

### 3.3 流水灯之软件延时

#### 能力目标

理解并掌握通过改进算法提高编程效率的方法。

#### 任务目标

LED 流水灯仿真电路如图 3-34 所示，要求实现流水灯效果，即按 LED0~LED7 的顺序依次点亮，每次仅限 1 个 LED 发光，周期为 4 秒。

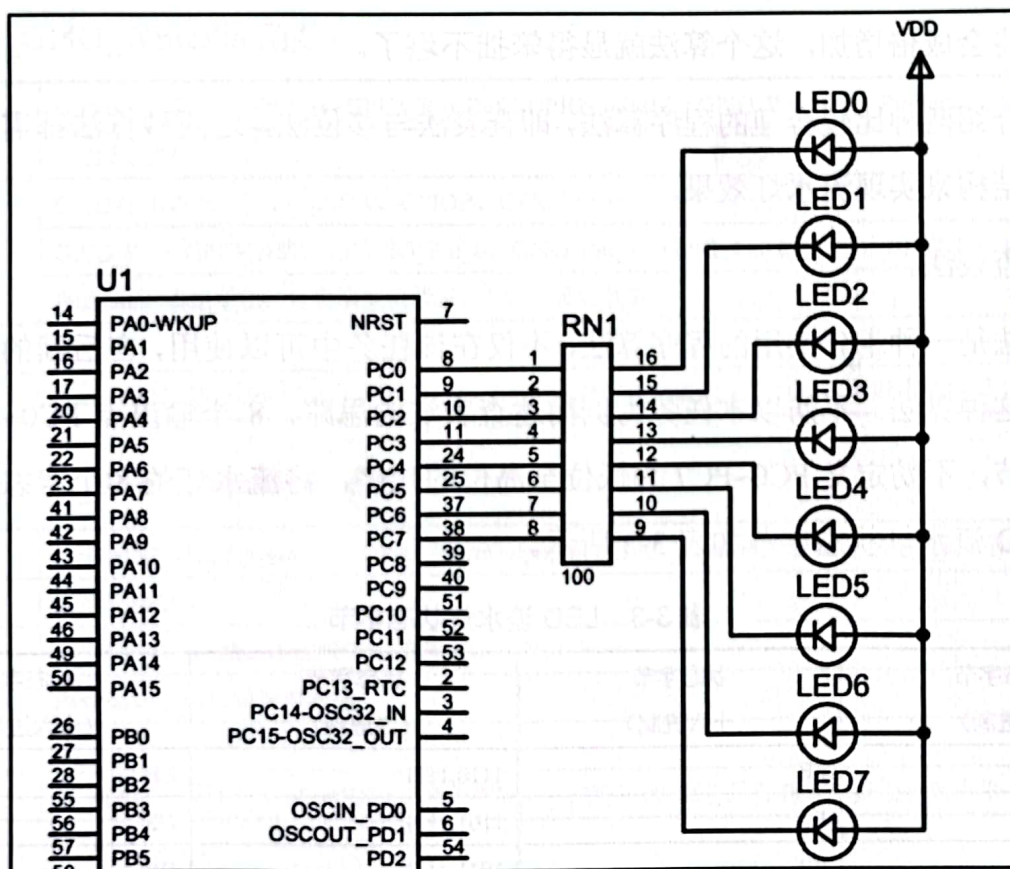


图 3-34 LED 流水灯仿真电路

LED 流水灯仿真电路中的虚拟元器件如表 3-2 所示。

表 3-2 LED 流水灯仿真电路中的虚拟元器件

名 称	说 明
STM32F103R6	单片机
RX8	排阻
LED-GREEN	绿色发光二极管

### 3.3.1 程序算法

算法 (Algorithm) 是指解决方案准确而完整的描述, 是一系列解决问题的清晰指令, 算法代表着用系统的方法描述解决问题的策略机制。简单来讲, 程序算法就是编写程序的指导思想。

初学者可能第一时间想到的算法就是点亮一个 LED, 熄灭后再点亮下一个 LED……以此类推, 循环往复。我们不妨称这个算法为“位操作法”, 它虽然简单直白, 但不具备普遍适用性, 如果现在 LED 的个数不是 8 个而是 16 个甚至更多个, 那么程序的代码量将会成倍增加, 这个算法就显得笨拙不堪了。

