智能制造基础技术系列教材
“互联网＋”新形态一体化教材

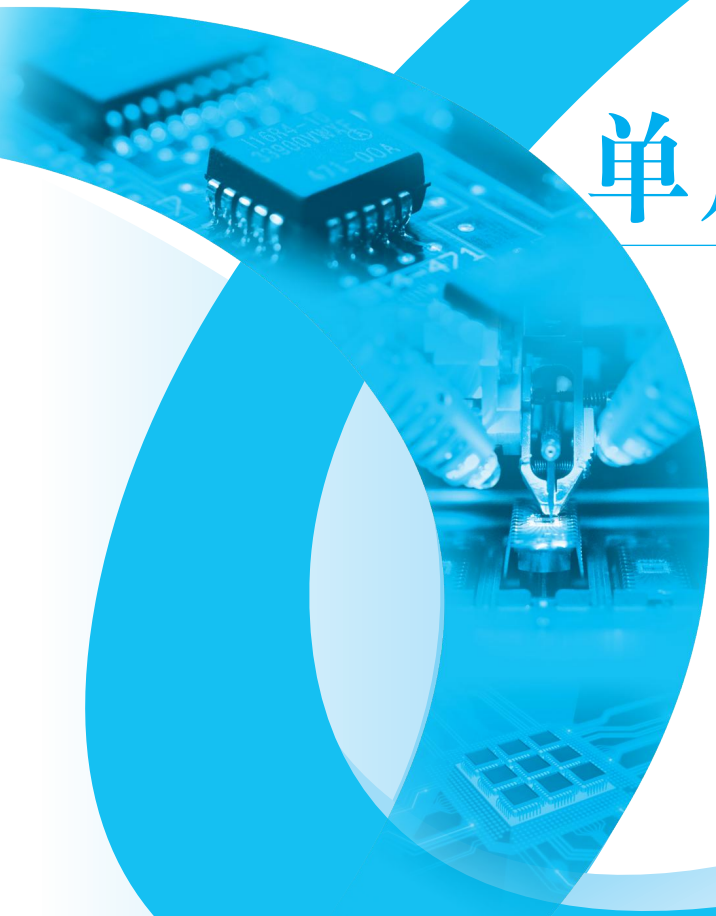
单片机应用技术

主编　胡祝兵
主审　韩志凌

教材附赠
微课视频
教学课件
素材文件

中国科学技术出版社
CHINA SCIENCE AND TECHNOLOGY PRESS

智能制造基础技术系列教材
“互联网＋”新形态一体化教材



单片机应用技术

主编　胡祝兵
主审　韩志凌

中国科学技术出版社
·北　京·

图书在版编目(CIP)数据

单片机应用技术/胡祝兵主编. --北京：中国科学技术出版社，2023.12
ISBN 978-7-5236-0381-9

Ⅰ.①单… Ⅱ.①胡… Ⅲ.①微控制器-高等学校-教材 Ⅳ.①TP368.1

中国国家版本馆 CIP 数据核字(2023)第 235727 号

策划编辑	王晓义
责任编辑	杨　洋
封面设计	唐韵设计
正文设计	梧桐影
责任校对	邓雪梅
责任印制	徐　飞

出　　版	中国科学技术出版社
发　　行	中国科学技术出版社有限公司发行部
地　　址	北京市海淀区中关村南大街 16 号
邮　　编	100081
发行电话	010-62173865
传　　真	010-62173081
网　　址	http://www.cspbooks.com.cn

开　　本	787mm×1092mm　1/16
字　　数	380 千字
印　　张	16.5
版　　次	2023 年 12 月第 1 版
印　　次	2023 年 12 月第 1 次印刷
印　　刷	北京荣玉印刷有限公司
书　　号	ISBN 978-7-5236-0381-9/TP・465
定　　价	56.00 元

(凡购买本社图书，如有缺页、倒页、脱页者，本社发行部负责调换)

前　言

20 世纪 70 年代，随着微型计算机的出现，单片机诞生了，由于电子技术的发展和应用，单片机应用技术在家用电器、工业智能仪器仪表、机电设备、信息处理、航空航天等领域的应用越来越广泛。

笔者遵循有效教学的基本规律，结合单片机应用技术的学习特点，基于“教学项目化、学习自主化、学生角色双元化”的行动导向教学模式，通过对企业调研、岗位分析、职业能力分析制定了全新的课程标准，基于实际工程项目，设计了项目化教学的结构。

本书共有 5 个教学项目：项目一设计与实现彩灯控制电路；项目二设计与实现电子时钟；项目三设计与实现温度测量系统；项目四设计与实现物体姿态远程测控系统；项目五设计与实现电子秤。将单片机应用技术的知识点和关键内容融入这 5 个项目中，基于 Proteus 和 Keil C，通过仿真与硬件调试，虚实结合，使学生在 5 个项目的动手操作过程中，既能掌握单片机的基本知识，还可以熟悉单片机控制系统的硬件设计和软件设计的方法。

本书是笔者在多年教学、指导毕业设计、各类竞赛和工程实践经验的基础上，通过大量的工程和仿真实例，以上述 5 个教学项目为载体，基于 Proteus 和 Keil C，全方位地介绍了 51 系列单片机的基础知识、单片机控制系统的硬件设计和软件设计的方法和技巧。党的二十大报告提出推进数字化教育，建设全民终身学习的学习型社会、学习型大国。数字化教育是未来教育的发展趋势，基于此，本书配备了微课视频、参考程序、习题测试等电子资源，扫描书中的二维码即可获得学习资源，方便学生根据自身需要有选择性地自主学习。

本书由单片机应用技术课程教学团队编写，胡祝兵担任主编，康金、雷鹏娟、魏战刚担任副主编，其中，项目一、二、三、五主体部分由胡祝兵编写，项目四由康金编写，各项目中素质目标的要点和拓展学习部分由魏战刚编写，微课由胡祝兵、雷鹏娟和康金共同录制，全书由胡祝兵统稿，韩志凌审稿。此外，刘剑在习题测试、部分原理图设计及相应程序调试中给予了很大的帮助；承德市中瑞自动化工程有限公司总经理高树红对本书的编写提出了许多宝贵的意见和建议，在此表示衷心的感谢。

由于笔者水平有限，书中难免存在不足之处，敬请广大读者批评指正。

本书可以作为单片机学习爱好者和工程实践者的参考书，也可以作为教师教学参考用书。笔者还为广大一线教师提供了本书的教学资源，有需要者可致电 13810412048 或发邮件至 2393867076@ qq. com。

在线课程学习指南

本教材是国家智慧教育公共服务平台省级精品在线课程《单片机应用技术》的配套教材，在线课程设置了与教材同步的项目任务内容、微课与在线测试，登录国家智慧教育公共服务平台即可在线自主学习。

1. 输入平台网址：https：//www. smartedu. cn，搜索框中输入“单片机应用技术”，点击搜索。

2. 进入课程学习界面，根据开课周期自主选择学习时间与内容。

目　录

项目一

设计与实现彩灯控制电路

知识目标

了解单片机的结构和工作原理；

掌握单片机的最小系统和 I/O 接口工作原理；

掌握单片机存储器结构及分布；

掌握 Proteus 软件和 KEIL 软件的基本操作；

掌握 C51 的基本程序结构，掌握分支语句和循环语句的含义。

能力目标

能设计单片机最小系统；

能基于单片机 I/O 接口，设计典型的 LED 显示电路，实现各类流水灯功能；

会基于 Proteus 软件设计指示灯、流水灯电路，利用 KEIL 软件完成程序设计、调试，实现自定义功能。

素质目标

端正学习态度，加强责任感；

具备规则意识，对制度、规程等有敬畏感并自觉践行；

树立崇尚科学的精神，坚持求真、求实的科学态度，夯实严谨、细致的工作作风；

注重研讨、协作，有较强的集体意识和团队合作精神，学以致用，坚持理论联系实际。

任务一

初识单片机

一、什么是单片机

单片机(single-chip microcomputer)是在一片集成电路芯片上集成了微处理器(CPU)、存储器(ROM 和 RAM)、片内资源(定时器、计数器等)和多种输入/输出接口电路(I/O 接口电路),从而构成的单芯片微型计算机。STC89 系列、双列直插式单片机外形图,如图 1-1 所示。

图 1-1　STC89 系列单片机外形图

单片机其实离我们的生活很近,现在的家用电器基本上都采用了单片机控制,从电饭锅、洗衣机、电冰箱、空调、电视、音响视频器材到电子称量设备,五花八门,无所不在。

单片机已经应用于我们生活中的各个领域,几乎很难说出哪个领域没有单片机的踪迹。计算机的网络通信与数据传输,工业自动化过程的实时控制和数据处理,广泛使用的各种智能 IC 卡,民用豪华轿车的安全保障系统,录像机、摄像机、全自动洗衣机的控制,以及程控玩具、电子宠物等都离不开单片机,更不用说自动控制领域的机器人、智能仪表、医疗器械了。单片机广泛应用于仪器仪表、家用电器、医用设备、航空航天、专用设备的智能化管理及过程控制等领域,因此,单片机的学习、开发与应用将造就一批计算机应用与智能化控制的科学家、工程师。只需掌握单片机硬件 MCU、指令系统、软件编程、接口芯片等的原理及应用,就可以成为一名单片机开发工程师。当前,世界正经历百年未有之大变局,新一轮科技革命和产业变革方兴未艾。党的二十大吹响了以中国式现代化全面推进中华民族伟大复兴的嘹亮号角,我国正加快从制造大国迈向制造强国。要在全球科技革命和产业变革中赢得主动权,要实现中国式现代化,我国亟须一大批具有国际视野、创新能力、担当精神的工程师集智攻关。掌握单片机知识,成长为单片机开发工程师,和其他科学家和工程师一起,为实现中国梦添砖加瓦,何其光荣!

二、单片机的组成

单片机的组成主要包括中央处理器、存储器、输入/输出接口等,还包含了中断系统、

总线控制系统、定时计数器系统等，在后续的章节中会一一介绍。8051 系列单片机的引脚图，如图 1-2 所示。

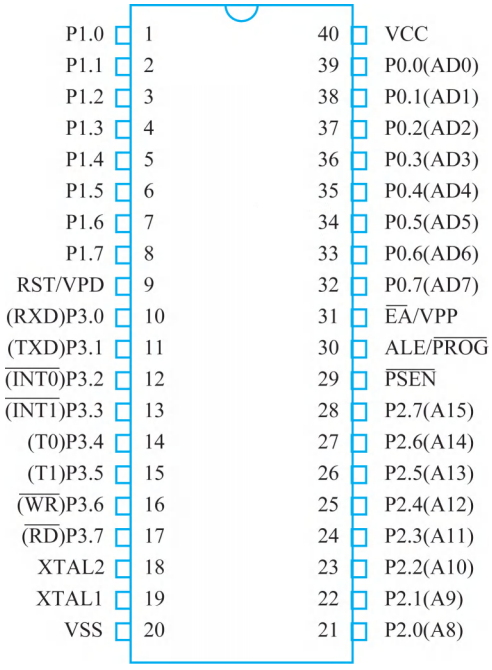


图 1-2　8051 系列单片机的引脚图

(一) 中央处理器

也叫 CPU，包括运算器、控制器和寄存器，是单片机的核心，相当于人的大脑，所有的事情都由它来指挥。CPU 是单片机的控制核心，其核心地位是单片机有条不紊地进行控制的基础。

(二) 存储器

一般包括 ROM 和 RAM，目前，大多数的单片机还有 E2PROM。ROM 是只读存储器，用来存储编写好的程序，所以又叫程序存储器。它的最大特点是断电后程序并不丢失，但下载程序需要专门的电路或者软件的配合，这有点类似于 PC 机主板中存储 BIOS 的存储器。RAM 是随机读取存储器，用来存储程序运行时处理的一些数据变量，比如你定义一个无符号字符型变量，它就占用了一个字节的 RAM 空间。它的最大特点是断电后数据就消失了，这类似于 PC 机内存，就像你在用软件做设计时，突然断电了，如果没有备用电源，电脑就关机了，再一开机，你发现辛苦一上午做的设计全没了，一下又回到昨天晚上什么都没做的状态了。如果数据在运算后需要保存，就需要用到 E2PROM。E2PROM 是电可擦除可编程只读存储器。它的最大特点是断电后数据不丢失，可编程，一般通过电脉冲擦除，主要用来存储需要保存的运算结果，这类似于我们用到的闪存。以前进行单片机开发时，对存储器的规划是十分讲究的，因为资源非常有限，MCS51 只有 4KB 的 ROM，

128B 的用户 RAM，没有 E2PROM，所以，整个开发过程变成了如何榨取存储空间的过程。当然，现在就没有那么大的压力了，因为目前市场上销售的单片机的存储空间越来越大，对于开发者来说，存储空间的使用变得得心应手了。

(三)输入/输出接口

1. 电源线

VCC(引脚号 40)：芯片电源，接+5V。VSS(引脚号 20)：接地端。

2. 时钟线

XTAL1(引脚号 19)：内部振荡电路反相放大器的输入端，是外接晶振的一个输入引脚。

XTAL2(引脚号 18)：内部振荡电路反相放大器的输出端，是外接晶振的另一个输入引脚。

3. 控制总线

ALE/$\overline{\text{PROG}}$(引脚号 30)：地址锁存允许，主要功能是提供一个定时的时钟。

$\overline{\text{EA}}$/VPP(引脚号 31)：访问外部存储器控制信号。如果使用内部 ROM 作为程序存储器，此引脚需接高电平(VCC)；如果使用外部 ROM 作为程序存储器，则要将此引脚接地。

RST/VPD(引脚号 9)：复位信号输入端。当系统主电源发生故障，降低到规定的电压以下时，可以通过 VPD 端为单片机提供备用电源，以保证存储在单片机中的 RAM 中的信息不会丢失。

$\overline{\text{PSEN}}$(引脚号 29)：外部程序存储器 ROM 读选通信号。当单片机需要从外部 ROM 读取指令或数据时，此引脚输出低电平信号。控制总线一般是单片机在跟外围设备进行数据交互时用到，但现在用得越来越少了，读者以了解为主，熟悉了单片机的工作机理之后，可以尝试应用。

4. 输入/输出

P0.0~P0.7(引脚号 39~32)：双向输入/输出端口。

P1.0~P1.7(引脚号 1~8)：双向输入/输出端口。

P2.0~P2.7(引脚号 21~28)：双向输入/输出端口。

P3.0~P3.7(引脚号 10~17)：双向输入/输出端口。当该端口不作为输入/输出端口使用时，每一个引脚也可以有第二功能，如：P3.0(RXD)：串行输入口；P3.1(TXD)：串行输出口；P3.2($\overline{\text{INT0}}$)：外部中断 0 输入口；P3.3($\overline{\text{INT1}}$)：外部中断 1 输入口；P3.4(T0)：定时/计数器 0 外部事件脉冲输入口；P3.5(T1)：定时/计数器 1 外部事件脉冲输入口；P3.6($\overline{\text{WR}}$)：写信号；P3.7($\overline{\text{RD}}$)：读信号。

三、单片机控制彩灯

在生活中，我们会看到很多各种各样的彩灯。如交通路口的红绿灯，城市景观亮化

灯，各种节日彩灯，比如河北省承德市的武烈河大桥就安装有七彩灯，所以本地人又把这座桥叫作彩虹桥，到了夜晚景色很美。如果你来承德旅游，夜晚可以漫步武烈河畔，欣赏美丽的彩虹桥，见图 1-3。

这些彩灯其实就是由各种样式和颜色的 LED 灯组成的。图 1-4 是直插式 LED 灯，它有各种各样的颜色，有两个引脚，一般情况下，一个引脚长，为正极，一个引脚短，为负极，也就是“长正短负”。当二极管正向导通，流过足够的电流时，LED 灯就会发光，这个电流一般为 10mA 左右(不同型号有差别)。另外，不同的颜色和型号，正向导通压降是不一样的，一般为 2V 左右，在设计电路时，需要注意。

