第六章 通用 I/O 接口基本结构与输出应用

从本章开始，将从 AVR 单片机的基本功能单元入手，讲解其各个外围功能部件的基本组成和特性，以及它们的应用。

ATmega16 芯片有 PORTA、PORTB、PORTC、PORTD（简称 PA、PB、PC、PD）4 组 8 位，共 32 路通用 I/O 接口，分别对应于芯片上 32 根 I/O 引脚。所有这些 I/O 口都是双（有的为 3）功能复用的。其中第一功能均作为数字通用 I/O 接口使用，而复用功能则分别用于中断、时钟/计数器、USRAT、I2C 和 SPI 串行通信、模拟比较、捕捉等应用。这些 I/O 口同外围电路的有机组合，构成各式各样的单片机嵌入式系统的前向、后向通道接口，人机交互接口和数据通信接口，形成和实现了千变万化的应用。

6.1 通用 I/O 口的基本结构与特性

6.1.1 I/O 口的基本结构

图 6-1 为通用 I/O 口的基本结构示意图。从图中可以看出，每组 I/O 口配备三个 8 位寄存器，它们分别是方向控制寄存器 DDRx，数据寄存器 PORTx，和输入引脚寄存器 PINx（x=A\B\C\D）。I/O 口的工作方式和表现特征由这 3 个 I/O 口寄存器控制。

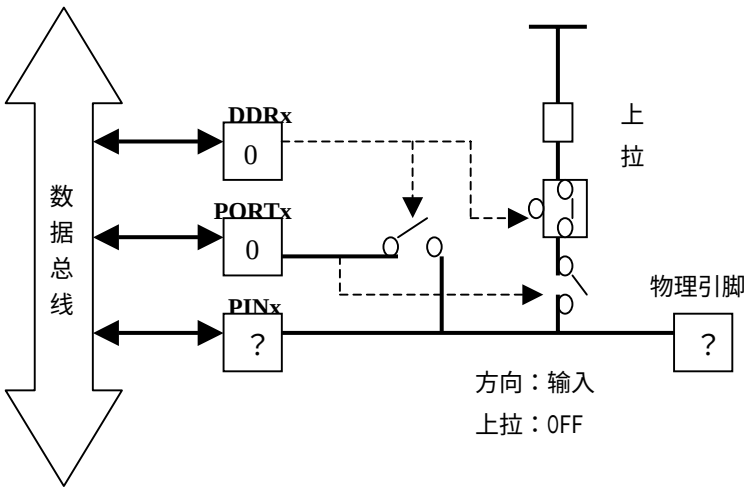

图 6-1 通用 I/O 口结构示意图

方向控制寄存器 DDRx 用于控制 I/O 口的输入输出方向，即控制 I/O 口的工作方式为输出方式还是输入方式。

当 DDRx=1 时，I/O 口处于输出工作方式。此时数据寄存器 PORTx 中的数据通过一个推挽电路输出到外部引脚（图 6-2）。AVR 的输出采用推挽电路提高了 I/O 口的输出能力，当 PORTx=1 时，I/O 引脚呈现高电平，同时可提供输出 20mA 的电流；而当 PORTx=0 时，I/O 引脚呈现低电平，同时可吸纳 20mA 电流。因此，AVR 的 I/O 在输出方式下提供了比较大的驱动能力，可以直接倾动 LED 等小功率外围器件。

当 DDRx=0 时，I/O 处于输入工作方式。此时引脚寄存器 PINx 中的数据就是外部引脚的实际电平，通过读 I/O 指令可将物理引脚的真实数据读入 MCU。此外，当 I/O 口定义为输入

时（DDRx=0），通过 PORTx 的控制，可使用或不使用内部的上拉电阻（图 6-3）。

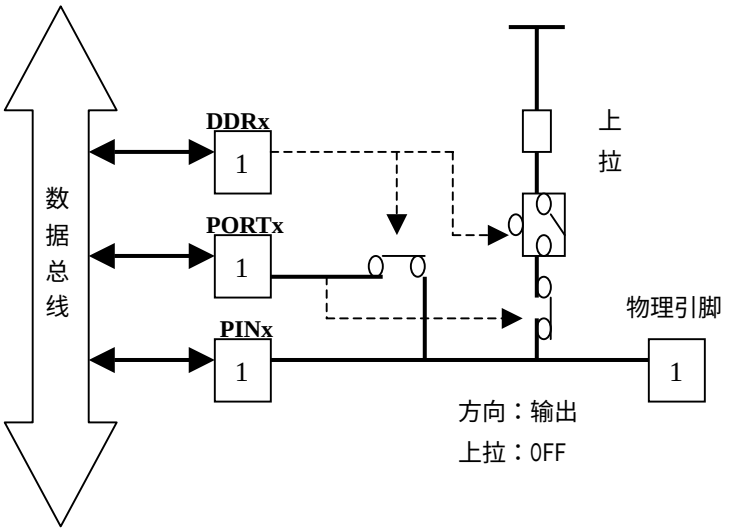

图 6-2　通用 I/O 口输出工作方式示意图

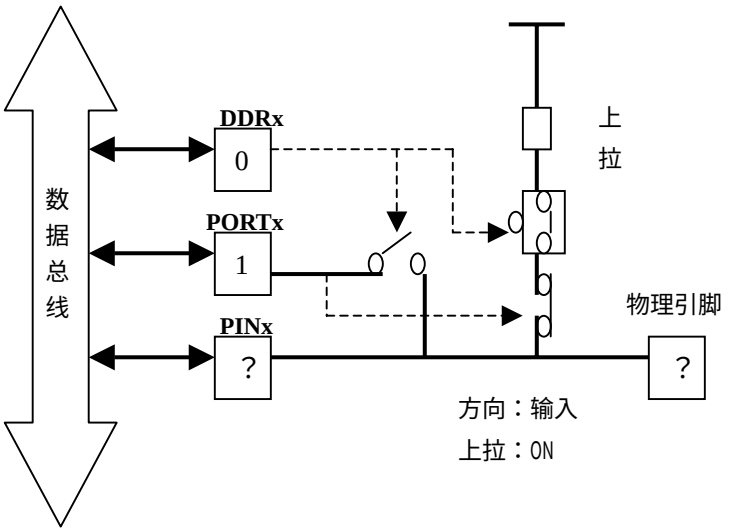

图 6-3　通用 I/O 口输入工作方式示意图（带内部上拉）

表 6.1 是 AVR 通用 I/O 端口的引脚配置情况。

表 6.1 I/O 口引脚配置表

DDRXn	PORTXn	PUD	I/O 方式	内部上拉电阻	引脚状态说明
0	0	X	输入	无效	三态（高阻）
0	1	0	输入	有效	外部引脚拉低时输出电流(uA)
0	1	1	输入	无效	三态（高阻）
1	0	X	输出	无效	推挽 0 输出，吸收电流(20mA)
1	1	X	输出	无效	推挽 1 输出，输出电流(20mA)

表中的 PUD 为寄存器 SFIOR 中的一位，它的作用相当 AVR 全部 I/O 口内部上拉电阻的总开关。当 PUD=1 时，AVR 所有 I/O 内部上拉电阻都不起作用（内部不上拉）；而 PUD=0 时，各个 I/O 口内部上拉电阻取决于 DDRXn 的设置。

AVR 通用 I/O 端口的主要特点为：

✓ 双向可独立位控的 I/O 口

ATmega16 的 PA、PB、PC、PD 四个端口都是 8 位双向 I/O 口，每一位引脚都可以单独的进行定义，相互不受影响。如用户可以在定义 PA 口第 0、2、3、4、5、6 位用于输入的同时定义第 1、7 位用于输出，互不影响。

✓ Push-Pull 大电流驱动（最大 40mA）

每个 I/O 口输出方式均采用推挽式缓冲器输出，提供大电流的驱动，可以输出（吸收）20mA 的电流，因而能直接驱动 LED 显示器。

✓ 可控制的上拉电阻

每一位引脚内部都有独立的，可通过编程设置的，设定为上拉有效或无效的内部上拉电阻。当 I/O 口被用于输入方式，且内部上拉电阻被激活（有效）时，如果外部引脚被拉低，则构成电流源输出电流（uA 量级）。

✓ DDRx 可控的方向寄存器。

AVR 的 I/O 端口结构同其它类型单片机的明显区别是，AVR 采用 3 个寄存器来控制 I/O 端口。一般单片机的 I/O 仅有数据寄存器和控制寄存器，而 AVR 还多了一个方向控制器，用于控制 I/O 的输入输出方向。由于输入寄存器 PINx 实际不是一个寄存器，而是一个可选通的三态缓冲器，外部引脚通过该三态缓冲器与 MCU 的内部总线连接，因此，读 PINx 时是读取外部引脚上的真实和实际逻辑值，实现了外部信号的同步输入。这种结构的 I/O 端口，具备了真正的读-修改-写（Read-Modify-Write）特性。

