4x4 矩阵按键检测

1. 实验目的

- 掌握 4x4 矩阵按键检测电路的工作原理。
- 掌握 STC15W4K32S4 系列单片机 4x4 矩阵按键检测的程序设计。

2. 实验内容

- 编写程序实现操作不同矩阵按键，用户指示灯闪烁次数不同。
- 编写程序实现操作不同矩阵按键，串口调试助手输出不同键值信息。

3. 硬件电路设计

3.1. 矩阵按键检测介绍

在讲解输入按键检测时，总结了按键检测常见的有 GPIO 高低电平检测和 ADC 电阻分压检测。分压式接法，使用的单片机引脚须具有 ADC 功能，根据检测口测得不同的电压值来识别是哪个按键被按下。如下图。

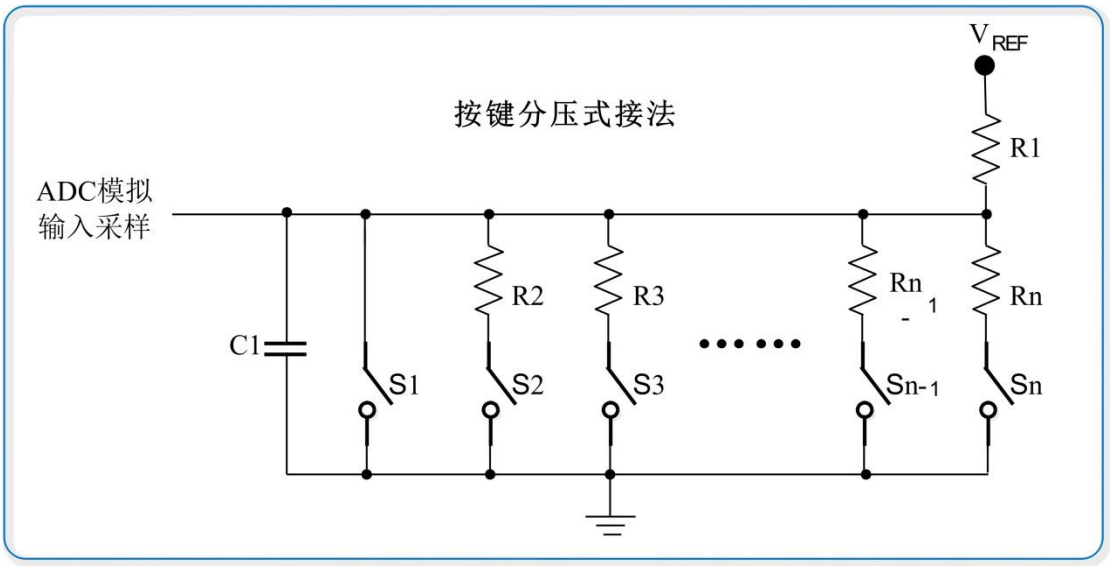

图 1：分压式接法原理

ADC 电阻分压检测按键的方法适合单片机 IO 口资源非常紧张的场合，如一些电磁炉的按键使用的就是这种方法。高低电平式接法检测按键需要用到的单片机 IO 口较多，而高低电平式接法又可分为两种：独立式接法和行列式接法。

行列式接法是利用单片机的 GPIO 口组成行与列，在行与列的每一个交点处连接按键。故也称为矩阵式按键，该接线方法最大的优势是可以使用较少的 GPIO 口实现较多按键的检测。根据行和列的不同可以有很多种矩阵按键组合，比如 2x3 矩阵按键、2x4 矩阵按键、3x3 矩阵按键、4x4 矩阵按键等等。

