

目 录

第 1 章 单片机 8051 简介	(1)
1.1 8051 特性	(1)
1.1.1 8051 系列成员	(2)
1.2 其他 8051 兼容芯片简介	(3)
1.2.1 ATMEL 89C51 系列单片机	(3)
1.2.2 DALLAS DS80C320 单片机	(3)
1.2.3 WINBOND W78C31 单片机	(4)
1.3 8051 引脚说明	(4)
1.4 系统重置	(7)
1.5 内存空间	(7)
1.5.1 只读存储器	(8)
1.5.2 随机存储器	(9)
1.5.3 地址 00H~7FH	(10)
1.5.4 特殊功能寄存器	(11)
1.5.5 外部随机存储器	(13)
1.6 8051 内部控制寄存器	(15)
1.6.1 IE; 中断允许寄存器	(15)
1.6.2 IP; 中断优先次序寄存器	(16)
1.6.3 TMOD; 定时器模式控制寄存器	(16)
1.6.4 TCON; 计时控制寄存器	(17)
1.6.5 SCON; 串行端口控制寄存器	(18)
1.6.6 PCON; 电源控制寄存器	(18)
1.7 习题	(19)
第 2 章 实验环境设定	(20)
2.1 实验必备的硬件配置	(20)
2.2 软件使用工具	(22)
2.3 硬件接口卡	(23)
第 3 章 8051 C 编译器使用说明	(26)
3.1 MICRO-C51 编译器特性	(26)
3.1.1 MICRO-C51 编译器特性	(26)
3.2 MICRO-C51 编译器组成	(27)
3.2.1 磁盘内容	(28)
3.2.2 代码兼容性	(31)
3.3 内存模式	(31)

3.3.1	极小型模式	(32)
3.3.2	小型模式	(32)
3.3.3	压缩型模式	(33)
3.3.4	中型模式	(33)
3.3.5	大型模式	(33)
3.3.6	局部变量存取	(34)
3.3.7	全局变量存放	(34)
3.4	编译程序	(34)
3.4.1	前置处理器	(35)
3.4.2	编译器	(35)
3.4.3	最优化处理器	(35)
3.4.4	汇编语言编译器	(35)
3.4.5	链接器	(35)
3.5	综合的编译程序	(35)
3.5.1	CC51 指令格式	(36)
3.5.2	编译器出现的错误消息	(37)
3.6	工作环境设置	(37)
3.7	操作实例	(39)
3.8	以 ROM 模拟器来做程序测试	(47)
3.8.1	X.BAT 内容	(48)
3.8.2	T.BAT 内容	(49)
3.9	使用 89C51 烧录模拟器来做程序测试	(51)
3.9.1	X1.BAT 内容	(52)
3.10	MICRO-C51 程序设计技巧	(53)
3.10.1	存取 8051 单片机特殊功能寄存器	(53)
3.10.2	位的控制	(54)
3.10.3	中断子程序的设计	(55)
3.10.4	内存应对式 I/O	(56)
3.10.5	程序中加入汇编语言语句	(57)
第 4 章	8051 多功能控制板设计	(59)
4.1	控制板设计概念	(59)
4.1.1	单片机控制板基本功能	(59)
4.2	8051 多功能控制板特性	(60)
4.3	8051 基本控制电路	(61)
4.4	8051 内存扩充设计	(62)
4.4.1	系统总线	(63)
4.4.2	内存使用	(64)
4.4.3	I/O 解码	(66)
4.5	通信接口	(66)
4.6	LCD 接口	(67)
4.6.1	LCD 特性	(67)
4.6.2	引脚说明	(68)

4.7	8255 接口	(69)
4.8	7 段数码管及按键输入	(70)
4.9	D/A 语音接口	(71)
4.9.1	引脚说明	(72)
4.9.2	DAC0800 接口设计	(73)
4.9.3	音频放大电路	(74)
4.10	声效接口	(75)
4.10.1	芯片特性	(75)
4.10.2	内部结构	(76)
4.10.3	引脚说明	(77)
4.10.4	可编程声效发生器接口设计	(78)
4.11	LED 显示及蜂鸣器控制	(80)
4.12	电源控制电路	(81)
第 5 章 8051 多功能控制板制作及测试		(83)
5.1	8051 多功能控制板快速安装及测试	(83)
5.2	单片机基本工作验证	(84)
5.3	测试 RS232 接口	(85)
5.4	测试 8255 接口	(86)
5.5	共阴极 7 段数码管测试	(87)
5.6	测试按键输入	(87)
5.7	测试蜂鸣器	(87)
5.8	测试 8 只 LED	(87)
5.9	声效测试	(88)
5.10	测试 D/A 接口	(88)
5.11	测试 8255 I/O 扩充接口	(89)
5.12	测试 LCD 接口	(89)
5.13	加装电源控制	(89)
第 6 章 8255 接口控制		(91)
6.1	8255 简介	(91)
6.2	8255 引脚说明	(92)
6.3	8255 工作说明	(93)
6.3.1	模式设定	(94)
6.4	8255 工作模式	(95)
6.5	8255 模式 1 工作	(96)
6.5.1	模式 1 的输入控制方式	(97)
6.5.2	模式 1 的输出控制方式	(99)
6.5.3	模式 1 的组合	(101)
6.6	8255 模式 2 工作	(101)
6.6.1	模式 2 的组合方式	(102)
6.7	8255 端口 C 的交互式控制信号状态读取	(103)

6.8	8255 接口电路测试	(103)
6.8.1	8255 接口电路测试功能	(104)
6.8.2	P51 I/O 控制头文件 P51.H	(104)
6.9	习题	(107)
第 7 章 多功能控制板基本 I/O 功能		(108)
7.1	单板上工作指示 LED	(108)
7.2	“走马灯”式电路控制	(109)
7.3	读取 DIP 开关设定	(112)
7.4	扫描控制 7 段数码管	(115)
7.4.1	7 段数码管控制	(115)
7.4.2	扫描控制 7 段数码管	(115)
7.5	键盘扫描	(118)
7.6	键盘扫描及 7 段数码管控制	(125)
7.7	蜂鸣器控制	(129)
7.8	习题	(131)
第 8 章 中断控制		(132)
8.1	I/O 控制的方式	(132)
8.1.1	询问式	(132)
8.1.2	中断控制式	(132)
8.1.3	DMA 处理	(132)
8.2	8051 中断控制结构	(133)
8.3	相关控制寄存器	(134)
8.3.1	TCON: 计时控制寄存器	(134)
8.3.2	IE: 中断允许寄存器	(135)
8.3.3	IP: 中断优先权寄存器	(135)
8.4	8051 C 语言中断程序的写法	(135)
8.5	外部中断控制实验 1	(136)
8.6	外部中断控制实验 2	(138)
8.7	习题	(141)
第 9 章 8051 计时计数器		(142)
9.1	计时计数器相关控制寄存器	(142)
9.2	计数器模式 0 的工作	(143)
9.2.1	计时工作脉冲	(144)
9.2.2	启动计数器	(144)
9.2.3	计时时间长短设定	(145)
9.2.4	计时溢出如何处理	(145)
9.3	计数器模式 1 的工作	(147)
9.4	计数器模式 2 的工作	(149)
9.5	计数器模式 3 的工作	(152)

9.6 驱动 7 段数码管	(153)
9.6.1 计数器 0 及计数器 1 同时存在	(156)
9.7 驱动 7 段数码管及按键扫描	(159)
9.8 计时时钟的制作	(164)
9.9 手动计数器实验	(169)
9.10 简易频率计实验	(173)
9.11 习题	(178)
第 10 章 串行接口控制	(179)
10.1 串行数据传送原理	(179)
10.1.1 并行通信	(179)
10.1.2 串行通信	(180)
10.1.3 非同步串行数据传输	(180)
10.1.4 传输速率——波特率	(181)
10.2 8051 串行传输接口	(181)
10.2.1 串行传输模式 0	(182)
10.2.2 串行传输模式 1	(182)
10.2.3 串行传输模式 2	(182)
10.2.4 串行传输模式 3	(183)
10.3 串行传送控制寄存器	(183)
10.4 串行传输波特率的设定	(184)
10.5 PC 上的 RS232 通信程序	(185)
10.5.1 工作命令 cmd	(185)
10.5.2 通信协议参数 byte	(185)
10.5.3 通信端口 port 指定	(186)
10.5.4 通信端口状态	(187)
10.5.5 MODEM(调制解调器)状态	(187)
10.6 串行传送驱动程序	(193)
10.6.1 初始化串行通信端口	(193)
10.6.2 传送数据	(194)
10.6.3 接收数据	(197)
10.7 使用 MICRO C51 函数	(200)
10.7.1 由串行端口输出数据	(200)
10.8 输入一字符串	(205)
10.9 输入一数字	(207)
10.10 建立交互式的 8051 系统开发环境	(209)
10.11 习题	(212)
第 11 章 LCD 接口控制	(213)
11.1 LCD 内部结构介绍	(213)
11.1.1 CG ROM	(213)
11.1.2 DD RAM	(213)

11.1.3	CG RAM	(213)
11.1.4	控制方式	(214)
11.1.5	LCD 控制指令	(214)
11.2	LCD 驱动子程序	(216)
11.2.1	写命令到 LCD	(216)
11.2.2	写数据至 LCD	(217)
10.2.3	初始化 LCD	(217)
11.3	LCD 显示器测试	(218)
11.4	自定义 LCD 字型	(221)
11.5	习题	(225)
第 12 章	单片机 8051 声效设计	(226)
12.1	可编程声效发生器内部寄存器分析	(226)
12.1.1	音调控制产生寄存器 R0~R5	(226)
12.1.2	噪声产生寄存器 R6	(227)
12.1.3	音调/噪声混合及输入/输出应用控制寄存器 R7	(227)
12.1.4	振幅控制寄存器 R8,R9,R10	(227)
12.1.5	包络发生器控制寄存器 R11,R12,R13	(227)
12.1.6	输入输出端口寄存器 R14,R15	(229)
12.2	声效控制原理	(229)
12.2.1	单纯音调效果	(230)
12.2.2	噪声配合包络控制效果	(230)
12.2.3	频率扫描效果	(230)
12.3	可编程声效发生器声音频率计算	(230)
12.4	产生救护车警报声	(235)
12.5	产生机关枪声响	(237)
12.6	产生爆炸声响	(240)
12.7	产生激光枪声响	(242)
12.8	产生炸弹呼啸声效	(244)
12.9	测试各个单音音阶	(246)
12.10	演奏一段旋律	(248)
12.11	习题	(251)
第 13 章	数字模拟转换器接口	(252)
13.1	DAC 接口设计	(252)
13.2	测量 DAC 输出电压值	(253)
13.3	由 DAC 接口发出声音	(254)
13.6	习题	(256)
第 14 章	利用 8051 输出语音	(257)
14.1	声音录音放音基本原理	(257)
14.2	产生及编辑语音波形文件	(258)

14.3	转换语音数据文件	(260)
14.4	让 8051 电路板播放语音	(263)
14.5	习 题	(267)
第 15 章 8051 控制 PC I/O 接口卡		(268)
15.1	8051 模拟 PC I/O 插槽信号	(268)
15.2	PC/8051 语音控制实验卡介绍	(269)
15.3	语音卡电路设计	(270)
15.4	8051 单板控制语音卡	(274)
15.5	PC/8051 多功能实验卡介绍	(283)
15.6	8051 单板控制多功能实验卡	(283)
第 16 章 8051 无线遥控接口		(301)
16.1	遥控模块特性说明	(302)
16.2	遥控模块系统组成	(301)
16.2.1	发射器	(302)
16.2.2	接收机	(302)
16.3	编解码 IC HT12 简介	(303)
16.3.1	HT 12 编解码器特性介绍	(304)
16.3.2	引脚说明	(304)
16.4	遥控模块电路说明	(305)
16.4.1	控制信号分析	(307)
16.4.1	引脚使用功能	(308)
16.5	8051 接收模块测试程序	(309)
16.6	8051 多功能控制板无线遥控接口	(314)
16.6.1	P51 接收无线电遥控的信号	(314)
16.6.2	P51 发射无线电的信号	(318)
第 17 章 8051 红外线遥控接口控制		(324)
17.1	红外线接口应用场合	(324)
17.2	红外线接口实验套件简介	(324)
17.2.1	红外线接口实验套件介绍	(325)
17.3	示范程序介绍	(326)
17.3.3	IR.C;PC 上红外线信号波形观察及学习程序	(326)
17.3.3	IC.C;PC 上 IR.SET 遥控器解码程序	(328)
17.3.3	I1.ASM;单片机 8051 IR.SET 遥控器解码程序	(329)
17.3.4	I2.ASM;单片机 8051 IR.SET 遥控器应用示范程序	(330)
17.3.5	KIR.C;PC 控制 CD 放音机控制程序	(331)
17.3.6	VIR.C;声控 CD 放音机控制程序	(333)
第 18 章 8051 声控电脑设计		(335)
18.1	声控电脑原理	(335)

18.2 系统特性及组成	(336)
18.2.1 DSP 语音识别声控系统特性	(336)
18.2.2 DSP 语音识别声控系统组成	(337)
18.3 DSP 控制板简介	(338)
18.3.1 DSP 控制板组成	(338)
18.3.2 DSP 控制板 I/O 接点说明	(339)
18.3.3 跳线设定	(339)
18.4 语音识别 DSP 控制命令	(340)
18.5 声控系统展示操作	(341)
18.6 声控系统展示控制程序	(342)
18.7 声控电脑应用	(343)
附录 A ROM 模拟器使用	(345)
附录 B 8051 多功能控制板零件表	(348)
附录 C AT89C1051/AT89C2051 特性介绍	(351)
附录 D 89CXX 烧录模拟器 EPM89 特性	(354)
附录 E 89CXX 烧录模拟器 EPM89 使用说明	(355)

第1章 单片机 8051 简介

单片机是把 CPU、内存及 I/O 压缩在同一块芯片上，再外加一些电子元件便可以构成一套简易的控制系统。如此一来可以降低硬件成本，由于单片机芯片设计及制造技术的限制，在面积有限的芯片上无法设计出太大的内存空间，因此单片机上的 ROM 及 RAM 的容量都比较小，不过却也加入了位输入输出控制，计时计数器及外部中断的控制功能，有些单片机还有串行传输的接口，甚至还提供有 A/D(模拟至数字转换)及 D/A(数字至模拟转换)的接口，真可谓麻雀虽小五脏俱全。

8051 是 INTEL 公司开发出来的一块相当成功的单片机，现在已普遍地应用在工业界中，由于其使用的普及，因此目前有好几家设计半导体芯片的公司也制造与 8051 兼容的单片机，有些公司所制造出来的单片机其执行速度更快，可以高达 40 MHz，读者若想加快单片机系统的执行速度时可以选用此一类型的 8051。

由于 8051 在业界的大量使用，未来的市场还是看好，无怪乎还有众多的厂商纷纷推出兼容的单片机及支持 8051 的程序开发工具，可以预见未来这几年单片机市场上 8051 还会占有一席之地。

1.1 8051 特性

8051 单片机是 INTEL 公司在 8048 的基础上，对其功能加以改进所开发出来的 8 位单片机，表 1-1 是 8048 与 8051 硬件功能的比较表。

表 1-1 8048 与 8051 的功能比较

比较项目	8048	8051
指令周期	2.5 μ s	1 μ s
内部 RAM	64 字节	128 字节
内部 ROM	1K 字节	4K 字节
外部 RAM	256 字节	64K 字节
外部 ROM	4K 字节	64K 字节
I/O 引脚数	27	32
中断源	2	5
定时器	8 位 1 组	16 位 2 组
串行端口	无	1 组

从表中可以看出 8051 在功能上比 8048 强很多，程序代码（存于外部 ROM 中）的设计空间如同传统 8 位的单片机，像 Z80、6502 CPU 等，寻址至 64K 字节的范围，更甚者，

其随机存储器（存于外部 RAM 中）可额外再寻址 64K 字节，这是 8051 特别优异的一点，加上 I/O 控制端口、中断功能、定时器及串行接口，使得在一块 8051 芯片上外加少许外接元件便可组成一个完整的单片机控制系统。图 1-1 是 8051 单片机内部的组成结构图。

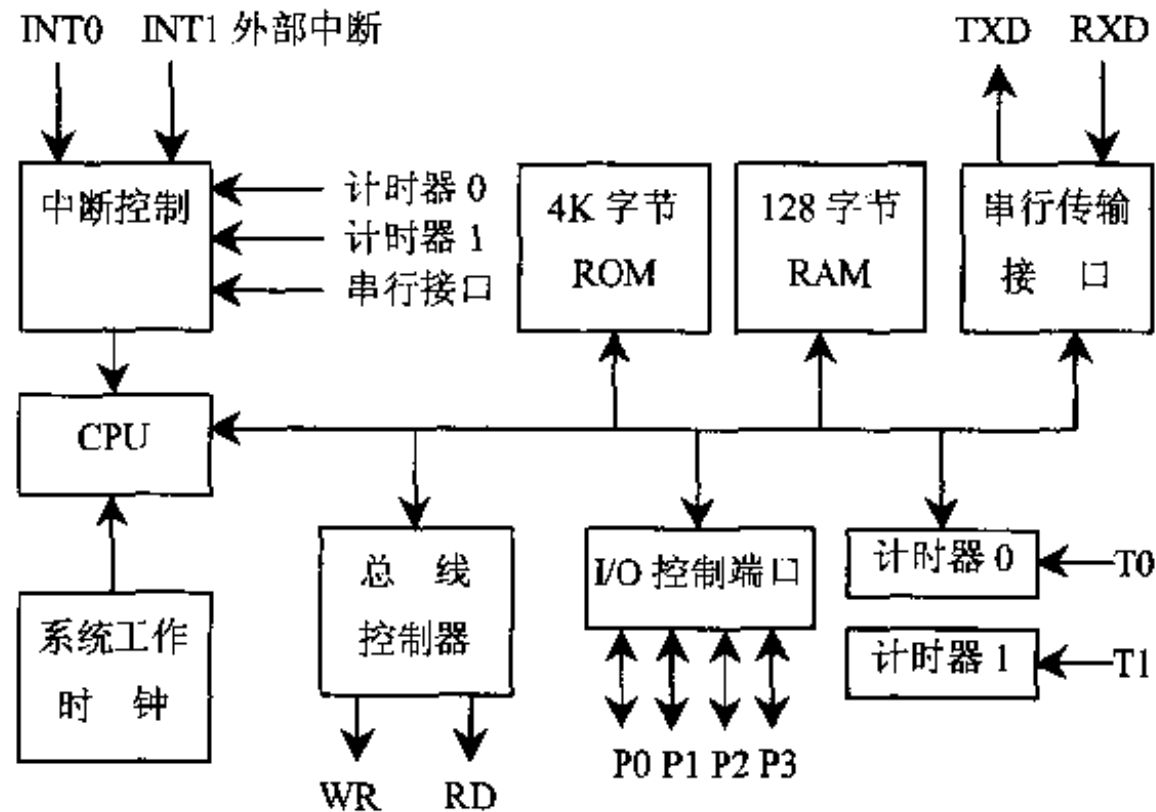


图 1-1 8051 的内部组成结构

1.1.1 8051 系列成员

表 1-2 列出了 8051 系列的成员：

表 1-2 8051 系列内部 ROM 和内部 RAM 的内存容量

编 号	电路类型	ROM 存储容量(字节)	RAM 存储容量(字节)
8051AH	HMOS	4K ROM	128
8031AH	HMOS	没有	128
8751H	HMOS	4K EPROM	128
80C51	CMOS	4K ROM	128
80C31	CMOS	没有	128
8052	HMOS	8K ROM	256
8032	HMOS	没有	256

其中 8751H 有可擦除可编程只读存储器（EPROM），可以存放程序代码，同时具有程序保密的特性，可以防止程序代码被任意地拷贝，只是价格较贵。电路构成类型如为 CMOS 则耗电较低，而 8031 与 8051 的差别在于 8031 内部本身没有可存放程序代码的存储空间（没有内部 ROM 的型态），因此程序代码必须由外部提供并外加 EPROM。

8051 主要功能列举如下：

- 为一般控制应用的 8 位单片机;
- 芯片内部有时钟振荡器 (传统最高工作频率可达 12MHz);
- 内部只读存储器 (ROM) 为 4K 字节;
- 内部随机存储器 (RAM) 为 128 字节;
- 外部只读存储器可扩充至 64K 字节;
- 外部随机存储器可扩充至 64K 字节;
- 32 条双向输入输出线, 且每条均可以单独做 I/O 的控制;
- 5 个中断向量源;
- 2 组独立的 16 位定时器;
- 1 个全双工串行通信端口;
- 8751 及 8752 单片机具有数据保密的功能;
- 单片机提供位逻辑运算指令。

1.2 其他 8051 兼容芯片简介

8051 可说是目前最热门、使用历史最久的单片机了, 除了 INTEL 公司开发的传统芯片外, 随着这几年半导体集成电路设计技术及制造技术的提高, 几家设计半导体芯片的公司陆续推出兼容的单片机, 这些单片机引脚设计及电气特性与 8051 一样, 还有如下优点:

- 更高的操作频率;
- 可电气烧录程序及擦除的功能;
- 内置看门狗计时器防止程序执行死机;
- 增加另一组串行端口。

以下将列举各种最新改进的 8051 兼容芯片。主要特点简介如下:

1.2.1 ATMEL 89C51 系列单片机

1. 89C51 工作频率可达 20 MHz。
2. 89C51 具有 4K 字节可电气烧录及擦除的程序空间, 可以快速擦除程序并烧录新的程序, 方便实验。
3. 89C52 具有 8K 字节可电气烧录及擦除的程序 ROM。
4. 89C55 具有 20K 字节可电气烧录及擦除的程序 ROM。
5. 89C1051(1K)、89C2051(2K) 为 20 引脚包装, 没有 I/O 端口 P_0 及 P_2 , 适合做更小型的电路设计。

1.2.2 DALLAS DS80C320 单片机

1. 工作频率可达 25 MHz。

2. 内置看门狗计时器。
3. 内置两组全双工的串行传输端口。
4. 两组 DPTR 方便程序设计。
5. 一个机器周期只需 4 个时钟工作周期，传统的 8051 为 12 个时钟工作周期。

1.2.3 WINBOND W78C31 单片机

工作频率可高达 40MHz。

读者如果对这些芯片感兴趣的话，可以翻阅电子杂志，查查有关国内代理商，如果您想改用 16 位 CPU，来做更复杂的控制系统设计的话，不妨考虑一下，使用这些芯片，至少有以下的一些优点：

- 不用重新学习汇编语言程序设计。
- 不必再花笔钱买开发工具。
- 原来的硬件只需稍微做修改便可工作。
- 原来的软件不必更换，执行速度更快。
- 既省力又省时间，可设计出功能更好的产品。

所以想从事单片机自动控制的设计行业，学习 8051 单片机绝对不会错的，厂商还会推出功能更强的兼容单片机，及支持 8051 的程序开发工具，现在是这样，以后仍是。

1.3 8051 引脚说明

图 1-2 是 8051 的引脚图，说明如下：

• VCC

8051 电源正极输入，接+5V 电压。

• VSS

电源接地端。

• XTAL1

单片机系统时钟的反相放大器输入端。

• XTAL2

系统时钟的反相放大器输出端，一般在设计上只要在 XTAL1 和 XTAL2 之间接上一只石英晶振，系统时间就可以工作了，此外可以在两引脚与接地引脚之间加入一 20PF 的小电容，可以使系统更稳定，避免杂波干扰而死机。

• RESET

8051 的重置引脚，高电位工作，当要对芯片重置时，只要将此引脚电位提升到高电位，并持续两个机器周期以上的时间，8051 便能完成系统重置的各项工作，使得内部特

特殊功能寄存器的内容均被设成已知状态，并且从地址 0000H 处开始读入程序代码而执行程序。

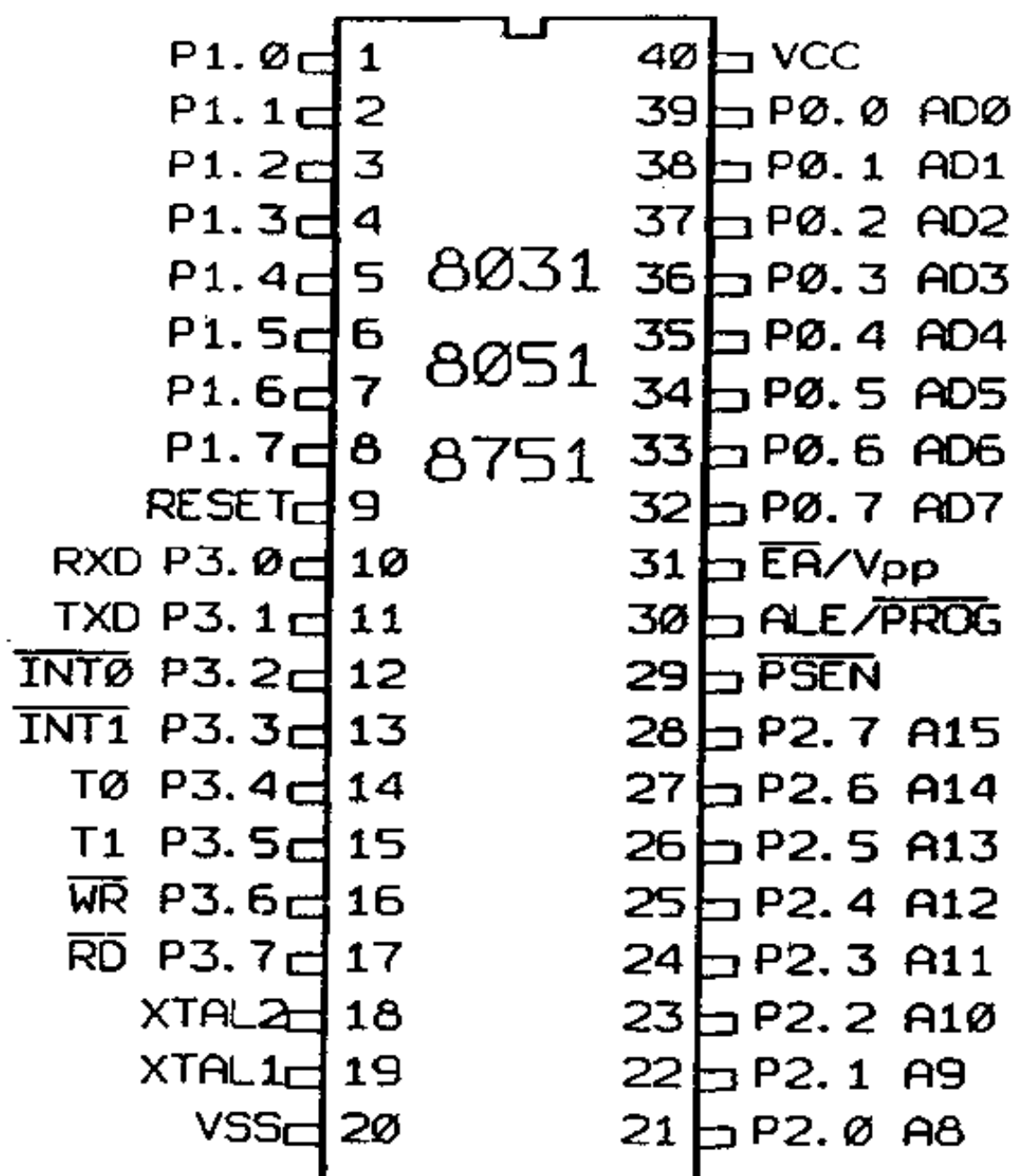


图 1-2 8051 引脚图

• EA/V_{pp}

EA 为英文“External Access”的缩写，表示存取外部程序代码的意思，低电位工作，也就是说当此引脚接低电位后，系统会读取外部的程序代码（存于外部 EPROM 中）来执行程序。因此在 8031 及 8032 中，EA 引脚必须接低电位，因为其内部无只读存储器空间。如果是使用 8751 内部程序空间时，此引脚要接成高电位。此外，在将程序代码烧录至 8751 内部 EPROM 时，可以利用此引脚来输入 21V 的烧录高压（V_{pp}）。

• ALE/PROG

ALE 是英文“Address Latch Enable”的缩写，表示地址锁存允许信号。8051 可以利用这个引脚来触发外部的 8 位锁存（如 74LS373），将端口 0 的地址总线（A0~A7）锁存进入锁存器中，因为 8051 是以多工的方式送出地址及数据。平时在程序执行时 ALE 引脚的输出频率约是系统工作频率的 1/6，因此可以用来驱动其他外围芯片的时钟输入。此外在烧录 8751 程序代码时，此引脚会被当成程序规划的特殊功能来使用。

• PSEN

此为“Program Store Enable”的缩写，其意为程序储存允许，当 8051 被设成为读取外部程序代码工作模式时（EA=0），会送出此信号以便取得程序代码，通常这个引脚是接到 EPROM 的 OE 引脚。8051 可以利用 PSEN 及 RD 引脚分别启用存在外部的 RAM 与 EPROM，使得随机存储器与只读存储器可以合并在一起而共用 64K 的寻址范围。

• PORT0 (P0.0~P0.7)

端口 0 是一个 8 位宽的漏极开路（Open Drain）双向输入输出端口，共有 8 位，P0.0 表示位 0，P0.1 表示位 1，依此类推。其他三个 I/O 端口（P1、P2、P3）则不具有此电路结构，而是内部有一提升电路，P0 在当做 I/O 用时可以推动 8 个 LS 的 TTL 负载。如果当 EA 引脚为低电位时（即取用外部程序代码或随机存储器），P0 就以多工方式提供地址总线（A0~A7）及数据总线（D0~D7）。设计者必须外加一锁存器将端口 0 送出的地址锁存为 A0~A7，再配合端口 2 所送出的 A8~A15 合成一完整的 16 位地址总线，而寻址到 64K 的外部内存空间。

• PORT2 (P2.0~P2.7)

端口 2 是具有内部提升电路的双向 I/O 端口，每一个引脚可以推动 4 个 LS 的 TTL 负载，若将端口 2 的输出设为高电位时，此端口便能当成输入端口来使用。P2 除了当作一般 I/O 端口使用外，若是在 8051 扩充外接只读存储器或随机存储器时，也提供地址总线的高字节 A8~A15，这个时候 P2 便不能作为 I/O 来使用了。

• PORT1 (P1.0~P1.7)

端口 1 也是具有内部提升电路的双向 I/O 端口，其输出缓冲器可以推动 4 个 LS TTL 负载。同样地，若将端口 1 的输出设为高电位，便是由此端口来输入数据。如果是使用 8052 或是 8032 的话，P1.0 又作为定时器 2 的外部脉冲输入引脚，而 P1.1 可以有 T2EX 功能，可以做外部中断输入的触发引脚。

• PORT3 (P3.0~P3.7)

端口 3 也具有内部提升电路的双向 I/O 端口，其输出缓冲器可以推动 4 个 TTL 负载，同时许多工具有其他的额外特殊功能，包括串行通信、外部中断控制、计时计数控制及外部随机存储器内容的读取或写入控制等功能。其引脚分配如下：

P3.0: RXD, 串行通信输入。

P3.1: TXD, 串行通信输出。

P3.2: INT0, 外部中断 0 输入。

P3.3: INT1, 外部中断 1 输入。

P3.4: T0, 计时计数器 0 输入。

P3.5: T1, 计时计数器 1 输入。

P3.6: WR, 外部随机存储器的写入信号。

P3.7: RD, 外部随机存储器的读取信号。

1.4 系统重置

系统重置是任何微处理机系统执行的第一步, 使整块控制芯片回到预先设定的硬件状态下。8051 单片机的系统重置是由 RESET 引脚控制的, 当此引脚送入高电位超过 24 个振荡周期时, 8051 即进入芯片内部重置的状态下, 而且一直在此状态下等待, 直到 RESET 引脚转为低电位后, 才检查 EA 引脚是高电位或低电位, 若为高电位则执行芯片内部的程序代码, 若为低电位便会执行外部的程序。

而 8051 在系统重置时, 将其内部的一些重要寄存器设定为某些特定的值, 我们在下一节介绍特殊功能寄存器时再做说明。至于内部 RAM 内的数据则不变。

对 8051 而言, 在芯片内部的 RESET 引脚接有一史密特触发电路, 以及一个电阻接在地端, 如图 1-3 所示, 我们只要在外部接上一个 10 μ F 的电容器便可成为 RESET 的重置电路。如果要使程序代码重新执行时, 可以按下外接的按钮开关, 将 RESET 引脚接至+5V 电源上, 使系统重置而重新执行程序。

此外在 RESET 接点上也可以连至外部 ROM 模拟器上的 RESET_HI 引脚, 使其成为遥控的系统重置功能。

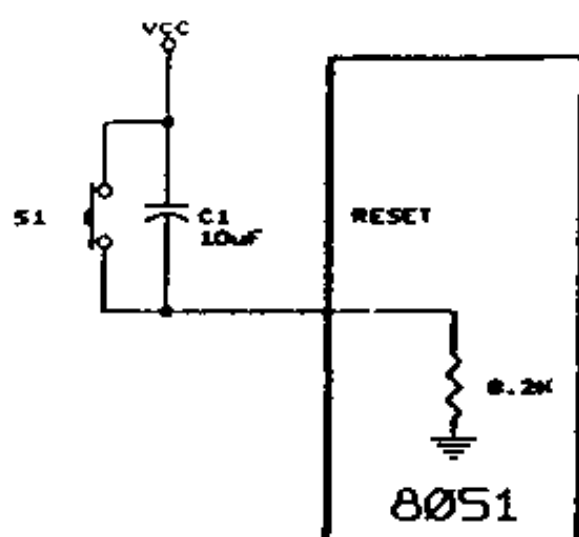


图 1-3 8051 RESET 电路

1.5 内存空间

8051 的内存分配共有以下 4 部分, 如图 1-4 所示。

1. 内部只读存储器 (ROM) 4K 字节;
2. 外部只读存储器 (ROM) 64K 字节 (含内部 4K 字节);
3. 内部随机存储器 (RAM) 256 字节;

4. 外部随机存储器 (RAM) 64K 字节。

8051 的 ROM 与 RAM 的地址空间是各自独立的, 可以说有 128K 的内存空间 (传统的 8 位 CPU 像 6502、Z80 只有 64K 字节的内存), 因此写起程序来更具弹性, 程序代码及变量可以分别利用 64K 字节, 此点使 8051 特别适用使用高级 C 语言来做控制程序的设计。

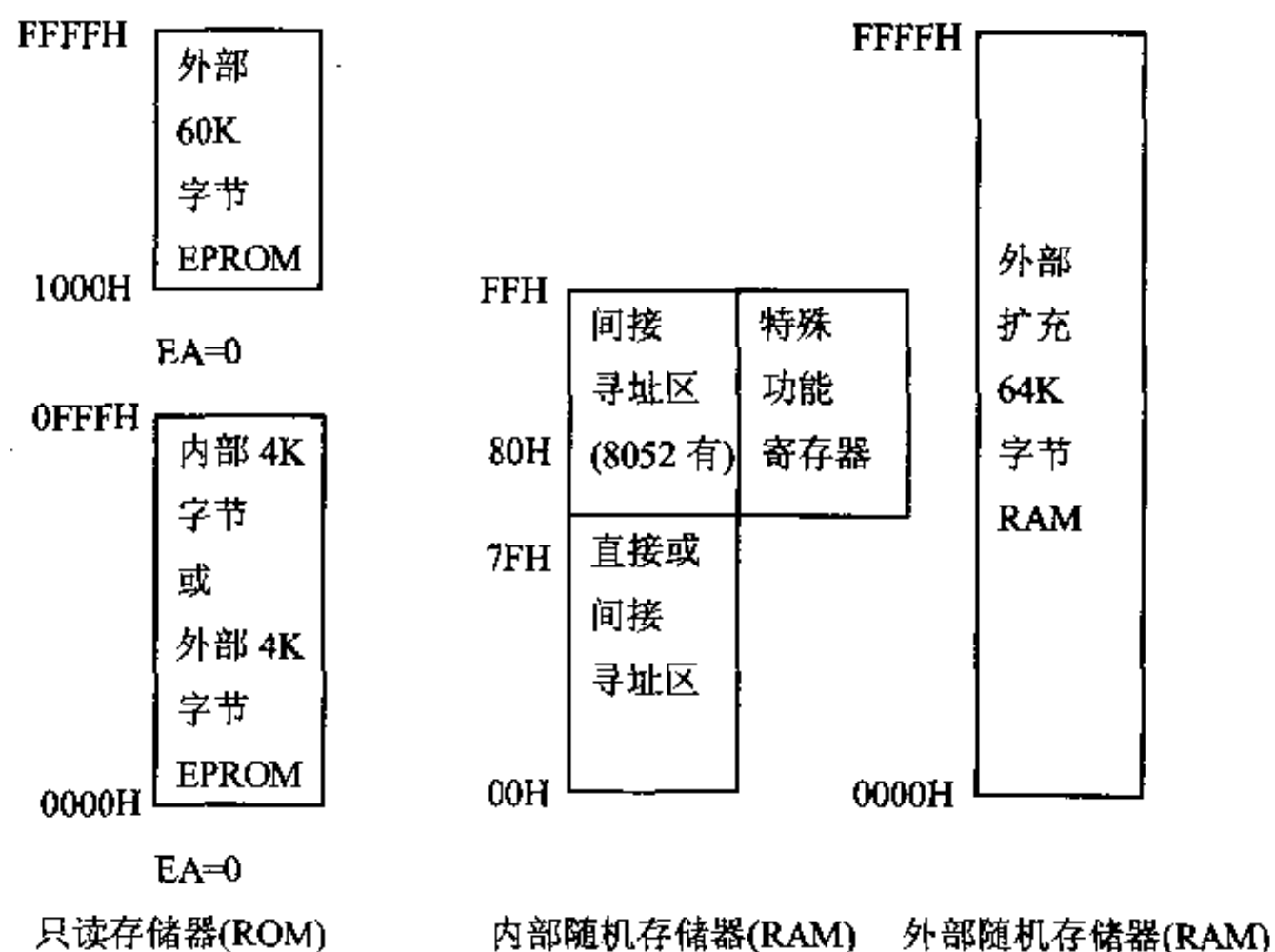


图 1-4 8051 内存分配图

1.5.1 只读存储器

只读存储器用来存放 8051 控制程序, 可以使用内部的 4K 字节 (EA 引脚为高电位), 或是外部的 64K 字节 (EA 引脚接地)。当系统 RESET 时, CPU 自动从地址 0 提取程序代码开始执行程序, 此区域的程序代码或数据只能被 CPU 读取而无法写入, 所以称为程序 ROM 区 (ROM 是只读存储器, 只能读取内部数据而无法写入数据)。

图 1-5 所示为 8051 外接 64K 字节的只读存储器线路图, 使用 EPROM 27512, 容量大小为 64K 字节, 由 8051 提供的系统总线如下:

1. 地址总线 (A0~A15)

A0~A7 由 P0.0~P0.7 提供, 经锁存器锁存。

A8~A15 由 P2.0~P2.7 提供。

2. 数据总线 (D0~D7)

由 P0.0~P0.7 供给。

3. 控制信号

- ALE:

地址锁存允许信号，接至锁存器，将端口 0 送出的信号锁存住，成为地址总线信号 (A0~A7)。

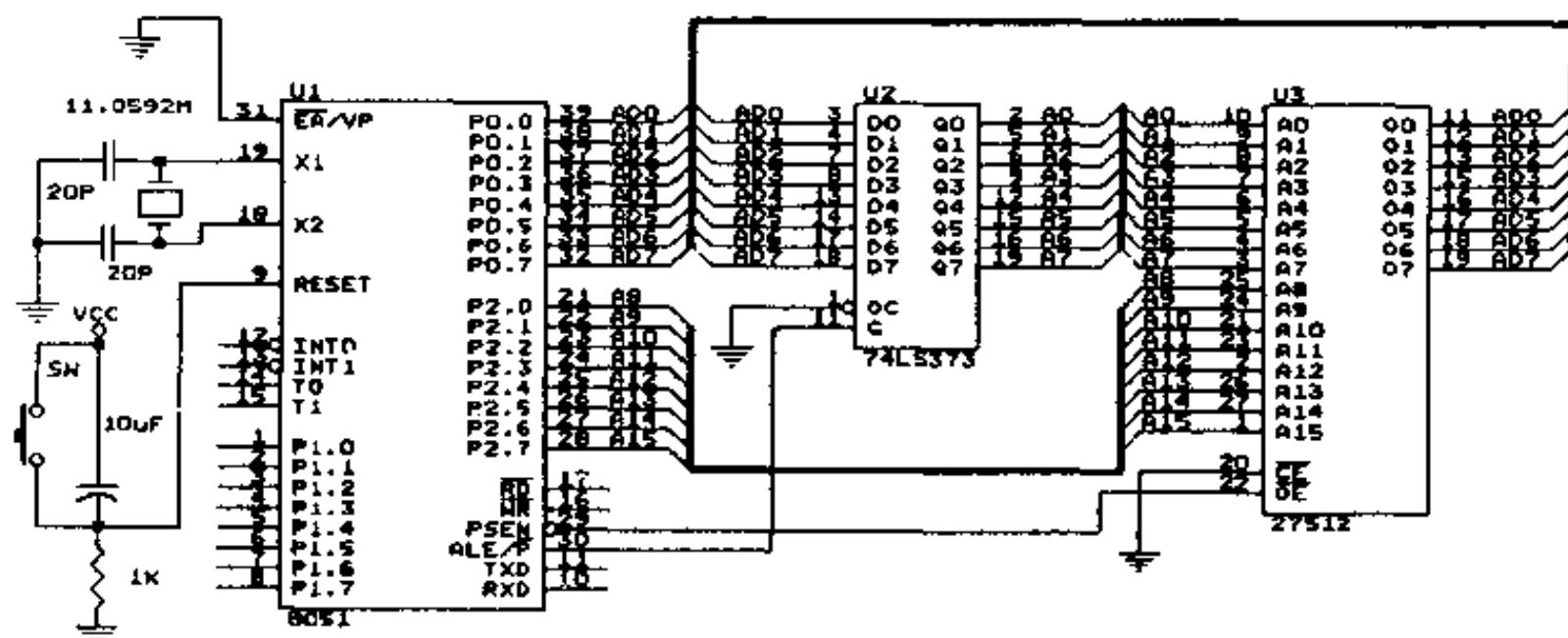


图 1-5 8051 外接 64K 只读存储器

• PSEN:

程序储存允许信号，接至程序 ROM 的 OE 引脚，用来读取外部程序代码。

此外与 8051 程序设计相关的几个只读存储器特殊地址，称为中断服务程序的进入点位置，如表 1-3 所示：

表 1-3 8051 程序设计只读存储器特殊地址

名称	中断源	向量位置
系统重置	RESET	00H
外部中断 0	INT0	03H
定时器 0 中断	TIMER0	0BH
外部中断 1	INT1	13H
定时器 1 中断	TIMER1	1BH
串行端口	UART	23H

当系统执行重置后，会从地址 0 起开始执行程序代码。如果产生定时器 0 计时中断后，则会到地址 0BH 执行计时中断服务程序，而在该地址可能是一个“LJMP”指令跳到真正的服务程序进入点位置去执行相关的中断工作。

1.5.2 随机存储器

8051 内含有 256 字节的随机存储器，其中分为两部分：

1. 存放数据的 RAM 内存地址 00H~7FH。

此区域是使用者可以真正使用的 RAM 空间。

2. 特殊功能寄存器 (Special Function Register, SFR) 地址 80H~FFH。

这些位置定义了 8051 内部一些特殊的寄存器, 若要充分利用 8051 的功能, 必须对这些寄存器加以了解。

图 1-6 所示为此两部分的内存分布图。对于 8051 存放数据的 RAM 只有 128 字节 (00H~7FH), 而 8052 则有另外的 128 个字节 (80H~FFH) 可供使用。至于特殊功能寄存器则占用内存地址为 80H~FFH。

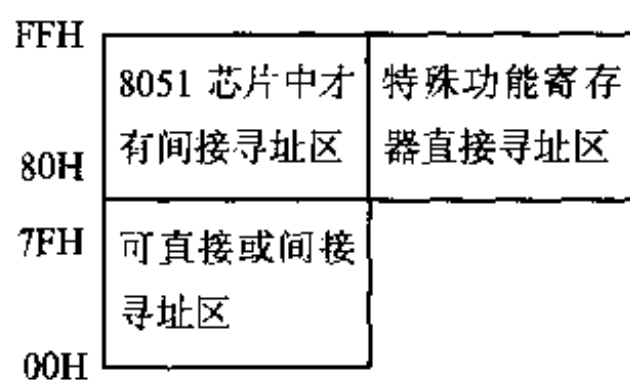


图 1-6 8051 内部数据内存分配

1.5.3 地址 00H~7FH

此 128 字节可以使用直接或间接寻址的方式来存取其内部的数据, 我们以图 1-7 来说明, 此地址可以分为以下 3 部分:

1. 寄存器库 00H~1FH;
2. 可位寻址区 20H~2FH;
3. 一般用途空间 30H~7FH。

一般用途 RAM	
30H	7FH
可位寻址 RAM	
20H	2FH
寄存器库 3	
18H	1FH
寄存器库 2	
10H	17H
寄存器库 1	
08H	0FH
寄存器库 0	
00H	07H

图 1-7 8051 内部 RAM 地址 00H~7FH

1.寄存器库 00H~1FH:

地址 00H~07H 称为寄存器库 0, 08H 至 0FH 为寄存器库 1, 依此类推, 共有 4 组寄存器库, 是我们在写程序时可以使用的工作寄存器, 每一组有 8 个字节, 称为 R0~R7。

例如: 汇编语言指令。

```
MOV R0,A
```

```
MOV R1,A
```

R0、R1 就是指目前工作寄存器的 R0 与 R1。但 8051 一共有 4 组寄存器组, 到底目前的工作寄存器库是那一组呢? 是由特殊功能寄存器 PSW 的 RS1 及 RS0 位来定义, PSW 在以下特殊功能寄存器中有说明。

2.可位寻址区 20H~2FH:

这 16 个字节是可以分别做位寻址的空间, 因此可以有 128 (16×8) 位可供使用。例如:

```
setb 20H.7
```

```
clr 21H.0
```

即是将地址 20H 的位 7 设为 1, 而将地址 21H 的位 0 清 0。

3.一般用途空间 30H~7FH:

此地址在 8051 中并未加以定义, 由使用者自由使用, 可以用来存放程序变量, 然而在程序执行时, 得设定堆栈区 (执行 CALL、PUSH、POP 指令会用到), 堆栈区的大小由使用者自行设定, 栈指针 SP 是往内存高处增加, 如果所设定的堆栈大小是 32 字节, 则可以将 SP 指向 5FH, 以下列指令来设定。

```
MOV SP,#5FH
```

1.5.4 特殊功能寄存器

8051 的特殊功能寄存器占用内存 80H~F8H 的地址, 在这 128 字节的空间中并未完全使用, 表 1-4 是其内存分布图。实际上只使用了 21 个地址, 各有不同的特殊用途。

表 1-4 8051 特殊功能寄存器分布图

名称	用途	位址
*ACC	累加器	0E0H
*B	寄存器 B	0F0H
*PSW	程序状态字	0D0H
SP	堆栈指针	81H

(续表)

名称	用途	位址
DPTR	数据指针 (DPH、DPL)	83H、82H
*P0	I/O 端口 0	80H
*P1	I/O 端口 1	90H
*P2	I/O 端口 2	0A0H
*P3	I/O 端口 3	0B0H
*IP	中断优先顺序控制	0B8H
*IE	中断使能控制	0A8H
TMOD	定时器模式控制	89H
*TCON	定时器控制	88H
TH0	定时器 0 高字节寄存器	8CH
TL0	定时器 0 低字节寄存器	8AH
TH1	定时器 1 高字节寄存器	8DH
TL1	定时器 1 低字节寄存器	8BH
*SCON	串行端口控制	98H
SBUF	串行端口数据缓冲器	99H
PCON	电源控制	87H

*: 表示可位定址

(1) 累加器 A (Accumulator)

写程序时大部分的指令运算都通过此寄存器, 包括数据转移、储存运算结果和条件跳越判断。

(2) B 寄存器

此寄存器主要用在乘法及除法指令中, 在乘法运算中存放乘积结果的高字节数据; 在除法运算中则存放余数。当然也可以做一般寄存器来使用。

(3) 程序状态字 PSW (Program Status Word)

此寄存器用于存放 CPU 的状态, 类似一般 CPU 的标志寄存器, 使用者可以改变其值来控制 CPU 的执行。图 1-8 是 PSW 的内容。

B7	B6	B5	B4	B3	B2	B1	B0
CY	AC	F0	RS1	RS0	OV	—	P

CY (PSW.7) : 进位标志。
 AC (PSW.6) : 辅助进位标志。
 F0 (PSW.5) : 一般用途标志。
 RS1 (PSW.4) : 寄存器库选择位 1。
 RS0 (PSW.3) : 寄存器库选择位 0。
 OV (PSW.2) : 溢出标志。
 — (PSW.1) : 保留。
 P (PSW.0) : 同位标志, 在每个指令周期中, 若累加器内“1”的位数是奇数个则 P=1, 偶数个则 P=0。

RS0、RS1 选择:

RS1	RS0	寄存器库	内存位址
0	0	0	00H~07H
0	1	1	08H~0FH
1	0	2	0H~17H
1	1	3	8H~1FH

图 1-8 PSW 寄存器内容

(4) 栈指针 SP

为 8 位的寄存器, 用以指示目前堆栈区的存放位置, 堆栈区最多只有 256, 而且一定在内部 RAM 中, 当 8051 系统重置后 SP 指向 07H, 因此程序一执行时通常会将堆栈往后移, 避免程序执行时把堆栈破坏掉。

(5) 数据指针 DPTR (Data Pointer)

DPTR 是一个 16 位寄存器, 由 DPH 及 DPL 两个寄存器组成, 系统 DPTR 可以看成是 16 位寄存器寻址到完整的 64K 内存空间或是看成两个 8 位寄存器来加以利用。一旦作为 16 位寄存器便可利用指令 MOVX A, @DPTR 来存取外部随机存储器, 或利用指令 MOVC A, @DPTR 来存取外部只读存储器。

(6) P0、P1、P2、P3 (端口 0~端口 3)

为 8051 4 个 I/O 端口的输出锁存寄存器。

(7) TH0、TL0、TH1、TL1 计时/计数寄存器

分别为计时器 0 及计时器 1 的工作寄存器, 这二对寄存器可以做 16 位的计时/计数用。

(8) 串行端口缓冲器 SBUF (Serial Buffer)

用来存放串行传输时数据进出的工作寄存器，通过串行端口将数据传送出去是将数据写入 SBUF，而在接收时则从 SBUF 内读取对方传送来的数据。

(9) 控制寄存器

IP、IE 寄存器是做 8051 的中断控制用；TMOD、TCON 寄存器用来做计时/计数器控制；SCON 则控制串行传输的工作模式设定。PCON 则做 8051 省电模式操作控制。我们将在 1.6 节中再对这些控制寄存器做进一步的说明。

1.5.5 外部随机存储器

8051 外部随机存储器可以扩充至 64K 字节，除了做一般的数据存取外，还可以以内存对应的方式做 I/O 的设计，一般常见的 I/O 特殊功能控制芯片有 8255、8279，可编程声效发生器也可由 I/O 来加以控制，对 8051 而言均是存取外部数据，对外部而言可分为内存或 I/O 芯片，使用指令均是：

```
MOVX A, @DPTR
```

```
MOVX A, @Ri
```

来读取外部数据，而使用指令：

```
MOVX @DPTR, A
```

```
MOVX @Ri, A
```

将数据写到外部 SRAM 或 I/O 控制芯片。

图 1-9 所示为外接 32K 字节存取外部数据 SRAM 的线路图，其解码地址为 0000H~7FFFH。8051 读写外部随机存储器时是分别使用 RD 及 WR 信号，而 8051 读取外部程序内存时是使用 PSEN 信号，可以看出，基本上 8051 可以完全控制两个独立 64K 字节的内存空间，一为只读存储器，一为随机存储器。

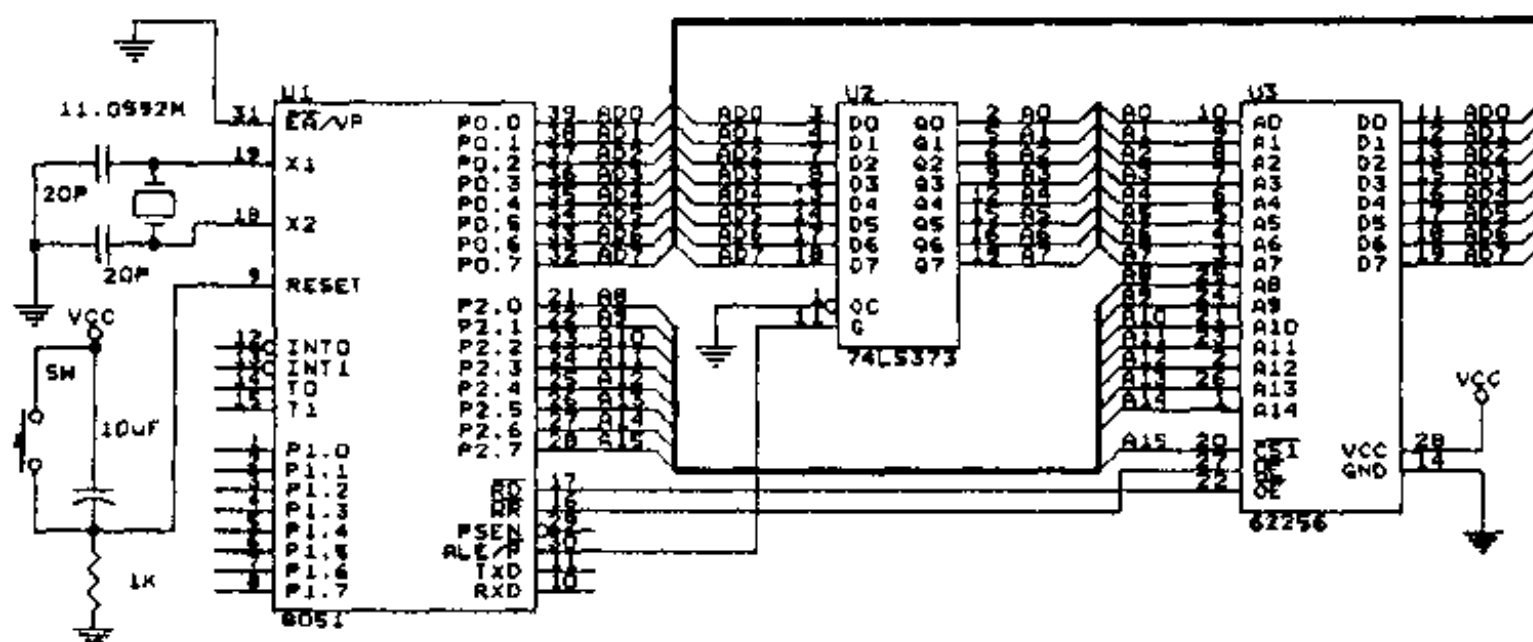


图 10-9 8051 外接 32K 字节数据 SRAM 线路图

程序通常是存在 EPROM 中，一旦烧录后即已固定，有时候，我们希望将程序代码移至 SRAM 中来执行时，则可将只读存储器与随机存储器合并在一起设计，图 1-10 便是这样的一个例子，将 RD 与 PSEN 信号经过一 AND 门合成 READ 信号来做 EPROM 或 SRAM 的读取控制，从线路图中可以看出 EPROM 27256 的寻址范围为 0000H~7FFFH，SRAM 62256 为 8000H~FFFFH。

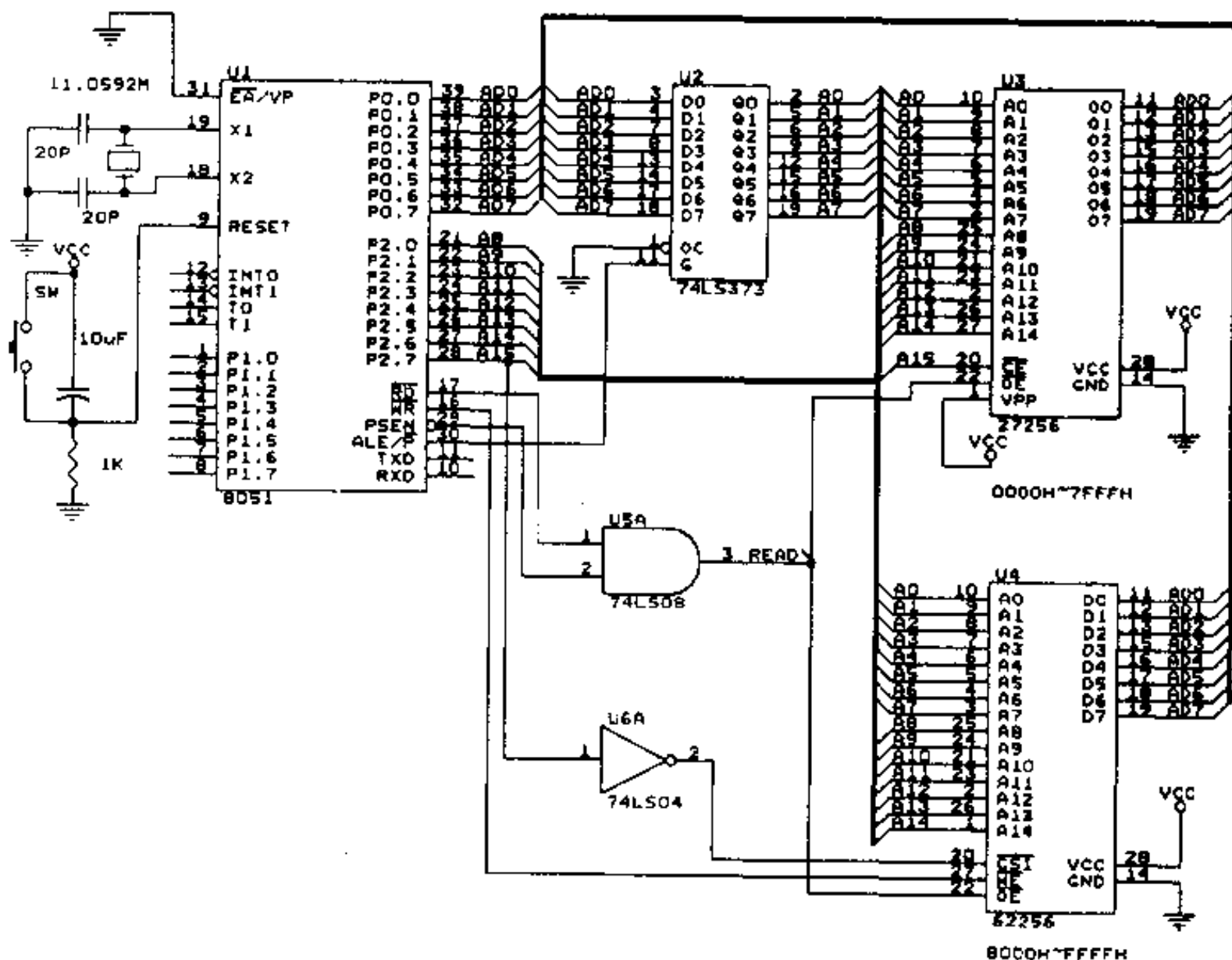


图 1-10 只读存储器与随机存储器合并在一起

当 8051 想写数据到外部内存时，是通过 WR 信号来控制 SRAM。至于读取数据时 (RD 线工作) 通过 READ 信号来控制 SRAM。当 8051 做程序代码读取 (PSEN 信号工作) 时，由 READ 线来读取 27256 内部的程序代码或是 62256 内部的程序代码 (由 8051 写入的数据)。例如：

LJMP 7000H 则是执行存在 EPROM 中的程序代码

LJMP 9000H 则是执行存在 SRAM 中的程序代码

1.6 8051 内部控制寄存器

8051 内部控制寄存器只有 6 个，想要充分发挥 8051 单片机的功能必须对这些寄存器有所了解。

1. IE、IP 寄存器：用于中断控制
2. TMOD、TCON 寄存器：计时/计数器用
3. SCON 寄存器：串行传输控制
4. PCON 寄存器：省电模式操作

以下分别加以介绍。

1.6.1 IE：中断允许寄存器

可位寻址，地址：A8H

B7	B6	B5	B4	B3	B2	B1	B0
EA	—	ET2	ES	ET1	EX1	ET0	EX0

- EA (IE.7) : EA=0 时，所有中断停用（禁止中断）。
EA=1 时，各中断的产生由个别的允许位决定。
- (IE.6) : 保留。
- ET2 (IE.5) : 允许定时器 2 溢出的中断（8052 使用）。
- ES (IE.4) : 允许串行端口的中断（ES=1 使能，ES=0 禁止）。
- ET1 (IE.3) : 允许定时器 1 中断。
- EX1 (IE.2) : 允许外部中断 INT1 的中断。
- ET0 (IE.1) : 允许定时器 0 中断。
- EX0 (IE.0) : 允许外部中断 INT0 的中断。

1.6.2 IP：中断优先次序寄存器

可位寻址，地址 B8H。

B7	B6	B5	B4	B3	B2	B1	B0
—	—	PT2	PS	PT1	PX1	PT0	PX0

- (IP.7) : 保留。
- (IP.6) : 保留。
- PT2 (IP.5) : 设定定时器 2 的优先次序（8052 使用）。
- PS (IP.4) : 设定串行端口的中断优先顺序。
- PT1 (IP.3) : 设定定时器 1 的优先顺序。
- PX1 (IP.2) : 设定外部中断 INT1 的优先顺序。
- PT0 (IP.1) : 设定定时器 0 的优先顺序。
- PX0 (IP.0) : 设定外部中断 INT0 的优先顺序。

1.6.3 TMOD：定时器模式控制寄存器

不可位寻址，地址 89H。

B7	B6	B5	B4	B3	B2	B1	B0
GATE	C/T	M1	M0	GATE	C/T	M1	M0
└─────────── 计时器 1 ───────────┘				└─────────── 计时器 0 ───────────┘			

GATE : 计时器工作门控位, 当 GATE=1 时, INT0 或 INT1 引脚为高电位, 同时 TCON 中的 TR0 或 TR1 控制位为 1 时, 计时/计数器 0 或 1 才会工作。若 GATE=0, 则只要将 TR0 或 TR1 控制位设为 1, 计时/计数器 0 或 1 即可工作。

C/T : 作计时器或计数器功能的选择位。C/T=1 为计数器, 由外部引脚 T0 或 T1 输入计数脉冲。C/T=0 时为计时器, 由内部系统时钟提供计时工作脉冲。

M1 : 模式选择位 1。

M0 : 模式选择位 0。

M1	M0	工作模式
0	0	13 位计时/计数器
0	1	16 位计时/计数器
1	0	8 位自动载入计时/计数器
1	1	计时器 1 停止工作, 计时器 0 分为两个独立的 8 位计时器 TH0 及 TL0

1.6.4 TCON: 计时控制寄存器

可位寻址, 地址 88H。

B7	B6	B5	B4	B3	B2	B1	B0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 (TCON.7) : 计时器 1 溢出标志, 当计时溢出时, 由硬件设定为 1, 在执行相对的中断服务程序后则自动清 0。

TR1 (TCON.6) : 计时器 1 启动控制位, 可以由软件来设定或清除。TR1=1 时启动计时器工作, TR1=0 时关闭。

TF0 (TCON.5) : 计时器 0 溢出标志, 当计时溢出时, 由硬件设定为 1, 在执行相对的中断服务程序后则自动清 0。

TR0 (TCON.4) : 计时器 0 启动控制位, 可以由软件来设定或清除。TR0=1 时, 启动计时器工作, TR0=0 时关闭。

IE1 (TCON.3) : 外部中断 1 工作标志, 当外部中断被检查出来时, 硬件自动设定此位, 在执行中断服务程序后, 则清 0。

IT1 (TCON.2) : 外部中断 1 工作形式选择, IT1=1 时, 由下降缘产生外部中断, IT1=0 时, 则为低电位产生中断。

- IE0 (TCON.1) : 外部中断 0 工作标志, 当外部中断被检查出来时, 硬件自动设定此位, 在执行中断服务程序后, 则清 0。
- IT0 (TCON.0) : 外部中断 0 工作形式选择, IT0=1 时为下降缘产生外部中断, IT0=0 时则为低电位产生中断。

1.6.5 SCON: 串行端口控制寄存器

可位寻址, 地址 98H。

B7	B6	B5	B4	B3	B2	B1	B0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

- SM0 (SCON.7) : 串行端口模式设定位 0。
- SM1 (SCON.6) : 串行端口模式设定位 1。
- SM2 (SCON.5) : 8051 连接多重处理器通信的控制位。
- REN (SCON.4) : 串行通信接收允许信号, 该位可以由软件来设定。
- TB8 (SCON.3) : 在串行通信模式 2 和模式 3 操作时第 9 个传送数据位。
- RB8 (SCON.2) : 在串行通信模式 2 和模式 3 操作时第 9 个接收数据位。
- TI (SCON.1) : 串行通信传送的中断处理标志。
- RI (SCON.0) : 负责串行通信接收的中断处理标志。

模式选择:

SM0	SM1	工作模式	说明	工作频率
0	0	0	移位寄存器	FOSC/12
0	1	1	8 位串行传送	可变
1	0	2	9 位串行传送	FOSC/64 或 FOSC/32
1	1	3	9 位串行传送	可变

1.6.6 PCON: 电源控制寄存器

不可位寻址, 地址 87H。

B7	B6	B5	B4	B3	B2	B1	B0
SMOD	—	—	—	GF1	GF0	PD	IDL

- SMOD : 双倍波特率控制位。
- : 保留 3 位。
- GF1 : 一般用途标志。
- GF0 : 一般用途标志。
- PD : 降低 8051 功率消耗控制位, PD=1 时设定, PD=0 为清除。
- IDL : 8051 芯片闲置状态操作控制位。

1.7 习 题

1. 何谓单片机控制系统?
2. 试画图说明单片机 8051 内部的组成结构。
3. 试说明下列 8051 单片机之间有何差别?
8051、8031、8052、80C51、8751
4. 说明下列 8051 的引脚功能。
EA、ALE、PSEN、PORT0 (P0.0~P0.7)、PORT2 (P2.0~P2.7)
5. 说明 8051 PORT3 (P3.0~P3.7) 除了做一般 I/O 控制外, 还有其他什么特殊功能?
6. 说明为何 8051 系统设计做 I/O 控制时经常保留 PORT0 及 PORT2?
7. 说明 8051 RESET 信号产生的 3 种情况。
8. 试说明如何执行 8051 内部及外部的只读存储器。
9. 试画图说明 8051 外接 64K 字节的只读存储器线路图, 使用 EPROM 27512。
10. 试说明 8051 中断服务程序进入点的 6 个位置。
11. 试说明 8051 内部 RAM 128 字节的使用可分为那 3 个区。
12. 试列举 8051 的 10 个特殊功能寄存器。
13. 试画图说明 8051 将只读存储器与随机存储器合并在一起的控制电路, EPROM 使用 27256, SRAM 使用 62256。

第 2 章 实验环境设定

在进行 8051 C 语言程序设计前，本章先对书中所需要的实验环境及软硬件工具作些说明，读者可以根据需要加以配置，包括以下几大项：

- 实验必备的硬件配置；
- 软件使用工具；
- 硬件接口卡。

2.1 实验必备的硬件配置

做硬件实验时手上若有一些必要的硬件工具，则实验做起来会比较顺利，所使用的硬件工具如下：

- 示波器；
- 逻辑笔；
- 数字式电表；
- PC 插槽接口保护器；
- 直流电源供应器；
- PC 个人电脑；
- 在线仿真器 ICE；
- ROM 模拟器；
- 烧录器；
- 紫外线清洗机；
- 面包板；
- 基本的焊接工具。

• 示波器

示波器是用来观测各种高低频模拟或数字信号波形的。基本上分为两种型号，一种为传统的模拟式示波器，另一种为储存式示波器（或称数字式储存示波器），功能后者更为强大，因为它除了有前者的功能外，还有适用的画面锁定的功能，尤其是在观察一些运行状态的波形时特别有用，在记录实验过程或信号除错上十分有利。手上有一台示波器可以方便我们观测编解码输入输出信号，此外对载波频率的调整也非常有用。

• 逻辑笔

逻辑笔是微处理机系统开发的必备工具，因为一套再复杂的数字系统也是由许多条单一的数字线所组成，而其信号不外乎为高电位、低电位、脉冲状态或高阻抗状态，一

且了解了此数字系统的特性后,便不难使用逻辑笔来测试线路进行故障排错和分析电路。

逻辑笔是最便宜的单片机电路检修及测试工具,实际上,普通的单片机和数字控制系统用逻辑笔检测就够了。

- 数字万用表

进行电路设计或专题制作时一定少不了一部万用表,只是现在大多采用数字万用表,在做单片机硬件制作上一样少不了,主要是做电压的测量和短路、断路的判断。现在很多数字万用表均配有短路声响报警的功能,只要测量到短路的情况则发出滴滴声来告知,此功能在做硬件电路板的线路检测上相当方便,在焊接完后便可以逐一检测线路的连接是否如电路图所示,在硬件的初步排错上帮助很大。

- PC 插槽接口保护器

进行 PC I/O 接口的实验时需要此装置,可以提高实验的效率。使用此装置有以下 3 点好处:

- 所设计的接口卡通过保护器而连到 PC 上,可以避免因线路设计不当而烧坏主板。
- 保护器将 PC 插槽上的接口信号延伸至主机外部,使用者可以很方便地测量接口卡上的各端点信号。
- 由于保护器置于 PC 外部,在拔插接口卡时十分方便,此时只要先将保护器上的电源开关切断即可,PC 主机也不必关机,可以节省再次重新开机的时间,因此硬件接口在制作完后可以快速反复地在 PC 上配合软件驱动程序做功能的验证。

- 直流电源供应器

在进行电子电路实验时,一定都会备有一部可调式的直流电源供应器,如 0V~20V,双电源供应器(限流 5A),可是在做数字硬件实验时便不是很理想。怎么说呢?因为微处理器系统常用的电压是 5V,若要常常将电源供应器的电压调整至 5V,岂不麻烦,万一不小心将电压调至 7V,那么 CPU 及 TTL IC 岂不报销。所以有必要备有一部 5V 的专用电源供应器,而限流 5A 即够用,市面上有交换式电源 5V 供应器,体积不大。

另外若要设计一些模拟电路,或接口电路,如使用 A/D(模拟至数字转换器)D/A(数位至模拟转换器)和 OPAMP(运算放大器),就可能会用到+12V 及-12V,甚至-5V 的电压了,当然一般可取至 PC 插槽上的电源,若是线路本身需用电流相当大时,建议还是外加电源供应器,此时可以再选购一台 PC 上的交换式电源供应器来供电,+5V、-5V、+12V、-12V 的电压都有了,而且电流数也够。

- PC 个人电脑

本书部分接口的实验是在 PC 上完成的,TURBO C 控制程序也需在 PC 上进行编译和链接,使用 XT 以上电脑均可,使用 586 的执行速度要快很多,可以加快实验时 TURBO C 编译的速度,提高实验的效率。

- 在线仿真器或 CPU 模拟器(ICE, In Circuit Emulator)

ICE 是单片机系统开发上效率最高的工具。一般它是通过 RS232 串行传输接口来与

PC 连接, 通过 40 个脚的信号线, 连至目标电路板的 CPU 插座上。由于它直接模拟 CPU 动作, 所以功能相当强, 对于系统板上硬件的侦错、软件的测试、除错都行, 是单片机设计者最佳的助手, 因为功能强, 市场的价格也较贵。

- ROM 模拟器

单片机系统执行时必需将其程序代码放置于内存中, 一般是烧录在 EPROM 中, 如果单片机像 8051 那样内部有程序代码的存储空间, 在开发时也可以外接至外部的 EPROM 来做程序的测试开发, 因此只要有 ROM 模拟器, 便可以做程序的模拟, 待程序设计完成, 最后才将其烧录至 EPROM 中。ROM 模拟器适合多种 CPU 控制程序的开发, 如 6502、Z80、8088 及单片机 8048、8051、Z8 等, 因此对单片机 8051 的初学者非常适用, 最主要的优点是其价格便宜、易学易用。

- 烧录器

最早的烧录器只做 EPROM 的烧录用, 近几年来由于单片机使用的普及, 现在已有许多厂商在烧录器中同时提供烧录 8748、8751、89C51、Z8 单片机的功能。此外也包括像 PAL(可编程逻辑元件)的烧录, 因此一部机器, 具有多种功能。

- 紫外线清洗机

紫外线清洗机用来将 EPROM 或 8751 的程序擦除, 以便由烧录器重新将数据烧入其中。

清洗机本身附有定时器, 时间一到便将电源切断, 通常清洗时间以 20 分钟为宜, 超过 30 分钟以上恐怕有破坏电子元件的危险。清洗机也可以自制, 但是紫外线灯管所发出的紫外线会伤害眼睛, 若采用自制的, 在操作时必需外加一密封的容器避免紫外光照射出来。

- 面包板

面包板是做模拟及数字逻辑电路实验时必备的器材, 可以将电子元件及 IC 反复地插在板子上, 通过单心线做电路的实验, 在接口实验中当然也需要它, 例如使用多功能实验卡或自制的接口板是插在 PC 上, 而所有实验可以通过信号线连接到面包板上来做相关电路的控制实验。

- 基本的焊接工具

电烙铁及焊锡、剪线钳、尖嘴钳、斜口钳等。

2.2 软件使用工具

本书所使用的软件工具包括以下 3 种:

- PC 上的 TURBO C 语言编译器;
- 8051 C 语言编译器;

- 8051 汇编语言编译器。

PC 上的 TURBO C 是用来学习基本的 C 语言程序设计用的, 另外本书中所提到的声卡驱动程序及声控程序, 无线电遥控模块, 也可以用 TURBO C 来设计。

而 8051 C 语言编译器是用来设计一般 8051 控制程序用的, 本书的硬件工作平台是以 8051 多功能控制板为主。

8051 是目前市面上相当流行的单片机, 因此国外有不少软件厂商纷纷推出支持 8051 的 C 语言编译器, 在台湾常见的有 IAR 公司及 2500 A.D. 公司发售的编译器, 上述这些编译器功能均相当的齐全, 属专业版, 价格自然不低。几年前加拿大 DAVE DUNFIELD 软件公司推出了一套 8051 C 编译器 MICRO-C51, 功能上虽然不支持浮点运算, 但价格是学生、业余玩家都能接受的。于是在本书中有关 8051 的实验控制程序是以此套 C 编译器进行编译的, 读者手上若有其他的 C 语言编译器, 只要稍加修改一下原始程序, 便可以适合自己的系统使用。

基本上, 使用 C 语言编译器设计程序有以下好处:

1. 程序易编写并具结构化。
2. 程序易读易懂, 修改容易。
3. 除错方便。
4. 可移植性好, 可在 PC 上测试, 再移植到 8051 C 编译器上编译执行。

因此从前写一个 8051 汇编语言控制程序的专业设计, 可能要花掉 3 天的时间。现在只要一个晚上便可设计好程序而进行排错和功能修改, 在效率上至少是 3 倍以上, 聪明的您是不是有点心动了呢?

本书的 8051 C 程序均是在 MICRO-C51 上进行编译产生可执行文件的, 通过 ROM 模拟器下载至 8051 电路板或是 8051 无线电遥控模块上而执行程序, 从软件编译到程序下载的测试环境都作了详细的操作说明, 看过本书, 您将会发现开发 8051 的 C 语言控制程序就像在 PC 上写 TURBO C 那么的方便、快速而有效。

当然读者也可以用汇编语言来设计 8051 的驱动程序, 只是一旦熟悉了 C 语言后, 您设计程序的效率会提高很多, 甚至可以修改 PC 上用 C 语言设计的软件, 把它改写成 8051 C 编译器可以接受的程序代码, 用在电路板上来做控制的工作, 因此可以节省很多程序设计的时间。当然典型的控制程序汇编语言一样可以设计, 只是要多花一些时间罢了, 基本上也不困难。

2.3 硬件接口卡

根据本书的实验要求选用下列硬件:

1. 万用实验接口卡
2. 8051 多功能控制板 P51_PCB

3. 8051 无线遥控模块	RF51
4. ROM 模拟器	ROM_EMU
5. PC/8051 语音控制实验卡	SP_CARD
6. PC/8051 多功能实验卡	MIO
7. 红外线学习板	IR_PCB
8. DSP 语音识别声控板	DSP_PCB
9. 89CXX 烧录模拟器	EPM89

其中万用实验接口卡，在一般电子商店都可买到。

1. 万用实验接口卡

可以根据本书所介绍的 8051 控制电路的部分电路将上面电路卡焊在此块万用实验接口卡上，制作属于自己的 8051 单片机控制板，市面上现成的电路板单片机功能可能无法完全满足自己课题的设计，可以使用电路板来做特殊硬件的制作，所以一块电路板，可做基本 I/O 的实验，可以针对制作的课题来扩充 I/O 接口，做单片机产品的开发。

2. 8051 多功能控制板 P51_PCB

此块控制板可供一般学生实验、课题制作及专业设计使用，本书所提到的实验可以使用此块控制板来得到验证，并提供有印刷电路板供初学者 DIY(Do It Yourself 自己做)自行焊接，这是一块标准的 8051 控制板，并考虑到将来功能的扩充，相关的硬件应用还在陆续的开发中。

3. 8051 无线遥控模块 RF51

8051 无线遥控模块是无线遥控系统的主体部分，可以独立操作，也可以配合其他的单片机系统做额外的特殊无线遥控功能的设计，而书中所提到的单片机系统是以 8051 多功能控制板为例来说明的。

4. ROM 模拟器 ROM_EMU

8051 控制程序是以 ROM 模拟器下载程序及测试，读者手上若有 8051 ICE 则更好，可以做进一步程序除错的功能。对一般单片机程序的初学者而言，用 ROM 模拟器开发程序即可，主要是价格便宜，如果有额外的预算再购买 ICE 也不迟，何况 ROM 模拟器可以适合多种 CPU 下载程序和测试，若要写 Z80 CPU 的控制系统使用 ROM 模拟器便可，不必再购买 Z80 ICE。

5. PC/8051 语音控制实验卡 SP_CARD

语音卡主要是用在 PC 上做语音的相关实验及设计声控电脑用。值得一提的是在 8051 多功能控制板上设计有一个与 PC AT 62 PIN 部分 I/O 信号兼容的插槽，一些 PC I/O 接口卡可不须修改线路而用 8051 来做控制，因此此块语音卡也可以插在 8051 单片机电路板上做语音录音与放音的实验。

6. PC/8051 多功能实验卡 MIO

这是一块 PC 的 I/O 接口卡, 用于 PC 的接口芯片控制, 使今后在 PC 上不论是课题制作还是接口实验上更有效。最重要的是此块接口卡提供标准的 I/O 接口设计, 可以在不须修改线路的情况下移植到 8051 多功能控制板上来执行, 只要重新改写 8051 的控制程序即可, 如此一来可以实现硬件 I/O 接口资源共享的优点。

7. 红外线学习板 IR_PCB

为了方便做红外线接口的相关实验, 我们设计了一块红外线接口实验板(IR_PCB), 可以分析一般市面上红外线遥控器所发射出来的信号, 也可以观察其波形, 还可以利用 PC 来存储多组信号, 以备将来使用。此外利用 IR_PCB 控制板可以进一步来设计红外线遥控器的解码程序。

8. DSP 语音识别声控板 DSP_PCB

这是一块以数字信号处理技术(DSP)制作的语音识别板, 可以配合 8051 多功能控制板来完成独立操作、可携带式的声控应用系统, 本系统适合特定语的单音、字、词的语音样本识别, 不限定说话语种, 国语、台语、英语都可。此套系统的识别率可达 95%以上, 并可以在嘈杂的工作环境下操作。系统配置及语音参考样本一旦输入后数据就可以长久保存, 系统采用模块化设计, 扩充性强。

9. 89CXX 烧录模拟器 EPM89

这是一块专门烧录 ATMEL 公司生产的 89CXX 系列 IC 的控制板, 可以烧录 89C51(4K)、89C52(8K)、89C55(20K)、89C1051(1K)、89C2051(2K)单片机, 烧录后可以直接模拟 40 PIN 或 20 PIN 8051 单片机。使用打印机并行口连接, 不须连接 I/O 接口卡, 可以用于笔记本电脑。采用交互式的软件控制程序, 单键指令操作, 完成烧录及模拟的工作。此外也可以做一般打印机并行口实验, 提供 16 BIT 输出, 8 BIT 输入, 并含有 TURBO C 的控制示范程序。

各块接口卡详细的功能及使用方法, 请参考相关章节的说明。

第 3 章 8051 C 编译器使用说明

若读者想以 C 语言做程序设计, 首先便要熟悉其编译器的特性, 如果已熟悉 PC 上任何一种 C 语言编译器的话, 使用 8051 C 编译器应该不难, 基本操作原理及步骤还是一样, 只是某些细节得注意。在本章中, 我们将介绍一套廉价的 8051 C 编译器系统 MICRO-C51 来编译本书所有的 8051 C 语言程序, 并说明如何快速的操作此编译器来产生可执行文件, 程序修改及编译的过程需反复不断的进行。以 PC 486 来做测试, 一般的小程序, 编译只需花费 4 秒钟(若以虚拟盘来操作则只需 1 秒), 便可以看到自己设计的 8051 C 程序执行, 既快又方便。

3.1 MICRO-C51 编译器特性

MICRO-C51 编译器(以下简称 MC51), 是加拿大的 DUNFIELD 软件开发公司发行支持单片机 8051 的 C 语言编译器。近几年来, 8051 使用相当普遍, 以 C 语言从事控制软件设计开发已成为流行的趋势, 市面上有不少 8051C 语言编译器, 不过价格均不低, 对于学生或业余玩家, 往往可望而不可及。笔者是单片机的业余玩家, 在看见此套开发工具后, 对于它的售价的确有些惊讶, 在好奇之余, 利用它来设计一些程序, 觉得小兵仍可立大功, 足以证明某些开发软件工具, 虽便宜却还是蛮好用的。

3.1.1 MICRO-C51 编译器特性

- 8051 C 语言编译器经济实用。
- 可执行文件小, 编译速度快。
- 按照标准的 UNIX C 语言编译器语法设计。
- 支持 5 种内存操作模式: 极小型、小型、压缩型、中型及大型模式, 可配合不同的系统内存设计。
- 提供多种函数库供程序设计调用, 并含有函数原始汇编语言程序供参考。
- 提供嵌套注释, 16 位常数, 可嵌入汇编语言, 可以用 C 语言设计中断程序。
- 未提供长整数、双精度、浮点运算、类型定义及位处理。
- 含有完整的原文技术设计及使用说明文件。
- 含有 8051 监控程序 (MON51.ASM 源程序), 提供指令如下:

B [bp address]	-Display/Set breakpoints
C <Reg> <value>	-Change register
D <aaaa>,[aaaa]	-Dump EXTERNAL memory
E <aaaa>	-Edit EXTERNAL memory

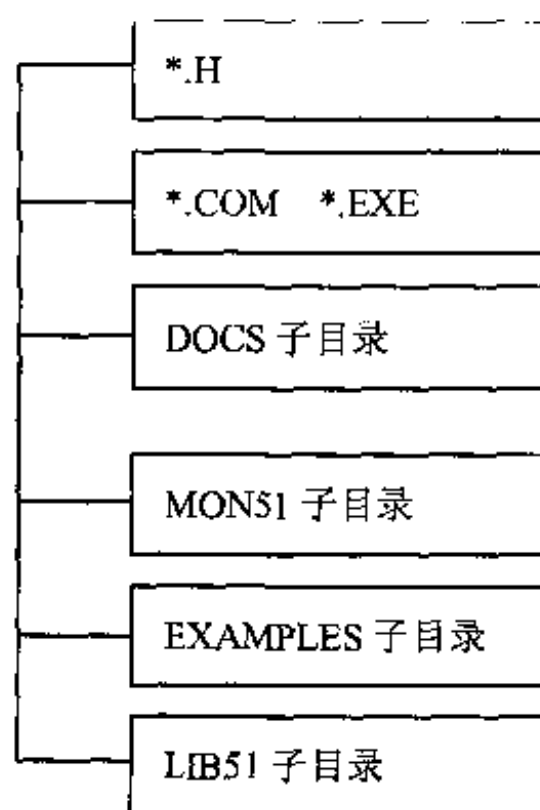
F <s>,<e><d>	-Fill a block of memory
G <aaaa>	-Go (begin execution)
I <aa>,<aa>	-Dump INTERNAL memory
L	-Load program into memory
R	-Display registers
S	-Single-Step one instruction
U <aaaa>,[aaaa]	-Unassemble program memory
?	-Display HELP summary

MC51 编译器基本上并不支持浮点运算，而且不是一个完整的 ANSI C 编译器，只可以算是一个 C 语言的“子集”编译系统，却可以解决 8051 程序设计中大部分的 C 程序编写的需求。在演算方面，自然不成问题。在语法方面，它支持 UNIX 编译器的语法标准。熟悉 8051 的朋友，自然会想到那么中断程序呢？计时/计数及串行传输的功能呢？一样可以用 C 语言来编写，多么惬意的事。当您的程序代码以 MC51 编译器执行测试后，若不够快，再用汇编语言改写也不迟。

总之，您可以少花钱一样拥有一套原版的 8051 C 语言编译器，因此以后做 8051 程序设计时也不必完全依靠汇编语言，一个指令一个指令逐一编写了，真的可以省下我们很多很多宝贵的时间和精力。

3.2 MICRO-C51 编译器组成

当读者将 MC51 工作磁盘安装至硬盘后，可以看到有如下的文件结构：



其中：

- *.H : C 语言的文件头定义。
- *.COM 及 *.EXE : MC51 的各个可执行文件。
- DOCS 子目录 : 完整的原文 MC51 操作技术数据文档。
- MON51 子目录 : 8051 监控程序 MON51.ASM 源程序及操作技术数据文档。
- EXAMPLES 子目录 : 内含一些 C 语言示范程序。
- LIB51 子目录 : 内含 MC51 的各种内存模式的程序库及程序库内的汇编语言程序源代码*.ASM 文件。

3.2.1 磁盘内容

Directory of D: \C51

READ	ME	3009	07-23-97	2: 13p
CC51	COM	4506	07-23-97	2: 13p
MCP	EXE	13481	07-23-97	2: 13p
MCC51	EXE	26951	07-23-97	2: 13p
MCO51	COM	14159	07-23-97	2: 13p
SLINK	EXE	11499	07-23-97	2: 13p
SLIB	COM	6785	07-23-97	2: 13p
SINDEX	COM	2280	07-23-97	2: 13p
SCONVERT	COM	3334	07-23-97	2: 13p
SRENUM	COM	3117	07-23-97	2: 13p
MAKE	COM	5907	07-23-97	2: 13p
TOUCH	COM	2447	07-23-97	2: 13p
DDSIDE	COM	33011	07-23-97	2: 14p
DDSIDE	HLP	7985	07-23-97	2: 14p
8051	IDE	1048	07-23-97	2: 14p
ASM51	EXE	17143	07-23-97	2: 14p
MACRO	EXE	11895	07-23-97	2: 14p
HEXFMT	EXE	11415	07-23-97	2: 14p
8051ADC	H	311	07-23-97	2: 14p
8051BIT	H	468	07-23-97	2: 14p
8051INT	H	3513	07-23-97	2: 14p
8051IO	H	189	07-23-97	2: 14p
8051REG	H	306	07-23-97	2: 14p

Directory of D: \C51\DOCS

MC51	DOC	205602	07-23-97	2: 14p
------	-----	--------	----------	--------

DDSIDE	DOC	49410	07-23-97	2: 14p
XASM51	DOC	41951	07-23-97	2: 14p
TECHSUPP	DOC	4053	07-23-97	2: 14p
CINTRO	DOC	119219	07-23-97	2: 14p
CATALOG		20671	07-23-97	2: 14p
REGISTER		1457	07-23-97	2: 14p

Directory of D: \C51\MON51

MON51	ASM	43044	07-23-97	2: 14p
MON51	DOC	10741	07-23-97	2: 14p

Directory of D: \C51\EXAMPLES

HELLO	C	193	07-23-97	2: 14p
MONITOR	C	2273	07-23-97	2: 14p
BJ	C	2217	07-23-97	2: 14p
TTT3D	C	5355	07-23-97	2: 14p
BYTESWAP	C	674	07-23-97	2: 14p
PTR2FUNC	C	1837	07-23-97	2: 14p
LONGCALC	C	3727	07-23-97	2: 14p
DEMO51	C	3201	07-23-97	2: 14p
DICE51	C	1580	07-23-97	2: 14p
INT51	C	1853	07-23-97	2: 14p
SERINT51	C	6348	07-23-97	2: 14p
BBSIO51	ASM	1588	07-23-97	2: 14p
8051	ASM	1536	07-23-97	2: 14p

Directory of D: \C51\LIB51

READ	ME	4320	07-23-97	2: 14p
TINY	LIB	959	07-23-97	2: 14p
SMALL	LIB	1268	07-23-97	2: 14p
COMPACT	LIB	1268	07-23-97	2: 14p
MEDIUM	LIB	1254	07-23-97	2: 14p
LARGE	LIB	1254	07-23-97	2: 14p
8051RLPT	ASM	1853	07-23-97	2: 14p
8051RLPS	ASM	1511	07-23-97	2: 14p
8051RLPC	ASM	2571	07-23-97	2: 14p

8051RLPM	ASM	2689	07-23-97	2: 14p
8051RLPL	ASM	3744	07-23-97	2: 14p
8051RLM	ASM	283	07-23-97	2: 14p
8051RLS	ASM	128	07-23-97	2: 14p
RUNLIB1	ASM	7754	07-23-97	2: 14p
RUNLIB2	ASM	916	07-23-97	2: 14p
RUNLIB3	ASM	1257	07-23-97	2: 14p
ATOI1	ASM	885	07-23-97	2: 14p
ATOI2	ASM	946	07-23-97	2: 14p
CONCAT	ASM	882	07-23-97	2: 14p
DELAY	ASM	523	07-23-97	2: 14p
ENABLE	ASM	127	07-23-97	2: 14p
FORMAT	ASM	6494	07-23-97	2: 14p
I2C	ASM	3904	07-23-97	2: 14p
ISFUNS	ASM	2530	07-23-97	2: 14p
LLIO1	ASM	1185	07-23-97	2: 14p
LLIO2	ASM	1162	07-23-97	2: 14p
LLIO3	ASM	1294	07-23-97	2: 14p
LLIO4	ASM	1606	07-23-97	2: 14p
LLIO5	ASM	680	07-23-97	2: 14p
LLIO6	ASM	427	07-23-97	2: 14p
LLIO7	ASM	1151	07-23-97	2: 14p
LONGJMP1	ASM	1004	07-23-97	2: 14p
LONGJMP2	ASM	1143	07-23-97	2: 14p
LONGJMP3	ASM	1261	07-23-97	2: 14p
LONGJMP4	ASM	1516	07-23-97	2: 14p
LONGMATH	ASM	5832	07-23-97	2: 14p
MALLOC2	ASM	5472	07-23-97	2: 14p
MATH	ASM	1898	07-23-97	2: 14p
MEMSET1	ASM	528	07-23-97	2: 14p
MEMSET2	ASM	594	07-23-97	2: 14p
MOVE1	ASM	1426	07-23-97	2: 14p
MOVE2	ASM	2306	07-23-97	2: 14p
PEEK	ASM	1303	07-23-97	2: 14p
PRINTF	ASM	341	07-23-97	2: 14p
RAND	ASM	616	07-23-97	2: 14p

SERINIT	ASM	723	07-23-97	2: 14p
SPRINTF	ASM	348	07-23-97	2: 14p
STRING1	ASM	1031	07-23-97	2: 14p
STRING2	ASM	454	07-23-97	2: 14p
STRING3	ASM	859	07-23-97	2: 14p
STRING4	ASM	455	07-23-97	2: 14p
TOFUNS	ASM	821	07-23-97	2: 14p
TPRINTF	ASM	5821	07-23-97	2: 14p
MALLOC1	ASM	5343	07-23-97	2: 14p

整个编译器系统组成相当简洁,读者若配合本章说明来操作 MC51,可以在短时间内熟悉其特性。本章仅将此编译器使用时的特殊注意事项加以整理,使读者可以快速的学会如何使用 MC51 编译器。

3.2.2 代码兼容性

MC51 按照标准 UNIX 编译器的语法设计,因此用 MC51 所写的程序无须太多修改便可在其他标准的编译器下进行编译。既然能与标准 C 编译器兼容,一些测试程序或是演算法可以先在 PC 上以较熟悉的编译器如 TURBO C 来完成,毕竟 PC 上的操作环境是我们较熟悉、习惯的工作环境,待一切验证无误后再做系统的转移及 I/O 必要功能的修改,便可以在 8051 单板上执行控制程序,在整体系统开发上效率较高。

MC51 现行版本还不支持下列标准 C 编译器所能处理的功能:

- 1.长整型 (Long)、双精度 (Double) 及浮点 (Float) 运算;
- 2.枚举类型 (Enumerated) 数据;
- 3.类型定义 (Type Define);
- 4.位处理 (Bit)。

然而 MC51 却提供了一些标准 C 编译器中未提供的功能,方便程序设计人员使用。

- 1.嵌套注释 (Nested Comments);
- 2.16 位字符常数;
- 3.可加入嵌入的汇编语言语句。

此外 MC51 综合的编译程序只占用内存 30K 字节,另配合虚拟盘操作,可以快速地完成程序编译的过程,省下很多时间。

3.3 内存模式

许多编译器在操作时都会将内存的使用情况分为多种模式,如 PC 上的 TURBO C 就分为 6 种工作模式。MC51 也不例外,一共提供有 5 种不同的内存操作模式,用来决定程

