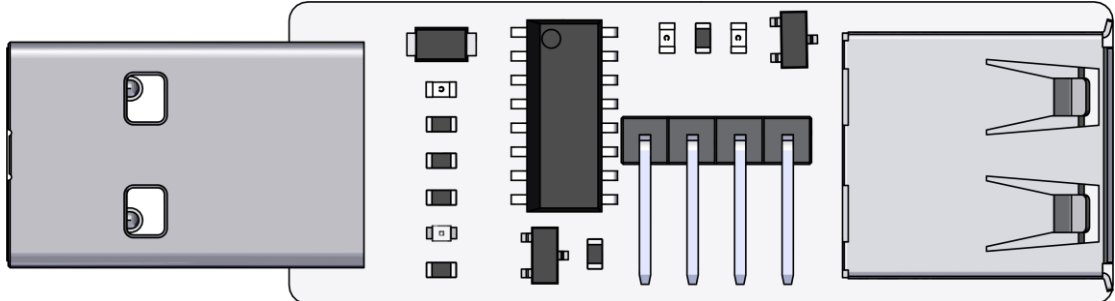


我们大致了解了开发板的硬件设计以后，编程时只需要关心模块对应的控制引脚，天问 51 开发板采用透明外壳，在主板背面都标明了对应的引脚编号，方便编程时查看。



STC-Link 下载器使用

STC-Link 支持硬件仿真和自动烧写 STC 芯片程序。同时板载 4P 排针，引出烧写口，方便用户使用 USB 转 TTL 工具或烧写、仿真其他 STC 设备。

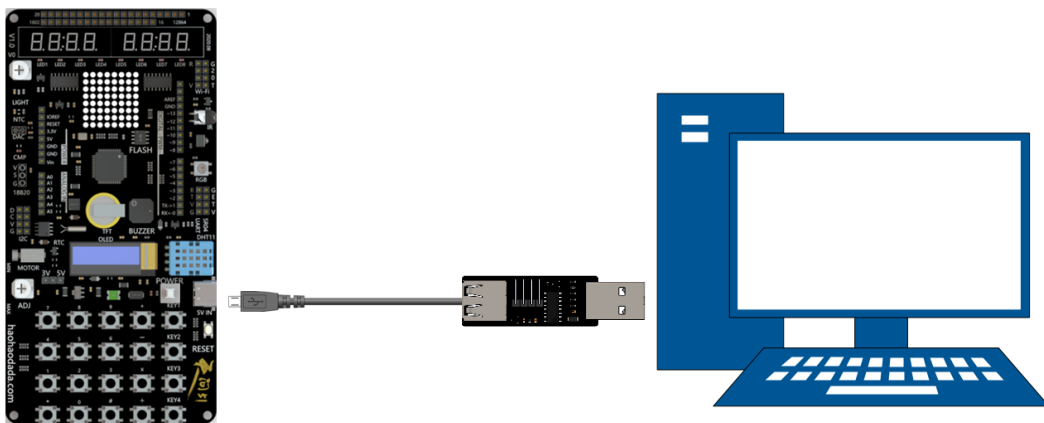


天问 51 采用 STC8H8K64U 芯片，带有 USB 功能，支持 USB 下载、串口下载、U 盘下载、无线下载程序四种方式。下面先以 U 盘下载方式为例。

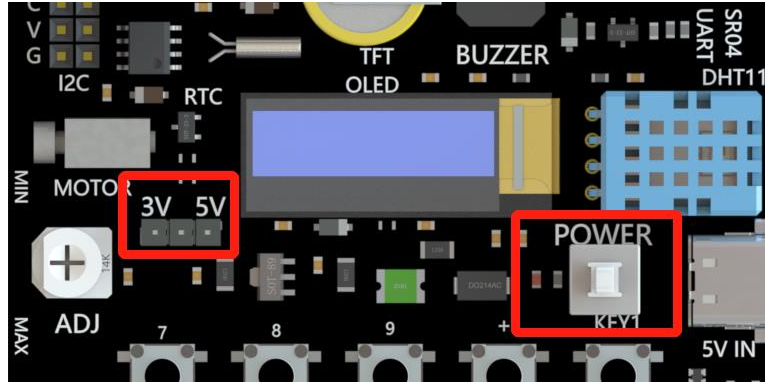
仿真功能只能在 Keil 软件中使用，因此在本教程不做说明，需要了解的可以去论坛查看对应资料。

运行第一个程序

1. 用 Type C 数据线连接天问开发板和 STC-Link 到电脑上。



2. 打开天问开发板的“POWER”电源键，确认旁边的红色电源指示亮起，同时确认 3/5V 系统电源选择跳线帽，插在 3V 或者 5V 端。



3. 打开 haohaodada.com 网站，找到资料页的天问 51 开发板栏目，进入天问 51 开发板资料专区



★ 天问51开发板学习资料专区



天问51开发板 全球第一款带USB的STC51全功能开发板，软硬件百分之百国产，支持图形化寄存器配置和Arduino接口，STC&好好搭搭联合出品。


[宣传视频](#) [硬件资料](#) [离线软件](#) [在线编程](#) [社区](#) [立即购买](#)

学习教程


配套资料

原理图	作者: 好好搭搭	芯片手册	作者: 好好搭搭
下载工具	作者: 好好搭搭	调试工具	作者: 好好搭搭
驱动	作者: 好好搭搭		


官方例程 [基础案例](#) [综合案例](#) [查看全部 >](#)




1.点亮一个LED灯
2020-08-20
Blue



2.点亮8个LED灯
2020-08-20
Blue



3.流水灯
2020-08-20
Blue

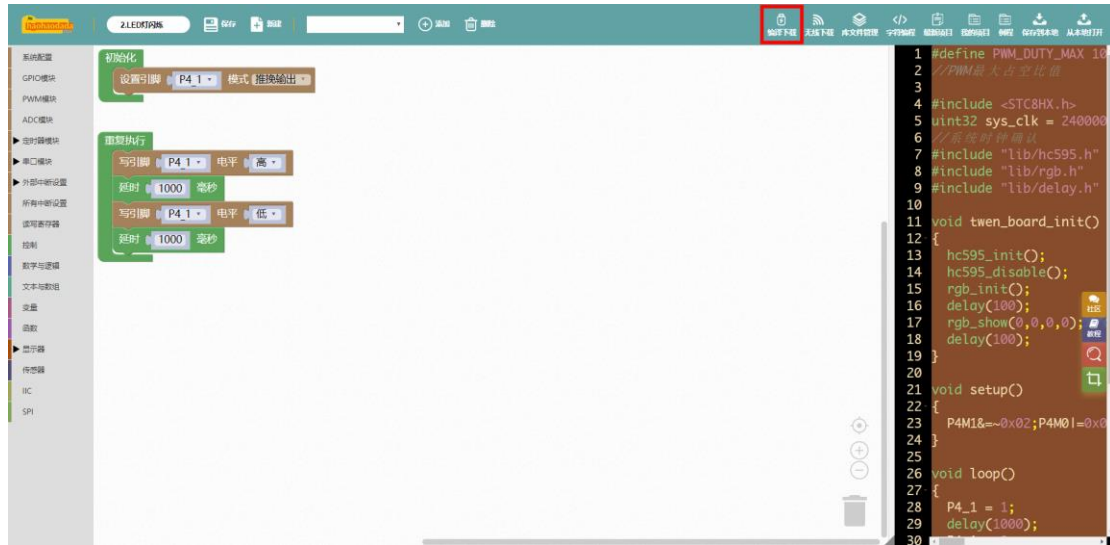


4.一位数码管显示数字
2020-08-20
Blue

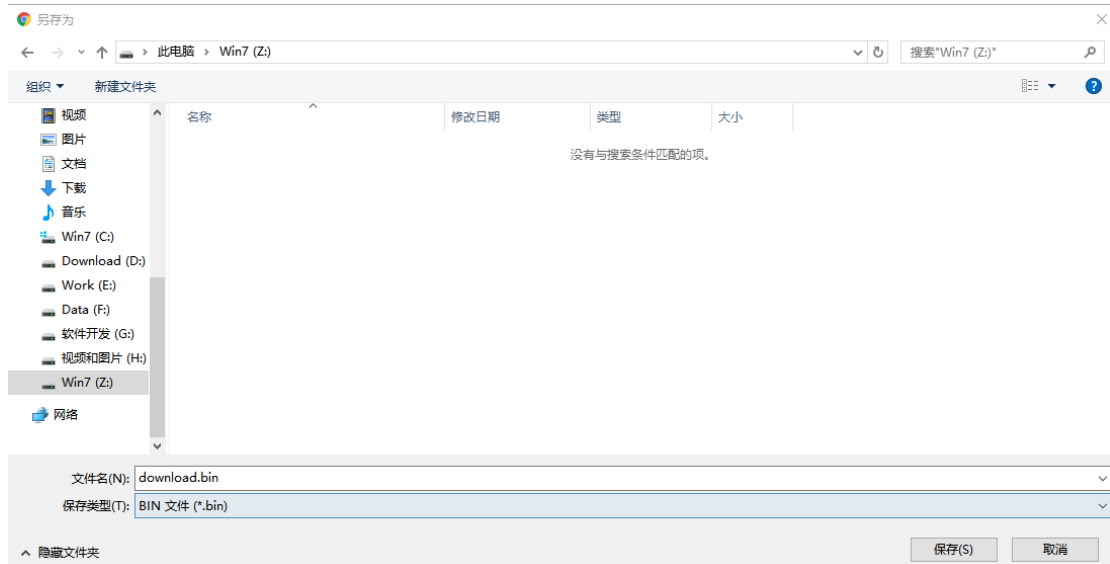
4. 进入“驱动”下载并安装“CP210x”驱动，进入“下载工具”下载并打开“TWEN-download.exe”。



驱动安装正常，串口会自动显示端口号，如果没有，请确认驱动安装是否正常。运行“TWEN-download.exe”会自动打开浏览器跳转到下图编程页面。



5. 软件会自动打开编程页面，天问 51 开发板的 P41 端口连接着一个 LED 灯，默认程序为一个 LED 闪烁程序，我们直接点击工具栏的“编译下载”，平台会自动云编译，编译完成后会提示保存 Bin 文件到电脑。



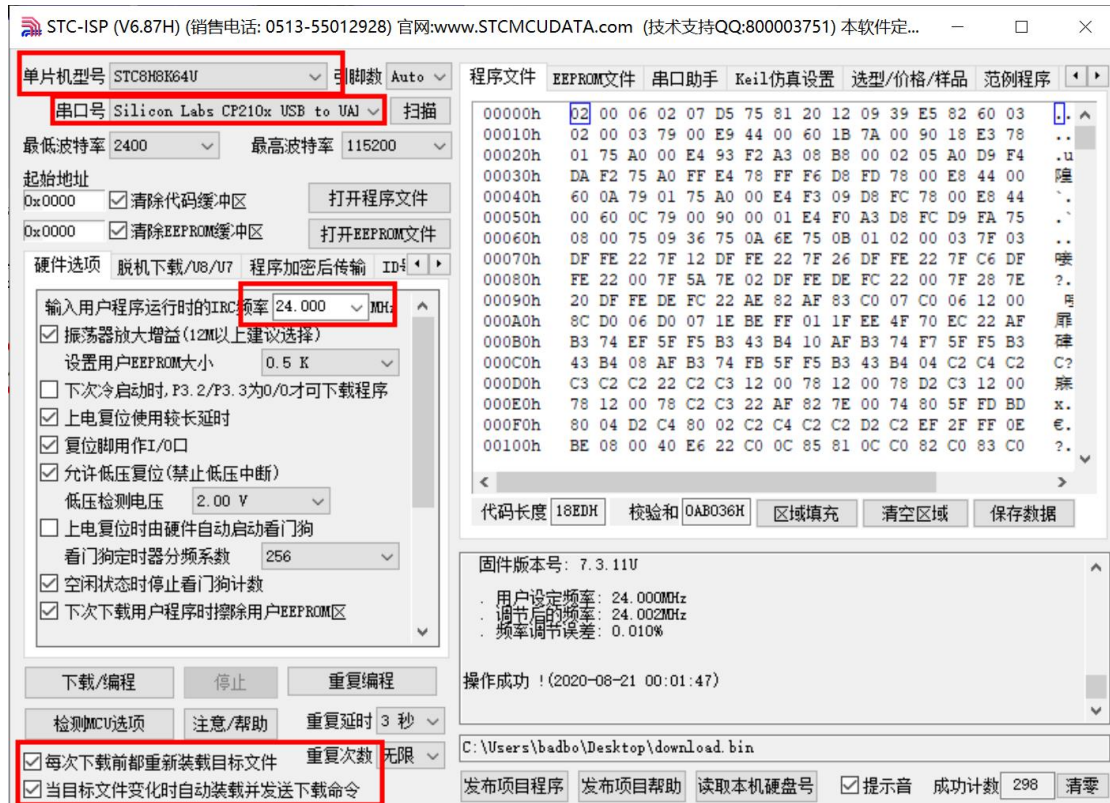
选择保存到电脑的“Z:”盘。部分电脑由于系统原因没有出现“Z”盘，请检查系统问题，如无法解决只能采用其他方式下载。

6. 烧录完成后，天问板上的“CMP”丝印标识的 LED 灯会开始闪烁。
7. 点击编程页面的“最新项目”可以看到每个人分享的作品，“例程”可以看到官方案例程序，“教程”可以看到官方教程，“社区”可以进入论坛讨论或分享。

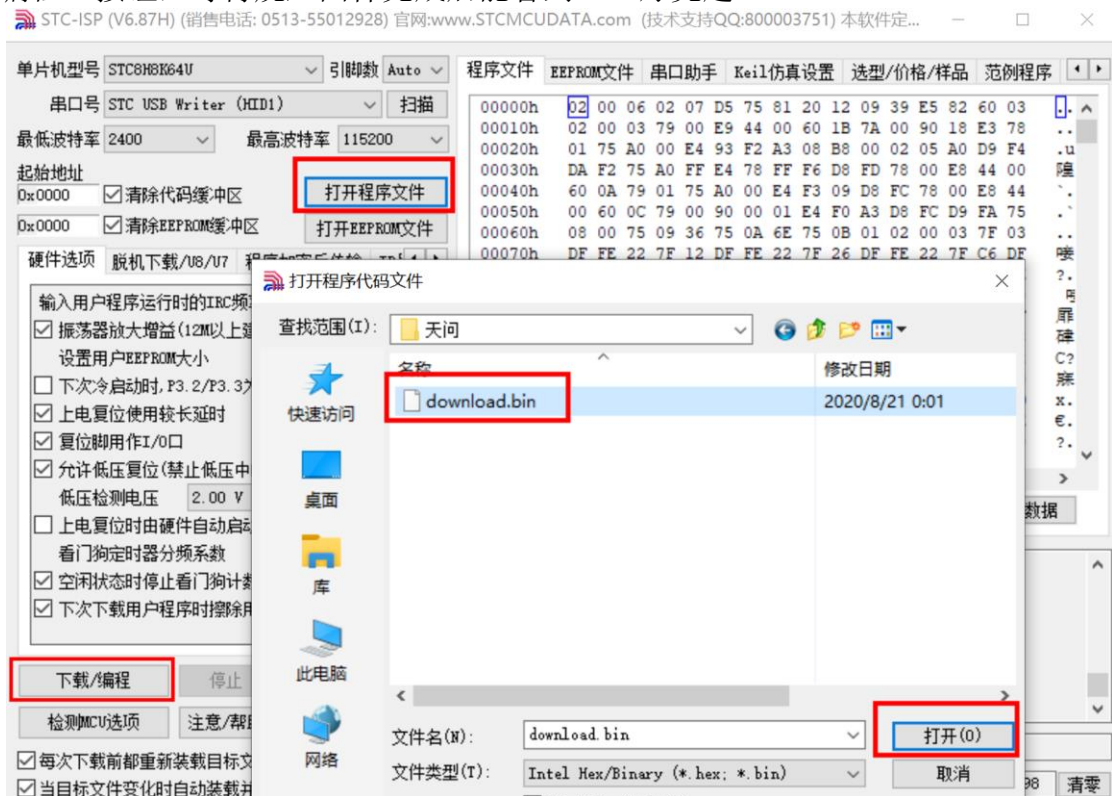
串口下载

1. 硬件连接参考前面的 U 盘下载模式，在资料页面下载并打开 STC-ISP 软件，单片机型号选择 STC8H8K64U，端口号选择刚才所示的带“CP210X”字样的端口号，

