SN8P2604

用户手册

SONiX 8 位单片机

修改记录

版本	日期	说明
VER 0.1	2004 年 1 月	第一版
VER 0.2	2004 年 4 月	1. 增加应用注意事项章节。 2. 调整了特性的说明。 3. 调整了“SN8P1604A 移植到 SN8P2604”表。 4. 在指令表中调整了 PUSH/POP 的说明。 5. 在 SSOP28 封装型号中调整了外围尺寸。 6. 更改了唤醒时间的公式。
VER 0.3	2004 年 7 月	1. ORG 8 处的第一条指令必须是“JMP”或“NOP”。 2. 把寄存器 Y，Z，H，L，R 的复位值由 00H 改为未知。 3. 操作环境温度中封装型号的后面插入“D”表示温度范围为-40℃～+80℃。 4. 调整了部分电气特性：ViL，ViH，Ilekg，Rup，IoH。 5. 在“应用注意事项”中增加“开发工具版本”。 6. 把“转接座”的名字改为“OTP 烧录引脚”并调整了部分内容。 7. 删除了“编程模板”。
VER0.4	2004 年 7 月	更改了指令周期中参数“S”的有关说明。
VER0.5	2004 年 7 月	1. 删除了 TC0 定时/计数器的说明并更改了烧录信息。 2. 更改了绿色模式的说明。

目　录

1 产品简介

1.1 特性

◆ **存储器配置**
OTP ROM：4K * 16 bits.
RAM：128 * 8 bits.

◆ **8 层堆栈缓存器**

◆ **I/O 引脚配置**
双向输入输出：P0, P1, P2, P5
漏极开路引脚：P1.0, P1.1
具有唤醒功能的引脚：P0, P1 的电平变换
内部上拉电阻：P0, P1, P2, P5
外部中断输入：P0.0，P0.1
外部中断：P0.0 由 PEDGE 控制，P0.1 由下降沿触发。

◆ **功能强大的指令集**
每个指令周期为一个时钟周期（1T）
大多数指令的执行时间均为一个指令周期
JMP 指令可直接寻址整个 ROM 区
CALL 指令可直接寻址整个 ROM 区
查表功能（MOVC）可直接寻址整个 ROM 区

◆ **4 个中断源**
2 个内部中断源：T0, TC1.
2 个外部中断源：INT0, INT1.

◆ **2 个 8 位定时/计数器**
T0：基本定时器
TC1：自动装载定时器/PWM1/Buzzer 输出

◆ **内置看门狗定时器**

◆ **系统时钟和操作模式**
外部高速时钟：RC，最大 10 MHz
外部高速时钟：晶体，最大 16 MHz
内部低速时钟：RC，16KHz(3V)，32KHz(5V)
普通模式：高低速时钟同时运行
低速模式：仅低速时钟运行
睡眠模式：高低速时钟均停止
绿色模式：由定时器 T0 周期性唤醒

◆ **封装(支持的芯片格式)**
SK-DIP 28
SOP 28
SSOP 28

☞ **特性比较表**

芯片型号	ROM	RAM	堆栈	定时器		I/O	绿色模式	PWM	唤醒功能引脚	封装
				T0	TC1			Buzzer		
SN8P1604A	4K*16	128	8		V	22	-	V	10	SK-DIP28/SOP28
SN8P2604	4K*16	128	8	V	V	24	V	V	11	SK-DIP28/SOP28/SSOP28

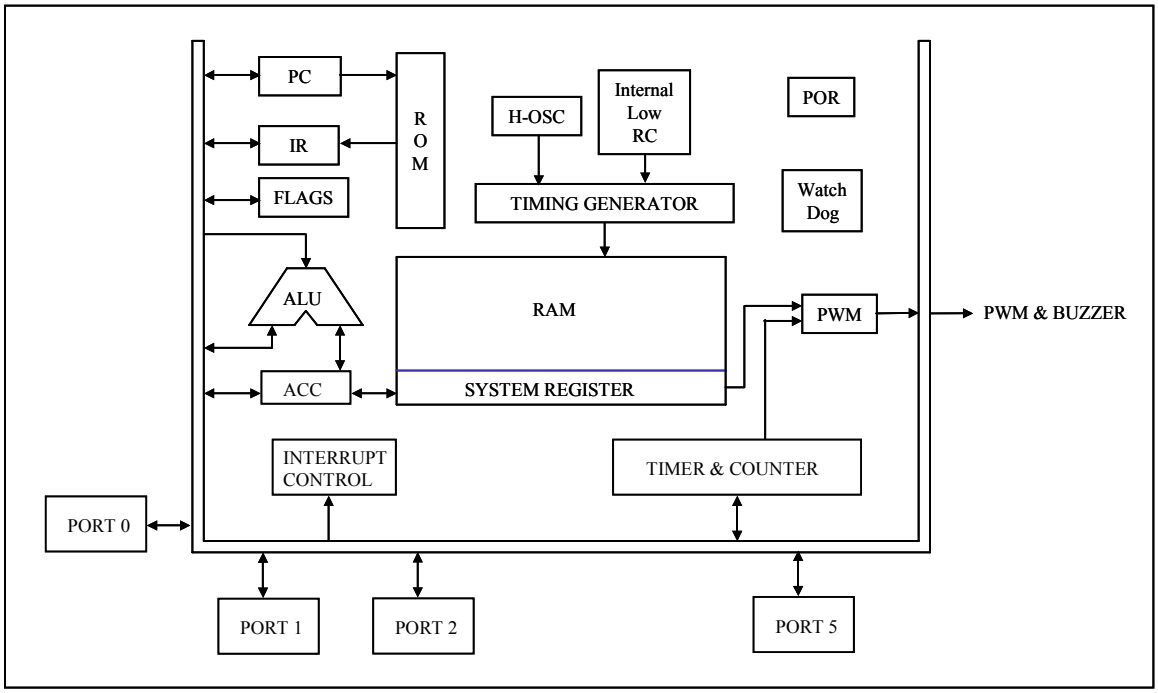
☞ **SN8P1604A 升级到 SN8P2604**

项目	SN8P2604	SN8P1604A
DAA	不可用	可用
PUSH/POP	可用	不可用
B0MOV M, I	I 不能是 0E6h 和 0E7h	-
B0XCH A, M	M 的地址不能是 80h~FFh	-
R0M 的 ORG8 处的有效指令	JMP 或 NOP	无限制
AC 抗干扰能力	很好（**添加了 1 个 47uF 的旁路电容**）	一般
运算能力(16Mhz Crystal)	最大 16 MIPS	最大 4 MIPS
高速 PWM	PWM 分辨率：8/6/5/4 位 High Clock = 16MHz 8 位分辨率时高达 31.25K 4 位分辨率时高达 500K	PWM 分辨率：只 8 位 在 16MHz 等于 7.8125K
定时器	T0/TC1	TC1
I/O 引脚的数目	24 (P0.2)	22
可编程漏极开路输出	P1.0/P1.1	-
内置上拉电阻	引脚配置（利用寄存器 PnUR）	端口配置（利用寄存器 PUR）
绿色模式	有（T0 定时器）	-
P0.0 中断触发沿	由寄存器 PEDGE 控制： P00G[1:0]：（位 4 和位 3） 00=保留，01=上升沿， 10=下降沿，11=改变电平	由寄存器 PEDGE 控制： PEDGEN(位 7)： 0=下降沿，1=由 P00G[1:0]定义 P00G[1:0]：(位 4 和位 3) 00=保留，01=下降沿 10=上升沿，11=改变电平
P0.1 中断触发边沿	下降沿	P0.1 无中断功能
P0.0 唤醒功能	电平变化	由寄存器 PEDGE 控制： PEDGEN(位 7)： 0=下降沿，1=由 P00G[1:0]定义 P00G[1:0]：(位 4 和位 3) 00=保留，01=下降沿 10=上升沿，11=改变电平
P0.1 唤醒功能	电平变化	由寄存器 PEDGE 控制： PEDGEN(位 7)： 0=低电平唤醒，1=改变电平
P1 唤醒功能	改变电平	由寄存器 PEDGE 控制： PEDGEN(位 7)： 0=低电平唤醒，1=改变电平
唤醒时间	1/Fosc * 4096 (sec) + X'tal 稳定时间	1/Fosc * 2048 (sec) + X'tal 稳定时间
施密特触发输入	所有输入引脚	P0.0，P0.1，RST，XIN
看门狗定时器时钟源	仅内部低速 RC 时钟	内部低速 RC 时钟和外部高速时钟
看门狗清零	MOV A, #0x5A B0MOV WDTR, A	B0BSET FWDRST
待机电流	1uA/5V	10uA/5V
LVD	1.8V，一直处于使能状态	1.8V，一直处于使能状态

注：电平变化触发指上升沿和下降沿触发。

1.2 系统时钟框图

SN8P2604



1.3 引脚配置

SN8P2604K (SK-DIP 28 pins)
SN8P2604S (SOP 28 pins)
SN8P2604X (SSOP 28 pins)

P0.1/INT1	1	U	28	RST/VPP/P0.2
VDD	2		27	XIN
P5.4	3		26	XOUT/Fcpu
VSS	4		25	P2.7
P0.0/INT0	5		24	P2.6
P5.0	6		23	P2.5
P5.1	7		22	P2.4
P5.2	8		21	P2.3
P5.3/BZ1/PWM1	9		20	P2.2
P1.0	10		19	P2.1
P1.1	11		18	P2.0
P1.2	12		17	P1.7
P1.3	13		16	P1.6
P1.4	14		15	P1.5

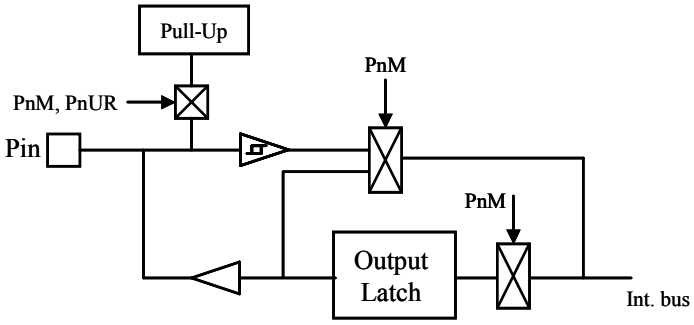
SN8P2604K
SN8P2604S
SN8P2604X

1.4 引脚说明

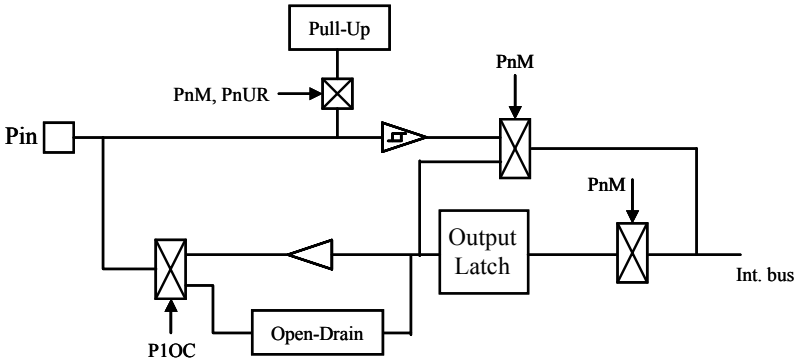
引脚名称	类型	说明
VDD, VSS	P	数字电路的电源输入引脚
P0.2/RST/VPP	I, P	P0.2：禁止外部复位功能时为单向输入引脚（施密特结构）/无内置上拉电阻。 RST：系统复位输入引脚，施密特结构，低电平触发，通常保持高电平。 VPP：OTP ROM 编程引脚。
XIN	I	选择外部振荡器(晶体或 RC)时为振荡器输入引脚。
XOUT/Fcpu	I/O	XOUT：选择外部晶体振荡器时为振荡器输出引脚 Fcpu：使能外部 RC 模式时为信号输出引脚。
P0.0/INT0	I/O	P0.0：双向输入输出引脚，施密特结构（输入模式下）。内置上拉电阻。 INT0 触发引脚（施密特结构）
P0.1/INT1	I/O	P0.1：双向输入输出引脚，施密特结构（输入模式下）。内置上拉电阻。 INT1 触发引脚（施密特结构） TC1 事件计数器时钟输入引脚。
P1.0~P1.1	I/O	P1.0, P1.1：双向输入输出引脚/漏极开路引脚/施密特结构（输入模式下）/内置上拉电阻。
P1.2~P1.7	I/O	P1.2~P1.7：双向输入输出引脚/施密特结构（输入模式下）/内置上拉电阻。
P2.0~P2.7	I/O	双向输入输出引脚/施密特结构（输入模式下）/内置上拉电阻。
P5.0~P5.2, P5.4	I/O	P5：双向输入输出引脚/施密特结构（输入模式下）/内置上拉电阻。
P5.3/BZ1/PWM1	I/O	P5.3：双向输入输出引脚/施密特结构（输入模式下）/内置上拉电阻。 TC1 /2 ：Buzzer 或 PWM 信号输出引脚。

1.5 引脚电路图

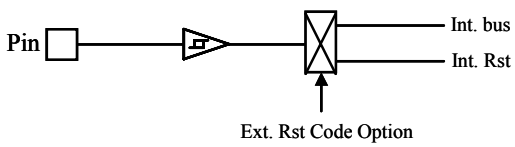
Port 0, 1, 2, 5 structure:



Port 1.0, P1.1 structure:



Port 0.2 structure:

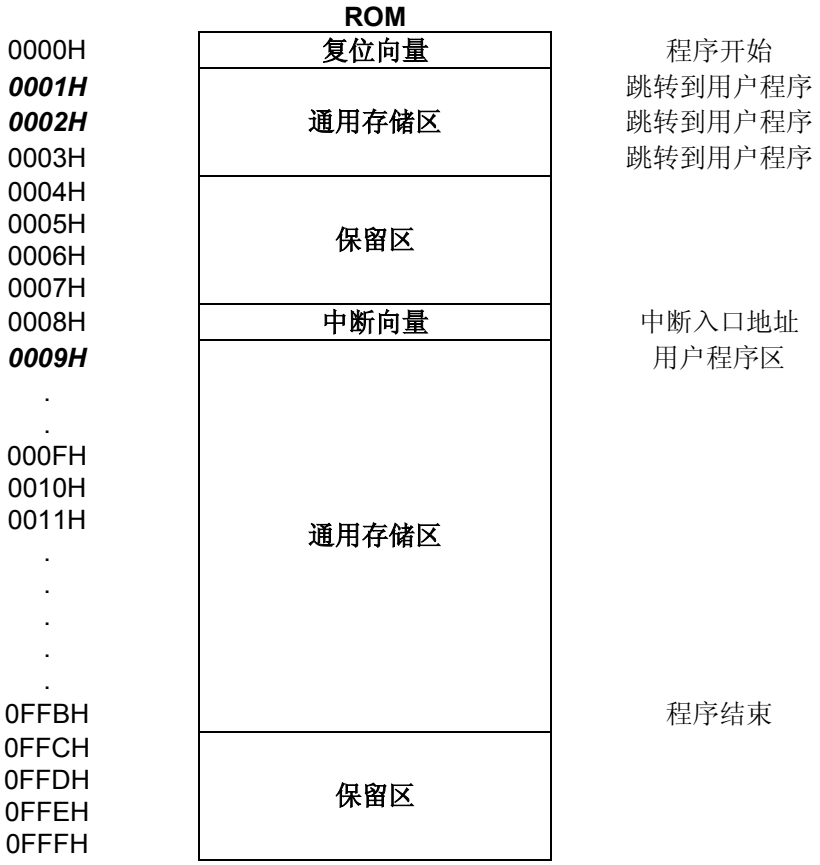


2 中央处理器(CPU)

2.1 内存

2.1.1 程序存储器(ROM)

☞ ***4K words ROM***



2.1.1.1 复位向量地址（000H）

one-word 的复位向量地址用来执行系统复位。

- ☞ **上电复位(NT0=1, NPD=0)**
- ☞ **看门狗复位(NT0=0, NPD=0)**
- ☞ **外部复位(NT0=1, NPD=1)**

上电复位或看门狗计数器溢出复位后，系统从地址 0000H 开始重新执行程序，所有的系统寄存器恢复为默认值。下面的例子给出了如何在程序存储器里定义复位向量。

➲ **例： 定义复位向量**

```
        ORG     0           ; 0000H
        JMP     START       ; 跳转到用户程序区
        .                   ; 0004H ~ 0007H 保留
        ORG     10H
START:                      ; 0010H,用户程序的起始位置
        .                   ; 用户程序
        .
        ENDP                ; 程序结束
```

2.1.1.2 中断向量地址（0008H）

一旦有中断响应，程序计数器（PC）的值就会存入堆栈缓冲器中并跳转至 0008H 处执行中断服务程序。用户使用时必须自行定义中断向量，在 ROM 的地址 8（ORG 8）处的指令必须为 **JMP** 或 **NOP**。下面的例子给出了如何在程序中定义中断向量。

✷ 注：“PUSH”，“POP”指令保存和恢复 ACC 和 PFLAG，除了 PFLAG 的复位标志（NT0，NPD）

✷ 注：ROM 地址 8 处的指令必须为“JMP”或“NOP”。

➲ 例 1：定义中断向量，主程序位于中断服务程序后面。

```
                ORG     0               ; 0000H
                JMP     START           ; 跳转到用户程序
                .                       ; 0004H ~ 0007H 保留

                ORG     8               ; 中断服务程序
                NOP
                B0XCH   A, ACCBUF       ; B0XCH 不会影响到 C, Z
                PUSH                    ; 保存工作寄存器
                .
                POP                     ; 恢复工作寄存器
                B0XCH   A, ACCBUF
                RETI                    ; 中断返回
START:                                  ; 用户程序起始地址
                .                       ; 用户程序
                .
                JMP     START           ; 用户程序结束

                ENDP                    ; 程序结束
```

