

目录

第 1 章 实验仪功能简介	5
1.1 电路外观	5
1.2 功能特点	6
1.3 实验项目	7
1.4 主要功能块	8
1.5 TKSMonitor51 仿真器说明	9
1.5.1 DP-51PROC 下载工作方式 (load)	9
1.5.2 DP-51PROC 运行工作方式 (run)	9
第 2 章 DP-51PROC 快速入门	10
2.1 Keil C51 简介	10
2.2 Keil C51 的安装	11
2.2.1 系统要求	11
2.2.2 软件的安装	12
2.3 μ Vision2 集成开发环境	14
2.3.1 μ Vision2 集成工具	14
2.3.2 菜单栏命令、工具栏和快捷方式	15
2.4 Keil C51 的使用	20
2.4.1 创建第一个 Keil C51 应用程序	20
2.4.2 程序文件的编译、链接	27
2.5 调试仿真功能的使用	31
2.5.1 如何进入调试状态	31
2.5.2 调试状态的存储器模型	32
2.5.3 调试前的准备工作	33
2.5.4 实战	35
2.6 脱机运行之 Flash 运行	40
2.6.1 如何进入运行状态	41
2.6.2 运行状态的存储器模型	41
2.6.3 运行程序实例	41
2.7 脱机运行之 ISP 单片机运行	43
2.7.1 如何进入 ISP 下载状态	43
2.7.2 运行状态的存储器模型	44
2.7.3 ZLGISP 软件简介	44
2.7.4 ZLGISP 软件的安装方法	44
2.7.5 ZLGISP 软件的使用方法	46
2.8 各功能模块的功能介绍	49
2.8.1 A1 区 ISP 下载电路	49

2.8.2	A2 区 MCU 总线接口及 IO 口连接区	50
2.8.3	A3 区 138 译码电路	51
2.8.4	A4 区并转串实验电路	51
2.8.5	A5 区串转并实验电路	51
2.8.6	A6 和 A7 区 PARK 扩展	52
2.8.7	B1 区语音实验区	52
2.8.8	B2 区非接触式 IC 卡实验区	53
2.8.9	B3 区 LCD 实验区	53
2.8.10	B4 区数字温度采集实验区	54
2.8.11	B5 区蜂鸣器实验区	55
2.8.12	B6 区 PWM 电压转换实验区	55
2.8.13	B7 区电压基准源	55
2.8.14	B8 区串行模数转换实验区	56
2.8.15	B9 区串行数模转换实验区	56
2.8.16	B10 区直流电机实验区	57
2.8.17	C1 区电压接口区	58
2.8.18	C2 区逻辑笔	58
2.8.19	C3 区 LED 点阵实验模块	58
2.8.20	C4 区运算放大器电路实验区	59
2.8.21	C5 电阻接口区	59
2.8.22	C6 区 555 电路实验区	59
2.8.23	C7 区继电器及其驱动电路	60
2.8.24	C8 区步进电机实验区	60
2.8.25	D1 区独立控制的 LED、拨动开关、键盘实验区	61
2.8.26	D2 区电位器实验区	61
2.8.27	D3 区红外收发实验区	62
2.8.28	D4 区 RS-485 实验区	62
2.8.29	D5 区 I ² C 实验区	63
2.8.30	D6 区接触式 IC 卡实验区	63
第 3 章	DP-51PROC 单片机实验	64
实验一	Keil C51 集成开发环境的使用练习	64
实验二	基于 Keil C51 集成开发环境的仿真与调试	66
实验三	单片机 I/O 口控制实验	68
实验四	蜂鸣器驱动实验	71
实验五	电子琴实验	74
实验六	定时器输出 PWM 实验	77
实验七	串转并的 I/O 口实验	80
实验八	并转串的 I/O 口实验	83
实验九	74HC138 译码器实验	86
实验十	16×16 LED 扫描输出实验	88
实验十一	555 电路实验	95
实验十二	运算放大器实验	97
实验十三	继电器控制实验	100

实验十四	RS232 串口通信实验	103
实验十五	RS485 差分串行通信实验	106
实验十六	I ² C 总线实验（实时时钟、EEPROM 和 ZLG7290 的实验）	109
实验十七	万年历时钟实验	114
实验十八	接触式 IC 卡读写实验	117
实验十九	数字温度传感器实验	124
实验二十	单总线和 I ² C 总线结合实现数字温度计实验	130
实验二十一	结合 555 电路实验和单片机定时器频率计实验	137
实验二十二	直流电机实验	140
实验二十三	步进电机控制实验	142
实验二十四	红外收发实验	144
实验二十五	字符型液晶显示实验	147
实验二十六	图形液晶显示实验	151
实验二十七	串行模数转换实验	157
实验二十八	串行数模转换实验	160
实验二十九	IC 卡身份识别开关实验	163
实验三十	USB1.1 接口控制演示实验	166
实验三十一	CAN-bus 接口控制实验	168
实验三十二	USB2.0 接口控制演示实验	174
实验三十三	基于以太网接口的 TCP/IP 实验	176
实验三十四	ISD1420 语音模块实验	179
实验三十五	非接触式 IC 卡读卡模块实验	180
实验三十六	并行模数转换实验	183
实验三十七	并行数模转换实验	186
实验三十八	8155 并口扩展实验	188
实验三十九	8255 并口扩展实验	191
第 4 章 Small RTOS51 的应用		194
4.1	编写 Small RTOS51 的驱动程序	194
4.1.1	函数的可重入	194
4.1.2	驱动程序的编写方法	194
4.2	16×2 LCD 模块的驱动	195
4.2.1	点阵字符型 LCD-TC1602A	195
4.3	I ² C 总线驱动程序的实现	205
4.3.1	I ² C 驱动程序的简介	205
4.3.2	驱动程序的使用	205
4.3.3	基本 I ² C 总线信号的产生	206
4.3.4	I ² C 总线初始化	208
4.3.5	发送和接收一个字节	208
4.3.6	对 I ² C 进行读操作	211
4.3.7	对 I ² C 进行写操作	214
4.4	Small RTOS51 应用实例和分析	216
4.4.1	实例简介	216
4.4.2	系统配置文件 Os_cfg.h:	216

4.4.3 CPU 配置文件 Os_cpu.h:	218
4.4.4 ZLG7290 应用函数程序 zlg7290.c	221
4.4.5 主程序 EXT1.c	224
第 5 章 Small RTOS51 实验	230
实验一 LED 和键盘扫描驱动程序演示实验	230
实验二 PCF8563 驱动程序演示实验	234
实验三 图形液晶显示驱动实验	238
实验四 软定时器实验	242
实验五 串口驱动程序实验	252
实验六 CAT24WC02 驱动程序演示实验	259
实验七 PDIUSB12 USB 驱动程序演示实验	265
实验八 SJA1000_CAN 驱动程序演示实验	272
附录 Small RTOS51 使用许可协议	278

第 1 章 实验仪功能简介

本章介绍 DP-51PROC 单片机综合仿真实验仪的硬件信息，让您对它的功能有一个大概的了解，对后面的系统电路实验具有必不可少的帮助。

1.1 电路外观

DP-51PROC 单片机综合仿真实验仪的电路布局如图 1.1 所示。

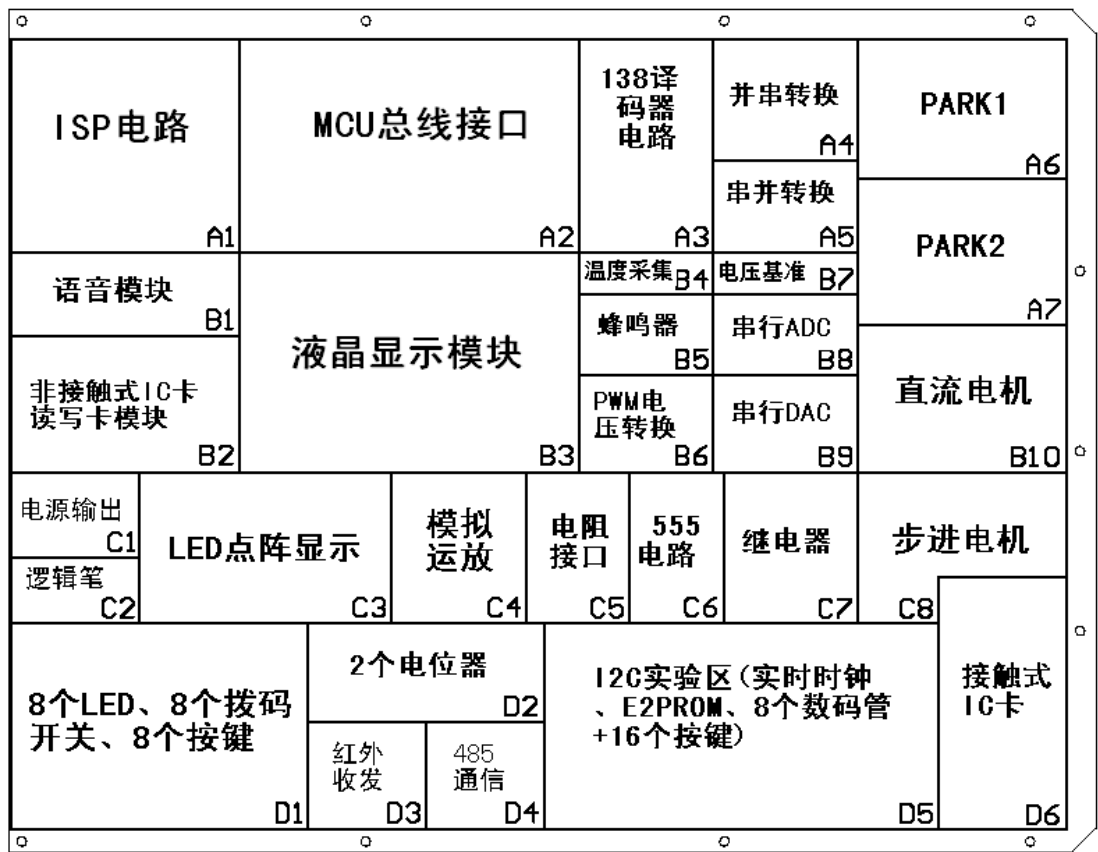


图 1.1 DP-51PROC 单片机综合仿真实验仪

由图 1.1 可以看出，它分为很多个功能块，各个功能块之间是相对独立的，每个功能块都有一个编号分别是竖数 A~D，横数 1~10。我们可以从编号，快速的找到功能块所在的位置。如 C3 功能块，就是第 3 行的第 3 个功能块，这样用户就可以比较方便的找到对应

的位置。

1.2 功能特点

DP-51PROC 单片机综合仿真实验仪集成有强大的硬件资源，并且为用户提供了多种选择，使用用户可以进行各种相关的实验。

1. 自带 5V、12V、-12V 电源，其中 5V 电源可提供 1A 电流，12V 可提供 500mA，-12V 可提供 500mA，含瞬时短路保护和过流保护；
2. 基于 KEIL MONITER 51 的仿真调试功能（使用 TKSMonitor51 仿真器）；
3. TKSMonitor51 仿真器内部带有 32KB 的 FLASH 用于用户的程序下载；
4. 能够实现 PHILIPS 单片机的 64KB FLASH 的 ISP 下载编程功能；
5. 灵活简单的 138 译码和 573 锁存电路，方便用户随意设置；
6. 集成 1 路完全功能的 CAN-bus 现场总线接口（可供用户选配）；
7. 集成 1 路 USB1.1 接口；
8. 集成 1 路 USB2.0 接口（可供用户选配）；
9. 集成 1 路 TCP/IP 以太网接口（可供用户选配）；
10. 支持 CPLD 实验，可选择使用 XILINX 公司的 XC95108 系列的 CPLD 或者 ALTERA 公司的 EPM7128S 系列的 CPLD（可供用户任意选择，需要或不需要）；
11. 带有 128*64 的点阵液晶模块及接口，和一个 16*2 字符型液晶模块的接口（可供用户任意选择）；
12. 16×16LED 点阵模块；
13. 步进电机、直流电机实验；
14. TLC549 串行 AD、TLC5620 串行 DA 实验；
15. 555 实验电路；
16. 由键盘显示芯片 ZLG7290 控制的 8 个 8 段数码管和 16 个按键；
17. 8 个拨码开关、8 个 LED、8 个独立的按键；
18. 接触式 IC 卡实验；
19. 非接触式 IC 卡读卡模块实验（可供用户选配）；
20. LM324 四运放，可以搭建各种运放电路，做运放实验；
21. 继电器驱动及控制电路，可做各种继电器控制实验；
22. I²C 接口的 EEPROM 和 RTC 实时时钟电路；
23. RS232 和 RS485 接口电路；
24. 交流蜂鸣器驱动控制电路；
25. PWM 脉宽调制输出接口；
26. 电位器电压调节电路；
27. 提供仿真器电源输出供给电压；
28. 74LS164 串转并、74LS165 并转串实验；
29. 红外收发数据实验；
30. 18B20 单总线数字式温度传感器实验；
31. ISD1420 语音模块实验；

- 32. 含有一个逻辑笔，可用于检查 TTL 电平的高低；
- 33. 包含有一个 8 路输出的时钟源。

1.3 实验项目

DP-51PROC 单片机综合仿真实验仪可以进行各种单片机实验，具体包括：

- 1. 单片机 I/O 口控制实验，如拨码开关信号输入，LED 发光二极管控制，按键输入等实验；
- 2. 定时器输出 PWM 实验；
- 3. 蜂鸣器驱动实验；
- 4. 结合单片机 I/O 口控制实验和蜂鸣器驱动实验的电子琴实验；
- 5. 串转并的 I/O 口实验；
- 6. 并转串的 I/O 口实验；
- 7. 74HC138 译码器实验；
- 8. 16×16LED 扫描输出实验；
- 9. 555 电路实验（如脉冲输出，频率调整等实验）；
- 10. 运算放大器实验（加减法，微积分等电路的实验）；
- 11. 继电器控制实验；
- 12. RS232 串口通信实验；
- 13. RS485 差分串行通信实验；
- 14. I²C 总线实验（实时时钟、EEPROM 和 ZLG7290 的实验）；
- 15. 结合 I²C 总线实验而扩展的万年历时钟实验；
- 16. 接触式 IC 卡读写实验；
- 17. 18B20 的单总线实验；
- 18. 结合 18B20 的单总线实验和 I²C 总线实验的温度计实验；
- 19. 结合 555 电路实验和单片机定时器频率计实验；
- 20. 直流电机实验；
- 21. 步进电机实验；
- 22. 红外收发实验；
- 23. LCD 16*2 字符型液晶显示实验；
- 24. LCD 128*64 点阵液晶显示实验；
- 25. 串行的模数转换实验；
- 26. 串行的数模转换实验；
- 27. 结合 I²C 总线实验、接触式 IC 卡读写实验、继电器控制实验的 IC 卡身份识别开关实验；
- 28. USB1.1 接口控制实验；
- 29. CAN-bus 接口控制实验（CAN PARK 可供用户选配）；
- 30. USB2.0 接口控制实验（USB2.0 PARK 可供用户选配）；
- 31. 基于以太网接口的 TCP/IP 实验（RTL8019AS PARK 可供用户选配）；
- 32. ISD1420 语音模块实验（ISD1420 语音模块可供用户选配）；
- 33. 非接触式 IC 卡读卡模块实验（ZLG500A 读卡模块及天线可供用户选配）；

34. 一系列 CPLD 综合实验 (CPLD PARK 可供用户选配);
 35. 基于 Keil C51 源码公开的 Small RTOS 嵌入式操作系统的实验、驱动开发与实战例程。

1.4 主要功能块

DP-51PROC 单片机综合仿真实验仪上的功能块如表 1.1 所示。

编号	功能块名称	功能说明
A1	ISP 下载电路	实现 PHILIPS 单片机的 64Kflash 的 ISP 下载编程功能。另外, RS232 串口通讯实验也用这个功能块来进行。
A2	总线 I/O 扩展区	该扩展区主要功能是把单片机的各功能管脚引出来, 方便用户选择使用各个 I/O 口或单片机总线。该扩展区还包含了一个 74HC573 对单片机的 P0 口进行锁存, 并扩展输出 A0~A7 总线地址。
A3	138 译码电路区	该区包含一片 74LS138 译码芯片
A4	并转串实验区	该区包含一片 74LS165 并转串芯片
A5	串转并实验区	该区包含一片 74LS164 串转并芯片
A6	PARK1	用于扩展连接各种扩展 PARK 模块, 包括 USB1.0、CAN-bus、USB2.0、以太网接口等 (其中 USB2.0 和以太网接口是选配的扩展 PARK 模块), 来进行相关的实验。它只能扩展一个 PARK 模块。
A7	PARK2	功能同 A4 区, DP-51PROC 单片机综合仿真实验仪可以同时 A4 和 A5 区分别扩展一个 PARK 模块, 同时进行两个 PARK 模块的实验。比如 A4 区扩展 USB1.0, A5 区扩展 CAN-bus, 这样用户就可以进行 USB 转 CAN-bus 的桥接实验了。
B1	语音模块	该区有一个 ZLG1420A 语音模块, 还有麦克风和扬声器, 用户可以在上面进行语音实验。
B2	非接触式 IC 卡读卡模块	该区有一个 ZLG500A 非接触式 IC 卡读卡模块接口 (ZLG500A 模块为选配件), 和相关的天线接口 (天线也是选配件), 用户可以利用该接口进行非接触式 IC 卡的实验, 在该区还有一个时钟源电路和 8 路分频输出接口。另外用户还可以选择在 B1 和 B2 区的扩展孔上扩展一个 CPLD 实验模块, CPLD 实验模块包括有 XILINX 的 XC95108 模块和 ALTERA 的 EPM7128 模块两种 (均为选配件) 以供用户选择, 进行 CPLD 的扩展实验。
B3	LCD 模块	该区包含有一个 LCD 液晶模块, 用户可以选择 128×64 的点阵图形液晶模块或者 16×2 的点阵字符液晶模块
B4	温度传感器区	该区包含一片 18B20 单总线 (1-Wire) 的数字温度传感器
B5	蜂鸣器区	该区包含一个交流蜂鸣器及其驱动电路
B6	PWM 输出实验区	该区把用户提供的 PWM 信号转换成电压输出
B7	电压基准源电路	该区提供一个 TL431 电压基准源电路。
B8	串行 AD 实验区	该区包含有一片 TLC549 8 位串行 AD 转换器
B9	串行 DA 实验区	该区包含有一片 TLC5620 8 位 4 通道串行 DA 转换器

B10	直流电机实验区	该区包含有一个可调速的直流电机及其驱动电路
C1	电源输出接口区	该区包含+5V、-12V、+12V 电源接口，方便用户外接使用
C2	逻辑笔电路	该区是一个检查 TTL 逻辑电平高低的逻辑笔，通过 LED 显示所检查电路的电平高低
C3	16×16 点阵 LED 模块	该区包含一个 16×16 点阵 LED 模块及其驱动电路
C4	运放实验区	该区包含一片 LM324 芯片
C5	电阻接口	该区为运算放大器提供电阻接口电路
C6	555 实验区	该区包含一片 555 芯片及相关的电阻、电容接口电路
C7	继电器实验区	该区包含一个继电器及其驱动电路
C8	步进电机实验区	该区包含有一个步进电机及其驱动电路
D1	I/O 实验区	该区域分别包含 8 个独立的 LED 发光二极管、拨动开关、按键
D2	可调电阻区	该区包含一个 10K 欧姆和一个 1K 欧姆的可调电阻
D3	红外收发区	该区包含一个红外发射管和一个带解码的红外接收器
D4	RS485 实验区	该区包含一片 SP485 芯片，用于 RS485 的电平驱动和接收
D5	I ² C 实验区	该区包含一片 24WC02 256 字节的 EEPROM，一片 PCF8563 实时时钟芯片及外围电路，一片 ZLG7290 键盘 LED 驱动芯片及 8 段 8 位数码管和 16 个按键。
D6	接触式 IC 卡实验区	该区包含一个可连接 SLE4442 卡的读卡头。

1.5 TKSMonitor51 仿真器说明

在 DP-51PROC 单片机综合仿真实验仪中为用户配备了一个 TKSMonitor51 仿真器，它实质上是一个基于 Keil 环境下的 MON51 仿真调试器，并且配备了一个装载用户程序的空间，可以脱离计算机使用仿真器运行用户程序，这就相当于一个单片机在运行用户程序一样，当用户将 HEX 文件下载入 TKSMonitor51 仿真器后，用户将仿真头插入锁紧座就相当于在锁紧座上插入了一块烧写好程序的单片机。它的工作方式有两种：下载工作方式和运行工作方式。下面将给予详细的说明。

1.5.1 DP-51PROC 下载工作方式 (load)

把拨动开关拨到 LOAD 一边就进入下载工作方式了。在该工作方式下，用户可以下载程序到 FLASH 中。

1.5.2 DP-51PROC 运行工作方式 (run)

把拨动开关拨到 RUN 一边就进入运行工作方式了。在该工作方式下用户可以跑下载到 FLASH 中的程序，也可以把 MON51 下载到 FLASH 中，然后使用 KEIL 来调试用户程序。

第 2 章 DP-51PROC 快速入门

DP-51PROC 单片机综合仿真实验仪是由广州致远电子有限公司设计的基于 Keil C51 集成开发环境下的 DP 系列单片机仿真实验仪之一，是一套功能强大的单片机应用技术学习、调试、开发工具，为各大院校的实践教学提供了一个较好的实验平台，是单片机教学的好帮手。

DP-51PROC 单片机综合仿真实验仪向用户提供了丰富的外围器件和设备接口，可使用户快速掌握单片机原理及其实用接口技术。同时，基于 Keil C51 集成开发环境下的 TKSMonitor51 仿真器具有硬件仿真的功能，用户可以在 Keil uVision2 环境下学习编写、调试单片机程序，是一套性能完美的 MCU 综合实验系统。通过学习，实验用户可以掌握运用单片机进行项目开发的过程、步骤和方法，积累一定的调试方法、技巧。在学习使用 DP-51PROC 单片机综合仿真实验仪前，用户有必要了解 Keil C51 集成开发环境。

2.1 Keil C51 简介

Keil C51 μ Vision2 集成开发环境是 Keil Software, Inc/Keil Elektronik GmbH 开发的基于 80C51 内核的微处理器软件开发平台，内嵌多种符合当前工业标准的开发工具，可以完成从工程建立到管理，编译，连接，目标代码的生成，软件仿真，硬件仿真等完整的开发流程。尤其 C 编译工具在产生代码的准确性和效率方面达到了较高的水平，而且可以附加灵活的控制选项，在开发大型项目时非常理想。Keil C51 集成开发环境的主要功能有以下几点：

- μ Vision2 for WindowsTM：是一个集成开发环境，它将项目管理、源代码编辑和程序调试等组合在一个功能强大的环境中；

- C51 国际标准化 C 交叉编译器：从 C 源代码产生可重定位的目标模块；
- A51 宏汇编器：从 80C51 汇编源代码产生可重定位的目标模块；
- BL51 连接/定位器：组合由 C51 和 A51 产生的可重定位的目标模块，生成绝对目标模块；
- LIB51 库管理器：从目标模块生成链接器可以使用的库文件；
- OH51 目标文件至 HEX 格式的转换器：从绝对目标模块生成 Intel HEX 文件；
- RTX-51 实时操作系统：简化了复杂的实时应用软件项目的设计。

这个工具套件是为专业软件开发人员设计的，但任何层次的编程人员都可以使用，并获得 80C51 微控制器的绝大部分应用。

Keil Software 提供了一流的 80C51 系列开发工具软件，下面描述每个套件及其内容：

- PK51 专业开发套件

PK51 专业开发套件提供了所有工具，适合专业开发人员建立和调试 80C51 系列微控制器的复杂嵌入式应用程序。专业开发套件可针对 80C51 及其所有派生系列进行配置使用。

- DK51 开发套件

DK51 开发套件是 PK51 的精简版，它不包括 RTX51 Tiny 实时操作系统。开发套件

可针对 80C51 及其所有派生系列进行配置使用。

- CA51 编译器套件

如果开发者只需要一个 C 编译器而不需要调试系统，则 CA51 编译器套件就是最好的选择。CA51 编译器套件只包含 μ Vision2 IDE 集成开发环境，CA51 不提供 μ Vision2 调试器的功能。这个套件包括了要建立嵌入式应用的所有工具软件，可针对 80C51 及其所有派生系列进行配置使用。

- A51 汇编器套件

A51 汇编器套件包括一个汇编器和创建嵌入式应用所需要的所有工具。它可针对 80C51 及其所有派生系列进行配置使用。

- RTX51 实时操作系统(FR51)

RTX51 实时操作系统是 80C51 系列微控制器的一个实时内核。RTX51 Full 提供 RTX51 Tiny 的所有功能和一些扩展功能，并且包括 CAN 通讯协议接口子程序。

- 比较表：表 2.1 列出了每个套件的功能，表的顶上一栏为工具套件名称，表的左边一列为软件组成部分，使用这个对照表可以选择符合您需要的套件。

表 2.1 比较表

部件	PK51	DK51 †	CA51	A51	FR51
μ Vision2 项目管理和编辑器	√	√	√	√	
A51 汇编器	√	√	√	√	
C51 编译器	√	√	√		
BL51 链接器/定位器	√	√	√	√	
LIB51 库管理器	√	√	√	√	
μ Vision2 调试器/模拟器	√	√			
RTX51 Tiny	√				
RTX51 Full					√

DP-51PROC 单片机综合仿真实验仪在自身强大硬件功能基础之上集成了 Keil C51 开发调试软件，使用户能够较轻松、快捷的掌握硬件设计方法和软件编程技巧。为了使您在较短的时间内熟悉和掌握这些技术，随机提供了测试版 Keil C51 V7.0 软件，其中包含了一些示范程序和受限制的工具。Keil C51 测试版工具软件在功能、创建应用程序和程序代码长度方面都有限制，对于大型应用程序或需要功能更全的 Keil C51 软件，则需要购买正版的 Keil C51 软件。

2.2 Keil C51 的安装

2.2.1 系统要求

安装 Keil C51 集成开发软件，必须满足最小的硬件和软件要求，才能确保编译器以及其他程序功能正常，必须具有：

- (1) Pentium、Pentium—II 或兼容处理器的 PC；
- (2) Windows95、Windows98、Windows NT4.0；

- (3) 至少 16MB RAM;
- (4) 至少 20MB 硬盘空间。

2.2.2 软件的安装

下面以 Keil C51 V7.0 版为例, 介绍如何安装 Keil μ Vision2 集成开发环境。

(1) 把随机赠送的 DEMO 光盘放入光驱中(假设 E: 盘), 进入 E:\Software\Keil C V7.0\Setup 目录下, 这时会看到 SETUP.EXE 的安装文件, 双击该文件即可开始安装。

(2) 这时会出现如图 2.1 所示的安装初始化画面, 稍后弹出一个安装向导对话框如图 2.2 所示, 询问用户是安装、修复更新或是卸载 Keil C51 软件, 用户可以根据需要进行选择, 当然若是第一次安装该软件应选择第一项 **Install Support for Additional...** 安装该软件。

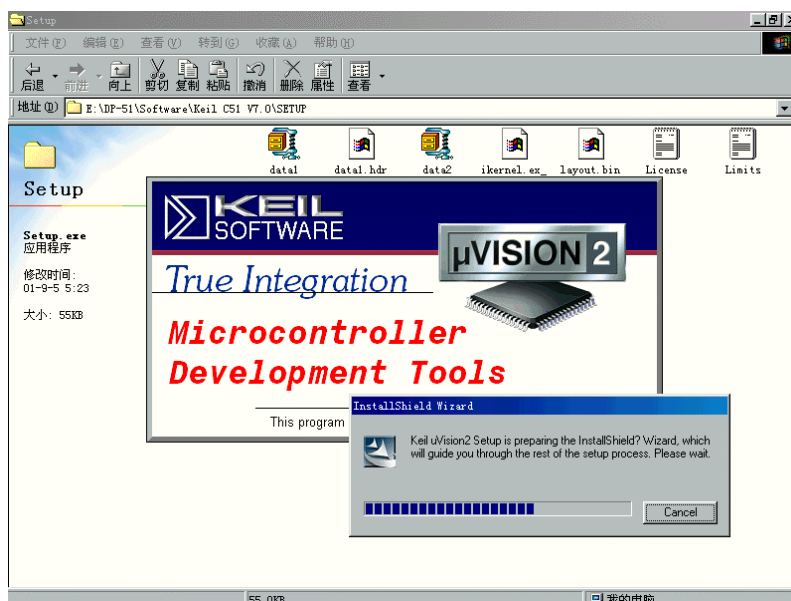


图 2.1 安装初始化

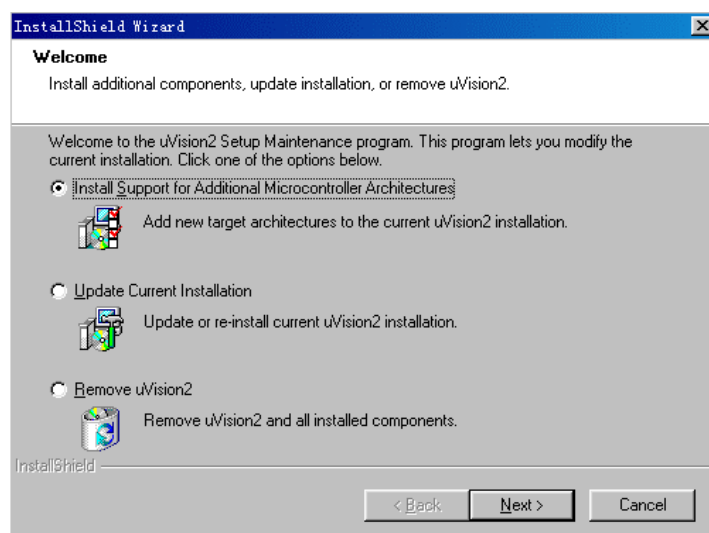


图 2.2 安装向导画面

(3) 单击 **Next** 命令按钮, 这时会出现如图 2.3 所示的安装询问对话框, 提示用户是安装完全版还是评估版。如果您购买了正版的 Keil C 软件当然是选择 **Full Version** 了, 否则您只能选择 **Eval Version** 选项。



图 2.3 安装询问画面

(4) 在此后弹出几个确认对话框中选择 **Next**，这时会出现一个如图 2.4 所示的安装路径设置对话框，默认路径是 C:\KEIL，当然用户可以点击 **Browse** 选择适合自己安装的目录，如 D:\Keil C51 V7.0。

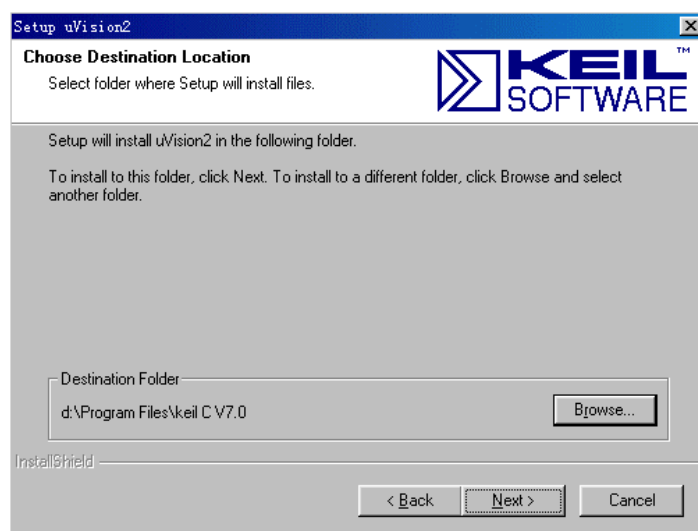


图 2.4 安装路径设置对话框

(5) 在接下来的询问确认对话框中选择 **Next** 命令按钮加以确认即可出现如图 2.5 所示的安装进度指示画面。

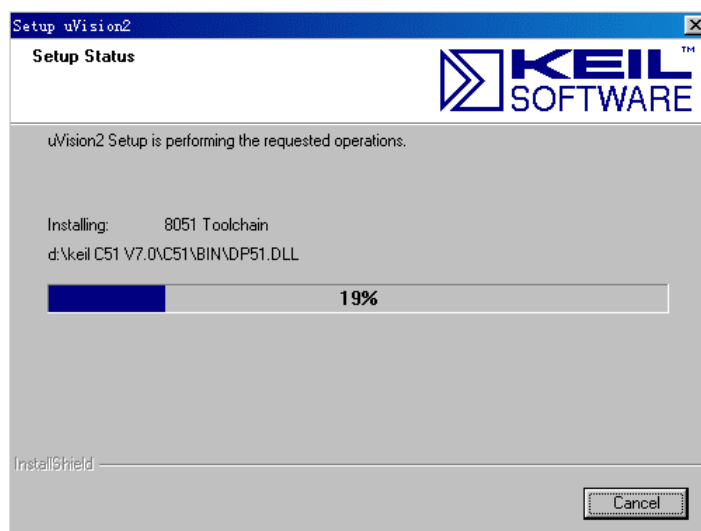


图 2.5 安装进度指示画面

(6) 接下来就是等待安装，安装完毕后单击 **Finish** 加以确认，此时您可以在桌面上看

到 Keil μ Vision2 软件的快捷图标如图 2.6 所示，双击它就可以进入 Keil C51 集成开发环境。



图 2.6 快捷图标

2.3 μ Vision2 集成开发环境

2.3.1 μ Vision2 集成工具

μ Vision2 支持所有的 Keil 80C51 的工具软件，包括 C51 编译器、宏汇编器、链接器/定位器和目标文件至 Hex 格式转换器， μ Vision2 可以自动完成编译、汇编、链接程序等操作。

(1) C51 编译器和 A51 汇编器

由 μ Vision2 IDE 创建的源文件，可以被 C51 编译器或 A51 汇编器处理，生成可重定位的 object 文件。Keil C51 编译器遵照 ANSI C 语言标准，支持 C 语言的所有标准特性。另外，还增加了几个可以直接支持 80C51 结构的特性。Keil A51 宏汇编器支持 80C51 及其派生系列的所有指令集。

(2) LIB51 库管理器

LIB51 库管理器可以从由汇编器和编译器创建的目标文件建立目标库。这些库是按规定格式排列的目标模块，可在以后被链接器所使用。当链接器处理一个库时，仅仅使用了库中程序使用了的目标模块而不是全部加以引用。

(3) BL51 链接器/定位器

BL51 链接器使用从库中提取出来的目标模块和由编译器、汇编器生成的目标模块，创建一个绝对地址目标模块。绝对地址目标文件或模块包括不可重定位的代码和数据。所有的代码和数据都被固定在具体的存储器单元中。

(4) μ Vision2 软件调试器

μ Vision2 软件调试器能十分理想地进行快速、可靠的程序调试。调试器包括一个高速模拟器，您可以使用它模拟整个 80C51 系统，包括片上外围器件和外部硬件。当您从器件数据库选择器件时，这个器件的属性会被自动配置。

(5) μ Vision2 硬件调试器

μ Vision2 调试器向您提供了几种在实际目标硬件上测试程序的方法。安装 MON51 目标监控器到您的目标系统，并通过 Monitor—51 接口下载您的程序；使用高级 GDI 接口，将 μ Vision2 调试器同类似于 DP-51PROC 单片机综合仿真实验仪或者 TKS 系列仿真器的硬件系统相连接，通过 μ Vision2 的人机交互环境指挥连接的硬件完成仿真操作。

(6) RTX51 实时操作系统

RTX51 实时操作系统是针对 80C51 微控制器系列的一个多任务内核。RTX51 实时内核简化了需要对实时事件进行反应的复杂应用的系统设计、编程和调试。这个内核完全集成在 C51 编译器中，使用非常简单。任务描述表和操作系统的一致性由 BL51 链接器/定位器自动进行控制。

此外 μ Vision2 还具有极其强大的软件环境、友好的操作界面和简单快捷的操作方法，其主要表现在以下几点。

- 丰富的菜单栏；
- 可以快速选择命令按钮的工具栏；
- 一些源代码文件窗口；
- 对话框窗口；
- 直观明了的信息显示窗口。

2.3.2 菜单栏命令、工具栏和快捷方式

安装 Keil C51 软件后，点击桌面 Keil C51 快捷图标即可进入如图 2.7 所示的集成开发环境，各种调试工具、命令菜单都集成在此开发环境中。其中菜单栏为您提供各种操作菜单，比如：编辑器操作、工程维护、开发工具选项设置、程序调试、窗体选择和操作、在线帮助。工具栏按钮可以快速执行 μ Vision2 命令，快捷键(您可以自己配置)也可以执行 μ Vision2 命令。

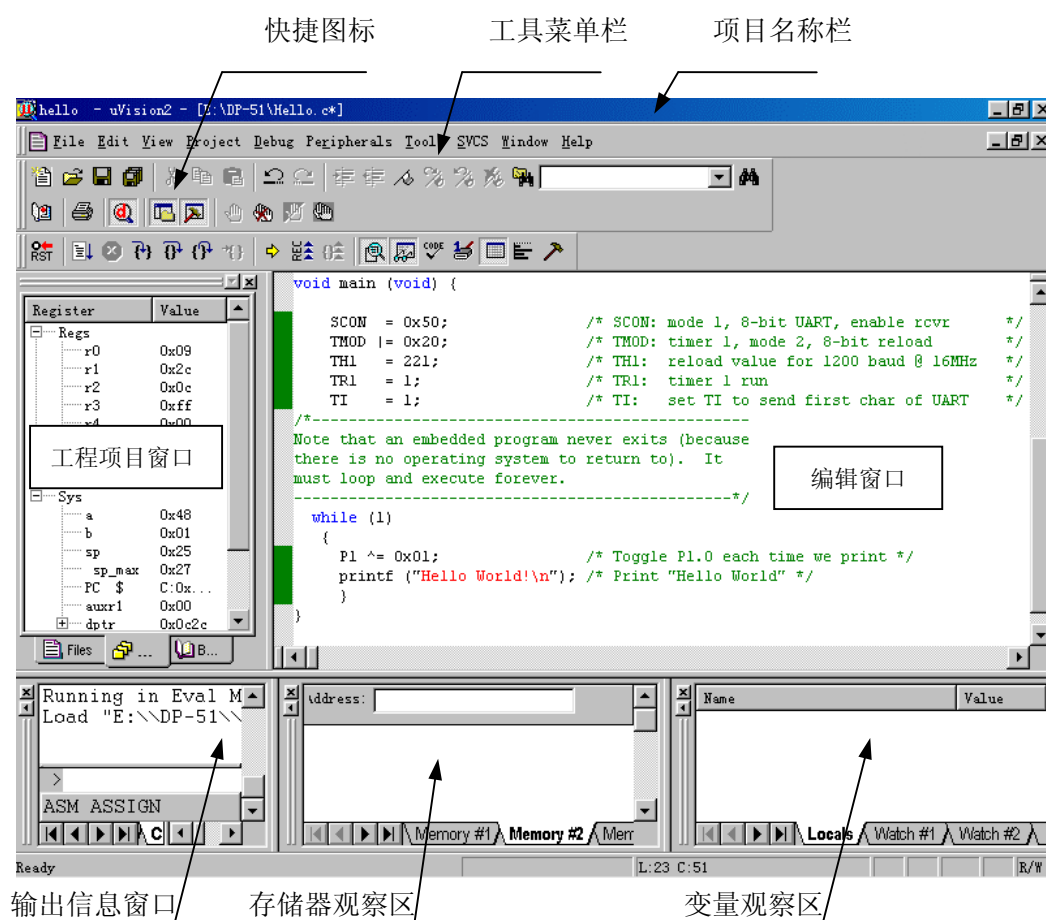



图 2.7 μ Vision2 操作界面

(1) 文件菜单和文件命令(File)

File 菜单	工具栏	快捷键	描述
New		Ctrl+N	创建一个新的源文件或文本文件
Open		Ctrl+O	打开已有的文件
Close			关闭当前的文件
Save		Ctrl+S	保存当前的文件
Save All			保存所有打开的源文件和文本文件
Save as...			保存并重新命名当前的文件
Device Database			维护μVision2 器件数据库
Print Setup...			设置打印机
Print		Ctrl+P	打印当前的文件
Print Preview			打印预览
1~9			打开最近使用的源文件或文本文件
Exit			退出μVision2，并提示保存文件

(2) 编辑菜单和编辑器命令(Edit)

Edit 菜单	工具栏	快捷键	描述
		Home	将光标移到行的开始处
		End	将光标移到行的结尾处
		Ctrl+Home	将光标移到文件的开始处
		Ctrl+End	将光标移到文件的结尾处
		Ctrl+←	将光标移到上一个单词
		Ctrl+→	将光标移到下一个单词
		Ctrl+A	选中当前文件中的所有文字
Undo		Ctrl+Z	撤销上一次操作
Redo		Ctrl+Shift+Z	重做上一次撤销的命令
Cut		Ctrl+X	将选中的文字剪切到剪贴板
		Ctrl+Y	将当前行的文字剪切到剪贴板
Copy		Ctrl+C	将选中的文字复制到剪贴板
Paste		Ctrl+V	粘贴剪贴板的文字
Indent Selected Text			将选中的文字向右缩进一个制表符位
Unindent Selected Text			将选中的文字向左缩进一个制表符位
Toggle Bookmark		Ctrl+F2	在当前行放置书签
Goto Next Bookmark		F2	将光标移到下一个书签
Goto Previous Bookmark		Shift+F2	将光标移到上一个书签
Clear All Bookmarks			清除当前文件中的所有书签
Find		Ctrl+F	在当前文件中查找文字
		F3	继续向前查找文字
		Shift+F3	继续向后查找文字
		Ctrl+F3	查找光标处(选中)的单词
		Ctrl+]	查找匹配的花括号、圆括号、方括号 (使用这个命令时请将光标移到一个



Replace Find in Files...		Ctrl+H	花括号、圆括号或方括号的前面) 替换特定的文字 在几个文件中查找文字
-----------------------------	---	--------	--

(3) 选择文本命令

在µVision2 中，您可以按下 Shift 键和相应的光标键来选择文字。例如，Ctrl+→是将光标移到下一个单词，而 Ctrl+Shift+→是选中从光标的位置到下一个单词开始前的文字，您也可以鼠标选择文字。


选择...	鼠标要...
任何数量的文字	在文字上拖动
一个单词	双击这个单词
一行文字	将鼠标移到行的左边，直到它变成一个指向右的箭头，然后点击
多行文字	将鼠标移到行的左边，直到它变成一个指向右的箭头，然后向上或向下拖动鼠标
垂直的一块文字	按着 Alt 键，然后拖动

(4) 视图菜单(View)

View 菜单	工具栏	快捷键	描述
Status Bar			显示或隐藏状态栏
File Toolbar			显示或隐藏文件工具栏
Build Toolbar			显示或隐藏编译工具栏
Debug Toolbar			显示或隐藏调试工具栏
Project Window			显示或隐藏工程窗口
Output Window			显示或隐藏输出窗口
Source Browser			打开源(文件)浏览器窗口
Disassembly Window			显示或隐藏反汇编窗口
Watch & Call Stack Window			显示或隐藏观察和堆栈窗口
Memory Window			显示或隐藏存储器窗口
Code Coverage Window			显示或隐藏代码覆盖窗口
Performance Analyzer Window			显示或隐藏性能分析窗口
Symbol Window			显示或隐藏符号变量窗口
Serial Window #1			显示或隐藏串行窗口 1
Serial Window #2			显示或隐藏串行窗口 2
Toolbox			显示或隐藏工具箱
Periodic Window Update			在运行程序时，周期刷新调试窗口
Workbook Mode Options...			显示或隐藏工作簿窗口的标签 设置颜色、字体、快捷键和编辑器选项

(5) 工程菜单和工程命令(Project)

Project 菜单	工具栏	快捷键	描述
New Project...			创建一个新的工程
Import µVision1			输入一个µVision1 工程文件


Project... Open Project... Close Project... Target Environment Targets , Groups, Files Select Device for Target Remove... Options...			打开一个已有的工程 关闭当前的工程 定义工具系列、包含文件、库文件的路径 维护工程的对象、文件组和文件 从器件数据库选择一个 CPU 从工程中删去一个组或文件 设置对象、组或文件的工具选项 设置当前目标的选项 选择当前目标
File Extensions Build Target Rebuild Target Translate... Stop Build 1~9		Alt+F7 F7 Ctrl+F7	选择文件的扩展名以区别不同的文件类型 转换修改过的文件并编译成应用 重新转换所有的源文件并编译成应用 转换当前的文件 停止当前的编译进程 打开最近使用的工程文件

(6) 调试菜单和调试命令(Debug)

Debug 菜单	工具栏	快捷键	描述
Start/Stop Debugging		Ctrl+F5	启动或停止µVision2 调试模式
Go		F5	运行(执行), 直到下一个有效的断点
Step		F11	跟踪运行程序
Step Over		F10	单步运行程序
Step out of current function		Ctrl+F11	执行到当前函数的程序
Stop Running		ESC	停止程序运行
Breakpoints...			打开断点对话框
Insert/Remove Breakpoint			在当前行设置/清除断点
Enable/Disable Breakpoint			使能/禁止当前行的断点
Disable All Breakpoints			禁止程序中所有断点
Kill All Breakpoints			清除程序中所有断点
Show Next Statement			显示下一条执行的语句/指令
Enable/Disable Trace Recording			使能跟踪记录, 可以显示程序运行轨迹
View Trace Records			显示以前执行的指令

Memory Map...			打开存储器空间配置对话框
Performance Analyzer...			打开性能分析器的设置对话框
Inline Assembly...			对某一行重新汇编，可以修改汇编代码
Function Editor			编辑调试函数和调试配置文件

(7) 外围器件菜单(Peripherals)

Peripheral 菜单	工具栏	快捷键	描述
Reset CPU			复位 CPU
Interrupt, I/O—Ports, Serial, Timer, A/D Converter, D/A Converter, I ² C Controller, CAN Controller, Watchdog			打开在片外围器件的对话框。对话框的列表和内容由您在器件数据库中选择 CPU 决定,不同的 CPU 会有所不同。

(8) 工具菜单(Tools)

通过工具菜单，可以配置和运行 Gimpel PC—Lint、Siemens Easy—Case 和用户程序。执行 Customize Tools Menu...，可以将用户程序添加到菜单中。

Tools 菜单	工具栏	快捷键	描述
Setup PC—Lint...			配置 Gimpel Software 公司的 PC—Lint
Lint			在当前的编辑文件中运行 PC—Lint
Lint all C Source Files			在工程的 C 源代码文件中运行 PC—Lint
Setup Easy—Case...			配置 Siemens Easy—Case
Start/Stop Easy—Case			启动或停止 Siemens Easy—Case
Show File(Line)			在当前编辑的文件中运行 Easy—Case
Customize Tools Menu...			将用户程序加入工具菜单

(9) 软件版本控制系统菜单(SVCS)

这个菜单可以配置和添加软件版本控制系统(Software Version Control System)命令。

SVCS 菜单	工具栏	快捷键	描述
Configure Version Control...			配置您的软件版本控制系统命令

(10) 视窗菜单(Window)

Window 菜单	工具栏	快捷键	描述
Cascade			层叠所有窗口
Tile			横向排列窗口(不层叠)
Horizontally			
Tile Vertically			纵向排列窗口(不层叠)
Arrange			在窗口的下方排列图标
Icons			
Split			将激活的窗口拆分成几个窗格
1—9			激活选中的窗口对象

(11) 帮助菜单(Help)

Help 菜单	工具栏	快捷键	描述
Help topics			打开在线帮助
About			显示 μ Vision 的版本号和许可信息
μ Vision			

2.4 Keil C51 的使用

2.4.1 创建第一个 Keil C51 应用程序

在 Keil C51 集成开发环境下使用工程的方法来管理文件的，而不是单一文件的模式。所有的文件包括源程序(包括 C 程序，汇编程序)、头文件、甚至说明性的技术文档都可以放在工程项目文件里统一管理。在使用 Keil C51 前，您应该习惯这种工程的管理方式，对于刚刚使用 Keil C51 的用户来讲，一般可以按照下面的步骤来创建一个自己的 Keil C51 应用程序。

- 新建一个工程项目文件；
- 为工程选择目标器件(例如选择 PHILIPS 的 P87C52X2)；
- 为工程项目设置软硬件调试环境；
- 创建源程序文件并输入程序代码；
- 保存创建的源程序项目文件；
- 把源程序文件添加到项目中。

下面以创建一个新的工程文件 Led_Light. μ V2 为例，详细介绍如何建立一个 Keil C51 的应用程序。

(1) 双击桌面的 Keil C51 快捷图标，进入如图 2.8 所示的 Keil C51 集成开发环境。看到了吗?或许与您打开 Keil C51 界面有所不同，但您不用着急，这是因为启动 μ Vision2 后， μ Vision2 总是打开用户前一次正确处理的工程，您可以点击工具栏的 Project 选项中的 **Close Project** 命令关闭该工程。

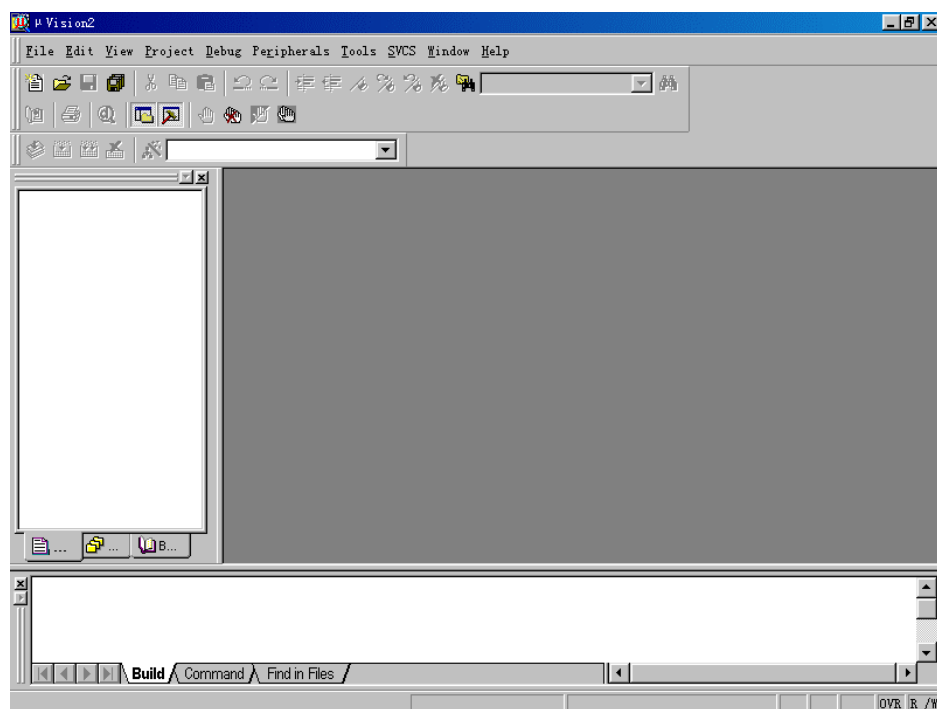


图 2.8 Keil C51 集成开发界面

(2) 点击工具栏的 Project 选项，在弹出如图 2.9 所示的下拉菜单中选择 **New Project** 命令，建立一个新的 μ Vision2 工程，这时可以看到如图 2.10 所示的项目文件保存对话框。

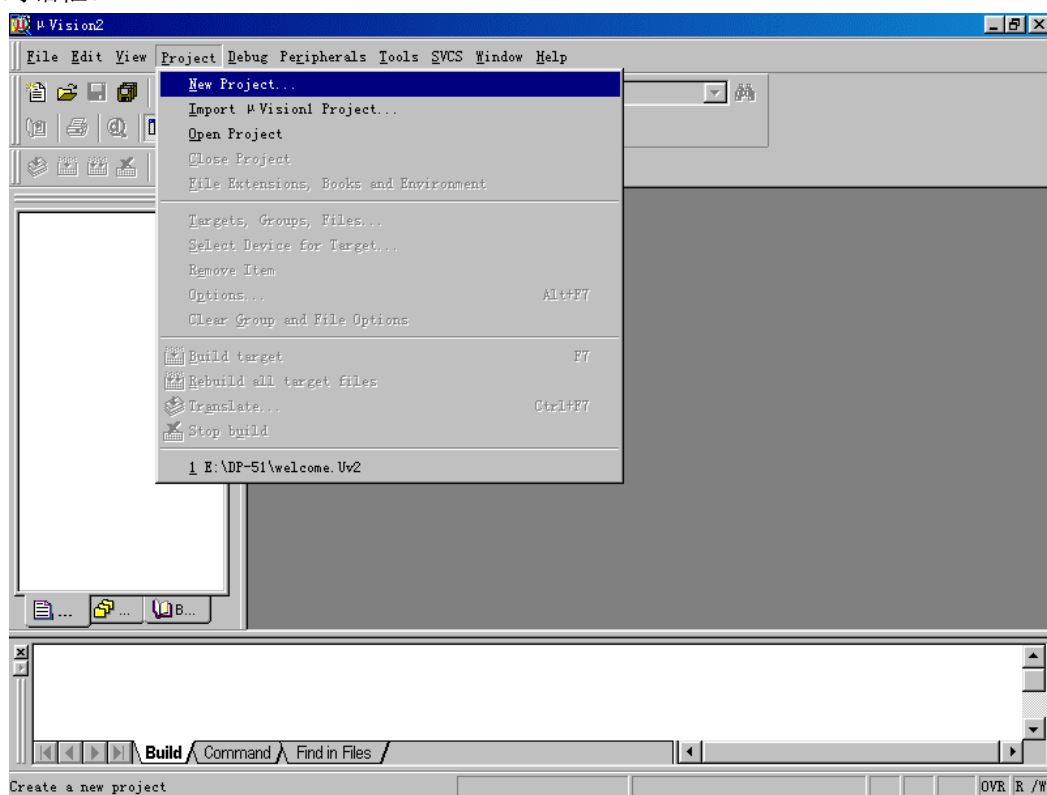


图 2.9 新建工程项目下拉菜单

在这里需要完成下列事情：

- 为您的工程取一个名称，工程名应便于记忆且文件名不宜太长；
- 选择工程存放的路径，建议为每个工程单独建立一个目录，并且工程中需要的所有文件都放在这个目录下；
- 选择工程目录 D:\示范程序\Led_Light 和输入项目名 Led_Light 后，点击**保存** 返回。

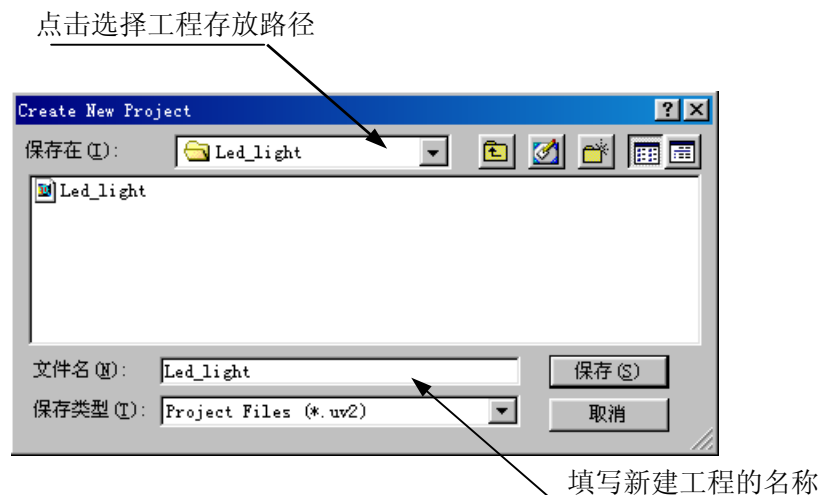


图 2.10 新建工程项目对话框

(3) 在工程建立完毕以后， μ Vision2 会立即弹出如图 2.11 所示的器件选择窗口。器件选择的目的是告诉 μ Vision2 最终使用的 80C51 芯片的型号是哪一个公司的哪一个型号，因为不同型号的 51 芯片内部的资源是不同的， μ Vision2 可以根据选择进行 SFR 的预定义，在软硬件仿真中提供易于操作的外设浮动窗口等。

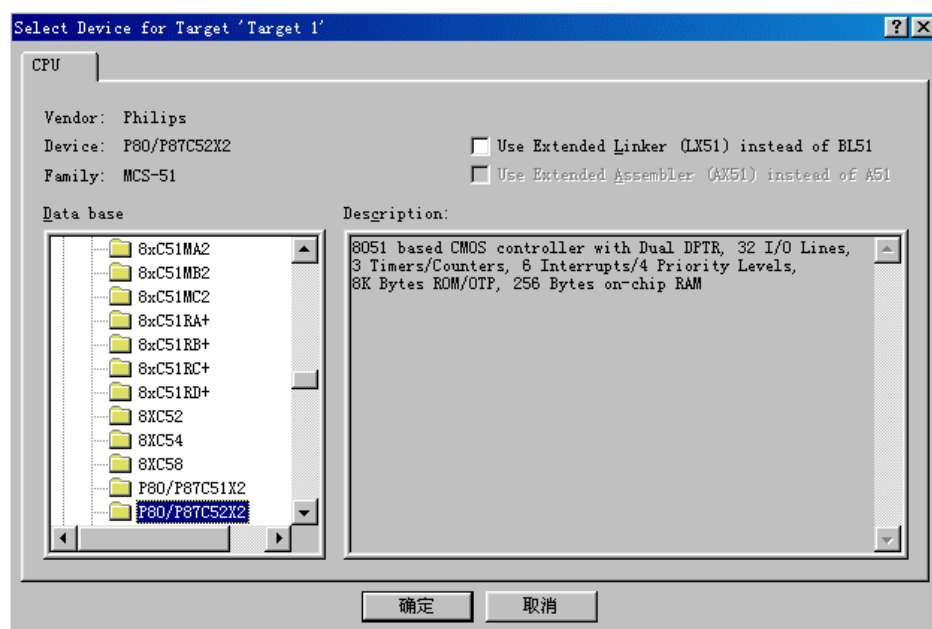


图 2.11 器件选择窗口

由图 2.11 可以看出， μ Vision2 支持的所有 CPU 器件的型号根据生产厂家形成器件组，用户可以根据需要选择相应的器件组并选择相应的器件型号，如 Philips 器件组内的 P80/P87C52X2 CPU。另外，如果用户在选择完目标器件后想重新改变目标器件，可点击工具栏 **Project** 选项，在弹出的如图 2.12 所示的下拉菜单中选择 **Select Device for**

Target ‘Target 1’ 命令，也将出现如图 2.11 所示的对话框后重新加以选择。由于不同厂家的许多型号性能相同或相近，因此如果用户的目标器件型号在 μ Vision2 中找不到，用户可以选择其它公司的相近型号。

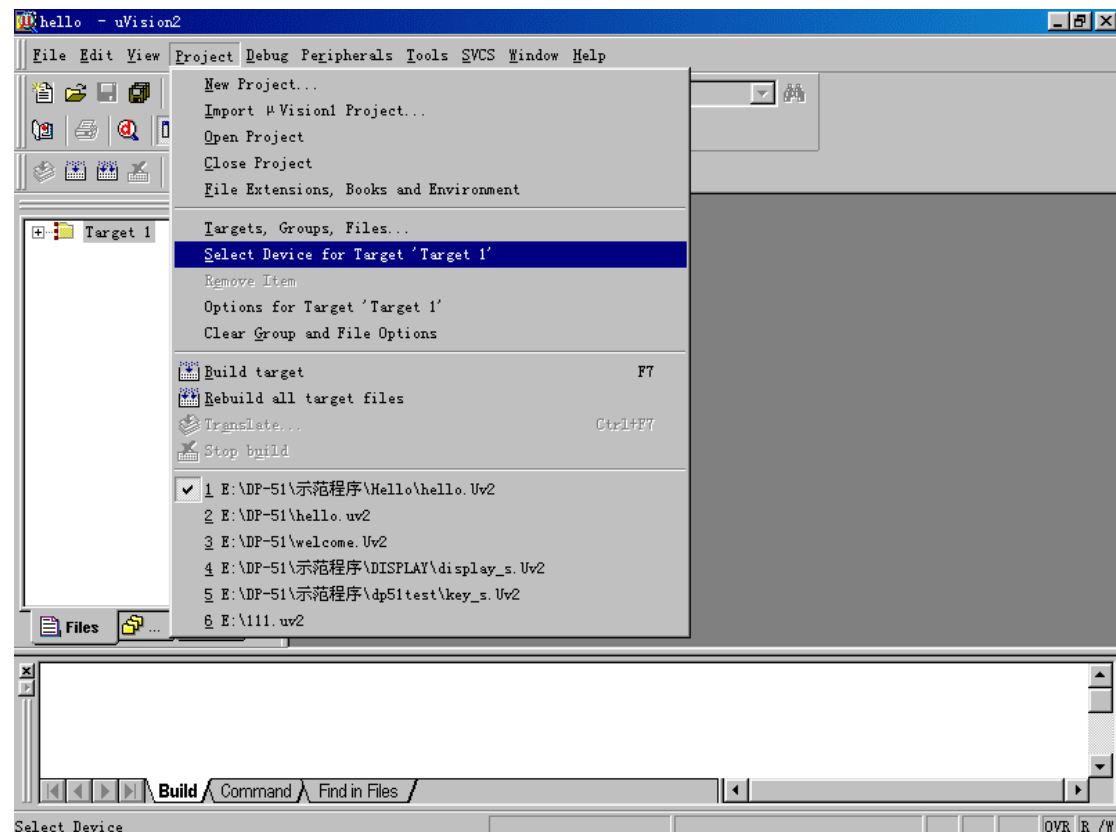


图 2.12 器件选择命令下拉菜单

(4) 到现在用户已经建立了一个空白的工程项目文件，并为工程选择好了目标器件，但是这个工程里没有任何程序文件。程序文件的添加必须人工进行，但如果程序文件在添加前还没有建立，用户还必须建立它。点击工具栏的 **File** 选项，在弹出的如图 2.13 所示的下拉菜单中选择 **New** 命令，这时在文件窗口会出现如图 2.14 所示的新文件窗口 Text1，如果多次执行 **New** 命令则会出现 Text2, Text3... 等多个新文件窗口。

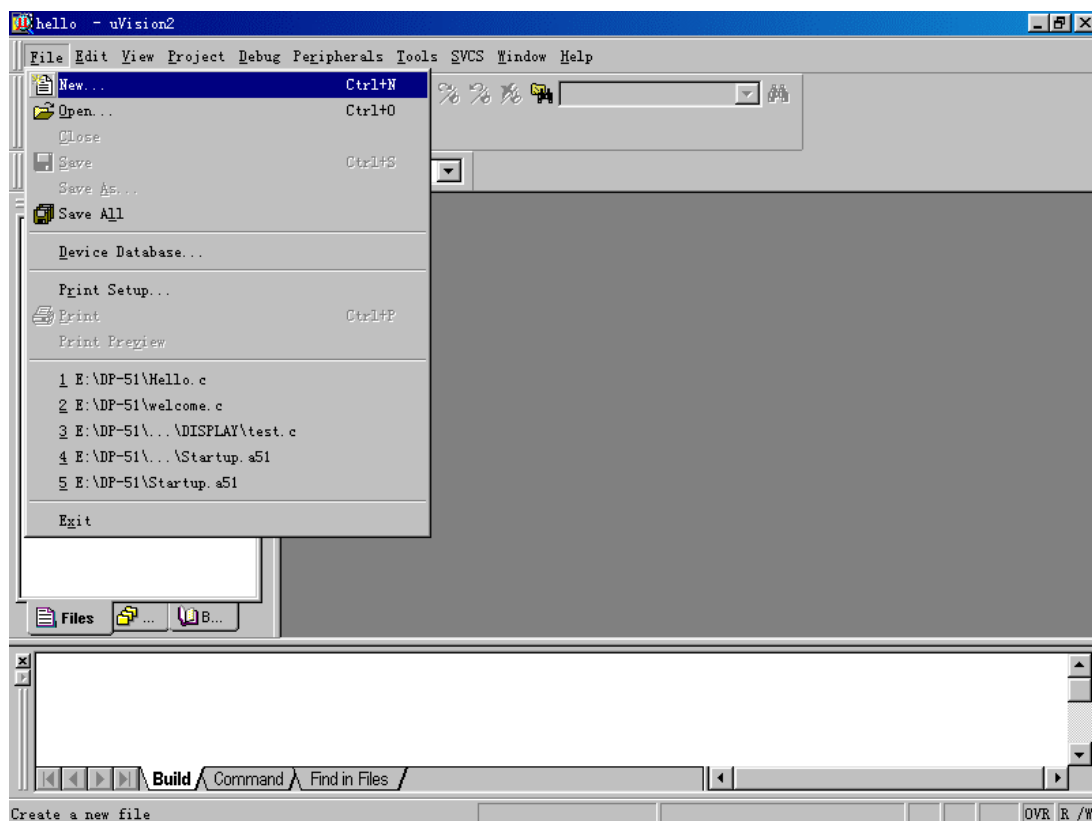


图 2.13 新建源程序下拉菜单

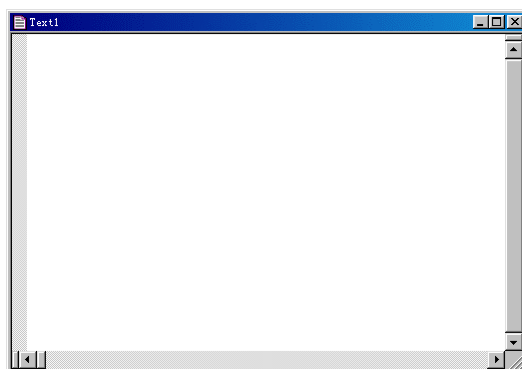


图 2.14 源程序编辑窗口

(5) 现在 Led_Light.µV2 项目中有了一个名为 Text1 新文件框架, 在这个源程序编辑框内输入自己的源程序 Led_Light.asm。在µVision2 中, 文件的编辑方法同其它文本编辑器是一样的, 用户可以执行输入、删除、选择、拷贝、粘贴等基本文字处理命令。当然你也可以使用其他编辑工具在编写源程序, 源程序编辑完毕后保存到磁盘中, µVision2 中有文件变化感知功能, 提示您外部编辑器改变了该文件, 是否需要把µVision2 中的该文件刷新。选择是命令按钮, 然后您就可以看到µVision2 中的源程序文件会自动刷新。下面是完整的 Led_Light.asm 源程序代码, 用户可以输入, 也可以在 E:\Example\Led_light 目录下找到它并把它拷贝到自己的程序中, 您可以采用这种建立/编辑程序文件的方法来创建其它的程序文件。

```

*****Copyright (c)*****
**
**      广州周立功单片机发展有限公司研究所
**      http://www.zlgmcu.com

```



```

;*****文件信息*****
;文件名: Led_Light.asm
;功能: 演示使用 DP-51PROC 单片机综合仿真实验仪仿真调试程序的方法
;说明:
;最后修改时间:2003 年 3 月 21 日
;*****
        ORG        0000H        ;伪指令, 指定程序从 0000H 开始存放
        AJMP       MAIN        ;跳转指令, 程序跳转到 MAIN 处

        ORG        0100H        ;伪指令, 指定以下程序从 0100 开始存放
MAIN:
        MOV        SP,#60        ;给堆栈指针赋初值
LIGHT:
        CPL        P1.0        ;取反 P1.0
        CPL        P1.1        ;取反 P1.1
        CPL        P1.2        ;取反 P1.2
        ACALL      DELAY        ;调延时子程序
        AJMP       LIGHT        ;跳转, 程序继续

;*****
;延时子程序
;*****
DELAY:
        MOV        R7,#10H
DELAY0:
        MOV        R6,#7FH
DELAY1:
        MOV        R5,#7FH
DELAY2:
        DJNZ       R5,DELAY2
        DJNZ       R6,DELAY1
        DJNZ       R7,DELAY0
        RET
;
END

```

(6) 输入完毕后点击工具栏的 **File** 选项, 在弹出的下拉菜单中选择 **保存** 命令存盘源程序文件, 这时会弹出如图 2.15 所示的存盘源程序画面, 在文件名栏内输入源程序的文件名, 在此示范中把 Text1 保存成 Led_Light.asm。注意由于 Keil C51 支持汇编和 C 语言, 且 μ Vision2 要根据后缀判断文件的类型, 从而自动进行处理, 因此存盘时应注意输入的文件名应带扩展名 .ASM 或 .C。源程序文件 Led_Light.asm 是一个汇编语言 A51 源代码程序, 如果用户建立的是一个 C 语言源程序, 则输入文件名称 Led_Light.c。保存完毕后请注意观察, 保存前后源程序有哪些不同, 关键字变成蓝颜色了吗?这也是用户检查程序命令行的好方法, 您体会到了吗?

