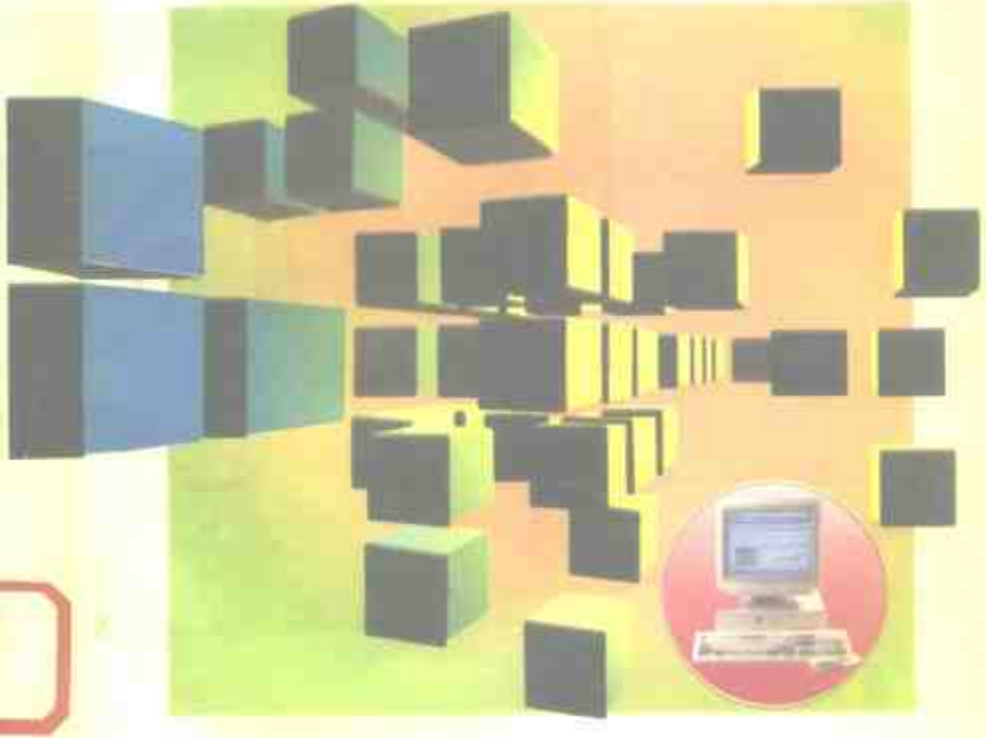
齐秋群　刚秀冰 等

Motorola

单片机实用技巧集萃

（一）



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

Motorola 单片机实用技巧集萃

(一)

齐秋群　刚寒冰等

电子工业出版社

内 容 提 要

本书详细地介绍了几十个实用性很强的 Motorola M68HC05/M6805/M68HC11 单片机应用实例，内容涉及通讯、家用电器、测量/控制/显示系统、仪表、单片机系统设计与系统扩展技术、单片机开发与编程等。每个例子都给出详细的硬件电路和具体软件程序，读者可以直接采用或稍加改动应用到相近的场合或改用其他型号的单片机。

本书实用性强，内容翔实，取材新颖，对开发、应用单片机具有很高的参考价值。本书适用于从事单片机开发与应用的工程技术人员、科研人员和高校师生阅读。

Motorola 单片机实用技巧集萃

（一）

齐秋群　刚寒冰等

责任编辑　郭　晓

*

电子工业出版社出版

北京市海淀区万寿路 173 信箱(100036)

电子工业出版社发行　各地新华书店经销

电子工业部情报所印刷

*

开本：787×1092 毫米 1/16 印张：22.25　字数：570 千字

1996 年 4 月第一版　　1996 年 4 月北京第一次印刷

印数：3500 册　　定价：38.00 元

ISBN7－5053－3334－8/TP・1270

前　言

Motorola M68HC05、M68HC11、M6805 系列是国际上应用最广泛的单片机，具有功能丰富、性能价格比高、功耗低、系统设计简单、使用方便、速度高等特点，在家电、通讯、测控系统、仪器、汽车领域获得了广泛应用。

目前国内 Motorola 单片机用户迅速增加，但目前市场上大都是主要介绍 Motorola 单片机原理或技术手册的图书，工程技术人员在开发、设计、应用单片机时可参考的有关实际应用例子的资料匮乏。我们编写这本书也正是为了弥补这一缺陷。本书详细介绍几十个成功地应用单片机的例子，每个实例都有具体的硬件电路和详细的软件程序，读者可直接采用或进行移植，对广大开发使用单片机的专家与同行无疑具有很高的参考价值。

本书的例子都是从许多应用文章精选出来的，实用价值高，内容翔实、完整，取材新颖，内容涉及家电、通讯、控制与测量系统、仪器仪表单片机系统设计与系统扩展、单片机开发技术等各个领域。

本书主要由齐秋群、刚寒冰编写，参加编写者还有刚砺韬、姜洪福、常安、文州峰、高京斋、姜朋、刘化亮、刘联、郭小吉、李宇仁。

本书不足和错误之处，请读者批评指正（北京理工大学电子工程系，100081）。

作　者

1995.9.

目　录

1. 具有 128K 字节寻址能力的 M68HC11 系统

M68HC11 系列单片机的最大寻址范围为 64K 字节，这在有些应用场合是不够用的。这里介绍两种存储器分页方法，使 M68HC11 的寻址范围扩大到 128K 字节。这种分页技术可以扩展到对多个 EPROM、RAM 或 EEPROM 存储器进行寻址的场合。

1.1 分页原理

M68HC11 MCU 可寻址 64K 字节连续的地址空间。当需要的地址空间大于 64K 字节时，要用另一个具有相同地址空间存储器块代替这一存储器块，这一技术称为分页(Paging)，是扩展地址空间的简单而有效的方法。在这种方法中，有多个数据块(页)重叠，但在某个时刻 CPU 只能访问其中一页。

有两种基本方法：方法 A 只采用软件和一个 I/O 口引脚控制最高位地址 A16；方法 B 采用少量硬件和软件控制最高 3 位地址 A14、A15 和 A16。

在下面的例子中，采用具有非公用地址/数据总线的 MC68HC11G5 说明分页技术。任何其他的 M68HC11 器件均可采用类似的方法。

方法 A 的特点是不需要额外的硬件，对软件的限制少，用户码可达 64K 字节并保持在同一页，但中断等待时间长。方法 B 的优点是不影响中断等待时间，有一份向量表。该例中用户码主程序最大长度为 48K 字节，另有 5 页分别为 16K 字节，用于子程序和表格。

1.2 方法 A——软件方法

EPROM 的地址 A16 直接由 M68HC11 的 PD5 控制，如图 1-1 所示，复位后 D 口设置为输入状态。复位后，控制地址 A16 的引脚状态是很重要的，因此，PD5 接一个 10kΩ 的上拉电阻，以强迫复位后 A16 为逻辑高电平状态。应该注意，在数据方向寄存器写入 1 设置为输出之前，必须先向数据寄存器写入 1。

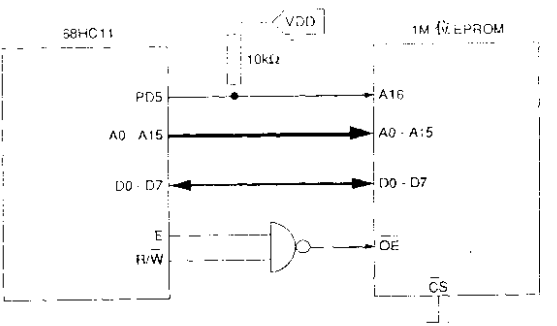


图 1-1 软件分页结构图

这种方法可允许 M68HC11 访问 128K 字节的 EPROM，该 EPROM 就如同两个均为 64K 字节的存储器，通过改变 EPROM 的地址线 A16 的状态分页。要确保口时序在地址选通（MC68HC11G5 的 E 时钟的上升沿）之前改变引脚状态，并且满足建立和保持时间。

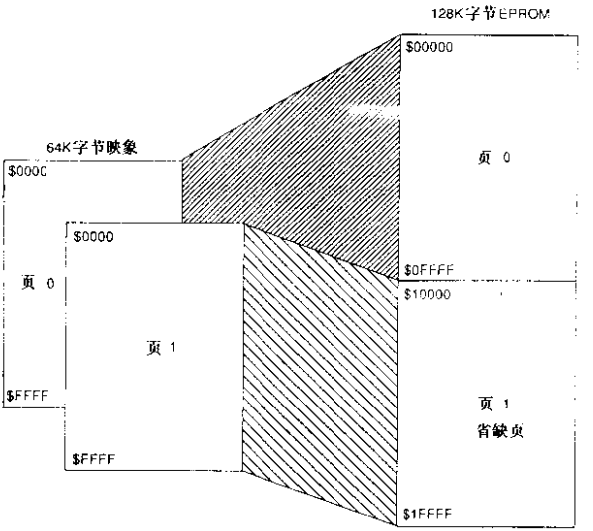


图 1-2 软件分页说明

图 1-2 是这种分页技术的原理图。它不能从一个 64K 页直接跳转到另一个 64K 页。解决该问题有两个方法：用户码在 RAM 中建立一个程序（它由两页公用，因为它位于内部，不受 PD5 的影响）；或在两页的相同地址具有页转换程序。后面的程序采用后一种方法。

1.2.1 中断程序

在 PD5 清零或置位之前，页转换程序存储当前页，而且页转换程序的跳转命令在两个存储器中必须具有相同的地址。因为 PD5 清零或置位将立即改变存储器页，但程序计数器正常加 1。因此，从页 0 变为页 1 时，为保证正确跳转，变址寄存器 X 必须装入在新的一页要执行的程序的地址，图 1-3 是改变页时从页 1 变为页 0 时的执行情况。

从中断返回的程序需要在执行中断程序之前先检查含有存储器页的 RAM 地址。然后，或者立即执行 RTI 命令（若处于同一页）；或者改变 PD5 的状态，再在正确的页执行 RTI 命令。注意，对于 JMP 0,X 指令，RTI 必须在两者的相同地址。CCR 的 I 位（中断禁止位）在该期间置位，以便正确运行，否则返回页可能变化。在当前中断之前用堆栈保持页号可克服这一问题。

1.2.2 其他程序

利用相同的转换程序可在任何时候从一页转换到另一页，无需在 RAM 中存储目前页，因此，变页程序可在 BCLR 或 BSET 开始，节省 4 个周期，可使换页延迟减少到 17 个周期。注意，不能用该例方法执行 JSR 命令进入其他页，因为 RTS 不会返回到原来的一页，但可进行适当

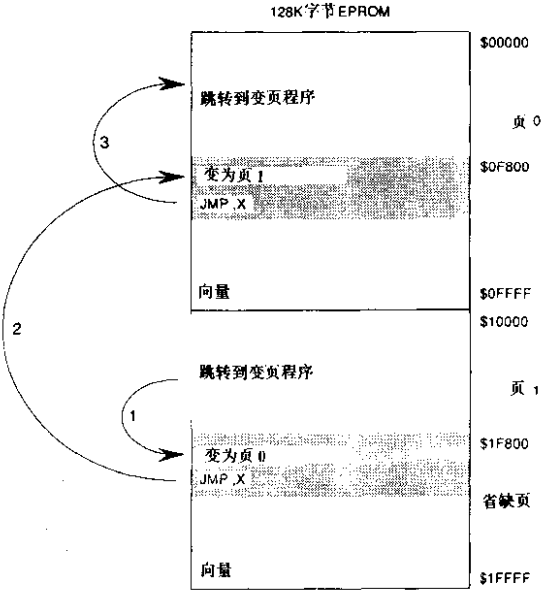


图 1-3 从页 1 变为页 0 的程序流程图

注:1. 跳转到变页程序 2. 变为页 0 3. 跳转到 X 寄存器的地址(页 0)

修改。在这种情况下,应该用堆栈保持正确的返回页,或将 CCR 的 I 位置位防止中断。

1.2.3 重要条件

控制复位后 A16 的状态是很重要的。该例中 PD5 复位后为输入,并用上拉电阻强迫 A16 为高。如果使用固定输出脚,则为逻辑 0(不需上拉电阻)。设置 I/O 口的初始化程序必须在省缺页,否则不能正确地控制 A16。类似地,若用双向 I/O 线,要有下拉电阻来确定复位后的地址线 A16。

RAM、寄存器和内部 EEPROM 若允许时,将总是出现在存储器映象中,因此应避免与这些地址相冲突。也可由改变 INIT 寄存器来改变 RAM 和寄存器的地址。

1.2.4 软件分页方法程序

* 1M 位(128 字节)EPROM 分为 2×64K 字节页
* A16 连至 PD5,其状态决定访问哪一页,复位后 PD5=1
* 该程序以 68HC11G5 编写,可容易地修改到其他 68HC11 器件

```
PORTA       EQU     $00
DDRA        EQU     $01
PORTB       EQU     $04
```

```
PORTC       EQU     $06
DDRC        EQU     $07
PORTD       EQU     $08
DDRD        EQU     $09
TMSK2       EQU     $24
TFLG2       EQU     $25
RTII        EQU     $40
RTIF        EQU     $40
PACTL       EQU     $26
DDRA7       EQU     $80
REGS        EQU     $1000
 * RAM 定义(从$0000～$01FF)
            ORG     $0000
 PAGE       RMB     1
 TIME       RMB     2
 *
 NPAGE      EQU     $20         PD5 页控制线
 ROMBASE    EQU     $0200
 CHANGE     EQU     $F800
 VECTORS    EQU     $FFCC

 * 主程序开始
 * 页 0(EPROM 的第一部分)
      ORG           ROMBASE
 RESET0 LDX         #RESET
   JMP              CHGPAGE0
 *
LOOPP0  BSET    PORTA,Y,#$10        位 4 翻转
        BCLR    PORTA,Y,#$10
        LDX     #LOOPP1             获得页 1 的返回地址
        JMP     CHGPAGE0            转变页程序
*  实时中断服务程序
RTISRV  BRSET   TFLG2,Y,#RTIF,RTISERV
        RTI                         若不是正确的中断源则返回，
  *                                 在页 1 时，因为中断向量只指向
                                    该处，故为 RTI
RTISERV
        LDAA    #%01000000          零页中断从此开始
        STAA    TFLG2,Y             清 RTI 标志
        LDAA    TIME+1              取时间计数器
        INCA
        STAA    PORTB+REGS          将时间存贮在 B 口
        LDX     TIME
        INX
```

```
        STX     TIME                    并存入 RAM 中
        JMP     RETRTI0                 转 RTI 程序
*   改变页程序
*   执行该程序时必须使 I 位置位,以防止中断。否则可能造成错误
*   该程序在两页的相同地址都出现
*   转移程序
        ORG     CHANGE                  该程序的地址固定
*                                       (周期)
CHGPAGE0
        LDAA    #0              (2)     目前页号置为 0
        STAA    PAGE            (2)     存入 PAGE
        BSET    PORTD,Y,#NPAGE  (8)     由设置 PD5 改变页
        JMP     0,X             (3)     该码在两页中均相同
*   从中断程序返回
RETRTI0
        LDAA    PAGE            (2)     获得发生中断的页
        CMPA    #1              (2)     是否为页 1?
        BEQ     RTIPAGE0        (3)     是,则变页
        RTI                     (12)    否则,从中断返回
RTIPAGE0
        BSET PORTD,Y,#NPAGE     (8)     变页并从中断返回

        RTI                     (12)    该码在两页中相同
*   向量
        ORG     VECTORS
        FDB     RESET0          EVENT 2
        FDB     RESET0          EVENT 1
        FDB     RESET0          TIMER OVERFLOW 2
        FDB     RESET0          INPUT CAPTURE 6 / OUTPUT COMPARE 7
        FDB     RESET0          INPUT CAPTURE 5 / OUTPUT COMPARE 6
        FDB     RESET0          SCI
        FDB     RESET0          SPI
        FDB     RESET0          PULSE ACC INPUT
        FDB     RESET0          PULSE ACC OVERFLOW
        FDB     RESET0          TIMER OVERFLOW 1
        FDB     RESET0          INPUT CAPTURE 4 / OUTPUT COMPARE 5
        FDB     RESET0          OUTPUT COMPARE 4
        FDB     RESET0          OUTPUT COMPARE 3
        FDB     RESET0          OUTPUT COMPARE 2
        FDB     RESET0          OUTPUT COMPARE 1
        FDB     RESET0          INPUT CAPTURE 3
        FDB     RESET0          INPUT CAPTURE 2
        FDB     RESET0          INPUT CAPTURE 1
        FDB     RTISRV          REAL TIME INTRRUPT
        FDB     RESET0          IRQ
```

```
        FDB     RESET0          XIRQ
        FDB     RESET0          SWI
        FDB     RESET0          ILLEGAL OPCODE
        FDB     RESET0          COP
        FDB     RESET0          CLOCK MONITOR
        FDB     RESET0          RESET
* 页1(EPROM的第二部分)
    * 主程序,不在中断控制之下
            ORG   ROMBASE
RESET       LDS   #$01FF
            JSR   SETUP                     初始化RTI中断和DDR
LOOP1       LDAA  #$FF
LOOP        BSET  PORTA,Y,#$08              位3翻转
            BCLK  PORTA,Y,#$08
            LDX   #LOOPP0                   转向另一页
            JMP   CHGPAGE1
LOOPP1
            DECA                            从另一页的返回点
            BNE   LOOP
            BRA   LOOP1                     再次开始循环
            *初始化程序
SETUP       SEI
            LDY                             #$1000  寄存器地址偏移量
            LDAA                            #$FF
            STAA                            DDRA+REGS  使A口均为输出
            STAA                            PORTD+REGS  确保PD5写入1
            STAA                            DDRD+REGS
LDAA        #%01000000
            STAA                            TFLG2+REGS  清RTI标志
            STAA                            TMSK2+REGS  允许RTI中断
            CLI
            RTS
*  Redirect to the Real time interrupt service routine
*  Page 1 routine for service routine located in page 0
INTRTI      BRSET TFLG2,Y,#RTIF,GOODINT
            RTI                             若不是正确的中断源则返回
GOODINT
            LDX #RTISERV                    (3周期)获得页0中断进入点
            JMP CHGPAGE1                    (3周期)转换页程序
* 换页程序
* 跳转程序
            ORG CHANGE
*                                           (周期数)
CHGPAGE1
            LDAA  #$1                       (2)使当前页号置1
```

```
                STAA PAGE                       (2)存储页号
                BCLR PORTD,Y,#NPAGE             (8)清 PD5,变页
                JMP 0,X                         (3)
*  从中断程序返回
RETRTI1
                LDAA PAGE                       (2)
                CMPA #0                         (2)
                EBQ  RTIPAGE 1  -               (3)
                RTI                             (12)
RTIPAGE1
                BCLR PORTD,Y,#NPAGE             (8)
                RTI                             (12)
* 向量
        ORG     VECTORS
        FDB     RESET           EVENT 2
        FDB     RESET           EVENT 1
        FDB     RESET           TIMER OVERFLOW 2
        FDB     RESET           INPUT CAPTURE 6 / OUTPUT COMPARE 7
        FDB     RESET           INPUT CAPTURE 5 / OUTPUT COMPARE 6
        FDB     RESET           SCI
        FDB     RESET           SPI
        FDB     RESET           PULSE ACC INPUT
        FDB     RESET           PULSE ACC OVERFLOW
        FDB     RESET           TIMER OVERFLOW 1
        FDB     RESET           INPUT CAPTURE 4 / OUTPUT COMPARE 5
        FDB     RESET           OUTPUT COMPARE 4
        FDB     RESET           OUTPUT COMPARE 3
        FDB     RESET           OUTPUT COMPARE 2
        FDB     RESET           OUTPUT COMPARE 1
        FDB     RESET           INPUT CAPTURE 3
        FDB     RESET           INPUT CAPTURE 2
        FDB     RESET           INPUT CAPTURE 1
        FDB     INTRTI          REAL TIME INTRRUPT
        FDB     RESET           IRQ
        FDB     RESET           XIRQ
        FDB     RESET           SWI
        FDB     RESET           ILLEGAL OPCODE
        FDB     RESET           COP
        FDB     RESET           CLOCK MONITOR
        FDB     RESET           RESET
        * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        END
```

1.3 方法 B——软硬件结合的方法

这种方法与前一种方法类似，但用一些硬件取代部分软件。M68HC11 的 A14 和 A15 经

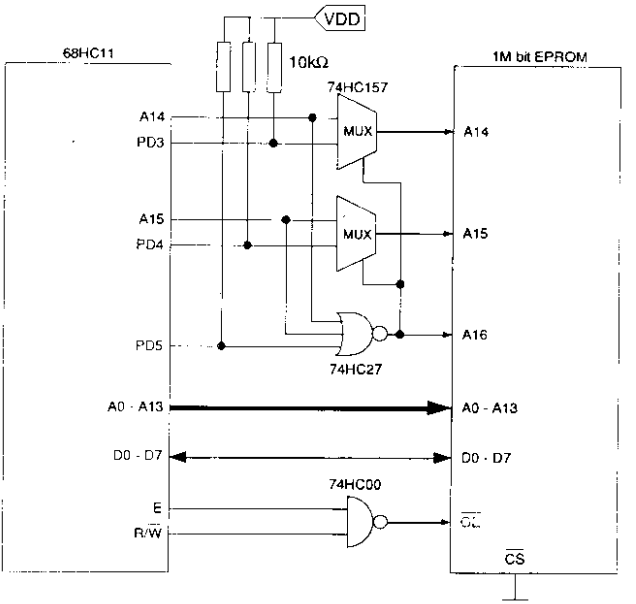


图 1-4　软硬件分页方法

或非后控制 EPROM 的 A16，如图 1-4 所示。它分为一个 48K 字节公共页和 5 个 16K 字节页。

例如，PD3 和 A14 地址线连到二选一多路器，或非门的输出决定这两个信号中哪个信号连到 EPROM 的 A14。类似地，或非门的输出还决定 PD4 和 A15 地址线这两个信号之中哪一个连到 EPROM 的 A15。若 A14 或/和 A15 为 1，则或非门输出(74HC27)为逻辑 0，即 EPROM 的 A16 为 0。这种情况下，HC11 的 A14 和 A15 分别控制 EPROM 的 A14 和 A15。这样，48K 字节主程序部分(EPROM)在任何时间都可在 $4000～$FFFF 访问。当 PD5、A14 和 A15 均为 0 时，(地址范围 $0000～$3FFF)，PD3 和 PD4 决定 EPROM 的 A14 和 A15(A16=1)。PD3、PD4 共有 4 种组合，即有 4 个 16K 字节页。这种结构共有 5 个 16K 字节页和一个 48K 字节页，该 48K 字节页总是处于存储器映象中，如表 1-1 所示。

因 PD3～PD5 均有上拉电阻，复位后为逻辑高状态(PD 口为输入)故复位后为主程序页和页 0。

1.3.1　C 语言的实现

程序由汇编语言写成，也可由 C 语言编写。后面给出了 C 语言程序和相应的汇编程序。

1.3.2　中断

当中断程序位于主 48K 页时，具有正常的等待时间(latency)，因为 CPU 总是可以访问这

48K 字节页，翻页中断程序可造成 25 个周期的延迟。

1.3.3 重要条件

这种方法要求向量必须指向换页程序位于的存储器的 48K 主页。处于其他页的程序只能通过 48K 页才能转移到另一页。

表 1-1 软硬件结合方法的页选择

PD5	A14	A15	A16	存储器页
0	0	0	1	由 PD3 和 PD4 决定页 1～页 4(每页 16K)
	0	1	0	48K 主页
	1	0	0	
	1	1	0	
1	0	0	0	页 0(每页 16K)
	0	1	0	48K 主页
1	0	0		
	1	1	0	

象方法 A 一样，RAM、寄存器以及 EEPROM，都出现在存储器映象中，因此，必须注意避免这些地址，或将 RAM 和寄存器地址移到不同的地址。

图 1-5 是软硬件方法分页示意图

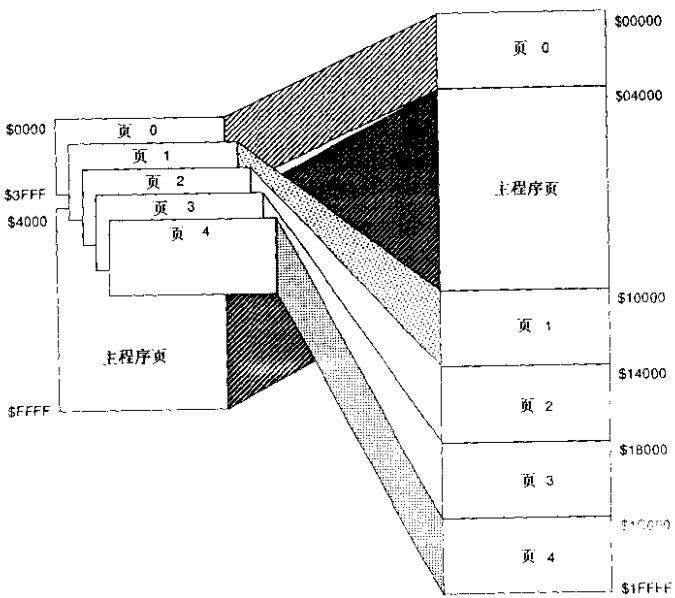


图 1-5 软硬件结合分页方法示意图

可以容易地改变主程序页和各页的大小，以适于不同的应用场合。例如，不用 PD3 控制 E-PROM 的地址线 A14，则主程序页为 32K 字节，另外 3 页也是 32K 字节。类似地，增加 I/O 口

线 PD2 来控制地址线 A13，则主程序页为 56K 字节，另有 9 页，每页为 8K 字节。

1.3.4 软硬件结合分页方法程序清单

```
*  128K 字节 EPROM 分为 48K 字节公共页 $4000～$FFFF 和页 0～页 1
*  复位后 PD3～PD5 均为 1，为主程序页和页 0
PROTA     EQU  $00
DDRA      EQU  $01              68HC11G5  only
PORTB     EQU  $04
PORTC     EQU  $06
DDRC      EQU  $07
PORTD     EQU  $08
DDRD      EQU  $09
TMSK2     EQU  $24
TFLG2     EQU  $25
RTII      EQU  $40
RTIF      EQU  $40
PACTL     EQU  $26
DDRA7     EQU  $80              68HC11E9  only
REGS      EQU  $1000
*  RAM 定义
          ORG $0000
TIME      RMB  2          实时中断程序计数器
*
ROMBASE0  EQU  $0200      避免 RAM($0～$1FF)
ROMBASE1  EQU  $4000
VECTORS   EQU  $FFCC
    *     PAGE 0=$00000～$03FFF    (A16=0,15=0,A14=0)
    *     MAIN  =$04000～$0FFFF    (A16=0)
    *     PAGE 1=$10000～$13FFF    (A16=1,A15=0,A14=0)
    *     PAGE 2=$14000～$17FFF    (A16=1,A15=0,A14=1)
    *     PAGE 3=$18000～$1BFFF    (A16=1,A15=1,A14=0)
    *     PAGE 4=$1C000～$1FFFF    (A16=1,A15=1,A14=1)
  START   EQU  $00
  PAGE0   EQU  $20
  PAGE1   EQU  $00
  PAGE2   EQU  $08
  PAGE3   EQU  $10
  PAGE4   EQU  $18
    * 页 0
          org                      ROMBASE0
LOOPP0 BSET PORTA,Y,#$08
    BCLR  PORTA,Y,#$08             口 A－3 翻转
    JMP   MAIN0                    返回主程序
*  主程序开始
          ORG  ROMBASE1
  RESET   LDS #$01
```

```
            JSR SETUP                初始化 RTI 中断和 DDR
LOOP        BSET PORTD,Y,#$40
            BCLR PORTD,Y,#$40        主程序使 PD2 翻转
            JSR CHGPAGE0             选页 0
            JMP LOOPP0               口 A－3 翻转
MAIN0       JSR CHGPAGE1             选页 1
            JSR LOOPP1               口 A－4 翻转
            JSR CHGPAGE2             选页 2
            JSR LOOPP2               口 A－5 翻转
            JSR CHPAGE3              选页 3
            JMP LOOPP3               口 A－6 翻转
MAIN3       JSR CHGPAGE4             选页 4
            JMP LOOPP4               口 A－7 翻转
MAIN4       BRA LOOP                 再次开始循环
*初始化程序
SETUP       SEI
            LDY  #$1000              寄存器地址偏移
            LDAA #$FF
            STAA DDRA+REGS           A 口均为输出(HC11G5)
            STAA DDRD+REGS           D 口均为输出
            CLR  TIME
            CLR  TIME+1
            CLRA
            STAA  PORTA+REGS
            STAA  PORTD+REGS
            LDAA  #%01000000
            STAA  TFLG2+REGS         RTI 标志清零
            STAA   TMSK2+REGS        允许 RTI 中断
            CLI
            RTS
* 实时中断服务程序
RTISRV      LDAA#%01000000
            STAA TFLG2+REGS          RTI 标志清零
            LDAA TIME+1
            STAA PORTB+REGS          计数器存入 B 口
            LDX TMIE                 取时间计数器
            INX
            STX TIME                 计数器值存入 RAM
            RTI                      从中断返回
*换页
CHGPAGE0
            LDAA PORTD+REGS          取 D 口数据
            ANDA #%11000111          使中间 3 位为低电平
            ADDA #PAGE0
            STAA PORTD+REGS          写回 D 口(只有位 3~5 变化)
            RTS
```

```
CHGPAGE0
        LDAA    PORTD+REGS
        ANDA    #%11000111
        ADDA    #PAGE0
        STAA    PORTD-REGS
        RTS
CHGPAGE1
        LDAA    PORTD+REGS
        ANDA    #%11000111
        ADDA    #PAGE1
        STAA    PORTD+REGS
        RTS
CHGPAGE2
        LDAA    PORTD+REGS
        ANDA    #%11000111
        ADDA    #PAGE2
        STAA    PORTD+REGS
        RTS
CHGPAGE3
        LDAA    PORTD+REGS
        ANDA    #%11000111
        ADDA    #PAGE3
        STAA    PORTD+REGS
        RTS
CHGPAGE4
        LDAA    PORTD+REGS
        ANDA    #%11000111
        ADDA    #PAGE4
        STAA    PORTD+REGS
        RTS
```

* 向量

```
ORG     VECTORS
FDB     RESET     EVENT 2
FDB     RESET     EVENT 1
FDB     RESET     TIMER OVERFLOW 2
FDB     RESET     INPUT CAPTURE 6 / OUTPUT COMPARE 7
FDB     RESET     INPUT CAPTURE 5 / OUTPUT COMPARE 6
FDB     RESET     SCI
FDB     RESET     SPI
FDB     RESET     PULSE ACC INPUT
FDB     RESET     PULSE ACC OVERFLOW
FDB     RESET     TIMER OVERFLOW 1
FDB     RESET     INPUT CAPTURE 4 / OUTPUT COMPARE 5
FDB     RESET     OUTPUT COMPARE 4
FDB     RESET     OUTPUT COMPARE 3
```

```
FDB     RESET     OUTPUT COMPARE 2
FDB     RESET     OUTPUT COMPARE 1
FDB     RESET     INPUT CAPTURE 3
FDB     RESET     INPUT CAPTURE 2
FDB     RESET     INPUT CAPTURE 1
FDB     RTISRV    REAL TIME INTRRUPT
FDB     RESET     IRQ
FDB     RESET     XIRQ
FDB     RESET     SWI
FDB     RESET     ILLEGAL OPCODE
FDB     RESET     COP
FDB     RESET     CLOCK MONITOR
FDB     RESET     RESET
```

```
* 页 1
        org     ROMBASE0
LOOPP1  BSET    PORTA,Y,#$10
        BCLR    PORTA,Y,#$10    口 A—4 翻转
        RTS
* 页 2
        org     ROMBASE0
LOOPP2  BSET    PORTA,Y,#$20
        BCLR    PORTA,Y,#$20    口 A—5 翻转
        RTS
* 页 3
        org     ROMBASE0
LOOPP3  BSET    PORTA,Y,#$40
        BCLR    PORTA,Y,#$40    口 A—6 翻转
        JMP     MAIN3           返主程序
* 页 4
        org     ROMBASE
LOOPP4  BSET    PORTA,Y,#$80
        BCLR    PORTA,Y,#$80
        JMP     MAIN
            ***************
        END
```

1.3.5 方法 B 的 C 语言程序

```
*       /*CHGPAGE.C
*         C coded extended memory control for 68HC11
*
*       */
                    **********************
/*      HC11 structure — I/O registers for MC68HC11 */
```

```
struct HC11IO {
unsigned char   PORTA;        /* Port A _ 3 input only, 5 output only */
unsigned char   Reserved;     /* Motorola's unknown register */
unsigned char   PIOC;         /* Parallel I/O control */
unsigned char   PORTC;        /* Port C */
unsigned char   PORTB;        /* Port B — Output only */
unsigned char   PORTCL;       /* Alternate port C latch */
unsigned char   Reservedl;    /* Motorcla's unknown register 2 */
unsigned char   DDRC;         /* Data direction for port C */
unsigned char   PORTD;        /* Port D */
unsigned char   DDRD;         /* Data direction for port D */
unsigned char   PORTE;        /* Port E */
};
/*     End of structure HC11IO */
                        **********
 *      #define regbase (*(struct HC11IO *) 0×1000)
 *      type of unsigned char byte;
 *
 *      /* Some arbitrary user defined values */
 *      #define page0 0×20
 *      #define page1 0×00
 *      #define page2 0×08
 *      #define pagemask 0×c7
 *
 *      /* Macro to generate in line code */
 *      #define chgpage(a) regbase, PORTD = (regbase.PORTD & pagemask) + a
 *
 *      /* Function prototype */
 *      void func _chgpage (byte p);
 *
 *      /* Externally defined functions in separate pages */
 *      extern void func _in _page0();   /* Dummy function in page 0 */
 *      extern void func _in _page2();   /* Dummy function in page 2 */
 *      ·····compiled assembly code ····· C source code ·····
 *
*                                                    main()
 6 0000              main: fbegin
 *                                                    {
*                                                    chgpage(page 2);
                                                     /* Change page using inline code */
 8 0000  f61008      ldab      $1008
 9 0003 c4c7         andb      #199
 10 0005 cb08        addb      #8
 11 0007 f71008      stab      $1008
```

```
                                        func _in _page2();
*                                       /* Call function in page 2 */
 13 000a >bd0000    jsr     func _in _page2
*                                       func _chgpage(page0);
                                        /* Change page using function call */
 15 000d cc0020     ldd #32
 16 0010 8d04       bsr     func _chgpage
*                                       func _in _page0();
                                        /* Call function in page 0 */
 18 0012 >bd0000    jsr     func _in _page0
*                                       }
 20 0015 39         rts
 21 0016            fend
*
                                        void func _chgpage(p)
*                                       byte p;
24 0016                     func _chgpage:  fbegin
25 0016 37         pshb
*                                       {
*                                       chgpage(p);
 27 0017 f61008     ldab    $1008
 28 001a c4c7       andb    #199
 29 001c 30         tsx
 30 001d eb00       addb    0,x
 31 001f f71008             stab $1008
  *                         }
 33 0022 31         ins
 34 0023 39         rts
 35 0024            fend
 36                 import  func _in _page2
 37                 import  func _in _page0
 38                 end
```

1.4 总结

在这两种方法中，寄存器可能移到更适当的地址。如果 RAM 的使用不严格，可在复位后立即向 INIT 寄存器写入 $00 使寄存器移到地址 $0000。该例中，寄存器和 RAM 保留在省缺地址，并且应注意，用户程序不能位于地址 $0000～$01FF 和 $1000～$107F（对于 HC11G5）。HC11E9 和 HC11A8 与 HC11G5 稍有不同。

图 1-6 是两种分页方法的比较。通过改变翻页程序和硬件，可容易地修改页数和页的大小。

当寻址范围超过 128K 字节时，这两种方法都可用 I/O 线来控制 A15 以上的地址线，相应地软件也应适当地修改。

这里介绍的两种方法是通过翻页控制一个大的 EPROM 或几个 EPROM 的基本方法。可方便地对这种方法进行修改，以适应不同的应用场合。软件方法成本低，软硬件结合的方法中

断速度快，无额外的等待时间。

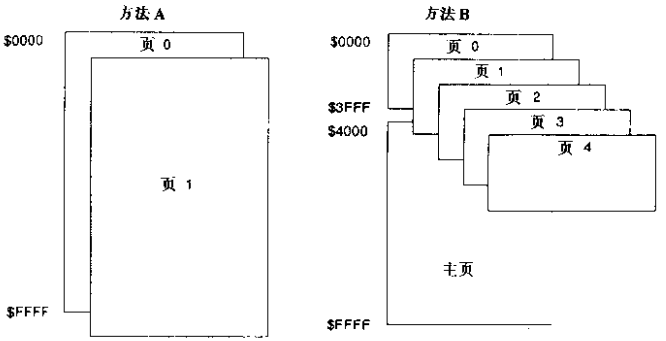


图 1-6 分页方法比较

2. MC68HC11K4 的存储器映象扩展

MC68HC11K4 具有存储器扩展能力，可以使 CPU 的寻址范围由 64K 字节扩展到 1M 字节，这里介绍扩展方法和硬件结构。

2.1 MC68HC11K4 存储器扩展逻辑

存储器扩展逻辑通过两个片内逻辑块扩展 CPU 的寻址范围。第一个逻辑块提供额外的地址线，它们只有在 CPU 需要时才有效。第二个逻辑块提供片选信号，使得易于与外部存储器接口。这两个逻辑块全部都是用户可编程的。

2.2 额外的寻址能力

扩展 68HC11 CPU 寻址能力的第一步就是提供额外的地址线。CPU 本身有 16 条地址线(A0～A15)，可寻址 64K 字节，每增加一条地址线可使寻址范围增加到原来的两倍。

为保持与 M68HC11 系列 CPU 的相容性，K4 所用的 CPU 不变，通过转换额外存储器的存储区(bank)来扩展寻址能力。为使这种方法具有最大的灵活性，转换的存储区块大小和数目都是用户可编程的。

为使用扩展的存储器，必须首先确定接入和断开的存储器区块的范围(称为窗口)。存储器扩展逻辑允许窗口定义为 8K、16K、32K 字节。在任何时候，只有一个存储区块显示在定义的窗口，因此，CPU 在每个时刻只能访问一个存储区块的内容。显示在窗口的存储区块由额外的地址线选择。用一个新的存储区块代替另一个的过程称为区域转换。

这如同幻灯片和幻灯机的情况一样，可以有很多的幻灯片，但在任何时候只能观看一张幻灯片。如果幻灯片都编上号，则可以选出所需观看的一张，而不必翻遍所有的幻灯片。存储区块就如同幻灯片，幻灯片的编号就如同地址。没有被显示的幻灯片可以替代正在观看的那张幻灯片。

为扩展 CPU 的寻址能力，增加有 6 条地址线：XA13、XA14、XA15、XA16、XA17 和 XA18。为使窗口大小具有最大灵活性，低 3 位地址线只用于取代相应的 CPU 地址。

为理解 XA13～XA15 的作用，考虑窗口为 8K 字节时的情况。当 CPU 正在访问窗口内的存储器时，XA16～XA18 才有效。XA16～XA18 可使 CPU 窗口的存储器增加 8 倍。当窗口为 8K 字节时，则 CPU 最多访问 8×8K＝64K 字节。为识别 8K 字节，CPU 只需要使用地址线 A0～A12。A13～A15 的变化将脱离窗口。3 条新地址线 XA13～XA15 可实现 512K 字节(8×8×8K)范围寻址。对于 16K 字节窗口，XA13～XA15 中只有 XA14 和 XA15 有用；对于 32K 字节的窗口，只有 XA15 有用。注意，存储器窗口的大小不一定必须在硬件设计过程中定义，因为可以将额外地址 XA13～XA15 编程，使其传输 A13～A15 信号。

存储器扩展逻辑实际上允许用户定义两个独立的窗口，因此，可以访问 1M 字节以上的存储器。

实现这么大的存储器范围时，由于采用了存储器扩展片选逻辑，所需的硬件大为简化。

2.3 片选

提供存储器扩展片选目的是为了使 MC68HC11K4 易于与外部存储器接口。4 个片选中有 3 个直接与存储器扩展逻辑有关。其中两个为通用片选，一个为程序片选，另一个为 I/O 片选。I/O 片选是为了简化外部外围芯片。

片选的基本功能是提供逻辑信号，以表明正在访问存储器的某一区域。例如，在主程序存储器范围内时，程序片选认为有效。当使用其他存储器区域时，相应的片选将有效。

通用片选最灵活，其功能与存储器扩展逻辑密切相关。它们可编程为在 CPU 64K 字节范围内有效，或编程为在任一个窗口的 512 字节范围内有效。在这两种情况下，所选的存储器大小从 2K 字节到 512K 字节完全可编程。

2.4 存储器扩展和片选寄存器

下面介绍有关存储器扩展逻辑和片选功能的内部寄存器。

MMWBR 寄存器用于定义 CPU 64K 字节地址范围内的两个窗口中每个窗口的起始地址。通常，窗口的起始地址在其它字节块的边界开始，例如，窗口为 8K 字节时，则从 8K 字节边界开始，即 $0000、$2000、$4000、…、$E000 等。16K 字节窗口则从 16K 字节边界开始，如 $0000、$4000、…、$C000。32K 字节窗口有些例外，通常其起始地址为 $0000 或 $8000，但该窗口总是位于 $4000 起始地址。

MMSIZ 寄存器设置所用的窗口大小，并选择片选是只对于 CPU 地址有效还是对外部扩展地址有效。

每个通用片选都有两个寄存器，分别称为 GP1CSC 和 GP1CSA，GP2CSC 和 GP2CSA。

控制寄存器 GPXCSCS 决定选中某区域存储器所需的逻辑输出(可能与其他片选进行逻辑组合)和片选有效的存储器范围。每个片选都可编程为：每当 CPU 地址进入到存储器扩展窗口时，片选有效(与实际选择的存储区块无关)。

地址寄存器 GPXCSA 决定被编程的片选的起始地址。该寄存器中有效的位取决于控制寄存器选中的片选范围的大小。

程序和 I/O 片选通过 CSCTL 寄存器是可编程的。

两个窗口寄存器 MM1CR 和 MM2CR 用于表明窗口中哪一个区块(bank)有效。每个寄存器都含有当 CPU 选中扩展存储器窗口内的地址时，所要输出的 XA13～XA18 的值。

用户向适当的窗口寄存器写入新的区块的地址，就可改变区块。

实际的存储器扩展地址线与 G 口引脚公用。G 口引脚作为地址线时，G 口引脚就不能作为 I/O。因此，用户只需要选择扩展逻辑所需的地址线，不用的线仍可作为 G 口通用 I/O。PGAR 寄存器决定使用哪些扩展地址线。若不需要某条扩展地址线，则 PGAR 相应的位应该清零。

2.5 存储器扩展实例

下面看一些扩展 K4 存储器的实例。每个例子中都有两个图，一个是存储器结构图，另一个是硬件结构图。

例 1：图 2-1 和图 2-2 分别是 8K 字节窗口和硬件图。64K 字节 EPROM 分为 8 个存储区块。注意逻辑地址为 A0～A12 和 XA13～XA15。使用片选 1。

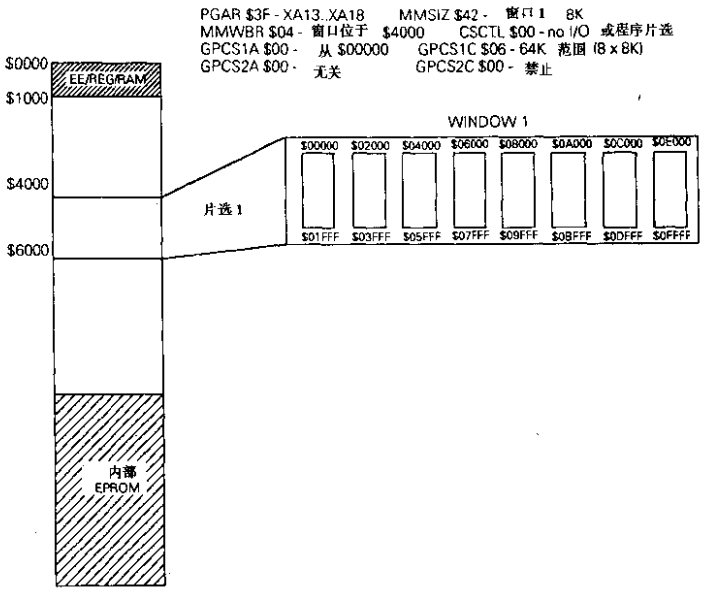


图 2-1 一个 8K 字节窗口

例 2：如图 2-3 和图 2-4 所示，具有两个窗口的存储器结构和硬件结构。其中 8K 字节窗口与例 1 类似。第二个窗口为 16K 字节，两片 128K 字节 RAM 构成 16 个存储区块。第二个窗口的逻辑地址由 A0～A13 和 A14～A17 决定(XA18 决定起始地址)。片选 2 用于第二个窗口。

例 3：如图 2-5 和图 2-7 所示，仍为两个窗口。与例 2 所不同的是，窗口的逻辑地址改变了。现在，第一个窗口的逻辑地址由 A0～A12 和 XA15～XA17 决定(XA13～XA14 无关)，第二个窗口的逻辑地址由 A0～A13 和 XA15～XA18 决定(XA14 无关)。注意，由于没有对某些地址线进行译码，两个窗口中的每个存储区块都有重复。第一个窗口每个区块为四重重复，第二个窗口为两重重复。

例 4：如图 2-6 和图 2-8 所示，是扩展最大 1M 字节存储器情况，第一个窗口(窗口 1)为 16K 字节，从 $0000 开始；窗口 2 为 32K 字节，从 $4000 开始。注意，内部 RAM 和寄存器被映射到窗口 1 的每一页，但窗口 2 的 EPROM 只出现在扩展存储器的第一个 64K 字节内。也就是，对于低于 $10000 的地址，内部 EPROM 出现在有关地址的存储器映象中。

两个窗口的逻辑地址均由 A0～A13 和 XA14～XA18 决定。因为窗口 2 不是从 32K 字节边界开始，故需用 XA14。两个片选都使用，分别对应一个窗口。

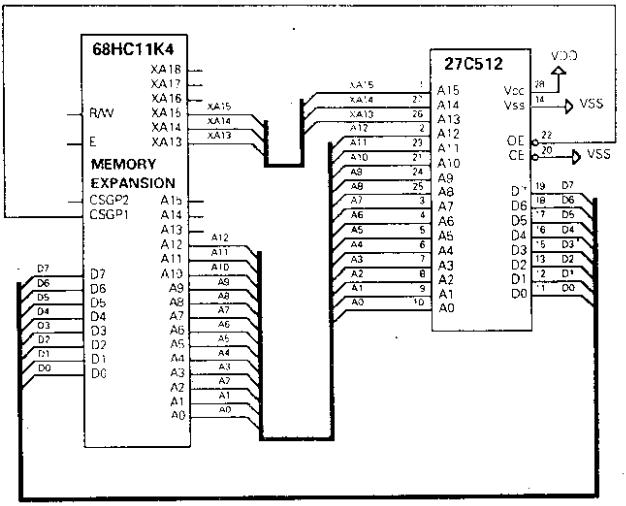


图 2-2　8K 字节窗口的硬件结构

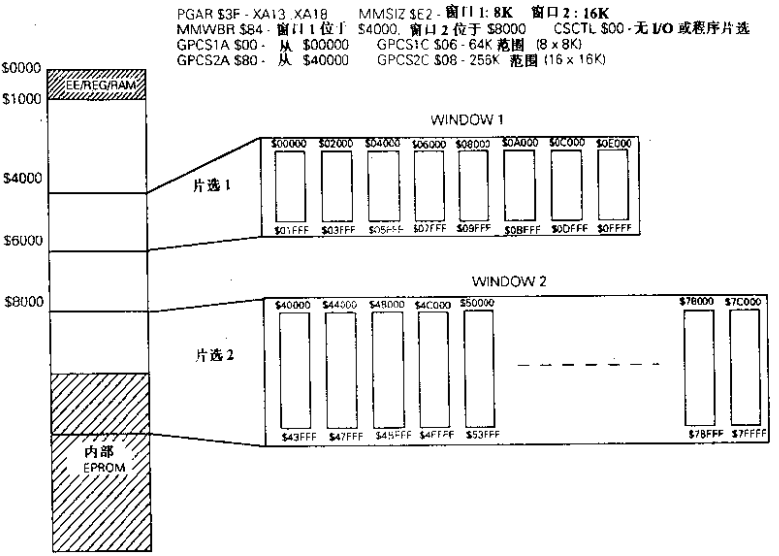


图 2-3　一个 8K 字节窗口和一个 16K 字节窗口

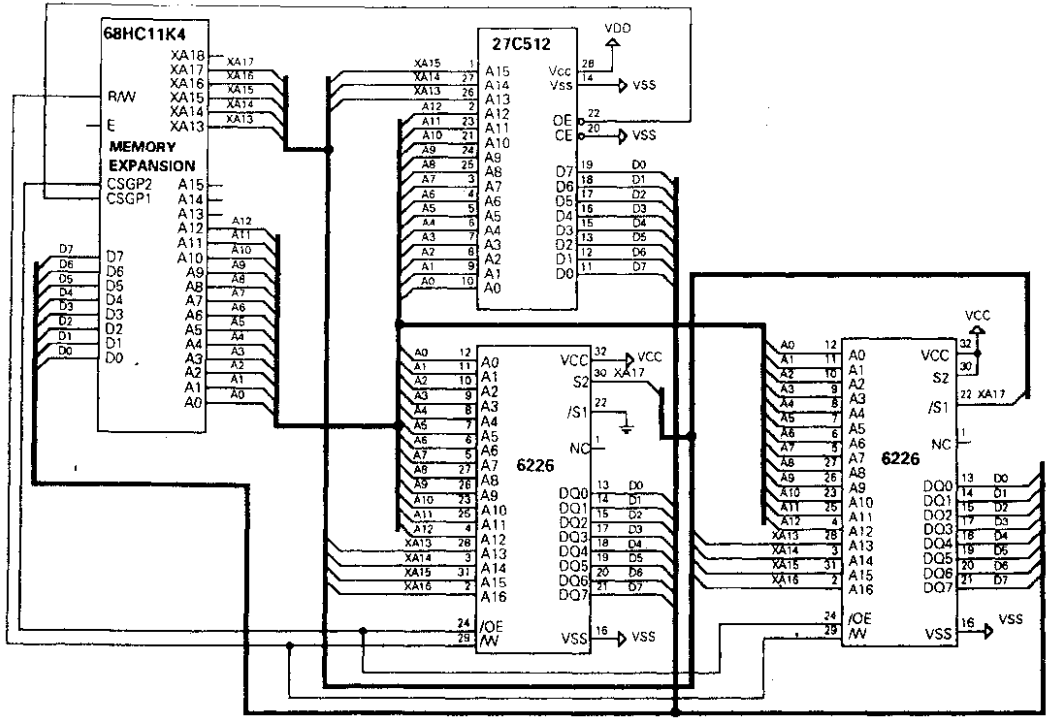


图 2-4　一个 8K 字节窗口和一个 16K 字节窗口的硬件结构

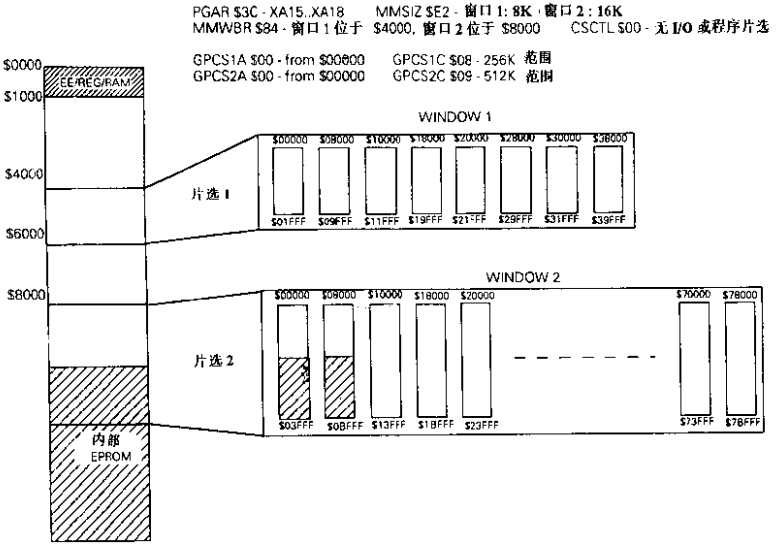


图 2-5 8K 字节窗口和 16K 字节窗口(与例 2 地址不同)

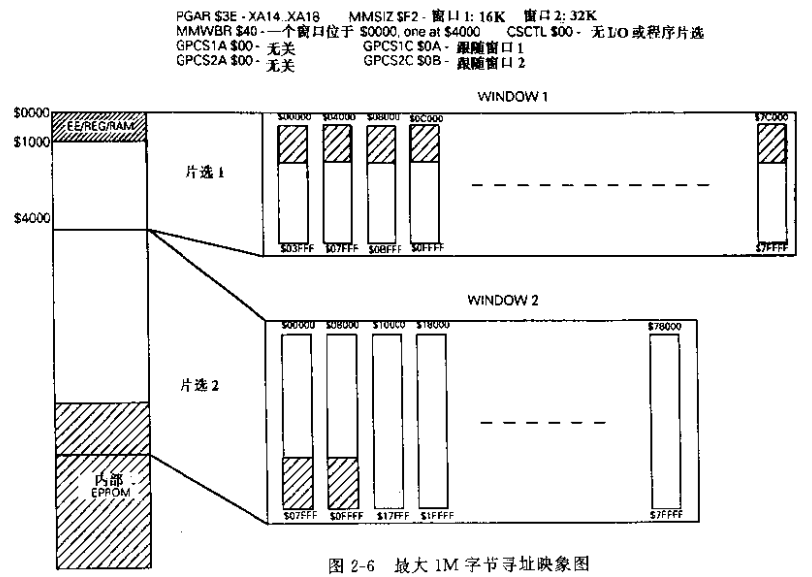


图 2-6 最大 1M 字节寻址映象图

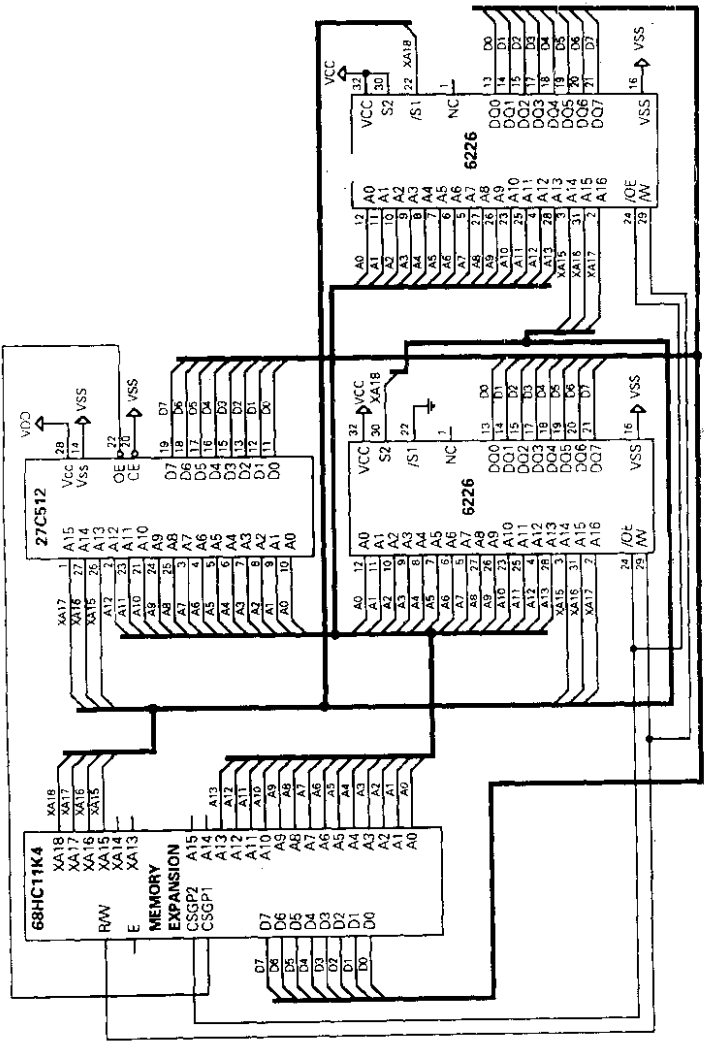


图 2-7 另一种 8K 字节窗口和 16K 字节窗口的硬件结构

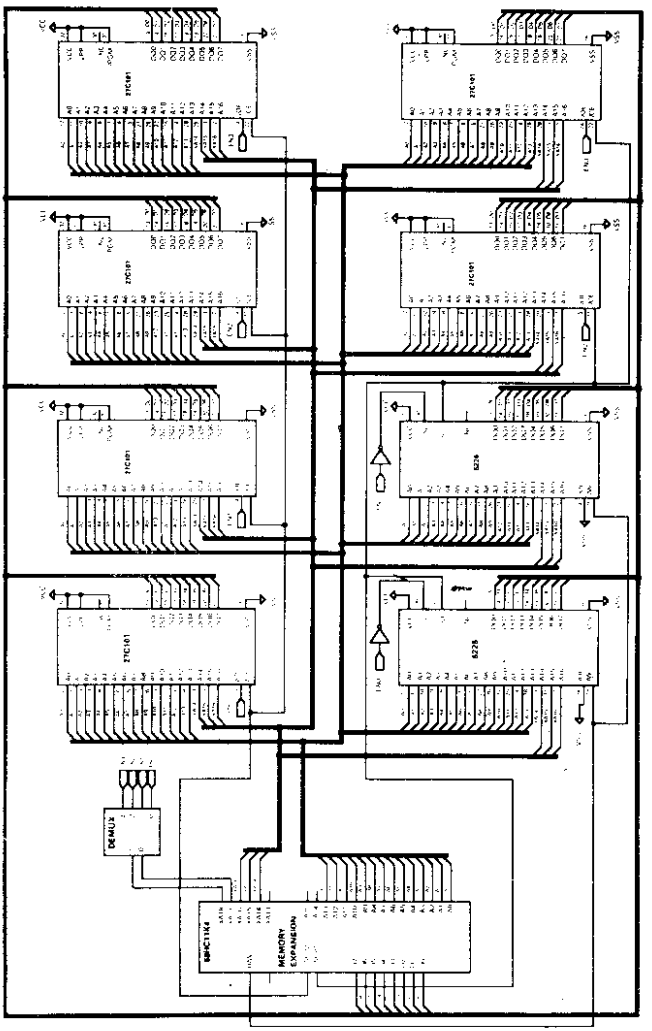


图 2 8　1M 字节寻址的硬件图

3. M68HC11 浮点运算软件包

M68HC11 是功能很强的单片机，具有 6 种寻址模式、位处理、16 位算术操作和 16 位变址寄存器，特别适于需要高速 I/O 和高速运算的控制场合。

在有些应用场合必须进行浮点运算。浮点运算可具有更快的速度和更大的灵活性。HC11 浮点软件包(HC11FP)除包括 4 种基本运算(加、减、乘、除)外，还具有将 ASCII 转换为浮点和将浮点转换成 ASCII 的程序、3 种基本三角函数(正弦、余弦和正切)以及度数和弧度之间相互转换的程序，也包括平方根函数和整数与浮点数相互转换的程序。

整个浮点软件包需要 2K 字节多的存储器，并且只需要 10 个字节零页 RAM 和堆栈 RAM。浮点软件程序所需的所有临时变量都位于堆栈。在使用任何程序之前只需保存 10 个字节零页 RAM，就可重新进入，从而允许中断程序和主程序使用浮点软件包，而相互没有影响。

3.1 浮点格式和浮点累加器格式

10 字节零页 RAM 用于两个浮点累加器：FPACC1 和 FPACC2。每个累加器为 5 字节，其中一个字节为指数，3 个字节为尾数，一个字节用来表示尾数的符号。指数字节用于表明二进制小数点的位置，用它进行十进制 128 偏移($80 偏移)，以便于浮点数进行比较。一字节指数给出的动态范围约 $1\times10^{\pm38}$。

尾数为 3 字节(24 位)，用于表示浮点数的整数和小数部分。尾数总是归一化的，即最高位字节的最高位总为 1。24 位尾数的精度大约相当于 7 位十进制数的精度。

另有一字节表明尾数的符号，这样，尾数并不是补码形式。因此，处理尾数时，就可以按无符号数运算，该字节为零($00)表明尾数为正；该字节为负 1($FF)表明尾数为负。

```
FPACC1    82  C90FDB  00     +3.1415927
FPACC2    82  C90FDB  FF     -3.1415927
```

3.2 存储器格式

浮点数存储在存储器中的方式(或称为浮点数格式)与浮点累加器格式稍有不同。为节省存储器，浮点数以“隐藏位归一化格式”存储在存储器中。在这种格式中，用 4 个连续字节存储一个浮点数，指数位于最低地址。尾数在随后的 3 个字节中，最高位字节存储在最低地址。由于归一化浮点数尾数的最高位总为 1，故该位可用来存储尾数的符号。最高位为 0 表明尾数为正，最高位为 1 表明尾数为负。例如：

```
82 c90 FDB     +3.1415927
82 c90 FDB     -3.1415927
```

有 4 个程序可用来保存和装入浮点累加器，并且同时在浮点累加器格式和存储器格式之间转换。

3.3 错误

HC11 浮点软件包可能有 7 种错误状态。发生错误时，CCR 的进位位置位并且累加器 A 返回错误码，以表明错误情况。错误码及其意义如表 3-1 所示。

表 3-1 错误码及意义

错误号	意　义
1	ASCII 至浮点转换格式错误
2	浮点溢出(上溢)
3	浮点下溢
4	除以 0
5	浮点数太大或太小,不能转换为整数
6	负数取平方根
7	π/2(90°)的正切

3.4 浮点运算与 ASCII 转换程序

下面浮点软件包中的每个程序包括子程序名、所执行的操作、子程序的大小、所需的堆栈空间、调用的其他子程序、输入、输出及可能的错误状态。

子程序所需的堆栈空间不仅包括某程序局部变量所需的堆栈空间,还包括被调用的其他子程序所需的堆栈空间(包括返回地址)。三角函数需要大量堆栈空间。

某些应用场合不一定需要浮点软件包中所有的程序,下面程序中含有所调用的其他子程序名,用户可便于取舍。

• ASCII 码转换为浮点

子程序名:　ASCFLT
操作:　ASCII(X)→FPACC1
占用空间:　352 字节(含 NUMERIC 子程序)
堆栈空间:　14 字节
调用子程序:　NUMERIC、FPNORM、FLTMUL、PSHFPAC2、PULFPAC2
输入:　X 寄存器指向 ASCII 串
输出:　FPACC1 含有浮点数
错误状态:　可能返回浮点格式错误
注:　当遇到非十进制字符时停止转移。输入格式非常灵活,例如:20.095,0.125,7.2984E+10,167.824E5,005.9357E−7,500 等。

• 浮点数相乘

子程序名:　FLTMUL
操作:　FPACC1 X FPACC2→FPACC1
占用空间:　169 字节
堆栈空间:　10 字节
调用子程序:　PSHFPAC2、PULFPAC2、CHCK0
输入:　FPACC1 和 FPACC2 含有相乘的数
输出:　FPACC1 含有两浮点数相乘的积,FPACC2 不变
错误状态:　上溢、下溢

• 浮点数相加

子程序名:　FLTADD
操作:　FPACC1+FPACC2→FPACC2
占用空间:　194 字节

堆栈空间：　6 字节
调用子程序：　PSHFPAC2、PULFPAC2、CHCK0
输入：　FPACC2 和 FPACC1 含有相加的浮点数
输出：　FPACC1 含有两数之和，FPACC2 不变
错误状态：　上溢、下溢
注：　浮点相加执行全有符号相加。两浮点累加器可能具有不同符号。

• 浮点数相减

子程序名：　FLTSUB
操作：　FPACC1－FPACC2→FPACC1
占用空间：　12 字节
堆栈空间：　8 字节
调用子程序：　FLTADD
输入：　FPACC1 和 FPACC2 含有相减的数
输出：　FPACC1 含有差，FPACC2 不变
错误状态：　上溢，下溢
注：　先改变 FPACC2 的符号，再调用 FLTADD，然后恢复 FPACC2 原来的符号

• 浮点数相除

子程序名：　FLDIV
操作：　FPACC1/FPACC2→FPACC1
占用空间：　209 字节
堆栈空间：　11 字节
调用子程序：　PSHFPAC2，PULFPAC2
输入：　FPACC1 和 FPACC1 含有被除数和除数
输出：　FPACC1 含有商，FPACC2 不变
错误状态：　除以零，上溢，下溢

• 浮点数转换为 ASCII

子程序名：　FLTASC
操作：　FPACC1→(X)
占用空间：　370 字节
堆栈空间：　28 字节
调用子程序：　FLTMUL、FLTCMP、PSHFPAC2、PULFPAC2
输入：　FPACC1 含有需转换为 ASCII 的数
输出：　变址寄存器 X 所指的缓冲器含有代表 FPACC1 的 ASCII 串。
错误状态：　无

• 浮点数比较

子程序名：　FLTCMP
操作：　FPACC1－FPACC2
占用空间：　42 字节
堆栈空间：　无
调用子程序：　无
输入：　FPACC1 和 FPACC2 含有被比较的数
输出：　CCR 被相应地置位，坷用所有的分支指令改变程序流。FPACC1 和 FPACC2 不变

错误状态：　　无

• 无符号整数转换为浮点数

子程序名：　　UINT2FLT
操作：　　(16 位无符号整数)→FPACC1
占用空间：　　16 字节
堆栈空间：　　8 字节
调用子程序：　　FPNORM,CHCK0
输入：　　FPACC1 尾数的低 16 位含有无符号 16 位整数
输出：　　FPACC1 含有浮点数
错误状态：　　无

• 有符号整数转换为浮点数

子程序名：　　SINT2FLT
操作：　　(16 位有符号整数)→FPACC1
占用空间：　　24 字节
堆栈空间：　　7 字节
调用子程序：　　UINT2FLT
输入：　　FPACC1 尾数的低 16 位含有 16 位有符号整数
输出：　　FPACC1 含有浮点数
错误状态：　　无

• 浮点数转换为整数

子程序名：　　FLT2INT
操作：　　FPACC1→(16 位有符号或无符号整数)
占用空间：　　74 字节
堆栈空间：　　2 字节
调用子程序：　　CHCK0
输入：　　FPACC1 可能含有浮点数，范围为 65535≤FPACC1≥−32767
输出：　　FPACC1 尾数的低 16 位含有 16 位有符号或无符号数
注：　　若 FPACC1 的浮点数为正，则转化为无符号数；若为负，则转化为有符号二进制初码整数。这样可允许 16 位地址浮点格式的正数表示。忽略任何小数。

• FPACC1 传输至 FPACC2

子程序名：　　TFR 1 TO 2
操作：　　FPACC1→FPACC2
占用空间：　　13 字节
堆栈空间：　　无
输入：　　FPACC1 含有浮点数
输出：　　FPACC2 含有与 FPACC1 相同的浮点数
错误状态：　　无

3.5 浮点函数

下面介绍浮点函数及其返回结果和错误状态。

· 平方根

子程序名：	FLTSQR
操作：	SQR(FPACC1)→FPACC1
占用空间：	104 字节
堆栈空间：	21 字节
调用子程序：	TFR1 TO 2、FLTDIV、FLTADD、PSHFPAC2、PULFPAC2
输入：	FPACC1 含有有效浮点数
输出：	FPACC1 含有原数的平方根，FPACC2 不变
错误状态：	若 FPACC1 为负，则返回 NSQRTERR，并且 FPACC1 不变。

· 正弦

子程序名：	FLTSIN
操作：	SIN(FPACC1)→FPACC1
占用空间：	380 字节(含 SINCOS 子程序)
堆栈空间：	50 字节
调用子程序：	ANGRED、SINCOS、DEG2RAD、PSHFPAC2、PULFPAC2
输入：	FPACC1 含有以弧度表示的角度，$-2\pi \leqslant FPACC1 \leqslant +2\pi$
输出：	FPACC1 含有原浮点数的正弦值，FPACC2 不变
错误状态：	无
注：	用泰勒(Taylor)展开级数计算角度在 0～45°(π/4)的正弦值。调用子程序 ANGRED 将输入角度减小到这一范围。整个输入范围的最大误差为 1.5×10^{-7}。

· 余弦

子程序名：	FLTCOS
操作：	COS (FPACC1)→FPACC1
占用空间：	384 字节(含 SINCOS 子程序)
堆栈空间：	50 字节
调用子程序：	ANGRED、FLTSIN、DEG2RAD、PSHFPAC2
输入：	FPACC1 含有角度值，范围为 $-2\pi \leqslant FPACC1 \leqslant +2\pi$
输出：	FPACC1 含有余弦值，FPACC2 不变
错误状态：	无
注：	同正弦函数

· 正切

子程序名：	FLTTAN
操作：	TAN(FPACC1)→FPACC1
占用空间：	35 字节(还需要 FLTSIN 和 FLTCOS)
堆栈空间：	56 字节
调用子程序：	TFR1TO2、EXG1AND2、FLTSIN、FLTCOS、FLTDIN、PSHFPAC2、PULFPAC2
输入：	FPACC1 含有角度，范围为 $-2\pi \leqslant FPACC1 \leqslant +2\pi$
输出：	FPACC1 含有输入角度的正切值，FPACC2 不变
错误状态：	求 $\pm\pi/2$ 的正切值返回最大合法数

注：先获得输入角度的正弦和余弦值，再由 TAN=SIN/CON 获得正切值。在 89.9°时正切函数的精度只有 5 位十进制数，大于 89.3°时，精度迅速下降

- 角度转换为弧度

子程序名：DEG2RAD
操作：FPACC1×π/180→FPACC1
占用空间：15 字节
堆栈空间：16 字节
调用子程序：GETFPAC2、FLTMUL
输入：以度数表示的任何有效的浮点数角度
输出：弧度
错误状态：无

- 弧度转换为度数

子程序名：RAD2DEG
操作：FPACC1×180/π→FPACC1
占用空间：8 字节（还需要 DEG2RAD 子程序）
堆栈空间：16 字节
调用子程序：DEG2RAD
输入：以弧度表示的任何有效的浮点数角度
输出：度数
错误状态：上溢，下溢

- 圆周率(PI)

子程序名：GETPI
操作：π→FPACC1
占用空间：6 字节
堆栈空间：无
调用子程序：无
输出：FPACC1 含有 π 的值
错误状态：无

3.6 格式转换程序

已经知道，浮点数在浮点累加器中的格式和存储器中的格式不同，目的主要是为了减少所需的存储器。下面的 4 个程序可改变浮点数格式，同时，将浮点数移入或移出浮点累加器。

- 获得 FPACC1 或 FPACC2

子程序名：GETFPAC1 或 GETFPAC2
操作：(X)→FPACC1；(X)→FPACC2
占用空间：22 字节
堆栈空间：无
输入：X 变址寄存器指向的存储器格式数移入浮点累加器
输出：X 所指的数位于浮点累加器
错误状态：无

- 存 FPACC1 和 FPACC2

子程序名：PUTFPAC1 和 PUTFPAC2

操作：　　FPACC1→(X)；FPACC2→(X)
占用空间：　　22字节
堆栈空间：　　无
输入：　　变址寄存器X指向存入浮点数的4个连续的存储器单元
输出：　　浮点累加器存入X所指的存储器单元
错误状态：　　无

3.7 程序清单

```
        ORG   $0000
FPACC1EX    RMB 1                   浮点累加器#1
FPACC1MN    RMB 3
MANTSGN1    RMB 1                   FPACC1的尾数符号(0为正,FF为负)
FPACC2EX    RMB 1                   浮点累加器#2
FPACC2MN    RMB 3
MANTSGN2    RMB 1                   FPACC2的尾数符号
FLTFMTER    EQU 1                   ASCFLT浮点格式错误
OVFERR      EQU 2                   浮点溢出错误
UNFERR      EQU 3                   浮点下溢错误
DIVQERR     EQU 4                   除零错误
TOLGSMER    EQU 5                   数太大或太小不能转化为整数
NSQRTERR    EQU 6                   负数求平方根错误
TAN90ERR    EQU 7                   90°正切错误
* * * * * * ASCII转换为浮点* * * * * *
* * * * * * * * * ASCFLT* * * * * * * *
*           ORG $0000
* 局部变量
EXPSIGN     EQU 0                   指数符号,0为正,FF为负
PWR10EXP    EQU 1                   幂10指数
            ORG $C000
ASCFLT      EQU *
            PSHX                    存指针
            JSR PSHFPAC2            存FPACC2
            LDX #0                  将零压栈,开始单元
            PSHX
            STX FPACC1EX            FPACC1清零
            STX FPACC1EX+2
            CLR MANTSGN1            最初使尾数为正
            TSY
            LDX 6,Y                 取指向ASCII串的指针
ASCFLT1     LDAA 0,X                取第一个字符
            JSR NUMERIC
            BCS ASCFLT4
ASCFLT2     CMPA #'-                为负否?
            BNE ASCFLT3             否,检查小数点
            COM MANTSGN1            是,设置尾数符号
```

```
        INX                 指向下一个字符
        LDAA 0,X
        JSR NUMERIC
        BCS ASCFLT4
ASCFLT3 CMPA #'.            为小数点
        BNE ASCFLT5         否,格式错误
        INX                 是,指向下一个字符
        LDAA 0,X
        JSR NUMERIC         小数点后至少有一个 1
        BCC ASCFLT5         转表明错误
        JMP ASCFLT11        转建立小数
ASCFLT5 INS
        INS
        JSR PULFPAC2        恢复 FPACC2
        PULX
        LDAA #FLTFMTER      格式错误
        SEC                 置错误标志
        RTS                 返回
ASCFLT4 LDAA 0,X
        JSR NUMERIC
        BCC ASCFLT10
        JSR ADDNXTD
        INX
        BCC ASCFLT4
ASCFLT6 INC FPACC1EX        对 D. P. 前遇到的每位数字加 1
        LDAA 0,X
        INX
        JSR NUMERIC
        BCS ASCFLT 6
        CMPA #'.
        BNE ASCFLT17
ASCFLT8 LDAA 0,X            取下一个字符
        JSR NUMERIC         为一个数字?
        BCC ASCFLT7         否,转检查指数
        INX                 指向下一个字符
        BRA ASCFLT8
ASCFLT7 CMPA #'E            为指数?
        BEQ ASCFLT13        是,则处理
        JMP FINISH          否,转 FINISH
ASCFLT13 INX
        LDAA 0,X
        JSR NUMERIC         判断是否为数字
        BCS ASCFLT9         是,取指数
        CMPA #'-            为负号?
        BEQ ASCFLT15
        CMPA #'+            为正号
```

```
                BEQ ASCFLT16        是,则忽略之
                BRA ASCFLT5         格式错误
ASCFLT15        COM EXPSIGN,Y
ASCFLT16        INX
                LDAA 0,X            取下一个字符
                JSR NUMERIC         为数?
                BCC ASCFLT5         否,格式错
ASCFLT9         SUBA #$30
                STAA PWR10EXP,Y     幂 10 指数
                INX
                LDAA 0,X
                JSR NUMERIC
                BCC ASCFLT14
                LDAB PWR10EXP,Y
                LSLB                乘 2
                LSLB
                ADDB PWR10EXP,Y
                LSLB
                SUBA #$30
                ABA
                STAA PWR10EXP,Y
                CMPA #38
                BHI ASCFLT5
ASCFLT14        LDAA PWR10EXP,Y
                TST EXPSIGN,Y       为负?
                BPL ASCFLT12        否,将其相加建立 10 幂次方指数
                NEGA
ASCFLT12        ADDA FPACC1EX
                STAA FPACC1EX
                BRA FINISH
ASCFLT10        CMP #'.
                BNE ASCFLT7
                INX
ASCFLT11        LDAA 0,X
                JSR NUMERIC
                BCC ASCFLT7
                BSR ADDNXTD
                INX
                BCS ASCFLT8
                DEC FPACC1EX
                BRA ASCFLT11
ADDNXTD         LDAA FPACC1MN       取高 8 位
                STAA FPACC2MN       存入 FPACC2
                LDD FPACC1MN+1      取尾数的低 16 位
                STD FPACC2MN+1      存入 FPACC2
                LSLD                乘以 2
```

```
                ROL FPACC1MN            溢出?
                BCS ADDNXTD1            是,不加该数字
                LSLD                    乘以 4
                ROL FPACC1MN            溢出?
                BCS ADDNXTD1            是,不加该数字
                ADDD FPACC2MN-1
                PSHA                    存 A
                LDAA FPACC1MN           取高 8 位
                ADCA #0                 加低 16 位可能的进位
                ADDA FPACC2MN           加到高 8 位上
                STAA FPACC1MN
                PULA
                BCS ADDNXTD1            溢出? 是则不加
                LSLD
                ROL FPACC1MN
                STD FPACC1MN+1          存低 16 位
                BCS ADDNXTD1            溢出? 是则不加
                LDAB 0,X
                SUBB #$30
                CLRA
                ADDD FPACC1MN-1
                STD FPACC1MN            存结果
                LDAA FPACC1MN           取高 8 位
                ADCA #0                 加可能的进位。溢出?
                BCS ADDNXTD1            是,原尾数拷贝到 FPACC2
                STAA FPACC1MN           否,OK!
                RTS
ADDNXTD1        LDD FPACC2MN+1          因溢出,恢复原尾数
                STD FPACC1MN+1
                LDAA FPACC2MN
                STAA FPACC1MN
                RTS
*
FINISH          EQU *
                STX 6,Y
                LDX #FPACC1EX
                JSR CHCK0
                BEQ FINISH3
                LDAA FPACC1EX
                STAA PWR10EXP,Y
                LDAA #$80+24
                STAA FPACC1EX
                JSR FPNOM               尾数归一化
                TST PWR10EXP,Y          10 的幂次为正或零?
                BEQ FINISH3             为零,结束
                BPL FINISH1             为正,乘 10
```

```
        LDX #CONSTP1        否,取常数 0.1
        JSR GETFPAC2
        NEG PWR10EXP,Y
        BRA FINISH2
FINISH1 LDX #CONST10        取常数 10
        JSR GETFPAC2
FINISH2 JSR FLTMUL          FPACC1 乘 FPACC2,结果位于 FPACC1
        DEC PWR10EXP,Y
        BNE FINISH2
FINISH3 INS
        INS
        JSR PULFPAC2        恢复 FPACC2
        PULX
        RTS
NUMERIC EQU *
        CMPA #'0            是否小于 ASCII 0?
        BLO NUMERIC1        是,非数字
        CMPA #'9            大于 ASCII 的 9?
        BHI NUMERIC1        是
        SEC                 是数字,置进位位
        RTS
NUMERIC1 CLC
        RTS
FPNORM  EQU *
        LDX #FPACC1EX       指向 FPACC1
        BSR CHCK0           是否为 0
        BEQ FPNORM3         是,返回
        TST FPACC1MN        是否已归一化
        BMI FPNORM3         是,返回
FPNORM1 LDD FPACC1MN+1      取尾数的低 16 位
FPNORM2 DEC FPACC1EX        每次移位指数减 1
        BEQ FPNORM4         指数为 0,下溢
        LSLD                低 16 位移位
        ROL FPACC1MN        高 8 位旋转,归一化?
        BPL FPNORM2         否,继续左移
        STD FPACC1MN+1      低 16 位放回 FPACC1
FPNORM3 CLC                 显示无错误
        RTS
FPNORM4 SEC
        RTS
CHCK0   EQU *
        PSHB
        PSHA
        LDD 0,X             取 FPACC 指数和高 8 位
        BNE CHCK01          不为零,返回
        LDD 2,X             检查低 16 位
```

```
CHCK01      PULA
            PULB
            RTS                     返回时 CC 置位
CONSTP1     FCB                     $7D,$4C,$CC,$CD     0.1 小数
CONST10     FCB                     $84,$20,$00,$00     10.0 小数
* * * * * * * 浮点相乘 * * * * * *
* * * * * * * FLTMUL * * * * * * * *(最坏情况为 2319 周期)
FTLMUL      EQU *
            JSR PSHFPAC2
            LDX #FPACC1EX
            JSR CHCK0               检查 FPACC1 是否为零
            BEQ FPMULT3
            LDX #FPACC2EX
            JSR CHCK0
            BNE FPMULT4
            CLRA
            CLRB
            STD FPACC1EX
            STD FPACC1MN+1
            BRA FPMULT3
FPMULT4     LDAA MANTSGN1
            EORA MANTSGN2           设置结果的符号
            STAA MANTSGN1           存入
            LDAA FPACC1EX
            ADDA FPACC2EX
            BPL FPMULT1             若结果为负
            BCC FPMULT2
FPMULT5     LDAA #0VFERR            溢出错误
            SEC                     置错误标志
            BRA FPMULT6
FPMULT1     BCS FPMULT2             若结果为正并且进位位为 1,则 OK
            LDAA #UNFERR            否则发生下溢
            SEC
            BRA FPMULT6
FPMULT2     ADDA #$80               加 128 偏移量
            STAA FPACC1EX
            JSR UMULT               转乘"整数"尾数
FPMULT3     TST FPACC1EX
            BEQ FPMULT5
            CLC
FPMULT6     JSR PULFPAC2            恢复 FPACC2
            RTS
UMULT       EQU *
            LDX #0
            PSHX
            PSHX
```

```
        TSX
        LDAA #24
        STAA 0,X
UMULT1  LDAA FPACC2MN+2     取被乘数的最低位字节
        LSRA                将最低位移入进位位
        BCC UMULT2
        LDD FPACC1MN+1
        ADDD 2,X
        STD 2,X
        LDAA FPACC1MN       取乘数的高 8 位
        ADCA 1,X
        STAA 1,X            存部分积
UMULT2  ROR 1,X             部分积右移
        ROR 2,X
        ROR 3,X
        ROR FPACC2MN        被乘数右移一位
        ROR FPACC2MN+1
        ROM FPACC2MN+2
        DEC 0,X
        BNE UMULT1
        TST 1,X             部分积需要归一化?
        BMI UMULT3          否
        LSL FPACC2MN
        ROL 3,X
        ROL 2,X
        ROL 1,X
        DEC FPACC1EX
UMULT3  TST FPACC2MN        部分积需要舍入?
        BPL UMULT4          否,返回
        LDD 2,X
        ADDD #1
        STD 2,X             存结果
        LDAA 1,X
        ADCA #0
        STAA 1,X
        BCC UMULT4          若进位位为零,则 OK
        ROR 1,X
        ROR 2,X
        ROR 3,X
        INC FPACC1EX
UMULT4  INS
        PULX                取部分积的高 16 位
        STX FPACC1MN        放入 FPACC1
        PULA                取部分积的低 8 位
        STAA FPACC1MN+2     放入 FPACC1
        RTS
```

```
* * * * * *浮点相加* * * * * *
* * * * * * FLTADD * * * * * * (最坏情况为 1030 周期)
FLTADD      EQU *
            JSR PSHFPAC2          存 FPACC2
            LDX #FPACC2EX         指向 FPACC2
            JSR CHCK0
            BNE FLTADD1
FLTADD6     CLC
FLTADD10    JSR PULFPAC2          恢复 FPACC2
            RTS
FLTADD1     LDX #FPACC1EX
            JSR CHCK0
            BNE FLTADD2
FLTADD4     LDD FPACC2EX
            STD FPACC1EX
            LDD FPACC2MN+1        移尾数的低 16 位
            STD FPACC1MN+1
            LDAA MANTSGN2         将 FPACC2 尾数的符号移入 FPACC1
            STAA MANTSGN1
            BRA FLTADD6
FLTADD2     LDAA FPACC1EX         取 FPACC1 的指数
            CMPA FPACC2EX         指数是否相同?
            BEQ FLTADD7           是,尾数相加
            SUBA FPACC2EX         否,FPACC1EX－FPACC2EX. FPACC1＞
                                  FPACC2?
            BPL FLTADD3           是,检查范围
            NEGA                  否,FPACC1＜FPACC2,使差为正
            CMPA #23              是否位于范围之内?
            BHI FLTADD4           否,FPACC2 较大,将其移入 FPACC1
            TAB                   差放入 B
            ADDB FPACC1EX
            STAB FPACC1EX
            LDX #FPACC1MN
            BRA FLTADD5
FLTADD3     CMPA #23
            BHI FLTADD6
            LDX #FPACC2MN
FLTADD5     LSR 0,X               尾数的第一个字节移位
            ROR 1,X
            ROR2,X
            DECA
            BNE FLTADD5
FLTADD7     LDAA MANTSGN1         取 FPACC1 尾数符号
            CMPA MANTSGN2         符号是否相同?
            BEQ FLTADD11          是,两尾数相加
            TST MANTSGN1          否,FPACC1 为负?
```

```
        BPL FLTADD8          否,FPACC1-FPACC2
        LDX FPACC2MN         是,减之前 FPACC1 和 FPACC2 交换
        PSHS
        LDX FPACC1MN
        STX FPACC2MN
        PULX
        STX FPACC1MN
        LDX FPACC2MN-2       取 FPACC2 的低 8 位和符号
        PSHX
        LDX FPACC1MN+2
        STX FPACC1MN+2
        PULX
        STX FPACC1MN+2
FLTADD8 LDD FPACC1MN+1       取 FPACC1 的低 16 位
        SUBD FPACC2MN+1      减 FPACC2 的低 16 位
        STD FPACC1MN+1       存结果
        LDAA FPACC1MN        取 FPACC1 尾数的高 8 位
        SBCA FPACC2MN        减 FPACC2 的高 8 位
        STAA FPACC2MN        存结果。结果为负?
        BCC FLTADD9          否,归一化
        LDAA FPACC1MN        是,尾数取负
        COMA
        PSHA
        LDD FPACC1MN+1       取低 16 位
        COMB                 取反
        COMA
        ADDD #1              取补
        STD FPACC1MN+1       存结果
        PULA
        ADCA #0              加可能的进位
        STAA FPACC1MN
        LDAA #$FF
        STAA MANTSGN1
FLTADD9 JSR FPNORM           结果归一化
        BCC FLTADD12         返回
        LDAA #UNFERR         归一化时发生下溢
        SEC                  置错误标志
        JMP FLTADD10         返回
FLTADD12 JMP FLTADD6
FLTADD11 LDD FPACC1MN+1      取 FPACC1 的低 16 位
        ADDD FPACC2MN+1      将其加到 FPACC2 的低 16 位上
        STD FPACC1MN+1       结果存入 FPACC1
        LDAA FPACC1MN        取 FPACC1 的高 8 位
        ADCA FPACC2MN        带进位加到 FPACC2 的高 8 位上
        STAA FPACC1MN        存结果
        BCC FLTADD12         无溢出,返回
```

```
                ROR FPACC1MN
                ROR FPACC1MN+1
                ROR FPACC1MN+2
                INC FPACC1EX
                BNE FLTADD12
                LDAA #OVFERR            结果太大,溢出
                SEC                     置标志位
                JMP FLATDD10            返回
* * * * * *浮点相减* * * * * *
* * * * * * *FLTSUB* * * * * *(最坏情况为 1062 周期)
FLTSUB          EQU *
                BSR FLTSUB1             符号取反
                JSR FLTADD              转浮点数相加程序
FLTSUB1         LDAA MANTSGN2           取 FPACC2 尾数符号
                EORA # $FF              符号取反
                STAA MANTSGN2
                RTS
* * * * * *浮点相除* * * * * *
* * * * * * *FLTDIV* * * * * *(最坏情况为 2911 周期)
FLTDIV          EQU *
                LDX #FPACC2EX
                JSR CHCK0               被除数为零?
                BNE FLTDIV1             否,除数为零?
                LDAA #DIVOEER           是
                SEC
                RTS
FLTDIV1         LDX #FPACC1EX
                JSR CHCK0
                BNE FLTDIV2
                CLC
                RTS
FLTDIV2         JSR PSHFPAC2
                LDAA MANTSGN2           取 FPACC2 尾数的符号
                EORA MANTSGN1           设置结果的符号
                STAA MANTSGN1
                LDX #0
                PSHX
                PSHX
                PSHX
                LDAA #24                堆栈中的循环计数器
                PSHA
                TSX
                LDD FPACC1MN            比较 FPACC1 和 FPACC2 的尾数
                CPD FPACC2MN            高 16 位是否相同?
                BNE FLTDIV3             否
                LDAA FPACC1MN+2         是,比较低 8 位
```

```
                CMPA FPACC2MN+2
FLTDIV3         BHS FLTDIV4             FPACC2 的尾数大于 FPACC1 的尾数？否
                INC FPACC2EX            指数加 1，保持相同。溢出？
                BNE FLTDIV14            否，尾数右移一位
FLTDIV8         LDAA #OVFERR            是
                SEC
FLTDIV6         PULX
                PULX
                PULX
                INS
                JSR PULFPAC2
                RTS
FLTDIV4         LDD FPACC1MN+1          若被除数尾数大于除数，则相减
                SUBD FPACC2MN+1
                STD FPACC1MN+1
                LDAA FPACC1MN
                SBCA FPACC2MN
                STAA FPACC1MN
                DEC 0,X                 循环计数器减 1
FLTDIV14        LSR FPACC2MN            除数右移一位
                ROR FPACC2MN+1
                ROR FPACC2MN+2
                LDAA FPACC1EX           取 FPACC1 指数
                LDAB FPACC2EX           取 FPACC2 指数
                NEGB
                ABA
                BMI FLTDIV5
                BCS FLTDIV7
                LDAA #UNFERR
                BRA FLTDIV6
FLTDIV5         BCS FLTDIV8
FLTDIV7         ADDA # $81              加偏移量再加 1
                STAA FPACC1EX
FLTDIV9         LDD FPACC1MN
                STD 4,X
                LDAA FPACC1MN+2
                STAA 6,X
                LDD FPACC1MN+1          取低 16 位相减
                SUBD FPACC2MN+1
                STD FPACC1MN+1          存结果
                LDAA FPACC1MN           取高 8 位
                SBCA FPACC2MN
                STAA FPACC1MN
                BPL FLTDIV10            减法结束。转移位
                LDD 4,X                 恢复原除数
                STD FPACC1MN
```

```
                LDAA 6,X
                STAA FPACC1MN+2
FLTDIV10        ROL 3,X                 进位位移入商中
                ROL 2,X
                ROL 1,X
                LSL FPACC1MN+2          被除数左移
                ROL FPACC1MN+1
                ROL FPACC1MN
                DEC 0,X
                BNE FLTDIV9
                COM 1,X
                COM 2,X
                COM 3,X
                LDD FPACC1MN+1
                SUBD FPACC2MN+1
                LDAA FPACC1MN
                SBCA FPACC2MN
                LDD 2,X                 取低 16 位
                BCC FLTDIV11
                CLC
                BRA FLTDIV13
FLTDIV11        ADDD #1
FLTDIV13        STD FPACC1MN+1          放入 FPACC1
                LDAA 1,X                取高 8 位
                ADCA #0
                STAA FPACC1MN
                BCC FLTDIV12
                ROR FPACC1MN
                ROR FPACC1MN+1
                ROR FPACC1MN+2
FLTDIV12        CLC
                JMP FLTDIV6             无错,返回
* * * * * * * 浮点转换为 ASCII * * * * * *
* * * * * * * FLTASC * * * * * * * *
FTLASC          EQU *
                PSHX
                LDX FPACC1EX
                JSR CHCK0
                BNE FLTASC1
                PULX
                LDD # $ 30              取 ASCII 字符加上终止字节
                STD 0,X
                RTS
FLTASC1         LDX FPACC1EX
                PSHX
                LDX FPACC1MN+1
```

```
            PSHX
            LDAA MANTSGN1
            PSHA
            JSR PSHFPAC2        存FPACC2
            LDX #0
            PSHX
            PSHX
            PSHX
            TSY
            LDX 15,Y
            LDAA #$20
            TST MANTSGN1        为负?
            BEQ FLTASC2         否
            CLR MANTSGN1
            LDAA #'-            是,将负号放入缓冲器
FLTASC2     STAA 0,X
            INX                 指向下一个单元
            STX 0,Y
FLTASC5     LDX #N9999999       指向常数9999999
            JSR GETFPAC2
            JSR FLTCMP          比较,FPACC1>9999999?
            BHI FLTASC3         是,将FPACC1除以10
            LDX #P9999999       指向常数999999.9
            JSR GETFPAC2        移入FPACC2
            JSR FLTCMP          比较,FPACC1>999999.9?
            BHI FLTASC4         是,继续转换
            DEC 2,Y
            LDX #CONST10        否,乘以10
FLTASC6     JSR GETFPAC2
            JSR FLTMUL
            BRA FLTASC5         再次比较
FLTASC3     INC 2,Y             计数器加1
            LDX #CONSTP1        指向常数"0.1"
            BRA FLTASC6         转FPACC1除10
FLTASC4     LDX #CONSTP5        常数"0.5"
            JSR GETFPAC2        移入FPACC2
            JSR FLTADD          将0.5加到FPACC1,进行舍入
            LDAB FPACC1EX       取FPACC1指数
            SUBB #$81
            NEGB
            ADDB #23
            BRA FLTASC17
FLTASC7     LSR FPACC1MN        右移,取整
            ROR FPACC1MN+1
            ROR FPACC1MN+2
            BECB
```

```
FLTASC17    BNE FLTASC7
            LDAA #1
            STAA 3,Y
            LDAA 2,Y            原数大于 9999999?
            ADDA #8             是,必须表示为科学计数形式
            BMI FLTASC8         原数小于 1?
            CMPA #8             是,必须表示为科学计数形式
            BHS FLTASC8         否,可用 7 位数字表示
            DECA
            STAA 3,Y
            LDAA #2             置指数为 0
FLTASC8     SUBA #2             从指数中减 2
            STAA 2,Y            存指数
            TST 3,Y             是否有整数部分?
            BGT FLTASC9
            LDAA #'.            是小数点
            LDX 0,Y             指针指向缓冲器
            STAA 0,X            小数点存入缓冲器
            INX                 指向下一个缓冲器地址
            TST 3,Y             数字计数到零?
            BEQ FLTASC18        否,或小于 0.1
            LDAA #'0            是,数字格式化为.0××××××
            STAA 0,X            将零放入缓冲器
            INX
FLTASC18    STX 0,Y
FLTASC9     LDX #DECDIG
            LDAA #7
            STAA 5,Y
FLTASC10    CLR 4,Y             清小数点累加器
FLTASC11    LDD FPACC1MN+1      取尾数的低 16 位
            SUBD 1,X            减去常数的低 16 位
            STD FPACC1MN+1      存结果
            LDAA FPACC1MN       取高 8 位
            SBCA 0,X            减高 8 位
            STAA FPACC1MN       存结果。下溢?
            BCS FLTASC12        是
            INC 4,Y
            BRA FLTASC11
FLTASC12    LDD FPACC1MN+1      取 FPACC1MN 尾数的低 16 位
            ADDD 1,X
            STD FPACC1MN+1
            LDAA FPACC1MN       取高 8 位
            ADCA 0,X
            STAA FPACC1MN
            LDAA 4,Y
            ADDA # $30          转换为 ASCII
```

```
                PSHX
                LDX 0,Y
                STAA 0,X        将数字放入缓冲器
                INX             指向下一个缓冲器单元
                DEC 3,Y
                BNE FLTASC16
                LDAA #'.        取小数点
                STAA 0,X
                INX
FLTASC16        STX 0,Y
                PULX
                INX
                INX
                INX
                DEC 5,Y
                BNE FLTASC10
                LDX 0,Y
FLTASC13        DEX
                LDAA 0,X
                CMPA # $30      为 ASCII 的零?
                BEQ FLTASC13
                INX
                LDAB 2,Y
                BEQ FLTASC15
                LDAA #'E
                STAA 0,X
                INX
                LDAA #'+
                STAA 0,X
                TSTB
                BPL FLTASC14
                NEGB
                LDAA #'-
                STAA 0,X
FLTASC14        INX
                STX 0,Y
                CLRA
                LDX #10         十进制指数除以 10
                IDIV
                PSHB            存余数
                XGDX            商放入 D
                ADDB # $30      转换为 ASCII
                LDX 0,Y         取指针
                STAB 0,X
                INX             指向下一个单元
                PULB            取第二个数字
```

```
                ADDB #$30
                STAB 0,X
                INX
FLTASC15        CLR 0,X
                PULX
                PULX
                PULX
                JSR PULFPAC2            恢复 FPACC2
                PULA
                STAA MANTSGN1
                PULX                    恢复 FPACC1
                STX FPACC1MN+1
                PULX
                STX FPACC1EX
                PULX                    指向 ASCII 串的开始处
                RTS
DECDTG          EQU *
                FCB $0F,$42,$40         十进制数 1,000,000
                FCB $01,$86,$A0         十进制数 100,000
                FCB $00,$27,$10         十进制数 10,000
                FCB $00,$03,$E8         十进制数 1,000
                FCB $00,$00,$64         十进制数 100
                FCB $00,$00,$0A         十进制数 10
                FCB $00,$00,$01         十进制数 1
P9999999        EQU *                   常数 9999999
                FCB $94,$74,$23,$FE
N9999999        EQU *                   常数 9999999
                FCB $98,$18,$96,$7F
CONSTP5         EQU *                   常数 0.5
                FCB $80,$00,$00,$00
* * * * * * 浮点数相比较程序 FLTCMP * * * * * *
FLTCMP          EQU *
                FST MANTSGN1            FPACC1 为负?
                BPL FLTCMP2             否,继续比较
                TST MANTSGN2            FPACC2 为负?
                BPL FLTCMP2             否,继续比较
                LDD FPACC2EX            是,均为负
                CPD FPACC1EX
                BNE FLTCMP1
                LDD FPACC2MN+1
                CPD FPACC1MN+1
FLTCMP1         RTS
FLTCMP2         LDAA MANTSGN            取 FPACC1 尾数符号
                CMPA MANTSGN2           二者均为正
                BNE FLTCMP1
                LDD FPACC2EX            取 FPACC1 的指数和尾数高 8 位
```

```
            CPD FPACC2EX
            BNE FLTCMP1
            LDD FPACC1MN+1          取 FPACC1 尾数的低 16 位
            CPD FPACC2MN+1          与 FPACC2 尾数的低 16 位相比较
            RTS
* * * * * * 无符号整数转化为浮点数* * * * * *
* * * * * *UINT2FLT* * * * * *
UINT2FLT    EQU *
            LDX #FPACC1EX
            JSR CHCK0
            BNE UINTFLT1
            RTS
UINT2FLT1   LDAA #$98
            STAA FPACC1EX
            JSR FPNORM              转化为归一化浮点值
            CLC
            RTS
* * * * * * 有符号整数转化为浮点数* * * * * *
* * * * * *SINT2FLT* * * * * *
SINT2FLT    EQU *
            LDD FPACC1MN+1          取 FPACC1 尾数的低 16 位
            PSHA
            BPL SINTFLT1            若为正进行转换
            COMA
            COMB
            ADDD #1                 取补
            STD FPACC1MN+1
SINTFLT1    BSR UINT2FLT            调转换程序
            PULA                    取原整数的符号
            LDAB #$FF               取"负号"
            TSTA                    为负?
            BPL SINTFLT2            否,返回
            STAB MANTSGN1           是,置 FPACC1 符号字节
SINTFLT2    CLC
            RTS
* * * * * * 浮点数转换为整数(有符号或无符号)* * * * * *
* * * * * *FLT2INT* * * * * *
FLT2INT     EQU *
            LDX #FPACC1EX
            JSR CHCK0
            BEQ FLT2INT3
            LDAB FPACC1EX           取 FPACC1 指数
            CMPB#$81                有整数部分
            BLO FLT2INT2
            TST MANTSGN1
            BMI FLT2INT1
```

```
                CMPB # $ 90             该数是否太大?
                BHI FLT2INT4            是,出错并返回
                SUBB # $ 98
FLT2INT5        LSR FPACC1MN
                ROR FPACC1MN+1
                ROR FPACC1MN+2
                INCB
                BNE FLT2INT5
                CLR FPACC1EX
                RTS
FLT2INT1        CMPB # $ 8F             该数是否太小?
                BHI FLT2INT4            是,出错返回
                SUBB # $ 98
                BSR FLT2INT5
                LDD FPACC1MN+1
                COMA
                COMB
                ADDD #1                 求补
                STD FPACC1MN+1          存结果
                CLR MANTSGN1            尾数符号清零
                RTS
FLT2INT4        LDAA TOLGSMER           数太大或太小,不能转化为整数
                SEC
                RTS
FLT2INT2        LDD #0
                STD FPACC1EX
                STD FPACC1MN+1
FLT2INT3        RTS
* * * * * * 平方根函数 * * * * * *
* * * * * * FLTSQR * * * * * * (最坏情况为 16354 周期)
FLTSQR          EQU *
                LDX #FPACC1EX
                JSR CHCK0
                BNE FLTSQR1
                RTS
FLTSQR1         TST MANTSGN1            为负?
                BPL FLTSQR2             否,求平方根
                LDAA #NSQRTERR          是,出错
                SEC
                RTS
FLTSQR2         JSR PSHFPAC2
                LDAA #4
                PSHA
                LDX FPACC1MN+1
                PSHX
                LDX FPACC1EX
```

```
            PSHX
            TSY
            BSR TFR1TO2
            LDAA FPACC2EX          取 FPACC1 的指数
            SUBA # $80             去除偏移量
            INCA
            BPL FLTSQR3            若大于 1,除以 2 并加偏移量
            LSRA                   若小于 1,除以 2
            BRA FLTSQR4            转计算平方根
FLTSQR3     LSRA
            ADDA # $80
FLTSQR4     STAA FPACC2EX          存指数/2
FLTSQR5     JSR FLTDIV             凭猜测除原始数
            JSR FLTADD             "猜测"加到商上
            DEC FPACC1EX           结果除以 2,产生新的猜测值
            BSR TFR1TO2            存入 FPACC2
            LDD 0,Y                取原始数
            STD FPACC1EX
            LDD 2,Y                取尾数的低 16 位
            STD FPACC1MN+1
            DEC 4,Y
            BNE FLTSQR5
            LDD FPACC2EX
            STD FPACC1EX
            LDD FPACC2MN+1
            STD FPACC1MN+1
            PULX
            PULX
            INS
            JSR PULFPAC2           恢复 FPACC2
            CLC
            RTS
* * * * * * FPACC1 传输至 FPACC2 程序 * * * * * *
* * * * * * TFR1TO2 * * * * * *
TFR1TO2     EQU *
            LDDD FPACC1EX          取 FPACC1 的指数和尾数高 8 位
            STD FPACC2EX           放入 FPACC2
            LDD FPACC1MN+1         取 FPACC1 尾数的低 16 位
            STD FPACC2MN+1         放入 FPACC2
            LDAA MANTSGN1          传输符号
            STAA MANTSGN2
            RTS
* * * * * * 浮点正弦函数 * * * * * *
* * * * * * FLTSIN * * * * * *
FLTSIN      EQU *
            JSR PSHFPAC2
```

```
                JSR ANGRED          使角度减到±π
                PSHB                存象限数
                PSHA                存正/余弦标志
                JSR DEG2RAD         度数转换为弧度
                PULA                恢复正/余弦标志
FLTSIN1         JSR SINCOS
                PULA
                CMPA #2             角度在1或2象限?
                BLS FLTSIN2         是,结束
                COM MANTSGN1        否,正弦在3.4象限为负
FLTSIN2         CLC
                JSR PULFPAC2        恢复FPACC2
                RTS
* * * * * * 浮点余弦函数 * * * * * *
* * * * * * FLTCOS * * * * * *
FLTSOS          EQU *
                JSR PSHFPAC2
                JSR ANGRED
                PSHB
                PSHA
                JSR DEG2RAD         转换为弧度
                PULA
                EORA #$01
                JSR SINCOS
                PULA
                CMPA #1             以下确定角度所在象限,以确定符号
                BEQ FLTCOS1
                CMPA #4
                BEQ FLTCOS1
                COM MANTSGN1
FLTCOS1         JMP FLTSIN2
* * * * * * 正/余弦子程序 * * * * * *
* * * * * * SINCOS * * * * * *
SINSOS          EQU *
                PSHA                正、余弦标志存入堆栈
                LDX FPACC1MN+1      存角度值
                PSHX
                LDX FPACC1FX
                PSHX
                LDAA MANTSGN1
                PSHA
                LDX #SINFACT        指向小数表格
                PSHX
                PSHX
                LDAA #$4            初始值计数
                PSHA
```

```
            TSY
            JSR TFR1TO2          FPACC1 传输到 FPACC2
            JSR FLTMUL           FPACC1 平方
            TST 10,Y
            BEQ SINCOS7
            LDX #COSFACT
            STX 1,Y
            JSR TFR1TO2
            BRA SINCOS4
SINCOS7     JSR EXG1AND2         X² 置于 FPACC2,X 置于 FPACC1
SINCOS1     JSR FLTMUL           产生 X³、X⁵、X⁷ 或 X³、X⁴、X⁶ 等
SINCOS4     LDX FPACC1MN+1
            PSHX
            LDX FPACC1EX
            PSHX
            LDAA MANTSGN1
            PSHA                 存尾数符号
            DEC 0,Y              已产生所有次方?
            BNE SINCOS1          否,继续
            LDAA #$4             置循环数
            STAA 0,Y
            TSX
SINCOS2     STX 3,Y
            LDX 1,Y
            JSR GETFPAC2
            INX
            INX
            INX
            INX
            STX 1,Y              存指针
            LDX 3,Y
            LDAA 0,Y             取数的符号
            STAA MANTSGN1
            LDD 1,X              取尾数的低 16 位
            STD FPACC1EX
            LDD 3,X              取尾数的高 8 位和指数
            STD FPACC1MN+1
            JSR FLTMUL           乘以 2
            LDX 3,Y
            LDD FPACC1MN+1
            STD 3,X
            LDD FPACC1EX
            STD 1,X
            LDAA MANTSGN1
            STAA 0,X
            INX                  指向下一个幂次方
```

```
            INX
            INX
            INX
            INX
            DEC 0,Y
            BNE SINCOS2
            LDAA # $ 3          取循环数
            STAA 0,Y
            LDX 3,Y             指向堆栈的结果
            DEX                 指向以前的结果
            DEX
            DEX
            DEX
            DEX
            STX 3,Y             存新指针
            LDAA 0,X
            STAA MANTSGN2
            LDD 1,X             取尾数低 16 位
            STD FPACC2EX
            LDD 3,X
            STD FPACC2MN+1
            JSR FLTADD          调相加程序
            DEC 0,Y
            BNE SINCOS3
            TST 10,Y
            BEQ SINCOS5
            LDX #ONE
            JSR GETFPAC2
            BRA SINCOS6
SINCOS5     LDAA 5,Y            取初始角度值
            STAA MANTSGN2
            LDD 6,Y
            STD FPACC2EX
            LDD 8,Y
            STD FPACC2MN+1
SINCOS6     JSR FLTADD
            TSX                 清堆栈
            XGDX
            ADDD # $ 31
            XGDX
            TSX                 更新堆栈指针
            RTS
ANGRED      EQU *
            CLRA
            PSHA
            INCA
```

```
            PSHA
            TSY
            LDX #THREE60        指向常数 360
            JSR GETFPAC2
            TST MANTSGN1        输入角度为负?
            BPL ANGRED1         否,跳过相加过程
            JSR FLTADD
ANGRED1     DEC FPACC2EX        使 FPACC2 的常数为 90°
            DEC FPACC2EX
ANGRED2     JSR FLTCMP          角度已小于 90°?
            BLS ANGRED3         是
            JSR FLTSUB          否,角度减少 90°
            INC 0,Y             象限数加 1
            BRA ANGRED2         再检查是否小于 90°
ANGRED3     LDAA 0,Y
            CMPA #1             原角度位于象限 1?
            BEQ ANGRED4         是,计算三角函数
            CMPA #3             否,原角度位于象限 3?
            BEQ ANGRED4         是,计算三角函数
            LDAA # $FF          否,计算与 90°互补值的三角函数
            STAA MANTSGN1
            JSR FLTADD          负角度值加上 90°
ANGRED4     DEC FPACC2EX        使 FPACC2 的角度值为 45°
            JSR FLTCMP          角度小于 45°?
            BLS ANGRED5         是,OK!
            INC FPACC2EX        否,取(90-角度值)
            LDAA # $FF          使 FPACC1 为负
            STAA MANTSGN1
            JSR FLTADD
            INC 1,Y
ANGRED5     PULB
            PULA
            RTS
            *
EXG1AND2    EQU *
            LDD FPACC1EX
            LDX FPACC2EX
            STD FPACC2EX
            STX FPACC1EX
            LDD FPACC1MN+1
            LDX FPACC2MN+1
            STD FPACC2MN+1
            STX FPACC1MN+1
            LDAA MANTSGN1
            LDAB MANTSGN2
            STAA MANTSGN2
```

```
                STAB MANTSGN1
                RTS
SINFACT         EQU *
                FCB $6E,$38,$EF,$1D          +(1/9!)
                FCB $74,$D0,$0D,$01          -(1/7!)
                FCB $7A,$08,$88,$89          +(1/5!)
                FCB $7E,$AA,$AA,$AB          -(1/3!)
COSFACT         EQU *
                FCB $71,$56,$0D,$01          +(1/8!)
                FCB $77,$B6,$0B,$61          -(1/6!)
                FCB $7C,$2A,$AA,$AB          +(1/4!)
                FCB $80,$80,$00,$00          -(1/2!)
ONE             FCB $81,$00,$00,$00          1.0
PI              FCB $82,$49,$0F,$DB          3.1415927
THREE60         FCB $89,$34,$00,$00          360.0
* * * * * *正切函数* * * * * *
* * * * * *FLTTAN* * * * * *
FLTTAN          EQU *
                JSR PSHFPAC2
                JSR TFR1TO2
                JSR FLTCOS                   计算余弦值
                JSR EXG1AND2                 结果放入 FPACC2,角度放入 FPACC1
                JSR FLTSIN                   求正弦值
                JSR FLTDIV                   求正弦/余弦
                BCC FLTTAN1                  若进位清零,则 OK
                LDX #MAXNUM                  角度接近 90°,取最大值
                JSR GETFPAC1
                LDAA #TAN90ERR
FLTTAN1         JSR PULFPAC2                 恢复 FPACC2
                RTS
MAXNUM          EQU *
                FCB $FE,$7E,$FF,$FF          可能的最大数
* * * * * *弧度与度数相互转换以及圆周率π值程序* * * * * *
* * * * * *DEG2RAD,RAD2DEG,GETPI* * * * * *
DEG2RAD         EQU *
                JSR PSHFPAC2
                LDX #PIOV180                 常数 3.1415927/180
DEG2RAD1        JSR GETFPAC2
                JSR FLTMUL                   度数转换为弧度
                JSR PULFPAC2
                RTS                          (注意,不能用 JMP 代替 JSR/RTS)
*
RAD2REG         EQU *
                JSR PSHFPAC2                 指向常数 180/3.1415927
                LDX #C180OVPI
                BRA DEG2RAD1                 转换并返回
```

```
*
GETPI       EQU *
            LDX #PI
            JMP GETFPAC1
PIOV180     EQU *
            FCB $7B,$0E,$FA,$35
C180OVPI    EQU *
            FCB $86,$65,$2F,$E1
* * * * * * FPACC2 压栈与弹出程序 * * * * * *
* * * * * * PSHPULFPAC2 * * * * * *
PSHFPAC2    EQU *
            PULX                    取返回地址
            PSHX                    分配4字节地址
            PSHX
            XGDX                    返回地址放入D
            TSX
            PSHB
            PSHA
            JMP PUTFPAC2
*
PULFPAC2    EQU *
            TSX                     指向返回地址
            INX
            INX
            JSR GETFPAC2            恢复FPACC2
            PULX
            INS
            INS
            INS
            INS
            JMP 0,X
* * * * * * 存储格式转换为浮点累加器格式的程序 * * * * * *
* * * * * * GETFPAC * * * * * *
GETFPAC     EQU *
            LDD 0,X                 取指数和尾数的高8位
            BEQ GETFP12             若为0,跳过设置最高位过程
            CLR MANTSGN1            设置为正数
            TSTB                    为负?
            BPL GETFP11             否
            COM MANTSGN1            是,置符号为负
GETFP11     ORAB #$80               恢复尾数的最高位
GETFP12     STD FPACC1EX
            LDD 2,X                 取低16位尾数
            STD FPACC1MN+1
            RTS
*
```

```
GETFPAC2    EQU *
            LDD 0,X
            BEQ GETFP22
            CLR MANTSGN2
            TSTB
            BPL GETFP21
            COM MANTSGN2
GETFP21     ORAB #$80
GETFP22     STD FPACC2EX
            LDD 2,X
            STD FPACC2MN-1
            RTS
* * * * * * 累加器格式转换为存储器格式的程序 * * * * * *
* * * * * * PUTFPAC * * * * * *
PUTFPAC1    EQU *
            LDD FPACC1EX        取 FPACC1 的指数和尾数的高 8 位
            TST MANTSGN1        为负?
            BMI PUTFP11         是,保持置位
            ANDB #$7F           否,最高位清零
PUTFP11     STD 0,X             存入
            LDD FPACC1MN-1      取尾数的低 16 位
            STD 2,X
            RTS
*
PUTFPAC2    EQU *
            LDD FPACC2EX
            TST MANTSGN2
            BMI PUTFP21
            ANDB #$7F
PUTFP21     STD 0,X
            LDD FPACC2MN-1
            STD 2,X
            RTS
```