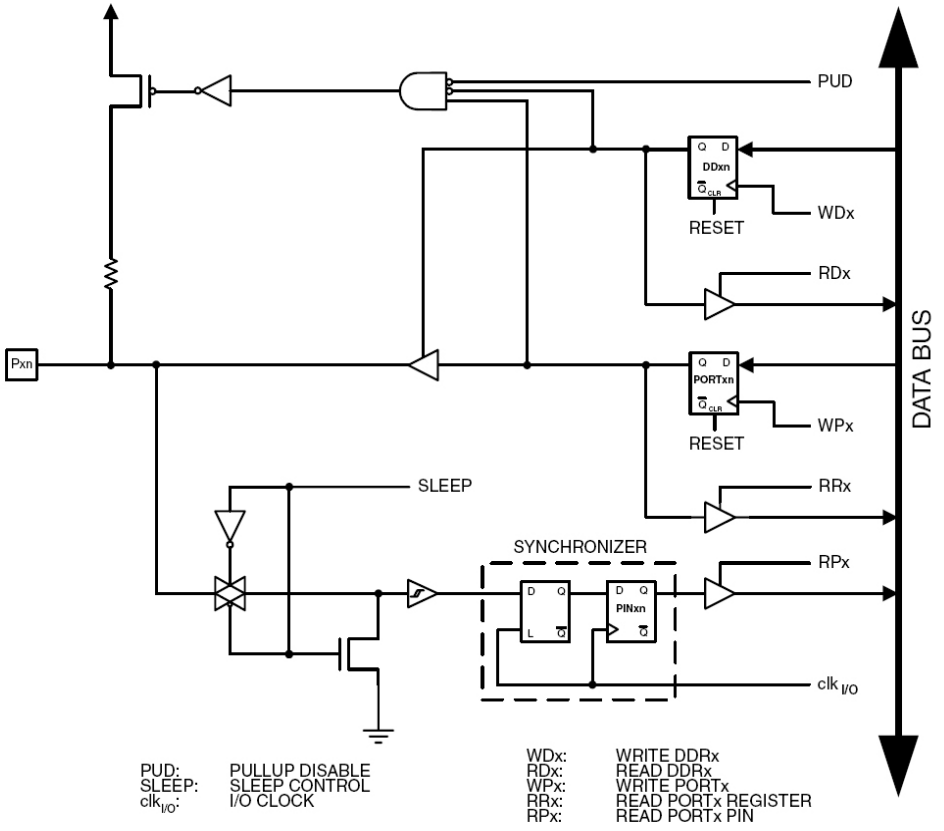

图 6-4　通用 I/O 口逻辑功能示意图

图 6-4 为 AVR 一个（位）通用 I/O 口的逻辑功能图。右上面的两个 D 触发器为方向控制寄存器和数据寄存器。

1） 使用 AVR 的 I/O 口，首先要正确设置其工作方式，确定其工作在输入方式还是输入方式。
2） 当 I/O 工作在输入方式，要读取外部引脚上的电平时，应读取 PINxn 的值，而不是 PORTxn 的值。
3） 当 I/O 工作在输入方式，要根据实际情况使用或不使用内部的上拉电阻。
4） **一旦将 I/O 口的工作方式由输出设置成输入方式后，必须等待一个时钟周期后才能正确的读到外部引脚 PINxn 的值。**

上面的第 4 点是由于在 PINxn 和 AVR 内部数据总线之间有一个同步锁存器（图 6-4 中的 SYNCHRONIZER）电路，使用该电路避免了当系统时钟变化的短时间内外部引脚电平也同时变化而造成的信号不稳定的现象，但它有产生大约一个时钟周期（0.5～1.5）的时延。

6.1.2 I/O 端口寄存器

ATmega16 的 4 个 8 位的端口都有各自对应的 3 个 I/O 端口寄存器，它们占用了 I/O 空间的 12 个地址（见表 6.2）。

表 6.2 ATmega16 I/O 寄存器地址表

名 称	I/O 空间地址	RAM 空间地址	作 用
PORTA	$1B	0x003B	A 口数据寄存器
DDRA	$1A	0x003A	A 口方向寄存器
PINA	$19	0x0039	A 口输入引脚寄存器
PORTB	$18	0x0038	B 口数据寄存器
DDRB	$17	0x0037	B 口方向寄存器
PINB	$16	0x0036	B 口输入引脚寄存器
PORTC	$15	0x0035	C 口数据寄存器
DDRC	$14	0x0034	C 口方向寄存器
PINC	$13	0x0033	C 口输入引脚寄存器
PORTD	$12	0x0032	D 口数据寄存器
DDRD	$11	0x0031	D 口方向寄存器
PIND	$10	0x0030	D 口输入引脚寄存器

下面是 PA 口寄存器—PORTA、DDRA、PINA 各个位的具体定义，以及其是否可以通过指令读写操作和 RESET 复位后的初始值。其它 3 个口的寄存器的情况与 PA 口相同，只是地址不一样。

位	7	6	5	4	3	2	1	0	
$1B（$003B）	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

位	7	6	5	4	3	2	1	0	
$1A（$003A）	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
复位值	0	0	0	0	0	0	0	0	

位	7	6	5	4	3	2	1	0	
$19 ($0039)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
读/写	R	R	R	R	R	R	R	R	
复位值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

1） 正确使用 AVR 的 I/O 口要注意：先正确设置 DDRx 方向寄存器，再进行 I/O 口的读写操作。
2） AVR 的 I/O 口复位后的初始状态全部为输入工作方式，内部上拉电阻无效。所以，外部引脚呈现三态高阻输入状态。
3） 因此，用户程序需要首先对要使用的 I/O 口进行初始化设置，根据实际需要设定使用 I/O 口的工作方式（输出还是输入），当设定为输入方式时，还要考虑是否使用内部的上拉电阻。
4） 在硬件电路设计时，如能利用 AVR 内部 I/O 口的上拉电阻，可以节省外部的上拉电阻。

4.1.3 通用数字 I/O 口的设置与编程

在将AVR 的 I/O 口作为通用数字口使用时,要先根据系统的硬件设计情况,设定各个 I/O 口的工作方式：输入或输出工作方式，既先正确设置 DDRx 方向寄存器，再进行 I/O 口的读写操作。如将 I/O 口定义为数字输入口时，还应注意是否需要将该口内部的上拉电阻设置为有效，在设计电路时，如能利用 AVR 内部 I/O 口的上拉电阻，可以节省外部的上拉电阻。

AVR 汇编指令系统中，直接对 I/O 寄存器的操作指令有以下 3 类：

1） IN/OUT

IN/OUT 指令实现了 32 个通用寄存器与 I/O 寄存器之间的数据交换，格式为：

```
IN Rd，A          ；从 I/O 寄存器 A 读数剧到通用寄存器 Rd
OUT A，Rr         ；通用寄存器 Rr 数据送 I/O 寄存器 A
```

2）SBI/CBI

SBI/CBI 指令实现了对 I/O 寄存器（地址空间在 0x00-0x31）中指定位的置 1 或清 0，格式为：

```
SBI A，b          ；将 I/O 寄存器 A 的第 b 位置 1
CBI A，b          ；将 I/O 寄存器 A 的第 b 位清 0
```

3）SBIC/SBIS

SBIC/SBIS 指令为转移类指令，它根据 I/O 寄存器（地址空间在 0x00-0x31）的指定位的数值实现跳行转移（跳过后面紧接的一条指令，执行后序的第二条指令），格式为：

```
SBIC A，b             ；I/O 寄存器 A 的第 b 位为 0 时，跳行执行
SBIS A，b             ；I/O 寄存器 A 的第 b 位为 1 时，跳行执行
```

ATmega16 的 4 个 8 位的端口共有 12 个 I/O 端口寄存器，它们在 AVR 的 I/O 空间的地址均在前 32 个之中，因此上面 3 类对 I/O 寄存器操作的指令都可以使用。在第五章的例程 Demo_5_1.asm 中，使用了 OUT 指令设置 PC 口的工作方式为输出，输出全 1：

```
.def temp1=r20                  ;定义寄存器R20用临时变量名temp1代表
°°°°°°°°°
ser temp1                       ;置temp1（R20）为0XFF
```

```
out ddrc,temp1                  ;定义PC口为输出
out portc,temp1                 ;PC口输出全“1”，LED不亮
```

在CVAVR中，我们可以直接使用C的语句对I/O口寄存器进行操作，如：

```
// 定义PortC口的工作方式
PORTC=0x01;                 // PC口的第0位输出“1”，LED不亮
DDRC=0x01;                  // 定义PC口的第0位为输出方式
PORTC.0 = ~PORTC.0;         // PC口第0位输出取反
```

其中PORTC.0 = 0 (或PORTC.0 = 1)是CVAVR中对C的扩展语句，它实现了对寄存器的位操作。这种语句在标准C中是没有的，该扩展更加适合编写单片机的系统程序，因为在单片机的系统程序中，是经常需要直接对位进行操作的。

更加标准的C程序可以采用以下的写法：

```
#define BIT0 0
#define BIT1 1
#define BIT2 2
#define BIT3 3
#define BIT4 4
#define BIT5 5
#define BIT6 6
#define BIT7 7
.........
PORTC = 1<<(BIT0) | 1<<(BIT3);   // PC口的第0位和第3位输出“1”，其它为“0”
```

这里，1<<(BIT0)表示逻辑1左移0位，结果为0b00000001;而1<<(BIT3) 表示逻辑1左移3位，结果为0b00001000。0b00000001在同0b00001000相与，结果为0b00001001。以上的逻辑运算不产生具体的操作指令，是由CVAVR在编译时运算完成，得到结果，最后只是产生将结果赋值到PORTC寄存器的操作指令。

这种表示方式，比直接赋值0b00001001更容易理解程序的作用，如在下面的有关AVR的USART串口的程序中大量使用了这样的描述方式。

```
#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)
.........
char status;
status = UCSRA;
if (( status & ( FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN )) == 0 )
```

```
{……}                    // 接收数据无错误处理过程
else
{……}                    //接收数据产生错误处理过程
```

程序中的 UCSRA 为 ATmega16 的串行接口 USART 的状态寄存器，UPE 是 UCSRA 的第 2 位，当 UPE 为 1 时表示接收到的数据产生了校验错误。程序中采用了定义语句，定义 PARITY_ERROR 为（1 << UPE），实际就是 0b00000100。因此一旦 USART 的值为 PARITY_ERROR 时，表示接受的数据产生了校验错误，使程序的阅读非常明了。

这样的程序编写方式，在 AVR 的汇编中也是可以使用的。

6.2 通用 I/O 口的输出应用

6.2.1 通用 I/O 输出设计要点

将通用 I/O 口定义为输出工作方式，通过设置该口的数据寄存器 PORTx，就可以控制对应 I/O 口外围引脚的输出逻辑电平，输出“0”或“1”。这样我们就可以通过程序控制 I/O 口，输出各种类型的逻辑信号，如方波脉冲，或控制外围电路执行各种动作。