图 1-3　武烈河大桥夜景

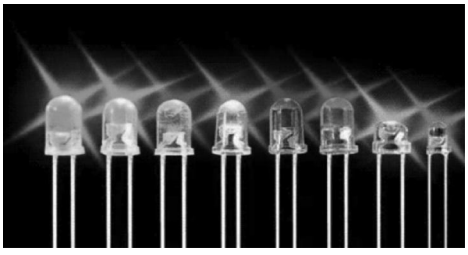


图 1-4　直插式 LED 灯

(一) 单片机可以控制彩灯吗

我们日常生活中经常用到计算机，也就是 PC 机(personal computer)。它由输入设备(键盘、鼠标、游戏手柄等)、输出设备(显示器等)、存储设备(硬盘、内存等)、CPU(控制器和运算器等)，以及其他各种各样的接口(如 USB 接口、HDMI 接口、耳机接口、麦克风接口等)组成。计算机的结构越复杂，功能就越多。在生活中，如果我们用一台计算机去控制上下课铃，明显就是大材小用了。因此，为了节省成本，在一块芯片上面集成了一些必要的功能(包括 CPU、存储器、输入/输出接口、中断系统、定时器等)。它的处理能力、存储容量、接口和其他资源的数量都是非常有限的，但可以专门用来做一些控制和数据测控系统等。随着技术的发展，目前很多单片机集成的资源越来越多，功能也就越来越强大，家里的智能冰箱、智能空调、智能洗衣机、智能体重秤等，控制核心都是强大的单片机，这也是我们学单片机的原因。可以预见，未来单片机的发展会越来越快，这也需要我们不断地探究、学习，养成终身学习的习惯，学以致用，坚持理论联系实际，才能更好地服务社会。

想一想

生活中，有哪些控制系统可以由单片机完成控制？

(二) MCS 系列单片机

MCS-51 系列单片机是市场中应用最广泛的，也是初学者最先接触到的。它由 Intel 公

司生产，包括 8 位的 CPU，4K 的 ROM、128B 的用户 RAM，4 个 8 位的并行 I/O 口，1 个全双工串行口、2 个 16 位定时/计数器、2 个外部中断。后来，Intel 公司向外出售了 51 单片机的核心技术，51 单片机一般就是兼容 8031 指令系统的单片机的统称了。如 AT89 系列、STC89 系列、MPC89 系列等，都是兼容 8031 指令系统的，我们都统称为 51 单片机。目前，51 单片机国产化产品虽然很多，但相对而言，处理能力比较弱，属于低端处理芯片。近些年，面对来自高端芯片国际市场的技术封锁，中芯国际和华为等企业迎头赶上，不断加强研发力度，已经生产出 7nm 高端芯片。中国的高端芯片技术未来可期。

目前，市场上的 51 单片机，资源越来越丰富，功能越来越强大，比如有的单片机集成了 PWM、AD、DA 等，为开发者提供了很多便利，因此，我们在使用时，要选择适合自己项目的单片机。本课程在讲解时，并不针对哪一个具体的 51 单片机，只要是 51 系列的，都是兼容的，理解这一点，对单片机的开发是很有帮助的。单片机可以控制彩灯吗？答案当然是肯定的，因为它的资源丰富、功能强大、可编程，因此，可以用来控制各种各样的彩灯。本项目的具体任务就是设计一个单片机控制简单的彩灯系统。设计程序，基于仿真和硬件调试的方式，介绍单片机控制系统的开发过程，最终实现具有多种工作状态的彩灯控制系统。

任务二

设计单片机最小系统

一、单片机最小系统的组成

单片机最小系统是指让单片机正常工作所必需的基本电路，主要由电源电路、复位电路、时钟电路组成。

(一) 电源电路

向不同的单片机供电的电压不一样，如：AT89S51 单片机的工作电压范围为 4.0～5.5V，一般接 5V 直流电源。电源正极接在 VCC，也就是单片机的 40 号引脚(这里指常见的 40 引脚的单片机，其他类型可能稍有差别)，电源地接 VSS，也就是单片机的 20 号引脚，这是数字系统的电源。单片机电源原理图如图 1-5 所示，如果系统中有模拟电路需要电源，最好进行隔离。

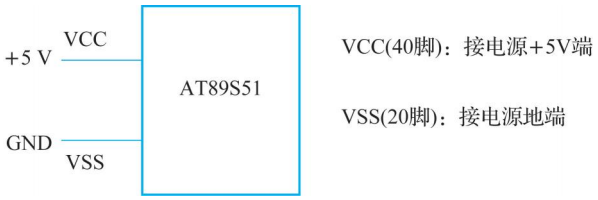


图 1-5　单片机电源原理图

(二) 时钟电路

单片机工作的时钟基准，决定了单片机工作速度，也就是节拍，相当于人体的心脏。目前，很多单片机都有内部时钟振荡电路，但随着单片机工作时温度的变化，频率会有细微的变化。所以，在要求精度高的地方，如串行通信，就需要外接时钟电路，这也是多数都采用外接晶振的原因。外接晶振的频率越高，单片机工作速度越快，但频率不是无限增加的。如 AT89S51 单片机的时钟频率范围为 0～33MHz，接法如图 1-6 所示，晶振接在单片机的 18 号和 19 号引脚，图中的电容 C1 和 C2 起稳定作用，电容值的大小根据不同的单片机有所差别，需要查看单片机的手册，根据推荐值进行选择。

而晶振是石英晶体振荡器的简称，晶振实物图如图 1-7 所示。

晶振又分为无源和有源，51 单片机一般均采用 11.0592MHz 的无源晶振。这个速度的设置是为了通信时能得到一个合适的波特率(后面讲到串行通信时再详细分析)。

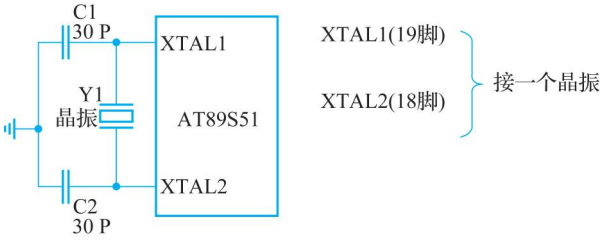


图 1-6　单片机时钟电路

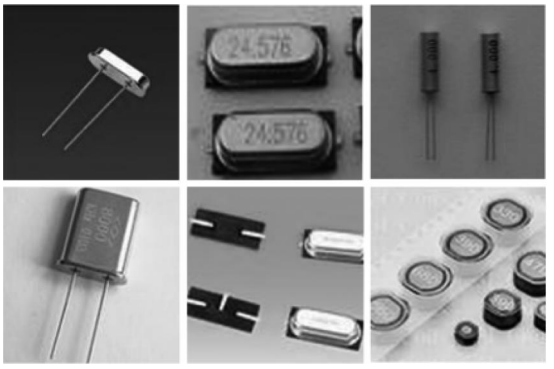


图 1-7　晶振实物图

(三)复位电路

复位电路产生复位信号，使单片机从起始状态开始工作，完成单片机的“启机”过程。51 单片机一般是高电平复位的，如果 RST 引脚维持 2 个机器周期的高电平(当然，为了保证可靠复位，一般都让时间大于这个值)，那么内部寄存器将会被置为合适的数值，使得系统顺序启动。正常工作时，RST 脚保持低电平。需要注意的是：时钟电路振荡频率一般用 fosc 表示，它等于晶振频率；时钟电路振荡周期等于 fosc 的倒数；对于传统的 12T(T 是指令周期，一般 51 单片机内核是 12 个时钟周期为 1 个指令周期，也就是说，晶振跳 12 下运行一个指令)的 51 单片机来说，机器周期等于振荡周期×12。例如：晶振频率 = 12MHz；振荡频率 = 12MHz；振荡周期 = 1/12μs；则机器周期 = 1μs。而现在有很多单片机是 1T、4T、6T 模式，分别对应是机器周期等于晶振周期的 1 倍、4 倍和 6 倍。也就是说，如果是 1T 单片机，理论上，其运算速度是普通单片机的 12 倍(实际没有)，这在使用时应当注意。

复位电路连接方式一般有以下三种。

1. 上电复位

指单片机接通电源时产生复位信号，完成单片机启动，使单片机回到初始工作状态。典型的 51 单片机的上电复位电路如图 1-8 所示。

当电源没有接通时，RST 引脚为低电平，当电源接通的瞬间，由于电容两端的电压不能突变，所以 RST 为高电平，随着电容的充电，RST 引脚的电压开始下降，最终降为低电

平，这个高电平持续的时间与 $R * C$ 的值有关系，精确的时间可以通过建模(学过动态电路分析或者自动控制原理的同学，可以试着建模)计算得到，因此可以通过调整 R 和 C 的值来调整高电平持续的时间。一般，电容取 10μF，电阻取 5.6kΩ 或者 10kΩ，可以保证单片机可靠复位。

2. 手动复位

手动按键产生复位信号，完成单片机启动，确定单片机的初始状态。按键复位电路如图 1-9 所示。

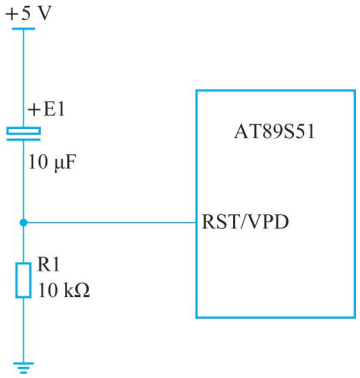


图 1-8　单片机上电复位电路

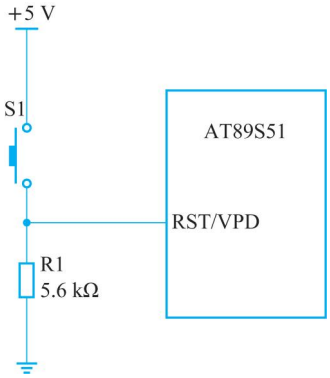


图 1-9　按键复位电路

当按键按下时，RST 为高电平，单片机复位；松开按键，RST 为低电平，单片机正常工作。通常在单片机工作出现混乱或“死机”时，使用手动复位可实现单片机“重启”。

3. 混合复位电路

将上电复位电路和手动复位电路结合到一起，我们通常使用的都是这种混合复位电路。图 1-10 所示是混合复位电路，既能够上电复位，也能够手动复位。

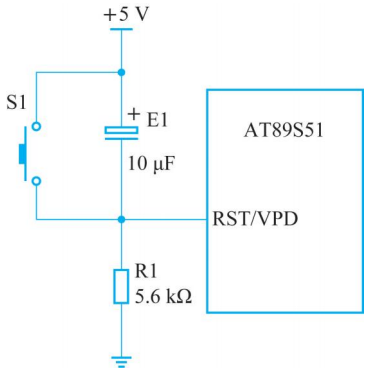


图 1-10　混合复位电路

(四) 单片机最小系统的实现

单片机最小系统电路图如图 1-11 所示。注意，$\overline{\text{EA}}$引脚一般接高电平，表示使用的是

单片机内部的 ROM，这也是最常用的情况。至于 P0、P1、P2、P3，是单片机的 I/O 接口，后面再详细介绍。

视频学习：单片机最小系统

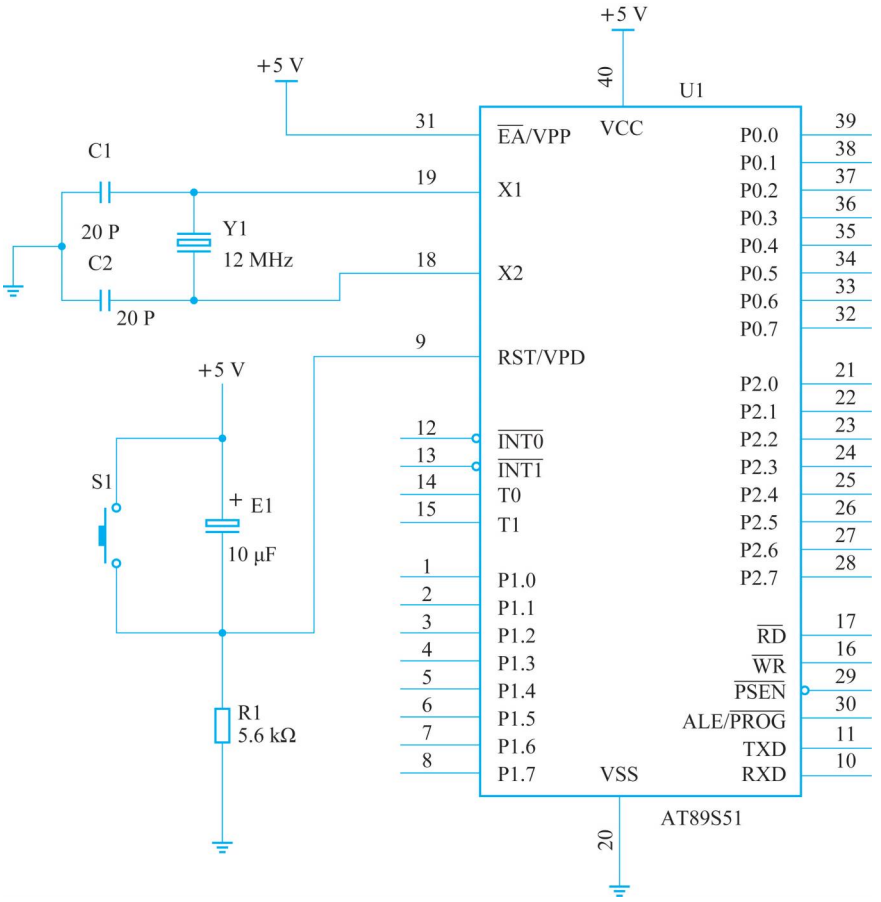


图 1-11　单片机最小系统电路图

想一想

如果单片机在运行时宕机，系统停止工作了，此时你还不知道，那你如何避免这种情况的发生呢？

二、单片机的初始状态

(一) 单片机的复位

单片机最小系统包含了复位电路。当单片机上电一瞬间，RST 引脚上会出现超过 2 个机器周期的高电平，单片机开始复位，进入初始状态，内部寄存器也回到初始状态，绝大

部分的内部寄存器初始状态都是 00H，这是 16 进制表示的数，表示一个字节，一个字节有 8 个位，一个位只有两种状态，0 和 1，00H 就表示 8 个位都是 0。也有几个寄存器初始状态不是 00H 的，如表 1-1 所示。

表 1-1　单片机复位后内部寄存器初始状态

特殊功能寄存器	初始状态	特殊功能寄存器	初始状态
ACC	00H	TCON	00H
B	00H	TH0	00H
PSW	00H	TL0	00H
SP	07H	TH1	00H
DPL	00H	TL1	00H
DPH	00H	SCON	00H
P0~P3	FFH	SBUF	不定
IP	＊＊＊00000B	PCON	0＊＊＊＊＊＊＊B
IE	0＊＊00000B	TMOD	00H

堆栈指针 SP，它指向单片机内部的 RAM，为了区分 RAM 中各个字节空间，将 RAM 的所有字节都编上一个号码，SP 的初始值为 07H，这个 07H 就是其中的一个 RAM 字节空间的编号，而堆栈是用来临时存储数据的，就像一个有底的柜子，你往里面放盘子，从底部开始放，第一个盘子放在底的上面，所以第一数据存放在 08H 这个存储单元里面，再放一个盘子，就存在 09H 这个单元。但拿盘子的时候，是从上面开始拿的，也就是先拿 09H 这个位置的盘子，所以堆栈的特点就是“先进后出”。

P0~P3 寄存器的初始值为 FFH，也就是 8 个位都是 1，意味着单片机的这 32 个引脚初始状态是高电平。因此，在设计电路时，一般都把电路设计成低电平驱动的。