➲ 例 2：定义中断向量，中断服务程序位于主程序后面。

```
                ORG     0               ; 0000H
                JMP     START           ; 跳转到用户程序.
                .                       ; 0004H ~ 0007H 保留

                ORG     08
                JMP     MY_IRQ          ; 0008H, 跳转到中断服务程序

                ORG     10H
START:                                  ; 0010H, 用户程序起始地址
                .                       ; 用户程序
                .
                JMP     START           ; 用户程序结束

MY_IRQ:                                 ;中断服务程序的起始地址
                B0XCH   A, ACCBUF       ; B0XCH 指令不会影响标志位 C，Z
                PUSH                    ; 保存工作寄存器
                .
                POP                     ; 恢复工作寄存器
                B0XCH   A, ACCBUF
                RETI                    ; 中断返回

                ENDP                    ; 程序结束
```

✷ 注：从上面的程序中很容易得出 SONIX 的主要编程规则，有以下几点：
1. 地址 0000H 处的“JMP”指令使程序从头开始执行。
2. 0004H~0007H 区域由系统保留，用户必须跳过 0004H~0007H。我们强烈建议用户在对 ROM 区作 CHECKSUM 时要跳过该区域，详见下面的 Checksum 计算章节。

2.1.1.3 查表功能

在 ROM 的查表功能程序中， Y 寄存器指向数据 ROM 地址的高 8 位，Z 寄存器指向低 8 位地址，执行 MOVC 后，数据的低字节存入累加器 ACC 中，而数据的高字节存入 R 寄存器中。

➲ **例：查找位于 “table1”的 ROM 数据.**

```
                B0MOV   Y, #TABLE1$M    ; 取得表格地址高字节
                B0MOV   Z, #TABLE1$L    ; 取得表格地址低字节
                MOVC                    ; 查表, R = 00H, ACC = 35H
                                        ; 索引地址加 1
                INCMS   Z               ; Z+1
                JMP     @F              ; 无进位
                INCMS   Y               ; Z 溢出 (FFH → 00) 则, → Y=Y+1
                NOP                     ;
                                        ;
@@:             MOVC                    ; 查表, R = 51H, ACC = 05H.
                .       .               ;
TABLE1:         DW      0035H           ; 定义表格数据
                DW      5105H           ; “
                DW      2012H           ; “
```

✷ **注意：当 Z 寄存器从 0XFFH 增至 0X00H 跨越边界时，Y 寄存器不会自动增加。所以，用户必须非常小心处理这种情况，避免查表错误。如果 Z 寄存器发生溢出，Y 寄存器必须增 1。下面给出的宏指令 INC_YZ 提供了解决此问题的方法。**

➲ **例：宏指令 INC_YZ**

```
INC_YZ          MACRO
                INCMS   Z               ; Z+1
                JMP     @F              ; Z 无溢出

                INCMS   Y               ; Y+1
                NOP                     ; Y 无溢出
@@:
                ENDM
```

➲ **例：通过宏指令“INC_YZ”调整上例。**

```
                B0MOV   Y, #TABLE1$M    ; 取得表格地址高字节
                B0MOV   Z, #TABLE1$L    ; 取得表格地址低字节
                MOVC                    ; 查表，R=00H，ACC=35H

                INC_YZ                  ; 索引地址加 1

@@:             MOVC                    ; 查表，R=51H，ACC=05H
                …
TABLE1:         DW      0035H           ; 定义一个字（16 位）的数据
                DW      5105H
                DW      2012H
                …
```

另一种方式的查表是通过累加器来增加间接寄存器 Y 和 Z，但要注意是否有进位发生。

➲ **例：执行指令 B0ADD/ADD 增加 Y、Z**

```
                B0MOV   Y, #TABLE1$M    ; 取得表格地址高字节
                B0MOV   Z, #TABLE1$L    ; 取得表格地址低字节

                B0MOV   A, BUF          ; Z = Z + BUF.
                B0ADD   Z, A

                B0BTS1  FC              ; 检查进位表示 C
                JMP     GETDATA         ; FC = 0
                INCMS   Y               ; FC = 1. Y+1.
                NOP
GETDATA:                                ;
                MOVC                    ; 查表，若 BUF = 0，结果是 0x0035
                                        ; 若 BUF = 1,结果是 0x5105
                                        ; 若 BUF = 2，结果是 0x2012
TABLE1:         DW      0035H           ; 定义一个 WORD 的表格数据
                DW      5105H           ; “
                DW      2012H           ; “
```

✷ **注：“B0MOV M, I”指令不支持“I=0xE6”和“I=0xE7”，在查表功能中，用户必须检查 Y，Z 的值，不能是“0xE6”和“0xE7”。在表格的开始设置 ROM 的地址，避开“0xE6”和“0xE7”。**

➲ **例：通过 ORG 指令设置 TABLE1 的开始地址以避开“0xE6”和“0xE7”。**

```
                B0MOV   Y, #TABLE1$M    ; 取得表格的高字节
                B0MOV   Z, #TABLE1$L    ; 取得表格的低字节

                B0MOV   A, BUF          ; Z = Z + BUF.
                B0ADD   Z, A

                B0BTS1  FC              ; 检查进位标志
                JMP     GETDATA         ; FC = 0
                INCMS   Y               ; FC = 1. Y+1.
                NOP
GETDATA:                                ;
                MOVC                    ; 查表，若 BUF = 0，数据为 0x0035
                                        ; 若 BUF = 1，数据为 0x5105
                                        ; 若 BUF = 2，数据为 0x2012
                …

                ORG     0x0100          ; 设置 TABLE1 的开始地址是 0x0100.

TABLE1:         DW      0035H           ; 定义一个 word（16 位）的表格数据
                DW      5105H           ; “
                DW      2012H           ; “
                …
```

2.1.1.4 跳转表

跳转表操作可以完成多个地址跳转功能，将程序计数器的低字节 PCL 与累加器 ACC 相加从而得到一个指向新的跳转地址的程序计数器值，这种方法可以方便多个任务的处理。

➲ **例：跳转表**

```
        ORG     0X0100          ; 跳转表最好放在 ROM 的边界位置

        B0ADD   PCL, A          ; PCL = PCL + ACC,但 PCH 不会改变
        JMP     A0POINT         ; ACC = 0, 跳转到 A0POINT
        JMP     A1POINT         ; ACC = 1, 跳转到 A1POINT
        JMP     A2POINT         ; ACC = 2, 跳转到 A2POINT
        JMP     A3POINT         ; ACC = 3, 跳转到 A3POINT
```

SONIX 提供了一条宏指令以保证安全的跳转表操作，这条宏指令会检查 ROM 的边界，并自动将跳转表移动到正确的位置。但宏指令会占用 ROM 的存储空间。

```
@JMP_A  MACRO   VAL
        IF      (($+1) !& 0XFF00) !!= (($+(VAL)) !& 0XFF00)
        JMP     ($ | 0XFF)
        ORG     ($ | 0XFF)
        ENDIF
        ADD     PCL, A
        ENDM
```

✷ **注：“VAL”为跳转列表的个数**

➲ **例：“@JMP_A” 在 SONIX 的宏文件中称为“MACRO3.H”.**

```
        B0MOV   A, BUF0         ; “BUF0” 的值为 0~4
        @JMP_A  5               ; 要跳转的总的地址数是 5.
        JMP     A0POINT         ; ACC = 0, 跳转到 A0POINT
        JMP     A1POINT         ; ACC = 1, 跳转到 A1POINT
        JMP     A2POINT         ; ACC = 2, 跳转到 A2POINT
        JMP     A3POINT         ; ACC = 3, 跳转到 A3POINT
        JMP     A4POINT         ; ACC = 4, 跳转到 A4POINT
```

如果跳转表格的位置是从 00FFH 到 0100H，那么宏指令“@JMP_A”将使跳转表格从 0100h 开始。

➲ **例：“@JMP_A”操作。**

; 程序编译前

```
ROM address
        B0MOV   A, BUF0         ; “BUF0” 的值为 0-4
        @JMP_A  5               ; 要跳转的总的地址数是 5.
0X00FD  JMP     A0POINT         ; ACC = 0, 跳转到 A0POINT
0X00FE  JMP     A1POINT         ; ACC = 1, 跳转到 A1POINT
0X00FF  JMP     A2POINT         ; ACC = 2, 跳转到 A2POINT
0X0100  JMP     A3POINT         ; ACC = 3, 跳转到 A3POINT
0X0101  JMP     A4POINT         ; ACC = 4, 跳转到 A4POINT
```

; 程序编译后

```
ROM address
        B0MOV   A, BUF0         ; “BUF0” 的值为 0-4
        @JMP_A  5               ; 要跳转的总的地址数是 5.
0X0100  JMP     A0POINT         ; ACC = 0, 跳转到 A0POINT
0X0101  JMP     A1POINT         ; ACC = 1, 跳转到 A1POINT
0X0102  JMP     A2POINT         ; ACC = 2, 跳转到 A2POINT
0X0103  JMP     A3POINT         ; ACC = 3, 跳转到 A3POINT
0X0104  JMP     A4POINT         ; ACC = 4, 跳转到 A4POINT
```

2.1.1.5 CHECKSUM 计算

ROM 中的 0004H~0007H 和最后的一个地址是系统保留区，用户应该在计算 Checksum 时跳过这一区域。

➲ **例：下面的程序给出了在计算 Checksum 时如何跳过保留区。**

```
                MOV     A,#END_USER_CODE$L
                B0MOV   END_ADDR1,A         ; 保存地址的低字节
                MOV     A,#END_USER_CODE$M
                B0MOV   END_ADDR2,A         ; 保存地址的高字节
                CLR     Y                   ; 清 Y 寄存器
                CLR     Z                   ; 清 Z 寄存器
@@:
                CALL    YZ_CHECK            ; 调用函数，判断是否到 0004H 保留区
                MOVC                        ;
                B0BSET  FC                  ;
                ADD     DATA1,A             ;
                MOV     A,R
                ADC     DATA2,A             ;
                JMP     END_CHECK           ; 跳转到判断代码结束的函数
AAA:
                INCMS   Z                   ; Z=Z+1
                JMP     @B                  ;
                JMP     Y_ADD_1             ;
END_CHECK:
                MOV     A,END_ADDR1
                CMPRS   A,Z                 ; 判断是否计算到代码结束位置
                JMP     AAA                 ;
                MOV     A,END_ADDR2
                CMPRS   A,Y                 ;
                JMP     AAA                 ; 不是则继续计算
                JMP     CHECKSUM_END        ; 是则结束计算
YZ_CHECK:                                   ; 检查是否到 0004H 位置
                MOV     A,#04H
                CMPRS   A,Z                 ;
                RET                         ; 不是 0004H 则返回继续计算
                MOV     A,#00H
                CMPRS   A,Y                 ;
                RET                         ;
                INCMS   Z                   ;
                INCMS   Z
                INCMS   Z
                INCMS   Z
                RET                         ;
Y_ADD_1:
                INCMS   Y                   ; 递增 Y，继续计算
                NOP
                JMP     @B                  ;
CHECKSUM_END:
                ……
                ……
END_USER_CODE:                              ;
```

2.1.2 编译选项表（Code Option）

编译选项	内容	功能说明
High_Clk	RC	外部振荡器采用廉价 RC 振荡电路，XOUT 为 Fcpu 输出引脚。（P1.2）
	32K X'tal	外部振荡器采用低功耗振荡器（如 32.768KHz）
	12M X'tal	外部振荡器采用高频晶体振荡器（如 12MHz）
	4M X'tal	外部振荡器采用一般晶体振荡器（如 4MHz）
Watch_Dog	Always_On	看门狗定时器在省电模式和绿色模式下仍然处于使能状态。
	Enable	使能看门狗定时器功能，看门狗定时器在省电模式和绿色模式下被禁止。
	Disable	禁止看门狗定时器功能
Fcpu	Fosc/1	指令周期为 1 个振荡器时钟。
	Fosc/2	指令周期为 2 个振荡器时钟。
	Fosc/4	指令周期为 4 个振荡器时钟。
	Fosc/8	指令周期为 8 个振荡器时钟。
Reset_Pin	Reset	使能外部复位引脚功能
	P02	P0.2 作为单向输入端，没有上拉电阻
Security	Enable	允许 ROM 程序代码加密
	Disable	禁止 ROM 程序代码加密
Noise_Filter	Enable	使能噪音滤除功能，Fcpu=Fosc/4~Fosc/8.
	Disable	禁止噪音滤除功能，Fcpu=Fosc/1~Fosc/8.

✷ 注：

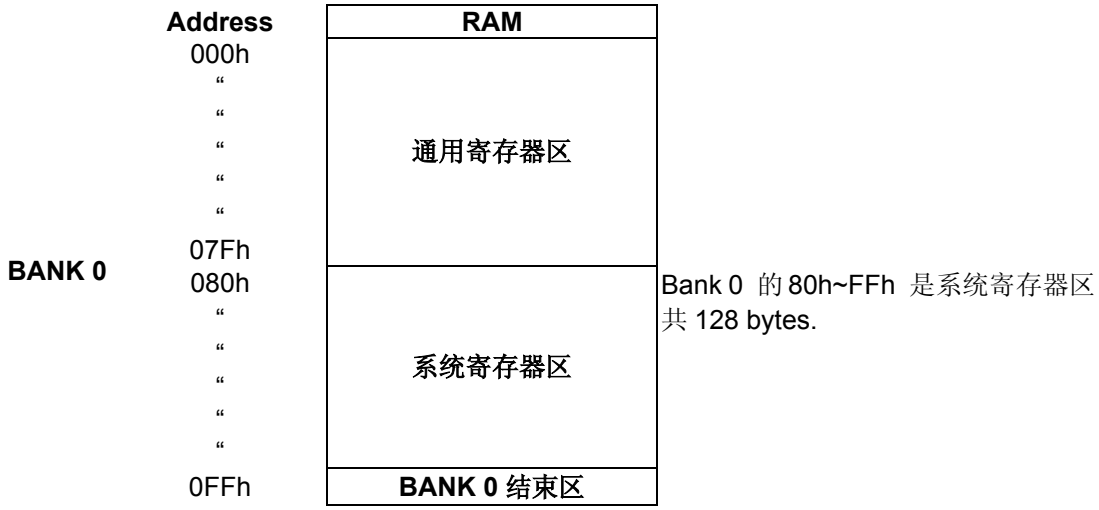
1.在高干扰环境下，强烈建议使能"Noise Filter"选项，并使看门狗选择"Always_On"选项。使能"Noise_Filter"后就会限制 Fcpu = Fosc/4 ~ Fosc/8.

2.如果用户选择看门狗为"Always_On"，编译器会自动使能看门狗功能。

3.Fcpu 编译选项仅对高速时钟有效。在 slow 模式下，Fcpu=Fosc/4(Fosc 为内部低速时钟)。

2.1.3 数据存储器(RAM)

☞ ***128 X 8-bit RAM***



2.1.4 系统寄存器

2.1.4.1 系统寄存器列表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	L	H	R	Z	Y	-	PFLAG	-	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
B	-	-	-	-	-	-	-	-	P0M	-	-	-	-	-	-	PEDGE
C	P1W	P1M	P2M	-	-	P5M	-	-	INTRQ	INTEN	OSCM	-	WDTR	TC1R	PCL	PCH
D	P0	P1	P2	-	-	P5	-	-	T0M	T0C	-	-	TC1M	TC1C	TC1R	STKP
E	P0UR	P1UR	P2UR	-	-	P5UR	@HL	@YZ	-	P1OC	-	-	-	-	-	-
F	STK7L	STK7H	STK6L	STK6H	STK5L	STK5H	STK4L	STK4H	STK3L	STK3H	STK2L	STK2H	STK1L	STK1H	STK0L	STK0H

说明：

PFLAG = ROM 页和特殊标志寄存器
H, L = 工作寄存器，@HL 和 ROM 寻址寄存器
P1W = P1 口唤醒功能寄存器
PnM = Pn 口输入/输出模式控制寄存器
P1OC = P1 口开漏控制寄存器
INTRQ = 中断请求寄存器
OSCM = 振荡模式寄存器
T0M = T0 模式寄存器
TC1M = TC1 模式寄存器
TC1R = TC1 自动装载数据缓存器
STKP = 堆栈指针
@YZ = 间接寻址寄存器

R = 工作寄存器和 ROM 查表数据缓存器
Y, Z = 工作寄存器，@YZ 和 ROM 寻址寄存器
PEDGE = P0.0 模式控制寄存器
Pn = Pn 口数据缓存器
PnUR = Pn 口上拉电阻控制寄存器
INTEN = 中断使能寄存器
PCH, PCL = 程序计数器
T0C = T0 计数寄存器
TC1C = TC1 计数寄存器
WDTR = 看门狗清零寄存器
STK0~STK7 = 堆栈
@HL = 间接寻址寄存器