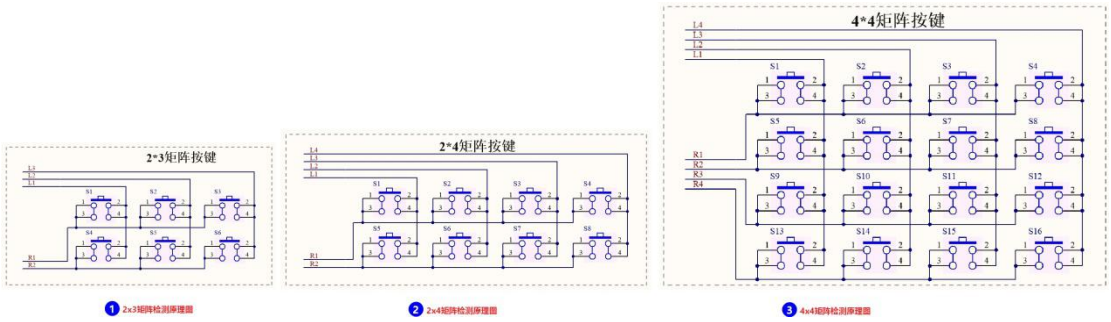

图 2：矩阵按键原理图

✧ 注：行的英文是 Row，列的英文是 Line。

上述矩阵按键原理图中没有加任何保护电阻，市场上很多矩阵按键检测模块，都是参考上图来的。这样的设计可以实现矩阵检测，但是并不是最规范的。

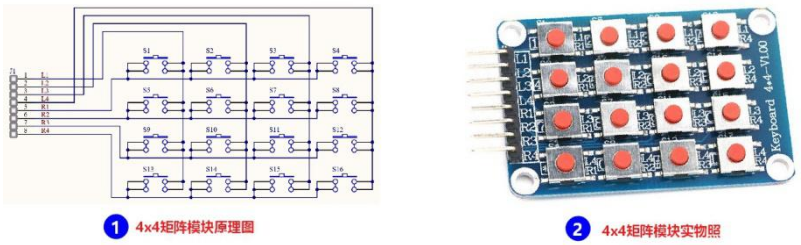

图 3：4x4 矩阵模块

进取者 STC15 开发板上设计了 4x4 矩阵按键检测电路，该检测电路在行检测和列检测连接单片机 GPIO 上串联了 220Ω 电阻，并且把行检测用 GPIO 口加了 10K 上拉电阻，如图。

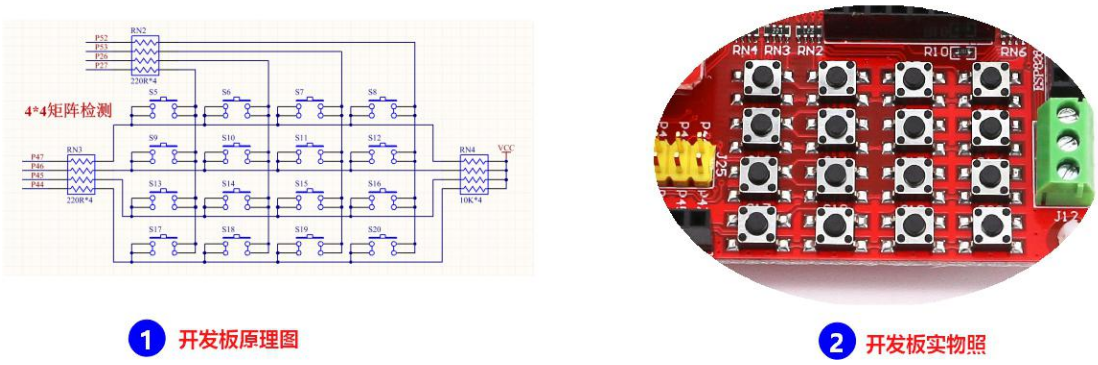

图 4：开发板 4x4 矩阵按键检测部分

✧ 上图开发板板载矩阵按键电路的 RN2 和 RN3 都是串在单片机 GPIO 口引脚的 220Ω 电阻，作用为：

1） 保护单片机 GPIO 口。若本该作为输入用的 GPIO 口不小心被配置成了输出（输出了高电平或低电平），按下按键外部会直接给 GPIO 口一个高电平或低电平。如果没有串接电阻，则若单片机 GPIO 口输出的电平和按键按下给的电平不一致，会损坏 GPIO 口。

