8051 单片机技术应用系列

光盘内含书中范例源程序

8051 单片机

彻底研究 经验篇

林伸茂 编著

中国电力出版社
www.infopower.com.cn

8051 单片机技术应用系列

8051 单片机彻底研究 入门篇
8051 单片机彻底研究 基础篇
8051 单片机彻底研究 实习篇
8051 单片机彻底研究 经验篇

本书共分三大部分：第一部分是8051的深度应用篇，这里所讲的都是8051开发人员所必须具备的常识；第二部分是烧录器的深度研究篇，主要介绍常用的AT89C51烧录器、EPROM烧录器和E^2PROM烧录器的开发过程，而且这些程序源码完全公开。第三部分是作者的经验分享，详细地描述了许多设计项目的来龙去脉。

本书适合的对象是对单片机有些开发经验的读者，或是了解软硬件的工程师。

- 8051 汇编语言 vs C
- 8051 程序的逆向工程研究
- 8051 单片机的是是非非
- 8051 开发经验的分享
- 亲手制作 AT89C51 烧录器
- AT89C51 烧录器程序的修改
- EPROM 烧录器的 DIY 步骤
- Flash 编程器的后续开发
- 智能化电饭锅测试线设计

责任编辑：刘 炽
封面设计：杜长清

ISBN 978-7-5083-5153-7
9 787508 351537 >
定价：32.00 元（含 1CD）

8051单片机技术应用系列

8051单片机

彻底研究 经验篇

林伸茂 编著

中国电力出版社
www.infopower.com.cn

内 容 提 要

本书共分成三大部分：第一部分是应用研究篇，介绍8051单片机的诸多设计理念与硬件保护手段。第二部分是介绍以FLAG51为主体的烧录器，包括Atmel Flash Microcontroller的烧录、EPROM的烧录以及E^2PROM的烧录等。第三部分是历年来作者接触到设计项目的心得以及对一些问题的看法。

本书属于8051进阶级书籍，适合对单片机已经有一些经验的读者，或了解软硬件开发的工程师阅读。

图书在版编目（CIP）数据

8051单片机彻底研究——经验篇／林伸茂编著. —北京：中国电力出版社，2007
（8051单片机技术应用系列）
ISBN 978-7-5083-5153-7

Ⅰ.8... Ⅱ.林... Ⅲ.单片微型计算机—基本知识Ⅳ.TP368.1
中国版本图书馆CIP数据核字（2007）第004573号

北京市版权局著作权合同登记号　图字：01-2006-5849号

版权声明

责任编辑：刘　炽
责任校对：崔燕菊
责任印制：李文志

丛 书 名：8051单片机技术应用系列
书　　名：8051单片机彻底研究——经验篇
编　　著：林伸茂
出版发行：中国电力出版社
　　地址：北京市三里河路6号　邮政编码：100044
　　电话：（010）68362602　传真：（010）68316497
印　　刷：北京丰源印刷厂
开本尺寸：185 × 260　　印 张：17　　字 数：408千字
书　　号：ISBN 978-7-5083-5153-7
版　　次：2007年5月北京第1版
印　　次：2007年5月第1次印刷
印　　数：0001—4000
定　　价：32.00元（含1CD）

8051 单片机学习地图

8051 单片机的应用到处都是，可是我一点基础都没有，要怎样开始学习呢?

8051单片机
彻底研究——入门篇

我对 8051 已经有了基本的认识，我想更进一步彻底学好 8051 汇编语言!

8051单片机
彻底研究——基础篇

要想学好 8051，练习是少不了的，我希望找一些有趣的题目自己动手做做看，不然怎样写都是一些简单的范例。

8051单片机
彻底研究——实习篇

基本功都练扎实了，想进一步提高水平，吸取前辈的经验是最好的办法。

8051单片机
彻底研究——经验篇

序　言

知识分享的喜悦

20年来我一直坐在研究与开发的位置上，曾经在风和日丽的天气下进入台电核三厂的控制室内，为的是协助找出发电机组跳闸的原因；曾经在寒风冷雨的冬季到新竹工研院，查看仪器连线为何经常出问题的原因；曾经带着工具箱与手册，到鲜血尚未清洗干净的外科手术室里，检查自己设计的电动手术台是否有足够的安全性和稳定性；也曾经参与电动环保汽车的部分研发工作。

不管这些项目的结果如何，都有一个共同点：它们都是 8051 单片机的应用范例。每一个项目都是8051 的深度应用。这也是10多年前我开始在《RUN!PC》杂志上写8051应用实例、陆陆续续大约写了三四十篇技术性文章的原因。与此同时，我个人的系统设计工作始终没有间断过。

20年来，我们碰到了许多技术上的瓶颈，很幸运我们解决了大部分，剩下的问题很可能不属于纯粹的技术问题了。那么这些宝贵的资料与经验永远放在旗威科技公司的文件柜中吗？我选择了出书这一方式，来把这些经验提供给广大读者朋友，虽然这样做很辛苦，但是套一句老话“喜欢做，甘愿受”。纵使外面的环境非常复杂，我个人还是认为值得这样做的。

千万不能只顾着吃鱼，也要学会打鱼。我个人觉得做任何研究或写程序应进一步掌握其实现过程。在本书中您会看到一件产品是被如何开发出来的，以及为何要这样处理，我们在书中都会有详细的说明。

我经常告诉别人：健康不能分享，金钱也不能分享，但是知识除了独享外，可以与别人分享。旗威科技公司很乐意邀请您一起来分享8051的专业知识，一起分享我们写的程序。

系列书籍介绍

《8051 单片机彻底研究——经验篇》作为 8051 单片机系列丛书的一部分，与《8051 单片机彻底研究——入门篇》《8051 单片机彻底研究——基础篇》、《8051 单片机彻底研究——实习篇》构成了一个系统的、有机的专业技术教程丛书。

如果您想彻底了解 8051 单片机与汇编语言的写法，请务必参考《8051 单片机彻底研究——基础篇》，书中会对 8051 各个指令的用法与寄存器的操作，做了深入详尽的交待。而《8051 单片机彻底研究——实习篇》主要强调在 8051 单片机系统扩展与集成应用两方面专业知识。

内容收录

本书的内容取材主要有以下三个方面：

历年来我们运用 8051 单片机的实例应用。旗威科技有限公司用 8051 单片机设计过 100 种以上的工业产品，产品范围涵盖医疗、精密测量、通信及自动化等专业领域。

取自《RUN!PC》杂志历年来所发表的技术与评论性文章，并删除部分不合时宜的内容，

最后又加上我们最近几年的设计心得。

网络上有关 8051 单片机应用与 C 语言的技术文献与报道，这方面我们深信外国人开放，许多资料在网站上唾手可得，不过只有您仔细消化后，才可以算是您自己的东西。

如何阅读本书

本书属于 8051 进阶级书籍，适合对单片机已经有一些经验的读者，或横跨软硬件的工程师阅读。我们希望您在阅读本书时，已经对 8051 的结构与程序有基本的认识，例如 SFR、bit addressable、SBUF 等等。

本书共分成三大部分：

第一部分是应用研究篇，介绍 8051 单片机的诸多设计理念与硬件保护手段。想要以子之矛攻子之盾，在这里有第一手的解答。我们也提到与硬件保护始终对立的逆向工程与 8051 之间的关系。最后还有一章专门提到 8051 的优势与缺憾。8051 当初被人称道的功能至今被贬为重大瑕疵，为何如此？请看我们的深入分析。

第二部分是介绍以 FLAG51 为主体的烧录器，包括 Atmel Flash Microcontroller 的烧录、EPROM 的烧录以及 E^2PROM 的烧录等，我们很乐意地公布这些烧录程序，而且是 100%的源代码公布。以前我们彻底研究别人的程序为的是理解当初设计者的理念。今天我们公布程序内容，欢迎您来研究我们写的程序，如果您是用心的读者绝对可以从中获得更多的启示。

第三部分是历年来我们接触到设计项目的心得以及我们对一些问题的看法。

致谢

编写 8051 单片机系列书绝对不是单一个人所能完成的，它绝对是一个团队的工作总整合，3 年前我就开始筹备新书的出版事宜，所有的文章与内容经过整理过滤与调整补充。我要特别感谢以下帮助我的人们：

王圣心小姐与姜莹贞小姐：初步整理已发表过的文章，光是校稿就校了多次，并拍摄许多额外的照片，让本系列的书籍得以完成初步的架构。

李浩蓁先生与曾琼惠小姐：进行本书版面调整与最后的校稿，整本书是在他们的手中完成的。

罗仕林、庄昱宏与黄芳川先生：提供最高级的示波器与逻辑分析仪，以及技术上的协助，让本书的图表资料与数据更有看头。

凌全伯先生：提供了对 8051 另类的观点，在本书第 6 章“8051 的是是非非”中，他也加入了独到的见解，面对 8 位 CPU 就要有 8 位的思维。

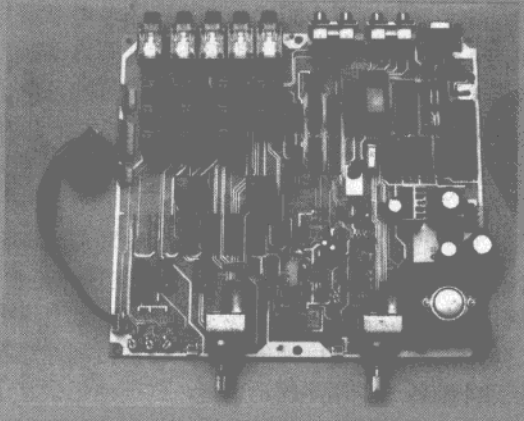
最后，我还是要谢谢家人所给予的鼓励，尤其是刚在牙牙学语的小女儿，没有他们几近狂热的激励与支持，就不会有本系列丛书的问世。

林伸茂

chipware@chipware.com.tw

目　录

1



KTV 使用的点歌与视频控制器，含红外线摇控及音量控制，右侧上方为 8052 单片机。

第1章　8051的汇编语言与C语言

有些程序设计人员对汇编语言说“恨之入骨”也不为过，习惯用C语言来设计系统。其实C语言和汇编语言各有其应用的空间，当你需要程序要精简且速度超快时，汇编语言绝对是首要之选。选择用C来开发就一帆风顺了吗?那也未必，欲知详情，请看本章彻底的分析。

最近，在BBS上的Electronic（即关于电子设备的）板上经常看到这样的讨论：8051的程序开发，是选用汇编语言较好，还是使用C语言较好?两种语言的支持者都引经据典并结合自己的实践经验为自己辩解。其实双方的说法都有道理。笔者在汇编语言与C语言的应用开发上都有超过10年以上的实践经验，愿意把自己从书本上看不到的体验与您分享。本章最终的目的并不是断定哪种语言是最好的，但是会引导读者根据自身情况以及开发的价格选用最适合自己的8051开发语言。

1-1　汇编语言短小精干

从学生时代开始我就使用汇编语言来写硬件的控制程序，最初是LED的灯号控制及七段显示器的数值显示，到多轴步进电动机的实时控制，都是借用强而有力的汇编语言直接控制CPU的运行。早期设有PC（个人计算机）及Apple II微电脑的时代，我是使用所谓的微电脑学习机来学习汇编语言的。首先我把汇编程序写在记事本的右方，然后由微电脑的DATABOOK上查找该指令的机器码（MACHINE CODE），将此机器码写在记事本的原始程序左侧，再将此机器码以十六进制码的方式一个接一个地输入到微电脑学习机上，最后一个操作是按下学习机上的“GO”键开始执行我们先前辛苦输入的程序。以上所有的操作都是手动的，除错的步骤当然也是纯手动的。那个时代有许多好的参考书籍与资料，反倒是现在资料变少了，想自己动手做的人也成了稀有品种。程序不幸若有错误时，所有的步骤都要重来一次，这也促使我们养成每个步骤都做双重核对的习惯，而且把每个子程序都模块化，以便其他场合也可派上用场，因为写汇编语言是相当艰苦的，有些程序上的臭虫（bug）要几星期的奋斗才可以确认出来，有些不应该出错的地方竟然出错了，这些苦头只有曾经亲自尝试过的人才有可能体会。

个人计算机流行之后，写程序就没这么累了，程序的编写与编译都可以在计算机上做，剩下的操作就是将程序下传给待测的控制板，然后确认所有的操作是否都如同原先所设置的。每次打印汇编程序的报表时，数据长度超过20页是很正常的，还有许多程序都是数百页以上的规模。我们的经验是汇编语言的程序长度加倍时，除错的时间绝对超过原来的两倍。开发这些程序的同时，我都会另外准备一本厚厚的记事簿，详细地记载整个开发的过程以及修改的地方，当然也包括了所有的测试步骤及方式。在原始程序SOURCE PROGRAM上我们也做了详尽的操作说明。

有人做过一个统计：开发汇编语言程序时，每 20 行大约需要 1 man-day 的成本；这也就是说：一个有 1000 行的汇编程序，若交由一个有经验的工程师来处理，至少要花上 50 天的时间才能完成。一个人一天只能完成 20 行的程序，好像有点不可思议!但是依我们多年的经验来看，这个说法在写汇编语言的场合中确实是对的，程序越大时这个说法越有其正确性。

由于汇编语言是直接控制系统的所有输入/输出点及存储器，所以刚开始写程序时，一定常会碰到系统突然死在某个循环中，无法跳出该循环的状态，最后只好采用最直接且最有效的方法——关机重来。写汇编语言对程序设计师来讲，是有相当的杀伤力及折磨性的，每次新的程序开始时，所有的程序小环节都要逐一测试与验证，所以压力是一定有的，完成一个程序后就好像历经一次母亲怀胎到生产的过程一样。写了多个大的汇编程序后，若无法适度调整心态及善用辅助的开发工具，很可能从此就永远不碰汇编程序了。真的，何苦来哉!

1-2 C 语言可以缩短开发时间

在 C 语言成功地由 UNIX 移转到 PC 后，写汇编语言的梦魇终于解除了，接着我们也可以看到适合各种 CPU 的 C 语言开发系统，其价格从几百元到上万元都有，这些开发软件工具的广告可在国外知名的专业杂志上看到，当然国内的 CPU 在线仿真器（In Circuit Emulator）开发厂商也会依国内用户的要求进口部分产品，但是数量不多，而且这些厂商对这方面的专业知识也是严重缺乏，所以凡事只有靠自己了。

我们先学习的是 PC 上的 C 语言程序（Microsoft C 及 Turbo C），这段学习时间至少持续一年以上，再进而购入 2500AD 的 8051C 语言编译程序，将我们原先的汇编程序改用 C 程序来处理，这时我们才发觉一个简单的控制程序若用 C 语言来撰写，要限时在 7 天以内完成是非常有可能的!举一个简单的例子来说：若要让 8051 的 P1 端口产生类似指示灯的 LED 亮法，用汇编语言的写法大概是这样的：

程序 1　用汇编语言写成的指示灯程序

```
                    ORG     0000H
START   NOV         R1,#00H             ;加入这段小 DELAY
$1                  DJNZ    R1,$1       ;等待系统稳定后再开始
                    MOV     SP,#60H     ;STACK 值的设置
LOOP                MOV     A,#01H
LOOP_NMOV           P1,A
                    CALL    DELAY
                    RL      A
                    CJNE    A,#80H,LOOP_N
                    SJMP    LOOP
;
;延迟一小段时间，以便可以观看出其变化
DELAY   MOV         R0,#00H
$1                  MOV     R1,#00H
$2                  DJNZ    R1,$2
                    DJNZ    R0,$1
                    RET
```

若改用 2500AD 的 C 语言来写时，可就轻松多了。看起来简单明了，何时看程序都是一清二楚的。唯一的缺憾是用 C 转换出来的程序代码过大，下面这个程序链接了链接库后，最

后的程序空间可能会超过 600 字节。

程序 2　改用 C 来处理的指示灯程序

```
#include"c8051io.h"
#include"c8051sr.h"
main()
{
car out;
     out=0x01;
     while(1)
     {
     P1=out;
     out=out<<1;/*shift left*/
     delay();
     if(out= =0x80)out=0x01;
     }
}

int delay()
{
int m;
   for(m=0;m<1000;m++){};/*just delay,do nothing*/
}
```

以下就是上述 C 语言程序经过转译成 8051 汇编语言的程序，由于篇幅的限制，我们仅列出程序重要的部分，其余的声明部分都暂时删除，这是由 2500AD 公司的 8051 C 编译程序转译的结果，改用其他编译器时结果可能不会相同。

```
32  0000                    _main:          .equal$
33                          ;
34                          ;
35  0000                    ?ASC0:          .equal0
36  0000                    ?TSC0:          .equal0
37  0001                    ?LSC0:          .equal1
38  0000        C2 AF                       clr     ea
39  0002        EE                          mov     a,r6
40  0003        24 FF                       add     a,#.low.-?TSC0-?LSC0
41  0005        FE                          mov     r6,a
42  0006        EF                          mov     a,r7
43  0007        34 FF                       addc    a,#.high.-?TSC0-?LSC0
44  0009        FF                          mov     r7,a
45  000A        D2 AF                       setb    ea
46                          ;out=0x01
47  000C        90 00 00                    mov     dptr,#-1+?LSC0
48  000F        74 01                       mov     a,#1
49  0011        12 00 00                    lcall   st_oper1_stack_char
50                          ;while
51  0014                    ?TOL0:          .equal $
52  ;(1)
53  ;
54  0014        79 00                       mov     r1,#0
```

```
 55  0016        74 01                          mov     a,#1
 56  0018        C3                             clr     c
 57  0019        99                             subb    a,r1
 58  001A        70 03       jnz                +3
 59  001C        02 00 55                       ljmp    ?BOL0
 60  001F                    ?PLC0:             .equal $
 61                          ;P1=out
 62  001F        90 00 00                       mov     dptr,#-1+?LSC0
 63  0022        12 00 00                       lcall   __ld_oper1_stack_char
 64  0025        F5 00                          mov     .low._P1,a
 65                          ;out=out<<1
 66  0027        79 01                          mov     r1,#1
 67  0029        90 00 00                       mov     dptr,#-1+?LSC0
 68  002C        12 00 00                       lcall   __ld_oper1_stack_char
 69  002F        12 00 00                       lcall       __sl_char
 70  0032        90 00 00                       mov     dptr,#-1+?LSC0
 71  0035        12 00 00                       lcall   __st_oper1_stack_char
 72                          ;delay()
 73  0038        12 00 62                       lcall_delay
 74                          ;(out==0x80)
 75  003B        79 80              mov         r1,#128
 76  003D        90 00 00                       mov     dptr,#-1+?LSC0
 77  0040        12 00 00                       lcall   __ld_oper1_stack_char
 78  0043        C3                             clrc
 79  0044        99                             subba,r1
 80  0045        60 03              jz          +3
 81  0047        02 00 52                       ljmp    ?BOL1
 82  004A                    ?PLC1:  .equal     $
 83                          ;out=0x01
 84  004A        90 00 00                       mov     dptr,#-1+?LSC0
 85  004D        74 01              mov         a,#1
 86  004F        12 00 00                       lcall   __st_oper1_stack_char
 87  ;;
 88                          ;
 89                          ;
 90  0052                    ?BOL1:  .equal $
 91  0052        02 00 14           ljmp    ?TOL0
 92  0055                    ?BOL0:  .equal $
 93                          ;
 94  0055                    ?BOF0:  .equal $
 95  0055        C2 AF              clr ea
 96  0057        EE                             mov     a,r6
 97  0058        24 01              add         a,#.low.?LSC0+?TSC0+0
 98  005A        FE                             mov     r6,a
 99  005B        EF                             mov     a,r7
100  005C        34 00              add         ca,#.high.?LSC0+?TSC0+0
101  005E        FF                             mov     r7,a
102  005F        D2 AF                          setb    ea
103  0061        22                             ret
104                         ;
```

```
105              0062        _delay:.equal $
106                          ;
107
108              0000        ?ASC1:      .equal 0
109              0000        ?TSC1:      .equal 0
110              0002        ?LSC1:      .equal 2
111 0062         C2 AF               clr ea
112 0064         EE                      mov     a,r6
113 0065         24 FE               add a,#.low.-?TSC1-?LSC1
114 0067         FE                      mov     r6,a
115 0068         EF                      mov     a,r7
116 0069         34 FF               addc    a,#.high.-?TSC1-?LSC1
117 006B         FF                      mov     r7,a
118 006C         D2 AF               setb    ea
119                          ;m=0;
120 006E         90 00 00            mov     dptr,#-2+?LSC1
121 0071         78 00               mov r0,#.low.0
122 0073         7A 00               mov r2,#.high.0
123 0075         12 00 00                lcall_st_oper1_stack_int
124 0078                     ?FLC2:  .equal $
125                          ;m<1000;
126 0078         79 E8               mov r1,#.low.1000
127 007A         7B 03               mov r3,#.high.1000
128 007C         90 00 00                mov     dptr,#-2+?LSC1
129 007F         12 00 00                lcall   _ld_oper1_stack_int
130 0082         12 00 00                lcall       __cmp_int
131 0085         30 E7 06                jnb     acc.7,+6
132 0088         30 D2 09                jnb     ov,+9
133 008B         02 00 B1                    ljmp        ?BOL2
134 008E         20 D2 03                jb      ov,+3
135 0091         02 00 B1                    ljmp        ?BOL2
136 0094         02 00 AE                    ljmp        ?FLB2
137 0097                     ?FLA2:      .equal $
138 ;m++)
139 0097         90 00 00                mov     dptr,#-2+?LSC1
140 009A         12 00 00                lcall   __ld_oper1_stack_int
141 009D         E8                      mov     a,r0
142 009E         24 01               add     a,#1
143 00A0         F8                      mov     r0,a
144 00A1         EA                      mov     a,r2
145 00A2         34 00               addc    a,#0
146 00A4         FA                      mov     r2,a
147 00A5         90 00 00                mov     dptr,#-2+?LSC1
148 00A8         12 00 00                lcall   __st_oper1_stack_int
149 00AB         02 00 78                    ljmp        ?FLC2
150 00AE                     ?FLB2:      .equal $
151                          ;
152 00AE         02 00 97                    ljmp        ?FLA2
153 00B1                     ?BOL2:      .equal $
```

```
154                             ;
155 00B1                        ?BOF1:      .equal $
156 00B1        C2 AF                       clr ea
157 00B3        EE                              mov     a,r6
158 00B4        24 02                       add a,#.low.?LSC1+?TSC1+0
159 00B6        FE                              mov     r6,a
160 00B7        EF                              mov     a,r7
161 00B8        34 00                       add ca,#.high.?LSC1+?TSC1+0
162 00BA        FF                              mov     r7,a
163 00BB        D2 AF                       setb    ea
164 00BD        22                                      ret
```

1-3 8051 编译器使用经验谈

写 8051 的汇编程序时是完全无法取巧的。如果刚开始学习 8051 的程序语言，我们强烈建议由汇编语言开始学起，因为只有这样才能促使学习者完全理解整个 8051 CPU 动作的来龙去脉。不会设计线路也不要紧，因为市面上 8051 书籍多得很，只要参考几种现成的线路后，就可以依样照做了，可是汇编程序的撰写，除了自己以外，别人都无法帮得上!我们的经验是别人的程序看得越多，你的程序架构就会写得越好。如果有好的写法或子程序也可以在自己充分地吸收后归为己有。如果身旁收集的工具程序很多，程序开发的时间会相对地减少许多，不需要每次都从头开始除错。

市面上的 8051 参考书籍适合初学者的居多，可能无法找到相当理想的参考程序，我们建议可以到旗威科技公司的网站（http://www.chipware.com.tw）上获取以下几个相当有参考价值的参考程序：

`readtemp.asm` 温度的测量显示与联机串行通信程序。
`flagdisp.asm` 七段显示器的控制程序。

以上程序除了包含有许多常用的汇编语言子程序外，都有使用定时中断及串行通信等等的使用范例，必须逐一推敲所有的程序，才能对控制系统有更进一步的理解与认识。这些学习的经历必须自己走过才算是把 8051 给带进门来。也可以把许多 8051 应用产品的程序 EPROM 拔下来，用 EPROM 烧录器将程序数据转存在 PC 机中，然后用 8051 的反汇编程序 DIS51.EXE（请自行到旗威科技公司的网站中下载此程序）将原先的机器码翻译成汇编语言的格式，再来分析其程序的写法与控制模式。我们所认识的汇编语言高手中，有不少人是用这种方式去磨练的，当然也包括我个人在内。这种学习方法最辛苦，但是收获却是最大的。我曾经用最原始的方法读过多个 Z80、6502 及 8051 满载 64KB 的系统程序，所以现在碰到类似的系统时，都可以在最短的时间内了解问题所在。至于旗威科技公司开发的其他 8051 应用程序，请不要全面予以反汇编，因为这些程序有大多数是用 8051 的 C 语言来处理的，程序规模相当大，全部通过堆栈传递参数，没有相当的程序基础是完全无法了解的。

我们另一方面也建议：如果有多余预算的话，不一定非得买 ICE 不可，但是买一台较像样的示波器，绝对可以在 8051 程序的开发过中帮上大忙。以前我们只用一台 20MHz 的 Hitachi（日立）示波器，再通过汇编程序中的无穷循环，就可以对硬件线路及系统程序进行全面性的除错。近几年来我们发现若用数字存储示波器 DSO（Digital Storiage Oscilloscope）来除错更是方便，尤其是串行通信及 AD 转换的程序除错，至少可以减少一半以上的时间。

当我们全面改用 C 语言来处理 8051 的控制程序时，整个开发时间明显地减少了许多，像我们近几年来陆续开发的 8051 应用控制板：FLAG51 单片机控制板、Chipware-PLC 控制程序、AT89CXX 烧录板、27256/27512 EPROM 烧录板以及 TH2030 温湿度控制板都是以 C 语言为主体的，而且许多程序是先在 PC 机上以 Turbo C 验证无误后，才试着移植到 8051 上。

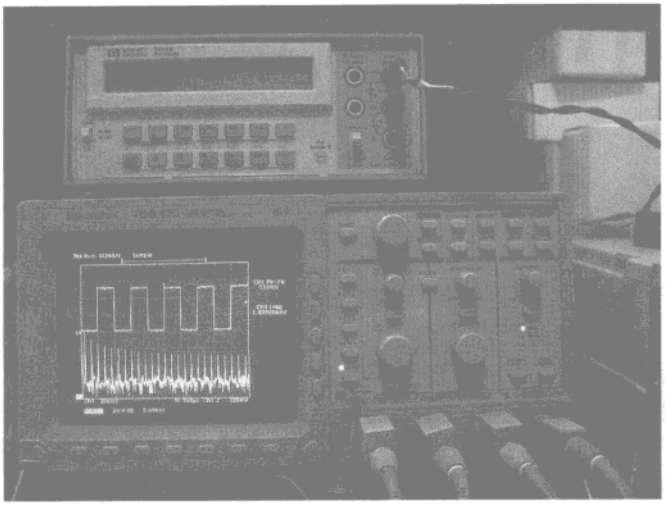


图 1-1　对 8051 的开发者而言，硬件工具类最重要的是数字电表与示波器

使用 C 语言来控制 8051 的最大好处是程序的流向非常清楚，不论何时去重新看程序都能迅速进入状态。不过，C 语言还是有程序空间过大（至少占汇编语言的 10 倍以上的程序空间）及执行速度缓慢等的缺点存在。汇编语言的执行及反应速度约在数十个微秒以内，而 C 的控制程序就要多花点时间来由堆栈存取各个参数值，而其执行速度约在数个毫秒左右。假设，只是想控制几个电器的开关或是接受信号后控制电动机或油压缸，选择 C 语言来处理是绰绰有余的。如果你的 8051 应用程序要求的速度很快，例如处理串行通信或读取定位用光学编码器（Encoder）的时候，最好还是用汇编语言来做才最恰当。

我们的 FLAG51 控制板的监控程序中，虽然主体架构是以 C 来处理的，也有一小段汇编程序 login.s01，用来处理暂时脱离 C 语言的控制程序，然后进入用户的应用程序，最后又返回 C 的控制程序。处理类似 C 与汇编语言程序前，一定要把原版的手册从头到尾看过多次之后，才试着去做做看。因为以 C 语言来处理时，系统会多添加一个未定数，我们必须先行判定是自己程序的问题还是 C 处理程序的问题？这方面的除错时间是相当费时的。我们曾经在某个 C 程序上多加了一行指令后，系统就全面停止，追查了好几天都没有结果，最后才由逻辑分析仪上看到：8051 正在向一个未经声明的 RAM 进行数据的存取，这当然会出问题的。最后还是加大 SRAM 区后才解决该问题。或许有人会问，程序空间或数据空间不够时，难道 C 不会提出警告吗？答案是不会的，这些问题一定要开启电源执行时，才会发现情况不对，接下来又是一连串痛苦的除错过程，在程序当中加入各种除错码，以便分析程序在哪里迷失了该走的路，这时我们就会叹息地说：还是用汇编语言比较容易控制，间接地除错点也会较容易设置。

1-4 C 语言是全能的吗？

C 语言确实好用，某些指令经过编译后，其精简的程度与我们用汇编语言写的相当，甚至更具有模块性。很幸运地，过去数年来我们曾经使用过市面上最常见的三种 8051 C 编译程序：2500 ADSOFTWARE、IAR C 及 Franklin C 等，也完成至少 20 个以上 8051 的控制应用，但是我们发现 C 语言好用但未必是全能的，以下就是我们使用 C 语言后所获得的心得。

（1）所有牵涉到精确定时的子程序中，用汇编语言会有更快的反应速度。

（2）所有的 8051 C 语言编译程序都会建议用户小心使用汇编语言，以免造成死机。

（3）C 语言编译时可以指定对执行速度或是程序代码进行优化，我们必须加以了解及使用。

（4）对于重启之后的 C 启动程序 STARTUP，请务必进行所有操作的确认。

（5）如果发觉某些子程序的执行速度不够快时，先试着用 C 进行改写，再不行就要用汇编语言。

（6）大部分 C 语言的优化仅仅是针对该语言而已，并未对 8051 的程序代码进行优化，若我们将 CPU 改为 Dallas 的高速 CPU 80C320，则仅有原来速度的 4 倍而已，如果还要更快时，就只有改用汇编语言了。若改用汇编语言后仍不够快时，我们建议这部分直接用硬件来处理，这才是治标兼治本之道。

（7）C 语言处理中断的速度并不逊于汇编语言，但是使用前须了解其进入及返回原程序的步骤，以免造成意外的死机。

（8）随时观察系统的输出 MAP 文件，以确认各个变量的存放位置，是否真的放进 SRAM 区内部。

（9）C 语言所写的控制系统比汇编语言而言，有更多不确定性存在，这些不确定性包含了我们程序写法的执行错误（RUNTIME ERROR），以及编译程序的隐藏性 bug，所以我们要有一套更严谨的程序验证步骤，对程序代码进行全面检查。

（10）汇编语言及 C 语言真的无法断定孰优孰劣，但是只要选对使用时机，你就是 8051 的赢家。

本章使用的软件

（1）2500AD C 与 Assembler。

（2）IAR C 与 Assembler。

（3）DIS51.EXE：由旗威科技公司开发的 8051 反汇编程序。

本章使用的硬件

（1）AT89C2051 学习板。

（2）Dallas 80C320：8051 兼容 CPU 但速度为 8051 的 4 倍。

（3）HP3478：五位半数字电表。

（4）TEK420：100MHz 全数字储存式示波器。

相关资料网站

可经由下列公司、网站获取更进一步的信息：

http://www.chipware.com.tw：获取 8051 单片机相关资料。

http://www.iar.com：获取 C 与汇编语言相关资料。

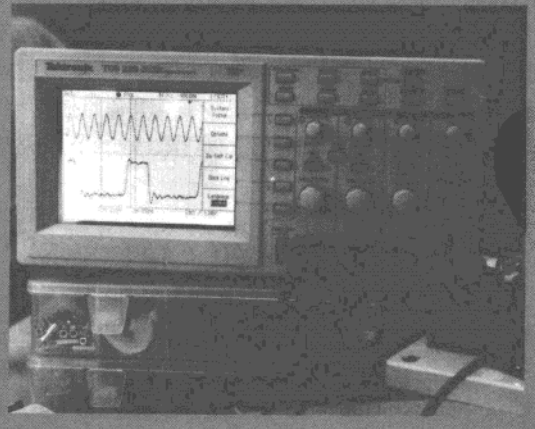
http://www.keil.com：获取 C 与汇编语言相关资料。

http://www.franklin.com：获取 C 与汇编语言相关资料。

http://www.tek.com：获取高级示波器相关资料。

http://www.agilent.com：获取仪器相关资料。

2



如果要从事单片机设计，首先要买的是数字电表，接下来的就是数字式示波器。

第 2 章　单片机保护程序的方法

这一章里我们不谈程序设计，反过来要与各位读者分享如何保护您的程序。

本章将介绍数种保护程序的方法与手段，但是，“道高一尺魔高一丈”，几乎所有的程序保护只能延后竞争者赶上的时间，却无法做到百分之百的防范，为何如此请看我们的深入分析。

2-1　案例一：喷气发动机的源代码

根据报纸的报道，某种型号飞机的发动机推力明显不足，原因是该发动机的控制系统完全是单片机控制的，自原厂出厂时就将发动机的推力限制在某个范围内，两个完全相同的喷气发动机推力可能相差 50%之多。记者接着说，如果我们能取得该发动机的控制源代码，则可以自行提高发动机推动力，但是该源代码应属公司的最高机密，原制造厂以事关国防安全为由宣称不能卖出。类似飞机发动机这样高精密及高稳定度的控制系统中，绝对会在其使用及维修手册上非常清楚地注明：“请勿更换或尝试更改本系统的任何一个零件，如有此行为而造成任何损失或灾害，制造公司恕不负责”。即使我们“间接”取得源代码，也要彻底知道该程序是如何保护的，否则随意修改一行代码都会使发动机完全不能启动，这是任何一位开发工程师绝对无法承担的责任。

2-2　案例二：苹果二号与桔子二号

早期的计算机系统的保护性都不是很完备，造成只要找得到零件，就可以依样画葫芦“照着做”，于是“苹果二号”（Apple Ⅱ）问世之后，类似的仿冒品，如“桔子二号”（Orange Ⅱ）等等阿猫阿狗牌计算机都跟着上市，计算机业也是由此时开始蓬勃发展，从而造就了不少计算机界的顶级人物，可是接下来的个人计算机却很少有 100%照抄的事情发生（初期 8088 的机器不算），正式说法是人们已有了保护知识产权的意识，另一个原因却是软硬件保护方法得宜，使仿冒者无法得逞。

2-3　案例三：仿冒的“快打旋风”

几年前电玩“快打旋风”曾经流行了好一阵子，日本推出半个月后，其他版本的“快打旋风”也出现了，令人不得不佩服技术动员能力之快速，如果“快打旋风”不做任何保护，那结果又会怎样呢？我们百分之百不赞成仿冒，但是如果我们的程序中不加入任何保护措施时，是否能有效约束其他人照抄？如果答案是否定时，我们就必须懂得如何自我保护了。设计单片机的系统若少了危机意识，只要该产品一流行就难逃被抄袭的命运。

2-4 我们都是看别人的程序成长起来的

在规划及写程序历程上，我是先参考及研究现有的程序，从中吸收足够的经验后，才能开始尝试写自己的小程序，从早期的 EDU80、Apple II、PC 的 BASIC、Turbo C 到近几年的 8051 与 Windows 软件，无不是循着别人程序的轨迹而成长。学校课堂上只教我们如何“写对”一个程序，但是却漏掉了如何“写好”一个程序，至于如何做“程序自我保护”这方面也很少提及，但是在真正的应用上，这些却是基本的常识（Common Sense）。日本的技术杂志每隔二年就用“技术者的基本 Common Sense”为读者做一期技术专题报导，可是一般的杂志却很少（或是说根本没有）做这方面的报导。好久以前就想写一篇在教科书上不能提的程序保护文章了，以下正是我多年来自己写程序与看别人程序的经验谈，其中一些情节纯属虚构，若有雷同则为巧合。

2-5 保护方法一：数据 CHECKSUM 法

所有单片机控制器都有一个以上的程序 EPROM 或 Flash，通常在程序尚未正式操作前，我们会把整个 EPROM 的内容（32KB 或 64KB）全部加起来，最后得到一个 8 位或 16 位的 CHECKSUM 值，再把此值与默认值比较，若相同则继续执行程序，反之则进入程序数据错误的例程当中。如果别人故意修改了程序内容，一定会影响到校验和的值，该值相同的机率极低（1/65536），所以能保持程序的完整性，有些时候我们烧录的 EPROM 会无缘无故地数据消失，或者可能仅有几个位状态变了，也可以运用这个方法检查程序 ROM 内容的数据完整性，不仅能确认系统的状态，也能做基本的程序防范。

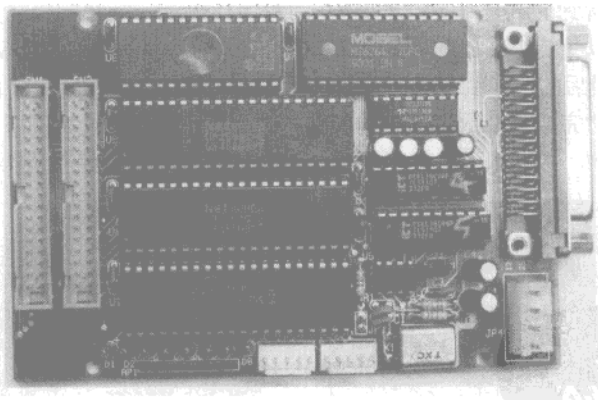


图 2-1 在 FLAG51 单片机控制板上可以进行保护的地方有：CPU、ROM、RAM 和 PEEL

2-6 保护方法二：SRAM 数据保护法

很多单片机系统的控制板中都有充电电池用于重要数据的保存，将数据存在耗电量极低的 SRAM 中，这类的 SRAM 的编号最后都有 LP（Low Power）的字样，除了储存重要数据外，通常还会留下部分的空间未使用，我们可以利用这些位置存放系统重要的口令，系统开

始运行前就检查这些值是否正确，不正确时系统立即停止，并且把整个 SRAM 清除成 00H，所有的重要数据完全不见了，让想修改的人拿不到所有的口令，当然也间接地保护了程序。采用这种保护方法时，我们要写另一个程序将所有的 Password 预先加载 SRAM 中，然后再把真正的程序 ROM 插到系统中，才能开始启动保护的作用。用充电电池做数据保存时要留意不要让电池没电了，或是电池两端的不正常短路，这两种情形都会导致系统保护程序的启动，机器就因此无法运行了，许多精密的控制仪器或设备都要求使用者不要拔掉电源接头，其主要的用意是让充电电池不间断持续地得到电能，以免充电电池真的没电之后，系统就完全停止了。

2-7 保护方法三：PAL/GAL/PEEL 保护法

PAL/GAL/PEEL 这类的可编程元件最初是用来简化线路的，每个元件可以取代 2～10 个标准 TTL 的 IC，同时 PAL/GAL/PEEL 都提供一个保护位（Security Bit），只要把保护位设成 1 后，就无法读出其内部的状态值，刚好可防止别人的非法抄袭。最近几年可编程元件的使用率大增，而且大部分的万用烧录器（Universal Programmer）都可以进行数据状态的烧录，不少设计工程师对这类 IC 已相当熟悉，拿来作为保护的诱因已逐渐消退。PAL/GAL/PEEL 的保护若想做得更周全一点时，最好选用可以 ERASE 重复使用的 PEEL，并且将两个以上的 PEEL 串接在一起，以增加其电路逻辑状态的复杂度。在旗威科技公司的 FLAG51 单片机控制板上也有两个 PEEL，一个取代 74LS373，另一个负责系统的译码电路，若想做些保护措施时，第一个方法可以更改 ROM/RAM 及 I/O 的译码地址，第二个方法是修改取代 74LS373 的 PEEL 内的状态值，例如只把 ADO 与 AD7 对调，就会导致程序 ROM 内的数据大幅变动，若别人拿程序 ROM 直接去反汇编时，看到的就是乱七八糟#$@！的程序代码了。当我们仍在开发测试阶段时，先不要把数据对调，以免徒增自己的困扰，等到所有都就绪后，再用 C 或 Visual Basic 写一个数据转换的程序，把最后的二进制 TSK 文件转换成供烧录用的另一个新文件，烧录一个 EPROM 插进 FLAG51 中，才完成整个数据保护的手续，过程很复杂。不过如果您想到自己辛辛苦苦花了一个月所写的程序，被别人三分钟内就抄了过去，这点手续是绝对值得去做的。

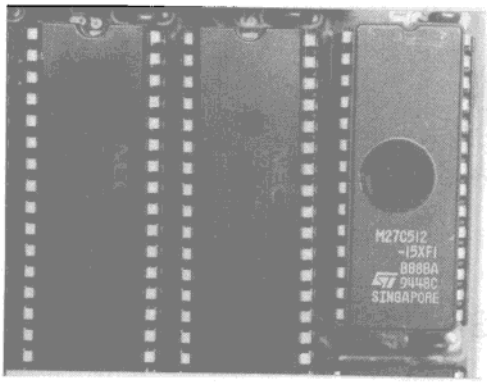


图 2-2 EPROM 的 CHECKSUM 法可以检查出数据是否曾被人修改过

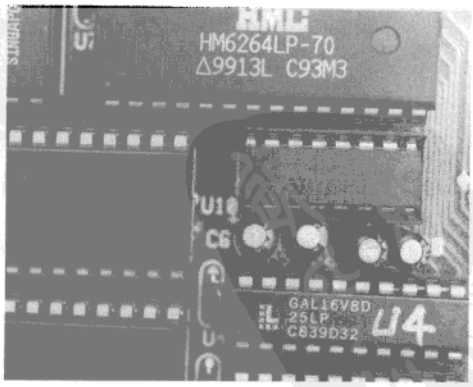


图 2-3 把 SRAM 6264 换成内含锂电池的 MK48T08B（8KBx8），就可以在停电时保留数据

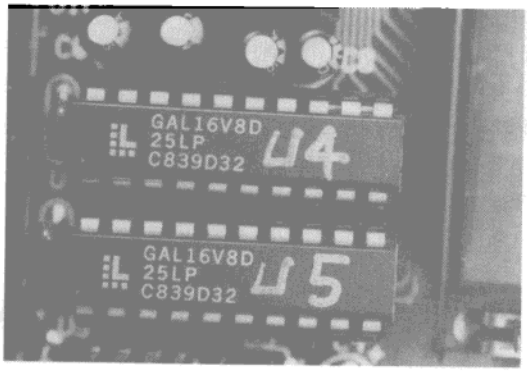


图 2-4　PEEL 是价格最划算的保护元件，但保密性有限

2-8　保护方法四：FPGA 保护法

FPGA 是 Field Programmable Gate Array 的缩写，翻译成中文是“现场可编程门阵列”，它是一种比 PEEL 组件还要复杂多倍的电子元件，通常 Gate Array 都是以多少门数代表它的容纳密度，Gate Array 之前又加上 Programmable 这个字，代表这些阵列出厂时都是开放自由的，使用者可以依照自己的需要赋予其特定的功能，最简单的 FPGA 至少有 2000 个门，约可取代数十个以上的标准 TTL 组件，如果设计得宜时，我们可以将 CPU 以外的所有电路都塞到 FPGA 内部，这样也就无形地保护了我们的程序和线路。有些 FPGA 是用电池来做状态保留（如 XILINX 的 20、30 系列），可以随时依需要修改其控制状态；也有些 FPGA 是烧死的（如 ACTEL 系列），亦即功能确定后就无法再做更改。使用 FPGA 的保密性是非常高的，但是 FPGA 的价格约在数十元至百元之间，除非是相当高价位的产品，否则不是很划算。

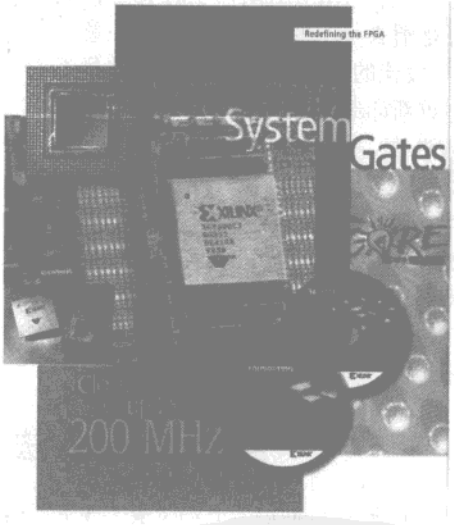


图 2-5　FPGA 的设计自由度可高达 50MHz 以上的工作频率，将是下一个程序保护的利器（本图由港商科成公司提供）

2-9　保护方法五：订做一个 CPU

使用通用 General Purpose CPU 的优点是参考资源丰富，市面上 Z80、6502 和 8051 的书籍多得数不清，但是资源愈是丰富，看得懂你的程序的人也就愈多，于是就有人想自己设计一个自用的 CPU，例如在开发阶段仍使用 Z80 CPU，并且在程序当中加入了特别的指

令，这些指令对 Z80 CPU 是无效码，但是若换上自行订做的 CPU 时，却是一个关键性的指令，除非竞争者能拿到相同订做的 CPU，否则是无法抄袭的。订做一个 CPU 的收费一般是要超过五位数的。1994 年日文版的《TRANSISTER 晶体管技术》杂志曾经做过一个示范，用 ALTERA 的 FPGA 做一个 6502CPU，有这方面兴趣的读者可以到图书馆找这篇文章来瞧瞧。最近甚为流行的 AT89C51，内含 4KB PROGRAM ROM，亦可做某一程度的保护，FLAG51 若使用 AT89C51 做程序保护时，必须把 EA 脚接成 HI，以便开机后先读内部的 4KB ROM。

2-10 保护方法六：掩人耳目 REMARK 芯片

从 20 年前废五金进口的年代开始，就有 REMARK（伪造仿制）的情形存在，不论购买任何计算机系统，我个人认为稳定性绝对比 CPU 执行速度还要重要，许多人在购买 PC 机时都要求要最快的，当然因此买到伪造的机会也就最大了。站在系统保护的观点上，让别人找不到 IC 编号可以适度地保护产品的投资，我们曾经买了一台 ICE 仿真器，打开外壳看内部的电路配置时，所有的 IC 编号全部被磨掉并重新 REMARK，工程真是耗大。可是消去 IC 的编号更能引人注意，所以，较佳的方法是把重要 IC 的编号消去并重新印制，让竞争者踏入预先埋设的陷阱中。伪造的对象可以是 CPU、ROM/RAM、TTL IC 或是任何电子元件，但是公司内部应该保留 IC REMARK 的更改记录表，以免以后故障品维修时砸到自己的脚。REMARK 芯片是一项不算技术的技术，但是有时还是能小兵立大功的。

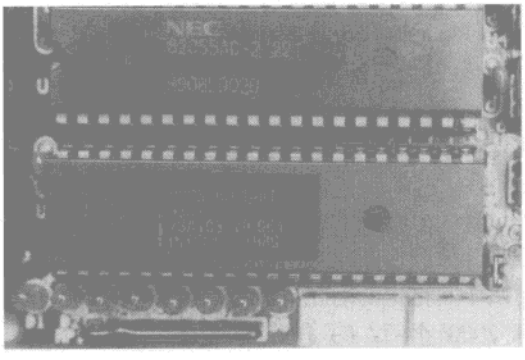


图 2-6 FLAG51 的 CPU 改用 AT89C51 时，必须把 JP1 的短路跳线 Jumper 取下

2-11 保护方法七：双 CPU 联机保护法

所有的保护程序及口令数据若放在 ROM 中，都会有被人破解的担忧，于是就有人想到了用两个 CPU 联机做程序的保护，联机的方式可以通过串行或并行端口，系统随机取得一个随机数后，就直接送到联机端口上，然后由联机端口读回一连串响应值，如果响应值全部正确时才继续工作。其实这种做法跟 PC 上打印机端口的 KEYPRO 类似，如果另一方的 CPU 又选用较冷门的 CPU 时，情况就很有看头了。在保护的程序领域当中，随机取随机数的技巧是很重要的，如果您固定送出一个检查码时，很容易就被别人看出漏洞，当然也就很容易被有心人解开了。

2-12 保护方法八：程序加入大量垃圾数据

如果别人要破解你写的任何程序，一定要通过反汇编程序先来了解程序内容。传统教科

书教我们写程序时，最好把程序段与数据段分开，最常见的例子是程序放前面，数据与查表数据安排在最后面，这种方法对解程序的高手是最方便的。真正好的程序要把程序与数据交替地安插在一起，尽量让破解程序的人感到不方便，并且故意在每段程序前加上一些错误的垃圾数据，让反汇编程序译错程序，这也可以减缓竞争者判读程序的速度，有些时候这些添加的垃圾数据还会帮上不少的忙呢！以单片机 8051 CPU 为例，不论查表或是做 I/O 存取时，一定会用到 MOV DPTR，#ADDR 这个指令，其机器码共占了 3 个字节，最前头的机器码一定是 90H，后两个字节才是 ADDR 地址值，所以反汇编程序只要一看到 90H 这个码一定会译成 MOV DPTR,#ADDR 这个指令。如果这个程序或子程序很重要，我们极不希望让别人一眼就看穿时，可以在该程序前加入一个假数据，比方说加上 64H 好了，当反汇编程序逐一翻译机器码时，会把 64H 和 90H 误认成指令 XRL A,#90H，而 ADDR 的地址值又被当成另一个指令，这个 MOV DPTR,#ADDR 指令就变成两个假的指令，很自然地隐藏在程序 ROM 中。

2-13 保护方法九：隐藏式地雷保护法

所有的保护程序都多多少少会有一些小漏洞，如果一发现程序被修改时就立即死机，这种做法很容易就被解开，较好的保护程序应该不是这个样子。首先程序执行后若发现有被修改过的迹象时，应该“不露痕迹地继续执行程序”，或是故意出现一些小问题，让竞争者很容易就解开，然后又持续地执行程序，由于程序被修改的标志位已被设定成真，所以程序会在某些个定点上“不定时”（这点相当重要）出错，让竞争者无法判定到底是程序出了问题，或是硬件线路出了差错，这就是隐藏式地雷保护法的精神所在，地雷的数量应该是越多越好，但是不能影响到原先程序的运行。不过，这类程序要写得不露痕迹还真的不容易！旗威科技公司曾经写过类似的程序，程序连续执行十天后，程序不知何种缘故误触了保护程序，结果当然是很惨了！

2-14 程序高手与解题高手

有一流的写保护程序高手之后，就会有一流的解题高手出现，一般我们看到的是后者比前者多了许多，许多人仍未有软件是有价的观念，在控制与单片机开发领域里，除了把程序处理好外，还要看情况加上适度的软硬件保护才行，在文章的最前头我们已经提到了这个观念：所有的保护只能减缓竞争者赶上的速度，绝不能做到 100%的完全保障，如果我们不能持续前进时，别人就会踏过我们的头上，逆水行舟不进则退，在单片机的学习过程中如此，其他的场合亦是如此。

本章使用的软件

（1）DIS51.EXE：旗威科技公司开发的 8051 反汇编程序。
（2）IAR 8051 C 编译器。
（3）Keil C 编译器。

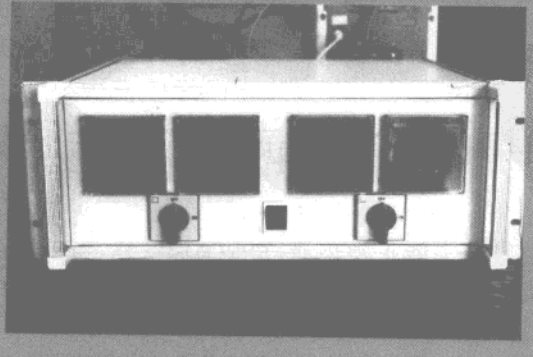
本章使用的硬件

（1）FLAG51 单片机控制板。
（2）PEEL IC:ICT 公司的可程序化零件。
（3）GAL IC:Lattice 公司的可程序化零件。
（4）Xilinx FPGA 高密度可程序化零件。
（5）Altera CPLD 高密度可程序化零件。

相关信息网站

可经由下列公司、网站取得更进一步的信息：
http://www.chipware.com.tw：取得 8051 单片机控制板相关资料。
http://www.ictpld.com：查询 PEEL 元件资料。
http://www.latticesemi.com：查询 CPLD 及 GAL 组件资料。
http://www.xilinx.com：查询 CPLD 及 FPGA 组件资料。
http://www.altera.com：查询 CPLD 及 FPGA 组件资料。
http://www.tek.com：查询示波器资料。
http://www.microchip.com：查询 Microchip PIC 相关资料。
http://www.iar.com：查询 8051 C 与 Assembler 相关资料。
http://www.micetek.com.tw：查询 8051 ICE 相关资料。

3



旗威科技公司开发的过电流保护元件测试仪，电动机、果汁机或碎纸机内部都有类似元件。

第 3 章　8051 程序的逆向工程

我们在本章里将做一个 8051 程序逆向追踪应用示范，供你参考及观摩用。读别人的程序是很痛苦的，尤其是汇编语言的部分。但是，我们遇见的几位 8051 的专家级的人物，都经历过这一段艰苦的日子，此时你下的苦心愈多，日后碰到的阻碍会愈少。

如果手边已有了一台 EPROM 烧录器，或是正准备买一台 EPROM 烧录器时，请不要以为 EPROM 烧录器就只能烧录 EPROM 了。如果能善用相关的工具或软件，一定会对工作或学习有所帮助的。以下是所做的一个 8051 程序逆向追踪应用示范，实际情况可能比所提到的复杂许多倍。

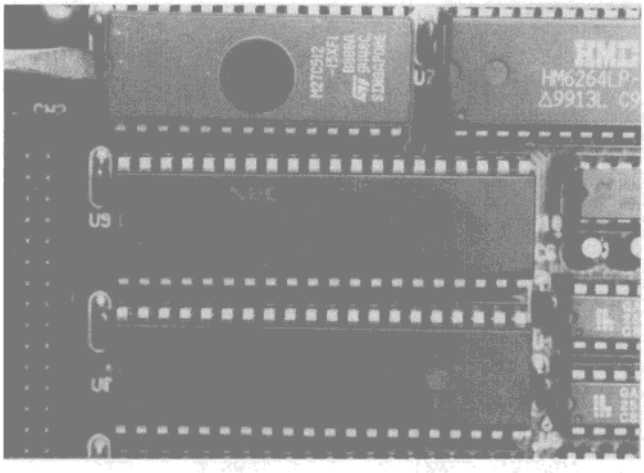


图 3-1　逆向工程的第一步就是取下研究对象的 EPROM，因为所有的奥秘都在其中

3-1　善用 EPROM 烧录器的上传与下载功能

一台 EPROM 烧录器除了能正常烧录 EPROM 以外，应该还能够由 PC 端下载（DOWNLOAD）烧录的数据，而且也应能将读入的数据上传到计算机当中。如果是单片机的开发者或学习者，前者使用的机率一定较高，但是试着当学习者的机会一定还是有的。这里所举的例子是一个 8051 的程序应用。假设拿到一个 FLAG51 单片机控制板，内部有一个 27256 的 EPROM，据说是个温度显示与控制用的程序。如果想“参考或观摩”别人程序的写法时，由于

没有开发时的原始程序，所以，只好通过 EPROM 烧录器先读入整个 EPROM 的内容了。以下就是整个操作过程的解说与示范，我们保证这些操作及说明很少出现在一般的教科书当中。

步骤 1：关闭电源后，小心地将该 27256 EPROM 从 FLAG51 控制板上移下来，如果是用“一”字型螺丝刀拆卸 IC 时，一定要分别从 IC 上下两方向慢慢挑起，以免 EPROM 引脚过度弯曲而折断。

步骤 2：在 PC 端执行 EPROM.EXE 烧录联机程序，并指定由 COM1 或 COM2 与旗威的 EPROM 烧录器连上线，EPROM 烧录器开机后，系统经过各种自我检验都成功后，会送回“EPROM 27256>”等的信息，这时代表 EPROM 烧录器与 PC 已经正式联机成功了。

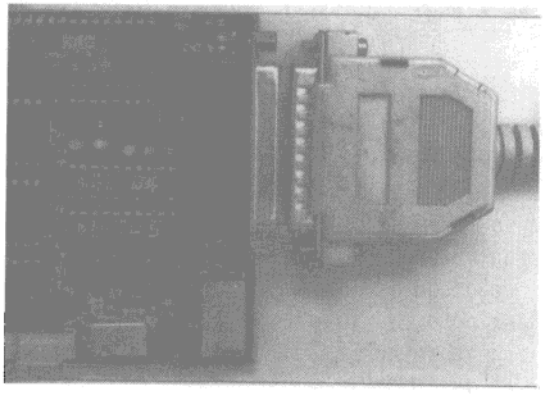


图 3-2　旗威科技公司的 EPROM 烧录器是通过 COM1 或 COM2 与 PC 连上线

步骤 3：将 EPROM 插到烧录器的测试夹上，按下“READ”或 R 键把 EPROM 的数据转存到系统的 SRAM 中。然后在键盘上按下大写“D”，并且输入由 0000H 开始显示 EPROM 上的数据，如果这时完全不去动键盘时，EPROM 烧录器会以一次一个 PAGE（256 字节）的方式，把该 PAGE 的内容分别以 HEX 及 ASCII 码的方式将数据显示在屏幕上。选用这个功能的目的是在观察该程序的正式结束地址大概在哪里。

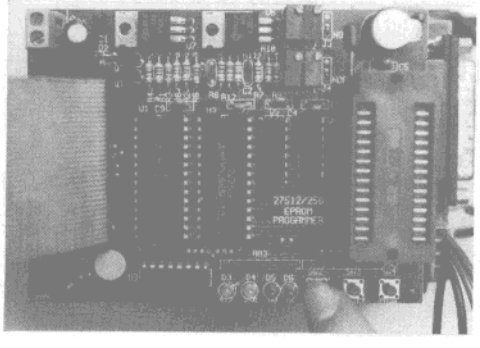


图 3-3　按下“READ”键就可以把 EPROM 的数据转存到烧录板的 64KB SRAM 中

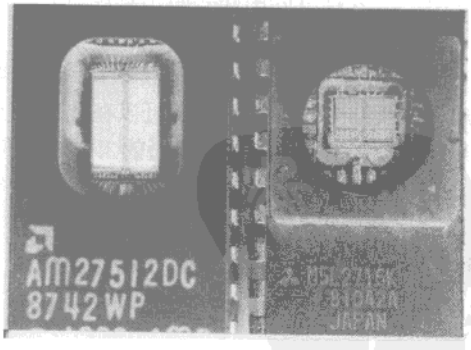


图 3-4　相同容量的 EPROM，其内部的芯片大小相差甚多，一个 27512 EPROM 当中，可以花费下工程师数月的设计心血

```
EPROM_27256[2DD0] 100μs width>d
Display RA< BUFFER by page, please input in HEX

Start addr :0
Display BUFFER SRAM 1 page(256 bytes)
Press any key to escape
Display BUFFER addr 0000H--00FFH
0000H 02 00 30 32 FF FF FF FF - FF FF FF 32 FF FF FF FF  |..02.....2....
0010H FF FF FF 32 FF FF FF FF - FF FF FF 32 FF FF FF FF  |...2.....2....
0020H FF FF FF 02 00 6B FF FF - FF FF FF FF FF FF FF FF  |.....k..........
0030H 75 81 60 75 90 FF 12 02 - E2 75 B0 FF C2 B2 12 01  |u. u.....u......
0040H DC 12 01 EB 12 01 DF 12 - 01 1C 12 01 2A 12 00 8F  |............*...
0050H 12 00 97 7A 00 7B 00 75 - 1F 00 C2 98 12 00 A0 80  |...z.[.u........
0060H FB 52 45 41 44 59 54 45 - 4D 50 3D 30 98 IC E5 99  |.READYTEMP=0....
0070H F4 F5 90 C2 98 B5 43 12 - B2 B2 02 00 87 C2 9D 30  |......C........0
0080H 98 FD E5 99 B4 54 03 12 - 01 64 C2 99 C2 98 32 75  |.....T...D....2u
0090H 98 D0 C2 98 C2 99 22 D2 - AC D2 AF C2 98 C2 99 22  |......"....2..."
00A0H 7C 00 7D 00 7B 00 7F 00 - 78 C9 20 B5 FD 30 B5 FD  |.}.~...x. ..0..
00B0H 0C 0E BC 00 01 0D BE 00 - 01 0F 20 B5 F3 0E 0A BE  |................
00C0H 00 01 0F BA 00 01 0B 30 - B5 F3 12 01 CF D8 E1 78  |.......0........X
00D0H 30 12 01 F4 78 30 E6 08 - FC E6 08 FD 7F 00 7F 00  |0...x0.........~.
00E0H 00 75 0F 00 75 0E 00 75 - 0D 00 75 0C D9 12 02 47  |.u..u..u..u....G
00F0H 78 38 12 01 F4 75 0F 00 - 75 0E 00 78 32 E6 F5 0C  |x8...u..u..x2...

EPROM_27256[2DD0] 100μs width>
```

图 3-5　先观看内容然后把有效的数据上传转存到 PC 的硬盘当中

步骤 4：一颗 27256 的 EPROM 可以存放相当多的 8051 汇编语言控制程序，通常很难把所有 32KB 的空间都占满，假设控制程序只有 5KB 时，剩下的 27KB 的空间都是空白的 FFH 或 00H，所以就可以由此判断出程序真正的结束地址了。假设看到的结束位置约在 1FFH 左右，这代表整个控制程序只有 512 字节而已。

步骤 5：键入“S”键以便上传数据到 PC 端，此时系统会要求输入存入的文件名称（TEST）与开始以及结束的地址，所以可以在此输入文件名，开始地址 0000H 与结束地址 01FFH，过一会儿就会在 PC 端产生一个新的二进制文件了。

步骤 6：如果手边有《8051 单片机彻底研究——基础篇》一书时，书后所附的光盘中有一个非常好的 8051 反汇编程序，该文件的名称为 DIS51.EXE，这是旗威科技公司参考并比较过数种厂商 ICE 的反汇编程序后，所自行开发出的 8051 专用反汇编程序，它所反汇编后的 ASCII 文件绝对比其他程序翻译的还要有看头。进行文件的反汇编前请先确定步骤 5 所存入的文件没有扩展名。

步骤 7：请键入“DIS51 TEST”，这代表反汇编程序将内含二进制的 TEST 文件翻译成原来的汇编语言格式，并且加入程序计数值与机器码，同时也将结果送到屏幕以及硬盘当中。正式执行时，程序会先告知该文件的真正长度，然后请输入想结束反汇编的真正地址。在此之后，反汇编程序就开始进行运行了，并且产生四组非常有效的文本文件数据。分别有反汇

编的 DIS 文件，指针 DPTR 参考专用的 DTR 文件，调用专用的 CAL 文件，还有跳转的 JMP 文件，只要你弄懂这几个文件，就可以掌握该程序的动向了。详细的操作说明请参考该书的附录 I 中的说明。

以下是这几个文件与二进制程序间的相对关系：

观察 DTR 文件：可以判断出数据区的分布及外部 I/O 的地址。

观察 CAL 文件：可以看出哪些子程序是最底层的子程序，以及子程序间调用的关系。

观察 JMP 文件：可以看出程序间相互跳转的关系。

步骤 8：反汇编后的 DIS 文件很类似 8051 汇编程序所输出的 LST 参考文件，只要把程序计数值及机器码的部分删掉，就可以将原程序重新编译成二进制文件的 TSK 数据文件。不过如果 8051 的程序中把数据区与程序区放在一起时，反汇编程序有可能会造成若干的误判，因而丧失了部分的地址标签，所以必须再用人工的方式做一次错误修正。

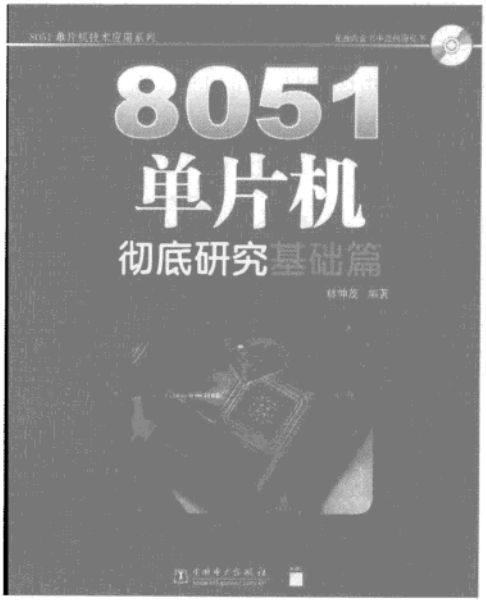


图 3-6 《8051 单片机彻底研究——基础篇》的书后附有一个相当好用的 8051 反汇编程序，如果你不曾使用过，那就辜负了作者的一番苦心了

```
Please input binary filename (max length 32768 bytes) :test
Disassembler on SCREEN (y/n) ?y
Checking.....
Your PROGRAM size= 2f5H
This program will generate 4 output files:
       1.ASCII disassembler file  :TEST.DIS
       2.Disassembler DPTR  file  :TEST.DTR
       3.Disassembler JUMP  file  :TEST.CAL
       4.Disassembler JUMP  file  :TEST.JMP
End address in HEX(0000-02F4):2f4
Exact end_address=02F4H
Pass 1...
Total JMP,CALL & DPTR count = 73
Pass 2...
02F4H   22                    RET                          ;0000011B
Pass 3...
Sorting CALL data...    Total LCALL & ACALL count       =   20
Sorting DPTR data...    Total DPTR count                =   3
Sorting JUMP data...    Total LJMP,AJMP & SJMP count    =   5
Sorting Bit addressable segment...
Sorting Data Memory Aree...
*** Table reference file in [TEST.TBL]***(please check)
```

图 3-7 执行 8051 反汇编时所看到的画面

步骤 9：在最新版的 DIS51.EXE 程序中，特地加了一个数据区与程序区的判断表，第一次执行 DIS51.EXE 程序时，它会自动产生一个.TBL 参考文件，这是程序分辨有关 DPTR 指令后所产生的数据文件，这个文件记载现在所分析的 8051 二进制程序文件中，疑似的数据区开始与结束的地址。可以用一般的编辑程序去查看其内容，也可以去修改这些地址。

```
Directory of E:\TEMP

.               <DIR>              10-11-95    12:16p
..              <DIR>              10-11-95    12:16p
TEST                                757    03-14-96    11:40p
DIS51    EXE                    107,248    03-14-96    6:10P
TEST     DIS                     19,344    03-14-96    11:43P
TMP                                 757    03-14-96    11:46P
TEST     JCD                        438    03-14-96    11:43P
TEST     TB1                         18    03-14-96    11:43P
TEST     TBL                         64    03-14-96    11:43P
TEST     CAL                        365    03-14-96    11:43P
TEST     DTR                         51    03-14-96    11:43P
TEST     JMP                         85    03-14-96    11:43P
TEST     BAS                         18    03-14-96    11:43P
TEST     DMA                        252    03-14-96    11:43P
     14 file(S)                129,387 bytes
                                8,607,744 bytes free

E:\TEMP>type test.tb1
0061 0065  ;length=5
0066 006A  ;length=5
0202 02E1  ;length=16
```

图 3-8　DIS51 反汇编程序所产生的 TABLE 数据文件内容

步骤 10：如果再次执行 DIS51.EXE 去反汇编刚刚才反汇编的 TEST 二进制文件，由于先前已产生了 TEST.TBL 疑似数据区产生文件，所以再次反汇编时，反汇编程序只要一碰到这一特定区域时，就会直接转译成 DB（Define Byte）的数据文件。这时再度查看最后的 DIS 反汇编文件时，就会发现程序与数据区都已被分隔开了，而这些额外的操作是一般简单型的 8051 反汇编程序所无法做到的。

步骤 11：取得反汇编程序后，可以先分析各个子程序的写法，然后逐渐摸索出整个程序的架构来，这段探讨的过程可能要花上几天或几星期的时间，由此才会体验出整个 8051 控制程序的流向与写法，有许多应用程序的写法是完全不公开的，唯有借助这种方法才可以得到这些精心之作，当然这些技巧是很难在一般书籍上看到的。

步骤 12：经过反汇编后的 DIS 程序只要删掉前面的程序计数值与机器码部分，并把程序扩展名改成.ASM 就可以重新编译这个 8051 程序文件，这个程序文件再经过 LINK 连接处理后的.TSK 二进制文件应该和原来的二进制文件完全一样才对，如果不是，一定是整个操作的环节中有一个地方出现问题了。

步骤 13：对逆向工程人员而言，到达此地步才算是刚开始而已，所有的子程序都必须一

一记录及测试，所有用过的 RAM 地址也必须知道其用途及使用的时机，然后再把整个控制环节解开，原来别人的程序是这样完成的！许多程序都加入了禁止别人修改的设置，只要一改了程序中任一个位置的内容后，就有可能使程序完全失效，知道这种写法吗？如果不知道时，就多看看别人怎样做到的吧！

```
43    0047    12 01 1C                    CALL    SET_TIMER
44    004A    12 01 2A                    CALL    ENABLE_TIMER
45    004D    12 00 8F                    CALL    ENABLE_SERTAL
46    0050    12 00 97                    CALL    ENABLE_INER
47    0053    7A 00                       MOV     R2,#00H
48    0055    7B 00                       MOV     R3,#00H
49    0057    75 1F 00                    MOV     COUNT,#00H
50    005A    C2 98                       CLR     RI
51                                ;       CALL    SEND_ID
52                                ;
53                                ;*****************************************
54                                ;       SETB    SM2          ;MULTI-PROCESS
55                                ;
56    005C    12 00 A0            AGAIN   CALL    READ_TEMP
57    005F    80 FB                       SJMP    AGAIN
58                                ;
59                                ;       JNB     RI,AGAIN
60                                ;       MOV     A,SBUF       ;RI=TRUE
61                                ;       CLR     RI
62                                ;       CJNE    A,ID_CODE,AGAIN
63                                ;       CALL    SNED_OUT     ;OUTPUT
64                                ;       JMP     AGAIN        ;NEXT LOOP
65                                ;*****************************************
66                                ;
67    0061    52 45 41 44 59MSG-ID DB      'READY'
68    0066                        MSG_TEMP
69    0066    54 45 4D 5D 3D       DB      'TEMP='
70
```

图 3-9（a） 我们的温度控制器的原始程序，加了‘:’后的叙述是无法出现在二进制文件上的，这个文件是经过 Assembler 所产生的 LST 文件

步骤 14：程序的逆向工程的合法性一直是个相当令人争议的话题，有些软件或操作系统还特别注明不得利用逆向工程的方式去分析其程序。如果您仍要进行软件开发，一定要把研究程序的人员与正式写程序的人员完全分隔开，并保留所有研究及开发时的文件及数据，以便在有争议时当成最有力的佐证。

近几年来，外接 EPROM 的控制板愈来愈少了，为了体积缩小或保密的缘故，许多 IC 制造商陆续推出内含 64KB（或是更多的存储空间）的微控器。但是只要你有办法从该 IC 读回完整的二进制程序文件，你仍旧可以沿用原来的方法进行程序的逆向工程，追踪程序的流程及写法。

```
0036H   12 02 E2              LCALL   L02E2H
0039H   75 B0 FF                MOV     P3,#FFH       ;11111111B
003CH   C2 B2                   CLR     P3,2
003EH   12 01 DC                LCALL   L01DCH
0041H   12 01 EB                LCALL   L01EBH
0044H   12 01 DF                LCALL   L01DFH
0047H   12 01 1C                LCALL   L011CH
004AH   12 01 2A                LCALL   L012AH
004DH   12 01 8F                LCALL   L008FH
0050H   12 00 97                LCALL   L0097H
0053H   7A 00                   MOV     R2,#00H       ;00000000B
0055H   7B 00                   MOV     R3,#00H       ;00000000B
0057H   75 1F 00                MOV     1FH,#00H      ;00000000B
005AH   C2 98                   CLR     SCON.0
005CH   12 00 A0        L005CH  LCALL   L00A0H
005FH   80 FB                   SJMP    L005CH
0061H   52 45           L0061H  ANL     45H,A
0063H   41 44                   AJMP    L0244H
0065H   59                      ANL     A,R1
0066H   54 45           L0066H  ANL     A,#45H        ;01000101B
0068H   4D                      0RL     A,R5
0069H   50 3D                   JNC     L00A8H
```

图 3-9（b） 反汇编程序若把数据区一反汇编时，0061H 开始原先是数据区，但是反汇编后就变成“无法理解的程序行”了

```
0044H   12 01 DF              LCALL  L01DFH
0047H   12 01 1C              LCALL  L011CH
004AH   12 01 2A              LCALL  L012AH
004DH   12 00 8F              LCALL  L008FH
0050H   12 00 97              LCALL  L0097H
0053H   7A 00                 MOV    R2,#00H      ;00000000B
0055H   7B 00                 MOV    R3,#00H      ;00000000B
0057H   75 1F 00              MOV    1FH,#001     ;00000000B
005AH   C2 98                 CLR    SCON.0
005CH   12 00 A0    L005CH    LCALL  L00A0H
005FH   80 FB                 SJMP   L005CH
0061H   52          L0061H    DB 52H    ;R
0062H   45                    DB 45H    ;E
0063H   41                    DB 41H    ;A
0064H   44                    DB 44H    ;D
0065H   59                    DB 59H    ;Y

0066H   54          L0066H    DB 54H    ;T
0067H   45                    DB 45H    ;E
0068H   4D                    DB 4DH    ;M
0069H   50                    DB 50H    ;P
006AH   3D                    DB 3DH    ;=
```

图 3-9（c） 再次反汇编后，DIS51 会参考 TBL 数据文件，把数据区接分隔开，原来 0061H 开始是一段 ASCII 文字

3-2 所有的设计都是从模仿开始

看别人的汇编程序太累了吧！可是几乎所有的程序设计师都会有一段“研究别人程序”的青涩时期，如果此时花下的心血越多，以后碰到的阻碍也会越少。如果你会善用 8051 反汇编程序，就会使在学习上更得心顺手。本文也附上一个不到 1KB 的 8051 程序文件，这是一个用在 AT89C2051 的温度显示程序，内部包含了 32 位的乘除运算及温度的测量与显示，同时还具备串行通信的能力。请试着依照所提的步骤做做看，并且完成整个程序的批注与说明，如果自认为写得够透彻时，请将程序说明寄给旗威科技公司，公司会赠送 AT89C2051 学习用的 PC 板一块，以资鼓励。