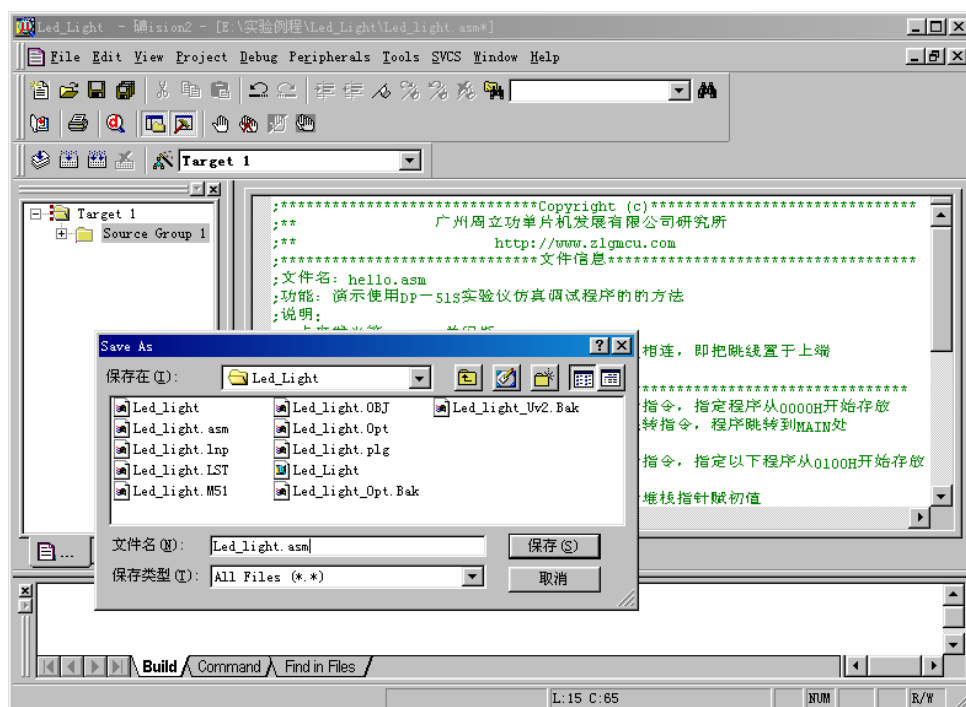


图 2.15 源程序文件保存对话框

(7) 需要特别提出的是，这个程序文件仅仅是建立了而已，Led_Light.asm 文件到现在为止跟 Led_Light.µV2 工程还没有建立起任何关系。此时用户应该把 Led_Light.asm 源程序添加到 Led_Light.µV2 工程中，构成一个完整的工程项目。在 **Project Windows** 窗口内，选中 **Source Group1** 后点击鼠标右键，在弹出如图 2.16 所示的快捷菜单中选择 **Add files to Group "Source Group1"** (向工程中添加源程序文件) 命令，此时会出现如图 2.17 所示的添加源程序文件窗口，选择刚才创建编辑的源程序文件 Led_Light.asm，单击 **Add** 命令即可把源程序文件添加到项目中。由于添加源程序文件窗口中的默认文件类型是 C Source File (*.c)，这样在搜索显示区中则不会显示刚才创建的源程序文件 (由于它的文件类型是 *.asm)。改变搜索文件类型为 Asm Source File (*.a*:*.*.src)，并最终选择 Led_Light.asm 源程序文件即可。

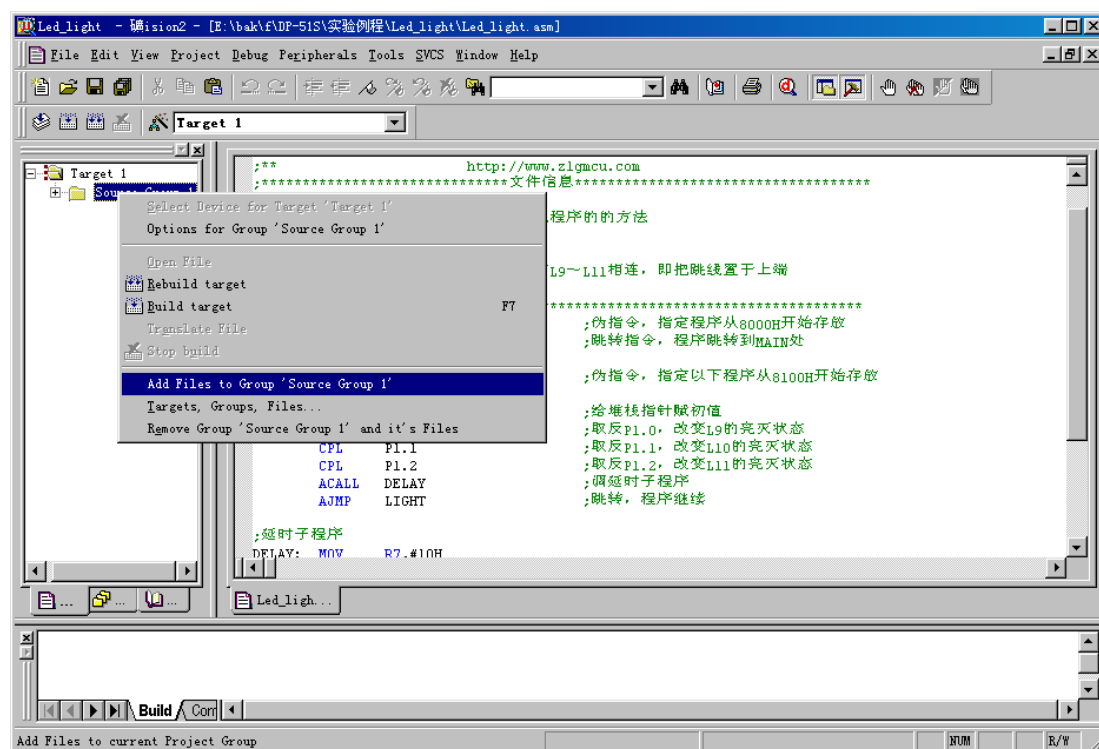


图 2.16 添加源程序快捷菜单

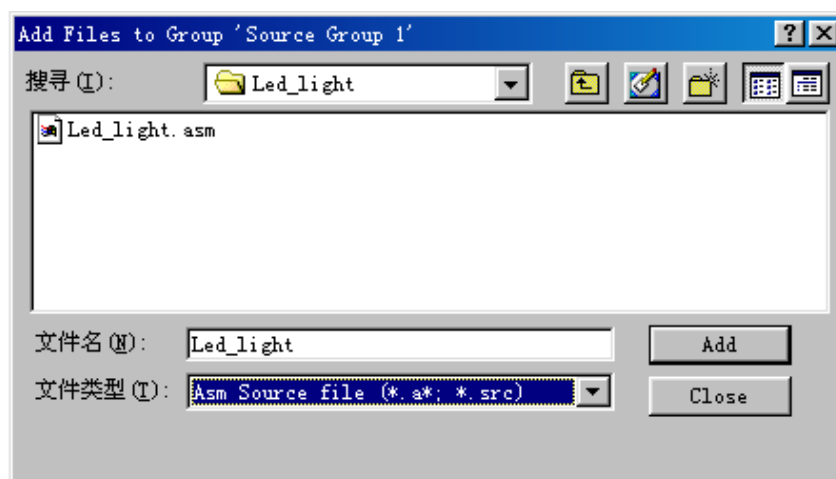


图 2.17 添加源程序文件窗口

2.4.2 程序文件的编译、链接

(1) 编译连接环境设置

μVision2 调试器可以调试用 C51 编译器和 A51 宏汇编器开发的应用程序，μVision2 调试器有两种工作模式，用户可以通过点击工具栏 **Project** 选项，在弹出如图 2.18 所示的下拉菜单中选择 **Option For Target 'Target 1'** 命令为目标设置工具选项，这时会出现如图 2.19 所示的调试环境设置窗口，点击 **Output** 选项卡在出现的窗口中选 **Create Hex File** 选项，在编译时系统将自动生成目标代码文件 *.HEX。选择 **Debug** 选项会出现如图 2.20 所示的工作模式选择窗口，在此窗口中我们可以设置不同的仿真模式。

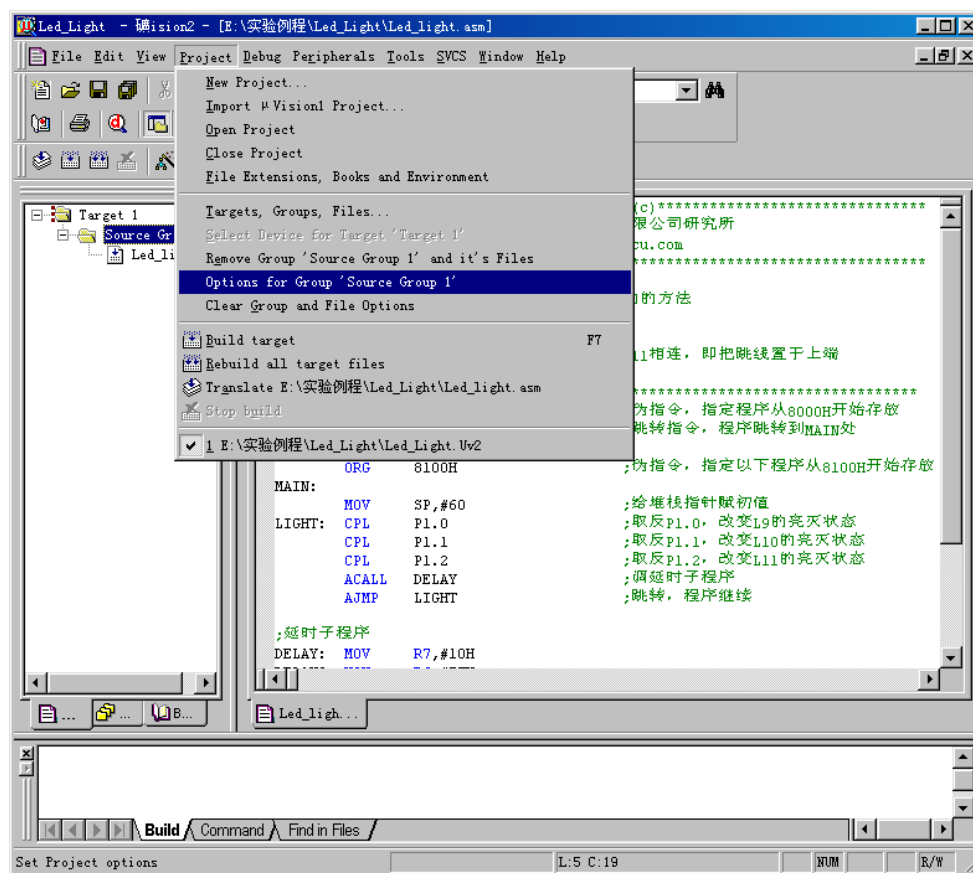


图 2.18 调试环境设置命令下拉菜单

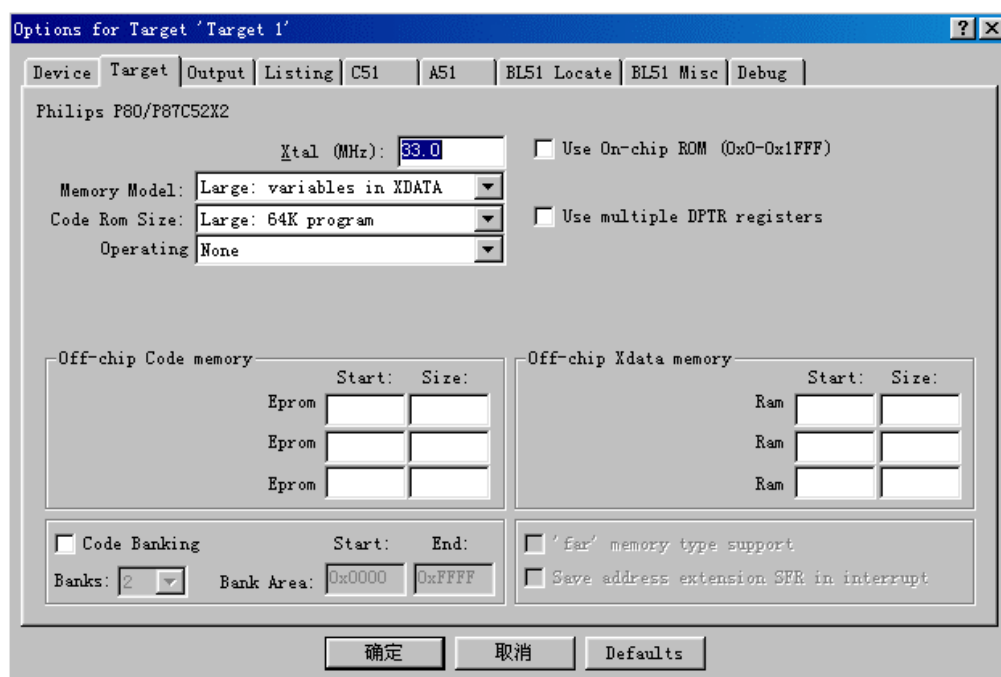


图 2.19 Keil C51 调试环境设置窗口

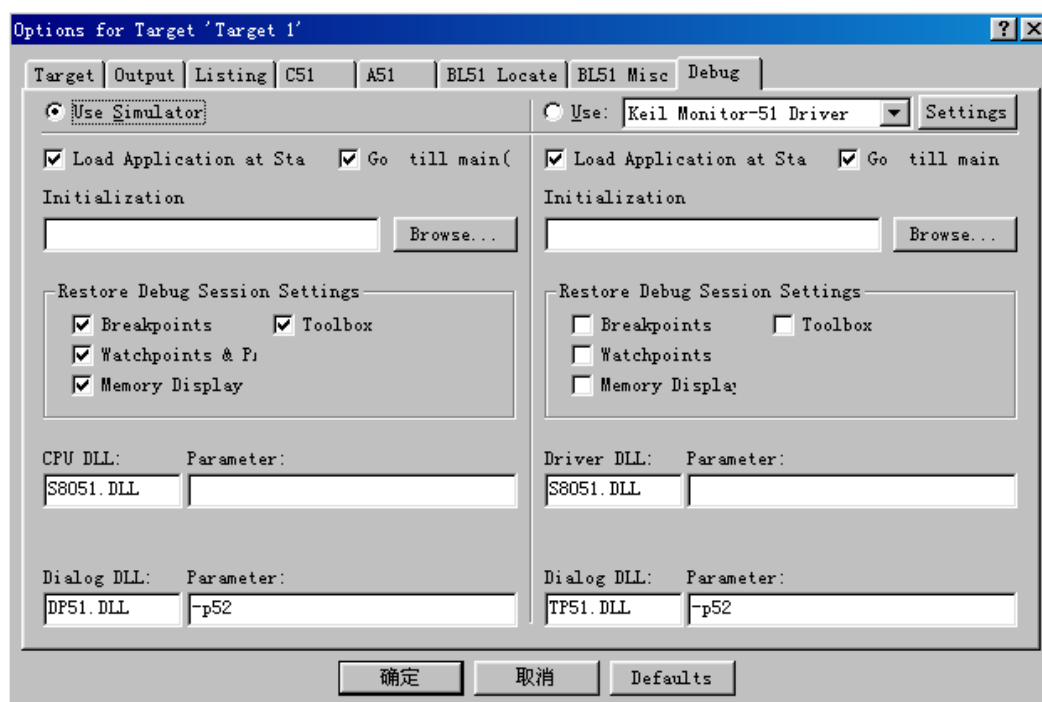


图 2.20 Debug 设置窗口

从图 2.20 可以看出， μ Vision2 的 2 种工作模式分别是：Use Simulator（软件模拟）和 Use（硬件仿真）。其中 Use Simulator 选项是将 μ Vision2 调试器设置成软件模拟仿真模式，在此模式下不需要实际的目标硬件就可以模拟 80C51 微控制器的很多功能，在准备硬件之前就可以测试您的应用程序，这是很有用的。

Use 选项有高级 GDI 驱动（TKS 仿真器）和 Keil Monitor-51 驱动（适用于像 DP-51PROC 单片机综合仿真实验仪的用户目标系统），运用此功能用户可以把 Keil C51 嵌入到自己的系统中，从而实现在目标硬件上调试程序。若要使用硬件仿真，则应选择 **Use** 选项，并在该栏后的驱动方式选择框内选这时的驱动程序库。在此由于只需要调试程序，因此用户可以选择软件模拟仿真，在图 2.20 中 Debug 栏内选中 **Use Simulator** 选项，点击 **确定** 命令按钮加以确认，此时 μ Vision2 调试器即配置为软件模拟仿真。

(2) 程序的编译、连接

完成以上的工作就可以编译程序了。点击工具栏 **Project** 选项，在弹出如图 2.21 所示的下拉菜单中选择 **Build Target** 命令对源程序文件进行编译，当然也可以选择 **Rebuild All Target Files** 命令对所有的工程文件进行重新编译，此时会在“Output Windows”信息输出窗口输出一些相关信息，如图 2.22 所示。

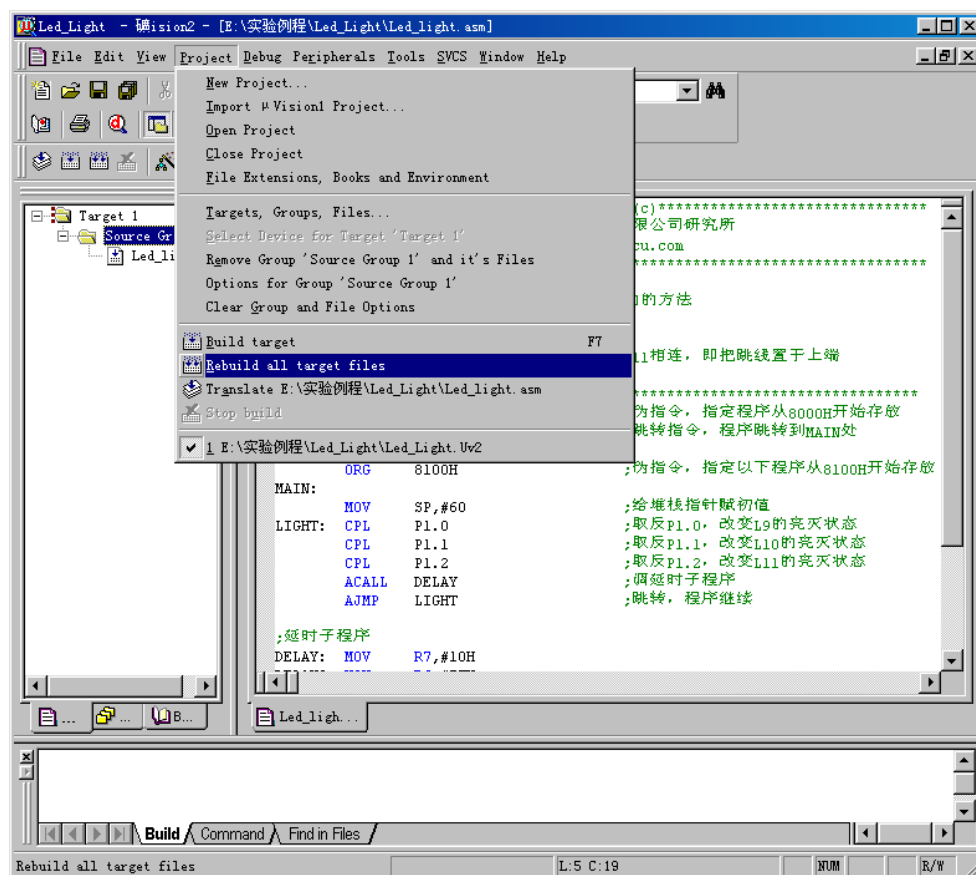


图 2.21 编译命令菜单

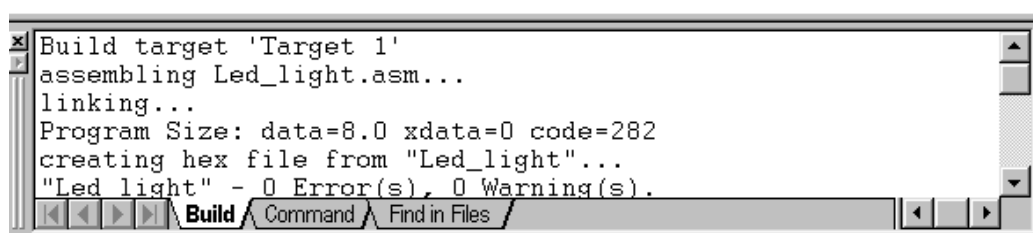


图 2.22 输出提示信息

其中第二行 assembling Led_Light.asm 表示此时正在编译 Led_Light.asm 源程序，第三行 linking... 表示此时正在连接工程项目文件，第五行 Creating hex file from 'Led_Light' 说明已生成目标文件 Led_Light.hex，最后一行说明 Led_Light.µV2 项目在编译过程中不存在错误和警告，编译链接成功。若在编译过程中出现错误，系统会给出错误所在的行和该错误提示信息，用户应根据这些提示信息，更正程序中出现的错误，重新编译直至完全正确为止。

至此一个完整的工程项目 Led_Light.µV2 已经完成，但别高兴得太早，一个符合要求的、好的工程项目(系统、文件或程序)是要经得起考验的。它往往还需要经软件模拟、硬件仿真、现场系统调试等反复修改、更新的过程。有了 DP-51PROC 单片机综合仿真实验仪这些繁复问题将会迎刃而解，无论是软件仿真还是软硬件的综合调试，这些繁复的工作都可以轻松搞定，而且它所提供的功能强大的 Keil C51 调试分析软件和功能丰富的硬件资源，将使您的开发倍感轻松，在使用过程中您也将发觉它是您学习、实验、调试必不可少的好工具、好帮手。

2.5 调试仿真功能的使用

调试仿真功能是指 DP-51PROC 单片机综合仿真实验仪运行 TKSMonitor51 仿真器中单片机 P87C52X2 内部的 MON51 监控程序，把用户的应用程序装载到外部 SRAM 中，从而实现运用 Keil C51 集成开发环境所提供的所有调试命令来调试用户的应用程序或仿真用户的应用系统。

2.5.1 如何进入调试状态

首先，把 TKSMonitor51 仿真器上的开关拨到 LOAD 模式，即在下载状态下，用户应将 TKSMonitor51 仿真器的仿真头插入到 DP-51PROC 单片机综合仿真实验仪的 U13 锁紧座上；然后 ISP 跳线 JP14 跳开（即不短接），按下复位按键“RESET”。此时，DP-51PROC 单片机综合仿真实验仪即进入下载状态。

然后将 TKSMonitor51 仿真器接上串口线，在 PC 机上双击 DPFlash 下载软件的快捷图标（DPFLASH 安装版在光盘的 SOFTWARE 目录内，运行安装即可），运行 DPFlash 下载软件。这时将出现如图 2.23 所示的 DPFlash 下载软件的操作界面。在 DPFLASH 上 **型号** 的下拉菜单选择 **DP-51PROC**，然后选择适当的通信口即可。另外 DPFlash 下载软件还内嵌一个串口调试器，如图 2.23 所示的菜单栏的 **串口调试器**，用户可以使用它进行串口调试。

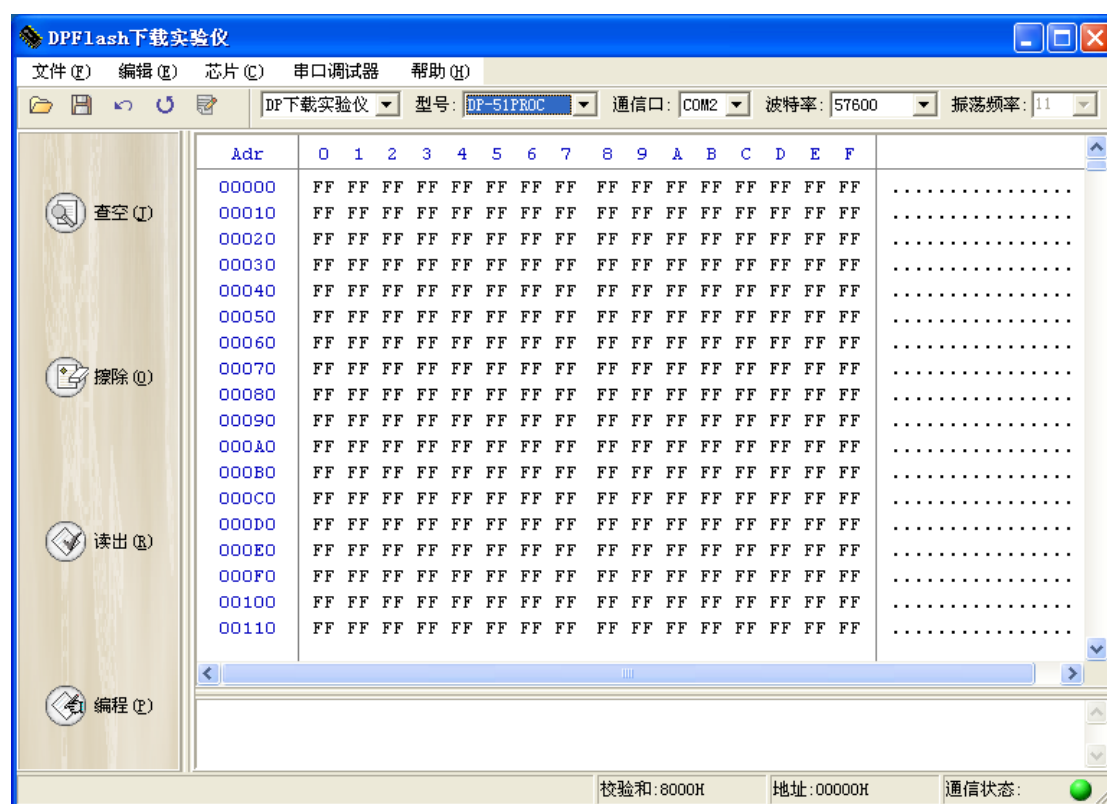


图 2.23 DPFlash 编程界面

然后在主界面中点击 **编程** 命令按钮，在出现如图 2.24 所示的编程窗口中选择 **其它编程选择** 栏的 **编程 MON51** 选项，单击 **编程** 命令按钮即可自动把 MON51.HEX 监控程序下

载到 TKSMonitor51 仿真器的 Flash 中。若无异常，则提示编程正常结束，这时关闭该窗口退出 DPFlash 软件。把 TKSMonitor51 仿真器上的工作模式选择开关切换到 RUN 处，然后按一下复位键（RESET），MON51 程序就开始运行了。此时，DP-51PROC 单片机综合仿真实验仪即进入调试状态。



图 2.24 MON51 编程界面

调试仿真功能是指 DP-51PROC 运行单片机 TKSMonitor51 仿真器中 P87C52X2 内部的 MON51 监控程序，把用户的应用程序装载到外部 SRAM 中，从而实现运用 Keil C51 集成开发环境所提供的所有调试命令来调试用户的应用程序或仿真用户的应用系统。

2.5.2 调试状态的存储器模型

当 TKSMonitor51 仿真器处于调试状态时将执行 MON51 监控程序，这样可在 Keil μ Vision2 集成开发环境下调试程序，即作为 MON51 调试器。在调试状态下 TKSMonitor51 仿真器的存储器模型，如图 2.25 所示。

FLASH（外部程序空间）	SRAM（外部数据空间）
	用户数据区 0FFFFH 0C000H
	用户程序区 0BFFFFH 8000H
内部 MON51 监控程序 7FFFH 0000H	用户扩展 I/O 映射区（由用户自由使用） 7FFFH 0000H

图 2.25 调试状态下存储空间分配图

系统复位后，TKSMonitor51 仿真器执行“MON51 监控程序”。在调试状态下，用户的应用程序必须从 SRAM 的 0x8000 地址开始存放，中断矢量也应从相应的地址单元转移到从 0x8000 开始的相应单元。

注意：调试状态下，定时器 T2、串行口 UART 已被“MON51 监控程序”所占用，用户不能再使用这些资源。

2.5.3 调试前的准备工作

1. 硬件环境

- (1) 拿出随机提供的串口通信电缆，一端连接 TKSMonitor51 仿真器的 RS-232 串行通信口，而另一端则连接到 PC 机的串口上(COM1 或 COM2)。
- (2) 打开实验仪的工作电源，此时即为仿真调试准备好了硬件环境。
- (3) 如 2.5.1 节所说的设置好跳线，并下载 MON51 监控程序到实验仪。

2. 软件调试环境的设置

- (1) 双击 Keil C51 快捷图标，进入 Keil C51 集成开发环境，这时 Keil C51 集成环境自动打开上次正确退出时所编辑的工程项目文件，如图 2.26 所示。

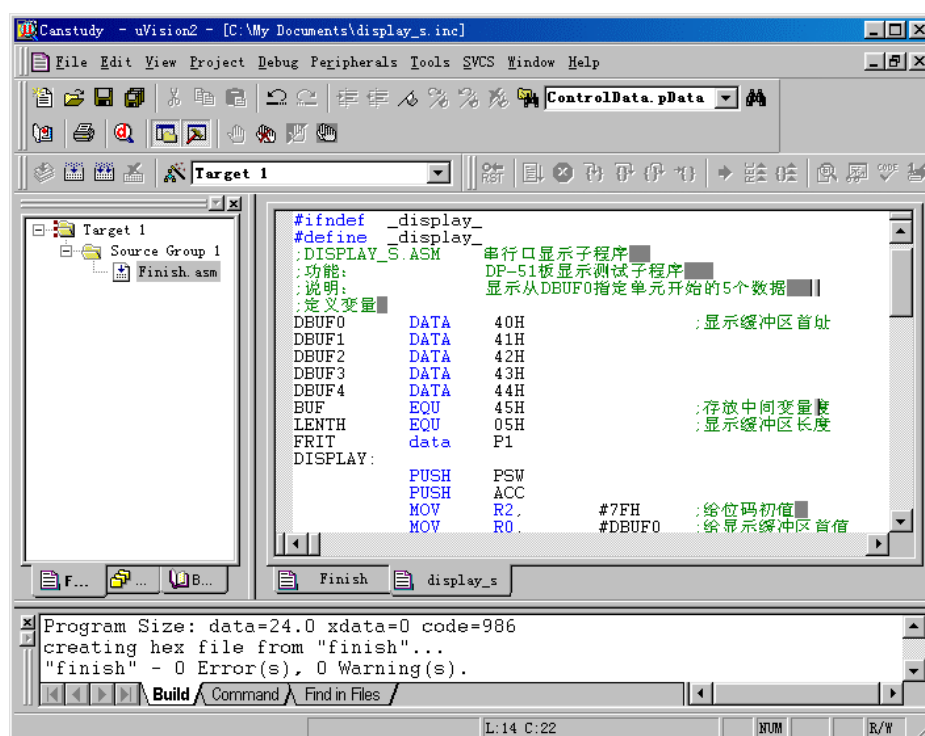


图 2.26 Keil C51 集成开发环境界面

- (2) 点击菜单栏上的 **Project** 项，会弹出下拉式菜单，这时选择 **Option for target 'target 1'** 将出现如图 2.27 所示的调试环境设置界面。

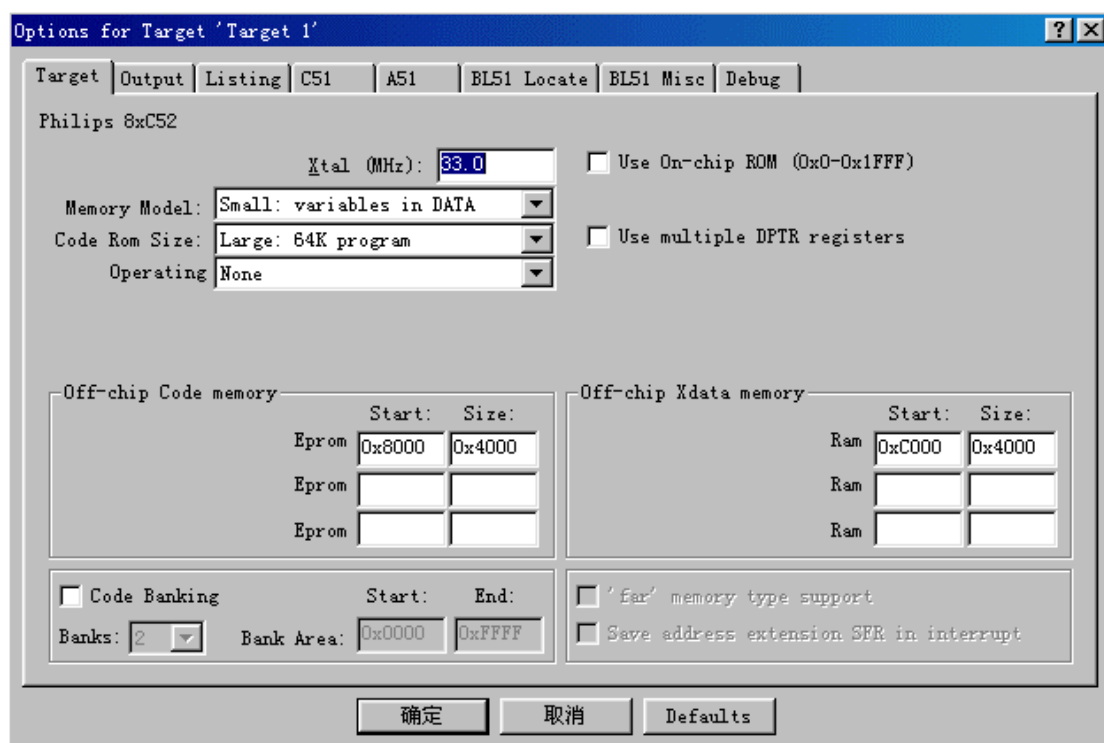


图 2.27 调试环境设置窗口

(3) 第 1 项 Target 属性的设置，对于在 TKSMonitor51 仿真器上进行的仿真、调试，由于 MON51 监控程序已经占用了从 0000H—7FFFH 地址单元的程序存储空间，因此用户的应用程序必须从 8000H 地址单元开始存放，即用户应设置 **Off—Chip Code Memory** 栏内的 Eprom 选项。具体配置请见图 2.27，对于第 4 项“C51”的配置请按图 2.28 进行设置。

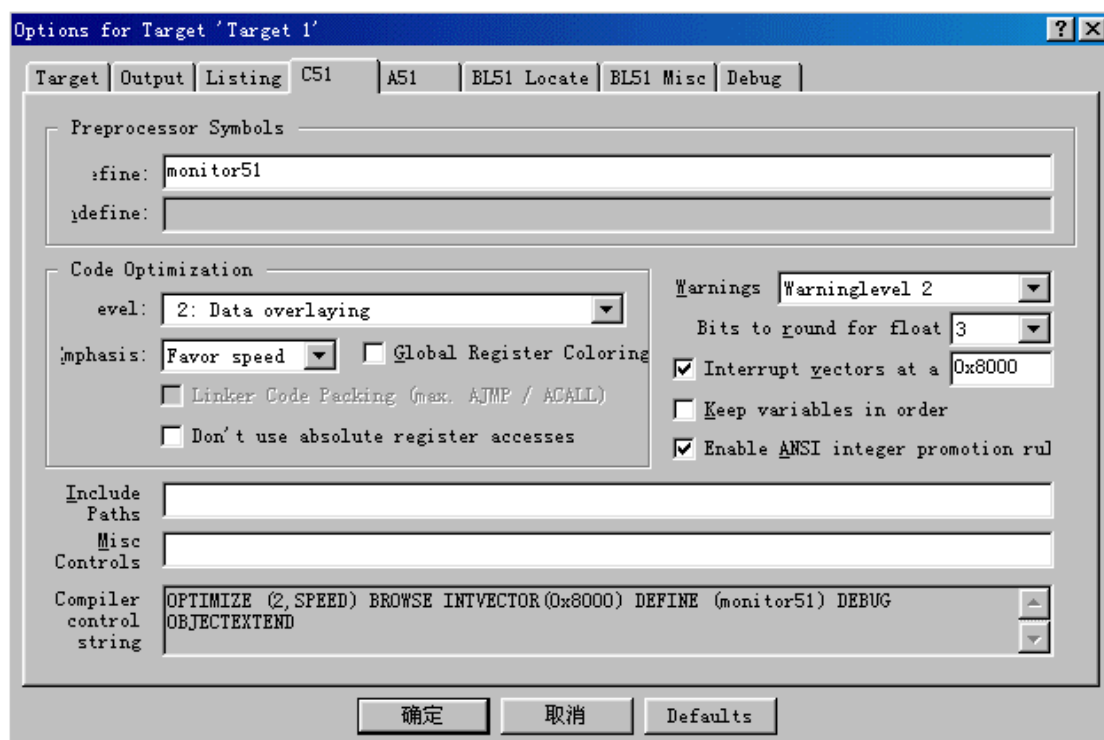


图 2.28 C51 属性栏的设置

(4) Debug 环境的设置：首先选择 **Debug** 项，进入如图 2.29 所示的设置画面，这

时就可以对其中的每项进行具体设置了,当然您完全可以按照图 2.29 进行设置(要点:在该选项中 Use Simulator 是软件模拟仿真,它只能对程序的语法及其结构做一般性的分析,与硬件没有联系;而 Use 选项则是硬件仿真,且根据所选用的驱动而使用不同的硬件仿真方式,对于 TKSMonitor51 仿真器而言应该选择 Keil Monitor-51 Driver 选项)。同时它的 **Settings** 项还为我们提供了一个串口通信设置环境,通过它我们可以灵活设置串行通信的端口和波特率,其设置环境如图 2.30 所示,但请注意:由于在调试模式下,TKSMonitor51 仿真器需要与上位机进行通信,因此它们的通信协议必须一致,波特率也必须相同且为 9600bps。当然要进入系统调试环境设置我们也可以点击工具栏上的快捷图标进入到环境设置窗口,Keil C51 集成开发环境为我们提供了很多这样的快捷功能,好好利用往往能达到事半功倍的效果。

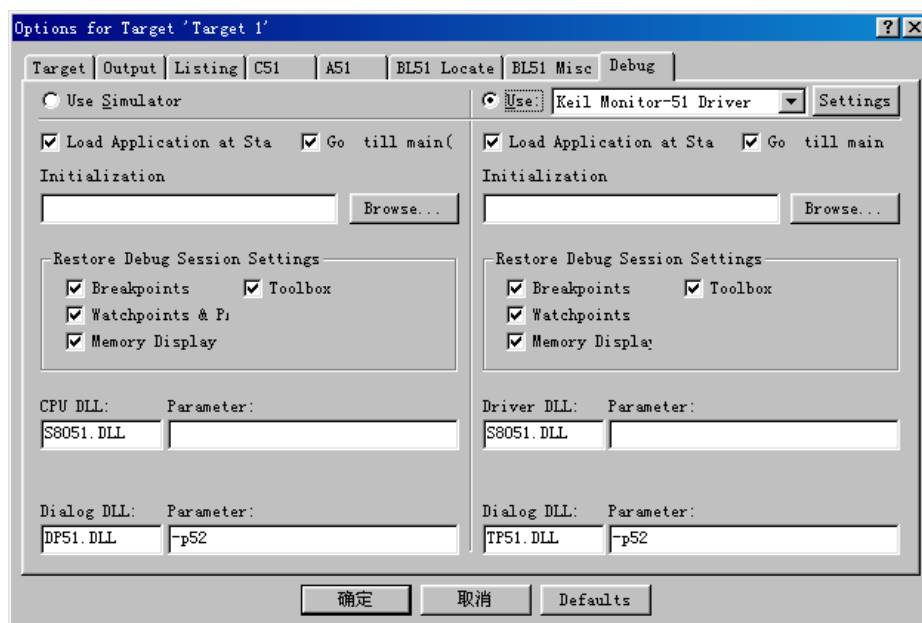


图 2.29 Debug 调试环境设置

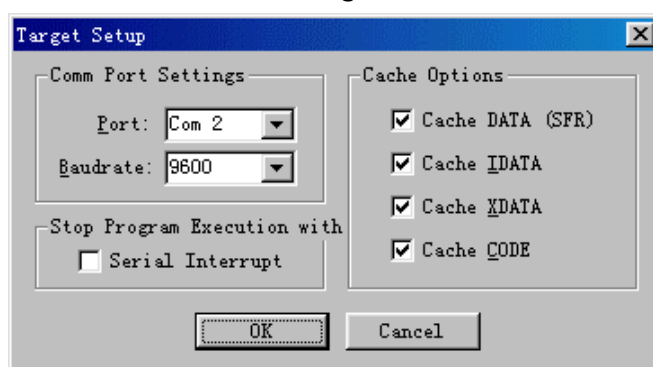


图 2.30 串口属性设置

(5) 至于其它的选项用户可按默认值进行设置或不用设置,最后点击**确定**加以确认。

2.5.4 实战

下面仍以 Led_Light.asm 为例,讲述运用 TKSMonitor51 仿真器在 Keil C51 集成开发环境下调试程序的步骤和方法。

(1) 打开示范程序的项目文件 Led_Light.pv2,这时在 Keil C51 的 Project Window 项目观察窗口中看到如图 2.31 所示的工程项目结构。在 **Source Group 1** 中可以看到文

件 Led_Light.asm，它就是该项目的核心——用户要调试的源程序文件。在此用户应特别注意：在 TKSMonitor51 仿真器下调试时，用户的程序代码是从 8000H 单元开始存放的，因此用户程序中定义程序块存放的伪指令 **ORG 0000H** 应该为 **ORG 8000H**、伪指令 **ORG 0100H** 应该为 **ORG 8100H**。若用户需要调试 C 语言源程序，则还需要添加 Startup.a51 系统配置文件(用户可以在\Keil\C51\Lib 目录下找到它，为了保证该文件不会被破坏，用户可以把它拷贝到该工程目录中)，其主要功能是定义数据段、程序段和堆栈的大小及其起始地址、RAM 的初始化、程序重新定位、初始化启动代码等等，用户必须把 **CSEG AT 0** 代码改为 **CSEG AT 8000H**，即用户的应用程序应从 8000H 开始存放，当然中断矢量地址也应跳转到相应的高端地址单元，如外部中断 0 的入口地址则应该是 8003H。



图 2.31 Led_Light 工程的结构

(2) 在使用 TKSMonitor51 仿真器进行的调试是一种软硬件相结合的综合调试方法，因此用户在使用 TKSMonitor51 仿真器进行仿真调试实验前应正确设置工程项目的软硬件环境，本例子的硬件设置请参考第 3 章实验二的实验步骤。在 KEIL 的环境下点击工具栏 **Project** 的选项，在弹出的如图 2.32 所示的下拉菜单中选择 **Option for Target 'Target 1'** 命令，对 hello.uv2 工程项目进行调试环境设置，具体设置方法请参考 2.5.3 的第 2 部分——软件调试环境设置。

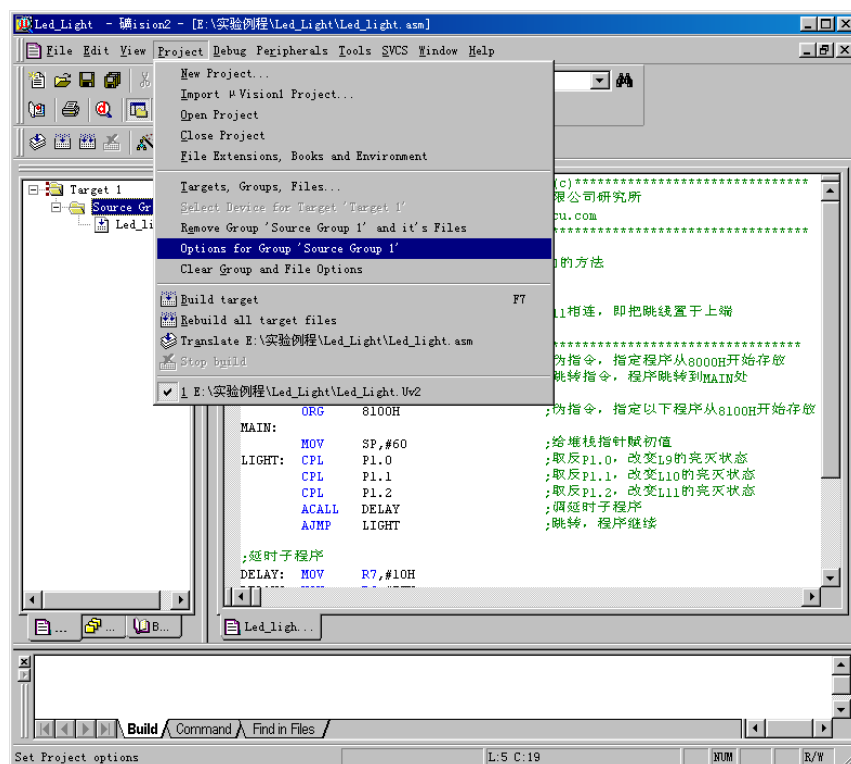


图 2.32 调试环境设置命令菜单

(3) 修改完毕执行 **Project** 菜单中的 **Rebuild all target files** 命令对工程项目文件进行重新编译、连接，此时会出现“编译正确、连接成功”的提示信息。若编译出错，它将提示出错的原因及所在的位置，更正后重新编译直至完全正确为止，接下来点击菜单栏内的“Debug”菜单，在出现的下拉式菜单中选择 **Start/Stop Debug Session** 调试命令，这样即可把用户程序就下载到 TKSMonitor51 仿真器的 SRAM 中。

(4) 此时出现如图 2.33 所示调试画面，看到了吗?或许和您看到的不一样，请不用担心。若在您的调试界面中没有看到变量观察窗口，您可以点击菜单栏中的 **View** 选项，在弹出如图 2.34 所示的下拉菜单中选择 **Watch & Call Stack Window** 即可以打开变量观察窗口，您可以使用同样的方法打开其它相关窗口。

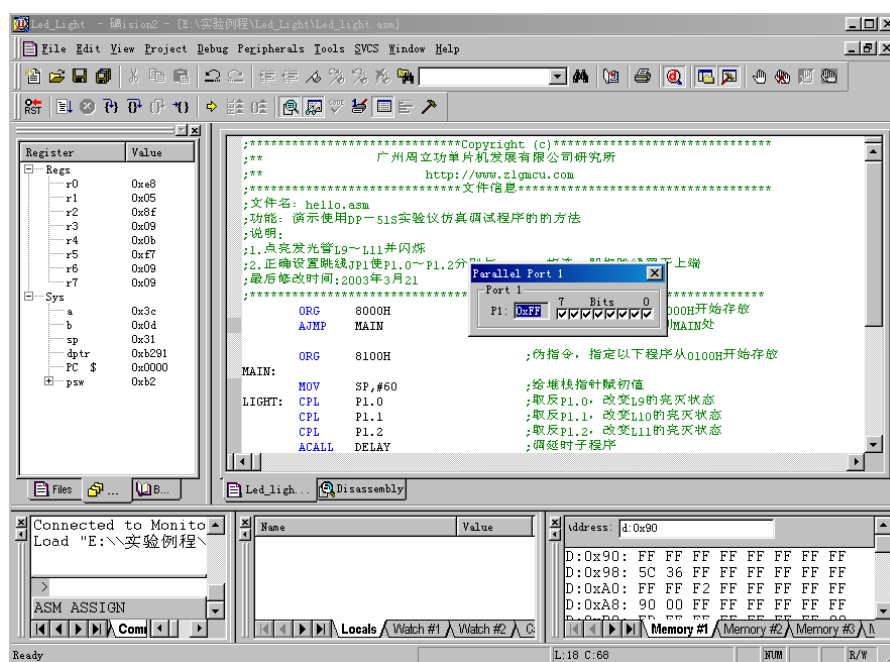
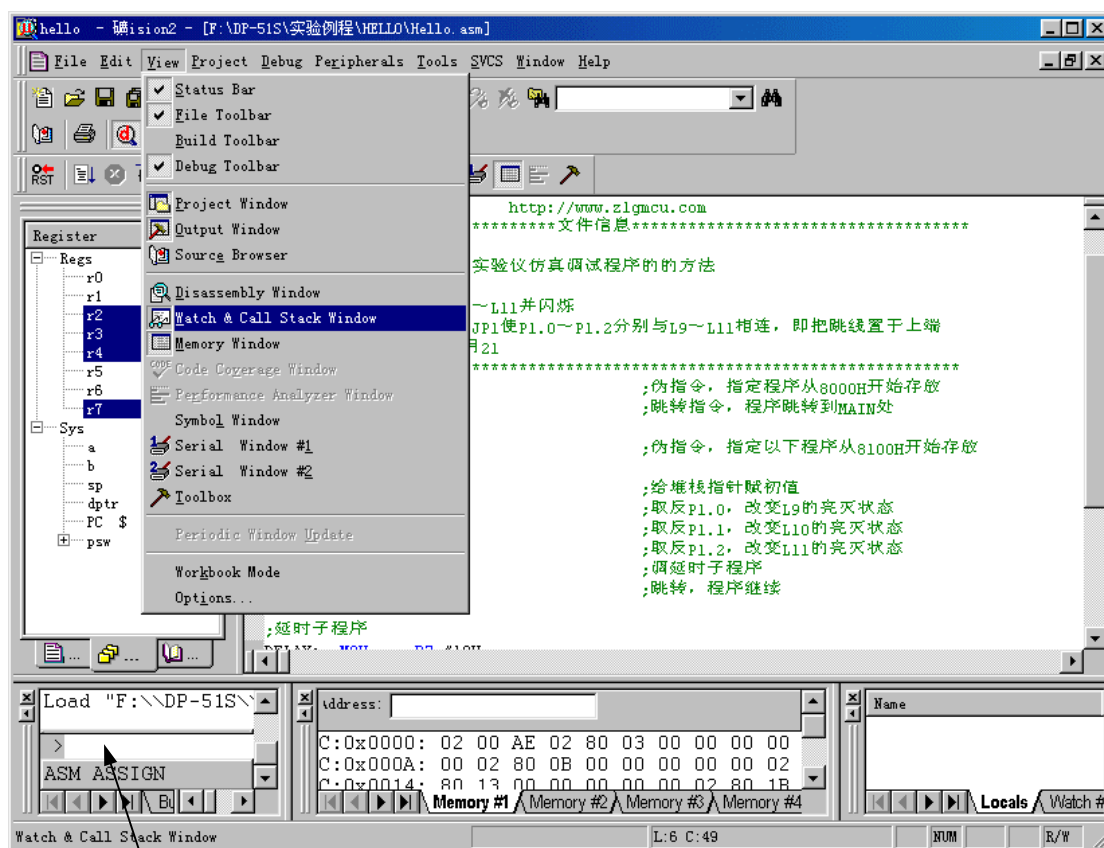


图 2.33 调试界面



Command 输入框

图 2.34 打开变量窗口命令菜单

(5) 此时请注意：当您调试 C 语言程序时，应在 Keil C 环境的 Command 输入框下执行 **g, main** 命令；而当您调试的是汇编语言程序时，在 Keil C 环境的下执行 **g** 命令进入程序调试状态。这时程序指针 PC 已指向第一命令语句 **AJMP MAIN** 处，并等待用户输

入各种调试命令。Keil C51 给出了许多调试快捷图标和调试命令，为了使用户更好的使用这些命令，下面将介绍几种常用的调试命令及方法。



断点

巧妙的设置一些断点，能够更好帮助用户分析程序的运行机制、程序中变量的变化状况，提高工作效率。μVision2 可以用几种不同的方法定义断点，即使在程序代码编译前，您也可以在编辑源程序的时候设置 Execution Breaks，设断点前要从 View 菜单中选中 File Toolbar 选项，让主窗口的顶部出现 File Toolbar 工具栏。断点可以用以下的方法定义和修改：

1. 用 File Toolbar 按钮。在 Editor 或 Disassembly 窗口中的代码行点击断点按钮即可在该处设置断点。
2. 用快捷菜单的断点命令。在 Editor 或 Disassembly 窗口中的代码行点击鼠标右键在打开的快捷菜单中选择 Insert/Remove Breakpoint 命令也同样可以在该行设置断点。
3. 在 Output Window—Command 输入框，可以使用 Breakset、Breakkill、BreakEnable、Breaklist、Breakpoint 命令来设置断点。

当然，设置断点还有一个最简单的方法就是在该行语句前双击即可。如果已经在某行设置了断点，再次在此行设置断点将取消该断点，断点设置成功后，会在该行的行首出现红颜色的断点标志。



复位 CPU

用 Debug 菜单或工具栏的 Reset CPU 命令。在不改变程序的情况下，若想使程序重新开始运行，这时执行此命令即可，执行此命令后程序指针返回到 0000H 地址单元，另外，一些内部特殊功能寄存器在复位期间也将重新赋值。例如 A 将变为 00H，DPTR 为 0000H，SP 为 07H，I/O 口变为 0FFH。



单步跟踪(F11)

用 Debug 工具栏的 Step 或快捷命令 StepInto 命令按钮可以单步跟踪程序，每执行一次此命令，程序将运行一条指令(以指令为基本执行单元)，当前的指令用黄色箭头标出，每执行一步箭头都会移动，已执行过的语句呈现绿色。单步跟踪在 C 语言环境调试下最小的运行单位是一条 C 语句，如果一条 C 语句只对应一条汇编指令，则单步跟踪一次可以运行 C 语句对应一条汇编指令；如果一条 C 语句对应多条汇编指令，则一次单步跟踪要运行完对应的所有汇编指令。在汇编语言调试下，可以跟踪到每一个汇编指令的执行。



单步运行(F10)

用 Debug 工具栏的 Step Over 或快捷命令 Step Over 按钮即可实现单步运行程序，此时单步运行命令将把函数和函数调用当作一个实体来看待，因此单步运行是以语句(这一条语句不管是单一命令行还是函数调用)为基本执行单元。



执行返回(Ctrl+F11)

在用单步跟踪命令跟踪到了子函数或子程序内部时，可以使用 Debug 菜单栏中的 Step Out of Current Function 或快捷命令按钮 Step Out 即可实现程序的 PC 指针返回到调用此子程序或函数的下一条语句。



执行到光标所在命令行(Ctrl+F10)

用工具栏或快捷菜单命令 **Run to Cursor Line** 即可执行此命令，使程序执行到光标所在行，但不包括此行，其实质是把当前光标所在的行当作临时断点。



全速运行(F5)

用 **Debug** 工具栏的 **Go** 快捷命令 **Run** 即可实现全速运行程序，当然若程序中已经设置断点，程序将执行到断点处，并等待调试指令；若程序中没有设置任何断点，当 μ Vision2 处于全速运行期间， μ Vision2 不允许任何资源的查看，也不接受其它的命令。



将鼠标箭头移到一个变量上可以看到它们的值。



按钮可以启动/停止调试(Ctrl+F5)

(6) 由于 **Led_light** 程序使用了系统资源 **P1** 口，为了更好的观察这些资源的变化，用户可以打开它们的观察窗口。点击菜单栏 **Peripherals** 选项，在打开的下拉菜单中选择 **I/O-Ports** 选项中的 **Port 1** 命令，即可打开并行 I/O 口 **P1** 的观察窗口，如图 2.35 所示。

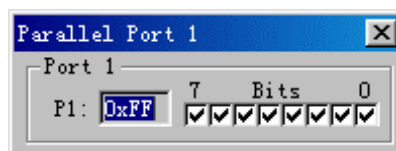


图 2.35 P1 口观察窗口

(7) 点击菜单栏的 **Debug** 选项，在弹出的下拉菜单中执行 **Step** 命令，观察项目窗口的特殊功能寄存器区域，如图 2.34 所示，看看 **PC** 指针和堆栈指针的值有何变化。多次执行单步运行指令 **Step**，注意观察 **P1** 观察窗口和 **DP-51PROC** 单片机综合仿真实验仪上用逻辑笔检测 **P1** 口状态。最后选择 **Go** 命令全速运行程序，此时 **DP-51PROC** 单片机综合仿真实验仪上看的 **P1.0**、**P1.1**、**P1.2** 电平在发生变化。

(8) 程序调试完毕可执行 **Debug** 菜单栏中的 **Start/Stop Debug Session** 命令停止调试(当然您按一下 **TKSMonitor51** 仿真器的复位按钮，即可较快地退出调试状态)。您会了吗?即便还不太熟练，那也不要紧，在以后的章节中将有大量的实战练习。在学习过程中，您千万不能偷懒，要一行一行地老实地将程序敲进去，只有这样您才有可能慢慢地细心地体会到程序设计思想。我们知道，单片机是一门实战性很强的学科。即便您考试考了 100 分，如果没有大量的实战训练，您终究可能还是个“门外汉”，没有办法登堂入室成为一名真正的单片机应用开发工程师，这是成千上万开发人员多年来积累的经验教训，即使万事开头难，但是切记：良好的开端等于成功一半！所以不要在学习方法上失败，认真迈好您的第一步。

2.6 脱机运行之 Flash 运行

您的项目工程经过在 **Keil C51** 集成开发环境下的软件模拟、**DP-51PROC** 单片机综合仿真实验仪上的硬件仿真，已基本实现和满足用户要求。但在实际的现场工作环境中它是否仍然符合我们的要求，这就需要制作样机脱机运行。脱机运行用户程序是指用户把经过调试、

仿真后生成的目标代码文件(*.hex)下载(编程、固化)到 TKSMonitor51 仿真器上的单片机内部 Flash 程序存储器中，系统复位后 TKSMonitor51 仿真器将全速执行用户程序，这样 DP-51PROC 单片机综合仿真实验仪就相当于用户的一个样机了。

TKSMonitor51 仿真器具有下载固化用户程序的功能，与上位机 DPFLASH 软件配合使用,把程序下载到实验仪自带的 FLASH 存储区中运行。

2.6.1 如何进入运行状态

首先，如 2.5.1 节的第一段所说的设置好实验仪。然后如 2.6.3 节所说的把项目修改一下配置，并生成 HEX 文件，然后使用 DPFLASH 软件把 HEX 文件下载，最后把 TKSMonitor51 仿真器上的开关切换到 RUN 模式然后按 RESET 按键即可。具体操作见 2.6.3 节。

2.6.2 运行状态的存储器模型

当 TKSMonitor51 仿真器处于运行状态时，将全速执行实验仪内部 Flash 中的用户程序。运行状态下，DP-51PROC 单片机综合仿真实验仪的存储器模型如图 2.36 所示。

FLASH		SRAM	
未使用	0FFFFH	32KB SRAM 存储区	0FFFFH
	8000H		8000H
外部用户程序区 32KB FLASH	7FFFH	用户数据区 用户扩展 I/O 映射区	7FFFH
	0000H		0000H

图 2.36 运行状态下存储器模型

2.6.3 运行程序实例

下面以 Led_Display.asm 源程序为例来介绍如何运行用户程序的步骤及方法。

(1) 在 Keil C51 集成开发环境下打开 Led_Display.µV2 工程，这时会出现如图 2.32 所示的编辑、编译环境，由于在运行状态下 TKSMonitor51 仿真器是运行单片机内部程序存储器中的程序，这样源程序文件必须从 0000H 单元开始存放，这与在 MON51 环境下调试程序不同(其程序存放起始地址为 8000H)。因此在编译源程序文件时，必须对程序及其编译环境进行重新设置。

(2) 首先更改源程序文件中的程序代码定位伪指令语句“ORG 8000H”为“ORG 0000H”，若您的源程序是 C 语言编写的，则系统配置文件“Startup.a51”中的命令代码“CSEG AT 8000H”改为“CSEG AT 0000H”，即让程序从 0000H 地址单元开始装载。

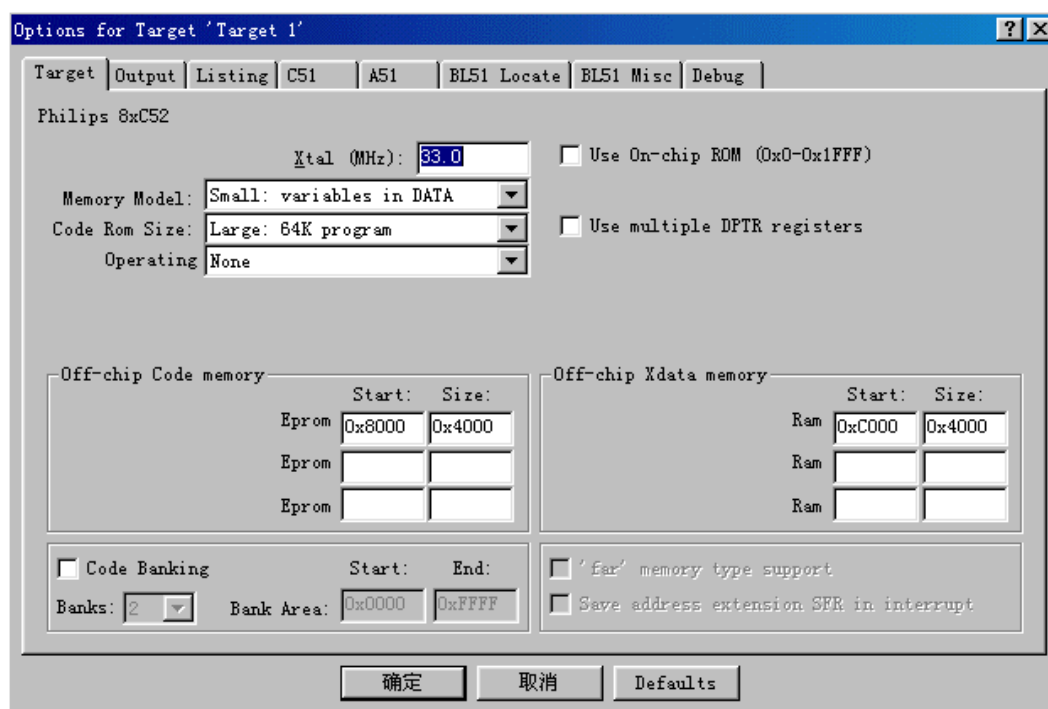


图 2.37 系统编译环境设置界面

(3) 其次重新设置 Keil C51 的编译环境，点击菜单栏上的 **Project** 项，在弹出的下拉式菜单中选择 **Option for target 'target 1'** 命令，将出现如图 2.37 所示的系统环境设置界面。在第一栏 **Target** 选项中把外部程序存储器空间起始地址改为：0000H。在第 2 栏 **Output** 选项中选中 **Create Hexe File**，即表示在编译连接后自动生成目标文件(*.HEX)。而第三栏 **C51** 选项中的中断入口地址向量改为：0000H，点击**确定**命令按钮加以确认。

(4) 然后运行 **Rebuild Target** 命令重新编译连接程序文件，这时会出现如图 2.38 所示的输出提示信息框。其中“Creating hex file from Led_Light”表示正在生成目标文件 Led_Light.hex，接下来的一行“‘Led_Light’ - 0 Error(s), 0 Warning(0).”提示编译成功，程序没有错误，也不存在警告。若在编译连接中出现警告信息用户可以不理会它；但是若提示程序中存在错误，必须查找并修改程序中的错误直至程序编译成功为止。

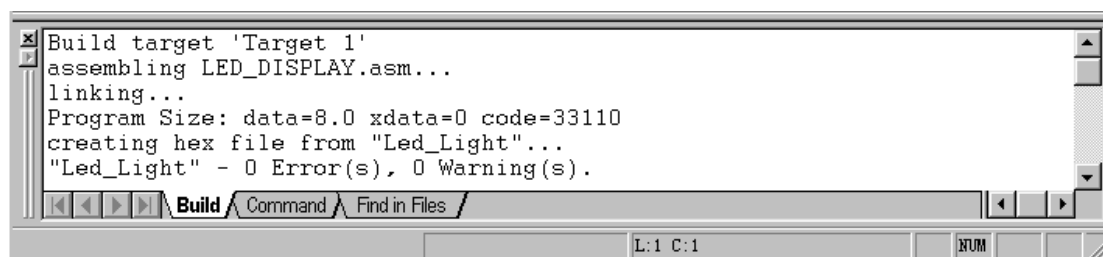


图 2.38 编译输出信息窗口

(5) 拿出随机附送的串口通信电缆并正确连接到 PC 机的串口和 TKSMonitor51 仿真器的串口上。

(6) 复位 TKSMonitor51 仿真器后，将状态开关拨置 LOAD 处。运行上位机的 DPFLASH 下载软件。然后装载 Led_Light.hex 文件，并点击下载软件左边的**编程**命令按钮（见图 2.23），则出现如图 2.39 所示的编程区域选择对话框，用户可以根据下载程序的代码大小加以选择，一般来说 8KB 程序空间已足够大，点击**编程**命令即可把 Led_Light.hex 文件下载到 FLASH 芯片中。



图 2.39 编程区域选择对话框

(7) 最后又把 TKSMonitor51 仿真器上的开关拨到 RUN 的一边，按下复位开关“RESET”，这时 TKSMonitor51 仿真器再次进入用户模式并全速运行用户程序。

2.7 脱机运行之 ISP 单片机运行

您的项目工程经过在 Keil C51 集成开发环境下的软件模拟、TKSMonitor51 仿真器的硬件仿真，已基本实现和满足用户要求。但在实际的现场工作环境中它是否仍然符合我们的要求，这就需要制作样机脱机运行。脱机运行用户程序是指用户把经过调试、仿真后生成的目标代码文件(*.hex)下载(编程、固化)到 DP-51PROC 单片机综合仿真实验仪锁紧座上的单片机内部 Flash 程序存储器中，系统复位后 DP-51PROC 单片机综合仿真实验仪将全速执行用户程序，这样 DP-51PROC 单片机综合仿真实验仪就相当于用户的一个样机了。

DP-51PROC 单片机综合仿真实验仪具有 ISP 下载固化用户程序的功能，与上位机的 ZLGISP 软件配合使用，用户可以对多种具有 ISP 功能的单片机进行在线编程，这样 DP-51PROC 单片机综合仿真实验仪不就是一款性价比高的实验仪+仿真器+下载编程器了吗？

2.7.1 如何进入 ISP 下载状态

首先，在 A1 区的锁紧座上插入具有 ISP 功能的 Flash 单片机(如 P89C60X2、P89C51RD2 等)；然后将 ISP 跳线 JP14 短接，用串口线连接 PC 机和实验仪串口 CZ1，短接 JP15 的 1、2 两个跳线(TXD、RXD)，最后按下复位按键“RESET”。此时，DP-51PROC 单片机综合仿真实验仪即进入 ISP 下载状态。然后再使用 ZLGISP 软件把已经生成的 HEX (HEX 文件的生成请看 2.6.3 节的(1)~(4))文件下载到具有 ISP 功能的 Flash 单片机即可，具体 ZLGISP 的操作见 2.7.3~2.7.5 节。

2.7.2 运行状态的存储器模型

当 DP-51PROC 单片机综合仿真实验仪处于运行状态时，将全速执行单片机内部 Flash 中的用户程序。运行状态下，DP-51PROC 单片机综合仿真实验仪的存储器模型如图 2.40 所示。



图 2.40 运行状态下存储器模型

2.7.3 ZLGISP 软件简介

ZLGISP 下载软件是广州致远电子有限公司开发的在线编程下载软件，其主要功能特点有：

- (1) 与下载器配合使用可对 PHILIPS 公司的 P89C6xX2、P89C51RX2、P89C66x 等 MCU 进行 ISP 编程；
- (2) 与 DP 系列开发实验仪配合使用可对多种具有 ISP 功能的单片机进行在线编程；
- (3) 界面好，操作简单方便；
- (4) 兼容性好，支持多种操作系统。

2.7.4 ZLGISP 软件的安装方法

(1) 首先进入软件所在的目录 E:\Software\ZLGISP(默认路径)下，这时您可以看到 Chinese 和 English 两个子目录，它们分别是该软件的中英文版，下面将以安装中文版的 ZLGISP 下载软件(现已升级为 1.2 版本)为例来介绍 ZLGISP 软件的安装方法。

(2) 进入 Chinese 子目录下，双击 **Setup.exe** 安装文件，此时即可出现如图 2.41 所示的初始化提示信息画面，它提示用户在安装 ZLGISP 软件时应关闭其它应用程序。

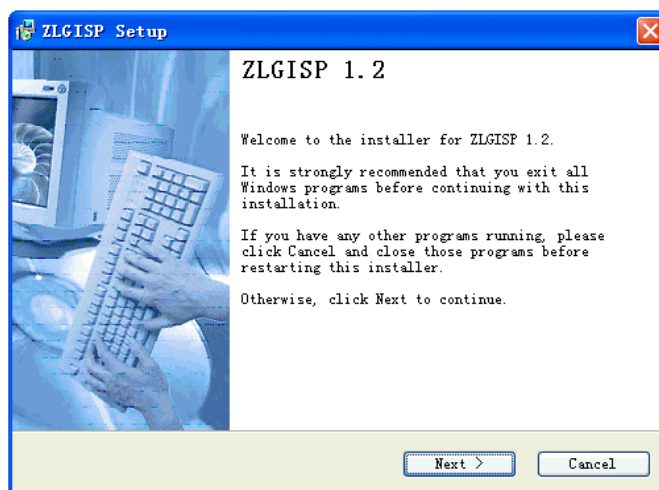


图 2.41 初始化提示信息画面

(3) 在图 2.41 所示的初始化提示信息画面中选择 **Next** 继续安装，此时会出现如图 2.42 所示的 User Information 用户信息对话框，输入相应的用户信息点击 **Next** 即可。