2.1.4.2 系统寄存器的位地址配置表

地址	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	备注
080H	LBIT7	LBIT6	LBIT5	LBIT4	LBIT3	LBIT2	LBIT1	LBIT0	R/W	L
081H	HBIT7	HBIT6	HBIT5	HBIT4	HBIT3	HBIT2	HBIT1	HBIT0	R/W	H
082H	RBIT7	RBIT6	RBIT5	RBIT4	RBIT3	RBIT2	RBIT1	RBIT0	R/W	R
083H	ZBIT7	ZBIT6	ZBIT5	ZBIT4	ZBIT3	ZBIT2	ZBIT1	ZBIT0	R/W	Z
084H	YBIT7	YBIT6	YBIT5	YBIT4	YBIT3	YBIT2	YBIT1	YBIT0	R/W	Y
086H	NT0	NPD				C	DC	Z	R/W	PFLAG
0B8H							P01M	P00M	R/W	P0M
0BFH				P00G1	P00G0				R/W	PEDGE
0C0H	P17W	P16W	P15W	P14W	P13W	P12W	P11W	P10W	W	P1W
0C1H	P17M	P16M	P15M	P14M	P13M	P12M	P11M	P10M	R/W	P1M
0C2H	P27M	P26M	P25M	P24M	P23M	P22M	P21M	P20M	R/W	P2M
0C5H				P54M	P53M	P52M	P51M	P50M	R/W	P5M
0C8H		TC1IRQ		T0IRQ			P01IRQ	P00IRQ	R/W	INTRQ
0C9H		TC1IEN		T0IEN			P01IEN	P00IEN	R/W	INTEN
0CAH				CPUM1	CPUM0	CLKMD	STPHX		R/W	OSCM
0CCH	WDTR7	WDTR6	WDTR5	WDTR4	WDTR3	WDTR2	WDTR1	WDTR0	W	WDTR
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH					PC11	PC10	PC9	PC8	R/W	PCH
0D0H						P02	P01	P00	R/W	P0
0D1H	P17	P16	P15	P14	P13	P12	P11	P10	R/W	P1
0D2H	P27	P26	P25	P24	P23	P22	P21	P20	R/W	P2
0D5H				P54	P53	P52	P51	P50	R/W	P5
0D8H	T0ENB	T0rate2	T0rate1	T0rate0					R/W	T0M
0D9H	T0C7	T0C6	T0C5	T0C4	T0C3	T0C2	T0C1	T0C0	R/W	T0C
0DCH	TC1ENB	TC1rate2	TC1rate1	TC1rate0	TC1CKS	ALOAD1	TC1OUT	PWM1OUT	R/W	TC1M
0DDH	TC1C7	TC1C6	TC1C5	TC1C4	TC1C3	TC1C2	TC1C1	TC1C0	R/W	TC1C
0DEH	TC1R7	TC1R6	TC1R5	TC1R4	TC1R3	TC1R2	TC1R1	TC1R0	W	TC1R
0DFH	GIE					STKPB2	STKPB1	STKPB0	R/W	STKP
0E0H							P01R	P00R	W	P0UR
0E1H	P17R	P16R	P15R	P14R	P13R	P12R	P11R	P10R	W	P1UR
0E2H	P27R	P26R	P25R	P24R	P23R	P22R	P21R	P20R	W	P2UR
0E5H				P54R	P53R	P52R	P51R	P50R	W	P5UR
0E6H	@HL7	@ HL 6	@ HL5	@ HL4	@ HL3	@ HL2	@ HL1	@ HL0	R/W	@ HL
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ
0E9H							P11OC	P10OC	W	P1OC
0F0H	S7PC7	S7PC6	S7PC5	S7PC4	S7PC3	S7PC2	S7PC1	S7PC0	R/W	STK7L
0F1H	-	-	-	-	S7PC11	S7PC10	S7PC9	S7PC8	R/W	STK7H
0F2H	S6PC7	S6PC6	S6PC5	S6PC4	S6PC3	S6PC2	S6PC1	S6PC0	R/W	STK6L
0F3H	-	-	-	-	S6PC11	S6PC10	S6PC9	S6PC8	R/W	STK6H
0F4H	S5PC7	S5PC6	S5PC5	S5PC4	S5PC3	S5PC2	S5PC1	S5PC0	R/W	STK5L
0F5H	-	-	-	-	S5PC11	S5PC10	S5PC9	S5PC8	R/W	STK5H
0F6H	S4PC7	S4PC6	S4PC5	S4PC4	S4PC3	S4PC2	S4PC1	S4PC0	R/W	STK4L
0F7H	-	-	-	-	S4PC11	S4PC10	S4PC9	S4PC8	R/W	STK4H
0F8H	S3PC7	S3PC6	S3PC5	S3PC4	S3PC3	S3PC2	S3PC1	S3PC0	R/W	STK3L
0F9H				-	S3PC11	S3PC10	S3PC9	S3PC8	R/W	STK3H
0FAH	S2PC7	S2PC6	S2PC5	S2PC4	S2PC3	S2PC2	S2PC1	S2PC0	R/W	STK2L
0FBH				-	S2PC11	S2PC10	S2PC9	S2PC8	R/W	STK2H
0FCH	S1PC7	S1PC6	S1PC5	S1PC4	S1PC3	S1PC2	S1PC1	S1PC0	R/W	STK1L
0FDH				-	S1PC11	S1PC10	S1PC9	S1PC8	R/W	STK1H
0FEH	S0PC7	S0PC6	S0PC5	S0PC4	S0PC3	S0PC2	S0PC1	S0PC0	R/W	STK0L
0FFH				-	S0PC11	S0PC10	S0PC9	S0PC8	R/W	STK0H

✷ 注：

1. 有些位元要固定放入**“0”**或**“1 ”**（如上表中所示），以免系统出错。
2. 所有寄存器的名称在 **SN8ASM** 编译器是默认的。
3. 寄存器中各位的名称已在 **SN8ASM** 编译器中以**“F”**为前缀定义过。
4. 指令**“B0BSET”, “B0BCLR”, “BSET”, “BCLR”** 仅对 **“R/W”** 寄存器有效。
5. 详细细节请查阅“系统寄存器参照表”。

2.1.4.3 累加器（ACC）

累加器 ACC 是一个 8 位专用数据寄存器，用来进行算术逻辑运算或数据存储器之间数据的传送和处理。如果对 ACC 的操作结果为零（Z）或者有进位产生（C 或 DC），那么这些标志将会影响 PFLAG 寄存器。

由于 ACC 不在数据存储器（RAM）中，所以在立即寻找模式下不能通过执行“B0MOV”指令访问 ACC。

➲ **例：读/写 ACC 中的数据**

```
; 把 ACC 中的数据送到 BUF 中
                MOV     BUF, A
; 给 ACC 送立即数
                MOV     A, #0FH
; 把 BUF 的数据送给 ACC
                MOV     A, BUF
```

中断响应时，系统不会自动保存 ACC 和 PFLAG，所以如果中断发生，必须将 ACC 和 PFLAG 寄存器值存储在其它的存储器中，“PUSH”和“POP”指令在缓存器中保存和恢复 ACC 和 PFLAG 的值。

➲ **例：保护 ACC 和工作寄存器**

```
INT_SERVICE:
                PUSH                    ; 保存 ACC 和 PFLAG
                .
                .
                POP                     ; 恢复 ACC 和 PFLAG
                RETI                    ; 中断返回
```

2.1.4.4 程序状态寄存器（PFLAG）

PFLAG 包括复位标志，进位标志（C），辅助进位标志（DC）和零标志（Z），如果操作结果为 0 或是有进位、借位发生，就存入 PFLAG 寄存器中。

086H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PFLAG	NT0	NPD	-	-	-	C	DC	Z
读/写	R/W	R/W	-	-	-	R/W	R/W	R/W
复位后	-	-	-	-	-	0	0	0

☞ **复位标志**

NT0	NPD	复位状态
0	0	看门狗超时复位
0	1	保留
1	0	由 LVD 复位
1	1	由外部复位引脚复位

☞ **进位标志**

C = 1：执行算术加法后有进位发生，执行算术减法后没有借位或移位指令后移出逻辑“1”
C = 0：执行算术加法后没有进位发生，执行算术减法后有借位或移位指令后移出逻辑“0”

☞ **辅助进位标志**

DC = 1：执行算术加法操作产生由低字节向高字节的进位或执行算术减法操作没有从高字节借位
DC = 0：执行算术加法操作没有产生由低字节向高字节的进位或执行算术减法操作从高字节借位

☞ **零标志**

Z=1：指令执行后，运算结果为零
Z=0：指令执行后，运算结果非零

2.1.4.5 程序计数器（PC）

程序计数器 PC 是一个 12 位专用二进制计数器，分为 4 位的高字节和 8 位的低字节，PC 指向下一条将要执行的指令的地址 ，一般的，在程序执行过程中，PC 会随着指令的执行自动加 1。

此外，执行程序调用（CALL）和跳转（JMP）指令时，下一条将要执行的指令地址会被装入 PC 的 0～11 位。

	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PC	-	-	-	-	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
复位后	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0
	PCH								PCL							

☞ **单地址跳转**

单地址跳转功能共有 9 条指令： CMPRS、INCS、INCMS、DECS、DECMS、BTS0、BTS1、B0BTS0 和 B0BTS1，如果运算的结果符合跳转条件，则程序计数器加 2，会跳过当前指令的下一条指令。

如果位测试结果符合跳转条件，那么 PC 将加 2，跳过当前指令的下一条指令：

```
            B0BTS1      FC          ; 如果 C=1 则跳过下一条指令
            JMP         C0STEP      ; 否则跳转到 C0STEP.
            .
C0STEP:     NOP

            B0MOV       A, BUF0     ; 把 BUF0 的值赋给 ACC.
            B0BTS0      FZ          ; 如果 Z=0，则跳过下一条指令
            JMP         C1STEP      ; 否则跳到 C1STEP.
            .
C1STEP:     NOP
```

如果 ACC 与立即数或存储器中的内容相等，那么 PC 将加 2，跳过下一条指令

```
            CMPRS       A, #12H     ; 如果 ACC = 12H. 则跳过下一条指令
            JMP         C0STEP      ; 否则跳到 C0STEP.
            .
C0STEP:     NOP
```

如果增加 1 后的结果是从 0xffh 到 0x00h，那么 PC 将加 2，跳过下一条指令

INCS:

```
            INCS        BUF0
            JMP         C0STEP      ; 如果 ACC 不为 0 则跳到 C0STEP
            …
C0STEP:     NOP
```

INCMS:

```
            INCMS       BUF0
            JMP         C0STEP      ; 如果 BUF0 不为 0 则跳到 C0STEP
            …
C0STEP:     NOP
```

如果减小 1 后的结果是从 0x01h 到 0x00h，那么 PC 将加 2，跳过下一条指令

DECS:

```
            DECS        BUF0
            JMP         C0STEP      ; 如果 ACC 不为 0 则跳到 C0STEP
            …
C0STEP:     NOP
```

DECMS:

```
            DECMS       BUF0
            JMP         C0STEP      ; 如果 BUF0 不为 0 则跳到 C0STEP
            …
C0STEP:     NOP
```

☞ **多地址跳转**

用户可以通过 JMP 和“ ADD　PCL，A ”指令实现多地址跳转。 “ADD　PCL, A”执行后若有进位发生，进位标志并不会影响 PCH 寄存器。

➲ **例： 如果 PC = 0323H　(PCH = 03H、PCL = 23H)**

```
; PC = 0323H
                MOV       A, #28H
                B0MOV     PCL, A          ; 跳转到 0328H
                .         .
                .         .
; PC = 0328H    .         .
                MOV       A, #00H
                B0MOV     PCL, A          ; 跳转到 0300H
```

➲ **例： 如果 PC = 0323H　(PCH = 03H、PCL = 23H)**

```
; PC = 0323H
                B0ADD     PCL, A          ; PCL = PCL + ACC, PCH 的值不会改变.
                JMP       A0POINT         ; ACC = 0, 跳转到 A0POINT
                JMP       A1POINT         ; ACC = 1, 跳转到 A1POINT
                JMP       A2POINT         ; ACC = 2, 跳转到 A2POINT
                JMP       A3POINT         ; ACC = 3, 跳转到 A3POINT
                .         .               ;
```

✷ **只有“ADD M, A”，“ADC M, A”和“B0ADD M, A”支持 PCL 溢出和 PCH=PCH+1，其它的指令均不支持。**

2.1.4.6 H, L 寄存器

8 位系统专用寄存器 H 和 L 主要用作工作寄存器和间接寻址 RAM 数据。数据寄存器@HL 位于 RAM bank_0 的 E6H 单元，H 和 L 寄存器的内容决定了被访问的 RAM 单元的地址，可通过累加器 ACC 对此单元进行读/写。 H 寄存器的低 4 位决定了单元所在的 RAM 页，L 寄存器给出该单元在某 RAM 页中的具体地址。H 寄存器的高 4 位在间接寻址中无意义。

H 初始值=0000 0000

081H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
H	HBIT7	HBIT6	HBIT5	HBIT4	HBIT3	HBIT2	HBIT1	HBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	X	X	X	X	X	X	X	X

L 初始值=0000 0000

080H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
L	LBIT7	LBIT6	LBIT5	LBIT4	LBIT3	LBIT2	LBIT1	LBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	X	X	X	X	X	X	X	X

➲ **例：间接寻址读取 RAM bank_0 的 20H 单元数据。**

```
B0MOV   H, #00H     ; 把 BANK0 的 RAM 高 8 位地址送到 H 寄存器
B0MOV   L, #20H     ; 把 BANK0 的 RAM 低 8 位地址送到 L 寄存器
B0MOV   A, @HL      ; 读上面地址对应储存器的值到 ACC
```

➲ **例：用@HL 对 RAM bank 0 进行清零**

```
        CLR     H           ; 清零（Bank0）
        MOV     A, #07FH
        B0MOV   L, A        ; L = 7FH，数据存储器的低地址
CLR_HL_BUF:
        CLR     @HL         ; 清零
        DECMS   L           ; L 递减 1，L = 0 子程序结束
        JMP     CLR_HL_BUF  ; 不为零继续清零

        CLR     @HL
END_CLR:                    ; 结束通用区所有存储器的清零程序
```

2.1.4.7 Y, Z 寄存器

- 通用工作寄存器
- 通过寄存器@YZ 用作访问 RAM 的数据指针
- 通过 MOVC 查表指令，可访问 ROM 中的数据

084H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Y	YBIT7	YBIT6	YBIT5	YBIT4	YBIT3	YBIT2	YBIT1	YBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	X	X	X	X	X	X	X	X

083H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Z	ZBIT7	ZBIT6	ZBIT5	ZBIT4	ZBIT3	ZBIT2	ZBIT1	ZBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	X	X	X	X	X	X	X	X

➲ **例：间接寻址模式访问 RAM bank 0 中 025H 单元。**

```
B0MOV   Y, #00H     ; 用 Y 确定 BANK0
B0MOV   Z, #25H     ; 用 Z 确定 RAM 中的位置
B0MOV   A, @YZ      ; 25H 的数据放到 ACC 中
```

➲ **例：用寄存器@YZ 将数据存储器中 bank 0 的通用数据存储器清零**

```
        B0MOV   Y, #0           ; Y = 0, bank 0
        B0MOV   Z, #07FH        ; Y = 7FH, 数据存储区的最大地址

CLR_YZ_BUF:
        CLR     @YZ             ; @YZ =0

        DECMS   Z               ;
        JMP     CLR_YZ_BUF      ;

        CLR     @YZ
END_CLR:                        ; 完成对 BANK0 中的数据的清零
        .
```

2.1.4.8 R 寄存器

R 寄存器是一个 8 位缓存器，有两个主要功能：

- 工作寄存器
- 存放 ROM 查表的高字节数据

（MOVC 指令执行完后，指定的 ROM 地址中数据的高 8 位将存放在 R 寄存器中，低 8 位存放在 ACC 中）

082H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R	RBIT7	RBIT6	RBIT5	RBIT4	RBIT3	RBIT2	RBIT1	RBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	X	X	X	X	X	X	X	X

✷ **注：请参考使用 R 寄存器的查表功能“查表说明”。**

2.2 寻址模式

2.2.1 立即寻址模式

将一个立即数送入累加器或指定的 RAM 单元。

➲ **例：把立即数 12H 送入 ACC。**

```
MOV     A, #12H         ;立即数 12H 存入 ACC
```

➲ **例：把立即数 12H 送入 R 寄存器。**

```
B0MOV   R, #12H         ; 立即数 12H 存入 R 寄存器
```

✷ **注：在立即数寻找模式下，特殊的 RAM 必须是 0x80~0x87 工作寄存器。**

2.2.2 直接寻址模式

通过单元地址访问存储器就是直接寻找。

➲ **例：把 RAM 中的 0x12 送入 ACC。**

```
B0MOV   A, 12H          ; bank 0 中 12H 的数据送入 ACC
```

➲ **例：从 ACC 中把 0x12 移入 RAM 中。**

```
B0MOV   12H, A          ; 把 bank0 中的 12H 从 ACC 中取出并保存在 RAM 中
```

2.2.3 间接寻址模式

通过数据指针寄存器（H/L，Y/Z）访问存储器。

➲ **例：利用@HL 寄存器间接寻址。**

```
B0MOV   H, #0           ; 清 H，指向 bank 0.
B0MOV   L, #12H         ; 送入一个立即数 12H 到 L 寄存器。
B0MOV   A, @YZ          ; 用数据指针@HL 取得相应存储器中的数据 012H 中的数据送给 ACC
```