下面介绍两种比较合理的程序算法, 即查表法与移位法, 这两种算法都需要结合有限次循环结构来实现流水灯效果。

#### (1) 查表法。

查表法是一种十分实用的程序算法, 不仅在该任务中可以使用, 在后续的章节也会陆续用到这种算法。不妨以本任务为例简述查表法的思路, 8 个输出点 PC0~PC7 刚好构成一字节, 不妨定义 PC0~PC7 为低位至高位的顺序, 将流水灯的 8 个状态定义为八字节。LED 流水灯状态字节如表 3-3 所示。

表 3-3 LED 流水灯状态字节

状态字节 (二进制)	状态字节 (十六进制)	状态字节 (二进制)	状态字节 (十六进制)
1111,1110	FE	1110,1111	EF
1111,1101	FD	1101,1111	DF
1111,1011	FB	1011,1111	BF
1111,0111	F7	0111,1111	7F

查表法结合有限次循环结构即可实现流水灯效果。

#### (2) 移位法。

移位法是利用 C 语言的移位运算符 “<<” “>>” 实现状态字节的循环移位的。但由于 C 语言的移位运算符只能实现单向移位 (移出位丢弃, 空白位补 0), 因此必须通过一定的算法来间接实现状态字节的循环移位, 具体的做法是: 假设  $M$  位数据  $A$  需要循



环左移  $N$  位 ( $M > N$ ), 先将  $A$  左移  $N$  位得到  $B$ , 再将  $A$  右移  $M - N$  位得到  $C$ , 最后将  $A$ 、 $B$  按位求或即可获得最终结果。

例如, 8 位二进制数各位均用字母表示为 “ABCDEFGH”, 需要循环左移 3 位, 可先将原数左移 3 位, 得到 “DEFGH000”; 再将原数右移 5 位, 得到 “00000ABC”; 最后将两数按位求或即可得到循环左移 3 位的结果 “DEFGHABC”。

3.3.2 任务程序的编写

任务程序分别采用位操作法、查表法与移位法编写, 使用的新 API 函数如下。

HAL\_GPIO\_WritePin 函数:

函数	void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
功能简述	写引脚函数
形参	GPIOx: GPIO 组, 如 GPIOA、GPIOB、GPIOC……
	GPIO_Pin: GPIO 引脚, 如 GPIO_Pin_0、GPIO_Pin_1、GPIO_Pin_2、……、GPIO_Pin_All
	PinState: 输出状态, 0 表示复位状态; 1 表示置位状态
返回值	无
应用举例	HAL_GPIO_WritePin(GPIOC,GPIO_PIN_0,GPIO_BIT_RESET); //PC0 引脚输出低电平

LL\_GPIO\_WriteOutputPort 函数:

函数	__STATIC_INLINE void LL_GPIO_WriteOutputPort(GPIO_TypeDef * GPIOx, uint32_t PortValue)
功能简述	写端口函数
形参	GPIOx: GPIO 组, 如 GPIOA、GPIOB、GPIOC……
	PortValue: 输出到端口的值
返回值	无
应用举例	LL_GPIO_WriteOutputPort(GPIOC,0xffffffff); //整个 PC 端口只有 PC0 引脚输出低电平

值得注意的是, HAL\_GPIO\_WritePin 函数属于 HAL 库, LL\_GPIO\_WriteOutputPort 函数属于 LL 库, 这两个库只能二选一, 默认选择为 HAL 库。若读者需要使用 LL\_GPIO\_WriteOutputPort 函数, 则必须在如图 3-35 所示的驱动选择器界面中将 GPIO 引脚的驱动库改为 LL 库。

驱动选择器界面打开方法为, 首先打开图形化配置界面, 然后依次选择 “Project Manager” (工程管理器) → “Advanced Setting” (高级设置) 选项。

