1 需求分析

用实验箱上 4*4 键盘的按键模拟汽车的挡杆，用发光二极管显示挡位，用数码管显示汽车的速度。

1、“1”键启动系统，汽车以最低速度行驶，同时用 1 盏绿灯显示挡位，数码管显示速度（最低速度为 5Km / h）。当需要换档时，用键盘键入 2、3 键，并加用一盏黄灯和一盏红灯显示，同时在数码管上显示相应的速度。

2、汽车慢加速时用“A”键，急加速时用“B”键。慢刹车时用“C”键，急刹车时用“D”键。加速和刹车时用数码管显示相应的速度变化。

3、当汽车需紧急停车时，键入“ESC”键，所有发光二极管熄灭，同时数码管显示“0”。

4、各档位车速：

1 挡：5～25 Km / h

2 挡：25～60 Km / h

3 挡：60～120 Km / h

计算机系统的信息交换有两种方式：并行数据传输方式和串行数据传输方式。并行数据传输是以计算机的字长为传输单位，通常是 8 位、16 位或 32 位，一次传送一个字长的数据。适合于外部设备与微机之间进行近距离、大量和快速的信息交换，如微机与并行接口打印机、磁盘驱动器等。并行传输方式是微机系统中最基本的信息交换方式，例如,系统板上各部件之间(CPU 与存储器，CPU 与外设接口电路等)，I/O 通道扩充板上各部件之间等的数据交换都是并行传输。

为了满足以上需求，同时提高外设与 CPU 通信的效率和灵活性，本课程设计选择使用并行传输方式，应用微机原理、汇编语言及接口技术实践了 PC 机基于用 8255A 芯片与 LED 数码管、矩阵键盘的具体可行的通信和控制的方式，本设计使用的通信和控制的方式具有稳定可靠及节省信号线、节省端口的特点。

2 概要设计

8255A 是 Intel86 系列微处理器的配套并行接口芯片，它可为 80x86 系列 CPU 与外部设备之间提供并行输入/输出的通道。

本课程设计主要使用 8255A 的 A、B 和 C 三个端口实现对不同外设的通信和控制。

首先由 A 端口控制七段数码管的段码，B 端口控制 LED 灯和位码，最后 C 端口检测 4*4 键盘的输入。图 2.1 为课程设计的主要循环流程。

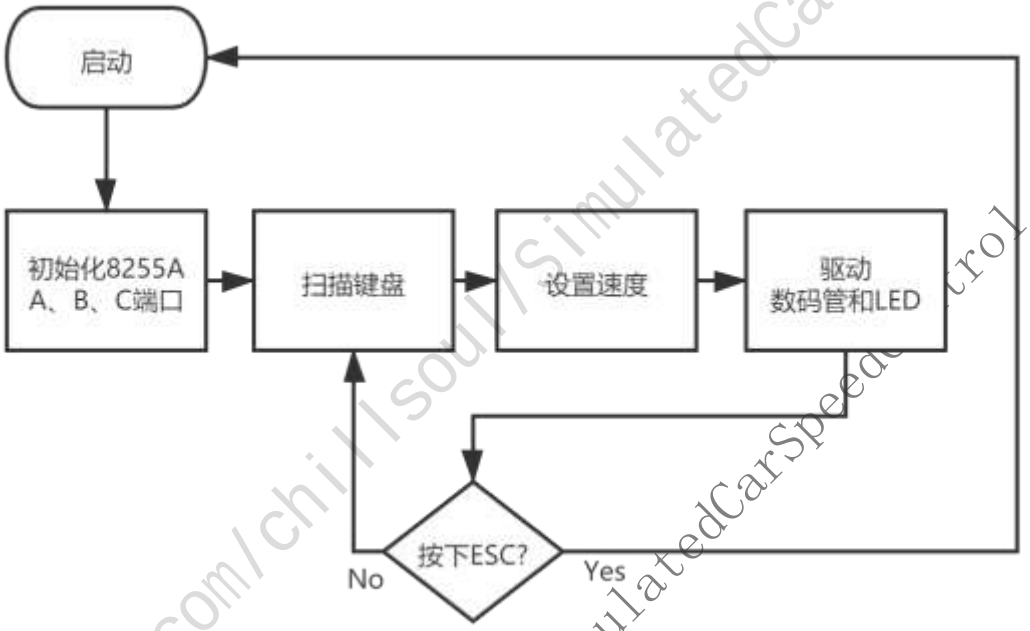

图 2.1 概要流程

程序编译执行后，将等待键盘按下 1 以控制 LED 数码管逐渐提速到 5（初始速度），此后将等待加速、提档、减速等多个对应键盘按键按下以转移程序分支。本次课程设计本质上是一个状态机，循环中改变状态变量，不断改变输出的值。当主机按下 ESC 键时，数码管复位 000，LED 灯全灭，表示紧急刹车。

