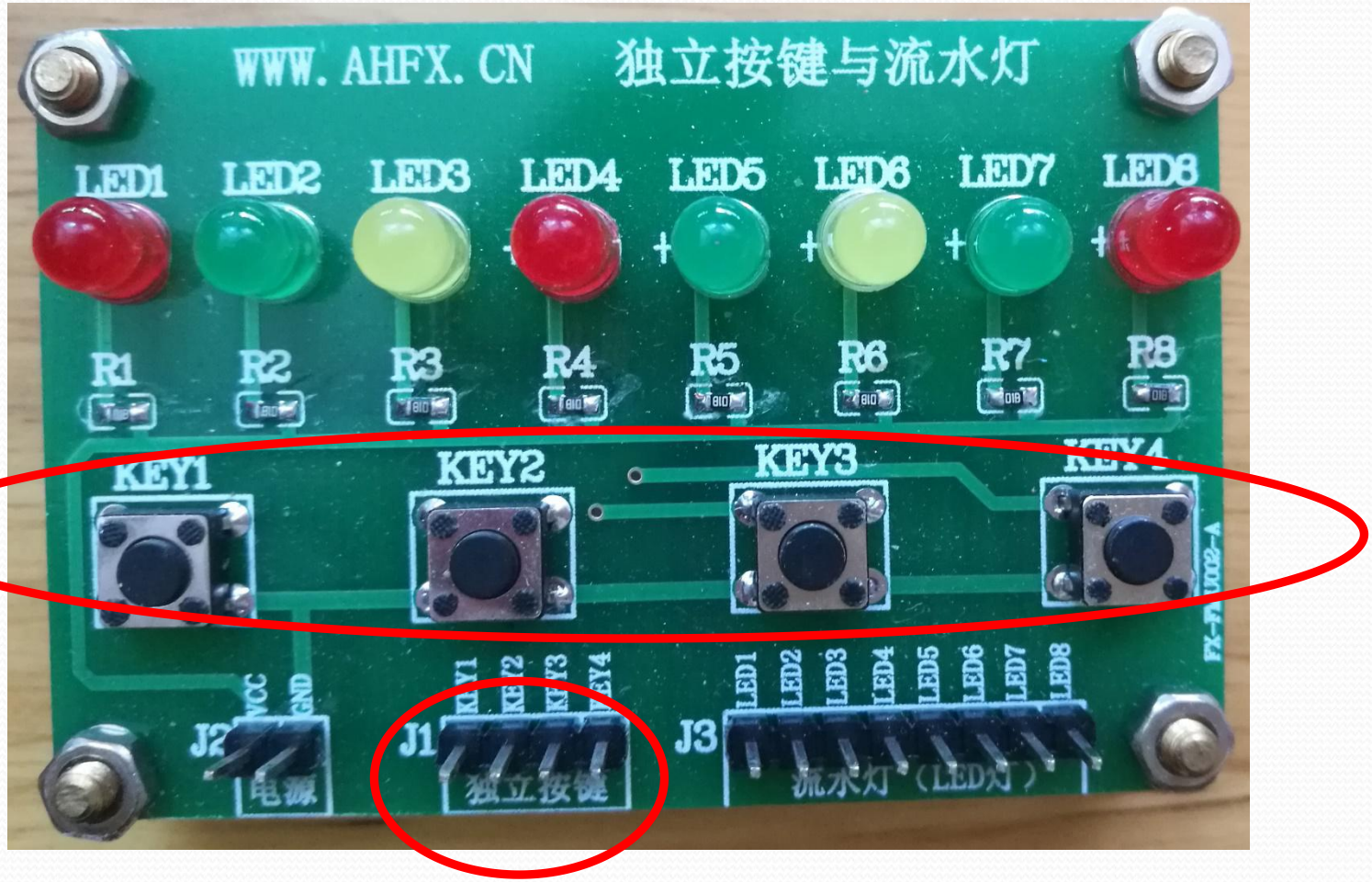
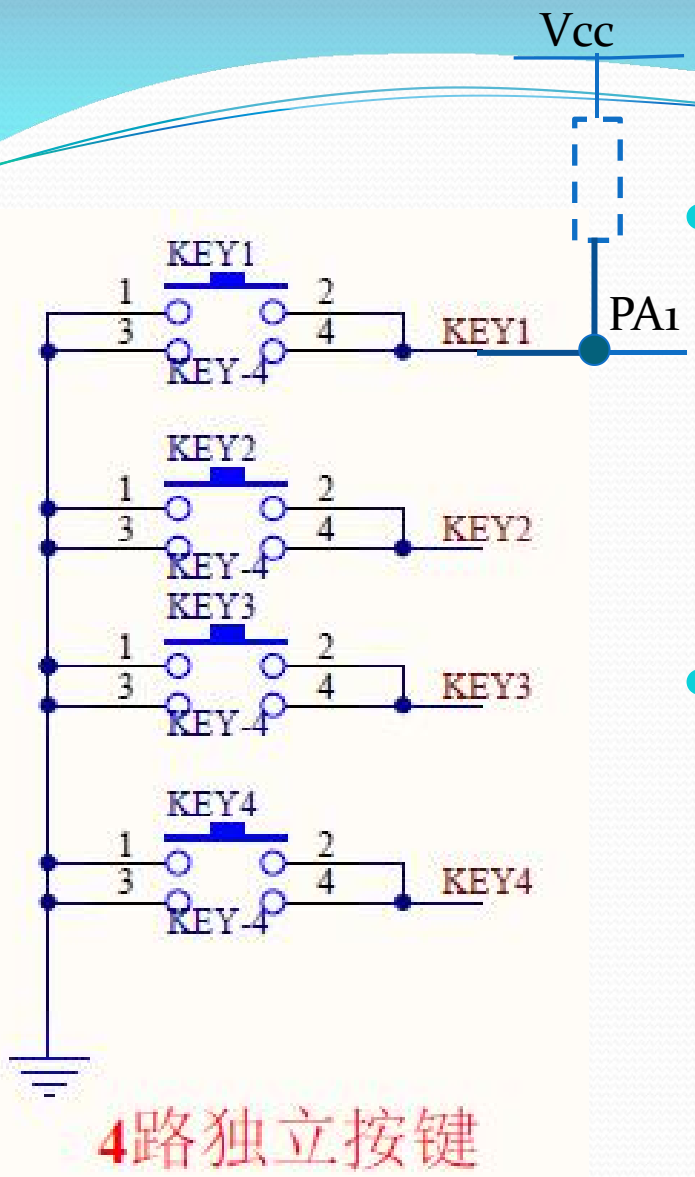
通过GPIO操纵常用输入设备