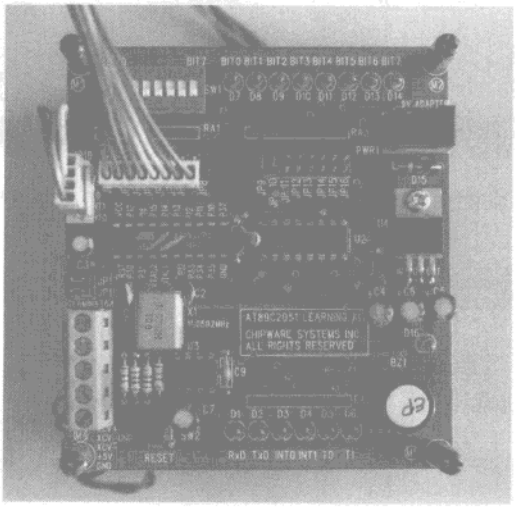


图 3-10 AT89 C2051 学习板的实物图

EPROM 的保存与清除

EPROM 本身是 CMOS 元件，当湿度小时很容易因人体所累积的静电冲击而损坏。在 IC 生产工厂中，手上若没挂接地线而去触摸任何 IC 时，是有可能被“开除”的，其严重性可由此点看出。所以国外 IC 的制造商一定会在包装袋上加入“小心静电”的警语。如果是向电子元件零售商购买 EPROM 时，请商家务必将 EPROM 先放在防静电袋中，再放到一般的塑料袋中。如果购买 EPROM 的零售商没有如此处理，那就代表他们的专业知识有待加强了。

当 EPROM 内的数据不符需要清除时，可以到电子元件零售商购买清除 EPROM 专用的擦除器，内部附有机械式的定时装置，只要把 EPROM 放到该擦除器内，紫外灯就会在设置好的时间内点亮并清除 EPROM 内的所有数据。也可以到医疗仪器行购买杀菌用的紫外灯（外观与一般日光灯相同，但灯管是透明的），再到灯具行购买合适的灯座，就可以拿来做 EPROM 的擦除器了。通常 EPROM 是先插在黑色的防静电袋上，然后放在紫外灯灯管下约 5cm 处，只要照射 20～30min 就可以将 EPROM 内部所有的数据全部清除成 FFH。EPROM 擦除器所照射出的强烈紫外线对眼睛是绝对有害的，最好避免直视擦除器的灯管，如果使用的场合可

能会有儿童靠近时，一定要将该擦除器放在小朋友无法开启且眼睛无法直视的地方，以免发生意外。

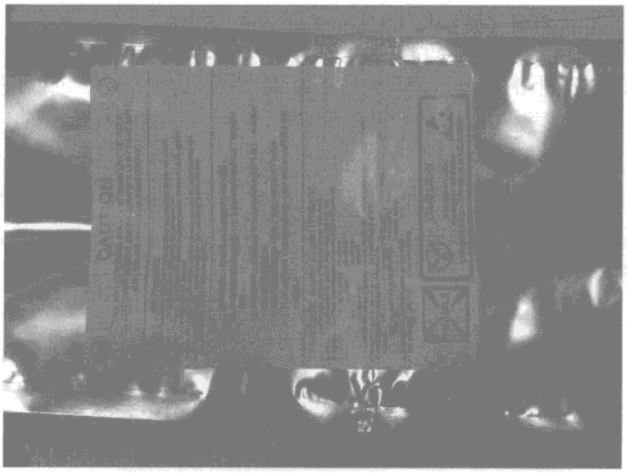


图 3-11　包装 IC 的静电袋外，一定会标示“小心静电”等英文警告标示语

本章使用的软件

（1）8051 Assembler 汇编程序。
（2）DIS51.EXE“旗威科技”所开发的 8051 反汇编程序。

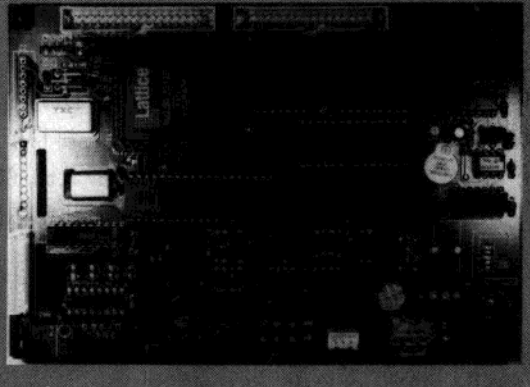
本章使用的硬件

（1）FLAG51 单片机控制板。
（2）EPROM 烧录板。
（3）EPROM 27512（512Kb）/27256（256Kb）。
（4）AT89C2051：内置 2KB Flash 的 8051 单片机 CPU。
（5）万用烧录器（Universal Programmer）。
（6）紫外灯：清洗 EPROM 数据专用。

相关资料网站

可经由下列公司、网站取得更进一步的信息：
http://www.chipware.comtw：查询 8051 单片机相关资料。
http://www.rs232.com.tw：查询各种电子零件资料。
http://www.atmel.corn：查询 AT89C 系列 Micro controller 及 8051 共享软件。
http://www.goole.com：查询 EPROM 引脚及制造商。

4



工业级的 8051 控制板，为了有更灵活的 IO 应用，我们用 Lattice 的 CPLD 来取代 8255。

第 4 章　实验桌上的反思

DIY 是有风险存在的，但是隐含在风险之后的知识及经验却是你我都需要的。我们常鼓励读者 DIY，许多技巧与精髓是隐藏在动手做的过程当中，唯有亲自 DIY 时，才有机会吸取到相关的经验与 KNOW-HOW，这也是 DIY 最吸引人的地方。

4-1　电解电容的爆炸

当我拿到 EPROM 烧录板刚出炉的第一片空 PC 板时，心里唯一的信念就是先把所有的元件装好，进行所有 EPROM 烧录操作的确认，自认是非常细心谨慎的我最后竟然犯了一个严重的错误，不加思索地打开电源后，突然觉得操作好像有点不太对，随即低头下去查看 PC 板上的元件，这时突然听到一声类似鞭炮的巨响，随即看到实验桌上充满了烟雾，再仔细看了一下：EPROM 烧录板上电源输入端的电容已经爆开了，而且爆开电容内的填充物喷得到处都是，幸好已戴了近视眼镜多年，否则极有可能变成独眼龙的。

造成电解电容的爆炸只有一种可能：电解电容的极性接反了。原来是我把 24V 直流电压的电源输入端接反了，才造成该电容的爆炸。电解电容的爆炸是很危险的，首先内部的电解液因极性颠倒会先沸腾，然后电容体积膨胀到原来的数倍，之后随即爆开并喷出滚烫的电解液。为了防止电解电容的爆炸而伤害到人，电容量超过 10μF 的电解电容都会在电容的顶部打上一个类似奔驰汽车商标的三叉符号，以便电容膨胀准备炸开时，先引导其从顶部爆开，以免造成更大的伤害。其实，我们写程序时也要有类似的观念，当系统察觉有问题时，一定要指引使用者的特别注意，而不是避开问题不去处理。

上一次我经历的电容爆炸好像是装功率放大器的时候，算来也有 10 年以上的时间了，通常加上电源前我都会再确定一下极性是否正确，但是纵使仅有一次的疏忽，也有可能造成许许多多的灾难，以这次电容爆炸为例，几乎板上的 IC 元件都损坏了，幸好这些元件都还有备品可以更换的，而且花费还不算太大。自己 DIY 的风险就在这里，可是如果没有经历这些令人难忘的爆炸情景，绝对无法使一个人牢记这类的教训，DIY 的风险可以换来我们做任何事情的双重核对，我认为这是非常值得的。所谓居安思危，讲得一点都没错。如果尚未看过电容爆炸的惨状，请特别注意开关机时的意外。有人说搭飞机最危险的时侯是起飞与降落的时侯，应该也是这个道理。如果没有准备妥当，贸然打开电源所要付出的代价是相当高的。

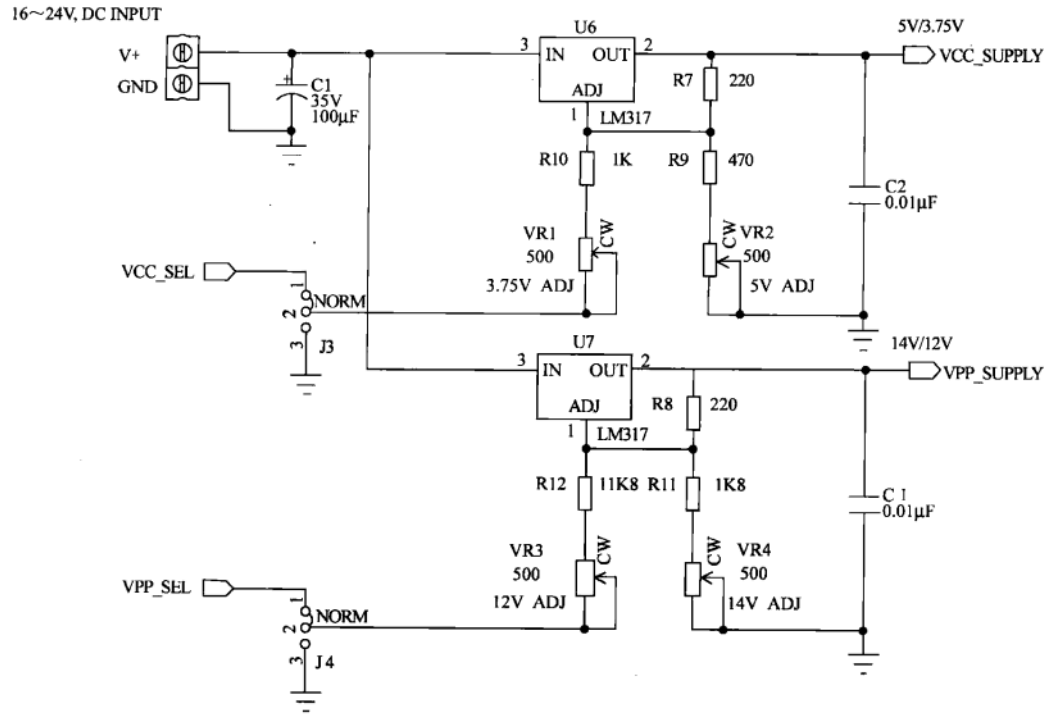


图 4-1　由于输入烧录电源的反接，造成 C1 电解电容的爆炸，这使我们在 DIY 时要更加小心了。此图中的 R9、R10、R11 及 R12 的安排是另有目的，请看文中的叙述

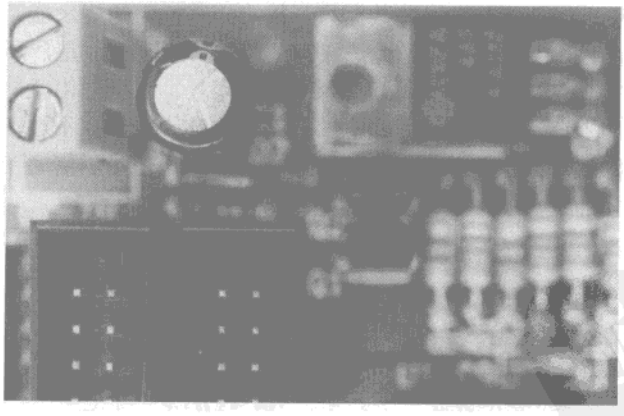


图 4-2　大容量的电解电容的顶部都有一个三叉的符号，看了本文就知道它的用意

4-2　EPROM 烧录器之后

从《RUN！PC》创刊到现在已经超过两年了（收录了自 1996 年的文章），在这两年期间我们陆续在单片机专栏上谈到许多 8051 单片机的应用，这些都是属于 FLAG51 单片机的

相关应用，我们不仅公布了硬件线路与对应程序的写法，而且也具体交待了自己装 DIY 的步骤，我们相信我们所讲的与所做的对众多 DIY 读者而言绝对是最完整的。经过两年的辛苦耕耘，我们已开发出一些实用的工具，例如 I/O 监视板、七段显示器以及 EPROM 与 AT89CXX 烧录器，这些都是我们在学习 8051 单片机不可或缺的工具。我们常鼓励读者不要只依照书做完实验后就认为完全弄懂了，因为杂志或书都只是传递观念的工具，许多技巧与精髓是隐藏在动手做的过程当中，唯有我们亲自 DIY 时，才有机会吸取到相关的经验与 KNOW-HOW，如果请别人代做或捉刀时，那就绝对无法获得其中的奥妙，当然这也是 DIY 最吸引人的地方。

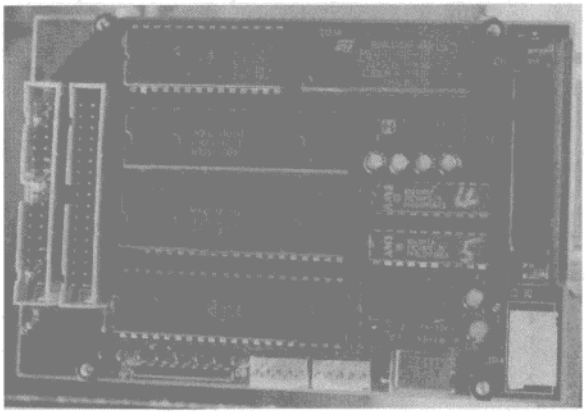


图 4-3　FLAG51 单片机控制板，所有的单片机应用都少不了它

最近几年来我们一直在找寻可程序化元件 PEEL 及 GAL 的烧录数据，以便自行设计一台 PEEL 的烧录器，最后的结论是：这些数据只开放给一些特定的烧录器制造厂商，并且也签订了相关的保密协议，一般的消费者是无法取得这些数据的。IC 制造厂所持的理由是怕烧录数据一经外泄后，就会有竞争者加入，而失掉现有的市场。这个理由就好像国防上的“军事机密”一样，只要加上这 4 个字，所有善意的查询都会吃闭门羹。不过，无论如何，我们各项烧录器的开发只好到此为止了，至于这些可程序化的 IC 还是可以借助一般的万用烧录器烧录，只是要花费较多的钱罢了。

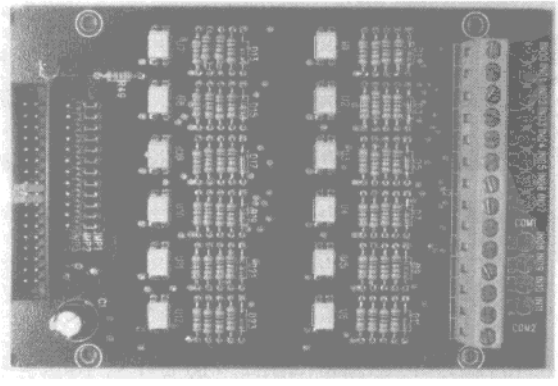


图 4-4　隔离输入板已用在 PLC 应用上，或是单片机控制的输入上

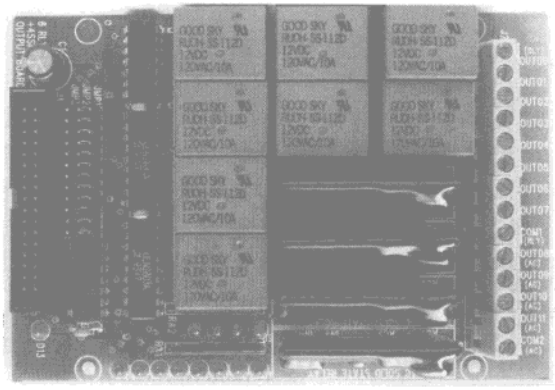


图 4-5　RELAY 输出板共有 12 个输出点，一般的应用已足够有余了

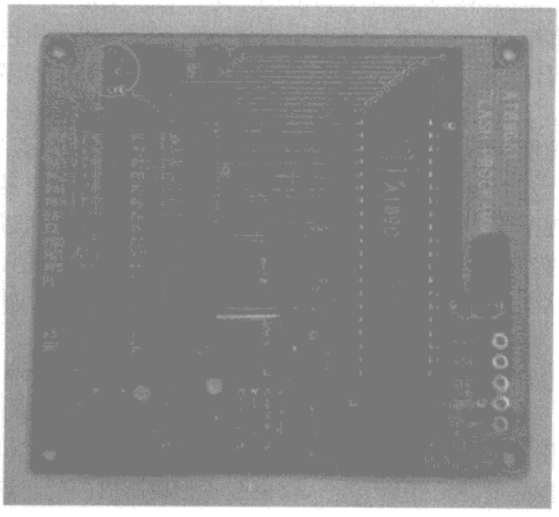


图 4-6　AT89CXX 烧录板可以烧录内含闪存的微控制器（Microcontroller）

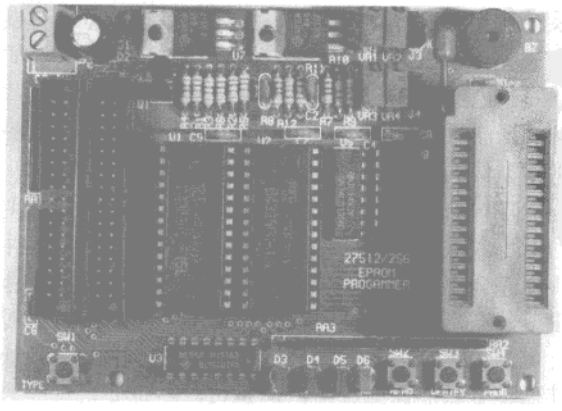


图 4-7　EPROM 烧录器可以烧录常用的 27256 及 27512

4-3 如何成为单片机专业人士

最近有一些用功的读者来信询问如何练得一身好功夫，我的建议是多订一些专业性的杂志来看，老实说：国内信息类技术杂志中值得每期珍藏保存的并不多见，看到较好的文章顶多影印起来而已，但是国外的杂志却不是如此。尤其是日文的“实践派”的杂志编辑得特别好，我个人从 1984 年开始就订阅日文的《晶体管技术月刊》，至今已接近 20 年的时间，只要懂基本的日文语法及外来语，就可以参考这些日文杂志的内容，这些杂志的特点是每月有一个特定的专题，这些专题所谈的技术与 KNOW-HOW 都是很好的选择，文中的插画、图表、照片以及解说都特别多且详尽，所以只要弄懂这些环节，大概就对这些专题有了必要的了解，可是这方面中文杂志比起来就相差甚多了，多半是报道性的文章居多，一过了时效后其保存价值就降低许多。

其实我们在这里的范例讲解及线路有许多构想是来自日文杂志，所以文章中的线路要求绝对是最清晰而且没有错误，同时照片及解说也要是同类杂志中最出色的。如果是个用功的读者，我们相信绝对可以从文章中吸取最多的经验。我们认为除了多多吸取专业杂志的内容外，最好采取“做中学”（Learning by doing）的方式来增进自己的程序设计与除错能力。如果想学习 AD 与 DA 转换器的相关知识，何不给自己定个较大的目标，例如多点温度控制器或是可编程的电源供应器。也许你会说，这些不都是市面上已有的产品，我跟着做这些东西有什么用呢？倒不如做个以前不曾看过的。如果有类似的看法时，可以试着想想看，一件内置单片机的产品从构思到完成，中间经过了许多的努力与思考，有硬件线路的规划，也有软件程序的特别安排，这些都是身历其境的人才得以体会的，如果我们始终置身于外时，仅能体会其中的十分之一。相信吗？不信的话请看以下的例子：

以旗威科技公司最近几个月热卖的 EPROM 烧录器为例，请看图 4-1，这是烧录电压（+13V）调整的线路，我们在每个可变电阻上方都加入了一个电阻，在我们试制烧录线路时，是没有加上这 4 个电阻的，所以 4 个可变电阻的阻值都是不一样的。可是如果考虑到产量及读者 DIY 调整时，就要尽量将元件单一化，以免准备材料时的困扰，同时也要使调整更加容易，所以我们事先计算出 4 个添加电阻的约略阻值来，以便在读者 DIY 焊接完成后，尚未调整可变电阻 VR 前，输出的烧录电压早已保持在可以容许的范围内。这也就是说，如果读者自己装的 EPROM 烧录板若元件没有装错的话，完全不调整 VR 就直接进行烧录时，也不会对 EPROM 造成损坏。这些一连串妥善的硬件安排，可以让调整失误的机会降到最低，这就像棒球投手投出球的瞬间，已经将球的方向保持在好球区内，当然这些事前的安排绝对不是光看线路图就可以理解的，而且这也是我们在自行开发初期所产生的概念，如果不亲自动手的话，压根不会去考虑到类似的问题。

第 2 个例子是当我们正式着手写 EPROM 烧录程序时，我们才发现烧录的速度才是整个烧录程序的重心，而且市面上的烧录器也一直强调烧录时间的长短。当我们开始进行 EPROM 实际测试时，才发现许多完全正确的子程序因为处理时间过长都要重新改写！而从改写的过程当中，还得到一个重要的启示：以前我们都一直以为烧录 EPROM 时，每烧录一个字节就要做一次确认的动作，可是如果真的这样做时，会导致整个烧录时间加倍，反而不是最好的做法。我们并未追踪其他烧录器厂商的程序，但是可以断定如果真的边烧录边做验证时，一定无法在 10s 以内烧录完一个 32KB 的 27256，所以一定有些烧录器厂商用其他迂回的方式

做确认，比方说：每 256B 内只检查其中的几个字节，或是累积更多个烧录字节之后才进行验证等等。

在旗威科技公司的 EPROM 烧录器上，我们所顾虑的因素更多了，因为如果仍用 8051 做 CPU 时，烧录速度没法提高到相当快，如果再加上烧录时的数据验证，那会使烧录时间更形加长，所以我们只好采用权宜之烧录完后才逐一验证的做法，并且还加入了烧录数据的错误显示信息，自动判断是否可以再烧录或是必须到紫外灯下清除数据。这些都是我们深入探讨 EPROM 烧录时，才获得的最后心得，如果没有研究与开发的工作，那对 EPROM 而言就完全只是断断续续的报导与信息而已。

我们一直很纳闷为何有些厂商始终采取抄袭别人产品的态度？如果自己不研究开发的话，又如何使技术生根呢！1996 年的农历春节前我们完成了整个 EPROM 烧录器的程序开发，但是我们的脑海中已经浮现出另外一台完全用硬件构成的 EPROM 烧录器的方框图，烧录时间可以更短，只要时机许可时我们就可以设计类似的产品了，自己研究开发的产品或许短时间的利润不是很好，但是长远的角度来看应该还是值得的。

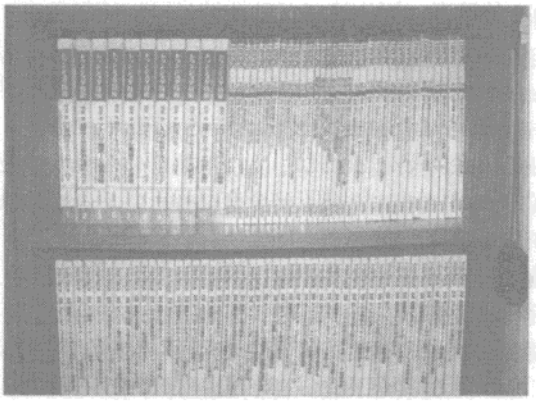


图 4-8　书架上的日文杂志《晶体管技术》是我个人认为很好的参考资料来源

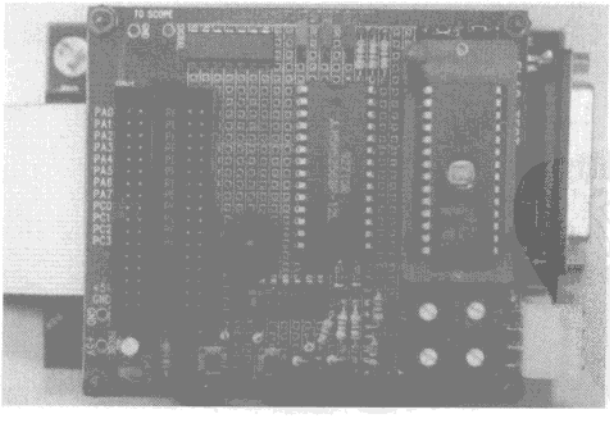


图 4-9　一颗 27256 是内置有 32KB 的 EPROM，能在 10s 内完成烧录，
有点不可思议，但是早已有厂家做出来了，但是他们不会立刻告诉您怎么做到的，
我们也试出来了，图中是我们的 EPROM 烧录器的 PROTOTYPE

4-4 有认真的读者才有用心的作者

如果您经常到书店看书，一定会发现在计算机丛书的作者队伍中增加了许多新面孔，而如果是翻译的书籍也都会清楚地交待了授权翻译的著作权声明，这种现象对信息产业确实是个好消息，在某些特定行业中的专业技术人员把他的经验写下来，以造福广大的学习者，尤其是因特网的应用，更需要新作者的加入。在众多的计算机信息书籍当中，我们建议读者不要被计算机软件的版本给限制住了，当新版本一经问世后，旧版的书籍就沦入冷宫或退货销毁的命运，所以只要一有新的软件推出，市面上马上就涌现许多几乎类似的书籍出现，不仅写法一样，插图又几乎雷同。其实，“真理始终是不需要升级的”，只要我们懂得软件其中的道理，版本的更新只是使我们在操作更加方便的工具而已。

我们认为：读者与作者间的关系就好像鸡生蛋然后蛋生鸡一样，而且有认真的读者之后，才会有用心的作者。每次接到读者的来信或电话指出书中哪一部分或软件操作有问题时，我们都会再三地答谢这些用心的读者，因为这些互动的结果可以让作者更加用心去写说明和范例，而不是随随便便敷衍几句就交待过去了。几年前我们用了一整年的时间，开发了一套可编程控制器 PLC 的系统程序，另外也按该 PLC 的架构编写了 8051 单片机彻底研究的第二本书《PLC 应用篇》，书推出数月后我接到一个读者的电话：林先生你在书上第××页说到，市面上的 PLC 都要上万元，可是我用的 PLC 只要几千元而已，所以书上这段叙述是否有问题存在？这位读者是以价格来认定 PLC，不过他可能忽略了一个产品还有许多部分是无法用价格来衡量的。

我本身也使用过类似的 PLC，操作性与稳定度都还不错，不过我最不能忍受的是其中文使用手册处理的草率感，完全不把使用者的观点纳入，所以我的回答是这样的：如果您本身已经有了 PLC 的使用经验，购买现成的 PLC 控制器当然成本最低。可是假使您对 PLC 只是一些模糊的概念，那要跨过这些“看了仍不懂”的使用手册大门槛，那可要花相当多的精力的，另外您的几千元只是用来买该台 PLC，碰到难题时可是不含任何的 service 费用的。我们的做法就比较弹性些了，您可以买书来看，书里涵盖了所有 PLC 的相关知识，然后您可以用积木的方式组合出一套包括 24 点 I/O 的 PLC 系统来。这些一连串的做法与策略我个人必须承认是受数年前的《音响技术月刊》的影响甚大，而其中最重要的观念是知识的分享，虽然财富很难共享，但是知识确定是可以大家分享的。

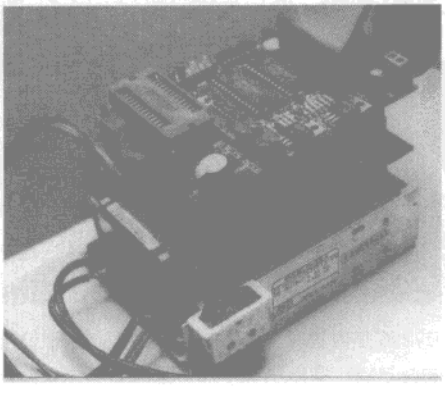


图 4-10 对技术者而言，亲自参与 EPROM 烧录器的开发项目，所获得的经验绝对是别人所无法赋予的

图 4-11　令人怀念的音响技术月刊杂志

本章使用的软件

（1）8051 Assembler。
（2）2500AD 8051 C Compiler。
（3）IAR C Compiler。

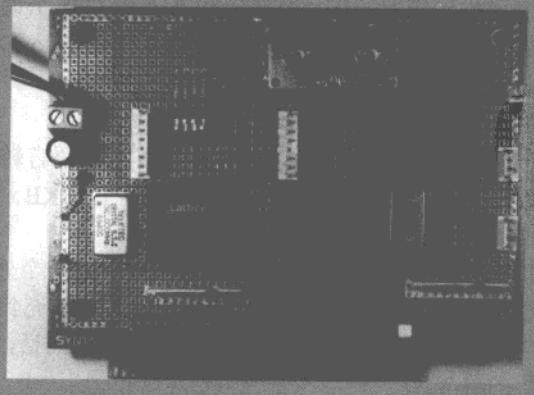
本章使用的硬件

（1）FLAG51 8051 单片机控制板。
（2）EPROM 烧录板。
（3）隔离输入板。
（4）RELAY 输出板。
（5）AT89CXX 烧录板。

相关资料网站

可经由下列公司、网站取得更进一步的信息：
http://www.chipware.com.tw：取得单片机控制板相关资料。
http://www.atmel.com：取得 AT89C 系列 Microcontroller 相关资料。
http://www.cqpub.co.jp：取得日文杂志《晶体管技术》资料。

5



旗威科技试做 Bit Error Rate 测试专用的随机数信号发生器。

第5章　C语言的导入：SDCC

用C语言来学8051或是用8051来学C语言，已经是不可避免的潮流了，前者可能入门较易，但是要历经较久的时间才能对8051通盘理解。后者已经对8051有足够的认识，学了C语言的写法以后，系统的开发时间会明显减少，而且还能写出非常结构化的程序模块。不论你是属于何种类型，看完本章再决定是否好好地学一下C语言。

5-1　使用C语言学习8051

8051 CPU在20年前就问世了，可算是单片机CPU的常青树，以8051为主体的应用范例太多了。C语言比8051更早被开发出来，我们相信当初Intel把8051定位在单片机控制器上，希望开发者尽量用汇编语言来设计程序，以便争取最快的执行速度。所以，内置的程序空间只有4KB，数据空间也只有128字节，虽然到了8052时，程序空间与数据空间都加倍，但对某些特定的应用场合来讲，还是觉得存储容量不太够用。

当8051与C语言搭上关系时，首先被人质疑的是：可能吗？合适吗？首先当然是考虑到程序空间的问题，在C语言里，有许多函数是相当耗费程序空间的，例如浮点运算以及printf函数，8051仅4KB的程序空间可能一下子就用完了，又如何拿来写应用程序呢？另一方面，C对堆栈的使用机会很高，传递自变量时，都会通过堆栈来做。对8051而言，可用的堆栈实在有限，数据存储区里有R0-R7寄存区（00H～1FH）要避开，可位寻址区（20H～2FH）也要避开，剩下80字节的空间又要分配给函数内的变量使用，导致真正给堆栈使用的空间可能不到50字节。50字节只能做25次调用而已。当8051真正在运行时，主程序也在做调用，各个中断服务程序依照其先后优先级也在调用，25个调用可能进到中断后一下子就分配完了，再多做一次调用后，堆栈立刻溢出（stack overflow），系统返回时找不到对的返回地址，随即死机了。

综上所述，如果要用C语言来驾驭8051，请记住：

第一，尽量加大程序空间（Program Memory），以便容纳C语言转换出来的机器码。

第二，扩充外部的数据空间（Data Memory）到2KB甚至32KB，让C有绝对足够的空间来保存变量及外部的堆栈。

对C而言，8051的堆栈是绝对不足的，所以C会自己建立一个外部堆栈的机制，以便更有效地运用外部的SRAM区。说更明白一点，将8051的系统扩充到ROM 64KB，SRAM也接近64KB，这时用C语言就没有任何疑虑了。如果不是这样的话，你可能要面对无数死机与“系统始终怪怪的”的挑战。

5-2 如何获取 C

传统的 C 编译器（Compiler）若要用在 8051 单片机上，一定要经过优化处理，当然其中一定包括先前我们谈到的 8051 堆栈特别处理。现在比较常见的 8051 C 编绎器应该是 IAR C 与 Keil C 两款，这些都是要用钱买的商业软件。通常我们可以通过因特网到上述公司的网站上下载所谓的试用版（trial version），这方面请直接到网站另行下载使用手册。在这里我们要介绍的是拥有许多支持者的共享软件 SDCC C Compiler。SDCC 主要支持的 CPU 为 Intel 的 8051 与 Zilog 的 Z80 CPU，另外也即将支持 Atmel AVR、DS390 与 Microchip PIC 等系列。

SDCC 是由 Sandeep Dutta 所写的，当开始在网络上流传时，历经了许多 bug 的修正与缺陷改进，这可是经过全世界数百个 C 的高手提出意见后的结果。1999 年年底，整个 SDCC 软件程序转移到 Sourceforge 机构上，以便让所有的用户兼开发者都可以接触到整个完整的原始程序（source tree），对 SDCC 做更大的奉献与改进。SDCC 最初只有 Linux 版本的，因为 Linux 才有软件共享的观念，可是面对用户众多的 Windows 操作系统，SDCC 也另外提供 Windows 版的 SDCC，当然这些版本都可以在网络上无偿获取。

以下就是到 http://sdcc.sourceforge.net 获取 Windows 版的重要画面，我们从网络上获取的版本为 V2.3.0 版。如果你是下载 Linux 版的 SDCC，但又想要在 Windows 的环境下执行，你还要到 http://sources.redhat.com/里下载 CYGWIN 的完整版本才能工作。

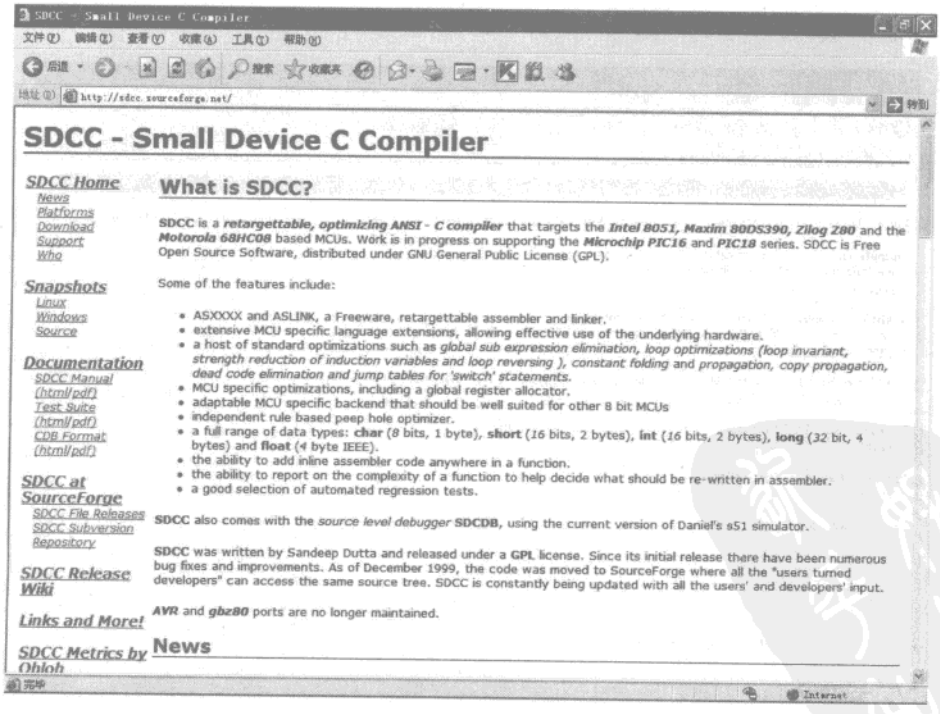


图 5-1 SDCC 的首页，告诉用户 SDCC 的由来以及如何下载

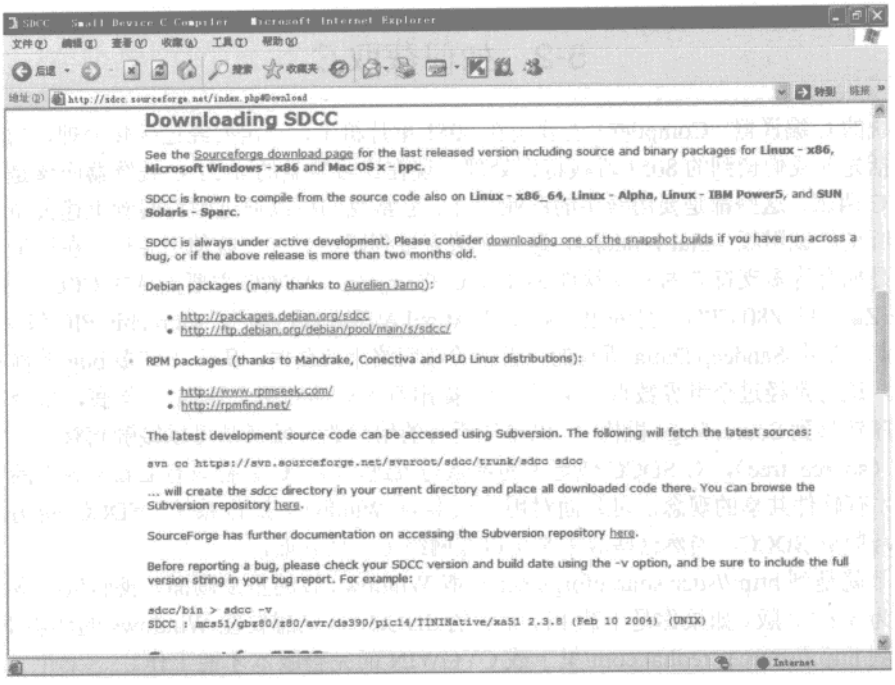


图 5-2　下载 Windows 版本的 SDCC

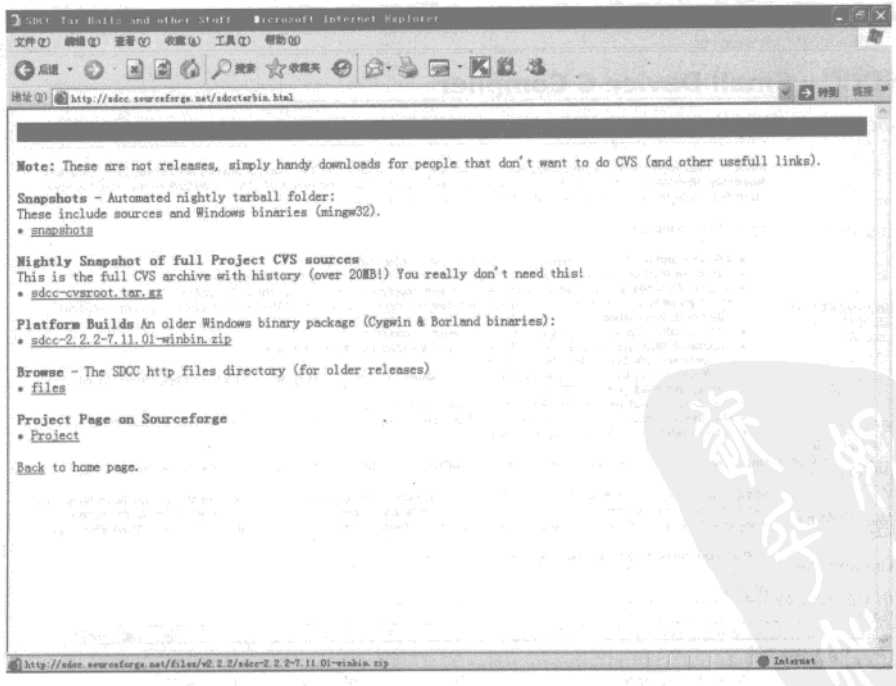


图 5-3　指定下载的文件名为 sdcc-2.2.2-7.11.01-winbin.zip

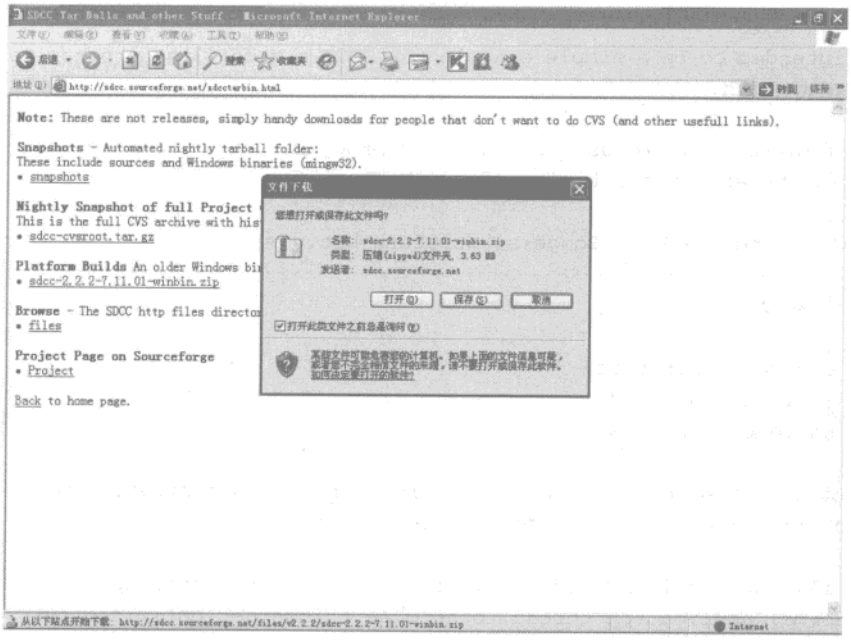


图 5-4　把文件转存在硬盘上，如果是用拨号上网的话，可能要多花一点时间才能完成

5-3　SDCC 操作程序

Windows 版的 SDCC 的文件约有 3.63MB，下载完毕后直接用 Winzip 将文件解压缩，假设指定解压缩的路径在 c:\sdcc，则解压缩完成后会有以下几个目录：

C:\sdcc	SDCC 的主目录。
C:\sdcc\bin	经过 Borland 编译过的 SDCC.exe 可执行文件。
C:\sdcc\cygbin	经过 Cygwin 编译过的 .exe 可执行文件。
C:\sdcc\include	C 语言使用的包含文件。
C:\sdcc\lib	C 语言的链接库，内部又分为 small 与 large 两种。

当然还有其他的说明文件与范例文件。

当我们把画面切换到 MS-DOS 后，须先指定可执行文件的路径，先键入 PATH=c:\sdcc\bin;%PATH%<enter>，让系统找得到 sdcc 的可执行文件。真正下的命令只有一行，分别要指定包含文件的路径以及链接库的路径：sdcc -I c:\sdcc\include -L c:\sdcc\lib\small 你的文件名.c

上面的命令行指定 C 的包含文件的路径在 c:\sdcc\include，而链接库的路径为 c:\sdcc\lib\small。

以下为 SDCC 提供的测试文件 hi.c 的原始程序内容。

```
/*-------------------------------------------------------------------------*/
  hi.c - This is a simple program designed to operate on basic MCS51 hardware at
    11.0592Mhz. It sets up the baudrate to 9600 and sends a "Hi\n" "There\n" message
    every few seconds. The timer interrupt is used.
```

```
    Its intended to be a simple example for SDCC,a good first program to compile
    and play with.

    The simulator can be used to run this program:
    s51 -Sout=testout.txt hi.ihx (run, stop, quit)

  6-28-01 Written by Karl Bongers(karl@turbobit.com)
|-------------------------------------------------------------------------*/
#include <8052.h>

typedef unsigned char byte;
typedef unsigned int word;
typedef unsigned long l_word;

//---- most of the following declares are simply to demostrate some
// of SDCC's variable storage declaration syntax

// volatile keyword is needed for variables shared by interrupt routine
// and normal application thread, otherwise things get optimized out.
volatile data byte timer;
volatile data byte hi_flag;

data byte a_data_byte;              // normal < 128 bytes of 8031 internal memory
                                       内容 Data Memory 的声明
idata byte a_idata_byte;            // in +128 byte internal memory of 8032 内容
                                       Data Memory 的声明
xdata byte a_xdata_byte;            // in external memory 存取外部内存
xdata at 0x8000 byte mem_mapped_hardware;  // example at usage,指定 I/O 地址

bit my_bit;                         // mcs51 bit variable, stored in single bit
                                       of register space,单一位声明

sfr at 0xd8 WDCON;                  // special function register declaration,SFR
                                       的声明
sbit LED_SYS = 0xb5;                // P3.5 is led, example use of sbit keyword,
                                       单一硬件位的声明
code char my_message { }= {"GNU rocks"}; // placed in code space,字符串声明

void timer0_irq_proc(void) interrupt 1 using 2;
                                    //timer0 的中断,使用 BANK2 的寄存器群

/*-------------------------------------------------------------------------
  timer0_int - Timer0 interrupt. Notice we are using register bank 2
    for this interrupt.
|-------------------------------------------------------------------------*/
void timer0_irq_proc(void) interrupt 1 using 2    //timer0 中断服务程序的内容
{
```

```
    if (timer != 0)
    {
       --timer;
    }
    else
    {
      hi_flag = 1;
      timer = 250;
    }

    TR0 = 0; /* Stop Timer 0 counting */
    TH0 = (~(5000)) << 8;
    TL0 = (~(5000)) & 8;
    TR0 = 1; /* Start counting again */
}

#if 0
/*-------------------------------------------------------------------------
  uart0_int - Interrupt 4 is for the UART, notice we use register bank 1 for the
    interrupt routine.
|-------------------------------------------------------------------------
void uart0_int(void) interrupt 4 using 1   // sio 串行中断使用 BANK1 的寄存器群
{
    if (RI)
    {
      c = SBUF;
      RI = 0;
    }
}
#endif
/*-------------------------------------------------------------------------
    tx_char - transmit(tx) a char out the serial uart.
|-------------------------------------------------------------------------*/
void tx_char(char c)
{
    while (!TI)
      ;
    TI = 0;
    SBUF = c;
}

/*-------------------------------------------------------------------------
    tx_str - transmit(tx) a string out the serial uart.
|-------------------------------------------------------------------------*/
void tx_str(char *str)
{
    while (*str)
    tx_char(*str++);
```

```
}

/*-------------------------------------------------------------------------
  stop - a break point in Daniel D's s51 can be set at 65535 memory location to
    stop the simulation. This routine also shows how to embed assembly.
    可以在C语言中加入汇编语言,请比较产生的LST文件,以理解中间的差异
|-------------------------------------------------------------------------*/
void stop(void)
{
  _asm;
   mov dptr, #65535;
   movx a, @dptr;
   nop;
  _endasm;
}

/*-------------------------------------------------------------------------
  main - Simple test program to send out something to the serial port.
|-------------------------------------------------------------------------*/
void main(void)
{
PCON = 0x80; /* power control byte, set SMOD bit for serial port */
SCON = 0x50; /* serial control byte, mode 1, RI active */
TMOD = 0x21; /* timer control mode, byte operation */
TCON = 0; /* timer control register, byte operation */

TH0 = (~(5000)) << 8; /* the initial time is not important */
TL0 = (~(5000)) & 8;

TH1 = 0xFA; /* serial reload value, 9,600 baud at 11.0952Mhz */
TR1 = 1; /* start serial timer */

TR0 = 1; /* start timer0 */
ET0 = 1; /* Enable Timer 0 overflow interrupt IE.1 */
EA = 1; /* Enable Interrupts */

TI = 0; /* clear this out */
SBUF = '.'; /* send an initial '.' out serial port */
hi_flag = 1;
//ES = 1; /* Enable serial interrupts IE.4 */

tx_str(my_message);

    for (;;)
    {
      if (hi_flag)
      {
        tx_str("Hi\n");
```

```
        tx_str("There\n");
        hi_flag = 0;
      }

stop();

#ifdef TEST_IDLE_MODE
    // this was a simple test of the low power sleep mode of a
    // dallas DS5000 cmos part, to see how much power requirements
    // dropped in sleep mode.

    // into idle mode until next interrupt. Draws only 3ma.
    PCON = 0x81;
#endif
    }
}
```

5-4 C 编译后所产生文件

传统的 C 语言书籍开头一定是个 printf（“hello\n”）的范例展示，SDCC 的 hi.c 也是如此，不过它是从串行端口每隔数秒就送出一个重复的字符串值。要做到这点对 8051 而言，已经是一大步了。hi.c 程序中也示范如何用定时中断以及如何在程序当中加入汇编语言。

以下是 hi.c 经过 SDCC C Compiler 编译过后的 LST 文件，如果你是汇编语言的高手但初次接触 C 的话，请花一点时间研究 SDCC 所产生的汇编语言文件。其实在某种程度来看，C 转出来的文件也是相当精简的。SDCC 最后产生的可烧录文件是 Intel 的 hex 文件（扩展名为 ihx），如果你的烧录器只能接受纯二进制（binary）文件时，只要再执行 hex2bin.exe 后就可以进行烧录了。Hex2bin 这个程序应该在有关烧录器的网站或“旗威科技”（www.chipware.com.tw）的网站上获取。

```
1 ;--------------------------------------------------------
2 ; File Created by SDCC : FreeWare ANSI-C Compiler
3 ; Version 2.2.2 Tue Apr 16 22:45:11 2002
4
5 ;--------------------------------------------------------
6 .module hi
7
8 ;--------------------------------------------------------
9 ; Public variables in this module 整体变量的声明
10 ;--------------------------------------------------------
11 .globl _my_message
12 .globl _main
13 .globl _stop
14 .globl _tx_str
15 .globl _tx_char
16 .globl _hi_flag
17 .globl _timer
18 .globl _timer0_irq_proc
19 ;--------------------------------------------------------
```

```
                20 ; special function registers 8051 SFR的声明
                21 ;--------------------------------------------------
0080            22 _P0      =   0x0080
0081            23 _SP      =   0x0081
0082            24 _DPL     =   0x0082
0083            25 _DPH     =   0x0083
0087            26 _PCON    =   0x0087
0088            27 _TCON    =   0x0088
0089            28 _TMOD    =   0x0089
008A            29 _TL0     =   0x008a
008B            30 _TL1     =   0x008b
008C            31 _TH0     =   0x008c
008D            32 _TH1     =   0x008d
0090            33 _P1      =   0x0090
0098            34 _SCON    =   0x0098
0099            35 _SBUF    =   0x0099
00A0            36 _P2      =   0x00a0
00A8            37 _IE      =   0x00a8
00B0            38 _P3      =   0x00b0
00B8            39 _IP      =   0x00b8
00D0            40 _PSW     =   0x00d0
00E0            41 _ACC     =   0x00e0
00F0            42 _B       =   0x00f0
00C8            43 _T2CON   =   0x00c8
00CA            44 _RCAP2L  =   0x00ca
00CB            45 _RCAP2H  =   0x00cb
00CC            46 _TL2     =   0x00cc
00CD            47 _TH2     =   0x00cd
00D8            48 _WDCON   =   0x00d8
                49 ;--------------------------------------------------
                50 ; special function bits
                51 ;--------------------------------------------------
0080            52 _P0_0    =   0x0080
0081            53 _P0_1    =   0x0081
0082            54 _P0_2    =   0x0082
0083            55 _P0_3    =   0x0083
0084            56 _P0_4    =   0x0084
0085            57 _P0_5    =   0x0085
0086            58 _P0_6    =   0x0086
0087            59 _P0_7    =   0x0087
0088            60 _IT0     =   0x0088
0089            61 _IE0     =   0x0089
008A            62 _IT1     =   0x008a
008B            63 _IE1     =   0x008b
008C            64 _TR0     =   0x008c
008D            65 _TF0     =   0x008d
008E            66 _TR1     =   0x008e
008F            67 _TF1     =   0x008f
0090            68 _P1_0    =   0x0090
0091            69 _P1_1    =   0x0091
```

```
0092          70 _P1_2        =   0x0092
0093          71 _P1_3        =   0x0093
0094          72 _P1_4        =   0x0094
0095          73 _P1_5        =   0x0095
0096          74 _P1_6        =   0x0096
0097          75 _P1_7        =   0x0097
0098          76 _RI          =   0x0098
0099          77 _TI          =   0x0099
009A          78 _RB8         =   0x009a
009B          79 _TB8         =   0x009b
009C          80 _REN         =   0x009c
009D          81 _SM2         =   0x009d
009E          82 _SM1         =   0x009e
009F          83 _SM0         =   0x009f
00A0          84 _P2_0        =   0x00a0
00A1          85 _P2_1        =   0x00a1
00A2          86 _P2_2        =   0x00a2
00A3          87 _P2_3        =   0x00a3
00A4          88 _P2_4        =   0x00a4
00A5          89 _P2_5        =   0x00a5
00A6          90 _P2_6        =   0x00a6
00A7          91 _P2_7        =   0x00a7
00A8          92 _EX0         =   0x00a8
00A9          93 _ET0         =   0x00a9
00AA          94 _EX1         =   0x00aa
00AB          95 _ET1         =   0x00ab
00AC          96 _ES          =   0x00ac
00AF          97 _EA          =   0x00af
00B0          98 _P3_0        =   0x00b0
00B1          99 _P3_1        =   0x00b1
00B2         100 _P3_2        =   0x00b2
00B3         101 _P3_3        =   0x00b3
00B4         102 _P3_4        =   0x00b4
00B5         103 _P3_5        =   0x00b5
00B6         104 _P3_6        =   0x00b6
00B7         105 _P3_7        =   0x00b7
00B0         106 _RXD         =   0x00b0
00B1         107 _TXD         =   0x00b1
00B2         108 _INT0        =   0x00b2
00B3         109 _INT1        =   0x00b3
00B4         110 _T0          =   0x00b4
00B5         111 _T1          =   0x00b5
00B6         112 _WR          =   0x00b6
00B7         113 _RD          =   0x00b7
00B8         114 _PX0         =   0x00b8
00B9         115 _PT0         =   0x00b9
00BA         116 _PX1         =   0x00ba
00BB         117 _PT1         =   0x00bb
00BC         118 _PS          =   0x00bc
00D0         119 _P           =   0x00d0
```

```
00D1          120 _F1          =   0x00d1
00D2          121 _OV          =   0x00d2
00D3          122 _RS0         =   0x00d3
00D4          123 _RS1         =   0x00d4
00D5          124 _F0          =   0x00d5
00D6          125 _AC          =   0x00d6
00D7          126 _CY          =   0x00d7
00AD          127 _ET2         =   0x00ad
00C8          128 _T2CON_0     =   0x00c8
00C9          129 _T2CON_1     =   0x00c9
00CA          130 _T2CON_2     =   0x00ca
00CB          131 _T2CON_3     =   0x00cb
00CC          132 _T2CON_4     =   0x00cc
00CD          133 _T2CON_5     =   0x00cd
00CE          134 _T2CON_6     =   0x00ce
00CF          135 _T2CON_7     =   0x00cf
00C8          136 _CP_RL2      =   0x00c8
00C9          137 _C_T2        =   0x00c9
00CA          138 _TR2         =   0x00ca
00CB          139 _EXEN2       =   0x00cb
00CC          140 _TCLK        =   0x00cc
00CD          141 _RCLK        =   0x00cd
00CE          142 _EXF2        =   0x00ce
00CF          143 _TF2         =   0x00cf
00B5          144 _LED_SYS     =   0x00b5
              145 ;--------------------------------------------------------
              146 ; internal ram data 内容Data Memory的使用情形
              147 ;--------------------------------------------------------
              148 .area DSEG (DATA)
0000          149 _timer::
0000          150 .ds 1
0001          151 _hi_flag::
0001          152 .ds 1
0002          153 _a_data_byte::
0002          154 .ds 1
              155 ;--------------------------------------------------------
              156 ; overlayable items in internal ram 重复使用区
              157 ;--------------------------------------------------------
              158 .area _DUMMY
              159 .area OSEG (OVR,DATA)
              160 ;--------------------------------------------------------
              161 ; Stack segment in internal ram 堆栈的使用
              162 ;--------------------------------------------------------
              163 .area SSEG (DATA)
0000          164 __start__stack:
0000          165 .ds 1
              166
              167 ;--------------------------------------------------------
              168 ; indirectly addressable internal ram data(80H~FFH)
              169 ;--------------------------------------------------------
```

```
                    170 .area ISEG (DATA)
0000                171 _a_idata_byte::
0000                172 .ds 1
                    173 ;--------------------------------------------------------
                    174 ; bit data 单一可位寻址区的使用情形
                    175 ;--------------------------------------------------------
                    176 .area BSEG (BIT)
0000                177 _my_bit::
0000                178 .ds 1
                    179 ;--------------------------------------------------------
                    180 ; external ram data 外部数据区
                    181 ;--------------------------------------------------------
                    182 .area XSEG (XDATA)
0000                183 _a_xdata_byte::
0000                184 .ds 1
8000                185 _mem_mapped_hardware = 0x8000
                    186 ;--------------------------------------------------------
                    187 ; interrupt vector 中断服务程序进入点
                    188 ;--------------------------------------------------------
                    189 .area CSEG (CODE)
0000                190 __interrupt_vect:
0000 02s00r00               191 ljmp __sdcc_gsinit_startup
0003 32                     192 reti
0004                        193 .ds 7
000B 02s00r38               194 ljmp _timer0_irq_proc
000E                        195 .ds 5
0013 32                     196 reti
0014                        197 .ds 7
001B 32                     198 reti
001C                        199 .ds 7
0023 32                     200 reti
0024                        201 .ds 7
002B 32                     202 reti
002C                        203 .ds 7
                            204 ;--------------------------------------------------------
                            205 ; global & static initialisations
                            206 ;--------------------------------------------------------
                            207 .area GSINIT (CODE)
                            208 .area GSFINAL (CODE)
                            209 .area GSINIT (CODE)
0000                        210 __sdcc_gsinit_startup:
0000 75 81 17               211 mov sp,#23 ;堆栈设在这里
0003 12s00r00               212 lcall __sdcc_external_startup
0006 E5 82                  213 mov a,dpl
0008 60 03                  214 jz __sdcc_init_data
000A 02s00r33               215 ljmp __sdcc_program_startup
000D                        216 __sdcc_init_data:
                            217 .area GSFINAL (CODE)
0000 02s00r33               218 ljmp __sdcc_program_startup
                            219 ;--------------------------------------------------------
```

```
                        220 ; Home
                        221 ;--------------------------------------------------------
                        222 .area HOME (CODE)
                        223 .area CSEG (CODE)
                        224 ;--------------------------------------------------------
                        225 ; code
                        226 ;--------------------------------------------------------
                        227 .area CSEG (CODE)
0033                    228 __sdcc_program_startup:
0033 12s00rA3           229 lcall _main
                        230 ; return from main will lock up
0036 80 FE              231 sjmp .
                        232 ;--------------------------------------------------------
                        233 ;Allocation info for local variables in function
                            'timer0_irq_proc'
                        234 ;--------------------------------------------------------
                        235 ; hi.c 47
                        236 ; -----------------------------------------
                        237 ; function timer0_irq_proc
                        238 ; -----------------------------------------
0038                    239 _timer0_irq_proc: ;定时中断从这里执行
0012                    240 ar2 = 0x12
0013                    241 ar3 = 0x13
0014                    242 ar4 = 0x14
0015                    243 ar5 = 0x15
0016                    244 ar6 = 0x16
0017                    245 ar7 = 0x17
0010                    246 ar0 = 0x10
0011                    247 ar1 = 0x11
0038 C0 E0              248 push acc
003A C0 F0              249 push b
003C C0 82              250 push dpl
003E C0 83              251 push dph
0040 C0 D0              252 push psw
0042 75 D0 10           253 mov psw,#0x10
                        254 ; hi.c 49
0045 E5*00              255 mov a,_timer
                        256 ; Peephole 110 removed ljmp by inverse jump logic
0047 60 04              257 jz 00102$
0049                    258 00107$:
                        259 ; hi.c 51
0049 15*00              260 dec _timer
                        261 ; Peephole 132 changed ljmp to sjmp
004B 80 06              262 sjmp 00103$
004D                    263 00102$:
                        264 ; hi.c 55
004D 75*01 01           265 mov _hi_flag,#0x01
                        266 ; hi.c 56
0050 75*00 FA           267 mov _timer,#0Xfa
0053                    268 00103$:
```

```
                        269 ; hi.c 59
0053 C2 8C              270 clr _TR0
                        271 ; hi.c 60
0055 75 8C 00           272 mov _TH0,#0x00
                        273 ; hi.c 61
0058 75 8A 00           274 mov _TL0,#0x00
                        275 ; hi.c 62
005B D2 8C              276 setb _TR0
005D                    277 00104$:
005D D0 D0              278 pop psw
005F D0 83              279 pop dph
0061 D0 82              280 pop dpl
0063 D0 F0              281 pop b
0065 D0 E0              282 pop acc
0067 32                 283 reti
                        284 ;------------------------------------------
                        285 ;Allocation info for local variables in function
                            'tx_char'
                        286 ;------------------------------------------
                        287 ; hi.c 83
                        288 ; ---------------------------------------
                        289 ; function tx_char
                        290 ; ---------------------------------------
0068                    291 _tx_char:
0002                    292 ar2 = 0x02
0003                    293 ar3 = 0x03
0004                    294 ar4 = 0x04
0005                    295 ar5 = 0x05
0006                    296 ar6 = 0x06
0007                    297 ar7 = 0x07
0000                    298 ar0 = 0x00
0001                    299 ar1 = 0x01
                        300 ; hi.c 88
0068 AA 82              301 mov r2,dpl
                        302 ; hi.c 85
006A                    303 00101$:
                        304 ; Peephole 111 removed ljmp by inverse jump logic
006A 30 99 FD           305 jnb _TI,00101$
006D                    306 00108$:
                        307 ; hi.c 87
006D C2 99              308 clr _TI
                        309 ; hi.c 88
006F 8A 99              310 mov _SBUF,r2
0071                    311 00104$:
0071 22                 312 ret
                        313 ;------------------------------------------
                        314 ;Allocation info for local variables in function
                            'tx_str'
                        315 ;------------------------------------------
                        316 ;str Allocated to registers r2    r3 r4
```

```
                        317 ; hi.c 94
                        318 ; ---------------------------------------
                        319 ; function tx_str 传送1 byte
                        320 ; ---------------------------------------
0072                    321 _tx_str:
                        322 ; hi.c 0
0072 AA 82              323 mov r2,dpl
0074 AB 83              324 mov r3,dph
0076 AC F0              325 mov r4,b
                        326 ; hi.c 97
0078                    327 00101$:
0078 8A 82              328 mov dpl,r2
007A 8B 83              329 mov dph,r3
007C 8C F0              330 mov b,r4
007E 12s00r00           331 lcall __gptrget
                        332 ; Peephole 105 removed redundant mov
0081 FD                 333 mov r5,a
                        334 ; Peephole 110 removed ljmp by inverse jump logic
0082 60 18              335 jz 00104$
0084                    336 00108$:
                        337 ; hi.c 98
0084 0A                 338 inc r2
0085 BA 00 01           339 cjne r2,#0x00,00109$
0088 0B                 340 inc r3
0089                    341 00109$:
0089 8D 82              342 mov dpl,r5
008B C0 02              343 push ar2
008D C0 03              344 push ar3
008F C0 04              345 push ar4
0091 12s00r68           346 lcall _tx_char
0094 D0 04              347 pop ar4
0096 D0 03              348 pop ar3
0098 D0 02              349 pop ar2
                        350 ; Peephole 132 changed ljmp to sjmp
009A 80 DC              351 sjmp 00101$
009C                    352 00104$:
009C 22                 353 ret
                        354 ;------------------------------------------
                        355 ;Allocation info for local variables in function
                            'stop'
                        356 ;------------------------------------------
                        357 ; hi.c 106
                        358 ; ---------------------------------------
                        359 ; function stop
                        360 ; ---------------------------------------
009D                    361 _stop:
                        362 ; hi.c 112
                        363 ;
009D 90 FF FF           364 mov dptr, #65535;
00A0 E0                 365 movx a, @dptr;
```

```
00A1 00                 366 nop;
00A2                    367 00101$:
00A2 22                 368 ret
                        369 ;------------------------------------------
                        370 ;Allocation info for local variables in function
                            'main'
                        371 ;------------------------------------------
                        372 ; hi.c 118
                        373 ; ---------------------------------------
                        374 ; function main 主程序在这里
                        375 ; ---------------------------------------
00A3                    376 _main:
                        377 ; hi.c 120
00A3 75 87 80           378 mov _PCON,#0x80
                        379 ; hi.c 121
00A6 75 98 50           380 mov _SCON,#0x50
                        381 ; hi.c 122
00A9 75 89 21           382 mov _TMOD,#0x21
                        383 ; hi.c 123
00AC 75 88 00           384 mov _TCON,#0x00
                        385 ; hi.c 125
00AF 75 8C 00           386 mov _TH0,#0x00
                        387 ; hi.c 126
00B2 75 8A 00           388 mov _TL0,#0x00
                        389 ; hi.c 128
00B5 75 8D FA           390 mov _TH1,#0xFA
                        391 ; hi.c 129
00B8 D2 8E              392 setb _TR1
                        393 ; hi.c 131
00BA D2 8C              394 setb _TR0
                        395 ; hi.c 132
00BC D2 A9              396 setb _ET0
                        397 ; hi.c 133
00BE D2 AF              398 setb _EA
                        399 ; hi.c 135
00C0 C2 99              400 clr _TI
                        401 ; hi.c 136
00C2 75 99 2E           402 mov _SBUF,#0x2E
                        403 ; hi.c 137
00C5 75*01 01           404 mov _hi_flag,#0x01
                        405 ; hi.c 140
                        406 ; Peephole 182a use 16 bit load of DPTR
00C8 90s00rF0           407 mov dptr,#_my_message
00CB 75 F0 02           408 mov b,#0x02
00CE 12s00r72           409 lcall _tx_str
00D1                    410 00104$:
                        411 ; hi.c 144
00D1 E5*01              412 mov a,_hi_flag
                        413 ; Peephole 110 removed ljmp by inverse jump logic
00D3 60 15              414 jz 00102$
```

```
00D5                    415 00110$:
                        416 ; hi.c 146
                        417 ; Peephole 182a use 16 bit load of DPTR
00D5 90s00rFA           418 mov dptr,#__str_0
00D8 75 F0 02           419 mov b,#0x02
00DB 12s00r72           420 lcall _tx_str
                        421 ; hi.c 147
                        422 ; Peephole 182a use 16 bit load of DPTR
00DE 90s00rFE           423 mov dptr,#__str_1
00E1 75 F0 02           424 mov b,#0x02
00E4 12s00r72           425 lcall _tx_str
                        426 ; hi.c 148
00E7 75*01 00           427 mov _hi_flag,#0x00
00EA                    428 00102$:
                        429 ; hi.c 151
00EA 12s00r9D           430 lcall _stop
                        431 ; Peephole 132 changed ljmp to sjmp
00ED 80 E2              432 sjmp 00104$
00EF                    433 00106$:
00EF 22                 434 ret
                        435 .area CSEG (CODE)
00F0                    436 _my_message:
00F0 47 4E 55 20 72 6F  437 .ascii "GNU rocks" 63 6B 73
00F9 00                 438 .db 0x00
00FA                    439 __str_0:
00FA 48 69              440 .ascii "Hi"
00FC 0A                 441 .db 0x0A
00FD 00                 442 .db 0x00
00FE                    443 __str_1:
00FE 54 68 65 72 65     444 .ascii "There"
0103 0                  445 .db 0x0A
0104 00                 446 .db 0x00
```

hi.c 的 map 文件内容，可以看出每个变量放的位置以及每个子程序所占的空间。

```
Hexadecimal
Area                  Addr    Size      Decimal Bytes (Attributes)
----------------      ----    ------    ------- ----- ------------

. .ABS.               0000    0000 =    0. bytes (ABS,OVR)

      Value  Global
--------------------------------
      0000    l_HOME
      0000    l_OSEG
      0000    l__CODE
      0000    l__DUMMY
      0000    s_BSEG
      0000    s_CSEG
      0000    s_XSEG
      0000    s__CODE
      0001    l_BSEG
```

```
      0001    l_ISEG
      0001    l_SSEG
      0001    l_XSEG
      0003    l_DSEG
      0003    l_GSFINAL
      000D    l_GSINIT
      0030    s_DSEG
      0033    s_OSEG
      0033    s_SSEG
      0033    s__DUMMY
      0080    s_ISEG
      0131    l_CSEG
      0131    s_GSINIT
      013E    s_GSFINAL
      0141    s_HOME

Hexadecimal

Area                Addr    Size      Decimal Bytes (Attributes)
----------------    ----    ------    ------- ----- ------------
_CODE               0000    0000 =    0. bytes (REL,CON)

Hexadecimal

Area                Addr    Size      Decimal Bytes (Attributes)
----------------    ----    ------    ------- ----- ------------
DSEG                0030    0003 =    3. bytes (REL,CON)

      Value  Global
--------------------------------
      0030    _timer
      0031    _hi_flag
      0032    _a_data_byte

Hexadecimal
Area                Addr    Size      Decimal Bytes (Attributes)
----------------    ----    ------    ------- ----- ------------
DUMMY               0033    0000 =    0. bytes (REL,CON)
Hexadecimal

Area                Addr    Size      Decimal Bytes (Attributes)
----------------    ----    ------    ------- ----- ------------
OSEG                0033    0000 =    0. bytes (REL,OVR)

Hexadecimal

Area                Addr    Size      Decimal Bytes (Attributes)
----------------    ----    ------    ------- ----- ------------
SSEG                0033    0001 =    1. bytes (REL,CON)
```

```
Hexadecimal

Area                 Addr    Size      Decimal Bytes (Attributes)
---------------      ----    ------    ------- ----- ------------
ISEG                 0080    0001 =    1. bytes (REL,CON)

      Value  Global
--------------------------------
      0080    _a_idata_byte

Hexadecimal

Area                 Addr    Size      Decimal Bytes (Attributes)
---------------      ----    ------    ------- ----- ------------
BSEG                 0000    0001 =    1. bytes (REL,CON,BIT)

      Value  Global
--------------------------------
      0B:0000     _my_bit
Hexadecimal

Area                 Addr    Size      Decimal Bytes (Attributes)
---------------      ----    ------    ------- ----- ------------
XSEG                 0000    0001 =    1. bytes (REL,CON,XDATA)

      Value  Global
--------------------------------
      0D:0000     _a_xdata_byte
Hexadecimal

Area                 Addr    Size      Decimal Bytes (Attributes)
---------------      ----    ------    ------- ----- ------------
CSEG                 0000    0131 =    305. bytes (REL,CON,CODE)

      Value  Global
--------------------------------
      0C:0000     A$hi$191
      0C:0003     A$hi$192
      0C:000B     A$hi$194
      0C:0013     A$hi$196
      0C:001B     A$hi$198
      0C:0023     A$hi$200
      0C:002B     A$hi$202
      0C:0033     A$hi$229
      0C:0036     A$hi$231
      0C:0038     A$hi$248
      0C:0038     _timer0_irq_proc
      0C:003A      A$hi$249
      0C:003C      A$hi$250
      0C:003E      A$hi$251
```

```
0C:0040     A$hi$252
0C:0042     A$hi$253
0C:0045     A$hi$255
0C:0047     A$hi$257
0C:0049     A$hi$260
0C:004B     A$hi$262
0C:004D     A$hi$265
0C:0050     A$hi$267
0C:0053     A$hi$270
0C:0055     A$hi$272
0C:0058     A$hi$274
0C:005B     A$hi$276
0C:005D     A$hi$278
0C:005F     A$hi$279
0C:0061     A$hi$280
0C:0063     A$hi$281
0C:0065    A$hi$282
0C:0067     A$hi$283
0C:0068     A$hi$301
0C:0068    _tx_char
0C:006A     A$hi$305
0C:006D     A$hi$308
0C:006F     A$hi$310
0C:0071     A$hi$312
0C:0072     A$hi$323
0C:0072     _tx_str
0C:0074     A$hi$324
0C:0076     A$hi$325
0C:0078     A$hi$328
0C:007A     A$hi$329
0C:007C     A$hi$330
0C:007E     A$hi$331
0C:0081     A$hi$333
0C:0082     A$hi$335
0C:0084     A$hi$338
0C:0085     A$hi$339
0C:0088     A$hi$340
0C:0089     A$hi$342
0C:008B     A$hi$343
0C:008D     A$hi$344
0C:008F     A$hi$345
0C:0091     A$hi$346
0C:0094      A$hi$347
0C:0096     A$hi$348
0C:0098     A$hi$349
0C:009A     A$hi$351
0C:009C     A$hi$353
0C:009D     A$hi$364
0C:009D     _stop
0C:00A0     A$hi$365
```

```
0C:00A1     A$hi$366
0C:00A2     A$hi$368
0C:00A3     A$hi$378
0C:00A3     _main
0C:00A6     A$hi$380
0C:00A9     A$hi$382
0C:00AC     A$hi$384
0C:00AF     A$hi$386
0C:00B2     A$hi$388
0C:00B5     A$hi$390
0C:00B8     A$hi$392
0C:00BA     A$hi$394
0C:00BC     A$hi$396
0C:00BE     A$hi$398
0C:00C0     A$hi$400
0C:00C2     A$hi$402
0C:00C5     A$hi$404
0C:00C8     A$hi$407
0C:00CB     A$hi$408
0C:00CE     A$hi$409
0C:00D1     A$hi$412
0C:00D3     A$hi$414
0C:00D5     A$hi$418
0C:00D8     A$hi$419
0C:00DB     A$hi$420
0C:00DE     A$hi$423
0C:00E1     A$hi$424
0C:00E4     A$hi$425
0C:00E7     A$hi$427
0C:00EA     A$hi$430
0C:00ED       A$hi$432
0C:00EF       A$hi$434
0C:00F0       _my_message
0C:0105      A$_startup$70
0C:0105     __sdcc_external_startup
0C:0108     A$_startup$72
0C:0109     A$_gptrget$71
0C:0109     __gptrget
0C:010A     A$_gptrget$72
0C:010C     A$_gptrget$76
0C:010E     A$_gptrget$77
0C:0110     A$_gptrget$78
0C:0111     A$_gptrget$79
0C:0113     A$_gptrget$80
0C:0114     A$_gptrget$81
0C:0116     A$_gptrget$82
0C:0117     A$_gptrget$83
0C:0119     A$_gptrget$87
0C:011B     A$_gptrget$88
0C:011D    A$_gptrget$93
```

```
      0C:011F     A$_gptrget$94
      0C:0120     A$_gptrget$95
      0C:0122    A$_gptrget$100
      0C:0123     A$_gptrget$101
      0C:0125     A$_gptrget$106
      0C:0126     A$_gptrget$107
      0C:0127     A$_gptrget$108
      0C:0129     A$_gptrget$113
      0C:012B     A$_gptrget$114
      0C:012C     A$_gptrget$119
      0C:012D     A$_gptrget$120
      0C:012F     A$_gptrget$121
      0C:0130     A$_gptrget$123

Hexadecimal

Area                 Addr    Size      Decimal Bytes (Attributes)
---------------      ----    ------    ------- ----- ------------
GSINIT               0131    000D =    13. bytes (REL,CON,CODE)

      Value  Global
--------------------------------
      0C:0131     A$hi$211
      0C:0134     A$hi$212
      0C:0137     A$hi$213
      0C:0139     A$hi$214
      0C:013B     A$hi$215

Hexadecimal

Area                 Addr    Size      Decimal Bytes (Attributes)
---------------      ----    ------    ------- ----- ------------
GSFINAL              013E    0003 =    3. bytes (REL,CON,CODE)
      Value  Global
--------------------------------
      0C:013E     A$hi$218

Hexadecimal

Area                 Addr    Size      Decimal Bytes (Attributes)
---------------      ----    ------    ------- ----- ------------
HOME                 0141    0000 =    0. bytes (REL,CON,CODE)
ASxxxx Linker V01.70 + NoICE + SDCC Feb 1999, page 1.

Files Linked [ module(s) ]

hi      [       hi ]

Libraries Linked               [    object file    ]
```

```
c:\sdcc\lib\small/libsdcc.lib        [          _startup ]
c:\sdcc\lib\small/libsdcc.lib        [          _gptrget ]

ASxxxx Linker V01.70 + NoICE + SDCC Feb 1999, page 2.

User Base Address Definitions

CSEG = 0x0000
DSEG = 0x0030
XSEG = 0x0000
ISEG = 0x0080
BSEG = 0x0000
```

