项目二　输入/输出功能

项目描述：我们所熟悉的电脑的输入设备有键盘、鼠标、麦克风等，输出设备有显示器、音响等。如同电脑，输入/输出也是单片机最基本的功能，单片机最常用的输入设备为键盘，最常用的输出设备为发光二极管 LED、数码管以及液晶显示器 LCD。本项目基于 KST-51 开发板，通过编程实现独立按键检测与 LED 灯点亮功能。

2.1　任务一：输出功能——点亮 LED 灯

2.1.1　LED 灯介绍

LED（Light-Emitting Diode），即发光二极管，俗称 LED 小灯，它的种类很多，参数也不尽相同，KST-51 开发板上用的是普通的贴片发光二极管。这种二极管通常的正向导通电压在 1.8V 到 2.2V 之间，工作电流一般在 1mA～20mA 之间。其中，当电流在 1mA～5mA 之间变化时，随着通过 LED 的电流越来越大，人的肉眼会明显感觉到这个小灯越来越亮，而当电流在 5mA～20mA 之间变化时，看到的发光二极管的亮度变化就不是太明显了。当电流超过 20mA 时，LED 就会有烧坏的危险，电流越大，烧坏得也就越快。所以在使用过程中应该特别注意它在电流参数上的设计要求。

LED 驱动电路如图 2.1 所示。若接入的 VCC 电压是 5V，发光二极管自身压降大概是 2V，那么在右边电阻 R34 上承受的电压就是 3V。若要求电流范围是 1mA～20mA 的话，就可以根据欧姆定律 $R=U/I$，把这个电阻的上限值和下限值求出来。U=3V，当电流是 1mA 的时候，电阻值是 3kΩ；当电流是 20mA 的时候，电阻值是 150Ω，也就是说 R34 的取值范围是 150Ω～3kΩ。这个电阻值大小的变化，可以直接限制整条通路的电流的大小，因此这个电阻通常称为“限流电阻”。图 2.1 中用的电阻是 1kΩ，可以计算出流过 LED 的电流大约为 3mA。

将图 2.1 变换一下，用一个单片机的 I/O 口来驱动 LED，有两种方式，如图 2.2 和图 2.3 所示。

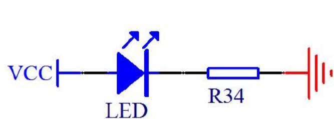

图 2.1　LED 驱动电路（一）

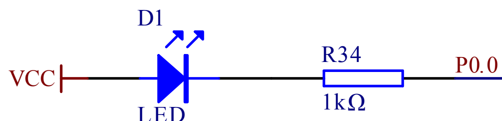

图 2.2　LED 驱动电路（二）

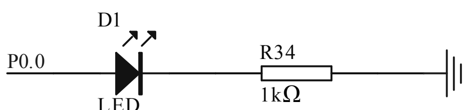

图 2.3　LED 驱动电路（三）

图 2.2 中，如果单片机 P0.0 引脚输出一个低电平，也就是跟 GND 一样的 0V 电压，就可以让 LED 发光；如果 P0.0 引脚输出一个高电平，就是跟 VCC 一样的+5V 电压，那么这个时候，左侧 VCC 电压和右侧 P0.0 的电压是一致的，就没有电压差，不会产生电流，因此 LED 灯不会亮，处于熄灭状态。由于单片机是可以编程控制的，因此可以通过编程使 P0.0 端口输出高电平或低电平，从而控制 LED 灯的亮与灭。同理，图 2.3 中，P0.0 引脚输出一个高电平，就可以让 LED 发光，输出一个低电平，就熄灭 LED。下面通过编程软件来控制 LED 灯的亮和灭。

2.1.2 任务实施

1．编程语言介绍

单片机的开发语言有两种：汇编语言和 C 语言，汇编语言是一种用文字助记符来表示机器指令的符号语言，是最接近机器码的一种语言。其主要优点是占用资源少、程序执行效率高。但是对于不同的 CPU，其汇编语言可能有所差异，所以不易移植。

C 语言是一种编译型程序设计语言，它兼顾了多种高级语言的特点，并具备汇编语言的功能，C 语言有功能丰富的库函数，运算速度快，编译效率高，具有良好的可移植性，可以直接实现对系统硬件的控制。C 语言是一种结构化程序设计语言，它支持当前程序设计中广泛采用的自顶向下结构化程序设计技术。此外，C 语言程序具有完善的模块程序结构，从而为软件开发中采用模块化程序设计方法提供了有力的保障。因此，使用 C 语言进行程序设计已成为软件开发的一个主流。用 C 语言来编写目标系统软件，会大大缩短开发周期，且明显地增加了软件的可读性，便于改进和扩充，从而可研制出规模更大、性能更完备的系统。

相比较来说，汇编语言比较接近单片机的底层，使用汇编语言有助于理解单片机内部结构。简单的程序用汇编语言开始，程序效率可能比较高，但是当程序容量达到成千上万行以后，汇编语言在组织结构、修改维护等方面就非常困难了，此时 C 语言就有不可替代的优势了。所以实际开发过程中，目前至少 90%以上的工程师都在用 C 语言做单片机开发，只有在很低端的应用中或者是特殊要求的场合，才会用汇编语言开发，所以本教材中所有项目开发都是用 C 语言。

