Fremont Micro Devices

AN-22016

8bit MCU 通用

Application Note

Rev1.01

www.fremontmicro.com

文档修改历史

日期	版本	描述
2021-11-29	1.00	初版
2022-05-06	1.01	增加硬件电路注意事项

目录

表目录 / List of Figures

1 8 bit MCU 应用注意事项

1.1 硬件电路注意事项

由于硬件电路设计或其他开关器件的影响，导致 MCU 芯片引脚上出现异常电压脉冲时，为避免芯片损伤，建议如下：

1) 如果 IO 或 VDD 引脚上的负电压脉冲 ＜ - 2V，建议在 IO 或 GND 上串 5Ω 电阻；

2) 如果 IO 引脚上的正电压脉冲 ＞ (VDD + 2V)，建议在 IO 上串 5Ω 电阻；

3) 如果 VDD 引脚上有过冲电压 ＞ 9V，建议在 VDD 上串 40Ω 电阻；

如硬件电路没有上述异常脉冲，则无需串电阻；

1.2 中断注意事项

芯片的中断向量地址为 04H。

程序运行过程中发生中断事件时，CPU 会将程序当前 PC 值存入堆栈，然后从 04H 地址开始执行中断程序，一直执行到 RETI 指令，取回堆栈中的 PC 值返回主程序继续执行。

中断保护现场和恢复现场程序用于保护程序进入中断时的现场信息，如累加器 W 和状态寄存器 STATUS 的值。如果在中断里改写了 PCLATH，则需要增加一个缓存寄存器来保护 PCLATH。如果在中断里没有改写 PCLATH 值，则可以不用保护。保护现场和恢复现场程序可以保证在中断发生前后的现场信息一致，中断返回后可以继续正确的执行主程序。

推荐的保护现场程序和恢复现场程序如下：

```
W_TMP       EQU     0x70    ;保护 W 值的缓存寄存器（0x70-0x7F 为公用区间）
S_TMP       EQU     0x71    ;保护 STATUS 值的缓存寄存器
ORG         0004H           ;伪指令，定义中断程序入口地址
STR         W_TMP           ;进入中断程序，开始保护现场，将 W 值存入缓存
SWAPR       STATUS,W        ;将 STATUS 高四位和低四位互换后存入 W
STR         S_TMP           ;将 W 值存入 S_TMP 缓存
BANKSEL     PORTA           ;设置 BANK 为 PORTA 所在的 BANK0
;中断处理程序               ;省略的中断处理程序
;中断处理程序               ;省略的中断处理程序
INT_RET:                    ;中断返回标号，开始恢复现场
    SWAPR   S_TMP,W         ;将 S_TMP 缓存高四位和低四位互换后存入 W
```

```
STR     STATUS      ;将 W 值存入 STATUS，恢复 STATUS 值
SWAPR   W_TMP,F     ;将 W_TMP 高四位和低四位互换
SWAPR   W_TMP,W     ;将 W_TMP 高四位和低四位互换后存入 W
RETI                ;中断返回
```

W_TMP 和 S_TMP 需要定义在 0x70-0x7F 这个所有 BANK 共用的地址区间，以防在其他 BANK 时进入中断会丢失中断信息。

中断保护现场后需要设置 BANK。

如果中断处理程序里要处理 BANK0 里的寄存器，那么需要将 BANK 设置在 BANK0。

如果中断处理程序里要处理 BANK1 里的寄存器，那么需要将 BANK 设置在 BANK1。

如果进入中断后不设置 BANK，那么程序在 BANK1 进入中断后，直接操作到 BANK0 里的寄存器时就会产生错误。同理程序在 BANK0 进入中断时，操作到 BANK1 里的寄存器也会产生错误。

以上为使用汇编程序编程时需要注意事项，用 C 语言编程不需要注意。

C 编译器会自动设置中断保护现场和恢复现场程序。

1.2 按键唤醒功能

应用需要省电时，通常会采用睡眠加按键唤醒的方式去处理。按键唤醒功能需要此 IO 具有电平变化中断功能，FMD 大部分芯片的电平变化中断功能在 PORTA 端口，也有部分芯片为下降沿变化中断（如 FT61F04X 系列的 PORTA），需要注意电平状态。