序执行时，全局变量及局部变量的存放位置，这 5 种模式分别是：

- 极小型模式 (Tiny Model)
- 小型模式 (Small Model)
- 压缩型模式 (Compact Model)
- 中型模式 (Medium Model)
- 大型模式 (Large Model)

下表列出了各种内存操作模式变量的有效存放位置。

内存模式	TINY	SMALL	COMPACT	MEDIUM	LARGE
编译选项	M=T	M=S	M=C	M=M	M=L
外部 RAM	没有	有	有	有	有
程序库 *.LIB	TINY	SMALL	COMPACT	MEDIUM	LARGE
起始模块*.ASM	8051RLPT	8051RLPS	8051RLPC	8051RLPM	8051RLPL

3.3.1 极小型模式

此模式并不需要外部 RAM，所有经初始化的全局变量均放在 ROM 中，而存取到外部内存均是使用 MOVC 指令，局部变量则是放在 8051 内部的 RAM 中。

优点：

1. 并不需要外加 RAM，使用最少的硬件便可执行程序。
2. 被初始化的变量仅占用 ROM 空间。

缺点：

1. 8051 内部只有 128 字节的 RAM 空间可供使用。
2. 初始化的变量无法被修改。

3.3.2 小型模式

程序 ROM 及数据 RAM 区域必需重叠放置在 64K 的寻址空间中，以 MOVX 指令存取数据。局部变量仍放在内部 RAM 中。

优点：

1. 允许 64K 字节的数据加上程序代码空间。
2. 局部变量可以快速的存取。
3. 被初始化的变量只占用 ROM 空间。

缺点：

1. RAM 与 ROM 必需重叠放在硬件中。
2. 所有程序代码及数据空间只有 64K 字节。
3. 局部变量空间相当有限。

4. 初始化过的变量无法被修改。

3.3.3 压缩型模式

类似小型模式的操作，不过局部变量被分配放在外部 RAM 中。

优点：

1. 允许 64K 字节的数据加上程序代码空间。
2. 局部变量的使用空间，可以是整个所有可以使用的外部 RAM 大小。
3. 初始化的变量只占用 ROM 空间。

缺点：

1. RAM 与 ROM 必需重叠放在硬件中。
2. 所有程序代码及数据空间被限制在 64K 字节中。
3. 局部变量存取较慢。
4. 被初始化过的变量无法被修改。

3.3.4 中型模式

类似小型模式操作，区别在于被初始化过的数据在程序执行时会先拷贝至 RAM 中，如此一来可以避免内存在硬件设计时 ROM 与 RAM 必需重叠的限制，同时允许被初始化过的变量可以在程序执行时进行修改。

优点：

1. 并不需要重叠的 ROM 与 RAM 设计。
2. 允许 64K 字节的数据空间及 64K 字节的程序代码空间。
3. 局部变量可以快速的存取。
4. 被初始化过的变量可以在执行时进行修改。

缺点：

1. 初始化过的变量占用 RAM 位置。
2. 局部变量的空间位置变得相当有限。

3.3.5 大型模式

类似压缩型操作模式，区别在于程序执行前被初始化过的数据会先拷贝到 RAM，如此一来可以避免 ROM 与 RAM 必需重叠设计的限制，同时允许初始化过的变量可以在程序执行时加以修改。

优点：

1. 并不需要 ROM 与 RAM 做重叠设计。
2. 允许 64K 字节的数据空间及 64K 字节的程序代码空间。
3. 局部变量可以使用整个所有 RAM 位置。

4. 初始化过的变量可以加以修改。

缺点:

1. 初始化过的变量占用 RAM 位置。
2. 局部变量存取较慢。

MC51 为了使用内部与外部内存位置, 利用 “register” 定义来加以区分。任何全局变量定义为 “register” 将会放在内部内存中, 就是存取时间较快。而未定义为 “register” 的整数变量会放在外部内存中。显然这与标准 C 编译器设计中 “register” 的使用方法不一致 (“register” 在正常情况下不可以使用在全局变量中), 之所以如此选择, 因为 “register” 变量存取时间快、效率高。

3.3.6 局部变量存取

在极小型、小型及中型内存操作模式下, 局部变量是放在 8031/8051 芯片内部的 RAM 中, 因为只有 120 字节 (芯片内部共有 128 字节的 RAM 空间, R0 至 R7 为工作寄存器, 此 8 字节, 编译器势必会用掉), 所以在使用上述的内存操作模式时, 对于局部变量的使用要多加考虑, 能省则省。若使用的芯片是 8032/8052, 则可以使用的内部 RAM 空间增至 248 字节。

在压缩型、大型内存操作模式下, 局部变量被放在外部内存中, 如此一来允许局部变量充分利用所有可用的 RAM 空间。

3.3.7 全局变量存放

全局变量若定义为 “register” 则会存放在内部内存中, 这些变量在定义时不能被初始化。若未定义为 “register” 的全局变量, 则会放在外部内存中。在极小型、小型及压缩型操作模式下, 任何在定义时被初始化的变量, 会被放在 ROM 中, 如此一来, 在程序执行时便无法改变其值了。

同理, 若定义为 “register” 的指针变量会指向内部内存位置, 而未定义为 “register” 的指针变量, 则指向外部内存。

3.4 编译程序

MC51 共有 5 个程序来完成一个编译系统的工作:

1. 前置处理器 (MCP.COM);
2. 编译器 (MCC51.COM);
3. 最优化处理器 (MCO51.COM);
4. 汇编语言编译器 (ASM51.COM);
5. 链接器 (SLINK.COM)。

3.4.1 前置处理器

前置处理器读取 C 语言程序源文件，执行 MACRO（宏）扩充后，再与头文件的内容合并，最后可以得到纯粹的 C 语言程序文件输出，功能较弱的前置处理器也可以合并到编译器内部，而省去此步骤。通常我们可以使用 define 的功能来定义一些基本运算，而由前置处理器来做处理，可以使整个程序设计起来更简洁。

3.4.2 编译器

编译器读取 C 语言源程序，检查其中是否有语法（Syntax）或语义（Semantic）的错误，若没有错误则将 C 语言转换为相对应的 8051 汇编语言程序。如果错误被检查出来时，则输出相关的错误消息提示用户修改源程序，并终止编译器的工作。

3.4.3 最优化处理器

最优化处理器读取编译器输出的汇编语言程序代码，逐一过滤程序代码，检查程序代码中是否有多余的无效语句。若有则以相同功能却能执行最优化的代码将之取代，通常经过最优化处理器处理过的程序代码，其文件大小较小，同时整个程序代码执行起来会更快些，但有时也有例外。

3.4.4 汇编语言编译器

汇编语言编译器将由编译器或最优化处理器产生的汇编语言程序代码转为 8051 目标文件（.OBJ）。若控制程序是用汇编语言来设计的话，也可以直接送到汇编语言编译器进行编译的工作，汇编语言编译器同时会做语法的侦错功能，只要有不合法的格式存在，将会停止编译的工作而显示相应的错误提示消息。

3.4.5 链接器

链接器将通过汇编语言编译器所产生的 8051 目标文件，链接必要的程序库而形成最后的可执行文件，其输出的程序代码属 HEX 文件，可以指定为 INTEL 或是 MOTOROLA HEX 格式。

图 3-1 所示为整个 MC51 的操作流程。一个 C 语言源程序经过前置处理器、编译器、可选择的最优化处理器做程序代码最优化处理后，再通过汇编语言编译器产生目标文件，最后经链接器链接而形成可执行的 HEX 文件，可用于一般的 ICE 开发系统的程序测试。

3.5 综合的编译程序

上节所谈的是整个 MC51 的各个细部工作，编译器本身提供有综合的公用程序 CC51.COM，使得一连串的前置处理、编译、汇编及链接的工作可以一气呵成，加速程序代码的产生。

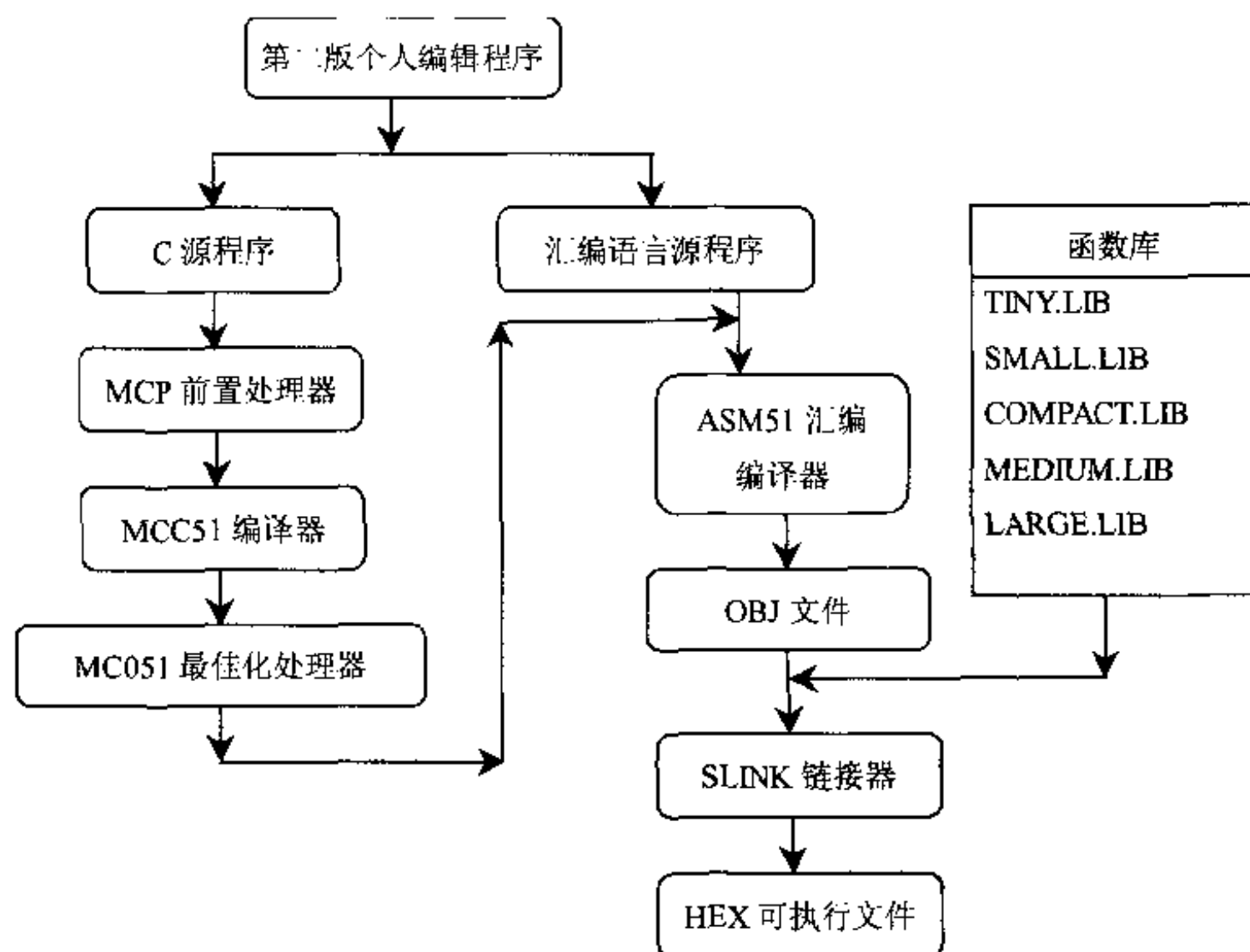


图 3-1 MC51 编译器的操作流程

3.5.1 CC51 的指令格式

CC51 C 程序文件名〔选项〕

有如下的选项指令（大小写均可）：

- A 产生汇编语言（.ASM）输出文件；
- C 在汇编语言输出文件中加上 C 源程序当做一对一的注释；
- F 折叠相同的字符串常数；
- I 产生 INTEL 的 HEX 格式文件（原设定产生 MOTOROLA 格式文件）；
- K 保留编译器执行时所有暂时产生的文件，不删除掉；
- L 产生汇编语言编译器编译后的列表文件（.LST）；
- M 启动汇编语言编译器中的 MACRO 处理器；
- O 使用 MC051 产生最优化程序代码；
- P 使用 MCP 做扩充的前置处理器处理；
- Q 编译器操作时若没有错误，并不产生任何的输出消息；
- X 产生扩充的汇编语言文件；
- M=TSCML 指 8051 内存操作模式（Tiny、Small、Compact、Medium、Large）；
- H=mcdir 指定 MC51 的工作目录路径；
- T=mctmp 指定编译工作时所产生的临时文件的存放路径；

3.5.2 编译器出现的错误消息

由于 MC51 是一个综合的公用程序, 若 C 语言源程序无法通过编译, 则会出现如下相对的错误消息:

错误代码	意义
2	找不到可执行文件 (请检查 MCDIR 及 PATH 设置)
3	找不到程序执行路径(请检查 MCDIR 及 PATH 设置)
254	程序执行时发现错误
255	编译器发现不正确的参数输入

3.6 工作环境设置

在将 MC51 工作磁盘安装至硬盘后, 必需先对编译器工作环境加以设置, 才能顺利的使用编译器来做程序的开发。包括对 PC 文件目录的设置及产生的程序代码如何放置到内存内, 内存操作模式的选择等。笔者手上的 MC51 是放在 D 盘 C51 文件目录下, 同时建立了虚拟盘来存放编译执行时产生的临时文件, 为了加快程序的执行, 可将部分编译程序的可执行文件搬到虚拟盘上来做程序的编译工作。

整个环境设定分为以下 5 个步骤:

步骤 1: 安装相关程序至硬盘

如 3.2 节编译器系统组成的说明, 将编译器程序安装到硬盘中。

步骤 2: 设置编译器相关的工作路径

在根目录下的自动批处理文件 AUTOEXEC.BAT 中加入下面 2 行:

```
SET MCDIR=D: \C51\
```

```
SET MCTMP=E: \
```

分别设置 MC51 的工作目录路径及临时文件的存放路径(E 盘为虚拟磁盘), 读者可以根据自己的工作路径而自行修改。例如从硬盘 C 来执行, 则做如下设置:

```
SET MCDIR=C: \C51\
```

```
SET MCTMP=C: \C51\
```

步骤 3: 决定内存操作模式

在上节中介绍过 5 种 MC51 的内存操作模式, 您应该清楚自己的 8051 控制板的硬件结构, 包括程序代码 ROM 空间的大小、外部数据 RAM 空间的大小, 由实际的硬件需求来选择适当的内存工作模式。对于最简单的 8051 硬件配置而言 (没有外接 RAM) 则选择极小型操作模式。在本书中主要的 8051 硬件工作是以 8051 多功能控制板 P51_PCB 为主 (详见第 4 章说明), 其硬件备有独立 64K 字节的 ROM 程序代码空间及 32K 字节的外

部 RAM 可供利用，如此一来只有中型模式及大型模式可供选择。

选择中型模式来操作时，局部变量可以快速的存取，程序执行速度较快，因此选择使用中型内存操作模式，所以程序库使用 MEDIUM.LIB。

步骤 4：修改编译器系统程序代码的执行起始地址

原 MC51 配备的 8051 硬件工作环境是 Dunfield 公司所开发的 DB-52 单片机单板，内含有除错系统，因此由 MC51 产生的程序代码起始地址都为 0800H，今天要如何修改成自己所使用的系统呢？可以修改编译器的起始模块（8051RLP?.ASM，?表各种内存操作模式，）内的“ORG”语句，而要进一步的参考数据可参阅程序库工作目录 LIB51 内 README 文件的说明。

由于我们决定以中型内存操作模式来产生 8051 程序代码，所以对 8051 RLPM.ASM 进行修改，可以发现旧的“ORG”语句如下：

```
?RAM      EQU $0000 RAM Start here
          ORG $0800 ROM Start here
```

修改为：

```
?RAM      EQU $0000 RAM Start here
          ORG $0000 ROM Start here
```

步骤 5：修改 MC51 的中断基地址设定

原 MC51 产生的程序代码中断基地址为 2048，此值定义在宏定义 8051INT.H 中，有关此部分的说明，读者可以参考工作目录 EXAMPLE 下的 ISR.C，中断范例程序内有更进一步的说明。所以接着对 8051INT.H 进行修改，原定义如下：

```
#define INTBASE 2048
```

修改为：

```
#define INTBASE 0
```

即将 8051 的 6 种中断向量地址设定在如下所示的绝对地址处：

中断来源	中断向量地址
IE0	3
TF0	11
IE1	19
TF1	27
SER	35
TF2	43

3.7 操作实例

在完成 MC51 工作环境的设定后，便可以利用 CC51 的综合公用程序来编译 8051 的 C 语言程序。首先写一个测试程序 TEST.C:

```
/* TEST.C */
#include "8051io.h"      /* 载入头文件 */
#include "8051reg.h"
#include "8051bit.h"
char *title="MC51 simple test program";
delay(int t)             /* 延迟子程序 */
{
    int i,j;
    for(i=0; i<t; i++)
        for(j=0; j<10; j++);
}
/*-----*/
main()
{
    while(1)
    {
        clrbit(P1.7); /* 设定 P1.7 位为低电位*/
        delay(100);   /* 延迟子程序 */
        setbit(P1.7); /* 设定 P1.7 位为高电位*/
        delay(100);   /* 延迟子程序 */
    }
}
```

其工作很简单，将从 8051 电路板上端口 1 位 7 (P1.7) 送出高低电位，使得连接其上的 LED 会交互闪烁着，这是一个简单的测试程序，用来验证整个编译器的工作环境设定是否正确。

接着使用 CC51 进行编译。若直接输入 CC51 <ENTER>则出现如下的消息，给出 CC51 的操作格式。

```
DDS MICRO-C 8051 Cross Compiler V3.1
Use:  CC51 <name>      [-acfiklmopqx h=homedir m=tscml
                        t=tmpdir] [symbol=value]
Copyright 1990-1994 Dave Dunfield
```

All rights reserved.

实例 1: 产生可执行 INTEL HEX 文件

CC51 TEST m=m-ip <ENTER>

即选择中型内存模式, 产生 INTEL 格式的 HEX 文件, 使用前置处理器。执行结果如下:

```
D: \C51>cc51 test m=m -ip
DDS MICRO-C 8051 Cross Compiler v3.1
test.c: Preprocess... Compile... Link MEDIUM... Assemble..
First pass... Second pass... 0 error(s).
All done
```

而产生了那些文件呢?

```
D: \C51>DIR TEST
Volume in drive D has no label
Volume Serial Number is 321B-1BDF
Directory of D: \C51
TEST C 336 08-21-94 11: 36a
TEST HEX 1645 08-21-94 4: 05p
2 file(s) 1981 bytes
6213632 bytes free
```

其中 TEST.HEX 为可执行的 HEX 文件, 内容如下:

```
D: \C51>TYPE TEST.HEX
: 020000000132CB
: 200032007581079002B579007A007B1C7C00EB4C601CE493A3CAC583CAC9C582C9F0A3CA85
: 20005200C583CAC9C582C91BBBFFE31C80E090001CE4F012028D12001B80FBC82581C8224E
: 20007200C92581C97A0022D083D082CF2581F581CFC082C08322CF2581F581CF22FBE0CB12
: 2000920022FCE0FBA3E0CC22FAE0F9A3E0CA2229F9E5F03AFA227C00CB30E7011CCB227578
: 2000B200F00030E70215F022FDE493A3FEE493A3FF4E7009E493FE740193FF800BE493A3E8
: 2000D200B5050BE493B5F0068F838E82E473A380D8FEEBA4FD8EF0ECA4FCEB8EF0A4C5F05D
: 2000F2002C2DC5F02211FD8EF0ED22C002C001AAF0F97E007D007F11C3E933F9EA33FADFB4
: 2001120008F5F0E9D001D00222ED33FDEE33FEC3ED9BF5F0EE9C40E0ADF0FED380DBBB0098
: 200132000122C333C5F033C5F0DBF722BB000122C3C5F013C5F013DBF7224028002240111
: 20015200C5F03400C5F022C394028003C39401C5F09400C5F022315EF4C5F0F4C5F0224531
: 20017200F07022042231C04024801A31C0401E7014801A31C0400E7014800A31C0500E804D
: 200192000431D46008E4F5F02231D460F8E4F5F0042231D440F780ED31D440F170E780ED02
```



```
: 2001B20031D440E170E780DD31D450E180D7C39BC5F09C20E70720D207C345F02220D2F90B
: 2001D200D322C5F0C39C7003E5F09B2290001CE0A360EEE0FFA3E0A3CF2582F582EF3583E9
: 2001F200F58380EB0581058105810581740075F00078FB12006DF608A6F078FB12006DE6BB
: 200212000886F078F712006D860308860412017745F07003020284801678FB12006DE60815
: 2002320086F012015018F608A6F012015E80CB740075F00078FD12006DF608A6F078FD1283
: 20025200006DE60886F07B0A7C0012017745F07003020281801678FD12006DE60886F01203
: 20027200015018F608A6F012015E80D102026802022B158115811581158122C297746475F2
: 20029200F000C0E0C0F01201F615811581D297746475F000C0E0C0F01201F61581158102AA
: 1F02B200028D22020043383035312073696D706C6520746573742070726F6772616D00C7
: 00000001FF
```

此 HEX 文件便可以下载至一般 ICE 来验证其功能是否正常了。

实例 2: 产生汇编语言程序文件

CC51 TEST m=m -ipa <ENTER>

即选择中型内存模式，产生 INTEL HEX 文件，启动前置处理器，同时产生汇编语言程序文件 TEST.ASM。

执行结果：

```
D: \C51>cc51 test m=m -ipa
DDS MICRO-C 8051 Cross Compiler v3.1
test.c: Preprocess... Compile... All done
```

果然产生了 ASM 文件：

```
D: \C51>DIR TEST
Volume in drive D has no label
Volume Serial Number is 321B-1BDF
Directory of D: \C51
TEST C 336 08-21-94 11: 36a
TEST ASM 1352 08-21-94 4: 28p
2 file(s) 1688 bytes
6193152 bytes free
```

实例 3: 产生 C 语言注释文件

CC51 TEST m=m -ipac <ENTER>

希望产生汇编语言文件 TEST.ASM 并且附有 C 语言的注释。

执行结果如下：

```
D: \C51>cc51 test m=m -ipac
DDS MICRO-C 8051 Cross Compiler v3.1
```

test.c: Preprocess... Compile... All done

检查相关文件:

D: \C51>dir test

```
Volume in drive D has no label
Volume Serial Number   is 321B-1BDF
Directory of D: \C51
TEST      C      336 08-21-94  11: 36a
TEST      ASM 2074 08-21-94:  32p
           2 file(s)          2410  bytes
           6184960  bytes free
```

观看 TEST.ASM:

D: \C51>type test.asm

```

*#file test.C
*1:
*#file 8051io.h
*5:
*6:
*9:
*10: extern register printf(), sprintf(), concat();
*11:
*12:
*#file 8051reg.h
*6:
*7:
*8: extern register unsigned char PSW, SP, DPH, DPL, P0,    P1, P2,    P3,
*9:  IP, IE,    TMOD, TCON, TH0, TL0, TH1, TL1,    SCON, SBUF, PCON,
*10:    T2CON, TH2, TL2, RCAP2H, RCAP2L;
*11:
*12:
*#file 8051bit.h
*8:
*9:
*13:
*17:
```

```
*21:
*22:
*#file test.C
*5:
*6: char  *title="C8051 simple test program";
$SE: 1
    DRW ?0+0
$SE: 3
title DS 2
*7:
*8: delay(int t)
*9: {
*10: int  i,j;
*11:     for(i=0; i<t; i++)
$SE: 0
delay INC SP
    INC SP
    INC SP
    INC SP
    MOV A,#0
    MOV B,#0
    MOV R0,#-5
    LCALL ?auto0
    MOV [R0],A
    INC R0
    MOV [R0],B
?1 EQU *
    MOV R0,#-5
    LCALL ?auto0
    MOV A,[R0]
    INC R0
    MOV B,[R0]
    MOV R0,#-9
    LCALL ?auto0
    MOV ?R3,[R0]
    INC R0
```

```
MOV ?R4,[R0]
LCALL ?lt
ORL A,B
JNZ *+5
LJMP ?2
SJMP ?3
?4 EQU *
MOV R0,#-5
LCALL ?auto0
MOV A,[R0]
INC R0
MOV B,[R0]
LCALL ?inc
DEC R0
MOV [R0],A
INC R0
MOV [R0],B
LCALL ?dec
SJMP ?1
?3 EQU *
*12: for(j=0; j<10;j++);
MOV A,#0
MOV B,#0
MOV R0,#-3
LCALL ?auto0
MOV [R0],A
INC R0
MOV [R0],B
?5 EQU *
MOV R0,#-3
LCALL ?auto0
MOV A,[R0]
INC R0
MOV B,[R0]
MOV R3,#10
MOV R4,#0
```

```
LCALL ?lt
ORL A,B
JNZ *+5
LJMP ?6
SJMP ?7
?8 EQU *
MOV R0,#-3
LCALL ?auto0
MOV A,[R0]
INC R0
MOV B,[R0]
LCALL ?inc
DEC R0
MOV [R0],A
INC R0
MOV [R0],B
LCALL ?dec
SJMP ?5
?7 EQU *
LJMP ?8
?6 EQU *
LJMP ?4
?2 EQU *
*13: }
DEC SP
DEC SP
DEC SP
DEC SP
RET
*14:
*15: main()
*16: {
*17:   while(1)
main EQU *
?9 EQU *
*18: {
```

```

*19:      asm{
* CLR P1.7
  CLR P1.7
*};
*20:      delay(100);
  MOV A,#100
  MOV B,#0
  PUSH A
  PUSH B
  LCALL delay
  DEC SP
  DEC SP
*21:      asm{
* SETB P1.7
  SETB P1.7
*};
*22:      delay(100);
  MOV A,#100
  MOV B,#0
  PUSH A
  PUSH B
  LCALL delay
  DEC SP
  DEC SP
*23: }
  LJMP ?9
?10 EQU  *
*24: }
  RET
$SE: 1
  DB 67,56,48,53,49,32,115,105,109,112,108,101,32,116,101,115
  DB 116,32,112,114,111,103,114,97,109,0
$SE: 3
?0 DS 26

```

其中前面标有*号的为源 C 语言程序 TEST.C 的行号。例如:

```
*20 : delay(100);
```

是 TEST.C 中行号为 20 的程序语句。

实例 4：产生汇编语言的列表文件

CC51 TEST m=m -ipl <ENTER>

产生列表文件 TEST.LST 执行结果：

```
D: \C51>cc51 test m=m -ipl
DDS MICRO-C 8051 Cross Compiler v3.1
test.c: Preprocess... Compile... Link MEDIUM... Assemble...
First pass... Second pass... 0 error(s).
All done
```

检查文件：

```
D: \C51>DIR TEST
Volume in drive D has no label
Volume Serial Number is 321B-1BDF
Directory of D: \C51
TEST      HEX      1645  08-21-94   4: 40p
TEST      C         336  08-21-94  11: 36a
TEST      LST     30791 08-21-9  44: 40p
          3 file(s)          32772 bytes
                              6152192 bytes free
```

其中 TEST.LST 为列表文件。

3.8 以 ROM 模拟器来做程序测试

本节说明如何利用廉价好用的程序开发测试工具 ROM 模拟器，配合 MC51 编译器使用 C 语言来设计 8051 控制程序，而硬件仍是以 8051 单板单片机为主。图 3-2 所示是结合 ROM 模拟器加上 MC51 综合程序 CC51.COM，来做程序开发测试的工作流程。其中 HEXBIN 是一个公用程序在 MC51 中，用来将 CC51 所产生的 HEX 格式文件转换为可执行的二进制文件。整个操作可用下面的一个批处理文件 X.BAT 来完成，此批处理文件使用中型内存模式(控制板上有使用外部 RAM)来编译 C 程序。本书大部分程序均是以 8051 多功能控制板(P51_PCB)为硬件测试平台，在控制板上安装有外部 SRAM(6116 2K 字节)，因此适合使用此批处理文件来快速编译程序。

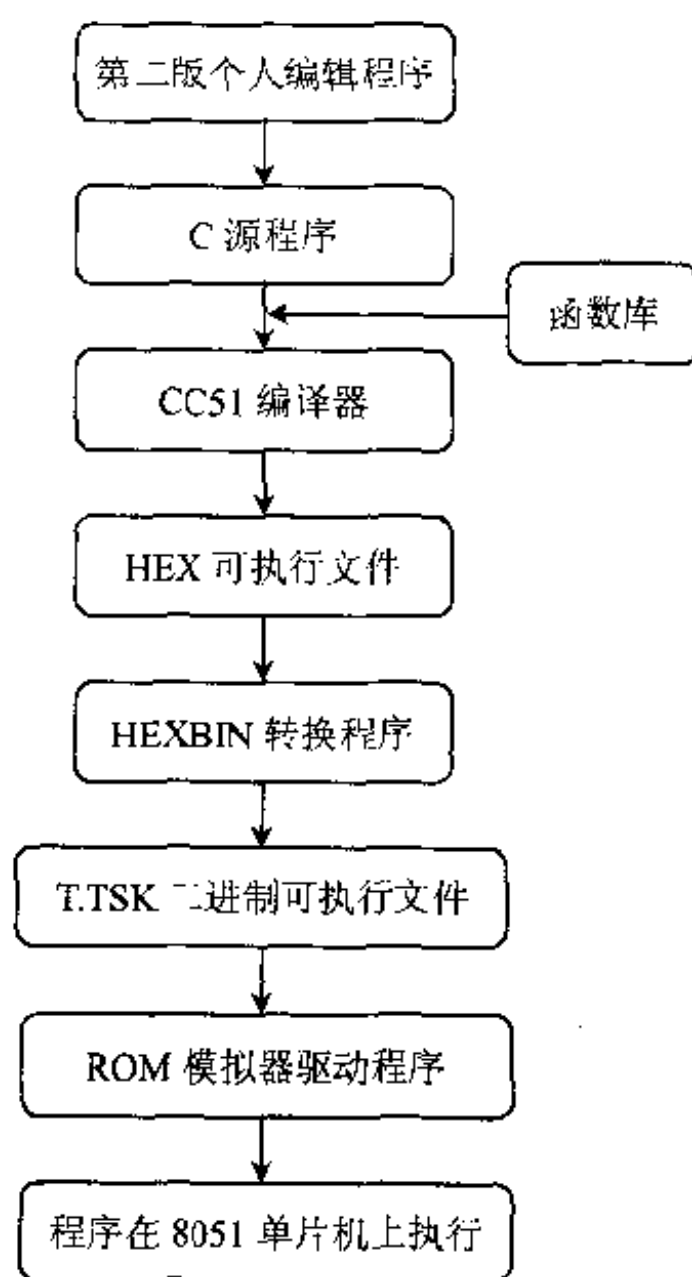


图 3-2 结合 ROM 模拟器的 MC51 操作流程

3.8.1 X.BAT 内容

```

@echo off
copy c: %1.c t.c          <-- 编译文件名都为 t.c
echo MC51 comp....        <-- 显示开始编译的消息
cc51 t.c m=m -ip          <-- 使用中型内存模式来编译程序
IF ERRORLEVEL 1 GOTO error <-- 若编译出错时则跳至标号 error
hexbin t.hex t.tsk i 1    <-- 将 HEX 文件转换为可执行文件
emu t.tsk 5 d             <-- 通过 ROM 模拟器下载程序并执行程序
se                         <-- 执行 RS232 监控程序
goto end
: error
echo MC51 comp. error!!!   <-- 显示编译错误的消息
: end
@echo on
  
```


其中

cc51.com : MC51 综合的编译程序。
 hexbin.exe : HEX 至二进制文件的转换程序。
 emu.exe : ROM 模拟器的驱动程序。
 se.exe : PC 通信串行端口 2 的 RS232 监控程序, 用来显示 8051 单板上内部执行时传出的任何消息。

若程序在编译的过程中有错误产生, 则停止整个批处理文件的执行, 并出现 comp error!!! 的消息而返回到 DOS 系统。

若以极小型内存模式(控制板上未使用外部 RAM)来编译 C 程序, 批处理文件可以使用 T.BAT。

3.8.2 T.BAT 内容

```
@echo off
copy c: %1.c t.c          <-- 编译文件名都为 t.c
echo MC51 comp...         <-- 显示开始编译的消息
cc51 t.c m=t-ip           <-- 使用极小型内存模式来编译程序
IF ERRORLEVEL 1 GOTO error <-- 若编译出错时则跳至标号 error
hexbin t.hex t.tsk i l    <-- 将 HEX 文件转换为可执行文件
emu t.tsk 5 d             <-- 通过 ROM 模拟器下载程序并执行程序
se                         <-- 执行 RS232 监控程序
goto end
: error
echo MC51 comp. error !!!  <-- 显示编译错误的消息
: end
@echo on
```

以下将上节介绍的测试程序 TEST.C 结合 ROM 模拟器及所设定的虚拟盘来做完整的 8051 程序开发示范, 有以下 3 个步骤:

步骤 1: 安装 ROM 模拟器至 PC 上

将 PC 主机关掉, 安装 ROM 模拟器至 PC 插槽上, 同时将 ROM 模拟器连至 8051 单板上, 接好 RESET 控制线。将 PC 电源打开, 一切应正常操作, 不然的话又要检查 ROM 模拟器安装是否正确。

步骤 2: 将工作目录下的可执行文件拷贝到虚拟盘

如此一来程序从虚拟盘中, 可以快速的载入到内存内执行, 加速编译的动作, 操作如下:

Copy *.* E: <ENTER>

检查虚拟盘中的所有文件:

E: \>dir

Volume in drive E is MS-RAMDRIVE

Directory of E: \

T	C	336	08-21-94	11: 36a
LARGE	LIB	1164	07-28-94	11: 38a
ASM51	COM	17056	07-28-94	11: 38a
CC51	COM	4259	07-28-94	11: 37a
T	TSK	721	08-21-94	11: 30p
T	HEX	1645	08-21-94	11: 30p
HEXBIN	EXE	13471	08-17-93	12: 12p
MACRO	COM	11432	07-28-94	11: 38a
MCC51	COM	28524	07-28-94	11: 37a
MCO51	COM	14142	07-28-94	11: 37a
MCP	COM	12414	07-28-94	11: 37a
SE	EXE	11654	08-15-94	11: 12p
SLIB	COM	6716	07-28-94	11: 37a
SLINK	COM	10606	07-28-94	11: 37a
8051ADC	H	321	07-28-94	11: 38a
8051BIT	H	478	07-28-94	11: 38a
8051INT	H	3516	08-10-94	11: 56p
8051IOH		199	07-28-94	11: 38a
8051REG	H	316	07-28-94	11: 38a
8051RLPL	ASM	12039	08-10-9	43: 03p
CS1	H	647	08-10-9	48: 36p
MC51	H	647	08-10-9	48: 37p
X	BAT	203	08-21-94	11: 27p
EMU	EXE	26386	07-05-94	12: 00p
24 file(s)		178892	bytes	
		1341952	bytes free	

步骤 3: 执行批处理文件 X.BAT

操作如下:

X TEST <ENTER>

执行结果如下:

```
E: \>x test
      1 file(s) copied
MC51 comp....
DDS MICRO-C 8051 Cross Compiler  v3.1
t.c:  Preprocess... Compile... Link MEDIUM... Assemble...
First pass... Second pass...  0    error(s).
All done

***** Hex to Binary converter V3.3 *****

Wait...
INTEL Hex to binary converter
Convert  complete

Open the file :  t.tsk ok !
file :  t.tsksize :  721
<< EMU.EXE  64k*8 ROM EMU V1.0 >>
Copyright by VICTOR LAB. 1994/4  All right reserved
TEL:  (07)2260258
download << t.tsk >> to  RAM  721 bytes ok !!
RED LED light ....
```

此时可以看到 ROM 模拟器上的红灯闪亮一下，同时位于 8051 电路板上工作的 LED 指示灯持续闪烁着，表示整个系统正确无误。而读者可以发现以虚拟盘来执行此批处理文件大约只需 1 秒钟的时间，相当的快，也就是说从修改完源程序并保存文件后，回到 DOS 状态下，执行批处理文件，在 1 秒后便可以看见单板上程序执行的结果了。至此我们已完成了一套快速的 8051 系统开发工作环境。在后面的章节中我们准备说明 MC51 在程序设计上的一些特殊技巧。

3.9 使用 89C51 烧录模拟器来做程序测试

本节说明如何利用方便实用的程序开发测试工具 89C51 烧录模拟器 EPM89，配合 MC51 编译器使用 C 语言设计 8051 控制程序，而硬件是以一般自制的 8051 单板单片机来做测试。

烧录模拟器 EPM89 的操作相当简单，一行指令便可以完成快速载入文件、烧录、直接模拟 89C51 的工作，结合 MC51 综合程序 CC51.COM 来做程序开发测试后，整个操作过程跟上一节所介绍的批处理文件 X.BAT 类似，文件名为 X1.BAT。

3.9.1 X1.BAT 内容

```

@echo off
copy c: %1.c t.c          <-- 编译文件名都为 t.c
echo MC51 comp....
cc51 t.c m=m-ip           <-- 使用中型内存模式来编译程序
IF ERRORLEVEL 1 GOTO error <-- 若编译出错时则跳至标号 error
hexbin t.hex t.tsk i 1    <-- 将 HEX 文件转换为可执行文件
epm89 t.tsk d             <-- 通过烧录模拟器下载程序并执行程序
se                         <-- 执行 RS232 监控程序
goto end
: error
echo MC51 comp. error !!!  <-- 显示编译错误的消息
: end
@echo on

```

若程序在编译的过程中有错误产生，则停止整个批处理文件的执行，并出现 comp error!!!的消息而返回到 DOS 状态。

操作如下：

X1 TEST <ENTER>

执行结果如下：

```

c: \>x1 test
      1 file(s) copied
MC51 comp....
DDS MICRO-C 8051 Cross Compiler  v3.1
t.c:  Preprocess... Compile... Link MEDIUM... Assemble...
First pass... Second pass...  0    error(s).
All done

***** Hex to Binary converter V3.3 *****

Wait...
INTEL Hex to binary converter
Convert  complete

-----
EPM89 (89C51/C52/C55) V1.1  86.12.25
Copyright (C) VICTOR uP  LAB. 1997,1998
TEL :  07-2260258  URL: wwwhome.fancy.com.tw/~ufvic

```

```
-----  
File << t.tsk >> length=721 BYTES  
Load file ok  
SUMS=42F1H  
Select << CHIP 89C51 >>  
Chip erased ....  
Over.....  
Read 4096 bytes and check blank ....  
BLANK check ok.....  
Programming 721 BYTES code.....  
TIME elapse : 0.1 second  
Read 721 bytes ....  
Read back SUMS=42F1H SUM0=42F1H  
Reset .....
```

此时可以看见烧录模拟器上的红灯闪亮一下，同时位于 8051 单板上工作的 LED 指示灯(P1.7)持续闪烁着，表示整个系统执行正确无误。

3.10 MICRO-C51 程序设计技巧

本节说明 MICRO-C51 程序设计的一些技巧，分为以下几个单元来做说明：

1. 存取 8051 单片机特殊功能寄存器；
2. 位的控制；
3. 中断子程序的设计；
4. 内存应对式 I/O；
5. 程序中加入汇编语言语句。

3.10.1 存取 8051 单片机特殊功能寄存器

我们知道单片机 8051 内部有一些特殊功能寄存器 (SFR)，如 P0、P1、P2、P3……对应的内存地址分别为 80H、90H、A0H、B0H……。在写汇编语言可以利用以下指令对端口 1 做输出控制：

```
MOV A,#0  
MOV P1,A
```

将端口 1 全部设为低电位。

在 MC51 下的 C 语言，则可以用下列方式来做控制：

```
#include "8051reg.h"
```

```
P1=0;
```

只要将定义文件 8051reg.h 含括进来即可, 原来在头文件中, 已对 8051 内部各个专用的寄存器加以定义了, 其内容如下:

```
/*
 * 8051 internal register definitions for DDS MICRO-C/51
 *
 * Copyright 1991-1994 Dave Dunfield
 * All rights reserved.
 */
extern register unsigned char PSW, SP, DPH, DPL, P0, P1, P2, P3,
    IP, IE,    TMOD, TCON, TH0, TL0, TH1, TL1,    SCON, SBUF, PCON,
    T2CON, TH2, TL2, RCAP2H, RCAP2L;
```

若想从端口 1 读取数据至变量 in 中, 可以用如下的指令:

```
#include "8051reg.h"
char in;
in=P1;
```

3.10.2 位的控制

8051 内部有一些特殊功能寄存器, 其中可以分别做位的控制, 如 P0、P1、P2、P3 等, 例如将端口 1 位 7 设为高电位可以用汇编语言指令:

```
setb P1.7
```

将其设为低电位可以用指令:

```
clr P1.7
```

将位取反可以用指令:

```
cpl P1.7
```

那么在 MC51 中呢? 可以用如下指令:

```
#include "8051io.h"    /* 一般 I/O 定义 */
#include "8051bit.h"    /* 位设定/取消宏定义 */
#include "8051reg.h"    /* 8051 寄存器定义 */
setbit(P1.7);          /* 设为高电位 */
clrbit(P1.7);          /* 设为低电位 */
cplbit(P1.7);          /* 位取反 */
```

上述二指令在 8051bit.h 定义中已用宏处理了。其内容如下：

```
/*
 * Macros to allow direct access to the I/O bits of the 8051
 * internal registers from DDS MICRO-C/51.
 * REQUIRES THE EXTERNAL PREPROCESSOR (MCP)!
 *
 * Copyright 1991-1994 Dave Dunfield
 * All rights reserved.
 */

#define setbit(bit) asm { /* Macro to set a I/O bit */
    SETB    bit
}

#define clrbit(bit) asm { /* Macro to clear an I/O bit */
    CLR bit
}

#define cplbit(bit) asm { /* Macro to complement an I/O bit */
    CPL bit
}
```

3.10.3 中断子程序的设计

在 8051 程序设计中若不会使用中断子程序的功能，往往无法完全发挥真正的芯片特性。用汇编语言来设计中断程序当然不成问题，那么改用 MC51 呢？一样可以用 C 语言来写中断程序，一点也不困难。在有关中断的头文件 8051INT.H 中，最后有关于中断向量的执行地址如下：

```
/*
 * Common 8052 interrupt vector addresses
 * (Must be decimal)
 */

#define IE0    3      /* External interrupt 0 */
#define TF0    11     /* Timer 0 rollover */
#define IE1    19     /* External interrupt 1 */
#define TF1    27     /* Timer 1 rollover */
#define SER    35     /* Serial port RX or TX */
#define TF2    43     /* Timer 2 rollover */
#define EXF2    43     /* Timer 2 external flag */
```

而中断程序的写法有如下的程序结构:

```
INTERRUPT (_TFO_) t0_isr()
```

```
{
    程序内容
}
```

便可以设计定时器 0 中断程序。

实例

以定时器 0 模式 1 做每隔 5ms 的定时中断,即每隔 5ms 执行一次中断服务程序 t0_isr()

```
#include "8051io.h"
#include "8051reg.h"
#include "8051bit.h"
#include "8051int.h"

#define HI 238 /* 5ms run 1 time ISR */
#define LO 0

INTERRUPT(_TFO_) t0_isr()
{
    TH0=HI;
    TL0=LO;
    .....
}

main()
{
    TMOD=0x01;
    TH0=HI;
    TL0=LO;
    IE=0x82;
    TCON= 0x10;
    .....
}
```

3.10.4 内存应对式 I/O

在 8051 控制板中, 外部 RAM 从 0000H 占用 32K 字节的空间, 从 8000H 以后的位置为 I/O 扩充用, 如何控制做 I/O 存取呢? 此段地址属于内存应对式 I/O, 因此可以用访

问相关内存的方式来存取 I/O 地址，如同存取内存内的数据一样。

实例 1

将外部 RAM 从 2000H 起的连续 100 字节全部填为“55H”，可用如下的程序来完成：

```
#define add (char *)0x2000
test_ext_ram()
{
    int i;
    char *pt;
    pt=add;
    for(i=0; i<100; i++)
        *pt++=0x55;
}
```

实例 2

从 8051 单板上 8255 端口 A 送出数据“55H”。

```
#define pa (*(char*) 0x8000)
#define pb (*(char*) 0x8001)
#define pc (*(char*) 0x8002)
#define cr (*(char*) 0x8003)

test_8255()
{
    cr=0x80;
    pa=0x55;
}
```

3.10.5 程序中加入汇编语言语句

虽然 C 语言是一种强有力的程序设计语言，然而有些时候 C 语言程序还是无法完全控制程序的工作（特别是低阶需要精确做时序控制的工作），则需要用汇编语言来完成这些特定的工作。用 C 语言来写控制程序的好处是程序设计效率高，但程序代码往往较长，因此遇到一些必需精确计算指令执行工作时间的控制问题，则一定得用汇编语言来设计了。

MC51 本身提供有“asm”语句，可以在 C 语言中加入汇编语言代码，有以下 2 种格式：

格式 1：

asm "汇编语言指令";

在此格式中，" " 内的所有指令将全部送到汇编语言编译器，注意在语句的最后要加上分号，作为语句的结束，如同一般 C 语言程序。

格式 2:

```
asm {  
    汇编语言指令  
}
```

在此格式中，所有位于 { } 间的每一行指令将原封不动送到汇编语言编译器做编译，若指令有错误的话，在执行编译时也会被检查出来，而告知错误的指令，并停止编译的执行。

实例

```
#include "8051io.h"  
#include "8051reg.h"  
#include "8051bit"  
/*-----*/  
test_asm()  
{  
    asm "nop";  
    asm "nop";  
    asm "clr P1.7";  
}  
/*-----*/  
test_asm1()  
{  
    asm {  
        NOP  
        NOP  
        CPL P1.7  
    }  
}  
/*-----*/  
main()  
{  
    test_asm();  
    test_asm1();  
}
```

第4章 8051 多功能控制板设计

由于 8051 在工业界及学校的使用相当普遍,在本章我们将介绍一块 8051 多功能控制板(P51_PCB),在本控制板上设计有可编程声效发生器的控制线路,并包括串行通信接口,一般 I/O 接口控制芯片 8255 线路,及 4 个 7 段数码管做一般用途的显示,同时线路备有扩充的功能。为了将部分 PC I/O 接口转移至 8051 电路板单片机来做实验,同时又设计了一个模拟 PC AT62 引脚的插槽,如此一来,从前在 PC 上设计好的 I/O 接口卡便可以不经线路的修改而轻易地移植到 8051 来控制了,只要重新改写 8051 的软件驱动程序即可,这对硬件的设计及实验是既省钱又省时间。

在本章中我们将本控制板的设计原理及基本常用 I/O 接口的功能加以说明,使读者能了解其功能,以便能利用它来学习 8051 的接口控制。

4.1 控制板设计概念

8051 多功能控制板是一块全方位设计的控制板,可供一般学生实验、专题制作、专业设计及自动控制使用。此块控制板还有一般单片机控制板的基本功能,并提供有相当多的硬件扩充接口,使用者可以根据需要而自行使用,同时提供有 C 语言及汇编语言的范例程序,因此初学者可以在最短的时间内来开发属于自己的控制系统。

4.1.1 单片机控制板基本功能

一般我们在做自动控制或是专题制作时,所考虑的单片机控制板,希望它能提供基本的输入输出功能,来方便我们控制程序的开发。而在单片机系统开发上经常会构建哪些基本的 I/O 功能呢?大概可以分为以下几种,下面分别说明:

1. 按键输入;
2. LED 工作指示灯;
3. 7 段数码管;
4. LCD 液晶显示器;
5. 喇叭或蜂鸣器;
6. RS232 串行接口。

1. 按键输入

用以控制程序执行流程或是输入数据,如果是前者的话,只需几个按键,3 到 5 个即已够用,若是后者的话则可能要 10 个左右的按键,像一般的单片机学习机则有 20 到 30 个按键。

2. LED 工作指示灯

LED 可以显示数字线上某一位的高低电位状态，是一种最简单的输出装置，当程序能正确执行时，可以令其闪动一下，若是不能如预期的工作，例如程序执行死循环，则看不到 LED 闪烁了，因此，使用 LED 可以指示系统目前的工作状态，此外 LED 也可当做电源指示灯用，当电路短路时，可以很容易的察觉。

3. 7 段数码管

7 段数码管是一般常用的输出结果显示元件，可用来显示 0~9 的数字，在某些应用场合中已够用了。若要显示多位数字时，可以串接起来利用单片机程序扫描的控制方式来做数字系统的显示。

4. LCD 液晶显示器

上述的 7 段数码管只能用来显示数字，若要显示文字（英文字），则可能就要考虑使用 LCD 了。它一次可以显示一行或两行的消息，因此在单片机控制程式的开发上可用来显示程序执行时的中间过程，在除错时很有帮助，另外可以做产品线上功能操作提示说明用。当然最佳的提示说明还是以语音为主，既方便又直接。

5. 喇叭或蜂鸣器

喇叭也是常见的输出装置，例如当有按键操作时，可以鸣响一声，用以指示有键被按下，也可以用来播放电脑音乐或是特殊声效。而蜂鸣器体积会比传统的喇叭要小，它是利用送出的振荡频率来发声的，一般的声响较为尖锐，如玩具、电子表、BP 机等，众多的消费性电子产品均会用得到。

6. RS232 串行接口

这是较高级的单片机控制系统中才会使用到的装置，主要是作为电脑连线控制用或是数据传输用。而在程序的开发过程中，可以给程序员一项非常有用的、方便的系统开发环境。上述的 LCD 可用来显示程序执行的中间过程，不过一次只能显示几十个字，若是能通过 RS232 串行接口将执行结果传回 PC，而由 PC 的串行传输接收程序负责接收消息将其显示在屏幕上，那么一项复杂的程序开发工作可能会变得更系统化，由于屏幕可以显示很多消息，这在程序开发上是最有效率的。任何在单片机上执行的数值运算结果，均可传回 PC 而显示出来，方便我们验证程序执行的正确性。

4.2 8051 多功能控制板特性

通过以上讨论，我们即针对常用的 I/O 接口来设计 8051 电路板单片机控制板，基本上它具有按键输入、LED 电源指示灯、LED 工作指示灯、7 段数码管及 RS232 串行接口，已经可以满足一般工程应用。此外为了共享原先设计在 PC 上的一般 I/O 接口卡，希望能移植到 8051 电路板上来做控制，于是特别加上了可以模拟 PC AT 插槽 62 PIN 部分引脚

的 I/O 接口信号。可以让使用者的 I/O 扩充方面更具弹性。现将控制板的特性列举如下：

- 一块全方位设计的控制板可供学生一般实验、专题制作及专业设计使用。
- 单片机可以使用 8031、8051、8751、89C51 等 8051 系列芯片。
- 系统频率：11.0592 MHz。
- 外部程序 ROM 可供选择 8K、16K、32K 或 64K。
- 外部数据 RAM 可供选择 2K、8K 或 32K。
- 8 个显示用 LED 及电源指示 LED。
- 16 个按键输入及工作指示 LED。
- 8 位 DIP 开关输入。
- 4 位 7 段数码管供一般用途使用。
- 可接蜂鸣器或一般喇叭。
- 有 RS232 通信接口。
- 使用声效芯片 AY-3-8910 做特殊声效、音乐产生控制。
- 与 PC AT 62 PIN 部分 I/O 信号兼容的插槽，一些 PC I/O 卡，可不通过线路修改而直接由 8051 控制板来做控制。
- I/O 引脚提供针座连接可方便与面包板连接来做实验。
- 控制板含 5V 电源稳压。
- 可扩充接口：两组 8255 接口、LCD 接口、D/A 接口、语音放音接口及看门狗电路防止程序死机。
- 可扩充接口：数据 RAM 电源备份电路，关机时数据不丢失。
- 可连接“PC/8051 多功能实验卡”MIO 做 A/D、D/A、UM3567 及 AD_LIB 声效的实验及设计。
- 可连接“PC/8051 语音实验卡”SP_CARD 来做语音录放音的相关实验。
- 提供 C 语言及汇编语言的范例程序磁盘。

4.3 8051 基本控制电路

8051 的程序代码可以外接程序 ROM 来执行，也可以使用内部 4K 字节的空间(如 8751、89C51)。图 4-1 为 8051 的基本工作电路，8051 EA 引脚连接 5V 电源，表示由内部程序 ROM 来提供程序代码。此电路可以使用的单片机有 8751 及 89C51，而程序的测试方法可以使用 ICE 或是直接烧录单片机，直接烧录单片机做测试较麻烦且会花上许多时间，如果控制程序简单还可以，要是做专题设计程序变得复杂了，则相当浪费时间，所以读者手上若有 ICE 的话就相当方便了，如果想做更复杂的设计或是功能扩充，我们建议使用下一节所介绍的外接程序 ROM 控制电路。

- 内存使用;
- I/O 解码。

4.4.1 系统总线

任何一块单片机会含有三种系统总线:

- 地址总线;
- 数据总线;
- 控制总线。

当 8051 取用外部程序代码 (EA 接低电位) 或数据内存或是做 I/O 扩充时, 就需用到系统总线来加以控制。数据总线是由 8051 的端口 0, P0.0~P0.7 直接拉出来, 称为 D0~D7, P0 本身利用多工的方式提供数据总线及地址总线, 当 ALE 输出信号为高电位时, 将 P0 送出来的数据锁入正反器中作为地址总线的低位地址线 (A0~A7), 并结合 P2 所送出的高位地址线 (A8~A15) 合成一完整的 16 位地址总线而寻址到外部 64K 字节的内存空间。

8051 控制总线的信号十分简单, 只有 3 个:

- RD
- WR
- PSEN

RD 是用来读取外部数据内存 (RAM) 的控制线, WR 则是用来写数据到 RAM 的控制线, PSEN 是用来存取外部程序内存 (存于 EPROM 中) 的读取控制线。以上所介绍的三种总线将连接到系统的内存或 I/O 的控制线路上。

而单片机 8051 本身需加上系统工作时钟, 在控制板上使用 11.0592MHz 的石英晶振, 此与串行传输接口波特率 (Baud Rate) 时钟设计有关。在此利用 RC 线路产生 RESET 信号来做系统重置用, 另外接有手动 RESET 开关按键。并预留有 RESET IN 接点。可以通过 ROM 模拟器的软件控制, 送进高电位脉冲实现系统自动 RESET 而重新执行程序。

在本电路中可以使用的 I/O 引脚数少了, 但是却可以做很多的功能扩充, I/O 引脚数如果不够用可以外接一块 8255 芯片, 如此一来可增加 24 位的 I/O 控制线, 本电路可以使用的 I/O 引脚如下所示:

8051 引脚功能	引脚
P1.0	1
P1.1	2
P1.2	3
P1.3	4
P1.4	5
P1.5	6
P1.6	7

(续表)

8051 引脚功能	引脚
P1.7	8
P3.0/RXD	10
P3.1/TXD	11
P3.2/TNT0	12
P3.3/INT1	13
P3.4/T0	14
P3.5/T1	15
P3.6/WR	16
P3.7/RD	17

这 16 个 I/O 引脚足够完成一般的接口实验及做简单的专题制作实验。在 P51 控制板上, 这些 I/O 引脚连至圆孔的 DIP IC 座, 可以通过 DIP 信号线或单心线连接到面包板上来做实验。

4.4.2 内存使用

本电路板设计的目的是希望能尽量利用 8051 的寻址空间而使控制程序可以写得很大, 并考虑使用高级的 C 语言来编写控制程序, 所以应考虑外部程序代码与内存空间分开来设计, 图 4-3 是其规划使用情况。根据此图我们来做内存设计及 I/O 解码, 8051 I/O 扩充是以内存应对 (Memory Map) 方式来控制, 也就是把外部 I/O 地址作为内存的一部分来读取数据或写入数据。

外部数据内存线路图请参看图 4-4, SRAM 的选择使用 6116(2K×8)或 6264(8K×8)或 62256 (32K×8) 可以使用跳线来设定, 如下所示:

SRAM 编号	容量	跳线设定
6116	2K BYTE	JP4,5 ON
6264	8K BYTE	JP4,6 ON
62256	64K BYTE	JP3,6 ON

在解码方面为求简化, 采用部分解码来控制, 由 A15 来控制 CS(芯片选择)信号。当 A15 为“0”时, 则选取到外部数据内存 SRAM 中的地址。

程序 EPROM 的设计考虑 4 种状态, 脚位都为 28PIN, 首先看看 4 种 EPROM 的引脚异同点, 列表如下:

EPROM 型号	容量大小	地址线	引脚 27	引脚 1
2764	8K×8	A0~A12	PGM	V _{pp}
27128	16K×8	A0~A13	空接	V _{pp}
27256	32K×8	A0~A14	A14	V _{pp}
27512	64K×8	A0~A15	A14	A15

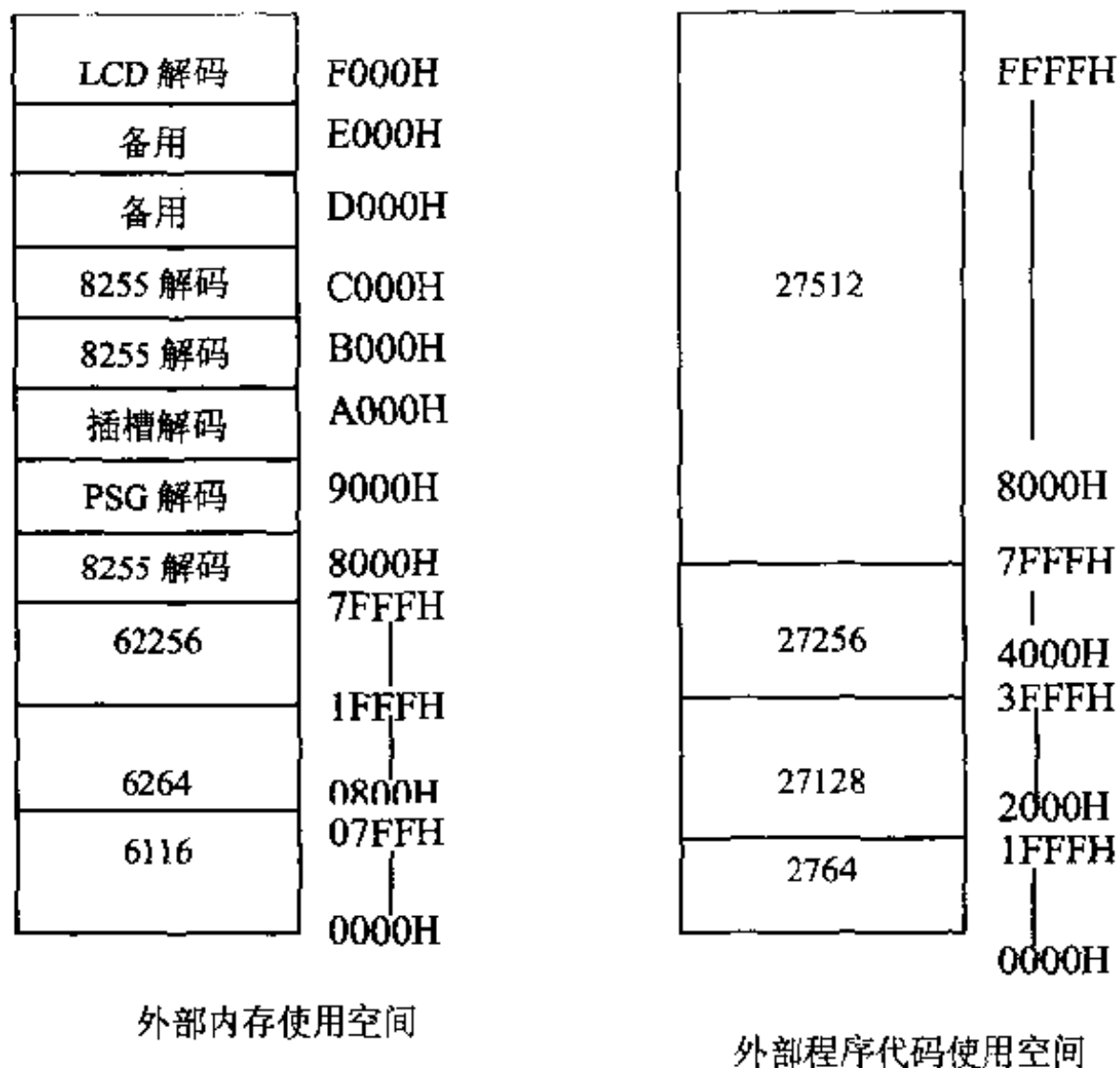


图 4-3 外部程序代码及内存使用

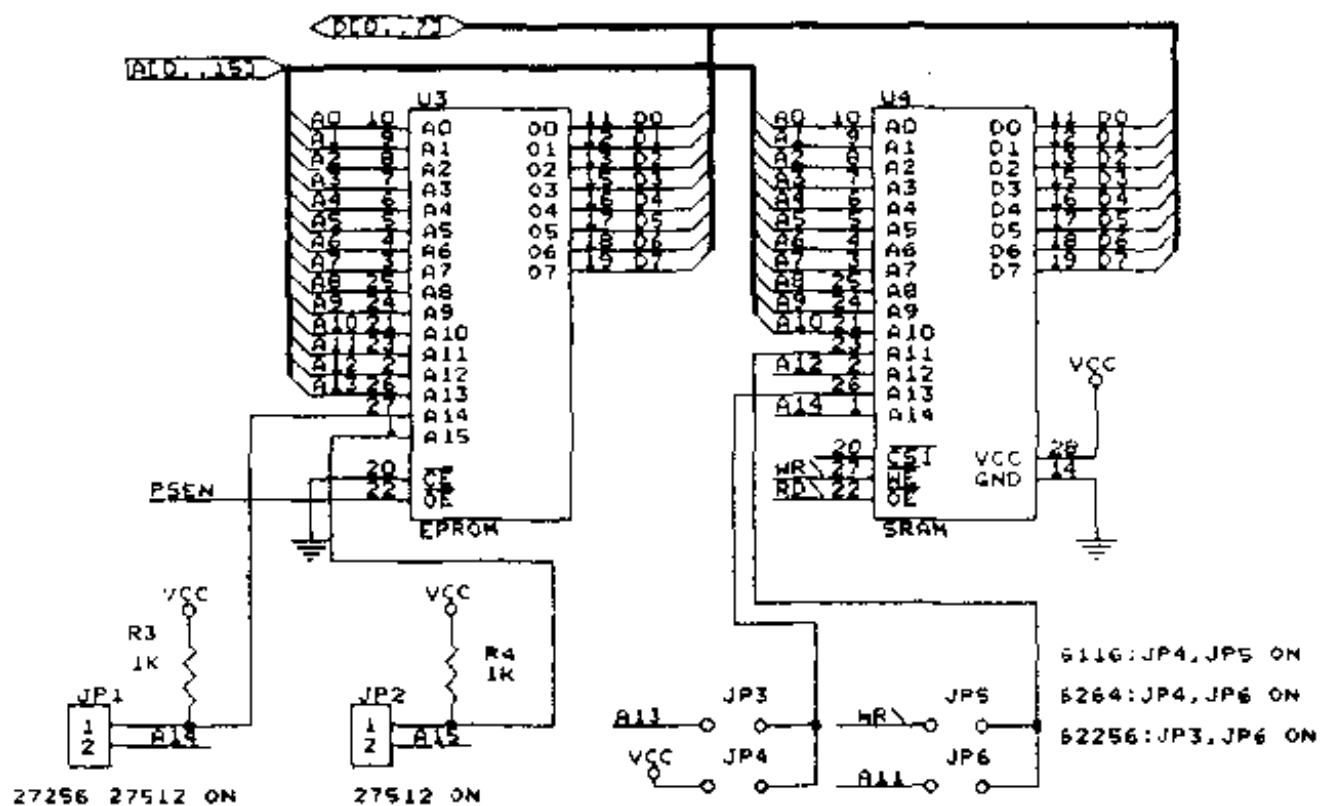


图 4-4 内存电路原理图

其中 PGM 引脚为烧录 EPROM 时接低电位，平时正常操作接高电位，Vpp 引脚为提供烧录的高压脉冲输入，平时也应接高电位，因此可以使用 2 只短路插座来做各种 EPROM 使用的调整。列表如下：

SRAM 编号	容量	跳线设定
2764	8K BYTE	JP1,2 OFF
27128	16K BYTE	JP1,2 OFF
27256	32K BYTE	JP1 ON
27512	64K BYTE	JP1,2 ON

ON 表示接点短路, 否则由提升电阻 (R3、R4) 而拉至高电位。而 EPROM 的 OE (输出使能) 引脚则由 PSEN 信号来控制, CE (芯片使能) 引脚直接接地即可。

4.4.3 I/O 解码

从外部内存空间的使用分布图来看, 可以使用一只 74LS138 (3 对 8 的解码器) 便能完成 I/O 解码的工作。图 4-5 为 I/O 解码的线路, 目前只用到 6 个地址 (部分解码) 其 I/O 解码基地址如下:

- 8000H: 主 8255 芯片使能控制地址
- 9000H: PSG 声效芯片解码
- A000H: 模拟 PC 62 PIN 插槽 I/O 地址解码
- B000H: 扩充第一块 8255 解码
- C000H: 扩充第二块 8255 解码
- D000H: 未使用
- E000H: 未使用
- F000H: LCD 解码

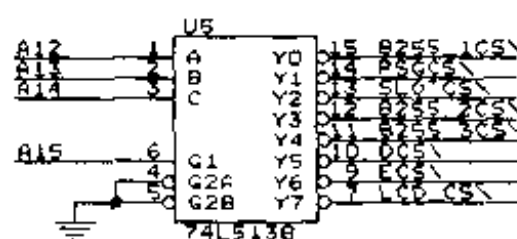


图 4-5 I/O 解码电路

4.5 通信接口

8051 本身提供一组全双工的串行传输接口, 是由 TXD (11 引脚) 来传送串行数据, 而由 RXD (10 引脚) 来接收数据的, 其工作逻辑电位都为 TTL 电位 (0V、5V), 如果要与 PC 做串行数据传输或是连接控制用必需经过 RS232 信号 (+12V、-12V) 电平的转换。目前市面上已有现成的 TTL 到 RS232 电平转换的芯片 IC 编号为 ICL232 或 MAX232, 只要外加四只电容器, 便能完成接口电位转换的工作了。

图 4-6 是设计在电路板上的 RS232 通信接口电路, 通过 MAX232 转换出来的 RS232

串行信号再连至 J10 D9 PIN 插座与外部 PC 连接, 便可建立 RS232 的通信接口。

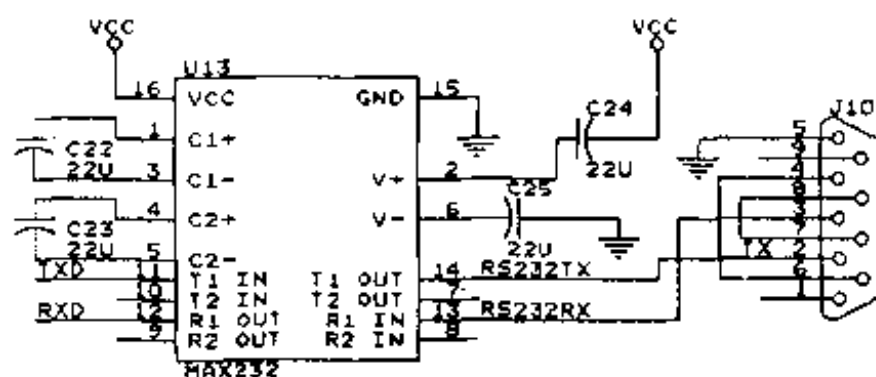


图 4-6 RS232 通信接口电路

4.6 LCD 接口

LCD 在电子产品设计中使用率相当高, 普通的 7 段数码管只能用来显示数字, 若遇到要显示英文字母时, 则一定会选择使用 LCD (液晶显示器), 常见的使用场合有测量仪器及高级电子产品, 但是 LCD 价格不是很便宜。我们在电子市场买到的 LCD, 其背面有控制电路, 上面有专门的 IC 来完成 LCD 的工作, 在自行设计的接口电路中, 只要输入适当的命令代码和要显示的数据, LCD 便会将其字符显示出来, 在程序控制上非常方便。

LCD 可以分为两种类型, 一种是文字模式 LCD, 另一种为绘图模式 LCD。市面上有各种不同厂牌的文字显示类型的 LCD, 仔细的查看一下, 我们可以发现大部分的控制器都是使用同一块芯片来控制的, 编号为 HD44780A, 或是兼容的控制芯片。一般它提供有以下几种显示类型:

- 16 字×1 行
- 20 字×1 行
- 20 字×2 行
- 24 字×2 行
- 40 字×2 行

4.6.1 LCD 特性

1. +5V 电压, 亮度可调整。
2. 内含振荡电路, 系统内含重置电路。
3. 提供各种控制命令, 如清除显示器、字符闪烁、光标闪烁、显示移位等多种功能。
4. 显示用数据 RAM 共有 80 个字节。
5. 字符发生器 ROM 有 160 个 5×7 点阵字型。
6. 字符发生器 RAM 可由使用者自行定义 8 个 5×7 的点阵字型。

4.6.2 引脚说明

图 4-7 为 LCD 引脚图, 由于生产 LCD 厂商众多, 在使用时请注意其电源引脚(PIN 1 及 PIN 2)的接法, 各家厂商可能不同。其引脚功能说明如下:

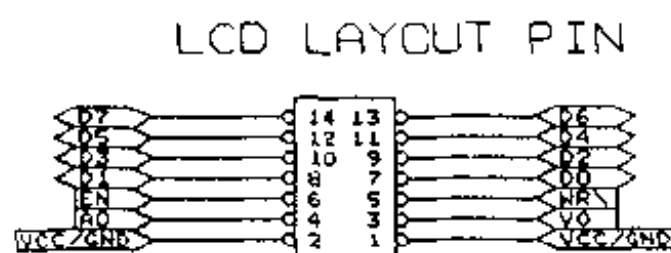


图 4-7 LCD 引脚图

• D0~D7

双向的数据总线, LCD 数据读写方式可以分为 8 位及 4 位 2 种, 以 8 位数据进行读写则 D0~D7 都有效, 若以 4 位方式进行读写, 则只用到 D7~D4。

• RS

寄存器选择控制线, 一般连接地址线 A0。当 RS=0 时, 并且做写入的工作时, 可以写入指令寄存器; 若 RS=0, 且做读取的工作, 可以读取忙碌标志及地址计数器的内容。如果 RS=1 则为读写数据寄存器用。

• R/W

LCD 读写控制线, R/W=0 时, LCD 执行写入的工作, R/W=1 时则做读取的工作。

• EN

使能控制线, 高电位工作。

• VCC

电源正极。

• VO

亮度调整电压输入控制引脚, 当输入 0V 时字符显示最亮。

• GND

电源地端。

看过 LCD 的特性及引脚说明后, 我们来看看 LCD 的控制电路, 图 4-8 是将 LCD 设计在 8051 电路板上的线路图, 使用两个 I/O 解码地址 F000H 及 F001H。F000H 为 LCD 命令写入或状态查询的 I/O 地址, F001H 则为数据读出或写入的地址, 此 2 地址是由 A0 引脚连接 LCD RS 引脚来加以控制, 而 R/W 控制线接 8051 的 WR 引脚, 当 I/O 写入时送出低电位工作的信号。

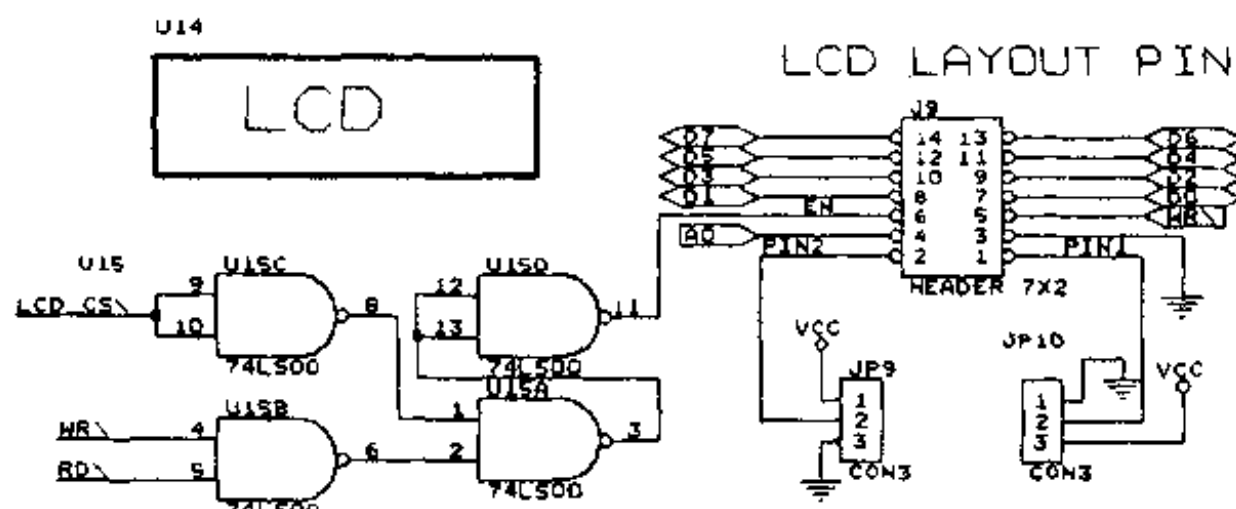


图 4-8 LCD 控制线路图

在 EN 引脚的控制上比较复杂，当程序对 LCD 读取或写入数据时，与非门第 6 PIN 会呈现高电位，配合 LCD 解码信号在与非门第 3 PIN 会产生低电位信号，经反相器后可以正确产生高电平 LCD 使能信号连接 EN 引脚，如果直接将 LCD 解码信号经反相器接往 EN 引脚是无法正确控制 LCD 电路工作的。

4.7 8255 接口

一般控制使用 8751 或是 89C51 做电路板控制，内部程序只有 4K 字节，对特定的程序设计可能够用，但是程序代码一旦超过 4K 时必需以外部程序 ROM 来做程序内存的扩充，最大程序内存可以有 64K 的空间，然而 8051 的端口 0 P0 及端口 2 P2 必需作为系统总线信号使用，让少了 16 位的 I/O 控制线不够用时可以再连接可编程的 I/O 控制芯片。

这类芯片很多，其中最常用的是 8255，只要连接一块 8255，便可以多出 24 位的控制线来，使用起来非常方便。8255 是一块可编程的多功能 I/O 接口控制芯片，不断提供有一般用途的输入输出端口外，还可规划为交互式控制，无怪乎很多单片机在做接口设计时都会考虑到它，本电路板设计也不例外。

图 4-9 是其线路的连接图，8255 的控制信号有 RD、WR、A0、A1、RESET 及 CS。前 5 个信号分别连至 8051 系统相对的控制信号上，A0、A1 选取 8255 的 4 个内部控制寄存器，即端口 A、端口 B、端口 C 及控制字组。而 CS 信号控制整块芯片的工作，其解码地址(部分解码)如下：

1. 8000H~8003H：主 8255 芯片使能控制地址
2. B000H~B003H：扩充第一块 8255 解码
3. C000H~C003H：扩充第二块 8255 解码

当 CS=0 时，数据线 D0~D7 与 8051 数据总线接通，相反当 CS=1 时，数据线则呈现高阻抗与系统数据总线隔离开来。主 8255 芯片端口 A 做 D/A 输出控制，端口 B 负责 7 段数码管数据扫描输出，端口 C 做 PC0~PC3 按键扫描数据输出，PC4~PC7 则做按键扫描返回代码输入，另外两块 8255 则分别做 I/O 功能扩充用，其 3 个 I/O 端口，PA、PB、

PC 则分别连至扩充引脚以方便做外部接口的控制。

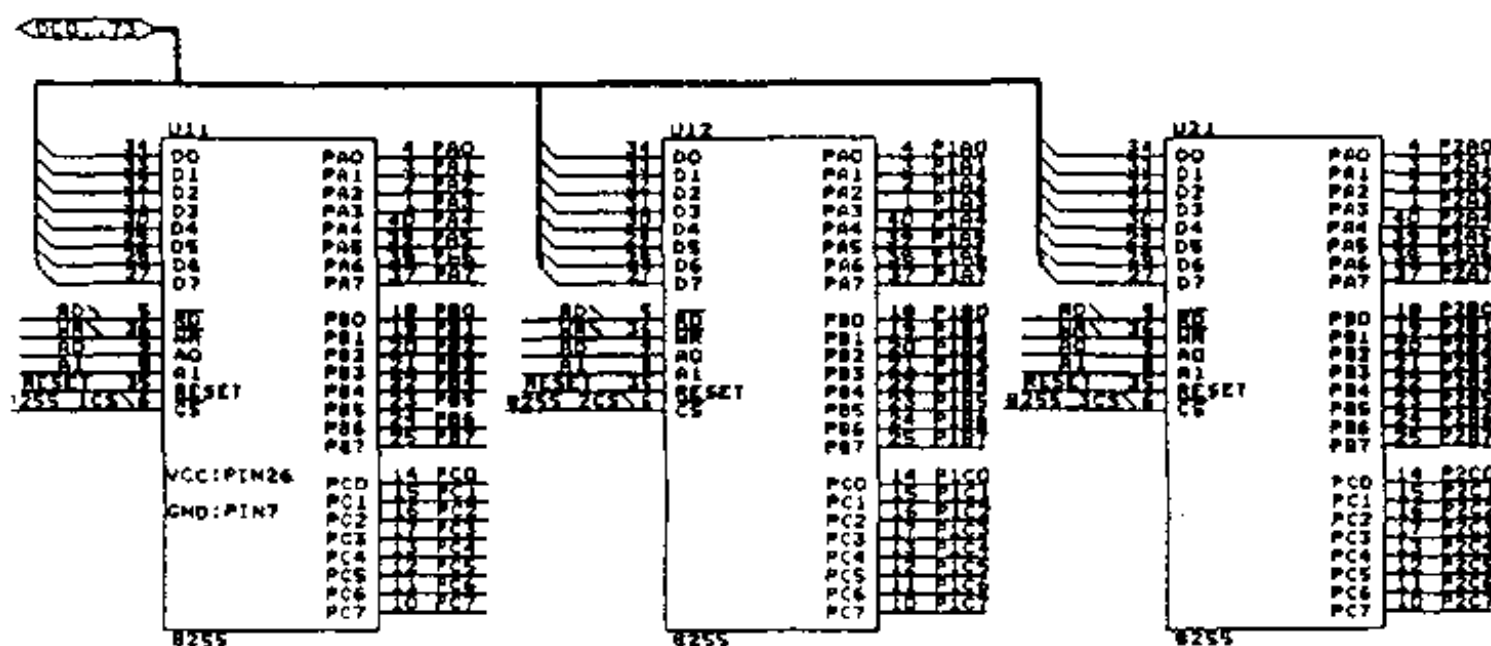


图 4-9 8255 控制线路图

4.8 7 段数码管及按键输入

7 段数码管是常用的数值数据显示元件，对于多位数码管的驱动，最常使用的方法是扫描法，以动态扫描方式来控制较节省硬件成本，一般相关的按键扫描电路也可以一并设计上去。动态扫描方式推动数码管显示数据，可以通过解码器(如 7448)来完成，在电路板上是直接由 8255 输出端口来送出数字数据，直接驱动数码管的优点是可以显示其他字型，如 a、b、c 等字母。

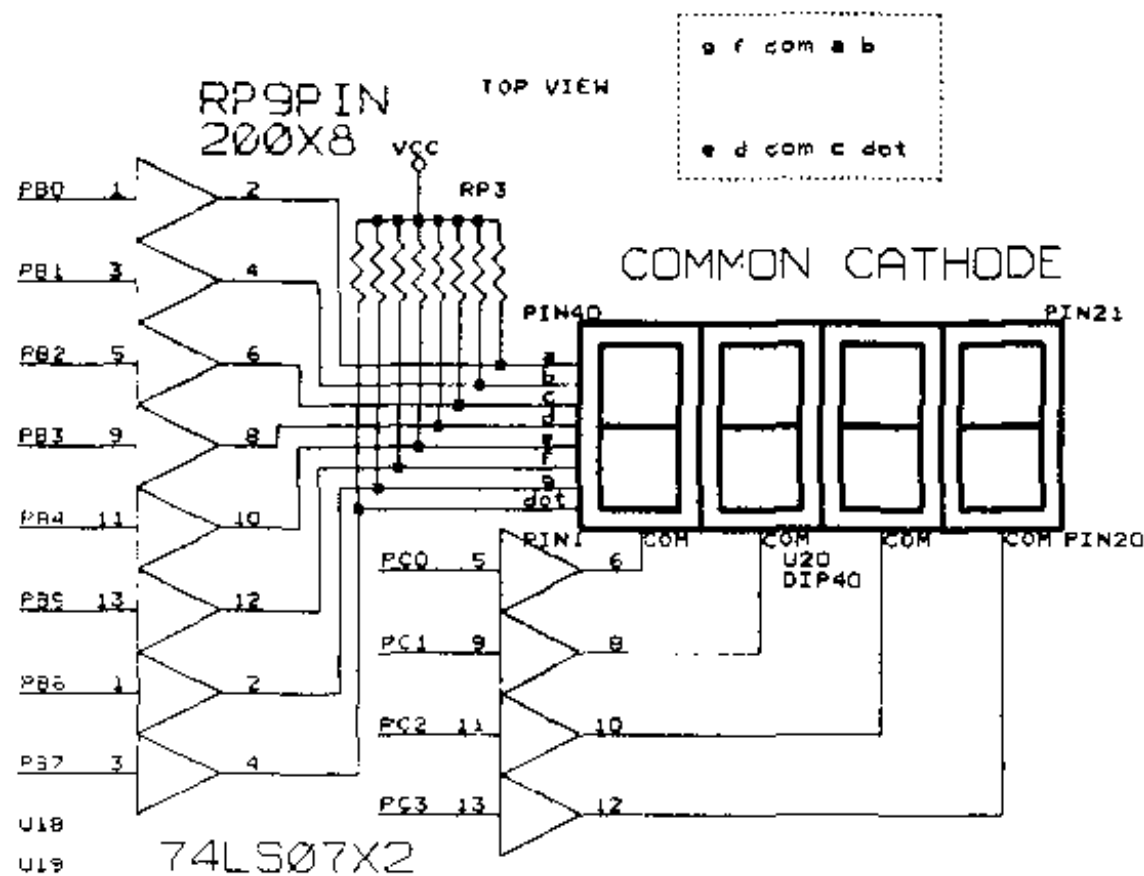


图 4-10 7 段数码管显示控制电路

图 4-10 是电路板上 7 段数码管显示控制电路，此电路是以扫描方式驱动 4 位共阴极

7 段数码管显示数据。由 8255 端口 B 送出数据位，高电位工作，控制信号对应如下：

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
dot	g	f	e	d	c	b	a

由端口 C PC0~PC3 送出扫描代码，低电位工作，任何时候只有 1 个位是低电位输出，即一次点亮一位数做数字显示。PB 送出数据的位先经过 7407 的非反向放大器放大电流信号，例如 PB0 位必需驱动 4 位数码管的“a”引脚，加上提升电阻 200 欧姆电阻做限流电阻，显示器下方的放大器则做位扫描点控制用，每次只点亮一位数字做显示，延迟一小段时间后再点亮下一位数字做显示，因为人们的视觉暂留而感觉 4 位数字同时被显示，一般整体扫描频率只要高于视觉暂留频率(16Hz~20Hz)即可。

在控制电路中，通常需要以按键来控制程序执行流程或是输入数据，如果是按键数不多时可以使用一个按键对应一条输入位线，但是若按键数太多时，通常是以矩阵式扫描法来做按键检测，图 4-11 是 16 个按键扫描的控制电路，使用 8255 端口 C 8 条 I/O 线做 16 个按键的键盘扫描，由 PC0~PC3 送出扫描信号，而由 PC4~PC7 读取按键数据返回代码。

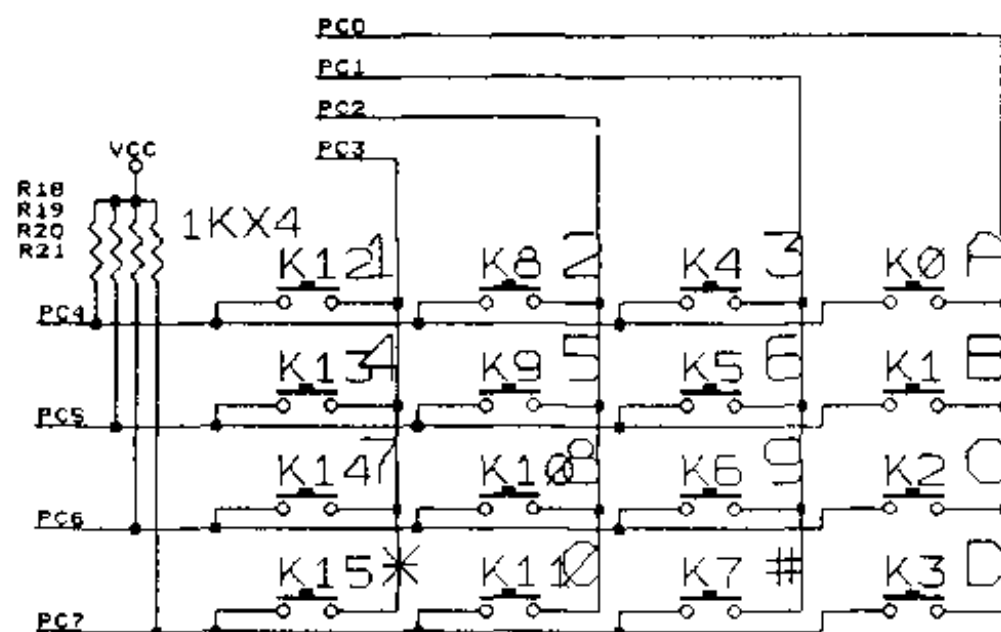


图 4-11 16 个按键扫描控制电路

4.9 D/A 语音接口

为了方便做数字至模拟转换(D/A)的实验，提供简单的语音放音接口，我们在电路板上内建有数字至模拟转换芯片 DAC0800 或是 DAC08。DAC08 其主要规格如下：

- 稳定时间为 100 ns；
- 最大误差为 1LSB；
- 输出电压可调整在 -10V~+18V 之间；
- 具互补的输出电流， I_{out} 及 $I_{out\backslash}$ ；

- 工作电压可从 $\pm 4.5\text{V}$ 到 $\pm 18\text{V}$ 。

图 4-12 是其引脚图，图 4-13 为基本工作线路图。

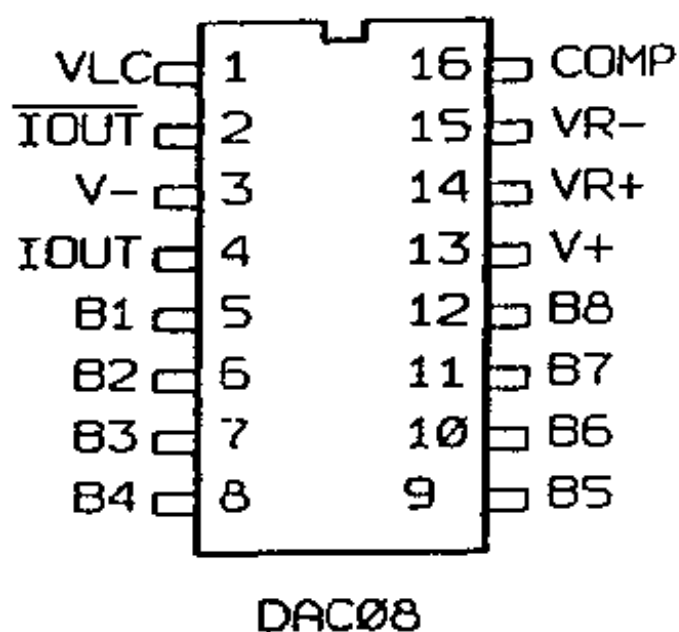


图 4-12 DAC08 引脚图

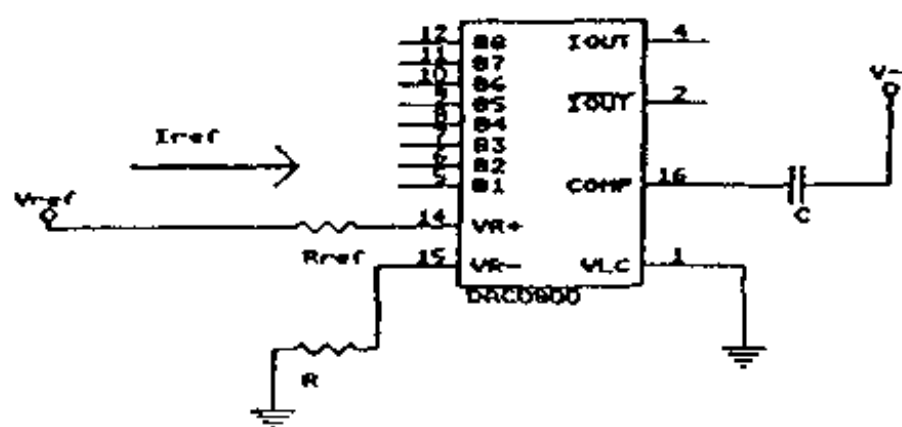


图 4-13 DAC0800 基本工作线路图

4.9.1 引脚说明

- V+

正极输入引脚，电压范围为 $4.5\text{V} \sim 18\text{V}$ 之间，一般取 $+5\text{V}$ 输入。

- V-

负极输入电压范围为 $-18\text{V} \sim -4.5\text{V}$ 之间，一般取 -12V 输入。

- VR+、VR-

参考电压的输入引脚，用来决定满刻度时，电流的大小，其中参考电流 I_{ref} 、参考电压 V_{ref} 及满刻度电流 I_{fs} 的关系如下：

$$0.2\text{mA} \leq I_{\text{ref}} \leq 4\text{mA}$$

$$I_{\text{ref}} = V_{\text{ref}} / R_{\text{ref}}$$

$$I_{\text{fs}} = I_{\text{ref}} \times 255 / 256$$

- VLC

逻辑工作临界值控制引脚，当 DAC0800 输入信号为 TTL 电位时，此引脚接地即可。

- COMP

引脚接上电容至 V- 负极, 可形成频率补偿电路, 以防止高频振荡。

- B1~B8

数字数据输入, 其中 B1 为 MSB (最高位), B8 为 LSB (最低位)。

- Iout、Iout\

模拟互补式电流输出, Iout 与 Iout\ 电流总和为定值。

$$I_{out} + I_{out\backslash} = I_{fs}$$

其中 I_{fs} 为满刻度电流。

4.9.2 DAC0800 接口设计

图 4-14 为以 8255 端口 A 来控制 D/A 的线路。由于 DAC0800 输出电流信号而非电压信号, 在此利用一个运算放大器来做电流到电压的变换, 线路工作分析请参考图 4-15。在电路中, 设定 $V_{ref}=2V$, $R_{ref}=1K$, 可得 $I_{ref}=2mA$, 因此 DAC 输出满刻度电流 I_{fs} 为:

$$I_{fs} = I_{ref} \times 255/256 = 1.992 (mA)$$

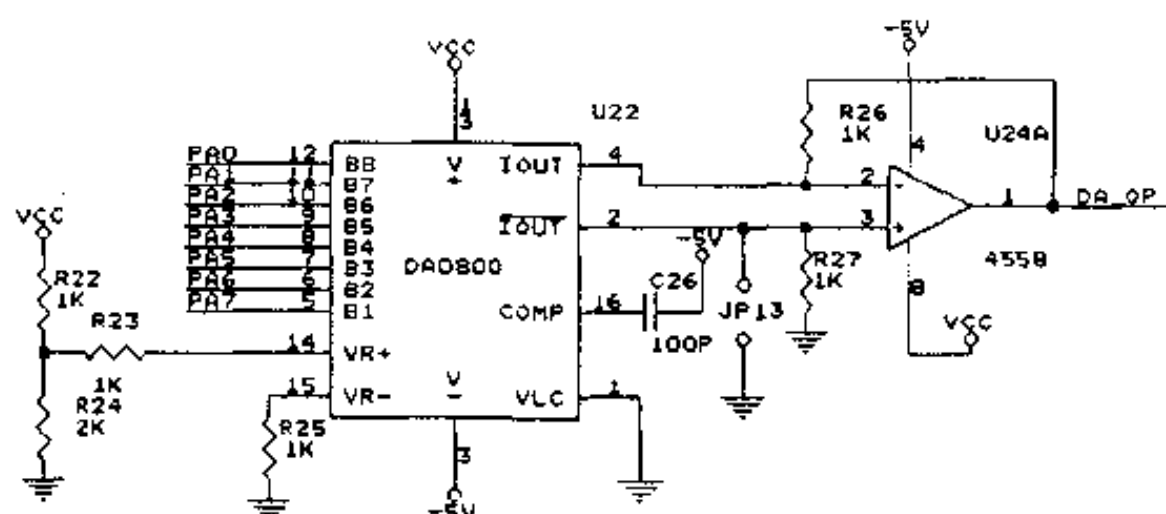


图 4-14 8255 控制 D/A 的电路

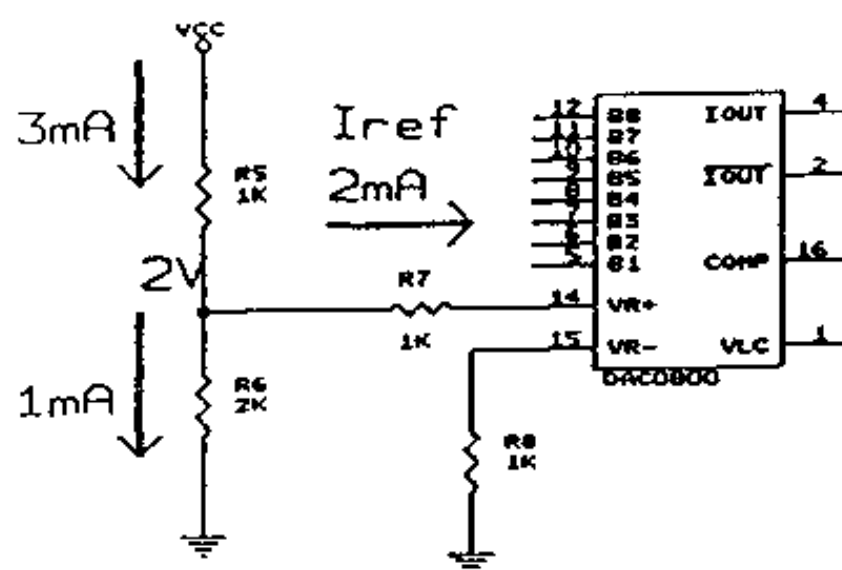


图 4-15 DAC0800 I_{ref} 电流分析

若运算放大器电路中的电阻为 2K，则最大输出电压为：

$I_{fs} \times 2K = 3.98\text{ V}$

至于每个数字信号输出位最高解析电压可以计算如下：

$3.98 \times 1/255 = 15\text{ mV}$

整个数字到模拟转换关系如下表所示：

数字输出值	模拟转换电压
0	0 V
1	15 mV
2	30 mV
.	.
.	.
255	3.98 V

模拟电压输出 V_o ，计算如下：

$V_o = 15\text{ mV} \times (\text{数字输出值})$

只要我们适当的送出数据便可以轻易地产生相对的输出模拟电压，进而输出各种波形。

4.9.3 音频放大电路

图 4-16 为音频放大器电路，用来将微弱的小信号加以放大，滤波而驱动喇叭发出声音。放大器使用 LM386，本身可以提供约 0.5W 的功率，其中可变电阻 VR1 是用于音量调整。放大器 LM386 的电压可以使用 +12V 或 +5V，由 JP12 来选择，选择 +12V 电压可以产生较大的音量输出。此一音频放大电路加上了音频混音电路，由 R9、R10、R11 来分别将以下 3 种音源进行混音：