2） 便于检测。串接 220Ω 电阻可有效降低按键按下时产生的抖动峰值电压。

✧ 上图开发板板载矩阵按键电路的 RN4 作用：保证行检测用单片机 GPIO 口在按键没有按下时为有效高电平。

✧ 4x4 矩阵按键检测电路占用的单片机的引脚如下表：

表 1：4x4 矩阵按键检测电路引脚分配

序号	功能名	功能描述	对应 IO 口	说明
1	R1	矩阵检测行 1	P4.7	RC522 模块接口 J18
2	R2	矩阵检测行 2	P4.6	RC522 模块接口 J18
3	R3	矩阵检测行 3	P4.5	RC522 模块接口 J18
4	R4	矩阵检测行 4	P4.4	RC522 模块接口 J18
5	L1	矩阵检测列 1	P2.7	RC522 模块接口 J18
6	L2	矩阵检测列 2	P2.6	独立 GPIO
7	L3	矩阵检测列 3	P5.3	4 芯通用接口 J15
8	L4	矩阵检测列 4	P5.2	4 芯通用接口 J15

✧ 注：独立 GPIO 表示开发板没有其他的电路使用这个 GPIO，非独立 GPIO 说明开发板有其他电路用到了该 GPIO。

3.2. 矩阵按键检测原理介绍

矩阵按键检测的工作原理：按键设置在行、列线交点上，行、列线分别连接到按键开关的两端。行线通过上拉电阻接到 VCC 电源上。无按键按下时，行线处于高电平的状态，而当有按键按下时，行线电平由与此行线相连的列线电平决定。

矩阵按键检测方法有多种，这里介绍下比较常用的行列扫描法。原理解析如下：

1） 将行线（R1~Rm）配置为输出口并控制输出为高电平，将列线（L1~Ln）配置为输出口并控制输出为低电平。再配置行线（R1~Rm）为输入口、列线（L1~Ln）仍是输出口，检测行线（R1~Rm）的 GPIO 口电平变化。如果有按键按下，按键按下的对应行线被拉低，否则所有的行线（R1~Rm）都为高电平。

2） 将列线（L1~Ln）配置为输出口并控制输出为高电平，将行线（R1~Rm）配置为输出口并控制输出为低电平。再配置列线（L1~Ln）为输入口、行线（R1~Rm）仍是输出口，检测列线（L1~Ln）的 GPIO 口电平变化。如果有按键按下，按键按下的对应列线被拉低，否则所有的列线（L1~Ln）都为高电平。

3） 在第一步和第二步判断有键按下后，延时 10ms 消除机械抖动，再次读取相应的行值或列值。

4） 开始扫描按键位置，采用行扫描，得到行值；采用列扫描，得到列值，再分别把行值和列值保存起来。

5） 将保存的行值和列值合并，得到按键对应的编码值，通过对编码值的操作可得到对应的按键信息，完成对矩阵按键的检测。

✧ 注：m 代表矩阵按键检测电路的行数，n 代表矩阵按键检测电路的列数。

4. 软件设计

4.1.矩阵按键扫描实验 – 指示灯闪烁

✧ 注：本节的实验源码是在“实验 2-2-2： GPIO 输入按键检测（多个 c 文件）”的基础上修改。本节对应的实验源码是：“实验 2-18-1：4x4 矩阵按键扫描实验 - 指示灯闪烁”。

4.1.1. 工程需要用到的 c 文件

本例需要用到的 c 文件如下表所示，工程需要添加下表中的 c 文件。

表 2：实验需要用到的 c 文件

序号	文件名	后缀	功能描述
1	led	. c	用户 LED 有关的用户自定义函数。
2	key_4x4	. c	4x4 矩阵按键有关的用户自定义函数。
3	delay	. c	包含用户自定义延时函数。

4.1.2. 头文件引用和路径设置

■ 需要引用的头文件

```
1.  #include " led.h"
2.  #include " delay.h"
3.  #include " key_4x4.h"
```