SBUF 是串行通信数据缓冲器，它的初始状态是不定的，这一点也要注意。

另外，还有一个很重要的寄存器——PC(Program Counter)程序指针。PC 的初始值为 0000H，这说明 PC 是 16 位的寄存器，它可以由 0000H 变化到 FFFFH，也就是 16 个 0 变化到 16 个 1，一共 65536 种状态，也就是 64K，PC 是指向单片机的程序存储器 ROM 的，所以 ROM 的大小最大就是 64K，超过这个大小就没有它的编号了，也就识别不出来了。PC 的值是多少，就告诉单片机去 ROM 的哪个位置取指令。初始值为 0000H，首先 PC 的指令地址放到地址总线，然后按地址从存储器中取出指令，送给 CPU，这时 PC 的值自动 +1，CPU 执行完指令后，又开始取第二条指令，一直如此循环下去，这也是单片机的工作过程。

(二)单片机存储器结构

可以看到，PC 值就是 CPU 的指挥棒，它指哪，CPU 就打哪。所以，程序的流程控制也

就是 PC 值的控制。可见，单片机的运行实质上就是对存储器操作的过程，因此，我们需要了解存储器的结构。单片机的系统结构采用哈佛结构，也就是程序存储器和数据存储器分开，互相独立编址(编址也就是编号，以保证每个字节存储空间之间能区分开来)。程序存储器和数据存储器各有自己的操作模式(也就是寻址方式和控制方式)。物理上有 4 个分区：片内 ROM、片外 ROM、片内 RAM 和片外 RAM，如图 1-12 所示。

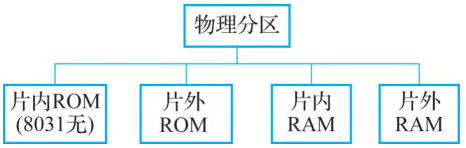


图 1-12　存储器物理分区图

在片内的不够用时，用片外的进行扩展，目前的单片机基本上不存在这个问题，所以我们就重点看片内的 ROM 和 RAM。编址时，片内 ROM 和片外 ROM 统一编址，如前文所说，ROM 的最大空间只有 64K，因为地址总线只有 16 位，类似于 PC 机的硬盘分区，以前有 FAT32 格式，就存放不了超过 4G 的单个文件，但 NTFS 格式就行，其实质就是地址总线(文件分配表)的长度不一样。

RAM 片内和片外是分别编址的，我们也只关注片内的 256 字节。这 256 个字节每个字节一个编号的话就是 256 个编号，只需要 8 位就能表达过来，也就是 00H—FFH 正好是 256 种状态，所以片内 RAM 的地址只需要 8 位就可以了。读者可能发现一个问题：51 单片机不是只有 128B 的 RAM 吗，这里怎么又出来了 256B？因为那 128B 是操作者可以用的 RAM，另外的 128B 被特殊功能寄存器 SFR 占用，下面我们来看 ROM 和 RAM 具体的编址情况。

1. 程序存储器

当 EA 引脚接高电平时，用的是单片机内部的 ROM，EA 为低电平时，用的是外部的 ROM，所以一般 EA 为高电平。只用内部的 4KB 空间时，地址编号是 0000H—0FFFH。ROM 中，这 5 个中断的入口地址需要注意：

外部中断 0 的入口地址：0003H

外部中断 1 的入口地址：0013H

定时器 0 中断的入口地址：000BH

定时器 1 中断的入口地址：001BH

串行口中断的入口地址：0023H

2. 内部的数据存储器 RAM

内部的数据存储器 RAM 地址编号为 00H—FFH，其中 00H—7FH 这 128 个单元是用户 RAM，80H—FFH 这 128 个单元是特殊功能寄存器 SFR。

需要特别注意的是可位寻址的区域，20H—2FH 这 16 个单元，每个单元有 8 位(即一个字节)，一共有 128 位，这 128 位是可以一个位一个位操作的，这样我们就可以在这些位置定义位变量，当然为了区分这 128 位，可以给每一位编号，20H 的第 0 位编号位 00H，一直到第七位 2FH 编号为 FFH。这个编号虽然与 RAM 的地址编号一样，但含义不同，读者了解就可以，因为 C 语言编译器会自己去区分。但如果用汇编语言的话，程序员

就必须非常清楚它们的含义。表 1-2 就是可位寻址 RAM 的位地址编号表。

表 1-2　可位寻址 RAM 的位地址编号表

BYTE	MSB	—	—	—	—	—	—	LSB
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	76	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00

特殊功能寄存器的地址分配表如表 1-3 所示。可以发现，有的特殊功能寄存器，如累加器 ACC，除了有字节地址，还有位地址，这表示 ACC 也是可以按位来操作的。所以，单片机中一共有 221 个可寻址的位，其中 93 个是特殊功能寄存器中的位。如果我们按字节地址来操作这些特殊功能寄存器，是肯定记不住的，好在可以用标识符(汇编语言编译器会识别标识符，认为标识符和字节地址是相等的)来表示这些寄存器，但如果是 C 语言编译器的话，就需要把标识符和字节地址画一个等号；否则，编译器无法识别，这个等号就是在头文件(reg51. h)中画的，这一点暂时了解即可。

表 1-3　特殊功能寄存器的地址分配表

标识符	名　称	位地址	字节地址
* ACC	累加器	E0H—E7H	0E0H
* B	B 寄存器	F0H—F7H	0F0H
* PSW	程序状态字	D0H—D7H	0D0H
SP	堆栈指针	—	81H
DPTR	数据指针		83H 和 82H
* P0	口 0	80H—87H	80H
* P1	口 1	90H—97H	90H
* P2	口 2	A0H—A7H	0A0H
* P3	口 3	B0H—B7H	0B0H
* IP	中断优先级寄存器	B8H—BDH	0B8H
* IE	中断允许寄存器	A8H—AFH	0A8H
TMOD	定时/计数器方式控制	—	89H
* TCON	定时/计数器控制	88H—8FH	88H
T2CON	定时/计数器 2 控制	C8H—CFH	0C8H
TH0	定时/计数器 0(高位字节)	—	8CH
TL0	定时/计数器 0(低位字节)	—	8AH
TH1	定时/计数器 1(高位字节)	—	8DH
TL1	定时/计数器 1(低位字节)	—	8BH
+TH2	定时/计数器 2(高位字节)	—	0CDH
+TL2	定时/计数器 2(低位字节)	—	0CCH
+RLDH	定时/计数器 2	自动再装载(高位)	0CBH
+RLDL	定时/计数器 2	自动再装载(低位)	0CAH
* SCON	串行通信控制	98H—9FH	98H
SBUF	串行数据缓冲器	—	99H
PCON	电源控制	—	87H

任务三

设计彩灯控制电路的硬件系统

一、单片机的 I/O 接口

单片机有 4 个 8 位的并行 I/O 接口，分别叫作 P0、P1、P2、P3 口。它们的工作原理比较复杂，读者可以先简单了解，随着学习的深入，逐步理解。

(一) P0 口的结构及工作原理

1. P0 口的结构

P0 口由锁存器、输入缓冲器、切换开关、一个与非门、一个与门及场效应管驱动电路构成。标号为 P0. X 引脚的图标，可以是 P0.0 到 P0.7 的任何一位，即在 P0 口由 8 个与本图相同的电路。P0 口工作原理如图 1-13 所示。

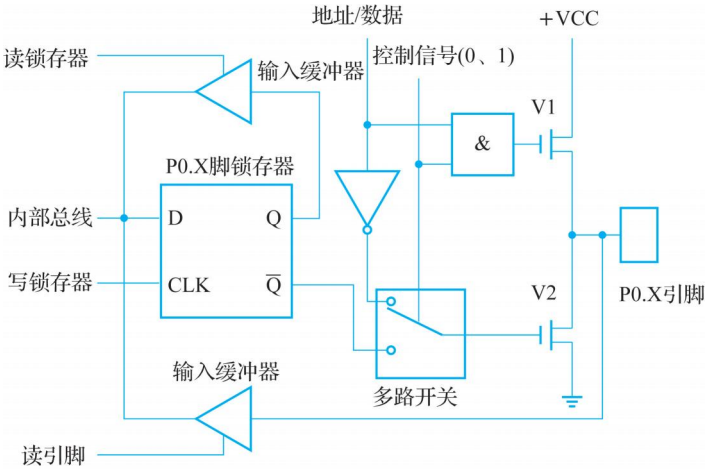


图 1-13　P0 口工作原理图

1) 输入缓冲器

在 P0 口中，有两个三态的缓冲器，三态门的输出端有 3 个状态：高电平、低电平和高阻状态(或称为禁止状态)。图 1-13 上方的输入缓冲器是读锁存器的缓冲器，要读取 D 锁存器输出端 Q 的数据，那就得使这个缓冲器的三态控制端(图中标号为“读锁存器”端)有效。下方的输入缓冲器是读引脚的，要读取 P0. X 引脚上的数据，也要使标号为“读引脚”的这个三态缓冲器的控制端有效，引脚上的数据才会传输到我们单片机的

内部数据总线上。

2）D 锁存器

对于 D 锁存器，当 D 输入端有一个输入信号，如果这时控制端 CP 没有信号（也就是时序脉冲没有到来），这时输入端 D 的数据是无法传输到输出端 Q 及反向输出端 Q 非的。如果时序控制端 CP 的时序脉冲到了，D 端输入的数据就会传输到 Q 及 Q 非端。数据传送过来后，当 CP 时序控制端的时序信号消失了，输出端还会保持着上次输入端 D 的数据（即把上次的数据锁存起来了）。如果下一个时序控制脉冲信号来了，D 端的数据才再次传送到 Q 端，从而改变 Q 端的状态。

3）多路开关

在 51 单片机中，当内部的存储器够用（也就是不需要外扩展存储器时，这里讲的存储器包括数据存储器及程序存储器）时，P0 口可以作为通用的输入/输出端口（I/O）使用，对于 8031（内部没有 ROM）的单片机或者编写的程序超过了单片机内部的存储器容量，需要外扩存储器时，P0 口就作为“地址/数据”总线使用。多路选择开关就是用于选择作为普通 I/O 接口使用还是作为“数据/地址”总线使用的选择开关。

4）输出驱动部分

从图 1-13 我们看出 P0 口的输出是由两个 MOS 管组成的推拉式结构。也就是说，这两个 MOS 管同一时刻只能导通一个，当 V1 导通时，V2 就截止；反之，亦然。

2. P0 口作为 I/O 端口使用工作原理

P0 口作为 I/O 端口使用时，多路开关的控制信号为 0（低电平），P0 口内部数据总线向引脚输出示意图如图 1-14 所示。

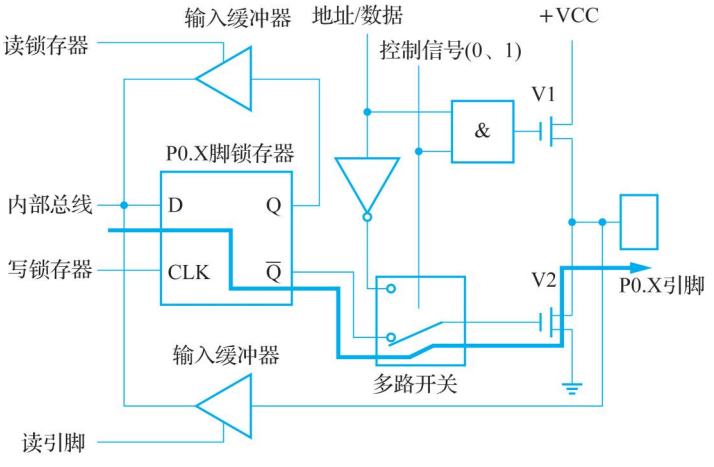


图 1-14　P0 口内部数据总线向引脚输出示意图

图 1-14 中多路开关的控制信号同时与门的一个输入端相接，与门的逻辑特点是“全 1 出 1，有 0 出 0”。当控制信号是“0”时，与门输出的也是一个“0”（低电平），V1 管就截止，此时多路开关与锁存器的 Q 非端相接（即 P0 口作为 I/O 端口线使用）。

1）P0 向引脚输出

P0 口用作 I/O 接口线，其由数据总线向引脚输出（即输出状态 Output）的工作过程：当写锁存器信号 CP 有效，数据总线的信号→锁存器的输入端 D→锁存器的反向输出 Q 非端→多路开关→V2 管的栅极→V2 管的漏极到输出端 P0. X（X 可以是 0～7）。当多路开关的控制信号为低电平“0”时，与门输出为低电平，V1 管是截止的。所以，作为输出口时，P0 是漏极开路输出，类似于 OC 门。当驱动上接电流负载时，需要外接上拉电阻，上拉电阻一般选 10k。图 1-14 中的粗线箭头就是由内部数据总线向 P0 口输出数据的信号流向。

2）P0 读引脚

P0 口用作 I/O 接口线，由引脚向内部数据总线输入（即输入状态 Input）的工作过程：数据输入时（读 P0 口）又有两种情况。

（1）读引脚。读芯片引脚上的数据，读引脚数时，读引脚缓冲器打开（即三态缓冲器的控制端要有效），通过内部数据总线输入，如图 1-15 所示（粗线箭头）。

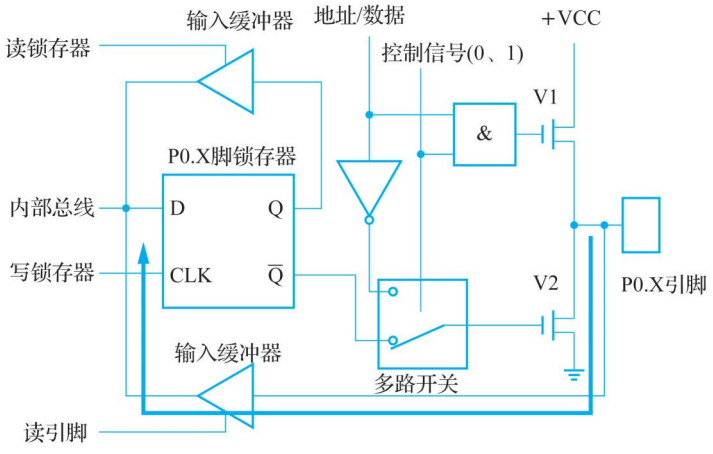


图 1-15　P0 口读引脚示意图

（2）P0 读锁存器。读锁存器时，通过打开读锁存器三态缓冲器读取锁存器输出端 Q 的状态，如图 1-16 所示（粗线箭头）。

在输入状态下，从锁存器和从引脚上读来的信号一般是一致的，但也有不一致的时候。为此，8051 单片机在对端口 P0—P3 的输入操作上约定：凡属于读—修改—写方式的指令，从锁存器读入信号，其他指令则从端口引脚线上读入信号，一般情况，都认为是读引脚。但由于 P0 口在做普通 I/O 接口时，不存在高阻抗状态，因此，也叫作准双向口。为保证引脚信号正确输入，先置 P0 口为“1”再读，所以，单片机复位之后，P0 口的初始状态是“1”，这就能保证读取准确的数据。而 P0 口做地址/数据复用功能时，它又是一个双向口。