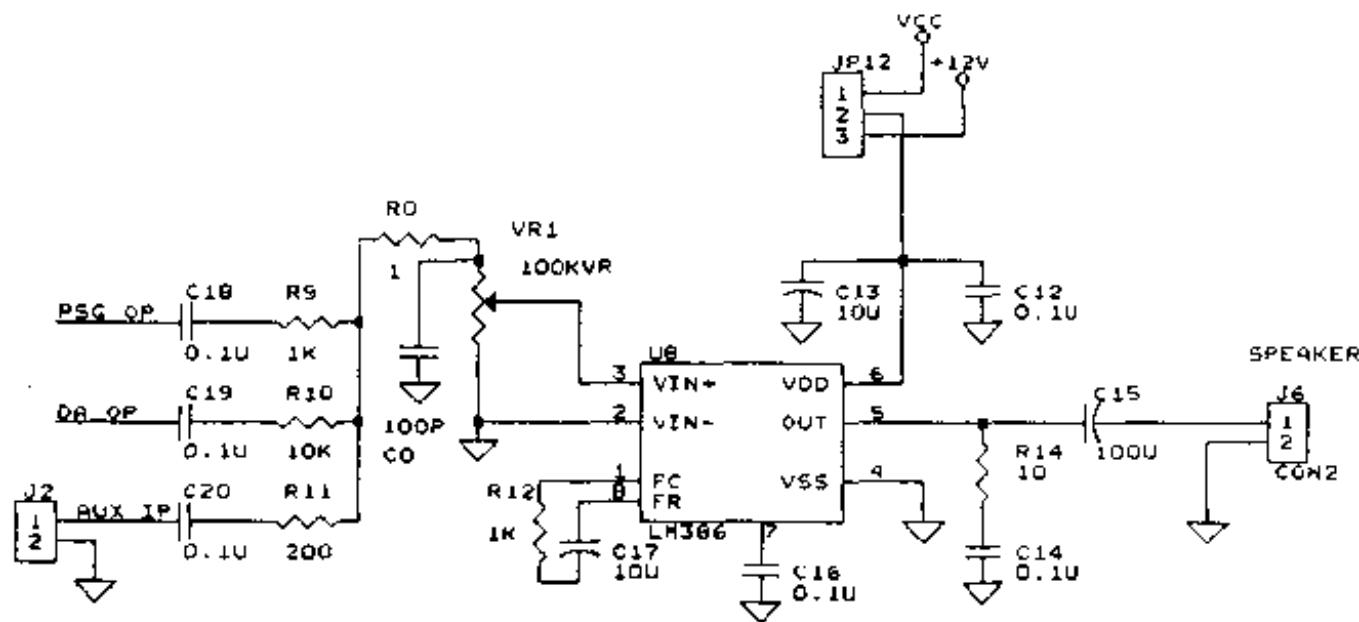


图 4-16 音频放大器电路

1. PSG_OP: 可编程声音发生器声音输出;
2. DA_OP: D/A 声音信号输出;
3. AUX_IP: 外部辅助音源输入。

4.10 声效接口

利用单片机来产生一般的声音, 只要产生适当的频率, 并加以放大便可驱动喇叭发音。只是在产生声音时, CPU 便要负责去做输出的控制, 而无暇去做其他的工作, 像键盘扫描、算术逻辑运算等功能。举个例子来说, 若要产生一个 9 度 C (8372Hz) 的单音时, CPU 就必需每隔 $60\mu\text{s}$ 去触发一次输出端口, 产生方波信号来推动音频放大器。或许我们可以使用定时器产生的中断信号来驱动喇叭, 如此一来, CPU 便可以做其他的工作了。这当然是好方法, 只是当您设计的系统本身功能已相当复杂时, 计时器必需担任其他更重要、更精确的计时工作, 多么希望能有专门的声效处理芯片来完成声音的处理, 那么就可以选择可编程声音发生器 (PSG) 控制芯片来加到单片机系统上。

目前已有多种类型的 PSG 可供使用, 在本节中我们介绍 GI 公司设计的 AY-3-8910 声效产生控制芯片, 此芯片可以很容易与任何的系统总线相连接, 经过适当的程序控制后, 可以产生特殊声效、音乐合成、全音域信号等用途。PSG 的声音输出以 4 位对数比例来调整, 可以提升声音产生的动态响应效果。

一般使用 PSG 来产生各种声音, 有以下三个优点:

1. 程序控制容易

在 CPU 下命令 PSG 产生声效或发出某个频率的声音时, 便可以去做其他的事了, PSG 会自动去处理产生的声音。

2. 声效逼真

动人的音乐往往需要多重频率的声音及多个声道的音乐组成才能完成, 巧妙的设计 PSG 控制程序可以得到令人满意的结果。

3. 接口设计容易

PSG 产生的各种声音可以由程序完全控制。此外 GI 在设计 PSG 时, 考虑到 CPU 的接口控制能力, 也提供有两个独立的 I/O 控制端口, 可供程序员作为一般的输入输出控制。

4.10.1 芯片特性

- 可完全由软件程序控制来产生各种声效。
- 可连接至 8 位或 16 位的微处理机来控制。
- 具有三组独立可编程规划的模拟输出声道。
- 内含二组 8 位一般目的输入输出端口。
- 内含 16 个寄存器与微处理机作接口沟通。

- 只需+5V 电压供电。

4.10.2 内部结构

PSG 内部主要由以下 7 大部分组成：

1. 音调发生器 (Tone Generators)
2. 噪声发生器 (Noise Generators)
3. 混音器 (Mixers)
4. 振幅控制器 (Amplitude Control)
5. 包络发生器 (Envelope Generators)
6. D/A 转换器 (D/A Converters)
7. I/O 端口 (I/O Ports)

1. 音调发生器

共有三个波道 A、B、C 可供选择，各个波道可以分别产生独立、不同频率范围的基本方波输出。

2. 噪声发生器

利用随机数产生各种任意脉冲宽度的方波，加在各个独立的波道输出而产生各种噪声的效果。

3. 混音器

将上述的音调发生器及噪声发生器产生的音源相混合而由波道 A、B、C 送出去。

4. 振幅控制器

PSG 内部具有控制波道振幅大小（即控制声音大小）的寄存器 R8、R9、R10，可以提供固定及可变振幅两种控制模式，其中固定振幅的控制位有 4 位（L0、L1、L2、L3），因此可以作 16 种固定音量的调整，而可变振幅则由包络发生器来控制。

5. 包络发生器

包络发生器所提供的可变振幅声效输出，可以得到振幅调变的波形，即输出声音的频率不变，但是振幅则由包络发生器控制寄存器 R11、R12、R13 所控制，由包络发生器可以制造出许多特殊的声音效果。

6. D/A 转换器

PSG 内部处理的声音通过数字到模拟的转换后得到的模拟信号，直接由三个独立波道 A、B、C 送出。

7. I/O 端口

PSG 提供有二组一般目的的输入输出端口可供使用者做接口的控制用。此两 I/O 端口与声效的产生无关，当声效持续产生时，仍可对其 I/O 端口做存取。

4.10.3 引脚说明

图 4-17 是 AY-3-8910 的引脚图，每个 IC 引脚功能说明如下：

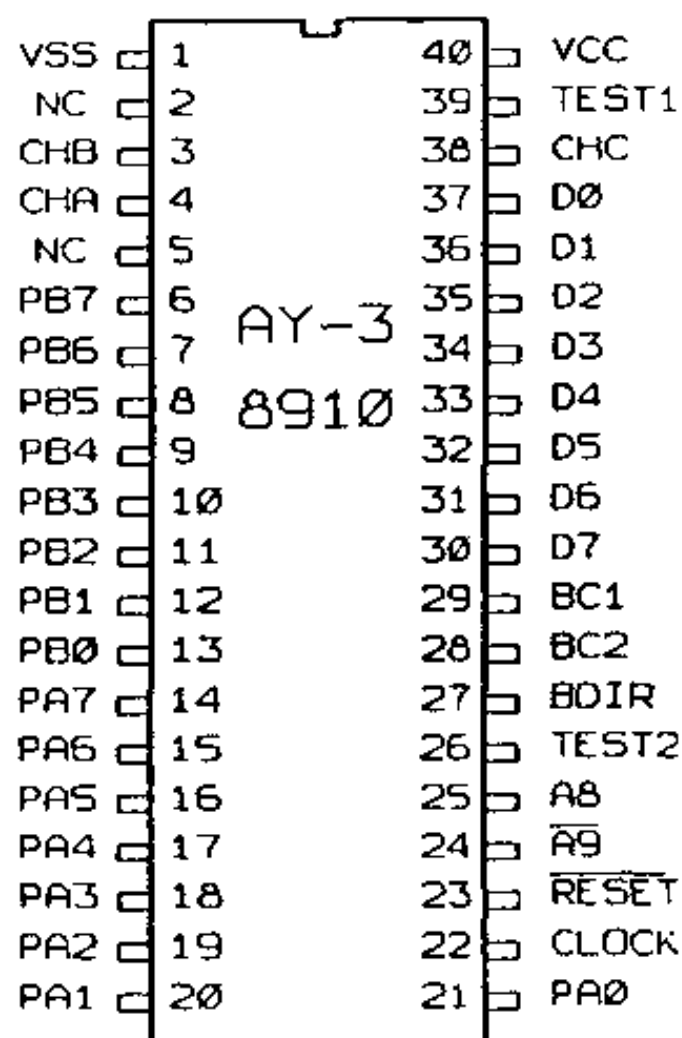


图 4-17 AY-3-8910 引脚

• D0 至 D7 (输入/输出/高阻抗)

8 位双向输入输出数据总线，作为 CPU 与 PSG 间传送数据的沟通管道。当 CPU 选取 PSG 内部寄存器时是使用 D0~D3 共有 4 个位，故可选到其内部的 16 个寄存器。在对某一寄存器写入数据时则是利用 D0~D7 8 个位。

• A8、A9 (输入)

地址解码控制线，A8 高电位工作，A9 则低电位工作。当所连接的 PSG 不只一块时，可以利用来做 PSG 地址使能信号控制引脚。若仅有一块时，A8 接 +5V，A9 接地即可。当 A8 低电位，A9 高电位则总线 D0~D7 呈现高阻抗。

• RESET (输入)

芯片重置引脚，当此引脚接低电位时，将内部各个寄存器值均清 0。通常可以由微处理器送出控制信号来 RESET PSG，用软件来完成芯片重置的目的。

• CLOCK (输入)

PSG 基本工作时钟输入，以提供音调、噪声、包络发生器等内部电路工作所需的基本工作频率，其值不宜过高，一般在 2MHz 以下。

• BDIR、BC1、BC2 (输入)

BDIR (BUS Direction, 总线方向控制)、BC1 (Bus Control 1, 总线控制 1)、BC2 (Bus

Control 2, 总线控制 2), 这 3 个引脚是用于 CPU 与 PSG 间数据的读写控制, 其工作如下所示。

BDIR	BC2	BC1	工作
1	0	0	锁存某一 PSG 地址寄存器 (R0~R15)
1	1	0	CPU 将数据写入 PSG 内部
0	1	1	将 PSG 某一寄存器的内容读回 CPU

• CHA、CHB、CHC (输出)

PSG 三个独立的模拟声道输出, 它可以提供峰对峰的 1V 复合声音输出。

• PA0 至 PA7 (输入/输出)、PB0 至 PB7 (输入/输出)

此为 PSG 提供的两组一般目的输入/输出端口, 因为 I/O 端口的每一个引脚内部均有提升电阻, 因此若规划为输入, 从端口输入数据将读到高电位。此外, 若规划为输出时, 需设为低电位工作。

• TEST1、TEST2

此引脚是 GI 公司用于测试 PSG 芯片的, 正常使用时空接即可。

• VCC

电源正端输入, 接 +5V 电压。

• VSS

电源地端, 接地。

4.10.4 可编程声效发生器接口设计

图 4-18 所示为 PSG AY-3-8910 的控制线路。

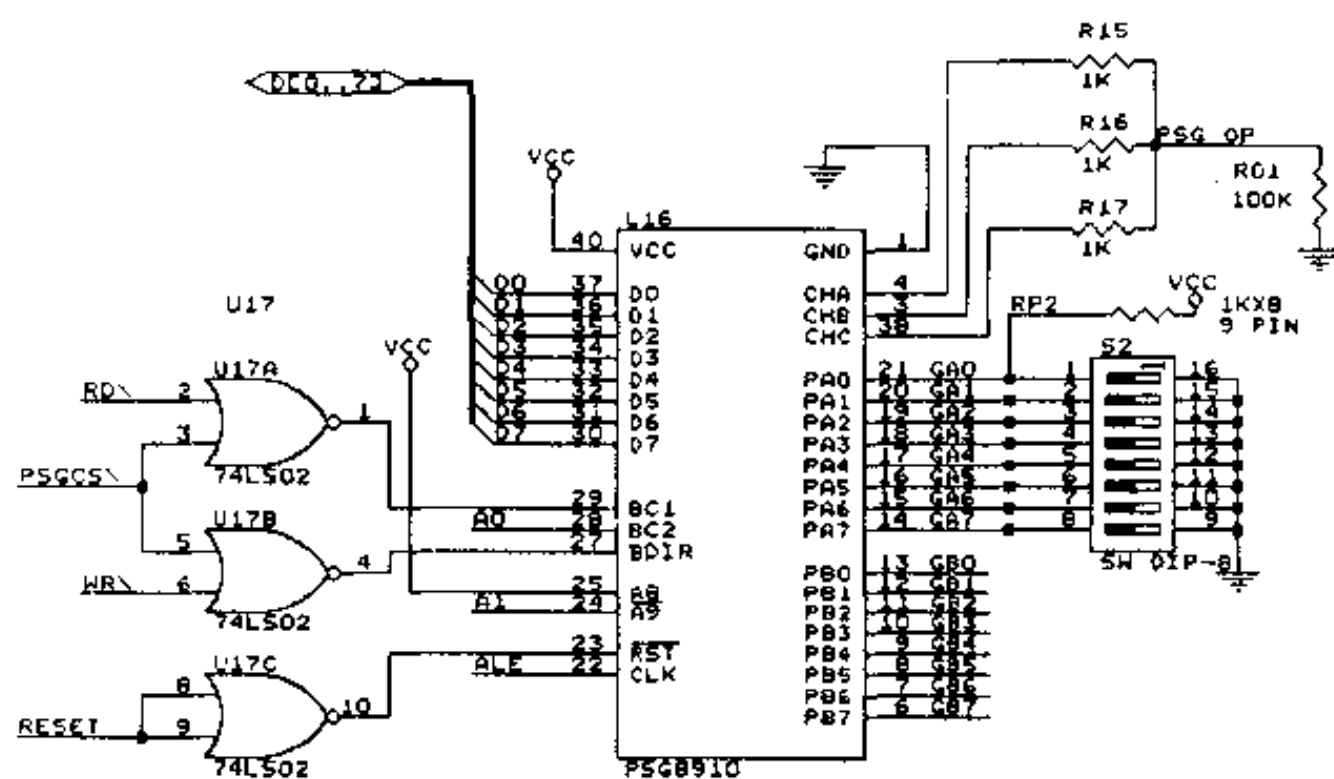


图 4-18 PSG 控制电路设计

前面引脚说明谈及 BDIR、BC1、BC2 的功能,用来完成 CPU 对 PSG 的控制,重新整理如下:

状况	BDIR	BC2	BC1	功能
1	1	0	0	锁存 PSG 某一寄存器
2	1	1	0	CPU 将数据写入 PSG 的某一寄存器内
3	0	1	1	将 PSG 某一寄存器的内容读回 CPU

根据此表可以完成如图 4-19 的设计:

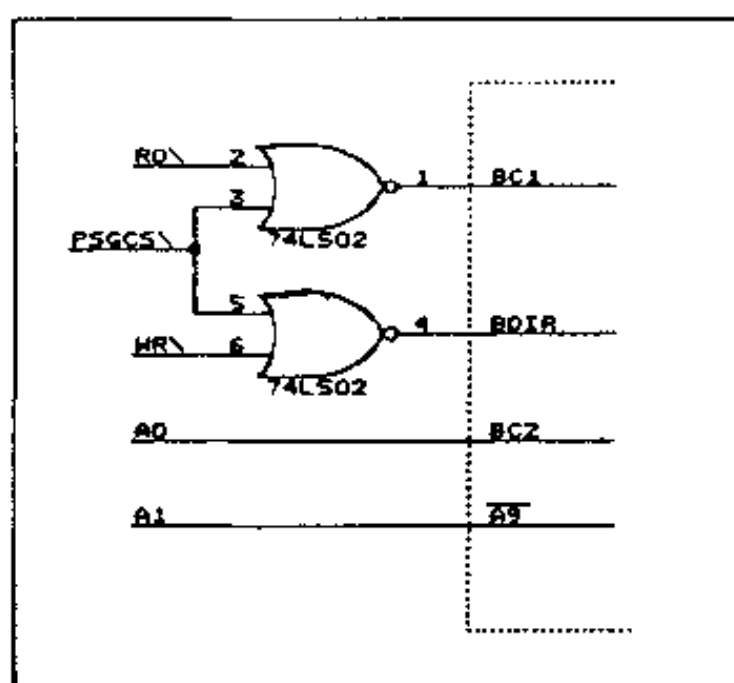


图 4-19 PSG 控制电路分析

其中 PSGCS 的工作地址为 9xxxH, A9 需低电位, 因此 A1 为“0”, 若采用部分解码, 使用两个地址 9000H 及 9001H 便可完成对 PSG 的读写控制了。分析如下:

1. 锁存 PSG 的寄存器

工作地址为 9000H 时, A0=0, BC2=0 (0 表低电位, 1 表高电位)。当 CPU 对 PSG 执行写入操作时, \overline{WR} 工作, 经异或门后, BDIR=1, BC1=0, 此为状态 1 即锁存住 PSG 内部某一寄存器, 至于锁存住那一寄存器则由数据总线 D0~D3 决定。

2. 写入 PSG 寄存器的数据

工作地址为 9001H 时, A0=1, BC2=1。当 CPU 对 PSG 执行写入操作时, \overline{WR} 为低电平, 经异或门, 可得到 BDIR=1, BC1=0 为状态 2, 对 PSG 内部某一寄存器写入数据, 所写入的数据出现在数据总线上。

3. 读取 PSG 内部寄存器的数据

工作地址为 9001H 时, A0=1, BC2=1。当 CPU 对 PSG 执行读取操作时, \overline{RD} 为低电平, 经异或门可得 BDIR=0, BC1=1, 为状态 3, 对 PSG 内部某一寄存器读取数据, 所读取的数据会出现在数据总线上。

而对 A8 的控制接 +5V 电压即可, RESET 信号与系统 RESET 信号相反, 利用另一组异或门作为反相器连接来控制。至于 PSG 的工作时钟, 为求简便, 直接取用 8051 的 ALE

信号, 工作频率约为 1.84MHz (11059200/6)。

PSG 产生的声音可由 3 个独立输出声道 A、B、C 送出, 通过 3 只电阻进行混音而送至音频放大器, 做信号放大而驱动喇叭发出声音, 请参考图 4-16 音频放大器电路。

至于两个 I/O 端口 PA 及 PB 则可以做一般的 I/O 用途使用, 其中 PA 设定为输入, 可以接至 8 组 DIP 开关, 作为设定输入用, 图 4-20 为 DIP 开关设定输入连接图。DIP 开关接至 I/O 端口作为输入时, 通过端口 A 连接排阻 RP2 的 9 PIN 提升电阻并接至 +5V 电源, 平时读取为高电位, 若有某一位扳于 ON 时, 则该位被设定为 0, 因此读取输入端口, 则知道每一位 ON/OFF 的状态, 而做各种状态的初值设定。

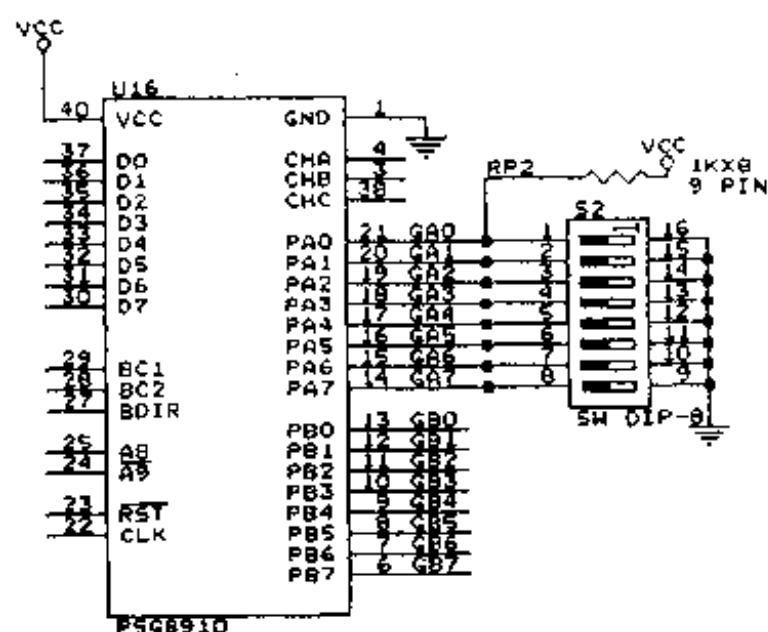


图 4-20 DIP 开关设定输入

4.11 LED 显示及蜂鸣器控制

在电路板上 8051 端口 1 位 7 (P1.7) 接有一 LED 指示灯, 我们称为工作指示 LED。工作指示 LED 可以用来表示线上某一位的高低电位状态, 或是特定的状态, 例如程序开始执行时, 可以令其闪动两下。若是程序执行至状态一则 LED 闪动一下, 程序执行至状态二则 LED 闪动两下, 进入死循环中, 则 LED 持续闪烁。因此使用 LED 可以用来指示系统目前的工作状态, 是一种最简单的程序除错指示。

此外在 8051 端口 1 P1 通过电阻排 RP1 的 9 PIN 提升电阻排, 连接一只 8 个条状 LED 灯可以做“走马灯”式的展示, 图 4-21 为其控制电路。当相对的位送出低电位时, LED 顺向导通而发亮, 输出高电位则使 LED 熄灭。

图 4-22 为蜂鸣器控制电路。在电路板上 8051 端口 1 位 0 (P1.0) 是设计做蜂鸣器的驱动位, 持续送出工作脉冲可以推动喇叭发出“嘀”的声响, 当工作频率越高时, 声音越清脆, 工作频率低时, 则声音较低沉, 同理由喇叭的“嘀”声也可以用来指示系统的工作状态, 是另外一种简单的指示方法。

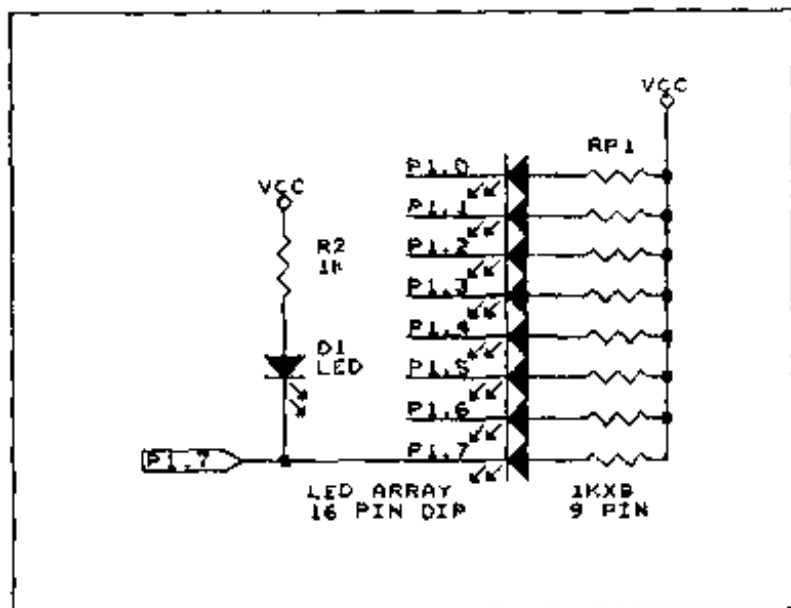


图 4-21 LED 显示控制电路

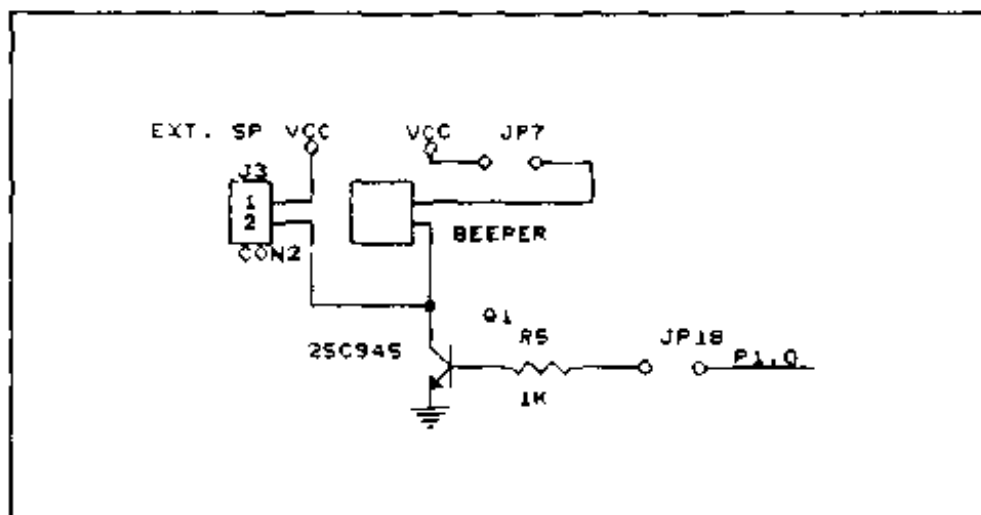


图 4-22 蜂鸣器控制电路

在电路图中标名为“BEEPER”为蜂鸣器的接点，而脚座 J3 也可以接入一般的小型喇叭，短路座 JP18 则是用于蜂鸣器启用控制。

4.12 电源控制电路

图 4-23 是电路板上的电源控制电路, 共分为以下几部分:

- +5V 稳压;
- -5V 电压转换。

+5V 电压是为电路板上主要芯片 (IC) 提供的, 由外部的 9V 直流电源调整器通过引脚 J4 输入, 并以 7805 IC 做稳压, 本身需加上散热片进行散热, 否则当温度持续升高时, 芯片会烧毁, 如果使用者有很多外部的扩充控制电路, 请由 J5 引脚自行加入外部 +5V 电压, 当然此时 9V 直流电源调整器要先移开。

—5V 电压主要使用在 DA08 数字至模拟转换上, 为了简化电路设计, 使用 U23 7660 进行 +5V 至 -5V 电压转换, 可以得到 -5V 电压输出。

第5章 8051 多功能控制板制作及测试

8051 多功能控制板(P51_PCB)已将一般专题制作上常见的基本接口都设计在上面,使用者可以根据需要而自行加以扩充,如果只是想使用部分电路功能,建议采用空的电路板来自行组装,可以体会自己装的乐趣,一些元件在一般电子商店均可以买到,请参考附录中所附的使用元件来逐一组装。读者若买了空电路板打算自己组装,可以将本书配套的磁盘(购买联系地址:北京海淀路 82 号科海培训中心,电话:(010)62562449)中的 8051 单板测试程序代码 TP51.ROM 烧录于一片 EPROM 2764 中,以便做单板功能验证用。

在准备好了相关元件后,请按照本章所述步骤来组装,可以避免不必要的麻烦。我们采用的方法是依阶段性来进行组装,在焊接了某一部分元件后即先行测试,可以方便除错。如果不能工作一定是电路虚焊、漏焊或焊接短路,只要元件没接错,按照以下的步骤组装,应该很容易组装一块 8051 单片机控制板。共分为以下几个装配阶段,我们在下面几节中分别加以说明。

1. 单片机基本工作验证;
2. 测试 RS232 接口;
3. 测试 8255 接口;
4. 共阴极 7 段数码管测试;
5. 测试按键输入;
6. 测试蜂鸣器;
7. 测试 8 只 LED;
8. 声效测试;
9. 测试 D/A 接口;
10. 测试 8255 I/O 扩充接口;
11. 测试 LCD 接口;
12. 加装电源控制。

5.1 8051 多功能控制板快速安装及测试

读者如果购买 8051 多功能控制板成品的话,在控制板上已经装上单板测试程序 ROM 了,使用者可以直接进行测试,以下说明其快速安装及测试步骤。

1. 连接喇叭接头至 J6。
2. 连接电源线接头至 8051 单板上,并打开电源,此时电源 LED 灯亮,7 段数码管显示“0123”,工作 LED 闪烁,LCD 屏幕显示如下:

P51 test fn V1.0

12-> test port...

- 按键“1”：测试 U12 8255 芯片。
- 按键“2”：测试 U21 8255 芯片。
- 按键“3”：测试 U22 DA08 芯片，分别输出 -1.4V、0V 及 2V，可以用万用表测量 J24 DA/OP 点。
- 按键“4”：由 DA08 芯片产生方波信号，可以听见嘀一声，调整 VR1 可以控制音量大小。
- 按键“5”：蜂鸣器发出声响。
- 按键“6”：8 只 LED 闪烁。
- 按键“7”：对 SRAM 填入测试样本。
- 按键“8”：读取 SRAM 测试样本，看看是否正确，若错误则工作 LED 会闪烁着。先执行按键“7”，再执行按键“8”。
- 按键“9”：由 U16 PSG AY-3-8910 发出 10 次声响，调整 VR1 可以控制音量大小。

3. 详细的使用说明可参考配套磁盘中的说明文件 P51.DOC 或是本书的各节说明。

5.2 单片机基本工作验证

读者如果选择 DIY 则从本节起，开始介绍如何自己组装一块 8051 控制板。为了方便除错及元件重复使用，建议所有的 IC 全部焊上 IC 引脚。在此阶段是完成 8051 能工作的最基本电路，请参阅电路图会很清楚的。

步骤 1：焊上如下的 IC 引脚：

U1：40 PIN IC 引脚(8051 单片机)

U2：20 PIN IC 引脚(74LS373)

U3：28 PIN IC 引脚(EPROM 2764 烧录有测试程序 TP51.ROM)

U4：28 PIN IC 引脚(SRAM)

步骤 2：焊上全部为 1K Ω ，1/4W 的 R1、R2、R3、R4、R7 电阻。

步骤 3：焊上电源指示 LED D4，注意极性，长脚接“+”端。同时焊上工作指示 LED D1。

步骤 4：焊上 +5V 电源接头 J5，注意 +5V 在右端，接地 (GND) 在左端 (电路板上已标示出来)。

步骤 5：接上 RESET 按钮开关 S1 及电容 C1 10 μ F，注意极性，长脚接“+”端。

步骤 6：接上石英晶振 11.0592MHz，若没有，可以采用 12MHz 的晶振，只是在 RS232 功能验证时会不正确，因为传输频率变了。同时焊上 C2、C3、20P 的 2 只小电容。

步骤 7: 焊上 JP0 的 3PIN 插针, 同时将上头两端插上短路器使其短路 (电路板上标示为 EA\), 由 EPROM 来提供 8051 程序代码。

步骤 8: 焊上短路接点 2 PIN 插针 JP1、JP2、JP3、JP4、JP5、JP6。

步骤 9: 焊上 JP17 的 3PIN 插针, 并将短路器放于 VCC 端为 SRAM 提供电源。

步骤 10: 焊上 JP11 的 3PIN 插针, 并将短路器放于 A15 端为 SRAM 提供解码信号。

步骤 11: 插上 IC 元件, 注意, 看是否有引脚插歪了, 或未插入 IC 座中, IC 插入时请注意方向, 缺口在左方, 请勿反插而烧坏 IC。有关 SRAM 的选择如下:

SRAM 型号	JP3	JP4	JP5	JP6
6116	OFF	ON	ON	OFF
6264	OFF	ON	OFF	ON
62256	ON	OFF	OFF	ON

标示 ON 者是将短路器放入短路接点, 如选择使用 62256 则将 JP3、JP6 置为 ON。

步骤 12: 此为本阶段的最后一步骤, 也是最重要的步骤, 请正确检查您的电源供应器是否为+5V 电压, 其极性的正负连接是否正确后, 再加入单板上的接头 J5。

注意: 请再三验证接头 J5 的极性再接入+5V 电源。

若一切顺利, 当电源接通时, 则可看到电源 LED 亮, 而工作 LED 指示灯闪动, 完成了本阶段的测试, 好的开始是成功的一半。如果不能工作, 则不外乎是以下几点问题, 请自行检查。

- 电路虚焊;
- 电路接点漏焊;
- 焊接接点短路;
- 元件插错, 注意有极性的元件;
- IC 引脚未完全插入 IC 座中;
- 元件不良。

5.3 测试 RS232 接口

单片机控制板若可以与 PC 建立连接后, 便可以快速方便地进行软件及硬件的除错, 本节说明 8051 单板的 RS232 接口电路连接。

步骤 1: 焊上 U13 16PIN IC 座。

步骤 2: 焊上 C22~C25 4 只 22uF 的电解质极性电容, 请按照电路板上的标示插上电容, 长脚为“+”。

步骤 3: 焊上 J10 D9 型公接头 (其接点为凸出者)。

步骤 4: 准备一条两边母的 D9 PIN RS232 电缆线, 连接 8051 单板 J10 接头至 PC 的串行通信端口 2(COM2), 必要时连接 25 PIN 转接头。也可以使用串行通信端口 1(COM1)。

步骤 5: 执行通信程序 SE.EXE 或 SEV.EXE。

SEV 0 3(使用 PC 通信端口 COM1);

SE(使用 PC 通信端口 COM2);

其中传输协议如下:

<9600,N,8,1>

9600 : 波特率为 9600 bps;

N : 没有校验位;

8 : 8 个数据位;

1 : 1 个停止位。

在接入电源到 8051 单板后, 可以看到由 8051 送出消息至 PC 显示如下:

```
-----
SE.EXE  PC  RS232  COM2  <9600 N 8 1>
```

```
-----
P51 test  fn V1.1      Copyright VICTOR uP LAB.
                Inc. 1995-1996
```

```
1-> test  port1    2-> test  port2    3-> test  D/A
4-> da_beep          5-> beep          6-> test  leds
7-> fill   pat.     8-> test  pat.     9-> test  PSG
0-> test   key16    a-> test  DIP  SW
```

则可以做 8051 与 PC 间双向数据传输的简单测试。

5.4 测试 8255 接口

8051 单板的基本 I/O 功能是由 8255 U11 进行控制的, 安装如下:

步骤 1: 焊上 IC 座:

U5: 16 PIN IC 座(74LS138 I/O 解码);

U11: 40 PIN IC 座(8255)。

步骤 2: 装上 IC。

步骤 3: 加上 +5V 电源, 此时 PC 上执行通信程序, 则除工作画面出现外, 还出现一行测试消息:

TEST main 8255 ... 8255 test OK !

则表示 8255 测试正常。

5.5 共阴极 7 段数码管测试

此阶段是测试 4 只共阴极 7 段数码管是否显示正常, 此时应该焊上哪些 IC 呢? 请参看电路图。读者应该有能力找到才对, 一方面查看电路, 一边进行焊接可以很深切地体会到单片机工作的原理, 是由 8255 来控制 7 段数码管, 相关元件安装如下:

步骤 1: 焊上 IC 座:

U18 U19: 16 PIN IC 座(74LS07);

U20: 40 PIN IC 座(插入 4 只共阴极的 7 段数码管)。

步骤 2: 装上 IC 及 4 只共阴极的 7 段数码管。

步骤 3: 装上电阻排 RP3 ($200\Omega \times 8$)。

步骤 4: 加上 +5V 电源, 4 只 7 段数码管显示 “1234”, 则表示数码管测试正常。

5.6 测试按键输入

此阶段是测试 16 个测试按键输入, 相关元件安装如下:

步骤 1: 焊上 R18~R21 4 个电阻 $1K\Omega$ 。

步骤 2: 连接 16 个按键。

步骤 3: 加上 +5V 电源, 此时 PC 上执行通信程序, 则工作画面出现, 按下 PC 键盘上的 “0” 键, 或是单板上的 “0” 键, 都可以测试按键输入。

步骤 4: 逐一按下每个按键, 则会显示出所按下的按键, 若按下按键 “0” 则结束测试。

5.7 测试蜂鸣器

步骤 1: 焊上蜂鸣器至 BZ 处。

步骤 2: 焊上短路接点 2 PIN 插针 JP7、JP18, 并加上短路器。

步骤 3: 焊接 R5 $1K\Omega$ 电阻。

步骤 4: 焊接 Q1 2SC945 电晶体。

此时接通电源在看见 LED 闪烁后, 随即听见蜂鸣器 “嘀” 一声, 则测试正常。而每次按键后蜂鸣器也会 “嘀” 一声, 若按下按键 “5”, 则蜂鸣器会 “嘀” 4 声。

5.8 测试 8 只 LED

步骤 1: 装上电阻排 RP1 ($1K\Omega \times 8$)。

步骤 2: 焊上 16 PIN IC 座在 8 只 LED 引脚处。

步骤 3: 插入 20 PIN LED 引脚, 其中 4 PIN 未使用空接。

此时接入电源, 在看见 LED 闪烁后, 若按下键“6”, 则 8 只 LED 会闪烁。

5.9 声效测试

步骤 1: 焊上 IC 座:

U17: 14 PIN IC 座(74LS02);

U16: 40 PIN IC 座(PSG 8910);

U8 : 8 PIN IC 座(LM386)。

步骤 2: 焊上 100K 的可变电阻 VR1。

步骤 3: 接上电阻(请参考电路图)

R15~R17 1K Ω 、R01 100 K Ω 、R0 1 Ω 、R14 10 Ω , R9 1K Ω 。

步骤 4: 接上电容(请参考电路图)

C18、C14, 0.1 μ F 的纸介电容, C15 100 μ F 电解质极性电容, C0 100P 的陶瓷电容

步骤 5: 焊上 2 PIN 插针 JP8, 并加上短路器。

步骤 6: 焊上 3 PIN 插针 JP12, 并加上短路器于 5V 端。

步骤 7: 焊上 2 PIN 引脚至 J6。

步骤 8: 由 J6 的引脚连至 8 Ω 小型喇叭。

步骤 9: 装上 IC。

接通电源, 按下按键“9”, 由 PSG AY-3-8910 产生声效 10 声, 调整 VR1 可以控制音量大小。

5.10 测试 D/A 接口

步骤 1: 焊上 IC 座:

U22: 16 PIN IC 座(DAC0800);

U23: 8 PIN IC 座(7660);

U24: 8 PIN IC 座(4558)。

步骤 2: 接上电阻(请参考电路图);

R22、R23、R25~R27 1K Ω , R24 2K Ω , R10 10K Ω 。

步骤 3: 接上电容(请参考电路图)

C28、C14、C19 0.1 μ F 的纸介电容, C27、C29、C30 100 μ F 电解质极性电容, C26 100P 的陶瓷电容。

步骤 4: 由 J6 的引脚连至 8 Ω 小型喇叭。

步骤 5: 装上 IC。

接通电源, 按下按键“4”, 则通过 DAC0800 产生并发出响声, 调整 VR1 可以控制音量大小。

5.11 测试 8255 I/O 扩充接口

步骤 1: 焊上 IC 座:

U12 U21: 40 PIN IC 座(8255)。

步骤 2: 装上 IC。

接通电源, 此时 PC 上执行通信程序, 则出现工作画面, 按下按键“1”, 测试 U12 8255 芯片。按下按键“2”, 测试 U21 8255 芯片。

5.12 测试 LCD 接口

市售的 LCD 有两种引脚, 区别在于电源 VCC 及 GND 的控制, 在 8051 多功能控制板 P51_PCB 上利用短路器 JP9 及 JP10 来调整。使用者如果自行购买 LCD, 别忘了要索取 LCD 数据, 查看 LCD 的电源引脚以方便实验。安装步骤如下:

步骤 1: 焊上 IC 座:

U15: 14 PIN IC 座(74LS00)。

步骤 2: 装上 J9 14 PIN 插座。

步骤 3: 取出 14 PIN 插针焊在 LCD 上。

步骤 4: 焊上 3 PIN 插针 JP9 JP10, 并加上短路器于 5V 端及接地端(请查看电路), 并以数字电表量一下, 以确认电源连接是否正确。

接通电源, 则 LCD 显示如下:

P51 TEST FN V1.1

12-> test port...

5.13 加装电源控制

前面控制板的电源都是使用外部+5V 的电源, 在本节说明如何安装控制板上的电源, 只要使用市售的+9V 300 mA 电源调整器输入到控制板上, 便可以产生+5V 电源。相关元件安装如下:

- 步骤 1: 接上 J4 电源接头, 其中内部为正(+9V), 外部为接地。
- 步骤 2: 焊上 D2 D3 二极管 IN4001。
- 步骤 3: 接上电容(请参考电路图)C7、C4 100 μ F(25V)电解质极性电容。
- 步骤 4: 接上稳压 IC 7805 并装上散热片。
- 步骤 5: 取出+9V 350 mA 电源调整器连接到 J4 电源接头, 则控制板应该正常工作。

第6章 8255 接口控制

8255 是一块可编程的 I/O 接口控制芯片，在很多电子产品单片机硬件外围设备及工业自动控制板上均可以发现到它，主要是用来支持 Intel 单片机系统的外围控制芯片，由于其 I/O 工作具有一般标准的工作时间，可以用于其他各种型号的微处理器上如 6502、Z80、8088，甚至单片机 8048、8051，因此我们称它为通用型多功能的可编程 I/O 接口控制芯片。

在本章中除了对 8255 内部功能做介绍外，还将说明如何用 8051 来控制 8255，以及 8255 接口电路测试程序的说明。

6.1 8255 简介

图 6-1 是 8255 的内部结构图，8255 有 3 个可编程控制的输入输出端口（每个端口提供 8 个 I/O 位），共可提供 24 个 I/O 的控制引脚。而此 3 个 I/O 端口的输入或输出分别由 A 组及 B 组的内部控制器加以设定。其中 A 组由端口 A PA0~PA7 及端口 C 的上半部 PC4~PC7 组成，B 组则由端口 B PB0~PB7 及端口 C 的下半部 PC0~PC3 所组成。芯片控制的方式是通过双向的总线与单片机系统连接，通过写入控制来设定 8255 的工作方式。

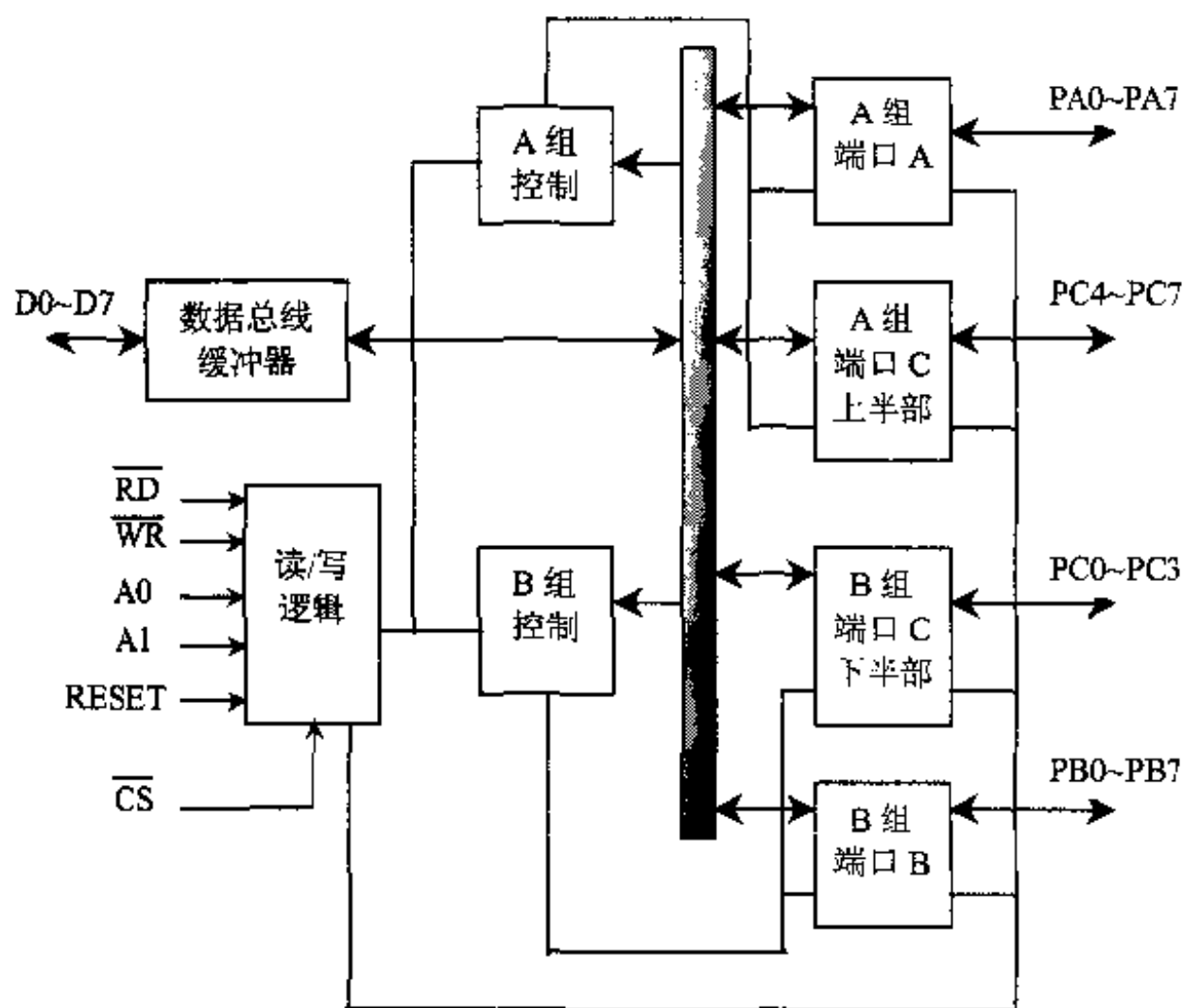


图 6-1 8255 内部流程图

6.2 8255 引脚说明

图 6-2 为 8255 的引脚结构图，我们将其电气特性分别说明如下：

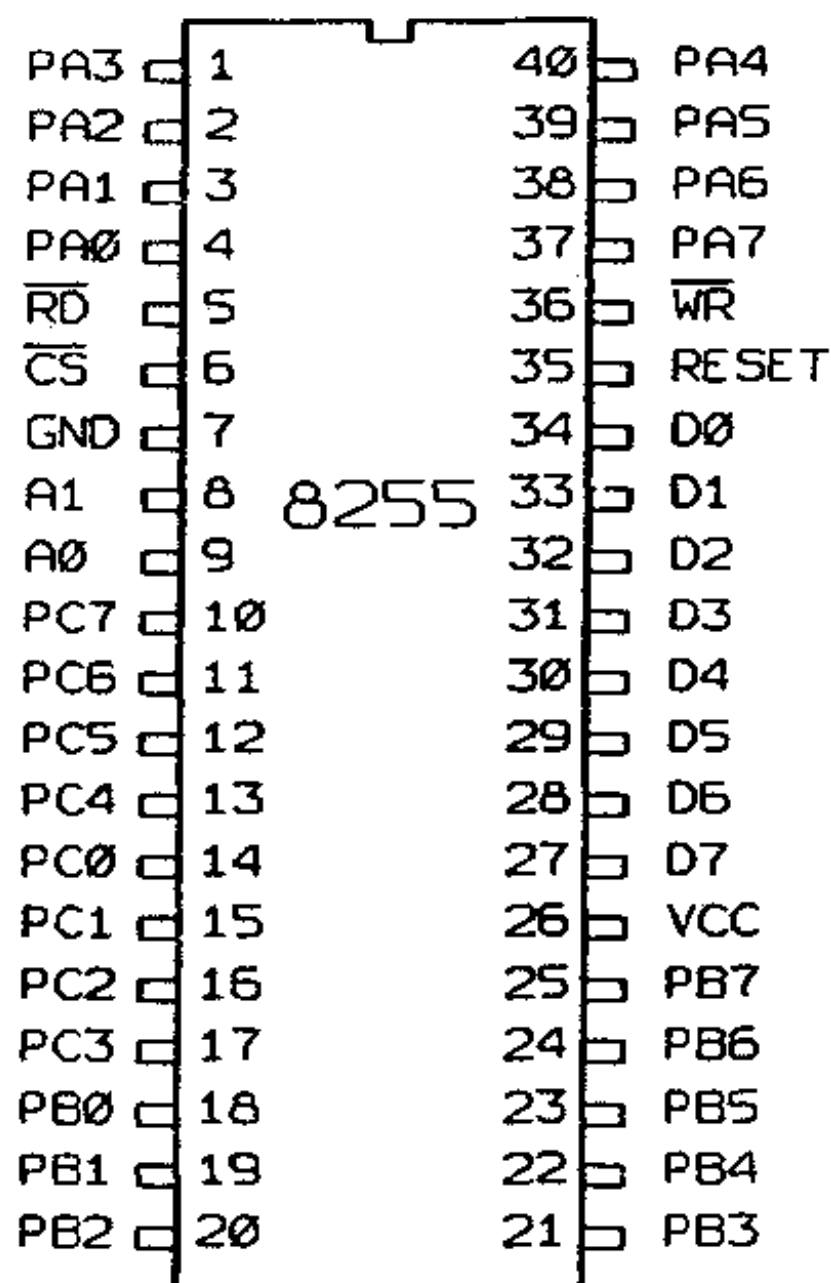


图 6-2 8255 引脚说明

• A0, A1 (I/O 端口选择线)：输入此二信号（一般接系统地址总线的 A0 及 A1）用来选择 8255 内部 4 个控制端口寄存器，分别为端口 A，端口 B，端口 C 及命令控制寄存器，如下所示：

A1	A0	8255 内部寄存器
0	0	端口 A
0	1	端口 B
1	0	端口 C
1	1	控制寄存器

• D0~D7 (数据总线)：双向输入输出

8255 的 8 条数据线连至系统的数据总线，微处理器在执行数据输出或输入指令时，都是通过此总线。当芯片选择(CS)信号为高电位时此数据总线呈现高阻抗。

- RESET (复位): 输入

高电位工作, 用来清除内部控制寄存器, 同时将 3 个 I/O 端口全部设为输入。

- CS (芯片选择信号): 输入

低电位工作, CS=0 时将内部数据总线与系统总线连接在一起, 可以直接控制 8255 工作。当 CS=1 时则 D0~D7 与系统总线切断, 成为高阻抗。

- WR (写入信号): 输入

低电位工作, 配合 CS 信号将处理器数据写入 8255 内。

- RD (读取信号): 输入

低电位工作, 配合 CS 信号读取 8255 内部寄存器的值。

- PA0~PA7 (端口 A): 输入或输出

- PB0~PB7 (端口 B): 输入或输出

- PC0~PC7 (端口 C): 输入或输出

8255 3 个可编程规划的输入输出端口, 用于外围设备连接。其中端口 A、端口 B 可以全部设为输入或输出, 端口 C 则可分别设定为两组 (PC0~PC3; PC4~PC7) 4 位的输入或输出端口。

- VCC (电源)

+5V 电源输入。

- GND (接地)

芯片接地。

6.3 8255 工作说明

8255 的 A0 及 A1 引脚可以选取 4 个内部寄存器端口 A、端口 B、端口 C 和控制寄存器, 其中 3 个 I/O 寄存器用来存储 3 个输入输出端口的数据, 表 6-1 列出其工作情况。

例如当(A1, A0) = (0, 0), 选取端口 A 的寄存器, 若 RD=0 是做端口 A 的数据读取 (由端口 A 读出数据); WR=0 便做端口 A 数据写入 (向端口 A 写入数据)。当(A1, A0) = (1, 1), WR=0 是做控制字的写入, 但无法进行读取。8255 共提供 3 种不同的操作模式来配合各种不同的 I/O 接口控制。

1. 模式 0: 基本 I/O 控制;
2. 模式 1: 触发式 I/O 控制;
3. 模式 2: 触发式双向 I/O 控制。

至于如何设定, 可以向控制寄存器写入适当的方式控制字。

表 6-1 8255 工作信号列表

A1	A0	RD	WR	CS	动 作
0	0	0	1	0	读取端口 A 的数据
0	1	0	1	0	读取端口 B 的数据
1	0	0	1	0	读取端口 C 的数据
1	1	0	1	0	错误情况
0	0	1	0	0	数据写入端口 A
0	1	1	0	0	数据写入端口 B
1	0	1	0	0	数据写入端口 C
1	1	1	0	0	写入控制寄存器
X	X	X	X	1	总线高阻抗
X	X	1	1	0	总线高阻抗

6.3.1 模式设定

图 6-3 为 8255 控制字格式:

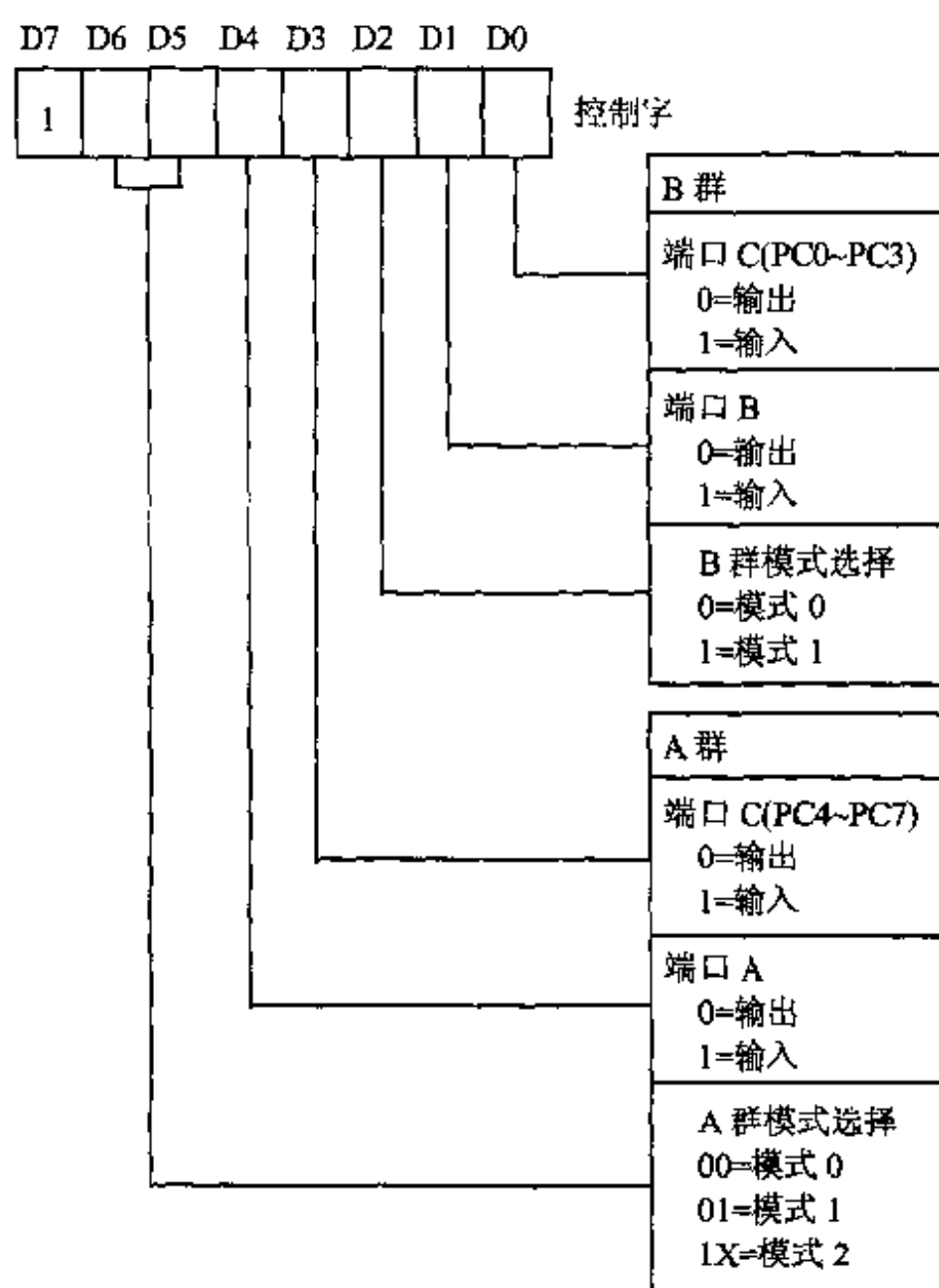


图 6-3 8255 控制字格式

其中:

D7 : D7=1, 是方式控制字的特征位。

D6, D5: 选择端口 A 及端口 C 下半部的操作模式。

D4 : 端口 A 的输入输出设定, “1”表示输入, “0”表示输出。“0”与“O”(out, 输出)相近, 可方便记忆。

D3 : PC4~PC7 输入输出设定。

D2 : 端口 B 及端口 C 上半部的操作模式设定。

D1 : 端口 B 的输入输出设定。

D0 : PC0~PC3 输入输出设定。

此外 8255 对端口 C 提供有位设定/清除的功能, 只要使用一个输出控制指令便可达到位控制的目的, 在做状态设定时非常方便。在控制字的 D7 位为 0 时是做端口 C 个别位控制用, 至于设定哪一个位由 D3, D2, D1 3 个位来选择, 设置情况如图 6-4 所示。例如: 设定位 2 为低电位, 送出控制字 0X04。

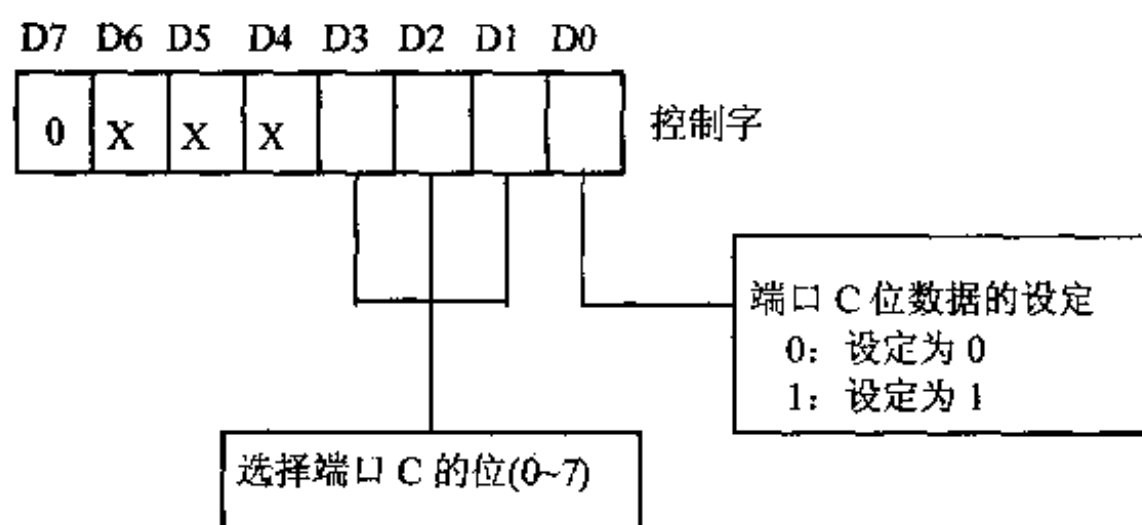


图 6-4 8255 位设定/清除格式

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	1	0	0

6.4 8255 工作模式 0

8255 工作模式 0 为基本输入输出控制, 通过控制字来规划 8255 的工作模式, 可以参考图 6-3 的说明。

例如: 控制字为 0X80:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0

则将 8255 设为模式 0 操作, 3 个 I/O 端口全部设为输出。现将在模式 0 操作下所有

可能的组合情况列表如下（表 6-2）：

表 6-2 8255 模式 0 控制寄存器

控制寄存器	PA0~PA7	PB0~PB7	PC4~PC7	PC0~PC3
80H	输出端口	输出端口	输出端口	输出端口
81H	输出端口	输出端口	输出端口	输入端口
82H	输出端口	输入端口	输出端口	输出端口
83H	输出端口	输入端口	输出端口	输入端口
88H	输出端口	输出端口	输入端口	输出端口
89H	输出端口	输出端口	输入端口	输入端口
8AH	输出端口	输入端口	输入端口	输出端口
8BH	输出端口	输入端口	输入端口	输入端口
90H	输入端口	输出端口	输出端口	输出端口
91H	输入端口	输出端口	输出端口	输入端口
92H	输入端口	输入端口	输出端口	输出端口
93H	输入端口	输入端口	输出端口	输入端口
98H	输入端口	输出端口	输入端口	输出端口
99H	输入端口	输出端口	输入端口	输入端口
9AH	输入端口	输入端口	输入端口	输出端口
9BH	输入端口	输入端口	输入端口	输入端口

6.5 8255 模式 1 工作

8255 模式 1 的工作提供一种触发式或交互式 (HandShaking) 的方式来与外围设备做数据的转移。图 6-5 为交互式数据传输工作图，当 8255 传送数据到外围设备时，先将数据送出并锁存在输出缓冲器中，然后通知对方数据已送出（即输出缓冲器已满），等对方收到数据并响应已收到数据的认可信号后，方才再送出下一行数据，如此双向信号的沟通保证可以将信号正确的传送到外围设备。

同理由 8255 要接收数据进来时，外围设备先将数据放置在 8235 的输入端口上，并输出数据锁存信号，告诉 8255 数据已送出，只要 CPU 未将传来的数据取回，会一直送出数据未读取（即输入缓冲器已满信号），一旦 CPU 读取数据后输入缓冲器已满信号清除，外围设备可以再送出下一行数据进来。

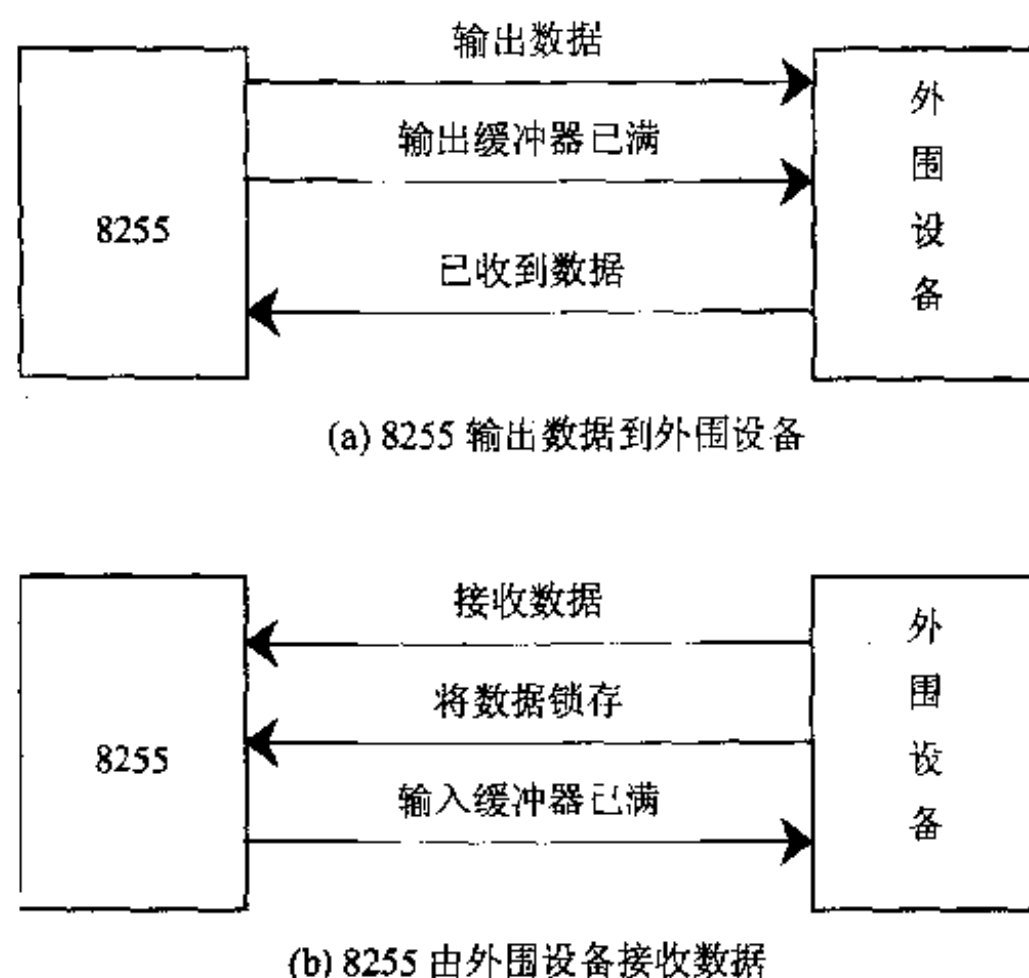


图 6-5 8255 交互式控制示意图

6.5.1 模式 1 的输入控制方式

8255 在此模式工作时，可以由端口 A 及端口 B 来输入数据，当由端口 A 来输入数据时，是由 PC3、PC4、PC5 来提供交互式控制信号，而 PC6、PC7 可以作为一般的 I/O 位来控制其中的交互式控制信号，说明如下：

- STB (Strobe): 锁存信号低电位工作，外围设备送至 8255。当外围设备将数据送至 8255 时，同时会送出此信号，使 8255 能够立即将数据锁存到输入缓冲器内。
- IBF (Input Buffer Full): 输入缓冲区已满信号高电位工作，由 8255 产生送到外围设备。告诉外围设备输入缓冲区内的数据尚未被 CPU 读取，暂时不要再送数据进来。
- INTR (Interrupt Request): 中断请求信号，高电位工作，由 8255 产生接到 CPU，CPU 产生中断请求，告诉 CPU 可以读取输入缓冲区内的数据。

其端口 A 与端口 B 的输入相对两组信号交换引脚分配如图 6-6 所示，其中“INTE A”信号是由 PC4 位做状态设定或清除；“INTE B”则由 PC2 位做控制。

整个交互式工作情况，我们以时序图来看比较容易，图 6-7 为其输入的工作时序图，工作如下：

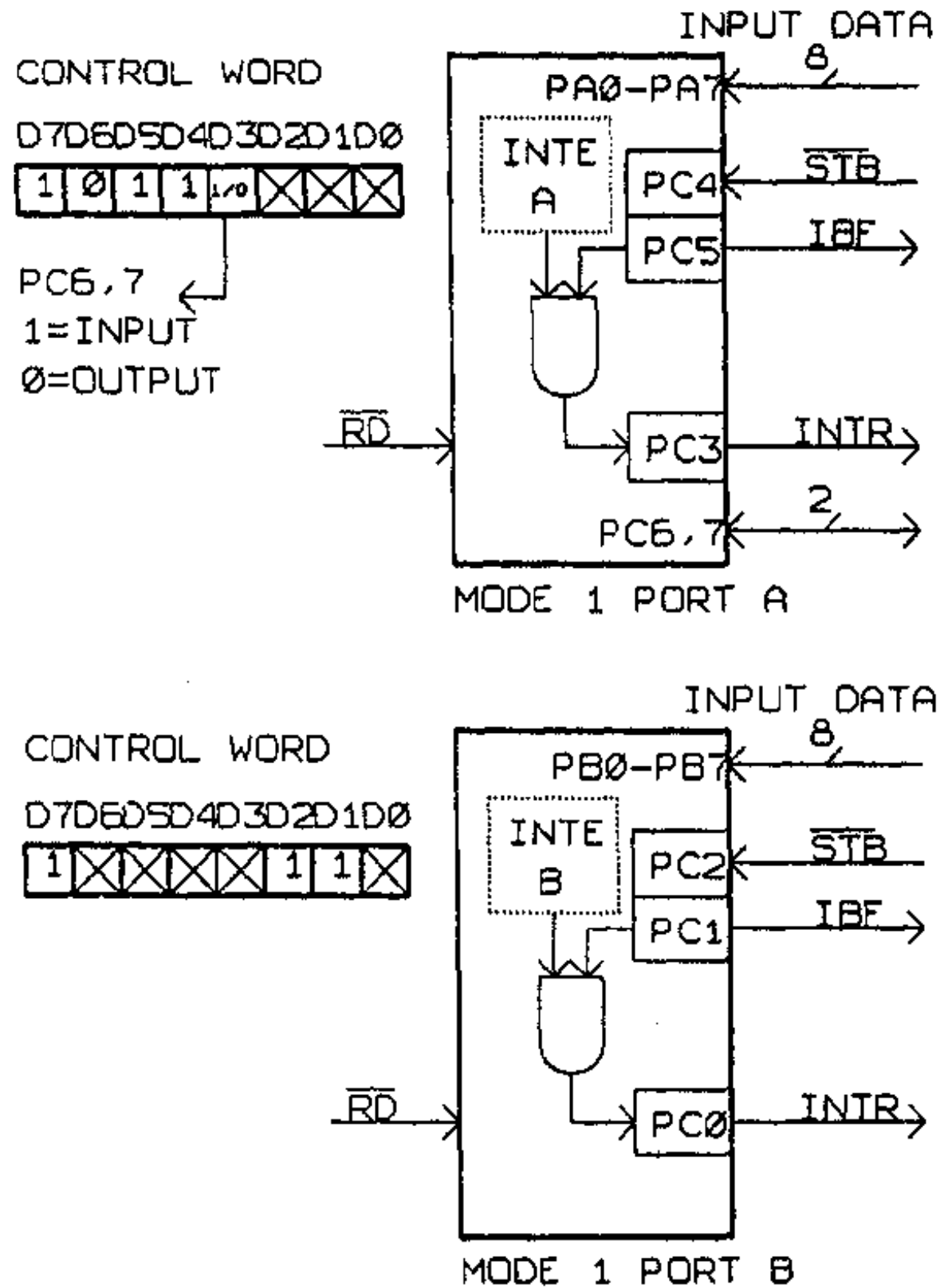


图 6-6 8255 模式 1 的输入控制

1. 外围设备先将数据送至 8255 的输入缓冲区内，同时送出 STB 信号，告诉 8255 将数据锁存住，当 8255 将该数据锁存住后，立即送出 IBF 信号（高电位工作）给外围设备，告诉外围设备先不要再送数据进来，8255 尚未处理完此数据。
2. 当 STB 信号恢复为 1 时，且 IBF 信号为 1（即 STB=1 和 IBF=1），促使 8255 产生 INTR 中断信号，要求 CPU 执行中断程序来读取 8255 输入缓冲器内部的数据。
3. 当 CPU 接受中断执行中断程序而读取 8255 输入缓冲器内的数据时，于是产生 RD 信号，在 RD 信号下降的同时将 INTR 信号加以清 0。
4. 当 RD 信号结束时（RD 上升缘），促使 IBF 信号为 0，告诉外围设备送来的数据已被读取，可以再送下一组数据进来。

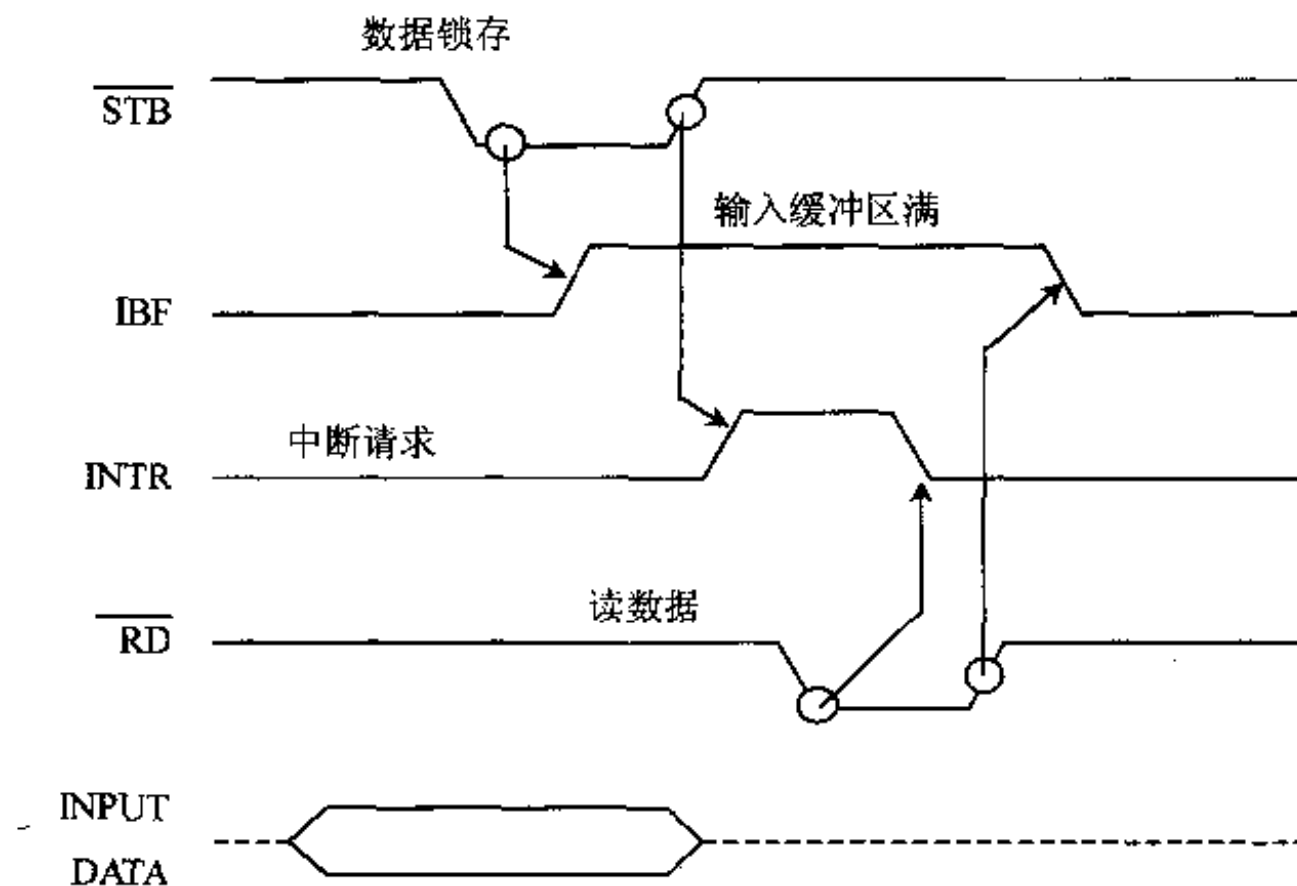


图 6-7 8255 模式 1 的输入操作时序图

6.5.2 模式 1 的输出控制方式

交互式的输出控制同输入一样也是分为端口 A 及端口 B 两组来送出数据，只是此时的交互式控制信号不同，变为 OBF，及 ACK，而 INTR 信号仍是中断信号，现分别说明如下：

- OBF (Output Buffer Full)：输出缓冲器已满信号，低电位工作，由 8255 送到外围设备告诉对方数据已经送过去了，请将数据取走。
- ACK (Acknowledge)：认可信号，低电位动作，由外围设备送过来，通知 8255，外围设备已将数据取走，可以传送下一行数据过来了。
- INTR：中断请求，高电位工作，由 8255 向 CPU 提出中断请求，CPU 可以将欲传送的数据写到输出缓冲器中。

图 6-8 为其控制方式，图 6-9 则是工作时序图，工作如下：

1. CPU 通过数据总线，将想要传送的数据送到 8255 的输出缓冲器上，同时 WR 信号为高电位。
2. 在 WR 信号恢复为高电位时，清除原先产生的中断信号 INTR，成为低电位，同时也引发 OBF 信号，告诉外围设备数据已准备好，可以来提取数据。
3. 当外围设备将数据读取后，随即响应 ACK 信号给 8255，在 ACK 信号的下降缘处将 OBF 信号清除。
4. 在 ACK 信号的上升缘处，同时 WR 及 OBF 信号都为高电位时引发中断信号，要求 CPU 中断，准备送出下一组数据。

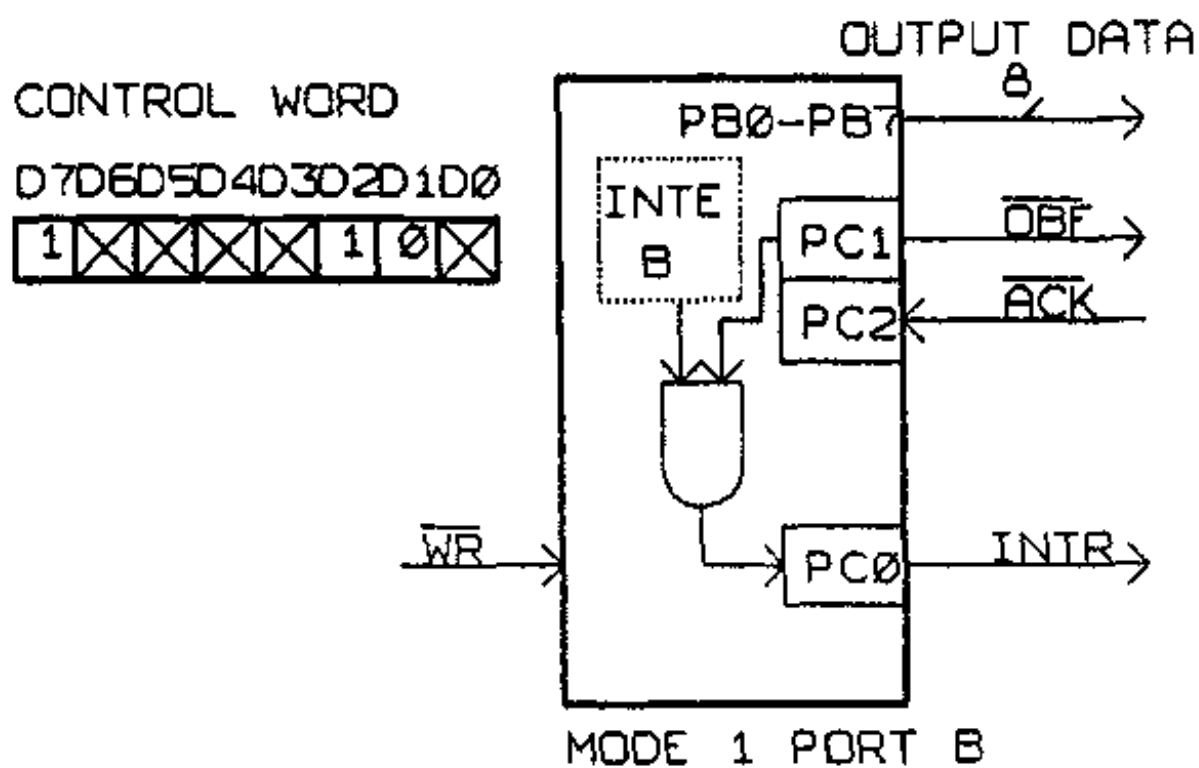
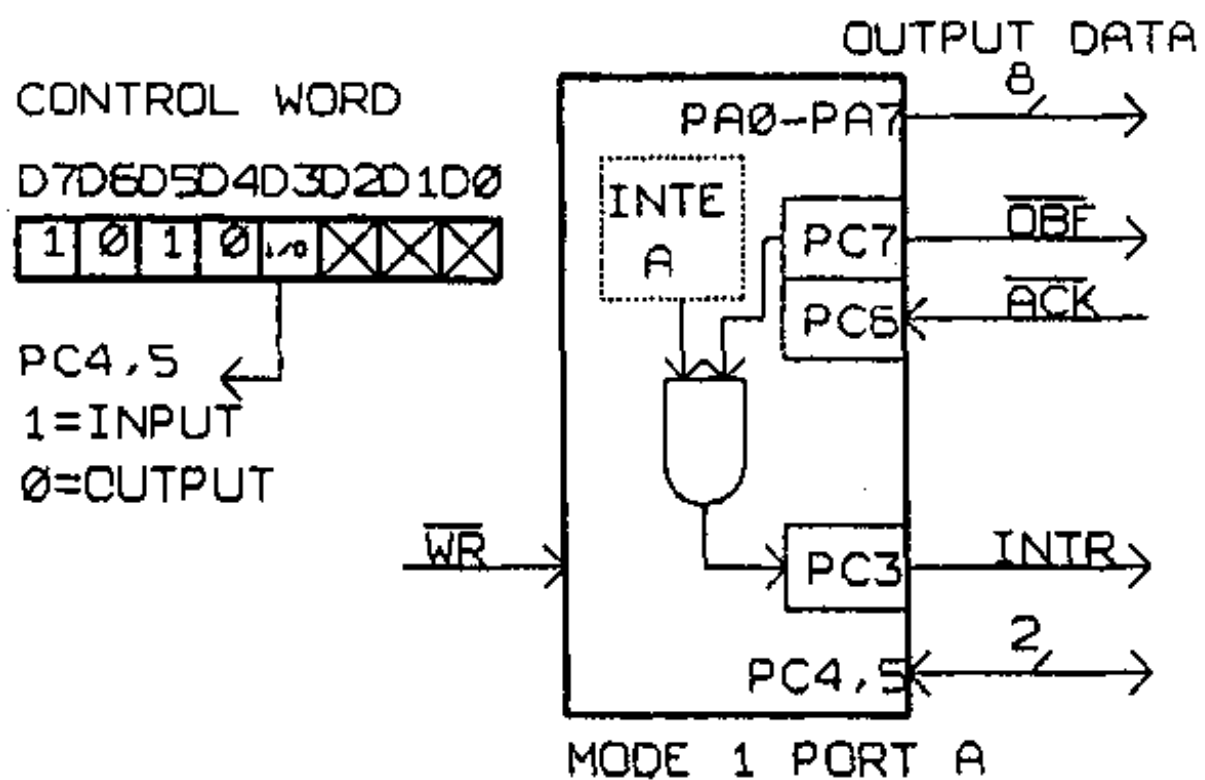


图 6-8 8255 模式 1 的输出控制

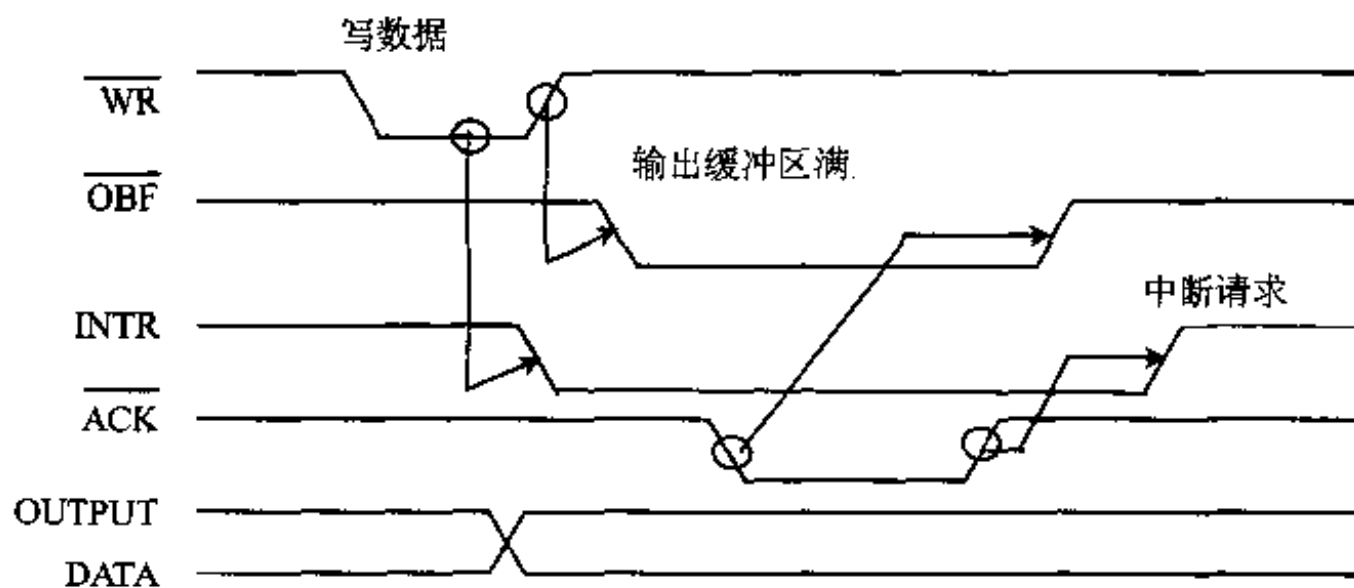


图 6-9 8255 模式 1 的输出操作时序图

6.5.3 模式 1 的组合

在模式 1 的操作下端口 A 与端口 B 可以个别的规划为输入或输出, 因此同样利用 2 块 8255 便可以互相传输数据。图 6-10 是其操作时的组合情况。

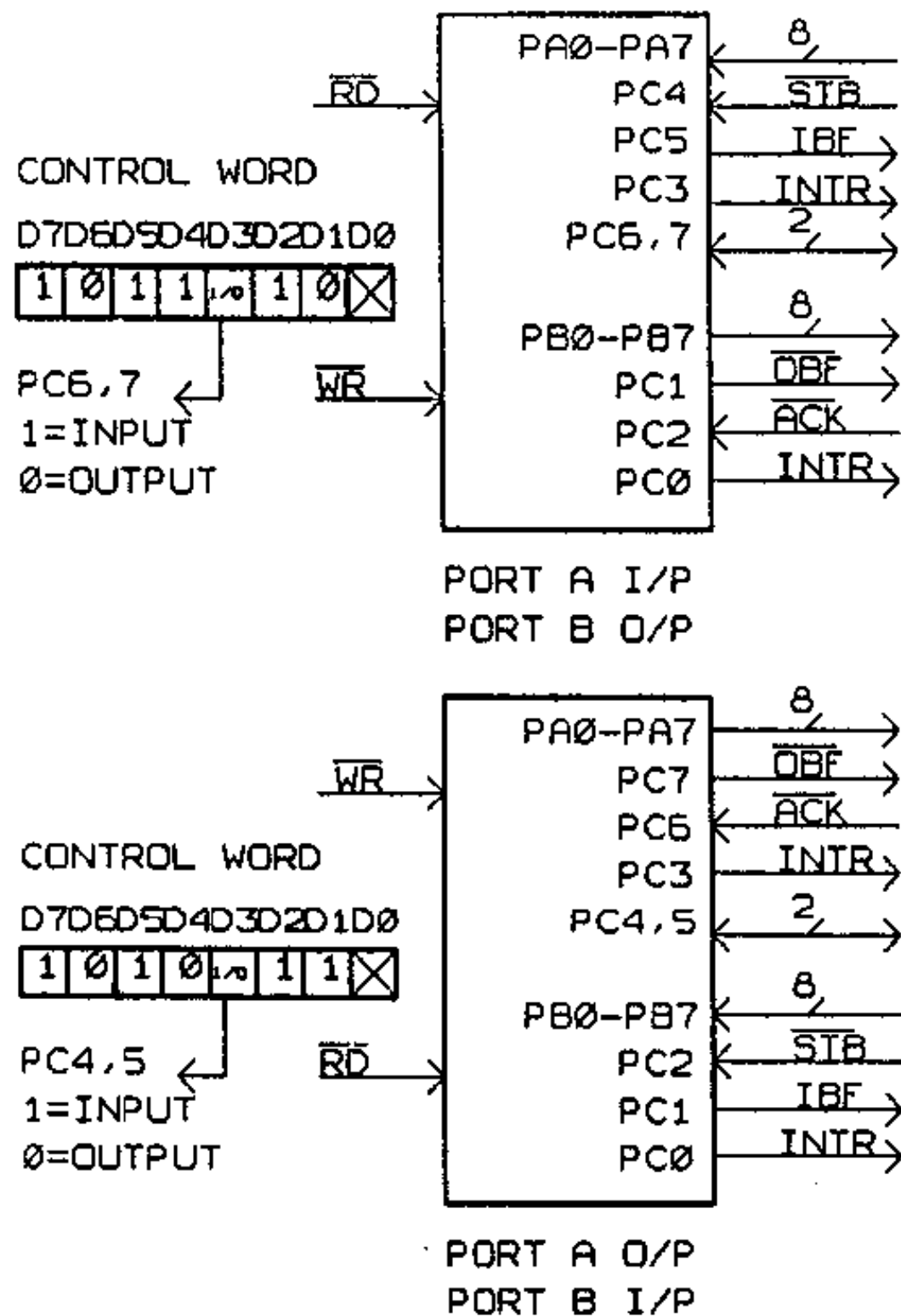


图 6-10 8255 模式 1 的组合方式

6.6 8255 模式 2 工作

8255 模式 2 的操作提供一种触发式双向总线输入输出的方法, 来与外围设备做数据的转移。其传输的方式同模式 1 一样, 也是使用交互及中断的控制方式。所不同的是数据的传送只能通过端口 A, 同时做输入或输出, 而由端口 C PC3~PC7 5 个引脚做交互的信号控制。图 6-11 为模式 2 控制字的组成示意图。在此模式中端口 B 只能在模式 0 或模式 1 下工作。模式 2 的交互控制信号与模式 1 一样, 其中中断请求信号 INTR 产生时, 表示进行下一行数据的转移。

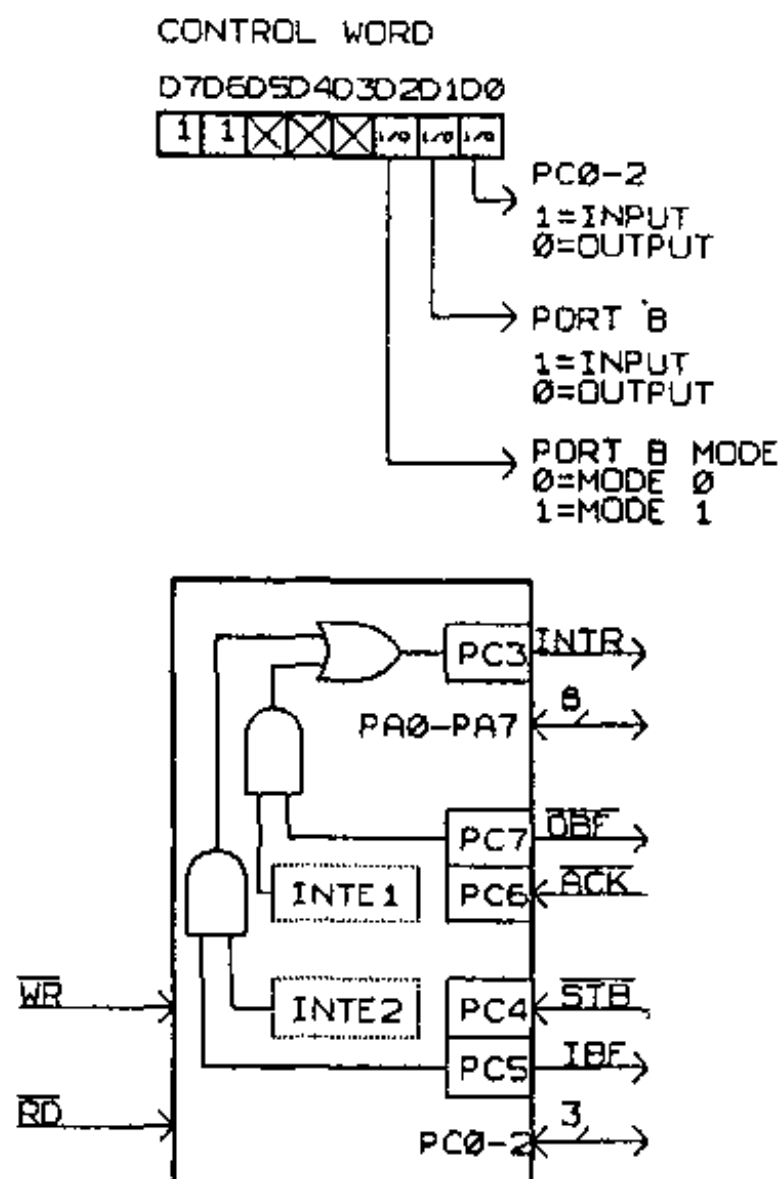


图 6-11 8255 模式 2 的控制字

6.6.1 模式 2 的组合方式

在此模式的操作下，共有以下 4 种组合情况，如图 6-12 所示。

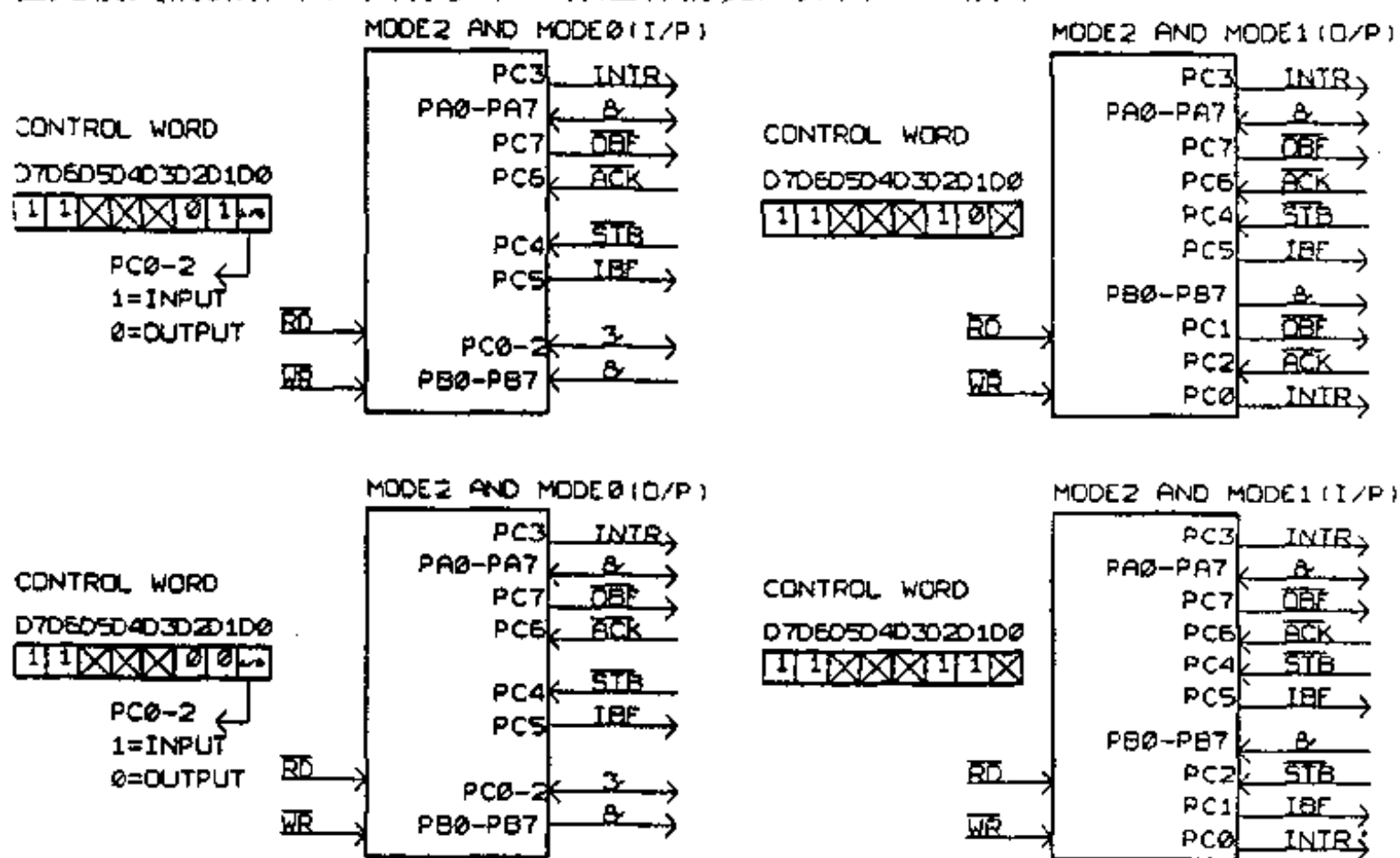
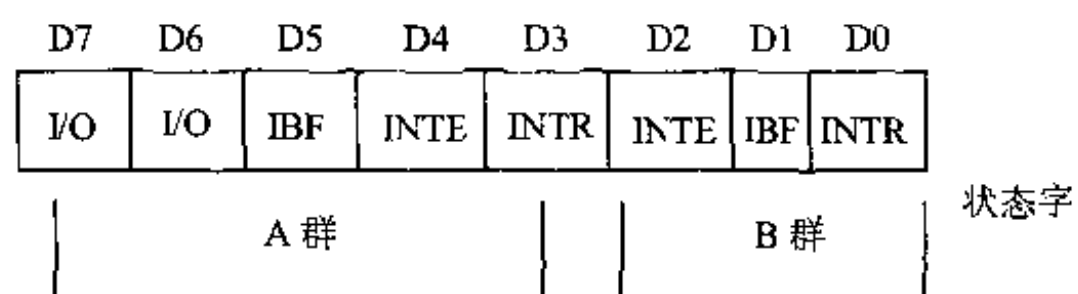


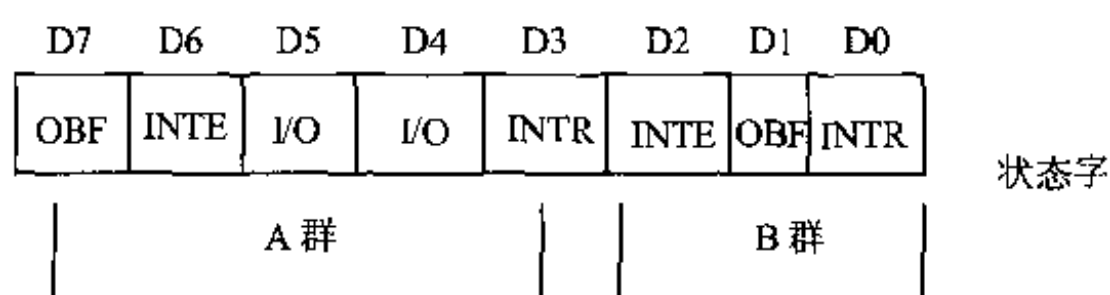
图 6-12 8255 模式 2 的组合方式

6.7 8255 端口 C 的交互式控制信号状态读取

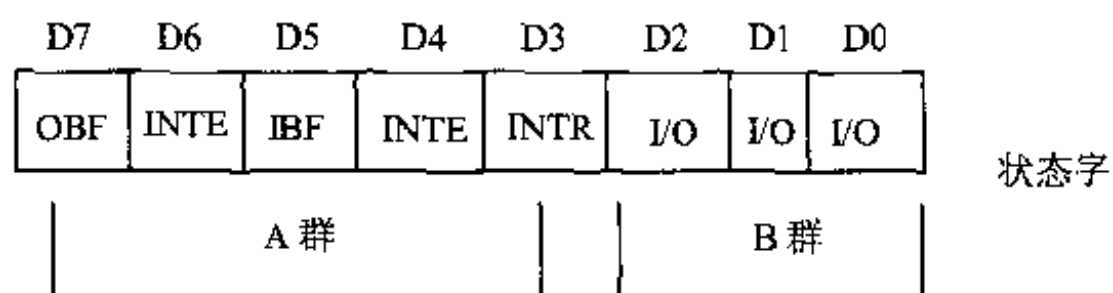
以上几节中我们介绍了 8255 工作模式 1 及模式 2 的交互式数据传送工作原理，这些交互式控制信号都是由端口 C 来完成的，我们可以读取端口 C 的内容来了解交互式控制信号的状态。图 6-13 为模式 1 及模式 2 的交互式控制信号状态字的定义。



(a) 模式 1 输入时端口 C 的状态字



(b) 模式 1 输出时端口 C 的状态字



(c) 模式 2 时端口 C 的状态字

图 6-13 8255 端口 C 所代表的状态字

例如想利用模式 1 端口 A 做数据输入可以以端口 C 位设定功能令 PC4 为 1，PC4 表示模 1 输入状态字中 INTE 的工作位，设为允许中断。此时一旦外界送入数据时可以产生 INTR 中断信号而通知 CPU 来读取端口 A 的数据。

6.8 8255 接口电路测试

看过完整的 8255 芯片工作说明后，本节介绍 8255 接口电路测试程序。在 8051 多功能控制板上，由于单片机 8051 可以通过外接程序 ROM 来执行程序，因此 I/O 的控制引脚减少了很多，为了考虑未来的功能扩充，在控制板上设计了 3 块 8255 芯片，其中 U11 设计用来做 7 段数码管，按键扫描及 D/A 接口控制，另外 2 块 8255 芯片可以完全来做 I/O

的额外硬件扩充。本节介绍的测试程序即是用来测试 U11 芯片是否正常,其他的芯片测试原理一样,只是解码地址不同。

6.8.1 8255 接口电路测试功能

将多功能控制板上的 RS232 接口连至 PC, 相对的执行消息会显示在屏幕上, U11 IC 座上如果未装有 8255 或是此芯片为次品, 程序执行后工作指示灯 LED 会一闪一闪持续下去并显示: 8255 test error !