如下图所示，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。

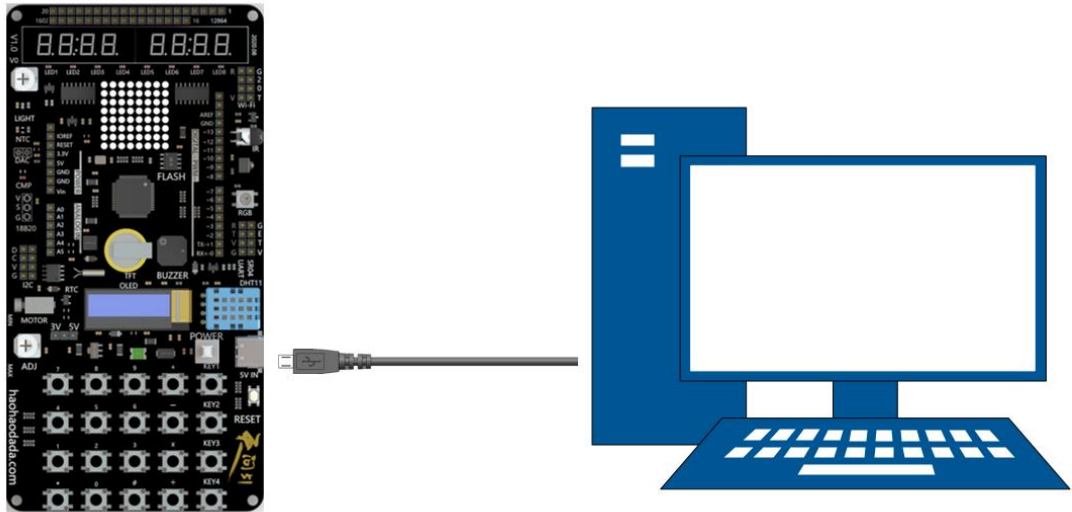


在 STC-ISP 软件里选择打开程序文件，打开平台编译保存的 Bin 文件，点击“下载/编程”按钮，等待烧入固件完成后能看到 LED 灯亮起。

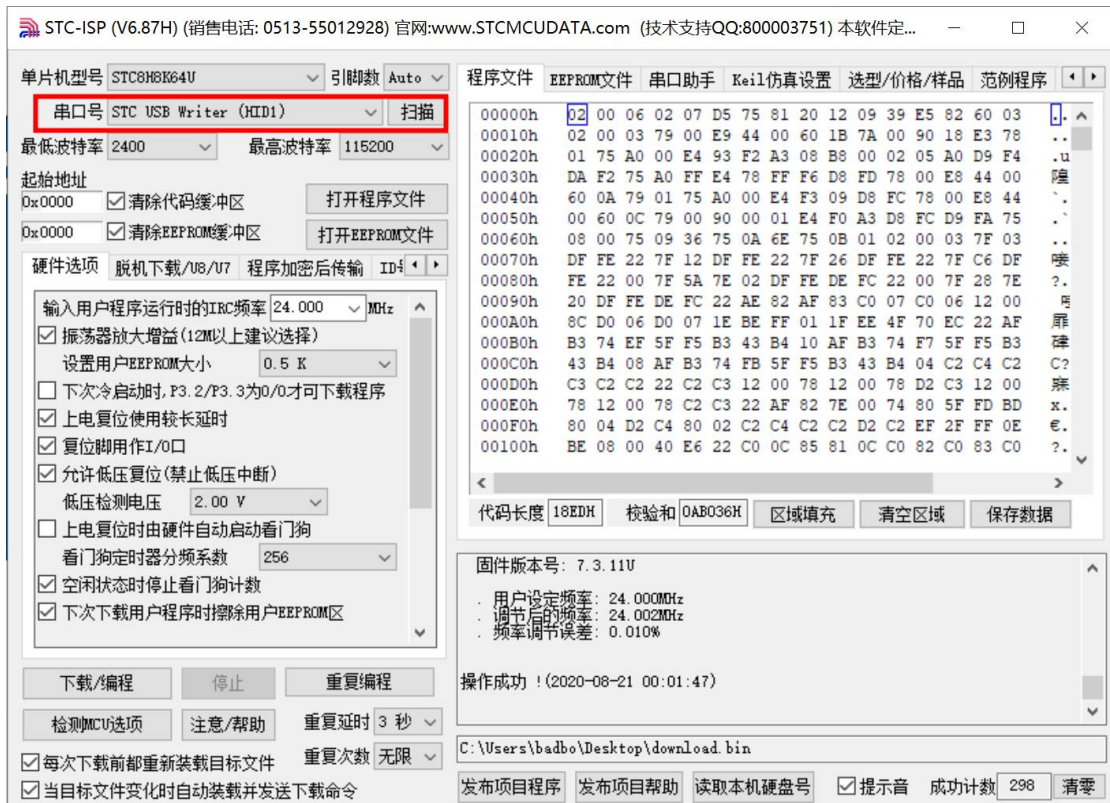


USB 下载方式

1. 用 Type C 数据线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”按钮配置中断后进入 ISP。这样后续只需要按一下“KEY1/USB”按钮就能进入下载模式，不再需要开关电源按钮。具体程序见下图。

无线下载方式

无线下载器是好好搭搭物联网技术产品，采用 mac 地址管理设备的模式，区分不同的下载器。无线下载器可以实现跨地域、跨网段远程下载到 STC8H 芯片中。另文介绍无线下载的设置和下载方法。

第二篇 平台篇

STC8 图形化设计思想

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 是一致的。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。

多年以来开发 51 程序大家都用 Keil C51 软件，资料齐全，案例众多，教材也是基本采用 Keil，Keil 成了大家不二的选择。Keil 软件正版需要几万元人民币，使用者会说有试用版、有 D 版等等。随着版权越来越重视，中美科技摩擦的加剧，大家用这种思维方式处理开发软件问题是很致命的，特别是嵌入式教育界。Keil 软件使用实际不简单，只不过用的人多了，推广比较不错，资料丰富，因此学习的人数较多了。

STC8H 的图形化编程系统由“好好搭搭在线”推出，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，输入后会自动列出所有 1602 有关的函数，减少你的记忆关键字量和出错。
6. 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. 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。

STC8 图形化界面介绍

图形化单片机在线编程平台网址，<http://haohaodada.com/C51/stc8.php>，目前有通用 51 版本和 STC8H 版本，后续会支持更多的芯片。

打开网址后，出现如下图界面：



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，还有编译下载、无线下载、库文件管理、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。

STC8 图形化编程的基本操作

1. 文件操作。

工具栏里有新建文件、重命名文件、保存文件、我的项目、保存到本地、从本地打开这几个图标，这些图标就构成了整个文件系统。工具栏如下图：



新建文件、重命名文件图标功能和我们常用文件操作功能一致，保存文件图标是指编程文件内容保存到云服务器上，文件名就是保存文件前面这个文本框的文本。云服务器上保存是以文件名为唯一识别标志的，也就是说同一文件名在同一帐号只有一个，文件名如果相同，服务器就会提醒替换原文件，这一点请大家特别注意。如果要把文件保存到本地计算机，请用“保存到本地”图标。如果要打开保存到本地的文件，请用“从本地打开”图标进行操作。

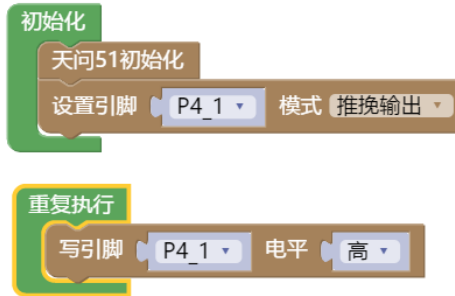
我的项目是你的“程序仓库”，单击“我的项目”图标后，就会进入你的“程序仓库”。可以管理你的程序，进行分享程序、删除程序操作，也可以修改你的程序封面，看起来更加生动。



2. 图形化程序的编写

编写图形化程序，先规划好需要实现的功能和程序逻辑，按所需功能到指令区的单片机配置模块找相应模块配置好单片机的 I/O 口、定时器、PWM、ADC、串

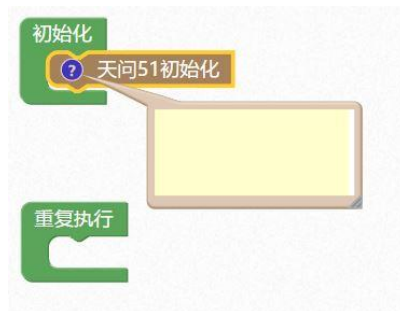
口和中断等。把所需指令拖动到初始化中，完成单片机的初始化配置。程序需要执行的指令按程序运行功能和逻辑把指令模块拖动到重复执行模块中。如下图所示：



指令删除：方法一、把不需要的指令模块拖动到“垃圾桶”图标删除；方法二、把不需要的指令模块拖动到最左边“指令区”删除；方法三、右击指令模块出现右键菜单删除。

指令禁用：在要设置的指令模块右击，在右键菜单中，选择“禁用块”，本条指令将无效不再生成C语言代码。如需重新启用生效，在指令模块上右击进入右键菜单选择“启用块”。

指令添加注释：在需要添加注释的指令模块右击，在右键菜单中，选择“添加注释”，模块左边就会出现一个“？”号，如下图：



指令的撤消和重做：方法一、撤消-快捷键 Ctrl+Z，重做-快捷键 Ctrl+Y；方法二、点击工具栏的撤消和重做图标。

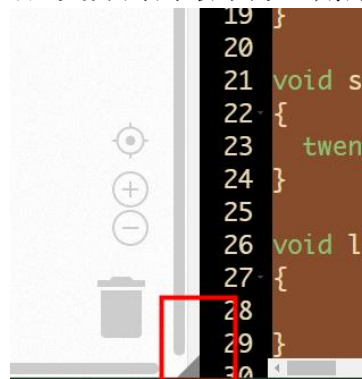
在图形模块中插入C语言：在“读写寄存器模块”中，有一个“输入代码”指令，可以在这个指令中直接输入C语言代码，输入格式和C语言编程一致。如下图：



在图形化模块中插入汇编语言：在图形化模块中插入汇编语言与插入 C 语言的指令一样也是用“输入代码”指令。输入的格式，开始行：“asm”，中间输入汇编程序内容，结束行：“endasm;”。注意双引号不是输入内容，如下图所示：

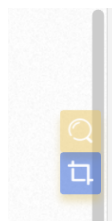


图形化编程界面中，可以编程区可以看到图形化模块指令，字符代码区可以看到生成的 C 语言代码。图形化编程区和字符代码区的，左右比例可以调整大小。调整按钮在编程区和字符代码区交界的最下方三角形。如下图所示：



可以通过单击工具栏的字符编程图标切换到字符编程界面。

图形化编程截图功能：图形化编程界面中，如需截取所编程程序模块透明图，可以使用“截图”工具。截图工具图标，位于编程界面的最右侧，和查找图标两个在一起。如图：



STC8 字符式编程基本操作

在图形化编程界面下，字符代码区为只读状态，不能输入代码。如需转入代码编辑状态，可直接单击工具栏上“字符编程”图标，编程界面切换到专业字符编程模式。在专业字符编程模式下，修改了原有程序想回到图形化编程界面，点击工具栏”图形编程“，”图形编程“和”字符编程“图标是同一个位置的图标，它会因状态不同而自动切换。专业字符编程模式下输入的代码，回到图形化编程模式，输入代码会被取消，专业字符编程模式只能保存成代码。

字符编程模式，有关键字提醒和代码补全功能，非常方便。右侧有查找和替换图标方便查找相应代码，更改代码。

STC8 库管理功能

点击工具栏”库文件管理“图标，出现私有库对话框，可以在对话框中操作建立库文件夹和上传库文件操作。库文件格式为”xxx.h”头文件格式，文件格式和 C 语言同文件一致。头文件的引入非常简单，上传到私有库中后，在根目录下，直接“#include <xxx.h>”。如果在私有库中建立了文件夹，引入头文件加上文件夹名就可以，例建立文件夹 mic，引入头文件就可以“#include <mic/xxx.h>”。库文件和库文件只支持英文，并且不能有任何非法字符。

第三篇 初级篇（基本模块使用）

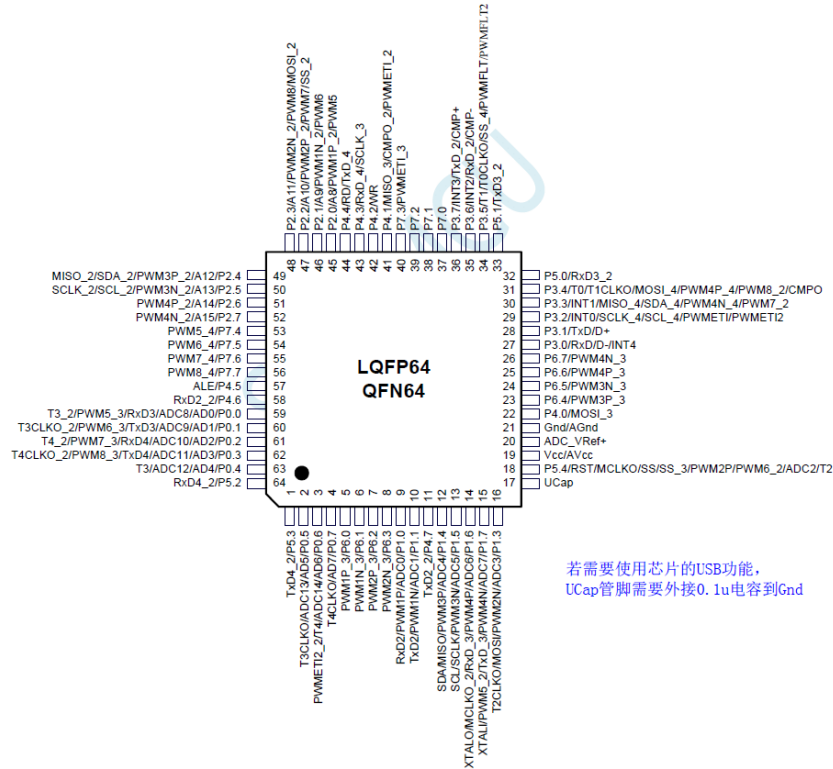
第一章 STC8H 单片机 IO 模块

第一节 单片机 IO 模块综述

我们已经编译下载并顺利的运行了一个 LED 灯闪烁的程序，看到 LED 灯一秒亮、一秒灭重复运行。通过查看原理图，我们知道这个 LED 灯连接着单片机的 P41 引脚。单片机里程序控制这个 P41 引脚一秒输出高电平，一秒输出低电平，从而让我们的 LED 灯闪烁。这里的 P41 引脚是单片机的其中一个 GPIO 口。

GPIO（英语：General-purpose input/output），通用型之输入输出的简称，是单片机和外部电路设备交流的必经之路。GPIO 口的状态由单片机内部的特定寄存器控制，这些寄存器我们可以想象成单片机里一个个小的开关，这些开关通过不同的组合，来控制内部电路实现不同的功能，而这些开关我们可以通过程序来设定。

天问 51 的 STC8H8K64U 芯片，含有 61 个 GPIO：P0.0-P0.7、P1.0-P1.7（无 P1.2）、P2.0-P2.7、P3.0-P3.7、P4.0-P4.7、P5.0-P5.4、P6.0-P6.7、P7.0-P7.7。我们通过命名方式，可以发现这些引脚被分成了 8 组，分别为 P0、P1、P2、P3、P4、P5、P6、P7，我们把这些叫作端口。每个 GPIO 我们称作引脚。



仔细观察上图后，我们会发现每个引脚后面都加了一些备注，类似“PWM4N_3”、“ADC8”、“RxD4_2”，这些是什么意思呢？我们的GPIO口除了可以输出高低电平外，GPIO都还有一些复用的功能，这些复用功能主要有：

- 读取模拟量数据：AD_x
- 输出PWM脉冲：PWM_xP_x
- 外部中断：INT_x
- 通用串行总线：RxD_x/_x
- I2C总线：SDA_x/SCL_x
- SPI总线：MISO_x/MOSI_x/SCLK_x/SS_x
- USB总线：D+/D-
- 其它特殊功能

这些功能具体什么和怎么用，我们后续章节里用到时再展开讲。具体的每个引脚对应哪些功能，详见芯片手册的管脚说明部分，如下图

管脚说明

编号		名称	类型	说明
LQFP64/QFN64	LQFP48/QFN48			
1	1	P5.3	I/O	标准 IO 口
		TxD4_2	O	串口 4 的发送脚
2	2	P0.5	I/O	标准 IO 口
		AD5	I	地址总线
		ADC13	I	ADC 模拟输入通道 13
		T3CLKO	O	定时器 3 时钟分频输出
3	3	P0.6	I/O	标准 IO 口
		AD6	I	地址总线
		ADC14	I	ADC 模拟输入通道 14
		T4	I	定时器 4 外部时钟输入
		PWMETI2_2	I	增强 PWM 的外部异常检测脚
4	4	P0.7	I/O	标准 IO 口
		AD7	I	地址总线
		T4CLKO	O	定时器 4 时钟分频输出
5		P6.0	I/O	标准 IO 口
		PWM1P_3	I/O	PWM1 的捕获输入和脉冲输出正极
6		P6.1	I/O	标准 IO 口
		PWM1N_3	I/O	PWM1 的捕获输入和脉冲输出负极
7		P6.2	I/O	标准 IO 口
		PWM2P_3	I/O	PWM2 的捕获输入和脉冲输出正极

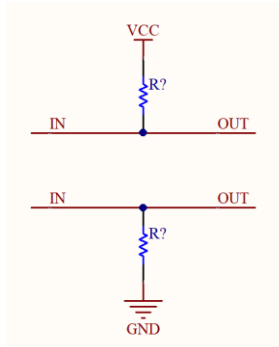
传统 51 芯片，引脚默认配置为准双向 IO 可以直接读写就可以输入或输出，STC8H 系列单片机，引脚除 P3.0 和 P3.1 外默认配置为高阻输入，只能读入不能输出，需要设置才能切换为其他模式。所有的引脚可以配置为 4 种工作模式：

- 准双向口（标准 8051 输出口模式）；
- 推挽输出；
- 高阻输入；
- 开漏输出。

可使用软件操作相应寄存器对 I/O 口的工作模式进行配置。

注意：除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式，另外每个 I/O 均可独立使能内部 4K 上拉电阻。

在开始讲 4 种工作模式前，我们先说下上拉和下拉。

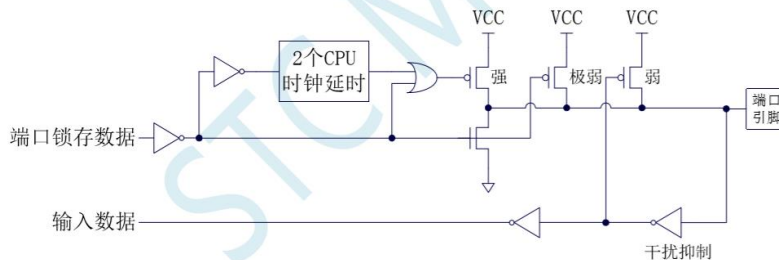


如上图，IN 端为单片机的引脚，通过一个电阻连接到了电源端，我们叫作上拉，如果连接到地，我们叫作下拉。上拉模式下，如果引脚配置为输出状态，并且输出低电平，OUT 端电平为低。如果输出高电平或者不输出，OUT 端都为高电平；如果引脚配置为输入状态，并且 OUT 端为高电平或者没电平，则 IN 端为高电平，如果 OUT 端为低电平，则 IN 端为低电平。而这里的电阻阻值的大小，决定了电流的大小，决定了驱动能力大小。这里的电流如果是从外部流向引脚内部的，我们叫作“灌电流”；如果从引脚内部流向外部，我们叫作“拉电流”。下拉同理。不过单片机内部的引脚不是一个简单的电阻，而是有很多晶体管、触发器、锁存器等数字电路组成。

接下来我们来看看四种模式的内部电路和几种模式的用途、区别。

准双向口（标准 8051 输出口模式）

弱上拉, 灌电流可达 20mA, 拉电流为 270~150 μ A（存在制造误差）。



准双向口（弱上拉）输出类型可用作输出和输入功能而不需重新配置端口输出状态。这是因为当端口输出为 1 时驱动能力很弱，允许外部装置将其拉低。当引脚输出为低时，它的驱动能力很强，可吸收相当大的电流。准双向口有 3 个上拉晶体管适应不同的需要。

在 3 个上拉晶体管中，有 1 个上拉晶体管称为“弱上拉”，当端口寄存器为 1 且引脚本身为 1 时打开。此上拉提供基本驱动电流使准双向口输出为 1。如果一个引脚输出为 1 而由外部装置下拉到低时，弱上拉关闭而“极弱上拉”维持开状态，为了把这个引脚强拉为低，外部装置必须有足够的灌电流能力使引脚上的电压降到阈值电压以下。对于 5V 单片机，“弱上拉”晶体管的电流约 250 μ A；对于 3.3V 单片机，“弱上拉”晶体管的电流约 150 μ A。