在应用 I/O 口输出时，在系统的软硬件设计上应注意的问题有：

- ✓ 输出电平的转换和匹配。如一般 AVR 系统的工作电源为 5 伏（手持系统往往采用 1.5v－3v 电源），所以 I/O 的输出电平为 5v。当连接的外围器件和电路采用 3v、9v、12v、15v 等与 5v 不同的电源时，应考虑输出电平转换电路。
- ✓ 输出电流的驱动能力。AVR 的 I/O 口输出为“1”时，可以提供 20mA 左右的驱动电流。输出为“0”时，可以吸收 20mA 左右的灌电流（最大为 40mA）。 当连接的外围器件和电路需要大电流驱动或有大电流灌入时，应考虑使用功率驱动电路。
- ✓ 输出电平转换的延时。AVR 是一款高速单片机，当系统时钟为 4M 时，执行一条指令的时间为 0.25us，这意味着将一个 I/O 引脚置“1”，再置“0”仅需要 0.25us，既输出一个脉宽为 0.25us 高电平脉冲。在一些应用中，往往需要较长时间的高电平脉冲驱动，如步进马达的驱动，动态 LED 数码显示器的扫描驱动等，因此在软件设计中要考虑转换时间延时。对于不需要精确延时的应用，可采用软件延时，编写软件延时的子程序，如果要求精确延时，要使用 AVR 内部的定时器。

6.2.2 LED 发光二极管的控制

LED 发光二极管是一种经常使用的外围器件，用于显示系统的工作状态，报警提示等，用大量的发光二极管组成方阵，就是构成一个 LED 电子显示屏，可以显示汉字和各种图形，如体育场馆中的大型显示屏。下面设计一个带有一排 8 个发光二极管的简易彩灯控制系统。

例6.1 简易彩灯控制系统

1） 硬件电路设计：

发光二极管一般为砷化镓半导体二极管，其电路入图 6-5 所示。当电压 U1 大于 U2 约 1V 以上时，二极管导通发光。当导通电流大于 5mA 时，人的眼睛就可以明显地观察到二极管的发光，导通电流越大，亮度越高。一般导通电流不要超过 10mA，否则将导致二极管的烧毁或 I/O 引脚的烧毁。因此在 LED 二极管电路中要串接一个限流电阻，阻值在 100～500Ω之间，调节阻值的大小可以控制发光二极管的发光亮度。导通电流与限流电阻之间的关系由下面的计算公式确定：

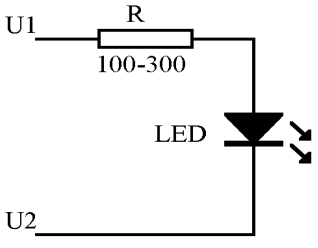

图 6-5 LED 电路

$$I = \frac{U1 - U2 - Vled}{R}$$

式中，Vled 为 LED 的导通电压。

由于 AVR 的 I/O 口输出“ 0 ”时，可以吸收最大 40mA 的电流，因此采用控制发光二极管负极的设计比较好。8 个 LED 发光二极管控制系统的硬件电路见图 6-6。

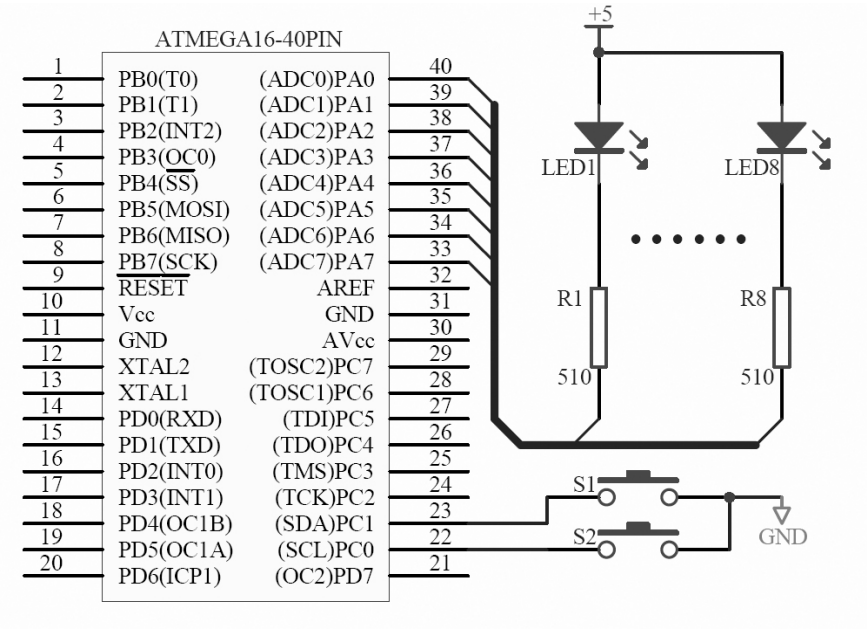

图 6-6 8 路 LED 发光二极管控制电路

在图 6-6 中，ATmega16 的 PA 口工作在输出方式下，8 个引脚分别控制 8 个发光二极管。当 I/O 口输出“ 0 ”时 LED 导通发光，输出“ 1 ”时 LED 截止熄灭。

2）软件设计

下面给出一个简单的控制程序，其完成的功能是 8 个 LED 逐一循环发光 1 秒，构成“ 走马灯 ”。程序非常简单，请读者自己分析。

```
/*********************************************
file name           : demo_6_1.c
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 4.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*********************************************/

#include <mega16.h>
#include <delay.h>

void main(void)
{
    char position = 0;              // position 为控制位的位置
```

```
    PORTA=0xFF;                    // PA 口输出全 1，LED 全灭
    DDRA=0xFF;                     // PA 口工作为输出方式

    while (1)
    {
        PORTA = ~(1<<position);     //
        if (++position >= 8) position = 0;
        delay_ms(1000);
    };
}
```

3) 思考与实践

- ✓ 调整程序中的 delay_ms()，延时时间为 1ms，彩灯闪亮有何变化？为什么？
- ✓ 计算并验证当延时时间小于多少毫秒时，“ 走马灯 ”的效果变成“ 全亮 ”？给出计算方法。
- ✓ 设计一个 4 种闪烁方式交替循环的彩灯，闪烁方式见图 6-7。

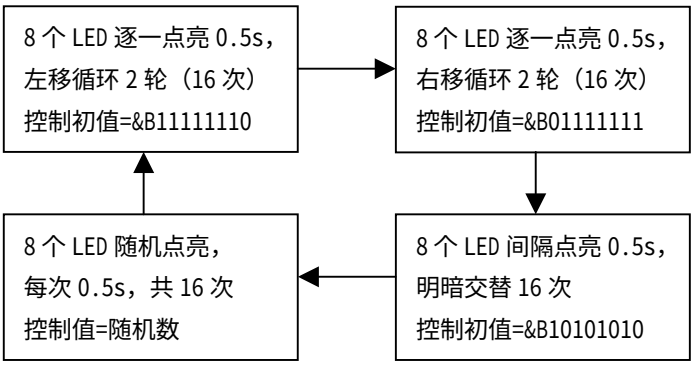

图 6-7 4 种不同控制方式的转换

提示：在 CVAVR 中，提供 int rand (void)和 void srand(int seed)函数，请参考 CVAVR 手册，尝试使用这两个函数。

6.2.3 继电器控制

在工业控制以及许多场合中，嵌入式系统要驱动一些继电器和电磁开关，用于控制马达的开启和关闭，阀门的开启和关闭等。由于 AVR 的 I/O 不能提供大的驱动电流，因此在外围硬件电路中要考虑使用功率驱动电路。

例6.2 控制恒温箱的加热的硬件电路设

恒温箱的加热源采用 500W 电炉，电炉的工作电压 220v，电流 2.3A。选用 HG4200 继电器，开关负载能力为 5A/AC220V，继电器吸合线圈的工作电压 5v，功耗 0.36W，计算得吸合电流为 0.36/5 = 72mA。设计控制电路如图 6-8。I/O 引脚输出“ 1 ”时，三极管导通，继电器吸合，电炉开始加热。I/O 引脚输出“ 0 ”时，三极管截止，继电器释放，加热停止。

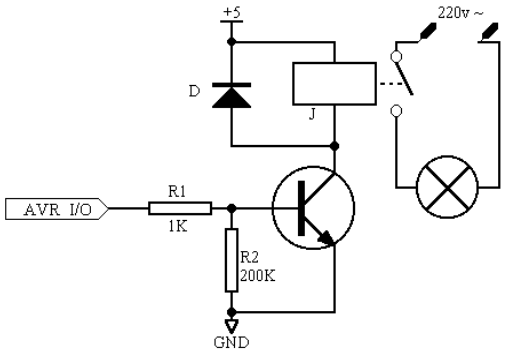

图 6-8 继电器控制电路

图中的三极管应采用中功率管，导通电流大于 300mA。电阻 R1 的作用是限制从 I/O 流出的电流太大，保护 I/O 端口，称为限流电阻。注意：三极管集电极的负载继电器吸合线圈在三极管截止时会产生一个很高的反峰电压，在吸合线圈两端并接一个二极管 D，其用途是释放反峰电压，保护三极管和 I/O 口不会被反峰电压击穿，提高系统的可靠性。设计中还要考虑系统在上电时的状态。由于 AVR 在上电时，DDRx 和 PORTx 的值均初始化为“0”，I/O 引脚呈高阻输入方式，因此电阻 R2 的作用是确保三极管的集电极电位在上电时为“0”电平，三极管截止，保证了加热电炉控制系统上电时不会误动作。

在工业控制中，尤其应认真考虑系统上电初始化时以及发生故障时 I/O 口的状态，应在硬件和软件设计中仔细考虑，否则会产生误动作，造成严重的事故！

在驱动电感性负载时，在硬件上要考虑采取对反峰电压的吸收和隔离，防止对控制系统的干扰和破坏。

6.2.4 步进电机控制

步进电机在自动仪表、自动控制、机器人、自动生产流水线等领域的应用相当广泛，如在打印机、磁盘驱动器、扫描仪中都有步进电机的身影。关于步进电机工作原理请参考有关资料，本节介绍一种普通微型单极 3 相步进电机的控制。