推荐的电平变化中断唤醒的程序如下：

```
GIE=0;                  //关闭总中断
ReadAPin = PORTA;       //读端口状态存入缓存，设置电平变化中断匹配初值
PAIF =0;                //清电平变化中断标志
IOCA2 =1;               //使能 PA2 的电平变化中断功能
PAIE =1;                //使能电平变化中断
NOP();                  //建议加 1 到 2 条 NOP 指令
SLEEP();                //进入睡眠
NOP();                  //睡眠后需加 1 条 NOP 指令
ReadAPin = PORTA;       //读端口状态存入缓存，消除电平变化中断匹配条件
```

```
PAIF =0;                    //清电平变化中断标志

IOCA2 =0;                   //关闭 PA2 的电平变化中断功能

PAIE =0;                    //关闭电平变化中断

GIE=1;                      //打开总中断
```

电平变化中断唤醒需要注意以下几点：

1）此按键 IO 的状态需要在初始化里设置好。例如设置为输入口并开启内部上拉电阻功能，IO 通过按键串联电阻连接到地。这样无按键时端口的电平状态固定在高电平，按键按下时端口电平由高变低，触发电平变化中断。

2）从读端口状态到进入睡眠，中间的指令越少越好。

3）睡眠前关闭总中断 GIE 的动作，会让芯片在唤醒后继续执行 SLEEP 后的程序。

4）如果睡眠前没有关闭总中断 GIE，那么唤醒后会去跳转到中断入口地址 04H 去执行中断程序，中断返回后继续执行 SLEEP 后的程序。因此需要在中断程序里加入读端口状态和清电平变化中断标志的动作，以免因为端口电平变化中断的条件一直匹配而导致程序反复进入电平变化中断而不执行后面的程序。

推荐例程如下：

```
void interrupt ISR(void)        //中断函数
{
    if(PAIE && PAIF)            //PA 电平变化中断
    {
        ReadAPin = PORTA;  //读端口状态存入缓存，消除电平变化中断匹配条件
        PAIF = 0;               //清电平变化中断标志
    }
}
```

1.3 EEPROM 注意事项

1）在使用 EEPROM 的程序时，为防止上电时电压不稳定引起读写 EEPROM 不稳定，程序上需要增加约 100ms 延时后再进行 EEPROM 操作。如果编译选项里使能了 PWRT（上电延时）则程序上只需增加约 40ms 延时再进行 EEPROM 读写操作。

2）写入 EEPROM 数据过程需要按照以下例程步骤进行：

```
void EEPROMwrite(unchar EEAddr,unchar Data)
{
        while(GIE)              //步骤 A:写数据必须关闭中断
        { GIE = 0; }            //步骤 B:等待 GIE 为 0
        EEADR = EEAddr;         //步骤 C:写入 EEPROM 的地址
        EEDAT = Data;           //步骤 D:写入待写 EEPROM 的数据
        EECON1 |= 0x34;         //步骤 E:置位 WREN1,WREN2,WREN3 三个标志.
        WR = 1;                 //步骤 F:置位 WR 启动硬件 EEPROM 编程
        NOP();                  //注意:芯片运行在 1T 时，这里要加 NOP()
        while(WR);              //步骤 G:等待 EEPROM 写入完成（过程约 2ms）
        GIE = 1;                //步骤 H:写入完成，开启中断
}
```

如果软件启用了看门狗，那么在进入 EEPROM 写程序后需要清看门狗。

如果部分应用在中断里还有其他功能要实现（如模拟 PWM 输出），则 EEPROM 写入过程的 2ms 时间内不能关闭中断，可以将步骤 H 与步骤 G 位置互换。

3）写入 EEPROM 数据完成后，应将 EEPROM 实际值和要写入的目标值进行比较核对。如果相等则说明写入成功，否则写入失败，程序可采取重写操作。

```
EEPROMwrite(0x00,0xCC);          //往 EEPOM 的 00H 地址写入 0xCC
EEReadData = EEPROMread(0x00);//读取 0x00 地址 EEPROM 值
if(EEReadData!=0xCC)             //判断实际值是否等于 0xCC
{
    EEPROMwrite(0x00,0xCC);      //重新往 EEPOM 的 00H 地址写入 0xCC
}
```

4）如果在 EEPROM 的写入过程中发生了外部复位、WDT 溢出复位、LVR 复位或者非法指令引起的复位，那么标志位 EECON1.WRERR 会被置 1。上电初始化时可通过读取此标志位来判断前一次的 EEPROM 写入过程中是否发生异常情况，进而采取相应处理措施。