第 2 个上拉晶体管，称为“极弱上拉”，当端口锁存为 1 时打开。当引脚悬空时，这个极弱的上拉源产生很弱的上拉电流将引脚上拉为高电平。对于 5V 单

片机，“极弱上拉”晶体管的电流约 18uA；对于 3.3V 单片机，“极弱上拉”晶体管的电流约 5uA。

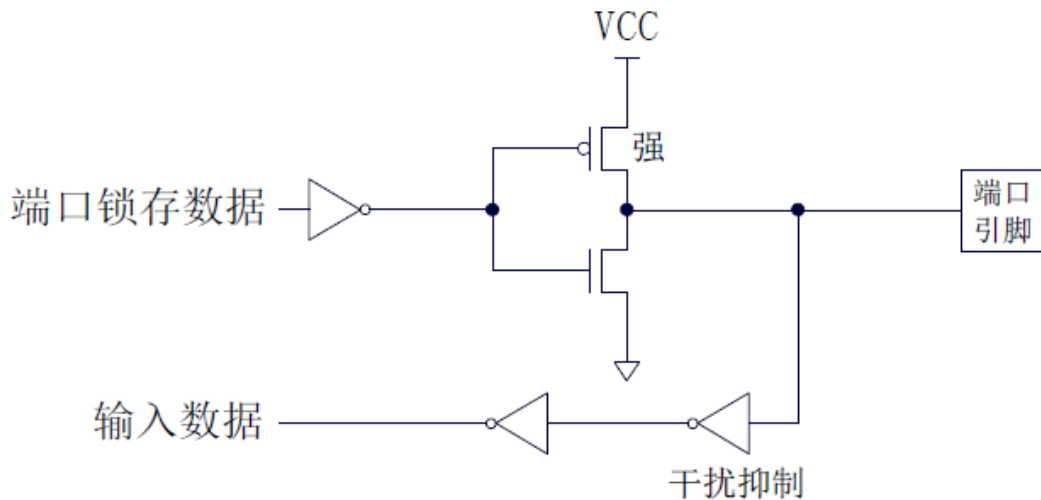
第 3 个上拉晶体管称为“强上拉”。当端口锁存器由 0 到 1 跳变时，这个上拉用来加快准双向口由逻辑 0 到逻辑 1 转换。当发生这种情况时，强上拉打开约 2 个时钟以使引脚能够迅速地上拉到高电平。

准双向口（弱上拉）带有一个施密特触发输入以及一个干扰抑制电路。准双向口（弱上拉）读外部状态前，要先锁存为 ‘1’，才可读到外部正确的状态。

大部分场合用这种模式，因为既可以做输入也可以做输出，不需要切换。

推挽输出

强上拉输出，可达 20mA，要加限流电阻。

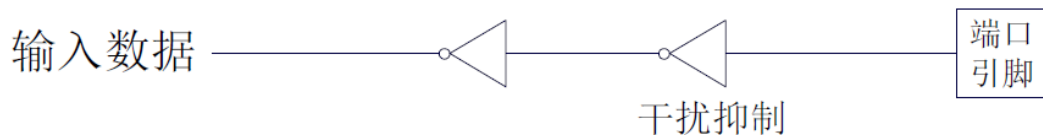


强推挽输出配置的下拉结构与开漏输出以及准双向口的下拉结构相同，但当锁存器为 1 时提供持续的强上拉。推挽模式一般用于需要更大驱动电流的情况。

常用在需要驱动能力大的地方，比如点亮 LED 灯等。

高阻输入

电流既不能流入也不能流出

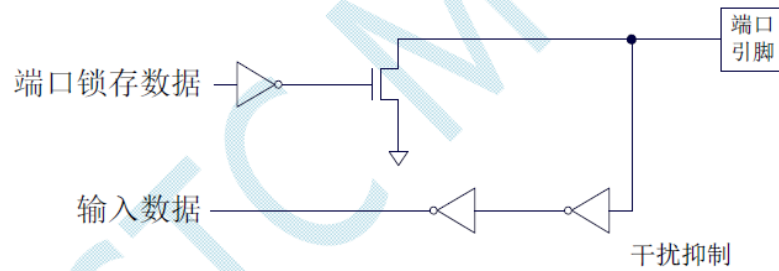


输入口带有一个施密特触发输入以及一个干扰抑制电路。

单片机复位后默认为高阻输入，这样引脚不会影响外部电路，也不会被外部电路干扰。

开漏输出

内部上拉电阻断开



开漏端口带有一个施密特触发输入以及一个干扰抑制电路。

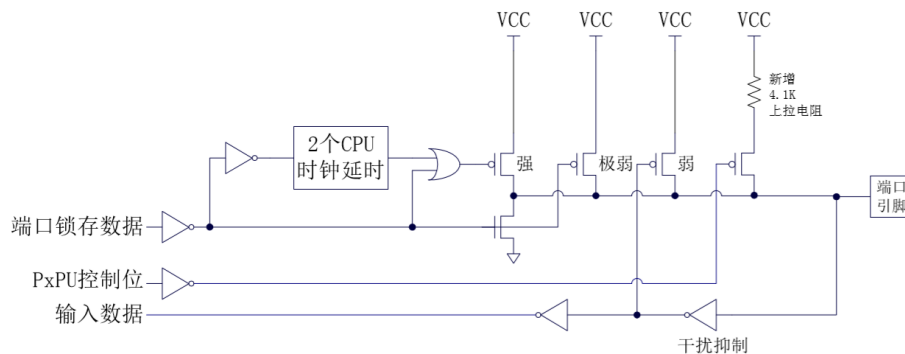
开漏模式既可读外部状态也可对外输出（高电平或低电平）。如要正确读外部状态或需要对外输出高电平，需外加上拉电阻。

当端口锁存器为 0 时，开漏输出关闭所有上拉晶体管。当作为一个逻辑输出高电平时，这种配置方式必须有外部上拉，一般通过电阻外接到 Vcc。如果外部有上拉电阻，开漏的 I/O 口还可读外部状态，即此时被配置为开漏模式的 I/O 口还可作为输入 I/O 口。这种方式的下拉与准双向口相同。

常用在 I2C 等可以级联的总线上。

4. 1K 上拉

STC8 系列所有的 I/O 口内部均可使能一个大约 4.1K 的上拉电阻（由于制造误差，上拉电阻的范围可能为 3K~5K）



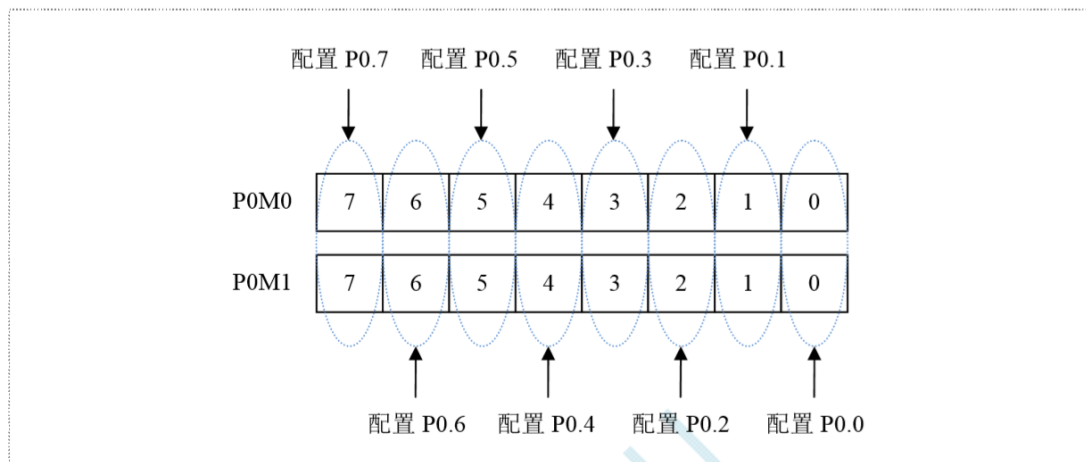
注意

虽然每个 I/O 口在弱上拉（准双向口）/强推挽输出/开漏模式时都能承受 20mA 的灌电流（还是要加限流电阻，如 1K、560Ω、472Ω 等），在强推挽输出时能输出 20mA 的拉电流（也要加限流电阻），但整个芯片的工作电流推荐不要超过 70mA，即从 Vcc 流入的电流建议不要超过 70mA，从 Gnd 流出电流建议不要超过 70mA，整体流入/流出电流建议都不要超过 70mA。

端口模式配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1
P0M1	93H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0
P1M0	92H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1
P1M1	91H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0
P2M0	96H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1
P2M1	95H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0
P3M0	B2H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1
P3M1	B1H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0
P4M0	B4H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1
P4M1	B3H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0
P5M0	CAH	-	-	-	P54M1	P53M1	P52M1	P51M1	P50M1
P5M1	C9H	-	-	-	P54M0	P53M0	P52M0	P51M0	P50M0
P6M0	CCH	P67M0	P66M0	P65M0	P64M0	P63M0	P62M0	P61M0	P60M0
P6M1	CBH	P67M1	P66M1	P65M1	P64M1	P63M1	P62M1	P61M1	P60M1
P7M0	E2H	P77M0	P76M0	P75M0	P74M0	P73M0	P72M0	P71M0	P70M0
P7M1	E1H	P77M1	P76M1	P75M1	P74M1	P73M1	P72M1	P71M1	P70M1

每个引脚的配置都需要使用两个寄存器来设置。以 P0 口为例，配置 P0 口需要使用 P0M0 和 P0M1 两个寄存器进行配置，如下图所示：



即 P0M0 的第 0 位和 P0M1 的第 0 位组合起来配置 P0.0 口的模式
 即 P0M0 的第 1 位和 P0M1 的第 1 位组合起来配置 P0.1 口的模式
 其他所有 I/O 的配置都与此类似。

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

端口数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
P1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
P3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
P4	C0H	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
P5	C8H	-	-	-	P5.4	P5.3	P5.2	P5.1	P5.0
P6	E8H	P6.7	P6.6	P6.5	P6.4	P6.3	P6.2	P6.1	P6.0
P7	F8H	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0

读写端口状态

写 0：输出低电平到端口缓冲区

写 1：输出高电平到端口缓冲区

读：直接读端口管脚上的电平

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

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

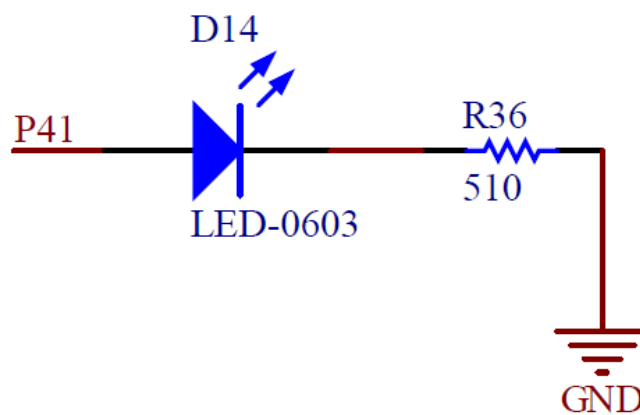
上述这些寄存器，就是前面说到过的芯片内部的一些小开关，对这些寄存器的对应位置写 0 或者 1，就是设置了相应的功能。

第二节 点亮一个 LED

1. STC8H GPIO 输出模式

天问 51 的 P41 引脚连接着一个 LED 灯，我们把 P41 引脚配置成推挽输出，同时输出高电平，就可以把 LED 灯点亮。

2. 硬件设计



红色 LED 灯为发光二极管，导通压降为 1.82-1.88V，电流 5-8mA。如果我们系统电源设置为 5V，P41 输出高电平，这里压降取 1.88，则流过 LED 的电流为 $(5-1.88)/510=0.0061A=6mA$ 。

3. 图形化编程



4. 字符式编程

```
#include <STC8HX.h> //引入 STC8H 头文件

void setup()
{
    P4M1&=-0x02;P4M0|=0x02; //推挽输出
    P4_1 = 1; //设置 P41 引脚高电平
}

void loop()
{
}

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

C 语言

上图中的这些代码就是 C 语言，C 语言是一种通用的高级语言，最初是由丹尼斯·里奇在贝尔实验室为开发 UNIX 操作系统而设计的。因为 C 语言可读性强，方便移植，同时效率高，作为嵌入式开发的首选。

编译

对于单片机来说，理解不了 C 语言，单片机的世界里只有 0 和 1 两种状态，所以我们的程序需要经过编译器先翻译成汇编语言，然后再生成单片机的机器语言。

```
;-----  
; Home  
;-----  
    .area HOME    (CODE)  
    .area HOME    (CODE)  
__sdcc_program_startup:  
    ljmp  _main  
; return from main will return to caller  
;-----  
; code  
;-----  
    .area CSEG    (CODE)  
;-----  
;Allocation info for local variables in function 'setup'  
;-----  
; main.c:3: void setup()  
; -----  
; function setup  
; -----  
_setup:  
    ar7 = 0x07  
    ar6 = 0x06  
    ar5 = 0x05  
    ar4 = 0x04  
    ar3 = 0x03  
    ar2 = 0x02  
    ar1 = 0x01  
    ar0 = 0x00  
; main.c:5: P4M1&=-0x02;P4M0|=0x02;//推挽输出  
    mov  r7,_P4M1  
    mov  a,#0xFD  
    anl  a,r7  
    mov  _P4M1,a  
    orl  _P4M0,#0x02  
; main.c:6: P4_1 = 1;//设置 P41 引脚高电平  
    setb _P4_1  
    ret  
;-----  
;Allocation info for local variables in function 'loop'  
;-----
```

```
; main.c:9: void loop()
; -----
;   function loop
; -----
_loop:
; main.c:12: }
ret
;-----
;Allocation info for local variables in function 'main'
;-----
; main.c:14: void main(void)
; -----
;   function main
; -----
_main:
; main.c:16: setup();
lcall _setup
; main.c:17: while(1){
00102$:
; main.c:18: loop();
lcall _loop
sjmp 00102$
.area CSEG (CODE)
.area CONST (CODE)
.area XINIT (CODE)
.area CABS (ABS, CODE)
```

上图是 C 语言生成的对应汇编语言的主要部分的截取。

main.hex																	
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00000000	3A	30	33	30	30	30	30	30	30	30	32	30	30	30	36	46	:03000000020006F
00000010	35	0D	0A	3A	30	33	30	30	35	46	30	30	30	32	30	30	5 :03005F000200
00000020	30	33	39	39	0D	0A	3A	30	33	30	30	30	33	30	30	30	0399 :030003000
00000030	32	30	30	37	30	38	38	0D	0A	3A	31	30	30	30	36	32	2007088 :100062
00000040	30	30	41	46	42	33	37	34	46	44	35	46	46	35	42	33	00AFB374FD5FF5B3
00000050	34	33	42	34	30	32	44	32	43	31	32	32	32	32	31	32	43B402D2C1222212
00000060	30	30	44	32	0D	0A	3A	30	36	30	30	37	32	30	30	36	00D2 :060072006
00000070	32	31	32	30	30	36	46	38	30	46	42	32	41	0D	0A	3A	212006F80FB2A :
00000080	30	36	30	30	33	35	30	30	45	34	37	38	46	46	46	36	06003500E478FFF6
00000090	44	38	46	44	39	46	0D	0A	3A	31	30	30	30	31	33	30	D8FD9F :1000130
000000A0	30	37	39	30	30	45	39	34	34	30	30	36	30	31	42	37	07900E94400601B7
000000B0	41	30	30	39	30	30	30	37	43	37	38	30	31	37	35	41	A0090007C780175A
000000C0	30	41	38	0D	0A	3A	31	30	30	30	32	33	30	30	30	30	0A8 :1000230000
000000D0	45	34	39	33	46	32	41	33	30	38	42	38	30	30	30	32	E493F2A308B80002
000000E0	30	35	41	30	44	39	46	34	44	41	46	32	37	35	34	43	05A0D9F4DAF2754C
000000F0	0D	0A	3A	30	32	30	30	33	33	30	30	41	30	46	46	32	:02003300A0FF2
00000100	43	0D	0A	3A	31	30	30	30	33	42	30	30	37	38	30	30	C :10003B007800
00000110	45	38	34	34	30	30	36	30	30	41	37	39	30	31	37	35	E84400600A790175
00000120	41	30	30	30	45	34	46	33	30	39	44	38	36	30	0D	0A	A000E4F309D860
00000130	3A	31	30	30	30	34	42	30	30	46	43	37	38	30	30	45	:10004B00FC7800E
00000140	38	34	34	30	30	36	30	30	43	37	39	30	30	39	30	30	84400600C7900900
00000150	30	30	31	45	34	46	30	41	33	31	38	0D	0A	3A	30	34	001E4F0A318 :04
00000160	30	30	35	42	30	30	44	38	46	43	44	39	46	41	46	41	005B00D8FCD9FAFA
00000170	0D	0A	3A	30	44	30	30	30	36	30	30	37	35	38	31	30	:0D00060075810
00000180	37	31	32	30	30	37	38	45	35	38	32	36	30	30	33	30	7120078E58260030
00000190	32	30	30	30	33	39	37	0D	0A	3A	30	34	30	30	37	38	2000397 :040078
000001A0	30	30	37	35	38	32	30	30	32	32	36	42	0D	0A	3A	30	00758200226B :0
000001B0	30	30	30	30	30	30	31	46	46	0D	0A						0000001FF

上图是生成的 main.hex 二进制文件，我们通过烧写工具烧写这个文件到单片机里，单片机就能执行我们写的程序了。具体单片机内部怎么根据这个文件去执行，不再展开讲了，可以查看微机原理相关书籍。

进制

我们日常使用十进制，有 0 到 9 数字组成，逢十进一。但是单片机内部都是各种数字电路组成，只有开和关两种状态，我们对应的用 0 和 1 来表示，数值逢二进一，书写时数字后面跟大写 B 结束来表示。

十进制	二进制	十进制	二进制
0	0000B	5	0101B
1	0001B	6	0110B
2	0010B	7	0111B
3	0011B	8	1000B
4	0100B	9	1001B

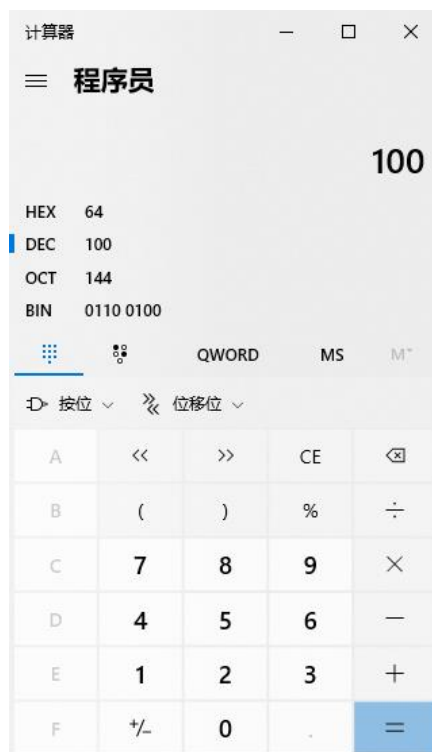
上表是十进制和二进制的对照表，根据这个规则，十进制的 100，用二进制表示为 110 0100B，我们会发现二进制虽然方便机器处理，但是不方便我们书写，太长了。于是我们又引进了十六进制。

十六进制是二进制的简短表示形式。十进制中的 0-15 分别表示为十六进制的 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F，书写时数字后面跟大写 H 结束来表示。

我们在程序里写的时候，加 “0x” 前缀，比如 0x03、0x1f、0x1F，不区分大小写。

十进制	十六进制	二进制	十进制	十六进制	二进制
0	00H	0000B	9	09H	1001B
1	01H	0001B	10	0AH	1010B
2	02H	0010B	11	0BH	1011B
3	03H	0011B	12	0CH	1100B
4	04H	0100B	13	0DH	1101B
5	05H	0101B	14	0EH	1110B
6	06H	0110B	15	0FH	1111B
7	07H	0111B	16	10H	10000B
8	08H	1000B	17	11H	10001B

我们需要熟练掌握各种进制之间的转化，初学者可以借助电脑的计算器程序，把他切换到程序员模式。



8421 码

我们把二进制以 4 位为最小单位，0001B 对应 $2^0=1$ 、0010B 对应 $2^1=2$ 、0100B 对应 $2^2=4$ 、1000B 对应 $2^3=8$ 。

$$1111B=8+4+2+1=15=0FH;$$

$$10001111B=8FH=8*16+15=143。$$

十六进制转二进制，就是上面的反向操作，把十六进制的每一位，拆分为 4 位二进制。

$$05H=0+4+0+1=0101B$$

$$09H=8+0+0+1=1001B;$$

$$10H=8+0+2+0=1010B;$$

$$15H=00010101B;$$

单位

在程序中我们做如下定义：

字节 (Byte) 是二进制数据的单位。一个字节通常 8 位长。

字节(Byte)=8 位(bit)

1KB(Kilobyte, 千字节)=1024B

1MB(Megabyte, 兆字节)=1024KB

1GB(Gigabyte, 吉字节, 千兆)=1024MB

寄存器

前面我们提到过寄存器类似单片机内部的一个个小开关，那具体有哪些寄存器，又怎么去找到他们。我们通过芯片手册可以找到寄存器表，表里定义了芯片所有寄存器的名称和地址。

8.6 特殊功能寄存器列表

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 端口	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111.1111
SP	堆栈指针	81H									0000.0111
DPL	数据指针 (低字节)	82H									0000.0000
DPH	数据指针 (高字节)	83H									0000.0000
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000.0000
S4BUF	串口 4 数据寄存器	85H									0000.0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011.0000
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000.0000
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000.0000
TL0	定时器 0 低 8 为寄存器	8AH									0000.0000
TL1	定时器 1 低 8 为寄存器	8BH									0000.0000
TH0	定时器 0 高 8 为寄存器	8CH									0000.0000
TH1	定时器 1 高 8 为寄存器	8DH									0000.0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000.0001
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000.x000
P1	P1 端口	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111.1111
P1M1	P1 口配置寄存器 1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111.1111
P1M0	P1 口配置寄存器 0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	0000.0000
P0M1	P0 口配置寄存器 1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111.1111
P0M0	P0 口配置寄存器 0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000.0000

(STC8H 寄存器表部分截图)

我们仔细观察上表，发现地址用十六进制表示，从 80H 开始，相邻地址间隔 8 位，这是因为我们的单片机是 8 位单片机，处理的数据总线宽度为 8 位。如果是 32 位单片机，则对应的寄存器间隔为 32 位。

位寻址

我们通过对寄存器的读写数据，就可以控制单片机。我们以地址 80H 的 P0 端口寄存器为例，当我们设置 P0=FFH 时，等于设置了从 P00 到 P07 8 个引脚的控制位为 1，也可以单独设置 P00=1。但是不是所有的寄存器都可以单独设置某一个位的。

80H–FFh 的位寻址区域则是所有的特殊功能寄存器中地址能被 8 整除的 16 个特殊功能寄存器（包括 80H、88H、90H、98H、A0H、A8H、B0H、B8H、C0H、C8H、D0H、D8H、E0H、E8H、F0H、F8H）的映射。

程序框架

#include <STC8HX.h>引入 STC8H 头文件，“include”为 C 语言的关键字，表示引入另外一个文件，C 语言规定了 include 前面加一个“#”表示预处理，这样告诉编译器，先把另外一个文件先复制到这个程序里，再执行编译。这里我们引入了 STC8HX.h 这个头文件，这个文件里定义了 STC8H 相关的寄存器。

