

工程师经验手记

51单片机轻松入门

——基于STC15W4K系列

(C语言版)

李友全 编著



 北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

联系方式：QQ 群（STC51-STM32）：324284310，个人 QQ：347305156，验证信息：单片机

邮箱：xgliyouquan@126.com

淘宝店地址：

http://shop117387413.taobao.com/search.htm?spm=a1z10.1-c.w4023-10438077395.34.1Wybca&orderType=hotsell_desc

百度网盘辅助参考视频与资料下载地址：

<http://pan.baidu.com/share/home?uk=4077802723&view=share#category/type=0>

内 容 简 介

本书以最新流行的不需要外部晶振与复位电路的可仿真的高速 STC15 系列单片机为核心，详细介绍了单片机内部功能模块，比如定时器、中断、串口、SPI 接口、片内比较器、ADC 转换器、可编程计数器阵列 (CCP/PCA/PWM) 等。每个重要知识点都有简短精炼的实例作验证，然后就是单片机常用外围接口的介绍与 STC15 系列单片机的实际产品运用实例分析。另外对单片机开发必须掌握的 C 语言基础知识与 Keil 开发环境也作了较为详细的介绍，对于没有学习过 C 语言的读者通过本书也能轻松进入以 C 语言开发单片机的学习状态。

为了快速验证本书的理论知识，作者设计与本书配套的双核（两个仿真型单片机）实验板，功能强大，操作简单，直观，除用于本书实验测试外，也可用于产品前期开发。

本书可作为普通高校计算机类、电子类、自动控制类、仪器仪表类、机电一体化类等相关专业教学用书，对已有一定设计经验的单片机工程师也有重要参考价值。

前 言

STC 单片机是在传统 8051 单片机内核的基础上进行大幅度改进升级优化而来的新一代 8051 单片机，具有高速、高可靠、低功耗、外围模块多、ISP 升级程序方便、价格低廉等显著优点，加上 STC/宏晶科技单片机的厂商“南通国芯微电子”属于中国大陆本土企业，当我们在产品设计过程中遇到问题时方便与厂家沟通获得技术支持，所以 STC 单片机已经被众多的产品设计工程师作为首选方案运用到自己的产品中去。

STC 单片机的指令系统与标准的 8051 内核完全兼容，过去的 51 单片机书籍仍然可以拿来作为辅助参考学习，对于已经熟悉传统 8051 内核单片机的读者，可以轻松过渡到 STC 可仿真的超级强大的 STC15 系列单片机或 STC 早期的 STC89 系列单片机，本书的编写建立在笔者十多年的产品设计经验基础之上，具体编写从前到后又花费了近 5 年的时间，笔者本着十年磨一剑的精神把每一个章节的内容写出水平，因此本书内容真实，言语简练，通俗易懂，对多年来传统单片机教材含糊不清的概念与重要知识都作了明确分析，全书程序代码编写规范，注重程序的通用性与移植性，让读者既能轻松看懂理论知识又能方便将程序代码移植到产品中去。

本教材主讲的单片机型号是 STC 公司的 IAP15W4K58S4（既能仿真又能 USB 直接下载程序），是目前 STC 最先进的芯片之一，内部资源十分丰富，具有 58K 字节程序存储器，4096 字节数据存储器，5 个定时器，4 个独立串口，8 通道 10 位高速 ADC 转换器，1 个 SPI 接口支持主机与从机模式、2 路 CCP/PCA/PWM、6 路带死区控制的专用 PWM、1 个比较器等，支持 USB 直接下载程序和串口下载程序，内部集成有高精度 R/C 时钟与高可靠复位电路，支持 2.5~5.5V 宽工作电压范围，只需提供电源就是单片机最小系统，只需加上一个 RS232 电平转换芯片或 USB 转串口芯片后连上电脑就成为一个功能完美的仿真系统，程序仿真调试非常方便，用此芯片可以完成本书很多高级实验，比如 TLC5615 数模转换芯片播放歌曲、SD 卡读写等，另外，此单片机在软件与硬件上都完全兼容资源略少的上一代单片机 STC15F2K60S2 系列，因此本书也完全适用于 STC15F2K60S2 系列的学习，为降低实际产品成本，本书还辅助性的介绍了 STC15W404S 系列，STC15W404S 系列资源更少一些，但管脚仍然很多，同样支持宽电压供电，带比较器功能，支持 SPI 主机与从机模式等，在功能要求比较简单的产品上，为进一步降低成本，读者也可使用 STC15W401AS 系列或 STC15W100 系列芯片。

编 者
2015 年 1 月

目 录

第 1 章 单片机高效入门

- 1.1 单片机简介
 - 1.1.1 认识单片机
 - 1.1.2 单片机的用途
 - 1.1.3 学习的典型芯片与 C 语言介绍
 - 1.1.4 本书配套实验板及相关学习工具介绍
- 1.2 点亮 1 个发光二极管
 - 1.2.1 单片机型号命名规则
 - 1.2.2 单片机引脚功能说明
 - 1.2.3 制作一个最简单的单片机实验电路
 - 1.2.4 使用 Keil uVision3 环境编写最简单的程序
 - 1.2.5 ISP 下载程序到单片机（将电脑上的目标代码“灌入”单片机中运行）
 - 1.2.6 程序解释
- 1.3 Keil 仿真
 - 1.3.1 软件仿真（标准 8051 方式仿真，不能仿真单片机新增功能）
 - 1.3.2 硬件仿真（利用 STC 专用仿真芯片仿真，可仿真所有功能）
- 1.4 经典流水灯实例
- 1.5 单片机 C 语言延时程序详解
 - 1.5.1 学会使用计算软件
 - 1.5.2 计算软件内部运算过程详解
 - 1.5.3 利用库函数实现短暂精确延时
 - 1.5.4 使用定时器/计数器实现精确延时
- 1.6 main()、void main() 和 int main() 的区别
- 1.7 printf 格式化输出函数

第 2 章 单片机开发必须掌握的 C 语言基础

- 2.1 简单数据类型与运算符
 - 2.1.1 原码、反码、补码、BCD 码、格雷码
 - 2.1.2 常量
 - 2.1.3 变量的数据类型(bit、char、int、long、float)
 - 2.1.4 变量存储空间(code、data、bdata、idata、xdata)
 - 2.1.5 变量存储类型(auto、static、extern)
 - 2.1.6 变量作用域
 - 2.1.7 运算符
 - 2.1.8 运算符的优先级与结合性
- 2.2 C51 构造数据类型
 - 2.2.1 数组：将相同类型数据组合在一起就构成数组（如数码管显示缓冲区）
 - 2.2.2 结构体：将不同类型数据组合在一起就构成结构体（如年月日 2014-12-31）
 - 2.2.3 共用体：不同变量占用相同内存地址就是共用体
 - 2.2.4 指针：用于直接读取或修改内存值
 - 2.2.5 #define 与 typedef 的差别

2.3 流程与控制

2.3.1 分支结构

2.3.2 循环结构

2.3.3 跳转结构

2.4 函数

2.4.1 函数定义

2.4.2 调用格式

2.4.3 传值调用与传地址调用 2 种方式对比

2.4.4 数组作为函数参数

2.4.5 使用指针变量作函数形式参数

2.4.6 使用结构体变量指针作为函数参数

2.4.7 函数作用域

2.4.8 库函数

2.5 模块化编程

2.5.1 头文件的编写

2.5.2 条件编译

2.5.3 多文件程序（模块化编程）

第 3 章 定时器/计数器、中断系统

3.1 定时器/计数器

3.1.1 单片机定时器/计数器工作原理概述

3.1.2 定时器/计数器的相关寄存器

3.1.3 定时器/计数器的工作方式

3.1.4 初值计算

3.1.5 编程举例

3.2 可编程时钟输出

3.3 中断系统

3.3.1 中断系统结构图

3.3.2 操作电路图中的开关(相关寄存器介绍)

3.3.3 编写中断函数

3.3.4 中断程序举例

3.3.5 外中断代码调试（按键的防抖技术）

第 4 章 串口通信

4.1 最基本的串口通信

4.1.1 串口数据发送格式

4.1.2 串口相关寄存器

4.1.3 波特率计算步骤详解

4.1.4 单片机与计算机通信的简单例子

4.2 彻底理解串口通信协议

4.3 串口隔离电路

4.4 计算机扩展串口（USB 转串口芯片 CH340G）

4.5 RS485 串行通信

4.6 SSI 通信

4.6.1 SSI 数据通信格式

4.6.2 SSI 硬件电路

4.6.3 SSI 软件实现

4.7 数据通信中的错误校验

4.7.1 校验和 (Checksum) 与重要的串口通信实例

4.7.2 CRC 校验 (全称: 循环冗余码校验)

4.8 单片机串口向计算机串口发送 2 进制、16 进制、数值与字符串

第 5 章 SPI 通信

5.1 SPI 总线数据传输格式

5.1.1 接口定义

5.1.2 传输格式

5.2 SPI 接口相关寄存器

5.2.1 SPI 相关的特殊功能寄存器

5.2.2 SPI 接口引脚切换

5.3 SPI 接口运用举例

第 6 章 I²C 通信

6.1 I²C 总线数据传输格式

6.1.1 各个位的传输要求

6.1.2 多字节传输格式

6.2 程序模块功能测试

6.2.1 硬件仿真观察 24C02 读写结果 (R/C 时钟: 22.1184MHz)

6.2.2 硬件仿真观察 24C32/64 读写结果 (R/C 时钟: 22.1184MHz)

6.3 24C02 运用实例 (断电瞬间存储整数或浮点数)

第 7 章 单片机内部比较器与 DataFlash 存储器

7.1 STC15W 系列单片机内部比较器

7.1.1 比较器结构图

7.1.2 寄存器说明

7.1.3 电路讲解与程序实例

7.2 DataFlash 存储器

7.2.1 DataFlash 操作有关的寄存器介绍

7.2.2 DataFlash 操作实例 (断电瞬间存储数据)

第 8 章 可编程计数阵列 CCP/PCA/PWM 模块(可用作 DAC)

8.1 PCA 模块总体结构图

8.2 PCA 模块的特殊功能寄存器

8.3 PCA 模块的工作模式与应用举例

第 9 章 模数转换器 ADC

9.1 模数转换器 ADC 主要技术指标

9.2 使用单片机内部的 10 位 ADC 转换器

9.2.1、ADC 相关的特殊功能寄存器

9.2.2、实例代码

- 9.3 12 位 ADC 转换芯片 MCP3202-B
- 9.4 16 位 ADC 转换芯片 ADS1110A0
- 9.5 18 位 ADC 转换芯片 MCP3421A0T-E/CH

第 10 章 数模转换器 DAC

- 10.1 TLC5615 数模转换电路与基本测试程序
- 10.2 TLC5615 产生锯齿波、正弦波、三角波
- 10.3 TLC5615 的高级运用（播放歌曲）

第 11 章 单片机实用小知识

- 11.1 复位
 - 11.1.1 外部 RST 引脚复位
 - 11.1.2 软件复位
 - 11.1.3 内部低压检测复位
 - 11.1.4 看门狗定时器复位
- 11.2 单片机的低功耗设计
 - 11.2.1 相关寄存器说明
 - 11.2.2 应用举例
- 11.3 单片机扩展 32K 外部数据存储器 62256
 - 11.3.1 电路讲解
 - 11.3.2 软件测试实例

第 12 章 常用单片机接口程序

- 12.1 数码管静态显示
- 12.2 数码管动态显示
- 12.3 独立键盘
- 12.4 矩阵键盘

第 13 章 1602 液晶

- 13.1 1602 液晶外形与电路图
- 13.2 1602 液晶运用举例
- 13.3 1602 液晶显示汉字与特殊符号

第 14 章 精密电压表\电流表\通用显示器\计数器制作

- 14.1 功能说明与电路原理分析
- 14.2 程序实例
 - 14.2.1 通用显示器功能检测程序（外部程序）
 - 14.2.2 计数器功能检测程序（外部程序）
 - 14.2.3 模块程序

第 15 章 步进电机测试

- 15.1 步进电机特点
- 15.2 步进电机的 3 种励磁方式
- 15.3 步进电机驱动电路
- 15.4 步进电机驱动实例
- 15.5 步进电机专用驱动器介绍

第 16 章 频率检测

- 16.1 频率检测的用途与频率定义
- 16.2 频率检测实例
- 第 17 章 DS1302 时钟芯片**
 - 17.1 DS1302 的 SPI 数据通信格式
 - 17.2 程序实例
- 第 18 章 红外通信**
 - 18.1 红外通信电路与基本原理
 - 18.2 红外接收软件实例
- 第 19 章 单总线 DS18B20 通信（长距离无线通信）**
 - 19.1 DS18B20 运用基础
 - 19.1.1 单只 DS18B20 温度检测电路
 - 19.1.2 DS18B20 通信时序
 - 19.1.3 DS18B20 内部功能部件 ROM、RAM、E²RAM、指令集
 - 19.1.4 读取温度步骤
 - 19.2 单只 DS18B20 的温度检测实例
 - 19.3 多只 DS18B20 的温度检测
 - 19.3.1 读取传感器代码实例
 - 19.3.2 读取传感器温度实例
- 第 20 章 SD 卡与 znFAT 文件系统**
 - 20.1 认识 SD 卡与 SD 卡驱动程序
 - 20.1.1 认识 SD 卡
 - 20.1.2 电路讲解
 - 20.1.3 通信时序与完整驱动程序说明
 - 20.2 znFAT 文件系统
 - 20.2.1 znFAT 的移植方法
 - 20.2.2 znFAT 移植实例
- 第 21 章 MP3 播放器实验（znFAT 文件系统运用实例）**
 - 21.1 MP3 介绍与电路讲解
 - 21.2 正弦测试
 - 21.3 通过 SD 卡播放 MP3 文件
- 第 22 章 数字存储示波器技巧与逻辑分析仪的操作**
 - 22.1 测量直流电源开关机瞬间输出毛刺浪涌
 - 22.2 测量稍纵即逝的红外发射信号
 - 22.3 精确测量直流电源纹波
 - 22.4 示波器带宽选用依据
 - 22.5 逻辑分析仪快速入门
- 附录 1 ASCII 码表**
- 参 考 文 献**

第 1 章 单片机高效入门

1.1 单片机简介

1.1.1 认识单片机

单片机全称是单片微型计算机，说计算机大家都知道它内部主要包含微处理器 CPU、硬盘、内存条等部件，一个单片机内部也包含了微处理器内核、程序存储器、数据存储器等，单片机的内核相当于计算机主板上的 CPU，单片机的程序存储器相当于计算机的硬盘，单片机的数据存储器相当于计算机的内存条，另外，编写过计算机运用程序的人都知道，计算机是按程序命令一条条执行语句完成所需的功能，单片机也是按程序命令一条条执行语句完成所需的功能，从这里可以看出，单片机与计算机实在是太相似了，这就是可以把它称为计算机的原因，另外，单片机拥有的这么多的结构部件都是集成在单一的一块集成电路芯片上的，加上体积微小，所以全称就是单片微型计算机，简称单片机。外观如图 1-1 所示。

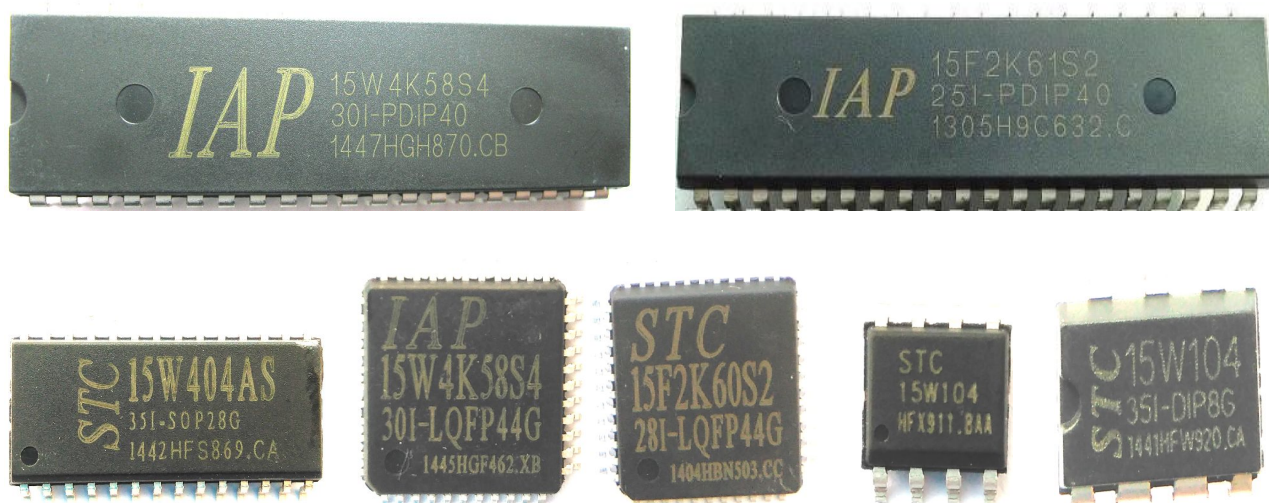


图 1-1 单片机常见外形图

单片机与普通集成电路的区别是：普通集成电路功能是固定死的，使用者无法更改，单片机的功能是可以编写程序进行更改的。事实上，由于单片机只是用在电子产品线路板上的一个集成电路芯片，完成一些常用的电气检测与控制功能，把它称为微型计算机太过夸大其词，于是又有人把它改名称为微控制器，英文名称：Micro Control Unit，缩写为 MCU，不管称为单片微型计算机还是微控制器或者 MCU，它本质上始终是用在电子产品线路板上的一个集成电路芯片，没什么神奇之处。

1.1.2 单片机的用途

单片机用途十分广泛，比如常见的家用电器洗衣机、空调、电磁炉等内部有单片机，现在的智能化仪器仪表内都有单片机，工业生产上的数控机床有位移检测用的光栅尺，光栅尺连接的控制仪表内就有单片机，作者设计过的在全国各地的国家粮食储备库与中央粮食储备库的计算机测温系统除计算机外的核心就是单片机，作者设计过的用在生产流水线检验家用热水器部件的检验设备和检验汽车部件的检验设备都是都是以计算机和单片机为核心构成的检验设备。

现在这个时代的电子产品，普遍都在使用单片机，所以学好单片机是非常重要的。

1.1.3 学习的典型芯片与 C 语言介绍

单片机种类较多，比较流行的有 51 单片机、AVR 单片机、PIC 单片机、MSP430 单片机、STM32 等，过去比较流行的 51 单片机典型型号是 AT89C51 与 AT89S51，现在已被功能更强大，使用更方便的 STC 单片机取代，STC 单片机对原有 51 内核进行了重大改进并增加了很多片内外设，第一代的 STC89 系列单片机性能就显著超越了 AT89 系列，又经历了几代发展，现在 STC 已发展到了 15 系列，具有低功耗、低价位、高性

能、使用方便等显著特点。STM32 是意法半导体公司使用 Cortex-M3 内核生产的 32 位单片机，运行速度更快，功能更强大，性价比高，现在运用也比较广泛，至于 AVR 单片机、PIC 单片机、MSP430 单片机等由价格高、供货渠道不稳定等多种因素，它们在市场的占有份额已经越来越小，所以学单片机重点要把 STC 和 STM32 学好，我这本书主讲 STC，把 STC 学精通后再学习 STM32 就很简单了，STC 单片机的例子几乎都可以用到 STM32 上。

STC15 系列单片机又分为多个子系列，STC15W100 /STC15F100W 系列→STC15W201S 系列 →STC15W401AS 系列→STC15W404S 系列→STC15W1K16S 系列→STC15F2K60S2 系列→STC15W4K32S4 系列等，它们的功能从简单到高级依次增强，由于芯片具体型号众多，不可能每一个都去学，本书主讲功能最强的 STC15W4K32S4 系列中的 IAP15W4K58S4，它的功能最全，15 系列中的其它型号功能都比它少，价格也更便宜，表 1-1 列出了 STC15 系列单片机典型型号与资源对比，IAP15W4K58S4 单片机兼容 STC15 系列其它型号单片机，在 IAP15W4K58S4 单片机上运行正常的程序不用任何修改就可以直接下载到同系列其它型号单片机上运行，在硬件上，IAP15W4K58S4 引脚排列也完全兼容相同封装的 15 系列其它型号，正因为如此，与本书配套的实验板除了可以做 IAP15W4K58S4 相关的实验外，也可以完成 15 系列其它型号单片机的实验，综上所述，我们只要学会了 IAP15W4K58S4，STC15 系列中其它型号芯片就都可以使用了。

表 1-1 STC15 系列单片机典型型号对比

型号	工作电压 (V)	Flash 程序存储器 字节	数据存储 SRAM 字节	定时器	PCA / PWM / CCP	6 通道带死区控制 PWM	串口数量	8 通道 10 位 ADC 转换器	SP I 接口	比较器	EEPROM	支持 USB 直接下载	支持外部晶振	参考价元
IAP15W4K58S4 (本身就是仿真器)	2.5-5.5	58K	4096	T0-T4	2 通道	有	4	有	主从	有	IAP	支持	5-35	5.9
STC15W4K56S4	2.5-5.5	56K	4096	T0-T4	2 通道	有	4	有	主从	有	2K	支持	5-35	5.9
IAP15F2K61S2 (本身就是仿真器)	4.5-5.5	61K	2048	T0-T2	3 通道	—	2	有	主	—	IAP	—	5-35	4.9
STC15F2K60S2	4.5-5.5	60K	2048	T0-T2	3 通道	—	2	有	主	—	1K	—	5-35	4.9
STC15W408S	2.5-5.5	8K	512	T0-T2	—	—	1	—	主从	有	5K	—	—	3.0

表格说明：

1. 型号为 IAP 开头的单片机可以在程序运行过程中由程序修改或者擦除整个 FLASH 程序存储区，让传统的只读程序存储器变成了可读写程序存储器，程序运行过程中写入 FLASH 的数据与程序一样，具有掉电不丢失的功能，表中 EEPROM 为 IAP 的表示 EEPROM 使用 FLASH 存储区剩余空间，型号不是 IAP 开头的单片机无论程序如何操作都是无法更改 FLASH 程序存储区的，使用 IAP 提高了程序的灵活性，不使用 IAP 有利于 FLASH 存储空间程序的安全性。

2. STC 单片机内部带有高精度 R/C 时钟，±1%温漂(-40℃~85℃)，通常的运用如串口通信、红外通信、18B20 通信类程序都是不需要外部晶振的，作为特殊运用，比如精密频率计需要外部晶振时(外部晶振频率稳定度通常都高于 0.01%，初始误差可通过调整与晶振连接的电容容量进行微调)，需要注意 15 系列的个别型号(比如 IAP15W4K61S4)目前只能外接 24MHz 的晶振，否则芯片可能无法正常工作，IAP15W4K58S4、IAP15F2K61S2、STC15F2K60S2 等都是可以使用外部 5-35MHz 晶振的。

学单片机除了要了解芯片内部功能模块外，还要学习编程语言，编程语言有汇编和 C 语言 2 种可供选择，汇编语言学习其实比 C 语言要简单，只要熟悉一下单片机的汇编指令，找几个简单的例子练一练就大致学会了，学习汇编语言还有个好处就是可以对单片机内部程序存储器与数据存储器部分的原理理解得比较清楚，C 语言本身也简单，只是学习的内容比汇编语言要多，也就是说，学 C 语言难度要略大于汇编语言，但是，汇编语言编写好的程序，别人是很难读懂的，就连自己编写的程序，隔上三五个月再看也是很难看懂的，C 语言就不同了，C 语言编写的程序比汇编语言程序容易理解，并且具有较强的移植性，一种单片机的代码可方便移植到另一种单片机上，更重要的一个问题，汇编语言编程水平不管高到哪里去，如果不精通 C 语言的话，还是不行的，因此本书主讲 C 语言。

1.1.4 本书配套实验板及相关学习工具介绍

本书配套了 2 个实验板，一个作为主实验板，外形如图 1-2 所示，可以完成流水灯、定时器/计数器、串口通信、I²C 通信、SPI 通信、按键、数码管、LCD1602 液晶、A/D 转换、D/A 转换、红外接收、DS18B20 温度传感器、TFT 工业彩色串口触摸屏等实验，另一个作为辅助实验板可直接插接到主实验板上，用于完成 SD 卡、MP3 播放器实验。使用配套实验板最大的好处是可以节省自己搭接实验电路的时间。