1.4 睡眠低功耗

在部分电池应用或需要低功耗的应用上，芯片必须使用睡眠来降低整机功耗。

FMD 芯片睡眠电流最低可以做到 3uA 以内，最低能到 0.3uA。

进入睡眠时，需要注意以下几方面：

1）芯片没有用到的功能外设尽量关闭。

2）LVD/LVR 功能启动后会有约 15uA 的电流，在睡眠时可根据实际情况来选择关闭 LVD/LVR 功能，睡眠唤醒后再打开。

3）普通 IO 在睡眠时应根据应用需要来设置。如果是普通输出口，则应设置成省电的输出。如果是普通输入口，则应使输入口保持固定的电平，不能处于浮空的状态。

4）编译选项里 FOSC 是否选为 INTOSC。如果选择 INTOSC，会导致芯片的 CLKO 脚位输出频率为 Fosc/4 的方波，影响睡眠电流。

5）部分芯片因为脚位没有全部封装出来，也需要将设置这些 IO，避免处于浮空输入的状态。

6）部分系列芯片的复位脚和 IO 口复用，且作为输入口时无内部上下拉电阻。需要特别注意如果此 IO 未使用到，需要将编译选项里 MCLRE 设置成 MCLR 功能，避免此 IO 处于浮空输入的状态。如 FT60F01X 的 PA3 和 FT60F02X 的 PA5。

7）如果应用特殊，关闭看门狗功能可以减少约 3uA 的电流。

1.5 内部上拉/下拉电阻功能

内部上拉电阻/下拉电阻只在 IO 处于数字输入口时生效。

1）先将 IO 口其他模拟功能先关闭（部分芯片默认是模拟口）。

2）设置相应的 IO 为输入端口。

3）打开相应的上拉电阻/下拉电阻的控制开关。

4）部分芯片的 PORTA 口还需设置 OPTION.7=0(/PUPA)来使能 PORTA 的上拉电阻。

1.6 PORTA 口操作注意事项

PORTA 是一个 8 位双向端口。与其相应的进出方向寄存器就是 TRISA 寄存器。

反之，将某一位设置为 0 会将该对应 PORTA 端口设置为输出端口。

在置为输出端口时，输出驱动电路会被打开，输出寄存器里的数据会被放置到输出端口。

当 IO 处于输入状态时，对 PORTA 进行读动作， PORTA 内容会是反映输入端口的状态。

在 PORTA 上进行写动作时， PORTA 内容会被写入输出寄存器。

所有的写操作都是“读-更改-写”的流程，即数据被读，然后更改，再写入输出寄存器的过程。

此特性在操作部分 LED 数码管时需要特别注意。

在部分应用中要视情况设置编译选项里的 RDCTRL 里为 LATCH，在输出状态下读 PORTA 返回的值为 PORTA 的缓存值，而非 PAD 上的真实电平值。

1.7 使用外部晶振注意事项

使用外部晶振时，MCU 晶振相邻脚输入输出高频脉冲时，会干扰晶振，引起晶振抖动，从而影响时序，造成计时偏差。

具体相邻脚位结合封装脚位图看：例如 FT61F145-TRB, PC0 就是晶振邻脚，使用外部晶振时，PC0 不要用做高频信号输入输出 IO 口，具体看下图：

型号	邻脚	晶振脚	邻脚	起振电容	相同型号
FT60F02X	PA1	PA7,PA6	VDD	15pF	
FT61F02X	VDD	PA7,PA6	PA5	15pF	
FT61F04X	NC	PA6,PA7	PC5	15pF	
FT60F22X	PA5	PA6,PA7	PB5	15pF	
FT60F12X	VDD	PA7,PA6	PA5	30pF	FT60F11X
FT61F13X	GND	PC1,PC0	PB7	15pF	FT62F13X
FT61F08X	PC0	PC1,PB7	GND	30pF	FT62F18X
FT61F14X	PC0	PC1,PB7	GND	30pF	
FT61F0AX	PC0	PC1,PB7	GND	30pF	FT64F0AX

表 1-1 不同型号外部晶振相邻脚

1.8 使用通信接口注意事项

1) I2C

外挂 FT24C02： SDA，SCL 设为开漏输出，需要外部上拉

2) SPI

外挂 FT25C64 接 4 根线，SO,SI,SCL 设为开漏输出，需要外部上拉，CS 用普通 IO 驱动

3) USART

TX 的方向寄存器要设为输出，RX 要设为输入

1.9 PWM 应用注意事项

虽然周期和占空比的双缓冲在很大程度上保证 PWM 输出不会产生毛刺，但如果软件非常接近 TIM 匹配时刻去写周期和占空比寄存器，特别是在 TIM 时钟频率比系统时钟快的情况下，则有可能出