```
C STC8HX.h ×
SDCC > include > mcs51 > C STC8HX.h > ...
130  __sbit __at(0xB7) P3_7;
131
132  __sfr __at(0xB1) P3M1;
133  __sfr __at(0xB2) P3M0;
134  __sfr __at(0xB3) P4M1;
135  __sfr __at(0xB4) P4M0;
136  __sfr __at(0xB5) IP2;
137  __sfr __at(0xB6) IP2H;
138  __sfr __at(0xB7) IPH;
139  __sfr __at(0xB8) IP;|
140  /* IP */
141  __sbit __at(0xB8) PX0;
142  __sbit __at(0xB9) PT0;
143  __sbit __at(0xBA) PX1;
144  __sbit __at(0xBB) PT1;
145  __sbit __at(0xBC) PS;
146  __sbit __at(0xBD) PADC;
147  __sbit __at(0xBE) PLVD;
148  __sbit __at(0xBF) PPCA;
149
150  __sfr __at(0xB9) SADEN;
151  __sfr __at(0xBA) P_SW2;
152  __sfr __at(0xBC) ADC_CONTR;
153  __sfr __at(0xBD) ADC_RES;
154  __sfr __at(0xBE) ADC_RES1;
155
156  __sfr __at(0xC0) P4;
157  /* P3 */
158  __sbit __at(0xC0) P4_0;
159  __sbit __at(0xC1) P4_1;
160  __sbit __at(0xC2) P4_2;
161  __sbit __at(0xC3) P4_3;
162  __sbit __at(0xC4) P4_4;
163  __sbit __at(0xC5) P4_5;
164  __sbit __at(0xC6) P4_6;
165  __sbit __at(0xC7) P4_7;
166
```

__sfr 是编译器定义的关键字表示特殊功能寄存器，__at(0xC1)定义寄存器的地址。__sbit 定义位。

函数

函数是一组一起执行一个任务的语句，由函数名称、输入参数、返回参数、主体组成。每个 C 程序都至少有一个函数，即主函数 `main()`，单片机工作时都是从 `main()` 函数开始运行。

```
void main()
{
    代码块
}
```

`void` 表示无返回值，输入参数 `()` 表示无输入参数，函数名可以有英文、数字、下划线组成，区分大小写，但是数字不能作为函数名的第一个字符出现，同时不能和编译器定义的关键字一样。一个 C 文件内不能有同名。

函数主体用一对大括号表示，大括号里可以放入其它函数或者代码。

我们的 `main.c` 程序里有三个函数，分别为 `main()`、`setup()`、`loop()`。

循环

```
while(条件)
{
    代码块
}
```

`while` 中文字面意思是“当”，当条件为真 (`true`) 时，执行大括号内部的代码；当条件为假 (`false`) 时，不执行内部代码块。我们一般用 `0` 表示 `false`，非 `0` 表示 `true`。

运行顺序

我们再回过头来看下我们的 `main.c` 程序。

从 `main` 函数开始运行，先运行 `setup()`，运行完后进入 `while(1)` 循环，因为条件一直为 `1` (非 `0`)，所以一直运行 `loop()` 函数。

`setup()` 函数里的程序只在单片机上电后运行一次，我们一般把一些初始化工作的相关代码都放在这里。

`loop()` 函数里程序一直循环往复运行，我们一般把需要不断运行的程序放在这里。

数据类型

C 语言中的数据类型分常量和变量两种。

常量指 `123`、`23.5` 等数字常量，“`Hello World`”、`'A'` 等字符常量和 `#define MAX_LED 8` 等用“`#define`”预定义的符号常量。`#define` 的作用类似取别名，方便程序员理解和增加程序可读性。

变量其实只不过是程序可操作的存储区的名称。C 中每个变量都有特定的类型，类型决定了变量存储的大小和布局，该范围内的值都可以存储在内存中，运算符可应用于变量上。

命名规范和前面说的函数名的命名规范一致。变量内部可以存放数据，数据可以通过程序来改变。

变量类型

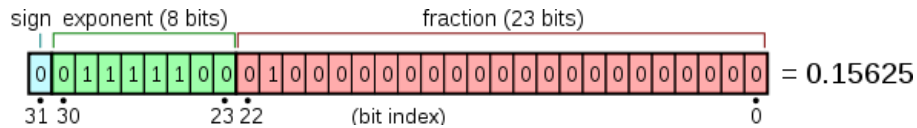
C 语言中的变量类型有如下几种：

类型 描述

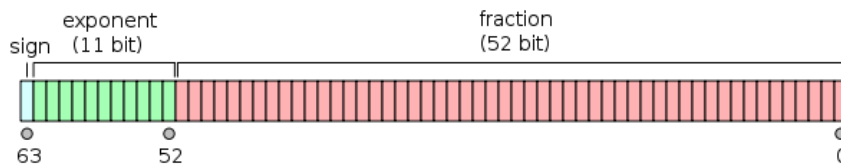
char 通常是一个字节（八位）。这是一个整数类型。

int 对机器而言，整数的最自然的大小。

float 单精度浮点值。单精度是这样的格式，1 位符号，8 位指数，23 位小数。



double 双精度浮点值。双精度是 1 位符号，11 位指数，52 位小数。



void 表示类型的缺失。

当我们需要表示负数的时候，把最高位作为符号标志，当最高位为 1 时表示负数，0 表示正数，这样就有了有符号和无符号两种分类。

类型	符号	关键字	所占位数	数据范围
整型	有	(signed) int	16	-32768-32767
	有	(signed) short	16	-32768-32767
	有	(signed) long	32	-2147483648-2147483647
	无	unsigned int	16	0-65535
	无	unsigned short int	16	0-65535
	无	unsigned long int	32	0-4294967295
实型	有	float	32	3.4e-38-3.4e38
	有	double	64	1.7e-308-1.7e308
字符型	有	char	8	128-127
	无	unsigned char	8	0-255

在写程序时要注意变量的范围，如果超出变量范围，会造成数据的溢出，导致程序不正常。

C 语言的二进制位操作运算

& (按位与)、| (按位或)、^ (按位异或)、~ (按位取反)、<< (左移) 和 >> (右移)。

(1) 按位与运算符(&)

按位与运算将两个运算分量的对应位按位遵照以下规则进行计算：

0 & 0 = 0, 0 & 1 = 0, 1 & 0 = 0, 1 & 1 = 1。

即同为 1 的位，结果为 1，否则结果为 0。

例如，0x23 & 0xFD = 0x21

00100011

11111101

00100001

嵌入式里常用用法，让某个寄存器的指定位置 0，其余位置保持不变。

(2) 按位或运算符(|)

按位或运算将两个运算分量的对应位按位遵照以下规则进行计算：

$0 | 0 = 0, 0 | 1 = 1, 1 | 0 = 1, 1 | 1 = 1。$

即只要有 1 个是 1 的位，结果为 1，否则为 0。

例如， $0x17 | 0x20 = 0x37$

00010111

00100000

00110111

嵌入式里常用用法，让某个寄存器的指定位置 1，其余位置保持不变。

(3) 按位异或运算符(^)

按位异或运算将两个运算分量的对应位按位遵照以下规则进行计算：

$0 ^ 0 = 0, 0 ^ 1 = 1, 1 ^ 0 = 1, 1 ^ 1 = 0。$

即相应位的值相同的，结果为 0，不相同的结果为 1。

嵌入式里常用用法，让某个位的状态在 0 和 1 之间来回切换。

(4) 按位取反运算符(-)

按位取反运算是单目运算，用来求一个位串信息按位的反，即哪些为 0 的位，结果是 1，而哪些为 1 的位，结果是 0。

例如， $-0x01=0xFE$

嵌入式里常用用法，配合&一起使用。

例如， $0x23 \& 0xFD = 0x21$ $-0x02 = 0xFD$ $0x23 \& (-0x02) = 0x21$

$0x02=00000010$ ，取反后为 111111101。

00100011

11111101

00100001

我们看到虽然两种运算结果都一样，但是我们一般都用 $0x23 \& (-0x02) = 0x21$ 这种模式，因为方便理解，我们一看就知道是要给寄存器的第 1 位置 0。

位运算符的优先级从高到低，依次为-、&、^、|，

其中-的结合方向自右至左，且优先级高于算术运算符，其余运算符的结合方向都是自左至右，且优先级低于关系运算符。

(5) 移位运算

移位运算用来将整型或字符型数据作为二进制信息串作整体移动。

有两个运算符：<< (左移) 和 >> (右移)

移位运算是双目运算，有两个运算分量，左分量为移位数据对象，右分量的值为移位位数。移位运算将左运算分量视作由二进制组成的位串信息，对其作向左或向右移位，得到新的位串信息。

移位运算符的优先级低于算术运算符，高于关系运算符，它们的结合方向是自左至右。

左移运算符(<<)

左移运算将一个位串信息向左移指定的位，右端空出的位用 0 补充。

例如 $0x81 \ll 2 = 4$,

10000001

00000100

左移时，空出的右端用 0 补充，左端移出的位的信息就被丢弃。在二进制数运算中，在信息没有因移动而丢失的情况下，每左移 1 位相当于乘 2。如 $4 \ll 2$ ，结果为 16。

右移运算符(>>)

为左移的反向操作。

嵌入式里常用用法，配合 (&)、(-) 一起使用。

例如，

$0x23 \& 0xFD = 0x21$

$0x23 \& (-0x02) = 0x21$ ($-0x02 = 0xFD$)

$0x23 \& (-0x01 \ll 1) = 0x21$

我们看到同样给寄存器的第 1 位置 0，通过这样的表示方式，可读性更强，也方便后面做一些自动化运算，比如想要设置第 3 位，那就是左移 3，其它部分不需要变动。

接下来需要配置 P41M1 为 0，P41M0 为 1，设置为推挽输出。

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1
P0M1	93H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0
P1M0	92H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1
P1M1	91H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0
P2M0	96H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1
P2M1	95H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0
P3M0	B2H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1
P3M1	B1H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0
P4M0	B4H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1
P4M1	B3H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0
P5M0	CAH	-	-	-	P54M1	P53M1	P52M1	P51M1	P50M1
P5M1	C9H	-	-	-	P54M0	P53M0	P52M0	P51M0	P50M0
P6M0	CCH	P67M0	P66M0	P65M0	P64M0	P63M0	P62M0	P61M0	P60M0
P6M1	CBH	P67M1	P66M1	P65M1	P64M1	P63M1	P62M1	P61M1	P60M1
P7M0	E2H	P77M0	P76M0	P75M0	P74M0	P73M0	P72M0	P71M0	P70M0
P7M1	E1H	P77M1	P76M1	P75M1	P74M1	P73M1	P72M1	P71M1	P70M1

对应代码如下

```
P4M1&=-0x02;
P4M0|=0x02;//推挽输出
```

也可以写成

```
P4M1&=-0x01<<1;
P4M0|=0x01<<1;
```

C 语言的一条语句以 (;) 分好结尾表示结束, (//) 双斜杠表示代码注释, 后面跟这条代码的说明文字。

C 语言中 (=) 不是“等于”而是“赋值”, 程序会把等式右边的数值或者变量的值赋值给等式左边的变量。

其中 (&=) (|=) 是一种简化的赋值运算,

运算符	描述	实例
=	简单的赋值运算符, 把右边操作数的值赋给左边操作数	C = A + B 将把 A + B 的值赋给 C
&=	按位与且赋值运算符	C &= 2 等同于 C = C & 2
=	按位或且赋值运算符	C = 2 等同于 C = C 2

同理, 我们还有额外的 (+=)、(-=)、(*=)、(/=)、(<<=)、(>>=)、(^=)、(%=)。

其中 (%) 在 C 语言中表示的是取余数。

5. 运行效果

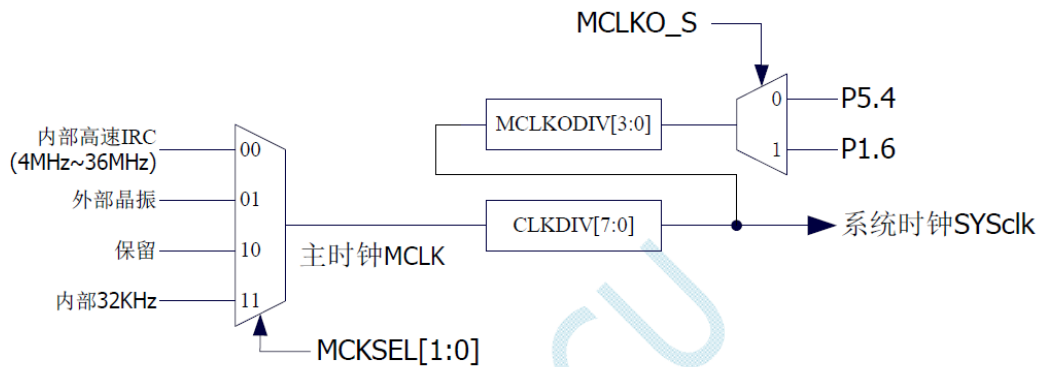
天问 51 开发板上的 CMP 标识旁边的 LED 灯常亮。

第三节 让 LED 闪烁

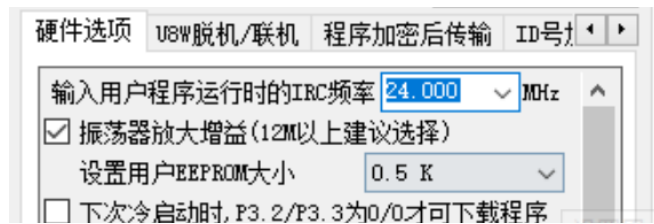
1. STC8H 系统时钟

单片机内部由晶振的震荡产生脉冲, 时钟脉冲是 CPU 的基本工作脉冲, 它控制着 CPU 的工作节奏。类似心跳, 如果震荡不起来, 单片机就没法工作。对于同一种单片机, 时钟频率越高, 单片机的工作速度就越快。

STC8H 有 3 个时钟源可供选择, 分别是内部高速 IRC、外部晶振、内部 32KHz, 可以通过寄存器 MCKSEL 设置。一般内部的 RC 时钟精度比外部的晶振低, 所以需要高精度的场合可以把时钟源切换到外部晶振, 天问 51 的外部晶振为 24M, 频率越高, 功耗也高, 当需要低功耗的需要把频率降低, 如果时钟源已经固定, 比如天问的外部晶振为 24M, 我们不方便去焊另外一个频率的晶振, 我们还可以通过内部的分频寄存器 CLKDIV 和 MCLKDIV 去降低频率。



在图形模块的系统配置里有对应的系统时钟设置。



其中内部 IRC 频率是通过 ISP 下载软件设置，如上图所示，所以在下载程序的时候一定要注意这里的设置。天问 51 的图形化平台默认都以 24M 为准。

2. 硬件设计

我们还是用的 P41 这个引脚，硬件和上一节一样。

3. 图形化编程



上面程序下载到开发板后能看到 LED 灯 1 秒亮，一秒灭的闪烁。如果我们在初始里设置了系统时钟的分频系数比如 2，则频率=24M/2=12M。我们重新下载程序后能观察到闪烁的频率变慢了。

4. 字符式编程

通过上面的图形化程序，我们接触到了延时函数，接下来我们来看看延时函数的内部。

首先我们先要引入延时函数相关的头文件