3 运行环境

操作系统：Windows XP

实验箱：清华科教厂 TPC-ZK-II

软件环境：TPC-ZK-II 集成开发环境

本课程设计为软硬件结合的汇编程序设计，若不具备上述运行环境，可能无法运行。

4 开发工具和编程语言

TPC-ZK-II 集成开发环境提供了本课程设计所需的全部开发工具，使用

8086/8088 的指令系统以汇编语言语言形式编程。

4.1 程序清单

wjyl.asm 包含本次课程设计所有的源代码。

5 详细设计

该部分将详细介绍本次课程设计所用的软硬件设计，包括了汇编语言子程序和芯片接线、控制逻辑。

5.1 8255A 介绍

一般来说，外设接口可以分成两类：

(1) 并行接口：一组数据在多根线上同时传送；

(2) 串行接口：一组数据按位顺序在一根线上依次传送。

出于效率的考虑，本实验使用并行接口芯片 Intel 8255A 实现 CPU 与外设直接的并行通信。

Intel 公司生产的可编程并行接口芯片 8255A 已广泛应用于实际工程中， 例如 8255A 与 A/D、 D/A 配合构成数据采集系统，通过 8255A 连接的两个或多个系统构成相互之间的通信，系统与外设之间通过 8255A 交换信息，等等，所有这些系统都将 8255A 用作为并行接口。

8255A 的原理结构如图 5.1.1 所示。 它采用 40 脚的 DIP 封装，其引脚定义如表 5.1.1 所示。

表 5.1.1 8255A 引脚定义

引脚名	功能	连接去向
$D_0 \sim D_7$	数据总线（双向）	CPU
RESET	复位输入	CPU
$\overline{CS}$	片选信号	译码电路
$\overline{RD}$	读信号	CPU
$\overline{WR}$	写信号	CPU
A_0, A_1	端口地址	CPU
$PA_0 \sim PA_7$	端口 A	外设
$PB_0 \sim PB_7$	端口 B	外设
$PC_0 \sim PC_7$	端口 C	外设
Vcc	电源（+5V）	/
GND	地	/

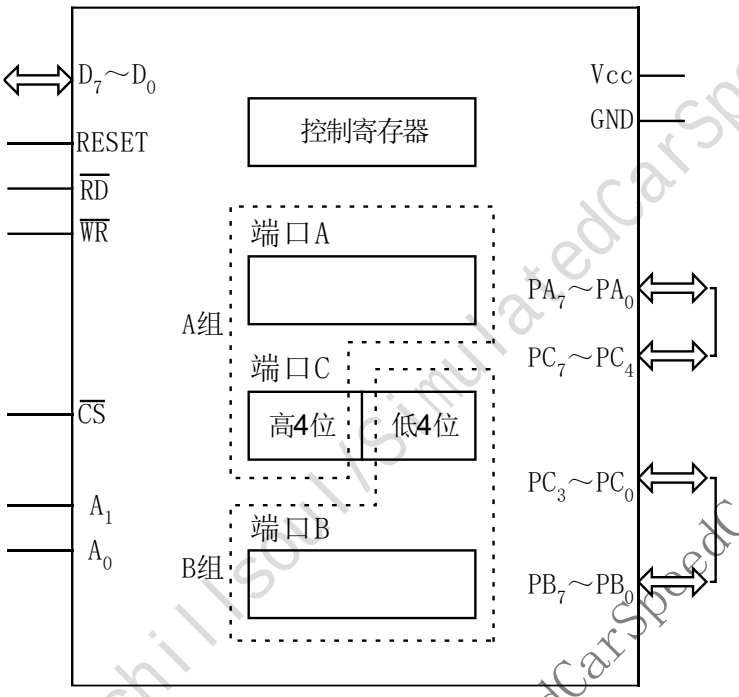

图 5.1.1 8255A 编程模型

8255A 为一可编程的通用接口芯片。它有三个数据端口 A、B、C，每个端口为 8 位，并均可设成输入和输出方式，但各个端口仍有差异：

端口 A（PA0~PA7）：8 位数据输出锁存/缓冲器，8 位数据输入锁存器；

端口 B（PB0~PB7）：8 位数据 I/O 锁存/缓冲器，8 位数据输入缓冲器；

端口 C（PC0~PC7）：8 位输出锁存/缓冲器，8 位输入缓冲器（输入时没有锁存）；在模式控制下这个端口又可以分成两个 4 位的端口，它们可单独用作为输出控制和状态输入。

端口 A、B、C 又可组成两组端口（12 位）：A 组和 B 组， 参见图 A.2。在每组中，端口 A 和端口 B 用作为数据端口，端口 C 用作为控制和状态联络线。

在 8255A 中，除了这三个端口外，还有一个控制寄存器，用于控制 8255A 的工作方式。

因此 8255A 共有 4 个端口寄存器，分别用A_0，A_1指定：

$A_1 = 0$， $A_0 = 0$，表示访问端口 A；

$A_1 = 0$， $A_0 = 1$，表示访问端口 B；

$A_1 = 1$， $A_0 = 0$，表示访问端口 C；

$A_1 = 1$， $A_0 = 1$，表示访问控制寄存器；

8255A 有三种基本工作方式：

方式 0：基本的输入/输出

方式 1：有联络信号的输入/输出；

方式 2：双向传送。

A 组可采用方式 0～方式 2，而 B 组只能采用方式 0 和方式 1，这由 8255A 的方式控制字控制。当向$A_1 = 1$、$A_0 = 1$的端口寄存器（即控制寄存器）发送$D_7 = 1$的控制字时，其作用为方式控制字，各个位的含义如图 5.1.2 所示。