- 独立按键
- 矩阵键盘

独立按键模块

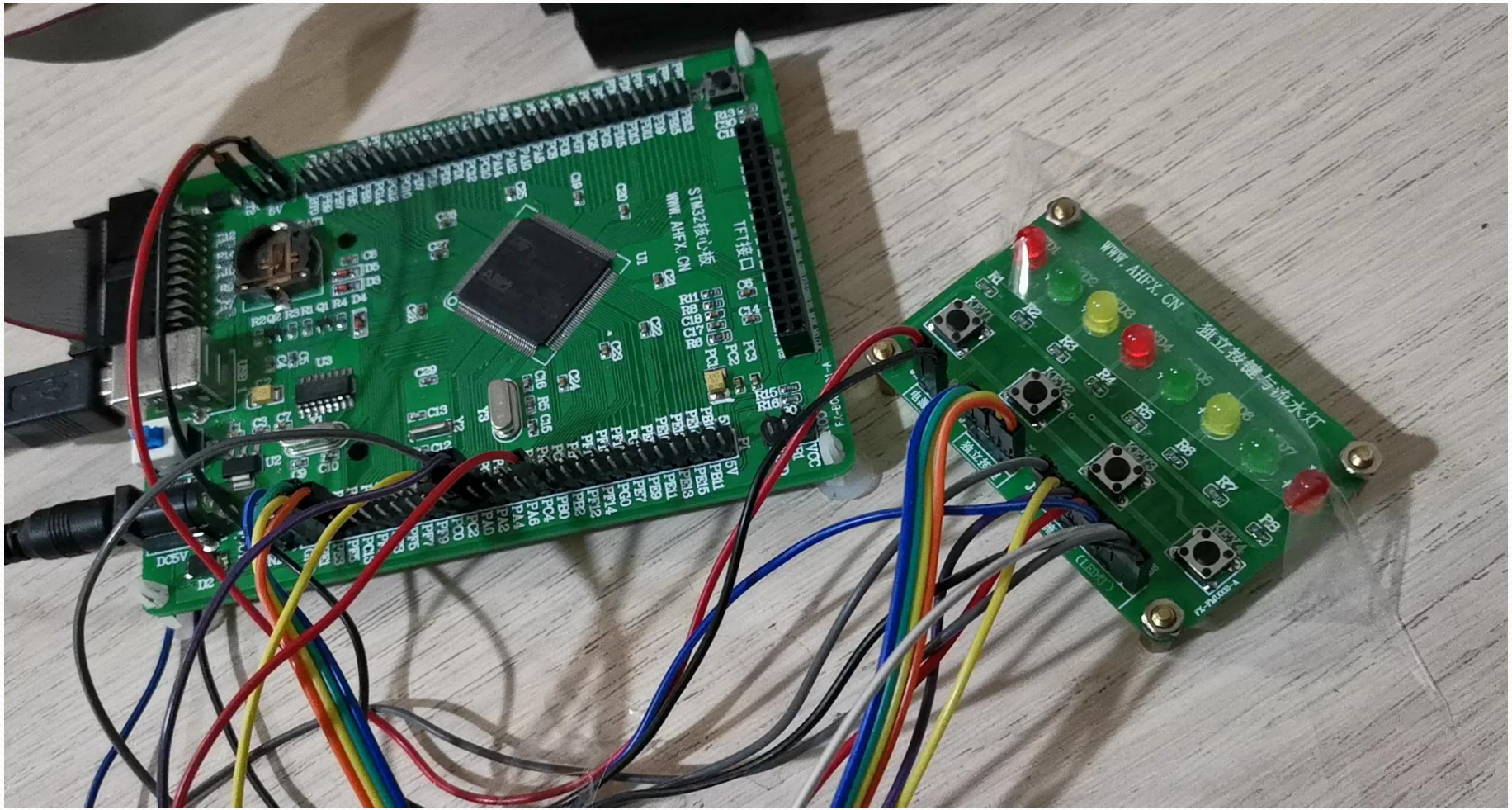

WWW.AHFX.CN
独立按键与流水灯
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED8
R1
R2
R3
R4
R5
R6
R7
R8
KEY1
KEY2
KEY3
KEY4
J2
VCC
GND
电源
J1
独立按键
J3
流水灯（LED灯）

