通用输入输出3

程晨闻

东南大学电气工程学院

课程回顾

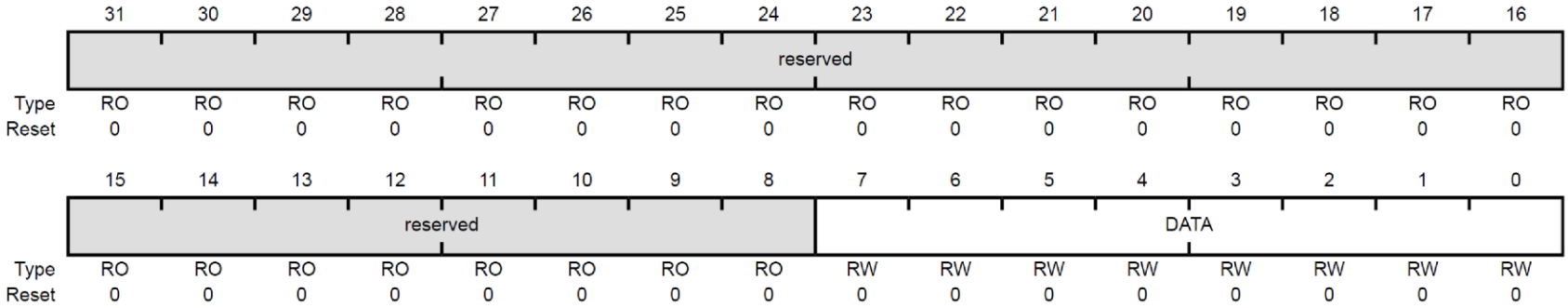
- 控制寄存器的访问
- 控制寄存器的功能
- 数据寄存器的功能与操作
- **GPIO模块的使能**

学习内容

- 控制寄存器的访问
- 控制寄存器的功能
- 数据寄存器的功能与操作
- GPIO模块的使能
- GPIO模块的应用1：矩阵键盘
- GPIO模块的应用2：数码管
- 矩阵键盘和数码管的应用

➢ 数据寄存器GPIODATA

- 低八位有意义，代表了每个引脚的电平
- 如果GPIODIR寄存器配置为输出，那么这个寄存器相应的位的电平，会直接传递到对应的引脚上。
- 如果GPIODIR寄存器配置为输入，那么这个寄存器相应的位的电平，代表对应的引脚上的电平。



Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	DATA	RW	0x00	GPIO Data This register is virtually mapped to 256 locations in the address space. To facilitate the reading and writing of data to these registers by independent drivers, the data read from and written to the registers are masked by the eight address lines [9:2]. Reads from this register return its current state. Writes to this register only affect bits that are not masked by ADDR[9:2] and are configured as outputs. See “Data Register Operation” on page 749 for examples of reads and writes.

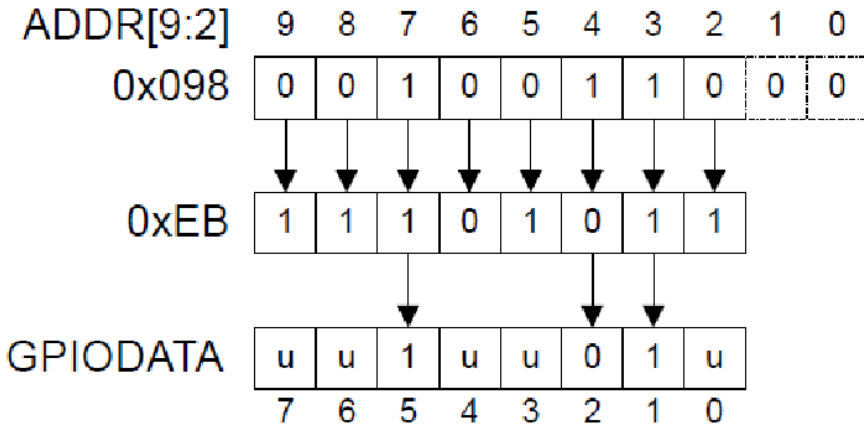
➢ GPIODATA的位带操作

– 使用总线的[9:2]位作为屏蔽位

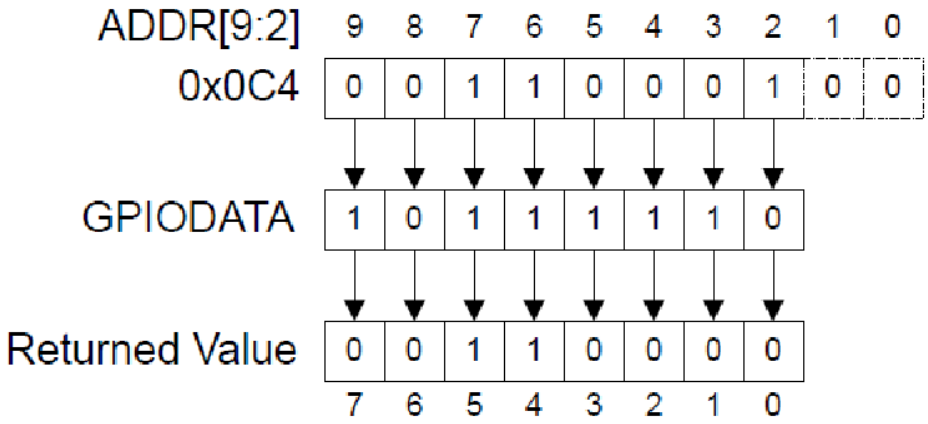
Offset	Name	Type	Reset	Description	See page
0x000	GPIODATA	RW	0x0000.0000	GPIO Data	759
0x400	GPIODIR	RW	0x0000.0000	GPIO Direction	760
0x404	GPIOIS	RW	0x0000.0000	GPIO Interrupt Sense	761
0x408	GPIOIBE	RW	0x0000.0000	GPIO Interrupt Both Edges	762
0x40C	GPIOIEV	RW	0x0000.0000	GPIO Interrupt Event	763

中间空出了一定的地址空间

写



读



– TivaWare在gpio.h中提供了GPIOPinWrite和GPIOPinRead函数，对GPIODATA寄存器进行读写

– 在gpio.c中，提供了这两个函数的具体实现方式

```
void GPIOPinWrite(uint32_t ui32Port, uint8_t ui8Pins, uint8_t
ui8Val)
{
    // Check the arguments.
    ASSERT(_GPIOBaseValid(ui32Port));
    // Write the pins.
    HWREG(ui32Port + (GPIO_O_DATA + (ui8Pins << 2))) = ui8Val;
}
int32_t
GPIOPinRead(uint32_t ui32Port, uint8_t ui8Pins)
{
    // Check the arguments.
    ASSERT(_GPIOBaseValid(ui32Port));
    // Return the pin value(s).
    return(HWREG(ui32Port + (GPIO_O_DATA + (ui8Pins << 2))));
}
```

- ui32Port为端口基地址
- ui8Pins是要操作的位
  - 是**GPIO_PIN_0-7**的组合（**或**逻辑）
- ui8Val是要操作的位对应的值
  - 用**GPIO_PIN_0-7**的组合（**或**逻辑）表示

```
#define GPIO_O_DATA               0x00000000  // GPIO Data
void
GPIOPinWrite(uint32_t ui32Port, uint8_t ui8Pins, uint8_t ui8Val)
{
    // Check the arguments.
    ASSERT(_GPIOBaseValid(ui32Port));
    // Write the pins.
    HWREG(ui32Port + (GPIO_O_DATA + (ui8Pins << 2))) = ui8Val;
}
 #define GPIO_PIN_0                 0x00000001  // GPIO pin 0
 #define GPIO_PIN_1                 0x00000002  // GPIO pin 1
 #define GPIO_PIN_2                 0x00000004  // GPIO pin 2
 #define GPIO_PIN_3                 0x00000008  // GPIO pin 3
 #define GPIO_PIN_4                 0x00000010  // GPIO pin 4
 #define GPIO_PIN_5                 0x00000020  // GPIO pin 5
 #define GPIO_PIN_6                 0x00000040  // GPIO pin 6
 #define GPIO_PIN_7                 0x00000080  // GPIO pin 7
```