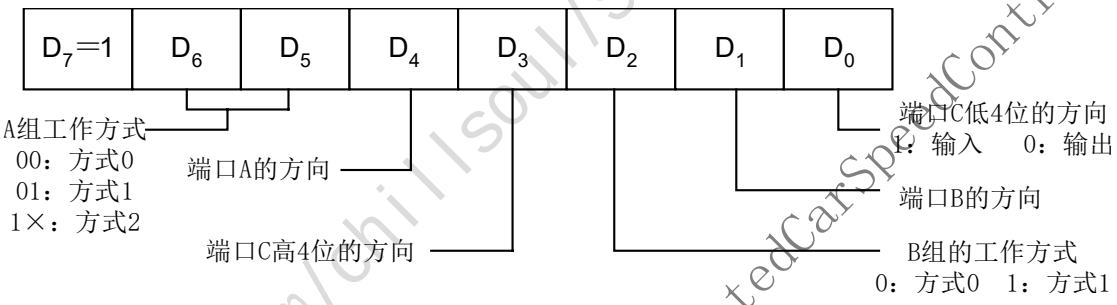

图 5.1.2 8255A 方式控制字

概要设计中提到，A 端口控制七段数码管的段码，B 端口控制 LED 灯和位码，最后 C 端口检测 4*4 键盘的输入。

出于简单考虑，本课设中 A、B 和 C 三个端口均使用方式 0，且 C 端口分配为高 4 位低 4 位分别为输入/输出（或输出/输入）方式即可满足需求。代码如下：

```
MOV DX,IO8255_K
MOV AL,10000001B ;将 8255 设为 A、B 口输出、C 口上半输出下半输入
OUT DX,AL
```

5.2 键盘扫描子程序

由于 TPC-ZK-II 实验箱配备的 4*4 简易键盘为矩阵结构，配备了 8 个信号引脚，故对于此键盘的按键识别，可以使用扫描法或反转法。

扫描法识别按键的原理如下:先使第 0 行接低电平，其余行为高电平，然后看第 0 行是否有键闭合。这是通过检查列线电位来实现的，即在第 0 行接低电平时，看是否有哪条列线变成低电平。如果有某列线变为低电平，则表示第 0 行和此列线相交的位置上的键被按下,如果没有任何一条列线为低电位，则说明第 0 行没有任何键被按下。此后，再将第 1 行接地，然后检测列线是否有变为低电位的线。如此往下一行一行地扫描，直到最后一行。在扫描过程中，当发现某一行

有键闭合时，也就是列线输入中有一位为 0 时，便在扫描中途退出，通过组合行线和列线即可识别此刻按下的是哪一键。

而反转法是在扫描法的基础上改进而来，省去了一次循环数据通信，故而效率更高，其不足之处在于需要连接行线和列线的接口电路必须支持动态改变输入输出，而 8255A 的 C 端口恰好支持。

反转法识别按键的原理为：首先，将行线作为控制线接一个输出端口，将列线作为检测线接一个输入端口。CPU 通过输出端口将行线（控制线）全部设置为低电平，然后从输入端口读取列线（检测线)。如果此时有键被按下，则必定会使某列线为“0”。然后，将行线和列线的作用互换，即将列线作为控制线接输出端口，行线作为检测线接输入端口。并且，将刚才读得的列值从列线所接端口输出，再读取行线的输入值，那么，闭合键所在的行线值必定为“0”。这样,当一个键被按下时，必定可以读得一对唯一的行值和列值。