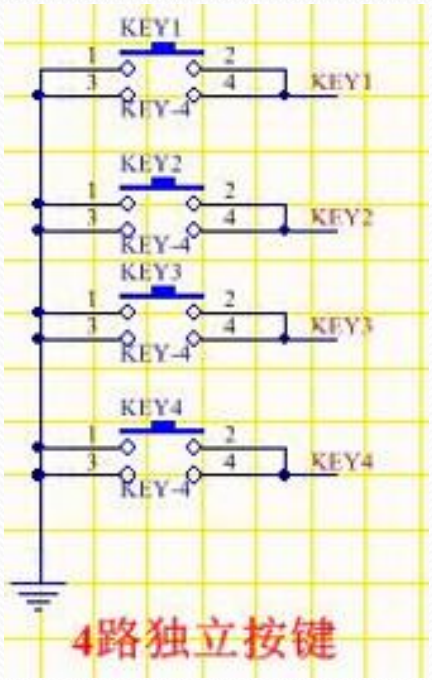


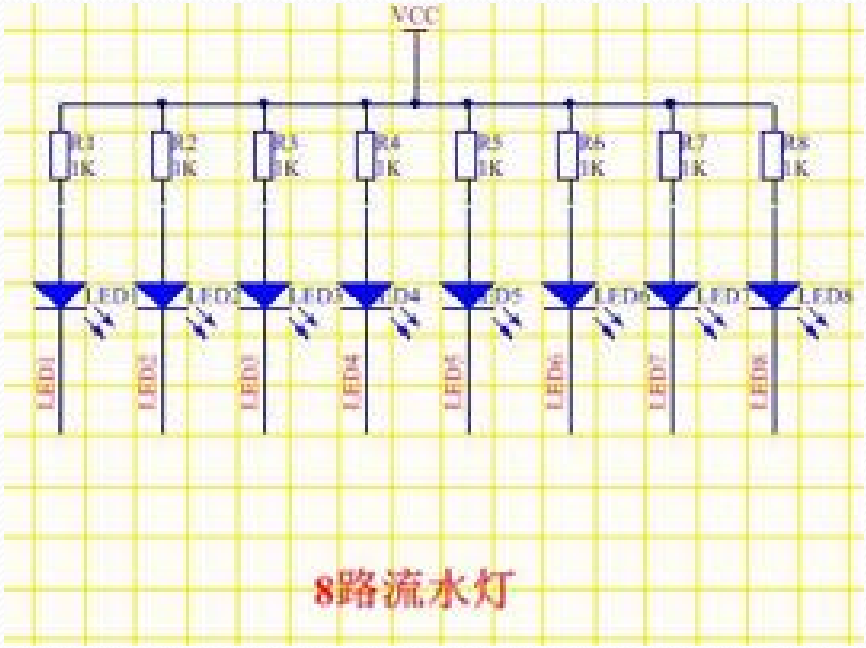
4路独立按键

- **独立按键**：每个按键都与单片机的一位 I/O 相连，每个按键的按下与 否对其他的按键所连接的单片机 I/O 不产生影响，这样的按键叫做独立按键；
- **工作原理**：按键Key1一端接地，另一端接PA1，当按键未按下时，如果PA1内部有上拉电阻，则其输入为高电平；当key1被按下后，PA1变为低电平。只要在程序中判断引脚PA1的状 态，即可知道按键key1是否处于闭合状态。

举例：独立按键扫描,据此改变LED状态







PE1--PE4对应检测Key1--Key4按键状态，
PC1--PC4对应控制Led1--LED4是否点亮。
每次按下KEYx，LEDx的状态翻转

GPIO模式常量：GPIOMode_TypeDef

引脚配置的功能	STM32枚举常量
模拟输入模式	GPIO_Mode_AIN
浮空输入模式(默认)	GPIO_Mode_IN_FLOATING
上拉输入模式	GPIO_Mode_IPU
下拉输入模式	GPIO_Mode_IPD
通用推挽输出模式	GPIO_Mode_Out_PP
通用开漏输出模式	GPIO_Mode_Out_OD
复用推挽输出模式	GPIO_Mode_AF_OD
复用开漏输出模式	GPIO_Mode_AF_PP

- 模拟输入模式一般用于ADC采集模拟电压
- 浮空输入模式时，由于其输入阻抗较大，一般用于I2C、USART等通信协议的接收端
- **对于独立按键查询而言，如果有外部上拉电阻，引脚可配置为浮空输入模式；如果没有，可以配置为上拉输入模式（此时有内部上拉效果）**

相关库函数

Table 179. GPIO 库函数

函数名	描述
GPIO_DeInit	将外设 GPIOx 寄存器重设为缺省值
GPIO_AFIODeInit	将复用功能（重映射事件控制和 EXTI 设置）重设为缺省值
GPIO_Init	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
GPIO_StructInit	把 GPIO_InitStruct 中的每一个参数按缺省值填入
GPIO_ReadInputDataBit	读取指定端口管脚的输入
GPIO_ReadInputData	读取指定的 GPIO 端口输入
GPIO_ReadOutputDataBit	读取指定端口管脚的输出
GPIO_ReadOutputData	读取指定的 GPIO 端口输出
GPIO_SetBits	设置指定的数据端口位
GPIO_ResetBits	清除指定的数据端口位
GPIO_WriteBit	设置或者清除指定的数据端口位
GPIO_Write	向指定 GPIO 数据端口写入数据
GPIO_PinLockConfig	锁定 GPIO 管脚设置寄存器
GPIO_EventOutputConfig	选择 GPIO 管脚用作事件输出
GPIO_EventOutputCmd	使能或者失能事件输出
GPIO_PinRemapConfig	改变指定管脚的映射
GPIO_EXTILineConfig	选择 GPIO 管脚用作外部中断线路

读入GPIO引脚值

- GPIO驱动程序有两个引脚输入函数
  - GPIO_ReadInputData，读取整个端口的16位
  - GPIO_ReadInputDataBit，读取某个位
- 按键检测使用位读取函数更方便

```
uint8_t GPIO_ReadInputDataBit
  ( GPIO_TypeDef *  GPIOx, uint16_t  GPIO_Pin )
```

  - GPIOx指定端口：GPIOA...GPIOG
  - GPIO_Pin指定引脚：GPIO_Pin_0...GPIO_Pin_15
  - 返回值是输入引脚的数值：0或1

帮助文档

配套程序编写--主程序

```
21  int main(void)
22  {
23    LED_GPIO_Config();  /* 配置led对应端口 */
24    Key_GPIO_Config();  /*配置按键对应端口*/
25    while(1)
26    {
27      if( Key_Scan(GPIOE,GPIO_Pin_1) == KEY_ON  )//扫描按键key1，如果有一次按键动作则让LED1翻转
28      {
29        /*LED1反转*/
30        GPIO_WriteBit(GPIOC, GPIO_Pin_1,
31          (BitAction)(1-(GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_1))));
32      }
33      if( Key_Scan(GPIOE,GPIO_Pin_2) == KEY_ON  )
34      {
35        /*LED2反转*/
36        GPIO_WriteBit(GPIOC, GPIO_Pin_2,
37          (BitAction)(1-(GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_2))));
38      }
39      if( Key_Scan(GPIOE,GPIO_Pin_3) == KEY_ON  )
40      {
41        /*LED3反转*/
42        GPIO_WriteBit(GPIOC, GPIO_Pin_3,
43          (BitAction)(1-(GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_3))));
44      }
45      if( Key_Scan(GPIOE,GPIO_Pin_4) == KEY_ON  )
46      {
47        /*LED4反转*/
48        GPIO_WriteBit(GPIOC, GPIO_Pin_4,
49          (BitAction)(1-(GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_4))));
50      }
51    }
52  }
```