4. 用 PC 机对 M68HC11 内部或外部 EEPROM 进行编程

本节介绍一种对 M68HC11 内部 EEPROM 或与该 MCU 外部总线相连的 EEPROM 的简单而可靠的方法。编程数据从 PC 机的串行通讯接口(SCI)装入，还可以检查 MCU 内部或外部存储器的内容与 PC 机盘上的数据是否一样。编程和校验所用数据都是 S 记录格式。

M68HC11 EEPROM 编程所需的最小系统如图 4-1 所示，它只包括 MCU、RS232 电平位移电路，8MHz 晶体和几个无源元件。

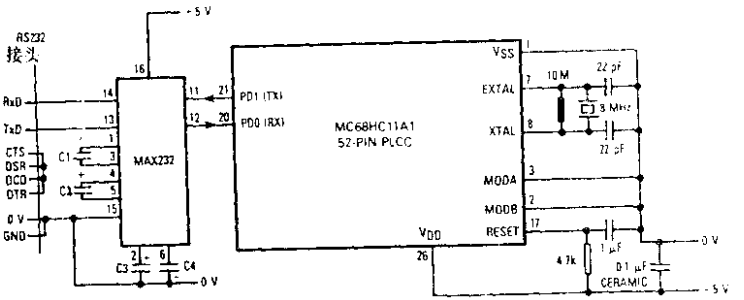


图 4-1 M68HC11 EEPROM 编程电路构成

注：C1、C2、C3 和 C4 为 22μF/25V。

为提高可靠性，所有不用的引脚应接 VSS 或 VDD.

PC 机通过串行口与 M68HC11 的串行口(SCI)相连，图中采用 Maxim MAX232，不需 ±12V 电源。MODA 和 MODB 引脚接低电平，复位后 MCU 进入特殊自引导方模。这时 MCU 开始执行自引导程序，该程序位于地址为 $BF40～$BFFF 的内部 ROM 中。在普通单片和扩展模式下，不能访问自引导 ROM。

对于自引导操作，所有向量重新定位于自引导(boot)ROM 区。除 RESET 向量外，它指向自引导 ROM 的开始处，其余中断向量均指向 RAM 中未初始化的转移表。转移表中每一项保留 3 个字节。自引导向量的存储器映象和 RAM 转移表分别如表 4-1 和表 4-2 所示。

表 4-1 自引导向量(自引导 ROM)

地址	向量	说　明
BFFE	BF40	自引导复位
BFFC	00FD	时钟监视
BFFA	00FA	COP 失效
BFF8	00F7	非法操作码
BFF6	00F4	SWI
BFF4	00F1	XIRQ
BFF2	00EE	IRQ
BFF0	00EB	实时中断
BFEE	00E8	定时器输入捕获 1

地址	向量	说　明
BFEC	00E5	定时器输入捕捉 2
BFEA	00E2	定时器输入捕捉 3
BFE8	00DF	定时器输出比较 1
BFE6	00DC	定时器输出比较 2
BFE4	00D9	定时器输出比较 3
BFE2	00D6	定时器输出比较 4
BFE0	00D3	定时器输出比较 5
BFDE	00D0	定时器溢出
BFDC	00CD	脉冲累加器溢出
BFDA	00CA	脉冲累加器输入边沿
BFD8	00C7	SPI
BFD6	00C4	SCI

表 4-2　RAM 转移表(内部 RAM)

地址	典型指令
00FD	JMP CLKMON
00FA	JMP COPFL
…	…

在自引导模式允许任何中断之前，应由软件初始化转移表中的进入点。本例不使用中断。

自引导程序设置 SCI 发送/接收器为 7812 波特，并检查 CONFIG 寄存器的 NOSEC 位。若该位为 0(允许保密)，则自引导程序擦除整个内部 EEPROM 以及 CONFIG 寄存器的内容。这在需要保密的应用场合特别有用。EEPROM 可存放一些特殊的程序和数据。若 NOSEC 为 1，则不执行擦除过程。检验 NOSEC 位后，自引导程序在 SCI 发送端发出中止(break)状态，并等待接收第一个字节。

这时需要启动 PC 机的 S 记录装入程序，称为 EELOAD. BAS(BASIC 语言编写)。PC 驻留程序现在设置 COM 通道为 1200 波特，一个停止位、无校验位，将二进制文件 EEPROGIX. BOO 从 PC 装入到 M68HC11。

M68HC11 自引导程序自动检测以不同波特率接收到的第一个字符，并将其 SCI 波特率改为 1200 波特。然后将二进制文件装入 256 字节 RAM 中，并转向地址 $0000(即第一个 RAM 单元)。

EEPROGIX. B00 由后面 M68HC11 源列表中的可执行码组成。该程序设计用于接收 PC 的 S 记录，并将数据编程到 EEPROM 地址。注意，EEPROGIX. B00 开始处的 $FF 只用于检测 PC 的波特率，并不返回，而其余的 256 字节由 M68HC11 的发送器送回。但在装入 EEPROGIX. B00 期间，PC 不检测返回数据，在该过程没有必要。

一旦 M68HC11 新装入的 S 记录编程程序开始执行，就设置 SCI 为 9600 波特，然后等待 PC 的控制字符，该字符决定其操作，如表 4-3 所示。注意，可用来装入和校验外部 RAM 以及外部 EEPROM。

表 4-3　S 记录装入程序操作模式选择

控制字符	操作模式
X	编程外部 EEPROM/RAM
I	编程内部 EEPROM
V	校验内部或外部 EEPROM/RAM

如果 S 记录编程程序已成功地装入，则 PC 驻留程序将一 -

(1)请求装入的数据是否必须返回屏幕；

(2)促使用户决定所需的操作模式；

(3)请求将要从 PC 装入的 S 记录文件名。

--旦开始装入，S 记录文件的每个字符都立即返回到 PC 机。PC 和 M68HC11 发送都是同时、同步的，这样可减少有关 EEPROM 编程延时以及所需的硬件应答。

4.1 校验

若校验发生错误，实际存储的字节值返回到 PC 显示，可很快地检查 EEPROM 数据和地址错误。

4.2 内部与外部选择

若在内部或外部编程模式发生编程错误，即编程后读回的数据与所需数据不同，则 M68HC11 驻留软件暂停。该状态由 PC 驻留程序检测，然后中止装入并 显示错误信息。若 PC 与 M68HC11 之间连接有错，也显示错误信息。

有一 种情况例外，改变 M68HC11 CONFIG 寄存器只能由以后的硬件 RESET 检测。若检测到 CONFIG 寄存器地址为 $ 103F，则不是在编程后直接读 CONFIG 寄存器，以避免中止装入过程。

为了允许能对 MC68HC11A 系列所有型号的 CONFIG 寄存器进行编程，并允许扩展模式操作，将 HPRIO(地址 $ 103C)寄存器的 MDA 位(位 5)置 1，MCU 驻留程序使用自引导模式转换为特殊测试模式。

若用户希望保持在自引导模式下操作(例如，检验内部 ROM 码)，则后面 EEPROMIX.ASC 中的第 8 行"BSET HPRIO,X,#MDA"应去掉。

4.3 内部 EEPROM 编程

内部与外部 EEPROM 的编程技术大不相同。对于内部 EEPROM，通常首先需要擦除所需的字节(擦除状态为 $ FF)，随后向同一地址写入数据。

内部编程包括访问 PPROG 寄存器($ 103B)来锁存 EEPROM 地址和数据总线，编程延时必须由软件实现。该例中采用软件循环，也可用 M68HC11 内部的一个定时器来提供延时。图 4-2 和图 4-3 分别是内部 EEPROM 擦除和写入过程。

4.4 外部 EEPROM 编程

图 4-4 是 M68HC11 与外部 2864 EEPROM 接口的硬件电路，MC68HC24 用于仿真 MC68HC11A8 单片方式。使 2864 的优点是可以修改软件程序和/或数据而不需从目标系统中去除仿真器。

为了从 PC 机对 2864 编程，在 EELOAD 菜单中必须选择外部操作模式(X)。

编程 2864 所需的操作比编程内部 EEPROM 简单，因为前者没有 PPROG 控制寄存器。另外，擦除过程和延迟时间自动由 2864 片内逻辑处理。

采用数据查询技术来确定编程延迟时间是否结束。即读出要写入的地址，检查被编程数据的最高位。(在编程延迟期间，最高位读出将是希望数据的补值)。

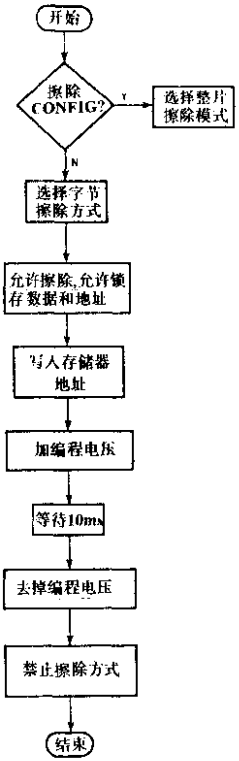


图 4-2 内部 EEPROM 擦除过程

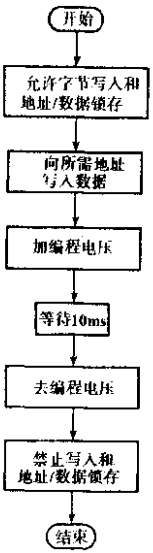


图 4-3 内部 EEPROM 写入过程

4.5 仿真器地址译码

图 4-4 中，MC68HC11 的地址线 A13 连到 2864 的 26 引脚。它在 2864 中是没用的，但这样可用 27128 16K 字节的 EEPROM 存储器替代 2864。

当使用 2864 时，存储器地址范围 $C000～$DFFF 被映象到通常所用的 8K 字节范围 $E000～$FFFF。当使用 27128 存储器时，其完整的 16K 字节范围 $C000～$FFFF 均可用。

S 记录编程器特点还包括：强迫程序在 S 记录地址字段由 S9 指定的地址开始执行(若地址不为 $0000)。S 记录格式如下：

S1 14C01E OB 29 BD CO 2A 18 38 6A 3B 6F 3B 39 18 09 26 FC 39 DE

① ② ③ ④ ⑤

①记录类型：S0、S1、S9

S0——S0 记录，记录的开始。其后有地址和检验和。

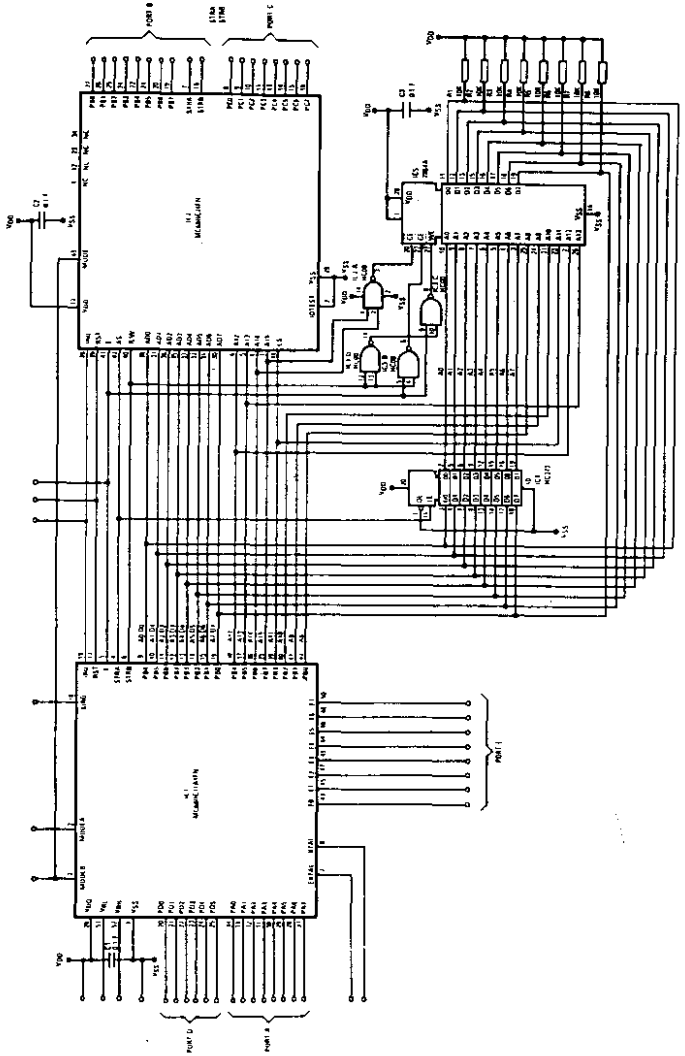


图 4-4　采用 2864 的 MC68HC11A8 仿真器

S1 — S1 记录，码/数据记录。码/数据字段含有可执行的码或数据以及检验和。两字节地址表示码/数据首地址。

S9——S9 记录，终止记录。其后有执行地址和检验和，无码/数据字段。

②该记录所包括的十六进制字符对(字节)数。$14 表示后面有 $14(十进制 20)字节数据，其中包括两字节地址、17 字节数据和一字节检验和。

③地址字段，该记录的地址为 $C01E。

④码/数据字段，该记录有 17 字节。

⑤检验和字节，是②～④所有字节之和的反码。该记录的检验和为 DE。

4.6 EELOAD.BAS 程序

```
40' This program downloads S record file to the MC68HC11 through special'
50' bootstrap program,designed to program either internal or external'
60' EEPROM in the 68HC11's memory map'
70' The loader can also verify memory against an S record file. '
80' Downloaded data is optionally echoed on terminal. '
90' ==========================
100 CR$ =CHR$ (13)
110 MIN$ =CHR$ (32)
120 MAX$ =CHR$ (127)
130 ERM$ ="Cant find"
140 LOADER$ ="EEPROGIX.BOO"
150 CLRLN$ =SPACE$ (80).
160 VER$ ="1.0":'Version number of EELOAD'
170 ERRTOT%=0:'Number of errors found by verify operation'
180 CLS
190 PRINT " <<<<<<<< EELOAD Version";VERS;" >>>>>>>"
200 PRINT " <<<<<<<<< 68HC11 Internal/External EEPROM loader/verifier >>>>>"
210 PRINT
220 PRINT "==>Before continuing,ensure 68HC11 is in bootsrtap mode,"
230 PRINT "     RESET is off,and COM1 or COM2 is connected to the SCI"
240 PRINT
250 ' First make sure loader program is available'
260 ON ERROR GOTO 880
270 OPEN LOADER$ FOR INPUT AS #2
280 CLOSE #2
290 ON ERROR GOTO 0
300 CHAN$ ="0"
310 ROW=CSRLIN:'Store current line number'
320 WHILE CHAN$ <>"1" AND CHAN$ <>"2"
330 GOSUB 1070
340 LINE INPUT "Enter com channel number (1/2):",CHAN$
350 WEND
360 CMS="COM"+CHAN$
```

```
370 ',Now set baud rate to 1200 and load EEPROG through boot loader'
380 ' by executing DOS MODE and COPY commands'
390 SHELL "MODE"+CMS+",1200,N,8,1"
400 SHELL "COPY"-LOADER$+" "+CMS
401 GOSUB 1070
402 FOR I%=1 TO 4:PRINT CLRLN$;:NEXT I%:PRINT:'Clear DOS commands from screen'
410 ECHO$=" "
420 WHILE ECHO$<>"y" AND ECHO$<>"N"
430 GOSUB 1070
440 LINE INPUT "Do you want echo to screen (Y/N):",ECHO$
450 WEND
470 ROW=CSRLIN:'Store current line number'
480 EEOPT$=" ":'Initialise option char'
490 WHILE EEOPT$<>"X" AND EEOPT$<>"I" AND EEOPT$<>"V"
500 GOSUB 1070
510 LINE INPUT "Select Internal,external or Verify EEPROM option (I/X/V):",EEOPT$
520 WEND
530 OPT$="Verify"
540 IF EEOPT$="I" THEN OPT$="Internal"
550 IF EEOPT$="X" THEN OPT$="External"
560 ROW=CSRLIN:'Store current line position in case of file error'
570 RXERR=0:  'Initialise number of RX errors allowed'
580 ON ERROR GOTO 910
590 GOSUB 1070
600 IF OPT$="Verify" THEN INPUT "Enter filename to verify:".F$ ELSE INPUT "Enter file-
name to download:".F$
610 CLOSE
620 OPEN F$ FOR INPUT AS #2
630 ON ERROR GOTO 0
640 'COM1 or 2 connected to SCI on HC11'
650 OPEN CM$+",9600,N,8,1" AS #1
660 'Establish contact with HC11 by sending CR char & waiting for echo'
670 ON ERROR GOTO 860:' Clear potential RX error'
680 PRINT #1,CR$;
690 GOSUB 990:'read char into B$'
700 'Transmit Internal ,External or Verify EEPROM option char to 68HC11'
710 PRINT #1,EEOPT$;GOSUB 990:        'NO echo to screen'
720 ON ERROR GOTO 930
730 PRINT " Starting download of <";F$;"> to:";OPT$;" Eeprom"
732 IF ECHO$="Y" THEN E%=1 ELSE E$=0
734 IF EEOPT$="V" THEN V%=1 ELSE V%=0
740 WHILE NOT EOF(2)
750 INPUT #2,S$
751 L%=LEN(S$)
```

```
752 FOR I%=1 TO L%
760   PRINT #1,MID$(S$,I%,1);:GOSUB 990:IF E% THEN PRINT B$;
770   IF V% THEN GOSUB 1030:IF C$<>""THEN PRINT ":";HEX$(ASC(C$));
785 NEXT I%
787 IF E% THEN PRINT
790 WEND
795 PRINT
800 PRINT "Download Complete"
810 IF V% THEN PRINT ERRTOT%;"error(s) found"
820 CLOSE #2
830 SYSTEM
840 END
850 '.....'
860 IF RXERR>5 THEN 940 ELSE RXERR=RXERR+1:RESUME 610
870 '.....'
880 PRINT:PRINT ERM$;LOADER$:PRINT "Program aborted"
890 GOTO 830
900 '.....'
910 PRINT ERM$;F$;SPACE$(40)
920 RESUME 580
930 '.....'
940 PRINT:PRINT "Communication breakdown: Download aborted"
950 GOTO 820
960 '.....'
970 '...SUB waits for received character,with time limit'
980 '..  returns with char in B$,or aborts if time limit exceeded'
990 TO%=0:WHILE LOC(1)=0:IF TO%>100 THEN 940 ELSE TO%=TO%+1:WEND
1000 B$=INPUT$(1,#1):RETURN
1010 '.....'
1020 '..SUB waits for received character,with time limit'
1025 '..  returns with char in C$,or null in C$ if time limit exceeded'
1030 TO%=0:C$="":WHILE LOC(1)=0 AND TO%<1:TO%=TO%+1:WEND
1040 IF LOC(1)>0 THEN C$=INPUT$(1,#1):ERRTOT%=ERRTOT%+1
1050 RETURN
1060 '.....'
1070 '.. SUB Clear line'
1080 LOCATE ROW,1,:PRINT CLRLN$
1090 LOCATE ROW,1,:RETURN
1100 '.....'
```

4.7 EEPROGIX.ASC 程序

该程序将 S 记录从主机装入到与 M68HC11 总线相连的外部 EEPROM 或 M68HC11 内部 EEPROM 中。每个装入的字节都返回到主机。当编程 2864 时，采用查询方式检测编程周期

是否完成。由于主机软件在装入下一个字节之前总是等待返回数据，因此，在数据查询期间主机暂时停止。

当编程内部 EEPROM 时，不采用查询方法，编程后每个字节都进行校验。

若发生错误，程序停止。主机装入程序应检测这一状态并采取补救措施。BASIC 程序 EELOAD 只显示“communication breakdown”，并终止程序。

当进行校验时，除正常返回每个字符外，存储器与 S 记录数据的任何差别也送回到主机。主机软件必须能够检测这一情况，并执行所需的操作。BASIC 装入程序 EELOAD 只简单地返回有错的字节，它位于所需字节旁边，用分号隔开。

在接收 S 记录之前，先从主机接收控制字符：

ASCII "X" 用于编程外部 EEPROM

ASCII "I" 用于编程内部 EEPROM

ASCII "V" 用于检验 EEPROM

```
*   该程序与 BASIC EELOAD 一起使用。
*   数据通过 SCI 传输，SCI 设置为 8 个数据位，9600 波特
*   常数
TDRE    EQU     $80
RDRF    EQU     $20
MDA     EQU     $20
SMOD    EQU     $40
mS10    EQU     10000/3     10ms 延迟(8MHz 晶体)
uS500   EQU     500/3       50μs 延迟
*   寄存器
BAUD    EQU     $2B
SCCR1   EQU     $2C
SCCR2   EQU     $2D
SCSR    EQU     $2E
SCDR    EQU     $2F
PPROG   EQU     $3B
HPRIO   EQU     $3C
CONFIG  EQU     $103F
*   变量
        ORG     $0
EEOPT   RMB     1
MASK    RMB     1
TEMP    RMB     1
LASTBYTE RMB    1
*   程序
        ORG $0
        LDS # $FF
        LDX # $1000             控制寄存器的偏移量
        CLR SCCR1,X             SCI 初始化为 8 个数据位，9600 波特
        LDD # $300C
        STAA BAUD,X
```

```
                STAB SCCR2,X
                BSET HPRIO,X #MDA       首先强迫为测试模式
*               BCLR HPRIO,#SMOD        扩展模式
                                        省缺状态为内部 EEPROM：EEOPT = 0；
READOPT         STS <EEOPT              MASK = $FF
                BSR READC               然后检查内部或外部控制字节
                CMPB #'1'               EEPROM 选择
                BEQ LOAD
                CMPB #'X'               若为外部 EEPROM
                BNE OPTVERF
                INC EEOPT               则改变选择为 1
                LDAA #$80
                STAA <MASK              选择数据查询
                BRA LOAD
OPTVERF         CMPB #'V'               若不检验，则
                BNE READOPT             下一个字符
                DEC EEOPT               使 EEOPT 标志为负
LOAD            EDU *
                BSR READC
                CMPB #'S                一直等待到接收到 S1 或 S9
                BNE LOAD                忽略以前 S1 的检验和
                BSR READC
                CMPB #'1
                BEQ LOAD1
                CMPB #'9
                BNE LOAD
                BSR RDBYTE              终止前读完 S9 记录
                TBA
                SUBA #2
                BSR GETADR              取 Y 中的执行地址
LOAD9           BSR RDBYTE              丢弃其余字节，包括检验和
                DECA
                BNE LOAD9
                CPY #0
                BEQ *
                JMP,Y
LOAD1           EQU *
                BSR RDBYTE              将 S1 记录的字节数读入 ACCB
                TBA                     并存入 ACCA
                SUBA #3                 去除地址和检验和字节数
                BSR GETADR              地址送入 X 寄存器
                DEY                     调整
                BRA LOAD1B
LOAD1A          LDAB EEOPT              更新 CCR
                BMI VERIFY              若不检验
                BEQ PATAPOLL            若编程外部 EEPROM
```

```
                LDAB #μS500
WAIT1           DECB                    等待最大 500μs
                BNE WAIT1
DATAPOLL        LDAB,Y                  测试写入到存储器的最后字节
                EORA <LASTBYTE          的最高位,等待编程完成;
                ANDB <MASK              或检验内部编程数据
                BNE DATAPOLL
LOAD1E          DECA
                BEQ LOAD                结束后取下一个 S 记录(忽略检验和)
LOAD1B          BSR RDBYTE              将下一数据字节读入 ACCB
                INY
                TST EEOPT
                BMI LOAD1D              若检验,则不编程
                BEQ                     若选择内部 EEPROM,则编程
                STAB,Y
LOAD1D          STAB <LASTBYTE
                BRA LOAD1A
VERIFY          LDAB,Y                  若编程字节正确,则读下一个字节
                CMPB <LAST BYTE
                BEQ LOAD1E              否则,在读下一个字节之前
                BSR WRITEC              将错误字节送回主机
                BRA LOAD1E
READC           EQU *
                BRCLR SCSR,X,#RDRF,*
                LDAB SCDR,X             读下一个字符
WRITEC          BRCLR SCSR,X,#TDRE,*
                STAB SCDR,X             并返回到主机
                RTS
RDBYTE          BSR READC               先读高 4 位,转换为二进制
                BSR HEXBIN
                LSLB                    移为高 4 位
                LSLB
                LSLB
                LSLB
                STAB <TEMP
                BSR READC               取 ACCB 的 ASCII 字符
                BSR HEXBIN
                ORAB <TEMP
                RTS
GETADR          EQU *
                PSHA                    存字节计数器
                BSR RDBYTE              读地址高位字节
                TBA                     并将其放入 ACCD 的高位字节
                BSR RDBYTE              将低位地址字节读入 ACCD 的低位字节
                XGDY                    将程序地址放入 Y
                PULA                    恢复字节数器
```

```
        RTS                     返回
*
NEXBIN  EQU *
        CMPB #'9                若 ACCB>9,则认为为 A~F
        BLS HEXNUM
        ADDB #9
HEXNUM  ANDB #$F
        RTS
*
PROG    EQU *
        PSHA
        LDAA #$16               省缺状态为擦除方式
        CPY #CONFIG             若字节地址为 CONFIG
        BNE PROGA
        LDAA #$06               则使用整片擦除
PROGA   BSR PROGRAM             擦除字节,或整个存储器和 CONFIG
        LDAA #2
        BSR PROGRAM             编程字节
        CPY #CONFIG             若字节为 CONFIG 寄存器
        BNE PROGX
        LDAB,Y                  则用H值装入 ACCB
PROGX   PULA                    恢复 ACCA
        BRA LOAD1D
*
PROGRAM EQU *
        STAA PPROG,X            允许内部地址/数据锁存
        STAB,Y                  向所需地址写入
        INC PPROG,X             允许内部编程电压
        PSHX
        LDX #MS10               等待 10ms
WAIT2   DEX
        BNE WAIT2
        PULX
        DEC PPROG,X             禁止内部编程电压
        CLR PPROG,X             禁止内部数据/地址锁存
        RTS                     返回
```

5. 减小 MCU 中 A/D 转换器的误差

许多应用场合都需要将模拟信号转换为数字量。当今越来越多的集成电路芯片都将 A/D 制作在内，这样系统更加紧凑，可靠性更高，应用也更方便。最有代表性的产品就是单片机(MCU)。MCU 片内的 A/D 还简化了模拟接口问题，软件也更为灵活。

毫无疑问，将外围器件集成在 MCU 片内比外接方法有许多优点。但由于线性系统的频带宽(如 A/D)，内部数字系统(如 CPU)速度高，使用不当会影响 A/D 转换器(ADC)的精度。在任何情况下，MCU 系统内置 ADC 的性能并不比外接 ADC 的系统的性能差。在 MCU 设计阶段采用一些适当的措施，就能避免潜在的问题。

下面的应用问题虽然是针对片内 ADC 的，但对于有多个芯片组成的含有 A/D 的系统，下面的论述也适用。

5.1 ADC 的类型与噪声的影响

了解 ADC 的机理有助于在使用时改善 A/D 的特性。ADC 分为五类：积分式、量化反馈式、并行比较式、逐次比较式和混合式。

积分式 ADC 的分辨率高(16 位或更高)，价格低，硬件电路简单，结构如图 5-1(a)所示。但其速度很低，一般很少集成在 MCU 内。

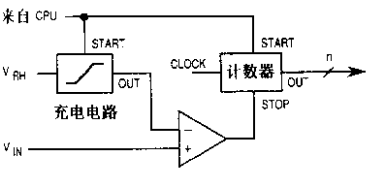
量化反馈式 ADC 具有很高的分辨率和转换时间，结构如图 5-1(b)所示。速度最快的是并行比较式 ADC，如图 5-1(c)所示，其频带通常超过 100MHz。这种 ADC 需要大量的比较器，硬件复杂，分辨率低，通常只有 4～6 位。输入电压加到所有比较器，经过比较器的延迟时间后，各比较器具有相应的输出值。将这些比较器的输出值进行译码可得对应输入模拟量的数字值。因这种结构太复杂，该 ADC 应用较少，在 MCU 中更是如此。

第四种 ADC 就是逐次逼近转换器(SAC)，其结构如图 5-1(d)所示。逐次比较 A/D 是最常用的 ADC，MCU 中的 A/D 也基本都是这种类型。这种 ADC 的特点是电路较为简单，转换速度适中，具有中高分辨率(8～16 位)。象其他 ADC 一样，SAC 也是用一个差分比较器使输入电压与基准电压进行比较。每一个时钟确定一位二进制码。通常只需要一个比较器，因此电路简单，硅芯片面积小。这种 ADC 在 MCU 中极为流行，如 M68HC11、M68HC05 系列 MCU 中的 A/D 都是逐次比较型。

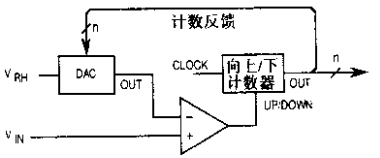
第五种 ADC 是混合型，它将多种 ADC 类型组合在一起实现 A/D 转换器，以获得所需的性能，例如，一些高速和高分辨率的 ADC 采用并行比较和逐次比较的方法。这种混合型 ADC 可能 的应用场合是需要高分辨率及高速转换的 MCU 系统中。

A/D 中的一个主要部件是比较器，其输出反映输入微小的电压变化。电压瞬间高速变化会产生噪声，影响 ADC。比较器有两方面的特性对噪声有影响：带宽和电源连线。

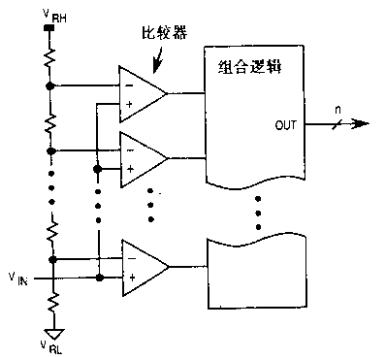
频带宽度对噪声有密切关系。例如，对于单斜率积分型 ADC，当比较器的一个输入端电压升高时，计数器开始计数，当积分到某一值时，比较器改变状态。如果比较器不能及时响应电压变化，计数器所寄存的计数值就不是正确的数值。这要求响应时间(主要取决于带宽)必须能很快地响应(既使是在慢速 ADC 中也是如此)。而宽频带比较器同时也会对叠加在信号上的噪



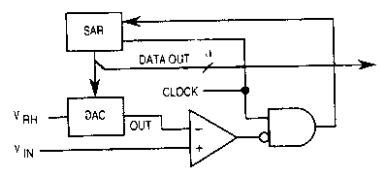
(a) 积分式 ADC



(b) 量化反馈 ADC



(c) 并行比较式 ADC



(d) 逐次逼近 ADC

图 5-1 ADC 的类型

声进行响应。

由于 ADC 中的比较器频带宽，系统设计时必须特别注意信号类型和电源线。因此，电路板和电路连线对成功地设计 MCU/ADC 系统是至关重要的。

5.2 合理的设计技术

MC68HC11E9 含有 ADC 子系统，如图 5-2 所示。通常，噪声对数字系统的影响远低于对模拟系统的影响。例如，对于 5V 电源电压的系统，100mV 的宽带噪声使输出高电压下降 100mV，这对逻辑电路没有什么影响。但 100mV 的噪声对于 ADC 系统的影响是很大的。例如，对于 8 位 A/D，基准电压为 5V，则 1LSB 对应的模拟电压约为 20mV，100mV 噪声会造成 5LSB 的误差。

为避免 MCU/ADC 系统噪声的影响，需要将有噪声的信号与易于受噪声影响的信号隔离开。隔离程度越高，则越好。对于有噪声的电路与噪声敏感的电路不能分离的情况，则应尽量避免它们之间的噪声耦合。

通常，功能失常主要是由于印刷电路板(PCB)的设计。因为 PCB 影响许多电路参数，PCB 的设计会提高或妨碍 ADC 的性能。PCB 的设计，包括印制线的宽度和位置等，对于许多电路设计者来说并不是十分了解。仅仅将需要连接起来的引脚连在一起是完全不够的，这样会降低 ADC 的性能，图 5-3 是一个 PCB 设计的例子。它虽然在电路功能上是正确的，但潜在有许多问题。

首先，MCU/ADC 离电源最远，MCU 的返回电流将与数字电路的信号电流混合在一起，尤其高速逻辑电路更是如此。在高频下，PCB 印制线的寄生电感也会在 M68HC11 地引脚和电路板地线端子之间产生明显的噪声。

第二，运算放大器(opamp)对传感器(Sensor)信号进行放大和缓冲。再与 MCU 的 ADC 相连。虽然 opamp 与 MCU 接近，但从电特性上讲，其位置很差。逻辑信号产生的高频噪声会通过电源线加到 opamp 上，运放与 MCU 之间的电压差比 MCU 到地线端的电压还大。因此，放置各器件位置时，引线之间的相互影响必须充分考虑。

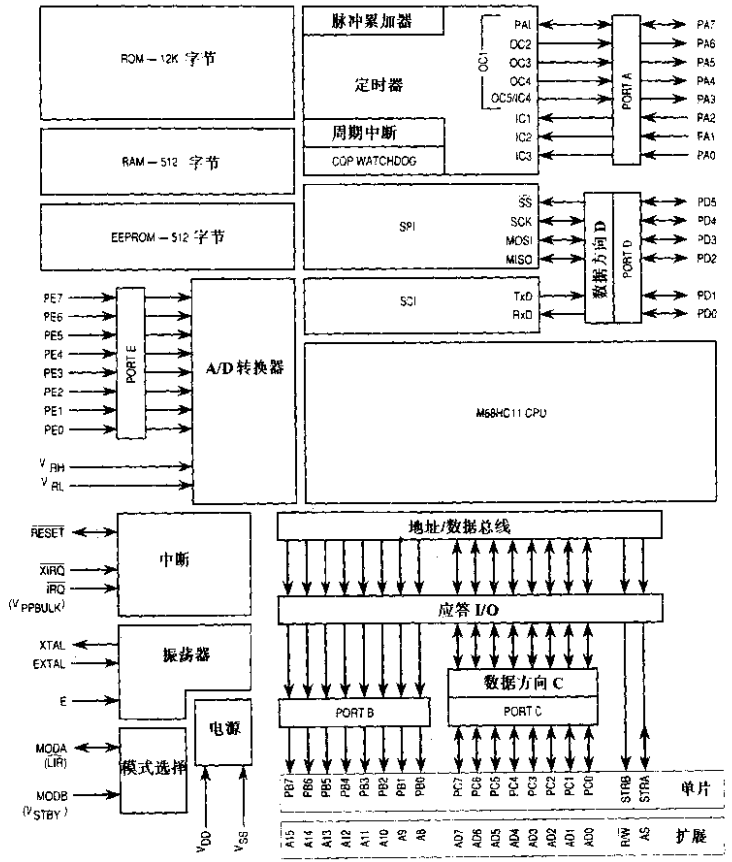


图 5-2 MC68HC11E9 的框图

第三，图中的旁路电容对于减小 V_{DD} 的高频噪声是无效的。为正确地去耦，旁路电容应尽量接近器件的电源引脚。另外，应该使电容和电源引脚的之间引线的 PCB 印制线电感最小。

含有 MCU/ADC 系统的 PCB 必须精心设计。合理的设计如图 5-4 所示，它没有图 5-3 存在的问题。

5.3 带 ADC 的专用 MCU

M68HC11 系列 MCU 含有逐次逼近型 A/D 转换器(SAC)，片内的 A/D 为 8 位，转换时间 16μs。其数-模转换器(DAC)用电容实现，而不是通常的 R-2R 电阻网络。虽然薄膜 R-2R 电阻网络的温度稳定性好，但基本都需要激光修正才能保证 ADC 精度。另外，调整电阻值会改

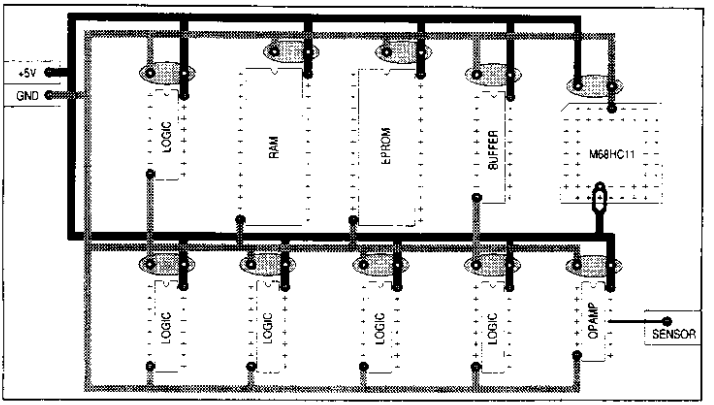


图 5-3　不合理的 PCB 设计

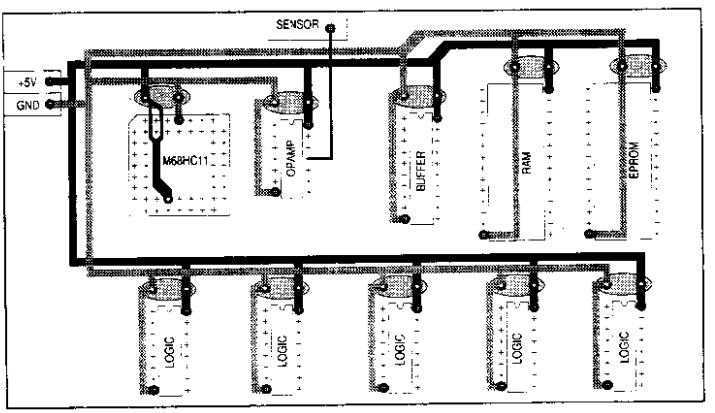
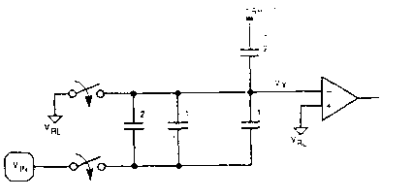


图 5-4　合理的 PCB 设计

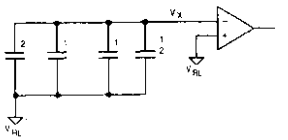
变电流，因此，需要多次重复修正。还需要精确地控制 MOS 开关的导通电阻。M68HC11 的电容 DAC 可避免上述这些问题。工艺上很容易实现电荷再分布，不需对电容和 MOS 开关进行修改微调。另外的优点是具有抽样保持功能，使 ADC 具有很高的频带。

下面以 2 位 ADC 为例说明 M68HC11 中 ADC 的转换过程。ADC 的工作分三个方式：抽样、保持和逼近，如图 5-5 所示。在图 5-5(a)的抽样方式中，上电极接 V_L(0V)，下电极接输入

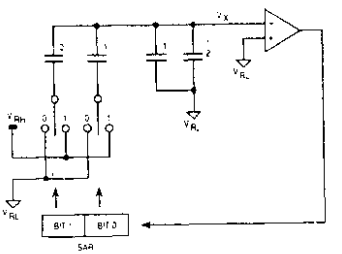
电压(V_{IN})，电容上的电荷正比于输入电压，即 V_X 正比于 V_{IN}。在图 5-5(b)的保持方式，上边的开关断开，下面的开关连到 V_L，V_X 点的电压变为负。在逼近模式，图 5-5(c)所示，依次将电容(首先是最大的电容)接 V_H，确定数字量各位(首先是最高位)是 1 还是 0。若最高位为 1，则对应电容还保持接 V_H，否则接 V_L，然后确定次最位，余类推。



(a) 抽样模式



(b) 保持模式



(c) 逼近模式

图 5-5 逐次逼近 ADC

下面外部电路参数影响 ADC 的误差：

5.3.1 ADC 输入引脚的漏电流

M68HC11 ADC 输入引脚的模型如图 5-6 所示。N 沟器件的漏电流会造成误差，并且温度越高，误差越大。该漏电流为固定流入；最大为 400mA，故它造成的误差总是使 ADC 的结果减小。ADC 模拟输入引脚的电阻不能太大。例如，基准电压为 5V，希望漏电流造成的误差小于 1LSB，则外接电阻不应该超过 20mV/400nA＝50kΩ。

5.3.2 充电时间和抽样电容

增大由 DAC 引脚源电阻和抽样期间 DAC 电容组成的时间常数(RC)，可能会造成误差。对于一定的 ADC 电容，减小 RC 时间常数造成的误差对电阻的要求与减小漏电流的影响是一致的。

5.3.3 V_{DD}/V_{SS}输入端噪声

M68HC11ADC 中不同的比较器从不同地方获得 V_{DD}和 V_{SS}。对于 ADC 来说，M68HC11 是噪声源，主要是由于方波的波形和谐波。另外，许多软件组合使时钟与噪声电压的关系更加复杂。因此，因 V_{DD}/V_{SS}产生的噪声而使 ADC 性能下降经常发生在执行某特殊软件程序的情况。

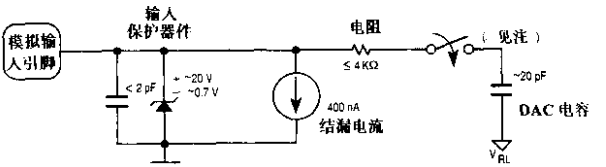


图 5-6 M68HC11 ADC 输入引脚的电模型

注：该模拟开关只在 12 个周期的抽样期间闭合。

注意，因为 M68HC11 使用宽带比较器，能响应 20MHz 以上的噪声分量，故必须要保证在输入端和 V_{DD}/V_{SS}引脚没有噪声。

若信号加到比较器的两个输入端，则由于比较器的共模抑制特性(CMRR)，可有效地减小

其影响。如果不能将噪声消除，应使噪声呈现在比较器的两个输入端，则可发挥 CMRR 共模抑制特性。

5.3.4 宽带输入信号

因比较器的频带很宽，当存在持续时间很短的尖峰噪声信号时，比较器会对噪声进行响应，也会造成错误。

5.4 实例

某工业控制组件使用扩展模式的 M68HC11。用户设计的程序存储器为 32K×8，RAM 为 2K×8，以及外部地址译码器、I/O、模拟缓冲器等。电路板为 6 层 PCB、独立的电源和地平面。

5.4.1 存在的问题

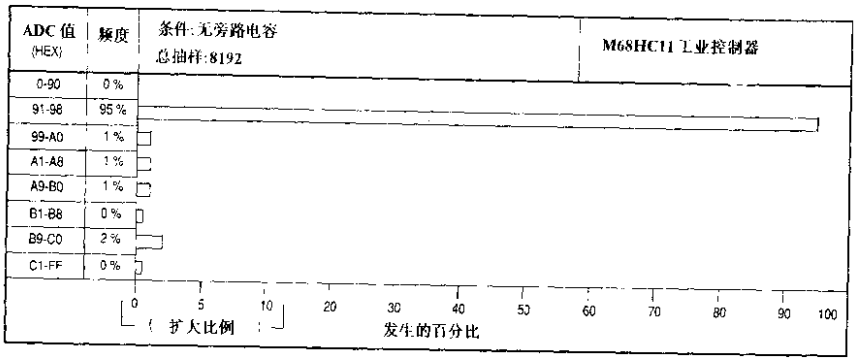
该系统用于检测某种危险状态。为获得最安全的响应时间和最大安全裕量，要求 64μs 连续转换(大于 15kHz 抽样频率)，转换结果与实际系统的测量值偏差不能超过±2LSB。选用的 M68HC11 精度为±1LSB。应用中会遇到误差高出许多倍的情况。如图 5-7 所示，旁路电容对 ADC 性能的影响是明显的。图中纵坐标为 A/D 转换值，横坐标为 A/D 转换值位于所选范围的百分比(频度)。可见，旁路电容可使正确 A/D 转换改善 3%。

5.4.2 理想的电路

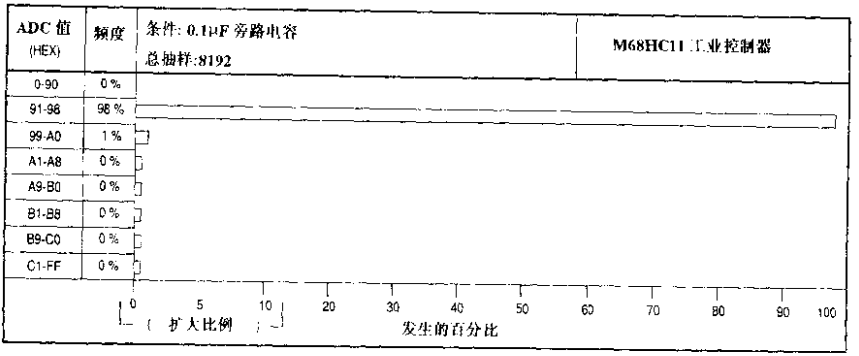
实际 ADC 失效与 $1V_{pp}$ 噪声有关(该噪声的时间间隔大约与 M68HC11 的 E 时钟速率相同)，它由 M68HC11 的 V_{DD} 引脚产生。因此，应尽量减小这一 V_{DD} 噪声。将 M68HC11 的电源总线隔离能减小这一噪声。将 M68HC11 隔离供电后，基本 100%ADC 值均位于期望范围内，只有 0.5%位于期望值范围以外。但在各个数值范围段(如 $99～$A0，$A1～$A8，…)都存在误差读数值，类似于图 5-7。将 M68HC11 电源线隔离并且增加旁路电容后(V_{DD} 与 V_{SS} 引脚之间)，结果比未加旁路电容时明显改善，且没有 ADC 数值位于 $B1～$B8 之内。

再进一步减小 ADC 误差只能在 MCU 与输入信号接口电路方面采取措施。线性模拟信号缓冲器为单位增益的 324 运放，它也具有较宽的频带。而来自各种器件的与 324 缓冲器相连的信号是缓慢变化信号，频率在 500Hz 以下。应滤除最高有用信号频率以上的频带。通过适当的滤波，高频信号噪声不会通过输入端进入到 ADC。M68HC11 的 ADC 输入引脚接 0.01μF 的滤波电容，ADC 测量结果如图 5-8 所示，可见有很大改善，但仍有部分数值在指定的 ADC 范围之外。

另一个减小误差的措施是减小相邻电路之间不需要的干扰(相互影响)。隔离 V_{DD} 的另一种方法是串联二极管，结果表明所有 ADC 值均位于希望值范围以内，如图 5-9(a)所示，完全消除了误差。为进一步检查其可靠性，将 ADC 值缩小为 $94～$96 之间，即±1LSB，也完全没有误差，如图 5-9(b)所示。



(a) 无旁路电容



(b) 有旁路电容

图 5-7 只有旁路电容的影响

注:数值四舍五入为 10%,频度为 0%并且幅度很低情况表示大于 0%,小于 0.5%。

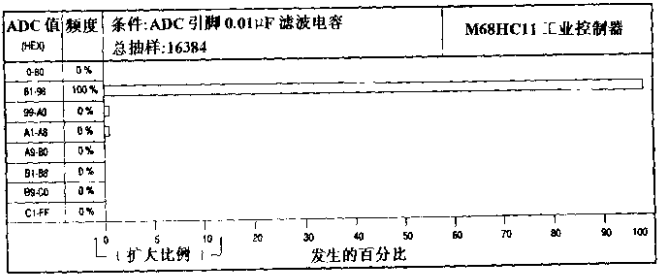
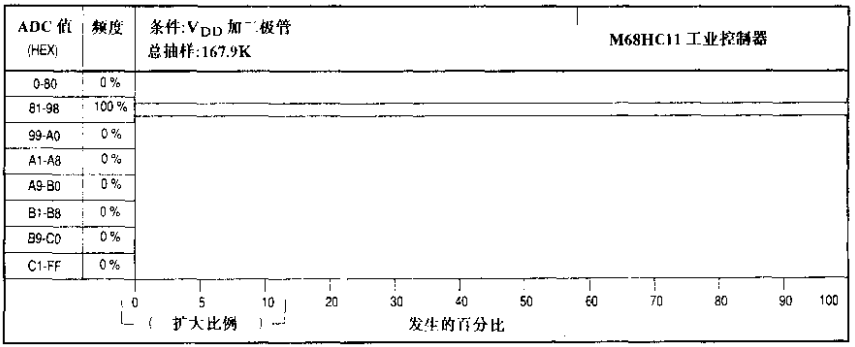
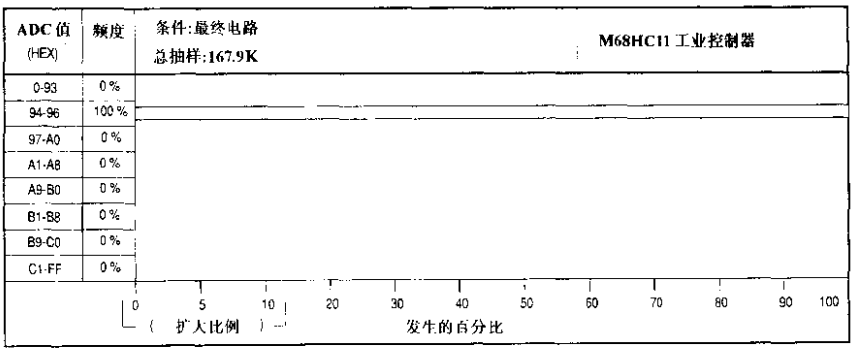


图 5-8 ADC 输入引脚加滤波电容的结果

注:读数值舍入到 1%,频度为 0%且幅度很小者表示大于 0%但小于 0.5%。



(a) V_{DD} 有二极管



(b) 减小 ADC 范围

图 5-9 V_{DD}串联二极管和缩小 ADC 范围的结果

6. M68HC11 的自引导模式

M68HC11 的自引导模式允许用户定义的程序由串行通讯接口(SCI)装入到内部 RAM 中,然后 M68HC11 执行该装入程序。该装入程序可以执行任何通常用户程序能执行的操作或进行工厂测试。虽然自引导模式为单片操作模式,但可利用扩展模式资源,因为在自引导模式下工作时能改变模式控制位。

本节主要讨论 M68HC11 的自引导模式的操作和应用,包括:

- M68HC11 自引导模式的基本操作
- M68HC11 自引导模式的使用
- 片内自引导逻辑的详细说明
- 自引导固件(Firmware)的详细说明
- 自引导固件与 EEPROM 保密
- 由另一个 M68HC11 MCU 驱动自引导模式
- 由 PC 机驱动自引导模式
- M68HC11 各种型号的自引导方式
- M68HC11 自引导 ROM 程序

6.1 基本自引导模式

当 M68HC11 在自引导模式下复位时,从内部自引导 ROM(bootstrap ROM 或 boot ROM)寻取复位向量。该自引导 ROM 的固化程序然后控制自引导装入过程。首先,初始化片内 SCI。最先接收到的字符($FF)决定以后装入操作使用两种波特率中的哪一种。下一步,SCI 系统接收二进制程序并存储于 RAM,最后,执行转移指令,控制从自引导程序转向用户装入的程序,自引导模式对器件和最终用户系统都很有用。

自引导模式的用途之一是控制片内 EEPROM 烧入程序。在 MCU 装入到最终产品之前,用户也可以用这一技术将程序装入到 M68HC11 的 EPROM 或 EEPROM 中。对于 M68HC11EVM 评估模块,也是用自引导模式对目标器件进行编程。也可用自引导模式对设置寄存器 CONFIG 进行编程。

自引导模式的另一种用途是产品最终测试。可将小型程序装入,以检验系统的各个部分,包括 MCU 外部的器件和电路,如果在开发过程中出现问题,可以将诊断程序装入来查找存在的问题,并将正确的程序装入进行检验。然后再将其组合到应用程序中。

还可以用自引导模式来校准关键的模拟信号传感器。由于这种校准是在最终系统组装完之后进行的,故能补偿在接口电路、传感器至 MCU 之间的电缆等产生的误差。

6.2 自引导模式逻辑

M68HC11 有一小部分逻辑专门用于自引导模式,主要是自引导 ROM。M68HC11A8 中的自引导 ROM 为 192 字节,有些型号达 448 字节。通常,只有当 MCU 在自引导方式下复位时,自引导 ROM 才出现在存储器映象中,以防止与通常的用户存储器空间冲突。最高优先中

断寄存器(HPRIO)中的 boot ROM 控制位(RBOOT),控制是否允许该自引导 ROM。当 MCU 处于特殊测试和特殊自引导模式时,可由软件写入 RBOOT 位。当 MCU 处于正常模式时,只能读 RBOOT 位。

图 6-1 是 MC68HC711E9 四种操作模式的存储器映象。HPRIO 中的 MDA 位和 SMOD 位决定为哪一种操作模式。而这两位又取决于复位期间 MODA 和 MODB 引脚的电平,如表 6-1 所示。

表 6-1 MC68HC11 模式选择

输入引脚		选择的模式	HPRIO 的控制位		
MODB	MODA		RBOOT	SMOD	MDA
1	0	正常单片	0	0	0
1	1	正常扩展	0	0	1
0	0	特殊自引导	1	1	0
0	1	特殊测试	0	1	1

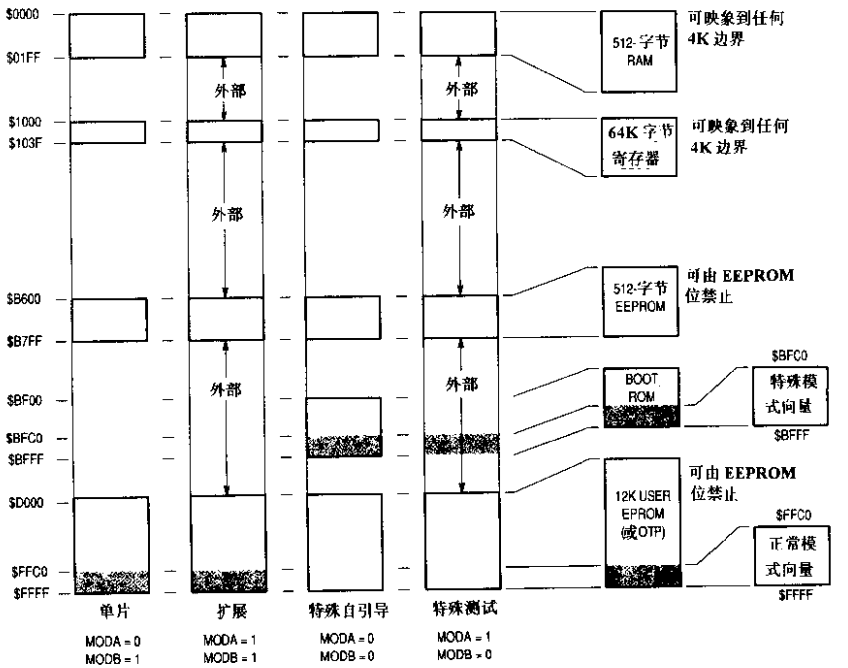


图 6-1 MC68HC711 E9 四种操作模式的存储器映象

MDA 控制位由 MCU 退出复位时 MODA 引脚的状态决定,它决定选择单片还是扩展操作模式。若 MDA 为 0,选中单片模式,即为正常单片模式或特殊自引导模式;若 MDA 为 1,选中扩展模式,即为正常扩展模式或特殊测试模式。

SMOD 位取决于 MCU 退出复位时 MODB 引脚的反相状态，SMOD 为 0 时，选中正常模式，即为正常单片或正常扩展模式；SMOD 为 1 时，选中特殊模式，即为特殊自引导或特殊测试模式。当为任一种特殊模式时(SMOD=1)，可以写入模式控制位，并且从 $BF××而不是从 $FF×× 寻取复位和中断向量。

若 SMOD 为 1，在所有向量寻取期间使地址线 A14 为低可获得改变的向量地址。对于特殊测试模式，该向量地址可保证从外部存储器空间能寻取复位向量，因此测试系统能控制 MCU 操作。在特殊自引导模式，将 RBOOT 置 1 则允许自引导 ROM 出现在存储器映象中，将从该 ROM 寻取复位向量，自引导程序将控制 MCU 操作。

在自引导模式 RBOOT 复位为 1，以允许自引导 ROM，在其他三种模式，RBOOT 均复位为 0。而在特殊测试模式，SMOD=1，允许用软件将 RBOOT 位写为 1，从而可将自引导 ROM 用于测试。

6.3 自引导 ROM 固化程序

自引导 ROM 中的主要程序是自引导装入程序(bootloader)，它在设置 MCU 为自引导方式后自动执行。有些型号还具有装入程序可以调用的其他程序。

某些 M68HC11 型号具有 EEPROM 保密功能。将 CONFIG 寄存器中的 EEPROM 保密位(VOSEC)编程可进行加密。一旦将 NOSEC 编程为 0，MCU 将忽略 MODA 引脚的逻辑状态，复位后，总是为正常单片模式或特殊自引导模式。

6.4 波特率的自动选择

MC68HC711E9 的自引导装入程序适于两种波特率。较高的波特率为 7812 波特(2MHz E 时钟，8MHz 晶体)或 8192 波特(2^{23}Hz=8.389MHz 晶体)，另一种波特率为 1200 波特(2MHz E 时钟)。省缺状态为 7812 波特。

图 6-2 表明了如何区分 7812 波特和 1200 波特。主机发送 $FF 字符，它主要是用来由自引导装入程序决定装入操作所用的波特率。对于上边的图形，接收的数据在[1]抽样，决定起始位的下降沿，并在起始位期间中点抽样，确认下降沿。然后在每个位时间(bit time，一位数据占用的时间)的中点抽样[2]，以确定接收的字符值。最后在停止位时间中点抽样。之后接收数据线空闲[4](除非紧随 $FF 后立即发送其他字符)。

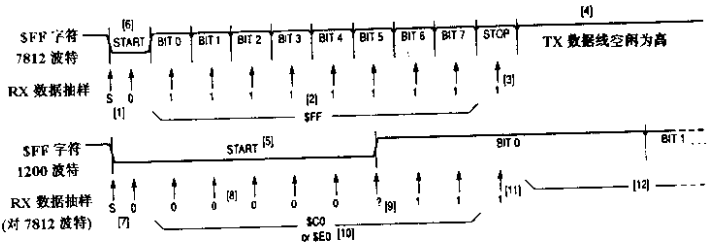


图 6-2 波特率的自动检测

在图 6-2 下半部分图形中，若主机以 1200 波特速率发送 $FF，则不能正确地接收 $FF(因为最初设置为 7812 波特)。也是在相同的时刻抽样，1200 波特的起始位时间[5]为 7812 波

特的起始位时间的 6.5 倍[6]。在[7]检测起始位的下降沿并确认，在[2]期间抽样检测值为逻辑 0。位 5 可能为 1 或 0。接收器认为接收到的字符为 $C0 或 $E0[10](用 7812 波特接收以 1200 波特发送的 $FF)。停止位也为 1[11]。之后为空闲状态[12]。这样，可以区分两种波特率，再按接收的数据设置为相同的波特率。若开始不是发送 $FF 而是其他字符，则导致 SCI 接收错误。

6.5 自引导装入主程序

图 6-3 是 MC68HC711E9 的自引导装入主程序。

自引导 ROM 的复位向量指向该程序的开始处[1]。初始化程序[2]建立初始状态并设置 SCI 和 D 口。X 变址寄存器指向寄存器块($1000)的开始处，因此可以用变址寻址。变址寻址占用的 ROM 字节比扩展寻址指令少，并且位处理指令不能用扩展寻址方式。串行外围接口(SPI)控制寄存器(SPCR)的 D 口线或模式位(MODE)被置位，D 口被设置为线或方式，以避免与使用 PD1/TXD 引脚的外部系统发生冲突。SCI 的波特率开始为 7812 波特(2MHz E 时钟)，接收到第一个字符后能自动设置为 1200 波特，SCI 发射和接收器均被允许。自引导装入过程需要接收器，发送器是用来向主机回送数据，以便校验。初始化[2]中的最后一项是设置字符间的延迟时间，以便当主机停机向 MC68HC711E9 发送数据时，终止装入过程。延迟常数存储在定时器输出比较 1(TOC1)寄存器中。自引导装入程序不使用定时器，这里用 TOC1 是减少所用的存储器。

初始化后，SCI 发送中止(break)字符[3]。接收到该 $00 字符后，立即转向片内 EEPROM 起始处[4](对 HC711E9 为 $B600)。

若首先接收到 $FF，则波特率认为是省缺值，即 2MHz E 时钟情况下为 7812 波特。如果主机以 1200 波特发送 $FF，则 SCI 接收到的数据为 $E0 或 $C0(因为二者波特率不匹配)，并且自引导装入程序转换为 1200 波特[5]，以接收其他装入程序。另外，延迟时间常数也改变为相应的新的值。

在[6]，变址寄存器 Y 初始化为 $0000，指向片内 RAM 的起始处。Y 用来指定下一个接收的数据字节将存储的 RAM 位置。主循环从[7]开始。

装入程序的字节数可为 0～512 字节之间的任何数。当最后装入的字符后至少有 4 个字符时间为空闲时间时，结束装入过程。对于 M68HC11 系列中只有 256 字节 RAM 的型号，装入的长度固定为 256 字节再加上 $FF 字符。

延迟计数器从[8]开始，它将 TOC1 的延迟常数装入到变址器 X 中。重复执行 19 个 E 周期的等待循环，直到接收到字符[9]或延迟时间到期[10]。对于 7812 波特，延迟常数为 10,241 个 E 周期(539×19)。4 个字符时间(7812 波特)为 10,240 个 E 周期(波特率预分频 4×波特率分频率 4×16 个内部 SCI 时钟/位时间×10 个位时间/字符×4 个字符时间)。从复位到初始 $FF 字符的延迟并不重要，因为直到接收到 $FF 后延迟计数器才开始计数。

为了终止自引导装入过程并转移到 RAM 开始处，只需发送 $FF。这与转移到 EEPROM 情况[4]类似。这要求 RAM 已装有有效的程序。

接收到字符后，装入的字节存储于 RAM 中[11]。该数据再回送到主机[12]，以表明装入过程正常进行。在[13]，RAM 指针加 1，指向下一个 RAM 地址。若 RAM 指针未到结束处，则从[7]至[14]重复。

装入了所有数据后，自引导程序转向[16]，因为达到规定延迟时间或已装入整个 512 字

节。在[16],X 和 Y 变址寄存器设置为调用 PROGRAM 子程序。最后,自引导装入程序转移到[17]。

6.6 UPLOAD 子程序

该 UPLOAD 程序通过 SCI 将数据从 MCU 传输到主机。注意,具有 EEPROM 保密功能的型号没有这一程序。在调用该程序之前,Y 变址寄存器被装入回送的第一个数据字节的地址。如果回送数据不使用目前的 SCI 波特率,用户程序必须写入 BAUD 寄存器确定波特率。UPLOAD 程序从 SCI 发送器发送连续的数据字节,直到发生复位(无限次循环发送)。

6.7 EPROM 编程(PROGRAM 程序)

PROGRAM 程序将数据编程到 MC68HC711E9 的内部 EPROM 中。它需要 12V 外部编程电压。使用这一程序的最简单方法是只装入含有转移到 PROGRAM 程序开始处($BF00)的转移指令的 3 字节程序。自引导装入程序在转移到装入的程序[16]之前设置 X 和 Y 为省缺值。当主机收到 $FF 后,发送 EPROM 编程数据,从 $D000 开始。当最后一个字节发送到 MCU,并且检验数据返回到主机后,使 MCU 复位,结束编程操作。

编程的字节数、第一个字节地址和编程时间可由用户控制,可将省缺值改为所需值。图 6-4 是 EPROM 编程流程图。左上图为 MCU 自引导 ROM 的 PROGRAM 程序流程图,右上图为主机用户驱动程序流程图,下面是 MCU 与主机操作关系时序图。三部分图中[ ]内的编号是一致的。

阴影区域[1]是 MCU 发送数据($FF)的软件和硬件延迟。类似地,阴影[2]是主机延迟(向 MCU 发送第一个数据字符情况下)。

整个过程从 MCU 向主机发送 $FF

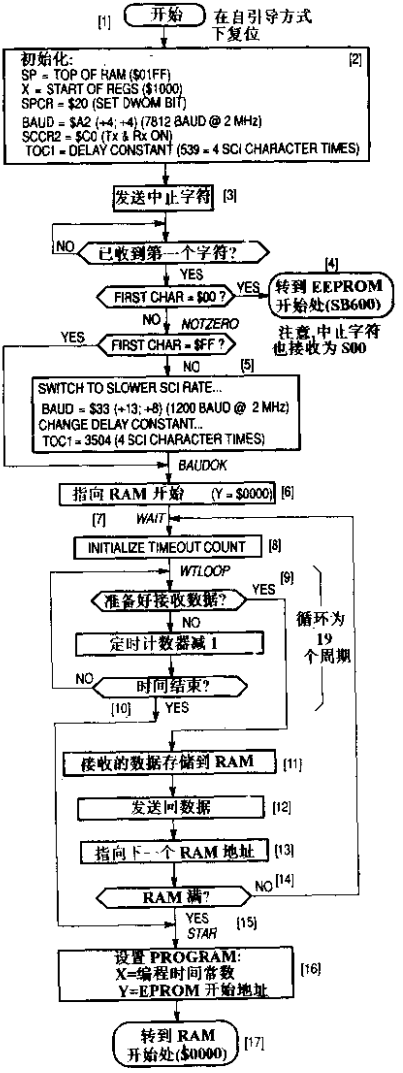


图 6-3 MC68HC11E9 的自引导装入流程图

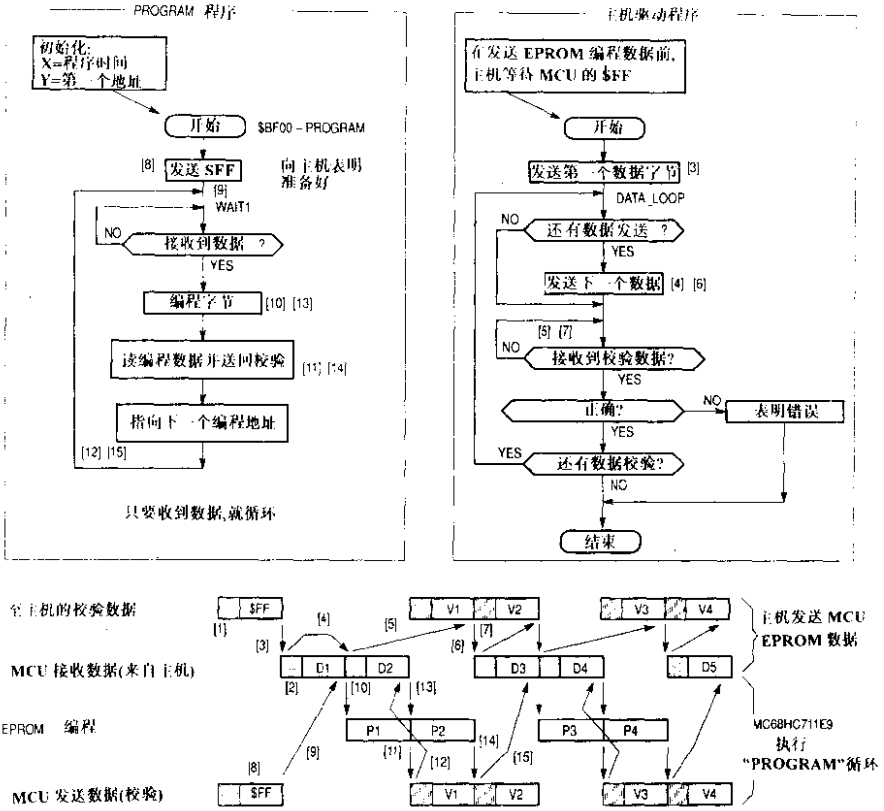


图 6-4 EPROM 编程期间主机和 MCU 的操作

开始,发送 $FF 表明收发正常。主机发送第一个数据字节[3]并进入主循环,在[4]发送第二个字节,然后主机等待从 MCU 返回的第一个验证字节。

MCU 发送 $FF 后[8],进入 WAIT1 循环[9],并等待来自主机的第一个数据字符。当接收到该字符时[10],MCU 将其编程到 X 变址器所指的地址。当编程时间结束后,MCU 读该编程数据,将其返送回主机进行校验[11],并重复 WAIT1 循环,等待第二个数据字符[12]。[13]~[15]是第二次 WAIT1 循环。

主机接收到第一个校验字符,发送第三个数据字符[6]。然后主机等待从 MCU 返回的第二个校验字符[7]。只要主机不断向 MCU 发送数据,这一过程就继续。

6.8 允许自引导模式

自引导方式使用方便,对于因改变 CONFIG 寄存器造成的故障或扩展数据/地址总线故

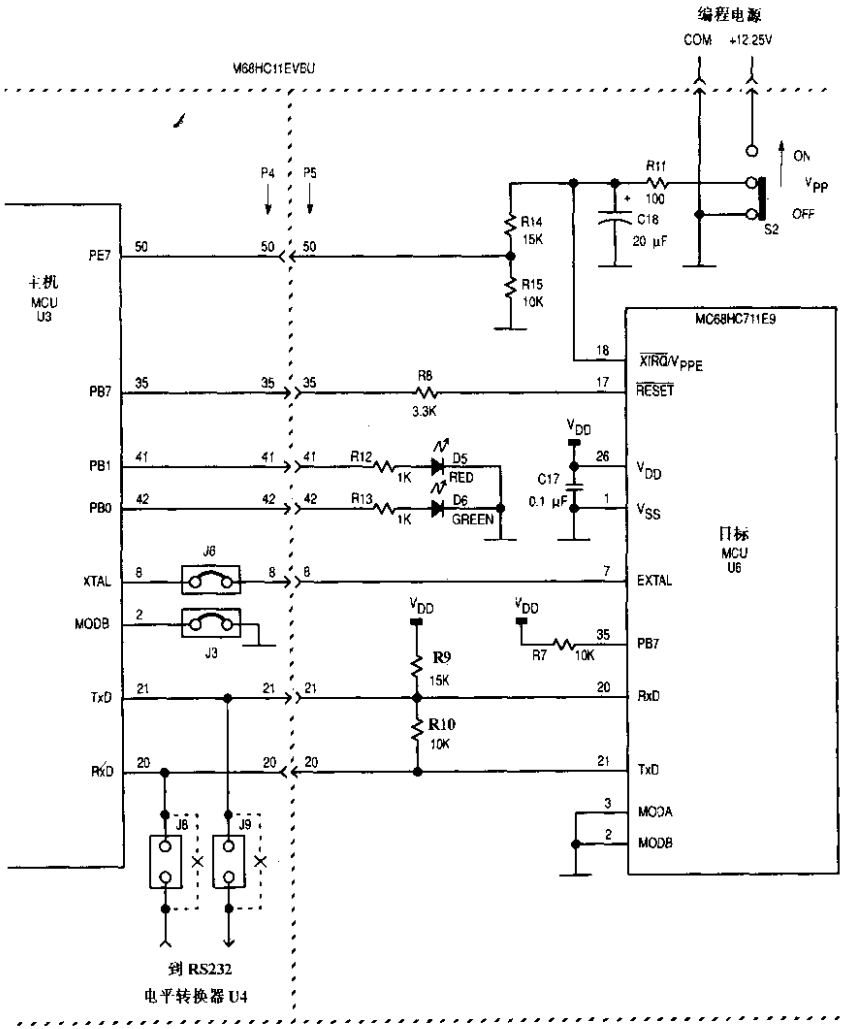


图 6-5　MCU 至 MCU EPROM 复制器结构

障，用自引导模式诊断和修复是很有用的。还可以用自引导模式编程 EPROM 和 EEPROM。

模式选择引脚：为了能强迫 MODA 和 MODB 引脚为逻辑零，这两引脚应通过上拉电阻接到 V_{DD}。类似地，接低电平时也应通过下拉电阻接地，而不能直接接 V_{SS} 或 V_{DD}。否则 MCU 只能为固定一种模式，难以改变操作模式。因为在调试期间经常需要重新复位进入另一种模式。

RXD 引脚：PD0/RXD 引脚用作来自主机的串行数据接收端。

TXD 引脚：自引导装入程序用 PD1/TXD 引脚向主机回送校验数据。为减少发生与该引脚相连的电路相冲突的可能性，在自引导程序初始化期间将 D 口设置为线或模式。这时需要外接一个上拉电阻，连到该引脚与 V_{DD}之间。

其他：自引导固化程序将 DWOM 位置为 1，使 D 口设置为线或方式。在自引导装入过程中，除 PD1/TXD 引脚外，所有 D 口引脚均设置为高阻输入端。所有作为输出的引脚应接一个上拉电阻，以免在自引导过程中为浮空状态。

6.9 用另一个 M68HC11 驱动自引导模式

可以用另一个 M68HC11 作为主机进行自引导装入过程。这种方法是用来检验或编程 EVM 中目标 M68HC11 的非挥发存储器。

图 6-5 是简单的 MC68HC711 E9 EPROM 复制器电路(与 M68HC11EVBU 相同)。为简明起见，只画出了 EVBU 的重要部分电路。

必须设置主机 MCU 为自引导模式。应将跳线 J6 相连，以使主机 MCU 的 XTAL 输出与目标 MCU 的 EXTAL 输入相连。J8 和 J9 必须切断，以使 SCI 与 RS232 电平转换器不连。J3 必须连接起来。

首先用其他方法将所需的 12K 字节程序编程到 EPROM，并将小的复制器程序编程到其 EEPROM。只将复制器程序编程到主机 M68HC11 E9 的 EEPROM，MC68HC11E9 中 ROM 的程序可拷贝到目标 MC68HC711E9 的 EPROM 中。主机 MCU 装到 EVBU 的插槽 U3 中，空白的 MC68HC711E9 插到 U6 处。

当 V_{PP}关闭时，电源加到 EVBU 系统。当电源加到 EVBU 时，复位后 MCU 为自引导模式(U3)。当 U3 复位时，其 PB7 输出强迫为零，则 U6 保持复位。以后主机 MCU(U3)释放复位信号。当 U6 处于复位时，目标 MCU(U6)的 TXD 和 RXD 引脚为高阻输入，因此，当 U3 退出复位时，不会影响主机 MCU 的 TXD 和 RXD 信号。由于目标 MCU 复位时 MODA 和 MODB 保持为零，它设置为 EPROM 仿真模式，PB7 是 EPROM 数据 I/O 引脚的输出允许信号。上拉电阻 R7 使得 D 口引脚(含 TXD 和 RXD)保持在高阻态，因此，它们不影响 MCU 退出复位时的 TXD 和 RXD 引脚。

U3 已脱离复位后，开始执行自引导固化程序。中止码从 U3 的 TXD 引脚发送，上拉电阻 R9 和 R10 使得可在 U3 的 RXD 引脚收到中止码。自引导程序跳转到主机 MCU 的 EEPROM 开始处，并开始执行复制器程序。

EEPROM 的复制器程序将 DWOM 位清零，U3 的 D 口改变为正常输出，以防止目标 MCU 变为有效时 R9 的影响。串联电阻 R9 使得 U3 的 TXD 可驱动 U3 的 RXD，并且 U6 的 TXD 可驱动 U3 的 RXD，而不会发生 TXD 输出缓冲器之间的冲突。

当目标 MCU(U6)脱离复位时，选中自引导方式，开始执行自 引导固化程序。从 U6 的 TXD 引脚发送中止码，同时，U3 的 TXD 引脚为高，R9 作为目标 MCU(U6)中 TXD 的上拉电阻。U3 接收到该中止码，主机 MCU 的 EEPROM 所运行的复制器程序得知目标 MCU 已准备好接收自引导程序。

主机 MCU 发送导引 $FF 字符以设置目标 MCU 的波特率。下一步，主机 MCU 将三指令程序传送给目标 MCU 并暂停，目标 MCU 的自引导程序将停止装入过程，并转移到装入的程序的起始处。该过程表明 MC68HC711E9 自引导程序可装入不同长度的程序。

装入到目标 MCU 的小程序将 DWOM 位清零，使 TXD 引脚改为正常 CMOS 输出，并转

移到目标 MCU 自引导 ROM 的 EPROM 编程程序。

注意，小的装入程序不必设置 SCI 或初始化 EPROM 编程过程的任何参数。在运行装入程序之前的自引导软件使 SCI 的设置与主机 MCU 相容。编程时间和目标 MCU EPROM 编程首地址也由自引导软件设置为省缺值。

在目标 MCU EPROM 编程之前，目标 MCU 的$\overline{\text{XIRQ}}$/VPPE 引脚必须加 V_{PP}电源。主机 MCU 的复制器程序监视该电压(有或无)。R14 和 R15 分压后送到 PE7 引脚，这两个电阻阻值并不要求很严格。直到主 MCU 检测到 V_{PP}并稳定 200ms 后，编程数据才能传送给目标 MCU。

当加 V_{PP}后，主机 MCU 开启红 LED，并开始向目标 MCU 传送数据。送到目标 MCU 的每个数据字节都被编程，并读编程单元，送回主机校验。若编程失败，则红和绿 LED 都关闭，中止编程操作。若整个 12K 字节都编程并校验无误，则红 LED 关闭，绿 LED 亮。编程所有 12K 字节约需要 30 秒。

MCU 至 MCU 复制器程序如下：

```
INIT        EQU $103D
SPCR        EQU $28
PORTB       EQU $04
*
RESET       EQU %10000000
RED         EQU %00000010
GREEN       EQU %00000001
PORTE       EQU $0A
SCSR        EQU $2E
* TDRE、TC、RDRF、IDLE。OR、NF、FE
TDRE        EQU %10000000
RDRF        EQU %00100000
SCDR        EQU $2F
PROGRAM     EQU $BF00       自引导 ROM 的 EPROM 程序
EPSTRT      EQU $D000       EPROM 开始地址
            ORG $B600
BEGIN       CLR INIT        寄存器为$0000～$3F
            LDAA #$04
            STAA SPCR       关闭 EVBU MCU 的 DWOM 位
            LDAA #RESET
            STAA PORTB      释放目标 MCU 复位
WT4BRK      BRCLR SCSR RDRF WT4BRK   循环，直到接收到字符
            LDAA #$FF
            STAA SCDR
            LDX #BLPROG
BLLOOP      BSR SENDJ       装入目标 MCU
            CPX #ENDBPR
            BNE BLLOOP      直到结束
* 延迟大约 4 个字符时间。
            LDX #1703       6 个周期的循环数
DLYLP       DEX             3 个周期
            BNE DLYLP       3 个周期，共 6 个周期
```

```
                LDAA SCSR                   读SCSR(RDRF 置位)
                LDAA SCDR                   读SCI 数据寄存器,以将 RDRF 清零
* 等待表明目标 MCU 准备好接收 EPROM 编程数据的字符
WT4FF           BRCLR SCSR RDRF WT4FF       等待 RDRF
                LDAA SCDR                   将 RDRF 清零
                LDX #EPSTRT                 指向 EPROM 开始
* 处理 VPP并启
WT4VPP          LDY #21053                  延时计数器(200ms)
                BCLR PORTB RED              关红 LED
DLYLP2          LDAA PORTE                  3 个周期,等待 VPP开启
                BPL WT4VPP                  3 个周期
                BSET PORTB RED              6 个周期,开启 LED
                DEY                         4 个周期
                BNE DLYLP2                  3 个周期,共 19 个周期
* VPP已稳定 200ms
                LDY #EPSTRT                 X 为 TX 指针,Y 为检验指针
                BSR SEND1                   向目标 MCU 发送第一个数据
DATALP          CPX #0                      最后 X 指向 $0000
                BEQ VERF
                BSR SEND1                   发送另一个字符
VERF            BRCLR SCSR RDRF VERF        等待 RX 准备好
                LDAA SCDR                   取字符并将 RDRF 清零
                CMPA 0,Y                    校验?
                BEQ VERFOK                  无错
                BCLR PORTB(RED+GREEN)       关闭 LEDS
                BRA DUNPRG                  编程失败
VERFOK          INY                         校验指针加 1
                BNE DATALP                  继续,直到结束
                BSET PROTB GREEN            开绿 LED
DUNPRG          BCLR PORTB(RESET+RED)       关红 LED,复位
                BRA *
* 取或发送 SCI 字符的子程序
SEND1           LDAA 0,X                    取字符
                TRDYLP BRCLR SCSR TDRE
                TRDYLP                      等待 TDRE
                STAA SCDR                   发送字符
                INX
                RTS
* 装入到目标"711E9"的程序
BLPROG          LDAA # $04
                STAA $1028
*注意,不能用直接寻址,因为目标 MCU 寄存器位于 $1000
                JMP PROGRAM                 转 EPROM 编程程序
END BPR EQU
                *
```

6.10 用PC机驱动自引导模式

该例中，PC机作为主机驱动MC68HC711E9的自引导程序。M68HC11EVBU作为目标MCU。大量的程序从PC机传输到目标MCU的EPROM。

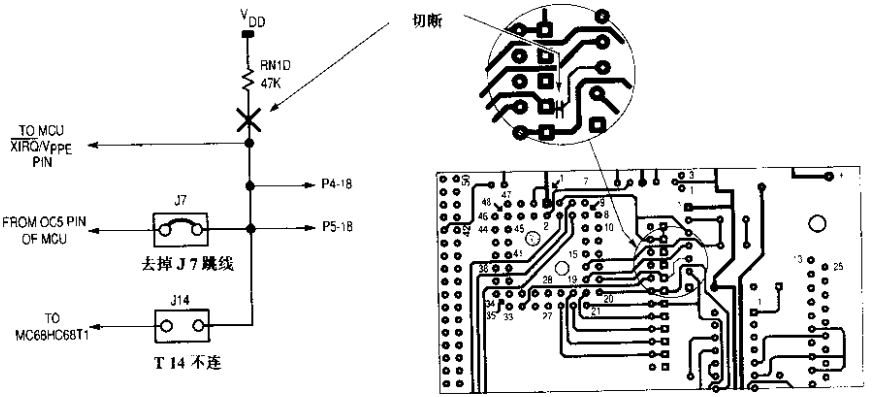


图6-6 将EVBU的$\overline{\text{XIRQ}}$引脚隔离

如图6-6所示，需作小的修改以使EVBU适于12V EPROM编程电压。$\overline{\text{XIRQ}}$引脚连至上拉电阻、两个跳线开关(Jumpers)和60引脚插头的P4与P5。修改的目的是将$\overline{\text{XIRQ}}$引脚分离，并连到编程电源上。如图6-6所示，切断与RN1D电阻相连的印制线，去掉J7和J4的跳线。