综合考虑，为了节约内存及编写方便，选择反转法识别。流程逻辑如图 5.2.1。

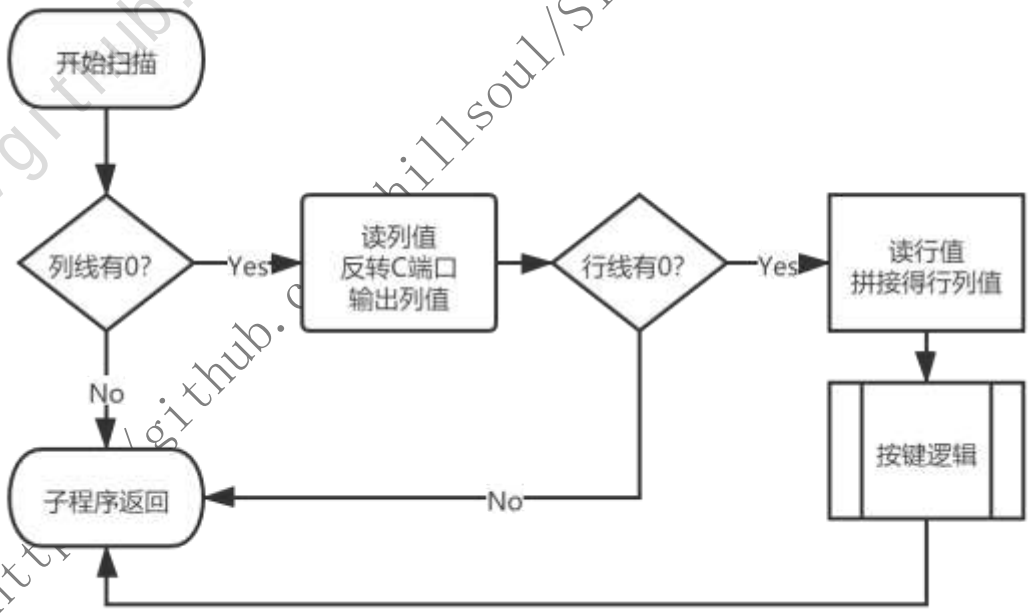

图 5.2.1 键盘识别子程序流程

Proteus 仿真接线可参见图 5.2.2，代码附后。

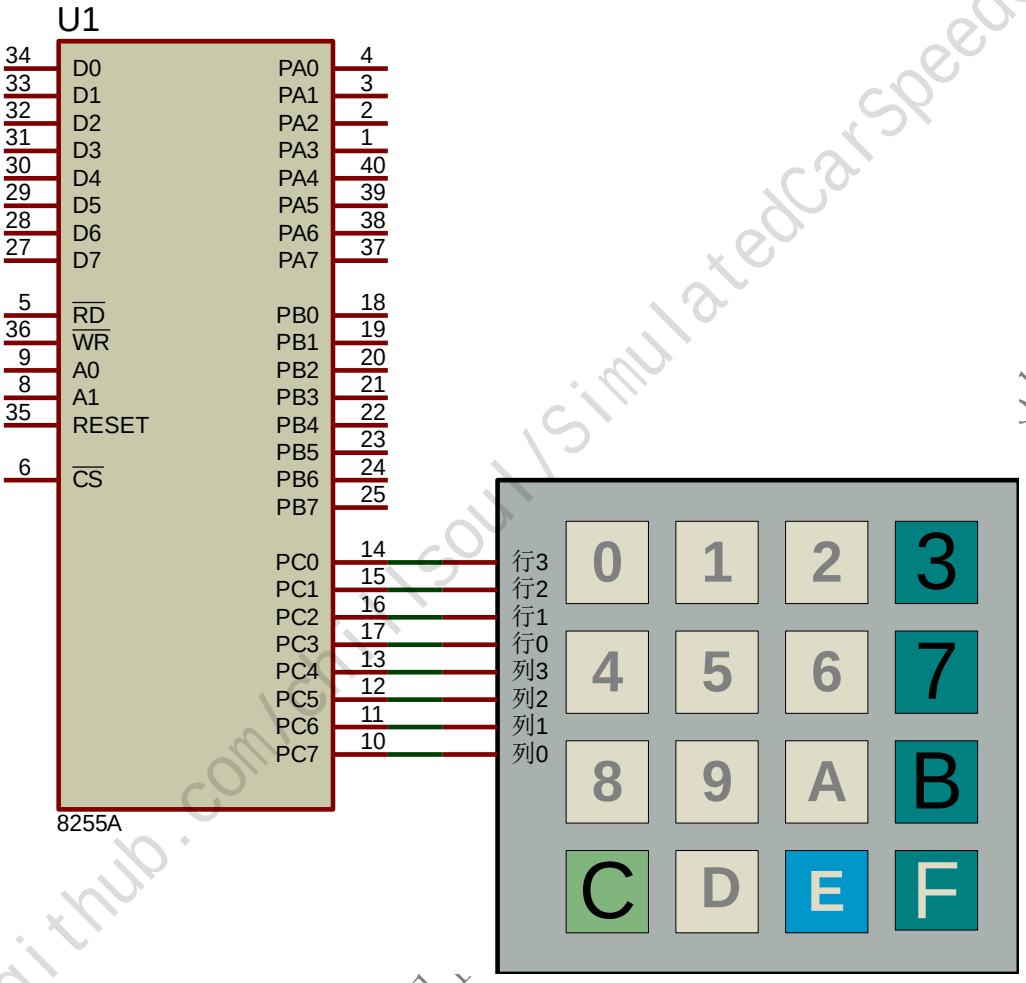

图 5.2.2 4*4 矩阵键盘接线

按键识别子程序代码：

```
;扫描键盘输入
scan4x4Keyboard PROC
    .... ;寄存器入栈
    ;反转法识别键盘
    MOV DX,IO8255_C
    MOV AL,00000000B ;八位为行 3210 列 3210，此处使行输出 0
    OUT DX,AL
    IN  AL,DX ;读取列值到低四位
    CMP AL,0FH ;判断低四位是否有 0
    JZ  endOfScan4x4Keyboard
STEP1:
    PUSH AX
    PUSH AX
    MOV DX,IO8255_K
    MOV AL,10001000B ;将 8255 设为 A、B 口输出、C 口上半输入下半输出
    OUT DX,AL
    MOV DX,IO8255_C
```

```
    POP AX     ;恢复上次扫描到的列值，低四位有效
    OUT DX,AL  ;将列值输出到列
    IN  AL,DX  ;读取行值到高四位
    AND AL,0F0H ;清空低四位
    POP BX     ;再次恢复上次扫描到的列值，低四位有效
    AND BL,0FH;清空高四位
    MOV AH,AL  ;行值移到 AH 高 4 位
    ADD AH,BL  ;列值移到 AH 低 4 位，此时行列值合并到 AH
    MOV SI,OFFSET KEYTABLE
    MOV DI,OFFSET KEY
    MOV CX,16  ;4x4 键盘共 16 个键需要比对
STEP2:
    CMP AH,[SI]
    JZ  STEP3
    INC SI
    INC DI
    LOOP STEP2
    ;未找到，恢复 8255 初始模式
    MOV DX,IO8255_K
    MOV AL,10000001B ;将 8255 设为 A、B 口输出，C 口上半输出下半输入
    OUT DX,AL
    JMP endOfScan4x4Keyboard ;未找到键
STEP3:
    .... ; 从[DI]获取键盘字母 char 比对及后续逻辑分支
endOfScan4x4Keyboard:
    .... ;寄存器恢复
    RET
scan4x4Keyboard ENDP
```