配置led用到的I/O端口

```
24  void LED_GPIO_Config(void)
25  {
26    GPIO_InitTypeDef GPIO_InitStructure;
27    RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOC, ENABLE);//激活GPIOC对应时钟
28
29    //使用引脚pc1--pc4
30    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4;
31      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; //推挽输出
32      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//输出频率50Mhz
33      GPIO_Init(GPIOC, &GPIO_InitStructure);//按照上述设定初始化GPIOC
34
35    GPIO_SetBits(GPIOC, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4);  // 熄灭led1--led4（共阳）
36  }
```

配置按键用到的I/O端口

```
40  void Key_GPIO_Config(void)
41  {
42    GPIO_InitTypeDef GPIO_InitStructure;
43
44    /*开启按键端口（PE）的时钟*/
45    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);
46
47    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4;
48    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;//上拉输入模式
49
50    GPIO_Init(GPIOE, &GPIO_InitStructure);
51  }
```

按键扫描

```
50  #define KEY_ON  0
51  #define KEY_OFF 1
52
53  uint8_t Key_Scan(GPIO_TypeDef* GPIOx,u16 GPIO_Pin)
54  {
55    /*检测是否有按键按下 */
56    if(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == KEY_ON )
57    {
58      /*延时消抖*/
59      Delay(10000);
60      if(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == KEY_ON )
61      {
62        /*等待按键释放 */
63        while(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == KEY_ON);
64        return  KEY_ON;
65      }
66      else
67        return KEY_OFF;
68    }
69    else
70      return KEY_OFF;
71  }
```

LED状态翻转

```
if( Key_Scan(GPIOE,GPIO_Pin_1) == KEY_ON  )//扫描按键key1，如果有一次按键动作则让LED1翻转
{
  /*LED1反转*/
  GPIO_WriteBit(GPIOC, GPIO_Pin_1,
    (BitAction)(1-(GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_1))));
}
```

