第四讲 外部中断的学习

提要：主要学习 ATmega8 的 PD 口的 PD2、PD3 两端口的第二功能外部中断。

前面我们学习了 ATmega8 的 I/O 口作为通用数字输入/输出口来用时对 LED 数码管控制和扫描按键的应用。但 ATmega8 多数的 I/O 口都是复用口，除了作为通用数字 I/O 使用，还有其第二功能，下面我们就先来学习PD2、PD3 两端口的第二功能：

4.1 外部中断的特点：

PD2 端口是外部中断源 0，PD3 端口是外部源 1。ATmega8 的外部中断就是由这两个引脚触发的。要注意的是：如果设置允许外部中断产生，即使是 INT0 和 INT1 引脚设置为输出方式，外部中断还是会触发的。外部中断的触发方式有三种可选择：上升沿触发、下降沿触发和低电平触发。具体方式是由 MCU 控制寄存器 MCUCR 以及 MCU 控制和状态寄存器 MCUCSR 决定的。当允许外部中断且设置为电平触发方式时，只有中断输入引脚保持低电平，就将一直触发产生中断。而对于上升沿或下降沿的中断触发，则需要 I/O 时钟信号的存在。

要使用外部中断我们首先要了解几个寄存器：

1 AVR的状态寄存器 SREG

2 MCU 控制寄存器 MCUCR

3 通用中断控制寄存器 GICR

4 通用中断标志寄存器 GIFR

1 AVR的状态寄存器 SREG 的每位都是一个标志位，这里先介绍位 7—I（全局中断允许位），该位为 1 时全局中断始能允许，单独的中断使能则有对应的中断寄存器控制。如果该位为 0 则不论单独允许位是否置 1，所有中断都被禁止，系统将不相应任何中断。因而要使用外部中断首先要对该位置 1。

2 控制寄存器 MCUCR 的位 0、1（ISC00、ISC01）是外部中断 0 的中断方式控制位 0 和位 1。其设置可参见下表：

ISC01	ISC00	INT0 的中断方式
0	0	INT0 的低电平产生一个中断请求
0	1	INT0 的下降沿和上升沿都产生一个中断请求
1	0	INT0 的下降沿产生一个中断请求
1	1	INT0 的上升沿都产生一个中断请求

MCUCR 的位 2、3（ISC10、ISC11）是外部中断 1 的中断方式控制位 0 和位 1。其设置与中断 0 的控制方式类似。

3 通用中断控制寄存器 GICR 的位 6—INT0 和位 7—INT1 分别控制外部中断 0 和中断 1 的使能。当上面提到的状态寄存器 SREG 的 I 位（全局中断允许位）置 1 时，INT0 置 1 则外部引脚中断 0 使能；INT1 置 1 则外部引脚中断 1 使能。

4 通用中断标志寄存器 GIFR 的为 6—INTF0 和位 7—INTF1 是外部中断 0 和外部中断 1 的标志位。当 INT0 引脚上的有效事件触发一个中断请求后，INTF0 位会变成 1。如果全局中断使能且外部中断 0 使能，则 MCU 将跳至相应的中断向量处开始执行中断服务程序，同时硬件自动将 INTF0 标志位清零。当外部中断 0 被设置为低电平触发方式时，标志 INTF0 位将始终为 0。

上面提到中断向量表，这里简单介绍一下：Atmega8 共有 18 个中断源，Flash 程序存储器空间的最低位置(0x000—0x012)定义为复位和中断向量空间，也就是

说把中断函数的地址保存在这里，当中断发生后就到这里找到对应函数的地址，然后去执行对应的函数。这里先把中断向量表给出来，我们熟悉一下：

中断向量号	向量地址	中断源	中断定义
1	0x0000	REST	上电、外部、BOD、看门狗复位
2	0x001	INT0	外部中断请求 0
3	0x002	INT1	外部中断请求 1
4	0x003	TIMER2_COMP	时钟/计数器 2 比较匹配
5	0x004	TIMER2_OVF	时钟/计数器 2 溢出
6	0x005	TIMER1_CAPT	时钟/计数器 2 捕获事件
7	0x006	TIMER1 COMPA	时钟/计数器 1 比较匹配 A
8	0x007	TIMER1 COMPB	时钟/计数器 1 比较匹配 B
9	0x008	TIMER1_OVF	时钟/计数器 1 溢出
10	0x009	TIMER0_OVF	时钟/计数器 0 溢出
11	0x00A	SPI，STC	SPI 串行传输完成
12	0x00B	USART，RCX USART	，Rx 完成
13	0x00C	USART, UDRE	USART，寄存器空
14	0x00D	USART，TXC USART	，Tx 完成
15	0x00E	ADC	ADC 转换完成
16	0x00F	EE_RDY	EEPROM 准备好
17	0x010	ANA_COMP	模拟比较
18	0x011	TWI	两线串行接口(IIC)
19	0x012	SPM_RDY	写程序存储器准备好