2．特殊功能寄存器和位定义

用 C 语言来对单片机编程，要了解单片机特殊的独有的几条编程语句，51 单片机也有，下面先介绍两条。

第一条语句是：sfr P0 = 0x80;

sfr 这个关键字，是 51 单片机特有的，它的作用是定义一个单片机特殊功能寄存器 SFR（Special Function Register）。51 单片机内部有很多个小模块，每个模块在存储器中都有一个唯一的地址，同时每个模块都有 8 个控制开关。如：P0 是一个功能模块，在 RAM 中的地址为 0x80，通过设置 P0 内部这个模块的 8 个开关，来让单片机的 P0 这 8 个 I/O 口输出高电平或者低电平。而 51 单片机内部有很多寄存器，如果想使用的话必须提前进行 sfr 声明。不过 Keil 软件已经把所有这些声明都预先写好并保存到一个专门的文件中去了，要使用的话只要在文件开头添加一行#include <reg52.h>即可。

第二条语句是：sbit LED = P0^0;

sbit，就是对 SFR 的 8 个开关中的一个进行定义。经过上面第二条语句后，以后只要在程

序里写 LED，就代表了 P0.0 口，注意这里 P 必须大写，这也就相当于给 P0.0 又取了一个更形象的名字叫做 LED。

了解了这两个语句后，来看一下单片机的特殊功能寄存器，如图 2.4 所示。需要注意是，每个型号的单片机都会配有生产厂商所编写的数据手册（Datasheet），打开 STC89C52 的数据手册，从 21 页到 24 页，全部是对特殊功能寄存器的介绍以及地址映射列表。在使用这个寄存器之前，必须对这个寄存器的地址进行说明。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
P0	80h	8-bit Port 0	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	1111,1111
P1	90h	8-bit Port 1	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	1111,1111
P2	A0h	8-bit Port 2	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	1111,1111
P3	B0h	8-bit Port 3	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	1111,1111
P4	E8h	4-bit Port 4	-	-	-	-	P4.3	P4.2	P4.1	P4.0	xxxx,1111

图 2.4 I/O 口特殊功能寄存器

P0 口地址是 0x80，一共有从 7 到 0 共 8 个 I/O 口控制位，后边有个 Reset Value（复位值），这个很重要，是对寄存器必看的一个参数，8 个控制位复位值全部都是 1。也就是说，当单片机上电复位的时候，所有的引脚的值默认都是 1，即高电平，在设计电路的时候要充分考虑这个问题。

上边那两条语句，写 sfr 的时候，必须要根据手册里相应地址（Add）去写，写 sbit 的时候，就可以直接将一个字节其中某一位取出来。在编程的时候，也有现成的写好寄存器地址的头文件，直接包含该头文件就可以了，不需要逐一去写了。

3．C 语言变量类型和范围

C 语言的数据变量基本类型分为字符型、整型、长整型及浮点型，如表 2.1 所示。

表 2.1 C 语言基本类型

基本类型	子类型	取值范围
字符型	unsigned char	0～255
	signed char	-128～127
整型	unsigned int	0～65535
	signed int	-32768～32767
长整型	unsigned long	0～4294967295
	signed long	-2147483648～2147483647
浮点型	float	-3.4×10^{-38}～3.4×10^{-38}
	double	（C51 里等同于 float）

表 2.1 中有四种基本类型，每个基本类型又包含了两个类型。字符型、整型、长整型，除了可表达的数值大小范围不同之外，都是只能表达整数，而 unsigned 型的又只能表达正整数，要表达负整数则必须用 signed 型，如要表达小数的话，则必须用浮点型。

在程序中定义变量时一定要注意变量的取值范围，变量类型使用不当有时会有预料不到

的后果。这里有一个编程宗旨，就是能用小不用大。就是说定义能用 1 个字节 char 解决问题的，就不定义成 int，一方面，节省了 RAM 空间可以让其他变量或者中间运算过程使用，另一方面，占空间小，程序运算速度也快一些。

4. *程序编写*

下面用 C 语言编写程序点亮 LED 灯。

```
#include <reg52.h>          //包含特殊功能寄存器定义的头文件
sbit LED = P0^0;            //位地址声明，注意：sbit 必须小写、P 大写！
void main()                 //任何一个 C 程序都必须有且仅有一个 main 函数
{                           //{}是成对存在的，在这里表示函数的起始和结束
    LED = 0;                //分号表示一条语句结束
}
```

先从程序语法上来分析一下。

①main 是主函数的函数名字，每一个 C 程序都必须有且仅有一个 main 函数。

②void 是函数的返回值类型，本程序没有返回值，用 void 表示。

③{}在这里是函数开始和结束的标志，不可省略。

④每条 C 语句都是以“;”结束的，并且“;”是通过英文输入法输入的，如果是中文输入法，编译时会报错。

从逻辑上来看，程序这样写就可以了，但是在实际单片机应用中，存在一个问题。比如程序空间可以容纳 100 行代码，但是实际上的程序只用了 50 行代码，当运行完了 50 行，再继续运行时，第 51 行的程序不是我们想运行的程序，而是不确定的未知内容，一旦执行下去程序就会出错从而可能导致单片机自动复位，所以通常在程序中加入一个死循环，让程序停留在希望的这个状态下，不要乱运行，有以下两种写法可以参考：

参考程序一：

```
#include <reg52.h>
sbit LED = P0^0;
void main()
{
    while(1)
    {
        LED = 0;
    }
}
```

参考程序二：

```
#include <reg52.h>
sbit LED = P0^0;
void main()
{
    LED = 0;
    while(1);
}
```

程序一的功能是程序在反复不断地无限次执行“LED = 0;”这条语句，而程序二的功能是“LED=0;”语句执行一次，然后程序直接停留下来等待，相对程序一来说程序二更加简洁一些。针对图 2.2，这两个程序都能够把 LED 灯点亮，但是这两个程序却都点不亮 KST-51 板子上的小灯，这是为什么呢？