■ 需要包含的头文件路径

本例需要包含的头文件路径如下表：

表 3：头文件包含路径

序号	路径	描述
1	..\ Source	led.h，key_4x4.h 和 delay.h 头文件在该路径，所以要包含。
2	..\User	15W4KxxS4.h 头文件在该路径，所以要包含。

MDK 中点击魔术棒，打开工程配置窗口，按照下图所示添加头文件包含路径。

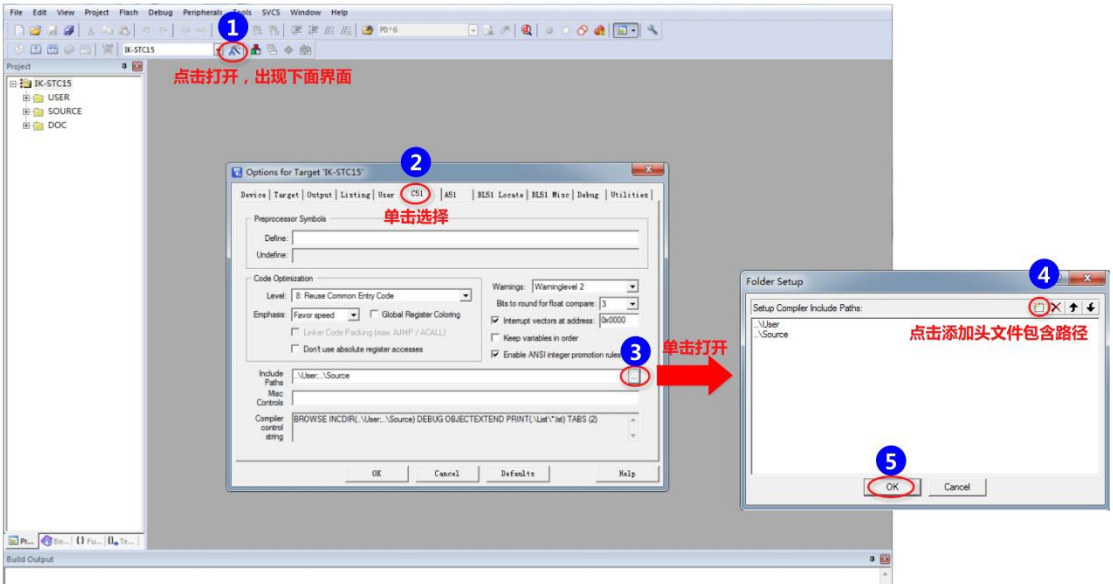

图 5：添加头文件包含路径

4.1.3. 编写代码

首先，在 key_4x4.c 文件中对 4x4 矩阵按键进行扫描，并返回键值。该 KeyScan 函数代码如下。

程序清单：4x4 矩阵扫描函数

```
/*********************************************************************************
 * 描  述 : 4x4 矩阵扫描函数
 * 入  参 : 无
 * 返回值 : 哪个按键按下的对应值
 *********************************************************************************/
uint8 KeyScan(void)
{
    uint8 X_temp,Y_temp,temp;

     X_temp=0xF0;          //列值赋初值
     Y_temp=0x0F;          //行值赋初值

     P2M1 &= 0x3F;  P2M0 |= 0xC0;      //设置 P2.6~P2.7 为强推挽输出
     P4M1 &= 0x0F;  P4M0 |= 0xF0;      //设置 P4.4~P4.7 为强推挽输出
     P5M1 &= 0xF3;  P5M0 |= 0x0C;      //设置 P5.2~P5.3 为强推挽输出


    ROW1=1;ROW2=1;ROW3=1;ROW4=1;  //行置高
      COL1=0;COL2=0;COL3=0;COL4=0;  //列置低

        //所用到行 IO 口配置为输入，进行检测
      delay_ms(10);
      P4M1 &= 0x0F; P4M0 &= 0x0F;      //设置 P4.4~P4.7 为准双向口
```

```
    delay_ms(10);

        if(ROW1 == 0)          //检测行 1 电平是否为低电平
        {
            delay_ms(10);
            if(ROW1 == 0)
            Y_temp &= 0x0E;
        }
        if(ROW2 == 0)          //检测行 2 电平是否为低电平
        {
            delay_ms(10);
            if(ROW2 == 0)
            Y_temp &= 0x0D;
        }
        if(ROW3 == 0)          //检测行 3 电平是否为低电平
        {
            delay_ms(10);
            if(ROW3 == 0)
            Y_temp &= 0x0B;
        }
        if(ROW4 == 0)          //检测行 4 电平是否为低电平
        {
            delay_ms(10);
            if(ROW4 == 0)
            Y_temp &= 0x07;
        }

      P2M1 &= 0x3F; P2M0 |= 0xC0;      //设置 P2.6~P2.7 为强推挽输出
      P4M1 &= 0x0F; P4M0 |= 0xF0;      //设置 P4.4~P4.7 为强推挽输出
      P5M1 &= 0xF3; P5M0 |= 0x0C;      //设置 P5.2~P5.3 为强推挽输出

        ROW1=0;ROW2=0;ROW3=0;ROW4=0;   //行置低