现不可预料的情况，导致寄存器不是期望值，所以强烈建议更新周期和占空比寄存器，只在 TIM 匹配中断里面做。

1.10 汇编语言编程注意事项

1) 伪指令 BANKSEL

伪指令 BANKSEL 用于设置寄存器的 BANK，作用是设置 BANK 为 BANKSEL 后面的寄存器所在的 BANK。

如 FT60F01X 和 FT60F02X：

BANKSEL PORTA; 因为 PORTA 在 BANK0，所以指令等同于 BCR STATUS,5

BANKSEL TRISA; 因为 TRISA 在 BANK1，所以指令等同于 BSR STATUS,5

在 FT61F02X 中，寄存器 BANK 共有 3 个：BANK0，BANK1，BANK2，需要通过 STATUS 的 BIT5，BIT6 去设置，所以一条 BANKSEL 等同于两条指令。

BANKSEL PORTA，等同于：

BCR STATUS,5

BCR STATUS,6

BANKSEL TRISA，等同于：

BSR STATUS,5

BCR STATUS,6

这在需要精确计算指令数时要特别注意。

2) 汇编指令周期

汇编指令周期指执行一条汇编指令所需要的时间。

按照不同系列的芯片，指令周期的时间会有所不同。

目前我们有 1T、2T、4T 的指令周期。

1T 的指令周期指的是执行一条汇编指令周期需要 1 个系统时钟周期。

2T 的指令周期指的是执行一条汇编指令周期需要 2 个系统时钟周期。

4T 的指令周期指的是执行一条汇编指令周期需要 4 个系统时钟周期。

例如：FT60F01X 为 4T 指令周期。

选择 OSCCON.IRCF[2-0]=111 时，内部振荡频率 Fosc=16M。系统时钟周期为 1/16M 秒。

此时一个指令周期即为 4*1/16M=1/4M 秒=0.25us。即执行一个 NOP 需要耗时 0.25us。

FT60F02X 为 2T/4T 可选指令周期。可以在编译选项的 Tsel 选择为 2T 指令周期。

选择 OSCCON.IRCF[2-0]=111 时，内部振荡频率 Fosc=16M。系统时钟周期为 1/16M 秒。

此时一个指令周期即为 2*1/16M=1/8M 秒=0.125us。即执行一个 NOP 需要耗时 0.125us。

FMD 汇编指令大部分都是单周期指令，即一条指令占用一个指令周期。

部分有跳转 PCL 功能的指令为双周期指令，如 LJUMP、LCALL、RET、RETI、RETW。

部分判断跳转指令在满足跳转条件时为双指令周期，不满足跳转条件时为单指令周期。如 BTSS、BTSC、INCRSZ、DECRSZ。

3) 汇编指令功能说明

助记符	影响标志	周期	功能说明
BCR R,b	NONE	1	寄存器 R 的第 b 位清 0
BSR R,b	NONE	1	寄存器 R 的第 b 位置 1
BTSC R,b	NONE	1 或 2	如果寄存器 R 的第 b 位为 0 则跳过下一条指令，发生跳转时为 2 周期
BTSS R,b	NONE	1 或 2	如果寄存器 R 的第 b 位为 1 则跳过下一条指令，发生跳转时为 2 周期
NOP	NONE	1	空指令
CLRWDT	/PF, /TF	1	清看门狗
SLEEP	/PF, /TF	1	睡眠指令
STR R	NONE	1	将累加器 W 的值存入寄存器 R
LDR R,d	Z	1	将寄存器 R 的值导入累加器 W(d=W/0)或寄存器 R(d=F/1)
SWAPR R,d	NONE	1	将寄存器 R 的高四位和低四位交换后存入累加器 W(d=W/0)或寄存器 R(d=F/1)
INCR R,d	Z	1	寄存器 R 的值加 1,结果存入累加器 W(d=W/0)或寄存器 R(d=F/1)
INCRSZ R,d	NONE	1 或 2	寄存器 R 的值加 1,结果存入累加器 W(d=W/0)或寄存器 R(d=F/1) 如果结果等于 0，则跳过下一条指令，发生跳转时为 2 周期
ADDWR R,d	C, HC, Z	1	将累加器 W 的值和寄存器 R 的值相加，结果存入累加器 W(d=W/0)或寄存器 R(d=F/1) 如果结果大于 255，则 C=1
SUBWR R,d	C, HC, Z	1	将寄存器 R 的值减去累加器 W 的值，结果存入累加器 W(d=W/0)或寄存器 R(d=F/1) 如果 R>=W，则 C=1
DECR R,d	Z	1	寄存器 R 的值减 1 结果存入累加器 W(d=W/0)或寄存器 R(d=F/1)
DECRSZ R,d	NONE	1 或 2	寄存器 R 的值减 1,结果存入累加器 W(d=W/0)或寄存器 R(d=F/1) 如果结果等于 0，则跳过下一条指令，发生跳转时为 2 周期