5-5 学了 C 以后

对于 SDCC 的更详细的操作请参考其操作手册（User Guide），许多人义务地对 SDCC 提供意见，自动除错并报告错误，这些都是我们所应该反省与学习的。所以，我们强烈建议：如果您下载了 SDCC，并成功地应用在您的学习、研究或产品上时，请加入 SDCC 的讨论组（http://lists.sourceforge.net/lists/listinfo/sdcc-user）中，让 SDCC 能够 bug 更少且更广为大众所用。SDCC 已经声明为共享软件，让大众无偿下载使用了，你是否也有相同的胸襟？

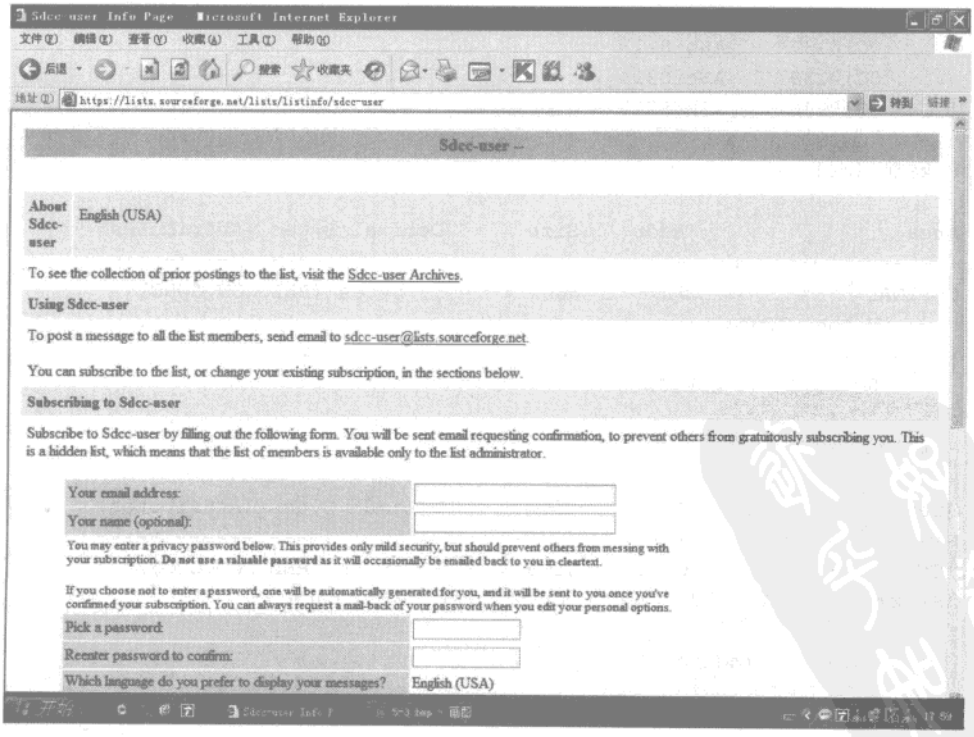


图 5-5 欢迎加入 SDCC 讨论组

本章使用的软件

Windows 版的 SDCC C 编译程序。
IAR C 编译程序。
Keil C 编译程序。
2500AD C 编译程序（2500AD 这家公司已不存在了）。
HEX2BIN.EXE：将 HEX 文件化成纯二进制文件。

相关资料网站

可经由下列公司、网站取得更进一步的信息：
http://www.chipware.com.tw：查询 8051 单片机控制板相关资料。
http://www.Intel.com：查询 8051 的硬件后续开发情况。
http://www.zilog.com：查询 Z80 CPU 的硬件后续开发情况。
http://www.atmel.com：查询 AT89C51 的硬件资料与 AVR 相关资料。
http://www.microchip.tom：查询 PIC 的相关资料。
http://www.redhat.com：下载 Cygwin 的执行程序。
http://sdcc.sourceforge.net：下载最新版的 SDCC 程序。
http://www.iar.com：查询 IAR 的 C 编译程序相关资料。
http://www.keil.com：下载试用版程序。
http://www.atmel.com：下载 8051 与 AVR 相关的共享软件。
http://www.google.com：查询 SDCC 相关的报导与文献。

6

Set Top Box 专用的高速串并行转换器，图中显示的 Bit Rate 约是 45MHz，通过仪器可以验证其误码率 BIR（bit error rate）。

第 6 章　8051 的是是非非

听过太多人讲学单片机就学 8051！也曾听过高手提过 8051 一点都不好用！8051 有许多优点，但是历经多年后，有些优点竟然变成缺点，很奇怪吧！不论如何，请看完我们的剖析，再来做决定。

8051 问世已经超过 20 年了，在市面上可以找到各式各样的 8051 变种 CPU，有的是内置 8KB 以上的 Flash Memory，有些是 RISC 架构的 8051，也有内含 100kHz 的 ADC，无论如何变化，这些 CPU 的相同点都是能够执行 Intel 8051 的机器码。原来的 8051 需要 12 个时钟周期才能执行一条指令，现在有些快速的 8051 CPU 只要 4 个时钟周期，甚至 1 个时钟周期就可以执行一条指令。平均的执行速度至少是传统 8051 的 3 倍以上。

另一方面来看，我们发现市面上 8051 的参考书籍也是最多最丰富的，许许多多的仪器设备里面都有 8051 CPU 的踪迹。以我们常看到 Atmel AT89C51 为例，称它是最佳的微处理器学习工具一点也不为过，AT89C51 内部有 4KB 的 Flash Memory 程序空间，工作频率可达 24MHz，AT89C51 的 Flash 可以通过专用的烧录器写入，而且清除再写入的次数可达一千次以上，所以一个开发项目，可能只要一颗 AT89C51 就可以担当重任，只要 IC 不要插错方向而烧毁，这颗 AT89C51 确实可以从一而终的。

上面我们谈到 8051 有如此多的优点，其实，一个 CPU 可以历久而弥新是有其道理的，我们先来看看 8051 有何过人之处。

6-1　8051 的众多优势

◆ 引脚的优势

Intel 8051 的引脚安排得太好了，8051 的 P1 端口被安排在第 1 引脚到第 8 引脚，第 9 引脚为 RESET，第 10 引脚到第 17 引脚为 P3 端口，第 18 引脚与第 19 引脚为振荡晶体输入点，第 20 引脚为 GND 接地脚。另一边则是数据与地址总线有关的引脚，而 $\overline{EA}$、ALE 与 $\overline{PSEN}$ 脚分别占用了第 31、30 与 29 引脚。引脚的布置都经过特别的安排，所以测试工程师在硬件除错时，只要稍微记下 8051 的引脚安排，即可接上电源试验，几乎不需要将线路图摆在身边。图 6-1 是 AT89C51 与 AT89C52 的引脚图。

◆ 线路与体积精简的优势

有 40 根脚的 8051 或是 AT89C51 都内置 4KB 的程序空间，只要接上电源及石英晶体，并且将 RESET 脚拉到低电位后，8051 就可以启动工作了，对许多初学或怕麻烦的人而言，确实是相当吸引人的。最近几年来流行的 AT89C2051 只有 20 根脚，少了 P0 与 P2 端口，虽然只有 2KB 的 Flash Memory，不过体积可以缩得更小。它已经是许多大学研究生数字小实验的首选。图 6-2 是 AT89C51、AT89C52 与 AT89C2051 的实物图。

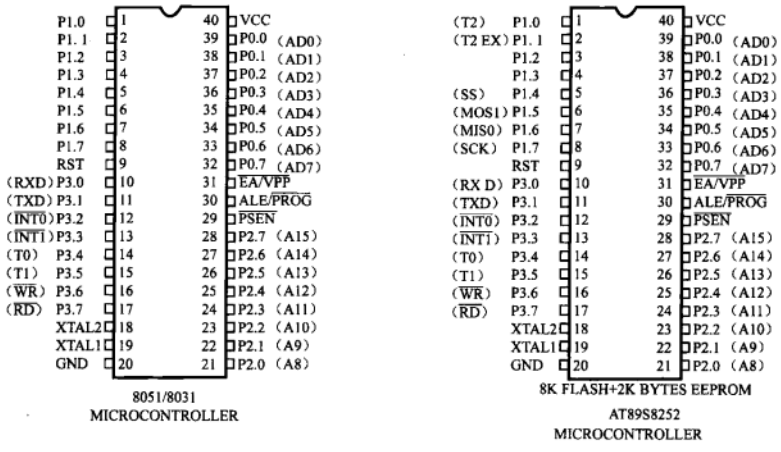


图 6-1　AT89C51 与 AT89C52 的引脚图

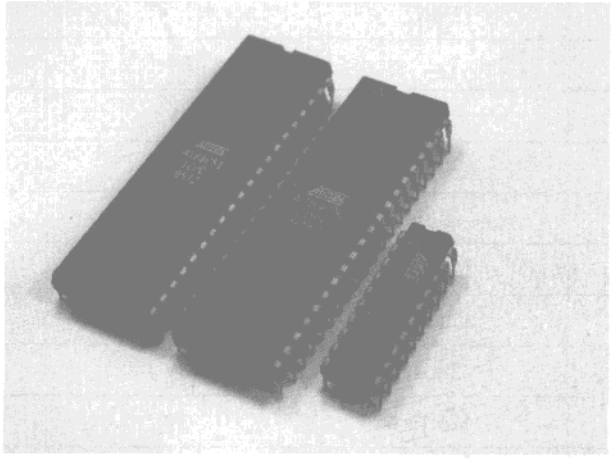


图 6-2　AT89C51、AT89C52 与 AT89C2051 的实物图

◆ 省电的优势

早期的 8051 没有所谓的 Power Down 及 Idle 模式，所以 CPU 随时都在全速工作，但是许多以电池为主的掌上型设备，却指定要有低功率消耗的功能，亦即工作后几分钟内，CPU 要自动进入睡眠状态，尽量减少电池的电量消耗。最近几年的 8051 或其变种 CPU 都已有低耗电的优势。表 6-1 是常用的 8051 省电模式比较表。

表 6-1　常用的 8051 省电模式比较表

87C51			
模式	频率	电压	电流（最大）
Active	12MHz	6V	25mA
1dle	12MHz	6V	4mA
Power Down			50μA

续表

AT89C51			
模式	频率	电压	电流（最大）
Active	12MHz	6V	20mA
1dle	12MHz	6V	5mA
Power Down		6V	100μA
Power Down		3V	40μA
AT89C52			
模式	频率	电压	电流（最大）
Active	12MHz		25mA
1dle	12MHz		6.5mA
Power Down		6V	100μA
Power Down		3V	40μA
AT89C2051			
模式	频率	电压	电流（最大）
Active	12MHz	6V	20mA
Active	12MHz	3V	5.5mA
1dle	12MHz	6V	5mA
1dle	12MHz	3V	1mA
Power Down		6V	100μA
Power Down		3V	20μA

◆ I/O 数众多的优势

8051 如果只使用内部程序空间，不要送出数据及地址总线，则其 I/O 数最多可以达到 32 个，对于一般的控制与应用，理论上都已足够。如果嫌程序空间太少，可以改用 AT89C52 或 8052，程序空间可以加倍到 8KB。但如果你是使用外部的存储空间，则能够用的 IO 数就只剩下 16 个了，即只有 P1 及 P3 端口可用。

◆ 参考资料丰富的优势

在市面上我们可以看到至少 20 种以上的 8051 变种 CPU，而且价格都在几十元上下。不过，RISC 架构的 8051 就会较为昂贵，但还是在能够接受的范围内。8051 的中英文参考书籍方面更是不在少数，只要到书店走一趟，关于单片机 8051 的书籍应该有 10 本以上，谈论的焦点都聚集在基本的电路设计及汇编语言方面。更进一步的参考资料可在许多专业的 8051 网站上取得。8051 元件随时买得到外加参考数据无虞匮乏，学习起来当然是得心应手了。

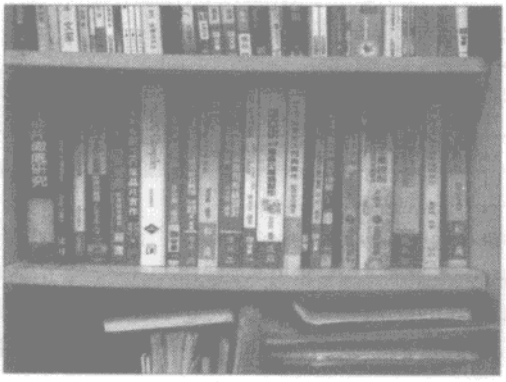


图 6-3　书架上都是 8051 的参考书籍

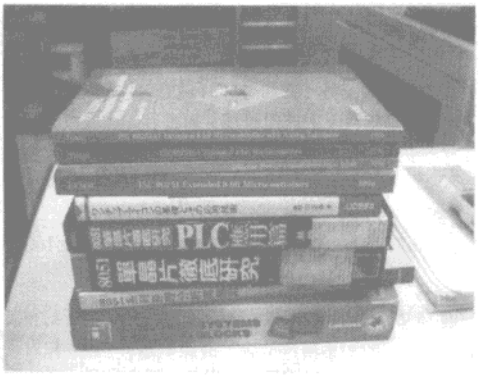


图 6-4　书桌上也是 8051 的参考书籍

◆ 产品线齐全的优势

现在我们在市面上看到的 8051 大部分是商业规格（Commercial）的，即其工作的温度范围只能从 0～70℃。其实 IC 制造厂还有工业用（Industrial）等级的 8051，工作温度可达-40～85℃，在冰天雪地或火热沙漠的气候中都可以工作。当处于如此恶劣的环境下，并不是只有 8051 正常就可以了，其他的元器件及电池都要有相同的等级才说得过去。当设计工程师遇到这类的问题时，这才会觉得一分钱一分货，如果 IC 元件任意削剪成本，就会碰到无缘无故的死机以及找不出毛病的故障点。

表 6-2　8051 商业用及工业用的等级比较表

等　　级	温 度 范 围	供 应 电 压
商用级（Commercial）	0～70℃	5V±20%
工业级（Industrial）	-40～85℃	5V±20%
车用级（Automotive）	-40～125℃	5V±20%

6-2　8051 的缺憾

8051 纵然有如此多的优势，我们用 8051 做过 100 种以上的线路设计应用，同时也有 10 年以上的应用实践，可是在开发及测试阶段还是发现 8051 有不少缺点存在，以下的负面分析相信是一般书籍所看不到的。

◆ DPTR 的缺憾

8051 有一个 16 位的指针 DPTR，当要做外部数据或 IO 的存取时，通常都要靠这个 DPTR 来指定位置。问题就出在 DPTR 只有一个，当我们的程序要移动一大块数据区域（Data Block Move）时，一组 DPTR 实在不够。如果要移动 SRAM 8000H～8FFFH 内的数据到 9000H～9FFFH，总共 4KB 的数据移动，8051 汇编语言的写法可能是以下这个样子：

```
MV4KB   MOV     DPTR,#8000H
        MOV     R0,#10H             ;R0 与 R1 是 counter
        MOV     R1,#00H
```

```
ILOOP   MOVX    A,@DPTR             ;取得第一个字节的内容
        MOV     B,A                 ;转存到 B
        PUSH    DPH                 ;把 DPH 临时存在 STACK 中
        MOV     A,#10H
        ADD     A,DPH
        MOV     DPH,A               ;DPH=DPH+10H=90H
        MOV     A,B                 ;取回原来的 A 值
        MOVX    @DPTR,A             ;存入（9000H）当中

        POP     DPH;DPH=80H
        INC     DPTR                ;DPTR 加 1 等于 8001H
        DJNZ    R1,ILOOP            ;LOOP256 次
        DJNZ    R0,ILLOP            ;LOOP16 次,256×15=4096 次
        RET
```

4KB 数据的移动程序示范马上就显示出 DPTR 真的不好用。更妙的是 8051 DPTR 对外部数据的存取指令只有 MOVX @DPTR,A 与 MOVX A，@DPTR，这也就是说，8051 没有 MOVX @DPTR，B 这类的指令，一定要通过 A 寄存器才行。这也使得读到的值要先存在其他寄存器中，等到要写入时才重新载回 A 中。如果要把 8000H～83DOH 内的数据移到 9158H 开始的位置时的写法就会更加复杂了。

◆ Data Memory 的缺憾

8051 内部的 Data Memory 有 128 字节，这段 Memory 最前面是寄存器 R0～R7 所占用的 BANK，接下来的是可位寻址区 20H～2FH，然后才是我们程序使用的数据缓冲区，最后还要保留部分空间给系统堆栈用。如果你的程序超过 16KB，128 字节的 Data Memory 空间一定是不够的。虽然 8051 已经将 Data Memory 扩充一倍到 256 字节，可是如果碰到复杂的运算时，就会捉襟见肘的。

◆ Clock 的缺憾

早期 8051 的石英晶体输入频率都在 12MHz 以内，不过每个基本指令要 12 个时钟周期才能完成，平均下来 8051 大概 1μs 执行一个指令。二十年前这种执行速度已经够快了，问题是现在许多人写的程序是用 C 编译器转成机器码，这种做法会使程序急剧膨胀到数十 KB，如果 CPU 速度不加快的话，一定会使系统的性能降低。

◆ ALE 的缺憾

8051 的 ALE 引脚正式名称为 Address Latch Enable，其输出频率约是系统振荡频率的 1/6，如果系统是使用 12MHz 的石英晶体时，ALE 的输出约是 2MHz，这根脚是用来锁存地址线的。早期的硬件设计者经常拿这个信号当其他 IC 的时钟源，但是这几年统统没有人敢这样做了。因为这样做的话，质量认证上的电磁干扰防治（简称 EMI）包准过不了关。当我们用频谱分析仪（Spectrum Analyzer）观看 8051 的高频干扰信号时，发现 8051 ALE 的谐波是很强的，如果不做压制的话，可能做出来的电子产品会有过大的干扰信号，会导致通不过质量认证标准。较进步的 8051 变种都会对 ALE 信号进行控制，有些是用软件控制 ALE 信号是否产生，有些内部已经有足够的程序空间，所以就把 ALE 信号取消掉。图 6-3 是用频谱分析仪观看 8051 的高频信号。

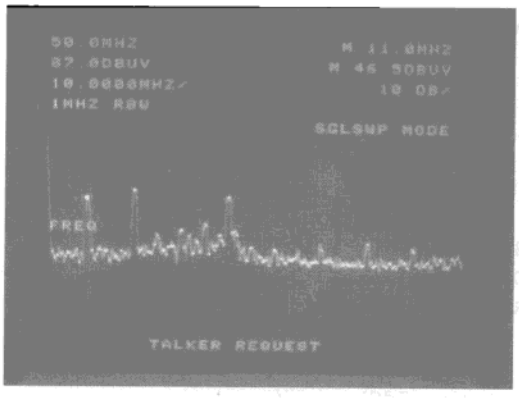


图 6-3　用频谱分析仪观看 8051 的高频信号

◆ UART 串行通信的缺憾

8051 内部的串行通信 SBUF 是许多人称赞的，共有 4 款模式可选择，Intel 在十几年前有如此的设计确实不简单。可是以现在的技术层次来看，还是有些改进的地方。以我们最常用的 model 为例，若石英晶体选定 11.0592MHz 时，最快的传输速率只达 19.2b/s，亦即每 0.5ms 要传送或接收一个完整字节的数据。这种速度已经不算快了，另一方面我们习惯用中断的方式来处理串行通信，如果 CPU 速度不够快或是串行服务程序处理太过冗长的话，很容易就导致通信上的失误。

8051 的 SBUF 只有 1 字节的长度，所以要传送出的数据要先放在 Data Memory 里，再慢慢地从 SBUF 端送出去。相反地，外部接收到的串行数据要尽快转存到 Data Memory 中，否则有可能会被接下来的数据给覆盖掉（overwrite），而且只要有一个数据未收到，更会造成接下来整串数据的错误。

◆ 死机没人知的缺憾

一个设计良好的 8051 电路是不会死机的，一个经过妥善设计的 8051 控制程序也是不会死机的。可是在前两项境界未达到之前，你的系统都会有可能死机，而且频频出错。但是死机时，8051 却没有任何硬件机制让系统重新启动，除非关机重来！虽然死机不是 8051 CPU 原始设计厂 Intel 的责任，可是若能加上 WDT（Watch Dog Timer）俗称“看门狗”的电路，就可以解除“死机无人知”的梦魇。WDT 的用法是 CPU 每隔一小段时间就要对某个 I/O 位置执行写入操作，若超过预设的时间仍然未操作时，代表 CPU 确定已经不知去向了，这时 WDT 会对 CPU 产生一个额外的 RESET 信号，让系统重新从 0000H 启动，如果你写的程序一直有问题时，就会看到 CPU 始终在执行 RESET 之后的操作。

6-3　市面上常见的 8051 CPU 变种

谈了如此多 8051 的是是非非，你觉得 8051 还可以用吗？我们的答案是正面的。纵使 8051 有一些技术上的缺憾，虽然这些弱点是事隔好几年才显现，但是它的方便性与参考性是其他 CPU 所无法比拟的。我们建议你先学会控制 8051 及其外部设备，这里有最详尽的参考数据与程序范例，你绝对可以从 8051 入门。如果想追求更快的速度与性能，其他 CPU 可能有解决方案，但你依旧可以从 8051 开始进入状态。

几款较常见变种（Variants）的8051微控器（Microcontroller）与其主要特点：

Atmel AT89C1051：1KB Flash Memory+2模拟输入。
Atmel AT89C2051：2KB Flash Memory+2模拟输入。
Atmel AT89C4051：4KB Flash Memory+2模拟输入。
Atmel AT89C51：4KB Flash Memory。
Atmel AT89C52：8KB Flash Memory。
Atmel AT89C55：20KB Flash Memory。
Atmel AT89S8252：8KB Flash Memory。
2KB EEPROM。
Dual DPTR。
WDT。

Dallas 80C320：4 clocks/machine cycle。
Dual DPTR。
WDT。
Power-fail RESET。
2 serial ports。

Winbond W78C32：
1KB SRAM。
WDT。
Dual DPTR。
2 UART serial ports。

Analog Device AduC812：
8 ch 12-bit AD。
2 ch 12-bit DA。
On-Chip temperature sensor。
8KB Flash/EE Program memory。
640 Bytes Flash/EE Data Memory。
2 wire serial I/O。
WDT。

TI MSC1201：
8 ch 24-bit AD。
4 clocks/machine cycle。
Dual DPTR。
32KB Flash Program Memory。
In-system serial programmable。
1280 Bytes Data SRAM
2KB BOOT ROM。
Dual UART。
16-bit PWM。

RDC R8034TA：
RISC架构(4 clocks/machine cycle)。
DC to 66MHz。
Dual DPTR。
Dual UART。
1KB SRAM。

```
        8ch 8-bit AD。
        Power fail Interrupt。
        2WDTs。

Acer Lab M6759:
        DC to 40MHz。
        64KB MTP-ROM。
        512 Bytes on-chip SRAM。
        software power-down mode for IDLE and STOP。
```

6-4 8051 另类观点剖析（本节内容由凌全伯先生撰写）

要说起 8051 的种种，岂是一两句话或一两本书可以交代清楚的？无论功过如何，毕竟在八进制的领域中，8051 系列的地位是无庸置疑的。在国外工程应用方面如何？手边无资料可佐证；但在 20 世纪 90 年代中期，因 8051 的专利权期满，又搭上 IC 设计业的起步，8051 在 IC 设计业中推动所谓 SOC（System On a Chip）的概念可是翘楚。

套一句电影中的台词：生命总是会找到出路的！虽然 8051 与生俱来的许多功能在因应现实应用上开始捉襟见肘，但它却是许多工程师所熟悉的系统，工作上所需求的动机远比学校学的来得强烈。况且经过这么多年的开发，8051 的开发环境（IDE，Integration Development Environment）也很完整，自然而然它便成为 IC 设计人员眼中最佳核心。

放眼 IC 设计业中最爱的核心无非是 6502 及 8051。6502 就是个人计算机先驱 APPLE II 的 CPU，也是“任天堂”的“红白机”的 CPU；它在 IC 设计优势上是：体积小，指令少。体积小是 IC 设计的命根子，您想想：一片同样 8 英寸芯片可以因每颗 IC 体积小而多出来的数量是很可观的。

如今 8051 也慢慢站上 IC 设计业中主角。除了以上所说的因素外，它的扩展性强及基本功能完整也是重要因素。IC 体积问题却因 IC 制程的不断提升而有所改善。可参考下表：

表 6-3 IC 体积问题却因 IC 制程的不断提升而有所改善

项　目	0.35μm 制程	0.5μm 制程
256×8 SRAM	217mil*mil	353.31mil*mil
8K×8 SRAM	2688mil*mil	7232mil*mil
8051 Core	1488mil*mil	5625mil*mil

注：mil*mil=645.16 μm*μm。

上表告诉我们什么？256×8 就是 8051 内部 SRAM，真正会在一颗完整功能 SOC 中占主要部分的不会是 8051。因为您可能会再加 A/D、PLL 或其他应用电路，8051 所占的比例会越来越小，而这些电路会弥补 8051 的应用困境。比如说 8051 对存储器存取速度太慢，可用 DMA （Direct Memory Access）电路来解决；没有 Watchdog？马上加一个！再偷偷地告诉你：8051 的逻辑电路约 1.2 万～2 万门电路（Gate，会因不同架构而略有差异）。跟现在一些动辄百万逻辑电路的 IC 来看，简直是小巫见大巫。另外，还有一项很重要的因素便是：因制程的不断提升，IC 的工作电压可一直下降，现在的 8051 在 IC 内部都可运行在 2.5V，甚至更低。另外，8051 可以在 IC 内部以 SRAM 为 Code Bank。因为我们都知道：SRAM 的访问速度较

EPROM 快且通过 PLL 的回路在外挂 6MHz 振荡器下，8051 可跑 24MHz 甚至 48MHz。而且因为包在 IC 内部，也可解决高频带来的电磁干扰问题，真是太完美了。不过话又说回来，当一些功能都被硬件线路所取代时，8051 要做什么呢？跑这么快做什么？（跑越快越耗电！）一般来说，留个微处理器在 IC 内部，可以保留系统应用弹性及解决硬件设计上的不足。（当然，您也可以说是解硬件的 bug。）我常开玩笑说：硬件的 bug 由软件（系统）解；系统的 bug 由软件解；软件的 bug 由业务解（改客户产品规格啊！）；业务的 bug 由老板解——因为已经不是工程问题了。

不过，您一定会问我说：既然如此，为何市面上还在卖一些标准 8051 呢？因为毕竟所谓 SOC，也算是 ASIC（Application specification IC）的一种，其系统的应用会被许多硬件规格所限制。况且我刚刚提到，因 IC 制程的提升，一不小心可能会有一缸子的 IC 产出，您又不用我要卖谁？而标准 8051 摆着自然不会有人嫌弃。

再回到主题，在网络上常常有人会问：我还学 8051 吗？从以上的说明您自然会明白，不是 8051 消失了，只是它以不同面貌躲在不同产品内，您可以这样的想象：过去您看到的 8051 系统板上面有可能将译码电路、内存、微处理器、多任务器及 A/D 或 D/A 等复杂电路都缩到一颗 IC 内部而已，甚至在一些特殊应用上，因为 8051 的数学运算能力不足，还会搭配 DSP（Digital Signal Processor）应用，（我这里提的不是 TI 公司那种 DSP，而是一些简易的 DSP。哪一家的？我想只要您是内行人自然会清楚。）但真正会发挥整个 SOC 性能的，便会是这个 8051，当然通过系统程序的开发，它会肩负起系统监督功能并且让整个系统更有弹性，更强大。

其实以 8051 为核心开发一颗 SOC 是蛮不错的想法，因为严格来说 8051 也算是半颗 SOC，它不像一颗 8088 或 6502 只是纯粹的微处理器（Micro-Processor）；而是一颗微控制器（Micro-Controller），因为它内部除了基本的算术处理及逻辑运算外，还包括了计数/定时器、UART 等功能。对 SOC 初步开发来说，8051 可以打开一初步功能验证，这就是 8051 成功的条件之一。

8051 有哪些功能是不错的呢？哪些功能是不方便的地方呢？

◆ P1 端口

8051 的 P1 端口大概是所有工程师公认最佳的 I/O 端口。它不只可以直接在硬件上输出值；也可以输入硬件上的状态，又可以位寻址及运算（bit），为系统 DEBUG 最佳工具。一般 GPIO（General Purpose I/O，多任务功能 I/O 端口）都还得定义 I/O 方向方能使用，您看 8255 这颗 IC 的功能或数据便知道了。

◆ 位寻址及逻辑运算能力

对于小而美的系统来说，实在是完美的应用环境，虽然现在系统或许不需那么斤斤计较，但对于简单的逻辑运算来说，我可以以位声明时又何必浪费用字节声明呢？另外，如果您能用字节搭配位使用：如共享同一地址来分别定义参数及逻辑运算，您会发现您的系统真是精简有力。

◆ Timer 及内部中断功能

8051 多变化的 Timer 功能及搭配内部中断，使得 8051 不再只是一颗单纯的微处理器，通过 Timer 设定而产生的分时计算功能，可以使得 8051 略具有实时多任务核心（RTOS）的能力。其实，当年 8051 的出现与汽车的引擎控制（包括燃油喷射或点火正时控制）是不可分的，您甚至可以说它就是为了汽车引擎控制而诞生的（因为汽车工业是工业的“火车头”）。

而引擎控制的基本功能就是 Timer/Counter 的应用，只是很可惜的是它这部分的功能没再继续强化而在那个产业生存下来，而让 Motorola 的 68HC11 系列所取代。

◆ 数据存储器与程序存储器分开处理

这一点就让 8051 有着优异的扩展能力，如今 8051 的程序内存都不只支持 64KB 而已，Code Bank 技术使得 8051 的程序开发很容易地不再局限在仅有的 64KB，现在的 8051 系统程序都可以写到 MB 级的，对于高级语言（如 C 语言）的方便性自然彰显，大大缩短了系统开发时间，对争取产品上市时效性有着正面帮助，也自然受大老板们的赏识。另外，在数据存储器方面，因为原来的 8051 所保留的 SFR 空间有限，使得在系统扩展性上受限制，现在的 IC 设计者便将脑筋动到数据存储器这一块身上。通过特定数据存储器地址的存取设定，可以搭配其硬件设计创造出原来 8051 所不具有的特殊功能。

◆ 指令周期时间过长

在标准的 8051 中，每个指令周期需花 12 个系统周期（Clock），所以，像简单的这种 SETB C 指令，执行时间是 1μs（以 12MHz Clock）而言，以现在外部外围的 IC 来看，实在跟不上。所以，现在大部分的 IC 设计公司均将这部分属于 8051 内部 Clock 操作给精简下来了，所以会有所谓的 RISC-8051 出现。

其实说到这里来看，要细数 8051 的是与非也是很难下一个结论。对 8051 系统的设计与应用原本就是一种艺术，艺术就是很难断定优劣或掺杂个人判断因素。但若将思考层级向上提升，比如说，设计一个 8051 应用系统不难，但要从设计过程中或其结果去创新或创造价值才是一位优秀的系统工程师所必须面对与思考的。未来不是只能靠制造或劳动力密集来求生存；而是需要不断的产品创新。想想当今有多少产品规格我们是走在前端制订者？如 USB、IEEEl394、PCMCIA 等。我们都只是一群实现者而非创造者，而执行者在市场上是很容易被取代的。这是值得我们警惕的！

接下来要谈到有关写系统程序的一些经验谈。或许是科技的不断进步，MCU 越跑越快，快到不用去在乎执行效率；内存越来越不值钱，所以写程序也越来越不重视程序的精简。不知是硬件技术宠坏了系统工程师，还是系统工程师逼得硬件技术不断升级，已经不得而知了。但确定的是：系统工程师越来越不会斤斤计较了，其实也不能怪他们，在这“十倍速的时代”里，谁能快速反应市场的谁就是最大赢家，只是大家没想到，当大家都能快速反应时，您致胜的因素是什么？

举个简单的例子：有某个影像产品设计例子，是需一组 A/D 将影像传感器信号捕获，通过USBl.1 传输进入PC，于是大家都标榜SOC 设计能力的优异性，于是设计了一组拥有 10MHz 转换速率的 A/D，标榜高影像品质的 12 位 A/D 转换器，再搭配 8051 以 DMA 方式来存取控制，看来是不错的设计。您是否知道在这设计中犯了什么错误？因为 USB1.1 的传输率最高为 12Mb/s（=1.5MB/s），答案是整个系统的瓶颈（Bottleneck）在 USB 而非 A/D。在 SOC 中 A/D 的面积不是用逻辑运算计算的，且 A/D 的设计是最富挑战，风险成本相对较高，如此一来，整个系统无形中背负了多少成本？万一这颗 SOC 月产值 100K 以上。

或许您觉得这个例子太远了，我再举一个简单一点的：一个简易的系统程序需要用到圆周率 3.14 的计算。您要如何写这样一个程序？大多数的人会辛苦一点写个：16/8 除法器子程序（因为可以用乘 314 再除以 100）；懒惰一点，会去调用个高级语言的子程序，没有人说您错了。如果是我写呢？

答案是：

参量×201 再右移六次=参量×201/64（201/64=3.140625）。

我上面提到过在 IC 内部除了 A/D 会占空间，除此之外便是内存，不论是程序存储器或数据存储器，程序越短越省钱。而这样的推算只要您事先在纸上推算过便可以。往往一些工程师都会匆匆忙忙急着上机写程序，辛辛苦苦的写完交差就了事，最后还埋怨系统资源与架构不足，却忽略基本的算法，讽刺的是，这个算法只是小学便教过的。

这样的说明是要告诉您：您的系统对象 8051 是八位世界，您就得以八位的思考逻辑来应用它，才能发挥它的最大效率；而不能老是用人类的十进制思考，要 8051 帮您做事，那你就得付出代价。

那又要如何做才是所谓“八位的思考逻辑”？

您还记得学校还曾教过您一个东西吧？无因次化（non-dimension）或许您也可以称为 Scaled 化。对八位来说它无因次化的基底便是 255。

写程序大家都会用一些#if...#endif 来让系统编译上更能符合不同的应用场合，虽然站在系统的维护与开发来说是不错的观点，但您应该也发现，在许多 MCU 的应用场合中，面对的是工程问题（或应该说自然物理现象）而非是纯粹的数学问题。在学校的工程数学教材中会教您说自然工程问题可以用数学表示，再以数学的方法解决！但大家毕业了，就还给老师了。其实，在写一些 MCU 的系统程序，尤其是讲求效率与解工程问题时（像有 A/D 这种代表着有自然工程问题），最适合引用这种观念。

所以，当您要开始写 MCU 应用程序前，您曾经细细地分析过您的系统工程问题吗？您怎么设计您的系统程序，尤其是在数学应用方面，曾真正地搭配或考虑过“它”是八位的吗？现在，我就用一个简单的例子来说明这种技巧。

要设计一个引擎点火控制器，它的点火角度范围为+30°（或是说有 60°范围）准确度要求为 1°以内，转速范围为 6400r/s 以内。那您要如何以 8051 处理这样的工程问题？您看到是 60，1，6400 等这些数字。您要如何让 8051 有效率地处理这些工程数字（尤其是那种百分比的东西）？

方法一：以一个字节声明 60 这个参数，它的精确度为一个字节，刚好是一度。合理！再以一个字符定义 6400。也合理！

方法二：将 60 无因次化成八位环境，所以 0～60 会对应到 0～255，它的工程精确度当场可提升到 60/255=0.235<1；同理，转速范围可定义为：0～255，每个值对应为 25，这个数字范围在转速界限的判断上足够了。除了相对的准确性提高外，无论是内存的使用与运算速度都明显有效率多了。而在真正的数学运算中，因为您事先的推导程序，您的 MCU 根本看不到物理数字，只需处理八位或是十六位即可。您要做的前期工作就是帮 8051 先做一次无因化工作，这样程序自然容易维护与精简，更不用说它的执行效率了。

详细的推导在此不详述！我就用一个简单的一维内差公式范例来得到所谓真正的物理量：而 MCU 根本没用到复杂的高级运算，您只要稍做修改一下，在没用到浮点运算却可得到小数点数字（函数的输入为 12 位 A/D 值），您千万不要用学校或补习班教您的那一套内差公式！记得：您的 MCU 是八位的 8051，不是 Pentium 4！

程序 1　引擎点火控制器的角度检测程序范例

```
Uint temp_cvt(Uint adc_v)
{
    Uchar cnt_offset;
```

```
    Uint offset;
    Uchar degree_cnt;
    Uchar cnt1;

    Uint BT_table[15]={
                        3795,           /* 0 deg C */
                        3729,           /* 5 deg C */
                        3654,           /* 10 deg C */
                        3567,           /* 15 deg C */
                        3470,           /* 20 deg C */
                        3362,           /* 25 deg C */
                        3242,           /* 30 deg C */
                        3114,           /* 35 deg C */
                        2977,           /* 40 deg C */
                        2832,           /* 45 deg C */
                        2682,           /* 50 deg C */
                        2528,           /* 55 deg C */
                        2372,           /* 60 deg C */
                        2213,           /* 65 deg C */
                        2063 };         /* 70 deg C */
    cnt1=0;
    while((cnt1<=14)&&(BT_table[cnt1]>=adc_v))cnt1++;

    if(cnt1= =0)
       return 0;
    else if(cnt1 = =15)
       return 70;
    else
    {
       cnt1--;

       cnt_offset=(unsigned char)(((BT_table[cnt1]-BT_table[cnt1+1])*2/50);
       if(cnt_offset & 0x01)
      {
          cnt_offset>>=1;
          cnt_offset++;
      }
       else
          cnt_offset>>=1;

       degree_cnt=0;
       offset=BT_table[cnt1]-(unsignd int)cut_offset;
       while(offset>=adc_v)
      {
          offset-=(unsigned int)cnt_offset;
          degree_cnt +=1;
       }
        return(cnt1*50+degree_cnt);
    }
}
```

您知道吗？一辆四汽缸的高级轿车的引擎管理系统（包括点火正时、多点燃油喷射等），它的 MCU 也是八位，不稀奇的是它是八位，稀奇的是它的主程序（含 PID 控制、二维内差公式、自然对数滤波公式等）不到 32KB。而每几年甚至每年都有引擎改款需动到系统程序，您该如何维护呢？

其实，这些问题的发生一点都不意外，一来刚毕业的工程师较没经验；二来或许计划实在太赶了，也牵扯太多人的工作范围，所以无法考虑周密就匆匆忙忙上机开发系统，没有太多的事先详细规划，尤其是在数值分析方面。

最后就下一个简单的结论，就学习一项技能来看，您学会单片机的应用，无论是硬件或软件，是您的努力所该得的，也是学校该教给您的。但您无须因此而自满，因为每年每位毕业的年轻学子都会拥有的，无论您是置身于园区工作环境中或从事其他相关研发工作，您们的竞争优势绝不是来自一个优异的工作者，或许您是“天才”，但我们的解释是：“天才”是只跟上天讲话的人，而“人才”是跟大家沟通的人。不断地集思广益，发挥团队的创新能力，才是您致胜的条件。您要追求的是如何利用这项技术来创造价值，无论是对自己或您的组织或老板，尤其是研发团队，当今竞争力是来自结合各项资源的团队，单打独斗的时代过去了，或许您会羡慕园区高科技的股价，但只要您能花点心思去探索一下他们的竞争力，您便能很容易发现这个中的道理。

一个简单 8051 您觉得功能不足，当您发现人家利用 IC 设计技术突破种种限制时，或许您会有点恨铁不成钢的感觉，但是您也不需太心灰意冷，因为您还是最幸福的，因为有太多前辈走在您的前面跟您分享他们成功的经验，主要还是您要如何发挥您的创意？因为创意是永远不嫌晚的。

另外，在工程技术方面，您不必一味地追求高深莫测的程序技巧，也不必利用一些花哨的硬件来彰显什么，往往能出奇制胜都是一些让人出乎意外的简单观念。数值分析是一项重要的程序技巧，它只是利用简单四则运算来达到快速又可接受的答案。计算机也不过是协助您计算的工具，真正能解释答案，评估产品优劣还是人。套一句广告词作为结束语：科技始终来自于人性。

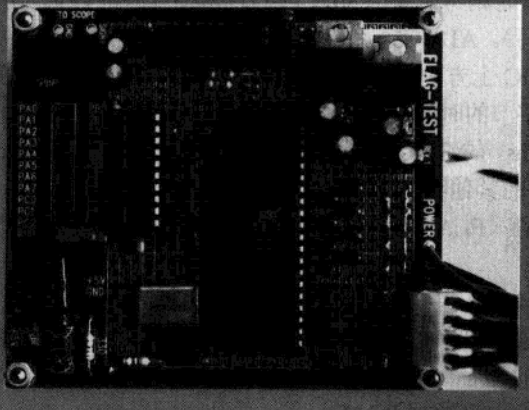
相关资料网站

可经由下列公司、网站取得更进一步的信息：

http://www.chipware.com.tw：查询 8051 单片机控制板相关资料。
http://www.intel.com：查询 8051 后续的开发情形。
http://www.atmel.com：查询 AT89C 系列 8051 的详细资料。
http://www.dallassemi.com：查询 80C320 系列资料。
http://www.rdc.com.tw：查询高速 8051 单片机资料。
http://www.ti.com：查询 8051 相关单片机资料。
http://www.analog.com：查询 8051 相关单片机资料。
http://www.winbond.com.tw：查询 8051 相关单片机资料。
http://www.philips.com：查询 8051 相关单片机资料。
http://www.infineon.com：查询 8051 相关单片机资料。
http://www.acer.com：查询 8051 相关单片机资料。

http://www.idrc.com.tw：查询精密测量仪器资料。
http://www.tek.com：查询精密测量仪器资料。
http://www.agilent.com：查询精密测量仪器资料。
http://www.fluke.com：查询精密电表及校正器资料。
http://www.advantest.com：查询精密测量仪器资料。

7



试做的高速AD转换电路，ADC是BB的产品，而控制芯片则是AT89C51。

第 7 章　单片机新成员 AT89C51 介绍

虽然 Intel 公司早已退出 8051 系列的开发竞技场，但有不少 IC 制造厂继续为 51 系列单片机开发后续产品，这章我们要讲述一个与 8051 单片机类似的 CPU：AT89C51，不过随着半导体制造技术的进步，ATMEL 公司也推出程序容量更大的 AT89C52 和 AT89C55。

如果您够仔细的话，应当会在每个 8051 单片机上发现“(c) 1980”等字样，这代表着 8051 早在 20 多年前就已被开发出来，但是初期价格较为昂贵，设计工程师还是偏好价位较低的 8048 与 8049 单片机。不过最近几年，8051 系列单片机的价格下降，使用上的方便性不比 Z80（八位 CPU 的长青树）差，所以其占有率正逐年增加中。经过了这么一段不算短的时间后，虽然 Intel 公司早已退出 8051 系列的开发竞技场，但仍有不少 IC 制造厂继续为 51 系列单片机开发后续产品。这节里我们要先讲述一个与 8051 单片机类似的 CPU：AT89C51，其主要功能仍与早期的 8051 相同，但是又增加了许多程序设计者梦寐以求的特殊功能。

7-1　AT89C51 内含 4KB 闪存 FLASH MEMORY

AT89C51 是由 ATMEL 公司开发出来，主要诉求是取代先前的 8051 和 8751，8051 须在工厂内做好 ROM MASK，不符合当今产品少量多样的需求。而 8751 内含 4KB EPROM，虽可由一般烧录器（EPROM PROGRAMMER）将程序加载，但是程序数据清除时，须将 8751 放在紫外灯下照射约半小时，才能把程序区内的程序代码全部变成 FFH，有一点方便又有一点不方便；另外 8751 的价格至少是 8051 的十倍以上，使用上始终觉得划不来，所以 8751 最常用在产品初步开发阶段，量产时则考虑采用 8051 或 OTP8051（只能烧录一次的 8051，OTP 为 One Time Programming 的缩写）。AT89C51 系针对上述缺点开发完成，其主要特点是提供内部 4KB 的 FLASH MEMORY，IC 上方没有圆形的玻璃天窗，烧录与清洗的时间非常短，依据厂方公布的数据看来，烧录 1 字节的时间小于 110μs，所以烧录 4KB 的时间约是 6s 上下，4KB 的程序数据清除时间只要 10ms，的确相当快速，价格也不贵，原厂保证一定低于 OTP8051 和 8751。

接下来是 AT89C51 的其他特点：

（1）4KB 可编程闪存（FLASH MEMORY），容许 1000 次左右的写入/清除周期，数据保存时间可达十年。

（2）完全与 Intel 的 MCS-51 兼容。

（3）外接石英晶体最高频率可达 20MHz，视该 IC 号码中最后的编号而定。

（4）共有三个阶段的程序数据保护功能，较 87C51 还多一道防护措施。

（5）共有 128 字节的内部数据 RAM。

（6）提供 2 个 16 位的定时计数器。

（7）提供 5 个中断源。

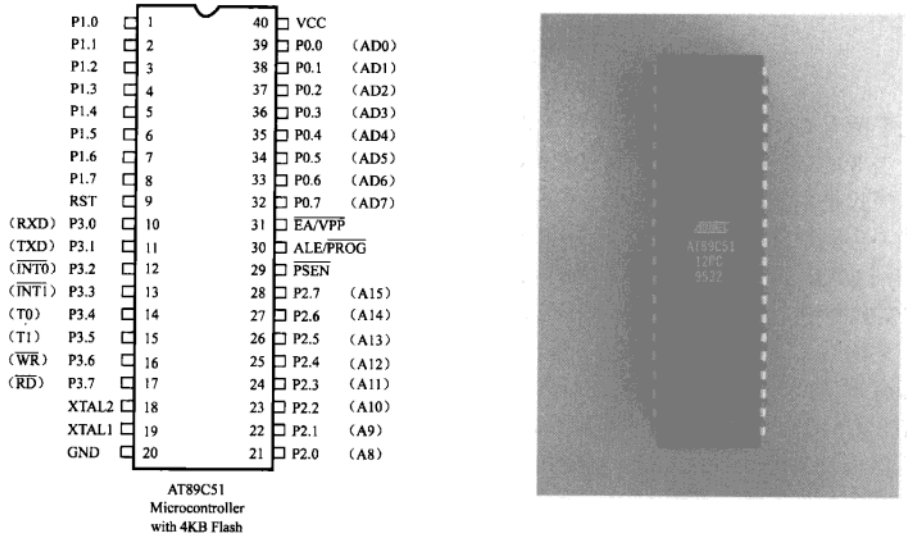


图 7-1　AT89C51 与 8751 的引脚与功能完全兼容

（8）提供可编程串行通信能力。

（9）CMOS 制程制造出来，所以有 LOW POWER IDLE 与 POWER DOWN 功能，颇有能源环保概念。

AT89C51 使用上几乎与 8751 或 87C51 相同，只要把原先的程序转到 AT89C51 上即可。大部分的万用烧录器都提供相关的烧录驱动程序，烧录时，一定要确认程序正确后才开始，否则结果会很惨的。数据保护方面，AT89C51 内部有三个 Lock Bit，分别是 LB1、LB2 和 LB3（请参考表 7-1）。

当三个 LB 都未设置时，我们随时可以通过烧录器或其他硬件线路来读取 AT89C51 内部的程序内容。而当其中的 LB1 被设置后，系统就禁止程序通过 MOVC 指令读取内部的 4KB 程序内容值。当 LB1 与 LB2 都被设置时，除了禁止 MOVC 指令的操作外，也同时禁止通过烧录器作程序内容验证（Verify）的操作。这也是说使用者无法通过程序将 AT89C51 内部的数据送出，也不能用烧录器强迫 AT89C51 进入 Verify 模式而读回其数据。当三个 LB 都被设置时，AT89C51 只能在内部 4KB 的程序空间内执行，并且完全无法读到其中的程序数据。单片机开发项目中的一个制作单元：在 FLAG 数字显示器上，就采用 AT89C51 作控制元件。不过，我们偏重在学习与程序控制上，所以三个 Lock Bit 都未上锁，有兴趣的读者都可以通过万用烧录器读到内部程序的所有内容。

表 7-1　AT89C51 三个 Lock Bit 的设置

LB1	LB2	LB3	作　用
×	×	×	可任意读取数据
O	×	×	禁止程序利用 MOVC 指令读取数据
O	O	×	禁止烧录器作程序内容验证的操作
O	O	O	完全无法读取程序数据

7-2 IDLE 与 POWER DOWN 模式

几乎所有 CMOS 类的 8051 单片机都有 IDLE（空闲）与 POWER DOWN（关机）的模式设置。当 89C51 一进入 IDLE 模式时，CPU 就开始休眠（SLEEP），但外围端口仍然继续工作。本模式是由程序控制后进入，之后芯片内的 Data Memory 与 SFR 值都不变，CPU 休眠时，就不再执行任何指令，直到有中断发生或是硬件 RESET 信号出现。AT89C51 提供的数据中提到 RESET 信号出现之后，程序并不是从 0000H 重新执行，而是由刚刚休眠指令后的地方开始继续执行，此时系统仍在 RESET 后阶段，所以禁止所有端口的写入操作，为了防止非预期的端口写入操作，在 CPU 醒来后被立即执行，原厂建议烧录器不要在 IDLE 指令之后立即有端口或外部 MEMORY 的写入（WRITE）指令。

AT89C51 进入 POWER DOWN 后，系统的振荡电路会中止，程序执行到 POWER DOWN 指令后停止，CPU 内部的 Data Memory 与 SFR 都保持原先的值，系统到此完全静止，所以耗电量减到最低（最低可到 120μW），只有硬件的 RESET 能让 CPU 重新工作。不过，RESET 之后 SFR 值会被更改成 RESET 后的默认值，但 Data Memory 值则不变。表 7-2 列出 AT89C51 的 IDLE 与 POWER DOWN 时各个端口的状态。

表 7-2 AT89C51 的 IDLE 与 POWER DOWN 的比较表

MODE	PROGRAM MEMORY	ALE	$\overline{\text{PSEN}}$	PORT0	PORT1	PORT2	PORT3
IDLE	INTERNAL	1	1	DATA	DATA	DATA	DATA
IDLE	EXTERNAL	1	1	FLOAT	DATA	ADDRESS	DATA
POWER DOWN	INTERNAL	0	0	DATA	DATA	DATA	DATA
POWER DOWN	EXTERNAL	0	0	FLOAT	DATA	DATA	DATA

7-3 如何设计 AT89C51 内部的闪存

刚出厂的 89C51 内部是处于 ERASE 模式，这代表 4KB 的闪存都是 FFH，并且立即可烧录数据。为了有更好的互换性，AT89C51 的烧录界面可以同时接受两种烧录电压（+12V 或 +5V），前者是适用于大部分的 8751 烧录器，后者只使用标准+5V 电压，烧录时就更为方便了。AT89C51 出厂时，内定是高电压（+12V）的烧录模式，89C51 烧录时的线路请看图 7-2，而其详细烧录程序如下：

（1）输入欲烧录的地址放在 AT89C51 的地址线（ADDRESS LINES）上。

（2）输入欲烧录的数据放在 AT89C51 的数据线（DATA LINES）上。

（3）启动相关的控制信号（CONTROL SIGNALS）。

（4）如果是高电压烧录时，将 EA/VPP 线提升到+12V；反之，只提升到 5V。

（5）在 ALE/$\overline{\text{PROG}}$ 脚上送入脉波信号就可烧录一个字节或一个 Lock Bit，AT89C51 会自行控制（SELF_TIMED）每次烧录的时间小于 1.5ms，重复 1～5 的操作直到所有数据都烧录完为止。

图 7-3 是 AT89C51 烧录时，各个引脚的相关图。

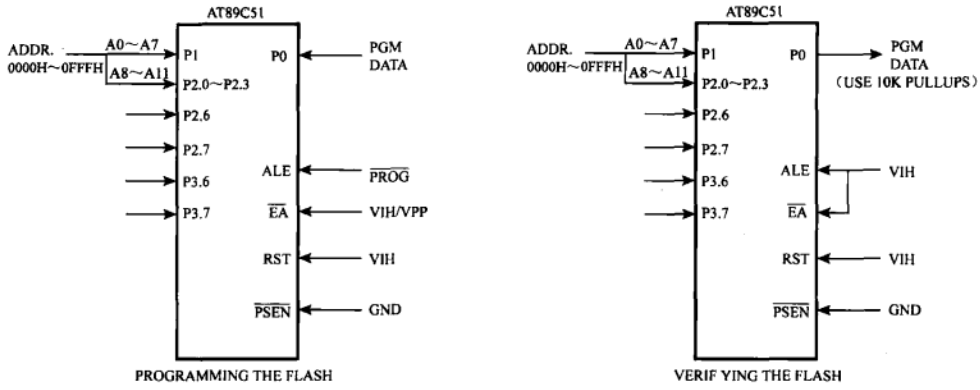


图 7-2　AT89C51 烧录与验证的线路

FLASH PROGRAMMING MODES

MODE	RST	PSEN	ALE/PROG	EA/V_{PP}	P2.6	P2.7	P3.6	P3.7
WRITE CODE DATA	H	L	⎍	H/12V (1)	L	H	H	H
READ CODE DATA	H	L	H	H	L	L	H	H
SELECT Vpp CODE (2)	H	L	⎍ (3)	12V	L	H	H	H
WRITE LOCK BIT-1	H	L	⎍	H/12V	H	H	H	H
BIT-2	H	L	⎍	H/12V	H	H	L	L
BIT-3	H	L	⎍	H/12V	H	L	H	L
CHIP ERASE	H	L	⎍ (3)	H/12V	H	L	L	L
READ SIGNATURE BYTE	H	L	H	H	L	L	L	L

图 7-3　AT89C51 烧录时各个引脚的相关图

注：（1）Vpp Select code 设置完后，就决定烧录时 Vpp 的电压。

（2）把数据 AAH 写到地址（55H）内时，选择 Vpp=Vcc 即+5V 电压。

把数据 55H 写到地址（AAH）内时，选择 Vpp=+12V。

（3）AT89C51 内部程序一次全部清除时，PROG 降为低电位的时间要有 10ms 之久。

虽然 AT89C51 的烧录过程正如上面所谈的简单，但是原厂还是提出有几点要注意的：

- 选定 V_{PP} 烧录电压：设置烧录电压时，ALE/PROG 脚要持续低电位 10ms。
- DATA 验证（DATA POLLING）：当烧录一个字节的数据时，在输入数据的 P0 端口上会输出原先数据的反相值，当数据已被正确写入闪存时，P0 端口则输出正确的输入值，代表可以做下一个字节的数据输入。
- READY/BUSY：烧录时亦可监视 P3.4 的 READY / BUSY 状态，以观察是否可继续下个字节的烧录。每次烧录时当 ALE 脚回升到高电位后，而 P3.4 脚若仍在低电位则

代表 BUSY，烧录程序要一直等到 P3.4 重新升到高电位时，代表数据已确实进入 FLASH MEMORY 中，才可以继续下个字节的烧录，标准 8751 的烧录 READY $\overline{\text{BUSY}}$ 验证即采用此一模式。

- 程序验证（PROGRAM VERIFY）：当 4KB 的数据全部写入闪存内后，可以将 AT89C51 设成程序验证的模式，由烧录器逐一判别数据是否正确。不过，如果事先若已把内部的 Lock bit 设置时，就无法读到闪存的所有内容了。
- 数据清除（CHIP ERASE）：AT89C51 内部的闪存是使用电气法消除内容，只要组合出正确的控制信号，并把 ALE/$\overline{\text{PROG}}$ 脚拉成低电位 10ms 后，即可一次全部清除完毕。烧录前一定要执行 CHIP ERASE 一次，否则烧录时会有错误出现。
- 读取 AT89C51 的身份码字（SIGNATURE BYTE）：当烧录器处于程序验证模式时，若把 P3.6 与 P3.7 降为低电位，并读取 030H 与 031H 两地址的数据，即可得到该芯片的身份码（SIGNATURE），这种做法在于区分制造厂商与型号，与程序的操作完全无关。AT89C51 的 SIGNATURE 有两个字节，分别是：

（030H）=1EH 代表该芯片由 ATMEL 制造。

（031H）=51H 代表该芯片属于 89C51。

7-4 AT89C51 烧录器的使用

烧录 AT89C51 最方便的方法是使用基于 PC 的万用烧录器（UNIVERSAL PROGRAMMER），像“三顺”、“河洛”、“新华”及“崇贸”等等。国外制造烧录器的始祖 DATA IO 也可以烧录，不过一台超过几万以上的身价，恐怕只有大型研究单位才负担得起。

一般的烧录器价格则维持在几千元间，视功能与烧录速度而定。对一般电子公司而言，一般烧录器的价格尚可接受，若对学生或初学者而言，只单纯烧录 AT89C51 或 EPROM 时，就显得较不能接受，所以我们也打算设计一块单纯烧录 AT89C51 的烧录卡，价格尽量压低以供应单片机的初级学习者；除此之外，也一并将烧录程序的写法公开。图 7-4 是初步的规划线路，只用两个 8255 就完成所有烧录 AT89C51 的控制信号，适用于 FLAG51 控制卡或 PC 的 8255 卡，线路本身是非常简单的，剩下的就看烧录程序如何做了。

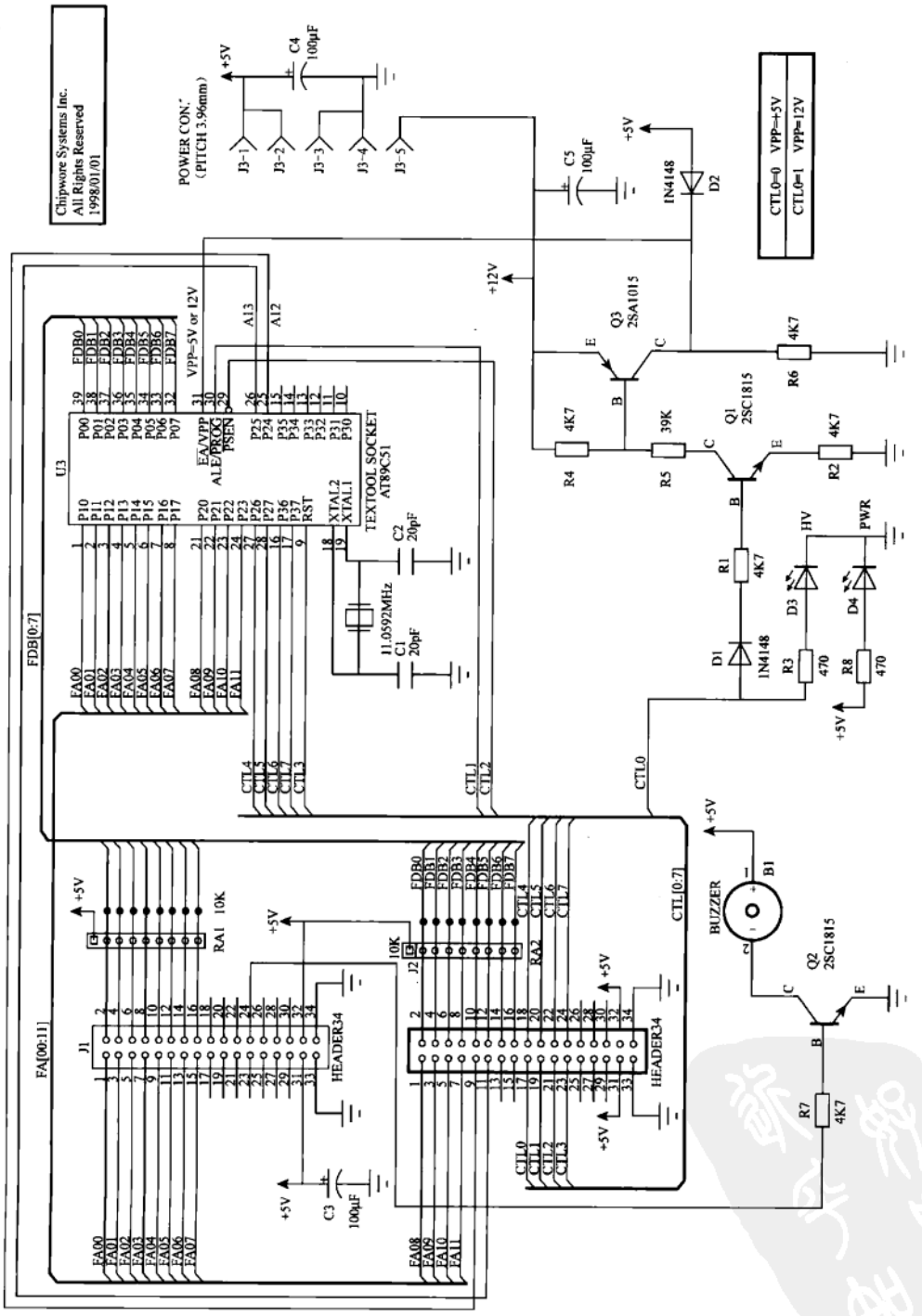


图 7-4　AT89C51 的烧录示范线路，本线路尚未完全确定，仅供参考试验

本章使用的软件

（1）2500AD 8051 Assembler。
（2）IAR 8051 C Compiler。

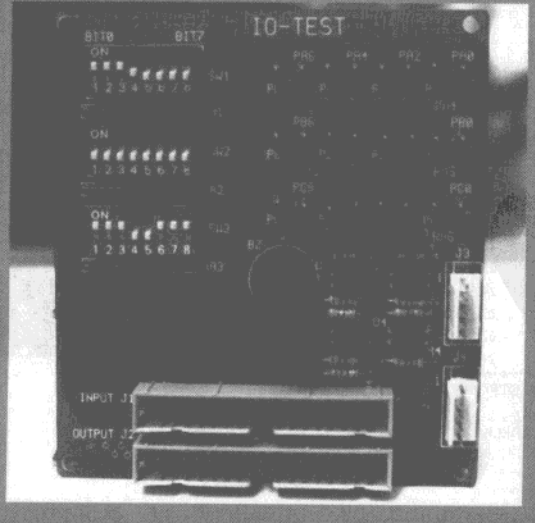
本章使用的硬件

（1）FLAG51 单片机控制板。
（2）AT89CXX 烧录板。
（3）AT89C51/AT89C52。
（4）Z80 CPU。
（5）“河洛”万用烧录器。
（6）DATA I/O 万用烧录器。

相关资料网站

可经由下列公司、网站取得更进一步的信息：
http://www.chipware.com.tw：查询单片机控制板相关资料。
http://www.atmel.com：查询 AT89C 系列 Microcontroller 的烧录资料。
http://www.zilog.com：查询 Z80 后续的开发情况。

8



FLAG51 专用的 8255 输入状态监视板。

第8章　8051单片机新成员AT89S8252介绍

在微电脑控制器上，设计者习惯将设置数据（setup）存在E^2PROM中，最常见的串行E^2PROM是93C46/C56/C66之类，但是若能在CPU内部就保有E^2PROM的功能，对程序设计者而言就更加方便了，ATMEL的AT89S8252正是兼具8KB Flash及2KB E^2PROM的8051型的CPU。

8-1　新CPU——AT89S8252的尝试

写底层程序最怕的就是更换CPU，因为换CPU就好像改朝换代一样，所有的子程序都要从头除错，这样的工程是相当浩大的，可是如果新的CPU真的有许多优异的功能，这就不得不迫使我们向代理商取得样品，以便对该芯片做进一步的研究。以我们亲身的经验为例，使用Z80 CPU的期间超过十年以上，当然也有部分时间使用8048与6502，但总觉得Z80比较顺手。稍后我们又花了一年的时间将CPU的对象由Z80转到8051上，虽然使用C语言的机会较多，但是若不同时懂得汇编语言，一定会被C的编译程序耍得团团转，而且也无法有效地使用CPU提供的中断服务支持。

图8-1是AT89S8252引脚图与实物图。

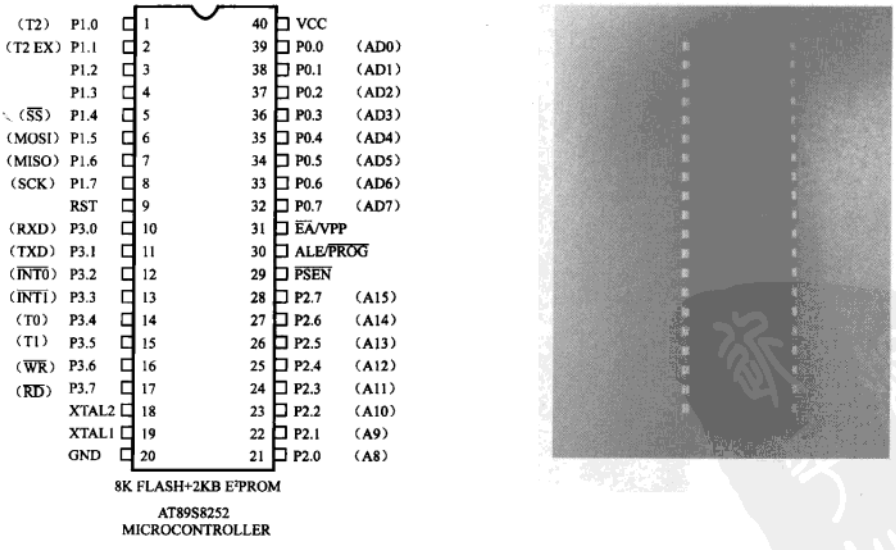


图8-1　AT89S8252引脚图与实物图

话虽如此，碰到新的CPU我们还是有兴趣想去试一试。最近我们就拿到一颗Atmel的单片机CPU样品，编号为AT89S8252，基本上它也是一颗与8052完全兼容的CPU，其石英晶

体的工作频率可以达到 33MHz，这个速度是原先 8051 的 2.75 倍。其内部有 8KB 的 Flash Memory 程序空间与 256 字节的 Data Memory，这些都和 8052 相同。程序代码的烧录可用一般的 Flash Memory 烧录方式，也可以用最近正流行的 ISP 方式（In System Programming）将程序代码下载到 AT89S8252 上，这也就是说：有 ISP 功能的芯片不要专用的烧录器就可以把程序代码放入芯片中，此时只要一条专用的下载电缆（Download Cable）就可将程序传入 AT89S8252 上。最特别的是 AT89S8252 内部还有 2KB 的 E^2PROM，我们可以把系统重要的参数与记录值通通存放在 CPU 内部的 E^2PROM 上，要的时候才通过串行通信接口传出，这会使整个系统的线路变成非常简单，系统简单的主要好处是信赖度会相对地提高许多。

假设 8051 单片机的系统中需要长时间储存特定的设定值时，最常见的做法就是把 SRAM 加电池做数据备份，如果您的系统一直在做数据的即时存取及交换时，这个方法是最省钱的，不过充电电池的部分要能随时充电而且绝对不能被短路，否则宝贵的数据就会在瞬间消失。

所以就有人改用更高级的 SRAM 加锂电池 IC，从此不怕无意的电源短路了，这种 IC 原本的使用量就不是很多，所以零售价格是出奇的贵！PC 上用的 RTC，内含 CMOS SRAM 与锂电池，但是 SRAM 的容量只有 64 或 128 字节，售价较为便宜，除非是高价位的工作站计算机，否则这种内置锂电池的 IC 其价位绝对会让人吃不消。如果储存的数据是偶尔才做一次时，就可以改采用外部的 EPROM，如只有八根引脚 93C46 之类的串行式 E^2PROM，不仅接线单纯而且几乎不须任何控制线路，所有精华都在软件程序上。有关这些线路的范例可以参考旗威科技公司的 TH2030 温湿度控制器。如果原线路中的 CPU 改用 Atmel 的 AT89S8252 时，就可以把其中的电池与 93C46 省下，虽然没办法省多少钱，但是线路却增加了保密性。

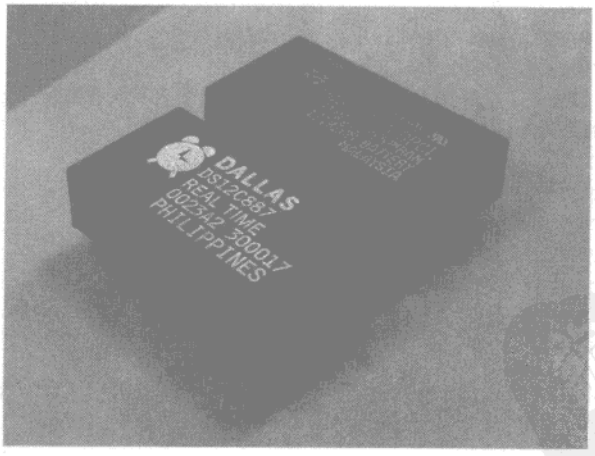


图 8-2　内置锂电池的 SRAM，其厚度较厚

AT89S8252 2KB 的 E^2PROM 的位置是设计在 0000H～07FFH，也是运用 MOVX 的指令进行存取，由于 8051 支持 64KB 的数据存储，AT89S8252 是用一个 WMCON 寄存器中的一个位来区分 EEMEN（E^2PROM ENABLE）选用内部的 E^2PROM 或外部的扩展存储器。只要先把 EEMEN=1，接着用 DPTR 指定地址（000H～7FFH），再下一个 MOVX @DPTR，A 后就可以把数据写入 E^2PROM 中。不过，E^2PROM 写入周期是要时间的，这段时间约是 2.5ms

左右，你必须等待此笔数据写入后，才又进行下个写入周期。这点是 E²PROM 与 SRAM 加电池的最大不同点，写入的操作较慢，但是读取时的速度又和 E²PROM 相当，当然不需要备份电池正是 E²PROM 的最佳卖点了。拿到 AT89S8252 的样品后，我们首先就是把它插入 FLAG51 单片机控制板的 CPU 座上，能够顺利启动并将开机后的信息传回给 PC 端，接下来是在 FLAG51 上试写一个汇编语言程序，看看是否能够顺利将数据写进 E²PROM 中（见程序 1）。

程序 1　T_8252.ASM

```
;FILENAME T_8252. ASM
WMCON       . REG  96H;
            ORG    8000H
            MOV    P1,#00H          ;LED PORT ALL OFF
            MOV    R2,#FFH          ;AS AN COUNTER;
NEXT        INC    R2
            CALL   DELAY            ;DELAY A WHILE;
            MOV    A,WMCON          ;GET WMCON
            ORL    A,#18H           ;SET EEMWE=1,EEMEM=1
            MOV    WMCON,A
            MOV    DPTR,#0100H
            MOV    A,R2
            MOVX   @DPTR,A          ;SAVE R2 VALUE INTO E²PROM
            MOV    A,WMCON
            ANL    A,#EFH           ;E²PROM WRITE ENABLE=0
            MOV    WMCON,A
            MOV    P1,#00H          ;LED OFF
            CALL   DELAY
            MOV    B,R2
            MOV    DPTR,#0100H
            MOVX   A,@DPTR
            CJNE   A,B,ERROR        ;如果不符,代表写入有错误
OK          MOV    P1,#81H
            JMP    EXIT
ERROR       MOV    P1,#7EH
EXIT        MOV    A,WMCON
            ANL    A,#E7H
            MOV    WMCON,A
            JNB    RI,NEXT          ;如果按任一键则返回监督程序
            RET
```

上述的汇编语言程序载入并执行时，可以发现 P1 端口上的 LED 一直显示写入正确的信息，显示的速度约是每秒两次左右。在 AT89S8252 的主要特性上写着：可以有 10 万次的写入与清除周期，这也就是说：E²PROM 的写入也是有次数的限制，无法像 SRAM 一样可以随时写入并修改，可是如果是控制系统的重要参数，读取的机会远比写入的机会还多时，选用 AT89S8252 确实是明智的决定，除此之外，该 CPU 上还有 POWER FAILURE 与 WATCH DOG TIMER 的标志位，当系统快没电或死机时，都可以把状态值存在 E²PROM 上，对工程师的除错确实是有所帮助的。

目前最新版的 AT89CXX 烧录程序已经可以支持烧录 AT89S8252，我们建议读者到 www.atmel.com 的网站上，下载 AT89S8252 的所有数据表，顺便研究一下，AT89S8252 的（In

System Progrmming）在线直接下载烧录的程序是如何完成的，如果你能完成这个下载程序的设计，那对 8051 的功力又要倍增了。

这几年来，许多 IC 大厂已经把所有 IC 数据电子化了，全部转成 Adobe 的 PDF 文件，并放在网站内供全世界的设计工程师浏览并下载，如果您还未顺利取得 PDF 文件的阅读程序时，请到 www.adobe.com 网站下载 Acrobat Reader 程序，以便顺利读取愈来愈多的 PDF 文件。

本章使用的软件

（1）2500AD 8051 C Compiler & Assembler。
（2）IAR 8051 C Compiler。

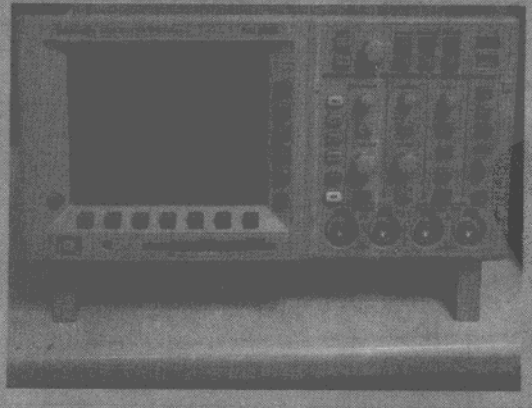
本章使用的硬件

（1）FLAG51 单片机控制板。
（2）AT89CXX 烧录板。
（3）AT89S8252。
（4）DS12C887 Real Time Clock 内置锂电池。
（5）M48T02 内置锂电池的 SRAM。

相关资料网站

可经由下列公司、网站取得更进一步的信息：
http://www.chipware.com.tw：取得单片机控制板相关资料。
http://www.atmel.com：取得 AT89S8252 的烧录资料。
http://www.dalsemi.com：取得 DSCl2C887 的资料。
http://www.iar.com：取得 8051 C Compiler 相关资料。
http://www.keil.coim：取得 8051 C Compiler 相关资料。
http://www.adobe.com：取得免费的 Acrobat Reader PDF 阅读程序。