单极 3 相步进电机有三个磁激励相，分别用 A、B、C 表示，每相有一个磁激线圈。通过控制三个磁激线圈电流通断以及通断的先后时间顺序和通断频率来改变转速和旋转方向，图 6-9 是单极 3 相步进电机的原理图。

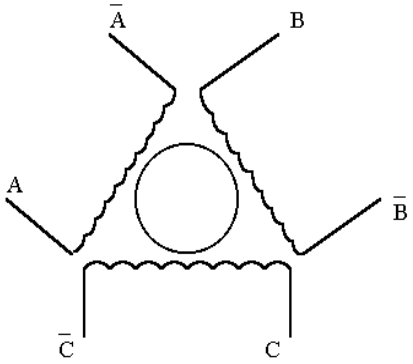

图 6-9 单极 3 相步进电机原理图

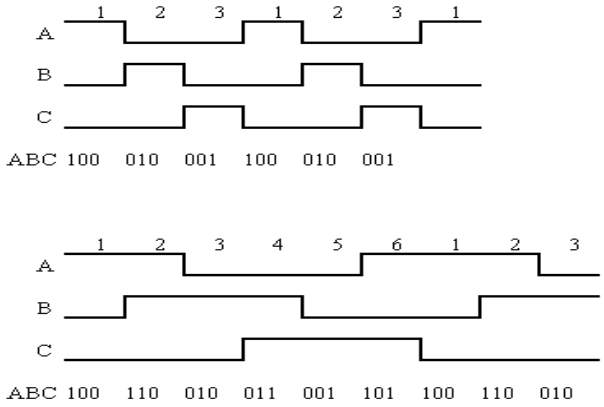

图 6-10 3 相步进电机控制时序图

单极 3 相步进电机有 3 相 3 拍和 3 相 6 拍两种驱动方式，图 6-10 给出它们的控制时序图。3 相 3 拍就是 A、B、C 三相分别通电，正转为 A－B－C－A－B－C，反转为 A－C－B－A－C－B，每拍转动 3°。3 相 6 拍中有三拍是两相同时通电，正转为 A－AB－B－BC－C－CA，反转为 A－AC－C－CB－B－BA，每拍转动 1.5°。

例 6.3 单极 3 相步进电机控制系统

1） 硬件电路设计：

本例使用的3相步进电机，型号为45BC340C，步距角1.5°/3°，相电压12vDC，相电流0.4A，空载启动频率500Hz，控制硬件电路原理图见图6-11。图中采用一片7位达林顿驱动芯片MC1413（Darlington transistor arrays），其驱动电流为0.5A，工作电压达50v。当I/O A

输出高电平“1”时，MC1413内部对应Q0的达林顿管导通，电流从电源正极（+12V）流过步进电机A相线圈，经Q0端流入地线；输出低电平“0”时，MC1413内部对应Q0的达林顿管截止，A相线圈中无电流流过。电阻R1-R3为限流保护电阻。系统上电时，AVR的I/O引脚为高阻，电阻R4-R5将MC1413的3个控制输入端拉低，以保证步进电机在上电时不会产生误动作。

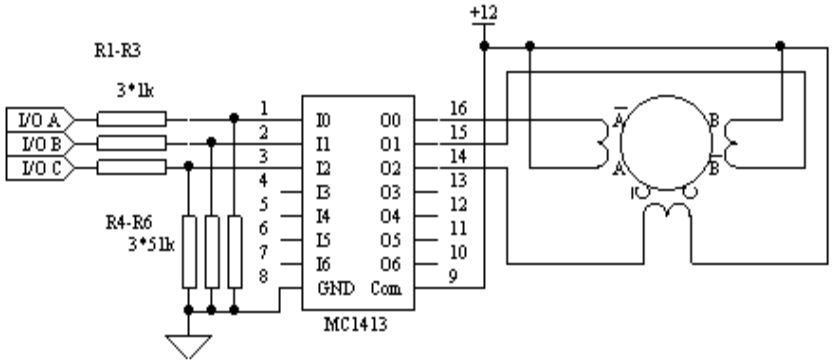

图 6-11 步进电机控制电路

2） 软件设计

```
/*********************************************
file name           : demo_6_3.c
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 4.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*********************************************/

#include <mega16.h>
#include <delay.h>

flash char step_out[6]={0x04,0x06,0x02,0x03,0x01,0x05};

void main(void)
{
    char i = 0;
    int delay = 500;

    PORTA=0x00;
    DDRA=0x07;

    while (1)
    {
        PORTA = step_out[i];
        if (++i >= 6) i = 0;
```

```
            delay_ms(delay);
        };
    }
```

3) 思考于实践

- ✓ 通过分析程序，你认为本例中是使用 AVR 的哪三个 I/O 端口控制步进电机的？采用的是 3 相 3 拍还是 3 相 6 拍的控制方式？电机的转动方向如何？
- ✓ 如果要改变电机的转动方向，仅仅改动硬件路线或只修改软件能够实现吗？各给出一个方案，并检验。
- ✓ 程序中定义的数组“ flash char step_out[6]={0x04,0x06,0x02,0x03,0x01,0x05} ”的作用是什么？采用如下的定义“ char step_out[6]={0x04,0x06,0x02,0x03,0x01,0x05} ”可以么？两中定义有何区别？在本例中使用哪种定义比较好，为什么？
- ✓ 程序中变量 Delay 的作用是什么？将 Delay 的数值调大或减小，对电机的转动有何影响？请仔细分析并验证（最好在实际系统上测试）。
- ✓ 本例使用的三相步进电机有一个指标参数“空载启动频率 500Hz”，该参数在软件设计中需要考虑吗？
- ✓ 如何设计软件能控制电机转动的圈数和转速？
- ✓ 设计一段步进马达的控制程序（马达拖动打印机的打印头），它能打印一行字，然后返回起始位置。打印过程为：慢速正转 20 圈（2 圈/秒），高速反转 20 圈（4 圈/秒）。

6.3 LED 数码显示器的应用

LED 数码显示器是单片机嵌入式系统中经常使用的显示器件。一个“ 8 ”字型的显示模块用“ a、b、c、d、e、f、g、h ” 8 个发光二极管组合而成，如图 6-12a 所示。每个发光二极管称为一字段。LED 数码显示器有共阳极和共阴极两种结构形式，电路设计时不要混淆。

LED 数码管显示的基本控制原理同上面所述发光二极管的控制，但在具体使用时有许多不同的设计和应用电路，软件的设计也各不相同，有许多技巧和变化。

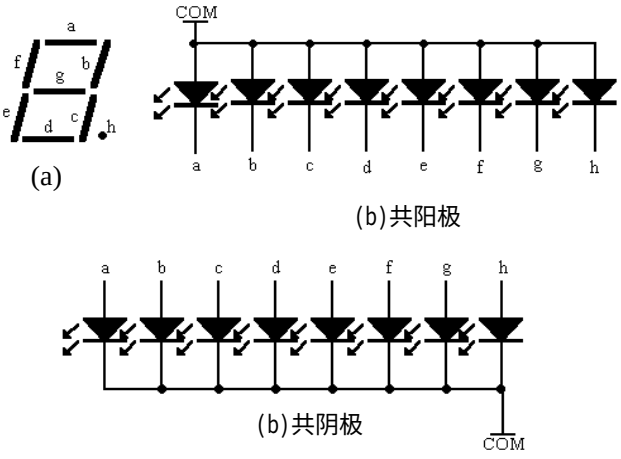

图 6-12 LED 数码显示器

6.3.1 单个 LED 数码管控制

我们以共阴极的数码管为例，先介绍如何控制一个 8 段数码管显示“ 0 ”—“ F ”16 个十六进制的数字。

例 6.4 单个 LED 数码管字符显示控制

1) 硬件电路设计：

很明显，用 AVR 的一个 I/O 口控制共阴极数码管的 8 个段位，分别置“ 1 ”或“ 0 ”，让某些段的 LED 发光，其它的熄灭，就可以显示不同的字符和图符号，硬件电路如图 6-13。

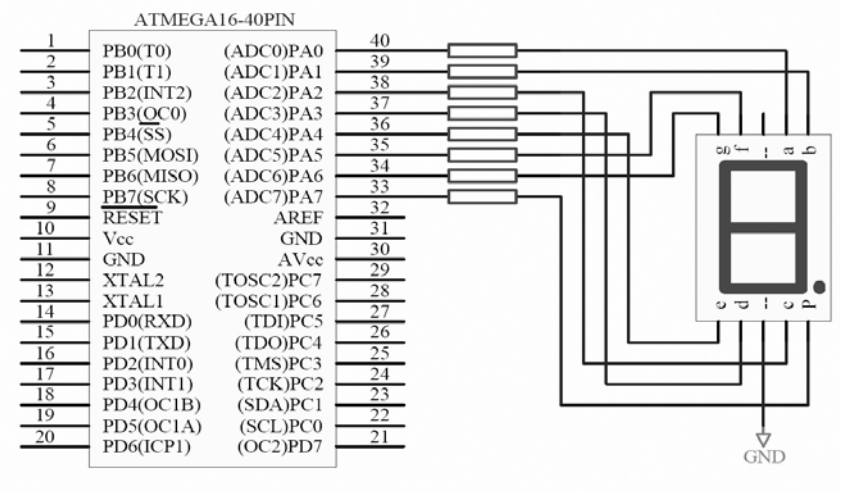

图 6-13 1 位共阴极 LED 数码显示器控制电路

2) 软件设计：

为了获得“0”—“F”16 个不同的字型符号，数码管各段所加的电平不同，因此 I/O 口输出的编码也不同。因此首先要建立一个字型与字段的编码表，见表 6.3。

表 6.3 8 段 LED 数码管字型字段编码表

显示字型	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	段码共阴极	段码共阳极
	h	g	f	E	d	c	b	a		
0	0	0	1	1	1	1	1	1	3FH	C0H
1	0	0	0	0	0	1	1	0	06H	F9H
2	0	1	0	1	1	0	1	1	5BH	A4H
3	0	1	0	0	1	1	1	1	4FH	B0H
4	0	1	1	0	0	1	1	0	66H	99H
5	0	1	1	0	1	1	0	1	6DH	92H
6	0	1	1	1	1	1	0	1	7DH	82H
7	0	0	0	0	0	1	1	1	07H	F8H
8	0	1	1	1	1	1	1	1	7FH	80H
9	0	1	1	0	1	1	1	1	6FH	90H
A	0	1	1	1	0	1	1	1	77H	88H
b	0	1	1	1	1	1	0	0	7CH	83H
C	0	0	1	1	1	0	0	1	39H	C6H
d	0	1	0	1	1	1	1	0	5EH	A1H
E	0	1	1	1	1	0	0	1	79H	86H
F	0	1	1	1	0	0	0	1	71H	8EH