将PORTn的0号引脚设置为1，将PORTn的1号引脚设置为0：

```
GPIOPinWrite(GPIO_PORTN_BASE, (GPIO_PIN_0| GPIO_PIN_1), GPIO_PIN_0);
```

将PORTn的0号引脚设置为0，将PORTn的1号引脚设置为1：

```
GPIOPinWrite(GPIO_PORTN_BASE, (GPIO_PIN_0| GPIO_PIN_1), GPIO_PIN_1);
```

将PORTn的0号引脚设置为1，将PORTn的1号引脚设置为1：

```
GPIOPinWrite(GPIO_PORTN_BASE, (GPIO_PIN_0| GPIO_PIN_1), GPIO_PIN_0 |
GPIO_PIN_1);
```

将PORTn的0号引脚设置为0，将PORTn的1号引脚设置为0：

```
GPIOPinWrite(GPIO_PORTN_BASE, (GPIO_PIN_0| GPIO_PIN_1), 0x00);
```

GPIO模块的使能

– 系统控制模块中的RCGCGPIO寄存器控制了14个GPIO的时钟，只有打开GPIO模块的时钟，GPIO模块才能工作

General-Purpose Input/Output Run Mode Clock Gating Control (RCGCGPIO)

Base 0x400F.E000
Offset 0x608
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0
Type	RO	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:15	reserved	RO	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
14	R14	RW	0	GPIO Port Q Run Mode Clock Gating Control Value Description 0 GPIO Port Q is disabled. 1 Enable and provide a clock to GPIO Port Q in Run mode.
13	R13	RW	0	GPIO Port P Run Mode Clock Gating Control Value Description 0 GPIO Port P is disabled.

- TivaWare库中，提供了**SysCtlPeripheralEnable**函数
  - 开启GPION模块时钟
  - SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
- SysCtlPeripheralEnable函数包含在driverlib.lib库中，其源代码在C:\ti\TivaWare_C_Series-2.2.0.295\driverlib\sysctl.c文件中实现

```
void
SysCtlPeripheralEnable(uint32_t ui32Peripheral)
{
    // Check the arguments.
    ASSERT(_SysCtlPeripheralValid(ui32Peripheral));
    // Enable this peripheral.
    HWREGBITW(SYSCTL_RCGCBASE + ((ui32Peripheral & 0xff00) >> 8),
              ui32Peripheral & 0xff) = 1;
}
#define SYSCTL_RCGCBASE         0x400fe600

#define SYSCTL_PERIPH_GPION     0xf000080c  // GPIO N
```

General-Purpose Input/Output Run Mode Clock Gating Control (RCGCGPIO)

Base 0x400F.E000
Offset 0x608
Type RW, reset 0x0000.0000

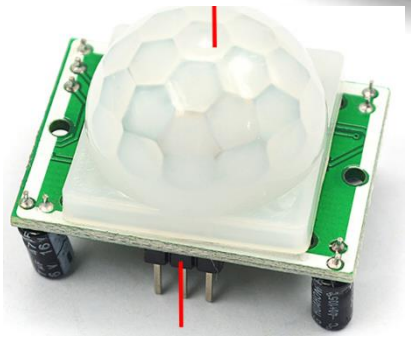
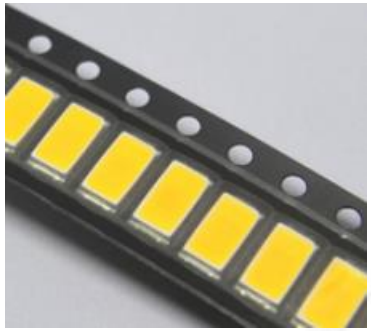
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0
Type	RO	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

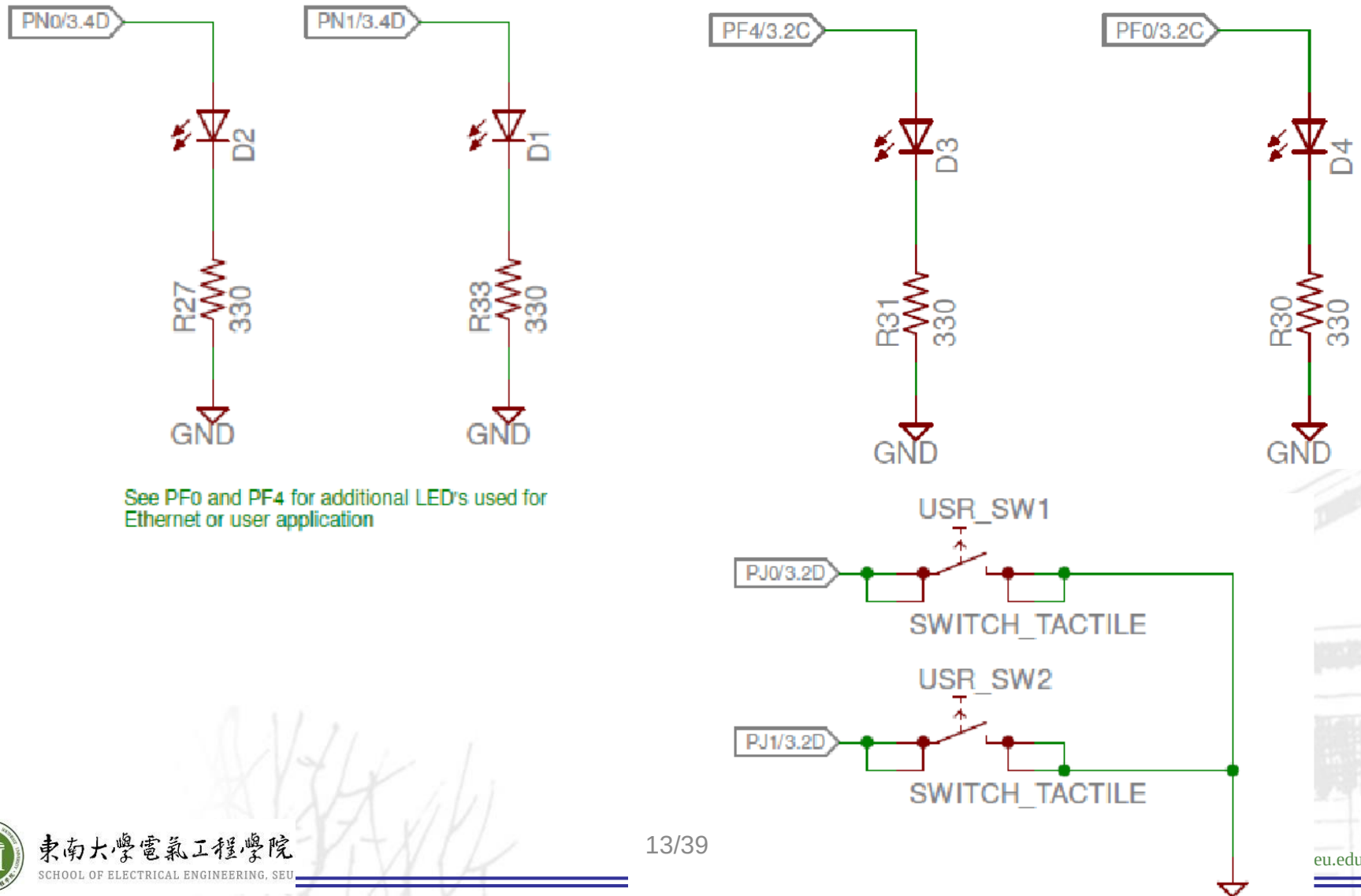
Bit/Field	Name	Type	Reset	Description
31:15	reserved	RO	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
14	R14	RW	0	GPIO Port Q Run Mode Clock Gating Control Value Description 0 GPIO Port Q is disabled. 1 Enable and provide a clock to GPIO Port Q in Run mode.
12	R12	RW	0	GPIO Port N Run Mode Clock Gating Control Value Description 0 GPIO Port N is disabled. 1 Enable and provide a clock to GPIO Port N in Run mode.