      COL1=1;COL2=1;COL3=1;COL4=1;   //列置高

        //所用到列 IO 口配置为输入，进行检测
        delay_ms(10);
      P2M1 &= 0x3F; P2M0 &= 0x3F;      //设置 P2.6~P2.7 为准双向口
      P5M1 &= 0xF3; P5M0 &= 0xF3;      //设置 P5.2~P5.3 为准双向口
    delay_ms(10);

        if(COL1 == 0)          //检测列 1 电平是否为低电平
        {
            delay_ms(10);
            if(COL1 == 0)
```

```
68.             X_temp &= 0xE0;
69.         }
70.         if(COL2 == 0)         //检测列 2 电平是否为低电平
71.         {
72.             delay_ms(10);
73.             if(COL2 == 0)
74.             X_temp &= 0xD0;
75.         }
76.         if(COL3 == 0)         //检测列 3 电平是否为低电平
77.         {
78.             delay_ms(10);
79.             if(COL3 == 0)
80.             X_temp &= 0xB0;
81.         }
82.         if(COL4 == 0)         //检测列 4 电平是否为低电平
83.         {
84.             delay_ms(10);
85.             if(COL4 == 0)
86.             X_temp &= 0x70;
87.         }
88.
89.         //将行值和列值合并，得到按键对应的编码值，该值与 16 个按键一一对应
90.         temp = X_temp|Y_temp;
91.       temp = ~temp;
92.
93.         //将按键检测的原始编码值解析对应按键值信息
94.     switch (temp)
95.     {
96.               case 0x11:return 1;   //1
97.               case 0x21:return 2;   //2
98.               case 0x41:return 3;   //3
99.               case 0x81:return 4;   //4
100.                  case 0x12:return 5;   //5
101.                  case 0x22:return 6;   //6
102.                  case 0x42:return 7;   //7
103.                  case 0x82:return 8;   //8
104.                  case 0x14:return 9;   //9
105.                  case 0x24:return 10;  //0
106.                  case 0x44:return 11;  //a
107.                  case 0x84:return 12;  //b
108.                  case 0x18:return 13;  //c
109.                  case 0x28:return 14;  //d
110.                  case 0x48:return 15;  //e
111.                  case 0x88:return 16;  //f
```

```
112.                 default:  return 0;
113.         }
114.     }
```