注：B、D 字型为小写 b、d，以同数字 8、0 字型区别

有了字型段码对照表，就可以用软件的方式进行 8 段码的译码。如要显示字型“1”，PA 口输出值为 0x06；显示字型“A”，PA 口输出值为 0x77。

在单片机嵌入式系统软件设计中，经常要考虑二进制、十六进制、十进制、BCD 码、压缩 BCD 码、八段码、ASCII 码之间的相互转换问题。人们计数习惯采用十进制，而单片机的计算、存储则为二进制形式最方便。此外传送字符用 ASCII 码，LED 数码显示要转化成相应的 7 段码等等。因此对与各种不同数制的使用和相互转换在软件设计中尤其重要，设计使用得当，可以简化程序设计和优化程序代码。

```
/*****************************************
file name          : demo_6_4.c
Chip type          : ATmega16
Program type       : Application
Clock frequency    : 4.000000 MHz
Memory model       : Small
External SRAM size : 0
Data Stack size    : 256
*****************************************/

#include <mega16.h>
#include <delay.h>

flash char led_7[16]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
                      0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71};
bit point_on = 0;
void main(void)
{
  char i = 0;

  PORTA=0xFF;
  DDRA=0xFF;

  while (1)
   {
      for (i=0;i<=15;i++)
      {
           PORTA = led_7[i];
           if (point_on) PORTA |= 0x80;
           delay_ms(1000);
      }
      point_on = ~point_on;
   };
}
```

在本程序中，数组 led_7[]有 16 个元素，是字型“0”－“F”的段码。由于硬件确定后段码值就固定了，不会改变，因此把它定义在 Flash 中，可以节省 RAM 存储器。LED 数码管的小数点则要根据实际情况使用。

2） 思考与实践

✓ 如何显示其它特殊符号，如“P”、“q”、“L”、“H”等？

✓ 修改程序，控制一个 8 段数码管循环显示“0”－“F”16 个十六进制的数字，每个字符显示 1 秒钟，并且在显示的 1 秒内，数码管的小数点（P 段）要亮 0.5s，灭 0.5s。

6.3.2 多位 LED 数码管的显示

一个 LED 数码管只能显示一位数字，一般在系统经常要使用多个 LED 数码管，如要显示

时间、温度、转速等等。在上面一位数码管控制显示的简单例子中，我们看到，一个数码管要使用 AVR 的 8 个 I/O 口线输出段码（共公端接 GND）。当使用多个数码管时，显然采用这样的控制方式有些问题，因为 AVR 是不能提供太多的 I/O 控制引脚的。比如系统要使用 4 个数码管，按上面例子中的控制方式，ATmega16 的全部 I/O 口将占用，这样其它的外围设备和电路就无法连接了。因此多位数码管的显示驱动系统的实现，有多种不同的方式可以采用，而且在硬件和软件的设计上也是不同的。

多位 LED 数码管显示电路按驱动方式可分为静态显示和动态显示两种方法。

采用静态显示方式时，除了改变显示数据时期外，所有的数码管均处于通电发光状态，每个数码管通电占空比为 100%（上例中的显示方式既为静态显示）。静态显示的优点有：显示稳定，亮度高，程序设计相对简单，MCU 负担小。缺点是：占用硬件资源多（如 I/O 口、驱动锁存电路等），耗电量大。

而所谓动态显示方式，就是一位一位地轮流点亮各个数码管（动态扫描方式）。对于每一位数码管来说，每隔一定时间点亮一次，所以当扫描的时间间隔足够小时，观察者就不会感到数码管的闪烁，看到的现象是所有的数码管一起发光（同看电影的道理一样）。在动态扫描显示方式中，数码管的亮度同 LED 点亮导通时的电流大小，每一位点亮的时间和扫描间隔时间三个因素有关。动态显示的优点有：占用硬件资源少（如 I/O 口、驱动锁存电路等），耗电量小。缺点是：显示稳定性不易控制，程序设计相复杂，MCU 负担重。

为了减轻 MCU 的负担和编程的复杂性，同时简化外围电路，还可以使用专用的数码管控制器件（见第九章）。

1．使用串行传送数据的静态显示接口

图 6-14 是一个采用串行传送数据的 8 位数码管静态显示接口。设计中将 8 片八位串行输入/并行输出移位寄存器 74HC164 串接，数码管为共阳极型。MCU 将 8 个要显示字符的段码字准备好，通过 Data Out 引脚，在 Clk Out 引脚产生的 cp 移位脉冲的作用下，一位一位地移入 74HC164 的 QA－QH 端（串行输入）。QA－QH 的输出（并行输出）直接作为数码管的段位控制。由于左边 74HC164 芯片的 QH（最低位）和右边 74HC164 芯片的数据串入端连接，经过 Clk Out 时钟线 64 个 cp 脉冲后，要显示的 8 个字符将会在 8 个数码管上显示，最先发送的显示字符段码将显示在最右边。

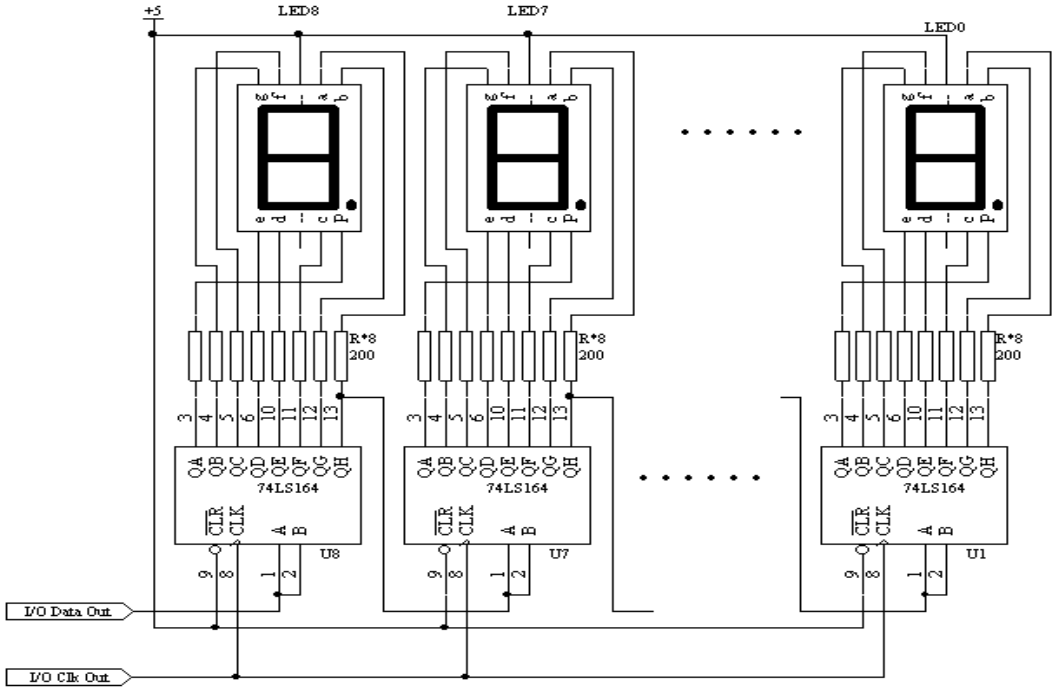

图 6-14 串行数据传送 8 位数码管静态显示接口电路

在这个电路设计中，硬件上使用了 8 片八位串行输入/并行输出移位寄存器 74HC164 串接，占用 AVR 的 2 个 I/O 口。软件的实现比较简单，MCU 只需要把新的显示内容通过两个 I/O 口线，一次串行输出即可。如果显示内容没有变化时，MCU 是不需要对显示部分进行任何操作的。

2．数码管显示器动态显示设计一

采用数码管动态扫描显示方式，可以节省硬件电路，但软件设计相对比较复杂。下面给出一个采用数码管动态扫描显示方式的设计，使用 6 个数码管组成时钟，两个一组，分别显示时、分、秒。

例 6.5 六位 LED 数码管动态扫描控制显示设计（一）

1）硬件设计电路：

图 6-15 给出硬件接口电路图。图中仅采用了 6 个共阴极的 LED 数码管。所有数码管段位 a 的引脚并接，由 PA0 控制；段 b 并接，由 PA1 控制；因此类推。既仍然用 ATmega16 的 PA 口作为段码输出。ATmega16 的 PC0－PC5 分别与 LED0－LED5 的共公端 COM 引脚连接,既 PC 口的低 6 位作为位扫描控制口。