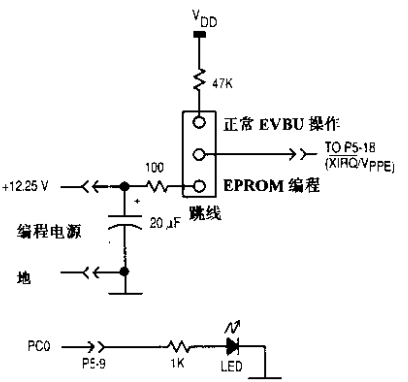


图6-7 PC至MCU编程电路

图6-7是需要加上去的电路，三端跳线可以使$\overline{\text{XIRQ}}$与编程电源或上拉电阻相连，100Ω电阻起限流作用。1k电阻和LED用来指示编程结束。

这里用BASIC作为编程语言。PC机的BASIC程序如下：

```
'*     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
'*
'* E9BUF. BAS - APROGRAM TO DEMONSTRATE THE USE OF THE BOOT MODE
'*              ON THE HC)1 BY PROGRAMMING AN MC68HC711E9 WITH
'*                   BUFFALO 3. 4
'*
'*         REQUIRES THAT THE S-RECORDS FOR BUFFALO (BUF34. S19)
'*           BE AVAILABLE IN THE SAME DIRECTORY OR FOLDER
'*
'*        THIS PROGRAM HAS BEEN RUN BOTH ON A MS-DOS COMPUTER
'*            USING QUICKBASIC 4. 5 AND ON A MACINTOSH USING
'*                 QUICKBASIC 1. 0
'*
'*     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
H$ = "0123456789ABCDEF"          'STRING TO USE FIR HEX CONVERSIONS
DEFINT B, I; CODESIZE% = 8192; ADRSTART= 57344!
BOOTCOUNT = 25                   'NUMBER OF BYTES IN BOOT CODE
DIM CODE%(CODESIZE%)             'BUFFAKO 3. 4 IS 8K BYTES LONG
BOOTCODE$ = ""                   INITIALIZE BOOTCODE$ TO NULL
REM * * * * * READ IN AND SAVE THE CODE TO BE BOOT LOADED * * * *
FOR I = 1 TO BOOTCOUNT           '# OF BYTES IN BOOT CODE
READ Q$
A$ = MID$(Q$, 1, 1)
GOSUB 7000                       'CONVERTS HEX DIGIT TO DECIMAL
TEMP = 16 * X                    'HANG ON TO UPPER DIGIT
A$ = MID$(Q$ 2, 1)
GOSUB 7000
TEMP = TEMP + X
BOOTCODE $ = BOOTCODE $ +
CHR $ (TEMP)                              'BUILD BOOT CODE
NEXT I
REM * * * * * S-RECORD CONVERSION STARTS HERE * * * * *
FILNAM$ = 'BUF34. S19'           'DEFAULT FILE NAME FOR S-ERCORDS
CLS
PRINT "Filename. ext of S-record file to be downloaded ("; FILNAM$ ;")";
INPUT Q$
IF Q$ <>"" THEN FILNAM$ =Q$
OPEN FILNAM$ FOR INPUT AS #1
PRINT : PRINT "Converting "; FILNAM$ ;"to binary···"
REM * * * * SCANS FOR 'S1'RECORDS * * * *
GOSUB 6000                       'GET 1 CHARACTER FROM INPUT FILE
IF FLAG THEN 1250                'FIAG IS EOF FLAG FROM SUBROUTINE
IF A$ <>"S" THEN 1000
GOSUB 6000
IF A$ <>"1" THEN 1000
REM * * * * * S1 RECORD FOUND.NEXT 2 HEX DIGITS ARE THE BYTE COUNT * * * *
GOSUB 60000
GOSUB 7000                       'RETURNS DECIMAL IN X
BYTECOUNT = 16 * X               'ADJUST FOR HIGH NIBBLE
GOSUB 6000
GOSUB 7000
BYTECOUNT = BYTECOUNT + X        'ADD LOW NIBBLE
```

```
1090  BYTECOUNT = BYTECOUNT - 3      'ADJUST FOR ADDRESS + CHECKSUM
1099  REM * * * NEXT 4 HEX DIGITS BECOME THE STARTING ADDRESS FOR THE DATA * * *
1100  GOSUB 6000                     'GET FIRST NIBBLE OF ADDRESS
1102  GOSUB 7000                     'CONVERT TO DECIMAL
1104  ADDRESS = 4096 * X
1106  GOSUB 6000                     'GET NEXT NIBBLE
1108  GOSUB 7000
1110  ADDRESS= ADDRESS+ 256 * X
1112  GUSOB 6000
1114  GOSUB 7000
1116  ADRESS= ADDRESS+ 16 * X
1118  GOSUB 6000
1120  GOSUB 7000
1122  ADDRESS= ADDRESS+ X
1124  ARRAYCNT =ADDRESS-ADRSTART     'INDEX INTO ARRAY
1129  REM * * * CONVERT THE DATA DIGITS TO BINARY AND SAVE IN THE ARRAY * * *
1130  FOR I = 1 TO BYTECOUNT
1140  GOSUB 6000
1150  GOSUB 7000
1160  Y = 16 * X                     'SAVE UPPER NIBBLE OF BYTE
1170  GOSUB 6000
1180  GOSUB 7000
1190  Y = Y + X                      'ADD LOWER NIBBLE
1200  CODE%(ARRAYCNT) = Y            'SAVE BYTE IN ARRAY
1210  ARRAYCNT = ARRAYCNT + 1        'INCREMENT ARRAY INDEX
1220  NEXT I
1230  GOTO 1000
1250  CLOSE 1
1499  REM * * * * DUMP BOOTLOAD CODE TO PART * * * *
1500  'OPEN "R", #2,"COM1:1200,N,8,1" 'Macintosh COM statement
1505  OPEN "COM1:1200,N,8,1,CD0,CS0,DS0,RS" FOR RANDOM AS #2 'DOS COM statement
1510  INPUT "Comm port open";Q$
1512  WHILE LOC(2) >0                'FLUSH INPUT BUFFER
1513  GOSUB 8020
1514  WEND
1515  PRINT : PRINT "Sending bootload code to target part..."
1520  A$ =CHR$(255) + BOOTCODE$ 'ADD HEX FF TO SET BAUD RATE ON TARGET HC11
1530  GOSUB 6500
1540  PRINT
1550  FOR I = 1 TO BOOTCOUNT         '# OF BYTES IN BOOT CODE BEING ECHOED
1560  GOSUB 8000
1564  K=ASC(B$):GOSUB 8500
1565  PRINT "Character #";I;"received = ";HX$
1570  NEXT I
1590  PRINT "Programming is ready to begin.": INPUT"Are you ready";Q$
1595  CLS
1597  WHILE LOC(2) > 0               'FLUSH INPUT BUFFER
1598  GOSUB 8020
1599  WEND
1600  XMT = 0:RCV = 0                'POINTERS TO XMIT AND RECEIVE BYTES
1610  A$ = CHR$(CODE%(XMT))
1620  GOSUB 6500                     'SEND FIRST BYTE
```

```
1625  FOR I = 1 TO CODESIZE%-1              'ZERO BASED ARRAY 0 -> CODESIZE-1
1630  A$ = CHR$(CODE%(I))                   'SEND SECOND BYTE TO GET ONE IN QUEUE
1635  GOSUB 6500                            'SEND IT
1640  GOSUB 8000                            'GET BYTE FOR VERIFICATION
1650  RCV = I-1
1660  LOCATE 10,1: PRINT "Verifying byte #";I;" "
1664  IF CHR$(CODE%(RCV)) = B$ THEN 1670
1665  K=CODE%(RCV):GOSUB 8500
1666  LOCATE 1,1:PRINT "Byte #";I;" "," - Sent ";HX$;
1668  K=ASC(B$): GOSUB 8500
1669  PRINT " Received"; HX$;
1670  NEXT I
1680  GOSUB 8000          'GET BYTE FOR VERIFICATION
1690  RCV = CODESIZE%-1
1700  LOCATE 10,1:PRINT "Verifying byte #";CODESIZE%;" "
1710  IF CHR$(CODE%(RCV))=B$ THEN 1720
1713  K=CODE(RCV):GOSUB 8500
1714  LOCATE 1,1:PRINT "Byte #";CODESTIZE%;" "," - Sent";HX$;
1715  K=ASC(B$):GOSUB 8500
1716  PRINT " Received";HX$
1720  LOCATE 8,1:PRINT : PRINT "Done!!!"
4900  CLOSE
4910  INPUT "Press [RETURN]TO QUIT...",Q$
5000  END
5900  '*  * * * * * * * * * * * * * * * * * * * * * * * * * * *
5910  '*  SUBROUTINE TO READ IN ONE BYTE FROM A DISK FILE
5930  '*  RETURNS BYTE IN A$
5940  '*  * * * * * * * * * * * * * * * * * * * * * * * * * * *
6000  FLAG = 0
6010  IF EOF(1) THEN FLAG = 1: RETURN
6020  A$ = INPUT$ (1,#1)
6030  RETURN
6490  '*  * * * * * * * * * * * * * * * * * * * * * * * * *
6492  '*  SUBROUTINE TO SEND THE STRING IN A$ OUT TO THE DEVICE
6494  '*  OPENED AS FILE #2.
6496  '*  * * * * * * * * * * * * * * * * * * * * * * * * *
6500  PRINT #2,A$;
6510  RETURN
6590  '*  * * * * * * * * * * * * * * * * * * * * * * * * * *
6594  '*  SUBROUTINE THAT CONVERTS THE HEX DIGIT IN A$ TO AN INTEGER
6596  '*  * * * * * * * * * * * * * * * * * * * * *
7000  X = INSTR(H$,A$)
7010  IF X = 0 THEN FLAG = 1
7020  X = X-1
7030  RETURN
7990  '*  * * * * * * * * * * * * * * * * * * * * * * *
7992  '*  SUBROUTINE TO READ IN ONE BYTE THROUGH THE COMM PORT OPENED
7994  '*  AS FILE #2. WAITS INDEFINITELY FOR THE BYTE TO BE
7996  '*  RECEIVED. SUBROUTINE WILL BE ABORTED BY ANY
7998  '*  KEYBOARD INPUT. RETURNS BYTE IN B$. USES Q$.
7999  '*  * * * * * * * * * * * * * * * * * * * * * * *
8000  WHILE LOC(2) = 0                      'WAIT FOR COMM PORT INPUT
```

```
8005  Q$ = INKEY$ : IF Q$<>""THEN 4900 'IF ANY KEY PRESSED,THEN ABORT
8010  WEND
8020  B$=INPUT$(1,#2)
8030  RETUEN
8490  '*   * * * * * * * * * * * * * * * * * * * * * * *
8491  '*   DECIMAL TO HEX CONVERSION
8492  '*   INPUT: K- INTEGER TO BE CONVERTED
8493  '*   OUTPUT: HX$ - TWO CHARACTER STRING WITH HEX CONVERSION
8494  '*   * * * * * * * * * * * * * * * * * * * * * * *
8500  IF   K > 255 THEN HX$ ="Too big":GOTO 8530
8510  HX$=MID$(H$,K16+1,1)                          'UPPER NIBBLE
8520  HX$=HX$+MIDS(H$,(K MOD 16)+1,1)               'LOWER NIBBLE
8530  RETURN
9499  '*   * * * * * * * * BOOT CODE * * * * * * * * *
9500  DATA 86,23              'LDAA # $23
9510  DATA B7,10,02           'STAA OPT2          make port C wire or
9520  DATA 86,FE              'LDAA # $FE
9530  DATA B7,10,03           'STAA PORTC         light 1 LED on port C bit 0
9540  DATA C6,FF              'LDAB # $FF
9550  DATA F7,10,07           'STAB DDRC          make port C outputs
9560  DATA CE,0F A0           'LDX #4000          2msec at 2MHz
9570  DATA 18,CE,E0,00        'LDY # $E000        Start of BUFFALO 3.4
9580  DATA 7E,BF,00           'JMP  $BF00         EPROM routine start address
9590  '*   * * * * * * * * * * * * * * * * * * * * * *
```

程序的25～45行为初始化和定义。50～95行从程序最后的DATA语句读自引导程序(源码)。该自引导码使PC0为低,X和Y寄存器初始化为EPROM编程,并跳转到该程序。从DATA读到的十六进制值由子程序转化为二进制。然后该二进制值作为数据串存储(BOOTCODE$)。

下面的97～1250行从外部磁盘文件(BUF 34.S19)读S记录,将其转化为整数并存储。这一部分说明了将ASCII S记录转化为二进制的技术。S记录翻译器中翻译S1记录,忽略其他S记录类型。

当发现S1记录时(1000～1024行),下两个字符是随后的十六进制字节数。BYTECOUNT进行减3调整,去除地址和检验和字节,只剩下目标码字节。

从1100行起,两个字节(四个字符)起始地址被转换为十进制。该地址是随后的目标码字节的起始地址。

从1130行开始的FOR-NEXT循环将目标码字节转换为十进制,并将它们存储在CODE%阵列。当前S记录的所有目标码字节转换结束后,进行下一个S1记录转换。

1500和1505行是为了适用Macintosh和PC型号的BASIC。

在打开COM口之后,将要装入的码在开始处增加$FF。数据串发送到COM口。M68HC11返回接收到的数据以进行校验。不提供自动校验,数字显示在屏幕由人工校验。

一旦MCU接收到该装入的码,自引导程序自动跳转到装入的码处,自引导程序又跳转到EPROM编程程序(位于BOOTROM)。EPROM编程请参见前面的论述。

装入其他程序需作如下改动:

①在30行,装入程序的长度必须赋予变量"CODESIZE%"。

②在30行,程序的起始地址应赋予变量"ADRSTART"。

③在 9570 行，程序的起始地址以十六进制存储在 DATA 语句的第三和第四项。

④若 9500～9580 行 DATA 语句的自引导码字节数发生任何变化，新的数值必须置于 25 行的变量"BOOTCOUNT"。

操作过程如下：将 EVBU 设置为自引导操作模式(跳线 J7 应不连)。EVBU 接至 DC 电源，EPROM 编程时开启 12V 编程电压。EVBU 的串行口与主机的串行口相连。

根据提示输入文件名，按[RETURN]省缺。程序将告知用户正在将 S 记录转换为二进制，该过程将需 30～60 秒。

提示"Comm port open"(在文件转换结束后)是确认 EVBU 都正常设置的最后一次机会。按[RETURN]则把自引导码发送到 MC68HC711E9。然后程序通知用户自引导码正发送到目标 MCU，并且返回的结果显示在屏幕上。

另外将出现提示"Programming is ready to begin. Are you?"，开启 12V 编程电压并按[RETURN]，则实际目标 EPROM 开始编程。

当进行编程时，所校验的字节数将在屏幕上不断更新。当编程结束时，将有提示显示，按[RETURN]退出。在关闭 EVBU 的 5V 电源之前，先关闭 12V 编程电压。

6.11 自引导模式常见问题

6.11.1 复位状态和自引导装入程序起始状态

不要将系统和控制位的复位状态与启动 RAM 中自引导装入程序时该系统及其控制位的状态相混淆。执行自引导装入程序改变某些系统的状态和控制位。

- SCI 系统初始化并开启(R_x 和 T_x)
- SCI 系统控制 PD0 和 PD1 引脚
- D 口输出设置为线或方式
- 堆栈指针初始化为 RAM 的顶

自引导模式是一种特殊模式，可访问受保护的控制位。自引导 ROM 出现在存储器映象。TET1 中的 DISR 位置位，禁止 COP 和时钟监视器复位。由于自引导是特殊模式，可由软件改变某些状态，甚至可以从单片模式转换为扩展模式，以访问外部存储器和外围器件。

6.11.2 将 RXD 连到 V_{SS} 不会使 SCI 接收中止码(break)

为强迫跳转到 EEPROM 起始处，自引导程序查找为 $00 的第一个字符(中止码)。接收逻辑识别到由 1 至 0 的跳变后(RXD 引脚)才开始接收字符。若 RXD 接到地，则不会发生 1 到 0 的跳变。中止字符馈至 RXD 引脚，从而跳转到 EEPROM。

6.11.3 数据装入 RAM 之前需 $FF 字符

初始字符(通常为 $FF)用来设置装入的波特率。

6.11.4 某些 M68HC11 型号装入 RAM 程序应为 256 字节

具有 256 字节 EEPROM 的 M68HC11 型号，装入的长度必须正好为 256 字节。

6.11.5 可变的装入长度

当片内 RAM 超过 256 字节时，串行装入的时间变长。可变装入长度这一特点能减小装入时间，并且不影响与早期固定装入长度型号的自引导程序的兼容性。它用最少 4 个字符时间 RXD 为空闲来表明装入结束。因此，当 PC 机向 MCU 发送装入数据时，字符应足够接近，以免检测为装入结束状态。采用 1200 波特(而不是快速波特率)有助于解决该问题。

S 记录程序在装入到 MCU 之前必须转换为二进制。从文件中读 S 记录和将其翻译成二

进制的过程可能较慢。具体时间取决于所用的计算和编程语言。将文件翻译成二进制并在装入过程开始之前存储在 RAM 中可克服这一问题，这样，读和装入数据就没有翻译或文件读取延迟。

当接收到 $FF 字符设置波特率时，装入结束机制开始有效。复位与发送 $FF(开始装入过程)之间可以为任意时间。

6.11.6 M68HC11 的 EPROM/OTP 型号具有 EPROM 仿真模式

将 MCU 设置为 EPROM 仿真模式基本与 MCU 在自引导方式下复位相同。当$\overline{\text{RESET}}$为低，并且模式选择引脚为自引导模式时，MCU 设置为 EPROM 仿真模式。用作为 EPROM 数据 I/O 线的口引脚可能是输入或输出，取决于仿真输出允许引脚($\overline{\text{OE}}$)。

6.12 各种型号的自引导 ROM

M68HC11 不同型号有不同的自引导 ROM 程序，各自特性如表 6-2 所示。MC68HC11F1、MC68HC11K4 和 MC68HC711K4 还具有额外的波特率选择。对于另外三种新的波特率，用来确定波特率的第一个字符不是 $FF，终止程序发送的延迟时间也不同，如表 6-3 所示。

表 6-2 M68HC11 各种型号自引导 ROM 特点

MCU 型号	保密	装入长度	JMP 至 BRK 或 $001	JMP 至 RAM①	省缺 RAM 地址②	EPROM PROGRAM 程序	UPLOAD 程序
MC68HC11A0	-	256	$B600	$0000	$0000-FF	-	-
MC68HC11A1	-	256	$B600	$0000	$0000-FF	-	-
MC68HC11A8	-	256	$B600	$0000	$0000-FF	-	-
MC68SEC11A8	Yes	256	$B600	$0000	$0000-FF	-	-
MC68HC11D3	-	0-192	$F000-ROM	-	$0040-FF	-	-
MC68HC711D3	-	0-192	$F000-EPROM	-	$0040-FF	Yes	Yes
MC68HC811E2	-	256	$B600	$0000	$0000-FF	-	-
MC68SEC811E2	Yes	256	$B600	$0000	$0000-FF	-	-
MC68HC11E0	-	0-512	$B600	-	$0000-1FF	-	-
MC68HC11E1	-	0-512	$B600	-	$0000-1FF	-	-
MC68HC11E9	-	0-512	$B600	-	$0000-1FF	-	-
MC68SEC11E9	Yes	0-512	$B600	-	$0000-1FF	-	-
MC68HC711E9	-	0-512	$B600	-	$0000-1FF	Yes	Yes
MC68HC11F1	-	0-1024	$FE00	-	$0000-3FF	-	-
MC68HC11K4	-	0-768	$0D80	-	$0080-37F	-	-
MC68HC711K4	-	0-768	$0D80	-	$0080-37F	Yes	Yes

注：①自引导模式复位后首先发送 SCI 字符 $00 或中止码(break)，执行 JMP，转移到该表中的地址而不是执行装入。除非特别说明，该地址为 EEPROM 起始地址。

②自引导模式复位后，如果接收的第一个字符为 $55，执行 JMP，转向片内 RAM 起始处，而不是执行装入。该 $55 字符必须以省缺波特率(7812 波特/E=2MHz)发送。

表 6-3 自引导装入程序波特率

同步字符	终止延迟	在下列 E 时钟时的波特率					
		E=2MHz	E=2.1MHz	E=3MHz	E=3.15MHz	E=4MHz	E=4.2MHz
$FF	4 个字符	7812	8192	11,718	12,288	15,624	16,838
$FF	4 个字符	1200	1260	1800	1890	2400	2520
$F0	4.9 个字符	9600	10,080	14,400	15,120	19,200	20,160
$FD	17.3 个字符	5208	5461	7812	8192	10,416	10,922
$FD	13 个字符	3906	4096	5859	6144	7812	8192

对于可变装入长度的器件，通过发送 $FF 而不发送其他 SCI 字符也有相同的结果。在 4 个 SCI 字符时间后，终止装入，执行 JMP，转向 RAM 起始处执行。

只有当 RAM 中以前已装有有用程序时，转向 RAM 这一特点才有用。

6.12.1 MC68HC711E9 自引导 ROM

```
* 具有变址偏移量 $1000 的定义
PORTD     EQU     $08
TCNT      EQU     $0E
TOC1      EQU     $16
TFLG1     EQU     $23
OC1F      EQU     $80
SPCR      EQU     $28
BAUD      EQU     $2B
SCCR2     EQU     $2D
SCSR      EQU     $2E
SCDAT     EQU     $2F
PPROG     EQU     $3B
BLAT      EQU     $20
EPGM      EQU     $01
* 存储器设置定义
EEPMSTR   EQU     $B600     EEPROM 开始
EEPMEND   EQU     $B7FF     EEPROM 结束
EPRMSTR   EQU     $D000     EPROM 开始
EPRMEND   EQU     $FFFF     EPROM 结束
RAMSTR    EQU     $0000
RAMEND    EQU     $01FF
* 延迟常数
DELAYS    EQU     3504      低速波特率延迟
DELAYF    EQU     539       快速波特率延迟
PROGDEL   EQU     4200      2ms 编程延迟
          ORG     $BF00
PROGRAM   JMP     PRGROUT   EPROM 编程程序
UPLOAD    EQU     *         UPLOAD 程序
* UPLOAD 子程序通过 SCI 从 MCU 向主机发送数据。调用 UPLOD 之前设置波特率，开启
SCI，并设置 Y 为首地址。
          LDX     #$1000          指向内部寄存器
UPLOAD    LDAA    0,Y             读字节
          BRCLR,SCSR,X,$80        * 等待 TDRE
STAA      SCDAT,X   发送
          INY
          BRA UPLOAD
* 调用 PROGRAM 这一 EPROM 编程子程序之前，要设置波特率，开启 SCI，置 X=2ms(编程
延迟)，Y 为编程首地址。结束时 SCI 允许，X=4200，Y 指向 EPROM 起始地址($D000)。
* 在 2.1MHz 时，X=4200；在 1.0MHz 时，X=2000
PRGROUT             EQU          *
```

```
                PSHX                        存延迟常数
                LDX     #$1000              指向内部寄存器
* 发送$FF表明准备好接收编程数据
                BRCLR SCSR,X    $80         * 等待 TDRE
                LDAA    #$FF
                STAA    SCDAT,X
WAIT1           EQU     *
                BRCLR SCSR,X    $20 *       等待 RDRF
                LDAB    SCDAT,X             取接收字节
                CMPB    $0,Y                已编程?
BEQ             DONEIT                      是,跳过编程周期
                LDAA    #ELAT               EPROM 置为编程模式
                STAA    PPROG,X
                STAB    0,Y                 写数据
                LDAA    #ELAT+EPGM
STAA            PPROG,X                     开启编程电压
                PULA                        存延迟时间常数
                PULB
                PSHB
                PSHA
                ADDD    TCNT,X              延迟常数+目前 TCNT
                STD     TOC1,X
                LDAA    #0C1F
STAA            TFLG1,X                     清以前的标志
                BRCLR   TFLG1,X 0C1F *      等待到达延迟时间
                CLR     PPROG,X             关闭编程电压
DONEIT          EQU     *
                BRCLR   SCSR,X  $80 *       等待 TDRE
                LDAA    $0,Y                读 EPROM
                STAA    SCDAT,X
                INY                         指向下一个地址
                BRA     WAIT1
* 主程序
BEGIN           EQU     *
                LDS     #RAMEND
                LDX     #$1000
                BSET    SPCR,X  $20         选择 D 口线或模式
                LDD     #$A20C              BAUD 在 A,SCSR2 在 B 中
                STAA    BAUD,X              SCPX 为除 4,SCRX 为除 4
                STAB    SCCR2,X             允许 TX 和 RX 延时
                LDD     #DELAYF
                STD     TOC1,X
                BSET    SCCR2,X  $01
                BRSET   PORTD,X $01 *       等待 RXD 引脚变为低
                BCLR    SCCR2,X  $01
                BRCLR   SCSR,X  $20 *       等待 RDRF
```

```
                LDAA    SCDAT,X         读数据
* 若接收到 BREAK 或 $00,数据则为 $00
                BNE     NOTZERO         若不为零跳过 JMP
                JMP     EEPMSTR         若为零则跳到 EEPROM
NOTZERO         EQU     *
                CMPA    #$FF
                EBQ     BAUDOK          若波特率正确,则接收为 $FF
* 否则变为除以 104(除以 13 和 8),1200 波特率(2MHz)
                BSET    BAUD,X $33
                LDD     #DELAYS
                STD     TOC1,X
BAUDOK          EQU     *
                LDY     #RAMSTR         指向 RAM 开始
WAIT            EQU     *
                LDD     TOC1,X          延迟常数移至 D
WTLOOP          EQU     *
                BRSET   SCSR,X $20      NEWONE 若 RDRF 置位则跳出循环
                XGDX
                DEX
                XGDX
                BNE     WTLOOP          时间未到则循环
                BRA     STAR            时间到则退出
*
NEWONE          EQU     *
                LDAA    SCDAT,X         取接收数据
                STAA    $00,Y           存储到下一个单元
                STAA    SCDAT,X         发送
                INY
                CPY     #RAMEND+1       是否满?
                BNE     WAIT
STAR            EQU     *
                LDX     #PROGDEL        初始化 X、Y
                LDY     #EPRMSTR
                JMP     RAMSTR
* 用零填充不用的字节
                BSZ     $BFD1 - *
                FCC     "A"
                FDB     $0000
                FDB     $71E9           用于确定 MCU 类型
* 向量——指向 RAM,伪向量 JMP
                FDB     $100-60         SCI
                FDB     $100-57         SPI
                FDB     $100-54         PULSE ACCUM INPUT EDGE
                FDB     $100-51         PULSE ACCUM OVERFLOW
                FDB     $100-48         TIMER OVERFLOW
                FDB     $100-45         TIMER OUTPUT COMPARE 5
```

```
        FDB     $100-42         TIMER OUTPUT COMPARE 4
        FDB     $100-39         TIMER OUTPUT COMPARE 3
        FDB     $100-36         TIMER OUTPUT COMPARE 2
        FDB     $100-33         TIMER OUTPUT COMPARE 1
        FDB     $100-30         TIMER INPUT CAPTURE 3
        FDB     $100-27         TIMER INPUT CAPTURE 2
        FDB     $100-24         TIMER INPUT CAPTURE 1
        FDB     $100-21         REAL TIME INT
        FDB     $100-18         IRQ
        FDB     $100-15         XIRQ
        FDB     $100-12         SWI
        FDB     $100-9          ILLEGAL OP-CODE
        FDB     $100-6          COP FALL
        FDB     $100-3          CLOCK MONITOR
        FDB     BEGIN           RESET
        END
```

6.12.2 MC68HC11D3 自引导 ROM

```
*寄存器定义
PORTD   EQU     $08
DDRD    EQU     $09
TCNT    EQU     $0E
TOC1    EQU     $16
TFLG1   EQU     $23
OC1F    EQU     $80
SPCR    EQU     $28
BAUD    EQU     $2B
SCCR1   EQU     $2C
SCCR2   EQU     $2D
SCSR    EQU     $2E
SCDAT   EQU     $2F
PPROG   EQU     $3B
LAT     EQU     $20
EPGM    EQU     $01
TEST1   EQU     $3E
CONFIG  EQU     $3F
*存储器设置
ROMSTR  EQU     $F000           ROM 开始与结束
ROMEND  EQU     $FFFF
RAMSTR  EQU     $0040           RAM 开始与结束
RAMEND  EQU     $00FF
*延迟常数
DELAYS  EQU     3504            慢波特率延迟
DELAYF  EQU     539             快波特率延迟
        ORG     $BF40
* 主程序
BEGIN   EQU     *
```

```
                LDS     #RAMEND         初始指针
                BSET    SPCR $20        D 口线或模式
                LDD     # $A20C        BAUD 和 SCCR2 分割在 A 和 B
                STAA    BAUD
* 向 BAUD 的 MSB 位写入 1 使计数链复位
                STAB    SCCR2           允许 RX 和 TX
                LDD     #DELAYF
                STD     TOC1
* 发送 BREAK 表明准备好装入
                BSET    SCCR2 $01
                BRSET   PORTD $01 *     等待 RXD 引脚变为低
                BCLR    SCCR2 $01
                BRCLR   SCSR $20 *      等待 RDRF
                LDAA    SCDAT           读数据
* 若接收到 BREAK 或 $00,则数据为 $00
                BNE     NOTZERO         若不是 $00,则跳转
                JMP     ROMSTR          转到 ROM(为 $00)
NOTZERO         EQU *
                CMPA    # $FF
                BEQ     BAUDOK          若波特率正确
* 否则选择除 104(13×8),1200 波特/2MHz
                BSET    BAUD $33
                LDD     #DELAYS         慢速延迟常数
                STD     TOC1
BAUDOK          EQU     *
                LDY     #RAMSTR
WAIT            EQU     *
                LDX     TOC1
WTLOOP          EQU     *
                BRSET   SCSR            $20 NEWONE 若 RDRF 置位,退出
                DEX
                NOP
                NOP
                BRN     * +2
                BNE     WTLOOP          时间未到则循环
                BRA     STAR            时间到则退出
                NEWONE  EQU *
                LDAA    SCDAT           取接收数据
                STAA    $00,Y           存储至下一个 RAM 单元
                STAA    CDAT            应答发送
                INY                     指针指向下一个 RAM 单元
                CPY     #RAMEND+1
                BNE     WAIT
STAR            EQU     *
                JMP     RAMSTR
* 用零填充未用字节块
```

```
            BSZ     $BFD1-*
            FCB     0
            FDB     $0000
            FDB     $11D3
* 向量——指向伪向量JUMP的RAM
            FDB     $100-60     SCI
            FDB     $100-57     SPI
            FDB     $100-54     PULSE ACCUM INPUT EDGE
            FDB     $100-51     PULSE ACCUM OVERFLOW
            FDB     $100-48     TIMER OVERFLOW
            FDB     $100-45     TIMER OUTPUT COMPARE 5
            FDB     $100-42     TIMER OUTPUT COMPARE 4
            FDB     $100-39     TIMER OUTPUT COMPARE 3
            FDB     $100-36     TIMER OUTPUT COMPARE 2
            FDB     $100-33     TIMER OUTPUT COMPARE 1
            FDB     $100-30     TIMER INPUT CAPTURE 3
            FDB     $100-27     TIMER INPUT CAPTURE 2
            FDB     $100-24     TIMER INPUT CAPTURE 1
            FDB     $100-21     REAL TIME INT
            FDB     $100-18     IRQ
            FDB     $100-15     XIRQ
            FDB     $100-12     SWI
            FDB     $100-9      ILLEGAL OP-CODE
            FDB     $100-6      COP FALL
            FDB     $100-3      CLOCK MONITOR
            FDB     BEGIN       RESET
            END
```

6.12.3 MC68HC711D3 自引导 ROM

```
* 寄存器定义和位定义与MC68HC11D3相同,此处略
* 存储器设置
EPRMSTR     EQU     $F000       EPROM 开始
EPRMEND     EQU     $FFFF       EPROM 结束
RAMSTR      EQU     $0040       RAM 开始
RAMEND      EQU     $00FF       RAM 结束
* 延迟常数
DELAYS      EQU     3504        低速波特率延迟
DELAYF      EQU     539         高速波特率延迟
PROGDEL     EQU     4200        2ms 编程延迟
            ORG     $BF00
PROGRAM     JMP     PRGROUT     EPROM 编程程序
UPLOAD      EQU     *           (向主机返回数据)程序
            LDAA    0,Y
            BRCLR   SCSR $80 *  等待 TDRE
            STAA    SCDAT       发送
            INY
            BRA     UPLOAD
```

```
PRGROUT     EQU     *
* 发送 $FF 表明准备好编程数据
            BRCLR   SCSR $80 *      等待 TDRE
            LDAA    #$FF
            STAA    SCDAT
WAIT1       EQU     *
            BRCLR   SCSR $20 *      等待 RDRF
            LDAB    SCDAT           取接收字节
            CMPB    $0,Y            是否已编程?
            BEQ     DONEIT          是,跳过编程周期
            LDAA    #LAT            EPROM 为编程模式
            STAA    PPROG
            STAB    0,Y             等待数据
            LDAA    #LAT+EPGM
            STAA    PPROG           开启编程电压
            PSHX                    延迟时间存入堆栈
            XGDX
            PULX
            ADDD    TCNT            延迟常数加上目前 TCNT
            STD     TOC1
            LDAA    #RC1F
            STAA    TFLG1           清任何标志
            BRCLR   TFLG1 OC1F *
            CLR     PPROG           关闭编程电压
DONEIT      EQU     *
            BRCLR   SCSR $80 *      等待 TDRE
            LDAA    $0,Y            读 EPROM
            STAA    SCDAT           发送,校验
            INY
            BRA     WAIT1
* 主程序
BEGIN       EQU     *
            LDS     #RAMEND
            BSET    SPCR $20
            LDD     #$A20C
            STAA    BAUD
* 向 BAUD 的 MSB 写入 1 使计数链复位
            STAB    SCCR2
            LDD     #DELAYF
            STD     TOC1
* 发送 BREAK 表明准备好装入
            BSET    SCCR2 $01
            BRSET   PORTD $01 *
            BCLR    SCCR2 $01
            BRCLR   SCSR $20 *
            LDAA    SCDAT
```

```
* 若接收到 BREAK 或 $00,则数据为 $00
                BNE     NOTZERO
                JMP     EPRMSTR
NOTZERO         EQU     *
                CMPA    #$FF
                BEQ     BAUDOK
                BSET    BAUD $33
                LDD     #DELAYS
                STD     TOC1
BAUDOK          EQU     *
                LDY     #RAMSTR
WAIT            EQU     *
                LDX     TOC1
WTLOOP          EQU     *
                BRSET   SCSR $20 NEWONE
                DEX
                NOP
                NOP
                BRN     * +2
                BNE     WTLOOP
                BRA     STAR
NEWONE          EQU     *
                LDAA    SCDAT
                STAA    $00,Y
                STAA    SCDAT
                INY
                CPY     #RAMEND+1
                BNE     WAIT
STAR            EQU     *
                LDX     #PROGDEL
                LDY     #EPRMSTR
                JMP     RAMSTR
* 用零填充未用字节
                BSZ     $BFD1-*
                FCC     "B"
                FDB     $0000
                FDB     $71D3
* 向量——与 MC68HC11D3 相同,此处略
```

6.12.4 MC68HC11F1 自引导 ROM

```
* 定义
PORTD           EQU     $08
DDRD            EQU     $09
TOC1            EQU     $16
SPCR            EQU     $28
BAUD            EQU     $2B
SCCR1           EQU     $2C
```

```
SCCR2       EQU     $2D
SCSR        EQU     $2E
SCDAT       EQU     $2F
PPROG       EQU     $3B
TEST1       EQU     $3E
CONFIG      EQU     $3F
*存储器定义
EEPSTR      EQU     $FE00       EEPROM 开始
EEPEND      EQU     $FFFF       EEPROM 结束
RAMSTR      EQU     $0000       RAM 开始
RAMEND      EQU     $03FF       RAM 结束
*延迟常数
DELAYS      EQU     3504        低速波特率延迟
DELAYF      EQU     539         快速波特率延迟
            ORG     $BF00
*主程序
BEGIN       EQU     *
            LDS     #RAMEND     初始化堆栈指针
            LDX     #$1000
            BSET    SPCR,X $20  选择 D 口线或模式
            LDD     #$B00C      波特率在 A,SCCR2 在 B 中
            STAA    BAUD,X      SCPx 为除 13,SCRx 为除 1
            STAB    SCCR2,X     允许 Rx 和 Tx
            LDD     #DELAYF
            STD     TOC1,X
* 发送 BREAK 表明开始装入
            BSET    SCCR2,X $01
            BRSET   PORTD,X $01 *
            BCLR    SCCR2,X $01
            BRCLR   SCSR,X $20 *    等待 RDRF
            LDAA    SCDAT,X
* 若接收到 BREAK 或$00,则数据为$00
            BNE     NOTZERO     不为$00 则跳过
            JMP     EEPSTR      若为$00 则转到 EEPROM
NOTZERO     EQU     *
* 检查除数因子为 13(2MHz 时为 9600 波特)
            CMPA    #$F0
            BEQ     BAUDOK
* 检查除数因子为 104(2MHz 时为 1200 波特)
            LDAB    #$33
            CMPA    #$80        #FF 将被接收为$80
            BEQ     SLOBAUD
* 检查除数因子为 32(2MHz 时为 3906 波特,相当于 4.2MHz 时为 8192 波特)
            LDAB    #$05
            BITA    #$20
            BEQ     SLOBAUD
```

```
* 改变为除 16(2MHz 时为 7812 波特,相当于 2.1MHz 时 8192 波特)
            LDAB    #$22
            STAB    BAUD,X
            BITA    #$08
            BNE     BAUDOK
* 改变为除 24(2MHz 时为 5208 波特,相当于 3.15MHz 时 8192 波特)
            LDAB    #$13            省缺值
SLOBAUD     EQU     *
            STAB    BAUD,X
            LDD     #DELAYS
            STD     TOC1,X
BAUDOK      EQU     *
            LDY     #RAMSTR
WAIT        EQU     *
            LDD     TOC1,X
WTLOOP      EQU     *
            BRSET   SCSR, X $20     若 RDRF 置位则退出循环
            XGDX    NEWONE          交换延迟常数
            DEX
            XGDX
            BNE     WTLOOP          来到时间则继续循环
            BRA     STAR
NEWONE      EQU     *
            LDAA    SCDAT,X         取接收数据
            STAA    $00,Y
            STAA    SCDAT,X
            INY
            CPY     #RAMEND+1
            BNE     WAIT
STAR        EQU     *
            JMP     RAMSTR
* 用零填充不用字节
            BSZ     $BFD1-*
* 向量——与 HC11D3/HC711D3 相同,此处略
```

6.12.5 MC68HC11K4 自引导 ROM

```
* 自动选择波特率:7812、1200、9600、5208、3906 波特(E=2MHz)
* 0~768 字节可变装入长度
* 定义
PORTB       EQU     $04
PORTF       EQU     $05
PORTD       EQU     $08
DDRD        EQU     $09
TCNT        EQU     $0E
TOC1        EQU     $16
TFLG1       EQU     $23
OC1F        EQU     $80
```

```
EPROG       EQU     $2B
ELAT        EQU     $20
EPGM        EQU     $01
PPROG       EQU     $3B
TEST1       EQU     $3E
CONFIG      EQU     $3F
SCBD        EQU     $70
SCCR1       EQU     $72
SCCR2       EQU     $73
SCSR1       EQU     $74
SCSR2       EQU     $75
SCDRH       EQU     $76
SCDRL       EQU     $77
EEPMSTR     EQU     $0D80       EEPROM 开始
EEPMEND     EQU     $0FFF       EEPROM 结束
ROMSTR      EQU     $2000       ROM 开始
ROMEND      EQU     $7FFF       ROM 结束
RAMSTR      EQU     $0080
RAMEND      EQU     $037F
* 延迟常数
DELAYS      EQU     5547        低速波特率延迟
DELAYF      EQU     854         高速波特率延迟
PROGDEL     EQU     4200        2ms 编程延迟
CYCLCOD     EQU     $BE40
            ORG     $BF00
* 主程序
BEGIN       EQU     *
            LDS     #RAMEND
            LDD     #$9696      若 B. F 口均为 10010110(只用于工厂测
            CPD     PORTB       试)
            BNE     CONTINU
            JMP     CYCLCOD
CONTINU     EQU     *
            LDD     #$001A
            STD     SCBD        2MHz 时为 9600 波特
            LDD     #$400C      线或模式
            STD     SCCR1
            LDD     #DELAYF
            STD     TOC1
            BSET    SCCR2 $01
            BRSET   PORTD $01 *
            BCLR    SCCR2 $01
            BRCLR   SCSR1 $20 *
            LDAA    SCDRL
            BNE     NOTZER0
            JMP     EEPMSTR
```

```
NOTZERO     EQU     *
* 检查分频因子 26(2MHz 时 9600 波特)
            CMPA    #$F0
            BEQ     BAUDOK
* 检查分频因子 208(2MHz 时 1200 波特)
            LDAB    #$D0
            CMPA    #$80            $FF 将接收为 $80
            BEQ     SLOBAUD
* 检查分频因子 64(2MHz 时 3906 波特,相当于 4.2MHz 时 8192 波特)
            LDAB    #$40
            BITA    #$20
            BEQ     SLOBAUD
* 改变为除 32(2MHz 时 7812 波特,相当于 2.1MHz 时 8192 波特)
            LDAB    #$20
            STAB    SCBD+1
            BITA    #$08
            BNE     BAUDOK
* 改为除 48(2MHz 时 5208 波特,相当于 3.15MHz 时 8192 波特)
            LDAB    #$30
SLOBAUD     EQU     *
            STAB    SCBD+1
            LDD     #DELAYS
            STD     TOC1
BAUDOK      EQU     *
            LDY     #RAMSTR
WAIT        EQU     *
            LDX     TOC1
WTLOOP      EQU     *
            BRSET   SCSR1 $20       NEWONE 若 RDRF 置位则退出循环
            DEX
            BNE     WTLOOP
            BRA     STAR
NFWONE      EQU     *
            LDAA    SCDRL
            STAA    $00,Y
            STAA    SCDRL
            INY
            CPY     #RAMEND+1
            BNE     WAIT
STAR        EQU     *
            JMP     RAMSTR
            BSZ     $BFD1-*         用零填充未用字节
* 向量——与 HC711E9、HC11D3 相同,此处略
```

6.12.6 MC68HC711K4 的自引导 ROM

```
* 寄存器、存储器定义与 MC68HC11K4 相同。
* 在 HC11K4 中,$2000 和 $7FFF 分别为 ROM 起始和结束地址。
```

而在 HC711K4 中，$2000 和 $7FFF 分别为 EPROM 起始和结束地址。

```
                ORG     $BF00
PROGRAM         JMP     PRGROUT         EPROM 编程程序
UPLOAD          EQU     *               UPLOAD 装入程序
                BSR     INIT            初始化子程序
UPLOAD          EQU     *
                LDAA    0,Y
                BRCLR   SCSR1 $80 *
                STAA    SCDRL
                INY
                BRA     UPLOAD
INIT            EQU     *
                LDAA    #$0F
                STAA    CONFIG
                XGDY
                ANDA    #$7F
                XGDY
                RTS
PRGROUT         EQU     *
                BSR     INIT
* 发送 $FF 表明准备好
                BRCLR   SCSR1 $80 *
                LDAA    #$FF
                STAA    SCDRL
WAIT1           EQU     *
                BRCLR   SCSR1 $20 *     等待 RDRF
                LDAB    SCDRL           取接收字节
                CMPB    $0,Y            是否已编程?
                BEQ     DONEIT          是,跳过编程周期
                LDAA    #ELAT           EPROM 为编程模式
                STAA    EPROG
                STAB    0,Y             写数据
                LDAA    #ELAT+EPGM
                STAA    EPROG           开启编程电压
                PSHX
                PULA
                PULB
                ADDD    TCNT            延迟常数加上当前 TCNT
                STD     TOC1
                LDAA    #0C1F
                STAA    TFLG1           清以前的任何标志
                BRCLR   TFLG1 0CIF *
                CLR     EPROG           关闭编程电压
DONEIT          EQU     *
                BRCLR   SCSR1 $80 *
                LDAA    0,Y
```

```
                STAA    SCDRL
                INY
                BRA     WAIT1
* 主程序
BEGIN           EQU     *
                LDS     #RAMEND
                LDD     #$9696          这部分程序用于工厂测试
                CPD     PORTB
                BNE     CONTINU
                JMP     CYCLCOD
CONTINU         EQU     *
                LDD     #$001A          波特率初始化
                STD     SCBD            2MHz 时 9600 波特
                LDD     #$400C
                STD     SCCR1
                LDD     #DELAYF
                STD     TOC1
* 发送 BREAK 表明准备好
                BSET    SCCR2 $01
                BRSET   PORTD $01 *     等待 RXD 引脚变为低
                BCLR    SCCR2 $01
BRCLR           SCSR1   $20 *           等待 RDRF
                LDAA    SCDRL
                BNE     NOTZERO
                JMP     EEPMSTR
NOTZERO         EQU     *
* 检查除以 26(2MHz 时为 9600 波特)
                CMPA    #$F0
                BEQ     BAUDOK
* 检查除以 208(2MHz 时 1200 波特)
                LDAB    #$D0
                CMPA    #$80
                BEQ     SLOBAUD
* 检查除以 64(2MHz 时 3906 波特,相当于 4.2MHz 时 8192 波特)
                LDAB    #$40
                BITA    #$20
                BEQ     SLOBAUD
* 改为除以 32(2MHz 时 7812 波特,相当于 2.1MHz 时 8192 波特)
                LDAB    #$20
                STAB    SCBD+1
                BITA    #$08
                BNE     BAUDOK
* 改变为除以 48(2MHz 时 5208 波特,相当于 3.15MHz 时 8194 波特)
                LDAB    #$30
SLOBAUD         EQU     *
                STAB    SCBD+1
```

```
        LDD     #DELAYS
        STD     TOC1
BAUDOK  EQU     *
        LDY     #RAMSTR
WAIT    EQU     *
        LDX     TOC1
WTLOOP  EQU     *
        BRSET   SCSR1 $20       若 RDRF 置位则退出循环
        DEX
        BNE     WTLOOP          时间未到,继续循环
        BRA     STAR
NEWONE  EQU     *
        LDAA    SCDRL
        STAA    $00,Y
        STAA    SCDRL
INY
        CPY     #RAMEND+1
        BNE     WAIT
STAR    EQU     *
        JMP     RAMSTR
        BSZ     $BFD1-*         用零填充不用字节
```

* 向量——与 HC711E9、HC11D3 相同,此处略

7. M68HC11 堆栈的使用

M68HC11 的堆栈有两个用途：①每当 CPU 执行 BSR 和 JSR 指令时，将返回地址压入堆栈，当程序从子程序返回时，CPU 恢复执行 JSR 或 BSR 后面的指令；②在 MCU 执行中断服务程序之前，CPU 将寄存器内容压栈，当 CPU 执行中断服务程序最后的 RTI 指令时，恢复寄存器的内容。这里讨论 M68HC11 的另外两种用途：存储局部或暂时变量值和子程序参数传递。

用堆栈作为局部变量和传递参数有如下优点：①由于进入时分配局部变量空间并且在退出时释放，可重复利用存储器空间，节省程序所需的 RAM 量；②易于调试程序；③将临时变量和参数放入堆栈有利于程序的模块化，因为程序所需的存储器由程序模块分配和重新分配，可容易地与主程序分开，重新使用。

本节包括下列内容：M68HC11 堆栈的基本操作、局部和全局变量的概念、子程序参数传递等。

7.1 M68HC11 堆栈操作

M68HC11 的 CPU 有一个 16 位堆栈指针(SP)寄存器。SP 可指向 64K 寻址空间的任何地址。每当压栈时，SP 自动递减(即由存储映像的高地址变为低地址)。SP 总是含有下一个可用的堆栈地址。

在子程序调用/返回和服务中断期间，CPU 硬件自动使用堆栈。当 JSR 或 BSR 指令调用子程序时，JSR 或 BSR 之后的指令地址自动压入堆栈。由于 M68HC11 只有 8 位数据总线，CPU 硬件执行两次压栈操作。在第一次压栈操作期间，返回地址的低 8 位(b7～b0)放入堆栈，第二次将返回地址的高 8 位(b15～b8)放入下一个较低地址中，即高 8 位在低地址，如图 7-1 所示。

表 7-1 有关 SP 的指令

指令	说明
PSHA/PSHB	将累加器 A/B 压入堆栈
PULA/PULB	将累加器 A/B 弹出堆栈
PSHX/PSHY	将变址寄存器 X/Y 压入堆栈
PULX/PULY	将变址寄存器 X/Y 弹出堆栈
INS	堆栈指针加 1
DES	堆栈指针减 1
TXS/TYS	将变址寄存器 X－1/Y－1 放入堆栈指针
TSX/TSY	将堆栈指针加 1 的内容放入变址器 X/Y 中

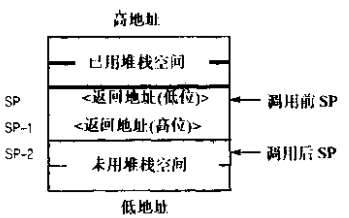


图 7-1 执行 JSR 和 BSR 后的堆栈

每当发生未屏蔽的中断时，所有 CPU 寄存器的内容(除 SP 本身外)都压入堆栈，如图 7-2 所示。压栈后，CPU 在中断程序向量指定的地址处继续运行。执行完中断服务程序后，RTI 指令恢复 CPU 寄存器，次序与压栈相反。与 SP 有关的指令如表 7-1 所示。

7.2 堆栈的使用

在高级语言中，堆栈还有另外两个功能：传递参数和存储局部或临时变量。

7.2.1 汇编语言中的变量

计算机程序很少对数据直接操作，而是通过变量。变量就是存储器映象的物理单元，可用来保持不同的值。用变量名代表数据比二进制或十六进制便于记忆。在汇编语言中通常用汇编伪指令说明变量，如下所示：

```
                ORG     $10
STANUM          RMB 1   STATION NUMBER REGISTER.
DATBLP          RMB 1   DATA TABLE POINTER REGISTER.
STAMSK          RMB 1   STATION BIT MASK REGISTER.
FCTNUM          RMB 1   FUNCTION NUMBER REGISTER FOR MODE SET.
XTEMP           RMB 2   X-REG. TEMPORARY STORAGE.
XTEMP1          RMB 2   X-REGISTER TEMPORARY STORAGE.
ATEMP1          RMB 1   A-REGISTER TEMPORARY STORAGE.
COUNT1          RMB 1   COUNT USED DURING STATION POLLING LOOP.
KPCNT           RMB 1   'NUMBER OF KEYS PRESSED' COUNT.
LSTFCN          RMB 1   LAST T/L FUNCTION THAT WAS PROCESSED.
CALLST          RMB 1   REMOTE CALL STATUS BYTE.
ATEMP2          RMB 1   A-REG. TEMPORARY STORAGE FOR THE DELAY SUBROUTINE.
XTEMP3          RMB 2   X-REG. STORAGE BEFORE CALL TO DELAY SUBROUTINE.
COUNT2          RMB 1   COUNT USED IN DELAY SUBROUTINE.
NONESL          RMB 1   'NONE SELECTED' REGISTER USED BY SSCHK
```

这是汇编语言设计者分配临时变量的典型例子，每当需要临时变量时，就定义一组全局变量。如果程序中有许多子程序需要临时存储数据，这种方法的 RAM 利用率低。

为更有效地利用 MCU 内部有限的 RAM，应采用“公用变量”。在每一行除了变量说明外，还有使用该变量的子程序，如下所示：

```
        OGR $0
* * * variables - used by: * * *
PTR0    RMB   2   main, readbuff, incbuff, AS
PTR1    RMB   2   main,BR,DU,MO,AS,EX
PTR2    RMB   2   EX,DU,MO,AS
PTR3    RMB   2   EX,HO,MO,AS
PTR4    RMB   2   EX,AS
PTR5    RMB   2   EX,AS,BOOT
PTR6    RMB   2   EX,AS,BOOT
PTR7    RMB   2   EX,AS
PTR8    RMB   2   AS
TMP1    RMB   1   main,hexbin,buffarg,termarg
```

```
TMP2    RMB   1   GO,HO,AS,LOAD
TMP3    RMB   1   AS,LOAD
TMP4    RMB   1   TR,HO,ME,AS,LOAD
```

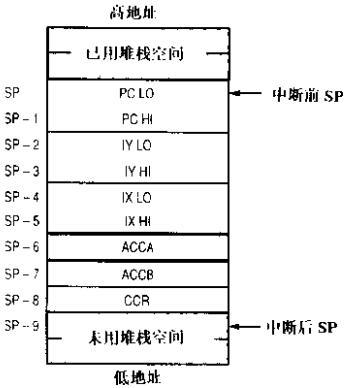


图 7-2　中断后的堆栈

这种公用变量的方法可能发生子程序的调用问题。如图 7-3 所示，子程序 A 和子程序 B 公用变量 Temp1。初看起来，子程序 A 和 B 不会相互调用，好象没问题。但是，当在执行子程序 A 时若发生中断，子程序 C 直接调用子程序 B，子程序 B 也可能用到变量 Temp1，这样会把中断前子程序 A 存在变量 Temp1 中的值破坏。这种程序很可能在大部分操作期间是正常的，但随时都可能发生上述问题。这样的问题也是很难查到的。

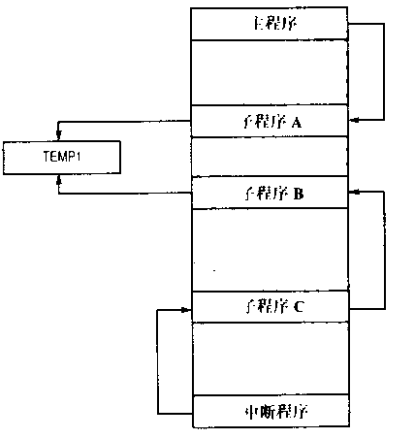


图 7-3　两个子程序公用一个变量

在含有成千上万个子程序的程序中，很难跟踪在某一时刻哪一个子程序在使用哪些变量，特别是主程序和中断服务程序公用子程序时。解决这一问题的方法很简单，即不允许任何子程序公用临时变量，但占用 RAM 较多。

7.2.2　模块化结构的高级语言

下面是 Pascal 和 C 语言定义全局变量的例子：

```
    pascal                              C
var
  x,y:integer;                          int x,y;
  j:char;                               char j;
  x:boolean;                            int z;
  num:array [1..10] of integer;         int num [9];
  Date:record                           struct Date {
    Month:integer;                          int x,y;
    Day:integer;                            int Day;
```

```
  Year:integer;                    int Year;
end; };

program (input,output);            main ()
.                                  {
.                                  .
.                                  .
end.                               }
```

这些全局变量类似于前面汇编语言，它们在整个程序执行期间都是存在的，可以由程序中的所有子程序访问。

对于局部或自动变量，当调用含有某个变量的函数时，这个变量才存在。当某个函数或过程执行结束时，局部变量也消失，所占用的存储单元可以再次利用。下面是 Pascal 和 C 语言定义局部变量的例子：

```
    pascal                           C
var
  x,y:integer;                     int x,y;
  x:boolean;                       int z;

procedure A;                       A ()
  var                              {
  i,j:integer;                     int i,j;
begin                              .
.                                  .
.                                  .
.                                  .
end;                               }
```

使用局部变量有如下好处：

第 1，可节省存储器容量。因为只有进入某个子程序时，其局部变量才占用存储单元，同一个存储单元可以由多个程序公用。

第 2，因为进入某个过程或函数时分配一组新的局部变量，使得程序能重新进入或恢复。当在实时、中断驱动环境下运行复杂程序时，中断处理程序可以调用被中断的程序。可大大简化实时环境调试过程期间的工作。

第 3，局部变量有利于模块化程序设计。其程序块可容易地与主程序分开，可方便地应用于另外的程序中。

第 4，易于调试。复杂程序可能含有很多相互影响的子程序，由于局部变量在每个子程序之间是相互隔离的，调试过程变得简单了。设计程序时不必担心无意修改了其他程序的局部变量。

但采用局部变量也是有代价的。对于 M68HC11，采用局部变量的程序比只采用全局变量的程序稍有增大并且稍有变慢。因访问局部变量的寻址方式会导致较长的执行时间。

7.2.3 传递参数

为使程序更具灵活性和改变其作用，必须将不同的信息传递给程序。通常，汇编语言使用 CPU 寄存器为子程序传递信息。

当需要传递的信息超过了 CPU 的能力时，可由一组全局变量传递信息，但也会造成难以

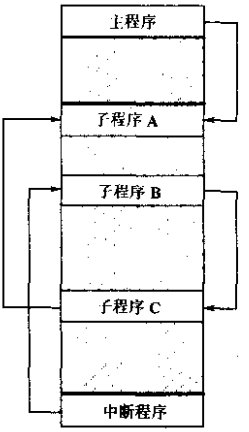


图 7-4　子程序调用

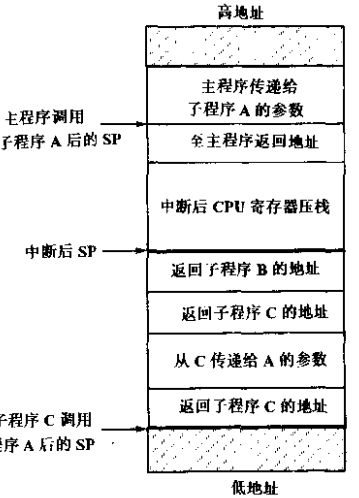


图 7-5　中断后的堆栈状态

调试的问题。如图 7-4 所示，设子程序 A 的参数由一组全局变量传递。如果主程序或子程序 C 调用子程序 A，都能正常工作。若执行子程序 A 期间发生中断，则子程序 C 调用子程序 A 时会破坏由主程序传递给子程序 A 的参数。当处理器从中断返回并恢复执行子程序 A 时，所使用的参数数据是不正确的。

上述问题极难确定。用堆栈传递参数可完全消除这种问题。当子程序 C 调用子程序 A 时（中断后），新的一组参数放在堆栈中，而原来参数保持不受影响，如图 7-5 所示。

7.3　M68HC11 堆栈的使用

本节讨论有关位于 M68HC11 堆栈的局部变量和参数的操作。M68HC11 的编程模型如图 7-6 所示。

累加器 A 和 B 用于存放操作数和算术/逻辑运算的结果，它们可以连接成 16 位累加器 D。变址寄存器 X 和 Y 用于 CPU 变址寻址方式。在变址寻址方式中，16 位变址寄存器的值加上定点 8 位无符号偏移量形成指令所用的操作数的有效地址。堆栈指针 SP 指向堆栈所用的 RAM 区。程序计数器 PC 为 16 位寄存器，用来保存所要执行的下一条指令的地址。状态码寄存器 CCR 含有表示操作结果的状态标志位和中断屏蔽位。

先介绍堆栈信息的访问方法。变址寄存器和变址寻址方式用来访问位于堆栈的局部变量或参数。对于变址寻址方式，16 位变址寄存器的内容加上定点无符号偏移量计算出指令操作数的有效地址。指令操作码之后的单字节无符号偏移量范围为 0～255。变址寻址方式有效地址的计算如图 7-7 所示。

在图 7-8 中，SP 指向下一个可用地址，Y 变址寄存器指向最后放入堆栈的数据。下面指令

```
LDD 1, Y
```

将局部变量 X 的值装入累加器 D。为访问参数“Num”，可用下列指令

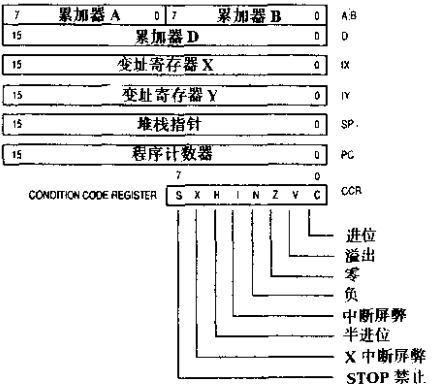


图 7-6 M68HC11 的编程模型

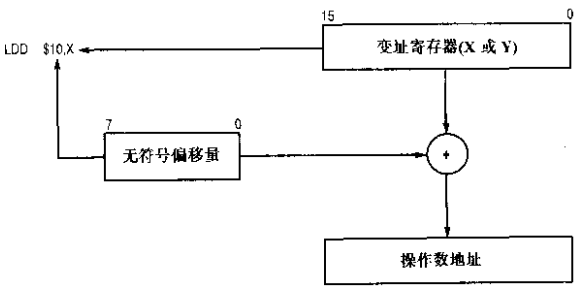


图 7-7 M68HC11 变址寻址方式有效地址的计算

LDD 7，Y

任何变址寻址方式所支持的指令都可以用来处理堆栈数据。

7.3.1 传递参数

CPU 压栈指令可将参数放入堆栈，PSHB、PSHA 执行将累加器 D 压栈，这两条指令必须按 PSHB 在前这一次序，以保持压栈值与存储在存储器中 16 位数据格式一致，即 16 位值的高位字节位于低地址。

参数传递可以是一个值或基准(reference)。例如，INT2ASC 函数，它将 16 位有符号整数转换为 ASCII 码并放入缓冲器。该函数需要两个参数：需要转换为 ASCII 的数和存储 ASCII 码的指针。前一个参数由一个值传递给子程序，第二个参数通过基准传递。C 语言函数定义如下：

```
void Int2Asc (int Num; char *Buff)
    {
```

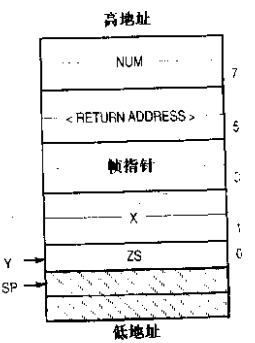


图 7-8 堆栈数据访问的例子

```
int Pwr10 = 10000;
char zs = 0
.
.
.
}
```

M68HC11 汇编语言将这两个参数压栈的程序如下所示：

```
LDX     ErrorNum        Get the value of the current error.
PSHX                    Place it on the stack.
LDX     #OutBuff        Get the address of the Output buffer.
PSHX                    Place it on the stack.
JSR     Int2Asc         Go convert the number.
```

进入子程序 Int2Asc 后堆栈传递的参数如图 7-9 所示。

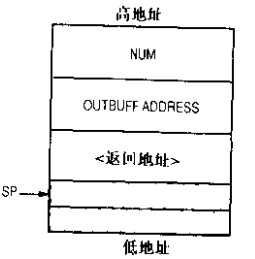


图 7-9　堆栈传递的参数位置

7.3.2　分配局部变量

有 4 种基本方法可用来确定堆栈的局部变量，具体方法取决于所需的存储单元数和是否赋予初始值。

确定堆栈中局部变量的方法之一是使用 SP 递减指令(DES)。这是简单而直接的方法，但当需要大量的存储单元时，该方法是不实际的。例如，子程序需要 100 个字节局部变量，就需要 100 个 DES 指令，因为每条 DES 指令都需要一个字节的程序存储器，显然是不能接受的。

由于定义局部单元只包括 SP 递减，可用 PSHX 指令来定义两个字节局部存储空间。X 变址寄存器的内容实际上是无关的，因为所关心的只是递减 SP。

很多情况在使用局部变量之前需要其具有特定的初值。可采用将变量的初值装入一个 CPU 寄存器并执行 PSH 指令来实现。下面是一个定义并初始化一个 8 位和 16 位局部变量的例子：

```
Int2Asc     equ     *
            :
            ldx     #10000      get the initial value of Pwr10.
            pshx                allocate and initialize it.
            clra                initial value of zs is zero.
            psha                allocate and initialize it.
```

如果子程序需要 13 个字节以上的局部存储单元，可采用下面类似的方法：

```
SINCOS      EQU     *
            :
            TSX                 SP+1→X
            XGDX                exchange the content of X and D
            SUBD    #××××       Substract the required amt. of Storage
            XGDX                place the result back into X
            TXS                 X-1→SP，更新 SP
```

因为没有一条指令能实现 SP 传输到累加器 D，必须使用两条指令(TSX 与 XGDX 或

TSY 与 XGDY)。再用 16 位减法指令调整 SP 的值,并将新值放回 SP。

7.3.3 产生完整的堆栈帧(frame)

完整的堆栈帧必须含有另外两点:返回地址和堆栈帧基址指针。当执行 JSR 或 BSR 时,M68HC11 自动将返回地址放入堆栈,如图 7-9 所示,它位于子程序参数下面。

在使用变址寄存器 X 或 Y 访问程序参数或局部变量前,必须首先保存寄存器的内容。称为堆栈帧指针的变址寄存器的内容可能含有程序堆栈帧的基地址。必须保存该指针,以便当控制返回到调用程序时,调用程序环境能恢复到以前的状态。既便程序没有参数或局部变量,作为堆栈帧指针的变址寄存器内容也必须保存。

存储以前堆栈帧指针值的最好时刻是进入子程序时立即进行,这样堆栈帧指针位于返回地址下面,如图 7-10 所示。

确定局部变量空间后,需要初始化新的子程序的堆栈帧指针。将 SP 的内容传输到 X 或 Y 变址寄存器,就可产生新的堆栈帧。

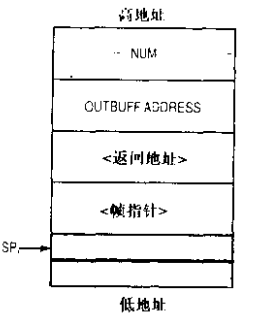


图 7-10 堆栈帧指针的位置

归纳起来,进入子程序后产生完整的堆栈帧的 3 个步骤是:

1. 进入子程序后,立即用 PSHX 或 PSHY 指令将作为堆栈帧指针的变址寄存器的内容保存起来。

2. 确定局部变量存储空间。

3. 用 TSX 或 TSY 指令初始化新的堆栈帧指针。

最后要讨论的是用哪一个变址寄存器作为堆栈帧指针。从程序码和速度考虑,应选用 X 变址寄存器,因为有关 Y 的指令都需要一个额外操作码字节和一个时钟周期。但是,如果程序不是很频繁使用堆栈,而是阵列和表格处理较多,选用 Y 变址寄存器较好。不论使用哪一个变址寄存器,都应尽可能地在整个程序使其专门用作堆栈帧指针。

7.3.4 访问参数和局部变量

使用支持变址寻址方式的指令可访问局部变量或参数。下面是支持变址寻址方式的指令:

ADCA	ADCB	ADDA	ADDB	ADDD
ANDA	ANDB	ASL	ASR	BCLR
BITA	BITB	BRCLR	BRSET	BSET
CLR	CMPA	CMPB	COM	CPD
CPX	CPY	DEC	EORA	EORB
INC	JMP	JSR	LDAA	LDAB
LDD	LDS	LDX	LDY	LSL
LSR	NEG	ORA	ORB	ROL
ROR	SBCA	SBCB	STAA	STAB
STD	STS	STX	STY	SUBA
SUBB	SUBD	TST		

图 7-11(a)是子程序所用的完整的堆栈帧。由于变址寻址方式将 8 位无符号偏移量加到 16 位变址寄存器,故变址寻址方式只能访问变址寄存器所含基地址以上 0～255 字节地址的

信息。当用堆栈传递参数时，不仅参数占用的空间不能用作局部变量，并且子程序返回地址和以前堆栈帧指针也使可用的空间减少 4 个字节。

7.3.5 堆栈帧的重新定位

执行完子程序时，必须将堆栈帧的堆栈空间释放，以便重新利用。堆栈帧的重新定位不仅要去除局部变量占用的空间，还要释放以前堆栈帧指针和为子程序传递参数所占的空间。

向上调整 SP 的值可释放堆栈帧占用的存储器。最简单的方法是分配堆栈帧的相反过程。这种去掉堆栈帧的方法分三步：①从堆栈区去除局部变量占用的存储器，可用下列方法：

```
LDAB    #LOCLEN     Get size of local storage into the B
ABX                 Add it to the current stack frame pointer
TXS                 Deallocate the local storage
```

②恢复以前堆栈帧指针。由于该堆栈帧指针位于堆栈顶，该操作只需执行 PULX 或 PULY。这时返回地址位于堆栈顶。执行 RTS 指令则使机器执行调用子程序之后的指令。③必须去掉调用子程序之前压入堆栈的任何参数。去掉子程序参数由调用程序完成，而不是由被调用程序完成。这种方法最容易接受，也是 C 语言中最常用的。堆栈的状态如图 7-11 所示。

另一种方法需要被调用程序去除整个堆栈帧，包括传递的参数。这种方法不是程序效率最高的方法，因为它需要固定数目的指令来释放占用的空间。当 X 作为堆栈帧指针时，堆栈帧重新定位程序如下：

```
LDY     RA,X        Load the return address into the Y register.
LDX     PSFP,X      Restore the previous stack frame pointer.
TXS                 Remove the entire stack frame.
JMP     0,Y         Return to the calling routine.
```

该过程需要 9 字节存储空间和 18 个周期的执行时间。这种方法的缺点是 X 或 Y 必须总是含有堆栈帧指针，既使不需局部变量或参数，也必须在进入执行 PSHX；TSX 或 PSHY；TSY 指令时指明堆栈的目前状态。上述程序中，RA 是＜返回地址＞的偏移量，PSFP 是＜以前堆栈帧指针＞的偏移量。

7.3.6 宏

下列宏有助于管理 M68HC11 程序的堆栈帧。这些宏不一定具有最小或最快的程序码，但可使得程序容易编写和调试。在宏体内，第一个参数用“；0”标明，第二个参数用“；1”注明，等等。

宏连接如下：

```
link    macro
        psh;0           Save the previous stack frame pointer.
        ts;0            Transfer the stack pointer into ;0
        xgd;0           Transfer ;0 into D.
        subd    #;1     subtract the required amount of local storage.
        xgd;0           Initialize the new stack frame pointer.
        t;0s            Update the stack pointer with new value.
        endm
```

它用来在进入子程序后确定堆栈帧，其作用为：①保存以前堆栈帧指针；②确定局部变量所需的字节数；③初始化新的堆栈帧指针。宏连接调用格式为：

```
link  <s.f.reg>, <storage bytes>
```

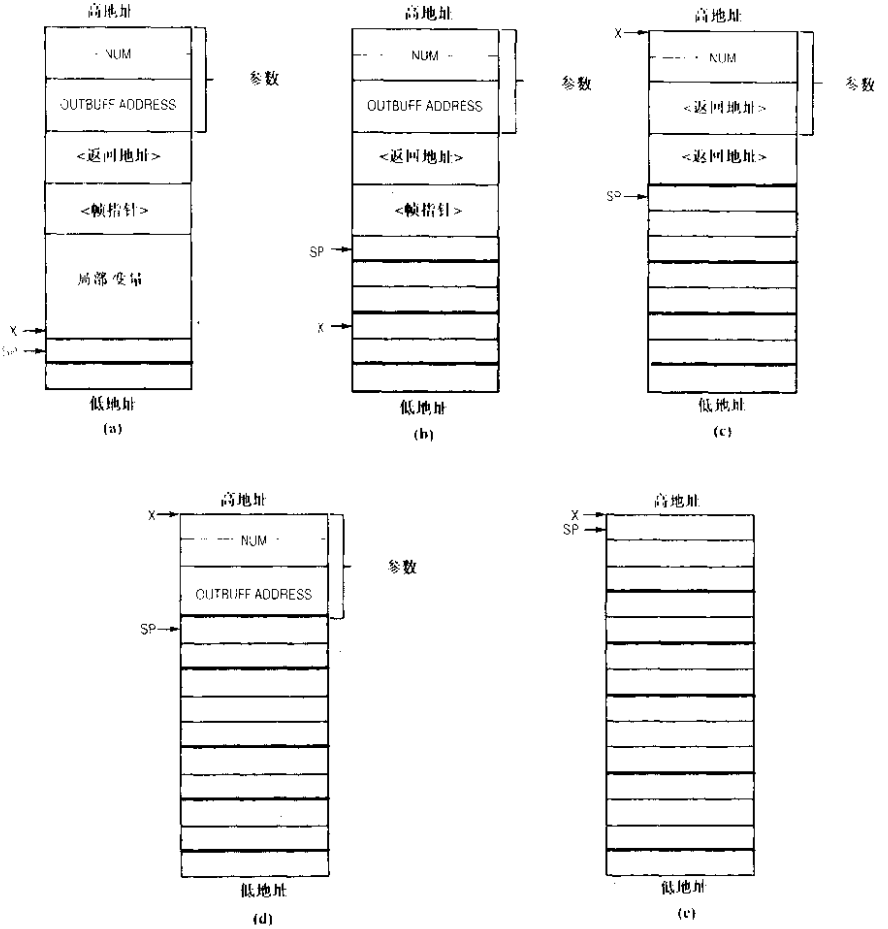


图 7-11　堆栈帧的状态

(a)重新定位过程前;(b)去掉局部变量后;(c)恢复堆栈帧指针后;
(d)执行 RTS 后;(e)结束后

传递到宏的第一个参数是作为堆栈指针(X 或 Y)的变址寄存器名;第二个参数是子程序所需的局部存储单元字节数。

返回和重新定位宏用(rtd)来重新分配子程序堆栈帧。rtd 功能如下:①重新分配存储器;②恢复以前堆栈指针;③返到调用程序。rtd 宏不去除从堆栈传递到子程序的任何参数。必须由调用程序去掉参数。当没有参数传递到子程序或由寄存器传递参数时,该宏很有用。

rtd 宏调用格式为：

rtd <s.f.reg>, <storage bytes>

其中的两个参数与 link 宏中的两个参数相同。rtd 宏如下所示：

```
rtd     macro
        ldab    #:1     number of bytes to deallocate.
        ab:0            add it to the current stack frame pointer.
        t:0s            deallocate storage by updating the stack pointer.
        pul:0           restore the previous stack frame pointer.
        rts             return to the calling routine.
        endm
```

该宏的唯一缺点是它使用累加器 B，不能将 16 位值返回到 D。解决方法是采用 LDAB 和将 B 加到 X 或 Y(ABX 或 ABY)与 PSHB/PULB，如下 frtd 所示：

```
frtd    macro
        pshb                    save the lower byte of the return value.
        ldab    #:1             number of bytes to deallocate.
        ab:0                    add it to the current stack frame pointer.
        pulb                    restore the lower byte of the return value.
        t:0s                    deallocate storage by updating the stack pointer.
        pul:0                   restore previous stack frame pointer
        rts                     return to calling routine.
        endm
```

第二个方法将所有返回值放入堆栈，调用程序重新获得返回值。

可用来完整地重新确定子程序堆栈帧(含传递的参数)的宏 rtdx/rtdy 如下所示：

```
rtdx    macro
        ldy     :0-2,x  Load the return address into the Y index register.
        ldx     :0,x    restore the previous stack frame pointer.
        txs             Update the stack pointer, removing the storage space.
        jmp     0,y     Return to the calling routine.
        endm

rtdy    macro
        ldx     :0+2,y  Load the return address into the X index register.
        ldy     :0,y    restore the previous stack frame pointer.
        tys             Update the stack pointer, removing the storage space.
        jmp     0,x     Return to the calling routine.
        endm
```

rtdx/rtdy 功能为：①确定堆栈帧，包括局部变量和传递的参数；②恢复以前的堆栈帧指针；③返回调用程序。rtdx 和 rtdy 的调用格式为：

rtdx <storage bytes>

或

rtdy <storage bytes>

该宏的优点是不使用累加器 A 和 B，允许在执行 rtdx/rtdy 宏之前将返回值装入 A、B 或 D。

采用 rtdx/rtdy 的唯一限制是，开始执行子程序将寄存器压栈时，以前子程序的有效堆栈帧指针必须存在于 X 或 Y。即使没有局部变量和没有参数传递，进入子程序也必须立即执行

PSHX 和 TSX，以保存以前的堆栈帧指针和表明目前堆栈状态。返回前必须执行 PULX。

7.3.7 举例

下面是处理局部变量、传递参数、分配/重新分配堆栈帧的几个程序例子。

```
* * *
link    macro
        psh:0           Save the previous stack frame pointer.
        ts:0            Transfer the stack pointer into :0.
        xgd:0           Transfer :0 into D.
        subd    #:1     subtract the required amount of local storage.
        xgd:0           Initialize the new stack frame pointer
        t:0s            Update the stack pointer with new value
        endm

* * *
rtd     macro
        ldab    #:1
        ab:0
        t:0s
        pul:0
        rts
        endm
* * *
frtd    macro
        pshb
        ldab    #:1
        ab:0
        pulb
        t:0s
        pul:0
        rts
        endm
* * *
rtdx    macro
        ldy     :0+2,x
        ldx     :0,x
        txs
        jmp     0,y
        endm
rtdy    macro
        ldx     :0+2,y
        ldy     :0,y
        tys
        jmp     0,x
        endm
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*        The pshd macro pushes the 16-bit d-accumulator onto the stack.
*        The b-accumulator is pushed first so that the least significant 8-bits of the 16-bit number appear
*        on the stack at the higher address. This is consistant with the way all 16-bit numbers are stored
*        in memory.
*        No parameters are required by the macro.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
pshd     macro
         pshb
         psha
         endm
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*        The puld macro pulls the top two bytes from the stack and places them in the 16-bit d-accumula-
*        tor. The first byte pulled from the stack is placed in the a-accumulator; the second byte pulled
*        from the stack is placed in the b-accumulator. The pull order is consistant with the way all 16-bit
*        numbers are stored in memory.
*        No parameters are required by the macro.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
puld     macro
         pula
         pulb
         endm
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*        The clrd macro uses the clra and clrb instructions to clear the 16-bit d-accumulator.
*        No parameters are required by the macro.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
clrd     macro
         clra
         clrb
         endm
```

```
*******************************************************************************
*
*    This subroutine converts a 16-bit binary integer to a null terminated
*    ASCII string. Three parameters are passed to the subroutine on the
*    stack. The first parameter is the 16-bit binary number to be converted.
*    The second parameter is the address of a buffer where the null terminated
*    ASCII string will be placed. The buffer should be at least 7 bytes long.
*    The third parameter is a boolean flag indicating whether the number passed
*    in the first parameter is a signed or unsigned 16-bit number. If the byte
*    flag is zero, the number is converted as an unsigned number. If the byte
*    is non zero, the number will be converted as a 16-bit signed number.
*    Parameters are pushed onto the stack in the following order: 1) Signed Flag;
*    2) Pointer to ASCII buffer; 3) Number to be converted. A typical
*    calling sequence would be:
*
*    clra                    ; Do the conversion as an unsigned number.
*    psha                    ; put the flag on the stack.
*    ldd    #Buffer          ; get the address of the ascii buffer.
*    pshd                    ; put the address on the stack.
*    ldd    Num              ; Get the number to convert.
*    pshd                    ; Put it on the stack
*    jsr    Int2Asc          ; Go convert the number.
*    .
*    .
*    .
*
*    This subroutine has two local variables. The first, zs, is a boolean variable
*    used to supress leading zeros when doing a conversion. It is located at an
*    offset of 0 from the stack frame pointer. The second local, Divisor, is a 16-bit
*    variable. It is used to divide the number being converted by succeedingly lower
*    powers of 10. Divisor is located at an offset of 1 from the local stack frame
*    pointer.
*
*    NOTE: This routine was written assuming that the previous stack frame pointer
*    is the x-index register. HOWEVER, because the x-index register is required
*    by the integer divide instruction, the y-index register is used as the
*    stack frame pointer WITHIN the Int2Asc subroutine.
*
*
*    Declare locals
*
PCSave          set     *               ; save the current PC value
                org     0               ; set PC to 0 for offsets to locals
zs              rmb     1               ; declare zs variable.
Divisor         rmb     2               ; declare Divisor variable.
LocSize         set     *               ; number of bytes of local storage.
                org     PCSave
*
*   Offsets to parameters
*
Num       equ   LocSize+4               ; offset to Num parameter.
BuffP           equ     LocSize+6       ; offset to BuffP parameter.
Signed          equ     LocSize+8       ; offset to Signed parameter.
*
Int2Asc         equ     *
[ 4]            pshx                    ; save the previous stack frame pointer.
[ 3]            ldd     #10000          ; initialize the divisor to 10000.
                pshd
[ 3]            pshb
[ 3]            psha
[ 2]            clra                    ; initialize zs to 0.
```

```
[ 3]            psha
[ 4]            tsy                     ; initialize the new stack frame pointer.
[ 6]            ldd     num,y           ; get the number to convert. Is it zero?
[ 3]            bne     Int2Asc1        ; no go do the conversion.
[ 3]            ldd     #$3000          ; yes.
[ 6]            ldx     BuffP,y         ; point to the buffer.
[ 6]            std     0,y             ; just put an ASCII 0 in the buffer.
[ 3]            bra     Int2Asc5        ; then return.
[ 7] Int2Asc1   tst     Signed,y        ; do the conversion as a signed number?
[ 3]            beq     Int2Asc2        ; no.
[ 2]            tsta                    ; yes. Is the number negative?
[ 3]            bpl     Int2Asc2        ; no. just go do the conversion.
[ 2]            coma                    ; yes. make it a positive number by negation.
[ 2]            comb
[ 4]            addd    #$1
[ 6]            std     Num,y           ; save the result.
[ 2]            ldaa    #'-'            ; get an ASCII minus sign.
[ 6]            ldx     BuffP,y         ; point to the buffer.
[ 4]            staa    0,x             ; put it in the buffer.
[ 3]            inx                     ; point to the next location in the buffer.
[ 6]            stx     BuffP,y         ; save the new pointer value.
[ 6] Int2Asc2   ldd     Num,y           ; get the remainder to convert.
[ 6]            ldx     Divisor,y
[41]            idiv
[ 6]            std     Num,y           ; save the remainder.
[ 3]            xgdx                    ; put the dividend into d.
[ 2]            tstb                    ; was the dividend 0?
[ 3]            bne     Int2Asc3        ; no. go store the number in the buffer.
[ 7]            tst     zs,y            ; are we still supressing leading zeros?
[ 3]            beq     Int2Asc4        ; yes. go setup for the next divide.
[ 2] Int2Asc3   addb    #'0'            ; make the dividend ASCII.
[ 2]            ldaa    #1
[ 5]            staa    zs,y            ; don't supress leading zeros anymore.
[ 6]            ldx     BuffP,y         ; get a pointer to the buffer.
[ 4]            stab    0,x             ; save the digit.
[ 3]            inx                     ; point to the next location.
[ 6]            stx     BuffP,y         ; save the new pointer value.
[ 6] Int2Asc4   ldd     Divisor,y       ; get the previous divisor.
[ 3]            ldx     #10
[41]            idiv                    ; divide it by 10.
[ 6]            stx     Divisor,y       ; save the dividend. Is it zero?
[ 3]            bne     Int2Asc2        ; no. continue with the conversion.
[ 6]            ldx     BuffP,y         ; get a pointer to the buffer.
[ 6]            clr     0,x             ; null terminate the string.
[ 3]            tsx                     ; this is only needed because we are using y as our
                                          sf pointer.
    Int2Asc5    rtdx    LocSize         ; return & deallocate locals & parameters.
[ 6]            ldy     LocSize+2,x     ; Load the return address into the y-index register.
[ 5]            ldx     LocSize,x       ; restore the previous stack frame pointer.
[ 3]            txs                     ; Update the stack pointer, removing the storage
                                          space.
[ 4]            jmp     0,y             ; Return to the calling routine.
*
*
*
*******************************************************************************
*
*    This subroutine performs a 16 x 16 bit unsigned multiply and produces a 32-bit
*    result. Two 16-bit numbers are passed to the subroutine on the stack.
*    The 32-bit result is returned on the stack in place of the two 16-bit
*    parameters. This allows the calling routine to easily pull the product
*    from the stack and store the result. Because multiplication is a
*    commutative operation  the order in which the parameters are pushed
*    onto the stack is unimportant. A typical calling sequence would be:
*
```

```
*     ldd       Num1
*     psnd
*     ldd       Num2
*     pshd
*     jsr       Mul16x16
*     puld
*     std       Result32
*     puld
*     std       Result32+2
*     .
*     .
*     .
*
*     This subroutine has four local variables. Each variable occupies 1 byte
*     on the stack. These four bytes are used to hold the partial product as
*     the final answer is being computed. These four byte variables are
*     treated as 16-bit variables during the calculation.
*
*     NOTE: This routine was written assuming that the stack frame pointer
*     is the x-index register.
*
*     Declare locals
*
PCSave          set     *               ; save the current PC value
                org     0               ; set PC to 0 for offsets to locals
Prd0            rmb     1               ; declare ms byte of partial product variable.
Prd1            rmb     1               ; declare next ms byte of partial product variable.
Prd2            rmb     1               ; declare next ls byte of partial product variable.
Prd3            rmb     1               ; declare ls byte of partial product variable.
LocSize         set     *               ; number of bytes of local storage.
                org     PCSave
*
*     Offsets to parameters
*
Fact1           equ     LocSize+4       ; offset to factor 1 parameter.
Fact2           equ     LocSize+6       ; offset to factor 2 parameter.
*
      cycles clear
*
Mul16x16        equ     *
[ 4]            pshx                    ; save the previous stack frame pointer.
                clrd                    ; clear the d-accumulator.
[ 2]            clra
[ 2]            clrb
                pshd                    ; allocate & initialize the locals prd0 - prd3
[ 3]            pshb
[ 3]            psha
                pshd
[ 3]            pshb
[ 3]            psha
[ 3]            tsx                     ; initialize the new stack frame pointer.
[ 4]            ldaa    Fact1+1,x       ; get the ls byte of factor 1.
[ 4]            ldab    Fact2+1,x       ; get the ls byte of factor 2.
[10]            mul                     ; multiply them.
[ 5]            std     Prd2,x          ; save the first term of the partial product.
[ 4]            ldaa    Fact1,x         ; get the ms byte of factor 1.
[ 4]            ldab    Fact2+1,x       ; get the ls byte of factor 2.
[10]            mul                     ; multiply them.
[ 6]            addd    Prd1,x          ; add the result into the partial product.
[ 5]            std     Prd1,x          ; save the result.
[ 4]            ldaa    Fact1+1,x       ; get the ls byte of factor 1.
[ 4]            ldab    Fact2,x         ; get the ms byte of factor 2.
[10]            mul                     ; multiply them.
[ 6]            addd    Prd1,x          ; add the result into the partial product.
[ 5]            std     Prd1,x          ; save the result.
[ 3]            bcc     Mul16           ; Was there a carry into Prd0?
[ 6]            inc     Prd0,x          ; yes. 'add' it in.
[ 4] Mul16      ldaa    Fact1,x         ; get the ms byte of factor 1.
[ 4]            ldab    Fact2,x         ; get the ms byte of factor 2.
```

```
[10]            mul                     ; multiply them.
[ 6]            addd    Prd0,x          ; add it to the partial product.
[ 5]            std     Fact1,x         ; overwrite the two parameters with the result.
[ 5]            ldd     Prd2,x
[ 5]            std     Fact2,x
                rtd     x,LocSize       ; return and deallocate the locals.
[ 2]            ldab    #LocSize        ; number of bytes to deallocate.
[ 3]            abx                     ; add it to the current stack frame pointer.
[ 3]            txs                     ; deallocate storage by updating the stack pointer.
[ 5]            pulx                    ; restore the previous stack frame pointer.
[ 5]            rts                     ; return to the calling routine.
    cycles total=170                    ; Total number of E cycles for a 16 x 16 multiply.
*
*
*******************************************************************************
*
*   This subroutine performs a 32 by 16 bit unsigned divide and produces a 32-bit
*   quotient and a 16-bit remainder. Both the divisor and dividend are passed to
*   the subroutine on the stack. The 32-bit quotient and 16-bit remainder are
*   returned on the stack in place of the divisor and dividend. This allows the
*   calling routine to easily pull the answer from the stack and store the result.
*   The divisor is pushed onto the stack first, followed by the lower 16-bits of
*   the dividend and finally the upper 16-bits of the dividend. A typical calling
*   sequence would be:
*
*   ldd       Divisor
*   pshd
*   ldd       Dividend+2
*   pshd
*   ldd       Dividend
*   pshd
*   jsr       Div32x16
*   puld
*   std       Quotient
*   puld
*   std       Quotient+2
*   puld
*   std       Remainder
*   .
*   .
*   .
*
*
*   This subroutine has two local variables. A 32-bit variable for partial quotient
*   results that is treated as two 16-bit variables and a 16-bit variable for
*   intermediate remainder results.
*
*   NOTE: This routine was written assuming that the previous stack frame pointer
*   is the x-index register. HOWEVER, because the x-index register is required
*   by the integer and fractional divide instructions, the y-index register is
*   used as the stack frame pointer WITHIN the Div32x16 subroutine.
*
*   Declare locals
*
PCSave          set     *               ; save the current PC value.
                org     0               ; set PC to 0 for offsets to locals.
Quo0            rmb     2               ; declare upper 16-bits of quotient.
Quo2            rmb     2               ; declare lower 16-bits of quotient.
Rem             rmb     2               ; declare remainder.
LocSize         set     *               ; number of bytes of local storage.
                org     PCSave
*
*   Offsets to parameters

Num0            equ     LocSize+4       ; upper 16-bits of Dividend.
Num2            equ     LocSize+6       ; lower 16-bits of Dividend.
Denm            equ     LocSize+8       ; 16-bit divisor.
*
    cycles  clear
*
```

```
Div32x16        equ     *
[ 4]            pshx                    ; save the previous stack frame pointer.
                clrd                    ; clear the d-accumulator.
[ 2]            clra
[ 2]            clrb
                pshd                    ; allocate & initialize the locals.
[ 3]            pshb
[ 3]            psha
                pshd
[ 3]            pshb
[ 3]            psha
                pshd
[ 3]            pshb
[ 3]            psha
[ 4]            tsy                     ; initialize y as the new stack frame pointer.
[ 6]            ldd     Num0,y          ; load the upper 16-bits of the dividend.
[ 7]            cpd     Denm,y          ; is the divisor>the upper 16-bits of the dividend?
[ 3]            blo     Div32x16a       ; yes. use a fractional divide on the initial value.
[ 6]            ldx     Denm,y          ; load the divisor into x.
[41]            idiv                    ; divide the upper 16 bits by the divisor.
[ 6]            stx     Quo0,y          ; save the partial quotient.
[ 6] Div32x16a  ldx     Denm,y          ; load the divisor into x.
[41]            fdiv                    ; resolve the remainder into a 16-bit fractional
                                          part.
[ 6]            stx     Quo2,y          ; save the partial result.
[ 6]            std     Rem,y           ; save the remainder of the fractional divide (par-
                                          tial remainder).
[ 6]            ldd     Num2,y          ; get the lower 16-bits of the dividend.
[ 6]            ldx     Denm,y          ; get the denominator again.
[41]            idiv                    ; resolve the remaining quotient.
[ 7]            addd    Rem,y           ; add the previous remainder to this remainder.
[ 6]            std     Rem,y           ; save the total remainder.
[ 3]            xgdx                    ; put the last partial quotient into the d-accumula-
                                          tor...
                                        ; & save the total remainder in x.
[ 7]            addd    Quo2,y          ; add partial quoient to the lower 16-bits of the
                                          quotient.
[ 6]            std     Num2,y          ; save the result.
[ 6]            ldd     Quo0,y          ; get the upper 16-bits of the quotient.
[ 2]            adcb    #0              ; add the possible carry to the lower 8-bits.
[ 2]            adca    #0              ; add the possible carry to the upper 8-bits.
[ 6]            std     Num0,y          ; save the result.
[ 3]            xgdx                    ; get the total remainder back into d.
[ 7]            cmpd    Denm,y          ; is the total fractional remainder > the divisor?
[ 3]            blo     Div32x16b       ; no. we're finished.
[ 7]            subd    Denm,y          ; yes. It will be < than 2*Divisor.
[ 6]            std     Rem,y           ; save the final remainder.
[ 6]            ldd     Num2,y          ; now we must add 1 to the 32-bit quotient.
[ 4]            addd    #1              ; add 1 to the lower 16-bits.
[ 6]            std     Num2,y          ; save the result.
[ 6]            ldd     Num0,y          ; get the upper 16-bits.
[ 2]            adcb    #0              ; add the possible carry to the lower 8-bits.
[ 2]            adca    #0              ; add the possible carry to the upper 8-bits.
[ 6]            std     Num0,y          ; save the result.
[ 6] Div32x16b  ldd     Rem,y           ; get the final remainder.
[ 6]            std     Denm,y          ; overwrite the divisor.
[ 3]            tsx                     ; need to do this for rtd to work correctly. See
                                          NOTE.
                rtd     x,LocSize       ; deallocate locals & return.
[ 2]            ldab    #LocSize        ; number of bytes to deallocate.
[ 3]            abx                     ; add it to the current stack frame pointer.
[ 3]            txs                     ; deallocate storage by updating the stack pointer.
[ 5]            pulx                    ; restore the previous stack frame pointer.
[ 5]            rts                     ; return to the calling routine.
*
     cycles total=347                   ; Total number of E cycles for a 32 x 16 divide.
*
```

8. MC68HC68T1 实时时钟的应用

HCMOS 外围芯片 MC68HC68T1 含有实时时钟/日历和 32 字节静态 RAM。它具有同步、串行 3 线接口(SPI),与 MCU 通讯方便,可以用晶体或 50/60Hz 电源作为时基。

MC68HC68T1 通常用于下列场合:正常工作时由电源供电作为时基,但掉电时仍要求时钟继续运行。正常操作时由电力线电源频率(50/60Hz)提供时基,掉电后,改由晶体提供正确的时基。由 50/60Hz 改为晶体的转换过程不能由 MC68HC68T1 完成,而由主 MCU 实现。

以交流电作为时基是因为它具有较高的精度,32.768kHz 晶体易受温度的影响,还受其工作电压的影响。温度系数通常在 0.035ppm/℃,老化通常超过 5ppm/年。照这样的灵敏度系数计算,其时钟误差可达 20 秒/月(室外工作环境)。

这里介绍掉电时转变为由晶体作为时基、交流电到来后又重新恢复由交流电作为时基的方法。

8.1 电路工作原理

后面的程序主要是为了说明其方法,子程序"SET_CLCK"总是将时钟设置为 00:00:00,Friday,January,1990(24 小时)。实际应用时应设置当前时间和日期。

软件使 MCU 处于不断的循环中,当读取 MC68HC68T1 的时间(小时、分、秒)时,将其转换为 ASCII 码,然后通过异步串行通讯接口(SCI)传输到显示器件,如 CRT 终端。不发送换行字符,故显示保持在一行,看起来时间不断增加。断电后,系统由 3.6V 的 NiCd 电池供电。

系统的结构如图 8-1 所示。MCU 采用 8MHz 晶体,内部总线频率为 2MHz,可产生 9600 波特的标准数据波特率。MC145407 线驱动器/接收器用于 RS-232 电平接口,它只需 5V 电源,片内可产生 RS-232C 所需的±10V 电平。

MCU 将 MC68HC68T1 设置为以交流电频率作为时基,这由向 MC68HC68T1 的时钟控制寄存器写入%11110100 这一值来实现。

时钟控制寄存器	D7	D6	D5	D4	D3	D2	D1	D0
	$\overline{\text{START}}$ / $\overline{\text{STOP}}$	LINE / $\overline{\text{XTAL}}$	XTAL SEL1	XTAL SEL0	50/60 Hz	CLK OUT2	CLK OUT1	CLK OUT0

D7 位=1,允许时钟计数器链。

D6 位=1,选择交流电为时基。

D5、D4 位=1 1,选择 32.768kHz 晶体。

D3 位=0,选择 60Hz 作为时基。

D2~D0 位=100,禁止 CLK OUT 引脚。

注意,当采用 50/60Hz 作为时基时 MC68HC68T1 掉电检测电路不能使用。这要求中断控制寄存器的 POWER SENSE 位(位 5)置为 0。该值为复位时的省缺值,不需用户初始化。

由于正常工作时向电池充电,掉电后由电池向 MC68HC68T1 供电,所以电池应为充电电池。

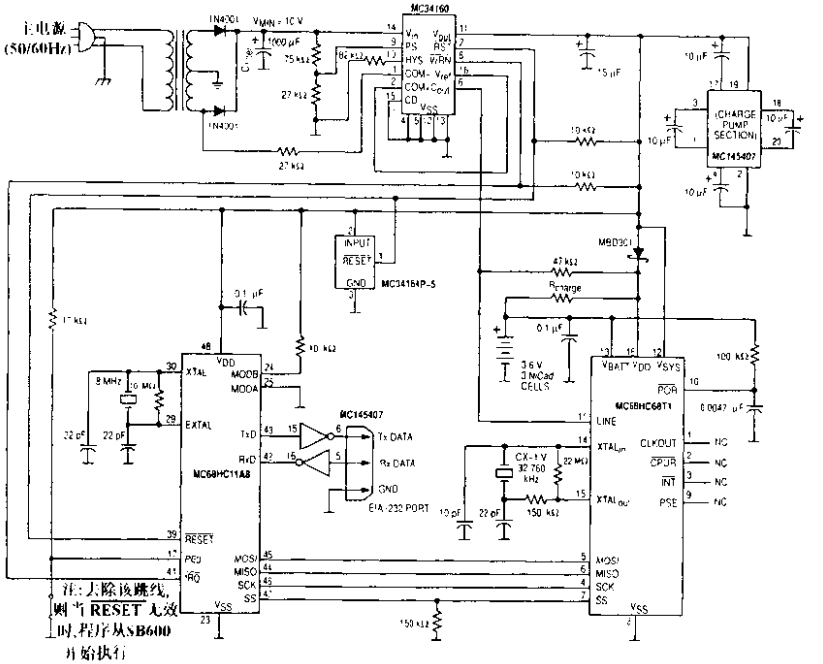


图 8-1　实时时钟的应用系统电路图

由于 MC68HC68T1 内部的电源掉电检测电路被禁止，这里采用 MC34160 电源监检电路检测电源状况。所关心的两个功能是电源报警和复位输出。当电源施加到该系统时，复位输出向 MCU 产生复位信号。当电压降到一定程度时，电源报警输出向 MCU 产生中断。产生报警输出时，电压并未降到最小工作电压以下，如图 8-2 所示。

MC34160 还具有电压比较器，用于将 60Hz 正弦波转换为驱动 MC68HC68T1 的 LINE 的逻辑信号。该电压比较器具有迟滞特性，改善了抗噪声干扰能力。

