第七章　中断系统与基本应用(节选)

7.4　外部中断应用实例

例 7.1 用按键控制的一位 LED 数码管显示系统

1）　硬件电路

图 7-2 为硬件原理图。其中 LED 数码管的控制显示连接与例 6.4 相同，PA 口工作于输出方式，作为 LED 数码管的段码输出，LED 数码管的位信号接地，因此这个一位的 LED 数码管工作于静态显示方式。图中使用了两个按键 K1、K2，按键的一端分别与 PD2（INT0）、PD3（INT1）连接。INT0 和 INT1 作为外部中断的输入，采用电平变化的下降沿触发方式，当 K1（K2）按下时，会在 PD2（PD3）引脚上产生一个高电平到低电平的跳变，触发 INT0 或 INT1 中断。

系统的功能还是控制一个 8 段数码管显示“0”－“F”16 个十六进制的数字。当系统上电时，显示“0”。K1 键的作用是加“1”控制键：按 1 次 K1 键，显示数字加 1，依次类推。当第 15 次按 K1 键时，显示“F”，第 16 次按 K1 键，显示又从“0”开始。K2 键的作用是减 1 控制键：按 1 次 K1 键，显示数字减 1，减到“0”后，再从“F”开始。

该电路可以在配套的实验板上通过连线轻松实现。

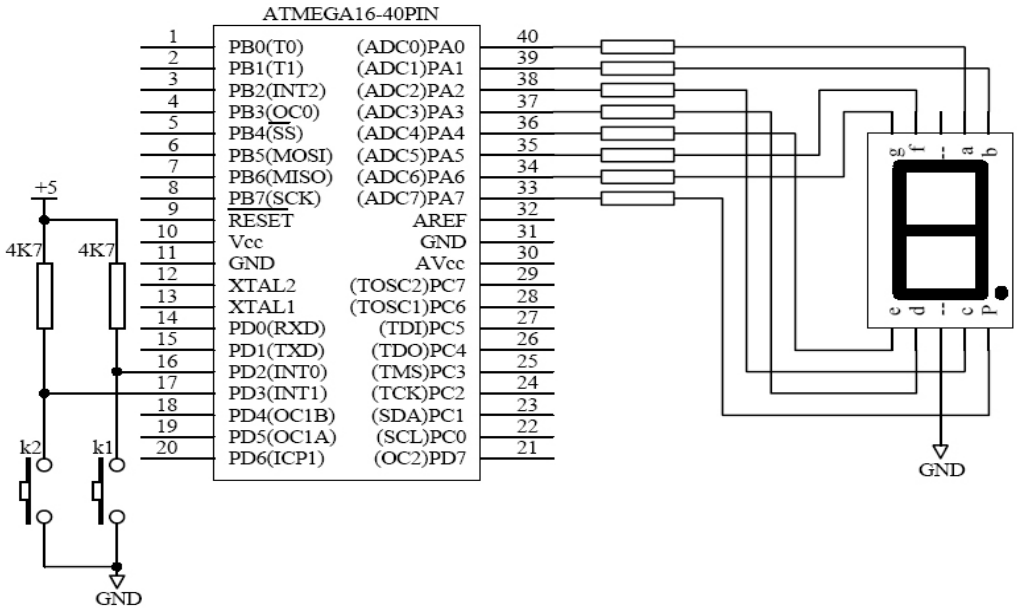

图 7-2 按键中断方式控制一位 LED 数码管显示

2）　软件设计

本实例的显示控制非常简单，主要是说明如何编写中断服务程序和掌握外部中断的基本使用方法。下面各给出一个采用汇编和 C 语言编写的系统程序。

首先是在 CVAVR 中使用 C 语言编写的程序。再次建议读者使用 CVAVR 中的程序生成向导功能来帮助你建立整个程序的框架，以及芯片的初始化部分的语句，可以省掉你过多的查看器件手册和考虑寄存器的设置值等。图 7-3 为 CVAVR 中利用程序生成向导功能配置产生外部中断初始化部分的界面。

