教学情境三　四路数字显示抢答器的设计

问题引入

在现代社会中，电子产品无处不在，种类繁多，功能日益强大。在设计这些产品时，一个至关重要的问题是它们的交互设计，尤其是按键的设计。按键作为用户与设备之间交互的桥梁，直接影响用户的使用体验。本教学情境旨在介绍如何通过单片机检测按键的输入，以实现人机交互的控制要求。

本教学情境通过三个任务详细介绍按键设计过程中的相关知识和技能要求，包括独立按键的检测、矩阵按键的检测和综合实训，使学生能循序渐进地掌握按键的设计原理，进而灵活掌握单片机 I/O 口的输入电平检测功能。

知识目标

1．掌握独立按键的检测原理和设计方法。

2．掌握按键的消抖原理和消抖方法。

3．掌握矩阵按键的检测原理和设计方法。

4．了解独立按键和矩阵按键的区别。

技能目标

1．能够设计独立按键检测电路及检测程序。

2．能够设计矩阵按键检测电路及检测程序。

3．能够应用单片机 I/O 口的输入电平检测功能设计综合电路和控制程序。

任务 3-1　独立按键的检测

工作任务

随着现代电子技术的不断发展和普及，越来越多的人机交互应用出现，为我们的生活带来了极大的便利和改变，人们对于简单、直观的操作体验有了更高的期待。独立按键的设计正是在这样的背景下应运而生的。本任务学习如何通过独立按键控制其他外设的运行。

本任务以 IAP15L2K61S2 为主控芯片设计单片机控制电路并编程，实现用一个独立按键控制一只 LED 的亮灭。

思路指导

查阅相关数据手册，了解单片机 I/O 口输入模式的设置方法；学习按键的特性，结合单片机 I/O 口的输入特性，实现按键的检测。

相关知识

独立按键

1．独立按键

在单片机的外围电路中，通常用到的开关都是机械开关。当开关闭合时，线路导通；当开关断开时，线路断开。本任务使用按键作为开关，按下按键时电路闭合，松开按键后电路自动断开。按键结构图如图 3-1 所示。其中，3、4 脚和 1、2 脚为内部短路连接，切记不可将此两组引脚作为开关。本任务使用按键控制 LED 的亮灭，通常由按键设计的键盘分为：独立键盘和矩阵键盘。

图 3-1　按键的结构图

每个按键单独占用一个 I/O 口，I/O 口的高低电平反映了对应按键的状态。单片机独立按键的检测原理：单片机的 I/O 口既可用作输出口，也可用作输入口，检测按键时利用的是它的输入功能。按键的一端接地，另一端与单片机的某个 I/O 口相连，同时在该引脚接入上拉电阻（STC15 单片机 I/O 口在准双向口模式下内部已经弱上拉，因此该电阻可以省略），即开始时先将该引脚置为高电平，然后利用单片机程序不断检测该引脚的

电平是否变为低电平。当按键被按下时，该引脚通过按键与地相连，变为低电平。程序一旦检测到 I/O 口变为低电平，就说明按键被按下，然后执行相应的程序。按键与 I/O 口的连接示意图如图 3-2 所示。

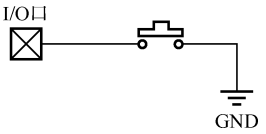


图 3-2　按键与 I/O 口的连接示意图

独立按键的检测流程如下。

（1）查询是否有按键被按下。

（2）查询哪个按键被按下。

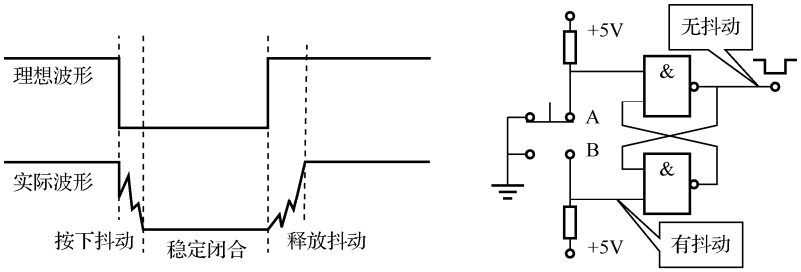
（3）执行被按下按键相应的处理程序。

2．按键消抖的原理

按键被按下时，其触点电压变化如图 3-3（a）所示。从图 3-3（a）中可以看到，理想波形与实际波形之间是有区别的：实际波形在按下和释放的瞬间都有抖动现象。其原因是按键触点松开时，机械弹性装置会将触点弹回原位置。然而，在弹回过程中，由于机械弹性装置的特性，触点可能会在短时间内多次触发信号，导致按键抖动，抖动时间的长短和按键的机械特性有关，一般为 3～5ms。手动按下按键后立即释放，这个动作中稳定闭合的时间超过 20ms。因此，单片机在检测按键是否被按下时都要加上消抖操作，可采用专用的消抖电路，也可采用专用的消抖芯片。消抖操作一般可分为硬件消抖和软件消抖。