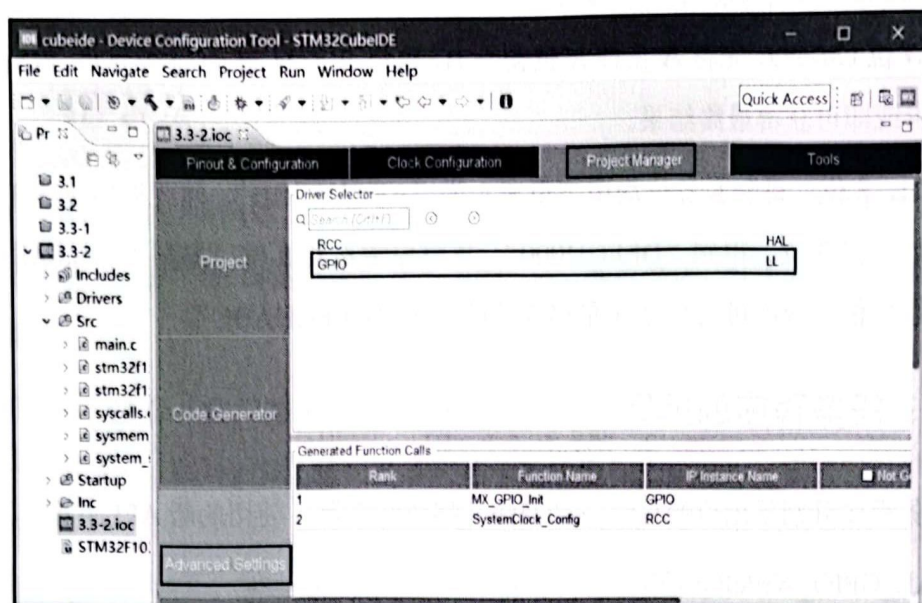


图 3-35 驱动选择器界面

(1) 采用位操作法编写的程序:

```
#include "main.h"
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        //LED0 亮
        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_7,1);
        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_0,0);
        HAL_Delay(500);
        //LED1 亮
        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_0,1);
        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_1,0);
        HAL_Delay(500);
        //LED2 亮
        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_1,1);
        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_2,0);
        HAL_Delay(500);
        //LED3 亮
        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_2,1);
        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_3,0);
    }
}
```



```

    HAL_Delay(500);
    //LED4 亮
    HAL_GPIO_WritePin(GPIOC,GPIO_PIN_3,1);
    HAL_GPIO_WritePin(GPIOC,GPIO_PIN_4,0);
    HAL_Delay(500);
    //LED5 亮
    HAL_GPIO_WritePin(GPIOC,GPIO_PIN_4,1);
    HAL_GPIO_WritePin(GPIOC,GPIO_PIN_5,0);
    HAL_Delay(500);
    //LED6 亮
    HAL_GPIO_WritePin(GPIOC,GPIO_PIN_5,1);
    HAL_GPIO_WritePin(GPIOC,GPIO_PIN_6,0);
    HAL_Delay(500);
    //LED7 亮
    HAL_GPIO_WritePin(GPIOC,GPIO_PIN_6,1);
    HAL_GPIO_WritePin(GPIOC,GPIO_PIN_7,0);
    HAL_Delay(500);
/* USER CODE END WHILE */
}
}
.....

```

## (2) 采用查表法编写的程序:

```

#include "main.h"
/* USER CODE BEGIN PV */
uint8_t Status[]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f}; //八状态字节
/* USER CODE END PV */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
int main(void)
{
    /* USER CODE BEGIN 1 */
    uint8_t i; //循环变量
    /* USER CODE END 1 */
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        for(i=0;i<8;i++)
        {
            LL_GPIO_WriteOutputPort(GPIOC,Status[i]); //状态字节的输出
            HAL_Delay(500);
        }
    }
}

```



```

    /* USER CODE END WHILE */
}
}
.....

```

### (3) 采用移位法编写的程序:

```

#include "main.h"
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
int main(void)
{
    /* USER CODE BEGIN 1 */
    uint8_t i, dat=0xfe; //循环变量、输出状态字节初始状态
    /* USER CODE END 1 */
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        for(i=0;i<8;i++) //8 次循环
        { //移位、输出一步完成
            LL_GPIO_WriteOutputPort(GPIOC, (dat<<i)|(dat>>(8-i)));
            HAL_Delay(500);
        }
    }
    /* USER CODE END WHILE */
}
}
.....

```

对比采用以上 3 种算法编写的程序,显然采用位操作法编写的程序最直观,代码量也最多。采用查表法与移位法编写的程序代码量相当,但采用查表法编写的程序提前将各状态字节进行人工计算形成了数表(这里就是字节数组),因此相比于采用移位法编写的程序大大降低了 CPU 的运算负荷。在这个任务中,查表法与移位法都不失为优秀的程序算法,值得读者学习和推敲。

另外,细心的读者应该已经发现上述程序中在定义变量的时候使用了第 2 篇未介绍过的关键字“uint8\_t”,这是因为意法半导体公司对 C 语言的数据类型进行了重命名,“uint8\_t”表示无符号 8 位整型数据,等效为“unsigned char”。同样,“int8\_t”等效为“char”,与此类似的还有“int16\_t”“uint16\_t”“int32\_t”“uint32\_t”“int64\_t”“uint64\_t”,相比于 C 语言原始的字符型关键字与整型关键字,这样能更好地表达数据类型的长度,