```
#include <STC8HX.h>
//系统时钟确认
#include "lib/delay.h"

void setup()
{
    P4M1&=-0x02;P4M0|=0x02;//推挽输出
}

void loop()
{
    P4_1 = 1;
    delay(1000);
    P4_1 = 0;
    delay(1000);
}

void main(void)
```

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

delay() 函数内部调用了一个延时 1ms 的函数，通过 while 循环，我们就能自由的设置需要延时多少毫秒，就能延时多少毫秒。

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

函数内部定义了一个名字为 time 的变量，他的类型为 uint16。我们前面提到过变量类型有 char、int、float 等等，但是肯定没有 uint16。其实这里的 uint16 是我们自己重定义的变量类型，这个重定义的代码在 STC8HX.h 头文件里有如下这样一段代码。

```
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
```

typedef 是 C 语言里的关键字，意思就是重定义，所以 uint16=unsigned int，这样重定义的好处有两点：1. 方便输入，打的字比较少。2. 很容易知道变量是多少位的。

delay(1000) 的函数，就是把函数内部的 time 变量赋初始值为 1000。

另外这里还用到了一个 C 语言循环

```
do  
{  
  //代码块  
}while (条件)  
语句。
```

功能就是先执行 do 大括号里的代码块，再判断条件。

和 while 的区别是，while 先判断后执行，do while 先执行后判断

当不满足循环条件时，while 循环一次都不会执行，do while 循环至少执行一次。

所以上述程序，就是当 time 的值大于 0 的时候，会执行一次内部的 delay1ms() 函数，然后再去执行 while 里的 --time 运算。-- 为自减运算符，可以放在变量前面 --time，也可以放在变量后面 time--，把变量的值减 1，但是两种方式执行的顺序不一样，区别如下：

--time 为先执行自减 1000-1=999，然后判断 999 是否为 0。判断过程为 999.....3、2、1 999 次，加上 do 最先执行的一次，总共 1000 次。

time--为先判断 1000 是否为 0，然后再执行 1000-1=999。判断过程为 1000、999.....3、2、1 循环 1000 次。所以如下程序等效于上述程序

```
while(time--)\n{\n    delay1ms();\n}
```

有自减操作符，同样有自增操作符++，使用方式类似，不再详说。

那这个延时 1ms 的函数内部又是什么内，请看下图。

```
void delay1ms() //1 毫秒@24.000MHz\n{\n    uint8 i, j;\n    _nop_();\n    i = 32;\n    j = 40;\n    do { while (--j);} while (--i);\n}
```

我们看到有一个_nop_()函数，这个是空指令，一条空指令是内部最简单的一条指令，单片机的系统时钟频率确定后，那么执行一条指令的时间也就确定了。前面我们已经了解过时钟频率 f，对应的时钟周期就是 T=1/F。在嵌入式 51 里，我们还有机器周期、指令周期。

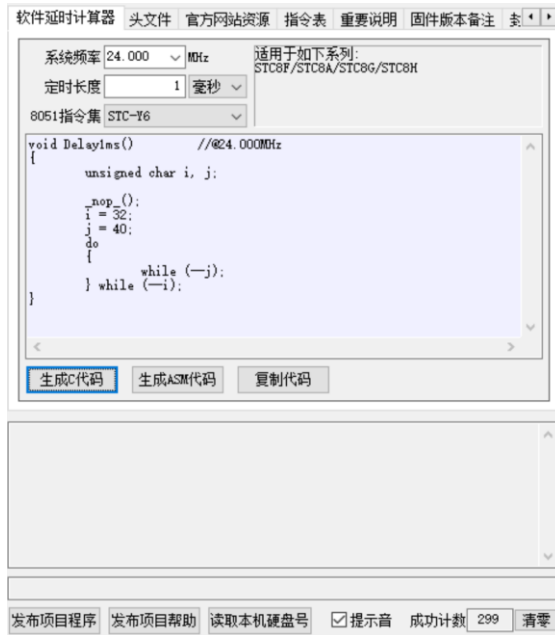
机器周期：单片机的基本操作周期，在一个操作周期内，单片机完成一项基本操作，如取指令、存储器读写等。它由 12 个时钟周期（6 个状态周期）组成。

指令周期：它是指 CPU 执行一条指令所需要的时间。一般一个指令周期含有 1——4 个机器周期。

标准 51 单片机是 12T 的，就是说 12 个时钟周期（晶振周期，例如 12M 的，周期是 1/12M，单位秒），机器做一个指令周期，刚好就是 1/12M*12=1us，常见指令例如_nop_就是一个周期，刚好 1us，其他的大多多于一个周期，乘除法更多。所以如果计算指令时间可以这样算。

而现在很多 51 核的单片机工艺质量上去后，频率大大提高，增强型 51 有 6T 的，如果接 12M 的话，一个 nop 就只需要 0.51uS，STC8H 单片机为 1T，那只需要 1/12us。

我们现在频率为 24M，则一个空指令执行时间为 1/24us。通过汇编语言，我们能准确的计算出延时 1ms 需要执行多少个 nop，但是 C 语言还需要翻译成汇编语言，所以我们没法准确的计算。这里我们用 STC 提供的官方工具里的软件延时计算器：



5. 运行效果

天问 51 开发板上的 CMP 标识旁边的 LED 灯 1 秒亮 1 秒灭。

第四节 跑马灯

1. STC8H GPIO 端口操作

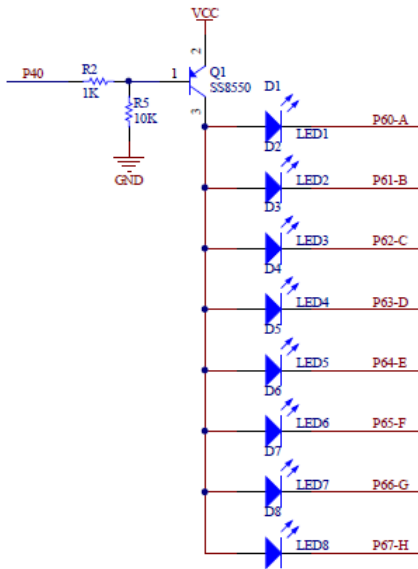
前面我们提到过，STC8H 把引脚分为 P0、P1、P2、P3、P4、P5、P6、P7 共 8 组，我们把这些叫作端口。我们除了可以单独设置引脚状态外，还可以一起设置同一组的引脚，即端口设置。

对应的图形块有以下三种：



2. 硬件设计

LED



这里 P40 控制一个 PNP 三极管，因为 R5 10K 电阻的原因，上电默认三极管导通，8 个 LED 为共阳接法，当 P6 口为低电平时，LED 灯导通发光。

因为这里的 P6 口是天问 51 开发板上最复杂的口，P6 口还同时接了数码管、点阵、彩屏、1602、12864 等等，同一时间只能使用一个，具体详见[天问 51 原理图](#)的 P6 端口详解部分。

3. 图形化编程

设置 8 个 LED 灯全亮



这里我们看到多了一个天问 51 初始化的图形块，我们可以试试不加初始化和加上初始化有什么区别。

通过观察，我们发现不加初始化时，开发板上的数码管和点阵模块，每次开机都会随机亮。加了初始化后，只有 8 个 LED 灯亮。

我们接下来看下对应的代码，里面到底初始化了什么。

4. 字符式编程

```
#include <STC8HX.h>
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.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
    P6M1=0x00;P6M0=0xff;//推挽输出
    P6 = 0;
}

void loop()
{
}

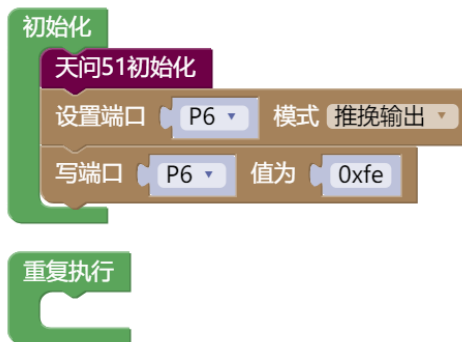
void main(void)
{
    setup();
    while(1){
        loop();
    }
}
```

```
}  
}
```

我们通过代码看到了初始化的函数里有一个 hc595 和一个 rgb。hc595 是一个移位寄存器芯片，他可以通过三根信号引脚控制多路输出，扩展了 IO 引脚。天问 51 里的 hc595 芯片控制着数码管和点阵模块，初始化函数默认把数码管和点阵关闭，具体 595 芯片怎么控制的我们后面再详细展开。另外一个 RGB 模块，也是默认先把他关闭，不然上电后会随机亮。

1.1 让 LED1 亮，其它灭。

LED1 为低电平，其它为高电平，P6=0xfe=11111110B



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

1.2 让 LED8 亮，其它灭。

P6=0x7f; 01111111B

1.3 让 LED1-LED4 亮，其它灭

P6=0xf0; 11110000B

1.4 让 LED 轮流点亮

要让 LED 轮流亮，我们可以类似上面三个案例一样计算出每种状况下的 P6 端口的值，如下表：

LED	P6(十六进制)	P6(二进制)
LED1	0xfe	11111110B
LED2	0xfd	11111101B
LED3	0xfb	11111011B
LED4	0xf7	11110111B
LED5	0xef	11101111B
LED6	0xdf	11011111B
LED7	0xbf	10111111B
LED8	0x7f	01111111B

于是要让 LED 轮流亮，只需要把 P6 口的值，按照表格的数值来依次设置就行。程序如下：



```
#include <STC8HX.h>
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.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  
  P6M1=0x00;P6M0=0xff;//推挽输出  
}
```

```
void loop()  
{  
  P6 = 0xfe;  
  delay(1000);  
  P6 = 0xfd;  
  delay(1000);  
  P6 = 0xfb;  
  delay(1000);  
  P6 = 0xf7;  
  delay(1000);  
  P6 = 0xef;  
  delay(1000);  
  P6 = 0xdf;  
  delay(1000);  
  P6 = 0xbf;  
  delay(1000);  
  P6 = 0x7f;  
  delay(1000);  
}
```

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

我们下载运行后能发现 LED 灯从第一个开始到第八个间隔 1 秒轮流点亮。延时 1 秒是为了方便观察，为了显示效果好，自己可以试试把时间改短。程序比较简单，就是有点长，假如我们要设置 100 个 LED 轮流点亮，按照上述写法，我们程序就太长了，那还有没有更好的程序，能实现相同的效果呢？

当然有，还不止一种，接下来我们来看看另外一种程序，如下：

初始化

code 无符号8位整数 led8 [8] 从 { 0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f } 创建数组

天问51初始化

设置端口 P6 模式 推挽输出

重复执行

使用 i 从范围 0 到 8 每隔 1

执行 写端口 P6 值为 led8 的第 i 项

延时 1000 毫秒

```
#include <STC8HX.h>
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"

uint8 i;
Code uint8 led8[8]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
```

```
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
    P6M1=0x00;P6M0=0xff;//推挽输出
}
```

```
void loop()
{
    for (i = 0; i < 8; i = i + 1) {
        P6 = led8[i];
        delay(1000);
    }
}
```



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

上面程序，我们下载运行后发现运行效果一样，而且程序简单了不少，但是程序里多了两个新的知识，分别是数组和 for 循环。

数组

C 语言中的数组是用来存储一个表数据，它往往被认为是一系列相同类型的变量。

led8	0	1	2	3	4	5	6	7
	0xfe	0xfd	0xfb	0xf7	0xef	0xdf	0xbf	0x7f

数组类似一张表格，如上，每一格里可以存放相同类型的数据，同时每个格子都有编号，我们可以通过编号来找到对应的格子，从而取出数据。

code 无符号8位整数 led8 [8] 从 { 0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f } 创建数组

```
code uint8 led8[8]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
```

我们可以通过上述方式，定义数组的长度和数组内部数据的初始值。

code 无符号8位整数 led8 [8]

```
code uint8 led8[8];
```

也可以通过上述方式，直接定义一个指定长度的的数组，具体数组里面的数据，我们可以后面通过程序再给他设置。

led8 的第 0 项

```
led8[0];
```

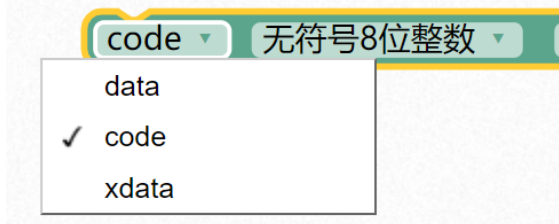
我们可以通过上述语句来获取对应格子里的数据。

led8 的第 0 项赋值为 0xfe

```
led8[0]=0xfe;
```

我们也可以通过上述语句对里面的数据更新。

另外我们注意到声明数组的时候，前面多了个前缀 code，



我们再回看下声明变量的时候，也有一个下拉菜单。



data、code、xdata 这些是什么意思呢？在我们单片机里有一块存储区域，它分为：

RAM：随机存取存储器（英语：Random Access Memory，缩写：RAM），也叫内存，是与 CPU 直接交换数据的内部存储器。

ROM：（只读内存(Read-Only Memory)简称）英文简称 ROM。ROM 所存数据，一般是装入整机前事先写好的，整机工作过程中只能读出，而不像随机存储器那样能快速、方便地加以改写。

STC8H 系列单片机内部集成了大容量的数据存储器。STC8H 系列单片机内部的数据存储器在物理和逻辑上都分为两个地址空间：内部 RAM(256 字节)和内部扩展 RAM。一般放在内部 RAM 里的数据和 CPU 交换数据比较快，但是容量比较小，外部 RAM 的读写速度比内部 RAM 稍慢。STC8H8K64U 系列单片内部集成了 64K 字节的 Flash 程序存储器（ROM）。

51 单片机的内存资源是非常可贵的，不像我们电脑，都是 4G、8G，只有 256 字节，程序里不注意的话，很容易把内存用完，这样会导致申请不到内存，内存溢出，导致程序异常。

为了合理分配内存，我们把数据量小，同时需要改变的数据放在内部 RAM 里，用 data 关键词来指定，默认 data 可以省略；把数据量大，同时需要改变的数据放在外部 RAM 里，用 xdata 关键词来指定；把不需要改变的数据放到 ROM 里，用 code 关键词来指定。

外部扩展 RAM，不同芯片，大小不一样，具体如下表所示。

STC8H 系列单片机内部集成的 RAM 可用于存放程序执行的中间结果和过程数据。

单片机系列	内部直接访问 RAM (DATA)	内部直接访问 RAM (IDATA)	内部扩展 RAM (XDATA)
STC8H1K08 系列	128 字节	128 字节	1024 字节
STC8H1K28 系列	128 字节	128 字节	1024 字节
STC8H3K64S4 系列	128 字节	128 字节	3072 字节
STC8H3K64S2 系列	128 字节	128 字节	3072 字节
STC8H8K64U 系列	128 字节	128 字节	8192 字节

for 循环

for (表达式 1; 表达式 2; 表达式 3)

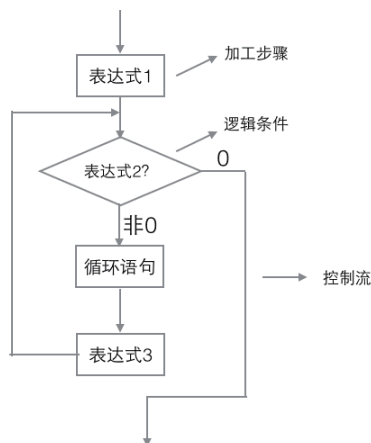
```
{
    循环语句
}
```

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：



```
for (i = 0; i < 8; i = i + 1) {
    P6 = led8[i];
    delay(1000);
}
```

我们来分析上面这部分代码，for 语句第一步，把 i 变量赋初值为 0，当 i 小于 8 的时候，执行 for 语句里的代码块，执行完后，把 i 这个变量加 1，然后再判断是否小于 8，直到 i=7，把 i 加 1 变为 8 以后，8 不小于 8，条件不成立，跳出 for 循环语句，执行后面的代码。其中 P6 = led8[i] 为从数组 led8 中取出第 i 个数据。

我们还可以用前面学过的移位和取反操作来实现流水灯，程序如下：



对应的代码如下

```
#include <STC8HX.h>
//系统时钟确认
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"

uint8 temp = 0;

uint8 i;

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
```

```

P6M1=0x00;P6M0=0xff;//推挽输出
}

void loop()
{
    temp = 0x01;
    for (i = 0; i < 8; i = i + (1)) {
        P6 = -temp;
        delay(50);
        temp = (temp<<1);
    }
    temp = 0x80;
    for (i = 0; i < 8; i = i + (1)) {
        P6 = -temp;
        delay(50);
        temp = (temp>>1);
    }
}