如果芯片测试正常则显示:

8255 test OK !

6.8.2 P51 I/O 控制头文件 P51.H

```

1  /*  P51.H */
2  /*****
3  HEADER:  P51.H    in P51_PCB
4  C Compiler:  MICRO-C51
5  Description:  P51_PCB header
6  Version: V1.0   97/7/7
7  Copyright (C) VICTOR uP LAB. 1997-1998
8  *****/
9
10 #define cr (*(char*) 0x8003)
11 #define pa (*(char*) 0x8000) /* U11 8255 基底 I/O 端口 */
12 #define pb (*(char*) 0x8001) /* D/A, 7SEG, KEY PORT */
13 #define pc (*(char*) 0x8002)
14
15 #define psg_reg (*(char*) 0x9000) /* PSG 基底 I/O 端口 */
16 #define psg_data (*(char*) 0x9001)
17
18 #define slot (*(char*) 0xA250) /* 插槽基底 I/O 端口 */
19
20 #define ad_port (*(char*) 0xA250) /* 语音卡 A/D 端口 */
21 #define da_port (*(char*) 0xA251) /* 语音卡 D/A 端口 */
22 #define io_port (*(char*) 0xA252) /* 语音卡 I/O 端口 */
23 #define cw_port (*(char*) 0xA253) /* 语音卡模式控制端口 */
24
25 #define crl (*(char*) 0xB003)

```



```

26 #define pa1      (*(char*) 0xB000) /*U12 8255 基底 I/O 端口*/
27 #define pb1      (*(char*) 0xB001)
28 #define pc1      (*(char*) 0xB002)
29
30 #define cr2      (*(char*) 0xC003)
31 #define pa2      (*(char*) 0xC000) /*U21 8255 基底 I/O 端口*/
32 #define pb2      (*(char*) 0xC001)
33 #define pc2      (*(char*) 0xC002)
34
35 #define lcd_com  (*(char*) 0xF000) /* LCD I/O 端口 */
36 #define lcd_data (*(char*) 0xF001)

```

程序清单

```

1  /*  T55.C  */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "c:p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  #define CWORD 0x88 /* PA PB PC0~3    o/p PC4~7 i/p */
9  char *title="Test P51_PCB 8255 ...";
10 /*-----*/
11 delay(int t) /* 延迟子程序 */
12 {
13     int i, j;
14     for(i=0; i<t; i++)
15         for(j=0; j<10; j++);
16 }
17 /*-----*/
18 led_bli() /* 工作 LED 闪动 */
19 {
20     /*  blink RED led P1.7  */
21     char i;
22
23     for(i=0; i<4; i++)
24     {

```

```
25 cplbit(P1.7); /* 位取反 */
26 delay(40);
27 }
28 }
29 /*-----*/
30 main() /* 主程序 */
31 {
32     led_bl(); /* 工作 LED 闪动 */
33     cr=CWORD; /* 设定 8255 工作模式 */
34     serinit(9600); /* 初始化串行接口 */
35     printf(title); /* 显示工作消息 */
36
37     test_8255(); /* 测试 P51 U11 8255 芯片 */
38     while(1){} /* 无穷循环 */
39 }
40 /*-----*/
41 test_8255() /* 测试 P51 U11 芯片 */
42 {
43     char err;
44     int i;
45     unsigned char in;
46
47     cr=0x80; /* 8255 3 个端口全部设为输出 */
48     err=0; /* 清除错误标志 */
49     printf("TEST main 8255 ...");
50     for(i=0; i<256; i++) /* PA 端口测试 */
51     {
52         pa=i; /* 写入样本 */ in=pa; /* 读回样本 */
53         if( in!=i) { err=1; goto error; }
54     }
55
56     for(i=0; i<256; i++) /* PB 端口测试 */
57     {
58         pb=i; in=pb;
59         if( in!=i) { err=1; goto error; }
60     }
```

```
61
62  for(i=0; i<256; i++) /* PC 端口测试 */
63  {
64    pc=i; in=pc;
65    if( in!=i) { err=1; goto error; }
66  }
67
68  error: /* 显示测试结果 */
69    if(err) printf("8255 test   error !\n");
70    else   printf("8255 test  OK !   \n");
71    if(err) while(1) led_bl(); /* 工作 LED 持续闪动 */
72  }
73  /*-----*/
```

6.9 习 题

1. 说明 8255 下列引脚功能: A0、A1、RD\、WR\。
2. 说明 8255 3 种工作模式有何不同。
3. 若要将 8255 端口 C 位 1 3 5 设为 1, 其余位设为 0, 用位设定/清除的功能该如何设计程序?
4. 试设计 8051 接口电路来控制 8255, 其 I/O 基地址为 0X9000。
5. 若 8255 控制字为 0X99, 则 8255 工作功能如何?
6. 若想规划 8255 工作为模式 0 操作, 端口 A 输入, 端口 B 及端口 C 为输出, 则 8255 控制字如何?
7. 写一程序来测试 P51 控制板上其他 2 块 8255 芯片。

第 7 章 多功能控制板基本 I/O 功能

当自制的 8051 单片机电路板测试成功后,我们可以利用 8051 单板来做一些基本的 I/O 控制接口,像是工作 LED 指示灯、“走马灯”式电路的控制、7 段数码管控制、DIP 开关控制及按键控制等实验,至于更复杂的接口可以按照需要而加以扩充。此外读者可以使用 DIP 信号线或是单心线,由 8051 单板拉至面包板上来做这些基础的 I/O 控制实验,除了可以了解基本硬件的电路控制外,也可以熟悉基本 C 程序语言的一些指令写法。

7.1 单板上工作指示 LED

在单板上 8051 端口 1 位 7 (P1.7) 接有一 LED 指示灯,我们称为工作指示 LED,送出低电位时,LED 点亮,高电位时则使 LED 熄灭。我们可以用来表示如下的一些状态:

- 程序执行到特定状态时,LED 闪动一下。
- 做状态区分表示,如状态 1 闪动一下,状态 2 闪动两下,余此类推。
- 程序执行遇到特殊错误时,持续闪动着。

因此靠一个 LED 闪动情况,可以判断程序执行的正确性及显示程序执行的结果,这是一个相当简单容易的输出装置,用来表示一个位的数字状态。

执行结果

程序执行后工作指示 LED 灯持续闪动着。

程序清单

```
1  /* WLED.C */
2  #include "8051io.h"      /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6
7  char *title="MC51 test P51_PCB working led";
8
9  delay(int t) /* 延迟子程序 */
10 {
11     int i,j;
12     for(i=0; i<t; i++)
```

```

13     for(j=0; j<10; j++);
14 }
15 /*-----*/
16 main()    /* 主程序 */
17 {
18     serinit(9600); /* 初始化串行接口 */
19     printf(title); /* 显示工作消息 */
20
21     cplbit(P1.7); /* P1.7 位取反 */
22     delay(600);
23
24     setbit(P1.7);
25     delay(600);
26
27     while(1) /* 无穷循环 */
28     {
29         cplbit(P1.7);
30         delay(50);
31     }
32 }

```

7.2 “走马灯”式电路控制

本节的实验是在 8051 端口 1 P1 (接有提升电阻), 连接 8 个 LED 做“走马灯”式电路展示, 其中使用左旋转指令, 当相对位输出低电位, LED 顺向导通而发亮, 输出高电位, LED 截止而熄灭, 串接提升电阻是作为限流电阻用, 避免 LED 流过过大电流损坏, 一般工作电流 5mA~10mA 即可, 当流过的电流越大则越亮。

执行结果

程序执行后接于 P1 的 8 个 LED 灯闪动 5 次, 然后进入循环一直执行左旋转 LED 灯工作。

程序清单

```

1  /* LED.C */
2  #include "8051io.h"    /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"

```

```
5  #include  "8051int.h"
6
7  char *title="MC51 test P51_PCB 8 leds\r\n";
8
9  delay(int t) /* 延迟子程序 */
10 {
11     int i,j;
12     for(i=0; i<t; i++)
13         for(j=0; j<10; j++);
14 }
15 /*-----*/
16 main() /* 主程序 */
17 {
18     char i,c;
19
20     serinit(9600);/* 初始化串行接口 */
21     printf(title);/* 显示工作消息 */
22
23     P1=0;        /* LED 全亮 */
24     delay(600);
25
26     P1=0xff;     /* LED 全灭 */
27     delay(600);
28
29     for(i=0; i<5; i++) /* LED 闪动 5 次 */
30     {
31         P1=0;
32         delay(50);
33
34         P1=0xff;
35         delay(50);
36     }
37
38     while(1)     /* 无穷循环 */
39     {
40         c=0xfe;  /* 设定初始值 */
```

```

41  P1=c;      /* 由 P1 送出 */
42  delay(50);
43  for(i=0; i<8; i++)
44  {
45      c=c<<1; /* 控制值左旋转一位 */
46      P1=c;
47      delay(50);
48  }
49  }/* loop */
50  }
51  /*-----*/

```

上一个程序所介绍的“走马灯”式电路展示方法是以左旋转指令来完成的，一般还可以使用查表的方式来做控制，所谓查表的方式是将一些特定的数据，在此是 LED 展示的变化组合数据事先存在数组中，而在程序中逐一由数组中取出个别的样本数据往 LED 输出端口 P1 送出，便可完成“走马灯”式电路展示的效果。由于展示的样本数据可以随意组合，因此以查表法来做“走马灯”式电路控制的变化较多，展示效果较佳。

执行结果

程序执行后 8 个 LED 做出 4 种不一样的“走马灯”式电路花样展示。

程序清单

```

1  /* LED1.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6
7  char *title="P51_PCB leds rotate by table\n";
8  /* LED 花样展示数据 */
9  char led1[]={0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE};
10 char led2[]={0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F};
11 char led3[]={0x7E, 0xBD, 0xDB, 0xE7, 0xE7, 0xDB, 0xBD, 0x7E};
12 char led4[]={0x7E, 0x3F, 0x1F, 0x0F, 0x07, 0x03, 0x01, 0x00};
13
14 /*-----*/
15 delay(int t)/* 延迟子程序*/
16 {

```

```
17 int i,j;
18   for(i=0; i<4; i++)
19     for(j=0; j<10; j++);
20 }
21 /*-----*/
22 rot(char *pt) /* 花样展示 */
23 {
24   char i;
25
26   for(i=0; i<8; i++)
27   { /* 由数组中取出个别的样本数据往 P1 送出 */
28     P1=pt[i];
29     delay(50);
30   }
31
32 }
33 /*-----*/
34 main() /* 主程序 */
35 {
36   serinit(9600); /* 初始化串行接口 */
37   printf(title); /* 显示工作消息 */
38
39   while(1) /* 无穷循环 */
40   {
41     rot(led1); /* 4 种花样展示 */
42     rot(led2);
43     rot(led3);
44     rot(led4);
45   }
46 }
```

7.3 读取 DIP 开关设定

DIP 开关一般用在单片机控制板上、硬件的解码设定或特殊状态设定上, 当它置于 ON 位置时, 则相对的位导通。DIP 开关接至 I/O 作为输入时通常通过提升电阻接至 +5V 电源, 平时读取为高电位, 一旦某位扳于 ON 位置时, 则该位被设定为 0, 此时电脑可以用

来做判断而做各种状态的设定。

在实验电路中, 8 位的 DIP 开关接于可编程声效发生器 U16 PSG 8910 的 PA 端口, 其设定值为 0~255, 其读取值可以由 PC 屏幕显示出来。有关 8910 详细的控制原理可以查看 12.1 节 PA PB I/O 设定说明。

注意: 8051 单板要连至 PC RS232 端通信端口 2, 并执行 SE.EXE 程序。

执行结果

程序执行后 PC 屏幕显示 DIP 开关设定的读取值, 例如将 DIP 开关 1、2、3 位置为 off 时, 则读值为 0x07, 如下所示:

位	B7	B6	B5	B4	B3	B2	B1	B0
DIP 开关编号	8	7	6	5	4	3	2	1
DIP 开关设定	on	on	on	on	on	off	off	off
读取值	0	0	0	0	0	1	1	1

程序清单

```

1  /*  DIP.C  */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  char *title="Test P51_PCB DIP SW.";
9  /*-----*/
10 delay(int t) /* 延迟子程序 */
11 {
12     int i,j;
13     for(i=0; i<t; i++)
14         for(j=0; j<10; j++);
15 }
16 /*-----*/
17 led_bl() /* 工作 LED 闪动 */
18 {
19     /*  blink RED led P1.7 */
20     char i;
21

```

```
22  for(i=0; i<4; i++)
23  {
24      cplbit(P1.7); /* 位取反 */
25      delay(40);
26  }
27 }
28 /*-----*/
29 cpsg(char r, char d) /* PSG I/O 控制 */
30 {
31     psg_reg=r; /* 写入寄存器 */
32     psg_data=d; /* 写入数据到寄存器中 */
33 }
34 test_dip() /* DIP 开关设定测试程序 */
35 {
36     char c;
37
38     cpsg(7,0x3f); /* 设定 PSG 工作 PA PB i/p */
39     cpsg(14,0xff); /* PA 设为高电位,并设定为 PA 端口 */
40     printf("READ PSG PA dip SW data SW 123 off --> 0x07\n");
41     while(1)
42     {
43         c=psg_data; /* 由 PA 端口读取数据 */
44         printf("%xH ", c); /* 由 RS232 送出读取值 */
45         delay(300); /* 延迟一会儿 */
46     }
47 }
48 /*-----*/
49 main() /* 主程序 */
50 {
51     led_bl(); /* 工作 LED 闪动 */
52     serinit(9600); /* 初始化串行接口 */
53     printf(title); /* 显示工作消息 */
54     test_dip(); /* DIP 开关设定测试 */
55 }
56 /*-----*/
```

7.4 扫描控制 7 段数码管

LED 只能显示几个位的数字状态, 如果要显示数字的话就要使用 7 段数码管。7 段数码管是一般常用的输出结果显示元件, 可用来显示 0 到 9 的数字, 字型 A 到 F 或是某些特殊的字符, 在许多的应用场合中都可以派上用场, 若是要显示多位数字时, 可以串接起来利用单片机程序扫描的方式来驱动。

7.4.1 7 段数码管控制

7 段数码管可以分为 2 种, 一种共阴极, 一种是共阳极, 二者均有现成的解码驱动 IC, 前者为 7448, 后者为 7447。7 段数码管的数据 (a、b、c、d、e、f、g、dot) 是由 8051 P1 的 8 个位来控制, 各个位之间并加上了限流电阻, 当某位送出低电位时则点亮相对的显示器位。若要共阳极 7 段数码管显示数字 1, 位 b、c 都置 ON, 其余位置 OFF, 可以推算如下:

位	B7	B6	B5	B4	B3	B2	B1	B0
数据	dot	g	f	e	d	c	b	a
数值	1	1	1	1	1	0	0	1

所以只要由端口 A 送 0XF9 的数据, 便可以显示数字 1。同理可以推算显示其他数字送出数据的代码如下:

显示数字	0	1	2	3	4	5	6	7	8	9
数据代码	C0H	F9H	A4H	B0H	99H	92H	82H	F8H	80H	90H

显示数据	A	B	C	D	E	F
数据代码	88H	83H	46H	A1H	86H	8EH

同理共阴极 7 段数码管显示数字送出数据代码如下:

显示数字	0	1	2	3	4	5	6	7	8	9
数据代码	3FH	06H	5BH	4FH	66H	6DH	7DH	07H	7FH	6FH

显示数据	A	B	C	D	E	F
数据代码	77H	7CH	B9H	5EH	79H	71H

7.4.2 扫描控制 7 段数码管

7 段数码管是常用的数值数据显示元件, 前面我们分别看过一位, 二位 7 段数码管驱动的控制, 是使用静态的控制方式, 欲显示的数据送入解码器后程序可以做其他事情, 在程序设计上十分容易, 但是在做多位 7 段数码管显示控制时就太浪费 I/O 的控制线了,

对于多位显示器的驱动，最常使用的方法是扫描法，以动态扫描方式来控制较节省硬件成本。

在单板上连接有 4 只共阴极的 7 段数码管，并直接由 U11 8255 芯片以动态扫描方式驱动显示器，其优点是可以显示其他字符，如 a、b、c 等字符。

由 PB 端口(PB0~PB7) 输出 8 位数据，高电位工作依序对应如下：

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
dot	g	f	e	d	c	b	a

由 PC 端口(PC0~PC3) 送出扫描代码，低电位工作，任何时候只有 1 个位是低电位输出，即一次驱动一位数做数字显示。

执行结果

程序执行后，工作 LED 灯闪烁一下，四位共阴极 7 段数码管显示数字“1234”。

程序清单

```

1  /*  SEG.C  */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  #define CWORD 0x88 /* PA PB PC0~3 o/p PC4~7 i/p */
9  char *title="Test P51_PCB 7seg X4 ...";
10 /* 0~9 的字型数据 */
11 char DATA_7SEG[10]={0x3f,0x06,0x5b,0x4f,0x66,
12 0x6d,0x7d,0x07,0x7f,0x6f};
13 char act[4]={0xfe, 0xfd, 0xfb, 0xf7} /* 扫描控制信号 */
14 char disp[4]; /* 四位数码管数据缓冲区 */
15 /*-----*/
16 delay(int t) /* 延迟子程序 */
17 {
18 int i,j;
19 for(i=0; i<t; i++)
20 for(j=0; j<10; j++);
21 }
22 /*-----*/

```

```
23 led_bl() /* 工作 LED 闪动 */
24 {
25     /* blink RED led P1.7 */
26     char i;
27
28     for(i=0; i<4; i++)
29     {
30         cplbit(P1.7); /* 位取反 */
31         delay(40);
32     }
33 }
34 /*-----*/
35 init_buf() /* 载入四位数码管字符数据*/
36 {
37     int i;
38     for(i=0; i<4; i++)
39         disp[i]=DATA_7SEG[i];
40 }
41 /*-----*/
42 scan_seg()
43 {
44     char i;
45
46     for(i=0; i<4; i++)
47     {
48         /* 由 PB 端口(PB0~PB7)送出字符数据 */
49         pb=disp[i];
50         /* 由 PC 端口(PC0~PC3)送出扫描控制信号 */
51         pc=act[i];
52         delay(3); /* 延迟一下 */
53     }
54 }
55 /*-----*/
56 main()
57 {
58     led_bl(); /* 工作 LED 闪动 */
```

```
59   cr=CWORD; /* 设定 8255 工作模式 */
60   init_buf(); /* 载入四位数码管字符数据*/
61
62   serinit(9600); /* 初始化串行接口 */
63   printf(title); /* 显示工作消息 */
64   /* 无穷循环扫描推动显示器显示数据 */
65   while(1) scan_seg();
66 }
67 /*-----*/
```

7.5 键 盘 扫 描

在控制电路中，通常需要以按键来控制程序执行流程或是输入数据，如果是按键数不多时可以使用一个按键对应一条输入位线，但是若按键数太多时，通常是以矩阵式扫描法来做按键检查，在 8051 单板上设计有 16 个按键的控制电路，使用 8255 端口 C 8 条 I/O 线做 16 个按键的键盘扫描，由 PC.0~PC.3 送出扫描信号，而由 PC.4~PC.7 读取按键数据返回代码。

以程序扫描的方式来检查那一按键按下，一次扫描一行四个按键，扫描的顺序如下：

1. 送出扫描信号 1110 以扫描 C1 行的四个按键，读取按键数据，判断该行是否有键按下，若有键被按下，则连接至被按下的该键返回线状态为 0。
2. 送出扫描信号 1101 以扫描 C2 行的四个按键，读取按键数据，判断该行是否有键按下。
3. 送出扫描信号 1011 以扫描 C3 行的四个按键，读取按键数据，判断该行是否有键按下。
4. 送出扫描信号 0111 以扫描 C4 行的四个按键，读取按键数据，判断该行是否有键按下。
5. 回到步骤 1 继续做按键扫描。

以上的步骤连续地重覆，若有按键被按下，就将该按键解码出来，至于如何解码，可以使用双重循环做计数编号，当某一按键按下时，其按键编号便是计数编号，有关按键编号、扫描信号及读取按键数据返回代码列表如下：

键号	按键数据输入代码 PC. 7 PC. 6 PC. 5 PC. 4				扫描输出信号 PC. 3 PC. 2 PC. 1 PC. 0				所检查的按键
0	1	1	1	0	1	1	1	0	K 0 键
1	1	1	0	1	1	1	1	0	K 1 键
2	1	0	1	1	1	1	1	0	K 2 键
3	0	1	1	1	1	1	1	0	K 3 键
4	1	1	1	0	1	1	0	1	K 4 键
5	1	1	0	1	1	1	0	1	K 5 键
6	1	0	1	1	1	1	0	1	K 6 键
7	0	1	1	1	1	1	0	1	K 7 键
8	1	1	1	0	1	0	1	1	K 8 键
9	1	1	0	1	1	0	1	1	K 9 键
10	1	0	1	1	1	0	1	1	K 10 键
11	0	1	1	1	1	0	1	1	K 11 键
12	1	1	1	0	0	1	1	1	K 12 键
13	1	1	0	1	0	1	1	1	K 13 键
14	1	0	1	1	0	1	1	1	K 14 键
15	0	1	1	1	0	1	1	1	K 15 键

8051 多功能控制板 P51_PCB 上键盘的 16 个按键对应如下:

按键编号	K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11	K12	K13	K14	K15
按键标名	A	B	C	D	3	6	9	#	2	5	8	0	1	4	7	*

键盘的按键位置分配如下:

1	2	3	A
4	5	6	B
7	8	9	C
*	0	#	D

执行结果

程序执行后, 控制程序不断做键盘扫描, 当有按键按下时, 由 PC 端显示出按键的结果。目前程序只显示“1”或“2”按键是否按下, 其他的按键则未处理。

程序清单

```
1 /* KEY.C */
```

```
2  /* 载入 MC51 头文件 */
3  #include "8051io.h"
4  #include "8051reg.h"
5  #include "8051bit.h"
6  #include "8051int.h"
7  #include "c: p51.h"      /* 载入 P51 I/O 控制头文件 */
8
9  #define CWORD  0x88      /* PA PB PC0~3 o/p  PC4~7 i/p */
10 char *title="Test P51_PCB keys ...";
11 char act[4]={0xfe, 0xfd, 0xfb, 0xf7} /* 扫描控制信号 */
12 /* 键盘扫描值 */
13 char key; /* 16 个键盘按键 */
14 char skey[16]={'A', 'B', 'C', 'D',   '3', '6', '9', '#',
15               '2', '5', '8', '0',   '1', '4', '7', '*'};
16 char scan_key(); /* 键盘扫描控制程序 */
17 /*-----*/
18 delay(int t) /* 延迟子程序 */
19 {
20     int i,j;
21     for(i=0; i<t; i++)
22         for(j=0; j<10; j++);
23 }
24 /*-----*/
25 led_bl() /* 工作 LED 闪动 */
26 {
27     /* blink RED led P1.7 */
28     char i;
29
30     for(i=0; i<4; i++)
31     {
32         cplbit(P1.7); /* 位取反 */
33         delay(40);
34     }
35 }
36 /*-----*/
37 char scan_key() /* 键盘扫描控制程序 */
```



```
38 {
39 char i,j, find, ini, inj;
40 char in;
41
42 find=0; /* 清除按键标志 */
43 for(i=0; i<4; i++)
44 {
45 /* 由 PC 端口(PC0~PC3) 送出扫描控制信号 */
46 pc=act[i];
47 delay(3); /* 延迟一会 */
48
49 /* 由 PC 端口(PC4~PC7) 读取按键返回代码 */
50 in=pc;
51 in=in>>4; /* 右移 4 位 */
52 in=in|0xf0; /* 高 4 位设为 1 */
53
54 /* 检查是否按键 ? */
55 for(j=0; j<4; j++)
56 if(act[j]==in)
57 {
58 find=1; /* 设定按键标志 */
59 inj=j; ini=i; /* 记录扫描指针值 */
60 }
61 } /* scan 1 time */
62
63 if(find==0) return 0; /* 没按键传回 0 值 */
64
65 /* i, j --> key : 0~15 */
66 key=ini*4+inj; /* 计算按键值 */
67 return 1; /* 有按键传回键值 */
68 }
69 /*-----*/
70 main() /* 主程序 */
71 {
72 led_bl(); /* 工作 LED 闪动 */
73 cr=CWORD; /* 设定 8255 工作模式 */
```

```

74  serinit(9600); /* 初始化串行接口 */
75  printf(title); /* 显示工作消息 */
76  loop();        /* 测试循环 */
77  }
78  /*-----*/
79  loop() /* 循环测试程序 */
80  {
81  char c;
82
83  while(1)/* 无穷循环 */
84  {
85  if(scan_key()==1)/* 若有按键则处理 */
86  {
87      /* 按键值转换 */
88      c=skey[key];
89      /*显示 '1'或 '2' 按键是否按下*/
90      if(c=='1')    printf("key 1 pressed ");
91      if(c=='2')    printf("key 2 pressed ");
92  }
93  /* DO any thing .....*/
94  }
95  }
96  /*-----*/

```

KEY.C 程序执行后，一旦按下“1”或“2”按键则显示出按键的结果，但是只按下一次，却显示了按下好几次的消息，这是由于按键扫描程序快于扫描的执行结果（在一次按键的时间内，程序已经进行了数次扫描），下面的程序便可以解决此问题。

执行结果

程序执行后，控制程序不断做键盘扫描，当有按键按下，必需等按键放开时才由 PC 端显示出按键的结果。目前程序只显示“1”或“2”按键是否按下，其他的键则未处理。

程序清单

```

1  /* KEY1.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"

```

```
6 #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
7
8 #define CWORD 0x88 /* PA PB PC0~3 o/p PC4~7 i/p */
9 char *title="Test P51_PCB keys ...";
10 char act[4]={0xfe, 0xfd, 0xfb, 0xf7} /* 扫描控制信号 */
11 char key; /* 键盘扫描值 */ /* 16 个键盘按键 */
12 char skey[16]={'A','B','C','D', '3','6','9','#',
13              '2','5','8','0', '1','4','7','*'};
14 char scan_key(); /* 键盘扫描控制程序 */
15 /*-----*/
16 delay(int t) /* 延迟子程序 */
17 {
18     int i,j;
19     for(i=0; i<t; i++)
20         for(j=0; j<10; j++);
21 }
22 /*-----*/
23 led_bl() /* 工作 LED 闪动 */
24 {
25     /* blink RED led P1.7 */
26     char i;
27
28     for(i=0; i<4; i++)
29     {
30         cplbit(P1.7); /* 位取反 */
31         delay(40);
32     }
33 }
34 /*-----*/
35 char scan_key() /* 键盘扫描控制程序 */
36 {
37     char i,j, find, ini, inj;
38     char in;
39
40     find=0; /* 清除按键标志 */
41     for(i=0; i<4; i++)
```

```
42 {
43 /* 由 PC 端口(PC0~PC3) 送出扫描控制信号 */
44     pc=act[i];
45     delay(3);
46
47 /* 由 PC 端口(PC4~PC7) 读取按键返回代码 */
48     in=pc;
49     in=in>>4; /* 右移 4 位 */
50     in=in | 0xf0; /* 高 4 位设为 1 */
51
52 /* 检查是否按键 ? */
53     for(j=0; j<4; j++)
54         if(act[j]==in)
55             {
56                 find=1; /* 设定按键标志 */
57                 inj=j; ini=i; /* 记录扫描指针值 */
58             }
59 /* scan 1 time */
60
61     if(find==0) return 0; /* 没按键传回 0 值 */
62
63 /* i,j ----> key : 0~15 */
64     key=ini*4+inj; /* 计算按键值 */
65     return 1; /* 有按键传回键值 */
66 }
67 /*-----*/
68 main() /* 主程序 */
69 {
70     led_bl(); /* 工作 LED 闪动 */
71     cr=CWORD; /* 设定 8255 工作模式 */
72     serinit(9600); /* 初始化串行接口 */
73     printf(title); /* 显示工作消息 */
74     loop(); /* 测试循环 */
75 }
76 /*-----*/
77 loop() /* 循环测试程序 */
```

```

78 {
79 char c;
80 while(1)/* 无穷循环 */
81 {
82 /* 等待按键 .....*/
83 get_key();
84 c=skey[key];/* 按键值转换 */
85 /* 显示 '1'或 '2' 按键是否按下*/
86 if(c=='1') printf("key 1 pressed ");
87 if(c=='2') printf("key 2 pressed ");
88 }
89 }
90 /*-----*/
91 get_key() /* 等待按键并等按键放开来*/
92 {
93 while(1)/* 无穷循环 */
94 if(scan_key()==1) break; /* 若有按键则跳出循环 */
95 while(1)/* 无穷循环 */
96 if(scan_key()==0) break; /* 等按键放开来 */
97 }
98 /*-----*/

```

7.6 键盘扫描及 7 段数码管控制

前面分别看过以扫描方法控制 7 段数码管，及以扫描法做键盘检查，本节实验将二者合并，使用 8255 端口 C 8 条 I/O 线做 16 个按键的键盘扫描，由 PC0~PC.3 送出扫描信号，并且可以作为显示器使能位，一次只送出一个位为 0 的值，由 PC.4~PC.7 读取按键数据返回代码，由端口 B 送出共阴极 7 段数码管数据。

执行结果

程序执行后工作指示灯闪烁一下，四位 7 段数码管显示“1234”，如果按下某一按键，则由 PC 端显示出按键的结果。目前程序只显示“1”或“2”按键是否按下，其他的键则未处理。

程序清单

```

1 /* SEGKEY.C */
2

```

```

3  #include "8051io.h" /* 载入 MC51 头文件 */
4  #include "8051reg.h"
5  #include "8051bit.h"
6  #include "8051int.h"
7  #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
8
9  #define CWORD 0x88 /* PA PB PC0~3 o/p PC4~7 i/p */
10 char *title="Test P51_PCB keys and 7seg X4 ...";
11 /* 0~9 的字符数据 */
12 char DATA_7SEG[10]={0x3f, 0x06, 0x5b, 0x4f, 0x66,
13 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
14 char act[4]={0xfe, 0xfd, 0xfb, 0xf7} /* 扫描控制信号 */
15 char disp[4]; /* 四位数码管数据缓冲区 */
16 char key; /* 键盘扫描值 */ /* 16 个键盘按键 */
17 char skey[16]={'A', 'B', 'C', 'D', '3', '6', '9', '#',
18 '2', '5', '8', '0', '1', '4', '7', '*'};
19 char scan_key(); /* 键盘扫描控制程序 */
20 /*-----*/
21 delay(int t) /* 延迟子程序 */
22 {
23 int i,j;
24 for(i=0; i<t; i++)
25 for(j=0; j<10; j++);
26 }
27 /*-----*/
28 led_bl() /* 工作 LED 闪动 */
29 {
30 /* blink RED led P1.7 */
31 char i;
32
33 for(i=0; i<4; i++)
34 {
35 cplbit(P1.7); /* 位取反 */
36 delay(40);
37 }
38 }

```

```
39  /*-----*/
40  init_buf() /* 载入显示器初值 */
41  {
42      int i;
43      for(i=0; i<4; i++)
44          disp[i]=DATA_7SEG[i];
45  }
46  /*-----*/
47  char scan_key() /* 键盘扫描控制程序 */
48  {
49      char i,j, find, ini, inj;
50      char in;
51
52      find=0; /* 清除按键标志 */
53      for(i=0; i<4; i++)
54      {
55          /* 由 PB 端口(PB0~PB7) 送出字型数据 */
56          pb=disp[i];
57          /* 由 PC 端口(PC0~PC3) 送出扫描控制信号 */
58          pc=act[i];
59          delay(3);
60
61          /* 由 PC 端口(PC4~PC7) 读取按键返回代码 */
62          in=pc;
63          in=in>>4; /* 右移 4 位 */
64          in=in|0xf0; /* 高 4 位设为 1 */
65
66          /* 检查是否按键 ? */
67          for(j=0; j<4; j++)
68              if(act[j]==in)
69              {
70                  find=1; /* 设定按键标志 */
71                  inj=j; ini=i; /* 记录扫描指针值 */
72              }
73      } /* scan 1 time */
74
```

```
75  if(find==0) return 0; /* 没按键传回 0 值 */
76
77  /* i,j ---> key : 0~15 */
78  key=ini*4+inj; /* 计算按键值 */
79  return 1; /* 有按键传回键值 */
80 }
81 /*-----*/
82 main() /* 主程序 */
83 {
84  led_bl(); /* 工作 LED 闪动 */
85  cr=CWORD; /* 设定 8255 工作模式 */
86  init_buf(); /* 载入显示器初值 */
87
88  serinit(9600); /* 初始化串行接口 */
89  printf(title); /* 显示工作消息 */
90  /* 测试循环 */
91  loop();
92 }
93 /*-----*/
94 loop() /* 循环测试程序 */
95 {
96  char c;
97
98  while(1) /* 无穷循环 */
99  {
100  if(scan_key()==1) /* 若有按键则处理 */
101  {
102      /* 显示 '1'或 '2' 按键是否按下*/
103      c=skey[key]; /* 按键值转换 */
104      if(c=='1') printf("key 1 pressed ");
105      if(c=='2') printf("key 2 pressed ");
106  }
107  /* DO anything .....*/
108
109  }
110 }
```



```
111 /*-----*/
```

7.7 蜂鸣器控制

在单板上 8051 端口 1 位 0 (P1.0) 是设计做蜂鸣器的驱动位, 持续送出工作脉冲可以驱动蜂鸣器发出“嘀”的声响, 当工作频率越高时, 声音越清脆, 工作频率越低时, 声音则较低沉, 工作频率太低无法使蜂鸣器发出声音。一般的蜂鸣器工作频率为数百至数千赫兹。

在实验时需先检查单板上 P1.0 的控制位设定, 要将 JP7 及 JP18 的短路器设定, 才能听见蜂鸣器发出的声音。

执行结果

程序执行后工作指示灯闪烁一下, 蜂鸣器会连续发出“嘀”声 10 次, 然后工作指示灯会持续闪烁下去。

程序清单

```
1  /*  BE.C  */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  char *title="Test P51_PCB BEEPER";
9  /*-----*/
10 delay(int t) /* 延迟子程序 */
11 {
12     int ij;
13     for(i=0; i<t; i++)
14         for(j=0; j<10; j++);
15 }
16 /*-----*/
17 led_bl() /* 工作 LED 闪动 */
18 {
19     /* blink RED led P1.7 */
20     char i;
21
```

```
22  for(i=0; i<4; i++)
23  {
24      cplbit(P1.7); /* 位取反 */
25      delay(40);
26  }
27 }
28 /*-----*/
29 main() /* 主程序 */
30 {
31     be();
32     led_bl(); /* 工作 LED 闪动 */
33     serinit(9600); /* 初始化串行接口 */
34     printf(title); /* 显示工作消息 */
35     test_beep(); /* 蜂鸣器测试 */
36 }
37 /*-----*/
38 be() /* 蜂鸣器“嘀”一声 */
39 {
40     char i;
41
42     for(i=0; i<100; i++)
43     {
44         cplbit(P1.0); /* 位取反 */
45         delay(1);
46     }
47 }
48 /*-----*/
49 test_beep() /* 蜂鸣器测试程序 */
50 {
51     char i;
52
53     delay(300);
54     be();
55     delay(1000);
56     for(i=0; i<10; i++) be(); /* 蜂鸣器“嘀”10 声*/
57     while(1) led_bl(); /* 工作指示灯持续闪烁*/
```

```
58 }  
59 /*-----*/
```

7.8 习 题

1. 以示波器来验证 WLED.C 程序中 LED 亮灭的间距多久。
2. 以示波器来验证 LED.C 程序中 delay(1)延迟多久的时间。
3. 说明 8 个 LED 做出那 4 种不一样“走马灯”式电路展示花样。
4. 另外再设计 4 种不一样“走马灯”式电路展示花样。
6. 修改原始程序使 7 段数码管显示字符“ABCD”。
7. 修改控制程序，使 7 段数码管产生闪烁的效果。

第 8 章 中 断 控 制

8051 中断控制功能除有两组定时计数器、串行传输接口外，还有外部的中断控制 INT1。定时工作可以让控制程序具有多工的结构，可以每隔一段时间定时去执行中断程序。串行传输接口提供单片机与其他控制系统通信的能力，在除错及连线控制上用途很多。外部的中断控制提供系统紧急事件的输入控制。本章先介绍外部中断的控制方式并举实例来做说明，接着在第 9 章中讲述定时计数器使用说明，在第 10 章中再介绍串行传输接口的应用。

8.1 I/O 控制的方式

在正式介绍 8051 中断控制结构前，我们先谈一下传统微处理机 I/O 控制的方式，较复杂的单片机系统在 I/O 控制方面，提供有中断处理控制方式及直接内存(DMA)存取方式。基本上，一套单片机系统在 I/O 功能控制上有三种基本方式：

- 询问式；
- 中断控制式；
- DMA 处理。

8.1.1 询问式

CPU 自动定时去检查各个 I/O 外围设备，是否有任何设备需要服务，当检查到有触发工作信号产生时，则执行相应的服务程序，因此纯靠软件程序浪费 CPU 时间在特定的 I/O 线上查询各种事先设定的状态是否产生，效率不高，但程序设计起来却比较容易，这是一般控制软件程序经常使用的方法。

8.1.2 中断控制式

当外围设备需要 CPU 服务时才发生中断请求信号，待 CPU 将目前的指令执行完毕后，便至中断向量处执行中断服务程序，这种控制方式 CPU 在平时可以有充分的时间去处理主要的事件，一旦外界有突发状况时也不会错过，因此效率较高。此时是由 I/O 外围设备自动送出中断请求信号，要求 CPU 来做处理。

8.1.3 DMA 处理

外围设备若要在内存进行数据的存取时，一般可以通过 CPU 来完成数据的转移，当传输的数据量很大时则采用直接内存存取方式来处理效率较高。当外界 I/O 需要服务时先送出 DMA 请求信号，待系统确认后响应认可信号便不须通过 CPU 的读写过程，直接做外围设备与内存之间的数据存取。

一个实际例子

在 PC 上谈到语音录音放音及声音播放时, 很多人一定会想到声霸卡, 那么 PC 是以怎样的方式来控制声霸卡工作的呢? 在使用说明书上您可以看到其所使用的 DMA 为 DRQ1, 而中断请求信号可使用 IRQ7, I/O 地址为 220H, 因此声霸卡的实际操作便涵盖了以上的 3 种 I/O 工作方式, 适时产生中断信号及 DMA 工作请求信号来与 PC 做数据的存取。有关中断控制及 DMA 工作情况, 读者若有兴趣, 请自行查阅市面上相关 PC 接口控制的书籍。

8.2 8051 中断控制结构

8051 内含有两个外部中断, 两个计时计数器中断及一个串行端口中断, 下表是其中断程序执行的向量表及中断控制标志列表:

中断源	工作标志	向量地址
外部中断 0	IE0	03H
计时器 0	TF0	0BH
外部中断 1	IE1	13H
计时器 1	TF1	1BH
串行端口传送	TI	23H
串行端口接收	RI	23H

其中串行端口传送及串行端口接收共用一个向量地址。以上所列出的工作标志是存在特殊功能寄存器 TCON 中的各个位, 当某种中断源产生中断时便将设定相对的中断工作标志, 在程序中只要判断这些标志便可以知道产生了那一种中断, 而向量地址是 8051 程序内存中最前面的几个特殊的地址, 用来决定各种中断服务程序的程序进入点地址, 当 8051 产生了中断工作后, 便会跳到某一固定的地址去执行中断服务程序。

• 外部中断:

外部中断信号由 INT0 及 INT1 引脚传送进来, 可以分为两种控制方式, 即电位触发式及下降沿触发式, 其设定方式是由 TCON 寄存器中的 IT0 及 IT1 两个位来决定, IT0 控制 INT0 的触发方式, 当 IT0=1 时为下降沿触发式, 当 IT0=0 时为电位触发式, IT1 则控制 INT1 的触发方式, 工作情况如同 IT0。

电位触发式中断是当 INT0 引脚呈现低电位时, IE0 工作标志设定, 若 INT0 引脚呈现高电位时则 IE0 工作标志清除。下降沿触发式中断是当 INT0 引脚由 1 变化到 0 的瞬间产生中断, 而设定工作标志 IE0, 此工作标志会一直保留着, 直到执行过中断服务程序后才会清除。

• 计时计数器中断:

8051 内部的计时计数器一旦设定初值并启动后, 会以一定速率做往上计数的工作,

当计数器值溢出后，便会产生计时计数器中断，而将设定相对的工作标志，计时器 0 设定为 TF0，计时器 1 设定为 TF1，若执行过相对的中断服务程序后才会将工作标志清除。

- 串行端口中断：

8051 处理串行端口的中断可分为传送及接收数据，当发送器将串行缓冲器中的数据传送出去后，便将设定 TI，而当接收器收到完整的一字节数据，并将数据放入串行缓冲器后，也会设定 RI 标志，在执行中断服务程序时，并不会自动将工作标志清除，通常在程序中必需加以判断此串行端口中断是由 TI 或是由 RI 产生的，而分别执行不同的控制程序，并将工作标志加以清除。

8.3 相关控制寄存器

有关中断处理的相关控制寄存器如下：

- 计时计数器控制寄存器 TCON；
- 中断允许控制寄存器 IE；
- 中断优先权控制寄存器 IP。

8.3.1 TCON：计时控制寄存器

可位寻址，地址 88H。用来记录各个中断源所产生的工作标志，并包含计时器启动控制位，各个位说明如下：

B7	B6	B5	B4	B3	B2	B1	B0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 (TCON.7)：计时器 1 溢出标志，当计时溢出时，由硬件设定为 1，在执行过相对的中断服务程序后则自动清除为 0。

TR1 (TCON.6)：计时器 1 启动控制位，可以由软件来设定或清除。TR1=1 时启动计时器工作，TR1=0 时关闭。

TF0 (TCON.5)：计时器 0 溢出标志，当计时溢出时，由硬件设定为 1，在执行过相对的中断服务程序后则自动清除为 0。

TR0 (TCON.4)：计时器 0 启动控制位，可以由软件来设定或清除。TR0=1 时，启动计时器工作，TR0=0 时关闭。

IE1 (TCON.3)：外部中断 1 工作标志，当外部中断被检查出来时，硬件自动设定此位，在执行过中断服务程序后，则消除为 0。

IT1 (TCON.2)：外部中断 1 工作形式选择，IT1=1 时，由下降沿产生外部中断；IT1=0 时，由低电位产生中断。

IE0 (TCON.1)：外部中断 0 工作标志，当外部中断被检查出来时，硬件自动设定此位，在执行过中断服务程序后，则清除为 0。

IT0 (TCON.0) : 外部中断 0 工作形式选择, IT0=1 时为下降沿产生外部中断, IT0=0 时为低电位产生中断。

8.3.2 IE: 中断允许寄存器

可位寻址, 地址 A8H, 用来允许各种中断信号的产生, 各个位说明如下:

B7	B6	B5	B4	B3	B2	B1	B0
EA	—	ET2	ES	ET1	EX1	ET0	EX0

- EA (IE.7) : EA=0 时, 所有中断停用 (中断不产生)。
EA=1 时, 各中断的产生由个别的启动位决定。
- (IE.6) : 保留。
- ET2 (IE.5) : 允许定时器 2 溢出的中断 (8052 使用)。
- ES (IE.4) : 允许串行端口的中断 (ES=1 启用, ES=0 停用)。
- ET1 (IE.3) : 允许定时器 1 中断。
- EX1 (IE.2) : 允许外部中断 INT1 的中断。
- ET0 (IE.1) : 允许定时器 0 中断。
- EX0 (IE.0) : 允许外部中断 INT0 的中断。

8.3.3 IP: 中断优先权寄存器

可位寻址, 地址 B8H, 用来设定各种中断信号产生的优先次序, 8051 中断源外部中断 INT0 具有最高的优先控制权, 当然在适当的时候可以由设计者加以规划此寄存器, 使得相对的中断源具有最高的优先中断权。各个位说明如下:

B7	B6	B5	B4	B3	B2	B1	B0
—	—	PT2	PS	PT1	PX1	PT0	PX0

- (IP.7) : 保留。
- (IP.6) : 保留。
- PT2 (IP.5) : 设定定时器 2 的优先次序 (8052 使用)。
- PS (IP.4) : 设定串行端口的中断优先顺序。
- PT1 (IP.3) : 设定定时器 1 的优先顺序。
- PX1 (IP.2) : 设定外部中断 INT1 的优先顺序。
- PT0 (IP.1) : 设定定时器 0 的优先顺序。
- PX0 (IP.0) : 设定外部中断 INT0 的优先顺序。

8.4 8051 C 语言中断程序的写法

在 8.2 节中看过 8051 中断控制的程序执行的向量表, 及中断控制标志列表, 当 8051

产生了中断工作后，会跳到某一固定的地址去执行中断服务程序，而中断服务程序的地址可以由程序来决定的。在 MC51 中可以使用 C 语言的程序代码直接来做中断处理，中断服务程序也如同一般子程序，区别是其开头必需以“INTERRUPT”来定义。例如：

```
INTERRUPT(_SER_) serial_isr()
{
    /* ... 处理串行端口中断的程序代码 ... */
}
```

其中 serial_isr() 名称可以自取。

同理：

```
INTERRUPT(_TF0_) t0_isr()
{
    /*... 处理计时计数器 0 中断的程序代码 ... */
}
```

```
INTERRUPT(_IE0_) int0_isr()
{
    /* ... 处理 int0 外部中断的程序代码... */
}
```

只要在程序执行后先对相关外部中断寄存器加以设定，如计时计数器控制寄存器(TCON)、中断允许控制寄存器(IE) 及中断优先权控制寄存器(IP)，以后一旦某种中断源产生中断时，便会自动去执行事先设定的各种中断服务程序控制代码了。

8.5 外部中断控制实验 1

看过相关中断控制的说明后，本节将编写程序来做外部中断 INT0 的控制实验，如果以 P51 控制板来做实验，可以直接拿单心线一端插入 J22 圆孔 IC 座上，另一端连接地线，以便产生中断信号，并注意程序执行的变化，借以验证外部中断的工作。

执行结果

程序执行后进入主控循环，蜂鸣器会每隔一段时间嘀一声(由 P1.0 位驱动)，一旦将 INT0 引脚接地后用以产生中断信号，程序自动跳出循环去执行外部中断 0 中断服务程序，而驱动工作 LED 灯闪动 4 下，中断服务程序执行完后，则回到原来离开的主程序循环，继续驱动蜂鸣器产生嘀声。

程序清单

```
1  /*  IN1.C  */
2  #include "8051io.h"    /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "c: p51.h"    /* 载入 P51 I/O 控制头文件 */
7
8  char *title="Test P51_PCB int0";
9  /*-----*/
10 delay(int t) /* 延迟子程序 */
11 {
12     int i,j;
13     for(i=0; i<t; i++)
14         for(j=0; j<10; j++);
15 }
16 /*-----*/
17 led_bl()      /* 工作 LED 闪动 */
18 {
19     char i;
20     for(i=0; i<4; i++)
21     {
22         cplbit(P1.7);
23         delay(40);
24     }
25 }
26 /*-----*/
27 INTERRUPT(_IE0_) int0_isr()    /* 外部中断 0 中断程序 */
28 {
29     char i;
30     for(i=0; i<2; i++)
31         led_bl();
32 }
33 /*-----*/
34 init_int0() /* 相关外部中断寄存器设定 */
35 {
```

```
36 setbit(TCON.0); /* IT0=1 设定 INT0 下降沿触发产生中断 */
37 setbit(IP.0);   /* PX0=1 设定外部中断 INT0 的优先顺序 */
38 setbit(IE.0);   /* EX0=1 允许外部中断 INT0 的中断 */
39 clrbit(TCON.1); /* IE0=0 清除外部中断 0 工作标志 */
40 setbit(IE.7);   /* EA=1 允许中断*/
41 }
42 /*-----*/
43 be() /* 蜂鸣器 “嘀” 一声 */
44 {
45     char i;
46
47     for(i=0; i<100; i++)
48     {
49         cplbit(P1.0);
50         delay(1);
51     }
52 }
53 /*-----*/
54 main()
55 {
56     led_bl(); /* 工作 LED 闪动 */
57     serinit(9600); /* 初始化串行端口 */
58     printf(title); /* 显示工作消息 */
59
60     init_int0(); /* 初始化外部中断 */
61     while(1) /* 无穷循环 */
62     {
63         be(); /* “嘀” 一声 */
64         delay(500);
65     }
66 }
67 /*-----*/
```

8.6 外部中断控制实验 2

本实验是上一节实验的功能扩充，利用 INT0 及 INT1 分别来产生 2 个外部中断信号

借以观察个别中断程序执行的结果, 同样的以 P51 控制板来做实验的话, 可以直接以 J22 圆孔 IC 座来做实验, 一端连接地, 另一端插入 INT0 或 INT1 接孔, 以便产生中断信号, 并注意程序执行的变化, 借以验证 2 个外部中断的工作。

执行结果

程序执行后进入主控循环, 蜂鸣器会每隔一段时间“嘀”一声, 一旦将 INT0 引脚接地后用以产生中断信号, 程序自动跳出循环去执行外部中断 0 中断服务程序, 而驱动工作 LED 灯闪动 4 下, 中断服务程序执行完后, 则回到原来离开的主程序循环, 继续驱动蜂鸣器产生“嘀”声。若将 INT1 引脚接地后, 程序自动跳出循环去执行外部中断 1 中断服务程序驱动工作 LED 灯闪动 8 下, 而后继续驱动蜂鸣器产生“嘀”声。

程序清单

```

1  /*  IN2.C  */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  char *title="Test P51_PCB int0  int1";
9  /*-----*/
10 delay(int t) /* 延迟子程序 */
11 {
12     int i,j;
13     for(i=0; i<t; i++)
14         for(j=0; j<10; j++);
15 }
16 /*-----*/
17 led_bl() /* 工作 LED 闪动 */
18 {
19     char i;
20     for(i=0; i<4; i++)
21     {
22         cplbit(P1.7);
23         delay(40);
24     }
25 }

```

```
26  /*-----*/
27  INTERRUPT(_IE0_) int0_isr() /* 外部中断 0 中断程序 */
28  {
29      char i;
30      for(i=0; i<2; i++)
31          led_bl();
32  }
33  /*-----*/
34  INTERRUPT(_IE1_) int1_isr() /* 外部中断 1 中断程序 */
35  {
36      char i;
37      for(i=0; i<4; i++)
38          led_bl();
39  }
40  /*-----*/
41  init_int() /* 相关外部中断寄存器设定 */
42  {
43      setbit(TCON.0); /* IT0=1 设定 INT0 下降沿触发产生中断 */
44      setbit(TCON.2); /* IT1=1 设定 INT1 下降沿触发产生中断 */
45
46      setbit(IP.0); /* PX0=1 设定外部中断 INT0 的优先顺序 */
47      setbit(IP.2); /* PX1=1 设定外部中断 INT1 的优先顺序 */
48
49      setbit(IE.0); /* EX0=1 允许外部中断 INT0 的中断 */
50      setbit(IE.2); /* EX1=1 允许外部中断 INT1 的中断 */
51
52      clrbit(TCON.1); /* IE0=0 清除外部中断 0 工作标志 */
53      clrbit(TCON.3); /* IE1=0 清除外部中断 1 工作标志 */
54
55      setbit(IE.7); /* EA=1 中断允许 */
56  }
57  /*-----*/
58  be() /* 蜂鸣器“嘀”一声 */
59  {
60      char i;
61
```

```
62   for(i=0; i<100; i++)
63   {
64       cplbit(P1.0);
65       delay(1);
66   }
67 }
68 /*-----*/
69 main()
70 {
71     led_bl();      /* 工作 LED 闪动 */
72     serinit(9600); /* 初始化串行端口 */
73     printf(title); /* 显示工作消息 */
74
75     init_int(); /* 初始化外部中断 */
76     while(1) /* 无穷循环 */
77     {
78         be(); /* “嘀”一声 */
79         delay(500);
80     }
81 }
82 /*-----*/
```

8.7 习 题

1. 说明一般单片机系统在 I/O 功能控制上有哪三种基本方式。
2. 举例说明 I/O 的中断控制。
3. 何谓 I/O 功能控制的 DMA 处理？
4. 8051 有哪 5 个中断源？并列举其中断控制向量地址。
5. 程序 IN1.C 中，INT0 为以下降沿触发方式产生中断，请修改程序为电位触发式中
断，并比较其执行结果。

第 9 章 8051 计时计数器

8051 内含有 2 个 16 位的计时计数器，称为计数器 0 及计数器 1，如同一般计时计数器的功能，其主要作用有两点：第一，做一段特定时间长短的计时。第二，可以计算由 T0 或 T1 引脚输入的脉冲数。前者在应用上可以产生正确的时间延迟及定时去执行中断服务程序，这是单片机在软件控制程序上常用到的技巧，而后者的应用则是计数器或是频率计的设计。

这两个计数器本身都有 4 种工作模式可供使用：

模式 0：13 位计时工作模式。

模式 1：16 位计时工作模式。

模式 2：具有自动重新载入计数值的 8 位计时工作。

模式 3：在此模式工作时，计数器 1 本身停止计时的工作，而计数器 0 分为两个独立的 8 位计数器由 TL0 及 TH0 来负责计时的任务。

模式 0 到模式 2 中，计数器 0 和计数器 1 的使用方法都一样，只有在模式 3 时才不同。在本章中，我们分别来介绍其各种计时计数器的工作模式，并举例来说明如何利用计数器来做各种控制程序的设计。

9.1 计时计数器相关控制寄存器

8051 内部特殊功能寄存器中与计时计数器相关的控制寄存器，有以下几个：

名称	地址	功能
TCON	88H	计数器控制寄存器
TMOD	89H	计数器工作模式选择寄存器
TH0	8CH	计数器 0 高 8 位计时寄存器
TL0	8AH	计数器 0 低 8 位计时寄存器
TH1	8DH	计数器 1 高 8 位计时寄存器
TL1	8BH	计数器 1 低 8 位计时寄存器

首先介绍 TCON 寄存器，每个位均可位寻址，可分别设定或清除，适当的控制这些位便可控制计数器的工作。

B7	B6	B5	B4	B3	B2	B1	B0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

- TF1：TCON.7，计数器 1 溢出标志，当计数器计时终止产生溢出时，硬件会自

动设为 1，而在执行过中断服务程序后，硬件会自动清除该位。

- TR1：TCON.6，计数器 1 计时启动位，以软件来设定或清除做启动或停止计数的功能。
- TF0：TCON.5，计数器 0 溢出标志，其功能同 TF1。
- TR0：TCON.4，计数器 0 计数器启动位，其功能同 TR1。
- IE1：TCON.3，外部中断 1（引脚 INT1）的中断设定标志。当 INT1 引脚由外部送入中断信号，而硬件检测到此信号时，会设定此位。在执行过中断服务程序后，硬件会自动清除此位。
- IT1：TCON.2，外部中断 1 的中断信号形态设定，当 IT1=1 时，中断信号为下降沿触发。若 IT1=0 时则为低电位触发。
- IE0：TCON.1，外部中断 0（引脚 INT0）的中断设定标志，当 INT0 引脚由外部送入中断信号，而硬件检测到此信号时将设定此位，在执行完中断服务程序后，硬件自动清除此位。
- IT0：TCON.0，外部中断 0 的中断信号形态设定，当 IT0=1 时，中断信号为下降沿触发。若 IT0=0 时，则为低电位触发。

再来介绍 TMOD，计时模式选择寄存器，此寄存器不可位寻址。

B7	B6	B5	B4	B3	B2	B1	B0
GATE	C/T	M1	M0	GATE	C/T	M1	M0
计数器 1				计数器 0			

- GATE：计数器工作的门控位，当 GATE=0 时，计数器在 TR0=1 或 TR1=1 时会工作。而当 GATE=1 时，且 TR0=1 或 TR1=1，计数器要在 INT0 或 INT1 引脚成为高电位时才会执行计时的工作。
- C/T：计时功能或计数功能的选择位，C/T=0 做计时工作，而 C/T=1 时才做计数的工作。
- M0、M1：工作模式选择

M1	M0	工作模式
0	0	模式 0
0	1	模式 1
1	0	模式 2
1	1	模式 3

9.2 计数器模式 0 的工作

图 9-1 是计数器 0 在模式 0 时的工作逻辑电路图，而计数器 1 工作原理同计数器 0。

在模式 0 时计数器 0 及计数器 1 做 13 位的计时，其内含有两个 8 位的计时寄存器，分别为 TL0 及 TH0（计数器 0），TL1 及 TH1（计数器 1），这两组计时寄存器特性及使用方法都一样，以下我们以计数器 0 为例来做说明。

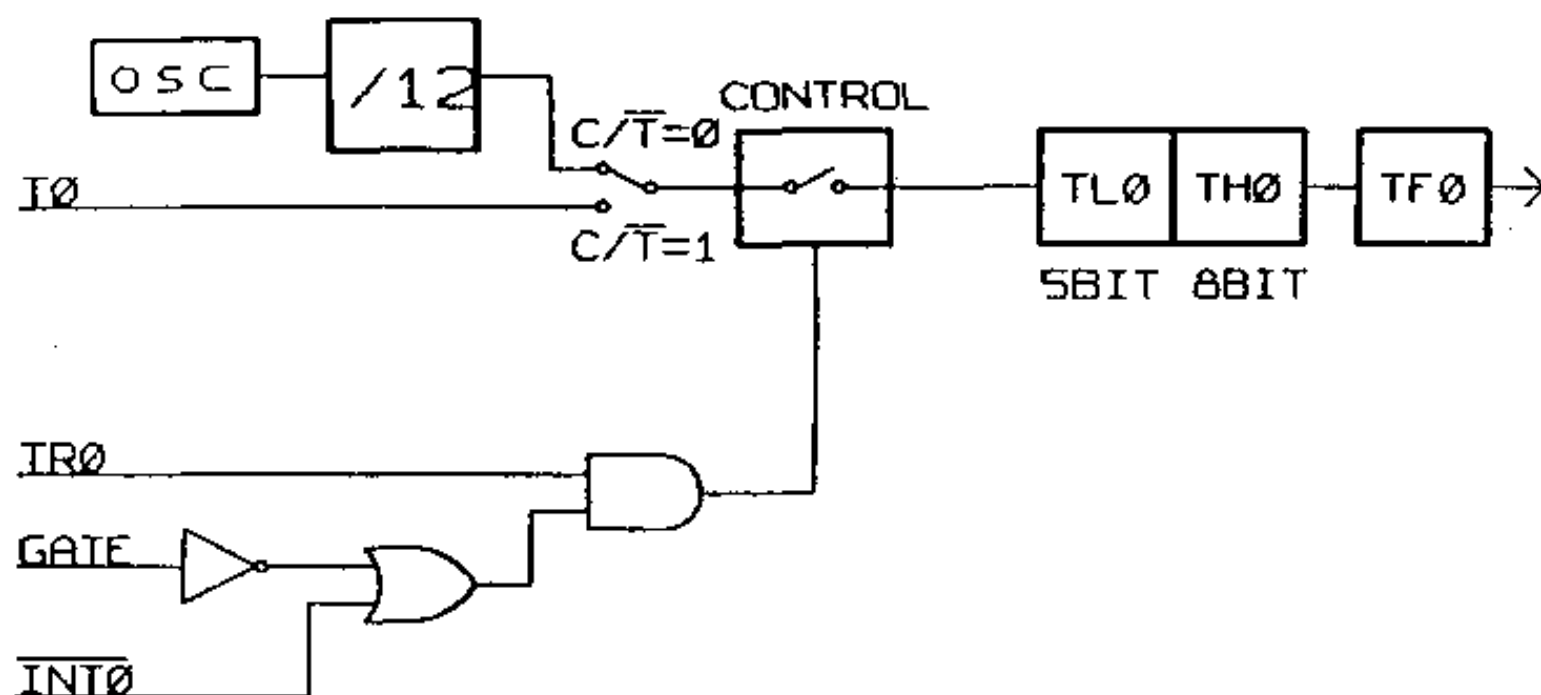


图 9-1 计数器 0 模式 0 工作逻辑电路图

9.2.1 计时工作脉冲

计数器的工作时钟可以由内部或是外部来提供，由 C/T 位（在 TMOD 中）来决定，当 C/T=1 时，由外部引脚 T0 来供给，作为计数器时使用。当 C/T=0 时，则由内部时钟来提供，作为一般的计数器使用。而计数器的时钟为系统工作时钟除以 12，以 8051 单板为例，石英晶振使用 11.0592MHz，所以计数器每一个计数时间脉宽为：

$$12/11.0592 \text{ M} = 1.085 \mu\text{s}$$

若石英晶振改为 12 MHz，计数器每一个计数时间脉冲宽为：

$$12/12 \text{ M} = 1.0 \mu\text{s}$$

计时模式 0 可做 13 位的计数，其值是以下列方式存储在寄存器 TL0 及 TH0 中

TH0	TL0
8bit	5bit

因此最长的计时时间为 $1.085 \mu\text{s} \times 8192 = 8.888\text{ms}$ 。

9.2.2 启动计数器

从图 9-1 中可以看出，计数器工作的必要条件，有以下 2 种：

1. 门控 GATE=0 时，只要 TR0=1，计数器 0 便会工作。
2. 门控 GATE=1 时，除了 TR0=1，外部 INT0 引脚，还必需是高电位才行。

利用条件 2，计数器必需等待外部引脚 INT0 的输入信号变为高电位才开始计时，因

此可以利用来测量输入 INT0 引脚的高电位信号时间多长，这在工厂产品自动化生产测试系统中是一项经常使用的测量技巧。平时在单片机程序控制中，若使用计数器内部时钟工作，C/T=0，GATE=0，在模式 0 工作下，只要下达指令：

```
TMOD=0x00;
```

便可以设定计数器 0 于模式 0 做计时工作。而计数器何时工作呢？将 TR0 (TCON 位 4) 设为 1，计数器便会开始工作了。

9.2.3 计时时间长短设定

8051 计数器是以上数的方式来计数，在模式 0 工作下共可计数 13 位，计数器值最多为 8192 (2 的 13 次方)，由 0 数到 8192 便产生溢出而引发中断信号，产生计时器 0 的中断 (TF0=1)。问题是我们可能只要计数 100 个脉冲便产生中断，只要将初值 8092 (8192-100) 载入计数器便可，一旦启动计数器后，计数变为 8093、8094、……一直到 8192 则产生中断，是不是就计数 100 次了，而时间长度为： $1.085 \mu s \times 100 = 108.5 \mu s$ ，也就是经过 108.5 μs 后便产生中断了。

如何载入计数器的初值呢？以计数器 0 为例，C 语言指令表示如下：

```
TL0=(8192-C)%32;
```

```
TH0=(8192-C)/32;
```

其中，C 为所要计数的值，“%”为取余数的运算，除以 32 后取余数部分。“/”为除法运算，在做完除法后取整数部分。

若计数次数为 100，C=100，可得：

```
TL0=28
```

```
TH0=252
```

9.2.4 计时溢出如何处理

当计数终止产生溢出时，程序如何处理呢？我们怎么知道系统产生计数器中断了呢？计数器应用的方式可以有以下 2 种：

1. 检查其中断控制寄存器 TCON 中的 TF0 及 TF1 位，若为 1 则代表产生计时溢出了。
2. 执行相对的中断服务程序。

一般程序设计中方式 2 较常用，一旦计数终止产生溢出后便引发计时中断，若先前已设定好计时中断服务程序，则会自动跳出去执行计时中断服务程序，在程序中再载入计数初值后重新计数，当计数终止时，又产生计时中断，如此一直循环下去，便可以每隔一段时间自动去执行中断服务程序，这样的设计结构是单片机程序设计中常用的技巧，也是一种简单的程序多工设计方法，以下我们用实际的例子来做实验。

执行结果

先将单板上 J22 P1.0 接点信号接至示波器上，将 JP18 “BEP_ON” 及 JP7 “BEEP”

ON, 程序执行后, 则可以从示波器上看到约 $500\mu\text{s}$ (0.5ms) 宽的方波信号, 并且蜂鸣器发出声响, 而工作 LED 持续闪动着。其中方波信号, 是由计数器 0 中断程序模式 0 定时每隔 $500\mu\text{s}$ 做 P1.0 的位取反来产生。

程序清单

```
1  /*  TM0.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7  char *title="P51_PCB timer0 mode 0 500  $\mu\text{s}$  pulse";
8  int value;
9  unsigned char hi, lo;
10
11 delay(int t) /* 延迟子程序 */
12 {
13     int i,j;
14     for(i=0; i<t; i++)
15         for(j=0; j<5; j++);
16 }
17 /*-----*/
18 led_b1() /* 工作 LED 闪动 */
19 {
20     int i;
21
22     for(i=0; i<4; i++)
23     {
24         cplbit(P1.7);
25         delay(50);
26     }
27 }
28 /*-----*/
29 INTERRUPT(_TF0_) t0isr() /* 计数器 0 中断程序 */
30 {
31     /*  MODE 0 */
32     TH0=hi; /* 载入初值 */
```

```

33  TL0=lo;
34  cplbit(P1.0); /* 位取反送出方波 */
35  }
36  /*-----*/
37  main() /* 主程序 */
38  {
39  led_bl(); /* 工作 LED 闪动 */
40
41  value=8192-500; /* 计数 500 次产生中断 */
42  TH0=hi=value/32; /* 载入初值 */
43  TL0=lo=value%32;
44
45  TMOD=0x00; /* 设定工作模式 */
46  IE=0x82; /* 设定启用状态 */
47  setbit(TCON.4); /* TR0=1; 启动计数器 0 */
48  /* 工作 LED 持续闪动 */
49  while(1) led_bl();
50  }

```

9.3 计数器模式 1 的工作

图 9-2 所示为 8051 计数器 0 模式 1 的工作示意图。

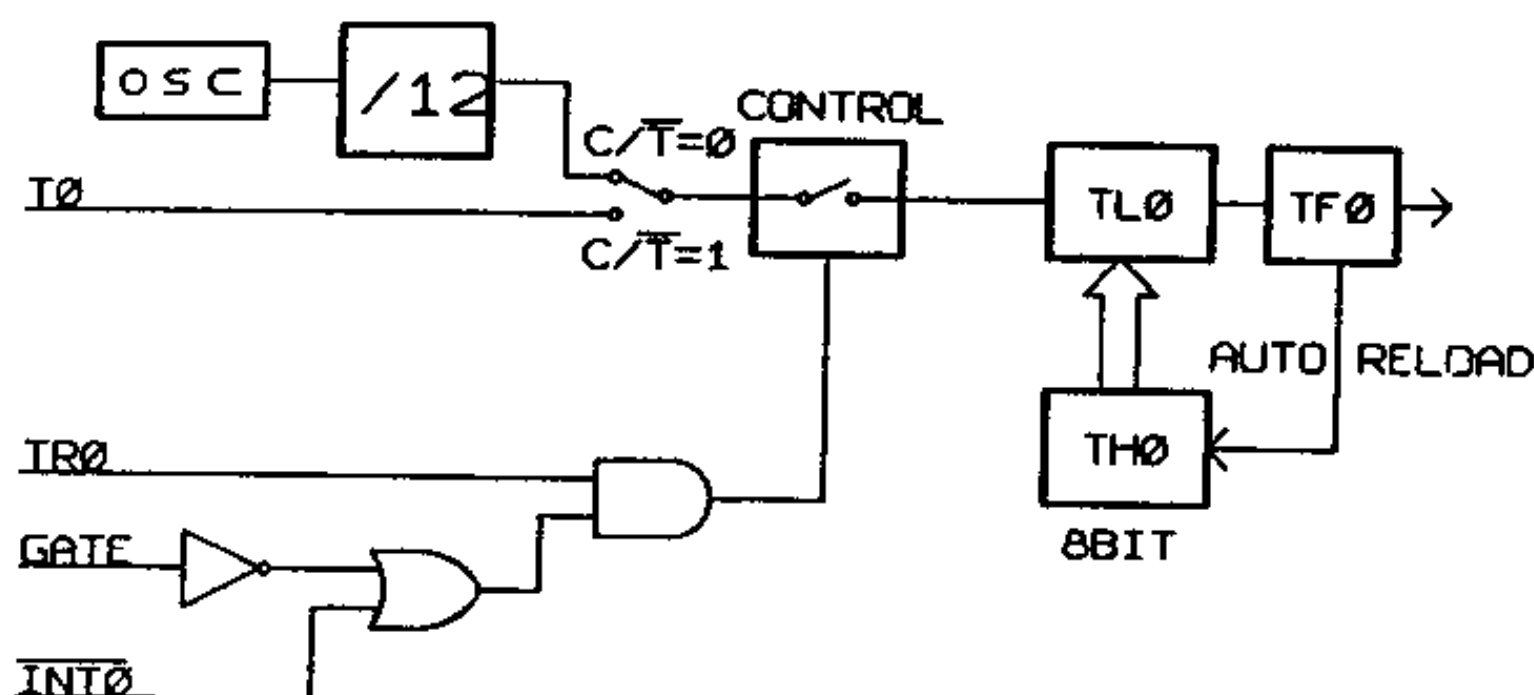


图 9-2 计数器 0 模式 1 工作逻辑电路图

模式 1 的工作同模式 0 的一样，只是它为 16 位的计数器，最大计数脉冲数为 65536 (2

的 16 次方), 计时时间最长为:

$$1.085 \mu s \times 65536 = 72ms$$

而计数初值的载入方法为:

$$TL0 = (65536 - C) \% 256;$$

$$TH0 = (65536 - C) / 256;$$

其中 C 为所要计数的次数, 计数时间的长短为:

$$1.085 \mu s \times C (\mu s)$$

执行结果

先将单板上 J22 P1.0 接点信号接至示波器上, 将 JP18 “BEP_ON” 及 JP7 “BEEP” ON, 程序执行后, 则可以从示波器上看到约 $500 \mu s (0.5ms)$ 宽的方波信号, 并且蜂鸣器发出声响, 而工作 LED 持续闪动着。其中方波信号, 是由计数器 0 中断程序模式 1 定时每隔 $500 \mu s$ 做 P1.0 的位取反来产生。

程序清单

```

1  /*  TM1.C */
2  #include "8051io.h" /* 载入 MCS1 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7  char *title="P51_PCB timer0 mode 1 500 μs pulse";
8  unsigned int value;
9  unsigned char hi, lo;
10
11 delay(int t) /* 延迟子程序 */
12 {
13     int i,j;
14     for(i=0; i<t; i++)
15         for(j=0; j<5; j++);
16 }
17 /*-----*/
18 led_bl() /* 工作 LED 闪动 */
19 {
20     int i;

```

```

21
22  for(i=0; i<4; i++)
23  {
24  cplbit(P1.7);
25  delay(50);
26  }
27 }
28 /*-----*/
29 INTERRUPT(_TF0_) t0isr() /* 计数器 0 中断程序 */
30 {
31  /*  MODE 1 */
32  TH0=hi; /* 载入初值 */
33  TL0=lo;
34  cplbit(P1.0); /*  位取反送出方波 */
35 }
36 /*-----*/
37 main()      /* 主程序 */
38 {
39  led_bl();   /* 工作 LED 闪动 */
40  value=65536-500; /*计数 500 次产生中断*/
41  TH0=hi=value/256; /* 载入初值 */
42  TL0=lo=value%256;
43
44  TMOD=0x01;    /* 设定工作模式 */
45  IE=0x82;      /* 设定允许状态 */
46  /*  TR0=1; 启动计数器 0 */
47  setbit(TCON.4);
48  /* 工作 LED 持续闪动 */
49  while(1) led_bl();
50 }

```

9.4 计数器模式 2 的工作

图 9-3 为计数器 0 模式 2 的工作逻辑电路图，模式 2 的工作与模式 1 的计时工作十分类似，不过它是一个 8 位计数器，在计时结束时，有重新再载入设定值的功能，将原先 16 位的计数器拆成两个 8 位寄存器 TL0 及 TH0，其中 TL0 为真正计数脉冲的计数器，而 TH0

则是放置重新载入值的寄存器，由程序所规划的初值载入只要放入 TH0 中即可。自动重新载入计数的优点是可以让计数器做更精确的计时，一般在模式 0 及模式 1 的工作时必须在中断服务程序中重新载入计数初值，而软件工作必需耗掉执行时间，造成计时的误差，一旦改用由硬件自动载入计数初值时，只要原先设定为模式 2 工作，并启动计数器后，一切计时工作便自动执行，一直循环下去，软件程序就不必要做载入初值的工作，使得在模式 2 下的计时会更准确。由于模式 2 是 8 位的计数器，最大计数脉冲数为 256 (2 的 8 次方)，计时时间比较短，为 $1.085 \mu s \times 256 = 0.28ms$

而计数初值的载入方法为：

$$TH0 = 256 - C;$$

其中 C 为所要计数的次数，计数时间的长短为 $1.085 \mu s \times C \mu s$

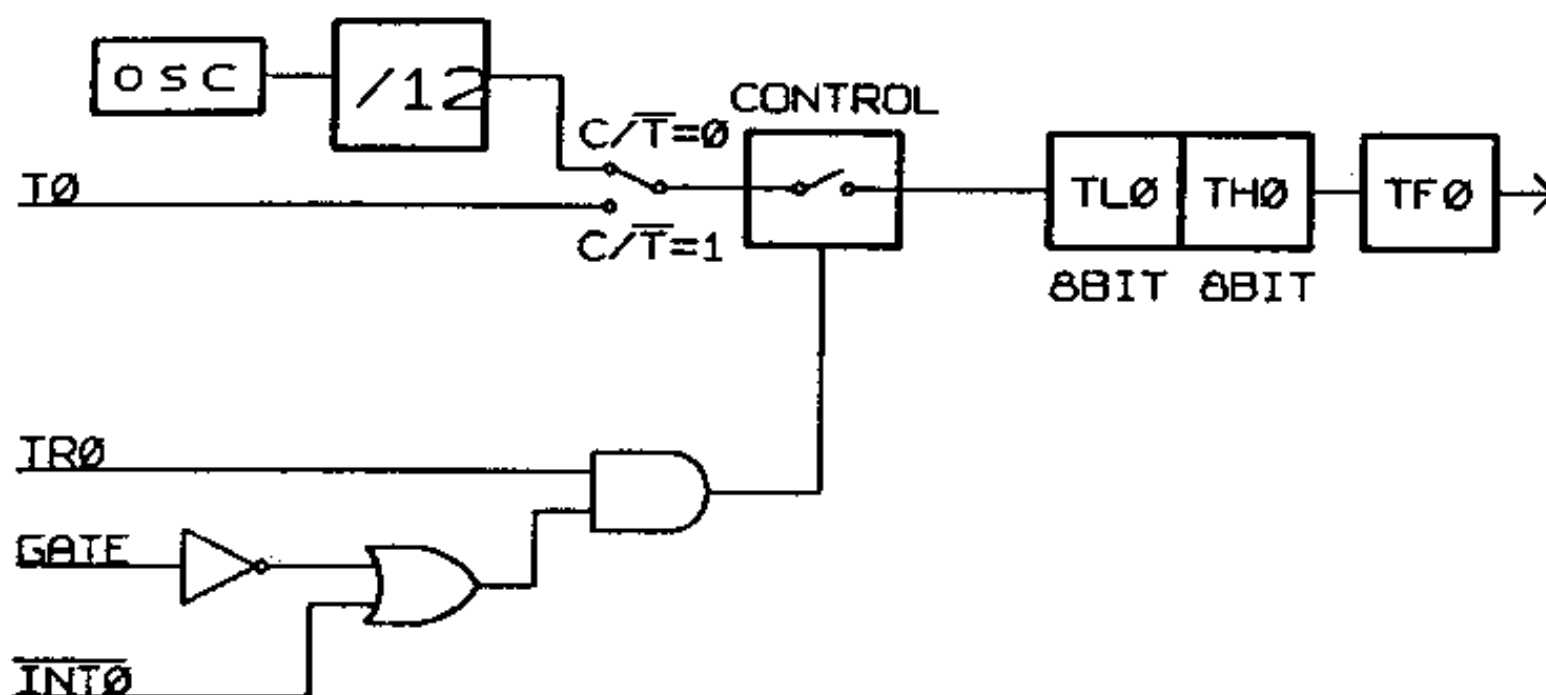


图 9-3 计时器 0 模式 2 工作逻辑电路图

执行结果

先将单板上 J22 P1.0 接点信号接至示波器上，将 JP18 “BEP_ON” 及 JP7 “BEEP” ON，程序执行后，则可以在示波器上看到约 $100 \mu s$ (0.1ms) 宽的方波信号，并且蜂鸣器发出声响，而工作 LED 持续闪动着。其中方波信号，是由计数器 0 中断程序模式 2 定时每隔 $100 \mu s$ 做 P1.0 的位取反来产生。

程序清单

```
1 /* TM2.C */
2 #include "8051io.h" /* 载入 MC51 头文件 */
3 #include "8051reg.h"
4 #include "8051bit.h"
5 #include "8051int.h"
6 #include "p51.h" /* 载入 P51 I/O 控制头文件 */
```

```
7
8 char *title="P51_PCB timer0 mode 2 100  $\mu$ s pulse";
9
10 delay(int t) /* 延迟子程序 */
11 {
12     int i,j;
13     for(i=0; i<t; i++)
14         for(j=0; j<5; j++);
15 }
16 /*-----*/
17 led_bl() /* 工作 LED 闪动 */
18 {
19     int i;
20
21     for(i=0; i<4; i++)
22     {
23         cplbit(P1.7);
24         delay(50);
25     }
26 }
27 /*-----*/
28 INTERRUPT(_TF0_) t0isr() /* 计数器 0 中断程序 */
29 {
30     /* MODE 2 */
31     cplbit(P1.0); /* 位取反送出方波 */
32 }
33 /*-----*/
34 main() /* 主程序 */
35 {
36     led_bl(); /* 工作 LED 闪动 */
37
38     TH0=256-100; /* 计数 100 次产生中断 */
39     TMOD=0x02; /* 设定工作模式 */
40     IE=0x82; /* 设定允许状态 */
41     /* TR0=1; /* 启动计数器 0 */
42     setbit(TCON.4);
```

```

43  /* 工作 LED 持续闪动 */
44  while(1) led_bl();
45  }

```

9.5 计数器模式 3 的工作

图 9-4 为计数器模式 3 的工作逻辑电路图，模式 3 的工作和前面我们所介绍的 3 种模式不太一样，计数器 0 被分为 2 个独立的 8 位计数器，分别由 TL0 及 TH0 来做计数，其中 TL0 使用原先的计数器 0 中断位 TF0、TR0，但 TH0 则占用计数器 1 所使用的 TF1 及 TR1 位，不过计数器 1 仍然可以在模式 0、1、2 下工作，也就是说 8051 的计时器在模式 3 工作时最多可以同时有以下 3 组计数器在工作：

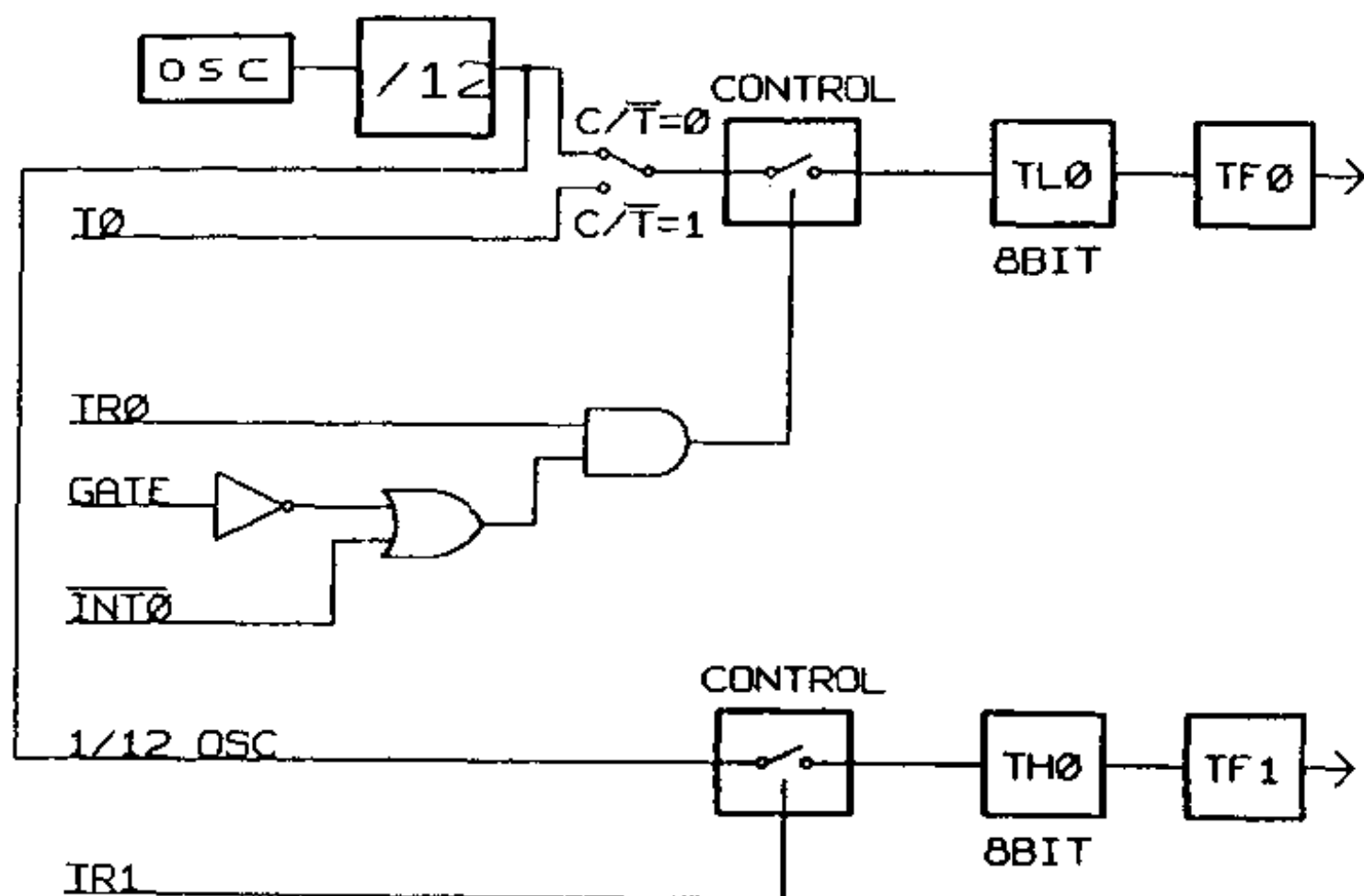


图 9-4 计数器 0 模式 3 工作逻辑电路图

1. 计数器 0 模式 3 TL0，一组 8 位计数器。
2. 计数器 0 模式 3 TH0，一组 8 位计数器。
3. 计数器 1 模式 0（或模式 1 或模式 2），TH1 及 TH0 计数器。

其中第 3 组计数器模式 2 自动载入初值的工作模式，是 INTEL 厂商一直建议使用者设计串行端口时使用的波特率时钟产生方法，因此在建立 8051 串行传输接口通信后，我们还可以利用模式 3 的工作方法获得 2 组分别独立工作的计数器。在此模式下工作的 TL0 及 TH0 载入初值的方法同模式 2 一样。

$TH0 = 256 - C;$

$TL0=256-C;$

其中 C 为所要计数的次数。而最大计数时间为:

$1.085\ \mu s \times 256 = 0.28ms$

9.6 驱动 7 段数码管

在第 7 章中我们谈过 8051 单板上 4 位 7 段数码管的驱动, 是以扫描的方式完成的, 任何时候只要看到显示器上有数据出现, 必需靠程序在循环中扫描送出启用信号, 因此在软件设计上的确造成负担。在我们学过 8051 计数器的应用后, 此问题便可以轻易的解决。可以将扫描 7 段数码管的工作, 放在固定的计时中断服务程序中来做就行了。所设计的中断服务程序如下:

```

INTERRUPT(_TF0_) t0isr()
{
    TH0=HI;
    TL0=LO;

    pb=disp[dpt];
    pc=act[dpt];
    dpt++;
    if(dpt>=4)    dpt=0;
}

```

程序一执行先载入计数初值, 以便下回中断可以继续产生, pb 是 8255 端口 B 的输出端口, 其功能是送出 7 段数码管的字形数据, 目前是存在 disp[i] 的字形缓冲区中。pc 是 8255 端口 C 的输出端口, 其功能是送出 7 段数码管的点亮启用位, 低电位工作。

在点亮了某位数字后必需持续一段时间, 让中断程序每隔 5ms 扫描一次, 缓冲区指针 dpt 每次加一, 若超过 4, 表示 4 位 7 段数码管都已扫描过一次, 于是将指针归 0。中断程序 5ms 执行一次, 那么计数器如何决定初值的载入? 参见上节说明可以模式 0 和模式 1 来设计, 在此以模式 1 计数器 0 为例做说明。

计数器 0 每隔 5ms 执行一次中断服务程序, 那么要计数几次呢?