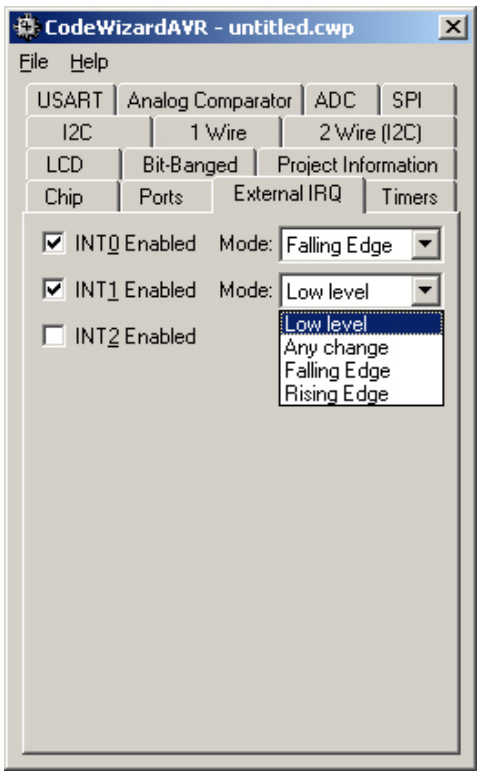

图 7-3 使用 CVAVR 的程序生成向导配置外部中断的界面

```
/*****************************************
File name           : Demo_7_1.c
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 4.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************/
#include <mega16.h>

flash char led_7[16]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
                      0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71};
char counter;

// INT0 中断服务程序
interrupt [EXT_INT0] void ext_int0_isr(void)
{
     if (++counter>=16) counter = 0;
}

// INT1 中断服务程序
interrupt [EXT_INT1] void ext_int1_isr(void)
```

```
{
    if (counter) --counter;
    else counter = 15;
}

void main(void)
{
    PORTA=0xFF;
    DDRA=0xFF;

    GICR|=0xC0;         // 允许 INT0、INT1 中断
    MCUCR=0x0A;         // INT0、INT1 下降沿触发
    GIFR=0xC0;          // 清除 INT0、INT1 中断标志位

    counter = 0;        // 计数单元初始化为 0

    #asm("sei")         // 全局中断允许

    while (1)
    {
        PORTA = led_7[counter];     // 显示计数单元
    };
}
```

上面的程序，就是先利用 CVAVR 的程序生成向导功能进行配置，然后在它生成的程序框架基础上完成的。程序中定义了一个计数变量 counter，执行一次 INT0 中断服务程序，counter 加 1，而执行一次 INT1 中断服务程序，counter 减 1。在主程序中只是显示 counter 的值。INT0、INT1 初始化为电平变化的下降沿触发。