9



高级示波器可以帮助工程师尽早找出产品设计上的盲点。我们的经验是仪器越用越好，开发时间就可提早。

第 9 章　自制的 89C51 烧录器

烧录器的程序若处理不慎的话，可能会损坏 IC，这一章我们一起来设计、规划一台可自行 DIY 的 AT89C51 烧录器。

最近几年来，我们用 8051 单片机至少完成 15 种以上控制器的开发，其中的系统有非常复杂的，也有相当单纯的控制器。通常在实时在线仿真器（ICE）上执行无误后，就直接烧录成 EPROM，插入控制系统后会出错的机会已不多了，但是在烧录 AT89C51 时，却一直有挫折感。原来是我一直委予重任的万用烧录器竟然频频出状况，不论是何种计算机或机器设备，标榜“万用”时反而问题最多，因为“Universal”（万能的）往往是最难以控制的，这个说法刚好印证在烧录 AT89C51 上。

在烧录 AT89C51 之前，曾经听过万用烧录器会损坏 AT89C51 的说法，但直到烧坏了 20 个 89C51 之后，我才真正“认同”了这项说法。话又说回来，哪里有 AT89C51 最合适的烧录器呢？是自制还是外购呢？最后我们选择了自己来，因为生产 AT89C51 的 Atmel 公司把整个烧录过程描述得相当清楚，所以我们初步地设计出烧录的线路（请参考第 7 章），只需单一的+5V 电源即可烧录 AT89C51，可是事与愿违。

在 Atmel 公司提供的资料中，提到了一个相当重要的关键点：所有出厂的 AT89C51，其 Vpp 电压都预先设计为 12V，Atmel 的用意是让 AT89C51 的烧录方式能符合大部分现有的 8751 烧录器。请再看看 AT89C51 的烧录引脚图（见图 9-1），在 AT89C51 的 6 种烧录模式中，当选择 Vpp 模式（Select Vpp Code）时，其 $\overline{EA}$/Vpp 脚一定要维持在+12V（而非+5V）。

MODE	RST	$\overline{PSEN}$	ALE/$\overline{PROG}$	$\overline{EA}$/Vpp	P2.6	P2.7	P3.6	P3.7
WRITE CODE DATA	H	L	一个负脉冲	H/12V (1)	L	H	H	H
READ CODE DATA	H	L	H	H	L	L	H	H
SELECT Vpp CODE (2)	H	L	一个负脉冲 (3)	12V	L	H	H	H
WRITE LOCK BIT-1	H	L	一个负脉冲	H/12V	H	H	H	H
BIT-2	H	L	一个负脉冲	H/12V	H	H	L	L
BIT-3	H	L	一个负脉冲	H/12V	H	L	H	L
CHIP ERASE	H	L	一个负脉冲 (3)	H/12V	H	L	L	L
READ SIGNATURE BYTE	H	L	H	H	L	L	L	L

图 9-1　AT89C51 的烧录引脚图

换句话说，当 Vpp 烧录电压更改时（+12V 换成+5V 或是+5V 换成+12V），$\overline{EA}$/Vpp 引脚必须提升到+12V 才算是有效，糟糕的是 AT89C51 刚出厂时，Vpp 烧录电压是定成 Vpp=+12V，若想把 Vpp 改成+5V 时，一定要再下一个 Select Vpp Code 命令给 AT89C51，但是下命令时其中的$\overline{EA}$/Vpp 引脚却必须是+12V，ATMEL 公司的美意竟然变成烧录时的绊脚石，所以本书第 7 章的烧录线路如果不做修改是无法使用的！这也可能是使用万用烧录器时，经常会损坏 AT89C51 内部闪存的主因。

既然改换 Vpp 电压时，要额外的+12V 电压，我们只好再修改烧录线路，以便$\overline{EA}$/Vpp 引脚能在+12V 与+5V 间进行切换。图 9-2 是经过我们修改后较恰当的 AT89C51 烧录线路。

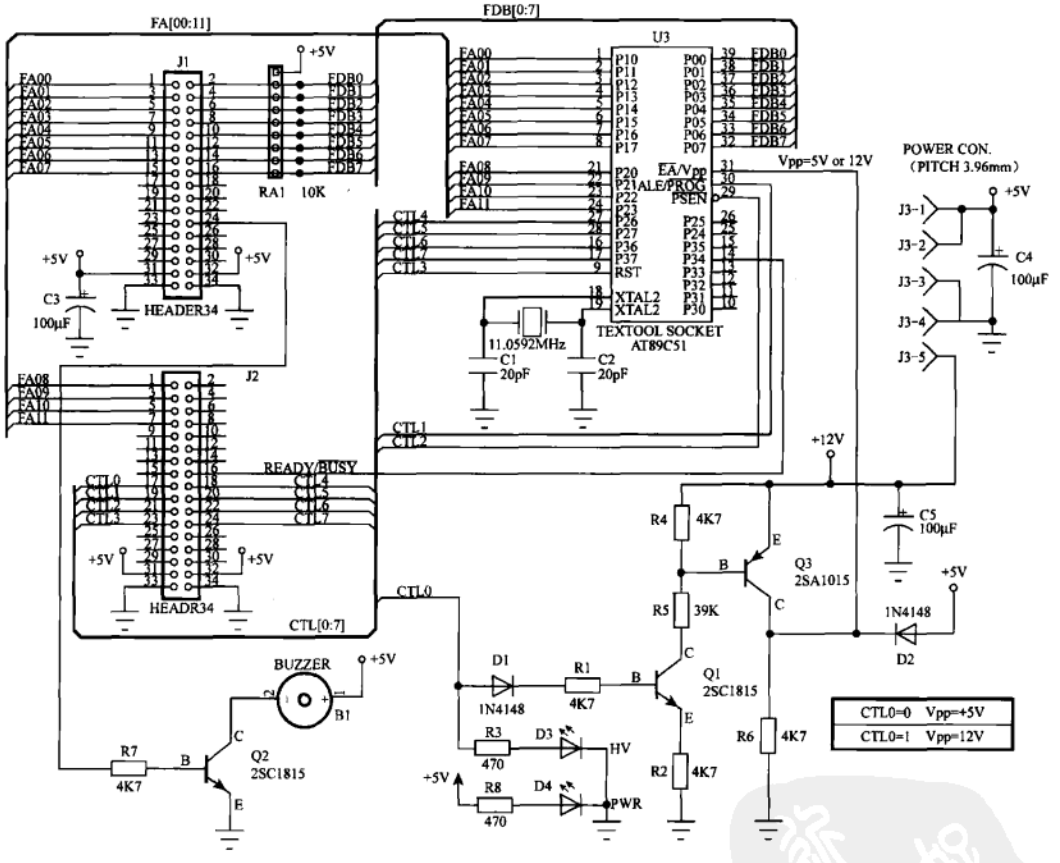
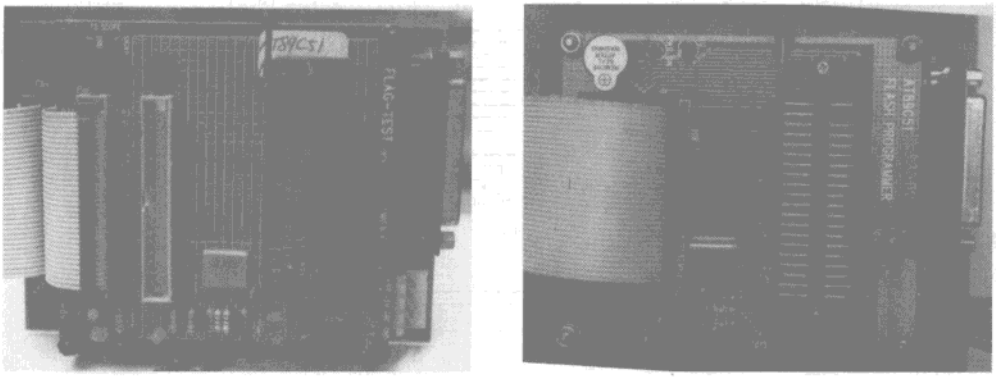


图 9-2 经过修改的 AT89C51 烧录线路，Vpp 有两段（+5V/+12V）可选择

9-1 烧录线路分析

我们设计的 AT89C51 烧录线路共需两颗 8255 来产生必要的烧录波形，我们建议直接使用 FLAG51 控制板来控制这个烧录线路，只要连接两条 34P 的数据线即可，这两条数据线也提供了+5V 的电源，AT89C51 的 12 条 ADDR 线分别由两个 8255 的 PA 端口提供。图

9-2 中 J1 接头上的 FDB7～FDB0 共 8 条线供应待烧录的数据（write）与已烧录数据的验证（read），所以这个端口必须是双向的（bidirectional）。J2 上另外提供了 8 条烧录控制线，这些信号都必须完全正确，才能产生图 9-1 中 5 种功能互异的烧录模式。为了 AT89C51 的第 31 引脚能有+5V 与+12V 的切换能力，我们用两颗晶体管做两种电压的切换，实际的切换控制点在 CTL0 上（此点接到 8255 的 PC0 位上），当 CTL0=0 低电位时，标示 HV（高电压）的 LED 不亮，Q1 晶体管得不到足够的偏压而未导通，连带导致 Q3 晶体管截止，此时仅有 D2 二极管顺向导通，提供约 4.5V 左右的电压给 89C51 的 EA/Vpp，4.5V 对 AT89C51 是一个有效的 H 电位。当 CTL0=l 高电位时，HV 的 LED 点亮，Q1 与 Q3 晶体管都连通，这会使得 Q3 的 C 集电极上出现接近+12V 的电压，而 D2 的 4148 则处于截止状态。另外为了有更佳的操作方便性，我们也加入了电源指示 LED 与烧录正确与否的蜂鸣器声响指示，图中的 U3 应该使用烧录专用的 40 引脚测试夹。图 9-3 则是 AT89C51 烧录板的完整照片。



（a）我们所做的 AT89C51 烧录 PROTOTYPE 试制品　　（b）AT89C51 烧录器完成品

图 9-3　AT89C51 烧录板的完整照片

9-2　DIY 自己装步骤

由于 AT89C51 烧录板使用的元件不多，非常适合读者自己安装，在此我们也对这方面作一些提示。图 9-4 是本烧录线路的 PC 板布置与元件位置图。

步骤 1：焊接电阻（R1～R7）、陶质电容（C1～C2）、电解电容（C3）和 11.0592MHz 的石英晶体。

步骤 2：焊接 40 引脚的圆孔 IC 座，本 IC 座稍后将插上另一个 40 引脚 IC 座，最上层才是 3M 的 IC 测试夹（整块烧录板焊接完成后才插上），这样安排的理由是更换测试夹时较为方便。

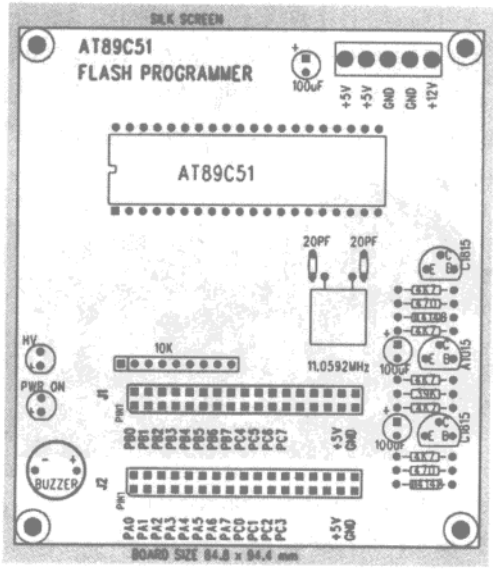
步骤 3：焊接 9P 的 10kΩ电阻，电阻的共同点通常有一圆标示点，请依板上的标示方向焊接。

步骤 4：焊接 3 个晶体管（2SC1815 与 2SA1015），Q1 与 Q2 是 2SC1815（NPN），Q3 则是 2SA1015（PNP），请依板上的标示方向焊接，搞错就无法控制 Vpp 电压与蜂鸣器了。

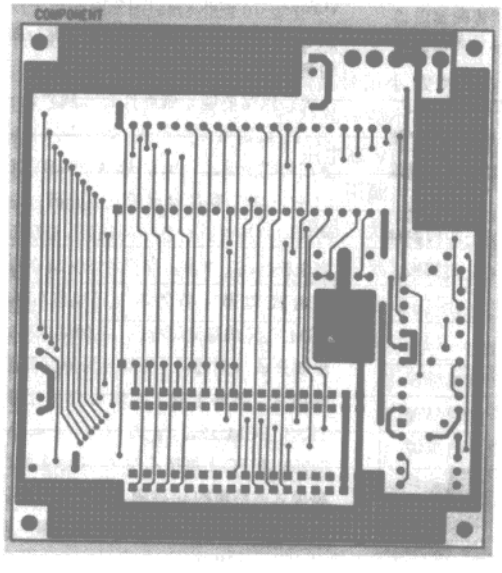
步骤 5：焊接 J1 与 J2 34P 排线接头，如果方向错了就没法烧录 AT89C51 了。

步骤 6：焊接 4P 电源接头（引脚间距为 3.96mm），PC 板上有标示+5V 与 GND 的位置。标示+12V 电压端另用一个接线柱焊起来，正常情况下烧录是不需要+12V 电压的，不过实际上还是需要+12V 的电源供应，详情请看我们的分析。

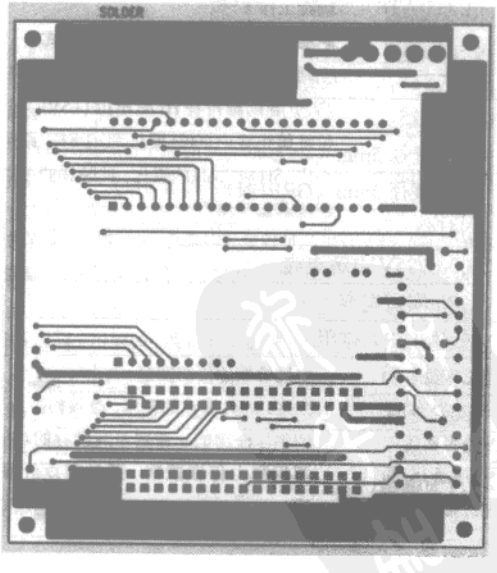
步骤 7：再次检查焊接点无误后，装上四根铜柱及螺丝，准备开始测试自己装的 89C51 烧录板。



（a）AT89C51 烧录板的元件位置图



（b）AT89C51 烧录板的元件面线路图



（c）AT89C51 烧录板的焊接面线路图

图 9-4　本烧录线路的 PC 板布置与元件位置图

9-3 烧录器的基本功能测试

由于烧录 AT89C51 需要较充裕的内存空间，所以做烧录器测试前请先扩展 FLAG51 上的 SRAM 内存空间（扩展方法请看稍后的叙述），开始进行测试前，请把烧录板与 FLAG51 控制板固定在一起，然后用 34P 排线将两块线路板连接起来（FLAG51 的 CN1 接烧录板的 J1，FLAG51 的 CN2 接烧录板的 J2）。请参看图 9-5 的示范照片，测试时的电源必须另有+12V 的输出点，以便 FLAG51 与烧录板连接的示范线路测量 Vpp 等于+12V 时电压准确。打开电源后，依照平常的操作程序，让 PC 与 FLAG51 完成联机，如果联机成功后，请准备示波器和三用电表（最佳）或是逻辑笔和三用电表（次之），以便对 89C51 烧录板进行基本功能的测试。

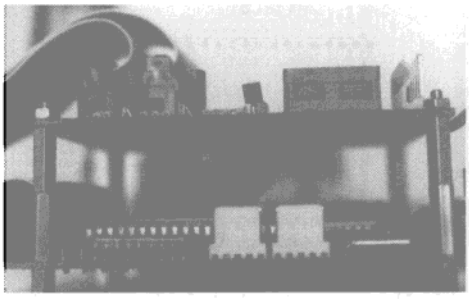


图 9-5 FLAG51 与烧录板连接的示范照片

表 9-1 AT89C51 PROGRAMMER 元件表

名　称	数　量	说　明
FLAG-AT89C51 烧录线路板	1	尺寸 85mm×95mm，双面通孔电路板
AT89C51	1	实验用 Vpp 烧录电压已经更改为 5V
2SC1815	2	NPN 小信号晶体管
2SA1015	1	PNP 小信号晶体管
LED 直径 5mm（RED）	1	指示 HV 高电压
LED 直径 5mm（GREEN）	1	指示电源用
IN4148	2	信号切换二极管
100μF/25V 电解电容	3	电源滤波用
20PF 陶质电容	2	振荡整形用
10kΩ（9P）排阻	1	上拉电阻
7.7kΩ电阻	5	1/4W
39kΩ电阻	1	1/4W
470 电阻	2	1/4W
11.0592MHz 石英晶体	1	系统振荡用
BUZZER 蜂鸣器	1	脚距 8mm，内含振荡器
34P 排线公接头（180 度）	2	双排接头引脚间距 2.45mm
4P 电源接头	1	引脚间距 3.96mm
1P 接线柱直径 1mm	1	额外电压+12V 输入用

烧录线路要能正确操作，有三个要素：

第一点：Vpp 烧录电压要正确。

第二点：PROG 烧录脉冲时间要准确，89C51 烧录一个字节至少要 1.5ms。

第三点：烧录模式要正确。

以下我们将对这些特性进行逐一的验证，若验证的结果完全符合时，才真正进行 AT89C51 的烧录。请先由 PC 端加载我们事先已设计好的 TEST_PGM.TSK 烧录测试程序，这个测试程序将测试步骤分成 8 个部分，每个部分刚好对应一种烧录模式。请参考图 9-1 的烧录模式图。当 TEST_PGM.TSK 程序成功地加载 SRAM 62256，在 PC 上每按一次 G 键就会执行一种烧录模式并持续约 40s，我们刚好趁这段时间用测试工具进行上述三要素的测量。总共按了 8 次 G 键后，就能完成 AT89C51 所有模式的验证工作。

验证一：WRITE CODE DATA 模式

测试程序会持续产生 PROG 的脉冲信号，EA/Vpp 引脚则维持在 HI 高电位（+5V），同时 12 条地址值也会由 000H 变化到 FFFH，数据线则分别作 55H 与 AAH 的变化，此时我们可以用示波器或逻辑笔检查。

（1）地址线是否有变化，A0 的变化速度最快，A11 则变化最慢，这种信号用示波器观看最方便。

（2）数据线也要有 0 与 1 的变化，若完全没变化代表烧录板连接有问题。

（3）观察 $\overline{\text{PROG}}$ 脉冲的宽度，应该约是 1ms 左右，如果您使用的示波器没有储存的功能时，请运用眼睛仔细观察，还是可以计算出其宽度。

（4）EA/Vpp 引脚的电压应该维持在 4.5V 以上。

（5）请用示波器或是逻辑笔检查控制线 P2.6、P2.7、P3.6 和 P3.7 的状态是否正确。

验证二：READ CODE DATA 模式

这个模式主要是在读入已烧录的数据，供作数据验证之用，测试程序会把地址值由 000H 变化到 FFFH，并完成所有控制线的控制。

（1）观察地址值是否有变化。

（2）由于测试夹上并未插上 89C51，所以 8 条数据线都应维持在高电位上。

（3）检查其他控制引脚的逻辑状态是否正确。

（4）EA/Vpp 引脚的电压应该维持在 4.5V 以上。

验证三：SELECT Vpp CODE

这个模式在选择 Vpp 烧录的电压，此时 $\overline{\text{EA}}$/Vpp 引脚上的电压一定要是+12V 才行，测试程序会持续把 AAH 的数据值送到地址 055H 上。做本验证时请将额外的+12V 电压加到烧录板上。请注意只有本模式时，才需要额外的+12V 电压。

（1）$\overline{\text{EA}}$/Vpp 一定要是+12V，如果不是时，请检查 Q1 与 Q3 晶体管是否有故障，或是外围元件有错误。

（2）检查 8255 送出的地址值应该是 055H。

（3）检查 8255 送出的数据值应该是 AAH。

（4）观察 $\overline{\text{PROG}}$ 脉冲的宽度，应该约是 1ms 左右。

（5）检查其他控制引脚的逻辑状态是否正确。

验证四：WRITE LOCK BIT-1

这个模式将对 AT89C51 加上第一道保护锁，测试程序完成控制脚的状态设定后，就持续地送出 $\overline{\text{PROG}}$ 脉冲。AT89C51 会禁止程序使用 MOVC 指令将内部 4KB 的 PROGRAM CODE 读出。

（1）$\overline{\text{EA}}$/Vpp 引脚的电压保持在 4.5V 以上。

（2）地址与数据线都完全没有变化。

（3）检查其他控制引脚的逻辑状态是否正确。

（4）观察 $\overline{\text{PROG}}$ 波的宽度，应该约是 1ms 左右。

验证五：WRITE LOCK BIT-1

这个模式将对 AT89C51 加上第二道保护锁，测试程序完成控制脚的状态设定后，就持续地送出 $\overline{\text{PROG}}$ 脉冲。AT89C51 会进而禁止我们使用 VERIFY 模式将内部 4KB 的 PROGRAM CODE 读回。

（1）$\overline{\text{EA}}$/Vpp 引脚的电压保持在 4.5V 以上。

（2）地址与数据线都完全没有变化。

（3）检查其他控制引脚的逻辑状态是否正确。

（4）观察 $\overline{\text{PROG}}$ 脉冲的宽度，应该约是 1ms 左右。

验证六：WRITE LOCK BIT-1

这个模式将对 AT89C51 加上第三道保护锁，测试程序完成控制引脚的状态设定后，就持续地送出 $\overline{\text{PROG}}$ 脉波。AT89C51 会限定程序长度在 4KB 以内，并且禁止所有的读出与验证的操作。

（1）$\overline{\text{EA}}$/Vpp 引脚的电压保持在 4.5V 以上。

（2）地址与数据线都完全没有变化。

（3）检查其他控制引脚的逻辑状态是否正确。

（4）观察 $\overline{\text{PROG}}$ 脉冲的宽度，应该约是 1ms 左右。

验证七：CHIP ERASE

这个模式一次把 4KB 的 FLASH MEMORY 清除掉，清除时间只要 10ms，测试程序会持续做清除的操作。

（1）$\overline{\text{EA}}$/Vpp 引脚的电压保持在 4.5V 以上。

（2）地址与数据线都完全没有变化。

（3）检查其他控制引脚的逻辑状态是否正确。

（4）观察 $\overline{\text{PROG}}$ 脉冲的宽度，应该约是 1ms 左右。

验证八：READ SIGNATURE

这个模式在于读取 AT89C51 的验证码（SIGNATURE），在烧录时程序应先读入该验证码是否正确，正确时才进行真正的烧录操作，否则会损坏待烧录的元器件，测试程序会分别送出地址值 030H 和 031H。如果是 ATMEL 公司制造的 AT89C51 时，送回的数据码必须是 1EH 和 51H。测试程序时，则依序送出两个地址值，并在 PC 屏幕上显示读回的两组数据。

（1）$\overline{\text{EA}}$/Vpp 引脚的电压保持在 4.5V 以上。

（2）检查其他控制引脚的逻辑状态是否正确。

（3）观察 SIGNATURE 值是否正常。

9-4 AT89C51 烧录器的使用

一般的烧录程序都尽量把操作者的操作简化，只要加载待烧录的数据，然后再按个键后就行了。其实整个烧录过程都相当具有学习研究性的，在这里我们想换个方式来处理整个烧录过程，换句话说，我们将用最基本的操作方式来说明如何烧录一个 AT89C51。经由这些过程的说明，您将可以对整个烧录的硬软件有相当程度的认识，由于我们对整个烧录程序是完全公开的，假使您对其中的操作过程不满意时，可以直接修改程序，以满足个别的需要。

旗威科技公司声明：由于 ATMEL 公司发现 Vpp 电压设定的问题一直对客户造成困扰，所以，已经将此功能取消，尔后你拿到的 AT89C51 或 AT89C52 一定要加+12V 电压方能进行烧录或数据清除。我们提供的 FLASH.ASM 程序也只有在 SELECT Vpp CODE 模式时，才须在 $\overline{EA}$ /Vpp 引脚上加上+12V。接着我们将详细地说明，并示范整个烧录的过程。烧录前请把 FLAG51 与烧录板连接妥当，并且要成功进入联机程序，亦即 PC 与 FLAG51 已完成相互的沟通，然后才正式进入烧录 AT89C51 的程序。此时请再确认两点：

（1）SRAM 已改成 62256。

（2）两个 8255 的地址已改为 6000H 和 7000H。

烧录步骤 1：清除待烧录区与验证区内的 SRAM 空间（A000H～BFFFH），请在 PC 的键盘上键入 F A000 BFFF FF<ENTER>，将该区域全部清除成 FFH。

烧录步骤 2：加载待烧录的二进制文件数据，该文件长度最大不得超过 4096 字节，假设文件名称为 XXX.TSK，请在 PC 的键盘上键入 L XXX.TSK<ENTER>，此时文件将由 SRAM 区的 8000H 开始放起，最长则放到 8FFFH。

烧录步骤 3：将烧录数据搬到待烧录区，请在 PC 的键盘上键入 M 8000 8FFF B000<ENTER>，此时 B000H 到 BFFFH 上已是即将进行烧录的数据。

烧录步骤 4：加载 AT89C51 的烧录程序 FLASH.TSK，请在 PC 的键盘上键入 L FLASH.TSK<ENTER>，真正的烧录程序已进驻 8000H 地址。

烧录步骤 5：将 AT89C51 放到 40P 的测试夹上，请注意方向绝对不能搞错，然后压下固定夹使得 AT89C51 不能移动。

烧录步骤 6：开始进行烧录，请在 PC 的键盘上键入 G<ENTER>，以下的操作将由 FLASH.ASM 程序完全控制。

烧录步骤 7：烧录程序首先检查待烧录 IC 的验证码（SIGNATURE）是否正确，即该 IC 的两个验证码必须分别 1EH 和 51H，不符合时立即结束本烧录程序。

烧录步骤 8：烧录程序接着送出 Vpp=5V 的命令，如果此时烧录板上加+12V 电源时，这个命令是有效的；反之，本命令无效。本命令执行时，标示 HV 的 LED 灯会闪一下，代表此时可能有高于+5V 的电压会加在 AT89C51 的 $\overline{EA}$ /Vpp 脚上。除此特殊情况外，$\overline{EA}$ /Vpp 脚都是处于数字电路的高电位状态。

烧录步骤 9：烧录程序送出 CHIP ERASE 命令，迫使 AT89C51 清除 4KB 的 FLASH MEMORY，百分之一秒（10ms）后，AT89C51 内部的程序存储区都将变成 FFH。

烧录步骤 10：清除完毕之后，烧录程序继续送出 READ CODE 的命令，连续读回 89C51

内部共 4096 个地址的内容。读回的数据放在 SRAM 的 A000H～AFFFH 区。

```
>G
o from addr 8000H...
ATMEL AT89C51 FLASH PROGRAMMER WRITTEN BY LIN S.M.
MAKE SURE 1ST 8255 IN CN1 ADDR MAP ARE 6000-6FFFH
MAKE SURE 2ND 8255 IN CN2 ADDR MAP ARE 7000-7FFFH
[READ SIGNATURE=FFH, FFH]
[SET Vpp CODE=5V]
[ERASE AT89C51 4K BYTES FLASH MEMORY]
[READ SIGNATURE=1EH, 51H]
[BLANK CHECK...]
[READ SIGNATURE=1EH, 51H]
[BLACK CHECK...]
[PROGRAMMING 4K BYTES(1P=256 BYTES)...]
PPPPPPPPPPPPPPPP
[VERIFY CODE DATA]
[WRITE LOCK_BIT1 AND LOCK_BIT2]
[BLACK CHECK...]
[READ SIGNATURE=FFH,FFH]
[AT89C51 CHIP PROGRAM SUCCESSFULLY]
```

图 9-6　烧录时在 PC 上所显示的画面

烧录步骤 11：检查 A000H～AFFFH 内的数据是否全部都是 FFH；若不是，代表未能完全清除，立即结束本烧录程序。

烧录步骤 12：送出烧录命令，开始烧录 4096 个字节的数据，B000H～BFFFH 的数据逐一被读取，并送到 AT89C51 的数据输入端口上。

烧录步骤 13：烧录完后，随即进入数据验证的阶段，再度送出 READ CODE 命令，读入已烧录的共 4096 个数据。这些数据都转存入 SRAM A000H～AFFFH 内。

烧录步骤 14：验证程序逐一比对两段存储区内的数据(B000H～BFFFH 与 A000H～AFFFH)，若完全相同时，表示烧录成功，只要有一个地址不符合，即代表烧录失败。

程序 1　AT89C51 烧录程序的主程序部分

```
        ORG     8000H
START
;       MOV     SP,#60H
        LCALL   DELAY
        LCALL   CLEAR_BUFFER
        MOV     DPTR,#MSG_MAIN
        LCALL   MSG_OUT
        MOV     DPTR,#SG-PPI1
        LCALL   MSG_OUT
        MOV     DPTR,#MSG-PPI2
        LCALL   MSG_OUT
;
```

```
        MOV     DPTR,#MSG_SIG
        LCAL    MSG-OUT
        LCALL   READ_SIGNATURE
;       CJNE    A,#OK,SIG_ERR
;
        MOV     DPTR,#MSG_Vpp
        LCALL   MSG_OUT
        LCALL   WRITE_Vpp_CODE;SELECT Vpp=5V
                                        ;VppMUST BE 12V TO CHANGE Vpp VALUE
;
        MOV     DPTR;#MSG_ERA
        LCALL   MSG_OUT
        LCALL   CHIP_ERASE              ;ERASE FLASH MEMORY
;
        MOV     DPTR;#MSG_SIG
        LCALL   MSG_OUT
        LCALL   READ_SIGNATURE;READ SIGNATURE AGAIN
;
        LCALL   READ                    ;READ IN 4K BYTES DATA FROM FLASH
        MOV     DPTR,#MSG_BLK
        LCALL   MSG_OUT
        LCALL   BLANK_CHECK
;CONTINUE
        MOV     DPTR,#MSG_PGM
        LCALL   MSG_OUT
        LCALL   BEEP_START
        LCALL   PROGRAMMING
;
        MOV     DPTR,#MSG_LF
        LCALL   MSG_OUT
        LCALL   DELAY
        LCALL   READ
        LCALL   READ
        LCALL   DELAY
        MOV     DPTR,#MSG_VER
        LCALL   MSG_OUT
        LCALL   VERIFY
        CJNE    A,#OK,VERIFY_ERR
;
        MOV     DPTR,#MSG_LOCK
        LCALL   MSG_OUT
        LCALL   LOCK_BIT
;
        LCALL   READ                    ;READ IN 4K BYTES DATA FROM FLASH
        MOV     DPTR,#MSG_BLK
```

```
        LCALL   MSG_OUT
        LCALL   BLANK_CHECK
        CJNE    A,#OK,LOCK_ERR
;
        MOV     DPTR,#MSG_SIG
        LCALL   MSG_OUT
        LCALL   READ_SIGNATURE;READ SIGNATURE AGAIN
;
        MOV     DPTR,#MSG_PASS
        LCALL   MSG_OUT
RETURN  RET
;
```

烧录步骤 15：显示烧录程序的总结果，通过或是失败。

9-5 烧录程序分析

整个烧录程序的控制流程大致与上述的烧录步骤吻合，在此由于篇幅实在有限，我们仅列出 FLASH.ASM 的重要声明与主程序部分，完整的烧录程序与说明将在后面列出来。

9-6 FLAG51 存储器 RAM 的扩展

FLAG51 控制板上有一个 SRAM 6264 共提供了 8KB 的空间，扣除系统占用的 1KB，使用者可用的空间约有 7KB 左右，写一般的应用控制程序是绝对足够的，但是若当成烧录工具时，总觉得 RAM 的空间不太足够！以烧录内置 4KB FLASH MEMORY 的 AT89C51 为例，待烧录的数据就有 4KB，若有做数据对比（VERIFY），则也要另外 4KB 的 RAM 存放读入的数据，这时的 8KB RAM 空间就明显的不足了。如果 SRAM 的空间能扩充到 32KB，应该就能满足这类的应用，而且原先开发的监控程序也不需做任何修改。市面上能买到 32KB SRAM 的编号是 62256，大部分是低功率消耗（Low Power）型的。图 9-7 是 6264 与 62256 的引脚图，其中的差异是 62256 多了两条地址线（A14 与 A13），理论上只要加上这两条线即可将 RAM 的容量由 8KB 扩充到 32KB。62256SRAM 的扩充需要以下元件：62256 一颗，内置新译码地址的 PEEL18CV8（或 GAL16V8），28P 的 IC 座和 5cm 左右的镀银线。请依照以下的更换步骤进行修改，以免有错误情况出现。

步骤 1：关闭电源，取下 FLAG51 控制板上的 6264 8K SRAM。

步骤 2：拿到 28P 的 IC 座，请把该座的第 26 引脚强行取下，再把该 IC 座插到原先的插 6264 的 IC 座上，以免新 IC 座的第 26 引脚不和原 IC 座的第 26 引脚相通。

步骤 3：插上新的 62256 SRAM，请注意此时 62256 的第 26 引脚已不和 FLAG51 控制板上相通。

步骤 4：从 U6 EPROM 的第 26 引脚接一条镀银线到 62256 的第 26 引脚上。

步骤 5：从 U6 EPROM 的第 27 引脚接一条镀银线到 62256 的第 1 引脚上。

步骤 6：更换 U5 PEEL18CV8，此 IC 重新定义 SRAM 和 8255 的寻址。

步骤 7：确认添加的 PEEL18CV8、IC 座与 62256 是否插牢。

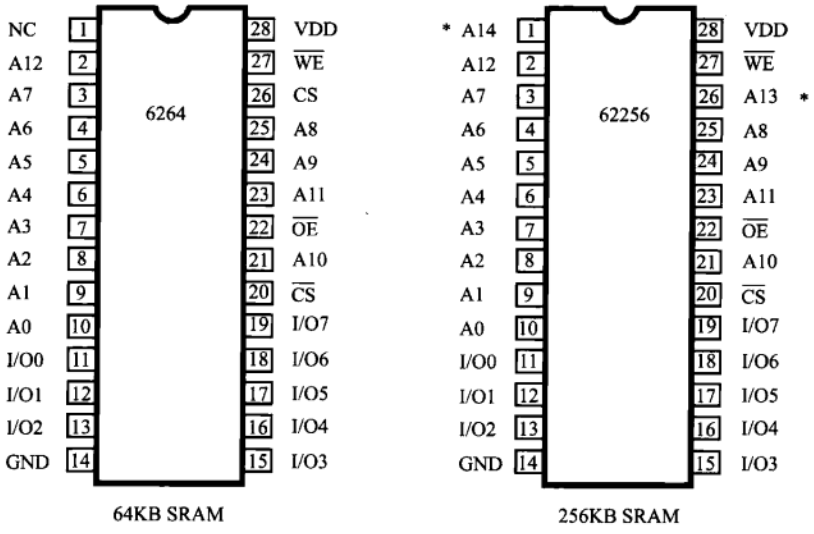


图 9-7　6264（8KB）与 62256（32KB）的引脚比较图

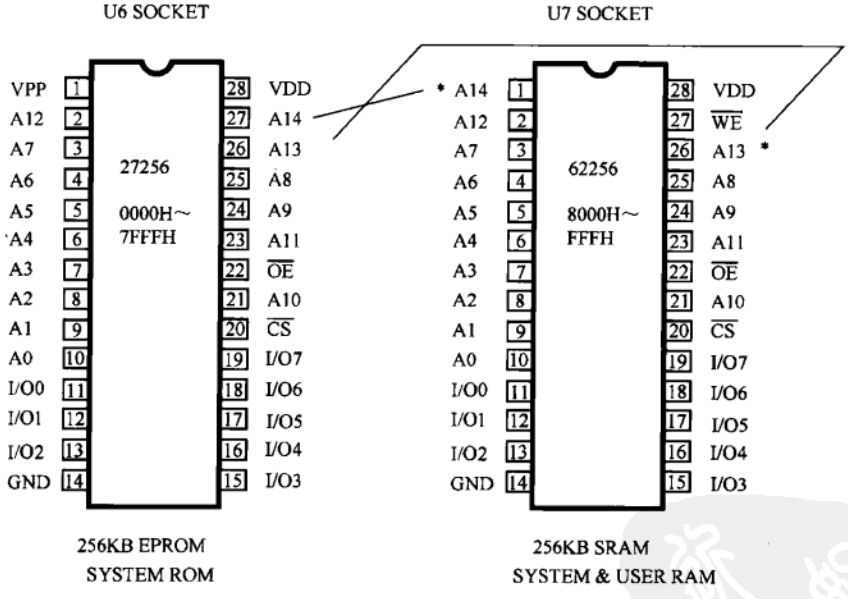


图 9-8　扩展的 SRAM 存储器空间需要加跳两条线，请留意 62256 的第 26 引脚不可以与 U7 插座的第 26 引脚相通，否则不能开机

步骤 8：执行 FLAG51.EXE 联机程序后，再打开 FLAG51 控制板的电源，应该可以在 PC 的屏幕上看到图 9-9 的画面，如果有的话，恭喜您了！然后 SRAM 62256 将占用地址 8000H～FFFFH 的外部存储器空间，两个 8255 则分别改占用 6000H～6003H 与 7000H～7003H 两段地址。

```
TITLE'APEEL  FILE FOR DECODER
DESIGNER:  LIN  S.M.
DATA MEMORY EXPAND TO 32K BYTES
     MAP:80000-FFFFH AS EXTERNAL DATA MEMORY      (READ/WRITE)
         8000-BFFFH AS EXTERNAL PROGRAM MEMORY     (READ ONLY)
DATE;1994/06/15'
PEEL18CV8
A12      PIN2
A13      PIN3
A14      PIN4
A15      PIN5
RESET    PIN6
ALE      PIN7
PSEN  PIN8
RD    PIN9
RAMCE        PIN 19=NEG COM FEED_PIN          "OVERLAP
18255CE1     PIN 18=NEG COM FEED_PIN
18255CE2     PIN 17=NEG COM FEED_PIN
SIOPORT      PIN 16=NEG COM FEED_PIN
INV_RESET    PIN 15=NEG COM FEED_PIN
INV_ALE      PIN 13=NEG COM FEED_PIN
RAMOE        PIN 12=NEG COM FEED_PIN          "OVERLAP   EQUATIONS
RAMCE        =(A15&!A14&   !PSEN&!RESET) #    "8000-BFFFH PRG
             (A15&         PSEN&!RESET)       "8000-FFFFH DATA
I8255CE1     =(A15&!A14A13&A12&PSEN&!RESET)   "6000-6FFFH EXT DATA
I8255CE2     =(A15&!A14A13&A12&PSEN&!RESET)   "7000-7FFFH EXT DATA
SIOPORT      =(A15&!A14A13&A12&!RESET) #      "F000-FFFFH
INV_RESET    =!RESET
INV_ALE      =ALE
RAMOE        =A15&            PSEN&!RD#       "8000-FFFFH DATA RD
             A15&!A14&        !PSEN&RD        "8000-BFFFH PROD RD
TEST_VECTORS
```

图 9-9　新修改的 U5 PEEL18V8 内部译码数据

```
D:\TC>FLAG51  2  指定由 COM2 和 FLAG51 相通
Last modified at 1992/03/30
FLAG-51 moni tor program written by Lin S.M.
P,O.BOX 1859,KAOHSIUNG TAIWAN
All right reserved
Anytime you could type  'Q'  to exit
Baud rate = 9600 BPS (8 bit,1 stop,no parity)
Communication port is COM2.
Try to connect to 8051 target system...
MCS-51 Monitor Program Vorsion 2.3
>

FLAG51 重新开机
8051 monitor program by Lin S.M  '1993/08/30
System memory check...
System address      (X000H)0123456789ABCDEF
Check MEM block (4K block).........RRRRRRRR
MCS-51 Monitor Program Version 2.3
>
```

图 9-10　FLAG51 控制板的 SRAM 扩充到 32KB 后，开机时在 PC 屏幕上所看到的正确画面

本章使用的软件

（1）2500AD 8051 Assembler。
（2）IAR 8051 C Compiler。

本章使用的硬件

（1）FLAG51 单片机控制板。
（2）AT89CXX 烧录板。
（3）AT89C51：4KB 闪存微控制器。
（4）62256 SRAM：32KB SRAM。
（5）PEEL18CV8：可编程元件。
（6）GAL16V8：可编程元件。

相关资料网站

可经由下列公司、网站取得更进一步的信息：
http://www.chipware.com.tw：查询单片机控制器相关资料。
http://www.atmel.com：查询 AT89C 系列 Microcontroller 相关资料。
http://www.ictpld.com：查询 PEEL 组件相关资料。
http://www.latticesemi.com：查询 GAL 组件相关资料。
由于小于 64KB SRAM 的利润微薄，许多原本生产 SRAM 的国际大厂纷纷停产低容量的 SRAM，所以，若想查询 SRAM 的引脚及时序时，请上网到 http://www.google.com，再用 SRAM 32K 的关键词去搜寻。

10

另一款工业用的 8051 控制板，旗威科技采用 RISC 架构的 8051，外挂 128KB 的 Flash 与 32KB E^2PROM。

第 10 章　AT89C2051 烧录器的程序修改

作为技术人员在设计工作上的投入，一星期七天随时都是处于设备行动（Standby）的阶段，所以下面我们以一周七天的方式，描绘 AT89C2051（内含 2KB 闪存）烧录程序的开发过程，同时也谈到从事开发者的一些“压箱底”的除错技巧。

现在一般电子类的杂志谈烧录器方面的技术文章已经很少见了，其中的原因之一是：一台万用的烧录器号称任何可编程的元件都可烧录，所以使用者自制烧录器的机会已经是少之又少了。另一个原因是大家已不习惯自己看 DATA SHEET 然后动手设计软硬件线路与程序了，这对产业的影响是相当深远的。有一次我们跟著名的电源供应器制造厂的高级主管聊天，不自觉地谈到想发展航太工业，其中有一句话让我思考甚久：“如果我们连最基础的 POWER SUPPLY 都做不好的话，谈任何尖端科技或产业都是不牢靠的”，由以上这句话就可以看出许多技术还是相当脆弱并且不可靠的。

10-1　星期一：烧录资料研究

其实在数月前就曾经看过 AT89C2051 的相关资料，但是因为工程尚未完全交差，只好一直搁在桌上。直到 11 月底顺利地将所有的稿件交出之后，整天又在沉思如何修改原来的 AT89C51 烧录器，以便能借用原有的线路去烧录只有 20 根引脚的 AT89C2051。大部分的万用烧录器对每根烧录脚都辅以“PIN-DRIVER”的架构，请看图 10-1，这种线路可以让每根引脚都有输出 0～25V 电压的能力，只要配合适当的烧录时序，就可以烧录所有的可编程元件（PROGRAMMABLE DEVICES）。不过使用 PIN-DRIVER 会使线路变得相当复杂，相对的硬件成本就很高了。如果我们要自己做烧录器的话，最好的方法还是用 8255（可编程 IO）来做，线路会变成非常简单，而唯一的缺点就是变化性较低，仅能适用某几个特定的元件，用 8255 来做烧录器的接口另一个好处是 I/O 数量够且价格非常便宜。

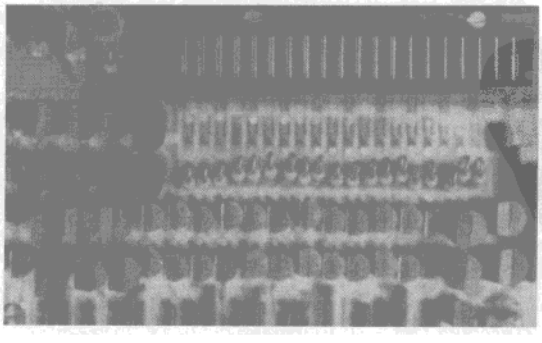


图 10-1　PIN-DRIVER 的照片，可输出 0～25V，输入为标准 TTL 电平，而且可以浮接，因此每个 PIN 都有相同的线路与架构

由 ATMEL 公司提供的数据来看，很明显地已经避开了 AT89C51 烧录时的诸多缺点，并且指定烧录时烧录电压一定是+12V，所以烧录时不会因指定的烧录电压有误造成元件的永久损坏。由于 AT89C2051 的引脚数减少一半，所以烧录时的地址指定方式做了一些修正：RESET 之后内部的地址指针值清除为 0，再用送到 XTAL1 的脉冲决定烧录的地址。烧录时要把 RST 引脚拉到+12V，先送出地址脉冲，接着送出待烧录的数据，然后启动烧录脉冲约 1.2ms，就可以将数据烧入，看起来与 AT89C51 的烧录时序雷同。

图 10-2 是原厂提供的烧录时序。一天之内只看就这几页数据表，是真的那么难懂吗？非也，其实真正的作用是利用一天完整的时间将烧录器的相关数据完全纳入大脑中，让所有的思绪充分地搅拌，准备明天的烧录线路的修改。做开发的工作这段酝酿的过程是很重要的，而且只要一开始动手做后，就不可中途暂停，否则成功的机率是会打折的！

Mode	RST	P3.2 / PROG	P3.3	P3.4	P3.5	P3.7
Write Code Data	12V	⎍	L	H	H	H
Read Code Data	H	H	L	L	H	H
Write Lock Bit-1	12V	⎍	H	H	H	H
Bit-2	12V	⎍	H	H	L	L
Chip Erase	12V	⎍	H	L	L	L
Read Signature	H	H	L	L	L	L

图 10-2　ATMEL 原厂提供的 AT89C2051 烧录时序

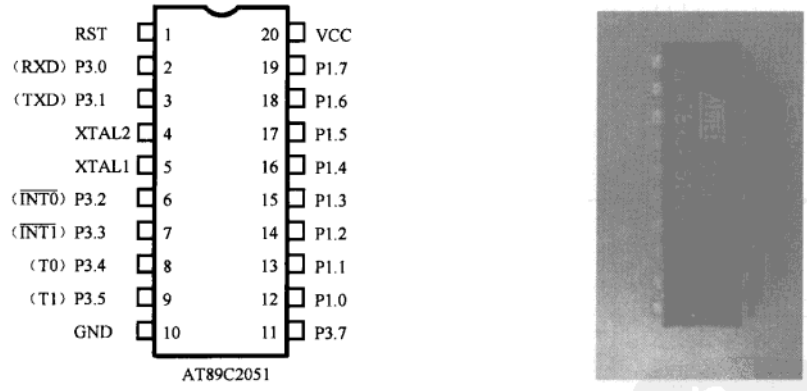


图 10-3　AT89C2051 的引脚图

10-2　星期二：线路修改

图 10-4 是原先 AT89C51 烧录器的线路图。图 10-5 则是我们新加入的线路，主要是变换烧录时各控制脚的引脚，如果我们的线路是 PIN-DRIVER 的话就不需要这个转换的线路了。增加的线路里添加了两个按键，我们打算拿来做数据读取与烧录的功能键，以便让这台烧录器不必与 PC 联机就能独立作业。当初公开 AT89C51 的线路后，就有部分的读者反应希望能加入该项功能，新增加的功能如果在线路板送 PC 板工厂制作前加入那就可以避开许多不必要的困扰，否则再做任何的修改都会使产品上市的时间落后三周以上。

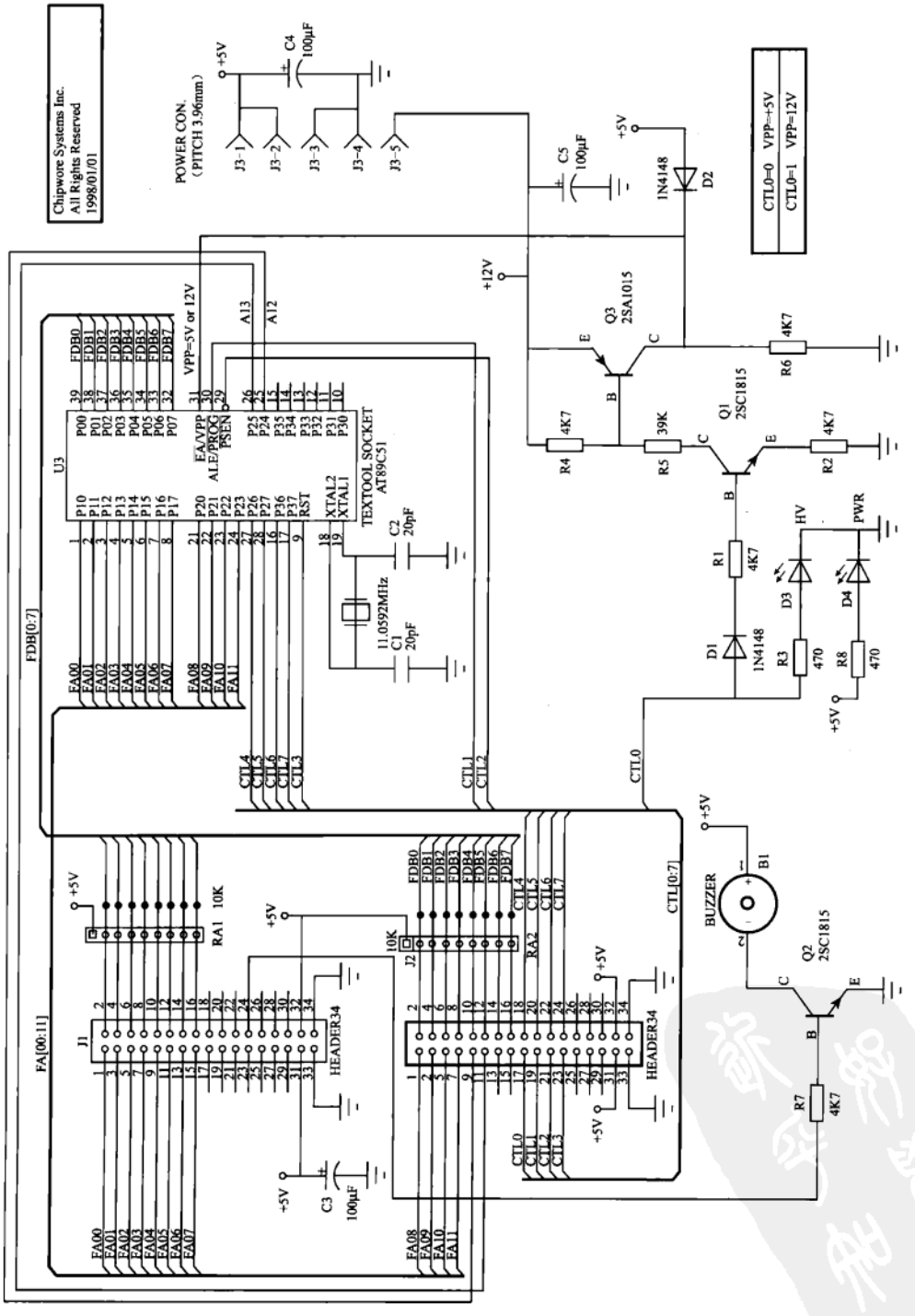


图 10-4 AT89C51 的烧录线路图

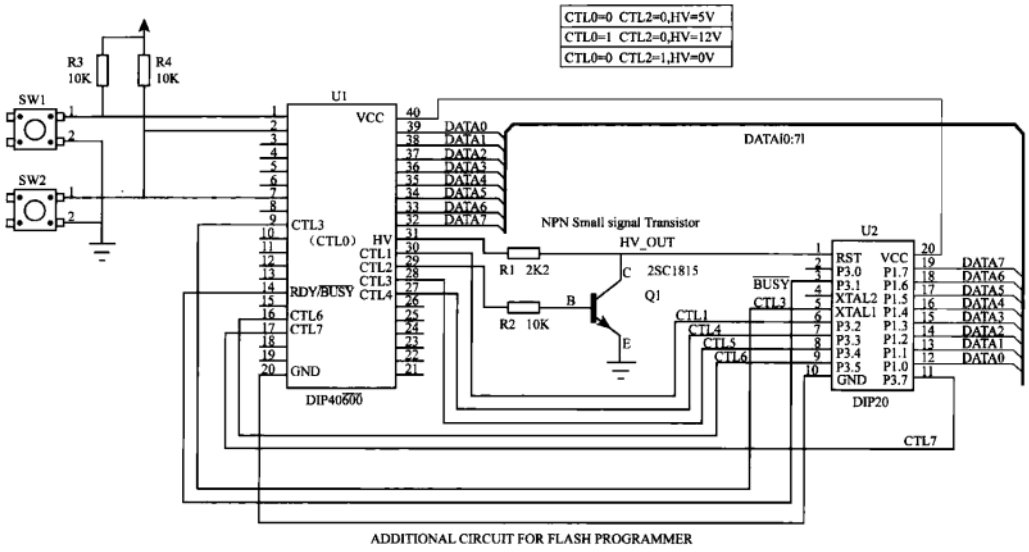


图 10-5　为了能烧录 AT89C2051 所增加的线路图

增加的小线路中较特殊的是把 RST 引脚的电平改成有三个电压（+12V、+5V 与 0V）的输出模式，原先在 AT89C51 烧录器中，RST 引脚只能做+5V 与+12V 的电压切换，但是在 AT89C2051 时，RST 引脚还要能够降到 0V，以便 RESET 内部的烧录地址，所以程序中改用两个位来控制 RST 引脚上的电压。图中的 NPN 小信号晶体管一经启动后，就会把 RST 引脚强迫拉到接近 0V 的电压。图 10-6 是我们试制的 PROTOTYPE 照片，我们是把线路焊在 FLAGTEST 万用实验板上，同时利用长脚的排针座与原来的 AT89C51 烧录器相连。请看图 10-7，焊这块实验板的诀窍是长脚的排针座要尽量靠边，以方便原来 40P 的烧录测试夹能够确实夹紧长脚的排针。

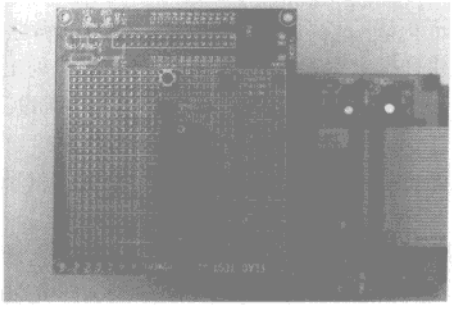


图 10-6　我们试制的烧录转换板 PROTOTYPE

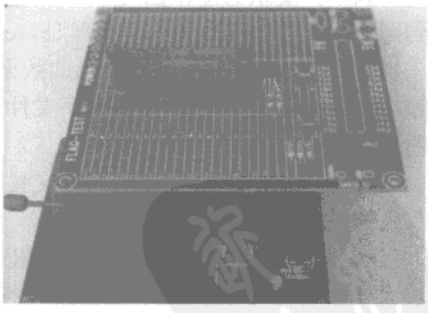


图 10-7　PROTOTYPE 另一个角度的照片，排针应该靠边以免卡到 40P 测试夹的开启

10-3　星期三：程序加入及验证

先前我们已经有 AT89C51 烧录程序的开发经验（整个程序是 2500AD 的 C 语言写的），因此 AT89C2051 的烧录程序架构应该雷同才对。这也就是说我们可以把原先的程序稍加修改

后，就可以给 AT89C2051 来用，两者最大的差异是：AT89C51 的烧录地址是用 12 条地址线来设定，而 AT89C2051 则只靠送到 XTAL1 的脉冲来递增烧录地址值。由于烧录程序是自行开发的，并且原先程序的注释也写得相当详细，所以修改的过程还算顺利。实际的修改顺序是：打印整个源文件内容，标出可能要修改的部分，修改烧录控制程序，整个程序再做优化处理，然后把原来的烧录操作改得更友好一点，如果我们一整天都在使用烧录器，一定会发觉哪里再做一些修改会更好，许多做法如果设计的人能多替别人想一下，就可以避免许多麻烦了，LET'S MAKE THIS PROGRAM BETTER!

10-4 星期四：波形观察及烧录

烧录程序刚写好后直接拿 AT89C2051 去烧录是很冒险的，我们的经验及一贯的除错步骤是这样的：

步骤 1：首先进行烧录的模拟，然后用示波器观察两个重要的烧录信号：RST 与 ROG（P3.2），确定 RST 可以做三种电压的输出，以及 PROG 的烧录脉冲宽度应该保持 1.5ms 以上的时间。

步骤 2：由于 AT89CXX 系列的 IC 都有身份码（SIGNATURE），所以一开始不加上+12V 的烧录电压，先读取 AT89C2051 的身份码，看看 READ CYCLE 对不对，对的话再继续做下面的试验。

步骤 3：取一颗新的且内部空白的 AT89C2051 并读取其身份码，如果与原厂公布的值相符时，再做一次读入 2KB 程序数据的操作，然后查看读入的数据，应该都是 FFH 才对，如果不是的话一定是控制引脚的状态值不对。

步骤 4：读入操作都验证无误后，加上+12V 的电源，并由 PC 端载入一个文字文件数据，文件长度最好是 2048 字节，然后直接进行烧录，由于先前我们已经验证过烧录电压与烧录脉冲的宽度，所以数据应该会很顺利地写到 AT89C2051 上。

步骤 5：烧录的操作在几秒内就完成了，通常我们会把烧录的数据与原先加载的数据从头到尾对比一次，这个操作称为验证 Verify，验证正确后就可以保证烧录数据的正确性。

步骤 6：验证完后再读入烧录的数据一次，并且指定用 DUMP 的方式在屏幕上列出烧录的数据，由于事前我们是加载文字文件的数据，所以可以很清楚地看到烧录到 AT89C2051 内的数据。

步骤 7：试验 AT89C2051 的 LOCK bit 1 的烧录功能，加上此一锁定后，该 IC 就不能再做写入的操作，除非是重新 ERASE。

步骤 8：试验 AT89C2051 的 LOCK bit 2 的烧录功能，加上此一锁定后，该 IC 内部的程序数据就完全读不到了，那也就是说执行空白检查时，烧录器会提示是“空白的 AT89C2051”，但是内部却是有数据存在。内部数据的完整保护模式 LOCK bit2 也可以在 ERASE 后解除保护，不过此时所有的程序数据已被全部清除成 FFH 了。

经过这一天的一连串实验，工作桌上的各种仪器与设备都应用后，总算把烧录程序的所有操作敲定了。在整个开发过程中，这个阶段是最紧张刺激的，有时都会错过吃中饭的时间，等到数据能够正确烧录后，时间也接近下午四点了，这个时候才觉得时间过得真快。

10-5 星期五：开始正式烧录

今天铁定是忙碌的一天，工作室里一共准备了三台计算机来做烧录 AT89C2051 的配合操作。

第一台 PC 做烧录主程序的修改用。

第二台 PC 则接万用烧录器，接收第一台计算机输出的二进制文件数据，立即烧录成 27512 或 27256 的 EPROM。

第三台 PC 则接 FLAG51 单片机控制板，双方以 RS232C 串行通信的协议连接，烧录好的 EPROM 则插到 FLAG51 的 ROM 插座上，FLAG51 控制板上则接上 AT89C51 烧录板，而 40P 的测试夹再夹上我们的 AT89C2051 PROTOTYPE 烧录板。

一整天的时间我们都在 AT89C2051 烧录器前做各项清除（Erase）、空白检查（Blank check）、烧录（Program）与数据对比（Verify）的操作，一发现有什么地方有不妥之处，就立即修正原始程序，然后烧录新的 EPROM，再查看修改后的结果是否正确，就这样修修改改又过了一整天的时间。不过，粗略地统计了一下，修正的地方大部分与烧录程序无关，因为最重要的部分已经在昨天通过了，今天的所有修改只算是“插花”而已。

```
*****************AT89C2051 PROGRAMMER commands list*****************
? or/        -Commands list
b lank       -Blank test or At89C2051
e rase       -Erase AT89C2051 flash memory 2048 bytes
p rogram     -Programming from RAM buffer(A000H-A7FFH)
A uto        -Erase,Blank check,Programming then Lock bit1 & bit2
v erify      -Verify AT89C2051 & RAM BUFFER
s ignature   -Read signature of the chip
L file       -Download binary file from PC to RAM BUFFER
S file       -Upload RAM BUFFER(A000H-A7FFH)to PC
1            -Lock bit1 of AT 89C2051(Futher programming is disabled)
2            -Lock bit1+2 of AT89C2051(Verify is disabled)
r ead        -Read then save at RAM BUF(A000H-A7FFH)
d isplay     -Display RAM BUFFER(A000H-A7FFH)
c lear       -Clear memory RAM BUFFER (A000H-A7FFH)
o            -back to DOS shell
o uit        -Exit to DOS
Hints for programming:
[COPY     mode]:(R)->new AT89C2051->(P)->(1)or(2)lock bit
[NORMAL   mode]:(L)a file->put on a new AT89C2051->(P)->(1)or(2)
[VERIFY   mode]:(L)a file->put on a AT89C2051->(V)
[AUTO     mode]:(L)a file->put on a new AT89C2051->(A)
*****************************************************************
```

图 10-8 AT89C2051 烧录时在 PC 上所看到的功能选择表

烧录程序最后加入了增加两个按键的检查，不过两个键可有三个功能，按下 READ 键并立即放开时，烧录程序会读取 AT89C2051 的程序数据到 FLAG51 的 SRAM 当中，再按下 PROGRAM 键就可以进行数据的烧录。如果连续按两下 READ 键时，代表验证烧录数据与 SRAM 内数据的对比，如果正确时，蜂鸣器会发出一长声，反之则发出三个短暂声响，表示

有错误出现。有了这两个按键之后，AT89C2051 烧录器若只单纯要拷贝程序数据时就不需要与 PC 联机了。

10-6 星期六：寿命测试

自己开发烧录程序有许多好处，首先就是有百分之百的自主权，可以随时修改程序，另一项好处是可依需要开发一些特定的测试程序。以这次我们开发的 AT89C2051 烧录器为例，该 IC 是随时可擦除的，所以我们在程序中加入一个测试指令“^”，只要按下这个键（按着 SHIFT 键不放开再按数字键 6）后，烧录器就会重复不断地执行清除、烧录然后验证等一连串的操作，每次烧录的时间约是 10s 左右，所以每分钟约可以做六次烧录。我们任意取出一颗 AT89C2051，进行长达一天左右的重复烧录与清除实验，烧录总次数超过了 5000 次以上！

由此证明 AT89C2051 标示有 1000 次可擦写的数据是可以相信的，另外也间接证明了我们设计的烧录时序是正确的！不良的烧录时序一定会对这类元件的寿命打了很多折扣。其实，可编程元件的烧录条件往往是开发烧录器厂商所无法完全掌握的，如果 IC 制造商不实时通知更改烧录时序，保证会让用户与烧录器制造商两方面吃亏，如果用户一直抱怨烧录某个 IC 有问题时，烧录器制造商应该有错就改，怎么可以对用户说“你准备挨告”呢，连技术上的事实陈述也不行。

```
AT89C2051 lifetest start...

*****PGM count=0001H*****
Programming...
Erase AT89c2051 flash memory
Programming...
pppppppp
Verify...
Readback Checksum=F360H

*****PGM count=0002H*****
Programming...
Erase AT89c2051 flash memory
Programming...
pppppppp
Verify...
Readback Checksum=F360H
```

图 10-9 进行寿命试验的画面

10-7 星期日：好戏上场

整个烧录程序与线路都确定无误后，接下来是 PC 板的布局和相关元器件的订购了。在 PC 板方面我们分别设计了烧录转换板与 8051 转换板，请看图 10-10，前者是供烧录用的转换板，后者可插在原有的 8051 控制板上，但仅限使用内部程序空间的应用。这也就是说，如果您是用 8751 或 AT89C51 做控制应用，并且完全没有使用外部的程序存储或数据存储时，而且您写的程序长度在 2048 字节以内，那就可以用 AT89C2051 来取代。不过，原先的 PC

板是 40P 的 8051 标准引脚，必须加上转换板后，才能符合 AT89C2051 的引脚。

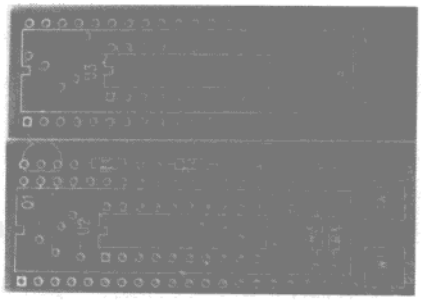


图 10-10　两块转换板的元件位置图

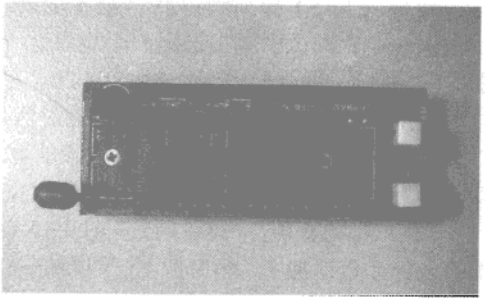


图 10-11　AT89C2051 烧录板的完成图

AT89C2051 特性简介

AT89C2051 是一个内置 2KB 闪存与 8051 单片机全兼容的 CPU，引脚数只有 20，引脚图请参看图 10-3，而且可用电气的方法直接清除内部的数据。由于引脚数减少了，所以 AT89C2051 的输入/输出引脚剩下 P1.7～P1.0 与 P3.7～P3.0（少了 P3.6 脚），整个 IC 的有效输入/输出引脚有 15 个，在一般的小应用场合中已经绰绰有余了。AT89C2051 仍保留了串行传输的 RxD 与 TxD 引脚，跟外界通信的协议与模式和 8051 完全相同。所以您以前在 8051 单片机下的各项“功夫”及程序，移植到 AT89C2051 上（除了 P3.6 外）百分之百行得通。

以下是制造厂公布的主要特性：

- 与原来的 MCS-51 相关产品完全兼容。
- 2KB 的内部闪存。
- 闪存可允许 1000 次的写入与清除，数据保留时间可超过 10 年。
- 工作电压可由 2.7V～6V。
- 工作频率可由 0Hz～24MHz。
- 提供两道数据保护锁（LOCK BIT）。
- 内置 128 字节的数据存储器。
- 两个 16 位的计数器。
- 五个中断源（INT0、INT1、T0、T1 和串行中断）。
- 串行通信的 UART 和 8051 完全相同。
- 输出点可直接驱动 LED，不需再加驱动缓冲器。
- 提供一个模拟式的比较器（Analog Comparator），其输出点在 P3.6 位上。
- 提供 LOW POWER 及 POWER DOWN 的功能。

ATMEL 原厂建议的烧录步骤如下。如果您想知道所有的烧录提示，还是要参考原厂的数据手册，一切烧录的操作应该以数据手册上的为准。

步骤 1：电源打开顺序：在 Vcc 与 GND 间加入+5V 的电源，把 RST 和 XTAL1 两点接到 GND 电位，其他引脚浮接至少保持 10ms。

步骤 2：把 RST 引脚提升到“H”，把 P3.2（PROG）提升到“H”。

步骤 3：切换控制引脚 P3.3、P3.4、P3.5 和 P3.7 到要烧录的状态模式（如果要进行烧录或验证时）。

步骤 4：把要烧录到地址 000H 的数据送到 AT89C2051 的 P1 端口上。

步骤 5：把 RST 提高到+12V，准备烧录。

步骤 6：在 P3.2 引脚上送出一个烧录脉冲，此时数据会烧入内部的闪存中，原厂的资料特别提到写入周期时间会自动调整为 1.2ms 左右，这也就是说，烧录 1 字节至少要 1.2ms 的时间，烧录完整个 AT89C2051 要 2.5s 的时间。

步骤 7：在烧录模式下若要验证烧录的数据时，把 RST 脚再降到“H”，再把四只控制引脚切到读入（READ）的模式，这时烧录的数据就会出现在 P1 端口中。

步骤 8：要烧录下一个地址的数据时，在 XTAL1 引脚上再送入正相的方波，这会使 AT89C2051 内部的地址计数器加 1，然后把下一组要烧录的数据放到 P1 端口上。

步骤 9：重复步骤 5～8 的操作，直到 2048 字节的数据都被烧写入 AT89C2051 为止。

步骤 10：烧录完后取出的顺序：把 XTAL1 降成“L”，把 RST 降成“L”，浮接所有的输入/输出引脚，关闭 Vcc 与 GND 电源。

下面这个程序 AT89C2051.C 是 FLAG51 控制板的烧录应用之一，我们很乐意将所有程序内容公开，再加上先前已经公开硬件线路了，这个程序可以烧录 AT89C51/52 以及 AT89C2051/1051 共四种 Microcontroller。

你也可以参照软硬件自己做一个 AT89C51 的烧录器。我们要提省你的是，程序当中的延迟时间是针对 FLAG51 的，如果系统的石英晶体频率不是 11.0592MHz 时，你可能要做一些修正，才能让烧录器正常工作。

AT89C2051 烧录程序请查看随书光盘中的 CH10_AT89C2051 烧录程序彻底公开文件。

本章使用的软件

（1）2500AD 8051 Assembler。

（2）IAR 8051 C Compiler。

本章使用的硬件

（1）FLAG51 单片机控制板。

（2）AT89CXX 烧录板。

（3）AT89C2051。

（4）“河洛”万用烧录器 Universal Programmer。

相关资料网站

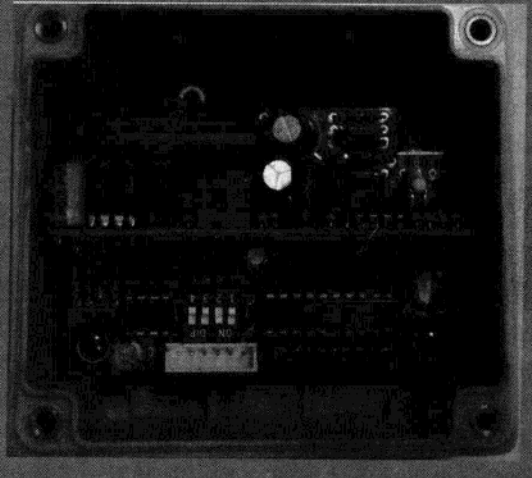
可经由下列公司、网站取得更进一步的信息：

http://www.chipware.com.tw：查询 8051 单片机控制器资料。

http://www.atmel.com：查询 AT89C2051 的烧录资料与应用范例。

http://www.adobe.com：下载免费的 Acrobat Reader 电子书 PDF 浏览程序。

11



旗威科技设计的高精度（24 位）AD 转换板，专门测量 load cell 或温度的信号。

第 11 章　Flash 编程器的后续开发

这一章里我们提出用 C 语言写烧录程序的可行性，虽然执行速度稍慢，可是开发时间却有效地减到最少，请看我们详尽的分析。

最近几个月电子类杂志的广告上，经常看到 AT89C51 的行踪，由这种现象看来，AT89C51 与 AT89C52 逐渐取代 Intel 8751 与 8752 已经是不能避免的。

11-1　AT89C51 会取代 8751 吗

AT89C51 将逐渐取代原有 8751 的趋势，主要因素有三：

第一个优势是 AT89C51 的 ERASE 擦除时间只要 10ms，而 8751 却要在紫外灯下，连续照 20min 左右方能清除数据。

第二个优势是 AT89C51 的价格要比 8751 便宜，8751 不仅较难买而且还比较贵，你还会考虑 8751 吗？

第三个优势是 AT89C51 的数据保护措施做得相当好，硬件高手能解开此 IC 的机率非常低；而 8751 虽号称可加密，禁止别人读取内部的程序，可是破解的方法却很简单，只要不到 100s 的时间就可以解开，在业界上这已经算是一个半公开的秘密。所以，AT89C51 取 8751 而代之，已是不可避免的。

11-2　AT89C52 已经上市了

当 AT89C51 烧录器的程序刊登出来没几天后，我即收到世环公司的 AT89C51 进一步资料，其中较引人注意的是：AT89C52 已上市了！ATMEL 公司已在 1994 年 7 月正式推出了容量加倍的 AT89C52，其最高的工作频率可达 24MHz，有了这颗内含闪存的 IC 后，写较大的应用控制程序就没什么困扰了，价格仍然比 Intel 的 87C52 便宜。基本上，AT89C52 的特点与 8752 是完全相通的，只差在前者内部的程序内存是快闪存，而后者是 EPROM 架构，清除时较花时间罢了。AT89C52 的烧录过程几乎与 AT89C51 类似，倒是 ATMEL 公司也发觉：AT89C51 烧录时用软件指定 Vpp 的烧录电压，稍有不慎就很容易造成芯片的损坏，于是就在制造时就直接指定烧录的电压，并且在 IC 的标示上做一区分。例如若只标示 AT89C52 则代表 Vpp 是 12V 的烧录电压，若另加有-5 的标示时，代表 Vpp 是 5V 烧录的，如果用程序读取 AT89C52 的验证码时，也可以判断 Vpp 的烧录电压是多少。图 11-1 是 AT89C52 烧录及验证时的模式设定表，应该与 AT89C51 几乎相同才是。

原先我们设计的烧录线路少了一条地址线 A12，只要用一条镀银线加焊上去即可烧录 AT89C52，当然烧录程序要再做稍微的修正才能够烧录满 8192 字节的空间，由于先前我们已把 FLAG51 控制板的 RAM 空间放大成 32KB，所以烧录加验证的空间都绝对足够，至于程

序如何做修正呢？请继续看以下的叙述。

AT89C52 FLASH PROGRAMMING MODES

MODE	RST	PSEN	ALE/PROG	EA /Vpp	P2.6	P2.7	P3.6	P3.7
WRITE CODE DATA	H	L	低脉冲	H/12V (1)	L	H	H	H
READ CODE DATA	H	L	H	H	L	L	H	H
WRITE LOCK BIT-1	H	L	低脉冲	H/12V	H	H	H	H
BIT-2	H	L	低脉冲	H/12V	H	H	L	L
BIT-3	H	L	低脉冲	H/12V	H	L	H	L
CHIP ERASE	H	L	低脉冲 (2)	H/12V	H	L	L	L
READ SIGNATURE BYTE	H	L	H	H	L	L	L	L

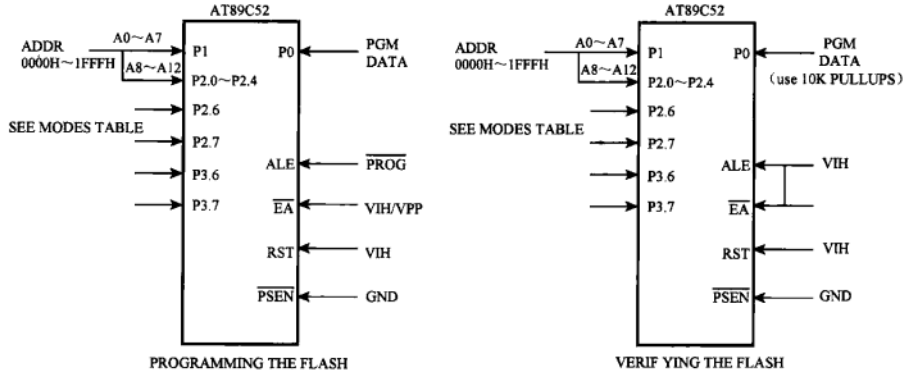


图 11-1　AT89C52 的烧录模式对照表

注：(1) AT89C52 不能用程序来定义 Vpp 的烧录电压，取而代之的是读取 032H 的验证码 SIGNATURE 值，如果 (032H) =FFH 代表采用+12V 电压烧录，(032H) =05H 时代表+5V 的电压烧录。