单片机程序开发，实际上算是硬件底层驱动程序开发，这种程序的开发，是离不开电路图的，必须根据电路图来进行程序的编写。如果设计电路板的电路图和图 2.2 一样的话，程序可以成功点亮 LED 灯，但是如果不一样，就可能点不亮。

KST-51 开发板上，还有一个 74HC138 作为 8 个 LED 灯的总开关，而 P0.0 仅仅是个分开

关。如同家里总是有一个供电总闸，然后每个电灯又有一个专门的开关，刚才的程序仅仅打开了那个电灯的开关，但是没有打开那个总电闸，所以程序需要加上这部分代码。

开发板上 LED 灯的硬件电路如图 2.5 所示，74HC138 电路如图 2.6 所示，分析可知：若要点亮 LED2，必须使得 DB0 端口（通过锁存器 74HC245 连接至单片机 P0.0 端口）输出低电平，同时 Q16 的三极管 9012 导通，即 LEDS6 端口输出低电平，而 LEDS6 接至 74HC138 的 $\overline{Y6}$ 端，$\overline{Y6}$ 端输出低电平的条件是 74HC138 正常工作（$\overline{E1}$、$\overline{E2}$ 端为低电平，E3 端为高电平）且 A2、A1、A0 端口电平分别为 1、1、0，因此，程序初始化时应将 ENLED 置 0，ADDR3 置 1，ADDR2 置 1，ADDR1 置 1，ADDR0 置 0。

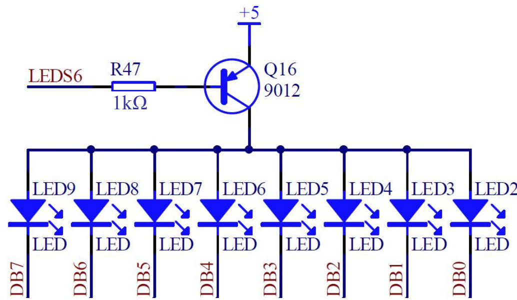

图 2.5　LED 驱动电路（四）

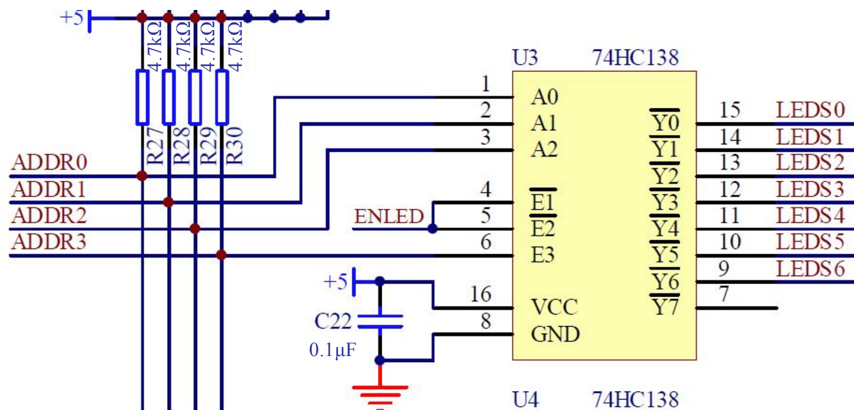

图 2.6　74HC138 连接电路

需要注意的是，ADDR0～ADDR3 这四个端口并不是直接接到 P1.0～P1.3 端口的，接口电路如图 2.7 所示，P1.0～P1.3 端口为显示译码与步进电机的复用端口，若要点亮 LED 灯（即 P1.0～P1.3 口作显示译码端口用），需将板子上 J13～J16 这四个端子上的跳线帽都拨到右边两个端口，即将 ADDR0～ADDR3 分别接 P1.0～P1.3 端口。

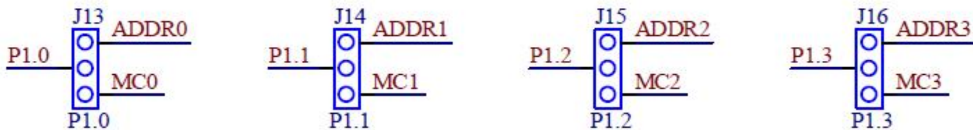

图 2.7　显示译码与步进电机的跳线选择

跳线帽插好以后，下面编写点亮板子上最右边 LED 小灯的程序：

```
#include <reg52.h>      //包含特殊功能寄存器定义的头文件
sbit LED = P0^0;        //位地址声明，注意：sbit 必须小写、P 大写！
sbit ADDR0 = P1^0;
sbit ADDR1 = P1^1;
sbit ADDR2 = P1^2;
sbit ADDR3 = P1^3;
sbit ENLED = P1^4;
void main()
{
ENLED = 0;              //使能 74HC138
ADDR3 = 1;              //使能 74HC138
                        //以下三条语句使得 74HC138 的 Y6 引脚输出低电平，从而导通 Q16
ADDR2 = 1;
ADDR1 = 1;
ADDR0 = 0;
LED = 0;                //点亮小灯
while (1);              //程序停止在这里
}
```

按项目一中任务二介绍的方法新建工程，将上述代码输入 Keil 软件程序代码区域，并编译生成 Hex 文件，下面介绍如何将 Hex 文件烧录到单片机中。

5. 程序下载

首先，要把硬件连接好，把板子插到电脑上，打开设备管理器查看所使用的是哪个 COM 口，如图 2.8 所示，找到“USB-SERIAL CH340（COM3）”这一项，这里最后的数字就是开发板目前所使用的 COM 端口号。注意要先在电脑上安装 USB 转串口的驱动程序才能看到相应的 COM 端口，否则如图 2.9 所示，将无法下载程序。