（1）硬件消抖。

在按键较少时可采用硬件消抖，如图 3-3（b）所示。在图 3-3（b）中，两个与非门构成一个 RS 触发器，当按键未被按下时，RS 触发器的输出为 1；当按键被按下时，RS 触发器的输出为 0。除采用 RS 触发器消抖电路外，有时还可采用 RC 消抖电路。工程设计中为了节省成本，一般不采用硬件消抖。



（a）按键的触点电压变化　　（b）硬件消抖

图 3-3　按键的触点电压变化和硬件消抖

（2）软件消抖。

当按键较多时，常用软件消抖，即检测到有按键被按下时执行一段延时程序，具体延时时间依机械性能而定，常用的延时时间是 3～5ms。按键抖动的这段时间内不进行检测，等到按键稳定后再读取 I/O 口电平；若按键稳定后检测到的电平仍然为闭合状态下的电平，则认为有按键被按下。

通常我们用软件延时的方法就能很容易地解决抖动问题，因此没有必要再添加多余的硬件消抖电路。

任务实施

使用按键控制 LED 的亮灭，当按键被按下时 LED 点亮，再次被按下时 LED 熄灭，其中按键由 P04 引脚控制，LED 由 P00 引脚控制，低电平点亮。

1．原理图设计

使用按键控制 LED 的原理图如图 3-4 所示。

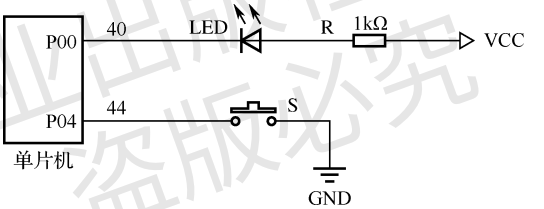


图 3-4　使用按键控制 LED 的原理图

根据图 2-7 和图 3-2，将 LED 和按键分别接到单片机的 P00 引脚和 P04 引脚上。

2．程序设计

使用按键控制 LED 的代码如下：

```
1 #include <STC15F2K60S2.H>
2 #define MAIN_Fosc 12000000L          //定义系统时钟频率为 12MHz
3 #define KEY1 P04
4 #define LED1 P00
5 void DelayMS(unsigned int ms)          //定义延时函数
6 {
      /*延时函数与任务 2-2 中的延时函数相同，省略*/
12}
13void main()
14{
15    char i=0;
```

```
16   P0M0=0x00;                          //初始化 I/O 口
17   P0M1=0x00;
18   while(1)
19   {
20     if(KEY1==0)                       //判断按键是否被按下
21         {
22               DelayMS(3);             //软件消抖
23               if(KEY1==0)             //再次判断按键是否被按下
24               {
25                     LED1=!LED1;       //LED 翻转
26                     while(KEY1==0);   //等待按键被释放
27               }
28         }
29   }
30}
```

第 3～4 行：使用宏定义定义按键和 LED 对应的 I/O 口，后续编程应尽量采用这种方法定义 I/O 口，方便程序的阅读和移植。

第 20～23 行：判断 KEY1（P04 引脚）是否为低电平，如果 KEY1 为低电平，说明按键被按下，延时 3ms 进行消抖；再次判断 KEY1（P04 引脚）是否为低电平，如果 KEY1 仍为低电平，则确认按键被按下。

第 25 行：执行按键被按下后的程序，这里对 LED1（P00 引脚）进行取反操作，即 LED 灯的亮灭状态发生转换。

第 26 行：等待按键被释放，完成一次完整的按键检测。

此程序虽然可以完成简单的按键检测任务，但是如果程序任务繁重，并且实时性要求比较高，那么在程序中使用 DelayMS(3)进行消抖和使用 while(KEY1==0)判断按键释放就会影响程序的效率，后续学完单片机定时器后我们就可以解决这个问题。