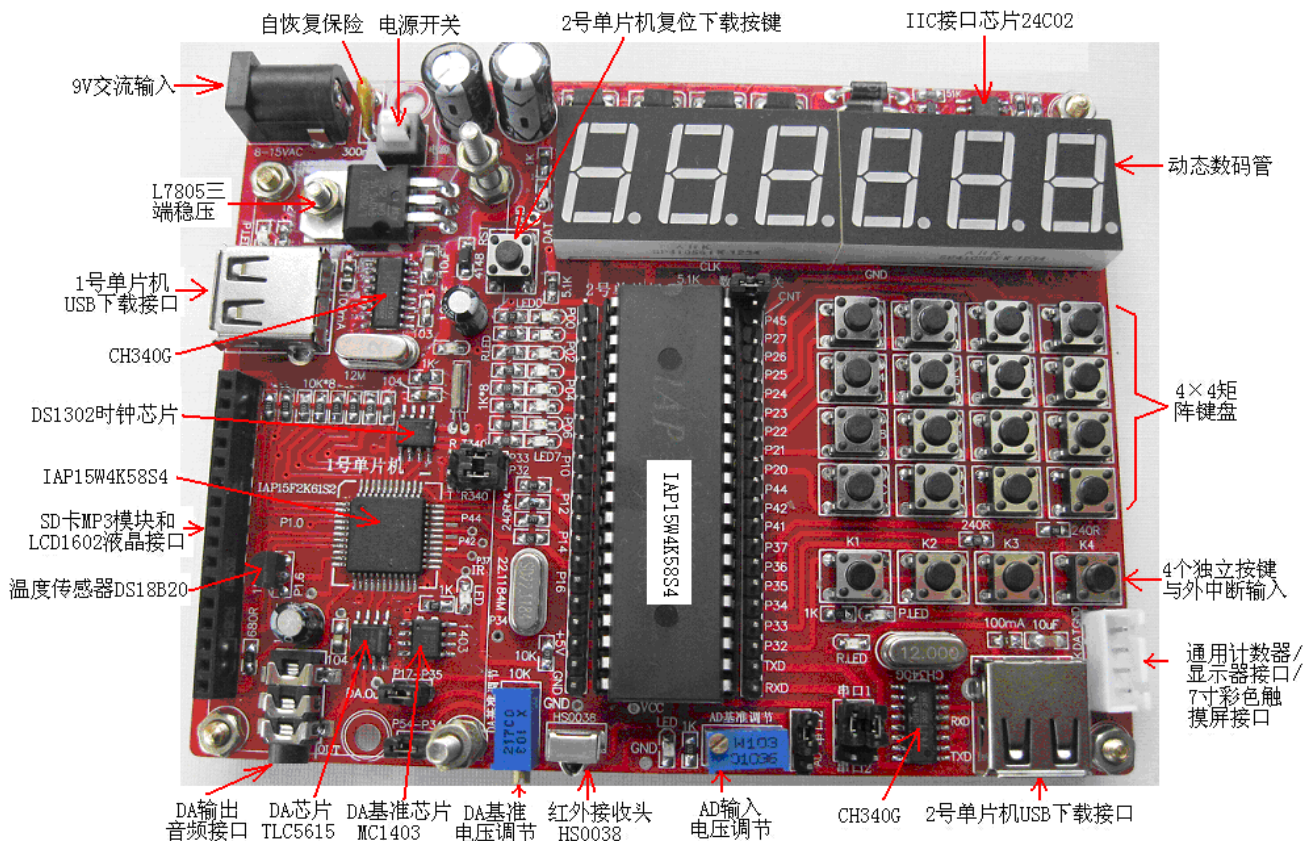


图 1-2 主实验板外形图

SD 卡与 MP3 辅助实验板外形如图 1-3 所示。

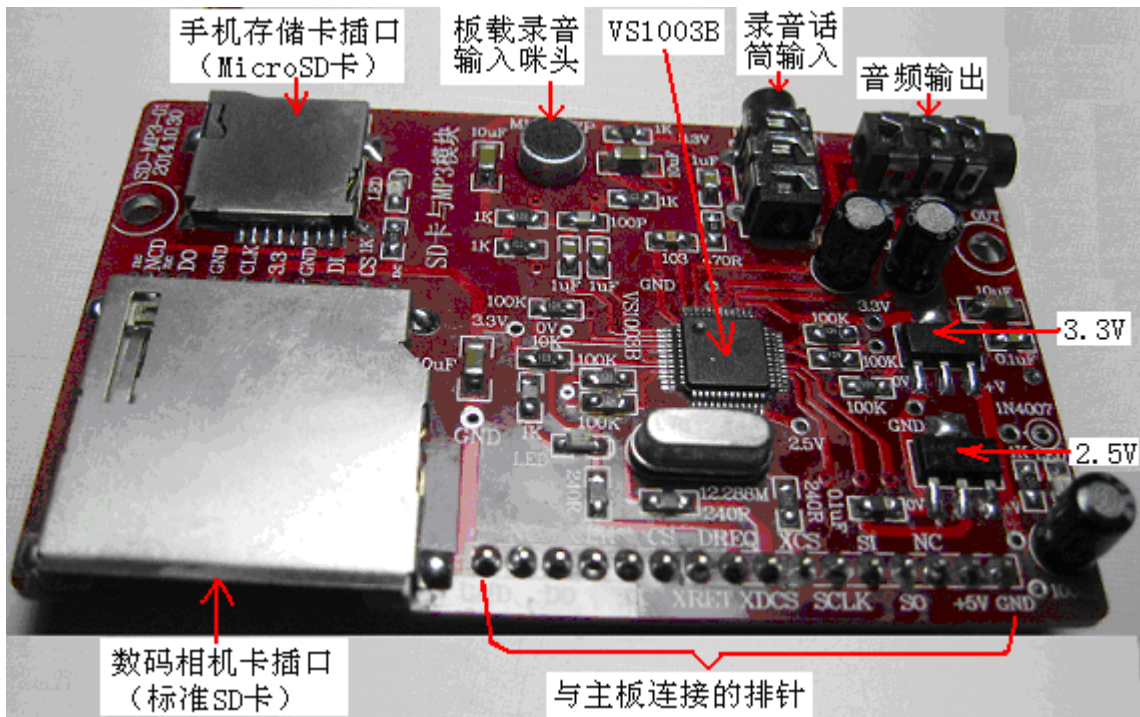


图 1-3 辅助实验板外形图

电路原理详细说明如下，熟悉电路图是编程与实验的重要基础，由于电路模块单元较多，可以在学习到相应章节时再回来仔细分析电路。

1、电源电路与EEPROM 断电检测电路如图 1-4 所示，图 1-4 有 2 路断电检测电路，一路是通过二极管 1N4007 全波整流采样交流电，适用于各个型号的单片，可靠性很高，可用于大量数据的断电瞬间存储，另一路是电源 VCC 与 GND 间的电阻串联分压值送入比较器输入口 P5.5，这种方式硬件更加简单，但只能用于内部带比较器的 STC15W 系列单片机。

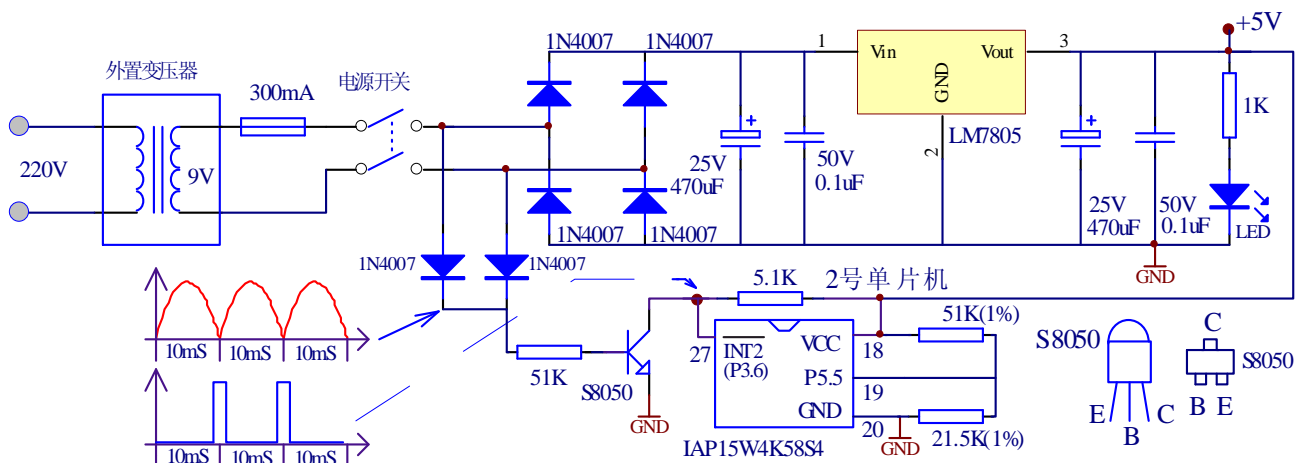


图 1-4 电源电路与 EEPROM 断电检测电路

2、双 CPU 电路如图 1-5 所示

本实验板采用双 CPU 电路，目的是要完成单片机与单片机之间高达 8MHz 的 SPI 数据通信实验，另外可以将一个 CPU 的输出脉冲作为计数源送入另一 CPU 完成计数器实验，采用多 CPU 方式还能够解决单片机 IO 口不足问题或两个高级中断谁也不能让谁的竞争问题。

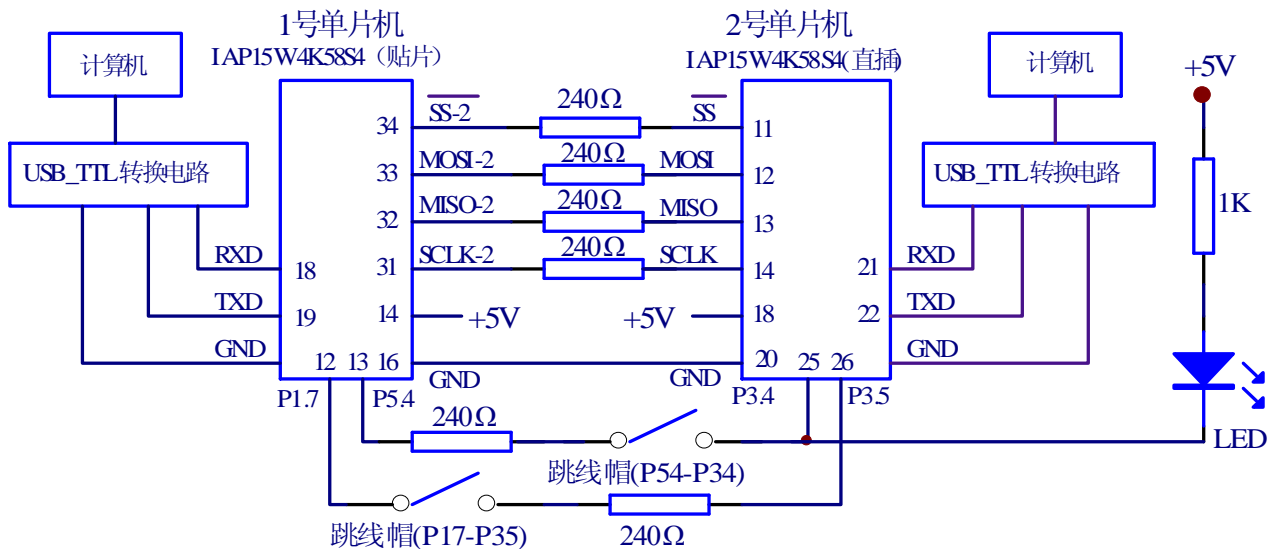


图 1-5 双 CPU 电路

3、2 号单片机晶振与复位电路图 1-6 所示

STC15 系列单片机内部 R/C 时钟精度高达 $\pm 1\%$ ($-40^{\circ} \sim +85^{\circ}$)，内部具有高可靠复位电路，因此一般情况下是不需要晶振与复位电路的，本实验板的 1 号单片机外部就完全没有这部分电路，为了做精确频率检测实验，2 号单片机使用了外部 22.1184MHz 晶振，外部晶振频率稳定度通常都高于 0.01%，初始误差可通过调整与晶振连接的电容容量进行微调，当然不是用示波器或频率计测量单片机晶振引脚频率后调整，示波器或频率计的接入相当于在晶振引脚并接电容到地，所以测量得到的频率误差会增大，可通过软件设置单片机在 P5.4 引脚输出主时钟，频率计测量这个位置得到的频率才是最真实的频率值。

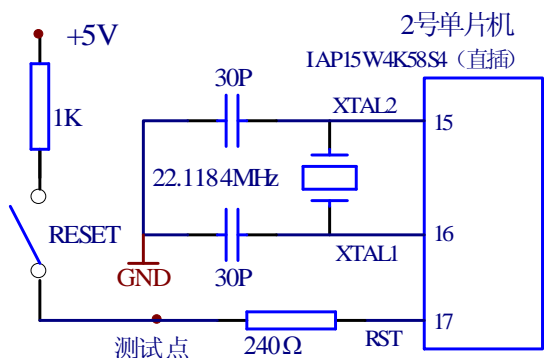


图 1-6 晶振与复位电路

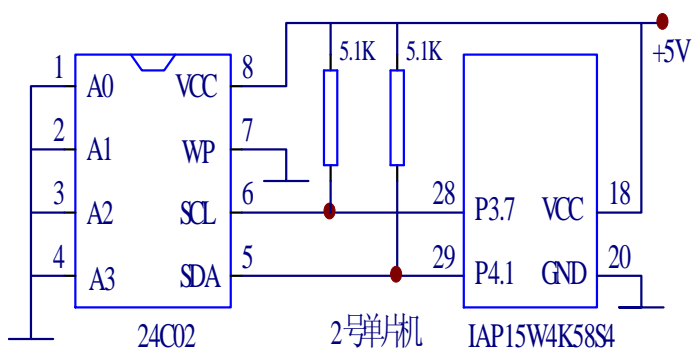


图 1-7 I²C 接口电路

4、2 号单片机与 I²C 器件 24C02 连接电路如图 1-7 所示。

5、2 号单片机串口下载与与双串口实验电路如图 1-8 所示。

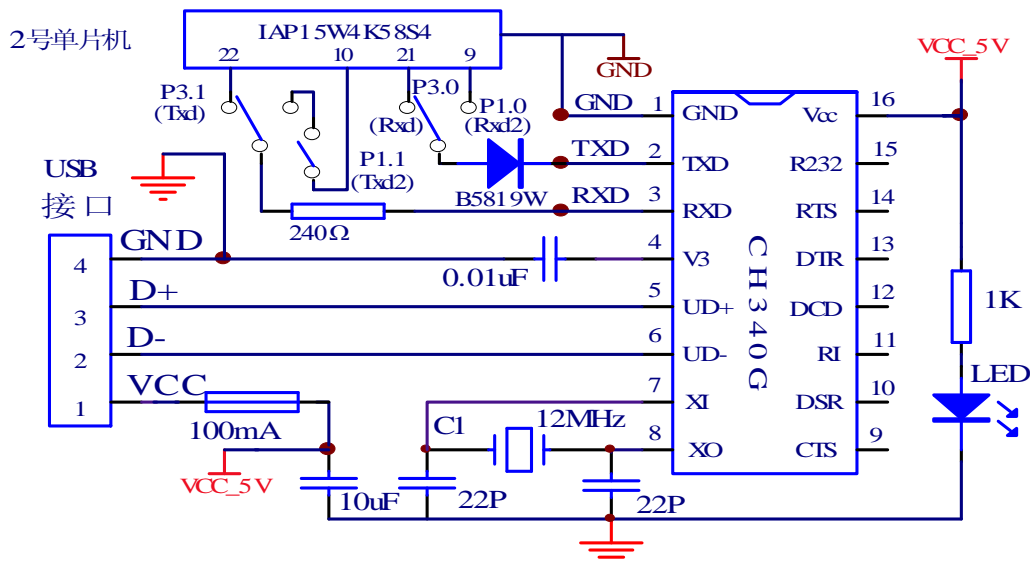


图 1-8 2 号单片机串口下载与双串口实验电路

6、2 号单片机数码管显示接口电路如图 1-9 所示

由于独立的 6 位一体数码管市场上很难购买，电路中采用的是 2 个完全相同的 3 位一体共阳型数码管。

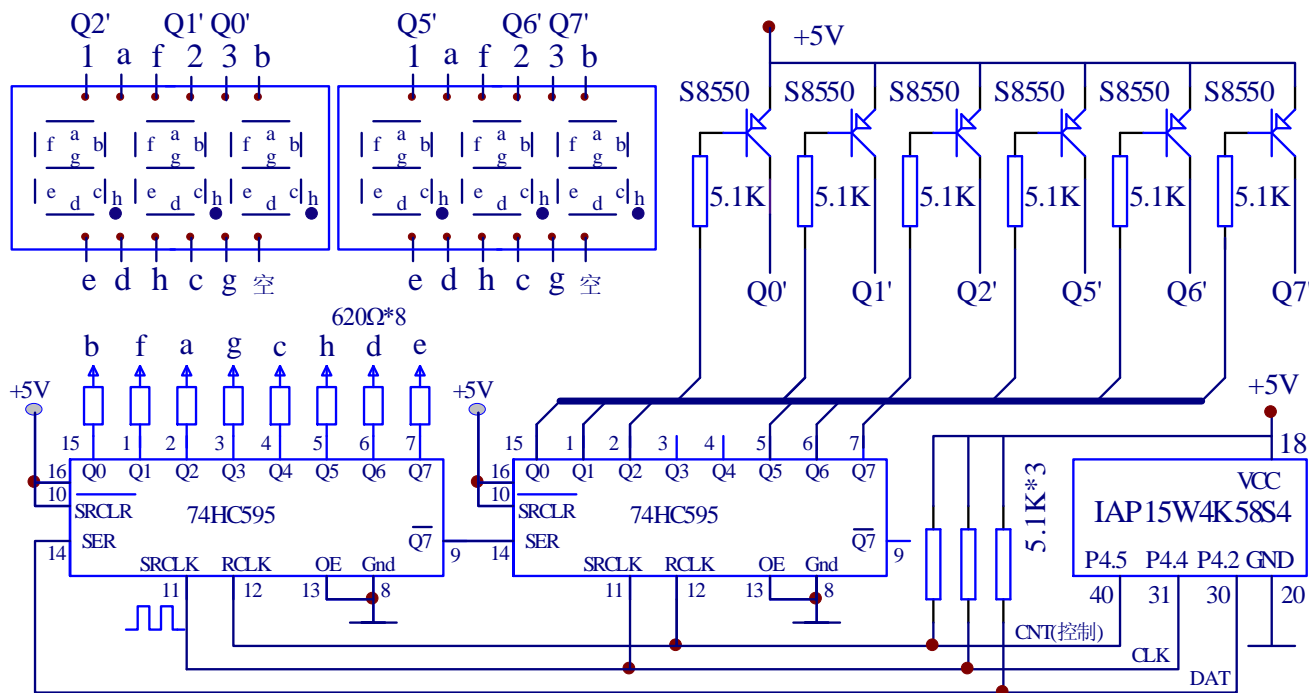


图 1-9 数码管显示接口电路

7、2 号单片机独立按键与外部显示器接口电路、矩阵按键电路等如图 1-10 所示。

与按键连接的单片机 I/O 口都串联了 240Ω 的电阻，用于防止软件设置错误（比如误设为强推挽）时单片机 I/O 口输出电流保护，外部显示器接口是通过 2 根信号线连接“电压表/电流表/计数器/显示器”模块，“电压表/电流表/计数器/显示器”模块的电路图将在第 14 章单独介绍，P3.4 与 P3.5 还用作计数器与频率计信号输入口，输入脉冲信号由 1 号单片机的 P5.4 与 P1.7 提供。

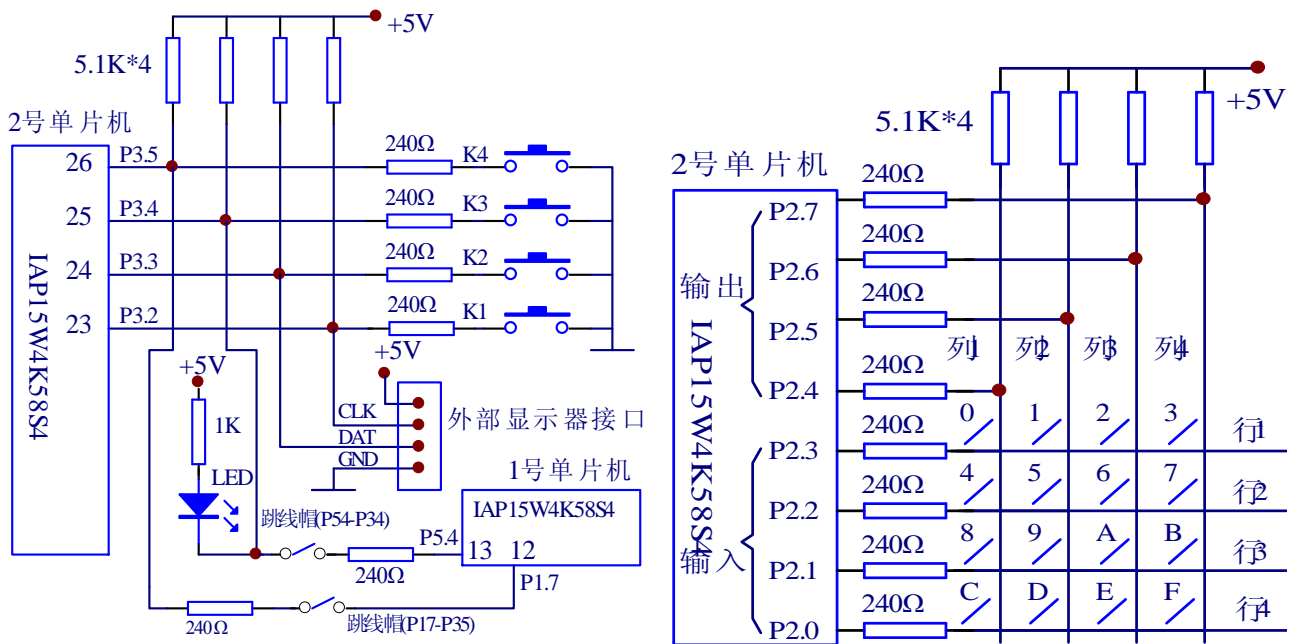


图 1-10 独立按键与外部显示器接口电路（图左）、矩阵按键电路（图右）

8、2号单片机 LED 指示灯电路图 1-11 所示。

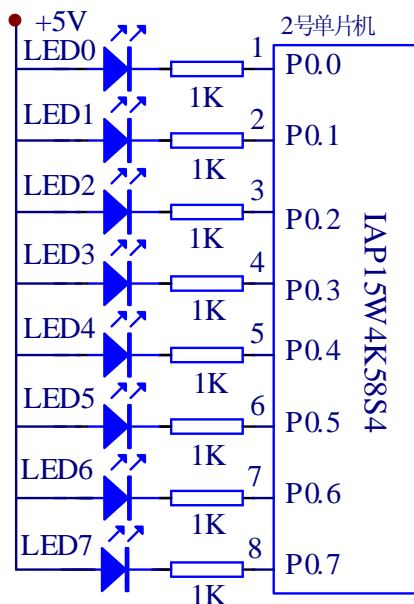


图 1-11 LED 指示灯电路

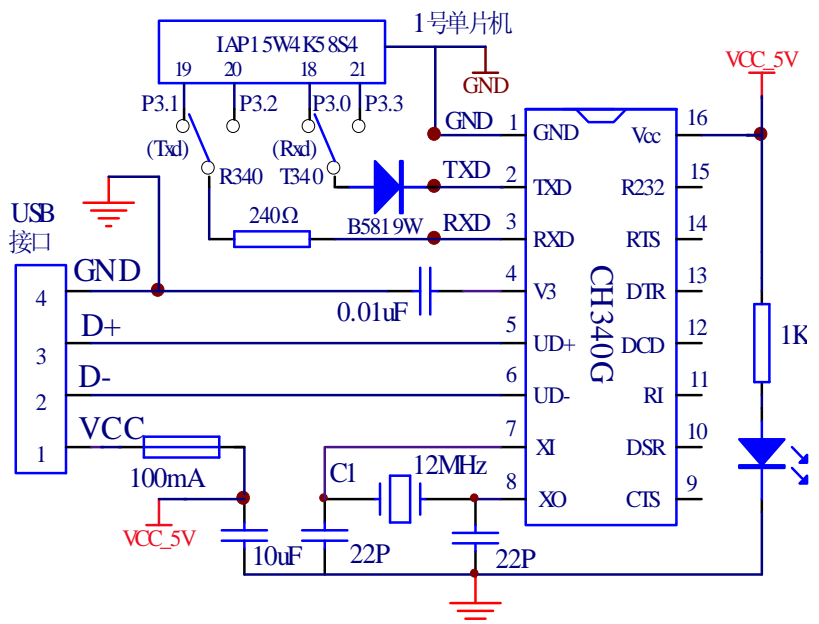


图 1-12 1号单片机串口下载与模拟串口实验电路

9、2号单片机 10 位 ADC 转换电路如图 1-13 所示，当电位器调到两端极限+5V 或 0V 时，如果程序中将 P1.1(ADC1)设为输出，将导致 I/O 口短路，串联 240Ω 电阻将把短路电流限制到单片机 I/O 口允许的 20mA 范围内，从而保护 I/O 口不损坏。