建议读者在编写 STM32 单片机程序的时候,尽可能使用意法半导体公司重新定义的数据类型名。

## 3.4 数码管动态显示

### 能力目标

在理解数码管工作原理与多位数码管电路构成的基础上,理解并掌握多位数码管动态显示字符的程序编写方法。

### 任务目标

8 位数码管动态显示仿真电路如图 3-36 所示,要求编程实现 8 位数码管依次分别显示十进制数字 1~8。

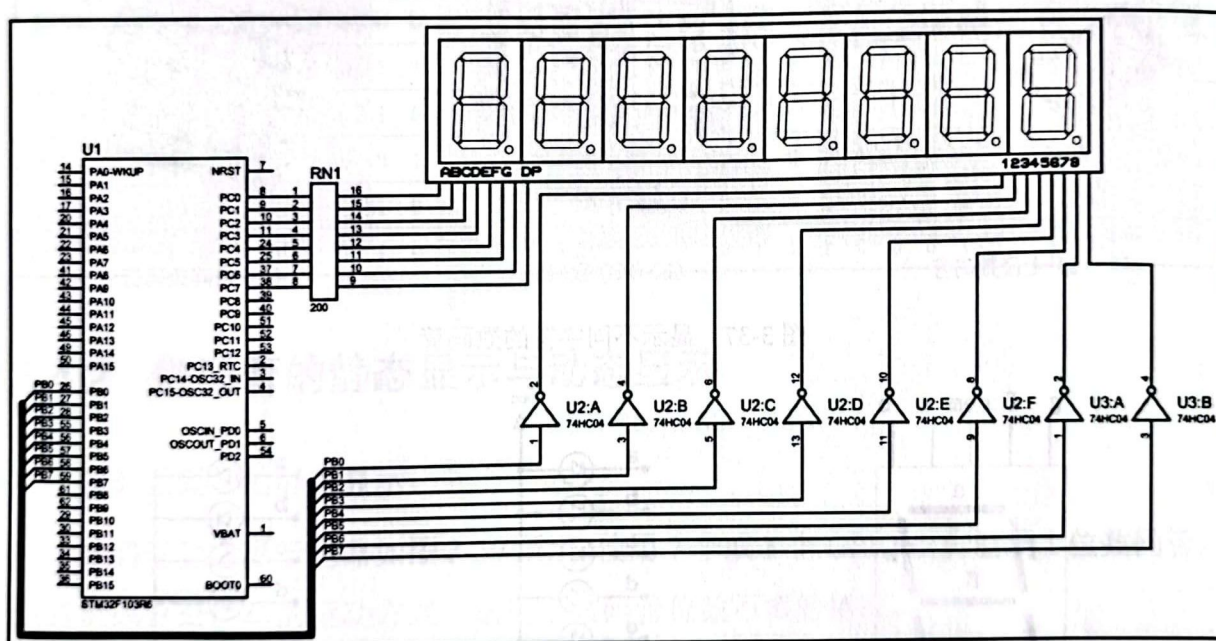


图 3-36 8 位数码管动态显示仿真电路

8 位数码管动态显示仿真电路中的虚拟元器件如表 3-4 所示。

表 3-4 8 位数码管动态显示仿真电路中的虚拟元器件

名 称	说 明
STM32F103R6	单片机
RX8	排阻
7SEG-MPX8-CA-BLUE	8 位蓝色共阳极七段数码管显示器
74HC04	非门电路

### 3.4.1 数码管的结构

数码管显示器（简称数码管）是一种常见的显示元器件，由于其显示清晰、价格低廉，因此得到了广泛应用。如图 3-37 所示，按照显示字符划分，常见的数码管有七段数码管、米字数码管、点阵数码管等。

本节仅为读者介绍七段数码管，它由 7 个或 8 个（多 1 个小数点位 dp 位）发光二极管组成，通过控制发光二极管的发光状态达到显示数字或字符的目的。根据发光二极管连接方式的不同，七段数码管又可分为共阳极数码管与共阴极数码管两种，如图 3-38 所示。