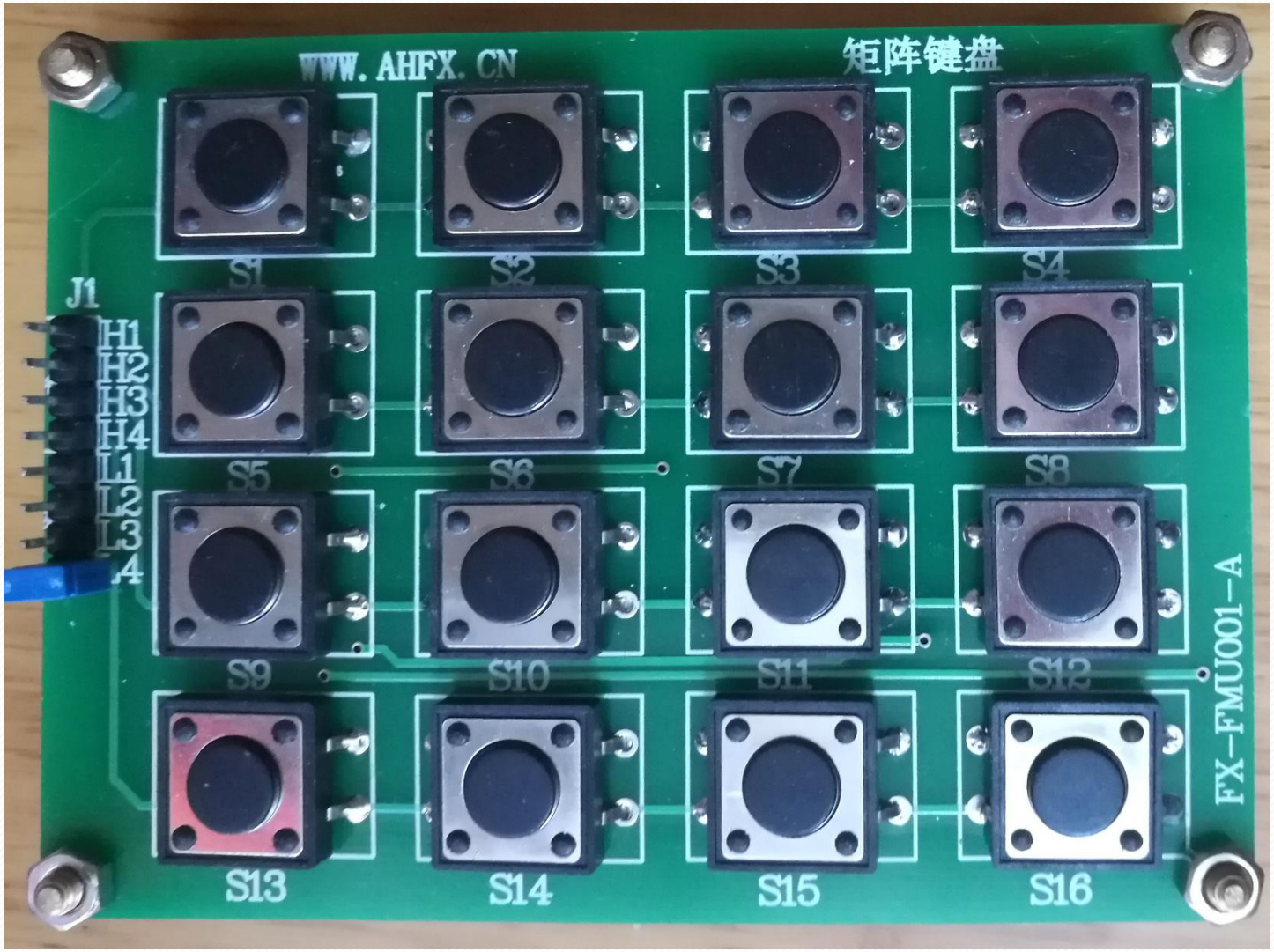
- 如果采用位带写法，诸如状态翻转的写法会相对简洁，比如：

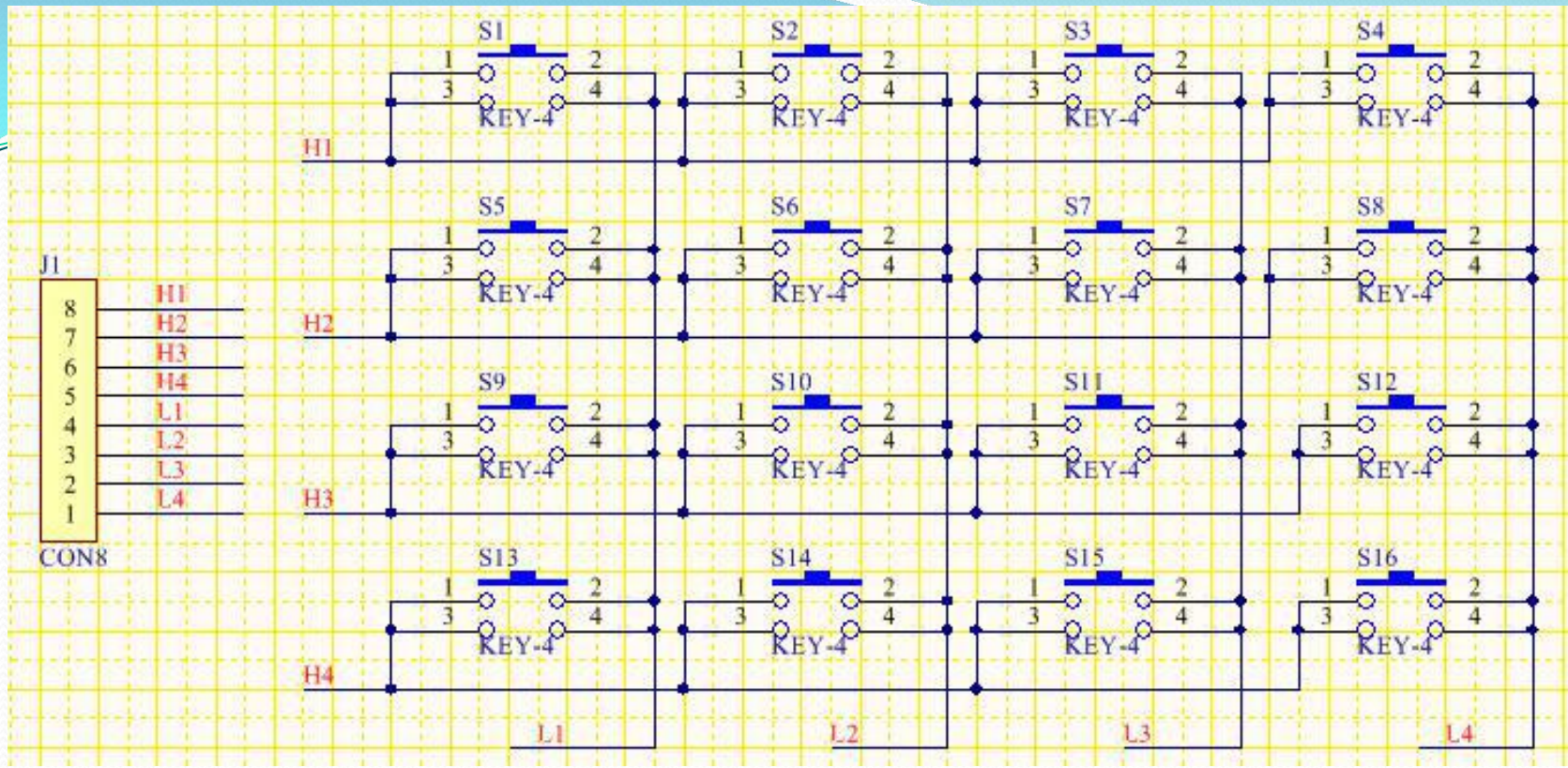
```
24  #define BITBAND(addr, bitnum) \
25  ((addr & 0xF0000000)+0x2000000+((addr &0xFFFFF)<<5)+(bitnum<<2))
26  #define MEM_ADDR(addr)  *((volatile unsigned long  *)(addr))
27  #define BIT_ADDR(addr, bitnum)   MEM_ADDR(BITBAND(addr, bitnum))
```

```
31  #define GPIOC_IDR_Addr     (GPIOC_BASE+8) //0x40011008
32  #define GPIOC_ODR_Addr     (GPIOC_BASE+12) //0x4001100C
```

```
46  #define PCout(n)   BIT_ADDR(GPIOC_ODR_Addr,n)  //输出
47  #define PCin(n)    BIT_ADDR(GPIOC_IDR_Addr,n)  //输入
48  #define OldValue PCin(3)
49  #define NewValue PCout(3)
50
51  NewValue=~OldValue;
```

矩阵键盘模块

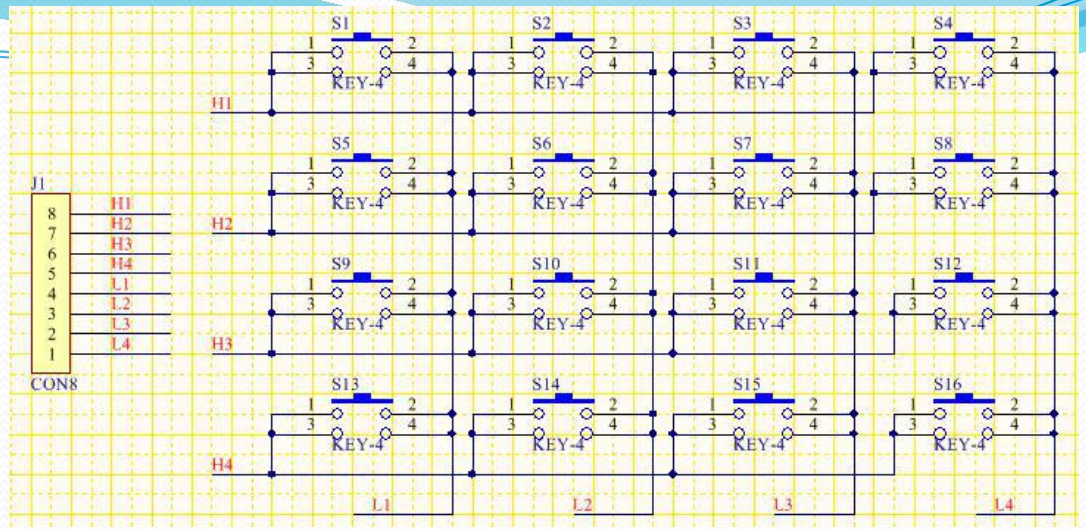

WWW.AHFX.CN
矩阵键盘
J1
H1
H2
H3
H4
L1
L2
L3
L4
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
S15
S16
FX-FMU001-A