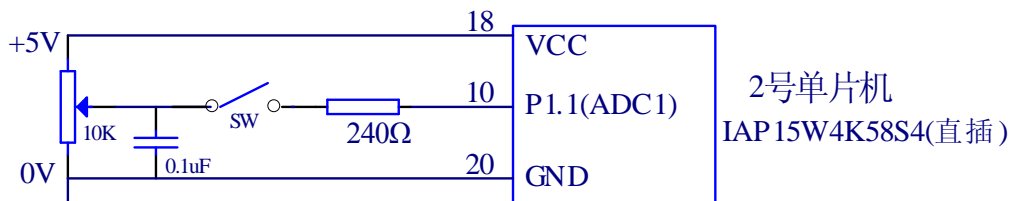


图 1-13 2号单片机 10 位 ADC 转换电路

10、1号单片机串口下载与模拟串口实验电路如图 1-12 所示。

11、1号单片机与 LCD1602 液晶显示器连接电路如图 1-14 所示，LCD1602 模块使用插针的方式直接插接到

主实验板对应的插座上。

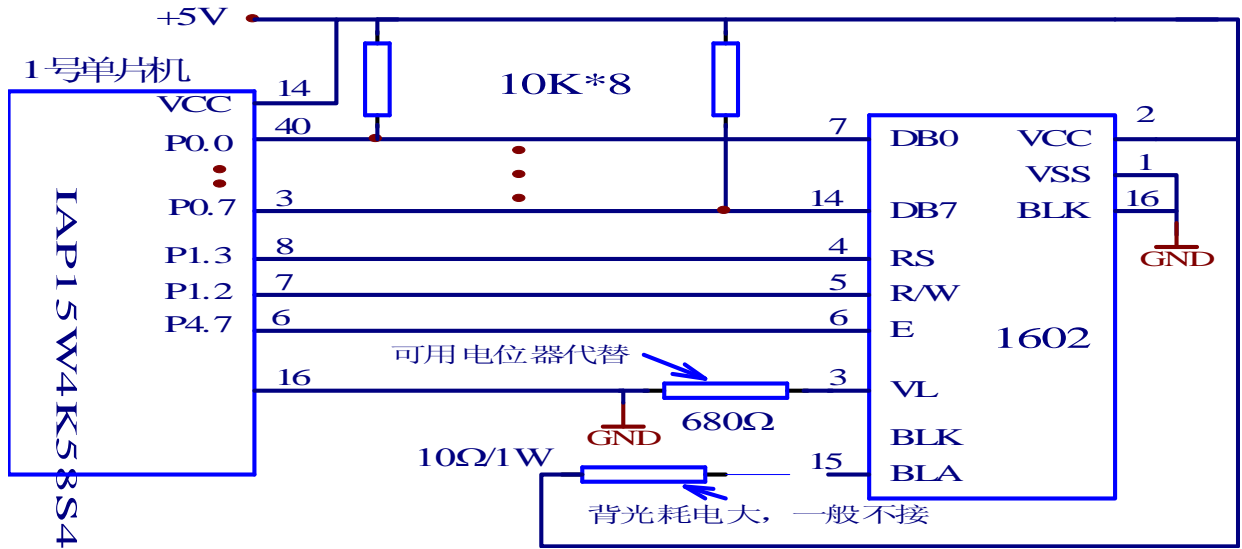


图 1-14 1 号单片机与 LCD1602 液晶显示器连接电路

12、1 号单片机与时钟芯片 DS1302 连接电路如图 1-15 所示。

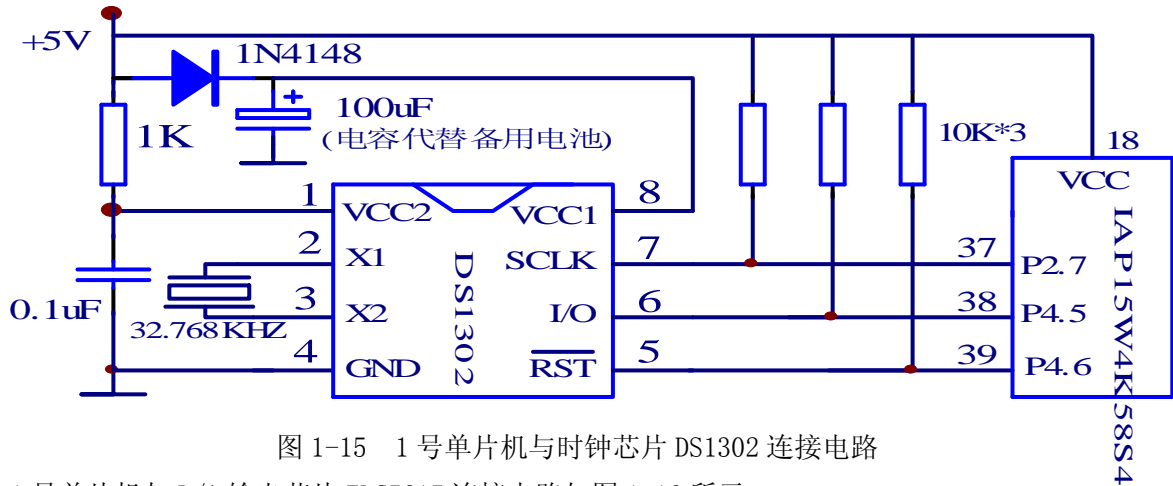


图 1-15 1 号单片机与时钟芯片 DS1302 连接电路

13、1 号单片机与 D/A 输出芯片 TLC5615 连接电路如图 1-16 所示

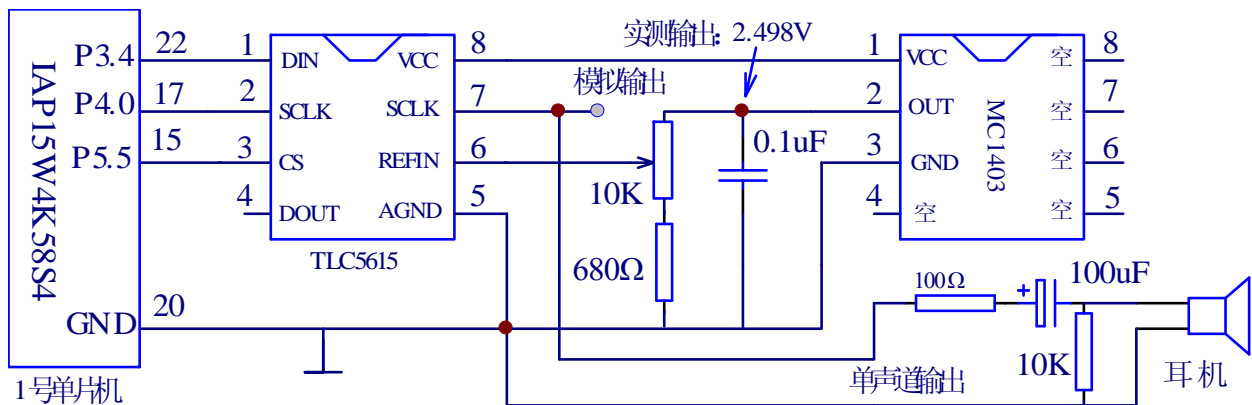


图 1-16 1 号单片机与 D/A 芯片 TLC5615 连接电路

14、1 号单片机温度检测与 LED 指示灯电路如图 1-17 所示



图 1-17 温度检测与 LED 指示灯电路

15、1 号单片机红外接收电路如图 1-18 所示

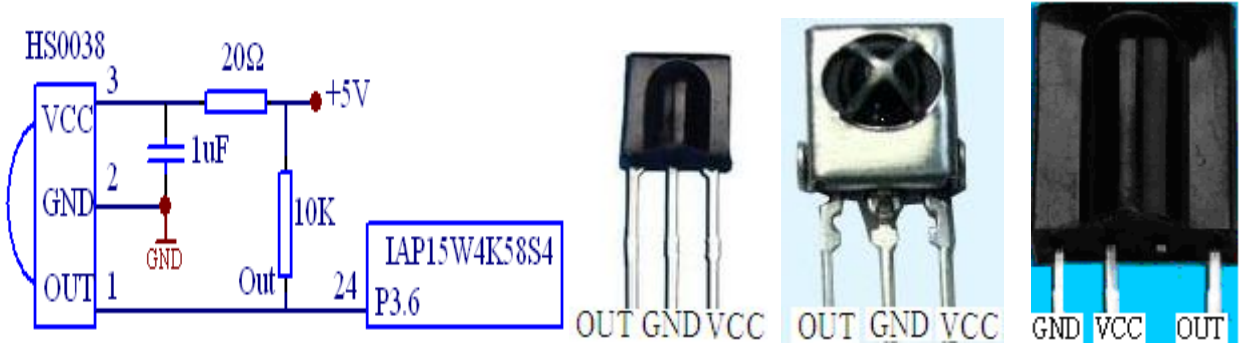


图 1-18 红外接收电路与红外接收头引脚定义

16、由 1 号单片机驱动的 SD 卡电路原理如图 1-19 所示，它与 MP3 部分的电路共同构成一个独立的电路板，使用插针方式直接插接到主实验板上的 LCD1602 液晶插座位置即可，由于单片机使用的是 5V 电压供电，IO 口输出电压也接近 5V，单片机驱动 3.3V 的 SD 卡，在程序中应将对应的单片机 IO 口设置为开漏输出方式。

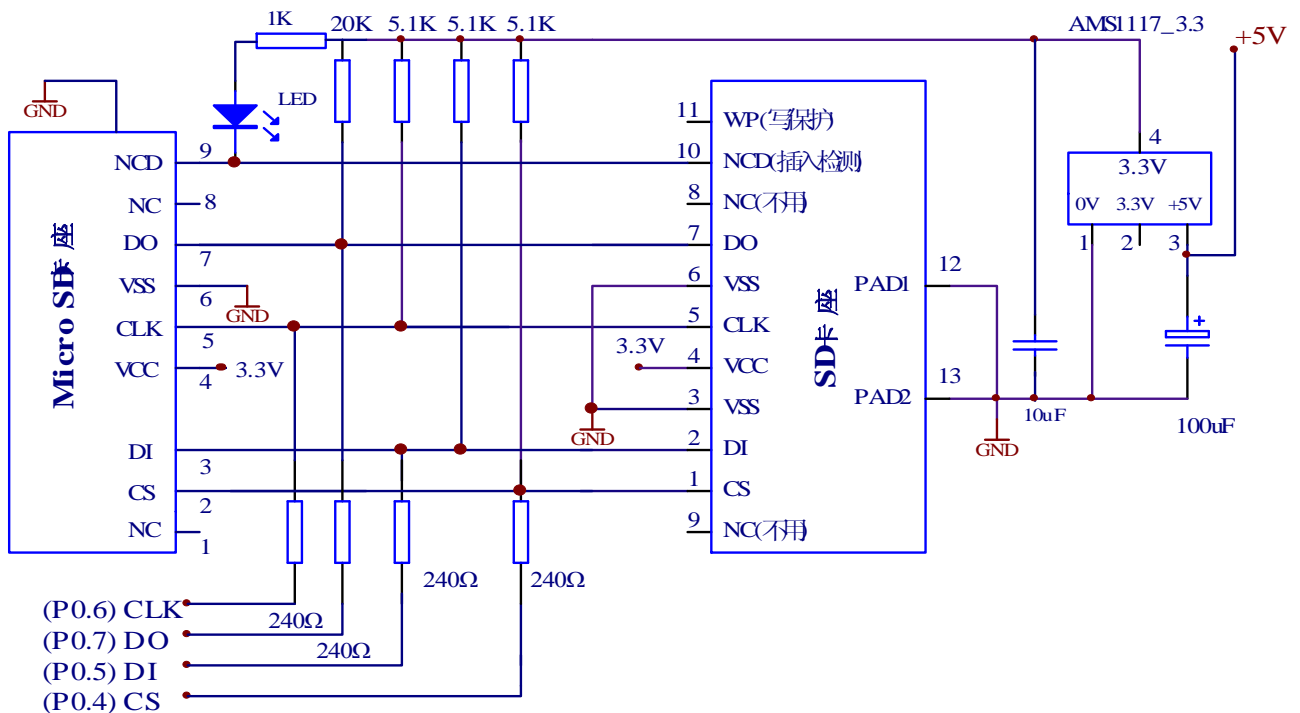


图 1-19 1 号单片机驱动的 SD 卡电路

17、由 1 号单片机驱动的 MP3 音频播放器模块电路原理如图 1-20 和图 1-21 所示。

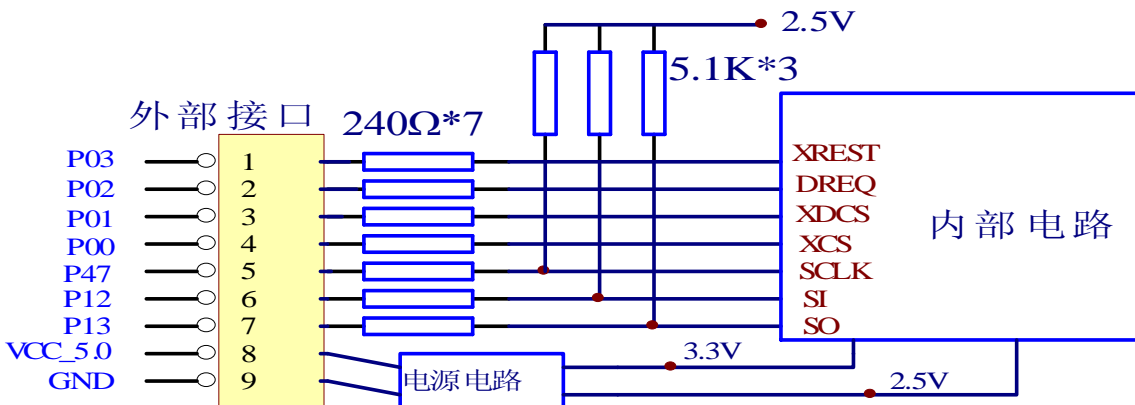


图 1-20 主实验板与 VS1003 模块接口

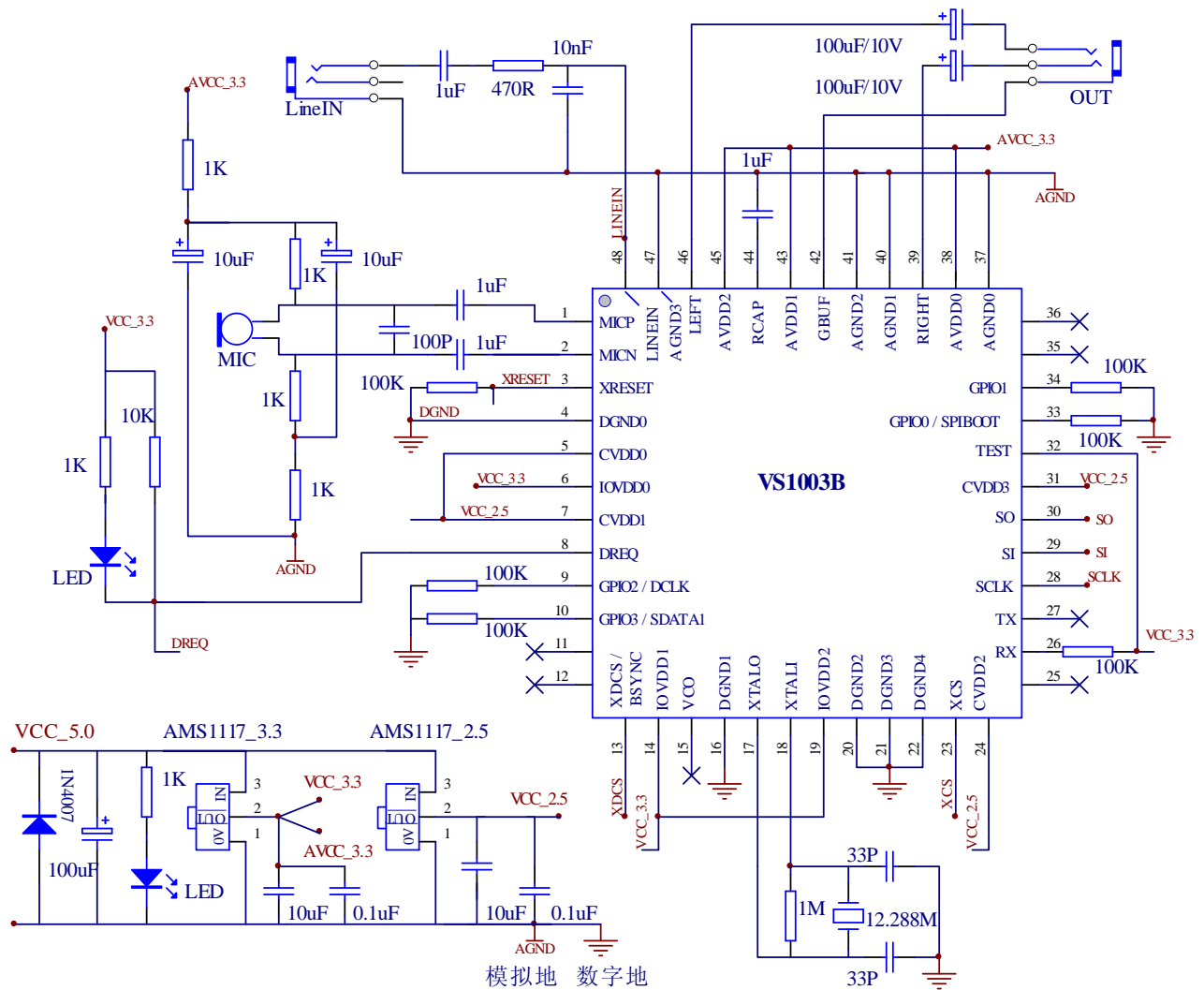


图 1-21 MP3 音频播放器模块内部电路

除开发板外还有其它几个常用工具：数字万用表、逻辑分析仪、数字存储示波器与计算机，数字万用如果没有现成的，建议购买“胜利 VC86E”或“胜利 VC97”，VC86E 直流电压精度比 VC97 更高，在做 A/D 转换实验时需要使用，VC97 的频率检测功能比 VC86E 更稳定，方便测量单片机输出信号频率，逻辑分析仪特别重要，初学时购买 24MHZ 采样率的就可以了，价格 100 元左右，外观与使用说明在本书最后章节有详细介绍，要想彻底明白书上的串口通信、SPI 通信、I²C 通信，没这个东西几乎是不可能的，不过也要提示一下，24MHZ 采样率的逻辑分析仪适合测量的信号频率在 1MHz 以内，信号频率过高的话，测出的波形将与实际不符，数字存储示波器建议选用 100M 带宽、4 通道并具有单次捕获功能的泰克示波器，示波器价格较高，有最好，没有也不影响本书实验，最后就是计算机，计算机配置要求并不高，但最好选用主板带 9 针 RS232 串口的，这样会省去很多麻烦。