图 2.8　查看 COM 端口（一）

图 2.9　查看 COM 端口（二）

然后打开 STC 系列单片机的下载软件——STC-ISP，如图 2.10 所示。

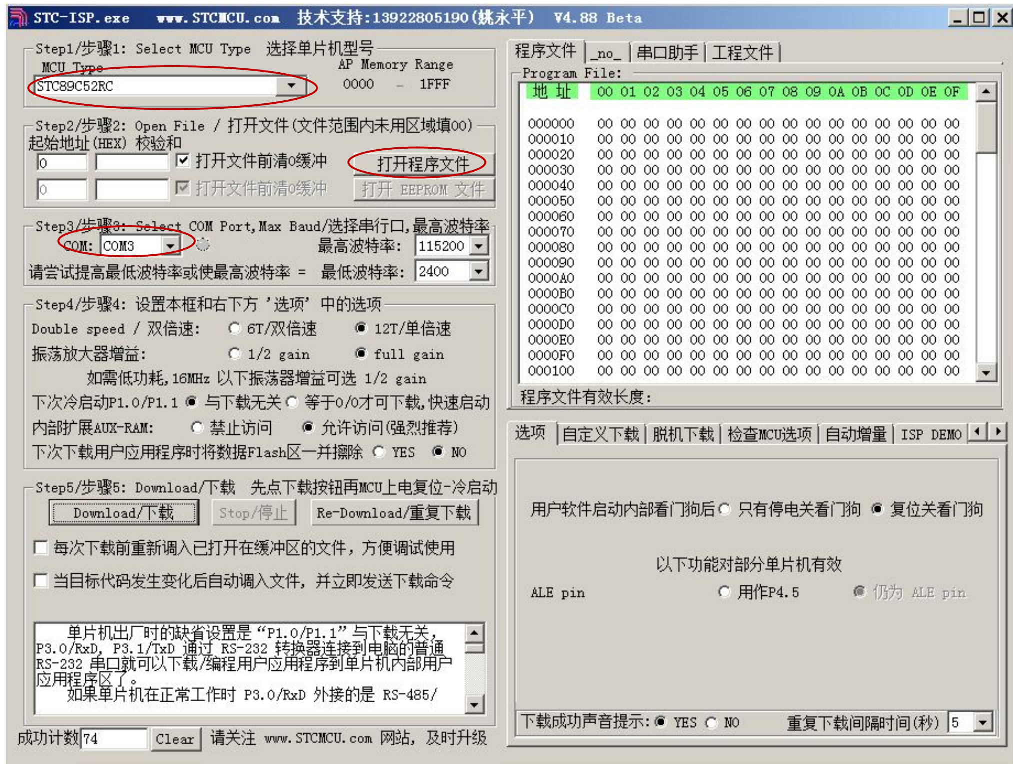

图 2.10　程序下载设置

下载软件烧录程序分五个步骤：步骤 1，选择单片机型号，开发板上用的单片机型号是 STC89C52RC，这个一定不能选错了；步骤 2，单击“打开程序文件”，找到刚才建立的工程文件夹，找到之前编译生成的 Hex 文件 LED.hex，单击打开；步骤 3，选择刚才查到的 COM 口，波特率（baud rate）使用默认的就行；步骤 4，这里的所有选项都使用默认设置，不要随便更改，有的选项改错了以后可能会产生麻烦；步骤 5，因为 STC 单片机要冷启动下载，就是先下载，然后再给单片机上电，所以要先关闭板子上的电源开关，然后单击“Download/下载”按钮，等待软件提示请上电后，如图 2.11 所示，再按下板子的电源开关，就可以将程序下载到单片机里边了。当软件显示“已加密”就表示程序下载成功了，如图 2.12 所示。

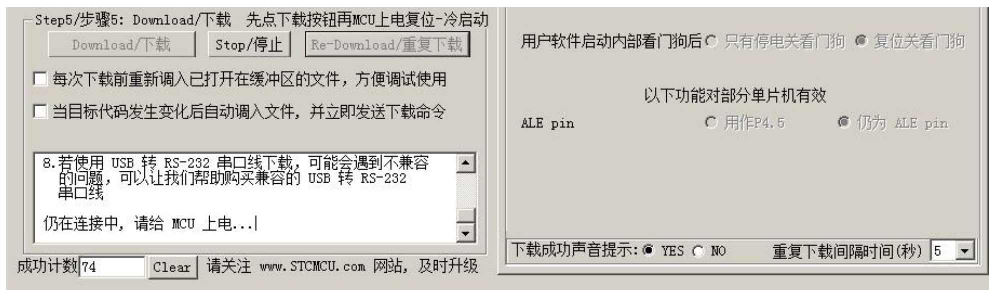

图 2.11　程序下载过程

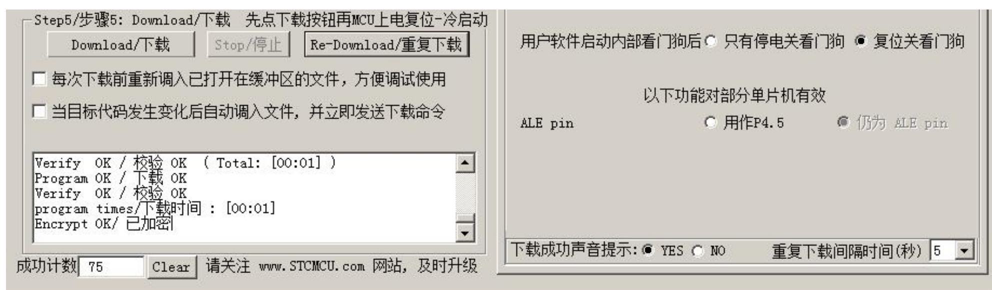

图 2.12　程序下载完毕

程序下载完毕后，会自动运行，大家可以在板子上看到那一排 LED 中最右侧的小灯已经发光了。现在如果我们把 LED=0 改成 LED=1，再重新编译程序下载进去新的 Hex 文件，灯就会熄灭。