ANDWR R,d	Z	1	寄存器 R 的值和累加器 W 的值相与,结果存入累加器 W(d=W/0)或寄存器 R(d=F/1)
IORWR R,d	Z	1	寄存器 R 的值和累加器 W 的值相或,结果存入累加器 W(d=W/0)或寄存器 R(d=F/1)
XORWR R,d	Z	1	寄存器 R 的值和累加器 W 的值异或,结果存入累加器 W(d=W/0)或寄存器 R(d=F/1)
COMR R,d	Z	1	寄存器 R 的值取反,结果存入累加器 W(d=W/0)或寄存器 R(d=F/1)
RRR R,d	C	1	寄存器 R 的值带 C 位右移，结果存入累加器 W(d=W/0)或寄存器 R(d=F/1)
RLR R,d	C	1	寄存器 R 的值带 C 位左移，结果存入累加器 W(d=W/0)或寄存器 R(d=F/1)
CLRW	Z	1	清累加器 W
CLRR R	Z	1	清寄存器 R
RETI	NONE	2	中断返回
RET	NONE	2	子程序返回
LCALL N	NONE	2	调用子程序 N
LJUMP N	NONE	2	跳转到标号 N
LDWI I	NONE	1	将立即数 I 导入累加器 W
ANDWI I	Z	1	累加器 W 的值和立即数 I 相与，结果存入累加器 W
IORWI I	Z	1	累加器 W 的值和立即数 I 相或，结果存入累加器 W
XORWI I	Z	1	累加器 W 的值和立即数 I 异或，结果存入累加器 W
RETW I	NONE	2	子程序返回，并将立即数 I 存入累加器 W
ADDWI I	C, HC, Z	1	累加器 W 的值和立即数 I 相加，结果存入累加器 W，如果结果大于 255，则 C=1
SUBWI I	C, HC, Z	1	累加器 W 的值和立即数 I 相减，结果存入累加器 W，如果 I>=W，则 C=1

表 1-2 汇编指令集

4) 汇编查表

在汇编指令中处理查表功能时需要特别注意 PCLATH 的处理。

汇编里通常通过 RETW XXX 来返回表格数据，常用的数码管查表程序例程如下：

```
LDR     DIG_2,W         ;将第二位赋值给 W
LCALL   GET_CODE        ;查表
STR     TEMP0           ;数据存入 TEMP0
ORG     0X300           ;定义表格程序地址到 300H 地址
GET_CODE:
```

```
LDWI    0X03
STR     PCLATH      ;设置 PCLATH=3（跟随表格实际地址）
LDR     BCD,W       ;查表
ADDWR   PCL,F       ;--GFEDCBA
RETW    B'00111111' ;0
RETW    B'00000110' ;1
RETW    B'01011011' ;2
RETW    B'01001111' ;3
RETW    B'01100110' ;4
RETW    B'01101101' ;5
RETW    B'01111101' ;6
RETW    B'00000111' ;7
RETW    B'01111111' ;8
RETW    B'01101111' ;9
RETW    B'00000000' ;OFF
```

5) LJUMP 和 LCALL 指令

因为 PC 宽度限制的问题，LJUMP 和 LCALL 最多只能访问到 2KROM 空间。

程序空间在 2K 内的芯片使用 LJUMP 和 LCALL 指令时无需考虑跳转出错的问题。

2K 容量以上的芯片在使用 LJUMP 和 LCALL 访问 2K 以上的空间时，需要设置 PCLATH。

超过 4K 程序空间的芯片，建议使用 C 语言编程，编译器会自动处理。

6) 标号的使用

在汇编里设置子程序标号或者位置标号时，需要注意在标号增加英文冒号：

不加冒号会被编译器识别成宏定义，如果头文件里没有这个宏定义就会引起编译器报错。

1.11 C 语言编程注意事项

1) 包含头文件

在程序起始位置必须用#include 指令包含编译器提供的“syscfg.h”文件，以实现特殊功能寄存器和其它特殊符号的声明。不用单独包含芯片型号的头文件，IDE 会自动调用。