当电源报警输出向 MCU 产生中断时，MCU 完成当前的指令，然后进入中断处理程序。该程序向时钟控制寄存器写入%10110100，将 MC68HC68T1 设置为以晶体为时基。注意，只有位 6 发生了变化。然后中断处理程序将有用的其他信息存储在 MC68HC68T1 的内部 RAM 中。在该例中，没有其他信息需要保存，故 MCU 进入 STOP 状态。

如果正在进行串行发送时接受中断，可能出错，因为中断处理程序会使发送中止。会导致 SLP 写冲突或向 MC68HC68T1 发出错误数据。解决的方法是：在该程序段开始时禁止中断，结束处再允许中断。

图 8-1 中电源滤波电容 C_{filter}的值必须这样选择：它应该提供足够长的时间使 MCU 在电

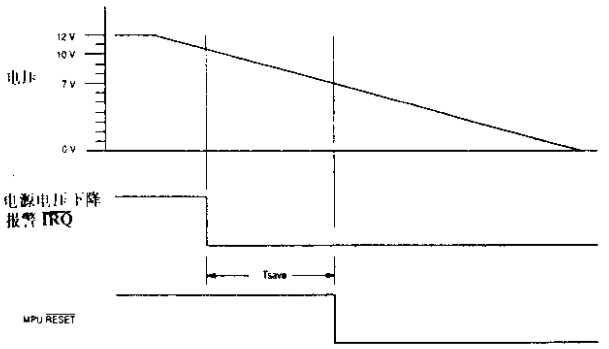


图 8-2　电源掉电时序

源降到正常工作电压之前执行完中断处理程序。按图中的值，电源报警输出产生中断后，可工作 150ms。

MC34164 低压检测电路是为了保证当电源电压降到 MC34160 正常工作电压以下时，MCU 保持在复位状态。这样可防止错误地操作。

8.2　软件

软件由 MC68HC11 交叉宏汇编编写。MCU 含有一个调试监控程序"BUFFALO"。软件存储于 MCU 的片内 EEPROM 中。该程序的目的是向与 MCU SCI(UART)相连的 CRT 终端输出时间信号。当发生掉电时，IRQ 使 MCU 中断，然后实时时钟改为由晶体控制。交流电恢复供电时，再恢复由 50/60Hz 电力线作为时基。

```
* MC68HC68T1 寄存器地址
Seconds     EQU  $ 20        秒寄存器
Minutes     EQU  $ 21        分寄存器
Hours       EQU  $ 22        小时寄存器
Day         EQU  $ 23        星期寄存器
Data        EQU  $ 24        日寄存器
Month       EQU  $ 25        月寄存器
Year        EQU  $ 26        年寄存器
Sec-Alarm   EQU  $ 28        秒闹钟
Min-Alarm   EQU  S 29        分闹钟
Hrs-Alarm   EQU  $ 2A        小时闹钟
Status      EQU  $ 30        状态寄存器
First-Up    EQU  %00010000
Control     EQU  $ 31        时钟控制寄存器
Start       EQU  %10000000   Start=1 允许分频链
Line        EQU  %01000000
```

```
XTL-SEL1    EQU  %00100000  选 32.768kHz
XTL-SEL0    EQU  %00010000
Line-50     EQU  %00001000  60Hz 时基清零
INT         EQU  $32        中断控制寄存器
Write       EQU  %10000000  地址字的写允许位
*MC68HC11A8 定义
REGBASE     EQU  $1000      I/O 寄存器基地址
IRQ_VEC     EQU  $EE        IRQ 的间接向量(3 字节)
SPSR        EQU  $29        SPI 状态寄存器
SPIF        EQU  %10000000  传输完成标志
SPCR        EQU  $28        SPI 控制寄存器
SPDR        EQU  $2A        SPI 数据寄存器
BAUD        EQU  $2B        SPI 波特率控制
SCCR1       EQU  $2C        SCI 控制寄存器 1
SCCR2       EQU  $2D        SCI 控制寄存器 2
TE          EQU  %00001000  发送允许
SCSR        EQU  $2E        SCI 状态寄存器
TDRE        EQU  %10000000  发送寄存器空标志
SCDR        EQU  $2F        SCI 数据寄存器
PDRTD       EQU  $08        D 口数据寄存器
SS          EQU  %00100000  至 HC68T1 的从机片选输出
DDRD        EQU  $09        D 口数据方向寄存器
*其它
CR          EQU   13
JMP         EQU   $7E
TOS         EQU   $35
            BSCT
*高速暂存地址
Time        RMB   3
String      RMB   8
            ORG   $B600     EEPROM 起始地址
Reset       EQU   *
            LDS   #TOS      堆栈初始化为 RAM 顶
            LDX   REGBASE   指向 I/O 寄存器的基址
*初始化 SCI 口
    LDAA    #%00110000          9600 波特(8MHz 晶体)
    STAA    BAUD,X
    BSET    SCCR2,X,#TE         允许 SCI 发送器
    *初始化 SPI
    BSET    DDRD,X,#00111000    设置 SS,SCK 和 MOSI 为输出
    LDAA    #%01010100          允许 SPI,位速率 1MHz,CPOL=1,
    STAA    SPCR,X              CPHA=1
```

*初始化间接向量表。HC11A8 的中断向量位于 ROM,不能修改。这些中断向量指向 RAM 中的间接向量表,因此用户能随意设置向量地址。

```
LDAA    #JMP
STAA    IRQ_VEC     将"JMP IRQ"存入表中
LDAA    $IRQ        IRQ 中断处理程序的地址
STD     IRQ_VEC+1
*由于需要不断地设置 MC68HC68T1,它总是需要重新初始化。
BSET    PORTD,X,#SS             允许 HC68T1
LDAA    #STATUS                 读状态寄存器
JSR     SQUIRT                  向 68HC68T1 发出地址
JSR     SQUIRT                  虚拟访问,以获得返回值
BCLR    PORTD,X,#SS             禁止 68HC68T1
TBA
BSET    PORTD,X#SS              允许 MC68HC68T1
LDAA    #CONTROL! +WRITE        写入控制寄存器
JSR     SQUIRT                  发送地址
*HC68T1 设置为交流电时基,晶体振荡器设置为 32768Hz
LDAA    #START! +LINE! +XTL_SEL1! +XTL_SEL0
JSR     SQUIRT
BCLR    PORTD,X,#SS
*设置时间
BITB    #FIRST_UP
BEQ     DKIP_SET
BSR     SET_CLCK                设置时钟时间
*允许 STOP 指令
SKIP_SET    TPA                 取 CCR 的值
            ANDA    #%01111111  STOP 屏蔽位清零
            TAP
MAIN        EQU     *
            SEI
            BSET    PORTD,X,#SS     允许 HC68T1
            LDAA    #SECONDS        先读秒寄存器
            JSR     SQUIRT
            JSR     SQUIRT
            STAA    TIME+2          存秒值
            JSR     SQUIRT          读分寄存器
            STAA    TIME+1          存入
            BSR     SQUIRT          读小时寄存器
            STAA    TIME            存入
            BCLR    PORTD,X,#SS
            CLI
*将 BCD 时间转换为 ASCII
LDAA    TIME        取小时值
BSR     BCD_ASCI    转换为 ASCII
STD     STRING
LDAA    TIME+1      取分值
```

```
        BSR     BCD_ASCI
        STD     STRING+3
        LDAA    TIME+2      秒值
        BSR     BCD_ASCI
        STD     STRING+6
        LDAA    #':
        STAA    STRING+2
        STAA    STRING+3
*将时间发送到 SCI 口
        LDAB    #8          发送 8 个字节
        LDY     #STRING     指向数据串
SEND_CHR LDAA   0,Y
        BSR     OUTCH       输出字符
        INY
        DECB
        BNE     SEND_CHR
        LDAA    #CR         发送回车
        BSR     OUTCH
        BRA     MAIN
*掉电时 MC34160 芯片产生 IRQ
IRQ     EQU     *                       中断程序
        LDX     #REGBASE                指向 I/O
        BSET    PORTD,X,#SS             允许 MC68HC68T1
        LDAA    #CONTROL! -WRITE        写入控制寄存器
        BSR     SQUIRT
        LDAA    #START! +XTL_SEL1! +XTL_SEL0
        BSR     SQUIRT
        BCLR    PORTD,X,#SS
        STOP
        PAGE
*子程序
*将 A 中的 BCD 码数转换为 ASCII 数,并位于 D 中
BCD_ASCI EQU    *
        PSHA
        ANDA    #%00001111
        ORAA    #'0             屏蔽除最低位数字外的所有数字
        TAB                     转换为 ASCII
        PULA                    置于 D 的低位
        LSRA
        LSRA
        LSRA
        LSRA
        ORAA    #'0
        RTA
```

```
*设置时间子程序,时间为 00:00:00 January 1,1990,Friday。
SET_CLCK  EQU   *
          BSET  PORTD,X,#SS
          LDAA  #SECOND! +WRITE 首先写入秒寄存器
          BSR   SQUIRT
          CLRA                  秒=0
          BSR   SQUIRT
          CLRA                  分=0
          BSR   SQUIRT
          CLRA                  小时=0
          BSR   SQUIRT
          LDAA  #6              星期=星期五
          BSR   SQUIRT
          LDAA  #1              日=1
          BSR   SQUIRT
          LDAA  #1              月=January
          BSR   SQUIRT
          LDAA  #$90            年=1990
          BSR   SQUIRT
          LDAA  PORTD,X,#SS
          RTS
```

*寄存器A的内容输出至SPI口,返回时A含有从MC68HC68T1接收的字符。其他寄存器返回时不变。从机选择必须由调用程序处理。

```
SQUIRT    EQU   *
          PSHX                  存工作寄存器
          LDX   #REGBASE        指向I/O寄存器
          STAA  SPDR,X
WAITSPIF  BRCLR SPDR,X,#SPIF,WAITSPIF  循环,直到发送完成
          LDAA  SPDR,X
          PULX                        恢复工作寄存器
          RTS                         返回调用程序
```

*该子程序将寄存器A的内容发送到SCJ口。

```
OUTCH     EQU   *
          PSHX
          LDX   #REGBASE
WAITTDRE  BRCLR SCSR,X,#TDRE,WAITTDRE等待发送缓冲器空
          STAA  SCDR,X
          PULX
          RTS
          END
```

9. 用单片机产生脉冲的方法

单片机经常需要用 MCU 产生脉冲来锁存显示数据、传送信息或执行其它功能。但每种应用场合对脉冲的精度或周期的要求不同，本节讨论各种脉冲的产生方法，分别介绍由不同类型的定时器产生短脉冲和长脉冲的方法。

象 MC68HC05J1 这类 MCU 具有较简单的定时器，因此，它产生的短脉冲精度较差，并且产生脉冲的软件较复杂。MC68HC05C8 等的 16 位定时器功能较强、使用也方便。下面讨论中均认为 MCU 的内部操作频率为 2MHz，晶体频率为 4MHz。M68HC11 系列单片机可以同时产生多路脉冲。

9.1 短脉冲

一般产生几十 μs 以上的脉冲容易实现，短脉冲的产生方法应视具体要求而定。若要产生脉冲持续时间与时钟周期差不多的短脉冲，则只需将某个引脚写为高电平，然后再写为低电平即可：

```
PULSE   BSET    BIT0,PORTA
        BCLR    BIT0,PORTA
```

HC05 单片机产生的脉冲持续时间为 2.5μs，68HC11A8 产生的脉冲为 3.0μs。如果所需的脉冲长于 2μs，上述两条指令可用空操作指令(NOP)或其它指令隔开，以产生所需的脉冲宽度。由于 NOP 指令占用 MCU 资源，也可由循环产生所需脉冲：

```
SETUP   LDA     DURATION
PULSE   BSET    BIT0,PROTA
LOOP    DECA
        BNE     LOOP
        BCLR    BIT0,PORTA
```

MC68HC05 产生的脉冲时间为 2.5+3.0×DURATION(μs)，MC68HC11A8 产生的脉冲时间为 3.0+2.5×DURATION。脉冲的分辨率为 2.5 或 3.0μs，也可在循环内或循环外增加几个 NOP 指令(1μs)或 BRN 指令(1.5μs)以提高脉冲持续时间精度。注意，产生短脉冲时不用内部定时器，因为设置定时器和读定时器的时间可能接近甚至超过脉冲时间。

9.2 长脉冲

对于 MC68HC11 MCU，利用其定时器可容易地产生多路脉冲，最长脉冲可达 131ms。最短可达 0.5μs(利用强迫输出比较)，程序如下：

```
PULSE  LDD    TIMER
       ADDD   #50
* The delay is selected according to
* timer prescaler, interrupts, etc.
* (min 33)
       STD    TOC1
       ADDD   #1       脉冲宽度
```

```
        STD     TOC2
        LDAA    #$40    驱动 A6 为高
        STAA    OC1M
        STAA    OC1D
        LDAA    #$80
        STAA    TCTL1   驱动 A6 为低
* ENABLE INTERRUPT, ETC
```

而 MC68HC05 中的定时器没有强迫比较功能，只能设置定时器控制寄存器(TCR)的 OLVL 并等待比较符合使 TCMP 为高。假设没有中断，下列程序段在 12μs 后产生脉冲：

```
        BSET    OLVL,TCR
* OUTPUT_COMPARD=TIMER+DELAY
        LDX     ACHR
        LDA     ACLR
        ADD     $DELAY
        BCC     OC1
        INCX
OC1     STX     OCHR
        STA     OCLR
```

DELAY 的值为 21 时产生约 12μs 延迟。注意写入或存储 16 位寄存器时总是先访问高位字节，以保持 16 位数据的连贯性。将所需的脉冲宽度加到输出比较值上，把新的值写入 OCR 并将 OLVL 置为 0，则产生所需脉冲。程序如下：

```
HERE    BRCLR   OCF,TSR,HERE
        LDA     OW_L            脉冲宽度的低位字节
        ADD     OCLR
        TAX                     暂存
        LDA     PW_H            脉冲宽度高位字节
        ADC     OCHR            含进位加
        STA     OCHR            禁止 TOC
        STX     OCLR
```

HC11A8 和 HC05C8 MCU 自动产生脉冲下降沿，不需软件干预。对于 MC68HC05J1 等 MCU，它们的 15 位定时器没有输出比较引脚，需采用中断的方法产生脉冲。

下面讨论用 MC68HC05J1 产生 10ms 脉冲的方法。HC05J1 只有周期中断功能，中断源可以是定时器溢出(512μs)或实时中断(RTI)。RT1 和 RT0 分别为 00、01、10、11 时对应的 RTI 中断周期为 8.2ms、16.4ms、32.8ms、65.5ms。

为获得 10ms，需定时器 5000 次计数(4MHz 晶体)。可采用下面方法获得 10ms：

一个 RTI 周期(RT1=0,RT0=0)=8192μs
三次定时器溢出 512×3 =1536μs
544 个内部周期(136 次计数 =272μs
} 共 10ms

这种方法主要一点是确定脉冲的起始时间。仔细计算每条指令的时间可获得高精度脉冲时间。另一种方法是只利用 TOF 溢出也能获得 10ms。因为每 512μs 产生一次中断，故对 CPU 的性能有些影响。下面的程序产生 10ms 脉冲(用 TOF 溢出易于确定脉冲开始边界)：

```
* 设脉冲宽度存放于 PWH 和 PWL 中，分辨率为 2μs，10ms 脉冲宽度对应的值为 $1388
START   BCLR    TOF,TCSR
        LDA     TIMER
        BSET    BIT0,PROTA
```

```
        COMA
        SBA     PW_L            脉冲宽度的低位字节
        BSS     PW1
        DEC     PW_H            借位
PW1     LDX     #$AA
        MUL
        STX     PW_L
        BSET    TOFE,TCSR
        CLI
*定时器中断执行其他工作
TOFI    DEC     PW_H
        BNE     END_T
        LDA     PW_L
        BEQ     PLS_L
LOOP    DECA
        BNE     LOOP
PLS_L   BCLR    BIT0,PORTA
END_T   RTI
```

上例中，若用两个字节对溢出次数进行计数，则脉冲长度可达 $2^{16}\times 512\mu s\approx 32s$，绝对误差均相同，因此长脉冲的相对误差很小。MC68HC05C8 等具有 16 位定时器的 MCU，也可以用类似的方法产生时间很长的脉冲。用这种方法能产生数分钟(最高可达 2 小时)的脉冲，精度可达百万分之一。中断占用的时间也很小，仅为 25μs/131ms，约为 0.2%。

MC68HC05C 系列 MCU 利用定时器输出比较功能产生脉冲的程序如下(脉冲宽度可从几微秒至 2 小时以上)：

```
*****常数定义*****
PORTA   EQU     0
DDRA    EQU     4       A 口数据方向寄存器
TCR     EQU     $12     定时器控制寄存器
ICIE    EQU     7       输入捕捉中断允许位
OCIE    EQU     6       输出比较中断允许位
TOIE    EQU     5       定时器溢出中断允许位
IEDG    EQI     1       输入边沿
OLVL    EQU     0       输出电平
TSR     EQU     $13     定时器状态寄存器
ICF     EQU     7       输入捕捉标志
OCF     EQU     6       输出比较标志
TOF     EQI     5       溢出标志
ICHR    EQU     $14     输入捕捉寄存器
ICLR    EQU     $15
OCHR    EQU     $16     输出比较寄存器
OCLR    EQU     $17
CHR     EQU     $18     定时器计数寄存器
CLR     EQU     $19
ACHR    EQU     $1A     第二个定时器计数寄存器
ACLR    EQU     $1B
```

```
*****程序常数*****
        ORG     $20
DELAY   FCB     6       脉冲起始延时
MIN_PLS FCB     5       最小脉冲宽度对应的时钟数
DO_PLS  FCB     $01,$C9,$C3,$80
*****变量*****
        ORG     $BA
PULSE   RMB     4       最大时间为 143.1655765 分钟
*设晶振为 4MHz,则最大脉冲时间为 2^16×2^16×2μs≈2.4 小时。
*对于较长脉冲,可用 32kHz 晶体以降低功耗,但要注意 MCU 的执行速度也降低 122 倍。
*类似地再增加一个字节对溢出次数计算,时间可达 25.45 天。
*定时器溢出中断和输出比较由起始脉冲子程序(STRT_PLS)初始化。
FLAGS   RMB     1
FIRE    EQU     7       表明已开始产生脉冲
LAST    EQU     6       表明已发生最后一次中断
*****复位程序*****
        ORG     $100
RST_INT:
        CLR     TCR
        CLR     FLAGS   将所有标志位清零
        LDA     #$FF
        STA     DDRA
        LDA     #$02
        STA     PORTA
*****主程序*****
MAIN    BRSET   FLRE,FLAGS,FIRED        复位后 FIRE 位清零
        LDX     #3                      装入 4 个字节
LDAD    LDA     DO_PLS,X
        STA     PULSE,X
        DECX
        BPL     LOAD
        JSR     STRT_PLS
FIRED   BRCLR   LAST,FLAGS,MAIN         LAST 标志清零,转主程序
        NOP
        BRA     MAIN
*****脉冲起始子程序*****
STRT_PLS:
        SEI                     禁止中断
*若 PULSE_WIDTH>$FFFF,则允许中断,否则禁止中断。
        BSET    7,PORTA         开启显示 LED
        BSET    OCIE,TCR        允许 TOC 中断
        LDA     PULSE+1
        SUB     #1
        STA     PULSE+1
```

```
        LDA     PULSE
        SBC     #0
        STA     PULSE
        BCC     SP1
        CLR     PULSE+1
        CLR     PULSE
        BCLR    OCIE,TCR
* 若 0<PULSE_WIDTH<MINIMUM,则 PULSE_WIDTH=MIMIMUM
SP1     TST     PULSE+2
        BNE     LONG_PLS
        LDA     #PULSE+3
        BEQ     LONG_PLS
        CMP     MIN_PLS
        BHI     LONG_PLS
        LDA     MIN_PLS
        STA     PULSE+3
LONG_PLS:
* 若脉冲宽度小数部分为 0 或大于等于 MIN_PLS 时,开始产生脉冲。
        BSET    OLVL,TCR
* OUTPUT_COMPARE:=TIMER+DELAY
        LDX     ACHR        必须先读高位字节
        LDA     ACLR        定时器=X:A
        ADD     DELAY
        BCC     MARK_1
        INCX
        BRA     OC1
MARK_1  NOP                 调整执行次数
        BRA     OC1
OC1     STX     OCHR        禁止 TOC
        STA     OCLR        允许 TOC
* 若 DELAY 正确,则立即产生脉冲
* TOC:=TURN_ON+PULSE_WIDTH MOD $10000
        ADD     PULSE+3
        STA     PULSE+3
        TXA
        ADC     PULSE+2
        TAX
        LDA     PULSE+3
        CLR     PULSE+3
        CLR     PULSE+2
* 若允许中断,则 OLVL=1;否则 OLVL=0,并且脉冲终止
        BRSET   OCIE,TCR,OC2
        BCLR    OLVL,TCR        若禁止中断
OC2     STX     OCHR
```

```
            TST     TSR
            STA     OCLR
            BSET    FIRE,FLAGS      表明脉冲已结束
    *到此为止脉冲结束,这时若允许中断,则立即服务中断
            CLI
            RTS
    *****定时器中断程序*****
    TCMP_INT:
    *只有完成$10000次计数时才产生中断。用户需要将脉冲宽度的整数部分减1,并且如果最后一次中断,则应清除中
断并将输出电平置为0。
    若PULSE_WIDTH>#10000,则PULSE_WIDTH:=PULSE_WIDTH-$10000
    LDA     PORTA
    EOR     #$03
    STA     PORTA
    LDA     PULSE+1
    SUB     #1
    STA     PULSE+1
    LDA     PULSE
    SBC     #0
    STA     PULSE
    BCC     NOT_LAST
    *否则禁止中断,并且OLVL=0
    *若脉冲为最后一次计数,则中断清零,并且OLVL清零
    CLR     PULSE+1
    CLR     PULSE
    BCLR    7,PROTA
    BCLR    OCIE,TCR
    BCLR    OLVL,TCR
    BSET    LAST,FLAGS
    *若不是最后一个计数,则OCF清零
    NOT_LAST
            LDA     TSR
            LDA     OCLR
            RTI
    *****伪中断程序*****
    SPI_INT     RTI
    SCI_INT     RTI
    IRQ_INT     RTI
    SWI_INT     RTI
    *****中断向量*****
            ORG     $1FF4
    SPI_VEC     FDB     SPI_INT
    SCI_VEC     FDB     SCI_INT
    TIM_VEC     FDB     TCMP_INT
    IRQ_VEC     FDB     IRQ_INT
    SWI_VEC     FDB     SWI_INT
    RST_VEC     FDB     RST_INT
```

10. 利用单片机的输入捕捉功能检测脉冲宽度

定时器的输入捕捉功能广泛用于测量脉冲的宽度和周期，例如测量转速、按键的持续时间、信号周期，以及在过程控制中检测阀门的开启时间等。时间范围从微秒至秒、分、小时。

有多个参数表示脉冲特性，下面讨论中均假设信号电平为 V_{DD} 或 V_{SS}，信号电平转换的上升和下降时间很小，对 MCV 来说可以忽略。MCU 所关心的参数是脉冲起始时间和持续时间。

用 MC68HC05 测量脉冲宽度比用 MC68HC11 复杂得多。先讨论一下测量精度。对于 MC68HC05C 系列等的 16 位定时器，每 8 个晶振时钟定时器计数器加 1，故分辨率为 2μs (4MHz 晶体)。对于较短的脉冲，例如测量 50μs 的脉冲，则误差为 4%。而测量 1 分钟的长脉冲时，其误差只有 3.3×10^{-8}。因此，长脉冲宽度测量时的误差主要来自晶振频率的误差，典型晶振频率误差为 0.001%。

被测脉冲分为 3 类：①很窄的短脉冲(小于 20μs)；②长脉冲(几十微秒以上)；③含噪声的脉冲。下面具体介绍。

10.1 短脉冲

测量窄脉冲时，处理好高达 10μs 的中断等待时间非常重要，精度由系统的等待时间限制。测量脉冲的程序如下(假设脉冲加到 IRQ 引脚)：

```
INTRUPT:
            CLRA
            BIL     END_PULS
T_LOOP      INCA                计数循环
A:          BIH     T_LOOP
END_PULS:
```

在 END_PULS，累加器已测量了脉冲宽度，分辨率为 3μs。考虑没有屏蔽中断的最坏情况，累加器第一次获得数值的时间是 13.4μs(若不用 MUL 测为 11.5μs)。最快时间为 9μs，任何脉冲时间小于该值的脉冲其时间值均为 0。若测得的脉冲时间值大于零，则脉冲宽度为 3μs 乘以累加器的值再加上 9～13.5μs 的等待时间。该程序可测的脉冲最大长度为 770μs(不考虑溢出)。

如果被测脉冲加到 TCAP 引脚，还要考虑附加的等待时间。如果有足够时间使 IEDG 位反相并使 ICF 清零(最小约 20μs)，则该脉冲就是第二类脉冲。如果脉冲比该时间短，则输入捕捉功能捕捉不到第二个边沿。如果需要检测的有很窄的脉冲，硬件设计应慎重考虑。例如，可将一个 I/O 脚与 TCAP 相连，用 BRSET/BRCLR 测试该 I/O 引脚的状态，这种方法能检测的最短脉冲仍并不比加到 IRQ 引脚上好。但用 TCAP 引脚可测量任意极性的脉冲，最长达 131ms，分辨率为 2μs。

当然，如果能预测输入脉冲为窄脉冲，并能预先估计其起始时间，就可用 busy—wait 的方法检查两个边沿。这需要不断地测试输入引脚的状态，并相应地产生分支。例如，预计脉冲宽度为 5～100μs，可用下列测试程序：

```
PULSE   CLRA
```

```
P0      BRCLR   PIN,PORT,P0    等待第一个边沿
        BRSET   PIN,PORT,P1
        BRSET   PIN,PORT,P2
        ⋮
        BRSET   PIN,PORT,PN
        INCA
PN      INCA
        ⋮
P2      INCA
P1      INCA
* ACC 含有 2.5μs 的脉冲宽度(每位)
```

该程序具有 2.5μs 的分辨率，最小可达 2.5μs，更小的脉冲将丢失。这种方法只适于测量窄脉冲。

额外的循环指令可简化程序，但分辨率增大：

```
PULSE   CLRA
P0      BRCLR   PIN,PORT,P0
P1      INCA
        BRSET   PIN,PORT,P1
* ACC 含有 4.0μs 脉冲宽度(每位)
```

10.2 长脉冲

当脉冲宽度大于几十微秒时，利用定时器的输入捕捉功能可方便地测量脉冲宽度。即两次跳变沿对应的输入捕捉寄存器值之差就是脉冲宽度。若计算定时器的溢出次数，则可测量任意宽度的脉冲。下面讨论精确测量 60 秒脉冲的情况。若定时器每次计数为 2μs，60 秒对应 30×10^6 次计数，需 4 个字节。30×10^6 对应的值为 $1C9C380。

16 位定时器记录脉冲的边沿，忽略溢出。脉冲的最后时间与起始时间之差就是脉冲宽度的低位两个字节。对于 60 秒情况，输出比较寄存器的脉冲起始时间值与结束时间值之差应为 $C380。脉冲期间定时器溢出 $1C9(=457)次。

本节最后列出了 M68HC05 单片机长脉冲检测的程序清单。把该软件集成到用户程序时，有几点需要注意，特别是中断等待时间的影响。但基本结构与功能不变。应注意的几点是：①测量程序采用独立中断，没有子程序，因此压栈时为 7 字节。由此确定最大堆栈字节数。②若采用其它中断，注意进入中断程序时，中断屏蔽位自动置位。若屏蔽位不能清零，则其它中断的执行时间(加上等待时间)必须小于 500μs(或脉冲宽度，取较小者)，以保证测量精度。③第三个方面要考虑的是中断对 MCU 运行速度的影响。当测量长脉冲时，脉冲测量程序所占用的时间仅为 0.02%，故该程序对大多数程序的处理能力没有明显的影响。但每次定时器溢出中断约需 24μs，对软件循环时间有一定影响。

10.3 有噪声的脉冲

脉冲中的噪声很难与脉冲边沿区分开来，故通常很难对含有持续时间与分辨率相同数量级的脉冲进行正确测量。也就是说，信号中应没有 1μs 的噪声。消除噪声的最简便方法就是利用硬件低通滤波器。注意，低通滤波器会使脉冲边沿延迟，对测量精度或多或少有些影响。另外，可对每个边沿进行两次抽样检验：

```
PULSE   CLRA
P0      BRCLR   PIN,PORT,P0
        BRCLR   PIN,PORT,P0
P1      INCA
        BRSET   PIN,PORT,P1
        BRSET   PIN,PORT,P1
* ACC 具有每次计数 6.5μs 脉冲
```

可采用复杂的数字滤波算法消除噪声影响，这些已超出本节讨论范围。消除噪声的常用软件方法是定期对信号抽样，并由多次抽样值确定脉冲的边沿。具体算法与应用场合有关。

10.4 长脉冲检测程序清单

下面是 MC68HC05C 系列(或其它具有 16 位定时器的 MCU)利用输入捕捉功能实现脉冲检测的程序。所检测的脉冲宽度从几个微秒至 2 小时以上。

```
* * * * * 地址定义 * * * * *
DDRA    EQU     $04
PORTA   EQU     $00
TCR     EQU     $12     定时器控制寄存器
ICIE    EQU     7       输入捕捉中断允许
OCIE    EQU     6       输出比较中断允许
TOIE    EQU     5       定时器溢出中断允许
IEDG    EQU     1       输入边沿
OLVL    EQU     0       输出电平
TSR     EQU     $13     定时器状态寄存器
ICF     EQU     7       输入捕捉标志
OCF     EQU     6       输出比较标志
TOF     EQU     5       定时器溢出标志
ICHR    EQU     $14     输入捕捉寄存器(高位)
ICLR    EQU     $15
OCHR    EQU     $16     输出比较寄存器(高位)
OCLR    EQU     $17
CHR     EQU     $18     定时器/计数器(高位)
CLR     EQU     $19
ACHR    EQU     $1A     第二个定时器/计数器(高位)
ACLR    EQU     $1B
        OPT     1
* * * * * 变量 * * * * *
        ORG     $BA
AC_OLFL RMB     2       最大时间为 143.1655765 分钟
PULSE_W RMB     2       存停止时间和总脉冲
START_T RMB     2       存脉冲开始时间
* 设晶体为 4MHz，两个字节可累加 2^32×2μs≈2.4 小时，分辨率 2μs。
* 3 字节可达 25.45 天，4 字节可达 17.83 年。
FLAGS   RMB     1
ARM     EQU     7       MCU 准备就绪时置位
```

```
GOT       EQU    6     捕捉到脉冲时置位
          ORG    $100
*****复位中断程序*****
RST_INT   BSET   7,DDRA
          BSET   7,PORTA
          CLR    FLAGS
          LDA    #100
          JSR    DELAY
*****主程序*****
MAIN      BRSET  ARM,FLAGS,ARMED
          BSR    GET_PLS
ARMED     NOP
          BRSET  GOT,FLAGS,GOT_IT
          NOP
GOT_IT    NOP
          BRA    MAIN
*****ARM 捕捉子程序*****
*调用该子程序开始测量脉冲,脉冲宽度受溢出累加器限制。本例为正脉冲。
*只需改变 IDEG 即可测负脉冲。调用该子程序 22μs 后开始测量。
GET_PLS:
          BRSET  IDEG,TCR
          LDA    TSR
          LDA    ICLR
          BSET   ICIE,TCR
          BSET   TOIE,TCR     开始对溢出次数计数
          BCLR   7,PORTA
          BSET   ARM.FLAGS
          CLI
          RTS
*****定时器延迟子程序*****
*若无中断,则在(5ms×累加器 A 的值)时间后返回。
DELAY     LDX    #249
DLA1      DECX
          NOP
          NOP
          NOP
          NOP
          NOP
          NOP
          NOP
          BNE    DLA1
          DECA
          BNE    DELAY
          RTS
```

```
* * * * * 定时器中断程序 * * * * *
TIM_INT:
* 若允许其它定时器中断,则需下列指令
*       BRCLR   ICF,TSR,NO_TIC
        BRCLR   IEDG,TCR,LAST_EDG
        BCLR    IEDG,TCR                准备捕捉后沿
        LDA     ICHR
        LDX     ICLR
        STA     START_T                 存上升边沿值
        STX     START_T+1
        CLI
        RTI
* 以下测量脉冲的下降沿。TIC 寄存器含有脉冲终止时间的最低两个字节。
* 从中减去开始时间,必要时向 AC_OVFL 借位。
LAST_EDG:
                BSET    GOT,FLAGS
                BSET    7,PORTA
                BCLR    ICIE,TCR
                LDX     ICHR
                LDA     ICLR
                STX     PULSE_W
                STA     PULSE_W+1
* 下面程序考虑溢出。若 ICHR= $FF,并且 ACHR=0 和溢出标志已清零,则累加溢出。
                INCA                    测试是否为 $FF
                BNE     CALC_PW
                TST     ACHR
                BNE     CLEAR_A1
                BRSET   TOF,TSR,CLEAR_A1
                LDA     AC_OVFL+1
                SUB     #1
                STA     AC_OVFL+1
                BCC     CLEAR_A1
                DEC     AC_OVFL
CLEAR_A1:
                LDA     ACLR
CALC_PW         LDA     PULSE_W+1
                SUB     START_T+1
                STA     PULSE_W+1
                LDA     PULSE_W
                SBC     START_T
                STA     PULSE_W
                LDA     AC_OVFL+1
                SBC     #0
                STA     AC_OVFL+1
```

```
                BCC     TIM_EXIT
                DEC     AC_OVFL+1
TIM_EXIT        RTI
*****伪中断程序*****
SPI_INT         RTI
SCI_INT         RTI
IRQ_INT         RTI
SWI_INT         RTI
*****中断向量*****
                ORG     $1FF4
SPI_VEC         FDB     SPI_INT
SCI_VEC         FDB     SCI_INT
TIM_VEC         FDB     TIM_INT
IRQ_VEC         FDB     IRQ_INT
SWI_VEC         FDB     SWI_INT
RST_VEC         FDB     RST_INT
```

11. MC68HC05 SR3 与 MC6805R3 性能比较

MC68HC05 SR3 是 HCMOS 工艺制造，MC6805R3 为 NMOS 工艺，MC68HC05 SR3 的性能比 MC6805R3 有所增强，两者引脚兼容。用 MC68HC05 SR3 代替 MC6805R3 时，只需改动少量软件。

11.1 SR3 的主要优点

与 NMO SR3 相比，HCMOS SR3 主要优点如下：

• SR3 具有更宽的工作电压范围，为 3.0～5.5V；而 R3 的工作电压范围为 4.75～5.75V。在较低工作电压下工作可减小功耗。

• SR3 具有低功耗操作方式。在 STOP 模式下，功耗电流只有几微安，大大低于 R3。

• SR3 的输出与 TTL/CMOS 兼容，NMOS 的 R3 器件需掩膜选择上拉电阻才能与 CMOS 兼容。

• SR3 具有键盘中断和上拉电阻，特别适于具有键盘输入的便携产品。

• 可以关闭 SR3 的 A/D，以减小功耗。

• SR3 的最大总线速度为 2MHz，而 R3 为 1MHz。

11.2 存储器映象

MC68HC05 SR3 和 MC6805R3 的存储器映象如图 11-1 所示。SR3 的 RAM 为 192 字节，比 R3 有所增加。SR3 中 ROM 的起始地址为 $1000，而 R3 的起始地址为 $80。SR3 的 ROM 比 R3 略有增加，两者的向量区也不相同，程序设计时应注意这一点。在向量区，增加了$\overline{\text{IRQ2}}$向量，如图 11-2 所示。在 MC6805R3 中，IRQ2 与定时器公用同一个中断向量。用户需要检查杂用寄存器(MR)的中断申请位(位 7)和定时器控制寄存器(TCR)的定时器中断申请位(TIR 位，TCR 的位 7)来确定中断源。而对于 SR3，$\overline{\text{IRQ2}}$引脚的下跳沿将产生有效中断，然后寻取 IRQ2 中断向量。

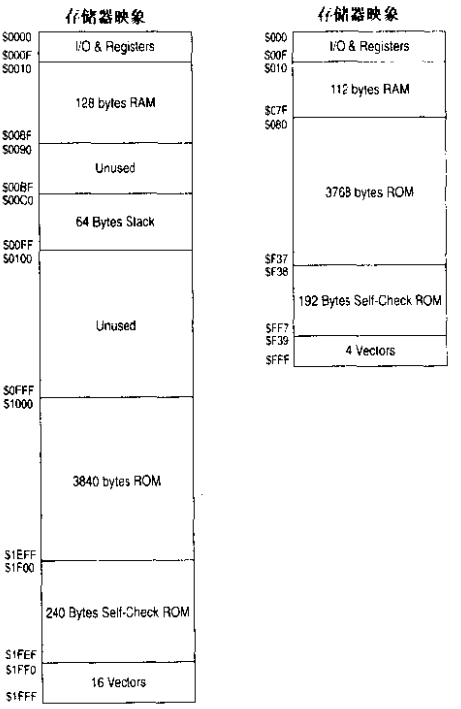


图 11-1 MC68HC05 SR3 和 MC6805R3 的存储器映象

11.3 输入/输出

SR3 为 CMOS 输入/输出电平。注意，SR3 的输出电平高于 R3。SR3 的输出驱动能力和输入噪声容限都有所增强，它可以直接驱动 CMOS 或 TTL 逻辑电路。

两种单片机的输出高低电平如图 11-3 所示。当电源电压为 5V 时，SR3 的输出高电平大于 4.7V(负载电流 0.8mA)，而 R3 的输出高电平为 2.4V 以上(负载电流 100μA)。

SR3 的输入噪声容限也大有提高。当电源电压为 5V 时，输入高电平电压为 $0.7V_{DD}$，即 3.5V。而 TTL 的最小输出高电平值为 2.7V，因此，若用 TTL 输出直接

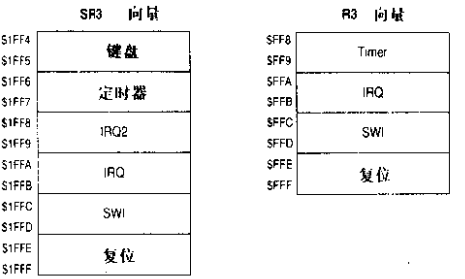


图 11-2 MC68HC05 SR3 和 MC6805R3 的中断向量

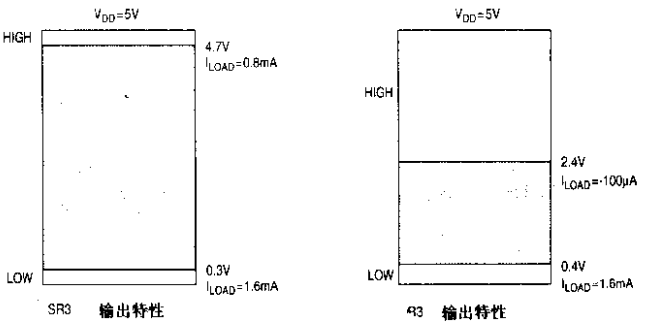


图 11-3 SR3 和 R3 单片机的输出特性

驱动 SR3 单片机，应加上拉电阻，提高输入电平。SR3 和 R3 单片机的输入特性如图 11-4 所示。另外，SR3 的 B 口还具有大电流驱动能力，能直接驱动 LED。将口选择控制寄存器(地址 $000A)的 PIL 位(位 5)置位，则 B 口的 PB5～7 可直接驱动 10mA 电流。这一功能在上电时选择。

11.4 可编程上拉电阻和键盘中断(KBI)

A、B、C 和 D 口的每个引脚都有可编程上拉电阻。将口选择控制寄存器的 PBP、PCP 和 PDP 位置位，则 B、C 和 D 口的引脚均有上拉电阻，阻值为 20kΩ。另外，口选择控制寄存器的 PB1 和 PB0 位分别决定 PB1 和 PB0 是否接一个小的上拉电阻，当 $000A 的 PB1 和 PB0 分别为 1 时，则 PB1 和 PB0 引脚分别接 1.8kΩ 的上拉电阻。口选择控制寄存器如下所示：

口选择控制寄存器	7	6	5	4	3	2	1	0
地址 $100A	0	0	PIL	PDP	PCP	PBP	PB1	PB0

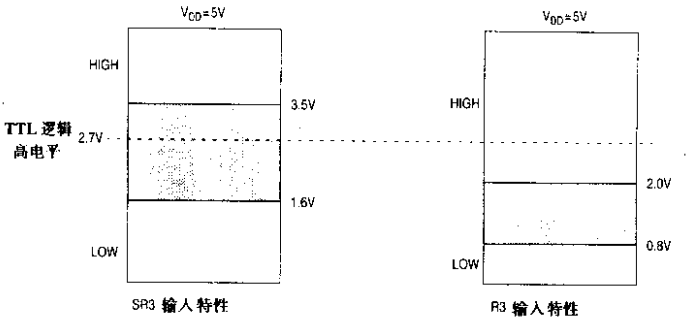


图 11-4 SR3 和 R3 的输入特性

A 口除具有 20kΩ 内部上拉电阻外，PA0～7 引脚还可作为键盘中断。此外，$\overline{\text{RESET}}$和$\overline{\text{IRQ}}$引脚均有 100kΩ 的上拉电阻。

11.5 $\overline{\text{IRQ}}$和$\overline{\text{IRQ2}}$

在 MC68HC05 SR3 中，可软件编程选择 IRQ 的触发方式，这由杂用寄存器(MCR)的 INTO 和 INTE 位控制。可选择 IRQ 为固定边沿触发或边沿和电平触发。而 MC6805R3 只能为边沿触发。在 SR3 中，MCR 的 IRQ2E 位决定是否允许第二个外部中断 IRQ2。MCR 的 IRQ2F 位表明 IRQ2 的中断状态。在 MC6805R3 中，IRQ2 由杂用寄存器($00A)的位 7 和位 6 控制。

11.6 低压复位

MC68HC05 SR3 具有低压复位功能，该功能可由软件允许/禁止，决定于 MCR 的 LVRE 位。在 MC6805R3 中，低压复位由掩膜选择设置。SR3 的低压复位范围为 2.8～4.2V，而 R3 的低压复位范围为 2.7～4.7V。

11.7 A/D 转换器

HCMOS 的 MC68HC05 SR3 的 A/D 转换时间为 32 个机器周期，而 MC6805R3 的转换时间为 30 个机器周期。在 R3 中，所有 A/D 通道均固定为输入；而在 SR3 中，A/D 口在设置为通用 I/O 口时，引脚可设置为输入或输出。使用 A/D 时，D 口为 A/D 输入通道。A/D 不用的 D 口引脚由 I/O 口逻辑控制，用作通用 I/O。用于 A/D 的通道的引脚，其对应的 DDR 位必须清零。

SR3 中的 A/D 状态和控制寄存器含有 ADRC 位。当 CPU 时钟低于 1MHz 时，应该将 ADRC 位置 1，使用片内的 RC 振荡器。A/D 状态和控制寄存器的 ADON 位控制是否允许 A/D。将 ADON 位置 1，则允许 A/D；将 ADON 位清零，则禁止 A/D。SR3 和 R3 的 A/D 控制与状态寄存器如下：

MC68HC05 SR3

A/D 状态与控制寄存器	7	6	5	4	3	2	1	0
地址 $000E	COCO	ADRC	ADON	—	—	CH2	CH1	CH0

MC6805R3

A/D 状态与控制寄存器	7	6	5	4	3	2	1	0
地址 $009	COCO	—	—	—	—	CH2	CH1	CH0

11.8 定时器

SR3 和 R3 定时器均有 7 位可编程分频器。复位时，R3 的预分频器设置为除以 128，而 SR3 的预分频器的省缺状态为除以 16。

11.9 上电延迟

对于 SR3，上电复位延迟是用来在首先执行指令前允许振荡器达到稳定，可由掩膜选择延迟时间。对于 EPROM 或 OTP 型 MC68HC705 SR3，延迟由 MOR 寄存器的 TIMR2～TIMR0 决定，如表 11-1 所示。

表 11-1 MC68HC705 SR3 的延迟时间

TIMR2	TIMR1	TIMR0	延迟(指令周期数)
0	0	0	256
0	0	1	512
0	1	0	1024
0	1	1	2048
1	0	0	4096
1	0	1	8192
1	1	0	16384
1	1	1	32768

11.10 总线频率

两者的总线频率是不同的。MC6805R3 的总线频率为晶振频率除以 4，而 MC68HC05 SR3 的总线频率为晶振频率除以 2。它们的最高晶体频率都是 4MHz。注意，R3 和 SR3 的指令执行周期时间也是不同的。因此，有关定时的程序必须修改。

11.11 数据方向寄存器(DDR)的读修改写

在 R3 中，DDR 为只写寄存器，对这些寄存器的读是未定义的。因为 BSET 和 BCLR 为读-修改写指令，它们不能用来将数据方向寄存器的位置 1。但对于 SR3，DDR 为读/写寄存器，因此，可以使用 BSET 和 BCLR 指令。

12. 用 MC68HC705C8A 代替 MC68HC705C8

MC68HC705C8A 是 MC68HC705C8 的增强型号。两者内部有些区别，用户应该注意，如 PC7 的电流特性，MOR 寄存器编程等。自引导 ROM 程序也有所改变。

12.1 用 705C8A 代替 705C8

1. 用 705C8A 代替 705C8 的区别之一是 MC68HC705C8A 的 PC7 引脚(C 口位 7)的输出电流驱动能力有所增强。该引脚可以直接驱动 LED。MC68HC705C8 和 MC68HC705C8A 输出特性如下：

特性	MC68HC705C8	MC68HC705C8A
$V_{DD}=5.0V\pm10\%$		
PC7 驱动电流(I_{OH}) @ $V_{OH}=V_{DD}-0.8V$	0.8mA	5.0mA
PC7 流入电流(I_{OL}) @ $V_{OL}=0.4V$	1.6mA	20.0mA
$V_{DD}=3.3V\pm10\%$		
PC7 驱动电流(I_{OH}) @ $V_{OH}=V_{DD}-0.3V$	0.2mA	1.5mA
PC7 流入电流(I_{OL}) @ $V_{OL}=0.3V$	0.4mA	6.0mA

这两种器件的输出电流特性以及 MC68HC705C8A 直接驱动 LED 的连接方法如图 12-1 所示。

2. 705C8A 增加了两个 MOR 寄存器，为了仿真 705C8，必须不对 MOR1 和 MOR2 进行编程(即不允许 B 口中断/上拉电阻和 C4A 型 COP 复位)。705C8 EPROM 的擦除状态为 0，因此，705C8A EPROM 的省缺值即为禁止状态。也就是，$1FF0 和 $1FF1 必须编程为 $00。

3. 编程特性与 705C8 类似，705C8A 的编程电压应该为 $V_{PP}=14.5\sim15.0V$，与 705C8 相同，编程时间仍保持不变，为 2ms/字节。

4. 705C8A 采用 1.2μm CMOS 工艺制造，而 705C8 采用 1.75μm 工艺。705C8A 的工作电流 I_{DD}、WAIT 方式和 STOP 方式下的 I_{DD}均与 705C8 相同，最大 I_{DD}指标保持不变。

5. 705C8 中 SPI 的小缺陷在 705C8A 中得到更正。当 CPHA=1，CPOL=0，705C8 的 SPI 处于从机时，SPIF 位有时不会置位，不能正确地指示传输是否结束状态，这一问题在 MC68HC705C8A 中已纠正。

6. 位于地址 $1FF0～$1FF1 地址的自引导 ROM 程序有所改变。

12.2 使用 705C8A 额外增加的特性

根据客户的需要，705C8A 增加了几个特性，如 C4A 型的 COP，B 口中断/上拉电阻，PC7 的 LED 驱动能力。

C4A 型 COP 与 705C8 的 COP 类似，但 COP 定时器溢出周期固定为 $2^{18}/f_{osc}$，如图 12-2 所示。C4A 型 COP 用 18 位串行计数器实现，在总线速率为 2MHz 时，COP 溢出周期为 65.54ms。该 COP 是用来仿真 MC68HC05C4A 的。将位于地址 $1FF1 的 EPROM 位 0

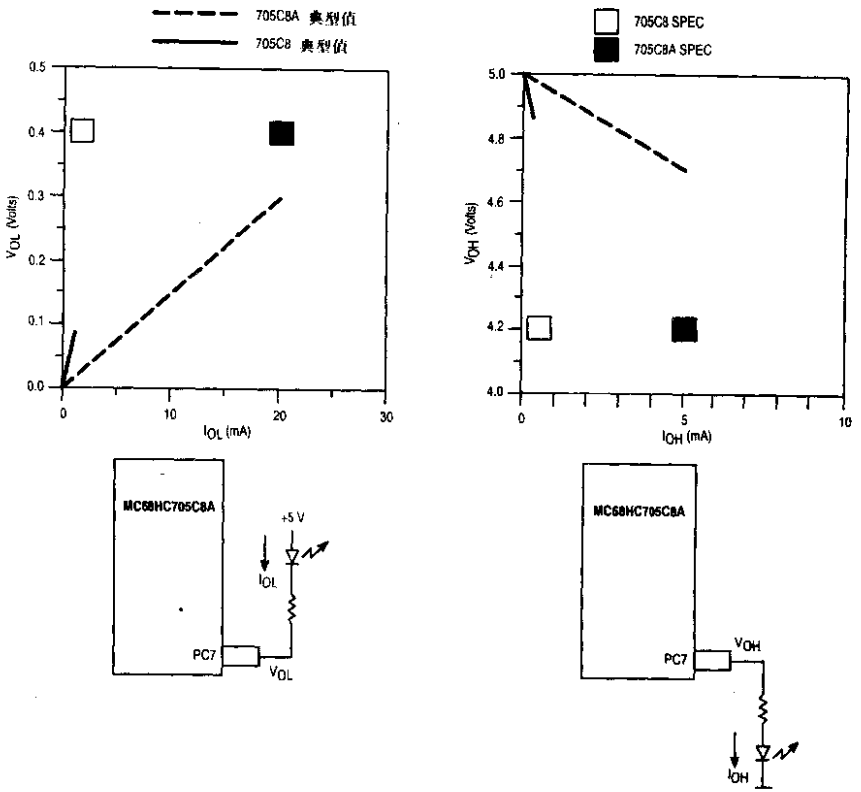


图 12-1　705C8 和 705C8A 的输出电流特性

(COPT 位)编程为 1，则允许 C4A 型 COP。若 COP 发生溢出，将发生系统复位。向地址 $1FF0 的 COPF 位(位 0)写入 0，可使 COP 定时器复位。读 $1FF0 地址将返回 MOR1 的内容。$1FF0 地址还用来控制上拉电阻，但写入 0 使 COP 复位对上拉电阻的状态没有任何影响。

705C8 型 COP 由 16 位定时器的一部分电路构成。当 $1FF1 的 EPROM 位。未编程时(状态 0)，允许 705C8 型 COP。时钟频率经 16 位定时器的前 11 位输出时为 $f_{op}/2^{15}$，COP 控制寄存器($1E)的位 1 和 0(CM1 和 CM0)可用来选择溢出周期，如表 12-1 所示。先向 COP 复位寄存器($1D)写入 $55，再向该寄存器写入 $AA，可使 COP 复位，防止 COP 溢出复位。

705C8A 的 B 口还具有键盘中断和上拉电阻。可以不必外接上拉电阻，简化硬件。中断功能可使软件设计简化，不必将所有口线线或接到 IRQ 引脚来检查按下的键，也不必用软件查询是哪个键按下。键盘连接方法如图 12-3 所示。当将用户程序编程时，将 $1FF0 的 MOR1 的相应位编程，则允许上拉电阻。一旦允许上拉电阻后，只能用紫外线(UV)擦除 EPROM 才能

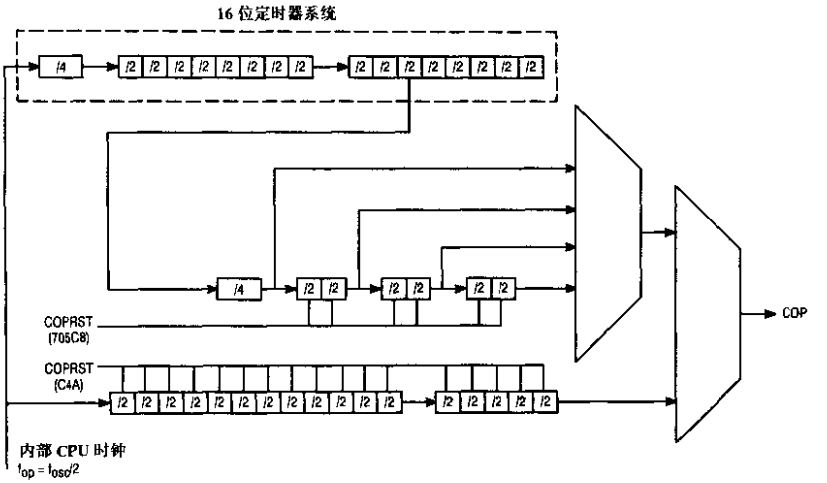


图 12-2　705C8A 的 COP 框图

禁止接上拉电阻。

表 12-1　705C8COP 与 C4A 型 COP 溢出周期比较

COP 类型	CM0	CM1	$f_{inf}/2^{15}$ 除以	$f_{osc}=4.0MHz$ $f_{op}=2.0MHz$	$f_{osc}=3.5795MHz$ $f_{op}=1.7897MHz$	$f_{osc}=2.0MHz$ $f_{op}=1.0MHz$	$f_{osc}=1.0MHz$ $f_{op}=0.5MHz$
705C8	0	0	1	16.38ms	18.31ms	32.77ms	65.54ms
705C8	0	1	4	65.54ms	73.24ms	131.07ms	262.14ms
705C8	1	0	16	262.14ms	292.95ms	524.29ms	1.048s
705C8	1	1	64	1.048s	1.172s	2.097s	4.194s
C4A	NA	NA	NA	65.54ms	73.24ms	131.07ms	262.14ms

下面是一个利用 B 口中断和上拉电阻实现键盘的例子。MOR1 必须编程为 $0F 以允许中断和上拉电阻，例子中首先设置 A 口和 C 口为输出，并分别向 A、C 口写入 $00 和 $55。C 口用来表明按下的键。每当按下一个键时，A 口加 1，并且如果抖动产生额外中断则将其记录。还将中断设置为只是边沿触发，将 PB7～PB4 设置为输出，PB0～PB3 输出为低。设置好 705C8A 后，清中断屏蔽位，执行 STOP 指令，以便在 705C8A 空闲时节省功耗。

一旦键按下，器件退出 STOP 方式，转向 IRQ 向量指示的中断服务程序。该程序等待 30ms，以防止抖动和确定按下的键位于哪一行哪一列。读行的状态之前，必须有足够的时间使上拉器件对刚释放的键进行充电。因为上拉器件为 P 沟道器件，其 IV 特性如图 12-4 所示。由于器件电流小，当键释放后不再为低电平时，上升到高电平需要较长的时间。读行状态之前应将这一上升时间考虑进去。

一旦将行和列译码后，程序保持循环，直到键释放为止。再次执行 30ms 暂停，以防止抖

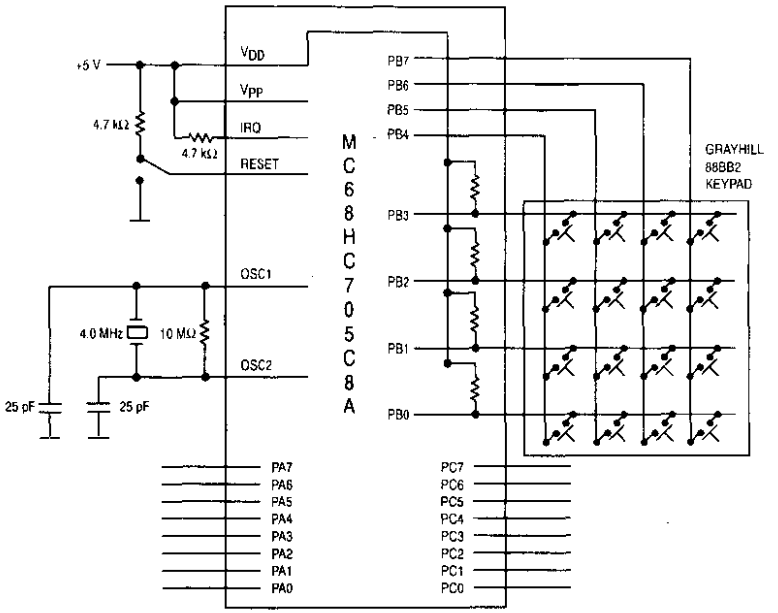


图 12-3　4×4 键盘例子

动。将行和列号写入 C 口输出，并使 A 口加 1 表明触动另一个键。服务程序以 RTI 结束，程序返回主程序，等待识别下一个键按下。

程序清单如下：

```
porta     equ     $ 00      * port A
portb     equ     $ 01      * port B
portc     equ     $ 02      * port C
portd     equ     $ 03      * port D
paddr     equ     $ 04      * port A data direction register
pddr      equ     $ 05      * port B data direction register
cddr      equ     $ 06      * port C data direction register
row       equ     $ 50      * row #
col       equ     $ 51      * column #
* 主程序
org       $ 200
start     lda     # $ 00
          sta     porta     * set port a low
          lda     # $ 55
          sta     portc     * set port c to $ 55
```

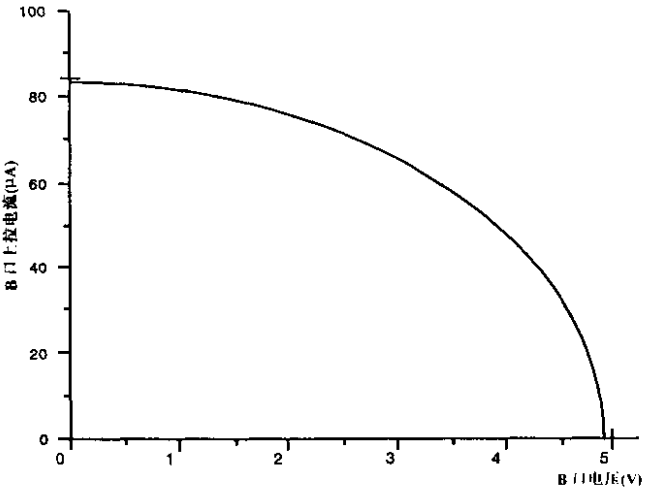


图 12-4 MC68HC705C8A B 口上拉器件电流特性(VDD=5V)

```
                lda     # $ ff
                sta     paddr       * port A — output
                sta     pcddr       * port C — output
                lda     # $ 00
                sta     $ 1fdf      * make interrupt edge only
*
lpl             lda     # $ 0f
                sta     portb       * drive PB7 — PB4 low
                coma
                sta     pbddr       * make PB7 — PB4 outputs
                cli                 * enable interrupts
                stop                * enter stop mode
                bra     lpl         * reconfigure port B
```

* 中断程序

```
org             $ ld00
irqsev          bsr     bounce      * wait 30 ms for key debounce
                lda     portb
                cmpa    # $ 0f      * check for a false interrupt
                beq     done
                lda     # $ 10      * start with the first column PB4
                sta     col
scol            lda     col         * enable columns one at a time
                sta     pbddr       * to determine the column
```

```
                lda     # $10       * wait until the pull — ups have
lp3             deca                * had a chance to pull the
                bne     lp3         * deselected columns high
                lda     # $fe       * check the rows one at a time
                sta     row
scan            lda     portb       * read the rows
                ora     # $f0       * don't care the high 4 bits
                cmpa    row
                beq     hold        * if match you found the row/col
                lda     row         * shift the row left and shift
                coma                * in a "1"
                lsla
                coma
                sta     row         * save next row
                cmpa    # $ef       * check to see if any rows left
                bne     scan
                lsl     col         * shift the column left
                bcc     scol
hold            lda     # $f0       * wait here until the key
                sta     pbddr       * has been released
                lda     portb
                anda    # $0f
                cmpa    # $0f
                bne     hold
                inc     porta       * inc the # times a key has been
                                    * pressed
                bsr     bounce      * wait for debounce
                lda     col         * write the row and column out
                coma                * to port c
                anda    row
                sta     portc
done            rti
bounce          lda     # $27       * debounce delay — 30 ms @ 2.0 MHz
again           ldx     # $ff
again2          decx
                bne     again2
                deca
                bne     again
                rts
* * *
org             $1ff0               * MOR1 — enable PB3—PB0
fcb             $0f                 * pullups
org             $1ffa
dw              irqsev              * IRQ vector
org             $1ffe
dw              start               * reset vector
```

13. MC68HC805B6 EEPROM 单片机编程模块

可用 MC68HC805B6 对 MC68HC05B6 和 MC68HC05B4 进行仿真。本文介绍 MC68HC805B6 内部 EEPROM 的编程技术。

1. 编程方法

采用多字节编程方法可将用户程序装入 MC68HC805B6 的 EEPROM 中，以便对 MC68HC05B6/B4 进行仿真。该操作由自引导操作模式实现。含在外部 EPROM 中的用户程序被拷贝到 MC68HC805B6 内部 EEPROM 中，并进行校验。

(1)EEPROM6 的擦除。晶振为 4MHz 时，仿真 MC68HC05B6/B4 ROM 的 6K EEPROM 通常擦除时间为 100ms。然后测试该区域的字节读出是否为 $FF。若有任一字节没有完全擦除，红 LED 将亮，并重新进行擦除过程，直到所有字节都擦净为止。

(2)EEPROM1 的擦除。该 256 字节是为了仿真 MC68HC05B 的 256 字节 EEPROM，其擦除与 EEPROM6 类似。擦除该 256 字节 EEPROM1 时，保密位也同时擦除。

(3)EEPROM 编程。EEPROM1 和 EEPROM6 按外部 EPROM 中的数据进行编程，MC68HC805B6 的非 EEPROM 地址被自动跳过。编程期间，绿 LED 以大约 3Hz 的速率闪烁。

(4)EEPROM 编程校验。EEPROM1 和 EEPROM6 的内容与 EPROM 的内容进行校验，若两者不同则红 LED 亮。编程无误则绿 LED 亮，红 LED 关闭。若红 LED 发出很弱的光，也是正常的。

2. 编程操作

MC68HC805B6 的 MCU 的 EEPROM 编程步骤如下：①未加电时安装 MCU 和 EPROM 器件。②置 S1 为 RESET 位置。③编程模块加＋5V 电源。④编程模块加＋19V 电源。⑤置 S1 为 RUN 位置。⑥一旦绿 LED 亮后，置 S1 为 RESET 位置。⑦去掉＋19V 电源。⑧去掉＋5V 电源。

注意，为防止损坏单片机，必须按上述步骤加电源或去掉电源。

3. 编程电路

编程电路如图 13-1 所示，电路中的元件如表 13-1 所示。电路应考虑下列几方面：

(1)IRQ 和 RESET。在复位期间要求 IRQ 引脚呈现大约 10V 电压。

(2)$V_{PP}6$。该电源由 PC7 引脚驱动的开关控制，该电路保证无论在什么时刻 $V_{PP}6$ 引脚的电平至少为＋5V 电压。在 EEPROM6 编程和擦除操作期间，＋19V 电压加到 $V_{PP}6$。齐纳二极管 ED1 防止电压超过＋20V，以保护 EEPROM。

(3)$V_{PP}1$。$V_{PP}1$ 引脚允许悬空，它是内部 EEPROM1 电荷泵(高压发生器)的输出。将该引脚嵌位至＋5V 可防止 EEPROM1 编程和擦除。

(4)D 口。自引导模式要求这些口引脚接 0V。

(5)PC5/PC6。在这种情况下，PC5 和 PC6 应接在一起。

表 13-1　MC68HC805B6 EEPROM 编程电路元件表

电阻		二极管		
R1	100K	D1,D2	1N914	
R2	1K	D3	LR3160	红 LED
R3,R4	470	D4	LG3160	绿 LED
R5	10M	D5,D6	1N5822	
R6	1K	D7	1N914	
R7	4K7	ZD1	BZY88C20	
R8	1K			
R9	4K7	插座		
R10	10K	IC1	52PLCC	
RP1-RP3	100K	IC2	28DIP	
电容				
C1	0.01μF			
C2	1.0μF			
C3	0.1μF	开关		
C4,C5	22pF	S1		
C6	100μF	其他		
C7	47μF	CR1	4MHz 晶体	
晶体管				
Q1,Q2	BC337-25			

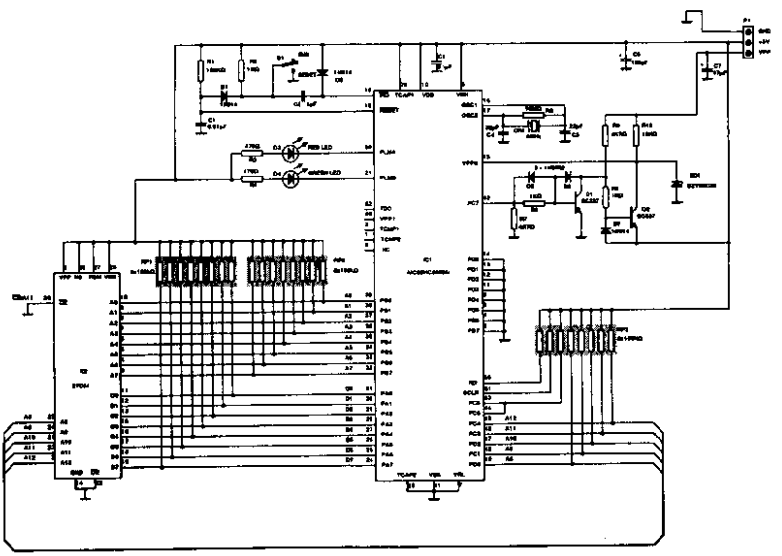


图 13-1　MC68HC805B6 编程电路

14. MC68HC05F2 DTMF 输出的廉价低压有源滤波器

MC68HC05F2 是全静态 CMOS 单片机，它有 256 字节 RAM，2048 字节 ROM，DTMF 发生器和 4 个大电流输出引脚(10mA)，可直接驱动 LED。

1. 电路说明

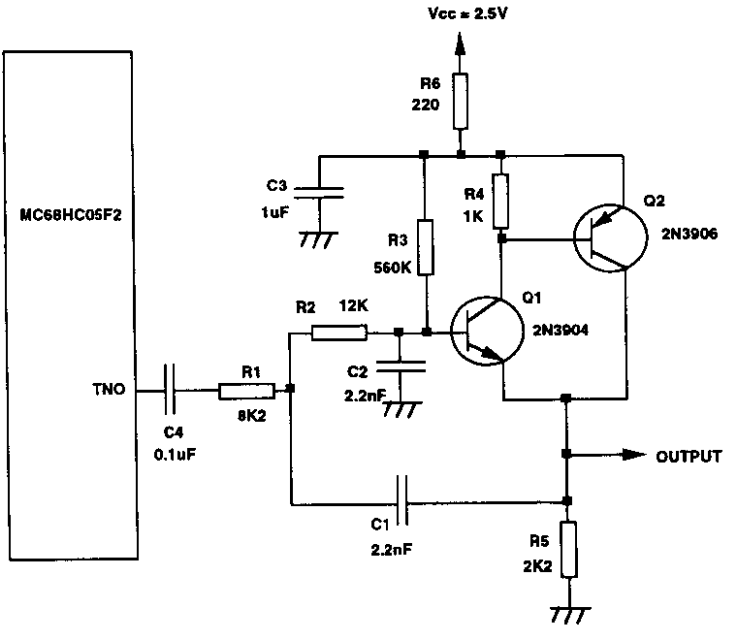


图 14-1 MC68HC05F2 DTMF 的滤波器

DTMF 输出的低通滤波器如图 14-1 所示。该电路为二阶低通 Butterworth 滤波器，截止频率约为 6.4kHz。晶体管 Q1 和 Q2 形成串行反馈。Q1 为增益为 1 的射极跟随器，Q2 形成负反馈，能改善线性度和稳定性。

改变 R4 可调整电路的静态电流。但 R4 的值太大会限制信号的正向电压摆幅。对于图中的元件值，静态电流低于 0.8mA(2.2V 时)。测得的静态电流为 0.3mA。

2. 低通滤波器

该滤波器为二阶 Butterworth 滤波器，传输函数为：

$$H(s)=\frac{W_0^2}{S^2+\sqrt{2}\,W_0 s+W_0^2}=\frac{\frac{1}{(R1\cdot C1\cdot R2\cdot C2)}}{S^2+\left(\frac{1}{R1\cdot C1}+\frac{1}{R2\cdot C1}\right)S+\frac{1}{R1\cdot C1\cdot R2\cdot C2}}$$

其中：

$W_0=2\pi\times 6.4\text{kHz}$

$(R1\cdot C1\cdot R2\cdot C2)^{-1}=(2\pi\times 6.4\text{kHz})^2$

$(R1\cdot C1)^{-1}+(R2\cdot C1)^{-1}=\sqrt{2}\,(2\pi\times 6.4\text{kHz})$

(取 C1=C2)

解上述方程，得

$R1=7.9\text{k}\Omega$

$R2=12.65\text{k}\Omega$

$C1=C2=2.486\text{nF}$

电路中采用最接近的阻容值，截止频率略高。该滤波器输出的典型频谱如图 14-2 所示($V_{CC}=2.2\text{V}$)。

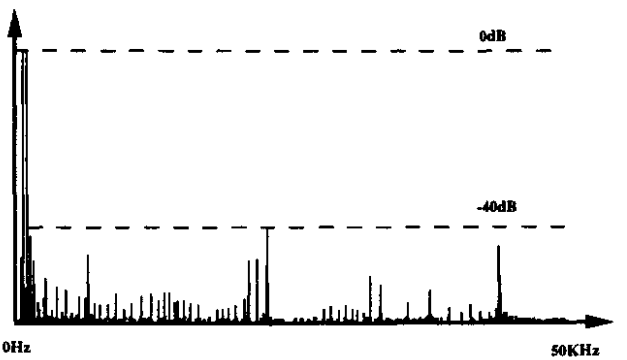


图 14-2　DTMF 滤波器响应的典型能谱图($V_{CC}=2.2\text{V}$)

15. 正确选用单片机

正确选用合适的单片机是设计 MCU 系统的关键。这里介绍选用单片机时应考虑的关键问题与具体选择方法。

1. 选择目的与过程

正确选择单片机主要目的是选择能满足系统要求(如性能、可靠性等)并且最廉价,以降低系统成本。

选择 MCU 之前,首先应弄清楚 MCU 所执行的工作。其次是根据有关资料选出能满足需求的单片 MCU,或使外部电路减到最少的 MCU,同时考虑费用和电路板空间(体积)、重量等。显然,单片 MCU 对减少成本和提高可靠性有利。最后,根据价格、开发工具、稳定性、制造厂商的支持等,从适用的单片机中选出最理想的型号。

2. 选择标准

下面按重要程度依次列出了选择单片机的主要标准:

· 应用系统的适用性:

(1)是否具有所需的 I/O 口引脚数? 太少则不能满足需求;太多则造成浪费。

(2)是否具所需的串行 I/O、RAM、ROM、A/D、D/A 等?

(3)CPU 是否能满足所需的计算处理能力,程序设计语言是否合适?

(4)项目资金是否可以承担得起采用该 MCU?

· 产品可用性:

该 MCU 是否能提供所需的用量,目前以及将来是否生产? 封装形式是否合适?

· 开发支持:

(1)汇编、编译程序。

(2)调试工具:评估模块(EVM)、电路内仿真器、调试监视器等。

(3)应用支持:应用工程师,应用示例等。

3. 系统需求

根据应用要求决定需要哪些外围器件。如接收数据所需什么操作,是采用中断、程序查询或人工响应? 需控制多少 I/O 引脚? 监视或控制的 I/O 器件有:RS-232C 终端、开关、继电器、键盘、传感器、音响、显示器(LED 或 LCD)、A/D、D/A 等。还应考虑电源电压及其误差和电源电流等。

另外,系统尺寸、重量、形状、外观是否有限制? 工作环境是否特殊(如温度、湿度、压力/高度)? 开发力量与开发时间等也是必须要考虑的。

4. MCU 资源

MCU 通常分为 8 位、16 位和 32 位。系统功能可尽量选用 8 位单片机完成。通常,随着时钟速度或总线速度的提高,计算能力增强,但功耗和成本也随之增加。并且,与之相关的电路费用也随着增加。一般 HCMOS 器件比 NMOS 器件速度高,而且功耗低。

MCU 内部资源的可靠性高,已经过制造者测试。通常片内资源包括存储器、定时器、系统时钟/振荡器、I/O 等。存储器包括 RAM、ROM、EPROM、EEPROM。定时器包括实时时钟和

周期中断定时器，注意定时器的分辨率和它的子功能，如输出比较/输入捕捉等。I/O 包括串行通讯口、并行口(I/O 线)、A/D、D/A、LCD 显示驱动器、VFD 显示驱动器等。另外还有 COP 监视系统(WATCHDOG)、时钟监视器、存储器设置等。

5. MCU 指令集

MCU 的指令集和寄存器对系统的功能起重要作用，因此应该仔细考虑。应考虑软件是否使用专门的指令，如乘、除、查表与插值等，低功耗 STOP 与 WAIT 指令、位操作指令(位置位、位清零、位测试、位置位分支、位清零分支)，以便于程序控制。

6. MCU 中断

当设计实时系统时，必须考虑中断结构，如系统需要多少中断，MCU 能否满足？中断级别及其屏蔽问题？承认中断后是否有单独的中 断处理程序向量？在速度要求较高的情况，如控制打印机，中断响应时间是确定 MCU 的关键。中断响应时间是指从中断开始到执行有关中断处理程序第一条指令的时间。

7. 厂商的支持

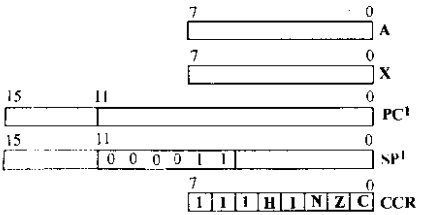
厂商或代理商不但要提供 MCU，还应该提供其他有关器件，如存储器、分离器 件、数字电路(74/54 系列、74HC 系列等)、专用器件(ASIC 和 CSIC)、可编程逻辑器件(PLD)等。

应考虑的有关问题包括：资料、开发支持、开发工具、供货货源及供货周期等。

如果没有适用的单片 MCU 能满足系统需求，并且所需的用量 又足够大，则用户可采用用户 MCU(CSIC)。用户自己确定 CPU 和所需要的 外围功能，厂商将这些功能集成在一起，是真正的单片系统，可靠性、稳定性高，使用方便。

8. Motorola 单片机

Motorola 单片机最常用的有 M68HC05 系列、M68HC11 系列、M68HC16 系列(16 位 MCU)、M68300 系列(32 位 MCU)。



注：长度取决于具体型号的存储/寻址空间，如2K 寻址空间 PC 为 11 位，16K 寻址空间 PC 为 14 位

图 15-1 M68HC05 编程模型

M68HC05 系列 8 位 MCU 具有 5 个 CPU 寄存器，如图 15 1 所示。M68HC11 系列是 8 位增强型 MCU，有 8 个 CPU 寄存器，如图 15-2 所示。M68300 系列为 32 位 MCU，有 32 个 CPU 寄存器，分为用户模式和管理(Supervisor)模式，编程模式分别如图 15-3 和 15-4 所示。同时提供各种简单和高级开发工具。EVS、EVM、EVB 只需几百美元。还有适用通用 PC 机的较强大的汇编程序，如支持宏、结构汇编、条件汇编、重定位模块(用于连接)。

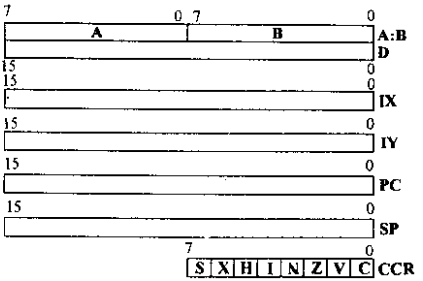


图 15-2 M68HC11 编程模型

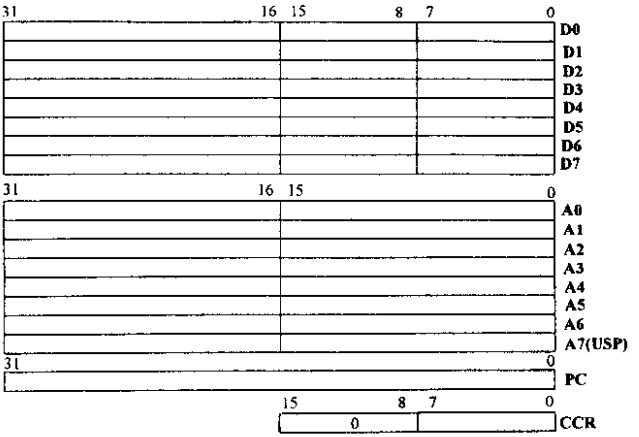


图 15-3 M68300 用户模式编程模型

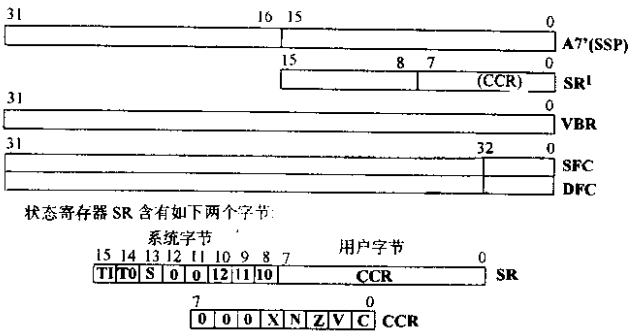


图 15-4 M68300 管理模式编程模型

16. HCMOS 单片机的电磁兼容(EMC)设计

目前 HCMOS 器件的操作速度早已达到最快双极型器件的速度，这对系统设计提出了更高要求。减小电磁干扰(EMI)将有利于减小系统噪声，提高抗干扰能力。设计阶段早期若没有考虑 EMC，可能导致负出重大代价，如重新设计印刷电路板(PCB)、导致延迟增大等。

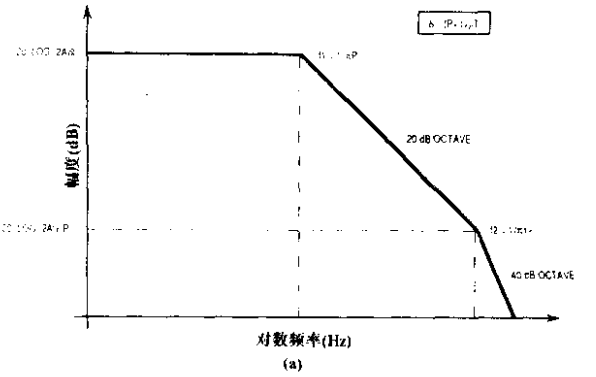


图 16-1 周期波形及其频谱
(a)频谱； (b)周期波形

16.1 射频干扰(RFI)问题

周期波形的频谱和方波波形如图 16-1 所示。通过 Fouriet 分析，周期波形的频谱由分离的频率分量组成，包括基频 f_0 和谐波($n \times f_0$)。对典型的梯形波，每种频率分量的幅度与基频、上升时间以及占空比有关。频率加倍、上升时间减小一半或使占空比减小一半则谐波频率幅度加倍。在图 16-1 中：

$$f_1 = 1/\pi P(Hz)$$

$$f_2 = 1/\pi t_r(Hz)$$

其中 P 为脉冲宽度；t_r 为上升时间；T 为周期；$\delta = (P + t_r)/T$；V 为幅度。

当频率高于 f_1 时，谐波幅度以－20dB/10 倍频程速度下降；在 f_2 以上，谐波幅度以－40dB/10 倍频程的速度下降。在许多应用场合，后面的谐波很小，可以忽略，这样，系统带宽通常定义为 $1/\pi t_r$。例如，HCMOS 器件产生的外部周期信号的边沿时间为 2ns。产生谐波的带宽高达 160MHz。任何 PCB 印制线、元件连线、电缆或接插件都将成为天线，以不同的程度辐射谐波。系统的辐射可能为差模或共模辐射，共模辐射更难减小。

16.2 差模辐射

差模辐射由系统导体的 RF 电流环路引起。对于小的环路面积，电场为：

$$E = 2.6(AI_L f^2)/R(\mu V/m)$$

其中 A 为环路面积(cm^2)，I_L 为环路电流(A)，f 为频率(MHz)，R 为距离(m)。

对于恒定的电流和环路面积，给定距离处的电场与频率的平方成正比(即以 40dB/10 倍频程增加)。将这一项考虑进去，在 f_2 以内，Fourier 包络以 20dB/10 倍频程增加，频率大于 f_2 后变为平坦。为减小给定距离处的电场，应在设计 PCB 时减小 A 和 I_L。

16.3 共模辐射

共模辐射(CM)由电路中不希望有的电压下降造成，它使得某些接地部分的实际电平高

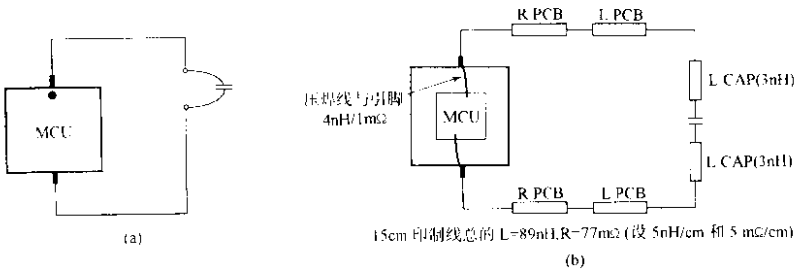


图 16-2 PCB 设计

(a)较差的电源去耦；(b)等效电路

于地电位，如图 16-2 所示。连接到受影响的地的电缆如同天线，辐射 CM 电势分量。其电场为：

$$E \approx (fI_{CM}L)/R \quad (V/m)$$

其中 L 为天线长度(m)，I_{CM}为共模电流(A)，f 为频率(Hz)，R 为距离(m)。

对于恒定的电流和天线长度，电场与频率成正比。考虑到这一点，频率低于 f_2 时，共模辐射是平坦的；频率高于 f2 时，CM 辐射为－20dB/10 倍频程。与差模辐射不同的是，CM 辐射更难以控制。显然为减小辐射，CM 电流应接近于零。

16.4 电源去耦

电源去耦不合理会减小系统抗干扰能力，并导致不正确、不可靠或不稳定的操作。例如，MC68HC1148 产生的瞬时电源电流峰值约 100mA。虽然平均电源电流只有数毫安，电源必须能够提供足够的瞬态电流，以保证操作正确。对于快速逻辑电路，若去耦设计较差，瞬时电源电流也足以产生 EMI 问题。

去耦网络用于减小电源阻抗。为确定去耦电容值，必须先确定可接受的电源波动。例如，对于 MC68HC11A8，$V_{DD}=5V$，无负载时：

$$V_{IL}-V_{OL}=0.2V_{DD}-0.1\approx 0.9V$$

$$V_{OH}-V_{IH}=4.9-0.7V_{DD}=1.4V$$

低电平噪声容限为 0.9V，最大波动 20%，则：

$$20\%\times 0.9V=180mV$$

瞬态时间为 10ns，则所需电容为：

$$\begin{aligned}C&=I_{DD}/(dV/dt)\\&=100mA/(180mV/10ns)\\&=0.006\mu F\end{aligned}$$

取最接近的电容值 0.01μF。当 M68HC11 工作在扩展模式时，总线操作产生的瞬态电流大得多，因此建议用一个 1μF 的钽电容与 0.01μF 的高频电容并联。在电路板的电源进入点要加一个 10～100μF 的电容。为避免电源噪声，去耦电容与电源连接处可加一个铁磁珠。

为使有效地滤波，网络与负载之间的阻抗必须很低；为减小 EMI，环路面积必须尽量地小。对于图 16-2，直流电压降为：

$$V_{drop}=77m\Omega\times 100mA=7.7mV$$

对于交流，总电感为 89nH，100mA 峰值电流的最小上升时间为 10ns，则

$$\begin{aligned}V_{drop}&=L\,di/dt\\&=89nH(100mA/10ns)\\&=890mV\end{aligned}$$

可见去耦网络与 MCU 之间的压降达 0.9V，考虑去耦的效果时，电感是最关键的因素。

采用厚的 PCB 印制线、地/电源平面、更短的路径可有效地减小寄生环路阻抗。去耦网络应该尽可能接近器件的电源引脚。表面封装电容的电感小，用表面封装电容可更有效地去耦。可将其安装在 PCB 非元件面的电源与地之间，这样引线距离最短。减小环路阻抗还应该减小环路面积。

16.5 连线端接

信号大约以 0.6 倍的光速（0.6ft/ns）在 PCB 印制线中传播。若连线未进行端接，则负载与传输线之间存在阻抗失配，造成信号反射。这种反射会引起减幅振荡和尖峰信号，产生明显的 EMI 问题。如果负载等于传输线的特征阻抗 Z_0，则负载就如同无传输线，没有反射。

在不匹配传输线情况，如果信号源的上升时间与传输线传播延迟时间相比足够慢，则反射将在信号上升时间内被信号源吸收。在所有其他情况下，连线应该当作传输线处理，并进行端接。通常，若线的单程传输延迟小于信号上升时间的 1/4，则不必进行端接。例如，HCMOS 器件的上升时间为 10ns，最大可不进行端接的连线长度为：

t delay<0.25×10ns−2.5ns

长度 length≤速度×t delay=0.6×2.5ft≈1.5ft

对于高速器件，如高速 TTL，上升时间为 3ns，则最大不端接的连线长度为 5.5in。

16.6 接地技术

地是一个系统内作为电位基准的点或平面。实际上，由于寄生电感和很大的地线电流不可避免地会造成电压下降，使接地处并非地电位，从而造成共模辐射。应仔细设计接地结构，减小环路面积，防止干扰弱信号电路。

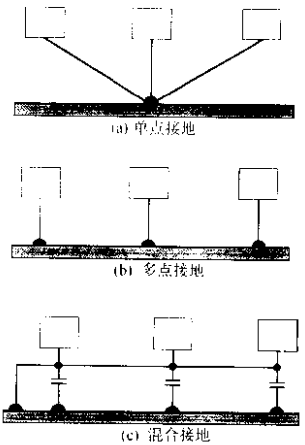


图 16-3 接地技术

信号接地分为三种，如图 16-3 所示。单点接地适于低频，应尽量缩短连线长度，以减小电感和辐射。多点接地应用于高频系统，如数字电路，其每个元件都连到其最近的低阻抗地平面。混合接地在低频类似于单点接地，在高频时类似于多点接地。通常的系统采用混合接地技术。

图 16-4 是典型的 MCU 应用系统接地结构，包括弱信号模拟电路、数字电路、I/O 缓冲器、大电流开关接地。电路板上的数字逻辑为多点接地。

16.7 模拟/数字混合系统

电路板上含有数字和模拟电路时，需要特殊考虑 PCB 的设计。图 16-5(a)是共模阻抗地耦合对模拟输入信号产生噪声的情况。例如，模拟部分为 12 位 A/D，则最低位数字量对应的模拟电压只有 1.25mV，故很容易受噪声影响。在图 16-5(a)中，模拟电路与有噪声的数字电路公用电源和地，而且 PCB 印制线很薄，也增加了寄生电感和直流电压降。较好的布线方法如图 16-5(b)所示，数字电源和地线都采用宽线条，模拟电路单独供电和单独接地。数字电路地线产生的压降不再会影响模拟电路，因为数字电路的电流不再经过模拟输入的环路。

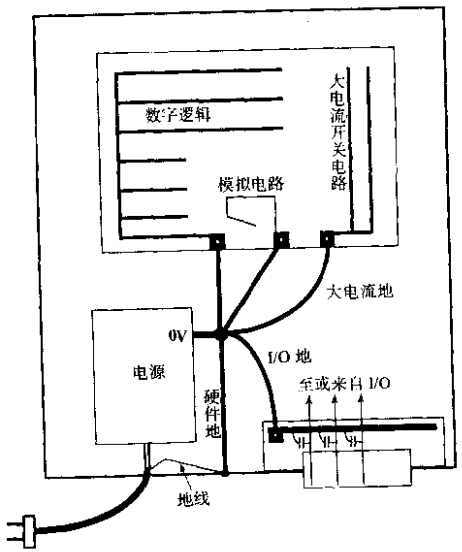


图 16-4　典型的 MCU 应用系统接地结构

适当的电源去耦也是减小模拟子系统噪声的必要方法。对于 MC68HC11 的片内 A/D，模拟输入推荐的去耦网络如图 16-6 所示。

16.8　PCB 设计指南

成功的 EMI 设计从良好的电路板设计开始。最关心的两个指标是信号路径电感和环路面积。电流返回通常上面平面导体(例如印制线)的电感为：

$$L = 2\ln(2\pi h/w) \quad nH/cm$$

其中 h 为电流返回通路上面的高度；w 为印制线宽度。

设 h＝1mm，w＝0.5mm，则 L＝5.1nH/cm。由于是对数关系，w 增加一倍不会使电感下降一半，但增加 w 仍是减小 L 的有效方法。例如，若 W 增大到 1mm，则 L＝3.7nH/cm；若 w＝5mm，则 L＝0.5nH/cm。

采用多层 PCB 可减小电感，但费用增加。推荐的方法是将电源和地平面放在外面，信号线夹在它们中间，这样具有屏蔽作用。为减小互扰，相邻层的信号应垂直布线。若不采用多层电路板，应将所有不用的区域做成地平面，以避免产生地回路，导致 EMI 问题。

通过去耦和仔细设计布线减小环路面积可减小射频干扰(RFI)。采用表面封装元件能减小环路面积。对于没有地平面的 PCB，信号线应该有尽可能接近的地返回通路，以使环路面积

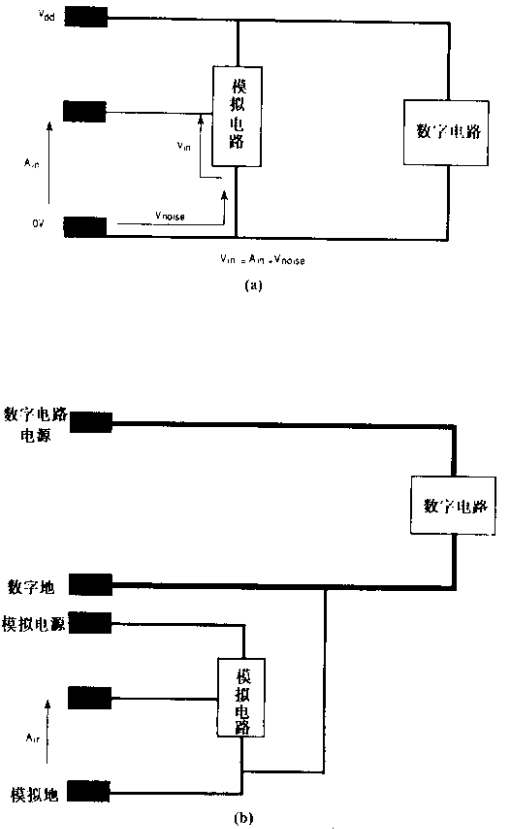


图 16-5　模拟电路的接地

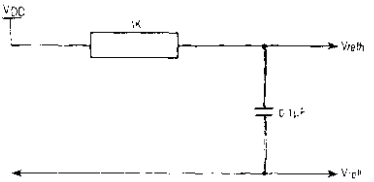
(a)较差的接地方法;(b)改进的接地方法

最小。对于数据/地址总线,应对每条数据/地址线安排一个与之相邻的地返回印制线,并且尽量短。对于地址总线,在相邻 A0 处(16 位总线时为 A1)安排一条地返回线,因为 A0(或 A1)的变化最频繁。

系统时钟通常是主要的辐射源。时钟元件应尽量集中,所有时钟线应尽量短,并应该有与之相邻的地线印制线或地平面。为避免互扰和辐射问题,时钟电路应该远离 I/O 信号线或电路,或进行屏蔽。例如,时钟和 I/O 缓冲器不应在一个芯片内。

另一个 RFI 源是 PCB 印制线方向的拐角,它如同阻抗不连续点,会产生辐射。对 HCMOS 电路,避免印制线 90°拐角方向变化是很重要的,如图 16-7 所示。

最后,HCMOS 器件不用的输入应进行端接,以减小 EMI 影响和避免 CMOS 电路处于线



Vrefl 不应低于 Vss -0.6V

图 16-6 MC68HC11 A/D 基准电压的去耦电路

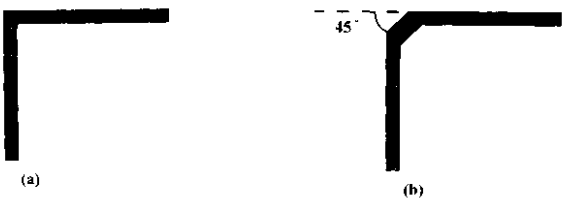


图 16-7 PCB 印制线的设计

(a)不合理的;(b)合理的

性区,产生明显的 DC 电流。

16.9 要点概述

解决差模辐射的方法:

1. 系统时钟通常是主要的辐射源。避免地回路和长印制线。可能的话,时钟线(或其他信号线)应有相邻的地返回印制线。使系统时钟所需的器件数最少,保证时钟电路和有关连线远离或屏蔽于 PCB 的 I/O 印制线和电路。不要用同一个封装壳内的驱动器作为 I/O、总线和时钟缓冲器,时钟单独使用一个缓冲器芯片。

2. 去耦电容尽量接近器件电源引脚,应采用一个大电容与一个或多个高频小电容并联的方法进行去耦。

3. PCB 电源去耦。

4. 对没有地平面的 PCB,每条地址/数据总线应有一条相邻的地返回印制线,并使其尽量短,以减小环路面积。

5. 避免地环路。由小间隙断开的环路在 RF 频率因电容的影响等于是闭合的,成为大的天线环路。大的地线环路除了辐射问题外,遭到外部 EMI 影响时使系统更容易失效。

6. 尽量增大电源和地印制线宽度,缩短其长度。

7. 不用的 HCMOS 输入端要进行端接。

8. 表面封装器件更有利于减小环路面积。

解决共模辐射的方法:

1. 增加电源和地宽度,减小共模压降。

2. 数字与 I/O(包括模拟部分)的地进行隔离,以减小 I/O 电缆辐射。在数字系统,屏蔽线的两端都应接到无噪声的地。

17. 用 MC68HC05B4 和 MC14489 测量和显示温度

本例是使用 MC68HC05B 系列来实现温度控制。程序设计部分包括查表线性插值运算，二进制码到 BCD 码的转变，摄氏温度到华氏温度的转变及一个实时时钟/计数部分。用 HC05B4 的 SCI 驱动一个多字节 LED 显示驱动芯片完成温度显示功能。热敏传感器是一个热敏电阻，将温度转换成电压输入到 HC05B4 的 A/D 转换口得到数字电压以便处理。这种电路元件少，易于与 MCU 接口。

17.1 温度测量

热传感器就是一个热敏电阻，它的可用温度范围是－40℃～80℃。阻抗温度特性如图17-1所示。为了获得最高的 A/D 转换精度，应尽量利用 V_{RH}～V_{RL}的范围。

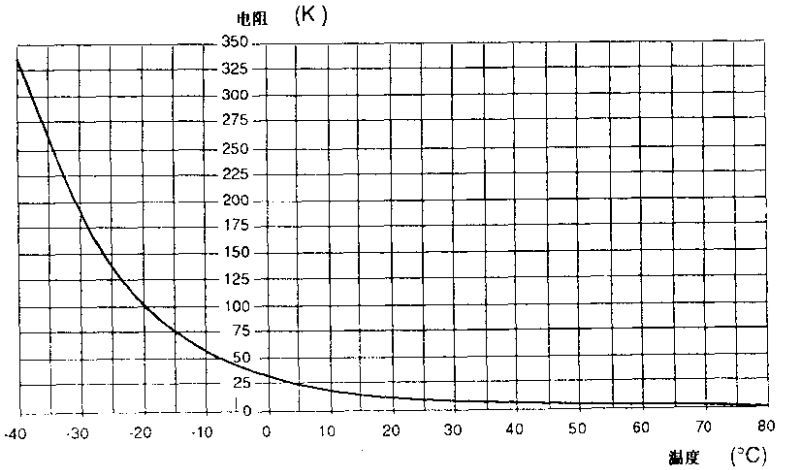


图 17-1 热敏电阻的阻值与温度的关系

在本例中，A/D 转换器的 $V_{RH}=V_{dd}$，$V_{RL}=V_{ss}$，热敏电阻与 20kΩ 的精确电阻组成一个分压器，信号电压范围大约是 0.3V～4.7V(对应温度变化范围－40℃～80℃)。输入模拟电压与温度的关系曲线如图 17-2 所示。

从图中可以看出，上述曲线是非线性的，所以可采用最简单的查表法从电压值求出相应的温度值，达到所需精度，可将一系列温度值存储到 ROM 中的一个表中，那么给定任意一个电压值即可在相邻两点经线性插值得出对应温度值。存入表中点的多少决定了温度值的精度。本例中选择了 16 个点。因为 A/D 转换结果的范围是 0～255，因此每两个点的间隔是 16。表17-1即是构成的表的数据项及相应的温度值。