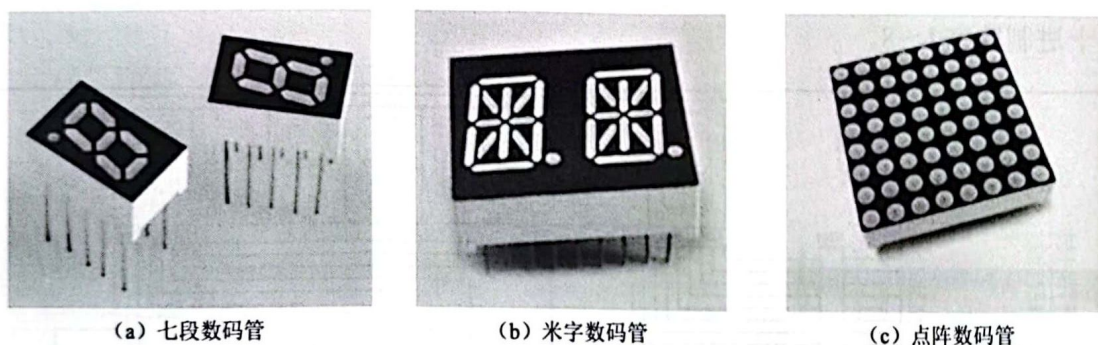


图 3-37 显示不同字符的数码管

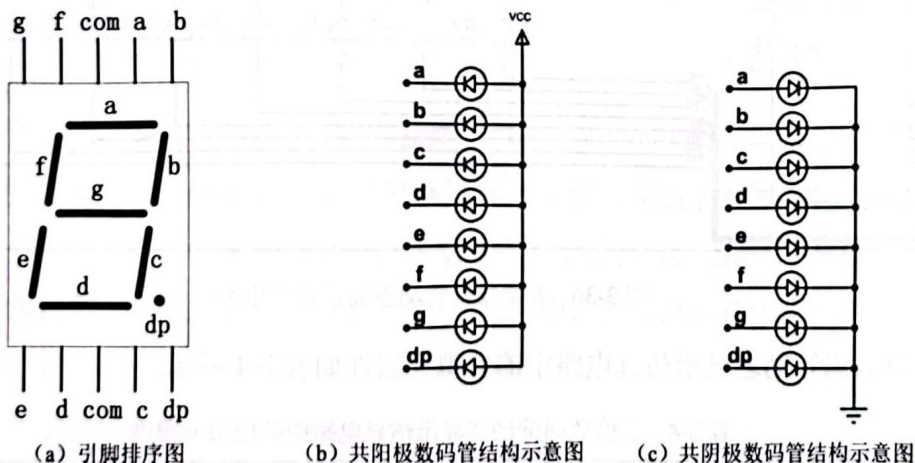


图 3-38 七段数码管的结构示意图

显然，共阳极数码管采用低电平驱动，而共阴极数码管采用高电平驱动。不妨假设数码管的 8 个输入点 a~dp 按由低位到高位顺序构成一字节，其段码即驱动编码，如表 3-5 所示。



表 3-5 数码管段码

显示字符	共阳极数码管段码			共阴极数码管段码		
	二进制段码		十六进制段码	二进制段码		十六进制段码
	dp	g f e d c b a		dp	g f e d c b a	
全灭	1	1 1 1 1 1 1 1 1	FFH	0	0 0 0 0 0 0 0 0	00H
0	1	1 0 0 0 0 0 0 0	C0H	0	0 1 1 1 1 1 1 1	3FH
1	1	1 1 1 1 1 0 0 1	F9H	0	0 0 0 0 0 1 1 0	06H
2	1	0 1 0 0 1 0 0 0	A4H	0	1 0 1 1 0 1 1 1	5BH
3	1	0 1 1 0 0 0 0 0	B0H	0	1 0 0 1 1 1 1 1	4FH
4	1	0 0 1 1 0 0 0 1	99H	0	1 1 0 0 0 1 1 0	66H
5	1	0 0 1 0 0 1 0 0	92H	0	1 1 0 1 1 0 1 1	6DH
6	1	0 0 0 0 0 0 1 0	82H	0	1 1 1 1 1 0 1 1	7DH
7	1	1 1 1 1 1 0 0 0	F8H	0	0 0 0 0 0 1 1 1	07H
8	1	0 0 0 0 0 0 0 0	80H	0	1 1 1 1 1 1 1 1	7FH
9	1	0 0 1 0 0 0 0 0	90H	0	1 1 0 1 1 1 1 1	6FH
A	1	0 0 0 1 0 0 0 0	88H	0	1 1 1 0 1 1 1 1	77H
B	1	0 0 0 0 0 0 1 1	83H	0	1 1 1 1 1 0 0 0	7CH
C	1	1 0 0 0 0 1 1 0	C6H	0	0 1 1 1 0 0 0 1	39H
D	1	0 1 0 0 0 0 0 1	A1H	0	1 0 1 1 1 1 1 0	5EH
E	1	0 0 0 0 0 1 1 0	86H	0	1 1 1 1 0 0 0 1	79H
F	1	0 0 0 0 1 1 1 0	8EH	0	1 1 1 0 0 0 0 1	71H