课后拓展

请说出按键检测的流程，并写出按键检测程序的模板框架。如果有多个按键，该如何处理？

任务 3-2　矩阵按键的检测

工作任务

在数字时代的浪潮中，我们见证了无数革命性技术的诞生，它们不断影响着我们的工作和生活方式。在探索人机交互的无限可能性中，矩阵按键的引入对按键的拓展具有重要意义。矩阵按键通过将多个按键以矩阵形式排列，大大增加了设备上可用的按键数量，如计算器和遥控器的按键控制面板。

本任务以 IAP15L2K61S2 为主控芯片，使用 8 个 I/O 口控制 16 个按键，设计单片机控制电路并编程，实现用矩阵按键控制一个共阳极数码管的静态显示。

思路指导

通过网络查阅矩阵按键的扫描原理，结合单片机 I/O 口的输入输出特性，动态扫描实现矩阵按键的检测。

相关知识

矩阵按键的检测

1．硬件分析

从前一个任务中我们得知，独立按键在设计的过程中，一个按键对应 1 个 I/O 口。若以相同的方式连接 16 个按键，则需占用 16 个 I/O 口，这不是很好的方法。对单片机电路而言，若需要使用多个按键，则通常会将这些按键组成矩阵阵列。例如，若需要使用 16 个按键，则将其排列成 4×4 矩阵阵列，称为矩阵按键，如图 3-5 所示。当按键 S1 被按下时，P20 和 P24 两根线就导通了。

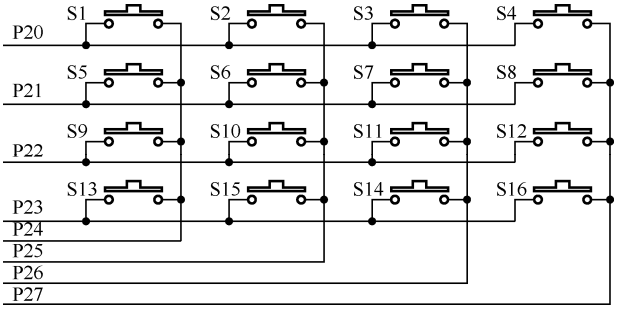


图 3-5　矩阵按键

2．软件分析

矩阵按键一般有两种检测方法：行扫描法和高低电平翻转法。假设给 P2 口赋值 0xfe，

那么 P27～P20 为 1111 1110，如果此时将 S1 按键按下，那么 P27～P20 将会变为 1110 1110，即 0xee，因为 S1 按键导通后，P24 引脚的电平将被 P20 引脚拉低，因此 P24 引脚也会变成低电平。

（1）行扫描法。

所谓行扫描法，就是先将 4 行按键中的某一行置为低电平，将别的行全部置为高电平，之后检测列对应的端口，若都为高电平，则没有按键被按下，否则有按键被按下，也可以将 4 列按键中的某一列置为低电平，将别的列全部置为高电平，之后检测行对应的端口，若都为高电平，则没有按键被按下，否则有按键被按下。行扫描法的具体方法如下。

首先，给 P2 口赋值 0xfe（1111 1110），这样只有第一行按键（P20 引脚）为低电平，别的行全为高电平，之后读取 P2 口的值，若 P2 口的值还是 0xfe，则没有按键被按下，若 P2 口的值不是 0xfe，则有按键被按下，具体是哪个按键被按下，由此时读到的值确定。如果值为 0xee（1110 1110），则表明被按下的按键是 S1；若值为 0xde（1101 1110），则表明被按下的按键是 S2（同理 0xbe→S3、0x7e→S4）。然后，给 P2 口赋值 0xfd（1111 1101），这样第二行按键（P21 引脚）为低电平，其余行全为高电平，读取 P2 口的值，若值为 0xfd，则表明没有按键被按下；若值为 0xed，则表明被按下的按键是 S5（同理 0xdd→S6、0xbd→S7、0x7d→S8）。这样依次将 P2 口赋值为 0xfb（检测第三行）、0xf7（检测第四行），就可以检测出 S9～S16 的状态。

（2）高低电平翻转法。

首先，使 P2 口高四位为 1，低四位为 0。若有按键被按下，则高四位中会有一个 1 翻转为 0，低四位不会变，此时即可确定被按下的按键的列位置。然后使 P2 口高四位为 0，低四位为 1。若有按键被按下，则低四位中会有一个 1 翻转为 0，高四位不会变，此时即可确定被按下的按键的行位置。将两次读到的数值按位进行或运算，就可以确定是哪个按键被按下了。下面举例说明。首先，给 P2 口赋值 0xf0；接着，读取 P2 口的值，若读取到的值为 0xe0，则表明第一列中有按键被按下；然后，给 P2 口赋值 0x0f 并读取 P2 口的值，若值为 0x0e，则表明第一行中有按键被按下，最后把 0xe0 和 0x0e 按位进行或运算，结果为 0xee，表明 S1 被按下。