在此中断向量表中，处于低地址的中断向量对应的中断优先级高，所以系统复位 REST 拥有最高优先级；外部中断 0 的优先级高于外部中断 1。其中系统复位 REST 不是中断。

用 ICCAVR 的编程，在 C 中只要用#pragma 伪指令和中断向量说明中断服务程序的入口地址即可：

```
#pragma interrupt_handler <函数名>:<中断向量>
```

例如要定义使用 INT0 中断服务程序：

```
#pragma interrupt_handler int0_fun:2
void int0_fun(void)
{
    ……
}
```

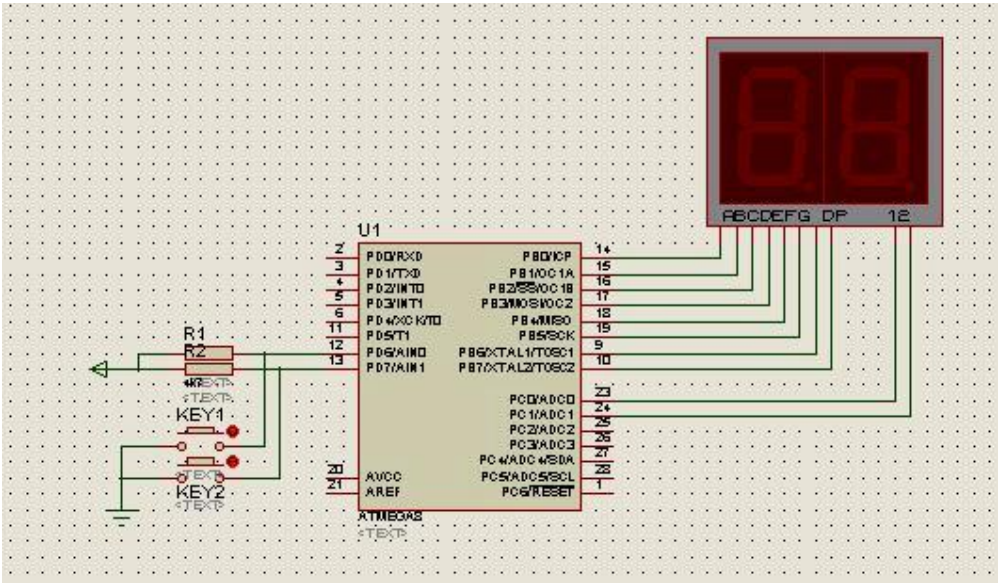
这里的中断向量就对应到该函数是会被那个中断调用，例如上面的 2 修改为 3 的话就变成 INT1 的中断服务程序了，因此也可以让多个中断调用同一个函数。例如：

```
#pragma interrupt_handler int_fun:2
#pragma interrupt_handler int_fun:3
```

就是外部中断 0 和中断 1 都调用 int_fun 函数。

4.2 应用实例----中断计数器

好了，前面的理论知识就先介绍这么多，接下来我们就做个小例子。我们用两个按键作为两个外部中断的触发源，为了复习前面的知识，我们再接一个 LED 数码管用来显示数据，我先画出电路图：