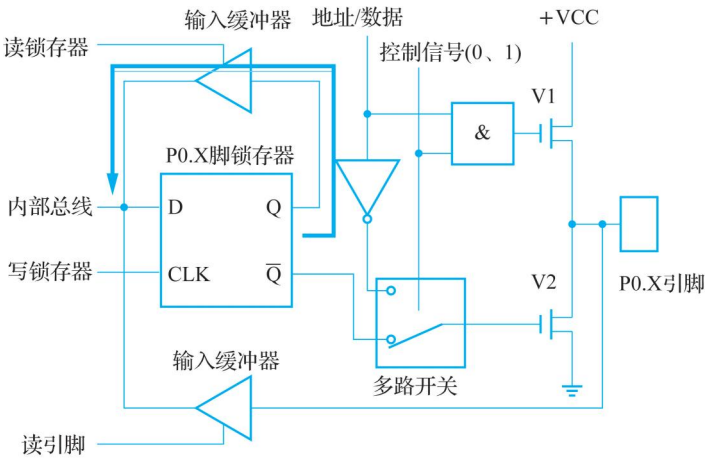


图 1-16　P0 口读锁存器时的流程图

(二)P1 口的结构及工作原理

P1 口的结构最简单，用途也单一，仅作为数据输入/输出端口使用，如图 1-17 所示。

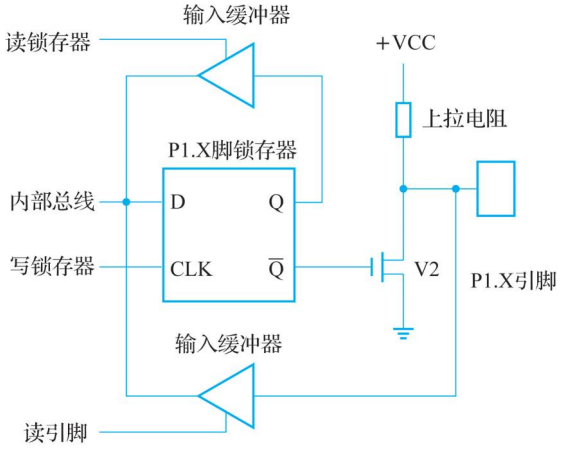


图 1-17　P1 口工作原理图

由图 1-17 可知，P1 端口与 P0 端口的主要差别在于，P1 端口用内部上拉电阻 R 代替了 P0 端口的场效应管 T1，并且输出的信息仅来自内部总线。由内部总线输出的数据经锁存器反相和场效应管反相后，锁存在端口线上，所以，P1 端口是具有输出锁存的静态口。要正确地从引脚上读入外部信息，在作引脚读入前，必须先对该端口写入“1”，也就是说 P1 是准双向 I/O 接口。8051 单片机的 P1、P2、P3 都是准双向口。单片机复位后，各个端口已自动地被写入了“1”，此时，可直接作输入操作。如果在应用端口的过程中，已向 P1—P3 端口线输出过“0”，则再要输入时，必须先写“1”后再读引脚，才能得到正确的信息。

(三) P2 口的结构及工作原理

P2 口可以作为 I/O 接口使用，也可以作为地址总线使用，如图 1-18 所示。

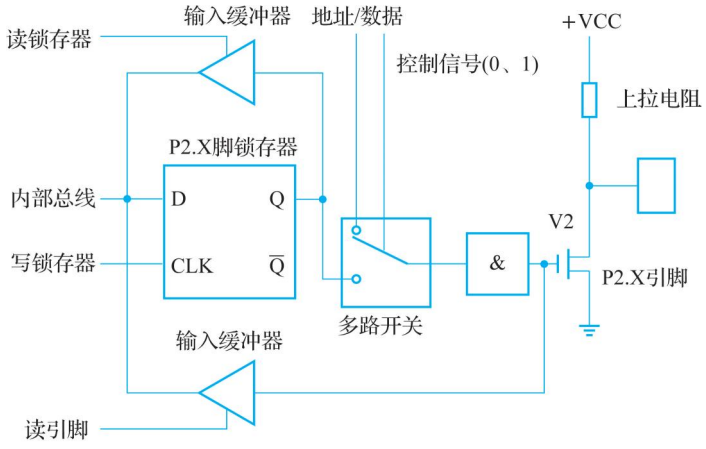


图 1-18　P2 口工作原理图

1. 作为 I/O 端口使用时的工作过程

当没有外部程序存储器或虽然有外部数据存储器，但容量不大于 56B，即不需要高 8 位地址时(在这种情况下，不能通过数据地址寄存器 DPTR 读写外部数据存储器)，P2 口可以作为 I/O 接口使用。这时，控制信号为“0”，多路开关转向锁存器同相输出端 Q，输出信号经内部总线→锁存器同相输出端 Q→反相器→V2 管栅极→V2 管漏极输出。由于 V2 漏极带有上拉电阻，可以提供一定的上拉电流，负载能力约为 8 个 TTL 与非门；作为输出口前，同样需要向锁存器写入“1”，使反相器输出低电平，V2 管截止，即引脚悬空时为高电平，防止引脚被钳位在低电平。读引脚有效后，输入信息经读引脚三态门电路到内部数据总线。

2. 作为地址总线使用时的工作过程

P2 口作为地址总线时，控制信号为“1”，多路开关转向地址线(即向上接通)，地址信息经反相器→V2 管栅极→V2 管漏极输出。由于 P2 口输出高 8 位地址，与 P0 口不同，无需分时使用，因此 P2 口上的地址信息(程序存储器上的 A15～A8)或数据地址寄存器高 8 位 DPH 保存时间长，无须锁存。

(四) P3 口的结构及工作原理

P3 口是一个多功能口，除了可以作为 I/O 口外，还具有第二功能。P3 端口的结构如图 1-19 所示。

可见，P3 端口和 P1 端口的结构相似，区别仅在于 P3 端口的各端口线有两种功能选择。当处于第一功能时，第二输出功能线为“1”。此时，内部总线信号经锁存器和场效应管输入/输出，其作用与 P1 端口作用相同，也是静态准双向 I/O 端口。当处于第二功能时，锁存器输出“1”，通过第二输出功能线输出特定的内含信号；在输入方面，即可以通

过缓冲器读入引脚信号，还可以通过替代输入功能读入片内的特定第二功能信号。由于输出信号锁存并且有双重功能，故 P3 端口为静态双功能端口。

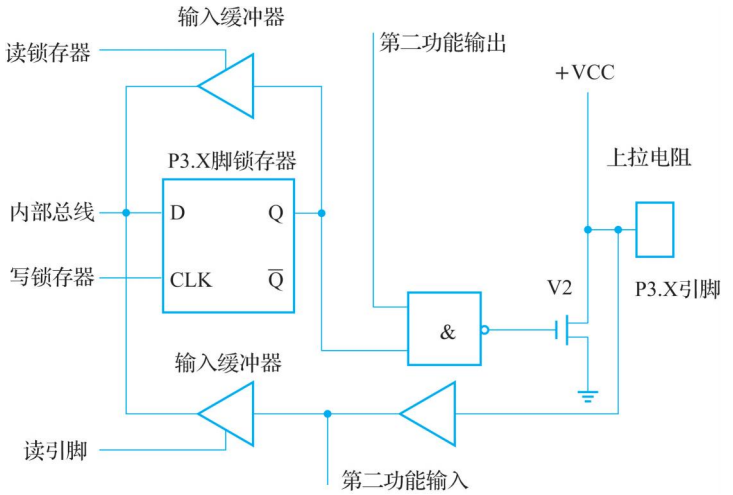


图 1-19　P3 口工作原理图

视频学习：单片机的IO接口

在应用中，如不设定 P3 端口各位的第二功能，则 P3 端口线自动处于第一功能状态，也就是静态 I/O 端口的工作状态。在多数情况下，根据应用的需要，会把几条端口线设置为第二功能，而另外几条端口线处于第一功能运行状态。这时，不宜对 P3 端口作字节操作，需采用按位操作的形式。

单片机的 I/O 接口结构虽然复杂，但“合抱之木，生于毫末；九层之台，起于累土”“不积跬步，无以至千里；不积小流，无以成江海”，建议读者从最基本应用学起，循序渐进，逐步弄懂 I/O 接口的工作原理。在由浅入深的学习过程中，养成精益求精和艰苦奋斗的工匠精神。

二、基于 Proteus 设计彩灯电路

Proteus 是英国著名的 EDA 工具(仿真软件)。从原理图布图、代码调试到单片机与外围电路协同仿真，再到一键切换到 PCB 设计，Proteus 真正实现了从概念到产品的完整设计。下面主要介绍该软件用于单片机应用电路的设计和仿真。

很多人初学单片机时，拿着一块实验板发呆，电路不懂，程序也不懂，只能慢慢琢磨，等他们琢磨明白了，实验板也基本报废了。Proteus 正好可以解决这个问题。它可以在原理图上仿真，不用做出 PCB 板，可以说是初学者的“神器”。下面我们来认识一下 Proteus。

(一) Proteus 简介

Proteus 启动界面如图 1-20 所示。

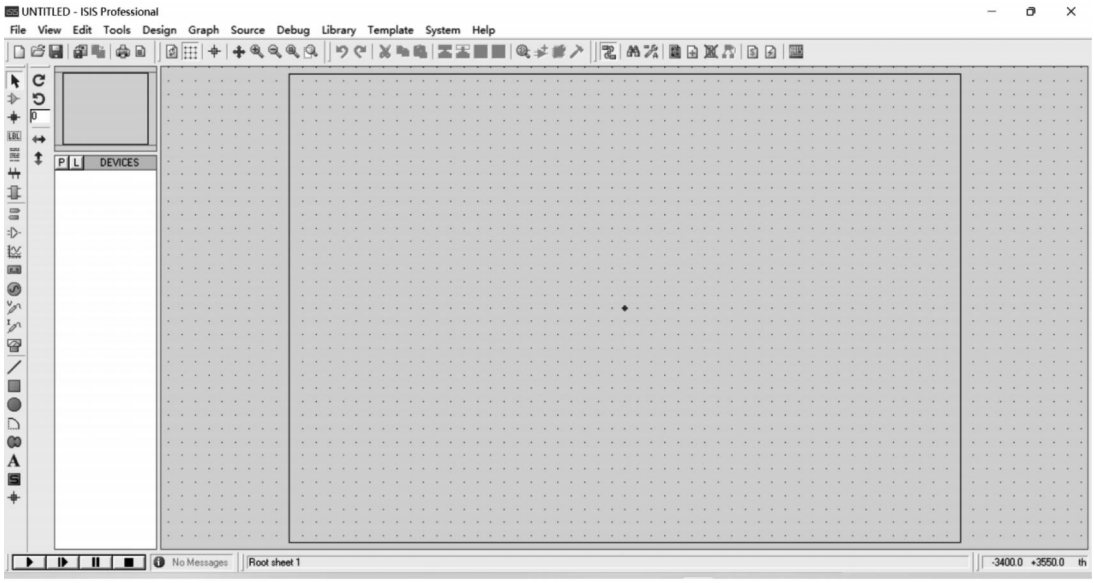


图 1-20　Proteus 启动界面

我们单击图 1-20 的【P】后，弹出选择元器件的界面，如图 1-21 所示。

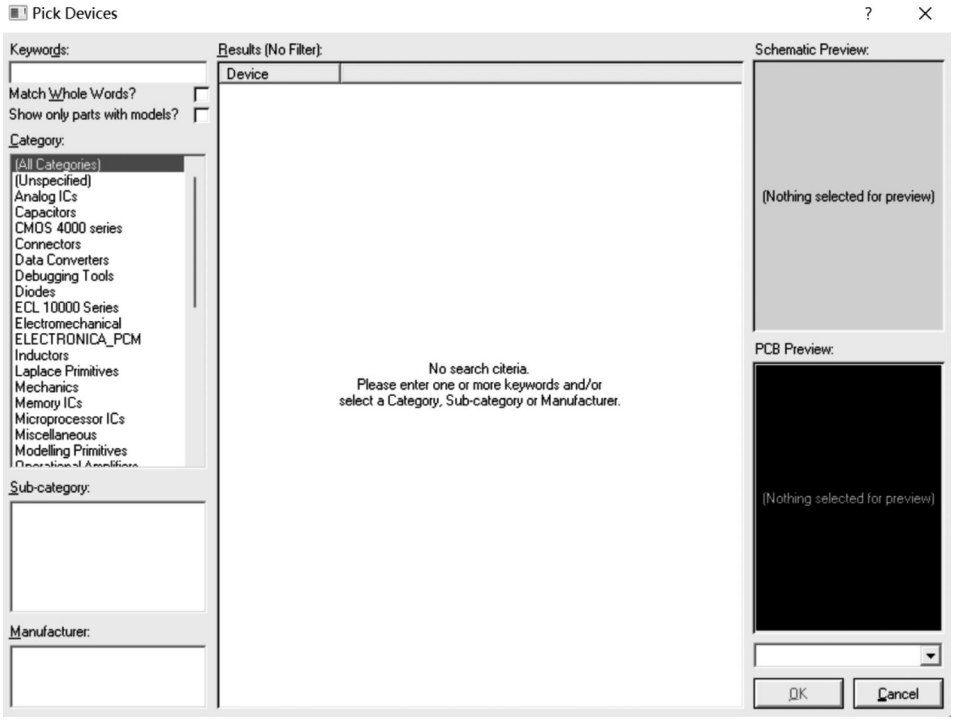


图 1-21　选择元器件界面

在【Keyword】处输入“at89c51”，出现“AT89C51”，双击它，左侧空白框中出现

AT89C51，左键单击它，它上方框中显示出它的原理图。把鼠标移到右侧框中，鼠标变成铅笔形状，单击左键，框中出现一个 AT89C51 原理图的轮廓图，如图 1-22 所示，可以移动。

鼠标移到合适的位置后，按下鼠标左键，单片机的原理图就放置好了。同样的方法可以选择其他元件。另外，点击左侧的终端模式图标，可以看到，ground 是电源地极，power 是电源正极。

这是 Proteus 的基本操作，还有其他的操作，如移动元器件、放大元器件、修改元器件、连接元器件等都可以通过鼠标左键、滚轮和右键来进行操作，具体可以参考 Proteus 的操作手册。

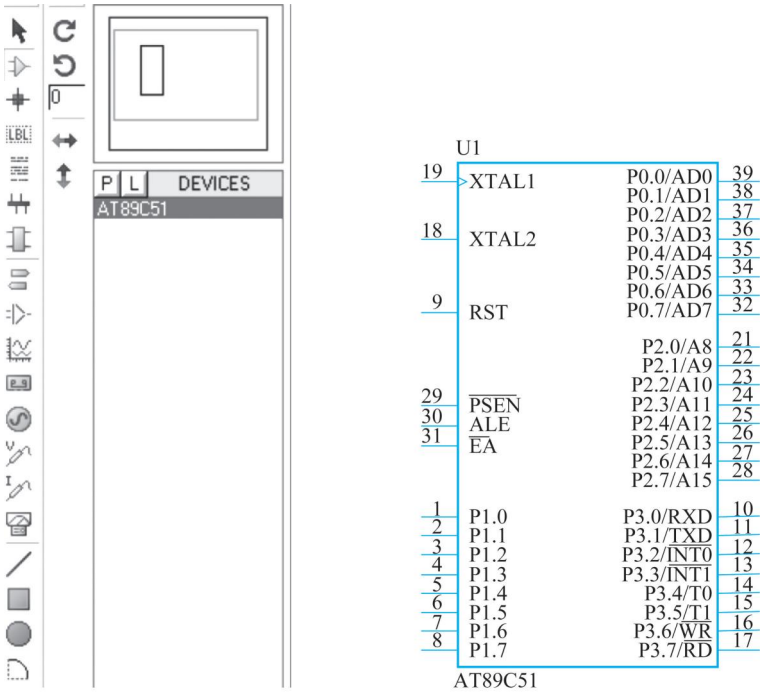


图 1-22　元器件放置图

(二) 彩灯电路的设计

1. 单片机如何控制 LED 灯

彩灯可以看成是由各种各样的 LED 灯组成，那用单片机如何控制 LED 灯呢?