任务实施

使用矩阵按键控制共阳极数码管，当按键 S1 被按下后，共阳极数码管显示 0；当按键 S2 被按下后，共阳极数码管显示 1；……；依次类推。其中，矩阵按键由 P2 口控制，共阳极数码管由 P0 口控制。

1．原理图设计

用矩阵按键控制共阳极数码管的原理图如图 3-6 所示。

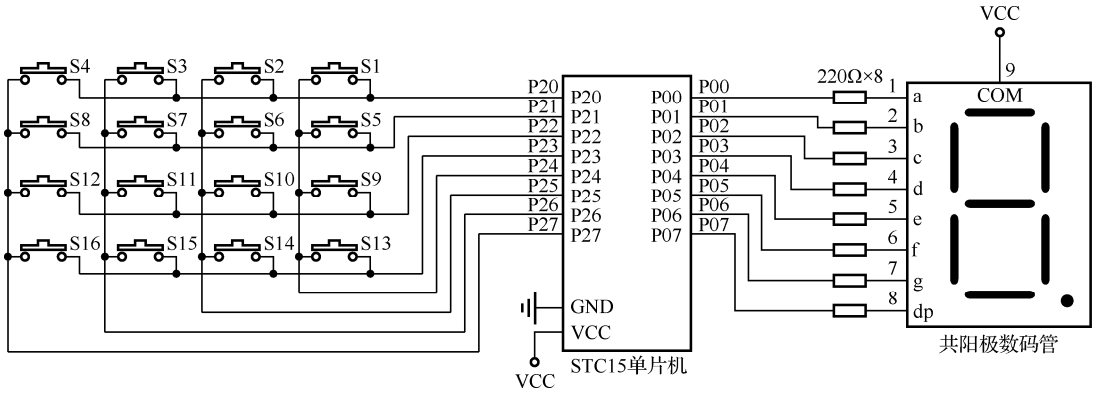


图 3-6　用矩阵按键控制共阳极数码管的原理图

采用行扫描法时，I/O 口作为输入口。如果 STC15 单片机 I/O 口作为输入口时没有上拉功能，则需要在外部添加上拉电阻，否则无法读取高电平；如果 STC15 单片机 I/O 口内部已经有上拉功能，则可以省去外部的上拉电阻。如果将 STC15 单片机的 I/O 口设置为准双向口模式，则内部有弱上拉功能，外部上拉电阻可以省略；如果将 STC15 单片机的 I/O 口设置为高阻输入模式，则内部无上拉功能，需要在 P04～P07 引脚接入 10kΩ 上拉电阻。