- 常用的IO驱动电路
- 简单按键开关
- 矩阵键盘
- 继电器
- LED灯
- 数码管
- 传感器输入



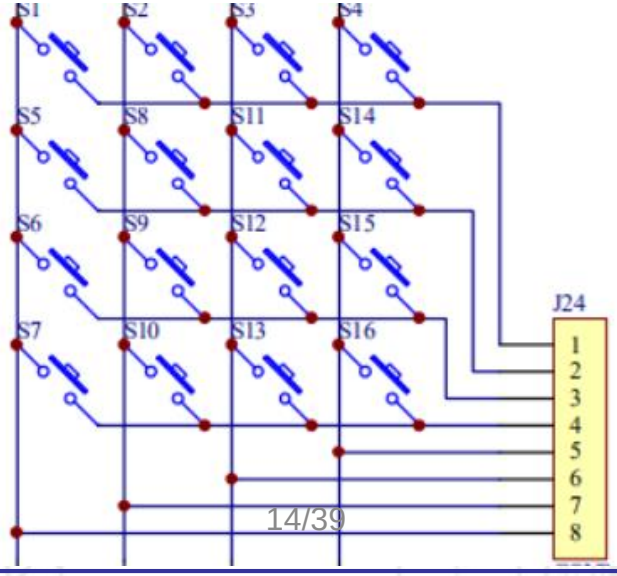
- 开发板上的按键和LED灯



- 矩阵键盘

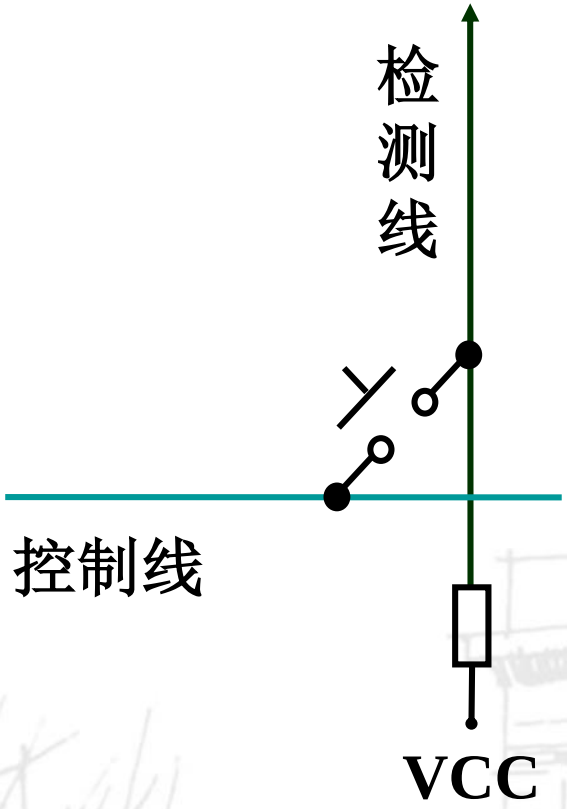

1
4
7
*
2
5
8
0
3
6
9
#
A
B
C
D


C4
C3
C2
C1
R1
R2
R3
R4
S13
S14
S15
S16


S1
S2
S3
S4
S5
S8
S11
S14
S6
S9
S12
S15
S7
S10
S13
S16
J24
1
2
3
4
5
6
7
8

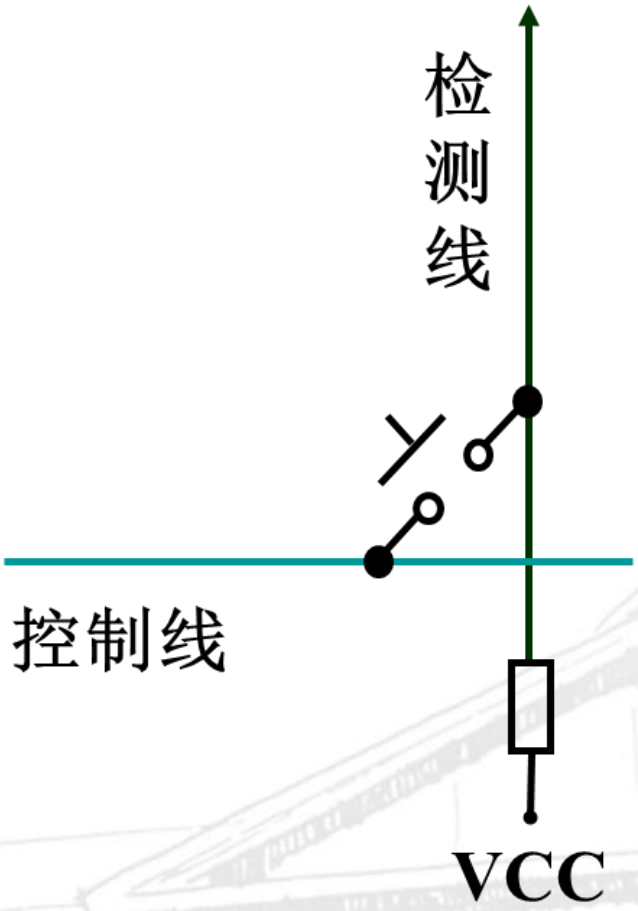
➢ 常用的矩阵结构键盘

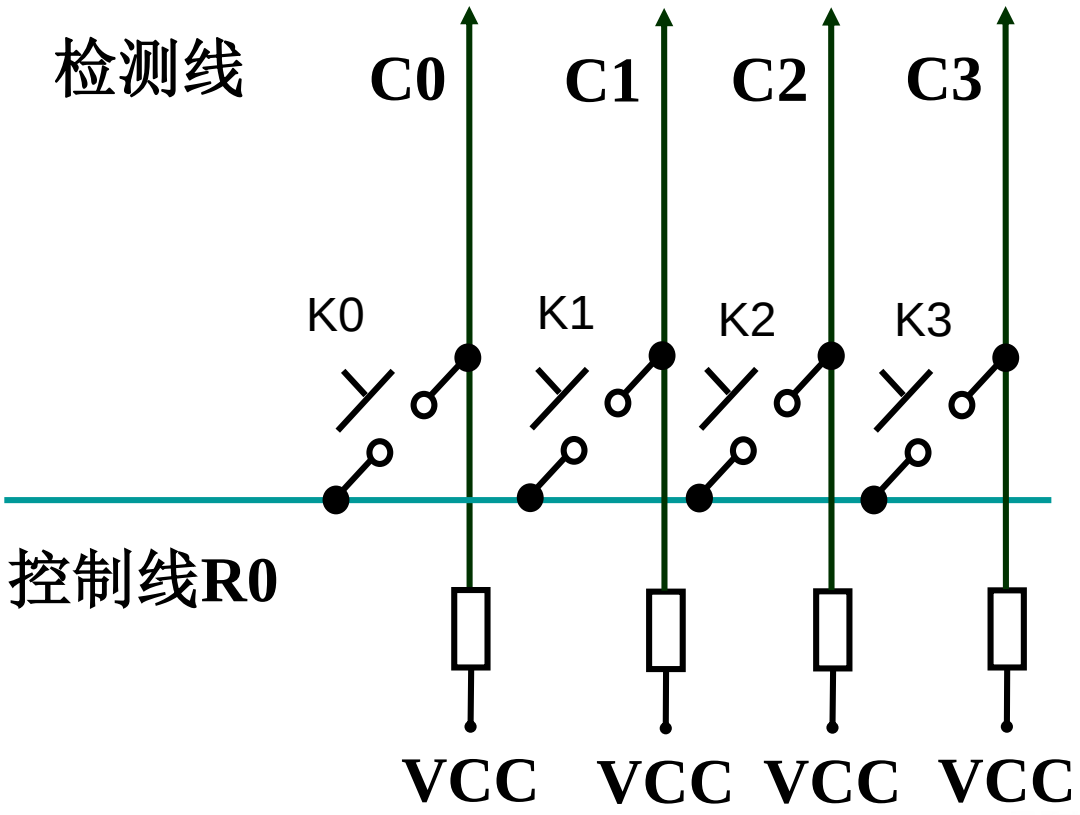
- 每一行连接一个引脚
- 每一列连接一个引脚
- 利用控制线为低、读取检测线来识别闭合键