典型的单片机控制 LED 灯的原理图，如图 1-23 所示。当单片机的 I/O 接口 P1.0 = 1 时，灯灭；P1.0 = 0 时，灯亮。为了形成彩灯效果，选择 Proteus 中的 4 种颜色的 LED 灯，选择 AT89C51 单片机和电阻 RES，我们可以通过刚才添加元器件的方式把这些元件添加进来，然后把这些元器件放到右边的图纸上，用导线把他们连接起来，再连接好电源。

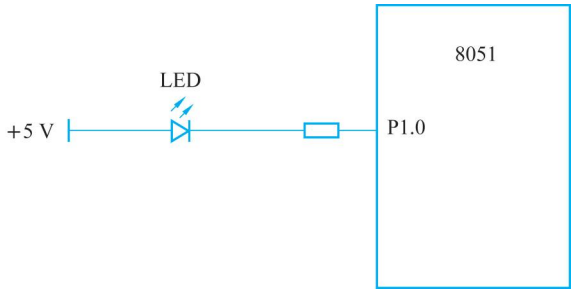


图 1-23　单片机控制 LED 灯原理图

2. Proteus 设计单片机控制的彩灯电路

基于单片机控制单个 LED 的原理，在 Proteus 中，可以轻松的设计单片机控制的彩灯电路，如图 1-24 所示。

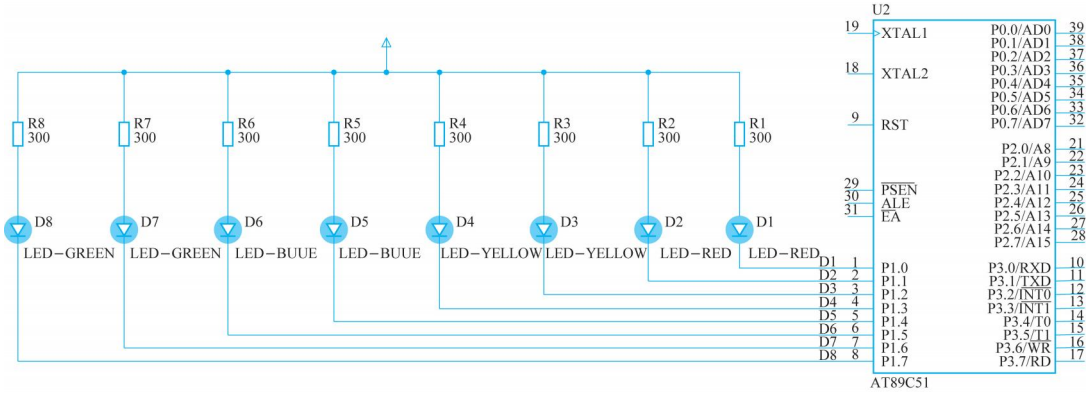


图 1-24　在 Proteus 中，单片机控制 LED 彩灯仿真图

为了形成彩灯的各种效果，这里 P1 口的 8 个位分别接了 1 个 LED 灯。这些位只要状态为“0”，对应的灯就亮；状态为“1”，对应的灯就灭。

在设计电路时需要注意几点：

视频学习：基于 Proteus设计彩灯电路

(1)电路中的限流电阻的大小：一般限流电阻的大小要根据 LED 灯的正向导通压降和工作电流确定，如正向导通压降为 2V，工作电流为 10mA，则限流电阻选择(5V-2V)/0.01A=300Ω。软件中电阻默认是 10k，不修改电阻的阻值，LED 灯是不会亮的。

(2)电路中并没有设计单片机的最小系统电路。因为仿真系统已经默认连接好了，双击单片机你会看到默认的晶振是 12MHz。

(3)元器件属性可以调整。双击任意一个元器件，都可以修改它的属性值。右键单击元器件可以进行移动、剪切、旋转等编辑工作。

(三)Keil 软件的使用步骤

(1)新建工程前，建议先新建一个文件夹用来存放工程。

(2)打开 Keil 软件后，首先需要创建一个工程，点击【文件】—【新建】—【新建工程】，给工程起一个名字，在输入工程名后点【保存】。

(3)随后，出现单片机型号选择框，可以选用 STC89C52，没有 stc 的选择 Atmel 的就可以，因为它们都是 51 内核，指令都是兼容的。因此，选择 AT89C51 或 AT89C52 都可以，选好后点击【确定】。

(4)弹出一个选择对话框，这里选择【否】，不需要建立启动文件。

(5)工程新建好了，但是里面并没有任何文件，现在可以给工程新建一个 C 源程序了，点击【新建一个新文件】后点左上角的保存按钮，输入文件名，记住扩展名必须为“.c”。

(6)此时，新建的 c 文件并没有进入工程中，需要手动添加。点击【目标 1(target1)】，右键【源代码组 1(sourse group1)】，点击【添加文件到组“源代码组 1”】，选择刚才的 c 文件，点击【add】后点【close】。

(7)现在【源代码组 1】中就已经添加了 c 文件，可以进行编程了。程序编好后，进行编译，如果没有提示任何错误信息，表明程序没有问题。但由于要将程序烧录到单片机中，需要用到 .hex 文件，需要 Keil 软件在编译时生成一个 .hex 文件，因此，单击【目标 1(target1)】的右边的第一个按钮，选择【输出(output)】，勾选【产生 HEX 文件】，再次编译，就可以生成 .hex 文件了。

关于 Keil 软件的使用，读者可以参考 Keil 软件的使用说明书，也可以参考视频学习，这里不再赘述。

任务四

设计彩灯控制电路的软件系统

一、C51 的基础知识

(一)C 语言的特点

(1)语言简洁、紧凑、使用灵活;
(2)运算符丰富;
(3)具有数据类型构造能力;
(4)具有很强的流程控制结构;
(5)语言生成的代码质量高;
(6)可移植性较好。

(二)C51 中基本数据类型

1. 变量和常量

1)变量

变量是值可以改变的量。变量，实际就是可以存放数据的地方，并且存放的数据是可以更换的，那这个数据到底存在哪里？能放多大的数据？这是需要知道的。所以，变量应该有一个名字，并且占据了存储器的一段内存空间，这个名字可以是存储器的物理地址(使用时不方便)，还可以是一个有意思的名字(便于记忆和程序设计，多数采用此种方式)；占据的存储器空间，可以由编创者来分配(很少用)，或者由编译器去分配(自动分配)。如果用汇编语言的话，定义一个变量就必须知道这个变量的物理地址和内存空间，而在 C51 中，只要给变量起一个名字，并且告诉编译器，是什么类型的变量就可以了，剩下的就是编创者对变量的数据操作了。

2)常量

在程序运行过程中，值不能改变的量为常量。比如 10(十进制)，0x45(16 进制)，00001111B(二进制)等，这些都属于直接常量，一眼就能看出来。

还有一类符号常量，是在 C51 中经常用到的。

如：宏定义#define　control　100

这就定义了一个符号常量 control，后面程序中，只要使用到 control，就代表 100 了，那如果调试程序时，想把 100 换成 80，那只需要在最开始的宏定义中，把值修改为 80 即可，而不需要在程序中逐条语句去修改了。可见，在程序设计时，使用符号常量有很多优

点。比如含义清楚。上例中的 control 就代表控制量。可见，符号变量可以做到“见名知意”，一改全改或全局修改时或调试程序时，会经常用到。

2. 数据类型

数据是具有一定格式的数字或者数值，是计算机操作的对象。使用任何语言、任何算法进行程序设计，最终实现的功能实际上就是数据的流动。数据的不同格式就叫作数据类型。数据按一定的数据类型进行的排列、组合及架构称作数据结构。在单片机控制系统的程序设计中，用到的数据类型是很有限的，C51 编译器中常见的数据类型见表 1-4。

表 1-4　C51 编译器中常见的数据类型表

数据类型	长度(位/bit)	长度(字节/byte)	取值范围
bit	1	—	只有 0 或 1
unsigned char	8	1	0~255
signed char	8	1	-128~127
unsigned int	16	2	0~65535
signed int	16	2	-32768~32767
unsigned long	32	4	0~4294967295
signed long	32	4	-2147483648~2147483647
float	32	4	±1.76E-38~±3.40E+38

3. C51 中常用的数据类型

1)位型变量

位型变量实际上是定义一个二进制位，用来存放数据，它的值只能是“0”或“1”，定义形式如下：bit 变量名；

如：

```
bit   flag;                          //定义了一个位变量 flag
```

flag 的取值可以是 0 或 1，默认初始值为 0。这个 flag 变量实际上是占用单片机 RAM 中可位寻址空间中的某一位，由于可位寻址空间中一共有 128 个位，所以在一个程序中，最多只能定义 128 个位变量。

2)字符型变量

字符型变量用来存放一个字节的数据，其定义形式为：

```
char   变量名                        //定义有符号的字符型变量
unsigned char   变量名;              //定义无符号的字符型变量
```

定义一个字符型变量时，编译器会为其分配一个字节的存储空间。这个字符型变量是不是只能存放字符？比如“a”“A”。其实，任何变量里面存放的都是 0 和 1，只是表示的含义不一样。比如，字符型变量存放的也是 0 和 1，它在保存“a”或“A”等这些字符时，是以

字符的 ASCII 码值保存在存储器里面的。所以，实际给字符型变量赋值时，也常赋给它整型的数值。如：

```
char  b=89;                    //定义一个字符型变量,并赋值89
```

3）整型变量

整型变量的基本类型是 int，在 C51 里面有整型和长整型之分，比如：

```
int  sum;                      //定义一个有符号整型变量
unsigned int  add1,add2;       //定义无符号整型变量 add1,add2
long  int  sum;                //定义一个有符号长整型变量
unsigned long int  add1,add2;  //定义无符号长整型变量 add1,add2
```

4）实型变量

实数又称为浮点数，包含单精度、双进度和长双进度，但 Keil C 中只支持单精度，也就是 float 型。其表示形式一般有两种：

十进制小数形式。由数字和小数点组成，如 1.5，17.9 等。

制数形式。如 7e2 或 7E2，表示 7 * 102。

实型变量（float）占用 4B，能提供的有效数字是有限的，有效位之后都会被舍弃，所以在计算时会带来误差，这在数据处理时要注意。另外，实型数据会给单片机的计算带来很大的工作量，增加程序长度，所以，在单片机数据处理中，能用整型数据处理的，尽量不用实型变量。

4. C51 中数据的存储类型

8051 单片机存储区有内部数据存储区、外部数据存储区以及程序存储区。

内部数据存储区是可读写的，一般只有 128B（52 系列有 256B，但只有低 128B 可直接寻址，高 128B 只能间接寻址），由于内部数据存储器 128B 中有 16B 是可位寻址的，所以，内部数据存储器分成了 3 个不同的存储类型：data、idata 和 bdata。

外部数据存储区也是可读写的，需要借助数据指针 DPTR 间接访问外部数据存储区。C51 中有两种不同的存储类型 xdata 和 pdata 访问外部数据存储区的数据。

程序存储区的数据只能读，不能写，要访问程序存储区的数据，C51 中采用 code 类型来访问。

因此，在程序设计时定义任何变量时，必须保证每个变量可以明确地分配到指定地存储空间。但对内部数据存储区的访问，比对外部数据存储区的访问要高效得多，所以，数据处理时尽量采用内部数据存储区，尤其是经常访问的变量定义到内部的数据存储区。各存储器的对应关系如表 1-5 所示。

表 1-5　各存储器的对应关系表

存储区	含　　义
data	片内 RAM 的低 128B
bdata	片内 RAM 的位寻址区
idata	片内 RAM 的高 128B
xdata	片外数据存储区
pdata	片外存储区的 256B
code	ROM

(三) C51 中常用的运算符

1. 算术运算符

C51 中常用的算术运算符如表 1-6 所示。

表 1-6　C51 中常用的算术运算符

算术运算符	含义	实　　例
+	加	4+3=7
-	减	5-3=2
*	乘	4*3=12
/	除	8/3=2
%	求余	11/10=1
++	自增 1	a 原为 1，a++，a 为 2
--	自减 1	a 原为 2，a--，a 为 1

2. 关系运算符

C51 中常用的关系运算符如表 1-7 所示。

表 1-7　C51 中常用的关系运算符

关系运算符	含义	关系运算符一般用来构成关系表达式
<	小于	a<b
<=	小于等于	a+b<=c
>	大于	b>a
>=	大于等于	a+b>=c
==	等于	a==b
!=	不等于	a!=b

3. 逻辑运算符

C51 中常用的逻辑运算符如表 1-8 所示。

表 1-8　C51 中常用的逻辑运算符

逻辑运算符	含义	逻辑运算符一般构成逻辑表达式
&&	逻辑与	a&&b
||	逻辑或	a||b
!	逻辑非	!a

4. 位操作

C51 中常用的位操作如表 1-9 所示，以下操作用 a=0x45，b=0x79 举例。

表 1-9　C51 中常用的位操作

位操作	含义	举　　例
&	按位与	a&b=0x40
|	按位或	a|b=0x7d
^	按位异或	a^b=0x3d
~	按位取反	~a=0xba
<<	位左移	a=a<<2，运算后 a=0x14(低位补零)
>>	位右移	a=a>>2，预算后 a=0x11(高位补零)

二、C51 中常用的控制语句

C51 中的语句与通用 C 语言基本一致，这里简要介绍常用的流程控制语句，包括分支语句和循环语句两种。

(一) C51 中判断语句 if 和 switch 的使用

1. if 语句

if 语句构成分支结构，根据给定的条件进行判断，决定执行某个分支程序。它一般有四种格式。

1) if(表达式)语句(图 1-25)

如果表达式的值为真(逻辑 1)，则执行其后的语句，否则不执行该语句，继续往下执行，这一点一定要注意，程序不会停止，而是继续往下执行。表达式一般是关系表达式或者多个关系表达式的逻辑运算，这要用到关系运算符大于“>”、大于等于“>=”、小于“<”、小于等于“<=”、等于

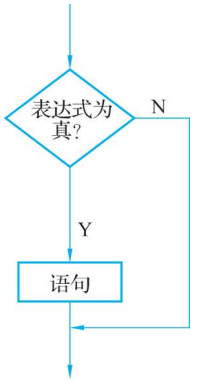


图 1-25　if 语句结构图

“==”、不等于“!=”和逻辑运算符逻辑与“&&”、逻辑或“||”、逻辑非“!”。

如：2>1，这个表达式的值就为真(逻辑 1)，2<1 这个表达式的值就为假(逻辑 0)，2>1&&2<1 的值为假，2>1||2<1 的值为真，!(2<1)的值为真。

注意：

一个等号“=”在 C 语言里面是赋值语句，有时候由于输入错误，本来是判断一个变量是不是等于某一个具体的值，这里应该是双等号“==”，但写成了等号“=”，条件到底是真还是假呢？例如把 if(a==1)写成了 if(a=1)，这里表达式 a=1，相当于把 1 赋值给 a，所以 a 总为 1，则认为表达式是真，并且一直为真。如果 a=0，则一直为假。

2)if-else 语句(图 1-26)

如果表达式的值为真，则执行语句 1，否则执行语句 2。例如，如果光线暗了，马路边的光控灯就打开；否则，马路边的光控灯就熄灭。语句 1 和语句 2 肯定要执行、且只会执行一个。

3)if-else—if 多分支语句(图 1-27)

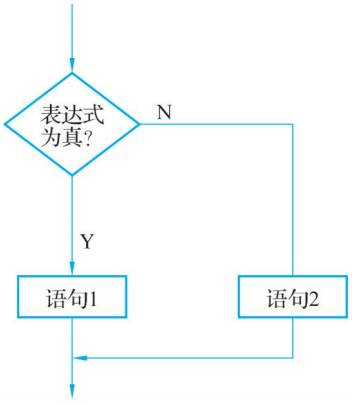


图 1-26　if-else 结构图

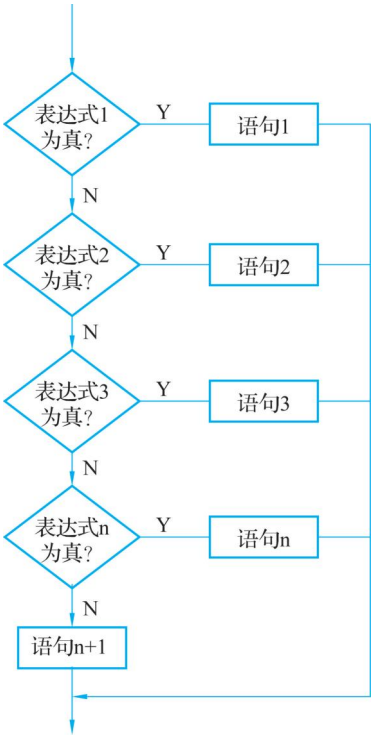


图 1-27　if-else—if 多分支结构图

If(表达式 1)语句 1；else if(表达式 2)语句 2；else if(表达式 3)语句 3；……else if(表达式 n)语句 n；else 语句 n+1。这种格式适合多种情形，例如：用于多种状态的彩灯控制，或者家用的有多种状态的遥控灯，都可以采用这种编程方式去实现。

4)If 语句的嵌套

当 if 语句中的执行语句又是 if 语句时，则构成了 if 语句的嵌套。

```
if(表达式 1)
    {if(表达式 2)语句 1；else 语句 2;}
else
    {if(表达式 3)语句 3；else 语句 4;}
```