2．程序设计

（1）采用行扫描法实现矩阵按键控制共阳极数码管的代码如下：

```
#include <STC15F2K60S2.H>
#define MAIN_Fosc 12000000L                    //定义系统时钟频率为 12MHz
#define KEYPORT P2                             //定义矩阵按键 I/O 口
#define SMGPORT P0                             //定义共阳极数码管 I/O 口
code char SEG[] =
  {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e};
                                               //共阳极数码管的驱动信号编码
void DelayMS(unsigned int ms)                  //定义延时函数
{
   unsigned int i;
   do{
            i = MAIN_Fosc / 13000;
            while(--i);
       }while(--ms);
}
char ScanKey(void)
{
   char Temp;
    char KeyNum=16;
```

```
18  KEYPORT = 0xfe;                          //检测第一行按键
19  Temp = KEYPORT;                          //读取 P2 口的值
20  if(Temp != 0xfe)                         //若不等于 0xfe，则表明有按键被按下
21  {
22      DelayMS(3);                          //软件消抖
23      Temp = KEYPORT;                      //读取 P2 口的值
24      if(Temp != 0xfe)                     //再次判断
25      {
26          Temp = KEYPORT;                  //读取 P2 口的值
27          switch(Temp)                     //判断键值对应的按键
28          {
29              case 0xee:KeyNum = 0;break;
30              case 0xde:KeyNum = 1;break;
31              case 0xbe:KeyNum = 2;break;
32              case 0x7e:KeyNum = 3;break;
33          }
34          while(KEYPORT != 0xfe);          //按键释放检测
35      }
36  }
37  KEYPORT = 0xfd;
38  Temp = KEYPORT;
39  if(Temp != 0xfd)
40  {
41      DelayMS(3);
42      Temp = KEYPORT;
43      if(Temp != 0xfd)
44      {
45          Temp = KEYPORT;
46          switch(Temp)
47          {
48              case 0xed:KeyNum = 4;break;
49              case 0xdd:KeyNum = 5;break;
50              case 0xbd:KeyNum = 6;break;
51              case 0x7d:KeyNum = 7;break;
52          }
53          while(KEYPORT != 0xfd);
54      }
55  }
56  KEYPORT = 0xfb;
57  Temp = KEYPORT;
58  if(Temp != 0xfb)
```

```
59  {
60      DelayMS(3);
61      Temp = KEYPORT;
62      if(Temp != 0xfb)
63      {
64      Temp = KEYPORT; switch(Temp)
65      {
66          case 0xeb:KeyNum = 8;break;
67          case 0xdb:KeyNum = 9;break;
68          case 0xbb:KeyNum = 10;break;
69          case 0x7b:KeyNum = 11;break;
70      }
71      while(KEYPORT != 0xfb);
72      }
73  }
74  KEYPORT = 0xf7;
75  Temp = KEYPORT;
76  if(Temp != 0xf7)
77  {
78      DelayMS(3);
79      Temp = KEYPORT;
80      if(Temp != 0xf7)
81      {
82          Temp = KEYPORT;
83          switch(Temp)
84          {
85              case 0xe7:KeyNum = 12;break;
86              case 0xd7:KeyNum = 13;break;
87              case 0xb7:KeyNum = 14;break;
88              case 0x77:KeyNum = 15;break;
89          }
90          while(KEYPORT != 0xf7);
91      }
92  }
93  return KeyNum;
94 }
95 void main()
96 {
97  char i=0;
98  char KeyVal;
99  P0M0=0x00;                          //初始化共阳极数码管 I/O 口
```

```
100  P0M1=0x00;
101  P2M0=0x00;                              //初始化矩阵按键 I/O 口
102  P2M1=0x00;
103  while(1)
104  {
105      KeyVal = ScanKey();                 //按键扫描后返回按键键值
106       if(KeyVal!=16)                     //若按键键值为 16，则表明没有按键被按下
107       {
108            SMGPORT=SEG[KeyVal];
109       }
110  }
```

（2）采用高低电平翻转法实现矩阵按键控制共阳极数码管的部分代码如下：

```
1 char ScanKey(void)
2 {
3    char RowTemp,ColumnTemp,RowColTemp,KeyNum;
4    KEYPORT = 0xf0;                         //给高四位赋高电平
5    RowTemp = KEYPORT & 0xf0;               //读取行值，确定是哪一行
6    if((KEYPORT & 0xf0) != 0xf0)
7    //判断是否有按键被按下
8    {
9        DelayMS(3);                         //消抖
10       if((KEYPORT & 0xf0)!= 0xf0)
11       {
12           RowTemp = KEYPORT & 0xf0;       //确认有按键被按下，那么读取行值
13           KEYPORT = 0x0f;                 //给低四位赋高电平
14           ColumnTemp = KEYPORT & 0x0f;    //读取列值，确定是哪一列
15           RowColTemp = RowTemp | ColumnTemp;
16           //对行值、列值按位进行或运算，从而确定按键的位置
17           while((KEYPORT & 0x0f) != 0x0f);  //松手检测
18       }
19   }
20   switch(RowColTemp)                      //确定按键
21   {
22       case 0xee: KeyNum = 0; break;
23         case 0xde:      KeyNum = 1; break;
24         case 0xbe:      KeyNum = 2; break;
25         case 0x7e:      KeyNum = 3; break;
26         case 0xed:      KeyNum = 4; break;
27         case 0xdd:      KeyNum = 5; break;
```

```
28          case 0xbd:      KeyNum = 6; break;
29          case 0x7d:      KeyNum = 7; break;
30          case 0xeb:      KeyNum = 8; break;
31          case 0xdb:      KeyNum = 9; break;
32          case 0xbb:      KeyNum = 10; break;
33          case 0x7b:      KeyNum = 11; break;
34          case 0xe7:      KeyNum = 12; break;
35          case 0xd7:      KeyNum = 13; break;
36          case 0xb7:      KeyNum = 14; break;
37          case 0x77:      KeyNum = 15; break;
38          default:        KeyNum = 16; break;
39      }
40      return KeyNum;
41  }
```