矩阵键盘：在矩阵式键盘中，每条水平线和垂直线在交叉处不直接连通，而是通过一个按键加以连接。这样，一个8位端口(如 PD7--PD0)就可以构成 4*4=16个按键，比之直接将端口线用于独立按键多出了一倍，而且线数越多，区别越明显。因此，在需要的键数比较多时，采用矩阵键盘是比较合理的。

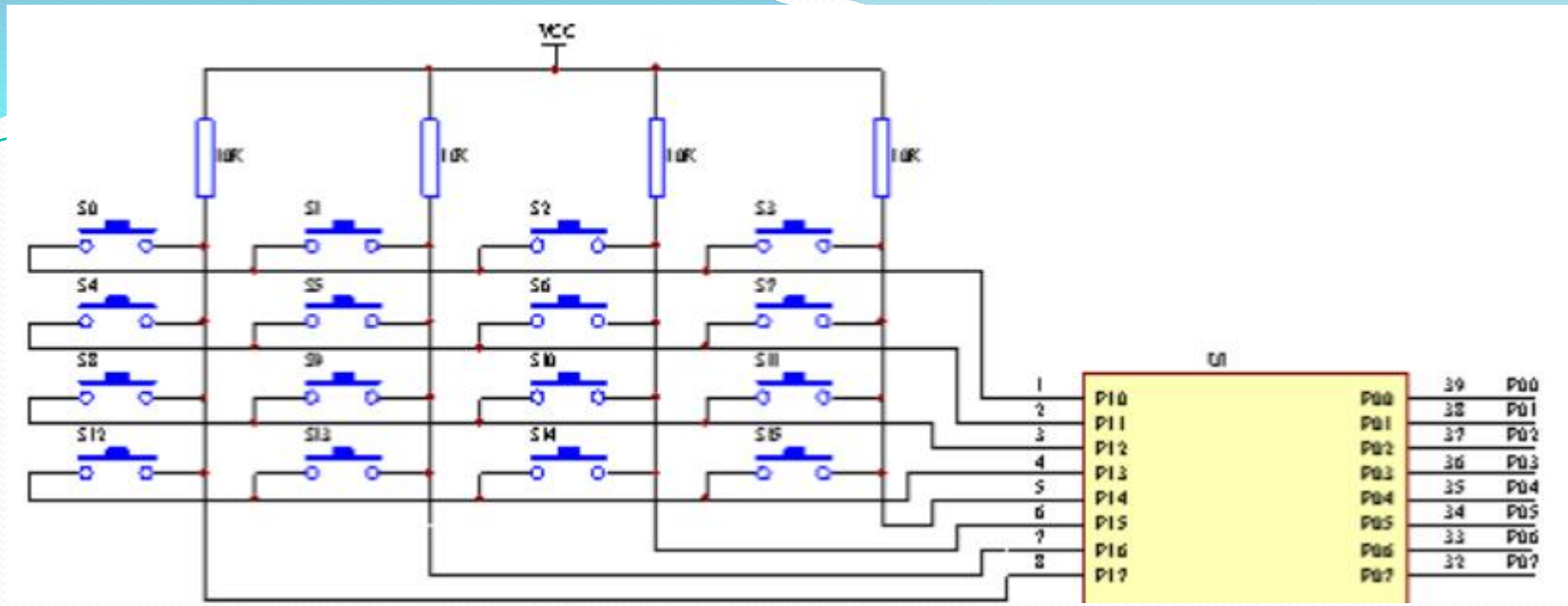
以4x4键盘为例说明按键**识别原理**：

H1--H4：行线
L1--L4：列线



第一种方法为**行扫描法**：

- **首先判断键盘中有无键按下**：将全部列线 L1-L4 置高电平，然后检测行线H1-H4 的状态（**思考：与H1--H4相连的引脚应设置为何种工作方式？**）。只要有一行的电平为高，则表示键盘中**有键被按下（闭合）**，而且闭合的键**位于高**电平行线与 4 根列线相交叉的 4 个按键之中（例如：若H1=1，则可能的按键位于第1行）。若所有列线均为低电平，则键盘中**无键被按下**；
- **接着，判断闭合键所在的位置**：在确认有键按下后，即可进入确定具体闭合键的过程。其方法是：依次将列线置为高电平，即在置某根列线为高电平时，其它列线为低电平。读出行线值，根据不同的行线值即可区分位于当前列不同行的键。（例如：在已知按键位于第1列时，令L1=1，L2=L3=L4=0，若读出H1H2H3H4=1000，则表明**第1行第1列**的按键闭合）