矩阵键盘 按键检测

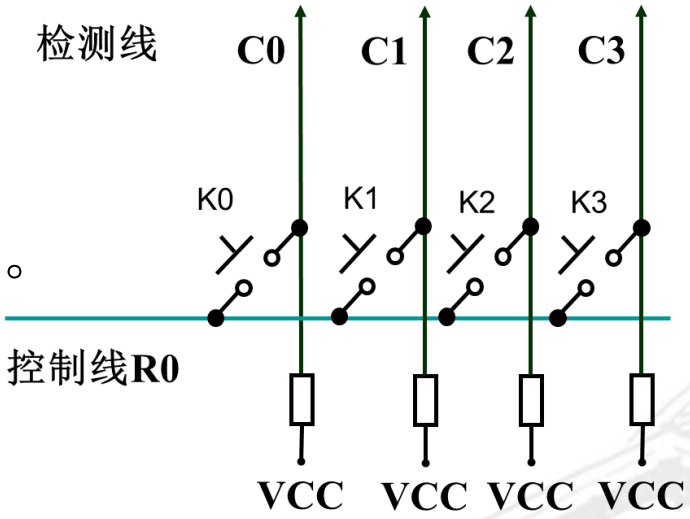
- 行线作为控制线，列线为检测线；
- 按键按下时，它所在的控制线与检测线相短接；
- 用上拉电阻把检测线上拉至VCC；
- 如果在控制线上输出低电平：按键没有按下时，检测线上的电压为高电平。按键按下时，检测线上的电压为低电平；
- 如果在控制线上输出高电平。那么无论有没有按键按下，检测线都是高电平。

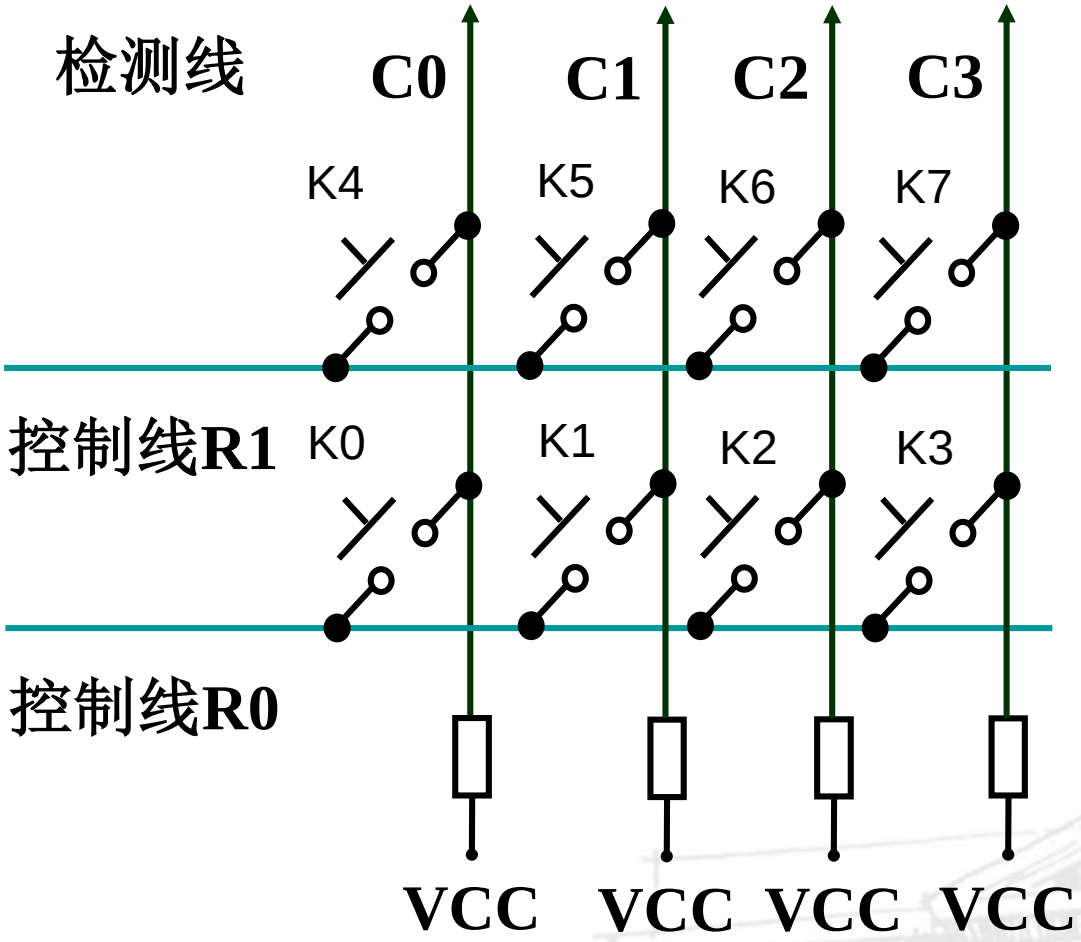



检测线
C0
C1
C2
C3
K0
K1
K2
K3
控制线R0
VCC
VCC
VCC
VCC

矩阵键盘 按键检测

- 行线作为控制线，列线为检测线；
- 按键按下时，它所在的控制线与检测线相短接；
- 用上拉电阻把检测线上拉至VCC；
- 如果在控制线上输出低电平：按键没有按下时，检测线上的电压为高电平。按键按下时，检测线上的电压为低电平；
- 如果在控制线上输出高电平。那么无论有没有按键按下，检测线都是高电平；
- 如果一行上有不同的按键按下，那么四个检测线上的电平也会有所不同。因此我们可以分辨按键在哪一个列。