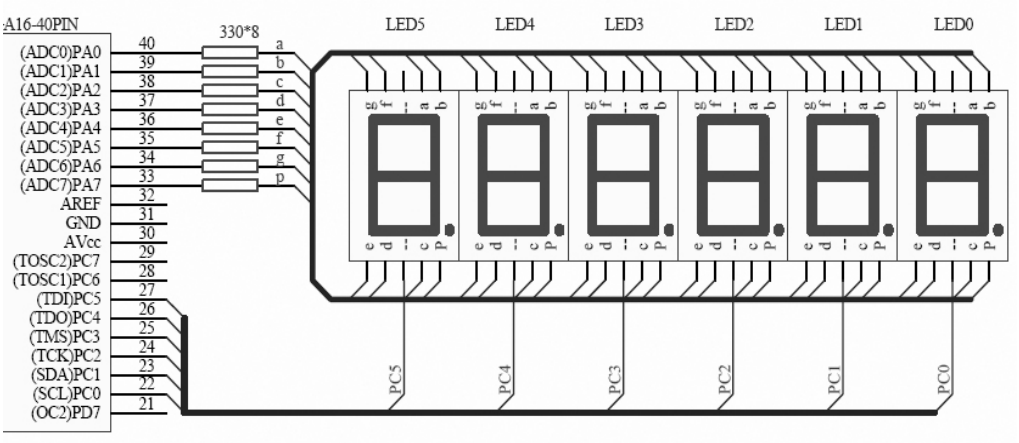

图 6-15 6 位数码管动态扫描显示接口电路

与静态方式的数码管驱动电路相比较，图 6-15 的电路图中没有使用外围器件，但占用了 14 个 I/O 口线（8 位时需要 16 个 I/O 口）。

2）软件设计

根据硬件电路，我们可以看出，在任何一个时刻，PC0-PC5 中只能有一个 I/O 口输出低电平，即只有一位数码管亮。而且，MCU 必须循环轮流控制 PC0-PC5 中的一位输出“ 0 ”，同时 PA 口要输出该位相应的段码值。即使显示的内容没有变化，MCU 也要进行不停的循环扫描处理。

软件的设计应保证从在外表看数码管显示的效果要连续（即在人眼里各个数码管全部亮），亮度均匀，同时没有拖尾现象。

为了保证各个数码管的显示的效果不产生闪烁情况，表象上全部点亮的话，则首先必须在 1 秒中内循环扫描 6 个数码管的次数应大于 25 次，这里是利用了人眼的影像滞留效应。本例中我们选择 40 次,既每隔 1000/40=25ms 将 6 个数码管循环扫描一遍。第二要考虑的是，在 25ms 时间间隔中，要逐一轮流点亮 6 个数码管，那么每个数码管点亮的持续时间要相同，这样亮度才能均匀。第三个要考虑的要点为每个数码管点亮的持续时间，这个时间长一些的话，数码管的亮度高一些，反之则暗一些。

通常，每个数码管点亮的持续时间为 1-2ms。我们将每个数码管的点亮持续时间定为 2ms，那么 6 个数码管扫描一遍的时间为 12ms，因此 MCU 还有 13ms 的时间处理其它事件，

为了简单起见，本例中还是使用了 delay_ms()软件延时函数进行定时，在以后的章节里，将介绍使用 AVR 的定时器产生更精确的秒计时脉冲。

```
/*****************************************
file name          : demo_6_5.c
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 4.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************/

#include <mega16.h>
#include <delay.h>

flash char led_7[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
flash char position[6]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf};
char time[3];                   // 时、分、秒计数
char dis_buff[6];               // 显示缓冲区，存放要显示的 6 个字符的段码值
char time_counter;              // 1 秒计数器
bit point_on;                   // 秒显示标志

void display(void)              // 扫描显示函数，执行时间 12ms
{
    char i;
    for(i=0;i<=5;i++)
    {
        PORTA = led_7[dis_buff[i]];
        if (point_on && ( i==2 || i==4 )) PORTA |= 0x80;    // (1)
        PORTC = position[i];
        delay_ms(2);                                        // (2)
        PORTC = 0xff;                                       // (3)
    }
}

void time_to_disbuffer(void)         // 时间值送显示缓冲区函数
{
    char i,j=0;
    for (i=0;i<=2;i++)
    {
        dis_buff[j++] = time[i] % 10;
        dis_buff[j++] = time[i] / 10;
    }
}
```

```
void main(void)
{
    PORTA=0x00;        // PORTA 初始化
    DDRA=0xFF;
    PORTC=0x3F;        // PORTC 初始化
    DDRC=0x3F;

    time[2] = 23; time[1] = 58; time[0] = 55; // 时间初值 23:58:55
    time_to_disbuffer();

    while (1)
    {
        display();                          // 显示扫描，执行时间 12ms
        if (++time_counter >= 40)
        {
            time_counter = 0;               //  (4)
            point_on = ~point_on;           //  (5)
            if (++time[0] >= 60)
            {
                time[0] = 0;
                if (++time[1] >= 60)
                {
                    time[1] = 0;
                    if (++time[2] >= 24) time[2] = 0;
                }
            }
            time_to_disbuffer();
        }
        delay_ms(13);                   // 延时 13ms，可进行其它处理 (6)
    };
}
```

3）思考与实践

彻底、全面、读懂、理解和体会该段程序，对有（n）注释标记的语句和程序段进行分析，你是否能回答以下问题：

- ✓ 时、分、秒的计算采用何种数制？到数码管的时间显示之间经过了几种数制的转换？为什么要转换（不转换行吗）？怎样转换的？
- ✓ Display()函数是如何工作的？每秒钟执行几次？
- ✓ 说明 time_to_buffer()的功能，每秒执行几次？
- ✓ 说出和深入体会程序中的变量 time_counter、point_on 的作用。
- ✓ 将程序中有（3）注释标记的语句去掉，会产生什么现象，为什么？说明该语句的作用。
- ✓ 将程序中有（4）注释标记的语句去掉，会产生什么现象？
- ✓ 如何调整程序，使数码管的显示亮度有变化？

- ✓ 程序中使用了显示缓冲区，占用了 6 个字节。如果不使用显示缓冲区能否实现时间的显示？而使用显示缓冲区有何优点？
- ✓ 该程序中采用软件延时的方法，其主要的缺点有那些？

3．数码管显示器动态显示设计二

在数码管显示器动态显示设计一中，尽管没有使用更多的外围器件，但是一共占用了 AVR 的 14 个 I/O 口。由于 AVR 的 I/O 口功能比较多，在实际应用中往往需要留出更多的 I/O 口应用于其它的控制和输入，因此经常采用外部增加少量的器件，以减少数码管显示驱动对 I/O 的占用。

例 6.6 六位 LED 数码管动态扫描控制显示设计（二）

1） 硬件设计

如果在硬件设计上多使用一片 74HC164（八位串行输入/并行输出的移位寄存器），就只要使用 PA 口的 8 根段输出线中的 2 根作为段码输出控制，其它 6 个 I/O 口就可以节省为它用了，电路图见图 6-16。

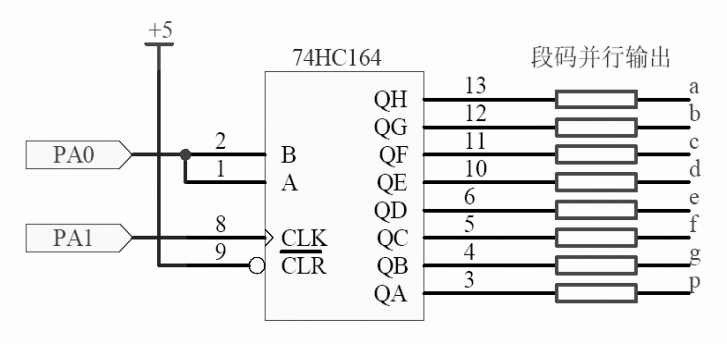

图 6-16 采用 74HC164 串入并出输出段码的接口电路

在图 6-16 中，仅给出的数码管段控制部分的接口电路。位控制仍然同图 6-15，使用 PC 口的 6 个 I/O。这个设计中使用了 AVR 的 PA0 和 PA1 两个 I/O 口串出段码，比图 6-15 中少使用 6 个 I/O 口。

2） 软件设计

根据硬件电路可以知道，数码管显示器的工作方式还是为动态显示的方法。因此，软件中应根据 74HC164 的逻辑真值表（表 6.4），增加一个使用 PA0（data）、PA1（clk）串行输出一个字节的功能函数。

表 6.4 74HC164 逻辑真值表

INPUTS(输入)				OUTPUTS(输出)			
CLR	CLK	A	B	QA	QB	……	QH
L	X	X	X	L	L	……	L
H	↓	X	X	No change			
H	↑	L	X	L	QAn	……	QGn
H	↑	X	L	L	QAn	……	QGn
H	↑	H	H	H	QAn	……	QGn

```
/*********************************************
file name           : demo_6_6.c
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 4.000000 MHz
```

```
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************/

#include <mega16.h>
#include <delay.h>

#define HC164_data PORTA.0
#define HC164_clk  PORTA.1

flash char led_7[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
flash char position[6]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf};
char time[3];                   // 时、分、秒计数
char dis_buff[6];               // 显示缓冲区，存放要显示的 6 个字符的段码值
char time_counter;
bit point_on;

void HC164_send_byte(char byte)
{
     char i;
     for (i=0;i<=7;i++)
     {
          HC164_data = byte & 1<< i;
          HC164_clk = 1;
          HC164_clk = 0;
     }
}

void display(void)
{
     char temp,i;
     for(i=0;i<=5;i++)
     {
          temp = led_7[dis_buff[i]];
          if (point_on && (i==2 || i==4))
               HC164_send_byte(temp | 0x80);
          else
               HC164_send_byte(temp);
          PORTC = position[i];
          delay_ms(2);
          PORTC = 0xff;
     }
}
```

°°°°°°°°°

程序的其它部分同 demo_6_5.c，只是对 display()函数中的段码输出语句进行了修改，变成调用 HC164_send_byte()函数输出段码。在 HC164_send_byte()函数中，程序控制 PA0、PA1，模拟出串行输出数据的时序，将一个字节的数据由低到高串行输出到 74HC164 中。