1.2 点亮 1 个发光二极管

1.2.1 单片机型号命名规则

STC15 系列典型单片机型号命名规则如图 1-22 所示。

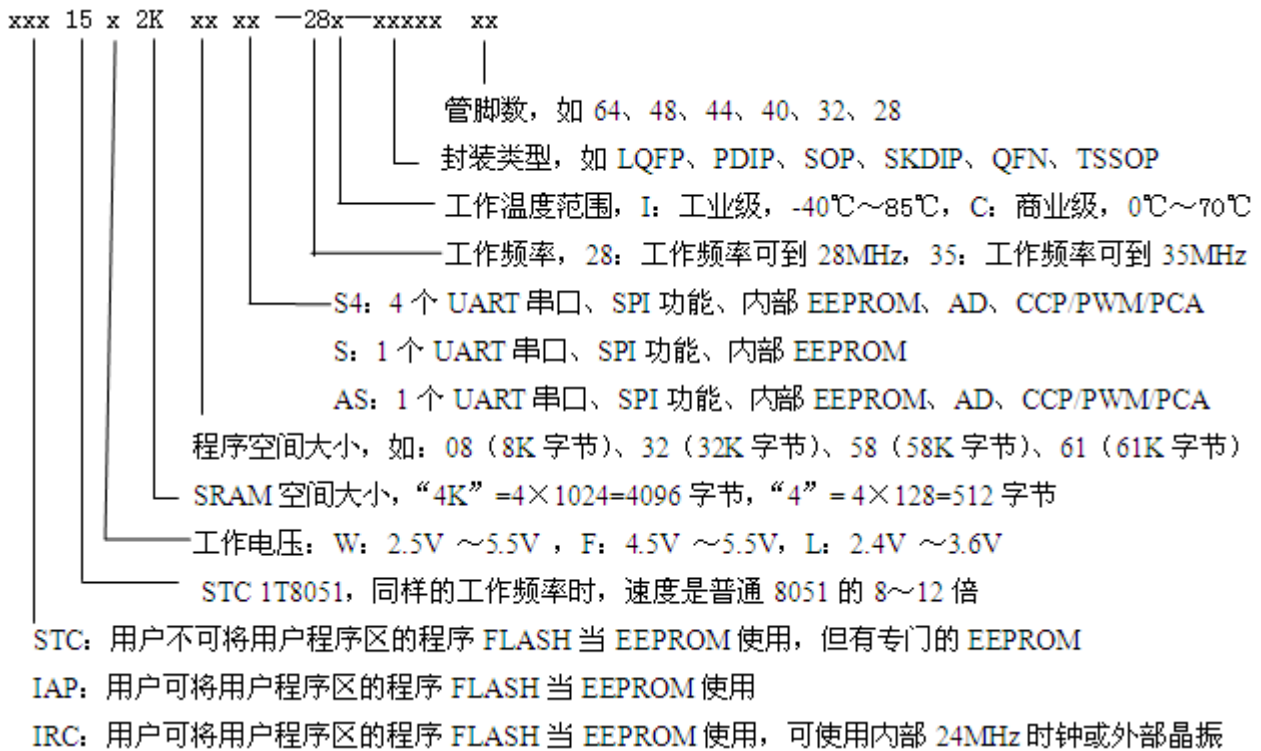


图 1-22 STC15 系列典型单片机型号命名规则

1.2.2 单片机引脚功能说明

1、封装图

IAP15W4K58S4 单片机有多种封装形式，最常用的封装如图 1-23、图 1-24、图 1-25。

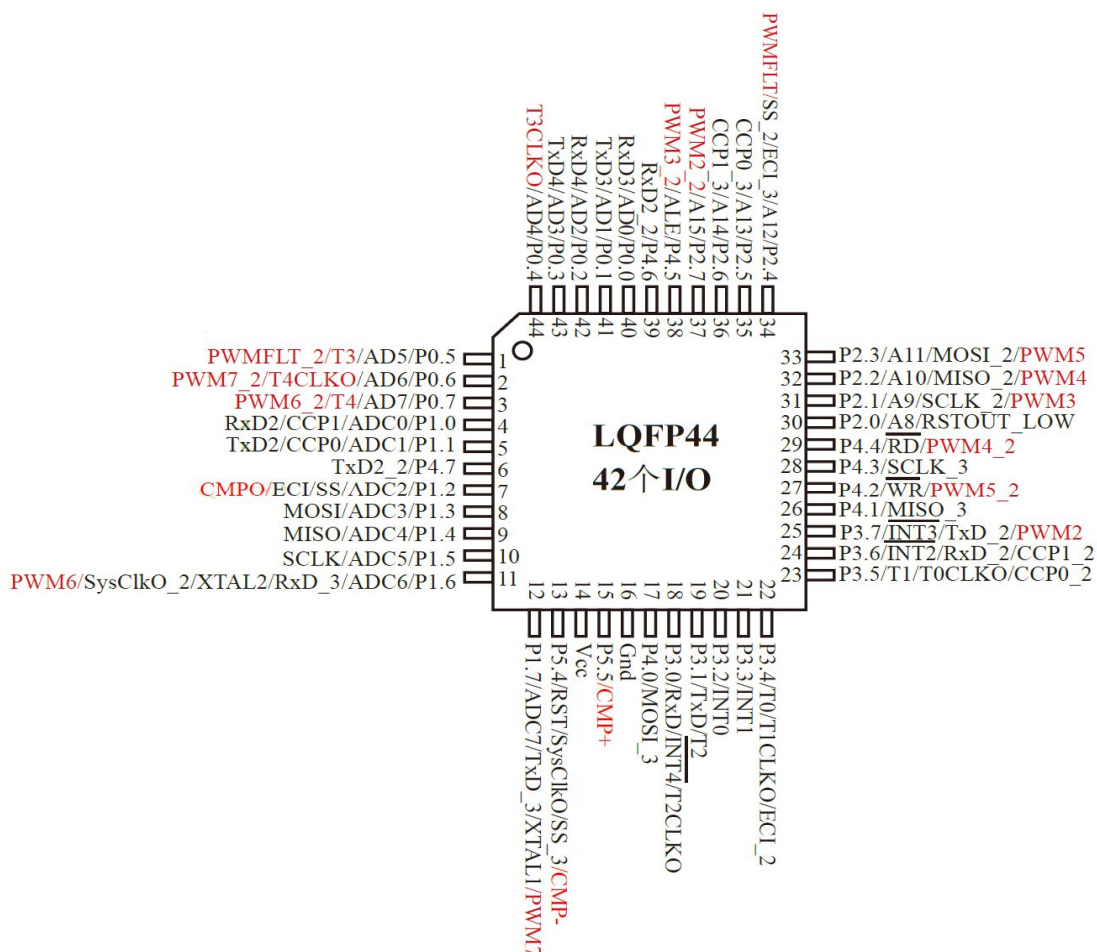


图 1-23 IAP15W4K58S4 单片机 LQFP-44 封装的引脚图

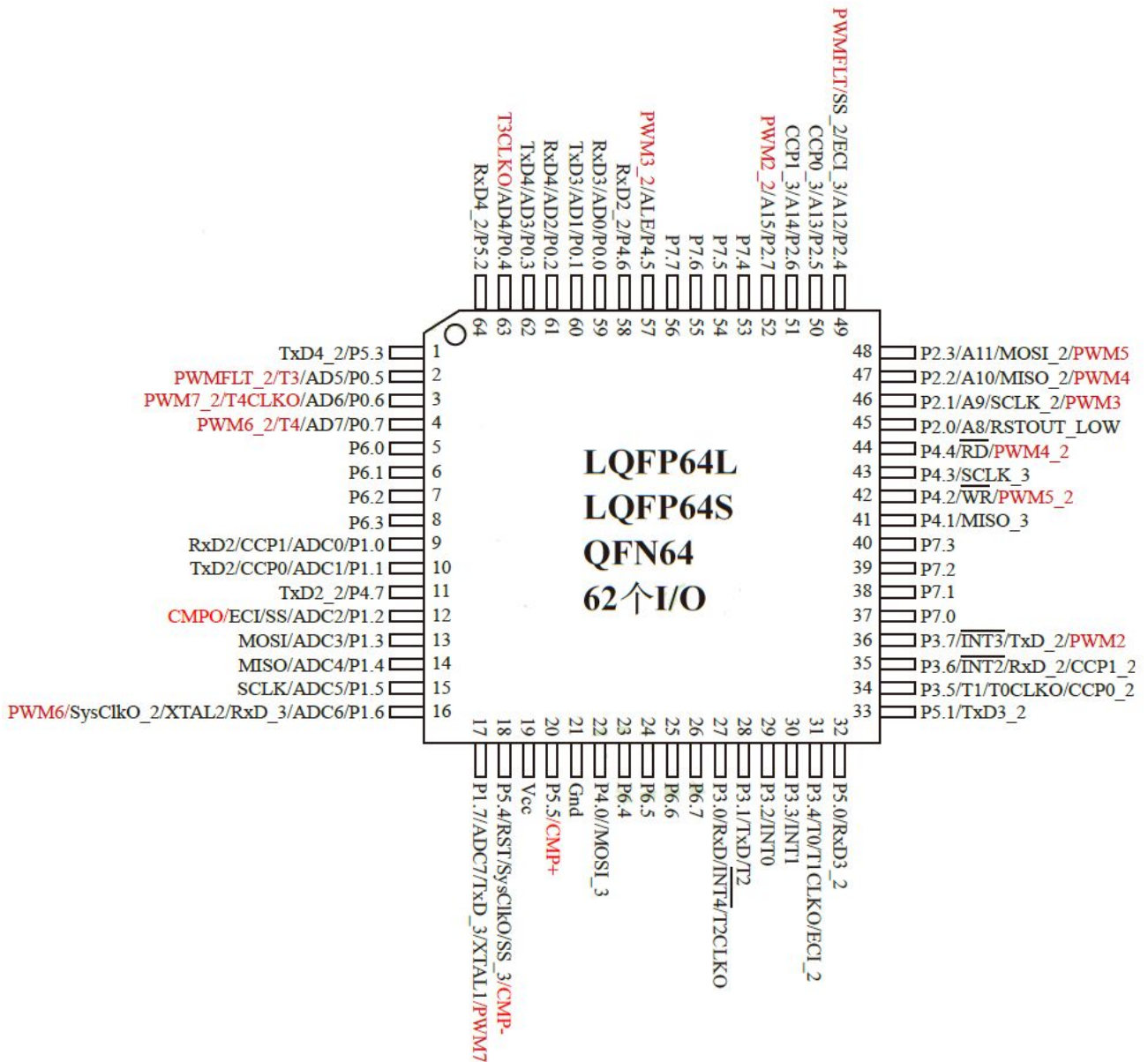


图 1-24 IAP15W4K58S4 单片机 LQFP64L/LQFP64S 封装的引脚图

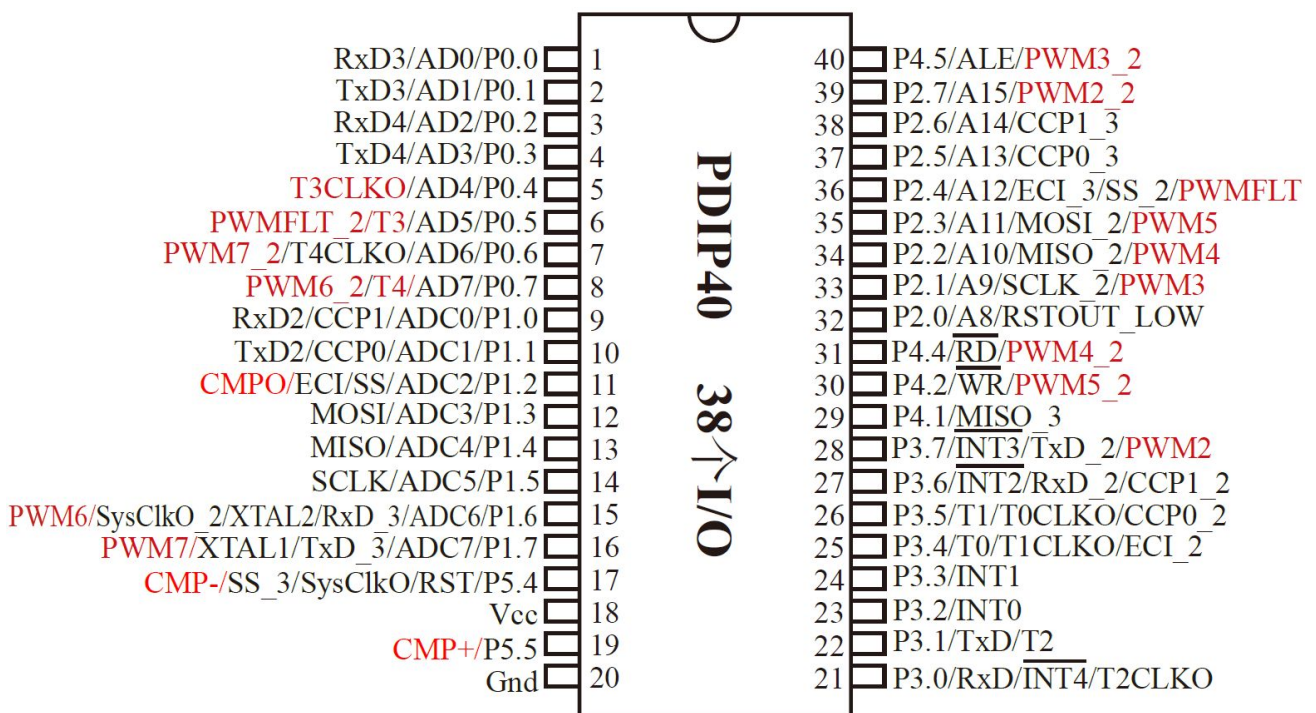


图 1-25 IAP15W4K58S4 单片机 PDIP-40 封装的引脚图

2、引脚功能说明

先说 PDIP40 封装引脚，除 18 与 20 脚用作电源引脚外，默认情况下，其余所有引脚都是数字输入输出 I/O 口，P4 口~P7 口的使用如同使用常规的 P0、P1、P2、P3 一样，并且都可以按位操作，I/O 口作为输入使用时，2.2V 以上时单片机认定为高电平，0.8V 以下时单片机认定为低电平，PDIP40 封装各引脚功能详细说明如下。

1~8 脚 P0 口，包括 P0.0~P0.7。

P0.0 还复用为 RxD3（串口 3 数据接收端）。P0.1 还复用为 TxD3（串口 3 数据发送端）。

P0.2 还复用为 RxD4（串口 4 数据接收端）。P0.3 还复用为 TxD4（串口 4 数据发送端）。

P0.4 还复用为 T3CLK0（定时器/计数器 3 的时钟输出）。

P0.5 还复用为 T3（定时器/计数器 3 的外部输入）与 PWMFLT_2（PWM 异常停机控制管脚）。

P0.6 还复用为 T4CLK0（定时器/计数器 4 的时钟输出）与 PWM7_2（脉宽调制输出通道 7）。

P0.7 还复用为 T4（定时器/计数器 4 的外部输入）与 PWM6_2（脉宽调制输出通道 6）。

在特殊情况下需要扩展外部数据存储器时，P0 口还可分时用作数据总线（D0~D7）与 16 位地址总线的低 8 位地址，P0 口到底是用作 I/O 口还是低 8 位数据/地址是不需要单独设置的，程序中如果是 I/O 操作命令，它就是 I/O 口，程序中如果是在执行访问外部数据存储器的命令，它就是 8 位数据/地址。

9~16 脚 P1 口，包括 P1.0~P1.7。同时复用为 8 通道模数转换器 ADC 输入口，STC15 系列 I0 口用作模数转换 ADC 时不需要对 I0 口输出状态作额外配置。

P1.0 还复用为 CCP1（捕获/脉冲输出/脉宽调制通道 1）与 RxD2（串口 2 数据接收端）。

P1.1 还复用为 CCP0（捕获/脉冲输出/脉宽调制通道 0）与 TxD2（串口 2 数据发送端）。

P1.2 还复用为 ECI（可编程计数阵列定时器的外部时钟输入）与 SS（单片机用作 SPI 从机时的从机片选输入控制端），P1.2 还复用为 CMP0（比较器的比较结果输出端）。

P1.3 还复用为 MOSI（SPI 主机输出从机输入）。

P1.4 还复用为 MISO（SPI 主机输入从机输出）。

P1.5 还复用为 SCLK（SPI 主机时钟输出或从机时钟输入）。

P1.6 与 P1.7 复用为外部晶振输入端口，若程序下载时勾选“选择使用内部 R/C 时钟”则 P1.6 与 P1.7 设置为普通 I0 口，不勾选“选择使用内部 R/C 时钟”则 P1.6 与 P1.7 设置为外部晶振输入端口，程序下载完毕后给单片机断电，重新上电后设置生效，P1.6 还复用为 RxD_3（串口 1 接收端备用切换引脚），P1.6 还复用为 MCLK0_2（主时钟输出备用切换引脚），P1.6 还复用为 PWM6（脉宽调制输出通道 6），P1.7 还复用为 TxD_3（串口 1 发送端备用切换引脚），P1.7 还复用为 PWM7（脉宽调制输出通道 7）。

17 脚 P5.4 口，若要用作外部复位引脚 RST，需在程序下载软件中设置，外部复位与内部的 MAX810 专用复位电路是逻辑或的关系，P5.4 还复用为 MCLK0，即可编程主时钟输出：无输出、输出主时钟、输出 0.5 倍主时钟、输出 0.25 倍主时钟，由于单片机所有 I/O 口对外允许最高输出频率为 13.5MHz，所以这里最高输出也不能超过 13.5MHz，主时钟指外部晶体振器频率或内部 R/C 时钟频率，P5.4 还复用为 SS_3（SPI 从机时的从机片选输入端备用切换引脚）与 CMP-（比较器负极输入端）。

18 脚 电源正，STC15W 系列使用 2.5~5.5V，STC15F 系列使用+4.5~5.5V，STC15L 系列使用 2.4~3.6V。

19 脚 P5.5，复用为 CMP+（比较器正极输入端）。

20 脚 GND。

21~28 脚 P3 口，包括 P3.0~P3.7。

P3.0 复用为 RxD（串口 1 数据接收端）、 $\overline{\text{INT4}}$ （外中断 4，只能下降沿中断）、T2CLK0（T2 时钟输出）。

P3.1 复用为 TxD（串口 1 数据发送端）、T2（定时器/计数器 T2 外部计数脉冲输入）。

P3.2 复用为 INT0 (外部中断 0 输入, 既可上升沿中断也可下降沿中断)。

P3.3 复用为 INT1 (外部中断 1 输入, 既可上升沿中断也可下降沿中断)。

P3.4 复用为 T0 (定时器/计数器 T0 外部计数脉冲输入)、T1CLK0 (T1 时钟输出)、ECI_2 (可编程计数阵列定时器的外部时钟输入备用切换引脚)。

P3.5 复用为 T1 (定时器/计数器 T1 外部计数脉冲输入)、T0CLK0 (T0 时钟输出)、CCP0_2 (捕获/脉冲输出/脉宽调制通道 0 备用切换引脚)。

P3.6 复用为 $\overline{\text{INT2}}$ (外部中断 2 输入, 只能下降沿中断)、RxD_2 (串口 1 数据接收端备用切换引脚)、CCP1_2 (捕获/脉冲输出/脉宽调制通道 1 备用切换引脚)。

P3.7 复用为 $\overline{\text{INT3}}$ (外部中断 3 输入, 只能下降沿中断)、TxD_2 (串口 1 数据发送端备用切换引脚)、PWM2 (脉宽调制输出通道 2)。

29 脚 P4.1, 复用为 MISO_3 (SPI 主机输入从机输出备用切换引脚)。

30 脚 P4.2, 复用为 /WR (扩展片外数据存储器的写控制端) 与 PWM5_2 (脉宽调制输出通道 5)。

31 脚 P4.4, 复用为 /RD (扩展片外数据存储器的读控制端) 与 PWM4_2 (脉宽调制输出通道 4)。

32~39 脚 P2 口, 包括 P2.0~P2.7, 在扩展外部存储器时作地址总线的高 8 位输出。

P2.0 复用为 RSTOUT_LOW 功能, 可通过程序下载软件设置上电复位后输出高电平还是低电平。

P2.1 复用为 SCLK_2 (SPI 时钟备用切换引脚) 与 PWM3 (脉宽调制输出通道 3)。

P2.2 复用为 MISO_2 (SPI 主机输入从机输出备用切换引脚) 与 PWM4 (脉宽调制输出通道 4)。

P2.3 复用为 MOSI_2 (SPI 主机输出从机输入备用切换引脚) 与 PWM5 (脉宽调制输出通道 5)。

P2.4 复用为 ECI_3 (可编程计数阵列定时器的外部时钟输入备用切换引脚)、SS_2 (SPI 从机时的从机片选输入端备用切换引脚)、PWMFLT (PWM 异常停机控制管脚)。

P2.5 复用为 CCP0_3 (捕获/脉冲输出/脉宽调制通道 0 备用切换引脚)。

P2.6 复用为 CCP1_3 (捕获/脉冲输出/脉宽调制通道 1 备用切换引脚)。

P2.7 复用为 PWM2_2 (脉宽调制输出通道 2)。

40 脚 P4.5, 复用为 ALE, 在扩展外部存储器时利用此引脚锁存低 8 位地址, 使 P0 口分时作地址总线低 8 位和 8 位数据总线, P2 口作地址总线高 8 位。P4.5 还复用为 PWM3_2 (脉宽调制输出通道 3)。

LQFP44 贴片封装比 PDIP40 插件封装多 P4.0、P4.3、P4.6、P4.7 引脚, 单独说明如下。

17 脚, P4.0 复用为 MOSI_3 (SPI 主机输出从机输入备用切换引脚)。

28 脚, P4.3 复用为 SCLK_3 (SPI 时钟备用切换引脚)。

39 脚, P4.6 复用为 RxD2_2 (串口 2 数据接收端备用切换引脚)。

6 脚, P4.7 复用为 TxD2_2 (串口 2 数据发送端备用切换引脚)。

LQFP64L/LQFP64S 封装比 LQFP44 封装增加的并且有复用功能的引脚说明如下。

32 脚, P5.0 复用为 RxD3_2 (串口 3 数据接收端备用切换引脚)。

33 脚, P5.1 复用为 TxD3_2 (串口 3 数据发送端备用切换引脚)。

64 脚, P5.2 复用为 RxD4_2 (串口 4 数据接收端备用切换引脚)。

1 脚, P5.3 复用为 TxD4_2 (串口 4 数据发送端备用切换引脚)。

3、I/O 口工作模式

IAP15W4K58S4 单片机所有 I/O 口都可由软件配置成 4 种工作模式之一: 准双向口 (标准 8051 单片机输出模式)、推挽输出、仅为输入 (高阻) 与开漏输出。每个口的工作模式由 2 个控制寄存器 (PnM1、PnM0) 中的相应位控制, 其中 n = 0、1、2、3、4、5、6、7, 例如 POM1 和 POM0 用于设定 P0 口, 其中 POM1.0 和 POM0.0 用于设置 P0.0, POM1.7 和 POM0.7 用于设置 P0.7, 依次类推, 设置关系如表 1-2 所示。STC15 系列中的 STC15W4K32S4 系列芯片, 上电后所有与死区控制专用 PWM 相关的 IO 口均为高阻态, 需将这些口设置为

准双向口或强推挽模式方可正常使用，相关 IO：P0.6/P0.7，P1.6/P1.7，P2.1/P2.2/P2.3/P2.7，P3.7，P4.2/P4.4/P4.5，其余 IO 口上电复位后都是 200uA 的弱上拉输出状态，可直接作输出口使用。

表 1-2 I/O 口工作模式

PnM1[7~0]	PnM0[7~0]	I/O 口工作模式
0	0	准双向口（标准 8051 单片机输出模式），灌电流可达 20mA，拉电流典型值为 200uA，由于制造误差，实际为 150~270 uA。
0	1	推挽输出，强上拉输出，可达 20 mA，外加限流电阻，尽量少用。
1	0	仅为输入（高阻）
1	1	开漏，内部上拉电阻断开，要外接上拉电阻才可以输出高电平。

例如，若设置 P1.7 为开漏，P1.6 为强推挽输出，P1.5 为高阻输入，P1.4、P1.3、P1.2、P1.1 和 P1.0 为弱上拉，则可使用下面的代码进行设置。

```
P1M1 = 0xa0; // 1010 0000B
```

```
P1M0 = 0xc0; // 1100 0000B
```

为了所有 IO 口都方便直接使用，可将所有 IO 口都配置为准双向口，函数代码如下。

```
void port_mode() // 端口模式
{
    POM1=0x00; POM0=0x00;P1M1=0x00; P1M0=0x00;P2M1=0x00; P2M0=0x00;P3M1=0x00; P3M0=0x00;
    P4M1=0x00; P4M0=0x00;P5M1=0x00; P5M0=0x00;P6M1=0x00; P6M0=0x00;P7M1=0x00; P7M0=0x00;
}
```

在使用单片机 I/O 口作灌电流输入或拉电流输出时，由于内部无电流限制功能，外部电路设计上一定要限制进出 I/O 口的电流不要超过 20 mA，另外，虽然 IAP15W4K58S4 单片机所有 I/O 口驱动能力都能达到 20 mA，但整个芯片的最大工作电流不要超过 120 mA，电流较大时可使用 74HC245、ULN2003 或三极管进行驱动。

1.2.3 制作一个最简单的单片机实验电路

这个实验电路很简单，但特别重要，即使你已经有了实验板，也建议你手工制作这个硬件电路，这样你才能体会到成功的喜悦，电路原理如图 1-26 所示。

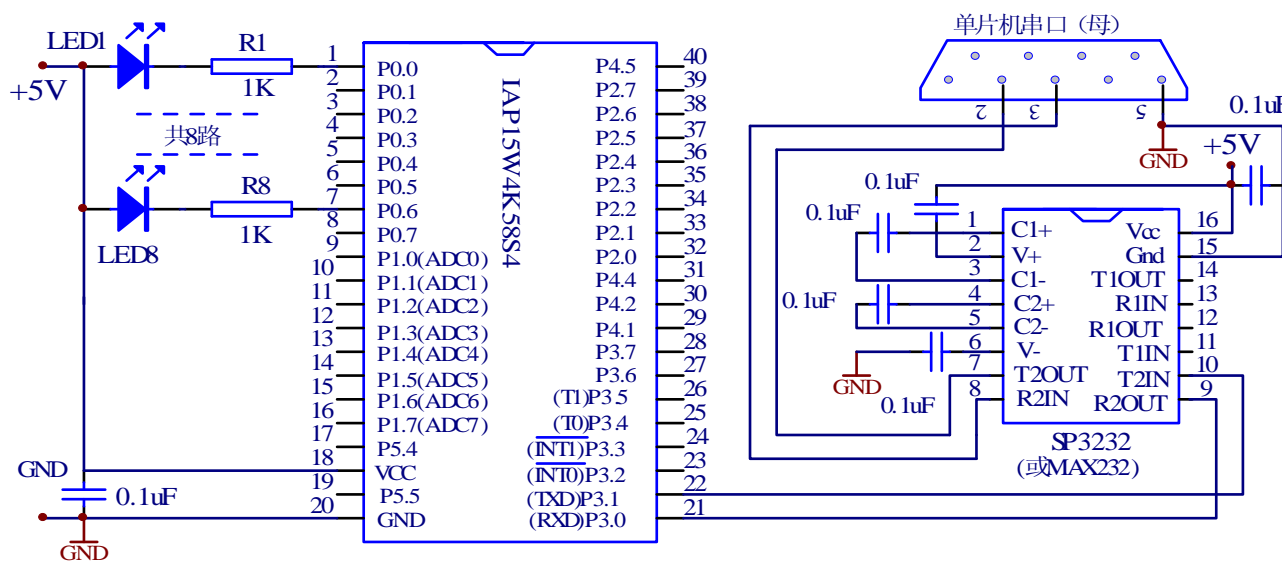


图 1-26 最简单的单片机实验电路

电路图左边需要输入 5V 直流电源，电源电压允许范围 3.0V-5.5V（单片机电压范围 2.5V-5.5V，SP3232EEN 电压范围 3V-5.5V），单片机 1 至 8 脚接 8 路发光二极管用于观察单片机程序运行结果，如果没有 5V 直流

电源，也可按图 1-27 制作，或者用 3 只 1.5V 的干电池串联成近似的 5V 电源。

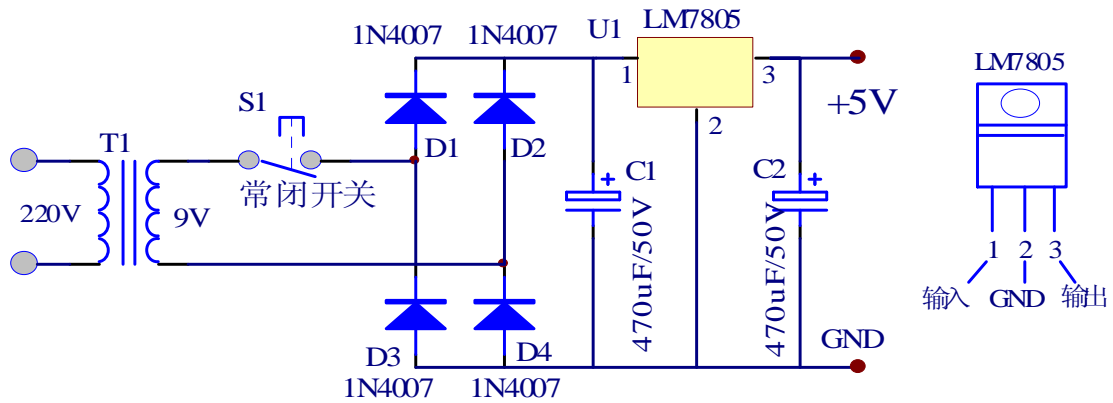


图 1-27 5V 直流稳压电源

检查电路制作无误后，将单片机串口母头插接到计算机 9 针串口公头上，RS232 串口公头与母头实物如图 1-28 所示，实物上的每一个引脚都是有编号的。



图 1-28 计算机串口公接头与单片机串口母接头

1.2.4 使用 Keil μ Vision3 环境编写最简单的程序

1、安装 Keil 软件

建议安装 Keil C51 V8.18，此版本功能齐全，使用稳定，如图 1-29 所示。



图 1-29 Keil C51 V8.18 安装图标

先在 C 盘新建一个文件夹并命名为“KEIL_818”，把安装程序路径修改为这个路径，安装过程非常简单，不需要输入系列号，在如图 1-30 所示界面的“First Name:”和“E-mail:”后面输入任意数字或字符即可，然后 Next 按钮就会变为可用状态，后面过程就不用再介绍了。

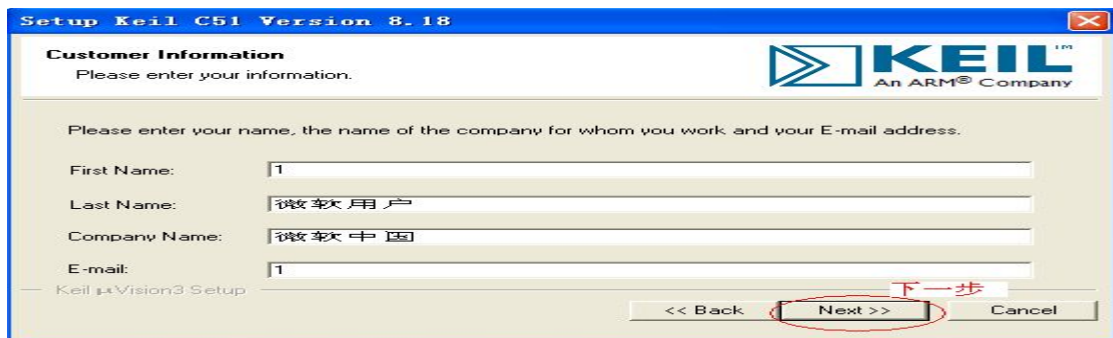


图 1-30 用户名与邮箱对话框

如果单片机需要处理中文数据，比如单片机串口需要向计算机串口发送汉字或单片机系统使用汉字显示屏或彩色显示屏，Keil 会过滤 0xfd 字符，使得部分汉字显示错误或出现乱码，这是 Keil 本身的一个错

误，最简单的解决办法是使用汉字补丁软件，将汉字补丁软件 cckeilv802.exe 放到你的 keil818\c51\bin 目录里，运行一下这个程序就可以了，如图 1-31 所示，支持 keil_8.02, keil_8.18, 其它版本没做过测试。



图 1-31 安装汉字补丁程序

2、输入代码并编译当前工程

在电脑某个位置新建一个文件夹并命名为“点亮一个发光二极管”，你也可随意取个名字，然后打开本书配套资源的任意一个例程，找到 STC15W4K.H 文件并复制到“点亮一个发光二极管”文件夹中，然后如图 1-32 所示，双击桌面“Keil μ Vision3”图标进入 Keil 软件。



图 1-32 双击进入 Keil 软件

刚进入 Keil 软件会打开一个 Keil 自带的文件，我们是不需要的，接着执行菜单 Project \rightarrow Close Project 关闭原有工程，然后执行菜单 Project \rightarrow New Project, 如图 1-33 所示。

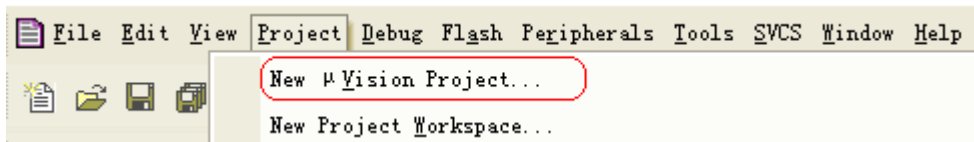


图 1-33 新建工程

弹出工程文件存放路径对话框，如图 1-34 所示，选择我们新建的文件夹“点亮一个发光二极管”，然后输入工程名，比如“led_light”，工程名最好是使用英文或汉语拼音，然后点“保存”按钮，不用管扩展名，自动进入下一步。