2.2　任务二：输入功能——按键检测

2.2.1　键盘介绍

在单片机应用系统中，键盘主要用于向计算机输入数据、传送命令等，是人工干预计算机的主要手段。键盘要通过接口与单片机相连，分为编码键盘和非编码键盘两类。

键盘上闭合键的识别由专用的硬件编码器实现，并产生键编码号或键值的称为编码键盘，如计算机键盘。而靠软件编程来识别的称为非编码键盘，在单片机组成的各种系统中，使用最广泛的是非编码键盘。当然，也有用到编码键盘的。

非编码键盘分为：独立键盘（如图 2.13 所示）和行列式（又称为矩阵式）键盘（如图 2.14 所示）。

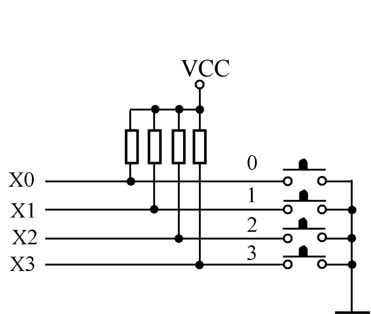

图 2.13　独立键盘

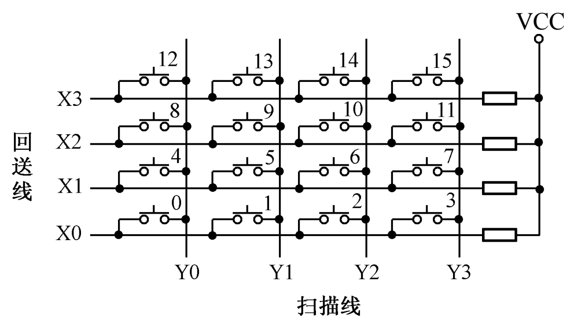

图 2.14　矩阵键盘

独立键盘每个键相互独立，各自与一条 I/O 线相连，CPU 可直接读取该 I/O 线的高/低电平状态。其优点是硬件、软件结构简单，判键速度快，使用方便；缺点是占 I/O 口线多。它多用于设置控制键、功能键，适用于键数少的场合。

矩阵键盘的键按矩阵排列，各键处于矩阵行/列的结点处，CPU 通过对连在行（列）的 I/O 线读取已知电平的信号，然后读取列（行）线的状态信息，逐线扫描，得出键码。其特点是键

多时占用 I/O 口线少，硬件资源利用合理，但判键速度慢。它多用于设置数字键，适用于键数多的场合。

本节以独立键盘为例介绍单片机的输入功能，矩阵键盘在后续章节再做介绍。图 2.13 中，4 条输入线接到单片机的 I/O 口上，当按键 0 按下时，+5V 通过与 X0 端相连的电阻，然后再通过按键 0 最终进入 GND 形成一条通路，那么这条线路的全部电压都加到了电阻上，X0 这个引脚就是低电平。当松开按键后，线路断开，就不会有电流通过，那么 X0 和+5V 就应该是等电位，是一个高电平。我们就可以通过 X0 这个 I/O 口的高低电平来判断是否有按键被按下。

2.2.2 MCS-51 单片机并行 I/O 接口结构

在 2.1 节中，I/O 口作为输出口时，只需要在程序中将 P0.0 端口设为低电平即可点亮 LED 灯，将 P0.0 端口设为高电平即可熄灭 LED 灯。STC89C52 单片机的输入功能比输出功能稍微复杂一些，在使用之前需进行一些设置，否则有可能无法准确识别输入端口电平。下面分别介绍单片机的 4 个 8 位并行 I/O 接口。

1. P0 口

图 2.15 所示为 P0 口的位结构图，它由一个输出锁存器、两个三态缓冲器、一个输出驱动电路和一个输出控制电路组成。其中，输出驱动电路由一对场效应管组成，其工作状态受输出控制电路控制。

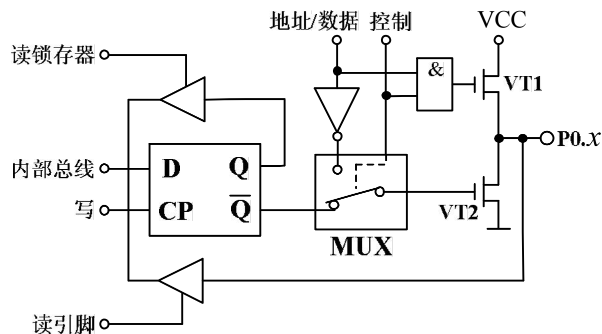

图 2.15 P0 口结构

当从 P0 口输出地址/数据信息时，控制信号应为高电平 1，模拟转换开关（MUX）把地址/数据信息经反相器与下拉场效应管 VT2 接通，同时打开输出控制电路的与门。输出的地址/数据信息通过与门驱动反相器与上拉场效应管 VT1，又通过反相器驱动 VT2。例如，若地址/数据信息为 0，则该 0 信号一方面通过与门使 VT1 截止，另一方面经反相器使 VT2 导通，从而使引脚上输出相应的 0 信号；反之，若地址/数据信息为 1，将使 VT1 导通而使 VT2 截止，引脚上将输出相应的 1 信号。

若 P0 口作为通用 I/O 接口使用，在 CPU 向接口输出数据时，对应的输出控制信号应为 0 信号，MUX 将把输出级与锁存器的 $\overline{Q}$ 端接通。同时，由于与门输出为 0，上拉场效应管 VT1 处于截止状态，因此输出级是漏极开路电路。这样，当写脉冲加在触发器的时钟端 CP 上时，则与内部总线相连的 D 端数据取反后就出现在触发器的 $\overline{Q}$ 端，再经过场效应管反相，在 P0 引