3） 思考与实践

- ✓ 如果要将一个字节的数据由高到低串行输出到 74HC164 中，HC164_send_byte()函数应该如何改动？硬件电路要如何调整？
- ✓ 在 HC164_send_byte()函数中的 FOR 循环中有 3 句语句，说明“HC164_data = temp & 1<<i;”的作用。此外这 3 句语句的执行顺序可以改动吗？
- ✓ 如果再增加一片芯片的话，还可以使用更少的 I/O 口实现数码管动态扫描显示的接口。请设计硬件电路和相应的软件显示控制程序，能够实现本例的功能（提示：可考虑 74HC164 或 74HC138）。

6.3.3 点阵 LED 显示控制

点阵 LED 在许多产品中也是经常使用的一种外围设备，如电梯中的运行指示，公交汽车里的站名广告显示，以及大型的电子广告牌等。这种 LED 的优点是可以通过点阵的形式显示汉字、图形等。实际上，PC 的显示屏、手机显示屏等，在上面显示汉字、图形的原理都是点阵显示的方法。

例 6.7 8*8 点阵 LED 显示控制设计

1） 硬件设计

8*8 点阵 LED 一般是一个方型的器件，由 8 行 * 8 列共 64 个 LED 发光二极管组成。图 6-17 是它的内部电原理图（4*4 点阵）。

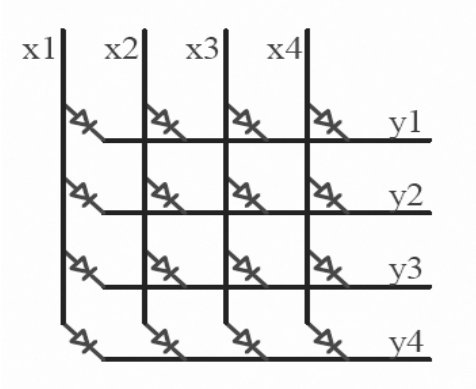

图 6-17 4*4 点阵 LED 原理图

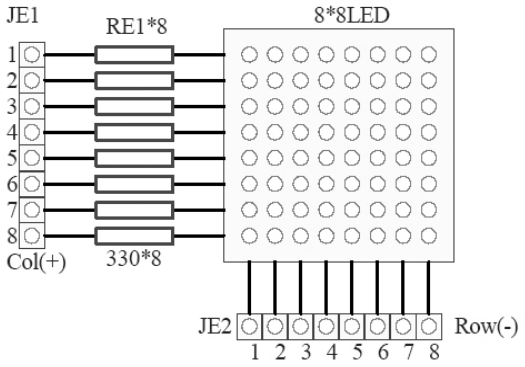

图 6-18 8*8 点阵 LED 模块

图 6-18 为 8*8 点阵 LED 的模块，我们使用 AVR 的 PA 口连接 JE1（Col+），PC 口连接 JE2（Row-），当 PA 输出一个字节数据，PC 口 8 位只有输出为“0”时，模块的其中一行（列）8 个 LED 就会根据 PA 的输出值点亮（熄灭）。

2） 软件设计

通过电路的分析，可以看出，8*8 点阵 LED 的显示控制方式与 LED 数码管的显示方式类似，也是使用动态扫描的工作方式。下面我们设计实现一个简单的，用于电梯中指示运行的向上运动的箭头“↑”。

首先我们要建立一个向上箭头“↑”的码表，见表 6.5。然后需要一个动态扫描显示的函数 display()。

表 6.5　上箭头“↑”的码表

		Col+								PA 口输出值
		1 (PA7)	2 (PA6)	3 (PA5)	4 (PA4)	5 (PA3)	6 (PA2)	7 (PA1)	8 (PA0)	
Row(-)	1(PC0)				●					0x10
	2(PC1)			●	●	●				0x38
	3(PC2)		●	●	●	●	●			0x7C
	4(PC3)	●	●	●	●	●	●	●		0xFE
	5(PC4)			●	●	●				0x38
	6(PC5)			●	●	●				0x38
	7(PC6)			●	●	●				0x38
	8(PC7)			●	●	●				0x38

```
/*****************************************
file name           : demo_6_7.c
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 4.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************/

#include <mega16.h>
#include <delay.h>

flash char char_7[8]={0x10,0x38,0x7C,0xFE,0x38,0x38,0x38,0x38};

void display(char row)
{
   char i;
   for (i=0;i<=7;i++)
   {
        if (row <= 7)
            PORTA = char_7[row];
        else
            PORTA = 0;
        PORTC = ~(1<<i);
        delay_ms(2);
        PORTC = 0xFF;
        if (++row >= 12 ) row = 0;
   }
}

void main(void)
{
```

```
    char time_counter,i = 0;
    PORTA=0x00;
    DDRA=0xFF;
    PORTC=0xFF;
    DDRC=0xFF;

    while (1)
    {
        display(i);
        delay_ms(9);
        if (++time_counter >= 4)
        {
            time_counter = 0;
            if(++i >= 12) i=0;
        }
    };
}
```

3）思考与实践

- ✓ 本例程与数码管显示程序有非常多的类似地方，请读者自己分析程序的功能。
- ✓ 请给出程序中的时间分配情况，如何调整箭头移动的速度的快慢，而又不影响正常的显示？
- ✓ 如果将 display()函数中的语句“ PORTC = ~(1<<i); ”该为“ PORTC = ~(1<<(7-i)); ”，显示有何变化？
- ✓ 设计一个水平移动的 8*8 点阵广告，能够显示“今天 OK？ ”

6.4 LCD 液晶显示器的应用

液晶显示器（LCD）由于体积小、重量轻、耗电小等优点已成为各种嵌入式系统所采用的理想显示器。近年来液晶显示器技术的发展迅猛，大面积的液晶显示器已开始取代 CRT 显示器，在使用电池供电的嵌入式电子产品中，如手机、PDA，以及家电产品，仪器仪表产品等，液晶显示器是首选的显示器。

6.4.1 LCD 的特点与分类

1．LCD 的特点

- ✓ 低电压微功耗。工作电压 3-5V，每平方厘米液晶显示屏的耗电量在 uA 级。
- ✓ 平板结构。易大量生产，物理体积小，占用空间少。
- ✓ 寿命长。
- ✓ 光线柔和。液晶显示器是被动发光器件，90%以上是外部物体对光的反射。被动显示适合人的视觉习惯，不会引起疲劳。
- ✓ 无电磁辐射。液晶显示器不会产生电磁辐射，是绿色器件。

2．LCD 显示器的分类

从液晶显示器的使用和显示内容来分，LCD 可分为字段式（笔划式），点阵字符式，点阵图形式三种。

字段式液晶显示器同 LED 数码显示器有些相同点，它是以长条笔划状或一些特殊固定图形与汉字显示象素组成的液晶显示器件,简称段型显示器。段型显示器以七段显示器为常见，特殊图形与字符类的段型液晶显示器一般要到生产厂家定做。段型液晶显示器在数字仪表、计数器，家电产品中应用较多。

点阵字符式液晶显示器一般是一个功能模块，它由小面积的液晶显示屏和驱动电路组合而成。模块中内置有 192 种字符、数字、字母、标点符号等可显示的字型点阵图形库，并提供可控制的并行或串行接口以及通信协议。市场上常见的有 1 行、2 行、4 行，每行可显示 8、12、16、24、32 个 5x7 点阵字符的通用液晶显示器。

点阵图形式液晶显示器一般显示面积大于点阵式液晶显示器，点阵从 80x32 到 1024x768 不等。点阵图形式液晶显示器的显示灵活性好，自由度大，可以显示各种图形、字符和汉字等。但点阵图形式液晶显示器的控制最复杂，硬件连接线多，占用 MCU 的资源也多。为了适应越来越多的液晶显示器应用，一些高性能的单片机已经将液晶显示器驱动功能集成在片内。目前国内一些厂商将驱动电路、汉字库和点阵液晶显示器屏做成一个组件模块，模块带有与 MCU 通信的并行或串行接口，使用时，只要 MCU 通过通信口下发相应的控制指令就能显示各种信息，方便了使用。

6.4.2 通用点阵字符 LCD 显示器应用

通用点阵字符液晶显示器是专用于显示数字、字母、图形符号和一些自定义符号的显示器。这类显示器把 LCD 控制器、点阵驱动器、字符存储器全做在一块 PCB 板上，构成便于应用的显示器模块。这类点阵字符液晶显示器模块在国际上已经规范化，一般都采用日立公司的 HD44780 极其兼容电路，如 SED1278、KS0066 等，作为 LCD 的控制器。

HD44780 具有简单而功能较强的指令集，可实现字符移动、闪烁等功能。与 MCU 的数据传输可采用 8 位并行或 4 位并行传输两种方式。可用于驱动 40 * 4，16 * 1，16 * 2，16 * 4，20 * 2，20 * 4，等多种点阵字符液晶显示器。HD44780 对外有 14 根引脚，与 MCU 的接口信号及定义见表 6.6。

表 6.6 HD44780 引脚功能定义表

引脚号	符号	I/O	功　　能
1	Vss		电源负端,接地（或接－5V）
2	Vdd		电源正端,接+5V
3	Vo		LCD 亮度调整电压 0-5V
4	RS	I	寄存器选择:RS=0,选指令寄存器;RS=1,选数据寄存器
5	R/W	I	读/写选择:R/W=0,写数据至 LCD;R/W=1,从 LCD 读数据
6	E	I	输入允许:R/W=0,E↓下降沿打入;R/W=1,E=1 有效
7-10	DB0-DB3	I/O	数据总线：使用 4 位并行传输时仅用(DB4—DB7)4 位
11-14	DB4-DB7	I/O	使用 8 位并行传输时使用(DB0—DB7)8 位
15-16			LCD 背光电源的正极和负极(有些模块没有背光功能)

从零开始编写 HD44780 的控制程序需要了解 HD44780 的内部结构、操作时序、指令集、内部 REM 与字符图形的对应关系和字符代码表等等。编写程序时需要先编写底层的倾动程序，再编写上层的应用接口程序，再加上程序调试时间，通常要花费 3-5 天时间。对于一般