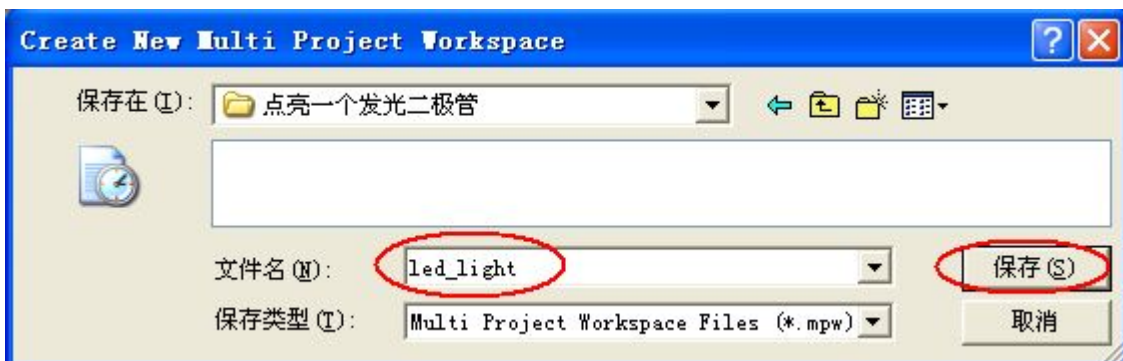
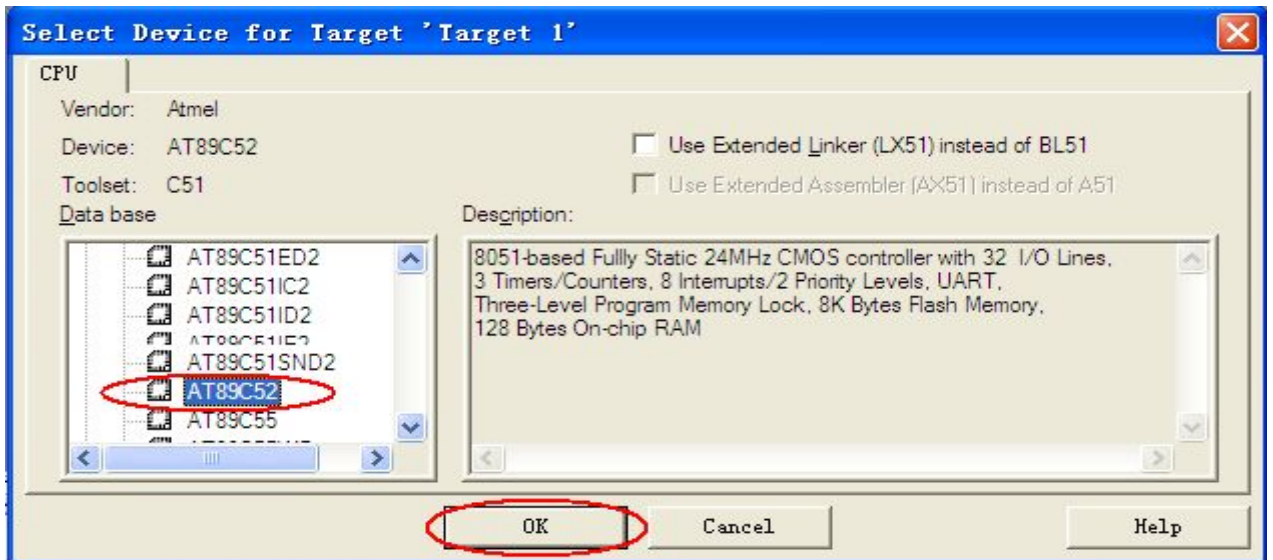


图 1-34 输入工程名

如图 1-35 所示，在窗口左边选择 Atmel 公司 AT89C52 后点“OK”确定，弹出图 1-36 对话框，选择“否”。



1-35 选择芯片型号



图 1-36 加载启动文件

如图 1-37 所示，单击新建文件图标新建一个空白文件，然后点保存图标存盘。

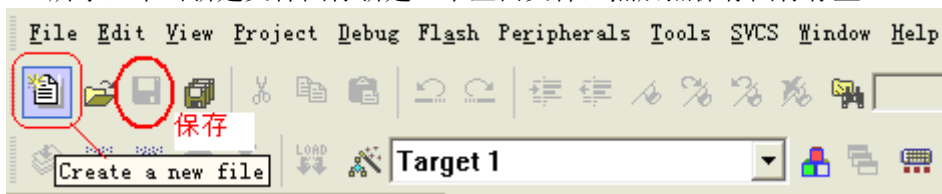


图 1-37 新建文件

如图 1-38 所示，输入文件名“led_light.C”，注意后缀名为.C，若是使用汇编语言编程，后缀名为.ASM，后缀名不用区分大小写，然后点“保存”按钮。

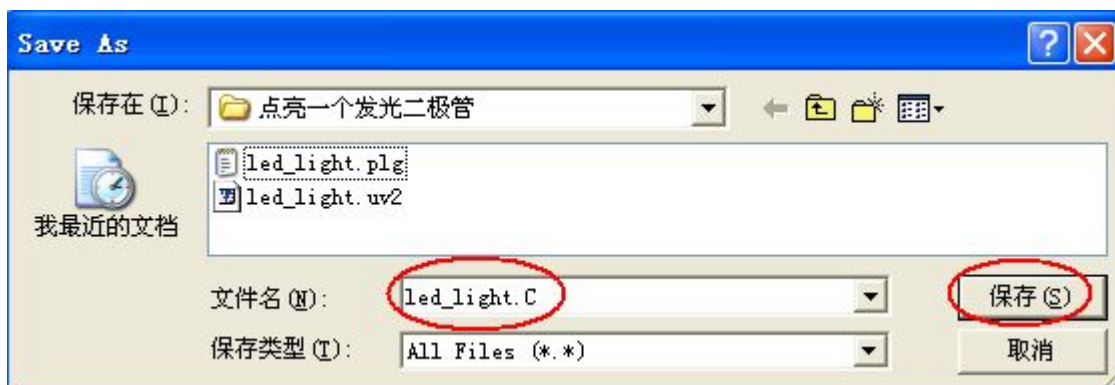


图 1-38 输入文件名

如图 1-39 所示，单击选中工程窗口 Target 1 下一层的 Source Group 1，右键选择 Add File to Group 'Source Group1'。

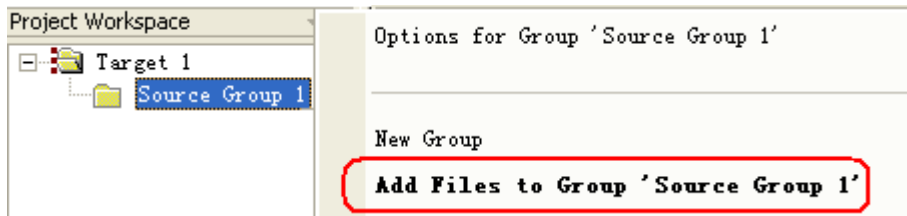


图 1-39 添加文件到工程

如图 1-40 所示，选择前面建立的” led_light.C”，然后点“Add”按钮，再点” Close”按钮，若多点了“Add”没关系，英文提示直接确定后再” Close”就行了，出现如图 1-41 所示的代码编辑窗口。



图 1-40 选择 C 文件

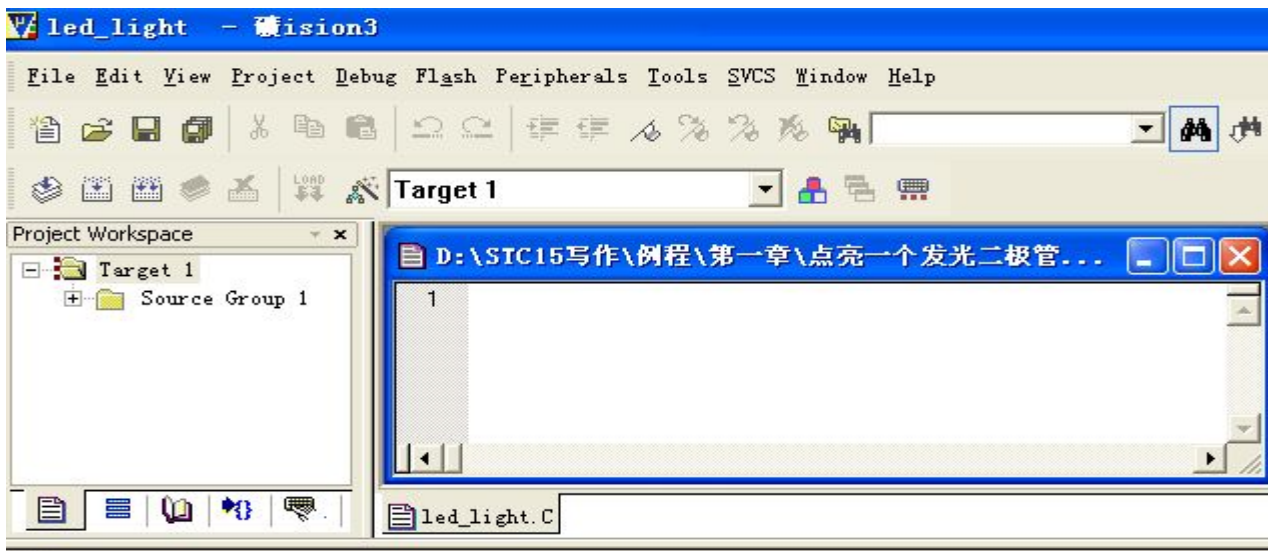


图 1-41 代码编辑窗口

如图 1-42 所示，在空白的文本编辑窗口输入代码（// 后面的注释可以不输入），然后保存，保存后点击设置工程的图标。

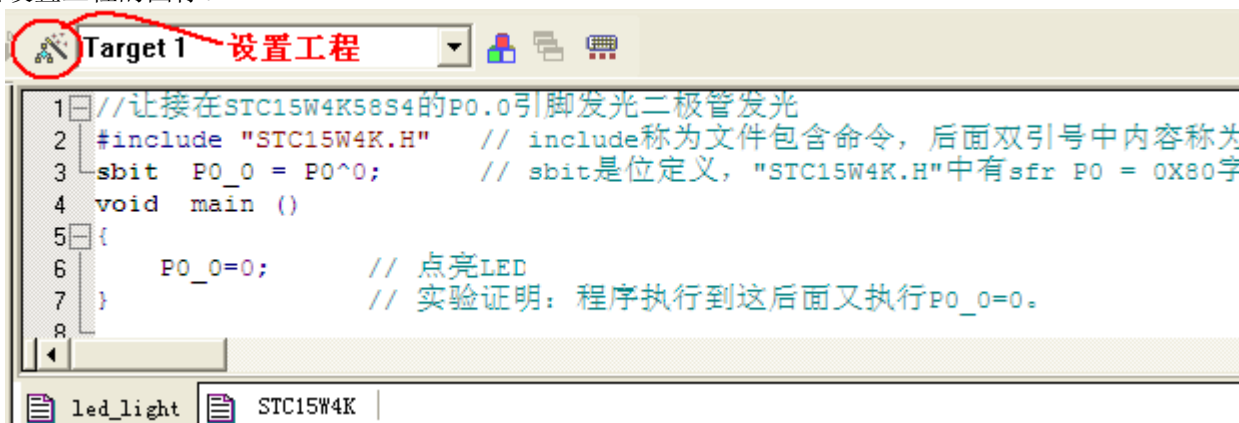


图 1-42 输入代码后的文本编辑窗口

进入设置工程窗口如图 1-43 所示，勾选 Create HEXFile，目的是得到最终下载到单片机中去的 HEX 目

标文件。



图 1-43 创建 HEX 文件

如图 1-44 所示，编译当前工程，编译提示“工程名”—0 Error, 0—Warning 就表示编译成功，data=9 表示程序占用内部数据存储器的大小为 9 个字节，IAP15W4K58S4 允许最大值为 256, xdata 表示程序占用外部数据存储器的大小，IAP15W4K58S4 允许最大值为 3840，特殊情况下通过硬件扩展允许最大值为 65536, code 是代码占用的程序存储器 Flash 空间，IAP15W4K58S4 允许最大值为 $58 \times 1024 = 59392$ 。若 data 超出 256, 编译结果会有错误提示，若 xdata 超出 3840, 不会出现错误提示，但会导致程序下载到单片机后出现运行功能错误，code 超出 59392 在程序下载时下载软件会有提示。

图 1-44 画红圈前面一个按钮 build target 用于编译当前 C 文件，画红圈的 rebuild all target files 用于编译工程中所有 C 文件，对于单文件程序，这两个都可编译文件，也没什么区别，当你编写大型多文件程序时，如果只修改了当前一个文件而其它文件并没有被修改，此时是不需要重新编译所有文件的，这样你用前一个按钮时，就只对当前修改过的文件进行编译，这样可以节省编译时间，如果修改过多个文件，就只能用第 2 个按钮，对所有文件全部重新编译一次。

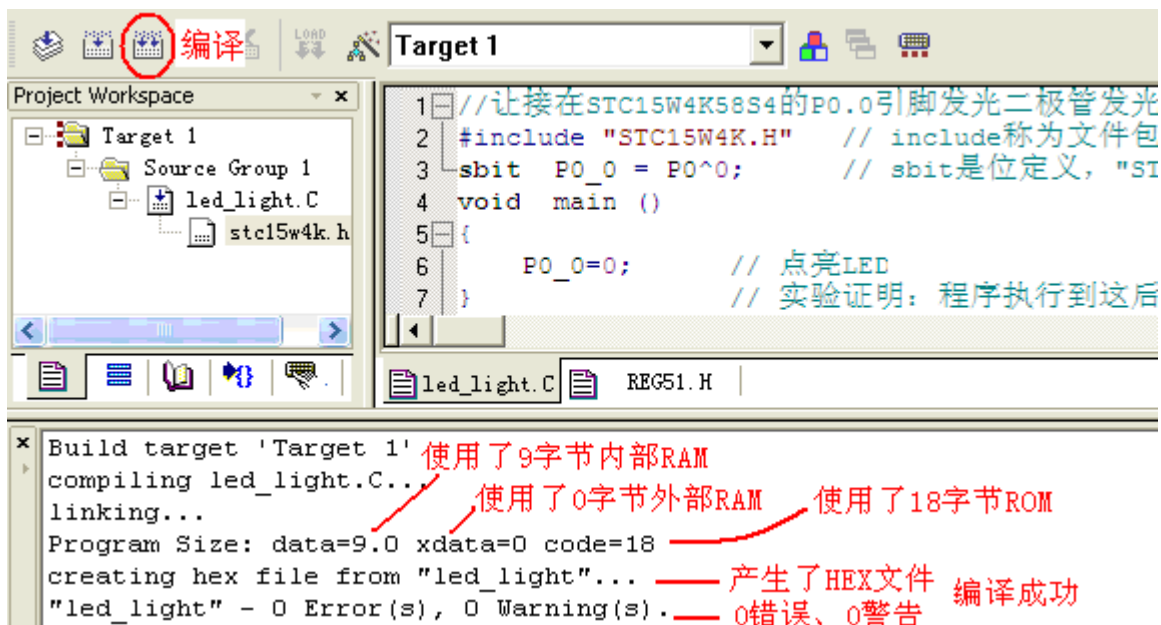


图 1-44 编译文件

1.2.5 ISP 下载程序到单片机（将电脑上的目标代码“灌入”单片机中运行）



双击 STC 系列单片机程序下载软件图标 `stc-isp-1...` 进入程序下载界面，如图 1-45 所示。首先需要选择单片机型号与计算机串口号，“打开程序文件”用于查找工程编译时生成的 HEX 文件，IRC 时钟频率最常用的就是 11.0592MHz，“当目标文件变化时自动装载并发送下载命令”一般都需要选中方便调试程序，然后单击“下载/编程”。

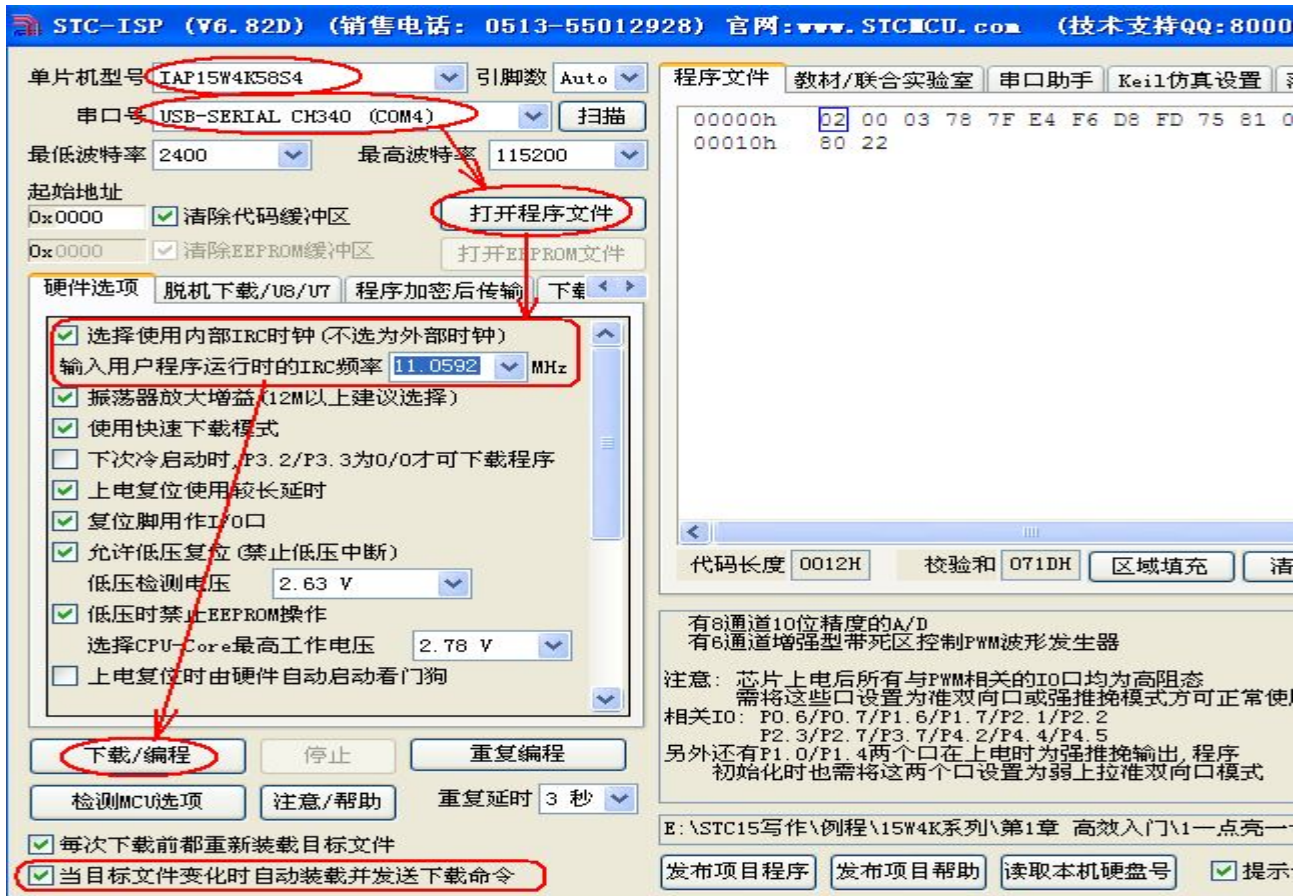


图 1-45 程序下载

出现图 1-46 所示界面后，给实验板上电，或是断一下电再上电，上电瞬间程序就开始向芯片下载了，下载完成后的界面如图 1-47 所示，需要注意的是：一定是先点“下载/编程”，后给单片机电路板上电，本实验下载成功后应该能看到电路板上单片机 1 脚连接的发光二极管被点亮了。



图 1-46 下载软件等待单片机上电

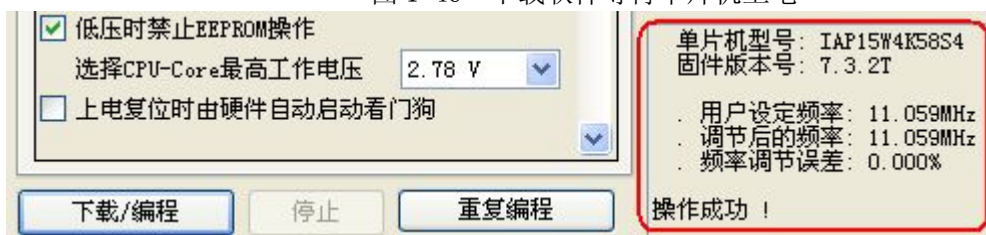


图 1-47 下载成功

1.2.6 程序解释

例 1.1 让接在 IAP15W4K58S4 的 P0.0 引脚发光二极管发光。完整代码如下：

```
#include "STC15W4K.H"    // include 称为文件包含命令，后面双引号中内容称为头文件
sbit P0_0 = P0^0;      // sbit 是位定义，“STC15W4K.H”中有 sfr P0 = 0x80 字节定义语句
void main ()
{
    P0_0=0;            // 点亮 LED
}                      // 实验证明：程序执行到这后面又执行 P0_0=0。
```

1、解释 #include "STC15W4K.H"

include 后面的双引号""表示让编译软件先到当前 C 文件所在目录中查找被包含的文件，若没找到则到 Keil\C51\INC 目录查找被包含文件，若使用尖括号 <>（电脑键盘上的小于大于符）则表示先到 Keil\C51\INC 目录查找被包含文件，若没找到再到当前 C 文件所在目录中查找被包含的文件，若使用 KEIL 自带的库函数对应的头文件（*.H 文件），则应该使用尖括号 <>，自己编写的头文件文件一般是放到当前 C 文件所在目录，这就应该使用双引号""，这样可减少程序编译时搜索文件的时间，#include "STC15W4K.H" 语句称为文件包含，相当于把被引用的文件中的所有内容复制一份到当前引用的位置，从表面上看，包含就是一行代码代替一大堆代码，在工程窗口展开 STC15W4K.H 可以看到一个内容较多的文件，我们先看下面 8 行代码。

```
sfr P0  = 0x80;    // 声明单片机 P0 口所在地址为 0x80,
sfr P1  = 0x90;    // 声明单片机 P1 口所在地址为 0x90
sfr P2  = 0xA0;    // 声明单片机 P2 口所在地址为 0xA0
sfr P3  = 0xB0;    // 声明单片机 P3 口所在地址为 0xB0
sfr P4  = 0xC0;    // 声明单片机 P4 口所在地址为 0xC0
sfr P5  = 0xC8;    // 声明单片机 P5 口所在地址为 0xC8
sfr P6  = 0xE8;    // 声明单片机 P6 口所在地址为 0xE8
sfr P7  = 0xF8;    // 声明单片机 P7 口所在地址为 0xF8
```

P0、P1、P2、P3、P4、P5、P6、P7 是单片机的 8 组 I/O 口，如图 1-24 所示，但是在 C 语言中，P0、P1、P2、P3、P4、P5、P6、P7 只能被看成符号，要想让这个符号与单片机硬件产生联系，就得使用上面格式的代码了，上面 8 条语句就是将单片机端口符号 P0、P1、P2、P3、P4、P5、P6、P7 与单片机内部硬件地址联系起来，程序运行时遇到这 8 个符号就去操作符号对应的硬件地址。注意 sfr 不是标准 C 语言的关键字，而是 KEIL C 编译器为了能够直接访问单片机内部的特殊功能寄存器提供的一个关键字，并且此关键字只能用于芯片内部特殊功能寄存器声明，语法格式：

sfr 符号 = 地址值； 符号一般使用特殊功能寄存器的名字，地址值是特殊功能寄存器的硬件地址。

如果单片机的某些功能模块在这个文件中找不到定义，就可以根据芯片手册找到功能模块对应的硬件地址，然后使用这个方法自己添加，添加完成后在代码中就可随心所欲操作增加的模块了，自己添加的模块与原有头文件中已有的模块在使用上效果上是完全一样的。

2、解释 sbit P0_0 = P0^0;

同样的道理，此语句将符号 P0_0 与 P0 口的第 0 位 P0.0 建立连接关系，如果我们同时操作一组 I/O 的所有硬件引脚，我们使用 sfr 定义的符号 P0、P1、P2 等就可以了，但如果只想控制 8 个硬件引脚其中的任意一个，我们就得使用这个命令了，此命令称为特殊功能位声明，最常用的格式：

sbit 位变量名 = SFR 名称^变量位地址值；

sbit 用于定义单字节（一个字节含 8 个位）可位寻址对象的某位，“单字节可位寻址对象”包括可位寻址特殊功能寄存器（比如 P0、P1、P2 等）和 RAM 中可位寻址区的 16 个字节。P0_0 是我们自己确定的符号，

你也可以写成 P00 或其它符号，只要方便看出是指 P0 口的第 0 位就可以了，P0^0 的 P0 是 sfr 已定义好的符号，表示 P0 口，^ 是电脑键盘数字 6 那个键对应的符号，是异或运算符，最后的 0 表示 P0 口的第 0 位。

3、解释 void main ()

每一个 C 语言程序有且只有这么一个主函数，函数后面一定有一对大括号“{}”，在大括号里书写其它程序。

4、解释 P0_0=0;

让 P0 口的 0 位输出低电平，因为硬件上是发光二极管正端接的+5V，负端通过 1KΩ 电阻连接单片机 P0_0 口，所以这句命令就可点亮二极管，点亮之后程序又如何执行呢？实际上，这个程序是有问题的，但 C 编译器及时发现了这个错误，为了不让单片机死机，让单片机执行了一条复位命令，复位后再次执行语句 P0_0=0；如此往复循环，所以我们看到 P0_0 连接的二极管一直是亮着的，严格的程序应该是这样的：

```
void main ()
{
    P0_0=0;    // 点亮 LED
    while(1);  // 程序停在这里不再向下运行，也可用 for (;;)代替，功能相同。
}
```

1.3 Keil 仿真

1.3.1 软件仿真（标准 8051 方式仿真，不能仿真单片机新增功能）

1、输入代码，先在 keil 文本编辑窗口输入例 1.2 所示的代码。

例 1.2 让接在 IAP15W4K58S4 的 P0.0 引脚发光二极管闪烁发光，R/C 时钟：11.0592MHz。

```
#include "STC15W4K.H" // include 称为文件包含命令，后面引号中内容称为头文件
sbit P0_0 = P0^0;    // sbit 是位定义，"STC15W4K.H"中有 sfr P0 = 0X80 字节定义语句
void delay500ms(void)
{
    unsigned char i,j,k;
    for(i=41;i>0;i--) // 注意后面没分号
    for(j=133;j>0;j--) // 注意后面没分号
    for(k=252;k>0;k--); // 注意后面有分号
}
void main ()
{
    for (;;) // for (;;) 让 for 下面 1 对大括号内程序无限循环
    {
        P0_0 =!P0_0; // 取反 P0_0 引脚
        delay500ms(); // 延时 500ms, 高电平 500ms, 低电平 500ms, 周期 1S
    }
}
```