### 3.4.2 数码管的静态显示与动态显示

#### (1) 数码管的静态显示。

数码管静态显示电路如图 3-39 所示,使用 7 个或 8 个 GPIO 引脚驱动 1 位数码管,这是数码管最简单的驱动方式,适用于数码管位数不多的情况。

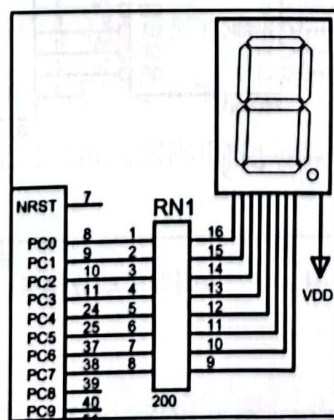


图 3-39 数码管静态显示电路

由于数码管属于电流驱动型元器件，有时为了减轻 STM32 单片机的负荷，可以考虑采用锁存器间接驱动数码管，如图 3-40 所示，采用了 74HC573 锁存器。除了锁存器间接驱动，还可以采用更为直接的做法，即采用硬件译码器，如图 3-41 所示，采用了 74LS247 译码器。关于 74HC573 锁存器与 74LS247 译码器的使用说明，读者可自行查阅相关资料。

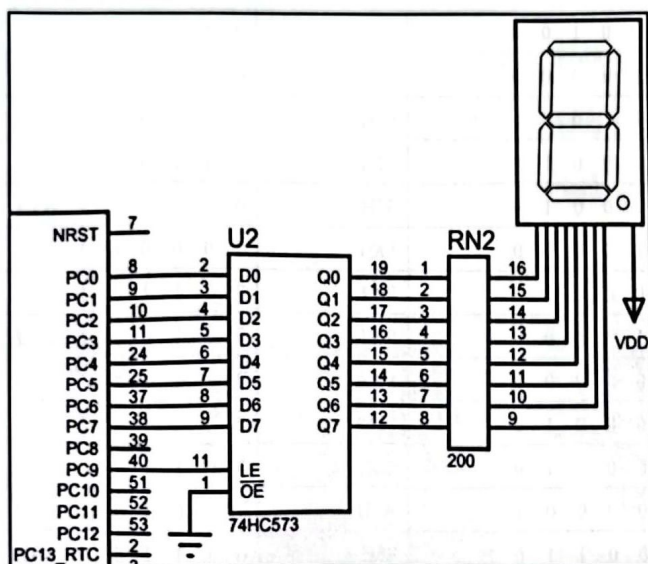


图 3-40 锁存器驱动数码管

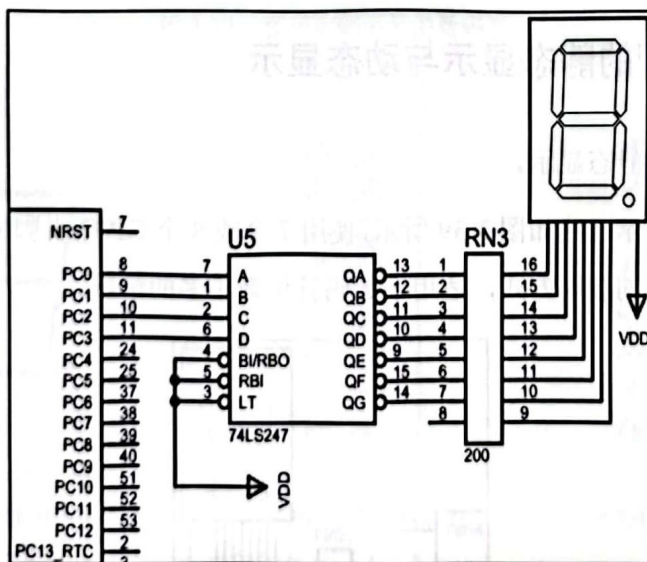


图 3-41 译码器驱动数码管

## (2) 数码管的动态显示。

在实际应用中，有时需要使用多位数码管，为节约 GPIO 引脚，通常选择动态显示



方案。数码管动态显示电路如图 3-36 所示。数码管动态显示即多位数码管的段码输入端共用同一组（7 个或 8 个）GPIO 引脚，使多位数码管按一定顺序（如从左往右）快速轮流显示字符信息的显示方式。人眼存在“视觉暂留”效应，当刷新速度够快时，这种效应会使人产生所有数码管同时发光的错觉。相关实验表明，每一位数码管显示字符的时间以 1ms 左右为宜，多个数码管均能稳定“同时”显示且抖动效果不明显。考虑到 GPIO 引脚输出电流能力有限（见 3.1 节），每一位数码管均由反相器（非门电路）输出高电平选通。