这里用大括号“{}”把 if 要执行的内容括在一起，这一点读者一定要注意，尤其是开始练习编程，养成良好的编程习惯，增加程序的可读性，能够极大地减少调试工作量。

2. switch 语句

if 语句虽然可以实现多分支的功能，但程序会比较复杂，有没有更加简单的方法实现多分支控制呢？答案是肯定的，这就是 switch 语句。

1)switch 语句的应用格式

```
switch(表达式)
{ case 常量 1：语句 1；break;
  case 常量 2：语句 2；break;
  …
  case 常量 n：语句 n；break;
  default：语句 n+1;}
```

首先，计算表达式的值，逐个与其后的常量值进行比较，当表达式的值与某个常量值相等时，即执行其后的语句，执行到 break 命令时，跳出 switch，结束 switch 语句的执行。如表达式的值与所有 case 后的常量表达式均不相同时，则执行 default 后的语句。

2)switch 语句应用注意

(1)在 case 后的各常量表达式的值不能相同，否则会出现错误。

(2)在 case 后，允许有多个语句，可不用“{ }”括起来，因为每一个 case 后面都必须带一个 break 语句，也只有遇到 break 语句，才跳出 switch 语句；否则，就继续往下执行。

(3)各 case 和 default 子句的先后顺序可以变动，而不会影响程序执行结果。当然，为了调试程序方便，一般按大小顺序排列。

(4)default 语句可以省略不用，但如果所有的 case 后面的常量都与 switch 表达式的值不一样，则任何程序都不执行，如果程序必须要执行一种运算，可以把这个运算放到 default 后面。这样，前面的都不相等时，保证能执行默认的程序。可以看到，switch 语句用来编写多分支的程序时很方便，用在多状态彩灯控制中就非常合适了。

(二)C51 中循环控制语句 while 和 for 的使用

循环结构是程序中一种很重要的结构。其特点是，在给定条件成立时，反复执行某程序段，直到条件不成立为止。给定的条件称为循环条件，反复执行的程序段称为循环体。C 语言提供了多种循环语句，可以组成各种不同形式的循环结构。最常用的是 while 和 for 循环控制语句。

1. while 语句

While 语句的基本格式是：While(表达式){语句}。

执行这条语句时，首先计算表达式的值，当值为真时，执行循环语句；再回去判断表达式是否为真，如果为真，继续执行循环语句；直到表达式为假，跳出循环，执行后面的语句。

由于单片机 CPU 的运行速度很快，而它外部的一些设备的运行速度或者反应速度都没有 CPU 快。因此，有时候 CPU 要等待外部设备响应，我们常常让单片机在 CPU 等待的这段时间去执行一些没有意义的操作，延时函数正是这样的模块。单片机只要执行语句就需要消耗时间，晶振快，消耗的时间短，晶振慢，消耗的时间就长。通常采用循环执行空语句的方式达到一定时间的延时效果。

例如：如果单片机使用 12MHz 的晶体振荡器，延时 1ms 需要循环执行空语句、变量增加 1 和判断语句大概 121 次，因此，可以这样来设计程序。

```
void delay_1ms()                    // 函数名称 delay_1ms
{                                   // 函数开始
    unsigned char i=0;              // 定义无符号字符型变量 i 并赋初值0
    while(i<=121)                   // 循环语句 while
    {                               // 循环体开始
        ;                           // 空语句
            i++;                    // 循环变量增加
    }                               // 循环体结束
}                                   // 函数结束
```

delay_1ms 是函数的名称，一般来说，我们定义变量和函数的名称时都会尽量用一些标识符来表示，如这里的 delay_1ms，函数以左大括号开始，以右大括号结束。在函数里面，首先定义了一个无符号字符型变量，并赋初始值 0。当然，如果不赋初始值，它默认也是 0，变量 i 占据了一个字节的空间，由于是没有符号的，因此 i 的取值范围是 0~255；然后利用 while 循环，实现“循环执行空语句、变量增加 1 和判断语句 121 次”的功能，这样单片机只要执行一次 delay_1ms 函数就大约消耗 1ms 的时间。

这就是 while 语句的基本用法。在应用时需要注意以下几点：

(1)while 语句中的表达式一般是关系表达式或逻辑表达式，只要表达式的值为逻辑真(非 0)即可继续循环。如果写成 while(1){语句;}则表示无限循环，也就是死循环，只要执行这条语句，程序就出不去了。在学习 C 语言时，老师经常说：“不能出现死循环。”但是单片机的程序设计中却会用到死循环。那什么时候用？为什么用？读者可以先思考一下，后面的章节中再解释。

(2)循环体如包含一个以上的语句，则必须用“{ }”括起来，以增加程序的可读性。

(3)如果循环次数有限，就必须在循环体内部修改循环条件，保证循环能够退出。否则循环退不出，就成死循环了。如：while(i<100){;; i++;}这里的 i++就是起到这个作用。

(4)另外还有 do{循环体}while(表达式)的形式。这个循环与 while 循环的不同在于：

它先执行循环中的语句，然后判断表达式是否为真，如果为真则继续循环；如果为假，则终止循环。因此，do-while 循环至少要执行一次循环语句。

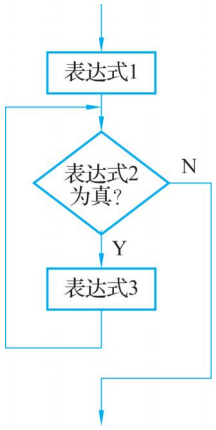


图 1-28　for 循环流程图

2. for 循环语句

for 循环语句的基本格式是

for(表达式 1；表达式 2；表达式 3)

　{语句;}

for 循环流程如图 1-28 所示，执行这条语句可以分成五个步骤：

(1)先求表达式 1；

(2)再求表达式 2，若其值为真，则执行 for 语句中指定的语句，然后执行步骤(3)；若其值为假(0)，则结束循环，直接转到步骤(5)，也就是结束循环，执行后面的语句；

(3)如果表达式 2 为真，求解表达式 3；

(4)转回上面执行步骤(2)；

(5)如果表达式 2 为假，循环结束，执行 for 语句下面的语句。

同样，我们也可以用 for 循环设计 1ms 延时程序。

```
void delay_1ms()                    // 函数名称 delay_1ms
{                                   // 函数开始
    unsigned char i=0;              // 定义无符号字符型变量 i 并赋初值0
    for(i=1;i<=121;i++)             // 循环语句 for
        {                           // 循环体开始
        ;                           // 空语句
    }                               // 循环体结束
}                                   // 函数结束
```

函数名称同样是 delay_1ms，定义无符号字符型变量 i 并赋初始值 0；再设计 for 语句，首先 i=1；然后判断 i 是否<=121 满足条件，则执行 for 循环中的空语句，然后执行 i++，再回去判断 i 是否<=121，如此循环下去，直到 i>121，退出 for 循环。可见，“空语句、变量增加 1 和判断语句”都被执行了 121 次，这样单片机只要执行一次 delay_1ms 函数就大约消耗 1ms 的时间。

for 语句在应用时也有几点要注意：

(1)for(表达式 1；表达式 2；表达式 3)中两个分号不能少；

(2)在 for 后，允许有多个语句，但必须用{}括起来，也建议一直用{}括起来；

(3)for(;;){语句;}可以实现无限循环，跟 while(1)类似。

3. for 循环嵌套结构

for(表达式 1；表达式 2；表达式 3)

{

```
    for(表达式4；表达式5；表达式6)
    {语句组;}
}
```

这是一个两层 for 循环嵌套结构，在编写程序时，一般分层次编写。第一个 for 循环，叫作外循环，里面的 for 循环，叫作内循环，语句组被执行的次数是外循环的次数乘以内循环的次数。较长时间的延时就可以采用这种结构。

如：单片机实现 1s 延时程序

```
void delay_1s()                    //函数定义,函数名 delay_1s()
{                                  //函数开始
    unsigned char i;               //定义无符号字符变量 i
    unsigned int j;                //定义无符号整数变量 j
    for(j=0;j<=1000;j++)           //for 语句,外循环1000次
    {                              //外循环开始
        for(i=0;i<=121;i++)        //for 语句,内循环121次
            {;}                    //空语句,内循环
    }                              //外循环结束
}                                  //函数结束
```

三、C51 中函数的应用

在进行 C51 程序设计时，为了增强程序的功能，使程序结构清晰，经常用到函数。所谓“函数”，是一段能够完成特定任务的其他函数可通过调用语句来执行的程序。函数又分成很多种，从函数定义角度可以分为用户自定义函数和库函数，这里主要介绍自定义函数。

从有无返回值角度，函数可以分为有返回值的函数和无返回值的函数；从数据传送角度，函数可以分为无参函数和有参函数。但每个工程里面都有一个特殊的函数，就是主函数，用 main( ) 表示。一个典型的单片机程序必须包含且只能包含一个主函数，即 main 函数。单片机复位后，从主函数的第一条语句开始执行。一个 C51 语言程序通常由一个 main 主函数和若干个其他函数构成。在主函数中调用其他的函数。

(一) 函数的定义

有人说，函数的声明就是定义函数的名称和函数代码，其实函数的声明和定义是有区别的。声明一般在主函数之前，通常放在程序的最前面，然后进行函数的定义。但习惯上，声明的同时对其进行定义，所以可能会看到 C51 中定义的函数都在主函数前面，因为这些函数一般都被主函数调用。

函数定义的一般形式是：类型声明符 函数名(形参类型声明，形式参数列表){语句：这是函数的主体功能部分；return 语句：这是函数计算值的返回。}

例如：下面的函数

```
Int   pf(int x)
      {
            int a;
            a=x*x;
            return(a);
      }
```

定义了一个整型的函数 pf，一个整型的形参 x，函数计算 x 的平方，再把计算结果返回给主调函数。该函数只是给出计算一个整型变量平方值的抽象方法。因此，它自己不能独立运行，必须由调用者(比如函数 main( ))通过调用语句，把要计算的对象传递给它。这是一种很常见的函数定义的形式。但在应用中还有其他形式：

(1)如果没有类型说明符出现，函数返回一个整型值。

(2)如果函数没有返回值，则可以采用 void 说明符。

(3)函数类型的说明必须处于对它的首次调用之前。这类似于变量必须要先定义，后使用，函数也一样。

(4)当函数被调用时，形式参数列表中的变量用来接收调用参数的值。

(5)如果函数没有返回值，则可以省略 return 语句。

(二)函数的返回值

了解了函数定义的基本形式后，再来看函数的参数与返回值。函数的参数又包含形参和实参。形参是在函数定义中出现的参数，可以看作是一个占位符，它没有数据，只能等到函数被调用时接收传递进来的数据，所以称为形式参数，简称形参。而实参是函数被调用时给出的参数，包含了实实在在的数据，会被函数内部的代码使用，所以称为实际参数，简称实参。形参和实参的功能是传递数据，发生函数调用时，实参的值会传递给形参。

例如：

```
int   pf(int x)
      {
            int a;
            a=x*x;
            return(a);
      }
```

这是之前定义的函数 pf，其中 int x 就是形参，它不是一个具体的值。又如：

```
void   main()
      {
            int   n, f;
            n=10;
            f=pf(n);
            while(1);
      }
```

在主函数中，通过 f=pf(n)这条语句，把 n=10 的值传递给了 x，n 就是实参。这是形参和实参最明显的区别。另外，我们还需要了解一下形参和实参的本质区别：

(1)形参变量只有在函数被调用时才会分配内存，调用结束后，立刻释放内存，所以形参变量只有在函数内部有效，不能在函数外部使用。

(2)实参可以是常量、变量、表达式、函数等，但在进行函数调用时，它们都必须有确定的值，以便把这些值传送给形参，所以应该提前用赋值、输入等办法使实参获得确定值。

(3)实参和形参在数量上、类型上、顺序上必须严格一致，否则会发生“类型不匹配”的错误。当然，如果能够进行自动类型转换，或者进行了强制类型转换，那么实参类型也可以不同于形参类型。不过不建议初学者使用。

(4)函数调用中发生的数据传递是单向的，只能把实参的值传递给形参，而不能把形参的值反向地传递给实参。简单地说，形参是函数定义时用的，没有赋值，实参是函数调用时用的，有确定的值。而函数的返回值是指函数被调用执行之后，最终返回给主函数的计算结果。函数的返回值通过 return 语句返回给主函数。

return 语句的一般形式为：

return(表达式)；

有时这个括号“()”可以不加。

(三)函数的使用

最后我们再来看函数调用的概念就很清晰了。所谓函数调用，就是我们在程序中如何使用函数。在 C51 语言中，函数调用的一般形式有两种。

1. 函数语句

如前面编写的没有返回值的延时函数，延时 nms 的函数等。

视频学习：C51中函数的应用

2. 函数表达式

如前文例子中调用的 pf 函数 s=pf(n)。

这两种形式是最简单、最常用的。当然，函数的定义、调用和参数的传递还有很多很复杂高级的用法，这取决于读者的 C 语言基础，但 C51 中其实并没有那么复杂，简单的不一定不行，实用最重要。

四、彩灯控制电路的软件设计

视频学习：彩灯控制电路的软件设计

本文实现彩灯功能的程序是基于 C 语言编写的，采用的是 Keil C 软件，下面就先来看一看，利用 Keil C 软件如何编写一个单片机的程序，并最终生成一个单片机可以执行的文件，也就是后缀为 .HEX 的文件。关于 Keil C 软件的使用请见任务三。

(一) 系统流程图设计

在程序编写之前，一般我们需要进行流程图的设计，建议养成这样的思考和编程习惯。

流程图样例如图 1-29 所示。首先，程序开始，然后进行初始化，初始化主要包括单片机一些片内资源的设置，变量的定义等，由于有 5 种模式，所以需要选择到底运行哪一种模式，这种模式运行完之后，因为紧接着要进行下一种模式的运行，需要进行模式的控制，再回去判断选择哪一种模式。为了方便后面进行外部控制，建议用这个思路去编写程序。

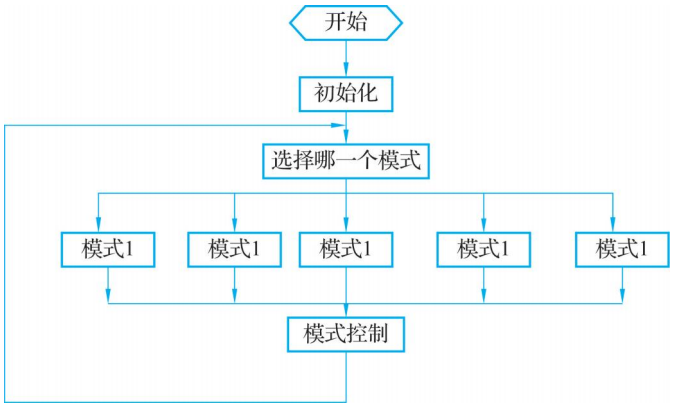


图 1-29　流程图样例

(二) 程序设计

1. reg51.h 头文件

首先是程序初始化定义部分。

```
#include<reg51.h>                    //头文件
sbit led1=P1^0;                      //位定义
sbit led2=P1^1;
sbit led3=P1^2;
sbit led4=P1^3;
sbit led5=P1^4;
sbit led6=P1^5;
sbit led7=P1^6;
sbit led8=P1^7;
unsigned char select;                //模式选择控制变量
```

初始化包括：头文件、位定义、主程序模式控制变量定义。头文件位 reg51.h，这个头文件里面主要是把 51 单片机里面的特殊功能寄存器进行了定义，如 P1。这样在程序里面输入 P1，注意是大写的“P”，编译器在编译时就不会提示错误信息。在用 C 语言编程时往往第一行就是头文件，51 单片机为 reg51.h 或 reg52.h。51 单片机相对来说

比较简单，头文件里面内容不多，像飞思卡尔、ARM 系列的单片机头文件往往内容就非常多，尽管如此，对一些读者来说，51 单片机的头文件还是搞不太清楚，这里具体说明一下。