(2) FLASH ERASE 时，PROG 的脉冲宽度至少是 10ms。

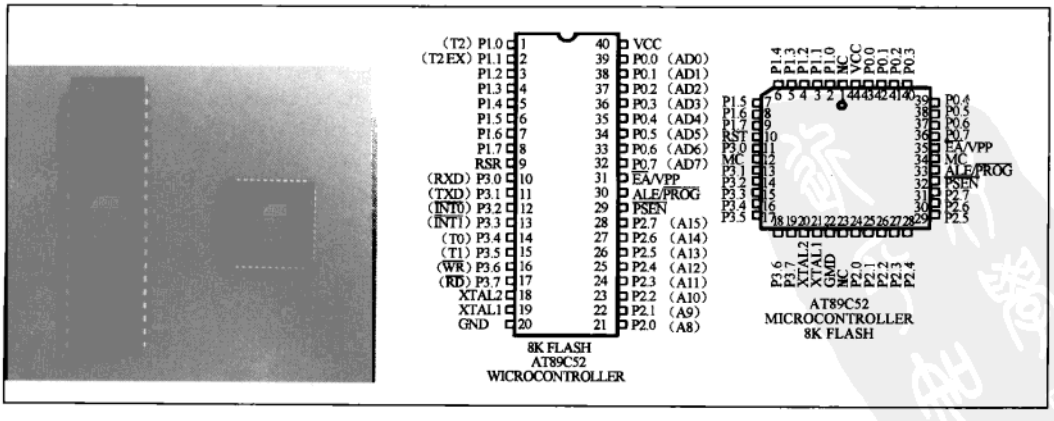


图 11-2　AT89C52 的实物图与引脚图（DIP 封装与 PLCC 封装）

11-3 AT89C51 烧录机的再修改

我们已在第 9 章中完完整整地介绍了 AT89C51 的烧录线路与程序，烧录操作最好是采用模块化的设计，不仅修正时较为方便，别人看程序也不会很吃力，同时也用这个烧录程序烧录不少 AT89C51；另一方面，我们也把 AT89C51 特殊的 LOCK bit 功能加上，只要按一个键就开始进行所有的烧录及程序锁定的操作，可是操作过程还是复杂了一些。

一位用户提供了一个意见：我的 AT89C51 烧录器打算给生产线上的员工使用，操作希望越简单越好。所以，我们又把整个程序重新研究一次，尽量朝方便操作的前提着手，不仅烧录程序更加完整，同时，PC 端的联机程序也要做一些修正才行。整个修正及程序修改的时间约持续一星期左右，整体操作过程试了又试，以期让用户能够顺顺利利地烧录完所有的 AT89C51。

其实，修改到此地步时，这台烧录器的特点与便利性已经和市售的万用烧录器相当了，虽然它只能烧录 AT89C51 内部的闪存，因为我们很详尽地公布了整个烧录线路以及源代码程序，所以它的教育性质是最佳的。如果您只想烧录 AT89C51 做程序的保护，那我们建议您买市售的万用烧录器，反之，如果您想知道整个烧录程序是如何分别用软硬件完成的，那旗威的 AT89C51 烧录器可能是一个相当完整的学习工具了。图 11-3～图 11-8 是在 PC 端操作 AT89C51 烧录器时所看到非常友好的界面。

```
AT89C51>/
**********************AT89C51 PROGRAMMER command list**********************
?   or /          -Commands list
5                 -Setting VPP of programming voltage= +5V
e   rase          -Blank test of AT89C51
p   rogram        -Erase AT89C51 flash memory 4096bytes
A   uto           -Programming form A000H-AFFFH(RAM BUFFER)
V   erify         -Erase,Blasnk check,Programming then Lock bit1&2
r   ead           -Verify AT89C51 & RAM BUFFER
S   ignature      -Read then save at A000H-AFFFH(RAM BUFFER)
L   file          -Read signa ture of the chip
S   file          -Load binary file from PC to RAM BUFFER
1                 -Save TSM BUFFER A000H-AFFFH to PC
2                 -Lock bit1 of AT89C51(MOVE opcode from ext.are disabled)
3                 -Lock bit1+2+3 of AT89C51(external execution disable)
d                 -Display RAM BUFFER(A000H-AFFFH)
X                 -Clear memory RAM Buffer(A000H-AFFFH)
O(upper)          -DOS shell
Q(upper)          -Exit to DOS
Hint for programming:
    [copy mode]:(R)->new AT89C51->(1) or (2)lock bit
    [test mode]:(L)a file->put on a new AT89C51->(P)
*******************************************************************************
AT89C51>
```

图 11-3 只要按下“？”或“/”键即可查询所有的功能

```
AT89C51>P
Programming...
set VPP code=5V
Erase AT89C51 flash memory
Programming...
PPPPPPPPPPPPPPPP
Verify...
AT89C51 chip program successfully
AT89C51>
```

图 11-4　烧录时所显示的信息

```
AT89C51>P
Programming...
Set VPP code=5V
Erase AT89C51 flash memory
Programming...
PPPPPPPPPPPPPPPP
Verify...
Fail in programming
Something wrong in programming AT89C51
Please check:
1.Is AT89C51 on the TEXTOOL socket?
2.Did you use command ˉ5ˇ  before programming?
3.Did you add optional+12V for command ˉ5ˇ command?
4.Maybe this chip is bad before programming.
5.Maybe this chip is not AT89C51,pls read signature.
```

图 11-5　烧录失败时，烧录器送回一连串可能的错误信息

```
AT89C51>1
Lock bit1 of AT89C51
MOVC instrucition from external program are disabled
Chip signature are:(30H)=1EH,(31H)=51H
```

图 11-6　LOCK bit 1 时，烧录器告诉用户不能使用 MOVC 指令去读取内部的 4KB 数据

```
AT89C51>2
Lock bit1+bit2 of AT89C51
Same as lockbit1 plus verify is disabled
Chip signature are:(30)=FFH,(31H)=FFH
```

图 11-7　LOCK bit 1 与 bit 2 时，提示用户不能使用 VERIFY 程序，同时显示出此时的验证码（SIGNATURE）

```
AT89C51>3
Lock bit1+bit2+bit3 of AT89C51
Same as lockbit2 plus eternal execution is disable
Chip signature are:(30H)=FFH,(31H)=FFH
AT89C51>
```

图 11-8　LOCK bitl+2+3 时，提示用户只能使用 4KB 的程序空间来写程序了

11-4 烧录程序也可用 C 语言来处理

前面我们曾提到只花了一星期左右的时间做程序的调整，由于最近几年来一直都在 8051 单片机方面打转，因此也累积了这方面不少的经验与设备，其中最值得一提的是：我们一直在使用 C 语言来写控制程序，而且愈来愈熟练。由于有此优势，因此 AT89C51 的烧录程序在做修正时，也慎重地考虑选用 C 语言来处理是否恰当，经过一番仔仔细细的评估后，认为可行性相当高，于是就开始着手进行修正及除错了。我们选用的 C 是美国 2500AD Software Inc. 的 C 编译程序，选用的理由主要是价格较为便宜，不过，使用手册写得并不很详细，造成使用与学习上的许多障碍。市面上另外可看到 FLANKLIN C 和瑞典的 IAR C，除非是有特别的研究预算，否则实在是舍不得买。

使用 C 语言做控制时，大部分的人都会怀疑：处理速度够快？我的经验是这样的，C 语言的处理时间约是汇编语言的 10～20 倍，汇编语言中每个模块（MODULE）的处理速度可用几十个微秒来计算，但是 C 语言的模块则必须以毫秒来计算其执行时间，但是 C 语言的开发时间可较汇编语言节省至少八成以上，所以除非是对时间要求相当苛求的应用，否则我们强烈建议初学 8051 单片机的读者若只想写程序，可以先从 C 学起，至少遭遇失败的机会较少一点。

但是如果要同时兼顾软硬件时，一定要把 8051 汇编语言学好后再来写 C。现在大部分的 C 程序编译器都支持所谓的 inline assembler（可在 C 程序中直接使用汇编语言），所以大部分的处理速度都是可以令人接受的，以修正 AT89C51 烧录器的程序为例，唯一须注意的是要产生一个 10ms 的时间延迟信号给 $\overline{\text{PROG}}$ 引脚。10ms 对 C 或汇编语言而言都不是问题，所以，我就大胆地用 C 来做烧录的各项控制了，过程当然是相当痛苦的，但是最后还是完成任务了。

程序修正后，我们做了汇编语言与 C 语言间的一些比较，其中比较明显的有：汇编语言烧录 AT89C51 的时间只要 7s 左右的时间，C 语言则花较长的时间约是 15s，较前者多了一倍。程序数据验证（VERIFY）时，汇编语言只花了 3s 不到的时间，用 C 来处理时还是要花 10s 的时间。C 语言在处理速度时是落后于汇编语言的，可是从另一个方向来看时，C 语言用于程序修正及除错的时间最短，一发现错误到修正完毕，通常只花 30min 的时间，这可是写汇编语言时很难办到的。十年前因工作上的需要，我将公司另一位工程师的 IEEE-488 接口汇编语言驱动程序（花了一年半的时间来开发）改用 C 来处理，却只花了两个月的时间就完成了，这也间接印证了我们用 C 语言来写 AT89C51 的烧录程序绝对是可行的。

11-5 烧录程序的最新版本

经过了几年的使用与用户的众多意见，我们最新的版本可以烧录多款 ATMEL 的 Microcontroller，分别有：

- AT89C51：4KB Flash。
- AT89C52：8KB Flash。
- AT89S8252：8KB Flash+2KB E^2PROM。
- AT89C2051：2KB Flash。
- AT89C1051：1KB Flash。

以上几个微控器已经包括许多应用领域了，虽然烧录板背后多了几条镀银线不是很好看，但是有所得一定会有所失的。

Atmel 烧录程序的发展历程

好几年前，当我们拿到 Atmel AT89C51 的完整资料时，就着手设计烧录线路与编写该芯片的烧录程序。AT89C52 正式出厂时，我们也跟着修改部分程序内容及硬件，以便烧录器能够同时处理 AT89C51 与 AT89C52 的烧录。严格讲起来，这些修正都算是小修改罢了。

直到我们拿到 20 引脚 AT89C2051 与 AT89C1051 的资料时，发现其烧录时地址线的设置方式完全不同，我们只好用再外加一个 20P 的测试夹应对。当然此时的软件程序也做了大幅度的修改，之后 Atmel 公司又推出 AT89C4051，也是 20 引脚的 8051 单片机，内部程序提高到 4KB，由于其烧录方式与 AT89C2051 极为类似，所以我们修正的只有程序的部分而已。稍后我们也从网络 http://www.atmel.com 下载了 AT89S8252 与 AVR90S1200 的烧录资料，前者烧录时与 AT89C52 类似，后者则与 AT89C2051 类似，所以，我们就把这几个 IC 的烧录程序合并在一起，统称 AT89CXX 烧录程序。

AT89CXX 烧录程序请查看随书光盘的 CH11_AT89CXX 烧录程序彻底公开文件。

本章使用的软件

（1）2500AD 8051 C Compiler & Assembler。
（2）IAR 8051 C Compiler。
（3）Franklin C Compiler。

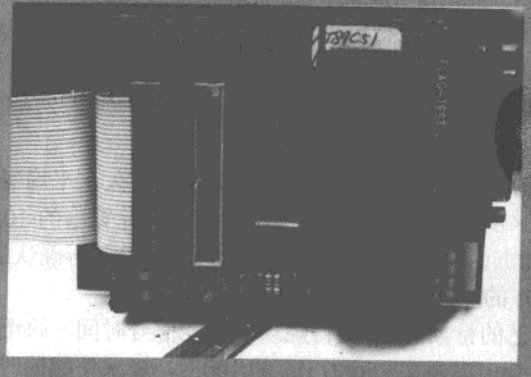
本章使用的硬件

（1）FLAG51 单片机控制板。
（2）AT89CXX 烧录板。

相关资料网站

可经由下列公司、网站取得更进一步的信息：
http://www.chipware.corn.tw：取得单片机控制板相关资料。
http://www.iar.com：取得 8051 C Compiler 相关资料。
http://www.keil.com：取得 8051 C Compiler 相关资料。

12



我们试做的 AT89C51 烧录器，需配合 FLAG51 单片机控制板才能工作。

第 12 章　揭开 EPROM 烧录的秘密

> 容量是 512KB 的 EPROM 竟然能在十几秒内完成烧录，这是 EPROM 制造厂的功劳还是烧录器设计者的“魔术功能”呢？请看看我们的技术深入分析。

作者是大三的学生时，在杂志上发表的第一篇文章就是 2716 EPROM 烧录器的制作，那时一颗 2K×8 的 2716 EPROM 叫价是 100 元上下。每次进行烧录时往往是战战兢兢的，深恐稍一不小心就会损失好多天的生活费。烧录一颗 2716 的时间约是 100s 左右，这段时间往往是最难熬的！可是最近几年推出的 EPROM 27512（64K×8）或 27256（32K×8），竟然能在十几秒内完成烧录，这是 EPROM 制造厂的功劳还是烧录器设计者的“魔术功能”呢？请看看我们的分析与探讨。

在所有我们使用的 IC 中，EPROM 的资料可能是最容易找到的，因为只要有微处理器的机器一定有机会用到 EPROM，最常见的例子是存放执行的程序（PROGRAM），也可以存放固定的数据（DATA）或是图像数据等等。早期的微处理器内部是不含程序空间的，所以一定是靠外部的 EPROM 存放程序数据。近几年来甚为流行的单片机（8051/Z8/68705）是内置程序空间的微处理器，但是程序都限定在 8K×8 以内，如果您写的程序超过 8KB 时，还是要靠外部的 EPROM 来存放部分的程序。

12-1　EPROM 烧录的方法

基本上，EPROM 烧录程序是非常单纯的，当我们要把数据烧录到 EPROM 内部时，首先要由外部供应一个烧录电压（大约是 12～13V 左右），然后在 EPROM 的地址线与数据线上摆好烧录的数据与地址，然后启动烧录使能信号，时间约是数百微秒到数十毫秒，这样就能把一个字节的数据烧录到我们所指定的地址上。如果您要烧录一颗 32K×8 的 27256 EPROM 时，您必须有 32K×8 的 SRAM 空间事先存放待烧录的数据，然后将数据一次一个字节送到烧录 IC 座上，并加入烧录的使能信号，重复上述的操作 32768 次之后就可以完成数据的烧录。

早期 2716 EPROM 的烧录使能信号一定要有 50ms 这么久，平均每秒仅能 200 个字节，所以烧录整个 2716（2K×8）要花 102s 的时间，而这里所计算出的 102s 是指烧录使能信号所要花的时间，如果再加上由 SRAM 取出再转存的时间及烧录再确认的时间，整个 2716 烧录过程势必要花超过 110s 的时间。

很幸运地，半导体工艺的持续进步，使得烧录的使能信号时间一再缩短，由最初 2716/2732 的 50ms、2764/27128 的 1ms 以及 27256/27512 的 100μs，烧录使能的时间减到了原来的五百分之一，但是内存的容量却是原先 2716 的 16 倍或 32 倍。随着内存容量的大增，原先沿用 EPROM 烧录的法则起了一些变化，各个厂商都提供各自最佳的烧录方式给烧录器制造商参

考，以下就是这几种烧录方式的介绍。选错烧录方式不仅会使烧录时间加长，而且会导致储存的数据非常不稳定，使得微电脑系统无法顺利使用正确的数据。

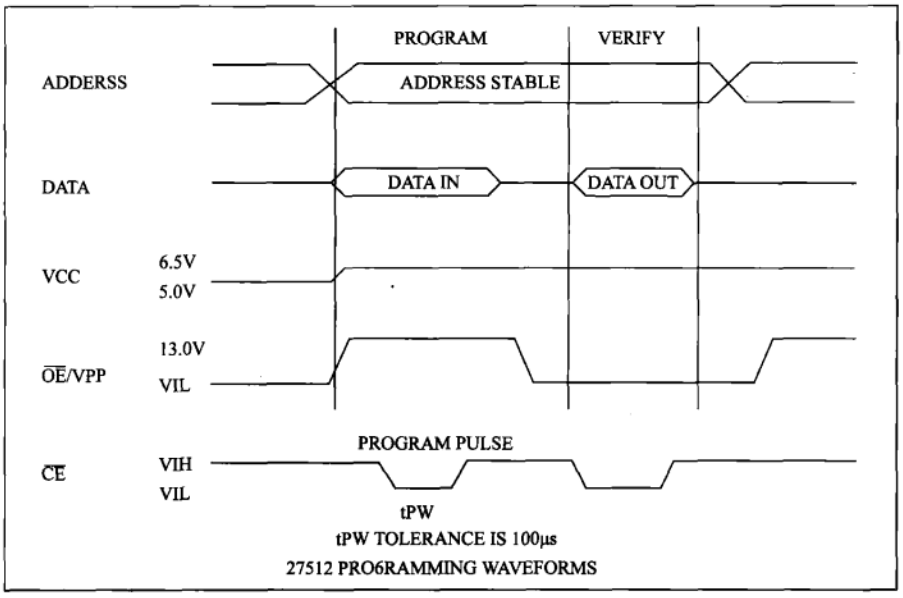
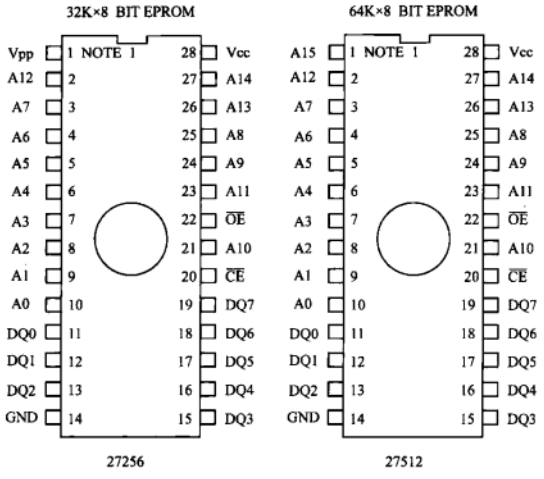


图 12-1　EPROM 的标准烧录时序，27512 与 27256 的 tPW 已经可以少到 100μs 以内

- NORMAL MODE：最初的烧录方式，50ms 的烧录使能信号宽度，早期古董级的 EPROM 一定要用此一方式烧录。如果 27512（64K×8）用此方式烧录时，可是要花上 50min 以上的时间。
- INTERATIVE MODE：这种烧录方式是每次加上 0.5ms 或 1.0ms 的使能信号，每次烧录使能后就检查该地址的数据是否已经是待烧录的值，是的话就继续下一个地址的数据烧录。当所有的地址都烧录完成后，再重新以两倍时间的烧录使能信号进行所有地址的烧录，让 EPROM“加深印象”。如果第一次的烧录使能信号后，对比验证不符时，烧录程序应该再 TRY 25 次，若都不成功则放弃该 EPROM 的烧录操作。
- FLASH MODE：这种模式是以 100μs 的使能信号对 EPROM 进行烧录。如果烧录不成功则重复试着烧录 25 次，否则才放弃烧录。这种模式也有人称为 QUICK MODE。
- RAPID MODE：这种模式也是以 100μs 的使能信号对 EPROM 进行烧录，不过它的烧录过程和 FLASH 有一点不同，RAPID 模式先对所有的地址进行 100μs 的烧录使能，然后再进行各个地址数据的对比验证（VERIFY）；如果不符时再做 10 次的重试，使能信号依然是 100μs 的宽度；若一直失败时就放弃该 IC 的烧录。

以上所提的各种烧录模式在 EPROM 制造商的数据手册上都有详细的记载，也有原厂建议的烧录模式，但是大抵是上述几种模式或是稍微的变异体。如果以现在较常用的 27512 或 27256 为例，烧录的模式应该是 FLASH MODE（QLJICK MODE）或 RAPID MODE，因为唯有采用这两种模式时，才可能把烧录时间缩减到 20s 以内。



EPROM的引脚图

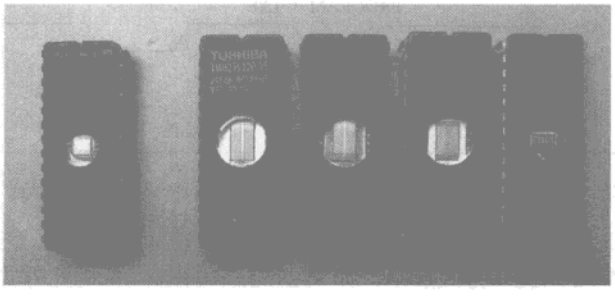


图 12-2　五个 EPROM，最左边的是 2764（8K×8），
右边四个全都是 27512，内部芯片随制造技术的改善愈来愈小

在烧录时还有一点要特别留意的是：除了 NORMAL MODE 外，其他模式烧录时，烧录器必须供应两个额外的电压，一是烧录电压，范围由 12.5V 到 13.75V 视 EPROM 的厂商而定。另一个是 EPROM 的供应电压，电压值由正常的 5V 提升到 6～6.5V 间，这会使每个字节烧录时间少到 100μs 以内。如果烧录时我们没有提升这个电压，可能无法以 100μs 的烧录脉冲烧录 EPROM。许多 EPROM 烧录器的制造商都有做这方面的电压调整，但是他们都不会告诉您为什么要这样做。

其实，每种电子产品的制造商除了要将产品尽量推广外，还附有教育消费者的义务。十几年前许多电子仪器设备都附有详细的框图以及基本的线路分析。但是现今的产品已经很少见到这样做的厂商了，就拿个人计算机来讲，主板的说明书就只有薄薄几页，许多专有名词完全不做交待。虽然计算机的变化甚多，但是至少要引导对这些有兴趣的消费者如何通过 Internet 或图书馆去查寻这些资料。

12-2　EPROM 烧录线路的安排

EPROM 烧录器的线路从理论的观点来看，应该是最单纯不过的，因为只要送出地址与数据线，再控制烧录的电压及烧录使能线，就可以进行烧录。但是如果我们把各种烧录状况

及电压调整都列入考虑时，情况就不是那么单纯了，因为随着存储器容量的倍增，这会使得几个重要控制脚的状态更加复杂，所以，在设计的初期我们就把要烧录的 EPROM 种类限定在最常用的 27512 与 27256 两种，而且是 12.5V 的烧录电压的 EPROM 方能烧录，符合这些规格的 EPROM 十年前就有了，现在的价格应该是几十元，所以把烧录目标限定在这里，除了给设计者方便外，使用者也可以用最合理的价格取得一台最有用的 EPROM 烧录器。

图 12-3 是我们所规划的烧录器线路图。图 12-4 是烧录电压的控制线路图。我们还是通过 FLAG51 单片机控制板上的两个 8255 来产生所有烧录的信号，所以线路板上安排了两个 34P 的接头与 FLAG51 的 CN1 与 CN2 相通。图 12-5 是本烧录器的方框图。线路上用了两个 32K×8 的 SRAM，供做烧录专用的 64KB 缓冲空间，这个空间是由 8255 另行控制的，与 FLAG51 的系统 SRAM 区完全没有关系，线路中还有一个 28P 的测试夹座。在 27512 与 27256 的烧录线路中，最重要的烧录控制点是第 1 引脚与第 22 引脚。当烧录 27256 时，第 1 引脚是 Vpp 的烧录电压输入引脚，第 22 引脚则是输出使能引脚 $\overline{OE}$（output enable），可是当换成 27512 时，其容量是 27256 的两倍，第 1 引脚变成了最后一条地址线 A15，第 22 引脚则变成 Vpp 烧录电压的输入引脚，可是正常读取程序数据时，又充当 $\overline{OE}$ 使能引脚。由于烧录时有这些需求，所以在电路当中，我们允许烧录测试夹上的第 1 引脚与第 22 引脚都有逻辑状态的 0 与 1 以及+12V 的输出能力。这些操作是靠晶体管 Q1 与 Q2 及外围的元件所构成的。

测试夹上的第 28 引脚为 EPROM 的电源供应引脚，它可以切换+5V 与+6V 的电压输出，这个操作是用 U6LM317 与两个可变电阻完成的，请回头看图 12-4 的电压切换控制线路图。烧录时我们要另外准备一个+15V 到+24v 的直流输入电源，直接由烧录板的电源端输入。我们所使用的 LM317 是一颗输出可调整的稳压 IC，其输出的公式可看图 12-6 上的说明，在我们的线路中 R2 是一个可调电阻，所以可以很方便地调出所要的电压来。在线路中我们特地加入一个控制点，当 2003 缓冲输出器的 VCC_SEL 是输出 HIGH 时，R10 与 VR1 不起作用，所以 VCC_SUPPLY 的电压是由 R7、R9 与 VR2 来决定，此时我们应该调整 VR2，使 VCC_SUPPLY 的电压保持在 6V 左右。当 VCC_SEL 输出为 0 时，VR1 的第 3 引脚如同接地一样，LM317 的输出电压改由 R7、R9、R10、VR1 与 VR2 来控制，这时我们再调整 VR1，以便在输出端得到+5V 的输出。烧录电压 VPP_SUPPLY 的调整及控制方式也是类似的，在烧录程序中，只要我们控制 2003 的输入点就可以让 EPROM 进入平常的读取模式或是烧录的模式。

线路中也安排了 4 个小开关、LED 显示灯及蜂鸣器，让我们在不接个人计算机的情况下，可以进行 EPROM 数据的读取/烧录/对比等等，蜂鸣器则在提醒操作者是否有错误发生，这些技巧在我们制作 AT89C51 烧录器时就已用得非常纯熟了。

许多万用烧录器为了应付许多种烧录电压的设定，都是先把+12V 用振荡器提升到+25V 左右，然后用数字到模拟转换器（DAC）再做具体的调整。由于我们设计这台 EPROM 烧录器的主要目的是对 EPROM 烧录的彻底理解，所以并没有将烧录器更复杂化的打算。如果您经常要烧录 EPROM，我们建议您另外买一台有两组直流输出（+5V 及+24V）的电源供应器，专供给烧录板与 FLAG51 使用。如果您偶尔才烧录一次 EPROM，那只要在烧录前，找来一台可调到+15V 以上的直流电源供应器，用接线引到烧录板上即可。

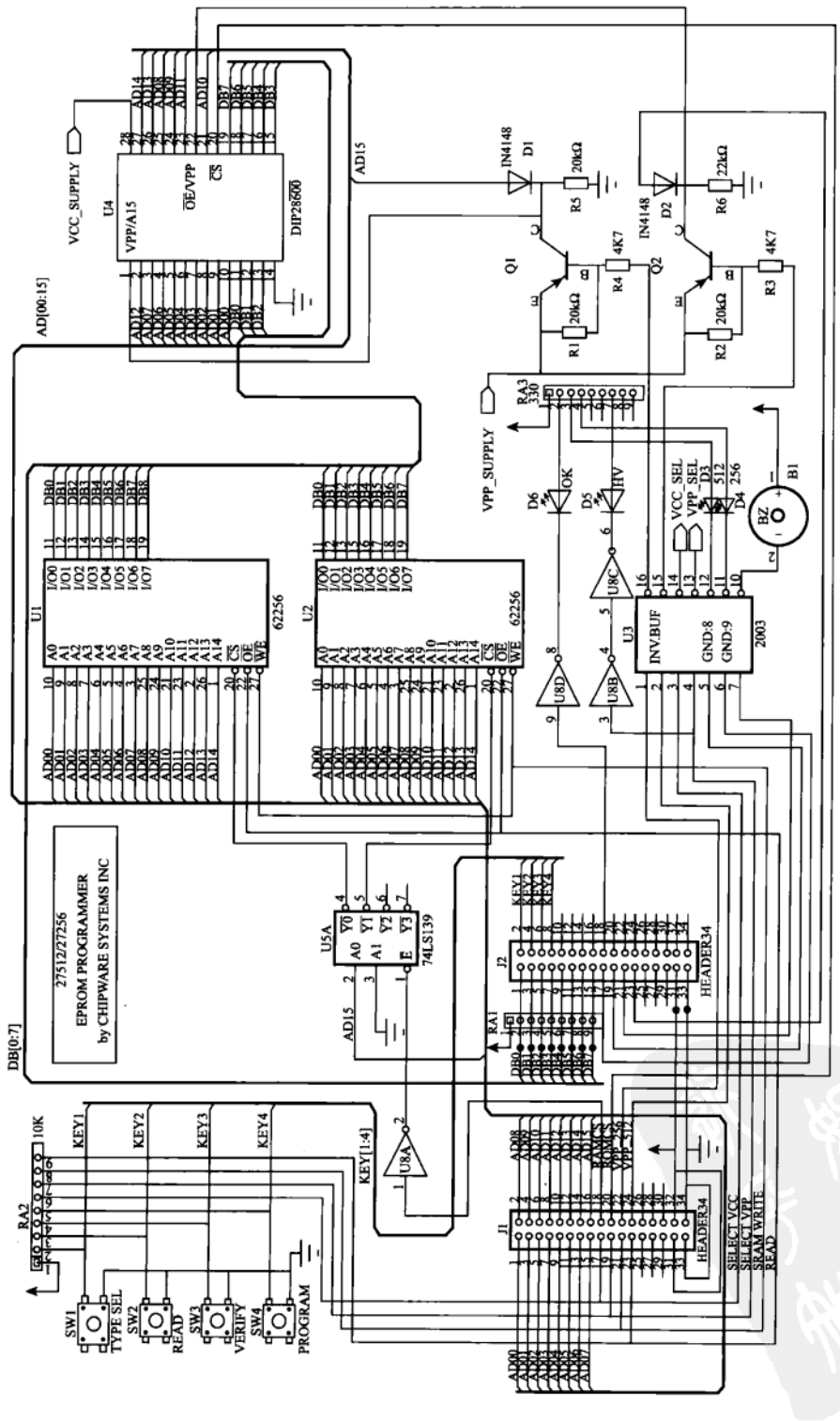


图 12-3　27512/27256 EPROM 烧录线路图

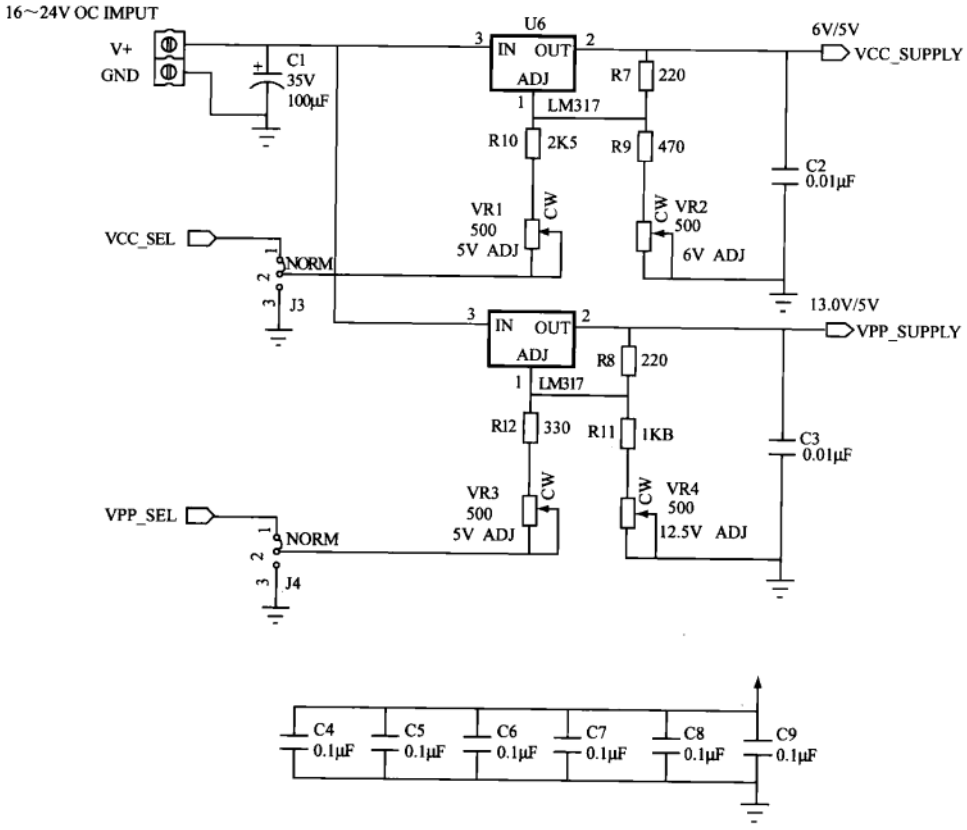


图 12-4　烧录时的电压控制电路

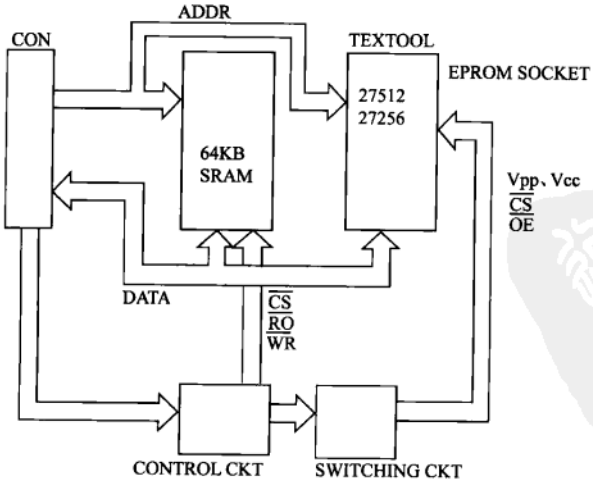


图 12-5　烧录器的控制框图

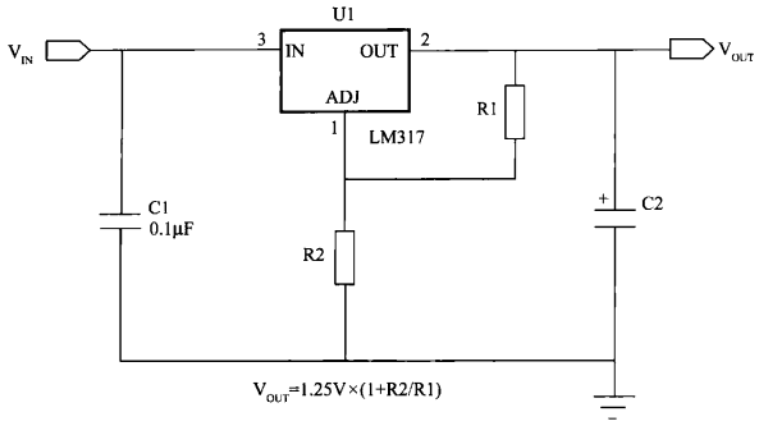


图 12-6　LM 317 的输出调整

12-3　知其然，再知其所以然

先前我们提过，每个 IC 制造厂对于自己的 EPROM 烧录资料绝对是毫不保留的，所以我们可以在没有任何阻碍的情况下，自行开发 EPROM 烧录器。以这次我们试做的 27512/27256 EPROM 烧录器为例，线路的焊接及检查花了两天的时间，烧录程序的规划与除错也只耗掉约五个工作日，而进行 EPROM 的实物烧录试验约是在第四天就开始了。所以开发一台 EPROM 烧录器并不如想象中那么难，真正的困难点却是在“如何在最短的时间内完成一个 EPROM 的烧录”，这句话看起来好像没什么障碍，可是在写烧录程序的主体时，却到处困难重重的。

如果您在杂志上曾经看到 EPROM 烧录器的广告，您会看到非常吸引人的一段话：烧录一颗 27256 EPROM 保证在 10s（或是更少的时间）内完成。如果我们用 100μs 的脉冲去烧录 27256 EPROM，光是实际的烧录时间就要 3.3s，烧录完若要再次让 EPROM 加深印象时，应该以第一次两倍的脉冲宽度再次进行烧录，这段时间也要 6.6s，光是这两个时间加起来就几乎是 10s 了。可是从 SRAM 区取一个字节的烧录值转到烧录插座上再加上一些程序上的判断，至少要花上 100μs（以 8051 为例）的时间，100μs 乘以 32768 等于 3.2s 左右，所以计算出来的烧录时间应该在 15s 以上。可是现有的 EPROM 烧录器号称能在 10s 内完成烧录，除了他们尽量采用更高速的 CPU，把存取时间缩短外，还有可能会把第二道 EPROM 加深印象的时间缩短（或是干脆拿掉），以便能在客户前演示时，用最短的时间完成烧录。也有可能把烧录最重要的烧录使能信号由厂方公布的 100μs 减到 50μs 或 30μs，这当然是可以大大地减少烧录时间，用这种方法存入的数据有可能漏失，只要 256K 位中有一个位由 1 变成 0，就有导致您系统完全停摆的可能。

许多要求高可靠性的仪器或设备（如航天飞机或核潜艇内的控制设备）中，若有用到 EPROM 的场合，一定不允许这种降低可靠性的烧录操作的。

我们认为：一个最佳的 EPROM 烧录器的烧录原则应该是类似 RAPID 的烧录模式，烧录完后还是要加上“印象加深”的 OVER-PROGRAMMING 处理，在数据对比验证（VERIFY）阶段，先用标准的+5V 电源加到 EPROM 的电源端上，进行第一次的验证，然后又将电压降到+4.5V 做第二次验证，最后把电压提高到+5.5V，做第三次的验证，如果三次都正确才算是

数据已确实进到 EPROM 了。这么做的 EPROM 烧录是最可靠性的，但是若以烧录 27256 为例，整个烧录过程恐怕要花上 40s 的时间。类似这些原理及产生的后果，如果烧录器厂商不明说（您如果没问我就不做说明）时，您会买 10s 就完成烧录的机器，或是 40s 才完成的慢速 EPROM 烧录机呢？在我们试做的 EPROM 烧录器里，我们就加上了一个烧录使能脉冲的选择键，可以让您每次改变 20μs 的脉冲宽度，实际体会其中的奥妙。

12-4 EPROM 烧录软件的功能

一个 EPROM 烧录程序，基本上应该能做 EPROM 的空白检查（BLANK CHECK）、读取（READ）、烧录（PROGRAM）和验证（VERIFY）四个操作，如果接上计算机后可做的事就更多了；接受二进制文件的数据下传到 64KB 的 SRAM 烧录缓冲区内（LOAD），显示缓冲区的内容（DISPLAY），填满一个区块为固定值（FILL），传送一个区块（MOVE），以及修改缓冲区的内容（EDIT）等等，除此之外还可以把 64KB 缓冲区内的数据上传到 PC 上（UPLOAD），还有选择 EPROM 的型号（27512/27256）及烧录法则的指定等等，大部分市售烧录器的功能也是如此。

由于这台烧录器是强调学习与实用并重，所以我们在验证的功能上稍做修改，当进行烧录数据与 64KB SRAM 对比时，若有错误且错误的地址小于 5 个时会自动把两组数据同时显示在 PC 的屏幕上，如果错误次数超过 5 次时，验证会自动放弃对比的操作。现有的 EPROM 烧录器只要一碰到错误就跳离，只知道哪个地址有错误而已，完全不晓得烧录的数据到底是怎么样？如果就只差一个位时，还可以再进行一次烧录，不必将 EPROM 放到紫外灯下清除，又多花了 30min 的时间。我们也考虑加入特别的功能，只要在 DOS 提示下，键入“EPROM COM2 27512 TEST.BIN”等字样，就可以把 TEST.BIN 二进制文件的数据通过 COM2 通信端口下载到 EPROM 烧录器中，并且立即执行烧录 27512 EPROM 的操作，如果一切都正确时，蜂鸣器会叫一长声代表烧录完成，如果叫三个短暂声时，表示烧录过程有问题。

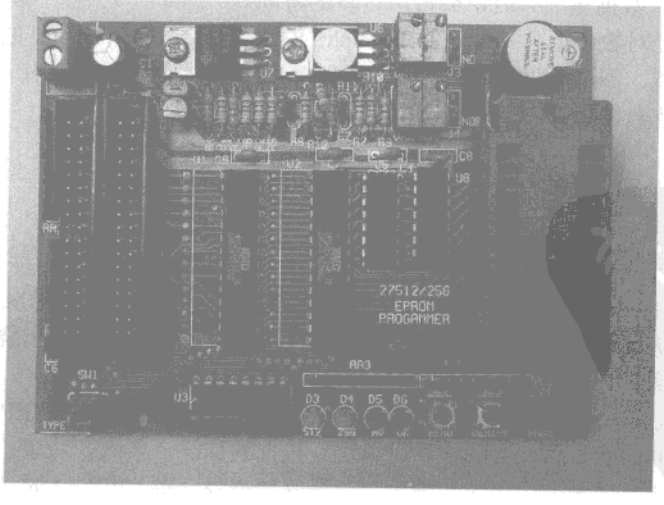


图 12-7 EPROM 烧录器的实物图

12-5 跟烧录时间赛跑

当我们的烧录器的试制品完成后，马上就进行 EPROM 的烧录测试，同时又用示波器观察每次烧录的时间，结果让我自己也吓一跳！竟然每个字节要 3ms 的时间，烧录完 27256 要花上接近两分钟，如果这样就推出的话包准被比了下去。于是又花了三天的时间，逐一计算每一行操作所花的时间，这时每个环节都要斤斤计较才行。最后才把每个字节的平均烧录时间缩到 350μs 左右，整个烧录时间也缩到 10s 以内，但是如果再加上烧录之后的验证时间，总共约要花 40s。如果要加快验证的速度，可以更换较快的 CPU，或者是用硬件做数据验证的操作，后者可能会使整个验证的时间减到几秒以内，在开发 EPROM 烧录程序后，我们得到一些宝贵的经验，在此一并提供给读者参考。

（1）烧录的使能时间已经与数据存取的速度相当，所以最好是用纯硬件的架构存取数据。

（2）如果烧录的数据是 FFH 时，这个值是 EPROM 清除后的空白值，可以不启动烧录使能信号，直接跳开这个地址，这可以节省 100μs 的时间，这些时间积少也有可能成多。

（3）烧录一个字节之后的立即验证（VERIFY ON PROGRAMMING），可以考虑取消，否则会使烧录时间更为加长。

（4）烧录前的空白检查如果能用硬件来做最好，假使所有的数据都通过 8051CPU 来转存，数据量一大时，会占用过多的时间。

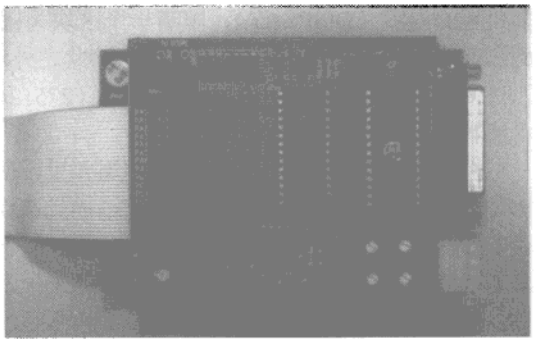


图 12-8 我们试制的 EPROM 烧录板，为了减少接线只有加上 32K 的 SRAM，另外两个稳压 IC 放在 PC 板的另一面

（5）我们曾经对数十个各种厂商的 EPROM 做各项烧录实验，100μs 的烧录使能信号时间是可以再做缩减，不过，每个厂商的 EPROM 所能接受的最短时间互有差异，但是 100μs 的宽度一定可以将数据烧入到 EPROM 中。

12-6 自己装的您有福了

如果您想同时拥有 EPROM 烧录器并且学得烧录 EPROM 的诸多特点，我们将提供您最完整的学习环境与数据。您可以只花半天的时间自己焊接完成一台 EPROM 烧录器，然后依照我们的详细说明进行基本的烧录电压调整，接着在 FLAG51 控制板换上烧录 EPROM 监督程序，就可以烧录 27256 或 27512 的 EPROM 了。

当与 PC 个人计算机不连接时，您可以用它来复制（COPY）或检查 EPROM，如果接上 PC 并执行 EPROM.EXE 程序后，那就可以进行数据的加载、修改或烧录，操作画面比一般的万用烧录器还要有亲和力。当您的程序告一段落要烧录一颗 EPROM 时，不需要花将几千元去买一台万用烧录器，而只用到其中十分之一不到的功能，也不需要多人合买一台 EPROM 烧录器。不过还要再次提醒您：这块 EPROM 烧录板必须与 FLAG51 控制板配合才能工作，

单独有 EPROM 烧录板是无法进行烧录的。如果您想独自焊接所有的烧录线路，我们绝对鼓励及支持您花下的苦心，所以只要您把装好的 PC 板照片寄来，我们就会免费地把烧录 EPROM 的所有相关数据及烧录程序 EPROM 寄给您。

EPROM 烧录器的 FAQ

问题一：烧录时，最后听到三声短促的哔声，而且显示“VERIFY FAIL”（验证失败）等字样，接下来怎么处理？

解答：请查看“VERIFY FAIL”之前的英文叙述，EPROM 控制程序对比验证时，会把错误的地址及数据送回到 PC 的屏幕上，但最多只送 5 个错误地址，并且程序会自动判断该地址是否可以再度烧录。如果可以的话，会在对比错误地址的最后加上“Re_programmable”的提示，反之则显示“Some bit（s）must be 1，but it's 0 now!!”，这代表有些 bit 应该是 1，但现在已经是 0，无法再变成 1，除非拿到紫外灯下将所有数据重新清除成 FFH。所以如果您看到的错误提示最后都是“Re_programmable”，请再执行一次烧录 EPROM 的操作，不需要烧录一有错误就把 EPROM 拿到紫外灯下清除。

问题二：“VERIFY FAIL”之后又烧录一次，似乎有部分数据烧录到 EPROM 中，但是仍有“VERIFY FAIL”的字样出现，而且这次出现错误的地址好像往后延了？

解答：碰到这类的问题通常是指定烧录脉冲太短或是 EPROM 为较早（1990 以前）的产品所引起的，所以请再做一次或一次以上的烧录操作即可过关，也可以按“+”键，增长烧录脉冲的时间，然后再度进行烧录。您也可以直接就按“2”键，指定用现有烧录脉冲宽度两倍的时间进行烧录。

问题三：是否每次 EPROM 烧录前，都要先检查 EPROM 内部是否为空白的码 FFH？

解答：如果您的 EPROM 已经经过紫外灯 30min 以上的照射时，理论上是可以直接烧录的。如果 FLAG51 上面的 CPU 是 8051，石英晶体是 11.0592 MHz 时，检查一个 27256 是否空白要花 16s 的时间，而烧录加上验证的时间要 32s，其实可以观察验证后的结果就能知道是否烧录正常了。

问题四：用最短的脉冲烧录 EPROM，会有什么影响或后果？

解答：如果您亲自做烧录实验时，您会发现不少厂商的 EPROM 用小于 100μs 的烧录脉冲都可以成功地烧录，有些甚至 20μs 的时间就可以烧录一个数据，不过“年纪较大”的 EPROM 还是需要较长的烧录时间。但是无论如何，当您烧录 EPROM 成功后，最好再执行一次“2”的脉冲加倍烧录（Over-programming），以便确认数据的安全性。如果您用最短的脉冲烧录，烧录的时间会最短，但验证失败的机率一定会较高，您也可以一直按“P”，直到所有的数据完全烧录到 EPROM 内为止。

问题五：有些 EPROM 烧录后验证都通过了，可是插到我的系统中却无法工作，这是什么原因呢？

解答：如果您的 EPROM 已经过两倍烧录脉冲的处理后，仍然无法让系统工作时，有可能是该EPROM的读取时间(Access time)太长，以致使CPU无法在限定的时间内取到EPROM内部的程序数据，造成程序无法顺利执行。解决的方法只有换一个 EPROM 试试看。大部分的 EPROM 正面上都会标示出其读取时间来，例如 27C256-15，这代表该 EPROM 的数据读取极限是 150ns。EPROM 的读取时间如果在 100～150ns 左右已可以算是相当快速的存储器元件了，较早期的 EPROM 其读取时间多为 200～300ns，烧录时的烧录脉冲较长，正常读取

时的速度也不可以过快。

问题六：何时才能判定 EPROM 已经损坏了？

解答：如果该 EPROM 已经经过紫外灯的数据清除，而且我们执行过 10 次以上的数据烧录，验证后 EPROM 内的数据仍是 FFH，有可能是内部控制烧录的电路已经损坏，此时的 EPROM 已经完全没有利用价值了。EPROM 通常是因为方向反插又加上电源后才损坏的。如果 EPROM 这次烧录不成功，我们习惯在该 EPROM 的底部用铅笔做一个记号，然后再拿去紫外灯下清洗，如果再次烧录不成功的记录次数超过 10 次时，就该将此 EPROM 丢入垃圾筒了。

问题七：我的程序长度只有 10KB 左右，可是烧录最后的验证时指出，比方说：7F00H 处的数据有一个位错误，这个 EPROM 还能用吗？

解答：如果您程序确实只有 10KB，所以 7F00H 处应该是系统完全没用到的，所以该 EPROM 应该还是可以使用的。但是，如果您的程序大到 32KB 左右时，这颗 EPROM 就可能无法使用了。

问题八：这一台 EPROM 烧录器的速度有可能再提高吗？

解答：提升速度的步骤有两个，第一个是将 CPU 换成 DALLAS 的 80C320，其他程序或线路完全不做调整，这样可使所有的读取/烧录/验证时间加快到原先的约三倍之多。第二个加速的做法是连 FLAG51 上的石英振荡晶体及程序都要修改。更改石英振荡器前，请先将磁盘中的 EPP80320.TSK 文件烧录到 27256（速度应该选择 120nS 以内）中，然后把 FLAG51 控制板上的 CPU 改成 DALLAS 的 80C320，并且更换系统的石英振荡晶体为 20MHz，再将新的 EPROM 插入 FLAG51 的 ROM 插座上，整个系统的执行速度约快了 5 到 6 倍左右。不论是烧录或验证的时间都缩短了许多。

问题九：这台 EPROM 烧录器在使用上有哪些地方要特别注意？

解答：请特别留意 EPROM 摆上与取出的时间，当 EPROM 烧录程序没事做时，我们特地安排 8051 做 27256/27512 TYPE 的 LED 闪烁显示，如果 LED 不是在闪烁的状态，一定是在处理烧录等事宜，而且一定会动到 EPROM 烧录板上的 64KB SRAM。由于整个 SRAM 的位置与数据线都是与烧录测试夹上的地址数据线并联，所以最好不要在这个时候摆上或取下 EPROM。

请记住以下两个原则：

原则一：HV 烧录高电压操作时，不要去搬动测试夹上的 EPROM。

原则二：代表 EPROM 型别的 LED 若没有闪烁时，也不要尝试去动在测试夹上 EPROM。

问题十：如果是 27256 却指定 27512 再烧录，结果会怎样？

解答：如果会有问题的话，应该是出在两者 Vpp 引脚位的不同上，若是以 27512 的方式烧录，Vpp 烧录电压是加在第 22 引脚上，该引脚对 27256 而言仅仅是输出使能引脚（OUTPUT ENABLE），应该只能有数字 0 与 1 的电平输入而已，可是此时却加上了超出标准的电平甚多的+13V Vpp 电压，有可能会永久损坏该 EPROM。反之，明明是 27512 的 EPROM，却指定用 27256 的引脚烧录时，也会使该 EPROM 受损。这也是我们加入两个 LED 做 EPROM 型显示的原因，而且又特地加入了闪烁的功能，提醒您烧录前再多看一眼。

问题十一：这台 EPROM 烧录器开机后内置是烧录 27256 的 EPROM，可以改成 27512 吗？因为我烧录 27512 的机会较多，我怕忘了更改 TYPE 因而损坏 EPROM。

解答：我们这台 EPROM 烧录器是很人性化的，本烧录器的二进制执行文件是放在

EPP8051.TSK 文件上，您可以自我复制供保存用。当烧录器一开机之后会去检查 FLAG51 控制板上 ROM 最后一个地址 7FFFH 的内容，如果是 00H 则默认的 EPROM 是 27256，若是其他值则切换成 27512。讲到这里您应该知道如何做修改了吧！

问题十二：这台 EPROM 烧录器的原始程序有机会公开吗？

解答：本 EPROM 烧录的控制程序是用 C 语言写成的，但关系烧录速度的部分则是用汇编语言完成的，整个控制程序超过 3000 行。请参阅本文最后的清单。

EPROM 烧录器的程序绝对是一个挑战性颇强的程序，因为它不仅要能够正确地将数据烧录到 EPROM 中，而且还要把执行时间大幅缩小。这对软件工程师来讲一定够震撼的。原来许多读取操作用 C 来就行了，可是卡在执行时间过长，只好将最重要的部分改用汇编语言重写，从烧录程序中，你可以看到很多软件的秘诀就在程序中。

EPROM 烧录程序请查看随书光盘的 CH12_EPROM 烧录程序彻底公开文件。

本章使用的软件

2500AD 8051 C Compiler and Assembler。

本章使用的硬件

（1）FLAG51 单片机控制板。
（2）EPROM 烧录板。
（3）EPROM 27256。
（4）EPROM 27512。
（5）Dallas 80C320：提升 FLAG51 的速度到原先的四倍。

相关资料网站

可经由下列公司、网站取得更进一步的信息：
http://www.chipware.com.tw：查询 8051 单片机控制板资料。
http://www.rs232.com.tw：查询 EPROM 等存储器资料。
http://www.atmel.com.查询 Atmel 89C 系列资料。
http://www.dalsemi.com：查询 DS80C320 CPU 的资料。
http://www.iar.com：查询 C 编译器的相关资料。

13

电动机车里程表的外观，做得出来不难，难得的是卖得出去，这才是重点所在。

第 13 章　EPROM 烧录器的组装步骤

现在谈 EPROM 烧录器是不是有点落后呢？不会的，因为 EPROM 的相关知识套用到现今的 Flash 或 NVRAM 都还是可行的。EPROM 提供给程序开发人员一个相当低廉且稳定存放程序的空间，看完本章后您将会发觉 EPROM 烧录器的背后，还是有许多可探讨的空间。

EPROM 烧录器对所有的单片机学习者而言，是一项绝对需要的工具。不过，一般电子公司的开发部门在购买烧录器时，都会选择购买万用型的烧录器，因为只要是可编程的 IC 都可以烧录，这类的万用烧录器一定要与个人计算机相连接，而其价格约在几千元上下，除非您是专业的开发人员，否则花下这些费用却只烧录 EPROM，就显得有一点点不划算。如果您还是学生的话当然也比较无法接受这个价格，这也是我们要开发 EPROM 专用 EPROM 烧录器的原因。

万用烧录器连接个人计算机好处多多，最大的优点是直接利用了计算机的各项资源，所以画面可以做得很漂亮，而且也不会有存储器容量不足的情形发生，可是如果脱离了计算机，这里所说的优点又全部变成绊脚石了。如果是电子工厂的生产出货单位时，烧录 EPROM 能够不用计算机就最好不用计算机，因为我们无法掌握操作者的程度，所以可以独立操作的 EPROM 烧录器还是有其存在的空间，而我们设计的 EPROM 烧录器刚好介于两者之间，可以跟计算机联机，也可以独立操作进行 EPROM 的复制及验证，其与万用烧录器内部最大的差别在于我们的 EPROM 烧录器内部有 8051 CPU，可以对所有的输入/输出点进行监控，而这刚好也是 8051 单片机彻底运用的最佳范例。

根据我们的统计，市面上至少有十种以上的 EPROM 烧录器，有的强调烧录速度超快，有的强调全中文的说明，也有特别指出其扩展功能超强，但是都没有可以让使用者自行 DIY 的产品，而我们正好是开路先锋了。如果您是工作上的需要，必须烧录多种 EPROM 时，我们还是建议您买市面上的万用烧录器，因为这类的烧录技术已经非常纯熟了，不论您是买哪一个厂商的万用烧录器都可以烧录市面上九成以上的可编程 IC，如果把买到的价格除以可烧录的元件数量，您会得到一个相当低的值。不过，如果您是一个喜欢自己动手做的人，而且也用过 FLAG51 单片机控制板时，我们强烈建议您试做 EPROM 烧录板，除了满足自己动手做的乐趣外，您还可以从中学习到以下非常独到的技巧，这些对 EPROM 烧录器制造厂而言，绝对是属于机密的资料：

（1）如何调整 EPROM 的烧录电压。
（2）如何正确使用 EPROM 烧录器。
（3）如何使用 EPROM。
（4）如何加快烧录速度。
（5）如何写 EPROM 烧录程序。

接下来我们将先用照片引导您 DIY 焊接时的几个重点，然后说明每个 DIY 的步骤，最后才加入了使用 EPROM 的几个常发生的问题及解答。您将发现 EPROM 的烧录绝对不是只

按几个按键而已，如果您对 EPROM 烧录器越清楚，就能对您写的程序越有帮助。

13-1 EPROM 烧录前您还需要哪些设备

除了 EPROM 烧录板外，您还需要一台有 15～24V 左右直流输出的电源供应器，以便在烧录时产生+6V 与+13V 的烧录电压值给 EPROM。最方便的方法就是买一台同时有+5V 与+24V 直流输出的开关电源供应器，其合理的价格应该在千元以内。由于 EPROM 烧录器的耗电量并不大，您也可以用变压器及桥式整流器，自己装一个简易的直流电源供应器，专供 EPROM 烧录器使用。另外如果您许久才烧录一次 EPROM，可以找来一台实验用的直流电源供应器，在烧录时才把电源加上去。不论您是使用何种外加的电源，共有两条电源接线要加到 EPROM 烧录板上，一是较高电压值的输出（我们建议使用橙色线），另一条则是地线（建议使用黑色线），如果外加的电源线的地线没有接时，会使得高电压的输入没有参考的电平，而无法烧录 EPROM 并且也无法读取烧录测试夹上 EPROM 的数据。

13-2 EPROM 烧录器的 DIY 步骤

当您自己开始制作 EPROM 烧录板前，请先清点所有的电子元器件是否齐全，并且注意以下几个元件是有方向性的：所有的集成电路 IC、34P 的简易牛脚插座、蜂鸣器、电解电容、二极管、LED 发光二极管、电阻（共有九根引脚）、晶体管等等，如果焊错方向时会使烧录板无法工作，请特别注意！

EPROM 烧录板的元件面上已经做了非常仔细的标示，所以焊接时请依上面的标示方向将元件插入，然后在背面焊接即可。您焊接的顺序应该是由元件高度最低的电阻电容开始，最后是简易牛脚以及四个 20 转的精密可变电阻。如果该零件的引脚数超过 3 根脚时，您可以先焊接零件两端的引脚，然后翻到元件面，检查元件的位置是否有歪斜。由于此时只焊两点要再做调整是很方便的，位置确定正确后再焊其他引脚，这样就能保证所有的元件都焊得正正直直的。在 EPROM 烧录板上共有三个小 IC（74LS139、74LS04 与 ULN2003），请直接焊在 PC 板上，至于两个 32KB SRAM 的位置则先焊上 IC 座，待系统调整完后才把 IC 插上。

在烧录的插座上，我们提供两种方法给您选择：

第 1 种方式：28P 的烧录测试夹直接焊在 EPROM 烧录板上，您可以一直使用本 EPROM 烧录器，直到该测试夹寿终正寝为止，然后再用吸锡枪把该测试夹取下，如果您不常使用 EPROM 烧录器，我们建议您采用这种方法。

第 2 种方式：如果您一天烧录 EPROM 的数量超过几十个时，可以用这种方式以便随时可以更换测试夹，28P 的烧录位置先焊上一个高品质的圆孔 IC 座，再插上一个 28P 的普通 IC 座，最后才插上 28P 的测试夹（插入时要有一点技巧才行），这种方法可以随时更换 28P 测试夹，但是烧录时若放 EPROM 的操作过于粗鲁时，测试夹很可能有松脱之虞。在 DIY 的当中，您可以依实际的需要做焊接的依据。

元件焊完后请再检查所有的焊接点是否有假焊或虚焊的情形，然后翻到 EPROM 烧录板的元件面上，再一次详细地检查所有元件的方向是否正确，若发现有错误时，应该立即用吸锡枪将该元件取下，重新焊回正确的方向与位置。在焊接时比较容易出错的零件有：IC 方向颠倒、LED 方向错误与 34P 简易牛脚方向的错误，其中以简易牛脚焊接错误最难处理，如果您把 34P

的简易牛脚插座搞颠倒了，接下来接 FLAG51 单片机控制板的排线接头时，就一定接不上了。

13-3 EPROM 烧录器调整步骤

如果您焊接完成而且检查也统统正确时，就可以开始进行通电测试了，此时请尽量依照我们建议的调整步骤来进行整个 EPROM 烧录板的调整与测试。

调整步骤 1：用两条 34P 的短数据线将 EPROM 烧录板与 FLAG51 控制板连在一起，其中 EPROM 烧录板的 J1 连接 FLAG51 的 CN1，J2 插座则接 FLAG51 的 CN2，然后再用铜柱将 EPROM 烧录板直接架在 FLAG51 单片机控制板上。

调整步骤 2：接着接上+5V 电源与额外的烧录电压电源，后者的电压最少要调整到+16V 以上，否则在 EPROM 烧录板上将无法调出我们所需要的+13V 烧录电压来，这是因为 LM317 稳压调整 IC 的输出入电压差必须有 3V 以上才能进行电压调整以及稳压。

调整步骤 3：第一次进行调整时，请不要把两个 32K×8 的 SRAM 插上，然后将标示有“EP”字样的 EPROM 插到 FLAG51 的 EPROM 座上，此时 FLAG51 将变成 EPROM 专用的烧录控制器，然后先在个人计算机上执行 EPROM.EXE 程序，该程序会先询问您是由 COM1 或 COM2 与 FLAG51 连接，您只要输入 1 或 2 后，画面就会显示出准备与 EPROM 烧录器连接的信息，然后再打开 FLAG51 的电源，FLAG51 控制板上的八个 LED 会逐一亮一次，如果连接正常时，应该可以在 PC 的屏幕上看到一些旗威科技公司版权的提示，最后出现了“EPROM_27256[？？？？]100μs width>”的字样，这就代表 EPROM 烧录器与 PC 间的连接已成功了。

```
EPROM PROGRAMMER V3.0 all rights reserved
Program written by Chipware Systems Inc. '96/02/05
Any questions about EPROM, please mail to:
Mr. Lin S.M.  P.O.BOX 1859,KAOHSIUNG 800,TAIWAN R.O.C
              or E-mail:chipware@seed.net.tw
Please read[README]file in disk before you program EPROM

System ROM(0000H-7FFFH)checksum OK!
System RAM(8000-9FFFFH)R/W function test-
SRAM ADDR 8000H...PASS
SRAM ADDR 9000H...PASS

Extend RAM BUFFER (64k) R/W function test-PASS
EPROM_27256[????]100μs Width>
```

图 13-1 旗威科技公司版权的提示

调整步骤 4：在 EPROM 的控制程序中已经附有完整的测试程序，所以连接成功后，请将 EPROM 烧录器的电源关闭，然后按住烧录板上的<TYPE>键，再次打开电源，请保持至少 5s 以上的时间，这个操作会使控制程序进入烧录电压调整的模式。这时您也可以看到屏幕上显示出与原先完全不同的信息。

调整步骤 5：此时 PC 的屏幕上会显示出“请调整 VR2 可变电阻，使 VCC_SUPPLY 的电压为+6.0V，调整 VR4，使 VPP_SUPPLY 的电压成为+13.0V”的英文叙述，这时请拿出小号的“一”字形螺丝刀开始调整 VR2，并且用万用电表监视 U6 LM317 中间脚与地 GND 间

的电压，正确的测量方法是先用鳄鱼夹把电表的负端与电源供应器的 GND 地点连在一起，用一只手拿电表的正端去接触 LM317 的中间脚，然后用另一只手拿螺丝刀调整 VR。当您转动 VR2 时应该可以看到电压值已经在改变了，请一直调整 VR2 直到 LM317 的输出值（第 2 引脚）为 6.0V 为止。在此烧录高电压调整的同时，代表高电压的 HV LED 会一直亮着，其他 LED 则依序点亮，最后蜂鸣器也会 Beep 叫一声。正式烧录时，控制程序会把 EPROM 的第 28 电源引脚由+5V 提高到+6V，VR2 即在决定第 28 引脚在烧录时的电压。

调整步骤 6：如果 HV 灯还一直亮着，请再调整 VR4，以便让 U7 LM317 的输出电压达到+13.0V，电压监视点为 U7 LM317 的中间脚，这个烧录电压信号会经过晶体管切换，传到 28P 测试夹的第 1 引脚或第 22 引脚。第一次调整时往往会花较多的时间，控制程序会在蜂鸣器叫 30 次后自动切换到 EPROM 正常读取数据的模式，此时 HV LED 会灭掉，而且画面会显示出“请调整 VR1 可变电阻，使 VCC_SUPPLY 的电压为+5.0V，调整 VR3，以便使 VPP_SUPPLY 的电压成为+5.0V”的英文叙述，如下所示。

```
Vcc=6.0V,Vpp=13.0V
Please check programming Voltage!!
Adjust VR2 for Vcc=6.0V
Adjust VR4 for Vpp=13.0V
```

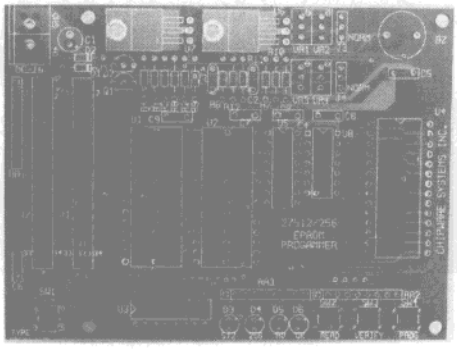


图 13-2　EPROM 烧录器的空 PC 板照片

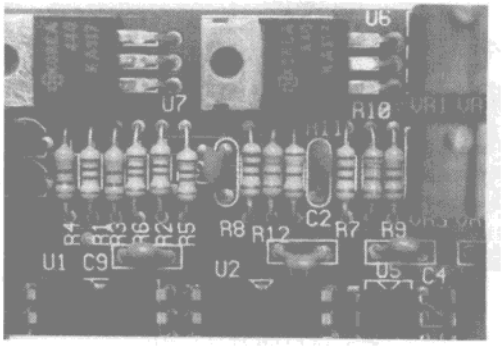


图 13-3　焊接后高度最矮的电阻与电容应该先进行焊接

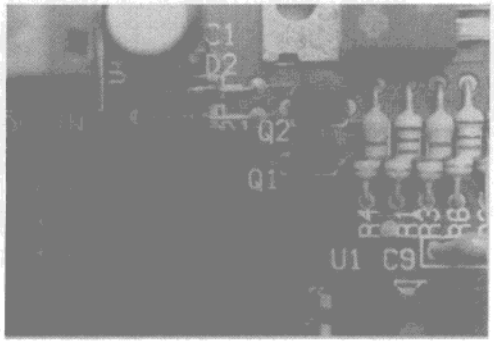


图 13-4　晶体管（2SC1015）与二极管按照 PC 板上的标示焊接，就不会出错

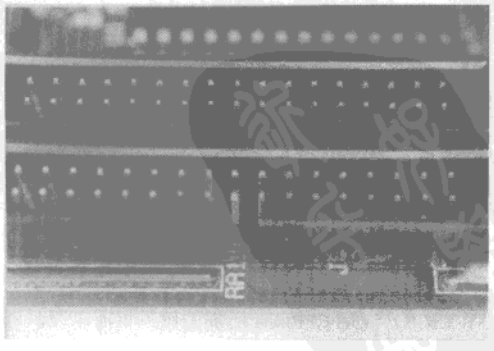


图 13-5　9P 的排阻与 34P 的简易引脚请特别注意其方向

图 13-6　LED（共有三个颜色）与按键开关应先焊一点，调整好位置后再焊其他点

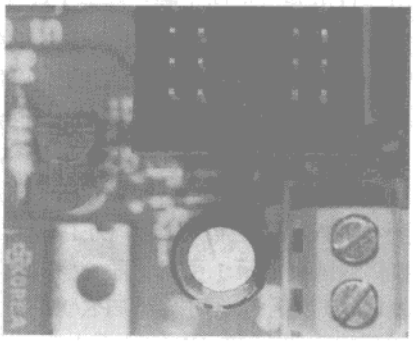


图 13-7　电源滤波电容上方会打上类似“奔驰”标志的 MARK，这是让电容如果故障爆炸时，首先由此处爆开

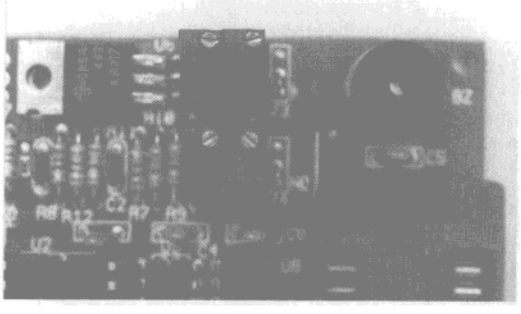


图 13-8　调整板上的精密 20 圈可调电阻就可以调整 EPROM 烧录的电压

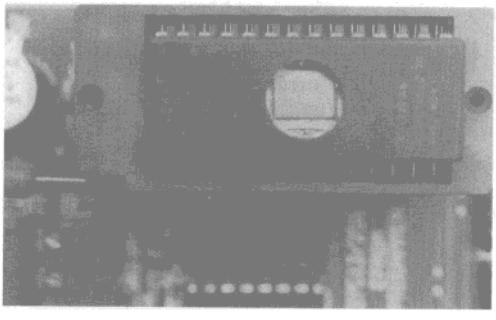


图 13-9　我们使用的 28P 测试夹不论其实质感或使用上的触感都比 3M 的测试夹好

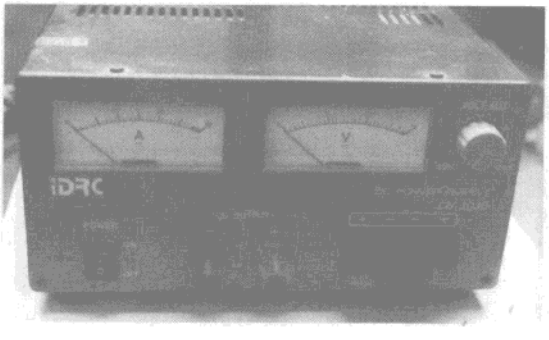


图 13-10　烧录 EPROM 所需的额外直流电源，可用直流电源供应器（下）或是开关电源（上），不过先决条件是输出电压要有+16V 以上

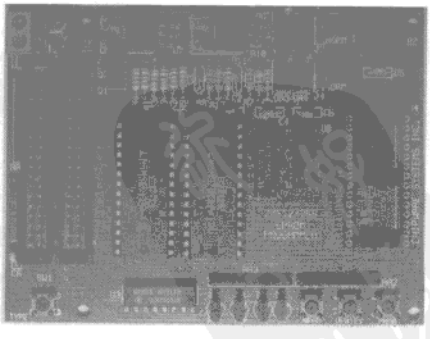


图 13-11　EPROM 烧录器的 DIY 焊接完后的照片

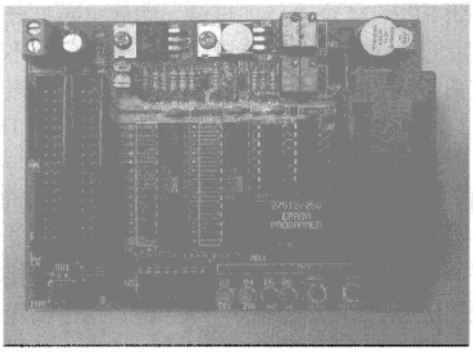


图 13-12　EPROM 烧录板上的 SRAM 可使用两种宽窄边的 IC，如果是窄边时，SRAM 应该靠右焊接

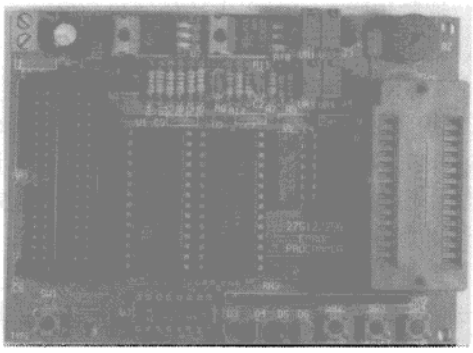


图 13-13　EPROM 烧录板烧录完成图

调整步骤 7：当 HV LED 灯灭掉时，烧录电压要降到+5V，EPROM 的电源供应电压也恢复为+5V，此时请调整 VR1，以便让 U6 LM317 的输出引脚刚好是+5V，然后再调整 VR3，让 U7 LM317 的输出引脚也刚好是+5V，最后的电压误差值只要在 0.1V 以内都是可以接受的。如果您对 VR1 调了再调，还是无法让电压降到刚好+5.0V 时，可能只差不到 0.1V，这是因为 R10 经过计算后的最佳值为 2.5kΩ 左右，但是市面上找不到 2.5kΩ 的电阻，而我们只好用 2.8kΩ 替代。如果您一直在意这一点点误差（对系统完全没有影响）时，请在 R10（2.8kΩ）的 PC 板背后并连一个 20kΩ 左右的电阻即可将 EPROM 的电源电压调到刚好+5V。在 DIY 的元件当中，我们会多一个 20kΩ 的电阻，就是让您做这方面的调整用的。

```
Vcc=5V,Vpp=5V alse
Please check Reading & Verifying Voltage!!
Adjust VR1 for Vcc=5.0V
Adjust VR3 for Vpp=5.0V
```

调整步骤 8：请再用电表量测 SRAM 第 28 引脚电源端的电压，这个电压值正是系统的+5V 供应电压值，假设量到的值是+5.1V 时，请再调整 VR1 与 VR3，以便让 U5 与 U6 的输出值也是在+5.1V 左右。

调整步骤 9：在调整 VR 的当中，LED 会逐一点亮，同时蜂鸣器也会叫，如果这些有方向性的元件焊错方向时是不会工作的，所以请检查并确认所有 LED 与蜂鸣器都是正常的。

调整步骤 10：分别按下<READ>、<VERIFY>与<PROGRAM>键，此时应该可以看到对应的信息出现在屏幕上，如果您按下<TYPE>键时，控制程序会切换到下一个调整模式，例如：现在是高电压的烧录模式时，会切换成数据读取模式，若在数据读取模式时，则切换成高电压的烧录模式，此时只要观看 HV 灯就知道现在所处的状态了。

```
Vcc=5V,Vpp=5V alse
Please check Reading & Verifying Voltage!!
Adjust VR1 for Vcc=5.0V
Adjust VR3 for Vpp=5.0V
<READ>button hit
<VERIFY>button hit
<PROGRAM>button hit
```