2、进入调试环境

如图 1-42 所示，点击设置工程的图标进入设置工程窗口，弹出窗口选择 C51 选项卡，Code Optimization 代码优化等级默认值为 8，一般不要修改，若调试复杂程序出现不正常情况可将它设为 0，程序调试完成后再改成 8 提高程序执行效率并保证延时函数延时时间的准确性。Debug 选项卡左边一半用

于软件模拟调试，右边一半用于硬件仿真调试，软件调试当然要勾选 Use Simulator。

然后编译工程，编译正常后执行菜单“Debug”→ Start/Stop Debug Session 开始调试，等一两秒后光标会运行到 main 主程序中，然后可选择过程单步或单步执行程序。进入仿真窗口后若出现的不是前面的源代码窗口，而是夹有反汇编代码的窗口，直接单击工具栏放大图标关闭该窗口，如果程序编译结果没有错误，下次进入也就直接进入源代码窗口了，进入调试窗口后工具栏很多按钮都变得可操作了，如图 1-48 所示。



图 1-48 调试窗口工具栏

从左到右依次是复位、全速运行、暂停、单步（进入函数内部）、过程单步（一步执行完整整个函数）、若在子程序或函数内部执行完当前函数、运行到当前光标所在行、下一状态、打开跟踪、观察跟踪、反汇编窗口、观察窗口、代码作用范围分析、1# 串行窗口、内存窗口、性能分析、模拟逻辑分析仪、工具按钮等命令。

3、单步与过程单步

学习程序调试，必须明确两个重要的概念，即单步执行与全速运行，全速执行是指一行程序执行完以后紧接着执行下一行程序，中间不停止，这样程序执行的速度很快，并可以看到该段程序执行的总体效果，即最终结果正确还是错误，但如果程序有错，则难以确认错误出现在哪些程序行。单步执行是每次执行一行程序，执行完该行程序后立即停止，等待命令执行下一行程序，此时可以观察该行程序执行完以后得到的结果，是否与我们写该行程序所想要得到的结果相同，借此可以找到程序中问题所在位置。程序调试中，这两种运行方式都要用到。

使用菜单 Debug→ Step 或相应命令按钮的快捷键 F11 可以单步执行程序，使用菜单 Step Over 或快捷键 F10 可以过程单步形式执行命令，所谓过程单步，是指将汇编语言中的子程序或高级语言中的函数作为一个语句来全速执行，图 1-49 是实际的调试窗口。

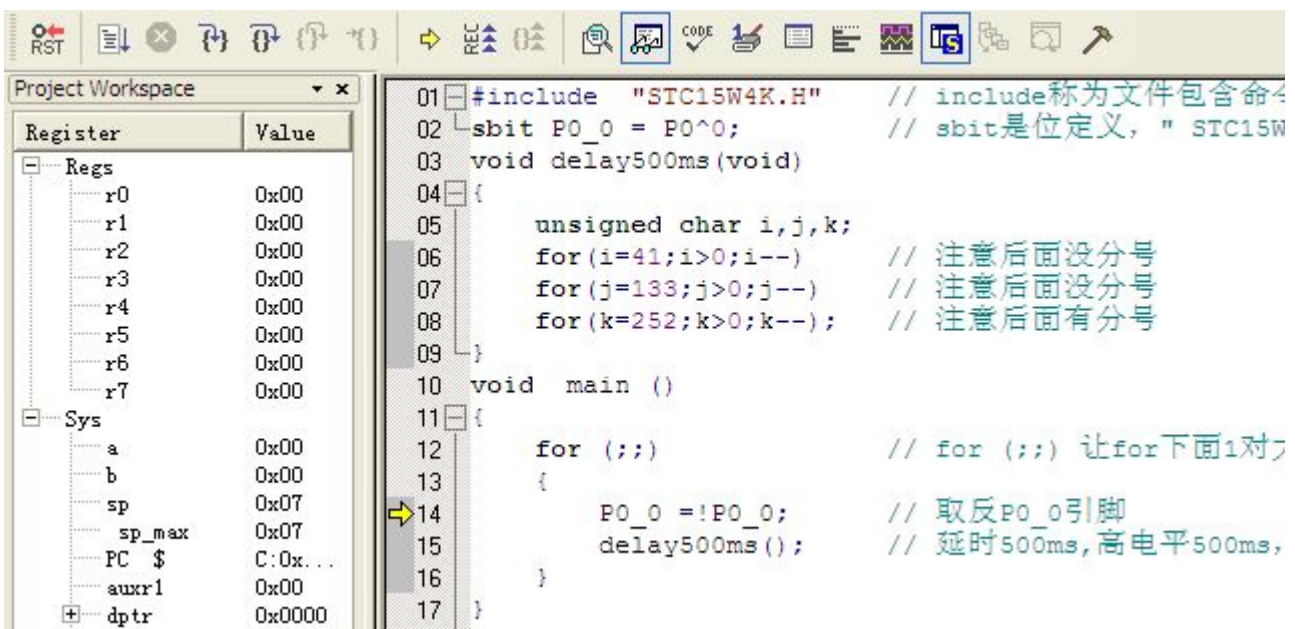


图 1-49 调试窗口

按下 F11 键，可以看到源程序窗口的左边出现了一个黄色调试箭头，指向程序中将要运行的一行，每按一次 F11，即执行该箭头所指程序行，然后箭头指向下一行，当箭头指向 delay500ms(); 行时，再次按下 F11，会发现，箭头指向了延时函数 delay500ms() 中的一行，不断按 F11 键，即可逐步执行延时函数，但有个问题，本例中的延时程序可能要按 F11 键上万次才能执行完延时函数，显然这种单步运行方式不合

适,为此,可以采取以下一些方法,第一,用鼠标在延时函数的最后一行“}”点一下,把光标定位于该行,然后用菜单 Debug→Run to Cursor line (执行到光标所在行),即可全速执行完黄色箭头与光标之间的程序行,第二,在进入该子程序后,使用菜单 Debug→Step Out of Current Function (单步执行到该函数外),使用该命令后,即全速执行完调试光标所在的函数并指向主程序中的下一行程序,第三种方法,在开始调试的,按 F10 而非 F11,程序也将单步执行,不同的是,执行到 delay500ms();行时,按下 F10 键,调试光标不进入函数内部,而是全速执行完该函数,然后直接指向主程序的下一行,灵活应用这几种方法,可以大大提高查错的效率。

4、断点设置

程序调试时,一些程序行必须满足一定的条件才能被执行到(如程序中某变量达到一定的值、按键被按下、串口接收到数据、有中断产生等),这些条件往往是异步发生或难以预先设定的,这类问题使用单步执行的方法是很难调试的,这时就要使用到程序调试中的另一种非常重要的方法——断点设置。断点设置的方法有多种,常用的是在某一程序行设置断点,设置好断点后,可以全速运行程序,一旦执行到该程序行即停止,可在此观察有关变量值,以确定问题所在。在程序行设置/移除断点的方法是将光标定位于需要设置断点的程序行,使用菜单 Debug→Insert/Remove BreakPoint 设置或移除断点(也可以用鼠标在该行双击实现同样的功能),Debug→Enable/Disable Breakpoint 是开启或暂停光标所在行的断点功能,Debug→Disable All Breakpoint 暂停所有断点,Debug→Kill All BreakPoint 清除所有的断点设置。这些功能也可以用工具条上的快捷按钮进行设置。

除了在某程序行设置断点这一基本方法以外,Keil 软件还提供了多种设置断点的方法,执行菜单 Debug→Breakpoints... 即出现一个对话框,该对话框用于对断点进行详细的设置,如图 1-50 所示。

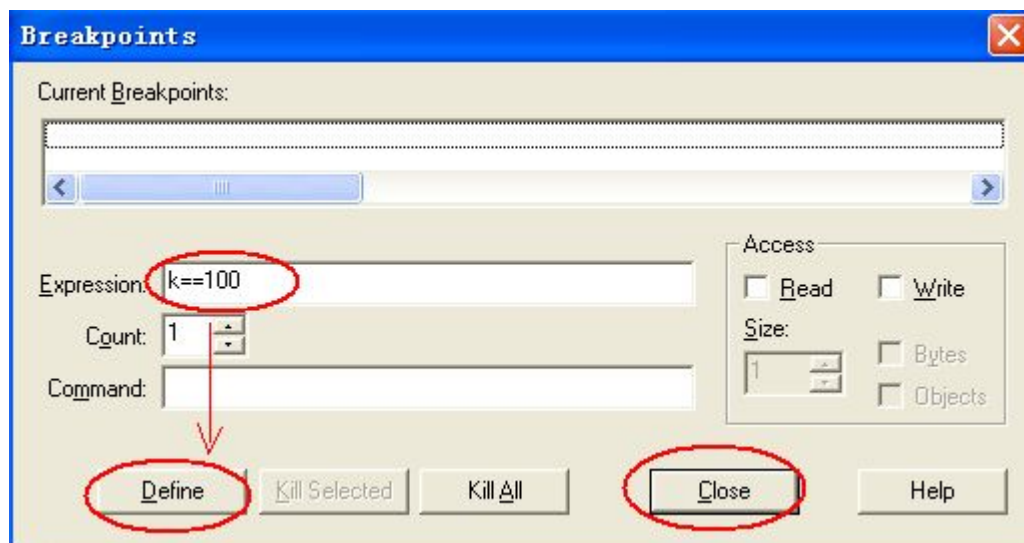


图 1-50 断点设置

图中 Expression 后的编辑框内用于输入表达式,该表达式用于确定程序停止运行的条件,举例说明如下。

① 假设当前的延时函数有问题,程序单步执行进入延时函数,此时在 Expression 中键入 k==100,再点击 Define 即定义了一个断点,如果还没进入延时函数就定义断点,由于变量 k 还没有分配内存空间,点 Define 按钮时会出现变量没定义的错误提示,如图 1-51 所示,也就是说设置断点的变量只能是全局变量和调试箭头所指函数中的局部变量。



图 1-51 断点定义错误

另外，k 后有两个等号，意即相等，该表达式的含义是：如果 k 的值到达 100 则停止程序运行，除使用相等符号之外，还可以使用 >、>=、<、<=、!=（不等于）、&（两值按位与）、&&（两值相与）等运算符号。

② 在 Expression 后键入 k==100，按 Count 后的微调按钮，将值调到 3，其意义是当第三次执行到 k==100 时才停止程序运行。

③ 在 Expression 后键入 k==100，在 Command 后键入 printf(“k==100\n”)，主程序每次执行到 k==100 时并不停止运行，但会在输出窗口 Command 页输出一行字符，即 k==100。其中“\n”的用途是回车换行，使窗口输出的字符整齐。

④ 设置断点前先在输出窗口的 Command 页中键入 DEFINE int a，如图 1-52 所示，然后再设置断点同③，但是 Command 后键入 printf(“k==100 %d times\n”, ++a)，则主程序每次执行到 k==100 时并不停止运行，而会在 Command 窗口输出该字符及被调用的次数，如：k==100 102 times

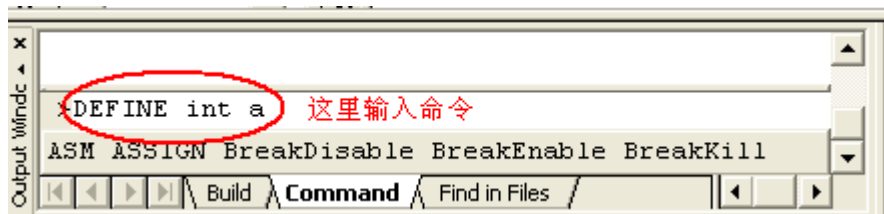


图 1-52 命令栏输入调试命令

5、常用调试窗口

① 变量观察窗口，如图 1-53 所示。

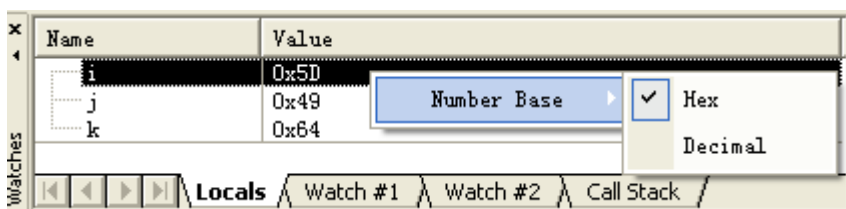


图 1-53 变量观察窗口

执行菜单 View → Watch & Call Stack Window 打开变量观察窗口，其中有一个标签页为 Locals，这一页会自动显示当前函数模块中的变量名及变量值。另外两个标签页 Watch #1 和 Watch #2 可以加入自定义的观察变量，单击 Watch #1 或 Watch #2 窗口中的 <type F2 to edit>，然后再按 F2 即可输入需要观察的变量，在程序较复杂，变量很多的场合，这两个自定义观察窗口可以筛选出我们自己感兴趣的变量加以观察。观察窗口中变量的值不仅可以观察，还可以修改，点击变量后面的值，再按 F2，该值即可修改（也可单击 2 次该值然后修改），该窗口显示的变量值可以十进制或十六进制形式显示，方法是在显示窗口点右键，在快捷菜单中选择。

② 存储器窗口，如图 1-54 所示。

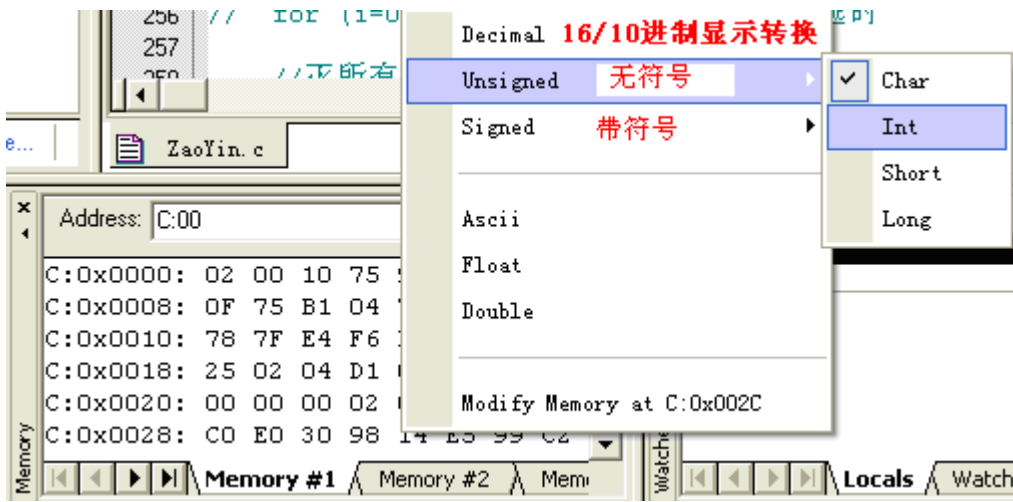


图 1-54 存储器数值显示方式

代码编译成功后执行 File → Open 打开与工程名对应的*.m51 文件可找到自定义变量名与内存地址的对应关系，然后执行菜单 View → Memory Window 打开存储器窗口，通过在 Address 后的编辑框内输入“字母：数字”，然后回车即可显示相应内存值，其中字母可以是 C、D、I、X，分别代表代码存储空间、直接寻址的片内存储空间、间接寻址的片内存储空间、扩展的外部 RAM 空间，数字代表想要查看的地址。例如输入 D: 0 即可观察到地址 0 开始的片内 RAM 单元值、键入 C: 0 即可显示从 0 开始的 FLASH 存储单元中的值，即查看程序的二进制代码。

③ 反汇编窗口，如图 1-55 所示。

点击 View→Dissassembly Window 可以打开反汇编窗口，该窗口可以显示纯汇编代码、也可 C 语言代码和汇编代码混合显示，可以在该窗口按汇编代码的方式单步执行程序，在反汇编窗口，点击鼠标右键，出现快捷菜单，其中 Mixed Mode 是以混合方式显示，Assembly Mode 是以纯汇编代码方式显示。

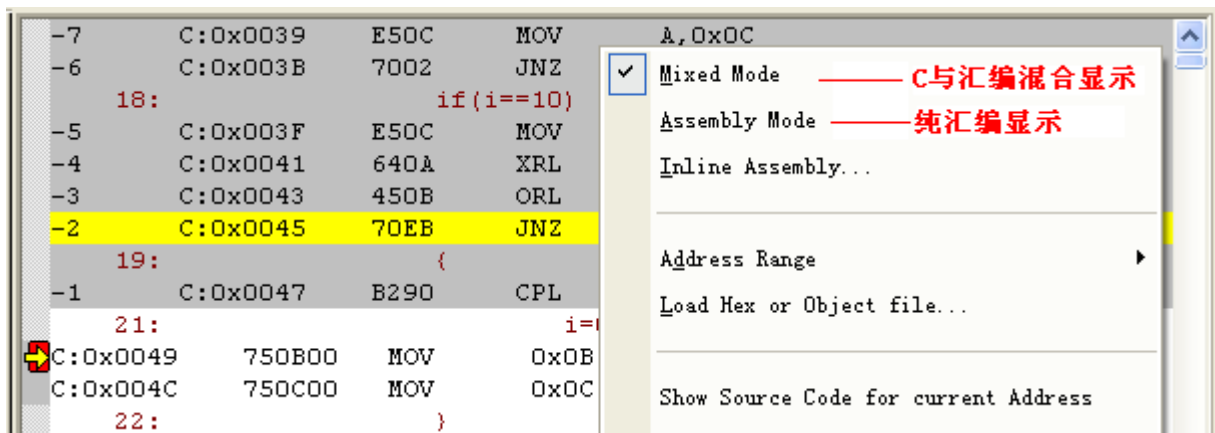


图 1-55 反汇编窗口

④ 外围窗口，如图 1-56 所示。

为了能够比较直观地了解单片机中定时器、中断、并行端口、串行端口等常用外设的使用情况，Keil 提供了一些外围接口对话框，通过 Peripherals 菜单选择，该菜单的下拉菜单内容与你建立项目时所选的 CPU 有关，如果是选择的 89C52 这一类“标准”的 51 机，那么将会有 Interrupt（中断）、I/O Ports（并行 I/O 口）、Serial（串行口）、Timer（定时器/计数器）这四个外围设备菜单。打开这些对话框，列出了外围设备的当前使用情况，各标志位的情况等，可以在这些对话框中直观地观察和更改各外围设备的运行情况，最常用的还是 8 位并口，P0、P1、P2、P3，图 1-56 中 P0 口内部寄存器输出与外部引脚电平不一定相同，比如内部为弱上拉高电平输出状态，但外部引脚可以强制接地成为低电平 0V。

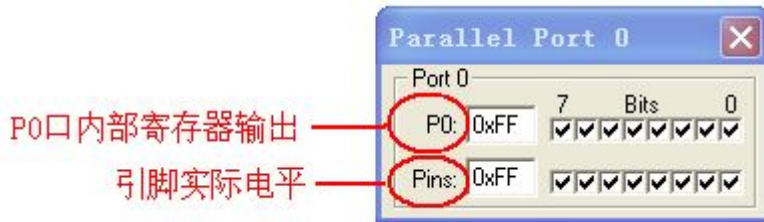



图 1-56 外围窗口

⑤ 逻辑分析仪窗口，如图 1-57 所示。

在软件仿真环境下，单击按钮进入逻辑分析仪窗口，在图 1-57 的第 4 步设置端口时，按“端口名.位”格式输入，显示时只能显示端口名，设置成功的位的显示在自动生成的那一栏的 Shift Right 后面，也可在波形显示界面的左边看到，注意设置完成必须点工具栏运行按钮才有波形输出，需要停止波形运行单击紧挨着运行按钮旁边的停止按钮即可，Zoom 下的 3 个按钮可放大、缩小与显示完整波形。

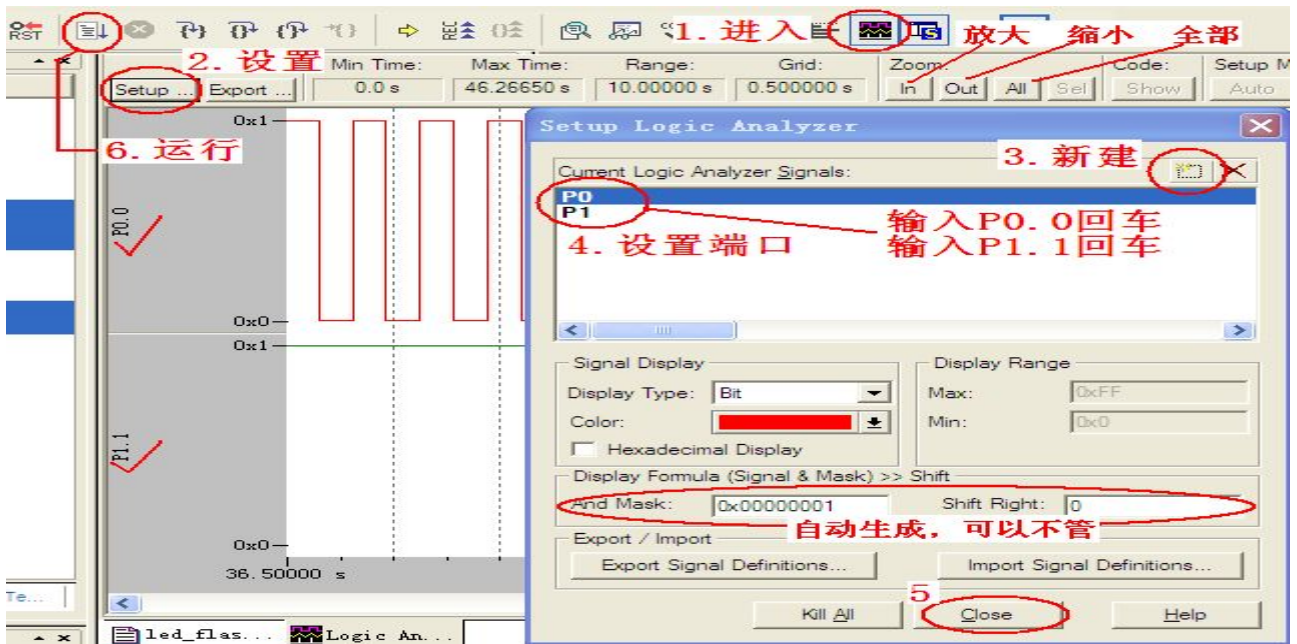


图 1-57 逻辑分析仪窗口

1.3.2 硬件仿真（利用 STC 专用仿真芯片仿真，可仿真所有功能）

(1) 仿真设置步骤

如图 1-58 所示，在 STC 程序下载软件中首先选择“Keil 仿真设置”页面，点击“添加型号和头文件到 Keil 中”，在出现的目录选择窗口中，定位到 Keil 的安装目录(比如“C:\Keil818\”)，“确定”后出现“STC MCU 型号添加成功”的提示信息，点“确定”。

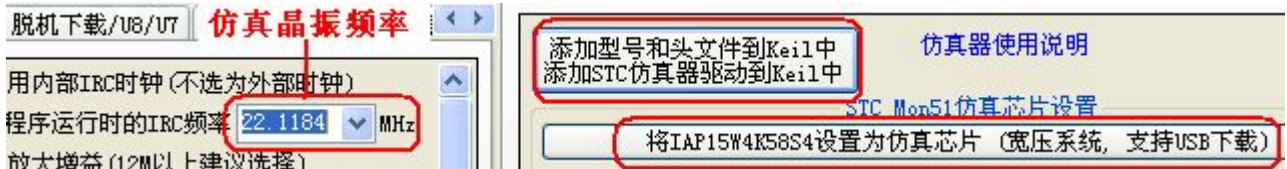


图 1-58 安装 Keil 版本的仿真驱动

在 Keil 中新建项目，出现如图 1-59 对话框，选择“STC MCU Database”项，然后从列表中选择相应的 MCU 型号，在此选择“STC15W4K32S4”，点击“OK”完成选择。

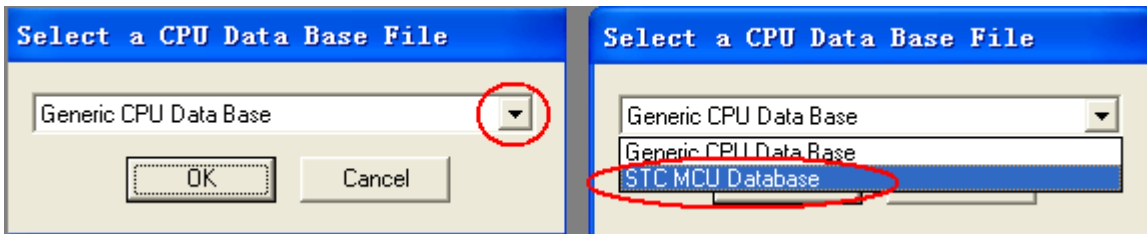



图 1-59 芯片种类选择

然后按通常的步骤编写代码，编译成功后，点 KEIL 工具栏图标  进入设置工程界面，选 Debug 选项卡，如图 1-60 所示，选择右侧的硬件仿真“Use ...”，在仿真驱动下拉列表中选择“STC Monitor-51 Driver”项，然后点击“Settings”按钮，对串口的端口号和波特率进行设置，波特率一般选择 115200 或者 57600。然后选中“Load Application at Startup”和“Run to main()”，最后确定即可。