其他子程序文件和头文件也需要在 Main 文件里用#include 指令包含。

2) 变量类型

常用变量的类型如下表：需要注意的是 char 默认为无符号字符变量。

类型	长度（位）	数学表达
bit	1	布尔型变量
char	8	字符变量，默认为无符号字符变量
signed char	8	有符号字符变量
unsigned char	8	无符号字符变量
int	16	整型变量，默认为有符号整型变量
signed int	16	有符号整型变量
unsigned int	16	无符号整型变量
long	32	长整型变量，默认为有符号长整型变量
signed long	32	有符号长整型变量
unsigned long	32	无符号长整型变量
float	24	浮点数
double	24	双精度浮点数

表 1-3 C 语言变量类型

3) 变量声明

const：常数型变量

如果在变量定义时加前缀 const 声明，那么此变量就会成为常数型变量，程序运行过程中不能对其修改，通常用于常量数组的定义申明。

volatile：常驻内存型变量

C 编译器会对程序进行优化，部分在程序里无实际作用的变量会被优化处理，在调试过程中就无法观察变量实际值。如果希望变量不被优化，需要在此变量定义时加 volatile 的前缀进行声明。

persistent：非初始化变量

在程序上电运行时会将所有定义的变量进行清零或设置初始值。但在许多实际应用中不运行程序对部分变量进行初始化，此时可以在此变量定义前加入 persistent 前缀声明。

4) 绝对地址变量

部分应用需要将 8 个位变量放在同一个字节中进行批量操作，可以通过两种方式：

a. 通过定义一个位域结构和一个字节变量联合来实现：

union {

```
        struct {
            unsigned bit0:    1;
            unsigned bit1:    1;
            unsigned bit2:    1;
            unsigned bit3:    1;
            unsigned bit4:    1;
            unsigned bit5:    1;
            unsigned bit6:    1;
            unsigned bit7:    1;
        } oneBit;
        unsigned char allBits;
} TESTFlag;
```

设置其中某一位时操作如下：　TESTFlag.oneBit.bit5=1;　//bit5 位置 1

位变量批量清零时操作如下：　TESTFlag.allBits=0;　//位变量批量清 0

b. 通过定义变量的绝对地址的方式来实现

```
volatile unsigned charTESTFlag @ 0x20; // TESTFlag 定义在地址 0x20
bit TESTBit0 @((unsigned)&TESTFlag*8)+0;//TESTBit0 对应于 TESTFlag 第 0 位
bit TESTBit1 @((unsigned)&TESTFlag*8)+1;//TESTBit1 对应于 TESTFlag 第 1 位
bit TESTBit2 @((unsigned)&TESTFlag*8)+2;//TESTBit2 对应于 TESTFlag 第 2 位
bit TESTBit3 @((unsigned)&TESTFlag*8)+3;//TESTBit3 对应于 TESTFlag 第 3 位
bit TESTBit4 @((unsigned)&TESTFlag*8)+4;//TESTBit4 对应于 TESTFlag 第 4 位
bit TESTBit5 @((unsigned)&TESTFlag*8)+5;//TESTBit5 对应于 TESTFlag 第 5 位
bit TESTBit6 @((unsigned)&TESTFlag*8)+6;//TESTBit6 对应于 TESTFlag 第 6 位
bit TESTBit7 @((unsigned)&TESTFlag*8)+7;//TESTBit7 对应于 TESTFlag 第 7 位
```

需要注意的是：编译器对绝对定位的变量不保留地址空间。

上面例程里变量 TESTFlag 的地址是 0x20，但最后 0x20 地址有可能又被编译器分配给了其它变量使用，这样就发生了地址冲突。因此用户自定义变量的绝对地址时需要在定义变量时加上前缀 volatile 声明，防止编译器自动优化分配地址。

这样的用法会降低 RAM 的使用效率，所以在一般的程序设计中不建议定义变量的绝对地址。

联系信息

Fremont Micro Devices Corporation

#5−8, 10/F, Changhong Building
Ke-Ji Nan 12 Road, Nanshan District,
Shenzhen, Guangdong, PRC 518057

Tel: (+86 755) 8611 7811
Fax: (+86 755) 8611 7810

Fremont Micro Devices (HK) Limited

#16, 16/F, Block B, Veristrong Industrial Centre,
34−36 Au Pui Wan Street, Fotan, Shatin, Hong Kong SAR

Tel: (+852) 2781 1186
Fax: (+852) 2781 1144

http://www.fremontmicro.com