调整步骤 11：如果上述的调整都好了，请关闭 EPROM 烧录器的电源，然后重新打开电源，这可以使 EPROM 控制程序进入正常准备烧录的模式。进入这个模式后，测试夹上不放任何 EPROM，请先在键盘上敲一个“B”键做空白检查，烧录程序做空白检查时，每检查完 4KB 就会回应一个“.”字符。如果您事先是选择 27256 时，显示完八个“.”之后就响应“This chip is blank”的字样，如果不是这样的话，请检查测试夹背面的线路是否有焊接不良或与其他线路短路的情形。最后放上一个已经清洗完毕的 EPROM，再按下“B”再做一次空白检查，应该也是响应此 IC 是空白的才对。

调整步骤 12：请准备示波器或逻辑笔并接妥电源及测试用的接地点，首先把测试棒接触测试夹的地址引脚上，先由 EPROM 最低的地址线 A0 开始，然后按下“R”键，这个命令会使控制程序读取测试夹上 EPROM 的内容，然后转存到烧录板上的 64KB SRAM 内。如果您选择 32KB 的 EPROM 时，A0～A14 都会有所变化，其中以 A0 的变化速度最快，A14 则只变化一次，所以我们可以在示波器的屏幕或是逻辑笔上的状态指示灯上看到这些变化。若是选择 64KB 的 27512 时，测试夹上的第 1 引脚 A15 也会有所变化。这里所做的观察唯一所要确定的是：所有的地址线都能做数字 0 与 1 的变化，当然也不可以与其他非地址线有短路的情形。

调整步骤 13：接下来要进行较紧张的测试了，请先移开测试夹上的 EPROM，并把万用电表（最好是数字式的电表）切到 DCV 直流电压挡，将负测试点接测试夹的第 14 引脚，而正测试点接第 28 引脚。按下“P”烧录开始键，这时烧录控制程序会把 Vcc 电源引脚上拉到+6.0V，较正常的电源电压高出 1V，所以开始烧录时，您可以看到电压确实已提升上去了。如果您量到的值不是+6.0V 时，请跳到步骤 5 重新进行电压的调整。烧录过程结束后，会进入数据对比的状态，此时第 28 引脚又会回到正常的+5V 电平，此点也请确认一下。由于测试夹上并未夹上 EPROM，所以验证一定是过不了关的。

调整步骤 14：按下“T”键把 EPROM 类型切换成 27256，并把正测试棒移到第 1 脚上，负测试棒不动，然后再按下“P”键，您也可以看到正在烧录时，第 1 脚已经上拉到+13V 左右的电压。当进入 VERIFY 阶段时，第 1 引脚又回到 0V 左右的电压。再按下“T”键把 EPROM 类型改成 27512，监视点改在第 22 引脚上，这根引脚正常情况下是当成输出使能引脚（OUTPUT ENABLE），平常只要在逻辑 1 即可，亦即只要超过+2.0V 以上就可以接受。当我们按下“P”键时，第 22 引脚变成 27512 的 Vpp 烧录电压输入端，所以电压会上拉到+13V 左右。当烧录完毕后第 22 引脚会再度回到逻辑 1 的状态。如果电压无法切换时，请先检查控制电压切换的晶体管 2SA1015 是否有问题。

调整步骤 15：接下来测试烧录时最重要的烧录使能脉冲的宽度（Pulse width），这个信号一定要用示波器来观察，请将测试点放在第 20 引脚的 CHIP SELECT 上，该引脚平常是数据读取的使能引脚，烧录时则是烧录脉冲的输入引脚，大部分 EPROM 制造厂商都会在 IC 的规格上注明：只要加上 Vpp 电压，然后在 CS 引脚上产生一个大约 100μs 的负向脉冲，就可以将一个字节的数据烧录进去。所以请按下“P”键让控制程序开始烧录，这时我们就可以看到这个重要的负向脉冲信号了。如果您用的是一般的示波器，请将触发时基定在 100μs 左右，烧录开始时就可以看到此烧录脉冲信号了。您也可以连续按五次“+”键，让烧录脉冲加长到 200μs，然后再按下“P”键，此时就可以看到烧录脉冲已经变成原来的两倍宽度了。

调整步骤 16：烧录电压正确了，测试夹上的各个输出也正确之后，请关闭电源等待数秒，然后装上两个 32KB 的 SRAM，再次打开 EPROM 烧录器的电源，这次不按任何键，让 EPROM

直接进入正常的烧录模式。当所有的状态都就绪时，可以在 PC 端看到“EPROM 27256 [？？？？] 100μs width>”等字样，中括号内的问号代表系统尚未将内部 SRAM 的校验和 CHECKSUM 算出，此时您可以按下“？”或“/”查询所有的指令。

调整步骤 17：看完 EPROM 烧录器的所有指令后，请按下“Z”键，这会使烧录程序彻底地检查 64KB SRAM 的读写功能是否正常，并且每检查完 4KB 时就让蜂鸣器哔一声，如果正在测试的 4KB SRAM 内读写功能都正常时，会显示出“PASS”，否则改显示失败“FAIL”等信息。这个测试操作会一直重复地执行，直到我们在 PC 的键盘上按下任一个键后才暂停。

调整步骤 18：此时请按下“C”键，强迫将 64KB SRAM 的所有值清除成 00H，经过一段时间后，SRAM 内部都清除完毕，此时如果按下“*”计算 SRAM 内的 CHECKSUM（校验和）值时，系统经过计算后的校验和也应该变成 0000H 才对。如果到此都顺利的话，您已经接近成功了，这时只剩下 EPROM 实际烧录操作的确认而已。

```
EPROM_27256[0000]100μs width>P
Programming 27256
......
Verify
......
EPROM chip programmed successfully
EPROM_27256[0000] 100uS width>P
Programming 27256
......
Verify
addr [BUF] [ROM]
0000H 00 FF     <=Re-programmable
0004H A3 FF     <=Re-programmable
0007H 00 FF     <=Re-programmable
0008H FE FF     <=Re-programmable
000AH 77 FF     <=Re-programmable
More than 5 errors..
Failed in programming
Something wrong in programming this EPROM
Please check:
1. Is EPROM on the TESTOOL socket?
2. Did you add extra power supply for Vpp?
3. Maybe this chip is not BLANK or BAD before programming.
EPROM_27256[0000] 100μs width>
```

调整步骤 19：请拿出一个空白的 27256 EPROM，依照正确的方向放到测试夹上，按下拉杆将 EPROM 固定，然后查看代表 27256 的 LED 灯是否在闪烁，不是的话再按一下“T”键，将 EPROM 型号改成 27256。接着按下“P”键或是 EPROM 烧录板上的<PROG>键，如果 FLAG51 控制板上的石英晶体是 11.0592MHz，CPU 是标准的 8051 时，以 100μs 的脉冲烧录时，大约要 10s 就可以完成烧录，另外花 22s 做 32KB 数据的对比与验证，这也就是说只要半分钟多一点的时间，我们就可以完成一颗 27256 EPROM 的烧录。而烧录 27512 则要 70s 左右的时间。

13-4 EPROM 烧录器的使用

本 EPROM 烧录器可以独立进行数据的读入/烧录/对比等操作，也可以通过串行端口与个人计算机连接，前者适合工厂大量生产用，后者则非常适合在一般学习及实验烧录数据用。当 EPROM 烧录器的电源刚打开，系统完成内部的基本检查后，会让蜂鸣器哔一声，然后指定烧录的 EPROM 为 27256，此时代表 27256 的 LED 会一直闪烁，如果您按下<TYPE>键时，会把烧录的EPROM改为27512，这时代表27512的LED灯会开始闪烁。只要您看到代表27256或 27512 的灯在闪烁时，您都可以把待烧录的 EPROM 取下或摆上，请尽量避免在其他的场合更换测试夹上的 EPROM，以免影响烧录程序的操作。如果 EPROM 烧录板上的 HV（烧录电压）启动 ON 时，绝对不可以松开测试夹上的拉杆，否则有可能造成 EPROM 的永久损坏，此点操作时请务必记在心中。

13-5 EPROM 烧录板上各个功能键的说明

按下<TYPE>功能键，切换 EPROM 烧录的种类（27256/27512）。

按下<READ>功能键，读入一个 32KB 或 64KB EPROM 的数据到 EPROM 烧录板的 SRAM 当中。

按下<VERI>功能键，对比烧录板上 SRAM 与 EPROM 的数据是否相符，对比的长度按<TYPE>的设定。

按下<PROG>功能键，开始进行 27256 或 27512 EPROM 的烧录，烧录完后会自动进行数据的对比。

13-6 个人计算机联机时可使用的命令

？或 /：查询所有的联机命令。

T　：EPROM 27256/27512 的选定。

R　：读入一个 32KB 或 64KB EPROM 的数据到 EPROM 烧录板的 SRAM 当中。

V　：验证 EPROM 内部的数据是否与烧录器上 SRAM 内的数据完全相同。

本章使用的软件

（1）2500AD 8051 C Compiler and Assembler。

（2）IAR 8051 C Compiler。

本章使用的硬件

（1）FLAG51 单片机控制板。

（2）EPROM 烧录板。

（3）EPROM 27256（32K×8）。

（4）EPROM 27512（64K×8）。
（5）SRAM 62256（32K×8）/6264（8K×8）。
（6）Dallas 80C320：提升 FLAG51 的速度到原先的四倍。
（7）“河洛”万用烧录器。
（8）“玄积”EPROM 烧录器。
（9）3MIC 测试夹。

相关资料网站

可经由下列公司、网站取得更进一步的信息：
http://www.chipware.com.tw：查询 8051 单片机控制板资料。
http://www.rs232.com.tw.查询 EPROM 等内存资料。
http://www.atmel.com：查询 Atmel 89C 系列资料。
http://www.dallas.com：查询 DS80C320 CPU 的资料。
http://www.iar.com：查询 C 编译器的相关资料。

14

2500 AD 公司的 8051 Assembler 与 C 编译器都是相当专业的工具软件。

第 14 章　华邦 E²PROM W27E512 烧录器

E²PROM 和 EPROM 都是储存程序的利器，但是性能还是有所差异的。E²PROM 可在非常短的时间内清除，但烧录的时序却和 EPROM 雷同。本章将说明如何将原先 EPROM 烧录器的线路改成可烧录华邦 E²PROM 的线路。

在单片机控制的领域当中，储存程序的 EPROM 占了相当重要的地位，而 EPROM 的容量由早期的 2708 1KB 或 2716 2KB（1Byte=8bits），发展到最近几年的 64KB，甚至到高达 16MB 的超大存储空间。这些超大型存储器的用途大都用在图形处理或复杂的控制系统中，我们可以在程序或数据确定后，用 EPROM 烧录器或万用烧录器将程序代码或图形文件烧录到 EPROM 内部。不过，在 EPROM 使用上最大的困扰是内部数据清除的不易，这是因为 EPROM 可以在几十秒的期间将数据存入，而且保证十年间不会有任何数据的变化，可是在数据清除时却要在紫外灯下照射 20min 以上，才能全部将数据清除。现今分秒必争的时代对这个缺点当然会有所改进的。

14-1　EPROM 与 E²PROM 的差异

在 EPROM 成功地运用在市场上的背后，还有一款存储元器件在后面急起直追，这个存储元器件正是 E²PROM，E²PROM 比 EPROM 还多一个英文 E，E²PROM 当中第一个 E 是代表电气可擦除（Electrically Erasable）的意思，这也就是说 E²PROM 已经不需要紫外灯的照射，而改用加入较高的直流电压，就可以清除内部的数据。至于 E²PROM 这个英文字该如何念呢？这方面有两种说法：有人念成“DOUBLE-E-P-ROM”，听起来满顺耳的，也有人念成“E-TWO-P-ROM”，这种说法也是可以的。由此 E²PROM 的制程先天上就比 EPROM 还要复杂一些，而且产量远较 EPROM 稀少，导致早期的容量与价格根本比不上 EPROM，当然引不起数字设计工程师关爱的眼神，可是这个情况最近几年下来已经有所转变了！

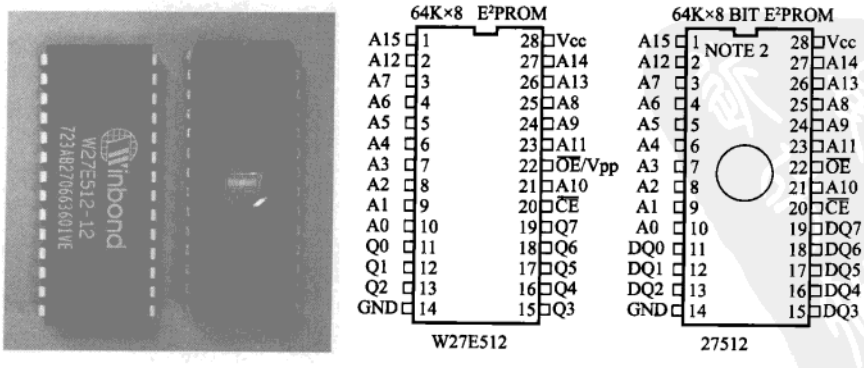


图 14-1　W27E512 与 EPROM 27512 的实物与引脚比较图

14-2 E²PROM 烧录线路的探讨与修正

旗威科技在数年前曾经开发出一套 EPROM 27256/27512 的烧录板，它是以 8051 单片机为控制主体，烧录市面上最常见的 32KB 与 64KB EPROM，烧录一颗 27256 只要十几秒钟就完成了。这台 EPROM 烧录器可以单机操作，也可以与个人计算机相互连接，进行烧录数据的下载及上传。在一个偶然的机会下，我们取得了华邦电子的 E²PROM W27E512 的烧录数据，经过仔细的研究与对照后，觉得可以把原先烧录 EPROM 的线路做稍微的修改，就可以烧录 W27E512 了。表 14-1 是 64KB EPROM 烧录时，几个控制引脚的电压状态表。而表 14-2 则是 W27E512 的控制引脚图表，看起来 W27E512 比 27512 EPROM 要复杂许多了，而且烧录电压除了+12V 之外，还多了一组+14V 的电气清除电压，比我们想象中的还难！W27E512 烧录时，其 Vpp 需要+12V 电压，在清除模式时，Vpp 的电压还要提高到+14V 上下，更难的是地址线 A9 在清除模式下还要能输出+14V 的电压。图 14-2 是原先我们 EPROM 烧录器的线路，幸好电路中有两组电压控制电路，要不然就无法修正了。一般的万用烧录器其每根烧录引脚都可以做任意电压值的输出，所以只要修正其引脚的定义就可以更改输出电平，但是这种做法会使线路变成非常复杂，而且制作费用相对提高许多。经过数天的修正及推敲后，我们将硬件线路修改成图 14-3 烧录电路及图 14-4 电压控制电路的模样。其中最重要的修正有：

- 修正 1　A9（线路图上的 AD9）可产生+14V 及数字 High/Low 三种信号电平。
- 修正 2　OE/Vpp 引脚上可以输出+14V、+12V 及数字 High/Low 四种信号电平。

表 14-1　EPROM 27512 的操作模式

Mode\P IN of 27512	$\overline{CE}$	$\overline{OE}/V_{PP}$	A0	A9	Output
Read	V_{IL}	V_{IL}	X	X	Dout
Output Disable	V_{IL}	V_{IH}	X	X	High Z
Standby（TTL）	V_{IH}	X	X	X	High Z
Standby（CMOS）	V_{CC}	X	X	X	High Z
Program	V_{IL}	V_{PP}	X	X	Din
Program Verify	V_{IL}	V_{IL}	X	X	Dout
Program Inhibit	V_{IL}	V_{PP}	X	X	High Z
Manufacture ID	V_{IL}	V_{IL}	V_{IL}	V_H	01H
Device ID	V_{IL}	V_{IL}	V_{IL}	V_H	10H

注：（1）X 可以是逻辑状态的 1 或 0。

（2）V_H=12.0V，误差为正负 0.5V。

（3）Vpp 的正确值请参考各厂商的数据资料。

（4）Manufacture 及 Device 的 ID 值会随制造厂商而变。

（5）本资料取自 AMD 1990 存储产品。

表 14-2　华邦 W27E512 的操作模式表

Mode\P IN of W27 E512	$\overline{CE}$	$\overline{OE}$ /V_{PP}	A0	A9	VCC	OUTPUTS
Read	V_{IL}	V_{IL}	X	X	VCC	DDUT
Output Disable	V_{IL}	V_{IH}	X	X	VCC	HIGH Z
Standby（TTL）	V_{IH}	X	X	X	VCC	HIGH Z
Standby（CMOS）	V_{CC}	X	X	X	VCP	HIGH Z
Program	V_{IL}	V_{PP}	X	X	VCP	DIN
Program Verify	V_{IL}	V_{IL}	X	X	VCP	Dout
Program Inhibit	V_{IH}	V_{PP}	X	X	VCC	HIGHZ
ERASE	V_{IL}	V_{PE}	V_{IL}	V_{PE}	VCC	DIH
Erase Verify	V_{IL}	V_{IL}	X	X	VCC	DOUT
Erase Inhibit	V_{IH}	V_{PE}	X	X	VCP	HIGHZ
Product ID-Manufacture	V_{IL}	V_{IL}	V_{IL}	V_{HH}	VCC	DA（hex）
Device ID-Device	V_{IL}	V_{IL}	V_{IH}	V_{HH}	VCC	08（hex）

注：（1）Vpp=12V，VPE=14V，VHH=12V，VCP=5V，X=VIH 或 VIL。

（2）本资料取自华邦 W27E512 的数据手册。

14-3　E²PROM 烧录软件的修正

把原来的线路修改后，接下来就是软件程序的修正了，W27E512 的烧录时序与 EPROM 非常类似，只是烧录前需要先做一个全面清除操作，另外也可以用特殊的时序去读取 W27E512 的原厂 ID 值（Manufacture & Product ID），这些都是原先 EPROM 烧录所欠缺的，有了这些特性后，可以使烧录的操作更为安全。当然依我们严谨的开发程序来做时，首先要写的是一个不算大的测试程序，用这个程序来检查几个重要的烧录引脚的电压是否正常运作？如果都正确后才继续着手写清除/烧录以及后段的验证等操作。诚然，如果一项产品是自行开发的，已保留了所有开发时的文件及原始程序文件，这些修正及更改都不是很棘手的，尤其是原先的 EPROM 烧录程序就是用 C 语言写的，旗威科技的工程师只要重新温习一下原来程序的写法，就知道如何下手修改了。

W27E512 烧录程序的操作和原先的 EPROM 烧录程序非常接近，首先执行一个 E²PROM.EXE 程序，让 PC 与 W27E512 烧录器连接，接下来键入“？”或“/”查询操作指令，最常用的指令分别是：

E：烧录数据的全面清除，这是 EPROM 烧录器上所没有的。

P：烧录 64KB 数据。

V：数据验证。

L：烧录代码的下载等。

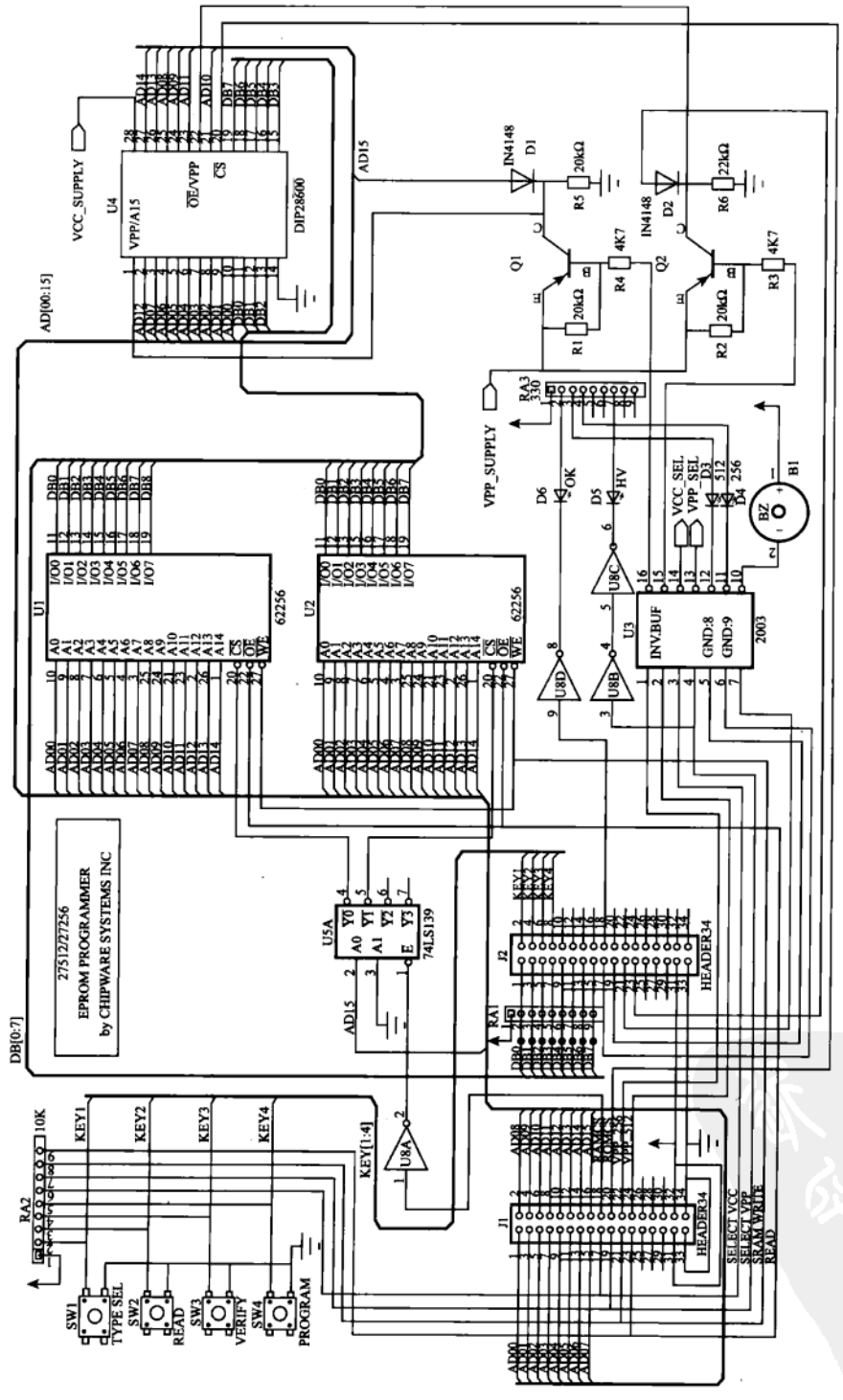


图 14-2 EPROM 烧录器的完整线路图（不含电压按制部分），图中的 Q1 与 Q2 分别产生两组 Vpp 电压给 27256 及 27512

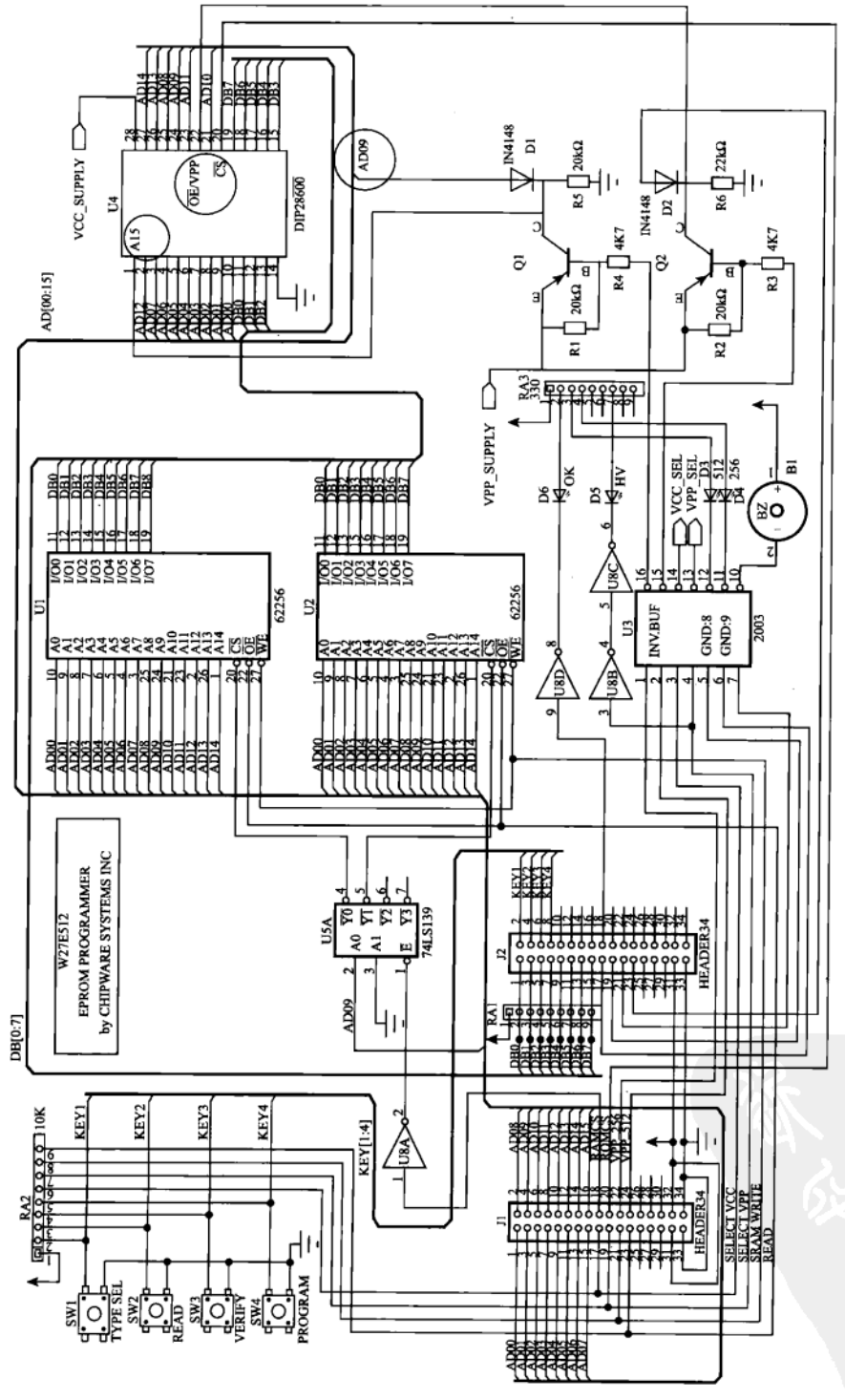


图 14-3　经过修改之后的电路图，Vpp 端可以产生 14V/12V 及数字 0/1 的电平，除此之外 AD9 引脚上也要能产生+14V 的清除电压，测试夹上的第 1 引脚已经固定是 A15 地址线了，修正部分已经用圆圈标示出来

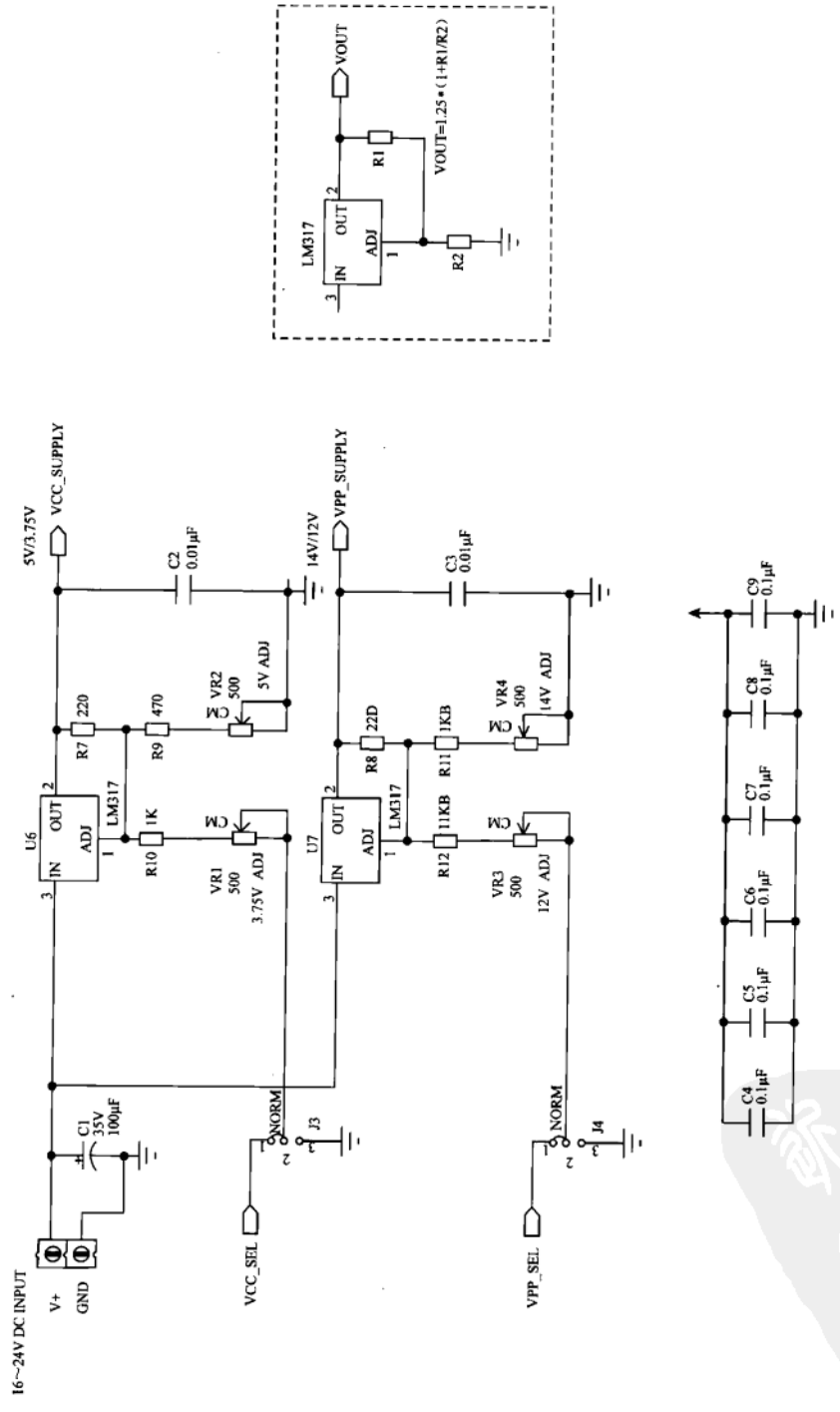


图 14-4 E²PROM 烧录板的电压控制电路，上方的 LM317 产生+5V/+3.75V 电压到测试夹的 Vcc（28 引脚），下方的 LM317 产生+12V/+14V 给 W27E512 做烧录及数据清除之用，须外加一个+24V 的直流电压输入

除此之外你还可以按下“1”键，进入重要的烧录引脚（Vpp/Vcc/A9）验证模式，这时可以选择数字 1～9 共 9 种方式，检查其中最重要烧录引脚的电平是否正常，输出电压的误差应该在 5%以内，否则就要调整烧录板上的 4 个 VR（VR1～VR4）可变电阻了，因为如果烧录电压电平不正确时，会导致烧录数据的错误，贸然不做这方面的电平验证是不行的。

W27E512 的烧录程序是放在一个标示为“E^{2}PROM PROGRAMMER”的 W27E512 上，这也代表我们确实完成了该 IC 的所有烧录程序了。您只要将原来 FLAG51 上的 EPROM 取下，然后换上该 E^{2}PROM 即可，接着接上两组 34P 数据线及+5V/+24V 电源后，就可以进行各项烧录电压的确认以及烧录的操作了。W27E512 烧录时各个烧录引脚间的电压变化是很快的，除非你是用示波器观看，才能看到波形的真正变化情况。想用数字电表去检查实际的烧录电压几乎是不可行的，因此，我们特地在开机时加上一个检查程序，只要你在开机时按住烧录板上的“type”键不放，就进入烧录电压的调整模式，倘若事先已经执行 E^{2}PROM.EXE 程序时，可以在 PC 的屏幕上看到一些信息，它会告诉您调整哪一个 VR，以便让烧录测试夹上的电压值达到额定值，你可以一手操纵数字电表的测试夹，另一手握住螺丝刀以便调整对应的 VR。这些重要的烧录电压未确认前，请千万不要进行任何烧录的操作。

由于我们已经把烧录的对象单纯化了，该烧录器仅能烧录华邦电子的 W27E512，所以电路一经修正后，就无法再烧录原先的 EPROM 27256 或 27512 了，这是因为 W27E512 除了有 ERASE +14V 的高电压外，在 ERASE 模式时，地址线 A9 上也加上+14V 的电压，如果你冒然将 EPROM 27512 插上时，有可能造成该 IC 的永久损坏，请特别留意。如果您已经有旗威科技的 EPROM 烧录板时，可以参照以下的步骤，进行 E^{2}PROM 烧录的改变，不过，修改之后就不能再度烧录传统的 27256 或 27512 了。

14-4 E^{2}PROM 烧录板的修改步骤

步骤 1：请准备好以下工具或设备：

（1）个人计算机。

（2）FLAG51 单片机控制板。

（3）E^{2}PROM 烧录板。

（4）有两组+5V/+24V 输出的直流电源供应器。

（5）取自旗威科技内含 E^{2}PROM.EXE 的工作磁盘。

步骤 2：参考线路图分别在线路板上 4 个地方用美工刀将线路确实切开，请参考图 14-5 与图 14-6。

步骤 3：用镀银线连接 4 个地方，正确的连接点位置请参考图 14-7。

步骤 4：拿出内含 E^{2}PROM 烧录程序的 W27E512，将其插到 FLAG51 的 U6 程序 ROM 插座上。

步骤 5：接+5V 及 GND 电源线到 FLAG51，+24V 及 GND 电源线到 E^{2}PROM 烧录板上，并且确定电源线的极性并未接错。

步骤 6：打开电源，并且用电表检查+5V 及+24V 电压都是正确的。

步骤 7：在 PC 端的 DOS 模式下，请执行 E^{2}PROM.EXE 程序，并指定由 COM1 或 COM2 与 FLAG51 控制板连接。

步骤 8：重新打开电源一次，并且按住烧录板最左边的“type”键，进入烧录电压模

式，请按照屏幕上的说明调整 Vpp 及 Vcc 电压，烧录板上的 4 个 VR（VR1～VR4）都要进行调整，必须将 Vpp 及 Vcc 电压调整到误差值 5%以内才行，接着请关闭电源。

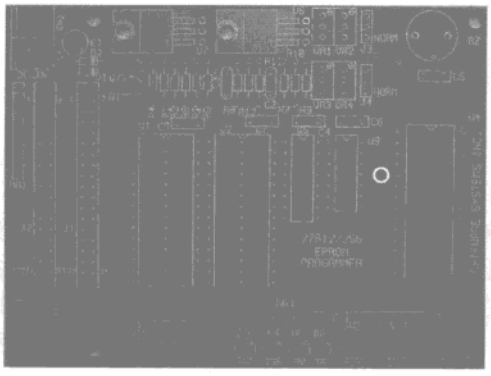


图 14-5　PC 板正面有一个地方的线路要切开

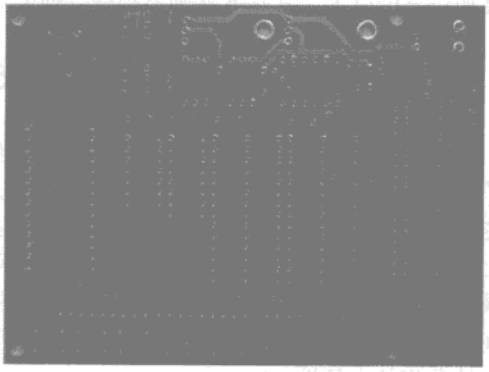


图 14-6　PC 板的焊接面有三个地方要切开隔离

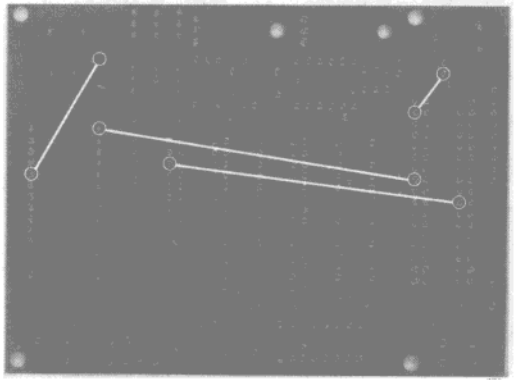


图 14-7　在焊接面上再补上 4 条镀银线就完成修正了

步骤 9：再度打开电源，不按烧录板上的任何键，让 W27E512 烧录程序启动，并在 PC 的键盘上按下 1 键，接着按下数字 1～9 共九个键，再度检查 W27E512 三根烧录引脚（Vpp/Vcc/A9）的电压是否正确。

当屏幕指示为数字的 HIGH 时，该点的电压只要在 2.0V 以上都是可以接受的，数字的 LOW 值则小于 0.4V 以内就对了。最后按“X”回到烧录主程序上。

步骤 10：取得一颗新的 W27E512，并且插入烧录夹上。在 PC 端上按下“E”键进行数据清除的动作，接着按“B”键做空白（全部是 FFH）检查，FLAG51 应该响应该 IC 确实是空白的。

步骤 11：按“L”键进行文件下载，接着输入该二进制文件的文件名称，并且指定由地址 0000H 开始存放这些烧录码。

步骤 12：按“P”键开始烧录，烧录程序会先执行 ERASE 的操作，然后进行数据的烧录，最后又做一次 E²PROM 与 SRAM 两者间数据的对比，都正确时才在屏幕上响应烧录完成。

步骤 13：按“V”键一次，再做一次数据的对比，如果还是正确时，代表所烧录的数据确实是正确完整的。

步骤 14：在烧录程序内部我们已经竭尽所能将烧录的时间缩短，如果您在使用上还是觉得速度不够快，请将 FLAG51 控制板上的 8051 CPU 改成 Dallas 的 80C320，就可以使执行速度加快三倍以上，其它元件则不须做任何的修正。

14-5 W27E512 的使用时机

W27E512 在使用上与 EPROM 27512 完全一样，其中最大的好处是数据清除的时间缩短了，以前更改程序时，都要准备十来个紫外线灯清除后空白的 EPROM，以便临时除错之用。也就是这个理由促使许多人改用 ROM 仿真器来修正程序，以免花太多时间在紫外线灯清洗上。不过，ROM 仿真器偶尔还是会有时序不对因而当机的困扰。改用 W27E512 E^2PROM 烧录后，直接插到系统 ROM 插座上，所花费的时间已经相当接近 ROM 仿真器的数据下载时间了，下一次规划或设计 8051 单片机的应用时，或许您可以尝试改用 W27E512，看看它所带给我们的方便性。

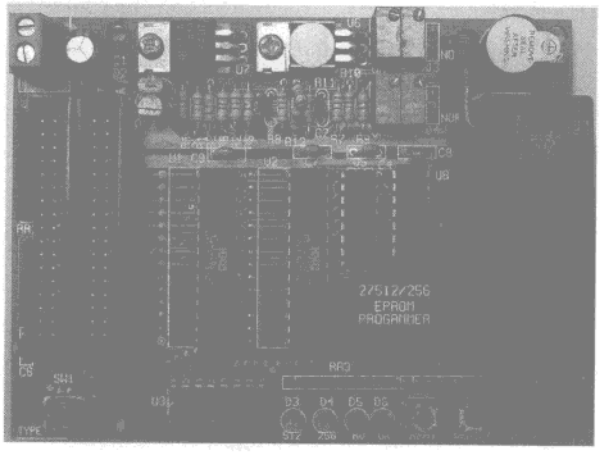


图 14-8 W27E512 烧录器外型看起来和 EPROM 烧录器一样，但是背焊接面补了几条线

E^2PROM 烧录程序是从 EPROM 烧录程序修改过来的，所以整个操作流程与 EPROM 烧录器类似，唯一的差别是程序中有 Erase 与 Vpp 设置的函数，程序要控制硬件产生必要的电压给 E^2PROM，方能清除内部的数据。

E^2PROM 烧录程序请查看随书光盘的 CH14_ E^2PROM 烧录程序彻底公开文件。

本章使用的软件

（1）2500AD 8051 C Compiler and Assembler。

（2）IAR 8051 C Compiler。

本章使用的硬件

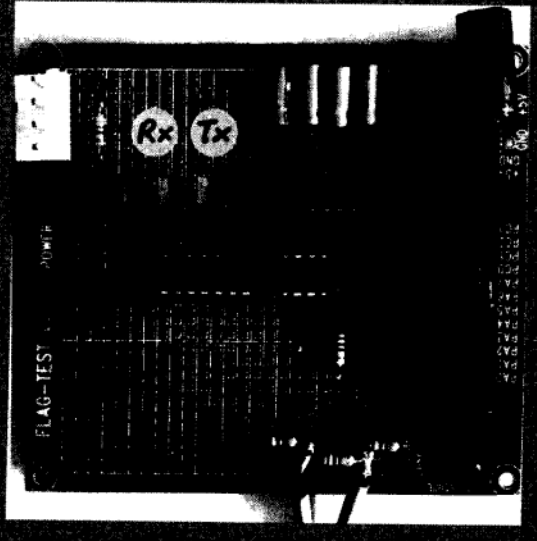
（1）FLAG51 单片机控制板。

（2）经过修改后的 EPROM 烧录板。
（3）华邦 W27E512（64K×8）。
（4）EPROM 27512（64K×8）。
（5）SRAM 62256（32K×8）/6264（8K×8）。
（6）Dallas 80C320：提高 FLAG51 的速度到原先的四倍。

相关资料网站

可经由下列公司、网站取得更进一步的信息：
http://www.chipware.com.tw：查询 8051 单片机控制板资料。
http://www.rs232.tom.tw：查询 EPROM 等存储器资料。
http://www.winbond.com：查询 W27E512 EEPROM 系列资料。
http://www.atmel.com：查询 Atmel 89C 系列资料。
http://www.dallas.com：查询 DS80C320 CPU 的资料。
http://www.iar.com：查询 C 编译器的相关资料。

15



第 15 章　IC 封装机的修改与规划

IC 封装是半导体制程的最后一个阶段，晶圆经过切割、打线后进行外部的封装，封装完后再进行电气功能测试及激光打印后就可以交付给客户了。封装过程的好坏也是会影响 IC 制造的合格率。

有一年五月中旬某天下午，突然接到一个电话："林兄是否有空？要不要到加工区去看一台设备？""好啊！现在正好闲着没事，一块去见识见识。"车开往加工区的途中，我们已经对这台设备有了初步的了解：这是一台相当昂贵的德国进口 IC 封装机（MODELING MACHINE），已经使用五年以上，机械油压部分虽有小问题，工厂的维修工程师还能做适当的处置，可是若碰到内部电子控制电路罢工时，就很难处理了。

对整个 IC 封装业而言，IC 封装机是非常重要的设备，封装机数量愈多代表能接受的订单愈大，因此如果封装机经常故障将会直接影响产量，对工厂的业务推广是非常不利的。更凑巧的是最近这些设备经常出状况，间接导致工厂无法正确预估产量及交货期，所以才有想要对 IC 封装机做系统更换的念头。由于该工厂的工程师主要任务在支援生产，才将封装机修改的项目转到外面来，工厂也知道能做这方面项目的厂商应该不会很多，如果行不通时只好买新的设备了。国外的新 IC 封装机价格至少要二十万美金以上，交货期又不确定，可能缓不济急，另一方面考虑是，若能借此培养出部分优秀供应商，对 IC 封装业未尝不是项好消息。

IC 封装业至有二十年以上的历史，接受委托商提供的晶圆，经过切割打线、封装、印刷及测试等过程后，才交回委托商的手中，每次订购数量少则数万个多则数百万个。因已累积了多年的经验，IC 封装业的产品不合格率都控制在数十个 PPM（百万分之一）以内，经过统计以及经验得知，大部分的不合格品是在 IC 封装的最后一道加工过程：封装（PACKAGE）时产生的，因此唯有妥善控制封装过程才能降低产品的不合格率。

15-1　IC 封装机，事实上就是精密的塑胶射出成型机

IC 封装机，事实上就是一台精密的塑胶射出成型机，它利用特殊的塑胶原料将已加工一半原先裸露的芯片及导线包起来，免得与外部的电路或环境接触。而其中的加工过程正是 IC 封装机的技术 KNOW-HOW 所在。首先，封装的材料（环氧树脂）要经过均匀的加熟，直到其软化为止，然后送入 IC 封装机中，封装机再以几乎恒温恒压的方式，将还是液体的环氧树脂灌到各个 IC 中，在维持该压力与温度约 60s 后，环氧树脂会逐渐硬化，这就成为我们常看到的 IC 黑色外观了。由于 IC 芯片加工的导线非常细微，超过 10 克的力量就可将导线扯断，所以环氧树脂灌入时的压力（力道）控制就显得非常重要，压力太小时环氧树脂可能无法完全填满整个 IC，而压力过大时，虽然 IC 外观是完整的，但是内部的导线已经断路或歪斜，以上两种情况所生产的 IC 都属不合格品。若 IC 封装机每次可以加工 120 个 40 引脚的 IC，

每个 IC 的单位成本假设 80 元，则一次加工就有可能面临将近万元的损失，并且做得越多亏得越多，若加工的 IC 是高价位的 CPU，如 486 或 586 时，那就更不容许出错了。

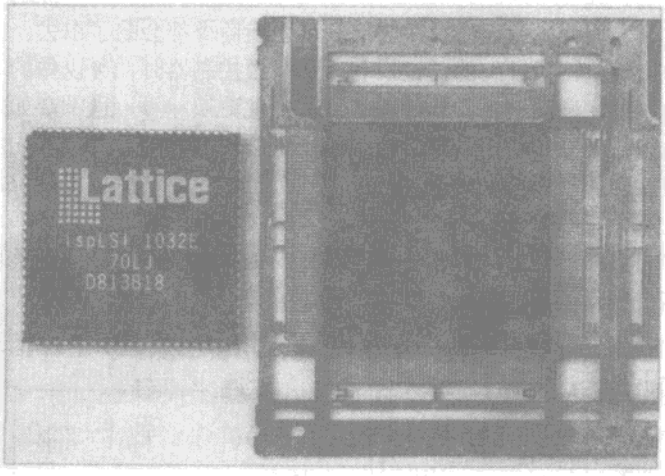


图 15-1　左边是一个完整 IC 的外观，右边是刚封装完的样子，还需要经过电镀、弯脚、激光刻印及功能测试后，才能交给客户

到达加工区后，又经过层层关卡及穿上头罩，口罩及防尘衣后才看到机器，平常看到的射出成型机都是横式的，IC 封装机却是直立式的，这些机器一字排开好不壮观，但是有几台已经罢工不做了，其他封装机则一天 24 小时不停忙碌地工作着，经过仔细地观看机器内部及机器的使用手册后，大略知道其工作的原理及控制的模式，以及封装机的操作方式。和射出成型机相同的是封装机也有相当完备的维修诊断功能及人员保护装置，这也是机器设备能行销全世界的必备条件。这台封装机以 MOTOROLA 68000 CPU 为主控制单元，采用 VME BUS 架构以及其他 3 块 I/O 组成压力与温度的控制电路，除此之外，还提供屏幕和兼顾防水防尘的键盘，供状态显示与数据输入用。现场的人员提到屏幕与键盘罢工的机率日渐升高，系统该更换的需求已相当明显了。

15-2 初 步 规 划

面对这些机器我们深深地觉得，国外在此方面的技术至少超前十年以上，虽然我们年年自称为“PC 个人计算机的生产量世界第一”，但是真正属于自行开发的部分则少之又少。类似这种无根的产业，只要碰到经济不景气时，保证一定会吃足苦头。可是真正有眼光的经营者又有几个呢？十年前的封装机已经做到屏幕显示以及几乎无缺点的控制，还有哪些可作修正的呢？

以下就是几个经验丰富的软硬件高手对 IC 封装机初步的更改目标：

（1）68K（6800 CPU 的简称）系统理论上不比 80X86 差，但是硬是被比了下去，因为现在的 PC 实在太便宜了。不过，应用在工厂的设备首重可靠性，所以决定使用工业级 PC，亦即工业控制上通称的 Industrial PC（IPC）。在市场上可以订到价格相当合理而且可靠性甚高的 IPC，IPC 在抗干扰及防尘方面都做了相当程度的加强。

（2）新系统硬件的引脚应该和原系统完全相同。如果要做试验时，只要将原系统的连接线移到新系统上，试验完毕后再恢复原状即可，这样的安排能使新旧系统更换时的变动率最小。万一在新系统试验发生问题时，旧有的系统还是能继续生产。类似这种项目最忌讳一次就把所有机器更新，假使发生无法解决的问题可能连商誉都会赔了出去。

（3）与原使用单位再做详细的讨论，利用 PC 做控制器时，可以做的事就很多了，如果新的系统只与旧系统相同，倒不如向德国原厂买零件更换即可，也不必如此太费周章了。

（4）所有的设备的开发与试验都应考虑到安全的问题，各个操作应先用模拟方式试验正确后，再与机械及油压控制系统结合，初步试验阶段一定要请原使用单位派工程师支援，以免试车人员发生意外。

（5）新系统可以毫无问题地加入中文显示以及加工曲线的指示，应该比原系统更具友善感与便利性，这会使得操作人员可立即由屏幕上获得必要的信息。

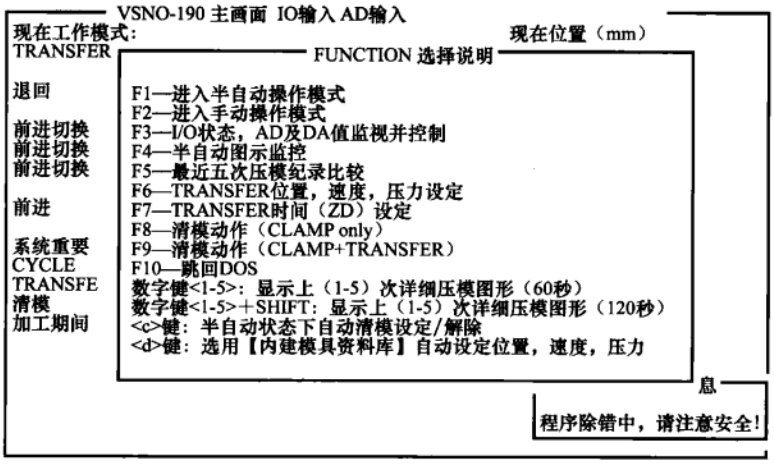


图 15-2　IC 封装机的主功能选择画面

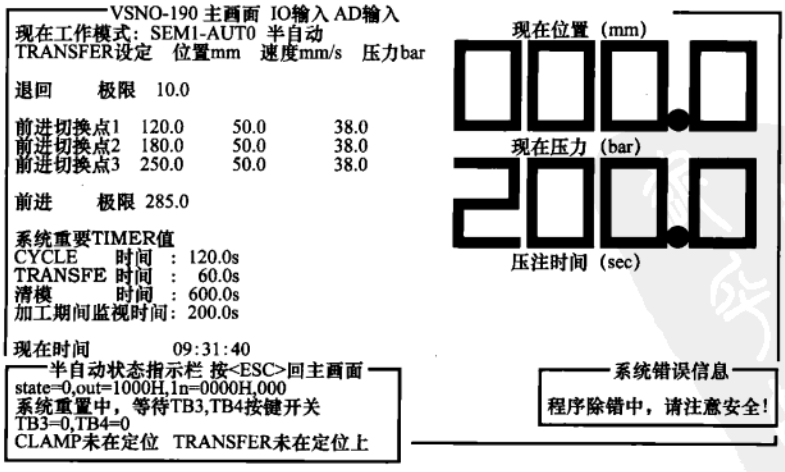


图 15-3　IC 封装机半自动模式下的画面，油压缸下降分别有对应的速度和压力

15-3 细 部 设 计

当整个系统要进行细部设计时，首先须拿到整台 IC 封装机械的所有操作手册及维修手册，这方面的说明以欧美厂商做得最完整而且毫不保留；日系厂商顶多画个框图介绍一下而已，其它的机械设备制造商更差，价值数百万的机器其手册可能只用个几十页简单的说明就交待过了。由此可见，我们的基础工业还是相当薄弱的。当技术者的我看到报纸的大标题："航空工业今年签约后，明年制造的飞机就可以起飞。"技术直觉马上告诉我：怎么有这么好的事，如果真的如此，保证会有一大票人立即投入这个行业，而不会像现在苦哈哈的 PC 业仍抱着不到 10%利润不放。

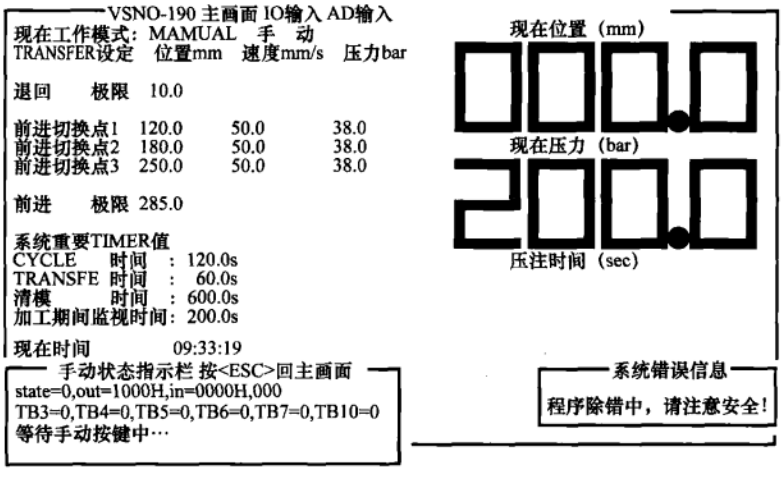


图 15-4 手动模式下可自由控制油压缸的升降

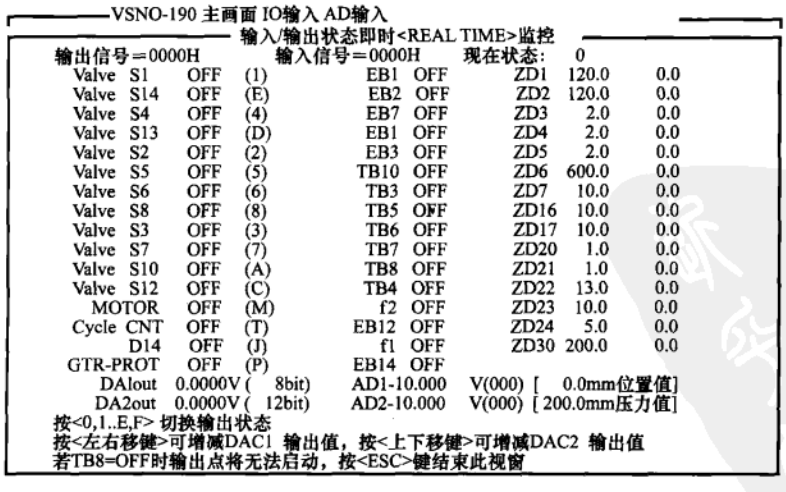


图 15-5 IC 封装机的输入/输出状态可以即时显示，并且视窗下方各有两组 AD/DA 转换器的输入/输出值

经过彻底的研究及讨论后，整个设计规划小组发现 IC 封装机的技术瓶颈在压力控制的比例调节阀（Proportional Valve）上，原系统将压力控制设计成一个硬件的闭环（Closed Loop）系统，由此比例调节阀控制整个系统油压缸的合模压力，并且随时都要保持恒定的压力。而这里所有的控制原理则是比例积分微分（Proportional - In- tegral -Differential, PID），这些理论在念书时都曾经读过，可是当时就一直质疑何时才会用到这些鬼理论，现在只好找书重新温故知新了。由此比例调节阀的技术资料得知，其频率响应的在 1kHz 左右，亦即比例阀每秒的控制速度无法超过 1000 次信号的检查与校正是“轻而易举”的事。所以，设计小组打算用 C 语言完成整个 IC 封装机的控制程序。软件程序中还包含一个油压系统的 PID 控制程序，程序基本的判断速度应该在每秒一千次以上，经过几个软件小试验后发现速度够快，于是就决定这样设计系统了，并且也暂定使用 BORLAND 公司的 TURBO C 做系统控制语言。为何如此？这是因为这方面的参考资料及文献最为齐全，而且万一有疑问时也有较多人可以咨询。接着，设计小组再分成以下几个组别，分别处理软件程序设计与硬件线路板的试制。

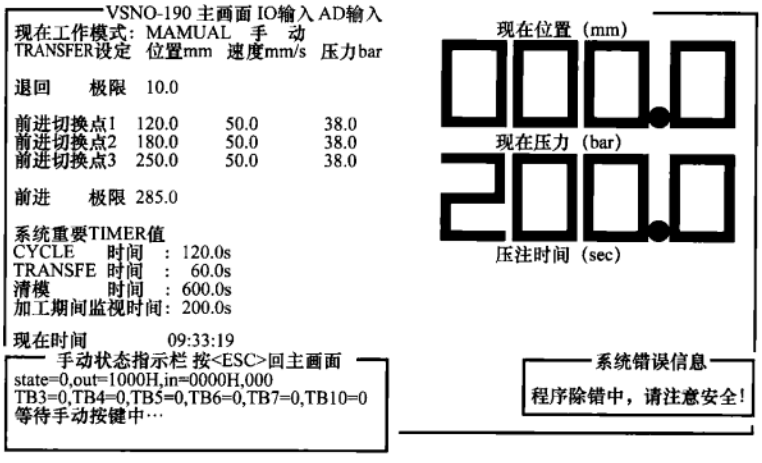


图 15-6 IC 封装机加工数百次之后一定要进行一次清模，将模子内的累积残渣排除，此为清模下的画面

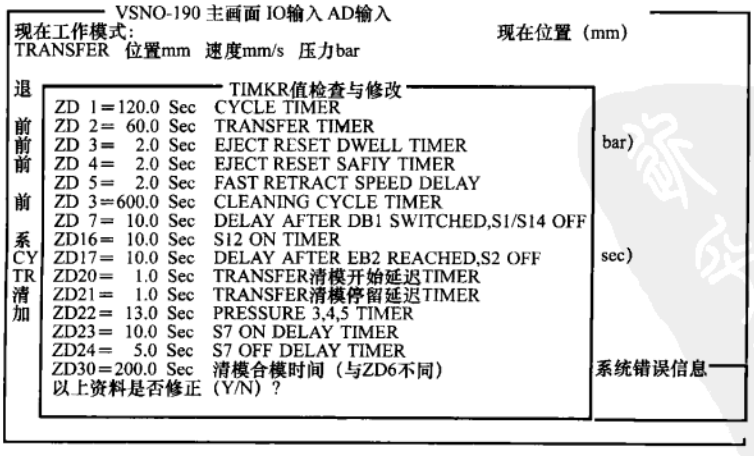


图 15-7 IC 封装机加工时，各个 Timer 的时间设定画面

小组一：负责系统程序的设计及控制时的画面显示，并完成第一阶段的模拟程序。

小组二：负责 PID 控制程序的设计，所有的判断及控制程序所耗的时间必须保证在千分之一秒内完成，把 PID 的理论基础变成程序会有一大堆问题要克服。

小组三：负责 PC 须添加的硬件线路设计，所有的信号及其电气特性必须与原系统相符，而且必须有更完备的人员保护设施，以保障现场操作人员的安全。

小组四：负责硬件线路板的制作以及 IPC 等硬件设备的组装。

项目经理：将以上软硬件整合起来，先完成第一阶段的模拟测试，然后到 IC 封装机现场进行第二阶段的软硬件测试，其中最重要的部分是 PID 控制部分。这部分若可行时才可以保证油压缸的恒压。

15-4 系 统 测 试

大部分的系统工程师碰到类似的设计，都会偏好用可编程控制器（Programmable Logic Controller，简称 PLC）来处理，PLC 对简单的逻辑顺序控制相当方便，而且可靠性也相当高，许多工业上的控制器都是用 PLC 组合起来的。可是，若同时要做画面显示与数据库管理时，PC 则有较高的优越性。在 IC 封装机的设计中，果然印证了我们的设计是正确的，而且硬件的费用也较为合理。至于软件方面的困难度，PC 和 PLC 应该都是相当复杂的，而且 PC 修改时的方便性及开发工具支持也远优于可编程控制器 PLC。

当第一阶段的模拟试验结束后，基本的画面显示与控制流程也有了初步的架构，为了使系统有更高的安全性，我们特地在硬件线路中加入 1kHz 的定时中断，在中断的服务程序中安排了系统的安全检查以及油压缸的压力控制，处理这些程序时，也曾造成计算机无数次的死机，但最后终于还是排除万难完成此一最艰巨的部分。不过不必高兴太早，目前为止仍是实验阶段，可不可行还得到现场试试才见真章。当大队人马将 IPC 及试验用硬件线路带进加工区时，真正的问题才逐一浮现出来，而且问题最为严重的竟是先前判断的关键：PID 控制！油压缸接受 IPC 控制时，曾有相当明显的抖动情形发生。有人说：写程序的人很少有“时间延迟”的概念；学电子的人有“重量观念”的人也是少之又少，所以，程序的操作就完全没有考虑到：机械的操作是需要时间延迟的。由于重量负荷的缘故，机械操作总会比程序的设定值慢几个 ms（千分之一秒）以上，这才造成机械本身不稳定的抖动，虽然程序又做了部分的修改，也经过十数天的校正，结果还是行不通！只好请老师傅出马了。经过高手查看后，认为 PID 控制还是用硬件线路制作为宜，尤像如此严格的压力控制回路，软件须长时间修改才能到达这个地步，所以线路中又加入了 PID 的硬件控制回路，经过几次校正后机械就没有抖动的情形了。令人讶异的是，增加的硬件成本不超过几百元，差点让已经搞得昏头转向的软件工程师跌破眼镜。所以说：处理事情有时候还是要做某些程度的变通才行，否则到头来还是害到自己。

15-5 试 用 结 果

系统正式进行测试时，处处仍以安全为重。首先测试所有的输入信号状态是否正确，然后配合中断服务程序的状态检查，立即验证 IC 封装机系统的状态是否正常，有许多状态在第一阶段中弄反了，都可以在此阶段做必要的修正。此时还可以与使用单位做更进一步的讨论，

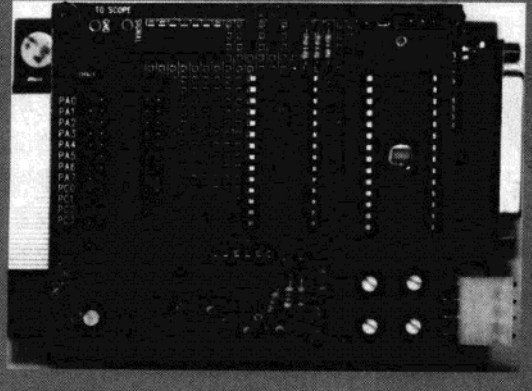
决定 IC 封装机最终的功能以及显示画面。最难的一关通过之后，设计小组依次完成了封装机手动模式（Manual）、清模（Clear）、加工参数设定与修改、模具数据库管理等等功能的试验，最后，才是半自动模式（SEMI-AUTO）操作的验证。此模式是 IC 封装机最常使用的，95% 的时间都是以此模式作封装加工。当然，刚开始时错误是颇多的，等到慢慢进入情况后，所有的控制信号就逐渐与原系统吻合了，IC 封装机的压力控制也能按照程序的设定变化，不论在哪个条件下都能一直维持在设定值上，这代表硬件线路做成的 PID 控制电路发挥了作用。

为了判断加工的状态是否有被其他条件干扰，设计小组在程序中加入加工图形显示的功能，可以在加工时立即将油压缸的位置以及压力变化情形，以曲线方式立即显示在屏幕上。这项功能对现场做维修与保养的工程师最有利，原系统缺少此项功能所以每次调整模具时都要花费不少时间和材料。由于机器仍处于系统整合与除错的阶段，所以，显示画面上一直有“系统除错中，请特别注意安全”的字样出现，等到所有问题都解决也相当值得学习：首先只换一台 IC 封装机，先进行一星期的试车，发现操作错误时，设计小组立即在现场作修正后，再进行三星期的系统测试，厂方亦可借此机会训练现场作业员，熟悉新 IC 封装机的各项功能，同时也有部分意见反应到设计小组这边，还有一些 bug 被找了出来，所以系统修正至此已经非常稳定了。接着厂方再以每周一台的速率换旧系统，直到所有封装机设备都换新为止。

15-6 结　论

如果公司的管理阶层能接受 MIS（管理信息系统）的观念时，可以将这些 IC 封装机用计算机网络连接起来，然后再与公司的计算机相通，只要一接到订单后，就自动分派工作到各台加工机械上，公司的主管可以立即查询每笔订单的执行进度，进而计算出正确的交货日期以及每笔订单的各项加工成本，这才真的像工厂全面自动化。产业要升级非得这样做不可。类似这种项目已不是单纯的电子电路或程序设计的问题，而是整合了管理、电子、信息、机械、自动控制等等的专业知识。做系统全盘规划的分析师若对上述专业领域没有起码的了解，是很难顺利完成这方面的项目。除此之外，公司的所有成员也可利用这个千载难逢的机会，尽量去吸收有关 IC 对封装方面的专业知识，这对专门接项目的公司而言，是相当重要的。一个项目能够顺利完成，除了公司必须有相当深广的专业知识外，现场支持原系统工程师的忠告与意见也是绝对不容忽视的，因为原系统在他们手上 RUN 了好几年了，原系统的任何缺陷与瑕疵他们是最清楚不过的。从开始接触 IC 封装机到第一台新设备顺利上线生产，共花了五个月的时间，比原先预期多了一个月的时间，这些设备上线以后，还经过几次程序小修改，至今已工作了一年多，再也没听过有严重死机的情形了，而且所有的控制电路和元件都在国内购买得到，不须像原系统一样所有维修及备用元件都得向德国原厂订购。由此我们也得到一个宝贵的经验：光修改 IC 封装机就要耗掉至少半年以上的时间，这还不包括机械部分的修改在内。

16



EPROM 烧录器的雏形板 Prototype。

第 16 章　自动化电饭锅测试线设计

本章作者将一个两地“整厂输出”项目的实际经验，毫无保留地告诉读者，绝对是别处找不到的好经验。

16-1　自动化电饭锅测试线的由来

承接自动化工程安装后的几个月内最怕接到客户求援的电话，路程近时还好，若工程修改还需搭飞机出差就苦了。1989 年我所服务的公司接了一条电饭锅生产测试线的工程，工厂座落在广州市郊，那时整个内地电饭锅市场仍是传统电饭锅的天下，当时厂方就自称这是内地第一条电饭锅生产线，所以担任整厂输出的与负责生产的双方都投入相当多的人力与物力，以便完成此项超过 250 万美元的整厂输出工程项目。

在生产电饭锅过程的最后一道关卡，必须对电饭锅做各种电气特性的检测，这方面发达国家或地区都有系统的测试标准，我们也不例外地有类似的标准测试程序，想要销售到某个先进地区时，当然要完全通过这些标准才行，电饭锅过了这一质量控制关卡后，就开始进行包装，然后分批装进货柜中。我们负责的部分是设计一条全自动化的电饭锅测试线，只要电饭锅经过共 8 台测试仪器的检测，即可得到非常详尽的电气数据，这些数据再通过 IEEE488 接口传到一台工业级的个人计算机上，由计算机自动判别哪个电饭锅合乎规格，哪个电饭锅必须再作调整才能出厂，当两个生产订单重叠时，也可以通过本测试线的计算机分辨出电饭锅的规格来。本条生产线预计年产 60 万台电饭锅，大部分行销祖国大陆。若一年有 300 个工作日，则每天大约要生产 2000 个以上的电饭锅，对一个员工数超过 2000 人的工厂是绝对没有问题的，唯一要顾虑的是产品品质与生产效率。

16-2　测试流程的安排

一个电饭锅装配完成后，随即由输送带送到最后一个电气测试线（长度约为 30m）上，工厂的作业员忙着为每个电饭锅接上电源，然后必须经过以下的一连串测试。若所得到的数据都在标准以内时，电饭锅光滑的表面上将没有任何注记；反之，测试设备会利用橡皮章自动在电饭锅表面上打一个记号，现场的维修人员再根据此标记进行零件的更换或调整。一个电饭锅出厂前，都必须符合规格方可上市，而生产线上的所有测试设备也是依此标准而设计的，我主要的工作正是规划并整合这些设备，如果市面上找不到现成的仪器时，就必须自行设计这类的仪器或设备，凑巧这方面正是公司开发部最擅长的。

测试站 1：加热线圈（HEATER）电阻测试，不同规格的电饭锅其加热线圈（heater）电阻必定不同，装配时若有错误，只要超出 5%的容许度时立刻可由本站发出警告。

测试站 2：接地电阻测试，电饭锅的接地电阻不可过大，否则易使操作者触电，本站即

做这方面的测试与判定。

测试站 3：低温功率测试，电饭锅刚插上电源时，由仪器判断其加热线圈功率值是否正确。

测试站 4：绝缘测试，电饭锅的加热线圈一定要与其他金属部位绝缘，否则也会使操作者有触电的危险，本站即在检验电饭锅内部的绝缘程度是否合乎标准。

测试站 5：耐压测试，电饭锅内部的线路必须有足够的耐压能力而不会损坏，本站产生一额定的高压信号，并加诸于电饭锅上一段时间，以确认电饭锅是否有此能耐。若造成耐压不足，则必定是电饭锅内部有绝缘不良的情形。

测试站 6：高温功率测试，当电饭锅加热有一段时间后，测试加热线圈所耗的功率值是否在额定范围内。电饭锅经过此测试站后，应由煮饭加热阶段跳到保温阶段；若开关未能动作时，代表电饭锅内部的温度开关有问题。

测试站 7：保温功率测试，检查电饭锅的保温功率是否在额定范围之内。

测试站 8：PC 总数据收集与 GOOD 和 NG（NO GOOD）判定。当电饭锅到达此站时，PC 会立即向前面共 6 个测试站上取得所有的数据，与内部的数据库对比。若合格，则站上的绿色指示灯点亮，反之，红色警告灯点亮并伴随着警铃声响，表示提醒操作员应将该电饭锅取下检修。

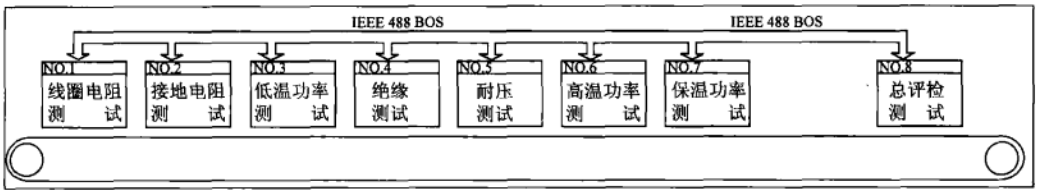


图 16-1 电饭锅测试线图

规划这类的项目绝对少不了参观工厂，以便对整个测试过程有更进一步的认识。以上的测试站是在参观完厂方原有的电饭锅生产线后所提的第一次建议案，除了拥有全自动化的测试特性外，每站送回的信号还包含了详尽的测试数据，例如低温功率测试过关，该测试站还送回该电饭锅的实际测量值×××瓦，这对产品的质量管理提供了最正确的数据。

厂方的技术人员相当能接受此方面先进的测试流程，因为原有的两条电饭锅测试线（分别由日本松下公司和无锡大学的教授所设计）都还没有类似的设计。当公司的高级技术人员兴奋地签妥订单分批回来后，才发现许多苦头正等着解决！说实在的，从事自动化这一行就得保证工程一定要能正常运转，否则是收不到半毛钱的。

16-3 测试站的局部设计

当开发人员进行测试仪器数据收集与整合时，方才发现很难找到合适的机种，而且大部分的测试仪器都属于“单机操作”型，一定要有个操作员在仪器旁边才行，与我们号称的“完全自动化”相距颇远；另一方面，几乎这类的仪器都欠缺与计算机联机能力；若有的话，价格也会让人吓一跳！以绝缘耐压测试仪器为例，原制造厂的回答很令人失望——“这类仪器我们至少已卖出千台以上，你们这种 APPLICATION 应用可能是第一次听到！”十二年前，我曾在大同公司的电饭锅厂内实习一个月，每天都用这类绝缘测试器修理有漏电的大同电饭锅，经过这么久之后，照理说应该有更进步的半自动化机种才对，可是，最后的答案却仍是否定的。我们只好改走另一条路：买现有的测试机种然后自行改装成智能化的设备。以下是一台

完整的测试站所必须拥有的功能：

功能 1：完全以 Z80 CPU 微处理器为控制中心，内含模拟转数字电路（ADC）以及数字输出电路。

功能 2：提供 IEEE488 仪器联机能力，可以较容易地与各种仪器设备或计算机联机。

功能 3：提供一光电式的近接开关，只要电饭锅一接近时，即可通知测试仪器开始测量。

功能 4：有一个气压缸输出点，当测量值超出容许范围时，由此点送出信号以便在电饭锅光滑的外壁上做一个记号，当问题点被修复后，这个记号就可以用干布擦拭掉。

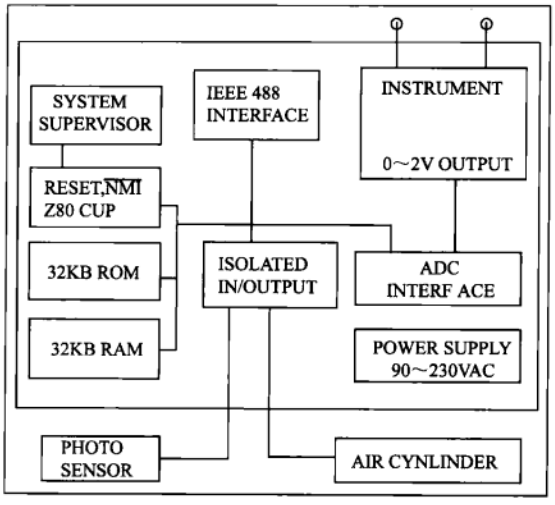


图 16-2　控制单元基本框图

功能 5：电气特性测试设备，其中包括功率表、极低电阻表以及绝缘与耐压计，这类仪器背部通常会有一个模拟输出点，可将测量值化成一个（0～2V）模拟电压值，只要经过适当的转换即可得知正确的测试值。

功能 6：必须有停电数据保留的功能，生产线会有因故障而导致全面电力中断的可能；当电源恢复时，原先的电饭锅数据必须能重新取得。

功能 7：每个测试站的实际测量时间约为 1～3s，大约可做 20 次左右的测量，这些值经过平均后才是最后的实际值，此值可立即显示在该站的显示器上。

16-4　进行模拟测试

所有的仪器订购与测试站的程序设计是同时进行的，当测试仪器交货且安装妥当时，程序也差不多有个样子了。接下来的一连串程序小修改是免不了的，当每个测试站的程序都完成后，就可以做与计算机的连接与数据传递测试，PC 这边必须建立各种电饭锅的基本数据以及容许误差，并完成电饭锅日周月三种报表的格式设定，并且反复测试各种情形。