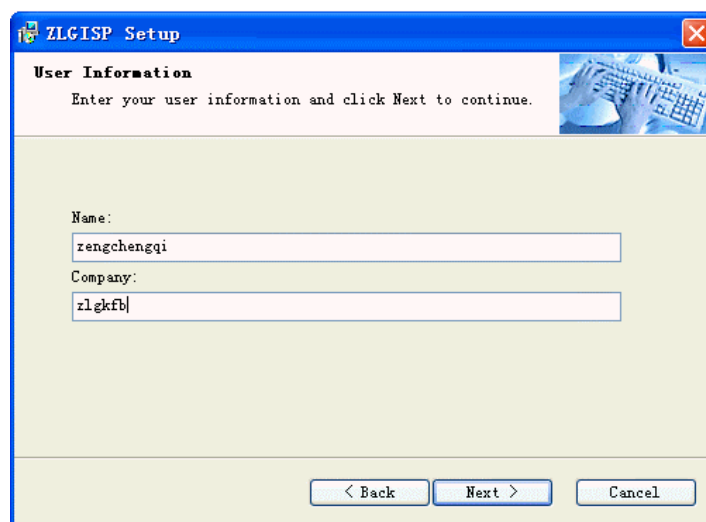


图 2.42 用户信息对话框

(4) 这时会出现如图 2.43 所示的安装目录选择界面，默认路径是 C:\program\ZLGISP，当然可以单击 **Change** 命令按钮来选择适合自己的安装目录，如 D:\program\ZLGISP，然后单击 **Next** 命令按钮即可。

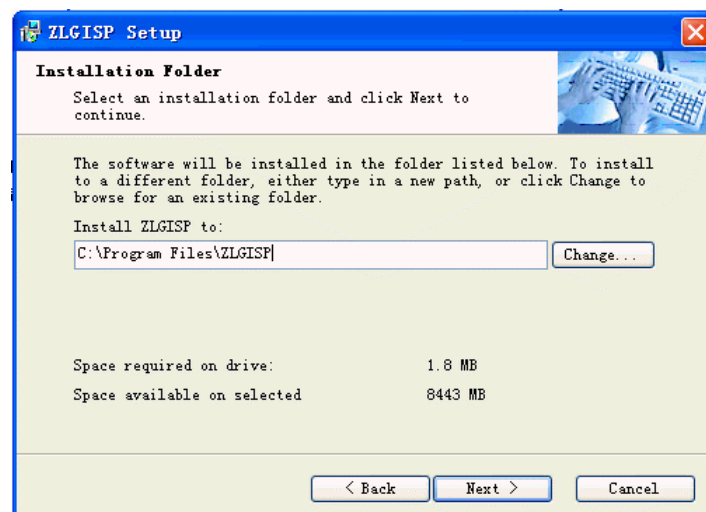


图 2.43 安装目录选择界面

(5) 在接下来的几项确认对话框中都选择 **Next** 选项, 即可自动开始安装。安装完毕会在 WINDWOS 的开始菜单中的程序栏内发现该软件的快捷图标如图 2.44 所示, 单击它就可运行该程序。



图 2.44 快捷图标

2.7.5 ZLGISP 软件的使用方法

(1) 拿出随机附送的串口通信电缆, 一端接在 PC 机的串口上, 而另一端则应连接到 DP-51PROC 单片机综合仿真实验仪的串口 CZ1 上。短接跳线 JP14, 使单片机的 PESN 引脚为低电平“0”, 单片机锁紧座换上支持 ISP 的单片机 (如 P89C60X2 或者 P89C51RD2) 然后打开实验仪的工作电源, 此时电源指示灯 POWER 应该发光。

(2) 双击桌面 ZLGISP 下载软件的快捷图标(如图 2.44 所示), 运行 ZLGISP 下载软件, 这时将出现如图 2.45 所示的 ZLGISP 下载软件的操作界面。由图 2.45 可以看出 ZLGISP 软件也是一个可视化的窗口操作界面, 其主要包括: 标题栏、菜单栏、工具栏、命令窗口、缓冲区窗口和信息提示窗口组成。

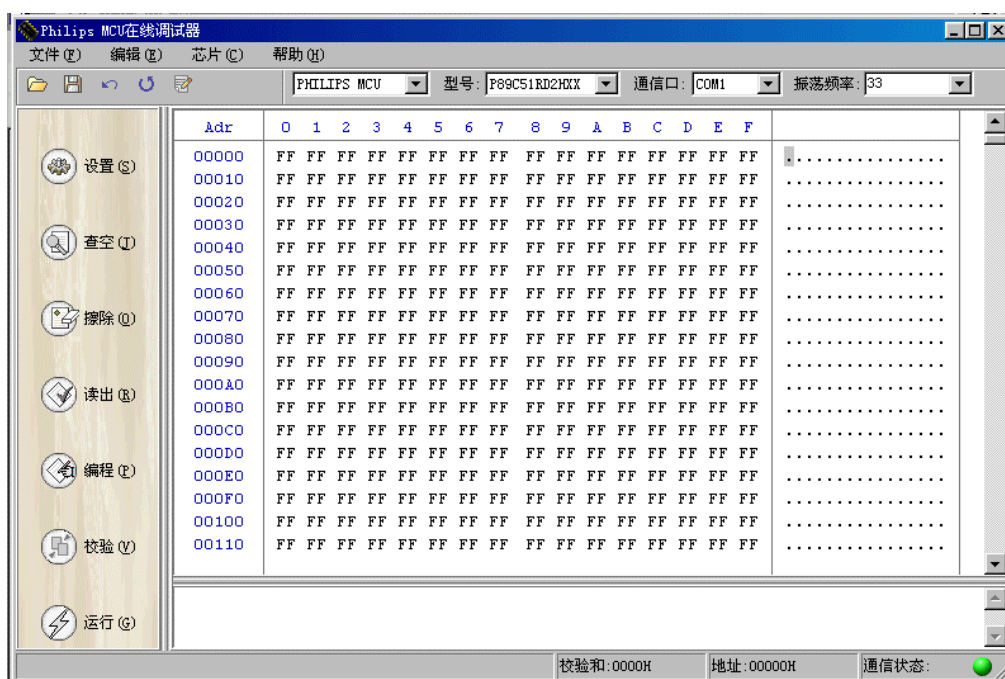


图 2.45 ZLGISP 操作界面

(3) 点击工具栏的 **型号** 选项将弹出如图 2.46 所示的器件选择菜单, 由图可以看出该软件支持所有的 PHILIPS 在线可编程器件(即具有 ISP 功能的单片机), 在使用该功能时用户应该插入具有 ISP 功能的单片机, 如 P89C60X2BP 并在该栏中选择 P89C60X2, 至于通信口和振荡频率用户可以根据实际使用情况灵活选择。

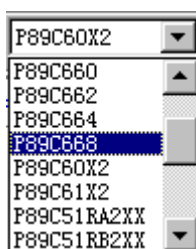


图 2.46 器件选择菜单

(4) 设置完毕用户就可以对所选择的器件 P89C60X2 芯片进行操作了。点击左边命令窗口中的**设置**命令，这时将出现如图 2.47 所示的设置窗口，通过该窗口用户可以读出或写入芯片配置字、加密等级和系统时钟等级等相关内容。由于芯片的状态字(Status)一旦是非 00 时，芯片复位后将直接进入 ISP 功能，这样系统将无法直接运行用户的程序，因此要让系统正常工作，状态字必须写成 00。对于某些芯片该窗口或许不同，如对 P89C51RD2XX 芯片进行配置时，它将多出一个可配置的向量字(Vector)，此时该字节必须写为 FC。



图 2.47 设置窗口

(5) 像其它编程器一样，任何(编程)写操作时必须保证该芯片是空的，否则会提示编程失败。点击命令窗口中的**擦除**命令，在出现如图 2.48 所示的擦除窗口中选择**按页选择**，并选中 **BLOCK 0(0x0000—x1FFF)**选项，最后点击**擦除**命令即可开始擦除操作。当然用户也可以选择**全部芯片空间**对整个芯片进行擦除，但用户应注意执行此操作将擦除芯片的配置字、加密位，若操作不当，芯片编程后系统将不能正常工作，因此建议用户使用按页选择这种方法。(用户也可以从此窗口看出单片机 P89C60X2 芯片共有 64KB FLASH 程序存储器，分为 5 个区：BLOCK0~BLOCK4，其大小分别对应为 8KB、8KB、16KB、16KB、16KB。)

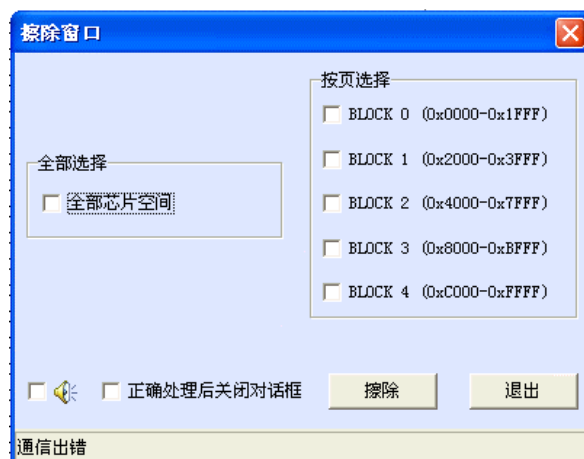


图 2.48 擦除窗口

(6) 点击菜单栏中的文件选项，在弹出如图 2.49 所示的下拉菜单中选择装载命令，这时将出现如图 2.50 所示的装载对话框，选择下载程序的目标代码*.Hex 文件。点击打开命令按钮后程序代码将被装载到 ZLGISP 软件的缓冲区中，如图 2.51 所示，这时表示程序装载成功。

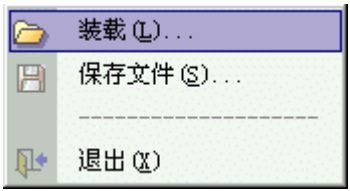


图 2.49 装载命令菜单



图 2.50 装载程序对话框

Adr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000	02	07	AE	E7	09	F6	08	DF	FA	80	46	E7	09	F2	08	DFF.....
00010	FA	80	3E	88	82	8C	83	E7	09	F0	A3	DF	FA	80	32	E3	..>.....2.
00020	09	F6	08	DF	FA	80	78	E3	09	F2	08	DF	FA	80	70	88x.....p.
00030	82	8C	83	E3	09	F0	A3	DF	FA	80	64	89	82	8A	83	E0d.....
00040	A3	F6	08	DF	FA	80	58	89	82	8A	83	E0	A3	F2	08	DFX.....
00050	FA	80	4C	80	D2	80	FA	80	C6	80	D4	80	69	80	F2	80	..L.....i...
00060	33	80	10	80	A6	80	EA	80	9A	80	A8	80	DA	80	E2	80	3.....
00070	CA	80	33	89	82	8A	83	EC	FA	E4	93	A3	C8	C5	82	C8	..3.....
00080	CC	C5	83	CC	F0	A3	C8	C5	82	C8	CC	C5	83	CC	DF	E9
00090	DE	E7	80	0D	89	82	8A	83	E4	93	A3	F6	08	DF	F9	EC
000A0	FA	A9	F0	ED	FB	22	89	82	8A	83	EC	FA	E0	A3	C8	C5"
000B0	82	C8	CC	C5	83	CC	F0	A3	C8	C5	82	C8	CC	C5	83	CC
000C0	DF	EA	DE	E8	80	DB	89	82	8A	83	E4	93	A3	F2	08	DF
000D0	F9	80	CC	88	F0	EF	60	01	0E	4E	60	C3	88	F0	ED	24`..N`....\$
000E0	02	B4	04	00	50	B9	F5	82	EB	24	02	B4	04	00	50	AFP....\$....P.
000F0	23	23	45	82	23	90	00	53	73	BB	01	06	89	82	8A	83	##E.##..Ss.....
00100	E0	22	50	02	E7	22	BB	FE	02	E3	22	89	82	8A	83	E4	."P.."....."
00110	93	22	BB	01	0C	E5	82	29	F5	82	E5	83	3A	F5	83	E0	.".....).....:
00120	22	50	06	E9	25	82	F8	E6	22	BB	FE	06	E9	25	82	F8	"P.%.%".....%
00130	E2	22	E5	82	29	F5	82	E5	83	3A	F5	83	E4	93	22	BB	."..).....:....."

图 2.51 缓冲区的内容

(7) 这时点击命令窗口中的编程命令，在出现如图 2.52 所示的编程对话框的按页选

择栏内选择 **BLOCK 0(0x0000—0x01FFF)** 选项，当然用户也可以用其它编程选项的 **编程文件区** 方式对器件进行编程，编程时若选中了 **包含设置内容** 也可以进行写配置字和加密等操作，最后点击 **编程** 命令即可把程序下载到 DP-51PROC 单片机综合仿真实验仪的 P89C60X2 芯片中。

(8) 注意用户的程序代码虽然已被下载到开发实验仪的 P89C60X2 中，但实验仪此时仍处于 ISP 在线下载状态，用户的程序代码并未执行。若需要运行用户的程序，用户须断开跳线 JP14，即跳开 JP14 上的跳线，而且要按一下复位按钮使系统复位。这样就可以全速运行运行用户下载的程序了。

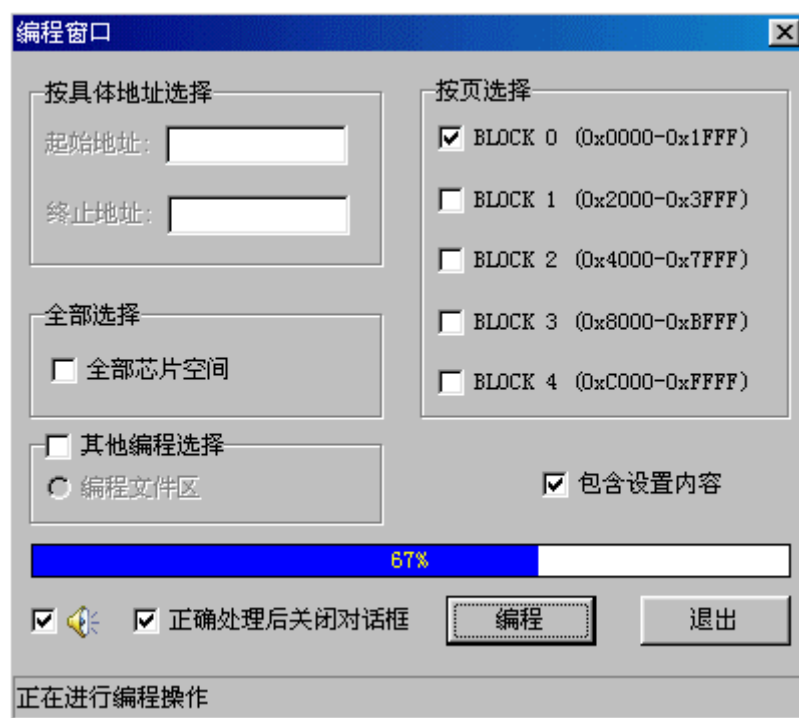


图 2.52 编程窗口

2.8 各功能模块的功能介绍

2.8.1 A1 区 ISP 下载电路

该区除了下载功能外，还包含了 RS232 实验的功能。电路如图 2.53 所示。

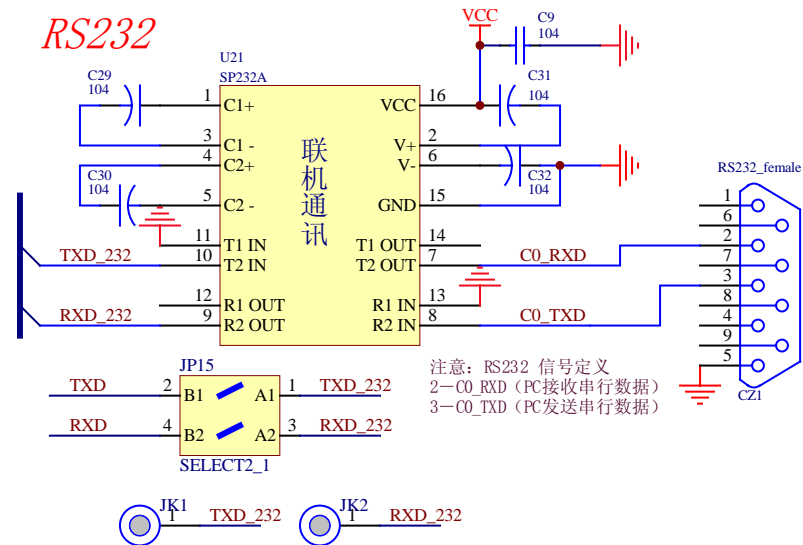


图 2.53 RS232 电路

短接 A1 区的 JP15 跳线组的 1 和 2（出厂设置已经短接好了），就把单片机锁紧座的 RXD 和 TXD 脚与 RS232 电平转换芯片 MAX232 连接起来了。如果用户想外接 RS232 接口，可以把短路器去掉，这时 JP15 的 1 那边是连接到 MAX232 芯片的，2 那边是连接到单片机锁紧座的。用户可以利用引线分别使用。

另外 A1 区的 J76 可以通过 40 针排线与 A2 区的 J79 相连，把单片机的功能管脚外引出去。

2.8.2 A2 区 MCU 总线接口及 IO 口连接区

A2 区是为了方便用户使用单片机的各个引脚，为用户提供了各个管脚的连接接口。各接口简介如表 2.3 所示。

表 2.3 接口一览表

名称	功能
J79	J79 可以通过 40 针排线与 A1 区的 J76 相连，把单片机的功能管脚引过来，也可以通过 TKS- HOOKS 仿真器（选购）的 40 针排线连接来对 DP-51PROC 进行仿真
J61	通过 J79 把单片机的 P1 口引出，可以把它当作 IO 口来使用
J60	通过 J79 把单片机的 P0 口引出，它只能当作单片机总线来使用，不能当作 IO 口使用。
J65	把 JP60 输入的 P0 口信号锁存再输出，它是相当于单片机的低 8 位地址线的引出，不可作 IO 口
J57	通过 J79 把单片机的 P3 口引出，可以把它当作 IO 口或 P3 口对应的特殊功能来使用
J63	通过 J79 把单片机的 P2 口引出，它只能当作单片机总线来使用，不能作 IO 口。

2.8.3 A3 区 138 译码电路

A3 区上面有一片 74HC138 芯片，用户可以在上面进行总线地址译码的实验。电路如图 2.54 所示。

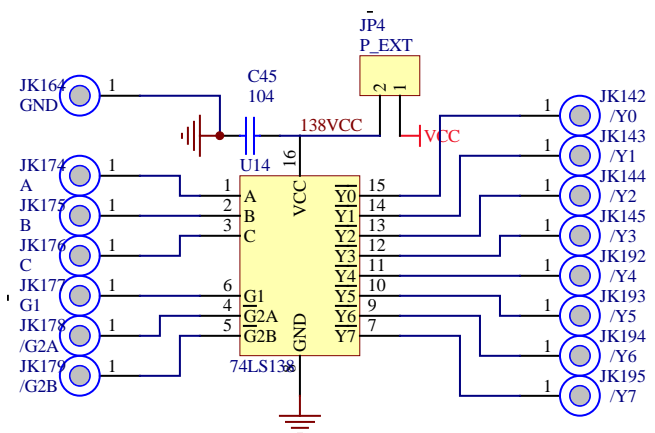


图 2.54 138 译码电路

如图 2.54 所示，/Y0~ /Y7 为译码后的输出端，A、B、C 为地址输入端，/G2A、/G2B、G1 为使能控制端，JP4 为电源控制（使用前先短接，做完实验后再断开）。

2.8.4 A4 区并转串实验电路

A4 区上面有一片 74HC165 芯片，用户可以在上面进行并行转串行的实验。电路如图 2.55 所示。

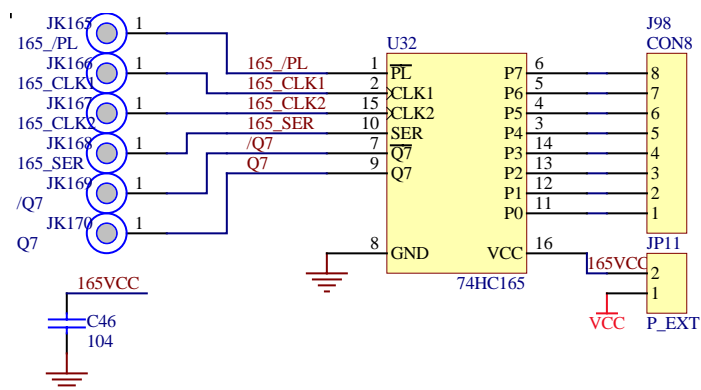


图 2.55 74HC165 电路

如图 2.55 所示，J98 为并行输入端，JP11 为电源控制（使用前先短接，做完实验后再断开），其余 6 个接线柱为串行输出端及控制端。

2.8.5 A5 区串转并实验电路

A5 区上面有一片 74HC164 芯片，用户可以在上面进行串行转并行的实验。电路如图 2.56 所示。

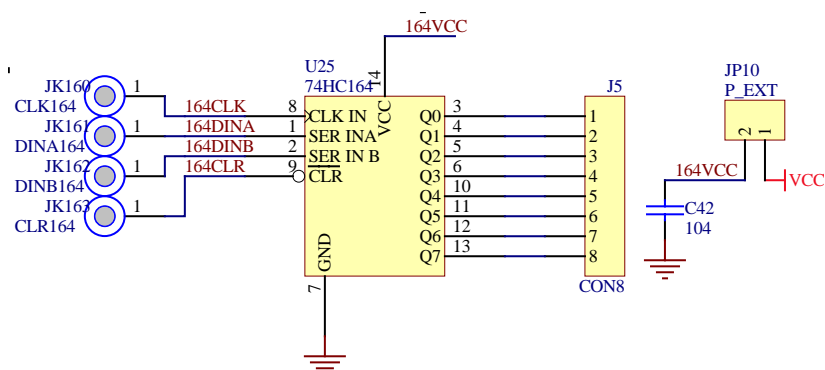


图 2.56 74HC164 电路

如图 2.56 所示，J5 为并行输出端，JP10 为电源控制（使用前先短接，做完实验后再断开），其余 4 个接线柱为串行输入端及控制端。

2.8.6 A6 和 A7 区 PARK 扩展

用户可以通过这两个区来扩展 USB1.1、USB2.0、以太网、CAN—bus 等模块，各区都可以独立连接一个 PARK，也就是说可以同时使用两个 PARK 模块。

A6 区的 J81 和 J83 用于扩展 PARK 模块的固定和连接，7 个接线柱用于 PARK 的 IO 控制连接。A7 区的 J82 和 J84 用于扩展 PARK 模块的固定和连接，7 个接线柱用于 PARK 的 IO 控制连接。I/O 控制连接对应表，如表 2.4 所示。

表 2.4 I/O 控制连接对应表

IO 口	USB 1.1 PARK	USB 2.0 PARK	CAN-bus PARK	以太网 PARK
IO1	-	WAKEUP	-	-
IO2	RSTUSB	RSTUSB	RSTCAN	NETRST
IO3	V_USB	VBUS	-	-
IO4	-	-	-	-
IO5	SUSP	SUSP	-	-
INT	INTUSB	INTUSB	INTCAN	NETINT
CS	CSUSB	CSUSB	CSCAN	NET_CS

如表 2.4 所示，IO2 都是负责模块的复位，INT 都是模块的中断输出脚，CS 都是模块的片选输入脚。而模块的其它管脚如 ADO~AD7、ALE、WR、RD 等引脚已经与单片机的总线直接相连。

2.8.7 B1 区语音实验区

用户可以通过该区进行语音的录放实验。电路如图 2.57 所示。

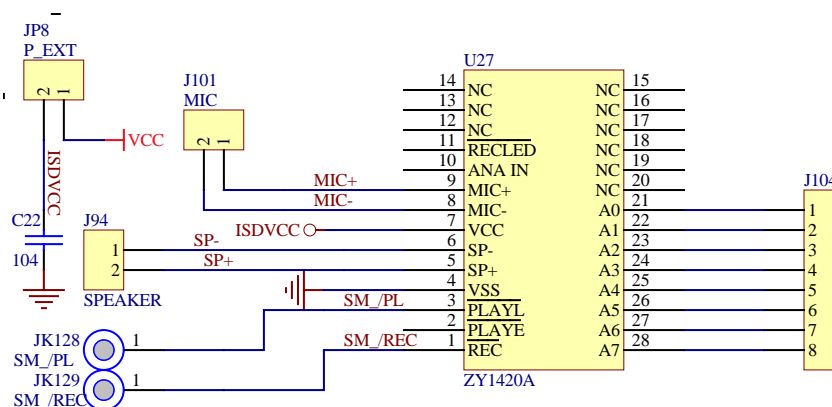


图 2.57 语音模块电路

跳线 JP8 为电源控制，SM_PL 和 SM_REC 为录放控制，J104 为地址选择，J101 为麦克风，J94 为喇叭，位于 B3 区板下方。

2.8.8 B2 区非接触式 IC 卡实验区

用户可以通过购买选配件 ZLG500A MF 卡读卡模块，可以在该区进行非接触式 IC 卡实验。电路如图 2.58 所示。

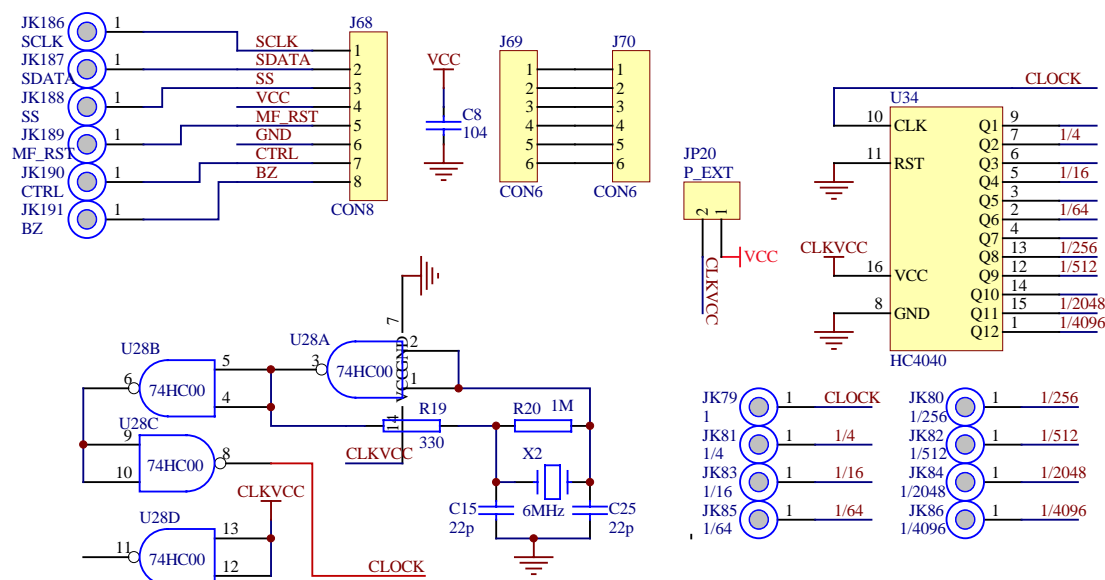


图 2.58 射频读卡模块和时钟源实验电路图

如图 2.58 所示，J68 和 J69 用于连接 ZLG500A 读卡模块，J70 用于连接天线，右侧的 6 个接线柱用于连接控制线和电源。下面的 8 个接线柱为时钟源分频输出接口，它分别输出从 1 分频到 4096 分频的时钟信号，JP20 为时钟源电源控制。注意，射频读卡模块和时钟源实验电路是分别独立的。

另外 B1 区和 B2 区联合起来还可以安装 CPLD 模块（选购），进行 CPLD 的综合实验。

2.8.9 B3 区 LCD 实验区

该模块包含有两个液晶模块接口，可以进行 128*64 图形点阵液晶模块和 16*2 字符点阵液晶模块（选购）的显示实验。电路图如图 2.59 所示。

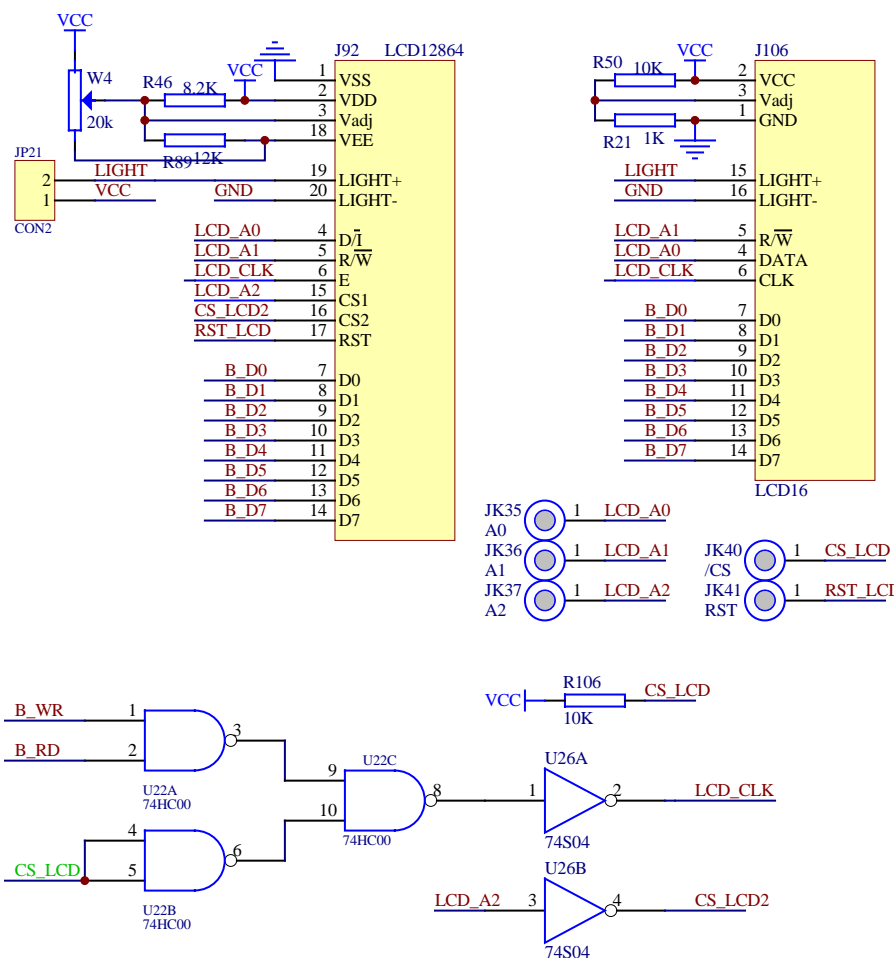


图 2.59 LCD 液晶显示电路

如图 2.59 所示，J106 用于连接 16*2 字符点阵液晶模块，J92 用于连接 128*64 图形点阵液晶模块，JP21 用于 LCD 背光的电源控制(短接为使用背光)，D0~D7 以及 B_WR 和 B_RD 已经与单片机的相应接口相连，其余 5 个接线柱为 LCD 控制 IO 连接线引出接口。

2.8.10 B4 区数字温度采集实验区

B4 区上面一片 18B20 单总线数字温度传感器，用户可以在上面进行单总线温度采集等相关实验。电路如图 2.60 所示。

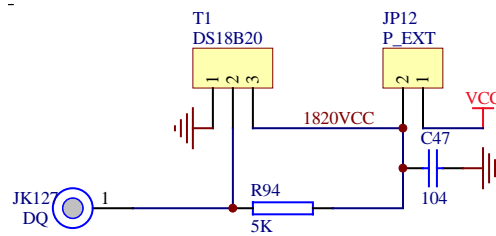


图 2.60 单总线数字温度采集电路

如图 2.60 所示，DQ 为控制信号输入端，JP12 为电源控制（使用前先短接，做完实验再断开）。

2.8.11 B5 区蜂鸣器实验区

B5 区为蜂鸣器电路，用户可以在上面进行蜂鸣器控制的相关实验。电路如图 2.61 所示。

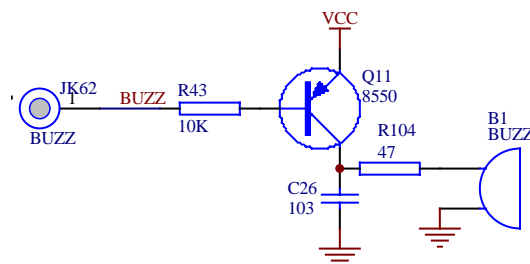


图 2.61 蜂鸣器及驱动电路

如图 2.61 所示，BUZZ 为控制信号输入端。

2.8.12 B6 区 PWM 电压转换实验区

该区可以进行 PWM 转换成电压的实验，电路如图 2.62 所示。

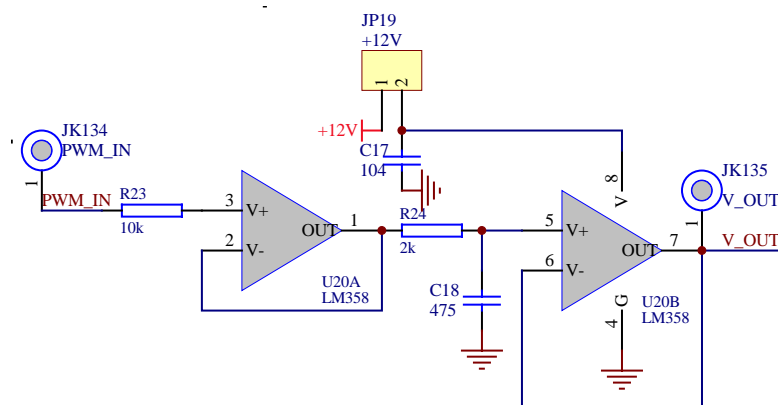


图 2.62 PWM 电压转换电路

如图所示，PWM_IN 为 PWM 信号输入接口，V_OUT 为转换后的电压输出。

2.8.13 B7 区电压基准源

该区用于提供 2.5V~5V 的可调电压。原理图如图 2.63 所示。

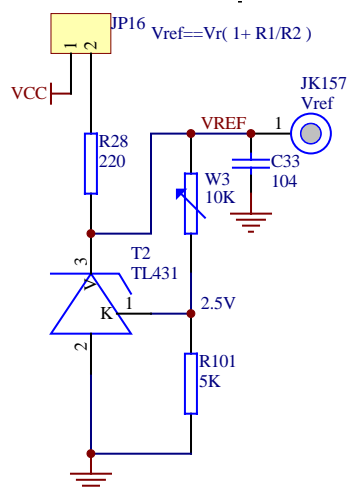


图 2.63 电压基准源

该区采用 TL431 并联型稳压源提供 2.5V~5V 稳定的输出电压，这个输出电压将用于 B7 和 B8 区。当用户调节电位器 W3 时，输出端 Vref 会得到不同的参考电压（公式中的 R1 为 W3 的电阻值，R2 为 R101 就是 5K 欧姆），JP14 是 TL431 的电源供给跳线，短接时该区功能有效。

2.8.14 B8 区串行模数转换实验区

该区用于进行 TLC549 串行 AD 转换实验。电路如图 2.64 所示。

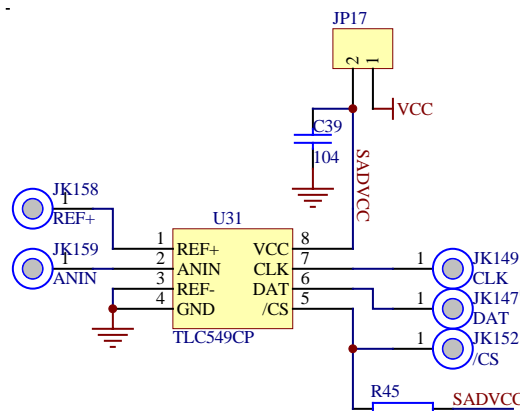


图 2.64 TLC549 串行 A/D 转换电路

如图 2.64 所示，参考电压输入端 REF+ 连接到 TL431 输出电压 Vref 上，CLK、DAT、/CS 是通讯接口，ANIN 是模拟量输入接口，JP17 是电源控制跳线（短接时使用该电路）。

2.8.15 B9 区串行数模转换实验区

B9 区为串行 DAC 实验区，其电路原理图见图 2.65 所示。

如图 2.66 所示, JP18 是电源控制跳线(短接时使用该电路), J78 为控制端, 当 ZDJ_A 的电平为高, ZDJ_B 为低电平则电机正转, 反之则电机反转, 如果两端电平相同则电机不转。

2.8.17 C1 区电压接口区

C1 区是为了方便用户使用 DP-51PROC 实验仪上的电源而引出的接口区。上面提供了 +5V (1A)、+12V (500mA)、-12V(500mA)等接口。另外还有电源指示灯 POWER。

2.8.18 C2 区逻辑笔

C2 区是一个逻辑笔电路, 用户可以利用它来检查实验仪某个接口的 TTL 逻辑电平。电路如图 2.67 所示。

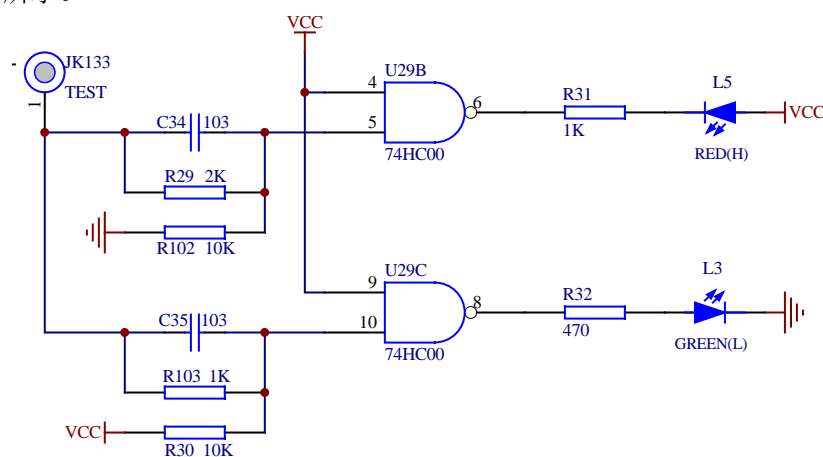


图 2.67 逻辑笔电路

如图 2.67 所示, TEST 为测量引出接口, 当该接口为低电平时, L3 绿色 LED 就会亮, 当该接口为高电平时, L5 红色 LED 就会亮, 平时悬空的时候 L3 和 L5 都不亮。注意: 该逻辑笔只能用于测量 0~5V 的电平。

2.8.19 C3 区 LED 点阵实验模块

C3 区是一个 16×16 LED 点阵及驱动电路。电路如图 2.68 所示。

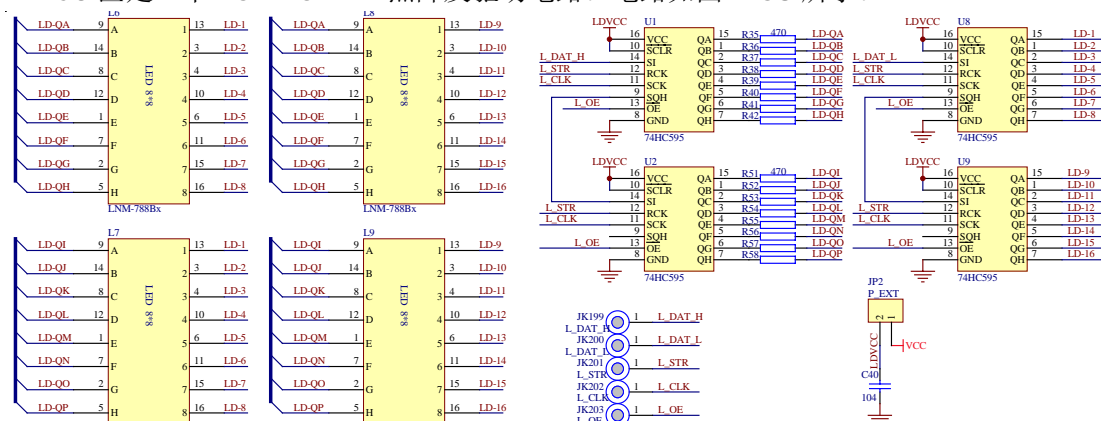


图 2.68 LED 点阵及其驱动电路

上面的 5 个接线柱分别为行列的数据、时钟和控制信号。JP2 为电源控制（使用前先短接，做完实验后再断开）。

2.8.20 C4 区运算放大器电路实验区

C4 区上面有一片 LM324 芯片，还有一些电阻、电容、插座，用户可以在上面进行运放的相关实验。电路如图 2.69 所示。

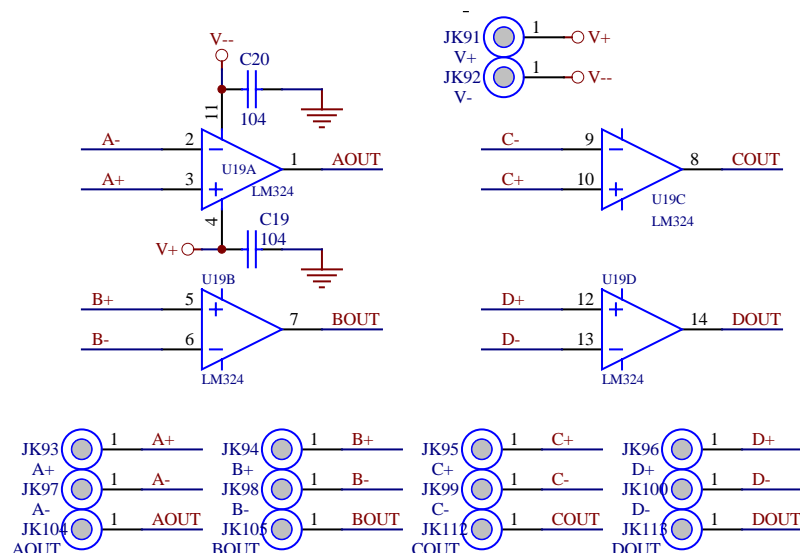


图 2.69 运算放大器电路

如图 2.69 所示，V+、V-为电源控制端（运放的正负电压输入端，使用前用导线引去相应电压源），其余 12 个接线柱为运放引脚的引出端，当用户需要使用电阻时需要到 C5 区去连接。

2.8.21 C5 电阻接口区

该区可以向 LM324 提供外围电阻。该区含有 6 个插件电阻插放位置，当用户插入电阻后，可以用导线连接到其它区。

2.8.22 C6 区 555 电路实验区

C6 区上面有一片 555 芯片，还有电阻、电容插座，用户可以在上面进行 555 的相关实验。电路如图 2.70 所示。

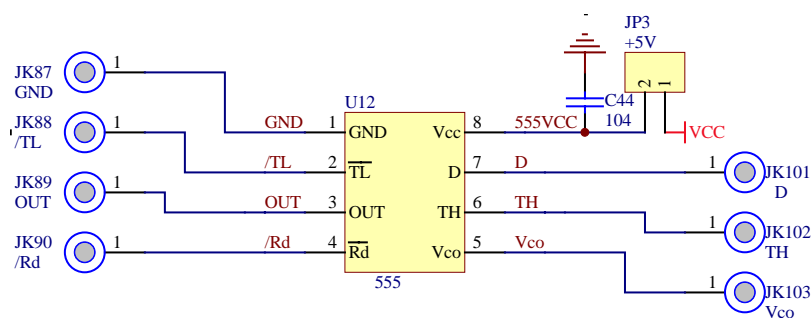


图 2.70 555 实验电路

如图 2.70 所示，7 个接线柱为 555 引脚外引端口，JP3 为电源控制（使用前先短接，做完实验再断开）。

2.8.23 C7 区继电器及其驱动电路

C7 区上面有一片 HRS2H-S-DC5V 继电器，用户可以在上面进行继电器的相关实验。电路如图 2.71 所示。

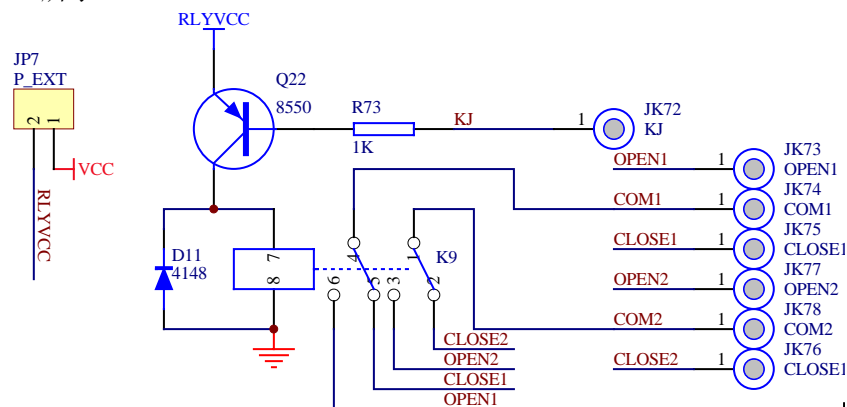


图 2.71 继电器及其控制电路

如图 2.71 所示，KJ 为继电器控制端，其余 6 个接线柱为继电器输入输出端，JP7 为电源控制（使用前先短接，做完实验再断开）。

2.8.24 C8 区步进电机实验区

C8 区上面有一个 4 相步进电机及其驱动电路，用户可以在上面进行步进电机控制的相关实验。电路如图 2.72 所示。

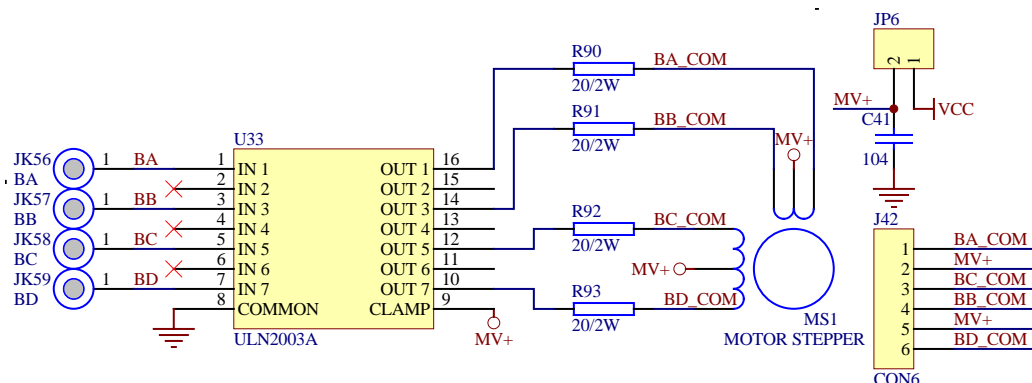


图 2.72 步进电机及其驱动电路

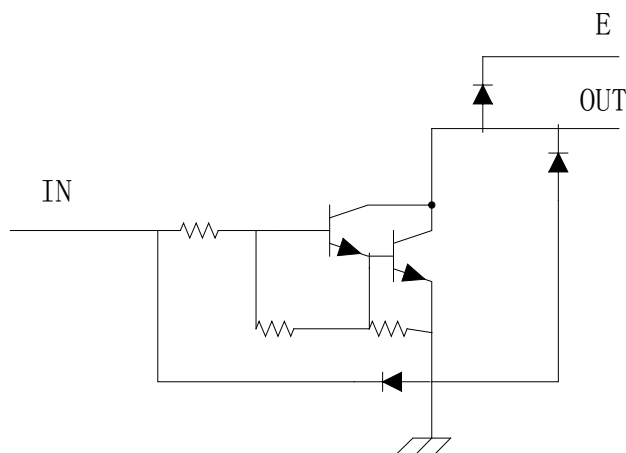


图 2.73 ULN2003A 原理图（可以参考数据手册）

如图 2.72 所示，J42 为步进电机连接接口（出厂时已经连好），BA、BB、BC、BD 为控制信号输入端，JP6 为电源控制（使用前先短接，做完实验再断开）。

2.8.25 D1 区独立控制的 LED、拨动开关、键盘实验区

D1 区上面有 8 个独立的 LED 发光二极管、拨动开关、按键电路，用户可以在上面进行相关实验。电路如图 2.74 所示。

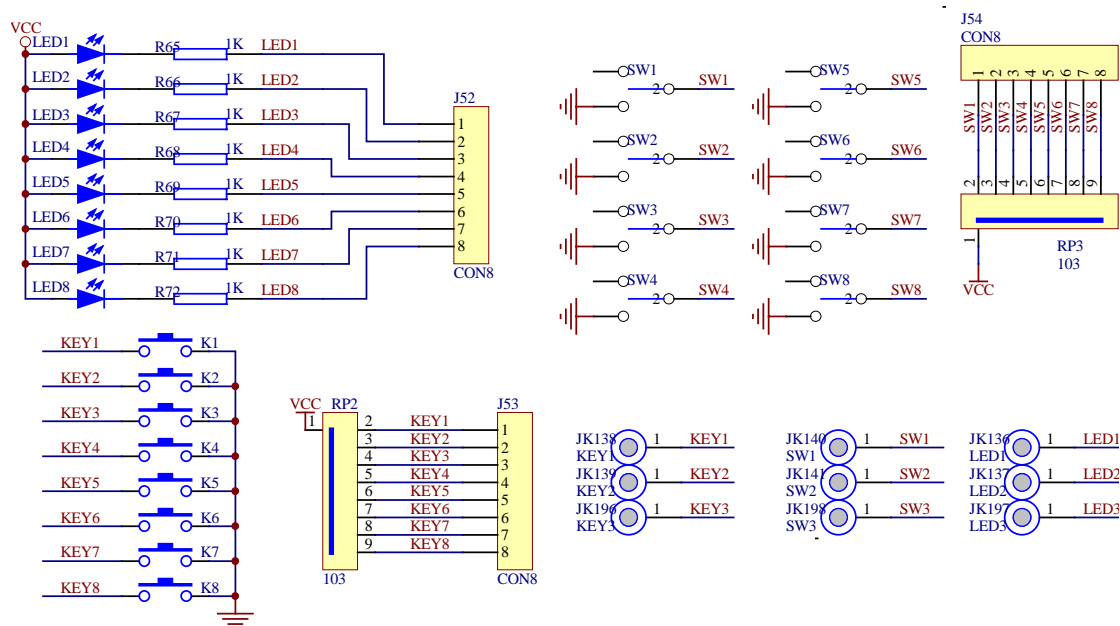


图 2.74 独立控制的 LED、拨动开关、键盘电路

如图 2.74 所示，J52 为 LED 控制接口，J53 为按键输出接口，J54 为拨动开关输出接口，另外，LED、按键和拨动开关的前三位由接线柱引出。

2.8.26 D2 区电位器实验区

D2 区上面有 2 个独立电位器，分别为 1K 欧姆和 10K 欧姆，用户可以在上面进行相关实验。电路如图 2.75 所示。

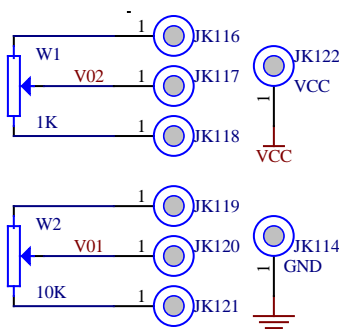


图 2.75 电位器电路

2.8.27 D3 区红外收发实验区

D3 区上面是红外收发电路，用户可以在上面进行红外收发控制的相关实验。电路如图 2.76 所示。

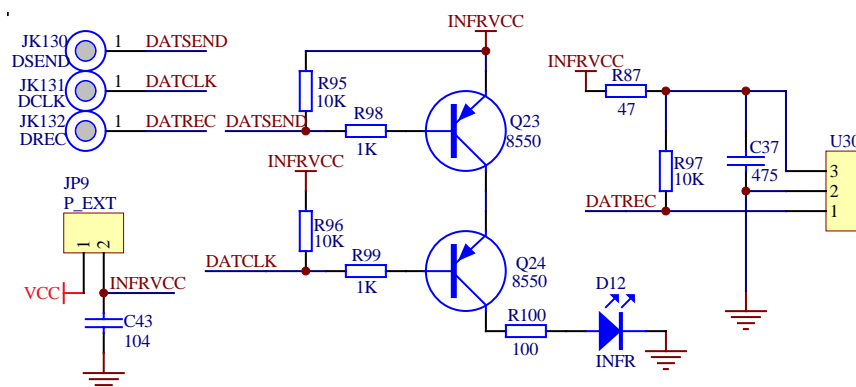


图 2.76 红外收发电路

如图 2.76 所示，3 个接线柱为控制信号输入输出端，U30 为红外接收头模块，JP9 为电源控制（使用前先短接，做完实验再断开）。

2.8.28 D4 区 RS-485 实验区

D4 区上面是一片 RS-485 收发芯片，用户可以在上面进行 RS-485 收发控制的相关实验。电路如图 2.77 所示。

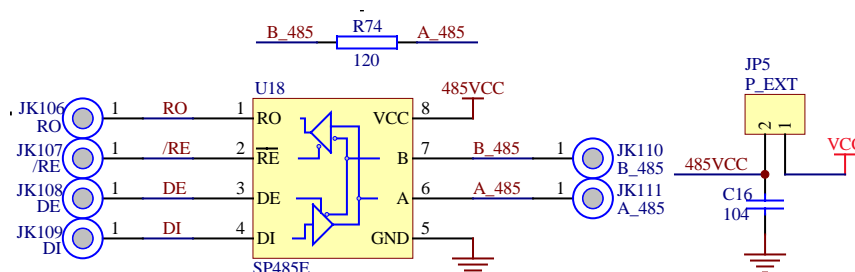


图 2.77 RS-485 电路

如图 2.77 所示，RO、/RE、DE、DI 为控制信号输入输出端，B_485、A_485 为 RS-485 通讯接口，JP5 为电源控制（使用前先短接，做完实验再断开）。另外 R74 是终端匹配电阻（120 欧姆）的插座。

2.8.29 D5 区 I²C 实验区

D5 区上面是 I²C 实验区，有 3 个芯片，分别是键盘 LED 驱动芯片 ZLG7290、EEPROM(256 个字节) CAT24WC02、实时时钟芯片 (RTC) PCF8563T，用户可以在上面进行各种相关的 I²C 实验。电路如图 2.78 所示。

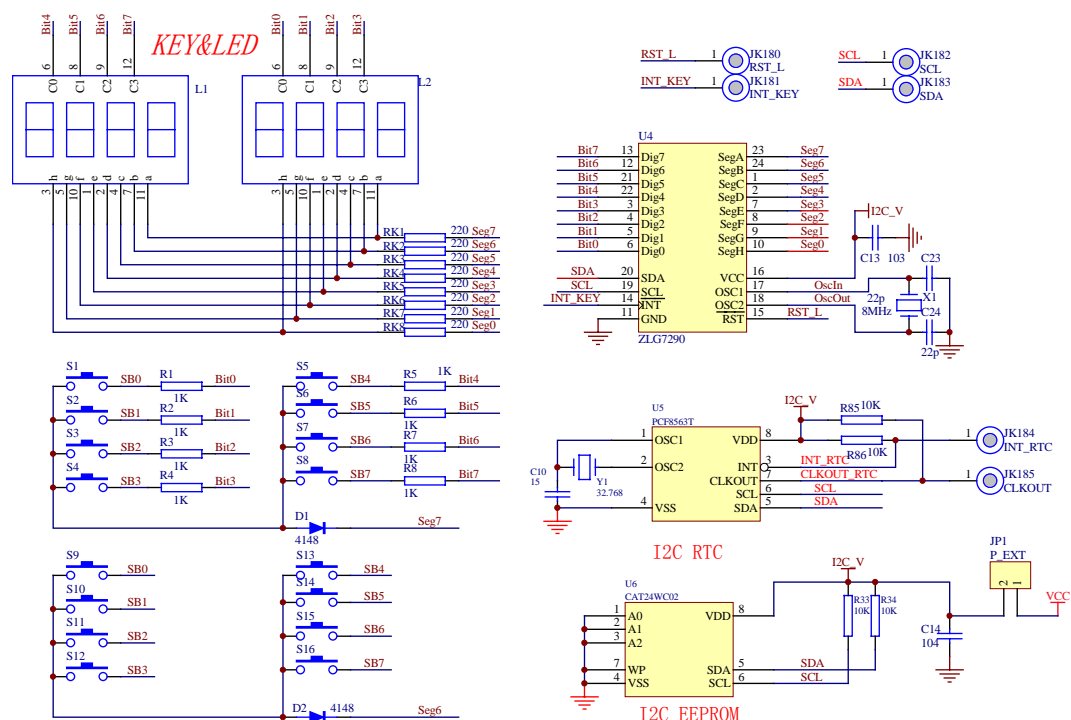


图 2.78 I²C 实验电路

如图 2.78 所示，SCL 和 SDA 为 I²C 控制信号端，RST_L 和 INT_KEY 为 ZLG7290 的复位控制和键盘中断接口，CLKOUT 和 INT_RTC 为 PCF8563T 的时钟输出和中断接口，JP1 为电源控制（使用前先短接，做完实验后再断开）。另外 ZLG7290 芯片同时驱动了 8 个数码管和 16 个按键（键值为 0X01~0X10）。

2.8.30 D6 区接触式 IC 卡实验区

D6 区上面是 IC 卡的卡座，用户可以在上面进行 IC 卡相关的实验。电路如图 2.79 所示。

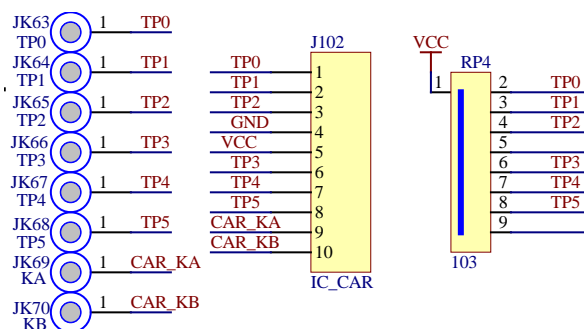


图 2.79 IC 卡电路

如图 2.79 所示，8 个接线柱为 IC 卡控制信号接口，其中 VCC 和 GND 已经连接到卡座上。

第 3 章 DP-51PROC 单片机实验

实验一 Keil C51 集成开发环境的使用练习

一. 实验目的

熟悉 Keil C51 集成开发环境的使用方法

二. 实验设备及器件

IBM PC 机 一台

三. 实验内容

按照本书的第 2 章的 2.1 节到 2.4 节内容进行 Keil C51 集成开发环境的安装和使用练习。然后按照以下内容建立文件并编译产生 HEX 文件。

```
ORG    0000H
LJMP   Main
ORG    00F0H
Main:
MOV    R7, #0
Loop:
MOV    R6, #0
DJNZ   R6, $
DJNZ   R6, $
DJNZ   R6, $
DJNZ   R6, $
DJNZ   R6, $
DJNZ   R7, Loop    ;延时

CPL    P1.0        ; P 1 .0 取反
CPL    P1.1        ; P 1 .1 取反
CPL    P1.2        ; P 1 .2 取反
CPL    P1.3        ; P 1 .3 取反
CPL    P1.4        ; P 1 .4 取反
CPL    P1.5        ; P 1 .5 取反
CPL    P1.6        ; P 1 .6 取反
CPL    P1.7        ; P 1 .7 取反
SJMP   Main
;
END
```

四. 实验要求

熟练掌握 Keil C51 集成开发环境的工程建立、编辑与编译功能。如果还有时间，可以把本书的第 2 章的 2.5 节内容也看一下。

五. 实验预习要求

认真阅读本书的第 2 章的 2.1 节到 2.4 节内容，还有时间，可以把本书的第 2 章的 2.5 节内容也看一下。

六. 实验思考题

- (1) 试写一条把片内 RAM 50H~59H 单元清零的程序。
- (2) 试写一条把片内 RAM 50H~59H 单元写入 01H 的程序。

实验二 基于 Keil C51 集成开发环境的仿真与调试

一. 实验目的

熟悉 Keil C51 集成开发环境调试功能的使用和 DP-51PROC 单片机综合仿真实验仪的使用。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三. 实验内容

按照本书的第 2 章的 2.5 节内容进行 Keil C51 集成开发环境的仿真调试练习。然后按照以下内容建立文件并编译仿真调试。

```
ORG    8000H
LJMP   Main
ORG    80F0H
Main:
MOV    R7, #0
Loop:
MOV    R6, #0
DJNZ   R6, $
DJNZ   R6, $
DJNZ   R6, $
DJNZ   R6, $
DJNZ   R6, $
DJNZ   R7, Loop    ;延时

CPL    P1.0        ; P 1 .0 取反
CPL    P1.1        ; P 1 .1 取反
CPL    P1.2        ; P 1 .2 取反
CPL    P1.3        ; P 1 .3 取反
CPL    P1.4        ; P 1 .4 取反
CPL    P1.5        ; P 1 .5 取反
CPL    P1.6        ; P 1 .6 取反
CPL    P1.7        ; P 1 .7 取反
SJMP   Main
;
END
```

四. 实验要求

熟练掌握结合 DP-51PROC 单片机综合仿真实验仪和 Keil C51 集成开发环境进行仿真调试。如果还有时间，可以把本书的第 2 章的 2.6 节、2.7 节内容也看一下。

五. 实验步骤

1. 用 40 针排线把 DP-51PROC 实验仪上的 A1 区 J76 接口和 A2 区 J79 接口相连，然后使用排线把 A2 区的 J61 接口与 D1 区的 J52 接口相连。如图 3.1 所示。

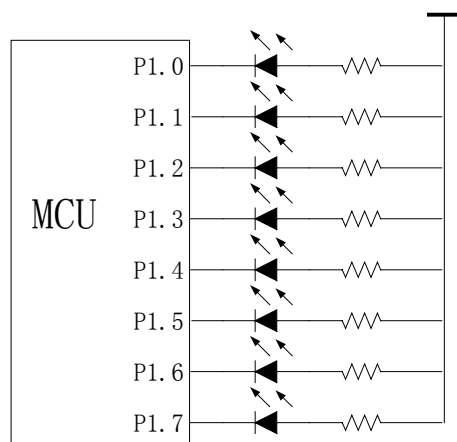


图 3.1 实验 1 原理图

2. 对 DP-51PROC 实验仪上电，然后按照本书的第 2 章的 2.5.1 小节设置 TKSMonitor51 仿真器和使用软件 DPFLASH 把 MON51 监控程序下载到 TKSMonitor51 仿真器。
3. 关闭 DPFlash 软件。把 TKSMonitor51 仿真器的工作模式选择开关切换到 RUN 处，然后按一下复位键（RST），MON51 程序就开始运行了。此时，TKSMonitor51 仿真器进入调试状态。
4. 用户使用 Keil C51 集成开发环境建立工程、编辑与编译“实验内容”所列的程序。然后按照本书的第 2 章 2.5.3 节的第 2 点（软件调试环境的设置）设置好，然后再编译一次。
5. 此时用户就可以按照本书的第 2 章 2.5.4 节所讲述的方法进行仿真调试。如果用户在退出仿真调试模式后想再次进入仿真调试，可以先按一下 TKSMonitor51 仿真器的复位键（RST）。用户可以在仿真调试环境下设置断点，单步，全速运行等。在调试过程中用户可以看见 D1 区的 LED 的亮灭是由用户程序来控制的。

六. 实验预习要求

认真阅读本书的第 2 章的 2.5 节内容。如果还有时间，可以把本书的第 2 章的 2.6 节、2.7 节内容也看一下。

七. 实验思考题

- (1) 如何仿真和调试 C51 程序呢？（用户可以把上面的例子改为 C51 程序然后再编译调试）

实验三 单片机 I/O 口控制实验

一. 实验目的

利用单片机的 P1 口作 IO 口, 使用户学会利用 P1 口作为输入和输出口。

二. 实验设备及器件

IBM PC 机 一台
DP-51PROC 单片机综合仿真实验仪 一台

三. 实验内容

1. 编写一段程序, 用 P1 口作为控制端口, 使 D1 区的 LED 轮流亮。
2. 编写一段程序, 用 P1.0~P1.6 口控制 LED, P1.7 控制 LED 的亮和灭(P1.7 接按键, 按下时 LED 亮, 不按时 LED 灭。)

四. 实验要求

学会使用单片机的 P1 口作 IO 口, 如果有时间用户也可以利用 P3 口作 IO 口来做该实验

五. 实验步骤

1. 用导线把 A2 区的 J61 接口与 D1 区的 J52 接口相连。原理如图 3.2A 所示。

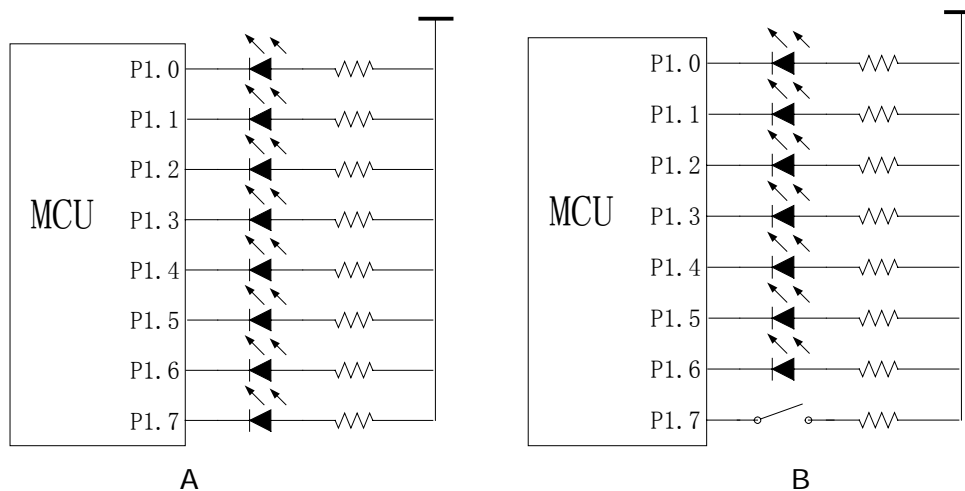


图 3.2 实验 2 原理图

2. 先编写一个延时程序。
3. 将 LED 轮流亮的程序编写完整并调试运行。
4. 使用导线把 A2 区的 J61 接口的 P1.0~P1.6 与 D1 区的 J52 接口的 LED1~LED7 相连, 另外 A2 区 J61 接口的 P1.7 与 D1 区的 J53 的 KEY1 相连。原理如图 3.2B 所示。
5. 编写 P1.7 控制 LED 的程序, 并调试运行。(按下 K1 看是否全亮)
6. A2 区 J61 接口的 P1.7 与 D1 区的 J54 的 SW1 相连。然后再运行程序, 查看结果。

六. 实验预习要求

阅读本书的 2.8.2 节内容，理解该实验的硬件结构。还可以先把程序编好，然后在 Keil C51 环境下进行软件仿真。

七. 实验参考程序

程序 1:

```
ORG 8000H ;此为硬件仿真调试程序，使用软件仿真或直接运行，应改为 0000H
LJMP Main
ORG 8100H ;此为硬件仿真调试程序，使用软件仿真或直接运行，应改为 0100H
```

Main:

```
MOV A,#0FFH
CLR C
```

MainLoop:

```
CALL Delay
RLC A
MOV P1,A ;把 A 的值输出到 P1 口
SJMP MainLoop
```

Delay: ;延时

```
MOV R7, #0
```

Loop:

```
MOV R6, #0
DJNZ R6, $
DJNZ R6, $
DJNZ R6, $
DJNZ R7, Loop
RET
;
END
```

程序 2:

```
ORG 8000H ;此为硬件仿真调试程序，使用软件仿真或直接运行，应改为 0000H
LJMP Main
ORG 8100H ;此为硬件仿真调试程序，使用软件仿真或直接运行，应改为 0100H
```

Main:

```
JB P1.7,SETLED ;按键没有按下时，跳转到 SETLED
```

CLRLED:

```
CLR P1.0
CLR P1.1
CLR P1.2
CLR P1.3
CLR P1.4
```

```
CLR    P1.5
CLR    P1.6
SJMP   Main
SETLED:
SETB   P1.0
SETB   P1.1
SETB   P1.2
SETB   P1.3
SETB   P1.4
SETB   P1.5
SETB   P1.6
SJMP   Main
;
END
```

八. 实验思考题

- (1) 请用户思考一下，想出几个实现以上功能的编程方法。
- (2) 请用户再思考一下，第二个程序中如果使用 KEY1 作为外部中断控制 LED 的亮和灭时，程序应如何修改。

实验四 蜂鸣器驱动实验

一. 实验目的

利用单片机的 P1 口作 IO 口, 使用户学会蜂鸣器的使用。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
频率计	一台

三. 实验内容

1. 编写一段程序, 用 P1.3 口控制(输出 3K 到 4K 频率的方波), 使 B5 区的蜂鸣器发出嘹亮的响声。
2. 按照例程输入一段程序, 用 P1.3 口控制, 使 B5 区的蜂鸣器发出“生日快乐”的音乐。