的初学者，花费 2-3 个星期也未必能完成软件的设计。但由于这种点阵字符液晶显示器模块在国际上已经规范化，因此在 CVAVR 中扩展提供了一些基本的 LCD 应用接口函数，所以在 CVAVR 平台的支持下，用户使用这类 LCD 点阵字符显示器就比较方便。你只要写几条语句，调用 CVAVR 的 LCD 提供的函数，化几分钟的时间，就能把要显示的信息在 LCD 上显示出来。

在 CVAVR 中，与 LCD 字符显示器有关的功能函数有：

1）void lcd_init(unsigned char lcd_columns)

该函数对 LCD 进行初始化，并清除 LCD 的显示，将显示位置回到第 0 行的第 0 列的起始位置处。函数的参数应是 LCD 显示器的列数（一行能够显示的字符数）。使用 LCD 显示器时，必须先使用该函数对 LCD 显示器进行初始化。

2）void lcd_clear(void)

该函数清除 LCD 的显示，并将显示位置回到第 0 行的第 0 列的起始位置处。

3）void lcd_gotoxy(unsigned char x, unsigned char y)

该函数将显示位置定位于第 x 行的第 y 列的位置处。注意，LCD 的行列定位都是从“0”起始的。

4）void lcd_putchar(char c)

该函数将字符 c 在当前的显示位置上显示出来。

5）void lcd_puts(char *str)

该函数将在从当前的显示位置开始，显示定义在 SRAM 中的字符串（str 为 SRAM 中定义的字符串的指针）。

6）void lcd_putsf(char flash *str)

该函数将在从当前的显示位置开始，显示定义在 Flash 中的字符串（str 为 Flash 中定义的字符串的指针）。

除了上面 6 个 LCD 函数外，在 CVAVR 中还有一些扩展的用于字符 LCD 控制的函数，能提供更多的功能。请读者在具体应用时，仔细阅读 CVAVR 的 Help 和使用手册，掌握这些函数的使用，如学会如何自己设计定义和使用特殊的符号和图形（HD44780 支持用户自己定义最多 8 个 5*8 点阵的字符和图形）。

例 6.8 16*2 标准 LCD 字符显示器应用设计

1） 硬件设计

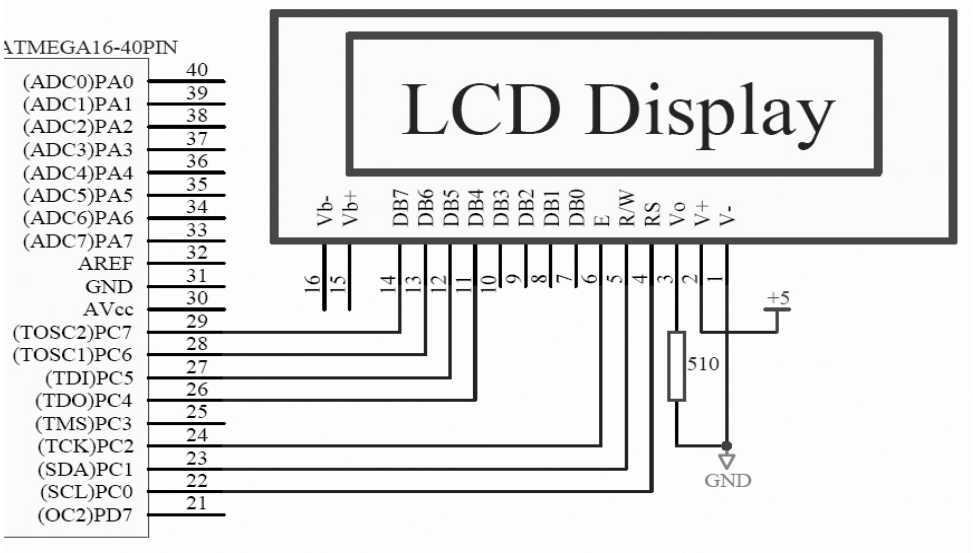

图 6-19 16*2 标准字符型 LCD 接口电路

图 6-19 所示为 16*2 标准字符型 LCD 的连接电路图。由于在 CVAVR 中，必须按照一定的规定连接 LCD 时，才能使用 CVAVR 内部提供的 LCD 函数，所以原理图是按照 CVAVR 的规定设计的。

如要使用 CVAVR 内部提供的 LCD 函数，硬件连接必须按以下要求实现。

1） 与 LCD 的连接必须使用 AVR 的同一个 8 位的 I/O 端口，如 PC（或者 PA、PB、PD）。

2） LCD 采用 4 位并行传输方式（既仅用 DB4—DB7，4 位数据总线）。

3） 具体连接定义为（以 PC 口为例）：

三根控制线　　PC0----RS，PC1----R/W，PC2----E

四根数据线　　PC4----DB4，PC5----DB5，PC6----DB6，PC7----DB7

2） 软件设计

```
/*********************************************
File name          : demo_6_8.c
Chip type          : ATmega16
Program type       : Application
Clock frequency    : 4.000000 MHz
Memory model       : Small
External SRAM size : 0
Data Stack size    : 256
*********************************************/
#include <mega16.h>
#include <delay.h>

#asm
.equ __lcd_port=0x15 ; PORTC 数据寄存器地址
#endasm
/*  [LCD]
    1 GND- 9  GND
    2 +5V- 10 VCC
    3 VLC- LCD HEADER Vo
    4 RS - 1  PC0 (M16)
    5 RD - 2  PC1 (M16)
    6 EN - 3  PC2 (M16)
   11 D4 - 5  PC4 (M16)
   12 D5 - 6  PC5 (M16)
   13 D6 - 7  PC6 (M16)
   14 D7 - 8  PC7 (M16) */
#include <lcd.h>
flash char dis_str[]="Hello World! This is a LCD display demo.";
void main(void)
{
  char flash *str;
```

```
    str = dis_str;
    lcd_init(16);               // initialize the LCD for 2 lines & 16 columns
    while(1)
    {
        lcd_clear();                    // clere the LCD
        lcd_putsf("It's demo_6_8.c");   // display the message
        lcd_gotoxy(0,1);                // go on the second LCD line
        lcd_putsf(str);                 // display the message
        if (*str++ == 0) str = dis_str;
        delay_ms(500);
     }
  }
```

该简单的 LCD 显示的演示程序全部调用的是 CVAVR 中的 LCD 函数，程序运行后，在 LCD 的第一行固定显示字符“ It’s demo_6_8.c ”，在第二行滚动显示“ Hello World! This is a LCD display demo. ”。

在程序的开始部分，嵌入了一句 AVR 汇编的伪指令：“ .equ __lcd_port = 0x15 ”，这也是 CVAVR 规定要使用的，它通知 CVAVR 编译系统，在硬件上 AVR 与 LCD 连接的 I/O 地址为 0x15，这是 PC 口的数据寄存器 PORTC 的地址。此时，用户不必关心 PC 口的初始化问题，在调用 lcd_init(16)函数时，该函数会对 PC 口进行必要的初始化工作。因此，使用标准字符 LCD 显示器时，必须先使用该函数进行初始化工作。

在 CVAVR 中还有一些扩展的用于字符 LCD 控制的函数，能提供更多的功能，如允许用户自己设计定义和使用特殊的符号和图形（HD44780 支持用户自己定义最多 8 个 5*8 点阵的字符和图形）等。下面的演示程序，可以在 LCD 上显示用户定义的简单汉字“ 天天向上 ”。

```
/*****************************************
File name           : Demo_6_9.c
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 4.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************/
#include <mega16.h>
// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x15
#endasm
#include <lcd.h>

typedef unsigned char byte;

/* table for the user defined character */
flash byte char0[8]={       // 天的字型
0b0011111,
```

```
0b0000100,
0b0000100,
0b0011111,
0b0000100,
0b0000100,
0b0001010,
0b0010001};
flash byte char1[8]={       // 向的字型
0b0000100,
0b0001000,
0b0011111,
0b0010001,
0b0011111,
0b0011011,
0b0011111,
0b0010001};
flash byte char2[8]={       // 上的字型
0b0000100,
0b0000100,
0b0000111,
0b0000100,
0b0000100,
0b0000100,
0b0000100,
0b0011111};

/* function used to define user characters */
void define_char(byte flash *pc,byte char_code)
{
     byte i,a;
     a=(char_code<<3) | 0x40;
     for (i=0; i<8; i++) lcd_write_byte(a++,*pc++);
}

void main(void)
{
     lcd_init(16);          // initialize the LCD for 2 lines & 16 columns

     define_char(char0,0);  // define user character 0
     define_char(char1,1);  // define user character 1
     define_char(char2,2);  // define user character 2
     lcd_clear();
     lcd_putsf("Demo_6_9.c");    // 第一行显示内容
     lcd_gotoxy(0,1);
```

```
        lcd_putsf("User define:");  // 第二行显示内容
        lcd_putchar(0);             // 接在后面显示“天天向上”
        lcd_putchar(0);
        lcd_putchar(1);
        lcd_putchar(2);
        while (1);
    }
```

有兴趣的读者,可以自己尝试和练习编写标准字符型LCD控制芯片HD44780的控制程序,关于HD44780的详细资料可以在本书附带的光盘中找到。

思考与练习

1. AVR单片机I/O口三个寄存器的名称和作用是什么？当I/O口用于输入和输出时如何设置和应用这三个寄存器？
2. 给出一个8位数码管显示器静态显示的硬件和软件设计方案以及一个动态扫描显示的硬件和软件设计方案，并比较这两个方案的优缺点。
3. 全面、仔细、深入分析程序demo_6_5.c，说明在动态扫描显示设计中，如何保证每个显示器的亮度一致，在系统应用中没有闪烁和熄灭现象。
4. 在动态扫描显示中，如果要调整显示的亮度，请给出三种硬件和软件的设计改动方法，并说明理由。
5. 认真分析和理解本章中所给出的所有示例，并进行实践，思考和回答所有的问题。