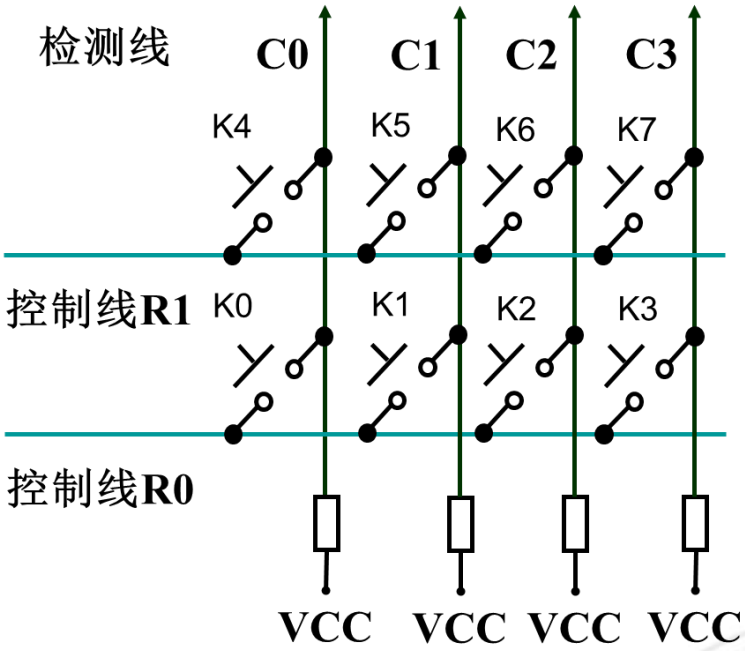



检测线
C0
C1
C2
C3
K4
K5
K6
K7
控制线R1
K0
K1
K2
K3
控制线R0
VCC
VCC
VCC
VCC

矩阵键盘 按键检测

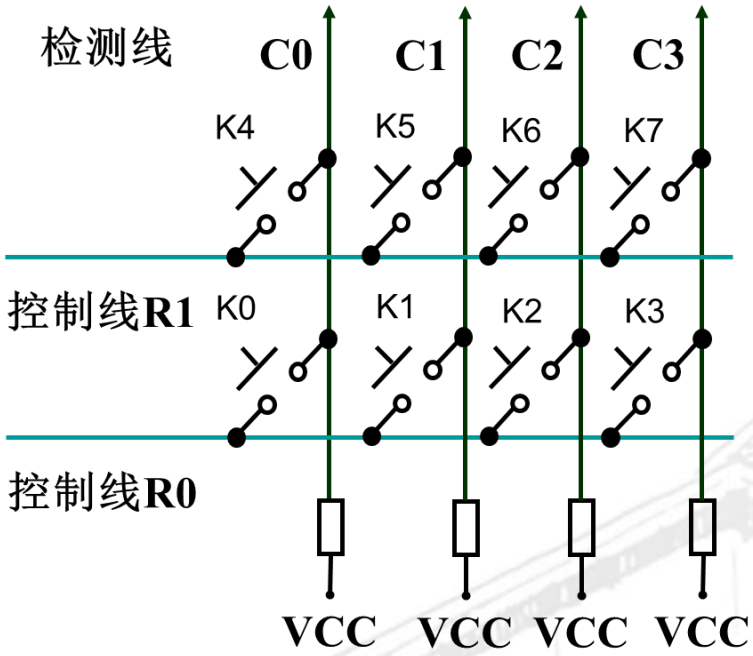
– 多个行；

– 如果控制线R0和R1都输出电平，那么如果检测到C1为低电平，还是无法区分是K1按下还是K5按下。



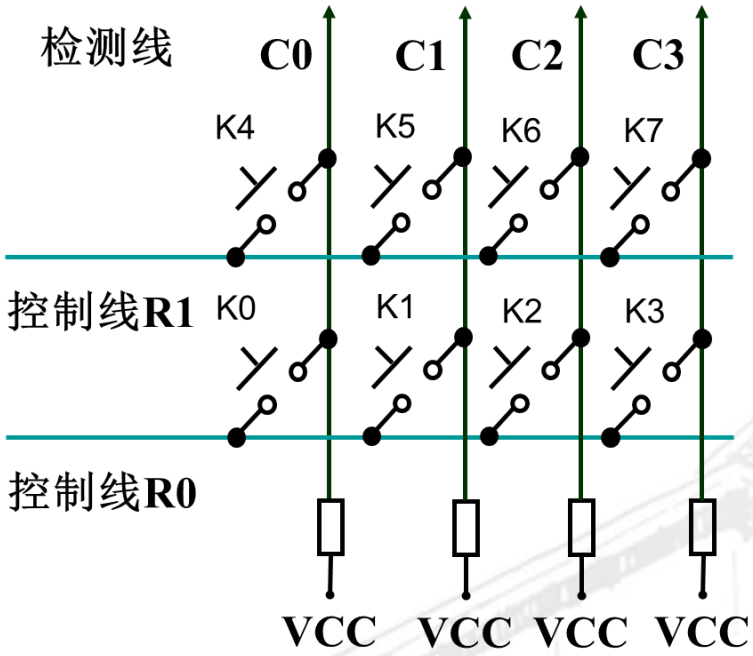
扫描法

- 先把R0设置输出低电平，R1设置输出高电平。如果C0-C3上检测到了低电平，说明按键在R0行。通过C0-C3的电平，可以判断是K0-K3中的哪个按键被按下；
- 然后把R0设置输出高电平，R1设置输出低电平。如果C0-C3上检测到了低电平，说明按键在R1行。通过C0-C3的电平，可以判断是K4-K7中的哪个按键被按下；
- 如果把R2、R3行画出来，也是以此类推；
- 扫描法是最常用的方法。



反转法

- 先把R0、R1…都设置为输出、然后输出低电平；
- 把C0、C1、C2、C3 都设置为输入，上拉；
- 如果C0-C3上检测到低电平，可以通过C0-C3的电平，判断按键在哪一列 M;
- 然后把C0、C1、C2、C3都设置为输出，然后输出低电平；
- 把R0、R1…都设置为输入，上拉；
- 如果R0、R1…都上检测到低电平，可以通过R0、R1…都的电平，判断按键在哪一行 N;
- 这样就可以检测出按键位于N行M列。

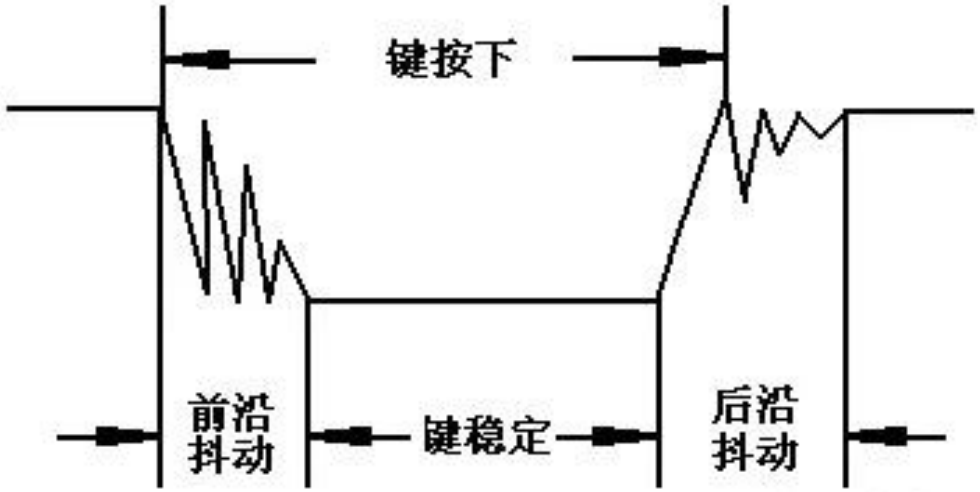


➢ 按键的抖动

– 机械按键存在抖动现象

– 当按下或释放一个键时，往往会出现按键在闭合位置和断开位置之间跳几下才稳定到闭合状态

– 抖动的持续时间通常不大于10ms。



– 采用硬件消抖电路或软件延时方法解决

重键

– 两个或多个键同时闭合

– 出现重键时，读取的键值必然出现一个以上的0，需要根据应用选择合适的办法应对重键。

- 简单情况：不予识别，认为是错误的按键
- 通常情况：只承认先识别出来的键
- 正常的组合键：都识别出来

➢ 实验套件底板上的矩阵键盘

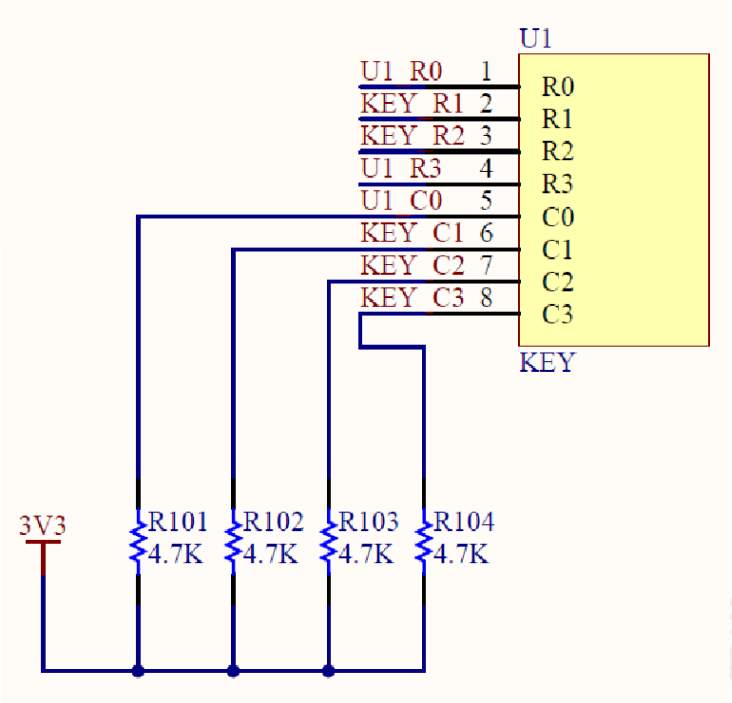
– 行悬空

– 列被上拉

– 所以行是控制线，列是检测线

表 3 TM4C1294 与键盘连接引脚

Boosterpack 引脚序号	GPIO	键盘引脚
D1_6	PD1	ROW_0
A2_5	PD4	ROW_1
A2_6	PD5	ROW_2
B1_7	PD7	ROW_3
D1_10	PP2	column_0
D2_8	PP3	column_1
A2_8	PP4	column_2
D2_3	PP5	column_3