高低电平翻转法其他部分的代码与行扫描法相同，使用此方法时需要注意：高四位、低四位的输入和输出模式正好是相反的，即高四位为输出时低四位为输入，低四位为输出时高四位为输入。因此，对某些单片机（如 STM32）来说，在使用高低电平翻转法时，需要进行 I/O 口工作模式的切换，即在上述代码的第 12 行和第 13 行之间增加 I/O 口工作模式切换语句，此时需要在中间添加一定的延时（5～10 个时钟周期），以避免读取数据错误。

无论是独立按键检测还是矩阵按键检测，目前我们编写的程序中还存在缺陷，请同学们考虑以下两个问题。

问题一：按键检测程序中我们都使用了 DelayMS(3)函数进行消抖，3ms 时间是长还是短呢？单片机执行一条指令的时间是 μs 级别的，因此 3ms 时间内单片机能做非常多的事情，而 DelayMS(3)函数却没有做任何事情，因此采用这种方法时，程序的效率就变得非常低，如果程序中还有其他任务，就会出现“卡顿”现象。

问题二：按键检测程序中我们都使用了 while 语句判断按键的释放动作，因此当按键被按下后，在不释放的情况下，程序就会出现“死机”现象。

针对以上问题，学完定时器以后才能解决，这里需要同学们注意的是，当我们写完一个程序时，应多思考程序执行效率是否高、如何才能使程序更加健壮，这样才能成为优秀的嵌入式工程师。

课后拓展

1．请说出行扫描法和高低电平翻转法编程的原理。

2．如果按键连接的 I/O 口和共阳极数码管连接的 I/O 口顺序是乱的（没有按顺序接在 P2 口和 P0 口上，而是随意连接 I/O 口），程序该如何处理？

任务 3-3　综合实训

工作任务

抢答器作为一种促进互动、提高参与度的工具，已经被广泛应用于教育、培训及团队建设等各种场合。它不仅增强了传统抢答游戏的趣味性和公平性，还通过数字化的方式实现了实时反馈和精确计时，提高了互动环节的效率和科技感。

本任务以 IAP15L2K61S2 为主控芯片，完成四路数字显示抢答器的设计。通过综合训练，掌握单片机 I/O 口作为输入口的使用方法，熟练掌握用 C 语言编写单片机程序的方法，学习按键识别电路及编程方法。

思路指导

复习数码管的静态显示及独立按键的检测，结合四路数字显示抢答器的要求，完成其功能设计。

任务实施

应用 IAP15L2K61S2 及简单的外围电路设计制作一个四路数字显示抢答器。要求：当按下“开始”按键后，选手进行抢答，使用 1 个数码管显示最先按下按键选手（抢答者）的号码，并保持到下一次抢答开始。

1．原理图设计

在常见的娱乐及知识问答节目中，抢答是一种娱乐性、竞争性较强的形式，也是比较吸引人的环节。本任务将 IAP15L2K61S2 的 P0 口用作输出口，控制数码管显示抢答者的号码；将 IAP15L2K61S2 的 P2 口用作输入口，使用 P20～P23 引脚连接 4 个独立按键。当有选手按下按键时，系统将其他选手的抢答信号屏蔽，抢答者号码的识别和显示通过程序实现。