图 17-3 是由 16 个点经过线性插值得到的 A/D 输入与温度的关系，可以看出它与图 17-2

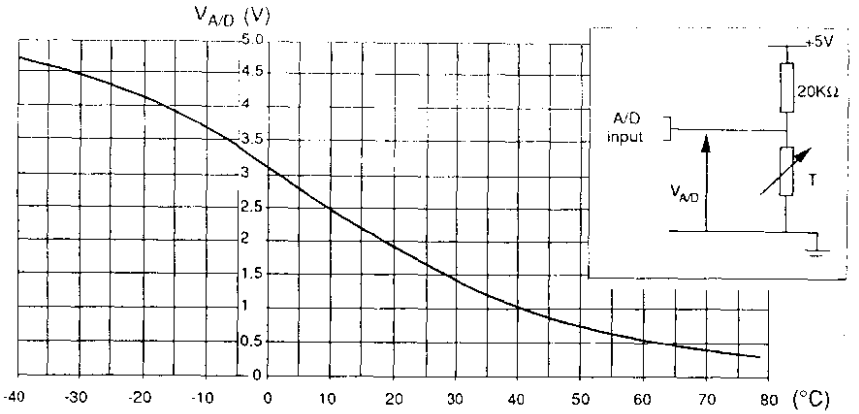


图 17-2　A/D 输入模拟电压与温度的关系

注：框内为测量电路

相差不大。

表 17-1　A/D 转换结果与其对应的温度

A/D 结果	A/D(伏)	温度(℃)	温度(℃,补码)
0	0	—	—
16	0.31	79	4F
32	0.63	56	38
48	0.94	43	2B
64	1.26	34	22
80	1.57	27	1B
96	1.88	21	15
112	2.20	15	0F
128	2.51	10	0A
144	2.82	5	05
160	3.14	−1	FF
176	3.45	−6	FA
192	3.77	−11	F5
208	4.08	−18	EE
224	4.39	−26	E6
240	4.71	−40	D8
255	5.0	—	—

温度显示每 1 秒钟更新一次，软件实现很简单。对于 4.1934MHz 的晶振，设置定时器每 128ms 溢出一次。定时器溢出中断处理程序更新系统实时时钟的 TICKS、SECS、MINS 和 HRS 单元，每 1 秒间隔置位 SEC 标志。

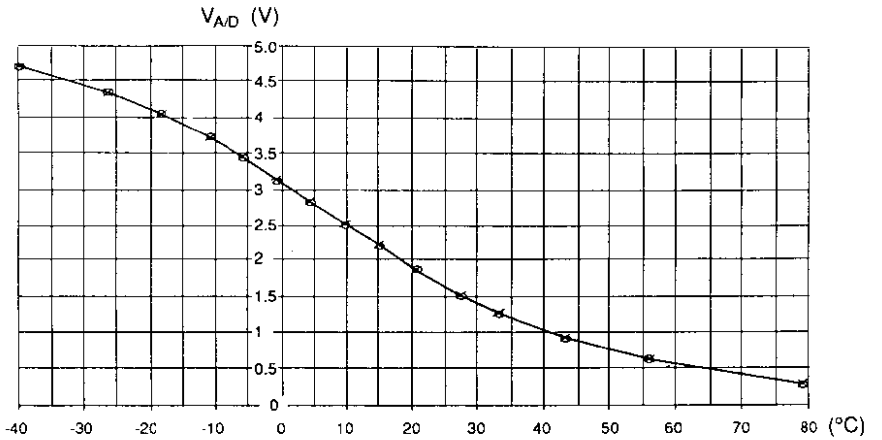


图 17·3　插值 A/D 输入电压与温度的关系

主程序每秒执行一次(通过 SEC 标志),在检查摄氏/华氏选择开关后,即由子程序 ADCONV 测量温度值。这个子程序先检查摄氏/华氏开关,然后相应设置 A/D 控制寄存器。在选定的通道上,依次完成 4 次 A/D 转换,结果相加存入累加器并转存到 TEMP。结果经 2 次右移完成除 4 运算,即得到 A/D 转换的平均值。取平均值可减小 A/D 输入噪声的影响。转换值的高 4 位用来查表获得基本温度值。如果超过温度界限,则在退出该子程序之前置 TLIMIT 标志。

温度表中的每一项都是 2 的补码。因此正数和负数可同样操作。插值的具体做法就是求出基本温度点与下一个点的差值,乘以转换值的低四位再除以 16,结果做为修正值。用基本温度值减去修正值即得相应温度值,并存入 NEWTMP 单元中。之所以减去修正值而不是加,是因为热敏电阻具有负的温度系数,所以 A/D 转换值的增加对应于温度的减少。

如果选择华氏温度,则在更新温度显示之前,还要先转向 CTOP 子程序完成摄氏温度值到华氏温度值的转换。

摄氏温度转换为华氏温度,公式是 $F=C\times1.8+32$。首先将摄氏温度的符号存入到 NEGNUM 标志单元中,在乘以 1.8 之前,先检查温度是否超出最高温度上限。乘以 1.8 可以先乘以 115 再除以 64 来完成。恢复结果的符号然后再加上 32,得到补码形式的华氏温度值。

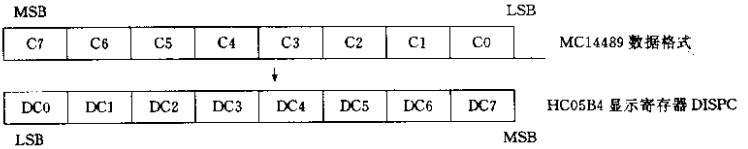
17.2　温度显示

MC14489 多字符显示驱动器可以显示"度"并且接口简单,几乎不需其它外部器件。如果要求显示的数字增多,还可以多个 MC14489 级联。一般 MC14489 由 MCU 的 SPI 口驱动。因为 68HC05B 系列没有 SPI 功能,这里由 SCI 串行时钟输出驱动也能满足要求。

在温度值写入到显示驱动器之前,必须先转换成正确的数据格式。

第 1 步是将二进制补码转换成 BCD 码。可从 SETDISP 中调用子程序 CONBCD 完成。温度符号在调用 SETDISP 之前先存入 NEGNUM 中。在检查温度是否越界后,调用 SETDISP 得到 BCD 码,存入到 DEC 0～2 中。

SETDISP 的另一部分用于设置显示寄存器 DLSP 1、2、3 和显示控制寄存器 DISPC。MC14489 的数据格式是最高位在前，而 68HC05B4 SCI 首先发送最低位，因此串行数据发送前，应做相应处理。这在设计 HC05B4 的产生特殊字符的程序时容易造成混淆。下面是 MC14489 的数据格式和对应的 HC05B4 的 DISP 1～3 和 DISPC 的各位位置：



MC14489 与 68HC05B4 的显示寄存器的数据格式如图 17-4 所示。

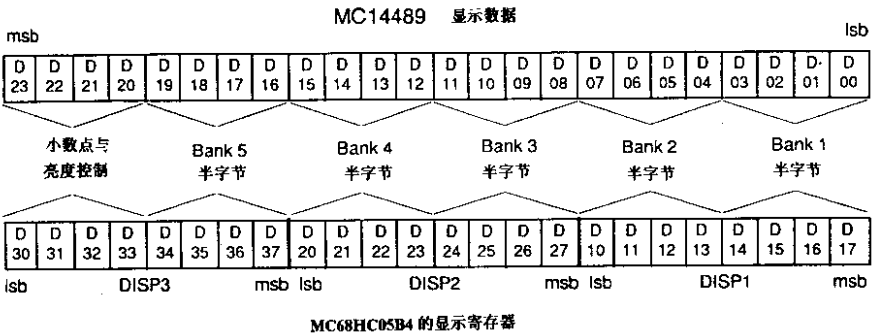


图 17-4　MC14489 与 HC05B4 显示寄存器的数据格式

主程序调用子程序 DISPL，通过 SCI 将显示寄存器的内容发送到 MC14489。MC14489 时钟最大可达 4MHz（电源 $V_{CC}=5V$），因此 SCI 最大发送波特率为 131.072kHz（晶振为 4.19304MHz）。仅当数据与以前有变化时，显示数据才会发送。如果有数据变化，则数据寄存器由 DISP 3 开始依次发送。发送完最后一个字节后，在返回主程序之前，先禁止 SCI 和 MC14489。

主程序调用的最后一个子程序是 MC14489 设置更新子程序 DISCON。它的操作方式与 DISPL 类似，它在发送之前，检查有无设置数据的变化。

17.3　硬件电路

该例的硬件电路很简单，如图 17-5 所示。只需要两个引脚读取开关的状态和一个引脚控制 MC14489。由于 SCI 禁止时其时钟端和数据输出端为高阻态，因此这两个引脚外接两个下拉电阻。显示 LED 为共阴极，只加一个外部电阻就可设置显示的亮度。光敏电阻 R12 用来控制显示亮度，以适应各种背景光情况。R12 的电阻随着亮度的增强而减小，R11 必须保证在很强的背景条件下 MC14489 不超过规定的最大电流值。R13 应保证在很暗的背景下，LED 仍有足够的驱动电流。

17.4　程序清单

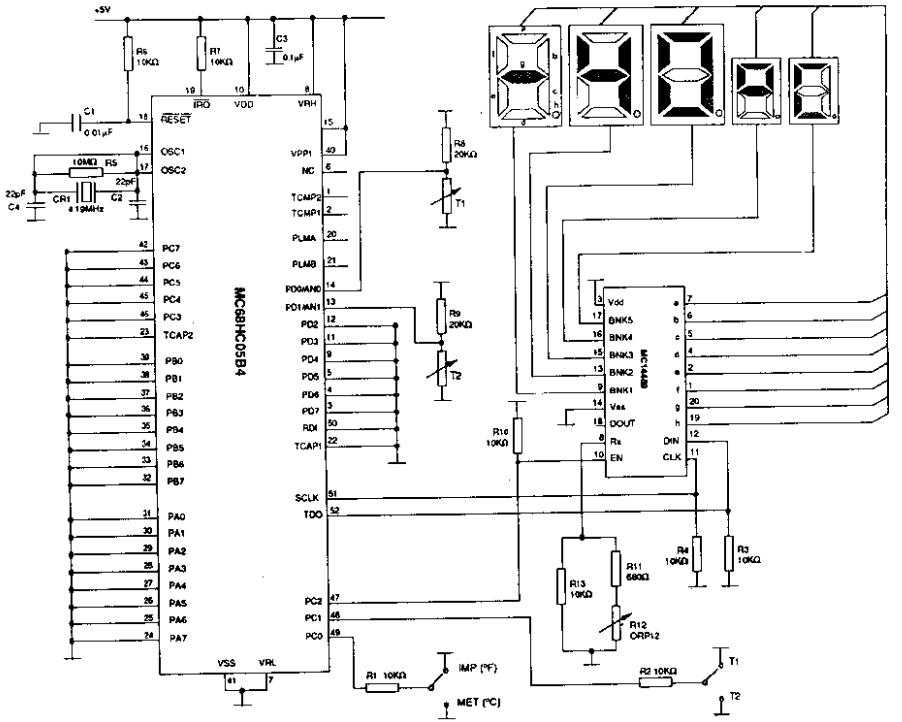


图 17-5　测温系统硬件电路图

```
* * * * * 定义 I/O 寄存器 * * * * *
PORTA       EQU     $ 00        A 口数据寄存器
PORTB       EQU     $ 01        B 口数据寄存器
PORTC       EQU     $ 02        C 口数据寄存器
PORTD       EQU     $ 03        D 口数据寄存器
DDRA        EQU     $ 04        A 口数据方向寄存器
DDRB        EQU     $ 05        B 口数据方向寄存器
DDRC        EQU     $ 06        C 口数据方向寄存器
* 定义 A/D 寄存器和标志位
ADDATA      EQU     $ 08        A/D 数据寄存器
ADSTCT      EQU     $ 09        A/D 状态和控制寄存器
COCO        EQU     7           转换完成标志位
* 定义 SCI 寄存器
BAUD        EQU     $ 0D        SCI 波特率寄存器
SCCR1       EQU     $ 0E        SCI 控制寄存器 1
```

```
SCCR2       EQU     $0F             SCI 控制寄存器 2
SCSR        EQU     $10             SCI 状态寄存器
TDRE        EQU     7
TC          EQU     6
SCDTA       EQU     $11             SCI 数据寄存器
* 定义定时器寄存器
TCR         EQU     $12             定时器控制寄存器
TOIE        EQU     5               定时器溢出中断允许位
OCIE        EQU     6               定时器输出比较中断允许位
ICIE        EQU     7               定时器输入捕捉中断允许位
TSR         EQU     $13             定时器状态寄存器
OCF2        EQU     3               定时器输出比较 2 标志位
ICF2        EQU     4               定时器输入捕捉 2 标志位
TOF         EQU     5               定时器溢出标志位
OCF1        EQU     6               定时器输出比较 1 标志位
ICF1        EQU     7               定时器输入捕捉 1 标志位
TIC1HI      EQU     $14             定时器输入捕捉 1 寄存器(16 位)
TIC1LO      EQU     $15
TOC1HI      EQU     $16             定时器输出比较 1 寄存器(16 位)
TOC1LO      EQU     $17
TIMHI       EQU     $18             定时器自由运行计数器(16 位)
TIMLO       EQU     $19
TIMAHI      EQU     $1A             定时器第二计数器(16 位)
TIMALO      EQU     $1B
TIC2HI      EQU     $1C             定时器输入捕捉 2 寄存器(16 位)
TIC2LO      EQU     $1D
TOC2HI      EQU     $1E             定时器输出比较 2 寄存器(16 位)
TOC2LO      EQU     $1F
* 定义存储器映射地址
TEST        EQU     $20             测试寄存器
ROM0        EQU     $0020           ROM0 起始地址
RAM         EQU     $0050           RAM 起始地址
UROM        EQU     $0F00           主用户 ROM 起始地址
* 定义 RAM 变量
            ORG     $0050
TICKS       RMB     1
SECS        RMB     1
MINS        RMB     1
HRS         RMB     1
FLAG        RMB     1
OVERFL      EQU     0
NEGNUM      EQU     1
TLIMIT      EQU     2
SEC         EQU     3
MODE        RMB     1
IMP         EQU     0
```

```
BIN0        RMB     1
DEC2        RMB     1
DEC1        RMB     1
DEC0        RMB     1
NEWTMP      RMB     1
TEMP        RMB     1
TEMP1       RMB     1
TEMP2       RMB     1
DESP1       RMB     1
DISP1       RMB     1
DISP2       RMB     1
DISP3       RMB     1
DISPC       RMB     1
OLDD1       RMB     1
OLDD2       RMB     1
OLDD3       RMB     1
OLDDC       RMB     1
            ORG     $20
ADTAB       FCB     $00,$4F,$38,$2B,$22,$1B,$15,$0F
            FCB     $0A,$05,$FF,$FA,$F5,$EE,$E6,$D8
*****初始化和主程序*****
            ORG     $F00
RESET       EQU     *
            LDA     #$0             端口初始化
            STA     PORTA
            STA     PORTB
            STA     DDRA
            STA     DDRB
            LDX     #OLDDC          所有用到的 RAM 地址初始化
INIRAM      STA     ,X              RAM 内容初始化
            DECX
            CPX     #RAM
            BNE     INIRAM
            LDA     #$04            PC2 输出高电平
            STA     PORTC
            LDA     #$04
            STA     DDRC
TIMINT      LDA     TSR
            LDA     TIMLO
            LDA     #$20
            STA     TCR
            CLI                     允许中断
*循环主程序
MAINUP      EQU     *
            BRCLR   SEC,FLAG,MAINUP
            BCLR    SEC,FLAG
```

```
        BCLR    IMP,MODE
        BRCLR   0,PORTC,NOIMPP      是C/F制来表示温度?
        BSET    IMP,MODE            如是F制,设置显示标志
NOIMP   BCLR    NEGNUM,FLAG         温度为正标志
        JSR     ADCONV              转到温度测量
        JSR     ADCONV              转到温度测量
        BRCLR   IMP,MODE,GOMETR     是否要转化成F制
        JSR     CTOF                是,转相应子程序
COMETR  LDA     NEWTMP
        BPL     GOMORE              温度值为正?
        NEGA                        否,取绝对值
        BSET    NEGNUM,FLAG         并设置符号标志
GOMORE  STA     BIN0
GODISP  JSR     SETDISP             设置显示字节
        JSR     DISPL               需要的话修改显示
        JSR     DISCON              需要的话更新14489设置
        BRA     MAINLUP
* * *CTOF子程序——将摄氏温度值转化为华氏温度
CTOF    EQU     *
        LDA     NEWTMP
        BPL     NONEC               温度值为正?
        BSET    NEGNUM,FLAG         否,设置负号标志
        NEGA                        取温度的绝对值
        BRA     MULIP8
NONEG   CMP     #53                 温度是否超过最高值127F?
        BLO     MULIPP8             否。转到两种表示转换
        BSET    TLIMIT,FLAG         是。设置出界标志
        RTS
MULIP8  LDX     #115                下面是转换运算,乘以
        MUL                         115除以64
        RORX                        (相当于乘以1.8)
        RORA
        RORX
        RORA
        RORX
        RORA
        RORX
        RORA
        RORX
        RORA
        RORX
        RORA
        BRCLR   NEGNUM,FLAG,        温度值是否为负?
                NONEG1
        NEGA                        是。用补码表示。
NONEC1  BCLR    NEGNUM,FLAG         否。加上32,完成
```

```
        ADD     #32             转化为F制温度值
        STA     NEWTMP          存入NEWTMP
        RTS
* * * SETDISP 子程序——用 DCD 码设置显示寄存器
SETDISP EQU     *
        BRSET   TLIMIT,FLAG,FORCE   温度出界吗?
        LDX     #$8             否
        JSR     CONBCD          将8位数变成3字节BCD码
        LDX     #4
        CLRA
LUPDIS1 LSR     DEC1            交换位次序,以使得SCI的LSB在前与
        ROLA                    MC14489的MSB在前二者相容
        DECX
        BNE     LUPDIS1
        LDX     DEC2
        BEQ     TSTNEG
        ORA     #$80
        BRA     STD1
TSTNEG  BRCLR   NEGNUM,FLAG,STD1    温度为负吗?
        ORA     #$B0            是。加一负号
STD1    STA     DISP1           存入第一显示寄存器
        LDX     #8
        CLRA
LUPDIS2 LSR     DEC0
        ROLA
        DECX
LUPDIS2 BNE     LUPDIS2
        ORA     #$0F
        STA     DISP2           存入到第二显示寄存器
        LDA     #$31
        BRCLR   IMP,MODE,STDIS3     是F/C制显示?
        LDA     #$F1            F1,改变A中的值
STDIS3  STA     DISP3           存入
        LDA     #$CB
        LDX     DEC2
        BEQ     STDISC
        LDA     #$8B
STDISC  STA     DISPC
        RTS
FORCE   LDA     #$BB            强制显示一℃或一℉
        STA     DISP1
        LDA     #$BF
        STA     DISP2
        LDA     #$31
        BRCLR   IMP,MODE,STDI3
        LDA     #$F1
```

```
STD13       STA     DISP3
            LDA     #$FB
            STA     DISPC
            RTS
* * *TOVINT 中断服务程序
TOVINT      BRCLR   TOF,TSR,NOOVF       计数器发生溢出了吗?
            INC     TICKS
            LDA     TICKS
            CMP     #8
            BLO     NOINC
            CLR     TICKS
            INC     SECS
            BSET    SEC,FLAG
            LDA     SECS
            CMP     #60
            BLO     NOINC
            CLR     SECS
            INC     MINS
            LDA     MINS
            CMP     #60
            BLO     NOINC
            CLR     MINS
            LDA     HRS
            CMP     #$FF
            BEQ     NOINC
            INC     HRS
NOINC       LDA     TIMLO
NOOVF       RTI
* * *ADCONV 子程序——A/D 转化线性插值查表
ADCONV      EQU     *
            BCLR    TLIMIT,FLAG
            CLR     TEMP
            BRSET   1,PORTC,CONT1       热敏电阻开关接 T1/T2?
            LDA     #$21
            BRA     SETAD
CONT1       LDA     #$20
SETAD       STA     ADSTCT              开始转换
            LDX     #4                  转换 4 次,取平均值
            CLRA
ADLUP1      BRCLR   COCO,ADSTCT,ADLUP1
            ADD     ADDATA
            BCC     DECCX
            INC     TEMP
DECCX       DECX
            BNE     ADLUP1
            ROR     TEMP
```

```
            RORA
            ROR     TEMP
            RORA
            STA     TEMP            A/D 结果存入 TEMP
TABL        LSRA                    考虑 A/D 值的前面位
            LSRA
            LSRA
            LSRA
            TAX
            LDA     ADTAB,X         与表内数据比较
            BEQ     TRANGE
            CMP     # $ D8          与表内数据边界比较
            BEQ     TRANGE
            STA     TEMP1           存入基点值
            INCX
            SUB     ADTAB,X         与下一项相减
            STA     TEMP2           存入差值
            LDA     TEMP            取出 A/D 转换数值
            AND     # $ 0F          屏蔽高 4 位
            LDX     TEMP2
            MUL                     与差值相乘
            RORA                    除以 16
            RORX
            RORA
            RORX
            RORA
            RORX
            RORA
            RORX
            STX     TEMP2           得到修正值
            LDA     TEMP1           基点值与修正值相减
            SUB     TEMP2
            STA     NEWTMP          存入新的温度值
            RTS
TRANGE      BSET    TLIMIT,FLAG
            RTS
* * * CONBCD 子程序——将二进制转化成 3 字节 BCD 码
CONBCD      EQU     *
            CLRA                    将 3 字节 BCD 码内容清零
            STA     DEC0
            STA     DEC1
            STA     DEC2
LUPBCD      ROL     BIN0
            ROL     DEC0
            ROL     DEC1
            ROL     DEC2
```

```
        LDA     DEC0
        SUB     #$0A
        BMI     TSTD1           检查BCD字节是否溢出
        STA     DEC0
        INC     DEC1
TSTD1   LDA     DEC1
        SUB     #$0A
        BMI     TSTD2
        STA     DEC1
        INC     DEC2
TSTD2   LDA     DEC2
        SUB     #$0A
        BMI     NOOVR
        LDA     #9
        STA     DEC2            BCD数已溢出,置标志位,并将高位数字置9
        BSET    OVERFL,FLAG
NOOVR   DECX
        BNE     LUPBCD
        RTS
***DISPL子程序——通过SCI更新14489显示寄存器的内容。新的温度值是否与上一次测量值相同?
DISPL   EQU     *
        LDA     DISP1
        CMP     DLDD1
        BNE     UPDATE
        LDA     DISP2
        CMP     OLDD2
        BNE     UPDATE
        LDA     DISP3
        CMP     OLDD3
        BNE     UPDATE
        RTS
UPDATE  LDA     #$01            不同。设置SCI控制寄存器
        STA     SCCR1
        DECA
        STA     BAUD            设置波特率
        LDA     #$08
        STA     SCCR2           发送允许
PREAM   BRCLR   TC,SCSR,PREAM
        BCLR    2,PORTC         允许向14489发送
        LDA     DISP3           开始发送第一字节
        STA     DLDD3
        STA     SCDAT
DWAIT1  BRCLR   TDRE,SCSR,DWAIT1 等待结束
        LDA     DISP2           开始发送第二字节
        STA     OLDD2
        STA     SCDAT
```

```
DWAIT2    BRCLR   TDRE,SCSR,DWAITW    等待结束
          LDA     DISP1               开始发送第三字节
          STA     OLDD1
          STA     SCDAT
DWAIT3    BRCLR   TC,SCSR,DWAIT3      等待结束
          LDA     #$00
          STA     SCCR2               禁止 SCI 发送
          BSET    2,PORTC             14489 使能端无效
          RTS
***DISCON 子程序——通过 SCI 更新 14489 设置寄存器
DISCON    EQU     *
          LDA     DISPC               设置寄存器内容是否与上
          CMP     OLDDC               次数值一致
          BNE     UPDCON
          RTS
UPDCON    LDA     #$01                否
          STA     SCCR1               设置 SCI 控制寄存器
          DECA
          STA     BAUD                设置波特率。4.19MHz 晶体为 131.072kHz
          LDA     #$08
          STA     SCCR2               允许 SCI
PREAM1    BRCLR   TC,SCSR,PREAM1      等待导引码结束
DOCONF    BCLR    2,PORTC             14489 使能信号有效
          LDA     DISPC               发送
          STA     OLDDC
          STA     SCDAT
DWAIT4    BRCLR   TDRE,SCSR,DWAIT4
          LDA     #$00                禁止 SCI
          STA     SCCR2
DWAIT5    BRCLR   TC,SCCR,DWAIT5
          BSET    2,PORTC             14489 使能无效
          RTS
*****向量地址表*****
          ORG     $1FF2
SCIINT    FDB     RESET
TOVFLW    FDB     TOVINT
TOCMP     FDB     RESET
TICAP     FDB     RESET
EXTINT    FDB     RESET
SOFT1     FDB     RESET
POR       FDB     RESET
```

18. MC68HC05T1 的屏幕显示(OSD)在 TV 中的应用

MC68HC05 系列中的 T 型号是 TV 和 VCR 增加屏幕显示功能的方便而价廉的有效方法。HC05T1 具有 8K ROM、320 字节 RAM、16 位定时器和 8 路 PWM D/A 转换器和 OSD 功能。有关 HC05T1 的 OSD 主要性能详见有关手册,这里主要讨论 OSD 的软件设计。

编写 OSD 软件有多种方法,需根据所用的 RAM 和 ROM 容量而定。一种方法是对每一种显示行都有一个独立的中断程序。这种方法所用的 RAM 少,但不适于 ROM。另一种方法是编写一个中断程序,每当需要显示新的一行时,中断程序将普通 RAM 中的显示信息传输到显示 RAM 中。所用的 RAM 容量取决于某个时刻所需显示的最大数据。选取哪一种方法取决于所要显示的数据类型。前一种方法适用于显示数据为固定的场合。后一种方法适用于显示数据为变量的场合。

本例采用后一种方法编程,RAM 用来复制所有被显示的数据。可以改变 RAM 的容量以改变显示的行数和每行的字符数。本例为 8 行,每行 16 个字符,该软件允许使用 10 行中的任意 8 行,但只能使用每行 18 个字符的前 16 个。在软件控制下显示行可移到右边。利用页 1 RAM 不会增加太多的执行时间,但页 0 RAM 可作为其它控制软件。这样可以采用直接寻址和位处理指令。

在页 1 可利用 1 字节变址寻址方式,可访问的地址达 $1FE。例如,本例中 OSDCLR 程序用于将 OSD 所用的 RAM 初始化,它采用 CLR DRAM-1,X 指令。

18.1 中断程序

每当发生中断时,OSD 更新中断程序(NLINE)将页 1 RAM 数据传输到显示 RAM。首先将选择下一行行号的指令加 1,该指针(OSDL)之后将适当的数据从页 1 RAM 传输到 OSD RAM。因此,任何行号可作为指针选择每一种显示类型。下一个行号写入 $34 寄存器,该行号与 OSD 硬件屏面的目前位置相比较,当二者相符时,产生中断,并执行下一个中断程序。

中断程序将有关的 OSD 数据从页 1 RAM 传输到 OSD 数据寄存器。采用重复程序,每个字节传输需 8 个周期(4μs),中断程序所花时间最少。若利用循环,可减少 ROM,但每个字节需用 28 个周期。实际中应在时间和 ROM 容量之间进行权衡。本例的中断程序所花时间为 121±4μs。采用循环传输 OSD 数据的方法(TOSD2)将增加 165μs(后面程序清单中作为说明列出)。

18.2 主程序

OSD 控制程序并不直接写入大多数显示寄存器。它将所需的显示和控制信息写入 RAM,并提供需要闪烁或窗口位置。必须写入显示控制寄存器($33、$35、$37)。主程序分 4 部分:空闲状态(IDLE)、通道名表、程序/通道号和模拟显示。

TV 应用中的 MCU 通常需要处理红外接收(IR)命令。本例的 OSD 中断程序编写时尽量减小执行时间。当作为 IR 接收端的 TACP 发生下降沿时,定时器的值被保存。该例中 OSD 软件所用的协议(发送芯片为 MC144105)中,边沿之间最小间隔为 1ms。

18.3 该例中 OSD 程序的特点

当按下 0～9 键或 PC－/PC＋键时，在屏幕的右下角出现新的节目号，字符的高度/宽度加倍，并且保持 5 秒。其上面显示频道名(若已定义)或频道号(正常大小)。5 秒之后，该显示取消，将无显示或显示正常字体的节目号。

对于大于或等于 10 的节目号，需按 3 个键。首先按"－"，显示两个短线，第一次按的 0～9 数字键(只有 0～4 有效)为十位数字，第二次按的数字为个位。如果在 30 秒内不选择节目号，TV 恢复以前的显示。

当按下 P/C 键时，节目号和频道名(或频道号)显示 5 秒，指明目前状态。若这期间再次按下该键，则 TV 改变为频道模式，并在后一次按键后保持 30 秒。显示(黄色)表明节目模式的节目号和频道号。频道号闪烁表示：如果数字、PC＋或 PC－键按下，则频道号改变，可选择新的频道。若按 STORE 键，则目前频道存储在当前节目号上。若 30 秒内无键按下，则 TV 恢复节目模式。若已改变频道，但没有按 STORE 键，则 TV 恢复为以前调谐的存储在目前节目号上的频道。

18.3.1 自动搜索

当按下 SEARCH 时，TV 进入频道模式，并且 OSD 显示如前所述。频道号以每秒 2 个频道的速度递增，直到发现信号为止，之后停止搜索。按 STORE 键使 TV 恢复节目模式，并将该频道存储在当前节目号中。

18.3.2 模拟显示

当选中模拟显示时，将显示适当的长方块和水平横线表明目前 D/A 转换器的值(满量程 63)。5 秒后恢复原状态(无或节目号)。当使用 ANALOGUE ＋/－键时，若未选中模拟显示，则显示音量(并可调整)。

18.3.3 频道名表格

多达 24 个频道可有 4 个字符名。频道号以及其它信息与目前节目号相对应。当选中某节目号时，或按 P/C 时，则显示频道名和频道号。当按 INDEX 键时，显示具有 6 行的表格。每行含有频道号、制式和频道名，所有这些数据都可由用户定义。

屏幕上某字符显示闪烁表示光标的当前位置。光标位置的字符可改变为 0～9、A～Z 或空格(按 PC－或 PC－－)，0～9 为频道号、PAL/SECAM 为制式。当改变字符(或制式)时，其颜色由黄变红。用 RED 和 GREEN 键可使光标左右移动，用 BLUE 和 YELLOW 可使光标上下移动。当前行出现在浅蓝窗口，其它行为深蓝窗口。当光标需要超出最下一行或最上面一行时，整个显示窗口上/下卷动。

STORE 键用于存储。它将目前的频道名和制式存储到频道号上。这时任何改变过的字符恢复为黄色。被存储的这一行之外的任何行中字符的改动无效。频道 00 不能有频道名。从表中去除一个频道名的步骤是：将该频道号置为 0，然后存储这一行。按 Teletext INDEX 键则退出表格显示。每个键的功能显示在最下一行。

18.4 程序清单

程序如下：

```
* * * * * 定义 TELETEXT RAM 变量 * * * * *
SUB1            RMB     1
```

```
R1          RMB     1
R2          RMB     1
R3          RMB     1
C1          RMB     1
C2          RMB     1
C3          RMB     1
C4          RMB     1
C5          RMB     1
C6          RMB     1
SUB2        RMB     1
R4          RMB     1
R5          RMB     1
R6          RMB     1
R7          RMB     1
SUB3        RMB     1
R8          RMB     1
R9          RMB     1
R10         RMB     1
R11         RMB     1
PH          RMB     1
PT          RMB     1
PU          RMB     1
LIFO        RMB     9
PAGE        RMB     7
PAG0        RMB     3
PAG1        RMB     3
PAG2        RMB     3
PAG3        RMB     3
PAGC        RMB     3
PAG1        RMB     3
PDP         RMB     1
ACC         RMB     4
WACC        RMB     1
ADDR        RMB     1
DPNT        RMB     1
SUBADR      RMB     1
IOBUF       RMB     4
STAT2       RMB     1
LINKC       RMB     1
STAT3       RMB     1
***** 定义一般的 RAM 变量 *****
PLLHI       RMB     1
PLLOW       RMB     1
W1          RMB     1
W2          RMB     1
W3          RMB     1
```

方式寄存器
页请求地址寄存器

显示寄存器
显示控制寄存器(NORMAL)
显示控制寄存器(News/sub)
显示方式寄存器

有效寄存器
有效行寄存器
有效列寄存器
有效数据寄存器
第二数据寄存器
第三数据寄存器
第四数据寄存器

页号输入缓冲器
AC0 页号
AC1 页号
AC2 页号
AC3 页号
浅蓝页号
变址页号
页数字指针

工作 ACC 号
IIC 地址
IIC 写数据指针(写入)

IIC 缓冲,后两字节保留用于 PLL

连接方式

PLL 除数高位
PLL 除数低位

```
COUNT       RMB   1          循环计数器
KOUNT       RMB   1          局部键盘计数器
CNT         RMB   1          12.8ms(向上,自由运行)
CNT1        RMB   1          12.8ms(向上,每一秒复位)
 *CNT2      RMB   1          3.25s(向上)
CNT3        RMB   1          3.25s(向下)
CNT4        RMB   1          12.8ms(清除 row 24 延时)
CNT5        RMB   1          12.8ms(向上)
TMR         RMB   1          瞬时显示秒计数器
STAT        RMB   1          0:电视/TELETEXT
 *                           1:ICC R/W
 *                           2:保持
 *                           3:IR 重复禁止
 *                           4:瞬时显示开启
 *                           5:时间保持
 *                           6:子页模式
 *                           7:IR 任务暂停
STAT4       RMB   1
PWR         RMB   1          复位为 $55,通常为 $AA
PROG        RMB   1          当前节目号
CHAN        RMB   1          当前频道号
DISP        RMB   1          当前显示号
FTUNE       RMB   1          调谐寄存器
AVOL        RMB   1          音量
KEY         RMB   1          按键代码
NUMO        RMB   4          LED 显示 RAM
IRRA1       RMB   1          IR 中断 TEMP
IRRA2       RMB   1          IR 中断 TEMP
IRRA3       RMB   1          IR 中断 TEMP
IRRA4       RMB   1          IR 中断 TEMP
DIFFH       RMB   1          IR 时间差
DIFFL       RMB   1
IRH         RMB   1          IR 代码位
IRL         RMB   1          收集
IRCODE      RMB   1
IRCNT       RMB   1
IRCMCT      RMB   1
OLDIR       RMB   1
*****定义立体声 RAM 变量*****
POLLTM      RMB   1
TONEA       RMB   1          调谐
LBAL        RMB   1          扬声器平衡变量
LVL         RMB   1          扬声器左声道音量
LVR         RMB   1          扬声器右声道音量
HVL         RMB   1          耳机左声道音量
HVR         RMB   1          耳机右声道音量
```

```
TONE      RMB   1          音调变量
MATRIX    RMB   1
MATNO     RMB   1          当前模式
WS1       RMB   1
WS2       RMB   1
VAV       RMB   1
MONCNT    RMB   1
STECNT    RMB   1
DULCNT    RMB   1
ERRCNT    RMB   1
RCOUNT    RMB   1
RANGE     RMB   1
TEMP      RMB   1
STAT5     RMB   1
STAT6     RMB   1          2 位节目输入
TMPRC     RMB   1          暂时节目号
*****定义 OSD RAM 变量*****
CAS1      RMB   1          第一行,颜色 1/2 及允许轮廓线
RAD1      RMB   1          行地址和字符大小
CCR1      RMB   1          窗口颜色及最后一列
CAS2      RMB   1          第二行,颜色 1/2 及允许轮廓线
RAD2      RMB   1          行地址和字符大小
CCR2      RMB   1          窗口颜色及最后一列
CAS3      RMB   1          第三行,颜色 1/2 及允许轮廓线
RAD3      RMB   1          行地址和字符大小
CCR3      RMB   1          窗口颜色及最后一列
CAS4      RMB   1          第四行,颜色 1/2 及允许轮廓线
RAD4      RMB   1          行地址和字符大小
CCR4      RMB   1          窗口颜色及最后一列
CAS5      RMB   1          第五行,颜色 1/2 及允许轮廓线
RAD5      RMB   1          行地址和字符大小
CCR5      RMB   1          窗口颜色及最后一列
CAS6      RMB   1          第六行,颜色 1/2 及允许轮廓线
RAD6      RMB   1          行地址和字符大小
CCR6      RMB   1          窗口颜色及最后一列
CAS7      RMB   1          第七行,颜色 1/2 及允许轮廓线
RAD7      RMB   1          行地址和字符大小
CCR7      RMB   1          窗口颜色及最后一列
CAS8      RMB   1          第八行,颜色 1/2 及允许轮廓线
RAD8      RMB   1          行地址和字符大小
CCR8      RMB   1          窗口颜色及最后一列
OSDL      RMB   1          当前 OSD 行指针
LIND      RMB   1          行表变址
BROW      RMB   1          字符闪烁行
BCOL      RMB   1          字符闪烁列
WROW      RMB   1          窗口闪烁行
```

```
*ROW1       RMB     1               第一行号
ANAL        RMB     1
ANAF        RMB     1
TEMP2       RMB     1
STACK       RMB     23              23个字节用做堆栈(允许1个中
SP          RMB     1               断9层嵌套子程序)
KEYI        EQU     $00
KEYO        EQU     $00
KEYIO       EQU     $00
SERO        EQU     $03
VOLU        EQU     $0A             D/A 2
CONT        EQU     $0B             D/A 3
BRIL        EQU     $0C             D/A 4
SATU        EQU     $0D             D/A 5
L1          EQU     $05
L2          EQU     $06
WIDE        EQU     $04             宽度短阵位
PST         EQU     $05             伪立体声矩阵位
VCR         EQU     $06             VCR有效位
LOUD        EQU     $05             音量效果有效位
MUT         EQU     $03             无声指示位
STADR       EQU     $80
NORMVOL     EQU     $25             通常音量
*****定义寄存器*****
PORTA       EQU     $00             端口A地址
PORTB       EQU     $01             端口B地址
PORTC       EQU     $02             端口C地址
PORTD       EQU     $03             端口D地址
DDRA        EQU     $04             端口A数据方向寄存器
DDRB        EQU     $05             端口B数据方向寄存器
DDRC        EQU     $06             端口C数据方向寄存器
DDRD        EQU     $07             端口D数据方向寄存器
TCR         EQU     $12             定时器控制寄存器
TSR         EQU     $13             定时器状态寄存器
ICRH        EQU     $14             输入捕捉寄存器
ICRL        EQU     $15
OCRH        EQU     $16             输出比较寄存器
OCRL        EQU     $17
TDRH        EQU     $18             定时器数据寄存器
TDRL        EQU     $19
MISC        EQU     $1C             杂用寄存器
OSD         EQU     $20             18 OSD数据寄存器
CAS         EQU     $32             颜色及状态寄存器
C34         EQU     $33             颜色3/4寄存器
RAD         EQU     $34             行地址及字符大小
WCR         EQU     $35             窗口/列寄存器
```

```
CCR       EQU   $36          列/颜色寄存器
HPD       EQU   $37          水平位置延时
MAD       EQU   $39          M 总线数据寄存器
MFD       EQU   $3A          M 总线分频
MCR       EQU   $3B          M 总线控制寄存器
MSR       EQU   $3C          M 总线状态寄存器
MDR       EQU   $3C          M 总线数据寄存器
TR1       EQU   $3E          测试 1,OSD/定时器/PLM
TR2       EQU   $3F          测试 2
DRAM      RMB   128
* * * * * OSD 更新子程序——行号和数据 * * * * *
NLINE     LDA   OSDL          增加行指针
          INCA
STAG      STA   OSDL
          ADD   LIND
          TAX
          LDA   LTABO,X
          BEQ   STAG
          LDX   OSDL          行指针乘以 3
          LDX   M3,X
          ORA   RAD1,X
          STA   RAD
          LDA   CAS1,X
          STA   CAS
          BCLR  7,WCR
          ROLA
          ROLA                得到 CAS 的第 6 位
          BCC   SKIPW
          BSET  7,WCR         允许窗口显示
SKIPW     LDA   CCR1,X
          STA   CCR
LLOK      LDX   OSDL          行指针
          LDX   M16,X         乘以 16
TOSD1     LDA   DRAM-16,X     取数据写入 OSD 寄存器
          STA   OSD           只需 128 个周期
          LDA   DRAM-15,X
          STA   OSD+1
          LDA   DRAM-14,X
          STA   OSD+2
          LDA   DRAM-13,X     快速 OSD 数据传输
          STA   OSD+3         只需 128(8×16)个周期
          LDA   DRAM-12,X
          STA   OSD+4
          LDA   DRAM-11,X
          STA   OSD+5
          LDA   DRAM-10,X
```

```
                STA     OSD+6
                LDA     DRAM-9,X
                STA     OSD+7
                LDA     DRAM-8,X
                STA     OSD+8
                LDA     DRAM-7,X
                STA     OSD+9
                LDA     DRAM-6,X
                STA     OSD+10
                LDA     DRAM-5,X
                STA     OSD+11
                LDA     DRAM-4,X
                STA     OSD+12
                LDA     DRAM-3,X
                STA     OSD+13
                LDA     DRAM-2,X
                STA     OSD+14
                LDA     DRAM-1,X
                STA     OSD+15
  *TOSD2        STX     TMP1            同样完成写入OSD寄存器功能，
  *             LDX     #16             时间稍长，458周期。
  *             STX     TMP2            该方法采用循环方法传输OSD数据
  *OSDOOP       LDX     TMP1
  *             LDA     <DRAM-1,X
  *             DEC     TMP1
  *             LDX     TMP2
  *             STA     <OSD-1,X
  *             DEC     TMP2
  *             BNE     OSDOOP
*****字符和窗口闪烁*****
                BRCLR   4,CNT,WBLK
  CHBLK         LDA     BROW            字符闪烁
                BEQ     NCHBK
                LDA     RAD
                AND     #$0F
                CMP     BROW
                BNE     NCHBK
                LDA     BCOL
                AND     #$0F
                TAX
                BSR     SPFL            第一个字符(低四位)
                LDX     BCOL
                LSRX
                LSRX
                LSRX
                LSRX
```

```
          BEQ     NCHBK          如高四位为零,则无第二字符
          BSR     SPFL
NCHBK     BRA     NOBLK
WBLK      LDA     WROW           窗口闪烁
          BEQ     NOBLK
          LDA     RAD
          AND     #$0F
          CMP     WROW
          BNE     NOBLK
          BCLR    7,$35
NOBLK     RTI
SPFL      LDA     OSD,X
          AND     #$3F
          BNE     NTSP
          LDA     OSD,X
          AND     #$C0
          ADD     #$0E
          STA     OSD,X
          RTS
NTSP      CLR     OSD,X
          RTS
*****OSD空闲条件*****
OSDEF     BRSET   3,STAT5,DOFF
          BSET    3,STAT5
          BRA     OSDLE
DOFF      BCLR    3,STAT5
OSDLE     SEI
          BCLR    4,STAT5
          BCLR    2,STAT5
          LDX     #29
DOOP      CLR     CAS1-1,X       OSD控制字节清零
          DECX
          BNE     DOOP
          JSR     CDISP2
          CLR     $30
          CLR     $31
          BRSET   3,STAT5,SKPDEF
          LDA     DRAM+12
          STA     $30
          LDA     DRAM+14
          STA     $31
SKPDEF    CLRX
          LDA     #127
          BSR     OSDCLR
          LDA     #%00100000     水平位置:零
          STA     HPD
```

```
            LDA     #%10100011
            BRCLR   2,STAT4,PMD             节目模式?
            LDA     #%10100110              否,颜色 0=黄
PMD         STA     CAS1
            LDA     #%00010000              单位宽度高度
            STA     RAD1
            LDA     #$0C
            STA     ANAL
            LDA     #AVOL
            STA     ANAF
            CLI
            RTS
OSDCLR      INCA
            STA     W1
DCLR        INCX
            CLR     DRAM-1,X
            CPX     W1
            BNE     DCLR
            RTS
*****节目频道/频道名显示*****
BNTAB       FCB     $33,$34,0,$23,$28,0,$33,$34
            FCB     $24,$0E,0,0,$2E,$21,$2D,$25,$C0
MTAB        FCB     0,$5C,0,$9E,0,$3C,0,$FE
            FCB     0,$8B,0,$8D,0,$6D,0,$A9,$C0
PRDSP       BRSET   2,STAT5,OSDLE
            BSET    2,STAT5
            LDA     #%00011010              颜色 2=绿
            STA     C34                     颜色 3=浅蓝
            LDA     #%11100001              开启 OSD 及 PLL
            STA     WCR                     窗口开(第 1 列)
            LDA     #%00100001              水平位置为 1
            STA     HPD
            CLR     $30                     第 17、18 字符为空格
            CLR     $31
            LDA     #%11100101              颜色 1,0=红
            STA     CAS1
            LDA     #%11100110              窗口打开
            STA     CAS8
            LDA     #%00010000              高/宽度
            LDX     #24
STLP        STA     RAD1-3,X
            DECX
            DECX
            DECX
            BNE     STLP
            LDA     #%11100110              颜色 1,0=红,黄
```

```
        LDX     #18
STLP2   STA     CAS2-3,X
        DECX
        DECX
        DECX
        BNE     STLP2
        CLR     OSDL
        LDA     #LTAB3-LTAB0
        STA     LIND
        LDA     #3              第3行开始
        STA     BROW
        LDA     #$03            第3列开始
        STA     BCOL
        LDA     #0
        STA     WROW
        LDA     #1
        STA     COUNT
        LDX     #128
ACLR    CLR     DRAM-1,X        清第1列第8行
        DECX
        BNE     ACLR
        CLRX
BNPL    LDA     BNTAB,X
        CMP     #$C0
        BEQ     FINBN
        STA     DRAM,X
        INCX
        BRA     BNPL
FINBN   LDX     #16
        BCLR    6,STAT5         清标准变化标志
        LDA     COUNT
        STA     W2
*****节目/频道/频道名主循环程序*****
BNLP    LDA     W2              电台号
        STX     W3
        JSR     CBCD
        STA     W1
        AND     #$0F
        ADD     #$10
        LDX     W3
        STA     DRAM+1,X
        LDA     W1
        LSRA
        LSRA
        LSRA
        LSRA
```

```
                BNE     NOTZR
                LDA     #$F0
NOTZR           ADD     #$10
STLSN           STA     DRAM,X
                LDA     W2              台号
                ADD     #$DF
                STA     SUBADR
                LDA     #$A0
                STA     ADDR
                JSR     READ
                LDA     IOBUF+1         频道号
                AND     #$7F
                JSR     CBCD
                STA     W1
                AND     #$0F
                ADD     #$10
                LDX     W3
                STA     DRAM+4,X        高位
                LDA     W1
                LSRA
                LSRA
                LSRA
                LSRA
                ADD     #$10
                STA     DRAM+3,X        低位
*****制式和底线*****
                BSET    5,STAT5
                BRSET   7,IOBUF+1,PALS
                BCLR    5,STAT5
PALS            JSR     CHGST
                LDA     W2
                JSR     GNAME2
                TXA
                ADD     #16
                TAX
                INC     W2
                CPX     #96
                BHI     NOJMP
                JMP     BNLP
NOJMP           CLRX
MTL             LDA     MTAB,X
                CMP     #$C0
                BEQ     ANFIN
                STA     DRAM+112,X
                INCX
                BRA     MTL
```

```
ANFIN     JSR     WIND
SEC30     BSET    4,STAT
          LDA     #30
          STA     TMR
          RTS
*****查找频道名*****
FNAME     BRSET   2,STA4,NONAME      频道模式
          LDA     #$A0
          STA     ADDR
          LDA     #$E0
          STA     SUBADR
          LDA     CHAN
          JSR     CHEX
          STA     COUNT
          BRCLR   1,PORTC,OLOOP      38.9MHz?
          BRCLR   5,PORTC,OLOOP      不，SECAM?
          BSET    7,COUNT            不，PAL
OLOOP     JSR     READ
          LDA     IOBUF+1
          BRSET   1,PORTC,IF38       38.9MHz?
          AND     #$7F               是，忽视制式
IF38      BEQ     CH0
          CMP     COUNT              频道
          BEQ     NOFND
CHO       INC     SUBADR
          LDA     SUBADR
          CMP     #$F7
          BLS     OLOOP
NONAME    LDX     W3
          LDA     #$23               无名字故显示频道号
          STA     DRAM+8,X
          LDA     #$28
          STA     DRAM+9,X
          LDA     CHAN
          LSRA
          LSRA
          LSRA
          LSRA
          ADD     #$10               第3字符
          STA     DRAM+10,X
          LDA     CHAN
          AND     #$0F
          ADD     #$10
          STA     DRAM+11,X          第4字符
          LDA     #$30
          STA     DRAM+13,X
```

```
                LDA     #$21
                STA     DRAM+14,X
                LDA     #$2C
                STA     DRAM+15,X
                BRCLR   1,PORTC,SPAL        38.9MHz?
                BRSET   5,PORTC,SPAL        否。为 PAL?
                LDA     #$33                否,为 SECAM
                STA     DRAM+13,X
                LDA     #$25
                STA     DRAM+14,X
                LDA     #$23
                STA     DRAM+15,X
SPAL            RTS
NOFND           LDA     SUBADR
                SUB     #$DF
GNAME2          LSLA
                LSLA
                ADD     #$7C
                STA     SUBADR
                JSR     READ
                LDX     W3
                LDA     IOBUF+1
                STA     DRAM+12,X           第 1 个字符
                LDA     IOBUF
                STA     DRAM+13,X           第 2 个字符
                INC     SUBADR
                INC     SUBADR
                JSR     READ
                LDX     W3
                LDA     IOBUF+1
                STA     DRAM+14,X           第 3 个字符
                LDA     IOBUF
                STA     DRAM+15,X           第 4 个字符
                RTS
*****光标控制(左和右)*****
CURTAB          FCB     3,4,12,13,14,15
CLFT            BSR     FCUR
                CPX     #5
                BLO     NRAP1
                LDX     #$FF
NRAP1           INCX
NEWC            LDA     CURTAB,X
                STA     BCOL
SEC32           JMP     SEC30
CRGT            BSR     FCUR
                TSTX
```

```
        BNE     NRAP2
        LDX     #6
NRAP2   DECX
        BRA     NEWC
FCUR    LDA     BCOL
        STA     W1
        LDX     #$FF
CRNF    INCX
        LDA     CURTAB,X
        CMP     W1
        BNE     CRNF
        RTS
WIND    LDA     #%00110001          窗口蓝,在17处关闭
        LDX     #18
STLP3   STA     CCR2-3,X
        DECX
        DECX
        DECX
        BNE     STLP3
        LDA     #%00010001          窗口黑,在17处关闭
        STA     CCR1
        STA     CCR8
        LDX     BROW
        DECX
        DECX
        LDX     M3,X
        LDA     CCR1,X
        STA     W2
        BSET    6,W2
        LDA     W2
        STA     CCR1,X
        RTS
*****光标控制(上和下)*****
CUP     LDA     BROW
        CMP     #3
        BLS     TOOSM
        DEC     BROW
        BRA     WIND
TOOSM   LDA     COUNT
        CMP     #1
SEC31   BEQ     SEC32
        DEC     COUNT
        BRA     FIN30
CDWN    LDA     BROW
        CMP     #8
        BHS     TOOBG
```

```
        INC     BROW
        BRA     WIND
TOOBG   LDA     COUNT
        CMP     #19
        BEQ     SEC31
        INC     COUNT
FIN30   JSR     SEC30
        JMP     FINBN
*****制式变化*****
CHST    BCLR    5,STAT5         默认 SECAM
        LDX     BROW
        DECX
        DECX
        DECX
        LDX     M16,X
        LDA     DRAM+6,X
        AND     #$3F
        CMP     #$30            PAL?
        BEQ     SZER
        BSET    5,STAT5         不,PAL
SZER    BSET    6,STAT5         制式变化
        JSR     SEC30
CHGST   LDA     #$30
        STA     DRAM+6,X
        LDA     #$21
        STA     DRAM+7,X
        LDA     #$2C
        STA     DRAM+8,X
        LDA     #0
        STA     DRAM+9,X
        STA     DRAM+10,X
        BRCLR   1,PORTC,PAL     38.9MHz?
        BRSET   5,STAT5,PAL     否,PAL?
SECAM   LDA     #$33            否,SECAM
        STA     DRAM+6,X
        LDA     #$25
        STA     DRAM+7,X
        LDA     #$23
        STA     DRAM+8,X
        LDA     #$21
        STA     DRAM+9,X
        LDA     #$2D
        STA     DRAM+10,X
PAL     BRCLR   6,STAT5,NSTCH
        TXA
        ADD     #5
```

```
          STA     COUNT
XLP       LDA     DRAM+6,X
          ADD     #$40
          STA     DRAM+6,X
          INCX
          CPX     COUNT
          BNE     XLP
NSTCH     RTS
*****字符变化*****
PLUS      BSR     GETIT
          INCA
          AND     #$3F
          CMP     #$19          9
          BHI     MT9
LTE9      CMP     #$10          0
          BHS     NLT0
          LDA     #$10          0
          BRA     NLT0
MT9       CMP     #$21          A
          BHI     MTA
          LDA     #$21          A
MTA       CMP     #$3A          Z
          BLS     NLTO
SPACE     LDA     #$00          空格
NLTO      ORA     #$40
          STA     DRAM,X
          JMP     SEC30
MINUS     BSR     GETIT
          DECA
          AND     #$3F
          CMP     #$21          A
          BLO     LTA
GTEA      CMP     #$3A          Z
          BLS     NLTO
          LDA     #$3A          Z
          BRA     NLTO
LTA       CMP     #$19          9
          BLS     LT9
          LDA     #$19          9
LT9       CMP     #$10          0
          BHS     NLT0
          BRA     SPACE
GETIT     LDA     BROW
          SUB     #2
          LSLA                  ×2
          LSLA                  ×4
```

```
                LSLA    ×8
                LSLA    ×16
                ADD     BCOL
                TAX
                LDA     DRAM,X
                RTS
*****存储名*****
  SAVE          LDA     #$A0
                STA     ADDR
                LDA     COUNT
                ADD     BROW
                LSLA
                LSLA
                ADD     #$70
                STA     SUBADR
                LDA     #3
                STA     W1
                STA     W2
                LDX     BROW
                DECX
                DECX
                LSLX
                LSLX
                LSLX
                LSLX
                STX     W3
                LDA     DRAM+12,X
                AND     #$3F
                STA     IOBUF
                LDA     DRAM+13,X
                AND     #$3F
                STA     IOBUF+1
                LDX     #SUBADR
                JSR     WRITE
                INC     SUBADR
                INC     SUBADR
                LDX     W3
                LDA     #3
                STA     W1
                STA     W2
                LDA     DRAM+14,X
                AND     IOBUF
                LDA     DRAM+15,X
                AND     #$3F
                STA     IOBUF+1
                LDX     #SUBADR
```

```
                JSR     WRITE
                LDX     E3
                LDA     DRAM+3,X
                LSLA
                LSLA
                LSLA
                LSLA
                STA     W1
                LDA     DRAM+4,X
                AND     #$0F
                ADD     W1
                JSR     CHEX
                STA     IOBUF
                LDX     W3
                LDA     DRAM+6,X
                AND     #$3F
                CMP     #$33
                BEQ     STSEC
                BSET    7,IOBUF
STSEC           LDA     COUNT
                ADD     BROW
                ADD     #$DC
                STA     SUBADR
                LDA     #2
                STA     W1
                STA     W2
                LDX     #SUBADR
                JSR     WRITE
                JSR     SEC30
                JMP     FINBN
*****OSD行号表*****
LTAB0           FCB     10,0                    空闲显示
LTAB1           FCB     9,8,0                   PR/CH 显示
LTAB2           FCB     7,8,10,0                模拟显示
LTAB3           FCB     2,3,4,5,6,7,8,9,0       PR/CH/CTD/名字表
M16             FCB     $10,$20,$30,$40,$50,$60,$70,$80     ×16
M3              FCB     0,3,6,9,12,15,18,21                 ×3
*****右下角节目/频道号显示*****
PCDSD           LDA     #$0C
                STA     ANAL
                LDA     #AVOL
                STA     ANAF
                BCLR    4,STAT5
                LDA     #%00001010              颜色 2=绿
                STA     C34                     颜色 3=蓝
                LDA     #%01110000              OSD 及 PLL 开
```

```
        STA     WCR             窗口关(第16列)
        LDA     #%00100010      水平位置:2
        STA     HPD
        CLR     $30             第17,18字符为空格
        CLR     $31
        CLRX
        LDA     #9
        JSR     OSDCLR          清除未用字符
        LDX     #16
        LDA     #31
        JSR     OSDCLR          清除未用字符
PNAME   LDA     #16
        STA     W3
        LDA     PROG
        BEQ     SKPGN
        JSR     FNAME
SKPGN   LDA     #1              从1开始
        STA     OSDL
        LDA     #LTAB1-LTAB0
        STA     LIND
        LDA     #%10100011      颜色1,0=红,青
        BRCLR   2,STAT4,PMD2    节目模式?
        LDA     #%10100110      否,颜色0=黄
PMD2    STA     CAS1
        STA     CAS2
        LDA     #%11010000      双宽/高
        STA     RAD1
        LDA     #%00010000      单宽/高
        LDA     #%00010000
        STA     RAD2
        LDA     #%00010010      窗口浅蓝
        STA     CCR1
        STA     CCR2
SEC5    BSET    4,STAT
        LDA     #30
        BRSET   2,STAT4,S30     频道模式?
        BRSET   0,STAT6,S30     否,2位节目号输入?
        LDA     #6              否,仅6秒
S30     STA     TMR
        RTS
*****底行模拟条*****
CHAR    FCB     $0E,$12,$11,$13     条块字符
ANCH    FCB     $63,$6F,$6E,$74,$A2,$B2,$A9,$AC
        FCB     $63,$B3,$21,$F4,$F6,$EF,$EC,$F5
ANOSD   STA     W3
        BRSET   4,STAT5,LOGO        模拟
```

```
        BSET    4,STAT5          建立跳变标志
        CLRX
        LDA     #127
        JSR     OSDCLR           清除所有字符
        LDX     #29
COOP    CLR     CAS1-1,X
        DECX
        BNE     COOP
        LDA     #%00001010       颜色 2=绿
        STA     C34              颜色 3=蓝
        LDA     #%11100001       OSD 及 PLL 开启
        STA     WCR              窗口开启(第 1 列)
        LDA     #%00100010       水平位置:2
        STA     HPD
        CLR     $30              第 17、18 字符为空格
        CLR     $31
        LDA     #%11100110       颜色 1,0=红、绿
        STA     CAS1             窗口开启
        STA     CAS2
        LDA     #%10100110
        STA     CAS3
        LDA     #%00010000
        STA     RAD1
        STA     RAD2
        STA     RAD3
        LDA     #%11100011       窗口白,在 3 处关闭
        STA     CCR1
        STA     CCR2
        LDA     #%00010001       窗口黑,在 17 处关闭
        STA     CCR3
        CLR     OSDL
        LDA     #LTAB2-LTAB0
        STA     LIND             第二个表
*****模拟显示块*****
LOGO    LDX     ANAL
        LDA     ANCH,X
        STA     DRAM
        INCX
        LDA     ANCH,X
        STA     DRAM+1
        INCX
        LDA     ANCH,X
        STA     DRAM+16
        INCX
        LDA     ANCH,X
        STA     DRAM+17
```

```
*****模拟长条显示*****
                LDA     W3
                CLR     W2
                LSRA
                ROL     W2
                LSRA
                ROL     W2
                STA     W3
                LDX     #16
LRAN            DECX
                CPX     W3
                BEQ     STAR
                BHI     DOT
                LDA     #$14
                BRA     SKST
DOT             LDA     #$0E
                BRA     SKST
STAR            STX     W1
                LDX     W2
                LDA     CHAR,X          .,1,2或3
                LDX     W1
SKST            STA     DRAM+32,X
                TSTX
                BNE     LRAN
                JMP     SEC5
                END
```

19. MC68HC05B6 的 RAM 和 EEPROM1 的串行自引导方式

MC68HC05B6 片内有 256 字节 EEPROM，称为 EEPROM1，用于存储非挥发性数据。另外，将一小段程序装入 RAM 并执行是试验软件程序的简便方法。该例子介绍串行装入 EEPROM1 的方法。

MC68HC05B6 有内部 RAM 串行自引导程序，可通过 SCI 很容易地将程序装入 RAM。但“B6”不能接收普通的 S 记录。协议要求数据为纯二进制数据，没有地址信息，自引导程序总是从 RAM 地址 $50 开始装入程序。第一个字节为程序字节数(含该字节)，从 $51 开始为程序。每装入(接收)一个字节，计数字节就减 1。当计数字节减为零时，转向 $51 地址，并开始执行已装入的程序。该文包括下列两部分内容：

- 如何将汇编输出转换为 MC68HC05B6 RAM 自引导程序所能接收的格式。
- 如何自引导装入 MC68HC05B6 的 EEPROM1。

19.1 S 记录格式的转换

RAM 自引导方式的设置如图 19-1 所示。当满足这些条件时，复位后开始执行串行自引导程序，通过 RS-232(9600 波特)装入程序。由于自引导程序所接受的格式不是 S 记录，需要一个转换程序。另外一个问题是，当文件拷贝到 PC 机的通讯口进行传输时，传输的是 ASCII 码，而不是二进制数。例如，通过 COM 通讯口将含有字节数据 $A5 的数据拷贝到 MC68HC05B6 时，B6 实际上接收两个字节：$41 和 $35，它们分别代表字符 A 和 5。

因此，转换程序必须去掉 S 记录格式并将数据转换为二进制格式，以传输到 HC05B6。而且，在输出文件的开始处还必须插入计数字节。

后面的 PASCAL 程序 BINCONV 执行上述三项工作，BINCONV 的流程图如图 19-2 所示。当调用 BINCONV 时，将出现 S 记录输入文件名和二进制输出文件名提示信息。读入每一个 S 记录后，就将其转换为二进制数据并存入临时文件中。读入每个 S 记录时并将其返回到屏幕。所有 S 记录读入并处理后，将出现提示并询问是否要加计数字节。对于 68HC05B6 RAM 自引导程序，回答总是要加计数字节。在数据从临时文件拷贝到输出文件之前，将计数字节写入到输出文件。最后，计数字节显示出来，以便用户确认。注意计数字节等于程序字节数加上 1。自引导程序只接收标准 S 记录格式，若检测到无效的字符或数据格式将出错。

当 PC COM 口设置为 9600 波特，68HC05B6 按图 19-1 设置时，可传输二进制文件。执行过程如下：

(1)HC05B6 复位；

(2)PC 机输入命令：“COPY XXXX.YYYCOM1\B”。

程序则传输到 B6 并自动开始执行。注意\B 是用来指明二进制文件传输，因此，若在文件中间发现文件结束字符(EOF)，不会中止拷贝过程。

将 S 记录文件转换为二进制格式的 BINCONV 程序如下：

```
var
    SrecFile: text;
```

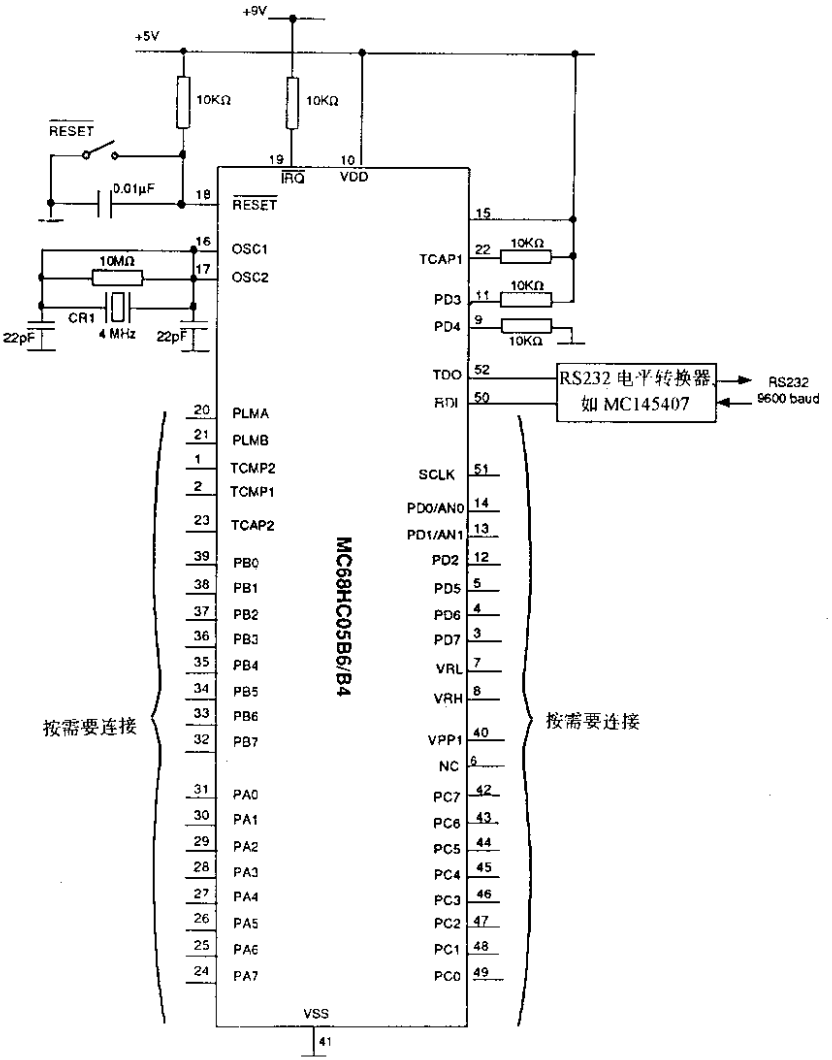


图 19-1　MC68HC05B6 的自引导方式

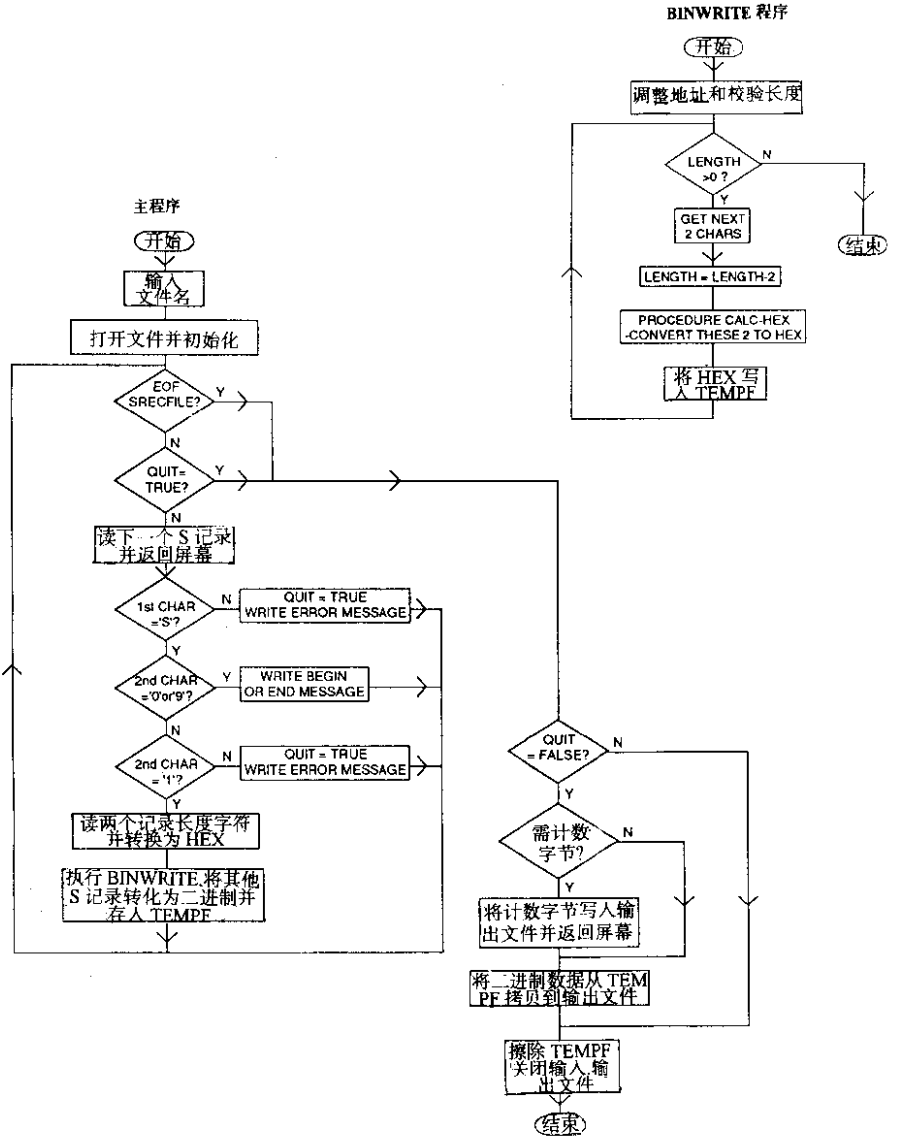


图 19-2　BINCONV 程序流程图

```
    BinFile: file;
    Tempf: file;
    srec: string [100];
    Transfer: array [1..20000] of char;
    numread, numwritten: word;
    answer: char;
    fnamei: string [15];
    fnameo: string [15];
    bytout: char;
    countbyt: integer;
    datcnt: integer;
    datval: integer;
    point: integer;
    cnt1: integer;
    cnt2: integer;
    quit:boolean;
    Count: boolean;
{ ------------------------------------------ }
Procedure Calc hex (chr1, chr2: integer);
    { Combines 2 characters into a single byte value i.e A5→165, error
      signaled if non hex character detected }

        Begin
        Case chr1 of
        48..57: chr1: = chr1 - 48;
        65..70: chr1: = chr1 - 55;{Is this a valid hex character?}
        else
          begin
            writeln ( 'invalid data - conversion aborted ');
              quit: = true
          end
    Case chr1 of
48..57: chr2: = chr2 - 48;
65..70: chr2: = chr2 - 55;
    else
      begin
        writeln ( 'invalid data - conversion aborted ');
          quit: = true
      end
    end;
```

```
    datval: = chr1 * 16 + chr2;      (Convert to single byte)
    end;

{ ------------------------------------------ }
    Procedure Binwrite (length, dpoint: integer);
{ Converts an S — record line to hex and stores it in a temporary file }

    begin
    length: = length — 3;          { Allow for address and checksum bytes }
    countbyt: = countbyt + length;      { Update running byte total }
    length: = length * 2;          { Twice as many characters as bytes }
    while length > 0 do
    begin
        cnt1: = Ord (srec [dpoint]);      {Get the next two characters }
        cnt2: = Ord (srec [dpoint + 1 ]);
        dpoint: = dpoint + 2;                          {Update pointer and length }
        length: = length — 2;

        Calc_hex (cnt1,cnt2);      { Convert two characters into single byte }
        bytout: = Chr (datval);      { — now convert that single byte into a }
        blockwrite (tempf, bytout, 1)      { character and save it in temporary file }

    end
    end;
{ * * * * * * * * * * MAIN PROGRAM STARTS BELOW * * * * * * * * * * }

begin
    writeln (' S — record to Binary conversion utility');
    writeln;
    writeln;
    write ( 'Input S—record file name? → ');
    readln (fnamei);
    assign (SrecFile, fnamei);
    write ( ' Binary output file name? → ');
    readln (fnameo);
    assign (BinFile,fnaneo);
    assign (tempf, 'temp. tmp');
    quit: = false;
    countbyt: = 1;
    Reset (SrecFile);          { open the two }
```

```
Rewrite (BinFile,1);      { -selected files }
Rewrite (tempf, 1);       { + a temporary file }
while not Eof (SrecFile) and not quit do
begin
    readln (SrecFile, srec);      { read S-rec into char string srec }
    writeln (srec);

    If srec [1] = 'S' then        { If string does not start with S then quit }
      begin
        CASE srec [2] of
        '1':          { If not S1 record then loop back }
          begin
              cnt1: = Ord (srec [3]);      { get the 2 record length }
              cnt2: = Ord (srec [4]);      { characters }
              calc_hex (cnt1,cnt2);        { func to produce hex in datcnt from
                                             cnt1 & 2 }
              datcnt: = datval;
              point: = 9;                   { point to first data character }
              binwrite (datcnt, point)      { convert the data in this s-rec line
                                              to binary and store in temp file }
          end;
        '0': writeln ('Conversion started');
        '9': writeln ('last S-record done');
        else
          begin    { If not S0, SlorS9 record then abort }
            quit: = true;
            writeln ('Non standard S-record detected - Conversion aborted')
          end
        end
      end
      else
        begin      { If lst char not an S then abort }
          quit: = true;
          writeln ('Non standard S-record detected - Conversion aborted')
        end
    end;

    If quit = false then

{ If no errors then copy the temporary file to the output file and add in a count byte if
```

```
required }

        begin
        Reset (tempf, 1);
        writeln;
        write ('Do you want a count byte added to start of output file? →');
        readln (answer);
        If upcase (answer) = 'Y' then
              Begin
                writeln ('Total size including count byte = ', countbyt);
                bytout: = chr (countbyt);
                blockwrite (binfile, bytout, 1)
              end;
            repeat
                blockread (tempf, transfer, sizeof (transfer), numread);
                blockwrite (binfile, transfer, numread, numwritten);
            until (numread = 0) or (numwritten <> numread)
        end;

        close (tempf);
        erase (tempf);      { Finished with temporary file so erase it }
        close (SrecFile);
        close (BinFile)     { Close files before quiting }
end.
```

19.2 自引导装入 EEPROM1

用户可利用前面介绍的 RAM 自引导程序将程序装入 MC68HC05B6 的 EEPROM1 中。方法是将 EEPROM1 引导程序汇编并装入 HC05B6 的 RAM，在此执行该程序，再将数据或程序装入 EEPROM1。

HC805B6 内部有 EEPROM 自引导程序，与之相应的 PC 程序为 E2B6，它将 S 记录装入器件进行编程。下面介绍 HC05B6 的 EEPROM1 自引导方法。EEPROM1 自引导程序与 HC805B6 PC 程序 E2B6 相同，不必开发另外的 PC 程序。

HC05B6 有 176 字节 RAM，可用于 EEPROM1 自引导程序，协议应尽量简单，程序效率要高。E2B6 程序的格式是：传输 2 个地址字节，随后是该地址的编程数据。同时，HC05B6 将以前编程地址数据返回，以便校验。后面的 EE1BOOT 为 HC05B6 的自引导程序，它分为 4 部分：主程序、擦除程序、编程程序和 SCI 服务程序。编程和擦除子程序的核心是扩展寻址程序 EXTSUB，用来访问 EEPROM1 阵列。

注意，EEPROM1 地址在编程前总是先擦除。定时器输出比较功能用来提供编程和擦除所需的 10ms 延时。若数据为 $FF，则跳过编程步骤。自引导装入 HC05B6 的 EEPROM1 的过程如下：

(1)按图 19-1 设置 68HC05B6。

(2)将 EE1BOOT 程序汇编,并用上述 BINCONV 将汇编后的 EE1BOOT 程序转换为二进制。

(3)设置 PC COM 口为 9600 波特,然后使 HC05B6 复位。

(4)用命令:"COPY EE1BOOT. BINCOM1/B"将 EE1BOOT 装入 HC05B6 的 RAM。这时开始执行 EE1BOOT。

(5)启动 PC 机的 E2B6 程序,按指示将 S 记录装入 68HC05B6 的 EEPROM1 中。

注意:

①只有与 EE1BOOT 一起使用,E2B6 的装入过程才有效。

②一旦将 EEPROM1 的保密位置位,则 68HC05B6 的 RAM 自引导程序不再运行。

EE1BOOT——68HC05B6 EEPROM1 串行自引导程序如下:

```
* 该程序通过 RAM 自引导程序装入 MC05B6 的 RAM 中,然后开始执行。格式与 MC805B6 相同,因此可以用 E2B6 程序将 EEPROM1 编程。
* 注意,初始化时 E2B6 返回无效字符,因此,该程序忽略 SCI 接收的第一个字符。
* I/O 和内部寄存器定义
* I/O 寄存器
PORTA     EQU   $00      port A.
PORTB     EQU   $01      port B.
PORTC     EQU   $02      port C.
PORTD     EQU   $03      port D.
DDRA      EQU   $04      port A DDR.
DDRB      EQU   $05      port B DDR.
DDRC      EQU   $06      port C DDR.
EECONT    EQU   $07
E1ERA     EQU   2
E1LAT     EQU   1
E1PGM     EQU   0
BAUD      EQU   $0D
SCCR1     EQU   $0E
MBIT      EQU   4
SCCR2     EQU   $0F
SCSR      EQU   $10
RDRF      EQU   5
SCDAT     EQU   $11
* TIMER 寄存器
TCR       EQU   $12      Timer control register.
TOIE      EQU   5        Timer overflow interrupt enable.
OCIE      EQU   6        Timer output compares interrupt enable.
ICIE      EQU   7        Timer input captures interrupt enable.
TSR       EQU   $13      Timer status register.
OCF2      EQU   3            Timer output compare 2 flag.
ICF2      EQU   4            Timer input capture 2 flag.
TOF       EQU   5            Timer overflow flag.
```

```
OCF1        EQU     6               Timer output compare 1 flag.
ICF1        EQU     7               Timer input capture 1 flag.
TOC1HI      EQU     $16             Timer output compare register 1 (16-bit).
TOC1LO      EQU     $17
TIMHI       EQU     $18             Timer free running counter (16-bit).
TIMLO       EQU     $19
*其它定义
LDAEXT      EQU     $C6             OP-Code for LDA extended.
MS10        EQU     $14             10ms delay constant.
*程序开始
            ORG     $51
RESET       LDA     #$00
            STA     DDRA            All Ports inputs.
            STA     DDRB
            STA     DDRC
SCIINT      BCLR    MBIT,SCCR1      Initialise SCI - 8 data bits.
            LDA     #$C0
            STA     BAUD            9600 baud at 4MHz
            LDA     #$0C            Enable transmit and receive.
            STA     SCCR2
            STA     SCSR            Clear pending flags.
            LDA     #LDAEXT         Init extended addressing subroutine to LDA.
            STA     OPCDE
            BSR     SCREAD          Wait here and ignore 1st char (E2B6 init).
LOOP        BSR     EXTSUB          Load Acc with data from last programmed addr
            STA     SCDAT           Send it back for host to verify.
            BSR     SCREAD          Get high address
            STA     ADDHI           - and store it.
            BSR     SCREAD          Get low address
            STA     ADDLO           - and store it.
            BSR     SCREAD          Get the data to be programmed
            STA     DATA            Store it temporarily.
            INC     OPCDE           Change the ext addr subroutine to STA aaaa.
            BSR     ERASEE          Erase the selected address for 10ms.
            BSR     PROGEE          Now prog the data for 10ms.
            DEC     OPCDE           Restote ext addr subroutine to LDA aaaa.
            BRA     LOOP
*服务 SCI 子程序
SCREAD      BRCLR   RDRF,SCSR,*
            LDA     SCDAT
            RTS
*访问全部存储器映象的扩展寻址子程序
EXTSUB      EQU     *
OPCDE       FCB     0
```

```
ADDHI       FCB     0
ADDLO       FCB     0
RTS
DATA        FCB     0           Reserved Byte for data during erasing.
```

EEPROM1 擦除子程序

```
ERASEE      BSET    E1LAT,EECONT
            BSET    E1ERA,EECONT
            BSR     EXTSUB
            LDA     #MS10
DEL1        BSET    E1PGM,EECONT
            STA     TIMLO       Set up timer for a 10ms count
            STA     TOC1HI
            STA     TSR         — using output compare 1 function.
            STA     TOC1LO
            BRCLR   OCF1,TSR,*  Wait here for end of erase time
            CLR     EECONT      — erase finsished.
            RTS
```

* EEPROM1 编程子程序

```
PROGEE      BSET    E1LAT,EECONT
            LDA     DATA
            BSR     EXTSUB
            INCA
            BEQ     SKIP        Skip programming if data = $FF
            LDA     #MS10
DEL         BSET    E1PGM,EECONT
            STA     TIMLO       Set-up timer for 10ms count
            STA     TOC1HI
            STA     TSR         — using output compare 1 function.
            STA     TOC1LO
            BRCLR   OCF1,TSR,*  Wait here for programming to finish.
SKIP        CLR     EECONT
            RTS
```

20. MC68HC805B6 和 MC68HC705B5 串行/并行编程电路

图 20-1 所示电路可以用来对 MC68HC805B6 和 MC68HC705B5 进行编程。通过跳线开关，可选择两种编程模式：并行模式和串行模式。

在并行编程模式，外部 EPROM 中的用户程序拷贝到 MCU 内部的 EEPROM 或 EPROM；而在串行编程模式，通过串行口读或编程 MCU 内部的 EEPROM 或 EPROM。

注意，若 MC68HC705B5 的保密位有效，则不能进行编程操作。对于 MC68HC805B6 若保密位有效，则首先擦除。

表 20-1 是 68HC805B6 和 68HC705B5 这两种器件的 4 种操作方式，表中 J2 和 J3 是电路中的跳线。

表 20-1 操作方式

跳线 J2	跳线 J3	68HC805B6	68HC705B5
SERIAL	BOOT ONLY	串行装入(无擦除)	RAM/EPROM 串行自引导
SERIAL	ERASE+BOOT	串行装入，带擦除	EPROM 擦除检验
PARALLEL	BOOT ONLY	并行 RAM 自引导	并行 RAM 自引导
PARALLEL	ERASE+BOOT	并行 EEPROM 自引导	并行 EPROM 自引导

20.1 并行编程模式

该模式将外部 27C64 EPROM 的内容直接编程到 MCU 的内部 EEPROM(HC805B6)或内部 EPROM(HC705B5)中。功能包括编程、校验和擦除(只对于 HC805B6)。

在该模式中，编程前自动擦除 HC805B6 的所有内部 EEPROM。若 EEPROM 未擦除干净则红 LED 亮，并再次擦除。EEPROM 擦除时间通常为 50ms。EEPROM 的擦除状态为 $FF。

27C64 EPROM 应含有编程 HC805B6 6K 字节 EEPROM 的数据，因此，外部 EPROM 只应含有地址 $800～$1FFF 的数据即可。注意，HC805B6 的 256 字节 EEPROM1 阵列不能由外部 EPROM 编程。

当编程 HC805B6 时，对应于非内部 EERPOM 或 EPROM 地址的外部 EPROM 地址自动跳过。编程操作期间，绿 LED 以大约 1 秒的周期发光闪烁，表明正常编程方式。编程结束后，编程内容与外部 EPROM 进行校验。若发现错误则红 LED 亮；若两者相符则绿 LED 亮。

将跳线开关分别置于“SERIEL”和“ERASE+BOOT”位置，HC705B5 的 V_{PP} 引脚加+5V 电源，可以检验 HC705B5 EPROM 的擦除状态。在这种情况下，按照并行编程模式，忽略步骤(4)和(5)，但 IC2 插槽中不要插 27C64 EPROM。绿 LED 亮表明擦除成功，红 LED 亮表明 EPROM 未擦除干净。

20.2 并行编程操作

用并行编程模式将 MCU 内部 EPROM/EEPROM 编程的步骤如下：

(1)去掉电源，插入 MCU 和 27C64 EPROM。

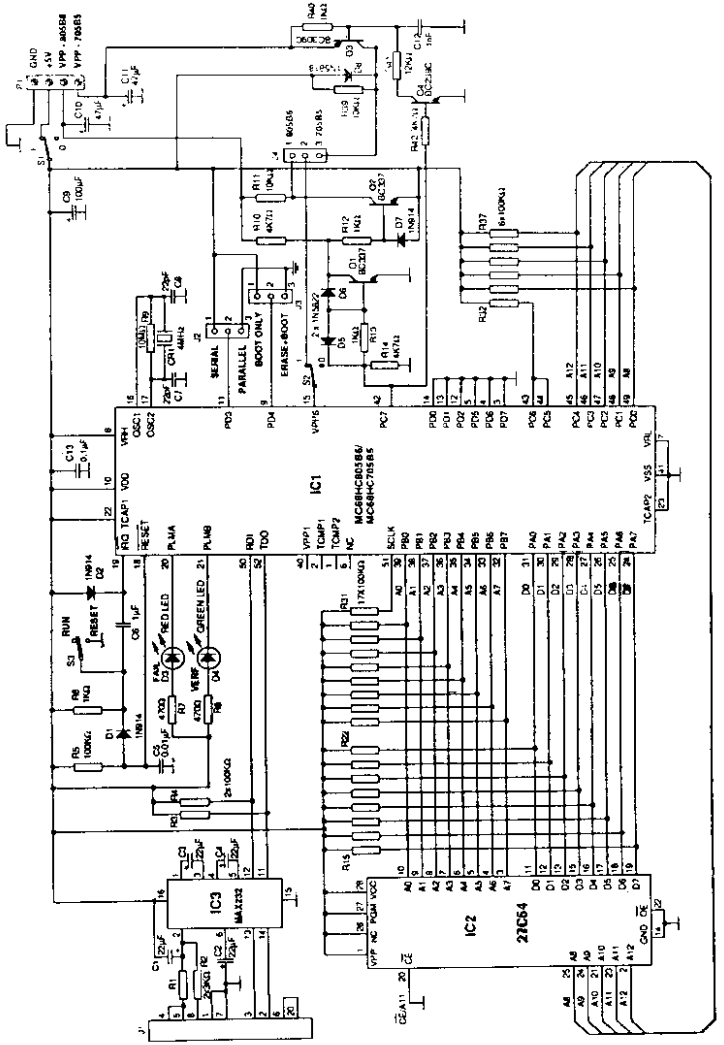


图 20-1　HC805B6 和 HC705B5 串行/并行自引导编程器

(2)断开电源开关 S1 和 S2,S3 开关位于 RESET 位置,接+5V 电源和适当的 V_{PP}电源(按 HC705B5 和 HC805B6 要求)。

(3)根据编程器件(HC705B5 和 HC805B6),设置跳线 J4。

(4)置跳线 J3 位于"ERASE+BOOT"位置。

(5)置跳线 J2 于"PARALELL"位置。

(6)开启+5V 电源,S1 为"ON"。

(7)开启 V_{PP}电源,S2 为"ON"。

(8)置开关 S3 于"RUN"位置。

(9)一旦绿 LED 停止闪烁,并且继续保持亮,置 S3 于"RESET"位置。

(10)断开 V_{PP}电源(S2 断开)。

(11)断开+5V 电源(S1 断开)。

20.3 串行编程模式

该模式通过串行口读或编程 MCU 内部 EPROM 或 EEPROM。采用计算机和 E2B6 这样的软件,可将数据装入 MCU,或将 MCU 的数据返回主机。

串行模式编程包括,MCU 从串行口读一个字节数据,编程到内部 6K EPROM/EEPROM 阵列,从编程地址读取数据,并送至串行口。主机应该对从编程板返回的数据进行检验。

象并行编程模式一样,数据为 $FF 的 EEPROM 编程字节和数据为 $00 的 EPROM 编程字节自动跳过,以便节省时间。但仍能将数据返回,以便校验。

1. MC68HC805B6

E2B6 程序适用于 IBM PC,通过 RS-232 连接。该程序可将数据从 HC805B6 传输至 PC(μpload),称为读 EEPROM,也可以用 S1 记录文件对 HC805B6 的 EEPROM 编程。

与并行编程模式相同,在内部 EEPROM 编程之前,自动擦除 EEPROM 区域。在串行模式,这一操作是可选择的,J3 可选择为"BOOT ONLY"或"ERASE+BOOT"。若 EEPROM 保密位有效,不论 J3 为何位置,都进行擦除。若 J3 置于"ERASE+BOOT",或保密位有效,则读 EEPROM 也会使得擦除 EEPROM。

2. MC68HC05B6

可以用前述第 19 个应用例子所述方法对 HC05B6 ROM 型 MCU 内的 256 字节 EEPROM 编程。采用这里所述的编程板时,跳线开 关应按 HC805B6 设置,J2 和 J3 分别为"SERIEL"和"ERASE+BOOT",但 HC805B6 的 V_{PP}引脚应接至+5V。

3. MC68HC705B5

EPB5 程序适于 PC 机通过 RS-232 与编程板通讯。该程序可上行装入(数据从 HC705B5 传输至 PC 机),即读 EPROM;也可以将 S1 记录文件装入到 HC705B5,对 EPROM 编程。

20.4 串行编程操作

(1)运行 E2B6 程序(HC805B6)或 EPB5 程序(HC705B5),与被编程器件进行通讯。

(2)去掉电源,安装 MCU 和主机与串行口之间的连线。

(3)断开开关 S1 和 S2,开关 S3 置于"RESET"位置,连接+5V 电源和 HC805B6/HC705B5 的 V_{PP}电源。

(4)按 HC705B5 和 HC805B6 设置跳线开关 J4。

(5)J3 跳线开关应置于适当位置。例如,若从 MCU 读数据,应置于"BOOT ONLY"位置;若对 EEPROM 编程,应置于"ERASE+BOOT"位置(只对于 HC805B6)。

(6)置 J2 于"SERIEL"位置。

(7)开+5V 电源,S1 为 ON。

(8)开 V_{PP}电源,S2 为 ON。

(9)主机控制程序出现提示时,将开关 S3 置于"RUN"位置。

(10)根据上行/下行装入程序进行数据传输。

(11)操作结束时,将 S3 打到 RESET 位置。

(12)断开 V_{PP}电源,断开 S2。

(13)断开+5V 电源,断开 S1。

编程几个器件时,重复上述(2)~(13)。

21. MC68HC05E0 EPROM 仿真器

MC68HC05E0 是通用单片机，与其他型号不同，它没有片内 ROM，可寻址 64K 外部存储空间。该单片机具有通常的 I/O 口、定时器等，可根据需要外接 ROM、EPROM 程序存储器或 RAM 以及其它硬件。

这里介绍的 EPROM 仿真器是这种单片机的一种典型应用情况。除程序 EPROM 外，它还使用键盘、LCD、串行通讯和 64K RAM。该仿真器可用 RAM 代替程序 ROM 或 EPROM（多达 64K×8）。

目标码可以由 EPROM 串行装入，利用键盘和 LCD 显示可检查或修改目标码。目标系统可以利用新的或修改的程序，而不必在每次改变软件后重新进行 EPROM 擦除和编程过程。可以在 $0100 选择偏移量，以使目标码位于正确的目标系统存储器映象。

对于控制程序含在 27(C)16/32/64/128/256/512 中的任何系统，该仿真器能容易地调试硬件和软件。控制软件包括 6805 目标码的分支偏移量计算，特别适用于 M68(HC)05/01/11 单片机系统的调试。

有两种方法将程序装入。第一种方法是利用外部 EPROM，该 EPROM 的内容可由单片机传输到 RAM。这种方法在修改 EPROM 之前检查有较小改动的情况下是很有用的。可由目标系统接口（通过仿真器的缓冲器）读 EPROM，也可由直接与单片机相连的单独的插槽读 EPROM。

另一种方法是通过 RS－232 以 S 记录格式将数据串行装入，数据可以来自 PC 机的 COM 口（用 COPY 命令）。

21.1 操作原理

图 21-1 是仿真器的 3 种操作模式的框图。其数据/地址方向由 74HC245 双向三态 8 位缓冲器控制。这些缓冲器组成两个 18 位地址和控制信号缓冲器和两个 8 位数据缓冲器。方向控制信号和允许信号由 MC68HC05E0 单片机提供。

1. 模式 A

在模式 A，允许缓冲器 B1 和 B2，从左至右提供地址（RAM 也接收该地址，但其数据输出被禁止），单片机可读目标系统接口的 EPROM 内容。缓冲器 B3 和 B4 也允许，可从 EPROM 向单片机返回数据，RAM 被禁止，故不影响 B3 和 B4 之间的数据总线。

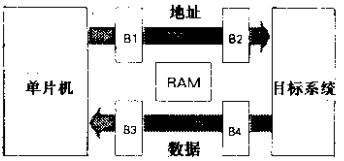
2. 模式 B

在模式 B，单片机可以读或写入 RAM。缓冲器 B1 被选通，由单片机为 RAM 提供地址。B3 被选通，允许从 RAM 读或向 RAM 写入数据。B3 的双向控制由 MC68HC05E0 的 R/W 信号执行。B4 被禁止，B2 也被禁止。

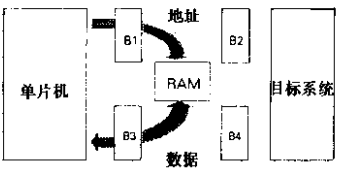
3. 模式 C（仿真模式）

在模式 C，缓冲器 B2 和 B4 被选通，目标系统访问 RAM，B1 和 B3 被禁止。

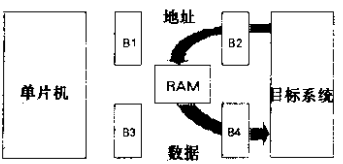
3 种模式下的控制线状态如表 21-1 所示。



(a)



(b)



(c)

图 21-1 仿真器的 3 种操作模式框图

(a)模式 A:直接访问目标系统接口;(b)模式 B:访问仿真器 RAM;(c)模式 C:目标系统访问 RAM

表 21-1 3 种模式的控制线

控制线	模式		
	A	B	C
4,B 口	1	1	0
5,B 口	0	0	1
6,B 口	0	1	0
7,B 口	1	0	0

21.2 电路

仿真器电路如图 21-2 所示,MC74HC138 用来提供片选允许。仿真器硬件允许的地址范围为 $4000～$7FFF,EM64K 程序 EPROM 27(C)64 为 $C000～$FFFF。若 EM64K 程序位于 27(C)16,其引脚 21(VPP)应保持为高。

另一个插槽地址为 $8000～$BFFF,目的是为了使程序从 EPROM 装入,而不必断开至目标系统的电缆。仿真 RAM 占据的地址范围为 $4000～$7FFF。由于只为 16K 空间,RAM 进行分页。I/O 口线(B 口的位 0 和位 1)选择 4 页。仿真器的存储器映象如图 21-3 所示。

PB4～7 控制线通过电阻接相应电平,使其保持在模式 B。在仿真器硬件调试期间,建议采用限流 50～100mA 的电源,以避免总线竞争造成过大电流,损坏缓冲器。

6 位 LCD 显示由 MC145000 显示驱动器驱动,该驱动器由单片机的两条线控制。该 LCD 采用 4 个后板(BP)信号扫描形式。另一种静态显示方式如图 21-4 所示。它采用 3 片 MC144115 驱动器,这种形式需要的连线多,PB3 用来提供 MC144115 的片选允许信号。静态显示可直接由 5V 电源供电,多路扫描显示需较低电压,图 21-2 中用 20kΩ 的电位器调节亮度。

键盘采用 MC14028 译码器,以减少所用的 I/O 引脚数。注意 PA6 用于键盘和显示驱动器。LOAD1 和 LOAD2 键重写仿真 RAM 的内容,不应该随意按下。

该电路中除 RS－232 接口外,全是 CMOS,单片机为 STOP 时电流很小。在仿真模式(模式 C),MC68HC05E0 为 STOP 模式,目标系统没有总线操作时,电源电流应小于几 μA(不包括 RS－232 接口或 LCD 驱动器的电流)。

电源电流与选用的 RAM 有关。MCM60L256 的备用 I_{CC} 通常小于 2μA。许多情况并不需要 64K RAM,只需选用能满足要求的 RAM 即可。6116 2K RAM 可用于 2～4K 应用场合,MCM60L64 可用于 8～16K 应用场合。若用 6064,其第二片选允许引脚(E2)应接高电平。串行装入程序要求回送传输至 RAM 的每个字节,因此,企图写入不存在的 RAM 地址将产生错误信息。若需要 16K 或更小,则两片 74HC245 处理地址,A14 和 A15 可省略。若只用一个存储