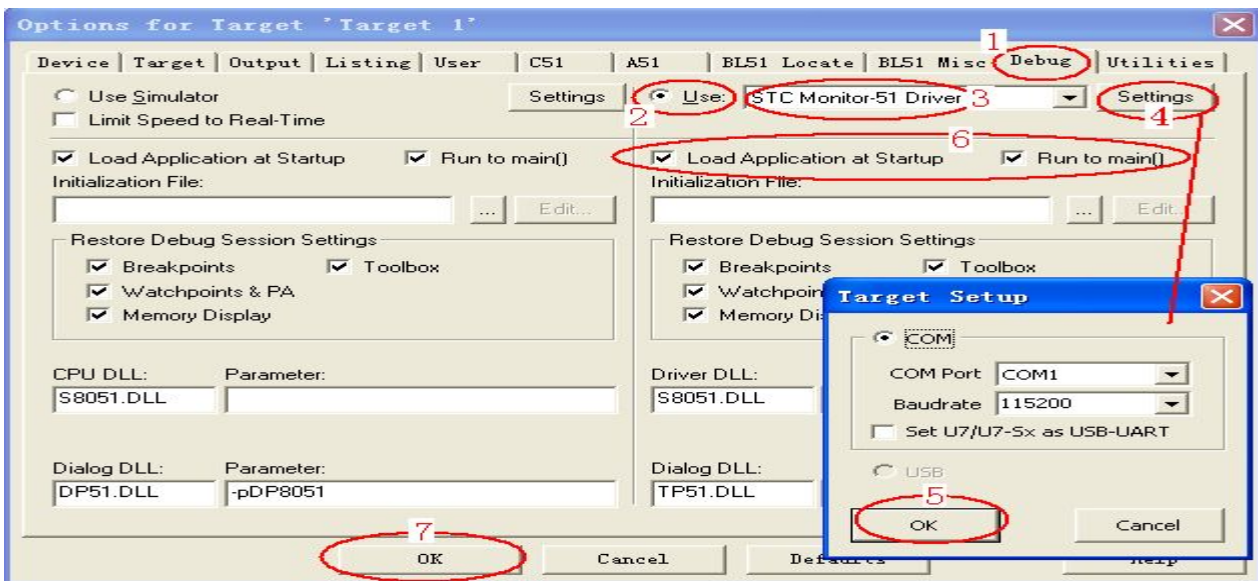



图 1-60 硬件仿真设置步骤

然后准备一颗 IAP15W4K58S4 按正确方向插入前面的实验电路，实验电路仍然与电脑串口相连，然后按图 1-58 所示操作，先选择仿真芯片运行时的 R/C 时钟频率或使用外部晶振，然后点击“将 IAP15W4K58S4 设置为仿真芯片”按钮，给电路板上电，此时就将会有程序向芯片中下载，下载时间最快大约需要 10 秒，当程序下载完成后仿真器便制作完成了，IAP15W4K58S4 设置成仿真芯片后，要想再变成一般的单片机无需任何操作，直接将它当作单片机下载程序使用就可以了。

确认前面我们所创建的项目编译没有错误后，按“Ctrl+F5”或工具栏图标  开始调试，若硬件连接无误的话，将会进入到与软件仿真类似的调试界面，只是现在可以一步一步执行程序并控制硬件动作了，断点设置的个数目前最多允许 20 个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。

有时进入调试环境可能会失败，这时可将整个程序代码注释掉，只写一个最简单的主函数，编译后再尝试进入调试环境，若顺利进入，说明是软件代码（比如串口程序）占用了仿真调试接口，否则可能是仿真串口号选择有误或硬件问题，比如仿真芯片问题或计算机出来的串口工作不正常。

(2) 仿真代码占用资源

- ① 程序空间：仿真代码占用程序区最后 6K 字节的空间，比如用 IAP15W4K58S4 仿真，用户程序只能占 52K (0x0000~0xCFFF) 空间，程序最大不能超过 $52 \times 1024 = 53248$ 字节，用户程序不要使用从 0xD000 到 0xE7FF 的 6K 字节空间。
- ② 常规 RAM (data, idata) : 0 字节
- ③ XRAM(xdata) : 占用最后 768 字节空间 (0x0D00 - 0x0FFF，用户在程序中不要使用)，对于

IAP15W4K58S4，只要程序占用 XRAM 不超过 4096-256-768=3072 字节即可。

④ I/O : P3.0/P3.1, 用户在程序中不得操作 P3.0/P3.1, 不要使用 INT4/T2CLK0/P3.0, 不要使用 T2/P3.1。对于 IAP 型号单片机, 对 EEPROM 的操作是通过对多余不用的程序区进行 IAP 模拟实现的, 此部分要修改程序(IAP 起始地址)。如 IAP15W4K58S4 单片机的 EEPROM 区的位置如图 1-61 所示。

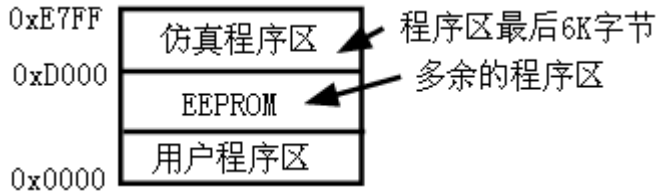


图 1-61 仿真过程中 EEPROM 位置安排示意图

1.4 经典流水灯实例

例 1.3 让接在 IAP15W4K58S4 的 P0.7 引脚发光二极管 1 秒钟闪烁 1 次, R/C 时钟: 11.0592MHZ。

```
#include "STC15W4K.H" // 注意宏定义语句后面无分号
void delay500ms()
{
    unsigned char i, j, k; // i, j, k 由由软件计算出并验证正确。
    for(i=41; i>0; i--) // 注意后面没分号
    for(j=133; j>0; j--) // 注意后面没分号
    for(k=252; k>0; k--); // 注意后面有分号
}
void port_mode() // 端口模式
{
    POM1=0x00; POM0=0x00; P1M1=0x00; P1M0=0x00; P2M1=0x00; P2M0=0x00; P3M1=0x00; P3M0=0x00;
    P4M1=0x00; P4M0=0x00; P5M1=0x00; P5M0=0x00; P6M1=0x00; P6M0=0x00; P7M1=0x00; P7M0=0x00;
}
void main()
{
    port_mode(); // 将单片机所有端口配置为准双向弱上拉方式
    while(1)
    {
        P0 &=~(1<<7); // 将端口单独某位置 0 (提示: C 语言中很重要的技巧)
        delay500ms(); // 延时 500ms
        P0 |= (1<<7); // 将端口单独某位置 1 (提示: C 语言中很重要的技巧)
        delay500ms(); // 延时 500ms
    }
}
```

例 1.4 利用异或运算符 ^ 实现 IAP15W4K58S4 的 P0.4 引脚发光二极管 1 秒钟闪烁 1 次。

// 异或运算符 ^ 最重要的用途是对字节中的某位取反 (提示: C 语言中很重要的技巧)

```
#include "STC15W4K.H" // 注意宏定义语句后面无分号
void delay500ms()
{ // 同例 1.3
```

```

}
void main()
{
    while(1)
    {
        P0 ^= (1<<3);      // 等效于 P0 = P0 ^ (1<<3);
        delay500ms();
    }
}

```

例 1.5 最精简的流水灯实例(A)

//核心思想: 左移位数不断改变, 被移动的数值固定为 0000 0001。(变位数, 定数值), R/C 时钟: 11.0592MHZ。

```

#include "STC15W4K.H"      // 注意宏定义语句后面无分号
void delay100ms()
{
    unsigned char i, j, k;    // i, j, k 由软件计算出并验证正确。
    for(i=157; i>0; i--)      // 注意后面没分号
    for(j=9; j>0; j--)        // 注意后面没分号
    for(k=194; k>0; k--);     // 注意后面有分号
}
void port_mode()            // 端口模式
{
    ..... // 同例 1.3
}
void main()
{
    unsigned char a;
    port_mode();             // 将单片机所有端口配置为准双向弱上拉方式
    while(1)
    {
        P0 ^= (1<<a++);      // 第一次运行时 0000 0001<< 0 = 0000 0001
        delay100ms();
        if (a==0x08)         // 允许左移 8 次。
        {
            a=0;
        }
    }
}

```

例 1.6 最精简的流水灯实例(B)

//核心思想: 左移位数固定为 1, 被移动的数值是前一次移位的结果。(定位数, 变数值)。

```

#include "STC15W4K.H"      // 注意宏定义语句后面无分号
void delay100ms()
{
    // 同例 1.5
}

```

```

}
int main()
{
    unsigned char i;
    unsigned char k;           // k 用于保存移位后的值
    port_mode();              // 将单片机所有端口配置为准双向弱上拉方式
    while(1)
    {
        k=0x01;                // 先给 K 一个初值 0000 0001 等待移位
        for(i=8;i>0;i--)
        {
            P0=~k;
            delay100ms();
            k=k<<1;           // 把 k 左移 1 位, 左移时数值最低位填 0 补充
        }
    }
}

```

C 语言代码书写提示:

① C 语言中可以在一行写多个语句，每个语句用分号表示接束。比如上例也可这样写:

```

for(i=8;i>0;i--)
{
    P0=~k;delay100ms(); k=k<<1;    // 把 k 左移 1 位, 左移时数值最低位填 0 补充
}

```

② C 语言中可以将一个语句写在多行，在 Visual Basic 中，前一行的结尾用空格和下划线表明后一行的内容与前一行作为一个语句，但 C 语言中不用任何标记，直接把一个语句的其它内容写在下一行即可，但注意单词（如变量名、关键字等）不能分成多行，否则编译不能通过。

1.5 单片机 C 语言延时程序详解

1.5.1 学会使用计算软件

在本章前面反复出现了一个延时函数，格式如下:

```

void delay500ms()           // 大范围精确延时函数
{
    unsigned char i, j, k;   // i, j, k 由软件计算出确定。
    for(i=41;i>0;i--)       // 注意后面没分号
    for(j=133;j>0;j--)      // 注意后面没分号
    for(k=252;k>0;k--);    // 注意后面有分号
}

```

在本书后面还会出现一个延时函数，格式如下:

```

void delay (unsigned char t) // 小范围精确延时函数
{
    while(--t);
}

```


}

读者需要使用延时程序时，直接将这里的延时函数复制到自己的程序中，然后修改函数中的变量值就可以得到精确的延时时间，如果需要延时时间较长，比如流水灯类程序时间大于 100uS，使用前一个延时函数，延时时间很短的 18B20 通信类程序要求小于 100uS，使用后一个延时函数，函数中的 i、j、k 和 t 的具体数值由作者编写好的软件计算，软件界面如图 1-62 所示。

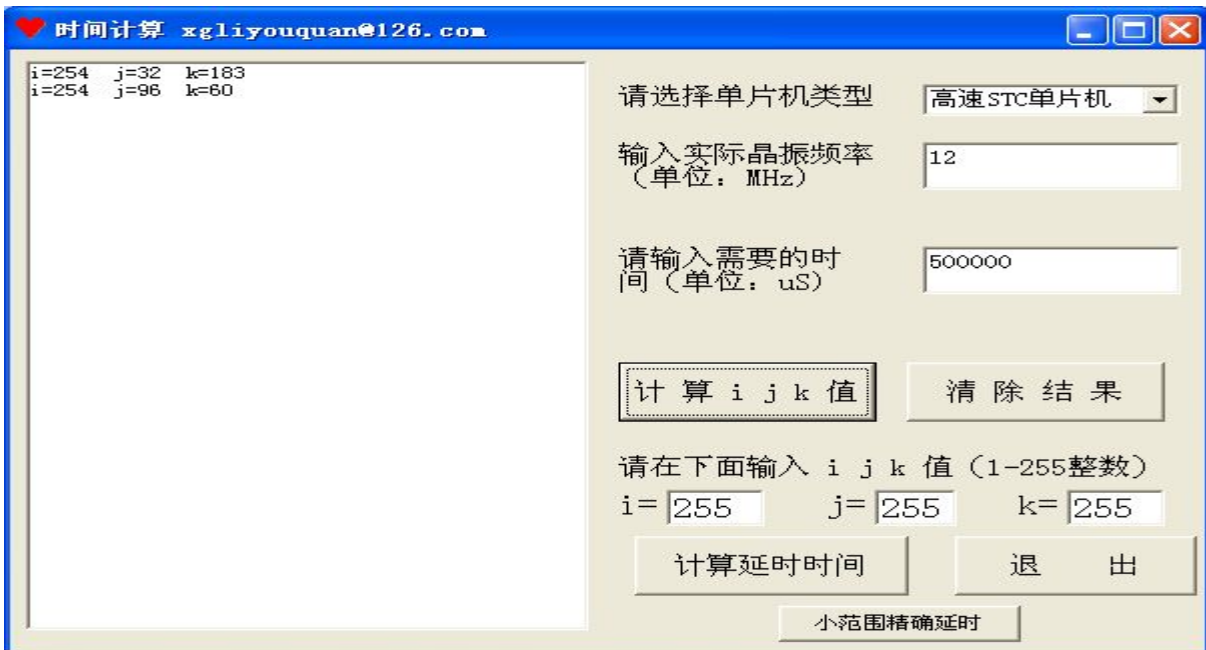


图 1-62 单片机延时程序变量计算

从图 1-62 中可以看出不但可以根据要求的延时时间计算 i j k 的值，也可由 i j k 的值计算出延时时间值，使用非常方便。读者使用这个软件时可以把文件夹“安装程序”复制到 C 盘根目录下，在 C 盘根目录下运行安装程序，安装路径随意，默认路径 C:\Program Files\延时时间\，安装完毕后找到安装路径下的图标，如图 1-63 所示，然后右键选择“发送到-桌面快捷方式”方便随时使用。除使用这个软件外，读者也可以使用 STC 下载软件中附带的“软件延时计算器”。



图 1-63 发送到桌面快捷方式

重要提示：

- ① 程序计算出来通常会有多组数据，随便选一组都可以。
- ② 为保证延时时间的准确性，需要确认工程设置中的代码优化等级为默认的为 8 级或 6 级，优化等级更低时延时时间可能会不准确（变长）。另外还要确定延时函数运行过程中不应有中断函数打断，否则延时时间会变长。
- ③ 延时函数使用到单片机 4 条汇编指令：(1)MOV Rn #data，需 2 个时钟。(2)DJNZ Rn rel，需 4 个时钟 (3) RET，需 4 个时钟 (4) LCALL，需 4 个时钟。对于 STC11、STC12、STC15 系列，这几条指令执行时间是一样的，因此界面中选择“高速 STC 单片机”计算出的延时函数变量对这 3 个系列是通用的。

1.5.2 计算软件内部运算过程详解

固定格式大范围精确延时函数计算公式（后面的详细分析步骤总结出的结果）：

$$\text{STC 51: 延时总时间} = (4 i j k + 6 i j + 6 i + 12) \times 1/F \quad (\text{单位: uS})$$

固定格式小范围精确延时函数计算公式（后面的详细分析步骤总结出的结果）：

$$\text{STC 51: 延时总时间} = (t \times 4 + 10) \times 1/F \quad (\text{单位: } \mu\text{S})$$

说明：F—代表任意 R/C 时钟或外部晶振频率（单位：MHz），延时时间单位是 us，延时时间连同主程序调用时间也计算在内。

i, j, k 的最小值只能为 1，假设晶振 12M，延时时间 AT89C51=13 uS, IAP15W4K58S4= 2. 333333 uS。

i, j, k 的最大值 255，假设晶振 12M，延时时间 AT89C51=33. 358595S, IAP15W4K58S4=5. 559766S。

当设定时间大于最长时间时，软件会自动给出最长时间提示。

1、大范围精确延时详细分析

以下是在 WAVE 环境中调试通过的一个完整程序（WAVE 环境状态栏第 2 格显示指令执行时间）

```

MAIN:MOV P1, #0FFH
      LCALL DELAY ; 执行时间 10070, LCALL 指令本身要占用 2 各机器周期 (2 uS)
      MOV P1, #00H
      CALL DELAY
      LJMP MAIN
DELAY:MOV R7, #5 ; 执行时间: 10068
D1:MOV R6, #10
D2:MOV R5, #99
   DJNZ R5, $
   DJNZ R6, D2
   DJNZ R7, D1
   RET
   END
    
```

以下是延时程序代码详细分析过程：

	89C51 单片机	STC_1T 单片机
DELAY: MOV R7, #5	1uS	1/6 uS
D1: MOV R6, #10	1uS	1/6 uS
D2: MOV R5, #99	1uS	1/6 uS
DJNZ R5, \$	2 × 99 = 198uS	1/3 uS
DJNZ R6, D2	2uS	1/3 uS
DJNZ R7, D1	2uS	1/3 uS
RET	2uS	1/3 uS

大循环

不管多少层循环嵌套，只要按照以下步骤都能极容易算出精确延时时间。（关键点：将最内层循环当作 1 条指令，求出此指令执行时间，再将中层循环当作 1 条指令，求出此指令执行时间，最后加上循环外的时间）。

第 1 步：先在代码行后面写出各行代码执行时间。

第 2 步：用方框从内到外框出逐层循环代码区域。

第 3 步：从内到外逐层计算时间如下：

最内层循环时间：算出单次循环总时间（1 + 198 + 2）uS

找出单次循环执行次数（单次循环最后一条指令的 R6 中）：10

最内层循环时间 = $(1 + 198 + 2) \times 10 = 2010 \text{ uS}$ // $(2 \times R5 + 3) \times R6$

中层循环时间：算出单次循环总时间 $(1 + 2010 + 2) \text{ uS}$

找出单次循环执行次数（单次循环最后一条指令的 R7 中）：5

中层循环时间 = $(1 + 2010 + 2) \times 5 = 10065$ // $[(2 \times R5 + 3) \times R6 + 3] \times R7$

延时程序时间 = 循环时间 + 循环外时间

DELAY = $1 + 10065 + 2 = 10068 \text{ uS} = 10.068\text{ms}$ // $[(2 \times R5 + 3) \times R6 + 3] \times R7 + 3$

若连主程序调用 2us 计算在内，则总时间计算公式

延时总时间 = $[(2 \times R5 + 3) \times R6 + 3] \times R7 + 5$ (提示：此公式只对普通 51_12M 晶振有效)

在 keil 中新建工程，输入延时程序，编译后进入调试环境，打开反汇编窗口如图 1-64。

```

10: void main()
11: {
12:   delay500ms();
13:   while(1);
14: }
15: void delay500ms()
16: {
17:   unsigned char i,j,k;
18:   for(i=15;i>0;i--)
19:     for(j=202;j>0;j--)
20:       for(k=81;k>0;k--);
21: }
22: }
23: }
24: }
25: }
26: }
27: }
28: }
29: }
30: }
31: }
32: }
33: }
34: }
35: }
36: }
37: }
38: }
39: }
40: }
41: }
42: }
43: }
44: }
45: }
46: }
47: }
48: }
49: }
50: }
51: }
52: }
53: }
54: }
55: }
56: }
57: }
58: }
59: }
60: }
61: }
62: }
63: }
64: }
65: }
66: }
67: }
68: }
69: }
70: }
71: }
72: }
73: }
74: }
75: }
76: }
77: }
78: }
79: }
80: }
81: }
82: }
83: }
84: }
85: }
86: }
87: }
88: }
89: }
90: }
91: }
92: }
93: }
94: }
95: }
96: }
97: }
98: }
99: }
100: }
101: }
102: }
103: }
104: }
105: }
106: }
107: }
108: }
109: }
110: }
111: }
112: }
113: }
114: }
115: }
116: }
117: }
118: }
119: }
120: }
121: }
122: }
123: }
124: }
125: }
126: }
127: }
128: }
129: }
130: }
131: }
132: }
133: }
134: }
135: }
136: }
137: }
138: }
139: }
140: }
141: }
142: }
143: }
144: }
145: }
146: }
147: }
148: }
149: }
150: }
151: }
152: }
153: }
154: }
155: }
156: }
157: }
158: }
159: }
160: }
161: }
162: }
163: }
164: }
165: }
166: }
167: }
168: }
169: }
170: }
171: }
172: }
173: }
174: }
175: }
176: }
177: }
178: }
179: }
180: }
181: }
182: }
183: }
184: }
185: }
186: }
187: }
188: }
189: }
190: }
191: }
192: }
193: }
194: }
195: }
196: }
197: }
198: }
199: }
200: }
201: }
202: }
203: }
204: }
205: }
206: }
207: }
208: }
209: }
210: }
211: }
212: }
213: }
214: }
215: }
216: }
217: }
218: }
219: }
220: }
221: }
222: }
223: }
224: }
225: }
226: }
227: }
228: }
229: }
230: }
231: }
232: }
233: }
234: }
235: }
236: }
237: }
238: }
239: }
240: }
241: }
242: }
243: }
244: }
245: }
246: }
247: }
248: }
249: }
250: }
251: }
252: }
253: }
254: }
255: }
256: }
257: }
258: }
259: }
260: }
261: }
262: }
263: }
264: }
265: }
266: }
267: }
268: }
269: }
270: }
271: }
272: }
273: }
274: }
275: }
276: }
277: }
278: }
279: }
280: }
281: }
282: }
283: }
284: }
285: }
286: }
287: }
288: }
289: }
290: }
291: }
292: }
293: }
294: }
295: }
296: }
297: }
298: }
299: }
300: }
301: }
302: }
303: }
304: }
305: }
306: }
307: }
308: }
309: }
310: }
311: }
312: }
313: }
314: }
315: }
316: }
317: }
318: }
319: }
320: }
321: }
322: }
323: }
324: }
325: }
326: }
327: }
328: }
329: }
330: }
331: }
332: }
333: }
334: }
335: }
336: }
337: }
338: }
339: }
340: }
341: }
342: }
343: }
344: }
345: }
346: }
347: }
348: }
349: }
350: }
351: }
352: }
353: }
354: }
355: }
356: }
357: }
358: }
359: }
360: }
361: }
362: }
363: }
364: }
365: }
366: }
367: }
368: }
369: }
370: }
371: }
372: }
373: }
374: }
375: }
376: }
377: }
378: }
379: }
380: }
381: }
382: }
383: }
384: }
385: }
386: }
387: }
388: }
389: }
390: }
391: }
392: }
393: }
394: }
395: }
396: }
397: }
398: }
399: }
400: }
401: }
402: }
403: }
404: }
405: }
406: }
407: }
408: }
409: }
410: }
411: }
412: }
413: }
414: }
415: }
416: }
417: }
418: }
419: }
420: }
421: }
422: }
423: }
424: }
425: }
426: }
427: }
428: }
429: }
430: }
431: }
432: }
433: }
434: }
435: }
436: }
437: }
438: }
439: }
440: }
441: }
442: }
443: }
444: }
445: }
446: }
447: }
448: }
449: }
450: }
451: }
452: }
453: }
454: }
455: }
456: }
457: }
458: }
459: }
460: }
461: }
462: }
463: }
464: }
465: }
466: }
467: }
468: }
469: }
470: }
471: }
472: }
473: }
474: }
475: }
476: }
477: }
478: }
479: }
480: }
481: }
482: }
483: }
484: }
485: }
486: }
487: }
488: }
489: }
490: }
491: }
492: }
493: }
494: }
495: }
496: }
497: }
498: }
499: }
500: }
501: }
502: }
503: }
504: }
505: }
506: }
507: }
508: }
509: }
510: }
511: }
512: }
513: }
514: }
515: }
516: }
517: }
518: }
519: }
520: }
521: }
522: }
523: }
524: }
525: }
526: }
527: }
528: }
529: }
530: }
531: }
532: }
533: }
534: }
535: }
536: }
537: }
538: }
539: }
540: }
541: }
542: }
543: }
544: }
545: }
546: }
547: }
548: }
549: }
550: }
551: }
552: }
553: }
554: }
555: }
556: }
557: }
558: }
559: }
560: }
561: }
562: }
563: }
564: }
565: }
566: }
567: }
568: }
569: }
570: }
571: }
572: }
573: }
574: }
575: }
576: }
577: }
578: }
579: }
580: }
581: }
582: }
583: }
584: }
585: }
586: }
587: }
588: }
589: }
590: }
591: }
592: }
593: }
594: }
595: }
596: }
597: }
598: }
599: }
600: }
601: }
602: }
603: }
604: }
605: }
606: }
607: }
608: }
609: }
610: }
611: }
612: }
613: }
614: }
615: }
616: }
617: }
618: }
619: }
620: }
621: }
622: }
623: }
624: }
625: }
626: }
627: }
628: }
629: }
630: }
631: }
632: }
633: }
634: }
635: }
636: }
637: }
638: }
639: }
640: }
641: }
642: }
643: }
644: }
645: }
646: }
647: }
648: }
649: }
650: }
651: }
652: }
653: }
654: }
655: }
656: }
657: }
658: }
659: }
660: }
661: }
662: }
663: }
664: }
665: }
666: }
667: }
668: }
669: }
670: }
671: }
672: }
673: }
674: }
675: }
676: }
677: }
678: }
679: }
680: }
681: }
682: }
683: }
684: }
685: }
686: }
687: }
688: }
689: }
690: }
691: }
692: }
693: }
694: }
695: }
696: }
697: }
698: }
699: }
700: }
701: }
702: }
703: }
704: }
705: }
706: }
707: }
708: }
709: }
710: }
711: }
712: }
713: }
714: }
715: }
716: }
717: }
718: }
719: }
720: }
721: }
722: }
723: }
724: }
725: }
726: }
727: }
728: }
729: }
730: }
731: }
732: }
733: }
734: }
735: }
736: }
737: }
738: }
739: }
740: }
741: }
742: }
743: }
744: }
745: }
746: }
747: }
748: }
749: }
750: }
751: }
752: }
753: }
754: }
755: }
756: }
757: }
758: }
759: }
760: }
761: }
762: }
763: }
764: }
765: }
766: }
767: }
768: }
769: }
770: }
771: }
772: }
773: }
774: }
775: }
776: }
777: }
778: }
779: }
780: }
781: }
782: }
783: }
784: }
785: }
786: }
787: }
788: }
789: }
790: }
791: }
792: }
793: }
794: }
795: }
796: }
797: }
798: }
799: }
800: }
801: }
802: }
803: }
804: }
805: }
806: }
807: }
808: }
809: }
810: }
811: }
812: }
813: }
814: }
815: }
816: }
817: }
818: }
819: }
820: }
821: }
822: }
823: }
824: }
825: }
826: }
827: }
828: }
829: }
830: }
831: }
832: }
833: }
834: }
835: }
836: }
837: }
838: }
839: }
840: }
841: }
842: }
843: }
844: }
845: }
846: }
847: }
848: }
849: }
850: }
851: }
852: }
853: }
854: }
855: }
856: }
857: }
858: }
859: }
860: }
861: }
862: }
863: }
864: }
865: }
866: }
867: }
868: }
869: }
870: }
871: }
872: }
873: }
874: }
875: }
876: }
877: }
878: }
879: }
880: }
881: }
882: }
883: }
884: }
885: }
886: }
887: }
888: }
889: }
890: }
891: }
892: }
893: }
894: }
895: }
896: }
897: }
898: }
899: }
900: }
901: }
902: }
903: }
904: }
905: }
906: }
907: }
908: }
909: }
910: }
911: }
912: }
913: }
914: }
915: }
916: }
917: }
918: }
919: }
920: }
921: }
922: }
923: }
924: }
925: }
926: }
927: }
928: }
929: }
930: }
931: }
932: }
933: }
934: }
935: }
936: }
937: }
938: }
939: }
940: }
941: }
942: }
943: }
944: }
945: }
946: }
947: }
948: }
949: }
950: }
951: }
952: }
953: }
954: }
955: }
956: }
957: }
958: }
959: }
960: }
961: }
962: }
963: }
964: }
965: }
966: }
967: }
968: }
969: }
970: }
971: }
972: }
973: }
974: }
975: }
976: }
977: }
978: }
979: }
980: }
981: }
982: }
983: }
984: }
985: }
986: }
987: }
988: }
989: }
990: }
991: }
992: }
993: }
994: }
995: }
996: }
997: }
998: }
999: }
1000: }