四路数字显示抢答器的原理图如图 3-7 所示。

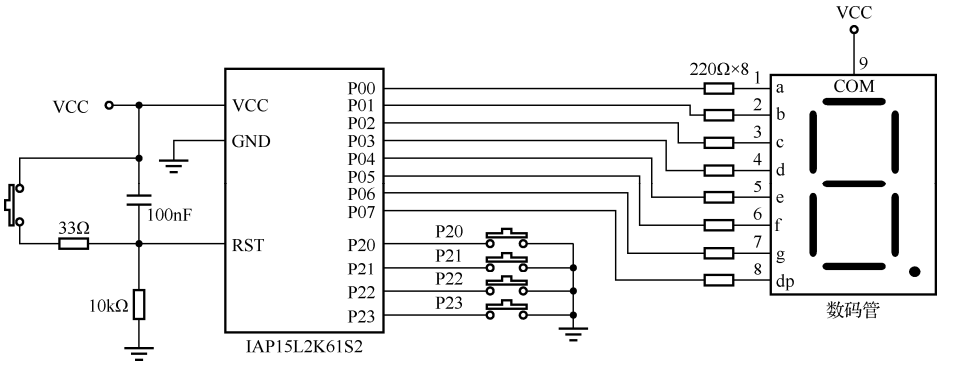
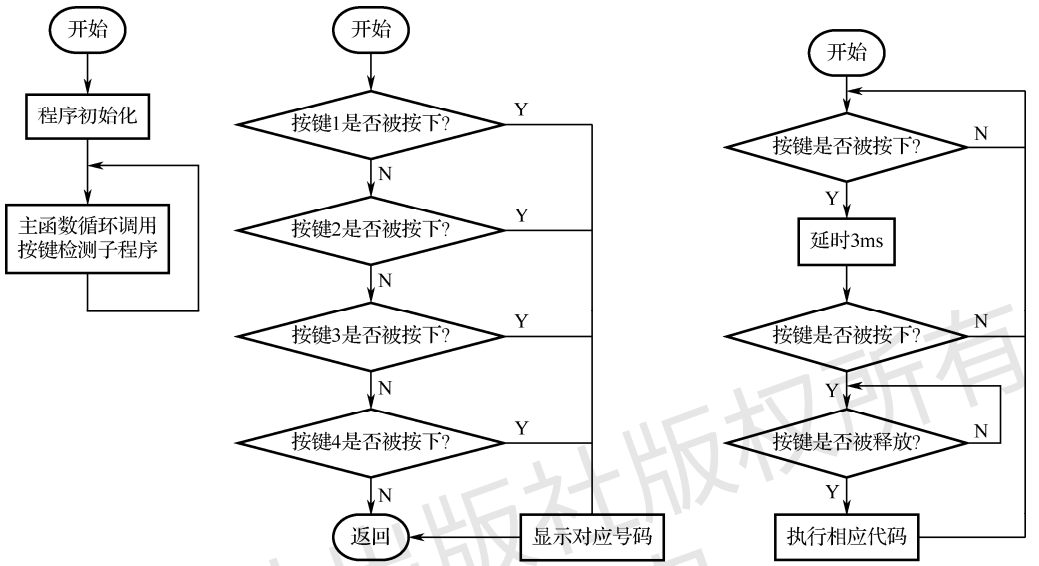


图 3-7　四路数字显示抢答器的原理图

2．程序设计

数码管上的数字要根据按键的识别情况进行显示，因此程序应使用分支结构。四路数字显示抢答器的程序流程图如图 3-8 所示。



（a）主函数　　（b）按键检测子程序　　（c）按键检测

图 3-8　四路数字显示抢答器的程序流程图

四路数字显示抢答器的代码如下：

```
1 #include <STC15F2K60S2.H>
2 #define MAIN_Fosc 12000000L                    //定义系统时钟频率为 12MHz
3 #define KEY1 P20                               //定义按键 I/O 口
4 #define KEY2 P21
5 #define KEY3 P22
6 #define KEY4 P23
7 #define SEG_PORT P0
8 code char SEG[]={0xff,0xf9,0xa4,0xb0,0x99};       //定义数码管编码表：全灭、1、2、3、4
9 char DisNum=0;
10
11void DelayMS(unsigned int ms)                  //定义延时函数
12{
      /*延时函数与任务 2-2 中相同，省略*/
18}
19
20void KeyScan()                                 //按键检测子程序
21{
22       if(DisNum!=0) return;                  //如果已经有选手抢答，则不再检测按键
```

```
    if(KEY1==0)                          //判断按键 1 是否被按下
        {
            DelayMS(3);                  //按键消抖
            if(KEY1==0)                  //再次判断按键 1 是否被按下
            {
                while(KEY1==0);          //等待按键被释放
                DisNum=1;
                return;
            }
        }
    if(KEY2==0)
        {
            DelayMS(3);
            if(KEY2==0)
            {
                while(KEY2==0);
                DisNum=2;
                return;
            }
        }
    if(KEY3==0)
        {
            DelayMS(3);
            if(KEY3==0)
            {
                while(KEY3==0);
                DisNum=3;
                return;
            }
        }
    if(KEY4==0)
        {
            DelayMS(3);
            if(KEY4==0)
            {
                while(KEY4==0);
                DisNum=4;
                return;
            }
        }
}
```

```
64void main()
65{
66    char i=0;
67    P0M0=0x00;                          //初始化数码管 I/O 口
68    P0M1=0x00;
69    P2M0=0x00;                          //初始化按键 I/O 口
70    P2M1=0x00;
71    while(1)
72    {
73        KeyScan();                      //按键检测子程序
74        SEG_PORT=SEG[DisNum];           //显示抢答者的号码
75    }
76}
```