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

```

上述程序，第一个 for 语句执行过程，各个变量的值变化变化如下：

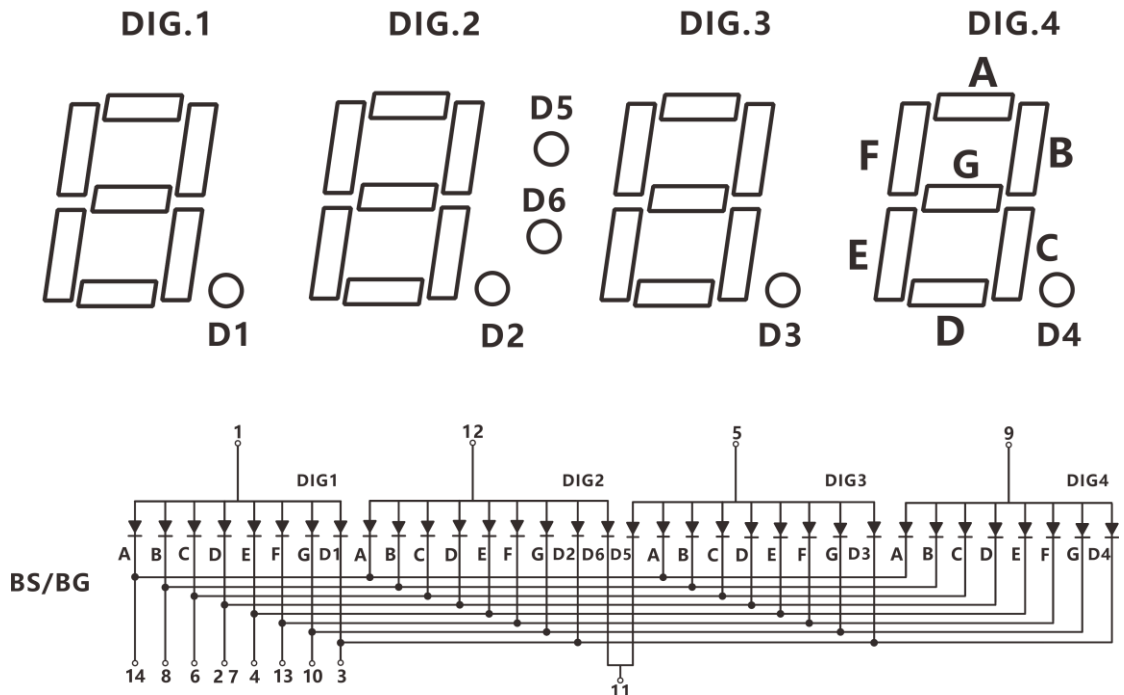
i	P6(十六进制)	P6 (二进制)	temp(十六进制)	temp (二进制)
0	0xfe	11111110B	0xfd	11111101B
1	0xfd	11111101B	0xfb	11111011B
2	0xfb	11111011B	0xf7	11110111B
3	0xf7	11110111B	0xef	11101111B
4	0xef	11101111B	0xdf	11011111B
5	0xdf	11011111B	0xbf	10111111B
6	0xbf	10111111B	0x7f	01111111B
7	0x7f	01111111B	0xfe	11111110B

通过观察二进制的 0 位置，可以看到 0 在一位一位左移，这样就可以观察到 LED 灯从左到右亮起，第二个 for 语句效果为从右到左的来回亮起，形似跑马，故名“跑马灯”又叫“流水灯”。

通过本节课程，我们发现要实现相同的效果，程序上可以有很多种，所以我们要慢慢学会举一反三，找到效率最高的程序写法，这也就是算法。

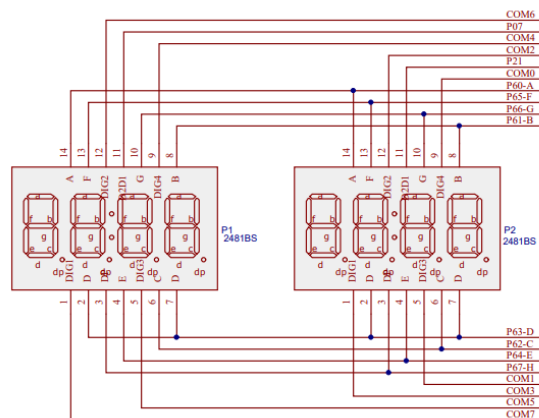
第五节 数码管显示原理

1. 硬件设计

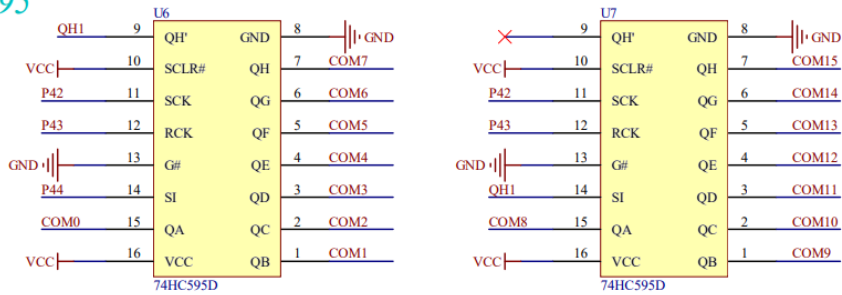


我们先看下数码管的手册，数码管内部其实也是一个个 LED 组成，其中一个“8”字需要 A、B、C、D、E、F、G 共 7 段 LED 组成，外加一个小数点 D1、D2、D3、D4。我们可以通过设置指定的段来显示不同数字或者字符。通过手册，我们还能看到中间的一个冒号 D5、D6 连接到 11 脚。其中每个“8.”的 LED 阳极并联，分别到 DIG1、DIG2、DIG3、DIG4 四个位选控制引脚。通过程序控制让 DIG 轮流高电平导通，就能让四个 8 字轮流显示，共用段选引脚。

天问 51 开发板上的对应原理图如下：



74HC595



数码管的段选都对应连接到了 P6 端口，同时位选连接到了 595 芯片的 COM0-COM7。

2. 段码显示

1.1 图形化编程



```
#include <STC8HX.h>
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#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
    led8_disable();//关闭 8 个 LED 流水灯电源
    hc595_enable_nix();
    P6M1=0x00;P6M0=0xff;//推挽输出
    P6 = 0;
}

void loop()
{

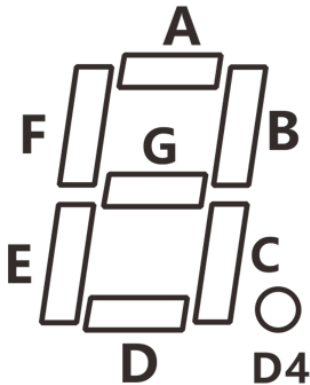
}

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

```

第一步关闭流水灯电源，第二步开启数码管，第三步设置 P6 端口为 0，我们可以看到数码管的 8 个“8”和小数点全部亮起。

接下来我们试下怎么让数码管显示 3，



我们对照上图，要显示 3，就是让 A、B、C、D、G 这 5 段亮，其它段灭。根据原理图，我们已经知道每一段和引脚的对应关系。

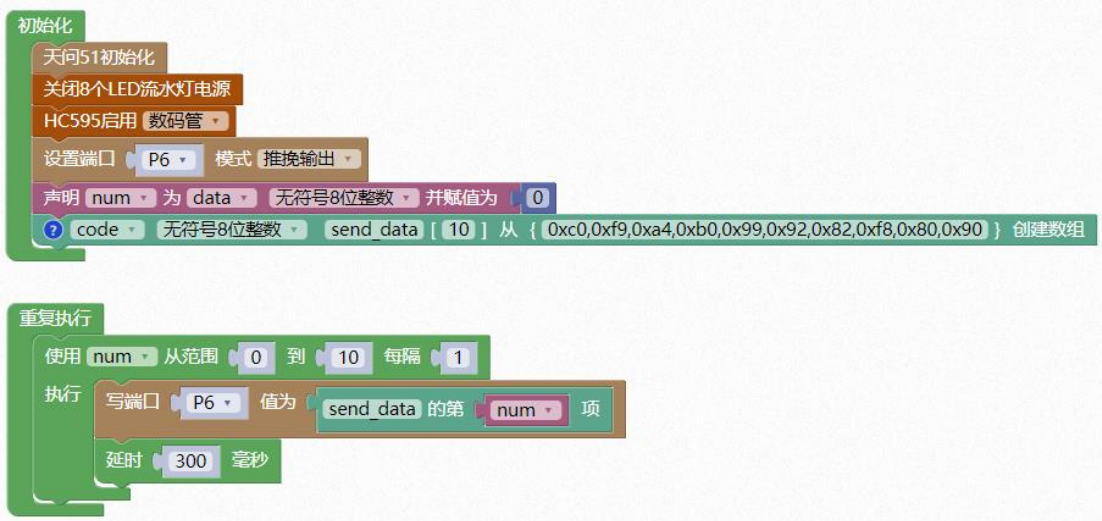
P67	P66	P65	P64	P63	P62	P61	P60
DP	G	F	E	D	C	B	A
1	0	1	1	0	0	0	0

根据上表我们知道，显示 3，应该给 P6 赋值 0xb0。

我们可以修改下上面程序，下载验证下。这样我们还可以计算出 0-9 对应的段码值，如下表：

0	1	2	3	4	5	6	7	8	9
0xc0	0xf9	0xa4	0xb0	0x99	0x92	0x82	0xf8	0x80	0x90

下面我们让数码管显示 0-9，程序如下：



```
#include <STC8HX.h>
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"

uint8 num = 0;
code uint8 send_data[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};

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
    hc595_enable_nix();
    P6M1=0x00;P6M0=0xff;//推挽输出
}
```

```
void loop()
{
  for (num = 0; num < 10; num = num + (1)) {
    P6 = send_data[(int)(num)];
    delay(300);
  }
}

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

3. 位码显示

上一节的程序效果，我们看到天问 51 开发板上的 8 个“8”同时在变化，那么怎么让指定某个“8”来显示呢？

我们还有控制 595 单独选通的函数，如下图所示

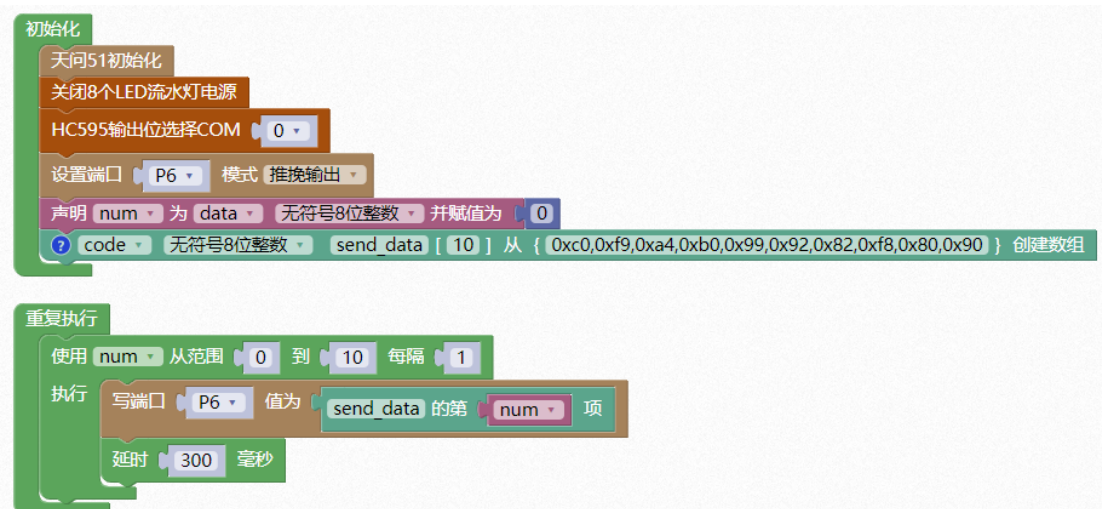


HC595输出位选择COM 0

```
hc595_bit_select(0);
```

函数参数范围为 0-15，其中 COM0-COM7 对应控制数码管的 DIG 位选控制引脚。595 的内部具体原理后面章节会深入讲，我们先学会使用。

1.1 编程



初始化

- 天问51初始化
- 关闭8个LED流水灯电源
- HC595输出位选择COM 0
- 设置端口 P6 模式 推挽输出
- 声明 num 为 data 无符号8位整数 并赋值为 0
- code 无符号8位整数 send_data [10] 从 { 0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90 } 创建数组

重复执行

- 使用 num 从范围 0 到 10 每隔 1
- 执行 写端口 P6 值为 send_data 的第 num 项
- 延时 300 毫秒

我们把上述代码下载到开发板能看到数码管的第 0 位，即最右边的“8”字显示，其它的不显示。

4. 扫描显示

上一节我们已经学会了数码管显示单独一个位置的方法，接下来我们要显示 123，应该怎么办呢？



595 同一时间我们只选通一个位，因为选通多个位没意义，显示的数据是重复的。那么要显示 123，我们就需要先设置 P6 口的段码为 1，同时选通第 2 位位码，设置段码为 2，选通第 1 位位码，设置段码为 3，选通第 0 位位码。这样只要速度够快，人的眼睛还没反应过来，就已经显示下一位了，这是利用了人眼的视觉暂留现象。那这个速度需要多快呢？我们通过下面的程序来自己试验下。

1.1 编程

```
#include <STC8HX.h>
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"

uint8 num = 0;
code uint8 send_data[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,
0x90};
```

```
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
    P6M1=0x00;P6M0=0xff;//推挽输出
}

void loop()
{
    for (num = 0; num < 3; num = num + (1)) {
        hc595_bit_select((2 - num));
        P6 = send_data[(int)((num + 1))];
        delay(1);
    }
}

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

我们通过改变上面的延迟时间，实验得到延迟时间在 2—8 毫秒之间显示相对可以，大于 8 毫秒后开始闪烁，小于 2 毫秒以后残影明显。

接下来我们显示 12345678，只需要对上面程序做稍微修改，如下

```

初始化
天问51初始化
关闭8个LED流水灯电源
设置端口 P6 模式 推挽输出
声明 num 为 data 无符号8位整数 并赋值为 0
code 无符号8位整数 send_data [ 10 ] 从 { 0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90 } 创建数组

重复执行
使用 num 从范围 0 到 8 每隔 1
执行
HC595输出位选择COM 7 num
写端口 P6 值为 send_data 的第 num + 1 项
延时 2 毫秒

```

再次测试延迟时间，发现 2 毫秒相对好一点，但是每个数字，仔细观察那些不亮的段，还是有隐隐约约的发光，这个就是扫描导致的残影，要消掉这个很简单，就是显示完毕以后，再把 P6 口设置高电平，让 LED 里的电流不再流动就行。

```

初始化
天问51初始化
关闭8个LED流水灯电源
设置端口 P6 模式 推挽输出
声明 num 为 data 无符号8位整数 并赋值为 0
code 无符号8位整数 send_data [ 10 ] 从 { 0xc0,0xf9,0xa4,0xb0,0x99,0x92,0xf8,0x80,0x90 } 创建数组

重复执行
使用 num 从范围 0 到 8 每隔 1
执行
HC595输出位选择COM 7 num
写端口 P6 值为 send_data 的第 num + 1 项
延时 2 毫秒
写端口 P6 值为 0xff

```

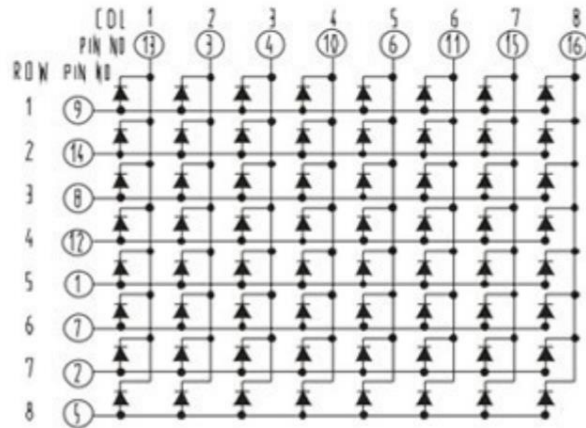
我们再下载程序观察一下，显示效果非常不错。

5. 数码管库函数

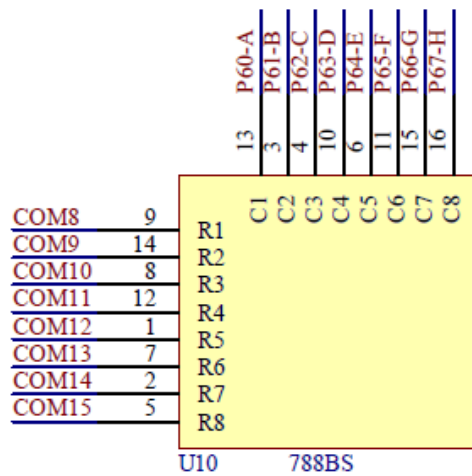
到这里我们已经掌握数码管的显示原理，针对不通的需求，天问 51 在线编程平台里已经提供好了多种数码管的驱动函数。具体使用方式可以查看模块 API 接口文档。想要学习模块内部实现，可以查看开源驱动库对应文件。

第六节 点阵显示原理

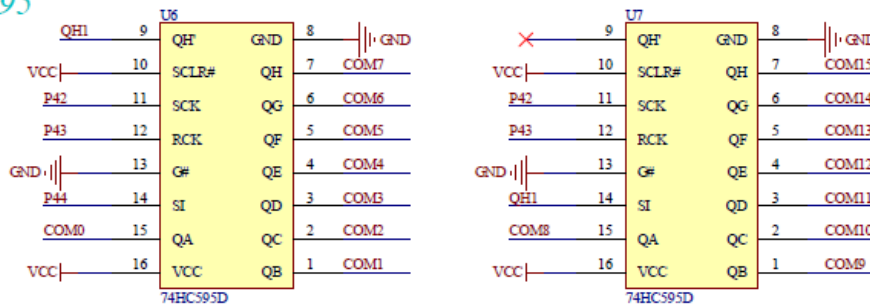
1. 硬件设计



我们通过查看点阵的手册看到上图所示，点阵内部和数码管类似，也是由一个个LED组成，共有 8*8 64 个LED。



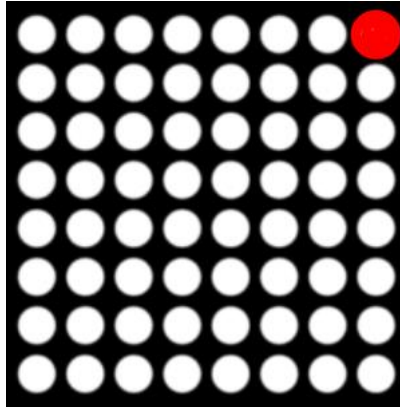
74HC595



天问 51 上的点阵和数码管一样，阳极都连接到了 595 的输出端。阴极连接到了 P6 口。

2. 行列显示

接下来，我们让点阵的第一行第8列的这个点点亮，我们对照原理图，就是要让 COM8 输出高电平，让 P60 输出低电平。



对应程序如下：



```
#include <STC8HX.h>
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#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
```

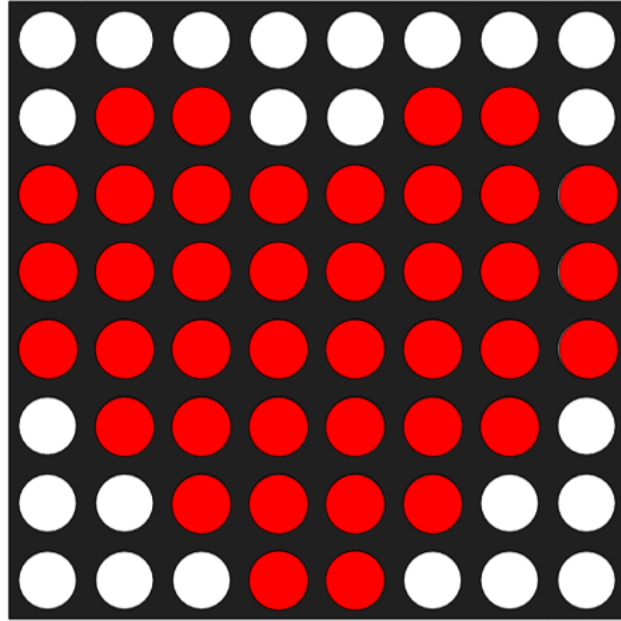
```
delay(10);
}
void setup()
{
    twen_board_init();//天问 51 初始化
    led8_disable();//关闭 8 个 LED 流水灯电源
    hc595_bit_select(8);
    P6M1=0x00;P6M0=0xff;//推挽输出
    P6 = 0xfe;
}
void loop()
{
}
void main(void)
{
    setup();
    while(1){
        loop();
    }
}
```

和数码管一样，因为 P6 口共用的问题，我们要先把 8 个 LED 的电源关闭，然后设置 595 的输出为 COM8，设置 P6 口的值为 0xfe。

同理要显示第 8 列上的其它点，只需要设置对应的值给 P6 口就行。要在其它行显示，设置 595 输出到对应的列上就行。

3. 扫描显示

上面的程序只能单独显示某一行，如果要显示如下这样一个爱心图案，那就需要扫描来实现，和数码管的扫描一样的原理，让每一列轮流显示。



对应的程序如下：

初始化

- 天问51初始化
- code 无符号8位整数 mylist [8] 从 { 0xe3,0xc1,0x81,0x03,0x03,0x81,0xc1,0xe3 } 创建数组
- 关闭8个LED流水灯电源
- 设置端口 P6 模式 推挽输出

重复执行

- 使用 i 从范围 8 到 16 每隔 1
- 执行 HC595输出位选择COM i
- 写端口 P6 值为 mylist 的第 i 项
- 延时 2 毫秒
- 写端口 P6 值为 0xff