5.3 速度设置子程序

速度设置部分是本次课程设计的核心，由于课程设计题目要求分支众多，逻辑复杂，为了避免遗漏条件转移，我采用状态机的设计模式，即根据系统不同的状态，输出（设置）相应的结果，保证将所有的状态变量的排列组合全部实现，避免遗漏。

程序中主要的状态变量及设置为：

```
    global_state DB 0 ;=0 停止，1 启动第一档,2 第二档，3 第三档
    limit        DW  0,0 ;限速设置，最低，最高
    speed_state DB  0 ;=0 匀速，<0 减速，>0 加速
```

子程序设计流程见图 5.3.1。

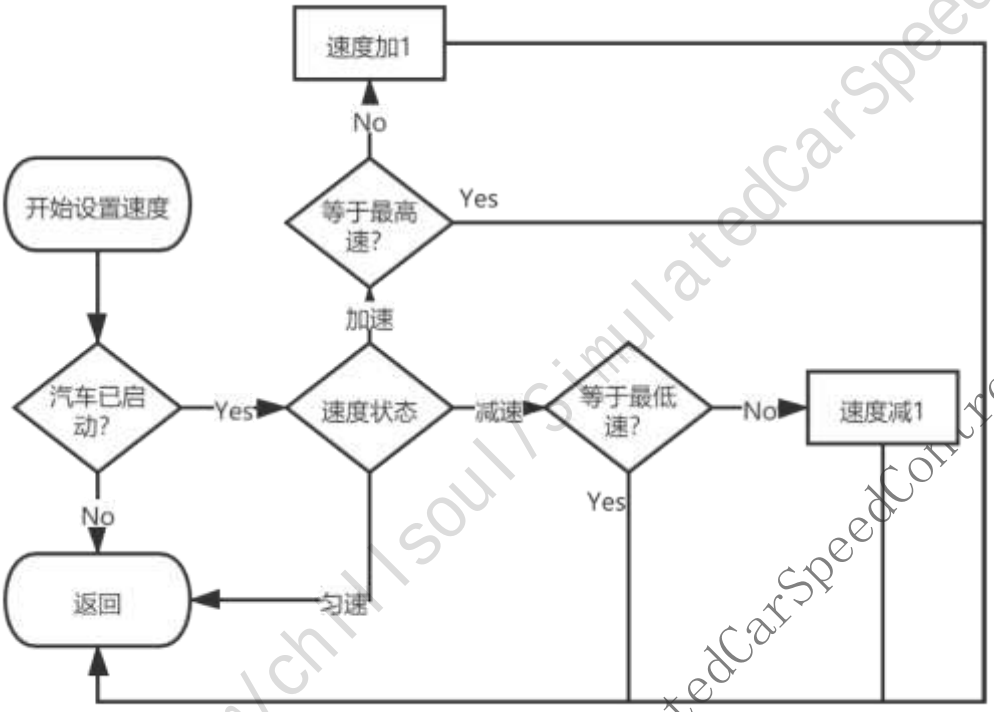

图 5.3.1 速度设置子程序流程

5.4 数码管、LED 灯驱动子程序

最简单的显示设备是发光二极管（LED），而本次设计不仅用到了最简单的 LED 灯，还使用了由七段发光二极管组成的 LED 数码管。LED 数码管是一种应用很普遍的显示器件，许多微型机都用它作为输出显示。

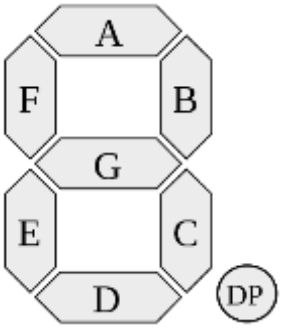

图 5.4.1 七段数码管

LED 数码管的主要部分是 7 段发光管，如图 5.4.1 所示。这 7 段发光管顺时针分别称为 A、B、C、D、E、F、G，部分产品还附带一个小数点 DP，通过这七个发光段的不同组合，数码管可以显示 0~9 和 A~F 共 16 个字母数字，从而实现十六进制数的显示。当然，7 段数码管也可以显示个别特殊字符，如“-”号，字母“P”等。LED 数码管的所有状态见图 5.4.2。