脚上出现的数据正好对应于 CPU 内部总线的数据。

当 P0 口作为通用 I/O 口使用时，如果从 P0 口输入数据，则此时上拉场效应管一直处于截止状态。引脚上的外部信号既加在下面一个三态缓冲器的输入端，又加在下拉场效应管的漏极。假定在此之前曾输出锁存数据 0，则下拉场效应管是导通的。这样 P0 引脚上的电位就始终被嵌位在 0 电平，使输入高电平无法读入。因此，P0 作为通用 I/O 接口使用时是准双向口，即输入数据时，应先向 P0 口写 1，使两个场效应管均截止，然后方可作为高阻抗输入。

综上所述，P0 口既可作为地址/数据总线口使用，又可作为通用 I/O 口使用，可驱动 8 个 LS 型 TTL 负载。在访问外部存储器时，P0 口作为地址/数据总线复用口，是双向口，分时送出地址的低 8 位和发送/接收相应存储单元的数据。作为通用 I/O 接口使用时，P0 口是漏极开路的准双向口，需要在外部引脚处接上拉电阻。

图 2.16 是 P0 口输出测试电路，由 2.1 节可知，只要在程序中令 P0.0 端口输出一个高电平 1 就可以点亮 LED 灯，但是实际上在仿真时 P0.0 端口的灯并不能亮，为什么呢，大家思考一下有什么解决办法？

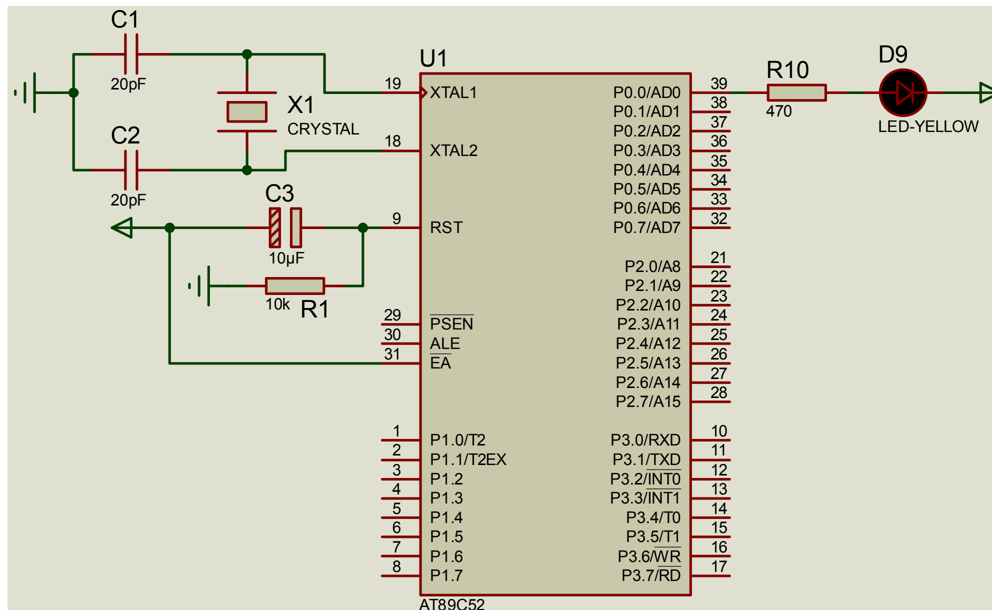

图 2.16 P0 口输出测试电路

2. P2 口

图 2.17 所示为 P2 口的位结构图，它与 P0 口基本相同，为了使逻辑上一致，将锁存器的 Q 端与输出场效应管相连。只是输出部分略有不同，P2 口在输出场效应管的漏极上接有上拉电阻，这种结构不必外接上拉电阻就可以驱动任何 MOS 负载，且只能驱动 4 个 LS 型 TTL 负载。P2 口常用作外部存储器的高 8 位地址口。当不用作地址接口时，P2 口也可作为通用 I/O 口使用，这时它是准双向 I/O 接口。

3. P1 口

图 2.18 所示为 P1 口的位结构图，它与 P2 口基本相同，只是少了一个模拟转换开关（MUX）和一个反相器，无选择电路，且为保持逻辑上的一致，将锁存器的 $\overline{Q}$ 端与输出场效应管相连。

输出场效应管的漏极上接有上拉电阻，不必外接上拉电阻就可以驱动任何 MOS 负载，带负载能力与 P2 口相同，只能驱动 4 个 LS 型 TTL 负载。P1 口常用作通用 I/O 接口，是准双向 I/O 接口，作为输入口使用时必须先将锁存器置 1，使输出场效应管截止。

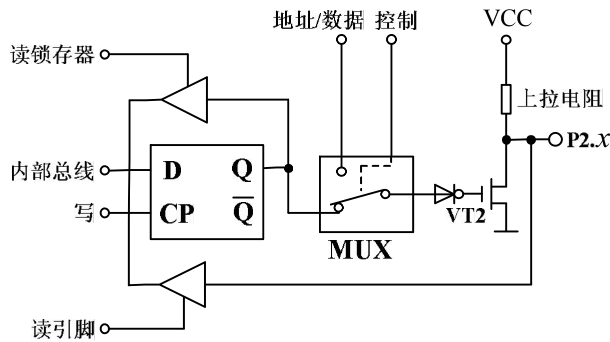

图 2.17　P2 口结构

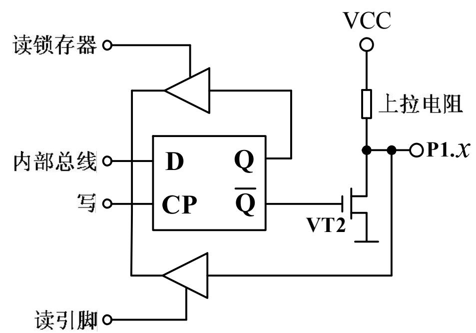

图 2.18　P1 口结构

4. P3 口

图 2.19 所示为 P3 口的位结构图，它是双功能口，第一功能与 P1 口一样可用作通用 I/O 接口，也是准双向 I/O 接口。另外，它还具有第二功能。其结构特点是不设模拟开关（MUX），增加了第二功能控制逻辑，多增设一个与非门和缓冲器，内部具有上拉电阻。