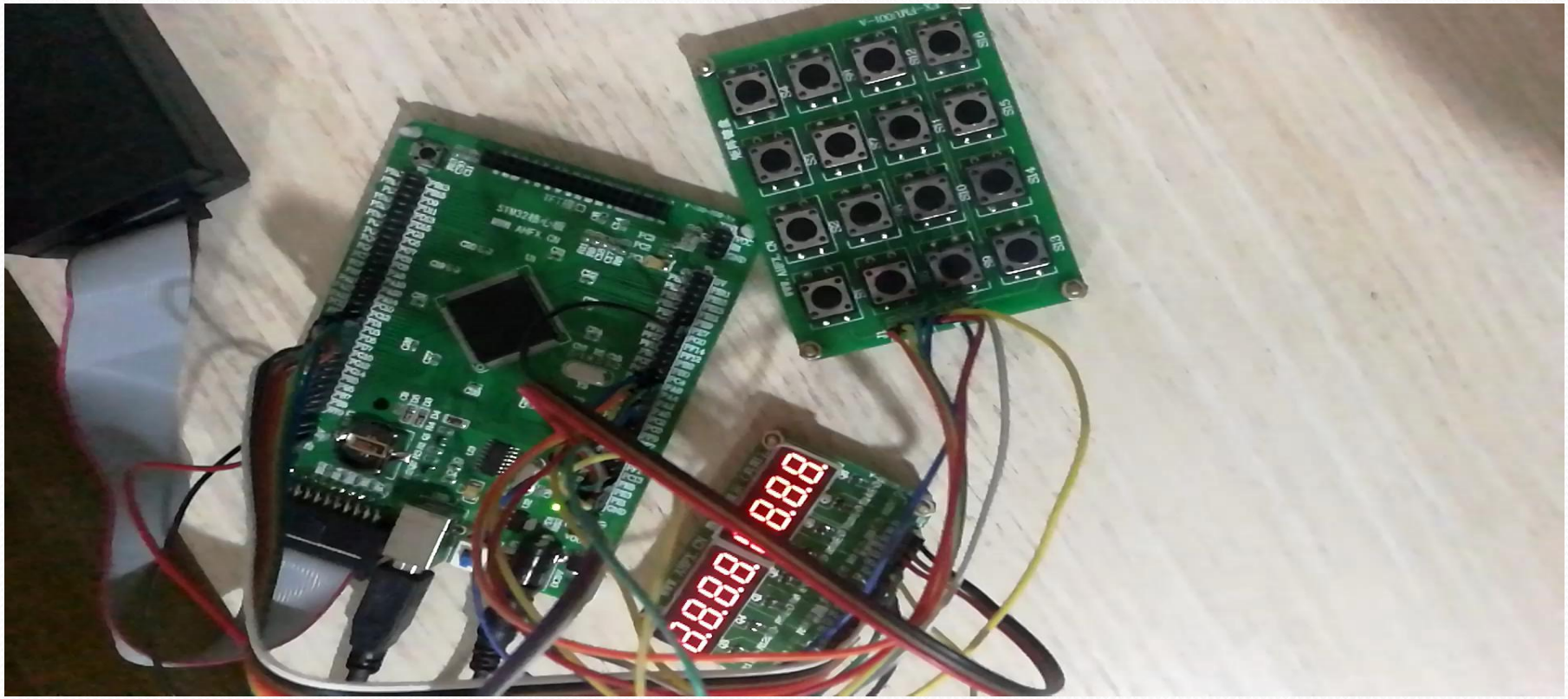
第二种方法为行列反转法：

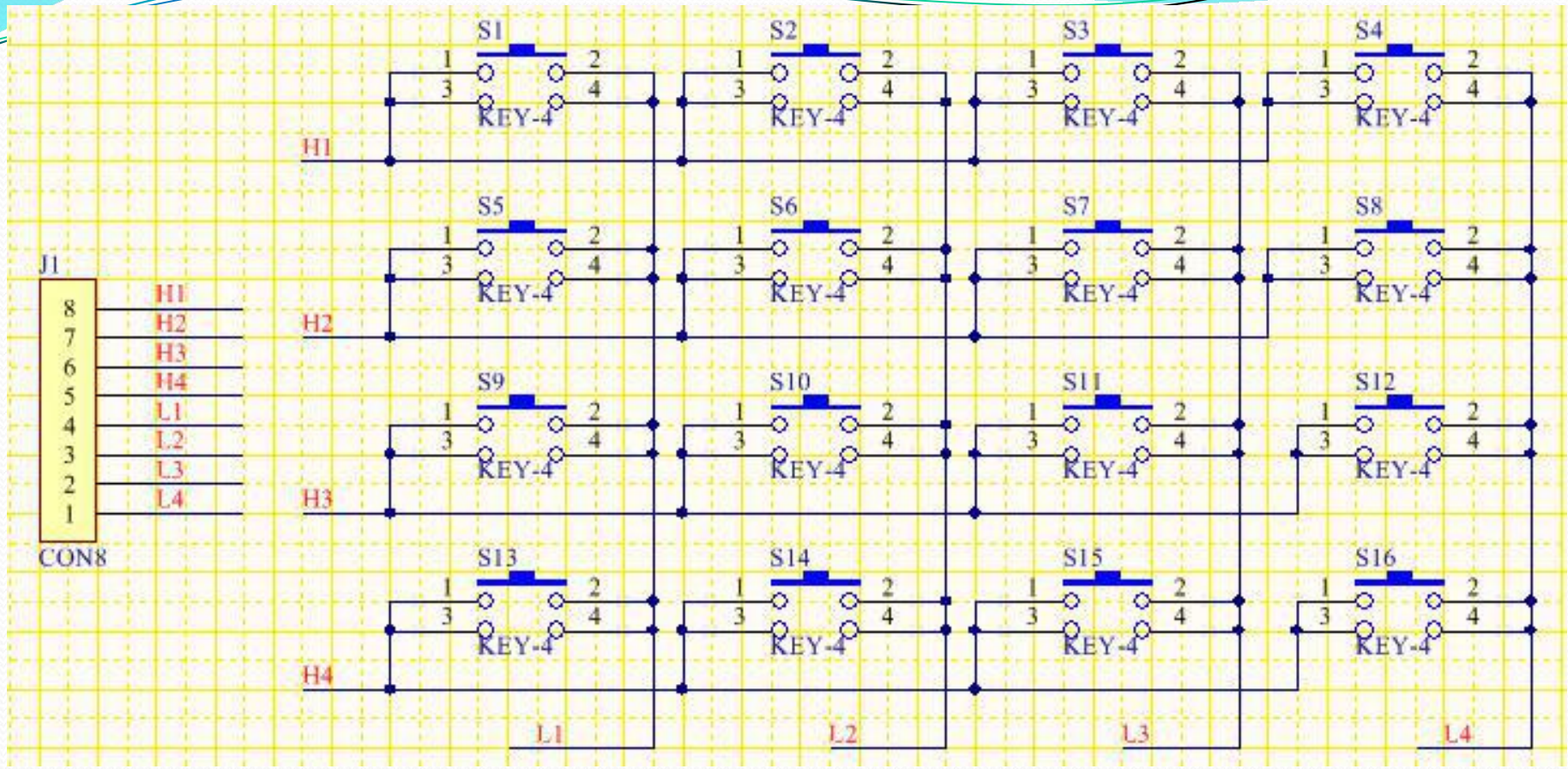
先设定列线为输出，行线为输入，所有列线置成低电平，读取所有行值（**行扫描**），如果有键按下，行值为0的列即为闭合键所在行；