下面给出使用 AVR 汇编编写的系统程序。汇编程序的思想方法同 C 语言的程序相同，读者可以参考本章 7.3.1 中汇编程序框架和 demo_7_1.c，仔细体会使用汇编编写系统程序时，如何正确的编写中断向量区、中断初始化设置以及中断服务程序。

```
;*********************************************************
;AVR 汇编程序实例:Demo_7_1.asm
;使用 INT0、INT1 控制 LED 数码管显示
;Mega16  4MHz
;*********************************************************
.include "m16def.inc"
.def temp     =    r23                ;临时变量
.def counter  =    r24                ;计数变量

;中断向量区配置，FLASH 空间$000~$028
.org  $000
jmp  RESET          ; 复位处理
jmp  EXT_INT0       ; IRQ0 中断向量
jmp  EXT_INT1       ; IRQ1 中断向量
```

```
reti             ; Timer2 比较中断向量
nop
reti             ; Timer2 溢出中断向量
nop
reti             ; Timer1 捕捉中断向量
nop
reti             ; Timer1 比较 A 中断向量
nop
reti             ; Timer1 比较 B 中断向量
nop
reti             ; Timer1 溢出中断向量
nop
reti             ; Timer0 溢出中断向量
nop
reti             ; SPI 传输结束中断向量
nop
reti             ; USART RX 结束中断向量
nop
reti             ; UDR 空中断向量
nop
reti             ; USART TX 结束中断向量
nop
reti             ; ADC 转换结束中断向量
nop
reti             ; EEPROM 就绪中断向量
nop
reti             ; 模拟比较器中断向量
nop
reti             ; 两线串行接口中断向量
nop
reti             ; IRQ2 中断向量
nop
reti             ; 定时器 0 比较中断向量
nop
reti             ; SPM 就绪中断向量
nop

.org  $02A
RESET:                    ; 上电初始化程序
    ldi r16, high(RAMEND)
    out SPH, r16
    ldi r16, low(RAMEND)
    out SPL, r16          ; 设置堆栈指针为 RAM 的顶部
```

```
    ser temp
    out ddra, temp          ; 设置 PORTA 为输出,段码输出
    out porta,temp          ; 设置 PORTA 输出全 1

    ldi temp, 0x0a
    out mcucr, temp         ; INT0、INT1 下降沿触发
    ldi temp, 0xc0
    out gicr, temp          ; 允许 INT0、INT1 中断
    out gifr, temp          ; 清除 INT0、INT1 中断标志位

    clr counter
    sei                     ; 使能中断

MAIN:
    clr r0
    ldi zl, low(led_7 * 2)
    ldi zh, high(led_7 * 2)     ; Z 寄存器取得 7 段码组的首指针
    add zl,counter              ; 加上要显示的数字
    adc zh,r0                   ; 加上低位进位
    lpm                         ; 读对应七段码到 R0 中
    out porta, r0               ; LED 段码输出
    rjmp MAIN                   ; 循环显示

EXT_INT0:
    in temp, sreg
    push temp               ; 中断现场保护
    inc counter             ; 计数单元加 1
    cpi counter, 0x10       ; 与 16 比较
    brne EXT_INT0_RET       ; 小于 16 转中断返回
    clr counter             ; 计数单元清 0
EXT_INT0_RET:
    pop temp
    out sreg, temp          ; 中断现场恢复
    reti                    ; 中断返回

EXT_INT1:
    in temp, sreg
    push temp               ; 中断现场保护
    dec counter             ; 计数单元减 1
    cpi counter, 0xFF       ; 与 255 比较
    brne EXT_INT1_RET       ; 未到 255 转中断返回
    ldi counter, 0x0F       ; 计数单元设置为 15
EXT_INT1_RET:
    pop temp
```

```
    out sreg, temp          ; 中断现场恢复
    reti                    ; 中断返回

led_7:                      ;7 段码表
.db 0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07
.db 0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71
```

在这两段例程中，都定义了一个计数单元变量 counter，在主程序中只是完成将该计数变量的值转换成 7 段码后输出送显示。外部中断 INT0 和 INT1 都采用引脚上电平变化的下降沿触发中断，INT0 的中断服务程序中将 counter 加一处理，而 INT1 的中断服务程序中则是将 counter 做减一处理。

3) 思考与实践

- ✓ 平稳的按下 K1 和 K2，观察显示的变化。
- ✓ 修改程序，将 INT0、INT1 的中断触发方式分别改成电平变化的下降沿触发中断，以及电平变化的触发中断和低电平触发中断，然后运行程序，使显示数据加一或减一变化，体会与使用电平变化的下降沿触发中断的情况做比较，有何不同？
- ✓ 不管使用哪种中断触发方式，经常会产生按键控制不稳定的现象，如显示为“5”时，按下 K1 键一次，应该加一显示“6”，但显示“7”或“8”，甚至更多，这是为什么？
- ✓ 查看在 CVAVR 开发环境中编写编译 demo_7_1.c 的过程中，CVAVR 都生成建立了那些文件？这些文件的内容是什么？文件的扩展名是什么？有何作用？
- ✓ 查看 CVAVR 生成的 demo_7_1.lst 的内容，回答以下问题：a)CVAVR 对中断向量是如何处理的。b)counter 变量分配在什么地方。c)CVAVR 中，如何对中断现场进行保护和恢复的，是使用硬件堆栈进行保护的吗。
- ✓ 整理出编译 demo_7_1.c 生成的汇编代码，体会宏的使用，并与 demo_7_1.asm 进行比较。
- ✓ CVAVR 在编译 demo_7_1.c 生成的汇编代码中还做了哪些工作？
- ✓ 参考 CVAVR 的 HELP，尝试采用在 CVAVR 中用嵌入汇编的方式编写 INT0 和 INT1 的中断服务程序。