然后，主函数中通过检测按键扫描的键值来控制用户指示灯闪烁不同的次数。具体代码如下。

代码清单：主函数

```
1.  int main(void)
2.  {
3.          uint8 temp;
4.  ////////////////////////////////////////////////
5.  //注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 IO 口均为
6.  //       高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
7.  //相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
8.  //         P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
9.  ////////////////////////////////////////////////
10.     P0M1 &= 0x3F;   P0M0 &= 0x3F;     //设置 P0.6~P0.7 为准双向口
11.     P2M1 &= 0x3F;   P2M0 &= 0x3F;     //设置 P2.6~P2.7 为准双向口
12.     P4M1 &= 0x0F;   P4M0 &= 0x0F;     //设置 P4.4~P4.7 为准双向口
13.     P5M1 &= 0xF3;   P5M0 &= 0xF3;     //设置 P5.2~P5.3 为准双向口
14.
15.   while(1)
16.   {
17.         temp=KeyScan();      //得到键值
18.       if(temp)
19.         {
20.             switch(temp)
21.             {
22.                 case 1: //按下按键 S5
23.                     LED_FLI_B(1);    //蓝色指示灯闪烁 1 次
24.                     break;
25.                 case 2: //按下按键 S6
26.                     LED_FLI_B(2);    //蓝色指示灯闪烁 2 次
27.                     break;
28.                 case 3: //按下按键 S7
29.                     LED_FLI_B(3);    //蓝色指示灯闪烁 3 次
30.                     break;
31.                 case 4: //按下按键 S8
32.                     LED_FLI_B(4);    //蓝色指示灯闪烁 4 次
33.                     break;
34.                 case 5: //按下按键 S9
35.                     LED_FLI_B(5);    //蓝色指示灯闪烁 5 次
36.                     break;
37.                 case 6: //按下按键 S10
```

```
38.                     LED_FLI_B(6);    //蓝色指示灯闪烁 6 次
39.                     break;
40.                 case 7: //按下按键 S11
41.                     LED_FLI_B(7);    //蓝色指示灯闪烁 7 次
42.                     break;
43.                 case 8: //按下按键 S12
44.                     LED_FLI_B(8);    //蓝色指示灯闪烁 8 次
45.                     break;
46.                 case 9: //按下按键 S13
47.                     LED_FLI_R(1);    //红色指示灯闪烁 1 次
48.                     break;
49.                 case 10:    //按下按键 S14
50.                     LED_FLI_R(2);    //红色指示灯闪烁 2 次
51.                     break;
52.                 case 11:    //按下按键 S15
53.                     LED_FLI_R(3);    //红色指示灯闪烁 3 次
54.                     break;
55.                 case 12:    //按下按键 S16
56.                     LED_FLI_R(4);    //红色指示灯闪烁 4 次
57.                     break;
58.                 case 13:    //按下按键 S17
59.                     LED_FLI_R(5);    //红色指示灯闪烁 5 次
60.                     break;
61.                 case 14:    //按下按键 S18
62.                     LED_FLI_R(6);    //红色指示灯闪烁 6 次
63.                     break;
64.                 case 15:    //按下按键 S19
65.                     LED_FLI_R(7);    //红色指示灯闪烁 7 次
66.                     break;
67.                 case 16:    //按下按键 S20
68.                     LED_FLI_R(8);    //红色指示灯闪烁 8 次
69.                     break;
70.             }
71.         }
72.     }
73. }
```