不过，任何的模拟测试只能尽量减少程序明显的错误，至于程序或系统上的隐藏 bug 是无法在此时测出的。类似整厂输出项目的变数是相当多的，若估价时所保留的宽裕度不足，很可能会使得该项目变得更为复杂。系统仿真进行一段时间后，接着系统移到装配生产线的工厂，做进一步的动态测试，此时是将数个电饭锅样品摆到测试线上，进行实际测试以及测试数据的收集等，此时几乎公司内部的技术或业务人员都曾到现场提供部分宝贵的意见，等到按这些意见都修改完毕时，整厂设备完工的日子也快到了，剩下的小修改只好留到实际现场试验时再验证了。

公司派到现场的工程人员共分成三批：

首批是系统配线与安装人员，先完成系统的组装，并连妥生产线与装配线间的电气配线，初步调整整条线的生产速度。

第二批则是 PC 与测试站的程序设计人员，查看系统在试运行下还有哪些部分需做调

整，并且对系统的安装使用与维护提供必要的训练与说明。

第三批则负责系统的维护与最后的测试。

公司有擅于系统集成的工程师，也要有擅于人员调度的管理经理，否则一个整厂输出项目做完后，人员的流动率大增，反而对公司不利，这点是公司高级人员务必了解的。

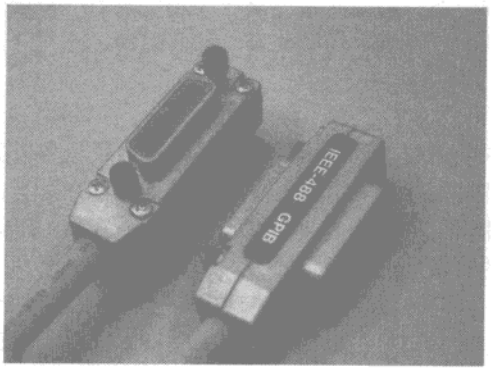


图 16-3　GPIB Cable 的特写近照

16-5　现场实地测试

“问题一箩筐”，是实地测试时的第一个印象。

在开发设计时，PC 执行是从不死机的，可是在这里却水土不服死机连连；另外每个测试站的操作好像都有一点问题，整条生产线动起来就是“不顺”，现场一连七天试验下来，大伙都心惊胆跳的。

此时，两地每日传真不断，一问一答或是一问数答，问题与情况当然愈来愈少，不过，终于有一天测试时致命的一击发生了，由于厂方电力供应的极度不稳定，竟然导致供应测试在线上的 AVR（自动电压调整器）与 UPS（不断电装置）全部损坏，测试只好中断，此时才有空仔仔细细观察电力的供应情形与品质，结果我们有了重大的发现：标示 380V 的交流输入电压，若容许误差为 10%时，最大的输入电源值应该为 418V 左右，可是用电表测量时，有时候可能升高到 450V 以上！难怪这边去的 AVR 与 UPS 都出了严重的差错，查看了损坏的机器内部，发现都是电容耐压不足全部爆开了，大部分机器设计的输入电源电压容许度都在 10%以内，可是这里的情况较为特殊，解决的方法呢？购买当地的 AVR 与 UPS 就没事了。

另外，负责绝缘测试的测试站经常会有不明死机的情形，只好把该站与产生高压的设备隔离远些，并做好系统接地后就没事了。从生产线正式开始安装到测试告一段落已超过 45 天了。

16-6　结　　论

实际上，并不是所有的公司都有能力做“整厂输出”的项目，唯有在某个特定技术上能够出类拔萃且行之多年，才能拥有此方面的资格，这也意味着经验愈丰富的公司愈容易处理或承接自动化这方面的项目。公司如此，个人也是如此。当问题出现时，最重要的是先确认

发生问题的原因，只有对症下药才能药到病除。接了这个项目之后，公司研究人员写程序的能力又增强许多外，对于机器外围环境的掌握与监控也增加了不少经验，而个人最大的收获是多了整体的规划与协调的能力，并且发现在自动化方面竟然还有许多市场尚未开发。

本生产线安装完毕系统成功地运行了两年多后，某一天下班前又接到该厂传来的传真信函，不过这次不是询问哪个测试站如何维护修理，而是该工厂又想扩充另一条电热水器的生产线，请公司务必提前到厂里详谈，当然我方一定会配合。

IEEE488 概述

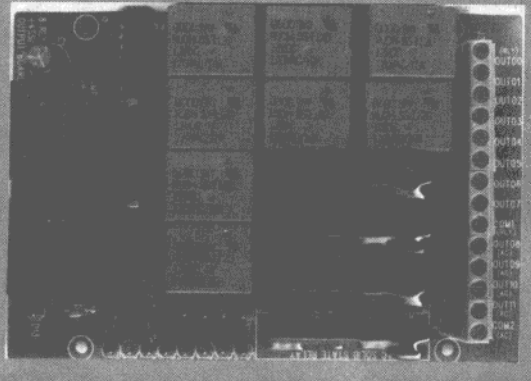
早期的测量仪器都未考虑到自动化测量的功能，每台仪器各以其独特的硬件接口与计算机连接，造成计算机联机时相当多的困扰。擅长于仪器制造的美国 HP 大厂，早在 1965 年时就着手为公司的各项仪器设计一共同的接口标准，此标准在 HP 公司内部称为 HPIB。

它定义不同仪器间的连接方式与通信协议，使得所有 HP 公司生产的仪器能更顺利地做各种自动化的测量应用。同时，HP 也将此接口标准提供给 IEC 协会和 IEEE 协会参考，在 1975 年经 IEEE 协会通过此接口标准，编号为 IEEE488。隔年，美国也将此标准列为其国家标准。以上就是 IEEE488 接口标准的由来。

只要仪器制造厂遵循此标准，就可以很顺利地与其他厂商的 IEEE488 仪器或设备连接，而完成一更复杂的自动化测试系统（Automatic Test System，简称 ATE）。如果你在仪器上看到 IEEE488、GPIB（General Purpose Interface Bus）或 HPIB 等字样时，代表这些仪器都提供一方便的连接接口，当然多了此功能的设备会比一般仪器都还贵上数万元以上，不过随着 PC 的广泛应用，这方面价格上的差异将逐渐减少。

图 16-4　备有 GPIB 接口的仪器，其售价至少可提高 1500 美元以上，图为示波器的背后

17



RELAY 与 SSR 输出控制板，我们经常利用它去做一些简单的输入/输出控制应用。

第 17 章　橡胶加硫机的设备改善

> 许多设备改善的背后是隐藏着许多的 Know-how 与 Know-why。当我们对系统愈了解时，我们得手的机率会更高，当我们对事情愈理解时，我们愈能掌控所有的突发事件。

在某杂志上有一个笑话：要害一个人的话，就叫他去办杂志。在“控制”的行业中，也有类似的笑话：如果要害一个人的话，那就叫他去接项目。以我个人的经验来看，很少有自动化的项目，只要评估可行性超过七成以上时，就值得去做了。如果可行性超过九成时，代表太过于简单，所以想当然就不会轮到我们来做了，而且项目通常延迟一到两个月才交货是很常见的。

在 8051 单片机的第二本书交付印刷后，心里就觉得很轻松，每天依然是做相关产品的研究与开发的事情。某一天下午突然接到一个求救的电话，电话的大意是：某个工厂的生产设备因故障停机，修改的控制电路一改再改，整个工程改善方案已经延期数个月了，如果还不解决的话，恐怕就不是写写报告就能够了事的。类似的“紧急救火工作”以前不是没做过，可是总觉得好像划不来似的。例如某电厂每年的年度大修时，以前我所处的工程公司都会跟着进入紧张状态，随时准备支持，公司经营了将近十几年了，虽然徒有名声，但是仍未摆脱经营上的窘境。接项目的背后，其实隐藏许许多多的危机。

到了工厂的现场看完机器设备后，大约可以知道使用单位的问题，也顺便统计了所需要的输入/输出点数，真正所要控制的输入点约有 10 点，输出点则有 8 点，似乎不是很复杂的系统。基本上，这是一个标准的过程控制机器，它接受控制面板上的各个按键开关，然后去控制一个大型的油压缸，通过油压缸去控制模具，以便完成产品加工时橡胶加硫的一连串操作。这些设备在加工时都与两个重要的因素有关：温度与压力，所以，控制器正式加工时，还要随时掌握温度与压力的变化情形，以便使产品的品质得以均一化。个人过去的经验有许多项目都是做类似的事情，所以处理起来应该相当熟练才对。

由于输出点不多，所以就建议直接就用 FLAG－PLC 控制器来做，只要单片机控制板、隔离输入板及 RELAY 输出板各一片就行了。正式安装时，上述的控制板是和其他的大电量开关安装在整个控制盘上，然后再由控制盘上拉出所有的控制开关，这些信号最后送到隔离输入板上。RELAY 输出板的输出则是先切换外部的 POWER RELAY，然后再控制高达 100A 以上的电磁开关，由此开关去切换模具的上升或是下降。当正式加工时，操作人员会先把待加工的工件摆到下模模具内，控制器会在收到启动的信号后，开始让上模模具下降，降到定位后接着灌入生橡胶，然后加大模具内的压力，以便橡胶能够与加工工件紧密地结合在一起，保持这个压力数分钟后同时把上模升高及下模下降，以便操作人员取出工件，几分钟内就完成一次完整加工的操作。FLAG－PLC 就是做上述的控制，只要能够确认所有控制的时序（TIMING），就可以开始着手写控制的 PLC 程序了。

由于本项目是属于“救火性质”的，所以拿到控制时序后，就立即设计程序，相同时间内工厂方面也配合修改控制箱内的电路，只要程序一好后就可以立即测试，但是实际下手去做时才发现问题还不少，以下是这些问题点的症结：

（1）FLAG－PLC 无法做在线计数或定时值的修改，我们原先是规划 PLC 在测试修改时，必须靠笔记本电脑重新加载程序，可是实际的情况是现场的工作人员不一定会操作计算机，少数的设置值最好能开放给现场的领班或资深的人员来操作。

（2）紧急停止的定义与 FLAG－PLC 好像有所不同，本设备中最重要的控制组件是油压电动机，所以紧急状况时应该立即关闭电动机，让油压缸控制的模具停止，这个操作牵涉到操作人员的安全，所以要再对该操作做多重的确认。

（3）一台安全的机器设备开始操作前题是：操作人员双手已经远离模具，再加上该人的左右手（请注意是双手）都同时按下启动键，这时才算是正式的启动。在工厂的加工现场中首重的是“人员的安全”。

上面的问题中首先要解决的是增加部分的硬件线路，以便让操作人员能在加工前设置几个重要的加工参数。我们的处理方式是另外设计一个由 AT89C2051 控制的 0～9 指拨开关输入接口，随时读入设置值，然后通过单片机控制板上的扩展接头（内含 P3.2～P3.5），把设置值传给 FLAG－PLC。由于我们能够百分之百完全掌握 PLC 的控制程序，所以针对这个 PLC 的应用程序，我们加入了 XREAD 与 COMBINE 的额外指令，前者每次可以由 P3.2～P3.4 四点读入多达 80 点的数字数据整合成数组有效的设置值，我们的 PLC 程序再将这些值转送给定时或计数器使用，这样就能达到随时可修改加工参数的目的了。如果我们的 PLC 控制板上有一个简单的 LCD 接口以及数个按键的话，或许就不需如此动刀动枪了，所以下一版的 PLC 控制器上，我们一定会把 LCD 的接口给纳入，以方便测试或修改。

实际测试前，还是发生了两个小插曲：由于 PLC 程序设计与修正只花了几天的时间，在尚未做实物测试前，我们都是使用了 I/O 监视板来模拟输入/输出的操作，如果无法在该监视板上看到正确的 LED 操作，那就不需要到现场试验了。当我们转换到控制箱内做试验时，每次只要一启动切换 100A 的电磁开关时，FLAG－PLC 就莫名其妙地死机！操作十次约有八次会死机！这未免太离谱了，一知道这个消息后，就带了存储式示波器及各种设备到现场测量 8051CPU 及输入/输出板所收到的信号，结果看起来都很正常呀！搞了一整天人仰马翻依然没有结果。可是隔天出发前却接到一个喜讯，现场人员在检查接线时，无意间发现有一条电源地线没接牢固，才造成系统非常不稳定的死机情形。锁紧此条电源线后，就不再死机了。另一个插曲是在确认每个开关的动作时，其中某些操作也会切换与油压缸有关的电磁开关，这时偶尔会造成系统的 RESET，这时 8051 控制板上的八个 LED 会从头各自亮一次，然后再进入 PLC 的控制程序，所幸我们已经对这个现象有多次的处理经验了，不必怕！这是 FLAG51 控制板上的 RESET IC T518 对电源的变动太敏感所引起的，只要把 T518 取下，并将 RESET 用的 one-shot 电容值加大即可，该单片机控制板再经过硬件小手术后，就不会胡乱 RESET 了。

实际测试时，我们是把 FLAG51 上的 8KB SRAM 改成 NVRAM，以便让系统保存最后一次加载的 PLC 程序，由于先前已克服了死机与异常 RESET 的现象，所以我们让系统从头到尾仿真加工一次，并随时查看所有的时序是否都是如我们之前预料的。如果不是，则立刻在笔记本电脑上修正程序，直到所有操作都正确为止。当然还要即兴刻意地去按下紧急开关，看看 FLAG－PLC 紧急处理的反应是否正确？这时做测试时心里的重担已经除去大半了，所以只要所有的操作都正常后，这个开发项目也准备告一段落了。在这里我们为这个 PLC 开发项目做了相当详尽的介绍与解说，不知道对您的工作是否有所启示？如果您认为在这里确实有收获时，也请您来信给我们鼓励或建议。在公司的专属档案柜里，储存有不少类似的已归档自动化案例，如果无法做经验的传承时，谁的损失最大呢？

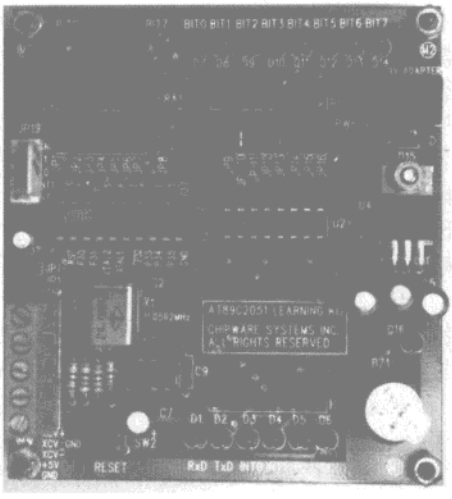


图 17-1　由于 FLAG－PLC 没有更多的输入点，所以我们另外用一片内置 AT89C2051 的控制板去读取 10 组 BCD 指拨开关的输入设置值

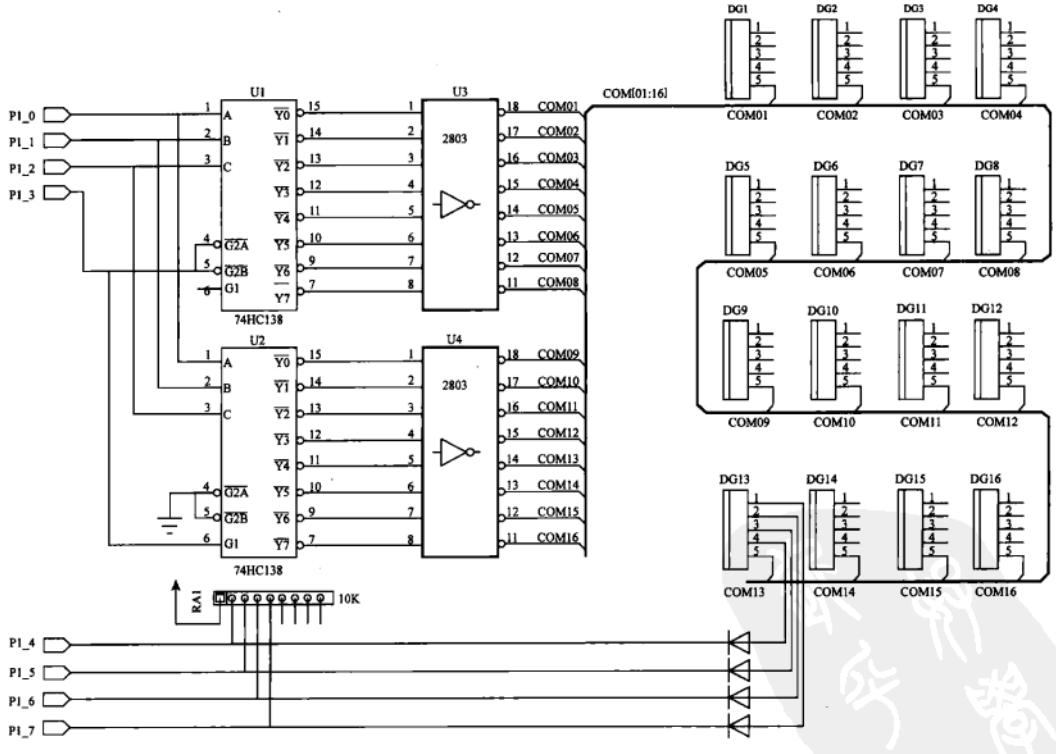


图 17-2　我们只用了 P1 端口上的 8 根引脚，就可以读入最多达 16 组指拨开关的值，图中的二极管应该有 64 个，但怕影响线路图的简洁性，只列出其中 4 个而已

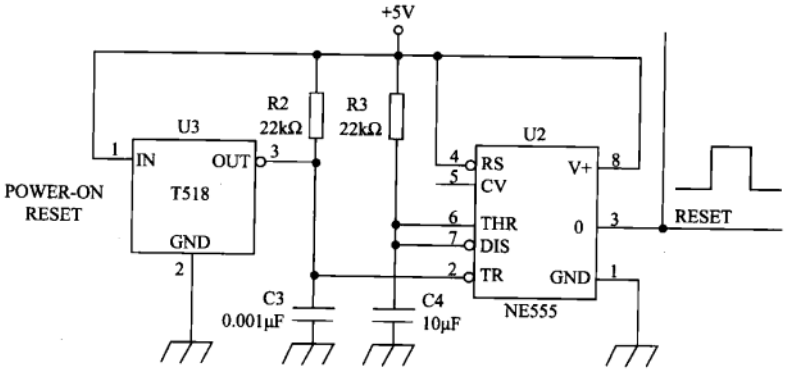
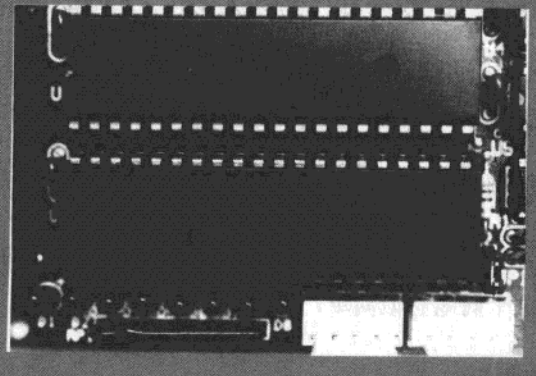


图 17-3　FLAG51 上的 RESET IC 对电源的变动太过于敏感时，会使得系统经常 RESET，如果您有这方面的困扰请把 T518 移开，并将 C3 电容加大即可

18



老兵不是死只是凋零，8051 虽然已有 20 年以上的历史，但它已深入扫描仪、数码相机、影碟机与引擎控制器中，很难被淘汰的。

第 18 章　经验谈：鱼跃龙门的思索

> 认证考试只是获取一张证书而已，它不是就业的护身符，也不是就业后一帆风顺的保证。

18-1　“证书”不等于“保证就业”

假如您关心 IT 信息的话，最近应可以在许多报纸媒体上看到各种计算机技能鉴定的消息。此项鉴定考试是由专门的机构主办，而由委员会负责执行。鉴定的职业技能分成：计算机软件应用、计算机软件设计和计算机硬件等类，在考试前最雀跃不已的是各种计算机补习班了，早已备好各种大补贴：考前总复习、考前秘籍和各式各样的光盘及教材等等，准备大发利市，感觉上和每年一次的高考一样，可是鉴定合格就能保证在计算机业中畅行无阻吗？这是一个非常值得探讨的问题。

当我们在中学准备高中考试时，老师常会以非常严肃的口吻说：考上后就一切天下太平了！而当我们顺利地进入高中后，又得面临三年后的大学考试，此时听了最感到欣慰的话就是：考上大学后由你玩四年！幸运的话，还要由不得你再加考四年。所以，在进入社会大学前的莘莘学子，都已是身经百考的战士了。有人由于高考制度的缘故进入 IT 业，也有人因为兴趣选择了 IT 业，当然也有人因志趣不合而离开变化诡异的 IT 业。我个人认为 IT 业本身的变化性非常大，除非经常吸收与阅读相关知识，否则不出两三年就自然会被淘汰掉，所以，鉴定合格与保证就业间并非是划等号的。

18-2　企业需要会思考的信息人员，而非证照合格人员

由于自己本身工作性质都属于系统设计方面，曾经帮过不少家公司（电子、信息、机械和自动化公司）应征及遴选软硬件工程师，我们并不在意应征者的学历，在意的反而是应征者以前的经验以及其对信息的获取（Aggressive）程度，Aggressive 的中文意义是“有非常大的侵略性”的意思，因为在这个领域中要学习的东西太多了，只有不断学习与研究的人，才得以脱颖而出。通常只要出二到三个问题就可以看出应征者的功力了，例如在某自动化机械公司应征应用工程师时，我习惯会出一道问题：请问你如何将一个机械的开关（Switch）信号读回到计算机内？这个问题看似简单，但是经过仔细分析后却有许多问题须要过滤及探讨：

（1）该开关的种类及电平。

（2）该开关是否要经过抖动处理（Bounce）。

（3）该开关是否要经过隔离（Isolation）。

（4）该开关的切换速度。

（5）计算机对该开关的反应速度。
（6）该开关是否容易受外界干扰。
（7）该开关是否要做安全规范上的考虑。
（8）该开关的可靠性如何。

再举一例，假设某科技公司要招考数名软件应用工程师时，我的问题绝对不会问你如何用 C 或 CLIPPER 写一个数据库管理程序，而是问你：假如公司要开发一套自动提款机的程序，你会怎么做？这个问题实际上涵盖了数据库管理、密码验证确认、数据传输和实时交易等问题，如果要好好地回答以上的问题可能要花上许多时间。如果应征者想要几天后再提出方法也行，这是对个人在 IT 界的经验逐一验证，主要在观察应征者的思考路径与反应能力，这些都是学校现有的教科书上无法解答的。唯有多方涉猎信息的人才能回答的很完备也很恰当，而这正是企业和高科技公司所想要的人员——一个会思考的工程师。我们也深信绝对无法用数小时之内的学科考试方式找到合适的工程人员，这当然也包含此次的信息大考在内。

18-3 信息教育应该做适度的调整

十几年前我在学校学习计算机概论时，期末的成绩刚好在及格的边缘，并非不认真学习，而是感觉到老师想要把所有他懂的知识完全倾囊相授，但是自己吸收有限，所以大部分都还给当初授课的老师了。现在回想起来，当初若能用计算机应用鲜明的实例来引导学生，而不是照着课本从头教到尾的话，或许能多培养出几个计算机尖兵来，以下是最近几年来计算机业中相当著名应用实例：

- 爱国者导弹如何击落飞毛腿导弹。
- 股票证券如何买卖及凑合。
- 自动提款机如何完成一次提款交易。
- 商店如何利用 POS 完成货品的盘点与订购。

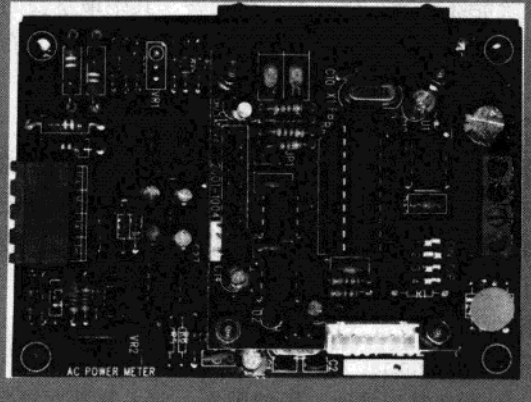
以上都是计算机的深度应用实例，如果老师们能以实例说明来引起学生的兴趣，就不会再有计算机恐惧症的学生了。如果，读者是 IT 业的前辈，应还记得十年前银行的提款机是非常非常难操作的，上年纪的人完全无法接受这种操作方法，家里的老婆婆每次提款宁可用存折，也不愿意操作 ATM 自动提款机，她常会问：为什么我已经按了提款金额了，还要按一个“确认”键才能领钱？其实“确认”键正是计算机键盘上的键，你若不按该键，ATM 将无法得知你要提领的金额多少。话虽如此，可是当时的系统设计人员以为所有的提款人都是计算机专家，所以每次领钱都要按上十几个键，给人很冷漠且不通情理的感觉。不过，最近新银行陆续改进后，每次提款就方便多了，只要按下密码，指定快速提款以及提款金额代号，不到八个键就完成提款的程序。我们的教育制度也是有类似的毛病：想把每位学生都教导成无敌金刚，所以拼命地塞东西给学生，忽略了常识和研究是应该有所区别的。常识应该让所有的人晓得，有兴趣的学生再辅以专门的教导，使得知识能平均分布于社会各阶层中。

18-4 期盼 IT 业生力军的加入

IT 界向来是充满活力与变化的，只要能像海绵一样随时接受各种新知识，一定能在业界有立足之地，合格证书只是有此资格而非成功或就业保证书，所以不论你是否参加此次认证

考试，请记住：天下没有免费的午餐，也没有鱼跃龙门这种事情。一些出名的大厂譬如“宏基”计算机曾经摔过跤，“伦飞”也跌倒过又爬了起来，“佳佳”和“凌亚”计算机就没有那么幸运了。数年前他们都跃过所谓的“龙门”了，今日的结果却截然不同！假如你报了名参加今年的软件考试，请全力以赴；反之则请忘掉鱼跃龙门这件事，不论做什么事情只有全心投入才是最重要的。

19



旗威科技公司的 AC 能源监控器，可以监视线上的电压电流与 pF 值。

第 19 章　套装软件的诚信问题

> 购买昂贵的套装软件对大公司而言是一项严格的考验，而一般消费者也必须认清“中文化”背后的许许多多枝节。

您买任何产品时，是否曾经把内附的说明书从头到尾看一次呢？我的态度习惯是如此的，同时我会把这些手册小心地收藏好，以便以后有需要时可以随时翻阅。最近因为工作上的需要以及想更深入地了解一下计算机印前系统，所以经过一些基本的评估之后，添购了一些计算机设备与专业的套装排版软件，至此，我才发现：一个国外相当盛行的软件，碰到中文后往往水土不服，而且经过所谓的中文化处理后，样子变了许多，其中“中文化”的处理过程及态度颇值得商榷。

19-1　好软件应该内外都吸引人

我们认为一个好的套装软件除了程序本身要吸引人外，使用者的操作手册设计的是否得体也占了相当重要的份量，如果使用手册写得很详细时，实际的使用者在初次安装及使用时，实在没有必要再去买相关的参考书籍了，因为只有开发套装软件的人员是最熟悉自己的东西，程序做到哪里要注意哪些事情，实际参与设计程序的人是最清楚不过的。所以说，一份好的使用手册必须有以下的特点：

（1）清楚标示基本硬件配备

最好是在套装软件的封面上就注明清楚，然后在手册中的前几页再提醒使用者系统可以工作的基本配备，以及系统能够顺畅执行时的顶级配备，其中最重要的是说明到底要有多少 DRAM 才够用。千万不要系统都安装好，正式执行时才响应“存储空间不足”。

（2）安装时的详细说明

如果使用者已经有了基本的配备，初次安装时多多少少都会碰到一些状况或问题，例如 PC 系统的原先设定及内存的管理等等，这部分的说明最好能为使用者说明各种状况的处理原则，而不是直接要求使用者改成软件需要的格式，这往往会导致存在硬盘的其他程序无法执行。许多游戏软件往往有这种霸权式的做法，要求您改这个改那个，改完后游戏是可以执行的，但是隔天就发现计算机好像不太听话，怎么其他程序都罢工了。

（3）详细的初次操作说明指引

很多厂商把这一部分称为 Tutorial（即学习教程），如果套装软件要能被使用者接受，Tutorial 绝对是居首功的，因为唯有正确地引导使用者如何去熟悉并接受这套系统，才能让使用者了解该软件的诀窍，通常这一部分中会举一些简单的操作实例，教导使用者实际去操作并体验其中的精髓。要成为一个熟练的使用者，这一关绝对是必修的，只靠看书而不做任何

演练，就能学会计算机的几率应该不高。如果您不看任何的资料，直接就操作各种套装软件，我想您只是学到基础入门的技巧而已，至于更高深的操作示范及软件的精华您是无缘接触的，因为只有亲自跟着学习教程或用户手册操作，才会知道相关的基础知识。经过这一关卡以后，您再看市面上的应用书籍时，得到的灵感与心得可能会更多。

（4）正式的使用手册

这部分应该是叫用户手册（User's Guide）。这部分才是软件的正式使用手册，它是专给入门者所使用的，如果您要进一步学习新的操作或命令，应该能在这里找到足够的信息及示范。

（5）参考手册或称 Reference Guide

所有的套装软件的命令以及用法应该包含在本参考手册内，并且以索引的方式供用户参考及对照，对用户愈重视的厂商，这方面做得愈好。

19-2 中文化的程度就要看原厂的态度了

通常，较负责的国外软件开发厂商都会有类似上述的手册配合套装软件的推出，但是在遇到中文界面时就会衍生出以下相当复杂的改版问题了。

（1）不是完整的中文环境

现在所谓的“中文个人计算机”应该还不能算是完整的中文环境，它只有把部分的英文信息改成中文而已，所以整个系统还是在英文环境之下执行，不过，大部分的中文版软件都只做到这里而已，由国外的公司拿到原始的程序代码，再将部分重要的信息及菜单改成中文就算交差了。这种做法当然就完全不会考虑到当地用户的特殊用法或需求了。

（2）中文手册的编写

如果您以为所有套装软件的中文手册是在我们本地编印，那可就大错特错了。国外的套装软件厂商通常是先找到合适的配合厂商翻译，然后再交由专家校阅，无误后才发行中文的使用手册。当然上述的做法是最理想状况，当遇到要如期推出中文版时，可能就不是这样处理了，最常见的例子是把手册拆散并交给好几个团队翻译，然后直接集合成书，这种推出手册的速度最快，但是如果没有经过仔细的校对，保证会有文字翻译前后不符的情形发生，而且操作的文意没办法贯通，相同的英文关键字在手册中有不同的解释，而且如果手册中又完全不加入中英文对照时，会让用户看了手册后，一直不懂其中的意义，这些误解很可能都是编写态度有问题造成的。

（3）删掉绝对不能删的部分

为了能够如期推出中文版，某些软件是将 Macintosh 的版本稍做修改就推出 Windows 版，或是全盘再由“日文版”改来，系统的稳定度与效率已经打了折扣，原本在 Macintosh 上完全不出问题的套装软件竟然变成问题丛生，可是随着交货日期的逐渐接近，厂商竟然把英文版原有的 Help 文件与 Tutorial（初期教学）文件砍掉，以减少系统当机与出错的机会，这迫使用户必须随时翻阅使用手册才能得知各种命令的用法，问题是如果中文手册又交待得不清不楚时，那就要长时间去试了，这种“锯箭式”的错误改版方法竟然还行得通，我们认为这才是最不负责任的中文版本。

19-3 代理商最不愿意看到的英文字：TERMINATE

如果您有看过电影 TERMINATOR（魔鬼终结者），就应该知道 TERMINATE 这个字的意思。九月初兴冲冲地到凯悦饭店参加某个排版软件中文版的发布会，这才发现该软件的原代理权已终结，现在改由新的厂商代理，对我而言，只要该软件的手册写得完整，我都可以一一地试出来，几乎不需要代理商的服务，但是对于众多使用该软件做排版的业者而言，如果未被充分地告知，这个对客户的“诚信”问题似乎是相当无法理解的。至于现场的价格更是令人热血沸腾，“在上市发布会前买该软件的费用竟然比较贵，而且贵得很离谱”。所以，我们要奉劝所有的消费者，购买任何进口的计算机设备或套装软件时，最好该品牌在该地区已有正式的分公司，碰到类似的“诚信”问题还有地方可以投诉，如果您是向该地区的代理商购买时，也该确认一下这家代理商的服务态度如何。如果该系列产品一直处于热卖或销售不佳时，都有可能代理权随时被原厂收回，不过不论是哪种情况，您的套装软件可能会在一夜之间变成软件孤儿，无法取得任何的协助。我曾经向某大代理商买了 NORTON COMMANDER，也依照书上的说明寄出了用户登记卡，至今还未收到该公司的任何产品更新或相关的资料，我一直在怀疑代理商把这张注册卡摆到哪里了？至于寄到国外的注册卡有用吗？我们真的不得而知了。所以我们以一流的价格买到二流中文版套装软件，而且碰到 N 流的中文手册。

面对众多进口软件中文版的陆续推出，我们的建议是：

（1）先加强自己的语言能力，要求厂商提供英文版的手册，稍稍做一下中英文的翻译对照，顺便检查一下原厂中文化的态度，很多手册的中文版比英文版难上数倍，举例说：某排版软件中文版有一个功能叫“修葺图形”，但是又没有提供中英文对照表，所以就相当难了解其功能了。

（2）购买前要求能先进行测试，不过一般这种要求可能无法被代理商接受。

（3）千万不要完全相信广告上的宣传，当任何新产品推出时，所有的广告所提的优点都很类似，因为这些资料都来自同一个地方，当然只提该套装软件的优点了。

（4）至今尚未有非常公正的机构为这些套装软件写测试报告，所以相信自己就好，其实询问正式的用户最能得到较客观的评价。虽然有许多杂志都推出所谓的编辑选择奖推荐某些产品，但是几天初步的试用仅能看到表面的假象而已。对消费者而言，您买了它后，烦恼才刚刚开始而已，尤其是套装软件的中文版。

20

无线的串行通信一定要做 Bit Error Rate 测试，以验证接收端的电路是否正确。

第 20 章　WHO CARE

Who care？谁关心周围的大大小小事务呢？技术人员与您一样都是相当关心的，许多政策或决策出发点错了之后，很可能导致全盘皆错。写程序也是相同道理，当所有的结果都是负面时，你才会发觉不重来不行了。这篇文章发表在 1996 年 3 月的 RUN！PC 杂志上，虽然事隔多年但是读起来仍然让人会心一笑。

曾经认识一个相当尽职且专业的医师，如果问他：这件事 Who Care？他一定会回答：Nurse care。身为技术人员每天除了技术领域钻研外，我们对社会诸多跟技术有关的事还是相当注意的，如果把目光放在最近一年上，我们还会看到许多令技术人很难接受而且很难相信的事情，但是这些事情 Who care？

20-1　捷运系统的百分之百安全才通车

每次坐飞机出差时，总会有两件事情让我印象深刻，一是机场上空始终有一层不算薄的雾，可见空气污染的问题始终没有解决；另一项则是到处可见的捷运施工，捷运系统的各站早已完工，偶而看到电联车的来回测试，可见对于何时能够全面通车，没有能够拍胸脯保证确定的日期。可是对技术或工程人员而言，哪来的“百分之百保证”呢？航天飞机的安全够高吧！可是还是有一艘在发射的过程中不幸爆炸，而核电厂都不能保证百分之百的安全，捷运系统当然无法达到所谓的 100%安全了，如果这个官员说法是 100%真的时，那捷运可能就 100%通车无期了。可是万一捷运通车了，那就代表绝对安全吗？Who care？

20-2　全民医保计算机化无限商机变成了无限修改

1995 年 3 月本地开始实施全民医保，这绝对是项好的政策，可是医保部门要求各个医院及诊所购买个人计算机，通过连网程序申报各项医保资料，引起不少工厂厂商的高度关切。因为由此而衍生的商机是无可限量的，可是医保部门的申报格式及内容从 1995 年 3 月开始一改再改，工厂软件厂商的无限商机这下子变成无限的修改，不仅商机尽失还得随时对医院修正软件程序，否则医院就无法领取医保的医疗费用。这种一改再改的动作正意谓着整个医保的制度还是处于学习及尝试错误的阶段。

由此也可以见到医保制度并未经过仔细地推演与精算就仓促成军，其想法是对的，但是事后的诸多修正反而造成更多人及医院的不满与不便，如果所有重大的决策都是这样做决定的话，这绝对不是好事。

20-3 高速公路的低速自动投币收费系统

1995 年年底高速公路的收费站上启用一套自动投币系统，据说该系统一共花了几千万的经费，并且早已完成验收，但是一直不敢正式启用，最后终于上场收费了。这套收费系统可以收各种硬币，且可以自动判断是否是伪币，不过整个判断时间要花上 20s 的时间，而一般的收票动作只要三秒钟就完成了。不论是电视或报纸的报导，在报导结束前，媒体记者都会加上一段话：请练好投币技术，否则车子经过该投币系统的时间可能就不只 20s 了。

我们认为当初如果知道一辆车的自动投币要花上七倍以上的时间，是否还要花上几千万来证明这样做是行不通的？而且所有币别都能接受的设计也是值得商榷的，因为如果要判别所有的币值，一定会浪费不少机器判断时间，这样收费站前的车流量一定有所影响，何不限定只收 1 元硬币呢？如果我判断没错的话，这套低速的收费系统“应该”在今年内就会下台一鞠躬，且让我们拭目以待吧！不过事后也不会有人对此事负责，因为“这是汽车驾驶员的投篮技术差，而非系统不管用”。所以，Who care？（编注：这篇文章是实施全民医保时的文章，全民医保已经要改用 IC 卡了，且今年高速公路还在为自动收费系统要用“无线式”或“红外线”感应纷扰不已，而“又”流标了。）

20-4 春节前的铁路语音订票系统竟然死机

1996 年的铁路售票可以用电话订购，回想当年为了要买车票返乡必须整夜排队买票的惨况，终于有一点改善了。不过第一天却完全卖不出一张票，依据一般的说法是，有太多电话一次涌进语音订票系统，竟然造成订票计算机主机的死机。这种说法或许可以对上层交待，但是这些可能出现的状况在系统规划或测试时都没有被发现吗？或者是这个订票系统还未经过测试就准备上阵接受众多旅客的订票，这不出问题那才怪呢！碰到这种事的返乡客只好自认倒霉了。

很庆幸地听完连声抱歉，又隔了 24 小时之后，计算机系统才恢复正常订购车票，如果数据无法及时的修正，而是要求所有旅客自行到车站去排队买票时，那可是会引起公愤的。您认为这种事还会发生吗？我们百分之百认为会再发生，但是过完春节后，Who care？

以技术人员的眼光来看，这些诸多问题应该不会发生才对，因为这些毛病早在设计初期就应该被挑出来才对，而不是到正式上机时才想要修正，这时的修正绝对是太慢太慢了，而且对承包商的商誉影响也是相当大的。

20-5 Internet 上技术询问信函的泛滥

如果您经常通过 Internet 查询资料时，除了浏览各个公司的主页外，一定也会加入几个专业的讨论群。以我个人为例，隔几天我就会去看 eletronics 讨论群中的内容，其中有几种类型信函，我是完全不给予 reply 的。第一类正是课题做不出来的求救函，理工科的学生如果课题做不出来是无法顺利毕业的，所以这类求助信函每年快到毕业的月份时，一定会涌现到各个相关的讨论群中。第二类信函正是标准不用大脑的求助信，例如标题是“步进电动机有六条线，请问如何量？”而信中只有少少的一段话：“as title”。

早期推广 Internet 时常会引用一个例子：某位研究生想定某方面的论文，所以他说在 Internet 上发一个求助信函，几天之后就收到来自世界各地高手的解答，当然也就很快又很顺利地完成论文的写作。这个例子给我们做了一个彻底的“错误的示范”，如果大家都发出求助函，那谁会有义务提供您进一步的资料呢？在 Internet 上大部分的人都是相互且义务性的 HELP，如果期待与您完全陌生的人们伸出援手时，下次别人有难时您也会伸出援手吗？所以自行收集资料是应该的，碰到问题自行解决也是应该的，而运用 Internet 只是诸多方法之一。

下次如果您碰到问题时，请先在自己这边先行发酵，一定要自己尝试解决问题，否则不幸有一天电话线不通时，那 Who care？

对我而言，1995 年年初才是我的视窗 Windows 第一年，选用 Windows 软件的主要原因是使用 Windows 下的 Pagemaker 排版软件，虽然这套排版软件已因代理权的两次转移变成了软件孤儿，但是个人认为它已为我解决了 95%以上的排版问题，倒是 Windows 3.1 中文版在打印时经常会死机，造成一些操作上的困扰。经过几次教训后，随时备份资料变成为休息前的反射动作。这些套装软件虽然稍有瑕疵，但是我不会因为其中的小毛病而舍弃 Pagemaker 或 Windows 而不用，如果真的要等到完全没有 bug 时，就如同“捷运通车=绝对安全”一样，不仅不合实际，也无法用合理的价格取得该套软件。

以我们推出的 EPROM 烧录器 DIY 为例，我们程序在正式推出前，至少烧录及测试过 30 种以上各个厂牌的 EPROM，但是如果您要我们保证所有 EPROM 都能烧录的话，当然是可以的，不过该 EPROM 烧录器的价格要是现在价格之后再加两个 0，这时您还能接受吗？Who care？We care, Chipware care！

21

旗威科技公司所开发的第一代电池残电控制器。

第 21 章　几个深刻经验与教训

在呕心沥血的项目中，有些是很难完成而最后忍痛放弃的，但是这些失败的经验却是身为技术人员的最佳教材与借鉴。

10 项产品上市后真正存活的可能仅剩两三项而已，不过在尚未正式生产，有许多产品雏形被设计工程师一改再改，仍然无法符合规格上的要求，最后只好宣告放弃，身为技术人员当然最不乐意见到的就是这种情事，但是不幸的事情还是有可能发生的。

以前有机会授课的时候，课程最后总结时，我总会把地图画到黑板上，接着分成北、中和南三大部分，然后询问学员想听哪个地区的应用范例，因为在这一行里已经待了相当长的一段时间，也实际参与执行了数百件设计的项目，各大研究单位大概都曾经接触过，光是这些应用实例就可以讲好几天了。可是，在这些呕心沥血的项目当中，也有历尽艰辛完成的，同时也有最后忍痛放弃的。这些经验是用许多心血及金钱换来的，我们常常自我提醒不要再犯相同的错误。许多人对这类伤心的往事往往不愿再度提及，可是“失败的经验”对于想踏入此行的技术人而言，绝对是一个最佳的教学教材。

21-1　经验 1：弹尽援绝——智能刺绣机的开发

飞行员衣服上的胸章臂章以及鸭舌帽上的徽章，图像鲜明且立体感强，这些图案不是用人工一针一针绣上去的，而是智能刺绣机的得意杰作。实际制造刺绣时，是先请打样的人员将待刺绣的图样放大数倍，然后用数字板（digitizer）把刺绣的颜色及顺序一一输入到计算机里面，最后再把这些刺绣数据传到刺绣机器上，按下 START 键后就开始加工了。刺绣的机械操作看起来好像很容易，可是却处处危机重重。如果您踩过缝纫机的话，应该可以注意到刺绣的针是一针一针刺下的，踩得愈快时针下来的次数就愈多而且也愈快，我们必须在刺绣的针头离开布的瞬间，马上移动下方的布匹到达下一个落针的位置，确实定位后才等待针的下降，这段可利用的时间通常都不到 0.1s，也就是 100ms 以内要分别控制 XY 轴的电动机到下个定位点。如果针戳到布之后电动机仍在移动时，针可是会马上折断的，而且一次断 10～15 根针，因为这类的绣花机械一次都可以绣 10～15 件加工物。针断后刺绣机器要立刻查觉并马上停机，等待操作者更换新的针头。计算机控制的刺绣机首先要克服的就是避免刺绣针的折断。一个平常的臂章约要一万多针才能绣完，所以当中不能有任何错误出现，否则就会把加工时间延长。

原本机械厂的意图是直接“模仿”国外的机种，可是经过仔细的研究与探讨之后，结论是不可行的。第一，如果要照抄就不需要我们的协助，因为有人对照抄这方面比我们还行。第二个理由是刺绣机原厂所使用的零件，其中有不少是国内买不到的。最后一个理由是，如果要开发类似的设备，何不自己来培养这些基础人才呢？就是在这个理念的坚持下，我们花了将近八个月的时间完全了解雏形机电子部分的开发及测试，相同的时间下机械部分也组装

完成了，接着就是实际的试验了。

图 21-1　计算机刺绣的成品图

这段期间内我们也透彻研究了刺绣数据的传输文件格式与其通信协议，所以，原先的刺绣数据磁盘，可以在个人计算机的屏幕上先看到最终的刺绣结果，然后再通过我们专用的接口将数据传到加工刺绣机上，刺绣的速度由最初的两秒一针逐渐提升到每秒 10 针的速度，我们最终的目的是到达每秒 15 针的速度。加上这些实际测试的时间后，整个刺绣机的开发时间已经接近一年了。不仅开发者及委托设计的机械厂都开始不耐烦了，开发雏形机的我方希望再增加一些经费，以便将剩余的机械操作测试完成，可是机械厂却不同意，结果是导致双方都僵持在此点上。机器已经确实在操作了，如果真的是因为经费的问题而就此打住，实在是划不来的。由于事情发展的不是很明确，所以稍后我就离开这家参与开发的公司，并且留下所有的控制原始程序以及参考数据，我想这是一个负责任的技术人员所应当做到的。

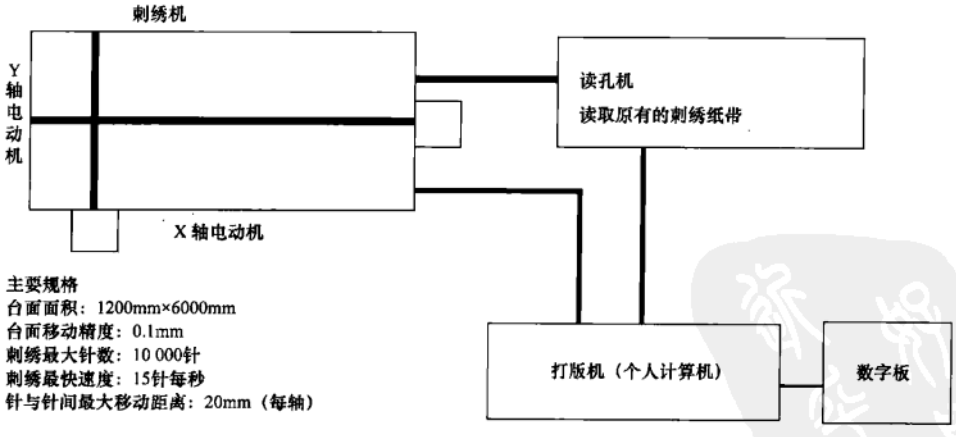


图 21-2　刺绣机的示意图

自动化的设计方案如果跟机械操作有关，绝对是非常有趣的，每次看到设计的刺绣机在工作时，所有参与设计的工程人员都会兴奋不已。这个设计案当中最值得一提的是：从头到尾我们只用了 8KB 的程序空间，每当控制程序超过 8192B，我们就回头看程序的内容，把常

用的部分子程序再浓缩及归类，以便再度缩减程序的空间。所以从头到尾都是使用一个 2764 EPROM。至此之后，我每年还是会到这个机械厂拜访一到两次，听说该项目后来转到某个研究单位又执行了好几年，结果就不得而知了。

21-2 经验 2：尚未上演就下台——电容电感的质检自动化

接一个项目可能会强迫工程师猛“啃”5 种以上的专业知识，所以持续下来获利最多的就是参与设计的工程师群了。如果公司无法好好地保留这些人员，那绝对是公司的一大损失，这点是所有公司的主管都要理解的。以前我很难去想象规模庞大的电子公司是如何对供货商的元件进行质检的，若今天进料仓库收到 50 万个电容或电感类的电子元件，要怎么样才能确认这批货的误差率在容许范围以内，当然不可能是 50 万个元件全检吧！这里面还包括许多统计上的学问及技巧。而公司所生产的 PC 专用电源供应器上有数十个元件组成，如何使其都符合标示上的规格呢？这些都是质检工程师所要去掌握的。

我们所负责的任务就是设计一台电容/电感/变压器等多用途的测量仪器，以方便进货的质检人员确认以上产品的误差及容许度。实际上我们做的是一个用计算机控制的工具（Fixture）及信号切换箱，它连接了待测物以及一台高精度的电感测量仪（LCR Meter），然后依照元件的规格去切换电感测量仪的功能，并且通过 IEEE488 接口读回测到的值，质检人员随机抽检几十个零件后，计算机就可以印出完整的报表来，并且分析该元件是否可采用或是要做退货。用计算机来做统计上的分析是最方便不过的，而且用计算机读取测量值保证不会有人为的误差出现。只要我们得到一连串元件的数据后，可以分别计算出其平均数、标准差以及超出规格的不良比例，据此就可以判断出整批元件的好坏了。

由于先前我已经对 IEEE488 的概念非常纯熟了，而且也出了一本关于 IEEE488 应用的书籍，所以通过 IEEE488 BUS 读取电感测量仪部分的程序几天就解决了，信号的开关切换箱也不是难事，较麻烦的是 IQC 质检概念的建立及程序撰写，但是事情只要用心做下去就不会有所谓难的了。

不论客户在哪里，通常上班时间一到达时，我们就已经在工厂的入口处办理手续准备进入厂区。这也算是一种处事的精神吧！该工厂的实际位置在苗栗铜锣，而我们是从高雄出发，由此您就可估算出我们的出发时间了，经过几次的高雄与铜锣往返之后，所有的测试操作及报表都按厂方的要求修改好了，可是我却发现工厂正弥漫着股票准备上市的不安定气氛中。我们依照惯例对质检人员施以系统操作上的基本训练，接着当然是系统的验收及完成。

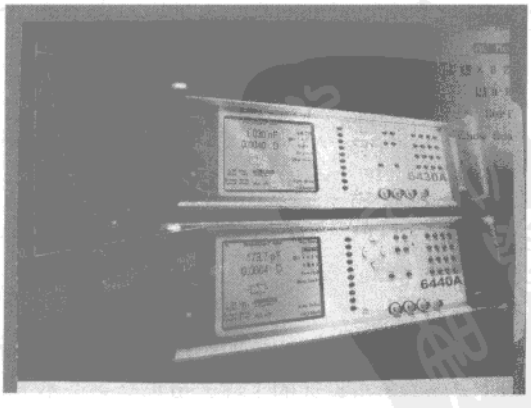


图 21-3 LCR METER 须配合其他工具才能发挥最大的功能

整个系统验收通过后，我们却觉得这个公司已经在大幅调整人员职位了，而且着手遣散部分员工。训练员工绝对是要花上数个月甚至数年以上的时间，再加上无数的心血，但是要他离开却只要不到半天的时间。我们不知道这套元件测试系统最后转到哪

家工厂或单位上，但是这家公司最后的结局是股票上市被驳回，公司高级主管纷纷离职，电厂关闭后，PC 与显示器厂也准备被拍卖，而罗姓的公司负责人至今仍是官司缠身，这是又一个异业投资且扩充过快，因而导致失败的鲜明例子。

艾钜科技电源器 IQC 检验报告

检验编号 RV300　　共 3 页 第 2 页

供应厂商：TDK　　温　　度：28℃　　湿　　度：25%
品　　名：RES　　规　　格：10KR
重送日期：08-10-1991　　交货数量：10000　　MAJOR：PASS
检验日期：08-19-1991　　抽验数量：100　　MINOR：PASS
检验仪器：WAYRE KERR 3245 + 3220A
检验依据：NIL-STD-105E　　JUDGE：ACC
备注说明：SECOND PAGE

测试数据（标准值 1 mVDC min）

003.777	003.760	003.761	003.760	003.763	003.759	003.766	003.756
003.767	003.754	003.769	003.758	003.757	003.749	003.755	003.752
003.750	003.751	003.751	003.765	003.747	003.755	003.748	003.746
003.758	003.752	003.747	003.746	003.757	003.753	003.746	003.745
003.745	003.756	003.753	003.742	003.740	003.745	003.741	003.741
003.745	003.746	003.749	003.739	003.734	003.737	003.737	003.731
003.744	003.736	003.732	003.742	003.730	003.730	003.728	003.732
003.724	003.737	003.739	003.729	003.726	003.729	003.737	003.727
003.727	003.727	003.730	003.732	003.720	003.719	003.725	003.722
003.717	003.723	003.716	003.717	003.719	003.718	003.717	003.721
003.720	003.710	003.707	003.713	003.716	003.718	003.709	003.712
003.710	003.707	003.710	003.705	003.700	003.704	003.706	003.691
003.703	003.703	003.696	003.702				

次数分配图表

组中值	次数f	u	fu	fu^2
3.6910	1	−5	−5	25
3.6996	4	−4	−16	64
3.7082	11	−3	−33	99
3.7168	13	−2	−26	52
3.7254	11	−1	−11	11
3.7340	13	+0	+0	0
3.7426	16	+1	+16	16
3.7512	15	+2	+30	60
3.7598	11	+3	+33	99
3.7684	4	+4	+16	64
3.7770	1	+5	+5	25
合　计		累计次数f＝100	+9	515

Ca=(X－μ)/(T/2)%
　=(3.735－1.000)/(0.086)%=3179.99%
Cp=(T/2)/(3σ)
　=(0.086)/(3.0＊2.735)=0.010
P=P1%＋P2%=30.30%＋0.00%=30.30%
　Z1=3Cp(1＋Ca)=　1.031查常态分配表得 P1=30.30%
　Z2=3Cp(1－Ca)=－0.969查常态分配表得 P2=0.00%
精确度Ca=CLASS　D. 精密度Cp=CLASS　D. 总评P=CLASS　D

标准值：1mVDC
全　距：0.086
组　数：10
组　距：0.0086
平均值：3.735
标准差：2.735

品管主管：　　确　　认：　　检 验 员：LINDA

最小值误差表示

图 21-4　IQC 检验报告的范例

21-3 经验 3：开机状态的困扰——电表智能化测量

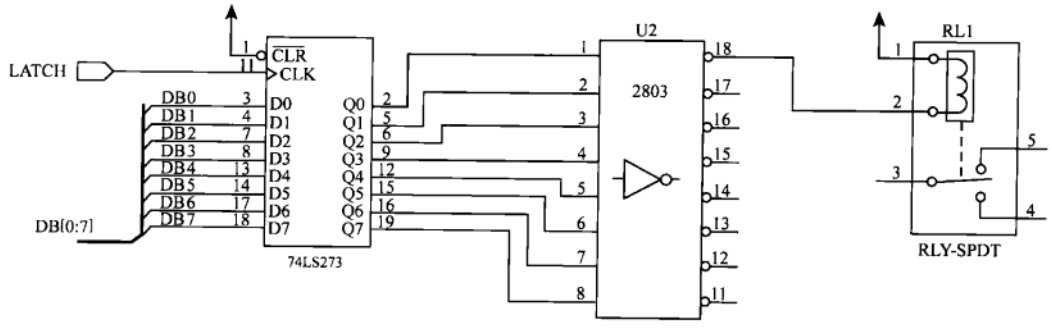
每年电力公司的发电厂都会安排一段时间，对厂区内所有的发电机组及仪器设备进行检查及维修，我们通常把这个重要的活动称为“大修”，大修期间整个发电机组都要停下来，所以大修都选在全省耗电量较少的冬天执行，这段期间所有的仪表及设备都要一一进行检查及校验。功能不正常的要更换，指示有误差的要重新调整，同时所有的调整及校正都要有正式的书面报告出来，所以，用计算机来进行所有的测试及调整是势在必行的。不仅可以节省大部分的人力，而且能够自动产生各种统计报表，如果大修有限时完成的期限时，全自动计算机化确实是必走的路。

我们被派的任务是完成一个具 IEEE488 的开关控制箱（Switch Box），一边接实验室级的校验器（Calibrator），它可以产生标准的电压电流等标准信号，另一端可以连接 8 台待测的电子仪表，借由内部的切换就可以进行所有信号的测量及调整。由于内部可能有大电流通过，因此开关箱内的信号切换都是用超大型的继电器做切换。其中的所有控制都是由一台工业级的个人计算机再加 IEEE488 卡来控制。由于本项目是相当特殊的规格，而且使用部门要求较高的稳定度，所以我们特地为该台设备制作正式的 PC 板，整个控制器最后安装在标准的 19 英寸仪器箱中。

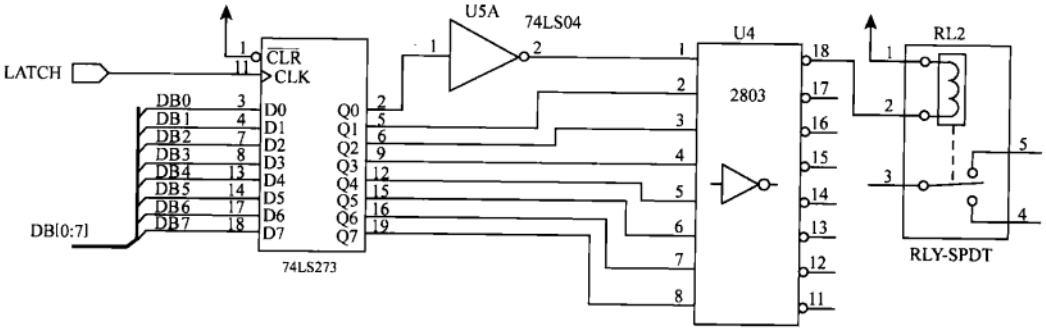
由于整个项目必须在电厂大修前完成，所以我们一完成 PC 板的布局后就立即送 PC 板的制造工厂，进行板子的制造，等到我们拿到板子开始焊接电路板，并且进行测试时，这才发觉线路板的设计上犯了一个大错误。请看我们所公布的类似电路图。图 21-4 中是用 74LS273 当成数据的 LATCH，而 74LS273 的输出再接到 ULN2803 缓冲及电流放大 IC 上，通常我们是用 2803 去驱动发光二极管 LED 或继电器 RELAY，电路上的接法是完全无误的，如果 2803 之后是 LED 问题并不大，可是若是延迟就会有大麻烦的。

请再看电路中的 74LS273 LATCH，其清除引脚（CLR）是固定接到+5V 端，当我们要让某个 RELAY ON 时，就要在 74LS273 的输入信号处放入一个数字 1 的信号，然后在 CLK 端产生一个脉冲信号，那就可以让对应的延迟接通。可是当刚开机的瞬间，电源刚加入时，对 74LS273 的 CLR 引脚而言，却是一个有效的清除（CLEAR）信号，这个信号会使输出全部变成 0。对 ULN2803 而言，输入 0 代表不工作，所以不会启动任何一个延迟。但是另一方面，CLK 引脚在电源接通也要一小段时间后才升为 1，如果 CLK 的信号很不幸地比 CLR 慢一点点时间才升为数字 1 时，那就会把 74LS273 的输入信号直接反应到输出上，由于电源刚加上的那一瞬间，CPU 通常都还没有正式工作，因此，DATA BUS 上都是数字 1 的状态，这将使得 74LS273 的输出全部变成 1，导致我们没有料想到的结果：所有 RELAY 信号全部都为 ON 状态。这可能会使得产生输出信号的校验器短路。虽然程序启动后可以立刻关闭这些 RELAY 信号，但是这个缺陷若不改善，将是设计上的一大败笔，所以更佳的做法是在 74LS273 与 ULN2803 间插入反相闸，以改善开机瞬间的状态。

类似的案件我们也接触了不少，有许多实验室的测量设备原先是用人工来记录的，加上 IEEE488 的接口后，就可以与其他的自动测量仪表结合，完成数据后段的分析整理与统计，只要有相同的应用出现时，我们一定会把“开机瞬间”状态值列入考虑，避开这些问题就是减少系统出状况的机会，这也是公司最宝贵的知识财富了。



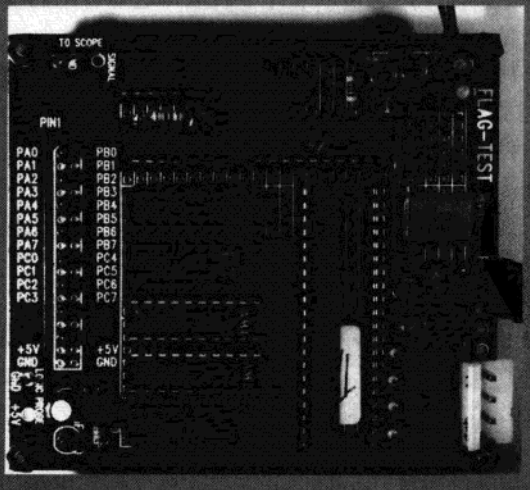
（a）BEFORE



（b）AFTER

图 21-5　未修正前的线路，开机瞬间有可能使延迟全部都 ON。74LS273 后加入反相后，可以使延迟的错误操作降低，但这并不一定是最佳的方法

22



我们试做的 AD 转换板，量产时逻辑电路可以用一枚 ASIC 取代以节省 PC 板面积。

第 22 章　旗威上网了

> 一般人获得知识的来源分别是书本、专业性杂志以及技术研讨会，现在“旗威技术交流网”提供另一种更快速便捷的信息来源。

从 1996 年底开始，旗威科技公司除了正忙于繁重业务外，也积极地筹划上网的事宜，一直有人认为把上网当成一个流行趋势，准备好一些基本资料敷衍一下就行了，这种例子到处都是。某些网站虽然有站长信箱，可是如果您真的寄一封信给他时，千万别期望对方会回信。为什么我会这么悲观呢？这是实际尝试后的真正结果。我们的建议是：除非不上传到 Internet，否则就应该有随时把主页做好的打算。

曾经我们有一段时间内始终从某家计算机厂商进货，可是有一次听到该公司的业务主管提到，他们的 ISO-9000 质量认证打算用钱买到，原来这家公司的质量态度是这样的，我们隔天就随即停止向该公司订货。只因为我们做事与处理的态度绝对不是这样的，而且对于这些产品，我们也不再建议客户购买了。

22-1 旗威技术交流网诞生了

旗威技术交流网的网址是 http://www.chipware.com.tw，我们上网的初步用意是提供一个技术者或正在学习技术者可以相互交流的地方，并且与所有爱好 DIY 的实践派与技术工作者做直接的接触，所以我们把许多单片机的相关数据手册及中文说明都放在这里。当然，其中也包括我们所开发的许多工具程序，您可以立即下载取得这些有用的资料与文件。

一般人获得知识的来源分别是书本、专业性杂志以及技术研讨会，当碰到问题及阻碍时，只有靠以往的实际经验来判断。可是如果您刚好是初学者时，所遭遇的问题往往是最多的。“旗威技术交流网”的用意就是架起经验丰富的老师傅与初学者的桥梁，让已知的经验（具机密性的当然排除在外）能够迅速地传承到初学者上。比如说，要如何收集某方面的资料，材料要到哪里买以及测试设备的使用方法等，如果您要找这方面的数据可能要到图书馆翻遍数年所有的杂志，而您上了旗威的网站后，就可以立即找到一些有效的解答，但是也要经过您再做深入的试验后，才会有最终及最适当的答案。我们一直在强调一个观念：终身学习，如果您一停止学习后，就会迅速被别人超越的。

在进一步规划网站的内容时，我们特别加入了工程师常识区 common sense 与各种测试工具的介绍，这些资料是参考了许多欧美日杂志后所整理出来，我们认为如果您想做个合格的工程师或研发人员时，就必须懂得这些细节。当然您如果还是个学生的话，这些必备知识绝对会给予您正面的帮助。同时我们也会将以前所发表过的文献与 DIY 制作文章做一个总整理，您可以在网络上较不阻塞的时段，download 下载这些文件，先在计算机上观看，真的有需要时再用身边的喷墨或激光打印机打印出来，其结果将和我们所整理与安排的完全一致。要达到这种效果是不能将所有的文件数据变成 HTML 文件的，因为一篇文章经过 HTML 处

理后，在各种浏览器观看时，多少还是会有稍许的差异，但是资料转变成 Adobe 公司的 PDF 文件后，在任何计算机平台（DOS、Windows、Mac 或 UNIX）上的结果却都是相同的。

22-2 PDF 文件是跨平台的

如果您整天做的是设计的工作，身旁一定会有个大书架，上面摆满了各种参考资料与厂商资料，其中一定会有十数本以上各种厂商的 IC 数据资料书，这些数据资料书的体积都是超大号的，每一本的页数可都是超过千页以上，可是我们经常查阅的还是其中的数十页而已！每次新年度一开始后我们都不会忘记向零件商索取新的数据资料书。我们身边还保留 20 世纪 80 年代 TI（德州仪器）的 TTL 数据资料书，虽然已历经数十年的寒暑，但是有些数据还是这里写得最翔实了。不过，又重又厚的数据资料书可能会渐渐式微了，取而代之的是薄薄的 CD 光盘。一张光盘可以放进整套的百科全书，放进十本以上的数据资料书当然不是问题。所以，近几年来已经有不少的 IC 制造厂改用光盘存放所有 IC 的规格特性与使用方法，不仅减少纸张的消耗有环境保护的概念外，还可以非常便捷地邮寄这些资料，以前的数据资料书可能是免费的，但邮资却不是一笔小数目。可是改用光盘后，又如何保持数据与图形的完整呢？各个行业的工程师使用各式各样的计算机上的 CD-ROM 电子文档的确是个大问题，所以计算机排版界的 Adobe 公司便提出了一个 PDF 格式文件的构想，只要我们所编排的图形或数据遵循这个格式转换后，就可以跨越任何计算机平台来读取，Adobe 公司保证所看到与印出的结果和数据转换前是完全相同的，只要想编印数据的厂商向 Adobe 购买 Acrobat 转换程序，转成 PDF 文件的内容后，就可以在任何计算机上使用该公司名叫 Acrobat Reader（目前已改称 Adobe Reader）的程序读取这些 PDF 文件。

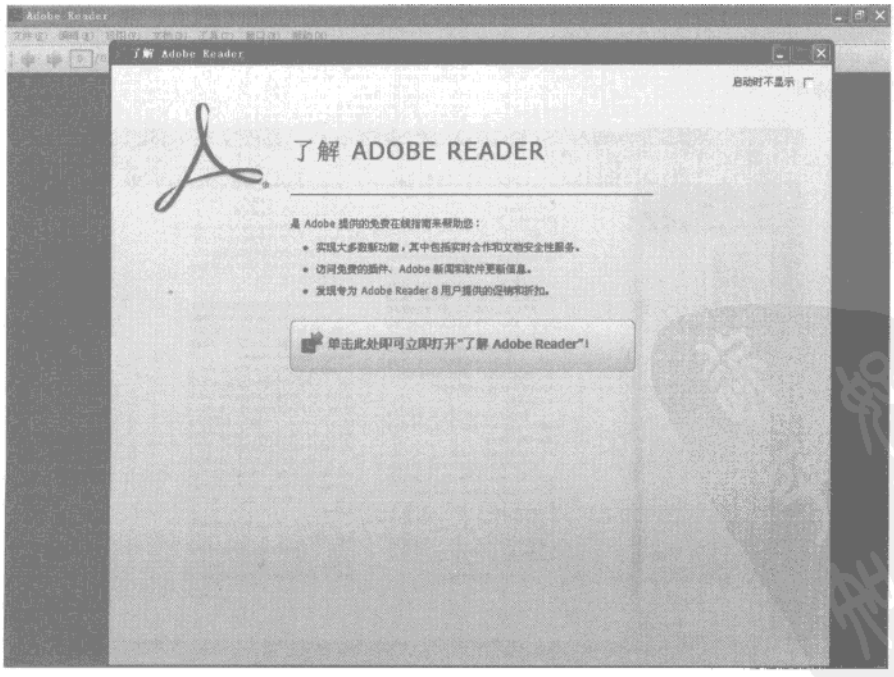


图 22-1 Adobe 公司的 Acrobat，内部包含了 PDF 的转换程序及阅读程序

Acrobat Reader 程序可以在 Adobe 公司的网站（http://www.adobe.com）上免费下载，您可以依个人需要下载 DOS 版、WINDOWS 版、MAC 版或 UNIX 版的 Acrobat Reader。据我们的了解，世界级的 IC 大厂如 TI、NS、Harris 及 Sony 都已改用光盘推出数据手册，每季都推出更新版的光盘，这些有环保概念无纸的 Databook 应该可以改称为 Datadisk，其间所有的资料格式正是 ADOBE 的 PDF 文档。

图 22-2　各式各样的 IC Databook 是工程师的工具书

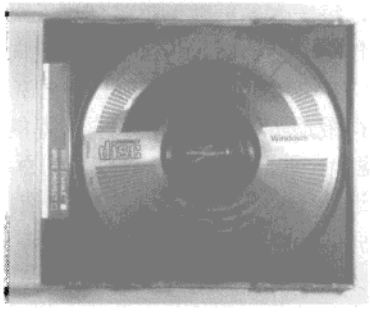


图 22-3　1996 年 9 月的日文 transister 杂志附赠的 Datadisk，内置多家厂商的 IC 的 PDF 文件资料

在旗威科技公司的网站里，如果是新的资料我们会先用 HTML 的格式让所有的网友直接用浏览程序观看，这时每个章节段落也许不会很恰当，经过一段时间后，我们会再度将原稿用 PageMaker 编排图文后再改成 PDF 文件的格式，此时文件一定会比较大。您可以从旗威科技公司的网站直接下载此 PDF 文件后，用 Acrobat Reader 观看，所有的图文及照片都在一定的水准之上，真的有需要时才保留，若还是觉得看之无味且无保留价值时，就直接把它删除掉吧。

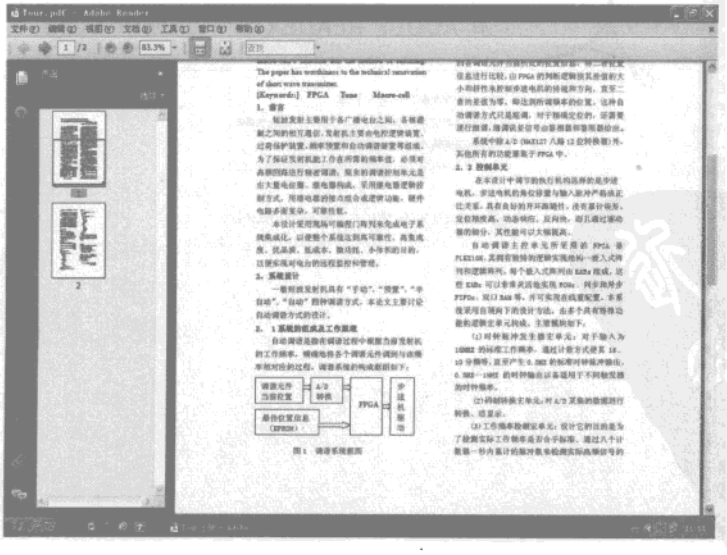


图 22-4　PDF 的技术资料可以在各种电脑平台上观看

有一年年底我们向 Adobe 的代理商购买这套软件时，却发生了一小段插曲：我们询问好了价格寄上了现金支票后，经过十数天，仍未收到 Acrobat 套装软件，经过数次的电话查询后，确认是有收到钱但是旗威的邮寄地址不见了！又过了数天之后才在当年的最后一天（12 月 31 日）拿到 Acrobat 套装软件，如是预先知道会是这种二流的服务，我们老早就直接跟 Adobe 原厂订了。

这些软件代理商若无法做服务态度的改善时，迟早会被客户拒绝的。旗威科技公司愿意为每一位客户提供周到的服务，数年前有位香港的读者传真 FAX 埋怨说：买了 AT89C51 烧录器之后，才发现还要有一块单片机控制板 FLAG51 才能工作，他认为我们应该在广告上载明清楚才对。当我们辗转由旗标出版公司收到这封信时，马上向该位读者发出传真致歉函，在问清楚这位读者的确实地址后，随即补寄一块单片机控制板给这位读者，当然稍后的广告用语也多加了一些更正及说明，我们认为此时的商誉及信用绝对比一块 FLAG51 控制板的价值还要重要。

22-3 E-mail 处理及回复的原则

每个月旗威科技公司从电子信箱（chipware@seed.net.tw）中收到一百多封电子邮件，除非是无暇兼顾的非常时期，否则我们都会在一周以内做答复。经过两年来也累积了近五百封左右的问题与答复（Q&A），我们会把这些问题集锦（FAQ）分门别类后放置在网站上供需要的人查询。如果您有自动化设计或任何单片机应用的问题时，请把问题说明清楚，并且记载您已经做过哪一方面的尝试，再把信件传递给旗威科技公司，新启用的电子信箱为 chipware@seed.net.tw，我们会依照问题的种类在站上公开回复。

“自助者人恒助之”，假使您的问题是不假思索地问“步进电动机怎么使用？”，我想保证是不会有人答复的，因为你都放弃了，我们还怎么帮助你呢？如果您问的问题本身有些敏感性，希望私底下答复时，请将信件传到我们原来种子网络 SEEDNET 的电子信箱 chipware@seed.net.tw 中，并在信中特别注明私下答复。

如果您还没有机会使用计算机接上 Internet 时，用一般信函寄给我们也是可以的，来信请寄到高雄市 807 三民区昌裕街 18-1 号。不过，我们还是希望您能尽快把这条获得知识的通路打通，因为唯有这样您才能在最短的时间内，取得各方面第一手的技术资料。“知识就是力量”这句话的含义您应该懂吧！至少可以 download 一些有用的资料来看看。

旗威网站现阶段将用户的问题归成以下数大类：

（1）单片机 8051 的应用：8051 的各种应用及疑难解答都在这里。

（2）自动化设备的设计：公司有许多设备稍加修改后就可以自动化，怎么做呢？这里有高手指点。

（3）测试仪器的使用：技术性的设备购买与使用时，有哪些要特别注意的。

（4）Chipware-PLC 的使用：最新版的程序欢迎下载使用，还有我们所有测试程序范例可供参考。

（5）机电集或应用：步进与伺服电动机的使用，各种机构如何设计等等都有专家在旁指导。

（6）FPGA 与 VHDL 的使用：想对数字电路设计更上一层楼时，可以学习新一代的可编程新元件 FPGA。

其实，现有的学术网络 TANET 中的 BBS 站也有上面几项的应用讨论组，我们想利用旗威网站做脱离学术圈且更深入的探讨。可能的也会保留部分精华的内容，也唯有这样的态度与处理，属于我们的各种技术才能源源不绝地传递下去。网站上的独特法是旗威科技公司一直要做的理想，至于公司能否获利，那就要看几年之后了。旗威网站上或许没有漂亮的图表及吸引人的 Java 动画，但是内容绝对是您不可错过的。最后我们再度将网址公布一次：网址：http://www.chipware.com.tw，电子邮件信箱为 chipware@seed.net.tw，我们欢迎您随时上网查询。