例 7.2 采用外部中断方式，用外部振荡源为基准的时钟系统

1) 硬件电路

在第六章中的例 6.5“六位 LED 数码管动态扫描控制显示设计（一）”中，使用调用 CVAVR 中软件延时函数的方法给出了一个使用 6 个数码管组成时钟系统。采用软件延时，时钟是不准确的，因为一旦系统中使用了中断，就可能打断延时程序的执行，使得延时时间的变化。下面给出以外部振荡源为基准，采用外部中断方式实现的时钟系统的设计。

在“AVR-51 多功能实验开发板”上，有一个采用 CD4060 和 2.048M 晶体构成的 50%占空比、0-5v 的标准方波振荡源。我们将它 10 个标准频率中 500Hz 的输出端与 ATmega16 的 PD3 脚连接，作为外部输入信号，INT1 采用下降沿方式触发中断，那么 500 次中断就是 1 秒钟了。此时，外部 500Hz 的振荡源就是时钟系统的计时基准，这样的时钟系统比使用软件延时的方法要准确的多。

时钟系统显示部分的硬件电路是同图 6-15 相同的，只需要使用一根连线，将板上标准方波振荡源的 500Hz 输出端与 ATmega16 的 PD3 脚连接在一起。

2) 软件设计

下面是一个采用 C 编写的系统源程序。

```
/*******************************************

File name           : Demo_7_2.c
```

```
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 4.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************/
#include <mega16.h>
flash char led_7[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
flash char position[6]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf};

char time[3];                   // 时、分、秒计数单元
char dis_buff[6];               // 显示缓冲区，存放要显示的 6 个字符的段码值
int time_counter;               // 中断次数计数单元
char posit;                     //  (1)
bit point_on, time_1s_ok;

void display(void)                          // 6 位 LED 数码管动态扫描函数
{
  PORTC = 0xff;                             //  (2)
  PORTA = led_7[dis_buff[posit]];
  if (point_on && (posit==2||posit==4)) PORTA |= 0x80;
  PORTC = position[posit];
  if (++posit >=6 ) posit = 0;              //  (3)
}

// 外部中断 INT1 服务函数
interrupt [EXT_INT1] void ext_int1_isr(void)
{
  display();                                // 调用 LED 扫描显示
  if (++time_counter>=500)
```

```
  {
      time_counter = 0;                    //  (4)
      time_1s_ok = 1;                      //  (5)
  }
}

void time_to_disbuffer(void)               // 时钟时间送显示缓冲区函数
{
  char i,j=0;
  for (i=0;i<=2;i++)
  {
      dis_buff[j++] = time[i] % 10;
      dis_buff[j++] = time[i] / 10;
  }
}

void main(void)
{
  PORTA=0x00;                              // 显示控制 I/O 端口初始化
  DDRA=0xFF;
  PORTC=0x3F;
  DDRC=0x3F;

  time[2] = 23; time[1] = 58; time[0] = 55; // 设时间初值 23:58:55
  posit = 0;
  time_to_disbuffer();

  GICR|=0x80;                              // INT1 中断允许
  MCUCR=0x08;                              // INT1 下降沿触发
  GIFR=0x80;                               // 清 INT1 中断标志
  #asm("sei")                              // 全局中断允许
```

```
    while (1)
    {
        if (time_1s_ok)                    // 1秒到
        {
            time_1s_ok = 0;                //  (6)
            point_on = ~point_on;
            if (++time[0] >= 60)           // 以下时间调整
            {
                time[0] = 0;
                if (++time[1] >= 60)
                {
                    time[1] = 0;
                    if (++time[2] >= 24) time[2] = 0;
                }
            }
            time_to_disbuffer();           // 新调整好的时间送显示缓冲区
        }
    };
  }
```