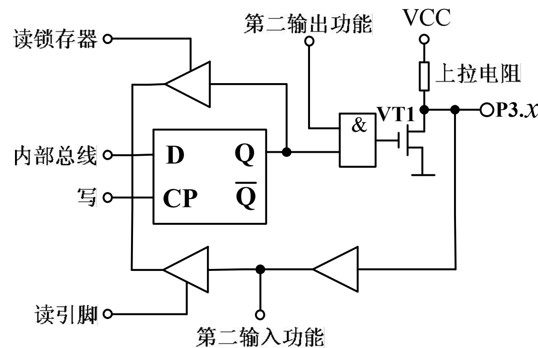

图 2.19　P3 口结构

P3 口作为通用输出口使用时，内部第二功能线应为高电平 1，以保证与非门的畅通，维持从锁存器到输出口数据输出通路畅通无阻，锁存器的内容经 Q 端输出。此时 P3 口的功能和带负载能力与 P1 口相同。P3 口作为第二功能输出口时，锁存器应置高电平 1，保证与非门对第二功能信号的输出是畅通的，从而实现内部第二输出功能的数据经与非门从引脚输出。

P3 口作为输入口使用时，对于第二功能为输入的信号引脚，在 I/O 接口上的输入通路增设了一个缓冲器，输入的第二功能信号即从这个缓冲器的输出端取得。而 P3 口作为通用 I/O 接口输入端时，信号取自三态缓冲器的输出端。因此，无论通用 I/O 接口的输入还是内部第二功能的输入，锁存器的输出端 Q 和内部第二功能线均应置为高电平 1，使与非门输出为 0，这样，驱动电路不会影响引脚上外部数据的正常输入。P3 口工作在第二功能时各引脚定义见表 2.2。

表 2.2　P3 口工作在第二功能时各引脚定义表

引脚	功能	引脚	功能
P3.0	串行数据接收口（RXD）	P3.4	定时器/计数器 0 的外部输入口（T0）
P3.1	串行数据发送口（TXD）	P3.5	定时器/计数器 1 的外部输入口（T1）
P3.2	外部中断 0（$\overline{INT0}$）	P3.6	外部 RAM 写选通信号（$\overline{WR}$）
P3.3	外部中断 1（$\overline{INT1}$）	P3.7	外部 RAM 读选通信号（$\overline{RD}$）

2.2.3　独立按键扫描

单独的按键扫描程序执行后看不到任何现象，为了有个直观的效果，可以将之前的点亮 LED 灯的程序加进来，当 K1 键按下时点亮一个 LED 灯（如板子最右侧的 LED2）。下面围绕这一思路来分析软件代码如何编写。

1．构建独立按键

由于开发板上没有独立按键，只有一个 4*4 的矩阵键盘，如图 2.20 所示，如何将矩阵键盘变为独立按键呢？

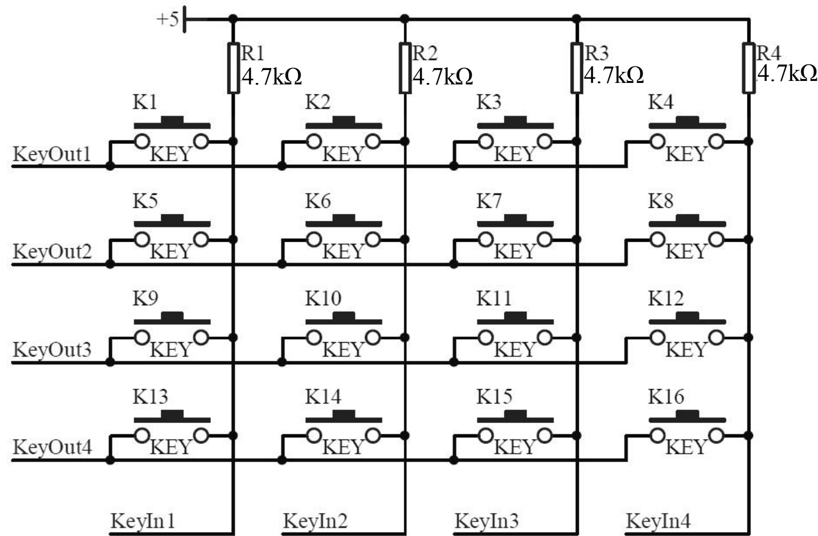

图 2.20　4*4 矩阵键盘

对比独立按键电路和矩阵键盘电路可知，若要将 K1 变为独立按键，只需将 KeyOut1 端接地即可，因此，只要将单片机的 P2.3（KeyOut1 接至 P2.3 端口）输出低电平，通过检测单片机的 P2.4（KeyIn1 接至 P2.4）端口电平状态来判断按键是否按下，就可以将 K1 看成是一个独立按键。