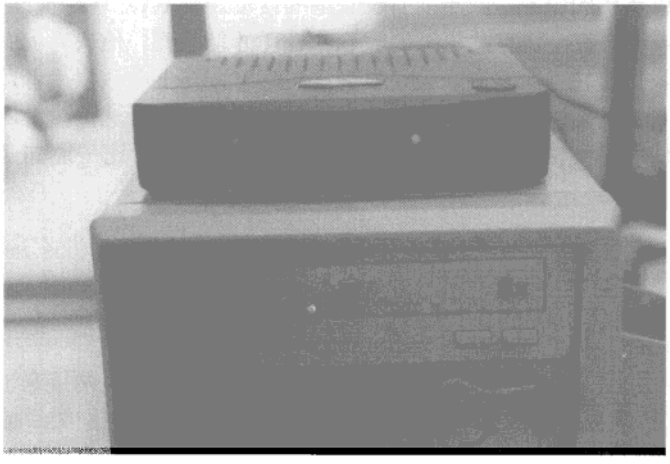
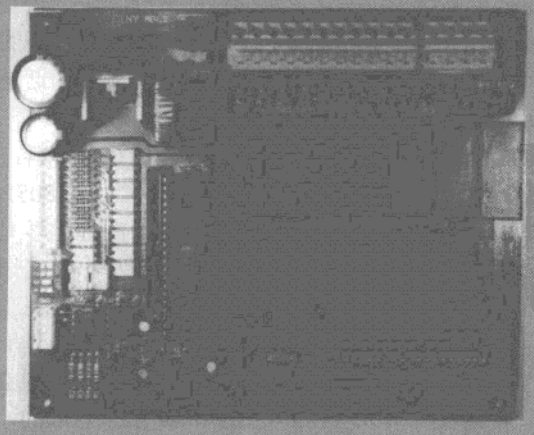


图 22-5　计算机加上调制解调器就可以取得许多第一手的资料

23



医疗仪器用的 RELAY 控制板，电路与程序上的考量才是重点。

第 23 章　当然你也可以做到

> 当我们遇到困难瓶颈时，总会期待别人适时的帮忙，纵使没有任何帮忙时，还是有人会顺利地度过难关，这靠的是我们面对困难时的态度。悲观的人总是看到事情的种种障碍，乐观的人看到的却是困难之后的机会。

23-1　北部与南部理应无技术上的差距

每当有人在埋怨学习芯片 8051 学得很辛苦时，我总会举下面这个例子来鼓励初学者：这是好几年前的事了，我曾经在高雄授课，教的是单片机 8051 的程序设计与硬件彻底剖析，上课的对象清一色都是工程师，每周上课两次三小时，在众多听众中有两位是来自台北及基隆，这两位同学的年龄都超过 30 岁，为了听课他们每周有两天必须在中午就搭车南下，听完课后约在午夜又搭车北上，回到家里应该是隔天清晨吧！这种努力的学习态度持续了三个半月之久，直到全部课程结束为止。所以，别人都这样卖力地学习，你这点辛苦算什么！事后我问这两位远从北部来的同学："北部应该也有类似的教学课程，何以一定要到南部学习呢？"他们的回答是："很少有老师这样毫无保留地将程序设计的知识传承给学生，而林老师您就是其中一位，如果您愿意到台北开课的话，保证一定会爆满的。"

话虽如此，我还是喜欢有空时到台北走走，顺便拜访一些老朋友。其实在信息发达的现在，北部与南部的各方面差距已愈来愈小了，许多元件厂商的资料打个电话后隔日就可以拿到了，而且 Internet 通行无阻后，从计算机终端机上都可取得世界各地最新的文献资料，这时身在何方已经不是最重要的问题了。去年农历年假初五晚上十一点我从台中太太的娘家回高雄时，路经高速公路员林收费站时发现一个非常怪异的现象，南下的车只有我们一辆，北上却是满满大排长龙的车队，这种塞车的社会成本如果仔细精算的话是相当高的，我很庆幸当初"回到南部"的决定是对的。

最近，旗威科技公司抽空整理了芯片控制板 FLAG51 的相关资料，从《8051 单芯片彻底研究》一书问世之后，至今已超过十年的时间，从我们手中卖出的 8051 控制板不知不觉地超过了五千套。在众多的使用者中，我们发现拿来做实际应用的远比学习的多，而使用者的身份又以工程师级与教师级的居多，大专院校采购的数量也不少。我们知道也有部分单芯片控制板卖到香港及东南亚各地，大陆当然也有不少台商的订单，这代表当初我们所花下的苦心是绝对有所回馈的。

1997 年年初旗威科技公司的网站（www.chipware.com.tw）已经成立，许多使用者的问题都可以通过 E-mail 答复，使用者与旗威科技公司的距离是愈来愈近的，我们也把常见的问题整理成 FAQ，您可以从网站上立即找到许多问题的解答。在 1997 年内我们就已把有关模拟转数字电路（AD 与 DA）的产品纳入，这样一来您就可以用 8051 做各种模拟或数字的应用

了。最近旗威科技公司拟扩大业务规模，打算找数字助理工程师加入开发的行列，如果您是南部对单片机相当用功的新人，特别相当认同 DIY 的理念，请抽空来找我们谈谈。

23-2 马上学习单片机都来得及

也许有人会问：8051 单片机现在学还来得及吗？我们的答复是绝对来得及的，Intel 的 8051 开发至今已有十多年的历史了，但是五六年前才在台湾被大量采用。旗威科技公司用 8051 CPU 陆续完成至少五十种产品的开发与量产，其中的 FLAG51 只是其中一个学习应用而已。近几年来 8051 的 second source 如 Dallas、Atmel 或 Philips 都已经开发出功能较 8051 还要强的兼容芯片，只要您有 8051 相关的 Assembler 或 C compiler，就可以顺利将程序码移植到这些兼容芯片上。可是，如果您要好好地应用这些芯片时，重新好好地复习一下 Data Sheet 是不少了的。

以 Atmel 的 AT89C51、AT89C52 为例，我们可以将石英晶体的频率提高到 24MHz，不过如果要继续做串行通信时，有部分设定参数是要再做修改的。而 Dallas 的 80C320 与 AT89S8252 的内部有两组 DPTR，比起原有的 8051 在存取数据时会更有效率。若不看相关资料时是不知道如何使用的。这些与 8051 兼容的 CPU 都有特定的优点，也只有亲自动手试一下，才能更深一步体会其中的奥秘所在。这些最新资料都可以通过 Internet 到制造厂商的网站上取得，而正式的样品就需要找本地的代理商才要得到。由此看来，8051 系列的单片机还是有其特定的存在空间，就像 Z80 CPU 老兵不死一样，至少在五年内 8051 CPU 还是会有发展的空间的，所以立即学习 8051 绝对是正确的，怕的就是你只是看看书而少了亲自动手。

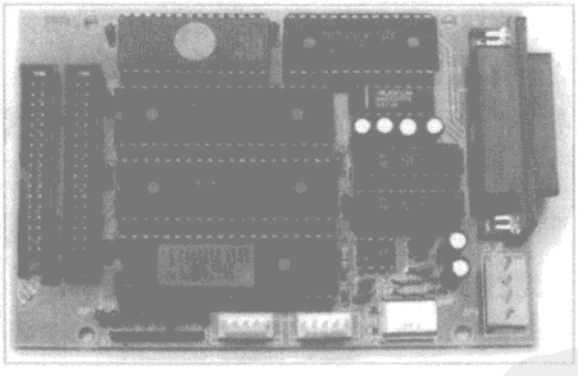
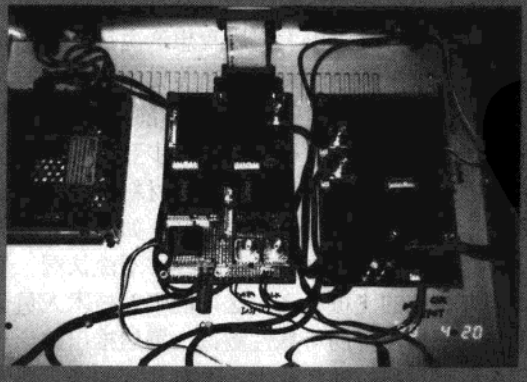


图 23-1 FLAG51 单机片控制板的应用是没有限制的

24



许多数字电路的设计，以今天的方式来做时，大部分都可以用 FPGA 或 CPLD 来模拟或验证。

第 24 章　自主性研究的重要

> 一项技术如果是买来的，对买方而言，永远希望是最先进。但对卖方而言，既然是可割让的，就称不上是先进。这中间的落差，值得我们深思。

24-1　企业引进技术所浮现的问题

目前在美国爆发了某公司涉嫌窃取生物科技的工业间谍案，双方的说词当然是南辕北辙！我们认为这只是众多技术引进所衍生的问题之一，还有许多问题尚未浮出台面。想了解一个地方的竞争实力，看看那儿的专业书籍或杂志就可以得知其端倪了。如果翻译性的文章过多泛滥时，该地区八成是技术学习为主，而且自主性的研发一定偏低。反之，假使创作性文章颇多时，这绝对是值得我们一直学习的对象。我们认为现阶段仍是处于学习的阶段，无法像先进地区可以做整厂或是完整技术的输出。

我们不知道企业对技术引进的态度是否只着重于眼前的重大利益，为了一项技术而付费是否只顾虑到短期间就要获利，所以如果“用带的”就带回，要不然就用买的，再不行时就买下整个公司，以便能大量投料生产，可是这是企业的唯一生存之道吗？技术人员认为是一值得思索的好问题。

24-2　技术是长年的积累，别人是学不来的

如果您到垦丁渡假的话，应该会路过位于高雄小港的中钢公司，假使时间允许的话，我们建议您绕道一下到公司进出口大门参观一下，您会发现一个很奇怪的场景：所有骑摩托车的人都会在门口处停车，然后将车子缓慢地推过警卫室，步行过一小段距离后才又骑上摩托车，所有骑摩托车的人都是以这种方式进出大门。我想这说是中钢特有的行事态度，别的单位是很难照学的。而中钢却有二十来岁的厂龄，比起美日历史悠久的大炼钢厂还真的是差上一大截，因为这些钢铁厂里长年积累了绝对无法学习的经验与技术，要想到这个地步除了不断地自我超越外，还是要时间慢慢地培育才行，想要一步登天几乎是不可能的。技术人员曾经到航发中心拜访，该中心的工程师取出一大叠的飞机材料测试报告给我们看，我们认为除了人之外，这些数据及报告才是航发中心最重要的资产，如果国外的飞机制造厂直接就指定用某种材料，我们就无从得知其缘由了。这也是我们经常不能认同一般的重大投资消息：今年成立航空工业，明年就会有自产的飞机起飞，这可能吗？技术人员认为这肯定是非常冒险的做法，而且种飞机尚未完成所有飞行的可靠性测试，冒然推出的话一定会出问题的。

相同的原理也可以用在所有高科技的产业上，没有几年的辛苦研发是上不了台面的。旗威科技公司最近几年在 8051 单片机以及 PLC 控制程序方面积累了不少软硬件技术，我们的线路及程序有相当的稳定性及信赖度，同时我们也很乐意将相关的知识传播给大家，技术人

员认为除非对手也花下相同的心血，否则旗威科技公司在这方面绝对是有稍许的领先优势。

近几年来，旗威科技公司也接了一些相当令人兴奋的计划，比方说：电动汽车的部分计划，我们这才发现有些人可真是财大气粗，可以花下数千万去国外订购一台电动车的原型机（Prototype），但是投注研究经费及人力却少得可怜，而且买了原型机也不意谓就直接等于可量产的机种，而所谓的研发人员大部分都是“外形设计”人员，内部的核心技术开发与设计人员却不多。当事者以为国外的月亮一定是圆的，因此电动车所有的技术 Know-how 都可以靠钱买到，所以大笔的钱源源不绝地到了国外，但是却忽略了别人研究的精神及态度却是花再多钱也买不到的，因为这些是无法存在于所有的说明文件上。许多企业主到现在还是认为：只要到国外买回最先进的生产设备，再投入生产原料就可以保有领先的地位，有些部分原料产品用这个方法是可行的，可是有更多有原创性的高科技产品却永远无法出现。

只要朋友到国外出差回来后，我总会提一个问题：我们与别人的先进技术水平差多少年？得到的信息通常是：通信技术上至少有十年以上的差距，其他具原创性的科技如生物科技及太空科技则差距更大，在交通方面也远远地落后。这是因为我们有许多重大建设或工程都直接由国外引进，花最多的钱引进最新的设备，但是本身的研发技术水平却毫没有相对提升。捷运系统却是一个特殊案例，在法国马特拉撒手不管之后，反而培养出一批“逆向工程”的程序及线路分析好手。其实，做基础的研发人员还是有许多发挥的空间！

24-3 研究与创意的始源在于教育

我有一位博士朋友曾经在美国的 NASA 太空总署工作多年，谈到与美国博士们间的差异时，他总会说：奇怪，别人的创意是比我的还多？接着我们总会提到：原来都是我们的教育制度在做怪，这是因为我们的教育制度都是填鸭式的，迫使学生普遍变成考试的工具，学生被教导成碰到什么问题就用什么方式解题，长久下来，当然所有的创意就全部被磨灭掉了，进入社会后当然就不会有所发挥了。其实，相同的情况日本也是有相同的隐忧，问题也是出在教育上。

技术人员在校的成绩也是平平的，距离大学毕业当年已经超过 15 个年头了，我也忘掉当年大部分考试的成绩，可是我却永远记得一件事情：有一次的数字电路实习作业，我针对其中一个电路设计题目，总共用七种方法去尝试实验，结果都失败了，最后第 8 次才正式解出来。我把这些失败的电路呈现在报告中，也特别在作业最后注明：这只是其他一个较正式的解答而已，应该还有其他的方法才对。当时教授数字电路的老师从头到尾仔细地看完了我的说明，并且在作业的第一页上打了满分 100 分。这件事情给我一个相当大的启发：大部分的事情都没有绝对的答案，所有事情只要我们曾经努力过，绝对是有影响力的。所以踏入社会后，不论是写程序或设计硬件电路，我们一定先找齐所有的参考资料，并且经过多次的深思之后，才开始下手处理。我们会参考别人的程序或电路，但我们绝对不会抄袭，因为抄袭无法抄到原创性。碰到任何问题时，我们会把所有可能的问题都列出来，然后用各种方法证明问题是否存在，并且一一过滤，最后都没法解答时才求助，不过，经过我们这种方法思考后，实际需要求助的案例是少之又少了。这也就是我们经常在杂志或是书上呼吁的：请把问题先自行消化后才提出来，否则谁有义务去回答呢？如果我们的教育制度还是硬要区分优等生与劣等生时，受害的一定是现在的学生，何不让学生有一片无障碍可思考及完成掌握的空间呢？技术人员有时会去想：如果我是出生在美国或是法国，20 年成长并接受教育后会变成

哪一种样子呢。

24-4 提升技术最后还是在于自己

一项技术如果是买来的，对出大钱的购买者而言，永远希望是最先进的，但是对技术输出者而言，这项技术应该是属于可割让的，所以已经不能算是最先进的。当然购买者可以要求备妥各种技术文件，但这些文件资料的背后，还有许多许多资产是无法用钱来购买及衡量的。有人说：21 世纪是中国人的世纪，我们一定相信：除了生产计算机主板赚取微薄的 5% 利润以外，还是有许多发展的舞台存在。当我们的企业主正在全球各地引进、吞并或购买任何 Know-how 时，也许可以停下来思考一下，自己来做或许不一定会成功，但是用其他的方式绝对不会是使企业根基牢固的唯一方法。

附　录

不一样的书给你不一样的感觉，请查阅经过我们妥善整理的 8051 相关资料，本书的附录仍颇有参考的价值。

附录 A　ASCII 表

	0	1	2	3	4	5	6	7
0	NUL 00 0	DEL 10 16	SP 2D 32	0 30 48	@ 40 64	P 50 80	' 60 96	p 70 112
1	SOH 01 1	DC1 11 17	! 21 33	1 31 49	A 41 65	Q 51 81	a 61 97	q 71 113
2	STX 02 2	DC2 12 18	" 22 34	2 32 50	B 42 66	R 52 82	b 62 98	r 72 114
3	ETX 03 3	DC3 13 19	# 23 35	3 33 51	C 43 67	S 53 83	c 63 99	s 73 115
4	EOT 04 4	DC4 14 20	$ 24 36	4 34 52	D 44 68	T 54 84	d 64 100	t 74 116
5	ENQ 05 5	NAK 15 21	% 25 37	5 35 53	E 45 69	U 55 85	e 65 101	u 75 117
6	ACK 06 6	SYN 16 22	& 26 38	6 36 54	F 46 70	V 56 86	f 66 102	v 76 118
7	BEL 07 7	ETB 17 23	' 27 39	7 37 55	G 47 71	W 57 87	g 67 103	w 77 119
8	BS 08 8	CAN 18 24	(28 40	8 38 56	H 48 72	X 58 88	h 68 104	x 78 120
9	HT 09 9	EM 19 25	) 29 41	9 39 57	I 49 73	Y 59 89	i 69 105	y 79 121
A	LF 0A 10	SUB 1A 26	* 2A 42	: 3A 58	J 4A 74	Z 5A 90	j 6A 106	z 80 122
B	VT 0B 11	ESC 1B 27	+ 2B 43	; 3B 59	K 4B 75	[5B 91	k 6B 107	{ 7A 123
C	FF 0C 12	FS 1C 28	, 2C 44	< 3C 60	L 4C 76	\ 5C 92	l 6C 108	| 7B 124
D	CR 0D 13	GS 1D 29	- 2D 45	= 3D 61	M 4D 77	] 5D 93	m 6D 109	} 7C 125
E	SO 0E 14	RS 1E 30	. 2E 46	> 3E 62	N 4E 78	^ 5E 94	n 6E 110	~ 7D 126
F	SI 0F 15	US 1F 31	/ 2F 47	? 3F 63	O 4F 79	_ 5F 95	o 6F 111	DEL 7F 127

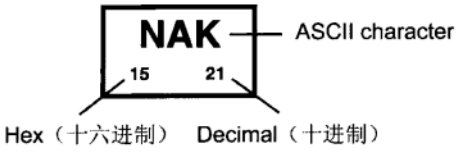
高位

低位

Example

NAK

HEX　DECIMAL



如何利用本表格

如果我们收到了 30H 这个十六进制码时，代表何种意义呢？请先查列 3，再查行 0，两条线交叉刚好是 0（数字 0），这表示我们收到的是一个数字。如下表所示。许多可与计算机连接的仪器设备由于只传输数据，所以送回的值都介于 30H～39H 间，查 ASCII 表的结果，刚好就是数字 0～9。另外，如果我们送一个 45H 给 PC 时，PC 屏幕上就会显示出 E 的字样。

step 1

step 2	0	1	2	3	4	5	6	7
0	NUL 00 0	DEL 10 16	SP 2D 32	0 30 48	@ 40 64	P 60 80	' 60 96	p 70 112
1	SOH 01 1	DC1 11 17	! 21 33	1 31 49	A 41 65	Q 51 81	a 61 97	q 71 113
2	STX 02 2	DC2 12 18	" 22 34	2 32 50	B 42 66	R 52 82	b 62 98	r 72 114
3	ETX 03 3	DC3 13 19	# 23 35	3 33 51	C 43 67	S 53 83	c 63 99	s 73 115
4	EOT 04 4	DC4 14 20	$ 24 36	4 34 52	D 44 68	T 54 84	d 64 100	t 74 116
5	ENQ 05 5	NAK 15 21	% 25 37	5 35 53	E 45 69	U 55 85	e 65 101	u 75 117
6	ACK 06 6	SYN 16 22	& 26 38	6 36 54	F 46 70	V 56 86	f 66 102	v 76 118
7	BEL 07 7	ETB 17 23	' 27 39	7 37 55	G 47 71	W 57 87	g 67 103	w 77 119
8	BS 08 8	CAN 18 24	(28 40	8 38 56	H 48 72	X 58 88	h 68 104	x 78 120
9	HT 09 9	EM 19 25	) 29 41	9 39 57	I 49 73	Y 59 89	i 69 105	y 79 121
A	LF 0A 10	SUB 1A 26	* 2A 42	: 3A 58	J 4A 74	Z 5A 90	j 6A 106	z 80 122

附录 B　8051 指令集总整理

影响标志位的指令整理

Instruction	Flag		
	Carry	Overflow	Aux Carry
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
NUL	0	X	
DIV	0	X	
DAA	X		
RRC	X		
RLC	X		
SETB C	1		
CLR C	0		
CPL C	X		
ANL C, bit	X		
ANL C, /bit	X		
ORL C, bit	X		
ORL C, /bit	X		
MOV C, bit	X		
CJNE	X		

注　SFR 字节地址 208 或位地址 209～215（即 PSW 或 PSW 中的位）的操作也会影响标志位设置。

指令集与寻址模式

Rn	Register R7-R0 of the currently selected Register Bank.
direct	8-bit internal data location's address. This could be an Internal Data RAM location (0-127) or a SFR[i.e., I/O port, control register, status register, etc. (128-255)].
@RI	8-bit internal data RAM location (0-255) addressed indirectly through register R1 or R0.
#data	8-bit constant included in instruction.
#data 16	16-bit constant included in instruction.
addr 16	16-bit destination address. Used by LCALL and LJMP. A branch can be anywherre within the 64K byte Program Memory address space.
addr 11	11-bit destination address. Used by ACALL and AJMP. THe branch will be within the same 2K byte page of program memory as the first byte of the following instruction.
rel	Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is -128 to +127 bytes relative to first byte of the following instruction.
bit	Direct Addressed bit in Internal Data RAM or Special Function Register.

8051 指令集汇总

	0	1	2	3	4	5	6	7
0	NOP	JBC bit, rel [3B, 2C]	JB bit, rel [3B, 2C]	JNB bit, rel [3B, 2C]	JC rel [2B, 2C]	JNC rel [2B, 2C]	JZ rel [2B, 2C]	JNZ rel [2B, 2C]
1	AJMP (P0) [2B, 2C]	ACALL (P0) [2B, 2C]	AJMP (P1) [2B, 2C]	ACALL (P1) [2B, 2C]	AJMP (P2) [2B, 2C]	ACALL (P2) [2B, 2C]	AJMP (P3) [2B, 2C]	ACALL (P3) [2B, 2C]
2	LJMP addr16 [3B, 2C]	LCALL addr16 [3B, 2C]	RET [2C]	RETI [2C]	ORL dir, A [2B]	ANL dir, A [2B]	XRL dir, A [2B]	ORL C, bit [2B, 2C]
3	RR A	RRC A	RL A	RLC A	ORL dir, #data [3B, 2C]	ANL dir, #data [3B, 2C]	XRL dir, #data [3B, 2C]	JMP @A + DPTR [2C]
4	INC A	DEC A	ADD A, #data [2B]	ADDC A, #data [2B]	ORL A, #data [2B]	ANL A, #data [2B]	XRL A, #data [2B]	MOV A, #data [2B]
5	INC dir [2B]	DEC dir [2B]	ADD A, dir [2B]	ADDC A, dir [2B]	ORL A, dir [2B]	ANL A, dir [2B]	XRL A, dir [2B]	MOV dir, #data [3B, 2C]
6	INC @R0	DEC @R0	ADD A, @R0	ADDC A, @R0	ORL A, @R0	ANL A, @R0	XRL A, @R0	MOV @R0, @data [2B]
7	INC @R1	DEC @R1	ADD A, @R1	ADDC A, @R1	ORL A, @R1	ANL A, @R1	XRL A, @R1	MOV @R1, #data [2B]
8	INC R0	DEC R0	ADD A, R0	ADDC A, R0	ORL A, R0	ANL A, R0	XRL A, R0	MOV R0, #data [2B]
9	INC R1	DEC R1	ADD A, R1	ADDC A, R1	ORL A, R1	ANL A, R1	XRL A, R1	MOV R1, #data [2B]
A	INC R2	DEC R2	ADD A, R2	ADDC A, R2	ORL A, R2	ANL A, R2	XRL A, R2	MOV R2, #data [2B]
B	INC R3	DEC R3	ADD A, R3	ADDC A, R3	ORL A, R3	ANL A, R3	XRL A, R3	MOV R3, #data [2B]
C	INC R4	DEC R4	ADD A, R4	ADDC A, R4	ORL A, R4	ANL A, R4	XRL A, R4	MOV R4, #data [2B]
D	INC R5	DEC R5	ADD A, R5	ADDC A, R5	ORL A, R5	ANL A, R5	XRL A, R5	MOV R5, #data [2B]
E	INC R6	DEC R6	ADD A, R6	ADDC A, R6	ORL A, R6	ANL A, R6	XRL A, R6	MOV R6, #data [2B]
F	INC R7	DEC R7	ADD A, R7	ADDC A, R7	ORL A, R7	ANL A, R7	XRL A, R7	MOV R7, #data [2B]

注　[2B]=2 Byte，[3B]=3Byte，[2C]=2Cycle，[4C]=4 Cycle，Blank=1 byte/1 cycle。

续表

	8	9	A	B	C	D	E	F
0	SJMP REL [2B, 2C]	MOV DPTR, #data 16 [3B, 2C]	ORL C, /bit [2B, 2C]	ANL C, /bit [2B, 2C]	PUSH dir [2B, 2C]	POP dir [2B, 2C]	MOVX A, @DPTR [2C]	MOVX @DPTR, A [2C]
1	AJMP (P4) [2B, 2C]	ACALL (P4) [2B, 2C]	AJMP (P5) [2B, 2C]	ACALL (P5) [2B, 2C]	AJMP (P6) [2B, 2C]	ACALL (P6) [2B, 2C]	AJMP (P7) [2B, 2C]	ACALL (P7) [2B, 2C]
2	ANL C, bit [2B, 2C]	MOV bit, C [2B, 2C]	MOV C, bit [2B]	CPL bit [2B]	CLR bit [2B]	SETB bit [2B]	MOVX A, @R0 [2C]	MOVX @R0, A [2C]
3	MOVC A, @A + PC [2C]	MOVC A, @A + DPTR [2C]	INC DPTR [2C]	CPL C	CLR C	SETB C	MOVX A, @RI [2C]	MOVX @RI, A [2C]
4	DIV AB [2B, 4C]	SUBB A, #data [2B]	MUL AB [4C]	CJNE A, #data, rel [3B, 2C]	SWAP A	DA A	CLR A	CPL A
5	MOV dir, dir [3B, 2C]	SUBB A, dir [2B]		CJNE A, dir, rel [3B, 2C]	XCH A, dir [2B]	DJNZ dir, rel [3B, 2C]	MOV A, dir [2B]	MOV dir, A [2B]
6	MOV dir, @R0 [2B, 2C]	SUBB A, @R0	MOV @R0, dir [2B, 2C]	CJNE @R0, #data, rel [3B, 2C]	XCH A, @R0	XCHD A, @R0	MOV A, @R0	MOV @R0, A
7	MOV dir, @R1 [2B, 2C]	SUBB A, @R1	MOV @R1, dir [2B, 2C]	CJNE @R1, #data, rel [3B, 2C]	XCH A, @R1	XCHD A, @R1	MOV A, @R1	MOV @R1, A
8	MOV dir, R0 [2B, 2C]	SUBB A, R0	MOV R0, dir [2B, 2C]	CJNE R0, #data, rel [3B, 2C]	XCH A, R0	DJNZ R0, rel [2B, 2C]	MOV A, R0	MOV R0, A
9	MOV dir, R1 [2B, 2C]	SUBB A, R1	MOV R1, dir [2B, 2C]	CJNE R1, #data, rel [3B, 2C]	XCH A, R1	DJNZ R1, rel [2B, 2C]	MOV A, R1	MOV R1, A
A	MOV dir, R2 [2B, 2C]	SUBB A, R2	MOV R2, dir [2B, 2C]	CJNE R2, #data, rel [3B, 2C]	XCH A, R2	DJNZ R2, rel [2B, 2C]	MOV A, R2	MOV R2, A
B	MOV dir, R3 [2B, 2C]	SUBB A, R3	MOV R3, dir [2B, 2C]	CJNE R3, #data, rel [3B, 2C]	XCH A, R3	DJNZ R3, rel [2B, 2C]	MOV A, R3	MOV R3, A
C	MOV dir, R4 [2B, 2C]	SUBB A, R4	MOV R4, dir [2B, 2C]	CJNE R4, #data, rel [3B, 2C]	XCH A, R4	DJNZ R4, rel [2B, 2C]	MOV A, R4	MOV R4, A
D	MOV dir, R5 [2B, 2C]	SUBB A, R5	MOV R5, dir [2B, 2C]	CJNE R5, #data, rel [3B, 2C]	XCH A, R5	DJNZ R5, rel [2B, 2C]	MOV A, R5	MOV R5, A
E	MOV dir, R6 [2B, 2C]	SUBB A, R6	MOV R6, dir [2B, 2C]	CJNE R6, #data, rel [3B, 2C]	XCH A, R6	DJNZ R6, rel [2B, 2C]	MOV A, R6	MOV R6, A
F	MOV dir, R7 [2B, 2C]	SUBB A, R7	MOV R7, dir [2B, 2C]	CJNE R7, #data, rel [3B, 2C]	XCH A, R7	DJNZ R7, rel [2B, 2C]	MOV A, R7	MOV R7, A

注　[2B]=2 Byte，[3B]=3Byte，[2C]=2Cycle，[4C]=4 Cycle，Blank=1 byte/1 cycle。

8051 指令集——单字节指令整理 1

	0	1	2	3	4	5	6	7
0	NOP							
1								
2			RET [2C]	RET [2C]				
3	RR A	RRC A	RL A	RLC A				JMP @A+DPTR [2C]
4	INC A	DEC A						
5								
6	INC @R0	DEC @R0	ADD A, @R0	ADDC A, @R0	ORL A, @R0	ANL A, @R0	XRL A, @R0	
7	INC @R1	DEC @R1	ADD A, @R1	ADDC A, @R1	ORL A, @R1	ANL A, @R1	XRL A, @R1	
8	INC R0	DEC R0	ADD A, R0	ADDC A, R0	ORL A, R0	ANL A, R0	XRL A, R0	
9	INC R1	DEC R1	ADD A, R1	ADDC A, R1	ORL A, R1	ANL A, R1	XRL A, R1	
A	INC R2	DEC R2	ADD A, R2	ADDC A, R2	ORL A, R2	ANL A, R2	XRL A, R2	
B	INC R3	DEC R3	ADD A, R3	ADDC A, R3	ORL A, R3	ANL A, R3	XRL A, R3	
C	INC R4	DEC R4	ADD A, R4	ADDC A, R4	ORL A, R4	ANL A, R4	XRL A, R4	
D	INC R5	DEC R5	ADD A, R5	ADDC A, R5	ORL A, R5	ANL A, R5	XRL A, R5	
E	INC R6	DEC R6	ADD A, R6	ADDC A, R6	ORL A, R6	ANL A, R6	XRL A, R6	
F	INC R7	DEC R7	ADD A, R7	ADDC A, R7	ORL A, R7	ANL A, R7	XRL A, R7	

注　[2B]=2 Byte，[3B]=3Byte，[2C]=2Cycle，[4C]=4 Cycle，Blank=1 byte/1 cycle。

8051 指令集——单字节指令整理 2

	8	9	A	B	C	D	E	F
0							MOVX A, @DPTR [2C]	MOVX @DPTR, A [2C]
1								
2							MOVX A, @R0 [2C]	MOVX @R0, A [2C]
3	MOVC A, @A + PC [2C]	MOVC A, @A + DPTR [2C]	INC DPTR [2C]	CPL C	CLR C	SETB C	MOVX A, @RI [2C]	MOVX @RI, A [2C]
4			MUL AB [4C]		SWAP A	DA A	CLR A	CPL A
5								
6		SUBB A, @R0			XCH A, @R0	XCHD A, @R0	MOV A, @R0	MOV @R0, A
7		SUBB A, @R1			XCH A, @R1	XCHD A, @R1	MOV A, @R1	MOV @R1, A
8		SUBB A, R0			XCH A, R0		MOV A, R0	MOV R0, A
9		SUBB A, R1			XCH A, R1		MOV A, R1	MOV R1, A
A		SUBB A, R2			XCH A, R2		MOV A, R2	MOV R2, A
B		SUBB A, R3			XCH A, R3		MOV A, R3	MOV R3, A
C		SUBB A, R4			XCH A, R4		MOV A, R4	MOV R4, A
D		SUBB A, R5			XCH A, R5		MOV A, R5	MOV R5, A
E		SUBB A, R6			XCH A, R6		MOV A, R6	MOV R6, A
F		SUBB A, R7			XCH A, R7		MOV A, R7	MOV R7, A

注　[2B]=2 Byte，[3B]=3Byte，[2C]=2Cycle，[4C]=4 Cycle，Blank=1 byte/1 cycle。

8051 指令集——双字节指令整理 1

	0	1	2	3	4	5	6	7
0					JC rel [2B, 2C]	JNC rel [2B, 2C]	JZ rel [2B, 2C]	JNZ rel [2B, 2C]
1	AJMP (P0) [2B, 2C]	ACALL (P0) [2B, 2C]	AJMP (P1) [2B, 2C]	ACALL (P1) [2B, 2C]	AJMP (P2) [2B, 2C]	ACALL (P2) [2B, 2C]	AJMP (P3) [2B, 2C]	ACALL (P3) [2B, 2C]
2					ORL dir, A [2B]	ANL dir, A [2B]	XRL dir, A [2B]	ORL C, bit [2B, 2C]
3								
4			ADD A, #data [2B]	ADDC A, #data [2B]	ORL A, #data [2B]	ANL A, #data [2B]	XRL A, #data [2B]	MOV A, #data [2B]
5	INC dir [2B]	DEC dir [2B]	ADD A, dir [2B]	ADDC A, dir [2B]	ORL A, dir [2B]	ANL A, dir [2B]	XRL A, dir [2B]	
6								MOV @R0, @data [2B]
7								MOV @R1, #data [2B]
8								MOV R0, #data [2B]
9								MOV R1, #data [2B]
A								MOV R2, #data [2B]
B								MOV R3, #data [2B]
C								MOV R4, #data [2B]
D								MOV R5, #data [2B]
E								MOV R6, #data [2B]
F								MOV R7, #data [2B]

注　[2B]=2 Byte，[3B]=3Byte，[2C]=2Cycle，[4C]=4 Cycle，Blank=1 byte/1 cycle。

8051 指令集——双字节指令整理 2

	8	9	A	B	C	D	E	F
0	SJMP REL [2B, 2C]		ORL C, /bit [2B, 2C]	ANL C, /bit [2B, 2C]	PUSH dir [2B, 2C]	POP dir [2B, 2C]		
1	AJMP (P4) [2B, 2C]	ACALL (P4) [2B, 2C]	AJMP (P5) [2B, 2C]	ACALL (P5) [2B, 2C]	AJMP (P6) [2B, 2C]	ACALL (P6) [2B, 2C]	AJMP (P7) [2B, 2C]	ACALL (P7) [2B, 2C]
2	ANL C, bit [2B, 2C]	MOV bit, C [2B, 2C]	MOV C, bit [2B]	CPL bit [2B]	CLR bit [2B]	SETB bit [2B]		
3								
4	DIV AB [2B, 4C]	SUBB A, #data [2B]						
5		SUBB A, dir [2B]			XCH A, dir [2B]		MOV A, dir [2B]	MOV dir, A [2B]
6	MOV dir, @R0 [2B, 2C]		MOV @R0, dir [2B, 2C]					
7	MOV dir, @R1 [2B, 2C]		MOV @R1, dir [2B, 2C]					
8	MOV dir, R0 [2B, 2C]		MOV R0, dir [2B, 2C]			DJNZ R0, rel [2B, 2C]		
9	MOV dir, R1 [2B, 2C]		MOV R1, dir [2B, 2C]			DJNZ R1, rel [2B, 2C]		
A	MOV dir, R2 [2B, 2C]		MOV R2, dir [2B, 2C]			DJNZ R2, rel [2B, 2C]		
B	MOV dir, R3 [2B, 2C]		MOV R3, dir [2B, 2C]			DJNZ R3, rel [2B, 2C]		
C	MOV dir, R4 [2B, 2C]		MOV R4, dir [2B, 2C]			DJNZ R4, rel [2B, 2C]		
D	MOV dir, R5 [2B, 2C]		MOV R5, dir [2B, 2C]			DJNZ R5, rel [2B, 2C]		
E	MOV dir, R6 [2B, 2C]		MOV R6, dir [2B, 2C]			DJNZ R6, rel [2B, 2C]		
F	MOV dir, R7 [2B, 2C]		MOV R7, dir [2B, 2C]			DJNZ R7, rel [2B, 2C]		

注　[2B]=2 Byte，[3B]=3Byte，[2C]=2Cycle，[4C]=4 Cycle，Blank=1 byte/1 cycle。

8051 指令集——3 字节指令整理 1

	0	1	2	3	4	5	6	7
0		**JBC** **bit, rel** [3B, 2C]	**JB** **bit, rel** [3B, 2C]	**JNB** **bit, rel** [3B, 2C]				
1								
2	**LJMP** **addr16** [3B, 2C]	**LCALL** **addr16** [3B, 2C]						
3					**ORL** **dir, #data** [3B, 2C]	**ANL** **dir, #data** [3B, 2C]	**XRL** **dir, #data** [3B, 2C]	
4								
5								**MOV** **dir, #data** [3B, 2C]
6								
7								
8								
9								
A								
B								
C								
D								
E								
F								

注　[2B]=2 Byte，[3B]=3Byte，[2C]=2Cycle，[4C]=4 Cycle，Blank=1 byte/1 cycle。

8051 指令集——3 字节指令整理 2

	8	9	A	B	C	D	E	F
0		**MOV DPTR, #data 16** [3B, 2C]						
1								
2								
3								
4				**CJNE A, #data, rel** [3B, 2C]				
5	**MOV dir, dir** [3B, 2C]			**CJNE A, dir, rel** [3B, 2C]		**DJNZ dir, rel** [3B, 2C]		
6				**CJNE @R0, #data, rel** [3B, 2C]				
7				**CJNE @R1, #data, rel** [3B, 2C]				
8				**CJNE R0, #data, rel** [3B, 2C]				
9				**CJNE R1, #data, rel** [3B, 2C]				
A				**CJNE R2, #data, rel** [3B, 2C]				
B				**CJNE R3, #data, rel** [3B, 2C]				
C				**CJNE R4, #data, rel** [3B, 2C]				
D				**CJNE R5, #data, rel** [3B, 2C]				
E				**CJNE R6, #data, rel** [3B, 2C]				
F				**CJNE R7, #data, rel** [3B, 2C]				

注　[2B]=2 Byte，[3B]=3Byte，[2C]=2Cycle，[4C]=4 Cycle，Blank=1 byte/1 cycle。

附录 C　8051 指令整理（按功能划分）

助记符		说　明	字节	指令周期
ARITHMETIC OPERATIONS				
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Rj	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#data	Add immediate data to Acc with Carry	2	12
SUBB	A,Rn	Subtract Register from Acc with borrow	1	12
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12
SUBB	A,@Ri	Subtract indirect RAM from ACC with borrow	1	12
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment direct RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement register	1	12
DEC	direct	Derement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A&B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12

注　All mnemonics copyrighted @Intel Corp., 1980.

助记符		说 明	字节	指令周期
LOGICAL OPERATIONS				
ANL	A,Rn	AND register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12

助记符		说 明	字节	指令周期
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A+PC	Move Code byte relative to DPTR to Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@DPTR	Move Exernal RAM(16-bit addr) to Acc	1	24
MOVX	@Ri,A	Move Acc to External RAM (8-bit ddr)	1	24
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12

助记符		说　明	字节	指令周期
BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear Carry	1	12
CLR	bit	Clear diret bit	2	12
SETB	c	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to CARRY	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move direct bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	REL	Jump if Carry is set	2	24
JNC	REL	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24

助记符		说　明	字节	指令周期
PROGRAM BRANCHING				
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE	Rn,#data,rel	Compare immediate to register and Jump if Not Equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	2	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

附录 D　8051 指令整理（按十六进制排列）

十六进制代码	字　节　数	助　记　符	操　作　数
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr，code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3

续表

十六进制代码	字　节　数	助　记　符	操　作　数
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr，code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A，#data
25	2	ADD	A，data addr
26	1	ADD	A，@R0
27	1	ADD	A，@R1
28	1	ADD	A，R0
29	1	ADD	A，R1
2A	1	ADD	A，R2
2B	1	ADD	A，R3
2C	1	ADD	A，R4
2D	1	ADD	A，R5
2E	1	ADD	A，R6
2F	1	ADD	A，R7
30	3	JNB	bit addr，code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A，#data
35	2	ADDC	A，data addr
36	1	ADDC	A，@R0
37	1	ADDC	A，@R1
38	1	ADDC	A，R0
39	1	ADDC	A，R1

续表

十六进制代码	字　节　数	助　记　符	操　作　数
3A	1	ADDC	A，R2
3B	1	ADDC	A，R3
3C	1	ADDC	A，R4
3D	1	ADDC	A，R5
3E	1	ADDC	A，R6
3F	1	ADDC	A，R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr，A
43	3	ORL	data addr，#data
44	2	ORL	A，#data
45	2	ORL	A，data addr
46	1	ORL	A，@R0
47	1	ORL	A，@R1
48	1	ORL	A，R0
49	1	ORL	A，R1
4A	1	ORL	A，R2
4B	1	ORL	A，R3
4C	1	ORL	A，R4
4D	1	ORL	A，R5
4E	1	ORL	A，R6
4F	1	ORL	A，R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr，A
53	3	ANL	data addr，#data
54	2	ANL	A，#data
55	2	ANL	A，data addr
56	1	ANL	A，@R0
57	1	ANL	A，@R1
58	1	ANL	A，R0

续表

十六进制代码	字　节　数	助　记　符	操　作　数
59	1	ANL	A，R1
5A	1	ANL	A，R2
5B	1	ANL	A，R3
5C	1	ANL	A，R4
5D	1	ANL	A，R5
5E	1	ANL	A，R6
5F	1	ANL	A，R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	date addr，A
63	3	XRL	date addr，#data
64	2	XRL	A，#data
65	2	XRL	A，data addr
66	1	XRL	A，@R0
67	1	XRL	A，@R1
68	1	XRL	A，R0
69	1	XRL	A，R1
6A	1	XRL	A，R2
6B	1	XRL	A，R3
6C	1	XRL	A，R4
6D	1	XRL	A，R5
6E	1	XRL	A，R6
6F	1	XRL	A，R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C，bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A，#data
75	3	MOV	data addr，#data
76	2	MOV	@R0，#data

续表

十六进制代码	字　节　数	助　记　符	操　作　数
77	2	MOV	@R1，#data
78	2	MOV	R0，#data
79	2	MOV	R1，#data
7A	2	MOV	R2，#data
7B	2	MOV	R3，#data
7C	2	MOV	R4，#data
7D	2	MOV	R5，#data
7E	2	MOV	R6，#data
7F	2	MOV	R7，#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C，bit addr
83	1	MOVC	A，@A+PC
84	1	DIV	AB
85	3	MOV	data addr，data addr
86	2	MOV	data addr，@R0
87	2	MOV	data addr，@R1
88	2	MOV	data addr，R0
89	2	MOV	data addr，R1
8A	2	MOV	data addr，R2
8B	2	MOV	data addr，R3
8C	2	MOV	data addr，R4
8D	2	MOV	data addr，R5
8E	2	MOV	data addr，R6
8F	2	MOV	data addr，R7
90	3	MOV	DPTR，#data
91	2	ACALL	code addr
92	2	MOV	bit addr，C

续表

十六进制代码	字　节　数	助　记　符	操　作　数
93	1	MOVC	A，@A+DPTR
94	2	SUBB	A，#data
95	2	SUBB	A，data addr
96	1	SUBB	A，@R0
97	1	SUBB	A，@R1
98	1	SUBB	A，R0
99	1	SUBB	A，R1
9A	1	SUBB	A，R2
9B	1	SUBB	A，R3
9C	1	SUBB	A，R4
9D	1	SUBB	A，R5
9E	1	SUBB	A，R6
9F	1	SUBB	A，R7
A0	2	ORL	C，/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C，bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0，data addr
A7	2	MOV	@R1，data addr
A8	2	MOV	R0，data addr
A9	2	MOV	R1，data addr
AA	2	MOV	R2，data addr
AB	2	MOV	R3，data addr
AC	2	MOV	R4，data addr
AD	2	MOV	P5，data addr
AE	2	MOV	R6，data addr

续表

十六进制代码	字　节　数	助　记　符	操　作　数
AF	2	MOV	R7，data addr
B0	3	ANL	C，/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A，#data，code addr
B5	3	CJNE	A,data addr，code addr
B6	3	CJNE	@R0，#data，code addr
B7	3	CJNE	@R1，#data，code addr
B8	3	CJNE	R0，#data，code addr
B9	3	CJNE	R1，#data，code addr
BA	3	CJNE	R2，#data，code addr
BB	3	CJNE	R3，#data，code addr
BC	3	CJNE	R4，#data，code addr
BD	3	CJNE	R5，#data，code addr
BE	3	CJNE	R6，#data，code addr
BF	3	CJNE	R7，#data，code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A，data addr
C6	1	XCH	A，@R0
C7	1	XCH	A，@R1
C8	1	XCH	A，R0
C9	1	XCH	A，R1
1CA	1	XCH	A，R2

续表

十六进制代码	字　节　数	助　记　符	操　作　数
CB	1	XCH	A，R3
CC	1	XCH	A，R4
CD	1	XCH	A，R5
CE	1	XCH	A，R6
CF	1	XCH	A，R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	A,data addr,code addr
D6	1	XCHD	A，@R0
D7	1	XCHD	A，@R1
D8	2	DJNZ	R0，code addr
D9	2	DJNZ	R1，code addr
DA	2	DJNZ	R2，code addr
DB	2	DJNZ	R3，code addr
DC	2	DJNZ	R4，code addr
DD	2	DJNZ	R5，code addr
DE	2	DJNZ	R6，code addr
DF	2	DJNZ	R7，code addr
E0	1	MOVX	A，@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A，@R0
E3	1	MOVX	A，@R1
E4	1	CLR	A
E5	2	MOV	A，data addr
E6	1	MOV	A，@R0

续表

十六进制代码	字　节　数	助　记　符	操　作　数
E7	1	MOV	A，@R1
E8	1	MOV	A，R0
E9	1	MOV	A，R1
EA	1	MOV	A，R2
EB	1	MOV	A，R3
EC	1	MOV	A，R4
ED	1	MOV	A，R5
EE	1	MOV	A，R6
EF	1	MOV	A，R7
F0	1	MOVX	@DPTR，A
F1	2	ACALL	code addr
F2	1	MOVX	@R0，A
F3	1	MOVX	@R1，A
F4	1	CPL	A
F5	2	MOV	data addr A
F6	1	MOV	@R0，A
F7	1	MOV	@R1，A
F8	1	MOV	R0，A
F9	1	MOV	R1，A
FA	1	MOV	R2，A
FB	1	MOV	R3，A
FC	1	MOV	R4，A
FD	1	MOV	R5，A
FE	1	MOV	R6，A
FF	1	MOV	R7，A

附录 E　8051 SFR 表与 RESET 后的初始值

F8H									FFH
F0H	B 00000000								F7H
E8H									EFH
E0H	ACC 00000000								E7H
D8H									DFH
D0H	PSW 00000000								D7H
C8H	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			CFH
C0H									C7H
B8H	IP XX000000								BFH
B0H	P3 11111111								B7H
A8H	IE 0X000000								AFH
A0H	P2 11111111								A7H
98H	SCON 00000000	SBUF XXXXXXXX							9FH
90H	P1 11111111								97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXX00XX0		8FH
80H	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000		PCON 0XXX0000	87H

附录 F　SFR 特殊功能寄存器整理表

寄存器名称	地　址	可位寻址	8051	8052
ACC 累加器	E0H	*	*	*
B 通用寄存器	F0H	*	*	*
PSW 程序状态字	D0H	*	*	*
SP 堆栈指针	81H		*	*
DPH 数据指针（高位）	83H		*	*
DPL 数据指针（低位）	82H		*	*
P0 端口 0	80H	*	*	*
P1 端口 1	90H	*	*	*
P2 端口 2	A0H	*	*	*
P3 端口 3	B0H	*	*	*
IP 中断优先控制器	B8H	*	*	*
IE 中断使能控制	A8H	*	*	*
TMOD 定时/计数模式控制	89H		*	*
TCON 定时/计数控制	88H	*	*	*
T2CON 第 2 定时/计数控制	C8H	*		*
TH0 TIMER0 高位设置	8CH		*	*
TL0 TIMER0 低位设置	8AH		*	*
TH1 TIMER1 高位设置	8DH		*	*
TL1 TIMER1 低位设置	8BH		*	*
TH2 TIMER2 高位设置	CDH			*
TL2 TIMER2 低位设置	CCH			*
RCAP2H 捕获寄存器（高位）	CBH			*
RCAP2L 捕获寄存器（低位）	CAH			*
SCON 串行通信控制器	98H	*	*	*
SBUF 串行数据缓冲器	99H		*	*
PCON 电源控制寄存器	87H		*	*

PSW 各位定义

CY	AC	F0	RS1	RS0	OV	–	P
D7	D6	D5	D4	D3	D2	D1	D0

TCON 控制寄存器

TF1	TR1	TF0	TR0	ID1	IT1	IE0	IT0
D7	D6	D5	D4	D3	D2	D1	D0

T2CON 各位定义

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$
D7	D6	D5	D4	D3	D2	D1	D0

SCON 串行控制端口位说明

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
D7	D6	D5	D4	D3	D2	D1	D0

PCON 电源控制寄存器各位说明

SMOD	—	—	—	GF1	GF0	PD	IDL
D7	D6	D5	D4	D3	D2	D1	D0

IP 中断优先级寄存器各位定义

—	—	PT2	PS	PT1	PX1	PT0	PX0
D7	D6	D5	D4	D3	D2	D1	D0

IE 中断使能寄存器各位定义

EA	—	ET2	ES	ET1	EX1	ET0	EX0
D7	D6	D5	D4	D3	D2	D1	D0

附录 G　日文专业杂志的订购

数个月之前，我们在试做某个原型（prototype）时，刚好要焊一个稳压 IC，焊接前当然要先知道其引脚，可是找了不少书籍与杂志，却始终无法找到 IC 的引脚图，最后才在日文杂志的专辑中找到答案。我们一直有一个疑问：每年卖出几百亿的个人计算机以及外围设备，可是连一个常用 IC 的引脚图却找不到，这是我们的出版业没有远见？还是心态的问题呢？

我每个月都要阅读两种以上的日文专业杂志，主要是《晶体管技术》（Transister Technology）与《接口》（Interface）这两份杂志，这些专业性杂志都是 CQ 出版社发行的，主要在探讨电子与计算机相关方面的设计与使用。我是通过台北市一家专卖日本杂志的书店代为订购，每次订阅的期限是一年。不过，订书手续听说非常繁杂，我们拿到书店寄来的第一本杂志已经是两个月之后了，由于这些杂志都是重量级的（广告多但是内容又实在），造成邮局的邮寄费用愈来愈贵，每年还要花至少七百元以上的邮费，所以我改到南部日系百货公司的图书部订书，价格稍微贵了一点，但是每个月强迫自己到百货公司走走、运动一下，顺便拿书未尝不是一件好事。

图 G-1　几本参考性质甚佳的日文杂志

附录 H　PRO 族：善用因特网上的各种 BBS

如果你想成为信息相关行业的重要开发人员，我们要提醒你一定要学会两种重要的工具，一是外语的能力，这包括了英日文甚至德法文的阅读能力，因为这样才能得到最先进的技术资料。另一项是要赶快学习使用因特网，学会用浏览器去看各个公司的首页（Homepage），学会用 Yahoo 的搜索引擎去找寻我们所需要的资料及文献。学习如何加入因特网各式各样的新闻组（News Group），因为全世界各方面的高手都会聚集在这里讨论，有任何问题在这边提出可能是最有效的，可是要提问题时，外语能力不加强怎么行呢？如果你还会运用因特网上 BBS 电子布告栏时，某些新闻组里早已有各种精华集以及 FAQ（常问问题解说），看完这些问题解析之后，可能就想到你的问题症结所在了，而剩下的事就是实际的行动了。

从因特网的查询当中，我们得知 Intel 已有几款与 8051 兼容且新的芯片 80c251/80c151 即将推出，这种查询及反应的速度比打电话到 Intel 分公司，等了三个月才拿到资料快多了。另外，我们也得知 Atmel 推出了内置 2KB E^2PROM 的微控制器，Harris 公司有许多新的 AD 转换 IC 问世等等第一手的信息，原先以为这些公司都离我们好远好远，但是通过因特网，这才发现大家都是住在“地球村”的村民，再不学会因特网的各项功能真的是不行的。

最近我们通过因特网的浏览器分别查询以下几个常去的网址：

http://www.intel.com　单片机 8051 的开山始祖厂商。

http://www.atmel.tom　Atmel 闪存的制造商。

http://www.analog.com　著名模拟 IC 的制造厂商。

http://www.nsc.com　National Semiconductor 电子元件制造厂。

http://www.harris.com　Harris Semiconductor 电子元件制造厂。

http://www.xilinx.com　Xilinx 公司，FPGA 的著名开发者。

http://www.altera.com　Altera 公司，也是 FPGA/CPID 的著名开发者。

http://www.ti.com　TI 也是世界知名的半导体 IC 大厂。

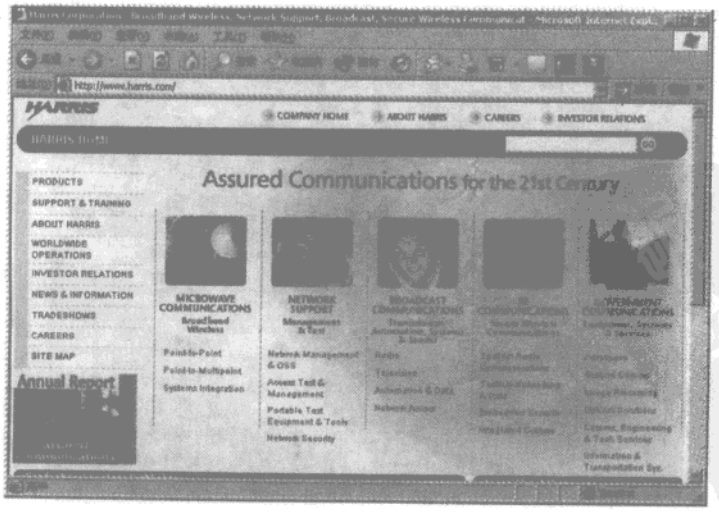


图 H-1　Harriops 半导体公司的主页

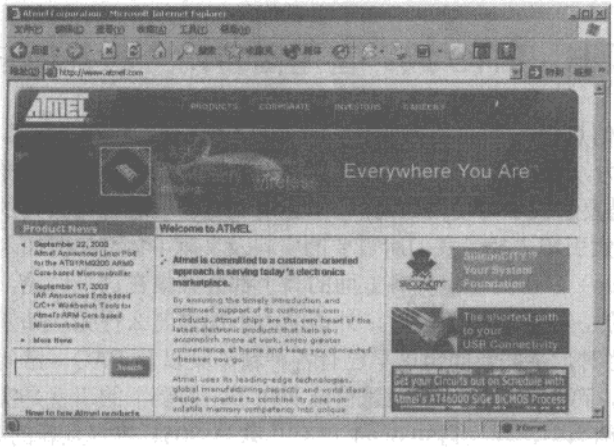


图 H-2　Atmel 的主页

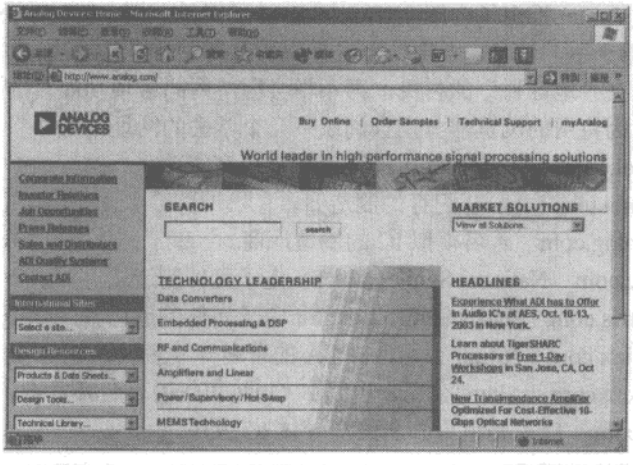


图 H-3　Analog 公司的主页

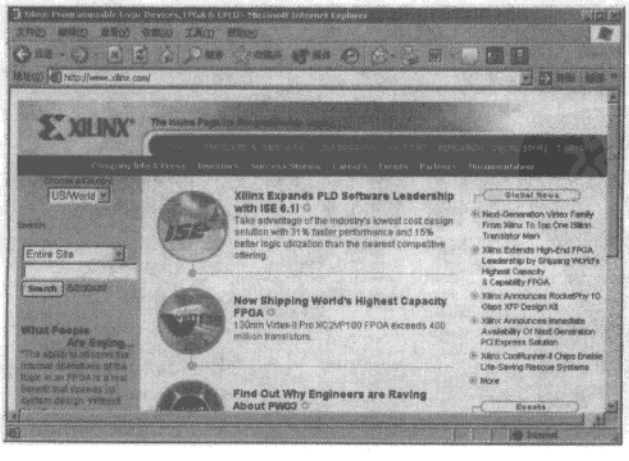


图 H-4　Xilinx 公司的主页

附录 I　DIS51 的深度应用

我们以前曾经让读者来函索取 8051 的反汇编程序（DIS51.EXE），用这个程序来反汇编我们的二进制文件范例程序 READTEMP，相信许多用功分析程序的读者一定从中获得宝贵的知识。这个程序总共有三百多个读者索取。不过，有一天清晨两点计算机要关机前，我照惯例再进到因特网查询是否有 E-mail 邮件进来，结果在清晨一点半的时候，有一位正在加班的工程师传来一封信，信中的大意是：用 DIS51.EXE 来看一个 4KB 的 8051 程序，看得很辛苦，我想这个程序是别人写的吧！程序中用了可位寻址的标志位，所有的流程及操作都是以这些标志位为依据。当然如果不知道这些标志位的用意时，会使得程序可读性变成相当低，所以程序看了好几天还是不得其门而入。看来今天又要无法入眠了，这种限期完工的压力是相当大的。

在我们的开发 8051 程序经验中，一定会用或多或少个可位寻址的标志位当成流程控制的依据。如果你不是原始程序的开发者时，就很难在短时间理解程序了，所以一定要费更多的时间去推敲。当 DIS51 反汇编完程序后，除了 DIS 反汇编文件之外，还会产生一个 BAS 文件，上面会记载每个可位寻址（Bit Addressable Segment）的调用次数及调用的地址，我们建议你把整个程序打印出来，报表纸通常会有数百张之多，你可以借由 BAS 文件找出使用次数最多的位地址，然后查询它在哪一部分使用了 JB/JNB 做跳转的依据，接着找出该位在何处被设置及重设置，从这三方面去推断程序的用意。不过可以理解的是：这类程序不是一两天能搞得清楚的，有些程序要反复多次推敲才能弄懂的。看别人的 8051 程序从头到尾看了二三十次是很正常的，我们保证绝对是不假的。

如果分析的 8051 控制器还有输入/输出电路的关系时，看起来又更复杂了。首先，我们会把整个电路画成线路图，哪些点对应到输入，哪些点对应到输出，这些全要弄清楚才可以开始分析程序。不过有些程序为了阻止其他人的读取，会加入软硬件的保护，这么一来，若不先解开这些障碍，就无法顺利读取程序了。

假使待分析的控制板上有一个空余的独立 I/O 点时，我们可以拿来做程序的除错或修改开关，并且适时地把状态值送出来，如果你可以做到这个地步时，就可以重新改写程序了。看别人的程序太累了！的确是这样的，但是有些时候又非得如此，今年五月的时候我就碰到一个案例：某大学某系的实验器材多波段（Multichannel）高速 AD 转换仪器故障，该设备已用超过五年的时间，早已过保修期，原厂也不愿意提供维修服务，只好求助于旗威科技公司来处理，因为有好几个研究生要靠这台设备去做论文，我们只好死马当活马来医了。首先我们仔细地研究了一下机器，发现应该只有其中一组 AD 转换电路损坏，开机后就是因为该 CH 没有响应，而使得机器一直停在等待响应状态上。由于没有线路图可供维修，而且没有备用元件可以更换，所以我们只好走另一条路试着修改内部固件程序了，那就可以让机器跳过故障点继续工作了。所以朝着这个方向去反汇编程序并且分析程序，然后重新烧录一个 EPROM，两三天内就把这台机器修好了。可是，如果不这样做时，原单位可能要多花不少的预算去购买新的设备了。

这一章我们尝试用另一种方式来回答读者所遇到的问题，希望对您有所帮助，如果您有类似的问题时，也可以通过 E-mail 传给旗威科技公司。不过要向读者致歉的是，您先前寄来的若是属于 MIME 格式的信函，在我们这端看来都是乱码，所以无法回复您的问题了。所幸

这些信函格式的问题已经解决了，请尽量以 E-mail（chipware@seed.net.tw）来提出问题，以便我们有足够的时间来思考与探讨。

反汇编程序（DIS51.EXE）的操作与使用

我们必须承认学习的第一步就是模仿，学习单片机 8051 也是如此，市面上有许多 8051 的相关产品，若你想对这些东西做更进一步的研究时，该怎么办呢？首先，我们来看看其中的困难点，这类的产品本身都有某些程度的保护，纵使这些保护线路可顺利被解开，若想进一步研究或修改程序也是困难重重，因为程序经过编译、连接后已变成机器码（Machine Code），我们很难由这些机器码中倒推原来的程序写法和动向，而且这些程序机器码中有指令也有数据，必须花相当的心血才能顺利地把指令和数据分开。

所有的 ICE 仿真器都可以将程序机器码反汇编成汇编语言的格式，分别显示在屏幕或打印在报表上，这种反汇编的方式是逐行翻译的，可以知道程序的写法，但是对程序整体性的分析则略嫌不足，因为我们看程序时，首重各个程序的操作分析，只要知道各个程序的来龙去脉，就能很容易地看出程序的重要关键点。

针对上述的问题，我们特地开发一个 8051 的反汇编程序，只要得到 8051 的二进制文件数据（必须是正确的），并存入软盘或硬盘中，就可以通过这个反汇编程序——DIS51.EXE 把机器码转换成汇编语言的格式，除此之外，它另外产生三个参考数据文件：

（1）反汇编后的汇编程序文件其扩展名为 DIS。

（2）DPTR 参考数据文件其扩展名为 DTR。

（3）CALL 参考调用文件其扩展名为 CAL。

（4）JUMP 参考跳转文件其扩展名为 JMP。

DIS51.EXE 额外产生的文件把程序中所有关于 CALL、DPTR 和 JUMP 的位置和次数整理出来，并分别存到三个文件中。我们可以通过这些信息，很快地分析出哪些程序被调用的次数最多，DPTR 后若接着是 MOVC 指令，则 DPTR 所指的位置一定是数据而非程序，而 DPTR 出现后若接着是 MOVX 指令时，该地址一定是 RAM 或 I/O 地址。我们另外还可轻松地算出有多少个 I/O 端口接在这个系统上，这些信息确实是 ICE 仿真器所无法提供的。

DIS51.EXE 最多可翻译 32KB 的程序数据，其操作步骤如下：

步骤 1：取得正确的 ROM 程序存储器数据，通常由 ICE 仿真器中存入的数据都是正确的，或是由 EPROM 烧录器将程序 ROM 读入后再以二进制格式存入硬盘或软盘中，存入的文件不必指定子文件名，假设我们已得到某个程序 ROM，其存入软盘的文件名称为 A:TEST。

步骤 2：由本书所附的光盘中加载 DIS51.EXE，若该文件已存入硬盘 C 区时，我们只要在 C>下输入 DIS51<ENTER>即开始进入反汇编程序。

步骤 3：DIS51 加载计算机后，会在屏幕上要求输入待反汇编文件的文件名称，这时我们可以输入先前已存入的文件名称 A:TEST。

步骤 4：DIS51 反汇编程序首先检查程序的长度，并将结果以十六进制的方式显示该长度。例如：

长度是 2000H 时，代表该程序有 8KB 长。

长度是 4000H 时，代表该程序有 16KB 长。

长度是 8000H 时，代表该程序有 32KB 长。

步骤 5：反汇编程序接着会显示它将产生四个标准文字文件：

（1）反汇编程序文件 A:TEST.DIS

（2）该程序所使用的 DPTR 文件 A:TEST.DTR

（3）该程序所使用的 CALL 文件 A:TEST.CAL

（4）该程序所使用的 JUMP 文件 A:TEST.JMP

步骤 6：程序接着要求输入停止反汇编的地址值，这个值应该小于等于该文件的长度值。

步骤 7：反汇编程序开始运行，并产生对应的四个文本文件，这些文本文件可用一般的编辑程序（PE2 或 KS2）来查看其中的内容。

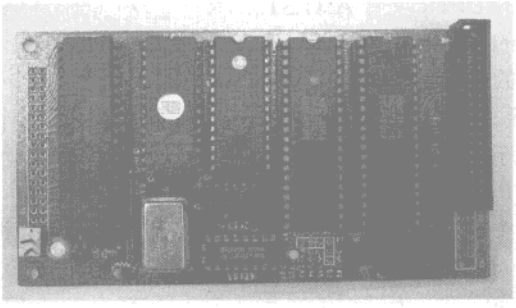
步骤 8：上述的 4 个文本文件中，以 TEST.DIS 反汇编后的程序文件占的空间最大，通常会超过 100KB 以上，这时可再用另一个可执行程序（TEST_DVD.EXE），将 TEST.DAT 切割成数小部分，以免空间过大而造成编辑程序无法处理。

步骤 9：TEXT_DVD.EXE 执行时，只需分别输入原文本文件的名称（不加扩展名）及欲输出的文件名称（也不必加扩展名）即可。

步骤 10：TEXT_DVD.EXE 将 TEST.DIS 文本文件每隔 4000 行分割成一个文本文件，若 TEST.DIS 有 10000 行，且输出的文件指定为 TESTX 时，则共产生三个文本文件：TESTX.DS1 占 4000 行、TESTX.DS2 也占 4000 行，TESTX.DS3 则只占 2000 行。

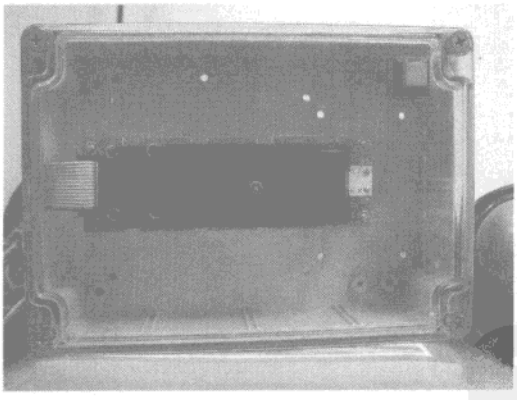
我们也用另一个汇编程序分别对照原程序和反汇编后的程序间有何不同点。对此程序写法有兴趣的读者可直接与作者联络，以便索取该原始程序。

附录 J　一张照片一个故事



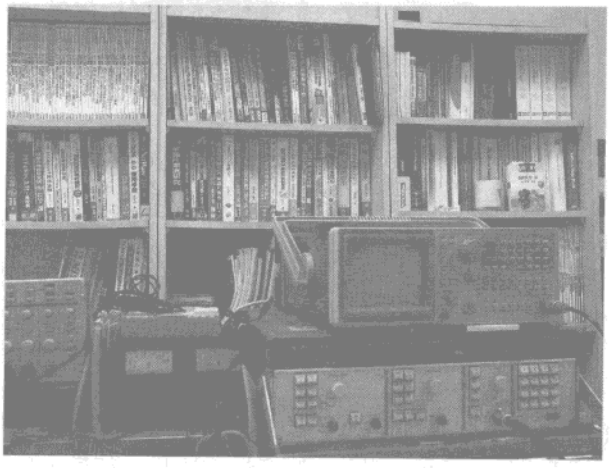
Z80 控制板

早期的单片机控制器是用 Z80 CPU、ROM、SRAM 与 I/O 组合而成的，一个复杂的控制系统的所有程序就放在一颗 16KB 的 27128 EPROM 内部。所有的 I/O 则通过 8255 来处理。系统的振荡晶体为 4MHz 的 OSC，一般的机械控制用这种组合就够了，我们曾经用这块控制板去操作整台智能化刺绣机。整个程序是用 C 来处理的，开始试机的时候程序占用 8KB 的空间，程序整个完成后也是 8KB，听起来有点不可思议，说穿了是我们对程序做了结构化与优化处理，程序空间才能节省下来。现在的开发人员写程序时，已经不会考虑程序空间是否足够，重要的是先写出来卖。



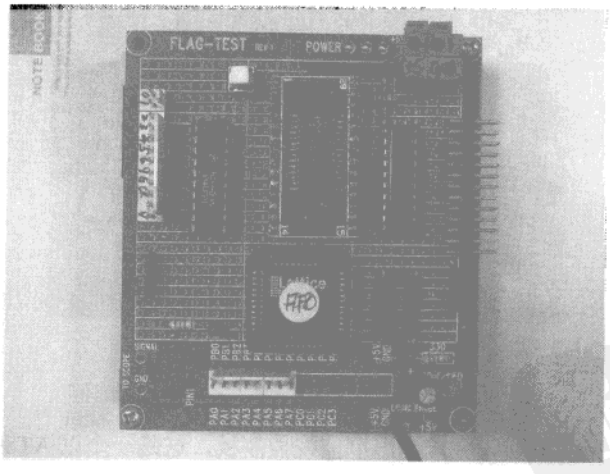
LCD 模块照片

做一般的控制器最怕的就是除错了，如果你设计的控制器又不是基于 PC 的，那问题就更复杂了，我们习惯在控制板上设计一个除错专用的 LCD 接口，硬件的花费可能不要 50 元，当系统程序正在架构中，该 LCD 接口可以立即显示重要的参数值与状态值。当所有操作都就绪后，再把 LCD 相关的程序删除。如果程序空间允许的话，我们会保留这些程序代码。当对系统有疑问时，我们只要按个开关或接个跳线（jumper），就可以从 LCD 的画面上进行状态的诊断。



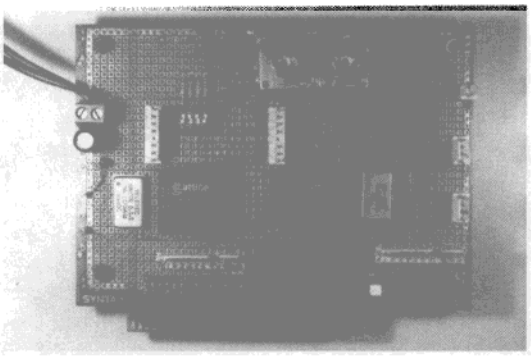
书与仪器孰轻孰重

一本书几十元，一百本书籍也只有几千元而已，可是一台信号发生器（Signal Generator）加上频谱分析仪（Spectrum Analyzer）可能就得几万元。你觉得哪个比较重要呢？其实仪器只是工具，我们用它来验证设计上是否有缺陷，但是，书籍却提供我们构想与远景，好的书会帮助我们、启发我们，让我们在设计电路或编写程序时，提供足够的信息。我们认为“一本好书所获得的好处，远比一堆仪器获得的还多”，书可以带到床上看，仪器却不可以。



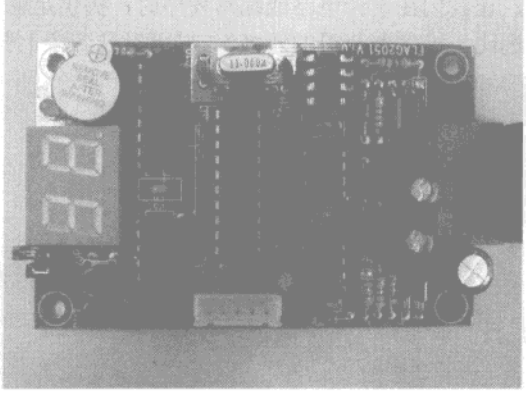
自制的 32KB FIFO

每隔一阵子就会在 BBS 站上看到“哪里有卖 Dual Port RAM”的信息，如果真的买得到，价格也是不便宜的。5 年前我们在设计通信设备的测试仪器时，也是碰到类似的困扰。由于我们对 CPLD 已经相当了解，自己写些 ABEL 程序仿真 Dual Port RAM 的硬件，花不到两天的时间，我们就把 CPLD 与 SRAM 弄成 Dual Port RAM。不到一周就完成测试的验证工作。设计的工作不难，难的是碰到事情之前的准备工作。



停电保留的陷阱

这是数年前我们所设计的 AC 控制板，上面的 AT89C2051 CPU 只控制 4 个 AC 开关，当初的构想是做停电保留的功能，若第一组开关原本是 ON 的，如果瞬间停电了，当电力又重新供应时，系统应该继续对第一组开关供电。操作看似简单，我们是把开关的状态值以串行的方式存入 E^2PROM 93C46 中，每次电源被打开时，就从 93C46 上读回数据，然后重新设置开关的状态值。可是，在试验阶段发现，E^2PROM 的状态值有时候会被无缘无故变动，而且出错的比例超过 10%，这是硬件还是软件的问题呢？这个问题一直困扰着我们，直到有一天我们看到“AT89C2051 在 2V 以上仍然能工作”的提示，这才找到解决的方法。



新的 AT2051 温度控制板

这是旗威科技公司 2002 年完成的温度监视控制板，这是一个以 AT89C2051 为主体的控制板，板上有一个温度感测 IC，它可以把摄氏温度值用脉冲的方式传给 CPU。另外，它具有标准的 RS485 接口，只要你的仪器或 PC 也有此接口，就可以与这块温度控制板通信，并取得板上所有的设置数据。AT2051 温度控制板的电压输入范围很大，从+7～+32V 都可以接受，详细的操作情形请参考《8051 单片机彻底研究——基础篇》中的说明。

[General Information]
书名=8051单片机彻底研究　经验篇
作者=林伸茂编著
页数=258
出版社=中国电力出版社
出版日期=2007
SS号=12271487
DX号=000006205422
URL=http://book.szdnet.org.cn/bookDetail.jsp?dxNumber=000006205422&d=11DA6D6258E167D2943772 3BDDE44316

附录 J　一张照片一个故事