根据上面的图，外部中断 0 设置为下降沿触发，中断 1 设置为低电平触发，调用同一个中断函数，在中断函数中做数值加 1，然后在 LED 数码管中显示。我们可以比较两种中断方式的不同，好想一想我们的程序应该这么写呢？

```
/*****************************************/
//文件名：Int0.c
//功能：  外部中断的应用
//作者：  young
//时间：  2006.10.1
//目标MCU:ATmega8
//晶振：  8MHZ
/*****************************************/
#include <iom8v.h>
#include <macros.h>
#include "Delay.h"

unsigned char CountNum;   //全局变量，用来计数

#pragma interrupt_handler int_fun:2
#pragma interrupt_handler int_fun:3
void int_fun(void)
{
    if(++CountNum>=100)
        CountNum=0;
}
```

```
//主函数，显示数据时先关中断，然后再打开
void main()
{
    unsigned char temp,temp2;
    unsigned char num[10]={0x3F, 0x06, 0x5B, 0x4F, 0x66,
                           0x6D, 0x7D, 0x07, 0x7F, 0x6F};
    //初始化端口
    DDRB=0xFF;    //设置B口为输出模式
    PORTB=0xFF;   //置高电平
    DDRC=0x03;
    PORTC=0xFF;
    DDRD=0xFF;
    PORTD=0xFF;

    MCUCR=0x02;   //INT0下降沿触发，INT1低电平有效
    GICR|=0xC0;   //打开INT0,INT1中断
    GIFR=0xC0;    //清除INT0、INT1中断标志位

    CountNum=0;   //初始化全局变量
    SEI();        //打开全局中断
    while (1)
    {
        CLI();    //关闭全局中断
        temp=CountNum/10;
        PORTC&=~(1);
        PORTB=num[temp];
        delay_ms(1);
        PORTC|=0xFF;
        temp=CountNum%10;
        SEI();
        PORTC&=~(2);
        PORTB=num[temp];
        delay_ms(1);
        PORTC|=0xFF;
        SEI();   //打开全局中断
        delay_ms(1);
     }
}
```

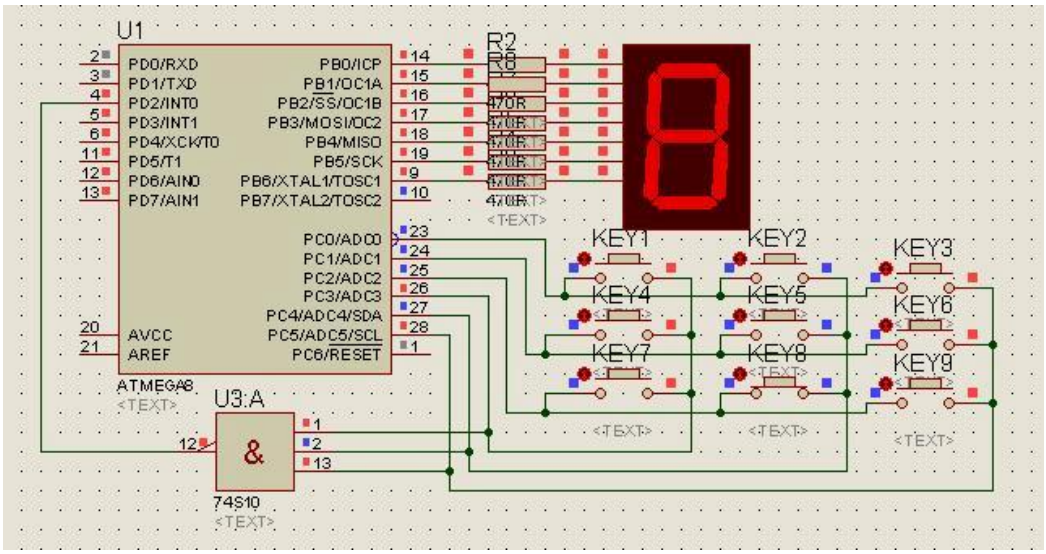
怎么样，你的程序也跑起来了吗，现在对外部中断已经基本了解了吧。其实如果在中断的端口接上振荡源，我们就可以在中断函数里面计数做定时器，这样的定时器要比我们现在用的 delay_ms 函数准确多了，当然 ATmega8 本身就带有三个定时/计数器我们后面会陆续学到。