反过来设定行线为输出，列线为输入，所有行线置成低电平，读取所有列值（**列扫描**），列值为o的列即为闭合键所在列；

行列交叉点处的键即为闭合键。

举例：矩阵键盘读取，并在数码管上显示键号（0-9，A-F）




J1
8
7
6
5
4
3
2
1
CON8
H1
H2
H3
H4
L1
L2
L3
L4
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
S15
S16
KEY-4

Keyboard_init按键端口初始化

```
5  #define COL_ALL GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3
6  #define ROW_ALL GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7
7
8  void KeyBoard_Init(void)
9  {
10   GPIO_InitTypeDef GPIO_InitStructure;
11   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
12
13   GPIO_InitStructure.GPIO_Pin = COL_ALL;//pa0--pa3对应从左到右四根列线
14   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
15   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//置为推挽输出模式
16   GPIO_Init(GPIOA, &GPIO_InitStructure);
17
18   GPIO_InitStructure.GPIO_Pin = ROW_ALL;//pa4--pa7对应从上到下四根行线
19   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
20   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;//置为下拉输入模式
21   GPIO_Init(GPIOA, &GPIO_InitStructure);
22
23   GPIO_SetBits(GPIOA, COL_ALL);//列线全部置1
24 }
```

Keyscan按键扫描

```
34  s16 Read_KeyValue(void)
35  {
36    s16 KeyValue=-1;//无按键
37    if((GPIO_ReadInputData(GPIOA)&0xff)!=0x0f)//列线全部置1后读入的行线值不等于全0，则表明有键按下
38    {
39      Delay_ms(2);//软件消抖
40      if((GPIO_ReadInputData(GPIOA)&0xff)!=0x0f)//延时等待后如果读入的行值仍然不全等于0，则确认有键按下
41      {
42        GPIO_SetBits(GPIOA, GPIO_Pin_0);//仅第1列列线置1,检查按键是否位于第1列
43        GPIO_ResetBits(GPIOA, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3);//第2、3、4列列线置0
44        switch(GPIO_ReadInputData(GPIOA)&0xff)//获取PA口低8位PA[7..4]PA[3..0]
45        {
46          case 0x11: KeyValue = 0; break;//PA[7..4]PA[3..0]=0001_0001表明第1行第1列之间的按键闭合
47          case 0x21: KeyValue = 4; break;//PA[7..4]PA[3..0]=0010_0001表明第2行第1列之间的按键闭合
48          case 0x41: KeyValue = 8; break;//PA[7..4]PA[3..0]=0100_0001表明第3行第1列之间的按键闭合
49          case 0x81: KeyValue = 0x0C;break;//PA[7..4]PA[3..0]=1000_0001表明第4行第1列之间的按键闭合
50        }

          。 。 。

69        GPIO_SetBits(GPIOA, GPIO_Pin_3);//仅第4列列线置1，检查按键是否位于第4列
70        GPIO_ResetBits(GPIOA, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2);
71        switch(GPIO_ReadInputData(GPIOA)&0xff)
72        {
73          case 0x18: KeyValue = 3; break;
74          case 0x28: KeyValue = 7; break;
75          case 0x48: KeyValue = 0x0B;break;
76          case 0x88: KeyValue = 0x0F;break;
77        }
78        GPIO_SetBits(GPIOA, COL_ALL);//全部列线置1
79        while((GPIO_ReadInputData(GPIOA)&0xff)!=0x0f);//等待按键释放
80        return KeyValue;
81      }
82    }
83    return -1;//没有键按下
84  }
```

NixieTube_Init数码管接口初始化

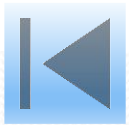
```
5   void NixieTubes_Init(void)
6   {
7
8    GPIO_InitTypeDef GPIO_InitStructure;
9    RCC_APB2PeriphClockCmd (RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE,ENABLE);
10
11   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
12   GPIO_InitStructure.GPIO_Pin = 0xff;//使用Pin0--Pin7
13   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
14   GPIO_Init(GPIOD, &GPIO_InitStructure);  //GPIOD作为位选
15   GPIO_Init(GPIOE, &GPIO_InitStructure);  //GPIOE作为段选
16  }
```

NixieDisplay数码管显示

```
18    void NixieDisplay(s16 value)
19  {
20    //DP G F E D C B A
21    //0  0 1 1 1 1 1 1
22    //以上为适合共阴接法的 0 的字形码
23    u16 num[16] = {0x3f, 0x06, 0x5b, 0x4f,
24                   0x66, 0x6d, 0x7d, 0x07,
25                   0x7f, 0x6F, 0x77, 0x7c,
26                   0x39, 0x5e, 0x79, 0x71};
27    s16 j;
28    j=value;
29
30    if(j>=0)
31    {
32  //PD[7..0]对应选择从左到右的8个数码管，低电平选用，高电平禁用
33      GPIOD->ODR = 0xff;//关闭全部显示
34      GPIOE->ODR = ~num[j];//字形表格按共阴接法设计，对于共阳接法，取反使用
35      GPIOD->ODR = 0x00;//选择全部管子
36      Delay_ms(2);
37    }
38  }
```

主程序

```
 6  int main()
 7  {
 8      s16 kval;
 9
10      SysTick_Init();         //初始化系统时钟
11      KeyBoard_Init();        //初始化矩阵键盘GPIOA
12      NixieTubes_Init();      //使用GPIOD、GPIOE控制数码管
13
14      while(1)
15      {
16          kval = Read_KeyValue();
17          NixieDisplay(kval);
18      }
19
20  }
```



实验3：GPIO外设—输入设备

- 任务1：读取4个独立按键，对应控制4个led灯，键按下则灯状态翻转
- 任务2:读取4个独立按键，控制8个流水灯产生4种不同显示花样
- 任务3（选做）：读取4x4矩阵键盘，从上到下，从左到右，依次定义为0—9，A—F，按下相应按键，则在8段数码管上显示对应数符。