```
#include <STC8HX.h>
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.h"
#include "lib/led8.h"
uint8 i;
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
```

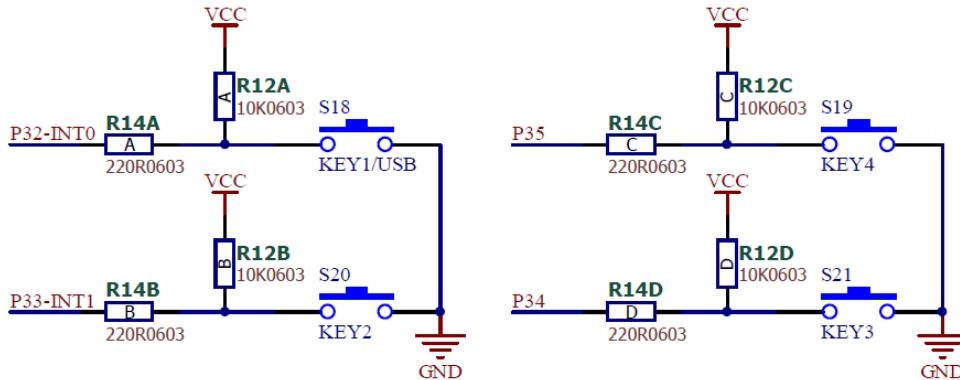
```
delay(10);  
rgb_show(0,0,0,0);//关闭 RGB  
delay(10);  
}  
Code uint8 mylist[8]={0xe3,0xc1,0x81,0x03,0x03,0x81,0xc1,0xe3};  
void setup()  
{  
    twen_board_init();//天问 51 初始化  
    led8_disable();//关闭 8 个 LED 流水灯电源  
    P6M1=0x00;P6M0=0xff;//推挽输出  
}  
void loop()  
{  
    for (i = 8; i < 16; i = i + (1)) {  
        hc595_bit_select(i);  
        P6 = mylist[(int)((i - 8))];  
        delay(2);  
        P6 = 0xff;  
    }  
}  
void main(void)  
{  
    setup();  
    while(1){  
        loop();  
    }  
}
```

我们用数组保存每列显示的数据，然后让 595 输出从 COM8 到 COM15 轮流输出，同样动态扫描都需要消影，显示完后延迟 2 毫秒后把 P6 口设置高电平。

第七节 独立按键控制 LED

1. 硬件设计

KEY



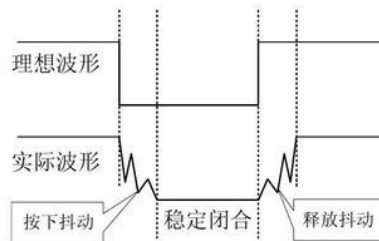
天问 51 开发板上有 4 个独立按键，分别连接到 P32、P33、P34、P35，其中 P32、P33 引脚还有外部中断功能，我们下一节会讲到。

通过观察上面的原理图，我们看到引脚默认为高电平，当按键按下时为低电平，我们就可以通过查询引脚的电平状态来知道按键是否按下。

但是因为机械按键在我们按的瞬间，会有一段不稳定的抖动。抖动时间的长短由按键的机械特性决定的，一般为 5ms 到 10ms。按键稳定闭合时间的长短则由操作人员的按键动作决定的，一般为零点几秒至数秒。按键抖动会引起按键被误读多次。为了确保 CPU 对按键的一次闭合仅作一次处理，必须进行消抖。

按键的机械抖动现象

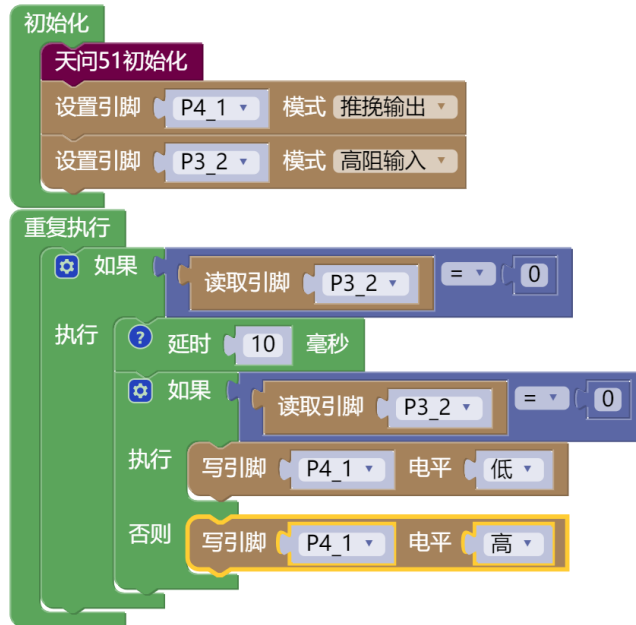
按键在闭合和断开时，触点会存在抖动现象：



按键消抖有两种方式，一种是硬件消抖，另一种是软件消抖。为了使电路更加简单，通常采用软件消抖。我们开发板也是采用软件消抖，一般来说一个简单的按键消抖就是先读取按键的状态，如果得到按键按下之后，延时 10ms，再次读取按键的状态，如果按键还是按下状态，那么说明按键已经按下。

下面我们通过按键 KEY1 控制 P41 的 LED 灯为例，来实际测试一下。

2. 图形化编程



3. 字符式编程

```
#include <STC8HX.h>
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.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
    P4M1&=-0x02;P4M0|=0x02;//推挽输出
    P3M1|=0x04;P3M0&=-0x04;//高阻输入
```

```
}  
  
void loop()  
{  
  if(P3_2 == 0){  
    // 按键消抖  
    delay(10);  
    if(P3_2 == 0){  
      P4_1 = 0;  
    }  
    else{  
      P4_1 = 1;  
    }  
  }  
}  
  
void main(void)  
{  
  setup();  
  while(1){  
    loop();  
  }  
}
```

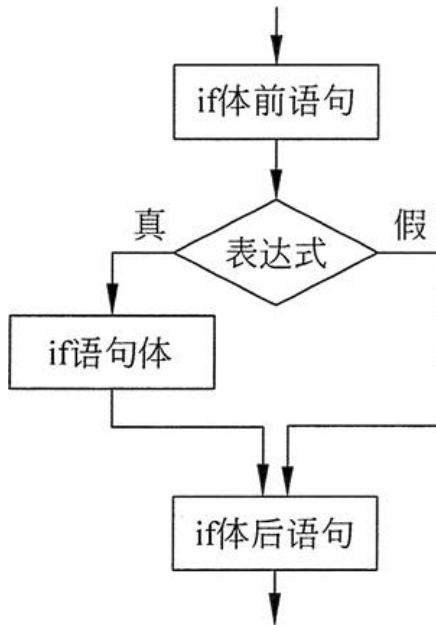
if else 语句

上述程序中，我们运用到了 C 语言中的条件判断语句，if 语句是一种分支结构，当条件满足时，有“执行该操作语句”和“跳过执行该操作语句”的两条分支。

语法如下：

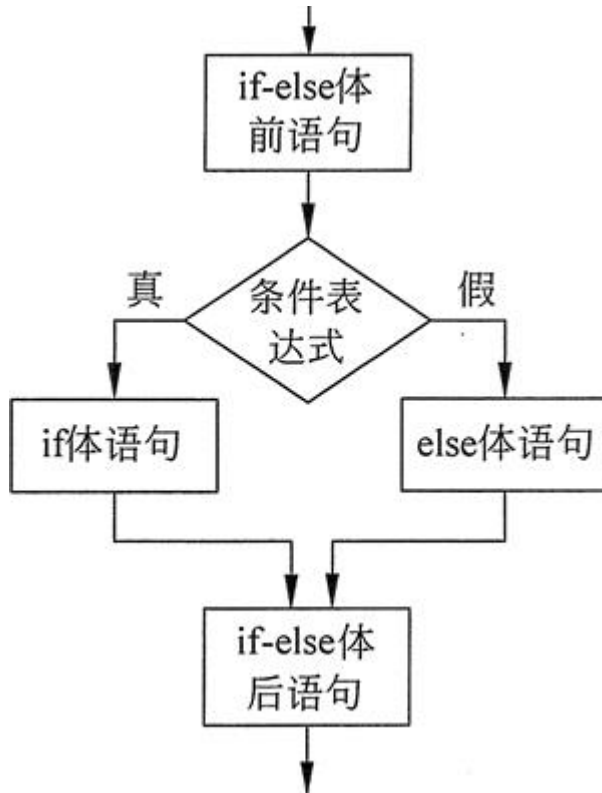
```
if (条件表达式)  
{  
    复合语句 A;  
}
```

if 语句的执行流程：首先判断关键词 if 后括号内条件表达式的值，如果该表达式的值为逻辑真（非 0），则执行 if 体，接着执行 if 体后的其他语句；否则，若该表达式的值为逻辑假（0），则不执行该 if 体，直接执行 if 体后的其他语句。



if-else 语句的执行流程：首先判断关键词 if 后括号内条件表达式的值，如果该表达式的值为逻辑真（非 0），则执行 if 体（语句 A），而不执行 else 体（语句 B），然后继续执行 if-else 之后的其他语句；否则，若该表达式的值为逻辑假（0），则不执行该 if 体（语句 A），而执行 else 体（语句 B），然后继续执行 if-else 之后的其他语句。if-else 语句的执行流程图如图 2 所示。

```
if(条件表达式)
{
    复合语句 A; //if 体
}
else
{
    复合语句 B; //else 体
}
```



由于表达式的值在逻辑上只有真和假，故 if 和 else 在执行流程上是互斥的，执行且只能执行两者中的一个。

if 语句嵌套

以下情况均属于 if 结构嵌套。

if 语句体中可以含有 if 语句或 if-else 语句。

if-else 语句中的 if 体或者 else 体中含有 if 语句或 if-else 语句。

注意：

1) 在嵌套结构中会有多个“if”与多个“else”关键词，每一个“else”都应对应的“if”相配对。原则：“else”与其前面最近的还未配对的“if”相配对。

2) 配对的 if-else 语句可以看成一条简单语句。

3) 一条 if 语句也可以看成一条简单语句。

级联 else-if 多分支语句

if (条件表达式 1)

 语句 1;

else if (条件表达式 2)、

 语句 2;

...

else if (条件表达式 n)

 语句 n;

else

语句 n+1;

该级联的 if-else-if 多分支结构的执行流程是：从前往后计算各个表达式的值，如果某个表达式的值为真，则执行对应的语句，并终止整个多分支结构的执行。如果上述所有表达式均不成立，即均为逻辑假时，则执行对应的 else 部分。else 部分可以省略，但一般情况下不省略。

4. 运行效果

上述程序为先初始化按键引脚为输入模式，LED 为输出模式，然后判断按键引脚电平，当为低电平时，延迟 10ms 后再次判断电平状态，如果还是低电平就确认按键按下，P41 输出低电平，LED 灭，按键松开，P41 输出高电平，LED 亮。

第八节 外部中断

1. STC8H 外部中断配置

中断系统是为使 CPU 具有对外界紧急事件的实时处理能力而设置的。

当中央处理机 CPU 正在处理某件事的时候外界发生了紧急事件请求，要求 CPU 暂停当前的工作，转而去处理这个紧急事件，处理完以后，再回到原来被中断的地方，继续原来的工作，这样的过程称为中断。实现这种功能的部件称为中断系统，请示 CPU 中断的请求源称为中断源。微型机的中断系统一般允许多个中断源，当几个中断源同时向 CPU 请求中断，要求为它服务的时候，这就存在 CPU 优先响应哪一个中断源请求的问题。通常根据中断源的轻重缓急排队，优先处理最紧急事件的中断请求源，即规定每一个中断源有一个优先级别。CPU 总是先响应优先级别最高的中断请求。

来举一个生活事例：

你打开火，烧上一壶水。然后去洗衣服，在洗衣服的过程中，突然听到水壶发出水开的报警声，这时，你停止洗衣服动作，立即去关掉火，然后将开水灌入暖水瓶中，灌完开水后，你又回去继续洗衣服。这个过程中实际上就发生了一次中断。

当 CPU 正在处理一个中断源请求的时候(执行相应的中断服务程序)，发生了另外一个优先级比它还高的中断源请求。如果 CPU 能够暂停对原来中断源的服务程序，转而去处理优先级更高的中断请求源，处理完以后，再回到原低级中断服务程序，这样的过程称为中断嵌套。这样的中断系统称为多级中断系统，没有中断嵌套功能的中断系统称为单级中断系统。

用户可以用关总中断允许位 (EA/IE. 7) 或相应中断的允许位屏蔽相应的中断请求，也可以用打开相应的中断允许位来使 CPU 响应相应的中断申请，每一个中断源可以用软件独立地控制为开中断或关中断状态，部分中断的优先级别均可用软件设置。高优先级的中断请求可以打断低优先级的中断，反之，低优先级的中

断请求不可以打断高优先级的中断。当两个相同优先级的中断同时产生时，将由查询次序来决定系统先响应哪个中断。

那么芯片是如何来切换到中断程序，中断程序执行完了后如何知道要回到哪里继续工作。芯片里有一种类似堆栈的机制，当发生中断的时候，把目前执行时的一些变量值，程序地址都记录到里面，然后去执行中断函数，当中断执行完成后，会根据前面记忆的程序地址返回回来，继续刚才的程序。要保存刚才这些状态，51 芯片内部有工作寄存器组 R0-R7 共有 4 组（分别是 BANK 0、1、2、3），在任何时刻，都只有 1 个工作组生效！这 4 个组（BANK）在 RAM 中的位置分别是 [00H, 07H]、[08H, 0FH]、[10H, 17H]、[18H, 1FH]，换句话说，RAM 中的 00H 地址、08H 地址、10H 地址、18H 地址，这四个地址的名字都叫 R0，那么在汇编程序中我们经常看到类似 MOV R0, #07 这样的语句，这个 #07 到底被放到了 RAM 的哪个地址中去了呢？00H?08H?10H?还是 18H? 到底是这 4 个地址中的哪一个，取决于 51 的 PSW 寄存器的 RS1 和 RS0 两个位，若 PSW.RS=2，就意味着第 2 组工作寄存器生效，R0 的地址就是 10H。

51 在上电后，PSW 的 RS 两个位默认为 0，也即 51 默认使用工作寄存器组 BANK 0，在默认状态下，对于普通的 C 语言函数，其传参、申请局部变量、导出函数的返回值等功能，将其翻译成汇编以后，肯定要使用 R0-R7；对于 51 的中断服务函数，它没有形参，也不用返回值，但是一般肯定有局部变量，这时就需要用到 R0-R7 了；试想，在执行普通函数时，R0-R7 已经被使用了，在执行普通函数时，一旦发生中断，而中断函数也需要使用 R0-R7，那怎么办？我们最先想到的是，在执行中断服务函数前先把 R0-R7 入栈，在中断服务完成后把 R0-R7 出栈，然后就能恢复现场，回到普通函数中去了，但是这 8 个 Rn 不能直接入栈，PUSH R0 这样的语句是不允许的，要想 R0 入栈只能用两句：MOV A R0; PUSH A; 这样的后果是，每次工作寄存器入栈都需要 2*8=16 条汇编语句才能完成，再加上 A、B、PSW 等寄存器入栈等，相当于每次中断都要消耗大量的时间来出入栈，影响程序速度。如何解决这一问题呢？51 提供了这样一种机制，切换工作寄存器组，过程如下：

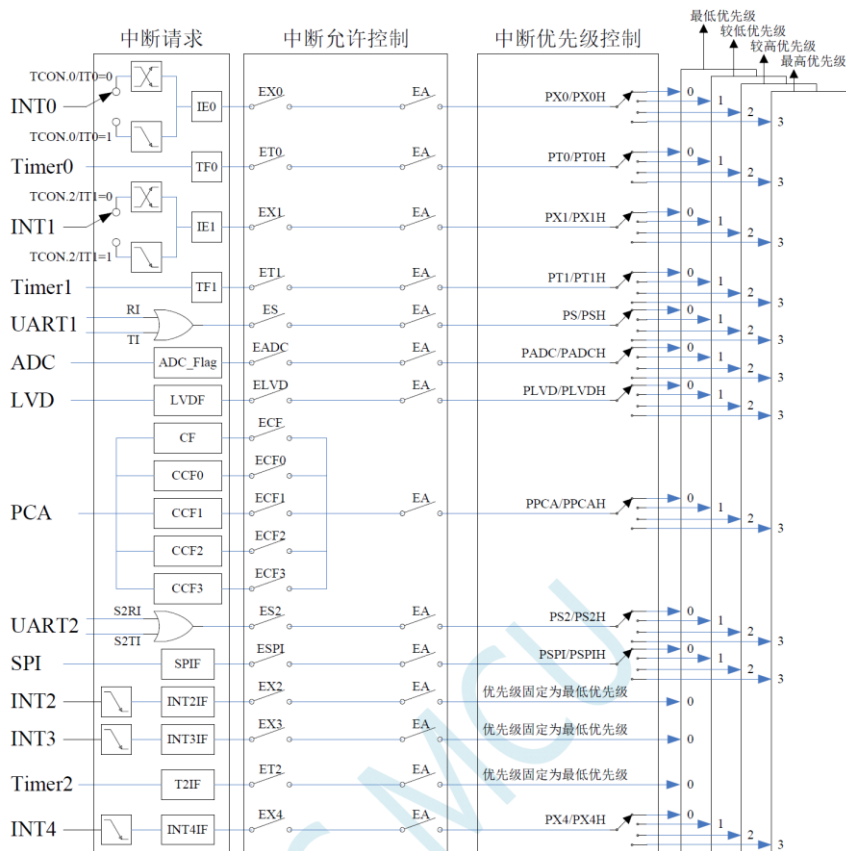
普通函数的执行过程中正在使用 BANK0 的 R0-R7，执行过程中突然发生了中断，而中断函数也想使用 R0-R7，在执行中断服务函数前，我们切换工作寄存器组，切换的具体方法就是直接修改 PSW 的 RS 两个比特位，而不必把 BANK 0 入栈，程序中使用 using 1，就是说，在进入外部中断 0 的服务函数前，先入栈 CPU 寄存器，再把工作寄存器组由 0 切换成 1，在退出中断服务后，先由 BANK1 切换回 BANK0，并弹出 CPU 寄存器，由于 BANK0 和 BANK1 处在不同的 RAM 空间，互不干扰，切换回 BANK0 之后就把那个普通函数的现场给恢复了。

对于同一优先级的中断，可以 using 同一个寄存器组，因为同一优先级的中断不会互相打断，也就是说，例如：我把同一优先级的中断函数都 using 2，这些函数也不会冲突地使用 R0~R7，他们只会分时复用 BANK 2。但是对于不同优先级的中断函数，必须手动设定为让他们使用不同的 BANK，因为高优先级的中断会打断低优先级的中断，如果使用相同的 BANK，一旦发生中断嵌套，低优先级服务函数正在使用的 R0~R7 将会被覆盖。

STC8H 的中断源有很多种类别，具体可以查看芯片手册的 11.1 节，本节我们只介绍外部中断，STC8H 外部中断有 5 个 INTO 到 INT4，天问 51 开发板上的 INTO 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-。

本节我们以独立按键 KEY1 为例，来讲解中断的使用。



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。

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。

2. 图形化编程

初始化

天问51初始化

设置引脚 P4_1 模式 推挽输出

设置引脚 P3_2 模式 高阻输入

设置外部中断 0 下降沿 触发

重复执行

外部中断 0 执行函数 INTO 寄存器组 1

写引脚 P4_1 电平 非 读取引脚 P4_1

3. 字符式编程