```

图 1-64 大范围精确延时函数反汇编结果

可以看出，反汇编代码与前面介绍的汇编语言编写的延时程序完全相同，程序共有三层循环，重新抄录如下：

以下 T 代表晶振时钟周期：

89C51

STC51

	DELAY: MOV R7, #15	12T	2T
大循环	D1: MOV R6, #202	12T	2T
	D2: MOV R5, #81	12T	2T
	DJNZ R5, \$	24T × R5	4T × R5
	DJNZ R6, D2	24T	4T
	DJNZ R7, D1	24T	4T
	RET	24T	4T

最内层循环时间：算出单次循环总时间 $(12T + 24T \times R5 + 24T) \text{ uS}$

找出单次循环执行次数（单次循环最后一条指令的 R6 中）

最内层循环时间 = $(12T + 24T \times R5 + 24T) \times R6 \text{ uS} = (24T \times R5 + 36T) \times R6 \text{ uS}$

中层循环时间：算出单次循环总时间 $(12T + (24T \times R5 + 36T) \times R6 + 24T) \text{ uS}$

$$= (24T \times R5 + 36T) \times R6 + 36T \text{ uS}$$

找出单次循环执行次数（单次循环最后一条指令的R7中）

$$\text{中层循环时间} = (24T \times R5 + 36T) \times R6 + 36T \times R7$$

延时程序时间 = 循环时间 + 循环外时间

$$\begin{aligned} \text{DELAY} &= 12T + (24T \times R5 + 36T) \times R6 + 36T \times R7 + 24T \\ &= (24T \times R5 + 36T) \times R6 + 36T \times R7 + 36T \end{aligned}$$

若连主程序调用 24T 计算在内，则总时间计算公式

$$\begin{aligned} \text{延时总时间} &= (24T \times R5 + 36T) \times R6 + 36T \times R7 + 36T + 24T \\ &= (24T \times R5 + 36T) \times R6 + 36T \times R7 + 60T \\ &= (24 \cdot R5 \cdot R6 \cdot T + 36 \cdot R6 \cdot T + 36T) \times R7 + 60T \\ &= 24 \cdot R5 \cdot R6 \cdot R7 \cdot T + 36 \cdot R6 \cdot R7 \cdot T + 36 \cdot R7 \cdot T + 60T \\ &= (24 \cdot R5 \cdot R6 \cdot R7 + 36 \cdot R6 \cdot R7 + 36 \cdot R7 + 60) \cdot T \\ &= (24 \cdot R5 \cdot R6 \cdot R7 + 36 \cdot R6 \cdot R7 + 36 \cdot R7 + 60) \times 1/F \end{aligned}$$

由上面反汇编窗口可看出 i = R7, j = R6, k = R5

$$\text{延时总时间} = (24 i j k + 36 i j + 36 i + 60) \times 1/F$$

对于 STC 高速单片机：

$$\text{最内层循环时间：} (2T + 4T \times R5 + 4T) \times R6 = (4T \times R5 + 6T) \times R6$$

$$\text{中层循环时间：} (2T + (4T \times R5 + 6T) \times R6 + 4T) \times R7 = (4T \times R5 + 6T) \times R6 + 6T \times R7$$

$$\begin{aligned} \text{延时程序时间：} & 2T + ((4T \times R5 + 6T) \times R6 + 6T) \times R7 + 4T \\ &= ((4T \times R5 + 6T) \times R6 + 6T) \times R7 + 6T \end{aligned}$$

$$\begin{aligned} \text{主程序调用总时间：} & ((4T \times R5 + 6T) \times R6 + 6T) \times R7 + 6T + 6T \\ &= ((4T \times R5 + 6T) \times R6 + 6T) \times R7 + 12T \\ &= (4 \cdot R5 \cdot R6 \cdot T + 6 \cdot R6 \cdot T + 6T) \times R7 + 12T \\ &= 4 \cdot R5 \cdot R6 \cdot R7 \cdot T + 6 \cdot R6 \cdot R7 \cdot T + 6 \cdot R7 \cdot T + 12T \\ &= (4 \cdot R5 \cdot R6 \cdot R7 + 6 \cdot R6 \cdot R7 + 6 \cdot R7 + 12) \cdot T \\ &= (4 \cdot R5 \cdot R6 \cdot R7 + 6 \cdot R6 \cdot R7 + 6 \cdot R7 + 12) \times 1/F \end{aligned}$$

由上面反汇编窗口可看出 i = R7, j = R6, k = R5

$$\text{延时总时间} = (4 i j k + 6 i j + 6 i + 12) \times 1/F$$

2、小范围精确延时详细分析

在 keil 中新建工程，输入小范围精确延时程序，编译后进入调试环境，打开反汇编窗口如图 1-65 所示

```

2: void delay (unsigned char t)
3: {
4:     while(--t);
C:0x0017    DFFE    DJNZ    R7, delay(C:0017)
5: }
C:0x0019    22     RET

```

图 1-65 小范围精确延时函数反汇编结果

这里先假定单片机是时钟频率为 12MHz，则一个机器周期为 1us，可以看出，这时代码只有 1 句，共占用 24 个时钟周期，精度达到 2 μs，循环体耗时 t×24 个时钟周期，但这时应该注意，t 初始值不能为 0。执

行 DJNZ 指令需要 24 个时钟周期，RET 指令同样需要 24 个时钟周期，根据输入 t，在不计算调用 delay() 所需时间的情况下，具体延时时间计算如表 1-3 所列：

表 1-3 小范围精确延时时间规律

t	普通 51	STC 高速 51
1	$1 \times 24T + 24T$	$1 \times 4T + 4T$
2	$2 \times 24T + 24T$	$2 \times 4T + 4T$
N	$N \times 24T + 24T$	$N \times 4T + 4T$

当在 main 函数中调用 delay(200) 时，进行反汇编结果如图 1-66 所示。

```

6: void main()
7: {
8:     P1=~P1;
C:0x000F    6390FF    XRL     P1(0x90),#0xFF
9:     delay(200);
C:0x0012    7FC8     MOV     R7,#0xC8
C:0x0014    020017   LJMP   delay(C:0017)
2: void delay (unsigned char t)
3: {
4:     while(--t);
C:0x0017    DFFE     DJNZ   R7,delay(C:0017)
5: }
C:0x0019    22      RET

```

图 1-66 小范围延时函数反汇编窗口

调用 delay() 时，多执行了两条指令，其中 MOV R7, #0xc8 需要 12 (STC = 2) 个时钟周期，LJMP 需要 24 (STC = 4) 个时钟周期，即调用 delay() 需要 36 个时钟周期。

所以普通 51 延时总时间： $t \times 24T + 24T + 36T = (t \times 24 + 60)T = (t \times 24 + 60) \times 1/F$ (单位：uS)

STC 高速 51 延时总时间： $t \times 4T + 4T + 6T = (t \times 4 + 10)T = (t \times 4 + 10) \times 1/F$ (单位：uS)

对于 STC 高速 51/11.0592MHz 晶振，最小的时间延时为 $(1 \times 4 + 10) / 11.0592 = 1.26\mu s$ ，最大的时间延时为 $(255 \times 4 + 10) = 93.1\mu s$ ，延时函数的分辨率为 0.36uS，这个函数延时范围虽然比前面的多层循环小得多，但是可对时间作更细致的调整，例如 STC51 传入参数 t=从 100 变到 101 时，延时时间变化 0.36uS，多层循环 i=100、j=100、k=100，当 k 从 100 变到 101 时，延时时间变化 3617uS，为方便使用，根据上面理论分析获得的计算公式，在 Visual Basic 中编写计算程序，具体实现代码可参见配套资源的完整工程文件。

例 1.7 延时程序实验 (STC 单片机, R/C 时钟: 11.0592MHZ)

```

#include "STC15W4K.H" // 注意宏定义语句后面无分号
sbit P0_0=P0^0;
void delay1000ms(void)
{
    unsigned char i,j,k;
    for(i=93;i>0;i--) // 注意后面没分号
    for(j=235;j>0;j--) // 注意后面没分号
    for(k=125;k>0;k--); // 注意后面有分号
}
void delay (unsigned char t)
{
    while(--t);
}

```



```

void main()
{
    while(1)
    {
        P0_0=0;          // 点亮 LED
        delay1000ms(); // 软件计算时间 1S, 逻辑分析仪实测 1.000 208 125S
        P0_0=1;
        delay(219);     // 软件计算时间 80.114uS, 逻辑分析仪实测 80.937uS
    }
}

```

计算软件是非常精确的，晶振的频率误差对软件计算结果的影响也是极其微小，读者使用 void delay (unsigned char t)函数实现很短时间的精确延时，比如延时 2uS, 通常通过 LED 灯输出来测量延时是否准确，如下所示，此时要注意测试方法是否正确。

```

void main()
{
    while(1)
    {
        P00=!P00;    // 要占用时间
        delay(3);    // 精确计算时间 2uS } 此为高或低电平时间，明显要大于 2uS (实测 2.6uS)
        .....
        .....
        P00=!P00;    // 要占用时间
        delay(3);    // 精确计算时间 2uS } 此为高或低电平时间，明显要大于 2uS (实测 2.6uS)
    }
}

```

下面函数就更不能通过测量 LED 高或低电平验证时间了。

```

void main()
{
    while(1)
    {
        P00=!P00;    // 要占用时间
        delay(3);    // 精确计算时间 2uS } 此为一个周期，LED 高或低电平时间实测 3uS。
    }
}

```

1.5.3 利用库函数实现短暂精确延时

在 C 文件中可使用 _NOP_() ;命令实现短暂精确延时，此 _NOP_() 与汇编语言中的 NOP 空操作命令效果完全相同，相当于是在 C 语言文件中插入汇编指令，但在使用此命令的 C 文件中必须加上 #include <intrins.h> 声明，对于 STC11、STC12、STC15 系列，NOP 指令执行时间都是 1 个时钟周期，1 个时钟周期的时间 $T=1/F$ ，F 代表系统时钟频率，默认值与 R/C 时钟或外部晶振频率相同，假设 $F=11.0592\text{MHz}$ ， $T=90\text{nS}=0.09\text{uS}$ ，这个时间太快了，单个 _NOP_() ;一般不够用，可以这样：

```
#define NOP _nop();_nop();_nop();_nop();_nop();
```

用 1 个“NOP” 代替 6 个“_nop();”， $6 \times 0.09 = 0.54 \mu\text{s}$ ，如果时间不够，可继续增加 _nop(); 命令，使用时直接在代码行输入 NOP; 即可。

1.5.4 使用定时器/计数器实现精确延时

假设使用频率为 12 MHz 的 R/C 时钟或外部晶振，若定时器工作在方式 0，16 位自动重装方式，可实现精确延时 ($0 \sim 65536 \mu\text{s}$)，工作在方式 1，普通 16 位计数方式，延时 ($0 \sim 65536 \mu\text{s}$)，工作在方式 2，8 位自动重装方式，可实现精确延时 ($0 \sim 256 \mu\text{s}$)。如果延时时间大于 $65536 \mu\text{s}$ ，可使用普通 16 位计数方式 1，并在定时中断函数中使用软件计数器对中断次数计数，这样可获得几秒，几分钟或很多小时的定时时间，但是每次进入中断函数时对软件计数器的加一操作、重装定时器初值的操作要占用时间，中断函数在进入与退出时编译器会自动加上 PUSH ACC、PUSH PSW、POP PSW 和 POP ACC 语句，这些语句都要占用时间，要想计算这些细微的时间是非常复杂的，实际中如果要求非常精确，一般在中断函数退出时对一只发光二极管执行取反操作，然后运行程序，使用逻辑分析仪测量发光二极管上的高低电平时间，根据测量结果调整定时器初值直到满意为止，另外也可以使用定时器的 16 位自动重装方式 0 获得更精确的时间。

使用定时器/计数器延时一般是在定时器中断函数中占用 CPU 很少时间，前面软件计算方式与使用 _nop(); 命令都是以独占 CPU 的方式运行的。

1.6 main()、void main()和 int main() 的区别

在 C 语言中 main() 和 void main() 区别：一个有返回值（没声明类型的默认是返回值 int 型），一个无返回值，特别在单片机运用中由于主函数没有其它函数调用它，所以返回的值也就没什么用。所以一般都写的 void main()，这时程序中不需要 return 语句，如果 main() 函数前没有 void（默认为 int），或者写为 int main() 程序中就必须有 return 语句，比如：

```
int main()
{
    return 0;    // 表示程序正常退出
}
```

在单片机程序中一般写作 void main() 最方便，但在其它一些 C 编译器中，写作 void main() 编译是不能通过的，需要写成 int main()，int main() 是 C 语言的标准格式。

1.7 printf 格式化输出函数

前面已经学会了使用 LED 显示程序运行结果，也学会了软件与硬件仿真，其实仿真一般是用在程序比较复杂或遇到一些奇怪的现象时才使用，简单的程序直接下载到单片机观察最终运行结果会更省事一点，但目前我们只会使用 LED 显示，显示的信息太少了，为此，我们学习使用计算机显示器显示单片机运行过程中的更多信息。

例 1.8 计算机串口助手显示单片机内部简单信息

```
#include "STC15W4K.H"
#include <stdio.h>    // 为使用 KEIL 自带的库函数 printf 而加入
void printstar()
{
    printf("*****\n");
}
```

```

void print_message()
{
    printf("hello world");           // 最简单输出
    printf("How do you do!\n");     // 输出换行符\n
    printf("欢迎学习 STC51 单片机\n"); // 中文输出
}

void UART_init(void)
{
    // 下面代码设置定时器 1
    TMOD = 0x20;    // 0010 0000 定时器 1 工作于方式 2 (8 位自动重装方式)
    TH1 = 0xFD;    // 波特率: 9600 /11.0592MHZ
    TL1 = 0xFD;    // 波特率: 9600 /11.0592MHZ
    TR1 = 1;
    // 下面代码设置定串口
    AUXR = 0x00;    // 很关键, 使用定时器 1 作为波特率发生器, S1ST2=0
    SCON = 0x50;    // 0101 0000 SM0.SM1=01(最普遍的 8 位通信), REN=1 (允许接受)
    TI=1;           // 很关键, 使用 printf 函数时必须要有此命令
}

void main()
{
    UART_init();    // 初始化串口
    printstar();    // 输出*****
    print_message(); // 输出说明文字
    printstar();    // 输出*****
    while(1) ;     // 停在这里
}

```

我们把例 1.8 程序下载到单片机中, 打开程序下载软件的串口助手, 接收缓冲区选择文本模式, 波特率 9600, 打开串口, 给实验板断电后上电, 可以看到单片机发给计算机的信息如图 1-67 所示, 如果显示的个别字符出现乱码或连续接收大量数据显示不正常, 可换用其它串口助手软件, 比如“丁丁串口调试助手 SSCOM 3.3”即可解决。



图 1-67 串口助手显示单片机信息

例 1.9 计算机串口助手显示单片机内部变量值

```
#include "STC15W4K.H"
```

```

#include <stdio.h> // 为使用 KEIL 自带的库函数 printf 而加入
void printf_char_int_long(void)
{
    char a=-100;
    int b=-2000;
    long c=6553600;
    printf ("char %bd int %d long %ld\n", a, b, c); // 10 进制输出
    // 输出带符号 10 进制整数（正数不输出符号）
    // 实际输出: char -100 int -2000 long 6553600
    printf ("char_0x%bx int_0x%x long_0x%lx\n", a, b, c); // 16 进制输出
    // 输出无符号 16 进制整数，x 表示按小写输出，X 表示按大写输出
    // 实际输出: char_0x9c int_0xf830 long_0x640000
    printf ("a_int %d\n", (int)(a)); // 不用宽度标识符
    // 实际输出: a_int -100
    printf ("char %bd, int %d, long %ld\n", a, b, c); // %bd 后也可以有其它普通字符
    // 输出带符号 10 进制整数（正数不输出符号）
    // 实际输出: char -100, int -2000, long 6553600
}

void printf_float(void)
{
    float a= 3.14159265798932;
    float f = 10.0, g = 22.95;
    printf("Max is %f\n", a); // Max is 3.141593, 默认小数点后 6 位
    printf("%.4f\n", a); // 3.1416
    printf("%.12f\n", a); // 3.140593000000
    //%.4f 表示按小数点后 4 位数据输出，%.12f 表示按小数点后 12 位数据输出
    printf ("%f , %g\n", f, g); //输出: 10.000000 , 22.95
}

void printf_String(void)
{
    char buf [] = "Test String";
    char *p = buf;
    printf ("String %s is at address %p\n", buf, p);
    // 输出: String Test String is at address i: 0022
}

void UART_init(void)
{
    // 同例 1.8
}

void main()
{
    UART_init(); // 初始化串口
}

```

```

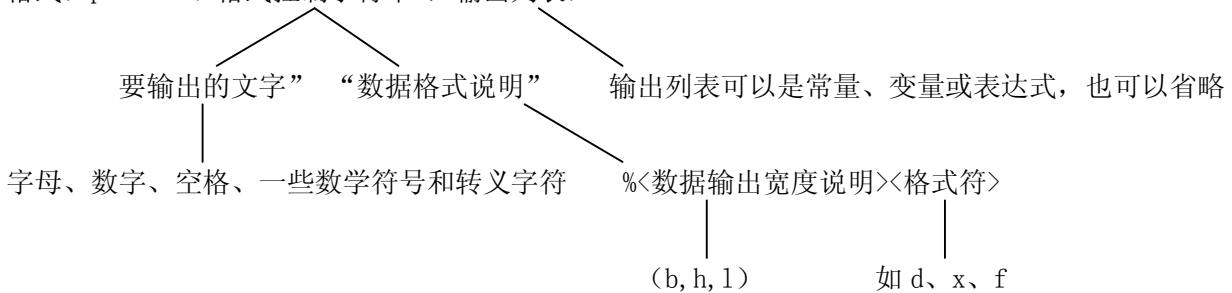
printf_char_int_long();
printf_float();
printf_String();
while(1); // 停在这里
}

```

Keil C51 中 printf 与标准 C 是有区别的，Keil 里扩展出了 b, h, l 来对输入字节宽度设置：b 表示 8 位，h 表示 16 位（默认值，可省略标识符），l 表示 32 位，如果没有宽度标识符，除整型数据（int 或 unsigned int 型）外，其余类型都会出现错误，如果不用宽度标识符，也可使用强制类型转换的方法，将 char 或 unsigned char 的变量强制转换成 int 或 unsigned int，最终实现效果与使用宽度标识符 b、h、l 完全相同。

库函数 printf 基本格式如下，我们可以简单的认为：“格式控制字符串”是要输出的字符串，内部包含的 %d 由输出列表中的变量一一对应代换。

格式：printf（“格式控制字符串”，输出列表）



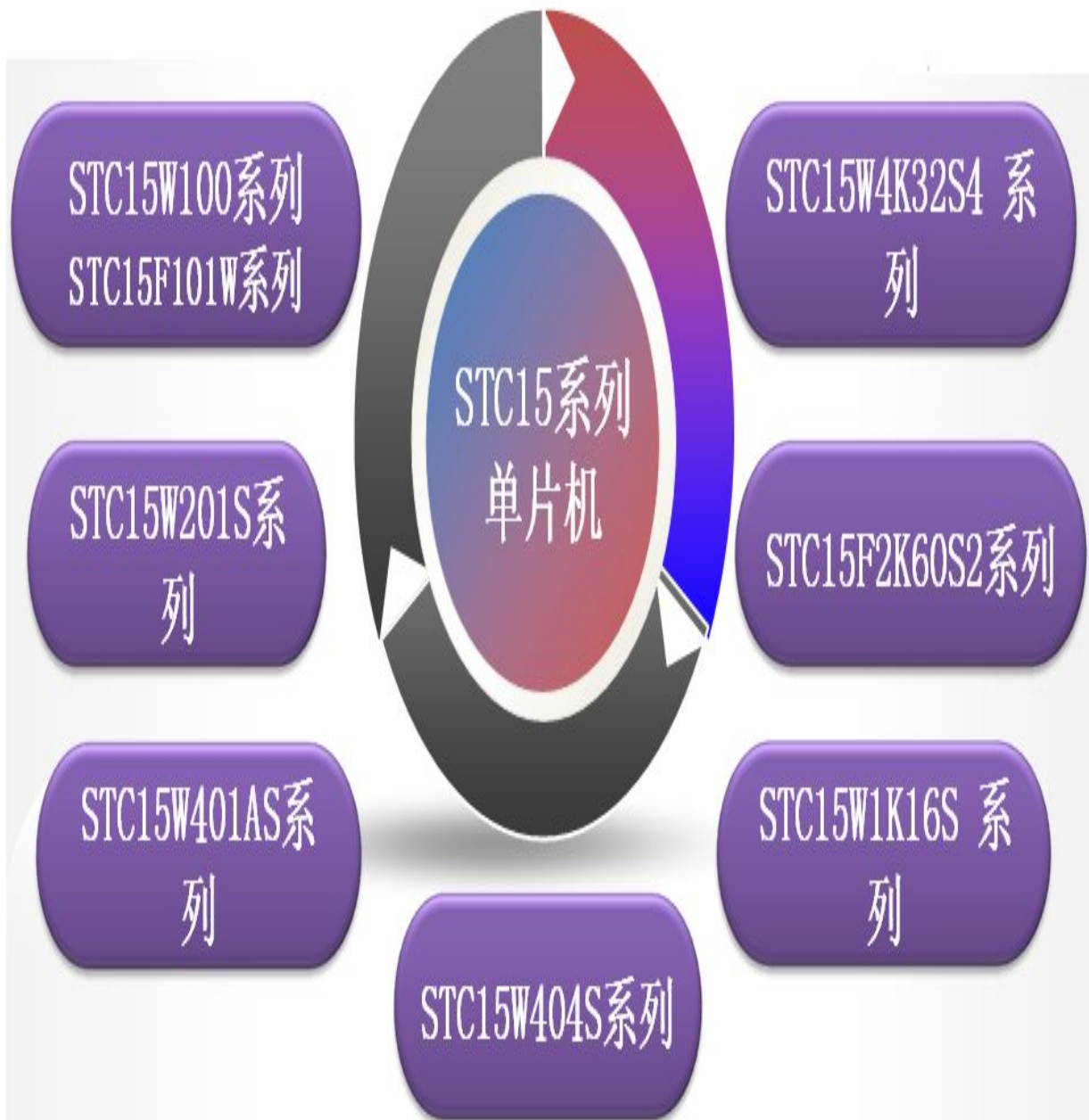
格式控制字符串：由“要输出的文字”和“数据格式说明”组成，“要输出的文字”可以使用字母、数字、空格、一些数学符号和转义字符，常用转义字符见表 1-4，“数据格式说明”由 % 开头，一个 % 要对应一个输出列表元素，printf 的“数据格式说明”的完整格式：% b 或 h 或 l 格式字符，格式字符见表 1-5。提示一下，<数据输出宽度说明><格式符>的 <> 表示此项内容可能实际不需要，本书后面部分的内容也遵从这个约定。

表 1-4 常用转义字符

转义字符	意义
\n	回车换行符，光标移到下一行行首 ▲▲▲▲▲（用得最多）
\t	横向跳格（每 8 位为一格，光标跳到下一格起始位置，如 9 或 17 位等
\\	用于输出反斜杠字符“\”
\'	用于输出单引号字符“'”
\”	用于输出双引号字符“””

表 1-5 格式字符

d	输出带符号 10 进制整数（正数不输出符号） ▲▲▲▲▲（用得最多）
x, X	输出无符号 16 进制整数，x 表示按小写输出，X 表示按大写输出
c	输出单个字符
s	输出字符串
f	输出小数，并取到小数点以下六位，四舍五入。
e, E	输出指数形式小数
g, G	浮点格式和指数格式中自动选择一种格式较短的格式输出
p	输出地址。 例如：char a=123;printf(“%p”, &a);
%	输出 %。 例如：printf(“%%”); 输出结果：%



本书特色：

1. 内容真实、言语简洁、通俗易懂。
2. 讲解功能强大的最新主流芯片，学会后即可用于产品开发。
3. 配套视频教程可辅助参考。
4. 提供作者邮箱答疑、QQ 在线答疑、国内知名网站单片机论坛答疑。