1)“文件包含”处理概念

所谓“文件包含”是指在一个文件内将另外一个文件的内容全部包含进来。因为被包含的文件中的一些定义和命令使用的频率很高，几乎每个程序中都可能要用到，为了提高编程效率，减少编程人员的重复劳动，将这些定义和命令单独组成一个文件，如 reg51. h，然后用#include<reg51. h>包含进来，这个就相当于工业上的标准零件，拿来直接用就可以了。这种标准零件的工作形式很方便，因此，一个程序员的硬盘里都保存了很多这样的可以重复使用的，对于他自己来说是标准的零件，也就是自定义的头文件。从而能极大地减少重复编程。

2)寄存器地址及位地址声明的原因

reg51. h 里面主要是一些特殊功能寄存器的地址声明，对一些可为寻址的特殊功能寄存器，还包括一些位地址的声明。如：

```
sfr P1=0x90;
sfr IE=0xA8;
sbit EA=0xAF;
```

sfr P1=0x90 这句话表示：P1 口所对应的特殊功能寄存器 P1 在内存中的地址为 0x90，sbit EA=0xAF 这句话表示 EA 这一位的地址为 0xAF。注意，这里出现了一个使用很频繁的 sfr 和 sbit。

sfr 表示特殊功能寄存器，它并非标准 C 语言的关键字，而是 Keil 为能直接访问 80C51 中的 SFR 而提供了一个新的关键词，其用法是：

sfr 特殊功能寄存器名=地址值。注意对于头文件里的“特殊功能寄存器名”，用户是可以修改的，如 P1=0x80，可改为 A1=0x80，但 sfr 和地址值则不能更改，否则会编译出错。不过，如果定义名称改成 A1 了，则程序中使用时就需要用 A1。所以，为了方便使用，一般定义的名称都与 51 单片机的内部寄存器定义名称一致。

sbit 表示位，它也是非标准 C 语言的关键字，编写程序时如需操作寄存器的某一位(可位寻址的寄存器才能用)时，需定义一个位变量，此时就要用到 sbit，如 sbit deng=P1^0，sbit EA= 0xAF。需要注意的是，位定义时有些特殊，用法有 3 种：

第一种方法：sbit 位变量名=寄存器位地址值

第二种方法：sbit 位变量名=SFR 名称^寄存器位值(0~7)

第三种方法：sbit 位变量名=SFR 地址值^寄存器位值

如：

```
sbit IT0=0x88                  //0x88是 IT0的位地址值
    sbit led=P1^2              //其中 P1必须先由 sfr 定义
    sbit EA=0xA8^7             //0xA8就是 IE 寄存器的地址值
```

以上 3 种定义方法需注意的是：IT0、led、EA 可由用户随便定义，但必须满足 C 语言对变量名的定义规则。除它们以外，其他的则必须按照上面的格式写，如“名称^变量位地址值”中“^”，它是由 Keil 软件规定的，不能写成其他格式，只能这样才能编译通过。以上对寄存器地址和位地址的定义和声明作了简单的解释，需要注意的是：只有对寄存器及相关位进行声明地址后，我们才能对其赋相关的值，Keil 软件才能编译通过。

3）内存、SFR、位、地址等的通俗解释

寄存器地址和位地址（前提是可以按位寻址）声明的目的是告诉 C 编译器相应寄存器及其位在内存中的位置编号，这样我们对寄存器及一些位赋的变量和数值才能正确保存，然后才能供 CPU 调用，完成相应的功能。

寄存器（SFR）、位，地址、内存等，以及前面出现 ROM、RAM 等名词，可以这么理解：内存好比宾馆，ROM、RAM、SFR 相当于宾馆里具体的有 3 种不同功能楼层（具体这个宾馆多少层即多少 ROM、RAM、SFR，视各个宾馆或者每种单片机决定），每层 8 个房间有 8 个位置，每个位置要么住男人要么住女人，相当于每位要么放入数字 1 要么放入数字 0，Keil 编译器就相当于宾馆的工作人员，旅客去住旅馆相当于写程序的过程，住宾馆的人必须事先要给工作人员说你是哪一层哪一个房间（即声明寄存器地址和位地址），宾馆工作人员才能把你带到你的房间里去。即：只有对寄存器及相关位进行声明地址后，我们才能对其进行赋相关的值，Keil 软件才能编译通过。

4）REG51. H 头文件的具体解释

```
#ifndef__REG51_H__
#define__REG51_H__
/*   BYTE Register */
sfr P0    =0x80;              //三态双向 I/O 接口 P0口,低8位地址总线/数据总线
sfr P1    =0x90;              //准双向 I/O 接口 P1口
sfr P2    =0xA0;              //准双向 I/O 接口 P2口,高8位地址总线,一般也作普通 I/O 接口用
sfr P3    =0xB0;              //双功能
                              //1. 准双向      I/O 接口          P3口
                              //2.    P30        RXD 串行数据接收
                              //      P31        TXD 串行数据发送
                              //      P32        外部中断0
                              //      P33        外部中断1
                              //      P34        定时/计数器 T0
                              //      P35        定时/计数器 T1
                              //      P36        WR   片外 RAM 写脉
                              //      P37        RD   片外 RAM 读脉冲
sfr PSW  =  0xD0;             //可位寻址(C 语言编程时可不考虑此寄存器)
                              //程序状态寄存器 Program Status WO
                              //psw.7(CY)      进位标志
                              //psw.6(AC)低四位向高四位进位辅助进位标志位,主要用于十进制
                                调整
                              //psw.5(F0)用户可自定义的程序标志位
```

```
                        //psw.4(RS1)psw.3(RS0)工作寄存器选择位任一时刻只有一组寄存器在
                          工作
                        //RS1 RS0 为 0 0 对应 0 区      00H~07H
                        //RS1 RS0 为 0 1 对应 1 区      08H~0fH
                        //RS1 RS0 为 1 0 对应 2 区      10H~17H
                        //RS1 RS0 为 1 1 对应 3 区      18H~1FH
                        //psw.2(OV)      溢出标志位
                        //psw.1(  )      保留位 ,不可使用
                        //psw.0(P)       奇偶校验位
sfr ACC = 0xE0;         //累加器 A      特殊功能寄存器      可位寻址
sfr B = 0xF0;           //寄存器 B      主要用于乘除运算
sfr SP = 0x81;          //堆栈指针寄存器 SP
sfr DPL = 0x82;
sfr DPH = 0x83;         //数据指针寄存器 DPTR、对片外 RAM 及扩展用
sfr PCON = 0x87;        //电源控制寄存器、不能位寻址,管理单片机的电源部分包括上电复位、
                          单片机复位时 PCON 被全部清 0
                        //D7 SMOD 该位与串口通信波特率有关,SMOD=0,串口方式 1 2 3 波
                          特率正常,SMOD=1,串口方式 1 2 3 波特率加倍
sfr TCON = 0x88;        //定时器/计数器   控制寄存器   可位寻址
                        //D7 TF1      定时器 1 溢出标志位
                        //D6 TR1      定时器 1 运行控制位
                        //D5 TF0      定时器 0 溢出标志位
                        //D4 TR0      定时器 0 运行控制位
                        //D3 IE1      外部中断 1 请求标志
                        //D2 IT1      外部中断 1 触发方式
                        //D1 IE0      外部中断 0 请求标志
                        //D0 IT0      外部中断 0 触发方式
sfr TMOD = 0x89;        //定时器/计数器工作方式寄存器,不能位寻址
                        //D7 定时器 T1GATE 门控制位,GATE=0;T1 由 TRX 启动,GATE=1;
                        //定时器/计数器由 TR1 和外部中断引脚(init0,1)来共同控制
                        //D6 定时/计数器 T1 选择位,为 0 选择定时器模式,为 1 选择计数器模式
                        //D5 D4 M1 M0      定时器 T1      工作方式选择
                        // M1M0 为 0 0      工作方式 0      13 位定时器/计
                        // M1M0 为 0 1      工作方式 1      16 位定时器/计数
                        // M1M0 为 1 0      工作方式 2      8 位自动重装定时
                        // M1M0 为 1 1      工作方式 3      仅适用 T0
                        //D3 定时器 T0GATE 门控制位,GATE=0;T 器 0 由 TRX 启动,GATE=1;
                        //定时器/计数器 T0 由 TR0 和外部中断引脚(init0,1)来共同控制
                        //D3 定时器/计数器 0 选择位,为 0 选择定时器模式,为 1 选择计数器模式
                        //D1 D0 M1 M0      定时器 T0      工作方式选择
                        // M1M0 为 0 0      工作方式 0      13 位定时器/计
                        // M1M0 为 0 1      工作方式 1      16 位定时器/计数
                        // M1M0 为 1 0      工作方式 2      8 位自动重装定时
                        // M1M0 为 1 1      工作方式 3      仅适用 T0
sfr TL0 = 0x8A;         //定时器/计数器 T0 低 8 位
```

```
sfr TL1 = 0x8B;          //定时器/计数器 T1低8位
sfr TH0 = 0x8C;          //定时器/计数器 T0高8位
sfr TH1 = 0x8D;          //定时器/计数器 T1高8位
sfr IE = 0xA8;           //中断允许寄存器,可位寻址
                         //D7     EA   终端总允许
                         //D6     空,预留
                         //D5     空,预留
                         //D4     ES   串行口中断允许
                         //D3     ET1  定时器 T1中断允许
                         //D2     EX1  外部中断1中断允许
                         //D1     ET0  定时器 T0中断允许
                         //D0     EX0  外部中断0中断允许
sfr IP = 0xB8;           //中断优先级寄存器,可位寻址
                         //D7  空,预留
                         //D6  空,预留
                         //D5  空,预留
                         //D4  PS 串行口优先级控制位
                         //D3  PT1 定时器 T1优先级控制位
                         //D2  PX1 外部中断1优先级控制位
                         //D1  PT0 定时器 T0优先级控制位
                         //D0  PX0 外部中断0优先级控制位
sfr SCON = 0x98;         //串行口控制寄存器,可位寻址
                         //D7 D6 SM0 SM1 ,串行口工作模式选择
                         //SM0 SM1为 0 0,8位移位寄存器
                         //SM0 SM1为 0 1,10位异步收发
                         //SM0 SM1为 1 0,11位异步收发
                         //SM0 SM1为 1 1,11异步收发
                         //D5 SM2 ,多机通信控制
                         //D4 REN , 允许串行接收
                         //D3 TB8 ,方式2,3中发送的第8位
                         //D2 RB8 ,方式2,3中接收的第8位
                         //D1 TI ,发送中断标志位
                         //D0 RI ,接受中断标志位
sfr SBUF = 0x99;         //串行数据缓冲区 下面是位寻址区,上面做过解释的就不再一一解释了
************************************************************************
/*  BIT Register  */
/*  PSW  */
sbit CY   = 0xD7;
sbit AC   = 0xD6;
sbit F0   = 0xD5;
sbit RS1  = 0xD4;
sbit RS0  = 0xD3;
sbit OV   = 0xD2;
sbit P    = 0xD0;
/*  TCON  */
```

```
sbit TF1    = 0x8F;
sbit TR1    = 0x8E;
sbit TF0    = 0x8D;
sbit TR0    = 0x8C;
sbit IE1    = 0x8B;
sbit IT1    = 0x8A;
sbit IE0    = 0x89;
sbit IT0    = 0x88;
/*   IE   */
sbit EA     = 0xAF;
sbit ES     = 0xAC;
sbit ET1    = 0xAB;
sbit EX1    = 0xAA;
sbit ET0    = 0xA9;
sbit EX0    = 0xA8;
/*   IP   */
sbit PS     = 0xBC;
sbit PT1    = 0xBB;
sbit PX1    = 0xBA;
sbit PT0    = 0xB9;
bit PX0 0xB8;- 63-
sbit PX0    = 0xB8;
/*   P3   */
sbit RD     = 0xB7;
sbit WR     = 0xB6;
sbit T1     = 0xB5;
sbit T0     = 0xB4;
sbit INT1=0xB3;
sbit INT0=0xB2;
sbit TXD    = 0xB1;
sbit RXD    = 0xB0;
/*   SCON   */
sbit SM0    = 0x9F;
sbit SM1    = 0x9E;
sbit SM2    = 0x9D;
sbit REN    = 0x9C;
sbit TB8    = 0x9B;
sbit RB8    = 0x9A;
sbit TI     = 0x99;
sbit RI     = 0x98;
#endif
```

另外，位定义 led1=P1^0，输入 led1 和输入 P1^0 是等效的，但写 led1，就增加了程序的可读性，也方便程序员写程序时加以区别。再定义一个模式控制变量，这个变量在这里定义的话，它是一个全局变量，在整个文件的所有地方都可以对它进行操作。

2. 主程序

再看主程序。

```
void main()
{
    while(1)
    {
        switch(select)
        {
            case 0:light_0();break;
            case 1:light_1();break;
            case 2:light_2();break;
            case 3:light_3();break;
            case 4:light_4();break;
            default:break;
        }
        select ++;
        if(select==5)select =0;
    }
}
```

主程序的主体部分，都在 while(1)死循环里面，这样能够保证 while 里面的程序无限循环执行。注意，单片机在执行程序时，不能执行 end。当然也可以用 for 语句去实现，效果是一样的。While 里面首先是一个 switch 语句，根据 select 的值，选择 5 种运行模式，执行完一种模式后退出 switch 语句，紧接着控制 select 的值，然后回去运行 switch 语句，一直如此循环下去。这里面，5 种模式分别对应了 5 个子程序。5 种模式分别是：向左方向流水灯(慢速)、向左方向流水灯(稍快)、向左方向逐个点亮(慢速)、向右方向逐个熄灭(快速)、整体闪烁 4 次。

我们以第一个模式的程序为例，介绍彩灯是如何控制的。

根据之前设计的彩灯控制电路图，只要 P1 口对应位输出低电平，对应的灯就亮，输出高电平，对应的灯就灭。也就是只要控制定义的位变量 led1～led8 的输出就可以了。所以顺序是，led1=0 第一个灯亮，延时 500ms，也就是灯亮 500ms，然后第一个灯灭，紧接着第二个灯亮，再延时 500ms 后灯灭……一直到第八个灯。当然，这只是其中一种实现方式，按位操作的模式，同样的功能，可以通过多种方式去实现。比如，可以对 P1 寄存器按字节整体进行操作，如 P1=0xfe，这个时候相当于 P1=11111110B，也是第一个灯亮，其他灯灭。

其他模式的程序也是类似的编写方法，以 2019 年全国大学生电子设计大赛为例，高职组有一个题目就与彩灯相关，可以查阅该题目，实战一下。另外，还可以利用 LED 灯组成不同的形状，实现不同的控制效果。比如，设计五星红旗外形的流水灯，编写程序控制，让国旗在你的程序设计中酷炫起来。