– 将PD1、PD4、PD5、PD7设置为输出

– 将PP2、PP3、PP4、PP5设置为输入

– PD7的设置比较特殊，需要先解锁

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);
///////解锁PD7//////
HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
HWREG(GPIO_PORTD_BASE + GPIO_O_CR)  = 0x80;
GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_7);
HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
HWREG(GPIO_PORTD_BASE + GPIO_O_CR)   = 0x00;
HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = 0;
///////解锁PD7//////
GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE,
GPIO_PIN_1|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_7);
GPIOPinTypeGPIOInput(GPIO_PORTP_BASE,
GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5);     //初始化键盘行、列引脚
```

Table 10-1. GPIO Pins With Special Considerations

GPIO Pins	Default Reset State	GPIOAFSEL	GPIODEN	GPIOPDR	GPIOPUR	GPIOPCTL	GPIOCR
PC[3:0]	JTAG/SWD	1	1	0	1	0x1	0
PD[7]	GPIO[a]	0	0	0	0	0x0	0
PE[7]	GPIO[a]	0	0	0	0	0x0	0

a. This pin is configured as a GPIO by default but is locked and can only be reprogrammed by unlocking the pin in the **GPIOLOCK** register and uncommitting it by setting the **GPIOCR** register.

➢ 底板提供的键盘扫描的示例函数

– 定义一个函数KeyWrite，将输入数据ui8Val的八个位，由低到高，依次赋值给PD1、PD4、PD5、PD7、PP2、PP3、PP4、PP5。

```
void
KeyWrite(uint8_t ui8Val)
{
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1, (ui8Val & 0x01)*2);//将ui8Val最低位写入PD1
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, (ui8Val & 0x02)*8);//将ui8Val倒数第二位写入PD4
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_5, (ui8Val & 0x04)*8);//将ui8Val倒数第三位写入PD5
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_7, (ui8Val & 0x08)*16);//将ui8Val倒数第四位写入PD7
        GPIOPinWrite(GPIO_PORTP_BASE, GPIO_PIN_2, (ui8Val & 0x10)/4);//将ui8Val倒数第五位写入PP2
        GPIOPinWrite(GPIO_PORTP_BASE, GPIO_PIN_3, (ui8Val & 0x20)/4);//将ui8Val倒数第六位写入PP3
        GPIOPinWrite(GPIO_PORTP_BASE, GPIO_PIN_4, (ui8Val & 0x40)/4);//将ui8Val倒数第七位写入PP4
        GPIOPinWrite(GPIO_PORTP_BASE, GPIO_PIN_5, (ui8Val & 0x80)/4);//将ui8Val最高位写入PP5

}
```

➢ 底板提供的键盘扫描的示例函数

– 定义一个函数KeyRead，将键盘的引脚的数据读出来，返回值的从低位到高位依次是PP5,PP4,PP3,PP2,PD7,PD5,PD4,PD1的电平状态

```
uint8_t KeyRead(void)
{
uint8_t temp;
int32_t temp1,temp2;
temp1=GPIOPinRead(GPIO_PORTD_BASE,
GPIO_PIN_1|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_7);
temp2=GPIOPinRead(GPIO_PORTP_BASE,
GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5);
temp=(temp1&0x02)/2+(temp1&0x30)/8+(temp1&0x80)/16+(temp2&0x3C)*4;//temp从最高位到最低位的值分别等于PP5,PP4,PP3,PP2,PD7,PD5,PD4,PD1
return temp;
}
```

➢ 底板提供的键盘扫描的示例函数

– 定义一个函数check_key，检测是哪个按键按下

```
void check_key(void)
{
        unsigned char row ,col,tempout,tempin,tmp1,tmp2;
        // tmp1用来设置键盘引脚输出值，使列引脚电平低四位中有一个为0
        tmp1 = 0x08;
        for(row=0;row<4;row++)
        {
                KeyWrite(0x0F);//输出引脚电平低四位为全1
                tempout=0x0F-tmp1;
                KeyWrite(tempout);//输出引脚电平低四位有一个为0
                tmp1 /= 2; // tmp1 右移一位
                tempin=KeyRead();
                if ((tempin & 0xF0) < 0xF0)// 是否键盘引脚高四位有一个为0
                {
                        tmp2 = 0x10; // tmp2用于检测出哪一位为0
                        for(col =0;col<4;col++)// 列检测
                        {
                                if((tempin & tmp2)==0x00)// 是否是该列
                                {
                                        key_val = row*4 + col;  // 获取键值 退出循环
                                        break;
                                }
                                tmp2 *= 2; // tmp2左移一位
                        }
                }
        }
}
```

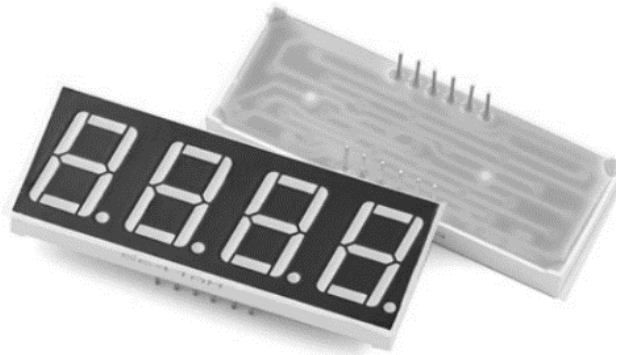
➢ 底板提供的键盘扫描的示例函数

– 定义key_event函数消抖：不能直接使用check_key函数读取按键，还需要加入消抖的环节。