3) 思考与实践

该程序同第六章的例 6.5 有许多地方系统或类似，清读者彻底、全面、读懂、理解和体会该段程序，对有（n）注释标记的语句和程序段进行分析，你是否能回答以下问题：

- ✓ Display()函数是如何工作的？与例 6.5 中的 display()函数的不同点在何处？该函数每秒钟执行几次？每执行一次的时间是多少？
- ✓ 在 INT1 的中断服务程序的一开始就调用一次 display()函数，那么整个中断服务程序的执行时间是多少？
- ✓ 可以看出，LED 数码管的显示还是动态扫描方式，那么每一位 LED 数码管在一次扫描过程中点亮的时间是多少？在 1 秒中内循环扫描 6 个数码管的次数是多少？数码管的显示会闪烁吗？
- ✓ 将 500Hz 的输入换接为 GND（0Hz）、125Hz、250Hz、2K、4K 等输入，显示有何变化？显示的时间有何变化？说明原因。
- ✓ 将 500Hz 的输入换接成 64K 的输入，观察显示。然后将 display()中第一条语句“ PORTC = 0xff; ”去掉，再运行程序（输入保持 64K），观察显示同原来有何区别？说明为什么，并给出“ PORTC = 0xff; ”语句的作用。
- ✓ 深入体会程序中的变量 time_counter、posit、time_1s_ok 的作用，以及在程序中使用

的地方。如果将程序中带有标识的（3）、（4）、（5）、（6）分别去掉，或随便改变语句出现的位置时，程序能完成正常的功能吗？说明原因。

- ✓ 将原程序中 INT1 的中断触发方式改为电平变化触发方式，时钟系统有和变化？说明原因。
- ✓ 同第六章的例 6.5 程序 demo_6_5.c 比较，有那些优点和缺点？MCU 的利用有何变化？

有能力的读者请尝试修改 demo_7_2.c 程序，在主程序中加上休眠功能。

思考与练习

1. 什么是中断？计算机采取中断有什么好处？说明中断的作用和用途。
2. 什么叫中断源？ATmega16 有那些中断源？各有什么特点？
3. 什么是中断系统？中断系统的功能是什么？
4. 中断优先级有什么作用？AVR 的中断优先级如何确定的？
5. 什么叫断点？什么叫中断现场？断点和中断现场的保护和恢复有什么意义？
6. 在 AVR 中，中断断点和中断现场的保护是如何实现的？
7. 什么是中断向量？AVR 的中断系统是如何构成的，它完成哪些功能？
8. 对于不使用的中断，它的中断向量处应如何处理？试比较放入一条“ JMP 0000 ”和放入一条“ RETI ”的区别，哪个更好些？为什么？
9. AVR 对中断采用两级控制方式，它是如何控制的？
10. AVR 响应中断是有条件的，请说出这些条件是什么。
11. 请详细说明 AVR 中断响应的全过程。在这个过程中，硬件完成了哪些工作，软件完成了哪些工作？
12. 以 AVR 外部中断使用为例，说明在中断初始化程序中需要设置那些内容，以及如何正确的编写中断的初始化程序？
13. 说明全局中断允许标志位、各个独立中断允许标志位和中断标志位的作用。如何清除中断标志位？
14. 系统上电时，AVR 的中断允许标志位、各个独立中断允许标志位和中断标志位的初始值是什么？全局中断允许标志位只是通过指令来置位或清除吗？那么各个独立中断允许标志位呢？
15. 在编写中断服务程序时，应该注意那些问题？有那些原则需要注意的？
16. AVR 中断响应的最短时间为多少？如何估计和计算中断服务的时间？
17. 全面分析总结中断服务程序和一般的子程序的相同点和不同点。
18. 可以在程序中使用语句直接调用中断服务程序吗？
19. 比较用汇编语言和 C 语言编写中断服务程序的优劣，在哪些情况下应考虑采用汇编（或嵌入部分汇编）编写中断服务程序？
20. AVR 的外部中断有几种触发方式？适合那些应用场合？
21. 使用低电平触发方式的外部中断时应该注意那些问题？
22. 在 AVR 中，如果要使 A 中断能够打断 B 中断而形成中断嵌套，而其它中断不允许中断嵌套，试描述 A 的中断服务程序应做怎样的处理。