视频学习：基于LEILC彩灯控制程序设计

另外，这个程序里面还用到了一个延时子程序。这里用 for 循环语句构成了一个带参数传递的延时 nms 的子程序。

3. 参考程序

建议先独立编写程序，实在调不出来，再参考。

```
#include<reg51. h>
sbit led1=P1^0;
sbit led2=P1^1;
sbit led3=P1^2;
sbit led4=P1^3;
sbit led5=P1^4;
sbit led6=P1^5;
sbit led7=P1^6;
sbit led8=P1^7;
unsigned char select;
void delay(unsigned int nms)                    // 延时程序
{
      unsigned int i,j;
      for(i=nms;i>0;i- - )
      for(j=120;j>0;j- - );
}
void light_0(void)                              // 彩灯模式0
{
led1=0;delay(500);led1=1;
led2=0;delay(500);led2=1;
led3=0;delay(500);led3=1;
led4=0;delay(500);led4=1;
led5=0;delay(500);led5=1;
led6=0;delay(500);led6=1;
led7=0;delay(500);led7=1;
led8=0;delay(500);led8=1;
}
void light_1(void)                              // 彩灯模式1
{
      led1=0;delay(200);led1=1;
      led2=0;delay(200);led2=1;
      led3=0;delay(200);led3=1;
      led4=0;delay(200);led4=1;
      led5=0;delay(200);led5=1;
      led6=0;delay(200);led6=1;
      led7=0;delay(200);led7=1;
      led8=0;delay(200);led8=1;
}
void light_2(void)                              // 彩灯模式2
{
```

```
        led1=0;delay(500);
        led2=0;delay(500);
        led3=0;delay(500);
        led4=0;delay(500);
        led5=0;delay(500);
        led6=0;delay(500);
        led7=0;delay(500);
        led8=0;delay(500);
        P1=0xff;delay(500);
}
void light_3(void)                          // 彩灯模式3
{
        P1=0x00;
        delay(200);led8=1;
        delay(200);led7=1;
        delay(200);led6=1;
        delay(200);led5=1;
        delay(200);led4=1;
        delay(200);led3=1;
        delay(200);led2=1;
        delay(200);led1=1;
        delay(200);
}
void light_4(void)                          // 彩灯模式4
{
        P1=0x00;    delay(300);
        P1=0xff;    delay(300);
        P1=0x00;    delay(300);
        P1=0xff;    delay(300);
        P1=0x00;    delay(300);
        P1=0xff;    delay(300);
        P1=0x00;    delay(300);
        P1=0xff;    delay(300);
}
void main()                                 // 主程序
{
        while(1)
        {
            switch(select)
            {
                case 0:light_0();break;
                case 1:light_1();break;
                case 2:light_2();break;
                case 3:light_3();break;
                case 4:light_4();break;
```

```
                default:break;
            }
            select ++;
            if(select==5)select =0;
        }
    }
```

4. 编程的注意事项

(1)程序的层次、结构要清晰，养成良好的习惯，这会减少调试的工作量。

(2)确定变量和函数定义时，最好采用助记符的方式，增加程序可读性。

(3)各类控制语句后面紧跟{ }，在输入时成对输入，能减少调试的工作量。

(4)敢于下手，刚开始不会编写程序，可以多参考其他程序，并尝试改写、增删功能，以达到融会贯通。

(5)功能复杂时，逐个调试。如本例有 5 个工作模式，可以先调试 1 个，成功后再调试其他，逐个击破，事半功倍。

(6)调试程序时，先调“看得见的”，后调“黑箱子”。“看得见的”一般指显示，因为你能看见，知道哪错误了，根据现象判断错误出现的地方，这是需要不断地练习才能提升的核心能力。其他看不见的功能就可以根据“看得见的”部分辅助调试。

(7)调试程序时，善于利用信息输出窗口提示的错误信息，能快速定位错误位置。

任务五

拓展学习

一、彩灯电路的硬件调试

前面的讲解均基于仿真软件，仿真软件并不能百分之百地反映实际硬件电路的功能和特性。所以，有很多读者还是喜欢基于实验板学习单片机，关于单片机实验板，需要说明几个问题：

(1)是否需要单片机实验板？很多人认为仿真软件问题多多，但实际上，目前的仿真软件功能越来越强大，Proteus 和 Keil 软件实际上都具有仿真的功能，能有效地帮助理解单片机。当然，实验板也是有必要的，因为它非常直观，可以提高学习兴趣，增强对单片机控制系统的认识。

(2)需要什么样的单片机实验板？目前网上购物平台的 51 单片机实验板有成千上万种，那要选择什么样的实验板呢？总体原则就是：①涵盖单片机学习时需要用到的基本功能。如显示类的彩灯、数码管、液晶，初学者常用的液晶是 1602 和 12864；键盘有独立按键和矩阵键盘；信号采集和输出类的 AD 和 DA；扩展学习类的，这类有没有都可以，等学到这些的时候，可以自己动手设计。②操作方便，尤其是下载程序，最好一根 USB 线就解决问题，满足学习就可以了。

(3)不能依赖实验板，要敢于动手“设计”电路。实际上，简单的控制系统，需要大家自己设计电路的地方越来越少，更多的是整合现有的电路模块，类似于系统集成。

(一)硬件功能介绍

1. 实验板

本课程使用的实验板如图 1-30 所示。包含供电、程序下载和串口三合一的电路(这样不需要扩展 USB 转串口模块，一根 USB 线就能够完成程序下载)，流水灯模块，矩阵按键模块，独立按键模块，直流电机驱动模块，液晶接口(包括 1602 和 12864)模块，8 位数码管模块，ADDA 芯片 PCF8591，EEPROM 芯片 24C02，蜂鸣器以及单片机的最小系统(包括复位、晶振、电源)。最关键的是，单片机是可以随时更换的，如果操作不当，导致单片机被烧毁，单片机还可以更换，延长实验板的使用寿命。

2. 实验板原理图

实验板原理如图 1-31 所示。

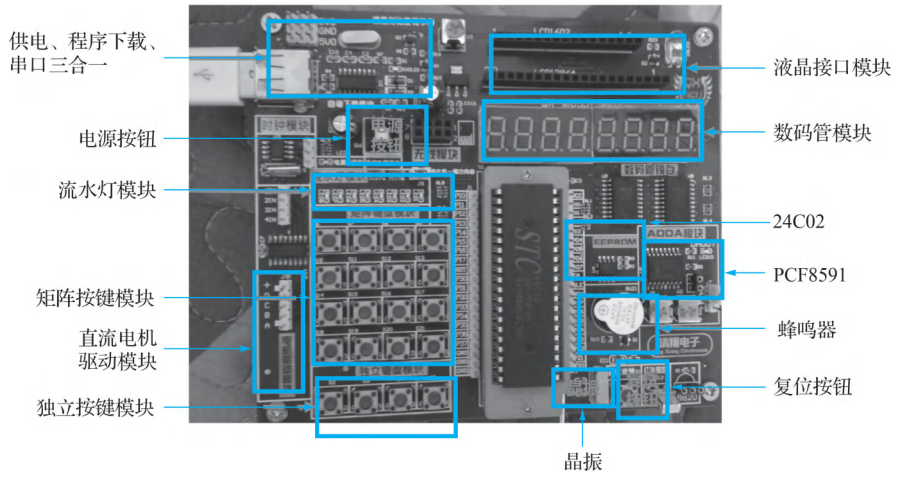


图 1-30　实验板结构图

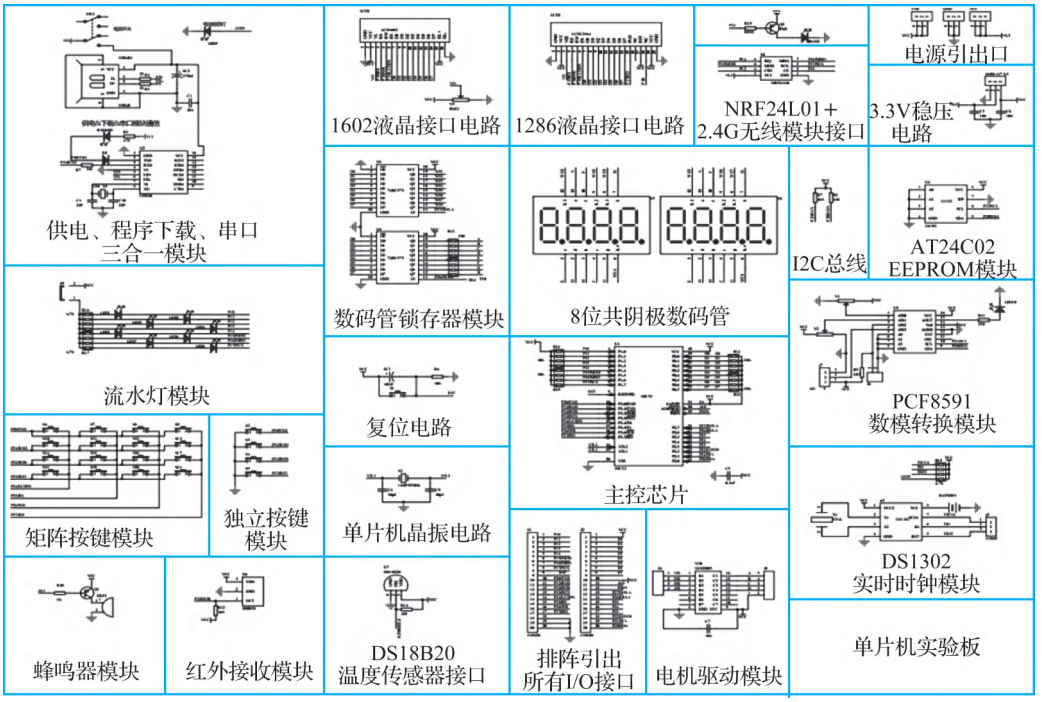


图 1-31　实验板原理图

目前学到的几个电路有：

(1)复位电路：采用的是上电复位和手动复位的混合复位电路。

(2)晶振电路：采用的是 11.0592MHz 的无源晶振。

(3)另外，电源是 5V 电源，直接通过 USB 接口供电。

实验板彩灯电路原理如图 1-32 所示，可以看到，8 个发光二极管的正极同一个 4.7kΩ 的电阻，接在了电源的正极，二极管的负极接在了 P1 口，并通过 J9 进行跳线的选择，以

方便其他功能的扩展。

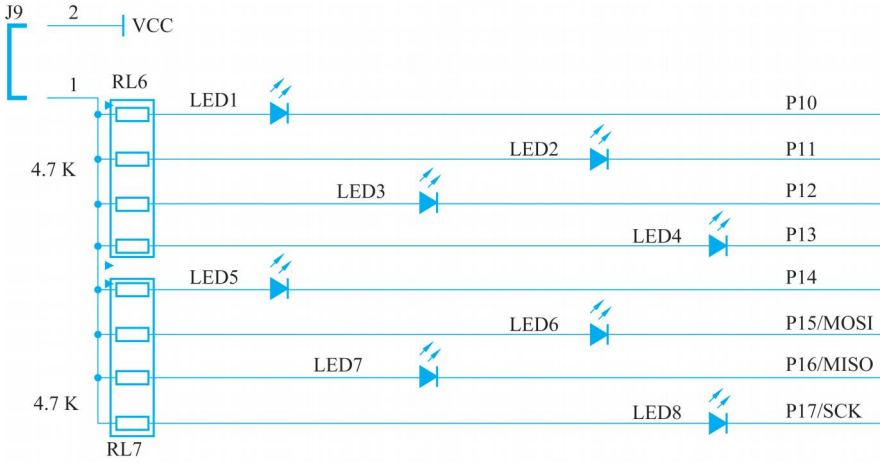


图 1-32　实验板彩灯电路原理图

视频学习：实验板功能介绍

(二)彩灯程序的下载

彩灯的功能演示，是基于 Proteus 仿真软件设计的，那这个程序如何下载到实验板中？需要修改程序吗？

视频学习：单片机下载程序

STC 单片机程序的下载

由于实验板用的单片机是 STC89C52，因此我们需要用到 STC 专门的程序下载软件。双击软件快捷方式，打开软件，进入初始界面，如图 1-33 所示。

图 1-33　STC-ISP 初始界面

首先选择单片机的型号，一定要与你的实验板上的 STC 单片机的型号完全一致，选 STC89C52。把实验板和电脑用 USB 线连接好，由于笔记本电脑一般没有串口，所以需要 USB 转串口的芯片才能保证单片机和笔记本电脑的通信。本例选择的实验板上面是有 USB 转串口的芯片 CH340 的，当连接好实验板和电脑后，“串口号”栏就会自动出现 USB-SERIAL CH340 这个串口号了。然后打开程序文件，找到之前编译好的 light. hex 文件，确认，最后点击“下载/编程”按钮。右下角的窗口显示“正在监测目标单片机”，这时，把实验板电源打开，当右下角显示图 1-34 界面时，说明程序下载完成了。

下载完成后的运行程序，发现不对程序进行任何的更改，仿真软件上的功能也能在实验板上正常运行，这主要是由于仿真的电路图和实验板上的电路图是一样的。

视频学习：彩灯程序的下载与调试

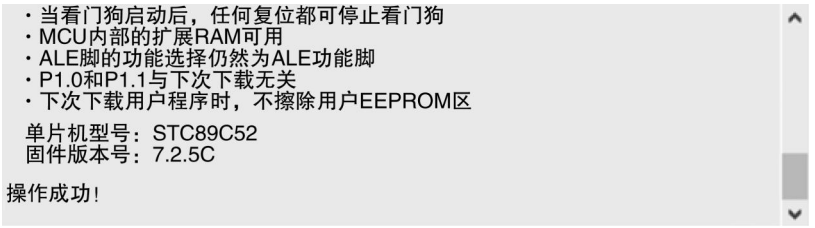


图 1-34　下载成功界面

二、如何学好单片机

作为单片机初学者，该从哪方面入手，才能更快地学好单片机？

(一)学什么样的单片机

目前，常见的单片机是以 51 为内核的 8 位单片机。它的学习资料非常多，学习成本低。51 单片机作为智能设备开发的入门学习机是非常适合的，各方面的成本均低、开发简单、下载程序便捷(大多可以一键下载)。而 ARM 太神秘、PLC 太高贵，AVR、PIC、MPS430 等虽然功能很多，但恰恰因为功能多，反而给初学者带来很多困惑，还是 51 单片机最合适。

(二)多看书

单片机是一项非常重视动手实践的科目，如果你在单片机方面比较擅长，那一定也是计算机编程和电子技术高手。既然是这样，学习单片机，是不是就以动手为主呢？书可以少看吧？学习单片机必须得看书，因为从书中你可以了解单片机各个功能寄存器(比如引脚控制寄存器、定时、中断、串口相关寄存器)，控制单片机的核心是用程序去控制单片机的各个功能寄存器，给寄存器赋值二进制数据 0 或者 1。对于像中断、定时器、串口、AD 转换、PWM 等内部资源，单片机也是通过二进制数据 0 和 1 进行设置和使用的。比如单片机引脚寄存器 P1，语句 P1=0xfe(汇编写作 MOV P1，#0FEH)指控制单片机 P1 口的第一个引脚输出低电平，其他引脚输出高电平，在高低电平的作用下，

自然会对外设电路产生影响。比如中断允许寄存器 IE，语句 IE = 0x81（汇编写作 MOV IE，#81H）说明控制中断打开总开关和外部中断 0 的子开关。因此，看书时只需大概了解单片机各管脚、各个功能寄存器是干什么的，能实现什么功能就够了。

（三）动手

可以从简单的单片机最小电路——控制一个 LED 闪烁的电路开始。可以使用面包板，只要经过电路设计、焊接、程序编写、程序下载等环节，并成功实现功能，在这个过程中便掌握了如何下载程序到单片机内部、如何识别单片机、如何设计单片机最小电路的技能，并会对单片机硬件系统有所了解。单片机编程就是与单片机对话，所以，要了解谈话的对象，才不至于谈话内容风马牛不相及。

（四）练习

实在不会焊板子，那就买块单片机实验板吧，最好是插件的那种，自己有台电脑则更方便，把实验板和电脑连好，安装上必备的软件（Keil、Proteus、STC-ISP 等），下载参考程序，并修改参考程序，从最简单的彩灯实验开始，等你发现你能控制彩灯，了解彩灯的软硬件设计的时候，你已经一只脚迈进单片机应用的门槛了。这时，你就会发现单片机的魅力，后续的学习也会更加有信心。

（五）汇编还是 C 语言

选择汇编语言还是 C 语言作为编程语言？这个问题困扰了很多人，汇编语言可能就像学一门外语一样，C 语言则更像是用母语，当然还是母语上手快了。所以，建议用 C 语言比较好，模块化管理编程方便，移植性强，适合编写大程序。如果已经有 C 语言的基础更好，如果没有，也可以边学单片机边学 C 语言。

那汇编语言是不是不学了呢？不是的，一定要学习。首先入门单片机的时候要用汇编语言，它的语法简单。其次，如果你要做单片机程序的高级设计师，那汇编语言是非掌握不可的，很多高级单片机像 ARM 都是汇编语言作为引导代码的，还有就是很多新出品的单片机起先也是先有汇编语言编译器之后才有 C 语言编译器。所以，对于单片机学习的过程是汇编语言入门，C 语言精通，汇编语言再升华，最后二者可以混合使用。