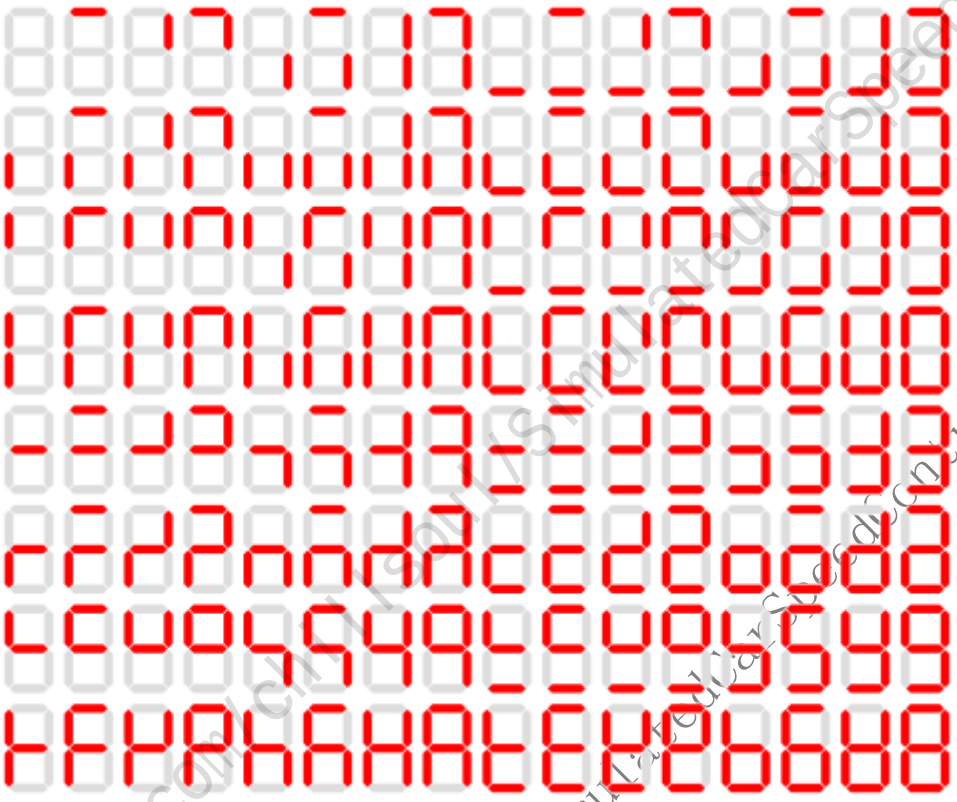

图 5.4.2 七段数码管的所有状态

LED 数码管有共阳极和共阴极两种结构。如为共阳极结构，则共用的阳极应接高电平为有效，各段则输入低电平有效。如为共阴极结构，共用的阴极必须接低电平，而各段处于高电平时便发光。例如，当 A、B、G、E、D 为高电平时相应段发光,而其他段为低电平不发光，则显示数字“2”。

由于发光二极管发光时，通过的平均电流为 10~20mA，而通常的输出锁存器不能提供这么大的电流，所以 LBD 各段必须接驱动电路。例如，对于共阴极数码管，阴极接地，则阳极要加驱动电路。驱动电路可由三极管构成，也可以采用小规模集成电路。

为了将一位十六进制数（4 位二进制数）在一个 LED 数码管上显示出来，就需要将一位十六进制数译为 LED 的 7 位显示代码。一种方法是采用专用的带驱动器的 LED 段译码器，实现硬件译码。另一种常用的方法是软件译码。在程序设计时，将 0~F 这 16 个数字对应的显示代码组成一个表。例如，用共阴极数码管如图 10-17e 所示连接，则 O 的显示代码为 3FH，1 的显示代码为 06H,……，并在表中按顺序排列：

```
LEDTABLE    DB  3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH ;段码
```

于是，要显示的数字可以很方便地通过 8088 的换码指令译码为该数字对应的显示代码。

但在实际使用时，往往要用几个数码管实现多位显示。这时，如果每一个数码管占用一个独立的输出端口那么将上用太多的通道，而且，驱动电路的数目也很多。所以，要从硬件和软件两方面想办法节省硬件电路。