$C=5000\ \mu s / 1.085\ \mu s = 4608$ 次

于是载入初值决定如下:

$TH0=(65536-4608)/256=238$

$TL0=(65536-4608)\%256=0$

程序 TSEG.C 是一个以计数器 0 驱动 4 位 7 段数码管的例子。

执行结果

程序执行后, 7 段数码管显示“1234”, 而工作 LED 持续闪动着。

程序清单

```

1  /*  TSEG.C  */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  #define HI 238 /* mode 1 5ms 初值设定 */
9  #define LO 0
10
11 #define CWORD 0x88 /* PA PB PC0~3 o/p PC4~7 i/p */
12 char *title="P51_PCB test 7seg X4 by timer0 ...";
13 /*0~9 的字符数据 */
14 char DATA_7SEG[10]={0x3f, 0x06, 0x5b, 0x4f, 0x66,
15 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
16 char act[4]={0xfe, 0xfd, 0xfb, 0xf7} /* 扫描控制信号 */
17 char disp[4]; /* 四位数码管数据缓冲区 */
18 char dpt=0; /* 数据缓冲区指针 */
19 /*-----*/
20 delay(int t) /* 延迟子程序 */
21 {
22 int i,j;
23 for(i=0; i<t; i++)
24 for(j=0; j<10; j++);
25 }
26 /*-----*/
27 led_bli() /* 工作 LED 闪动 */
28 {
29 /* blink RED led P1.7 */
30 char i;
31

```

```
32  for(i=0; i<4; i++)
33  {
34  cplbit(P1.7);
35  delay(40);
36  }
37  }
38  /*-----*/
39  init_buf() /* 载入显示器初值 */
40  {
41  int i;
42  for(i=0; i<4; i++)
43  disp[i]=DATA_7SEG[i];
44  }
45  /*-----*/
46  INTERRUPT(_TF0_) t0isr() /* 计数器 0 5ms 中断程序 */
47  {
48  TH0=HI; /* 载入初值 */
49  TL0=LO;
50  /* 由 PB 端口(PB0~PB7) 送出字符数据 */
51  pb=disp[dpt];
52  pc=act[dpt]; /* PC 端口(PC0~PC3)送出扫描控制信号 */
53  dpt++; /* 指针+1 */
54  if(dpt>=4) dpt=0; /* 清除指针 */
55  }
56  /*-----*/
57  init_t0() /* 初始化计数器 0 */
58  {
59  TMOD=0x01; /* 设定工作模式 */
60  TH0=HI; /* 载入初值 */
61  TL0=LO;
62  IE=0x82; /* 设定允许状态 */
63  setbit(TCON.4); /* TR0=1 启动计数器 0 */
64  }
65  /*-----*/
66  main() /* 主程序 */
67  {
```

```

68  led_bl(); /* 工作 LED 闪动 */
69  cr=CWORD; /* 设定 8255 工作模式 */
70  init_buf(); /* 载入显示器初值 */
71
72  serinit(9600); /* 初始化串行接口 */
73  printf(title); /* 显示工作消息 */
74
75  init_t0(); /* 初始化计数器 0 */
76  while(1) led_bl(); /* 工作 LED 持续闪动 */
77 }
78 /*-----*/

```

9.6.1 计数器 0 及计数器 1 同时存在

在设计程序时，除了以计时中断程序来定时扫描 7 段数码管外，我们还希望仍能使使用 8051 的串行端口来与 PC 做消息沟通，有关 8051 的串行接口，我们将在下一章来做介绍，其中串行传输波特率的设定是使用计数器 1 模式 2 来做设定，下面的例子将说明计数器 0 及计数器 1 同时存在时要怎设计。由计数器 0 来定时扫描 7 段数码管显示数据，而由计数器 1 做传输波特率的设定。

执行结果

程序执行后，PC 端执行 RS232 监控程序，则 7 段数码管显示“1234”，PC 屏幕上会不断的出现从单板上送出的 RS232 消息如下：

Test P51_PCB 2 timer RS232+7seg ...

程序清单

```

1  /* TSEG1.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  #define HI 238 /* mode 1 Sms 初值设定 */
9  #define LO 0
10
11 #define CWORD 0x88 /* PA PB PC0~3 o/p PC4~7 i/p */
12 char *title="Test P51_PCB 2 timer RS232+7seg ...\n";

```

```
13  /* 0~9 的字符数据 */
14  char DATA_7SEG[10]={0x3f, 0x06, 0x5b, 0x4f, 0x66,
15      0x6d, 0x7d, 0x07, 0x7f, 0x6f};
16  char act[4]={0xfe, 0xfd, 0xfb, 0xf7} /* 扫描控制信号 */
17  char disp[4]; /* 四位数码管数据缓冲区 */
18  char dpt=0; /* 数据缓冲区指针 */
19  /*-----*/
20  delay(int t) /* 延迟子程序 */
21  {
22      int i,j;
23      for(i=0; i<t; i++)
24          for(j=0; j<10; j++);
25  }
26  /*-----*/
27  led_bf() /* 工作 LED 闪动 */
28  {
29      /* blink RED led P1.7 */
30      char i;
31
32      for(i=0; i<4; i++)
33      {
34          cplbit(P1.7);
35          delay(40);
36      }
37  }
38  /*-----*/
39  init_buf() /* 载入显示器初值 */
40  {
41      int i;
42      for(i=0; i<4; i++)
43          disp[i]=DATA_7SEG[i];
44  }
45  /*-----*/
46  INTERRUPT(_TF0_) t0isr() /* 计数器 0 5 ms 中断程序 */
47  {
48      TH0=HI; /* 载入初值 */
```

```
49  TL0=LO;
50  /* 由 PB 端口(PB0~PB7) 送出字符数据 */
51  pb=disp[dpt];
52  pc=act[dpt]; /* PC 端口(PC0~PC3) 送出扫描控制信号 */
53  dpt++; /* 指针+1 */
54  if(dpt>=4) dpt=0; /* 清除指针 */
55  }
56  /*-----*/
57  init_t0() /* 初始化计数器 0 */
58  {
59  TMOD=0x01; /* 设定工作模式 */
60  TH0=HI; /* 载入初值 */
61  TL0=LO;
62  IE=0x82; /* 设定允许状态 */
63  setbit(TCON.4); /* TR0=1 启动计数器 0 */
64  }
65  /*-----*/
66  init_timer()
67  {
68  /* 设定工作模式 */
69  TMOD=0x21; /* 计数器 1 模式 2,计数器 0 模式 1 */
70  TH0=HI; /* 载入初值 */
71  TL0=LO;
72
73  IP=0x10; /* PS=1 设定串行端口的优先中断顺序 */
74  IE=0x82; /* 设定允许状态 */
75  setbit(TCON.4); /* TR0=1 启动计数器 0 */
76  setbit(TCON.6); /* TR1=1 启动计数器 1 */
77  }
78  /*-----*/
79  main() /* 主程序 */
80  {
81  led_b1(); /* 工作 LED 闪动 */
82  cr=CWORD; /* 设定 8255 工作模式 */
83  init_buf(); /* 载入显示器初值 */
84
```

```

85  serinit(9600); /* 初始化串行接口 */
86  printf(title); /* 显示工作消息 */
87
88  init_timer(); /*初始化计数器 */
89  while(1)      /* 无穷循环 */
90  {
91  printf(title); /* 送出工作消息 */
92  delay(100);
93  }
94  }
95  /*-----*/

```

9.7 驱动 7 段数码管及按键扫描

看过以计数器驱动 7 段数码管后，如果要将按键扫描的工作加入程序中，如何设计效果才比较好呢？由于按键扫描的程序代码较长，而计时中断程序的设计希望程序代码越短越好，因此仍采用循环扫描的方式来做按键扫描，而以计数器 0 来驱动 7 段数码管，并同时启动 RS232 串行接口来与 PC 建立通信。以下是程序范例：

执行结果

程序执行后，PC 端执行 RS232 监控程序，则 7 段数码管显示“1234”，PC 屏幕上会出现从单板上送出的 RS232 消息如下：

Please i/p digit ?

要求由单板上按键输入数字(0~9)，如果按下“1”则单板上会送出如下的消息：

get digit 1

程序清单

```

1  /*  TSKEY.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h"     /* 载入 P51 I/O 控制头文件 */
7
8  #define HI    238     /* mode 1 5ms 初值设定 */
9  #define LO    0

```

```

10
11 #define CWORD 0x88 /* PA PB PC0~3 o/p PC4~7 i/p */
12 char *title="P51_PCB test key i/p+7seg+RS232.....\n";
13 /* 0~9 的字符数据 */
14 char DATA_7SEG[10]={0x3f, 0x06, 0x5b, 0x4f, 0x66,
15                      0x6d, 0x7d, 0x07, 0x7f, 0x6f};
16 char act[4]={0xfe, 0xfd, 0xfb, 0xf7} /* 扫描控制信号 */
17 char disp[4]; /* 四位数码管数据缓冲区 */
18 char dpt=0; /* 数据缓冲区指针 */
19
20 char key; /* 键盘扫描值 */ /* 16 个键盘按键 */
21 char skey[16]={'A', 'B', 'C', 'D', '3', '6', '9', '#',
22              '2', '5', '8', '0', '1', '4', '7', '*'};
23 char scan_key(); /* 键盘扫描控制程序 */
24 /*-----*/
25 delay(int t) /* 延迟子程序 */
26 {
27     int i,j;
28     for(i=0; i<t; i++)
29         for(j=0; j<10; j++);
30 }
31 /*-----*/
32 led_bl() /* 工作 LED 闪动 */
33 {
34     char i;
35
36     for(i=0; i<4; i++)
37     {
38         cplbit(P1.7);
39         delay(40);
40     }
41 }
42 /*-----*/
43 init_buf() /* 载入显示器初值 */
44 {
45     int i;

```



```
46   for(i=0; i<4; i++)
47   disp[i]=DATA_7SEG[i];
48   }
49   /*-----*/
50   INTERRUPT(_TF0_) t0isr() /* 计数器 0 5 ms 中断程序 */
51   {
52     TH0=HI; /* 载入初值 */
53     TL0=LO;
54     /* 由 PB 端口(PB0~PB7) 送出字符数据 */
55     pb=disp[dpt];
56     pc=act[dpt]; /* PC 端口(PC0~PC3) 送出扫描控制信号 */
57     dpt++; /* 指针+1 */
58     if(dpt>=4) dpt=0; /* 清除指针 */
59   }
60   /*-----*/
61   init_t0() /* 初始化计数器 0 */
62   {
63     TMOD=0x01; /* 设定工作模式 */
64     TH0=HI; /* 载入初值 */
65     TL0=LO;
66     IE=0x82; /* 设定允许状态 */
67     setbit(TCON,4); /* TR0=1 启动计数器 0 */
68   }
69   /*-----*/
70   init_timer() /* 初始化计数器 */
71   {
72     /* 设定工作模式 */
73     TMOD=0x21; /* 计数器 1 模式 2, 计数器 0 模式 1 */
74     TH0=HI; /* 载入初值 */
75     TL0=LO;
76
77     IP=0x10; /* PS=1 设定串行端口的优先中断顺序 */
78     IE=0x82; /* 设定允许状态 */
79     setbit(TCON,4); /* TR0=1 启动计数器 0 */
80     setbit(TCON,6); /* TR1=1 启动计数器 1 */
81   }
```

```
82  /*-----*/
83  t0_on() /* TR0=1 启动计数器 0 */
84  {
85      setbit(TCON.4); /* T0 run */
86  }
87  /*-----*/
88  t0_off() /* TR0=0 关闭计数器 0 */
89  {
90      clrbit(TCON.4); /* T0 stop */
91  }
92  /*-----*/
93  char scan_key() /* 键盘扫描控制程序 */
94  {
95      char i,j, find, ini, inj;
96      char in;
97
98      find=0; /* 清除按键标志 */
99      for(i=0; i<4; i++)
100  {
101      /* 由 PB 端口(PB0~PB7) 送出字符数据 */
102      pb=disp[i];
103
104      /* 由 PC 端口(PC0~PC3) 送出扫描控制信号 */
105      pc=act[i];
106      delay(3);
107
108      /* 由 PC 端口(PC4~PC7) 读取按键返回代码 */
109      in=pc;
110      in=in>>4; /* 右移 4 位 */
111      in=in | 0xf0; /* 高 4 位设为 1 */
112
113      /* 检查是否按键 ? */
114      for(j=0; j<4; j++)
115          if(act[j]==in)
116          {
117              find=1; /* 设定按键标志 */
```

```
118     inj=j; ini=i; /* 记录扫描指针值 */
119 }
120 }/* scan 1 time */
121 /* 没按键传回 0 值 */
122 if(find==0) return 0;
123
124 /* i,j ---> key : 0~15 */
125 key=ini*4+inj; /* 计算按键值 */
126 return 1; /* 有按键传回 1 */
127 }
128 /*-----*/
129 get_key() /* 等待按键输入并放开来 */
130 {
131     while(1) if(scan_key()==1) break;
132     while(1) if(scan_key()==0) break;
133 }
134 /*-----*/
135 main() /* 主程序 */
136 {
137     char c;
138
139     led_bl(); /* 工作 LED 闪动 */
140     cr=CWORD; /* 设定 8255 工作模式 */
141     init_buf(); /* 载入显示器初值 */
142
143     serinit(9600); /* 初始化串行接口 */
144     printf(title); /* 显示工作消息 */
145
146     init_timer(); /* 初始化计数器 */
147     while(1) /* 无穷循环 */
148     {
149         /* 请输入数字 */
150         printf("Please i/p digit ? ");
151         t0_off(); /* 关闭计数器 0 */
152         get_key(); /* 等待按键输入 */
153         t0_on(); /* 开启计数器 0 */
```

```

154
155  c=skey[key]; /* 按键值转换 */
156  c=c-0x30; /* 按键值转换为数字 */
157  /*      printf("%d ", c); */      /* 按键无效 */
158  if( (c>=10) || (c<0) ) { printf("\n"); continue; }
159  printf("get digit %d \n", c); /* 显示数字 */
160  }
161  }
162  /*-----*/

```

9.8 计时时钟的制作

在看过以计数器中断方式驱动 7 段数码管的方法后, 本节将设计一个计时时钟程序, 读者可以发现以 C 语言来写的话, 程序相当的简洁易懂并具有结构化, 否则以汇编语言来设计, 程序代码会长一点。4 位 7 段数码管显示格式如下:

分分	秒秒
----	----

若按下电路板上任何按键则显示格式变为:

时时	分分
----	----

再按一下“4”键, 则会恢复到原来的显示方式。计时中断程序也是 5ms 执行一次, 而只要执行过 200 次中断程序后, 便是一秒钟了。

计数器使用如下:

1. 计数器 0: 5ms 计数;
2. 计数器 1: 计数器 1 模式 2 串行传输波特率的设定, 驱动 7 段数码管及按键扫描以循环来处理。因此必需经常做循环扫描否则显示器会有闪烁的现象, 或是根本不亮。

执行结果

程序执行后, 4 位 7 段数码管显示“00 00”并开始计时, 秒数为每秒自动累加 1, 等于 60 秒时则分数加 1, 按下单板上任何按键则显示方式做如上所示的变化。

程序清单

```

1  /*  CL.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */

```

```

3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h"      /* 载入 P51 I/O 控制头文件 */
7
8  #define HI    238      /* mode 1 5ms 初值设定 */
9  #define LO    0
10 #define DEDA  200 /* 200*5ms=1sec */
11
12 #define CWORD 0x88      /* PA PB PC0~3 o/p  PC4~7 i/p */
13 char *title="Test P51_PCB CLOCK MM: SS.....\n";
14 /* 0~9 的字符数据 */
15 char DATA_7SEG[10]={0x3f, 0x06, 0x5b, 0x4f, 0x66,
16                      0x6d, 0x7d, 0x07, 0x7f, 0x6f};
17 char act[4]={0xfe, 0xfd, 0xfb, 0xf7} /* 扫描控制信号 */
18 char disp[4]; /* 四位数码管数据缓冲区 */
19
20 char key; /* 键盘扫描值 */ /* 16 个键盘按键 */
21 char skey[16]={'A', 'B', 'C', 'D', '3', '6', '9', '#',
22              '2', '5', '8', '0', '1', '4', '7', '*'};
23 char scan_key(); /* 键盘扫描控制程序 */
24 /*-----*/
25 char hour=0, min=0, sec=0; /* 清除时间 */
26 char deda=0; /* 计数值 */
27 char mode=0; /* 显示模式 mode=0 MM: SS  mode=1 HH: MM */
28
29 /*-----*/
30 delay(int t) /* 延迟子程序 */
31 {
32     int i,j;
33     for(i=0; i<t; i++)
34         for(j=0; j<10; j++);
35 }
36 /*-----*/
37 led_bl() /* 工作 LED 闪动 */
38 {

```

```
39  char i;
40
41  for(i=0; i<4; i++)
42  {
43      cplbit(P1.7);
44      delay(40);
45  }
46 }
47 /*-----*/
48 init_timer() /* 初始化计数器 */
49 {
50     /* 设定工作模式 */
51     TMOD=0x21; /* 计数器 1 模式 2,计数器 0 模式 1 */
52     TH0=HI; /* 载入初值 */
53     TL0=LO;
54
55     IP=0x10; /* PS=1 设定串行端口的优先中断顺序 */
56     IE=0x82; /* 设定允许状态 */
57     setbit(TCON.4); /* TR0=1 启动计数器 0 */
58     setbit(TCON.6); /* TR1=1 启动计数器 1 */
59 }
60 /*-----*/
61 char scan_key() /* 键盘扫描控制程序 */
62 {
63     char i,j, find, ini, inj;
64     char in;
65
66     find=0; /* 清除按键标志 */
67     for(i=0; i<4; i++)
68     {
69         /* 由 PB 端口(PB0~PB7) 送出字符数据 */
70         pb=disp[i];
71
72         /* 由 PC 端口(PC0~PC3)送出扫描控制信号 */
73         pc=act[i];
74         delay(3);
```

```
75
76 /* 由 PC 端口(PC4~PC7)读取按键返回代码 */
77 in=pc;
78 in=in>>4;
79 in=in|0xf0;
80
81 /* 检查是否按键 ? */
82 for(j=0; j<4; j++)
83     if(act[j]==in)
84     {
85         find=1; /* 设定按键标志 */
86         inj=j; ini=i; /* 记录扫描指针值 */
87     }
88 } /* scan 1 time */
89 /* 没按键传回 0 值 */
90 if(find==0) return 0;
91
92 /* i,j ---> key : 0~15 */
93 key=ini*4+inj; /* 计算按键值 */
94 return 1; /* 有按键传回 1 */
95 }
96 /*-----*/
97 get_key() /* 等待按键输入并放开来 */
98 {
99     while(1) if(scan_key()==1) break;
100     while(1) if(scan_key()==0) break;
101 }
102 /*-----*/
103 INTERRUPT(_TF0_) t0isr() /* 计数器 0 5 ms 中断程序 */
104 {
105     TH0=HI; /* 载入初值 */
106     TL0=LO;
107
108     deda++; /* 计数器加 1 */
109     if(deda==DEDA) /* 1 秒钟到了 */
110     {
```

```
111     sec++; deda=0;
112 }
113 }
114 /*-----*/
115 conv() /* 时间值转换为 7 段数码管数据 */
116 {
117     if(sec==60) {min++; sec=0;}
118     if(min==60) {hour++; min=0;}
119     if(hour==24) hour=0;
120
121     if(mode==1) /* 显示" 时时 分分" */
122     {
123         disp[0]=DATA_7SEG[hour/10];
124         disp[1]=DATA_7SEG[hour%10];
125         disp[2]=DATA_7SEG[min/10];
126         disp[3]=DATA_7SEG[min%10];
127     }
128     else /* 显示" 分分 秒秒" */
129     {
130         disp[0]=DATA_7SEG[min/10];
131         disp[1]=DATA_7SEG[min%10];
132         disp[2]=DATA_7SEG[sec/10];
133         disp[3]=DATA_7SEG[sec%10];
134     }
135 }
136 /*-----*/
137 main() /* 主程序 */
138 {
139     led_bl(); /* 工作 LED 闪动 */
140     cr=CWORD; /* 设定 8255 工作模式 */
141     conv(); /* 时间数据转换 */
142
143     serinit(9600); /* 初始化串行接口 */
144     printf(title); /* 显示工作消息 */
145     init_timer(); /* 初始化计数器 */
146 }
```



```

147 while(1)/* 循环 */
148 {
149 conv(); /* 时间数据转换 */
150 if(scan_key()==1)/* 有按键了 */
151 {
152 while(1)/* 等按键放开来 */
153 {
154 conv();/* 时间数据转换 */
155 if(scan_key()==0) break;
156 }
157 mode=1-mode; /* 显示模式切换 */
158 }/* if keyed */
159 }/* loop */
160 }
161 /*-----*/

```

9.9 手动计数器实验

前面已讲过以计时中断方式来驱动 7 段数码管做数字的显示，本节将做手动计数器的实验。利用 8051 单板 P51 上的 4 位 7 段数码管来显示计数的数值，每次有按键，在放开按键时计数值加 1，4 位 7 段数码管显示 4 位数字，格式如下：

千	百	十	个
---	---	---	---

其显示范围为 0000 至 9999，在功能扩充方面，例如将 8051 单板上的位 P1.0 接出来做输入控制，将传感器的输出送往此输入位，便可以完成自动计数的功能，常见的有红外线自动计数器。平时红外线接收器一直收到由发射器所发射的信号，一旦有人通过时便将信号屏蔽掉，因此检测到有人通过，便送出一个计数脉冲信号给 8051 单板的 P1.0 输入位，完成计数自动加一的功能，这样的设计常用在展览会场入口处，用于统计参观人数自动计数器上。

执行结果

程序 TCO.C 是其执行程序，利用 8051 单板上的 4 位 7 段数码管来显示计数的数值，当每次有按键，在放开按键时蜂鸣器“嘀”一声，计数值加 1，若一直按着按键不放，则显示器的计数数据不变。

程序清单

```

1  /* TCO.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  #define CWORD 0x88 /* PA PB PC0~3 o/p PC4~7 i/p */
9  char *title="Test P51_PCB manual counter.....\n";
10 /* 0~9 的字符数据 */
11 char DATA_7SEG[10]={0x3f, 0x06, 0x5b, 0x4f, 0x66,
12 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
13 char act[4]={0xfe, 0xfd, 0xfb, 0xf7} /* 扫描控制信号 */
14 char disp[4]; /* 四位数码管数据缓冲区 */
15
16 char key; /* 键盘扫描值 */ /* 16 个键盘按键 */
17 char skey[16]={'A', 'B', 'C', 'D', '3', '6', '9', '#',
18 '2', '5', '8', '0', '1', '4', '7', '*'};
19 char scan_key(); /* 键盘扫描控制程序 */
20 int count; /* 计数值 */
21 /*-----*/
22 delay(int t) /* 延迟子程序 */
23 {
24 int i,j;
25 for(i=0; i<t; i++)
26 for(j=0; j<10; j++);
27 }
28 /*-----*/
29 led_b1() /* 工作 LED 闪动 */
30 {
31 char i;
32
33 for(i=0; i<4; i++)
34 {
35 cplbit(P1.7);

```

```
36 delay(40);
37 }
38 }
39 /*-----*/
40 char scan_key() /* 键盘扫描控制程序 */
41 {
42     char i,j, find, ini, inj;
43     char in;
44
45     find=0; /* 清除按键标志 */
46     for(i=0; i<4; i++)
47     {
48         /* 由 PB 端口(PB0~PB7) 送出字符数据 */
49         pb=disp[i];
50
51         /* 由 PC 端口(PC0~PC3) 送出扫描控制信号 */
52         pc=act[i];
53         delay(3);
54
55         /* 由 PC 端口(PC4~PC7) 读取按键返回代码 */
56         in=pc;
57         in=in>>4; /* 右移 4 位 */
58         in=in | 0xf0; /* 高 4 位设为 1 */
59
60         /* 检查是否按键 ? */
61         for(j=0; j<4; j++)
62             if(act[j]==in)
63             {
64                 find=1; /* 设定按键标志 */
65                 inj=j; ini=i; /* 记录扫描指针值 */
66             }
67     } /* scan 1 time */
68     /* 没按键传回 0 值 */
69     if(find==0) return 0;
70
71     /* i,j --> key : 0~15 */
```

```
72   key=ini*4+inj; /* 计算按键值 */
73   return 1; /* 有按键传回 1 */
74 }
75 /*-----*/
76 conv() /* 计数值转换为 7 段数码管数据 */
77 {
78   disp[0]=DATA_7SEG[count/1000];
79   disp[1]=DATA_7SEG[count/100%10];
80   disp[2]=DATA_7SEG[count/10%10];
81   disp[3]=DATA_7SEG[count%10];
82 }
83 /*-----*/
84 be() /* 蜂鸣器“嘀”一声 */
85 {
86   char i;
87
88   for(i=0; i<10; i++)
89   {
90     cplbit(P1.0);
91     delay(1);
92   }
93 }
94 /*-----*/
95 main() /* 主程序 */
96 {
97   led_bl(); /* 工作 LED 闪动 */
98   cr=CWORD; /* 设定 8255 工作模式 */
99   be(); /* 蜂鸣器“嘀”一声 */
100
101   count=0; /* 计数值清为 0 */
102   conv(); /* 计数数据转换 */
103
104   serinit(9600); /* 初始化串行接口 */
105   printf(title); /* 显示工作消息 */
106
107   while(1) /* 循环 */
```

```

108 {
109 conv();/* 计数数据转换 */
110 if(scan_key()==1)/* 有按键了 */
111 { /* wait key off*/
112 be(); /* 蜂鸣器“嘀”一声 */
113 while(1)/* 循环等按键放开来 */
114 {
115 conv();/* 计数数据转换 */
116 if(scan_key()==0) break;
117 }
118 /* 计数值加 1 */
119 count++;
120 }/* if keyed */
121 }/* loop */
122 }
123 /*-----*/

```

9.10 简易频率计实验

8051 的计数器如何应用于计数呢？本节举一实际的例子——频率计实验来做说明。计频仪是用来测量周期信号的频率大小，即每秒钟可以测量到多少个输入脉冲，而到底有多少个输入脉冲送入 8051 单片机中，此时可以启动计时/计数器的计数功能，由 T0 或 T1 引脚来输入脉冲，进而计下脉冲的数字；再设一个定时器，只要每隔 1 秒就将计数器中的值读出显示在 7 段数码管上，便可得知待测频率是多少了。

我们可以利用计数器 0 作为 5ms 定时的计时，在计时中断产生时执行中断服务程序做秒的时间计时，若 1 秒计时到了则将计数值取出。而外部的待测信号如何产生呢？我们可以由 T0 引脚来输出方波信号，送到 T1 引脚作为计数输入，因此启动计数器 1 作为计数器来使用。

计数器 1 的设定是工作于模式 1 做 16 位的计数，因此任何时刻想读取计数器的计数值可以用指令：

```
count=TH1X256+TL1;
```

便可以将计数值放入变量中，通过数码转换而显示在 7 段数码管中。

执行结果

程序 TFR.C 为简易频率计控制程序，程序中我们利用 T0 引脚送出一测试方波信号，因此程序一执行时显示器上显示为 0，此时没有信号输入，若将 T0 引脚与 T1 引脚（脉

冲输入引脚)接在一起,马上可见其频率读取值为 100,表示此方波测试信号为 100Hz,此方波测试信号是由引脚 P3.4 (T0 引脚)送出,其每隔 1 次计时中断(5ms)便反向一次,因此其周期为 10ms,所以我们可以测得其信号频率为 100Hz。

程序清单

```

1  /*  TFR.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  #define CWORD 0x88 /* PA PB PC0~3 o/p PC4~7 i/p */
9  char *title="P51_PCB freq. counter    μ se T0 o/p pulse, T1 i/p\n";
10
11 #define HI 238 /* mode 1 5ms 初值设定 */
12 #define LO 0 /* mode 1 5ms 初值设定 */
13 #define DEDA 200 /* 200*5ms=1sec */
14 /* 0~9 的字符数据 */
15 char DATA_7SEG[10]={0x3f, 0x06, 0x5b, 0x4f, 0x66,
16                      0x6d, 0x7d, 0x07, 0x7f, 0x6f};
17 char act[4]={0xfe, 0xfd, 0xfb, 0xf7} /* 扫描控制信号 */
18 char disp[4]; /* 四位数码管数据缓冲区 */
19 char key; /* 键盘扫描值 */ /* 16 个键盘按键 */
20 char skey[16]={'A', 'B', 'C', 'D', '3', '6', '9', '#',
21              '2', '5', '8', '0', '1', '4', '7', '*'};
22 char scan_key(); /* 键盘扫描控制程序 */
23 unsigned int count; /* 外部输入计数值 */
24 int deda=0; /* 时间计数值 */
25 /*-----*/
26 delay(int t) /* 延迟子程序 */
27 {
28     int i,j;
29     for(i=0; i<t; i++)
30         for(j=0; j<10; j++);
31 }
32 /*-----*/

```

```
33 led_bl() /* 工作 LED 闪动 */
34 {
35     char i;
36
37     for(i=0; i<4; i++)
38     {
39         cplbit(P1.7);
40         delay(40);
41     }
42 }
43 /*-----*/
44 char scan_key() /* 键盘扫描控制程序 */
45 {
46     char i,j, find, ini, inj;
47     char in;
48
49     find=0; /* 清除按键标志 */
50     for(i=0; i<4; i++)
51     {
52         /* 由 PB 端口(PB0~PB7) 送出字符数据 */
53         pb=disp[i];
54
55         /* 由 PC 端口(PC0~PC3) 送出扫描控制信号 */
56         pc=act[i];
57         delay(1);
58
59         /* 由 PC 端口(PC4~PC7) 读取按键返回代码 */
60         in=pc;
61         in=in>>4; /* 右移 4 位 */
62         in=in|0xf0; /* 高 4 位设为 1 */
63
64         /* 检查是否按键 ? */
65         for(j=0; j<4; j++)
66             if(act[j]==in)
67             {
68                 find=1; /* 设定按键标志 */
```

```
69     inj=j; ini=i; /* 记录扫描指针值 */
70 }
71 /* scan 1 time */
72 /* 没按键传回 0 值 */
73 if(find==0) return 0;
74
75 /* i,j ---> key : 0~15 */
76 key=ini*4+inj; /* 计算按键值 */
77 return 1; /* 有按键传回 1 */
78 }
79 /*-----*/
80 conv() /* 计数值转换为 7 段数码管数据 */
81 {
82     disp[0]=DATA_7SEG[count/1000];
83     disp[1]=DATA_7SEG[count/100%10];
84     disp[2]=DATA_7SEG[count/10%10];
85     disp[3]=DATA_7SEG[count%10];
86 }
87 /*-----*/
88 main() /* 主程序 */
89 {
90     led_b1(); /* 工作 LED 闪动 */
91     cr=CWORD; /* 设定 8255 工作模式 */
92
93     count=0;
94     serinit(9600); /* 初始化串行接口 */
95     printf(title); /* 显示工作消息 */
96
97     init_timer(); /* 初始化计数器 */
98     while(1)
99     {
100     conv(); /* 计数值转换为 7 段数码管数据 */
101     scan_key(); /* 扫描驱动 7 段数码管显示数据 */
102     }
103 }
104 /*-----*/
```



```
105 init_timer() /* 初始化计时计数器 */
106 {
107     /* 计数器 1 模式 1, 计数器 0 模式 1 */
108     TMOD=0x51;
109     TH0=HI; /* 载入计时初值 */
110     TL0=LO;
111
112     IE=0x82; /* 设定允许状态 */
113     /* TR0=1; 启动计数器 0 */
114     setbit(TCON.4);
115
116     /* TR1=1; 启动计数器 1 */
117     setbit(TCON.6);
118
119 }
120 /*-----*/
121 INTERRUPT(_TF0_) t0isr() /* 计数器 0 5 ms 中断程序 */
122 {
123     /* 载入初值 */
124     TH0=HI;
125     TL0=LO;
126     /* 计数器加一 */
127     deda++;
128     if(deda==DEDA) /* 1 秒钟到了 */
129     {
130         cplbit(P1.7); /* 工作 LED 闪动 */
131         deda=0; /* 计数器加 1 */
132         count=TH1*256+TL1; /* 计算计数器值 */
133         TH1=TL1=0; /* 计数器值清 0 */
134     }
135
136     /* 由 T0 输出方波测试信号 */
137     cplbit(P3.4);
138 }
139 /*-----*/
```

9.11 习 题

1. 以计数器 0 工作模式 0 设计一程序产生 3 ms 宽的方波信号。
2. 以计数器 0 工作模式 1 设计一程序产生 5 ms 宽的方波信号。
3. 以计数器 0 工作模式 2 设计一程序产生 0.2 ms 宽的方波信号。

第10章 串行接口控制

电脑的通信接口十分有用,除了可以做基本数据的传送、遥控系统的设计外,还可以完成特殊硬件扩充的连接工作,这在做数据收集或自动控制工程应用上均是相当重要的技术。而 8051 在通信接口上提供了我们方便且好用的功能,在单片机内部备有串行传输接口,只要外加信号电平转换 IC 便可完成常用的 RS232 串行通信传输,可做多台单片机系统的控制,可与 PC 连接做控制,很多单片机学习机或开发工具(如 ICE)均是以此方式来做控制的。

在本章中我们将说明串行传送的通信原理,及 8051 串行端口的使用,并以实验来说明串行数据的接收及传送,这些都是一些非常基本的测试程序,熟悉这些程序的设计,在以后 8051 单板专题制作上用途很多,可以做多片 8051 单片机的系统连接控制也可以与 PC 建立数据间通信。

10.1 串行数据传送原理

单片机间通信基本的传送方式可分为 2 种:

1. 并行通信;
2. 串行通信。

10.1.1 并行通信

并行通信数据传送的方式,一次送出或接收一个字节(8 个 bit),如图 10-1 所示,通常在单片机的 I/O 上会接有接口控制芯片,其数据总线可视为是并行传送的一种,只不过是在单片机系统内部运作而已,并未对外做通信。典型的例子如 PC 连接的打印机就是使用并行通信的方式,称为 Centronics 接口。其中包含有交握式的控制接口,以保证数据传送的正确性。使用并行数据传送的优点是速度快,适合近距离的传送,对远距离的电脑通信,由于传输线成本增加、电子信号衰减等问题,会考虑使用串行通信的传送技术。

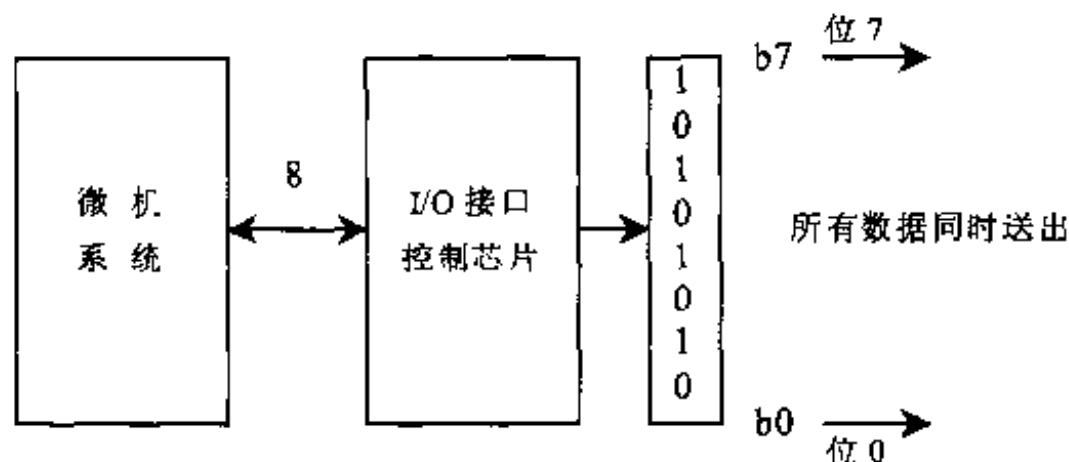


图 10-1 并行数据传输

10.1.2 串行通信

串行通信是以一连串的位形式将数据传送出去或接收进来，在任一瞬间则只传送一个位，如图 10-2 所示。数据传送较费时，但却可以降低传输线的硬件成本，特别适合做较长距离的电脑通信。典型的串行通信传输方式是使用 RS232 接口，属于一种非同步传送格式，使用相当普遍，像是一些激光打印机，较高级的仪器设备如数字示波器、自动测量仪器均会提供此一通信接口，使得与电脑间可以很容易的建立连接，增加整个仪器本身的扩充能力。而设计 RS232 通信接口并不困难，在了解了非同步串行数据传送的原理后，以单片机 8051 结合必要的驱动程序便可轻易的做通信控制了。

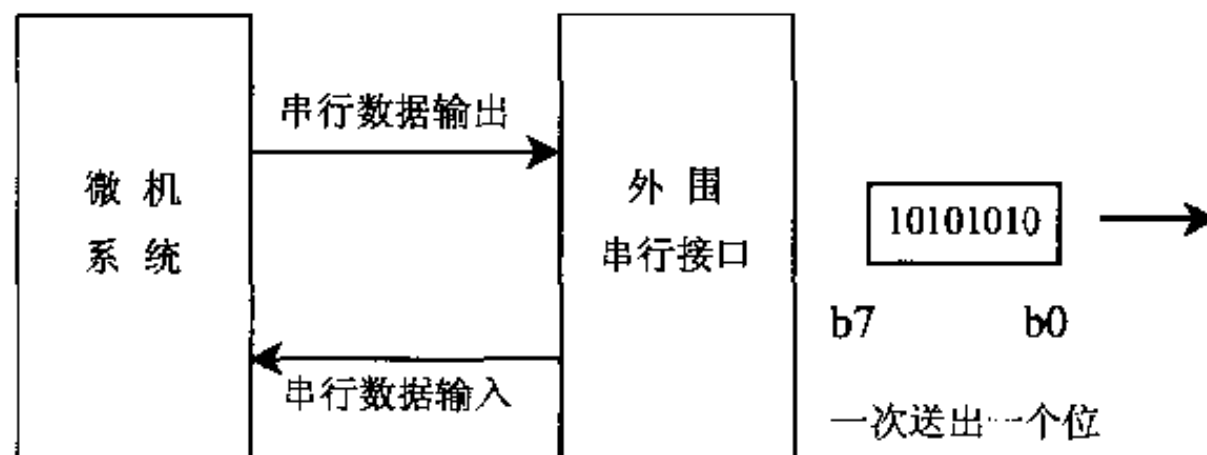


图 10-2 串行数据传输

10.1.3 非同步串行数据传输

非同步串行数据传输的格式共由以下 5 项组成：

标记	起始位	数据位	校验位	停止位
----	-----	-----------	-----	-----

1. 标记

当串行传输线上不传送数据时，它所处的状态称为标记状态，用以告知对方目前是处于待机闲置的状态下，此信号一直保持在高电位下。

2. 起始位

在真正传送数据位前，会先送出一个低电位的位，以告知接收端马上就要传送数据出去了，标记一直保持在高电位下，一旦送出起始位低电位后，在这转态的瞬间，接收端与发送端便取得同步。

3. 数据位

真正传送的数据在起始位送出后，便逐一将位一个一个送出去（位 0 最先送出）。数据的长度可以是 5 到 8 个位，例如是英文的文字文件，则只要用到 7 个位传送即可，使用 8 个位便可以传送文档或任何数据文件。

4. 校验位

在传送完每一个位数据后，接着送出校验检查位，用来检查数据在传送的过程中是

否发生错误, 校验位检查可以是奇校验或偶校验。采用奇校验做检查, 表示所有数据位加上校验位后, “1”的总数要为奇数, 反之偶校验则是所有数据位加上校验位, “1”的总数应为偶数个。当然, 也可以不使用校验位检查, 在数据传送中, 少传一个位, 可加快传输速度。

5. 停止位

在一连串的传送位的最后一个位称为停止位, 用以表示一个字节的的数据已传送完毕。停止位可以是1个、1.5个或2个, 按照需要而做选择。很明显的, 在串行传输中, 加入开始及结束位的主要功能是让收发两端可以随时取得同步, 使得数据传输无误。

图10-3为字节6BH经串行传输接口送出时的波形图, 传送一个字节共花了11个位宽的传送时间。除了数据项8位外多加了起始、停止及校验检查位, 其中可以看出校验检查是采用奇校验, 因为数据项加上校验位, 共有5个“1”, 奇数个“1”。至于传送的速度到底多快, 这就与波特率(Baud Rate)有关。

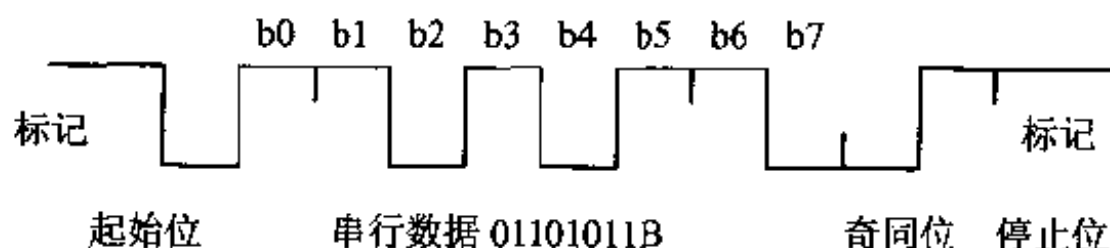


图 10-3 串行数据“6B”传送波形图

10.1.4 传输速率——波特率

每秒钟可以传送几个位的数据称为波特率, 其单位是BPS (Bit Per Second)。典型的传输速率有1200、2400、4800和9600BPS, 以9600BPS为例, 表示每秒可以传送9600位的数据, 若传送如图10-3的数据, 共花了11个位, 以9600除以11可以得到873, 表示每秒可以传送873个字节, 波特率越高传送时间愈短, 至于应采用何种传输率来传送数据呢? 此乃收发双方的事, 两方均要一致, 便不会有问题。只要数据传输不出错, 当然是越快越好, 较常使用的非同步串行传输通信协议为(9600,8,N,1), 即波特率为9600PBS, 传送或接收8个数据位, 没有校验检查, 1个停止位, 而起始位一直会存在着。

10.2 8051 串行传输接口

8051 内部含有一组全双工的串行传输接口, 可以同时传送或接收外部送来的数据, 而内部是如何完成串行传送的工作呢? 8051 串行数据的传送及接收均是通过特殊功能寄存器中的SBUF来处理, 只要设定好通信协议的模式后, 以指令“MOV SBUF,A”就可以将存在SBUF寄存器内的数据通过引脚TXD, 以串行方式传送出去; 而指令“MOV A,SBUF”则会将外界的串行信号通过RXD引脚读进来, 转换成并行数据而放到A寄存器中。在程序的控制上均是使用SBUF寄存器, 但是8051内部各自含有发送和接收的缓

冲器，只是一个做读取，一个做写入，二者可以分别独立工作。8051 串行传送接口共提供了 4 种操作模式，由设计者来自由使用。

10.2.1 串行传输模式 0

此模式基本上是做串行传送 I/O 控制，而非真正的串行通信应用，工作于此模式时，由 TXD 引脚送出移位同步脉冲，由 RXD 引脚送出或接收串行数据。而串行数据的形式如何呢？它不具有起始及结束位，纯粹为 8 位数据，至于同步脉冲的宽度是固定的，为系统工作振荡周期的 1/12，等于是 8051 一个机器周期的时间。

当做串行输出时，可将 TXD、RXD 引脚接到串行输入并行输出（SIPO）转换 IC，如 74LS164，做额外的硬件输出扩充用，当 I/O 的输出控制位不够用时，便可应用此技巧做额外输出端口的应用。做串行输入时，则将 TXD、RXD 引脚接到并行输入串行输出（PISO）转换 IC，如 74LS165，做额外的硬件功能扩充。

10.2.2 串行传输模式 1

此为经常使用的串行传输工作模式，串行数据位由 TXD 引脚传送出去，由 RXD 引脚将对方送来的串行数据接收进来。而数据格式共有 10 个位，包括前方的起始位，8 位串行数据位及最后的停止位。至于传输率（波特率）快慢则由定时器 1 来规划，只要将不同的计数初值载入定时器中，可以做不同的波特率值设定。在串行数据传输中，如果两套系统同是使用 8051 单片机来做设计，传输距离又不长，便可以采用图 10-4 所示的直接连接方式做单片机间的连接作业，当然双方面波特率必须设为相同。

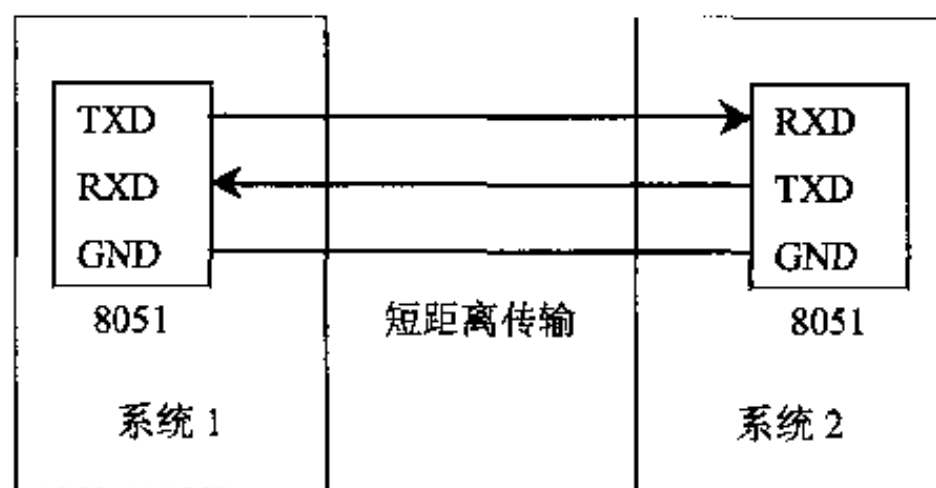


图 10-4 两组 8051 系统做短距离数据传送

10.2.3 串行传输模式 2

此传输模式与模式 1 十分类似，不过数据一共送出了 11 个位，包括 1 个起始位，8 个数据位及 1 个可编程设定的第 9 个数据位和停止位。此第 9 个可编程设定的数据位是位于特殊功能寄存器 SCON 中的位 3 中（TB8），8051 可以利用此一特殊位来做多处理机的系统连接控制。此外在通信协议中的校验位检查也可以此位来做处理。至于传送速度只有 2 种，分别为系统工作时钟频率的 1/32 或 1/64。

10.2.4 串行传输模式 3

模式 3 的传输方式与模式 2 几乎完全一样，同样是传送 11 个位串行数据，差别在于其传输速度是可变的，如同模式 1 一样由 8051 内部计时/计数器 1 所控制。

10.3 串行传送控制寄存器

8051 串行传送控制寄存器是由特殊功能寄存器 SCON 来做控制，SCON 设定了串行传送的工作模式，是否允许接收，发送接收时的第 9 个数据位及发送接收时中断指示工作标志。而波特率的设定是由定时器 1 来做控制，所以通信协议的规划以 8051 而言即是设定 SCON 及定时器 1 的工作模式及计数值。首先介绍 SCON 寄存器，每个位均是可位寻址，可分别设定或清除。

B7	B6	B5	B4	B3	B2	B1	B0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SM1	模式
0	0	0
0	1	1
1	0	2
1	1	3

- SM0、SM1：用于串行传输模式选择，共分为 4 种。
模式 0：移位寄存器控制 I/O，波特率固定为工作频率/12。
模式 1：8 位串行数据传送，波特率由定时器 1 来控制。
模式 2：9 位串行数据传送，波特率可分为 2 种，工作频率/32 或工作频率/64。
模式 3：9 位串行数据传送，波特率可用定时器 1 来控制。
- SM2：在串行传输工作模式 2 或模式 3 时，用于多处理机控制功能。
- REN：串行接口接收位，当 REN=1 时表示允许接收。
- TB8：在模式 2 或 3 时，所送出的第 9 个数据位，可以由软件指令来做控制设定或清除。
- RB8：在模式 2 或 3 时，所接收到的第 9 个数据位，存放在此位中。
- TI：串行传输数据发送中断产生标志，当工作于模式 0 时，送出 8 个数据位后 TI=1。而在其他工作模式时，在送出停止位时，TI 也会被设为 1，此位必需由软件来清除，所以在传送完数据后，要下达“CLR TI”指令来清除 TI 标志。
- RI：串行传输数据接收中断产生标志，工作于模式 0 时，当收到第 8 个串行输入数据位后，RI 会设为 1，在其他工作模式，收到停止位的一半时，硬件会自动将此位设为 1，同样的此位必需以软件指令“CLR RI”来清除。

10.4 串行传输波特率的设定

8051 串行传输波特率的设定根据不同的操作模式而定, 其中模式 0 及模式 2 属固定波特率, 而模式 1 及模式 3 为可变波特率, 由计时计数器 1 加以规划。

模式 0 波特率设定:

在模式 0 的操作下, 波特率是固定的, 为工作振荡频率的 1/12。

模式 2 波特率设定:

SMOD

在模式 2 操作下, 波特率 = $\frac{2}{64} \times (\text{工作振荡频率})$

其中 SMOD 为 SFR 中的 PCON 位 7:

当 SMOD=1 时, 波特率 = (工作频率) / 32

当 SMOD=0 时, 波特率 = (工作频率) / 64

模式 1 及模式 3 波特率设定:

在模式 1 及模式 3 操作下的波特率设定由内部计数器 1 来控制, 计数器的工作模式一共有 4 种, 模式 0 至模式 3, 必需工作于模式 2, 自动重新载入计时模式。在模式 2 的计时下, 使用的计数器寄存器为 TL1, 而 TH1 则是在做自动载入计时值的设定, 而波特率的计算公式为:

$$\text{波特率} = \frac{2}{32} \times \frac{\text{工作振荡频率}}{12 \times [256 - TH1]}$$

设计时我们是先定出波特率再求 TH1 的值, 将上式加以整理可得:

$$TH1 = 256 - \frac{2 \times (\text{工作振荡频率})}{384 \times \text{波特率}}$$

在 8051 单板上石英晶振使用 11.0592MHz, 各串行通信的波特率定为 9600 bps, SMOD 设为 0, 则可求得 TH1 如下:

$$TH1 = 256 - \frac{11059200}{384 \times 9600} = 253$$

同理, 我们可以将常用的波特率值代入公式而求得一些计时器自动载入值 (写入 TH1 中), 整理如下:

波特率	工作频率	SMOD	计数器 1 重新载入值
62500	12MHz	1	253
19200	11.0592MHz	1	253
9600	11.0592MHz	0	253
4800	11.0592MHz	0	250
2400	11.0592MHz	0	244
1200	11.0592MHz	0	232

10.5 PC 上的 RS232 通信程序

介绍过有关 8051 串行传输接口通信协议规划后,为了能在 PC 上做 8051 的通信传输实验,在本节中,我们先介绍 PC 上通信程序的控制方式。在 PC 上做串行传输的控制相当方便,因为它已内含有 RS232 通信接口,包括有通信端口 1 COM1 及通信端口 2 COM2。平时通信端口可能接有鼠标或是连接有 MODEM,现在只要将通信端口经 D25 型接头,接到 8051 单板上便可做 8051 与 PC 间的通信实验了。

硬件的连接是如此的简单,至于软件呢?基本上也不困难,熟悉 TURBO C 语言的读者应该知道其本身提供有程序库可供我们做这方面的应用,利用现成的函数来做 I/O 控制,我们可以不必深入去了解硬件的结构,便能够来写控制程序。

TURBO C 函数中对于通信端口的控制是使用 bioscom 函数,其原型定义如下: int bioscom (int cmd, char brte, int port); 输入参数有 3 种:

- 1.cmd: 控制函数的工作命令;
- 2.byte: 通信协议参数、发送或接收的数据;
- 3.port: 通信端口设定

10.5.1 工作命令 cmd

- cmd=0: 设定通信协议参数,以 byte 定义来完成通信协议的设定。
- cmd=1: 将 byte 的值写入通信端口中。
- cmd=2: 从通信端口中读取一个字节数据。
- cmd=3: 传回通信端口目前的状态。

10.5.2 通信协议参数 byte

- 传输率 (波特率)

0X00=110 bps

0X20=150 bps

0X40=300 bps

0X60=600 bps

0X80=1200 bps

0XA0=2400 bps

0XC0=4800 bps

0XE0=9600 bps

• 校验检查位

0X00=没有校验检查位

0X08=采用奇校验

0X18=采用偶校验

• 数据字符长度

0X02=7 个数据位

0X03=8 个数据位

• 停止位个数

0X00=1 个停止位

0X04=2 个停止位

其中个别的项可以任意组合, 例如, 常用的通信协议为 (9600,8,N,1), 即:

波特率 9600 bps : 0XE0

8 个数据位 : 0X03

没有校验检查位 : 0X00

1 个停止位 : 0X00

则参数设定为 0XE3

计算方式如下:

$0XE0+0X00+0X03+0X00=0XE3$

10.5.3 通信端口 port 指定

0=com1, 通信端口 1

1=com2, 通信端口 2

执行 bioscom 函数后, 传回值为一整数, 其高字节 (b15~b8, 位 15 到位 8) 为通信端口状态, 低字节 (b7~b0) 则定义如下:

当 cmd=0 时, 表示调制解调器状态。

cmd=1 时, 无意义。

cmd=2 时, 表示所接收到的数据。

cmd=3 时, 调制解调器状态。

10.5.4 通信端口状态

若该位为 1 表示所描述的状态设定。

B15: 传输时间用完, 时间到了 (Time Out)。

B14: 传送移位寄存器 (Transmit Shift Register, TSR) 空了。

B13: 传送保持寄存器 (Transmit Holding Register, THR) 空了。

B12: 检查到中断 (Break) 信号。

B11: 发生帧格式 (Framing) 错误。

B10: 校验检查错误。

B9: 发生覆盖 (Overrun) 错误。

B8: 接收数据准备好。

其中帧格式错误表示串行输入缓冲器接收到一个完整的字符数据后, 却检查不到停止位。而覆盖错误表示存在串行输入缓冲器内的数据未被读取又收到下一个接收进来的字符数据, 因此前一个字符数据被新的字符数据覆盖掉而遗失。

10.5.5 MODEM (调制解调器) 状态

B7: 指示 DCD (Data Carrier Detect) 的状态。

B6: 指示 RI (Ring Indicator) 的状态。

B5: 指 DSR (Data Set Ready) 的状态。

B4: 指示 CTS (Clear To Send) 的状态。

B3: 检查到 DCD 信号有变化。

B2: 检查到 RI 信号有变化。

B1: 检查到 DSR 信号有变化。

B0: 检查到 CTS 信号有变化。

在看过 bioscom 函数的使用说明后, 要写一个通信程序便十分容易了, 程序 SE.C 是一个实用的串行通信程序, 其传输格式为 (9600,8,N,1), 使用通信端口 2, 因为平时我们的通信端口 1 往往接至鼠标。

执行结果

执行后, 出现如下画面:

```
-----  
SE.EXE   PC RS232  COM2 <9600 N 8 1>  
-----
```

此时凡是由通信端口 2 所接收到的字符均会显示在屏幕上, 用户从键盘输入的任何字符会出现在屏幕上, 同时也通过通信端口而传送出去。当按下“ESC”键则结束程序执行。

程序清单

```
1  /* SE.C */
2  /*  PC <~> C8051 RS232 COM2  <9600 N 8 1> */
3  #include <bios.h>
4  #include <stdio.h>
5
6  #define PROTOCOL 0xe3  /*定义 RS232 通信协议*/
7  #define PROT      0
8  #define TX        1
9  #define RX        2
10 #define STATUS    3
11 int port=1; /* 系统通信端口使用 com2 */
12 /*-----*/
13 main()
14 {
15     int s;
16     unsigned char c;
17     /* 设定 RS232 通信协议 */
18     bioscom(PROT, PROTOCOL, port);
19
20     clrscr(); /* 显示工作画面 */
21     puts("-----");
22     puts("SE.EXE    PC RS232  COM2  <9600 N 8 1> ");
23     puts("-----");
24
25     while(1)  /* 循环 */
26     {
27         /* 检查接收数据是否准备好 */
28         s=bioscom(STATUS, 0, port) & 0x100;
29         if(s)
30         { /* 接收数据进来 */
31             c=bioscom(RX, 0, port);
32             printf("%c",c); /* 显示在屏幕上 */
33         }
34
35         /* 检查 PC 上有按下任何键 */
```

```

36  if( kbhit() )
37  {
38      c=getch(); /* 读取按键 */
39      switch(c)
40      {
41          case '\': /*空白键则将清除屏幕 */
42              clrscr(); /* 显示工作画面 */
43              puts("-----");
44              puts("SE.EXE    PC RS232  COM2  <9600 N 8 1>");
45              puts("-----");
46              break;
47          /* 若是按下"ESC"键则结束程序执行 */
48          case 27 :  exit(0);  break;
49          default :  printf("%c",c); /* 按键值印出来 */
50                      bioscom(TX, c, port); /* 由 RS232 送出 */
51                      break;
52      }
53  }
54  }
55
56  }
57  /*-----*/

```

有时候若我们希望 RS232 通信端口改为 com1, 同时也能改变传输率的波特率值, 以方便做不同的实验, SEV.C 便是一个这样的程序, 基本上是由以上介绍的 SE.C 程序修改而来。

执行结果

可执行文件为 SEV.EXE, 执行后, 出现画面, 告诉 SEV 的线上操作指令:

```

SEV  usage :
SEV  com_port  baud_rate

com_port  =  0 -- com 1
            1 -- com 2
baud_rate =  0 -- 1200
            1 -- 2400
            2 -- 4800

```

3 -- 9600

例如我们希望使用 COM1 作为通信端口, 波特率为 9600 bps, 则可输入:

SEV 0 3 <ENTER>

出现画面:

```
-----
SEV.EXE  PC RS232  COM 1 <9600 N 8 1>
-----
```

则进入 RS232 监控程序中, 其操作方法同 SE.EXE。

程序清单

```
1  /* SEV.C */
2  /*  PC <--> C8051 RS232  <bps N 8 1> */
3
4  /* usage :  SEV
5      SEV  com_port  baud_rate  */
6
7  #include <bios.h>
8  #include <stdio.h>
9
10 #define PROT    0  /*定义 RS232 通信协议*/
11 #define TX      1
12 #define RX      2
13 #define STATUS  3
14
15 int port=1; /* 系统通信端口使用 com2 */
16 unsigned char pro;
17 int bps;
18
19 /*-----*/
20 main(argc,argv)
21 int argc;
22 char *argv[];
23 {
24 int s;
25 unsigned char c, b;
```

```
26 /* 指令格式不符合, 则显示操作说明 */
27 if(argc!=3) { help(); exit(0); }
28 /* 将输入的第 2 个参数的第 1 个字转换为通信端口的值 */
29 port=argv[1][0]-0x30;
30 if(port>1) port=1; /* COM2 */
31 /* 读取第 3 个参数的第一个字符转换为波特率值 */
32 b=argv[2][0]-0x30;
33 switch(b)
34 {
35 case 0 : pro=0x83; bps=1200; break;
36 case 1 : pro=0xA3; bps=2400; break;
37 case 2 : pro=0xC3; bps=4800; break;
38 case 3 : pro=0xE3; bps=9600; break;
39 default: pro=0xE3; bps=9600; break;
40 }
41 /* 设定 RS232 通信协议 */
42 bioscom(PROT, pro, port);
43
44 clrscr(); /* 显示工作画面 */
45 puts("-----");
46 printf("SEV.EXE   PC RS232  COM %d <%d N 8 1>\n",
        port+1, bps);
47 puts("-----");
48
49 while(1) /* 循环 */
50 {
51 /* 检查接收数据是否准备好 */
52 s=bioscom(STATUS, 0, port) & 0x100;
53 if(s)
54 { /* 接收数据进来 */
55 c=bioscom(RX, 0, port);
56 printf("%c",c); /* 显示在屏幕上 */
57 }
58
59 /* 检查 PC 上是否有按下任何键 */
60 if( kbhit() )
```

```
61 {
62     c=getch(); /* 读取按键 */
63     switch(c)
64     {
65         case ' ': /*空白键则将清除屏幕 */
66             clrscr(); /* 显示工作画面 */
67             puts("-----");
68             printf("SEV.EXE   PC RS232   COM %d <%d N 8 1>\n",
                    port+1, bps);
69             puts("-----");
70             break;
71         /* 若是按下"ESC"键则结束程序执行 */
72         case 27 : exit(0); break;
73         default : printf("%c", c); /* 按键值打印出来 */
74                     bioscom(TX, c, port); /* 由 RS232 送出 */
75                     break;
76     }
77 }
78 }
79
80 }
81 /*-----*/
82 help() /* 在线操作指令说明 */
83 {
84     puts("SEV  usage : ");
85     puts("SEV  com_port  baud_rate  \n");
86     puts("      com_port  = 0 -- com 1");
87     puts("                1 -- com 2");
88     puts("      baud_rate = 0 -- 1200");
89     puts("                1 -- 2400");
90     puts("                2 -- 4800");
91     puts("                3 -- 9600");
92 }
```


10.6 串行传送驱动程序

在介绍过 8051 串行传送接口相关控制寄存器及波特率的设置后，本节以 C 语言来设计串行传送的驱动程序。

10.6.1 初始化串行通信端口

在初始化串行传输接口时有以下 3 个步骤：假设传输协议为 9600bps，传送 8 个位数据，没有校验位，1 个停止位。

步骤 1：设定控制寄存器 SCON

以串行传输模式 1 做数据传送，并允许接收，则相对 SCON 寄存器值可以做如下设定：

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
0	1	0	1	0	0	0	0

其值为 0X50，C 语言语句为：

SCON=0X50;

步骤 2：设定定时器 1 工作模式

规划 TMOD 寄存器，使用定时器 1，工作在模式 2，自动重新载入计数值。

GATE	C/T	M1	M0	GATE	C/T	M1	M0
0	0	1	0	0	0	0	0
└── 定时器 1 ─┘				└── 定时器 0 ─┘			

其值为 0X20，C 语言语句为：

TMOD=0X20;

有关定时器 1 的工作可参考第 9 章说明。

步骤 3：设定波特率

在 8051 单板上石英晶振采用 11.0592MHz 作为系统工作时钟，波特率为 9600bps，所以设定定时器 1 为重新载入 253 (0XFD)，对 TH1 写入计数值，而 TL1 可以不管。程序为：

TH1=0XFD;

步骤 4：启动定时器 1

启动定时器 1 使能正确地产生波特率时钟，用如下指令控制：

```
setbit(TCON.6);
```

令特殊功能寄存器 TCON 的位 6 (TR1) 变为 1。

因此将上述的 4 条指令合并, 可以完成 8051 串行端口的初始化工作。

```
init_rs232()
{
    SCON=0x50;
    TMOD=0x20;
    TH1 =0xFD;
    setbit(TCON.6); /* TR1=1 */
    setbit(SCON.1); /* TI=1 */
}
```

10.6.2 传送数据

8051 串行传输模式 1 发送数据时, 一次传送 10 个位, 其中包括 1 个起始位, 8 个数据位, 最后是 1 个位的结束位, 可以没有校验检查位。在做完串行端口的初始化工作后便可以传送数据。当 SCON 的位 1 (TI) 变为 1 时, 表示 10 个串行位均已传送完毕, 可以再送出下一笔串行数据了。以程序来控制时便是等待 TI 位 (SCON.1) 转变为 1, 便将数据写入 SBUF 寄存器内, 并将 TI 位清除为 0。传送子程序设计如下:

```
tx_char(unsigned char c)
{
    while(1)
        if (SCON & 0x02)==0x02) break; /* TI=1 */
    clrbit(SCON.1); /* TI=0; */
    SBUF=c;
}
```

有了串行传输的初始化及传送子程序, 便可以编写一程序来测试 8051 传送的功能, 程序 STX.C 为一个 8051 串行数据发送程序, 为了方便用示波器观察串行信号送出的波形, 在每次传送一个字符串后, 使工作 LED (P1.7) 反向一次, 如果要观察串行信号可以此信号作为示波器的同步参考信号用。

执行结果

首先在 PC 上执行 SE.EXE (假设通信端口是使用 COM2), 然后将程序代码载入 8051 单板上测试, 单板上首先送出 “test” 字样, 并以两种传输字符串的方式送出一段消息。

程序清单

```
1 /* STX.C */
```

```
2 #include "8051io.h" /* 载入 MC51 头文件 */
3 #include "8051reg.h"
4 #include "8051bit.h"
5 #include "8051int.h"
6 #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7
8 char *title="MC51 test P51_PCB serial port TX data";
9
10 delay(int t) /* 延迟子程序 */
11 {
12     char i,j;
13     for(i=0; i<t; i++)
14         for(j=0; j<10; j++);
15 }
16 /*-----*/
17 led_bl() /* 工作 LED 闪动 */
18 {
19     char i;
20
21     for(i=0; i<4; i++)
22     {
23         cplbit(P1.7);
24         delay(40);
25     }
26 }
27 /*-----*/
28 /* b7 b6 b5 b4 b3 b2 b1 b0
29 TCON| TF1 TR1 TF0 TR0 IE1 IT1 IE0 IT0
30 SCON| SM0 SM1 SM2 REN TB8 Rb8 TI RI
31 */
32 /* 初始化串行端口 */
33 init_rs232() /* 通信协议: <9600 N 8 1> */
34 {
35     SCON=0x50; /* 设定串行接口工作于模式 1, 允许接收数据 */
36     TMOD=0x20; /* 设定定时器 1 工作于模式 2 */
37     TH1 =0xFD; /* 设定波特率为 9600 BPS */
```

```
38  setbit(TCON.6) /* TR1=1 启动定时器 1 开始计数 */
39  setbit(SCON.1); /* TI=1 发送准备好 */
40  }
41  /*-----*/
42  tx_char(unsigned char c) /* 送出一个字符 */
43  {
44      while(1) /* 循环 */
45      /* 判断 TI 是否为 1 */
46      if( (SCON & 0x02)==0x02) break;
47      /* 清除发送中断标志 TI=0; */
48      clrbit(SCON.1);
49      SBUF=c; /* 将字符送至串行输出缓冲器 */
50      cplbit(P1.7); /* P1.7 位取反 */
51  }
52  /*-----*/
53  tx_str(char *str) /* 送出字符串 */
54  {
55      char i;
56      for(i=0; i<strlen(str); i++)
57          tx_char(str[i]);
58  }
59  /*-----*/
60  tx_str1(char *str) /* 以指针的方式送出字符串 */
61  {
62      do{ tx_char(*str++); }
63      while(*str!='\0');
64  }
65  /*-----*/
66  main()
67  {
68      led_b1(); /* 工作 LED 闪动 */
69      init_rs232(); /* 初始化串行端口 */
70      /* 字符输出测试 */
71      tx_char('t'); tx_char('e');
72      tx_char('s'); tx_char('t');
73      tx_char('\n'); tx_char('\r');
```

```

74    tx_str(title); /* 字符串输出测试 */
75
76    tx_char('\n'); tx_char('\r');
77    tx_str1(title); /* 字符串输出测试 */
78    while(1) {} /* 无穷循环 */
79 }
80 /*-----*/

```

10.6.3 接收数据

8051 串行数据的接收控制也是使用串行传输模式 1 来接收数据，一次接收 10 个位，在对串行端口初始化后，便可做数据的接收。当 SCON 寄存器位 0 (RI) 变为 1 时，表示已有一笔完整的串行数据进入 SBUF 中，可以将其取回。所以程序只要判断 RI 位转变为 1 时，便可从 SBUF 中读出数据，再将 RI 位清除为 0。接收子程序设计如下：

```

char rx_char()
{
    while(1)
        if (SCON & 0x01) break;
        clrbit(SCON.0); /* RI=0; */
    return SBUF;
}

```

程序 SRX.C 便是一个 8051 串行端口接收测试程序。

执行结果

首先 PC 上执行 SE.EXE，在 PC 键盘上输入的任何字，都会经过 RS232 通信端口而传送到 8051 单板上，因此 8051 随时监测是否有任何数据进入串行端口中，若有则接收进来，并将其传回 PC 而显示出来。所以从 PC 上输入的任何字符，除了显示在屏幕上，其后的相同字符则是由 8051 在接收后传回来的字符，若 8051 接收及传送无误，在 PC 上输入“1”，应出现 2 个“1”才对。

程序清单

```

1  /*  SRX.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7

```

```
8 char *title="MC51 test Pr1_PCB serial port RX data";
9
10 delay(int t) /* 延迟子程序 */
11 {
12     char i,j;
13     for(i=0; i<t; i++)
14         for(j=0; j<10; j++);
15 }
16 /*-----*/
17 led_bl() /* 工作 LED 闪动 */
18 {
19     char i;
20
21     for(i=0; i<4; i++)
22     {
23         clrbit(P1.7);
24         delay(40);
25     }
26 }
27 /*-----*/
28 /* b7 b6 b5 b4 b3 b2 b1 b0
29 TCON| TF1 TR1 TF0 TR0 IE1 IT1 IE0 IT0
30 SCON| SM0 SM1 SM2 REN TB8 Rb8 TI RI
31 */
32 /* 初始化串行端口 */
33 init_rs232() /* 通信协议: <9600 N 8 1> */
34 {
35     SCON=0x50; /* 设定串行接口工作于模式 1, 允许接收数据 */
36     TMOD=0x20; /* 设定定时器 1 工作于模式 2 */
37     TH1 =0xFD; /* 设定波特率为 9600 BPS */
38     setbit(TCON.6); /* TR1=1 启动定时器 1 开始计数 */
39     setbit(SCON.1); /* TI=1 发送准备好 */
40 }
41 /*-----*/
42 tx_char(unsigned char c) /* 送出一个字符 */
43 {
```

```
44 while(1) /* 循环 */
45 /* 判断 TI 是否为 1 */
46 if( (SCON & 0x02)==0x02) break;
47 /* 清除发射中断标志 TI=0; */
48 clrbit(SCON.1);
49 SBUF=c; /* 将字符送至串行输出缓冲器 */
50 }
51 /*-----*/
52 tx_str(char *str) /* 送出字符串 */
53 {
54 char i;
55 for(i=0; i<strlen(str); i++)
56 tx_char(str[i]);
57 }
58 /*-----*/
59 char rx_char() /*由串行端口接收字符进来*/
60 {
61 while(1) /* 循环 */
62 /* 判断数据接收是否准备好 */
63 if( (SCON & 0x01)==0x01) break;;
64 /* 清除 RI 标志 */
65 clrbit(SCON.0);
66 return SBUF; /*由串行端口接收数据进来*/
67 }
68 /*-----*/
69 main()
70 {
71 char c;
72 led_bl(); /* 工作 LED 闪动 */
73 init_rs232(); /* 初始化串行端口 */
74
75 tx_str(title); /* 字符串输出测试 */
76 tx_char('\r');
77 tx_char('\n');
78 while(1) /* 无穷循环 */
79 {
```

```

80  c=rx_char(); /* 接收字符 */
81  tx_char(c); /* 送出字符*/
82  }
83  }
84  /*-----*/

```

10.7 使用 MICRO C51 函数

在 8051 串行接口方面, MC51 编译器本身的函数库中提供有 3 个函数, 可供使用者自行应用:

- 1.serinit(): 串行端口的初始化设定;
- 2.printf() : 将数据送往串行端口;
- 3.getch() : 由串行端口输入一个字节数据。

在 PC 上 TURBO C printf()函数的操作是将数据送往标准输出设备, 即屏幕, 而在 8051 单片机上因为已经配备有现成的串行输出端口, 因此顺理成章地将所要送出的数据转到串行端口上, 只要将串行端口与 PC 连接, 便可以在屏幕上看到所传送出来的数据。而在 TURBO C 中, getch()函数是接收一个键盘输入的数据, 作为数据的输入, 同理在 8051 上直接利用串行端口来输入数据则相当的方便。只要先执行 serinit()函数做串行端口的初始化工作, 便可利用 getch()函数来输入数据, printf()函数来输出数据。

10.7.1 由串行端口输出数据

pr.c 为一个测试 MICRO C-51 函数 printf()功能的程序。

执行结果

执行后我们可以看到 MC51 对各种数据类型处理的表示方法, 执行结果如下:

(在 PC 上记得先执行 SE.EXE)

```

-----
SE.EXE   PC RS232  COM2  <9600 N 8 1>
-----

```

MC51 test printf ...

<0> test string

string --> MC51 test P51_PCB printf

<1> show char. 0x41

binary --> 1000001b

char --> A

decimal --> 65


```

    octal    --> 101o
    unsigned --> 65
    hex      --> 41h
<2> show value <char>...
    decimal --> -128 ~ 127
<3> show value <unsigned char>...
    decimal --> 0 ~ 255
<4> show value ...
    decimal --> -32768 ~ 32767
<5> show value <unsigned int>...
    decimal --> 0 ~ 65535
<6> show sprintf...
    decimal --> 0 ~ 65535

```

程序清单

```

1  /* PR.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  char *title="MC51 test P51_PCB printf";
9
10 char c, c1;
11 unsigned char c2, c3;
12 int c4, c5;
13 unsigned int c6, c7;
14
15 delay(int t) /* 延迟子程序 */
16 {
17     char i,j;
18     for(i=0; i<t; i++)
19         for(j=0; j<20; j++);
20 }
21 /*-----*/
22 led_b1() /* 工作 LED 闪动*/

```

```
23 {
24 char i;
25
26 for(i=0; i<4; i++)
27 {
28     clrbit(P1.7);
29     delay(40);
30 }
31 }
32 /*-----*/
33 main()
34 {
35 char mess[30];
36
37 led_bl();      /* 工作 LED 闪动 */
38 serinit(9600); /* 初始化串行端口 */
39 printf("MC51 test printf ...\n");
40 /* 送出字符串数据 */
41 printf("<0> test string ..... \n");
42 printf("    string --> %s\n", title);
43 /* 送出字符数据 0X41, 并以不同的数据表示方式来显示结果 */
44 c=0x41;
45 printf("<1> show char. 0x41 ..... \n");
46 printf("    binary --> %bb\n", c);
47 printf("    char   --> %c\n", c);
48 printf("    decimal --> %d\n", c);
49
50 printf("    octal   --> %oo\n", c);
51 printf("    unsigned --> %u\n", c);
52 printf("    hex      --> %xh\n", c);
53 /* 输出带符号的字符数据 */
54 c=-128; c1=127;
55 printf("<2> show value <char>... \n");
56 printf("    decimal --> %d ~%d \n", c, c1);
57 /* 输出无符号的数据 */
58 c2=0; c3=255;
```

```

59    printf("<3> show value <unsigned char>...\n");
60    printf("    decimal --> %d ~ %d \n", c2, c3);
61    /* 输出带符号整型的数据 */
62    c4=-32768; c5=32767;
63    printf("<4> show value <int>...\n");
64    printf("    decimal --> %d ~ %d \n", c4, c5);
65    /* 输出无符号整型的数据 */
66    c6=0; c7=65535;
67    printf("<5> show value <unsigned int>...\n");
68    printf("    decimal --> %d ~ %u \n", c6, c7);
69    printf("\n");
70    /* 测试函数 sprintf() 的使用方法 */
71    printf("<6> show sprintf...\n");
72    sprintf(mess, "    decimal --> %d ~ %u \n", c6, c7);
73    printf(mess);
74    while(1);
75 }

```

由串行端口输入数据

get.c 是一个用来展示函数 getch()功能的程序。

执行结果

在 PC 上首先执行 SE.EXE，在与 8051 单板取得连接后，从 PC 上输入的任何字符都会经过 RS232 接口传至 8051 单板上，8051 接收到字符数据后会传回 PC 以确定接收无误。若按下“1”则屏幕上会出现两个“1”，若按下“q”键，则结束程序测试。

程序清单

```

1  /* GET.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  char *title="MC51 test P51_PCB serial port getch()";
9
10 delay(int t) /* 延迟子程序 */

```

```
11 {
12   char i,j;
13   for(i=0; i<t; i++)
14     for(j=0; j<10; j++);
15 }
16 /*-----*/
17 led_bl() /* 工作 LED 闪动 */
18 {
19   char i;
20
21   for(i=0; i<4; i++)
22   {
23     cplbit(P1.7);
24     delay(50);
25   }
26 }
27 /*-----*/
28 main()
29 {
30   char c;
31
32   led_bl(); /* 工作 LED 闪动 */
33   serinit(9600); /* 初始化串行端口 */
34
35   printf(title); /* 显示工作消息 */
36   printf("\n");
37   printf("Please key in ... 'q' to exit \n");
38   while(1) /* 无穷循环 */
39   {
40     c=getch(); /* 等待按键输入 */
41     if(c=='q') break; /* 若按下"q" 键则结束程序测试*/
42     else printf("P51_PCB RX  %c\n",c);
43   }
44   printf("exit getch test !  LED blinking ....");
45   while(1) /* 无穷循环 */
46   led_bl(); /* 工作 LED 闪动 */
```

```

47 }
48 /*-----*/

```

10.8 输入一字符串

在上一节中我们看过函数 `getch()` 的用途，其功能是由串行端口接收一个字符，而将数个字符汇集在一起便可形成字符串，通常在 PC 上输入的各个字符我们先将它接收进来放入一个数组中，直到接收到 `<ENTER>` 键时(代码为 `0X0A`)，则将完整的字符串送出去，子程序 `gets()`，即是这样的一个函数。

执行结果

`gets.c` 是字符串接收函数的测试程序，PC 输入的字符串会经过 RS232 接口传至 8051 单板上，8051 接收到字符串数据后也会传回 PC，以确定接收无误。

程序清单

```

1  /* GETS.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  char *title="MC51 test P51_PCB function gets()";
9
10 delay(int t) /* 延迟子程序 */
11 {
12     char i,j;
13     for(i=0; i<t; i++)
14         for(j=0; j<10; j++);
15 }
16 /*-----*/
17 led_bl() /* 工作 LED 闪动 */
18 {
19     char i;
20
21     for(i=0; i<4; i++)
22     {

```

```
23     cplbit(P1.7);
24     delay(50);
25 }
26 }
27 /*-----*/
28 gets(char *str) /* 字符串接收子程序 */
29 {
30     char c;
31
32     while(1) /* 无穷循环 */
33     {
34         c=getch(); /* 等待接收字符 */
35         if(c==0x0a) break; /* 接收到<ENTER> 键 */
36         *str++=c;
37     } /* 在字符串的最后，并加上结束代码*/
38     *str='\0';
39 }
40 /*-----*/
41 main()
42 {
43     char str[80];
44
45     led_bl(); /* 工作 LED 闪动 */
46     serinit(9600); /* 初始化串行端口 */
47     printf("%s\n",title);
48
49     while(1) /* 无穷循环 */
50     {
51         printf("Please type in string \n");
52         gets(str); /* 字符串接收测试 */
53         printf("\nP51_PCB rx string :  %s\n", str);
54     }
55 }
56 /*-----*/
```

10.9 输入一数字

gets()子程序可以由 8051 串行端口接收字符串, 若将字符串的内容加以转换, 便能够直接由 PC 输入数字数据了。例如送进来的字符串是“1234”, 若将其解释为数字便是 1234 了。其中我们利用 MC51 程序库所提供的函数 atoi(str), 将送进来的字符串 str 内容转换为整数输出。

执行结果

atoi.c 为其测试程序。首先会提示从 PC 上输入一整数, 再要求输入一无符号整数, 然后分别显示出由 8051 接收后的数值, 执行例子如下:

```
-----
SE.EXE   PC   RS232  COM2  <9600 N 8 1>
-----
```

```
MC51    test P51_PCB function atoi()
Please key in an integer ? 1234
P51_PCB rx value : 1234
Please key in an unsigned integer ? 65000
P51_PCB rx value : 65000
Please key in an integer ? -1234
P51_PCB rx value : -1234
Please key in an unsigned integer ? 0
P51_PCB rx value : 0
```

程序清单

```
1  /* ATOLC */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  char *title="MC51 test P51_PCB function atoi()";
9
10 delay(int t) /* 延迟子程序 */
11 {
12     char i,j;
13     for(i=0; i<t; i++)
```

```
14     for(j=0; j<10; j++);
15 }
16 /*-----*/
17 led_bl() /* 工作 LED 闪动 */
18 {
19     char i;
20
21     for(i=0; i<4; i++)
22     {
23         cplbit(P1.7);
24         delay(50);
25     }
26 }
27 /*-----*/
28 gets(char *str) /* 字符串接收子程序 */
29 {
30     char c;
31
32     while(1) /* 无穷循环 */
33     {
34         c=getch(); /* 等待接收字符 */
35         if(c==0x0a) break; /* 接收到<ENTER> 键 */
36         *str++=c;
37     } /* 在字符串的最后，并加上结束代码*/
38     *str='\0';
39 }
40 /*-----*/
41 main()
42 {
43     char str[80];
44     int d;
45     unsigned int d1;
46
47     led_bl(); /* 工作 LED 闪动 */
48     serinit(9600); /* 初始化串行端口 */
49     printf("%s\n",title);
```



```

50
51 while(1) /* 无穷循环 */
52 { /* 提示输入一整数 */
53     printf("Please key in an integer ? ");
54     gets(str); /* 等待字符串输入 */
55     d=atoi(str); /* 字符串的内容加以转换 */
56     printf("\nP51_PCB rx value : %d\n", d);
57 /* 提示输入一无符号的整数 */
58     printf("Please key in an unsigned integer ? ");
59     gets(str); /* 等待字符串输入 */
60     d1=atoi(str); /* 字符串的内容加以转换 */
61     printf("\nP51_PCB rx value : %u\n", d1);
62 }
63 }
64 /*-----*/

```

10.10 建立交互式的 8051 系统开发环境

8051 单片机所提供的串行传输接口端口一般的使用情况有以下几种:

- 1.与外界做数据传输;
- 2.通过串行接口使系统除错容易;
- 3.做多处理机设计;
- 4.做额外硬件扩充。

因此在专题设计中若没有使用串行传输接口实在太可惜了,加上串行接口后使得一些系统软件硬件功能开发及除错变得相当的方便。在本节我们举个简单的例子来说明如何在 8051 单板上建立一套交互式的系统开发环境。

在上面几节中我们已经学会了利用 PC 来传送数据到 8051 上来,也可以将执行结果送回 PC,因此凡是在 8051 上执行完的每一个步骤,我们都可以仔细地在 PC 上观察其执行结果,很多 BUG 将可因此而找到。

执行结果

TA.C 便是这样一个交互式的展示程序,执行时系统提示从 PC 上输入 2 个数字,在单板接收到此 2 数字后,便将其乘积通过串行端口传送回 PC 而显示在屏幕上,看看结果是否正确?执行结果如下:

```
Please key in    an integer d1 ?  11
```

```

P51_PCB rx value1 : 11
Please key in. an integer d2 ? 11
P51_PCB rx value2 : 11
Result : d1 * d2 = 121

```

程序清单

```

1  /* TA.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  char *title="P51_PCB test RS232 talking system ";
9
10 delay(int t) /* 延迟子程序 */
11 {
12     char i,j;
13     for(i=0; i<t; i++)
14         for(j=0; j<10; j++);
15 }
16 /*-----*/
17 led_bl() /* 工作 LED 闪动 */
18 {
19     char i;
20
21     for(i=0; i<4; i++)
22     {
23         cplbit(P1.7);
24         delay(50);
25     }
26 }
27 /*-----*/
28 gets(char *str) /* 字符串接收子程序 */
29 {
30     char c;
31

```

```
32 while(1) /* 无穷循环 */
33 {
34     c=getch(); /* 等待接收字符 */
35     if(c==0x0a) break; /* 接收到<ENTER> 键 */
36     *str++=c;
37 } /* 在字符串的最后，并加上结束代码*/
38 *str='\0';
39 }
40 /*-----*/
41 main()
42 {
43     char str[80];
44     int d, d1, d2;
45
46     led_bl(); /* 工作 LED 闪动 */
47     serinit(9600); /* 初始化串行端口 */
48     printf("%s\n",title); /* 显示工作消息 */
49
50     while(1) /* 无穷循环 */
51     { /* 提示输入整数 d1 */
52         printf("Please key in an integer d1 ? ");
53         gets(str); /* 字符串接收 */
54         d1=atoi(str); /* 将字符串转换为整数*/
55         printf("\nP51_PCB rx value1 :  %d\n", d1);
56         /* 提示输入整数 d1 */
57         printf("Please key in an integer d2 ? ");
58         gets(str); /* 字符串接收 */
59         d2=atoi(str); /* 将字符串转换为整数*/
60         printf("\nP51_PCB rx value2 :  %d\n", d2);
61         /* 计算结果 */
62         d=d1*d2;
63         printf("Result :    d1 * d2 = %d\n", d);
64     }
65 }
66 /*-----*/
```

10.11 习 题

1. 用示波器测量 8051 TXD 引脚所输出的同步脉冲宽度。
2. 写一程序使用通信协议(9600,N,8,1)与 PC 建立连接,当 PC 上按键时,则 8051 做出相应的响应:
 PC 按键 1 --> 8051 响应: "KEY 1 TEST"
 PC 按键 2 --> 8051 响应: "KEY 2 TEST"

第 11 章 LCD 接口控制

在 4.6 节中我们已经介绍过 LCD 的特性及引脚功能，并看过其硬件控制电路，本节将进一步说明 LCD 内部结构，并说明如何设计 LCD 控制程序。

11.1 LCD 内部结构介绍

LCD 内部内存共分为 3 种：

1. 固定字型 ROM，称为 CG (Character Generator) ROM。
2. 数据显示 RAM，称为 DD (Data Display) RAM。
3. 使用者自定义字型 RAM，称为 CG RAM。

11.1.1 CG ROM

CG ROM 内存储着 192 个 5×7 点阵的字型，这些字型均已固定，例如我们将“A”写入 LCD 中，就是将“A”的 ASCII 代码 41H 写入 DD RAM 中，同时到 CG ROM 中将“A”的字型点阵数据找出来显示在 LCD 上。

11.1.2 DD RAM

DD RAM 内用来存储写入 LCD 内部的字符，DD RAM 的地址分布从 00H 到 67H，分别代表 LCD 的各行位置，如图 11-1 所示，例如我们要将“A”写入第 2 行的第 1 个位置，就先设定 DD RAM 地址为 40H，然后写入 41H 至 LCD 即可。

(1) 16 字 \times 1 行

显示位置	0	1	2	...	13	14	15
第 1 行 DDRAM 地址	00H	01H	02H	...	0DH	0EH	0FH

(2) 20 字 \times 2 行

显示位置	0	1	2	...	17	18	19
第 1 行 DDRAM 地址	00H	01H	02H	...	11H	12H	13H
第 2 行 DDRAM 地址	40H	41H	42H	...	51H	52H	53H

图 11-1 LCD 显示数据 RAM 地址

11.1.3 CG RAM

此区域只有 64 字节，可由使用者将自定义的字型写入 LCD 中，一个字的大小为 5×7 点阵，共可以存储 8 个字型，其显示代码为 00H 到 07H。

11.1.4 控制方式

用 CPU 来控制 LCD 模块, 方式十分简单, LCD 模块其内部可以看成两组寄存器, 一个为指令寄存器, 一个为数据寄存器, 由 RS 引脚来控制。所有对指令寄存器或数据寄存器的存取均需检查 LCD 内部的忙碌标志 (Busy Flag), 此标志用来告知 LCD 内部正在工作, 并不允许接收任何的控制命令。而此位的检查可以令 RS=0, 用读取位 7 来加以判断, 当此位为 0 时, 才可以写入指令或数据寄存器。

11.1.5 LCD 控制指令

LCD 控制指令有以下几项:

1. 清除显示器

指令代码为 0X01, 将 LCD DD RAM 数据全部填入空白代码 20H, 执行此指令将清除显示器的内容, 同时光标移到左上角。

2. 光标归位设定

指令代码为 0X02, 地址计数器被清 0, DD RAM 数据不变, 光标移到左上角。

3. 设定字符进入模式

指令格式为:

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	1	I/D	S

I/D: 地址计数器递增或递减控制, I/D=1 时为递增, I/D=0 时为递减。每次读写显示 RAM 中字符代码一次则地址计数器会加一或减一。光标所显示的位置也会同时向右移一个位置 (I/D=1) 或向左移一个位置 (I/D=0)。

S: 显示屏移动或不移动控制, 当 S=1, 写入一个字符到 DD RAM 时, 显示屏向左 (I/D=1) 或向右 (I/D=0) 移动一格, 而光标位置不变。当 S=0 时, 则显示屏不移动。

4. 显示器开关

指令格式为:

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	1	D	C	B

D: 显示屏开启或关闭控制位, D=1 时, 显示屏开启, D=0 时, 则显示屏关闭。

C: 光标出现控制位, C=1 时, 则光标会出现在地址计数器所指的位置, C=0 则光标不出现。

B: 光标闪烁控制位, B=1 光标出现后会闪烁, B=0, 光标不闪烁。

5. 显示光标移位

指令格式为:

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	1	S/C	R/L	X	X

X 表示 0 或 1 都可。

S/C	R/L	工作
0	0	光标向左移
0	1	光标向右移
1	0	字符和光标向左移
1	1	字符和光标向右移

6. 功能设定

指令格式为：

B7	B6	B5	B4	B3	B2	B1	B0
0	0	1	DL	N	F	X	X

DL : 数据长度选择位。

DL=1 时为 8 位数据转移, DL=0 时则为 4 位数据转移, 使用 D7~D4 4 个位, 分 2 次送入一个完整的字符数据。

N : 显示屏为单行或双行选择。

N=1 为单行显示, N=0 则为双行显示。

F : 大小字符显示选择。

当 F=1 时, 为 5×10 点阵字会大些, 当 F=0 时, 则为 5×7 点阵字型。

7. CG RAM 地址设定

指令格式为：

B7	B6	B5	B4	B3	B2	B1	B0
0	1	A5	A4	A3	A2	A1	A0

设定 CG RAM 地址为 6 位的地址值, 便可对 CG RAM 读/写数据。

8. DD RAM 地址设定

指令格式为：

B7	B6	B5	B4	B3	B2	B1	B0
1	A6	A5	A4	A3	A2	A1	A0

设定 DD RAM 为 7 位的地址值, 便可对 DD RAM 读/写数据。

9. 忙碌标志读取

指令格式为：

B7	B6	B5	B4	B3	B2	B1	B0
BF	A6	A5	A4	A3	A2	A1	A0

LCD 的忙碌标志 BF 用以指示 LCD 目前的工作情况, 当 BF=1 时, 表示正在做内部数据的处理, 不接受外界送来的指令或数据。当 BF=0 时, 则表示已准备接收命令或数据。当程式读取此数据的内容时, 位 7 表示忙碌标志, 而另外 7 个位的地址值表示 CG RAM 或 DD RAM 中的地址, 至于是指向那一地址则根据最后写入的地址设定指令而定。

10. 写数据到 CG RAM 或 DD RAM 中

先设定 CG RAM 或 DD RAM 地址, 再将数据写入其中。

11. 从 CG RAM 或 DD RAM 中读取数据

先设定 CG RAM 或 DD RAM 地址, 再读取其中的数据。

11.2 LCD 驱动子程序

看过前面 LCD 特性及控制线路的介绍后, 在本节我们将说明有关 LCD 驱动程序的设计, 主要有 3 个子程序。

write_com(): 写命令到 LCD;
write_data(): 写数据到 LCD;
init_lcd(): 对 LCD 做初始化的工作。

11.2.1 写命令到 LCD

经过命令端口(lcd_com)将命令写至 LCD, 其间必需先查看 LCD 忙碌标志, 可令 RS 引脚为 0, 读取位 7 来判断(即读取 lcd_com 端口位 7), 若为 0, 表示 LCD 闲置着, 可以接受命令的写入。

```
write_com(unsigned char c)
{
    unsigned char in;

    while(1)/* 循环 */
    {
        /* 读取状态端口 */
        in=lcd_com;
        /* 判断 LCD 是否空闲 */
        if( (in & 0x80)==0 ) break;
    }
    lcd_com= c; /* 写命令至 LCD */
}
```



```
}
```

11.2.2 写数据至 LCD

将数据经过数据端口 (lcd_data) 写入 LCD, 同样是先读取忙碌标志是否为 0, 若为 0 才将数据写入 LCD 内。

```
write_data(unsigned char d)
{
    unsigned char in;
    while(1)/* 循环 */
    {
        /* 读取状态端口 */
        in=lcd_com;
        /* 判断 LCD 是否空闲 */
        if ( (in & 0x80) ==0) break;
    }
    lcd_data=d; /* 写命令至 LCD */
}
```

10.2.3 初始化 LCD

对 LCD 做初始化工作包括以下 4 项:

1. 选择 LCD 显示功能;
2. 设定 LCD 显示方式;
3. 设定 LCD 字符进入模式;
4. 清除显示屏。

在此我们设定 LCD 为如下的工作模式:

1. LCD 数据转移采用 8 位形式, 双行显示, 字型使用 5×10 点阵, 控制代码为 0X3C。
2. 打开显示屏, 同时出现光标, 但并不闪烁, 控制代码为 0X0E。
3. LCD 字符每次向右移一位, 但显示屏不移动, 控制代码为 0X06。
4. 清除 LCD 显示屏, 控制代码为 0X01。

```
init_lcd()
{
    write_com(0x3c); /* 双行显示, 字型使用 5×10 点阵 */
    write_com(0x0e); /* 出现光标, 不闪烁 */
    write_com(0x06); /* 每次向右移一位, 显示屏不移动 */
    write_com(0x01); /* 清除 LCD 显示屏 */
}
```

11.3 LCD 显示器测试

上一节说明过有关 LCD 3 个驱动程序的设计, 本节利用这些子程序来写控制程序测试 LCD 基本显示功能。

执行结果

程序执行后, LCD 显示屏出现:

```
P51_PCB LCD TEST >>>
0123456789
```

程序清单

```
1  /* LCD.C */
2  #include "8051io.h"    /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  char *title="Test P51_PCB LCD";
9  char mess[40]; /* LCD 数据输出缓冲区 */
10 /*-----*/
11 delay(int t) /* 延迟子程序 */
12 {
13     int i,j;
14     for(i=0; i<t; i++)
15         for(j=0; j<10; j++);
16 }
17 /*-----*/
18 led_bl() /* 工作 LED 闪动 */
19 {
20     /* blink RED led P1.7 */
21     char i;
22
23     for(i=0; i<4; i++)
24     {
25         cplbit(P1.7); /* 位取反 */
26         delay(40);
```

```
27     }
28 }
29 /*-----*/
30 main() /* 主程序 */
31 {
32     led_bl(); /* 工作 LED 闪动 */
33     init_lcd(); /* 初始化 LCD */
34     serinit(9600); /* 初始化串行接口 */
35     printf(title); /* 显示工作消息 */
36     test_lcd(); /* 测试 LCD 接口 */
37     while(1){} /* 无穷循环 */
38 }
39 /*-----*/
40 write_com(unsigned char c) /* 写命令至 LCD */
41 {
42     unsigned char in;
43     while(1) /* 循环 */
44     { in=lcd_com; /* 读取状态端口 */
45       if( (in & 0x80)==0 ) break; }
46     lcd_com= c; /* 写命令至 LCD */
47 }
48 /*-----*/
49 write_data(unsigned char d) /* 写数据至 LCD */
50 {
51     unsigned char in;
52     while(1) /* 循环 */
53     { in=lcd_com; /* 读取状态端口 */
54       if( (in & 0x80) ==0 ) break; }
55     lcd_data= d; /* 写数据至 LCD */
56 }
57 /*-----*/
58 init_lcd() /* 初始化 LCD */
59 {
60     write_com(0x3c); /* 双行显示, 字型使用 5×10 点阵 */
61     write_com(0x0e); /* 出现光标, 不闪烁 */
62     write_com(0x06); /* 每次向右移一位, 显示屏不移动 */
```

```
63   write_com(0x01); /* 清除 LCD 显示屏 */
64   delay(100);
65 }
66 /*-----*/
67 print(char line, char *str)
68 {
69   char i;
70   if(line==1)
71   { write_com(0x80); /* 移至第 1 行首 */
72     /* 清除该行文字 */
73     for(i=0; i<24; i++) write_data(' ');
74     write_com(0x80); /* 移至第 1 行首 */ }
75   else
76   { write_com(0xc0); /* 移至第 2 行首 */
77     /* 清除该行文字 */
78     for(i=0; i<24; i++) write_data(' ');
79     write_com(0xc0); /* 移至第 2 行首 */ }
80   i=0; /* 将字符串数据写至 LCD */
81   do{ write_data(*str++); }
82   while(*str!='\0');
83 }
84 /*-----*/
85 test_lcd() /* 测试 LCD 显示功能 */
86 {
87   char i;
88
89   sprintf(mess, "P51_PCB LCD TEST >>>>>>>>>>>>");
90   print(1, mess); /* 由第 1 行输出文字 */
91   write_com(0xc0); /* 移至第 2 行首 */
92   /* 写入数字 0~9 */
93   for(i=0; i<10; i++)
94     write_data(i+0x30);
95   delay(1500); /* 延迟一下 */
96   led_bl(); /* 工作 LED 闪动 */
97   sprintf(mess, "TEST OVER >>>>>>>>>>>>");
98   print(2, mess); /* 由第 2 行输出文字 */
```

```

99  }
100 /*-----*/

```

11.4 自定义 LCD 字型

在前面介绍 LCD 内部内存时曾提到过 CG RAM 的位置, 此区域用来存储使用者自定义的字型, 共可以存储 8 个字, 而每一个字的大小为 5×8 点阵, 而显示代码的编号为 00H 到 07H, 例如要将编号 0 的字显示出来, 只要将 00H 写到 DD RAM 内, 则会将 CG RAM 内地址 00H~07H 所存放的字型显示在 LCD 上, 同理将 01H 写入 DD RAM 内则会取用 CG RAM 地址 08H~0FH 内的字型做显示, 余此类推。

至于怎样自定义字型呢? 可以在一 5×8 的方格内填入自定义字型, 例如要显示 “ㄣ” 字, 如图 11-2 所示, 可以将此造型转换为 8 个字节的数据而存在一个数组内:

```
char pat[8]={ 0x04, 0x08, 0x1f, 0x01, 0x01, 0x09, 0x06, 0x00};
```

其中每个字节的最高 3 个位未使用到, 可以填为 “0”, 用到的点填 “1”, 否则填 “0”。

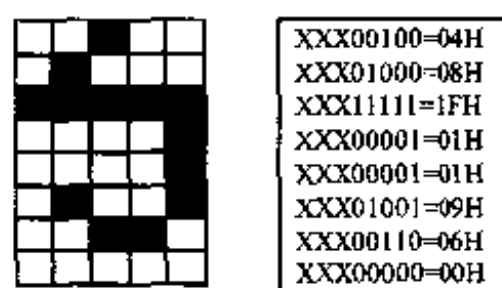


图 11-2 “ㄣ” 的字型设计表

执行结果

程序执行后, 将自定义的字型数据写入 CG RAM 内, 然后将此字型显示在 LCD 上。从 LCD 上显示自定义字型 “ㄣ”:

```

Show .....
ㄣㄣㄣㄣㄣㄣㄣ...