```
void key_event(void)
{
        unsigned char tmp;
        KeyWrite(0xF0);// 设置键盘引脚输出值 使得如果有键按下 键盘引脚电平将不为全高
        tmp = KeyRead();
        if ((key_pressed == 0x00)&&((tmp & 0xF0) < 0xF0))//是否有键按下
        {
                key_pressed = 1; // 如果有按键按下，设置key_pressed标识
                SysCtlDelay(1000000); //消除抖动
                check_key(); // 调用check_key(),获取键值
        }
        else if ((key_pressed ==1)&&((tmp & 0xF0) == 0xF0))//是否按键已经释放
        {
                key_pressed = 0; // 清除key_pressed标识
                key_flag = 1;
        }
}
```

➢ 数码管

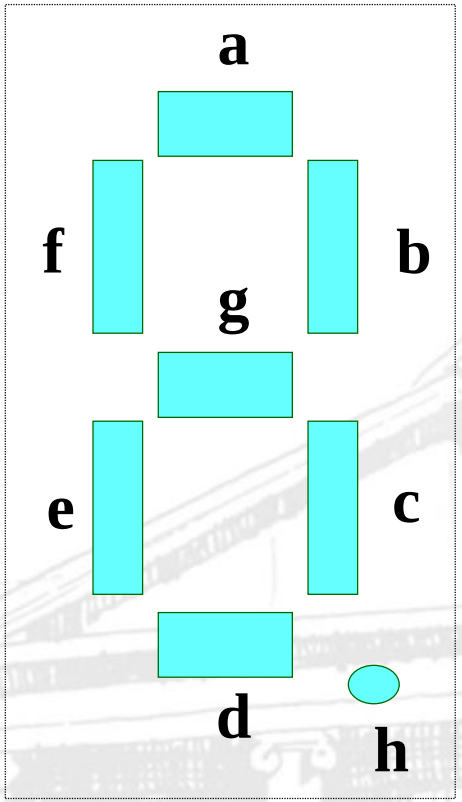
– 发光二极管LED是最简单的显示设备，由7段LED就可以组成的LED数码管，一般还有一个LED代表小数点



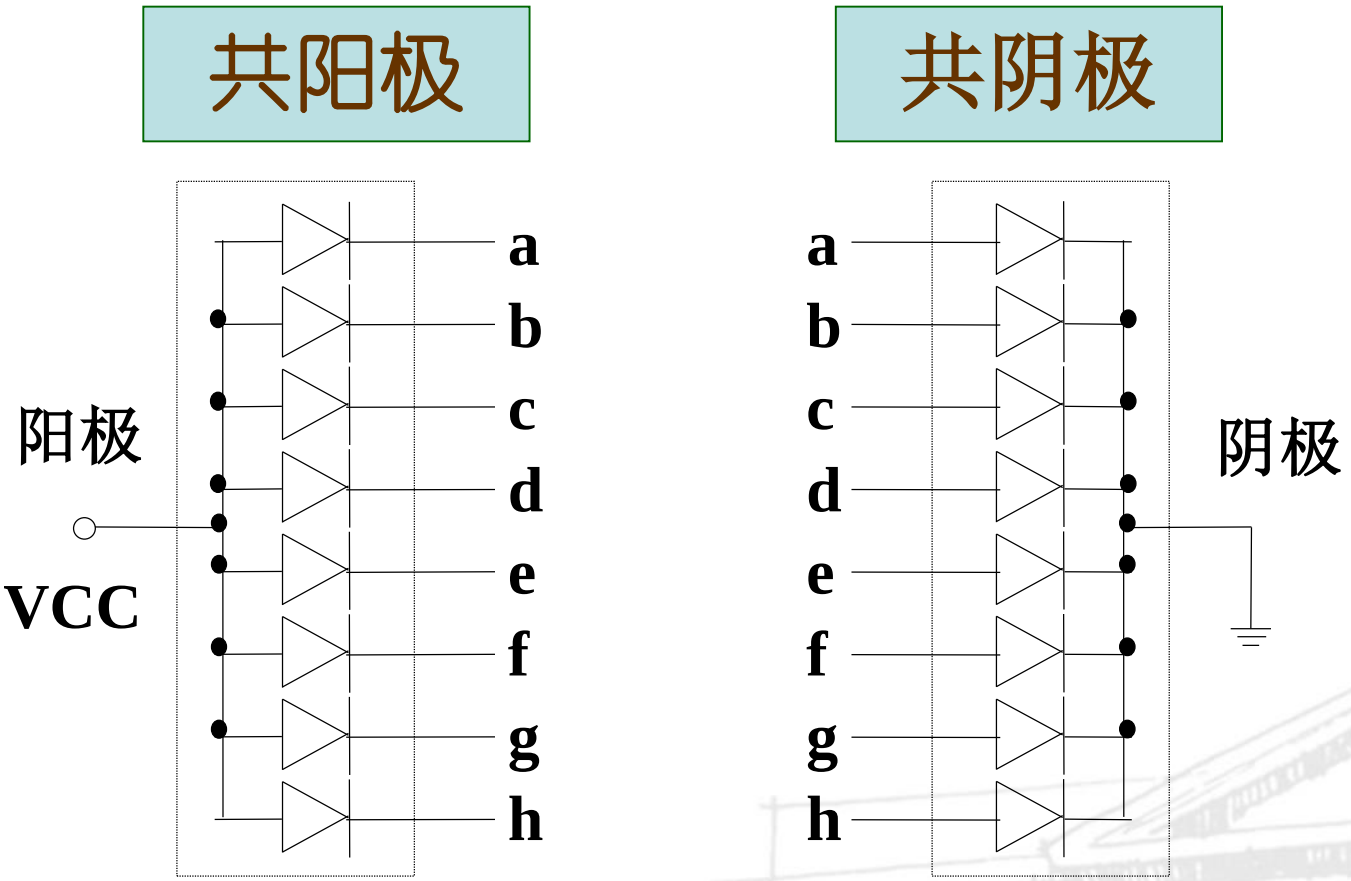
– LED数码管广泛用于微机系统，作为显示设备。

LED数码管的结构

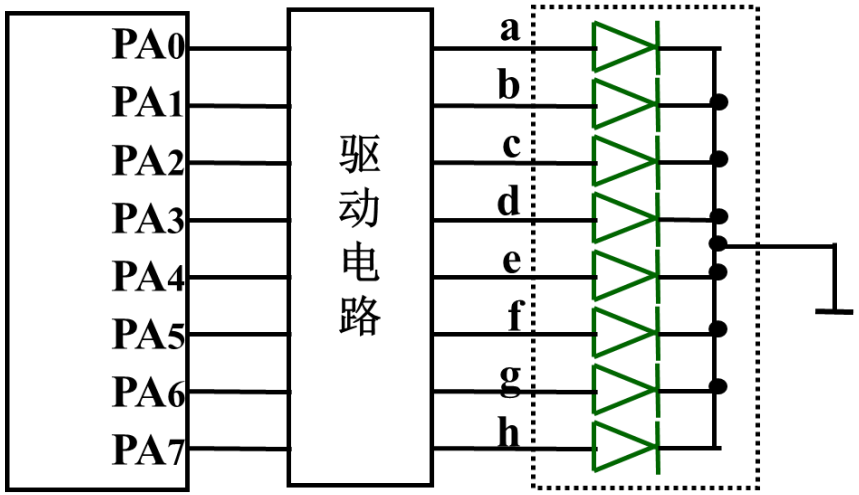
- 主要部分是7段发光管
- 顺时针分别称为a、b、c、d、e、f、g
- 有的产品还附带有一个小数点h
- 通过7个发光段的不同组合
  - 主要显示0～9
  - 也可以显示A～F（实现16进制数的显示）
  - 还可以显示个别特殊字符，如－、P 等



LED数码管的结构



➢ 驱动电路

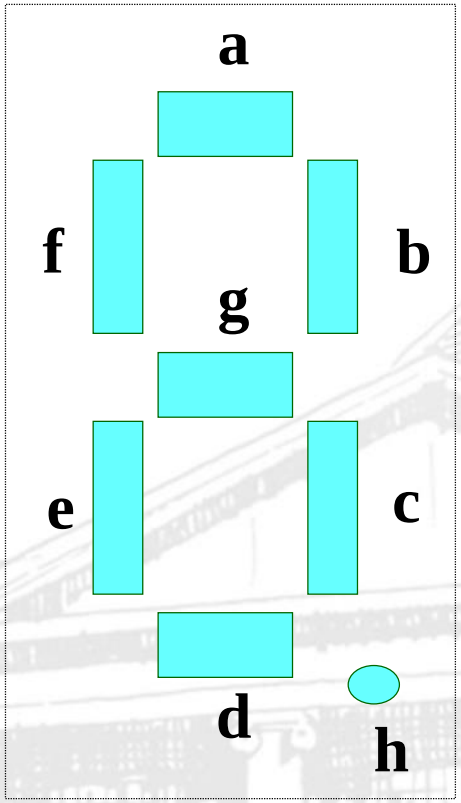

PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
驱动电路
a
b
c
d
e
f
g
h


R0 200
SEG0
SEG1
SEG2
SEG3
SEG4
SEG5
SEG6
SEG7
10
11
13
14
15
16
17
18

➢ 数码管的显示

– 通过不同的编码，显示不同的数组和字符：

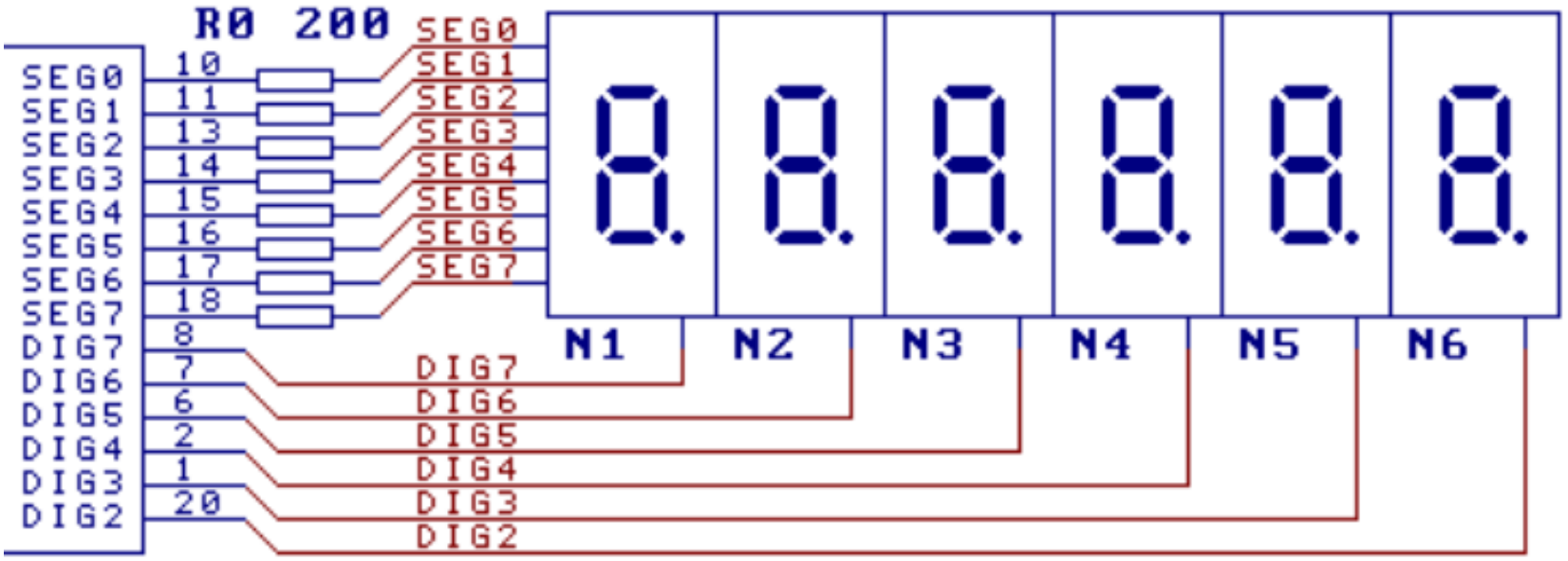
显示字形	共阳极字段码 gfedcba	共阴极字段码 gfedcba	显示字形	共阳极字段码 gfedcba	共阴极字段码 gfedcba
0	C0H	3FH	9	90H	6FH
1	F9H	06H	A	88H	77H
2	A4H	5BH	b	83H	7CH
3	B0H	4FH	C	C6H	39H
4	99H	66H	d	A1H	5EH
5	92H	6DH	E	86H	79H
6	82H	7DH	F	8EH	71H
7	F8H	07H	“熄灭”	FFH	00H
8	80H	7FH			



➢ 多个LED数码管的显示

– 共阴极连接，轮流控制N1到N6的阳极为高电平，每次只显示一个数字

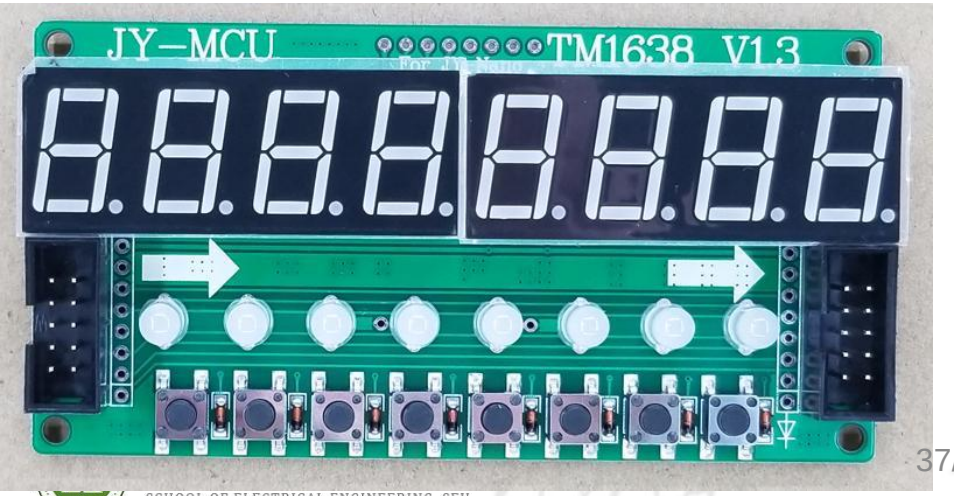
– 快速轮流显示N1到N6，就可以利用视觉暂留现象，以为6个数字同时显示了

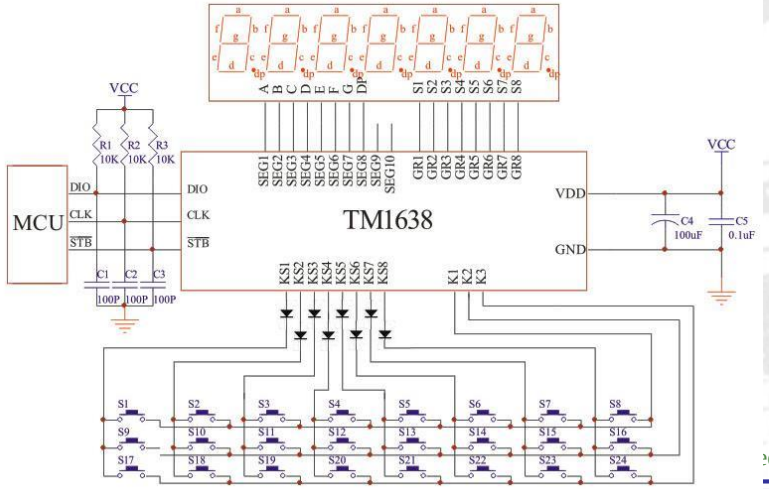


矩阵键盘和数码管的应用

- 矩阵键盘和数码管，即使使用了扫描功能，依旧占用了过多的引脚。
- 在嵌入式系统中，主控MCU一般不会使用宝贵的引脚资源直接控制矩阵键盘和数码管。
  - 通过异步串行通信端口控制的显示模块，除电源VCC和GND之外，只需要两个引脚

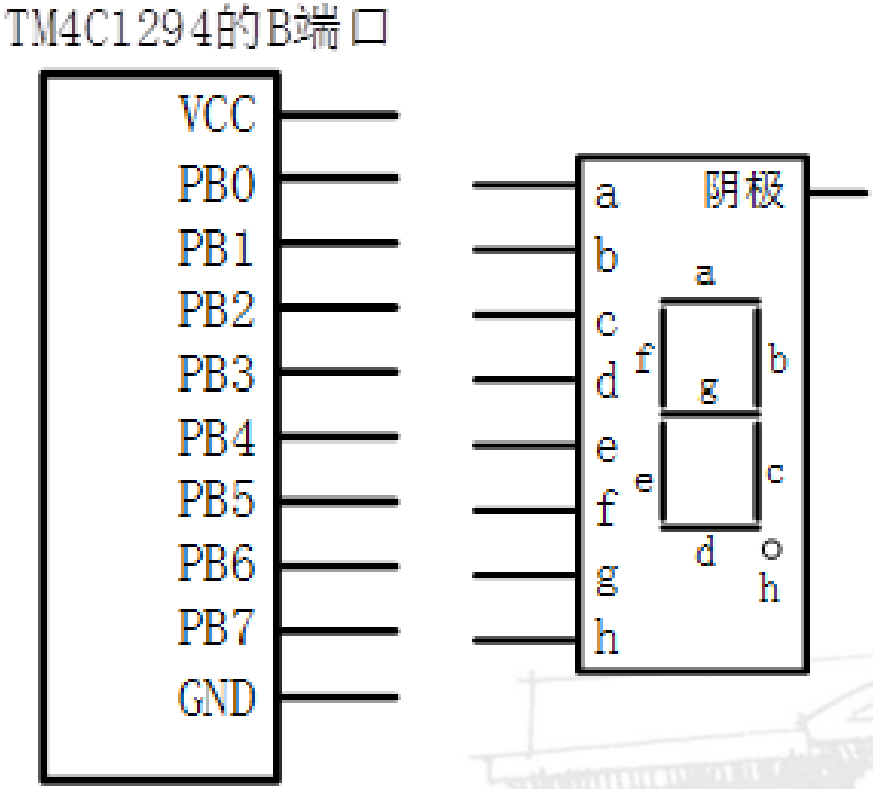
  - 通过专用芯片TM1638控制的数码管和键盘模块，与主控MCU之间，通过IIC总线通信，除电源VCC和GND之外，只需要两个引脚





作业

– 使用PB口驱动一个7段数码管，画出端口与数码管的电路接线。写出端口的初始化程序。在数码管上显示数字7，写出相关的程序代码。



谢谢！