4.1.4. 硬件连接

本实验需要用到用户 LED，所以 P06 和 P07 要使用短路帽短接，实验连接图如下。

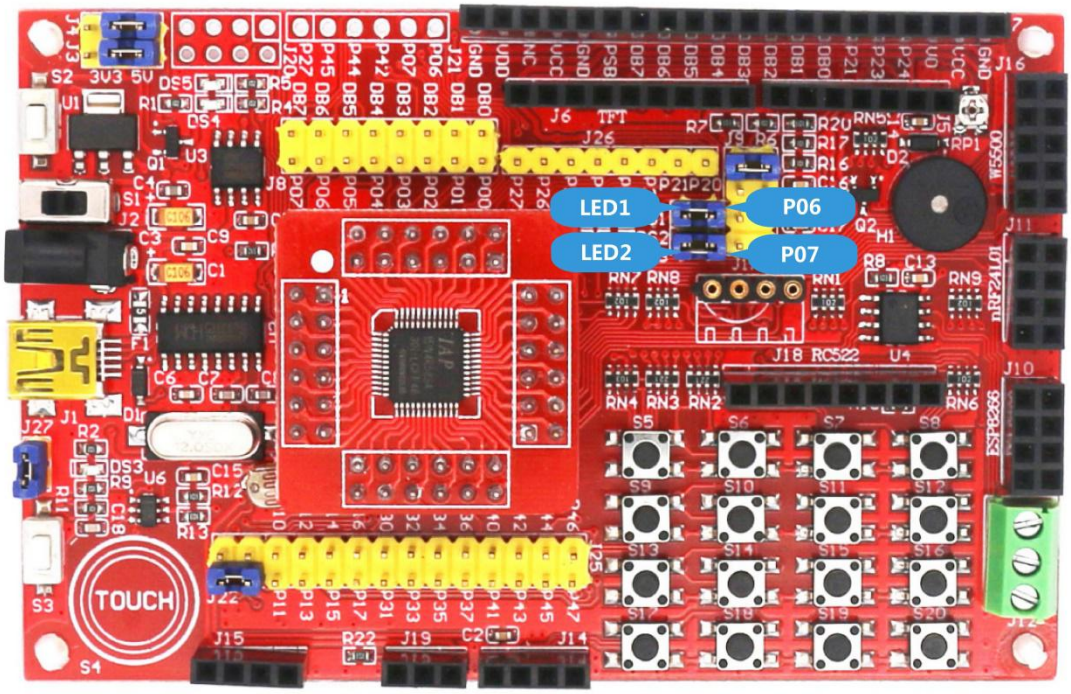

图 6：4x4 矩阵按键检测实验连接图

4.1.5. 实验步骤

1. 解压“…\第 3 部分：配套例程源码\ 1 - 基础实验程序\”目录下的压缩文件“实验 2-18-1：4x4 矩阵按键扫描实验 - 指示灯闪烁”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”。
2. 启动 Keil C51。
3. 在 Keil C51 中执行“Project→Open Project”打开“…\KeyBroad_4x4\projec”目录下的工程“KeyBroad_4x4.uvproj”。
4. 点击编译按钮编译工程。注意查看编译输出栏，观察编译的结果，如果有错误，修改程序，直到编译成功为止。编译后生成的 HEX 文件“KeyBroad_4x4.hex”位于工程目录下的“Output”文件夹中。
5. 打开 STC-ISP 软件下载程序 。下载使用内部 IRC 时钟，IRC 频率选择为 11.0592MHZ。
6. 程序运行后，按下按键 S5，开发板蓝色指示灯闪烁一次，按下按键 S6，开发板蓝色指示灯闪烁两次，以此类推，按下按键 S12，开发板蓝色指示灯闪烁八次。按下按键 S13，开发板红色指示灯闪烁一次，按下按键 S14，开发板红色指示灯闪烁两次，以此类推，按下按键 S20，开发板红色指示灯闪烁八次。

4.2.矩阵按键扫描实验 – 串口调试助手

✧ 注：本节的实验源码是在“实验 2-18-1：4x4 矩阵按键扫描实验 - 指示灯闪烁”的基础上修改。本节对应的实验源码是：“实验 2-18-2：4x4 矩阵按键扫描实验 - 串口调试助手”。

4.2.1. 工程需要用到的 c 文件

本实验需要用到的头文件以及添加头文件包含路径的方法请参考“实验 2-18-1：4x4 矩阵按键扫描实验 - 指示灯闪烁”部分。

4.2.2. 编写代码

首先，在 key_4x4.c 文件中对 4x4 矩阵按键进行扫描，并返回键值。该 KeyScan 函数代码请参考“实验 2-18-1：4x4 矩阵按键扫描实验 - 指示灯闪烁”部分。