```

程序清单

```

1  /* LCD1.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
7

```

```
8 char *title="Test P51_PCB LCD pattern ";
9 char mess[40]; /* LCD 数据输出缓冲区 */
10 /*-----*/
11 delay(int t) /* 延迟子程序 */
12 {
13     int i,j;
14     for(i=0; i<t; i++)
15         for(j=0; j<10; j++);
16 }
17 /*-----*/
18 led_bl() /* 工作 LED 闪动 */
19 {
20     /* blink RED led P1.7 */
21     char i;
22
23     for(i=0; i<4; i++)
24     {
25         cplbit(P1.7); /* 位取反 */
26         delay(40);
27     }
28 }
29 /*-----*/
30 main() /* 主程序 */
31 {
32     led_bl(); /* 工作 LED 闪动 */
33     init_lcd(); /* 初始化 LCD */
34     serinit(9600); /* 初始化串行接口 */
35     printf(title); /* 显示工作消息 */
36
37     test_lcd_pat(); /* 测试 LCD 接口 */
38     while(1){} /* 无穷循环 */
39 }
40 /*-----*/
41 write_com(unsigned char c)
42 {
43     unsigned char in;
```

```
44 /* 写命令至 LCD */
45 while(1)
46 { in=lcd_com; if( (in & 0x80)==0 ) break; }
47 lcd_com= c;
48 }
49 /*-----*/
50 write_data(unsigned char d)
51 {
52 unsigned char in;
53 /* 写数据至 LCD */
54 while(1)
55 { in=lcd_com; if( (in & 0x80) ==0) break; }
56 lcd_data= d;
57 }
58 /*-----*/
59 init_lcd()
60 {
61 write_com(0x3c); /* 双行显示, 字型使用 5×10 点阵 */
62 write_com(0x0e); /* 光标出现, 不闪烁 */
63 write_com(0x06); /* 每次向右移一位, 显示屏不移动 */
64 write_com(0x01); /* 清除 LCD 显示屏 */
65 delay(100);
66 }
67 /*-----*/
68 print(char line, char *str)
69 {
70 char i;
71
72 if(line==1)
73 { write_com(0x80); /* 移至第一行首 */
74 /* 清除该行文字 */
75 for(i=0; i<24; i++) write_data(' ');
76 write_com(0x80); } /* 移至第 1 行首 */
77 else
78 { write_com(0xc0); /* 移至第 2 行首 */
79 for(i=0; i<24; i++) write_data(' ');
```

```
80 write_com(0xc0); } /* 移至第 2 行首 */
81 i=0; /* 将字符串数据写至 LCD */
82 do{ write_data(*str++); }
83 while(*str!='\0');
84 }
85 /*-----*/
86 test_lcd() /* 测试 LCD 显示功能 */
87 {
88 char i;
89
90 sprintf(mess,"P51_PCB LCD TEST >>>>>>>>>>>>");
91 print(1, mess); /* 由第 1 行输出文字 */
92
93 write_com(0xc0); /* 移至第 2 行首 */
94 /* 写入数字 0~9 */
95 for(i=0; i<10; i++) write_data( i+0x30);
96 delay(1500); /* 延迟一下 */
97 led_bl(); /* 工作 LED 闪动 */
98 sprintf(mess,"TEST OVER >>>>>>>>>>>>");
99 print(2, mess); /* 由第二行输出文字 */
100 }
101 /*-----*/
102 /* 自定义字型数据 */
103 char pat[8]={ 0x04, 0x08, 0x1f, 0x01, 0x01, 0x09, 0x06, 0x00};
104 test_lcd_pat() /* 测试 LCD 显示自定义字型功能 */
105 {
106 int i;
107
108 delay(100);
109 write_com(0x40); /* 设定 CG RAM 地址 */
110 for(i=0; i<8; i++)
111 write_data(pat[i]); /* 写入字型数据 */
112 delay(100);
113 /* 显示 8 个自定义字型"ㄣ" */
114 print(1, "Show ....");
115 for(i=0; i<8; i++) write_data(0);
```


116 }

11.5 习 题

1. 修改原始程序让 LCD 屏幕显示个人学号。
2. 修改原始程序让 LCD 屏幕显示 3 个自定义字型。

第 12 章 单片机 8051 声效设计

这一章我们将介绍 8051 在声效控制的接口, 使用了一片 GI 公司的可编程声效发生器(PSG)AY-3-8910, 此 IC 可以产生各种常见的声效, 如机关枪声、鸟鸣声、子弹飞啸声等声效, 这些声效常常出现在电动玩具上, 大家应该不陌生, 在本章中, 将详细说明其工作原理, 使读者可以自行应用在自己的接口设计及实验中。

12.1 可编程声效发生器内部寄存器分析

在前面我们介绍过可编程声效发生器 PSG 的内部功能, 及如何在单板上以 8051 来控制 PSG 工作的线路, 本节就内部寄存器做分析。

图 12-1 所示为 PSG 内部 16 个寄存器功能列表, 这些寄存器我们需要彻底地研究, 方能正确有效地控制它来产生特殊声效、演奏曲调及做一般的 I/O 控制。

寄存器 \ 位		B7	B6	B5	B4	B3	B2	B1	B0
R0	通道 A 音调周期	8 位微调 A 通道							
R1		4 位粗调 A 通道							
R2	通道 B 音调周期	8 位微调 B 通道							
R3		4 位粗调 B 通道							
R4	通道 C 音调周期	8 位微调 C 通道							
R5		4 位粗调 C 通道							
R6	杂音周期	5 位杂音周期控制							
R7	ENABLE (启用)	IN/OUT		NOISE		TONE			
		PB	PA	C	B	A	C	B	A
R8	通道 A 振幅				M	L3	L2	L1	L0
R9	通道 B 振幅				M	L3	L2	L1	L0
R10	通道 C 振幅				M	L3	L2	L1	L0
R11	包络周期	8 位音调微调							
R12		8 位音调粗调							
R13	包络外形/循环					CONT	ATT	ALT	HOLD
R14	输入/输出端口 A	端口 A 8 位 I/O 数据							
R15	输入/输出端口 B	端口 B 8 位 I/O 数据							

图 12-1 PSG 内部 16 个寄存器

12.1.1 音调控制产生寄存器 R0~R5

由寄存器 R0~R5 来控制三个波道音调的产生, 其中 R0、R2、R4 为音调频率细调(Fine tune)寄存器, 共占 8 位, 而 R1、R3、R5 则为 4 位的音调频率粗调(Coarse tune)寄存器, 共以 12 位宽的数据来决定音调产生的频率。其计算公式为:

$$f_T = f_{\text{CLOCK}} / [16 \times (256 \text{ CT} + \text{FT})]$$

式中: f_T : 希望输出音调频率;
 f_{CLOCK} : 输入至 PSG 的工作频率;
 CT : 粗调寄存器值, 存于 R1、R3、R5 的 4 个位值;
 FT : 细调寄存器值, 存于 R0、R2、R4 的 8 个位值。

12.1.2 噪声产生寄存器 R6

PSG 噪声产生的频率是先将输入工作频率除以 16, 再除以可编程化的噪声周期值, 此值是存于寄存器 R6 中的 5 个位值来表示。

$$f_N = f_{\text{CLOCK}} / [16 \times \text{NP}]$$

式中: f_N : 希望产生的噪音频率;
 f_{CLOCK} : 输入工作频率;
 NP : 噪声周期寄存器 R6 的内含值。

12.1.3 音调/噪声混合及输入/输出启用控制寄存器 R7

寄存器 R7 是一个 PSG 多功能启用寄存器, 用来控制三个音调、噪声混音器及两个一般功能 I/O 端口工作, 音调噪声启用位都是低电位工作。

例如: $\text{B0}=0$, $\text{B1}=1$, $\text{B2}=1$, 则仅 A 声道的音调发生器被启用, B, C 声道被禁用。而音调与噪声二者可以同时启用, 以实现混音的特殊效果。此外对于 I/O 端口 A 的控制是由位 6 来决定其工作方向, I/O 端口 B 由位 7 来决定, “0”表示输入, “1”则表示输出。

12.1.4 振幅控制寄存器 R8, R9, R10

PSG 三个声道音量 (即信号振幅) 大小的输出是由个别的寄存器 R8, R9, R10 来做 5 个位的控制, 振幅的控制可以分为两种, 即固定振幅及可变振幅模式, 利用第 4 个位 “M” 来做控制。当 $\text{M}=0$ 时, 振幅是由 4 个位 ($\text{L0} \sim \text{L3}$) 的固定振幅值所定义。此值可以由程序来控制做音量的变化, 共有 16 种不同输出位。而当 $\text{M}=1$ 时声道的输出是由包络发生器的 4 个位所定义。

所以位 4 “M” 为振幅模式控制位, 可以视为 “包络启用” 位, 当 $\text{M}=0$ 时没有使用包络发生器, 当 $\text{M}=1$ 时则包络发生器有效。

12.1.5 包络发生器控制寄存器 R11、R12、R13

PSG 提供两种独立的方式来对包络进行控制, 第一是使用寄存器 R11 及 R12 去改变包络的频率, 第二是使用寄存器 R13 去改变包络相关的轮廓外型及循环周期。

寄存器 R11 及 R12 组成一个 16 位的计数器来控制包络周期, 首先将工作频率除以 256, 再由 16 位可编程化的寄存器 R11 及 R12 的内容去除频率。R11, R12 分别表示包络细音调及粗音调寄存器, 产生包络的频率计算公式为:

$f_E = f_{CLOCK} / [256 \times EP]$ $EP = 256 \times CT + FT$

- 式中：fE ：希望产生的包络频率；
fCLOCK ：输入时钟频率；
EP ：包络周期的 16 位值；
CT ：粗音调寄存器 R12 的内含值；
FT ：细音调寄存器 R11 的内含值。

而包络的外型及循环变化是由寄存器 R13 的低 4 位来设定，有不同的变化形态，至于振幅大小变化则由 4 位计数 E3、E2、E1、E0 来做变化。

4 位定义如下：

- Hold: 保持 当此位设定为 1 时，包络只有一个循环，并保持最后的包络计数 (E3~E0=0000 或 1111)，若是在下限则为计数 0000，若在上限计数为 1111。
- Alternate: 交替 若此位设定时，包络计数器在每次循环结束时执行反向计数，当保持及交替此二位都为 1 时，包络计数器会在一个计数周期之后，保持住原先开始计数的值。
- Attack: 上升 当此位为 1 时，包络计数器将从 E3~E0 为 0000 向上增加到 1111，当此位为 0 时则会下降，由 1111 到 0000。
- Continue: 延续 当此位为 1 时，循环形态将由保持的位定义。当此位为 0 时，包络计数将在一个循环之后重新设定 E3~E0 为 0000，且保留该值。

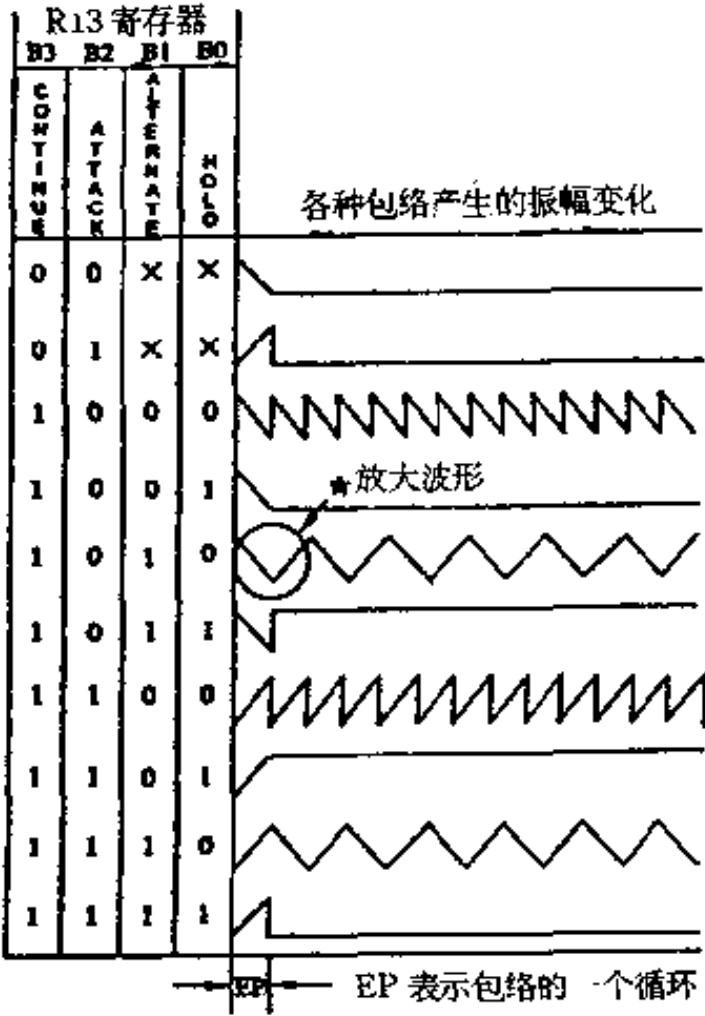


图 12-2 各种包络产生器波形变化

图 12-2 是各种包络产生的变化示意图, 其中标有 “*” 之处的波形放大, 由图 12-3 来表示, 代表包络振幅 E3~E0 的变化情形。

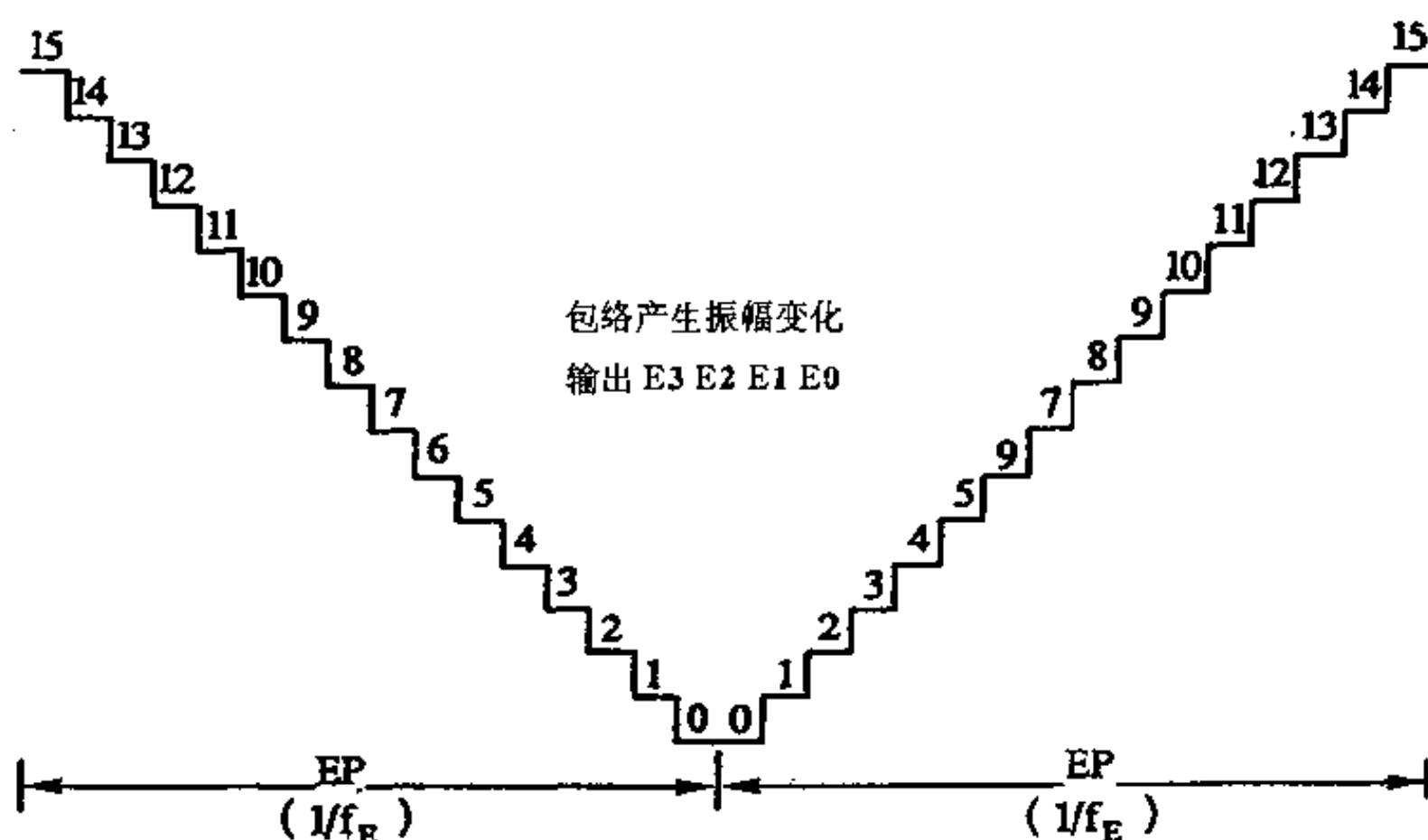


图 12-3 波形放大图

12.1.6 输入输出端口寄存器 R14、R15

R14、R15 寄存器分别做端口 A、端口 B 数据输入或输出的存储。设定为输入或输出是由控制寄存器 R7 的位 6 及位 7 来决定, “0” 表示输入, “1” 表示输出。

12.2 声效控制原理

PSG 的主要功能就是产生各种有趣的声效, 像机关枪声、子弹爆炸声、鸟鸣声, 这些声音在一些游戏机中, 大家应该不陌生, 本节将详细说明上述声效的制作原理, 及如何利用 PSG 来设计这些声效, 在以后几节中再配合单片机 8051 来完成这些声效的制作。

利用 PSG 来产生各种声效, 基本上可以分为以下几种方式:

1. 单纯音调效果;
2. 噪声配合包络控制效果;
3. 频率扫描效果。

综合起来, 我们可以看出是使用 PSG 来产生各种频率的声音, 再配合延迟时间或是利用噪声及包络产生不同声音波形来制作特殊的效果, 而控制的方式是将相关的数据写入对应的寄存器中, 其中假设送入 PSG 的工作时钟为 1.84MHz, 这是 8051 单板单片机

上产生的时钟。

12.2.1 单纯音调效果

声效 1: 救护车声

原理: 依序产生两种不同频率的声音, 而彼此间有一段延迟时间。如果持续地循环下去, 则成为救护车的警报声。

12.2.2 噪声配合包络控制效果

声效 2: 机关枪声

原理: 利用 PSG 噪声发生器产生随机数波形及可变振幅控制 (即包络发生器) 制造衰减的包络外型所产生出来的效果。

声效 3: 炸弹爆炸声

原理: 原理同上, 只是噪声的周期及包络粗调频率的值不同。其频率较低。启动所有三个声道的目的可以得到较具震撼效果的爆炸声响。

12.2.3 频率扫描效果

声效 4: 激光枪效果

原理: 利用扫描音调寄存器的内含值使频率递增或递减来达到星际大战游戏中类似的激光枪音响效果, 若改变扫描时间, (即每一扫描阶段的延迟时间) 可以得到不同的声声效果。

声效 5: 炸弹呼啸声

原理: 同上, 只是减慢音调寄存器内扫描频率的扫描时间, 最后再串接上述的炸弹爆炸的声音效果。

12.3 可编程声效发生器声音频率计算

看过 PSG 产生各种声效的原理及控制方式后, 在本节中先介绍一个以 TURBO C 程序编写的 PSG 声音频率计算公用程序, 方便我们来计算各种音阶产生所对应的 PSG 粗调及细调寄存器内含和及噪声产生频率。

在前面曾经介绍过 PSG 的各个内部寄存器功能及各种频率产生的计算公式, 根据这些公式, 套入 PSG 的工作频率, 便能完成各种频率产生的数值计算。频率计算程序文件名为 CAX.EXE, 执行结果如下:

```
==== PSG AY-3-8910 parameter calculator ====  
XTAL = 1.8432 M  
1 --> tone list  
2 --> tone cal.  
3 --> noise freq list
```

4 --> envelope tone cal.

ESC --> exit

共有 4 种计算项目:

1. 计算各个音符频率相对的 PSG 粗/细调寄存器值。

下表列出各个音符对应频率的值:

简谱	1	2	3	4	5	6	7	1	2	3	4	5	6	7
音符	C5	D5	E5	F5	G5	A5	B5	C6	D6	E6	F6	G6	A6	B6
频率	523	587	659	698	784	880	987	1046	1174	1318	1396	1567	1760	1975

根据此表, 并利用公式可以求得各个音阶对应到 PSG 音调控制寄存器 R0~R5 的内容, 其中 R0、R2、R4 表示音调细调寄存器 (8 个位), 而 R1、R3、R5 则表示粗调寄存器 (4 个位)。计算结果如下所示:

f= 523	fine tune=220	coarse tune=	0
f= 587	fine tune=196	coarse tune=	0
f= 659	fine tune=174	coarse tune=	0
f= 698	fine tune=165	coarse tune=	0
f= 784	fine tune=146	coarse tune=	0
f= 880	fine tune=130	coarse tune=	0
f= 987	fine tune=116	coarse tune=	0
f=1046	fine tune=110	coarse tune=	0
f=1174	fine tune= 98	coarse tune=	0
f=1318	fine tune= 87	coarse tune=	0
f=1396	fine tune= 82	coarse tune=	0
f=1567	fine tune= 73	coarse tune=	0
f=1760	fine tune= 65	coarse tune=	0
f=1975	fine tune= 58	coarse tune=	0

any key to continue ...

2. 计算某一频率产生相对 PSG 粗/细调寄存器值:

输入某一频率值, 而计算出相对于 PSG 粗调寄存器和细调寄存器的内含值。

3. 计算噪声产生寄存器 R6 能制造出的噪音频率值。

噪声产生寄存器是以 5 个位来表示, 噪声的频率范围可以从 49.7KHz 到 3.7KHz。以下是其计算结果。

PSG noise freq list for REG6 5 bits

R6= 1 freq= 49664 R6= 2 freq= 57600

R6= 3	freq= 38400	R6= 4	freq= 28800
R6= 5	freq= 23040	R6= 6	freq= 19200
R6= 7	freq= 16457	R6= 8	freq= 14400
R6= 9	freq= 12800	R6= 10	freq= 11520
R6= 11	freq= 10472	R6= 12	freq= 9600
R6= 13	freq= 8861	R6= 14	freq= 8228
R6= 15	freq= 7680	R6= 16	freq= 7200
R6= 17	freq= 6776	R6= 18	freq= 6400
R6= 19	freq= 6063	R6= 20	freq= 5760
R6= 21	freq= 5485	R6= 22	freq= 5236
R6= 23	freq= 5008	R6= 24	freq= 4800
R6= 25	freq= 4608	R6= 26	freq= 4430
R6= 27	freq= 4266	R6= 28	freq= 4114
R6= 29	freq= 3972	R6= 30	freq= 3840
R6= 31	freq= 3716		

any key to continue ...

4. 计算某一包络产生频率值相对 PSG 寄存器 R11, R12 分别表示包络细音调及粗音调寄存器, 只要输入欲产生的包络频率, 便可求出寄存器内含值。

程序清单

```

1  /*  CAX.C */
2  /*  cal, PSG 8910 various freq. */
3  #define TNO 14      /* 设定音阶表总数, 共 2 个 8 度音, 14 个音阶*/
4  /*  各个音阶频率值 */
5  int tonep[]={ 523, 587, 659, 698, 784, 880, 987,
6      1046, 1174, 1318, 1396, 1567, 1760, 1975};
7  int ft[TNO], ct[TNO]; /*存储细调寄存器及粗调寄存器值*/
8
9  /*-----*/
10 menu()      /* 主画面的功能选单 */
11 {
12     clrscr();
13     puts("=== PSG AY-3-8910   parameter calculator ===");
14     puts(" XTAL =  1.8432 M\n");
15     puts(" 1  --> tone list ");
16     puts(" 2  --> tone cal. ");

```



```
17  puts(" 3  --> noise freq list");
18  puts(" 4  --> envelope tone cal. ");
19  puts(" ESC --> exit ");
20  }
21  /*-----*/
22  main()
23  {
24
25  char c;
26
27  while(1)
28  {
29  menu();      /* 显示主画面的功能菜单 */
30  c=getch();
31  if(c==27) break;    /* 当按下"ESC" 键时，结束程序执行*/
32  switch (c)
33  {
34  case '1' :  list_tone(); break;
35  case '2' :  cal_tone(); break;
36  case '3' :  list_noise_freq(); break;
37  case '4' :  cal_anv_tone(); break;
38  default  :  break;
39  }
40  }
41  }
42  /*-----*/
43  list_tone() /* 计算音阶相对的粗细调寄存器内含值 */
44  {
45
46  int x,i,f;
47
48  clrscr();
49  for(i=0; i<TNO; i++)
50  {
51  f=tonep[i];
52  x=1843200/(16*f);
```

```

53  ct[i]=x/256; ft[i]=x%256;
54  printf("f=%4d   fine tune=%3d   coarse tune=%3d\n",
55  tonep[i], ft[i], ct[i]);
56  }
57  puts("any key to continue  ...");
58  getch();
59  }
60  /*-----*/
61  cal_tone()
62  {
63  /*   针对某一输入的音阶频率而计算相对的粗细调寄存器内含值 */
64  int ct, ft, f;
65  unsigned x;
66
67  clrscr();
68  printf("PSG tone cal.  please input desire freq ? ");
69  scanf("%d", &f); puts("");
70
71  x=1843200/(16*f);
72  ct=x/256; ft=x%256;
73  printf("f=%4d   fine tune=%3d   coarse tune=%3d\n", f, ft, ct);
74  puts("any key to continue  ...");
75  getch();
76  }
77  /*-----*/
78  list_noise_freq()    /* 计算噪音频率产生的范围 */
79  {
80  int i;
81  unsigned f;
82
83  clrscr();
84  puts("PSG   noise freq list   for REG6   5 bits ");
85
86  for(i=1; i<32; i++)
87  {
88  f=1843200/(16*i);

```

```

89  printf("R6= %2d   freq= %u   ", i, f);
90  if(i>=13) printf(" ");
91  if(i%2==0) printf("\n");
92  }
93  puts("\nany key to continue ...");
94  getch();
95  }
96  /*-----*/
97  cal_anv_tone()
98  { /*计算相对包络发生器频率的粗细调寄存器的内含值*/
99  int ct, ft, f;
100 unsigned x;
101
102  clrscr();
103  printf("PSG envelope tone cal.  please input desire freq ? ");
104  scanf("%d", &f);  puts("");
105
106  x=1843200/(256*f);
107  ct=x/256; ft=x%256;
108  printf("f=%4d   fine tune=%3d   coarse tune=%3d\n", f, ft, ct);
109  puts("any key to continue ...");
110  getch();
111  }
112  /*-----*/

```

12.4 产生救护车警报声

在前面我们说过利用 PSG 产生各种声效的原理及控制方式，从本节起将实际以 8051 单板 P51 及 C 语言编写程序来做相关声效的实验，本节先介绍产生救护车警报声的实验。

警报声的控制可以依序产生两种不同频率(440Hz 及 180Hz)的声音，而彼此间有一段延迟时间。如果持续地循环下去，则成为救护车的警报声。

控制方法：

使用寄存器	载入值	说明
R0	5	利用声道 A 的音调发生器
R1	1	制造 440Hz 的声音

(续表)

使用寄存器	载入值	说明
R7	62	启动声道 A 的音调发生器
R8	15	声道 A 的音量设为最大 (延迟一般时间)
R0	104	产生 180Hz 的声音
R1	2	 (延迟一般时间)
R8	0	关闭声道 A 的音量

执行结果

程序执行后工作指示 LED 灯闪动，产生救护车警报声效 10 声。

程序清单

```
1  /*   AM.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
6
7  char *title="test P51 PCB PSG ambulance sound";
8
9  delay(int t) /* 延迟子程序 */
10 {
11  int i,j;
12  for(i=0; i<t; i++)
13  for(j=0; j<5; j++);
14 }
15 /*-----*/
16 led_bl() /* 工作 LED 闪动 */
17 {
18  /*  blink RED led P1.7  */
19  char i;
20
21  for(i=0; i<4; i++)
22  {
23  cplbit(P1.7);
```

```

24     delay(40);
25 }
26 }
27 /*-----*/
28 cpsg(char r, char d)    /* PSG I/O 控制 */
29 { /* 写入寄存器 */
30     psg_reg=r; psg_data=d; /* 写入数据到寄存器中 */
31 }
32 /*-----*/
33 ambl() /* 产生救护车警报 */
34 {
35     cpsg(0,5); cpsg(1,1);
36     cpsg(7,62); cpsg(8,15);
37     delay(300);
38     cpsg(0,104); cpsg(1,2);
39     delay(300);
40     cpsg(8,0);
41 }
42 /*-----*/
43 main()
44 {
45     char i;
46     cpsg(7, 0x3F); /* 设定 PSG 的工作状态 */
47     led_bl(); /* 工作 LED 闪动 */
48
49     ambl(); /* 产生声效 */
50     delay(1000);
51     for(i=0; i<10; i++) /* 产生声效 10 声 */
52         ambl();
53     while(1){} /* 无穷循环 */
54 }

```

12.5 产生机关枪声响

机关枪声音是利用 PSG 噪声发生器产生的，产生随机数波形及可变振幅控制（即包络发生器），制造衰减的包络外型所产生出来的效果。

控制方法:

使用寄存器	载入值	说 明
R6	15	设定噪音频率到中间值
R7	7	启动 3 个声道的噪声发生器
R8	16	启动声道 A 的包络发生器
R9	16	启动声道 B 的包络发生器
R10	16	启动声道 C 的包络发生器
R11	16	设定包络频率为 2Hz
R12	8	
R13	0	设定包络为“衰减”形式，只有产生一个周期的声音

执行结果

程序执行后工作指示 LED 灯闪动，产生机关枪声音效果 10 声。

程序清单

```
1 /* GUN.C */
2 #include "8051io.h" /* 载入 MC51 头文件 */
3 #include "8051reg.h"
4 #include "8051bit.h"
5 #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
6
7 char *title="test P51_PCB PSG gun sound";
8
9 delay(int t) /* 延迟子程序 */
10 {
11     int i,j;
12     for(i=0; i<t; i++)
13         for(j=0; j<5; j++);
14 }
15 /*-----*/
16 led_bl() /* 工作 LED 闪动 */
17 {
18     /* blink RED led P1.7 */
19     char i;
20
21     for(i=0; i<4; i++)
```

```
22  {
23      cplbit(P1.7);
24      delay(40);
25  }
26  }
27  /*-----*/
28  cpsg(char r, char d)    /* PSG I/O 控制 */
29  { /* 写入寄存器 */
30      psg_reg=r; psg_data=d; /* 写入数据到寄存器中 */
31  }
32  /*-----*/
33  gun() /* 产生机关枪声音效果 */
34  {
35      cpsg(7,7);
36      cpsg(6,15);
37      cpsg(8,16); cpsg(9,16); cpsg(10,16);
38      cpsg(11,16); cpsg(12,14);
39      cpsg(13,0);
40  }
41  /*-----*/
42  main()
43  {
44      char i;
45
46      cpsg(7, 0x3F); /* 设定 PSG 的工作状态 */
47      led_bl(); /* 工作 LED 闪动 */
48      gun(); /* 产生声音效果 */
49      delay(2000);
50      /* 产生声音效果 10 声 */
51      for(i=0; i<10; i++)
52      {
53          gun();
54          delay(300);
55      }
56      while(1){} /* 无穷循环 */
57  }
```

12.6 产生爆炸声响

原理与上一节介绍产生机关枪声响的方法相同，只是噪声的周期及包络粗调频率的值不同。其频率较低。启动所有三个声道的目的可以得到较具震撼效果的爆炸声响。

控制方法：

使用寄存器	载入值	说 明
R6	0	设定噪音频率为最低
R7	7	启动 3 个声道的噪声发生器
R8	16	启动声道 A 的包络发生器
R9	16	启动声道 B 的包络发生器
R10	16	启动声道 C 的包络发生器
R11	32	设定包络频率为 0.5Hz
R12	20	
R13	0	设定包络为“衰减”形式只有产生一个周期的声效

执行结果

程序执行后工作指示 LED 灯闪动，产生爆炸声 10 声的声音效果。

程序清单

```

1  /*   EXP.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
6
7  char *title="test P51_PCB PSG explosion sound";
8
9  delay(int t) /* 延迟子程序 */
10 {
11     int i,j;
12     for(i=0; i<t; i++)
13         for(j=0; j<5; j++);
14 }
15 /*-----*/
16 led_b1() /* 工作 LED 闪动 */
17 {

```



```
18  /* blink RED led P1.7  */
19  char i;
20
21  for(i=0; i<4; i++)
22  {
23      cplbit(P1.7);
24      delay(40);
25  }
26 }
27 /*-----*/
28 cpsg(char r, char d)      /* PSG I/O 控制 */
29 { /* 写入寄存器 */
30     psg_reg=r; psg_data=d; /* 写入数据到寄存器中 */
31 }
32 /*-----*/
33 exp()      /* 产生 10 声爆炸声音效果 */
34 {
35     cpsg(6,0);
36     cpsg(7,7);
37     cpsg(8,16); cpsg(9,16); cpsg(10,16);
38     cpsg(11,32); cpsg(12,28);
39     cpsg(13,0);
40 }
41 /*-----*/
42 main()
43 {
44     char i;
45     cpsg(7, 0x3F);      /* 设定 PSG 的工作状态 */
46     led_bl(); /* 工作 LED 闪动 */
47     exp(); /* 产生声音 */
48     delay(2000);
49     /* 产生声效 10 声 */
50     for(i=0; i<10; i++)
51     {
52         exp();
53         delay(1300);
```

```
54 }
55 while(1){
56 }
```

12.7 产生激光枪声响

激光枪声响效果的产生可利用扫描音调寄存器的内含值，使频率递增或递减来实现星际大战游戏中类似的激光枪音响效果，若改变扫描时间(即每一扫描阶段的延迟时间)可以得到不同的声音效果。

控制方法：

使用寄存器	载入值	说 明
R7	62	启动声道 A 的音调发生器
R8	15	声道 A 的音量设为最大
R1	0	由声道 A 的音调发生器做频率递减扫描，起始频
R0	48	率为 2000Hz，结束频率为 1000Hz，每一阶段延
R0	f	迟时间约 5ms
R0	112	
R8	0	关闭声道 A 的音量

执行结果

程序执行后工作指示 LED 灯闪动，产生 10 声激光枪声音效果。

程序清单

```
1 /* LASER.C */
2 #include "8051io.h" /* 载入 MC51 头文件 */
3 #include "8051reg.h"
4 #include "8051bit.h"
5 #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
6
7 char *title="test P51_PCB PSG laser sound";
8
9 delay(int t) /* 延迟子程序 */
10 {
11 int i,j;
12 for(i=0; i<t; i++)
13 for(j=0; j<5; j++);
```

```
14 }
15 /*-----*/
16 led_bl() /* 工作 LED 闪动 */
17 {
18     int i;
19     for(i=0; i<4; i++)
20     {
21         cplbit(P1.7);
22         delay(40);
23     }
24 }
25 /*-----*/
26 cpsg(char r, char d) /* PSG I/O 控制 */
27 { /* 写入寄存器 */
28     psg_reg=r; psg_data=d; /* 写入数据到寄存器中 */
29 }
30 /*-----*/
31 laser() /*产生激光枪声音效果*/
32 { int i;
33
34     cpsg(7,62); cpsg(8,15);
35     cpsg(1,0);
36     for(i=57; i<115; i++) /*频率扫描 */
37     { cpsg(0,i); delay(10); }
38     cpsg(8, 0);
39 }
40 /*-----*/
41 main()
42 {
43     char i;
44     cpsg(7, 0x3F); /* 设定 PSG 的工作状态 */
45     led_bl(); /* 工作 LED 闪动 */
46     laser(); /* 产生声音 */
47     delay(2000);
48     /* 产生 10 声声音 */
49     for(i=0; i<10; i++)
```

```

50 {
51     laser();
52     delay(300);
53 }
54 while(1){ /* 无穷循环 */
55 }

```

12.8 产生炸弹呼啸声效

炸弹呼啸声产生原理类似上一节产生激光枪声响，只是减慢音调寄存器内扫描频率和扫描时间，最后再串接上述的炸弹爆炸声音效果。

控制方法：

使用寄存器	载入值	说 明
R7	62	启动声道 A 的音调发生器
R8	15	声道 A 的音量设为最大
R1	0	由声道 A 的发生器做频率
R0	48	递减扫描起始频率为 2000Hz，结束频率为 1000Hz，每一阶段的
	└	延迟时间约 10ms
R0	255	

串接爆炸声响的控制方法

执行结果

程序执行后工作指示 LED 灯闪动，产生 10 声炸弹呼啸的声音效果。

程序清单

```

1  /* BOMB.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
6
7  char *title="test P51_PCB PSG bomb sound";
8
9  delay(int t) /* 延迟子程序 */
10 {
11     int i,j;

```

```
12   for(i=0; i<t; i++)
13   for(j=0; j<10; j++) ;
14 }
15 /*-----*/
16 led_bl() /* 工作 LED 闪动*/
17 {
18   int i;
19
20   for(i=0; i<4; i++)
21   {
22     cplbit(P1.7);
23     delay(50);
24   }
25 }
26 /*-----*/
27 cpsg(char r, char d) /* PSG I/O 控制 */
28 {
29   psg_reg=r; psg_data=d;
30 }
31 /*-----*/
32 exp() /* 产生炸弹爆炸声音*/
33 {
34   cpsg(6,0);
35   cpsg(7,7);
36   cpsg(8,16); cpsg(9,16); cpsg(10,16);
37   cpsg(11,32); cpsg(12,28);
38   cpsg(13,0);
39 }
40 /*-----*/
41 bomb() /* 产生炸弹呼啸声音 */
42 { int i;
43   cpsg(7,62); cpsg(8,15);
44   cpsg(1,0);
45   for(i=57; i<255; i++)
46   { cpsg(0, i); delay(10); }
47   exp();
```

```
48 }
49 /*-----*/
50 main()
51 {
52     char i;
53
54     cpsg(7, 0x3F); /* 设定 PSG 的工作状态 */
55     led_bl();      /* 工作 LED 闪动 */
56     bomb();        /* 产生声效 */
57
58     delay(2000);
59     /* 产生声效 10 声 */
60     for(i=0; i<10; i++)
61     {
62         bomb();
63         delay(300);
64     }
65     while(1){} /* 无穷循环 */
66 }
```

12.9 测试各个单音音阶

在 12.3 节中曾经介绍过声效控制声音频率计算程序 CAX.EXE，由功能选项 1，可以计算出各个音阶频率对应的 PSG 粗细调寄存器内含值。本节用程序来测试 PSG 所产生的各个单音音阶。

执行结果

程序执行后工作指示 LED 灯闪动，接着产生各个单音音阶声音。

程序清单

```
1 /* TONE.C */
2 #include "8051io.h" /* 载入 MC51 头文件 */
3 #include "8051reg.h"
4 #include "8051bit.h"
5 #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
6
7 char *title="test P51_PCB PSG tone";
```

```
8
9 #define PDELAY 100    /* 一拍音长延迟 */
10 #define TNO 14        /* 音阶频率总数 */
11 /* 各个音阶频率对应的细调寄存器内含值 */
12 unsigned char ft[]={220, 196, 174, 165, 146, 130, 116,
13    110, 98, 87, 82, 73, 65, 58};
14 char beat=1;
15
16 delay(int t) /* 延迟子程序 */
17 {
18     int i,j;
19     for(i=0; i<t; i++)
20         for(j=0; j<5; j++);
21 }
22 /*-----*/
23 led_bl() /* 工作 LED 闪动 */
24 {
25     /* blink RED led P1.7 */
26     char i;
27
28     for(i=0; i<4; i++)
29     {
30         cplbit(P1.7);
31         delay(40);
32     }
33 }
34 /*-----*/
35 cpsg(char r, char d) /* PSG I/O 控制 */
36 {
37     psg_reg=r; psg_data=d;
38 }
39 /*-----*/
40 play(int f, int len) /* 发出一个单音 */
41 {
42     cpsg(0, ft[f]);
43     cpsg(1, 0);
```

```

44   delay(len*PDELAY);
45 }
46 /*-----*/
47 test_tone() /* 发出各个音阶测试音 */
48 {
49   int i;
50
51   cpsg(7, 62);
52   cpsg(8, 15);
53   for(i=0; i<TNO; i++)
54     { play(i, beat); led_bl();}
55
56   delay(500);
57   for(i=TNO-1; i>=0; i--)
58     { play(i, beat); led_bl();}
59
60   cpsg(8, 0);
61 }
62 /*-----*/
63 main()
64 {
65   cpsg(7, 0x3f);      /* 设定 PSG 的工作状态 */
66   led_bl();           /* 工作 LED 闪动 */
67   test_tone();        /* 发出各个音阶测试音 */
68   while(1) {}        /* 无穷循环*/
69 }

```

12.10 演奏一段旋律

上一节已经讲过如何控制 PSG 测试各个单音音阶，将各个单音连接在一起便可以组成一个曲子或是演奏一段旋律，为了方便音阶设计，曲子音阶表示直接以数字表示，例如：

曲子音阶指针数据数组如下：

SONG: DB 3,5,5,3,2,1,2

表示: "ME"、"SO"、"SO"、"ME"、"RE"、"DO"、"RE"

执行结果

程序执行后工作指示 LED 灯闪动, 开始演奏一段旋律。

程序清单

```

1  /*  SONG.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "c: p51.h" /* 载入 P51 I/O 控制头文件 */
6
7  char *title="test P51_PCB PSG play a song";
8  #define PDELAY 100 /* 一拍音长延迟*/
9  #define TNO 14 /* 音阶频率总数*/
10 /* 各个音阶频率对应的细调寄存器内含值 */
11 unsigned char ft[]={220, 196, 174, 165, 146, 130, 116,
12 110, 98, 87, 82, 73, 65, 58};
13 char beat=1;
14 /* 歌曲音阶频率值 */
15 char song1[]="355321 23532 355321 23211 6qq756 6q756\
16 355321 23532 355321 23211 6qq756 6q756";
17 /* 歌曲音长值 */
18 char leng1[]="122111 11112 111112 13111 121112 12131\
19 111213 13112 123112 13321 122212 11123";
20 delay(int t) /* 延迟子程序 */
21 {
22 int i,j;
23 for(i=0; i<t; i++)
24 for(j=0; j<10; j++);
25 }
26 /*-----*/
27 led_bl() /* 工作 LED 闪动 */
28 {
29 int i;
30
31 for(i=0; i<4; i++)
32 {
33 cplbit(P1.7);
34 delay(50);
35 }
36 }
37 /*-----*/

```

```

38  cpsg(char r, char d)      /* PSG I/O 控制 */
39  {
40      psg_reg=r;
41      psg_data=d;
42  }
43  /*-----*/
44  play(int f, int len)      /* 发出一个单音 */
45  {
46      cpsg(0, f);
47      cpsg(1, 0);
48      delay(len*PDELAY);
49  }
50  /*-----*/
51  playc(char ch, int len)   /* 依据字符发出一个单音 */
52  {
53      switch(ch)
54      {
55          case '1': play(0, len); break;
56          case '2': play(1, len); break;
57          case '3': play(2, len); break;
58          case '4': play(3, len); break;
59          case '5': play(4, len); break;
60          case '6': play(5, len); break;
61          case '7': play(6, len); break;
62          case 'q': play(7, len); break;
63          case 'w': play(8, len); break;
64          case 'e': play(9, len); break;
65          case 'r': play(10, len); break;
66          case 't': play(11, len); break;
67          case 'y': play(12, len); break;
68          case 'u': play(13, len); break;
69          case ' ': break; /* no sound */
70          default : break;
71      }
72  }
73  /*-----*/
74  /* 分析曲子字符串数据来演奏歌曲 */
75  play_song(char *tones, char *len)
76  {
77      int l;
78      cpsg(7, 62);
79      cpsg(8, 15); /* 音量设为最大 */

```

```
80  playc('3', 1); /* 发出单音"ME" */
81  playc('5', 1); /* 发出单音"SO" */
82  playc('5', 1); /* 发出单音"SO" */
83
84  do{
85      l=(*len++)-0x30; /* 计算音长 */
86      playc(*tones, l); /* 发出一个单音 */
87      delay(20);
88  } while(*tones++ !='\0');
89  cpsg(8,0); /* 音量设为最小 */
90  }
91  /*-----*/
92  main()
93  {
94      cpsg(7, 0x3f); /* 设定 PSG 的工作状态 */
95      led_bl(); /* 工作 LED 闪动 */
96      play_song(song1, leng1); /* 演奏一首曲子 */
97      while(1){} /* 无穷循环 */
98  }
99  /*-----*/
```

12.11 习 题

1. 说明可编程声效发生器(PSG)的功能。
2. 说明 PSG 启用控制寄存器 R7 的功能。
3. 试设计一程序产生其他 3 种不同的声效。

第 13 章 数字模拟转换器接口

数字模拟转换器简称 DAC(Digital-Analog Converter)，是将数字信号转换成连续的模拟信号的元件，一般用在数字接口或微处理机的接口输出控制上，典型的应用有以下几种：

1. 数字激光唱盘 CD 的放音转换；
2. 电脑 VGA 接口卡中的显像输出转换电路；
3. 直流马达速度控制；
4. 数字式电源供给器；
5. 任意波形发生器；
6. 电脑合成乐器控制；
7. FM 音源器的输出转换；
8. 电脑数字放音控制。

其中的电脑数字放音在许多新型的电子产品中我们都可以看到，几乎任何品需要语音提示的产品，都可派上用场，熟悉此控制接口技巧将可以在自行设计的产品中加入语音的功能，提升产品的附加价值。

在本章中主要介绍 DAC 的基本概念，及举实例做说明，同时以 8051 多功能控制板来做简单的 DAC 实验。

13.1 DAC 接口设计

图 13-1 是一般 DAC 接口的组成结构，由电脑送出的数字数据经过并行输出接口将数值信号锁存在 DAC 控制芯片上，再经过 DAC 做数字至模拟信号的转换，最后输出相对的模拟信号。在本章中以 8051 多功能控制板 P51 来做实验，由 8051 连接 8255 做 I/O 扩充，使用 8255 端口 A PA 控制 DAC 接口芯片，DAC 接口芯片则是采用市面上电子材料行容易买到的元件 DAC0800。

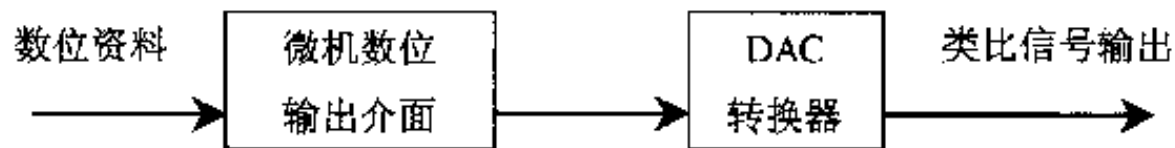


图 13-1 DAC 接口组成结构

在 4.9 节中我们已经说过 DAC0800 的工作原理，以 8255 端口 A 来控制 D/A 的线路，请参考图 4-14，DAC 的接口实验请参考以下几节说明。

13.2 测量 DAC 输出电压值

看过 DAC0800 的控制电路后, 本节用程序来做 DAC 转换的实验, 借以求证 DAC0800 如何输出模拟电压, 通过输出端测量实际 DAC 转换出来的直流电压值。

执行结果

先将万用表拨到直流电压挡, 黑线测试棒跨接于电源散热片的接地端, 红线则接于控制板的 J24 DA/OP 点上。执行程序后, 可以看到电表指针指示约为 -1.4V , 过一会儿变为 0V , 然后又变为 2V , 如此一直循环下去。注意电表测量值因观测的方式不同会存在测量误差, 如 -1.45V , -1.5V 。

程序清单

```
1  /* DAV.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "c:p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  #define CWORD 0x88 /* PA PB PC0~3 o/p PC4~7 i/p */
9  char *title="Test P51_PCB D/A o/p voltage";
10 /*-----*/
11 delay(int t) /* 延迟子程序 */
12 {
13     int i,j;
14     for(i=0; i<t; i++)
15         for(j=0; j<10; j++);
16 }
17 /*-----*/
18 led_bl() /* 工作 LED 闪动 */
19 {
20     /* blink RED led P1.7 */
21     char i;
22
23     for(i=0; i<4; i++)
24     {
25         cplbit(P1.7);
```

```
26     delay(40);
27 }
28 }
29 /*-----*/
30 main()
31 {
32     led_bl();          /* 工作 LED 闪动*/
33     cr=CWORD;          /* 设定 8255 工作模式 */
34     serinit(9600);      /* 初始化串行接口*/
35     printf(title);      /* 显示工作消息*/
36     while(1)           /* 无穷循环 */
37     { /* 送出测试的数值进行转换 */
38         printf("D/A port test ..... \n");
39         printf("o/p 0 (-1.4v) \n");
40         pa=0;          delay(2000);
41         led_bl();
42
43         printf("o/p 128 (0v)  \n");
44         pa=128; delay(2000);
45         led_bl();
46
47         printf("o/p 255 (2v)  \n");
48         pa=255; delay(2000);
49         led_bl();
50     }
51 }
52 /*-----*/
```

13.3 由 DAC 接口发出声音

单片机语音接口的输出控制经常需要使用到 DAC 接口, 本节先用简易的 DAC 接口输出声音, 在下一章将由 8051 利用此接口来输出语音, 第 15 章则介绍如何在 8051 上进行语音的录放音实验, 有兴趣的读者请自行翻阅。本节实验请将喇叭接于控制板的 J6 接头上, 才可以听见声音。

执行结果

执行程序后, 可以听见喇叭发出“嘀”声 10 次。

程序清单

```
1  /* DABE.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "c:p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  #define CWORD 0x88 /* PA PB PC0~3 o/p PC4~7 i/p */
9  char *title="Test P51_PCB D/A beeper.....";
10 /*-----*/
11 delay(int t) /* 延迟子程序 */
12 {
13     int i,j;
14     for(i=0; i<t; i++)
15         for(j=0; j<10; j++);
16 }
17 /*-----*/
18 led_bl() /* 工作 LED 闪动 */
19 {
20     /* blink RED led P1.7 */
21     char i;
22
23     for(i=0; i<4; i++)
24     {
25         cplbit(P1.7);
26         delay(40);
27     }
28 }
29 /*-----*/
30 beep() /* DAC 接口喇叭发出“嘀”声 */
31 {
32     char i;
33     for(i=0; i<30; i++)
34     {
35         pa=0; delay(1);
```

```
36    pa=255; delay(1);
37    }
38    pa=128; /* 0v */
39    }
40    /*-----*/
41    main() /* 主程序 */
42    {
43    char i;
44
45    led_bl(); /* 工作 LED 闪动 */
46    cr=CWORD; /* 设定 8255 工作模式 */
47    serinit(9600); /* 初始化串行接口 */
48    printf(title); /* 显示工作消息 */
49    /* DAC 接口喇叭发出“嘀”声 10 次 */
50    for(i=0; i<10; i++)
51    {
52    beep(); /* “嘀”一声 */
53    led_bl(); /* 工作 LED 闪动 */
54    }
55    while(1){} /* 无穷循环 */
56    }
57    /*-----*/
```

13.6 习 题

1. 列举 4 种 DAC 的应用例子。
2. 若要由 DAC 输出 1.0V 及 1.5V, 其输入数字数据应分别为何?
3. 将 DAC 输出端改接至示波器, 写一程序送出周期 1 ms 振幅为 2V 的方波信号。
4. 将 DAC 输出端改接至示波器, 写一程序送出周期 1 ms 振幅为 2V 的三角波信号。
5. 将 DAC 输出端改接至示波器, 写一程序送出周期 1 ms 振幅为 2V 的锯齿波信号。

第 14 章 利用 8051 输出语音

语音提示功能在电子业的使用越来越普及，像来客报知器会说“欢迎光临”、“谢谢光临”；电梯中会自动告知到了几楼；有些自动提款机会以语音提示告知如何操作；语音时钟会自动以语音告知时间；汽车超速检查器会以语音通知驾驶员要小心驾驶。生活中已经有很多具有语音功能的产品，更多的语音应用产品正在陆续开发中，毕竟以语音提示来引导操作最为方便，不用眼睛看便可了解。

由于语音产品技术的提高，目前国内有些半导体设计厂家已开发出专用的语音播放控制芯片，使得批量生产时的成本可以降低很多。语音播放的时间从一二秒到二三十秒的规格都有，也有的可以分段播放语音，视设计者的使用场合而定。

本章并不讨论专业的声音录放控制技术，而是仅探讨基本的录放音原理，并利用设计在 PC 上的现成语音卡录制一些语音，并做语音数据转换，好让 8051 单板直接推动 DAC 接口而将原先录到的声音还原而播放出来。

14.1 声音录音放音基本原理

声音信号是一种连续性的模拟信号，而用电脑来做录放音控制，得先将声音信号数字化读入计算机内存，这数字化的工作我们以图 14-1 来做说明。原来的连续性的语音波形，经过数字取样的处理后可以得到数字的声音数据，其中所要的硬件设备为 A/D（模拟/数字转换）接口，在软件处理上是每隔一小段时间拾取一点声音数字数据至语音内存内，这一小段时间称为取样周期。

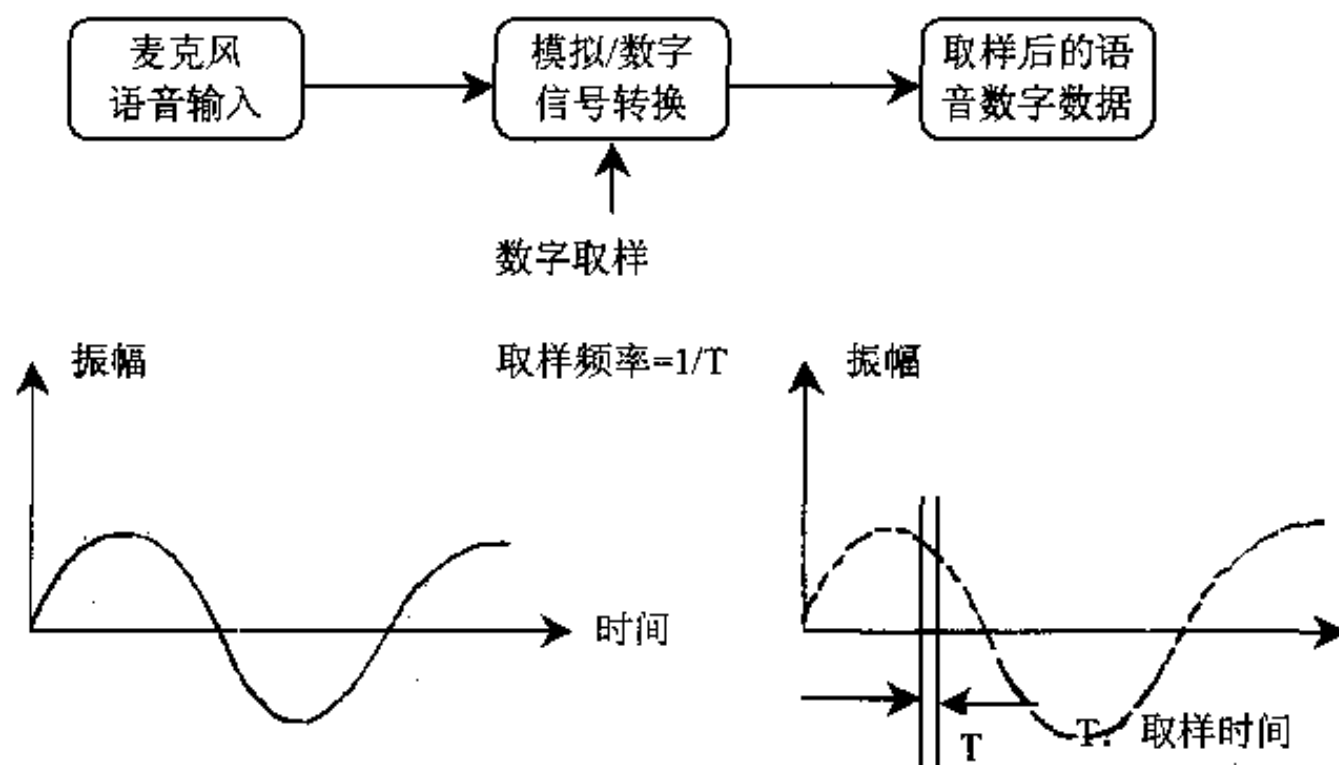


图 14-1 声音录音数位取样示意图

取样周期的倒数称为取样频率，即每秒钟对原始信号做多少次数字取样，由采样定理来看，取样频率不能小于原始信号最高频率的两倍，否则会造成取样失真，即所得到的数字数据不能将原来的模拟声音信号原本的还原。

取样频率设定对声音的还原（声音品质）有重要的影响，当取样频率越高时，所占内存会增加，相对的放音品质佳，一般音乐的频率响应范围为 20Hz~20KHz，因此像 CD 音响其数字取样频率则高达 44KHz，保证可以将声音信号原音重现出来。

一旦声音经数字取样存入电脑内存内，可加以分析、取参数做压缩存储或是做语音识别，如要将声音还原，只要根据原先的取样频率通过 D/A 接口处理，便可将声音原音重现出来，其过程如图 14-2。

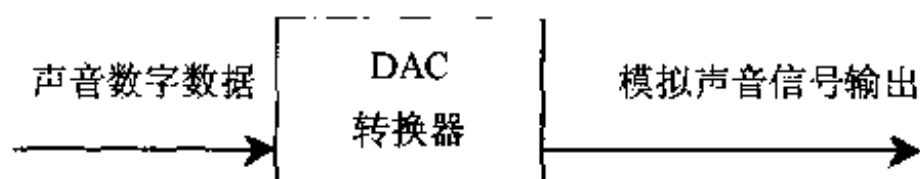


图 14-2 数位放音控制流程

14.2 产生及编辑语音波形文件

本节说明如何利用 PC 语音控制实验卡的语音编辑程序，来产生及编辑语音波形文件，以方便本章所做相关的实验。有关 PC 语音控制实验卡的使用可以参考下一章说明。本节只介绍如何录制及编辑语音波形文件。

我们是利用 PC 语音卡语音录放编辑程序 MP.EXE 来录制并编辑语音波形文件，以下说明语音提示“电脑”的语音数据文件如何产生。使用前请先安装鼠标，语音分析及识别综合系统的可执行文件为 MP.EXE。

输入 MP <ENTER>

出现如图 14-3 的主画面，详细的操作步骤如下：

- 步骤 1：移动鼠标至 MIC ICON 区，按下鼠标左键，说出“电脑”，再次按下鼠标左键，则系统完成录音的工作，并将波形显示出来，如图 14-4。
- 步骤 2：移动鼠标到语音波形的前端位置单击鼠标左键，再移动鼠标到波形的末端位置，再次单击鼠标左键，可以将整段语音剪切下来，系统将此范围内的语音重播，同时波形予以放大，执行结果如图 14-5 所示。此时可以按下“s”键，对所分离出来的语音数据进行保存，按下“t”键，再次播放声音。
- 步骤 3：按下“s”键，输入文件名 COMP.SP，保存为“电脑”的语音文件。
- 步骤 4：单击鼠标右键返回主画面，按“ESC”键，返回 DOS。

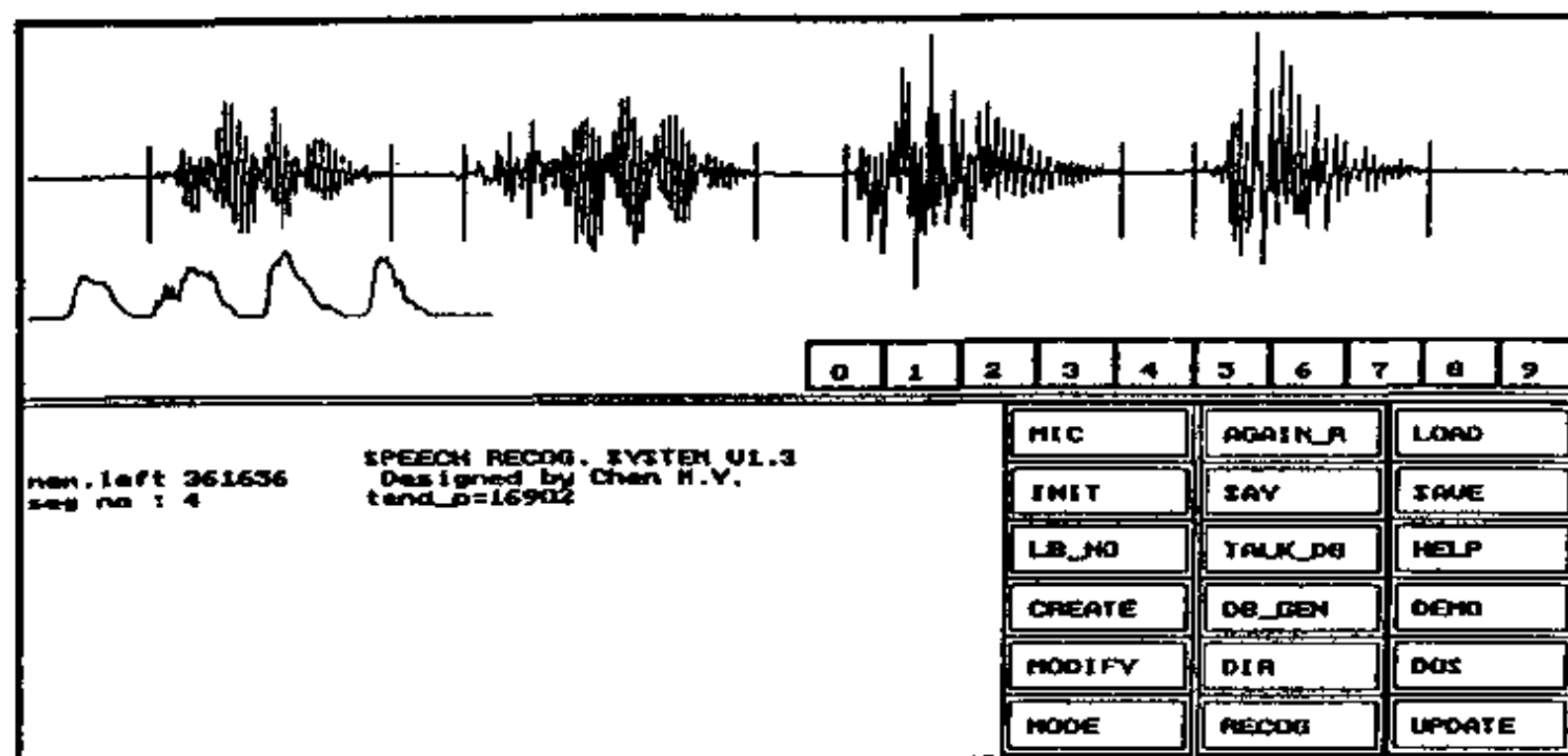


图 14-3 语音编辑程序执行操作画面

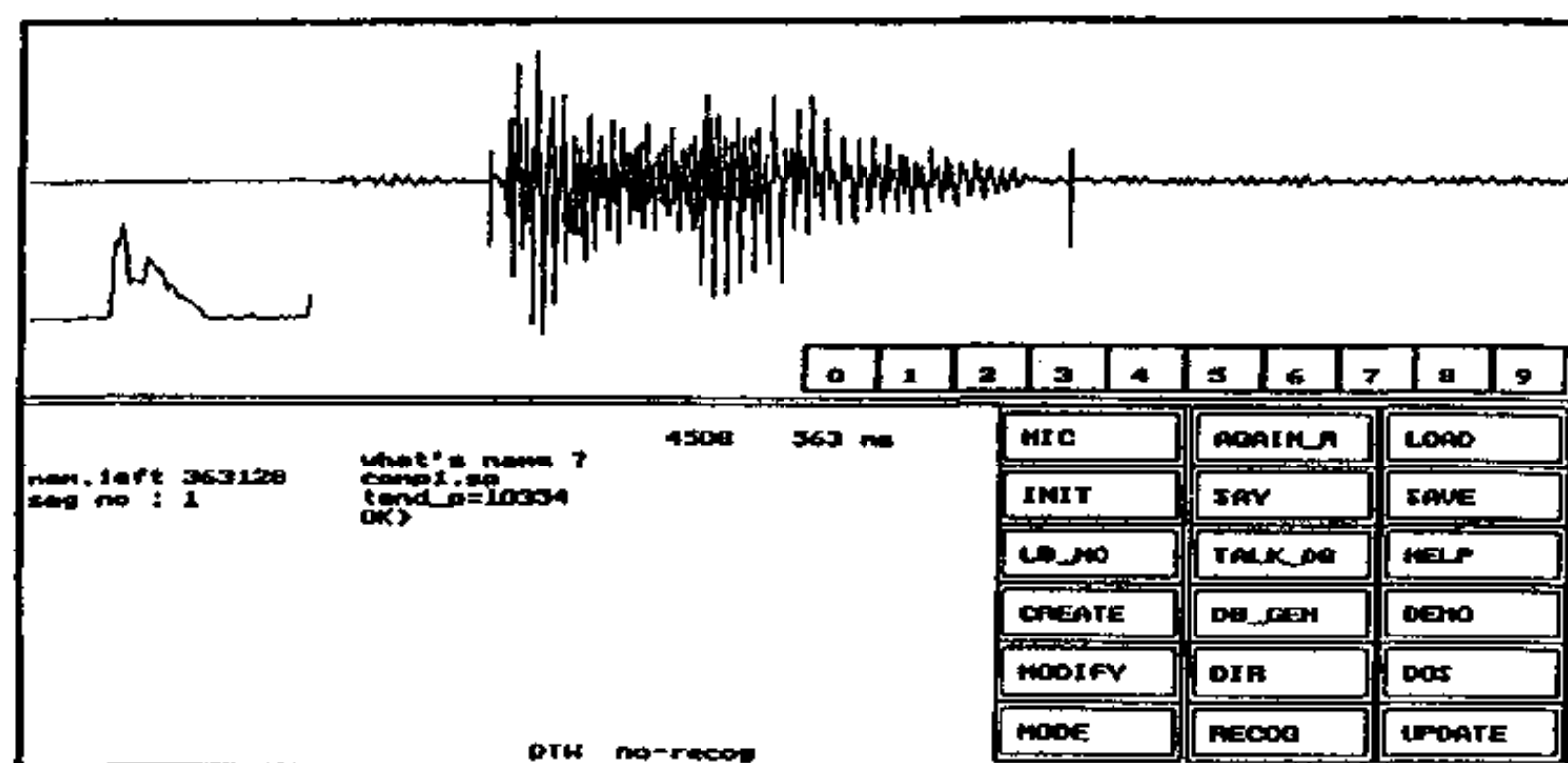


图 14-4 所录制的声音波形

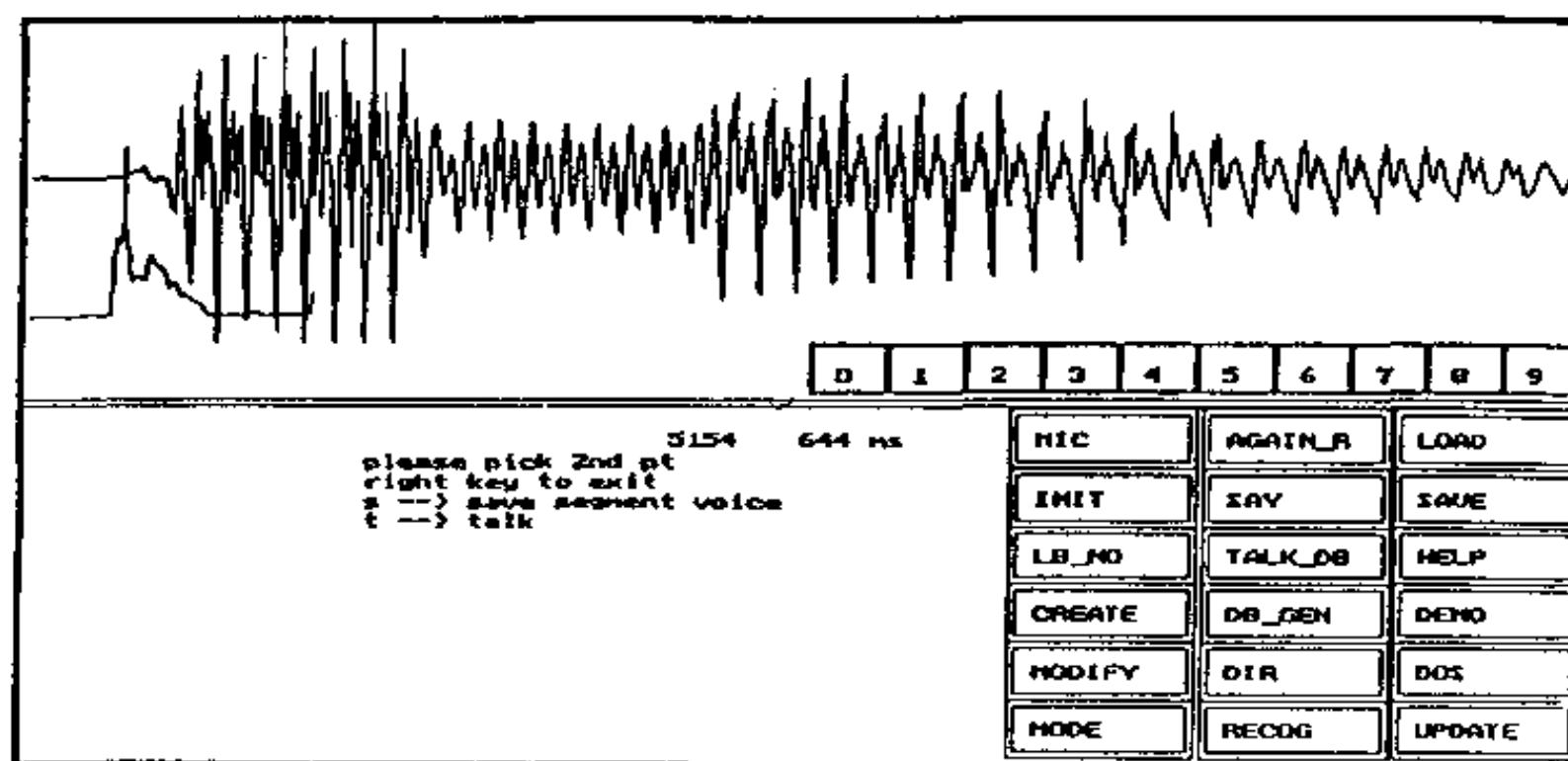


图 14-5 语音波形剪切及放大

步骤 5: 查看语音文件大小。

```
DIR  DB_VOICE\COMP <ENTER>
Volume in drive D has no label
Volume Serial Number is 321B01BDF
Directory of  C: \MP\DB_VOICE
COMP      SP      4096  11-21-94   8: 38p
      1 file(s)      4096 bytes
                        2584576 bytes free
```

只有 4096 字节。

步骤 6: 利用语音播放公用程序 SPP.EXE 来检查语音文件内容。

```
SPP  COMP.SP <ENTER>
```

可以听到电脑音箱放出“电脑”语音。至此我们已经制作好了一个语音文件，下次可以依需要再制作其他的语音文件，在下一节将介绍如何将语音数据文件加以转换来供 8051 单板使用。

14.3 转换语音数据文件

在上一节中已介绍过如何利用语音编辑程序来产生及编辑语音波形文件，我们最终的目的是希望能通过 8051 说出语音提示语，当然可以先将声音录到单板的内存内，再播放出来，最简单的方法是将语音数据放到只读的 ROM 中，在程序执行时直接从 ROM 中读取并送往 DAC 控制而播放出声音。

要把语音数据存入 ROM 中，用 C 语言做程序设计的话，可以将语音数据作为一般的数组数据并初始化，而语音数据文件的取得可以通过 PC 上的语音卡录制并进行编辑，语音数据文件的设计步骤如下：

1. 产生语音数据文件

用语音卡录制声音并用分析综合系统编辑好语音数据文件，之所以要编辑语音数据，一来可以将静音及杂音擦除以提高音质，二来可以减少语音数据的长度，在上节已讲过如何编辑一个语音文件，并产生了一个语音文件，文件名为 COMP.SP，其内容为“电脑”可以直接拿来使用。

2. 将语音数据文件转换为十六进制数组数据文件

语音数据文件.SP 数据的存储格式为一般的二进制文件，我们希望的语音数据是能包括在 C 语言内，如下所示的十六进位数组数据文件：

```
char buf[]={
0x83,0x85,0x84,0x84,0x84,0x84,0x81,0x81,0x80,0x80,
```

```

0x7f,0x7d,0x7e,0x7e,0x7e,0x7e,0x7f,0x7f,0x7f,0x80,
0x81,0x81,0x82,0x82,0x83,0x83,0x83,0x83,0x83,0x83,
.....
};

```

于是我们写一个转换语音数据文件为文字文件的程序，文件名为 VOCD.C。执行的方法为：

VOCD 语音数据文件 转换输出文件

例如：

VOCD COMP.SP COMP.AR

执行后可以产生 COMP.AR 的文字文件，如下所示：

```

0x83,0x85,0x84,0x84,0x84,0x84,0x81,0x81,0x80,0x80,
0x7f,0x7d,0x7e,0x7e,0x7e,0x7e,0x7f,0x7f,0x7f,0x80,
0x81,0x81,0x82,0x82,0x83,0x83,0x83,0x83,0x83,0x83,
.....
0x55,0xaa};

```

其中 “0x55,0xaa” 为文件结束代码。

3. 用文书处理器编辑产生语音数据头文件

在 COMP.AR 文件的前面加上 “ char buf[]={”，格式如下：

```

char buf[]={
0x83,0x85,0x84,0x84,0x84,0x84,0x81,0x81,0x80,0x80,
0x7f,0x7d,0x7e,0x7e,0x7e,0x7e,0x7f,0x7f,0x7f,0x80,
0x81,0x81,0x82,0x82,0x83,0x83,0x83,0x83,0x83,0x83,
.....
0x55,0xaa};

```

文件 COMP.AR 变成 C 语言可以编译的语音数据文件了。

程序清单

```

1 /* VOCD.C */
2 /* Read .SP file and convert to text HEX array file */
3 /* usage : VOCD *.sp *.ar */
4
5 #include <stdio.h>

```

```
6  #include <dos.h>
7
8  char *fname1, *fname2;
9  /*-----*/
10 main(argc,argv)
11 int argc;
12 char *argv[];
13 {
14     if(argc!=3)
15     {
16         clrscr(); /* 告知使用方法 */
17         puts("Usage : VOCD *.sp *.ar ");
18         getch();
19         exit(1); /* 结束程序执行 */
20     }
21     fname1=argv[1]; fname2=argv[2];
22     conv();
23 }
24 /*-----*/
25 conv()
26 {
27     FILE *in, *out;
28     unsigned i;
29     unsigned char c;
30
31     /* 载入原始语音文件 */
32     in=fopen(fname1,"rb");
33     if (in==NULL)
34     {
35         printf("cannot open the file : %s ok !!! ",fname1);
36         exit(1);
37     }
38     printf("open the file : %s ok !!!\n",fname1);
39     /* 打开输出文件 */
40     out=fopen(fname2,"wt");
41     if (out==NULL)
```

```

42     {
43         printf("cannot open the file : %s ok !!! ",fname2);
44         exit(1);
45     }
46     printf("open the file : %s ok !!!\n",fname2);
47
48     /* 读取语音文件数据 */
49     i=0;
50     while (!feof(in))
51     {
52         c=getc(in)+128; i++;
53         fprintf(out,"0x%02x,", c);
54         if(i%10==0) { printf("."); fprintf(out,"\n"); }
55     }
56     fprintf(out,"0x%02x, 0x%02x;", 0x55,0xaa); /* 加上文件结束代码 */
57     /* 关闭文件 */
58     fclose(in); fclose(out);
59 }
60 /*-----*/

```

14.4 让 8051 电路板播放语音

既然已经产生了 C 语言可以编译的语音数据文件 COMP.AR, 只要将此数据文件包括到控制程序中一起来进行编译即可, 在程序中如何播放语音数据, 只要用一循环将数组 buf[] 内的数据往语音卡的输出端口输出, 便可以听到 8051 电路板播放出声音, 其中数组的大小并不需要精确的控制, 只要不大于实际数组的大小, 例如 COMP.SP 文件的大小为 4096, 于是放音的长度为 4000 或 4080 都可以。

由于控制程序用 MICRO-C51 C 语言编译器来编译, 而此编译器将初始化的数组数据在程序执行前读入 RAM 中, 因此在做此实验前, 记住将电路板上的 SRAM (6116) 换成 62256 大容量的 SRAM, 否则内存不足, 无法正常运行。

执行结果

8051 电路板播放声音的控制程序为 SAY1.C, 程序执行后会播放出“电脑”的录音, 再次按下“1”可以再播放一次。

程序清单

```
1 /* SAY1.C */
```

```
2 #include "8051io.h"    /* 载入 MC51 头文件 */
3 #include "8051reg.h"
4 #include "8051bit.h"
5 #include "8051int.h"
6 #include "c: p51.h"    /* 载入 P51 I/O 控制头文件 */
7 char *title="P51 PCB play Voice data";
8 #define CWORD  0x88    /* PA PB PC0~3 o/p  PC4~7 i/p */
9 char key; /* 键盘扫描值 */
10 char act[4]={0xfe,    0xfd, 0xfb, 0xf7}/* 键盘扫描控制代码 */
11 /* 16 个键盘按键 */
12 char skey[16]={'A','B','C','D',    '3','6','9','#','2',
13              '5','8','0','1','4',    '7','*'};
14 char scan_key();/* 键盘扫描控制程序 */
15
16 /*-----*/
17 delay(int t) /* 延迟子程序 */
18 {
19     int i,j;
20     for(i=0; i<t; i++)
21         for(j=0; j<10; j++);
22 }
23 /*-----*/
24 led_b1() /* 工作 LED 闪动 */
25 {
26     /* blink RED led P1.7 */
27     char i;
28
29     for(i=0; i<6; i++)
30     {
31         cplbit(P1.7);
32         delay(50);
33     }
34 }
35 /*-----*/
36 be() /* 蜂鸣器“嘀”一声 */
37 {
```



```
38 char i;
39
40 for(i=0; i<100; i++)
41 {
42     cplbit(P1.0); /*
43     delay(1);
44 }
45
46 cplbit(P1.7); /* LED */
47 delay(100);
48 cplbit(P1.7);
49 }
50 /*-----*/
51 char scan_key() /* 键盘扫描控制程序 */
52 {
53     char i,j, find, ini, inj;
54     char in;
55
56     find=0; /* 清除按键标志 */
57     for(i=0; i<4; i++)
58     {
59         /* 从 PC 端口(PC0~PC3) 输出扫描控制信号 */
60         pc=act[i];
61         delay(3);
62
63         /* 从 PC 端口(PC4~PC7) 读取按键返回代码 */
64         in=pc;
65         in=in>>4;
66         in=in| 0xf0;
67
68         /* 检查是否按键 ? */
69         for(j=0; j<4; j++)
70             if(act[j]==in)
71             {
72                 find=1; /* 设定按键标志 */
73                 inj=j; ini=i; /* 记录扫描指针值 */
```

```
74     }
75     }/* scan 1 time */
76
77     if(find==0) return 0; /* 没按键传回 0 值 */
78
79     /* i,j --> key : 0~15 */
80     key=ini*4+inj; /* 计算按键值 */
81     return 1; /* 有按键传回键值 */
82 }
83 /*-----*/
84 main()
85 {
86     char c;
87
88     led_b1(); /* 工作 LED 闪动 */
89     cr=CWORD; /* 设定 8255 工作模式 */
90     serinit(9600); /* 初始化串行接口 */
91     /* 显示工作消息 */
92     printf("%s\n",title);
93     printf("> K1 --> SAY \n");
94     say(); /* 语音播放 */
95
96     while(1)
97     { /* 若有按键则处理 */
98         if(scan_key()==1)
99         {
100             be(); /* “嘀” 一声 */
101             c=skey[key]; /* 按键值转换 */
102             switch(c)
103             { /*语音播放 */
104                 case '1': say(); break;
105                 default : break;
106             }
107         } /* if keyed */
108     } /* loop */
109 }
```

```
110 /*-----*/
111 #include "c: comp.ar" /* 载入语音数据文件 */
112 say()
113 {
114     int i; /* 从 DAC 语音输出端口输出数据*/
115     for(i=0; i<4000; i++) pa=buf[i];
116 }
```

14.5 习 题

1. 若数字语音录音的采样频率为 8K，而分辨率为 8 位，则录音 1 分钟将占用多少内存？
2. 若数字语音录音的采样频率为 8K，而放音频率各为 4K 及 16K，则会产生什么样的效果？
3. 编写一控制程序，在 8051 电路板上播放出“欢迎光临”及“谢谢光临”的录音。

第 15 章 8051 控制 PC I/O 接口卡

在 8051 多功能控制板 P51_PCB 上除了有一些常用的控制芯片外, 还设计有一插槽, 可以用 8051 来模拟简单的 PC I/O 插槽, 一些 PC 上的 I/O 接口卡可以不须修改线路, 便可移植到 8051 单板单片机上来执行, 只要重新改写 8051 的控制程序即可, 如此一来可以实现硬件 I/O 接口资源共享的功能。

本章将选择以下两块 PC I/O 实验接口卡来做实验:

1. PC/8051 语音控制实验卡(SP_CARD);
2. PC/8051 多功能实验卡(MIO)。

15.1 8051 模拟 PC I/O 插槽信号

本节将介绍 8051 模拟 PC I/O 插槽信号, 了解此一接口设计原理, 我们便可以在 8051 单板上控制简单的 PC I/O 接口实验卡。

对于 PC AT I/O 接口的连接有 3 种总线, 即地址总线、数据总线及控制总线, 其中控制总线对 I/O 接口设计较常用到的信号有以下几种:

1. RESET 引脚 B2;
2. IOW, 引脚 B13;
3. IOR, 引脚 B14;
4. AEN, 引脚 A11。

其中引脚 A 表示零件面的金手指引脚, 引脚 B 表示焊锡面的金手指引脚。所表示的意义如下:

- RESET : 作为系统重置用, 高电位工作。
- IOW : CPU 写数据到输出单元的控制信号。
- IOR : CPU 读取输入单元数据的控制信号。
- AEN : Address Enable, 地址使能, 是控制地址线工作的信号, 使 CPU 与 DMA 元件对系统总线的控制不相冲突。当 AEN 是低电位时, 由 CPU 取得控制权。AEN 如为高电位, 则 DMA 获得控制权。这些控制权包括地址线、数据线、读写控制信号及内存、输入输出单元的使用权限。

了解上述的控制信号后, 如要在 8051 单板上设计一个能模拟上述工作的 PC XT I/O 插槽接口并不困难, 图 15-1 所示即是这样的一个插槽接口定义:

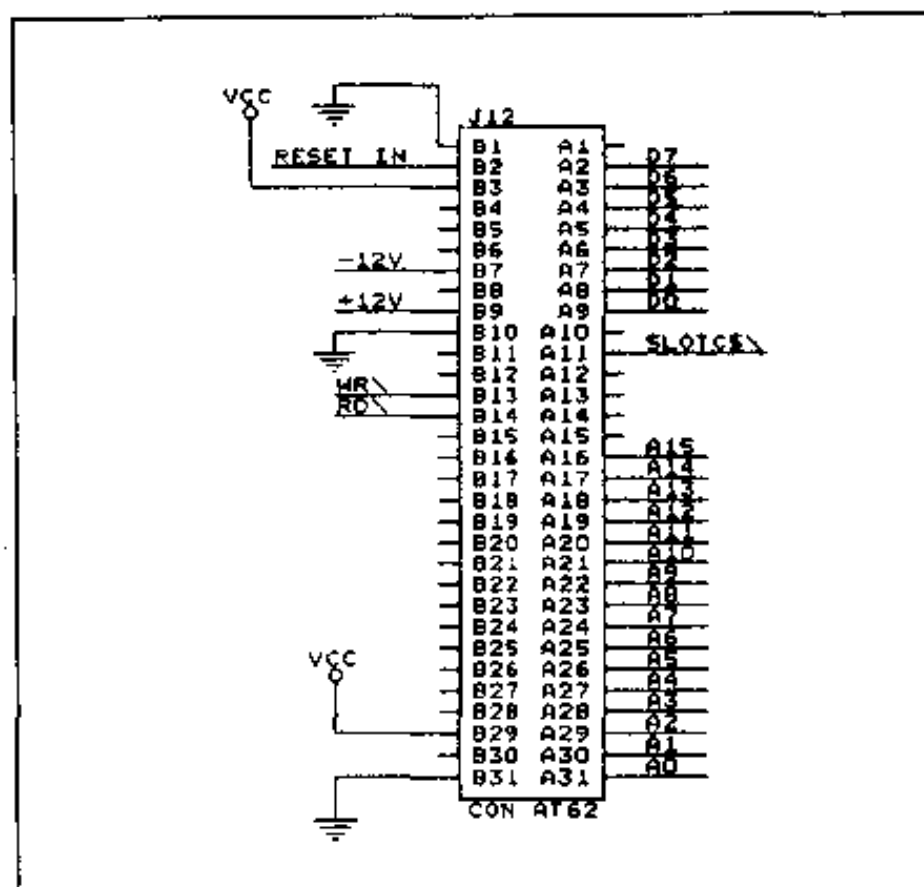


图 15-1 8051 模拟 PC I/O 接口信号

1. 地址总线由单片机地址线 A0~A15 提供。
2. 数据总线由单片机数据线 D0~D7 供给。
3. RESET 信号由 RESET IN 提供。
4. IOW 由 8051 WR 提供。
5. IOR 由 8051 RD 提供。
6. AEN 由解码器产生的 SLOTCS 提供，地址解码为 AxxxH，低电位工作。
7. 加入适当的电压 +5V、GND、+12V、-12V。

定义了接口信号后，对于 I/O 的解码地址只要加以修改即可。例如原先在 PC 上的 I/O 地址为 250H，修改后在 8051 上便成了 A250H。

15.2 PC/8051 语音控制实验卡介绍

语音卡是一块 PC 上的 I/O 接口卡，此块接口卡上含有 A/D 接口（使用 AD0804）、D/A 接口（使用 DA08）和模拟放大电路，可以实现语音的录放功能，既然它是设计在 PC 上的一块 I/O 接口卡，同理也可以将其移植到 8051 单板上来做语音的录放工作，当然控制程序要重新改写并不困难。

此块语音卡在 8051 单板上除了做录放音外，最主要是可以用它来播放语音提示，很多电子产品都有提供语音接口的功能，所以我们可以利用 8051 单板来实现语音提示，制造一些有趣的、生动的效果。

语音卡设计的目的，主要是在 PC 上做单片机语音识别(有 TURBO C 声控程序库的支持)的功能，如果能顺利地将其转移到 8051 单板上来做控制，将可增加它的适用价值。语音卡的功能有以下的一些特点：

- 具有 8 位的 A/D 和 D/A 转换接口。
- 使用指向式(一般唱歌用)麦克风。
- 输出可直接驱动小型喇叭。
- I/O 解码位置可以调整, 避免与现有 I/O 接口卡相冲突。
- 可以在 PC 上做一般的 I/O 控制。
- 结合 PC 上的控制软件完成特定受话者的语音识别。
- 所有硬件及软件驱动程序全部公开, 功能可以继续做扩充。
- 在 PC 上支持有语音分析、编辑、识别综合程序。
- 在 PC 上支持有 TURBO C 声控程序库。
- 在 PC 上可连接红外线接口控制板(IR_PCB)做红外线接口实验。
- 在 PC 上可连接无线电遥控模块(RF51)做 PC 无线电遥控的接口实验。
- 可在 8051 单板(P51_PCB)上做语音录放音的实验。
- 提供语音卡印刷电路板供学生硬件 DIY 专题制作。

15.3 语音卡电路设计

看过语音控制实验卡的特性介绍后, 本节将说明语音卡的电路设计原理, 其使用到的硬件有 A/D 及 D/A 芯片, 只要结合此二芯片, 加上必要的模拟处理接口电路, 即可完成接口的设计。为了实验方便, 我们加上 8255 芯片做数字接口处理而设计成一片接口卡, 专门用于声音录放控制实验, 我们称为语音卡。

语音卡整个电路可以分为以下 3 大部分:

1. A/D 线路;
2. 数字接口;
3. D/A 线路。

图 15-2 为 A/D 转换的工作原理图, 图 15-3 则为 D/A 转换的工作原理图。

1. A/D 线路

A/D 线路由模拟语音信号处理线路及 A/D 转换线路组成。图 15-4 为整个模拟语音信号处理的电路图, 即包括图 15-2 所示的前端三部分: 前置放大器、4KHz 低通滤波器及电位调整线路。首先语音信号由麦克风输入, 一般麦克风的输入电位只有几十毫伏左右, 因此经过约 100 倍的前置放大器 U7B 放大至 1 伏特左右的电位, 以便驱动后方相关线路, 而实际的放大倍数可以由可变电阻 VR1 来调整, 以配合使用的需要。

此外为了满足采样定理——至少要以信号最高工作频率的 2 倍来进行取样, 而一般语音的说话频谱分布大部份小于 4KHz, 因此低通滤波器(由 U7C 及相关 R、C 电路组成)的截止频率设计为 4KHz, 而系统的取样频率(由个人电脑的驱动程序控制)定为 8K, 即每秒钟取样 8000 点语音数据, 每一点数据以一个字节来表示。

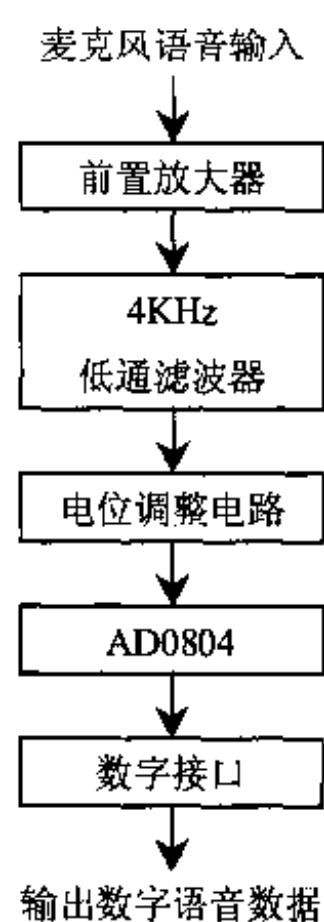


图 15-2 语音卡 A/D 转换工作原理图

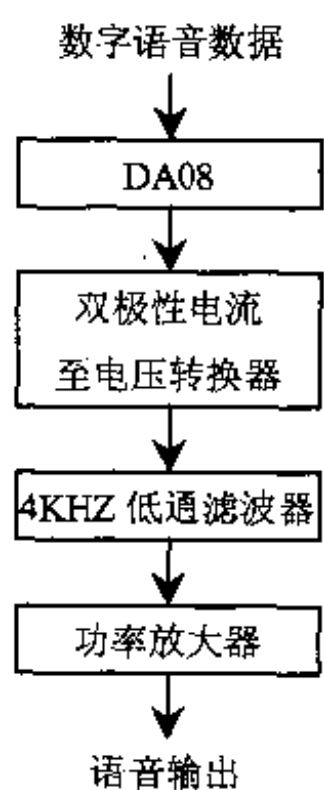


图 15-3 语音卡 D/A 转换工作原理图

经过低通滤波器的语音信号，然后接到由 R16、R17 及 U7D 所构成的电位调整线路，将原先具有双极性的语音信号电位转换至 0 到 5V 的范围，以满足 AD0804 的接口信号要求。图 15-5 为 8 位 A/D 转换电路，构成此电路的主要元件是 AD0804，将模拟信号（0 至 5V，由 $V_{in}(+)$ 输入）转换为 8 位的数字数据，其转换时间为 100 微秒（ μs ）。电路中 R1 及 C20 提供 A/D 所需的工作时序，实际的工作频率为

$$1/1.1RC=1/[1.1 \times 10K \times 100P]=910KHz$$

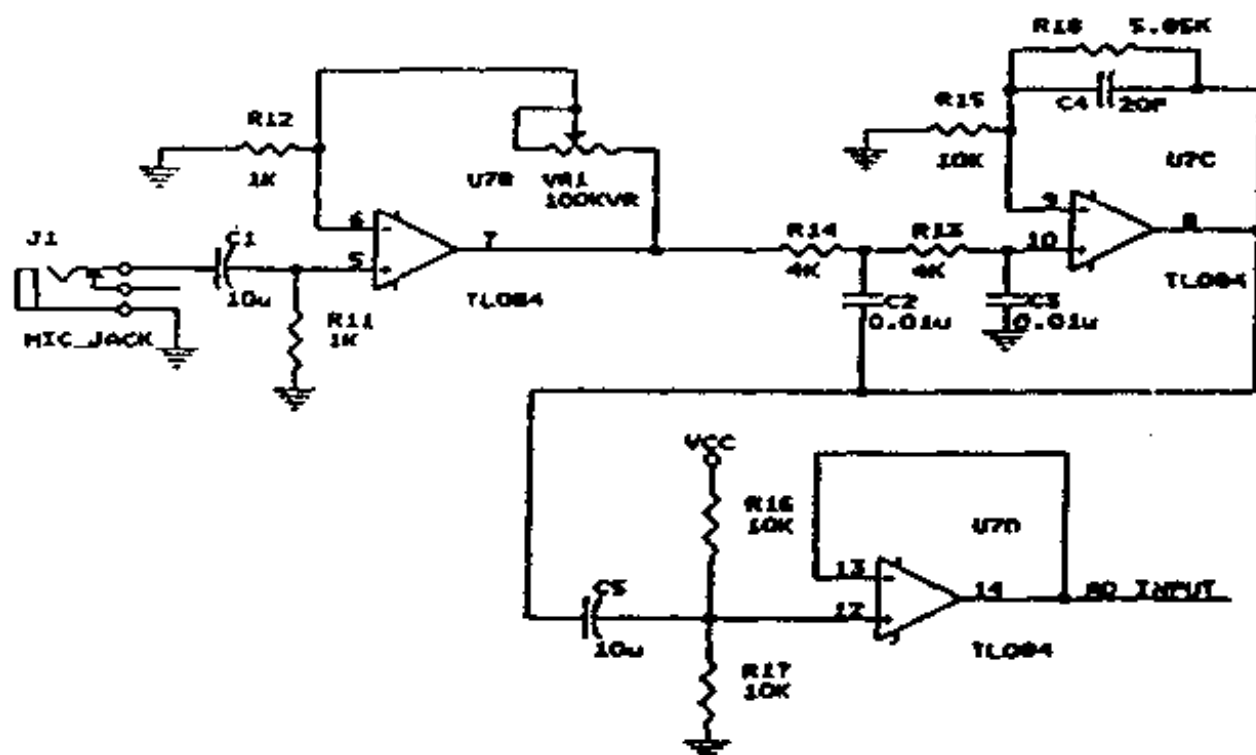


图 15-4 模拟语音信号放大电路

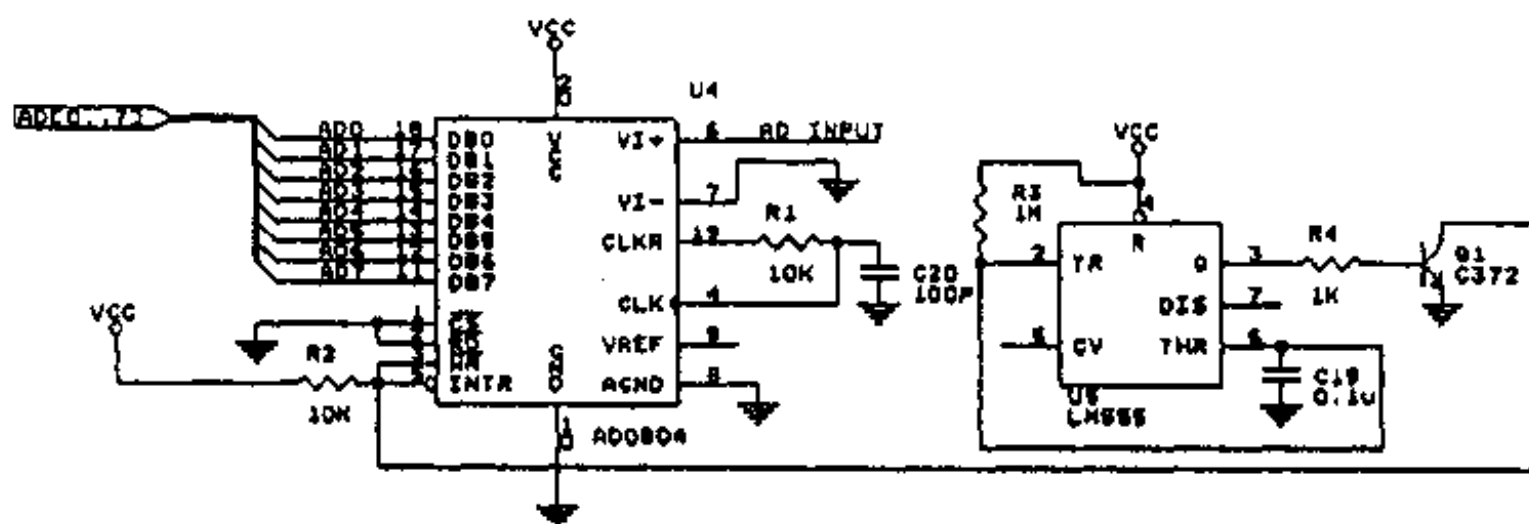


图 15-5 A/D 转换电路

此外为了简化与 PC 的接口设计，在此利用 555 产生单击的起始触发信号，用于在电源接通时产生低电位工作脉冲，送入 WR 引脚，使芯片能开始转换，在转换完后由 INTR 产生低电位工作脉冲，又促使 WR 引脚为低电位，使得芯片又重新进行转换的工作，如此一直循环下去。以此方式设计的话，一些控制信号，像 CS、RD、WR 和 INTR，便不必另外处理了。最后转换完的数字语音信号，再经过数字接口而读至 PC 来加以处理。

2. 数字接口

图 15-6 为数字接口电路。PC 方面的 I/O 解码是由 U1、U2 及 U3 来担任，其解码地址为 200H、210H 至 270H（采用部分解码），可经过 DIP 开关 S1 来选择。避免与现有任何接口卡的 I/O 端口相冲突。任何时候 S1 只有一个位 ON，其余为 OFF。在电路中主要靠 8255 担任数字接口的工作，8255 本身有 3 个 I/O 端口，PA、PB 及 PC。其中 PA 被规划为输入，用于 A/D 语音输入取样，PB 被规划为输出，用于 D/A 数字放音，端口 C 则用于特殊的硬件扩充。

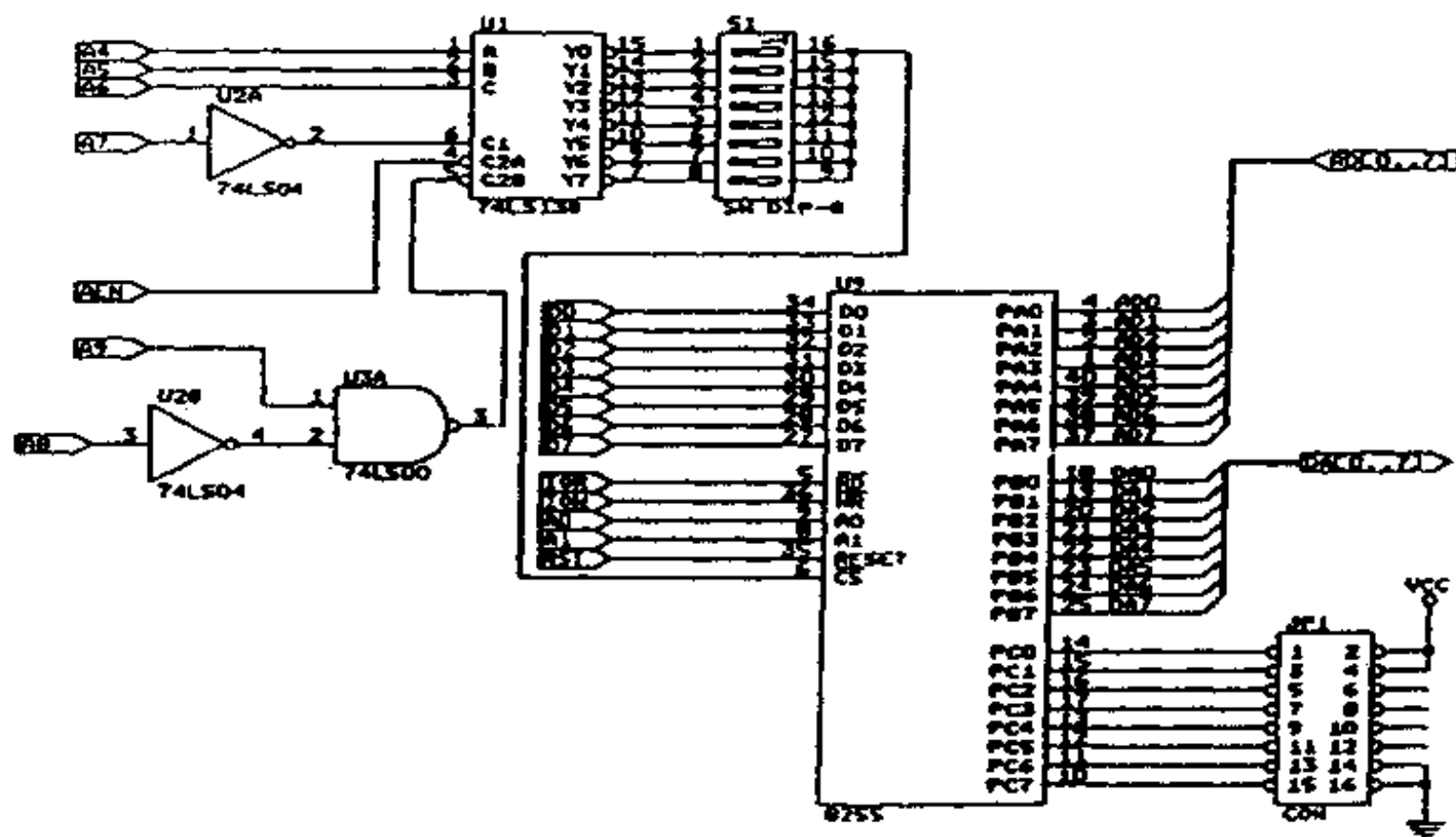


图 15-6 数字接口电路

3. D/A 电路

图 15-7 为 D/A 的电路设计，我们所用的 D/A 芯片是 DA08，分辨率为 8 位，具有双极性电流驱动的数字至模拟转换器。由于我们所要转换输出的信号为音频信号，其本身是一双极性的信号，因此对杂音的处理尤其重要。在无信号输出时，希望其输出对地是零电位，在此应用另一组运算放大器 U7A 作为电流到电压的转换器，U6 的正向输出至运算放大器的负端输入，而反向接至运算放大器的正端输入，实现双极性控制的目的。

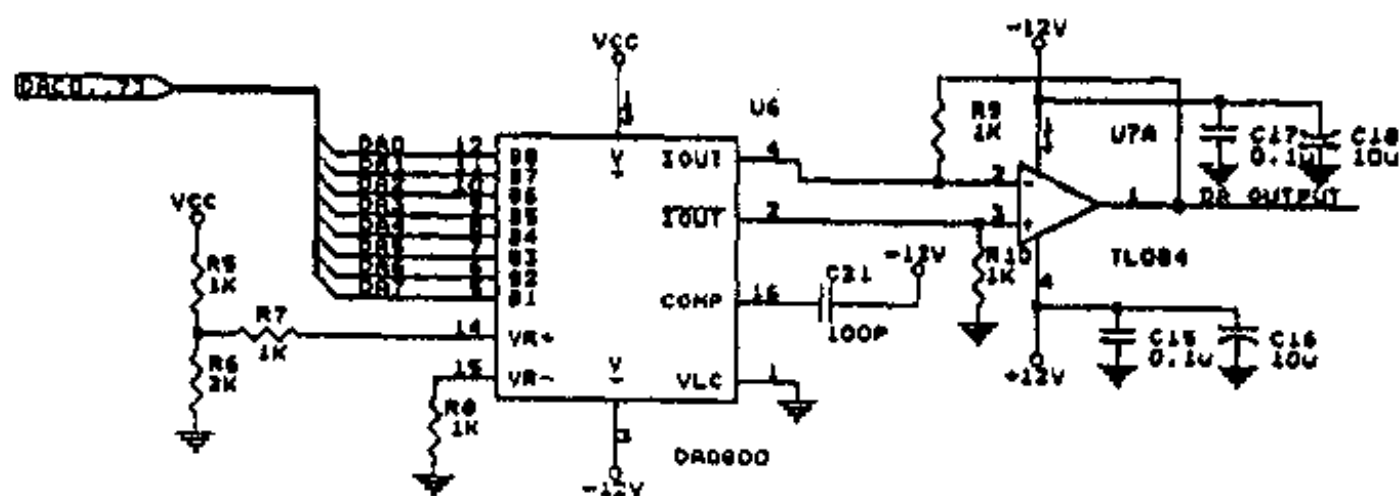


图 15-7 D/A 转换电路

原先由 VR+提供有 2mA 的电流, 在满刻度输入时, I/O 引脚可以输入 1.992mA ($2 \times 255/256$) 的电流, 在理论上双极性控制时, 可以达到正负 1.992V (1.992×1), 此输出信号的振幅足以驱动后级的相关模拟电路。

图 15-8 为语音输出滤波及放大电路。原先的数字信号经过 D/A 转换、双极性电流至

电压变换后已成为模拟的语音信号，经 4KHz 的低通滤波器（由 R20 及 C22 组成）以便滤除 4KHz 以上的高频刺耳杂音，再送往 U8 小功率放大器，做适当的功率提升而驱动喇叭，其中可变电阻 VR2 用于音量控制。

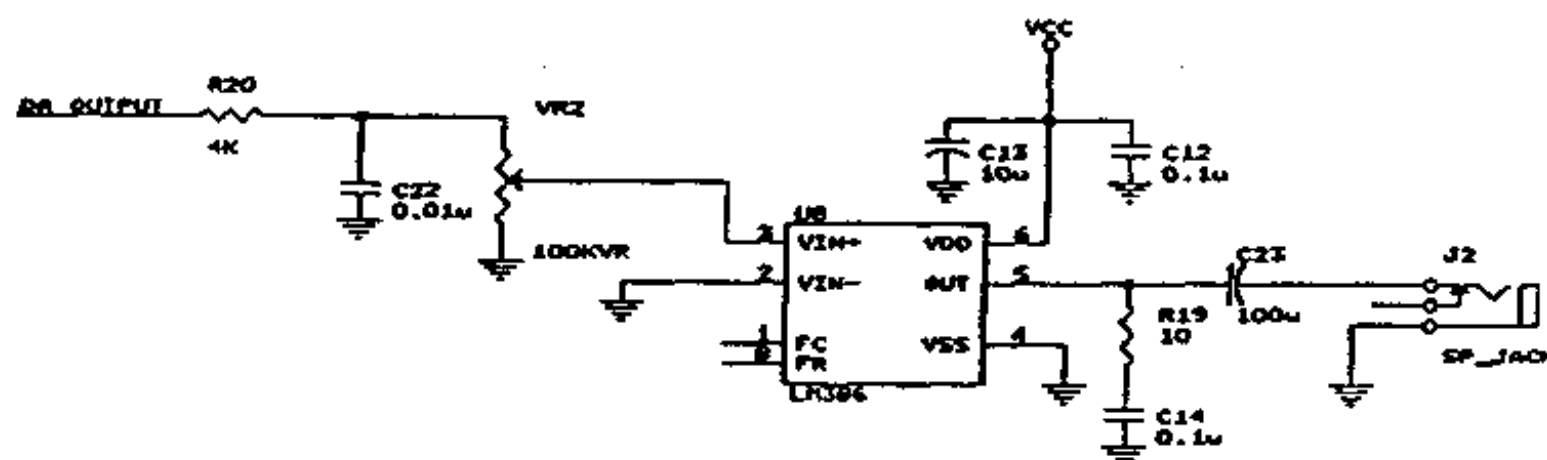
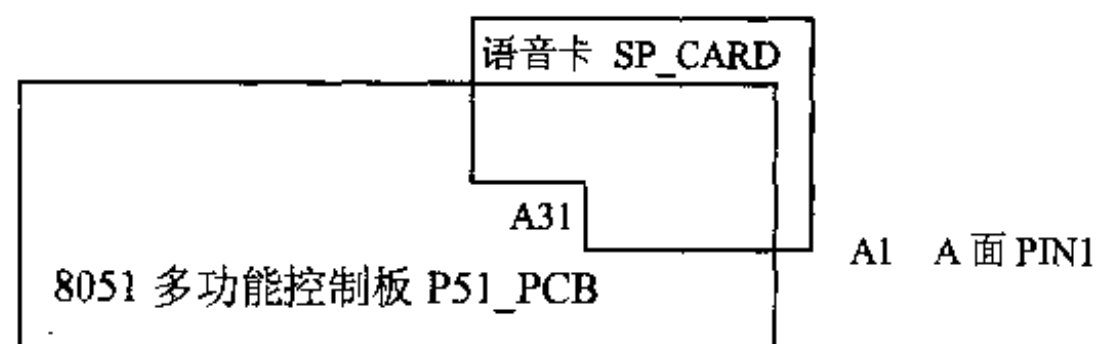


图 15-8 语音输出滤波器及放大器电路

15.4 8051 单板控制语音卡

看过 8051 模拟 PC I/O 插槽信号及语音卡控制电路后，本节将介绍如何用 8051 单板来控制语音卡，做一些简单的语音录放实验。整套 8051 单板控制语音卡系统组成如下：



由 8051 多功能控制板(P51_PCB)及上一节所介绍的语音卡所组成，此块语音卡除了可以在 PC 上做语音录放及语音识别的实验外，还可以在 8051 单板上作相关的实验，其关键在于单板上设计有一与 PC 部分 I/O 信号兼容的插槽，因此完全不用修改语音卡电路便可由 8051 来控制。

在 PC 上用 TURBO C 控制语音卡的录放工作，其采样频率为每秒 8K，利用系统计时器 8253 通道 0 来定时对 A/D 端口读取语音数据，若在 8051 单板上程序反而容易控制，因为 8051 本身有两组计数器可供我们使用。当然对声音的录放其采样频率大小会影响声音的品质，当采样频率越高时，放音的品质越好，却要耗用较大的内存。在 PC 上内存空间可能够用，可以动态配置一块 64K 字节的内存不存在问题。但在 8051 单板上最多 SRAM 只能使用 62256 芯片（32K 字节），这是要考虑的因素，所以适当的选择采样频率是十分重要的。

最简单的采样频率设定可以用一延迟参数来表示，当延迟参数较小时，采样频率就高，重播的效果较好；延迟参数值较大时，采样频率就低，声音会变得低沉。这时录音

与放音均使用相同的速率。若录音采样频率正常，而放音速度慢些或快些便可制造出一些特殊效果。

实验注意事项

1. 将单板上的 SRAM (6116) 换成 62256 大容量的 SRAM，否则内存不足，无法正常工作。
2. 在插入语音卡时请注意方向，语音卡零件面朝向 7 段数码管，插槽引脚 1 在右方。
3. 请先将原先的 J4 +12V 外部电源供给器移开，以免重复供给 +12V 电源，造成不可预期的后果。
4. 在单板 J11 引脚加入 +12V 及 -12V 电压，注意极性。

执行结果

程序 SPP51.C 为利用 8051 单板 P51_PCB 来控制语音卡的测试程序，请将 RS232 线连至单板上、PC 上并执行 SE.EXE 连线程序，当 8051 程序执行后，可在 PC 的显示器上看到如下的画面：

```
> K1 --> MIC    input
> K2 --> SP      output
> K3 --> SP      output1
> K4 --> SP      output2
> K5 --> int. D/A SP output
```

8051 单板上的按键功能如下：

- 按键 “1” ： 做语音录音的工作；
- 按键 “2” ： 语音重播出来；
- 按键 “3” ： 语音重播的速度变快；
- 按键 “4” ： 语音重播的速度变慢；
- 按键 “5” ： 由单板上的 J6 喇叭输出语音。

先按下键 K1，开始做语音录音的工作。一旦声音缓冲区录满数据后则结束录音，并将刚刚录到的声音重播出来。按下 K2 键会将声音以正常速度重播一次，按下 K3 键重播的速度变快，按下 K4 键则重播的速度变慢，因此可以得到不同的效果，试着去录一些有趣的声音来制造不同的特殊效果。

程序清单

```
1 /* SPP51.C */
2 #include "8051io.h" /* 载入 MC51 头文件 */
3 #include "8051reg.h"
```

```

4  #include  "8051bit.h"
5  #include  "8051int.h"
6  #include  "p51.h"
7
8  char *title= "P51 PCB control SP_CARD";
9  /* P51_PCB 控制语音卡的 I/O 地址 */
10 #define AD_PORT  (*(char*) 0x0A250) /* A/D 输入 */
11 #define DA_PORT  (*(char*) 0x0A251) /* D/A 输出 */
12 #define IO_PORT  (*(char*) 0x0A252) /* I/O 扩充 */
13 #define CW_PORT  (*(char*) 0x0A253) /* 控制寄存器设定 */
14
15 #define MAX_LEN  7000 /* 语音缓冲区长度 */
16 #define CWORD  0x88 /* P51 上 8255 控制寄存器设定 */
17 /* 8255 PA PB PC0~3 o/p PC4~7 i/p */
18 signed char buf[MAX_LEN]; /* 语音缓冲区 */
19 int len; /* 语音长度 */
20 char key; /* 键盘扫描值 */
21 char act[4]={0xfe, 0xfd, 0xfb, 0xf7} /* 扫描控制信号 */
22 /* 16 个键盘按键 */
23 char skey[16]={'A','B','C','D', '3','6','9','#','2',
24              '5','8','0','1', '4','7','*'};
25 char scan_key(); /* 键盘扫描控制程序 */
26 /*-----*/
27 delay(int t) /* 延迟子程序 */
28 {
29     int i,j;
30     for(i=0; i<t; i++)
31         for(j=0; j<10; j++);
32 }
33 /*-----*/
34 led_bl() /* 工作 LED 闪动 */
35 {
36     /* blink RED led P1.7 */
37     char i;
38
39     for(i=0; i<6; i++)

```

```
40  {
41      cplbit(P1.7);
42      delay(50);
43  }
44  }
45  /*-----*/
46  be() /* 蜂鸣器“嘀”一声 */
47  {
48      char i;
49
50      for(i=0; i<100; i++)
51      {
52          cplbit(P1.0);
53          delay(1);
54      }
55
56      cplbit(P1.7); /* LED */
57      delay(100);
58      cplbit(P1.7);
59  }
60  /*-----*/
61  char scan_key() /* 键盘扫描控制程序 */
62  {
63      char i,j, find, ini, inj;
64      char in;
65
66      find=0; /* 清除按键标志 */
67      for(i=0; i<4; i++)
68      {
69          /*由 PC 端口(PC0~PC3) 送出扫描控制信号 */
70          pc=act[i];
71          delay(3);
72
73          /* 由 PC 端口(PC4~PC7) 读取按键返回代码 */
74          in=pc;
75          in=in>>4; /* 右移 4 位 */
```

```
76 in=in|0xf0; /* 高 4 位设为 1 */
77
78 /*检查是否按键 ? */
79 for(j=0; j<4; j++)
80     if(act[j]==in)
81     {
82         find=1; /* 设定按键标志 */
83         inj=j; ini=i; /* 记录扫描指针值 */
84     }
85 } /* scan 1 time */
86 /* 没按键传回 0 值 */
87 if(find==0) return 0;
88
89 /* i, j --> key : 0~15 */
90 key=ini*4+inj; /* 计算按键值 */
91 return 1; /* 有按键传回 1 */
92 }
93 /*-----*/
94 main() /* 主程序 */
95 {
96     int i;
97     char c;
98
99     len=0;
100     led_bl(); /* 工作 LED 闪动 */
101
102     cr=CWORD; /* 设定 P51 8255 工作模式 */
103     CW_PORT=0x91; /* 设定语音卡工作模式 */
104
105     serinit(9600); /* 初始化串行接口 */
106     /* 显示工作消息 */
107     for(i=0; i<35; i++) printf( "-"); printf( "\n");
108     printf( "%s\n", title);
109     for(i=0; i<35; i++) printf( "-"); printf( "\n");
110     /* 显示操作说明 */
111     printf( "> K1 --> MIC input \n");
```

```

112 printf("> K2 --> SP output \n");
113 printf("> K3 --> SP output1 \n");
114 printf("> K4 --> SP output2 \n");
115 printf("> K5 --> int. D/A SP output \n");
116 printf("Please select key ..... \n");
117
118 while(1) /* 无穷循环 */
119 {
120 if(scan_key()==1) /* 如果有按键 */
121 {
122 be(); /* “滴” 一声 */
123 c=skey[key]; /* 按键转换 */
124 switch(c) /* 执行相对控制程序 */
125 {
126 case '1': pro_key1(); break; /* 做语音录音的工作 */
127 case '2': pro_key2(); break; /* 语音重播出来 */
128 case '3': pro_key3(); break; /* 语音重播的速度变快 */
129 case '4': pro_key4(); break; /* 语音重播的速度变慢 */
130 case '5': pro_key5(); break; /* 由单板上的 J6 喇叭输出语音 */
131 /* ..... */
132 default : break;
133 }
134 } /* if keyed */
135 } /* loop */
136 }
137 /* ..... */
138 pro_key1() /* 做语音录音的工作 */
139 {
140 be(); /* “滴” 一声 */
141 printf("begin MIC input .....");
142 len=mic_ip(); /* 语音录音 */
143 printf("length=%d\n", len);
144 /* 录音信号电位调整 */
145 /* for(i=0; i<len; i++) buf[i]-=128; */
146 /* 观察录音信号值 */
147 /* printf("data : \n");

```

```
148 for(i=0; i<len; i++) printf( "%d ", buf[i]);
149 printf( "\n");*/
150 /* 语音重播出来 */
151     printf( "Talking .....");
152     da(0, len);
153 }
154 /*-----*/
155 pro_key2() /* 语音重播出来 */
156 {
157     clrbit(P1.7); /*LED 亮 */
158     printf( "Talking .....");
159     da(0, len);
160     setbit(P1.7); /*LED 灭 */
161 }
162 /*-----*/
163 pro_key3() /* 语音重播的速度变快*/
164 {
165     clrbit(P1.7);
166     printf( "Talking .....");
167     da1(0, len);
168     setbit(P1.7);
169 }
170 /*-----*/
171 pro_key4() /* 语音重播的速度变慢*/
172 {
173     clrbit(P1.7);
174     printf( "Talking .....");
175     da2(0, len);
176     setbit(P1.7);
177 }
178 /*-----*/
179 pro_key5() /* 由单板上的 J6 喇叭输出语音*/
180 {
181     clrbit(P1.7);
182     printf( "Talking .....");
183     da0(0, len);
```



```
184  setbit(P1.7);
185  }
186  /*-----*/
187  int mic_ip(void)
188  {
189  int i,d;
190
191  i=0; clrbit(P1.7); /* LED 亮 */
192  while(1) /* 循环 */
193  {
194    for(d=0; d<3; d++); /* 延迟 */
195    buf[i]=AD_PORT; /* 由 A/D 端口读取数据 */
196    i++; /* 录音指针加 1 */
197    if(i>=MAX_LEN) break; /* 检查语音缓冲区是否满了 */
198  }
199  setbit(P1.7); /* LED 灭 */
200  return i; /* 传回录音长度 */
201  }
202  /*-----*/
203  da(unsigned int bp, unsigned int ep)
204  { /* 语音正常速度重播出来 */
205  unsigned i;
206  int d;
207
208  for(i=bp; i<ep; i++)
209  {
210    for(d=0; d<3; d++); /* 延迟 */
211    DA_PORT=buf[i];
212  }
213  printf( "End of talk ..\n");
214  }
215  /*-----*/
216  dal(unsigned int bp, unsigned int ep)
217  { /* 语音重播的速度变快 */
218  unsigned i;
219  int d;
```

```
220
221  for(i=bp; i<ep; i++)
222  {
223      /* quick .....*/
224      for(d=0; d<2; d++); /* 延迟 */
225      DA_PORT=buf[i];
226  }
227  printf( "End of talk ..\n");
228  }
229  /*-----*/
230  da2(unsigned int bp, unsigned int ep)
231  { /* 语音重播的速度变慢*/
232  unsigned i;
233  int d;
234
235  for(i=bp; i<ep; i++)
236  {
237      /* slow .....*/
238      for(d=0; d<4; d++); /* 延迟 */
239      DA_PORT=buf[i];
240  }
241  printf( "End of talk ..\n");
242  }
243  /*-----*/
244  da0(unsigned int bp, unsigned int ep)
245  { /* 由单板上的 D/A 端口驱动喇叭输出语音 */
246  unsigned i;
247  int d;
248
249  for(i=bp; i<ep; i++)
250  {
251      for(d=0; d<3; d++); /* 延迟 */
252      pa=buf[i];
253  }
254  printf( "End of talk ..\n");
255  }
```

256 /*-----*/

15.5 PC/8051 多功能实验卡介绍

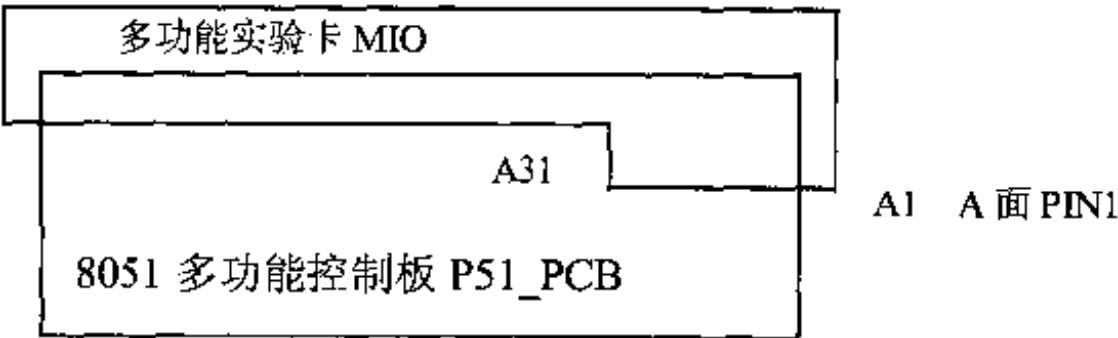
在 PC I/O 的接口设计及实验中, 我们免不了要焊接一些基本的接口控制芯片, 做相关的实验并做功能扩充, 焊接是需要有点技术及耐心的, 有时候辛辛苦苦焊接了测试正常的作品一不小心扯断了一条线, 得又要重新 DEBUG。有时候在做复杂一点的实验时, 往往需要再有一块额外的接口卡, 重新焊接又太辛苦了, 这是许多对电脑硬件制作有兴趣的朋友共同的心声吧! 于是在此我们总结了过去的一些经验而设计了一块在 PC 上的多功能 I/O 接口控制卡, 使今后的专题设计或是接口实验上可以提高效率。最重要的一点是此块接口卡提供标准的 I/O 接口设计, 可以不经线路修改就可移植到 8051 单板单片机上来执行, 只要重新改写 8051 的控制程序即可, 如此一来可以实现硬件 I/O 接口资源共享的功能。基本上此块接口卡有以下的一些特性:

- 一块实验卡可以用来学习 PC 及 8051 接口自动控制。
- 提供两块 8255 及 8253 可编程控制芯片。
- 使用 AD0804 做 A/D 接口转换。
- 使用 DA0800 做 D/A 接口转换。
- 可扩充的 LCD 接口电路。
- 所有 I/O 控制信号均可由信号线连接到外部做实验。
- 配合无线遥控模块(RF51)可做无线遥控相关的控制实验。
- 提供 AD_LIB 声效卡兼容的声效接口, 玩 GAME 时可以播放声音。
- 内含多种乐器的可编程声效发生器 UM3567 接口。
- 本块 PC 多功能接口实验卡不必更改电路便可移植到 8051 多功能控制板(P51_PCB)上做相关的控制实验。

以上是多功能实验卡的特性说明, 有关多功能实验卡的详细控制实验, 请参考陈西文著“PC I/O 接口实作入门与应用——使用 C”一书, 书中对此块实验卡的控制程序及应用有详细的介绍。

15.6 8051 单板控制多功能实验卡

看过 8051 模拟 PC I/O 插槽信号及多功能实验卡的特性后, 本节将介绍如何由 8051 单板 P51_PCB 来控制多功能实验卡 MIO, 做一些简单的测试实验。整套控制系统组成如下:



PC/8051 多功能实验卡上可以扩充至 7 种 I/O 芯片，在 PC 上其芯片解码地址及功能如下：

芯片	解码信号	地址	功能
8255	8255A	0X300~0X303	8255 I/O 控制
8255	8255B	0X304~0X307	8255 I/O 控制
8253	8253	0X308~0X30B	8253 计时器
LCD	LCD	0X30C、0X30D	LCD 接口
A/D D/A	ADDA	0X310	A/D D/A 控制
UM3567	UM3567	0X314、0X315	声效控制
YM3812	YM3812	0X388、0X389	AD LIB 声效

8051 单板上控制多功能实验卡，其芯片解码地址如下：

芯片	解码信号	地址	功能
8255	8255A	0XA300~0XA303	8255 I/O 控制
8255	8255B	0XA304~0XA307	8255 I/O 控制
8253	8253	0XA308~0XA30B	8253 计时器
LCD	LCD	0XA30C、0XA30D	LCD 接口
A/D D/A	ADDA	0XA310	A/D D/A 控制
UM3567	UM3567	0XA314、0XA315	声效控制
YM3812	YM3812	0XA388、0XA389	AD LIB 声效

实验注意事项

1. 在插入实验卡前请注意方向，实验卡零件面朝向 7 段数码管，插槽引脚 1 在右方。
2. 请先将原先的 J4 +12V 外部电源供给器移开，以免重复供给+12V 电源，造成不可预期的结果。
3. 在单板 J11 引脚加入+12V 及-12V 电源，注意极性。
4. 由于实验卡及 8051 单板 P51_PCB 使用的 IC 负载过大，请将 P51_PCB 部分未使用的 IC 移开减轻负载，否则 U9 7805 +5V 稳压 IC 负载过大，恐怕有烧毁的危险。

执行结果

程序 TMIO51.C 为利用 8051 单板 P51_PCB 来控制实验卡的测试程序，请将 RS232

线连至单板上, PC 上并执行 SE.EXE 连线程序, 当 8051 程序执行后, 可在 PC 的显示器上看到如下的画面:

1 --> TEST 8255A	2 --> TEST 8255B	3 --> TEST A/D
4 --> TEST D/A	5 --> TEST LCD	6 --> TEST 3812
7 --> TEST UM3567	8 --> TEST 8253	

在 PC 上的按键功能如下:

- 按键 “1” : 测试第 1 块 8255 的功能。
- 按键 “2” : 测试第 2 块 8255 的功能。
- 按键 “3” : 测试 AD0804 功能, 由于输入电压正常时为 2.5V, 因此所读取到的数值应为 128 左右。
- 按键 “4” : 测试 DA08 的功能, 此时要以万用表测量 DA_OUT 接点, 分别送出 10V, 5V 及 0V 的模拟电压。
- 按键 “5” : 测试 LCD 模块是否正常。
- 按键 “6” : 测试 AD LIB 声效接口, 此时您若接有喇叭则会发出音阶测试音。
- 按键 “7” : 测试 UM3567 声效接口。
- 按键 “8” : 测试 8253 的计时功能。

程序清单

```

1  /*  TMIO51.C                      */
2  /*  MC51 test MIO card              */
3  /*  Copyright (C) VICTOR uP  LAB. 1996, 1997 */
4
5  #include "8051io.h" /* 载入 MC51 头文件 */
6  #include "8051reg.h"
7  #include "8051bit.h"
8  /* P51_PCB 控制 MIO 卡的 I/O 地址 */
9  #define CR (*(char*) 0xA303) /* 第 1 块 8255 */
10 #define PA (*(char*) 0xA300)
11 #define PB (*(char*) 0xA301)
12 #define PC (*(char*) 0xA302)
13
14 #define CR1 (*(char*) 0xA307) /* 第 2 块 8255 */
15 #define PA1 (*(char*) 0xA304)
16 #define PB1 (*(char*) 0xA305)
17 #define PC1 (*(char*) 0xA306)

```

```
18
19 #define IO (*(char*) 0xA310) /* A/D D/A 接口 */
20
21 #define CTR (*(char*) 0xA30b) /* 8253 接口 */
22 #define CT0 (*(char*) 0xA308)
23 #define CT1 (*(char*) 0xA309)
24 #define CT2 (*(char*) 0xA30a)
25
26 #define REG1 (*(char*) 0xA314) /* UM3567 接口 */
27 #define DATA1 (*(char*) 0xA315)
28
29 #define REG (*(char*) 0xA388) /* YM3812 接口 */
30 #define DATA (*(char*) 0xA389)
31
32 #define lcd_com (*(char*) 0xA30c) /* LCD 接口 */
33 #define lcd_data (*(char*) 0xA30d)
34
35 char *title= "MC51 test MIO card 86/7/23 ";
36
37 delay(int t) /* 延迟子程序 */
38 {
39     int i,j;
40     for(i=0; i<t; i++)
41         for(j=0; j<20; j++);
42 }
43 /*-----*/
44 led_blink() /* 工作 LED 闪动 */
45 {
46     /* blink RED led P1.7 */
47     char i;
48
49     for(i=0; i<3; i++)
50     {
51         clrbit(P1.7);
52         delay(30);
53         setbit(P1.7);
```

```

54     delay(30);
55 }
56 }
57 /*-----*/
58 main()
59 {
60     char c;
61
62     led_blink(); /* 工作 LED 闪动 */
63     serinit(9600); /* 初始化串行接口 */
64     printf(title); /* 显示工作消息 */
65     printf("\n");
66     menu(); /* 显示工作选单 */
67
68     init_lcd(); /* 设定 LCD 接口 */
69     print(1, "MC51 test MIO card ");
70     print(2, "By CHEN SI WEN .....");
71
72     printf("PC --- 8051 on line test.\n");
73     while(1) /* 循环 */
74     {
75         c=getch(); /* 等待由 PC 传来的控制键 */
76         if(c=='1') test_8255a();
77         if(c=='2') test_8255b();
78         if(c=='3') test_ad();
79         if(c=='4') test_da();
80         if(c=='5') test_lcd();
81         if(c=='6') test_3812();
82         if(c=='7') test_3567();
83         if(c=='8') test_8253();
84     }
85 }
86 /*-----*/
87 menu() /*显示工作选单*/
88 {
89     printf("1-->TEST 8255A 2--> TEST 8255B 3--> TEST A/D\n");

```

```
90 printf( "4-->TEST D/A 5 --> TEST LCD 6 --> TEST 3812\n");
91 printf( "7-->TEST UM3567 8 --> TEST 8253 ");
92 }
93 /*-----*/
94 test_8255a() /* 测试第 1 块 8255 的功能 */
95 {
96 char err;
97 int i;
98 unsigned char in;
99
100 CR=0x80; /* 设定工作模式 */
101 err=0; /* 清除错误标志*/
102 printf( " TEST 8255 (a) 3 PORTS...\n");
103 printf( "TEST PORT A ");
104 for(i=0; i<256; i++)
105 {
106 PA=i; in=PA; /* 写入测试样本在读出比较 */
107 if( in !=i) { err=1; goto error; }
108 }
109
110 printf( "TEST PORT B ");
111 for(i=0; i<256; i++)
112 {
113 PB=i; in=PB;
114 if( in !=i) { err=1; goto error; }
115 }
116
117 printf( "TEST PORT C ");
118 for(i=0; i<256; i++)
119 {
120 PC=i; in=PC;
121 if( in !=i) { err=1; goto error; }
122 }
123
124 error: /*显示测试结果 */
125 if(err) printf( "8255(a) test error !\n");
```



```
126     else printf( "8255(a) test OK !\n");
127
128     if(err) err_show();
129     else led_blink();
130 }
131 /*-----*/
132 test_ad() /* 测试 AD0804 功能 */
133 {
134     unsigned char c, i;
135     /* 所读取到的数值应为 128 左右 */
136     printf( " TEST ADC0804 ..... \n");
137     printf( " DATA should be 127 128 \n");
138     for(i=0; i<20; i++)
139     {
140         c=IO;
141         printf( "%d ", c);
142         delay(100);
143     }
144 }
145 /*-----*/
146 test_da()
147 { /* 送出 10V 的模拟电压 */
148     printf( " TEST DAC0800 --> send out 10 V \n");
149     IO=255;
150     getch(); /* 等待 PC 是否按键 */
151     /* 送出 5V 的模拟电压 */
152     printf( " TEST DAC0800 --> send out 5 V \n");
153     IO=127;
154     getch();
155     /* 送出 0V 的模拟电压 */
156     printf( " TEST DAC0800 --> send out 0V \n");
157     IO=0;
158 }
159 /*-----*/
160 /* LCD 控制接口 */
161 /*-----*/
```

```
162 char mess[30]; /* LCD 字符串输出缓冲区 */
163 write_com(unsigned char c) /* 写命令至 LCD */
164 {
165     unsigned char in;
166
167     while(1) /* 循环 */
168     { in=lcd_com; /* 读取状态端口 */
169       if( (in & 0x80)==0 ) break; }
170     lcd_com= c; /* 写命令至 LCD */
171 }
172 /*-----*/
173 write_data(unsigned char d) /* 写数据至 LCD */
174 {
175     unsigned char in;
176     while(1) /* 循环 */
177     { in=lcd_com; /* 读取状态端口 */
178       if( (in & 0x80)==0 ) break; }
179     lcd_data= d; /* 写数据至 LCD */
180 }
181 /*-----*/
182 init_lcd() /* 初始化 LCD */
183 {
184     write_com(0x3c); /* 双列显示, 字型使用 5×10 点阵 */
185     write_com(0x0e); /* 光标出现, 不闪烁 */
186     write_com(0x06); /* 每次向右移一位, 显示屏不移动 */
187     write_com(0x01); /* 清除 LCD 显示屏 */
188     delay(100);
189 }
190 /*-----*/
191 print(char line, char *str)
192 {
193     char i;
194
195     if(line==1)
196     { write_com(0x80); /* 移至第 1 行首 */
197       for(i=0; i<24; i++) write_data(' '); /* 清除该行文字 */
```

```

198     write_com(0x80); /* 移至第 1 行首 */ }
199     else
200     {     write_com(0xc0); /* 移至第 2 行首 */
201         for(i=0; i<24; i++) write_data(' '); /* 清除该行文字 */
202         write_com(0xc0); /* 移至第 2 行首 */ }
203
204     i=0; /* 将字符串数据写至 LCD */
205     do{ write_data(*str++); }
206     while(*str!='\0');
207 }
208 /*-----*/
209 test_lcd() /* 测试 LCD 显示功能 */
210 {
211     char i;
212
213     init_lcd(); /* 初始化 LCD */
214
215     delay(300); /* 由第 1 行输出文字 */
216     sprintf(mess, "8051 PCB LCD TEST >>>>>>>>>>>>");
217     print(1, mess);
218
219     write_com(0xc0); /* 移至第 2 行首 */
220     /* 写入数字 0-9 */
221     for(i=0; i<10; i++) write_data(i+0x30);
222     delay(300); /* 延迟一下 */
223     led_blink(); /* 工作 LED 闪动 */
224     sprintf(mess, "TEST OVER >>>>>>>>>>>>");
225     print(2, mess); /* 由第 2 行输出文字 */
226 }
227
228 /*-----*/
229 /* UM3567 控制接口 */
230 /*-----*/
231 /* PSG I/O 控制 */
232 cpl(char reg, unsigned char data)
233 {

```

```
234  REG1=reg;
235  delay(2);
236  DATA1=data;
237  delay(2);
238  }
239  /*-----*/
240  sound_off() /* 关闭声音接口 */
241  {
242  int i;
243  for(i=0; i<9; i++)
244  {
245  cpl(0x10+i, 0);
246  cpl(0x20+i, 0);
247  }
248  }
249  /*-----*/
250  test1() /* 测试单音 */
251  {
252  printf( "test1 .....");
253  cpl(0x30, 0x10); /* violin , max volumn */
254  cpl(0x10, 0xcc); /* tone "DO" */
255  cpl(0x20, 0x1a); /* sound */
256  delay(300);
257  sound_off();
258  }
259  /*-----*/
260  test_3567() /* 测试 PSG 16 种乐器声音 */
261  {
262  int c;
263
264  printf( "Test UM3567 .....");
265  printf("INS : ");
266  for(c=1; c<16; c++)
267  {
268  printf("%d ", c);
269  cpl(0x30, c*16);
```

```
270     cpl(0x10, 0xab);
271     cpl(0x20, 0x1a);
272     delay(250);
273     sound_off();
274 }
275
276 sound_off();
277 }
278 /***/
279 /*  YM3812  控制接口 */
280 /***/
281 /*  PSG I/O  控制 */
282 cp(char reg, unsigned char  data)
283 {
284     REG=reg;
285     delay(10);
286     DATA=data;
287     delay(10);
288 }
289 /*-----*/
290 int notes[12] = {0x16B, /*  0:  Do#  */
291                 0x181, /*  1:  Re   */
292                 0x198, /*  2:  Re#  */
293                 0x1B0, /*  3:  Mi   */
294                 0x1CA, /*  4:  Fa   */
295                 0x1E5, /*  5:  Fa#  */
296                 0x202, /*  6:  So   */
297                 0x220, /*  7:  So#  */
298                 0x241, /*  8:  La   */
299                 0x263, /*  9:  La#  */
300                 0x287, /* 10:  Si   */
301                 0x2AE  /* 11:  Do   */
302             };
303
304 char piano[] = {
305     0x31, 0x21, 0x8a, 0x40,
```

```
306         0xf1, 0xf1, 0x4c, 0xdd,
307         0x00, 0x00, 0x06
308     };
309
310     char bass[] = {
311         0x17, 0xff, 0x00, 0xff,
312         0xc2, 0xaf, 0x83, 0x9f,
313         0x00, 0x00, 0xf4
314     };
315
316     char bell[] = {
317         0x07, 0x12, 0x4f, 0x00,
318         0xf2, 0xf2, 0x60, 0x72,
319         0x00, 0x00, 0x08
320     };
321     /*-----*/
322     reset_psg() /* RESET PSG */
323     {
324         int i;
325         for (i=1; i<=245; ++i)
326             write_register(i, 0);
327     }
328     /*-----*/
329     write_register(int register_no, int data)
330     { /* PSG I/O 控制 */
331         char i, in;
332
333         REG = register_no;
334         for (i=0; i<6; ++i) in = REG;
335
336         DATA = data;
337         for (i=0; i<35; ++i) in = REG;
338     }
339     /*-----*/
340     /* int channel_no : 0~8 */
341     key_off(int channel_no) /* 音阶关闭 */
```

```
342 {
343   int register_no;
344   register_no = 0xB0 + channel_no;
345   write_register(register_no,0);
346 }
347 /*-----*/
348 /*   int channel_no : 0~8
349   int octave      : degree
350   int note_index  : 0~11 */
351 /*   音阶输出 */
352 key_on(int channel_no, int octave, int note_index)
353 {
354   int register_no;
355   int data,note_lsb,note_msb,note_on;
356
357   register_no = 0xA0 + channel_no;
358   note_lsb=notes[note_index] & 0xFF;
359   note_msb=notes[note_index] >> 8;
360   write_register(register_no,note_lsb);
361
362   register_no = 0xB0 + channel_no;
363   note_on = 0x20; /* BIT5 = 1 */
364   octave  = octave << 2; /* move BIT4 to BIT2 */
365   data = note_on | octave | note_msb;
366   write_register(register_no,data);
367 }
368 /*-----*/
369 /*   int channel_no : 0~8*/
370 /*int vol      : 0~63 */
371 /*   设定音量 */
372 set_volume(int channel_no,int vol)
373 {
374   int register_no;
375   register_no = 0x40 + channel_no;
376   write_register(register_no,vol);
377 }
```

```
378 /*-----*/
379 int no[7]={11,1,3,4,6,8,10};
380 test() /* 测试单音 */
381 {
382
383     int j;
384     int octave;
385
386     for (octave=2; octave<6; ++octave)
387     {
388         key_on(0,octave-1,no[0]);
389         delay(50);
390         key_off(0);
391
392     for (j=1;j<7;++j)
393     {
394         key_on(0,octave,no[j]);
395         delay(70);
396         key_off(0);
397         printf(".");
398     }
399 }
400 }
401 /*-----*/
402 set_instrument(int    channel_no, char *ins) /* 设定乐器种类 */
403 {
404     unsigned char modulator_cell,carrier_cell;
405     int offset;
406
407     offset    = channel_no % 3 + (channel_no / 3) * 8;
408
409     modulator_cell = 0x20 + offset;
410     write_register(modulator_cell,ins[0]);
411     carrier_cell =    modulator_cell + 3;
412     write_register(carrier_cell,ins[1]);
413 }
```



```
414 modulator_cell = 0x40 + offset;
415 write_register(modulator_cell,ins[2]);
416 carrier_cell = modulator_cell + 3;
417 write_register(carrier_cell,ins[3]);
418
419 modulator_cell = 0x60 + offset;
420 write_register(modulator_cell,ins[4]);
421 carrier_cell = modulator_cell + 3;
422 write_register(carrier_cell,ins[5]);
423
424 modulator_cell = 0x80 + offset;
425 write_register(modulator_cell,ins[6]);
426 carrier_cell = modulator_cell + 3;
427 write_register(carrier_cell,ins[7]);
428
429 modulator_cell = 0xE0 + offset;
430 write_register(modulator_cell,ins[8]);
431 carrier_cell = modulator_cell + 3;
432 write_register(carrier_cell,ins[9]);
433
434 modulator_cell = 0xC0 + channel_no;
435 write_register(modulator_cell,ins[10]);
436 }
437 /*-----*/
438 test_3812() /* 测试 PSG */
439 {
440     printf("Test YM3812 ");
441     reset_psg();
442     set_instrument(0, piano);
443     set_instrument(1, bass);
444     set_instrument(2, bell);
445
446     set_volume(1,1);
447     key_on(0,4,6); /* ch1 Do */
448     key_on(1,4,3); /* ch2 Mi */
449     key_on(2,4,6); /* ch3 So */
```

```
450 delay(500);
451
452 key_off(0);
453 key_off(1);
454 key_off(2);
455
456 test();
457 reset_psg();
458 }
459 /*-----*/
460 /*****/
461 /* 8253 控制接口 */
462 /*****/
463 /* use mode3 gate=1, count always */
464 /* 测试 8253 的计时功能 */
465
466 test_8253()
467 {
468 unsigned char c, c1, cx;
469 int i;
470 char err;
471
472 err=0;
473 printf("Test 8253    ....\n");
474 printf("Pin out0 should have 1KHZ    pulse output ..\n");
475
476 CTR=0x36; /* channel 0/ mode2 */
477 CT0=2000%256; /* load low  byte */
478 CT0=2000/256; /* load high byte */
479
480 printf("Wait .....");
481 for(i=0; i<256; i++)
482 {
483 CTR=0x06; /* latch counter */
484 c=CT0; /* read low byte */
485 /* inportb(CT0); */ /* read high byte */
```

```
486 cx=CT0;
487 delay(1);
488
489 CTR=0x06; /* latch counter */
490 cl=CT0; /* read low byte */
491 cx=CT0; /* read high byte */
492
493 /* printf("%02x %02x ", c, cl); */
494 if( c==cl) { err=1; break; }
495 }
496
497 if(err) printf("\n8253 test error!");
498 else printf("\n8253 OK!");
499
500 if(err) err_show();
501 else led_blink();
502 }
503 /*-----*/
504 test_8255b() /* 测试第 2 块 8255 的功能 */
505 {
506 char err;
507 int i;
508 unsigned char in;
509
510 CR1=0x80;
511 err=0;
512 printf(" TEST 8255 (b) 3 PORTS...\n");
513 printf("TEST PORT A ");
514 for(i=0; i<256; i++)
515 {
516 PA1=i; in=PA1;
517 if( in!=i) { err=1; goto error1; }
518 }
519
520 printf("TEST PORT B ");
521 for(i=0; i<256; i++)
```

```
522     {
523         PB1=i;in=PB1;
524         if( in!=i) { err=1; goto error1; }
525     }
526
527     printf("TEST PORT C ");
528     for(i=0; i<256; i++)
529     {
530         PC1=i;in=PC1;
531         if( in!=i) { err=1; goto error1; }
532     }
533
534 error1:
535     if(err) printf(" 8255(b) test error !\n");
536     else printf(" 8255(b) test OK!\n");
537
538     if(err) err_show();
539     else led_blink();
540 }
541 /*-----*/
542 err_show() /* 显示错误结果 */
543 {
544     while(1) led_blink();
545 }
```

第 16 章 8051 无线遥控接口

市面上常见的遥控器有红外线及无线电遥控,红外线遥控有其空间的限制,一般用在家电产品控制上,而无线电的遥控距离可以较远,扩充也比较有弹性,只是有关无线电技术涉及到的相关知识很多,往往一般人不得其门而入,如果有现成的模块可以加以利用,再结合 8051 单片机的控制功能,便可以设计出特殊的电子装置,其附加价值相当高。在本章中我们将介绍一组现成的遥控模块(RF51),并说明如何利用 8051 来控制这套模块,当读者了解其基本工作原理后便可应用它来附加在自行设计的专题制作中,使自己的产品多了一项遥控的功能。

此外也将介绍如何以 8051 多功能控制板(P51_PCB)来控制 RF51,可以从单板上发射和接收数据,除了做基本的实验外,也可以做各种专题设计的组合。

16.1 遥控模块特性说明

无线遥控模块是由一组发射器和接收机组成,系统主要特性如下:

1. 具有 UHF 发射电路,可以做无线电传输及控制的相关实验。
2. 有编解码 IC,控制不受外界信号干扰。
3. 遥控器上有密码设定装置,由 DIP 开关调整。
4. 内含看门狗电路,不受死机困扰。
5. 内含一组 7 段数码管,可当数据显示用。
6. 具有串行传输接口,可以做系统通信用。
7. 有 4 组继电器,可以直接控制家电。
8. 由 8051 直接控制,并可外接 EPROM,使用 ROM 模拟器便可以自行设计及测试程序。
9. 可以连到 PC 上做无线电遥控设计,增加 PC 的控制能力。
10. 可做电脑无线数据传送。

基本上,读者手上的任何电子产品想加装遥控的功能,均可利用此套模块进行改装,因此在未来的应用上可以分为以下几种,一系列的相关应用在陆续开发之中。读者可以利用此套无线电遥控模块,加入个人专题制作中,增加额外的一项无线电遥控的功能。有关此套无线电遥控模块的应用实例可以参考“PC/8051 无线遥控专题制作”一书。

16.2 遥控模块系统组成

图 16-1 是整个系统的组成，分为 2 部分：

1. 发射器；
2. 接收机。

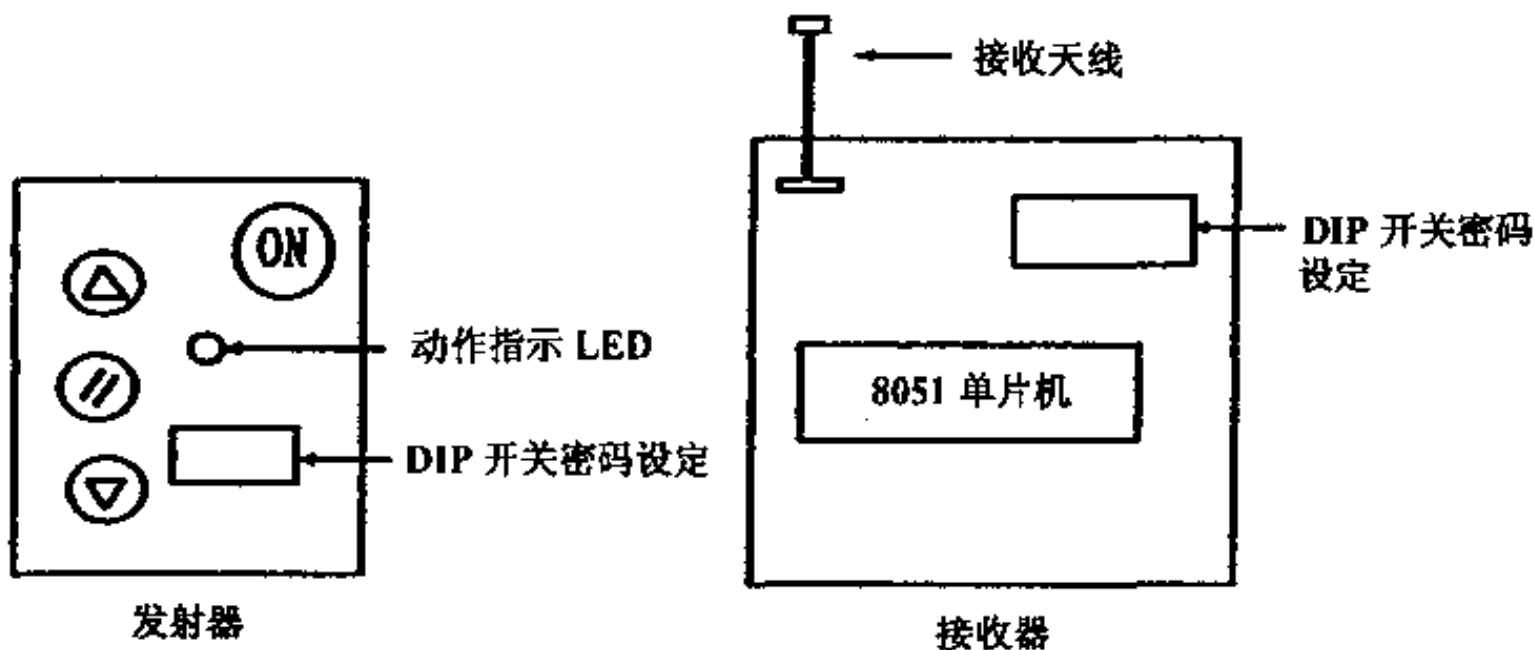


图 16-1 无线电遥控模块组成

发射器用电池供电，体积小可以随身携带，而接收机由市电供电，在工作时二者密码必需一致才能工作，一部接收器可以根据需要配备多部发射器。

16.2.1 发射器

图 16-2 是发射器的工作原理图。发射器本身由小型的 12V 电池供电，内含 8 组 DIP 开关，可以提供有 256 组密码设定，其上有 4 组按键，当按下任何一键时工作指示 LED 灯亮，则发送不同的数据出去，由接收机接收而做相对的工作，发射器本身平时并不需耗电，当有任何按键按下时才会耗电，因此不需装上电源开关。由于无线电遥控是靠高频的载波频率传送数字代码数据，因此在出厂时，其工作频率已先行设定调整完成，使用者请勿自行调整板子上的小型可变电容器，否则工作频率不吻合时，则无法正常的接收。在空旷的室外其遥控的距离可达几十米远。

16.2.2 接收机

图 16-3 是接收器的工作原理图，接收机的结构较复杂，由以下几部分组成。

- 高频电路

接收来自发射器送来的信号，并加以放大而送至解码 IC。

- 解码 IC HT12D

负责将代码的数据进行解码，取出原先发射的数据而送到 8051 单片机上，密码的设

定可由 DIP 开关加以调整，必需与发射机上的一致方可工作。

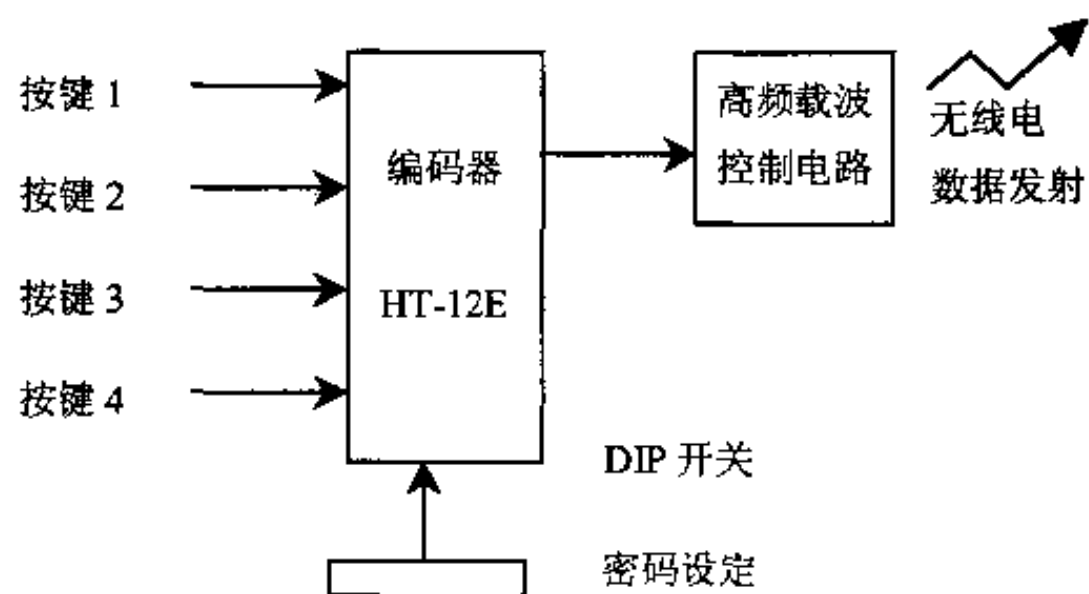


图 16-2 发射器工作原理图

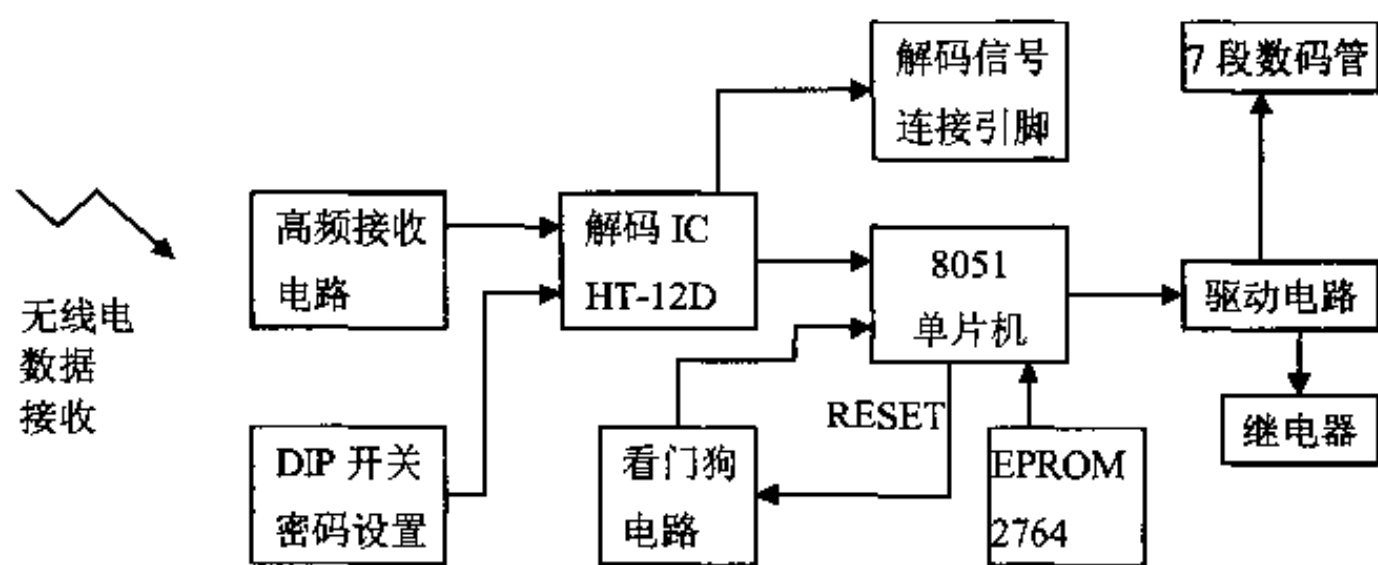


图 16-3 接收器工作原理图

• 8051 单片机控制

控制程序可以经过 ROM 模拟器来做测试，为了工作于环境较差，干扰信号较大的场所，于是另外加装“看门狗”电路，当系统受信号干扰以致于死机时，可以自动 RESET 8051 再次重新执行程序。

• 驱动电路

包括驱动 4 只继电器工作的晶体管及 7 段数码管的解码 IC。

16.3 编解码 IC HT12 简介

一般在遥控器上均设有密码的功能，因此可以有多种的组合设定，以避免代码的重复而造成使用上的互相干扰。本节将介绍无线遥控模块(RF51)内编解码 IC 的工作原理。此一模块内使用 HT-12 系列编解码器，市面上应用的场合相当多，在发射器上是用 HT-12E 编码器，而在接收模块内则是以 HT-12D 作解码器。

16.3.1 HT-12 编解码器特性介绍

1. 应用 CMOS 技术制造具有省电、耐干扰的优点。
2. 工作电压范围 2V~13V, 可用电池供电。
3. 内含振荡电路, 只需外加一只电阻即可提供工作频率。
4. 容易与一般遥控接口搭配, 如无线电、红外线、超音波等。
5. HT12E 编码器(发射器)可以有 4096(2 的 12 次方)组密码, 并可传送 4 个位的数据。
6. 接收器(解码器)具有两种方式:
 - (1) HT-12D: 4 位数据输出, 8 位密码设定。
 - (2) HT-12F: 无数据输出型, 具有 12 位的密码设定。
7. 解码器输出数据具有锁存的功能。

16.3.2 引脚说明

图 16-4 为 HT12 系列的引脚图, IC 为 18 引脚 DIP 封装。

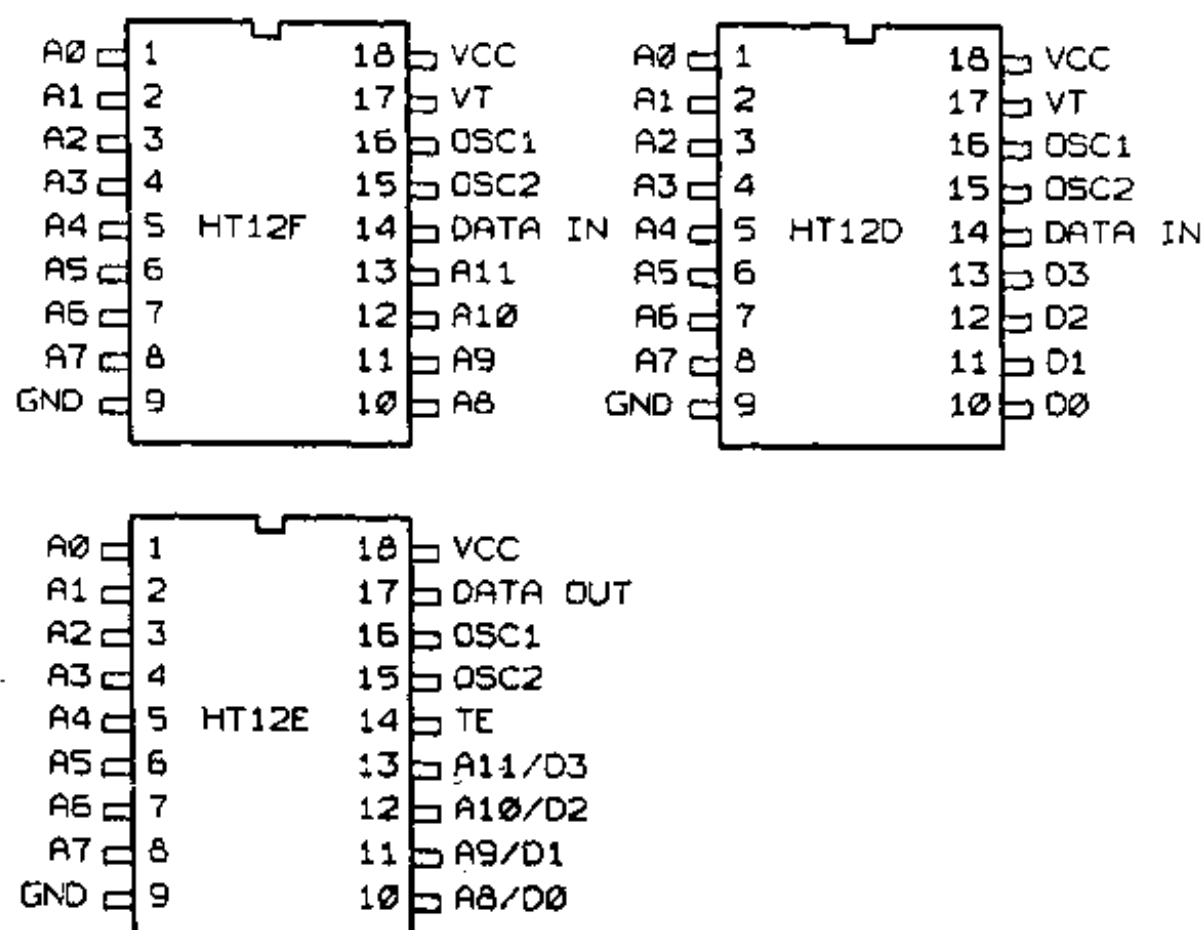


图 16-4 HT12 系列 IC 引脚

- A0~A11:
密码的位设定, 共有 4096 种组合。
- HT12E D0~D3:
数据输入位。

- HT12D D0~D3:

数据输出位。如 HT12E 与 HT12D 搭配使用, 当 A0~A7 密码一致时, HT12E 所传送的 4 位数据会出现在 HT12D 的 D0~D3 引脚上。

- DATA OUT :

数据发送端。

- DATA IN :

数据接收端。

- TE\:

允许发射信号, 低电位工作, 此时所编码的信号由 DATA OUT 引脚送出。

- VT:

当发射端与接收端密码一致, 接收端接收进来的数据解码完成, 数据出现在 D0~D3 引脚时, 此引脚会出现高电位信号。

- OSC1, OSC2:

振荡电路控制引脚, 只要加入一只电阻即可工作。需注意的是解码器的振荡工作频率约为编码器的 50 倍, 所选择的电阻如下表所示:

HT12E		HT12D/F	
振荡电阻	振荡频率	振荡电阻	振荡频率
1.5 M	3 KHz	75 K	150 KHz
1 M	4.3 KHz	47 K	240 KHz

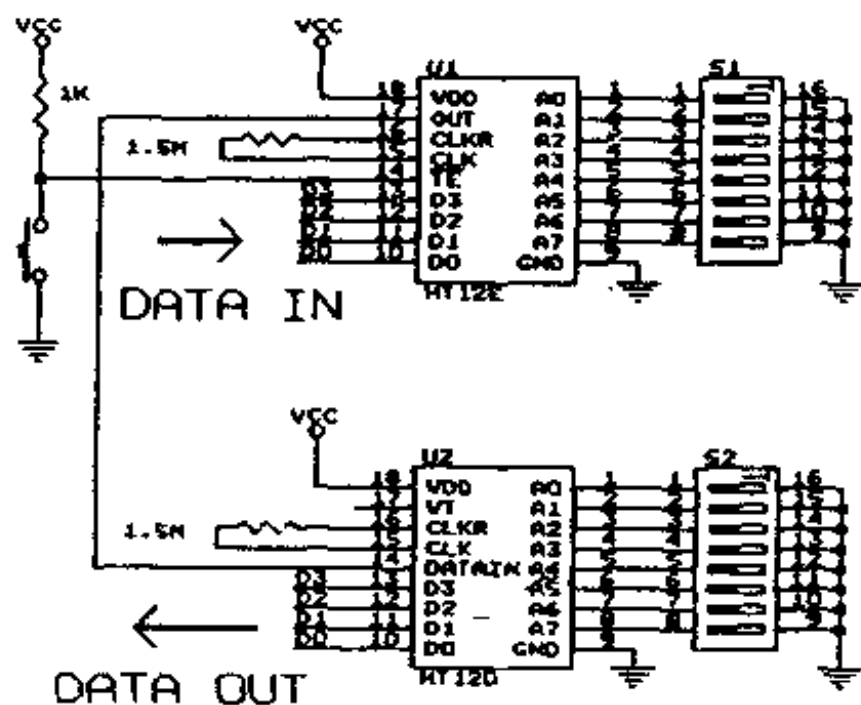


图 16-5 HT12 编解码器的基本工作电路

图 16-5 为 HT12 编解码器的基本工作电路, 由 DIP 开关设定 8 位的密码值, 当两边

J3 引脚插座的控制 8051 程序代码由外部 EPROM 提供, 一般 EA 接地, 由 U2 2764 提供程序代码, 若使用 8751 时则 EA 接 +5V 电压。TXD、RXD 引脚由 J5 引脚插座拉出来可以提供多块 8051 做串行端口的连线控制用, 若与 PC 连线 (RS232 接口), 则需外加 MAX232 等信号电位转换 IC。

高频模块 (RF Module) 提供经过高频接收机电路接收进来的信号及解码器转换出来的数据 (D0~D3), 此时也送出数据使能信号 EN (高电位工作) 及 EN \bar (低电位工作), 其中 EN \bar 接至 8051 INT0 引脚。DATA_OUT 则为高频接收电路送出的工作信号, 可以做进一步的信号分析用。以上的相关控制信号均连至扩充引脚插座 J4 (16PIN) 插针, 可以连至 PC 上做额外的功能扩充用。

U4 74LS48 负责共阴极 7 段数码管的解码驱动, 用来做一般数据的显示用。另外由 8051 控制的 4 组继电器, 其驱动电路如图 16-7 所示, 使用者可以经过继电器直接做一般家电遥控。

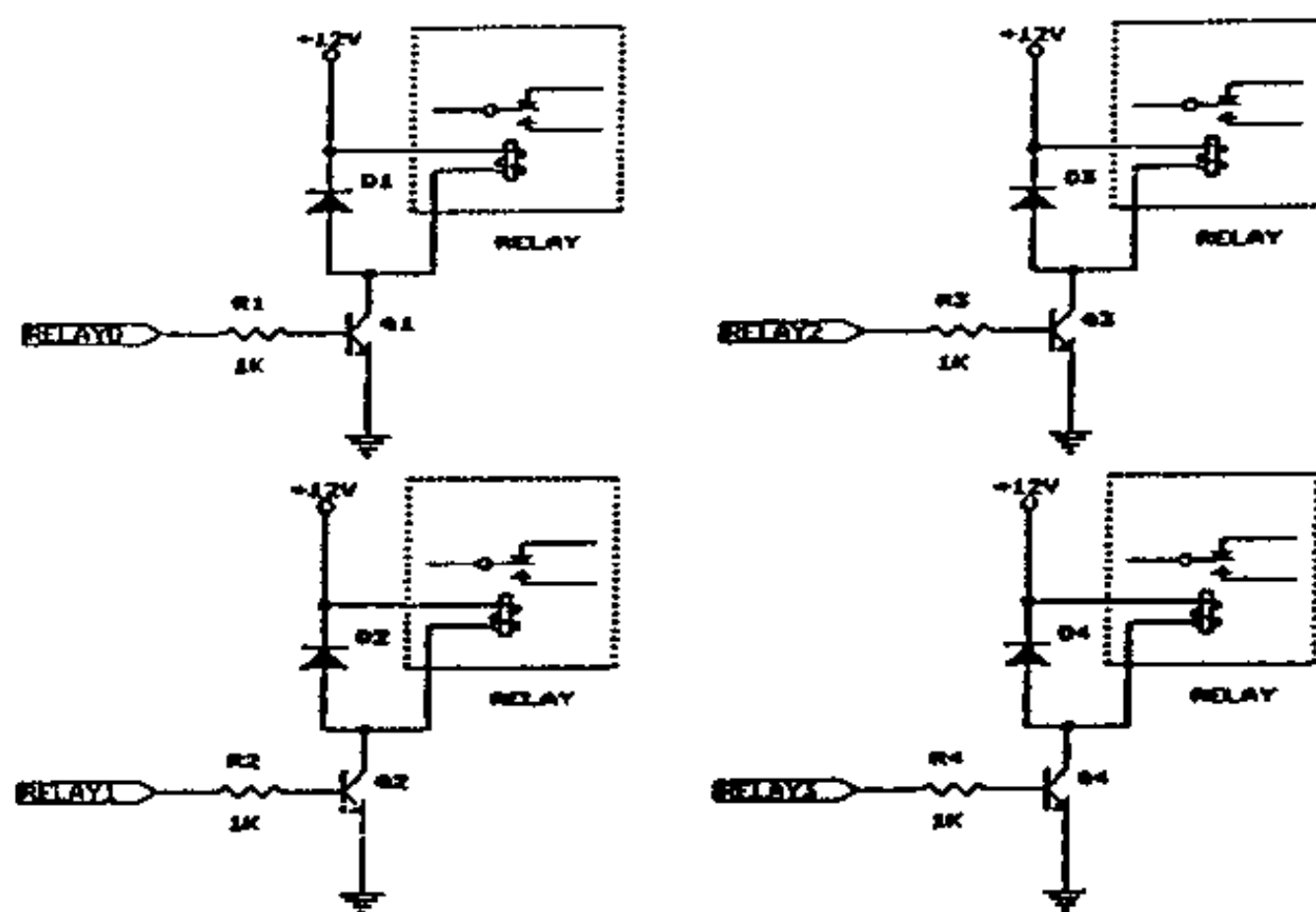


图 16-7 继电器的驱动电路

16.4.1 控制信号分析

1. 无线遥控数据为 4 个位, D0~D3 到信号 EN \bar , 连接 8051 的位引脚分配如下:

信号	D3	D2	D1	D0	EN \bar
控制位	P1.5	P1.4	P1.6	P1.7	P3.2(INT0)

2. 继电器驱动位分配如下:

继电器	R3	R2	R1	R0
引脚	P1.3	P1.2	P1.1	P1.0

当该位送出高电位可以使继电器工作。

3. 驱动 7 段数码管:

控制位	B3	B2	B1	B0
引脚	P3.5(T1)	P3.4(T0)	P3.3(INT1)	P3.7(RD)

数字显示控制方式:

控制位	B3	B2	B1	B0
数字 0	0	0	0	0
数字 1	0	0	0	1
数字 2	0	0	1	0
数字 3	0	0	1	1
数字 4	0	1	0	0
数字 5	0	1	0	1
数字 6	0	1	1	0
数字 7	0	1	1	1
数字 8	1	0	0	0
数字 9	1	0	0	1

16.4.2 引脚使用功能

在 8051 遥控模块中, 共使用 7 组引脚插座, 其用途如下:

J1: 直流 12V 输入, 电路板上具有稳压电路。可以使用市售的 12V 电源调整器经过转接线输入, 极性不分。

J2: 启用看门狗电路, 接通时可将 RESET 信号送往 8051。

J3: 外部 EPROM 读取使能控制。

J4: 16PIN 功能扩充引脚插座。

J5: 串行端口输入输出引脚插座。

J6: 提供+8V 及+5V 稳压直流电压输出。

J7: RESET 输入引脚, 可由 ROM 模拟器送入 RESET 信号。

注意事项

1. 接收器及发射器上的可变电容请勿调整, 以免频率偏移。
2. DIP 开关用于密码设定, 接收器及发射器上需设定一致。
3. 接收器及发射器是配对的, 工作时频率必须一样, 密码一致。

16.5 8051 接收模块测试程序

本节介绍如何用 MICRO.C51 C 语言来设计无线遥控模块 RF51 接收控制程序。由于 RF51 上 8051 并没有扩充额外的外部内存，而在前面所使用的 MC51 内存操作模式为中型模式(使用外部内存)不适用，必需修改为极小型模式来操作，以综合的批处理文件 T.BAT 进行编译。使用方法为：

```
t test <ENTER>
```

便可以编译 test.c 程序

T.BAT 内容

```
@echo off
copy c: %1.c t.c
echo MC51 comp....
cc51 t.c m=t -ipq
IF ERRORLEVEL 1 GOTO error
hexbin t.hex t.tsk i l
emu t.tsk 5 d
se
GOTO end
: error
ECHO MC51 comp. error !!!
: end
@echo on
```

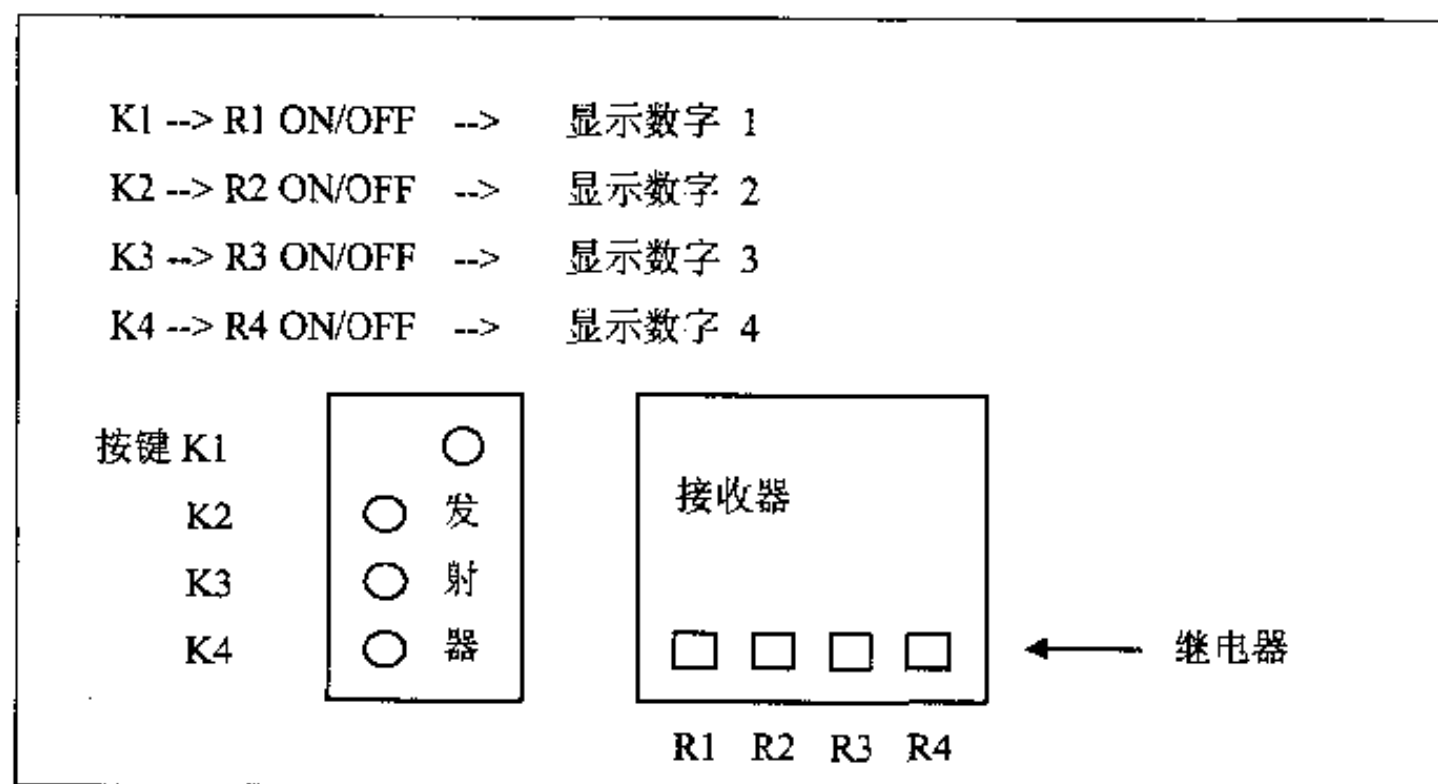
而发射器的按键与接收到的数据(D0~D3)列表如下：

8051 引脚	P1.4	P1.5	P1.6	P1.7
按键	D3	D2	D1	D0
K1	1	1	1	0
K2	0	1	1	1
K3	1	0	1	1
K4	1	1	0	1

当由发射器按下某按键(如 K1)时，该组按键数据(1110)会出现在 8051 的相对引脚上，同时 EN 信号(接至 8051 P3.2 INT0)会降为低电位以通知 8051 目前解码数据有效，可以读取其数据了。

执行结果

测试程序文件名为 RTEST.C, 工作示意图如下:



首先测试 7 段数码管的工作, 7 段数码管依序显示数字 0 到 9。然后测试无线遥控模块接收的状态, 当按下 K1 键然后放开时, 相对的接收机继电器 R1 则导通 (ON), 随后自动切断 (OFF), 若按下 K2 键, 则可以控制 R2 的工作, 余此类推。

程序清单

```

1  /*  RTEST.C  */
2  /*  RF51 MODULE test :  MC51  test BASIC I/O */
3  /*  use T.BAT to compiler in tiny mode    */
4
5  #include "8051io.h"
6  #include "8051reg.h"
7  #include "8051bit.h"
8  #include "8051int.h"
9
10 char *title="C8051  test RF51 :  BASIC I/O";
11 /*  延迟子程序 */
12 delay(int t)
13 {
14     int i,j;
15     for(i=0; i<t; i++)
16         for(j=0; j<10; j++);

```

```
17 }
18 /*-----*/
19 main()
20 {
21     int i;
22     /* 令 4 只继电器 OFF */
23     clrbit(P1.0);
24     clrbit(P1.1);
25     clrbit(P1.2);
26     clrbit(P1.3);
27
28     for(i=0; i<10; i++)
29     { /* 7 段数码管显示 0~9 */
30         show_seg(i);
31         delay(500);
32     }
33     show_seg(0);
34     receive(); /* 执行接收测试程序 */
35 }
36 /*-----*/
37 /* 8051 P3.5 P3.4 P3.3 P3.7 */
38 /* 74LS48 d b c a */
39 /*-----*/
40 show_seg(int no) /* 7 段数码管显示子程序 */
41 {
42     switch(no)
43     {
44     case 0 : clrbit(P3.5); clrbit(P3.4);
45             clrbit(P3.3); clrbit(P3.7); break;
46     case 1 : clrbit(P3.5); clrbit(P3.4);
47             clrbit(P3.3); setbit(P3.7); break;
48     case 2 : clrbit(P3.5); clrbit(P3.4);
49             setbit(P3.3); clrbit(P3.7); break;
50     case 3 : clrbit(P3.5); clrbit(P3.4);
51             setbit(P3.3); setbit(P3.7); break;
52     case 4 : clrbit(P3.5); setbit(P3.4);
```

```
53      clrbit(P3.3);  clrbit(P3.7); break;
54
55 case 5    :  clrbit(P3.5);  setbit(P3.4);
56      clrbit(P3.3);  setbit(P3.7); break;
57 case 6    :  clrbit(P3.5);  setbit(P3.4);
58      setbit(P3.3);  clrbit(P3.7); break;
59 case 7    :  clrbit(P3.5);  setbit(P3.4);
60      setbit(P3.3);  setbit(P3.7); break;
61 case 8    :  setbit(P3.5);  clrbit(P3.4);
62      clrbit(P3.3);  clrbit(P3.7); break;
63 case 9    :  setbit(P3.5);  clrbit(P3.4);
64      clrbit(P3.3);  setbit(P3.7); break;
65 default : break;
66 }
67 }
68 /*-----*/
69 op_relay(unsigned char in) /* 继电器控制子程序 */
70 {
71     if( (in & 0x80)==0) /* 按键 K1 按下 */
72     {
73         show_seg(1);    /* 7 段数码管显示 1 */
74         setbit(P1.3);    /* 继电器 ON */
75         delay(500);     /* 延迟一段时间 */
76         clrbit(P1.3);    /* 继电器 OFF */
77         return;
78     }
79
80     if( (in & 0x20)==0) /* 按键 K3 按下 */
81     {
82         show_seg(3);    /* 7 段数码管显示 3 */
83         setbit(P1.1);    /* 继电器 ON */
84         delay(500);     /* 延迟一段时间 */
85         clrbit(P1.1);    /* 继电器 OFF */
86         return;
87     }
88 }
```



```
89   if( (in & 0x10)==0) /* 按键 K2 按下*/
90   {
91       show_seg(2);    /* 7 段数码管显示 2 */
92       setbit(P1.2);    /* 继电器 ON */
93       delay(500);      /* 延迟一段时间*/
94       clrbit(P1.2);    /* 继电器 OFF */
95       return;
96   }
97
98   if( (in & 0x40)==0) /* 按键 K4 按下*/
99   {
100      show_seg(4);     /* 7 段数码管显示 4 */
101      setbit(P1.0);     /* 继电器 ON */
102      delay(500);      /* 延迟一段时间*/
103      clrbit(P1.0);     /* 继电器 OFF */
104      return;
105   }
106 }
107 /*-----*/
108 receive() /* 接收测试程序 */
109 {
110     unsigned char in;
111
112     /* P3.2 设为高电位，一旦接收信号准备好后会变为低电位*/
113     setbit(P3.2);
114
115     /* 查询 P3.2 引脚是否变为低电位 */
116     /* 若 P3.2 引脚变为低电位，表示发射器有按键，则接收信号 */
117     while(1)
118     if( (P3 & 0x04)==0)
119     {
120         in=P1;
121         /* 等待发射器的按键是否放开来 */
122         while(1) if( (P3 & 0x04)!=0x04) break;
123         op_relay(in);
124     }
```

```
125 }
126 /*-----*/
```

16.6 8051 多功能控制板无线遥控接口

本节将介绍如何将 8051 无线遥控模块(RF51)连至 8051 多功能控制板(P51_PCB)，使得单板上可以增加无线电遥控接口的能力，包括由 P51 接收无线电遥控的信号，及发射无线电的信号出去，我们来看看硬件和程序是如何来设计的。

16.6.1 P51 接收无线电遥控的信号

图 16-8 是 P51 接收无线电遥控信号的电路图，以一 16 PIN 排线将无线电遥控模块(RF51)的 J4 连至 P51 J28 16 PIN 插针上，RF51 的 J4 引脚插座将输出所接收到的无线电遥控数字信号，而 P51 J28 引脚插座则连至 8255 端口 A，并经过程序规划为输入端，当无线电发射器上有按键时，则会送出相对的数字信号控制代码，经过 8255 端口 A 由 8051 读取。其中发射器上按键及 8255 接收信号对应如下：