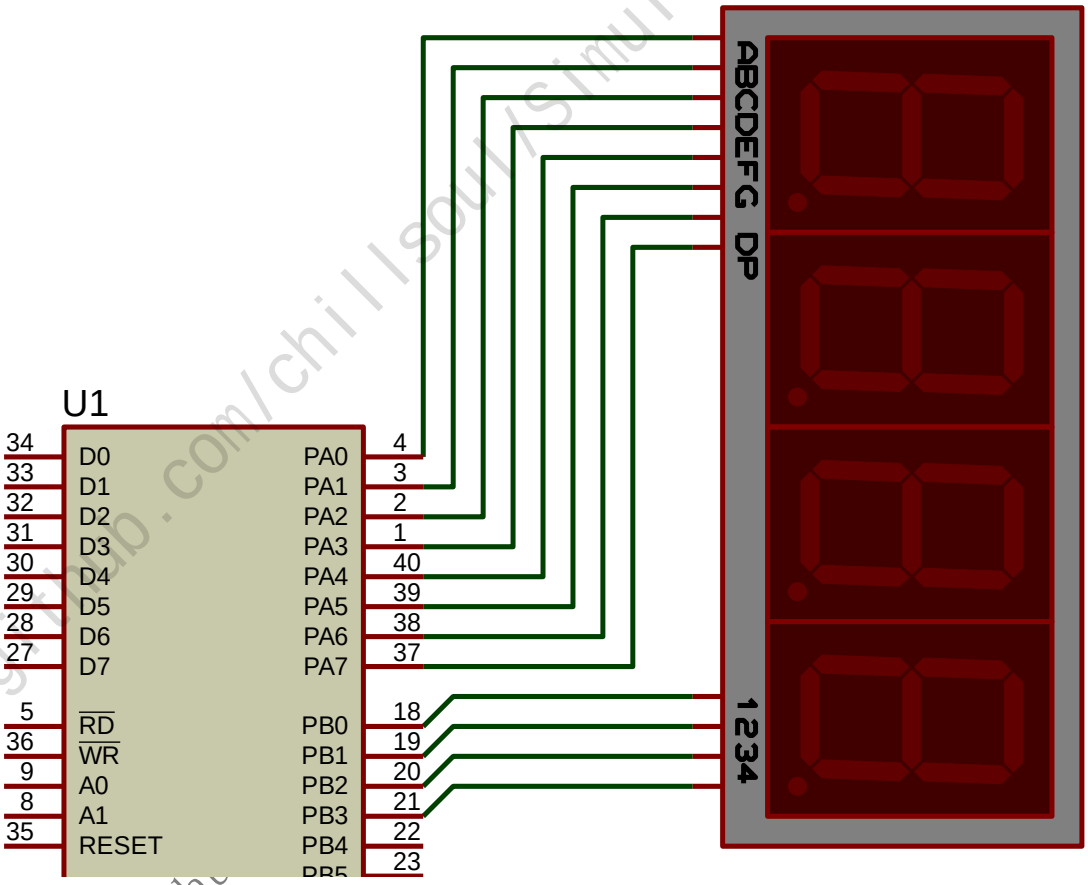

图 5.4.3 多位显示接线

图 5.4.3 是我本次课程设计所采用的多位 LED 数码管显示电路示意图。在这种方案中，硬件上用公用的驱动电路来驱动各数码管；软件上用扫描方法实现数码显示，从图中可以看到，12 位端口就可以控制 4 位数码管的显示（本次课设只用到 3 个数码管，11 位输出端口）。

由上所述，只要 CPU 通过段控制端口 A 送出段显示代码，然后通过位控制端口 B 送出位显示代码，指定的数码管便显示相应的数字。如果 CPU 顺序地输出段码和位码，依次让每个数码管显示数字，并不断地重复显示，利用眼睛的视觉惯性,当重复频率达到一定程度,从数码管上便可见到相当稳定的数字显示。显而易见，重复频率越高，每位数码管延时显示的时间越长，数字显示得就越稳定，

显示亮度也就越高。通过控制重复频率和延时时间就可以得到各种显示效果。

在这种节省硬件的多位显示电路中，往往要用软件完成段译码，CPU 需要不断重复扫描每个数码管。为此，程序设计时可以开辟一个数码缓冲区,存放要显示的数字，第一个数字在最左边的数码管显示，下一个数字送到左边第二个数码管显示，依次类推。另外，还需要建立一个显示代码表，从前向后依次存放 0~F 对应的 7 段显示代码。

```
    LEDTABLE    DB  3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH ;段码
    buffer_bin  DB  0,0,0 ;LED 输出缓冲区，个十百位
    buffer_hex  DW  0 ;LED 输出缓冲区，十六位
    LED_bit     DB  ? ;位码缓存
```

下面是本次课程设计中控制三位数码管依次显示、以及三颗 LED 灯显示档位的子程序。只要按照一定频率重复调用它，就可以获得稳定的显示效果。

```
;显示数码管
lightLED PROC
    .... ;寄存器入栈
    ;软件延时实现单端复用
    MOV DI,OFFSET buffer_bin
    MOV CH,0
    MOV CL,abs_a
LEDLOOP1:
    MOV BH,02
LEDLOOP2:
    MOV [LED_bit],BH ;保存位码以节约 BH 寄存器
    PUSH DI
    MOV BH,0
    MOV BL,LED_bit
    ADD DI,BX
    MOV BL,[DI] ;BL 为要显示的数字
    POP DI
    MOV BH,0 ;清空 BH 以便加法运算
    MOV SI,OFFSET LEDTABLE
    ADD SI,BX ;SI 为要输出数字的段码地址
    MOV AL,BYTE PTR [SI] ;AL 为要输出的段码
    MOV DX,IO8255_A
    OUT DX,AL
    PUSH CX
    MOV BH,1
    MOV CL,LED_bit
    SHL BH,CL
```

```
    MOV AL,BH ;使对应数码管亮
    POP CX
    PUSH CX
    CMP global_state,0
    JE SKIPSTATELED
    MOV CH,0
    MOV CL,global_state
    MOV AH,100000B
STATELED:
    ADD AL,AH
    SHL AH,1
    LOOP STATELED
SKIPSTATELED:
    POP CX
    MOV DX,IO8255_B
    OUT DX,AL ;B 口输出位码
    PUSH CX
    MOV CX,100
DELAY:
    LOOP DELAY ;延时
    POP CX
    MOV AL,00H
    OUT DX,AL
    DEC BYTE PTR [LED_bit]
    MOV BH,LED_bit
    JNS LEDLOOP2 ;循环复用驱动多个数码管
    LOOP LEDLOOP1 ;循环延时后再增速度
    .... ;恢复寄存器
    RET
lightLED ENDP
```

5.5 LED 缓冲区同步子程序

在速度设置子程序中，倘若直接使用段码缓冲区（三个二进制字节数）进行速度控制，速度的进位之间将变得非常麻烦，一方面速度应该显示为 BCD 码，而 BCD 码指令最多只对字范围内加法友好，故又设置了一个二进制字缓冲区，直接表示汽车速度（0~65535），在速度设置中只对该字缓冲区进行 BCD 码加减操作，调整后调用子程序同步到二进制缓冲区中。子程序代码如下：

```
;LED 缓冲区同步
bufferSync PROC
    ;寄存器入栈
    PUSH AX
```

```
    PUSH DX
    MOV AX,[buffer_hex]
    MOV DL,AL
    AND DL,0FH ;取低四位（个位）
    MOV buffer_bin[0],DL
    MOV DL,AL
    AND DL,0F0H ;取高四位（十位）
    PUSH CX
    MOV CL,4
    SHR DL,CL
    POP CX
    MOV buffer_bin[1],DL
    MOV DL,AH
    AND DL,0FH ;取第四位（百位）
    MOV buffer_bin[2],DL
    ;恢复寄存器
    POP DX
    POP AX
    RET
bufferSync ENDP
```