第 8 行：使用查表法定义数码管的 5 个显示状态，分别为“全灭”“1”“2”“3”“4”。

第 9 行：DisNum 为抢答者的号码，初始值为 0，即数码管不显示任何值，当有选手按下按键后，DisNum 的值更改为其对应的号码。

第 22 行：根据 DisNum 判断是否已经抢答完毕，如果 DisNum 不为 0，表示已经有选手抢答，后面其他选手按下按键时则无效，抢答结束。

第 23～32 行：这几行为按键 KEY1 的检测程序，具体实现可参考任务 3-1，确认按键被按下后，设置 DisNum 为 1，其他三个按键的检测程序与其一致。

第 74 行：通过 DisNum 设置 SEG_PORT 端口，在数码管上显示 DisNum 的值，即显示抢答者的号码。

程序的执行过程：单片机上电或执行复位操作后，从主程序开始执行。执行主程序前，先进行相关初始化并定义 P2 口的 P20～P23 引脚连接 4 个按键，分别命名为 KEY1～KEY4。进入主程序，然后执行 while 循环。在 while 循环中，先执行按键扫描子程序，再执行“SEG_PORT=SEG[DisNum];”，将 DisNum 的值送至数码管进行显示。

按键检测子程序的执行过程：先检测抢答是否已经结束，如果 DisNum 不为 0，则说明抢答结束，不再检测按键。否则，检测按键 1 是否被按下，若已被按下（KEY1==0），则进入当前的 if 语句循环体；若没有被按下（KEY1 != 0），则执行下一条 if 语句，判断按键 2 是否被按下。进入当前 if 语句循环体后，先利用延时函数消除按键的抖动，再判断按键的状态。若消抖后按键依然是被按下的状态，则等待按键被释放，按键被释放后，将要显示的数字赋给 DisNum，然后退出按键检测子程序；若消抖后按键不是被按下的状态，则说明刚才的判断是误操作，直接退出当前循环。一次抢答结束后，主持人通过复位按键进行显示数据的消除，等待下一次抢答开始。

本程序使用了 RST 引脚的复位功能，因此在下载程序的时候，需要在 STC-ISP 软件中取消勾选“复位脚作 I/O 口”复选框，如图 3-9 所示。

图 3-9　取消勾选“复位脚作 I/O 口”复选框

课后拓展

采用 EDA 软件绘制四路数字显示抢答器的原理图和 PCB 图，制作 PCB 样板、焊接元器件、调试、下载程序，实现四路数字显示抢答器的设计。

单元小结

本教学情境以按键为对象，从独立按键的检测开始，讲解了按键的基本原理，分析了按键在实际操作过程中产生抖动的原因，给出了软件消抖和硬件消抖的方法。为在单片机 I/O 口有限的情况下进一步扩展按键的数量，讲解了矩阵按键的设计方法，分析了采用行扫描法和高低电平翻转法时的两种程序设计方法。最后综合前面所学知识，讲解了四路数字显示抢答器的设计，使同学们能灵活应用所学知识解决实际问题。

思考与练习

一、填空题

1．通常由按键设计的键盘分为_____和_____。

2．当 STC15 单片机的 I/O 口用于检测时，一般将 I/O 口设置为_____或_____模式。

3．按键抖动时间的长短和按键的机械特性有关，一般为_____ms，可采用_____和_____方法消抖。

4．矩阵按键和独立按键相比，主要解决的问题是_____。

5．矩阵按键的检测方法一般可分为_____和_____。

二、填空题

1．下列关于图 3-1 所示按键的说法中，正确的是（　　）。

A．1、2 脚为开关　　B．3、4 脚为开关

C．1、3 脚之间短路　　D．1、3 脚为开关

2．要设计 25 个按键，可以采用下面哪种矩阵按键？（　　）

A．4×4 矩阵　　B．4×5 矩阵　　C．5×5 矩阵　　D．4×6 矩阵

3．当将 STC15 单片机的 I/O 口设置为准双向口模式时，其内部具有（　　）功能。

A．弱上拉　　B．强上拉　　C．高阻　　D．开漏

4．用软件程序进行按键处理时，下列哪一项不是必需的？（　　）

A．进入中断　　B．延时消抖　　C．等待释放　　D．三项都必需

5．采用行扫描法识别有效按键时，如果读入的列值不全为 1，则说明（　　）。

A．有按键被按下　　B．一个按键被按下

C．多个按键被按下　　D．没有按键被按下

三、综合题

1．按键消抖的原理是什么？请编写一段延时 5ms 的按键消抖程序。

2．在本教学情境三个任务编写的按键检测程序中，如果按键一直被按下，会出现什么现象？程序如何运行？

3．利用 switch-case 语句编写四路数字显示抢答器的控制程序。