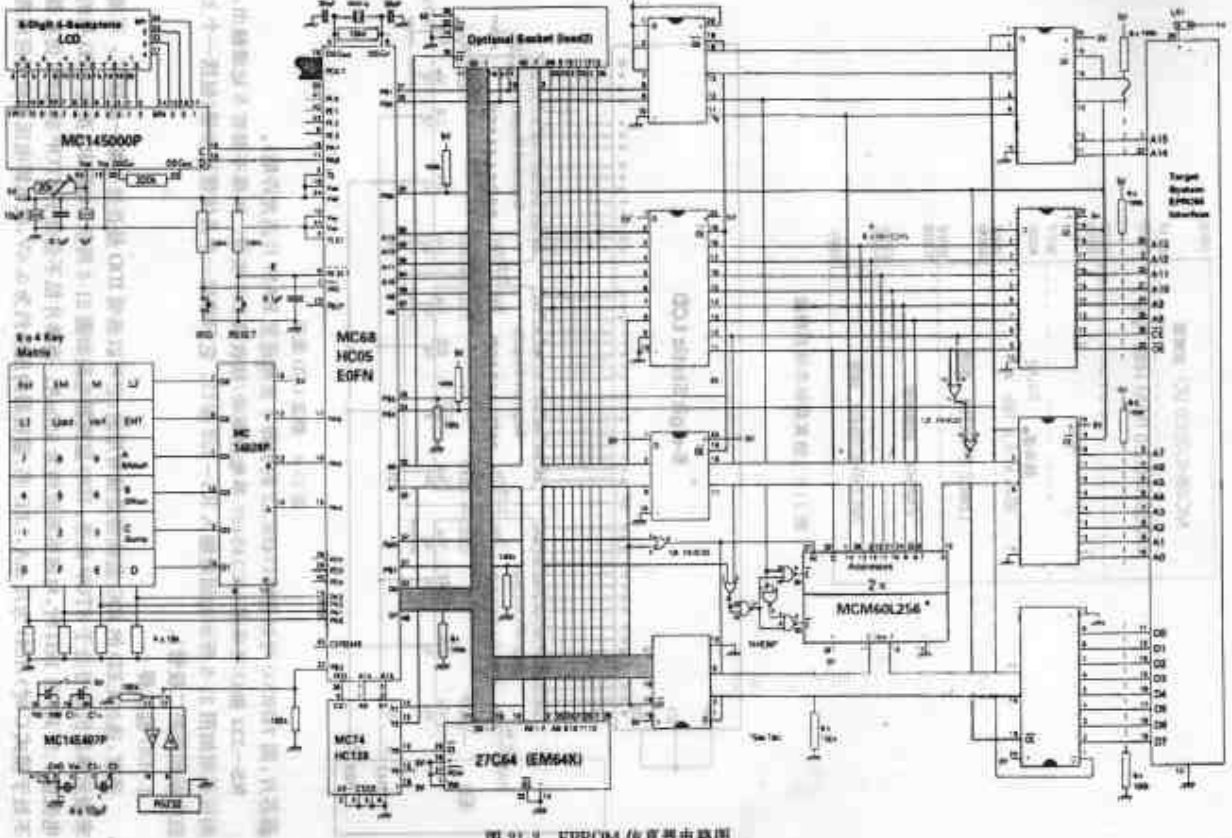


图 21-2 EPROM 仿真器电路图

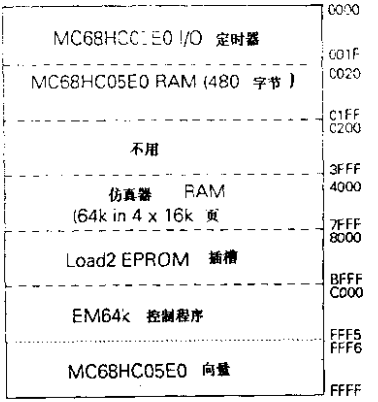


图 21-3 仿真器的存储器映象

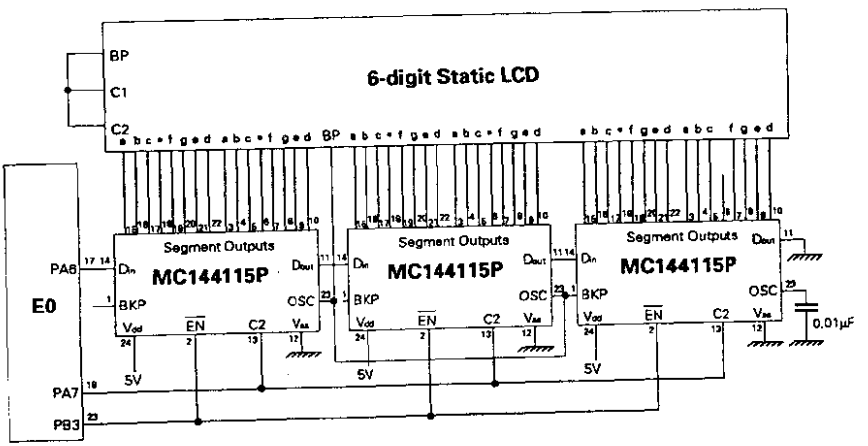


图 21-4 静态 LCD 显示

器芯片，则 74HC00 可以省略(74HC32 的引脚 3 直接连至 RAM 片选允许端)。

RS—232 接口可采用 MC145407 单电源驱动/接收器芯片实现。如果不需要 S 记录输出，则可采用如图 21-5 所示的固定输入 RS—232 接口。它只需要一个晶体管反相器(包括一个上拉电阻和保护二极管)。

1. IDD 监视器

通常，通过 LED 的 IDD 监视器是很有用的，它可以表明 IDD 是否处于某一值以下。可用来指示单片机是否处于 STOP 状态。IDD 监视器电路如图 21-6 所示。电阻 R1 决定 LED 亮的电流值。当 R1 为 1kΩ 时，电流临界值约为 500μA。当单片机不是处于 STOP 状态时(仿真器不处于模式 C 时)，IDD 为几 mA，LED 亮。监视器的压降约为 0.6V，故电源电压也应相应地提

高 0.6V。

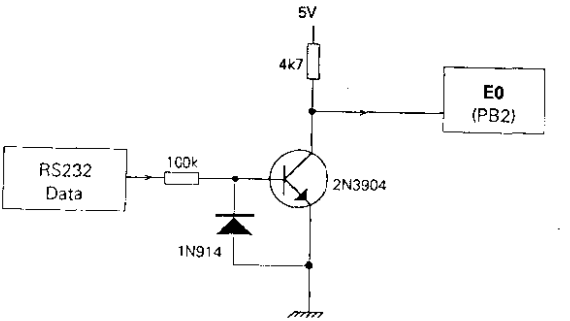


图 21-5　简单的固定输入 RS—232 接口

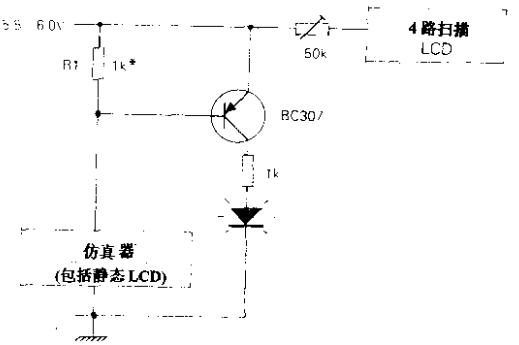


图 21-6　简单的 IDD 监视器

2. 地址陷阱

仿真器允许检验和改变存储器地址，但没有开发系统的断点和跟踪功能。增加图 21-7 所示的地址比较器可以用来指示目标系统运行的程序是否遇到开关所选择的地址。74HC74 用于锁存。为表明重复事件的发生，可以加单稳电路。

21.3　串行装入

为装入外部 S 记录，应按串行装入键(LOAD)，LCD 将显示“LOAD”。RS—232 接口以 9600 波特(8 位，无校验位)提供 S 记录。当接收到 S9 终止记录时，提示返回。如果在串行装入期间检测到错误，装入程序停止，并显示出错的地址和出错类型。错误类型如下：

1. 校验和错误，传输数据或接口错。

2. RAM 读返回错，RAM 错或不存在的地址。

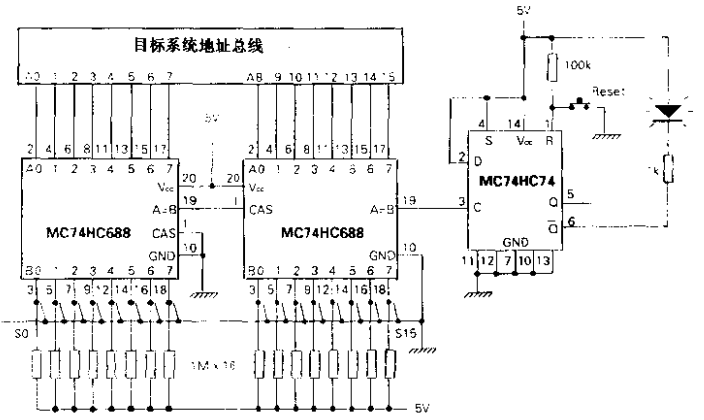


图 21-7　地址陷阱电路

3. 接收的 ASCII 字符小于 $30(零)。

4. 接收的 ASCII 字符位于 $39(9)至 $41(A)之间。

5. 接收的 ASCII 字符大于 $46(F)。

6. S 记录与仿真 RAM 比较时校验错误。

若使用仿真器时目标系统发生不正常,可用校验功能检验仿真 RAM。VERIFY(校验)功能与 LOAD 类似,但主要是为了比较 RAM 和 S 记录。

每个 S 记录的起始地址决定程序码位于目标系统的地址。该地址有时与程序码需要装入到仿真 RAM 的地址不同,因此,可能需要使用偏移量。偏移量字节用适当的键输入,可以是 $0100 的倍数。由 S 记录地址的 MSB 减去偏移量,修改后的地址是数据装入到仿真 RAM 的物理地址。在传送 S 记录之前,S 记录输出程序将偏移量加上。复位和上电时偏移量初始化为零。

所有用 MEMORY-MODIFY、BRAOFF 和 DUMP 输入的地址都使用目标系统的实际地址。若偏移量为零,这些地址与 RAM 物理地址相同。

21.4　并行装入

当使用并行装入功能时(LOAD1 和 LOAD2),OFFSET 对程序码装入 RAM 没有影响。当进行存储器修改,仍可以用来对 RAM 地址进行偏移,使之与目标系统程序的实际地址相对应。

21.5　27(C)64/128/256 仿真

仿真 27(C)512 时,使用所有 RAM。仿真较小 EPROM 时,需较小 RAM。若不用的高位地址为低,所用的存储器将位于 RAM 开始处。但使这些高位地址的某一位或某几位为高是很方便的。例如,只要程序装入 $E000～$FFFF,不改变硬件就能仿真 27(C)64。它允许向量位于目标系统存储器的顶部,通常是很方便的。若单片机的地址空间小于 64K,必须利用仿真器的偏移量特性将 S 记录装入 $E000～$FFFF。当从小 EPROM 装入时,VPP 和 PRG 引脚应正

确设置。EPROM 引脚如表 21-2 所示。

表 21-2 EPROM 引脚

引脚	27512	27256	27128	2764
1	A15	VPP	VPP	VPP
2	A12	A12	A12	A12
3	A7	A7	A7	A7
4	A6	A6	A6	A6
5	A5	A5	A5	A5
6	A4	A4	A4	A4
7	A3	A3	A3	A3
8	A2	A2	A2	A2
9	A1	A1	A1	A1
10	A0	A0	A0	A0
11	D0	D0	D0	D2
12	D1	D1	D1	D1
13	D2	D2	D2	D2
14	VSS	VSS	VSS	VSS
15	D3	D3	D3	D3
16	D4	D4	D4	D4
17	D5	D5	D5	D5
18	D6	D6	D6	D6
19	D7	D7	D7	D7
20	片选	片选	片选	片选
21	A10	A10	A10	A10
22	输出允许	输出允许	输出允许	输出允许
23	A11	A11	A11	A11
24	A9	A9	A9	A9
25	A8	A8	A8	A8
26	A13	A13	A13	Nc
27	A14	A14	PGM	PGM
28	VCC	VCC	VCC	VCC

21.6 目标系统接口

最简单方法是仿真器和目标系统公用一个电源。若采用单独电源时，引脚 28 不应该相连，并应注意两电源不应相差 0.5V 以上。

在仿真模式，目标系统控制 RAM(除 R/W 线)。可将 RAM 当作其 ROM 或 EPROM。在按下 IRQ 或 RESET 退出仿真模式之前，目标系统应停止操作，以便"EPROM"不再存在。

21.7 EM64K 程序

EM64K 控制程序小于 2K 字节，可位于 27C64 或 27C16。电路所示的是 27C64，并假设程序在 EPROM 起始处开始。EPROM 允许地址为 $C000(不用 A13 时为 $E000)。

EM64K 键盘功能如表 21-3 所示。

表 21-3 EM64K 功能

功 能	键	功 能 说 明
LOAD1	L1	由目标系统接口装入 RAM($4000～$7FFF)
LOAD2	L2	从第二插槽装入 RAM($8000～$BFFF)

续表

SERIEL LOAD	LOAD	通过 RS232 接口由 S 记录装入仿真 RAM，装入期间 LCD 显示“LOAD”
VERIFY	Verf	通过 RS232 接口，仿真 RAM 与 S 记录比较，LCD 显示“VfriFy”
EMULATE	EM	仿真模式，提示“EP”？去掉 EPROM，连接目标系统，置单片机于 EMULATE 模式
MEMORY MODIFY	M	显示/改变 RAM 单元。按下时显示最后地址。按 ENTER 显示该地址内容；或按 ENTER 后输入新数据。需改变时，ENTER 后输入新地址。按 ENTER 移至下一个地址，M 移至上一个地址，ESCAPE 退出。
ENTER	Ent	输入地址或数据（在 MEMORY MODIFY 移至下一个地址）
ESCAPE	Esc	从目前功能（OFFSET、BRAOFF、DUMP 或 MEMORY MODIFY）退出
BRAOFF	A	计算分支偏移量，需要分支指令地址和目的地址。若计算出有效分支，则写入存储器并显示。若无效，则显示“OR”表示超出地址范围。Esc 返回正常操作
OFFSET	B	允许输入仿真 RAM 地址的偏移量。从 S 记录指定的高位字节减去偏移量，DUMP 功能将偏移量加到地址上
DUMP	C	通过 RS232 接口输出仿真 RAM 内容。需 RAM 起始/终止地址。应在 ENTER 后输入，第二次 ENTER 后，开始输出 S 记录。
IRQ	IRQ	终止仿真并返回仿真监视器
RESET	RESET	仿真器复位，显示提示（口）。没有接收到 S9 记录时，RESET 可以使其从 LOAD 或 VERIFY 退出

21.8 软件

MC68HC05E0 的 D 口可用作通用 I/O 口，也可用来提供专用信号。该应用例子中使用 5 个专用功能。这些功能由 $12 地址的寄存器选择。这些功能是：P02、R/W、A13、A14 和 A15。P02 用于 74HC138 产生片选信号，R/W 用于控制仿真 RAM。

使用的唯一寄存器（除 I/O 数据和方向寄存器外）是中断控制寄存器（$0E）。程序中向该寄存器写入 $01 将中断标志清零（位 3），但 INTMX 仍置位，它允许外部中断。MC68HC05E0 的一个重要位是 XROM 位（即 $0C 的位 2），其省缺值为 1，正好适于本应用情况。当该位清零时，数据总线为固定输入。

MC68(HC)05 有 8 位变址寄存器，不能含有 16 位扩展地址。因此，装入和存储仿真器 RAM 都是用单片机 RAM 的小程序进行。该程序由扩展 LDA 或 STA 指令和随后的两个字节地址组成。4 字节程序位于 W2、ADDEH、ADDRL 和 W3 单元的 RAM 中。用程序中产生的地址可访问整个 64K 空间。

21.9 串行接口

带有 LOAD/DUMP 开关的 RS－232 电路如图 21-8 所示。它通过 PC COM 口或在由 RS－232连接的主机与终端之间使用仿真器的串行 LOAD 和 DUMP 程序。当使用 PC 时，应使用主机插槽。表 21-4 是 RS－232 接口的功能。

表 21-4 RS－232 接口功能和使用

设　置	功　能	S1	S2	说　明
主机和终端	Load	On	L	终端与主机相连，单片机检查从主机传送到终端的数据（引脚 3）
	Dump	On	D	终端与主机断开，S 记录发送到主机(2)和终端(3)
PC"COM"口	Load	On	L	从引脚 3 装入 S 记录
	Dump	On	D	S 记录发送到主机插槽的引脚 2（和终端槽的引脚 3）

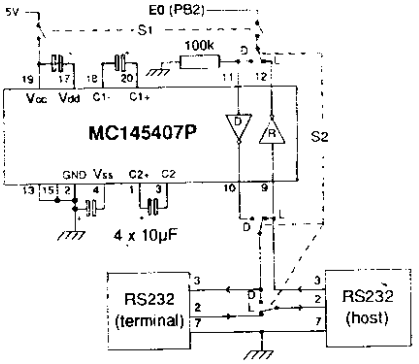


图 21-8 带有 LOAD/DUMP 开关的 RS－232 电路

21.10 EM64K 控制程序

```
PORTA     EQU     $00          口地址
PORTB     EQU     $01
PORTC     EQU     $02
PORTD     EQU     $03
PORTE     EQU     $04
PORTAD    EQU     $05          口数据方向寄存器(DDR)
PORTBD    EQU     $06
PORTCD    EQU     $07
PORTDD    EQU     $08
PORTED    EQU     $09
ICR       EQU     $0E          中断控制寄存器
PORTDSF   EQU     $0F          D 口功能选择寄存器
          ORG     $0020        RAM 定义
DTABL     RMB     6            LCD 缓冲器
TEMP      RMB     2
W1        RMB     1
W2        RMB     1            RAM 子程序 LDA 或 STA
ADDEH     RMB     1            RAM 子程序地址 MSB
ADDRL     RMB     1            RAM 子程序地址 LSB
```

```
W3          RMB     1       RAM 子程序 RTS
W4          RMB     1
W5          RMB     1
W6          RMB     1
ADDRH       RMB     1
STAT        RMB     1       状态字节
*                           2:实际地址;4:无效地址;
*                           5:校验;6:单独寄存器
CHKSUM      RMB     1       检验和
COUNT       RMB     1       位计数器
TMP1        RMB     1
TMP2        RMB     1
BCNT        RMB     1       S 记录字节数
ERTYP       RMB     1       错误类型
ERDAT       RMB     1       错误数据
OFF         RMB     1       S 记录偏移量
            RMB     185     未用
STACK       RMB     12      13 字节(一个中断和 4 层嵌套子程序)
SP          RMB     1
*
*空闲循环和检查按下键的子程序
            ORG     $E000
SCAN        JSR     KEYSCA  有键按下?
            BCC     SCAN    否,循环
            CLRX
            STA     W1      存按下的键的码
RJ          LDA     CTAB,X  寻取键码
            CMP     W1
            BEQ     PJ
            CMP     LAST
            BEQ     GETCMD
            INCX
            INCX
            INCX
            INCX
            BRA     RJ
PJ          LDA     #1
            STA     ICR
            INCX
            JMP     CTAB,X
*
*复位程序
START       LDA     #$E3    允许 D 口特殊功能:P02、R/W、A13、A14、A15
            STA     PORTDSF
            CLR     PORTA
            LDA     #$F0    显示/键盘
```

```
                STA     PORTAD
                LDA     # $58       模式 2,允许 144115 高
                STA     PORTB
                LDA     # $FB       位 0、1、3～7 为输出,位 2 为输入
                STA     PORTBD
                CLR     PORTC
                LDA     # $FF       均为输出
                STA     PORTCD
                CLR     PORTD
                LDA     # $1C       位 2、3、4 为输出,不用
                STA     PORTDD
                CLR     PORTE
                LDA     # $0F       位 0～3 为输出,不用
                STA     PORTED
                CLR     STAT
                CLR     ADDRL       初始化地址
                CLR     ADDRH
                CLR     OFF
GETCMD          LDA     # $58       模式 2
                STA     PORTB
                JSR     CLRTAB
                LDA     # $33       打印提示
                STA     DTABL
DSCN            JSR     DISTAB
                RSP
                BRA     SCAN
*
*键盘程序返回累加器中按下的键的码
KEYSCN          CLRA
                LDX     #6          设置行
KEY1            ADD     # $10
                STA     PORTA
COLUMN          LDA     PORTA       读键盘,并存储
                STA     W3
                BIT     # $0F       键闭合?
                BEQ     COLRET
                BSR     DBOUNC
                LDA     PORTA       重新读键盘
                CMP     W3          相同键闭合?
                BNE     COLRET      否
                SEC
COL1            LDA     PORTA
                BIT     # $0F       键释放?
                BNE     COL1        否,重试
                BSR     DBOUNC      是,去抖
                LDA     PORTA
```

```
                BIT     #$0F
                BNE     COL1
                LDA     W3
COLRET          BCS     KEY2            若有效,转 KEY2;否则转下一行
                DECX
                BNE     KEY1
KEY2            RTS
DBOUNC          LDA     #10             40ms
                STA     W6
DLP             LDA     #$FF
DLOOP           BRN     *               ≈256×12 周期
                BRN     *
                DECA
                BNE     DLOOP
                DEC     W6
                BNE     DLP
                RTS
*
*键盘表
CTAB            FCB     $51             P:从 EPROM 装入($4000)
                JMP     DUMP1
                FCB     $68             S:从 EPROM 装入($8000)
                JMP     DUMP9
                FCB     $28             C:S 记录输出
                JMP     PUNCH
                FCB     $52             L:装入 S 记录
                JMP     TLOAD
                FCB     $54             V:校验(S 记录)
                JMP     VERIFY
                FCB     $62             G:进入仿真模式
                JMP     MODE3           M:读/修改存储器
                FCB     $64
                JMP     MEMEX
                FCB     $38             B:地址偏移量
                JMP     OFFSET
LAST            FCB     $48             分支偏移量计算
                JMP     BRAOFF
*
STABL           FCB     $11             0
                FCB     $21             1
                FCB     $22             2
                FCB     $24             3
                FCB     $31             4
                FCB     $32             5
                FCB     $34             6
                FCB     $41             7
```

```
        FCB     $42             8
        FCB     $44             9
        FCB     $48             A  分支偏移量
        FCB     $38             B  装入偏移量
        FCB     $28             C  输出S记录
        FCB     $18             D
        FCB     $14             E
        FCB     $12             F
        FCB     $61             10 Esc取消命令
        FCB     $58             11 E  输入命令
        FCB     $68             12 S  从$8000装入
        FCB     $64             13 M  存储器检验/改变
        FCB     $62             14 G  仿真
        FCB     $54             15 V  校验RAM
        FCB     $52             16 L  装入RAM
        FCB     $51             17 P  从$4000装入
*
*分别在TEMP、TEMP+1和ADDRH、ADDRL建立开始和结束地址
BLDRNG  BCLR    4,STAT
        BCLR    2,STAT          仿真地址
        JSR     CLRTAB          打印"BA"
        LDA     #$F4
        STA     DTABL+4
        LDA     #$77
        STA     DTABL+5
        JSR     DISTAB
        JSR     BLDADR          取得源地址
        BCC     BLDRN1          有效?
        LDA     ADDRH
        STA     TEMP            是,存储
        LDA     ADDRL
        STA     TEMP+1
        JSR     LOAD            寻取指令的操作码
        STA     W6              存入
        JSR     CLRTAB
        LDA     #$F1            打印"EA"
        STA     DTABL+4
        LDA     #$77
        STA     DTABL+5
        JSR     DISTAB
        JSR     BLDADR          取得目的地址
        BCC     BLDRN1
        LDA     ADDRH           有效?
        RTS
BLDRN1  BSET    4,STAT          无效
        RTS
```

```
*
*显示信息
DISP    STX     W6
        CLR     COUNT
DISLP   LDX     W6
        LDA     DLOAD,X
        LDX     COUNT
        STA     DTABL,X
        INC     W6
        INC     COUNT
        LDA     COUNT
        CMP     #6
        BLO     DISLP
        JMP     DISTAB
DLOAD   FCB     0,0,$D0,$D7,$77,$E6
VERF    FCB     $D6,$F1,$60,$06,$71,$B6
*
*计算分支偏移量
BRAOFF  BSR     BLDRNG
        BRSET   4,STAT,ORET
        LDA     ADDRL
        SUB     #2
        STA     ADDRL
        LDA     ADDRH
        SBC     #0
        STA     ADDRH
        LDA     ADDRL
        SUB     TEMP+1
        STA     ADDRL
        LDA     ADDRH
        SBC     TEMP
        STA     ADDRH
        LDA     W6          检验操作码
        CMP     #$1F
        BLS     OFFST1
        LDA     ADDRH
        CMP     #$FF        加或减偏移量
        BEQ     OFFST2
        TSTA
        BNE     OVRERR
        LDA     ADDRL
        CMP     #$7F
        BHI     OVRERR
        BRA     OK1
OFFST2  LDA     ADDRL
        CMP     #$FF
```

```
        BEQ     OVRERR
        CMP     #$80
        BLO     OVRERR
OK1     BSR     USE             若有效则打印
        JMP     SCAN
ORET    JMP     GETCMD
USE     JSR     CLRTAB
        LDA     #$D6            打印"USED"
        STA     DTABL
        LDA     #$B5
        STA     DTABL+1
        LDA     #$F1
        STA     DTABL+2
        LDA     #$E6
        STA     DTABL+3
        LDA     ADDRL           打印
        JSR     PRTDAT
        TAX
        LDA     TEMP+1
        ADD     #1
        STA     ADDRL
        LDA     TEMP
        ADC     #0
        STA     ADDRH
        TXA
        BCLR    2,STAT
        JMP     STORE
OFFST1  LDA     ADDRL           调整位分支
        SUB     #1
        STA     ADDRL
        LDA     ADDRH
        SBC     #0
        STA     ADDRH
        CMP     #$FF            偏移量取反?
        BEQ     OFFST3
        TSTA    CHECK FOR
        BNE     OVRERR
        LDA     ADDRL
        CMP     #$7F
        BHI     OVRERR
        BRA     OK2
OFFST3  LDA     ADDRL
        CMP     #$FE
        BHS     OVRERR
        CMP     #$80
        BLO     OVRERR
```

```
OK2         INC     TEMP+1
            BNE     OFFITS
            INC     TEMP
OFFITS      BSR     USE             若有效则打印
            BRA     SCJMP
OVRERR      LDA     #$D7            打印"OR"
            STA     DTABL+4
            LDA     #$60
            STA     DTABL+5
            JSR     PRTADR
SCJMP       JMP     SCAN
*
*显示表内容
DISTAB      BCLR    3,PORTB         MC144115 的 EN 为低
            LDX     #5
DISCHR      LDA     DTABL,X
NT1         STX     W1              存入变址
            BCLR    6,PORTA         数据清零
            LDX     #8
DIS1        LSLA
            BCC     DIS2
            BSET    6,PORTA
DIS2        BSET    7,PORTA
            BCLR    7,PORTA
            BCLR    6,PORTA
            DECX
            BNE     DIS1
            LDX     W1              恢复变址
            DECX
            BPL     DISCHR
            BSET    3,PORTB         MC144115EN 为高
            RTS
*
*S 记录输入 RAM 访问
RAMACC      BSET    1,PORTB         5   5   XFER A14 & A15 TO PORTB
            BRSET   7,ADDRH,A15H    5   10  A15 HIGH?
            BCLR    1,PORTB         5   15  NO
A15H        BSET    0,PORTB         5   20  YES
            BRSET   6,ADDRH,A14H    5   25  A14 HIGH?
            BCLR    0,PORTB         5   30  NO
A14H        LDX     ADDRH           3   33  YES
            STX     ADDEH           4   37
            BSET    6,ADDEH         5   42  A14 HIGH
            BCLR    7,ADDEH         4   47  A15 LOW
            BRSET   5,STAT,L3       5   52  读?
            LDX     #$C7            2   54  否,写入
```

```
                STX     W2              4  58  存入
                JSR     W2              16  74  RAM 子程序
                STA     W4              4  78
L3              LDX     #$C6            2  80
                STX     W2              13  93
                JMP     W2              14  107
*显示表清零
CLRTAB          LDX     #5
CLRLOC          CLR     DTABL,X
                DECX
                BPL     CLRLOC
                RTS
*
*仿真模式
MODE3           JSR     CLRTAB
                LDA     #$F1            E
                STA     DTABL
                LDA     #$73            P
                STA     DTABL+1
                LDA     #$63            ?
                STA     DTABL+3
                JSR     DISTAB
KSC             JSR     KEYSCN          等待直到 EPROM 去除
                BCC     KSC
                CMP     #$62            仿真确认?
                BEQ     CONF
                JMP     GETCMD
CONF            LDA     #$28            模式 3,144115EN 为高
                STA     PORTB
                LDA     #$F1            E
                STA     DTABL
                LDA     #$D6            U
                STA     DTABL+1
                LDA     #$D0            L
                STA     DTABL+2
                LDA     #$77            A
                STA     DTABL+3
                LDA     #$F0            T
                STA     DTABL+4
                LDA     #$F1            E
                STA     DTABL+5
                JSR     DISTAB
STP             STOP
*
*EPROM 内容从仿真插槽传送至 RAM
DUMP1           JSR     CLRTAB
```

```
        JSR     DISTAB
        LDA     #$58            模式 2,144115EN 为高
        STA     PORTB
        BSET    2,STAT          实际地址
        CLR     ADDRL
        CLR     ADDRH
LLP1    BSET    1,PORTB         5  5  A14 和 A15 为 PORTB
        BRSET   7,ADDRH,AD15H   5  10 A15 为高?
        BCLR    1,PORTB         5  15 否
AD15H   BSET    0,PORTB         5  20 是
        BRSET   6,ADDRH,AD14H   5  25 A14 为高?
        BCLR    0,PORTB         5  30 否
AD14H   LDX     ADDRH           3  33 是
        STX     ADDEH           4  37
        BCLR    7,ADDEH         5  42 A15 为低
        BSET    6,ADDEH         5  47 A14 为高
        BCLR    6,PORTB
        BSET    7,PORTB         从仿真器读入
        JSR     LOAD            装入字节
        BSET    6,PORTB
        BCLR    7,PORTB         写入 RAM
        JSR     STORE           存储字节
        INC     ADDRL
        BNE     SKPH
        INC     ADDRH
SKPH    LDA     ADDRH
        BNE     LLP1            MSB 为零?
        LDA     ADDRL           是,LSB 为零?
        BNE     LLP1            是,结束
        JMP     GETCMD
*
*S 记录装入/发送的偏移量
OFFSET  BSET    6,STAT          没有地址增/减
        BSET    2,STAT          实际地址
        LDA     #$D7            O
        STA     DTABL
        LDA     #$71            F
        STA     DTABL+1
        STA     DTABL+2
        CLR     DTABL+3
        CLR     ADDEH
        CLR     ADDRH
        LDA     #0FF
        STA     ADDRL
        JMP     MEMEX3
*
```

```
*EPROM 内容从辅助插槽($8000)传至 RAM
DUMP9   JSR     CLRTAB
        JSR     DISTAB
        LDA     #$58        模式 2,144115EN 为高
        STA     PORTB
        BSET    2,STAT      实际地址(无偏移量)
TLOP    BSR     T19
        INC     PORTB       下一页
        LDA     PORTB
        AND     #3
        CMP     #3          最后一页
        BLO     TLOP
        BSR     T19         是
        JMP     GETCMD
T19     CLR     ADDRL
        CLR     ADDRH
LLP9    LDX     ADDRH       是
        STX     ADDEH
        BSET    7,ADDEH     A15 高
        BCLR    6,ADDEH     A14 低
        JSR     LOAD        从辅助插槽读
        BCLR    7,ADDEH     A15 低
        BSET    6,ADDEH     A14 高
        JSR     STORE       写入仿真 RAM
        INC     ADDRL
        BNE     SKPH9
        INC     ADDRH
SKPH9   LDA     ADDRH
        CMP     #$40        最后地址 $3FFF
        BNE     LLP9        结束?
        RTS
*
*RS232(9600 波特)S 记录接收器(4MHz)
VERIFY  BSET    5,STAT      串行校验
        LDX     #6
        BRA     L4
TLOAD   BCLR    5,STAT      串行装入
        CLRX
L4      LDA     #$81        RTS
        STA     W3
        JSR     DISP        显示"LOAD OR VEriFy"
INPUT   BSR     INCHD
        CMP     #'S'        是 S?
        BNE     INPUT       否
        BSR     INCHD       是,取下一个字符
        CMP     #'9'        9?
```

```
        BEQ     NINE            是,结束
        CMP     #'1'            为1?
        BNE     INPUT
LNGTH   CLR     CHKSUM          是,检验和清零
        CLR     TMP2
        JSR     BYTEI           存储并取字节数
        STA     BCNT
ADDR    JSR     BYTEI
        SUB     OFF
        STA     ADDRH
        JSR     BYTEI
        STA     ADDRL
DLOP    JSR     BYTEI           75      取一个字节
        BEQ     CHCK            3  78   最后一个字节?
        BRCLR   5,STAT,L5       5  83   否,校验?
        STA     W6              4  87   是
        JSR     RAMACC          66 153  读RAM
        CMP     W6              3  156  相同?
        BNE     ERR?            3  159
        BRA     L6              3  162
L5      JSR     RAMACC          67 150  否,写入RAM
        CMP     W4              3  153  读回
        BNE     ERR2            3  156  OK?
L6      INC     ADDRL           5  161  5  167 增加低地址
        BNE     NOOVR           3  164  3  170 溢出?
        INC     ADDRH           5  169  5  175 是,增加高地址
NOOVR   BRA     DLOP            3  172  3  178
*
*检验和字节和出错程序
CHCK    ADD     CHKSUM
        STA     CHKSUM
        CMP     #$FF
        BEQ     INPUT
        LDX     #1
ERR     STA     ERDAT
        STX     ERTYP
        TXA
        JSR     PRTDAT
        CLR     DTABL+4
        JSR     PRTADR
        JMP     SCAN
*
*输入程序,MC68HC05E0:0.5μs。在9600波特每位周期数:208
INCHD   BSR     DEL191          191
INCH    BRCLR   2,PORTB,*         5
        BRSET   2,PORTB,*         5
```

```
            LDX     #7              2   6
            STX     COUNT           4   10
            BSR     DEL110          110 120
INBT        BSR     DEL191          191
            BRCLR   2,PORTB,ZER     5   196
ZER         RORA                    3   199
            DEC     COUNT           5   204
            BNE     INBT            3   207
            LSRA                    3   16
            RTS                     6   22
NINE        JMP     GETCMD
ERR2        LDX     #2              从 RAM 读回
            BRA     ERR
ERR3        LDX     #3              小于 ASCII 码 0
            BRA     ERR
ERR4        LDX     #4              位于 ASCII 码 9 和 A 之间
            BRA     ERR
ERR5        LDX     #5              大于 ASCII 码 F
            BRA     ERR
ERR7        LDX     #7              校验有错
            BRA     ERR
*
*字节输入子程序
BYTEI       BSR     INCHD           22      高 4 位
            BSR     ASCII           35  57
            LSLA                    3       移位并存入
            LSLA                    3
            LSLA                    3
            LSLA                    3   69
            STA     TMP1            4   71
            LDA     TMP2            3   74
            ADD     CHKSUM          3   79
            STA     CHKSUM          4   83
            BSR     INCHD           22      低 4 位
            BSR     ASCII           35  57
            ADD     TMP1            3   60  加到高 4 位上
            STA     TMP2            4   64
            DEC     BCNT            5   69  字节计数器减 1
            RTS                     6   75
ASCII       CMP     #$30            2       小于 0?
            BLO     ERR3            3   5   是,非法
            CMP     #$39            2   7   大于 9?
            BHI     MT9             3   10
            SUB     #$30            3   13  0~9,转换为 16 进制
            RTS                     6   19
MT9         CMP     #$41            2   12  小于 A?
```

```
                BLO     ERR4            3  15   是,非法
                CMP     #$46            2  17   大于F?
                BHI     ERR5            3  20   是,非法
                SUB     #$37            3  23   A~F转换为16进制
NFND            RTS                     6  29
DEL191          LDX     #29             2
                BRA     DELAY           3  5
DEL110          LDX     #16             2
DELAY           DECX                    3
                BNE     DELAY           3      6X
                RTS                     6      12+6X(INC BSR)
*
*RS232(9600/4MHz)S记录发送
PUNCH           BSET    2,PORTBD        位2为输出
                JSR     BLDRNG
                BRSET   4,STAT,NINE     新地址?
                LDX     TEMP            否
                STA     TEMP
                STX     ADDRH
                LDA     ADDRL
                LDX     TEMP+1
                STX     ADDRL
                STA     TEMP+1
                BCLR    7,STAT          结束标志清零
                BCLR    2,STAT          仿真地址
LOOP1           LDA     TEMP+1          最后LSB
                SUB     ADDRL           目前LSB
                STA     W6
                LDA     TEMP            结束MSB
                SBC     ADDRH           目前MSB
                BNE     LOTS            MSB为零?
                LDA     W6              是,查看LSB
                INCA
                BEQ     LOTS            是$FF?
                CMP     #$20            大于23?
                BHI     LOTS
                BSET    7,STAT          否,最后S1记录
                BRA     LTE20           小于等于20
LOTS            LDA     #$20
LTE20           ADD     #$03            加字节数和地址
                STA     BCNT
                LDA     #'S'
                JSR     OUCH
                LDA     #'1'
                BSR     OUCH
                CLR     CHKSUM
```

```
                LDA     BCNT            字节数
                BSR     BYTEO
                LDA     ADDRH           高位地址
                BSR     BYTEO
                LDA     ADDRL
                BSR     BYTEO
LOOP2           JSR     LOAD            取字节
                INC     ADDRL           增加地址
                BNE     NOVR            溢出
                INC     ADDRH           是,增加高位字节
NOVR            BSR     BYTEO           发送字节
                BNE     LOOP2           最后字节?
*
*检验和字节
                LDA     CHKSUM
                COMA
                BSR     BYTEO
                BSR     CRLF
                BRCLR   7,STAT,LOOP1
*
*S9 记录
                LDA     #'S'            5
                BSR     OUCH
                LDA     #'9'            9
                BSR     OUCH
                LDA     #$03            3 字节
                BSR     BYTEO
                LDA     #$00
                BSR     BYTEO
                LDA     #$00
                BSR     BYTEO           地址
                LDA     #$FC
                BSR     BYTEO           检验和
                BSR     CRLF
                BCLR    2,PORTBD        位 2 为输入
                JMP     GETCMD
CRLF            LDA     #$0D            CR
                BSR     OUCH
                LDA     #$0A            LF
*
*输出程序,每位 208 周期
OUCH            BSET    2,PORTB         5      确定为高
                LDX     #10             2  7   发送 10 位
                STX     COUNT           4  11  起始位、8 个数据位、停止位
                NOP
                NOP
```

```
                JSR     DELAY       72  83
                CLC                 2   85  开始为零
                BRA     STAR        3   88
OUTBT           LDX     #28         2
                JSR     DELAY       180 182
                NOP                 2   184
DEL3            SEC     2   186
                RORA                3   189
STAR            BCS     OU1         3   192
                BCLR    2,PORTB     5   197
                BRA     OBD         3   200
OU1             BSET    2,PORTB     5
                BRA     OBD         3
OBD             DEC     COUNT       5   205
                BNE     OUTBT       3   208
                RTS
ASCIO           ADD     #$30        3       转换为 ASCII 码
                CMP     #$39        2   5       0～9?
                BLS     NMT9        3   8
                ADD     #$07        3   8   否,A～F
MMT9            RTS                 6   14
*
*字节输出子程序
BYTEO           STA     TMP1
                LSRA                        右移获得 MSB 并转换
                LSRA
                LSRA
                LSRA
                BSR     ASCIO
                BSR     OUCH
                LDA     TMP1
                ADD     CHKSUM
                STA     CHKSUM
                LDA     TMP1
                AND     #$0F            LSB
                BSR     ASCIO           转换
                BSR     OUCH
                DEC     BCNT            字节数减 1
                RTS
*
*MC145000段译码
CTABL           FCB     $D7     0
                FCB     $06     1
                FCB     $E3     2
                FCB     $A7     3
                FCB     $36     4
```

```
                FCB             $B5         5
                FCB             $F5         6
                FCB             $07         7
                FCB             $F7         8
                FCB             $B7         9
                FCB             $77         A
                FCB             $F4         B
                FCB             $D1         C
                FCB             $E6         D
                FCB             $F1         E
                FCB             $71         F
ERROR           JSR             CLRTAB
                LDA             #$F1
                STA             DTABL+1
                LDA             #$60
                STA             DTABL+2
                STA             DTABL+3
                JMP             DSCN
*
*输入一个字符,A、X 寄存器含有 16 进制值
CHRIN           JSR     KEYSCN
                BCC     CHRIN
                CLRX
CHRIN1          CMP     STABL,X
                BEQ     CHRIN2
                INCX
                BRA     CHRIN1
CHRIN2          TXA
                RTS
*
*存储器检查/改变
MEMEX           BCLR    2,STAT      仿真地址
                JSR     GETADR      取地址
                CMP     #$10
                BEQ     MEMEX4
MEMEX3          BSR     LOAD        装入数据
                JSR     PRTDAT      打印
                JSR     GETNYB      取新的 4 位
                CMP     #$10
                BEQ     MEMEX4
                CMP     #$11
                BEQ     ADRINC
                CMP     #$13        存储器?
                BEQ     ADRDEC
                CMP     #$0F
                BHI     CMDMDL      有效 16 进制?
```

```
MEMEX1      JSR     PRTDAT      打印
            JSR     GETBY2      移入下一个字节
            BCS     MEMEX1
CMDMDL      CMP     #$11
            BNE     MEMEX2
            LDA     W2          恢复 ACCA
            BSR     STORE       存储
            BCS     MEMEX3      存储有效?
ADRINC      BRSET   6,STAT,MEMEX4
            INC     ADDRL
            BNE     MEMEX5
            INC     ADDRH
MEMEX5      JSR     PRTADR
            BRA     MEMEX3      重复
MEMEX2      CMP     #$13        M?
            BNE     MEMEX4      否
            LDA     W2
            BSR     STORE
            BCS     MEMEX3
ADRDEC      BRSET   6,STAT,MEMEX4
            TST     ADDRL       是,则取以前地址
            BNE     CMDMB2
            DEC     ADDRH
CMDMB2      DEC     ADDRL
            JSR     PRTADR      打印
            BRA     MEMEX3      重复
MEMEX4      BRCLR   6,STAT,NORM2
            CLR     ADDRL
NORM2       BCLR    6,STAT
            JMP     GETCMD
*
*在地址 ADDRH(EH)、ADDRL 装入/存入
STORE       LDX     #$C7        设置程序执行两字节存储
            BSR     LDSTCM
            STA     W4
            BSR     LOAD
            CMP     W4
            BEQ     STRTS
            SEC
STRTS       RTS
LOAD        LDX     #$C6        设置程序执行两字节装入
LDSTCM      STX     W2
            LDX     #$81
            STX     W3
            BRSET   2,STAT,NORM 实际地址?
            STA     W4
```

```
                LDA     ADDRH
                SUB     OFF
                STA     W5
                LDA     W4
RMCC            BSET    1,PORTB
                BRSET   7,W5,A15HI
                BCLR    1,PORTB
A15HI           BSET    0,PORTB
                BRSET   6,W5,A14HI
                BCLR    0,PORTB
A14HI           LDX     W5
                STX     ADDEH
                BSET    6,ADDEH
                BCLR    7,ADDEH
NORM            JMP     W2
*
*建立一个字节
GETBY2          STA     W2
                BSR     GETNYB
                BCC     GETBRT
                ASL     W2
                ASL     W2
                ASL     W2
                ASL     W2
                ORA     W2
                SEC
GETBRT          RTS
*
*ACCA 获得一个字符。若为 16 进制,C 置位
GETNYB          JSR     CHRIN           取得字符
                CLC
                CMP     #$0F            有效 16 进制?
                BHI     GETRET
                SEC
GETRET          RTS
*
*建立地址,若为新地址则 C 置位
GETADR          JSR     CLRTAB          空显示
                BSR     PRTADR
BLDADR          BSR     GETNYB          取字符
                BCS     GETAD1          有效 16 进制?
                CMP     #$10
                BEQ     GETRTS
                CMP     #$11
                BEQ     GETRTS
                BRA     GETADR
```

```
GETAD1    CLR     ADDRH       初始化高位地址
          STA     ADDRL
          BSR     PRTADR      打印新地址
GETALP    BSR     GETNYB      取另一个字符
          BCC     GETARG      有效?
          ASLA                是,移位
          ASLA
          ASLA
          ASLA
          LDX     #4
GETASF    ASLA
          ROL     ADDRL
          ROL     ADDRH
          DECX
          BNE     GETASF
          BSR     PRTADR      打印新地址
          BRA     GETALP      取另一个字符
GETARG    CMP     #$10
          BEQ     GETRTS
          CMP     #$11
          BNE     GETALP
          SEC
GETRTS    RTS                 置标志位
*
*打印一个字节。打印地址 ADDRH、ADDRL
PRTDAT    LDX     #4          在最后两个 LCD 位打印
PRTBYT    STX     W1
          STA     W4
          LSRA
          LSRA
          LSRA
          LSRA
          TAX
          LDA     CTABL.X
          LDX     W1
          STA     DTABL.X
          LDA     W4
          AND     #$0F
          TAX
          LDA     CTABL.X
          LDX     W1
          STA     DTABL+1.X
          JSR     DISTAB
          LDA     W4
          RTS
PRTADR    STA     W5          打印地址(前 4 位)
```

```
                STX     W3
                LDA     ADDRH
                CLRX
                BSR     PRTBYT
                LDA     ADDRL
                LDX     #2
                BSR     PRTBYT
                LDA     W5
                LDX     W3
                RTS
*
* MC68HC05E0 向量
                ORG     $FFF4
                FDB     START           串行口
                FDB     START           定时器 B
                FDB     START           定时器 A
                FDB     GETCMD          外部中断
                FDB     START           SWI
                FDB     START           RESET
                END
```

22. 用 M6805/M68HC05 驱动 LCD

M6805 单片机片内具有 A/D、D/A、串行口、定时器及显示驱动器。MC68HC05M4 具有高达 40V 的荧光显示驱动能力，MC68HC05L 型号均具有 LCD 驱动器。MC68HC05 L7/L9 具有 8 或 16 个后板输出和 60/40 个前屏输出，可驱动高达 640/960 个 LCD 段/点阵。用 3 片 MC68HC68L9 LCD 扩展器可使 L9 的前屏输出扩展到 205 个。没有专门 LCD 电路的单片机可用来直接驱动单后板(backplane)信号的 LCD，但要用软件资源。可用 LCD 显示驱动器芯片与单片机接口。这一部分给出各种 LCD 驱动显示的硬件和软件。这些方法也同样适用于其它系列的单片机(如 M68HC11)系列、M6801 系列等)和其他型号的 LCD 驱动器芯片。

22.1 单后板信号显示

单后板信号显示通常用在所需段数有限的情况。它与多路扫描显示相比的优点是：对比度和可视角度好，工作电压和温度范围宽。可由单片机直接驱动，驱动的段数受可设置为输出的引脚数限制。I/O 口直接装入所需显示对应的段数据。这种情况下，单片机必须在规定的间隔提供信号(后板和前屏)，以便满足显示器接收具有较小直流分量的 AC 波形的需要。

另一种驱动单后板 LCD 的方法是使用 LCD 驱动器，图 22-1 是 6 位 7 段 LCD 显示电路，它使用 3 片 MC144115 LCD 驱动器。这种芯片与单片机采用串行方式连接。用这种方法可以级联更多的 LCD 驱动器芯片，以增加 LCD 位数。MC144115 串行接口有三条线：时钟、数据和片选($\overline{EN}$)。若每个外围器件都有单独的允许线，时钟和数据线可以和其他外围器件公用。如果

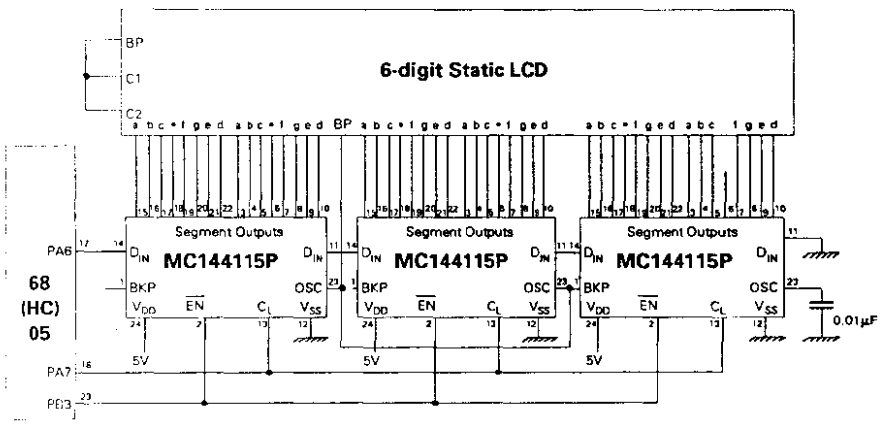


图 22-1 使用 MC144115 LCD 驱动器的 LCD 显示电路

没有其他芯片公用时钟和数据，则片选允许线可以由时钟驱动。LCD 显示的程序如下：

```
PORTA    EQU    $00    PORT A ADDRESS
PORTB    EQU    $01    PORT B ADDRESS
```

```
            ORG     $0050
R           RMB     6           WORKING NUMBER
TMP1        RMB     1           POSITION OF LSB
TMP2        RMB     1           POSITION OF MSB
TMP3        RMB     1
TMP4        RMB     1
            ORG     $1000
*取得显示 LCD 的 BCD 数据对应的段码
DISP        LDA     #$05
            STA     TMP3
            LDX     TMP1        LSB
D3          LDA     0,X
            STX     TMP4
            TAX
            LDA     STABL,X     FIND 7 SEGMENT CODE
            LDX     TMP3
            STA     R,X         PUT IN DISPLAY TABLE
            DEC     TMP3
            LDX     TMP4
            DECX
            CPX     TMP2        FINISHED?
            BNE     D3
*发送显示驱动器需要的 48 位数据
OUTT        BCLR    3,PORTB     ENABLE LOW
            LDX     #5          SEND DISPLAY TABLE TO 144115
DISCHR      LDA     R,X
DISPLY      STX     TMP3        SAVE INDEX
            BCLR    6,PORTA     CLEAR DATA
            LDX     #8
DISI        LSRA                SET UP
            BCC     DIS2        BIT OF
            BSET    6,PORTA     ACCUMULATOR
DIS2        BSET    7,PORTA     CLOCK
            BCLR    7,PORTA     IT
            BCLR    6,PORTA     CLEAR DATA
            DECX                COMPLETE?
            BNE     DIS1        NO
            LDX     TMP3        RESTORE INDEX
            DECX
            BPL     DISCHR
            BSET    3,PORTB     ENABLE HIGH
            RTS
*LCD 段码表
STABL       FCB     $EB         0  SEGMENT
            FCB     $60         1
            FCB     $C7         2  CODES
```

```
FCB     $E5     3
FCB     $6C     4   FOR THE
FCB     $AD     5
FCB     $AF     6   MC145000/144115
FCB     $E0     7
FCB     $EF     8   LCD DRIVER
FCB     $ED     9
```

22.2 三后板信号 LCD 显示

MC68HC05L6 可驱动 24 个前屏和 3 或 4 个后板信号(由软件选择)。其数据格式如下:

LCD 数据锁存器 OO	7	6	5	4	3	2	1	0
($00)	Bp1	Bp2	Bp3	Bp4	Bp1	Bp2	Bp3	Bp4
	Fp01				Fp02			

可见每个前屏占据 12 字节 RAM 的半个字节。这样,RAM 单元与 4 后板 7 段显示具有简单对应关系:每个数字(两个前屏信号)对应一个 RAM 字节。对于三个后板信号的显示,每个数字对应 3 个半字节(1.5 字节),因此,将段码转换为显示 RAM 数据的软件更加复杂。后面给出具体程序。图 22-2 是 MC68HC05L6 的 LCD 显示结构图。

当使用没有 LCD 驱动能力的单片机时,可用显示驱动器来驱动多路 LCD。图 22-3 是采用 ICM7231B 驱动器的 LCD 显示电路。ICM7231B 要求每个字符通过引脚 A0、A1 和 A2 寻址,数据要求写入到引脚 D0～D3。这种并行控制方法比使用并行结构的驱动器(MC145000、MC145001、MC144115 等)要用较多的 I/O 引脚。

数据以 16 进制接收,由驱动器译码成段信号,这样使软件简化,但只能显示驱动器的 16 个字符。ICM7231B 可显示 0～9、一、E、H、L、P 和熄灭,ICM7231A 可显示 0～9、A～F。

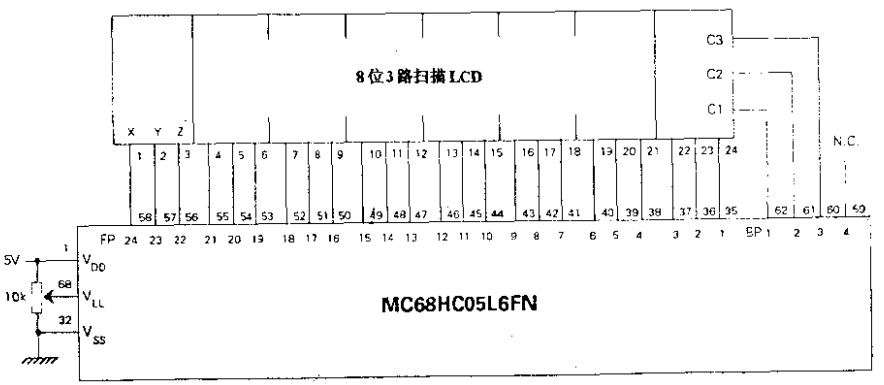


图 22-2 MC68HC05L6 的 LCD 显示结构

对任何多路扫描 LCD 驱动器,对比度取决于驱动器有关电路的电源电压。在 ICM7231 情况下,可用图 22-3 中引脚 2 的电位器来调整对比度。

1. 用 MC68HC05L6 直接驱动 LCD 显示器的程序

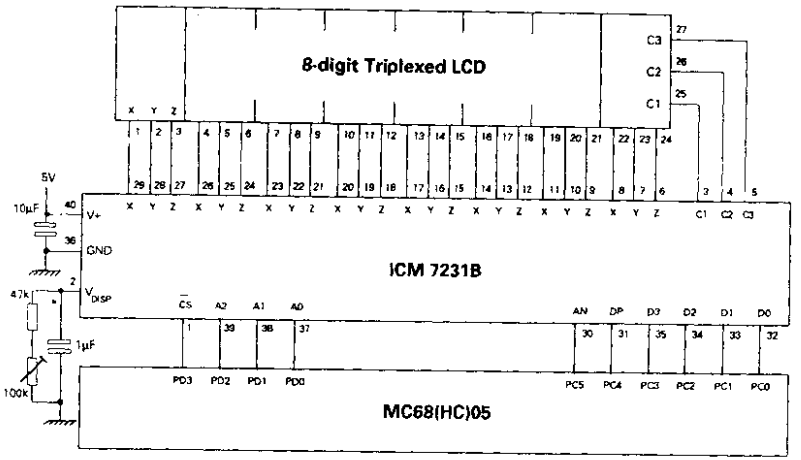


图 22-3　采用 ICM7231 的 LCD 显示电路

```
LADD    EQU     $0009       LCD ADDRESS REGISTER
LDAT    EQU     $0008       LCD DATA REGISTER
        ORG     $0050
Q       RMB     8           DISPLAY REGISTER
W1      RMB     1
W2      RMB     1
        ORG     $0100
*
*LCD 段码表
L6TAB   FCB     $AC,$CA     0
        FCB     $00,$C0     1
        FCB     $E4,$8E     2
        FCB     $E0,$CE     3
        FCB     $48,$C4     4
        FCB     $E8,$4E     5
        FCB     $EC,$4E     6
        FCB     $80,$C8     7
        FCB     $EC,$CE     8
        FCB     $E8,$CE     9
        FCB     $40,$04     -   (A:CC CC)
        FCB     $EC,$0E     E   (B:6C 46)
        FCB     $4C,$C4     H   (C:AC 0A)
        FCB     $2C,$02     L   (D:64 C6)
        FCB     $CC,$8C     P   (E:EC 0E)
        FCB     $00,$00         (F:CC 0C)
*
```

```
* MC68HC05L6 直接驱动 LCD 的主程序
* 该程序假设 Q 含有显示 HEX 数据。每个字符需要 1.5 个字节(9 位含在 3 个半字节(nibble)
中,每个半字节中有一位(后板)没有
* 每执行一次循环处理 2 个字符
START   CLR   W2          Initialise digit pointer.
        LDA   # $80       First write to $09;Bus/LCD ratio=256.
        STA   LADD        4-backplane. fast charge enabled.
L60P    LDX   W2
        LDA   Q,X         Get HEX data.
        AND   # $0F       Only lower nibble is relevant.
        LSLA              ×2(two bytes per digit in table).
        TAX
        LDA   L6TAB,X     Get first byte from segment table.
        STA   LDAT        Send it to LCD data latch.
        INC   LADD        Keep LCD on. move to next latch.
        LDA   L6TAB+1,X   Get second byte of segment data.
        LSRA              From this byte only the
        LSRA              upper nibble is relevant. lower
        LSRA              nibble is lost as upper nibble
        LSRA              is shifted down.
        STA   W1          Save nibble. to be combined with first
        INC   W2          nibble of next digit.
        LDX   W2          Address of next digit in Q.
        LDX   Q,X         Get HEX data.
        AND   # $0F       only lower nibble is relevant.
        LSLA              ×2(two bytes per digit in table).
        TAX
        LDA   L6TAB,X     Get first byte from segment table.
        LSLA              From this byte only the
        LSLA              lower nibble is relevant. upper
        LSLA              nibble is lost as lower nibble
        LSLA              is shifted up.
        ADD   W1          Combine nibble with last nibble
        STA   LDAT        of previous digit and send byte to LCD.
        INC   LADD        Next LCD data latch.
        LDA   L6TAB+1,X   Get second byte from segment table.
        STA   LDAT        and send it to LCD.
        INC   LADD        Next latch(three per loop).
        INC   W2          Next digit(two per loop).
        LDA   LADD
        CMP   #12         Finished?
        BNE   L60P        If not. do next two digits.
        RTS
```

2. 使用 ICM7231 LCD 驱动器的 LCD 显示程序

```
PORTD   EQU   $0001       PORT B DATA
```

```
PORTC     EQU     $0002          PORT C DATA
          ORG     $0050
Q         RMB     8              DISPLAY REGISTER
          ORG     $0100
*
*显示Q的内容
DISP      LDA     PORTD          CLEAR
          AND     #$F0           LS NIBBLE OF PORTD
          STA     PORTD          IE DIGIT ADDRESS=0
          LDX     #8
AGAIN     LDA     Q-1,X
          STA     PORTC
          BCLR    3,PORTD        LATCH
          BSET    3,PORTD        DIGIT
          DECX
          BEQ     OUT            DONE ?
          INC     PORTD          NO,GOTO NEXT DIGIT
          BRA     AGAIN
OUT       CLR     PORTC
          RTS
```

22.3 四后板信号 LCD 显示

如前所述，MC68HC05L6 可直接驱动 4 个后板输出和 24 个前屏输出 LCD 驱动信号，可驱动 12 个 7 段码 LCD 数字。其软件类似于 3 个后板信号 LCD 显示情况，但更为简单。当使用没有 LCD 驱动能力的单片机时，MC145000/MC145001 等可用来进行 LCD 显示驱动。MC145000 可直接驱动 6 个数字（12 个前屏信号），并且可以加一个或多个 18 引脚的 MC145001 进行扩展。MC145001 可提供 11 个前屏信号。图 22-4 是采用 MC145000 显示 6 个数字的电路图（程序见后面）。其软件与使用 MC144115 的情况很相似，这是因为两种芯片都具有相同的移位寄存器/锁存器结构，尽管实际的输出信号不同。

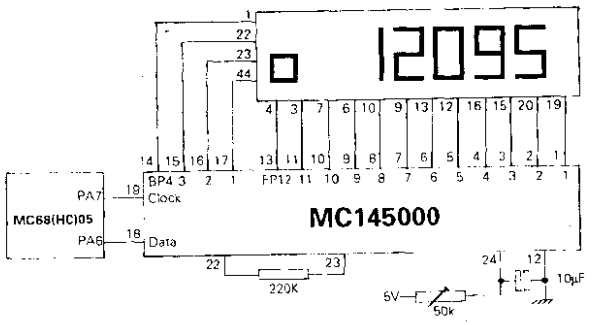


图 22-4 采用 MC145000 的 LCD 显示

MC145000 与 MC144115 的区别在于 MC145000 没有片选允许输入端。它可以与其他外

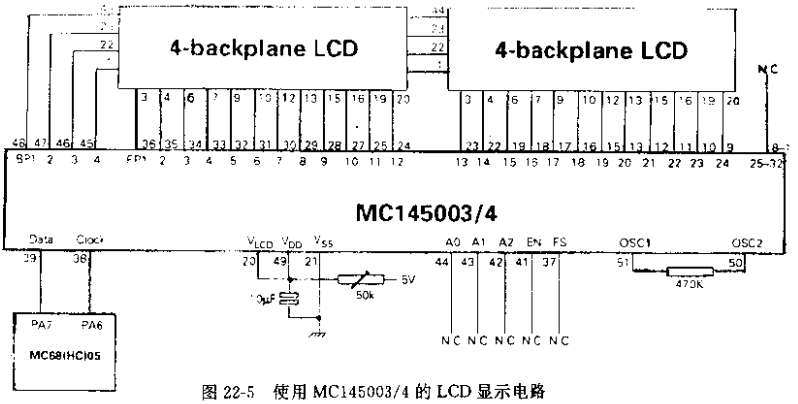


图 22-5　使用 MC145003/4 的 LCD 显示电路

图器件公用数据线，但必须有专用的时钟，这样单片机才能正确地向其提供数据。

MC145003/4 可提供 32 个前屏输出，可驱动 128 段 LCD。MC145003 和 MC145004 完全相同。只是串行传输协议不同。MC145004 为 IIC(I^2C)接口，具有通常的应答过程，MC145003 与其相同，但没有应答。接收到 128 位数据后，自动将数据锁存。如果需要在其他时间锁存数据，要使用使能(EN)引脚。

在 V_{DD} 和 V_{led} 连接在一起的应用场合，LCD 的对比度由调整 V_{DD} 进行调整。如果输入数据来自具有较高电源电压的芯片，输入引脚的高电平可能高于 MC145003/4 的电源。对于时钟和数据引脚这是允许的。但对于使能(EN)引脚，建议不要这样。因此，如果 MC145003/4 的 V_{DD} 与提供数据的芯片电源不同，建议不要使用使能引脚。可将 EN 接高电平或浮空。

图 22-5 的 LCD 显示电路不使用 EN 引脚。软件每次执行时发送 128 位，故自动进行锁存。该电路驱动两个 6 位 LCD 显示器，每个显示器有 12 个前屏信号。

1. 使用 MC145000 的 LCD 显示程序

```
PORTA   EQU   $00      PORT A DATA
BAUD    EQU   #0D      SCI BAUD RATE REGISTER
SCR1    EQU   $0E      SCI CONTROL REG. No. 1
SCR2    EQU   $0F      SCI CONTROL REG. No. 2
SCSR    EQU   $10      SCI STATUS REG.
SDAT    EQU   $11      SCI DATA REG.
        ORG   $0050
R       RMB   6        WORKING NUMBER
W2      RMB   1        POSITION OF LSB
W3      RMB   1
W4      RMB   1
W5      RBM   1
W6      RMB   1        POSITION OF MSB
*
* 显示程序第一部分，用七段码代替 BCD 码
        ORG   $1000
```

```
DISP        LDA     # $ 05
            STA     W4
            LDX     W2          LSB
D3          LDA     0,X
            STX     W5
            TAX
            LDA     STABL,X     FIND 7 SEGMENT CODE
            LDX     W4
            STA     R,X         PUT IN DISPLAY TABLE
            DEC     W4
            LDX     W5
            DECX
            CPX     W6          FINISHED ?
            BNE     D3
*
* 显示程序第二部分。发送显示驱动器所需的 48 位数据。
* 为便于比较,给出两种程序,一个使用 A 口线,另一个使用 SCI
OUTT        LDX     #5          SEND DISPLAY TABLE TO 144115/145000
DISCHR      LDA     R,X
DISPLY      STX     W3          SAVE INDEX
            BCLR    6,PORTA     CLEAR DATA
            LDX     #8
DIS1        LSRA                SET UP
            BCC     DIS2        BIT OF
            BSET    6,PORTA     ACCUMULATOR
DIS2        BSET    7,PORTA     CLOCK
            BCLR    7,PORTA     IT
            BCLR    6,PORTA     CLEAR DATA
            DECX                COMPLETE ?
            BNE     DIS1        NO
            LDX     W3          RESTORE INDEX
            DECX
            BPL     DISCHR
*
* SCI  LCD 驱动器接口
            LDX     #5          INITIALISE X
MORE        LDA     R,X         FETCH DIGIT
            BRCLR   7,SCSR,*    WAIT UNTIL TDRE=1
            STA     SDAT        WRITE IT TO SCI TX REG.
            DECX                NEXT DIGIT
            BPL     MORE        DONE ?
            BRCLR   6,SCSR,*    WAIT UNTIL TC=1
            RTS
STABL       FCB     $ EB        0  SEGMENT
            FCB     $ 60        1
```

```
        FCB     $C7     2  CODES
        FCB     $E5     3
        FCB     $6C     4  FOR THE
        FCB     $AD     5
        FCB     $AF     6  MC145000/MC144115
        FCB     $E0     7
        FCB     $EF     8  LCD DRIVER
        FCB     $ED     9
```

2. 使用 MC145003/4 LCD 驱动器的程序

```
IICP    EQU     $02     PORTC
IICDD   EQU     $06     PORTCD
SCL     EQU     $06     IIC-Clock line
SDA     EQU     $07     IIC-data line
DIN     EQU     $40     INPUT DATA
DOUT    EQU     $C0     OUTPUT DATA
        ORG     $0050
W1      RMB     1
DPNT    RMB     1       IIC WRITE POINTER
ADDR    RMB     1       IIC ADDRESS
IIC     RMB     16      IIC BUFFER(128 BITS)
        ORG     $0B00
START   LDA     #$7E
        STA     ADDR
        LDA     #17
        STA     W1
        LDX     #ADDR
        STX     DPNT
SEND2   BCLR    SDA,IICP    START CONDITION
        BCLR    SCL,IICP    DATA GOES LOW WHILE CLOCK HIGH
SLOOP   LDX     DPNT        DATA BUFFER POINTER
        LDA     0,X         GET A BYTE
        LDX     #8          8 BITS TO SHIFT
SLOP    ROLA
        BCC     DZERO       BIT=0 ?
        BSET    SDA,IICP    NO,BIT=1
DZERO   BSET    SCL,IICP    CLOCK HIGH
        BCLR    SCL,IICP    CLOCK LOW
        BCLR    SDA,IICP    DATA LOW
        DECX
        BNE     SLOP
*       LDA     #DIN        DATA LINE AN INPUT
*       STA     IICDD
*       BSET    SCL,IICP    CLOCK
*       BCLR    SCL,IICP    ACKNOWLEDGE BIT
*       LDA     #DOUT
```

```
*           STA     IICDD       BACK TO AN OUTPUT
            INC     DPNT        NEXT BYTE
            DEC     W1
            BNE     SLOOP       LAST BYTE ?
I2CEND      BSET    SCL,IICP    STOP CONDITION
            BSET    SDA,IICP    DATA GOES HIGH WHILE CLOCK HIGH
            RTS
```

22.4 8/16 后板信号 LCD 显示

LCD 点阵显示常采用 8 或 16 后板信号。对于需要显示大量象素(点阵)的情况,引脚数和对比度决定后板信号数。通常采用 8/16 后板输出,这样既可以有较好的对比度,又便于使用通常的 8×5 点阵显示格式。MC68HC05L7/L9 可用于直接驱动 8/16 后板 LCD 显示格式。MC68HC05L7 有 16 个后板信号输出和 60 个前屏输出,可驱动 960 个象素或 24 个 8×5 点阵的数字。MC68HC05L9 只有 40 个前屏输出(16 个 8×5 点阵数字),但它可以与 MC68HC68L9 LCD 扩展器一起使用。每个 MC68HC68L9 可提供 55 个前屏输出,可以增加 3 个 MC68HC68L9。L9 单片机和 3 个扩展器芯片共有 205 个前屏输出,可驱动 3280 个 LCD 象素,或 82 个 8×5 点阵的数字。

MC68HC05L10 可提供 32/41 后板输出,与 LCD 驱动器 MC141511(具有 128 个前屏输出)--起使用能驱动 5K～20K 个 LCD 象素(可以扩展 4 个 MC141511 芯片)。MC68HC05L11 可以驱动高达 40K 的 LCD 象素,适用于大屏幕和汉字 LCD 显示场合。

MC68HC05L7/L9 的显示 RAM 含有 8×5 点阵格式每一行的 5 位数据,因此,每个 LCD 数字需要 8 个单元,这样安排便于寻址。

22.5 LCD 驱动器芯片简介

表 22-1 列出了常用的 LCD 驱动器芯片的主要性能。

表 22-1 LCD 驱动器主要性能

器 件	后板	前屏	可驱动 7 段码数字	驱动方式	扩展	引脚数	其 他
MC14543	1	7	1	并行	并行	16	BCD
MC14544	1	7	1	并行	并行	18	
MC144115	1	16	2	3 线	是	24	
MC144117	2	16	4	3 线	无	24	
MC145000	4	12	6	2 线	MC145001	24	
MC145001	(4)	11	5.5	2 线	—	18	扩展器
MC145003	4	32	16	2 线/3 线	并行	52	2 或 3 线
MC145004	4	32	16	2 线	并行	52	IIC
MC145453	1	33	4	2 线	并行	40	也有 44 脚
ICM7231	3	24	8	并行	无	40	BCD

23. 采用 MC68HC805B6 的 MCM2814 编程器

MCM2814 EEPROM 广泛应用在各种设备中，用来存储半永久性或用户定义的数据。最常用的场合是 TV。在 TV 中，用 EEPROM 非挥发性存储器(NVM)存储频道号、每个节目号的频率或其他信息。在批量生产情况，需要将初始数据写入 EEPROM 中。图 23-1 是 EEPROM MCM2814 编程器电路，它可以以并行方式将 8 个 MCM2814 编程，并分别进行校验。该编程器由 MC68HC805B6 单片机控制。

MC68HC805B6 具有 256 字节 EEPROM，与 MCM2814 相同，故特别适用这一场合。MCM2814 的数据可以存入并保持在 HC805B6 中。MCM2814 和 HC895B6 的 256 字节 EEPROM 都有一个字节专用作数据保护，故实际使用 255 字节。

23.1 操作原理

编程器应便于使用，控制尽量简单。当按下 S1 按钮时，EEPROM 上电，编程器开始操作。释放该开关时，EEPROM 断电，编程器其他部分仍与电源相连。其操作有 3 种。

1. 数据装入编程器。按下并保持 S3 按钮选择这一过程，按下并保持 S1 开关执行这一过程。在插槽＃0 EEPROM 中的内容将装入编程器，需 12 秒钟时间。MC68HC805B6 将该数据保持在其 EEPROM 中。装入期间插槽＃0 的 LED 都亮，其他关闭。一旦写入 HC805B6，其内容就与外部＃0 EEPROM 进行校验，若绿 LED 亮表明成功，红 LED 亮表明校验失败。这期间＃1～7 插槽应该空。按下 S3 时，S2 的位置没有影响。由于 S1 为外部 EEPROM 提供电源，在该过程结束之前必须一直保持 S1 连接。若校验结束之前释放 S1，则不能获得可靠的结果。

2. 编程过程。S2 置于“P/V”(编程并校验)选中这一过程。NVM MCM2814 应置于插槽中，按下 S1 并保持，直到编程和校验结束(4 秒)。编程期间，红和绿 LED 都亮。

这一过程将 NVM 的写保护字节清零，用存储在 HC805B6 中的数据以并行方式对 NVM 编程。然后分别地校验 EEPROM，校验结果显示在 LED 上(绿 LED 亮表示通过，红 LED 亮表示有错)。

3. 只进行校验。S2 置于“V”选中这一操作。该步骤与编程类似，S1 应一直保持到校验结束(最多 1 秒)。

23.2 电路

MCM2814 NVM 用其 IIC 片选引脚(引脚 1 和 2)进行连接。这两引脚设置 IIC 地址的两个最低位。位 0 决定器件是接收还是发送。软件只使用地址 1010000X，因此，只有这两个片选引脚都为低时，NVM 才被寻址。用公共软件可以由 I/O 线分别对每个 NVM 进行寻址。这里使用 6 条 I/O 线，即 PB0～PB5。

当 S1 闭合时，NVM 才加电，这时可以插拔芯片。当 S1 按下时，5V 电源也加到与 BC107NPN 晶体管相连的 RC 电路。在 10μF 电容充电时，为晶体管提供基极电流。在这一瞬态过程中，晶体管 BC107 导通，HC805B6 复位。电容充电后，该晶体管截止，开始执行程序。MC34064 也连至 MCU 的复位引脚，以保证当电源低于 4.75V 时使单片机复位。在任何含有

EEPROM 的系统中，都应该采用这一方法，以保持操作可靠。

电源中串联 33Ω 电阻是为了防止器件插错起限流作用。若发生这种情况，BC327 PNP 晶体管开启，红 LED 亮。实际 NVM 电源引脚电压由 PD5 监视。单片机可以判别是否插反，若发生这种情况，软件可终止 NVM 的编程和校验。

A 口和 C 口分别用于 EEPROM 的检验结果，D 口的两个固定输入引脚 PD0 和 PD1 分别用来读开关 S3 和 S2 的位置。

23.3 软件

复位时，程序对 I/O 口、IIC 地址和用作状态标志的 RAM 单元(STAT)进行初始化，然后检查 D 口位 0 的状态。除非按下 S3，PD0 应为高。若未按下 S3，则根据与 S2 相连的 PD1 的状态进行编程与检验或只进行校验。

编程程序(PROG)使所有 LED 亮，允许所有的 NVM EEPROM，并向地址 $FF 写入 $00，将写保护字节清零。从编程器 HC805B6 读一个字节并写入 NVM，这一过程循环执行 255 次。只传输 255 字节，MCM2814 中的最后一个字节保留用作写保护状态。

校验程序(VERF)关闭所有 LED，并将 B6 中的数据与每个 EEPROM 中的数据进行比较。RAM 单元 COUNT 含有正在检验的插槽个数，校验结束时设置适当的 LED 亮。

NVM 以 2×4 矩阵形式排列，使用两个 IIC 片选。所用的 IIC 地址总是 $A1(写入为 $A0)，这样，只有两个片选都为低时，MCM2814 才响应。在编程期间，所有线均为低；但在校验期间，每行和每列只有一条线为低，单独选择每个芯片。若选择 LOAD 程序(写入 HC805B6)，则将 MCM2814 的内容传输到 MC805B6 的 EEPROM 中，含有程序的 EEPROM 应插在 #0。

调试期间使用三个程序：WRT、RDALL 和 WRTAL。HC805B6 不使用这些程序，但在开发系统上运行时可以使用。READ 子程序用 ADDR 的 IIC 地址和 SUBARD 的子地址读一个字节。SEND 子程序在 IIC 地址 ADDR 向 IIC 总线写入数据。发送的字节数由存储在累加器中的值决定。在本例中，如果必须发子地址，则它应该位于变址寄存器指向的 RAM 单元，随后的单元含有数据。SEND 程序适于多数据 IIC 外围器件。

IIC 程序对数据寄存器使用简单的 BCLR 和 BSET 指令，产生有效的高电平。在 IIC 总线并不严格地要求这样，但在系统中没有其他主机的情况，这是可行的。

23.4 程序清单

```
PORTA     EQU     $00          PORT A ADDRESS
PORTB     EQU     $01          PORT B ADDRESS
PORTC     EQU     $02          PORT C ADDRESS
PORTD     EQU     $03          PORT D ADDRESS
PORTAD    EQU     $04          PORT DATA DIRECTION REG.
PORTBD    EQU     $05
PORTCD    EQU     $06
EECTL     EQU     $07          EEPROM CONTROL REGISTER
E2PR      EQU     $0100        EEPROM
BUFF      EQU     $0F00        DEBUG USE ONLY
*
```

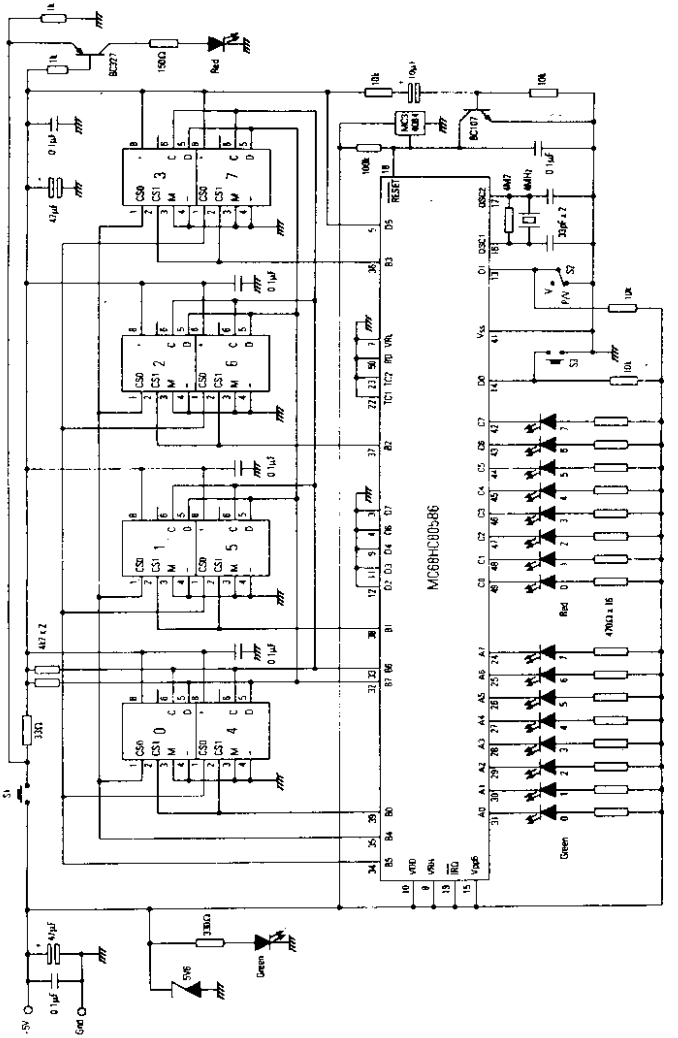


图 23-1 MCM2814 编程器电路

```
* * * *RAM 定义
         ORG     $0050
ADDR     RMB     1       IIC ADDRESS
DPNT     RMB     1       IIC DATA POINTER
SUBADR   RMB     1       IIC SUB-ADDRESS
IOBUF    RMB     2       IIC BUFFER
W1       RMB     1       W
W2       RMB     1       O
W3       RMB     1       R
W4       RMB     1       K
W5       RMB     1       I
W6       RMB     1       N
W7       RMB     1       G
COUNT    RMB     1       LOOP COUNTER
STAT     RMB     1       STATUS BYTE:-
*                        0:not used
*                        1:IIC R/W 1:READ,0:WRITE
*                        2:ACKNOWLEDGE OK
         RMB     154     not used
STACK    RMB     7       8 BYTES USED (0 INTERRUPTS
SP       RMB     1       AND 4 NESTED SUBROUTINES)
         ORG     $0800
*        page 2
*
* * * *主程序
START    LDA     #$C0
         STA     PORTB   CONTROL LINES LOW
         LDA     #$FF
         STA     PORTA   LEDS OFF
         STA     PORTC   LEDS OFF
         STA     PORTAD  ALL OUT-GREEN LEDS
         STA     PORTCD  ALL OUT-RED LEDS
         LDA     #$3F    6,7 IN-IIC BUS
         STA     PORTBD  0-5 OUT-NVM SELECTION
         LDX     #100    WAIT 230mS FOR MCM2814 SUPPLY
         JSR     PN      TO SETTLE
         BRSET   5,PORTD,CONT    NVMs POWERED UP ?
         BRA     *       NO,DO NOT PROCEED
CONT     LDA     #$A0    YES,INITIALISE IIC ADDRESS
         STA     ADDR
         CLR     EECTL   READ FROM EEPROM(B6)
         CLR     SUBADR
         CLR     STAT
         CLRX
         BRSET   0,PORTD,SKLD    LOAD B6 EEPROM?
         JMP     LOAD    YES
```

```
SKLD        BRSET     1,PORTD,VERF     NO,VERIFY ONLY?
*
* * * * *  B6  EEPROM    ($101-$1FF)->8 NVMs ($00-$FE)
PROG        CLR       PORTA            GREEN LEDS ON
            CLR       PORTC            RED LEDS ON
            LDA       #$C0
            STA       PORTB            ENABLE ALL NVMS
            LDA       #$FF
            STA       SUBADR           SUBADR=$FF
            JSR       READ             CLEAR POR WRITE PROTECT
            CLRA                       DATA=$ $00
            BSR       WR1              CLEAR WRITE PROTECT BYTE
            CLR       SUBADR
            CLRX
OLOOP       LDA       E2PR+1,X         GET BYTE AND
            BSR       WR1              SEND IT TO NVM
            CPX       #255             DONE?
            BNE       OLOOP
            BRA       VERF             YES,VERIFY
WR1         STA       IOBUF
            LDA       #2               NO,BYTES TO SEND(INC. SUB-ADDRESS)
            LDX       #SUBADR
            JSR       WRITE            IIC WRITE
            INC       SUBADR           NEXT LOCATION
            LDX       SUBADR
            RTS
*                     page 3
*
* * * * *    verify.
*           B6 EEPROM  ($0101-$01FF)v. NVMs ($00-$FE).
VERF        LDA       #$FF
            STA       PORTC            ALL LEDS OFF
            STA       PORTA
            CLR       COUNT            START AT SOCKET #0
VLP         LDX       COUNT
            LDA       TAB1,X           SELECT NVM
            STA       PORTB            ACC. TO COUNT
            BSR       VRF1
            BCC       P0
F0          BRSET     2,STAT,FINP      IF NO ACK. THEN NO LED
            LDX       COUNT
            BNE       F1
            BCLR      0,PORTC
F1          CPX       #1
            BNE       F2
            BCLR      1,PORTC
```

```
F2          CPX     #2
            BNE     F3
            BCLR    2,PORTC
F3          CPX     #3
            BNE     F4
            BCLR    3,PORTC
F4          CPX     $4
            BNE     F5
            BCLR    4,PORTC
F5          CPX     #5
            BNE     F6
            BCLR    5,PORTC
F6          CPX     #6
            BNE     F7
            BCLR    6,PORTC
F7          CPX     #7
            BNE     FINP
            BCLR    7,PORTC
FINP        BRA     OUT
*                   page  4
*
* * * * * Verify (continued).
*           B6 EPROM ($0101— $01FF) v. NVM ($.00— $FE).
P0          LDX     COUNT
            BNE     P1
            BCLR    0,PORTA
P1          CPX     #1
            BNE     P2
            BCLR    1,PORTA
P2          CPXP    #2
            BNE     P3
            BCLR    2,PORTA
P3          CPX     #3
            BNE     P4
            BCLR    3,PORTA
P4          CPX     #4
            BNE     P5
            BCLR    4,PORTA
P5          CPX     #5
            BNE     P6
            BCLR    5,PORTA
P6          CPX     #6
            BNE     P7
            BCLR    6,PORTA
P7          CPX     #7
            BNE     OUT
```

```
        BCLR    7,PORTA
OUT     INC     COUNT           NEXT SOCKET
        LDA     COUNT
        CMP     #7
        BLS     VLP             FINISHED?
        CLR     PORTB           YES,ALL CONTROL LINE LOW
        BRA     *               AND WAIT HERE
VRF1    CLR     SUBADR
        CLRX
VLOOP   JSR     READ            GET A BYTE FROM NVM
        BRSET   2,STAT,FAIL     ACKNOWLEDGE OK ?
        LDA     IOBUF           YES,CHECK DATA
        LDX     SUBADR
        CMP     E2PR+1,X        COMPARE WITH B6 EEPROM BYTE
        BNE     FAIL            SAME ?
        INC     SUBADR          YES,CONTINUE
        LDX     SUBADR
        CPX     #255            FINISHED (255 BYTES) ?
        BNE     VLOOP
        CLC                     PASS,EXIT WITH C CLEAR
        RTS
FAIL    SEC                     FAIL,EXIT WITH C SET
        RTS
TAB1    FCB     $EE,$ED,$EB,$E7,$DE,$DD,$DB,$D7
*               page 5
*
* * * * *  NVM (#0,$00-$FE)→B6 EEPROM ($101-$1FF).
LOAD    LDA     #$EE            SELECT POSITION 0
        STA     PORTB
        LDA     #$FE            POSITION 0 LEDS ON.
        STA     PORTA
        STA     PORTC
ILOOP   JSR     READ            GET BYTE FROM NVM
        BRSET   2,STAT,SKIP     ACKNOWLEDGE OK ?
        LDA     IOBUF           YES,GET DATA
        BSET    1,EECTL         SET E1LAT
        BSET    2,EECTL         SET E1ERA
        BSR     DOIT            ERASE BYTE
        JSR     P10             WAIT 9.2 ms
        BSET    1,EECTL         SET E1LAT TO WRITE BYTE
        BSR     DOIT
        INC     SUBADR
        LDX     SUBADR
        CPX     #255
        BNE     ILOOP
SKIP    JMP     VERF
```

```
DOIT    LDX     SUBADR          GET ADDRESS
        STA     E2PR+1,X        LATCH DATA
        BSET    0,EECTL         START PROGRAMMING
        JSR     P20             WAIT 18.4ms
        CLR     EECTL           STOP
        RTS
*               page 6
*
***** Debug routines for use with EVM,not used
*       in 805B6
***** RAM ($F00-$FFE)→B6 EEPROM ($101-$1FF).
WRT     CLR     SUBADR
        CLRX
LOOP2   LDA     BUFF,X
        BSET    1,EECTL         SET E1LAT
        BSET    2,EECTL         SET E1ERA
        BSR     DOIT            ERASE BYTE
        JSR     P20
        BSET    1,EECTL         SET E1LAT TO WRITE BYTE
        BSR     DOIT
        INC     SUBADR
        LDX     SUBADR
        CPX     #255
        BNE     LOOP2
        BRA     *
*
***** NVM ($00-$FF)→RAM($0F00-$0FFF).
RDALL   CLR     SUBADR
        LDA     #$EE            SELECT POSITION 0
        STA     PORTB
RLOOP   JSR     READ
        LDX     SUBADR
        LDA     IOBUF
        STA     BUFF,X
        INC     SUBADR
        BNE     RLOOP
        BRA     *
*
***** RAM ($0F00-$0FFF)→NVM ($00-$FF).
WRTAL   CLR     SUBADR
        LDA     #$EE            SELECT POSITION 0
        STA     PORTB
        CLRX
WLOOP   LDA     BUFF,X
        JSR     WR1
        BNE     WLOOP
```

```
                BRA     *
*               page 7
*
***** IIC routines.
IICP    EQU     $01             PORTB
IICDD   EQU     $05             DDRB
SCL     EQU     $06             IIC—clock line
SDA     EQU     $07             IIC—data line
DIN     EQU     $7F             INPUT DATA
DOUT    EQU     $FF             OUTPUT DATA
OPEN    EQU     $3F             TRI-STATE BOTH
SEND    BCLR    1,STAT          WRITE IIC DATA
        STA     W1              SAVE No.    BYTES TO SEND
        BRA     IICBUS
WRITE   BCLR    1,STAT          WRITE TO NVM
        STA     W1              SAVE No.    BYTES TO SEND
        BSR     IICBUS
        JSR     P10             9.2ms NVM WRITE TIME
READ    BSET    1,STAT          READ IIC DATA
IICBUS  BCLR    SCL,IICP        CLOCK LOW
        BSET    SDA,IICP        DATA HIGH
        BSET    SCL,IICP        CLOCK HIGH
        LDA     #DOUT           DRIVE
        STA     IICDD           BOTH
        BCLR    0,ADDR          RW=0 ALWAYS WRITE (SUB-ADDRESS)
        LDA     ADDR            SEND CHIP ADDRESS
IICD3   BCLR    SDA,IICP        START CONDITION
        BCLR    SCL,IICP        DATA GOES LOW WHILE CLOCK HIGH
        STX     DPNT
        BSR     SHIFT           CHIP ADDRESS OUT
        BRCLR   1,STAT,WRBUS    READ OR WRITE ?
*               page 8
*
***** Send sub-address and read data from bus.
RDBUS1  LDA     SUBADR
        BSR     SHIFT           WRITE SUB-ADDRESS
RDBUSQ  BSET    0,ADDR          SET BIT 0 FOR READ
        LDA     ADDR            CHIP ADDRESS
        BSET    SDA,IICP        BOTH HIGH,NO
        BSET    SCL,IICP        STOP CONDITION
        BCLR    SDA,IICP        START CONDITION
        BCLR    SCL,IICP
        BSR     SHIFT           RE-SEND CHIP ADDRESS
RDBUS3  LDA     #DIN            DATA IN FROM BUS
        STA     IICDD
        BSR     RDB             READ 8 BITS
```

```
*               BSR     RACKF           DUMMY ACKNOWLEDGE
*               LDA     IOBUF           MOVE
*               STA     IOBUF+1         UP
*               BSR     RDB             AND READ 8 MORE
                BSR     RACK
                BRA     IICEND
RDB             LDX     #8
RDBUS2          BSET    SCL,IICP        CLOCK HIGH
                BRSET   SDA,IICP,*+3    DATA LINE (RESULT IN CARRY)
                BCLR    SCL,IICP        CLOCK LOW
                ROL     IOBUF
                DECX
                BNE     RDBUS2
                RTS
RACK            BSET    SDA,IICP        LAST BYTE READ;SDA HIGH
RA2             LDA     #DOUT           SDA OUT
                STA     IICDD           DUMMY ACKNOWLEDGE
                BSET    SCL,IICP        CLOCK
                BCLR    SCL,IICP
                BCLR    SDA,IICP
                LDA     #DIN            SDA IN
                STA     IICDD
                RTS
RACKF           BCLR    SDA,IICP        ACKN. WITH FOLLOWING BYTE
                BRA     RA2
*               page    9
*
* * * *         Send sub-address and write data onto bus.
WRBUS           LDX     DPNT            DATA BUFFER POINTER
                LDA     0,X             DATA
                BSR     SHIFT
                INC     DPNT
                DEC     W1              No. BYTES
                BNE     WRBUS
IICEND          BSET    SCL,IICP        STOP CONDITION
                BSET    SDA,IICP        DATA GOES HIGH WHILE CLOCK HIGH
                LDA     #OPEN           TRI-STATE
                STA     IICDD
                RTS
*
* * * * *       Shift out 8 bits and check acknowledge bit.
SHIFT           LDX     #8
SHIFT1          ROLA
                BCC     SHIFT2          SHIFT OUT 8 BITS 0?
                BSET    SDA,IICP        NO,DATA=1
                BRA     SHIFT3
```

```
SHIFT2    BCLR    SDA,IICP          DATA=0
          BRA     SHIFT3            DELAY
SHIFT3    BSET    SCL,IICP          CLOCK HIGH
          BCLR    SCL,IICP          CLOCK LOW
          BCLR    SDA,IICP          DATA LOW
          DECX
          BNE     SHIFT1
WACK      LDA     #DIN              WRITE ACKNOWLEDGE
          STA     IICDD
          BCLR    2,STAT            CLEAR FLAG
          BSET    SCL,IICP          CLOCK
          BRCLR   SDA,IICP,ACOK     ACKNOWLEDGE OK ?
          BSET    2,STAT            NO,SET FLAG
ACOK      BCLR    SCL,IICP
          BCLR    SDA,IICP
          LDA     #DOUT
          STA     IICDD
          RTS
*                 page  10
*
* * * * *   Delay (W2×2.3ms with a 2MHz bus).
P10       LDX     #4                9.2ms
          STX     W2
          BRA     TPAU
P20       LDX     #8                18.4ms
PN        STX     W2
TPAU      CLRX
DLOOP     BSR     DALLY             12(周期数)
          DECX                      3  15
          BNE     DLOOP             3  18  18×256/2=2304μs
          DEC     W2
          BNE     TPAU
DALLY     RTS
*
* * * * *   B6 reset and interrupt vectors.
          ORG     $1FF2
          FDB     START             SCI
          FDB     START             TIMER(OVER)
          FDB     START             TIMER(OUT CMP)
          FDB     START             TIMER(IN CAP)
          FDB     START             IRQ
          FDB     START             SWI
          FDB     START             RESET
          END
```

24. 由 M68HC11 单片机控制步进电机

步进电机是将电信号脉冲转化为角位移的设备。与只能连续旋转的普通电机相比，其优点是：①能快速起停、间歇运动和小角度运动；②转角和转速只与驱动脉冲频率有关；③转角和转速不受工作条件和环境（如电压波动、负载变化以及温度、气压变化、振动等）的影响；④步距误差不会长期积累。

由 M68HC11 控制的步进电机原理图如图 24-1 所示。此步进电机的转子是按步转动的。它们通过按一定顺序激发的四个线圈来完成预定的动作。转子每一步转过标准的角度（为 3.75°或 7.5°）。步进电机的四个线圈由四只晶体管 Q1～Q4 驱动，晶体管的基极直接由 M68HC11 的输出引脚 PA3～PA6 驱动。

步进电机的转动是由电脉冲按一定的顺序加到电机线圈上来实现的。线圈通电顺序与步进电机转角和转动方向的关系如表 24-1 和表 24-2 所示。表中从表顶向下的顺序为顺时针；从表底向上的顺序为逆时针。表 24-1 为每步转 3.75°。它是在半步旋转每步转 7.5°的基础上增加一个中间步来实现的。

表 24-1 半步转动顺序（顺时针方向转动）

	Q4	Q3	Q2	Q1
1	0	1	0	1
2	0	0	0	1
3	1	0	0	1
4	1	0	0	0
5	1	0	1	0
6	0	0	1	0
7	0	1	1	0
8	0	1	0	0
1	0	1	0	1

表 24-2 全步转动顺序（顺时针方向）

	Q4	Q3	Q2	Q1
1	0	1	0	1
2	1	0	0	1
3	1	0	1	0
4	0	1	1	0
1	0	1	0	1

24.1 控制原理

步进电机的转角与时间的关系为：

$$\theta(t)=A\cdot\sin(\omega t)$$

步进电机的运动可简化为用几个字节来表示。例如，运动可归纳为以下类型：

(1)按规定的速度顺时针旋转一定的步数。

(2)按规定的速度逆时针旋转一定的步数。

(3)在当前位置上停一定的时间。

所对应的操作码和操作数可以为：

01——顺时针运动。下一个字节是步数，再下一个字节是速度。

02——逆时针运动。下一个字节是步数，再下一个字节是速度。

03——保持在当前位置。下一个字节停留时间。

04——暂停。

下面的数据包含的意义为：

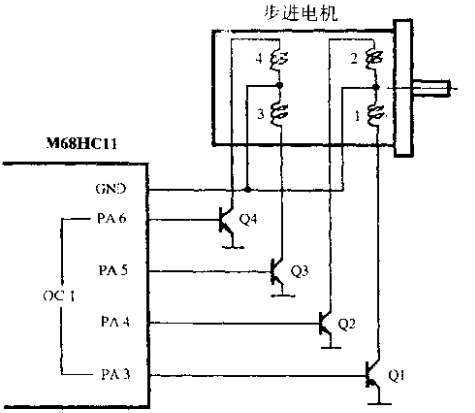


图 24-1　M68HC11 步进电机控制原理图

01
05　}用 $0A 表示的速度，顺时针转动 $5 个脉冲。
0A

03
2B　}停在当前位置，停留时间为 $2B。

02
12　}逆时针转动 $12 个脉冲，速度为 $1C。
1C

04—暂停。

24.2　步进电机的可编程定时器控制

步进电机的运动控制采用 M68HC11 中的定时器系统完成。首先要确定程序中的控制数据，根据这些数据控制电机的运动方式、运动方向和运动速度等。

1. 控制数据

控制数据分为 3 部分，分别存放在存储器中的 5 个字节中，如下所示：

0000	方　式
0001	方　向
0002	溢出次数(n)
0003	宽　度(W1)
0004	宽　度(W2)

(1)方式——1 个字节

方式字节表示步进电机旋转还是在一个特定的位置上停一定时间。“1”：旋转；“0”：不旋转。

(2)方向——2 个字节

方向字节决定电机顺时针旋转还是逆时针旋转。$00：顺时针旋转；$FF：逆时针旋转。

(3)时间间隔——3 个字节

时间间隔用来确定步进电机每两步间的时间间隔。一共 3 个字节，用 E 时钟周期数来表示。因为输出比较寄存器 TOC 只有两个字节(16 位)，需要使用溢出计数器去处理多于 32ms

的步间隔时间。3 个字节中，第一个字节为 n，表示溢出中断次数。后两个字节 W1、W2 表示宽度(剩下的 E 周期数)。两步间隔时间为：

$$步间时间=(n*\$FFFF+\$W1W2)*E(周期)$$

2. 数据调整

在运行时，需要在存储器中建立一个 OC1 中断寄存器和一个计数字节，以处理下一步的有关数据。每一步数据输出前，需完成如下工作：

(1)将宽度 W1W2 值(时间间隔的后两个字节)加到输出比较寄存器 1(TOC1)当前值上。

(2)定时器溢出计数器置为 0。

(3)调整溢出时间计数器内容。

因为宽度 W1W2 加 TOC1 的当前值后，溢出中断信号有可能出现在宽度 W1W2 之内(情况 1)；也可能出现在 W1W2 之后(情况 2)。而控制步距的 OC1 中断每次到来时，使计数器溢出计数字节与 n 比较。如果等于 n，则下一步开始。在情况 1，计数器溢出计数字节将提前增加。而情况 2 是正常的。正确的做法是，情况 1 应将计数字节减 1。这可通过宽度 W1W2 加到 TOC1 上时，检测进位位来判断是否减 1。

3. 输出顺序控制

为对步进电机进行输出顺序控制，可采用 M68HC11 特殊的定时器 OC1 输出控制功能，即用 OC1 同时控制 PA6～PA3 四个引脚。

OC1 控制 PA6～PA3 引脚利用 OC1 屏蔽寄存器 OC1M 和 OC1 数据寄存器 OC1D。这两个寄存器的最高 5 位分别对应 A 口的一个引脚。应用时，将 OC1M 中的相应位置 1，即可使 A 口对应的引脚受 OC1 控制。当 OC1D 中的某位置 1，则比较成功，对应的引脚将成为高电平；而 OC1D 为 0 的位对应的引脚则无反应。OC1M 和 OC1D 如下：

OC1M	7	6	5	4	3	2	1	0
$100C	OC1M7	OC1M6	OC1M5	OC1M4	OC1M3	0	0	0
	0	0	0	0	0	0	0	0
OC1D	7	6	5	4	3	2	1	0
$100D	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3	0	0	0
	0	0	0	0	0	0	0	0
	PA7/PA1	PA6/OC2	PA5/OC3	PA4/OC4	PA3/OC5	PA2/IC1	PA1/IC2	PA0/IC3

这种控制方式可同时给四个引脚提供信号。因此，OC1M 寄存器中对应引脚 PA6～PA3 的位(OC1M6～OC1M3)必须同时置 1。当 OC1 发生中断时，OC1D 寄存器中对应的位状态自动输出到 PA6～PA3。每次送出的引脚信号可用变址寄存器 Y 来选择。在每一个 OC1 中断后，Y 寄存器被更新，并且将下一个顺序值写入 OC1D 寄存器。同时将被更新的值存起来，以便在下一个循环中检索。