然后，在主函数中对串口 1 进行初始化，并通过串口 1 发送按键扫描的键值。具体代码如下：

代码清单：主函数

```
int main(void)
{
 uint16 temp;
//////////////////////////////////////////////////
//注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 IO 口均为
//     高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
//相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
//       P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
//////////////////////////////////////////////////
  P2M1 &= 0x3F;  P2M0 &= 0x3F;   //设置 P2.6~P2.7 为准双向口
  P3M1 &= 0xFC;  P3M0 &= 0xFC;   //设置 P3.0~P3.1 为准双向口
  P4M1 &= 0x0F;  P4M0 &= 0x0F;   //设置 P4.4~P4.7 为准双向口
  P5M1 &= 0xF3;  P5M0 &= 0xF3;   //设置 P5.2~P5.3 为准双向口

  Uart1_Init();                  //串口 1 初始化
  EA = 1;                        //使能总中断
  delay_ms(10);                  //初始化后延时

 while(1)
 {
    temp=(uint16)KeyScan();                  //得到键值
   if(temp)
    {
      printf("\r\n 4x4 矩阵按键键值： %d\r\n",temp);  //串口打印扫描的键值信息
      temp = 0;                              //清零
    }
  }
}
```

4.2.3. 硬件连接

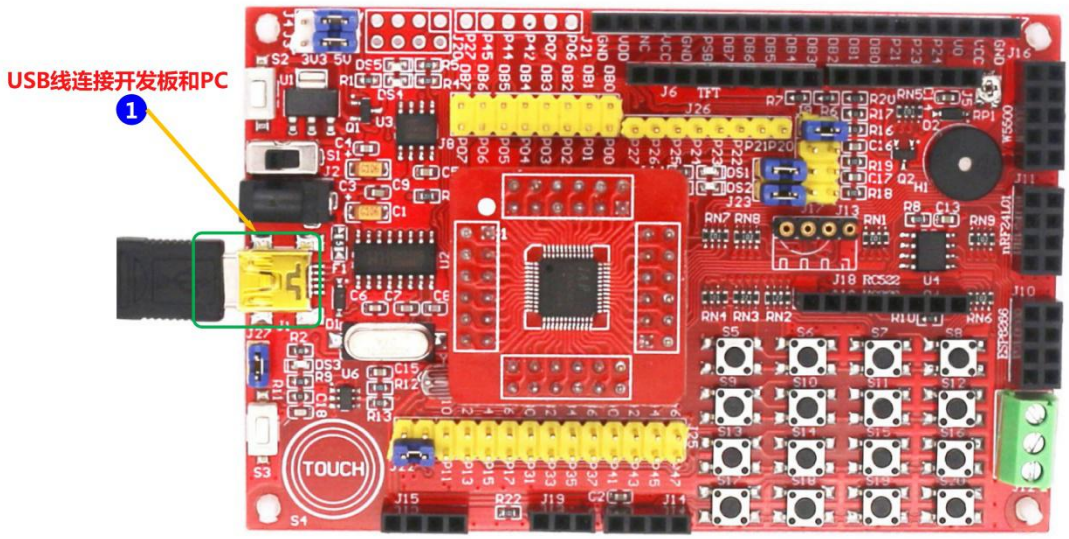

图 7：4x4 矩阵按键检测实验连接图

4.2.4. 实验步骤

1. 解压“…\第 3 部分：配套例程源码\ 1 - 基础实验程序\”目录下的压缩文件“实验 2-18-2：4x4 矩阵按键扫描实验 - 串口调试助手”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”。
2. 启动 Keil C51。
3. 在 Keil C51 中执行“Project→Open Project”打开“…\KeyBroad_4x4\project”目录下的工程“KeyBroad_4x4.uvproj”。
4. 点击编译按钮编译工程。注意查看编译输出栏，观察编译的结果，如果有错误，修改程序，直到编译成功为止。编译后生成的 HEX 文件“KeyBroad_4x4.hex”位于工程目录下的“Output”文件夹中。
5. 打开 STC-ISP 软件下载程序 。下载使用内部 IRC 时钟，IRC 频率选择为 11.0592MHZ。
6. 打开串口调试助手选择正确的串口号，波特率设置为 9600，数据位为 8、停止位为 1，选择 HEX 显示。
7. 程序运行后，依次按下按键 S5~S20，可在串口调试助手的接收窗口中实时显示“4x4 矩阵按键键值： xxxx”内容。（xxxx 是实际的数字）