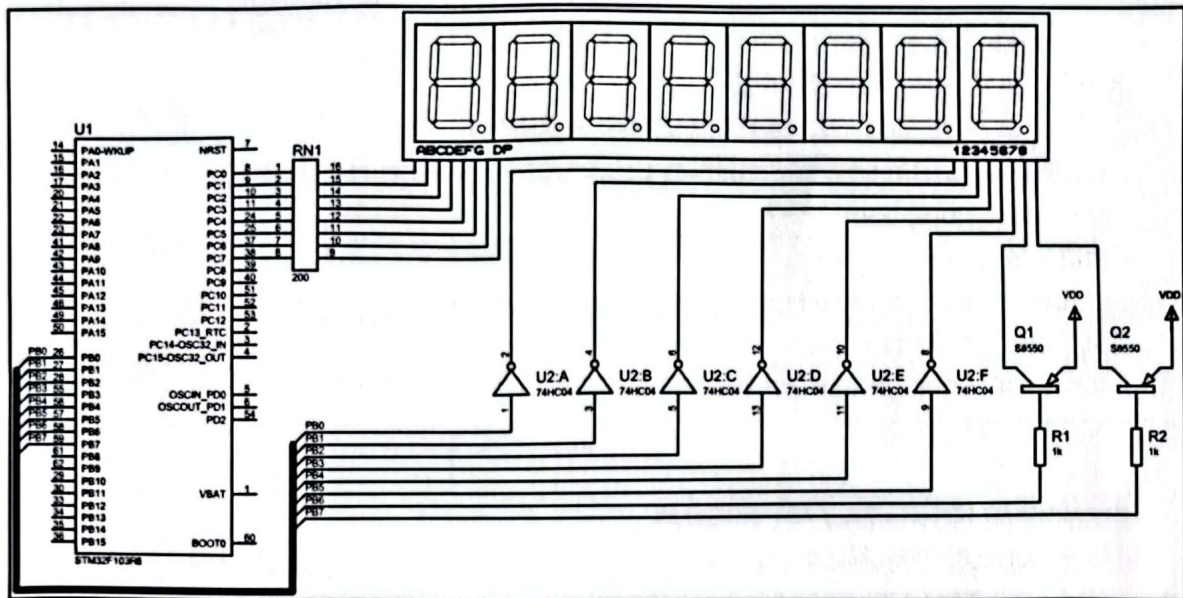


图 3-42 数码管动态显示电路

值得注意的是，图 3-36 中用到的反相器 74HC04 仅有 6 个通道，如果 8 位数码管全部使用反相器输出选通，那么需要使用 2 个 74HC04。出于节约的目的，数码管动态显示电路第 7、8 位数码管的选通实际上使用了 PNP 型三极管 S8550，这里将三极管用作电子开关，如图 3-42 所示。仿真电路之所以全部使用反相器而非三极管，是因为三极管作为电子开关在仿真过程中响应速度太慢，从而导致仿真效果与实际效果相差过大，这也说明了 Proteus 并不能完全仿真实际电路的运行效果。

### 3.4.3 任务程序的编写

main.c 程序：

```
#include "main.h"
/* USER CODE BEGIN PV */
```



```

//共阳极数码管 0~9 的段码
const uint8_t SegmentCodes[] =
{
    0xc0, 0xf9, 0xa4, 0xb0, 0x99,
    0x92, 0x82, 0xf8, 0x80, 0x90
};

/* USER CODE END PV */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
int main(void)
{
    /* USER CODE BEGIN 1 */
    uint8_t i; //循环变量
    uint64_t num=12345678; //需要显示的数字
    uint8_t unit[8]; //显示数字的每一位
    /* USER CODE END 1 */
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        //8 位十进制数字每一位的分解
        unit[7]=num/10000000;
        unit[6]=num%10000000/1000000;
        unit[5]=num%1000000/100000;
        unit[4]=num%100000/10000;
        unit[3]=num%10000/1000;
        unit[2]=num%1000/100;
        unit[1]=num%100/10;
        unit[0]=num%10;
        //通过有限次循环实现数码管动态显示
        for(i=0;i<8;i++)
        {
            //熄灭所有数码管
            LL_GPIO_WriteOutputPort(GPIOB, 0xff);
            //输出段码
            LL_GPIO_WriteOutputPort(GPIOC, SegmentCodes[unit[i]]);
            //选通(点亮)当前位数码管
            LL_GPIO_WriteOutputPort(GPIOB, (0x7f>>i) | (0x7f<<(8-i)));
            //延时 1ms
            HAL_Delay(1);
        }
    }
}

```



```

    }
    /* USER CODE END WHILE */
}
}
.....