2．独立式按键的软件设计

在单片机应用系统中主程序一般是循环结构，单片机按键控制系统的主程序流程图如图 2.21 所示。

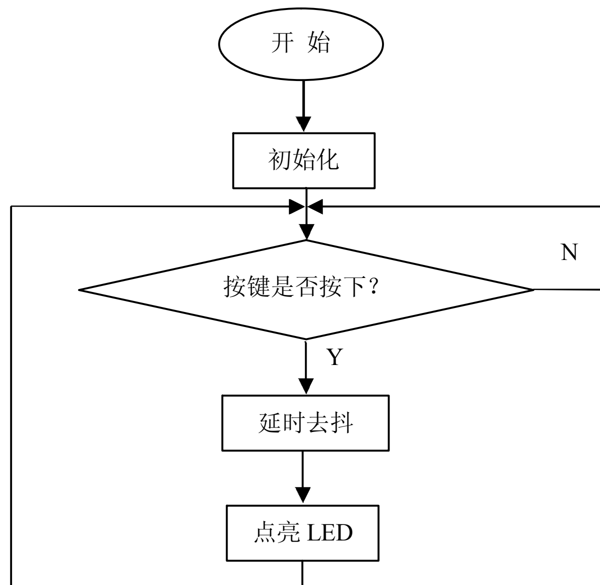

图 2.21 主程序流程图

3．按键消抖

在键盘的软件设计中还要注意按键的去抖动问题。由于按键一般是由机械式触点构成的，在按键被按下和断开的瞬间均有一个抖动过程，时间大约为 5ms～10ms，这可能会造成单片机对按键的误识别。物理按键抖动如图 2.22 所示。

按键消抖一般有两种方法，即硬件消抖和软件消抖。硬件消抖方法的原理如图 2.23 所示。

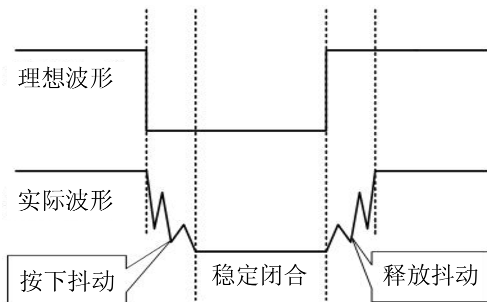

图 2.22 物理按键抖动波形图

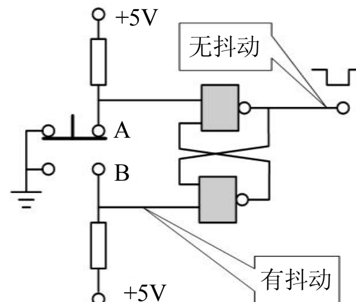

图 2.23 硬件消抖方法

在软件设计中，当单片机检测到有键按下时，可以先延时一段时间越过抖动过程再对按键识别。

实际应用中，一般希望按键一次按下且单片机只处理一次，但由于单片机执行程序的速度很快，按键一次按下可能被单片机多次处理。为避免此问题，可在按键第一次按下时延时 10ms 之后再检测按键是否按下，如果此时按键仍然按下，则确定有按键输入。这样便可以避免按键的重复处理。软件消抖流程图如图 2.24 所示。

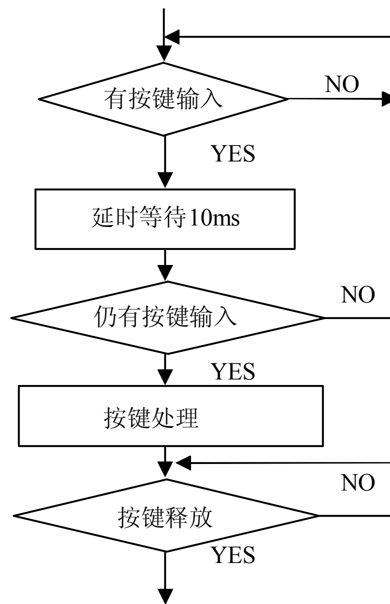

图 2.24 软件消抖流程图

2.2.4 任务实施

```
#include<reg52.h>                          //包含特殊功能寄存器定义的头文件

sbit LED0=P0^0;                            //位地址声明
sbit ADDR0=P1^0;
sbit ADDR1=P1^1;
sbit ADDR2=P1^2;
sbit ADDR3=P1^3;
sbit ENLED=P1^4;
sbit KeyIn1=P2^4;
sbit KeyOut1=P2^3;

void delay_ms(unsigned int cnt)            //延时函数定义
{
    unsigned char i;
    while(cnt--)
    {
        for(i=0; i<=110; i++);
    }
}

void main()                                //主程序
{
```

```
        KeyIn1 = 1;                         //向输入端口写 1，为输入做准备
        KeyOut1 = 0;                        //将 K1 作为独立按键使用
        ENLED = 0;
        ADDR3 = 1;
        ADDR2 = 1;
        ADDR1 = 1;
        ADDR0 = 0;
        while(1)
        {
            if(KeyIn1 == 0)                 //判断 K1 键是否按下
            {
                delay_ms(10);               //延时去抖
                if(KeyIn1 == 0)
                {
                    LED0 = 0;               //点亮 LED 灯
                    while(KeyIn1==0);       //等待按键释放
                }
            }
        }
}
```

程序写完以后，按照 Keil 写程序的过程操作：建立工程→保存工程→建立文件→添加文件到工程→编写程序→编译→下载程序。程序下载完成以后，可以发现，按下 K1 键，开发板上最右侧的 LED 灯（LED2）点亮。

思考：

1．编程实现 8 个 LED 灯循环点亮（即 LED 流水灯）功能。

2．编程实现如下功能：第一次按 K1 键，LED2 亮，第二次按 K1 键，LED2 灭，如此反复。