6 调试分析

在课程设计每个模块的设计阶段均遇到了不同程度 Bug，所幸最终完美完成课程设计题目需求，并在最后添加了“0”键匀速功能，下面列出两个最为困扰问题的调试分析。

6.1 键盘识别

在键盘扫描模块中，先是参照课本反转法自行写了一遍，但始终无法正确识别字符，后发现 TPC-ZK-II 集成开发环境中有 4*4 反转法识别键盘按键的演示实验，检查其代码后发现自己的思路是正确的，遂使用集成开发环境自带的 Debug 功能，下断点后单步调试，发现是键盘段码表不正确，子程序读取的行列值高低位的顺序写反，改之，完成键盘识别。

6.2 BCD 码运算

在数码管显示完成后，调试速度设置模块时，使用 AAA 进行 BCD 码字加法时遇到了无法进位的问题，查阅多方资料后在课本 P37 页发现，AAA 是实现

非压缩 BCD 码加法，而 DAA 才是压缩 BCD 码加法的运算指令，在 5.5 中提到了，速度设置模块操作的是字缓冲区，该字是二进制的汽车速度，考虑在十六进制下，正好是每 4 位表示一个 BCD 码（压缩 BCD 码），因此此处应该使用 DAA 操作压缩 BCD 码加法。

7 测试结果

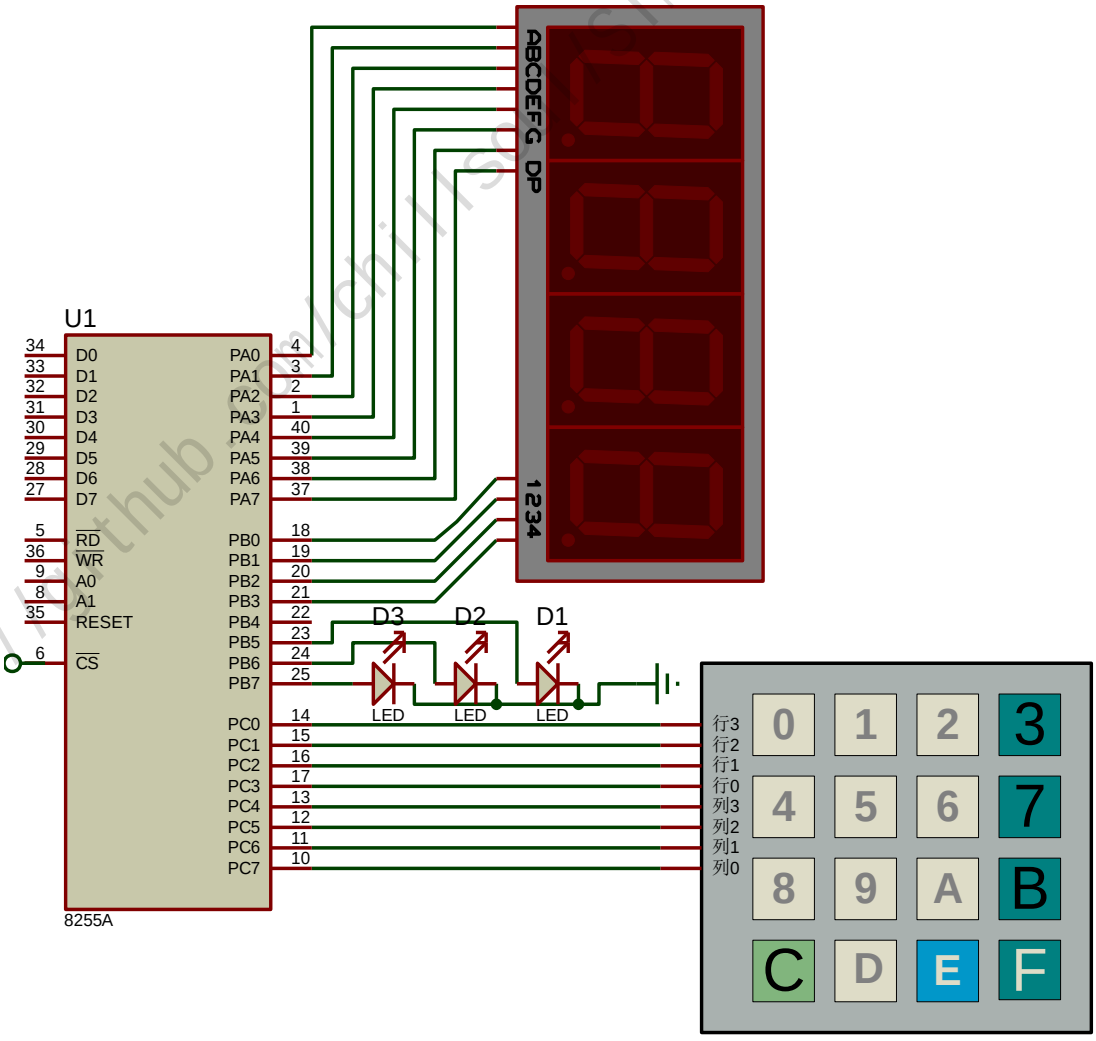

图 7.1 设计结果电路图

图 7.1 为本次课程设计中的最终电路图（模拟），在清华科教厂 TPC-ZK-II 实验箱上接线测试并答辩演示启动、加速、提档、减速、匀速、降档和急停所有功能均成功，过程顺利。

课程设计有待改进的地方在于，汽车速度控制使用了软件定时，可以使用 8259A 中断控制配合计时器 8254 实现硬件定时作为进一步改进。