```

值得注意的是，当 GPIO 引脚驱动选择使用 LL 库的时候，就不能同时使用 HAL 库，这里我们使用了与流水灯相同的算法来实现数码管的按序选通。

### 3.5 矩阵式键盘

#### 能力目标

理解矩阵式键盘的电路组成及工作原理，掌握矩阵式键盘程序编写方法。

#### 任务目标

4×4 矩阵式键盘仿真电路如图 3-43 所示，要求编程实现当按下任意一个按钮时，数码管立即显示当前按下按钮对应的键值。

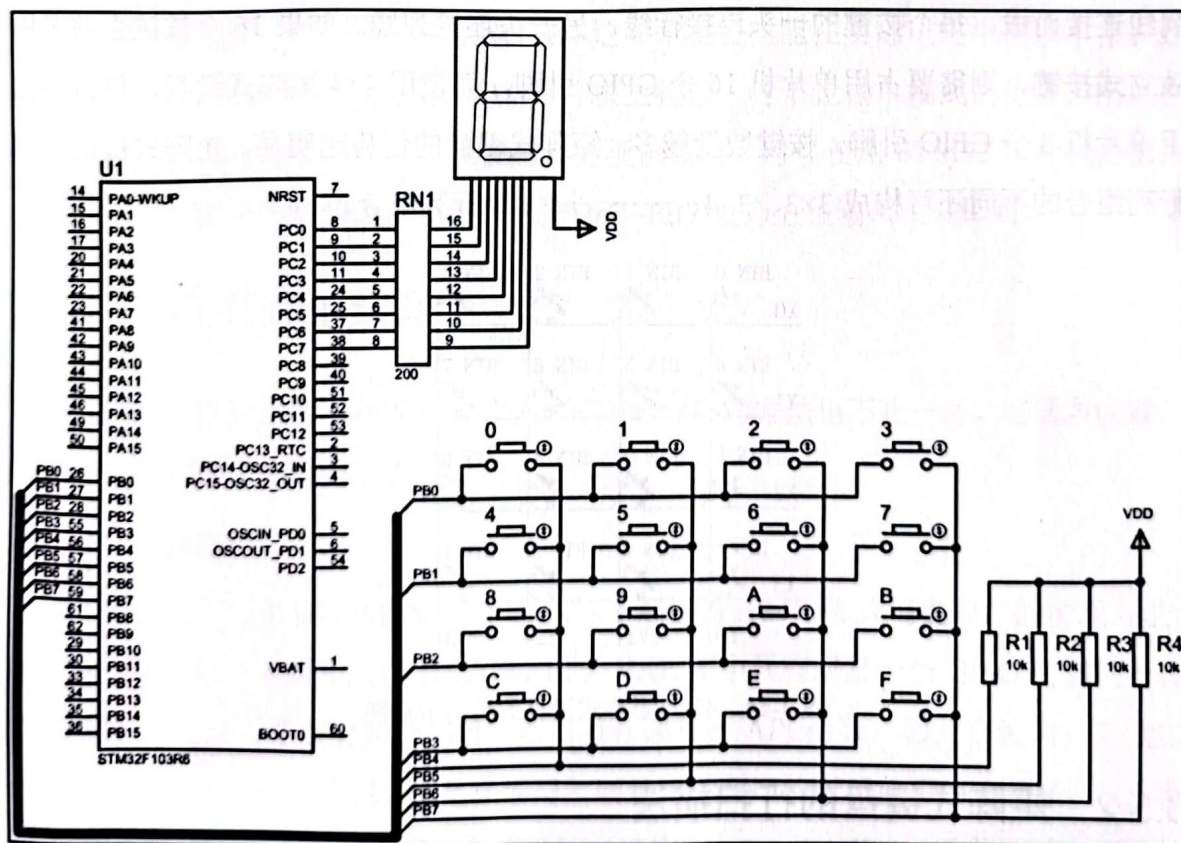


图 3-43 4×4 矩阵式键盘仿真电路