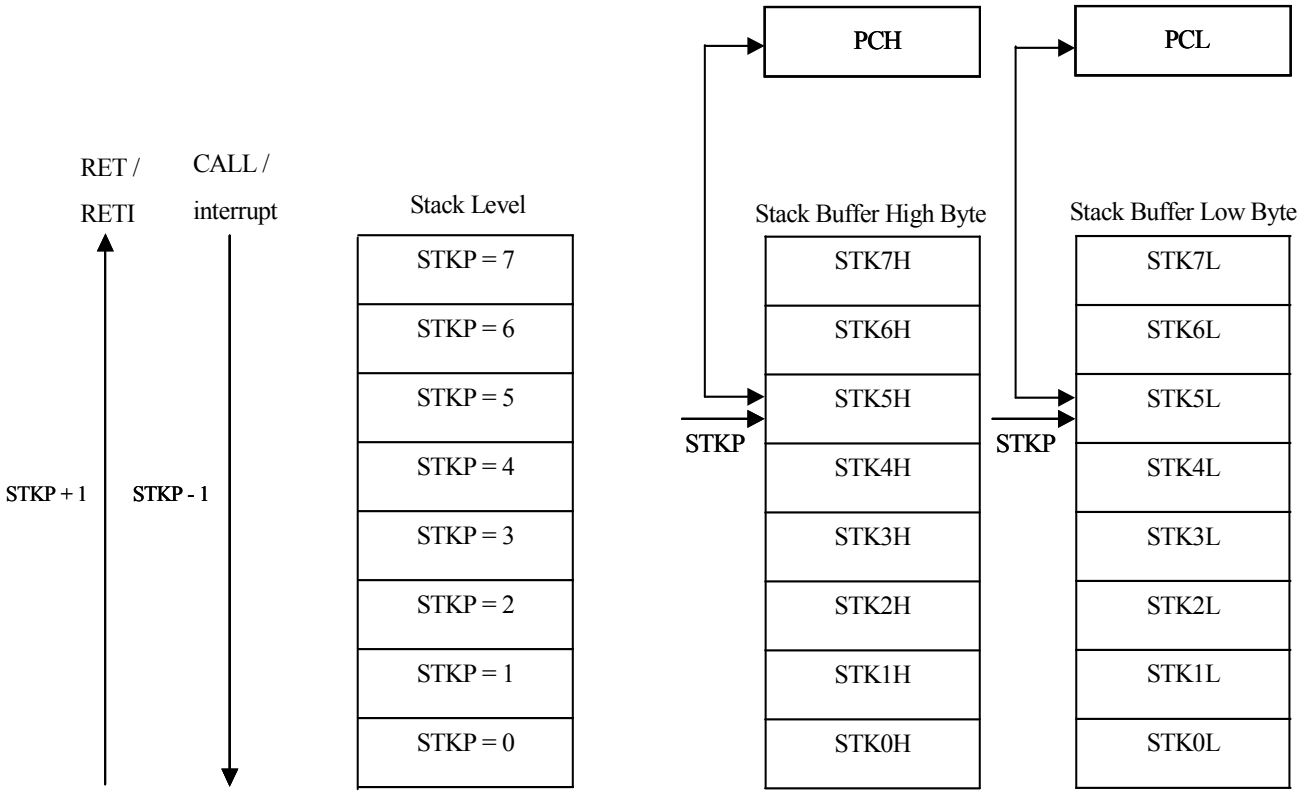
➲ **例：利用@YZ 寄存器间接寻址。**

```
B0MOV   Y, #0           ; 清 Y，指向 bank 0.
B0MOV   Z, #12H         ; 送入一个立即数 12H 到 Z 寄存器。
B0MOV   A, @YZ          ; 用数据指针@YZ 取得相应存储器中的数据 012H 中的数据送给 ACC
```

2.3 堆栈操作

2.3.1 概述

SN8P2604 共有 8 层堆栈。堆栈缓存器 STKnH 和 STKnL 在中断现场保护和恢复时用来存放程序计数器 PC 的数据。堆栈指针 STKP 指示当前栈顶位置以便保护和恢复数据。



2.3.2 堆栈指针寄存器

堆栈指针 STKP 是一个 3 位的寄存器，用来指示堆栈栈顶位置，12 位的数据存储器（STKnH 和 STKnL）用来存储程序计数器（PC）的值。

堆栈操作有两种，入栈（PUSH）和出栈（POP）。执行入栈（PUSH）操作，STKP 就会减一，执行出栈（POP）操作，STKP 就会加一。这样，STKP 总是指向栈顶位置。

执行 CALL 指令和响应中断时，程序计数器（PC）的值会被保存在堆栈中，堆栈操作遵循后进先出的原则。堆栈指针寄存器（STKP）和缓冲器 STKnH、STKnL 位于 BANK0。

0DFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKP	GIE	-	-	-	-	STKPB2	STKPB1	STKPB0
读/写	R/W	-	-	-	-	R/W	R/W	R/W
复位后	0	-	-	-	-	1	1	1

STKPBn：堆栈指针 (n = 0 ~ 2)
GIE：全局中断控制位， 0 = 禁止，1 = 使能。详见中断章节。

➲ **例：栈指针 (STKP)复位，我们强烈建议在程序的开始将堆栈指针清零。**

```
MOV     A, #00000111B
B0MOV   STKP, A
```

0F0H~0FFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKnH	-	-	-	-	SnPC11	SnPC10	SnPC9	SnPC8
读/写	-	-	-	-	R/W	R/W	R/W	R/W
复位后	-	-	-	-	0	0	0	0

0F0H~0FFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKnL	SnPC7	SnPC6	SnPC5	SnPC4	SnPC3	SnPC2	SnPC1	SnPC0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

STKn= <STKnH , STKnL> (n = 7 ~ 0)

2.3.3 堆栈操作举例

程序调用指令（CALL）和中断都涉及到对堆栈指针 STKP 的操作和将程序计数器 PC 保存在堆栈缓冲区。在两种操作中，堆栈指针 STKP 都会减 1 并指向下一个可用的堆栈区域，堆栈缓存器中则保存了程序指针。入栈保护操作如下表所示：

堆栈层数	*STKP*			*堆栈缓存器*		*说明*
	STKPB2	**STKPB1**	**STKPB0**	高字节	低字节	
0	1	1	1	Free	Free	-
1	1	1	0	STK0H	STK0L	-
2	1	0	1	STK1H	STK1L	-
3	1	0	0	STK2H	STK2L	-
4	0	1	1	STK3H	STK3L	-
5	0	1	0	STK4H	STK4L	-
6	0	0	1	STK5H	STK5L	-
7	0	0	0	STK6H	STK6L	-
8	1	1	1	STK7H	STK7L	-
> 8	1	1	0	-	-	**堆栈溢出，出错**

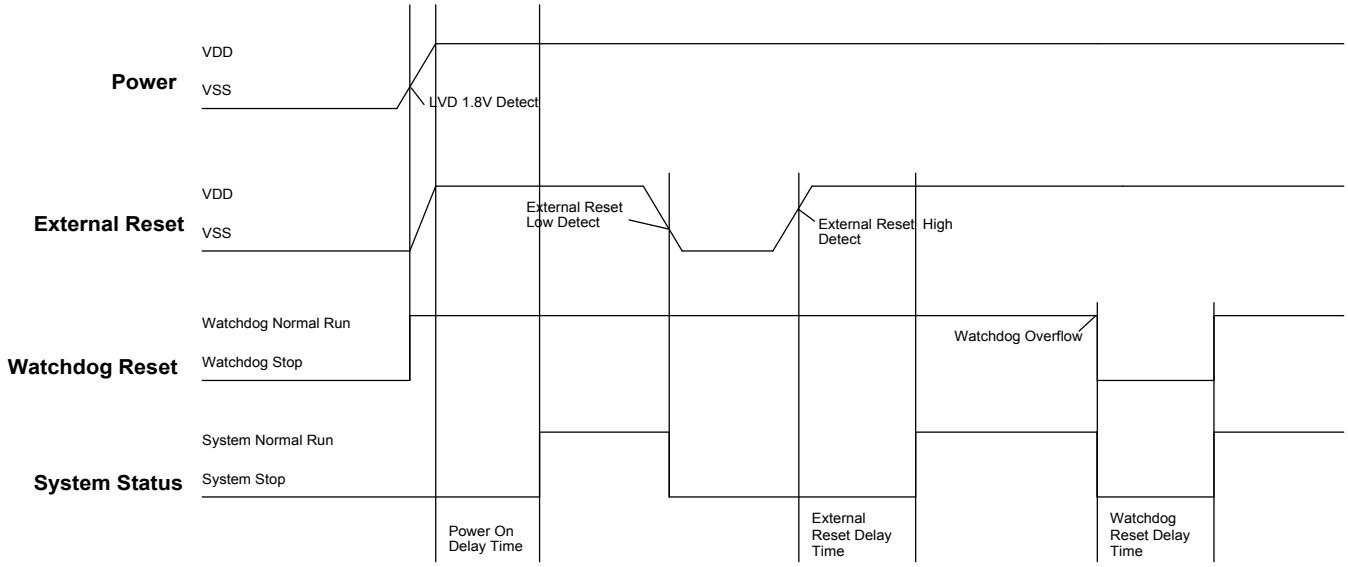
对应每个入栈操作，都有一个出栈操作来恢复程序计数器 PC 的值。RETI 指令用于中断服务程序中，RET 用于子程序调用。出栈时，STKP 加 1 并指向下一个空闲堆栈缓存器。堆栈恢复操作如下表所示：

堆栈层数	*STKP*			*堆栈缓存器*		*说明*
	STKPB2	**STKPB1**	**STKPB0**	高字节	低字节	
8	1	1	1	STK7H	STK7L	-
7	0	0	0	STK6H	STK6L	-
6	0	0	1	STK5H	STK5L	-
5	0	1	0	STK4H	STK4L	-
4	0	1	1	STK3H	STK3L	-
3	1	0	0	STK2H	STK2L	-
2	1	0	1	STK1H	STK1L	-
1	1	1	0	STK0H	STK0L	-
0	1	1	1	Free	Free	-

3 复位

3.1 概述

MCU 的复位方式有上电复位，外部复位和看门狗复位。当任何一种复位发生，系统复位并初始化系统寄存器。复位后，系统进入普通模式。复位时序图如下：



3.2 上电复位

接通电源当电源从 VSS 上升到 VDD 时，系统就会复位，进入普通模式。上电到系统正常运行之间的这段时间叫做“上电延迟时间”，系统会在系统延迟时间后正常运行。

VDD	晶振	上电延迟时间
3V	4MHz	大约 131ms
5V	4MHz	大约 66ms
3V	32768Hz	大约 239ms
5V	32768Hz	大约 174ms

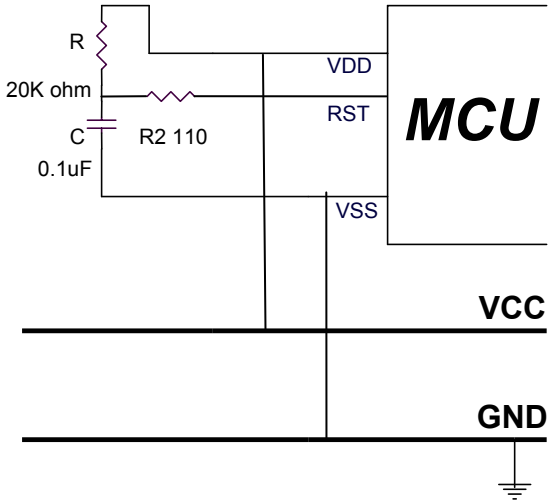
3.3 外部复位

外部复位引脚为施密特触发结构，当它设置为复位引脚时，就使能了外部复位功能。外部复位为低电平有效，当复位引脚侦测到低电压（低于 0.3VDD）时，系统开始复位，直到侦测到的电压达到高电平（高于 0.7VDD）。用户可以使用外部复位电路控制系统。

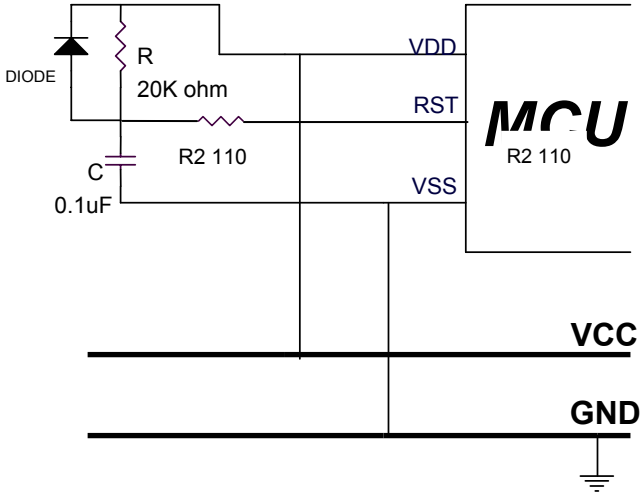
VDD	晶振	外部复位延迟时间
3V	4MHz	大约 131ms
5V	4MHz	大约 66ms
3V	32768Hz	大约 193ms
5V	32768Hz	大约 127ms

3.3.1 外部复位电路

外部复位电路就是一个简单的 RC 回路，如下所示：



简单 RC 复位电路



添加二极管后的节电复位电路

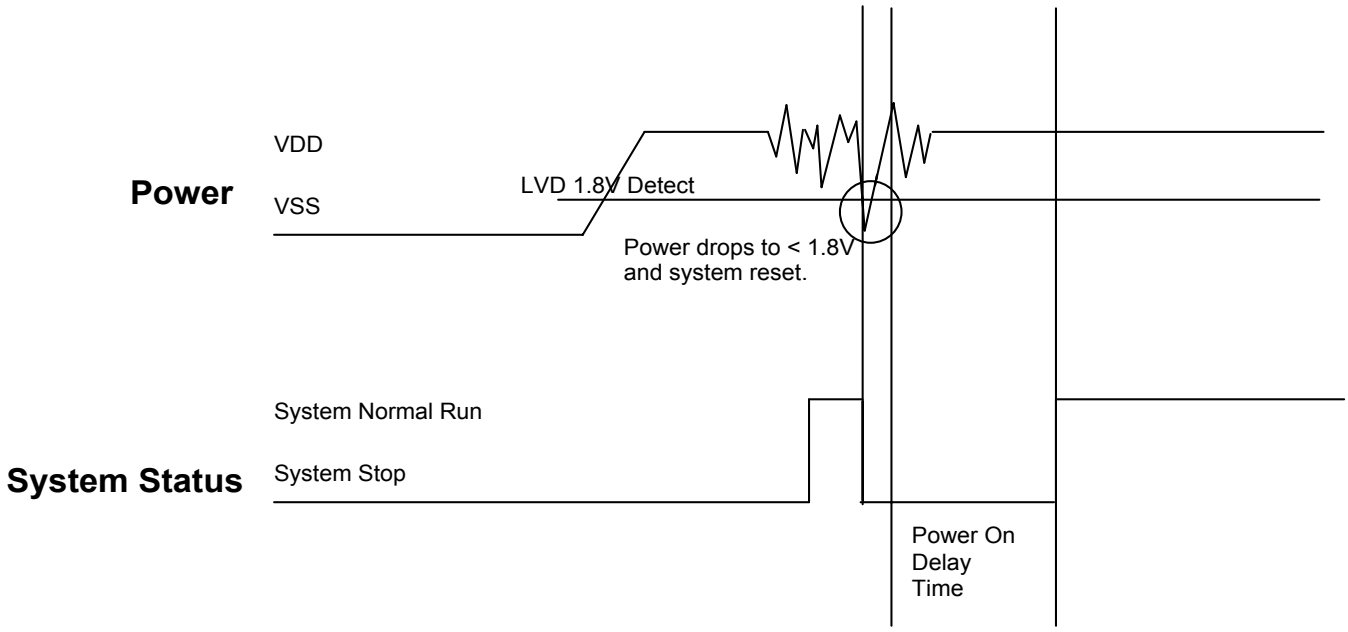
3.4 看门狗复位

看门狗复位是对系统的一种保护。在正常情况下，系统可以可以通过程序对看门狗清零保证正常运行，但在错误状态下，系统位于未知状态，在看门狗定时器溢出前无法由程序对看门狗清零。看门狗定时器溢出后，系统复位，系统重启并进入普通模式。

VDD	晶振	看门狗复位延迟时间
3V	4MHz	大约 145ms
5V	4MHz	大约 73ms
3V	32768Hz	大约 207ms
5V	32768Hz	大约 127ms

3.5 低电压侦测（LVD）

LVD 为低电压侦测，当侦测到 VDD 低于设定值时，系统就会复位。低电压侦测的电压值为 1.8V，如果 VDD 低于 1.8V，系统就会复位。如果 VDD 返回到 1.8V 以上，系统开始执行上电程序，在上电延迟时间结束后，系统开始正常运行，这叫做 BROWN OUT 复位。在高干扰环境下，上电会有些不稳定，LVD 可以防止噪音影响系统工作。当 VDD 保持 1.8V 以上的电压时，系统就会保持稳定的工作状态；当 VDD 低于 1.8V 时，LVD 使系统复位并等待 VDD 返回到正常电压。LVD 复位延迟就是上电复位延迟时间。



4 系统时钟

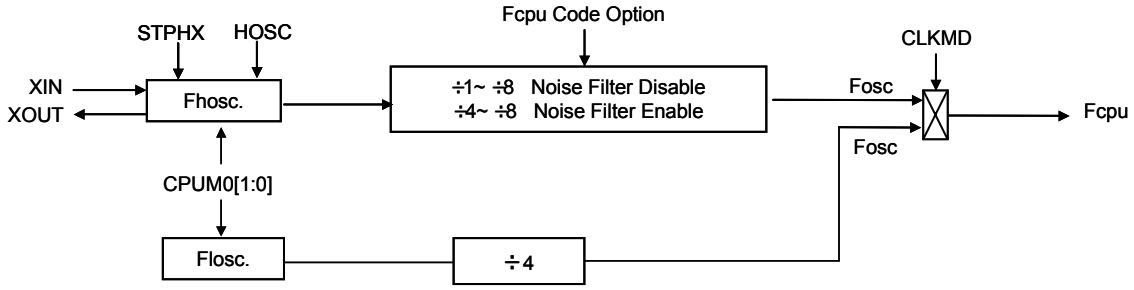
4.1 概述

SN8P2604 是具有高速时钟和低速时钟的双时钟微控制器。高速时钟是由外部振荡电路提供，而低速时钟则是由内置低速 RC 振荡电路（ILRC 16KHz @3V, 32KHz @5V）提供。

高低速时钟均可作为系统时钟（Fosc）。

- ☞ **普通模式（高速时钟）：** **Fcpu = Fosc/N**，N = 1~8，N 由 Fcpu 编译选项选择。
- ☞ **低速模式（低速时钟）：** **Fcpu = Fosc/4**