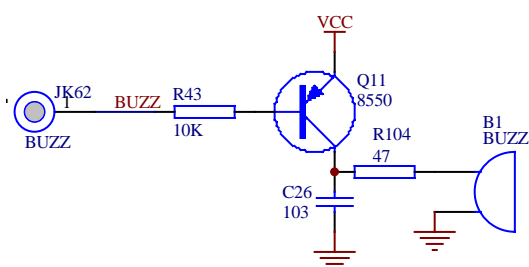


图 3.3 蜂鸣器原理图

四. 实验步骤

1. 使用导线把 A2 区的 P13 与 B5 区的 BUZZ 接线柱相连。
2. 先编写一个延时程序 (120~200us)。
3. 再编写一个循环程序, 改变 P1.3 脚的电平, 然后延时。这样, 这个循环就使 P1.3 口输出一个频率为 2.5KHz~4KHz 的方波。在 DP-51PROC 单片机综合仿真实验仪上运行该程序时, B5 区的蜂鸣器将发出嘹亮的响声
4. 按以下例程输入, 然后运行, 蜂鸣器发出“生日快乐”的音乐。

五. 实验参考程序

```

ORG 8000H
JMP MAIN
ORG 800BH
JMP INTT0
ORG 8100H

MAIN:
MOV SP,#60H
MOV TMOD,#01H ;初始化定时器及器中断
SETB ET0 ;开定时器 0 中断
    
```

```

        SETB  EA
        SETB  TR0           ;启动定时器 0
START0:
        SETB  P1.3
        MOV   30H,#00H
NEXT:
        MOV   A,30H
        MOV   DPTR,#TABLE   ;从 TABLE 中读取数据——声响时间
        MOVC  A,@A+DPTR
        MOV   R2,A
        JZ    ENDD
        ANL   A,#0FH
        MOV   R5,A
        MOV   A,R2
        SWAP  A
        ANL   A,#0FH
        JNZ   SING
        CLR   TR0
        JMP   D1
SING:
        DEC   A
        MOV   22H,A
        RL    A
        MOV   DPTR,#TABLE1  ;从 TALBE1 中读取数据——声调
        MOVC  A,@A+DPTR
        MOV   TH0,A
        MOV   21H,A
        MOV   A,22H
        RL    A
        INC   A
        MOVC  A,@A+DPTR
        MOV   TL0,A
        MOV   20H,A
        SETB  TR0
D1:
        CALL  DELAY         ;声音延时
        INC   30H
        JMP   NEXT
ENDD:
        CLR   TR0
        JMP   START0

INTTO:                                     ;定时器 0 中断服务程序
        PUSH  PSW

```



```

        PUSH ACC
        MOV  TL0,20H
        MOV  TH0,21H
        CPL  P1.3
        POP  ACC
        POP  PSW
        RETI
DELAY:                                     ;R5 的值就是声响持续时间
        MOV  R7,#02
DELAY0:
        MOV  R4,#187
DELAY1:
        MOV  R3,#248
        DJNZ R3,$
        DJNZ R4,DELAY1
        DJNZ R7,DELAY0
        DJNZ R5,DELAY
        RET
TABLE:
        DB 82H,01H,81H,94H,84H,0B4H,0A4H,04H
        DB 82H,01H,81H,94H,84H,0C4H,0B4H,04H
        DB 82H,01H,81H,0F4H,0D4H,0B4H,0A4H,94H
        DB 0E2H,01H,0E1H,0D4H,0B4H,0C4H,0B4H,04H
        DB 82H,01H,81H,94H,84H,0B4H,0A4H,04H
        DB 82H,01H,81H,94H,84H,0C4H,0B4H,04H
        DB 82H,01H,81H,0F4H,0D4H,0B4H,0A4H,94H
        DB 0E2H,01H,0E1H,0D4H,0B4H,0C4H,0B4H,04H,00H
TABLE1:
        DW 64260,64400,64524,64580,64684,64777,64820,64898
        DW 64968,65030,65058,65110,65157,65178,65217
        ;
        END
    
```

六. 实验思考题

- (1) 请用户思考一下，如何通过程序来编写出音乐。

实验五 电子琴实验

一. 实验目的

利用实验仪上提供的按键 K1~K7 作为电子琴按键, 控制蜂鸣器发声, 使用户了解计算机发声原理, 熟悉定时器和键盘扫描电路的工作原理及编程方法。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
频率计	一台

三. 实验内容

1. 编写一段程序, 用 P3.3 口控制(输出 7 种音阶标称频率的方波), 使 B5 区的蜂鸣器发出不同的音调。程序检测按键的状态, 当某一键按下时, 蜂鸣器发出对应的音调。
2. 按照歌曲的音调, 使用 D1 区的按键 K1~K7, 弹奏一首简单的音乐。

四. 实验步骤

1. 用导线将 A2 区 P3.3 口 (INT1) 和 B5 区的 BUZZ 接口相连, 然后将 D1 区的 J53 接口和 A2 区的 J61 接口一一对应相连。如图 3.4 所示。

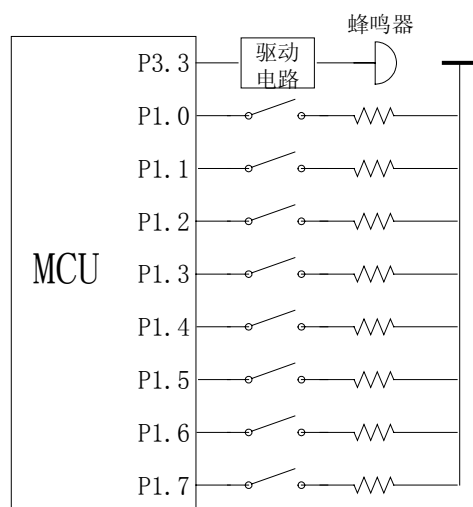


图 3.4 实验 5 原理图

2. 编写按键的动态键盘扫描程序, 根据不同音阶的频率编写蜂鸣器的音调控制程序, 然后完成电子琴的主程序设计。
3. 调试编写好的程序, 使用频率计校准音阶的频率, 然后使用键盘演奏一段好听的音乐。

五. 实验参考程序

```
BUZZ EQU P3.3 ;端口定义
ORG 8000H
LJMP MAIN
```

```

    ORG    800BH
    LJMP   INT_TO
    ORG    8100H

MAIN:
    MOV    SP,#60H      ;初始化堆栈指针
    MOV    P1,#0FFH     ;设置 P1 口为输入模式
    MOV    TMOD,#01H    ;设置定时器 0 为工作模式 1
    SETB   ETO          ;开定时器 0 中断
    SETB   EA           ;开总中断
    CLR    TR0          ;关闭定时器 0

START:
    MOV    R0,P1
    CJNE   R0,#0FFH,KEY1 ;键盘扫描
    CLR    TR0
    SJMP   START

KEY1:
    CJNE   R0,#0FEH,KEY2;K1 键按下
    MOV    30H,#0FBH     ;设置音阶 1
    MOV    31H,#0E9H
    LJMP   SET_TIMER

KEY2:
    CJNE   R0,#0FDH,KEY3;K2 键按下
    MOV    30H,#0FCH     ;设置音阶 2
    MOV    31H,#5CH
    LJMP   SET_TIMER

KEY3:
    CJNE   R0,#0FBH,KEY4;K3 键按下
    MOV    30H,#0FCH     ;设置音阶 3
    MOV    31H,#0C1H
    LJMP   SET_TIMER

KEY4:
    CJNE   R0,#0F7H,KEY5;K4 键按下
    MOV    30H,#0FCH     ;设置音阶 4
    MOV    31H,#0EFH
    LJMP   SET_TIMER

KEY5:
    CJNE   R0,#0EFH,KEY6;K5 键按下
    MOV    30H,#0FDH     ;设置音阶 5
    MOV    31H,#045H
    LJMP   SET_TIMER

KEY6:
    CJNE   R0,#0DFH,KEY7;K6 键按下
    MOV    30H,#0FDH     ;设置音阶 6

```

```
        MOV    31H,#92H
        LJMP   SET_TIMER
KEY7:
        CJNE   R0,#0BFH,NOKEY;K7 键按下
        MOV    30H,#0FDH    ;设置音阶 7
        MOV    31H,#0D6H
SET_TIMER:
        SETB   TR0          ;发声
        SJMP   START
NOKEY:
        CLR    TR0          ;无键按下
        SJMP   START
INT_T0:
                                ;T0 中断服务程序
        MOV    TH0,30H      ;定时器赋初值
        MOV    TL0,31H
        CPL    BUZZ         ;输出方波
        RETI
        ;
        END
```

六. 实验思考题

结合实验仪上的硬件，设计一个可以任意选曲播放的电子音乐盒。

实验六 定时器输出 PWM 实验

一. 实验目的

利用定时器控制产生占空比可变的 PWM 波。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
示波器	一台

三. 实验内容

编写一段程序，用 P1.0 口输出 PWM 波，用 D1 区的按键 KEY1 和 KEY2 实现占空比的增加和降低。用示波器查看 P1.0 口的输出波形。

四. 实验要求

学会使用单片机的定时器产生 250Hz 的 PWM 波。

五. 实验步骤

1. 用导线连接 A2 区的 P11 与 D1 区的 KEY1。
2. 用导线连接 A2 区的 P12 与 D1 区的 KEY2。
3. 将示波器的探针连接到 A2 区的 P10。
4. 用示波器观测 P1.0 口的 PWM 波形。

六. 实验预习要求

认真阅读本节的实验内容，提前做好实验准备工作。

七. 实验参考程序

```
PWMH      DATA  30H      ;高电平脉冲的个数
PWM        DATA  31H      ;PWM 周期
COUNTER    DATA  32H
TEMP       DATA  33H

ORG 8000H
AJMP MAIN
ORG 800BH
AJMP INTTO

ORG 8100H
MAIN:
MOV SP,#60H      ;给堆栈指针赋初值
MOV PWMH,#02H
MOV COUNTER,#01H
```

```

MOV    PWM,#15H
MOV    TMOD,#02H      ;定时器 0 在模式 2 下工作
MOV    TL0,#38H       ;定时器每 200us 产生一次溢出
MOV    TH0,#38H       ;自动重装的值
SETB   ET0            ;使能定时器 0 中断
SETB   EA             ;使能总中断
SETB   TR0            ;开始计时

KSCAN:
JNB     P1.1,K1CHECK   ;扫描 KEY1,
JNB     P1.2,K2CHECK   ;扫描 KEY2,如果按下 KEY2,跳转到 KEY2 处理
程序
SJMP    KSCAN
K1CHECK:
JB      P1.1,K1HANDLE   ;去抖动,按下 KEY1,跳转到 KEY1 处理程序
SJMP    K1HANDLE
K1HANDLE:
MOV     A,PWMH
CJNE    A,PWM,K1H0      ;判断是否到达上边界
SJMP    KSCAN           ;是,则不进行任何操作
K1H0:
MOV     A,PWMH
INC     A
CJNE    A,PWM,K1H1      ;如果在加 1 后到达最大值
CLR     TR0             ;定时器停止
SETB    P1.0            ;置 P1.0 为高电平
SJMP    K1H2
K1H1:
CJNE    A,#02H,K1H2     ;如果加 1 后到达下边界
SETB    TR0             ;重开定时器
K1H2:
INC     PWMH            ;增加占空比
SJMP    KSCAN

K2CHECK:
JB      P1.2,K2HANDLE   ;去抖动,按下 KEY2,跳转到 KEY2 处理程序
SJMP    K1HANDLE
K2HANDLE:
MOV     A,PWMH
CJNE    A,#01H,K2H0     ;判断是否到达下边界
SJMP    KSCAN           ;是,则不进行任何操作
K2H0:
MOV     A,PWMH
MOV     TEMP,PWM
DEC     A

```

```

        CJNE  A,#01H,K2H1      ;如果在减 1 后到达下边界
        CLR   TR0              ;定时器停止
        CLR   P1.0             ;置 P1.0 为低电平
        SJMP  K2H2

K2H1:
        DEC   TEMP
        CJNE  A,TEMP,K2H2      ;如果到达上边界
        SETB  TR0              ;启动定时器

K2H2:
        DEC   PWMH             ;降低占空比
        SJMP  KSCAN

INTT0:
        PUSH  PSW              ;现场保护
        PUSH  ACC
        INC   COUNTER          ;计数值加 1
        MOV   A,COUNTER
        CJNE  A,PWMH,INTT01    ;如果等于高电平脉冲数
        CLR   P1.0             ;P1.0 变为低电平

INTT01:
        CJNE  A,PWM,INTT02     ;如果等于周期数
        MOV   COUNTER,#01H     ;计数器复位
        SETB  P1.0             ;置 P1.0 为高电平

INTT02:
        POP   ACC              ;出栈
        POP   PSW
        RETI
;
END

```

八. 实验思考题

(1) 请用户思考一下，用另一种方式实现定时器产生 PWM 波。

实验七 串转并的 I/O 口实验

一. 实验目的

熟悉并掌握串转并的 I/O 口扩展方法。

二. 实验设备及器件

IBM PC 机 一台
DP-51PROC 单片机综合仿真实验仪 一台

三. 实验内容

1. 写程序，通过单片机的 P1 口控制 74HC164 的串行输入端口，实现串并转换。
2. 验证串并转换数据的正确性。

四. 实验要求

熟悉串并转换芯片的工作原理，学会使用串并转换芯片扩展单片机的 I/O 口资源。

表 3.1 74HC164 真值表

输入				输出			
Clear	Clock	A	B	Q _A	Q _B	...	Q _H
L	X	X	X	L	L		L
H	L	X	X	Q _{AO}	Q _{BO}		Q _{HO}
H	-	H	H	H	Q _{An}		Q _{Gn}
H	-	L	X	L	Q _{An}		Q _{Gn}
H	-	X	L	L	Q _{An}		Q _{Gn}

五. 实验步骤

1. 短接 A5 区 JP10 接口，将 A5 区的 CLK164、DINA164、DINB164、CLR164 与 A2 区的 P10~P13 对应相连（CLK 对 P10 等等）。如图 3.5 所示。

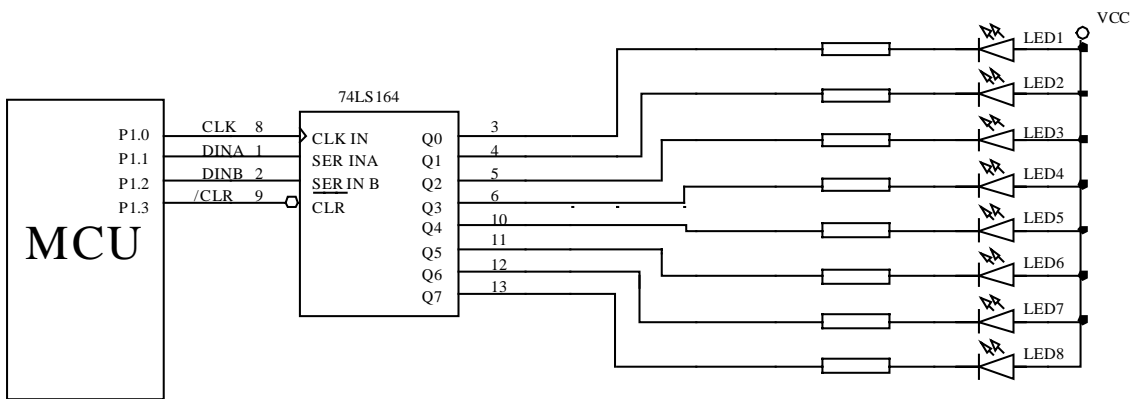


图 3.5 实验 7 原理图

2. 运行编写好的软件程序，完成一次串并转换。
3. 使用 C2 区的逻辑笔或 D1 区的 LED 指示灯测试并行输出数据 Q0~Q7 数据的正确

性。

六. 实验预习要求

认真阅读本实验内容，提前做好实验准备工作。

七. 实验参考程序

```
CLK    EQU    P1.0
DINA   EQU    P1.1
DINB   EQU    P1.2
CLR164 EQU    P1.3
```

```
ORG    8000H
LJMP   MAIN
ORG    8100H
```

MAIN:

```
MOV    SP,#60H    ;设置堆栈指针
NOP                    ;设置以下端口的初始状态
CLR    CLK        ;CLK=0
SETB   DINB       ;DINB=1
CLR    CLR164     ;CLR=0 输出端口清零
SETB   CLR164     ;CLR=1
MOV    A,#0AAH    ;用户输出数据初始化
MOV    R4,#08H
```

SLCHG:

```
RLC    A
MOV    DINA,C      ;串行输出一位数据
SETB   CLK         ;置位时钟
NOP
CLR    CLK
NOP
DJNZ   R4,SLCHG
SJMP   $           ;程序结束,完成一次串并转换
;
END
```

八. 实验思考题

参考图 3.6 电路图，尝试编写软件程序，实现 8 位 LED 流水灯的控制。

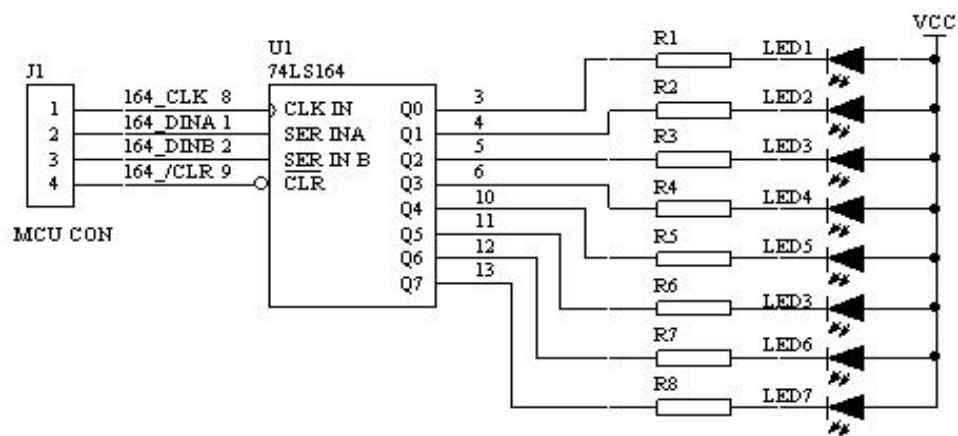


图 3.6 实验原理图

实验八 并转串的 I/O 口实验

- 一. 实验目的
- 熟悉并掌握并转串的 I/O 口扩展方法。
- 二. 实验设备及器件
- IBM PC 机 一台
- DP-51PROC 单片机综合仿真实验仪 一台
- 三. 实验内容
1. 编写程序，通过单片机的 P1 口控制 74HC165 的串行数据输入端口，实现并串转换。
2. 采用 8 位的拨码开关作为输入数据，验证程序设计的正确性。
- 四. 实验要求
- 熟悉并串转换芯片的工作原理，学会使用并串转换芯片扩展单片机的 I/O 口资源。

表 3.2 74HC165 真值表

输入					内部输出		输出
Shift/Load	Clock Inhibit	Clock	串行	并行	QA QB	QH	
				A . . . H			
L	X	X	X	a . . . h	a b		h
H	L	L	X	X	QA0 QB0		QH0
H	L	-	H	X	H QAN		QGN
H	L	-	L	X	L QAN		QGN
H	H	X	X	X	QA0 QB0		QH0

- 五. 实验步骤
1. 短接 A4 区 JP11 跳线，将 A4 区的 165_PL、165_CLK1、165_CLK2、165_SER、/Q7、Q7 与 A2 区的 P10~P15 对应相连 (/PL 对应连接 P10 等等)。如图 3.7 实验原理图。

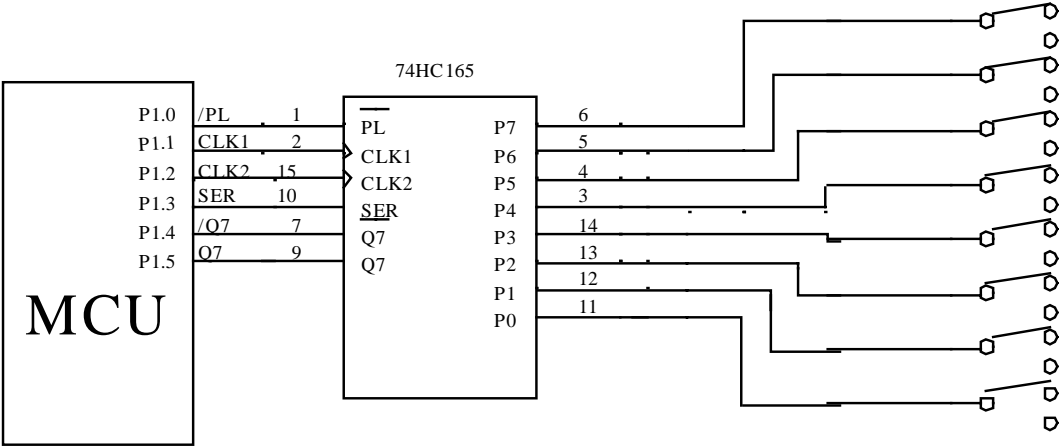


图 3.7 实验 8 原理图

2. 将 D1 区的 J54 接口连接到 A4 区 J98 并行数据输入接口，设置拨码开关 SW1~SW8 的状态。
3. 打开程序调试软件，下载运行编写好的软件程序，完成一次并串转换操作，把拨码开关的状态读出来。
4. 查看程序运行结果是否正确。

六. 实验预习要求

认真阅读本实验内容，提前做好实验准备工作。

七. 实验参考程序

```
PL      EQU      P1.0
CLK1    EQU      P1.1
CLK2    EQU      P1.2
SER      EQU      P1.3
Q7       EQU      P1.5
```

```
ORG      8000H
LJMP     MAIN
ORG      8100H
```

MAIN:

```
MOV      SP,#60H    ;设置堆栈指针
MOV      R4,#00     ;延时
DJNZ     R4,$
MOV      A,#0        ;变量清零
SETB     Q7          ;Q=1,端口设置为输入状态
CLR      SER         ;SER=0
CLR      CLK2        ;CLK2=0
CLR      PL          ;/PL=0
NOP                      ;锁存并行输入数据
SETB     PL          ;/PL=1
NOP
MOV      R4,#08H     ;设置循环变量
CLR      CLK1
```

PLCHG:

```
MOV      C,Q7        ;读入一位串行数据
RLC      A
SETB     CLK1        ;时钟脉冲
NOP
CLR      CLK1
NOP
DJNZ     R4,PLCHG
MOV      R7,A         ;保存数据，8个拨码开关的状态保存于 R7 中
SJMP     $           ;完成一次并串转换，在此设置断点，查看 R7 的内容。
```

;
END

八. 实验思考题

根据图 3.8，采用并串转换方式，尝试编写扩展八位按键输入的键盘动态扫描程序。

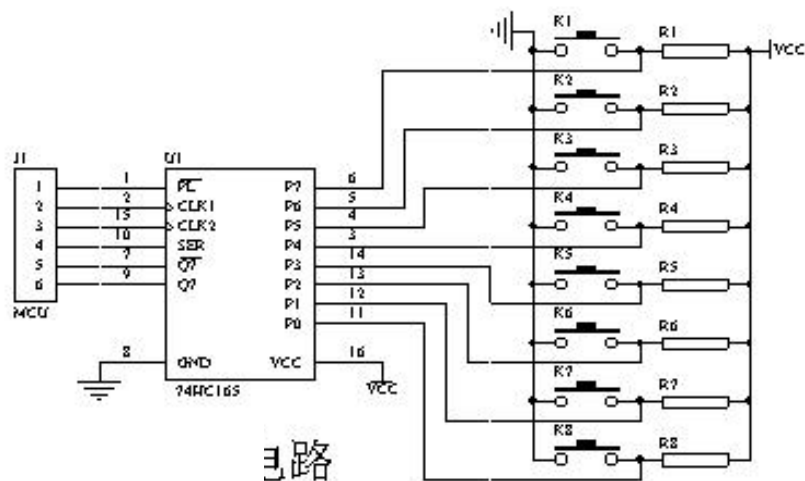


图 3.8 思考题原理图

实验九 74HC138 译码器实验

一. 实验目的

熟悉译码器的使用方法，灵活应用 74HC138 进行电路设计。

二. 实验设备及器件

- IBM PC 机 一台
- DP-51PROC 单片机综合仿真实验仪 一台

三. 实验内容

- 编写程序，通过单片机的 P1 口控制 74HC138 的数据输入端，从而选通相应的数据输出位。
- 将译码数据输出端口连接到 8 个 LED 指示灯，验证译码的正确性。

四. 实验要求

通过实验掌握译码器的工作原理和实际应用方法。

表 3.3 74HC138 真值表和管脚图

Inputs					Outputs							
Enable		Select										
G1	G2*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

A 1 16 VCC
B 2 15 Y0
C 3 14 Y1
G2A 4 13 Y2
G2B 5 12 Y3
G1 6 11 Y4
Y7 7 10 Y5
GND 8 9 Y6

五. 实验步骤

- 短接 A3 区 JP4 接口上的短路帽，将 A3 区 A、B、C、/G1、/G2A、/G2B 与 A2 区的 P10~P15 相连。如图 3.9 所示。

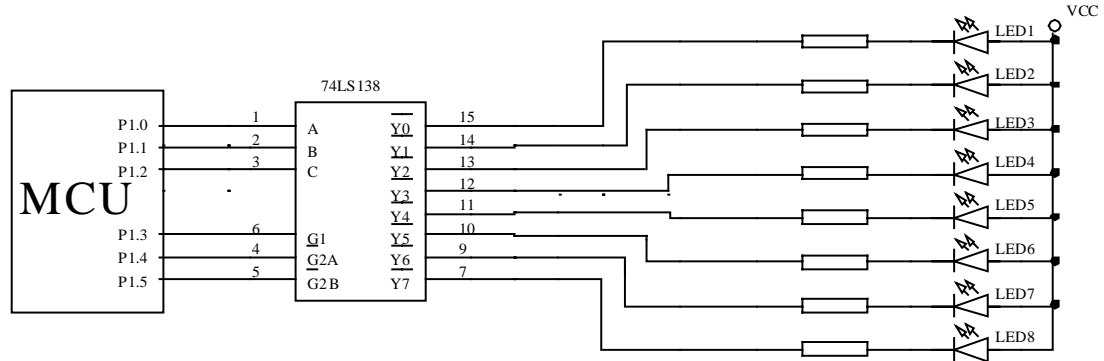


图 3.9 实验 9 原理图

2. 将 D1 区的 LED1、LED2、LED3 分时连接到 A3 区译码数据输出接口/Y0~Y7。
3. 打开程序调试软件，下载运行编写好的软件程序，查看程序运行结果是否正确。

六. 实验预习要求

认真阅读本书这一节的实验内容，提前做好实验准备工作。

七. 实验参考程序

```
ORG    8000H ;此为硬件仿真程序，使用软件仿真或直接运行，应改为 0000H
LJMP   Main
ORG    8100H ;此为硬件仿真程序，使用软件仿真或直接运行，应改为 0100H
```

MAIN:

```
MOV     SP,#60H
MOV     R4,#0
DJNZ    R4,$
        ;设置译码器使能

CLR     P1.5
CLR     P1.4
SETB    P1.3
        ;译码数据输入

CLR     P1.0 ;设置 A=0
CLR     P1.1 ;设置 B=0
SETB    P1.2 ;设置 C=1
SJMP    $
;
END
```

八. 实验思考题

请用户思考一下，在单片机电路里面，74HC138 是如何产生片选信号的。

实验十 16×16 LED 扫描输出实验

一. 实验目的

使用户学会利用单片机的 IO 口进行 LED 点阵的扫描显示。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三. 实验内容

编写一段程序，用 P1 口控制 C3 区 4 片 74HC164 进行行列扫描，使 C3 区的 16×16 LED 点阵显示用户指定的汉字。

四. 实验要求

学会使用单片机对 LED 点阵进行扫描显示。

五. 实验步骤

1. 使用导线将 A2 区的 P10~P14 与 C3 区的 L_DAT_H、L_DAT_L、L_CLK、L_OE、L_STR 依次连接。
2. 运行光盘中的程序，显示“感谢您使用 DP-51PROC 单片机综合仿真实验仪！”，先是向左滚动，后是向上滚动
3. 如果用户希望改变显示的汉字，可以使用光盘中附带的字模提取软件 (Pctolcd2002) 提取用户希望显示汉字的字模，字模提取软件的设置为阴码、逐列式、逆向、十六进制、C51 格式自定义，将生成的字模拷贝到程序中所指定的位置插入即可。

六. 实验预习要求

阅读本书的 2.8.19 节内容，理解硬件结构，还可以先把程序编好，然后在 Keil C51 环境下进行软件仿真。

七. 实验参考程序

```
#include <reg52.h>
typedef unsigned char byte;
typedef unsigned int word;
sbit datah595=P1^0;
sbit datal595=P1^1;
sbit clk595=P1^2;
sbit oe595=P1^3;
sbit str595=P1^4;
word data datah,datal;
byte code displaydata[]=
{
```


0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
//在以下位置插入字模
0x00,0x04,0x00,0x43,0xFC,0x70,0x14,0x00,0xD4,0x39,0x54,0x41,
0x54,0x41,0xD4,0x49,
0x04,0x50,0x3F,0x42,0xC4,0x41,0x45,0x61,0x36,0x0A,0x04,0x34,
0x00,0x27,0x00,0x00,/*"感",0*/

0x40,0x00,0x42,0x00,0xCC,0x3F,0x04,0x50,0x00,0x29,0xFC,0x11,
0x56,0x4D,0x55,0x83,
0xFC,0x7F,0x50,0x00,0x90,0x41,0x10,0x80,0xFF,0x7F,0x10,0x00,
0x10,0x00,0x00,0x00,/*"谢",1*/

0x80,0x00,0x40,0x20,0x30,0x38,0xFC,0x03,0x03,0x38,0x90,0x40,
0x68,0x40,0x06,0x49,
0x04,0x52,0xF4,0x41,0x04,0x40,0x24,0x70,0x44,0x00,0x8C,0x09,
0x04,0x30,0x00,0x00,/*"您",2*/

0x40,0x00,0x20,0x00,0xF0,0x7F,0x1C,0x00,0x07,0x40,0xF2,0x41,
0x94,0x22,0x94,0x14,
0x94,0x0C,0xFF,0x13,0x94,0x10,0x94,0x30,0x94,0x20,0xF4,0x61,
0x04,0x20,0x00,0x00,/*"使",3*/

0x00,0x80,0x00,0x40,0x00,0x30,0xFE,0x0F,0x22,0x02,0x22,0x02,
0x22,0x02,0x22,0x02,
0xFE,0xFF,0x22,0x02,0x22,0x02,0x22,0x42,0x22,0x82,0xFE,0x7F,
0x00,0x00,0x00,0x00,/*"用",4*/

0x08,0x20,0xF8,0x3F,0x08,0x20,0x08,0x20,0x08,0x20,0x10,0x10,
0xE0,0x0F,0x00,0x00,/*"D"*/
0x08,0x20,0xF8,0x3F,0x08,0x21,0x08,0x01,0x08,0x01,0x08,0x01,
0xF0,0x00,0x00,0x00,/*"P", 5*/
0x00,0x00,0x00,0x00,0x00,0x00,0x80,0x00,0x80,0x00,0x80,0x00,
0x80,0x00,0x80,0x00,
0x80,0x00,0x80,0x00,0x80,0x00,0x80,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,/*"—",6*/
0x00,0x00,0xF8,0x19,0x08,0x21,0x88,0x20,0x88,0x20,0x08,0x11,
0x08,0x0E,0x00,0x00,/*"5"*/
0x00,0x00,0x10,0x20,0x10,0x20,0xF8,0x3F,0x00,0x20,0x00,0x20,
0x00,0x00,0x00,0x00,/*"1", 7*/

0x08,0x20,0xF8,0x3F,0x08,0x21,0x08,0x01,0x08,0x01,0x08,0x01,
0xF0,0x00,0x00,0x00,/*"P"*/
0x08,0x20,0xF8,0x3F,0x88,0x20,0x88,0x00,0x88,0x03,0x88,0x0C,
0x70,0x30,0x00,0x20,/*"R", 8*/
0xE0,0x0F,0x10,0x10,0x08,0x20,0x08,0x20,0x08,0x20,0x10,0x10,
0xE0,0x0F,0x00,0x00,/*"O"*/
0xC0,0x07,0x30,0x18,0x08,0x20,0x08,0x20,0x08,0x20,0x08,0x10,
0x38,0x08,0x00,0x00,/*"C", 9*/

0x00,0x08,0x00,0x08,0xF8,0x0B,0x28,0x09,0x29,0x09,0x2E,0x09,
0x2A,0x09,0xF8,0xFF,
0x28,0x09,0x2C,0x09,0x2B,0x09,0x2A,0x09,0xF8,0x0B,0x00,0x08,
0x00,0x08,0x00,0x00,/*"单",10*/

0x00,0x80,0x00,0x40,0x00,0x30,0xFE,0x0F,0x10,0x01,0x10,0x01,
0x10,0x01,0x10,0x01,
0x10,0x01,0x1F,0x01,0x10,0x01,0x10,0xFF,0x10,0x00,0x18,0x00,
0x10,0x00,0x00,0x00,/*"片",11*/

0x08,0x04,0x08,0x03,0xC8,0x00,0xFF,0xFF,0x48,0x00,0x88,0x41,
0x08,0x30,0x00,0x0C,
0xFE,0x03,0x02,0x00,0x02,0x00,0x02,0x00,0xFE,0x3F,0x00,0x40,
0x00,0x78,0x00,0x00,/*"机",12*/

0x20,0x22,0x30,0x23,0xA8,0x22,0x67,0x12,0x32,0x12,0x00,0x20,
0x0C,0x11,0x24,0x0D,
0x24,0x41,0x25,0x81,0x26,0x7F,0x24,0x01,0x24,0x05,0x24,0x09,
0x0C,0x31,0x00,0x00,/*"综",13*/

0x40,0x00,0x40,0x00,0x20,0x00,0x50,0x7E,0x48,0x22,0x44,0x22,
0x42,0x22,0x41,0x22,
0x42,0x22,0x44,0x22,0x68,0x22,0x50,0x7E,0x30,0x00,0x60,0x00,
0x20,0x00,0x00,0x00,/*"合",14*/

0x40,0x00,0x20,0x00,0x10,0x00,0xEC,0x7F,0x07,0x40,0x0A,0x20,
0x08,0x18,0x08,0x06,
0xF9,0x01,0x8A,0x10,0x8E,0x20,0x88,0x40,0x88,0x20,0xCC,0x1F,
0x88,0x00,0x00,0x00,/*"仿",15*/

0x00,0x10,0x04,0x90,0x04,0x90,0x04,0x50,0xF4,0x5F,0x54,0x35,
0x5C,0x15,0x57,0x15,
0x54,0x15,0x54,0x35,0x54,0x55,0xF4,0x5F,0x04,0x90,0x06,0x90,
0x04,0x10,0x00,0x00,/*"真",16*/

```
0x00,0x00,0x10,0x82,0x0C,0x82,0x04,0x42,0x4C,0x42,0xB4,0x23,
0x94,0x12,0x05,0x0A,
0xF6,0x07,0x04,0x0A,0x04,0x12,0x04,0xE2,0x14,0x42,0x0C,0x02,
0x04,0x02,0x00,0x00,/*"实",17*/
```

```
0x02,0x08,0xFA,0x08,0x82,0x04,0x82,0x24,0xFE,0x40,0x80,0x3F,
0x40,0x22,0x60,0x2C,
0x58,0x21,0x46,0x2E,0x48,0x20,0x50,0x30,0x20,0x2C,0x20,0x23,
0x20,0x20,0x00,0x00,/*"验",18*/
```

```
0x40,0x00,0x20,0x00,0xF0,0xFF,0x0C,0x00,0x03,0x40,0x00,0x40,
0x38,0x20,0xC0,0x10,
0x01,0x0B,0x0E,0x04,0x04,0x0B,0xE0,0x10,0x1C,0x20,0x00,0x60,
0x00,0x20,0x00,0x00,/*"仪",19*/
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0xF0,0x5F,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,/*"! ",20*/
```

//至此字模插入结束

```
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00
```

```
};
```

```
byte *p=&displaydata[0];
```

```
byte *q=&displaydata[32];
```

```
void delay(word a)
```

```
{
```

```
    word b;
```

```
    for(b=0;b<a;b++);
```

```
}
```

```
void senddata(word datah,datal)
```

```
{
```

```
    byte i=0;
```

```
    word m,n;
```

```
    oe595=0;
```

```
    str595=0;
```

```
    for(;i<16;i++)
```

```
    {
```

```
        clk595=0;
```

```
        m=datah;
```

```
        n=~datal;
```

```

        m&=0x8000;
        n&=0x8000;
        datah595=(bit)m;
        datal595=(bit)n;
        datah<<=1;
        datal<<=1;
        clk595=1;
    }
    str595=1;
    str595=0;
}

void horizontal(byte time,word counth) //水平移动子程序
{
    byte x,y;
    word j,k,z;
    for(z=0;z<counth;z++)
    {
        for(y=0;y<time;y++)
        {
            datal=0x0001;
            for(x=0;x<16;x++)
            {
                p+=3;
                j=(word)*p;
                j<<=8;
                j&=0xff00;
                p-=1;
                k=(word)*p;
                k&=0x00ff;
                datah=j|k;
                if(x)
                {
                    datal<<=1;
                }
                //datah=~datah;
                senddata(datah,datal);
            }
            p-=32;
        }
        p+=2;
    }
    p=&displaydata[0];
    oe595=1;
}

```

```

}

void vertical(byte a,time,word countv) //垂直移动子程序
{
    byte x,y,e,w=0;
    word j,k,z;
    word datah1,datah2;
    for(z=countv;z>0;z--)
    {
        for(e=0;e<16;e++)
        {
            for(y=0;y<time;y++)
            {
                datal=0x0001;
                for(x=0;x<16;x++)
                {
                    p+=3;
                    j=(word)*p;
                    j<<=8;
                    j&=0xff00;
                    p-=1;
                    k=(word)*p;
                    k&=0x00ff;
                    datah1=j|k;
                    datah1>>=w;
                    q+=3;
                    j=(word)*q;
                    j<<=8;
                    j&=0xff00;
                    q-=1;
                    k=(word)*q;
                    k&=0x00ff;
                    datah2=j|k;
                    datah2<<=(16-w);
                    datah=datah1|datah2;
                    if(x)
                    {
                        datal<<=1;
                    }
                    //datah=~datah;
                    senddata(datah,data1);
                }
                p-=32;q-=32;
            }
        }
    }
}

```

```
        w++;
        if(w==16) w=0;
    }
    if((a==16)&&(z==2))
    { p+=32;q+=16; }
    else
    { p+=32;q+=32; }
}
p=&displaydata[0];
q=&displaydata[32];
oe595=1;
}

void main(void)
{
    byte time=8;
    word size=sizeof(displaydata);
    word countv=((size-2)>>5)-1;
    word counth=countv<<4;
    byte a=(byte)((size-2)%32);
    if(a==16)
    {
        counth+=8;
        countv+=1;
    }
    while(1)
    {
        horizontal(time,counth);
        delay(65535);
        delay(65535);
        vertical(a,time,countv);
        delay(65535);
        delay(65535);
    }
}
```

八. 实验思考题

- (1) 请用户思考一下, 如何改变字体滚动的速度。
- (2) 请用户再思考一下, 如何实现字体的反白显示。

实验十一 555 电路实验

一. 实验目的

掌握 555 集成定时器电路的工作原理和特点, 掌握用 555 集成定时器电路构成单稳态触发器, 多谐振荡器的方法。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
示波器	一台
5.1K 电阻	一只
10K 电阻	二只
105 电容	一只
474 电容	一只
104 电容	二只

三. 实验内容

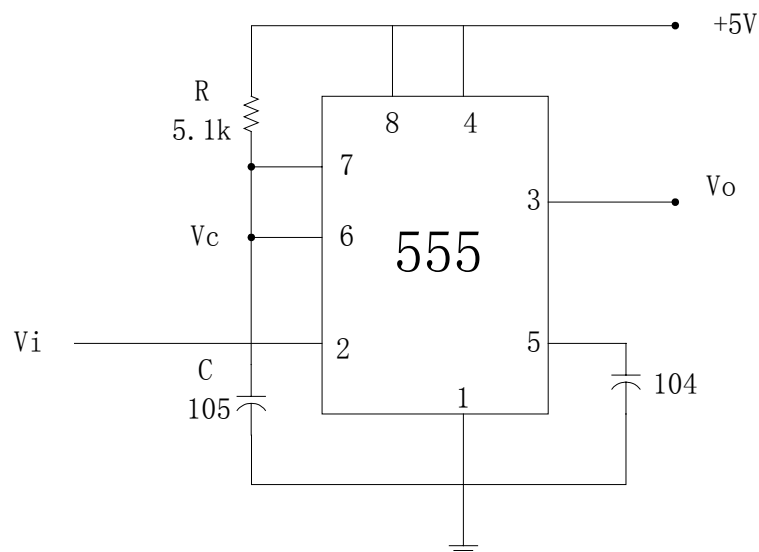
1. 进行单稳态触发器的实验, 计算出输出的脉冲宽度并画出个测量点的波形图。
2. 进行多谐振荡器的实验, 计算出输出信号的频率并画出个测量点的波形图。

四. 实验步骤

1. 用导线把 C6 区的 555 电路按图 3.10 单稳态触发器的接法, 连接好电容电阻, 并短接 JP3。
2. 在 B2 区的 X2 使用 4MHz 或更低的晶振, 短接 B2 区的 JP20, 在 B2 区的时钟源上产生一个小于 1KHz 的信号源输入到图 3.10 中的 Vi 端。
3. 记录下 Vi、Vo、Vc 的波形, 并进行比较分析。
4. 用导线把 C6 区的 555 电路按图 3.11 多谐振荡器的接法, 连接好电容电阻, 并短接 JP3。
5. 记录下 Vo、Vc 的波形, 并进行比较分析, 然后把电容 C 换成 104 再测量分析。

五. 实验电路

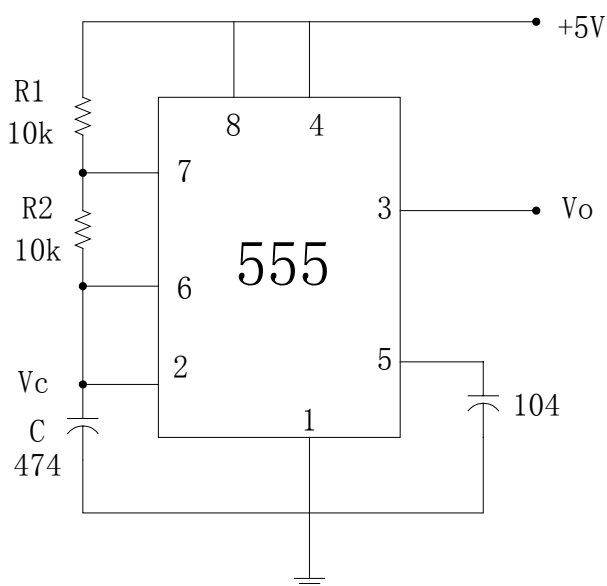
1. 单稳态触发器



V_o 输出的脉冲宽度 $=1.1 \times R \times C$

图 3.10 单稳态触发器电路图

2. 多谐振荡器



$$f_{V_o} = 1 / (0.7 \times (R1 + 2 \times R2) C)$$

图 3.11 多谐振荡器电路图

六. 实验思考题

- (1) 请用户再思考一下，如何搭配图 3.11 的电阻电容才能使多谐振荡器输出 2KHz 的方波。
- (2) 请用户再思考一下，计算出多谐振荡器电路图的输出波形的占空比。

实验十二 运算放大器实验

一. 实验目的

加深理解运算放大器电路的工作原理和特点, 掌握用 LM324 运算放大电路构成基本信号运算电路, 测定它们的运算关系。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
示波器	一台
100K 电阻	二只
10K 电阻	二只
10K 电位器	一只
1M 电阻	一只
104 电容	一只

三. 实验内容

1. 进行反相比例运算放大电路的实验。
2. 进行同相比例运算放大电路的实验。
3. 进行差动运算放大电路的实验。
4. 进行积分运算电路的实验。

四. 实验要求

灵活运用课本中的知识, 掌握 LM324 运算放大集成电路的使用。

五. 实验步骤

1. 用导线把 C4 区的 V- 与 -12V、V+ 与 +12V 相连。
2. 然后使用导线按照图 3.12 反相比例运算放大电路进行连接, 包括连接电阻, 电位器。

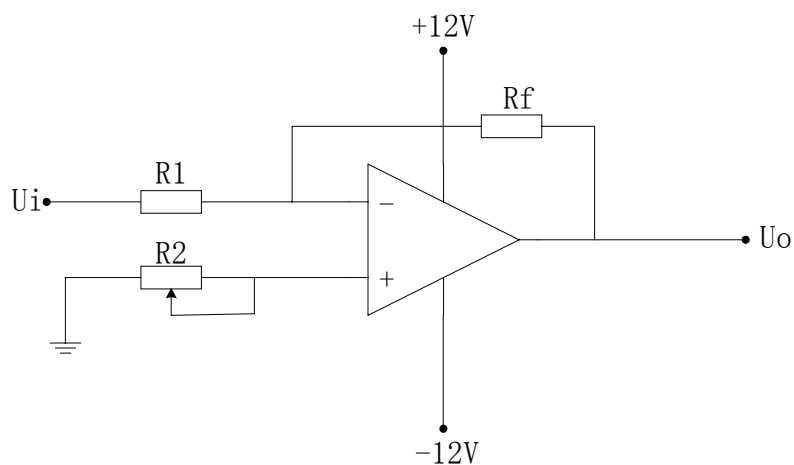


图 3.12 反相比例运算放大电路

电路参数: $R_1=10K$ 、 $R_f=100K$ 、 $R_2=R_1 \parallel R_f \approx 9.1K$

3. 按照下表进行测量和计算

输入电压 U_i (mv)		+100	+200	+300
测量	输入 U_i (mv)			
	输出 U_o (V)			
实测值计算 $A_{uf}=U_o/U_i$				
理论值计算 $A_{uf}=-R_f/R_1$				

4. 使用导线按照图 3.13 同相比例运算放大电路进行连接, 包括连接电阻, 电位器。

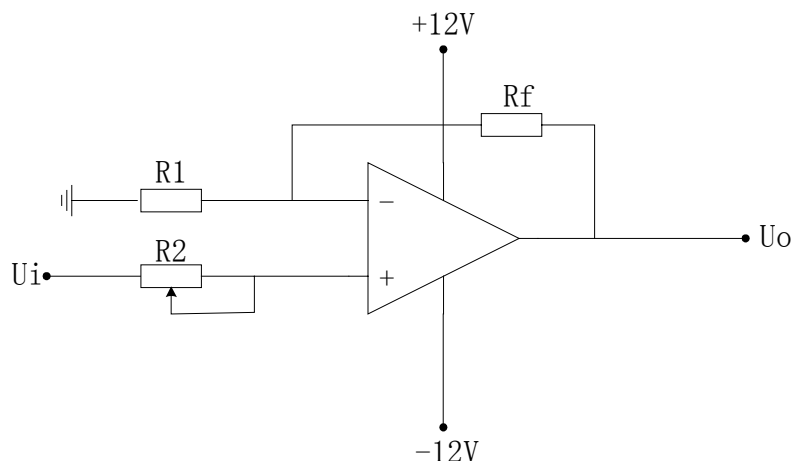


图 3.13 同相比例运算放大电路

电路参数: $R_1=10K$ 、 $R_f=100K$ 、 $R_2=R_1 \parallel R_f \approx 9.1K$

5. 按照下表进行测量和计算

输入电压 U_i (mv)		+100	+200	+300
测量	输入 U_i (mv)			
	输出 U_o (V)			
实测值计算 $A_{uf}=U_o/U_i$				
理论值计算 $A_{uf}=(R_1+R_f)/R_1$				

6. 使用导线按照图 3.14 差动运算放大电路进行连接, 包括连接电阻, 电位器。

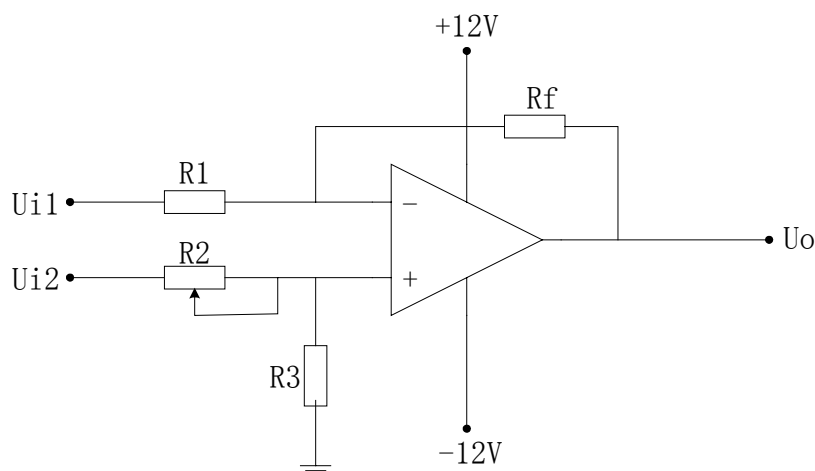


图 3.14 差动运算放大电路

电路参数: $R_1=R_2=10K$ 、 $R_f=R_3=100K$

7. 按照下表进行测量和计算

输入	U_{i1} (mv)	+100	+200	+300
	U_{i2} (mv)	+200	+350	+500
计算 $U_{i2}-U_{i1}$ (mv)				
测量 U_o (V)				
实测值计算 $A_{uf}=U_o/(U_{i2}-U_{i1})$				
理论值计算 $A_{uf}=R_f/R_1$				

8. 使用导线按照图 3.15 积分运算电路进行连接, 包括连接电阻, 电容, 电位器。

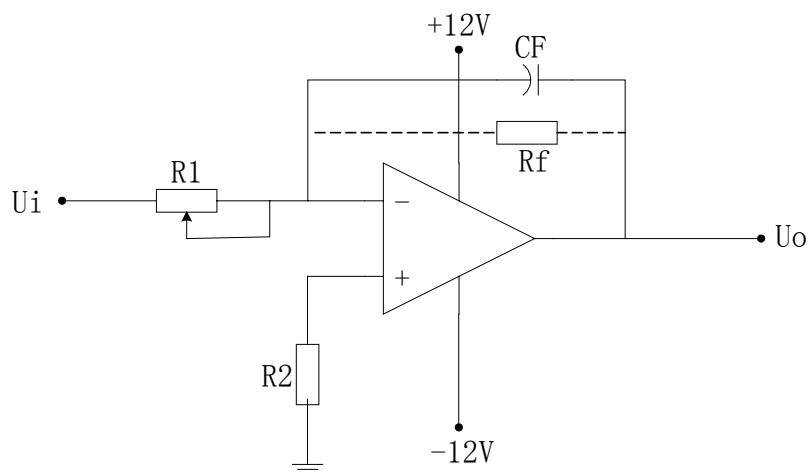


图 3.15 积分运算电路

电路参数: $R_1=R_2=10K$ 、 $CF=0.1\mu F$ 、 $R_f=1M$ (防止自激)

9. 按照下表进行测量和计算

T(秒 S)	0	5	10	20	∞
$U_{i1}=0.5V$, U_o (V)					
理论值计算 $U_o=-tU_i/RC$					
$U_{i1}=-0.5V$, U_o (V)					
理论值计算 $U_o=-tU_i/RC$					

六. 实验思考题

(1) 请用户思考一下, 差动运算放大电路中 $U_{i1}=U_{i2}$ 时, U_o 输出是多少。

实验十三 继电器控制实验

一. 实验目的

加深理解继电器的工作原理和特点，掌握利用单片机的 IO 口控制继电器的一般方法。

二. 实验设备及器件

IBM PC 机 一台
DP-51PROC 单片机综合仿真实验仪 一台

三. 实验内容

1. 利用 D1 区的拨动开关和 LED，学习继电器的工作原理和特点。
2. 编写一段程序，用 P1.0 口控制继电器，继电器控制 LED 的亮和灭，(COM 与 CLOSE 连通时，一盏 LED 亮；不连通时该 LED 灭。COM 与 OPEN 连通时，另一盏 LED 亮，不通时该 LED 灭)。

四. 实验要求

学会继电器的使用和利用单片机的 IO 口控制继电器的方法。

五. 实验步骤

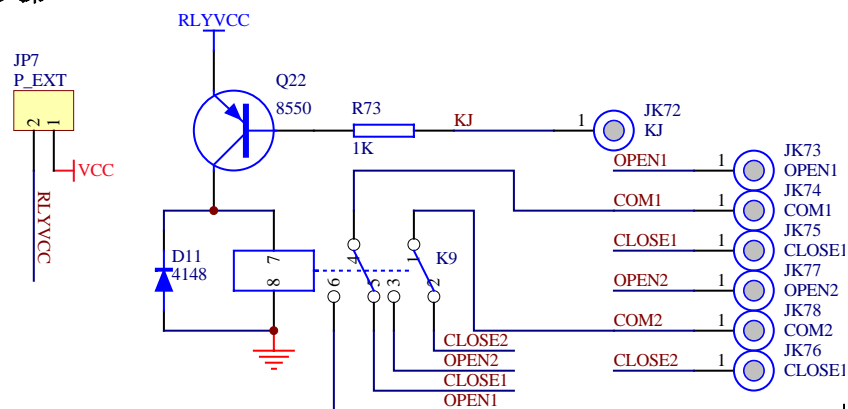


图 3.16 继电器驱动控制电路图

1. 用短路帽短接 JP7，使用导线把 D1 区的 SW1 与 C7 区的 KJ 相连接。
2. 使用导线把 D1 区的 LED1、LED2 与 C7 区的 OPEN1,CLOSE1 分别相连，另外 C7 区的 COM1 接地 (GND)。
3. 接好线后，用户可以拨动 D1 区的 SW1 拨动开关，观察现象（拨到 1 时 LED2 亮，拨到 0 时 LED1 亮），并得出结论。

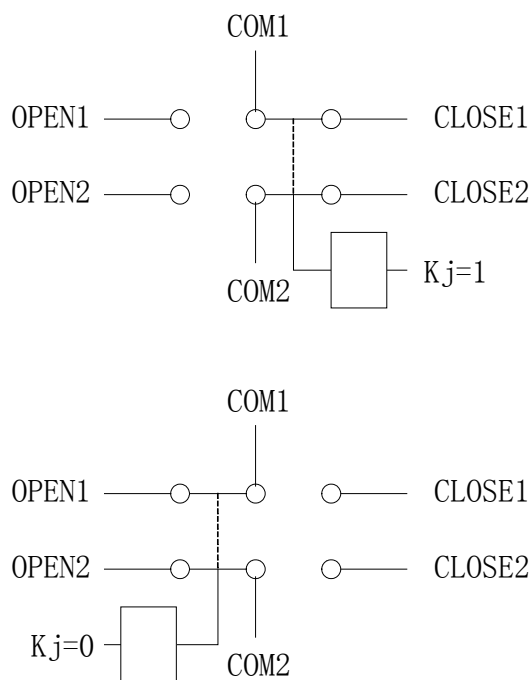


图 3.17 实验 13 原理图

- 然后把 C7 区的 KJ 改接到 A2 区的 P10。再编写一个程序程序（参考实验二），使 P1.0 口延时一段时间后改变电平值，来控制继电器的开关。

六. 实验预习要求

理解继电器驱动控制电路图，还可以先把程序编好，然后在 Keil C51 环境下进行软件仿真。

七. 实验参考程序

```

ORG    8000H
LJMP   Main
ORG    80F0H

Main:
MOV    R7, #0    ;延时
Loop:
MOV    R6, #0
DJNZ   R6, $
DJNZ   R6, $
DJNZ   R6, $
DJNZ   R6, $
DJNZ   R6, $
DJNZ   R7, Loop

CPL    P1.0      ; P 1 .0 取反
SJMP   Main
;
END
    
```

八. 实验思考题

- (1) 请用户思考一下，改由 OPEN2、COM2、CLOSE2 时本实验如何进行。
- (2) 请用户再思考一下，继电器的用途，并举例说明。

PC 机一端，以接收到 0X41（‘A’）为完成，波特率为 9600 Bps。（该程序不能在 DP-51PROC 上进行仿真，所以只能下载，下载的操作可以参考本书的 2.6 节）。

5. 下载程序运行后，先从 PC 机发送一个 0X55（‘A’），这时可以在 PC 的接收软件看见接收到“A”。

六. 实验预习要求

阅读本书的 2.6 节内容，理解硬件结构，还可以先把程序编好，然后在 Keil C51 环境下进行软件仿真。还要学会 PC 机上的串口调试软件的使用（DPFLASH 也内嵌一个串口调试器）。

七. 实验参考程序

程序 1:

```

ORG    0000H
LJMP   Main
ORG    00F0H

Main:
    MOV    SP,#60H           ;给堆栈指针赋初值
    MOV    TMOD,#20H        ;设置 T1 为方式 2
    MOV    TH1,#0FDH        ;设置波特率为 9600
    MOV    TL1,#0FDH
    MOV    SCON,#50H        ;设置串口位方式 1
    MOV    PCON,#00H
    SETB   TR1              ;定时器 1 开始计数

Mainloop:
    MOV    SBUF,#55H        ;开始发送
SENDWT:
    JBC    TI,Mainloop
    AJMP   SENDWT
;
End
    
```

程序 2:

```

ORG    0000H
LJMP   Main
ORG    00F0H

Main:
    MOV    SP,#60H           ;给堆栈指针赋初值
    MOV    TMOD,#20H        ;设置 T1 为方式 2
    MOV    TH1,#0FDH        ;设置波特率为 9600
    MOV    TL1,#0FDH
    MOV    SCON,#50H        ;设置串口位方式 1
    MOV    PCON,#00H
    SETB   TR1              ;定时器 1 开始计数

REC:
    
```



```
JBC    RI,SENDWT
AJMP   REC
SENDWT:
MOV    A,SBUF
CLR    RI
CJNE   A,#55H,REC
MOV    SBUF,#41H      ;开始发送
AJMP   $
;
End
```

八. 实验思考题

- (1) 请用户思考一下，如果是单片机与单片机之间进行串行口通讯应如何进行。
- (2) 请用户再思考一下，第二个程序使用中断的方法如何修改。

实验十五 RS485 差分串行通信实验

一. 实验目的

在上一个 RS232 通讯实验的基础上, 利用单片机的 TXD、RXD 口, 学习 RS485 差分串行接口的使用。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	二台
120 欧姆电阻	二只

三. 实验内容

参考上一个 RS232 通讯实验, 编写一段程序, 利用单片机的串行口发送 0X55。再编写一段程序, 接收 RS485 上传的数据。

四. 实验要求

深刻理解 MAX485 (75176) 芯片的作用, 学会在单片机的串行口上使用 RS485。

五. 实验步骤

1. 使用导线将两台 DP-51PROC 单片机综合仿真实验仪 D4 区的 A_485 与 A_485 相连, B_485 与 B_485 相连。另外在 D4 区的 R74 上插上 120 欧姆电阻, 短接 JP5。

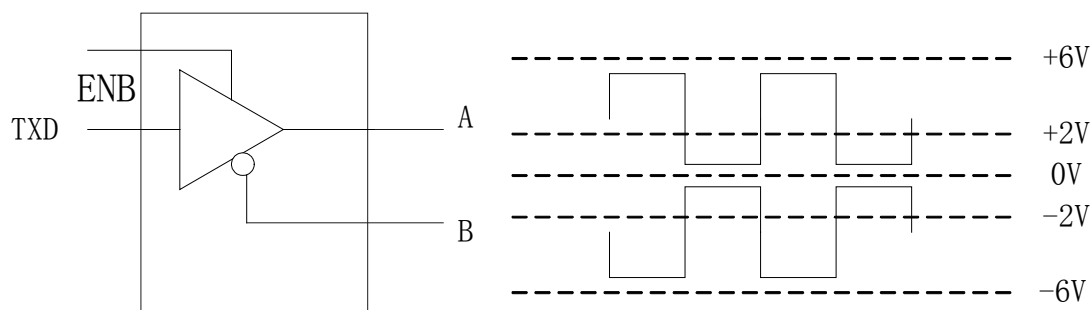


图 3.19 RS485 传输方式

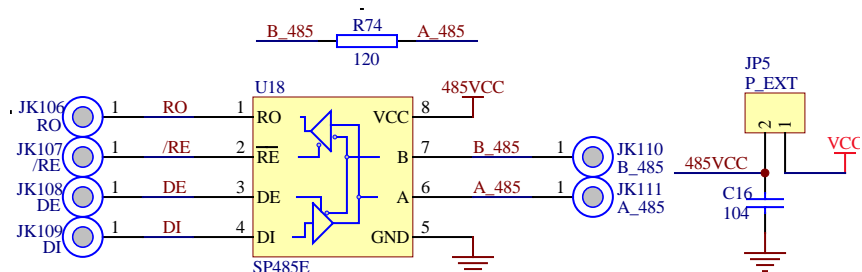


图 3.20 RS485 串行口电路图

2. 使用导线连接 D4 区的 /RE、DE 到 A2 区的 T0、T1, 连接 D4 区的 RO、DI 到 A2 区的 RXD、TXD (RO~RXD、DI~TXD)。
3. 将 D1 区的 J52 接口和 A2 区的 J61 接口一一对应相连。
4. 编写一段程序, 利用单片机的串行口发送 0X55。波特率为 9600 Bps。(该程序不

能在 DP-51PROC 上进行仿真，所以只能下载，下载的操作可以参考本书的 2.6 节)

5. 编写一段程序，利用单片机的串行口接收 RS485 上传的数据。波特率为 9600 Bps。(该程序不能在 DP-51PROC 上进行仿真，所以只能下载，下载的操作可以参考本书的 2.6 节)
6. 两个程序分别在两台机上运行，观察接收的数据和 D1 区 8 个 LED 灯的状态。

六. 实验预习要求

认真阅读本书的 2.8.28 节内容和 MAX485 或 75176 芯片的数据手册，理解硬件结构，还可以先把程序编好，然后在 Keil C51 环境下进行软件仿真。

七. 实验参考程序

发送程序:

```
ORG    0000H
LJMP   Main
ORG    00F0H
```

Main:

```
MOV    SP,#60H           ;给堆栈指针赋初值
MOV    TMOD,#20H         ;设置 T1 为工作方式 2
MOV    TH1,#0FDH         ;设置波特率为 9600
MOV    TL1,#0FDH
MOV    SCON,#50H         ;设置串口位工作方式 1
MOV    PCON,#00H
SETB   TR1               ;定时器 1 开始计数
SETB   P3.4
SETB   P3.5
```

SEND:

```
MOV    SBUF,#55H         ;开始发送
JNB    TI,$
CLR    TI
AJMP   SEND
;
End
```

接收程序:

```
ORG    0000H
LJMP   Main
ORG    00F0H
```

Main:

```
MOV    SP,#60H           ;给堆栈指针赋初值
MOV    TMOD,#20H         ;设置 T1 为工作方式 2
MOV    TH1,#0FDH         ;设置波特率为 9600
MOV    TL1,#0FDH
```

```
MOV    SCON,#50H        ;设置串口位工作方式 1
MOV    PCON,#00H
SETB   TR1              ;定时器 1 开始计数
CLR     P3.4
CLR     P3.5
```

REC:

```
JNB     RI,$
MOV     A,SBUF           ;接收数据
CLR     RI
CPL     A                ;对接收数据取反
MOV     P1,A             ;显示接收到的数据
AJMP    REC
;
End
```

八. 实验思考题

- (1) 请用户思考一下, RS485 通讯如何实现既接收又发送。
- (2) 请用户再思考一下, 如果在各 RS485 节点进行通讯过程中, 正在发送的节点死机了, 会发生什么情况。

实验十六 I²C 总线实验（实时时钟、EEPROM 和 ZLG7290 的实验）

一. 实验目的

加深用户对 I²C 总线的理解，熟悉 I²C 器件的使用，提供用户实际开发的能力。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三. 实验内容

进行 I²C 总线控制的实时时钟、EEPROM、ZLG7290 键盘 LED 控制器实验。

四. 实验要求

熟练掌握 I²C 总线的控制，灵活运用 I²C 主控器软件包，深刻理解实时时钟、EEPROM、ZLG7290 键盘 LED 控制的各种功能。

五. 实验步骤

1. 使用导线连接 D5 区的 SCL、SDA 到 A2 区的 P16、P17 (SCL~P16、SDA~P17)，连接 D5 区的 RST_L、INT_KEY 到 A2 区的 P10、INT0 (RST_L~P10、INT_KEY~INT0)，短接 D5 区的 JP1 跳线。
2. 把模拟 I²C 软件包“VIIC_C51.C”文件加入到 Keil C51 的项目中，程序源文件的开头包含“VIIC_C51.H”头文件。修改 VIIC_C51.C 文件中的 sbit SDA=P1^7; 和 sbit SCL=P1^6;。

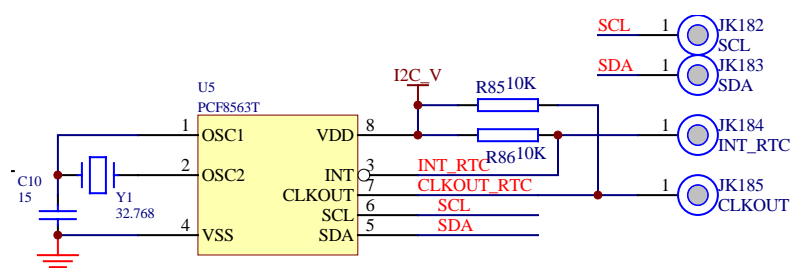


图 3.21 RTC 原理图

3. 使用函数 ISendStr(uchar sla, uchar suba, uchar *s, uchar no) 对 PCF8563T 实时时钟进行设置初始时间，再使用 IRcvStr(uchar sla, uchar suba, uchar *s, uchar no) 对 PCF8563T 实时时钟的时间进行读取。

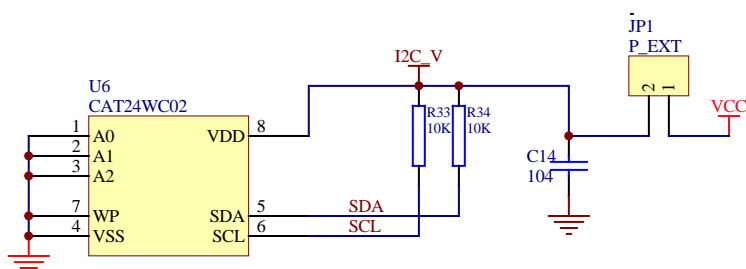


图 3.22 EEPROM 原理图

4. 使用函数 ISendStr(uchar sla, uchar suba, uchar *s, uchar no); 对 24WC02

EEPROM 进行写入，再使用 IRcvStr(uchar sla,uchar suba,uchar *s,uchar no)：对 24WC02 EEPROM 进行读取。

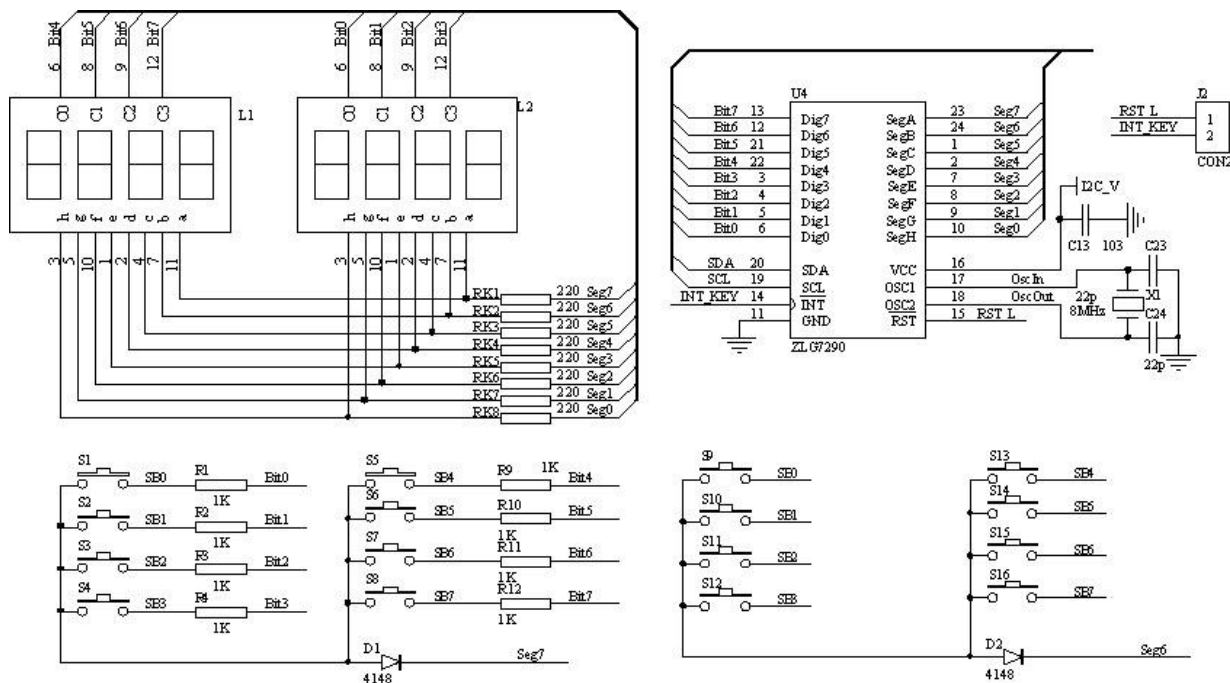


图 3.23 ZLG7290 原理图

5. 对 ZLG7290 键盘 LED 控制器的操作也同理，只是在程序开始的地方增加复位操作和程序中间增加查询是否有键按下。另外还要增加 ZLG7290 软件包“ZLG7290.C”文件加入到 Keil C51 的项目中，程序源文件的开头包含“ZLG7290.H”头文件。