接下来我们就可以利用中断复习一下前面扫描键盘的内容。

4.3 中断触发键盘扫描

这个内容要简单些，我们就是要使按下键 盘中的任一个按键就触发一个中断，然后在中断函数中来调用键盘处理函数。

好了，我先画出电路图：



其中比前一章的 3X3Key 的实例多了一个 74S10 的与非门，作用是任一个按键按下都可以触发一个 INT0 中断。

我们要实现的内容是：任一个按键按下触发一个 INT0 中断，INT0 设置为下降沿触发方式，在中断中做一个标志，表示有按键按下，然后在主函数中判断该标志位，有按键按下，消除抖动干扰，再做确认哪个按键按下，最后在 LED 数码管上显示按键的值。好了先想一想该怎么做，我给出一个参考：

```
/*****************************************/
//文件名：IntKey.c
//功能：  中断触发键盘扫描的应用
//作者：  young
//时间：  2006.11.6
//目标MCU:ATmega8
//晶振：  8MHZ
/*****************************************/
#include <iom8v.h>
#include <macros.h>
#include "Delay.h"

unsigned char KeyDown;
```

```
//按键扫描函数，返回按键的值
unsigned char ScanKey(void)
{
    unsigned char temp,temp1,key;
    temp=PINC;
    temp&=0x38;
    switch(temp) //判断行中哪条线有低电平
    {
    case 0x30:
        DDRC=0X38;
        PORTC=0X07;
        delay_us(1);
        temp1=PINC;
        temp1&=0x07;
        switch(temp1) //判断列中哪条线有低电平
        {
            case 0x06: key=0x01;  //得到键值
                break;
            case 0x05: key=0x04;
                break;
            case 0x03: key=0x07;
                break;
            default:   key=0;
                break;
        }
        DDRC=0X07;
        PORTC=0X38;
        break;
```

```
    case 0x28:
        DDRC=0X38;
        PORTC=0X07;
        delay_us(1);
        temp1=PINC;
        temp1&=0x07;
        switch(temp1)
        {
        case 0x06: key=0x02;
            break;
        case 0x05: key=0x05;
            break;
        case 0x03: key=0x08;
            break;
        default:   key=0;
            break;
        }
        DDRC=0X07;
        PORTC=0X38;
        break;
```

```
    case 0x18:
        DDRC=0X38;
        PORTC=0X07;
        delay_us(1);
        temp1=PINC;
        temp1&=0x07;
        switch(temp1)
        {
        case 0x06: key=0x03;
            break;
        case 0x05: key=0x06;
            break;
        case 0x03: key=0x09;
            break;
        default:   key=0;
            break;
        }
        DDRC=0X07;
        PORTC=0X38;
        break;
    default:
        key=0;
        break;
    }
    return (key);
}
```

```
//中断函数，设置一个标志，表示按键按下
#pragma interrupt_handler int_fun:2
void int_fun(void)
{
    KeyDown=1;      //在中断中仅设置一个标志
}

//主函数，扫描按键显示数据
void main()
{
    unsigned char temp, keynum;
    unsigned char num[10]={0x3F, 0x06, 0x5B, 0x4F, 0x66,
                           0x6D, 0x7D, 0x07, 0x7F, 0x6F};
    //初始化端口
    DDRB=0xFF;    //设置B口为输出模式
    PORTB=0xFF;   //置高电平
    DDRC=0X07;
    PORTC=0X38;
    DDRD&=0x0F;
    PORTD|=0xFC;

    MCUCR=0x03;   //INT0上升沿触发，INT1低电平有效
    GICR|=0x40;   //打开INT0中断
    GIFR=0xC0;    //清除INT0、INT1中断标志位
```

```
    KeyDown=0;      //初始化全局变量
    SEI();          //打开全局中断
    while(1)
    {
       PORTB=0x40;
       if(KeyDown==1)          //检测是否有按键按下
       {
           GIFR&=0xCF;         //关闭INT0中断
           KeyDown=0;
           delay_ms(5);
           temp=PINC;
           temp&=0x38;
           if(temp==0x38)       //确认是否有按键按下
           {
               GICR|=0x40;     //打开INT0中断
               continue;
           }
           keynum=ScanKey();
           PORTB=num[keynum];
           while(temp!=0x38)
           {
              temp=PINC;
              temp&=0x38;
           }
           GICR|=0x40;         //打开INT0中断
           DDRC=0X07;
           PORTC=0X38;
       }
    }
}
```

这里的 ScanKey 函数比前面的例子有一点改动，是为了画图的方便；在中断函数里只是做一个标志，而不是去确定哪个按键按下，在 LED 数码管中显示按键值是因为，中断函数要做到占用时间越短越好，而消除抖动干扰的时候要等待 10MS 后再做确认，所以，我放在主程序里处理了。另外在主函数中处理确认按键和显示按键值时做了一次关中断和开中断的操作，应注意的是，关中断后要对应的打开中断，否则就再也不能进入中断了。

好了，关于外部中断的应用我想你现在熟悉的已经差不多了吧。关于中断的应用后面还有很多，我用到时再说吧。