4.2 时钟框图



- HOSC：High_Clk 编译选项
- Fhosc：外部高速时钟
- Flosc：内部低速 RC 时钟

4.3 OSCM 寄存器

OSCM 寄存器是一个振荡器控制寄存器，它控制着振荡器的状态和系统模式。

0CAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OSCM	0	0	0	CPUM1	CPUM0	CLKMD	STPHX	0
读/写	-	-	-	R/W	R/W	R/W	R/W	-
复位后	-	-	-	0	0	0	0	-

Bit1 **STPHX**：外部高速振荡器控制位
0 = 运行
1 = 停止，内部低速振荡器仍处于运行状态。

Bit2 **CLKMD**：系统高/低速模式控制位
0 = 普通模式（双重时钟），系统时钟为高速时钟。
1 = 低速模式，系统时钟为内部低速时钟

Bit[4:3] **CPUM1, CPUM0**：CPU 运行模式控制位。
00 = 普通模式
01 = 睡眠（省电）模式
10 = 绿色模式
11 = 保留

➲ **例：停止高速振荡器**

```
B0BSET    FSTPHX    ; 仅停止高速振荡器
```

➲ **例：在省电模式下，高速振荡器和内部低速振荡器都停止。**

```
B0BSET    FCPUM0    ; 停止外部高速和内部低速振荡器进入省电（睡眠）模式
```

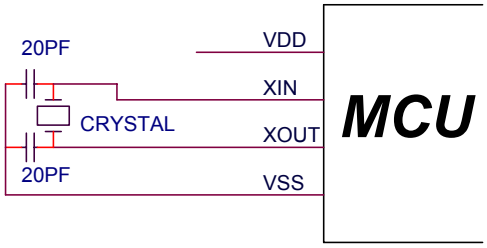
4.4 系统高速时钟

4.4.1 外部高速时钟

外部高速时钟有三种（晶体，RC和外部时钟信号)。高速振荡电路由编译选项的HIGH_CLK选项控制。

4.4.1.1 晶体振荡器

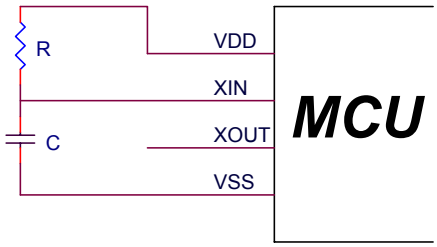
晶体振荡由 XIN，XOUT 引脚驱动。当使用高/一般/低的频率时，驱动电流是不一样的。HIGH_CLK 的编译选项支持不同的频率。 高速：12M-option.(如：12M)； 一般：4M-option (如：4M)；低速：32k_option(如：32.768K)



晶体振荡器

4.4.1.2 RC 振荡器

可以由 HIGH_CLK 选项中的 RC 选项来选择 RC 振荡器。RC 振荡器的频率可以达到 16M。建议改变“R”的值来改变频率。“C”的大小一般为 50P~100P。P1.4 是 XOUT，为无上拉电阻的单向输入引脚。



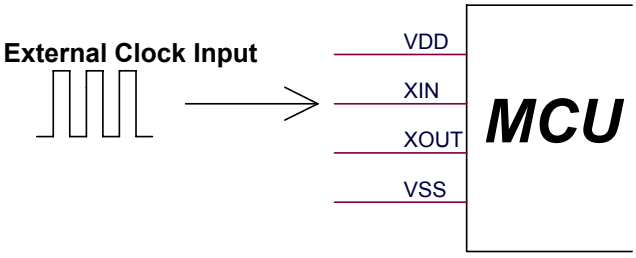
RC 振荡器

✷ 注：R 要尽可能地靠近 MCU 的 VDD.

✷ 注：C 要尽可能地靠近 MCU 的 VSS。

4.4.1.3 外部时钟输入

外部时钟输入做为系统时钟信号，由 HIGH_CLK 选项中的 RC 选项来选择。外部时钟输入信号由 XIN 引脚输入。P1.4 为 XOUT，是无上拉电阻的单向输入引脚。



外部时钟输入

✷ 注：外部振荡器电路的地线必须和微控制器的 Vss 引脚直接相连接。

4.4.2 系统低速时钟

SN8P2604 内置内部低速振荡器，其时钟源由 RC 振荡电路提供。

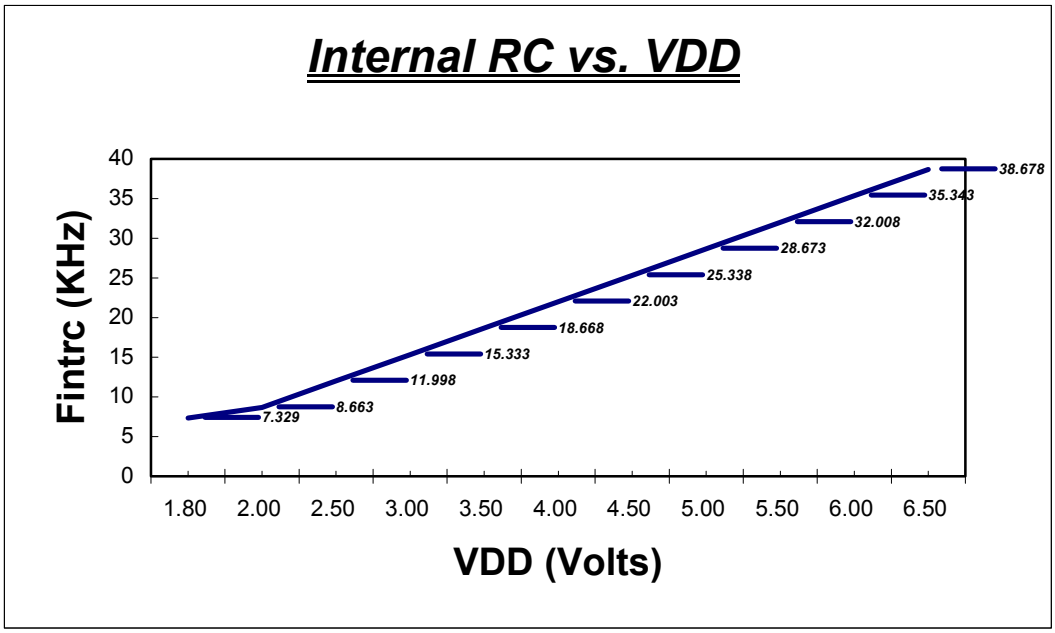
☞ **系统低速时钟（Fcpu）= 内部低速时钟（16KHz @3V，32KHz @5V）/ 4**

■ **例：停止内部低速时钟。**

```
B0BSET    FCPUM0      ; 停止外部高速和内部低速振荡器进入省电（睡眠）模式
```

✷ **注：内部低速时钟不能被单独停止，由 OSCM 寄存器的 CPUM0 位控制。**

低速振荡器采用 RC 振荡电路，频率受系统电压和温度的影响。通常情况下，RC 振荡器的频率约为 16KHZ（3V）、32KHZ（5V）。RC 频率和电压之间的关系如下图所示：



➲ **例：由 Fcpu 测试内部 RC 频率。 内部 RC 频率等于 4 倍的 Fcpu。我们可以从 Fcpu 得到内部 RC 的频率.**

```
        B0BSET    P1M.0       ; 设置 P1.0 为输出模式

        B0BSET    FCLKMD      ; 进入内部低速模式
@@:
        B0BSET    P1.0        ; 在低速模式下输出频率信号
        B0BCLR    P1.0        ; 通过测量指令周期来测试频率
        JMP       @B
```

4.4.3 振荡器频率测试

在 RC 模式，用户可以由指令周期（Fcpu）来测试 Fosc 的周期。

➲ **例：由测量指令周期来测量外部振荡器的周期**

```
        B0BSET    P1M.0       ; 设置 P1.0 为输出模式
@@:
        B0BSET    P1.0        ; 输出方波信号
        B0BCLR    P1.0        ; 通过测量指令周期来测试频率
        JMP       @B
```

✷ **注：不能直接由 XIN 测试 RC 频率；探针会影响 RC 的频率值。**

5 系统操作模式

5.1 概述

SN8P2604 具有低功耗的特性，能在 4 种不同的操作模式下相互转换。

- 普通模式
- 低速模式
- 省电模式（睡眠模式）
- 绿色模式

■ 普通模式

普通模式下，系统时钟源是外部高速时钟。上电复位后，系统就在普通模式下运行。指令周期为 Fosc/4，当外部高速振荡器的频率为 3.58MHz 时，指令周期就是 3.58MHz/4=895KHz。系统可以在普通模式下转入省电模式，低速模式和绿色模式。

☞ 低速模式

低速模式时，系统时钟源为内部低速 RC 时钟。设置 CLKMD=1，系统就进入内部低速模式。低速模式下的运行与普通模式一样（除了内部 RC 时钟），仅是时钟频率有所降低。系统可在低速模式下转入普通模式及低功耗的绿色模式和省电模式。

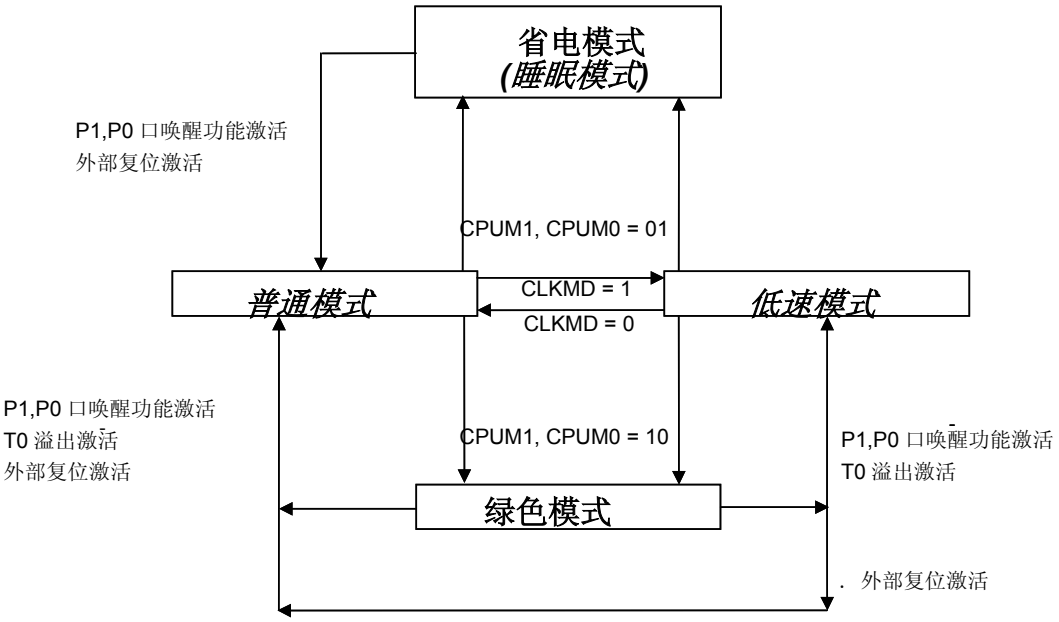
☞ 绿色模式

绿色模式提供一个定时唤醒功能。用户可通过设置定时器 T0 来确定系统的唤醒时间。系统可以从普通模式和低速模式进入绿色模式。普通模式下，T0 定时器的溢出时间非常短，而在低速模式下，其溢出时间则较长。用户可根据具体的应用选择合适的模式。系统可以由 T0 定时器或由 P0、P1 的触发信号来唤醒。

☞ 省电模式

省电模式也称睡眠模式，系统进入睡眠状态后 MCU 停止工作。设 CPUM0=1，系统进入省电模式，外部高速和内部低速振荡器均停止工作，P0、P1 的触发信号可将系统唤醒。

5.2 系统模式转换



系统模式转换简图

操作模式说明

模式	普通模式	低速模式	绿色模式	省电模式 （睡眠模式）	备注
HX osc	运行	由 STPHX 位决定	由 STPHX 位决定	停止	
LX osc	运行	运行	运行	停止	
CPU	执行	执行	停止	停止	
T0	*有效	*有效	*有效	无效	*程序激活
TC1	*有效	*有效	无效	无效	*程序激活
看门狗定时器	Watch_Dog	Watch_Dog	Watch_Dog	Watch_Dog	Enable Disable Always_on
内部中断	全部有效	全部有效	T0	全部无效	
外部中断	全部有效	全部有效	全部有效	全部无效	
系统唤醒	-	-	P0，P1，T0 复位	P0，P1 复位	

5.2.1 系统模式转换

➲ **例：普通/低速模式进入省电（睡眠）模式**

```
B0BSET      FCPUM0      ; 设置 CPUM0 = 1.
```

✸ **注：系统在睡眠模式中，只有具有唤醒功能的引脚和复位引脚能够将系统唤醒。**

➲ **例：普通模式转换为低速模式**

```
B0BSET      FCLKMD      ; 设置 CLKMD = 1, 进入低速模式
B0BSET      FSTPHX      ; 停止高速振荡器以省电
```

➲ **例：低速模式转换为普通模式(外部高速振荡器仍然运行)**

```
B0BCLR      FCLKMD      ;设置 CLKMD = 0
```

➲ **例：低速模式转换为普通模式（外部高速振荡器停止）**

如果外部高速时钟停止时程序欲重新回到普通模式，就必须延迟至 10mS 以等待外部时钟稳定下来。

```
            B0BCLR      FSTPHX      ; 启动外部高速振荡器

            B0MOV       Z, #27      ; 若 VDD = 5V, 则内部 RC 的频率为 32KHz (典型值)
@@:         DECMS       Z           ; 高速振荡器稳定时间 0.125ms X 81 = 10.125ms
            JMP         @B
                                    ;
            B0BCLR      FCLKMD      ;进入普通模式
```

➲ **例：进入绿色模式并使能 T0 的唤醒功能**

```
; 设置定时器 T0 的唤醒功能.
            B0BCLR      FT0IEN      ; 禁止 T0 中断
            B0BCLR      FT0ENB      ; 禁止 T0 计数
            MOV         A,#20H      ;
            B0MOV       T0M,A       ; 设置 T0 时钟源 = Fcpu / 64
            MOV         A,#74H
            B0MOV       T0C,A       ; 设置 T0C 初始值= 74H (设置 T0 定时间隔 = 10 ms)
            B0BCLR      FT0IEN      ; 禁止 T0 中断
            B0BCLR      FT0IRQ      ; 清 T0 中断请求标志
            B0BSET      FT0ENB      ; 使能 T0 计数
; 进入绿色模式
            B0BCLR      FCPUM0      ;设置 CPUMx = 10
            B0BSET      FCPUM1
```

✸ **注：若 T0ENB=0，T0 不能将系统从绿色模式唤醒进入普通/低速模式。**

5.3 唤醒时间

5.3.1 概述

外部高速振荡器从停止到运行需要一段时间的延迟，这段延迟时间对振荡器的稳定工作是必需的。在有些应用中，高速时钟可能需要经常的开停。外部高速振荡器重新启动需要的这一延迟时间称为唤醒时间。

下面两种情况需要唤醒时间：一种是从省电模式转换到正常模式，另一种是从低速模式转换到普通模式。对前一种情况，SN8P2604 提供了 4096 个振荡时钟作为唤醒时间，后一种情况需要用户自行计算唤醒时间。

5.3.2 唤醒时间

当系统处于省电（睡眠）模式时，外部高速振荡器停止运行。从睡眠模式唤醒时，SN8P2604 提供 4096 个外部高速振荡周期作为唤醒时间，以使振荡电路达到稳定状态。唤醒时间结束后，系统进入正常模式，唤醒时间的计算方法如下：

唤醒时间 = 1/Fosc×4096 (sec)＋X'tal 稳定时间

X'tal 稳定时间决定于 X'tal 的类型，一般的，约为 2～4ms（4MHz 晶体振荡器）

➲ **例：省电（睡眠）模式下，系统由 P0、P1 的触发信号唤醒。唤醒时间后，系统进入普通模式，P0、P1 唤醒功能的唤醒时间如下：**

唤醒时间=1/Fosc*4.96=1.14ms (Fosc=3.58MHz)

总的唤醒时间=1.14ms+X'tal 稳定时间

5.3.3 P1W 唤醒控制寄存器

省电（睡眠）模式下，具有唤醒功能的 P0 和 P1 都能将系统唤醒，P0 永远具有唤醒功能，而 P1 的唤醒功能受寄存器 P1W 控制。当 P0 或 P1 口的电平发生变化就可产生唤醒信号。