六. 实验预习要求

认真阅读 PCF8563T、24WC02、ZLG7290 的数据手册和模拟 I²C 软件包使用手册，理解硬件结构，还可以先把程序编好，然后在 Keil C51 环境下进行软件仿真。（如果没有在 C 语言环境下进行过调试仿真，可以阅读本书的 2.5 节关于 C 语言环境进行仿真的设置）

七. 实验参考程序

实时时钟程序:

```
#include "reg52.h"
#include "VIIC_C51.H" //包含 VI2C 软件包

#define PCF8563 0xA2 //定义器件地址
#define WRADDR 0x00 //定义写单元首地址
#define RDADDR 0x02 //定义读单元首地址
```

```
unsigned char DelayNS(unsigned char no)
{
    unsigned char i,j; //延时参数
```

```
    for(; no>0; no--)
    {
        for(i=0; i<100; i++)
            for(j=0; j<100; j++);
    }
    return 0;
}

void    main()
{
    unsigned char td[5]={0x00,0x12,0x00,0x30,0x12}; //定义初始化字
    unsigned char rd[5];                             //定义接收缓冲区
    //初始化 PCF8563,如果需要的话可以去掉
    ISendStr(PCF8563,WRADDR,td,0x5);
    DelayNS(10);
    while(1)
    {
        IRcvStr(PCF8563,RDADDR,rd,0x3);    //读现在的时、分、秒
        DelayNS(10);
    }
}
```

EEPROM 程序:

```
#include "REG52.h"
#include "VIIC_C51.h"

#define    CSI24WC02    0XA0
#define    uint8 unsigned char
unsigned char    delay(unsigned char    j)
{
    unsigned char    k,l; //延时参数
    for(l=0;l<=j;l++)
        for(k=0;k<=250;k++);
    return 0;
}

void    main()
{
    /*定义发送缓冲区*/
    uint8 td[5]={0x00,0x12,0x55,0x30,0x12};
    uint8 rd[5];                             /*定义接收缓冲区*/
    ISendStr(CSI24WC02,0,td,0x5);            /*发送数据*/
    delay(100);
    while(1)
```

```
    {
        IRcvStr(CSI24WC02,0,rd,0x5);    /*读取数据*/
        delay(100);
        while(1);
    }
}
```

ZLG7290 程序:

```
#include "reg52.h"
#include "VIIC_C51.H"           //包含 VI2C 软件包
#include "zlg7290.h"
```

```
sbit RST=P1^0;
sbit KEY_INT=P3^2;
```

```
unsigned char DelayNS(unsigned char no)
{
    unsigned char i,j;           //延时参数
    for(; no>0; no--)
    {
        for(i=0; i<100; i++)
            for(j=0; j<10; j++);
    }
    return 0;
}
```

```
void main()
{
    unsigned char i,KEY;
    RST=0;                       //zlg7290 复位
    DelayNS(1);
    RST=1;
    DelayNS(10);
    while(1)
    {
        if(KEY_INT==0)
        {
            KEY=ZLG7290_GetKey();    //读键
            DelayNS(10);
            for(i=0; i<8; i++)
            {
                ZLG7290_SendCmd(0x60+i,KEY); //显示键值
                DelayNS(1);
            }
        }
    }
}
```



```
    }  
  }  
}
```

八. 实验思考题

(1) 请用户再思考一下, I²C 的从设备控制程序应该如何编写。

实验十七 万年历时钟实验

一. 实验目的

进行一次实际开发的实验，提高用户实际开发的能力。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三. 实验内容

结合 I²C 总线控制的实时时钟、ZLG7290 键盘 LED 控制器进行万年历时钟的设计。

四. 实验要求

熟练掌握 I²C 总线的控制，灵活运用 I²C 主控器软件包，深刻理解实时时钟、ZLG7290 键盘 LED 控制的各种功能，并能把它们相结合，组成具有实际功能的万年历时钟。

五. 实验步骤

1. 用导线连接 D5 区的 SCL、SDA 到 A2 区的 P16、P17 (SCL~P16、SDA~P17)，连接 D5 区的 RST_L、INT_KEY 到 A2 区的 P10、INT0 (/RST~P10、INT_KEY~INT0)，短接 D5 区的 JP1 跳线。
2. 模拟 I²C 软件包“VIIC_C51.C”文件加入到 Keil C51 的项目中，开头包含“VIIC_C51.H”头文件。修改 VIIC_C51.C 文件中的 sbit SDA=P1^7; 和 sbit SCL=P1^6;。另外还要增加 ZLG7290 软件包“ZLG7290.C”文件加入到 Keil C51 的项目中，开头包含“ZLG7290.H”头文件。
3. 先编写显示的子函数，可以分别显示日期和显示时间。用按键来切换显示的内容(初步设计用 D5 区的 S1 按键来切换)。
4. 然后再进行整体编程，如果还有困难可以先画流程图，再编写程序。
5. 显示结果为：上电运行时，8 位数码管显示时间，时分秒；按一下 S1 切换到显示年月日，再按一下 S1 又切换回时分秒。

六. 实验预习要求

认真阅读 PCF8563T、ZLG7290 的数据手册和模拟 I²C 软件包使用手册，理解硬件结构，还可以先把程序编好，然后在 Keil C51 环境下进行软件仿真。

七. 实验参考程序

实时时钟程序：

```
#include      "reg52.h"
#include      "VIIC_C51.H"      //包含 VI2C 软件包
#include      "zlg7290.h"
```

```

#define PCF8563    0xA2        //定义器件地址
#define WRADDR     0x00        //定义写单元首地址
#define RDADDR     0x02        //定义读单元首地址

sbit KEY_INT=P3^2;
sbit RST=P1^0;
unsigned char disp_buf[8]={0,0,0,0,0,0,0,0}; //显示缓存
unsigned char KEY;                //保存键值
bit swich_date=0;
unsigned char display_time(unsigned char *sd)
{

    sd[0]=sd[0]&0x7f; //秒屏蔽保留位
    sd[1]=sd[1]&0x7f; //分屏蔽保留位
    sd[2]=sd[2]&0x3f; //时屏蔽保留位
    disp_buf[0] =(sd[0]%16);
    disp_buf[1] =(sd[0]/16);
    disp_buf[2] = 31;
    disp_buf[3] =(sd[1]%16);
    disp_buf[4] =(sd[1]/16);
    disp_buf[5] = 31;
    disp_buf[6] =(sd[2]%16);
    disp_buf[7] =(sd[2]/16);
    ZLG7290_SendBuf(disp_buf,8);
    return 0;
}
unsigned char display_date(unsigned char *sd)
{
    sd[0]=sd[0]&0x3f; //日屏蔽保留位
    sd[2]=sd[2]&0x1f; //月屏蔽保留位
    disp_buf[0] =(sd[0]%16);
    disp_buf[1] =(sd[0]/16);
    disp_buf[2] =(sd[2]%16)+0x80; //后面加小数点
    disp_buf[3] =(sd[2]/16);
    disp_buf[4] =(sd[3]%16)+0x80; //后面加小数点
    disp_buf[5] =(sd[3]/16);
    disp_buf[6] =0;
    disp_buf[7] =2;
    ZLG7290_SendBuf(disp_buf,8);
    return 0;
}
unsigned char DelayNS(unsigned char no)
{
    unsigned char i,j;                //延时参数

```

```

        for(; no>0; no--)
            for(i=0; i<100; i++)
                for(j=0; j<100; j++);
        return 0;
    }
    void    main()
    {
        unsigned char code td[9]={ 0x00,0x12,0x00,0x30,0x12,
                                     0x06,0x05,0x02,0x04}; //定义初始化字
        unsigned char rd[7]; //定义接收缓冲区
        RST=0;
        DelayNS(1);
        RST=1;
        //初始化 PCF8563,如果需要的话可以去掉
        ISendStr(PCF8563,WRADDR,td,0x5);
        DelayNS(1);
        //初始化 PCF8563,如果需要的话可以去掉
        ISendStr(PCF8563,WRADDR+5,&td[5],0x4);
        while(1)
        {
            DelayNS(1);
            IRcvStr(PCF8563,RDADDR,rd,0x7); //读现在的时、分、秒
            DelayNS(1);
            if(swich_date)
                display_date(rd+3); //调显示日期子程序
            else
                display_time(rd); //调显示时间子程序
            if(KEY_INT==0)
            {
                KEY=ZLG7290_GetKey();
                if(KEY==1)
                    swich_date=~swich_date;
            }
        }
    }
}

```

八. 实验思考题

- (1) 请用户思考一下, 如果要断电还能不丢失时间应该做那些措施。
- (2) 请用户再思考一下, 如果要降低功耗, 软件设计时应做那些措施。

实验十八 接触式 IC 卡读写实验

一. 实验目的

了解接触式 IC 卡的知识，学会如何根据时序逻辑图编写实用程序。

二. 实验设备及器件

- IBM PC 机 一台
- DP-51PROC 单片机综合仿真实验仪 一台

三. 实验内容

根据 IC 卡的读写时序图编写程序，实现 IC 卡(SLE4442)的读写。

四. 实验要求

根据 IC 卡的时序图编写 51 单片机程序读写逻辑加密 IC 卡(SLE4442)读写程序。

五、实验线路与实验原理

接触式 IC 卡的触点定义遵循 ISO7816 规定，IC 卡 8 个触点分布位置如图 3.23 所示，对应着 DP-51PROC 单片机综合仿真实验仪“D8IC 卡”区 IC 卡座上方的引线。本实验使用的是 SLE4442 卡，SLE4442 的触点安排见图 3.24。

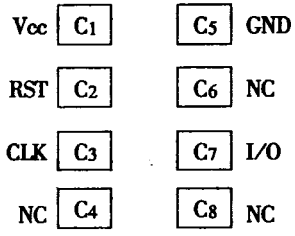
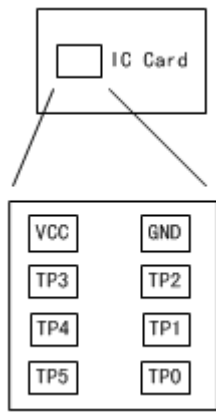


图 3.24 IC 卡触点分布图

图 3.25 SLE4442 的触点安排

本实验 SLE4442 卡与单片机的引脚连线关系见表 3.4。

表 3.4 SLE4442 与单片机引脚连线

单片机	实验仪板上对应引线	SLE4442 引脚
P1.0	TP1	I/O
P1.1	TP4	CLK
P1.2	TP3	RST

下面介绍 SLE4442 的有关知识

5.1 概述

SLE4442 是德国西门子(SIMENS)公司设计的逻辑加密存储卡。它具有 2K 位的存储容量和完全独立的可编程代码存储器(PSC)。内部电压提升电路保证了芯片能够以单+5V 电压供电，较大的存储容量能够满足通常应用领域的各种需要。因此是目前国内应用较多的

一种 IC 卡芯片。芯片采用多存储器结构, 2 线连接协议(串行接口满足 ISO7816 同步传送协议), NMOS 工艺技术, 每字节的擦除/写入编程时间为 2.5ms。存储器具有至少 10000 次的擦写周期, 数据保持时间至少 10 年。

SLE4442 IC 卡主要包括三个存储器 :

(1) 256x8 位 EEPROM 型主存储器。地址 0~31 为保护数据区, 该区数据读出不受限制, 写入受保护存储内部数据状态的限制。当保护存储器中第 N 位 (N=0~31) 为 1 时, 对应主存储器中第 N 个字节允许进行擦除和写入操作。地址 32~255 后 223 字节为应用数据区, 数据读出不受限制, 擦除和写入受加密存储器数据校验结果的影响。这种加密校验的控制是对整个主存储器实施的(即包括保护数据区和应用数据区)。

(2) 32 x1 位 PROM 型保护存储器。一次性编程以保护主存储器保护数据区, 防止一些固定的标识参数被改动。保护存储器同样受加密存储器数据校验结果的影响。

(3) 4x8 位 EEPROM 型加密存储器。第 0 字节为密码输入错误计数器 (EC)。EC 的有效位 是低三位, 芯片初始化时设置成 “111”。这一字节是可读的。EC 的 1, 2, 3 字节为参照字存储区。这 3 个字节的内容作为一个整体被称为可编程加密代码(PSC)。其读出, 写入和擦除均受自身 “比较” 操作结果的控制。

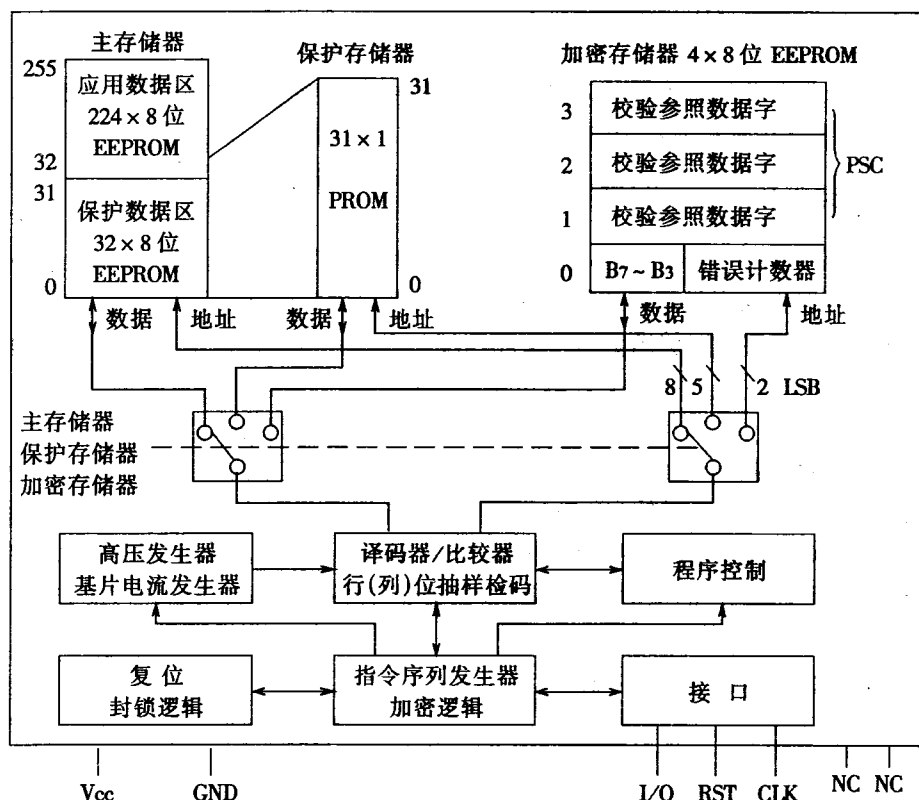


图 3.26 SLE4442 卡的内部结构图

5.2 传送协议

(1) 复位和复位响应

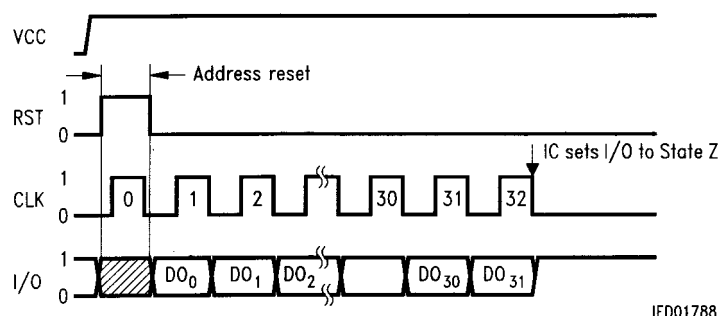


图 3.27 复位时序图

复位和复位响应是根据 ISO7816-3 标准来进行的。在操作期间的任意时候都可以复位。开始，地址计数器随一个时钟脉冲而被设置为零。当 RST 线从高状态(H)置到低状态(L)时,第一个数据位(LSB)的内容被送到 I/O 上。若连续输入 32 个时钟脉冲,主存储器中的前四个字节单元中的内容被读出。在第 33 个脉冲的下降沿, I/O 线被置成高状态而关闭。

(2)命令模式

复位响应以后,芯片等待着命令。每条命令都以一个“启动状态”开始。整个命令包括 3 个字节。随后紧跟着一个附加脉冲并用一个“停止状态”来结束操作。

启动状态: 在 CLK 为高状态 (H 状态) 期间, I/O 为下降沿时为启动状态。

停止状态: 在 CLK 为高状态 (H 状态) 期间, I/O 为上升沿时为停止状态。

在接受一个命令之后,有两种可能的模式:输出数据模式(即读数据)和处理数据模式。

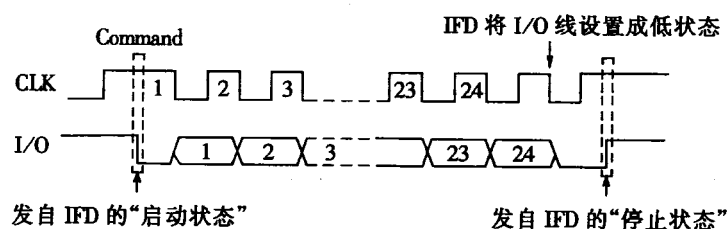


图 3.28 命令模式的时序图

(3) 输出数据模式

这种模式是将 IC 卡芯片中的数据传送给外部设备接口(IFD)的一种操作。

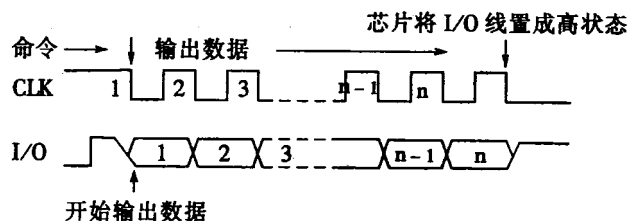


图 3.29 输出数据模式的时序图

在第一个 CLK 脉冲的下降沿之后, I/O 线上的第一位数据变为有效。随后每增加一个时钟脉冲,芯片内部的一位数据被送到 I/O 线上。数据的发送从每个字节的最低位 (LSB) 开始。当所需要的最后一个数据送出以后,需要在附加一个时钟脉冲来把 I/O 置成高状态,以便接受新的命令。

在输出数据期间,任何“启动状态”和“停止状态”均被屏蔽掉。

(4) 处理数据模式

这种模式是对 IC 芯片作内部处理。

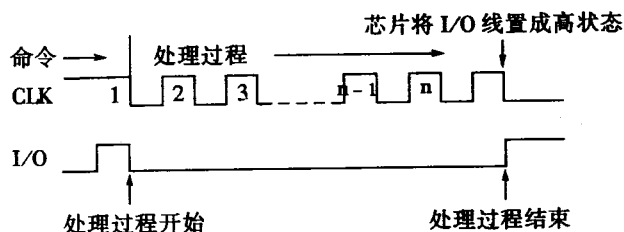


图 3.30 处理模式的时序图

芯片在第一个脉冲的下降沿将 I/O 线从高状态拉到低状态并开始处理。此后芯片在内部连续计时计数，直到低 n 个时钟脉冲之后的附加一个时钟脉冲的下降沿 I/O 线再次置高，完成芯片的处理过程。在整个处理过程中 I/O 线被锁定成低状态。

5.3 SLE4442 卡的应用

1. 芯片的复位方式

(1) 外部复位：SLE4442 是基于同步复位响应的传送协议。芯片的复位时序如前述。

(2) 加电复位：在把操作电压连接到 V_{cc} 段之后，芯片内部进行复位操作。I/O 线被置为高状态。必须在对任意地址进行读操作或做一个复位响应操作之后才可以进行数据交换。

(3) 中止：在 CLK 为低状态期间，如果 RST 置为高状态，则任何操作均无效。I/O 线被锁定到高状态。需要一个最小维持时间 $t_{res}=5\mu s$ 之后，芯片才能接受新的复位，中止状态的时序关系如图 3.31。中止状态之后，芯片又准备下一个操作。

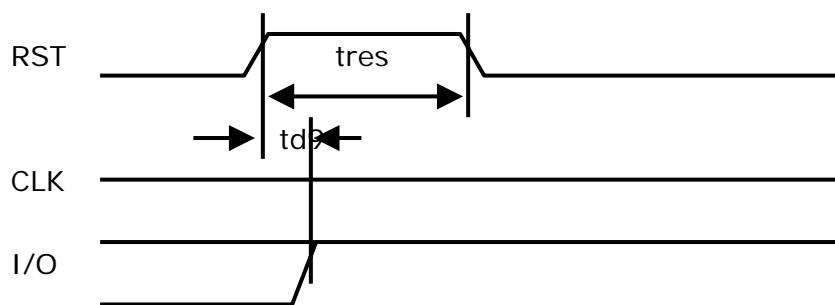


图 3.31 中止时序图

芯片的操作命令

命令格式：

(1) 每条命令包含三个字节，其排列顺序如下：

表 3.5 命令格式

MSB	控制字							LSB	MSB	地址字							LSB	MSB	数据字							LSB
B7	B6	B5	B4	B3	B2	B1	B0		A7	A6	A5	A4	A3	A2	A1	A0		D7	D6	D5	D4	D3	D2	D1	D0	

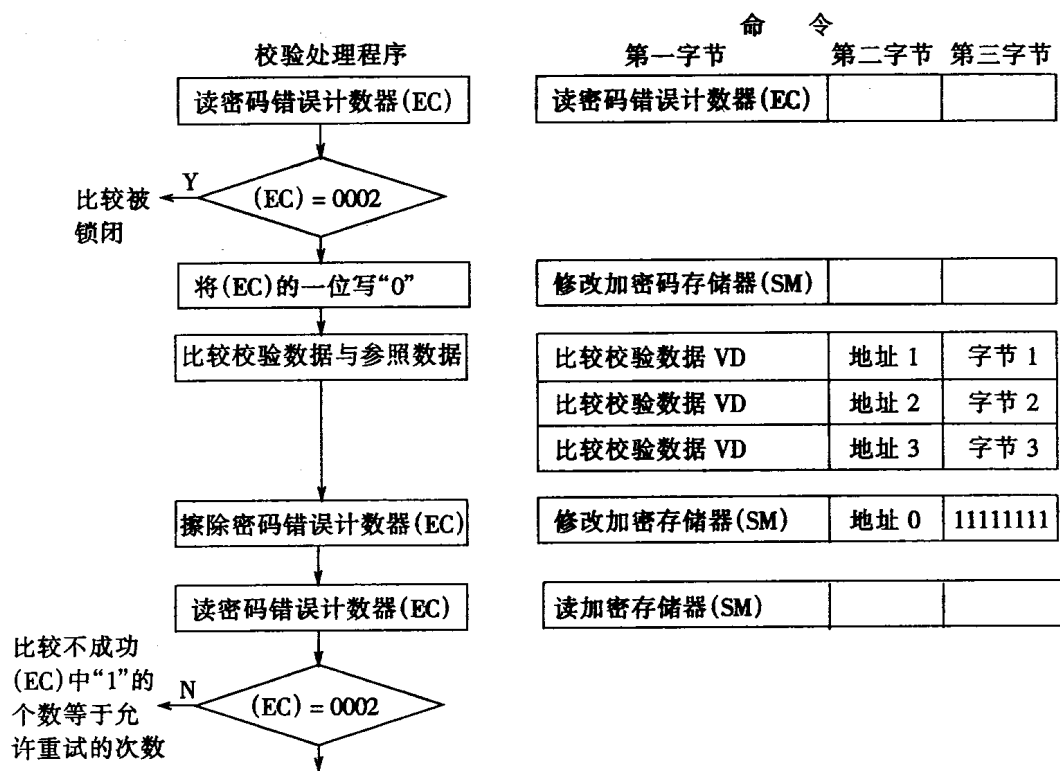
(2) SLE4442 芯片具有七种命令，其格式和功能见表 3.6。

表 3.6

字节 1 (控制) B7~B0	字节 2 (地址) A7~A0	字节 3 (数据)	功能	命令模式
30H	地址数	无效	读主存储器	输出数据模式
38H	地址数	输入数据	修改主存储器	处理模式
34H	无效	无效	读保护存储器	输出数据模式
3CH	地址数	输入数据	写保护存储器	处理模式
31H	无效	无效	读加密存储器	输出数据模式
39H	地址数	输入数据	修改加密存储器	处理模式
33H	地址数	输入数据	比较校验数据	处理模式

注意：每个字节来说总是从最低 LSB 开始读出。写入时首先传送的也是字节的最低为 (LSB)。对保护存储器进行修改时，输入数据必须与原有数据相等，才能正确保护。

比较校验数据流程如下。



比较校验数据的程序流程

图 3.32 比较校验数据的程序流程图

六 实验步骤

1. 在 DP-51PROC 单片机综合仿真实验仪上按表 3.4(SLE4442 与单片机引脚连线) 在 A2 区和 D6 区之间连接好 IC 卡与单片机之间的连线。
2. 将 SLE4442 卡触点朝下插入 IC 卡座中。
3. 运行 " SLE4442 实验程序"(实验程序一)。
4. 运行 C51 开发环境观察程序运行结果。

实验程序一、从主存储区的 0 地址读 8 个字节

```

ORG    8000H
AJMP   MAIN
ORG    8100H
MAIN:
    MOV    SP,#60H           ;设置栈底
Read_Insert_Card:
    MOV    ByteNum, #08H
    MOV    StartAdr, #00H    ;从主存储区的 0 地址读 8 个字节
    LCALL  ReadCard
    JMP    Read_Insert_Card
$INCLUDE(SLE4442.INC)        ;包含 SLE4442 驱动程序
;
END

```

在 JMP Read_Insert_Card 处设置断点，观察 RecBuf 的内容，RecBuf 的定义见 SLE4442.INC 文件。LE4442.INC 的源码及注释请见 SLE4442. INC 文件。

实验程序二、从 0x20 地址开始写入 2 个数据 0x55 和 0x66，再读出来，由于 SLE4442 写主存储器时需要验证密码，所以执行下面的程序时请确保你 SLE4442 卡的密码的正确性，如果密钥错误，你的 SLE4442 校验三次便会锁死报废。

```

ORG    8000H
AJMP   MAIN
ORG    8100H
MAIN:
    MOV    SP,#60H           ;设置栈底
Write_Read_Card:
    MOV    R0,#PSWD          ;密码缓冲区首址
    MOV    @R0,#0x11
    INC    R0
    MOV    @R0,#0x22
    INC    R0
    MOV    @R0,#0x33          ;输入密码:填充密码缓冲区
    LCALL  CheckPassword      ;校验密码
    JNZ    WRExit             ;如果校验不成功,退出
                                ;密码校验成功,进入写卡程序
    MOV    R0,#WriteBuf       ;WriteBuf 为写卡缓冲区首址
    MOV    @R0,#0x55
    INC    R0;
    MOV    @R0,#0x66          ;填充写缓冲区
    MOV    ByteNum,#0x02
    MOV    StartAdr,#0x20
    LCALL  WriteCard           ;调用写卡子程序
    JNZ    WRExit             ;如果写不成功,退出
    MOV    ByteNum,#0x02      ;下面读出刚刚写入的数据

```

```
MOV    StartAdr,#0x20
LCALL  ReadCard
```

;如果读成功，才可以执行这个循环，在此设置断点，

```
JZ      Write_Read_Card
```

;观察读缓冲区内容

WRExit:

```
MOV     A,#00H                ;在此设置断点
```

```
$INCLUDE(SLE4442.INC)         ;包含 SLE4442 驱动程序
```

```
;
```

```
END
```

请在 JZ Write_Read_Card 语句处设置断点，在读成功时可以观察读缓冲区的数据。在 MOV A, #00H 语句处设置断点，这样当发生错误时程序不乱跑。

七、实验思考题

- (1) 如果单片机的晶振频率选用 24MHz，程序应该做那些修改，才能正确读写 SLE4442 卡。
- (2) 编程序，根据 SLE4442 的时序图用 C 语言编写读写程序。

实验十九 数字温度传感器实验

一. 实验目的

熟悉数字温度传感器 DS18B20 的使用方法和工作原理，了解单总线的读写控制方法。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
温度计	一个

三. 实验内容

1. 编写程序，通过单片机的 P3.3 口对 DS18B20 进行操作，实现数字温度的采集。
2. 记录采集到的温度数据，分析实验结果是否正确。

四. 实验要求

熟悉单总线方式的工作原理及应用，初步了解温度传感器的使用技巧。

五. 实验步骤

1. 安装 B4 区 JP12 接口上的短路帽，将 B4 区的 DQ 针与 A2 区的 INT1(P3.3)相连。
2. 运行编写好的软件程序，完成多次温度采集并记录采集到的温度数据。
3. 使用温度计测量环境的实际温度与实验数据相比较，判断采集数据的准确度。

六. 实验预习要求

结合本实验的温度采集程序设计基础，认真阅读本书 2.8.10 节的实验内容和 DS18B20 的数据手册，完成温度采集显示的综合实验。

七. 实验参考程序

```

    TEMPER_L    EQU    36H    ;存放读出温度低位数据
    TEMPER_H    EQU    35H    ;存放读出温度高位数据
    TEMPER_NUM  EQU    60H    ;存放转换后的温度值
    FLAG1       BIT    00H
    DQ          BIT    P3.3   ;一线总线控制端口

    ORG    8000H
    LJMP    MAIN
    ORG    8100H

MAIN:
    MOV     SP,#70H
    LCALL  GET_TEMPER          ;从 DS18B20 读出温度数据
    LCALL  TEMPER_COV         ;转换读出的温度数据并保存
    SJMP   $                  ;完成一次数字温度采集

;以下为 DS18B20 单总线操作子程序

```

```

;读出转换后的温度值
GET_TEMPER:
    SETB    DQ                                ;定时入口
BCD:
    LCALL   INIT_1820
    JB      FLAG1,S22
    LJMP    BCD                                ;若 DS18B20 不存在则返回
S22:
    LCALL   DELAY1
    MOV     A,#0CCH                            ;跳过 ROM 匹配 OCCH
    LCALL   WRITE_1820
    MOV     A,#44H                            ;发出温度转换命令
    LCALL   WRITE_1820
    NOP
    LCALL   DELAY
    LCALL   DELAY
CBA:
    LCALL   INIT_1820
    JB      FLAG1,ABC
    LJMP    CBA
ABC:
    LCALL   DELAY1
    MOV     A,#0CCH                            ;跳过 ROM 匹配
    LCALL   WRITE_1820
    MOV     A,#0BEH                            ;发出读温度命令
    LCALL   WRITE_1820
    LCALL   READ_18200                          ;READ_1820
    RET
;读 DS18B20 的程序,从 DS18B20 中读出一个字节的数据
READ_1820:
    MOV     R2,#8                                ;读取 8 个位, 一个字节
RE1:
    CLR     C
    SETB    DQ
    NOP
    NOP
    CLR     DQ
    NOP
    NOP
    NOP
    SETB    DQ
    MOV     R3,#7                                ;延时
    DJNZ    R3,$
    MOV     C,DQ                                ;读取一个位

```

```

        MOV    R3,#23                ;延时
        DJNZ   R3,$
        RRC    A
        DJNZ   R2,RE1
        RET
;写 DS18B20 的程序
WRITE_1820:
        MOV    R2,#8                ;读取 8 个位，一个字节
        CLR    C
WR1:
        CLR    DQ
        MOV    R3,#6                ;延时
        DJNZ   R3,$
        RRC    A
        MOV    DQ,C                ;发送一个位
        MOV    R3,#23                ;延时
        DJNZ   R3,$
        SETB   DQ
        NOP
        DJNZ   R2,WR1
        SETB   DQ
        RET
;读 DS18B20 的程序,从 DS18B20 中读出两个字节的温度数据
READ_18200:
        MOV    R4,#2                ;将温度高低位从 DS18B20 中读出
        MOV    R1,#36H
;低位存入 36H(TEMPER_L),高位存入 35H(TEMPER_H)
RE00:
        MOV    R2,#8                ;读取 8 个位，一个字节
RE01:
        CLR    C
        SETB   DQ
        NOP
        NOP
        CLR    DQ
        NOP
        NOP
        NOP
        SETB   DQ
        MOV    R3,#7                ;延时
        DJNZ   R3,$
        MOV    C,DQ                ;读取一个位
        MOV    R3,#23                ;延时
        DJNZ   R3,$

```

```

RRC    A
DJNZ   R2,RE01
MOV     @R1,A
DEC     R1
DJNZ   R4,RE00
RET

```

;将从 DS18B20 中读出的温度数据进行转换

TEMPER_COV:

```

MOV     A,#0F0H
ANL     A,TEMPER_L           ;舍去温度低位中小数点后的四位温度

```

数值

```

SWAP    A
MOV     TEMPER_NUM,A
MOV     A,TEMPER_L
JNB     ACC.3,TEMPER_COV1   ;四舍五入温度值
INC     TEMPER_NUM

```

TEMPER_COV1:

```

MOV     A,TEMPER_H
ANL     A,#07H
SWAP    A
ADD     A,TEMPER_NUM
MOV     TEMPER_NUM,A        ;保存变换后的温度数据
LCALL   BIN_BCD
RET

```

;将 16 进制的温度数据转换成压缩 BCD 码

BIN_BCD:

```

MOV     DPTR,#TEMP_TAB
MOV     A,TEMPER_NUM
MOVC    A,@A+DPTR
MOV     TEMPER_NUM,A
RET

```

TEMP_TAB:

```

DB 00H,01H,02H,03H,04H,05H,06H,07H
DB 08H,09H,10H,11H,12H,13H,14H,15H
DB 16H,17H,18H,19H,20H,21H,22H,23H
DB 24H,25H,26H,27H,28H,29H,30H,31H
DB 32H,33H,34H,35H,36H,37H,38H,39H
DB 40H,41H,42H,43H,44H,45H,46H,47H
DB 48H,49H,50H,51H,52H,53H,54H,55H
DB 56H,57H,58H,59H,60H,61H,62H,63H
DB 64H,65H,66H,67H,68H,69H,70H,71H
DB 72H,73H,74H,75H,76H,77H,78H,79H
DB 80H,81H,82H,83H,84H,85H,86H,87H
DB 88H,89H,90H,91H,92H,93H,94H,95H

```

DB 96H,97H,98H,99H

;DS18B20 初始化程序

INIT_1820:

```
SETB  DQ
NOP
CLR   DQ
MOV   R0,#80H
```

TSR1:

```
DJNZ  R0,TSR1      ;延时
SETB  DQ
MOV   R0,#25H      ;96US
```

TSR2:

```
DJNZ  R0,TSR2
JNB   DQ,TSR3
LJMP  TSR4          ;延时
```

TSR3:

```
SETB  FLAG1        ;置标志位,表示 DS1820 存在
LJMP  TSR5
```

TSR4:

```
CLR   FLAG1        ;清标志位,表示 DS1820 不存在
LJMP  TSR7
```

TSR5:

```
MOV   R0,#06BH     ;200US
```

TSR6:

```
DJNZ  R0,TSR6      ;延时
```

TSR7:

```
SETB  DQ
RET
```

;重新写 DS18B20 暂存存储器设定值

RE_CONFIG:

```
JB    FLAG1,RE_CONFIG1 ;若 DS18B20 存在,转 RE_CONFIG1
RET
```

RE_CONFIG1:

```
MOV   A,#0CCH      ;发 SKIP ROM 命令
LCALL WRITE_1820
MOV   A,#4EH       ;发写暂存存储器命令
LCALL WRITE_1820
MOV   A,#00H       ;TH(报警上限)中写入 00H
LCALL WRITE_1820
MOV   A,#00H       ;TL(报警下限)中写入 00H
LCALL WRITE_1820
MOV   A,#7FH       ;选择 12 位温度分辨率
```



```
        LCALL  WRITE_1820
        RET
;延时子程序
DELAY:
        MOV    R7,#00H
MIN:
        DJNZ   R7,YS500
        RET
YS500:
        LCALL  YS500US
        LJMP   MIN
YS500US:
        MOV    R6,#00H
        DJNZ   R6,$
        RET
DELAY1:
        MOV    R7,#20H
        DJNZ   R7,$
        RET
;
END
```

八. 实验思考题

设计一个单总线工作方式下，采用多个 DS18B20 实现多路数字温度采集的实验程序，完成多个点的温度采集。

实验二十 单总线和 I²C 总线结合实现数字温度计实验

一. 实验目的

通过本实验,理解掌握单总线器件和 I²C 总线器件的应用,熟悉串行总线的操作技巧。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三. 实验内容

1. 编写程序,通过单片机的 P3.3 口控制一个 DS18B20 完成数字温度的采集,然后用程序处理采集到的数据结果。
2. 编写程序,通过 I²C 总线器件 ZLG7290 实现温度数据的输出显示。
3. 结合以上两部分程序,编程实现数字式温度计的程序设计。

四. 实验要求

熟练掌握单总线方式器件的应用,熟悉 I²C 总线协议,学习 I²C 器件的使用方法。

五. 实验步骤

1. 安装 B4 区 JP12 接口上的短路帽,将 B4 区的 DQ 与 A2 区 INT1(P3.3)相连。
2. 安装 D5 区 JP1 接口上的短路帽,将 D5 区的 SDA、SCL 分别与 A2 区的 P17、P16 相连。
3. 将 D5 区的 RST_L 针接上高电平。
4. 运行编写好的软件程序,观察 D5 区数码管显示的温度数据。然后改变 DS18B20 的表面温度,查看显示的温度数据是否有变化,并调整实验程序使测量值更准确。

六. 实验预习要求

认真学习本书 2.8.10 节的实验内容和实验原理,做好实验前的准备工作。

七. 实验参考程序

TEMPER_L	EQU	36H	;存放读出温度低位数据
TEMPER_H	EQU	35H	;存放读出温度高位数据
TEMPER_NUM	EQU	37H	;存放转换后的温度值
FLAG1	BIT	00H	
DQ	BIT	P3.3	;单总线控制端口
SDA	BIT	P1.7	;I ² C 总线定义
SCL	BIT	P1.6	
MTD	EQU	40H	;发送数据缓冲器
MRD	EQU	49H	;接收数据缓冲区
;定义器件地址,变量			
ZLG7290	EQU	70H	;ZLG7290 的器件地址

```

ACK          BIT    10H    ;应答标志位
SLA          DATA  50H    ;器件的从地址
SUBA         DATA  51H    ;器件的子地址
NUMBYTE      DATA  52H    ;读/写的字节数变量

```

```
ORG 8000H
```

```
LJMP MAIN
```

```
ORG 8100H
```

;温度计主程序如下:

MAIN:

```
MOV SP,#70H
```

DISP_LOOP:

```
LCALL GET_TEMPER          ;从 DS18B20 读出温度数据
```

```
LCALL TEMPER_COV          ;转换读出的温度数据并保存
```

```
LCALL DELAY
```

```
MOV MTD,#60H
```

```
MOV MTD+1,TEMPER_NUM      ;温度值低位
```

```
ANL MTD+1,#0FH
```

```
MOV SLA,#ZLG7290          ;指定器件地址
```

```
MOV SUBA,#07H             ;指定子地址
```

```
MOV NUMBYTE,#02H          ;发送 2 字节数据
```

```
LCALL IWRNBYTE            ;调用写 2 字节数据程序
```

```
MOV MTD,#61H
```

```
MOV A,TEMPER_NUM
```

```
SWAP A
```

```
ANL A,#0FH
```

```
MOV MTD+1,A               ;温度值高位
```

```
MOV SLA,#ZLG7290          ;指定器件地址
```

```
MOV SUBA,#07H             ;指定子地址
```

```
MOV NUMBYTE,#02H          ;发送 2 字节数据
```

```
LCALL IWRNBYTE            ;调用写 2 字节数据程序
```

```
LCALL DELAY
```

```
SJMP DISP_LOOP            ;温度循环采集显示
```

;读出转换后的温度值

GET_TEMPER:

```
SETB DQ                  ;定时入口
```

BCD:

```
LCALL INIT_1820
```

```
JB FLAG1,S22
```

```
LJMP BCD                  ;若 DS18B20 不存在则返回
```

S22:

```
LCALL DELAY1
```

```
MOV A,#0CCH              ;跳过 ROM 匹配-----OCC
```

```
LCALL WRITE_1820
```

```
MOV A,#44H               ;发出温度转换命令
```

```

        LCALL  WRITE_1820
        NOP
        LCALL  DELAY
        LCALL  DELAY
CBA:
        LCALL  INIT_1820
        JB     FLAG1,ABC
        LJMP   CBA
ABC:
        LCALL  DELAY1
        MOV    A,#0CCH                ;跳过 ROM 匹配
        LCALL  WRITE_1820
        MOV    A,#0BEH                ;发出读温度命令
        LCALL  WRITE_1820
        LCALL  READ_18200              ;READ_1820
        RET
;读 DS18B20 的程序,从 DS18B20 中读出一个字节的数据
READ_1820:
        MOV    R2,#8                  ;读取一个字节
RE1:
        CLR    C
        SETB   DQ
        NOP
        NOP
        CLR    DQ
        NOP
        NOP
        NOP
        SETB   DQ
        MOV    R3,#7                  ;延时
        DJNZ   R3,$
        MOV    C,DQ                  ;读取一个位
        MOV    R3,#23                 ;延时
        DJNZ   R3,$
        RRC    A
        DJNZ   R2,RE1
        RET
;写 DS18B20 的程序
WRITE_1820:
        MOV    R2,#8                  ;发送一个字节
        CLR    C
WR1820:
        CLR    DQ
        MOV    R3,#6                  ;延时

```

```

    DJNZ  R3,$
    RRC   A
    MOV   DQ,C           ;发送一个位
    MOV   R3,#23         ;延时
    DJNZ  R3,$
    SETB  DQ
    NOP
    DJNZ  R2,WR1820
    SETB  DQ
    RET

```

;读 DS18B20 的程序,从 DS18B20 中读出两个字节的温度数据

READ_18200:

;将温度高位和低位从 DS18B20 中读出

```

    MOV   R4,#2
    MOV   R1,#36H

```

; 低位存入 36H(TEMPER_L),高位存入 35H(TEMPER_H)

RE00:

```

    MOV   R2,#8           ;读取一个字节

```

RE01:

```

    CLR   C
    SETB  DQ
    NOP
    NOP
    CLR   DQ
    NOP
    NOP
    NOP
    SETB  DQ
    MOV   R3,#7           ;延时
    DJNZ  R3,$
    MOV   C,DQ            ;读取一个位
    MOV   R3,#23         ;延时
    DJNZ  R3,$
    RRC   A
    DJNZ  R2,RE01
    MOV   @R1,A
    DEC   R1
    DJNZ  R4,RE00
    RET

```

;将从 DS18B20 中读出的温度数据进行转换

TEMPER_COV:

```

    MOV   A,#0F0H
    ANL   A,TEMPER_L

```

; 舍去温度低位中小数点后的四位温度数值

```

SWAP  A
MOV   TEMPER_NUM,A
MOV   A,TEMPER_L
JNB   ACC.3,TEMPER_COV1      ;四舍五入去温度值
INC   TEMPER_NUM

```

TEMPER_COV1:

```

MOV   A,TEMPER_H
ANL   A,#07H
SWAP  A
ADD   A,TEMPER_NUM
MOV   TEMPER_NUM,A           ;保存变换后的温度数据
LCALL BIN_BCD
RET

```

;将 16 进制的温度数据转换成压缩 BCD 码

BIN_BCD:

```

MOV   DPTR,#TEMP_TAB
MOV   A,TEMPER_NUM
MOVC  A,@A+DPTR
MOV   TEMPER_NUM,A
RET

```

TEMP_TAB:

```

DB 00H,01H,02H,03H,04H,05H,06H,07H
DB 08H,09H,10H,11H,12H,13H,14H,15H
DB 16H,17H,18H,19H,20H,21H,22H,23H
DB 24H,25H,26H,27H,28H,29H,30H,31H
DB 32H,33H,34H,35H,36H,37H,38H,39H
DB 40H,41H,42H,43H,44H,45H,46H,47H
DB 48H,49H,50H,51H,52H,53H,54H,55H
DB 56H,57H,58H,59H,60H,61H,62H,63H
DB 64H,65H,66H,67H,68H,69H,70H,71H
DB 72H,73H,74H,75H,76H,77H,78H,79H
DB 80H,81H,82H,83H,84H,85H,86H,87H
DB 88H,89H,90H,91H,92H,93H,94H,95H
DB 96H,97H,98H,99H

```

;DS18B20 初始化程序

INIT_1820:

```

SETB  DQ
NOP
CLR   DQ
MOV   R0,#80H

```

TSR1:

```

DJNZ  R0,TSR1      ;延时
SETB  DQ

```

```

        MOV    R0,#25H                ;96uS
TSR2:
        DJNZ   R0,TSR2
        JNB    DQ,TSR3
        LJMP   TSR4                    ;延时
TSR3:
        SETB   FLAG1                  ;置标志位,表示 DS1820 存在
        LJMP   TSR5
TSR4:
        CLR    FLAG1                  ;清标志位,表示 DS1820 不存在
        LJMP   TSR7
TSR5:
        MOV    R0,#06BH                ;200uS
TSR6:
        DJNZ   R0,TSR6                ;延时
TSR7:
        SETB   DQ
        RET

;重新写 DS18B20 暂存存储器设定值
RE_CONFIG:
        JB     FLAG1,RE_CONFIG1        ;若 DS18B20 存在,转 RE_CONFIG1
        RET
RE_CONFIG1:
        MOV    A,#0CCH                ;发 SKIP ROM 命令
        LCALL  WRITE_1820
        MOV    A,#4EH                  ;发写暂存存储器命令
        LCALL  WRITE_1820

        MOV    A,#00H                  ;TH(报警上限)中写入 00H
        LCALL  WRITE_1820
        MOV    A,#00H                  ;TL(报警下限)中写入 00H
        LCALL  WRITE_1820
        MOV    A,#7FH                  ;选择 12 位温度分辨率
        LCALL  WRITE_1820
        RET

;延时子程序
DELAY:
        MOV    R7,#00H
MIN:
        DJNZ   R7,YS500
        RET
YS500:

```

```

        LCALL  YS500US
        LJMP   MIN
YS500US:
        MOV    R6,#00H
        DJNZ   R6,$
        RET
DELAY1:
        MOV    R7,#20H
        DJNZ   R7,$
        RET
$INCLUDE(VI2C_ASM.INC)      ;包含 VIIC 软件包
;
END
    
```

八. 实验思考题

参考图 3.33 电路图，设计一个程序能够实现多路 DS18B20 数字温度采集功能，并能使温度数据通过数码管循环显示。

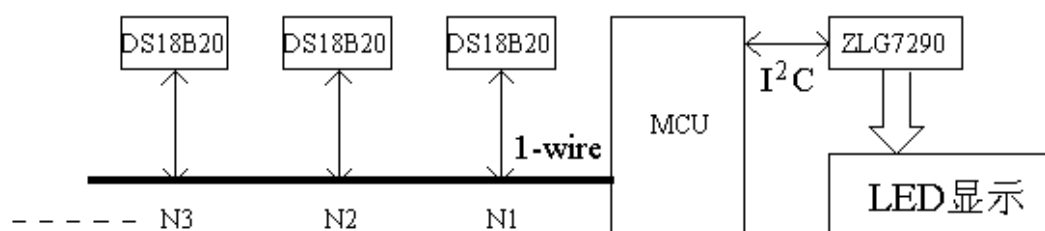


图 3.33

实验二十一 结合 555 电路实验和单片机定时器频率计实验

一. 实验目的

利用单片机的定时器 / 计数器功能, 开发设计一个低频信号频率计。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
10K 电阻	二只
104 电容	二只

三. 实验内容

利用实验十一的 555 多谐振荡器产生的输出为输入 (频率低于 6KHz), 用单片机算出频率, 并在 LED 上显示。

四. 实验要求

用定时器 0 作为定时器, 定时 1s; 用定时器 1 作为计数器, 对输入的脉冲进行计数。利用 ZLG7290 键盘显示芯片在 LED 上显示五位的频率数值。

五. 实验步骤

1. 首先按实验十一的 555 多谐振荡器电路图连接, 其中将 474 电容换为 104 电容。
2. 用导线连接 A2 区的 T1 与 C6 区的 OUT (即 555 的输出)。
3. 用导线连接 A2 区的 P16 与 D5 区的 SCL。
4. 用导线连接 A2 区的 P17 与 D5 区的 SDA。
5. 将 D5 区的 RST_L 连接到 +5V。
6. 短接 C6 区 JP5 的 555 电源跳线和 D5 区 JP1 的电源跳线。
7. 编写程序并运行, 可以在 D5 区的数码管上显示频率值。

六. 实验预习要求

认真阅读实验参考程序, 理解程序的算法和原理。

七. 实验参考程序

```
#include "reg52.h"
#include "Zlg7290.h"      //Zlg7290 库
#include "viic_c51.h"      //I2C 库

unsigned char scout;

void timer0_int() interrupt 1
{
    TR0=0;                //关闭定时器
    TH0=0x4C;              //重装定时器值
```

```

    TL0=0x19;
    TF0=0;                                //清除溢出标志
    scount--;
    if(scount>0)                          //到 1s 了吗?
        TR0=1;                            //没到, 开定时器
    else
        TR1=0;                            //到了, 停止 T1 的计数
}

main()
{
    unsigned char a[5];
    unsigned char i,resh,resl;
    unsigned long int freq;
    TMOD=0xD1;
    //定时器 0 工作在定时方式 1, 定时器 1 工作在计数方式 1
    TH0=0x4C;                            //定时 50ms
    TL0=0x19;
    TH1=0;                                //计数值清 0
    TL1=0;
    scount=20;                            //定时 1s
    ET0=1;                                //开定时器 0 中断
    EA=1;                                  //开总中断
    TR0=1;                                //启动定时器和计数器
    TR1=1;

    for (i=0;i<5;++i)
        a[i]=0;
    ZLG7290_SendBuf(a,5);                //在 LED 上显示 5 位 0。
    while(1)
    {
        if(!scount)                      //1s 时间到
        {
            resh=TH1;                    //取出计数值
            resl=TL1;
            TH1=0;                        //计数值清零
            TL1=0;
            TH0=0x4C;                    //重装定时器 0
            TL0=0x19;
            scount=20;                    //定时 1S
            TR0=1;                        //启动定时器和计数器
            TR1=1;
            freq=resh*256+resl;           //计算频率值
            a[0]=freq%10;                //将各位分离显示
        }
    }
}

```

```
        a[1]=(freq/10)%10;
        a[2]=(freq/100)%10;
        a[3]=(freq/1000)%10;
        a[4]=freq/10000;
        ZLG7290_SendBuf(a,5);    //送 ZLG7290 显示
    }
}
}
```

八. 实验思考题

- (1) 请编写一段程序实现 6KHz 以上的频率计。
- (2) 请用硬件实现，用本程序制作 6KHz 以上的频率计。

实验二十二 直流电机实验

一. 实验目的

利用 PWM 控制直流电机的转动速度。

二. 实验设备及器件

IBM PC 机 一台

DP-51PROC 单片机综合仿真实验仪 一台

三. 实验内容

学习如何控制直流电机。PWM 功率驱动电路如下：

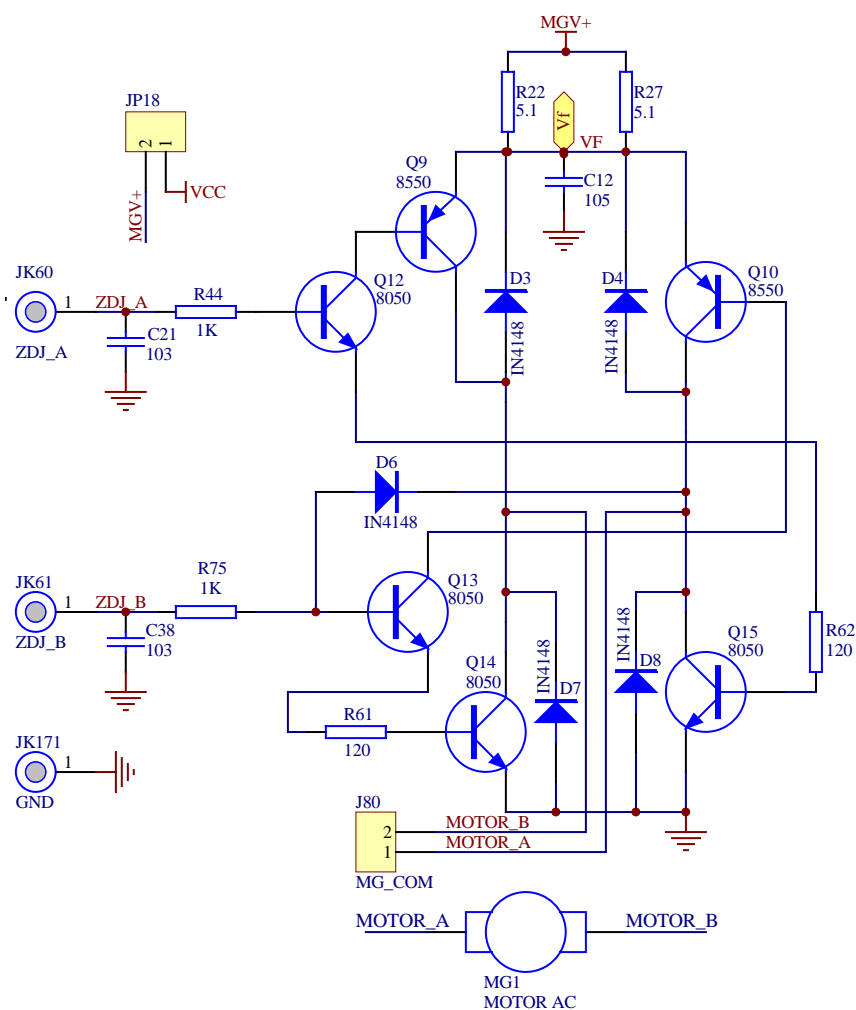


图 3.34 直流电机驱动原理图

原理图如图 3.34，只要 ZDJ_A 的电压比 ZDJ_B 的电压高，电机正转。如果 ZDJ_B 的电压比 ZDJ_A 高，电机反转。

四. 实验要求

利用实验六的程序，用 D1 区的按键 KEY1 与 KEY2 改变 PWM 的占空比来控制直流电

机的转速。

五. 实验步骤

1. 用导线连接 A2 区的 P11 与 D1 区的 KEY1。
2. 用导线连接 A2 区的 P12 与 D1 区的 KEY2。
3. 用导线连接 A2 区的 P10 与 B10 区的 ZDJ_A。
4. B10 区的 ZDJ_B 连接到 C1 区的 GND。
5. 短接 B10 区 JP18 的电机电源跳线。

六. 实验预习要求

认真阅读本书的 2.8.16 节内容，理解电机驱动电路的原理。

七. 实验参考程序

参见实验六的程序。

八. 实验思考题

请编写一段程序实现电机的正、反转。

实验二十三 步进电机控制实验

一. 实验目的

了解步进电机的工作原理，掌握它的转动控制方式和调速方法。

二. 实验设备及器件

IBM PC 机 一台
DP-51PROC 单片机综合仿真实验仪 一台

三. 实验内容

1. 编写程序，通过单片机的 P1 口控制步进电机的控制端，使其按一定的控制方式进行转动。
2. 分别采用双四拍（AB→BC→CD→DA→AB）方式、单四拍（A→B→C→D→A）方式和单双八拍（A→AB→B→BC→C→CD→D→DA→A）方式编程，控制步进电机的转动方向和转速。
3. 观察不同控制方式下，步进电机转动时的振动情况和步进角的大小，比较这几种控制方式的优缺点。

四. 实验要求

学会步进电机的工作原理和控制方法，掌握一些简单的控制电路和基本的电机基础知识。

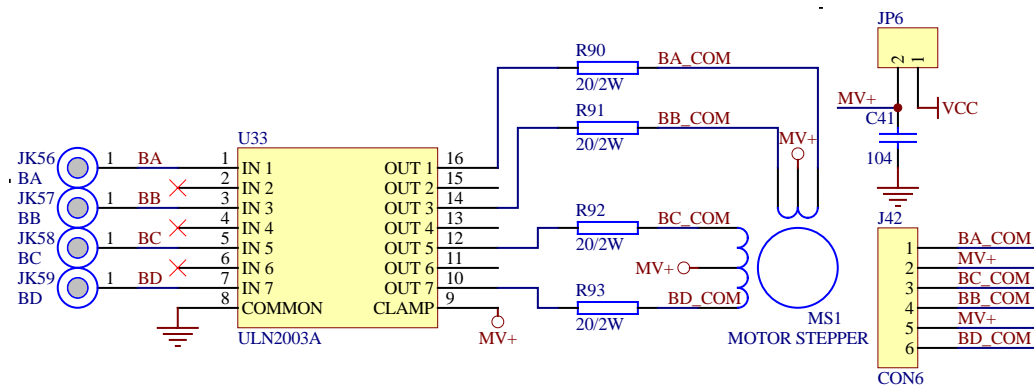


图 3.35 步进电机驱动原理图

五. 实验步骤

1. 安装 C10 区 JP6 接口上的短路帽，将 C10 区 BA、BB、BC、BD 与 A2 区的 P10~P13 对应相连。
2. 打开程序调试软件，下载运行编写好的软件程序，观察步进电机的转动情况。
3. 修改步进电机的控制程序，再次运行程序，比较它们的不同控制效果。

六. 实验预习要求

认真预习本书 2.8.24 节的内容，认真按照本实验的要求提前做好实验准备工作，阅读实验参考程序。

七. 实验参考程序

```

BA      EQU      P1.0
BB      EQU      P1.1
BC      EQU      P1.2
BD      EQU      P1.3
ORG     8000H
LJMP    MAIN
ORG     8100H

MAIN:
    MOV    SP,#60H      ;设置堆栈指针
    ACALL  DELAY
SMRUN:      ;电机控制方式为单双八拍
    MOV    P1,#08H      ;A
    ACALL  DELAY
    MOV    P1,#0CH      ;AB
    ACALL  DELAY
    MOV    P1,#04H      ;B
    ACALL  DELAY
    MOV    P1,#06H      ;BC
    ACALL  DELAY
    MOV    P1,#02H      ;C
    ACALL  DELAY
    MOV    P1,#03H      ;CD
    ACALL  DELAY
    MOV    P1,#01H      ;D
    ACALL  DELAY
    MOV    P1,#09H      ;DA
    ACALL  DELAY
    SJMP   SMRUN        ;循环转动
DELAY:      ;延时程序
    MOV    R4,#10
DELAY1:
    MOV    R5,#250
    DJNZ   R5,$
    DJNZ   R4,DELAY1
    RET
;
END

```

八. 实验思考题

设计一个完整的步进电机控制程序，使用户可以通过按键控制电机转动的方向，并且能够调节电机转动的速度。

实验二十四 红外收发实验

一. 实验目的

了解红外通讯知识，能够应用红外进行无线控制设计。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三. 实验内容

使用单片机的串口发送并接收数据，TXD 接到红外发送管，RXD 接到红外接收头，实现无线通讯。

四. 实验要求

通过实验掌握红外通讯的基本原理。

五. 实验步骤

1. B2 区 X2 插入 20MHz 的晶振。
2. 将 B2 区的 1/512 频率输出端接到 D3 区的 DCLK。(约为 39KHz, 用于信号调制)
3. 将 A2 区的 RXD、TXD 分别连接到 D3 区的 DREC、DSEND。
4. 用短路器将 D3 区 JP9 短接(D3 区电路供电电源)。
5. 将 A2 区的 P10 连接到 D1 区的 LED1。
6. 下载程序并运行，使用较厚的白纸挡住红外发射管红外信号，使其反射到接收头，观察 LED1 是否点亮。

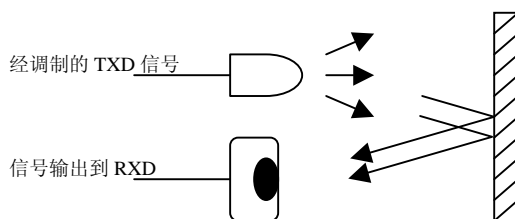


图 3.36 红外收发实验示意图

说明：一般红外接收模块的解调频率为 38KHz，当它接收到 38KHz 左右的红外信号时将输出低电平，但连续输出低电平的时间是有限制的(如 100mS)，也就是说发送数据的低电平宽度是有限制的。

注意：发送管应与接收头平行，否则接收头可能接收不到来自发射管的反射光。

六. 实验预习要求

阅读本书的 2.8.27 节内容，理解本实验硬件结构，编写实验程序，编译连接后使用 DPFlash 软件将 HEX 文件下载到 TKSMonitor51 仿真器中，再复位运行调试。

七. 实验参考程序

```

/*****
* 文件名: DP_51PRO_IRDA.C
* 功能: DP_51PRO 实验仪红外收发实验。使用串口发送数据经调制后从红外管输出,
*       并通过红外接收模块把接收到的数据返回串口接收端, 通过判断接收到的数
*       据来控制 LED 灯的亮或灭。
*****/
#include <Reg52.H>

#define uint8      unsigned char
#define uint16     unsigned int

sbit      LED_CON    = P1^0;                // 定义 LED 控制口

/*****
* 名称: UART_SendByte()
* 功能: 向串口发送一字节数据。
* 入口参数: dat      要发送的数据
* 出口参数: 无
*****/
void UART_SendByte(uint8 dat)
{
    SBUF = dat;                // 发送数据
    while(TI == 0);           // 等待发送完毕
    TI = 0;                    // 清零 TI 标志
}

/*****
* 名称: UART_RcvByte()
* 功能: 接收一字节串口数据。
* 入口参数: dat      接收变量的地址指针
* 出口参数: 返回 0 表示没有数据, 返回 1 表示接收到数据
*****/
uint8 UART_RcvByte(uint8 *dat)
{
    if(RI == 0) return(0);     // 若没有接收到数据则返回 0
    *dat = SBUF;               // 取得接收的数据
    RI = 0;                    // 清除 RI 标志
    return(1);
}

/*****
* 名称: UART_Init()
* 功能: 串口初始化。模式为 1 位起始位, 8 位数据位, 1 位停止位, 波特率为 9600。
* 入口参数: 无
* 出口参数: 无
* 说明: 晶振为 11.0592MHz, 使用 T1 作为波特率发生器。
*****/
void UART_Init(void)
{
    SCON = 0x50;
    TMOD = 0x20;
    TH1 = 0xFD;
    TR1 = 1;
}
    
```

```

}

/*****
* 名称: main()
* 功能: 主函数, 初始化串口后不断地发送及接收数据, 若接收到所发送的数据则
*       点亮 LED。
*****/
int main(void)
{
    uint8 i;
    uint16 j;
    uint8 rcv_dat;
    uint8 count;

    UART_Init();
    while(1)
    {
        count = 0; // 计数变量清零
        for(i=0; i<50; i++) // 发送及接收 50 个数据
        {
            UART_SendByte(0x5A);
            if( UART_RcvByte(&rcv_dat)!=0 )
            {
                if(0x5A==rcv_dat) count++; // 若接收的数据为 0x5A, 则计数变量加 1
            }
        }
        if(count>40) LED_CON = 0; // 若接收到 0x5A 的个数大于 40 个时, 点亮 LED
        else LED_CON = 1; // 否则熄灭 LED

        for(j=0; j<500; j++);
    }

    return(0);
}

```

八. 实验思考题

- (1) 如何编写其它编码格式的通讯程序?
- (2) 红外通讯的距离与什么因素有关? 使用两台实验仪进行测试, 一台发送, 另一台接收。

实验二十五 字符型液晶显示实验

一. 实验目的

掌握字符型液晶模块的控制方法，能够编写驱动程序及高级接口函数。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三. 实验内容

控制 TC1602A 字符型液晶模块显示字符串。

四. 实验要求

掌握 TC1602A 字符型液晶模块(驱动芯片为 HD44780)控制方法。

五. 实验步骤

1. 将 A2 区的 A0、A1 分别连接到 B3 区的 A0、A1。
2. 将 A4 区的/Y0 连接到 B3 的/CS。
3. 将 A2 区的 A15~A10 分别连接到 A4 区的相应接线柱，连接关系如下：

A15	——	/G2B
A14	——	/G2A
A13	——	G1
A12	——	C
A11	——	B
A10	——	A

4. 在 B3 区的 J106 插入 TC1602A 字符型液晶模块。
5. 将 B3 区的 JP21 短接，A4 区的 JP4 短接。
6. 编写代码，使用 TKSMonitor51 仿真器进行仿真调试。

六. 实验预习要求

阅读本书的 2.8.9 节内容，理解本实验硬件结构，分析实验连线分配给 LCM 的地址，编写实验程序，编译连接后使用 TKSMonitor51 仿真器进行仿真调试。

七. 实验参考程序

```
/*
* 文件名: LCMDRV.C
* 功能: TC1602A 驱动程序。提供初始化、发送命令、发送数据、发送字符串等函数。
*/
#include "LCMDRV.H"
#include <intrins.h>
```

```

/*****
** 函数名称: DelayShort()
** 功能描述: 短延时操作, 延时时间约为 dly 个机器周期。
** 输 入: dly    延时控制, 即延时周期数。(参数为 0 时不进行延时)
** 输 出: 无
*****/
void DelayShort(uint8 dly)
{ for( ; dly>0; dly--) _nop_();
}

/*****
** 函数名称: LCMSendDate()
** 功能描述: 向 LCM 模块发送数据 date。
** 输 入: odata    要输出的数据
** 输 出: 无
** 注意: 使用此函数前要初始化好 LCM, 再用此函数发送显示数据。
*****/
void LCMSendDate(uint8 odate)
{ LCMWR_DAT = odate;
  DelayShort(100);
}

/*****
** 函数名称: LCMSendComm()
** 功能描述: 向 LCM 模块发送命令 comm。
** 输 入: comm      要输出的命令
** 输 出: 无
** 注意: 调用此函数来初始化、设置 LCM
*****/
void LCMSendComm(uint8 comm)
{ LCMWR_COM = comm;
  DelayShort(100);
}

/*****
** 函数名称: LCMDispStr()
** 功能描述: 向 LCM 模块发送字符串数据, 从 DDRAM 的指定地址 adr 开始一直写, 直到
** 字符串结束或超出屏幕显示范围。此函数可以自动换行。
** 输 入: dstr      要写的字符串指针
**        adr      写入的 DDRAM 起始地址(如 LCM_LINE1)
** 输 出: 无
** 注意: 使用此函数前要初始化好 LCM。
**        (适用于双行 LCM 设置, 且显示为光标移动, DDRAM 地址向上增长)
*****/

```

```

void LCMDispStr(uint8 adr, char *dstr)
{
    if( adr<LCM_LINE1 ) return;
    if( adr>(LCM_LINE2+15) ) return;
    if( (adr>(LCM_LINE1+15)) && (adr<LCM_LINE2) ) return;

    /* 检查字串是否为空 */
    if( (*dstr)=='\0' )return;

    LCMSendComm(adr);                // 设置 DDRAM 地址
    while(1)
    {
        if( (*dstr)=='\0' )break;    // 若字串已经结束，则退出发送
        LCMSendDate(*dstr);
        dstr++;
        /* 若第一行显示完毕，则指向第二行，进行显示输出 */
        if( adr==(LCM_LINE1+15) )
        {
            adr = LCM_LINE2;
            LCMSendComm(adr);        // 从第 2 行开始显示
        }
        else
        {
            adr++;
        }
        if( adr>LCM_LINE2+15 )return; // 地址超出
    }
}

/*****
** 函数名称: LCMIIni()
** 功能描述: 初始化 LCM 模块，设置为 LCM_MODE、LCM_NOFLASH、
** LCM_NOSHIFT、LCM_SH，然后清屏。
** 输 入: 无
** 输 出: 无
*****/
void LCMIIni(void)
{
    LCMSendComm(LCM_RST);           // 软复位
    LCMSendComm(LCM_MODE);          // 设置 LCM 模式(系统方式设置)，值为 6
    LCMSendComm(LCM_NOFLASH);       // 打开 LCM 显示，无光标，4
    LCMSendComm(LCM_NOSHIFT);       // 字符向地址递增，光标移动，3
    LCMSendComm(LCM_SH);            // 重新设为光标移动，向右移，5(单指令)
    LCMDispStr(LCM_LINE1, CLRSTR);  // 清屏
    LCMDispStr(LCM_LINE2, CLRSTR);
}

/*****
* 文件名: LCM_DEMO.C
*****/

```

- * 功能: DP_51PRO 实验仪字符型液晶显示实验。屏幕第一行中央显示"DP-51PROC", 第
- * 二行则显示"WWW.ZLGMCU.COM"和"020-38730916", 并不断的循环向左移动。

*****/

```
#include "LCMDRV.H"
```

```
#include <string.h>
```

* 名称: DelayS()

* 功能: 长软件延时。

* 入口参数: delayno 延时时间控制

*****/

```
void DelayS(uint8 delayno)
```

```
{ uint8 i,j;
```

```
for(; delayno>0; delayno--)
```

```
{ for(i=0; i<250; i++)
```

```
for(j=0; j<200; j++);
```

```
}
```

```
}
```

* 名称: main()

* 功能: 主函数, 控制 LCM 显示。先初始化液晶模块, 然后显示字符串, 并不断的循环向

* 左移动。

*****/

```
void main(void)
```

```
{ char code DISP_STR[]
```

```
= " WWW.ZLGMCU.COM 020-38730916 ";
```

```
uint8 start_no;
```

```
uint8 len;
```

```
LCMIni(); // 初始化 LCM
```

```
LCMDispStr(LCM_LINE1, " DP-51PROC ");
```

```
len = strlen(DISP_STR)-16;
```

```
while(1)
```

```
{ for(start_no=0; start_no<=len; start_no++)
```

```
{ LCMDispStr(LCM_LINE2, &DISP_STR[start_no]);
```

```
DelayS(1);
```

```
}
```

```
}
```

```
}
```

八. 实验思考题

- (1) 如何将用户的字模数据写入到 CG RAM?