24.3 程序设计与程序清单

步进电机控制系统的程序流程图如图 24-2 所示。该程序采用 OC1 中断控制 A 口引脚。程序清单如下：

```
SEQ1    EQU    $28    0101
SEQ2    EQU    $48    1001   步进电机通电顺序
SEQ3    EQU    $50    1010
SEQ4    EQU    $30    0110
```

```
OC1MASK   EQU   $78          设置 OC1M
MODE1     EQU   $01
CW        EQU   $00
CCW       EQU   $FF
REGBAS    EQU   $1000        控制寄存器基地址
OC1M      EQU   $0C
OC1D      EQU   $0D
TOC1      EQU   $16
TMSK1     EQU   $22
TFLG1     EQU   $23
TMSK2     EQU   $24
TFLG2     EQU   $25
OC1JMP    EQU   $DF          OC1 中断向量
TOFJMP    EQU   $D0          TOF 中断向量
          ORG   $0000
*变量和常数
MODE      RMB   1            方式
DIRECT    RMB   1            方向
TOFVAL    RMB   1            步间溢出
WIDTH     RMB   2            宽度
TOFCNT    RMB   1            当前 TOF 计数值
YSAVE     RBM   2            当前 Y 变址寄存器值
SEQSTRT   RMB   4            步进顺序
          ORG   $C000
*初始化
          LDAA  #MODE1       存初始方式
          STAA  MODE
          LDAA  #CW          存方向
          STAA  DIRECT
          LDAA  #$00         每步缺省溢出数
          STAA  TOFVAL
          LDD   #$6000       步间缺省附加时间(宽度)
          STD   WIDTH
          LDX   #REGBAS
          LDD   #SEQSTRT
          STD   YSAVE
          LDAA  #$00
          STAA  TOFCNT       当前定时器溢出数
          LDAA  #SEQ1        存步进电机顺序值
          STAA  SEQSTRT
          LDAA  #SEQ2
          STAA  SEQSTRT+1
          LDAA  #SEQ3
          STAA  SEQSTRT+2
          LDAA  #SEQ4
          STAA  SEQSTRT+3
```

```
        LDAA    #$7E            跳转到 OC1 和 TOF 中断服务程序
        STAA    OC1JMP
        STAA    TOFJMP
        LDD     #OC1INTR
        STD     OC1JMP+1
        LDD     #TOFINTR
        STD     TOFJMP+1

        LDAA    #OC1MASK
        STAA    OC1M,X
        LDAA    #SEQ1           顺序 1 开始
        STAA    OC1D,X
        LDAA    #$80            允许 OC1 和 TOF 标志清零
        STAA    TFLG1,X
        STAA    TMSK1,X
        STAA    TFLG2,X
        STAA    TMSK2,X
        CLI                     清 CCR 寄存器中中断屏蔽位
*主程序
        BRA     *
*OC1 中断程序
OC1INTR LDAA    TOFCNT
        CMPA    TOFVAL          溢出(TOF)是否足够?
        BGE     STEPMOT         若是,电机步进
        LDAA    #$80            返回前清 OC1 标志
        STAA    TFLG1,X
        RTI
STEPMOT LDY     YSAVE
        CLR     TOFCNT          清 TOF 值
        LDAA    DIRECT          判断方向
        BNE     BACKWRD
FORWARD INY                     如果向前,IY 增加 1
        CPY     #SEWSTRT+4      检查是否太快
        BNE     NEXTSEQ
        LDY     #SEQSTRT        如果太快,调整
        BRA     NEXTSEQ
BACKWRD DEY                     如果是向后,IY 减 1
        CPY     #SEQSTRT-1      检查是否太快
        BNE     NEXTSEQ
        LDY     #SEQSTRT+3      若太快,调整
NEXTSEQ STY     YSAVE           存新 IY 值
        LDAA    0,Y             得到下一个顺序值
        LDX     #REGBAS
        STAA    OC1D,X          存下一个顺序值
        LDAA    #$80
        STAA    TFLG1,X         清 OC1 标志
```

```
                LDD     TOC1,X          读当前 TOC1 计数
                ADDD    WIDTH           当前计数加宽度
                STD     TOC1,X          更新值存入 TOC1
                BCC     RETURN          如果在宽度内出现 TOF,则调整
                DEC     TOFCNT
RETURN          RTI
*定时器溢出程序
TOFINTR     INC     TOFCNT          TOF 计数器加 1
            LDX     #REGBAS
            LDAA    #$80            清 TOF 标志
            STAA    TFLG2,X
            RTI
```

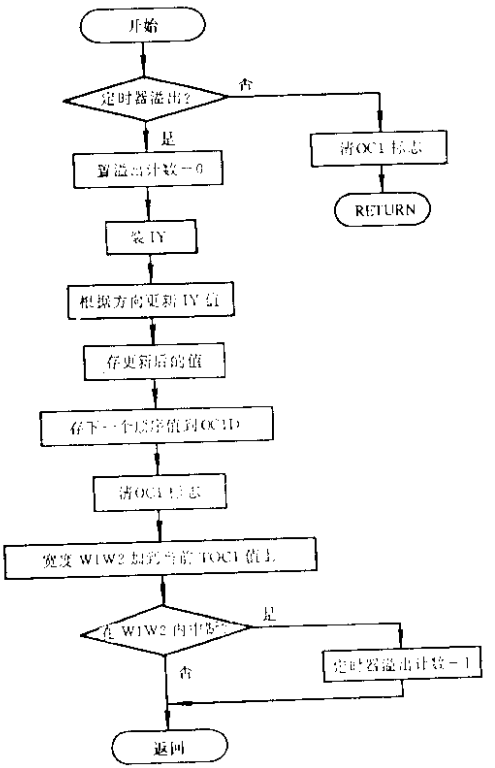


图 24-2　步进电机控制程序流程图

25. M68HC11 全应答方式并行 I/O 通讯

25.1 并行传输规程

两个 M68HC11 单片机使用并行口传输数据时通常要用并行 C 口和两个选通引脚(STRA 和 STRB)。一个单片机的 STRA 选通引脚与另一个单片机的 STRB 引脚相连。选通引脚上的信号设置控制寄存器 PIOC 的选通 A 状态标志位 STAF(位 7)。并行传输方式 MCU 的连接以及 MCU 共同遵守的协议如图 25-1 所示。操作过程如下:

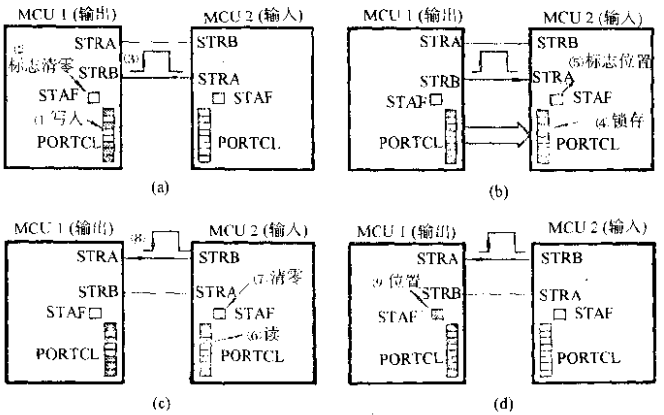


图 25-1 M68HC11 并行通讯的连接和通讯规程

1. MCU1 输出数据

如图 25-1(a)所示,开始时,并行 I/O 口控制寄存器 PIOC 的选通 A 标志位 STAF 为 1。STAF 不随选通或应答方式变化,而是随 STRA 引脚的有效沿变化。在全输入应答方式,读 PORTCL 寄存器使 STAF 为零;在全输出应答方式,写入 PORTCL 使 STAF 为零。图中操作步骤意义如下:①数据写入 MCU1 的 C 口锁存寄存器 PORTCL 中。②MCU1 为全输出应答方式,写入 PORTCL 自动将 STAF 清零。③写入 PORTCL 后,STRB 引脚自动发出脉冲(认为 STRB 为高有效,STRA 为上升沿有效),表明外部系统可以使用 C 口数据。

2. MCU2 接收数据

如图 25-1(b)所示。MCU2 工作在全输入应答方式,STRA 的有效沿将数据锁存到 PORTCL 寄存器,并将 STAF 置位。④MCU2 的 STRA 上升沿使 C 口输入数据锁存在 PORTCL。⑤STRA 的上升沿同时使 MCU2 的 STAF 为 1。这表明可以从 PORTCL 读出数据。

3. MCU2 准备接收新数据

如图 25-1(c)所示。工作在全输入应答方式的 MCU2 从 PORTCL 读数据,将 STAF 清零,

并产生 STRB 且保持两个 E 周期。⑥从 PORTCL 寄存器读数据。⑦从 PORTCL 读出数据时，STAF 自动清零。⑧同时 MCU2 自动产生 STRB，表明已准备就绪，可以接收新数据。

4. MCU1 准备发送新数据

如图 25-1(d)所示。MCU1 工作在全输出应答方式，其 STRA 的输入脉冲表示已经接收完 MCU1 C 口数据。⑨MCU1 的 STRA 引脚的输入上升沿将 STAF 置位，表明可以将新数据写入 PORTCL 中。之后重复上述步骤。

25.2 数据传输程序

1. 从键盘输入字符并输出至 C 口的程序

```
PIOC        EQU     $1002
PORTCL      EQU     $1005
DDRC        EQU     $1007
INCHAR      EQU     $E51A
            ORG     $C000
            LDAA    #$FF
            STAA    DDRC            数据方向寄存器设置为输出
*设置C口为全输出应答,脉冲选通,STRA为上升沿有效,选通为高有效($1F)
            LDY     #PIOC
            CLR     0,Y
            BSET    0,Y $1F
NEXTCHAR    JSR     INCHAR
            STAA    PORTCL          输出A值到口C
            BRCLR   0,Y $80 *       等待,直到STAF置位
            BRA     NEXTCHAR
```

2. 从 C 口取一个字符并送至终端的程序

```
PIOC        EQU     $1002
PORTCL      EQU     $1005
DDRC        EQU     $1007
OUTA        EQU     $E4D1
            ORG     $C000
            CLR     DDRC            设置数据方向寄存器为输入。
*设置C口为全输入应答,脉冲选通,STRA上升沿有效,选通高有效($17)
            LDY     #PIOC
            CLR     0,Y
            BSET    0,Y $17

GETCHAR     BRCLR   0,Y $80 *       等待,直到STAF为1
            LDAA    PORTCL          A得到字符,STAF清零
            JSR     OUTA            A中字符送到终端
            BRA     GETCHAR
```

26. 由单片机实现的免校准压力传感器系统

MPX2000 系列半导体压力传感器的输出电信号与所受的压力成正比。该传感器将单片硅膜和薄膜电阻网络集于一个芯片上，每个芯片的输出、零点偏移和温度补偿都经过激光修正，如图 26-1 所示。这里介绍利用 MC68HC05B6 HCMOS MCU 片内 A/D 转换器构成的精密而可靠的压力测量系统。

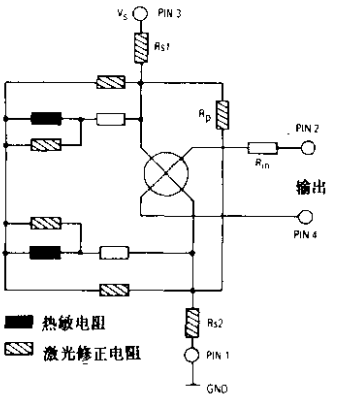


图 26-1 用 7 个激光修正电阻器和 2 个热敏电阻进行校准和温度补偿的压力传感器

26.1 系统分析

测量系统由压力传感器、放大器和 MCU 组成，每个部分都有器件差异和温度效应，应该分别进行分析。例如，8 位 A/D 转换器的量化误差约为±0.2%，应从总的最大误差中减去这一误差，才是其他器件的允许误差。MPX2000 系列压力传感器的输出灵敏度为 4mV/V(满量程压力)，在电源电压为 5V 时，输出 20mV(满量程压力)。

必须考虑 A/D 和压力传感器的线性，以确定系统的结构，并由此设计不同形式的信号放大电路。

运算放大器(op amp)应具有良好的共模抑制比，以消除压力传感器上直流电压的影响。另外，大多数应用场合的精度在 0～40kPa 范围内不受影响，这样，系统只要在 40～100kPa 范围内获得可以接受的误差就可以了。

26.2 压力传感器的特性

图 26-2 是 MPX2000 在 25℃的差分输出电压。若系统中其他元件不引入额外误差，其输出电压决定了不进行校准时系统可获得的最好误差。

温度对满量程输出和零点的影响如图 26-3 所示。注意，零点变化大于满量程输出，两者都有+8.0μV/℃的正温度系数(+5V 电源)。这说明满量程输出可以由放大器中能产生负温度系数(−250ppm/℃)的元件进行补偿。

26.3 运算放大器的特性

因系统只有一个电源，采用如图 26-4 所示仪表放大器对于压力传感器是很合适的。这种放大器具有很高的共模抑制比 CMRR 和输入阻抗，这种结构只需要极少的元件来调整放大器的增益。反馈电阻 Rf 采用温度系数为−250ppm/℃的电阻，可方便地补偿压力传感器温度系数。

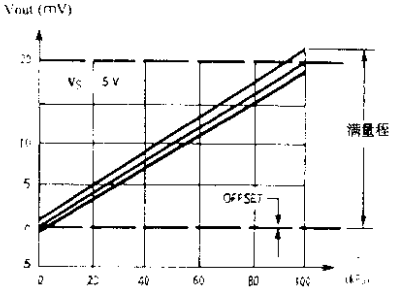


图 26-2 传感器输出电压与所受压力的关系

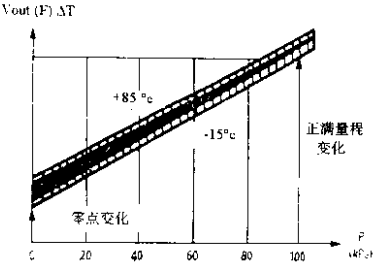


图 26-3 输出电压/零点与温度的关系

放大器的差模电压增益 Avd 为：

$$Avd=(V1-V2)/(Vs2-Vs4)=(1+2Rf/Rg)$$

运算放大器的主要误差源是失调电压和输入偏置电流。采用不同运放时的输出电压 V1 和 V2 如图 26-5 所示。图 26-6 和图 26-7 分别是 MC33078 运算放大器的输入失调电压和输入偏置电流随温度的变化关系。必须精心设计以获得最大输出电压摆幅，提高 A/D 的分辨率。

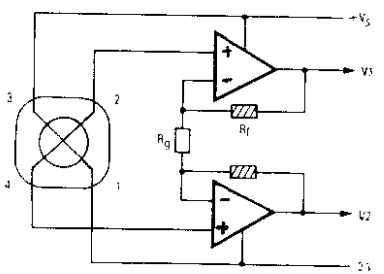


图 26-4 差分输出放大器

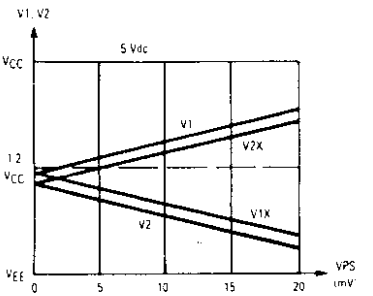


图 26-5 不同仪表放大器的电压传输特性

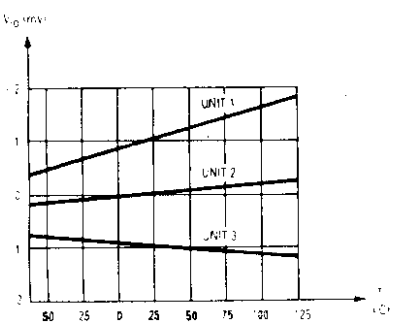


图 26-6 不同 MC33078 运放的输入失调电压与温度的关系

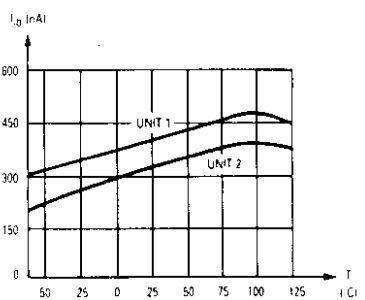


图 26-7 不同 MC33078 运放的输入偏置电流与温度的关系

这种方法必须花很大精力和时间进行零位校准和温度补偿。下面介绍的方法能消除任何校准步骤。

26.4 MCU 单片机的作用

采用 MCU 可以很容易地消除零点偏差，其电路原理如图 26-8 所示，仪表放大器的输入端增加由单片机 I/O 引脚控制的模拟开关。由于仪表放大器具有很高的输入阻抗，模拟开关的导通电阻不会引入误差。

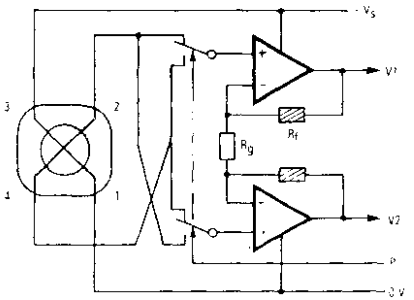


图 26-8 输入端由 MCU I/O 引脚控制的仪表放大器

借助于这两个模拟开关，MCU 能获得输出信号 V1、V2。首先，模拟开关处于某位置时将 V1、V2 进行转换，并存储在寄存器 R1、R2 中。然后，模拟开关位置互换，将这时的输出电压 V2x 和 V1x 进行转换，并存储于寄存器 R2x 和 R1x 中。然后，MCU 进行下列计算：

$$RESULT=(R1-R2)+(R2x-R1x)$$

该结果是两倍的差分电压值。这种方法可以消除由仪表放大器产生的所有误差。

26.5 系统计算

具体定量分析如下：

传感器 2 端的输出为：$Vs2=a(P)+of2$

传感器 4 端的输出为：$Vs4=b(P)+of4$

放大器的输出 V1 为：$V1=Avd(Vs2+OF1)$

放大器的输出 V2 为：$V2=Avd(Vs4+OF2)$

交换模拟开关位置后，两个输出端电压分别为：

$$V1x=Avd(Vs4+OF1)$$

$$V2x=Avd(Vs2+OF2)$$

则

$$V1-V2=Avd(Vs2-Vs4+OF1-OF2)$$

$$V2x-V1x=Avd(Vs2-Vs4-OF1+OF2)$$

这两式相加，则

$$\begin{aligned}Vout\ V=RESULT&=2*Avd(Vs2-Vs4)\\&=2*Avd*[a(P)+of2-b(P)-of4]\\&=2*Avd*[V(P)+Voffset]\end{aligned}$$

可见，这样能完全消除放大器的失调 OF1 和 OF2。最后的 A/D 结果值完全对应于输入电压。

图 26-9 是 MCU 使用两个输入模拟通道和一个输出引脚实现的压力传感器放大/转换系统。转换的数字量为：

$$COUNT=Vout\ V*255/(V_{RH}-V_{RL})$$

其中 V_{RH} 和 V_{RL} 是 A/D 的基准电压，通常分别接＋5V 电源和地。255 是 A/D 的最大数字量值。

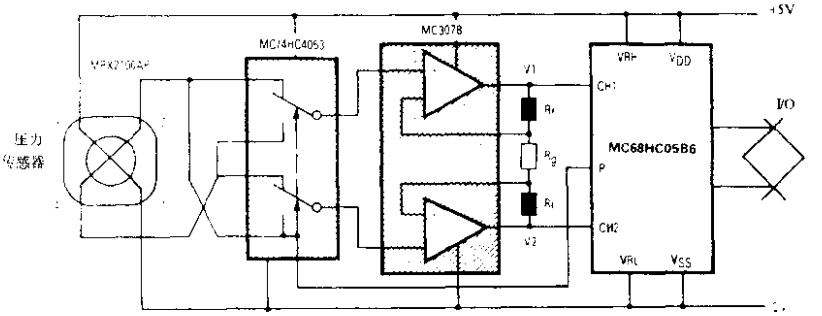


图 26-9　采用 MCU 的压力传感器系统

当允许的满量程压力误差在±2.5%范围内时，压力传感的零点偏移可以忽略，即系统不需任何校准步骤。

若 $V_{RH}-V_{RL}=5V$，则

$$COUNT=2*Avd*V(P)*51/V$$

其中 Avd 是放大器的差模增益，若 Rf＝510kΩ，Rg＝9.1kΩ，则 Avd＝113。

在满量程压力下，MCU 寄存器的最大数字量为：

$$COUNT(Full\ Scale)=2*113*0.02V*51/V=230$$

上述计算中假设 MPX2100AP 压力传感器在 5V 电源和 100kPa 满量程压力下的输出电压为 20mV。系统的分辨率为 100kPa/230，即每个计数值 0.43kPa。

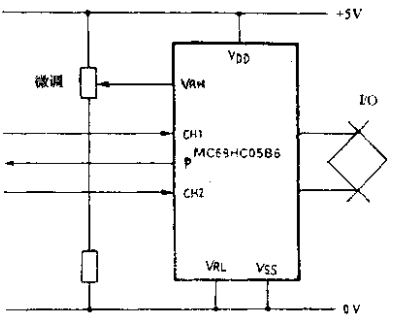


图 26-10　采用基准电压 $V_{RH}-V_{RL}$ 的满量程校准

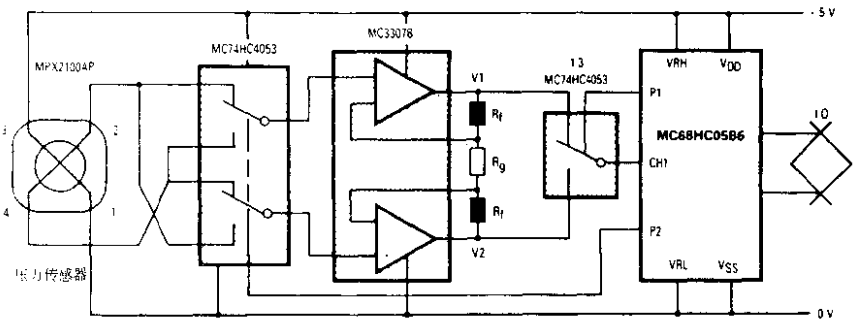


图 26-11　MCU 使用一个输入通道和两个输出通道的系统结构

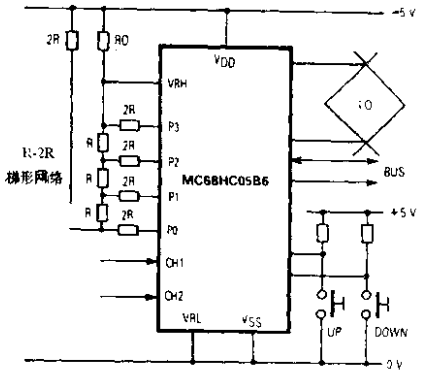


图 26-12　MCU 校准系统结构

如果要求系统误差在±1%范围内，应按图 26-10 所示方法进行校准，即在当地大气压力条件下，在 25℃、100kPa 满量程时，使 A/D 转换值为满量程值。

由于 MC68HC05B6 MCU 中的 A/D 具有很高的输入阻抗，也可以采用如图 26-11 所示的另一种电路结构，它只使用一个输入通道。需要注意的是，这里可以采用任何运算放大器，有利于减小系统成本。

当要求精确的满量程压力时，可采用各种校准方法，例如，可直接用 Up/Down 按钮校准。如图 26-12 所示，用 MCU 的 I/O 引脚直接驱动 R－2R 梯形网络电路，可以改变 A/D 转换器的 VRH 电压，从而调整满量程值。

不同压力单位的相互转换如表 26-1 所示。

表 26-1　压力转换表

单　位	Pa	mbar	Torr	atm	at＝kp/cm²	mWS	psi
1Nm²＝1 帕(Pa)	1	0.01	75.10³	—	—	—	—
1mbar	100	1	0.75	—	—	0.0102	0.014
1Torr＝1mmHg	133.32	1.333	1	—	—	—	0.019
1atm①	101325	1013.2	760	1	1.033	10.33	14.69
1at＝1kp/cm²②	98066.5	981	735.6	0.97	1	10	14.22
1 米水压(mWS)	9806.65	98.1	73.56	0.097	0.1	1	1.422
1lb/sqin＝1psi	6894.8	68.95	51.71	0.068	—	—	1

①标准大气压；

②技术大气压。

27. 用单片机提高压力传感 A/D 转换分辨率的方法

本文论述使用 8 位 A/D 获得大于 8 位分辨率的方法。这种方法非常简单，并且使用标准元件。

27.1 原理

用压力传感器和放大器可将压力转换为模拟电压输出，用图 27-1 所示的电路可将模拟电压转换为数字量，再由单片机进行处理。

若压力传感器量程为 200kPa。200kPa 压力时产生的输出电压为 4.5V，则系统的灵敏度为：

$$S=4.5V/200kPa=0.0225V/kPa$$

即

$$S=22.5mV/kPa$$

若使用 8 位 A/D，高低基准电压分别为 0V 和 5V，则分辨率为：

$$Rv=5V/(256-1)=0.01960V=19.60mV$$

即分辨率为每位 19.60mV，对应的压力分辨率为：

$$Rp=(19.60mV/位)/(22.5mV/kPa)$$
$$=0.871kPa/位$$

若需要的分辨率为 0.1kPa/位，则至少应采用 12 位的 A/D。实际应用时可以采用如下所述措施提高 A/D 的分辨率。

对于图 27-1，设测量压力为 124kPa，该系统中 A/D 的输入电压信号为(假设没有零点误差)：

$$Vm(测量)=124kPa\times 22.5mV/kPa=2790mV$$

经过 A/D 转换后，MCU 接收的数字量为：

$$M=2790mV/(19.60mV/位)$$
$$=142.35$$
$$=142(取整)$$

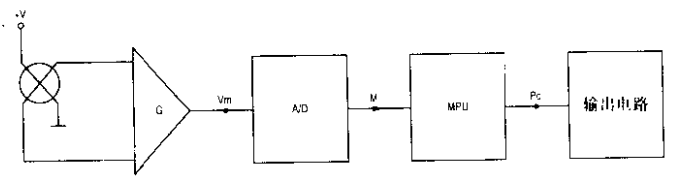


图 27-1 系统框图

由该数字量计算得的电压为：

$$Vc(计算)=142\times 19.60mV=2783mV$$

单片机将存储的 M 值对应的电压输出至 D/A，D/A 输入电压为 2783mV。该电压对应的压力

值为：

$$Pc(计算)=2783mV/(22.5mV/kPa)$$
$$=123.7kPa$$

压力误差为：

$$E=实际压力值-计算压力值$$
$$=124kPa-123.7kPa$$
$$=0.3kPa$$

可见，该误差大于要求的 0.1kPa 分辨率。

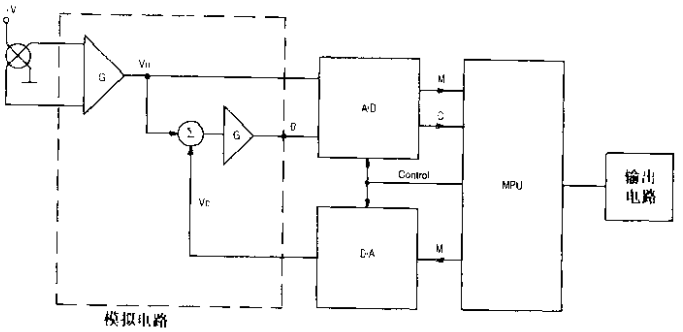


图 27-2　减小误差的系统框图

图 27-2 所示的系统可用来减小由 A/D 分辨率较低造成的误差。单片机使用存储的 M 值使 A/D 产生相应的 D/A 输出电压 Vc，用差分放大器从测量电压 Vm 中减去 Vc，将其差值进行放大。设该放大器增益为 10，则输出为：

$$D=(Vm-Vc)*G$$
$$=(2790mV-2783mV)*10$$
$$=70mV$$

该电压经 A/D 转换后的数字量为：

$$C=70mV/(19.60mV)=3.6=3(取整)$$

单片机按如下方法计算实际压力：

$$修正后电压=Vc+[(C\times R)/G]$$
$$=2783+3\times 19.60/10$$
$$=2789mV$$

其中 R 是 8 位 D/A 的分辨率。

修正后电压对应的压力为：

$$修正后的压力=2789mV/(22.5mV/kPa)$$
$$=123.9kPa$$

压力误差为：

$$124kPa-123.9kPa=0.1kPa$$

图 27-3 和图 27-4 是详细的模拟电路部分。

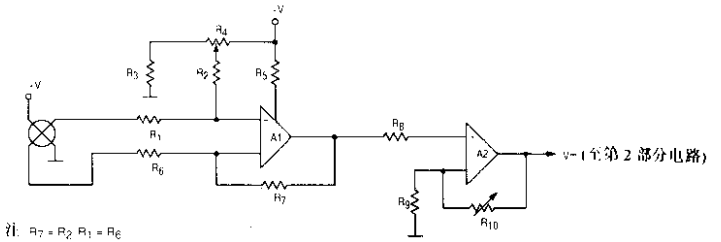


图 27-3 第一部分:差分放大器、零点和增益调节

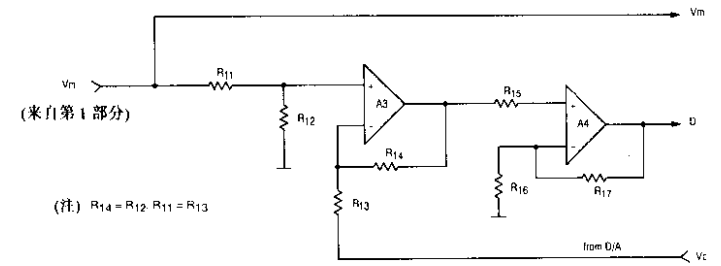


图 27-4 第二部分:差分放大

27.2 第一部分放大电路

这一部分包括压力传感器、零点和增益调整。这里使用 MPX2200 压力传感器,其满量程 200kPa 压力时输出为 40mV。压力传感器的输出接到 A1 放大器的输入端,其增益 G1 为 R7/R6。压力传感器典型的零位电压为 1mV,图 27-3 也给出了所需的零位误差补偿电路。A1 的输出送至 A2 放大器的同相输入端,其增益 G2 为(1+R10/R9)。设计的 G2 应使满量程压力时输出电压为 4.5V。

27.3 第二部分放大电路

A2 放大器的输出(Vm=G1 * G2 * Vs)接至 A3 放大器的同相输入端和 A/D 输入端。该放大器将 Vm 和由 D/A 得来的 Vc 电压的差值进行放大。A4 放大器为所需的分辨率提供合适的差模增益,其输出 D 为:

$$D=(Vm-Vc)\times G3$$

$$G3=(R14/R13)(1+R17/R16)$$

其中 G3 是 A3 和 A4 的增益。A1～A4 放大器均为 1/4 四运放 MC33079。

28. 补偿型压力传感器与单片机/数字电路的接口电路

象 MPX2000 系列这类补偿型半导体压力传感器比较容易与 MCU 或其他数字电路接口。这里介绍的电路将压力转换为 0.5～4.5V 输出电压，该输出可直接与 MCU A/D 转换器输入端或数字电路相连，也可以直接与 LM3914 相连，很容易构成巴图(Bar Graph)压力计。

图 28-1 所示的评估板采用 MPX2100 压力传感器，可以测量 100kPa 压力范围。它有两个输入口，P1 为压力口，位于传感器的上部；P2 为真空端口，位于传感器的下部。P1 和 P2 分别可承受 100kPa 的压力或高达 100kPa 的差压。任何相同的压差都产生相同的输出。

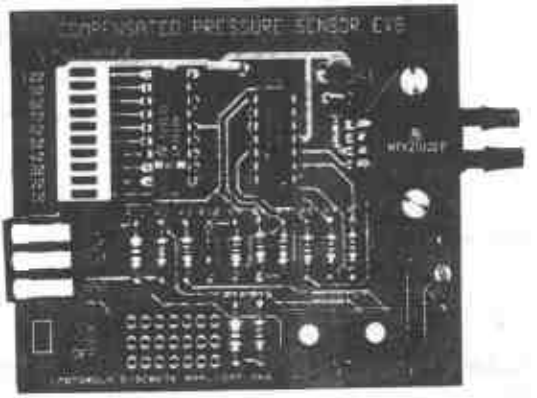


图 28-1 补偿压力传感器评估板

其基本输出是 10 段 LED 巴图显示器，依次表示增加满量程压力的 10%，即 10kPa(对于 MPX2100)。模拟输出为 0.5～4.5V，即当压差为零时输出为 0.5V，满量程差压(100kPa)时输出为 4.5V。零压力和满量程压力时的输出可以分别独立调节。

28.1 电性能和端子说明

评估板电路如图 28-2 所示，其印刷电路板(PCB)的丝网面如图 28-3 所示。主要电气性能如表 28-1 所示。端子说明如下：

表 28-1 电路主要性能

参 数	符 号	最 小	典 型	最 大	单 位
电源电压	B+	6.8	—	13.2	V
满量程压力	P_{FS}	—	—	100	kPa
过压	P_{MAX}	—	—	700	kPa
满量程模拟输出电压	V_{FS}	—	4.5	—	V
零压力模拟电压	V_{OFF}	—	0.5	—	V
模拟电压灵敏度	S_{AOUT}	—	40	—	mV/kPa
静态电流	I_{CC}	—	40	—	mA
满量程电流	I_{FS}	—	160	—	mA

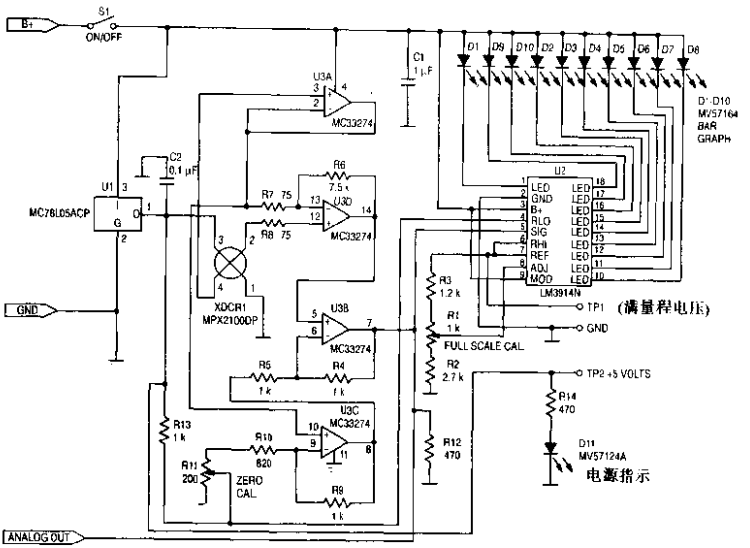


图 28-2　补偿型压力传感器评估板电路图

B+

电源输入端，其最大和最小电压值分别为 13.2V 和 6.8V。上限值取决于 10 个 LED 都亮时 LM3914 的功耗。

OUT

模拟电压输出端。在零压力和满量程压力时，OUT 的输出电压分别为 0.5V 和 4.5V。零压力时的输出电压由 R11 调节。该输出可直接与 MCU 的 A/D 通道接口，例如接至 M68HC11 的 E 口或 MC68HC05B6 的 A/D 输入引脚。

GND

有两个地连接端。电路中左边的地连接端 GND 为电源负极，右边的 GND 为的是便于与仪表地相连接。

TP1

测试点 1，与 LM3914 满量程基准电压相连。该电压通过 R1 调节满量程压力。

TP2

测试点 2，与＋5V 输出电压相连。通过该测试点，可以验证电源电压是否在 4.75～5.25V 范围内。

P1 和 P2

压力端口和真空端口，这两个口均没有标记。正常测量范围为 100kPa，最大安全压力为 700kPa。

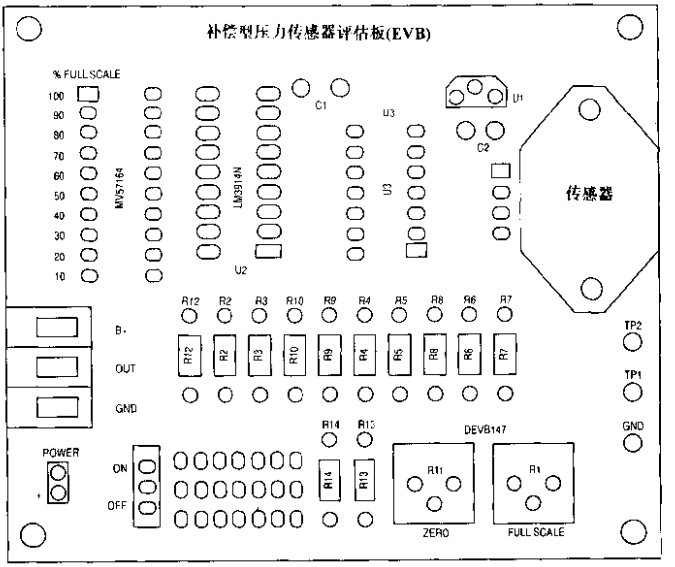


图 28-3 电路板丝网面

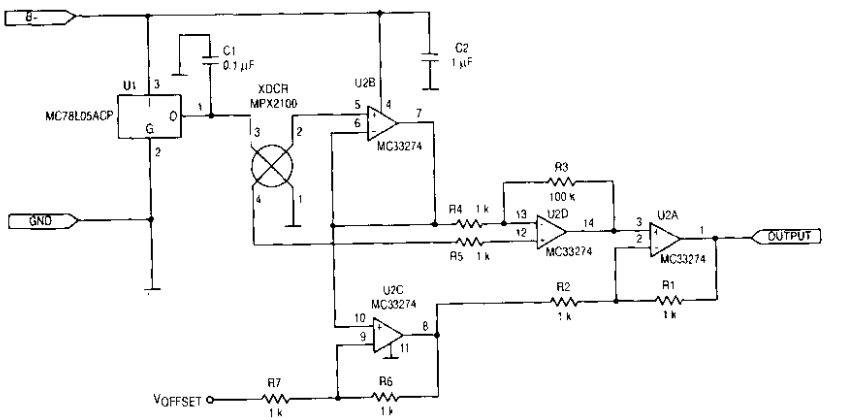


图 28-4 补偿型压力传感器接口电路

28.2 设计考虑

该典型应用设计的关键是如何获取小的直流差动信号，并经过放大处理后产生以地为基

准的输出电压，以便能驱动MCU的A/D输入端。合理的接口电路如图28-4所示。它采用一个四运放和一些电阻对传感器的电压信号进行放大、转换，并进行电平位移。放大器的增益主要是接成差动方式的U2D，U2B使得U2D与传感器的正输出端隔离。U2B的目的是防止R3和R4的反馈电流流入传感器。在零压力时，压力传感器2脚与4脚的电压为零。例如，放大器输入端共模电压为2.5V，零压力时U2D的14脚的输出电压为2.5V，U2C和U2A将该2.5V零压力直流输出电压进行电平位移并转换为所需的零压力偏移电压V_{OFFSET}。下面假设V_{OFFSET}为0.5V，具体分析转换过程。

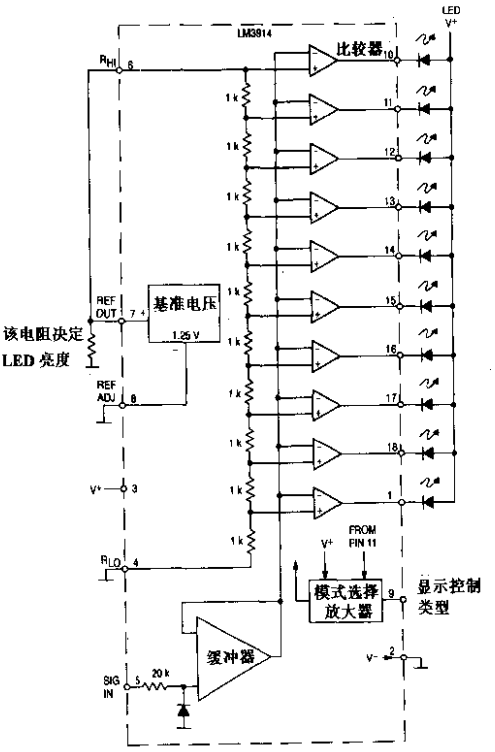


图28-5 LM3914框图

引脚10为2.5V，则引脚9也是2.5V，R7的电压为2.5－0.5＝2.0V。由于放大器输入端电流为零，R6与R7的电流相同，则R6的电压也是2.0V。引脚8的电压为0.5V＋2.0V＋2.0V＝4.5V。同样地，引脚2和3的电位都是2.5V，R2的压降为2.0V，则R1的压降也是2.0V，故引脚1的电压为0.5V。由上述分析可见，只要满足R6/R7＝R2/R1，则直流输出电压与传感器的共模电压无关。

该电路的增益接近于但并不严格地是R3/R4(R1/R2＋1)，按图中值，增益为200，严格分析结果增益为199.9。

U2A的0.5V～4.5V输出可直接与单片机的A/D输入端接口，也可直接连接至LM3914。LM3914的结构框图如图28-5所示。比较器输入端分压器的RLO直接接地，每个电阻的电压为0.5V。在图28-2中，通过R13和电位器R11来调节RLO电位，使得R11的电压正好是图28-4中的V_{OFFSET}。图中R10和R11的选择应使得U3C的源阻抗近似为1kΩ。R11电位器的输出接至RLO，用于巴图压力计校零。

借助于R2和R3，调节R1改变比较器的基准电压，使满量程压力传感器的输出电压与满

量程压力相对应。LED 直接与 LM3914 的输出相连，LED 电流约为 R1、R2、R3 电流的 10 倍（R1～R3 的电流为 9.2mA）。

28.3 应用

模拟输出可直接为 MCU 提供压力信息。输出电压范围设计为 0.5V～4.5V（零压力～满量程压力）是为了最佳利用 MCU 的 A/D 输入特性，减小误差。使用时可将评估板的模拟输出直接连到 MCU A/D 的输入端，如 M68HC11 的 E 口任意引脚，如图 28-6 所示。为获得最好的 A/D 转换精度，V_{REFH} 接至 4.85V，V_{REFL} 接至 0.3V，这两个电位由精密电阻分压获得。

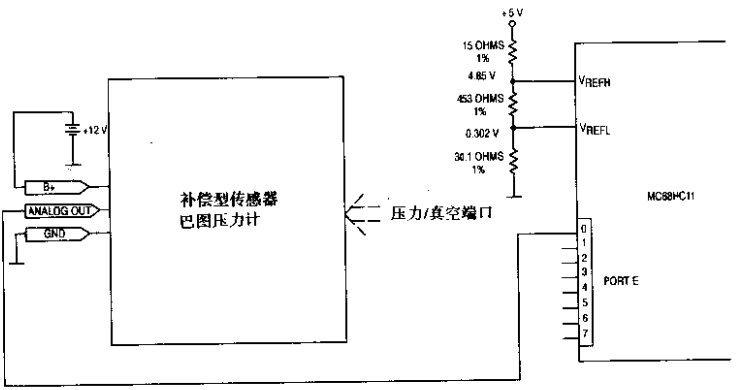


图 28-6　应用举例——接口电路与 M68HC11 连接方法

29. 能与 MCU 接口的 MPX2000 系列压力传感器频率输出系统

通常，半导体压力传感器将压力转换为小信号电压。再对传感器的输出进行温度补偿，并放大为较大模拟信号。这种方法中温度补偿电路和信号调节电路往往额外增加大量成本。另外，还应考虑输出信号如何应用或如何与其他电路进行接口。大多数情况下，放大后的传感器信号输出至单片机(MCU)、控制逻辑电路或模拟电路。

随着智能化单片机的普及，8 位单片机已成为传感器系统必不可少的一部分。为使压力传感器的电压输出信号与单片机连接，MCU 必须具有片内 A/D，或另加 A/D 转换器。在外加 A/D 转换器的情况，若 A/D 与 MCU 并行通讯，则占用 I/O 较多。若以串行方式与 MCU 通讯，则要求 A/D 具有与 MCU 一致的通讯协议。可见系统较为复杂。

单片机的特点之一是测量脉冲周期或脉冲宽度极为方便。传感器与 MCU 的另一种连接方法是将传感器输出转换为频率信号，由 MCU 的定时/计数器系统测量。通常的 MCU 都具有定时器，这样在选择 MCU 时就更为灵活。多数 MCU 定时器的分辨率都在 1μs 以下。若将传感器输出转换成与所加压力成正比的频率信号，则可用廉价(无 A/D)MCU 构成精密的传感器应用系统。另外，这种系统抗噪声干扰性能好，适用于有噪声的应用环境和遥感场合。

29.1 设计考虑

MPX2000 系列是经温度补偿和校准的压力传感器(零位和满量程都精密微调)，量程为 10kPa(1.5PSI)至 700kPa(100PSI)。传感器的输出与电源电压成正比。在规定的最大电源电压(16V)时，传感器的满量程压力输出为 64mV，但 MPX2010 的满量程输出为 40mV。压力传感器的主要性能如表 29-1 所示。许多仪表放大器可作为传感器的信号调节电路，也可以用运算放大器(op amp)对信号进行放大处理，取决于所要求的精度和温度特性。输出信号应尽量利用电源电压范围，使输出有较大幅度。另外，还要求能灵活调节零压力和满量程输出。所用的电压/频率变换电路(V/F)决定在何处和如何进行调节/校准。

1. 电压－频率转换。因半导体压力传感器为电压输出，必须将电压信号转换为与电压成正比的频率信号。因输出电压与所受压力成正比，故频率与所受压力成正比。有许多商品化的电

表 29-1 传感器的主要性能

参　数	符　号	最小值	典型值	最大值	单　位
电源电压	B+	10		16	V
满量程压力	P_{FS}				
MPX2010				10	kPa
MPX2050				50	kPa
MPX2100				100	kPa
MPX2200				200	kPa
MPX2700				700	kPa
满量程输出	f_{FS}		10		kHz
零压力输出	f_{OFF}		1		kHz
灵敏度	S_{AOUT}		$9/P_{FS}$		kHz/kPa
静态电流	I_{CC}		55		mA

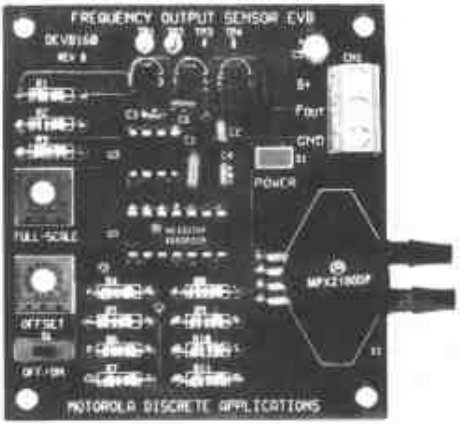


图 29-1 MPX2000 传感器的频率输出系统

压—频率转换器可完成 V/F 变化这一功能。

2. 开关时间。V/F 变换器的开关时间影响脉冲或方波频率信号，开关速度取决于检测开关边沿所用的硬件。M68HC05/M68HC11 系列 MCU 的输入捕捉功能可方便地捕捉上升或下降沿。对于高速 CMOS 电路，其最大上升或下降时间为几百纳秒(ns)，能有利于加速 V/F 变换器的开关边沿。

29.2 应用

采用 V/F 变换除了不需用 A/D 外，频率输出还可用于长距离传输情况或有噪声干扰的环境。若输出为电压信号，则电磁场感应的噪声将影响实际的电压。而频率信号具有很高的抗干扰性能，并在输入到 MCU 时能有效地滤除噪声。对于 50/60Hz 主要噪声源，若频率信号在 kHz 范围内，则很容易滤除 50/60Hz 干扰噪声。

29.3 频率输出电路

图 29-1 是压力传感器的信号调节和频率变换电路板。它将压力转换为单端输出电压，然后转换为频率信号。在零压力时输出为 1kHz，满量程压力时输出为 10kHz。可调节零压力和满量程压力输出频率，该电路适合于 MPX2000 系列中任何型号的压力传感器。但对于 MPX2010 传感器，需要调节电阻 R5，改变增益，以获得所需的满量程压力。

频率输出接口电路如图 29-2 所示。各端子说明如下：

1. B+。电源输入端，其最大和最小值分别为+16V 和+10V。

2. F_{OUT}。电平为 5V 的频率输出端。零压力和满量程压力时的输出频率分别为 1kHz 和 10kHz。零压力频率由 R12 调整，满量程压力由 R13 调整。该端能直接与 MCU 定时器的输入捕捉引脚相连。

3. GND。电源和信号地。TP3 也连接至地，以便于测试。

4. TP1～TP4。测试点。TP1 为频率输出；TP2 为+5V 电源输出；TP3 为地；TP4 为+8V 电源输出。

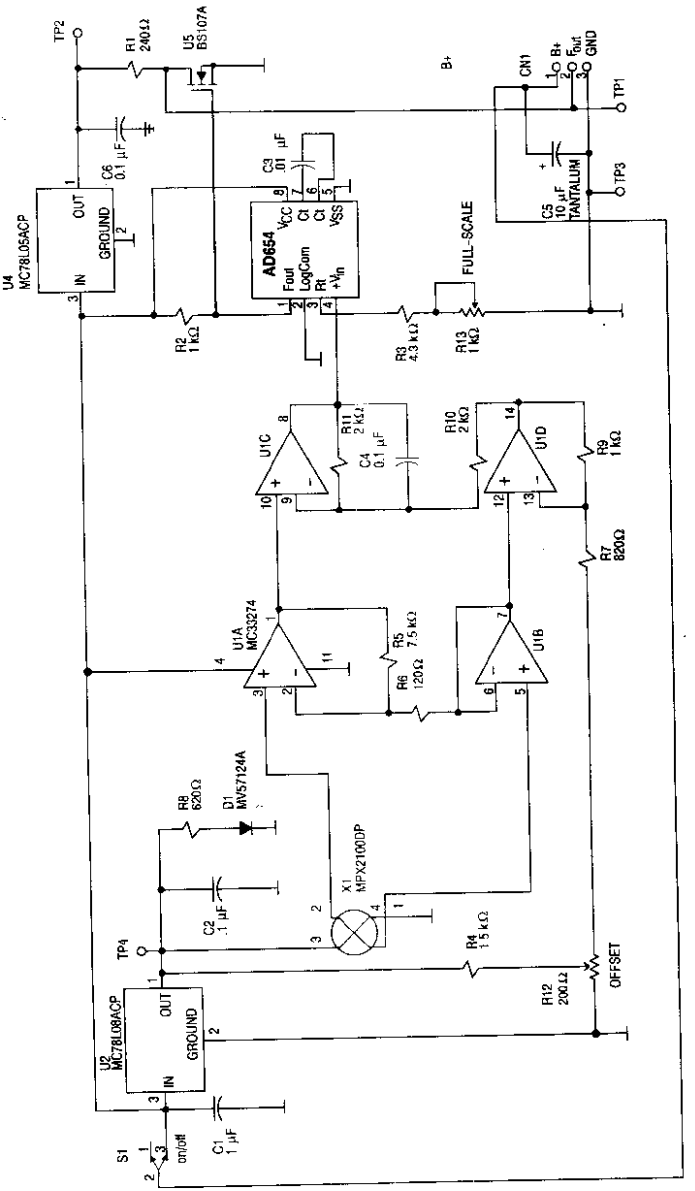


图 29-2 MPX2000 系列传感器的频率输出电路图

29.4 电路操作原理

该电路的电压信号放大部分是传统的仪表放大器的一种变型形式，它具有差模增益高、共模抑制比好和很高的输入阻抗，还很容易进行零点和偏置调节。它采用四个运放，增益主要由U1A差分放大器获得。U1B防止反馈电阻的电流流经传感器。零压力时，传感器引脚2至引脚4之间的电压也精确地调到零。传感器这两引脚的共模电压都是4V(电源电压的一半)。在零压力时，U1A的引脚1的输出电压也是4V。U1C和U1D使得输出为所需的零压力输出电压。零压力电压由R4和R12获得。R7的选择应使引脚13的电阻约为1kΩ。该电路的增益为：

$$G=\frac{R5}{R6}(1+\frac{R11}{R10})$$

按图中电阻值，G=125。选择增益为125是为了使压力传感器的32mV满量程输出产生4V的电压幅度(传感器在10V电源时的满量程输出为40mV，在8V电源时满量程输出为32mV)。从U1C输出的0.5～4.5V电压然后由V/F变换器变换为1～10kHz频率信号。AD654 V/F变换器接收U1C运放引脚8的传感器输出信号。满量程频率取决于R3、R13和C3，其关系为：

$$F_{OUT}(满量程)=\frac{V_{in}}{(10V)(R3+R13)C3}$$

最好情况下，R3和R13在满量程电压时，应在AD654的引脚3产生1mA的驱动电流。AD654的输入级是一个运算放大器，引脚3和4是其输入端，引脚4的电压与引脚3相同。由于传感器信号放大后的满量程输出为4.5V，故R3和R13应近似为4.5kΩ，以获得最佳性能。当U3的引脚3到地之间的电阻确定后，C3则决定V/F的满量程输出频率。满量程频率由R13调节，零压力频率通过调节传感器信号调节电路的零点电压进行调整。

小信号FET反相器(由MOS晶体管和R1组成)加速频率输出的转换边沿，最后的输出可以直接与MCU的定时器输入端或高速CMOS电路(74HC)连接。

29.5 测试/校准步骤

1. B+和GND之间加12V电源。

2. 频率计数器或示波器探头接F_{OUT}或TP1，仪器地接GND或TP3。

3. 接通电源开关S1，LED D1将亮。确认TP2和TP4分别为5V和8V，在频率输出端应有占空比为50%的方波信号。

4. 在MPX2100的P1口加100kPa的压力。

5. 调节微调电位器R13使输出频率为10kHz。若不能获得10kHz频率，调节微调电位器R12获得10kHz输出。

6. 在传感器的P1口加零压力(两个端口P1和P2均接大气，无差压)，调节OFFSET微调电位器R12使输出频率为1kHz。

7. 检验满量程时输出频率是否为10kHz，若不是，则重复上述4～6步骤，直到满量程和零压力时的输出频率分别为10kHz和1kHz为止。

29.6 结论

将传统的压力传感器电压信号转换为频率输出有许多优点，如，特别适于遥感和有噪声干扰的应用场合；线路较为简单；频率输出易于与MCU或HCMOS电路相连；可采用片内无A/D的廉价单片机等。

30. 半导体压力传感器与单片机的接口方法

多数压力传感器为差动电压输出形式，满量程输出电压通常为 25～50mV。其与单片机(MCU)的接口电路通常包括：微弱差分输出信号的获取、差分信号放大和单端化，并使输出信号幅度与 MCU 或其他电路相适应。另外，模拟压力电压信号还可以转换为频率信号或 4～20mA 电流环，这两种都具有很强抗干扰能力，适于长距离传送。这里具体介绍传感器的放大、A/D 转换、V/F 变换以及 4～20mA 电流环接口方法。

30.1 压力传感器

压力传感器为电桥式结构，如图 30-1 所示。当加有压力时，电阻 RP1 和 RP2 增加，而电阻 RV1 和 RV2 减小同样的量，从而造成电桥不平衡，在 S+和 S－产生差动电压。这种结构的特点之一是差分输出电压信号与电源电压成正比。这说明压力测量的精度直接与传感器上的电压误差有关。电桥电阻为硅膜电阻，具有正温度系数。电阻 RC1 和 RC2 近似相等，电桥输出电压基本固定为 B+的一半。对典型的 MPX2100 传感器，电桥电阻标称为 425Ω，RC1 和 RC2 为 680Ω。这种情况下若 B+为 10V，则电阻 1.8Ω 的变化在满量程压力时产生 40mV 输出电压。

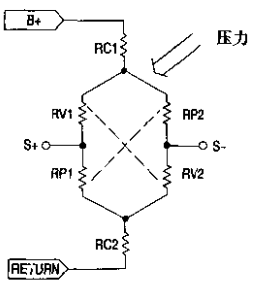


图 30-1 压力传感器等效电路

30.2 仪表放大器接口

仪表放大器是压力传感器常用的接口电路，由廉价仪表放大器构成的接口电路如图 30-2 所示。这里采用 MC33274 四运放构成仪表放大器，但有一点不同。对仪表放大器，电阻 R3 通常接地，这样差分传感器信号为零时，输出电压为零。单片机接口输入端通常要求在零压力时有大约 0.3～0.8V 的电压，因此 R3 接至 U1D 的 14 引脚，U1D 接成电压跟随器，由电位器 R6 获得适当电压。R6 上的电压也就是 U1C 输出端电压。

当 R10 为 240Ω 时，增益为 125，可将满量程压力输出的 32mV 电压放大至 4V。零压力电压设置为 0.75V，则输出电压范围为 0.75～4.75V，可直接与 MCU 的 A/D 输入端连接。

对于需要较高精度的情况，可采用象 LTC1100CN8 这样的全集成化仪表放大器，如图 30-3 所示。该放大器增益为 100，并将双端信号转换为单端输出。精密分压器和 U2B 提供零压力输出(0.5V)，该输出接到 LTC1100CN8 的引脚 1，相当于图 30-2 中 U1D 的输出接到 R3。另一级同相放大器 U2A 使输出的电压范围达到 4V。R2 也接到 U2B 的输出，以便使最后的零压力的输出为 0.5V。该放大器接口电路的输出电压范围为 0.5～4.5V。

上述两种电路各有特点，图 30-2 所示的电路成本低，分离器件较多；图 30-3 所示的电路性能好，因采用精密薄膜电阻，故精度高，但成本高。

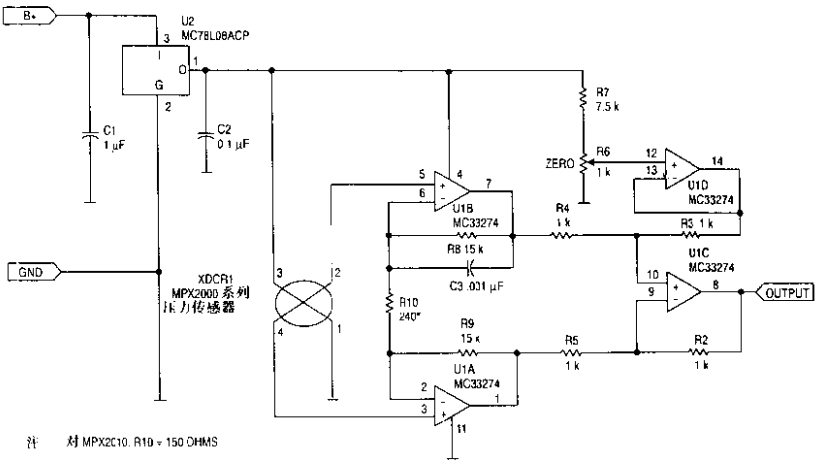


图 30-2 传感器与 MCU 的仪表放大器接口电路

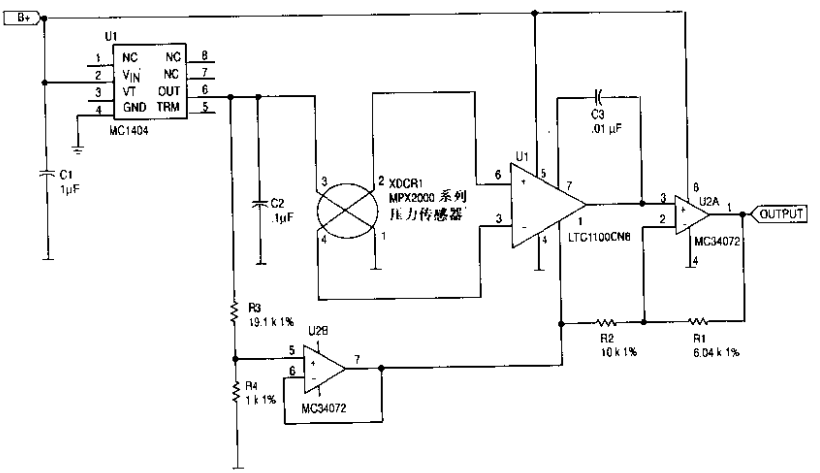
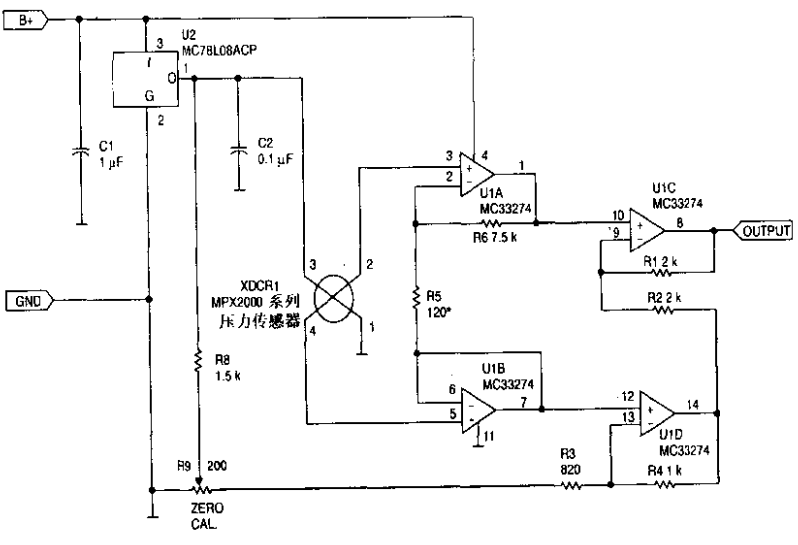


图 30-3 精密仪表放大器接口电路

30.3 传感器专用接口放大器

图 30-4 所示的专门用于压力传感器的廉价接口电路能改善图 30-2 所示电路的性能。它采用四运放和一些电阻对传感器信号进行处理。U1A 主要起放大作用，U1B 主要起隔离作用，防止反馈电阻 R5/R6 的电流流经传感器。零压力时传感器引脚 2 和 4 之间电压为零。对图中情况，传感器的共模输出电压为 4.0V，零压力时 U1A 的输出也是 4.0V。U1C 和 U1D 使输出在零压力时为所需的电压。

图 30-5 为简化的传感器专用接口电路。传感器输出共模电压为 4.0V，则 U1D 和 12 和 13 引脚也是 4.0V，R3 的压降为 4.0V－V_{OFFSET}，R4 的压降也是此值，即 U1D 的 14 脚为 7.5V（V_{OFFSET}＝0.5V）。由于 U1C 的 9 引脚也为 4.0V，则 R1 和 R2 的压降均为 3.5V。U1C 的输出（8 脚）为 0.5V（即 V_{OFFSET}）。只要满足 R4/R3＝R2/R1，则接口电路零压力输出电压与传感器的共模电压无关。在图 30-4 中，V_{OFFSET}取决于 R8 和 R9 电位器。



注：对 MPX2010，R5 = 75 OHMS

图 30-4 传感器专用接口电路

电路的增益近似为：

$$G=(R6/R5)(R1/R2+1)=125$$

可使满量程 32mV 的压力传感器输出信号达到 4V 的输出范围（B＋为 8V）。

该电路以及前述图 30-2 和图 30-3 所示的电路都可以直接与 MCU 的 A/D 输入端相连。用 M68HC11 作为例子，接口电路的输出可以接到 E 口的任一个引脚，图 30-6 是接到 PE0 的情况。图中为获得最好的 A/D 转换精度，V_{REFH}和 V_{REFL}分别接 4.85V 和 0.3V。

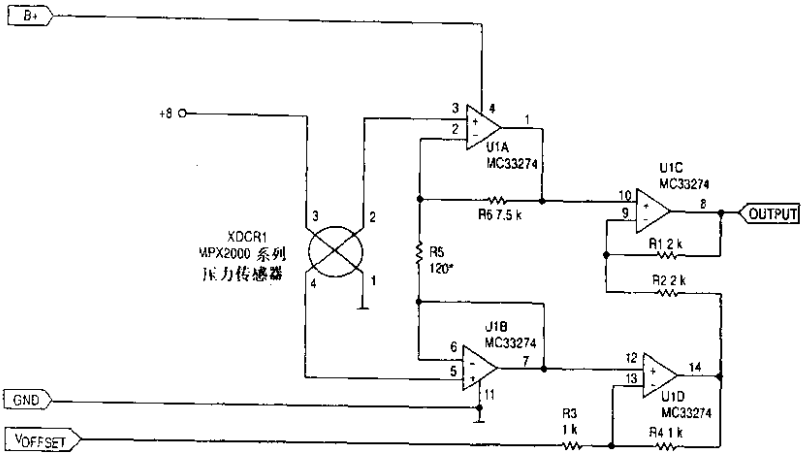


图 30-5　简化的传感器专用接口电路

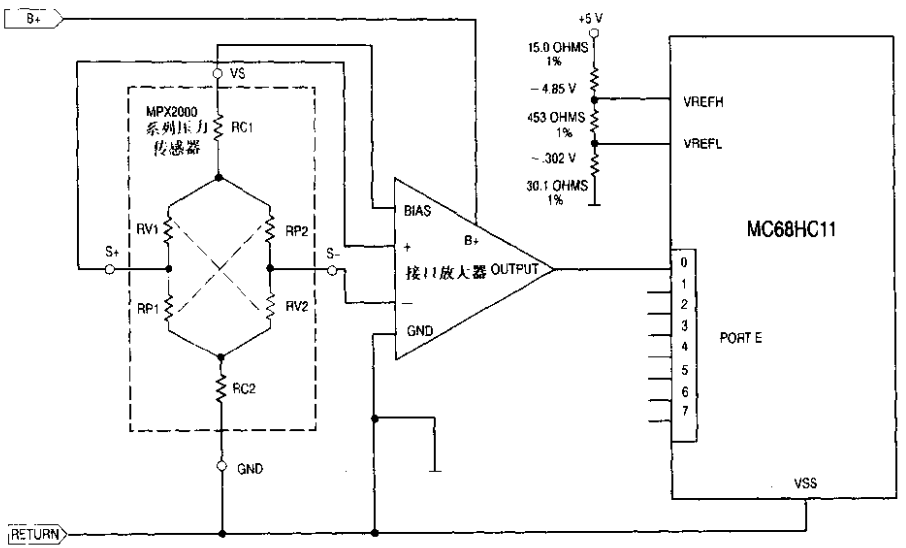


图 30-6　接口电路应用例子

30.4 单斜率 A/D 接口电路

单片机片内的 8 位 A/D 转换器通常适用于压力传感器，当需要的 A/D 位数大于 8 位时，图 30-7 所示的电路可将分辨率扩展到 11 位。

斜波发生器用电流源 U5 和电容器 C3 产生斜波电压，Q1 使 C3 的电压初始值近似为地电压。按图中元件值，470μA 电流流入 0.47μF 时，电压从 0 到 5V 的时间大约为 5ms。若传感器为零压力时比较器 U2A 和 U2B 为相同电压，因此，当电压上升到共模电压时，PA1 和 PA0 同时变为低。MCU 对 PA0 和 PA1 变低的时间间隔(时钟周期数)进行计数。零压力时计数值为零。

该电路中 U4A 和 U4B 形成仪表放大器的前端，将传感器信号进行差分放大。U1 和 U3 对放大后的差分信号进行抽样并保持。采样和保持的目的是为了使得在进行 A/D 转换期间输入信号稳定。在传感器承受压力时，U1 的输出电压高于 U3，因此，U2B 的输出状态将在 U2A 之前发生变化，PA0 变低与 PA1 变低有一定的时间间隔(时钟周期数)。该时钟周期数代表输出电压的大小，这样就完成了 A/D 转换。

当斜波电压达到 R9 和 R10 决定的基准电压时，比较器 U2C 翻转，通知 MCU 发出复位命令。复位时，PA7 控制 Q1 开启，PB1 使抽样和保持电路复位。分辨率取决于时钟频率和斜波电压的线性度。对图 30-7 中的斜波发生器和 2MHz 的时钟频率，分辨率为 11 位。

软件包括：抽样和保持电路的锁存、读 MCU 中自由运行计数器的值、关闭 Q1、等待 3 个比较器的输出状态由 1 变为 0。模拟输入电压由 PA0 和 PA1 变低之间的时钟周期数决定，分辨率为 2μs。

30.5 长距离接口电路

在传感器与 MCU 相距较远时，可采用如下两种接口方法：频率输出和 4～20mA 电流环。在频率输出情况，压力被转换为频率与压力成正比的脉冲信号(电压幅度 0～5V)。如果应用中要求最少的连线数量，4～20mA 电流环是最好的解决方法。它采用电源和地作为 4～20mA 信号线，只需两根线。流入传感器 4mA 对应零压力，流入 20mA 对应满量程压力。

压力至频率转换电路如图 30-8 所示，它包括 3 部分。接口放大器与图 30-4 相同，其 0.5～4.5V 输出直接接至 AD654 电压/频率(V/F)变换器，AD654 的 C3 设置输出频率。零压力时输出 1kHz，由 R3 调节；满量程输出为 10kHz，由 R12 调节。AD654 的输出然后接至缓冲器。由 Q1 和 R10 组成的缓冲器用来使输出整形，并将幅度变为 5V。其优点是 MCU 定时器易于获取频率输出。当距离很长时，需用 3 根线(电源、地和输出)与传感器连接。

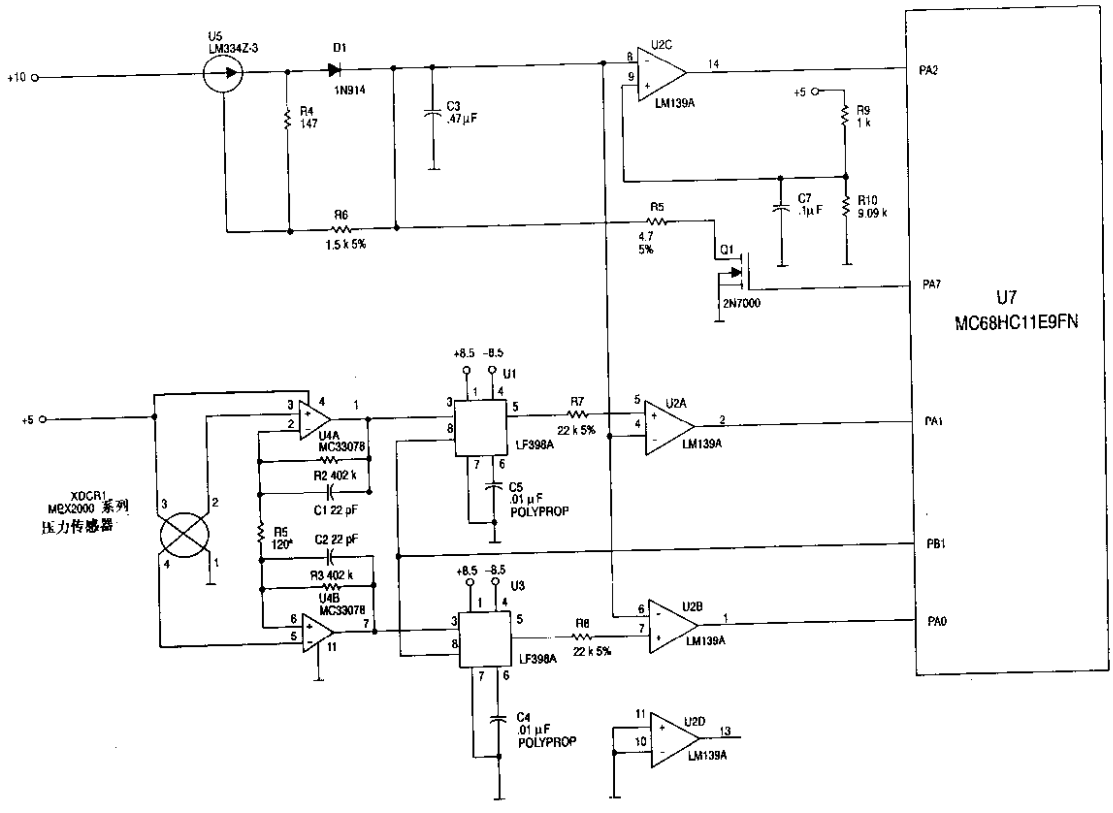
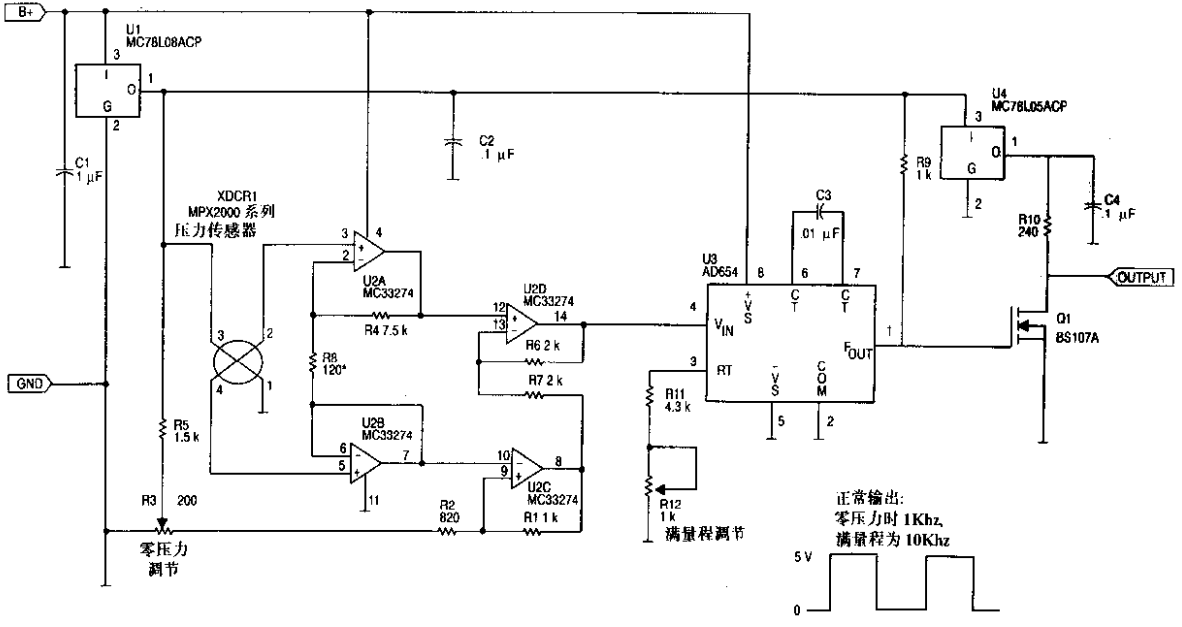


图 30-7　单斜率 A/D 转换器接口



* 注：对 MPX2010, R8 = 75 OHMS

图 30-8　压力传感器的频率输出接口

4～20mA 电流环能使连线减为 2 根，电流环如图 30-9 所示。其传感器为 MPX7000 高阻传感器。该电路在零压力和满量程压力时流入电流分别为 4mA 和 20mA。接收端为 240Ω 电阻，产生的电压信号为 0.96V～4.8V，它适于 MCU 的 A/D 输入端。

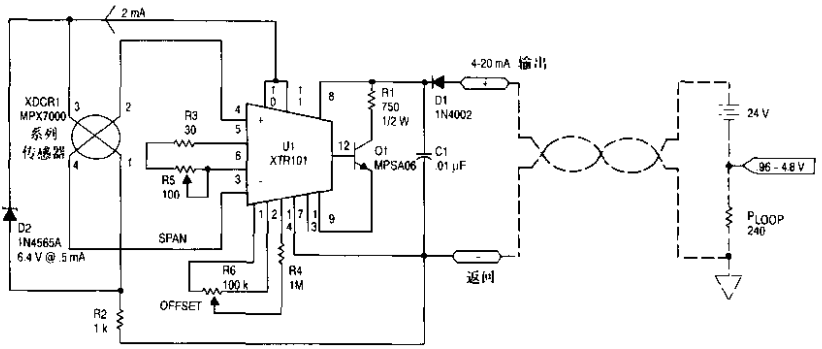


图 30-9　压力传感器的 4～20mA 电流环变送器接口

传感器的偏置由两个 1mA 电流源(XTR101 的 10 和 11 引脚)提供，IN4565A(6.4V)齐纳二极管进行温度补偿。传感器的输出接至 XTR101 的同相和反相输入端。由调节 R6 使零压力时电流为 4mA。当传感器偏压为 6.4V 时，传感器满量程输出为 24.8mV。R3 与 R5 之和为 64mA，通常产生 16mA 电流范围。由 R5 的调节使满量程为 20mA。

XTR101 差分输入端 3 和 4 脚要求的共模电压为 4～6V。R2 使共模电压提高 2V，使 3 和 4 脚电位达 5.2V。D1 起保护作用，防止极性接反造成破坏。这种结构的优点是简单和两线接口。

30.6　MCU 与集成化传感器直接接口

集成化传感器与具有片内 A/D MCU 的接口最为简单。图 30-10 是由 MPX5100 集成化传感器和 M68HC05 单片机组成的 LCD 压力计。虽然整个电路较为复杂，但传感器与 MCU 的接口极为简单，只需将传感器输入接至 MCU 的 A/D 输入端即可。MPX5100 内部有放大器，输出为 0.5～4.5V 信号。

系统软件应按如下方法设计：上电时 MCU 认为传感器所受压力为零，MCU 读传感器的输出电压，并做为零压力值存储起来。满量程由跳线 J1 和 J2 调节。例如，软件可设置为 J1 拔下/J2 插上时输出增加 1.5%；J1 插上/J2 拔下时输出减小 1.5%。

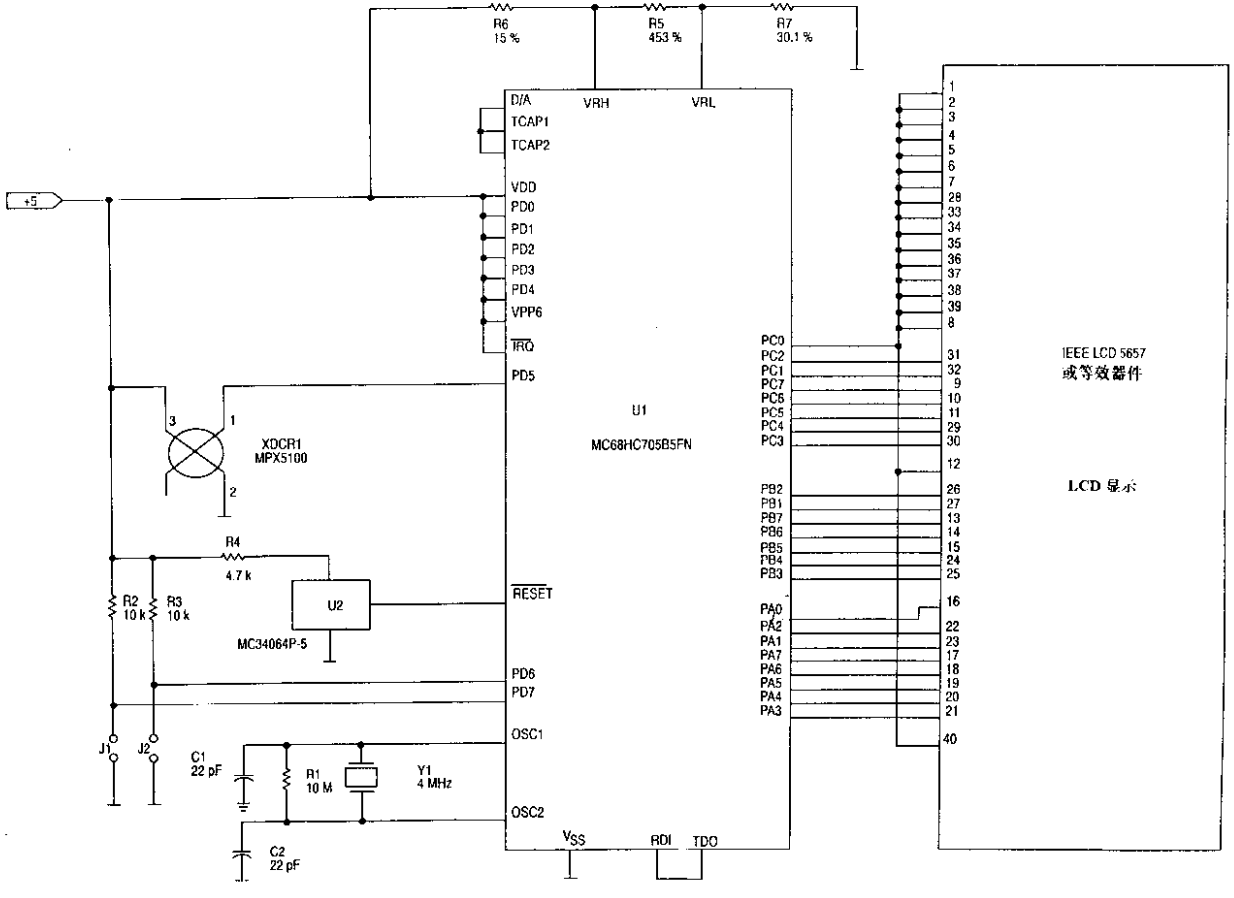


图 30-10　传感器和 MCU 构成的 LCD 压力计

31. MC68HC05C5 SIOP 与 I²C 外围器件的接口方法

使用单片机时经常遇到的问题是 MCU 片内不含有所需的所有外围功能，解决方法是外接外围器件。理想的接口是同步串行通讯口。本文说明 MC68HC05C5 的 SIOP(串行 I/O 口)如何与 I²C 外围器件 PCF8573 时钟/定时器进行接口。选择 MC68HC05C5 是因为它的 SIOP 具有可编程的时钟极性。

31.1 SIOP 简介

如图 31-1 所示，SIOP 是 3 线主/从系统，包括串行时钟 SCK、串行数据输入端 SDI 和串行数据输出端 SDO。可编程选择 SIOP 首先处理最高位(MSB)还是最低位(LSB)。有 3 个寄存器与 SIOP 有关。

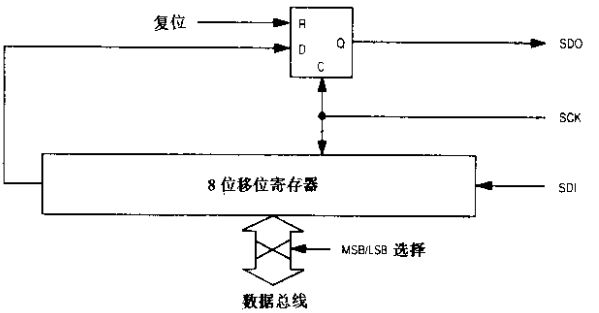


图 31-1 SIOP 框图

1. 串行时钟 SCK。两次传输之间 SCK 的状态可以为逻辑 1 或 0。SCK 信号的第一个下降沿开始传输过程。这时，接收数据的第一位呈现在 SDI 引脚，发送数据的第一位呈现在 SDO 引

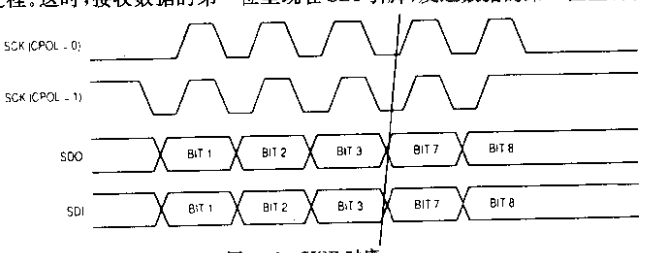


图 31-2 SIOP 时序

脚，如图 31-2 所示。当 CPOL(见后面的 SIOP 寄存器说明)为 0 时，第一个下降沿发生在 SIOP 内部。SDI 引脚在 SCK 的上升沿捕捉数据，随后的下降沿将数据移位。当 CPOL=1 时，在 SCK 的第 8 个上升沿结束传送过程；当 CPOL=0 时，在 SCK 的第 8 个下降沿结束传送过程。

当 SIOP 为主机时，内部产生 SCK 时钟，可输出，当 SIOP 为从机时，需将外部时钟加到 SCK 端。除此之外，主机模式与从机模式相同。主机模式的传输频率为 E/4(E 为总线时钟)。

2. 串行数据输出端 SDO。在 SCK 的第一个下降沿，移出的第一位数据出现在该引脚。允许 SIOP 时，PB5/SDO 用作 SIOP 的串行数据输出端。

3. 串行数据输入端 SDI。允许 SIOP 时，SDI 引脚为输入。在 SCK 的下降沿，新数据出现在 SDI引脚。有效数据必须在SCK上升沿之前至少100ns到来，并在该边沿之后100ns保持有效。

4. SIOP 控制寄存器(SCR，$0A)。该寄存器位于 $000A，含有 3 位，如下所示：

SCR	7	6	5	4	3	2	1	0
$000A	0	SPE	0	MSTR	CPOL	0	0	0
复位	0	0	0	0	1	0	0	0

SPE——SIOP 允许位。

1＝允许 SIOP，并使 B 口数据方向寄存器(DDRB)初始化为 PB5/SDO 为输出，PB6/SDI 为输入，PB7/SCK 在从机时为输入，在主机时为输出。

0＝禁止 SIOP，P 口恢复为标准并行 I/O 口。

SPE 在任何时候可读可写，正在传输期间将 SPE 清零会中止传送过程，使 B 口为普通 I/O 口。

MSTR——主/从模式控制位。

1＝SIOP 为主机模式，SCK 为输出，向数据寄存器写入开始传输过程。同步数据时钟固定为总线频率的 1/4。

0＝SIOP 为从机模式。

复位时 MSTR 为 0。可在任何时候将 MSTR 置位，它与 SPE 无关。将 MSTR 清零将终止任何正在进行的传输。

5. SIOP 状态寄存器(SSR，$0B)。SSR 位于 $000B，它含有两位，如下所示：

SSR	7	6	5	4	3	2	1	0
$0B	SPIF	DCOL	0	0	0	0	0	0
复位	0	0	0	0	0	0	0	0

SPIF——SIOP 标志位。

1＝在最后一个时钟的上升沿 SPIF 置位，表明已完成数据传输。SPIF 置位时读 SSR，随后读或写入 SDR(SIOP 数据寄存器)，则 SPIF 清零。

0＝没发生数据传输。

DCOL——数据冲突标志位。只读位。

1＝表明发生无效的数据寄存器访问。在传输期间读或写入将导致发送或接收的数据无效。读 SSR，随后读或写入 SDR，则 DCOL 清零。如果 DCOL 置位但 SPIF 没有置位，则使用 SPE 位将 SIOP 关闭，然后再开启，才使 DCOL 清零。

6. SIOP 数据寄存器(SDR，$0C)。SDR 的地址为 $000C，为发送和接收数据寄存器。SDR 不是双缓冲结构，向该寄存器写入将破坏以前的数据。只有在允许 SIOP 时(SPE＝1)，才能读或写入该寄存器。SPIF 置位时，读 SDR 的接收数据才有效。复位时，该寄存器的值不确定。

31.2 I²C 接口简介

I²C 接口是两线半双工串行接口，发送或接收 MSB 在前。SDA 为串行数据线，SCL 为串行时钟线。

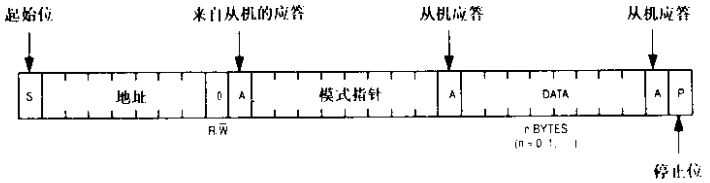


图 31-3　PCF8573 串行数据格式

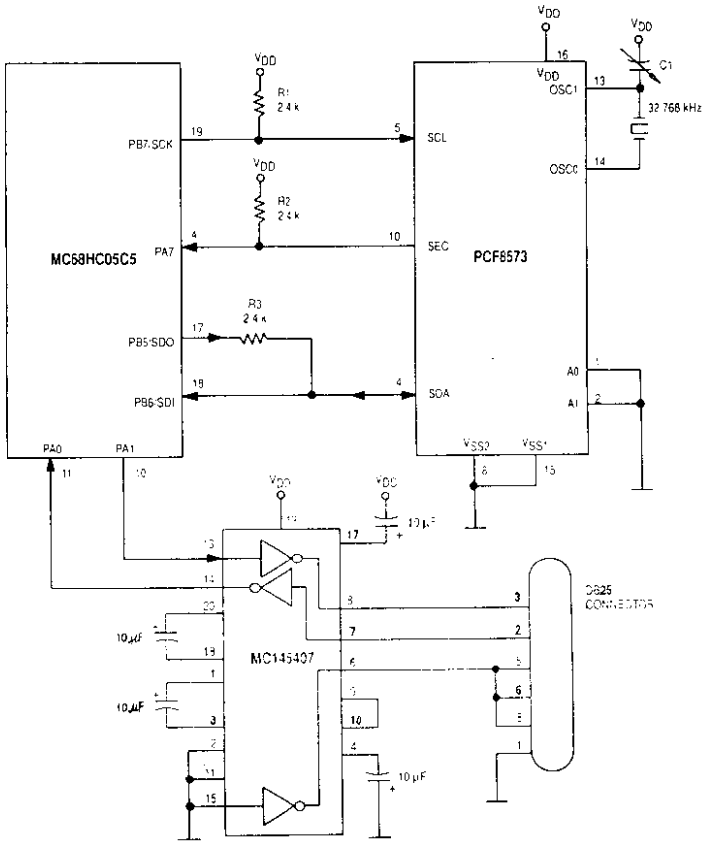


图 31-4　MC68HC05C5 与 PCF8573 的连接方法

I²C 协议包括开始条件、从机地址、n 字节数据和停止条件。起始条件定义为 SDA 引脚由高至低的跳变，而 SCL 为高(见图 31-5)；停止条件定义为 SDA 低至高的跳变，SCL 保持为高。应答信号为在第 9 个时钟周期由被寻址的器件发出的低电平。

PCF8573 的串行数据格式如图 31-3 所示。PCF8573 具有的地址为 1、1、0、1、0、A1、A0，A1 和 A0 由硬件引脚确定。地址字节的位 0 是读/写标志，如下所示：

7	6	5	4	3	2	1	0
1	1	0	1	0	A1	A0	R/$\overline{W}$
MSB							LSB

地址字节之后是模式指针，用来控制 PCF8573 内部的寄存器访问。模式指针之后的字节是读或写入时钟/定时器 PCF8573 的数据。

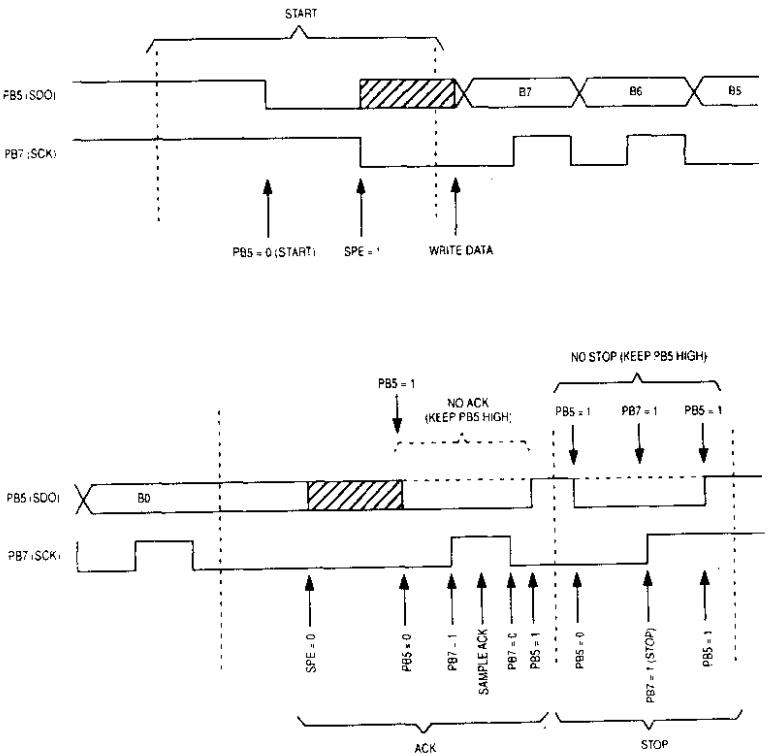


图 31-5 SIOP 产生的时序

31.3 硬件说明

MC68HC05C5 的 SIOP 用作主机，其 SPCR 的 MSTR 置 1，PB7/SCK 接至 SCL。因为 PCF8573 为双向引脚 SDA，而 SIOP 有独立的输入和输出引脚，需将 SDI 和 SDO 接在一起。因 PB 口不是漏极开路输出，SDI 和 SDO 连接时必须串联一只电阻。单片机 MC68HC05C5 与 PCF8573 的连接方法如图 31-4 所示。SEC 引脚每秒变为高，将其接至 PA7，由软件查询作为秒计数器。

当从时钟定时器 PCF8573 接收数据时，SIOP 发送 $FF，这样使得与 SDO 串联的电阻 R3 就如同上接至 V_{DD}，所以 SDO 不影响来自 SDA 引脚的数据。

31.4 软件说明

为了产生 I²C 所需的时序，需由用户控制 B 口引脚。SIOP 产生的时序如图 31-5 所示。在进行任何数据交换之前，PB5 和 PB7 初始化为高。关闭 SIOP 时（SPE＝0），PB5 将清零，而 PB7 仍为高，产生开始条件。然后允许 SIOP，并且 CPOL＝0、MSTR＝1，传输一个字节数据，发送结束后，关闭 SIOP（SPE＝0），并且 PB7 翻转为高，然后为低，产生应答时钟。如果 MCU 正发送数据，在应答时钟期间 PB5 强迫为高；否则强迫为低，以通知从机已接收到该字节。必要时将 PB5 清零，置 PB7，然后 PB5 置位产生停止条件。

为满足 PCF8573 的 100kHz 最大串行时钟的要求，MC68HC05C5 必须降低到 250kHz 的总线速度，这样其串行时钟速度为 62.5kHz。

程序流程图如图 31-6 所示。时钟数据初始化过程如下：

发送 $D0，带起始位（地址）
发送 $00，无起始位（控制）
发送小时数据（无起始位）、分钟数据（无起始位）
发送日数据，无起始位
发送月数据，无起始位
产生停止位

读时钟数据过程如下：

发送 $D0，带起始位（地址）
发送 $00，无起始位（控制）
设置发送应答位
发送 $D1，带起始位
发送 $FF 接收小时数据，发送 $FF 接收分，均无起始位

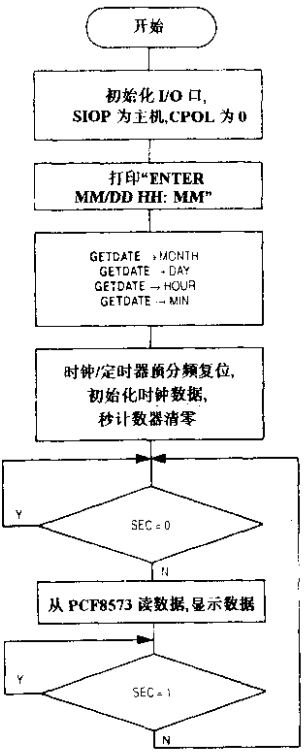


图 31-6 程序流程图

发送 $FF 接收日，发送 $FF 接收月，均无起始位
产生停止位

31.5 程序清单

```
* 该程序中主机为 MC68HC05C5，从机为 PCF8573 时钟/定时器
* 总线速度为 250kHz，MCU 以 2400 波特在终端显示时钟数据
porta       equ     $00         ;port a data register
portb       equ     $01         ;port b data register
portc       equ     $02         ;port c data register
ddra        equ     $04         ;port a data direction register
ddrb        equ     $05         ;port b data direction register
ddrc        equ     $06         ;port c data direction register
spcr        equ     $0a         ;spi control register
spsr        equ     $0b         ;spi status register
spdr        equ     $0c         ;spi data register
*
raddr       equ     $d1         ;peripheral address for read
waddr       equ     $d0         ;peripheral address for write
ram         equ     $80         ;start of ram space
*
            org     ram
sec         rmb     1           ;seconds byte
control     rmb     1           ;control byte
ack         rmb     1           ;acknowledge polarity
hour        rmb     1
min         rmb     1
month       rmb     1
day         rmb     1
savx        rmb     1
sava        rmb     1
xtemp       rmb     1
count       rmb     1
InChar      rmb     1
OutChar     rmb     1
atemp       rmb     1
cdelay      rmb     1           ;delay variable for serial routines
* start of program
            org     $0200
* all timing is based on a 500 kHz crystal
*
begin       equ     *
            lda     #%00000010
            sta     porta       ;TX pin high
            sta     ddra        ;PA1=TX pin,PA0=RX pin
            lda     #%10100000  ;pb7=pb5=output,pb6=input
```

```
                sta     portb
                sta     ddrb
                ldx     #3
                lda     delays,x
                sta     cdelay          ;2400 baud
                lda     #%00010000      ;mstr=1,cpol=0,siop still off
                sta     spcr
                sta     ack             ;ack flag non-zero,high acknowledge
                jsr     crlf
                jsr     outmsg          ;print "ENTER MM/DD HH:MM"
                jsr     getdate         ;get month
                sta     month
                jsr     inchar          ;dummy char '/'
                jsr     getdate         ;get day
                sta     day
                jsr     inchar          ;dummy space
                jsr     getdate         ;get hours
                sta     hour
                jsr     inchar          ;dummt '|  '
                jsr     getdate         ;get minutes
                sta     min
again           jsr     inchar          ;wait for <CR>
                cmp     #$0d
                bne     again
                jsr     crlf
*
* issue a reset prescaler command
*
                lda     #$20
                sta     control
                jsr     addrcntl
                jsr     stop
* initialize the clock
*
                lda     #$00
                sta     control
                jsr     addrcntl        ;send address/control bytes
                jsr     senddta         ;send 4 data bytes
*
* issue a reset prescaler command
*
                lda     #$20
                sta     control
                jsr     addrcntl
                jsr     stop
                clr     sec             ;clear seconds counter
```

```
sec_pin     brclr   7,porta,*           ;wait for SEC pin to go high
            jsr     dispdata            ;display clock data
            brset   7,porta,*           ;wait until pin goes low
*           bra     sec_pin
msg         fcc     "ENTER MM/DD HH:MM"
            fcb     $0d,$0a,$04
* 下面程序读两个 ASCII 字符并将其转换为两个 BCD 数字(位于 A)
getdate     equ     *
            jsr     inchar              ;get character
            sub     #$30                ;convert to binary
            lsla
            lsla
            lsla
            lsla                        ;make it upper nibble
            sta     sava
            jsr     inchar              ;get second ASCII char.
            sub     #$30
            add     sava                ;2 BCD digit is in Acc. A now
            rts
* 下面程序将 ACCA 的二进制字节转换为两个 BCD 数字,并作为两个 ASCII 字符显示
bin_dec     equ     *
            clrx                        ;clear number of subtraction counter
sub_more    sub     #10                 ;see how many times it is divisible
            bmi     no_tens             ;by 10
            incx                        ;increment counter
            bra     sub_more            ;subtract more tens
no_tens     add     #10                 ;restore number to positive
            lslx                        ;put 10's digit in upper nibble of X
            lslx
            lslx
            lslx
            stx     sava
            add     sava                ;merge both nibbles in Acc. A
            bsr     bcd                 ;display the 2 digits
            rts
* 下面程序将 HC05C5 的字节从 SPI 传送到外围器件,进入时数据位于 X
* w_start 是发送起始位的进入点;start 是发送没有起始位数据的进入点
w_start     equ     *
            bset    7,portb             ;take SCL line high
            bclr    5,portb             ;start condition
nostart     equ     *
            bset    6,spcr              ;enable spi,SPE=1
            stx     spdr                ;send data
wait        brclr   7,spsr,wait         ;wait for end of transmition
*
            bclr    6,spcr              ;clear SPE,disable spi
```

```
*
                tst     ack             ;test acknowledge flag
                bne     hi_ack          ;keep ack high
lo_ack          bclr    5,portb         ;else,clear ack bit
                bra     hi_ackl         ;generate ack clock
*
hi_ack          bset    5,portb         ;send high ACK bit
hi_ackl         bset    7,portb         ;take pb7 (SCL) high
                bclr    7,portb
                bset    5,portb         ;return data pin high
                rts
* 下面程序产生停止条件
stop            equ     *
                bclr    5,portb         ;bring sda low
                bset    7,portb         ;bring scl high
                bset    5,portb         ;bring sda high
                rts
* 下面程序发送两个字节,地址字节和控制字节
addrcntl        equ     *
                ldx     #waddr          ;(r/w=0)
                bsr     w_start         ;send address with start condition
                ldx     control
                bsr     nostart         ;send control byte without start
                rts
* 下面程序发送四个字节
senddta         equ     *
                ldx     hour
                bsr     nostart         ;send hours
                ldx     min
                bsr     nostart         ;send minutes
                ldx     day
                bsr     nostart         ;send days
                ldx     month
                bsr     nostart         ;send months
                bsr     stop            ;stop condition
                rts
* 输出两个 BCD 数字作为 ASCII 字符
bcd             equ     *
                sta     sava            ;save A
                jsr     outlhf          ;output left half
                lda     sava
                jsr     outrhf          ;output right half
                rts
* 以下程序读一个数据字节
read            equ     *
                lda     # $00
```

```
                sta     control
                inca
                sta     ack             ;high ack bit (ack non-zero)
                jsr     addrcntl        ;send address/control
                ldx     #raddr          ;(r/w=1)
                bsr     w_start         ;send address with start condition
                clr     ack             ;low ack bit
                ldx     #$ff            ;and read 4 data bytes
                bsr     nostart         ;keep mosi open drain (high)
                lda     spdr            ;get received data
                sta     hour            ;hours
                ldx     #$ff
                bsr     nostart
                lda     spdr
                sta     min             ;minutes
                ldx     #$ff
                jsr     nostart
                lda     spdr
                sta     day             ;days
                inc     ack             ;high ack bit for last bit received
                ldx     #$ff
                jsr     nostart
                jsr     stop            ;end session
                lda     spdr
                sta     month           ;months
                rts
```