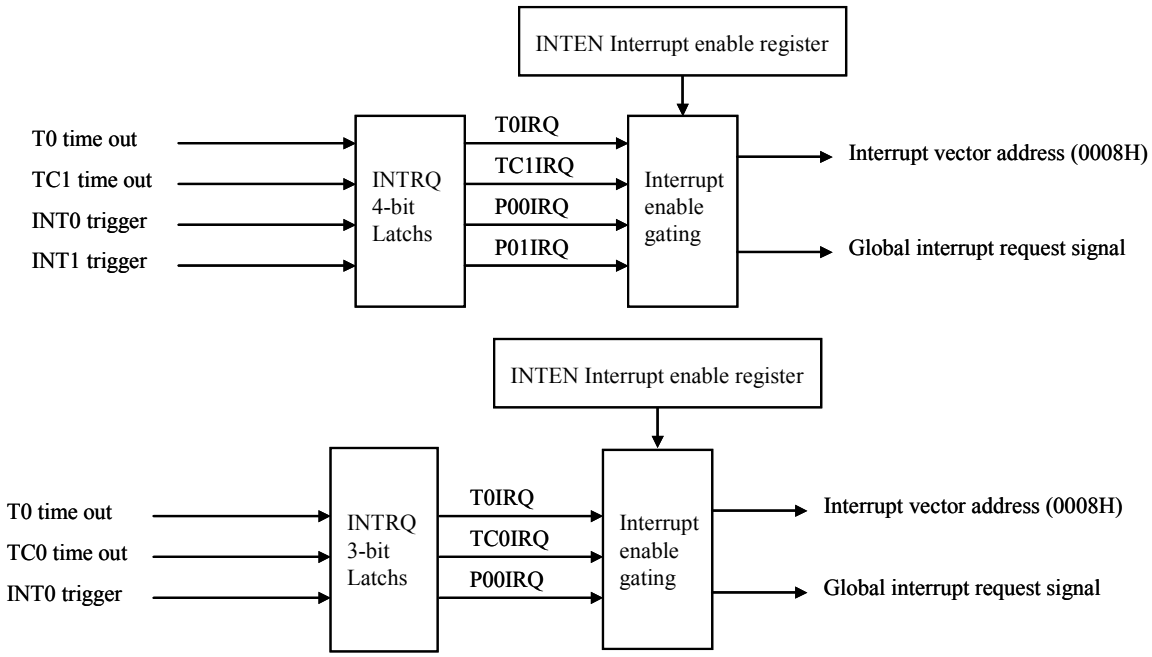
0C0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1W	P17W	P16W	P15W	P14W	P13W	P12W	P11W	P10W
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

Bit[5:0] **P10W~P17W**：P1 唤醒功能控制位
0=禁止 P1n 唤醒功能
1=使能 P1n 唤醒功能

6 中断

6.1 概述

SN8P2604 有 4 个中断源：两个内部中断源（T0/TC1），一个外部中断源（INT0/INT1）。外部中断能够唤醒睡眠模式进入高速模式。当系统进入到中断服务程序时，全局中断控制位 GIE 清零，系统退出中断服务后，GIE 被置为“1”以准备响应下一个中断请求。所有的中断请求存放于 INTRQ 中，用户可通过程序设置中断优先级。



✷ 注：GIE 使能后，才能响应各中断。

6.2 INTEN 中断使能寄存器

INTEN 为中断使能控制寄存器，包括内部中断和外部中断的控制位。INTEN 的某位被置“1”， 则相对应的中断请求便能够被响应。一旦有中断发生，程序将跳至 ORG 8 处执行中断服务程序。当执行到中断服务返回指令（RETI）时，将退出中断程序。

0C9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTEN	-	TC1IEN	-	T0IEN	-	-	P01IEN	P00IEN
读/写	-	R/W	-	R/W	-	-	R/W	R/W
复位后	-	0	-	0	-	-	0	0

Bit0 **P00IEN**：外部 P0.0 中断（INT0）控制位
0=禁止 INT0 中断功能
1=使能 INT0 中断功能

Bit1 **P01IEN**：外部 P0.1 中断（INT1）控制位
0=禁止 INT1 中断功能
1=使能 INT1 中断功能

Bit4 **T0IEN**：定时器 T0 中断控制位
0=禁止 T0 中断功能
1=使能 T0 中断功能

Bit6 **TC1IEN**：定时器 TC1 中断控制位
0=禁止 TC1 中断功能
1=使能 TC1 中断功能

6.3 INTRQ 中断请求寄存器

INTRQ 为中断请求寄存器，包含了所有的中断请求标志，当有中断发生时，INTRQ 寄存器中的相应位会置为“1”。中断请求标志需要用软件清零。用户通过检查中断请求寄存器可以知道中断的种类，从而执行相应的中断服务程序。

0C8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTRQ	-	TC1IRQ	-	T0IRQ	-	-	P01IRQ	P00IRQ
读/写	-	R/W	-	R/W	-	-	R/W	R/W
复位后	-	0	-	0	-	-	0	0

Bit0 **P00IRQ**：外部 P0.0 中断（INT0）请求位
0=无 INT0 中断请求
1=请求 INT0 中断

Bit1 **P01IRQ**：外部 P0.1 中断（INT1）请求位
0=无 INT1 中断请求
1=请求 INT1 中断

Bit4 **T0IRQ**：定时器 T0 中断控制位
0=无 T0 中断请求
1=请求 T0 中断

Bit6 **TC1IRQ**：定时器 TC1 中断控制位
0=无 TC1 中断请求
1=请求 TC1 中断

6.4 中断操作举例

6.4.1 总中断操作

GIE 是总中断控制位。所有的中断在 GIE 使能的前提下才能够得到响应。一旦有中断请求发生，程序计数器 PC 指向中断向量地址（ORG 8），堆栈层数加 1。

GIE 是总中断控制位。所有的中断在 GIE 使能的前提下才能够得到响应。一旦有中断请求发生，程序计数器 PC 指向中断向量地址（ORG 8），堆栈层数加 1。

0DFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKP	GIE	-	-	-	-	STKPB2	STKPB1	STKPB0
读/写	R/W	-	-	-	-	R/W	R/W	R/W
复位后	0	-	-	-	-	1	1	1

Bit7 **GIE**：全局中断控制位。
0=禁止
1=使能

➲ **例：设置全局中断控制位（GIE）**

```
B0BSET      FGIE            总中断使能
```

✷ **注：GIE 必须使能，所有中断才能被响应。**

6.4.2 PUSH, POP 程序

中断发生时，系统跳到 ORG 8 处执行中断服务程序，此时必须保存 ACC、PFLAG 的值，用 PUSH，POP 指令进入/退出中断服务程序。在中断服务程序完成后，用 PUSH，POP 指令保存/恢复 ACC，PFLAG 以避免主程序出错。

➲ **例：在中断服务程序发生时由指令 PUSH，POP 保存 ACC 和 PFLAG 的值。**

```
            ORG     0
            JMP     START

            ORG     8
            JMP     INT_SERVICE

            ORG     10H
START:
            …
INT_SERVICE:
            PUSH                    ; 保存 ACC 和 PFLAG
            .
            .
            POP                     ; 恢复 ACC 和 PFLAG

            RETI                    ; 中断返回
            …
            ENDP
```

6.4.3 INT0(P0.0)中断

当 INT0 中断发生时，不管 P00IEN 是否使能，P00IRQ 都会置“1”。若 P00IEN=1，且 P00IRQ=1，那么系统就进入中断向量地址（ORG 8）执行中断服务程序。但若 P00IEN=0，不管 P00IRQ 是否等于 1，系统都不进入中断。用户应注意多种中断下的处理。

✷ **注：P0.0 的中断触发的方式由 PEDGE 寄存器控制。**

0BFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PEDGE	-	-	-	P00G1	P00G0	-	-	-
读/写	-	-	-	R/W	R/W	-	-	-
复位后	-	-	-	1	0	-	-	-

Bit[4:3] **P00G[1:0]:** P0.0 中断触发边沿控制位：
00=保留
01=上升沿
10=下降沿
11=上升/下降沿双向（改变电平）

➲ **例：INT0 中断请求，双边沿触发。**

```
        MOV     A, #18H
        B0MOV   PEDGE, A        ; 设置 INT0 中断触发为双向触发

        B0BSET  FP00IEN         ; INT0 中断使能
        B0BCLR  FP00IRQ         ; 清 INT0 中断请求标志
        B0BSET  FGIE            ; 总中断使能
```

➲ **例：INT0 中断服务程序**

```
        ORG     8               ; 中断向量地址
        JMP     INT_SERVICE
INT_SERVICE:
        PUSH                    ; 保存 ACC 和 PFLAG

        B0BTS1  FP00IRQ         ; 判断是否有外部中断请求
        JMP     EXIT_INT        ; P00IRQ=1，退出中断

        B0BCLR  FP00IRQ         ; 清中断标志
        .       .               ; 中断服务程序
EXIT_INT:
        POP                     ; 恢复 ACC 和 PFLAG

        RETI                    ;中断返回
```

6.4.4 INT1(P0.1)中断操作

当 INT1 中断发生时，不管 P01IEN 是否使能，P01IRQ 都会置“1”。若 P01IEN=1，且 P01IRQ=1，那么系统就进入中断向量地址（ORG 8）执行中断服务程序。但若 P01IEN=0，不管 P01IRQ 是否等于 1，系统都不进入中断。用户应注意多种中断下的处理。

✷ **注：P0.1 由下降沿触发中断。**

➲ **例：初始化 INT1**

```
        B0BSET      FP01IEN         ; INT1 中断使能
        B0BCLR      FP01IRQ         ; 清 INT1 中断请求
        B0BSET      FGIE            ; 总中断使能
```

➲ **例：INT1 中断服务。**

```
        ORG         8               ; 中断向量地址
        JMP         INT_SERVICE
INT_SERVICE:
        PUSH                        ; 保存 ACC 和 PFLAG 的值
        B0BTS1      FP01IRQ         ; 判断是否有外部中断请求
        JMP         EXIT_INT        ;
        B0BCLR      FP01IRQ         ; 清中断标志
        .           .               ; INT1 中断服务程序
        .           .
EXIT_INT:
        POP                         ; 恢复 ACC 和 PFLAG
        RETI                        ; 中断返回
```

6.4.5 T0 中断操作

当 T0C 计数器溢出，无论 T0IEN 是否使能，T0IRQ 都会置“1”。若 T0IEN=1，且 T0IRQ =1，那么系统就进入中断向量地址（ORG 8）执行中断服务程序。若 T0IEN =0，不管 T0IRQ 是否等于 1，系统都不进入中断。用户应注意多种中断下的处理。

➲ **例：初始化 T0**

```
        B0BCLR      FT0IEN          ; 禁止 T0 中断
        B0BCLR      FT0ENB          ; 停止 T0 计数
        MOV         A, #20H         ;
        B0MOV       T0M, A          ; 设置定时模式 Fcpu / 64
        MOV         A, #74H         ; 设置时间常数
        B0MOV       T0C, A          ; 定时中断为 10 ms

        B0BCLR      FT0IRQ          ; 清 T0 中断请求标志
        B0BSET      FT0IEN          ; 使能 T0 中断
        B0BSET      FT0ENB          ; 开始 T0 计数

        B0BSET      FGIE            ; 使能总中断
```

➢ **注：避免错误的响应中断，在设置定时/计数器中断时必须先清除定时/计数器的中断请求标志位 T0IRQ，再开放中断和定时/计数器，如上例。**

➲ **例：T0 中断服务程序**

```
        ORG         8               ; 中断向量地址
        JMP         INT_SERVICE
INT_SERVICE:

        PUSH                        ; 保存 ACC 和 PFLAG

        B0BTS1      FT0IRQ          ; 检查是否是 T0 中断请求
        JMP         EXIT_INT        ;

        B0BCLR      FT0IRQ          ; 清 T0 中断标志
        MOV         A, #74H
        B0MOV       T0C, A          ; 重新装载时间常数
        .           .               ; T0 中断服务程序
        .           .
EXIT_INT:
        POP                         ; 恢复 ACC 和 PFLAG

        RETI                        ; 中断返回
```

6.4.6 TC1 中断操作

当计数器 TC1C 溢出时，无论 TC1IEN 是否使能，TC1IRQ 都会置“1”。若 TC1IEN =1，且 TC1IRQ =1，那么系统就进入中断向量地址（ORG 8）执行中断服务程序。若 TC1IEN =0，不管 TC1IRQ 是否等于 1，系统都不进入中断。用户应注意多种中断下的处理。

➲ **例：初始化 TC1**

```
        B0BCLR      FTC1IEN         ; 禁止 TC1 中断
        B0BCLR      FTC1ENB         ; 停止 TC1 计数
        MOV         A, #20H         ;
        B0MOV       TC1M, A         ; 设置 TC1 定时模式 Fcpu / 64
        MOV         A, #74H         ; 设置 TC1 的初始值 = 74H
        B0MOV       TC1C, A         ; 定时中断为 10 ms

        B0BCLR      FTC1IRQ         ; 清 TC1 中断请求标志
        B0BSET      FTC1IEN         ; 使能 TC1 中断
        B0BSET      FTC1ENB         ; 开始 TC1 计数

        B0BSET      FGIE            ; 使能总中断
```

➢ **注：避免错误的响应中断，在设置定时/计数器中断时必须先清除定时/计数器的中断请求标志位 TC1IRQ，再开放中断和定时/计数器，如上例。**

➲ **例：TC1 中断服务**

```
        ORG         8               ; 中断向量地址
        JMP         INT_SERVICE
INT_SERVICE:

        PUSH                        ; 保存 ACC 和 PFLAG

        B0BTS1      FTC1IRQ         ; 检查是否是 TC1 中断请求
        JMP         EXIT_INT        ;

        B0BCLR      FTC1IRQ         ; 清 TC1 中断标志
        MOV         A, #74H
        B0MOV       TC1C, A         ; 重新装载时间常数
        .           .               ; TC1 中断服务程序
EXIT_INT:
        POP                         ; 恢复 ACC 和 PFLAG

        RETI                        ; 中断返回
```

6.4.7 多个中断操作

大部分情况下，用户需要同时处理多个中断。处理多个中断就需要设置中断的优先权。中断请求由不同的事件控制，但是，有中断请求并不意味着系统就会去执行中断服务程序。不管中断是否使能，都可触发中断请求，一旦有中断发生，相应的中断请求标志就会被置为“1”。各中断与对应的触发事件关系如下表所示：

中　断	触　发　信　号
P00IRQ	由 PEDGE 寄存器控制 P0.0 触发
P01IRQ	P0.1 下降沿触发
T0IRQ	T0C 溢出
TC1IRQ	TC1C 溢出

在处理多中断请求下，用户必须对各中断进行优先权的设置，并根据 IEN 和 IRQ 的状态决定系统是否响应中断请求。用户必须在中断向量里检查中断控制位和中断请求标志位。

➲ **例：在多中断情况下，检查是否响应各中断请求**

```
                ORG         8               ; 中断向量地址
                NOP                         ; ORG 8 处的第一条指令
                PUSH                        ; 保存 ACC 和 PFLAG

INTP00CHK:                                  ; 检查是否有 INT0 中断
                B0BTS1      FP00IEN         ; 检查是否允外部中断 0
                JMP         INTTC0CHK       ; 跳转到下一个中断
                B0BTS0      FP00IRQ         ; 检测是否有外部中断 0 的请求
                JMP         INTP00          ; 跳转到 INT0 的中断服务程序
INTP01CHK:                                  ; 检查是否有 INT1 中断
                B0BTS1      FP01IEN         ; 检查是否允许外部 INT1 中断
                JMP         INTT0CHK        ; 跳转到下一个中断
                B0BTS0      FP01IRQ         ; 检测是否有外部中断 INT1 的请求
                JMP         INTP01          ; 跳转到 INT1 的中断服务程序
INTT0CHK:                                   ; 检查是否有 T0 中断
                B0BTS1      FT0IEN          ; 检查是否允 T0 中断
                JMP         INTTC0CHK       ; 跳转到下一个中断
                B0BTS0      FT0IRQ          ; 检测是否有 T0 中断请求
                JMP         INTT0           ; 跳转到 T0 中断服务程序
INTTC0CHK:                                  ; 检查是否有 TC1 中断
                B0BTS1      FTC1IEN         ; 检查是否允 TC1 中断
                JMP         INT_EXIT        ; 中断返回
                B0BTS0      FTC1IRQ         ; 检测是否有 TC1 中断请求
                JMP         INTTC1          ; 跳转到 TC1 中断服务程序
INT_EXIT:
                POP                         ; 恢复 ACC 和 PFLAG

                RETI                        ; 中断返回
```

7 I/O 端口

7.1 I/O 端口模式

寄存器 PnM 控制端口的输入输出方向，大部分端口均为双向（输入输出）模式。

0B8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0M	-	-	-	-	-	-	P01M	P00M
读/写	-	-	-	-	-	-	R/W	R/W
复位后	-	-	-	-	-	-	0	0

0C1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1M	P17M	P16M	P15M	P14M	P13M	P12M	P12M	P10M
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

0C2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2M	P27M	P26M	P25M	P24M	P23M	P22M	P22M	P20M
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

0C5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5M	-	-	-	P54M	P53M	P52M	P51M	P50M
读/写	-	-	-	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	0	0	0	0	0

Bit[7:0] **PnM[7:0]:** Pn 模式控制位 (n = 0~5).
0 = Pn 为输入模式.
1 = Pn 为输出模式

✷ 注：
1. **用户可用位操作指令(B0BSET, B0BCLR)对它们进行操作。**
2. **P0.2 是单向输入引脚，故 P0M.2 = 1。**

➲ **例：输入/输出模式选择**

```
CLR     P0M             ; 设置为输入模式
CLR     P2M
CLR     P1M
CLR     P5M

MOV     A, #0FFH        ; 设置为输出模式
B0MOV   P0M, A
B0MOV   P1M, A
B0MOV   P2M,A
B0MOV   P5M, A

B0BCLR  P1M.2           ; 设置 P1.2 为输入模式

B0BSET  P1M.2           ; 设置 P1.2 为输出模式
```

7.2 I/O 上拉电阻寄存器

0E0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0UR	-	-	-	-	-	-	P01R	P00R
读/写	-	-	-	-	-	-	W	W
复位后	-	-	-	-	-	-	0	0

0E1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1UR	P17R	P16R	P15R	P14R	P13R	P12R	P11R	P10R
读/写	W	W	W	W	W	W	W	W
复位后	0	0	0	0	0	0	0	0

0E2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2UR	P27R	P26R	P25R	P24R	P23R	P22R	P21R	P20R
读/写	W	W	W	W	W	W	W	W
复位后	0	0	0	0	0	0	0	0

0E5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5UR	-	-	-	P54R	P53R	P52R	P51R	P50R
读/写	-	-	-	W	W	W	W	W
复位后	-	-	-	0	0	0	0	0