实验二十六 图形液晶显示实验

一. 实验目的

了解图形液晶模块(单色)的控制方法, 实现简单图形显示算法。

二. 实验设备及器件

IBM PC 机 一台
DP-51PROC 单片机综合仿真实验仪 一台

三. 实验内容

控制图形液晶模块显示 ASCII 字符, 画直线。

四. 实验要求

掌握字符/图形在图形液晶上显示的理论知识, 并能够编写相关操作函数。

五. 实验步骤

1. 将 A2 区的 A0~A2 分别连接到 B3 区的 A0~A2。
2. 将 A2 区的 P10 连接到 B3 区的 RST。
3. 将 A3 区的 /Y0 连接到 B3 区的 /CS。
4. 将 A2 区的 A15~A10 分别连接到 A3 区的相应接线柱, 连接关系如下:

A15	———	/G2B
A14	———	/G2A
A13	———	G1
A12	———	C
A11	———	B
A10	———	A

5. 在 B3 区的 J92 插入图形液晶模块(单色, 128×64 点)。
6. 将 B3 区的 JP21 短接, A3 区的 JP4 短接。
7. 编写代码, 使用 TKSMonitor51 仿真器进行仿真调试。

六. 实验预习要求

阅读本书的 2.8.9 节内容, 理解本实验仪硬件结构, 分析实验连线分配给 LCM 的地址, 编写实验程序, 编译连接后使用 TKSMonitor51 仿真器进行仿真调试。

七. 实验参考程序

```
/******
```

```
* 文件名: LCM_DRIVE.C
```

```
* 功能: 图形液晶 TG12864B-2 驱动程序。
```

```
* 说明: 在 LCM_DRIVE.H 文件中定义了 LCM 操作地址, 左半屏的写命令操作地址为
```

```
* 2004H, 写数
```

```
* 据操作地址为 2005H, 右半屏的写命令操作地址为 2000H, 写数据操作地址为 2001H;
```

*于 GRAPHICS.C 中使用了 disp_buf 作为作图缓冲区，所以 LCM_WriteByte()、LCM_DispFill()均要更新 disp_buf。

*****/

#include "CONFIG.H"

/* LCM 复位控制脚定义 */

sbit LCM_RST = P1^0;

* 名称: LCM_Wr1Command()

* 功能: 写命令子程序，所选屏为左半屏(CS1)。

* 入口参数: command 要写入 LCM 的命令字

*****/

#define LCM_Wr1Command(command) LCMCS1W_COM = command

* 名称: LCM_Wr2Command()

* 功能: 写命令子程序，所选屏为右半屏(CS2)。

* 入口参数: command 要写入 LCM 的命令字

*****/

#define LCM_Wr2Command(command) LCMCS2W_COM = command

* 名称: LCM_Wr1Data()

* 功能: 写数据子程序，所选屏为左半屏(CS1)。

* 入口参数: wrdata 要写入 LCM 的数据

*****/

#define LCM_Wr1Data(wrdata) LCMCS1W_DAT = wrdata

* 名称: LCM_Wr2Data()

* 功能: 写数据子程序，所选屏为右半屏(CS2)。

* 入口参数: wrdata 要写入 LCM 的数据

*****/

#define LCM_Wr2Data(wrdata) LCMCS2W_DAT = wrdata

* 名称: LCM_DispIni()

* 功能: LCM 显示初始化。使能显示，设置显示起始行为 0 并清屏。

* 入口参数: 无

* 出口参数: 无

*****/

void LCM_DispIni(void)

{ uint16 i;


```

    LCM_RST = 0;                                // 复位驱动芯片
    for(i=0; i<500; i++);
    LCM_RST = 1;

    LCM_Wr1Command(LCM_DISPON);                  // 打开显示
    LCM_Wr1Command(LCM_STARTROW);                 // 设置显示起始行为 0
    LCM_Wr2Command(LCM_DISPON);
    LCM_Wr2Command(LCM_STARTROW);
    LCM_DispcIr();                                // 清屏

    LCM_Wr1Command(LCM_ADDRSTRY+0);               // 设置页(行)地址
    LCM_Wr1Command(LCM_ADDRSTRX+0);               // 设置列地址, 即列
    LCM_Wr2Command(LCM_ADDRSTRY+0);
    LCM_Wr2Command(LCM_ADDRSTRX+0);
}

/*****
* 名称: LCM_WriteByte()
* 功能: 向指定点写数据(一字节)。
* 入口参数: x          x 坐标值(0-127)
*           y          y 坐标值(0-63)
*           wrdata      所要写的数据
* 出口参数: 无
* 说明: 更新 disp_buf 相应存储单元
*****/
void LCM_WriteByte(uint8 x, uint8 y, uint8 wrdata)
{
    x = x&0x7f;                                // 参数过滤
    y = y&0x3f;

    y = y>>3;
    disp_buf[y][x] = wrdata;
    if(x<64)    // 选择液晶控制芯片(即 CS1--控制前 64 个点, CS2--控制后 64 个点)
    {
        LCM_Wr1Command(LCM_ADDRSTRX+x); // 设置当前列地址, 即 x 坐标
        LCM_Wr1Command(LCM_ADDRSTRY+y); // 设置当前页地址, 即 y 坐标
        for(x=0; x<5; x++);              // 短延时
        LCM_Wr1Data(wrdata);
    }
    else
    {
        x = x-64;                          // 调整 x 变量值
        LCM_Wr2Command(LCM_ADDRSTRX+x);
        LCM_Wr2Command(LCM_ADDRSTRY+y);
        for(x=0; x<5; x++);
        LCM_Wr2Data(wrdata);
    }
}

```

```

    }
}

/*****
* 名称: LCM_DispFill()
* 功能: 向显示屏填充数据
* 入口参数: filldata    要写入 LCM 的填充数据
* 出口参数: 无
* 说明: 会更新 disp_buf 相应存储单元
*****/
void LCM_DispFill(uint8 filldata)
{
    uint8 x, y;

    LCM_Wr1Command(LCM_STARTROW);    // 设置显示起始行为 0
    LCM_Wr2Command(LCM_STARTROW);

    for(y=0; y<8; y++)
    {
        LCM_Wr1Command(LCM_ADDRSTRY+y); // 设置页(行)地址
        LCM_Wr1Command(LCM_ADDRSTRX);   // 设置列地址
        LCM_Wr2Command(LCM_ADDRSTRY+y);
        LCM_Wr2Command(LCM_ADDRSTRX);

        for(x=0; x<64; x++)
        {
            LCM_Wr1Data(filldata);
            LCM_Wr2Data(filldata);
            disp_buf[y][x] = filldata;
            disp_buf[y][x+64] = filldata;
        }
    }
}
}

```

(ASCII 码显示函数及画直线函数见原文件)

```

/*****
* 文件名: LCM_DISP.C
* 功能: DP_51PRO 实验仪图形液晶显示实验。先在屏幕上的随机位置显示随机数字
* (0--9)，然后调用画直线函数以屏幕中心为中点画对称的直线，最后显示一个简单的窗
* 口。
*****/
#include "CONFIG.H"

/*****
* 名称: DelayS()
* 功能: 长软件延时。

```

* 入口参数: delayno延时时间控制

*****/

void DelayS(uint16 delayno)

{ uint16 i;

for(; delayno>0; delayno--)

{ for(i=0; i<1000; i++);

}

}

* 名称: main()

* 功能: 主函数, 初始化图形液晶模块, 在屏幕上的随机位置显示随机数字(0--9),

* 然后调用画直线函数以屏幕中心为中点画对称的直线, 最后显示一个简单的窗口。

*****/

void main(void)

{ uint8 i;

uint8 bak;

while(1)

{ LCM_DispIni(); // 初始化图形液晶模块

/* 提示加载程序, 即显示"Load..." */

LCM_DispStr(6, 3, "Load");

for(i=0; i<5; i++)

{ LCM_DispChar(6, i+7, '.');

DelayS(10);

}

/* 数字显示演示 */

srand(168); // 初始化随机种子

LCM_DispClr();

for(i=0; i<250; i++)

{ uint8 x, y;

x = rand()&0x07; // 取得随机显示位置

y = rand()&0x0f;

bak = rand()%10 + '0'; // 取得随机显示数字'0'--'9'

LCM_DispChar(x, y, bak);

DelayS(2);

}

#define CENTER_X 64

#define CENTER_Y 32

/* 直线演示。由于直线要基于中心点, 具有对称性, 所以只要随机取出第一个点,

即可计算出总线对称点，两点便可确定一条直线。这样做达到不同直线不同角度的目的。 */

```
LCM_DispClr();
for(i=0; i<250; i++)
{
    uint8 x0, y0;
    uint8 x1, y1;

    /* 取第一个点坐标 */
    x0 = rand() & 0x7f;          // 取得随机起点
    y0 = rand() & 0x3f;

    /* 计算出中心对称点 */
    x1 = 2 * CENTER_X - x0;
    y1 = 2 * CENTER_Y - y0;

    Line(x0, y0, x1, y1, 1);
    DelayS(5);
}

/* 画窗口演示 */
LCM_DispClr();
Rectangle(0, 0, 127, 63, 1);    // 显示窗口
HLine(0, 10, 127, 1);
RLine(10, 0, 10, 1);
Line(0, 0, 10, 10, 1);
Line(10, 0, 0, 10, 1);
RLine(120, 10, 63, 1);          // 显示滚动条
RectangleFill(120, 20, 127, 30, 1);
LCM_DispStr(4, 4, "DP-51PROC");  // 在窗口中显示内容
LCM_DispStr(5, 1, "www.zlgmcu.com");
DelayS(1500);
} // end of while(1) ...
}
```

八. 实验思考题

- (1) 如何进行汉字显示？
(提示：汉字是由固定点阵格式组成的图形，字模一般有 16×16 和 24×24 等)
- (2) 如何实现动画显示？
(提示：不断循环操作——显示图像，短延时，擦除原图像，显示新图像……)

实验二十七 串行模数转换实验

一. 实验目的

熟悉 A/D 转换的工作原理，学习使用串行模数转换芯片 TLC549 进行电压信号的采集和数据处理。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
万用表	一台

三. 实验内容

1. 编写程序，通过单片机的 P1 口控制串行 A/D 转换芯片 TLC549 实现模拟电压信号的采集。
2. 连接线路，调整 TLC549 的输入参考电压为 5V（调节图 3.37 中的 W3 电位器），运行程序实现 A/D 转换和电压信号采集。

四. 实验要求

理解掌握 TLC549 的 A/D 转换原理和串行 A/D 转换器接口的编程方法，学会使用 TLC549 串行模数转换器实现电压信号采集的方案设计。

五. 实验步骤

1. 安装 B8 区 JP17 的短路帽，然后将 VCC（+5V 电源）与 B8 区的 REF+ 连接，将 B8 区的 CLK、DAT、/CS 对应连接到 A2 区的 P10、P11、P12 针上。
2. 使用导线将 D2 区的 10K 电位器连接为电压调节模式，使用导线将其电压调整端连接到 B8 区的 ANIN 接线柱，作为 TLC549 的模拟电压信号输入。
3. 打开程序调试软件，下载运行编写好的程序，完成一次 A/D 转换，然后调节电位器改变输入模拟电压，多次测量并保存测量数据。
4. 使用万用表测量输入的模拟电压信号，分析采集到的 A/D 转换数据是否准确。

参考电路图如下所示：

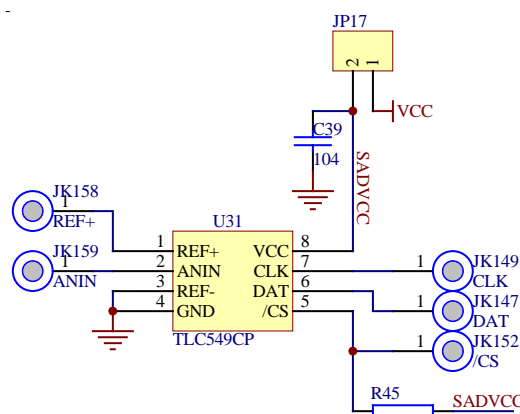


图 3.37 实验 30 电路图

六. 实验预习要求

认真预习本节实验内容，按照实验的要求提前做好实验准备工作，认真阅读 TLC549 的数据手册。

七. 实验参考程序

；功能：用串行 A/D 转换器 TL549 进行一路模拟量的测量
 ；驱动 TLC549，TLC549 是串行 8 位 ADC
 ；使用的接口 /CS = P1.2, DAT = P1.1, CLK = P1.0
 ；转换结果保存于内部 RAM 30H 单元

```
CS          BIT    P1.2
DAT         BIT    P1.1
CLK         BIT    P1.0
AD_DATA     DATA  30H
ORG 8000H
AJMP MAIN
ORG 8100H
```

MAIN:

```
MOV SP,#60H
ACALL TLC549_ADC
MOV R7,#0
DJNZ R7,$
ACALL TLC549_ADC      ;读取上次 ADC 值，并再次启动 AD 转换
MOV AD_DATA,A
SJMP $
```

；TLC549 串行 ADC 转换器的驱动程序

；TLC549 在读出前一次数据后，马上进行电压采样，ADC 转换，转换完后就进入 HOLD
 ；模式，直到再次读取数据时，芯片才会进行下一次 AD 转换。也就是说，本次读出的
 ；数据是前一次的转换值，读操作后就会再启动一次转换，一次转换所用的时间最长为
 ；17uS，芯片没有转换结束信号输出。TLC549 ADC 转换程序，读取前一次转换值并
 ；返回，然后再启动一次 ADC 转换。

TLC549_ADC:

```
CLR A
CLR CLK
CLR CS      ;选中 TLC549
MOV R6,#8
```

TLCAD_L1:

```
SETB CLK
NOP
NOP
MOV C,DAT
RLC A
CLR CLK      ;DAT=0，为读出下一位数据作准备
```

```

NOP
DJNZ  R6,TLCAD_L1
SETB  CS          ;禁能 TLC549，再次启动 AD 转换
SETB  CLK
RET
;
END
```

实验二十八 串行数模转换实验

一. 实验目的

学会使用 D/A 转换器生成用户所需要的波形。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
双踪示波器	一台

三. 实验内容

1. 设计软件程序, 用单片机的 I/O 口控制 TLC5620 实现 D/A 转换, 使其通道 1 产生一个三角波, 而通道 2 产生一个和通道 1 周期、幅度均相同的方波。
2. 连接线路, 调整 TLC5620 的参考电压为 2.6V, 运行程序, 用双踪示波器观察产生的波形。

四. 实验要求

实验前阅读有关 TLC5620 的数据手册, 并了解 TLC5620 的 4 种时序图, 以及产生波形幅度的计算方法。

五. 实验步骤

1. 短接 B7 区的电源供给跳线 JP16, 调节 B7 区的电位器 W3, 使其输出接线柱 Verf 的电压为 2.6V。
2. 将 A2 区 P16、P17、T0、T1 分别连接到 B9 区的 CLK、DAT、LDAC、LOAD, 将 B7 区 Verf 连接到 B9 区 REF 接线柱, 短接 B9 区电源跳线 JP13。
3. 运行光盘中的相应程序, 用双踪示波器的两个探头观察 DACA、DACB 输出的波形。

六. 实验预习要求

认真预习本节实验内容, 按照实验的要求提前做好实验准备工作。

七. 实验参考程序

```
SCLA    BIT    P1.6
SDAA    BIT    P1.7
LOAD    BIT    P3.5
LDAC     BIT    P3.4
VOUTA   DATA   30H
VOUTB   DATA   31H
```

```
ORG     8000H
AJMP    MAIN
ORG     8100H
```


MAIN:

```
MOV    SP,#60H
NOP
CLR     SCLA
CLR     SDAA
SETB    LOAD
SETB    LDAC
MOV     R3,#0A2H
MOV     R4,#00H
MOV     VOUTA,#00H
MOV     R5,#0A2H
MOV     R6,#00H
MOV     VOUTB,#00H
```

DACHANG:

```
MOV     R1,#01H
MOV     R2,VOUTA
LCALL   DAC5620
DJNZ    R3,CONTINUEA
MOV     R3,#0A2H
MOV     A,R4
CPL     A
MOV     R4,A
```

CONTINUEA:

```
CJNE    R4,#0FFH,CONTINUEB
DEC      R2
SJMP    CONTINUEC
```

CONTINUEB:

```
INC      R2
```

CONTINUEC:

```
MOV     VOUTA,R2
MOV     R1,#03H
MOV     R2,VOUTB
LCALL   DAC5620
DJNZ    R5,CONTINUED
MOV     R5,#0A2H
MOV     A,R6
CPL     A
MOV     R6,A
```

CONTINUED:

```
CJNE    R6,#0FFH,CONTINUEE
MOV     R2,#0A2H
SJMP    CONTINUEF
```

CONTINUEE:

```
MOV     R2,#00H
```

CONTINUEF:

MOV VOUTB,R2

LJMP DACHANG

;=====

; 函数名称: DAC5620

; 功能描述: TI 公司 8 位 4 通 DAC 芯片 TLC5620 的控制时序

; 入口参数: R1(通道选择和幅度增量)、R2(转换电压的数字值, 公式中的 CODE)

; 出口参数: 无

; 占用资源: ACC、R1、R2

; 全局变量: 无

; 调用模块: SENDBYTE

; 日期: 2004-10-27

; 备注: 使用双 8 位传输控制模式, 使用 LDAC 下降沿锁存数据输入

;=====

DAC5620:

MOV A,R1

CLR SCLA

MOV R7,#08H

LCALL SENDBYTE

MOV A,R2

CLR SCLA

MOV R7,#08H

LCALL SENDBYTE

CLR LOAD

SETB LOAD

CLR LDAC

SETB LDAC

RET

SENDBYTE:

SETB SCLA

RLC A

MOV SDAA,C

CLR SCLA

DJNZ R7,SENDBYTE

RET

END

实验二十九 IC 卡身份识别开关实验

一. 实验目的

掌握如何灵活运用已有的模块，做出 IC 卡身份识别开关。

二. 验设备及器件

IBM PC 机 一台
DP-51PROC 单片机综合仿真实验仪 一台

三. 实验内容

利用 I²C 模块、IC 卡读卡模块编一段身份识别程序，并由此控制继电器动作。

四. 实验要求

了解身份识别原理，熟练运用各模块。

五. 实验步骤

1. 连线。

引脚连线关系见表 3.7

表 3.7

单片机	实验仪板上对应引线	
(A2 区)P1.0	(D6 区)TP1	
(A2 区)P1.1	(D6 区)TP4	
(A2 区)P1.2	(D6 区)TP3	
(A2 区)P1.5	(D5 区)SDA	
(A2 区)P1.6	(D5 区)SCL	
(A2 区)P1.7	(C7 区)J9	
	(C7 区)OPEN1	(D1 区)LED1
	(C7 区)COM1	(C1 区)GND

另外，使用跳线帽短接 D5 区的 JP1 和 C7 区的 JP7。

2. 利用实验十八 IC 卡读写程序，初始化好一张卡号为 0x12345678 的卡（对 SLE4442 从 20H 单元开始，写入 4 个字节，分别为 0X12,0X34,0X56,0X78）。
3. 利用实验十六对 24WC02 从 20H 单元开始写入 4 个字节，分别为 0X12,0X34,0X56,0X78。
4. 分别利用 I²C 模块，IC 卡读写模块，编写 IC 卡读数据及数据比较程序。
5. 当卡合法时，控制继电器吸合，并用控制 LED1 亮来表示。

六. 实验预习要求

复习实验十八及实验十六的内容，并做好一张卡号为 0x12345678 的卡备用。

七. 实验参考程序

```
*****  
;  
;注意：单片机的晶振频率选用 11.0592MHZ
```

```
;在使用本程序前应先写好 IC 卡和 EEPROM 的数据
;在插入合法的卡 (IC 卡内从 20H 处开始的数据与 24WC02 中 20H 开始的数据相同)
;则继电器通电 LED 亮, 否则继电器断电 LED 灭
, *****
;
```

```
SLA      DATA  70h    ;器件从地址
SUBA      DATA  71h    ;器件子地址
NUMBYTE   DATA  72h    ;读 / 写的字
```

```
ACK      BIT    20H    ;位变量 ACK
```

```
;定义常量:
```

```
SDA      BIT    P1.5
SCL      BIT    P1.6    ;总线位
SW       BIT    P1.7    ;继电器控制脚
```

```
MTD:    DS      10H      ;发送数据缓冲区首址
MRD:    DS      10H      ;接收数据缓冲区首址
```

```
ORG      8000H
AJMP     MAIN
```

```
ORG      8100H
MAIN:
MOV      SP,#60H        ;设置栈底
```

```
Read_Insert_Card:
```

```
MOV      ByteNum,#04H
MOV      StartAdr,#20H  ;从主存储区的 20H 地址读 4 个字节
LCALL    ReadCard
```

```
;在此可查看内存 ReadBuf 中的数据是否合法
```

```
MOV      SLA,#0a0H      ;24WC02 的器件地址
MOV      SUBA,#20h
MOV      NUMBYTE,#4h
```

```
LCALL    IRDNBYTE        ;从 24WC02 的 20H 单元开始处读取 4 字节的数据
;放在 MRD 内
```

```
;在此可查看内存中 MRD 的数据是否合法
```

```
MOV      R7,#4
MOV      R1,#MRD
MOV      R0,#ReadBuf
```

```
;-----
;以下为比较从 EEPROM 中读出的数据和 IC 卡中读出的数
;据是否相同
```

```

;-----
CMPLOOP:
    MOV    A,@R0
    MOV    R6,A
    MOV    A,@R1
    XRL    A,R6
    JNZ    NOEQU
    DJNZ   R7,CMPLOOP
;-----数据相同则断开继电器
SNEQU:
    CLR    SW
    JMP    Read_Insert_Card
NOEQU:
;-----数据不同则闭合继电器
    SETB   SW
    JMP    Read_Insert_Card

$INCLUDE(SLE4442.INC)      ;包含 SLE4442 驱动程序
$INCLUDE(VI2C_24A.inc)     ;包含 I2C 驱动程序
;
    END
    
```

八. 实验程序流程图

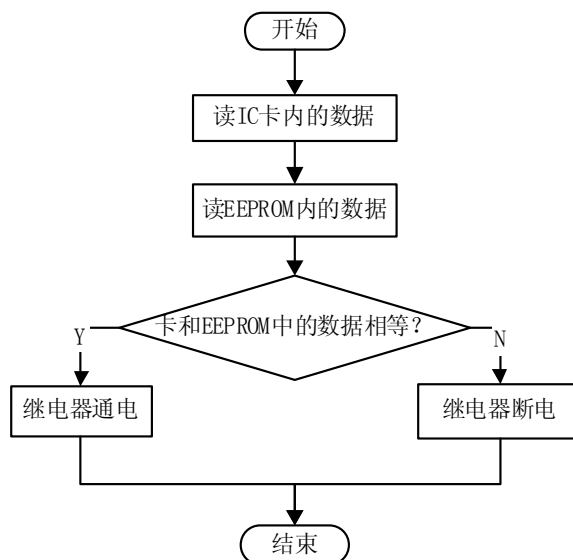


图 3.38 程序流程图

九. 实验思考题

- (1) 如果在实际中要进行身份识别, 本程序还应作哪些完善工作?

实验三十 USB1.1 接口控制演示实验

一. 实验目的

掌握基于 I²C 总线器件 24WC02 EEPROM 的应用；学习使用 PDIUSBD12 USB1.1 接口芯片设计 USB 设备。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
USB D12PARK 模块	一块
USB 连接线	一根

三. 实验内容

1. 编写串行 I²C EEPROM 读写程序；
2. 使用周立功公司提供的 USB51S 库函数编写应用程序，实现设备与 PC 连接及读写 24WC02；

四. 实验要求

1. 掌握串行 I²C EEPROM 应用（参考实验 16）；
2. 掌握 PDIUSBD12 的硬件连接（参阅配套光盘《PDIUSBD12 带并行总线的 USB 接口器件》文档）。如图所示；

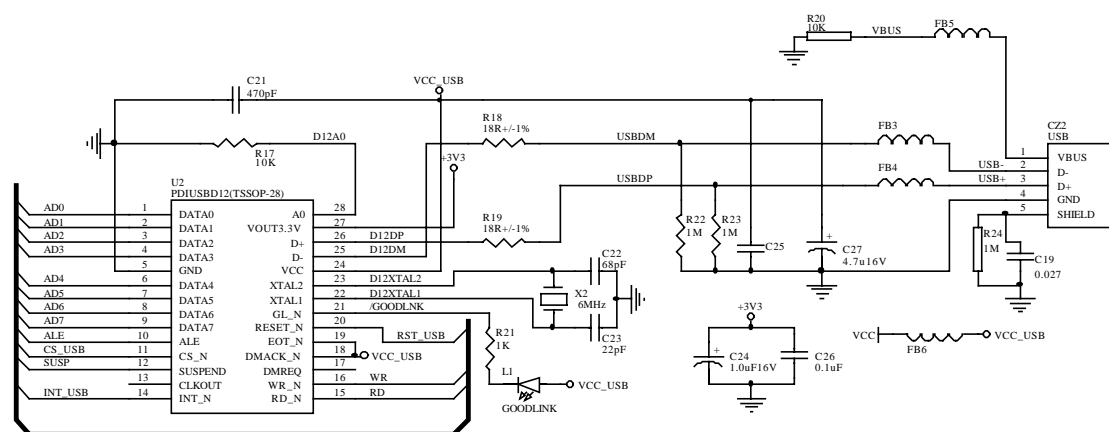


图 3.39 PDIUSBD12 硬件接线原理图

3. 掌握 USB51S 库函数使用，参阅《PDIUSBD12 固件编程与驱动开发》；

五. 实验步骤

1. D12 PARK 插到 A6 区的排针内；
2. 使用导线把 A2 区的 INT1 与 A6 区的 P1_INT 相连；
3. 使用导线把 A2 区的 T0 与 A6 区的 P1_IO2 相连；
4. 使用导线把 A2 区的 T1 与 A6 区的 P1_IO5 相连；
5. 使用导线把 A2 区的 A15 与 A6 区的 P1_CS 相连；

6. 使用导线把 A2 区的 P16、P17 分别与 D5 区的 SCL、SDA 相连;
7. 下载光盘关于 USB1.1 实验 usbeeprom.hex 文件到 DP-51PROC 并运行;
8. 使用 USB 连接线将 D12 PARK 与 PC 机 USB 接口连接;
9. 打开 ZlgEasyD12_DEMO 上位机软件, 读写 DP-51PROC 上的 24WC02。

六. 实验预习要求

阅读本书的 2.8.6 节内容, 理解硬件结构; 阅读《PDIUSB12 固件编程与驱动开发》一书的第 3 章及第八章了解 PIDUSB12 芯片的使用及 USB51S 库函数的使用。

七. 实验参考程序

请参阅《PDIUSB12 固件编程与驱动开发》一书中的第九章(基于 USB EEPROM 编程器); 及配套光盘内关于 USB1.1 实验的单片程序(使用 Keil C51 软件打开 dp-1581.Uv2 工程文件)。

八. 实验思考题

- (1) 设计一个基于 USB1.1 接口, 控制 DP-51PROC 的其它模块;
- (2) 设计一个基于 USB1.1 接口的数据采集器;

实验三十一 CAN-bus 接口控制实验

一. 实验目的

通过调用 CAN 程序库 SJA1000_PELI.LIB 的基本函数,实现实验板上 CAN 节点的初始化以及 CAN 节点的自发自收测试。

二. 验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三. 实验内容

编写一段程序,调用 SJA1000_PELI.LIB 中的函数,实现 CAN 节点的初始化,能够进行数据的自发自收,并能在 LED 上显示相关的信息。

四. 实验要求

学会对 CAN 节点的基本操作,理解实现 CAN 通信的基本流程。

五. 实验步骤

1. 将 CAN-bus PARK 插在到 A6 区中,用导线连接 A6 区的 P1_IO2 到 A2 区的 P10,连 A6 区的 P1_CS 到和 A2 区的 A15。
2. 使用导线把 A2 区的 P16 和 P17 分别于 D5 区的 SCL 和 SDA 相连。使用导线把 D5 区的 RST_L 与 VCC 相连
3. 如果用户采用中断方式,将 A6 区的 P1_INT 接到 A2 区的 INTO 或 INT1 另外改相应的程序即可。
4. 利用 SJA1000_PELI 库编写 CAN 节点的初始化和自发自收程序。
5. 利用 ZLG7290,将 CAN 节点自发自收数据的成功与否显示出来。

六. 实验预习要求

1. 阅读关于 CAN 和 CAN 相关器件的基本介绍,具备 CAN 和 CAN 相关器件的基本知识。
2. 阅读配套光盘中的《SJA1000_PELI 库说明及其使用》文档,了解在如何在程序中应用 SJA1000_PELI 库。

七. 实验参考程序

```
#include <REG52.H>
#include "VIIC_C51.h"           //I2C程序库头文件
#include "Sja1000_peli.h"       //CAN程序库头文件

#define uchar unsigned char
#define ZLG7290 0x70
sbit RESET_PIN=P1^0;

uchar Display_Buffer[5]={0x0d,0x15,0x15,0x10}; //显示GOOD
```



```

uchar    CAN_Baudrate_Filter_Buffer[9]={ 14,0,0,0,0,0xff,0xff,0xff,0xff};
//存放设置波特率和滤波器的数据
uchar    Send_CAN_Info_ID[5]={ 0x88,0x00,0x00,0x00,0x00};

//存放信息帧的数据,符合CAN2.0B
uchar    Send_Data_For_Self[8]={  0x01,0x02,0x03,0x04,
                                0x05,0x06,0x07,0x08};

//存放数据帧的数据
uchar    ScanNum[1];
/*-----
** 函数原型:    void    Delay_ms(uchar j)
** 功能描述:    该函数用于不精确的延时。在12M,6CLK下, 大约延时j*1ms
** 参数说明:    0-255
** 返回值:      无
**-----*/
void    Delay_ms(uchar j)
{
    uchar    k,l;
    for(l=0;l<=j;l++)
    {
        for(k=0;k<=250;k++)
        {
            ;
        }
    }
}

/*-----
** 函数原型:    unsigned char ZLG7290_SendCmd( unsigned char Data1,
                                                unsigned char Data2)
** 功能描述:    发送命令（对子地址7、8）
** 参数说明:    DATA1:    命令1
**              DATA2:    命令2
** 返回值:      0          失败
**              1          成功
**-----*/
unsigned char ZLG7290_SendCmd( unsigned char Data1,
                                unsigned char Data2)
{
    unsigned char Data[2];
    Data[0]=Data1;
    Data[1]=Data2;
    ISendStr(ZLG7290,0x07,Data,2);
    Delay_ms(10);
    return 1;
}

/*-----
** 函数原型:    void    ZLG7290_SendDisBuf( unsigned char *disp_buf,
                                                unsigned char num)
** 功能描述:    向显示缓冲区发送数据
** 参数说明:    *disp_buf    要发送数据的起始地址
**              num          发送个数

```

```

** 返回值:      无
/*-----*/
void ZLG7290_SendDisBuf(unsigned char *disp_buf,unsigned char num)
{
    unsigned char i;
    for(i=0;i<num;i++)
    {
        ZLG7290_SendCmd(0x60+i,*disp_buf);
        disp_buf++;
    }
}

/*-----*/
** 函数原型:      void      SJA1000_Config_Self(void)
** 功能描述:      对SJA1000的在自发自收模式下的初始化配置
** 参数说明:      无
** 返回值:      无
/*-----*/
void SJA1000_Config_Self(void)
{
    SJAEntryResetMode();                //进入复位模式
    WriteSJAREg(REG_CAN_CDR,0xc8);      //选择PeliCAN模式
    SetBitMask(REG_CAN_MOD,AFM_BIT);    //选择单滤波模式
    SetBitMask(REG_CAN_MOD,STM_BIT);    //选择自检测模式
    WriteSJAREgBlock(16,&CAN_Baudrate_Filter_Buffer[1],8);
    //设置验收代码/屏蔽寄存器
    WriteSJAREg(REG_CAN_OCR,0x1a);      //设置输出管脚
    SJASetBaudrateStandard(CAN_Baudrate_Filter_Buffer[0]);
    //设置总线定时器，确定波特率
    SJAQuitResetMode();                //退出复位模式
}

/*-----*/
** 函数原型:      unsigned char  Send_Sja1000_Self(void)
** 功能描述:      把数据传送到SJA1000的接收缓冲区，并进行自发自收的测试
** 参数说明:      无
** 返回值:      0          失败
**              1          成功
/*-----*/
unsigned char Send_Sja1000_Self(void)
{
    uchar i;
    SJA1000_Config_Self();
    WriteSJAREg(REG_CAN_IER,0x03);      //使能接收发送中断标志位
    Delay_ms(100);
    for(i=0;i<2;i++)
    {
        if((ReadSJAREg(REG_CAN_SR)&0x0c)==0x0c)//判断是否可以发送
        {
            WriteSJAREgBlock(16,Send_CAN_Info_ID,5);
            //扩展帧，向发送缓冲区写入5个数据
            WriteSJAREgBlock(21,Send_Data_For_Self,8);
            //扩展帧，向发送缓冲区写入8个数据

```

```

        SetBitMask(REG_CAN_CMR,SRR_BIT);        //使能自发送请求标志位
        return(1);                                //返回发送成功标志
    }
    else
    {
        Delay_ms(200);                            //延时200ms
    }
    return(0);                                    //返回发送失败标志
}

}

/*-----
** 函数原型:    void    main(void)
** 功能描述:    主函数
** 参数说明:    无
** 返回值:      无
/*-----*/
void main()
{
    RESET_PIN=0;
    //将SJA1000的复位线与P1.0相连接
    Delay_ms(1);
    RESET_PIN=1;
    //控制P1.0来实现SJA1000的复位
    SJA_CS_Point=&CAN_SJA_BaseAdr;                //确定SJA1000的基址
    ScanNum[0]=0;                                  //不显示
    Display_Buffer[0]=0x1f;
    ZLG7290_SendDisBuf(Display_Buffer,0);
    ISendStr(ZLG7290,0x0d,ScanNum,1);
    Delay_ms(100);
    if(Send_Sja1000_Self())
    {
        Delay_ms(200);
        if((ReadSJAREg(REG_CAN_IR)==0x03))        //查询接收/发送中断位
        {
            WriteSJAREg(REG_CAN_CMR,4);            //释放接收缓冲区
            ScanNum[0]=3;                          //扫描4位
            Display_Buffer[0]=0x0d;                //显示Good
            Display_Buffer[1]=0x15;
            Display_Buffer[2]=0x15;
            Display_Buffer[3]=0x10;
            ISendStr(ZLG7290,0x0d,ScanNum,1);
            ZLG7290_SendDisBuf(Display_Buffer,4);
        }
        else
        {
            ScanNum[0]=4;                          //扫描5位
            Display_Buffer[0]=0x18;                //显示Error
            Display_Buffer[1]=0x15;
            Display_Buffer[2]=0x18;
            Display_Buffer[3]=0x18;
            Display_Buffer[4]=0x0e;
            ISendStr(ZLG7290,0x0d,ScanNum,1);
        }
    }
}

```

```

        ZLG7290_SendDisBuf(Display_Buffer,5);
    }
}
else
{
    ScanNum[0]=4;                //扫描5位
    Display_Buffer[0]=0x18;       //显示Error
    Display_Buffer[1]=0x15;
    Display_Buffer[2]=0x18;
    Display_Buffer[3]=0x18;
    Display_Buffer[4]=0x0e;
    ISendStr(ZLG7290,0x0d,ScanNum,1);
    ZLG7290_SendDisBuf(Display_Buffer,5);
}
while(1)
{
    ;                            //死循环
}
}

```

在本示范程序中，采用了查询中断标志位的方式来判断自发自收数据是否成功。用户也可以将SJA1000的INT管脚连接到MCU的外部中断管脚上，并使能相应的中断位，在程序中编写中断服务程序来实现同样的功能。采用中断的方式，可以提高系统的实时性。特别是在接收数据的时候，采用中断方式可以在效率和实时性上比采用查询方式得到很大的提高。接收数据采用中断方式的程序流程图如下所示。

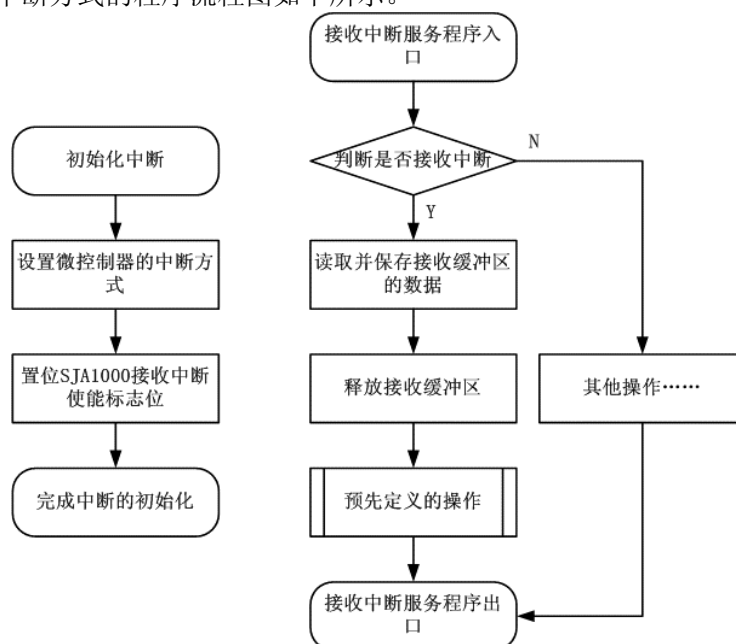


图3.40 程序流程图

八. 实验程序流程图

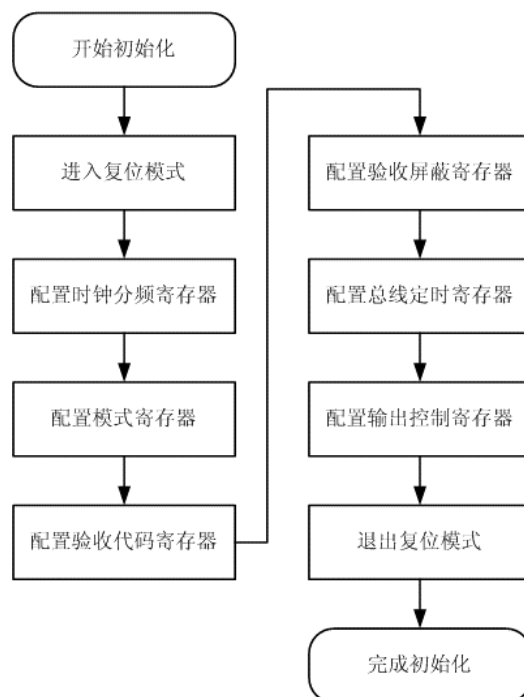


图 3.41 CAN 节点初始化基本流程图

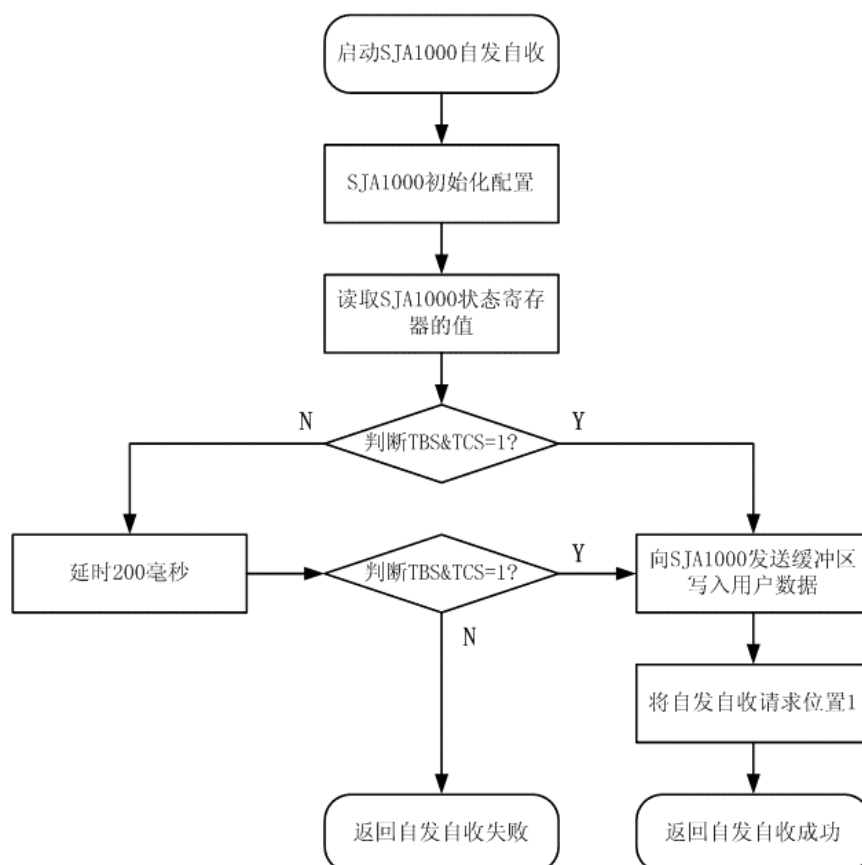


图 3.42 CAN 节点自发自收基本流程图

九. 实验思考题

- (1) 请用户思考一下，如何编写 CAN 总线数据的正常发送程序。
- (2) 请用户再思考一下，如何编写 CAN 总线数据的接收程序。

实验三十二 USB2.0 接口控制演示实验

一. 实验目的

掌握基于 I²C 总线器件 ZLG7290、PCF8563 常用器件的应用；学习使用 ISP 1581 USB2.0 接口芯片设计 USB2.0 设备。

二. 验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
1581 PARK 模块	一块
USB 连接线	一根

三. 实验内容

1. 编写 I²C 总线器件 ZLG7290、PCF8563 器件的应用程序；
2. 了解 USB2.0 接口芯片 ISP1581 单片机固件编写；

四. 实验要求

1. 掌握键盘显示芯片 ZLG7290 的应用（参考实验 16）；
2. 掌握时钟芯片 PCF8563 的应用（参考实验 16）；
3. 掌握 ISP1581 的硬件连接（参阅配套光盘《ISP1581 高速 USB 接口器件》文档），如下图所示；

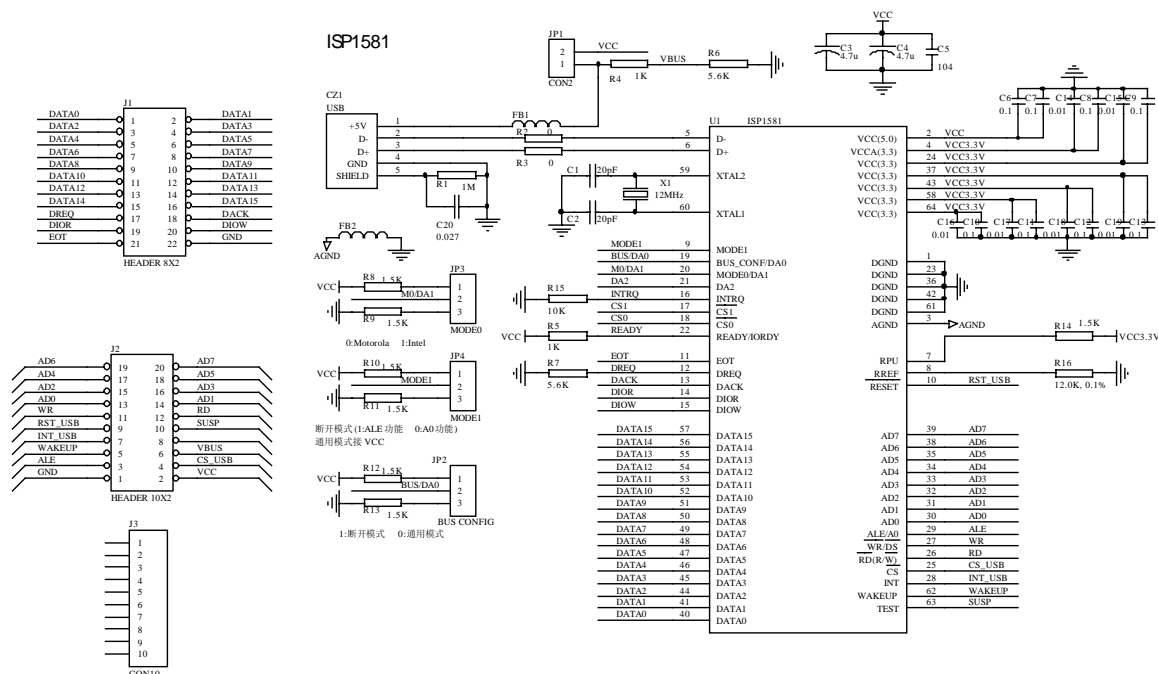


图 3.43 ISP 1581 硬件接线原理图

4. 了解 ISP 1581 单片机固件编程（参阅配套光盘《ISP 1581 编程指南》文档）。

五. 实验步骤

1. 将 1581 PARK 插到 A6 区的排针内 (注: 1581PARK 上的 JP1 断开、JP2 跳到 0、JP3 跳到 1、JP4 跳到 0);
2. 使用导线把 A2 区的 INTO 与 A6 区的 P1_INT 相连;
3. 使用导线把 A2 区的 T0 与 A6 区的 P1_I02 相连;
4. 使用导线把 A2 区的 T1 与 A6 区的 P1_I05 相连;
5. 使用导线把 A2 区的 A15 与 A6 区的 P1_CS1 相连;
6. 使用导线把 A2 区的 P16、P17 分别与 D5 区的 J4 接口的 SCL、SDA 相连;
7. 使用导线把 D5 区的 RST_L 与 VCC 相连;
8. 用 DPFLASH 软件下载光盘中 USB2.0 实验 DP-1581.HEX 文件到 TKSMonitor51 仿真器中并运行;
9. 使用 USB 连接线将 1581 PARK 与 PC 机 USB 接口连接;
10. 打开 TimeAndKey.exe 上位机软件, 可以在“数值”栏填上 8 个或 8 个以内的数字, 然后单击“修改”这时, 八位的数码管将显示用户在电脑上输入的数字;
11. 在上位机软件的“数值”栏下面的“当前时间”上显示的是 PCF8563 时钟芯片上的时间。当用户按动 DP-51PROC 上 S1~S8 的按键时, 对应于程序上的“按键状态”下的对应绿色圆点变成红色。

六. 实验预习要求

阅读本书的 2.8.6 节内容, 理解硬件结构; 复习参考实验 16 有关 ZLG7290 及 PCF8563 内容。阅读配套光盘《ISP1581 高速 USB 接口器件》、《ISP 1581 编程指南》文档了解有关 ISP1581 芯片资料及固件编程。

七. 实验参考程序

请参阅配套光盘内关于 USB2.0 实验的单片机程序。(使用 Keil C51 软件打开 DP-1581.Uv2 工程文件)。

八. 实验思考题

- (1) 设计一个基于 USB2.0 接口, 控制 DP-51PROC 的其它模块;
- (2) 设计一个基于 USB2.0 接口的数据采集器;

实验三十三 基于以太网接口的 TCP/IP 实验

一. 实验目的

进行一次 TCP/IP 的通讯实验，使用户初步了解以太网、IP、ARP、ICMP 等协议。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三. 实验内容

结合以太网通讯模块进行 PC 端的网络设置，单片机端的 IP 设置及 PING 命令的使用。

四. 实验要求

理解 IP 地址与掩码地址之间的关系，理解 MAC 地址在以太网通讯中的作用，掌握 ICMP 协议的作用和通讯过程。有时间可以把 TCP 和 UDP 的实验也一起做。

五. 实验步骤

1. 把以太网模块插到 A6 区中，用导线连接 A6 区的 P1_IO2 到 A2 区的 P10，连 A6 区的 P1_CS 到 A2 区的 A15。
2. 配套的网线是属于交叉线，就是用在双机互连的，用户用该网线把 PC 机与 DP-51PROC 相连。
3. 修改用户的 PC 机 IP 地址和局域网网关为 192.168.0.X (X≠94)，子网掩码为 255.255.255.0。DP-51PROC 的默认 IP 地址和网关地址分别为：192.168.0.94 和 192.168.0.1。使 DP-51PROC 与用户 PC 机处于同一子网内。
4. 使用集成串口调试器的 DPFLASH 软件将该“ne2000.HEX”文件（在光盘的实验程序中）下载到 TKSMonitor51 仿真器。下载完后把 TKSMonitor51 仿真器上的拨动开发拨到 RUN 的地方。（具体操作可以参考本书 2.6 节）
5. 打开 DPFLASH 软件的串口调试器。选择波特率 19200bps，数据位 8，无奇偶校验，1 位停止位。按一下复位键（RST）后会看到从串口输出的信息，如图 3.44。



图 3.44

6. 用户可在串口界面下通过输入命令 `setip xxx.xxx.xxx.xxx` (输入后要加回车键才会得到确认) 来设置单片机的 IP 地址, 使之与用户 PC 机处于同一子网内, 便于调试。(与步骤 3 的设置功效一样, 只是步骤 3 设置 PC 机, 这里是设 DP-51PROC)

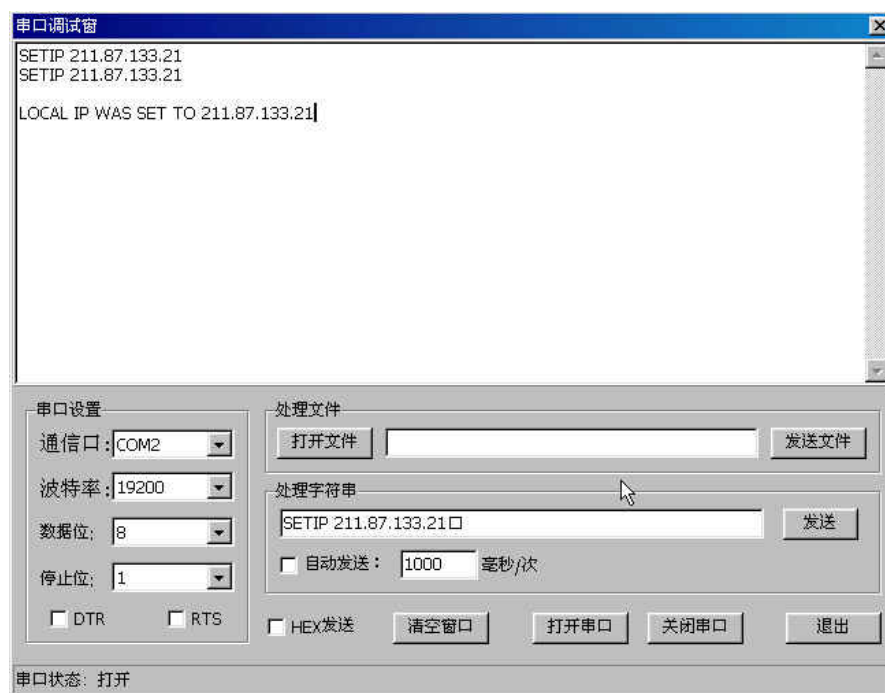


图 3.45

7. 使用 `ping xxx.xxx.xxx.xxx` (输入后要加回车键才会得到确认) 命令可以测试目的主机的可达性。如图示 (这里显示的是 PC 机的 IP 为 192.168.0.55), 当然用户也可以利用 PC 机来 PING DP-51PROC



图 3.46

六. 实验预习要求

认真阅读使用程序中的源码，了解 TCP/IP 协议栈的以太网、IP、ARP、ICMP 等协议的原理和工作过程。

七. 实验思考题

- (1) 请用户思考一下，网关的作用。
- (2) 请用户再思考一下，不在同一子网内的 PC 机与 DP-51PROC 是否可以通讯。

实验三十四 ISD1420 语音模块实验

一. 实验目的

通过本实验掌握 ISD1420 语音模块的工作原理和应用技巧,熟悉语音录放电路模块的设计。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
ISD1420 语音模块	一套

三. 实验内容

1. 连接线路通过实验板上的按键控制 ISD1420 语音模块的录放音功能。
2. 采用电平开关完成语音模块的语音存储分段控制功能。

四. 实验要求

熟悉语音录放电路的工作原理和应用方法。ISD1420 语音模块的典型应用电路如下:

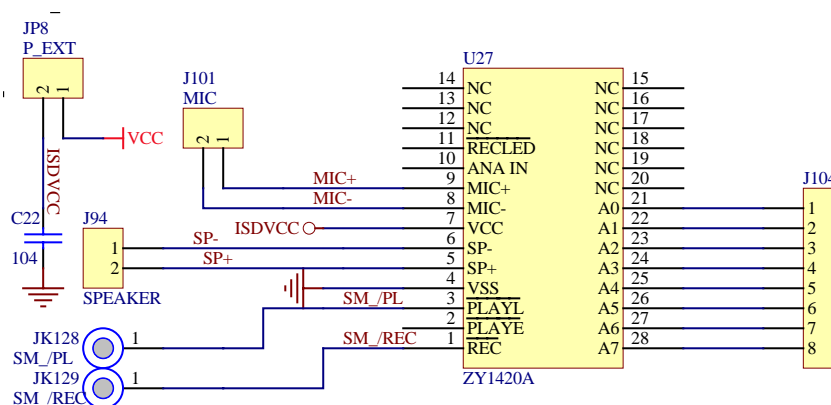


图 3.47 语音模块电路图

五. 实验步骤

1. 安装 B1 区 JP8 接口上的短路帽,将 B1 区 J104 接口与 D1 区的 J54 接口相连。
2. 用导线将 B1 区的 SM_/PL 和 SM_/REC 针脚分别连接到 D1 区的 KEY1 和 KEY2 针,使用 KEY1 键控制录音的播放,用 KEY2 键控制录音功能。
3. 测试语音模块,首先将 SW1~SW8 置 0 状态,按住 KEY2 键开始录音,最大录音 20 秒,释放 KEY2 键录音完毕。然后按下 KEY1 键即开始播放刚才录制的声音信息。
4. 改变 SW1~SW8 的状态,尝试录多段语音信息,然后分段播放。

六. 实验预习要求

认真阅读本节的实验内容,提前做好实验准备工作,认真阅读 ISD1420A 语音模块的使用手册。

七. 实验思考题

试用 ISD1420 语音模块设计一个语音报时时钟。

实验三十五 非接触式 IC 卡读卡模块实验

一. 实验目的

通过本实验学习 ZLG500A 读卡模块的应用方法, 了解射频读卡器在门禁系统、电子交费等领域的应用技术。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
RC500A 读卡模块	一套

三. 实验内容

1. 编写程序, 通过单片机的 I/O 口控制 ZLG500A 模块的用户接口, 实现对射频卡的读写控制。
2. 编写一个读卡程序, 检测射频卡的等待读取。

四. 实验要求

掌握 ZLG500A 模块三线 SPI 总线接口的应用, 初步了解射频读卡的工作原理和应用范围。

五. 实验步骤

1. 参考以下的电路图连接电路接口, 使用导线将 A2 区的 P10、P11、P12、INT0 接线柱与 B2 区的 SCLK、SDATA、MF_RST、SS 接口一一对应连接, 然后连接 ZLG500A 报警输出连接到 LED 指示灯和 B5 区的无源蜂鸣器上。连接 5V 电源输入。
2. 运行编写好的软件程序, 等待读取射频卡。将卡放到天线附近即可听到响声, 说明读卡成功。

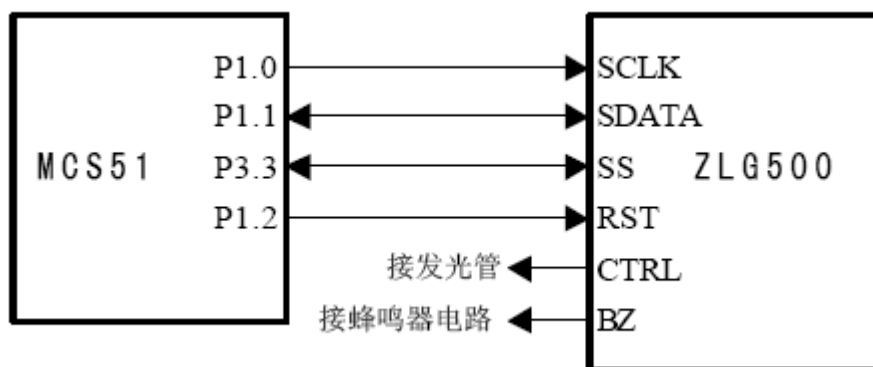


图 3.48 ZLG500A 模块接线图

六. 实验预习要求

整理本节的实验内容, 掌握 RC500A 的基本应用技术, 提前做好实验准备工作。

七. 实验参考程序

```

/*****读卡参考测试程序*****/
#include "zlg500.h"
sbit zlg500_RST=P1^2;
uchar code Nkey_a[6]    = {0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5};
uchar code Nkey_b[6]    = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

main()
{
    uchar  idata  tt[2];
    uchar  idata  card_snr[4];
    uchar  idata  size;
    uchar  idata  bankdata[16];
    long   idata  value=1;
    uchar           i,j;

    zlg500_RST=1;                               //模块复位
    for(i=255;i>0;i--)
        for(j=255;j>0;j--);
    zlg500_RST=0;
    for(i=255;i>0;i--)
        for(j=255;j>0;j--);
    spi_init();
    EA=1;
    i=mifs_config();                             //模块配置
    i=mifs_get_info(bankdata);                   //读信息
    i=mifs_clr_control_bit();
    i=mifs_set_control_bit();
    while(1)
    {
        while(mifs_request(IDLE,tt)!=0);        //请求
        if(mifs_anticoll(0,card_snr)!=0)        continue;//防碰撞
        if(mifs_select(card_snr,&size)!=0)      continue;//选择
        if(mifs_authKey(KEYA,5,Nkey_b)!=0)      continue;//证实

        bankdata[0]=0x10;
        bankdata[4]=~0x10;
        bankdata[8]=0x10;
        for(i=1;i<4;i++)
        {
            bankdata[i]=0x00;
            bankdata[4+i]=0xff;
            bankdata[8+i]=0x00;
        }
    }
}

```

```

    }
    bankdata[12]=0x14;
    bankdata[13]=~0x14;
    bankdata[14]=0x14;
    bankdata[15]=~0x14;
    if(mifs_write(20,bankdata)!=0) continue;    //写一个值块

    if(mifs_check_write(card_snr,KEYA,20,bankdata)!=0)
        continue;                            //检查写

    if(mifs_read(20,bankdata)!=0) continue;    //读回该块数据

    if(mifs_restore(20)!=0) continue;          //恢复 20 块数据
    if(mifs_transfer(21)!=0)                   //传送到 21
    {
        i=0;
        continue;
    }
    if(mifs_value(0xc0,20,&value,21)!=0)
    {
        i=0;
        continue;
    }
    if(mifs_read(21,bankdata)!=0) continue;    //读出
    mifs_halt();                               //使模块进入 HALT 状态

    if(mifs_write_E2(0x30,16,bankdata)!=0) continue;
    mifs_clr_control_bit();
    mifs_set_control_bit();
    for(i=255;i>0;i--)
        for(j=255;j>0;j--);
    mifs_buzzer(198,20);                      //输出蜂鸣器
}
}

```

八. 实验思考题

采用 ZLG500A 模块，设计一个简单的多用户识别门禁系统

实验三十六 并行模数转换实验

一、实验目的

熟悉 A/D 转换的工作原理，学习使用并行模数转换芯片 ADC0809 进行电压信号的采集和数据处理。

二、实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
数字万用表	一台
ADC0809 PACK 模块（选配件）	一套

三、实验内容

通过片外总线方式访问并行模拟数字转换器芯片 ADC0809，掌握模拟电压的通用采集方法。

四、实验要求

理解掌握 ADC0809 的 A/D 转换原理和并行 A/D 转换器接口的编程方法，学会使用 ADC0809 并行模数转换器实现电压信号采集的方案设计。

五、实验步骤

1. 将 ADC0809 PACK 模块（选配件）插入 PARK2 区；
2. 将 D2 区 1K 电位器和 10K 电位器的左端金属孔通过导线连接到该区的 GND 金属孔，而右端金属孔通过导线连接到该区的 VCC 金属孔；
3. 将 D2 区 1K 电位器的中间金属孔连接到 A7 区的 P2_IO2 金属孔，而 D2 区 10K 电位器的中间金属孔连接到 A7 区的 P2_IO1 金属孔；
4. 将 A7 区的 P2_IO3~P2_IO5 分别连接到 A2 区的 A2~A0；
5. 将 A7 区的 P2_CS 连接到 A2 区的 A15；
6. 运行编写好的软件程序，每次跑到断点就会停止，此时观察转换的结果和用数字万用表测量的结果相比较是否正确（所需观察的存储单元或者变量在程序中依照注释执行）；
7. 改变 1K 电位器和/或 10K 电位器的旋钮位置，用数字万用表测量中间金属孔的电压，再次运行程序至断点处，观察转换的结果是否正确。

六、实验预习要求

认真预习本节实验内容，按照实验的要求提前做好实验准备工作，认真阅读 ADC0809 的数据手册。

七、实验参考程序

1. 汇编语言程序：

;使用该程序前请先按照实验指导手册连接好连线，
;运行程序至断点，观察 30H 单元和 31H 单元转换的十六进制
;数据换算成实际电压值是否与万用表的测量结果相等？

;扭动电位器，改变 IN0 或/和 IN1 模拟输入电压，再次运行程序至
;断点，观察 30H 或/和 31H 单元的数值是否随之改变，改变的是否正确？

```

        ORG      8000H
        LJMP     MAIN
        ORG      8100H
MAIN:    MOV      SP,#70H
        MOV      R1,#30H      ;置数据区首地址，用于存放 A/D 转换结果
        MOV      DPTR,#7FF8H ;P2.7=0，且指向通道 0
        MOV      R7,#02H      ;置通道数
LOOP:    MOVX     @DPTR,A      ;启动 A/D 转换
        MOV      R6,#20H      ;软件延时，等待转换结束
DELAY:   NOP
        NOP
        NOP
        DJNZ     R6,DELAY
        MOVX     A,@DPTR      ;读取转换结果
        MOV      @R1,A        ;转存
        INC      DPTR          ;指向下一个通道
        INC      R1            ;修改数据区指针
        DJNZ     R7,LOOP       ;IN0、IN1 两个通道全采样完了吗？
        LJMP     MAIN
        END
    
```

2. C51 程序:

//使用该程序前请先按照实验指导手册连接好连线，
//运行程序至断点，观察 result0 和 result1 的转换结果
//浮点数值是否与数字万用表的测量结果相等？
//扭动电位器，改变 IN0 或/和 IN1 模拟输入电压，再次运行程序至断点，
//观察 result0 或/和 result1 的转换结果浮点数值是否随之改变，改变的是否正确？

```

#include <reg52.h>
#include <intrins.h>
#include <absacc.h>
    
```

```

#define PIN0    XBYTE[0x7ff8]
#define PIN1    XBYTE[0x7ff9]
    
```

```

typedef unsigned char byte;
typedef unsigned int  word;
    
```

```

void main(void)
{
    float  result0,result1;          //两个通道 A/D 转换结果的存储变量
    float  result0_reg,result1_reg;  //浮点数据输出存储
    byte   i,j;
    
```



```
while(1)
{
    for(i=0;i<2;i++)          //对 IN0、IN1 两个通道进行采样
    {
        if(!i)
            PIN0=0xff;        //启动通道 0 的 A/D 转换
        else
            PIN1=0xff;        //启动通道 1 的 A/D 转换
        for(j=0;j<0x50;j++);  //延时，等待转换的完成
        if(!i)
            result0_reg=PIN0;  //读取通道 0 的转换结果
        else
            result1_reg=PIN1;  //读取通道 1 的转换结果
    }
    result0=result0_reg*5/256;
    result1=result1_reg*5/256;
}
}
```

八、实验思考

1. 本实验采用了延时等待的方式等待 ADC0809 转换结束，请用户改用中断方式做一次这个实验，并说出这个实验采用中断方式与延时等待方式相比有哪些优点；
2. 请将 D5 区上的 ZLG7290 芯片和数码管与本实验相结合，设计出一个数字电压表，并与商品化的数字电压表测量值比较。

实验三十七 并行数模转换实验

一、实验目的

熟悉 D/A 转换的工作原理，学习使用并行数模转换芯片 DAC0832 进行数字信号到模拟信号的转换过程。

二、实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
示波器	一台
DAC0832 PACK 模块（选配件）	一套

三、实验内容

通过片外总线方式访问并行数模转换器芯片 DAC0832，掌握数字信号到模拟电压的转换方法。

四、实验要求

理解掌握 DAC0832 的 D/A 转换原理和并行 D/A 转换器接口的编程方法，学会使用 DAC0832 并行模数转换器实现电压信号采集的方案设计。

五、实验步骤

1. 将 DAC0832 PACK 模块（选配件）插入 PARK2 区；
2. 将模块上的 JP1 跳线帽跳至右侧的 VCC 处；
3. 将 A7 区的 P2_CS 连接到 A2 区的 A15；
4. 将 A7 区的 P2_IO2、P2_IO5 和 P2_INT 分别接入 C4 区的 A-、A+ 和 AOUT；
5. 将 C4 区的 V+ 和 V- 分别接至 C1 区的 +12V 和 -12V；
6. 运行编写好的软件程序，使用示波器观察 C4 区 AOUT 处的波形是否为锯齿波。

六、实验预习要求

认真预习本节实验内容，按照实验的要求提前做好实验准备工作，认真阅读 DAC0832 的数据手册。

七、实验参考程序

1. 汇编语言程序：

```
ORG    8000H
LJMP   MAIN
ORG    8100H
MAIN:  MOV    SP,#70H
        MOV    DPTR,#7FFFH
        MOV    A,#0FFH
```

```
LOOP:  MOVX    @DPTR,A
        DEC     A
        LJMP    LOOP
        END
```

2. C51 程序:

```
#include <reg52.h>
#include <absacc.h>
#define PA XBYTE[0x7fff]
typedef unsigned char byte;
```

```
void main(void)
{
    byte a;
    while(1)
    {
        for(a=255;a>0;a--)
        {
            PA=a;
        }
    }
}
```

八、实验思考

请改变上面的程序，使之输出三角波。

实验三十八 8155 并口扩展实验

一、实验目的

熟悉并口扩展芯片 8155 的内部结构,学会使用 8155 扩展并口和片外 RAM 和 14 位减法定时/计数器。

二、实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
8155 PACK 模块(选配件)	一套

三、实验内容

通过片外总线方式访问并口扩展芯片 8155,并使它的 PA 口闪烁点亮 8 个 LED。

四、实验要求

理解掌握并口扩展芯片 8155 的原理和该芯片的编程方法,学会使用并口扩展芯片 8155H 进行 8 位并行接口的扩展。

五、实验步骤

1. 将 8155 PACK 模块(选配件)插入 PARK2 区;
2. 使用导线连接 A7 区的 P2_IO2、P2_IO5 和 P2_CS 分别至 A2 区的 A8、P10 和 A15;
3. 使用排线连接 8155 PACK 模块上的 J1 接口(PA 口)至 D1 区的 J52 接口;
4. 运行编写好的程序,观察 8 个 LED 的点亮情况。