```
#include <STC8HX.h>
#include "lib/hc595.h"
#include "lib/rgb.h"
#include "lib/delay.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 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();
  }
}
```

通过上述程序，我们可以看到这么一句代码：

```
void INT0(void) interrupt 0 using 1
```

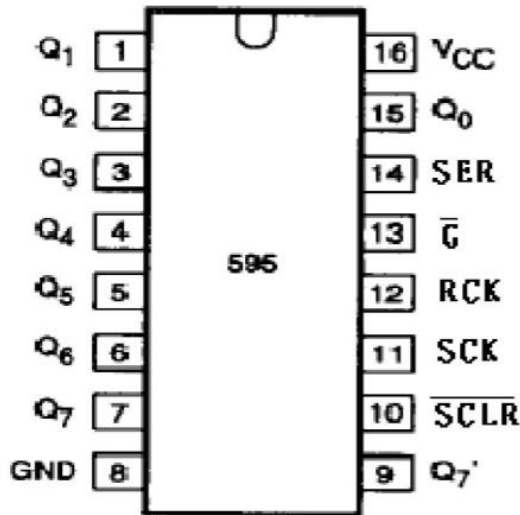
其中 interrupt 和 using 都是 C51 里的关键字，其中 interrupt 0 表示用的是第 0 号中断入口，每一个中断都有指定的编号，INT1 为 2，其它的我们可以查看芯片手册的 11.3 STC8H 系列中断列表。其中 INT0(void)为自己定义的函数，这样硬件上只要发现有中断进来，程序会自动跳到 INT0(void)的函数，执行中断函数里面的代码。另外 using 1 表示进入中断使用了芯片内部的第一组寄存器。

4. 运行效果

每按一次按键，LED 灯的亮灭状态就变化一次。

第九节 74HC595 移位寄存器

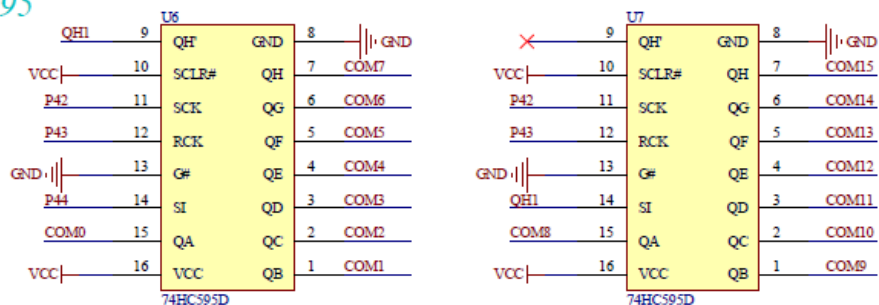
前面几节，我们已经用到过 74HC595 芯片，通过控制 595 来选通指定的 COM 口。这一节我们来看看 74HC595 的内部原理。



74HC595D 是一种具有 8 位锁存、8 位串行输入、8 位串/并行输出、串—并移位寄存器和三态输出功能的通用 LED 驱动芯片。

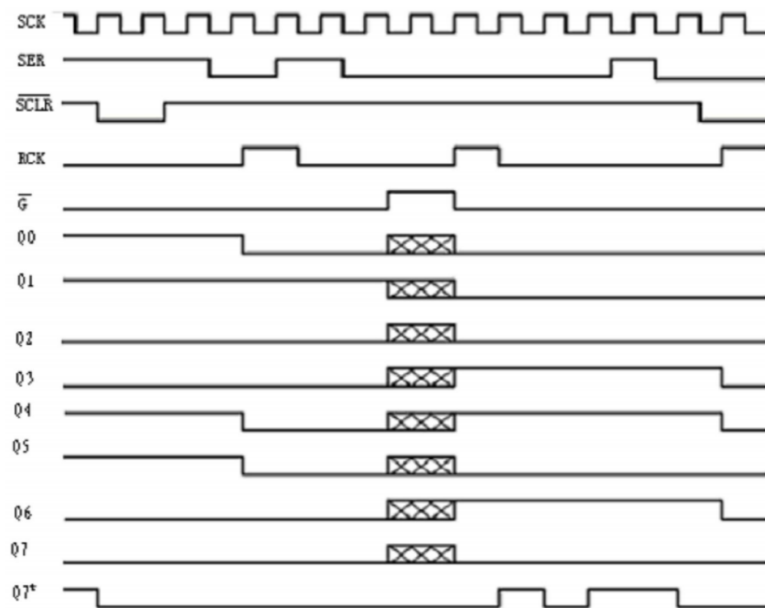
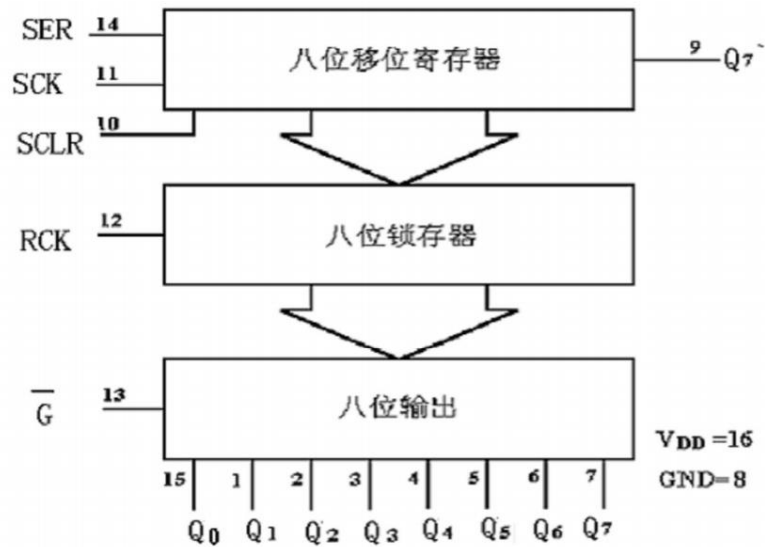
管脚序号	符号	管脚功能
10	$\overline{\text{SCLR}}$	复位端
11	SCK	移位寄存器时钟，上升沿移位
12	RCK	锁存寄存器时钟，上升沿存储
13	$\overline{\text{G}}$	输出使能端，为低电平使，输出选通；为高电平时，输出为 3 态
14	SER	串行数据输入端
15、1--7	Q0----Q7	并行输出端
9	Q7'	串行输出
8	GND	逻辑地
16	V _{CC}	电源端

74HC595



595 具有扩展功能，可以多个芯片级联，天问 51 开发板上用到了两个 595。对应上图引脚说明，天问 51 开发板上的第 11 脚 SCK 移位寄存器时钟、第 12 脚 RCK 锁存寄存器时钟、第 14 脚 SI 串行数据输入，三个做为 595 控制输入，连接

到对应的 STC8H 芯片的 P42、P43、P44。第 10 脚的 SCLR 复位端连接到电源，默认高电平；第 13 脚的 G 连接到低，默认低电平，使能输出。U6 的第 9 引脚的 QH 串行数据输出口连接到了 U7 的第 14 脚 SI 数据输入。U6 的并行输出端口对应连接到数码管的 COM0-COM7，U7 的并行输出端口对应连接到点阵的 COM8-COM15。



通过上图，我们能够知道数据需要输入到 595 芯片里，需要在 SCK 引脚发送高低电平脉冲，数据在 SCK 的上升沿状态时输入到芯片内部，在 RCK 的上升沿进入的存储寄存器中去，因为我们已经把 G 端口默认设置为低电平，所以在寄存器中的数据会直接输出到并行端口上。

如果我要设置 COM0 到 COM7 这 8 个端口电平状态全部为高电平。只需要设置引脚 P44 引脚 SI 一直为高电平，然后让 P42 引脚 SCK 发送 8 个脉冲，这样已经有 8 个高电平的数据存储在 595 芯片内部了，接下来我们把 P43 引脚 RCK 发送一

个脉冲，这样就把内部存储的数据输出到并行端口上了，于是对应的 COM0-COM7 全部输出为高电平。所以我们前面几节里有一个天问 51 初始化的函数，里面有一个 595 的初始化，这个就是把 595 芯片的并行端口默认输出低电平，因为不初始化的话，595 芯片内部的储存器内的数据是不确定的，这样会导致并行端口每次开机都不一样。至于这个脉冲的周期，芯片手册里有对应的时间要求。

接下来我们来看下对应的代码实现。

```
/*引脚定义*/
#ifndef HC595_DS
#define HC595_DS P4_4 //SI
#endif

#ifndef HC595_DS_MODE
#define HC595_DS_MODE {P4M1&=-0x10;P4M0|=0x10;} //推挽输出
#endif

#ifndef HC595_STCP
#define HC595_STCP P4_3 //RCK
#endif

#ifndef HC595_STCP_MODE
#define HC595_STCP_MODE {P4M1&=-0x08;P4M0|=0x08;} //推挽输出
#endif

#ifndef HC595_SHCP
#define HC595_SHCP P4_2 //SCK
#endif

#ifndef HC595_SHCP_MODE
#define HC595_SHCP_MODE {P4M1&=-0x04;P4M0|=0x04;} //推挽输出
#endif
#endif
```

先是用宏定义，定义三个控制引脚，方便我们后面的程序好理解。

```
//=====
// 描述：595 初始化引脚。
// 参数：none.
// 返回：none.
//=====

void hc595_init()
{
    HC595_DS_MODE; //引脚初始化
    HC595_STCP_MODE;
    HC595_SHCP_MODE;

    HC595_DS = 0;
    HC595_STCP = 0;
    HC595_SHCP = 0;
}
```

接下来就是三个引脚的端口设置，设置为推挽输出。

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

```
// 描述: 595 发送 8 位数据.
// 参数: 8 位数据.
// 返回: none.
//=====
void hc595_send_byte(uint8 outdata)
{
    uint8 i;
    for(i=0;i<8;i++) //将 8 位数据按位发送, 先发高字节再发低字节
    {
        if((outdata&0x80)==0x80)
        {
            HC595_DS = 1;
        }
        else
        {
            HC595_DS = 0;
        }
        HC595_SHCP = 0; //时钟线低电平
        HC595_SHCP = 1; //时钟线高电平
        outdata = outdata<<1;
    }
}
```

接下来就是根据我们要发送的 8 为数据的每一位高低电平状态, 来发送数据到 595 芯片内部的存储器里。

```
//=====
// 描述: 595 使能输出.
// 参数: none.
// 返回: none.
//=====
void hc595_out_enable()
{
    HC595_STCP = 0;
    delay10us();delay10us();
    HC595_STCP = 1;
    delay10us();delay10us();
    HC595_STCP = 0;
}
```

接下来就是设置 RCK 芯片脉冲, 来让 595 内部储存器数据输出到并行端口, 我们看到这里延迟 20us, 就是脉冲时间。

这样 595 最基本的功能都已经完成了, 对于前面几节数码管和点阵里用到的 595 几个函数, 如下:



就是设置对应的 COM 端口输出状态，对应的函数，有兴趣可以自己去看 hc595.h 驱动文件。

第二章 STC8H 的定时器

第一节 STC8H 定时器综述

第二节 定时器模式

第三节 定时器中断

第四节 设计一个倒计时

1. 图形化编程
2. 字符式编程
3. 运行效果

第五节 设计一个计数器

1. 图形化编程
2. 字符式编程
3. 运行效果

第六节 万年历实现

1. 图形化编程
2. 字符式编程

3. 运行效果

第七节 简易计算器

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果

第八节 定时器模拟 PWM

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果

第三章 STC8H 的 ADC 模块

第一节 STC8H ADC 模块综述

第二节 电位器

1. 硬件设计

2. 图形化编程

3. 字符式编程

4. 运行效果

第三节 光敏传感器吧

1. 硬件设计

2. 图形化编程

3. 字符式编程

4. 运行效果

第四节 NTC 温度传感器 A

1. 硬件设计

2. 图形化编程

3. 字符式编程

4. 运行效果

第四章 STC8H 的 PWM 模块

第一节 STC8H PWM 模块综述

第二节 震动马达

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果

第三节 蜂鸣器

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果

第四节 模拟 DAC

1. 硬件设计
2. 图形化编程

3. 字符式编程

4. 运行效果

第五章 STC8H 串口通信

第一节 STC8H 串口模块综述

第二节 轮询模式

1. 图形化编程

2. 字符式编程

3. 运行效果

第三节 中断模式

1. 图形化编程

2. 字符式编程

3. 运行效果

第四节 上位机控制

1. 图形化编程

2. 字符式编程

3. 运行效果

第六章 STC8H 的 I2C 模块（主机模式）

第一节 STC8H I2C 模块综述

第二节 RTC 实时时钟

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果

第三节 加速度传感器

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果

第四节 OLED 显示屏

1. 硬件设计

2. 图形化编程
3. 字符式编程
4. 运行效果

第七章 STC8H 的 SPI 模块

第一节 STC8H SPI 模块综述

第二节 SPI Flash

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果

第八章 STC8H 的比较器

1. STC8H 比较器模块综述
2. 比较输出实验
3. 硬件设计
4. 图形化编程

5. 字符式编程

6. 运行效果

第九章 STC8H 的 EEPROM

第一节 STC8H EEPROM 模块综述

第二节 EEPROM 实验

1. 图形化编程

2. 字符式编程

3. 运行效果

第十章 STC8H 的看门狗

第一节 STC8H 看门狗模块综述

第二节 看门狗实验

1. 图形化编程

2. 字符式编程

3. 运行效果

第十一章 STC8H 的低功耗模式

第一节 STC8H 看门狗模块综述

第二节 看门狗实验

1. 图形化编程
2. 字符式编程
3. 运行效果

第四篇 中级篇（总线式模块和扩展模块应用）

第一章 单总线

第一节 单总线综述

第二节 RGB 彩灯

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果

第三节 DHT11 温湿度传感器

1. 硬件设计

2. 图形化编程
3. 字符式编程
4. 运行效果

第四节 DS18B20 温度传感器

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果
5. 运行效果

第二章 并口总线

第一节 并口总线综述

第二节 LCD 1602

1. 硬件设计
2. 图形化编程
3. 字符式编程

4. 运行效果

第三节 LCD 12864

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果

第四节 TFT 彩屏

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果

第三章 扩展模块

第一节 超声波模块

1. 硬件设计
2. 图形化编程

3. 字符式编程

第二节 舵机模块

1. 硬件设计
2. 图形化编程
3. 字符式编程

第三节 继电器模块

1. 硬件设计
2. 图形化编程
3. 字符式编程

第四章 红外编码和解码

第一节 红外通讯协议综述

第二节 红外接收

1. 硬件设计
2. 图形化编程
3. 字符式编程

4. 运行效果

第三节 红外发送

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果

第五章 STC8H 的 PWM 高级模式

第一节 捕获模式

1. 硬件设计
2. 图形化编程
3. 字符式编程
4. 运行效果

第二节 互补输出

1. 硬件设计
2. 图形化编程

3. 字符式编程

4. 运行效果

第六章 STC8H 的 I2C 从机模式

1. 模拟 I2C 主机模式

2. 主从通讯实验

第七章 工业总线

第一节 工业应用总线综述

第二节 485 总线

1. 硬件设计

2. 图形化编程

3. 字符式编程

4. 运行效果

第五篇 高级篇（高级应用案例）

第一章 文件系统

第一节 文件系统综述

第二节 SD 卡读写

1. 硬件设计
2. 字符式编程
3. 运行效果

第三节 FAT 文件系统实现

1. 字符式编程
2. 运行效果

第二章 USB

第一节 硬件设计

第二节 虚拟串口

1. 字符式编程
2. 运行效果

第三节 HID 设备

1. 字符式编程
2. 运行效果

第四节 大容量设备

1. 字符式编程
2. 运行效果

第五节 MIDI 音乐

1. 字符式编程
2. 运行效果

第三章 物联网

第一节 物联网综述

第二节 蓝牙通讯

1. 硬件设计
2. 字符式编程
3. 运行效果

第三节 2.4G 无线通讯

1. 硬件设计
2. 字符式编程

3. 运行效果

第四节 以太网通讯

1. 硬件设计

2. 字符式编程

3. 运行效果

第五节 Wi-Fi 通讯

1. 硬件设计

2. 字符式编程

3. 运行效果

第六篇 综合应用（实战案例）

第一章 运动手表

第一节 功能需求分析

第二节 硬件搭建

第三节 程序框图

第四节 字符式编程

第五节 运行效果

第二章 数码相框

第一节 功能需求分析

第二节 硬件搭建

第三节 程序框图

第四节 字符式编程

第五节 运行效果

第三章 手写板

第一节 功能需求分析

第二节 硬件搭建

第三节 程序框图

第四节 字符式编程

第五节 运行效果

第四章 示波器

第一节 功能需求分析

第二节 硬件搭建

第三节 程序框图

第四节 字符式编程

第五节 运行效果

第五章 智能家居

第一节 功能需求分析

第二节 硬件搭建

第三节 程序框图

第四节 字符式编程

第五节 运行效果

附录

嵌入式 C 关键字

数据类型

常用编码格式和转换

ASSIC 码表

代码格式规范

常用算法