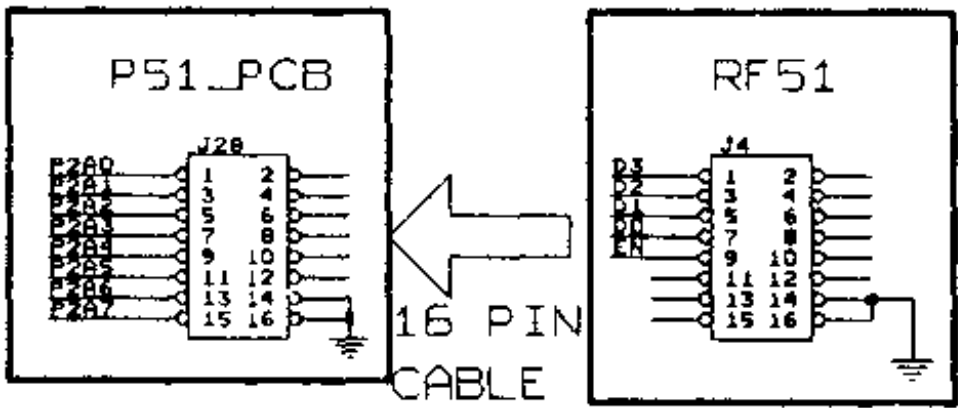


图 16-8 P51 接收无线电遥控信号的电路图

8255 PA	PA0	PA1	PA2	PA3
按键	D3	D2	D1	D0
K1	1	1	1	0
K2	0	1	1	1
K3	1	0	1	1
K4	1	1	0	1

当由发射器按下某按键（如 K1）时，该组按键数据（1110）会出现在 8255 的相对引脚上，同时 EN 信号（接至 8255 PA4 引脚）会成为高电位以通知 8051 目前解码数据有效，可以读取其数据了。

实验注意事项

- 1. RF51 J1 接头必需接入 12V 电源调整器。

2. 以 16 PIN 排线连接 RF51 J4 接头及 P51 J28 接头。
3. 将 RF51 上的 8051 先取下来, 避免造成负载效应而影响实验结果。

执行结果

发射器上有 4 个按键 K1~K4, 当按下某一按键并放开时, 单板会显示出您按下了那一按键, 同时蜂鸣器发出“嘀”声, 工作 LED 灯会闪动。例如按下 K3 键, 则会显示出您按下了 K3 键, 蜂鸣器“嘀”3 声, 工作 LED 灯会闪动 3 下。

程序清单

```

1  /*  PRX.C  */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h"      /* 载入 P51 I/O 控制头文件 */
7
8  char *title="P51_PCB RX RF51 MODULE 4 keys\n";
9  /*-----*/
10 delay(int t) /* 延迟子程序 */
11 {
12     int i,j;
13     for(i=0; i<t; i++)
14         for(j=0; j<10; j++);
15 }
16 /*-----*/
17 led_bl() /* 工作 LED 闪动 */
18 {
19     char i;
20     for(i=0; i<4; i++)
21     {
22         cplbit(P1.7);
23         delay(40);
24     }
25 }
26 /*-----*/
27 be() /* 蜂鸣器“嘀”一声 */
28 {

```

```
29 char i;
30
31 for(i=0; i<100; i++)
32 {
33     cplbit(P1.0);
34     delay(1);
35 }
36 }
37 /*-----*/
38 bex(char n) /* 蜂鸣器“滴”n声 */
39 {
40     char i;
41     for(i=0; i<n; i++)
42     {
43         led_bl();
44         be();
45     }
46 }
47 /*-----*/
48 /* 8255(2)
49 PA i/p    RF rx code
50 PB o/p    RF tx password
51 PC o/p    RF tx data, PC4 TE\
52 CR2=0x90
53
54 pa0 -- d3 --> k2  0111=7
55 pa1 -- d2 --> k3  1011=11
56 pa2 -- d1 --> k4  1101=13
57 pa3 -- d0 --> k1  1110=14
58 pa4 -- en (active high)
59 */
60
61 char rx() /* 接收测试程序 */
62 {
63     unsigned char in;
64
```

```
65  in=pa2; /* 查询 PA4 引脚是否变为高电位 */
66  if( (in & 0x10)==0x10) /* active high */
67  { /* 若 PA4 引脚变为高电位, 表示发射器有按键, 则接收信号 */
68  while(1)
69  {
70      in=pa2; /* 等待发射器的按键是否放开来 */
71      if( (in & 0x10)==0) break;
72  } /* 判断按键值 */
73  if( (in & 0x01)==0) return 2;
74  if( (in & 0x02)==0) return 3;
75  if( (in & 0x04)==0) return 4;
76  if( (in & 0x08)==0) return 1;
77  }
78  return 0; /* 没按键传回 0 值 */
79  }
80  /*-----*/
81  main()
82  {
83  char in;
84
85  be(); /* 蜂鸣器“嘀”一声 */
86  led_bl(); /* 工作 LED 闪动 */
87  serinit(9600); /* 设定 8255 工作模式 */
88  printf(title); /* 显示工作消息 */
89  /* 设定 8255 工作模式 */
90  cr2=0x90; /* RF PA i/p only */
91  printf("Please push RF_TX k1 k2 k3 k4 \n");
92  while(1) /* 无穷循环 */
93  {
94  in=rx(); /* 接收数据进来并检查是否按键 */
95  if(in==0) continue; /* 未按键则不做处理 */
96  else
97  { /* 显示按键值 */
98      printf("key %d pressed ... \n",in);
99      bex(in); /* 蜂鸣器发出“嘀”声 */
100  }
```

```
101 }
102 }
103 /*-----*/
```

16.6.2 P51 发射无线电的信号

图 16-9 是 P51 发射无线电遥控信号的电路图，以一 3 PIN 排线连至高频发射控制板，高频发射控制板在实验时是用发射器改装的，改装的步骤如下：

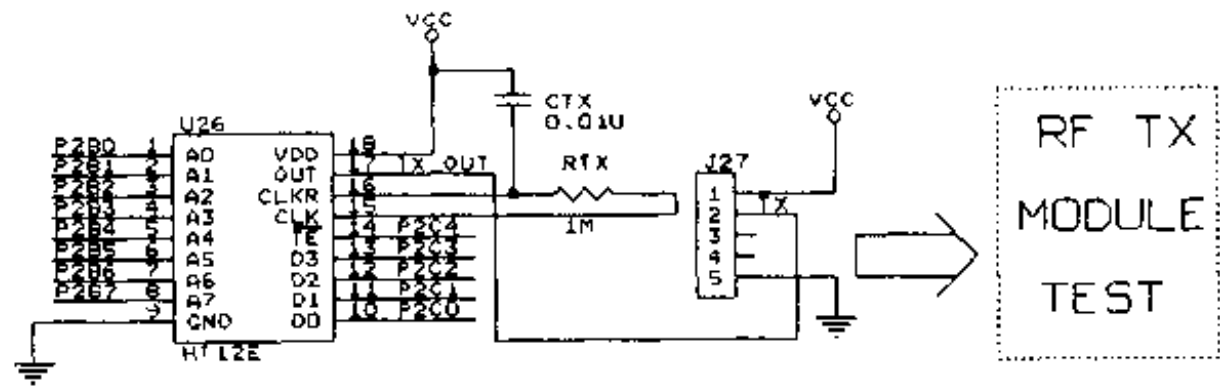


图 16-9 P51 发射无线电遥控信号的电路图

1. 以配对的小型发射器直接发射控制 RF51 看其工作是否正常。接收器及发射器是配对的，工作时频率必须一样，密码一致。
2. 准备另一组小型发射器 TEST(将进行改装)，可以控制 RF51 看其工作是否一样。
3. 将 TEST 的外壳螺丝松开，打开外壳，取出控制板及电池。
4. 以 3 PIN 排线连接 P51_PCB J27 接点 (VCC、TX、GND) 至 TEST 的控制板，信号连接如下：

J27 接点	TSET 的控制板
VCC	HT12E PIN 18
TX	TX_IN
GND	HT12E PIN 9

其中 TX_IN 接点位于 TEST 控制板上 HT12E PIN 17 与电阻 47KΩ之间，改装时请将电阻 47KΩ该点解焊，连到 J27 接点(TX)，也就是由 P51_PCB 上的 HT12E(U26)PIN 17 送出控制信号到电阻 47KΩ，以取代原来的发射器 TSET 控制板上的 HT12E。图 16-10 为其改装示意图。

发射器改装完成后，将其连至 P51 控制板的 J27 接头，8255 端口 B，经由程序规画成为输出端，用以设定 HT12E 的工作密码，由 8255 端口 C 送出数据，相对的控制信号如下：

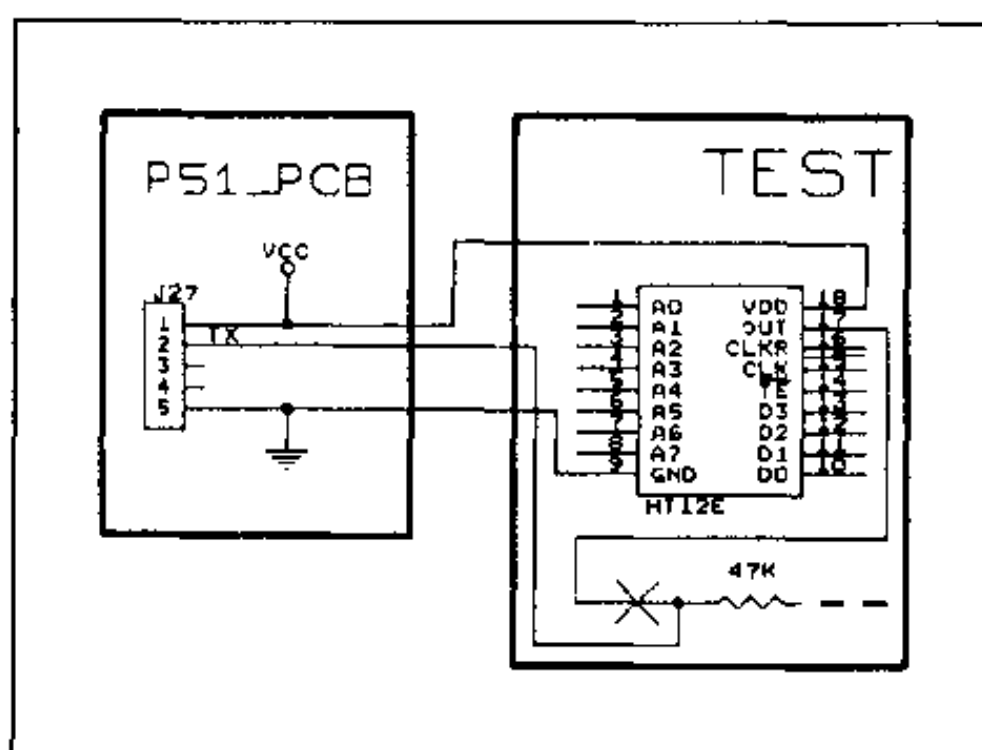


图 16-10 发射器改装示意图

HT12E 引脚	8255 PC 引脚
D0	PC0
D1	PC1
D2	PC2
D3	PC3
TE\	PC4

其中 PC0~PC3 送出 4 位的数据，并由 PC4 送出允许发射信号(低电位工作)，用以控制 HT12E TE\引脚，将代码数据经过高频发射控制板发射出去。

实验注意事项

1. RF51 J1 接头必需接入 12V 电源调整器。
2. 以小型发射器直接发射控制 RF51 看其工作是否正常。
3. 以 P51 发射控制信号，来控制 RF51，RF51 的工作应该一样。

执行结果

控制程序将发射的工作密码设为 0X07，因此无线遥控接收模块(RF51) 解码密码设定必需一致，先将 DIP 开关 123 拨到 OFF 位置。RF51 上有一 7 段数码管，用以显示接收的状态，当 P51_PCB 上按下按键 K1~K4 时，则会将相对的控制代码发射出去，由无线遥控接收模块接收数据，并显示如下：

P51_PCB 按键	RF51 显示
K1	1
K2	2
K3	3
K4	4

目前 P51_PCB 上的按键只有 K1~K4 有效, 而 RF51 使用测试接收程序 RTEST.C 即可, 所以 P51_PCB 的按键功能就如同小型发射器上的 4 个按键一样。

程序清单

```

1  /* PTX.C */
2  #include "8051io.h" /* 载入 MC51 头文件 */
3  #include "8051reg.h"
4  #include "8051bit.h"
5  #include "8051int.h"
6  #include "p51.h" /* 载入 P51 I/O 控制头文件 */
7
8  char *title="P51_PCB TX to RF_TX_PCB RF51 RX\n";
9  #define CWORD 0x88 /* PA PB PC0~3 o/p PC4~7 i/p */
10
11 char act[4]={0xfe, 0xfd, 0xfb, 0xf7} /* 扫描控制信号 */
12 char key; /* 键盘扫描值 */ /* 16 个键盘按键 */
13 char skey[16]={'A','B','C','D','3','6','9','#','2',
14              '5','8','0','1','4','7','*'};
15 rf_code[]={0, 14, 7, 11, 13}; /* 发射控制代码 */
16 char scan_key(); /* 键盘扫描控制程序 */
17 /*-----*/
18 delay(int t) /* 延迟子程序 */
19 {
20     int i,j;
21     for(i=0; i<t; i++)
22         for(j=0; j<10; j++);
23 }
24 /*-----*/
25 led_bl() /* 工作 LED 闪动 */
26 {

```



```
27 char i;
28 for(i=0; i<4; i++)
29 {
30     cplbit(P1.7);
31     delay(40);
32 }
33 }
34 /*-----*/
35 be() /* 蜂鸣器“嘀”一声 */
36 {
37     char i;
38
39     for(i=0; i<100; i++)
40     {
41         cplbit(P1.0);
42         delay(1);
43     }
44 }
45 /*-----*/
46 char scan_key() /* 键盘扫描控制程序 */
47 {
48     char i,j, find, ini, inj;
49     char in;
50
51     find=0; /* 清除按键标志 */
52     for(i=0; i<4; i++)
53     {
54         /* 由 PC 端口(PC0~PC3) 送出扫描控制信号 */
55         pc=act[i];
56         delay(3);
57
58         /* 由 PC 端口(PC4~PC7) 读取按键返回代码 */
59         in=pc;
60         in=in>>4; /* 右移 4 位 */
61         in=in | 0xf0; /* 高 4 位设为 1 */
62
```

```
63  /* 检查是否按键 ? */
64  for(j=0; j<4; j++)
65      if(act[j]==in)
66      {
67          find=1; /* 设定按键标志 */
68          inj=j; ini=i; /* 记录扫描指针值 */
69      }
70  } /* scan 1 time */
71  /* 没按键传回 0 值 */
72  if(find==0) return 0;
73
74  /* i, j ----> key : 0~15 */
75  key=ini*4+inj; /* 计算按键值 */
76  return 1; /* 有按键传回 1 */
77  }
78  /*-----*/
79  tx_rf(char k) /* 将代码数据经过高频发射控制板发射出去 */
80  {
81      char c;
82
83      c=rf_code[k]; /* 取出代码 */
84      pc2=0xe0+c; /* TE\ on */
85      delay(150); /* 延迟一下 */
86      pc2=0xff; /* TE\ off */
87  }
88  /*-----*/
89  main()
90  {
91      char c, f;
92
93      be(); /* 蜂鸣器响一声 */
94      led_bl(); /* 工作 LED 闪动 */
95      serinit(9600); /* 初始化串行接口 */
96      printf(title); /* 显示工作消息 */
97
98      cr=CWORD; /* 设定 8255 工作模式 */
```

```
99    cr2=0x90; /* RF PA i/p only */
100    pc2=0xff; /* TE\    off  */
101
102    /*  设定 HT12E 的密码 A0~A7 */
103    /*  RX mod.  dip 0=on  00000111b= 0x07 */
104    pb2=0x07; /*  由端口 B PB0~7 送出密码 */
105    printf("Please set RF51 rx module DIP SW  123 off\n");
106    printf("Please key in key   1 2 3 4   to tx RF out \n");
107
108    while(1) /*  无穷循环 */
109    {
110    f=scan_key(); /*  请输入数字 1~4  */
111    if(f==1) /*  有按键了  */
112    {
113        c=skey[key]; /*  按键转换 */
114        c=c-0x30; /*  数值转换 */
115        if( (c<5) && ( c>0) ) /*  检查数值是否有效 */
116        {
117            be(); /*  蜂鸣器 “嘀” 一声 */
118            printf("key %d pressed \n", c);
119            tx_rf(c); /*  送出该键的代码数据 */
120        }
121    } /* if keyed */
122 } /* loop */
123 }
124 /*-----*/
```

第 17 章 8051 红外线遥控接口控制

现在的家电产品大部分配备有红外线遥控器，在 PC 上也有红外线的传输接口。有些鼠标加上了红外线控制接口，鼠标变成了无线鼠标。电子游戏的操纵摇杆，加上了红外线控制接口变成了无线摇杆。所以红外线控制接口是一项相当有用的控制接口。在本章中将介绍红外线接口的应用场合及如何开始做红外线接口实验，并介绍红外线接口实验套件，方便有兴趣的朋友来研究。

17.1 红外线接口应用场合

日常生活中使用红外线接口控制的装置相当多，列举如下：

- 1.电视机、录放相机、空调机、音响等遥控器；
- 2.万用可学习型遥控器；
- 3.电脑无线红外线传输接口；
- 4.红外线无线耳机；
- 5.红外线无线鼠标；
- 6.红外线无线摇杆；
- 7.红外线无线键盘；
- 8.红外线防盗感知器；
- 9.计数器；
- 10.自动冲水设备；
- 11.障碍物碰撞检查。

红外线控制是一项相当有用且有趣的接口实验，了解此一方面的控制技术，将可以灵活应用在许多自动控制及智慧型电子产品设计上。

17.2 红外线接口实验套件简介

目前家电产品的遥控器都是以红外线遥控器为主，由于使用的普及，为了避免混淆，各个品牌的家电产品都有固定的红外线信号代码，于是市面上有一种红外线遥控器，其内部已经设定有几十家品牌的电视机、录放相机的红外线信号代码，因此使用者只需要设定所要遥控的电视机或是录放相机的品牌编号，便可以控制其工作，其方便性是以一个遥控器可以再重新设定使用在不同品牌的家电产品控制。此一型号遥控器称为通用型遥控器。

另外的一型号红外线遥控器是具有学习功能的遥控器, 可以将不同品牌的家电产品的遥控器控制键代码集成在同一个遥控器上, 其优点是只需要持有一个遥控器便可以到处去遥控不同的家电产品, 这一型号遥控器称为可学习型遥控器。

为了方便做红外线接口及遥控器相关的实验, 我们设计了一块红外线接口实验板 (IR_PCB), 可以分析一般市面上红外线遥控器所发射出来的信号, 还可以观察其波形, 也可以利用 PC 来存储多组信号, 以备将来使用。基本上已经有可学习型遥控器的基本功能, 特别适合做实验或研究用。

17.2.1 红外线接口实验套件介绍

红外线接口实验套件组成如下:

- IR_PCB 一块(含红外线接收模块及发射器);
- IR_PCB 电路图;
- 红外线遥控器(IR_SET 32 键);
- IR_SET 电路图及技术数据;
- 排线 16 PIN 连接语音卡 SP_CARD;
- 使用说明书;
- PC /8051 相关实验控制电路图;
- 学习示范程序 6 个。

学习示范程序(含源程序):

1. IR.C: PC 上红外线信号波形观察及学习程序 (TURBO C);
2. IC.C: PC 上 IR_SET 遥控器解码程序(TURBO C);
3. I1.ASM: 单片机 8051 IR_SET 遥控器解码程序 (8051 汇编语言程序);
4. I2.ASM: 单片机 8051 IR_SET 遥控器应用示范程序 (8051 汇编语言程序);
5. KIR.C: PC 控制 CD 放音机控制程序(TURBO C);
6. VIR.C: 声控 CD 放音机控制程序(TURBO C)。

相关学习示范程序所搭配的软硬件如下:

程序	搭配的硬件接口
IR.C	PC+IR_PCB+SP_CARD
IC.C	PC+IR_PCB+SP_CARD
I1.ASM	IR_PCB
I2.ASM	IR_PCB
KIR.C	PC+IR_PCB+SP_CARD
VIR.C	PC+IR_PCB+SP_CARD

其中 IR_PCB 为红外线接收及发射学习板, SP_CARD 为语音实验卡。有关 PC 上识

别示范程序的展示说明,可参考《声控电脑制作与应用入门》松岗版一书。

17.3 示范程序介绍

红外线接口实验套件设计的目的主要是方便初学者实验用,因此以 PC(需加上语音实验卡)及 8051(可以使用 IR_PCB 板上的 8051)为硬件的工作平台,使用者在熟悉其控制原理后,可以轻易地修改其功能,加入到自己的产品设计或专题制作中,为了方便初学者学习,本套件另外提供有原始控制程序。

有 6 个学习示范程序简介如下:

1. IR.C: PC 上红外线信号波形观察及学习程序(TURBO C);
2. IC.C: PC 上 IR_SET 遥控器解码程序(TURBO C);
3. I1.ASM: 单片机 8051 IR_SET 遥控器解码程序(8051 汇编语言程序);
4. I2.ASM: 单片机 8051 IR_SET 遥控器应用示范程序(8051 汇编语言程序);
5. KIR.C: PC 控制 CD 放音机控制程序(TURBO C);
6. VIR.C: 声控 CD 放音机控制程序(TURBO C)。

17.3.1 IR.C: PC 上红外线信号波形观察及学习程序

IR.C 是一个 PC 上的红外线信号学习及观察程序,除了可以显示红外线数据代码波形外,还可以发射红外线数据代码,因此使用 PC 便可以控制红外线遥控的家电产品。

系统组成

整个控制系统组成如下:

- PC 语音卡;
- 红外线接口实验板(IR_PCB);
- 控制程序 IR.EXE 及相关程序文件;
- 红外线遥控器(IR_SET 32 键)。

可执行文件为 IR.EXE,执行后显示如下工作画面:

```
-----  
IR LEARN SET via SP CARD    10 K S.R. ver1.0  
Copyright (C)  VICTOR uP LAB.  1996, 1997  
-----  
SPACE    --> learn code  
t    --> tx ir code  
w    --> look  wave  
l    --> tx 5   times  
s    --> save  IR data
```

```

I  --> load   IR data
f  --> o/p IR TEXT file  IR.T
d  --> DIR *.IR
o  --> DOS SHELL

```

共提供有以下的一些控制指令键：

- 空格键：识别红外线数据代码。
- 按键 ‘t’：发射红外线数据代码。
- 按键 ‘w’：显示红外线数据代码波形。
- 按键 ‘l’：连续发射红外线数据代码 5 次。
- 按键 ‘s’：将红外线数据代码保存为文件。
- 按键 ‘I’：载入红外线数据文件。
- 按键 ‘f’：产生红外线数据文字文件，文件名为 IR.T。
- 按键 ‘d’：显示所有红外线数据文件，文件名为 *.IR。
- 按键 ‘o’：暂时离开可执行文件而返回到 DOS 环境下，输入 “EXIT”，可以继续执行程序。

操作步骤如下：

1. 按空格键先识别红外线数据代码。此时系统正等待接收红外线信号进来，将红外线遥控器对着 IR_PCB，并按下遥控器上的某一按键，一切正常的话，则会显示红外线信号波形的高低电位数据及波形。执行结果如下：

```

tlen=37
HI:
44 5 16 16 5 4 5 5 15 16 16 5 16 5 16 16 5 4 16 4 4 16 4
4 5 16 5 16 16 5 16 16 16 400 20
LO:
89 7 6 7 6 7 7 6 7 7 6 7 6 7 6 7 6 7 7 7 7 7 7
7 7 6 7 6 7 6 7 6 7 6 91

```

2. 此时可以拿一个家中的红外线遥控器来试，例如拿 CD 音响的遥控器来做测试，看看是否仍可以看到与上面类似的波形，如果可以的话，那么您将可以直接由 PC 来控制 CD 音响的工作，就如同您使用遥控器一般。

3. 将 IR_PCB 对着 CD 音响并按下按键“t”，发射出刚刚所学习到的红外线数据代码。此时 CD 音响将会开始工作。

4. 可以按下“s”键将红外线数据代码保存为文件。红外线数据文件名为*.IR。

5. 按下“l”可以载入红外线数据文件。

测试结果

目前测试过的家电红外线遥控器有以下几种品牌：

- SANYO CD 音响；
- TOSHIBA VTR 录放相机；
- TOSHIBA TV 电视机；
- TOSHIBA 空调机；
- PIONEER CLD CD/LD 播放机；
- PIONEER CD 音响。

注意：

1. 由于市面上家电遥控器种类繁多，红外线数据代码格式众多，本程序只供实验用，本程序并无法保证可以百分之百完全识别市面上所有的红外线遥控器所发射的信号。
2. 识别红外线数据代码时请将台灯灯光移开 IR_PCB，以免灯光影响红外线信号学习，无法捕获真正的红外线信号。
3. 本红外线实验板适合分析载波频率为 38 KHz 的日本系列家电红外线遥控器。

17.3.2 IC.C: PC 上 IR_SET 遥控器解码程序

此一遥控器解码程序是配合遥控器 TX_SET 米做实验的，TX_SET 遥控器上有 32 个按键，当按下每个按键时，都会发射不一样的红外线信号代码，利用此一程序可以将红外线信号代码加以解码并显示 4 字节的数据，其格式如下：

第 1 字节	第 2 字节	第 3 字节	第 4 字节
86H	6BH	按键代码 1	按键代码 2

其中 86H，6BH 为 TX_SET 此一遥控器的客户代码，为固定代码。而按键代码 1 及按键代码 2，才是实际上的按键解码值，32 个按键值都不一样。

系统组成

整个控制系统组成如下：

- PC 语音卡；
- 红外线接口实验板 (IR_PCB)；
- 控制程序 IC.EXE 及相关程序文件；
- 红外线遥控器 (IR_SET 32 键)。

可执行文件为 IC.EXE 执行后显示如下工作画面：


```

-----
IR CODE decode  V1.0 96.6.19
Copyright (C)  VICTOR uP LAB. 1996,1997
SPACE  --> learn code
-----

```

按空格键系统会等待接收红外线信号进来，将红外线遥控器对着 IR_PCB，并按下遥控器上的 K1 键，若一切正常的话，则会显示红外线按键解码值。执行结果如下：

```

--0 1 1 0 0 0 0 1 --1 1 0 1 0 1 1 0
--0 1 0 0 1 0 0 0 --1 0 1 1 0 1 1 1
Code 0 =134 (86H)
Code 1 =107 (6BH)
Code 2 =18  (12H)
Code 3 =237 (EDH)

```

同理按 K2 键，则会显示红外线按键解码值。执行结果如下：

```

--0 1 1 0 0 0 0 1 --1 1 0 1 0 1 1 0
--0 0 0 1 1 0 0 0 --1 1 1 0 0 1 1 1
Code 0 =134 (86H)
Code 1 =107 (6BH)
Code 2 =24  (18H)
Code 3 =231 (E7H)

```

同理按 K3 键，则会显示红外线按键解码值。执行结果如下：

```

--0 1 1 0 0 0 0 1 --1 1 0 1 0 1 1 0
--1 0 1 0 1 0 0 0 --0 1 0 1 0 1 1 1
Code 0 =134 (86H)
Code 1 =107 (6BH)
Code 2 =21  (15H)
Code 3 =234 (EAH)

```

17.3.3 11.ASM: 单片机 8051 IR_SET 遥控器解码程序

IC.C 为 PC 上 IR_SET 遥控器解码程序，而 11.ASM 则是单片机 8051 IR_SET 遥控器解码程序，其工作原理如同 IC.C，此一遥控器解码程序是配合遥控器 TX_SET 来做实验。

系统组成

整个控制系统组成如下：

- 红外线接口实验板 (IR_PCB);
- 控制程序 I1.ROM;
- 红外线遥控器(IR_SET 32 键)。

IR_SET 遥控器上有 32 个按键,当按下某个按键时,会发射出红外线信号代码,利用 8051 可以将红外线信号代码加以解码,并将其值经过 RS232 传至 PC 上显示出 4 字节的数据。在 PC 端我们是执行 RS232 监控程序 SE.EXE,程序执行后,在 IR_SET 遥控器上按下 K1 K2 及 K3 键,8051 会通知我们按下了那一按键的 4 字节代码。执行结果如下:

```
-----
SE.EXE   PC RS232   COM2  <9600 N 8 1>
-----
```

```
IR rx decode 4 BYTES V1.0...
```

```
86H  6BH  12H  EDH  86H  6BH  18H  E7H  86H  6BH  15H  EAH
```

17.3.4 I2.ASM: 单片机 8051 IR_SET 遥控器应用示范程序

I2.ASM 为单片机 8051 IR_SET 遥控器应用示范程序,由于红外线信号代码差别只在按键代码 1 及按键代码 2 而每一按键对应的按键代码 1 都不一样,因此如果要区分是按下了那一键,可以直接判断按键代码 1 的值,这是一个较简单的方法,I2.ASM 便是这样的一个程序。

系统组成

整个控制系统组成如下:

- 8051 多功能控制板 P51_PCB;
- 红外线接口实验板(IR_PCB);
- 控制程序 I2.ROM;
- 红外线遥控器(IR_SET 32 键)。

在 PC 端我们是执行 RS232 监控程序 SE.EXE,程序执行后,在 IR_SET 遥控器上按下 K1、K2、K3 及 K4 键,8051 会提示我们按下了哪一按键,如果不是以上 4 个按键,则会显示“x”。执行结果如下:

```
-----
SE.EXE   PC RS232   COM2  <9600 N 8 1>
-----
```

```
IR rx decode K1--K4   V1.0.....
```

```
K1 KEYED  K2 KEYED  K3 KEYED  K4 KEYED  xxxx
```

注意: I1.ASM 及 I2.ASM 此 2 程序必需在 IR_PCB 上执行。

17.3.5 KIR.C: PC 控制 CD 放音机控制程序

本红外线遥控器接口控制实例是使用 PC 来识别遥控器的信号及发射其信号来直接控制家电产品工作。本节将说明如何用 PC 来控制 CD 放音机, 直接按下 PC 按键便可以控制音响工作, 当然在此之前的遥控器信号必需要先进行识别。

基本功能

控制系统所提供的功能键如下:

1. 红外线信号学习:

- 空格键: 识别红外线数据代码。
- 按键“t”: 发射红外线数据代码。
- 按键“w”: 显示红外线数据代码波形。
- 按键“q”: CD 放音机遥控器功能键识别, 例如当按下“q”再按下“0”则可以识别遥控器上的“0”键代码。

相对的 PC 按键的遥控器功能键如下:

PC 按键	遥控器功能键
0~9	NO 0~9 (数字键)
O	ON/OFF(电源开关)
UP/DOWN	VOL. control (音量控制)
LEFT/RIGHT	SEL. song (选择歌曲)
P	PLAY/PAUSE(放音/暂停)
X	STOP(停止)

其中“UP/DOWN”按键及“LEFT/RIGHT”按键为 PC 键盘上的上、下、左、右箭头键。

2. 红外线信号发射: PC 按键代码如上表, 直接按下单键便可以发射红外线信号。

本系统的使用注意事项:

1. 本系统使用前要先将遥控器信号进行识别。
2. 红外线信号识别需要按下 2 个按键, 红外线信号发射只需要按下单键。
3. 如果系统无法显示出红外线数据代码波形, 则表示遥控器信号识别失败, 就无法使用本系统来控制使用者的 CD 音响。

系统组成

整个控制系统组成如下:

- PC 语音卡(SP_CARD);
- 红外线接口实验板(IR_PCB);
- 控制程序 KIR.EXE 及相关程序文件;

- 具有遥控功能的 CD 放音机;
- CD 放音机的遥控器。

执行例子

执行控制程序 KIR.EXE, 显示如下工作画面:

```
-----
KIR PC control CD PLAYER via SP CARD   ver 1.0
Copyright (C)  VICTOR uP LAB.   1996, 1997
-----
```

SPACE --> learn IR code

t --> tx IR code

w --> look wave

```
===== PC VC FN.KEY define =====
```

Learn IR : q+[0--9] q+ lkey

Tx IR:

0--9 --> NO 0--9 o --> ON/OFF

UP/DOWN --> VOL. control p --> PLAY/PAUSE

RIGHT/LEFT --> SEL. song x --> STOP

```
=====
```

IR set list :

```
Learned :  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0
Tlen:      36 33 35 36 35 35 35 36 37 35 36 36 35 35 35 36 35 0 0 0
```

除了显示控制指令键外, 并列出了红外线信号识别的状态, 控制指令键使用说明请参考前面基本功能说明。系统共提供了 20 组红外线信号, 笔者实验使用的是 SANYO(三洋)CD 音响, 目前的红外线信号学习只用了 17 组, “Learned : 1” 表示该组已识别过了, “Tlen:” 则告知该组红外线信号的数据长度, 读者如果要做此一实验时, 必须先将自己的 CD 音响遥控器信号先识别进来, 才可以加以控制。

一般操作步骤如下:

1. 按空格键识别 IR 红外线数据代码, 按下 CD 音响的遥控器电源启动键来测试看是否仍可以看到类似的波形, 如果可以的话, 那么您可以直接由使用本套系统来控制 CD 音响。
2. 若可以看到类似的波形, 则将 IR_PCB 对着 CD 音响, 并按下按键 “t”, 发射出刚刚所识别到的红外线数据代码。此时 CD 音响电源将会打开。
3. 正式识别遥控器信号, 先识别电源开关键操作如下:

- 1) 按“q”键, 再按“o”键, 接着按下遥控器电源启动键。
- 2) 按“q”键, 再按“p”键, 接着按下遥控器放音键。
- 3) 同理识别其他按键, 音量大小控制键、歌曲前进后退选择键、停止键、数字键(0~9)。

4. 测试识别的效果操作如下:

- 1) 按“o”键, 查看 CD 音响电源是否打开?
 - 2) 按“p”键, 查看 CD 音响是否开始放音?
 - 3) 按“p”键, 查看 CD 音响是否暂停放音?
 - 4) 按“p”键, 查看 CD 音响是否又开始放音?
 - 5) 按“x”键, 查看 CD 音响是否停止放音?
5. 一切如果都正常, 则您便可以随心所欲以 PC 按键来控制自己心爱的 CD 音响了。

17.3.6 VIR.C: 声控 CD 放音机控制程序

本红外线遥控器接口控制实例是以声音来控制 CD 放音机工作的, 本实例结合 PC 红外线控制技术 & 声控语音识别技术来完成, 声控语音识别技术是直接使用语音卡声控程序库来完成的。有关声控语音识别技术可以参考《声控电脑制作及应用入门》一书。书中对语音识别技术及应用有深入的介绍。

基本功能

声控 CD 放音机系统, 让我们用语言直接控制 CD 放音机工作, 当然 CD 放音机遥控器信号必须先进行识别, 可以直接使用 KIR.EXE 的红外线数据文件, 来作为遥控器上各个按键所要发射的红外线数据代码。

控制系统所提供的功能键如下:

- 按键“c”: 产生并录制声控 CD 放音机参考样本语音数据文件。
- 按键“l”: 聆听原设定所要识别的语音数据内容, 可以听见 5 个单字音。
- 按键“m”: 修改参考样本语音数据文件。
- 空格键: 进行语音识别并控制 CD 放音机。

系统原设定所要识别的 5 个单音(以中文发音)及语音文件如下:

语音文件名	内 容
DBSP0.SP	ON (开机)
DBSP1.SP	OFF (关机)
DBSP2.SP	PLAY (放音)
DBSP3.SP	PAUSE (暂停)
DBSP4.SP	STOP (停止)

使用者只要说出以上的几个单音, 系统会自动识别出您所说出的那一个单音, 以语

音响应出识别结果并执行相应的工作，如果您所说的那一个单音并不存在于语音参考文件中，则系统会告知“无法识别”。

所执行的相应操作如下：

内 容	操作
ON	打开 CD 音响的电源
OFF	关闭 CD 音响的电源
PLAY	控制 CD 音响放音
PAUSE	控制 CD 放音暂停
STOP	放音停止

其中打开及关闭 CD 音响的电源都是控制发射遥控器上的“POWER”电源按键，而控制 CD 音响放音及暂停则是控制发射遥控器上的“PLAY”放音按键。例如使用者只要说出“开机”，则受控制的 CD 音响电源将会打开来。

系统组成

整个控制系统组成如下：

- PC 语音卡；
- 红外线接口实验板(IR_PCB)；
- 控制程序 VIR.EXE 及相关程序文件；
- 具有遥控功能的 CD 音响；
- CD 音响的遥控器。

执行例子

执行控制程序 VIR.EXE，显示如下工作画面：

```

-----
VOICE IR control CD player.....
Copyright (C) VICTOR uP LAB. 1996,1997
-----

c      --> create DATA BASE
l      --> listen DATA BASE
m      --> modify DATA BASE
SPACE --> SPEECH recog ....
Select ?

```

一般操作步骤如下：

1. 按按键“1”，听一下电脑会识别那几个单音。
2. 按空格键，直接进行语音识别，系统会发出嘀一声，此时可以对着麦克风说出几

个要识别的单音，同时查看 CD 音响是否正确的完成相应的工作。

3. 说出“开机”，查看 CD 音响电源是否打开？
说出“放音”，查看 CD 音响是否开始放音？
说出“暂停”，查看 CD 音响是否暂停放音？
说出“放音”，查看 CD 音响是否又开始放音？
说出“停止”，查看 CD 音响是否停止放音？
说出“关机”，查看 CD 音响电源是否关闭？

以下是其执行结果：

Recog.....			
. ON	Send IR NO:	10	Send IR out ...
. PLAY	Send IR NO:	15	Send IR out ...
. PAUSE	Send IR NO:	15	Send IR out ...
. PLAY	Send IR NO:	15	Send IR out ...
. STOP	Send IR NO:	16	Send IR out ...
. OFF	Send IR NO:	10	Send IR out ...

第 18 章 8051 声控电脑设计

在本书的最后一章中，我们将引导读者进入 8051 单片机另外一个设计领域——应用 8051 做声控电脑设计。其中声控电脑的设计使用语音识别的技术，而语音识别是目前一种热门的技术。电脑语音识别系统，可以用声音直接控制电脑工作，是人机接口中最具有人性化的方式。虽然语音识别的技术已研究开发多年，有不少识别的关键技术陆续发表过，但是要完成一套高效能，可以独立操作的识别系统成本会相当高。

在本系统中，我们利用单片机 8051 来控制语音识别专用 DSP 控制芯片(D6106)，使整个硬件成本降低，它的特点是可以独立操作，不必靠个人电脑来做语音识别的控制，最重要的是其语音识别效果佳且稳定。

本章将介绍 8051/DSP 声控系统的组成，本系统主要由 8051 多功能控制板 P51_PCB 及 DSP 语音识别声控板 DSP_PCB 所组成。使用者只要了解此套声控系统的特性及结构后便可以轻易地以 8051 C 语言来控制本套声控系统，然后按自己的需要完成属于自己的声控应用系统，此外利用本套系统可以自行设计独立操作型声控系统，产品附加价值高。

18.1 声控电脑原理

声控电脑是由人的声音发出指令来控制电脑工作的，可是电脑本身是完全不能识别人的声音，因此必需要让电脑先了解熟悉人讲话的声音及腔调，此步骤我们称为语音特征参数参考样本建立，将原先训练好的声音特色保存为语音参考样本，以便将来做识别时作对照参考。而实际在做识别时，使用者可以说出原先训练过的语音来操作，譬如说“目录”，电脑则做出 DOS 环境下的“DIR”的工作，不过，原先必需先训练电脑一听到“目录”的声音则知道要做“DIR”的指令。当然您也可以直接说“DIR”。图 18-1 所示为声控的处理流程：

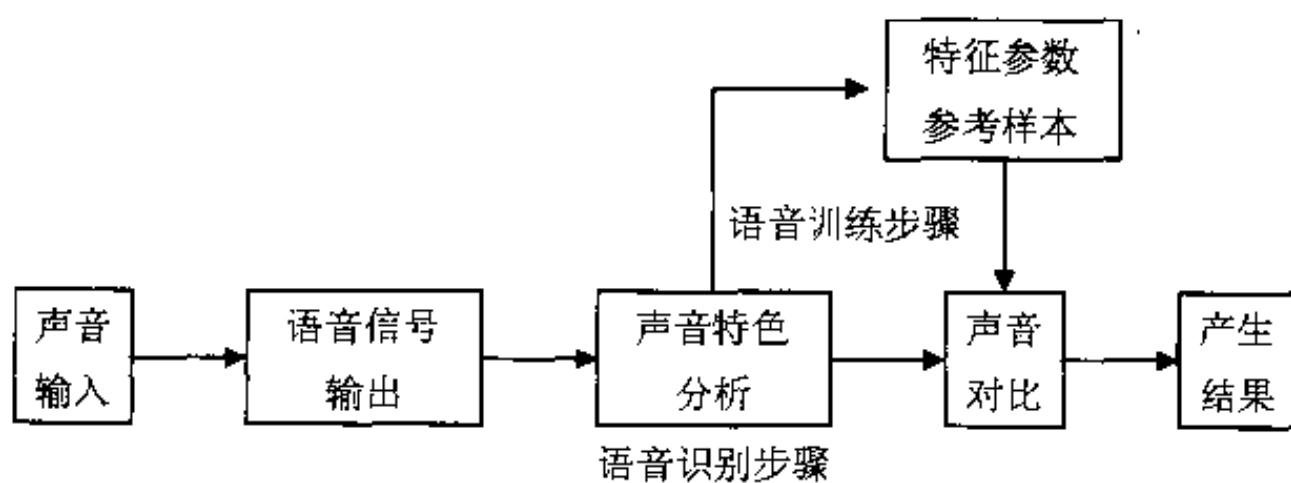


图 18-1 声控处理流程原理图

(1) 首先是声音输入，说话者用麦克风将声音经过语音卡输入电脑。

- (2) 电脑把静音切除，而从中将代表声音的部分取出来，我们称为语音切割。
- (3) 电脑进行声音的特色分析。
- (4) 将输入的语音特色与原先训练好的声音特色进行对比，找出最类似的结果。
- (5) 将结果转换成相对的工作而执行，完成声控的目的。

整个过程分为 2 个阶段：

- **语音训练**

将输入的语音经过分析保存为特征参数参考样本，即告诉电脑将来要识别那些声音。

- **语音识别**

将输入的语音经过分析与原先电脑内的参考样本做对比，找出最相近的声音作为识别结果。

有关声控语音识别技术可以参考“声控电脑制作及应用入门”一书。书中对语音识别技术细节处理有深入的介绍。

18.2 系统特性及组成

18.2.1 DSP 语音识别声控系统特性

- 利用本套系统可以自行设计独立操作型声控系统，产品附加价值高。
- 本系统由 8051 多功能控制板 P51_PC 及 DSP 语音识别声控板 DSP_PCB 组成。
- 本系统通电开机后马上可以进行识别。
- 本系统适合特定语者的单音、字、词及可选择的多组语音样本识别。
- 不限定说话语言，普通话、英语都可。
- 具有自动语音输入检查的功能。
- 识别率可达 95% 以上。
- 系统参数及语音参考样本一旦输入后数据可以长久保存。
- 系统采用模块化设计，扩充性佳，可适合不同的硬件工作平台。
- 线上输入的语音可以压缩成语音数据而由系统说出来。
- 利用 PC RS232 接口做数字信号处理机控制板及 C 语言程序的测试。
- 系统可以独立操作，不依靠 PC。
- 本制作的成品已接近工业产品的原型开发，只要再针对功能做开发便可以应用在多种场所。
- 备有完整技术数据供厂商技术合作。

18.2.2 DSP 语音识别声控系统组成

- 8051 多功能控制板 P51_PCB；

- DSP 语音识别声控板 DSP_PCB;
- 直流电源供给器;
- 小型喇叭;
- 系统工作磁盘;
- 声控展示操作说明。

P51_PCB 控制板用于主控系统, 主要做声控系统的人机接口处理, 所用到的接口包括按键输入及 LCD 消息显示, 并以单片机 8051 来控制 DSP 芯片用于语音识别。

DSP 语音识别声控板 DSP_PCB, 内含有 DSP 语音识别单片机, 及相关的语音输入放大电路及喇叭输出驱动电路。在控制板上并使用无指向性的电容式麦克风做语音输入, 因此使用上相当的方便, 不必一定要拿着麦克风才能做语音输入, 只要在控制系统一米内说话, 便可以做语音识别。

整个语音识别系统可以独立操作, 只要插上电源供给器便可以做声控应用, 基本的执行结果可以显示在 LCD 上, 如果将 RS232 线与 PC 连接后, 便可以将进一步的执行消息传回 PC 而显示在屏幕上, 利用 RS232 接口除了可以显示执行消息外, 也可以做程序设计时的执行消息除错, 在做系统开发时相当有用。

18.3 DSP 控制板简介

整个 DSP 控制板, 我们分以下几部分来做介绍:

1. DSP 控制板组成;
2. DSP 控制板 I/O 接点说明;
3. 跳线设定。

18.3.1 DSP 控制板组成

DSP 单片机中烧录有语音识别的处理程序, 由 8051 下达一系列的控制命令, 来完成整个声控电脑的工作, DSP 控制板由以下几部分组成:

- DSP 语音识别芯片;
- 麦克风语音输入放大电路;
- 喇叭输出驱动电路;
- 数字/模拟转换电路;
- 电源控制电路;
- 语音提示语 ROM;
- 参数存储 SRAM;
- SRAM 电源备份电路。

DSP 语音识别芯片是声控系统的核心, 为了方便测试及除错, DSP 芯片是经过 84

个引脚的小板子与 DSP 控制板相连接, 可以拔除。数字/模拟转换电路是将麦克风语音模拟信号加以转换成数字信号, 交由 DSP 芯片做分析及处理, 然后将其存储在 SRAM 中, 如果要说出所讲的声音, 同样是经过数字/模拟转换电路将数字信号转换为模拟信号, 经过喇叭输出驱动电路后, 可以将声音还原。

DSP 电路板工作时需要 +5V 及 -5V 的电压, 电源控制电路提供此两组独立的电源, 至于语音提示语 ROM 则用来存储出厂时原系统设定的几句语音提示语, 如果使用者想自行更改的话可以交由原厂做特殊方法处理, 以便烧录新的语音提示语。

新的语音输入后, 经过特征参数分析, 便将语音参数存储在 SRAM 中, 有以下两种用途:

1. 作为识别对比的参考样本;
2. 可以作为语音提示语告诉使用者要识别那些声音。

此外并利用 SRAM 电源备份电路, 将系统参数及语音特征参考样本数据长久保存在 SRAM 中, 并不会因为关机又要重新输入新的数据。

18.3.2 DSP 控制板 I/O 接点说明

DSP 控制板上的 I/O 接点功能如下:

- CON1: 20 PIN 排针插座, 连接至 8051 控制板上, DSP 与 8051 的信号传输都由此总线来完成的;
- CON2: 28 PIN 排针插座, 系统保留未使用;
- J1-J4: DSP 控制板插入的插座排针;
- J1A: 数字 5V 直流电压输入, 系统保留未使用;
- J2A: 喇叭输出座;
- J3A: 外部麦克风输入插座;
- J4A: 外部成音音频信号输入插座;
- J5: 外部成音音频信号输入点;
- J6: 3.6 V 备份电池输出接点;
- J7: 模拟 5V 直流电压输入;
- J8: 模拟音频信号输出插座;
- J9: 控制板上电容式麦克风输入;
- J10: 直流电源 9V 输入, 系统保留未使用;
- VR1: 麦克风输入电位控制;
- VR2: 喇叭音量控制。

18.3.3 跳线设定

- JP1: 电容式麦克风(无方向性)/动圈式麦克风(有方向性)选择跳线设定时选择电容式麦克风;

- JP2: 麦克风输入选择;
- JP3: 外部成音音频信号输入选择(JP2 与 JP3 不可同时 ON);
- JP4: 声音输入监听控制, 由麦克风输入的声音马上由喇叭输出;
- JP5: SRAM 62256 设定;
- JP6: ROM 27512 设定;
- JP7: 控制板上电容式麦克风输入设定;
- JP8: 数字与模拟电压接地设定。

18.4 语音识别 DSP 控制命令

整个语音识别过程所要处理的细节工作约可分为以下几种:

- 语音输入信号转换;
- 语音压缩存储;
- 语音合成放音;
- 语音特征参数求取;
- 语音参考样本存储;
- 语音识别对比。

以上所提供的一系列工作都是由 DSP 的控制算法来完成, 并可以配合一般的单片机 CPU 来下达控制命令以启动 DSP 做特定的工作, 完整的语音识别 DSP 控制命令有以下几个:

1. NOP: 不做任何工作

DSP 芯片不做任何工作, DSP 芯片接收到此命令后会回应 0 值。

2. SET_USER: 选取某一使用者

DSP 芯片最多支持至 8 个人。

3. GET_STATUS: 取回主控 CPU 状态值

主控 CPU 取回 DSP 芯片所存储的 CPU 部分状态值。

4. TRAIN: 语音参考样本输入

DSP 芯片将语音参考样本输入并可以压缩在 SRAM 中, 以便下回放音出来。

5. RECOG: 语音识别对比

DSP 芯片对某一测试语音样本与其他指定的语音参考样本进行识别对比, 并送出对比的所有可能结果。

6. SYNT: 语音合成

DSP 芯片说出语音提示语 ROM 内的语音内容。

7. SAVE_STATUS: 存储主控 CPU 状态值

DSP 芯片可以存储主控 CPU 的部分状态值, 可以 GET_STATUS 命令将其取回。

8. CONFIG: 系统状态设定

DSP 芯片做内部系统状态设定。

9. SELF_TEST: 芯片自我测试

DSP 芯片做内部程序 ROM、SRAM 及语音提示语 ROM 的测试, 并会传回所测试的结果。

10. COMPARE: 识别对比

DSP 芯片对某一测试语音样本与其他指定的语音参考样本进行识别对比, 找出最相近的一组样本出来。

11. SET_PARAMETER: 参数设定

DSP 芯片做内部声音取样及处理相关的参数设定

18.5 声控系统展示操作

DSP 语音识别声控系统在出厂前即做过完整的功能测试, 板上并附有一块测试 EPROM, 使用者可以按下列步骤逐一自行验证本控制系统的功能是否正常。在使用此套展示系统前, 请先详细阅读此一操作说明, 您可以轻易地操作此套采用世界最新技术的语音识别声控系统。

1. 将 RS232 连接到 8051 电路板上, PC 端执行 RS232 驱动程序如下:
COM1 --> SEV 03 使用 PC 通信端口 COM1;
COM2 --> SE 使用 PC 通信端口 COM2。
2. 连接喇叭接头至 DSP 板上 J2A。
3. 连接电源线至 8051 电路板上, 并打开电源, 此时电源 LED 灯亮, 工作 LED 闪烁, LCD 屏幕显示如下:

```
1-init 2-play 0
3-tr C-lis D-rec
```

PC 屏幕显示如下:

```
VDSP 8051 DSP VOICE CONTROL DEMO V1.0
Copyright Victor uP LAB. 1995-1996
P51 PCB key function :
1-->init 2-->play 3-->train all C-->listen D-->recog.
4-->up 7-->down *-->train one
```

屏幕显示出单板的测试功能键，分别说明如下：

- 按键“1”：系统参数设定键，若一切硬件正常，会听见“嘀”两声。
- 按键“2”：聆听原厂设定的语音 ROM 数据内容，可以听见 5 个单字音。
- 按键“3”：做语音参考样本训练输入，原设定为 5 组语音，在听到“嘀”的声响同时红色 LED 灯亮起后，输入语音单音，如“0”，此时黄色 LED 灯亮起，表示语音输入被接受，同时会将所输入的语音讲出来，系统将语音存在内存内，待识别时可以说出识别结果。例如我们要识别单音“0”至“4”，可以依序在亮起红灯后说出单音“0”至“4”。
- 按键“4”：向上移动语音参考样本指针，指针范围为 0~4。
- 按键“7”：向下移动语音参考样本指针。
- 按键“*”：语音参考样本修改，在按下此键后，系统会将原来的语音说出来，并且亮起红色 LED 灯，要求使用者重新说出一单音，来修改原先的语音参考样本。
- 按键“C”：聆听系统已存在的语音内容，作为想识别的字词。
- 按键“D”：进行识别，在听到“嘀”的声响后输入语音，约 0.5 秒后系统将识别结果说出来。

一般操作步骤如下：

1. 按下“1”：系统参数设定键。
2. 按“2”：聆听原厂设定的语音 ROM 数据内容。
3. 按下“3”：做语音参考样本训练输入，一次依序说出 5 个想识别的单音。
4. 按下“C”：聆听系统已存在的语音内容。
5. 按下“D”：进行识别。

按键“4”、“7”、“*”用于语音修改。

注意事项

- 当使用者第一次使用此系统时，不必输入新的语音样本，以原来的识别单音“0”至“4”便可以进行识别，一般男生应可以识别正确，如果是识别自己的声音，则可以高达 95% 以上的识别率。
- 请在亮起红灯时才做语音输入，以得到最佳的识别效果。
- 您可以依自己喜好来重新输入语音样本，如“JOHN”、“NANCY”、“PETER”、“MARY”、“SANDY”。
- 按下系统参数设定键后，一定得按下“3”键，做语音参考样本训练输入。
- 按下系统参数设定键后便可以按下“2”键，聆听原厂设定的语音 ROM 数据内容。

18.6 声控系统展示控制程序

声控系统展示工作磁盘含有完整的 8051 控制 DSP 原始 C 语言程序，使用者可以用

原始 C 语言程序开始修改, 马上可以建立自己的一套高级语音识别率的声控应用系统。工作磁盘内容如下:

- VDSP.DOC: DSP 语音识别声控系统使用说明文件;
- VDSP.C: 声控系统 C 语言主控程序;
- DSP.H: DSP 控制相关定义文件;
- DSP.C: DSP 控制程序;
- VDSP.ROM: DSP 语音识别声控可执行二进制文件;
- P51.H: P51_PCB 控制相关定义文件;
- IO.C: P51_PCB 基本 I/O 控制程序;
- X.BAT: MC51 快速编译/下载/执行的控制批处理文件;
- P51.DOC: 8051 多功能控制板 P51_PCB 使用说明文件;
- TP51.C: P51_PCB 基本 I/O 测试程序;
- P51.H: P51_PCB 控制相关定义文件;
- TP51.C: P51_PCB 基本 I/O 控制程序;
- TP51.ROM: P51_PCB 基本 I/O 测试程序可执行二进制文件。

有关 C 语言声控程序可用 DOS MICRO-C51 8051 C 编译器进行编译, 声控语音识别控制程序介绍可以参考《8051 声控电脑制作入门》(松岗版) 一书。书中对于语音识别 C 语言控制程序有进一步说明。

18.7 声控电脑应用

在本系统设计中, 我们应用 DSP 芯片制作一套语音压缩及合成的声控系统。本系统适合特定语言者的单音、字、词的语音识别, 同时不限定说话语言, 国语、台语、英语都可, 而且此一系统具有自动语音输入检查的功能, 当使用者说话结束时系统自动拾取语音信号, 而进行分析对比, 不到一秒的时间便可以将识别结果说出来, 反应速度快, 而且平均的识别率可达 95% 以上。

您是否曾想要利用声音来控制电脑, 或是机器人, 更甚者设计像“霹雳游侠”中的伙计? 当您看到本声控系统时, 将会发现似乎不是那么地困难, 不可能。只要您会 C 语言程序设计, 即可让您实现多年来的愿望, 聪明的您应该还可以想出更多的应用, 享受其中的乐趣和成就感。

笔者的单片机控制实验室, 除了摆几部 PC 外, 还有一些家电产品。有两台电视机、两台录放影机、空调及音响都可以使用遥控器来遥控, 于是工作台上出现了许多个遥控器, 现在便可以用一台 PC 来完全控制了, 进一步还可以使用语音直接下命令来启动, 想想十多年前科幻影片中声控操作, 今天已经实现了。例如说一声“电视”, 电视便会打开。更多的应用实例列举如下:

例子 1: 声控电视选台

通常要看电视时还要拿起遥控器来打开电源, 选出想要观看的电视台, 并控制音大小。例如说看“超人”, 现在您不必那么麻烦了, 只要说出“超人”, 电视上便马上出现“超人”的画面了。

例子 2: 声控灯光照明

如果将此系统安装在家中灯光照明用, 并事先输入电灯的相关声控命令, 晚上回到家中, 打开家门, 一片漆黑, 只需要说声“电灯”, 电灯马上打开, 不必摸黑找开关, 一声命令马上为您带来光明。

例子 3: 声控冷气

如果将此系统安装在家中空调上, 并事先输入冷气遥控的相关声控命令, 大热天回到家中, 打开家门, 只需要说声“冷气”, 则冷气马上打开, 不必要去开冷气或是找遥控器, 按下遥控器启动开关来打开冷器, 一声命令, 马上为您带来凉爽的感觉。还可以说“太热了”, 则冷气还会自动调整冷一点。

例子 4: 声控 CD 音响

想要听一首自己最想听的 CD 音乐吗? 不必拿遥控器了, 只要说声“吻别”, 张学友马上唱歌给您听。说声“火战车”, 提神的音乐马上播放出来。

举了许多的应用例子, 相信聪明的读者已经知道如何应用本套系统了。本工作室致力于语音识别声控应用的研究已有多年的, 在此期间积累了不少的经验, 新的产品及相关实验性的教材在陆续整理开发之中。

由于 DSP 语音识别声控系统的设计复杂度相当高, 根据不同的设计有相当多的参数需要调整, 才能设计出好用的产品, 为使初学者能够很快的进入状况, 减少尝试错误的时间, 能直接通过程序来控制 DSP 语音识别芯片, 我们推出了这套 DSP 语音识别声控展示系统并提供了 C 语言的原始控制程序, 因此非常适合学习 DSP 语音识别用, 如果使用者有进一步需求, 想要进一步设计出新型声控产品时, 我们也提供有整套完整的设计技术数据及开发工具, 欢迎与我们工作室洽商与讨论。

联络 E-MAIL: ufvicwen@ms2.hinet.net

附录 A ROM 模拟器使用

一般从事单片机产品的设计开发工作，一定得烧录 EPROM，即使是使用单片机做设计也可以外接 EPROM 来做开发时程序的快速载入测试，虽然它的功能没有 ICE 来得强，但却可以从事多种微处理机的程序开发工作。对于单片机 8051 的初学者可以直接使用学校所提供的 ICE，如果考虑在家中自己做实验，则使用 ROM 模拟器来做程序开发较合乎经济效益，本文说明书中建议的 ROM 模拟器使用方法，本套模拟器价格便宜，功能齐全，操作简单，是单片机程序设计师、单片机业余玩家、学生、个人工作室必备的单片机程序开发工具。

A.1 ROM 模拟器特性

- 使用 I/O 连接方式控制模拟器，程序代码下载速度快。
- 可模拟 EPROM 2716/2732/2764/27128/27256，由 DIP 开关设定，最高可模拟 64K 字节。
- 适合各种 CPU：6502、Z80、8088 及单片机 8048、8051、Z8 及其他各种型号的 CPU。
- PC 的 I/O 位置可以从卡上的 DIP 开关设定，一部 PC 可以同时控制多台 ROM 模拟器。
- 模拟器本身可产生 RESET 信号触发目标板而执行程序。
- 操作简单，使开发单片机系统程序如同开发 PC 上的应用程序那样方便而快速。
- 模拟器通过接口卡连接拉至 PC 外部，方便操作。
- 由模拟器上 LED 指示工作或由屏幕来显示工作的状态。
- 可执行文件格式为可执行的二进制文件。
- 交互式的软件控制程序，单键指令操作容易。
- 可从指令行中传入想模拟的程序文件名，直接进行程序模拟。
- I/O 解码地址可调整，避免与任何现有接口卡相冲突。

A.2 ROM 模拟器使用

图 A-1 是 ROM 模拟器组成示意图，I/O 解码控制卡是插在 PC 插槽上，ROM 模拟器放在 PC 外面方便操作。

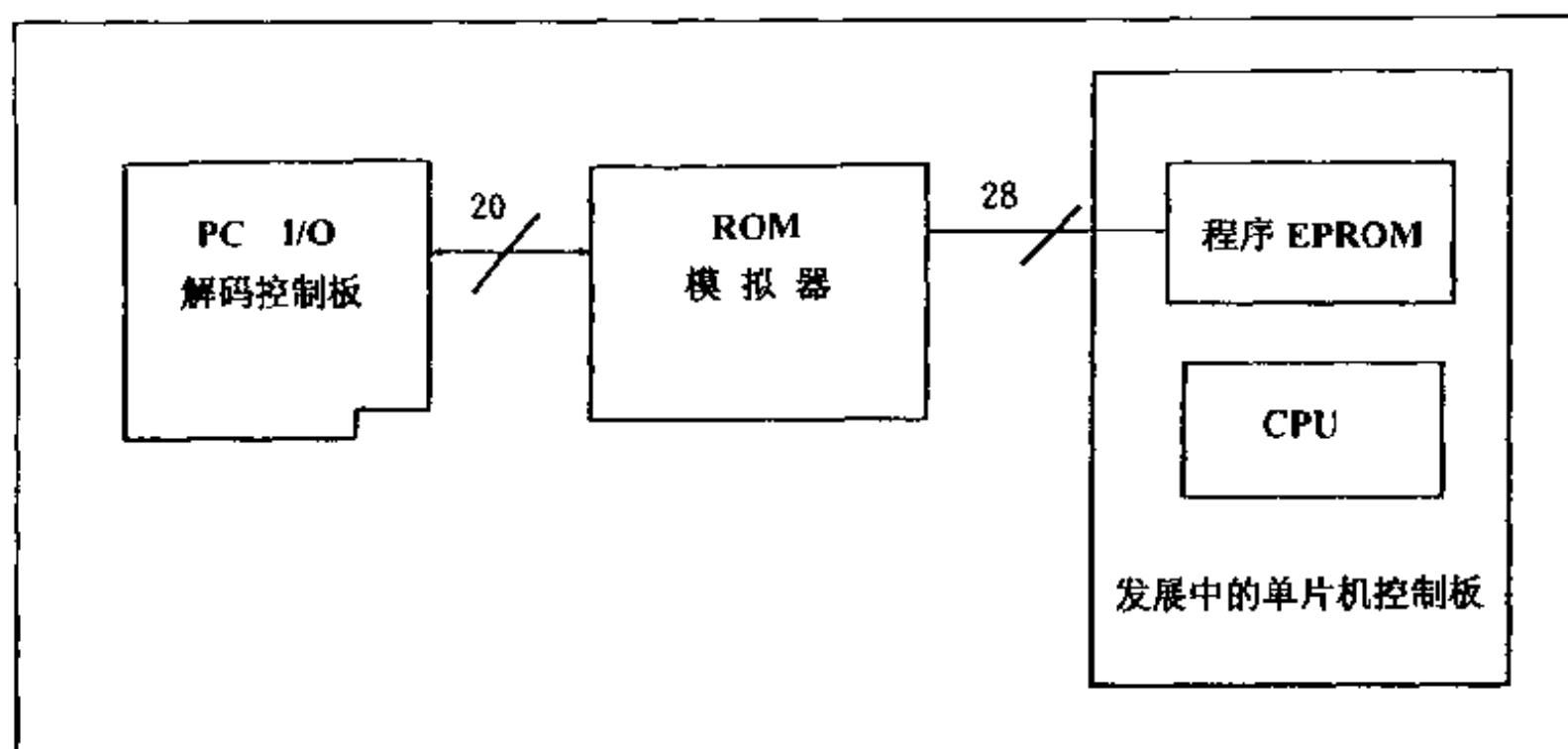


图 A-1 ROM 模拟器组成示意图

图 A-2 是 ROM 模拟器连接示意图，注意，排线请勿插反，在拔插连接接头时请先关闭电源，整个操作步骤如下：

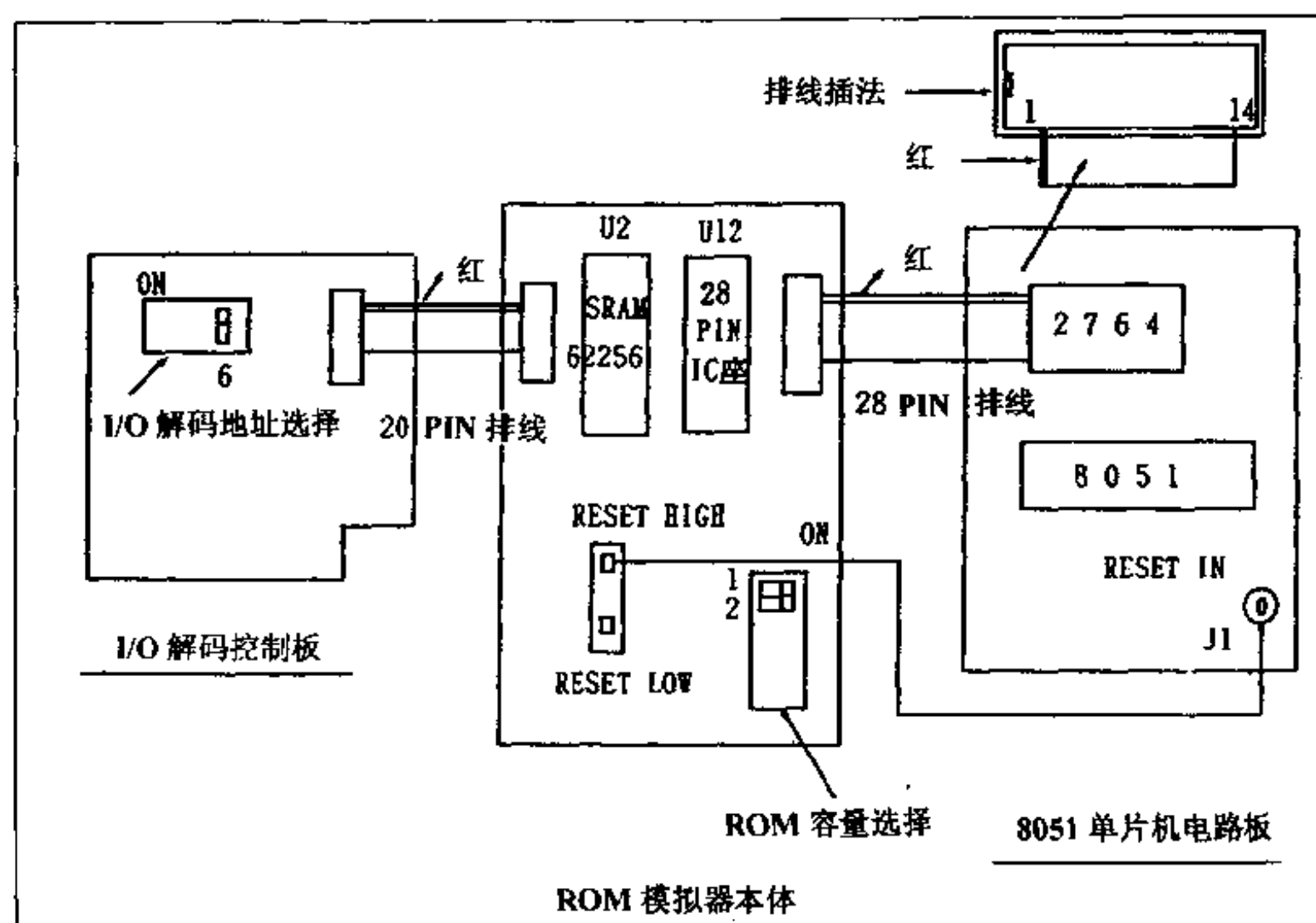


图 A-2 ROM 模拟器连接示意图

1. 安装 ROM 模拟器。
2. PC I/O 端口地址设定，系统原设定 I/O 地址为 0X250。
3. EPROM 型号设定，系统原设定为模拟 2764。
4. 以鳄鱼夹连接 RESET 信号至 8051 电路板上。

5. 同时载入模拟程序，设定 I/O 地址，并直接进入模拟的状态可以输入 EMU TEST.ROM 5 D<ENTER>，此时 8051 控制程序 TEST.ROM 已在 8051 电路板上执行。

详细的说明可以参考 ROM 模拟器使用说明文件。

附录 B 8051 多功能控制板零件表

B.1 集成电路 IC

序号	数量	参考符号	名称
1	1	U1	8051 或 8751 或 89C51(含 40PIN IC 座)
2	1	U2	74LS273(含 20PIN IC 座)
3	1	U3	2764 (8K BYTE EPROM)(含 28PIN IC 座)
4	1	U4	6116 (2K BYTE)(含 28PIN IC 座)
5	1	U5	74LS138(含 16PIN IC 座)
6	2	U6,U15	74HC00(含 14PIN IC 座)
7	1	U7	T518B
8	1	U8	LM386(含 8PIN IC 座)
9	2	U9,U10	7805
10	3	U11,U12,U21	8255(含 40PIN IC 座)
11	1	U13	MAX232(含 16PIN IC 座)
12	1	U14	LCM LCD 模块
13	1	U16	PSG8910(含 40PIN IC 座)
14	1	U17	74LS02(含 14PIN IC 座)
15	2	U18,U19	74LS07(含 14PIN IC 座)
16	4	U20	7 段数码管 X4(含 40PIN IC 座)
17	1	U22	DA08(含 16PIN IC 座)
18	1	U23	7660(含 8PIN IC 座)
19	1	U24	4558(含 8PIN IC 座)
20	1	U25	DS1232(含 8PIN IC 座)
21	1	U26	HT12E(含 18PIN IC 座)

B.2 电 阻

序号	数量	参考符号	名称
1	22	R1~R7 R9,R12	1K

		R15~R23	
		R25~R28	
2	2	R8,R11	200
3	2	R13,R0	1
4	1	R14	10
5	1	R24	2K
6	1	R01	100K
7	1	R10	10K
8	1	RTX	1M
9	1	VR1	100K
10	4	RP1,RP2,RP4,RP5	电阻排 1K 9 PIN
11	1	RP3	电阻排 470 9 PIN

B.3 电 容

序号	数量	参考符号	名称
1	2	C0,C26	100P(101) 陶瓷电容
2	9	C1,C11,C13,C17 C22~C25,C30	10 μ F/16V 电解质电容
3	2	C2,C3	10P 陶瓷电容
4	6	C4,C9,C15,C21 C27,C29	100 μ F/16V 电解质电容
5	10	C5,C6,C8,C12, C14,C16,C28,C10 C18~C20	0.1 μ F 纸介电容
6	1	C7	220 μ F/25V 电解质电容
7	1	CTX	0.01 μ F 麦拉电容

B.4 跳 线

序号	数量	参考符号	名称
1	11	JP1~JP8,JP13 JP16,JP18	2PIN 跳线
2	8	JP0,JP9~JP12 JP14,JP15,JP17	3PIN 跳线

B.5 插 座

序号	数量	参考符号	名称
1	2	J1	1PIN
2	7	J2,J3,J5 J6,J7,J14 J26	2PIN 插座
3	1	J27	5PIN 插座
4	1	J4	电源接头
5	1	J8	14PIN 针座
6	1	J9	14PIN 母座
7	1	J10	RS232 9 PIN 插座
8	1	J11	3PIN 插座
9	1	J25	4PIN 插座
10	1	J12	PC AT 62 PIN 插座
11	1	J13	40 PIN 针座
12	2	J15,J18	30 PIN 针座
13	2	J16,J17	20 PIN 针座
14	1	J19	8 PIN 针座
15	4	J20-J23	16 PIN DIP 针座
16	1	J24	6 PIN 针座母座
17	1	J28	16 PIN 针座

B.6 其 他

序号	数量	参考符号	名称
1	1	Q1	晶体管 2SC945
2	2	D1,D4	LED
3	3	D2,D3,D5	IN4001
4	16	K1~K16 S1,S3	按键
5	1	S2	DIP SW 16 PIN
6	1	U9	散热片
7	1	BAT	3.6V 充电电池
8	1	LED	20 PIN 条状 LED

附录 C AT89C1051/AT89C2051 特性介绍

ATMEL 推出可电擦除的 AT89C51 8051 兼容芯片，使之做 8051 的相关实验更方便。不久后又推出 AT89C1051 及 AT89C2051 8051 兼容芯片，包装只有 20 个引脚，可以缩小电路板体积设计，同时也可以用电信号来擦除内部数据，专门用于小型简单不需很多 I/O 控制的产品设计。本文将特性做介绍，提供给读者做参考。

图 C-1 是 AT89C1051 的引脚图。

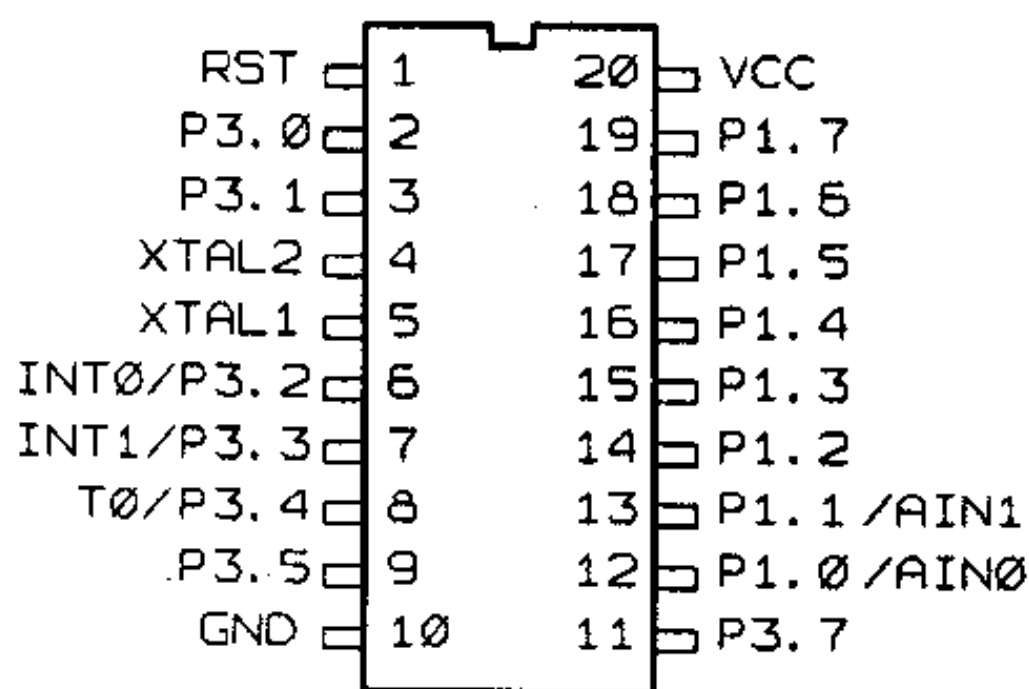


图 C-1 AT89C1051 引脚图

C.1 AT89C1051 特性介绍

- 指令与 MCS-51 芯片兼容；
- 内含 1K 字节的可反复电气烧录及擦除内存；
- 工作电压为 2.7V 至 6V；
- 工作频率最高可至 24 MHz；
- 内含 64 字节的 SRAM；
- 15 条可编程控制的 I/O 线；
- 一组 16 位的计时计数器；
- 三个中断触发源；
- 一个模拟比较器；
- 低功率省电模式控制。

端口 1 P1 可以作为 8 位双向 I/O 引脚控制，P1.2 到 P1.7 提供内部提升电阻，P1.0

及 P1.1 则需要外部的提升电阻，P1.0 也作为内部模拟比较器的正端输入(AIN0)，P1.1 则作为内部模拟比较器的负端(AIN1) 输入，P1 的输出缓冲器可以吸入 20mA 而直接驱动 LED 显示器。

端口 3 引脚 P3.0~P3.5 及 P3.7 可以作为 7 位双向 I/O 引脚控制，并提供内部提升电阻，P3.6 位用于内部比较器控制，无法做一般 I/O 控制，P3 的输出缓冲器可以吸入 20 mA 的电流。

端口 3 所提供的其他额外特殊功能如下：

PIN 引脚	特殊功能
P3.2	INT0 外部中断 0 输入
P3.3	INT1 外部中断 1 输入
P3.4	T0 计数器 0 输入

图 C-4 是 AT89C2051 的引脚图。

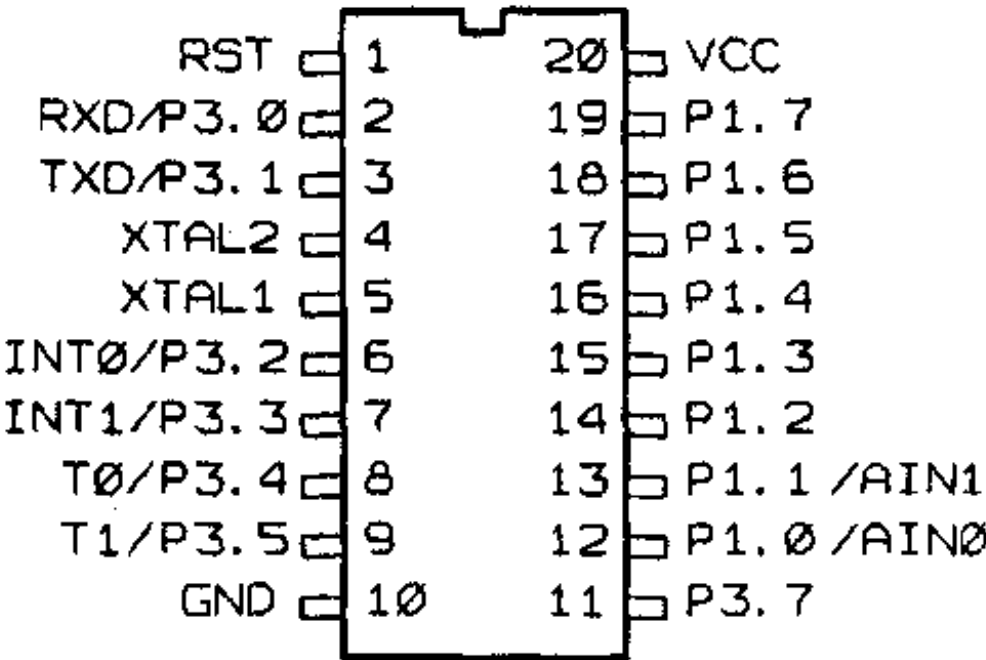


图 C-4 AT89C2051 引脚图

C.2 AT89C2051 特性介绍

- 指令与 MCS-51 芯片兼容；
- 内含 2K 字节的可反复电气烧录及擦除内存；
- 工作电压为 2.7V 至 6V；
- 工作频率最高可至 24 MHz；
- 内含 128 字节的 RAM；
- 15 条可编程控制的 I/O 线；
- 两组 16 位的计时计数器；
- 一组串行通信端口；
- 5 个中断触发源；

- 一个模拟比较器；
- 低功率省电模式控制。

端口 1 P1 可以作为 8 位双向 I/O 引脚控制，P1.2 到 P1.7 提供内部提升电阻，P1.0 及 P1.1 则需要外部的提升电阻，P1.0 也作为内部模拟比较器的正端输入(AIN0)，P1.1 则作为内部模拟比较器的负端输入(AIN1)，P1 的输出缓冲器可以吸入 20mA 而直接驱动 LED 显示器。

端口 3 引脚 P3.0 到 P3.5 及 P3.7 可以作为 7 位双向 I/O 引脚控制，并提供内部提升电阻，P3.6 位用于内部比较器控制，无法做一般 I/O 控制，P3 的输出缓冲器可以吸入 20 mA 的电流。

端口 3 所提供的其他额外特殊功能如下：

P1N 引脚	特殊功能
P3.0	RXD 串行通信输入
P3.1	TXD 串行通信输出
P3.2	INT0 外部中断 0 输入
P3.3	INT1 外部中断 1 输入
P3.4	T0 计数器 0 输入
P3.5	T1 计数器 1 输入

附录 D 89CXX 烧录模拟器 EPM89 特性

通常烧录 89C51 芯片可以使用多功能烧录器(或称为万用烧录器), 配合其烧录程序来进行烧录, 但由于功能多, 市场上价格便高很多, 通常学校的实验室或公司研发的单位会采用, 一般的业余玩家如果有额外的经济开支才会考虑购买。而 89CXX 烧录模拟器 EPM89 价格便宜, 操作简单, 一行指令便可以完成快速载入文件、烧录、直接模拟 89C51 的工作, 完全免拆排线, 最适合业余玩家、学生在家中从事 8051 程序设计、自动控制及专题制作使用。

D.1 EPM89 特性

- 专门烧录 ATMEL 公司生产的 89CXX 系列 IC。
- 可以烧录 89C51(4K)、89C52(8K)、89C55(20K)、89C1051(1K)、89C2051(2K) 单片机。
- 烧录后可以直接模拟 40 PIN 89C51(4K)、89C52(8K)、89C55(20K) 单片机。
- 烧录后可以直接模拟 20 PIN 89C1051(1K)、89C2051(2K) 单片机。
- 使用打印机并行口连接, 不需连接 I/O 接口卡, 可以用于笔记本电脑。
- 文件格式为可执行的二进制文件。
- 交互式的软件控制程序, 单键指令操作。
- 操作简单, 一行指令便可以完成烧录及模拟的工作。烧录后完全免拆排线直接模拟 8051 的 I/O 工作。
- 快速烧录及模拟的指令: EPM89 TEST.TSK D
- 可以做一般打印机并行口实验, 提供 16 BIT 输出, 8 BIT 输入, 并含有 TURBO C 的控制示范程序。
- 成品配件齐全: 含 40 PIN 排线, 20 PIN 排线, 打印机排线, 电源供给器, AT89C51 一块。
- 提供套件供使用者 DIY(Do it yourself 自己做)用。

附录 E 89CXX 烧录模拟器 EPM89 使用说明

1. 系统配件;
2. 文件说明;
3. 快速安装说明;
4. 注意事项;
5. 快速烧录模拟说明;
6. 模拟 40/20 PIN 8051 单片机;
7. 完整操作说明;
8. 打印机 I/O 示范程序。

E.1 系 统 配 件

1. 烧录模拟器控制板一块;
2. 打印机信号线一条;
3. -12V 直流电源供给器一个;
4. 20 PIN、40 PIN 模拟信号线各一条;
5. AT89C51 一块;
- 6 系统工作磁盘;
7. 使用说明书。

E.2 文 件 说 明

- EPM89.EXE: 89C51、89C52、89C55 烧录模拟程序 89C1051、89C2051 模拟程序。
- EP2051.EXE: 89C1051、89C2051 烧录程序。
- IR.ROM: 8051 红外线信号分析程序(配合红外线接口实验板 IR_PCB)。
- LED.TSK: 8051 P1 8 LED “走马灯”式电路示范可执行文件。
- LPD.PRJ: 打印机 I/O 示范程序计划文件。
- LPD.EXE: 打印机 I/O 示范程序可执行文件。
- LPD.C: 打印机 I/O 示范程序。
- LPIO.OBJ: 打印机 I/O 驱动程序。
- EP89.DOC: 文本文件。

E.3 快速安装说明

1. 电脑关机。
2. 连接打印机信号线至 PC 打印机端口，另一端连接 J8 脚座，红线标示为 PIN 1。
3. 烧录器接上+12V 直流电源供给器，则 D1 电源指示灯亮。
4. 打开电脑电源。
5. 安装系统烧录模拟程序。
6. 执行 EPM89.EXE，则 D2 工作指示灯闪动。
7. 系统会自动读取测试文件 IR.ROM 其核对代码为 8CA6H(SUM0)。
8. 按空格键可以自动烧录并读取其内容，核对代码应仍为 8CA6H(SUMS)，表示烧录成功。

E.4 注 意 事 项

1. 连接打印机的信号线请勿自行加长，否则会有烧录不稳定的现象发生。
2. 本烧录器的打印机端口使用 LPT1，烧录电压为 12V。
3. U0 及 U2 的 IC 座上不可同时放上 IC。
4. 为了方便测试及烧录，可以自行在 U0 及 U2 的 IC 座上加上测试座。
5. 20 PIN 及 40 PIN 测试座在加入时，请自行在 U0 及 U2 的 IC 座上再套上另一 IC 座。
6. 本烧录模拟的工作是以 89C51(4K)、89C52(8K)、89C55(20K)芯片来模拟 8051 或 89C1051(1K)、89C2051(2K)单片机。
7. 放入想烧录的 IC 时请注意 PIN 1 位置。
8. JP1 为+5V VCC 输出开关 ON 时，从烧录器上送出+5V 到 8051 目标板上，当 8051 目标板上负载过大时，或烧录不稳定时，请将其置 OFF，自行在 8051 目标板上加上 5V 电源。

E.5 快速烧录模拟说明

执行 EPM89 ?，系统会告知线上的快速烧录模拟说明：

```

-----
EPM89 (89C51/C52/C55) V1.1 97.12.25
Copyright (C) VICTOR uP LAB. 1997,1998
TEL : 07-2260258 URL: wwwhome.fancy.com.tw/~ufvic
-----

```

```

help      : EPM89 ?
direct EMU : EPM89 test.rom d
Select 89c51: EPM89 test.rom d 1 --> 4096 BYTES
Select 89c52: EPM89 test.rom d 2 --> 8192 BYTES
Select 89c55: EPM89 test.rom d 5 --> 20480 BYTES

```

- EPM89 ?：线上烧录模拟说明。
- EPM89 test.rom d：快速载入文件、烧录、直接模拟 89C51。
- EPM89 test.rom d 1：快速载入文件、烧录、直接模拟 89C51。
- EPM89 test.rom d 2：快速载入文件、烧录、直接模拟 89C52。
- EPM89 test.rom d 5：快速载入文件、烧录、直接模拟 89C55。

执行 EP2051 ?，系统会告知线上的快速烧录模拟说明。

```

-----
EP2051.EXE (89C1051/2051) writer V1.0 97.12.25
Copyright (C) VICTOR uP LAB. 1997,1998
TEL: 07-2260258 URL: wwwhome.fancy.com.tw/~ufvic
-----

```

```

help      : EP2051 ?
direct EMU : EP2051 test.rom d
Select 89c1051 : EP2051 test.rom d 1 --> 1024 BYTES
Select 89c2051 : EP2051 test.rom d 2 --> 2048 BYTES

```

- EP2051 ?：线上烧录说明。
- EP2051 test.rom d：快速载入文件、烧录 89C2051。
- EP2051 test.rom d 1：快速载入文件、烧录 89C1051。
- EP2051 test.rom d 2：快速载入文件、烧录 89C2051。

E.6 模拟 40/20PIN 8051 单片机

1. 连接 20 PIN 或 40 PIN 模拟信号线至开发中的目标板上的 IC 座上，20 PIN 连接 J4 脚座，红线标示为 PIN 1。

40 PIN 连接 J3 脚座, 红线标示为 PIN 1。

2. 若 8051 目标板的+5V 耗电不大, 可以将 JP1 短路, 从烧录器上直接供应+5V 电压。
3. 若 8051 目标板的+5V 耗电过大, 请务必将 JP1 置 OFF, 以免影响烧录器正常工作。
4. EPM89 test.rom d: 快速载入文件、烧录、直接模拟 89C51。

E.7 完整操作说明

- EPM89.EXE: 89C51、89C52、89C55 烧录模拟程序 89C1051、89C2051 模拟程序
- EP2051.EXE: 89C1051、89C2051 烧录程序执行 EPM89.EXE, 显示如下工作画面:

```
-----
EPM89 (89C51/C52/C55) V1.1  97.12.25
Copyright (C) VICTOR  uP LAB.  1997,1998
TEL: 07-2260258  URL: wwwhome.fancy.com.tw/~ufvic
-----
```

```
quick EMU  :      EPM89 T.TSK D
Printer port I/O =378H  379H  37AH
Program voltage: 12V
Chip  NO: 89C51      SIZE: 4096 BYTES
file  name: t0.rom    file size: 397  bytes
file  check sum: SUMS=0000H  SUM0=8CA6H
-----
```

```
NO EMULATING  ....
-----
```

```
t --> Type set      g --> Read sign
b --> Blank  check  e --> Erase Chip
R --> Read CHIP all  r --> Read CHIP x bytes to buffer
p --> Program x bytes  SPACE --> auto
1 2  3 --> lock bit  h --> Help
o --> Dir *.tsk      d --> DUMP buffer data
s --> Save code      l --> Load code
x --> Emulate/No Emulate  ESC  --> exit
Select ?
```

功能键说明如下:

- Esc: 结束程序执行。
- t: 烧录芯片编号选择 89C51/89C52/89C55。
- g: 读取 IC 形态识别代码。
- b: 芯片空白检查。

- e: 将芯片擦除为空白。
- r: 依据“file size”长度由芯片读取数据至内存。
- R: 依据“SIZE”长度读取芯片全部数据至内存。
- p: 依据“file size”长度烧录数据至芯片。
- 空格键: 自动将芯片擦除为空白, 并依据“file size”长度烧录数据至芯片, 并检查核对代码。
- 1: 烧录保密位 1。
- 2: 烧录保密位 2。
- 3: 烧录保密位 3。
- h: 线上说明文件。
- o: 显示当前工作目录中的可执行文件*.tsk 文件名。
- d: 显示内存内的数据。
- s: 将内存内的数据保存为文件。
- l: 载入文件至内存内准备烧录。
- x: 系统发出 RESET 信号, 使程序开始执行, 再按一下则停止程序执行。

执行 EP2051.EXE, 显示如下工作画面:

```

-----
EP2051.EXE (89C1051/2051) writer V1.0 97.12.25
Copyright (C) VICTOR uP LAB. 1997,1998
TEL : 07-2260258 URL: wwwhome.fancy.com.tw/~ufvic
-----

quick EMU : EP2051 T.TSK D
Printer port1 I/O =378H 379H 37AH
Program voltage: 12V
Chip NO: 89C2051 SIZE: 2048 BYTES
file name : t0.rom file size: 397 bytes
file check sum: SUMS=0000H SUM0=8CA6H
-----

t --> Type set          g --> Read sign
b --> Blank check       e --> Erase Chip
R --> Read CHIP all     r --> Read CHIP x bytes to buffer
p --> Program x bytes   SPACE --> auto
1 2 --> lock bit        h --> Help
o --> Dir *.tsk         d --> DUMP buffer data
s --> Save code         l --> Load code
ESC --> exit
Select ?

```

功能键说明如下:

- Esc: 结束程序执行。
- t: 烧录芯片编号选择 89C1051/89C2051。
- g: 读取 IC 形态识别代码。
- b: 芯片空白检查。
- e: 将芯片擦除为空白。
- r: 依据“file size”长度从芯片读取数据到内存。
- R: 依据“SIZE”长度读取芯片全部数据到内存。
- p: 依据“file size”长度烧录数据至芯片。
- 空格键: 自动将芯片擦除为空白, 并依据“file size”长度烧录数据至芯片, 并检查核对代码。
- 1: 烧录保密位 1。
- 2: 烧录保密位 2。
- h: 线上说明文件。
- o: 显示当前工作目录中的可执行文件 *.tsk 文件名。
- d: 显示内存内的数据。
- s: 将内存内的数据保存为文件。
- l: 载入文件至内存内准备烧录。

E.8 打印机 I/O 示范程序

本烧录器由 PC 打印机端口连接, 除了做 89CXX 系列芯片烧录外, 还可以做一般打印机并行口实验, 提供 16 BIT 输出, 8 BIT 输入, 并含有 TURBO C 的控制示范程序。

相关文件说明如下:

- LPD.PRJ: 打印机 I/O 示范程序计划文件;
- LPD.EXE: 打印机 I/O 示范程序可执行文件;
- LPD.C: 打印机 I/O 示范程序;
- LPIO.OBJ: 打印机 I/O 驱动程序。

打印机并行口实验由 J9 及 J10 脚座连接出来:

J9 脚座: OP0 O0~O7 8 BIT 输出 IN I0~I7 8 BIT 输入;

J10 脚座: OP1 O0~O7 8 BIT 输出;

控制板上 U0 及 U2 不可以插上烧录的 IC;

U3~U8 必须放置 74LS374;

U1 必须放置 74LS157。

LPD.EXE 为打印机 I/O 示范程序可执行文件, 执行后, 显示如下工作画面:


```
-----  
EP89 PCB LPT PORT I/O demo V1.0 98.2.7  
Copyright (C) VICTOR uP LAB. 1997,1998  
TEL: 07-2260258 URL: wwwhome.fancy.com.tw/~ufvic  
-----
```

```
Printer port1 I/O ≈378H 379H 37AH
```

```
1 --> test O/P port 0  
2 --> test O/P port 1  
3 --> test I/P port  
4 --> test loop port0 -->I/P  
5 --> test loop port1 -->I/P  
ESC --> back to DOS  
Select ?
```

功能键说明如下:

- 1: 持续由 OP0 送出方波信号。
- 2: 持续由 OP1 送出方波信号, D2 LED 将会快速闪动。
- 3: 循环由 IN 输入端口读取数据。
- 4: 由 OP0 送出 0~255 数值, 由 IN 输入端口读取数据测试。
- 5: 由 OP1 送出 0~255 数值, 由 IN 输入端口读取数据测试。

打印机 I/O TURBO C 驱动程序:

- find_port();
寻找打印机端口 LPT1 的 I/O 地址。
- init_port();
初始化打印机端口状态。
- opp0(unsigned char x);
由输出端口 OP0 输出一字节数据。
- opp1(unsigned char x);
由输出端口 OP1 输出一字节数据。
- unsigned char in();
由输入端口 IN 输入一字节数据。

[G e n e r a l I n f o r m a t i o n]

书名 = 8 0 5 1 单片机C语言控制与应用

作者 = B E X P

页数 = 3 6 1

下载位置 = h t t p : / / b o o k 1 . s s r e a d e
r . c o m / d i s k j s j / a 0 0 8 / 1 6 / ! 0 0
0 0 1 . p d g

目录
正文