六、实验预习要求

认真预习本节实验内容,按照实验的要求提前做好实验准备工作,认真阅读 8155 的数据手册。

七、实验参考程序

1. 汇编语言程序:

```
ORG    8000H
LJMP   MAIN
ORG    8100H
MAIN:  MOV    SP,#60H
      SETB   P1.0
      NOP
      NOP
      NOP
      NOP
      NOP
      CLR    P1.0
```

```

        MOV     DPTR,#7F00H
        MOV     A,#01H
        MOVX    @DPTR,A
        INC     DPTR
        DEC     A
        MOVX    @DPTR,A
LOOP:   LCALL   DELAY
        CPL     A
        MOVX    @DPTR,A
        LJMP    LOOP
DELAY:  MOV     R7,#0FFH
DEL1:  MOV     R6,#0FFH
        DJNZ    R6,$
        DJNZ    R7,DEL1
        RET
        END
    
```

2. C51 程序:

```

#include <reg52.h>
#include <absacc.h>
#include <intrins.h>
#define _Nop() _nop_()
#define PAC XBYTE[0x7f00]
#define PA XBYTE[0x7f01]
    
```

```

typedef unsigned char byte;
sbit RST=P1^0;
    
```

```

void ini_cpu(void)
{
    RST=1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    RST=0;
    PAC=0x01;
}
    
```

```

void delay(void)
{
    byte m,n;
    for(m=0;m<255;m++)
    
```

```
        for(n=0;n<255;n++);  
    }
```

```
void main(void)  
{  
    ini_cpu();  
    while(1)  
    {  
        PA=0x00;  
        delay();  
        PA=0xff;  
        delay();  
    }  
}
```

八、实验思考

1. 请练习使用 8155 的扩展 RAM 功能;
2. 请练习使用 8155 的定时/计数器功能。

实验三十九 8255 并口扩展实验

一、实验目的

熟悉并口扩展芯片 8255 的内部结构, 学会使用 8255 扩展并口。

二、实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
8255 PACK 模块 (选配件)	一套

三、实验内容

通过片外总线方式访问并口扩展芯片 8255, 并使它的 PA 口闪烁点亮 8 个 LED。

四、实验要求

理解掌握并口扩展芯片 8255 的原理和该芯片的编程方法, 学会使用并口扩展芯片 8255 进行 8 位并行接口的扩展。

五、实验步骤

1. 将 8255 PACK 模块 (选配件) 插入 PARK2 区;
2. 使用导线连接 A7 区的 P2_IO1、P2_IO2、P2_IO5 和 P2_CS 分别至 A2 区的 A8、A9、P10 和 A15;
3. 使用排线连接 8255 PACK 模块上的 J1 接口 (PA 口) 至 D1 区的 J52 接口;
4. 运行编写好的程序, 观察 8 个 LED 的点亮情况。

六、实验预习要求

认真预习本节实验内容, 按照实验的要求提前做好实验准备工作, 认真阅读 8255 的数据手册。

七、实验参考程序

1. 汇编语言程序:

```
ORG    8000H
LJMP   MAIN
ORG    8100H
MAIN:  MOV    SP,#60H
      SETB   P1.0
      NOP
      NOP
      NOP
      NOP
      NOP
      CLR    P1.0
      MOV    DPTR,#7FFFH
```

```

        MOV     A,#80H
        MOVX    @DPTR,A
        MOV     DPTR,#7CFFH
        MOV     A,#00H
        MOVX    @DPTR,A
LOOP:   LCALL   DELAY
        CPL     A
        MOVX    @DPTR,A
        LJMP    LOOP
DELAY:  MOV     R7,#0FFH
DEL1:  MOV     R6,#0FFH
        DJNZ    R6,$
        DJNZ    R7,DEL1
        RET
        END
    
```

2. C51 程序:

```

#include <reg52.h>
#include <absacc.h>
#include <intrins.h>
#define _Nop() _nop_()
#define PAC XBYTE[0x7fff]
#define PA XBYTE[0x7cff]
    
```

```

typedef unsigned char byte;
sbit RST=P1^0;
    
```

```

void ini_cpu(void)
{
    RST=1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    RST=0;
    PAC=0x80;
}
    
```

```

void delay(void)
{
    byte m,n;
    for(m=0;m<255;m++)
        for(n=0;n<255;n++);
}
    
```



```
}
```

```
void main(void)
{
    ini_cpu();
    while(1)
    {
        PA=0x00;
        delay();
        PA=0xff;
        delay();
    }
}
```

第 4 章 Small RTOS51 的应用

Small RTOS51 是一个源代码公开的实时操作系统，它有可移植、占先式、多任务、RAM 需求小等特点。用户如果要详细了解 Small RTOS51 可以参考北京航空航天大学出版社出版的《嵌入式实时操作系统 Small RTOS51 原理及应用》一书，源码可以在周立功单片机网站（www.zlgmcu.com）下载，在这我们只是针对在 DP-51PROC 上的应用进行叙述。

4.1 编写 Small RTOS51 的驱动程序

4.1.1 函数的可重入

我们在编写一个驱动程序之前要判断该函数是否为可重入函数。如果一个函数可能被多个任务和/或中断同时调用，那么该函数就必须是可重入的，而驱动程序一般要求可被多个任务/或中断同时调用，这就需要可重入了。一个函数具有可重入性的条件是：

- 1) 不使用任何共享资源（全局变量、共享外设、内部设备等）；
- 2) 如果必须使用一些共享资源，可在使用共享资源前关中断，使用完共享资源后开中断；
- 3) 如果必须使用一些共享资源，而且使用时又不能关中断，则任何使用这个共享资源的任务在使用共享资源前申请相应信号量，得到允许后再使用共享资源，使用完共享资源后释放信号量。

在 ANSI C 的标准实现中，以上条件比较好实现，但在 Keil C51 中就比较麻烦，最主要的是判断函数是否有局部变量分配在内存中。如何判断一个函数是否有局部变量分配在内存中？方法也不难。我们假设文件 file1.c 中有一个函数 Fuction，首先，让编译器将文件 file1.c 编译成汇编文件 file.src（在 Uv2 中，在相应的文件上点鼠标右键，选择 Options For File 'file1.c' 在 Properties 页选上 Assemble SRC File。注意是实勾非虚勾。然后编译工程即可）。打开 file.src，查找 ?DT? Fuction? FILE1、?PD? Fuction? FILE1 和 ?XD? Fuction? FILE1（如果 Fuction 有参数则查找 ?DT? _Fuction? FILE1、?PD? _Fuction? FILE1 和 ?XD? _Fuction? FILE1），看这些数据段下是否定义有变量，如果没有，恭喜您，Fuction 的所有局部变量都分配到寄存器中，可以按照一般的方法让函数具备重入性。如果其中任意一个数据段定义有变量，很不幸，您只有另想办法了。特别的，如果参数在这些数据段中定义，即使不将该函数与任何函数进行覆盖分析，在函数中使用 2)、3) 的方法都不能使函数具有重入性。这是因为在函数执行第一条语句前已经把参数复制到这些变量中，2)、3) 的方法的前提已经不存在了。

4.1.2 驱动程序的编写方法

1. 使用任务编写驱动程序

有一些设备需要 CPU 周期地为其服务，典型的是扫描显示和扫描键盘。可以给它们分配相应的任务，与用户任务一起调度。这样，就可以使用任何任务间通信的方法实现驱动程序。

2. 使用消息队列编写驱动程序

有一些设备具有自己的中断，典型的是串口输出。可以利用消息队列将用户任务需要的服务通过消息队列排队、缓冲起来，利用中断功能依次服务。这样，一般不需要考虑重入这个头疼的问题。

3. 使用信号量编写驱动程序

有一些设备既不需要 CPU 的周期服务，又不具有自己的中断，典型的是全局变量、模拟 I²C 总线等。假如该设备速度又比较慢，就需要用信号量来独占该设备了。这时，就不能将驱动程序编写成一个函数。这是因为函数调用 OSSemPend 后，编译器认为所有寄存器已经发生变化，它不会用这些寄存器保存有用的数据。而参数肯定是有用数据，必将分配到内存中。这就不可能使其具有重入性。替代的方法是按照一般的方法编写一个函数，这个函数只是对这个设备进行操作。然后定义一个宏，在宏中申请信号量，调用这个函数并释放信号量。这样，只要保证那个函数的局部变量全部分配到寄存器中就可以了。

4. 复合方法

有一些驱动程序比较复杂（例如 TCP/IP 通讯协议），可以结合两种或两种以上的方法实现。下面我们就举一些例子来说明，驱动程序的编写。

4.2 16×2 LCD 模块的驱动

4.2.1 点阵字符型 LCD-TC1602A

点阵字符型液晶显示器是专门用于显示数字、字母、图形符号及少量自定义符号的显示器。由于其具有功耗低、体积小、重量轻、超薄等优点，自问世以来 LCD 就得到了广泛应用。字符型液晶显示器模块在国际上已经规范化，在市场上内核为 HD44780 的较常见（可以参考配套光盘上的数据手册）。本章以 TC1602A 型 LCD 为例介绍其驱动程序的编写方法。

1. TC1602A 液晶显示器与 DP-51PROC 实验仪的连接

DP-51PROC 实验仪上有一标准的 LCD 液晶显示器接口 J106，标注为 LCD1602。它与 P87C52X2 以总线方式连接，其硬件连接如图 2.57 所示。

2. 驱动程序的使用

本驱动程序可以在没有 Small RTOS51 的情况下使用。此时，要使用本驱动程序只需要配置设置读写液晶模块 LCD1602 的数据、命令、状态的方法。定义它们的例子见程序清单 4.1。因为在驱动程序的主文件 lcd1602.c 仅包含一个文件 config.h，所以用户必须把它们放在文件 config.h 中。如果用户单独使用 lcd1602.c，还要在 config.h 包含 lcd1602.h 文件和其它必须的文件如 reg51 等；并定义宏 TRUE、FALSE 和与编译器无关的数据类型。在使用 Small RTOS51 的情况下，如果用户只有一个任务使用液晶模块 LCD1602 总线，用户只要在 config.h 定义这些方法就可以了。如果用户不止一个任务需

要访问液晶模块 LCD1602，则驱动程序需要使用信号量保证各个任务对液晶模块 LCD1602 的互斥操作。这时，需要将宏 LCD1602_SEM 定义为分配给液晶模块 LCD1602 驱动程序的信号量的索引，并在使用驱动程序前建立这个信号量。

在使用液晶模块 LCD1602 驱动程序前应该调用函数 Lcd1602Init() 初始化液晶模块。单独使用或单任务使用本驱动程序时，使用函数 Lcd1602DispStr() 在屏幕指定位置显示字符串，使用函数 Lcd1602Clr() 清除指定行。如果有特殊字符需要写入液晶模块，则可以调用函数 Lcd1602LoadC()。如果有多个任务需要对使用本驱动程序，则是分别调用宏 OSLcd1602DispStr()、OSLcd1602Clr() 和 OSLcd1602LoadC()。

当有多个任务需要对液晶模块 LCD1602 操作时，还要注意驱动程序的重入问题。如果用户不是在 Keil C51 中使用 Small RTOS51，可能不需要关心这个问题。但在 Small RTOS51 中，因为液晶驱动程序使用了通用指针，导致函数 Lcd1602DispStr() 和 Lcd1602LoadC() 不可重入。幸好这个问题并不严重，只要禁止所有使用了液晶的任务与 Lcd1602DispStr() 和 Lcd1602LoadC() 进行覆盖分析就对程序没有影响了。这是因为使用了信号量使各个任务互斥调用函数 Lcd1602DispStr() 和 Lcd1602LoadC()。如果只有一个任务对器件写，则不需要禁止对它们进行覆盖分析。

程序清单 4.1 DP-51PROC 中读写液晶模块 LCD1602 的数据、命令、状态的方法

```
/* 定义 LCD1602 操作地址 */
#define LCD1602_WR      XBYTE[0x2001]      /* 写数据操作地址 */
#define LCD1602_RD      XBYTE[0x2002]      /* 读状态操作地址 */
#define LCD1602_WC      XBYTE[0x2000]      /* 写命令操作地址 */
//写命令
#define LCD1602_SEND_COMMAND(a)
    LCD1602_WC = a;                          /* 写命令 */

//写数据
#define LCD1602_SEND_DATA(a)
    LCD1602_WR = a;                          /* 写数据 */

#ifdef IN_LCD1602
//返回状态
uint8 LCD1602_GET_FLAG(void)
{
    return (LCD1602_RD);                      /* 返回液晶状态 */
}
#endif
```

3. 对 TC1602A 操作的基本函数

1) 等待 TC1602A 操作完成

液晶模块 TC1602A 的控制器 HD44780 速度较慢，每次进行读写操作时，应首先检测上次操作是否完成，或在每次读写操作后延时 1ms 等待读写完成。这是通过调用函数 Lcd1602Delay() 来完成的。程序 Lcd1602Delay() 的代码见程序清单 4.2。

程序清单 4.2 等待 TC1602A 空闲

```

void Lcd1602Delay(void)
{
    uint8 i;

    i = 100;
    do
    {
        if ((LCD1602_GET_FLAG() & 0x80) == 0)
        {
            break;
        }
    } while (--i != 0);
}
    
```

(1)

(2)

(3)

(4)

程序首先设置循环次数（程序清单 4.2(1)），然后在循环中检测液晶模块是否空闲（程序清单 4.2(2)）。如果空闲，函数结束。设置循环上限目的是为了避免液晶损坏而使程序进入无限循环。

2) 向 TC1602A 发送命令

驱动程序使用函数 Lcd1602SendComm()向液晶模块 TC1602A 发送命令，它的唯一参数为将要发送的命令字。函数 Lcd1602SendComm()代码见程序清单 4.3。代码很简单，不作介绍。

程序清单 4.3 向 TC1602A 发送命令

```

void Lcd1602SendComm(uint8 Command)
{
    Lcd1602Delay();
    LCD1602_SEND_COMMAND(Command);
}
    
```

/* 等待任务 lcd1602 空闲 */

/* 写命令字 */

3) 向 TC1602A 发送数据

驱动程序使用函数 Lcd1602SendData ()向液晶模块 TC1602A 发送数据，它的唯一参数为将要发送的数据。函数 Lcd1602SendData ()代码见程序清单 4.4。代码很简单，不作介绍。

程序清单 4.4 向 TLC1602A 发送数据

```

void Lcd1602SendData(uint8 Data)
{
    Lcd1602Delay();
    LCD1602_SEND_DATA(Data);
}
    
```

/* 等待任务 lcd1602 空闲 */

/* 写数据 */

4. 初始化 TC1602A 液晶显示器

在使用 TC1602A 液晶显示器前必须对它进行初始化，这是通过调用函数 Lcd1602Init()

实现，其代码见程序清单 4.5。

程序清单 4.5 初始化 TC1602A

```
void Lcd1602Init(void)
{
    Lcd1602SendComm(LCD1602_MODE);           (1)
    Lcd1602SendComm(LCD1602_NO_FLASH);       (2)
    Lcd1602SendComm(LCD1602_NO_SHIFT);       (3)
    Lcd1602SendComm(LCD1602_SH);             (4)
    Lcd1602Clr(1);                            (5)
    Lcd1602Clr(2);                            (6)
}
```

程序首先设置液晶模块控制器 HD44780 的工作模式（程序清单 4.5(1)），其中 LCD1602_MODE 的值在文件 lcd1602.h 中定义，为 0x3c。从前面介绍可知，这是把 HD44780 设置为 8 位总线、两行显示、5*10 点阵字体。然后打开显示（程序清单 4.5(2)），其中 LCD1602_NO_FLASH 的值在文件 lcd1602.h 中定义，为 0x0c。从前面介绍可知，这是使液晶开始显示、不显示光标、光标不闪烁。接着设置液晶模块的输入方式（程序清单 4.5(3)），其中 LCD1602_NO_SHIFT 的值在文件 lcd1602.h 中定义，为 0x06。从前面介绍可知，这使模块数据输入为增量方式，显示内容不移动（光标移动）。接下来设置光标位移方式（程序清单 4.5(4)）；其中 LCD1602_SH 的值在文件 lcd1602.h 中定义，为 0x14。从前面介绍可知，这是使显示一个字符时光标左移，并且光标在下一个要显示的字符的位置。最后是清屏（程序清单 4.5(5)、(6)）。

4. 清除指定行

如果有多个任务需要操作液晶模块 TC1602A，则使用 OSLcd1602Clr() 清除显示模块的某一行。如果仅一个任务需要操作操作液晶模块 TC1602A，则使用 Lcd1602Clr() 清除显示模块的某一行。OSLcd1602Clr() 是一个宏，代码见程序清单 4.6。

程序清单 4.6 多任务中清除指定行

```
#define OSLcd1602Clr(y)
{
    OSSemPend(LCD1602_SEM, 0);               (1)
    Lcd1602Clr(y);                           (2)
    OSSemPost(LCD1602_SEM);                  (3)
}
```

程序通过在液晶模块 TC1602A 上清除指定行之前等待信号量（程序清单 4.6(1)）和在液晶模块 TC1602A 上清除指定行之后发送信号量（程序清单 4.6(3)）来实现对器件的互斥操作。这样做的原因可以参见 4.1 节。在宏中调用函数 Lcd1602Clr() 在液晶模块 TC1602A 清除指定行。而函数 Lcd1602Clr() 就是单任务情况下在液晶模块 TC1602A 清除指定行的函数，所以两者的参数相同。

函数 Lcd1602Clr() 的代码见程序清单 4.7。函数 Lcd1602Clr() 的流程图见图 4.1。函数 Lcd1602Clr() 有唯一参数指示需要清除的行。

程序清单 4.7 单任务中清除指定行

```

void Lcd1602Clr(uint8 y)
{
    uint8 i;

    i = 0; (1)
    if (y == 1) (2)
    {
        Lcd1602SendComm(LCD1602_LINE1); (3)
        i = 16; (4)
    }
    else if (y == 2) (5)
    {
        Lcd1602SendComm(LCD1602_LINE2); (6)
        i = 16; (7)
    }
    if (i != 0) (8)
    {
        do
        {
            Lcd1602SendDate(' '); (9)
        } while (--i != 0); (10)
    }
}

```

函数 Lcd1602Clr() 首先要根据清除的行号设置相应的行显示首地址（程序清单 4.7(3)、(6)）。LCD1602_LINE1 和 LCD1602_LINE2 的值在文件 lcd1602.h 中定义，分别为 0x80 和 0xc0，为各行的显示首地址+0x80（0x80 为设置显示地址命令）。然后函数 Lcd1602Clr() 判断行号是否有效（程序清单 4.7(8)）。这里利用了变量 i 作为标志来判断。变量 i 同时也存储需要清除的字符的个数。真正的清除行是通过显示 16（一行的字符数）个空格来实现的（程序清单 4.7(9)、(10)）。

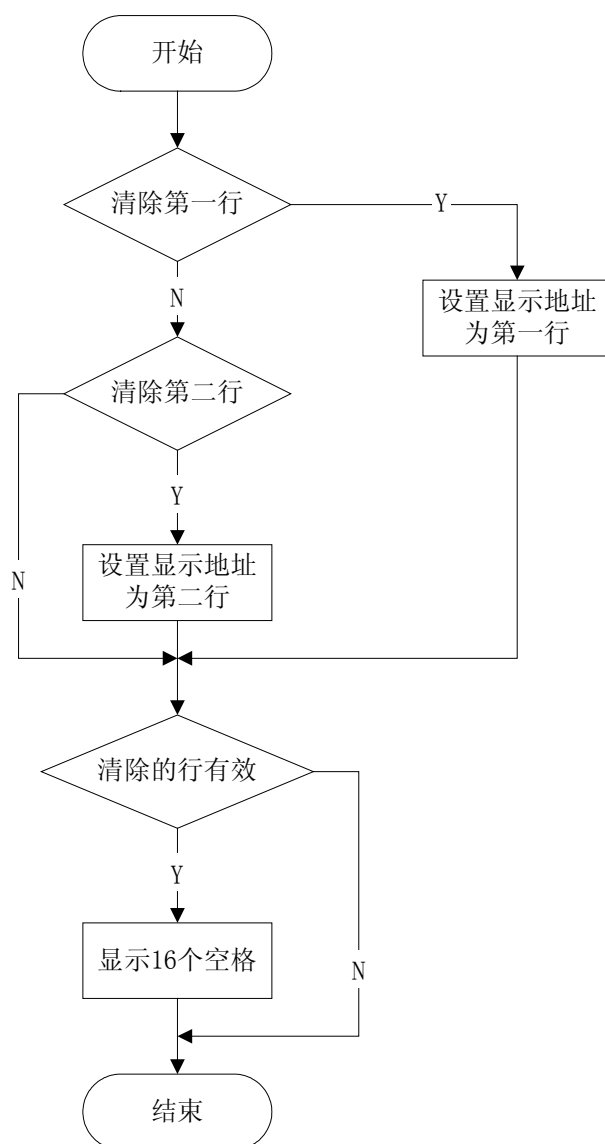


图 4.1 单任务清除指定行流程图

6. 在指定位置显示字符串

如果有多个任务需要操作液晶模块 TC1602A, 则使用 `OSLcd1602DispStr()` 来显示一个字符串。如果仅一个任务需要操作液晶模块 TC1602A, 则使用 `Lcd1602DispStr()` 来显示一个字符串。`OS Lcd1602DispStr()` 是一个宏, 代码见程序清单 4.8。

程序清单 4.8 多任务中在指定位置显示字符串

```

#define OSLcd1602DispStr(x, y, Data)
{
    OSSemPend(LCD1602_SEM, 0);                (1)
    Lcd1602DispStr((x), (y), (Data));          (2)
    OSSemPost(LCD1602_SEM);                   (3)
}
  
```


程序通过在液晶模块 TC1602A 上显示字符串行之前等待信号量（程序清单 4.8(1)）和在液晶模块 TC1602A 上显示字符串之后发送信号量（程序清单 4.8(3)）来实现对器件的互斥操作。这样做的原因可以参见前面的叙述。在宏中调用函数 Lcd1602DispStr() 在液晶模块 TC1602A 上显示字符串。而函数 Lcd1602DispStr() 就是单任务情况下在液晶模块 TC1602A 显示字符串的函数，所以两者的参数相同。

函数 Lcd1602DispStr() 的代码见程序清单 4.4。函数 Lcd1602DispStr() 的流程图见图 4.2，该图作了简化。函数 Lcd1602DispStr() 的参数中 x, y 指示字符串开始显示的位置坐标，其中液晶模块 TC1602A 的左上角坐标定义为 1,1。而参数 Data 指向将要显示的字符串（以 '\0' 作为结束标志）。该函数会自动换行，即当第一行显示不完整个字符串则从第二行开始处继续显示；但如果第二行显示不完则剩余的字符不再显示。

程序清单 4.9 单任务中在指定位置显示字符串

```
void Lcd1602DispStr(uint8 x, uint8 y, char *Data)
{
    if (y == 1)                                     (1)
    {
        if (x < (16 + 1))                           (2)
        {
            Lcd1602SendComm(LCD1602_LINE1 - 1 + x); (3)
            for( ; x < (16 + 1) && *Data != '\0'; x++) (4)
            {
                Lcd1602SendDate(*Data++);           (5)
            }
            if (*Data != '\0')                       (6)
            {
                x = 1;                                (7)
                y = 2;                                (8)
            }
        }
    }
    if (y == 2)                                     (9)
    {
        Lcd1602SendComm(LCD1602_LINE2 - 1 + x);    (10)
        for( ; x < (16 + 1) && *Data != '\0'; x++) (11)
        {
            Lcd1602SendDate(*Data++);               (12)
        }
    }
}
```

函数 Lcd1602DispStr() 首先判断字符串是否在第一行显示（程序清单 4.9(1)）；是否超过行尾（程序清单 4.9(2)）。如果在第一行的显示范围内开始显示，函数 Lcd1602DispStr() 将设置显示开始的地址（程序清单 4.9(3)），并开始写入显示字符（程序清单 4.9(5)）直到行尾或字符串结束（程序清单 4.9(4)）。接着判断显示字符串是否结

束（程序清单 4.9(6)），如果没有，重新设置显示的开始地址为第二行（程序清单 4.9(8)）第一列（程序清单 4.9(7)）。由于需要支持自动换行，函数 Lcd1602DispStr() 直接使用 if 判断是否在第二行显示（程序清单 4.9(9)）使字符串在第一行显示不完时可以在第二行开始处接着显示。如果在第二行显示，则也需要设置显示开始的地址（程序清单 4.9(10)），并接着写入显示字符（程序清单 4.9(12)）直到行尾或字符串结束（程序清单 4.9(11)）。因为液晶模块仅两行，所以不需要再次判断字符串是否显示完毕。

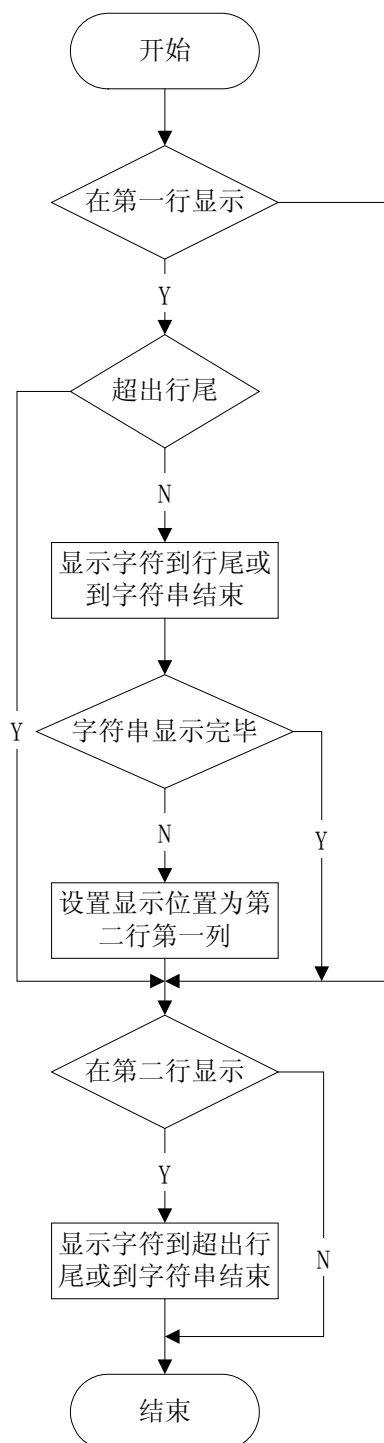


图 4.2 单任务在指定位置显示字符串流程图

7. 在指定地址向液晶模块写多个字符

如果用户需要把任意字符写入液晶模块 TC1602A 的任意地址，可以调用 OSLcd1602LoadC() 或 Lcd1602LoadC() 实现。当用户有多个任务需要操作液晶模块 TC1602A，使用 OSLcd1602LoadC() 来写多个字符。当用户仅一个任务需要操作液晶模块 TC1602A，则使用 Lcd1602LoadC() 来写多个字符。OSLcd1602LoadC() 是一个宏，代码见程序清单 4.10。

程序清单 4.10 多任务中在指定地址写多个字符

```
#define OSLcd1602LoadC(addr, dstr, no)
{
    OSSemPend(LCD1602_SEM, 0);                      (1)
    Lcd1602LoadC ((addr), (dstr), (no));              (2)
    OSSemPost(LCD1602_SEM);                          (3)
}
```

程序通过对液晶模块 TC1602A 写字符之前等待信号量（程序清单 4.10(1)）和对液晶模块 TC1602A 写字符之后发送信号量（程序清单 4.10(3)）来实现对器件的互斥操作。这样做的原因可以参见 4.1 节。在宏中调用函数 Lcd1602LoadC() 对液晶模块 TC1602A 写字符。而函数 Lcd1602LoadC() 就是单任务情况下对液晶模块 TC1602A 写字符的函数，所以两者的参数相同。

函数 Lcd1602LoadC() 的代码见程序清单 4.11。函数 Lcd1602LoadC() 的第一个参数 Addr 为将要写入字符的开始地址；第二个参数 Data 为指向将要写入的字符；第三个参数 NChar 为将要写入的字符数目。程序比较简单，这里不再说明。

程序清单 4.11 单任务中在指定地址写多个字符

```
void Lcd1602LoadC(uint8 Addr, uint8 *Data, uint8 NChar)
{
    Lcd1602SendComm(Addr | 0x80);                    // 设置写入地址
    do
    {
        Lcd1602SendData(*Data++);
    } while (--NChar != 0);
}
```

8. 驱动程序在 DP-51PROC 上使用的例子

在 DP-51PROC 上运行本程序后，液晶 TC1602A 的第一行闪动显示字符串 "Small RTOS51"，第二行滚动显示另一个长字符串。（接线可以参考实验 26 上的接法）

例子的主要代码见程序清单 4.12。程序比较简单，这里不再说明。

程序清单 4.12 驱动程序使用的例子主要代码

```
/* *****
** 函数名称: main
```

```

** 功能描述: 主函数, 用户程序从这里执行
** 输 入: 无
** 输 出: 无
** 全局变量: 无
** 调用模块: init(),OSStart(),LCMIni(),LCMCln();
*****/

void main(void)
{
    init();
    Lcd1602Init();
    OSStart();
}

/*****
** 函数名称: LcdDisplay1
** 功能描述: 一个任务, 在液晶第一行闪动字符串 "Small RTOS51"
** 输 入: 无
** 输 出: 无
** 全局变量: 无
** 调用模块: OSSemCreate(),OSLcd1602DispStr(),OSWait(),Lcd1602Clr()
*****/

void LcdDisplay1(void)
{
    OSSemCreate(LCD1602_SEM, 1);
    while (1)
    {
        OSLcd1602Clr(1);           // 第一行清屏
        OSWait(K_TMO, OS_TICKS_PER_SEC / 2); // 延时 0.5S
        OSLcd1602DispStr(4, 1, "Small RTOS51");
        // 第一行显示" Small RTOS51"
        OSWait(K_TMO, (OS_TICKS_PER_SEC + 1) / 2); // 延时 0.5S
    }
}

/*****
** 函数名称: LcdDisplay2
** 功能描述: 一个任务, 在液晶第二行滚动显示一个字符串
** 输 入: 无
** 输 出: 无
** 全局变量: 无
** 调用模块: OSLcd1602DispStr(),OSWait()
*****/

```

```

char xdata LogoStr[] = " Hello,World! Down it from www.zlgmcu.com";

void LcdDisplay2(void)
{
    uint8 *cp;

    cp = LogoStr;

    while(1)
    {
        OSLcd1602Clr(2);                // 第二行清屏
        OSLcd1602DispStr(1, 2, cp);     // 显示字符串
        OSWait(K_TMO, OS_TICKS_PER_SEC / 4); // 延时 0.25S

        cp++;
        if (*cp == '\0')
        {
            cp = LogoStr;
        }
    }
}

```

代码的其它部分参见本书配套光盘中的源代码。

4.3 I²C 总线驱动程序的实现

4.3.1 I²C 驱动程序的简介

本驱动程序为标准的 51 系列 CPU 编写，让 CPU 模拟成一个 I²C 总线主器件，并部分支持多个主器件同时存在。当 CPU 晶振为 12MHz 时，I²C 总线频率为不超过 100KHz。如果 I²C 总线上有多个 I²C 总线主器件，用户程序需要进行一些额外处理。本书配套光盘也包含一个模拟 400KHz 的 I²C 总线规范主器件的驱动程序。而 DP-51PROC 上的 I²C 器件并不是全都为 400KHz 的，如 ZLG7290 LED 键盘控制芯片的速度就比较慢，是 32KHz，所以该驱动程序不适合 ZLG7290。

4.3.2 驱动程序的使用

本驱动程序可以在没有 Small RTOS51 的情况下使用。此时，要使用本驱动程序只需要配置 I²C 总线使用的 IO 口。在驱动程序的主文件 iic_master.c 仅包含一个文件 config.h。用户需要的是在这个文件中设置 I²C 总线使用的 IO 口 SDA 和 SCL。定义 SDA 和 SCL 的例子见程序清单 4.13。如果用户单独使用 iic_master.c，还要在 config.h 包含 iic_master.h 文件和其它必须的文件如 reg51 等；并定义宏 TRUE、FALSE 和与编译器无

关的数据类型。在使用 Small RTOS51 的情况下，如果用户只有一个任务使用 I²C 总线，用户只要在 config.h 定义 SDA 和 SCL 和包含 iic_master.h 就可以了。如果用户不止一个任务使用 I²C 总线，则驱动程序需要使用信号量保证各个任务对 I²C 总线的互斥操作。这时，需要将宏 IICSem 定义为分配给 I²C 总线驱动程序的信号量的索引，并在使用驱动程序前建立这个信号量。

在使用 I²C 总线驱动程序前应该调用函数 IICInit() 初始化 I²C 总线。单独使用或单任务使用本驱动程序时，使用函数 IICRead() 对 I²C 总线进行读操作，使用 IICWrite() 对 I²C 总线进行写操作。如果有多个任务需要对 I²C 总线进行操作，则分别调用宏 OSIICRead() 和 OSIICWrite() 对其进行读写。

程序清单 4.13 定义 I²C 使用的 IO 口例

```
sbit SDA = P1^7;           //I2C 总线驱动使用的数据线
sbit SCL = P1^6;           //I2C 总线驱动使用的时钟线
```

4.3.3 基本 I²C 总线信号的产生

I²C 总线有很多基本总线信号，每一个基本总线信号由一个函数产生。这些函数都比较简单，读者对比 I²C 总线规范应该很容易读懂。

I²C 总线启动信号由函数 IICStart 产生，代码见程序清单 4.14。当操作成功，函数返回 TRUE。当函数返回 FALSE 时可能有别的总线主器件正在使用总线或总线故障。

程序清单 4.14 产生 I²C 总线启动信号

```
uint8 IICStart(void)
{
    SDA = 1;
    SCL = 1;
    if (SDA == 1)
    {
        SDA = 0;
        _nop_();
        SCL = 0;
        SDA = 1;
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}
```

I²C 总线中止信号由函数 IICStop 产生，代码见程序清单 4.15。函数没有返回值。

程序清单 4.15 产生 I²C 总线中止信号

```
void IICStop(void)
```

```
{
    SDA = 0;
    _nop_();
    _nop_();
    SCL = 1;
    SDA = 1;
    _nop_();
    _nop_();
    _nop_();
    SCL = 0;
}
```

I²C 总线应答信号（ACK）由函数 IIC_ACK 产生，代码见程序清单 4.16。函数没有返回值。

程序清单 4.16 产生 I²C 总线应答（ACK）信号

```
void IIC_ACK(void)
{
    SDA = 0;
    _nop_();
    _nop_();
    SCL = 1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    SCL = 0;
}
```

I²C 总线非应答信号（NO ACK）由函数 IIC_NO_ACK 产生，代码见程序清单 4.17。函数没有返回值。

程序清单 4.17 产生 I²C 总线非应答（NO ACK）信号

```
void IIC_NO_ACK(void)
{
    SDA = 1;
    _nop_();
    _nop_();
    SCL = 1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    SCL = 0;
}
```

```
    return;
}
```

4.3.4 I²C 总线初始化

在使用 I²C 总线前必须初始化 I²C 总线，使 I²C 总线处于空闲状态。这是通过发送总线停止信号来实现的。代码见程序清单 4.18。

程序清单 4.18 I²C 总线初始化

```
void IICInit(void)
{
    SCL = 0;
    IICStop();
}
```

4.3.5 发送和接收一个字节

发送一个字节和接收一个字节也是 I²C 总线的基本操作。对 I²C 的写操作需要用到这两个操作。发送一个字节使用函数 IICSend()实现。函数 IICSend()的代码见程序清单 4.19。流程图见图 4.3。函数同时处理了应答位。

程序清单 4.19 向 I²C 发送一个字节

```
uint8 IICSend(uint8 IIC_data)
{
    uint8 i;
    for (i = 0; i < 8; i++) (1)
    {
        IIC_data = IIC_data << 1; (2)
        FO = SDA = CY; (3)
        SCL = 1; (4)
        if (FO != SDA) (5)
        {
            SCL = 0; (6)
            return FALSE; (7)
        }
        _nop_(); (8)
        _nop_(); (9)
        SCL = 0; (10)
    }
    SDA = 1; (11)
    _nop_(); (12)
    _nop_(); (13)
    SCL = 1; (14)
    _nop_(); (15)
```



```
_nop_(); (16)
if (SDA == 1) (17)
{
    SCL = 0; (18)
    return FALSE; (19)
}
else
{
    SCL = 0; (20)
    return TRUE; (21)
}
}
```

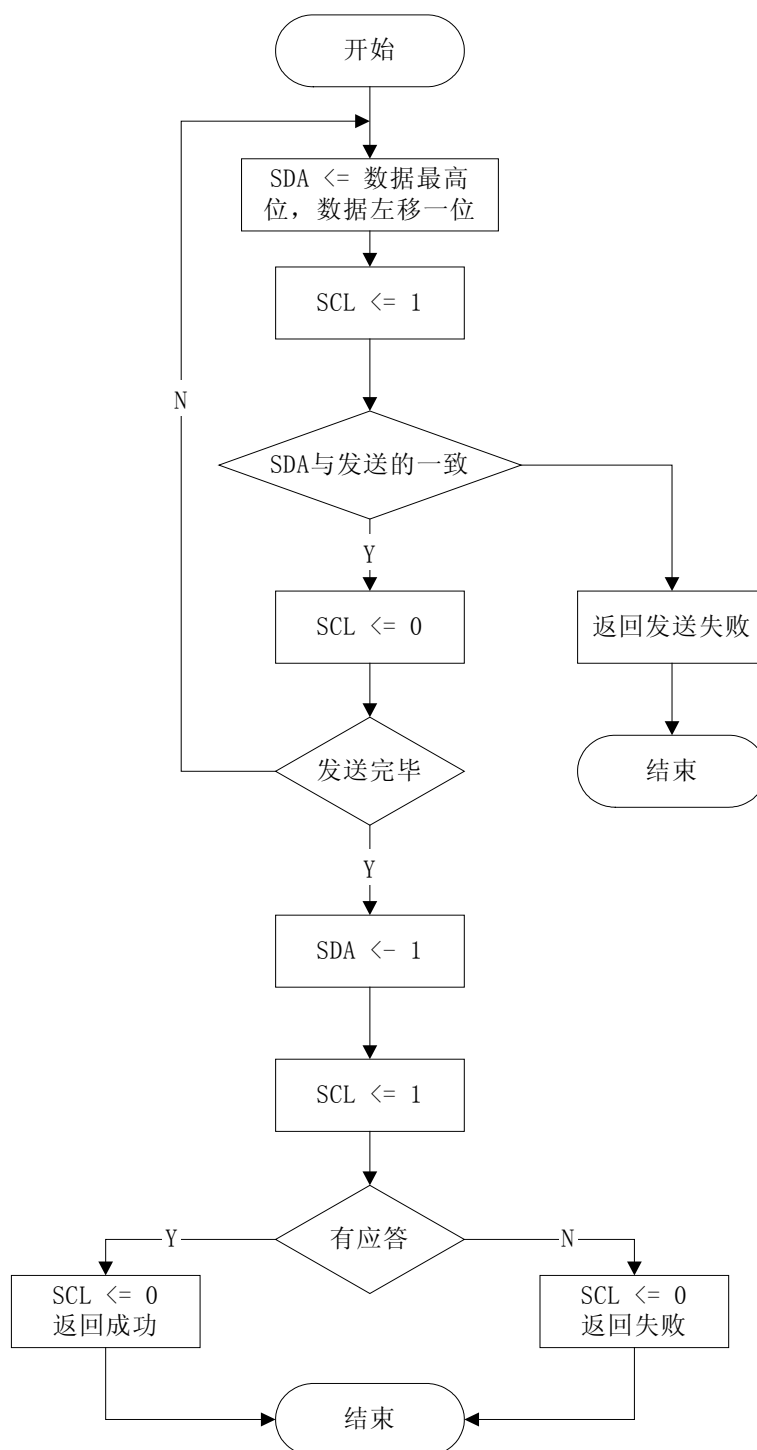


图 4.3 I²C 发送一个字节

有了流程图，程序应该比较容易看懂。唯一要注意的是程序清单 4.19 (2)、(3)句和 F0 的使用。在 Keil C51 中，左移(<<)操作会把最高位移到 CY 标志中。同理，右移移(>>)操作会把最低位移到 CY 标志中。这是不可移植的，但却是效率最高的方式。可移植的方式见程序清单 4.20。同理，F0 是 51 单片机中用户可使用的标志，在 PSW 中。虽然使用 F0 隐含不可移植性，但没有程序清单 4.19 (2)、(3)句那么严重，移植时只要定义一个全局（或局部）变量 F0 就可以了。

程序清单 4.20 可移植代码

```

if ((IIC_data & 0x80) != 0)
{
    FO = SDA = 1;
}
else
{
    FO = SDA = 0;
}
IIC_data = IIC_data << 1;

```

接收一个字节使用函数 `IICReceive()` 实现，代码见程序清单 4.21。由于接收一个字节后发送的应答信号不尽相同，函数没有处理应答信号。程序比较简单，读者对照 I²C 总线规范应该可以读懂，这里不再说明。

程序清单 4.21 从 I²C 从器件接收一个字节

```

uint8 IICReceive(void)
{
    uint8 i,r;

    r = 0;                                     (1)
    SDA = 1;                                  (2)
    for (i = 0; i < 8; i++)                   (3)
    {
        r = r * 2;                            (4)
        SCL = 1;                              (5)
        _nop_();                              (6)
        _nop_();                              (7)
        if (SDA == 1)                         (8)
        {
            r++;                               (9)
        }
        SCL = 0;                              (10)
    }
    return r;                                 (11)
}

```

4.3.6 对 I²C 进行读操作

如果有多个任务需要访问 I²C 总线，则使用 `OSIICRead()` 对 I²C 总线进行读操作。如果仅有一个任务需要 I²C 总线，则使用 `IICRead()` 对 I²C 总线进行读操作。`OSIICRead()` 是一个宏，代码见程序清单 4.22。

程序清单 4.22 多任务从 I²C 读数据

```

#define OSIICRead(a,b,c)

```

```

if (OSSemPend(IICSem,10) == OS_SEM_OK)           (1)
{
    IICRead(a,b,c);                               (2)
    OSSemPost(IICSem);                             (3)
}

```

程序通过在对器件读之前等待信号量（程序清单 4.22 (1)）和在对器件读之后发送信号量（程序清单 4.22 (3)）来实现对 I²C 总线的互斥操作。这样做的原因可以参见本章的 4.1.1 节。在宏中调用函数 IICRead() 对器件进行读操作（程序清单 4.22 (2)）。而 IICRead() 就是单任务情况下对 I²C 总线进行读操作的函数，所以两者的参数相同。

函数 IICRead() 的代码见程序清单 4.23。函数 IICRead() 的流程图见图 4.4。函数 IICRead() 的第一个参数是一个指针，读出的数据将从这里开始存放。函数 IICRead() 的第二个参数是将要访问的从器件的器件地址。函数 IICRead() 的第三个参数是将要读取的字节数目。当函数 IICRead() 成功读取数据时，返回 TRUE。函数 IICRead() 返回 FALSE 时可能有别的 I²C 总线主器件正在访问 I²C 总线，或是总线故障，或是从器件故障。

程序清单 4.23 单任务从 I²C 读数据

```

uint8 IICRead(uint8 OS_SEM_MEM_SEL *Ret,uint8 Addr,uint8 NByte)
{
    uint8 i;
    Addr = Addr | 0x01;                               (1)
    if (IICStart() == FALSE)                           (2)
    {
        return FALSE;                                  (3)
    }
    if(IICSend(Addr) == FALSE)                         (4)
    {
        return FALSE;                                  (5)
    }
    i = NByte - 1;                                     (6)
    if (i != 0)                                         (7)
    {
        do
        {
            *Ret++ = IICReceive();                      (8)
            IIC_ACK();                                  (9)
        } while (--i != 0);                             (10)
    }
    *Ret = IICReceive();                               (11)
    IIC_NO_ACK();                                       (12)
    IICStop();                                          (13)
    return TRUE;                                        (14)
}

```

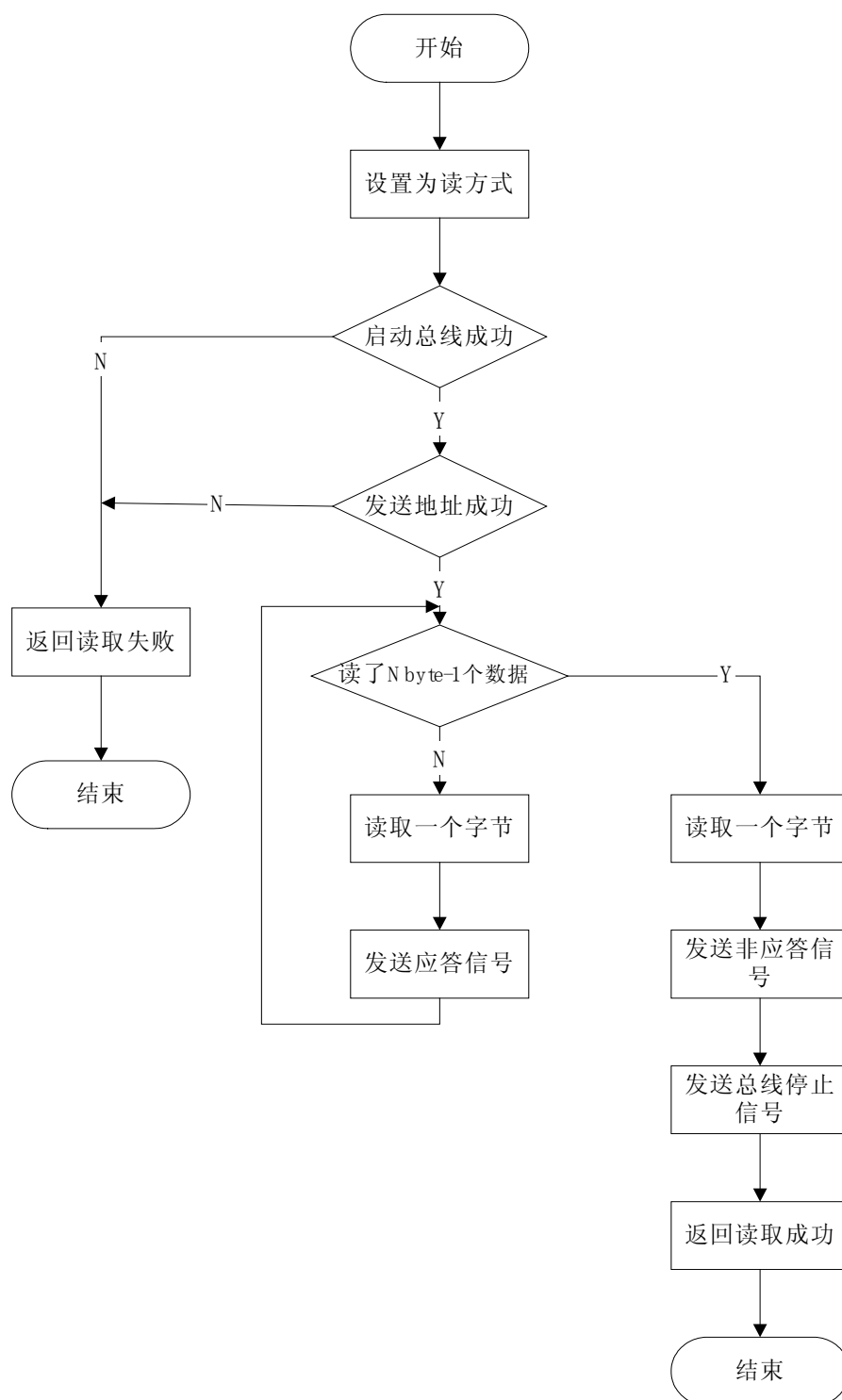


图 4.4 从 I²C 读取数据流程图

函数 `IICRead()` 首先将地址的最低位设置为 1（程序清单 4.23(1)）告诉从器件此次操作为读。然后启动总线（程序清单 4.23(2)）。如果启动不成功，函数返回 `FALSE`（程序清单 4.23(3)）。否则发送从器件地址（程序清单 4.23(3)）。如果发送不成功，函数返回 `FALSE`（程序清单 4.23(5)）。否则就读取比指定读取的字节数少 1 的字节数，在每次读取一个字节后发送应答（ACK）信号（程序清单 4.23(6)~(10)）。然后，接收最后一个字节（程序清单 4.23(11)），并发送非应答（NO ACK）信号通知从器件读取结束（程序清单 4.23(12)）。最后，函数停止总线（程序清单 4.23(13)），函数调用成功（程序清单

4.23(14))。

4.3.7 对 I²C 进行写操作

如果有多个任务需要访问 I²C 总线，则使用 OSIICWrite()对 I²C 总线进行写操作。如果仅一个任务需要 I²C 总线，则使用 IICWrite()对 I²C 总线进行写操作。OSIICWrite()是一个宏，代码见程序清单 4.24。

程序清单 4.24 多任务给 I²C 从器件写数据

```
#define OSIICWrite(a,b,c)
    if (OSSemPend(IICSem,10) == OS_SEM_OK)                (1)
    {
        IICWrite(a,b,c);                                    (2)
        OSSemPost(IICSem);                                  (3)
    }
```

程序通过在对器件写之前等待信号量（程序清单 4.24(1)）和在对器件写之后发送信号量（程序清单 4.24(3)）来实现对 I²C 总线的互斥操作。这样做的原因可以参见本章的 4.1.1 节。在宏中调用函数 IICWrite()对器件进行写操作（程序清单 4.24(2)）。而 IICWrite()就是单任务情况下对 I²C 总线进行写操作的函数，所以两者的参数相同。

函数 IICWrite()的代码见程序清单 4.25。函数 IICWrite()的第一个参数是一个指针，将要写入的数据将从这里开始存放。函数 IICWrite()的第二个参数是将要访问的从器件的器件地址。函数 IICWrite()的第三个参数是将要写入的字节数。当函数 IICWrite()成功写入数据时，返回 TRUE。函数 IICWrite()返回 FALSE 时可能有别的 I²C 总线主器件正在访问 I²C 总线，或是总线故障，或是从器件故障。

程序清单 4.25 单任务给 I²C 从器件写数据

```
uint8 IICWrite(uint8 Addr,uint8 OS_SEM_MEM_SEL *Data,uint8 NByte)
{
    uint8 i;
    Addr = Addr & 0xfe;                                     (1)
    if (IICStart() == FALSE)                                (2)
    {
        return FALSE;                                       (3)
    }
    if (IICSend(Addr) == FALSE)                             (4)
    {
        return FALSE;                                       (5)
    }
    i = NByte;                                              (6)
    do                                                       (7)
    {
        if (IICSend(*Data++) == FALSE)                     (8)
        {
```

```

        return FALSE;                                     (9)
    }
    } while (--i !=0 );
    IICStop();                                           (10)
    return TRUE;                                         (11)
}
    
```

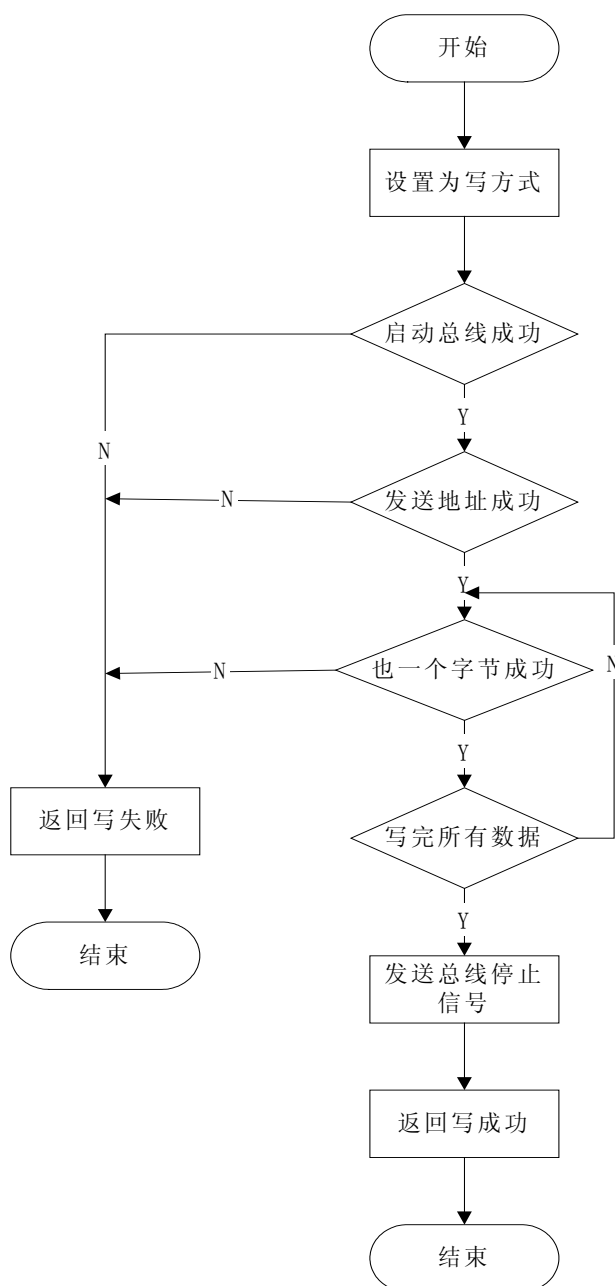


图 4.5 把数据写道 I²C 流程图

函数 IICWrite() 首先将地址的最低位设置为 0 (程序清单 4.25(1)) 告诉从器件此次操作为写。然后启动总线 (程序清单 4.25(2))。如果启动不成功, 函数返回 FALSE (程序清单 4.25(3))。否则发送从器件地址 (程序清单 4.25(4))。如果发送不成功, 函数返回 FALSE (程序清单 4.25(5))。否则就写入指定字节 (程序清单 4.25(6)~(9))。在每次写入一个字节后读取发送应答判断发送一个字节的函数是否调用正确 (程序清单 4.25(8)),

正确才继续执行，否则函数返回 FALSE（程序清单 4.25(9)）。最后，函数停止总线（程序清单 4.25(10)），函数调用成功（程序清单 4.25(11)）。

4.4 Small RTOS51 应用实例和分析

4.4.1 实例简介

这是 Small RTOS51 在 DP-51PROC 实验仪上实现一个简单游戏的程序。

程序使用按键 S7 发出“子弹”，而数码管每秒送出一个“8”，当子弹打中“8”，则“8”消失，而每秒“8”就往右移动一格，每两秒就会从最左边产生一个“8”，每个子弹只能打中一个“8”。其实，仅仅从上面的描述，我们就可以把它划分为几个任务了，显示、处理 8 的产生和移动、产生子弹、子弹的移动、中弹的处理。读者在 DP-51PROC 上的接线可以参考实验 16 的 I²C 接线。

由于仅仅是一个例子用来说明如何使用 RTOS 51 上编写程序，而不是研制一个产品，所以程序有一些细节没有完善。读者可以自行完善它。

为了锻炼读者分析代码的能力，本章不再分析程序代码，由读者自己分析。如果读者能够自行编写类似程序则说明读者已经初步具备在 RTOS51 编写应用程序的能力了。

4.4.2 系统配置文件 Os_cfg.h:

程序清单 4.26 Os_cfg.h

```

/*****
**
**                      Small RTOS51(51)
**                      The Real-Time Kernel(For Keil c51)
**
**                      (c) Copyright 2002-2002, chenmingji
**                      All Rights Reserved
**                      V1.12.1
**-----文件信息-----
**文 件 名: OS_CFG.H
*****/

#define    OS_MAX_TASKS            6
    /* 最大任务数 1~16 */
#define    OS_TICKS_PER_SEC        10
    /* 声明 1 秒系统节拍数 */
#define    EN_USER_TICK_TIMER      0
    /* 禁止(0)或允许(1)系统定时中断调用用户函数 UserTickTimer() */
#define    EN_OS_INT_ENTER         1
    /* 禁止(0)或允许(1)中断嵌套管理 */
#define    EN_TIMER_SHARING        1
    
```



```

    /* 禁止(0)或允许(1)定时器中断调用 OSTimeTick() */
#define    TICK_TIMER_SHARING    1
    /* 定义进入多少次硬件中断为一次系统定时器软中断 */

#define    EN_OS_Q                0
    /* 禁止(0)或允许(1)消息队列 */
#define    EN_OS_Q_CHK            0
    /* 禁止(0)或允许(1)校验消息队列指针 */
#define    OS_Q_MEM_SEL          xdata
    /* 消息队列存储空间选择, keil c51 有用, 必须为 idata、xdata  */
    /* 不是 keil c51 时它必须为空格 */
#define    EN_OS_Q_PENT          1
    /* 禁止(0)或允许(1)等待队列中的消息 */
#define    EN_OS_Q_ACCEPT        0
    /* 禁止(0)或允许(1)无等待的从队列中取得消息 */
#define    EN_OS_Q_POST          0
    /* 禁止(0)或允许(1)FIFO 方式向对列发送消息 */
#define    EN_OS_Q_POST_FRONT    1
    /* 禁止(0)或允许(1)LIFO 方式向对列发送消息 */
#define    EN_OS_Q_INT_POST      0
    /* 禁止(0)或允许(1)中断中 FIFO 方式相对列发送消息 */
#define    EN_OS_Q_INT_POST_FRONT 1
    /* 禁止(0)或允许(1)中断中 LIFO 方式相对列发送消息 */
#define    EN_OS_Q_NMsgs         1
    /* 禁止(0)或允许(1)取得队列中消息数 */
#define    EN_OS_Q_SIZE          0
    /* 禁止(0)或允许(1)取得队列总容量 */
#define    EN_OS_Q_FLUSH         0
    /* 禁止(0)或允许(1)清空队列 */

#define    EN_OS_SEM              1
    /* 禁止(0)或允许(1)信号量 */
#define    EN_OS_SEM_CHK         0
    /* 禁止(0)或允许(1)校验信号量索引 */
#define    OS_SEM_MEM_SEL        idata
    /* 信号量存储空间选择, keil c51 有用, 必须为 idata、xdata  */
    /* 不是 keil c51 时它必须为空格 */
#define    OS_MAX_SEMS           2
    /* 最大信号量数目 */
#define    EN_OS_SEM_PENT        1
    /* 禁止(0)或允许(1)等待信号量 */
#define    EN_OS_SEM_ACCEPT      0
    /* 禁止(0)或允许(1)无等待请求信号量 */
#define    EN_OS_SEM_INT_POST    0

```

```

    /* 禁止(0)或允许(1)中断中发送信号量*/
#define    EN_OS_SEM_POST    1
    /* 禁止(0)或允许(1)中发送信号量*/
#define    EN_OS_SEM_QUERY    0
    /* 禁止(0)或允许(1)查询信号量*/

#if EN_TIMER_SHARING == 0
#define TIME_ISR_TASK_ID    SHOW_TASK_ID
    /* 定义系统定时器软中断触发的任务 ID*/
#endif

#ifdef IN_OS_CPU_C
#if EN_USER_TICK_TIMER > 0
extern void UserTickTimer(void);
    /* 系统定时中断中调用的用户函数*/
#endif
#endif
/*****end*****/

```

4.4.3 CPU 配置文件 Os_cpu.h:

程序清单 4.27 Os_cpu.h

```

/*****
**
**                      Small RTOS51
**                      The Real-Time Kernel For Keil c51
**
**                      (c) Copyright 2002-2002, chenmingji
**                      All Rights Reserved
**                      V1.12.1
**-----文件信息-----
**文 件 名: OS_CPU.h
*****/
#define    EN_SP2    0    /* 禁止(0)或允许(1)软非屏蔽中断*/

#ifdef __C51__

typedef    unsigned char    uint8;
    /* 定义可移植的无符号 8 位整数关键字*/
typedef    signed    char    int8;
    /* 定义可移植的有符号 8 位整数关键字*/
typedef    unsigned int    uint16;
    /* 定义可移植的无符号 16 位整数关键字*/
typedef    signed    int    int16;

```

```

    /* 定义可移植的有符号 16 位整数关键字*/
typedef    unsigned long  uint32;
    /* 定义可移植的无符号 32 位整数关键字*/
typedef    signed    long  int32;
    /* 定义可移植的有符号 32 位整数关键字*/

#ifndef    NULL
#define    NULL 0
#endif

#if EN_OS_INT_ENTER >0
#define    OS_INT_ENTER() OSIntNesting++,EA=1
#endif

#define    OS_ENTER_CRITICAL()    EA = 0,Os_Enter_Sum++
    /* 禁止中断*/
#define    OS_EXIT_CRITICAL()    if (--Os_Enter_Sum==0) EA = 1
    /* 允许中断*/

#define    HIGH_BYTE    0    /* uint16 的高位字节*/
#define    LOW_BYTE    1    /* uint16 的低位字节*/

#define    OS_TASK_SW()    OSCtxSw()    /* 任务切换函数*/

#define    IDATA_RAM_SIZE    0x100    /* idata 大小*/

extern void OSCtxSw(void);
extern void OSIntCtxSw(void);
#ifndef    IN_OS_CPU_C
extern void OSStart(void);
#endif

#define    Sp2Space    4
    /* 高级中断（软非屏蔽中断）堆栈大小 EN_SP2 为 0 时无效*/
#define    OS_TIME_ISR    1
    /* 系统定时器使用的中断*/

#endif

#ifndef    __C51__

SET_EA    MACRO                                ;打开所有中断允许
        SETB    EA
ENDM

```

```
#endif
/*****end*****/
```

24.2.3 用户配置文件 Config.h:

程序清单 4.28 Config.h

```

/*****
**
**                               Small RTOS51
**                               The Real-Time Kernel For Keil c51
**
**                               (c) Copyright 2002-2002, chenmingji
**                               All Rights Reserved
**                               V1.12.1
**-----文件信息-----
**文 件 名: CONFIG.H
*****/
/*****/
/*      "以下为系统配置"      */
/*****/

#pragma    REGPARMS
#include    <reg52.h>
#include    <intrins.h>
#include    <absacc.h>
#define     const code

#ifndef     TRUE
#define     TRUE    1
#endif

#ifndef     FALSE
#define     FALSE   0
#endif

/*****/
/*      "操作系统定义"      */
/*****/

#include    "OS_CFG.H"
#include    "OS_CPU.H"
#include    "..\os\OS.H"
#include    "..\os\OS_Q.h"
#include    "..\os\OS_SEM.h"
#include    "VIIC_C51.h"
#include    "zlg7290.h"

```

```

/*****/
/*      "以下为程序配置"          */
/*****/

//任务定义
#ifdef IN_OS_CPU_C
extern void TaskA(void);
extern void TaskB(void);
extern void TaskC(void);
extern void TaskD(void);
extern void TaskE(void);
extern void TaskF(void);

void (* const TaskFuction[OS_MAX_TASKS])(void)=
    { TaskA,TaskB,TaskC,TaskD,TaskE,TaskF };
//函数数组 TaskFuction[]保存了各个任务初始 PC 指针,其按任务 ID 顺序保存

#endif
/*****end*****/

```

4.4.4 ZLG7290 应用函数程序 zlg7290.c

程序清单 4.29 zlg7290.c

```

/*****
**
**                      Zlg7290 API
**                      Zlg7290 Application Program Interface
**
**                      (c) Copyright 2003-2004, Yehaoben
**                      All Rights Reserved
**                      V1.0
**-----文件信息-----
**文    件    名: zlg7290.c
*****/

#include "REG52.h"
#include "viic_c51.h"

#define    zlg7290        0x70          /*ZLG7290 的 I2C 地址*/

#define    SubKey          0x01
#define    SubCmdBuf       0x07
#define    SubDpRam        0x10

```

```

/*****
** 函数名称:    DelayNS
** 功能描述:    长软件延时
** 输 入:  I      :   延时参数, 值越大时延时越久
** 输 出:  无
** 全局变量:    无
** 调用模块:    无
*****/

void    delayMS(unsigned char i)
{
    unsigned char j,k;
    for(k=0;k<i;k++)
        for(j=0;j<60;j++);
}

/*****
** 函数名称:    ZLG7290_SendData
** 功能描述:    发送数据
** 输 入:  SubAdd:   输入数据
**          DATA  :   输入值
**
** 输 出:  0        :   Fail
**          1        :   OK
** 全局变量:    无
** 调用模块:    delayMS
*****/

unsigned char ZLG7290_SendData( unsigned char SubAdd,
                                unsigned char Data)
{
    if(SubAdd>0x17)
        return 0;
    ISendStr(zlg7290,SubAdd,&Data,1);
    delayMS(10);
    return 1;
}

/*****
** 函数名称:    ZLG7290_SendCmd
** 功能描述:    发送命令 (对子地址 7、8)
** 输 入:  DATA1 :   命令 1
**          DATA2 :   命令 2
**
** 输 出:  0        :   Fail
**          1        :   OK
*****/

```

```

** 全局变量:    无
** 调用模块:    ISendStr、delayMS
** */
        unsigned char ZLG7290_SendCmd( unsigned char Data1,
                                         unsigned char Data2)
{
    unsigned char Data[2];
    Data[0]=Data1;
    Data[1]=Data2;
    ISendStr(zlg7290,0x07,Data,2);
    delayMS(10);
    return 1;
}

/*****
** 函数名称:    ZLG7290_SendBuf
** 功能描述:    向显示缓冲区发送数据
** 输 入:  * disp_buf      :   要发送的数据的起始地址
**          num            :   发送个数
**
** 输 出:    无
** 全局变量:  无
** 调用模块:  ZLG7290_SendCmd
** */
void ZLG7290_SendBuf(unsigned char * disp_buf,unsigned char num)
{
    unsigned char i;
    for(i=0;i<num;i++)
    {
        ZLG7290_SendCmd(0x60+i,*disp_buf);
        disp_buf++;
    }
}

/*****
** 函数名称:    ZLG7290_SendBuf1
** 功能描述:    向显示缓冲区发送非译码数据
** 输 入:  * disp_buf      :   要发送数据的起始地址
**          num            :   发送字节个数
**
** 输 出:    无
** 全局变量:  无
** 调用模块:  ISendStr
** */

```

```
void ZLG7290_SendBuf1(unsigned char * disp_buf,unsigned char num)
{
    ISendStr(zlg7290,0x10,disp_buf,num);
}

/*****
** 函数名称:    ZLG7290_GetKey
** 功能描述:    读取键值
** 输 入:      无
**
** 输 出:      >0    键值
**              =0    无键按下
** 全局变量:    无
** 调用模块:    IRcvStr、delayMS
*****/
unsigned char  ZLG7290_GetKey()
{
    unsigned char rece;
    rece=0;
    IRcvStr(zlg7290,1,&rece,1);
    delayMS(10);
    return rece;
}
/*****end*****/
```

4.4.5 主程序 EXT1.c

程序清单 4.30 EXT1.c

```

/*****
**
**                      Small RTOS51_Example
**                      The Game For Small RTOS51
**
**                      (c) Copyright 2003-2004, Yehaoben
**                      All Rights Reserved
**                      V1.0
**-----文件信息-----
**文 件 名: EXT1.c
*****/
#include    "config.h"

void        DISPLAY(unsigned char * disp);
/*全局变量*/
uint8    key=0xff;
uint8    disp_buf[8]={0x08,0x0f,0x0f,0x0f,0x0f,0x0f,0x0f,0x0F};

```



```
void main(void)
{
    TMOD = (TMOD & 0XF0) | 0X01;
    TLO = 0x0;
    TH0 = 0x80;
    TR0 = 1;
    ET0 = 1;
    TFO = 0;
    DISPLAY(disp_buf);
    EX0=1;
    TI=1;
    OSSemCreate(0,1);
    OSStart();
}

/*键盘中断服务函数*/
void int_ex1() interrupt 0
{
    OS_INT_ENTER();
    OS_ENTER_CRITICAL();
    key=ZLG7290_GetKey();    //读取键值
    OS_EXIT_CRITICAL();
    OSIntExit();
}

/*显示任务*/
void TaskA(void)
{
    while (1)
    {
        OS_ENTER_CRITICAL();
        DISPLAY(disp_buf);    /*显示*/
        OS_EXIT_CRITICAL();
        OSWait(K_TMO,1);
    }
}

/*8 的移动和产生任务*/
void TaskB(void)
{
    uint8 x,y;
    x=0;
    while (1)
```

```

{
    OS_ENTER_CRITICAL();
    for(y=0;y<8;y++)
    {
        if(disb_buf[y]==8)
        {
            disb_buf[y]=0x0f;
            disb_buf[y+1]=0x08;          /*8 往左移动一位*/
            y++;
        }
    }
    if(x==0)
    {
        disb_buf[0]=0x0f;                /*使终点不断出 8*/
        x=1;
    }
    else
    {
        disb_buf[0]=0x08;
        x=0;
    }
    OS_EXIT_CRITICAL();
    OSWait(K_TMO,15);
}

/*接收按键任务*/

void TaskC(void)
{
    uint8 x=0xFF;
    while (1)
    {
        x=key;
        if(x==0x07)                      /*如果有键值键入就发出发弹命令*/
        {
            x=0xff;
            key=0xff;
            OSSemPost(0);
        }
        OSWait(K_TMO,1);
    }
}

/*发子弹任务*/