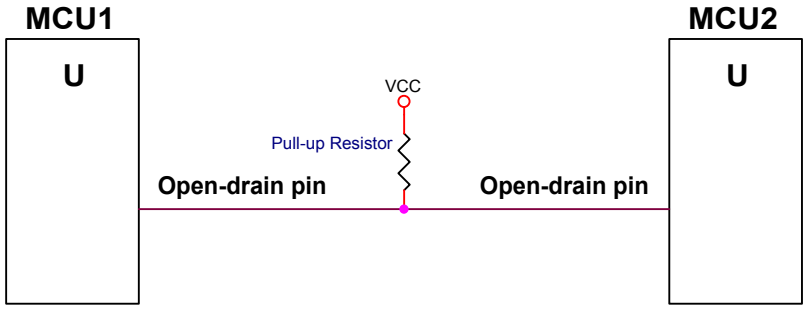
✸ **注：P0.2 是单向输入引脚，无上拉电阻。故 P0UR.2 = 1。**

➲ **例：I/O 上拉电阻寄存器**

```
MOV     A, #0FFH        ; 使能 P0、P1、P2、P5 口的上拉电阻
B0MOV   P0UR, A         ;
B0MOV   P1UR, A
B0MOV   P2UR,A
B0MOV   P5UR, A
```

7.3 I/O 漏极开路寄存器

SN8P2604 的 P1.0、P1.1 具有内置开漏功能，当使能开漏功能时，P1.0、P1.1 必须设置为输出模式。开漏外部电路如下所示：



上拉电阻驱动开漏电路输出高电平，MCU 的引脚则使其输出低电平。

0E9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1OC	-	-	-	-	-	-	P11OC	P10OC
读/写	-	-	-	-	-	-	W	W
复位后	-	-	-	-	-	-	0	0

Bit 0 **P10OC:** P10 开漏控制位
0=禁止漏极开路模式
1=使能漏极开路模式

Bit 1 **P11OC:** P11 开漏控制位
0=禁止漏极开路模式
1=使能漏极开路模式

➲ **例：使能 P1.0 为开漏模式并输出高电平。**

```
B0BSET      P1.0            ;

B0BSET      P10M            ; 使能 P1.0 的输出模式
MOV         A, #01H         ; 使能 P1.0 的开漏功能
B0MOV       P1OC, A
```

✷ **注：P1OC 是只写寄存器，必须用“MOV”指令设置 P10OC。**

➲ **例：禁止 P1.0 的开漏功能，输出低电平。**

```
MOV         A, #0           ; 禁止 P1.0 的开漏功能
B0MOV       P1OC, A
```

✷ **注：禁止 P1.0 的开漏功能后，P1.0 返回到上一个 I/O 模式。**

7.4 I/O 口数据寄存器

0D0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0	-	-	-	-	-	P02	P01	P00
读/写	-	-	-	-	-	R	R/W	R/W
复位后	-	-	-	-	-	0	0	0

0D1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1	P17	P16	P15	P14	P13	P12	P11	P10
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

0D2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2	P27	P26	P25	P24	P23	P22	P21	P20
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

0D5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5	P57	P56	P55	P54	P53	P52	P51	P50
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

✸ **当选择外部复位功能时，P02 保持为“1”**

➲ **例：从输入端读取数据**

```
B0MOV   A, P0       ; 读 P0 口的数据
B0MOV   A, P1       ; 读 P1 口的数据
B0MOV   A, P2       ; 读 P2 口的数据
B0MOV   A, P5       ; 读 P5 口的数据
```

➲ **例：写数据到输出端**

```
MOV     A, #0FFH    ; 所有端口写 FFH
B0MOV   P0, A
B0MOV   P1, A
B0MOV   P2, A
B0MOV   P5, A
```

➲ **例：写 1-bit 的数据到输出端口**

```
B0BSET  P1.3        ; 置 P1.3 和 P5.5 为“1”
B0BSET  P5.5

B0BCLR  P1.3        ; 置 P1.3 和 P5.5 为“0”
B0BCLR  P5.5
```

8 定时器

8.1 看门狗定时器

看门狗定时器(WDT)是一个二进制加 1 计数器，用来监控程序的运行状态，如果程序由于干扰进入未知状态，看门狗定时器将溢出，使微控制器复位。看门狗定时器的时钟由编译选项控制，其时钟源为内部低速振荡器（16K @3V，32K @5V）。

看门狗溢出时间=8192/内部低速振荡器（sec）

VDD	内部低速 RC 频率	看门狗溢出时间
3V	16KHz	512ms
5V	32KHz	256ms

✷ 注：
1. 如果看门狗选择“Always_On”模式，则即使在睡眠模式或绿色模式下，看门狗仍然运行。
2. 在 S8KD ICE 仿真时，必须用宏指令“@RST_WDT”清看门狗定时器，否则 S8KD 看门狗会出错。

看门狗定时器的清零由 WDTR 寄存器控制，写入 **0x5A** 到 WDTR 使看门狗定时器复位。

0CCH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WDTR	WDTR7	WDTR6	WDTR5	WDTR4	WDTR3	WDTR2	WDTR1	WDTR0
读/写	W	W	W	W	W	W	W	W
复位后	0	0	0	0	0	0	0	0

➲ 例：下面是看门狗定时器的操作，在程序的开始将看门狗定时计数器清零。

```
Main:
        MOV     A,#5AH          ; 清看门狗计数器
        B0MOV   WDTR,A
        .       .
        CALL    SUB1
        CALL    SUB2
        .       .
        .       .
        JMP     MAIN
```

➲ 例：用宏指令@RST_WDT 将看门狗定时器清零。

```
Main:
        @RST_WDT                ; 清看门狗计数器
        .       .
        CALL    SUB1
        CALL    SUB2
        .       .
        .       .
        JMP     MAIN
```

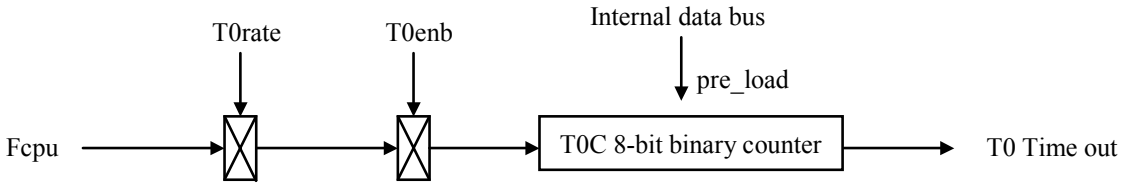
8.2 定时器 T0

8.2.1 概述

基本定时器 T0 是一个 8 位二进制加一计数器，由寄存器 T0M 选择 T0C 的输入时钟。当 T0 溢出 （从 FFH 至 00H）时，产生一个信号触发 T0 中断。T0 基本定时器的功能如下：

T0 的主要功能如下：

- **8 位可编程定时器：**根据所选的时钟频率，定时发出中断请求信号。
- **绿色模式唤醒功能：**T0ENB=1，T0 定时器溢出使系统从绿色模式返回到上一个操作模式。



8.2.2 T0M 模式寄存器

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0M	T0ENB	T0rate2	T0rate1	T0rate0	-	-	-	-
读/写	R/W	R/W	R/W	R/W	-	-	-	-
复位后	0	0	0	0	-	-	-	-

Bit[6:4] **T0RATE2~T0RATE0**：T0 内部时钟选择位。
000 = fcpu/256,
001 = fcpu/128
, …,
110 = fcpu/4,
111 = fcpu/2。

Bit7 **T0ENB**：T0 计数器控制位。
0 = 禁止，
1 = 使能。

8.2.3 T0C 计数寄存器

T0C 是一个 8 位计数寄存器。

0D9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0C	T0C7	T0C6	T0C5	T0C4	T0C3	T0C2	T0C1	T0C0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

T0 基本时间常数表

T0RATE	T0CLOCK	高速模式 Fcpu = 3.58MHz / 4)		低速模式 Fcpu = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间= max/256	最大溢出间隔时间	单步间隔时间= max/256
000	fcpu/256	73.2 ms	286us	8000 ms	31.25 ms
001	fcpu/128	36.6 ms	143us	4000 ms	15.63 ms
010	fcpu/64	18.3 ms	71.5us	2000 ms	7.8 ms
011	fcpu/32	9.15 ms	35.8us	1000 ms	3.9 ms
100	fcpu/16	4.57 ms	17.9us	500 ms	1.95 ms
101	fcpu/8	2.28 ms	8.94us	250 ms	0.98 ms
110	fcpu/4	1.14 ms	4.47us	125 ms	0.49 ms
111	fcpu/2	0.57 ms	2.23us	62.5 ms	0.24 ms

T0C 初始值的计算方法如下：

T0C 初始值=256 -（T0 中断间隔时间×输入时钟）

➲ **例：3.58MHZ 高速模式下，设 TC0 的时间间隔为 10ms。 TC0C (74H) = 256 - (10ms×fcpu/64)**

初始值 = 256 - (TC0 中断间隔时间×输入时钟)

= 256 - (10ms×3.58×10^6÷4÷64)

= 256 - (10^{-2}×3.58×10^6÷4÷64)

= 116

= 74H

8.2.4 T0 操作流程

T0 定时器就是一个定时器中断，设置 T0 的操作流程如下：

☞ **T0 定时器停止计数，禁止 T0 的中断功能并清 T0 的中断请求标志。**

程序		注释
B0BCLR	FT0ENB	T0 定时器停止
B0BCLR	FT0IEN	禁止 T0 中断功能
B0BCLR	FT0IRQ	清 T0 中断请求标志

☞ **设置 T0 定时器的速率。若使能 T0 的 RTC 功能，就不用设置 T0 的速率。**

程序		注释
MOV B0MOV	A, #00H T0M, A	T0 的速率由 T0M 的第 4~6 位控制，范围为 00h~70h

☞ **设置 T0 定时器的功能模式**

程序		注释
B0BSET	FT0IEN	使能 T0 中断功能

☞ **设置 T0 的间隔时间。若使能 T0 的 RTC 功能，则不设置 T0C。**

程序		注释
MOV B0MOV	A, #7FH T0C, A	设置 T0 的间隔时间

☞ **使能 T0 定时器**

程序		注释
B0BSET	FT0ENB	使能 T0 定时器计数

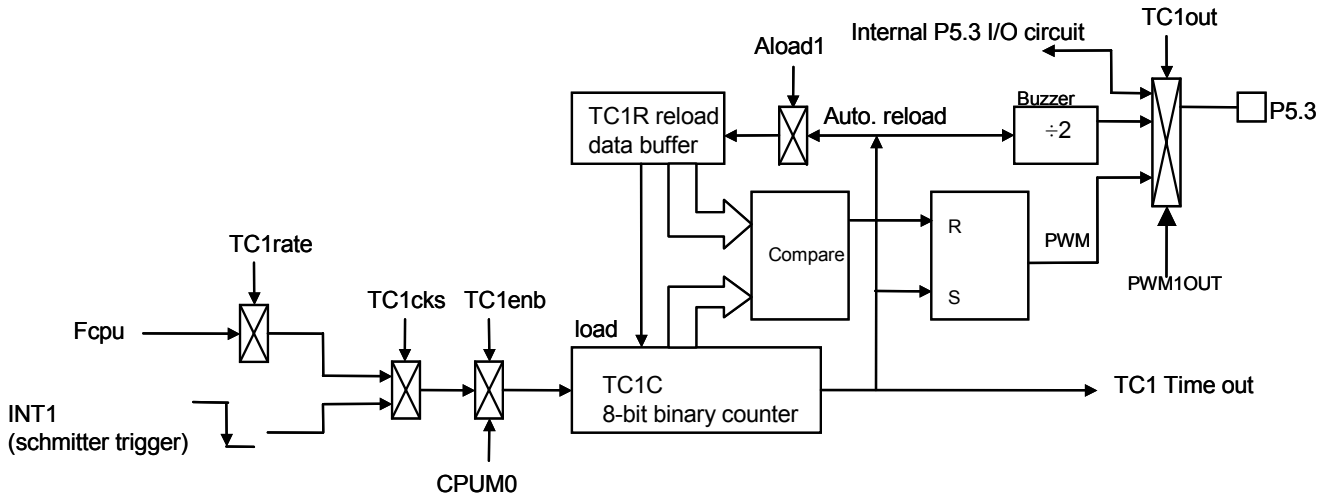
8.3 定时/计数器 TC1

8.3.1 概述

TC1 是一个二进制定时器/计数器，利用 TC1M 寄存器从 Fcpu 或者外部 INT1 引脚（下降沿触发）选择 TC1C 的时钟源，以进行精确的计时/计数。如果 TC1 溢出（从 FFH 到 00H），TC1 将继续计数并产生溢出触发信号，请求 TC1 中断服务。

TC1 定时器的主要功能如下：

- **8 位可编程定时器：**根据设定的时钟频率，产生定时中断。
- **外部事件计数器：**在 INT1 输入引脚端，用外部时钟信号的下降沿记录系统“事件”。
- **BUZZER 输出**
- **PWM 输出**



8.3.2 TC1M 模式寄存器

0DCH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1M	TC1ENB	TC1rate2	TC1rate1	TC1rate0	TC1CKS	ALOAD1	TC1OUT	PWM1OUT
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

Bit0 **PWM1OUT**：PWM 输出控制位。
0 = 禁止 PWM 输出
1 = 使能 PWM 输出，PWM 占空比由 TC1OUT、ALOAD1 控制。

Bit1 **TC1OUT**：TC1 溢出信号输出控制位。**仅在 PWM1OUT=0 时有效。**
0 = 禁止，P5.3 为基本输入/输出端
1 = 使能，P5.3 为 TC1OUT 信号的输出端。

Bit2 **ALOAD1**：自动加载控制位。**仅在 PWM1OUT=0 时有效。**
0 = 禁止
1 = 使能

Bit3 **TC1CKS**：TC1 时钟源选择位。
0 = 内部时钟（Fcpu）
1 = 来自 INT1/P0.1 引脚的外部时钟

Bit[6:4] **TC1RATE2~TC1RATE0**：TC1 内部时钟选择位。
000 = fcpu/256
001 = fcpu/128
…
110 = fcpu/4
111 = fcpu/2

Bit7 **TC1ENB**：TC1 计数器控制位。
0 = 禁止
1 = 使能

✷ 注：当 **TC1CKS=1，TC1 就是一个外部事件计数器，P01IRQ 始终为 0。**

8.3.3 TC1C 计数寄存器

TC1C 是 8 位计数寄存器，用来控制 TC1 的间隔时间。

0DDH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1C	TC1C7	TC1C6	TC1C5	TC1C4	TC1C3	TC1C2	TC1C1	TC1C0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后 0	0	0	0	0	0	0	0	0

定时/计数器 TC1 的间隔时间表

TC1 速率	TC1 时钟	高速模式(fcpu = 3.58MHz)		低速模式(fcpu = 32768Hz)	
		最大溢出间隔时间	单步间隔时间 = max/256	最大溢出间隔时间	单步间隔时间 = max/256
000	fcpu/256	73.2 ms	286us	8000 ms	31.25 ms
001	fcpu/128	36.6 ms	143us	4000 ms	15.63 ms
010	fcpu/64	18.3 ms	71.5us	2000 ms	7.8 ms
011	fcpu/32	9.15 ms	35.8us	1000 ms	3.9 ms
100	fcpu/16	4.57 ms	17.9us	500 ms	1.95 ms
101	fcpu/8	2.28 ms	8.94us	250 ms	0.98 ms
110	fcpu/4	1.14 ms	4.47us	125 ms	0.49 ms
111	fcpu/2	0.57 ms	2.23us	62.5 ms	0.24 ms

表 8-2　TC1 时间表

TC1C 的初始值计算如下：

TC1C 初始值=256-（TC1 中断间隔时间×输入时钟）

➲ **例：3.58MHZ 高速模式下，设 TC1 的时间间隔为 10ms。TC1C (74H) = 256- (10ms×fcpu÷64)。Fcpu=Fosc/4。**

TC1C 初始值 =256- (TC1 中断间隔时间×输入时钟)
=256- (10ms×3.58×10^6÷4÷64)
=256- (10^{-2}×3.58×10^6÷4÷64)
=116
=74H

8.3.4 TC1C 溢出时间

TC1 的溢出时间有两种（PWM 模式和无 PWM 模式）。无 PWM 模式，其溢出边界为 0~255；PWM 模式，溢出边界包括四种(0~16，0~32，0~64，0~255)，由 TC1M 寄存器的 TC1OUT，ALOAD1 位控制。

TC1C 溢出时间列表

PWM1	ALOAD1	TC1OUT	TC1C 有效值	TC1C 的边界类型	备注
0	x	x	0x00~0xFF	00000000b~11111111b	每计数 256 次溢出
1	0	0	0x00~0xFF	00000000b~11111111b	每计数 256 次溢出
1	0	1	0x00~0x3F	xx000000b~xx111111b	每计数 64 次溢出
1	1	0	0x00~0x1F	xxx00000b~xxx11111b	每计数 32 次溢出
1	1	1	0x00~0x0F	xxxx0000b~xxxx1111b	每计数 16 次溢出

8.3.5 TC1R 自动装载寄存器

TC1 的自动装载功能由 TC1M 的 ALOAD1 位控制，当 TC1C 溢出时，TC1R 的值装载到 TC1C 中。易于产生一个精确时间，在中断发生时用户不需重置复位 TC1C。

✷ **注：在 PWM 模式下，自动使能自动装载功能，ALOAD1 位选择溢出的边界。**

0DEH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1R	TC1R7	TC1R6	TC1R5	TC1R4	TC1R3	TC1R2	TC1R1	TC1R0
读/写	W	W	W	W	W	W	W	W
复位后	0	0	0	0	0	0	0	0