* 以下程序使秒计数器加 1,并且每秒在屏幕显示时钟数据

```
dispdata        equ     *
                lda     #$0d
                jsr     outchar         ;send <CR>
                jsr     read            ;read 4 bytes from clock
                lda     month           ;display month
                and     #$1f
                bsr     bcd             ;output 2 BCD digit
                lda     #'/
                jsr     outchar         ;outout '/'
                lda     day             ;display day
                and     #$3f
                bsr     bcd
                lda     #$20
                jsr     outchar         ;output space
                lda     hour            ;display hours
                and     #$3f
                bsr     bcd
                lda     #':
                jsr     outchar         ;output ':'
```

```
                lda     min             ;display minutes
                and     #$7f
                jsr     bcd
                lda     #':
                jsr     outchar         ;output ':'
                lda     sec             ;display seconds
                jsr     bin_dec         ;convert seconds to BCD and display
                lda     sec             ;read seconds byte
                inca                    ;increment it
                cmp     #60
                bne     not_sixty       ;not 60 yet
                clra
not_sixty       sta     sec             ;update seconds counter
                rts
* 下面是有关显示数据的程序
outmsg          equ     *               ;print character string
                clrx
prtmsg          lda     msg,x           ;get message character
                cmp     #$04            ;EOT yet?
                beq     finish          ;yes
                jsr     outchar         ;output character
                incx                    ;increment index
                bra     prtmsg
finish          rts
crlf            equ     *               ;print new line
                lda     #$0d
                jsr     outchar
                lda     #$0a
                jsr     outchar
                rts
* 寄存器 A 和 INCHAR 接收字符类型等. 进入时屏蔽中断,退出时允许中断
inchar          equ     *               ;input character from terminal
                stx     xtemp           ;save X
                lda     #8              ;number of bits to read
                sta     count
getc4           nop                     ;unmask to allow service,then
                sei                     ;mask while looking for start bit
                brset   0,porta,*       ;wait for hilo transition
                lda     #2
                bsr     delay           ;delay 1/2 bit to middle of stait bit
                brset   0,porta,getc4   ;false start bit test
*                                       ;main loop for getc
getc7           lda     #2
                bsr     delay           ;6  common delay routine
                lda     #6
                bsr     delay           ;6
```

```
            brclr   0,porta,getc6   ;5  test input and set c-bit
getc6       ror     InChar          ;5  add this bit to the byte
            lda     count           ;3  time-wasting way to decr count
            deca                    ;3
            sta     >count          ;5  extd addr to waste extra cycle
            bne     getc7           ;3  still more bits to get
            nop                     ;2  re-enable interrupts
            lda     #2
            bsr     delay           ;3  wait out the 9th bit
            lda     #6
            bsr     delay           ;3
            lda     InChar          ;get assembled byte
            and     #%1111111       ;mask off parity bit
            bsr     putcl           ;echo it back
            lda     InChar          ;get assembled byte
            sei                     ;re-enable interrupts
            rts
outchar     equ     *               ;output character to terminal
            stx     xtemp           ;don't forget about X
putcl       equ     *               ;sneaky entry from getc to avoid clobbering x
            sta     OutChar
            sta     atemp           ;save it in both places
            lda     #9              ;going to put out
            sta     count           ;9 bits this time
            clrx                    ;for very obscure reasons
            clc                     ;this is the start bit
            sei                     ;mask interrupts while sending
            bra     putc2           ;jump in the middle of things
*           main loop for outchar
putc5       ror     OutChar         ;5  get next bit from memory
putc2       bcc     putc3           ;3  now set or clear port bit
            bset    1,porta         ;5
            bra     putc4           ;3
putc3       bclr    1,porta         ;5
            bra     putc4           ;3  equalize timing
putc4       lda     #2
            bsr     delay           ;6
            lda     #6
            bsr     delay           ;6
            dec     count           ;5
            bne     putc5           ;3  still more bits
            bset    1,porta         ;5  send stop bit
            nop                     ;2  re-enable interrupts
*
            lda     #2
            bsr     delay           ;6  delay for the stop bit
```

```
                lda     #6
                bsr     delay           ;6
                ldx     xtemp           ;3  restore X and
lda             atemp   ;3  of course A
                sei                     ;2  re-enable interrupts
                rts                     ;6
*延时程序——1/2 位时间延迟(假设外部循环为 24 个周期)
delay           equ     *
                deca
                bne     delay
                nop
                rts
*  1/2 bit delay=24 cycles overhead+(6*A)+8+8, where A=2
*  1 bit delay=24 cycles overhead+[(6*A)+8+8]+[(6*B)+8+8],A=2,B=6
*波特率计算延时
delays          fcb     32              ;300 baud
                fcb     8               ;1200 baud
                fcb     2               ;4800 baud
                fcb     1               ;9600 baud
*输出作为 ASCII 的 ACCA 的左半个字节
outlhf          equ     *
                lsra
                lsra
                lsra
                lsra
outrhf          and     #$0f
                add     #$30            ;make ASCII
                jsr     outchar         ;send character to terminal
*****           rts
                org     $1ffa
irqv            fdb     begin
swiv            fdb     begin
resetv          fdb     begin
```

32. 采用 MC6805 单片机的电话拨号技术

MC6805 系列单片机适用于脉冲或双音多频(DTMF)拨号电话机中。图 32-1 是由 MC68705P3 构成的电话拨号电路图，其中开关 S1 选择拨号类型(双音多频或脉冲)，12 个接触按键作为输入。

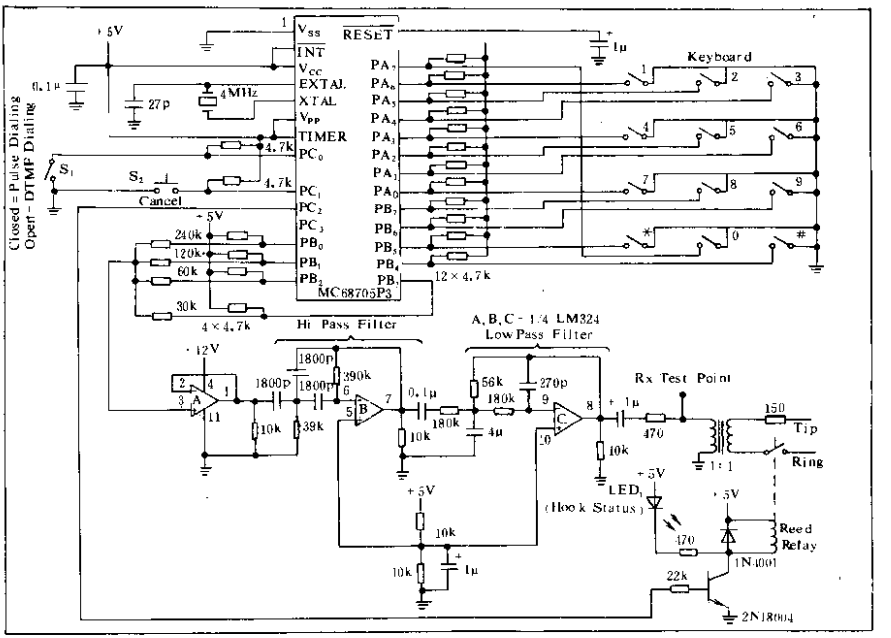


图 32-1 MC68705P3 构成的电话机拨号电路

脉冲拨号要求直接与电话线相连，这里采用 600Ω、1∶1 变压器进行接口。按键直接输入 PA7～PA0 和 PB7～PB4 中。脉冲拨号通过 PC2 端输出，驱动干簧继电器，双音多频拨号通过软件模拟双音多频数字信号，通过 PB3～PB0 输出至 4 位 D/A 转换电路，产生双音多频模拟信号至电话线；PC0、PC1 接 S1 和 S2 开关，S1 为脉冲拨号和 DTMF 拨号选择开关；S2 为取消键，LED1 灯用来指示挂勾/摘勾的状态(当用于脉冲拨号时)。此电路比较简单，既没有使用任何中断，也没有用片内定时器。

该电路的程序设计主要包括以下几部分：主程序、脉冲拨号子程序、双音多频拨号子程序、键输入子程序和延时子程序。下面分别介绍这几部分程序的设计和脉冲拨号、DTMF 脉冲的原理。

32.1 主程序设计

首先对 A、B、C 口进行初始化；然后检验是否有键按下，若无键按下，检验 PC1 位，若 PC1 为 1，则重新开始，否则，将 PC2 清 0，重新开始。若有键按下，检验 PC2 位的状态，若 PC2 为 0，则将 PC2 位置 1，重新开始；若 PC2 为 1，则进一步判断是进行脉冲拨号还是双音多频拨号，若是脉冲拨号，则应忽略键盘上的“*”和“#”号键，最后分别转到相应的子程序进行拨号处理。其主程序流程图如图 32-2 所示。

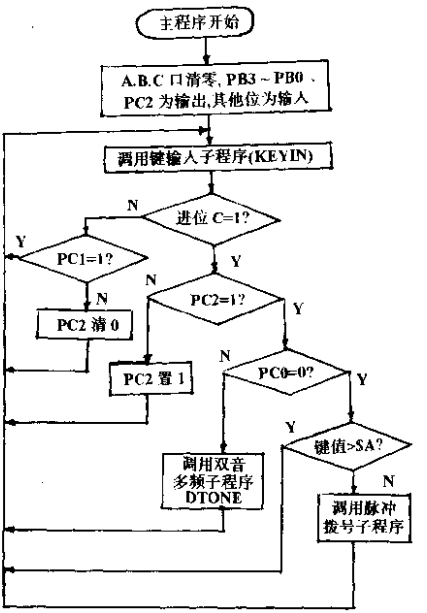


图 32-2 电话机拨号主程序流程图

32.2 脉冲拨号原理和程序设计

脉冲拨号是最为简单的一种电话拨号。根据按下的键值，由 C 口的 PC2 端发出一串脉冲。每个脉冲的负脉冲宽度为 61ms，正脉冲宽度为 39ms。因此电话线以 10pps(每秒 10 个脉冲)的速度发送信号。例如按下键盘的“3”号键，则要发出 3 个脉冲，如图 32-3 所示。注意当按下“0”号键时，将发出 10 个脉冲串。脉冲拨号的程序设计也很简单，当需要发出脉冲时，先将 PC2 端清 0，并延时 61ms，然后再将 PC2 端置 1，延时 39ms，这样就可以由 PC2 端发出一个脉冲，如果还需要发送一个脉冲，则重复以上过程。其程序流程图如图 32-4 所示。

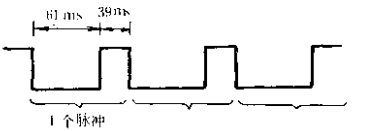


图 32-3 键值为 3 时发出的脉冲串

32.3 双音多频拨号原理及程序设计

双音多频 DTMF 拨号较为复杂。需要用软件模拟和附加的硬件。对于每一个键值，不是发出一串脉冲，而是发出两种不同频率正弦波的叠加信号。表 32-1 列出了对于不同键值所使用的两种正弦波频率(即双音频率)。

从表 32-1 中可以看出，低音频有四种频率：697、770、852、941(Hz)；高音频也有四种频率：1209、1336、1477、1633(Hz)。每一种键值分别对应于一种低音频和高音频正弦波之和。

表 32-1 键值与双音频率对照表

键 值	对应的 16 进制值	双音频率(Hz)	
0	$0	941	1336
1	$1	697	1209
2	$2	697	1336
3	$3	697	1477
4	$4	770	1209
5	$5	770	1336
6	$6	770	1477
7	$7	852	1209
8	$8	852	1336
9	$9	852	1477
A	$A	697	1633
B	$B	770	1633
C	$C	852	1633
D	$D	941	1633
*	$E	941	1209
#	$F	941	1477

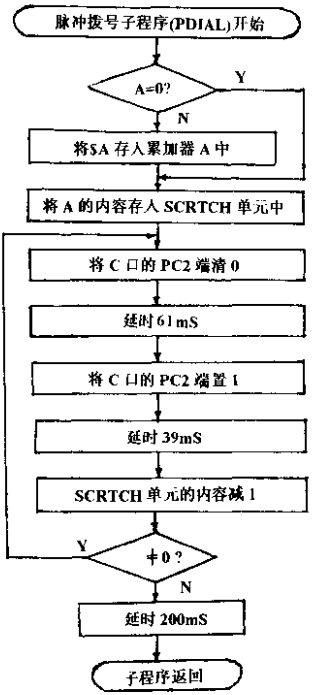


图 32-4 脉冲拨号子程序流程图

对双音多频拨号,需要用软件模拟这些不同频率的正弦波。这时可将每隔 122μs 对正弦波采样获得的采样值规格化为 0～7,并存入内存的音频发生器表中,每个采样值仅占 4 位。对于低音频正弦波,每个采样值存入音频发生器表的每个字节的高 4 位中;对于高音频正弦波,每个采样值存入每个字节的低 4 位中。然后根据每个键值所要求的低音频和高音频,将低音频和高音频正弦波的采样值相加,得到 4 位输出值,并将该值输出到 B 口的低 4 位 PB3～PB0 中,由电阻网络组成的 D/A 转换器进行 D/A 转换。4 位 D/A 转换的电阻网络中的电阻需 5%精度,由于 PB3～PB0 需较高电压输出,故要加 4 个上拉电阻。该电路会产生一些谐波,因此还需要相应高通和低通滤波器。该滤波器由 LM324 组成,其频率响应特性如图 32-5 所示。由图中可知低音频和高音频在频响曲线的中间。此外电话线的输出电平还可以通过 470Ω 电阻来调节。在 Rx 点上还可以接收音频信号,所以还可以用于双路通信中。

下面给出当按下键"2"时,软件模拟的两种正弦波波形和输出到电话线的输出波形。当键值为 2 时,由表 32-1 可知,它的输出波形由 697Hz 和 1336Hz 的正弦波叠加而成。对 697Hz 的正弦波,由音频发生器表 TN697 的高 4 位,可查出采样值为:7、7、5、3、2、0、0、1、2、4、6、7、7、6、5、3、1、0、0、1、2、4、6、7、7、6、4、3、1、0、0、1、3、5、6。

由这些采样值形成的正弦波如图 32-6(a)所示。对 1336Hz 的正弦波,由音频发生器表 TN1336 的低 4 位,可查出采样值为:

7、5、2、0、1、5、7、6、2、0、1、4、7、6、3、0、1、4、7、6、3、0、0、3、6、7、4、1、0、3、6、7、4、1、0、3、6。

由这些采样值形成的正弦波如图 32-6(b)所示。两种叠加后的波形如图 32-6(c)所示。理想状

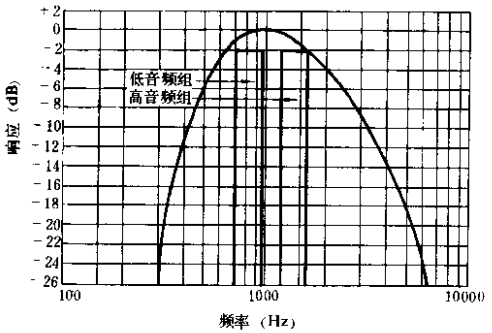


图 32-5　带通滤波器的频响特性

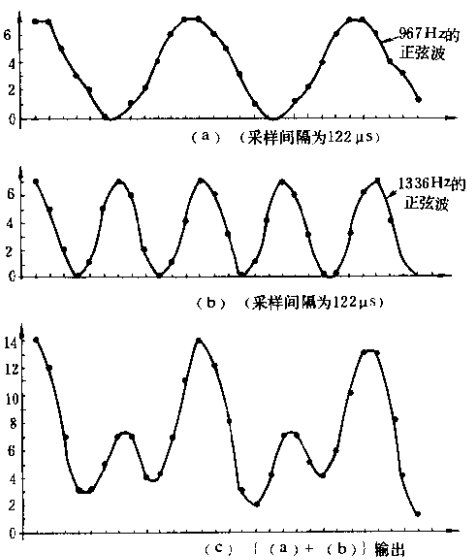


图 32-6　双音多频拨号时的两种正弦波和输出波形
（理想状态下，键值为 2）

态下，该波形就是双音多频拨号的输出波形。

在设计双音多频拨号程序时，需要设置两个表，一个是双音频地址查询表 TTAB，它存放着每个键值对应的低音频和高音频采样值的起始地址；另一个是音频发生器表（该表存放在以 $80 开始的 0 页存储区中，以便于寻址），在该表中，低音频的采样值存在每个字节的高 4 位中，高音频的采样值存在每个字节的低 4 位中，每种频率的采样值均以 $F 结束。根据键值从双音频地址查询表中查出存放两种正弦波采样值的起始地址，由此可取出两种正弦波的 3 位采样值，然后相加，得到对应的输出值，将该输出现送入 PB3～PB0 中。将逐点采样值进行相同的计算，顺序送入 PB3～PB0 中。若取出的采样值为 $F，则重

新回到起始地址处，循环重新开始，这样可产生一个连续的输出波形。对应于每一个键值，将产生 65ms 的双音多频输出信号和 25ms 的静音，最后返回主程序，等待下一个按键。编程时设置了几个变量：STARTH 和 STARTL 分别存放高音频和低音频的起始地址；TONEH 和 TONEL 分别存放高音频和低音频的当前地址；TIMEH 和 TIMEL 用于存放循环产生输出值的计数器的高 8 位和低 8 位。由于产生某一点输出值需要 122μs，因此要产生 65ms 的输出信号，需要进行 531 次循环。将 TIMEH 和 TIMEL 分别置初值 $FD 和 $EC，先使 TIMEL 加 1，直到 TIMEL 为 0 时，将 TIMEH 加 1，然后再重新开始使 TIMEL 加 1，加到 TIMEL 为 0 后，再使 TIMEH 加 1，直到 TIMEL 和 TIMEH 都为 0 为止，这样共产生了 531(19+256+256)次循环，持续时间为 64.782(531×122μs)ms，约为 65ms。其双音多频拨号程序流程图如图 32-7 所示。

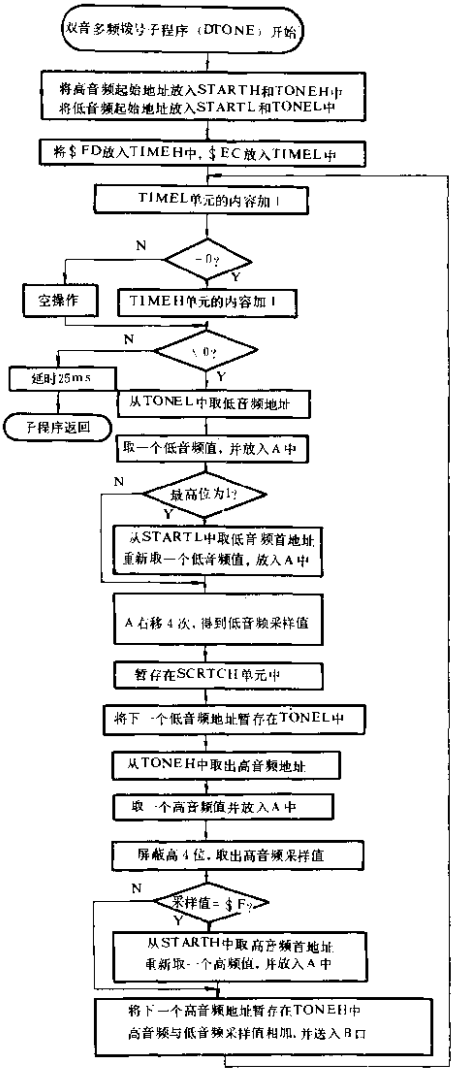


图 32-7　双音多频程序流程图

32.4　键输入子程序 KEYIN

该子程序对键盘进行二遍扫描，判断是否有键按下，若有键按下，则进位为 1，键值在累加器 A 中。在这个子程序中，又调用了一个键盘扫描子程序 CHECK，它用于扫描键盘。首先判断 A 口各位对应的键是否按下，再判断 B 口的 PB7～PB4 位对应的键是否按下，若有键按下，则进位 C 置 1，键值在累加器 A 中。图 32-8(b)为 CHECK 子程序流程图。在键输入子程序中，使用了一个工作变量 LSYKEY。如果两次扫描均无键按下，则将 $A

存入LSTKEY中；若第一次扫描后无键按下，并且LSTKEY中已为$A，则不进行第二次键盘扫描；若键盘扫描后有键按下，且键值与LSTKEY中的相同，则也认为是无键按下；否则将新键值存入LSTKEY单元中。两次键扫描间隔为25ms。键输入子程序KEYIN程序流程图如图32-8(a)所示。

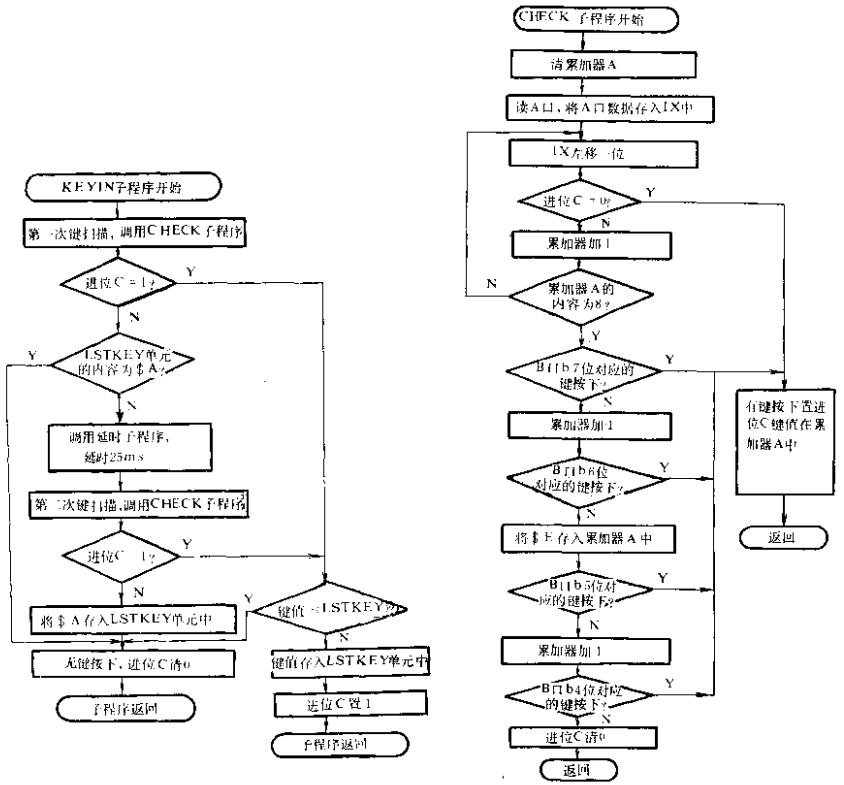


图32-8 键输入和扫描子程序流程图

(a)键盘输入子程序KEYIN流程图；(b)键盘扫描子程序CHECK流程图

32.5 延时子程序WAITMS

该子程序根据累加器A的内容，来确定延时多少毫秒。程序流程图如图32-9所示。

32.6 程序清单

```
x       设置掩膜选择寄存器
        ORG  $ 784
```

```
        FCB %01011111          选择晶振为系统时钟,定时器/预分频器时钟为内部时钟,预
                               分频器分频因子为128
*       端口定义
PORTA   EQU  $0                A口数据寄存器
PORTB   EQU  $1                B口数据寄存器
PORTC   EQU  $2                C口数据寄存器
PORTAD  EQU  $4                A口数据方向寄存器
PORTBD  EQU  $5                B口数据方向寄存器
PORTCD  EQU  $6                C口数据方向寄存器
TMRDAT  EQU  $8                定时器数据寄存器
TMRCON  EQU  $9                定时器控制寄存器
*       RAM存贮器分配
        ORG $40
TIMEH   RMB 1                  计数器
TIMEL   RMB 1                  计数器
STARTH  RMB 1                  高音频起始地址
STARTL  RMB 1                  低音频起始地址
TONEH   RMB 1                  高音频当前地址
TONEL   RMB 1                  低音频当前地址
SCRTCH  RMB 1                  暂存单元
LSTKEY  RMB 1                  暂存单元
*       主ROM存贮区
        ORG $100
*       冷启动,端口初始化
START   CLR PORTA              清A口
        CLR PORTB              清B口
        CLR PORTC              清C口
        LDA #%00001111         B口的低4位为输出,高4位为输入
        STA PORTBD
        LDA #%00000100         C口的PC2为输出,其他位为输入
        STA PORTCD
*       检查是否有键按下
SCAN    JSR  KEYIN             调用键输入子程序,若键输入,进位C=1,键值在A中
        BCS  DIAL              进位C=1,转DIAL
*       无键按下,检查"ON-HOOK"开关
        BRSET 1 PORTC,SCAN     若口的PC1位为1,转SCAN
        BCLR 2 PORTC           否则,C口的PC2位清0
        BRA SCAN               转移到SCAN
*       拨号开始
*       若前面是"ON-HOOK"状态,则接下是"OFF-HOOK"状态,并且等待键按下
DIAL    BRSET  2,PORTC,DODIAL  若PC2为1,转DODIAL
        BSET   2,PORTC
        BRA    SCAN
*是脉冲拨号还是DTMF拨号
```

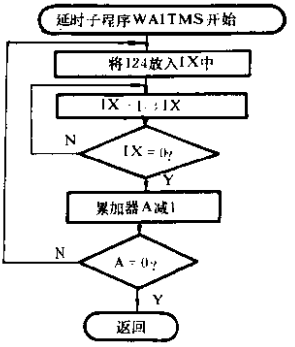


图 32-9　延时子程序 WAITMS 流程图

```
DODIAL    BRCLR   0,PORTC,PULSE   若 PC0 为 0,转脉冲拨号
          JSR     DTONE           否则转 DTMF 拨号
          BRA     SCAN
*脉冲拨号,"#"和"*"不予识别
PULSE     CMP     #$A
          BHI                     若大于,转 SCAN
          JSR     PDIAL           否则转脉冲拨号子程序
          BRA     SCAN
*WAITMS 延时子程序,根据累加器 A 中的内容确定延时多少毫秒
WAITMS    LDX     #124            IX 寄存器等于 124
WAIT1     DECX                    IX 寄存器减 1
          BNE     WAIT1           IX 不等于 0,转 WAIT1
          DECA                    等于 0,累加器 A 减 1
          BNE     WAITMS          累加器 A 不等于 0,转 WAITMS
          RTS                     等于 0,返回
*DTONE 子程序
*双音多频拨号子程序,产生 65ms 的双音多频信号,然后产生 25ms 的静音。
*双音多频拨号的键值存在累加器 A 中
DTONE     AND     #$F             屏蔽高 4 位
          TAX                     将 A 的值送入 IX 中
          LSLX                    IX 左移一位(IX 乘以 2)
          LDA     TTAB,X          取出高音频起始地址
          STA     STARTH          存入 STARTH 中
          STA     TONEH           存入 TONEH 中
          LDA     TTAB+1,X        取出低音频起始地址
          STA     STARTL          存入 STARTL 中
          STA     TONEL           存入 TONEL 中
*设置音频计数器的初值
          LDA     #$FD
          STA     TIMEH           TIMEH 单元为 $FD
          LDA     #$EC
          STA     TIMEL           TIMEL 单元为 $EC
*音频发生循环程序,每一次循环需要 122μs
*根据计数器的值,共循环 19+256+256=531 次,大约产生 65ms 的音频信号
TONELP    INC     TIMEL           TIMEL 单元的内容加 1
          BNE     NCARRY          不等于 0,转 NCARRY
          INC     TIMEH           TIMEH 单元的内容加 1
CONT      BNE     GETLO           不等于 0,转 GETLO
*产生 25ms 的静音
          LDA     #25
          BRA     WAITMS          转延时程序 WAITMS 并返回主程序
NCARRY    NOP
          BRA     CONT            转移到 CONT
*得低音频值
GETLO     LDX     TONEL           取出低音频当前地址
          LDA     0,X             取出低音频值
```

```
                BMI     GETNLO          若采样值为$F,转GETNLO
                NOP                     否则,执行2个空操作,转SHFLO
                NOP
                BRA     SHFLO
* 到低音频表的最后,复位到低音频开始
GETNLO          LDA     STARL           取出低音频起始地址
                LDA     0,X             取出低音频值
SHFLO           LSRA                    右移4次,得到低频正弦波采样值
                LSRA
                LSRA
                LSRA
* 暂存低音采样值
                STA     SCRTCH
* 暂存下一个低音频地址值
                INCX
                STX     TONEL
                LDX     TONEH           取高音频当前地址
                LDA     0,X             取高音频值
* 取出高频正弦波的采样值,该值在高音频值的低4位中
                AND     #$F             屏蔽高4位,得到采样值
                CMP     #$F             采样值与$F比较
                BEQ     GENHI           若相等,转GENHI
                NOP
                NOP
                BRA     ADDNUM          转ADDNUM
* 到高音频的最后,复位到高音频的开始
GENHI           LDX     STARTH          取高音频起始地址
                LDA     0,X             取高音频第一个值
* 暂存高音频当前地址
ADDNUM          INCX
                STX     TONEH
* 高频与低频正弦波采样值相加
                ADD     SCRTCH
* 将相加的结果存入B口
                STA     PORTB
*
                NOP
                NOP
                NOP
                BRA     TONELP          重复以上过程,转TONELP
* 双音频地址查询表
TTAB            FCB     TN1336  (0)
                FCB     TN941
                FCB     TN1209  (1)
                FCB     TN697
                FCB     TN1336  (2)
```

```
        FCB     TN697
        FCB     TN1477          (3)
        FCB     TN697
        FCB     TN1209          (4)
        FCB     TN770
        FCB     TN1336          (5)
        FCB     TN770
        FCB     TN1477          (6)
        FCB     TN770
        FCB     TN1209          (7)
        FCB     TN852
        FCB     TN1336          (8)
        FCB     TN852
        FCB     TN1477          (9)
        FCB     TN852
        FCB     TN1633          (A)
        FCB     TN697
        FCB     TN1633          (B)
        FCB     TN770
        FCB     TN1633          (C)
        FCB     TN852
        FCB     TN1633          (D)
        FCB     TN941
        FCB     TN1209          (*)
        FCB     TN941
        FCB     TN1477          (#)
        FCB     TN941
* KEYIN 子程序,判断是否有键按下。若有键按下,则进位 C 为 1。键盘值在累加器 A 中
KEYIN   BSR     CHECK           调用 CHECK 子程序
        BCS     KEY11           进位 C 为 1 吗? 为 1,有键按下,转 KEY11
* 第一次扫描无键按下,
* 检查 LSTKEY 单元的内容,若为 $A,则不进行第二次键扫描
        LDA     LSTKEY          累加器 A 中←(LSTKEY)
        CMP     #$A             A 中的内容为 $A 吗?
        BEQ     KEY22           是,转 KEY22,子程序退出。不是,进行第二次键扫描
* 延时 25ms 后,进行第二次键扫描
        LDA     #25             累加器 A=25
        JSR     WAITMS          调用延时子程序
        BSR     CHECK           调用 CHECK 子程序,进行第二次扫描
        BCS     KEY11           进位 C 为 1,有键按下,转 KEY11
* 第二次键扫描仍无键按下
        LDA     #$A             将 $A 装入累加器 A 中
        STA     LSTKEY          A 的内容存入 LSTKEY 单元中
KEY22   CLC                     进位 C 清 0
        RTS                     子程序返回
* 有键按下,判别是否是刚按下的键
```

```
KEY11   CMP     LSTKEY      A 与 LSTKEY 单元的内容比较
        BEQ     KEY22       相等,转 KEY22
        STA     LSTKEY      不相等,存新键值到 LSTKEY 单元中
        BEC                 进位 C 置 1
        RTS                 子程序返回
*CHECK 子程序
*扫描键盘,判别是否有键按下,若有键按下,进位 C 为 1,键值在累加器 A 中
CHECK   CLRA                清累加器 A
        LDX     PORTA       读 A 口的数据,并存入 IX 寄存器中
KEY1    LSLX                IX 的内容左移一位
        BCC     GOTKEY      进位 C=0,转 GOTKEY
        INCA                累加器加 1
        CMP     #8          累加器 A 的内容与 8 比较
        BNE     KEY1        不相等,转 KEY1
*A 口对应的键均没按下,判断 B 口对应的键
        BRCLR   7,PORTB,GOTKEY   是键 8 按下吗? 是,转 GOTKEY
        INCA                不是,累加器加 1
        BRCLR   6,PORTB,GOTKEY   是键 9 按下吗? 是,转 GOTKEY
        LDA     #$E         将 $E 放入累加器 A 中
        BRCLR   5,PORTB,GOTKEY   是"*"键按下吗? 是,转 GOTKEY
        INCA                累加器加 1
        BRCLR   4,PORTB,GOTKEY   是"#"键按下吗? 是,转 GOTKEY
*没有键按下
        CLC                 进位 C 清 0
        RTS                 子程序返回
*
GOTKEY  SEC                 有键按下,置位进位 C
        RTS                 键值在累加器 A 中,返回
***PDIAL 子程序***
PDIAL   TSTA                测试 A 的内容
        BNE     PDIAL1      若不等于 0,转 PDIAL1
*若键值为 0,则发送 10 个脉冲
        LDA     #$A         将 $0A 放入累加器 A 中
PDIAL1  STA     SCRTCH      将 A 的内容存入 SCRTCH 单元中
*产生一串脉冲,脉冲数由 SCRTCH 单元的内容决定
PDIAL2  BCLR    2,PORTC     将 C 口 PC2 端清 0
        LDA     #61         延时 61ms
        JSR     WAITMS
        BSET    2,PORTC     将 C 口 PC2 端置 1,
        LDA     #39         延时 39ms
        JSR     WAITMS
        DEC     SCRTCH      SCRTCH 单元的内容减 1
        BNE     PDIAL2      不等于 0,转 PDIAL2
*脉冲拨号间隔为 200ms
        LDA     #200        延时 200ms
        JMP     WAITMS      转移到 WAITMS 子程序
```

	ORG	$80
TN697	FCB	$77
	FCB	$76
	FCB	$53
	FCB	$30
	FCB	$21
	FCB	$03
	FCB	$06
	FCB	$17
	FCB	$25
	FCB	$42
	FCB	$60
	FCB	$71
	FCB	$74
	FCB	$67
	FCB	$57
	FCB	$34
	FCB	$11
	FCB	$00
	FCB	$02
	FCB	$15
	FCB	$27
	FCB	$46
	FCB	$64
	FCB	$71
	FCB	$70
	FCB	$62
	FCB	$45
	FCB	$3F
TN1336	FCB	$17
	FCB	$05
	FCB	$02
	FCB	$10
	FCB	$31
	FCB	$55
	FCB	$67
	FCB	$F6
TN770	FCB	$72
	FCB	$60
	FCB	$51
	FCB	$34
	FCB	$17
	FCB	$06
	FCB	$03
	FCB	$20
	FCB	$41
	FCB	$54
	FCB	$77
	FCB	$76
	FCB	$63
	FCB	$40
	FCB	$20
	FCB	$13
	FCB	$06
	FCB	$17
	FCB	$24
	FCB	$41
	FCB	$60
	FCB	$73
	FCB	$76
	FCB	$57
	FCB	$34
	FCB	$11
	FCB	$00
	FCB	$03
	FCB	$16
	FCB	$3F
TN1477	FCB	$57
	FCB	$65
	FCB	$F1
TN852	FCB	$70
	FCB	$63
	FCB	$46
	FCB	$27
	FCB	$03
	FCB	$00
	FCB	$11
	FCB	$35
	FCB	$57
	FCB	$75
	FCB	$72
	FCB	$60
	FCB	$42
	FCB	$16
	FCB	$07
	FCB	$04
	FCB	$20
	FCB	$41
	FCB	$64
	FCB	$7F

```
TN1633  FCB  $77
        FCB  $55
        FCB  $32
        FCB  $11
        FCB  $05
        FCB  $17
        FCB  $25
        FCB  $51
        FCB  $61
        FCB  $F4
TN941   FCB  $77
        FCB  $65
        FCB  $41
        FCB  $21
        FCB  $04
        FCB  $07
        FCB  $25
        FCB  $51
        FCB  $70
        FCB  $74
        FCB  $67
        FCB  $35
        FCB  $11
        FCB  $00
        FCB  $14
        FCB  $37
        FCB  $55
        FCB  $71
        FCB  $70
        FCB  $54
        FCB  $37
        FCB  $15
        FCB  $01
        FCB  $10
        FCB  $44
        FCB  $6F
        FCB  $FF
TN0FF   FCB  $00
        FCB  $FF
TN1209  EQU  TN697
        ORG  $7F8
        FDB START 定时器中断向量
        FDB START INT 中断向量
        FDB START SWI 软件中断向量
        FDB START 复位向量

        END
```

33. M68HC11 单片机与 M6805 单片机利用 SPI 进行同步串行通讯

MOTOROLA 公司的单片机最显著的特点之一是具有一个串行外围接口(SPI),它主要用于同步串行通讯中。利用片内 SPI 进行同步串行通讯,不仅不需要外围接口芯片,而且软件所占内存容量最少。MOTOROLA 单片机的 SPI 系统有两种基本形式:第一种是高性能的 SPI,如 M68HC11 MCU 和 MC68HC05C4 MCU 所具有的 SPI。一般只有 HCMOS 系列的单片机才具有这种形式的 SPI。另一种是简单的 SPI,一般 HMOS 系列的 MCU 是有这种形式的 SPI。SPI 不仅具有同步串行通讯功能,还具有其他功能。下面介绍在第一种形式的 SPI 和第二种形式的 SPI 之间进行同步通讯的例子,并且论述两种 SPI 之间的一些细微差别和使用两种 SPI 的软件编程方法。

33.1 SPI 的硬件连接与设置

图 33-1 是高性能的 M68HC11 MCU 和廉价的 MC6805L3 通过 SPI 进行同步串行通讯的电路连接图。两个 MCU 之间靠一根双向数据线和一根时钟线进行同步通讯。M68HC11 MCU 作为主机,由它提供同步时钟 SCK;MC6805L3 为从机,它不需要提供任何定时信号。此外两个 MCU 的$\overline{SS}$端都设置为输出,以防止 SPI 发生错误。两个 MCU 之间不需要硬件应答逻辑,而靠软件进行应答,因此 MCU 之间的连线最少。M68HC11 的应答信号由禁止和允许 SPI 自动产生,而 MC6805L3 的应答信号由 PD_3 管脚提供。MCU 的其他数据管脚可用于其他功能。

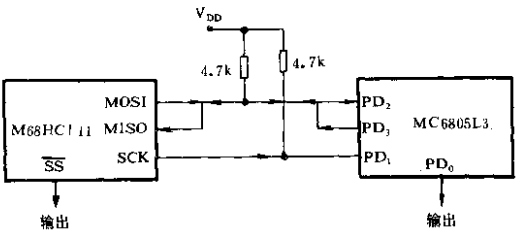


图 33-1 利用 SPI 串行通讯电路连接图

软件应答时,需注意以下两点:一是 MC6805L3 的接收数据寄存器未经缓冲,因此新数据只能在前一数据读走后才能到来;二是 MC6805L3 不可能由禁止时钟信号停止 M68HC11 MCU 传输数据。时钟信号由 M68HC11 的 SPI 提供,因此若 M68HC11 的 SPI 企图延缓时钟或终止时钟的传送,将会造成数据丢失。

33.2 数据传输和时序

图 33-2 是 SPI 应答和数据传输时序图。两个 MCU 的应答信号完全由软件产生,而 8 位数据和时钟由 SPI 硬件产生。

为了避免数据竞争,无论是在数据传输还是在应答过程中,两个 SPI 必须工作在线或模式(开路模式)。为了工作在开路模式,MC6805L3 的杂散寄存器的 PB3 位应置 1,而 M68HC11

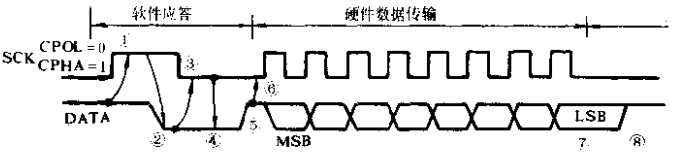


图 33-2 SPI 应答和数据传输时序图

* M68HC11 时钟控制位
①由主机禁止 SPI，使 SCK 为高
②由从机的 PD3 位将数据线清 0
③主机允许 SPI，使 SCK 为低
④只要时钟线变低，从机就将数据存入 SPI 数据寄存器中，并将对应的数据方向控制位置 1，并且允许 SPI
⑤从机使数据线变高
⑥主机将数据装入 SPI 数据寄存器中，开始发出 SCK 时钟
⑦主机和从机检测数据传输的末尾，从 SPI 寄存器中读出数据
⑧从机禁止 SPI，保证数据线变高

的 SPI 控制寄存器的 PB5 位需置 1。注意，M68HC05 系列 MCU 中的 SPI 不支持这种开路模式，因此不能应用在这种情况下。

SCK 时钟设置空闲为低。数据在上升沿输出，下降沿采样，两个 SPI 都必须设置成以相同的时钟形式工作，8 位数据传输的最大时钟速率为 125kHz，这个速率受限于 MC6805L3 的 SPI。数据的传输是在应答过程之后，这样可保证两个 MCU 都处于正确状态下，并准备好进行数据传输。

数据的最高位最先出现在第一个时钟的上升沿，在时钟的下降沿，数据被锁存在两个 SPI 的移位寄存器中。一旦最后一位数据被锁存，数据线就释放为高电平，以保证下一个应答过程正确工作。对 M68HC11 作为发送器的情况，数据传送完后，数据线自动产生高电平，这是 SPI 硬件结构决定的。但是 MC6805L3 发送完数据后，LSB 位就永远保持在数据线上，因此应由软件程序将数据线恢复为高电平。

33.3 软件程序设计

每个 MCU 的发送和接收程序基本相同。每个 MCU 在发送和接收时入口和出口情况如表 33-1 所示。入口和出口的数据均在累加器 A 中。对 MC6805L3 MCU，IX 寄存器中的内容表明传输过程的操作模式，因为同一个 I/O 管脚用来发送和接收数据，因此其数据方向必须靠 IX 中的值适当地改变。而对 M68HC11 MCU，它各用一个管脚接收和发送数据，因此它们的 DDR 管脚只需在程序开始时建立一次。

表 33-1 两种 MCU 入口和出口时的情况

M68HC11

	发送时 A 的内容	接收时 A 的内容	IX 寄存器的内容
入口	将要发送的数据	$FF	寄存器区首地址
出口	发送的数据	接收的数据	不变

MC6805L3

	发送时		接收时	
	A 的内容	IX 寄存器的内容	A 的内容	IX 寄存器的内容
入口	将要发送的数据	$5	无关	$1
出口	发送的数据	$D	接收的数据	$D

对于主机发送从机接收的情况，由于时钟由主机发出，当应答过程结束后，M68HC11 MCU 只要将需发送的数据装入 SPI 数据寄存器，就启动一次数据传送过程，即 M68HC11 的

SPI 开始发送数据和时钟，MC6805L3 开始接收数据。当主机接收从机发送时，MC6805L3 的 SPI 数据寄存器准备好要发送的数据，但数据传送的开始仍由 M68HC11 的 SPI 控制，当 M68HC11 MCU 将 $FF 装入 SPI 数据寄存器时，启动一次数据传送过程，即 M68HC11 的 SPI 开始接收数据，同时发送同步时钟 SCK，MC6805L3 开始在同步时钟的控制下发送数据。$FF 装入 M68HC11 的 SPI 数据寄存器是为了避免数据总线竞争。因此，M68HC11 作为主控时钟，不仅在发送数据时提供时钟信号，在接收来自于MC6805L3的数据时，也提供时钟信号。

标志位(SPIF)表明数据传输是否完成。对于 MC6805L3，该位是 SPI 控制寄存器的 PB7 位，而对 MC68HC11，该位是 SPI 状态寄存器的 PB7 位，该位置 1 表明数据发送或接收已完成。M68HC11 和 MC6805L3 的数据传输完成标志位的清零方式不同；对 MC6805L3 只要将 SPIF 写 0 即可将该标志位清 0；对 M68HC11，将 SPIF 标志位清，需要两步操作，在 SPIF 为 1 时读 SPI 状态寄存器，然后访问 SPI 数据寄存器可将 SPIF 位清 0。

对 MC6805L3，当检测到时钟线上为低电平时(即仅当 M68HC11 控制 SPI 时)，初始化程序才继续向下执行。在这之前，所有 I/O 端均被置为输入端，因此时钟线上要接上拉电阻。

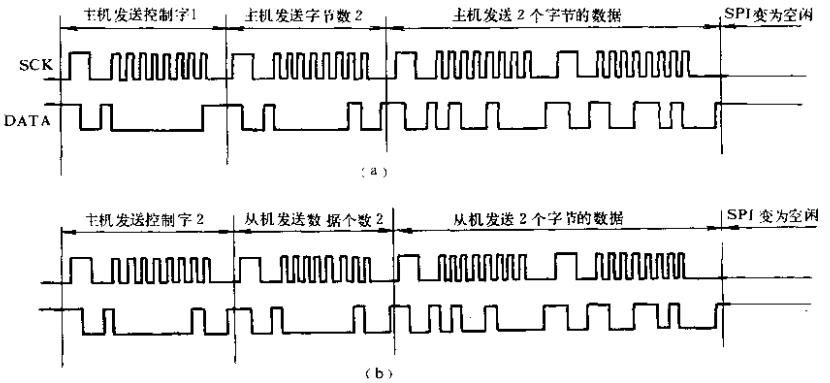


图 33-3 SPI 数据传输时序图

(a)主机发送从机接收时序图； (b)主机接收从机发送时序图

在编程之前，还要规定传输规程。由于主机控制数据传输方向和数据流长度，因此在传输数据之前主机(即 M68HC11)应发送一个控制字节，它指明是主机接收从机发送还是主机发送从机接收；然后主机或从机发送数据个数，最后是数据。一旦传送完最后字节，MC6805L3 等待新的控制字节或去执行其他任务，例如处理以前接收的数据。同样 M68HC11 可以发送新的控制字节或执行其他的任务。例如当主机发送控制字节 $01 时，为主机发送从机接收，以后发送数据个数 2，再发送两个字节的数据，如图 33-3(a)所示。当主机发送控制字节 $02 时，为主机接收，从机发送，以后主机接收数据个数 2，再接收 2 个字节的数据，如图 33-3(b)所示。以上这种数据传输方式，硬件连线最少，可用于远程数据通讯中，例如机车管理系统、仪表和数据记录设备等。

M68HC11 SPI 主程序流程图如图 33-4(a)所示、SPI 发送子程序和接收子程序以及 SPI 数据传输子程序流程图分别如图 33-4(b)、(c)、(d)所示。

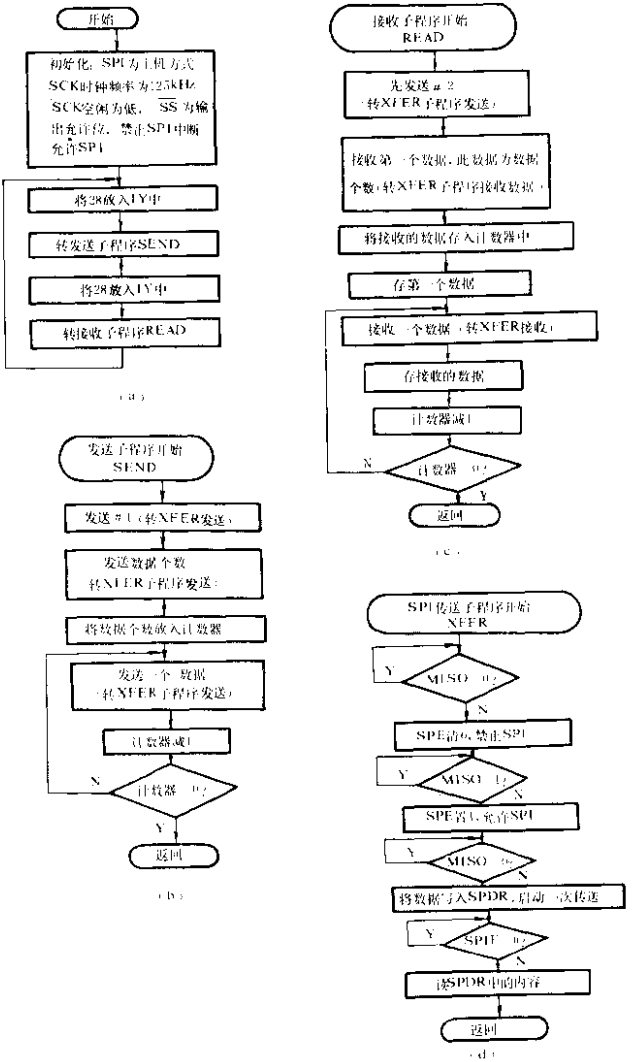


图 33-4　M68HC11 SPI 数据传输流程图

(a)M68HC11 SPI 主程序流程图；(b)SPI 发送子程序流程图

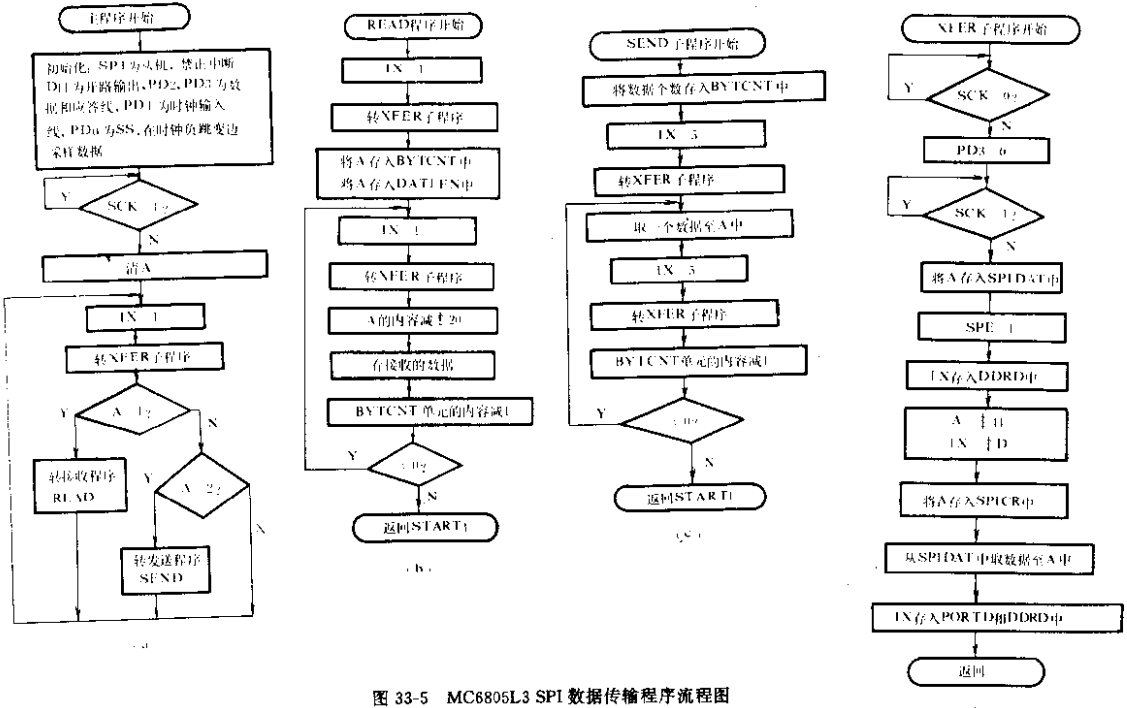
(c)SPI 接收子程序流程图；(d)SPI 传输子程序流程图

M68HC11 源程序：

```
PORTD       EQU     8               D 口数据寄存器
DDRD        EQU     9               D 口数据方向寄存器
SPCR        EQU     $28             SPI 控制寄存器
SPSR        EQU     $29             SPI 状态寄存器
SPDR        EQU     $2A             SPI 数据寄存器
TMSK2       EQU     $24             屏蔽寄存器 2
TFLG2       EQU     $25
*
MISO        EQU     4               主机输入，从机输出
MOSI        EQU     8               主机输出，从机输入
SCK         EQU     $10             SPI 时钟
SPIF        EQU     $80             SPI 传送完成标志
SPE         EQU     $40             SPI 允许标志
            ORG     $0
TIMCOUNT    RMB     1
MSG1        FCB     28              等于 $1C
    FCC  /BI—DIRECTIDNAL DATA TRANSFER/
*ORG                $C000
START       LDS     #S35            建立堆栈指针
            BSR     INIT            转移到 INIT
START1      LDY     #MSG1           将 $1C 放入 IY 中
            BSR     SEND            转发送子程序 SEND
            LDY     #MSG1           将 $1C 放入 IY 中，
            BSR     READ            转接收子程序 READ
            BRA     START1          重新开始
*发送子程序
SEND        EQU     *
            LDAA    #1              发送 1，A=#1
            BSR     XFER            转 SPI 传送子程序 XFER
            LDAA    ,Y              取发送数据个数
            TAB                     存入 B 中，B 作为计数器
            BSR     XFER            转 SPI 传送子程序 XFER
SEND1       INY                     指针 IY 加 1
            LDAA    ,Y              取发送数据
            BSR     XFER            转 SPI 传送子程序
            DECB                    B 减 1
            BNE     SEND1           不等于 0，转 SEND1，
            RTS                     否则返回
*接收子程序
READ        EQU     *
            LDAA    #2              SPI 发送#2
            BSR     XFER            转 SPI 传送子程序 XFER
            LDAA    #$FF            A=$FF
            BSR     XFER            转 SPI 传送子程序 XFER，将接收的数据存入 A 中
            TAB                     将 A 放入 B 中，B 作为计数器
```

```
        STAA    ,Y              存接收的第一个数据
READ1   INY                     指针 IY 加 1
        LDAA    #$FF            发送 $FF
        BSR     XFER            转 SPI 传送子程序 XFER
        STAA    ,Y              存接收的数据
        DECB                    B 减 1
        BNE     READ1           B 不等于 0,转 READ1
        RTS                     等于 0,返回
* 初始化程序
INIT    EQU     *
        LDX     #$1000          将寄存器区首地址放入 IX 中
        LDAA    #$38            将 $38 放入累加器 A 中
        STAA    DDRD,X          使SS、SCK、MOSI 为输出、MISO 为输入
        LDAA    #$76            将 $76 放入累加器 A 中
        STAA    SPCR,X          SPI 为主机方式,SCK 时钟为 125kHz,SCK 空闲为
                                低,禁止 SPI 中断,允许 SPI,SS为输出允许位
        LDAA    #$18
        STAA    PORTD,X         SS、SCK 输出初始状态为高,MOSI 为低。
        RTS                     子程序返回
        ********************************
        *                              *
        *                              *
        *                              *
        *   双向数据传送子程序 XFER    *
        *   需要发送的数据在累加器 A 中 *
        *   接收到的数据最后也存入累加器 A 中 *
        *   由 M68HC11 发送同步时钟信号 *
        *                              *
        *                              *
        *                              *
        ********************************
XFER    EQU     *
        BRCLR   PORTD,X,#MISO,*     MISO 位为 0 吗? 为 0,等待
        BCLR    SPCR,X,#SPE         为 1,将 SPE 清 0,禁止 SPI
        BRSET   PORTD,X,#MISO,*     MISO 位为 1 吗? 为 1,等待
        BSET    SPCR,X,#SPE         为 0,将 SPE 置 1,允许 SPI
        BRCLR   PORTD,X,#MISO,*     MISO 位为 0 码? 为 0,等待
        STAA    SPDR,X              为 1,SPI 传送开始
        BRCLR   SPSR,X,#SPIF,*      传送完成了吗? 没有,等待
        LDAA    SPDR,X              完成,将接收的数据读入 A 中
        RTS                         返回
* 复位表
        ORG     $FFFE
        FDB     START               $FFFE、$FFFF 为起始地址
        END
```

MC6805L3 SPI 主程序流程图如图 33-5(a)所示,SPI 接收和发送以及 SPI 传输子程序流程图分别如图 33-5(b)、(c)、(d)所示。



(a) (b) (c) (d)

图 33-5　MC6805L3 SPI 数据传输程序流程图

(a)MC6805L3 SPI 主程序流程图；(b)SPI 接收子程序流程图；

(c)SPI 发送子程序流程图；(d)SPI 传输子程序流程图

```
*************************
*                       *
*   MC6805L SPI 传输程序  *
*                       *
*   在 M68HC11 主机与 MC6805L3 从机之间进行 SPI 传输  *
*                       *
*                       *
*************************

PORTA   EQU   0          数据寄存器 A
PROTD   EQU   3          数据寄存器 D
DDRA    EQU   4          A 口数据方向寄存器
DDRD    EQU   7          D 口数据方向寄存器
TADAT   EQU   8          定时器 A 数据寄存器
TACR    EQU   9          定时器 A 控制寄存器
MISC    EQU   $A         杂用寄存器
TBDAT   EQU   $C         定时器 B 数据寄存器
TBCR    EQU   $D         定时器 B 控制寄存器
SPIDAT  EQU   $E         SPI 数据寄存器
SPICR   EQU   $F         控制寄存器
PRESCL  EQU   $10
*
SS      EQU   0          从机控制位,PD0
SCK     EQU   1          时钟 SCK,PD1
SDA     EQU   2          PD2
CLAMP   EQU   3          PD3
SPE     EQU   4          PD4
SPIF    EQU   7          PD7
        ORG   $20
BYTCNT  RMB   1          计数器
DATLEN  RMB   1          数据个数
DATA    RMB   40         数据
*
        ORG   $80
START   EQU   *
        BSR   INIT       转初始化子程序
        CLRA
START1  LXD   #1         IX=1
        BSR   XFER       转 XFER
        CMP   #1         A 与 1 比较
        BEQ   READ       相等转 READ
        CMP   #2         A 与 2 比较
        BEQ   SEND       相等转 SEND
        BRA   START1     转 START1
*
READ    EQU   *
        LDX   #1         IX=1
        BSR   XFER       转 XFER
        STA   BYTCNT     存接收的数据个数,它为计数器的初值
```

```
        STA     DATLEN          存入数据个数变量中
READ1   LDX     #1              IX=1
        BSR     XFER            转 XFER
        LDX     BYTCNT          IX=计数器值
        SUB     #$20            接收的数据值减去$20
        STA     DATA,X          存数据
        DEC     BYTCNT          计数器减1
        BNE     READ1           不等于0,转 READ1
        BRA     START1          返回 START1
*
SEND    EQU     *
        LDA     DATLEN          A=数据个数
        STA     BYTCNT          存入计数器中
        LDX     #5              IX=5
        BSR     XFER            转 XFER
SEND1   LDX     BYTCNT          IX=计数器值
        LDA     DATA,X          取一个数据至 A 中
        LDX     #5              IX=5
        BSR     XFER            转 XFER
        DEC     BYTCNT          计数器减1
        BNE     SEND1           不等于0,转 SEND1
        BRA     START1          转 START1
*
INIT    EQU     *
        LDA     #$48            禁止 INT,D 口为开路输出
        STA     MISC
        LDA     #$D             将$D送入 D 口。数据线、SS、PD3 置位,清复位
        STA     PORTD
        LDA     #9              将9送入 D 口数据方向寄存器。SPI 为从机
        STA     DDRD            PD2 为 I/P,PD3 和 SS 为 O/P
        LDA     #$44            将$44送入 SPICR 中。在负跳变边采样数据
        STA     SPICR           SPI 禁止
        BRSET   SCK,PORTD,*     等待主机的控制位
        RTS                     返回
*
*********************************
*     SPI 双向数据传输子程序     *
*     数据线为高时,SPI 禁止,PD2 为 I/P,PD3 为 O/P(高)  *
* 入口:Tx 模式:累加器 A=数据,IX=5  *
*      Rx 模式:累加器 A=IX,IX=1     *
* 出口:Tx 模式:累加器 A=Tx 数据,X=$D  *
*      Rx 模式:累加器 A=Rx 数据,X=$D  *
*********************************
XFER    EQU *
        BRCLR   SCK,PORTD,*     SCK 为0? 为0,等待
```

```
BCLR    CLAMP,PORTD     为1,PD3位清0
BRSET   SCK,PORTD,*     SCK为1? 为1,等待
STA     SPIDAT          将A存入SPIDAT中
BSET    SPE,SPICR       SPE置1
STX     DDRD            将IX存入D口的数据方向寄存器
*
LDA     #$44            A=$44
LDX     #$D             IX=$D
BRCLR   SPIF,SPICR,*    SPIF=0? 等于0,等待
STA     SPICR           SPIF=1,存数据
LDA     SPIDAT          取数据至A中
STX     PORTD           IX的值送入D口
STX     DDRD            IX的值送入D口数据方向寄存器
RTS                     子程序返回
END
```

34. 利用 MC6805R3 MCU 构成的温度控制器

34.1 M6805 系列 MCU 的 A/D 简介

M6805 系列 MCU 的逐次逼近 A/D 转换器如图 34-1 所示。A/D 控制寄存器(ACR)用于选择模拟输入通道和指示 A/D 的状态。进行一次 A/D 转换需 30 个机器周期。ACR 的位 7 是 A/D 转换完成标志位，该位为 1 表明可以从 A/D 结果寄存器(ARR)读取有效的数字量。ACR 的低 3 位(ACR_2～ACR_0)用来选择进行 A/D 转换的通道，如表 34-1 所示。其中 V_{RH}和 V_{RL}分别为高、低基准电压，表中的 V_{RH}、V_{RL}、$V_{RH/2}$、$V_{RH/4}$为内部通道，只用于内部校准。

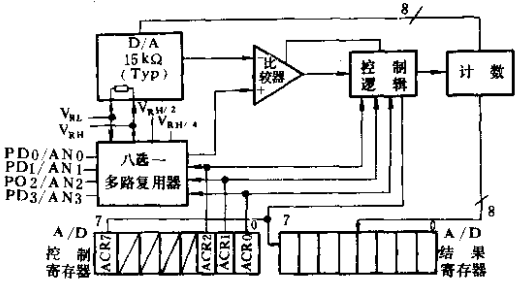


图 34-1 MC6805R 型 MCU 的 A/D 转换器

34.2 温度控制器硬件电路

表 34-1 MC6805 A/D 模拟输入选择

控制/状态寄存器的位 2～位 0(ACR2～ACR0)			选择的通道
0	0	0	AN_0
0	0	1	AN_1
0	1	0	AN_2
0	1	1	AN_3
1	0	0	V_{RH}*
1	0	1	V_{RL}*
1	1	0	$V_{RH/4}$*
1	1	1	$V_{RH/2}$*

* 只用于内部校准

MOTOROLA 的许多单片机，如 M68HC11、M6805 的 R 型和 S 型等，都带有 A/D 转换器。这里介绍用 MC68705R3 构成的简单温度控制器，这种控制器可用于需要控制温度的冰箱、空调等恒温装置或用于测试温度。

图 34-2 是采用 PN 结作为温度传感器的温度采集、转换以及数字显示的硬件电路图。温度传感器采用 MTS-102，经温度传感器转换来的弱信号经差分放大器进行放大，使其灵敏度为 1℉/20mV，差分放大器工作电压为 5V，其输出最大为 3.8V，温度传感器的温度范围为－40℉～＋140℉。差分放大器的输出直接接到 A/D 转换模拟通道 0(AN_0)，经 A/D 转换将模拟量变成数字信号，经过内部程序的正负处理和代码转换，将转换后的 BCD 码送入 B 口，由 B 口直接驱动三位共阳极 LED 显示器。三位 LED 显示器每 2048 个机器周期分时显示一位，每位的显示驱动器由 C 口的 PC2～PC0 控制，显示的时序由片内定时器决定。A/D 转换的参考电压 V_{RH}直接接 V_{FC}(＋5V)，V_{RL}直接接 V_{SS}。温度基准的校正可以通过电阻 R1，使温度传感器 MTS102 在冰块

内的显示值为 32℉。本系统较为简单，只有温度采集和显示，可以通过 A 口去控制加热、降温和风扇等，形成一个温度闭环控制，可用于冰箱、空调和恒温等装置中。

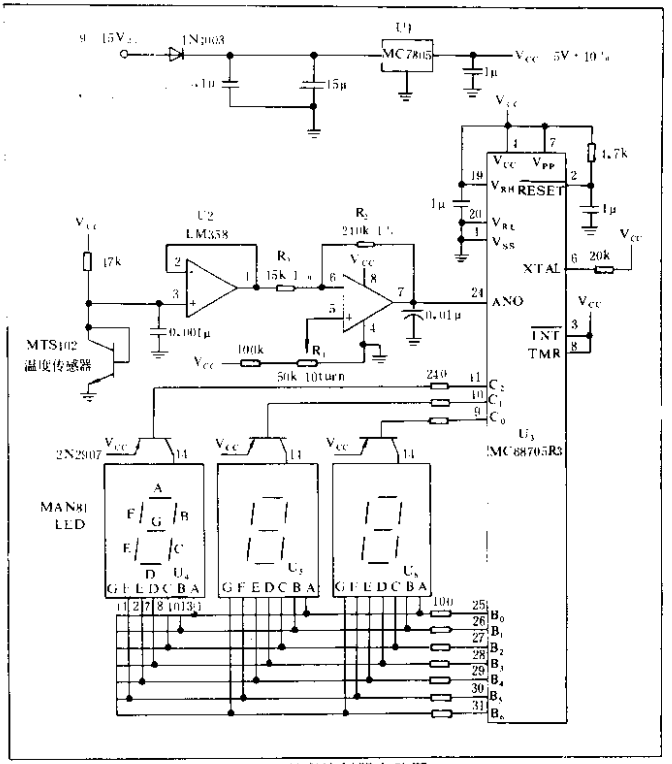


图 34-2　温度控制器电路图

34.3　程序设计

下面主要介绍温度控制器的程序设计。在进行程序设计时，设置了三个工作单元 BCDLO、BCDHI 和 SEGMNT。由于显示温度范围在－40℉～＋140℉，因此需要三位 BCD 码。当显示的温度在正华氏温度时，将 BCD 码的第 2 位存入 BCDHI 单元的低 4 位中，BCD 码的第 1 位和第 0 位存入 BCDLO 的高 4 位和低 4 位中，如果 BCD 码的第 2 位为 0，则显示闪烁，这时 BCDHI 单元的内容为 $0B(由查表法找到对应于闪烁的七段代码值，所以 $0B 实际上是相对于七段码表首地址的相对偏移量)。当显示的温度为负华氏温度时，将负号“－”存入 BCDHI 单元中(即将 $0A 存入 BCDHI 单元中，它为相对于七段码表首地址的相对偏移量)，负华氏温度下的两位 BCD 码存入 BCDLO 单元中。根据 BCD 码值，由查表法找到对应于第一位 BCD 码的七段代码值。SEGMNT 单元是为了控制显示器而设置的，SEGMNT 单元的低三位对应于 C 口的 PC2～PC0 位。由于显示器为共阳极 LED 显示器，所以当 PCi(i＝0～2)为 0 时，允许对应的显示驱动器，因而可显示出数字；当 PCi(i＝0～2)为 1 时，禁止显示驱动器。开始时，SEGMNT 单

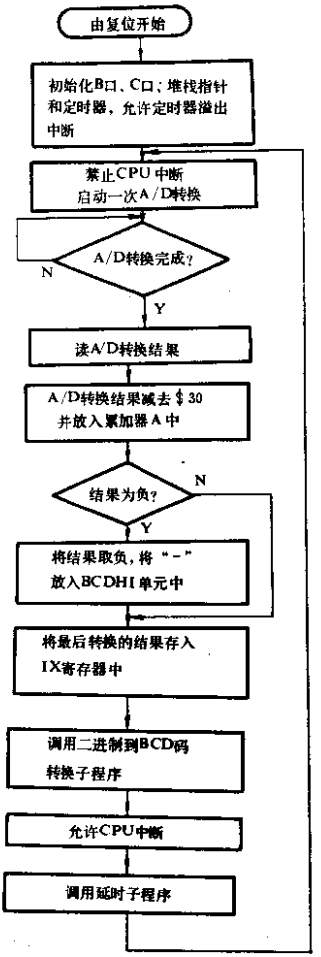


图 34-3　主程序流程图

元的内容为初值%11110111。定时器中断时将进位C置位，将SEGMNT单元的内容循环右移，变为%11111011，将该值送入C口，可以驱动显示BCD码的最高位，下一次定时器中断时，再将SEGMNT单元的内容循环右移，用于显示下一位。可以看出，每当发生定时器中断时，都将SEGMNT单元的内容循环右移，并且还要测试进位C，若C为0，则强迫SEGMNT单元的内容为%11111011，重新显示最高位，以后重复以上过程。温度控制器的程序主要包括三部分：一是主程序，包括初始化程序和温度采集、A/D转换程序；二是将A/D转换后的二进制数据转换为BCD码子程序；三是定时器中断及显示中断服务程序。下面分别介绍每一部分的编程方法。

1. 主程序设计。首先初始化C口、B口，将C口的PC2～PC0设置为输出端，将%11110111送入C口和SEGMNT单元中，以准备显示BCD码的最高位。将B口设置为输出口，并送入$FF，关闭所有3位显示驱动器。将定时器的向下计数器置为$FF，置计数器的分频因子为128，选择内部时钟作为定时器的基本时钟，并将预分频器清0，允许定时器溢出中断，这样定时器从启动到第一次溢出中断需要32768(128×256)个机器周期，以后将变为2048个机器周期中断一次(详见后)。然后进入温度采集和A/D转换程序，将A/D转换完成标志清0，选择通道0。注意，只要将A/D转换完成标志清0，则立即启动一次A/D转换。以后等待这次A/D转换完成，即不断测试A/D转换完成标志位，若仍为0，则等待，直到A/D转换完成标志为1，读A/D转换结果，进行正负温度校正。如果A/D转换的结果在正华氏温度范围内，BCDHI单元清0，否则BCDHI单元存入$0A。将A/D转换和校正后的结果(二进制数)存入IX寄存器中，调用二进制数到BCD码的转换子程序BCD，这时可允许CPU中断，调用延时子程序，最后再重新开始下一次A/D转换。其主程序流程图如图34-3所示。

2. 二进制数到BCD码转换子程序设计。该子程序将IX寄存器中的二进制数转换为BCD码。一个8位二进制数最多可转换为3位BCD码，因此将转换后的BCD码的第1位和第0位存入BCDLO单元中，BCD码的第2位存入BCDHI中。在BCD码转换过程中禁止CPU中断。首先清BCDLO单元，测试IX寄存器的内容，若不为0，是IX减1，BCDLO单元的内容加1。然后测试BCDLO单元的低4位，若不等于10，则重新测试IX的内容，重复以上过程；若等于10，则BCDLO单元的内容加$06，进行BCD调整，

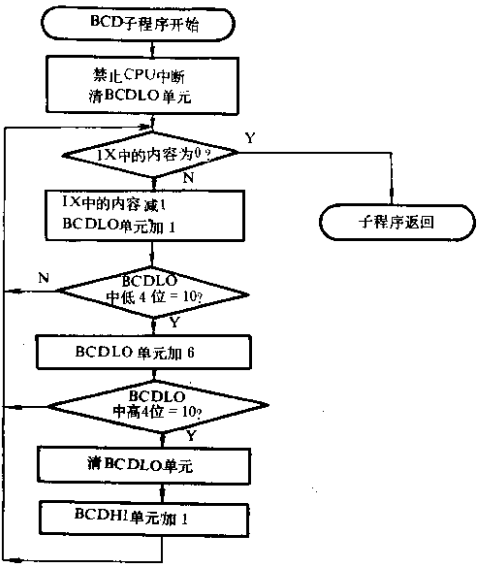


图 34-4　二进制到 BCD 码转换子程序流程图

并将相加的结果存回 BCDLO 单元中，接着测试 BCDLO 单元的高 4 位，若不等于 10，则重新测试 IX 的内容，重复以上过程；若等于 10，清 BCDLO 单元，BCDHI 单元加 1，重新测试 IX 的内容，重复以上过程，直到 IX 的内容为 0 为止。其程序流程图如图 34-4 所示。

3. 定时器中断和显示中断服务程序设计。首先将进位 C 置位，关闭显示驱动器。将 SEGMNT 单元的内容循环右移，若进位 C 为 0，则表示要重新开始显示 BCD 码的最高位，因此置 SEGMNT 单元的内容为%11111011；若不为 0，将 BCDLO 单元的内容送入累加器 A 中，并且根据 SEGMNT 单元的内容判断需要显示哪一位。若 SEGMNT 单元的第 1 位为 0，则显示 BCD 码的第 1 位，A 循环右移 4 次，否则判断 SEGMNT 单元的第 2 位，若为 0，则显示 BCD 码的第 2 位(即最高位)，将 BCDHI 单元的内容送入累加器 A 中，若 A 中的内容不为 0，显示对应的数字；若为 0，显示闪烁。若 SEGMNT 单元的第 2 位不为 0，则显示 BCD 码的第 0 位。不论显示哪一位，A 中的低 4 位为要显示的 BCD 码值，使用查表法得到对应于 BCD 码的七段代码值，并将对

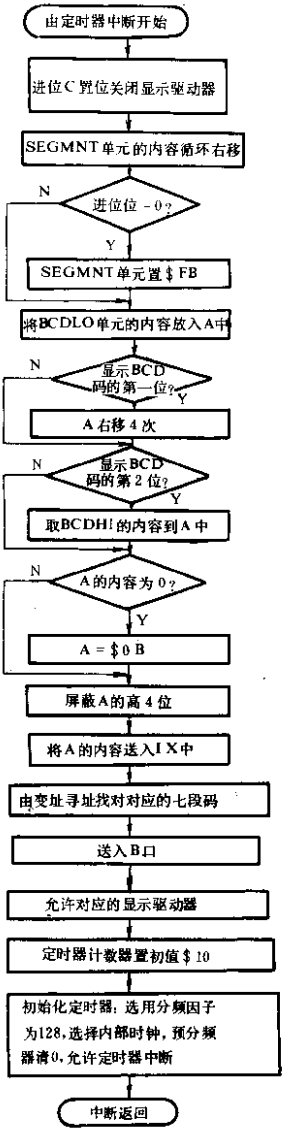


图 34-5　定时器中断和显示服务程序流程图

应的七段代码值送入 B 口，允许对应的显示驱动器，因而可显示数字。最后重新设置定时器的向下计数器的初值为 $10；定时器的分频因子为 128；选择内部时钟；预分频器清 0，允许定时器溢出中断。根据计数器的初值和分段因子，可知每经过 2048(16×128)个机器周期定时器产生溢出中断，即 2048 个机器周期显示一次。其程序流程图如图 34-5 所示。

34.4 程序清单

```
*  定义寄存器地址
PORTB       EQU     1               B 口
PORTC       EQU     2               C 口
DDRB        EQU     5               B 口数据方向寄存器
DDRC        EQU     6               C 口数据方向寄存器
TDR         EQU     8               8 位定时器计数器
TCR         EQU     9               定时器控制寄存器
ACR         EQU     14              A/D 转换控制寄存器
ARR         EQU     15              A/D 转换结果寄存器
BCDHI       EQU     $40             存放 BCD 码的第 2 位
BCDLO       EQU     $41             存放 BCD 码的第 1 位和第 0 位
SEGMNT      EQU     $42             控制显示位单元
            ORG     $80
   定义七段代码表
  FCB  %01000000        0           为 $40
  FCB  %01111001        1           为 $79
  FCB  %00100100        2           为 $24
  FCB  %00110000        3           为 $30
  FCB  %00011001        4           为 $19
  FCB  %00010010        5           为 $12
  FCB  %00000010        6           为 $02
  FCB  %01111000        7           为 $78
  FCB  %00000000        8           为 $00
  FCB  %00011000        9           为 $18
  FCB  %00111111        -           为 $3F
  FCB  %01111111        BLANK       为 $7F
* 主程序
ORG         $90
START       RSP     堆栈指针复位
            LDA     #$F7        初始化 C 口，选择第一次显示为最高位
            STA     PORTC       %11110111 存入 C 口
            STA     SEGMNT      %11110111 存入 SEGMNT 单元中
            LDA     #$07        使 C 口 PC0～PC2 为输出
            STA     DDRC
            LDA     #$FF        初始化 B 口
            STA     PORTB       关闭所有 3 位显示
            STA     DDRB
            STA     TDR         定时器计数器置初值 $FF
            LDA     #$0F        初始化定时器，使分频因子为 128，选
```

```
            STA     TCR             择内部时钟、预分频器清0、允许定时器中断
* A/D转换器程序：
ADC         SEI                     禁止CPU中断
            CLR     ACR             清转换完成标志，选择通道0，同时启动A/D
            LDA     ACR             读ACR
            BPL     *-2             检查转换完成标志，ACR的b7位=0，重新执行上一句
            LDA     ARR             ACR的b7位为1，读A/D转换结果，A/D转换完成
            SBC     #$30            将A/D转换结果减去$30
            CLR     BCDHI           清BCD码的高位
            CMP     #$CF            A中的结果为负吗？
            BLO     CONVERT         不为负，转CONVERT
            NEGA                    A取补
            LDX     #$0A            将BCDHI中放入负号"-"
            STX     BCDHI
COVERT      TAX                     将A的内容放入IX寄存器中
            JSR     BCD             调用二进制到BCD码转换子程序
*
            CLI                     允许CPU中断
            BSR     DELAY           调用延时子程序
            BRA     ADC             无条件转移到ADC，重新测试温度
***** 二进制转换为BCD码子程序 ***
*   IX寄存器中存放着A/D转换后的二进制数
*   转换后的BCD码存放在显示RAM存贮区中
BCD         EQU     *
            SEI                     禁止CPU中断
            CLR     BCDLO           清BCD码的低位
LOOP        TSTX                    测试IX寄存器
            BEQ     EXIT            若为0，退出
            DECX                    不为0，IX减1
            INC     BCDLO           BCD码的低位加1
            LDA     #$0F            屏蔽高4位
            AND     BCDLO
            CMP     #$0A            与$0A比较
            BNE     LOOP            不相等转LOOP
            LDA     BCDLO           相等，BCD调整(加6)
            ADD     #$06
            STA     BCDLO           存回BCDLO单元中
            CMP     #$A0            且与$A0比较
            BNE     LOOP            不相等，转LOOP
            CLR     BCDLO           相等，清BCDLO
            INC     BCDHI           BCDHI单元加1
            BRA     LOOP            转LOOP
EXIT        RTS                     子程序返回
* 延时子程序
DELAY       EQU     *
            LDX     #$FF            IX寄存器等于$FF
```

```
        LDA     # $FF          累加器 A 等于 $FF
        DECA                  A 减 1
        BNE     DELAY+4       不等于 0,重新执行 DECA 指令
        DECX                  等于 0,IX 减 1
        BNE     DELAY+2       不等于 0,重新执行 LDA # $FF 指令
        RTS                   子程序返回
* * * 定时器中断和显示服务程序 * * *
*     BCD 码转换为 7 段显示码并进行显示,首先显示最高位
*     显示速率为 30ms,采用定时器中断来显示
TMRINT  EQU     *             -
        SEC                   进位位置 1
        LDA     # $FF          关闭七段驱动器
        STA     PORTB
        ROR     SEGMNT        SEGMNT 单元的内容循环右移,进位 C 移入最高位
        BCS     LOBYTE        进位 C=1,转 LOBYTE,
        LDA     # $FB          C=0,A=%11111011
        STA     SEGMNT        将 A 值存入 SEGMNT,重新从高位显示
LOBYTE  LDA     BCDLO         取第 0 位和第 1 位 BCD 码
        BRSET   1,SEGMNT,HIBYTE  SEGMNT 单元中的位 1 为 1,转 HIBYTE
        LSRA                  不为 1,显示第 1 位 BCD 码,A 右移 4 次
        LSRA
        LSRA
        LSRA
HIBYTE  BRSET   2,SEGMNT,DISPLY  SEGMNT 单元中的位 2 为 1,转 DISPLY
        LDA     BCDHI         不为 1,显示 BCD 的第 2 位
        BNE     DISPLY        不等于 0,转 DISPLY
        LDA     # $0B         等于 0,A=#0B,显示闪烁
DISPLY  AND     # $0F         屏蔽高 4 位
        TAX                   将 A 中的内容送入 IX 中
        LDX     SEVSEG,X      取对应的七段显示码
        STX     PORTB         送入 B 口
        LDA     SEGMNT        允许对应的位显示
        STA     PORTC
        LDA     # $10         定时器计数器置初值 $10
        STA     TDR           2048 个机器周期显示一次
        LDA     # $0F         初始化定时器:使分频因子为 128,选择内部时钟,预分
        STA     TCR           频器清 0,允许定时器溢出中断
        RTI
        ORG     $F38
        FCB     $80           置程序屏蔽选择寄存器为 $80,选择 RC 时钟
        ORG     $FF8          置复位和中断向量
TMRVEC  FDB     TMRINT
IRQVEC  FDB     START
SWIVEC  FDB     START
RESET   FDB     START
        END
```

35. MC68HC705T3 自引导程序/自引导编程电路

图 35-1 是由外围 EPROM 对 MC68HC705T3 进行编程的电路。外部 EPROM 地址与 MCU 内部 EPROM 地址直接对应。其 OSD 显示字符应位于 $0400～$0AFF，程序位于 $2000 ～ $7EFF，向量位于 $7FF0 ～ $7FFF。必须注意，用 MC68HC705T3 对 MC68HC05T1/T2 进行仿真时，向量地址和程序起始地址应改为要仿真的型号所对应的地址。MC68HC05T3 的 OSD 缓冲器和 OSD 字符、RAM、ROM 均比 MC68HC05T1/T2 多。

MC68HC705T3 的自引导程序有 4 种操作方式，通过开关 S1 和 S2 选择，除能对 EPROM 进行编程和校验外，还能将程序装入位于 $0100～$01FF 的 RAM 中并执行。象 EPROM 编程器一样，RAM 装入程序从对应的外部 EPROM 地址传输数据，没有串行接口。

35.1 操作步骤

1. 关闭 V_{DD} 电源，S3 闭合（复位 RESET 位置），S4 为 I 位置。

2. 插上 MC68HC705T3。

3. 开启 V_{DD}，如果这一操作不在步骤 4 之前进行，可能损坏 MC68HC705T3。

4. 若要对 EPROM 进行编程，将 S4 打到 E 位置，加上 V_{PP} 电源（18.5V）。对 EPROM 进行校验、装入 RAM 并执行这些操作模式不需要 V_{PP}。但对于编程以外的其他 3 种模式在复位后要求引脚 2（$\overline{IRQ}$）为至少 9V 电压，这一电压由 S3 从 RESET 位置打到 RUN 位置和其他电路提供。

5. 用 S1 和 S2 选择操作模式。

6. 断开 S3，开始在选中的模式进行操作，红 LED 亮表示正运行自引导程序。编程结束后进行校验，如校验通过，则绿 LED 亮。若绿 LED 不亮，表明检验中发现错误。

7. 关闭 S3，可再次返回步骤 5。

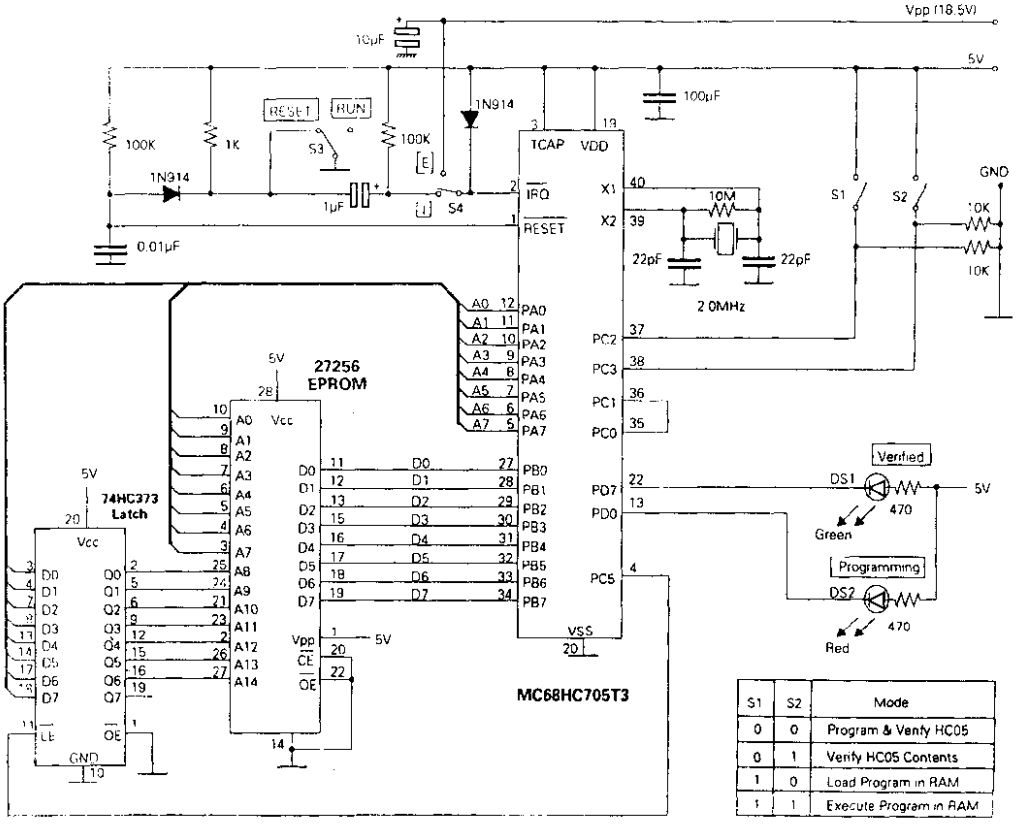
8. 若 S4 在位置 E，打回到 I 位置，断开 V_{PP}。

9. 关闭 V_{DD}。

10. 取出 MC68HC705T3。

35.2 软件

自引导程序位于 MC68HC05T3 的 ROM 自测试程序区：$7F00～$7FEF。从 $7F00 开始的程序立即检查 C 口位 2 和 3 的状态。若均为 1，则转至 $0100 执行，这样可运行以前装入到 RAM 中的程序。若 PC2 和 PC3 均为 0，则软件初始化 I/O 口。因 B 口为固定输入引脚，不需写入其数据方向寄存器。子程序 STXHIS 用来使地址总线初始化为所使用的第一个 EPROM 单元（$0400）。因 MC68HC705T3 以及所有 M6805 MCU 只有 8 位变址寄存器，有必要执行 RAM 中的程序，以便灵活地选择写入或读出片内 EPROM 的地址。对于写入 EPROM，该程序由扩展 STA 指令和随后的 2 字节地址以及 RTS 指令组成。将 TABLE 的 6 字节内容传输到 RAM 可建立该程序。第 5 和第 6 字节是用于控制 EPROM 写入时序的常数（实际上传输 8 字节，但只有前 6 个字节有关）。传输之后进行条件转移（取决于 PC2 的电平）。若



S1	S2	Mode
0	0	Program & Verify HC05
0	1	Verify HC05 Contents
1	0	Load Program in RAM
1	1	Execute Program in RAM

图 35-1　MC68HC705T3 自引导电路

PC2 为高，进入 RAM 装入程序；若 PC2 低，则开始 EPROM 引导程序（编程或校验，取决于 PC3 的电平）。

RAM 装入程序将外部 EPROM 的 $0100～$01FF 的内容传输到 MC68HC705T3 相同的地址并校验，若校验成功，则使 PD7 相连的 LED 亮，并进入 STOP 模式。若校验发现有错，则程序在出错地址停止。

EPROM 装入程序将外部 EPROM 的数据传输到 MCU 内部 EPROM。写入 EPROM 需几毫秒。自引导程序执行两个循环，每个字节写入时间均为 2ms。该循环结束后（需 110 秒），执行校验循环，若校验成功，绿 LED 亮。若校验发现错误，则在出错的地址停止。A 口为出错地址的低 8 位，高位地址位于外部 EPROM 和锁存器输出之间。

程序的最后部分是子程序 STXHI，它有两个进入点，子程序 NXTADR、TABLE 和自引导程序向量。NXTADR 用于增加 EPROM 地址，并跳过不需 EPROM 编程的区域。

35.3 自引导程序清单

```
PORTA     EQU     $00          Port A address
PORTB     EQU     $01          Port B
PORTC     EQU     $02          Port C
PORTD     EQU     $03          Port D
DDRA      EQU     $04          Port A data direction reg
DDRC      EQU     $06          Port C
DDRD      EQU     $07          Port D
PROG      EQU     $1C          EPROM Program register
TR1       EQU     $3E          Test register 1
          ORG     $40
RAM       RMB     1
ADDR      RMB     2
RET       RMB     1
LOOP      RMB     1
TIME      RMB     1
          ORG     $7F00
* 执行 RAM 中的程序，设置口
START     BRCLR   2,PORTC,BOOT    CHECK FOR JMP TO RAM OR BOOT
          BRCLR   3,PORTC,BOOT
RSTRT     JMP     $0100           JUMP TO PROGRAM IN RAM
BOOT      BSET    0,TR1           SWITCH OSD CHARACTER EPROM IN-
                                  TO MEMORY MAP
          LDA     #$FF
          STA     DDRA            ALL OUTS,ADDRESSES
          STA     DDRD            ALL OUTS,LEDS
          STA     PORTD           LEDS OFF
          LDA     #%00100010      0, 1: HANDSHAKE, 2 & 3: OPTION
                                  SWITCHES,
          STA     DDRC            4:TCAP,5:LATCH
          CLRA
          LDX     #4              X←00000100
```

```
        JSR     STXHIS          ADHI←$04,ADLO←$00
        LSLX                    X←$08
MOVE    LDA     TABLE-1,X
        STA     RAM-1,X         $C7,$04,$00,$81,$02,$02
        DECX
        BNE     MOVE
        BRCLR   2,PORTC,PRGVER  EPROM OR RAM
* 并行 RAM 装入/校验($0100~$01FF)
LDRAM   BSR     RMSTRT
PLOOP   BSR     HAND1           HANDSHAKE AND GET DATA
        BRSET   3,PORTC,NEXT    SKIP RAM LOAD?
        STA     $0100,X         NO
NEXT    INC     PORTA
        INCX
        BNE     PLOOP
        BSR     RMSTRT
PVERF   BSR     HAND1           HANDSHAKE AND GET DATA
        CMP     $0100,X
        BNE     *               HANG UP IF NOT OK
        INC     PORTA
        INCX
        BNE     PVERF
        BRSET   3,PORTC,RSTRT   IF PUT HIGH DURING VERIFY
        BRA     FIN             THEN EXECUTE PROGRAM IN RAM.
RMSTRT  LDX     #1
        CLRA                    START($0100)
        BSR     STXHIS
        CLRX
        RTS
* EPROM 编程和校验($0400~$0AFF,$2000~$7EFF,$7FF0~$7FFF)
PRGVER  BRSET   3,PORTC,VERF    DO A VERIFY ONLY?
PRGLOP  BSR     HAND2           HANDSHAKE AND GET DATA
        BSET    5,PROG          LATCH ADDRESS & DATA
        JSR     RAM             WRITE ONE BYTE
        BSET    0,PROG          APPLY VPP
        LDA     TIME            GET PROGRAMMING TIME IN ms
DELNMS  LDX     #$A6            2ms INNER LOOP(2MHz XTAL)
MS1     DECX
        BNE     MS1
        DECA                    X A OUTER LOOP
        BNE     DELNMS
        CLR     PROG            REMOVE VPP
        BSR     NXTADR          NEXT ADDRESS
        BNE     PRGLOP          DONE?
        LDX     #4              GET INITIAL MS ADDR
        STX     ADDR            ADDRHI←$0400
```

```
        BSR     STXHI       A8-A14← $04
        DEC     LOOP
        BNE     PRGLOP      DO PROG LOOPS TWICE
VERF    INC     RAM         CHANGE STA TO EOR
CHECK   BSR     HAND2       HANDSHAKE AND GET DATA
        JSR     RAM         COMPARE WITH EPROM BYTE
HANG1   BNE     HANG1       HANG UP IF DIFFERENT
        BSR     NXTADR      NEXT
        BNE     CHECK
FIN     LDA     # $7F
        STA     PORTD       VERIFY LED ON,REST OFF
        STOP
*应答和读外部 EPROM
HAND1   TXA                 ADDRLO(RAM)
        AND     # $7F        MAKE SURE VERIFY LED IS OFF
        BRCLR   3,PORTC,DISP DISPLAY RAM DATA OR ADDRESS?
        LDA     $0100,X     DATA
        BRA     DISP
HAND2   LDA     ADDR        ADDRHI(EPROM)
DISP    COMA
        STA     PORTD       DISPLAY ADDRESS (OR DATA)
        BSET    1,PORTC     HANDSHAKE HIGH
        BRCLR   0,PORTC,*
        BCLR    1,PORTC     AND LOW AGAIN
        BRSET   0,PORTC,*
        LDA     PORTB       READ AN EXTERNAL BYTE
        RTS
*子程序
STXHI   LDA     PORTA       ADDRLO,LOAD ORIG. CONTENTS
STXHIS  BSET    5,PORTC     UPDATE LATCH
        STX     PORTA       ADDRHI
        BCLR    5,PORTC     LATCH CONTENTS OF ADDRHI
        STA     PORTA       RESTORE CONTENTS OF ADDRLO
        RTS
NXTADR  INC     ADDR-1      INC. ADDRLO
        INC     PORTA
        BNE     GOBACK      RETURN IF NOT PAGE BOUNDARY
        LDX     ADDR        GET ADDRHI
        INCX                INC.  ADDRHI
        CPX     # $80
        BEQ     GOBACK      END OF VECTORS? IF SO,EXIT WITH Z=1
        CPX     # $7F
        BNE     NOTEND      END OF MAIN BLOCK?
        LDA     # $F0       MOVE TO USER VECTORS
        STA     ADDR+1      UPDATE ADDRLO
        STX     ADDR        UPDATE ADDRHI
```

```
        BSR     STXHIS          UPDATE LATCH (ADDRHI)
GOBACK  RTS                     Z=1 IF FINISHED
NOTEND  CPX     # $ 0B          WAS THAT END OF D EPROM?
        BNE     GO
        LDX     # $ 20         MOVE TO USER EPROM BEGINNING
GO      STX     ADDR            UPDATE ADDRHI
        BSR     STXHI           UPDATE EXTERNAL LATCH OF ADDRHI
        COMA                    CLEAR Z FLAG
        RTS
TABLE   FCB     $ C7            STA EXTENDED INSTRUCTION
        FDB     $ 0400          START ADDRESS
        FCB     $ 81            RTS INSTRUCTION
        FCB     2               2 PROGRAMMING LOOPS
        FCB     2               2 ms PROGRAMMING TIME (PER LOOP)
* 自引导向量
        ORG     $ 7FEE
RESET   FDB     START           RESET VECTOR
        END
```