```

```
void TaskD(void)
{
uint8 y;

while (1)
{
    if(OSSemPend(0,0)==OS_SEM_OK)           /*接收到发弹信号*/
    {
        OS_ENTER_CRITICAL();
        disp_buf[7]=0x0a;                   /*发弹*/
        for(y=0;y<7;y++)
            if(disp_buf[y]==0x0a)
                disp_buf[7]=0x0f;
        OS_EXIT_CRITICAL();
    }
    OSWait(K_TMO,10);
}
}
```

/*子弹飞行任务*/

```
void TaskE(void)
{
uint8 y;

while (1)
{
    OS_ENTER_CRITICAL();
    for(y=8;y>1;y--)
    {
        if(disp_buf[y-1]==0x0a)           /*弹向前飞行*/
        {
            disp_buf[y-1]=0x0f;
            disp_buf[y-2]=0x0a;
            y=1;
        }
    }
    OS_EXIT_CRITICAL();
    OSWait(K_TMO,3);
}
}
```

/*中弹处理任务*/

```
void TaskF(void)
{
```

```

uint8 y;

while (1)
{
    OS_ENTER_CRITICAL();
    for(y=6;y>0;y--)
    {
        if( ( disp_buf[y] == 0X08 ) && ( disp_buf[y+1] == 0X0A ) )
            /*如果弹碰到 8 就消失*/
            {
                disp_buf[y]=0x0F;
                disp_buf[y+1]=0x0F;
            }
    }
    OS_EXIT_CRITICAL();
    OSWait(K_TMO,2);
}
}

```

```

/*显示函数，由于该函数为不可重入函数，所以调用前先关中断*/
void    DISPLAY(unsigned char *disp)
{
    unsigned char buff[8];
    unsigned char i;
    for(i=0;i<8;i++)
    {
        switch(disp[i])
        {
            case 0x0f :
                buff[i]=0x00;
                break;
            case 0x08 :
                buff[i]=0xfe;
                break;
            case 0x0a :
                buff[i]=0x02;
                break;
            default :
                buff[i]=0x00;
                break;
        }
    }
    ZLG7290_SendBuf1(buff,8); //显示
}

```

/*****end*****/

例子程序的其它部分请参考本书配套光盘的中的源代码。

Small RTOS51 源码下载链接:

<http://www.zlgmcu.com/download/downs.asp?ID=447>

第 5 章 Small RTOS51 实验

实验一 LED 和键盘扫描驱动程序演示实验

一、实验目的

本程序展示了如何编写基于 Small RTOS 的键盘和 LED 扫描驱动程序。

二、实验设备及器件

PC 机 一台
DP-51PROC 单片机综合仿真实验仪 一台

三、实验步骤

1. 短接 A4 区的 JP11，使用导线将 A4 区的 /PL、CLK1 和 Q7 脚分别跟 A2 区的 P15、P16 和 P17 连接起来。将 A4 区的 CLK2 和 SER 脚接 GND。
2. 短接 A5 区的 JP10，使用导线将 A5 的 CLK 和 DINB 分别跟 A2 区的 P10、P11 连接起来。将 A5 区的 DINA 和/CLR 脚接 VCC。
3. 将 A4 区 J98、A5 区 J5 分别连接到 D1 区的 J52、J53。如下：

P0 ——— LED1	Q0 ——— KEY1
P1 ——— LED2	Q1 ——— KEY2
P2 ——— LED3	Q2 ——— KEY3
P3 ——— LED4	Q3 ——— KEY4
P4 ——— LED5	Q4 ——— KEY5
P5 ——— LED6	Q5 ——— KEY6
P6 ——— LED7	Q6 ——— KEY7
P7 ——— LED8	Q7 ——— KEY8
4. 下载 LED_Key.hex 文件到 DP-51PROC 中并运行。

四、实验参考程序主要部分

```

/*****
**
**          Small RTOS(51)
**      The Real-Time Kernel(For Keil c51)
**      (c) Copyright 2002-2004
**      All Rights Reserved
**          V1.20
*****/

#include "config.h"
uint8  Key_buffer[2];
uint8  show_key;
void   LED_Show();
void   Get_Key();
    
```

```
void init(void)
{
    TMOD = (TMOD & 0XF0) | 0X01;
    TH0 = (65536 - (11059200 / 12) / 100) / 256;
    TL0 = (65536 - (11059200 / 12) / 100) % 256;
    TR0 = 1;
    ET0 = 1;
    TF0 = 0;
}
```

```
void init_Port(void)
{
    P1 = 0xff;
}
```

```
/******
```

```
**函数名称: void KdTxByte(uint8 KdData)
```

```
**功能描述: 驱动 74HC164, 串转并
```

```
*****/
```

```
void KdTxByte(uint8 KdData)
```

```
{
    uint8 i;
    i = 8;
    do
    {
        KdClk = 1;
        if((KdData & 0x80)==0)
        {
            KdDat = 0;
        }
        else
        {
            KdDat = 1;
        }
        KdData <<=1;
        KdClk = 0;
    } while (--i != 0);
    KdClk = 1;
}
```

```
/******
```

```
**函数名称: uint8 Key_Scan(void)
```

```
**功能描述: 驱动 74HC165, 并转串
```

*****/

```
uint8 Key_Scan(void)
{
    uint8 i;
    uint8 key;

    key = 0;
    KeyPL = 0;
    KeyPL = 1;
    KeyClk = 0;
    for(i = 8; i > 0; i--)
    {
        key = (key<<1)|KeyIN;
        KeyClk = 1;
        KeyClk = 0;
    }
    return key;
}
```

*****/

```
main()
{
    OSInit();
    init();
    init_Port();
    OSTaskCreate(LED_Show , NULL , 0);
    while(1)
    {
        PCON = PCON | 0x01;          /* CPU 进入休眠状态 */
    }
}
```

****显示任务**

****功能：显示按键的状态**

*****/

```
void LED_Show(void)
{
    OSTaskCreate(Get_Key , NULL , 1);
    while(1)
    {
        KdTxdByte(show_key);
        OSWait(K_TMO , 1);
    }
}
```



```

}
/*****
**键盘扫描任务
**功能：获取键盘按键值
*****/
void Get_Key(void)
{
    while(1)
    {
        Key_buffer[0] = Key_Scan();
        OSWait(K_TMO , 1);
        Key_buffer[1] = Key_Scan();
        if(Key_buffer[0] == Key_buffer[1])
        {
            show_key = Key_buffer[0];
        }
        OSWait(K_TMO , 1);
    }
}

```

五、实验例程简析

本驱动程序创建了两个任务：显示任务和键盘扫描任务。创建方法如下：

```
OSTaskCreate(LED_Show , NULL , 0);
```

```
OSTaskCreate(Get_Key , NULL , 1);
```

显示任务的优先级比键盘扫描任务的要高。在键盘扫描任务中调用 **Key_Scan()** 函数获取按键的状态，为了实现按键的去抖动，在两次获取按键状态的中间调用了系统函数 **OSWait()** 来实现延时，从而达到按键去抖动效果。实际上 CPU 并不是在执行循环程序来实现延时，而是通过任务切换使显示任务运行。等待键盘扫描任务延时时间到，并且显示任务进入延时等待状态（调用 **OSWait()** 函数）时，键盘扫描任务才得以继续运行。

运行本程序的效果是按下某个键，相应的 LED 管就会发亮。松开按键，LED 管熄灭。以上的程序是 LED 和键盘扫描驱动程序在 Small RTOS51 中应用的部分源代码，完整的源代码请参考配套光盘中相关例子。

实验二 PCF8563 驱动程序演示实验

一、实验目的

PCF8563 是一款性价比极高的时钟芯片。它已被广泛用于电表、水表、气表、电话、传真机、便携式仪器以及电池供电的仪器仪表等产品领域。本实验示例展示了如何在 Small RTOS 51 中编写 PCF8563 的驱动程序。

二、实验设备及器件

PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三、实验步骤

- 1、使用导线连接 D5 区的 SCL、SDA 和 A2 区的 P16、P17（SCL~P16、SDA~P17），短接 D5 区的 JP1 跳线；
- 2、下载 PCF8563.hex 文件到 DP-51PRO.NET 中并运行；

四、实验参考程序主要部分

```

/*****
**
**                      Small RTOS(51)
**                      The Real-Time Kernel(For Keil c51)
**                      (c) Copyright 2002-2004, chenmingji
**                      All Rights Reserved
**                      V1.20
*****/
#include "CONFIG.h"
uint8  code td[9]={0x00,0x12,0x00,0x30,0x12,0x06,0x05,0x02,0x04}; //定义初始化字
uint8  disp_buf[8]={0,0,0,0,0,0,0,0}; //显示缓存
uint8  rd[7]; //定义接收缓冲区

void TaskA(void);
void TaskB(void);

/*****
*函数名称: unsigned char display_time(unsigned char *sd)
*功能描述: 驱动 ZLG7290 进行时间显示。显示格式: 时、分、秒
*****/
unsigned char display_time(unsigned char *sd)
{
    sd[0]=sd[0]&0x7f;//秒屏蔽保留位
    sd[1]=sd[1]&0x7f;//分屏蔽保留位
    sd[2]=sd[2]&0x3f;//时屏蔽保留位

```

```

    disp_buf[0] =(sd[0]%16);
    disp_buf[1] =(sd[0]/16);
    disp_buf[2] = 31;
    disp_buf[3] =(sd[1]%16);
    disp_buf[4] =(sd[1]/16);
    disp_buf[5] = 31;
    disp_buf[6] =(sd[2]%16);
    disp_buf[7] =(sd[2]/16);
    ZLG7290_SendBuf(disp_buf,8);
    return 0;
}

/*****
*函数名称: void Init_PCF8563(void)
*功能描述: 初始化 PCF8563 芯片
*****/
void Init_PCF8563(void)
{
    RST=0;
    _nop_();
    _nop_();
    _nop_();
    RST=1;
    ISendStr(PCF8563,WRADDR,td,0x5);
    _nop_();
    _nop_();
    _nop_();
    ISendStr(PCF8563,WRADDR+5,&td[5],0x4);
}

/*****
*时间显示任务
*****/
void TaskA(void)
{
    OSTaskCreate(TaskB,NULL,1);
    while(1)
    {
        OS_ENTER_CRITICAL();
        display_time(rd);
        OS_EXIT_CRITICAL();
        OSWait(K_TMO,1);
    }
}

```

```

/*****
* 取时间任务
*****/

void TaskB(void)
{
    while(1)
    {
        OS_ENTER_CRITICAL();
        IRcvStr(PCF8563,RDADDR,rd,0x7);
        OS_EXIT_CRITICAL();
        OSWait(K_TMO,1);
    }
}

void main()
{
    OSInit();
    TMOD = (TMOD & 0XF0) | 0X01;

    // 每 10ms 发生一次 T0 中断
    TL0 = (65536 - (11059200 / 12) / 100) % 256;
    TH0 = (65536 - (11059200 / 12) / 100) / 256;
    TR0 = 1;
    ET0 = 1;
    TF0 = 0;
    Init_PCF8563();
    OSTaskCreate(TaskA,NULL,0);
    while(1)
    {
        PCON = PCON | 0x01;                /* CPU 进入休眠状态 */
    }
}

```

五、实验示例程序简析

有关 PCF8563 芯片的应用和 ZLG7290 的使用，这里不再详述。用户可以阅读 DP-51PROC 的相关的实验例子和文档资料。

本实验程序使用动态创建任务的方法创建了两个任务：TaskA 和 TaskB。方法如下：

```
OSTaskCreate (TaskA, NULL, 0);
```

```
OSTaskCreate (TaskB, NULL, 1);
```

注意，TaskB 任务是在 TaskA 任务内部创建的。并且 TaskA 的优先级比 TaskB 的高。由于对 I2C 总线驱动程序的操作不具有可重入性，所以使用了以下宏

```
OS_ENTER_CRITICAL ();
```

```
OS_EXIT_CRITICAL ();
```

把 `display_time(rd)` 和 `IRcvStr(PCF8563,RDADDR,rd,0x7)` 变成临界代码区,从而确保 I2C 总线驱动程序的互斥操作。使用信号量也可以实现 I2C 总线驱动程序的互斥操作。若要使用信号量,则必须在 `OS_cfg.h` 的 `EN_OS_SEM` 置为 1,把 `SmallRTOS` 配置为允许使用信号量。示例程序中未对 `PCF8563` 进行校时处理,用户可以自行添加代码以实现此功能。

实验三 图形液晶显示驱动实验

一. 实验目的

本驱动程序展示了如何在 Small RTOS 中编写图形液晶显示器的驱动程序。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三. 实验步骤

1. A2 区的 A0~A2 分别连接到 B3 区的 A0~A2。
2. 将 A2 区的 P10 连接到 B3 区的 RST。
3. 将 A3 区的 Y0 连接到 B3 区的 /CS。
4. 将 A2 区的 A15~A10 分别连接到 A3 区的相应接线柱，如下：

A15	——	/G2B
A14	——	/G2A
A13	——	G1
A12	——	C
A11	——	B
A10	——	A

5. 在 B3 区的 J92 插入图形液晶模块(单色, 128×64 点)。
6. 将 B3 区的 J85 短接, A3 区的 JP4 短接。
7. 使用 TKSMonitor51 进行仿真调试。

四. 实验参考程序主要部分

```
#include "config.h"
void TaskA(void);
void TaskB(void);
void TaskC(void);
void TaskD(void);
void TaskE(void);

uint8 random(uint8 seed)
{
    uint16 temp;
    OS_ENTER_CRITICAL();
    temp = (((uint16) rand()) ^ ((uint16) rand()) + TL0) % (uint16)seed;
    OS_EXIT_CRITICAL();
    return temp;
}
```

```
void main(void)
{
    OSInit();
    TMOD = (TMOD & 0XF0) | 0X01;
    TL0 = 0x0;
    TH0 = 0x0;
    TR0 = 1;
    ET0 = 1;
    TF0 = 0;

    OSSemCreate(ZL12864_SEM, 1);
    LCM_DispIni();
    OSDispClr();

    //以下的字符串将在液晶上显示出来
    OSDispStr(0, 0, "Small RTOS the Real-Time Kernel");
    OSDispStr(4, 1, "ChenMingJi");
    OSDispStr(5, 3, "ZL12864 Ex1");

    OSTaskCreate(TaskA, NULL, 0);
    while(1)
    {
        PCON = PCON | 0x01;                /* CPU 进入休眠状态 */
    }
}

void TaskA(void)
{
    uint8 x,y;

    OSWait(K_TMO, 30);
    OSDispClr();

    OSTaskCreate(TaskB, NULL, 1);
    OSTaskCreate(TaskC, NULL, 2);
    OSTaskCreate(TaskD, NULL, 3);
    OSTaskCreate(TaskE, NULL, 4);

    while (1)
    {
        x = random(16);
        y = random(8);
        OSDispChar(y, x, '1');
        OSWait(K_TMO,1);
    }
}
```

```
    }
}

void TaskB(void)
{
    uint8 x,y;

    while (1)
    {
        x = random(16);
        y = random(8);
        OSDispChar(y, x, '2');
        OSWait(K_TMO,1);
    }
}

void TaskC(void)
{
    uint8 x,y;

    while (1)
    {
        x = random(16);
        y = random(8);
        OSDispChar(y, x, '3');
        OSWait(K_TMO,1);
    }
}

void TaskD(void)
{
    uint8 x,y;
    while (1)
    {
        x = random(16);
        y = random(8);
        OSDispChar(y, x, '4');
        OSWait(K_TMO,1);
    }
}

void TaskE(void)
{
    uint8 x,y;
```



```
while (1)
{
    x = random(16);
    y = random(8);
    OSDispChar(y, x, '5');
    OSWait(K_TMO,1);
}
}
```

实验四 软定时器实验

一. 实验目的

本程序展示了 Small RTOS51 中使用一个任务实现多个软定时器管理，软定时器的精度达到 1 个系统节拍。

二. 设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三. 实验步骤

1. 将 A2 区的 A0~A2 分别连接到 B3 区的 A0~A2。
2. 将 A2 区的 P10 连接到 B3 区的 RST。
3. 将 A3 区的 Y0 连接到 B3 区的/CS。
4. 将 A2 区的 A15~A10 分别连接到 A3 区的相关控制引脚，如下：

A15	---	/G2B
A14	---	/G2A
A13	---	G1
A12	---	C
A11	---	B
A10	---	A
5. 在 B3 区的 J92 插入图形液晶模块(单色，128×64 点)。
6. 将 B3 区的 J85 短接，A3 区的 JP4 短接。

四. 实验参考程序主要部分

```

/*****
**
**                      Small RTOS(51)
**                      The Real-Time Kernel(For Keil c51)
**                      (c) Copyright 2002-2002, chenmingji
**                      All Rights Reserved
**                      V1.20
*****/
#include "config.h"
uint8 TimeAdd[4];          //时间计数，用于纪录从复位以来的时间
void TimeSum(void);

/*****
** 函数名称: main
** 功能描述: 主函数，用户程序从这里执行
*****/
void main(void)
{

```

```

OSInit();

TMOD = (TMOD & 0XF0) | 0X01;
TH0 = (65536 - (11059200 / 12) / 100) / 256;
TL0 = (65536 - (11059200 / 12) / 100) % 256;
TR0 = 1;
ET0 = 1;
TF0 = 0;

OSSemCreate(ZL12864_SEM, 1);
LCM_DispIni();
OSDispClr();

OSTaskCreate(TimeSum, NULL, 0);
OSTaskCreate(SoftTimer, NULL, 1);

while(1)
{
    PCON = PCON | 0x01;                /* CPU 进入休眠状态 */
}

/*****计时*****/
void SoftTimerFunction2(void);

/*****
** 函数名称: SoftTimerFunction1
** 功能描述: 软定时器 0 运行的任务, 与软定时器 1 配合使 LCD 显示的 “:” 闪烁
*****/
void SoftTimerFunction1(void)
{
    OSDispChar(3, 7, ':');
    SoftTimerRun(1, (OS_TICKS_PER_SEC + 1) / 2, SoftTimerFunction2);
}

/*****
** 函数名称: SoftTimerFunction2
** 功能描述: 软定时器 1 运行的任务, 与软定时器 0 配合使 LCD 显示的 “:” 闪烁
*****/
void SoftTimerFunction2(void)
{
    OSDispChar(3, 7, ':');
    SoftTimerRun(0, OS_TICKS_PER_SEC / 2, SoftTimerFunction1);
}

```

/******

** 函数名称: SoftTimerFunction3

** 功能描述: 软定时器 2 运行的任务, 与任务 TimeSum 配合使 LCD 显示闪烁

*****/

void SoftTimerFunction3(void)

```
{
    OSDispChar(3, 5, ' ');
    OSDispChar(3, 6, ' ');
    OSDispChar(3, 8, ' ');
    OSDispChar(3, 9, ' ');
    SoftTimerRun(2, OS_TICKS_PER_SEC * 4, SoftTimerFunction3);
}
```

/******

** 函数名称: TimeSum

** 功能描述: 记录复位以来的时间的任务

*****/

void TimeSum(void)

```
{
    /* 初始化软定时器模块 */
    InitSoftTimer();

    /* 运行两个软定时器 */
    SoftTimerRun(1, OS_TICKS_PER_SEC / 2, SoftTimerFunction2);
    SoftTimerRun(2, OS_TICKS_PER_SEC * 4
+ OS_TICKS_PER_SEC / 2, SoftTimerFunction3);
    OSDispChar(3, 7, ':');
    while (1)
    {
        OSDispChar(3, 5, TimeAdd[0] + '0');
        OSDispChar(3, 6, TimeAdd[1] + '0');
        OSDispChar(3, 8, TimeAdd[2] + '0');
        OSDispChar(3, 9, TimeAdd[3] + '0');
        OSWait(K_TMO, OS_TICKS_PER_SEC);

        /* 计时并显示 */
        TimeAdd[3]++;
        if (TimeAdd[3] >= 10)
        {
            TimeAdd[3] = 0;
            TimeAdd[2]++;
            if (TimeAdd[2] >= 6)
            {
                TimeAdd[2] = 0;
            }
        }
    }
}
```

```

        TimeAdd[1]++;
        if (TimeAdd[1] >= 10)
        {
            TimeAdd[1] = 0;
            TimeAdd[0]++;
            if (TimeAdd[0] >= 10)
            {
                TimeAdd[0] = 0;
            }
        }
    }
}
}
}

```

SoftTimer. c 文件源代码

```
#define IN_SOFT_TIMER
```

```
#include "config.h"
```

/* 软定时器的数据结构 */

```
typedef SOFT_TIMER_MEM_SEL struct _TySoftTimerData
```

```

{
    uint8 Falg;                                /* 软定时器状态 */
    uint16 DelayTime;                          /* 软定时器运行时间 */
    void (const * Fuction)(void);             /* 软定时器溢出调用的函数 */
};

```

```
#if MAX_SOFT_TIMER > 0
```

```
uint16 SoftTimerCnt;                          /* 辅助定时器 */
```

```
uint16 SoftTimerThisDelay;                   /* 辅助定时器初始值 */
```

```
struct _TySoftTimerData SOFT_TIMER_MEM_SEL SoftTimerData[MAX_SOFT_TIMER];
```

/******

** 函数名称: SoftTimerSum

** 功能描述: 每次系统节拍处理时调用的函数，一个辅助定时器

```
void SoftTimerSum(void)
```

```

{
    if (--SoftTimerCnt == 0)
    {
        OSIntSendSignal(SOFT_TIMER_TASK_ID);
    }
}

```

/******

** 函数名称: InitSoftTimer

** 功能描述: 初始化软定时器模块

*****/

void InitSoftTimer(void)

```
{
    uint8 i;

    OS_ENTER_CRITICAL();

    for (i = 0; i < MAX_SOFT_TIMER; i++)
    {
        SoftTimerData[i].Falg = 0;
        SoftTimerData[i].DelayTime = 0;
        SoftTimerData[i].Fuction = NULL;
    }

    OS_EXIT_CRITICAL();
}
```

*****/

** 函数名称: SoftTimerRun

** 功能描述: 运行一个软定时器

** 输入: Index: 软定时器的索引

** Delay: 延时时间

** Fuction: 定时器溢出执行的函数

** 输出: NOT_OK: 参数错误

** SOFT_TIMER_OK: 操作正确

*****/

uint8 SoftTimerRun(uint8 Index, uint16 Delay, void (const * Fuction)(void))

```
{

#ifdef EN_SOFT_TIMER_CHK > 0
    if (Index >= MAX_SOFT_TIMER)
    {
        return NOT_OK;
    }
#endif

    if (Delay != 0 && Fuction != NULL)
    {
        OS_ENTER_CRITICAL();
        SoftTimerData[Index].Fuction = Fuction;
        SoftTimerThisDelay -= SoftTimerCnt;
        SoftTimerCnt = 0;
    }
}
```

```

        SoftTimerData[Index].DelayTime = Delay + SoftTimerThisDelay;
        SoftTimerData[Index].Falg &= ~SOFT_TIMER_TIMER_OUT;
        SoftTimerData[Index].Falg |= SOFT_TIMER_TIMER_RUN;
        OS_EXIT_CRITICAL();
        OSSendSignal(SOFT_TIMER_TASK_ID);
        return SOFT_TIMER_OK;
    }
    else
    {
        return NOT_OK;
    }
}

/*****
** 函数名称: SoftTimerStop
** 功能描述: 停止一个正在运行的软定时器
** 输 入: Index: 软定时器的索引
** 输 出: NOT_OK: 软定时器不存在
**        SOFT_TIMER_OK: 操作成功
** 全局变量: SoftTimerData
** 调用模块: OS_ENTER_CRITICAL(),OS_EXIT_CRITICAL()
*****/
uint8 SoftTimerStop(uint8 Index)
{
    #if EN_SOFT_TIMER_CHK > 0
        if (Index >= MAX_SOFT_TIMER)
        {
            return NOT_OK;
        }
    #endif

    OS_ENTER_CRITICAL();
    SoftTimerData[Index].DelayTime = 0;
    SoftTimerData[Index].Fuction = NULL;
    SoftTimerData[Index].Falg &=
    ~(SOFT_TIMER_TIMER_OUT | SOFT_TIMER_TIMER_RUN);
    OS_EXIT_CRITICAL();
    return SOFT_TIMER_OK;
}

/*****软定时器任务*****/
/*****
** 函数名称: SoftTimer
** 功能描述: 软定时器管理任务

```

*****/

```
void SoftTimer(void)
{
    uint16 ThisDelay;
    uint8 i;

    SoftTimerCnt = 0;
    while (1)
    {
        OS_ENTER_CRITICAL();

        // 找到定时最短的定时器
        ThisDelay = -1;
        for (i = 0; i < MAX_SOFT_TIMER; i++)
        {
            if (SoftTimerData[i].DelayTime != 0 &&
                SoftTimerData[i].DelayTime < ThisDelay)
            {
                ThisDelay = SoftTimerData[i].DelayTime;
            }
        }

        // 计算等待时间
        if (ThisDelay > -SoftTimerCnt)
        {
            SoftTimerCnt += ThisDelay;
        }
        else
        {
            SoftTimerCnt = 1;
        }
        SoftTimerThisDelay = ThisDelay;

        OSWait(K_SIG,0);
        ThisDelay = SoftTimerThisDelay;
        // 查询定时到的软定时器
        for (i = 0; i < MAX_SOFT_TIMER; i++)
        {
            SoftTimerData[i].Falg &= ~SOFT_TIMER_TIMER_OUT;
            if (SoftTimerData[i].DelayTime != 0)
            {
                if (SoftTimerData[i].DelayTime <= ThisDelay)
                {

```



```

        SoftTimerData[i].DelayTime = 0;
        if (SoftTimerData[i].Fuction != NULL)
        {
            SoftTimerData[i].Falg |= SOFT_TIMER_TIMER_OUT;
        }
    }
    else
    {
        SoftTimerData[i].DelayTime -= ThisDelay;
    }
}

SoftTimerThisDelay = 0;
OS_EXIT_CRITICAL();

// 执行软定时器指定的任务
for (i = 0; i < MAX_SOFT_TIMER; i++)
{
    if (((SoftTimerData[i].Falg & SOFT_TIMER_TIMER_OUT) != 0) &&
        (SoftTimerData[i].Fuction != NULL))
    {
        SoftTimerData[i].Falg &= ~SOFT_TIMER_TIMER_OUT;
        SoftTimerData[i].Fuction();
    }
}
}

#endif

```

五. 实验例程简析

本软件定时器所使用的存储空间由程序自己分配，并通过一个唯一的索引（即序号）来表示每一个软定时器。由于软定时器执行的任务时间等因素不确定，所以一般把这个任务优先级定得比较低（表示任务的优先权越高）。

要使用软定时器模块，首先需要初始化软定时器模块。这是由调用函数 **InitSoftTimer()** 实现的。初始化后，就可以调用函数 **SoftTimerRun()** 来运行一个软定时器，或是调用函数 **SoftTimerStop()** 来停止一个定时器。

在模块初始化之后，任务就可以调用函数 **SoftTimerRun()** 运行一个软定时器。**SoftTimerRun9()** 有三个参数。第一个参数 **Index** 用来指示使用的软定时器；第二个参数 **Delay** 用来设置定时器运行的时间，以系统时钟节拍为单位，范围大约是 0~65000；第三个参数 **Function** 是一个函数指针，指向一个无参数、无返回值的函数，当软定时器溢出时会调用这个函数。图 5.1 为 **SoftTimerRun()** 函数的伪流程图。

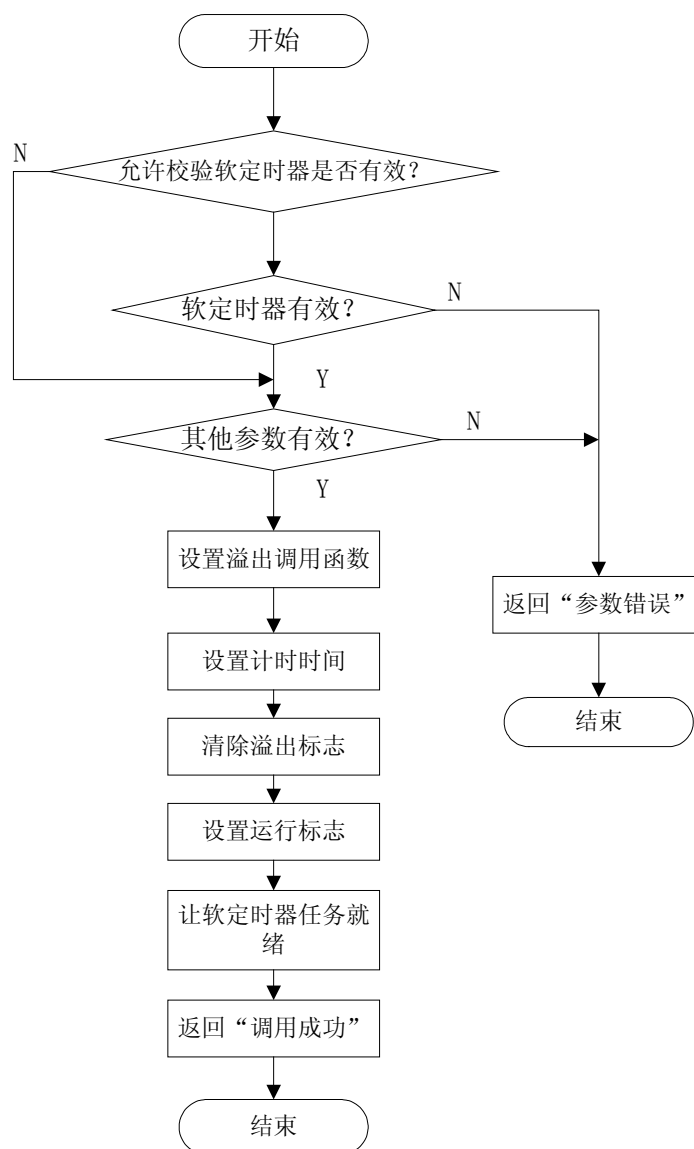


图 5.1 运行一个软定时器伪流程图

为了提高程序的执行效率，程序没有使用让每一个时钟节拍将每一个软定时器的计数值都减 1 的方法。这种方法虽然简单易懂，但会大量消耗 CPU 时间。而且，由于软定时器引入了不确定的函数，为了避免这些不确定的函数对系统造成大的影响，软定时器管理任务的优先级比较低。这样，当每个系统节拍都需要处理每一个软件定时器时，如果高优先级的任务运行时间超过一个时钟节拍，则软定时器就会出现误差，且误差会积累。本程序使用一个辅助减计数器来减少这些负面影响。软定时器管理任务首先要查找这些软定时器中等待时间最短的，然后将它的等待值赋予辅助减计数器。辅助减计数器每个系统节拍都减一，且与系统时钟节拍处理函数一起执行。当辅助减计数器溢出时通知软定时器管理任务，软定时器管理任务再进行其他处理。辅助减计数器的代码见函数 `SoftTimerSum()`。

图 5.2 为软定时器处理任务的流程图。

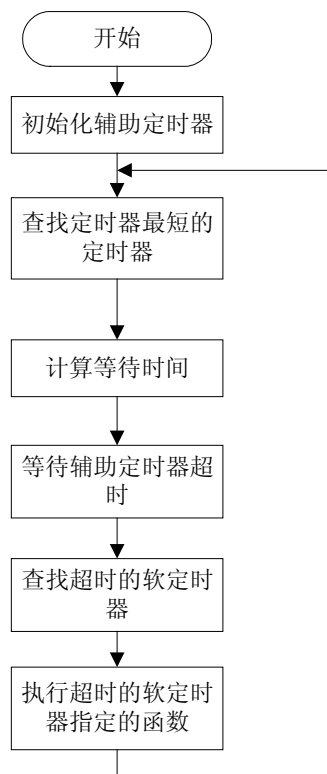


图 5.2 软定时器处理任务流程图

实验五 串口驱动程序实验

一. 实验目的

本示例程序展示了如何在 Small RTOS51 中编写串口通信驱动程序。

二. 实验设备及器件

IBM PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台

三. 实验步骤

1. 将 A2 区的 A0~A2 分别连接到 B3 区的 A0~A2。
2. 将 A2 区的 P10 连接到 B3 区的 RST。
3. 将 A3 区的 Y0 连接到 B3 区的/CS。
4. 将 A2 区的 A15~A10 分别连接到 A3 区的相关控制引脚，如下：

A15	----	/G2B
A14	----	/G2A
A13	----	G1
A12	----	C
A11	----	B
A10	----	A
5. 在 B3 区的 J92 插入图形液晶模块(单色，128×64 点)。
6. 将 B3 区的 J85 短接，A3 区的 JP4 短接。
7. 将程序下载到实验仪中，打开串口调试窗口设置波特率为 4800b/s 8 位数据位和 1 位停止位，在输入框中写入数据：

F0 F1 01 1E

按 16 进制方式向实验仪发送数据，LCD 将会显示 1。
PC 机向实验仪发送数据的通信协议：
0xf0+0xf1+要显示的数据（1 字节）+ 校验和（1 字节）

四. 实验参考程序主要部分

```
#include "config.h"
```

//指针的 NULL 为 0，这个变量占用 0 地址避免出现有效的 NULL 指针

```
uint8 OS_Q_MEM_SEL NotUse_at_ 0x0000;
uint8 OS_Q_MEM_SEL CommandData[16];      //给命令消息队列分配的队列空间
uint8 OS_Q_MEM_SEL SerialInData[16];      //给读串口消息队列分配的队列空间
uint8 OS_Q_MEM_SEL SerialOutData[32];     //给写串口消息队列分配的队列空间
void PutChar(uint8 Data);                  //发送一个字节
void Send(uint8 Data);                     //发送一个数据包
void Recuve(void);
void Command(void);
```

void TimeSum(void);

/******

** 函数名称: main

** 功能描述: 主函数，用户程序从这里执行

*****/

void main(void)

{

OSInit();

TMOD = (TMOD & 0XF0) | 0X01;

TH0 = (65536 - (11059200 / 12) / 100) / 256;

TL0 = (65536 - (11059200 / 12) / 100) % 256;

TR0 = 1;

ET0 = 1;

TF0 = 0;

OSSemCreate(ZL12864_SEM, 1);

LCM_DispIni();

OSDispClr();

OSDispStr(1,1, "Start...")

OSTaskCreate(Recuve, NULL, 0);

OSTaskCreate(Command, NULL, 1);

OSTaskCreate(TimeSum, NULL, 2);

while(1)

{

PCON = PCON | 0x01; /* CPU 进入休眠状态 */

}

}

/******主任务*****

** 函数名称: command

** 功能描述: 命令处理任务，高层命令由这个任务执行，相当于应用程序

*****/

void Command(void)

{

uint8 data temp;

while (1)

{

OSQPend(&temp,CommandData,0);

```

        /* 把接收到的数据在 LED 数码显示器上显示出来 */
        OSDispChar(3, 5, ' ');
        OSDispChar(3, 6, ' ');
        OSDispChar(3, 8, ' ');
        OSDispChar(3, 9, (temp % 10) + '0');
        if (temp >= 10)
        {
            OSDispChar(3, 8, ((temp / 10) % 10) + '0');
        }
        if (temp >= 100)
        {
            OSDispChar(3, 6, (temp / 100) + '0');
        }

        /* 把接收到的数据发送回去 */
        Send(temp);
    }
}

/*****串行通讯*****/
bit SerialCanSend = 1;

/*****
** 函数名称: SerialInit
** 功能描述: 初始化串行口
*****/
void SerialInit(void)
{
    SCON = 0x50;
    PCON = 0x80;
    TMOD = TMOD & 0x0f;
    TMOD = TMOD | 0x20;
    TH1 = 0xf4;                //com is 4800 b/s
    TL1 = 0xf4;
    TR1 = 1;
    ES = 1;
}

/*****
** 函数名称: Send
** 功能描述: 发送一个数据包
*****/
void Send(uint8 Data)
{
    PutChar(STARTBYTE1);

```

```

    PutChar(STARTBYTE2);
    PutChar(Data);
    PutChar(-(STARTBYTE1 + STARTBYTE2 + Data));
}

/*****
** 函数名称: PutChar
** 功能描述: 发送一个字节
*****/
void PutChar(uint8 Data)
{
    OS_ENTER_CRITICAL();
    if (SerialCanSend == 1)
    {
        SerialCanSend = 0;
        SBUF = Data;
    }
    else
    {
        OSQIntPost(SerialOutData,Data);
    }
    OS_EXIT_CRITICAL();
}

/*****
** 函数名称: Recuve
** 功能描述: 串口接收处理任务
*****/
void Recuve(void)
{
    uint8 data temp,temp1;
    uint8 Sum;

    /* 建立所需要的消息队列 */
    OSQCreate(CommandData,16);
    OSQCreate(SerialInData,16);
    OSQCreate(SerialOutData,32);
    SerialInit();                                /* 初始化串行口 */

    while (1)
    {
        OSQPend(&temp,SerialInData,0);          /* 接收一个字节 */

```

```

/* 处理通信协议 */
while (1)
{
    OSQPend(&temp1,SerialInData,0);          /* 接收一个字节 */
    if ((temp == STARTBYTE1) && (temp1 == STARTBYTE2))
    {
        break;                               /* 接收到起始字符 */
    }
    temp = temp1;
}
Sum = STARTBYTE1 + STARTBYTE2;
OSQPend(&temp1,SerialInData,0);          /* 接收数据 */
Sum += temp1;
OSQPend(&temp,SerialInData,0);          /* 接收校验和 */
Sum += temp;
if (Sum == 0)                             /* 检验接收到的数据包 */
{                                           /* 发送消息给主任务 */
    OSQPost(CommandData,temp1);
}
}
}

/*****
** 函数名称: comm
** 功能描述: 串口中断处理程序
*****/

#pragma disable         /* 除非最高优先级中断，否则，必须加上这一句 */
void comm(void) interrupt 4
{
    uint8 data temp;
    if (RI == 1)
    {
        OS_INT_ENTER();
        RI = 0;
        OSQPost(SerialInData,SBUF);
        OSIntExit();
        return;
    }
    if (TI == 1)
    {
        TI = 0;
        if (OSQAccept(&temp,SerialOutData) == OS_Q_OK)
        {
            SBUF = temp;
        }
    }
}

```



```

        else
        {
            SerialCanSend = 1;
        }
    }
}

/*****
** 函数名称: TimeSum
** 功能描述: 在 LCD 中显示 “: ”, 并闪烁
*****/
void TimeSum(void)
{
    while (1)
    {
        OSDispChar(3, 7, ':');
        OSWait(K_TMO, OS_TICKS_PER_SEC / 2);    /* 延时 0.5 秒 */
        OSDispChar(3, 7, ' ');
        OSWait(K_TMO, (OS_TICKS_PER_SEC + 1) / 2); /* 延时 0.5 秒 */
    }
}

```

五. 实验例程序简析

由于通信协议千变万化，不同的协议有不同的适用范围，不可能给出一个通用的串行通信协议驱动程序，只能给出一个例子。读者可以根据本实验例程的思路写出符合自己需要的独特的通信驱动程序。本例驱动程序使用的通信协议如下：

- (1) 波特率为 4800，有一个起始位、8 个数据位合 1 个停止位。
- (2) 数据包长度为 4 个字节。读者只需自行改造程序，把数据包长度增加后就可以在实际中使用了。
- (3) 起始字节为 STARTBYTE1、STARTBYTE2，中间字节不可能连续出现 STARTBYTE1、STARTBYTE2。在实际应用中，如果只是发送命令，就完全可以避免；如果是发送数据，就可能要加转义字符了，此时，起始字符其时可以只要一个字符。
- (4) 数据段为一个字符。读者增加数据包长度时应同时增加这个字段长度。
- (5) 校验字段为一个字符。加上这个字符后使得整个数据包（包括起始字节）按照字节依次加起来的和为 0。注意这是算术加而不是逻辑加（异或）。
- (6) 不提供握手，握手由用户程序提供。

本实验例程独立使用一个任务来处理串口通信协议，这个任务就是串口接收任务（Recuve），其工作流程如下图 5.3 所示：

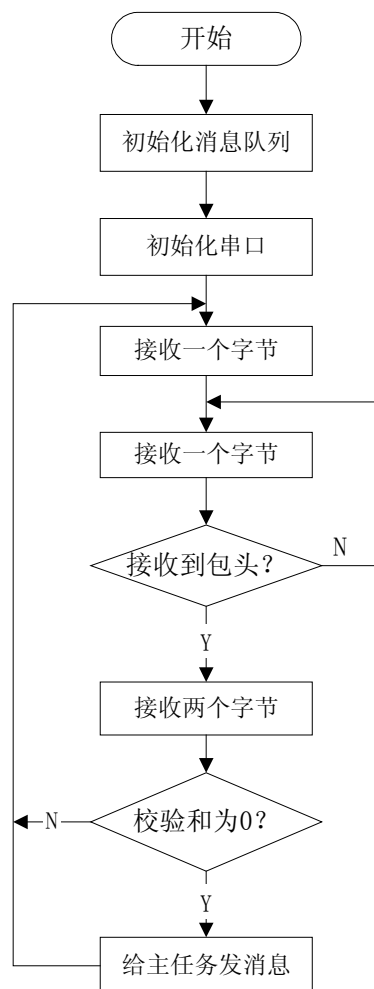


图 5.3 串口接收任务流程图


```

/*****
** 函数名称: init
** 功能描述: 初始化函数，一般在多任务环境启动前调用
*****/

void init(void)
{
    TMOD = (TMOD & 0XF0) | 0X01;
    TH0 = (65536 - (11059200 / 12) / 100) / 256;
    TL0 = (65536 - (11059200 / 12) / 100) % 256;
    TR0 = 1;
    ET0 = 1;
    TF0 = 0;
}

/*****
** 函数名称: main
** 功能描述: 主函数，用户程序从这里执行
*****/

void main(void)
{
    OSInit();
    init();
    OSTaskCreate(command,NULL,0);
    OSTaskCreate(Display,NULL,1);
    OSTaskCreate(TimeSum,NULL,2);
    while(1)
    {
        PCON = PCON | 0x01;                /* CPU 进入休眠状态 */
    }
}

/*****
**显示任务
*****/

void Display(void)
{
    while(1)
    {
        ShowCase[3] = 31;
        ShowCase[4] = 31;
        ShowCase[5] = 31;
        ShowCase[6] = 31;
        ZLG7290_SendBuf(ShowCase,8);
        OSWait(K_TMO,1);
    }
}

```

```

    }
}

/*****
** 函数名称: command
** 功能描述: 命令处理任务，高层命令由这个任务执行，相当于应用程序
*****/

void command(void)
{
    uint8 data temp;

    OSSemCreate(IICSem,1);
    OSSemCreate(Cat24WCxxSem,1);
    OSQCreate(SerialInData,10);
    OSQCreate(SerialOutData,10);
    SerialInit();

    while (1)
    {
        OSQPend(&temp,SerialInData,0);

        /* 把接收到的数据在 LED 数码显示器上显示出来 */
        ShowCase[0] = temp % 10;
        ShowCase[1] = (temp / 10) % 10;
        ShowCase[2] = temp / 100;

        /* 把接收到的数据写入 cat24wc02 */
        EepRomBuf[0] = 0;                //外部地址
        EepRomBuf[1] = 1;                //写多少字节
        EepRomBuf[2] = 0;                //开始写地址
        EepRomBuf[3] = temp;            //数据
        OSCat24WCxxWrite(EepRomBuf);
        OSWait(K_TMO,10);

        /* 从 cat24wc02 读出刚才写入的数据*/
        EepRomBuf[0] = 0;                //器件外部地址
        EepRomBuf[1] = 1;                //读多少字节
        EepRomBuf[2] = 0;                //开始读地址
        OSCat24WCxxRead(EepRomBuf);

        /* 把读出的数据发送回去 */
        Send(EepRomBuf[0]);
        OSWait(K_TMO,10);
    }
}

```

```
}

/*****
****
** 函数名称: SerialInit
** 功能描述: 初始化串行口
****
*****/

void SerialInit(void)
{
    SCON = 0x50;
    PCON = 0x80;
    TMOD = TMOD & 0x0f;
    TMOD = TMOD | 0x20;
    TH1 = 0xf4;                //波特率为 4800 b/s
    TL1 = 0xf4;
    TR1 = 1;
    ES = 1;
}

/*****
****
** 函数名称: Send
** 功能描述: 发送一个数据包
** 输 入: Data: 指向要发送的数据指针
****
*****/

void Send(uint8 Data)
{
    PutChar(Data);
}

/*****
****
** 函数名称: PutChar
** 功能描述: 发送一个字节
****
*****/

void PutChar(uint8 Data)
{
    OS_ENTER_CRITICAL();
    if (SerialCanSend == 1)
    {
        SerialCanSend = 0;
```

```

        SBUF = Data;
    }
    else
    {
        OSQIntPost(SerialOutData,Data);
    }
    OS_EXIT_CRITICAL();
}

/*****
** 函数名称: comm
** 功能描述: 串口中断处理程序
*****/
#pragma disable             /* 除非最高优先级中断，否则，必须加上这一句 */
void comm(void) interrupt 4
{
    uint8 data temp;
    if (RI == 1)
    {
        OS_INT_ENTER();
        RI = 0;
        OSQPost(SerialInData,SBUF);
        OSIntExit();
        return;
    }
    if (TI == 1)
    {
        TI = 0;
        if (OSQAccept(&temp,SerialOutData) == OS_Q_OK)
        {
            SBUF = temp;
        }
        else
        {
            SerialCanSend = 1;
        }
    }
}

/*****
** 函数名称: TimeSum
** 功能描述: 使 LED 显示 0~9 的任务
*****/
void TimeSum(void)

```

```
{
    while (1)
    {
        ShowCase[7] ++;
        if(ShowCase[7] > 9)
        {
            ShowCase[7] = 0;
        }
        OSWait(K_TMO,OS_TICKS_PER_SEC);          /* 延时 1 秒 */
    }
}
```

五. 实验例程序简析

由于对 I2C 总线驱动程序的操作具有不可重入性，故使用了信号量 Cat24WcxxSem 来实现互斥操作，为了使用信号量，必须在 OS_cfg.h 文件中将 EN_OS_SEM 置为 1。

以下为多任务对 Cat24WCxx 的写操作代码

```
#define OSCat24WCxxWrite(a) \
if (OSSemPend(Cat24WCxxSem,10) == OS_SEM_OK) \
{ \
    Cat24WCxxWrite (a); \
    OSSemPost(Cat24WCxxSem); \
}
```

以下为多任务对 Cat24WCxx 的读操作代码：

```
#define OSCat24WCxxRead(a) \
if (OSSemPend(Cat24WCxxSem,10) == OS_SEM_OK) \
{ \
    Cat24WCxxRead(a); \
    OSSemPost(Cat24WCxxSem); \
}
```

本程序中也使用的消息队列来缓冲数据，故也必须在 OS_cfg.h 文件中将 EN_OS_Q 置为 1，以使 Small RTOS 51 配置为允许使用消息队列。

从串口接收到的数据按 10 进制的方式在 LED 上显示出来。最左边的 LED 循环的显示 0~9 用于表示系统正在运行之中。

实验七 PDIUSBD12 USB 驱动程序演示实验

一. 实验目的

PDIUSBD12 (简称 D12) USB 芯片符合 USB1.1 协议, 广泛应用于各种的 USB 设备。本实验程序展示了如何在 Small RTOS 51 操作系统中实现 D12 驱动程序的编写。

示例主要实现了如下功能: USB 设备的枚举, 通过 D12 的端点 2 接受来自上位机的数据, 并通过单片机的串口转发回上位机, 同时驱动 ZLG7290 显示 0~9 的数字。

二. 实验设备及器件

PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
USB D12 PARK 模块	一台
USB 连接线	一根

三. 实验步骤

1. D12 PARK 插到 A6 区的排针内;
2. 使用导线把 A2 区的 INT1 与 A6 区的 P1_INT0 相连;
3. 使用导线把 A2 区的 T0 与 A6 区的 P1_IO2 相连;
4. 使用导线把 A2 区的 T1 与 A6 区的 P1_IO5 相连;
5. 使用导线把 A2 区的 A15 与 A6 区的 P1_CS 1 相连;
6. 使用导线把 A2 区的 P16、P17 分别与 D5 区的 SCL、SDA 相连;
7. 使用导线把 A2 区的 P10 与 D5 区的/RST 相连, 并短接 D5 区的 JP1;
8. 使用导线连接 A2 区的 P14 与 B10 区的 ZDJ_A;
9. B10 区的 ZDJ_B 连接到 C1 的 GND;
10. 短接 B10 区 JP18 的电机电源跳线

四. 实验参考程序主要部分

```
/******  
Small RTOS(51)  
The Real-Time Kernel(For Keil c51)  
(c) Copyright 2002-2004, chenmingji  
All Rights Reserved  
V1.20  
*****/  
#include "config.h"  
/******  
声明全局变量  
*****/  
extern EPPFLAGS bEPPflags; /*USB 事件标志*/  
extern uint8 xdata GenEpBuf[]; /*信号包缓冲区 (命令区)*/  
extern uint8 xdata EpBuf[]; /*信号包缓冲区 (数据区)*/
```

```

extern uint16 data D12_DATA;          /*D12 数据的地址变量*/
extern uint16 data D12_COMMAND;      /*D12 命令的地址变量*/
//指针的 NULL 为 0，这个变量占用 0 地址避免出现有效的 NULL 指针
uint8 OS_Q_MEM_SEL NotUse _at_ 0x0000;

uint8 xdata ShowCase[8];
uint8 xdata PWMH;                    //高电平脉冲的个数
uint8 xdata PWM;                     //PWM 周期
uint8 xdata COUNTER;
uint8 key_buf[2];

void enumerate_task(void);
void USB_REC_task(void);
void change_speed_task(void);
void get_key_task(void);

void INTT1() interrupt 3
{
    COUNTER++;
    if(COUNTER < PWMH)
    {
        P1_4 = 1;                    //P1.4 变为高电平
    }
    else
    {
        P1_4 = 0;
    }
}

void usb_ISR(void) interrupt 2
{
    OS_INT_ENTER();
    EX1 = 0;
    OSSemIntPost(D12_isr);
    OSIntExit();
}

/*****
**名称: init_port()
**功能: 端口初始化, D12SUSPD 复位为 0
*****/
void init_Hard()
{
    P0 = 0XFF;

```

```
P1 = 0XFF;
P2 = 0XFF;
P3 = 0XFF;
D12SUSPD = 0;
}
```

```
/******
```

```
** 函数名称: init
```

```
** 功能描述: 初始化函数，一般在多任务环境启动前调用
```

```
*****888*****/
```

```
void init(void)
```

```
{
    TMOD &= 0XF0;
    TMOD |= 0X01;

    // 每 10ms 发生一次 T0 中断
    TL0 = (65536 - (11059200 / 12) / 100) % 256;
    TH0 = (65536 - (11059200 / 12) / 100) / 256;
    ET0 = 1;
    TR0 = 1;
    PT0 = 0;
    EA = 1;
}
```

```
void control_init(void)
```

```
{
    PWMH=0x00;
    COUNTER=0x01;
    PWM=255;
    TMOD = TMOD & 0x0f;
    TMOD = TMOD | 0x20;    //定时器 1 在模式 2 下工作
    TL1=0x00;
    TH1=0x00;             //自动重装的值
    TR1=1;
    ET1 = 1;
}
```

```
/******
```

```
** 函数名称: void init_display(void)
```

```
** 功能描述: 初始化 LED 显示
```

```
*****/
```

```
void init_display(void)
```

```
{
    uint8 i;
```

```

    ZLG7290_RST = 0;
    _nop_();
    _nop_();
    ZLG7290_RST = 1;
    for (i = 0; i < 8; i++)
    {
        ShowCase[i] = 31;
    }
    ZLG7290_SendBuf(ShowCase,8);
}

void main(void)
{
    OSInit();
    init_Hard();                /*初始化硬件*/
    init();
    control_init();
    IT1 = 0;
    EX1 = 1;
    PX1 = 1;
    init_display();

    D12_DATA = 0x7002;          /*定义数据地址*/
    D12_COMMAND = 0x7003;      /*定义命令地址*/

    D12_SetDMA(0x0);           /*不使用 DMA 功能*/
    bEPPflags.value = 0;       /*初始化 USB 寄存器*/

    OSTaskCreate(USB_REC_task , NULL , 0);    //创建 USB 中断处理任务

    while(1)
    {
        PCON = PCON |0x01 ;    /* CPU 进入休眠状态 */
    }
}

/*****
** 函数名称: void enumerate(void)
** 功能描述: 完成 USB 请求处理的任务
*****/
void enumerate_task(void)
{
    //创建其他优先级低的任务
    OSTaskCreate(get_key_task, NULL,2);

```

```

    OSTaskCreate(change_speed_task, NULL,3);
while(1)
{
    OSWait(K_TMO, 2);
    IT1 = 0;
    EX1 = 1;
    reconnect_USB();
    while(USB_VIN)
    {
        usbserve();
        OSWait(K_TMO, 1);
    }
}
}

/*****
** 函数名称: void USB_REC(void)
** 功能描述: USB 中断处理任务
*****/
void USB_REC_task(void)
{
    OSSemCreate(D12_isr , 0);
    OSSemCreate(D12_use , 1);
    OSTaskCreate(enumerate_task , NULL ,1); //创建 USB 请求处理任务

while(1)
{
    OSSemPend(D12_isr , 0);
    OSSemPend(D12_use , 0);
    fn_usb_isr();
    OSSemPost(D12_use);
    EX1=1;
}
}

/*****
** 函数名称: void change_speed_task(void)
** 功能描述: 修改马达速度任务。
*****/
void change_speed_task(void)
{
while(1)
{
    OS_ENTER_CRITICAL();

```

```

        if( bEPPflags.bits.ep1_rxdone)
        {
            PWMH = GenEpBuf[0];
            bEPPflags.bits.ep1_rxdone = 0;
        }
        OS_EXIT_CRITICAL();
        ShowCase[0] =  PWMH %10;
        ShowCase[1] =  (PWMH /10)%10;
        ShowCase[2] =  (PWMH /100)%10;

        OS_ENTER_CRITICAL();
        ZLG7290_SendBuf(ShowCase,8);
        OS_EXIT_CRITICAL();
        OSWait(K_TMO,9);
    }
}

/*****
** 函数名称: void get_key_task(void)
** 功能描述: 获取按键值任务。
*****/
void get_key_task(void)
{
    while(1)
    {
        OS_ENTER_CRITICAL();
        key_buf[0] = ZLG7290_GetKey();
        OS_EXIT_CRITICAL();

        OSWait(K_TMO, 1);
        if(key_buf[0] == 0)
        {
            continue;
        }
        OS_ENTER_CRITICAL();
        key_buf[1] = ZLG7290_GetKey();
        OS_EXIT_CRITICAL();
        if(key_buf[0] != key_buf[1])
        {
            OS_ENTER_CRITICAL();
            D12_WriteEndpoint(3, 1, &key_buf[0]);
            OS_EXIT_CRITICAL();
        }
    }
}

```

```
}
```

五. 实验示例程序简析

本由于驱动程序中使用了 USB51s.lib 库, 故关于该库的使用方法详见实验仪配套光盘中的《PDIUSBD12 固件编程指南》和书《PDIUSBD12 固件编程与驱动开发》。这里仅仅就叙述如何在 Small RTOS51 下编写 D12 USB 驱动程序。

由于对 PDIUSBD12 的操作是一种不能重入的操作, 所以必须要在这一次操作完成后才能进行下一次对 PDIUSBD12 的操作。因此, 对 PDIUSBD12 器件的操作采用了一个信号量 (D12_use) 来管理。这种信号量管理的方法就像有很多个门共用一把钥匙, 必须一次只能开一个门, 而开这个门的先后则由任务的优先级来决定。

在驱动中使用了 2 个信号量, 其含义如下:

- **D12_use:** 不为 0, 表示 PDIUSBD12 处于空闲状态, 可以使用; 为 0, 则表示 PDIUSBD12 正在使用中。不允许其他任务对 PDIUSBD12 进行操作。此信号量在创建时被初始化为 1。
- **D12_isr:** 不为 0, 表示检测到 PDIUSBD12 产生的 USB 中断请求; 为 0, 表示没有 USB 中断请求或者 USB 中断请求已被执行处理。此信号量在创建时被初始化为 0。

控制马达转速的方法很多, PWM (脉宽调制) 方式是比较常见的一种, 实现较简单并且效果好。在 DP-51PRO.NET 中可以实现 PWM 控制马达转速实验, 其马达控制电路参见综合实验 7 电机实验。

本例程中, PWM 波形由定时器 1 的周期性中断产生。其代码实现见程序清单 5.4。定时器 1 中断服务函数 (INTT1) 对全局变量 COUNTER 进行自增计数, 并与 PWMH 进行比较, 根据比较结果控制 P1_4(波形输出引脚)输出电平的高低, 从而实现 PWM 的输出。马达的转速与 PWM 波形中的高电平之占空比成正比。

六. 程序演示

由于本示例程序需要跟 PC 机进行数据通讯, 故必须与 PC 机相关软件配合使用才能实现所有的功能。上位机软件可以使用配套光盘中的 usb 调试助手 (1.2 版) (有关 usb 调试助手的使用请见其帮助文件), 该软件用于向实验仪发送控制数据和接收发上来的按键值数据。使用 usb 调试助手软件之前还必须安装 D12 驱动程序方可正常使用。

首先将 USB_D12.hex 文件下载到实验仪中, 运行单片机中的程序。单片机将配置 PDIUSBD12 并进行 USB 枚举, USB D12PARK 模块上的 LED 灯将闪烁几次。、两、三秒后, 若 USB 枚举成功, 模块上的 LED 将常亮, 表示 USB 连接正常。此时打开 usb 调试助手, 选择名称为 Philips PDIUSBD12 Smart Evaluation Board 的设备, 设置接收端点和发送端点都为端点 1, 设置接收和发送管道大小为 16 字节。按下启动按钮, 这样便与实验仪建立的 USB 连接。向发送区填入 1 个字节的数据, 然后发送到实验仪, 这样便可实现马达的变速。按下实验仪中 D7 区的按键, 在 PC 上便可接收到该按键的键值。

实验八 SJA1000_CAN 驱动程序演示实验

一. 实验目的

本驱动程序展示了如何在 Small RTOS 中编写 SJA1000 的驱动程序。通过调用 CAN 程序库 SJA1000_PELI.LIB 的基本函数，实现实验板上 CAN 节点的初始化以及 CAN 节点数据收发测试。

二. 实验设备及器件

PC 机	一台
DP-51PROC 单片机综合仿真实验仪	一台
CAN PARK 模块	一台
CAN 连接线	一根

三. 实验步骤

- 1、将 CAN—bus PARK 插入到 A6 区中,用导线连接 A6 区的 P1_IO2 到 A2 区的 P10, 连接 A6 区的 P1_CS1 到 A2 区的 A15。
- 2、使用导线把 A2 区的 P16 和 P17 分别于 D5 区的 SCL 和 SDA 相连。使用导线把 D5 区的/RST 与 VCC 相连。
- 3、由于本程序使用中断方式响应 SJA1000 中断，故将 A5 区的 P1_INT 接到 A2 区的 INT0。
- 4、利用 CAN 连接线将两台已经安装了 CAN-Bus 模块的 DP-51PROC 连接起来，以组成简单的 CAN 网络实现 CAN 的接收和发送。
- 5、本驱动程序已经将输出文件路径设置为“E:\Temp”，用户可自行更改输出文件路径。将路径“E:\Temp”中的 CAN. hex 文件下载到两台 DP-51PROC 中运行。

四. 实验参考程序主要部分

```

/*****
*描述：    独立的 CAN 控制器 SJA1000PeliCAN 在 small rtos 中的应用示例
*文件名：    PELIRTOS.c
*应用语言：  KEIL C51
*应用系统：  small rtos
*版本：    V1.0
*广州周立功单片机发展有限公司 保留所有的版权
*****/

#define    _TIME_MODULE_H
#define    _SERIAL_H

/*****
**
**                      导入头文件
**
*****/

#include "INCLUDES.h"
#include    "Sja1000_peli.h"
sfr    IPH=0xb7;

```



```

sbit RESET_PIN=P1^0;
// 验收代码/屏蔽寄存器的内容 (4+4)
uint8 xdata Send_CAN_Filter[8]={0xf0,0xf0,0xf0,0xff,0xff,0xff,0xff,0xff};

// 帧信息和标示码 (1+4) 分别对应 TX,TX1,TX2,TX3,TX4
uint8 xdata Send_CAN_Info_ID[3]={0xc7,0x0A,0x0B};
uint8 xdata Recv_CAN_Info_ID[3];

// 待发送数据 (8)
uint8 xdata
Send_CAN_Data[13]={0xc7,0x0A,0x0B,0x04,0x05,0x06,0x07,0x08,0x07,0x08,0x07,0x08,
0x08};
uint8 xdata Recv_CAN_Data[14];

uint8 xdata time_Counter=0;
uint8 xdata BTR0,BTR1;
uint16 xdata *p;
uint8 xdata disp_buf[8];

void CAN_Send(void);
void display(void);
void CAN_Rcv(void);
void TimeSum(void);
void Delay_ms(uint8 j);
void SJA1000_Config_Normal(void);

void Init(void)
{
    CKCON=1; //应用 6clock
    TMOD = (TMOD & 0XF0) | 0X01;
    TCON=TCON|0x04; //MCU 的 INT1 下降沿触发,INT0 电平触发
    TH0 = (65536 - (11059200 / 12) / 100) / 256;
    TL0 = (65536 - (11059200 / 12) / 100) % 256;
    TR0 = 1;
    ET0 = 1;
    TF0 = 0;
}
/*****
**函数名称: void CAN_Init(void)
**功能描述: 复位 SJA1000, 并设置其工作在正常模式
*****/
void CAN_Init(void)
{

```

```

    RESET_PIN=0;                                //将 SJA1000 的复位线与 P1.0 相连接
    Delay_ms(1);
    RESET_PIN=1;                                //控制 P1.0 来实现 SJA1000 的复位
    SJA_CS_Point=&CAN_SJA_BaseAdr;
    SJA1000_Config_Normal();
    WriteSJAReg(REG_CAN_IER,RIE_BIT);    //使能 SJA1000 接收中断
    EX0=1;
}

/*****
** 函数原型: void Delay_ms(uchar j)
** 功能描述: 该函数用于不精确的延时。在 12M,6CLK 下, 大约延时 j*1ms
*****/
void Delay_ms(uint8 j)
{
    uint8    k,l;
    for(l=0;l<=j;l++)
    {
        for(k=0;k<=250;k++)
        {
            ;
        }
    }
}

/*****
** 函数原型: void    SJA1000_Config_Normal(void)
** 功能描述: 对 SJA1000 的正常模式的初始化配置
*****/
void SJA1000_Config_Normal(void)
{
    BTR0=0x00;
    BTR1=0x14;                                //设置为 80k 波特率通信
    SJAEntryResetMode();                      //进入复位模式
    WriteSJAReg(REG_CAN_CDR,0xc8);            //配置时钟分频寄存器, 选择 PeliCAN 模式
    WriteSJAReg(REG_CAN_MOD,0x01);            //配置模式寄存器, 选择双滤波、正常模式
    WriteSJARegBlock(16,Send_CAN_Filter,8);    //配置验收代码/屏蔽寄存器
    WriteSJAReg(REG_CAN_BTR0,BTR0);            //配置总线定时器 0
    WriteSJAReg(REG_CAN_BTR1,BTR1);            //配置总线定时器 1
    WriteSJAReg(REG_CAN_OCR,0x1a);            //配置输出管脚
    SJAQuitResetMode();                        //退出复位模式,进入工作模式
}

/*****

```

** 函数原型: void CAN_Data_Send(void)

** 功能描述: SJA1000 的单次发送子函数（注意在这个函数的末尾要置位接收中断）

*****/

void CAN_Data_Send(void)

```
{
    // 发送数据
    WriteSJAReg(REG_CAN_IER,0x02);           //使能 SJA1000 发送中断位
    WriteSJARegBlock(16,Send_CAN_Data,13);

    WriteSJAReg(REG_CAN_CMR,1);              //使能发送请求
    Delay_ms(10);
    WriteSJAReg(REG_CAN_IER,RIE_BIT);        //使能 SJA1000 接收中断
}
```

*****/

** 函数原型: void SJA1000_INT0 (void) interrupt 0

** 功能描述: SJA1000 中断响应函数

*****/

void SJA1000_INT0(void) interrupt 0

```
{
    OS_INT_ENTER();
    EX0 = 0;
    OSIntSendSignal(0); //无条件的令 CAN 接收中断处理任务(CAN_Rcv ())处于就绪状态
    //由于 CAN_Rcv ()的优先级最高，故中断退出后立刻执行
    CAN_Rcv ().
    OSIntExit();
}
```

void main(void)

```
{
    uint8 i ;
    OSInit();
    Init();
    CAN_Init();

    //初始化显示缓存
    for(i=0;i<8;i++)
    {
        disp_buf[i]=31;
    }

    //创建任务
    OSTaskCreate(CAN_Rcv, NULL, 0);
}
```

```

    OSTaskCreate(display, NULL, 1);
    OSTaskCreate(CAN_Send, NULL, 2);
    OSTaskCreate(TimeSum, NULL, 3);
    while(1)
    {
        PCON = PCON | 0x01; //令 CPU 进入睡眠状态
    }
}

/*****
**键值发送任务
*****/
void CAN_Send(void)
{
    uint8 key_data;
    while(1)
    {
        key_data = ZLG7290_GetKey();
        if(key_data)
        {
            Send_CAN_Data[3] = key_data;
            CAN_Data_Send();
            disp_buf[4] = key_data % 10;
            disp_buf[5] = (key_data / 10) % 10;
        }
        OSWait(K_TMO,5);
    }
}

/*****
**显示任务
*****/
void display(void)
{
    while(1)
    {
        OSWait(K_TMO,5);
        OS_ENTER_CRITICAL();
        ZLG7290_SendBuf(&disp_buf[0], 8);
        OS_EXIT_CRITICAL();
    }
}

/*****
**CAN 接收中断处理任务
*****/

```

```

*****/
void CAN_Rcv(void)
{
    while(1)
    {
        OSWait(K_SIG, 0);           //挂起当前任务，等待唤醒信号
        if(ReadSJAREg(REG_CAN_IR)&0x01)
        {
            //数据接收一定在释放缓冲区之前，释放后数据不确定
            ReadSJAREgBlock(16,Recv_CAN_Data,13);
            WriteSJAREg(REG_CAN_CMR,4);      //释放 SJA1000 接收缓冲区
            disp_buf[0] = Recv_CAN_Data[3] % 10;
            disp_buf[1] = (Recv_CAN_Data[3] / 10) % 10;
        }
        EX0 = 1;
    }
}

/*****
* *计数任务
*****/
void TimeSum(void)
{
    while(1)
    {
        OSWait(K_TMO, 5);
        disp_buf[7]++;
        if(disp_buf[7] > 9)
        {
            disp_buf[7] = 0;
        }
    }
}

```

五. 实验例程简析

本驱动程序采用中断方式接收 CAN 总线数据。采用中断的方式，可以提高系统的实时性。特别的在接收数据的时候，采用中断方式可以在效率和实时性上比采用非中断方式得到很大的提高。

按下 D5 区的按键时，左边的 LED 将显示按键键值，同时程序调用 CAN_Data_Send () 将检测到的键值通过 CAN 总线发送到另一台实验仪上。实验仪将从 CAN 总线上接收到的键值数据显示在 D5 区的右边 LED 上。

D5 区中最右边的 LED 管循环地显示 0~9，目的是为了显示系统正在运行。

有关 SJA1000_PEI 库的使用请阅读配套光盘中的《SJA1000_PEI 库说明及其使用》文档。

附录 Small RTOS51 使用许可协议

复制、发布和修改的条款和条件:

(1) 任何人可以免费获取 Small RTOS51 源代码用于非商业目的而无需作者同意。一旦您得到这些源代码,表示您接受本协议。如果您不同意本协议,请销毁它,或是将其退回原购买处。

(2) 任何人可以分发 Small RTOS51 源代码,条件是要完整分发,不能缺少任何一个文件,目录结构也不得改变,也不能修改它的任何部分,

(3) 如果要将 Small RTOS51 用于您的项目中,而您的项目有商业目的,您也无需支付任何费用,但您必须将项目的简要介绍、技术特点、应用范围、Small RTOS51 的使用情况等说明文字交给作者,联系方法请阅读 readme.txt 文件。

(4) 法律与本许可协议冲突的地区暂时不允许使用本产品。如果您确实需要,可以与作者联系。

没有担保

由于 Small RTOS51 只是最终产品的程序的一部分,作者不提供任何类型的担保。不论是明确的,还是隐含的。包括但不限于合适特定用途的保证。全部的风险,由使用者来承担。如果程序出现缺陷,使用者承担所有必要的服务、修改和改正的费用。