TC1R 初始值的计算如下：

TC1R 初始值 = N -（TC1 间隔时间*输入时钟）

➲ **例：设 Fosc=3.58MHz，时间间隔为 10ms。TC1R(74H) = 256- (10ms*fcpu/64)。Fcpu=Fosc/4**

TC0R 初始值 = 256 - (TC0 中断间隔时间 * 输入时钟)

= 256 - (10ms * 3.58 * 10^6 / 4 / 64)

= 256 - (10^{-2} * 3.58 * 10^6 / 4 / 64)

= 116

= 74H

8.3.6 TC1 操作流程

TC1 的操作包括定时器的中断，事件的计数。TC1OUT 和 PWM。TC1 的初始化操作如下：

☞ **TC1 停止计数，禁止 TC1 中断并清 TC1 中断请求标志。**

程序		注释
B0BCLR	FTC1ENB	停止 TC1 定时器，TC1OUT 和 PWM
B0BCLR	FTC1IEN	禁止 TC1 中断功能。
B0BCLR	FTC1IRQ	清 TC1 中断请求标志

☞ **设置 TC1 的速率 (事件计数器模式除外)**

程序		注释
MOV B0MOV	A, #00H TC1M,A	由 TC1M 的第 4～6 位控制 TC1 的速率，其值处于 00h~70h 之间

☞ **设置 TC1 定时功能模式**

程序		注释
B0BSET	FTC1IEN	使能 TC1 中断
B0BSET	FTC1OUT	使能 TC1OUT 功能
B0BSET	FPWM1OUT	使能 PWM 功能

☞ **设置 TC1 定时时钟源**

程序		注释
B0BCLR	FTC1CKS	从系统时钟中选择 TC1 的时钟源
B0BSET	FTC1CKS	从外部事件计数器中选择 TC1 的时钟源

☞ **设置 TC1 的自动装载模式.**

程序		注释
B0BCLR	FALOAD1	禁止自动装载模式
B0BSET	FALOAD1	使能自动装载模式

☞ **设置 TC1 的间隔时间和 PWM 占空比**

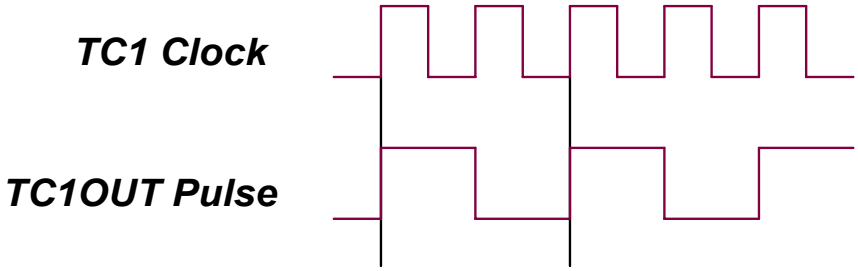
程序		注释
MOV B0MOV B0MOV	A, #7FH TC1C, A TC1R, A	设置 TC1 的间隔时间，并由 TC1C 设置 PWM 的占空比。TC1R 处于自动装载模式（PWM 模式下）
B0BCLR B0BCLR	FTC1OUT FALOAD1	设置 PWM 的占空比为 0~255

☞ **使能 TC1 定时器，TC1OUT 和 PWM 输出**

程序		注释
B0BSET	FTC1ENB	使能 TC1 定时/计数器

8.4 TC1 时钟频率输出(BUZZER 输出)

TC1 定时器/计数器提供了一个频率输出功能。通过设置 TC1 的时钟频率，可以从 P5.3 输出时钟信号，同时 P5.3 的基本输入/输出功会自动屏蔽。TC1 的输出信号是 2 分频的。由于 TC1 时钟源可以有多种选择，相应就可产生多种频率输出，这个功能常用作蜂鸣器输出。



➲ **例：设置 TC1 的 TC1OUT (P5.3)输出，要求外部高速时钟 4MHz，TC1OUT 的频率 0.5KHz。因为 TC1OUT 经过 2 分频 ，因此 TC1 的时钟设为 1KHz。TC1 时钟源来自外部振荡器，T1C 的速率为 Fcpu/4。所以 TC1RATE2~TC1RATE1 = 110。 TC1C = TC1R = 131**

```
MOV       A,#01100000B
B0MOV     TC1M,A          ; 设置 TC1 的分频数为 Fcpu/4

MOV       A,#131          ; 设置自动重新装载的时间常数
B0MOV     TC1C,A
B0MOV     TC1R,A

B0BSET    FTC1OUT         ; 允许 TC1 频率输出（P5.3），同时禁止 P5.3 的 I/O 功能
B0BSET    FALOAD1         ; 允许自动装载功能
B0BSET    FTC1ENB         ; TC1 开始计数
```

✸ **注：使能 buzzer 输出，“PWM1OUT”必须为 0。**

8.4.1 TC1OUT 频率表

Fosc = 4MHz, TC1 Rate = Fcpu/8

TC1R	TC1OUT (KHz)	TC1R	TC1OUT (KHz)	TC1R	TC1OUT (KHz)	TC1R	TC1OUT (KHz)	TC1R	TC1OUT (KHz)
0	0.2441	56	0.3125	112	0.4340	168	0.7102	224	1.9531
1	0.2451	57	0.3141	113	0.4371	169	0.7184	225	2.0161
2	0.2461	58	0.3157	114	0.4401	170	0.7267	226	2.0833
3	0.2470	59	0.3173	115	0.4433	171	0.7353	227	2.1552
4	0.2480	60	0.3189	116	0.4464	172	0.7440	228	2.2321
5	0.2490	61	0.3205	117	0.4496	173	0.7530	229	2.3148
6	0.2500	62	0.3222	118	0.4529	174	0.7622	230	2.4038
7	0.2510	63	0.3238	119	0.4562	175	0.7716	231	2.5000
8	0.2520	64	0.3255	120	0.4596	176	0.7813	232	2.6042
9	0.2530	65	0.3272	121	0.4630	177	0.7911	233	2.7174
10	0.2541	66	0.3289	122	0.4664	178	0.8013	234	2.8409
11	0.2551	67	0.3307	123	0.4699	179	0.8117	235	2.9762
12	0.2561	68	0.3324	124	0.4735	180	0.8224	236	3.1250
13	0.2572	69	0.3342	125	0.4771	181	0.8333	237	3.2895
14	0.2583	70	0.3360	126	0.4808	182	0.8446	238	3.4722
15	0.2593	71	0.3378	127	0.4845	183	0.8562	239	3.6765
16	0.2604	72	0.3397	128	0.4883	184	0.8681	240	3.9063
17	0.2615	73	0.3415	129	0.4921	185	0.8803	241	4.1667
18	0.2626	74	0.3434	130	0.4960	186	0.8929	242	4.4643
19	0.2637	75	0.3453	131	0.5000	187	0.9058	243	4.8077
20	0.2648	76	0.3472	132	0.5040	188	0.9191	244	5.2083
21	0.2660	77	0.3492	133	0.5081	189	0.9328	245	5.6818
22	0.2671	78	0.3511	134	0.5123	190	0.9470	246	6.2500
23	0.2682	79	0.3531	135	0.5165	191	0.9615	247	6.9444
24	0.2694	80	0.3551	136	0.5208	192	0.9766	248	7.8125
25	0.2706	81	0.3571	137	0.5252	193	0.9921	249	8.9286
26	0.2717	82	0.3592	138	0.5297	194	1.0081	250	10.4167
27	0.2729	83	0.3613	139	0.5342	195	1.0246	251	12.5000
28	0.2741	84	0.3634	140	0.5388	196	1.0417	252	15.6250
29	0.2753	85	0.3655	141	0.5435	197	1.0593	253	20.8333
30	0.2765	86	0.3676	142	0.5482	198	1.0776	254	31.2500
31	0.2778	87	0.3698	143	0.5531	199	1.0965	255	62.5000
32	0.2790	88	0.3720	144	0.5580	200	1.1161		
33	0.2803	89	0.3743	145	0.5631	201	1.1364		
34	0.2815	90	0.3765	146	0.5682	202	1.1574		
35	0.2828	91	0.3788	147	0.5734	203	1.1792		
36	0.2841	92	0.3811	148	0.5787	204	1.2019		
37	0.2854	93	0.3834	149	0.5841	205	1.2255		
38	0.2867	94	0.3858	150	0.5896	206	1.2500		
39	0.2880	95	0.3882	151	0.5952	207	1.2755		
40	0.2894	96	0.3906	152	0.6010	208	1.3021		
41	0.2907	97	0.3931	153	0.6068	209	1.3298		
42	0.2921	98	0.3956	154	0.6127	210	1.3587		
43	0.2934	99	0.3981	155	0.6188	211	1.3889		
44	0.2948	100	0.4006	156	0.6250	212	1.4205		
45	0.2962	101	0.4032	157	0.6313	213	1.4535		
46	0.2976	102	0.4058	158	0.6378	214	1.4881		
47	0.2990	103	0.4085	159	0.6443	215	1.5244		
48	0.3005	104	0.4112	160	0.6510	216	1.5625		
49	0.3019	105	0.4139	161	0.6579	217	1.6026		
50	0.3034	106	0.4167	162	0.6649	218	1.6447		
51	0.3049	107	0.4195	163	0.6720	219	1.6892		
52	0.3064	108	0.4223	164	0.6793	220	1.7361		
53	0.3079	109	0.4252	165	0.6868	221	1.7857		
54	0.3094	110	0.4281	166	0.6944	222	1.8382		
55	0.3109	111	0.4310	167	0.7022	223	1.8939		

Fosc = 16MHz, TC1 Rate = Fcpu/8

TC1R	TC1OUT (KHz)	TC1R	TC1OUT (KHz)	TC1R	TC1OUT (KHz)	TC1R	TC1OUT (KHz)	TC1R	TC1OUT (KHz)
0	0.9766	56	1.2500	112	1.7361	168	2.8409	224	7.8125
1	0.9804	57	1.2563	113	1.7483	169	2.8736	225	8.0645
2	0.9843	58	1.2626	114	1.7606	170	2.9070	226	8.3333
3	0.9881	59	1.2690	115	1.7730	171	2.9412	227	8.6207
4	0.9921	60	1.2755	116	1.7857	172	2.9762	228	8.9286
5	0.9960	61	1.2821	117	1.7986	173	3.0120	229	9.2593
6	1.0000	62	1.2887	118	1.8116	174	3.0488	230	9.6154
7	1.0040	63	1.2953	119	1.8248	175	3.0864	231	10.0000
8	1.0081	64	1.3021	120	1.8382	176	3.1250	232	10.4167
9	1.0121	65	1.3089	121	1.8519	177	3.1646	233	10.8696
10	1.0163	66	1.3158	122	1.8657	178	3.2051	234	11.3636
11	1.0204	67	1.3228	123	1.8797	179	3.2468	235	11.9048
12	1.0246	68	1.3298	124	1.8939	180	3.2895	236	12.5000
13	1.0288	69	1.3369	125	1.9084	181	3.3333	237	13.1579
14	1.0331	70	1.3441	126	1.9231	182	3.3784	238	13.8889
15	1.0373	71	1.3514	127	1.9380	183	3.4247	239	14.7059
16	1.0417	72	1.3587	128	1.9531	184	3.4722	240	15.6250
17	1.0460	73	1.3661	129	1.9685	185	3.5211	241	16.6667
18	1.0504	74	1.3736	130	1.9841	186	3.5714	242	17.8571
19	1.0549	75	1.3812	131	2.0000	187	3.6232	243	19.2308
20	1.0593	76	1.3889	132	2.0161	188	3.6765	244	20.8333
21	1.0638	77	1.3966	133	2.0325	189	3.7313	245	22.7273
22	1.0684	78	1.4045	134	2.0492	190	3.7879	246	25.0000
23	1.0730	79	1.4124	135	2.0661	191	3.8462	247	27.7778
24	1.0776	80	1.4205	136	2.0833	192	3.9063	248	31.2500
25	1.0823	81	1.4286	137	2.1008	193	3.9683	249	35.7143
26	1.0870	82	1.4368	138	2.1186	194	4.0323	250	41.6667
27	1.0917	83	1.4451	139	2.1368	195	4.0984	251	50.0000
28	1.0965	84	1.4535	140	2.1552	196	4.1667	252	62.5000
29	1.1013	85	1.4620	141	2.1739	197	4.2373	253	83.3333
30	1.1062	86	1.4706	142	2.1930	198	4.3103	254	125.0000
31	1.1111	87	1.4793	143	2.2124	199	4.3860	255	250.0000
32	1.1161	88	1.4881	144	2.2321	200	4.4643		
33	1.1211	89	1.4970	145	2.2523	201	4.5455		
34	1.1261	90	1.5060	146	2.2727	202	4.6296		
35	1.1312	91	1.5152	147	2.2936	203	4.7170		
36	1.1364	92	1.5244	148	2.3148	204	4.8077		
37	1.1416	93	1.5337	149	2.3364	205	4.9020		
38	1.1468	94	1.5432	150	2.3585	206	5.0000		
39	1.1521	95	1.5528	151	2.3810	207	5.1020		
40	1.1574	96	1.5625	152	2.4038	208	5.2083		
41	1.1628	97	1.5723	153	2.4272	209	5.3191		
42	1.1682	98	1.5823	154	2.4510	210	5.4348		
43	1.1737	99	1.5924	155	2.4752	211	5.5556		
44	1.1792	100	1.6026	156	2.5000	212	5.6818		
45	1.1848	101	1.6129	157	2.5253	213	5.8140		
46	1.1905	102	1.6234	158	2.5510	214	5.9524		
47	1.1962	103	1.6340	159	2.5773	215	6.0976		
48	1.2019	104	1.6447	160	2.6042	216	6.2500		
49	1.2077	105	1.6556	161	2.6316	217	6.4103		
50	1.2136	106	1.6667	162	2.6596	218	6.5789		
51	1.2195	107	1.6779	163	2.6882	219	6.7568		
52	1.2255	108	1.6892	164	2.7174	220	6.9444		
53	1.2315	109	1.7007	165	2.7473	221	7.1429		
54	1.2376	110	1.7123	166	2.7778	222	7.3529		
55	1.2438	111	1.7241	167	2.8090	223	7.5758		

8.5 PWM 功能说明

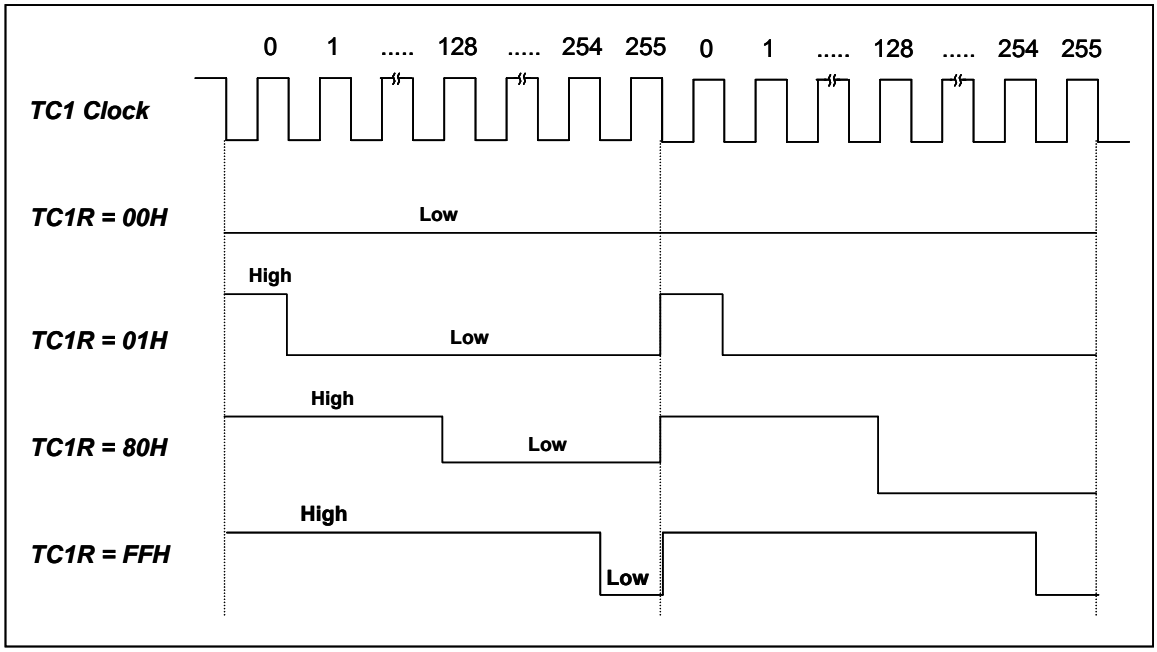
8.5.1 概述

PWM 功能使用的时基为 TC1，产生的 PWM 信号通过 PWM1OUT（P5.3）输出。8 位定时计数器的计数系数为 256、64、32、16，是由 ALOAD1、TC1OUT 位控制的。8 位计数器的值与 TC1R 的值相比较：当 TC1R 的值和 TC0C 的值相等时，PWM 输出低电平；当计数器的值回到 0 时，PWM 被迫输出高电平。PWM 输出的低高电平的比例（占空比）为：TC1R/256、64、32、16。

不断地向参考寄存器写入 00H 可以使 PWM 输出保持为低电平。在 PWM 运行时可以改变 PWM 的占空比来调整 TC1R。

ALOAD1	TC1OUT	PWM 占空比范围	TC1C 有效值	TC1R 有效位	最大 PWM 频率 (Fcpu = 4MHz)	备注
0	0	0/255~255/255	0x00~0xFF	0x00~0xFF	7.8125K	每计数 256 次溢出
0	1	0/63~63/63	0x00~0x3F	0x00~0x3F	31.25K	每计数 64 次溢出
1	0	0/31~31/31	0x00~0x1F	0x00~0x1F	62.5K	每计数 32 次溢出
1	1	0/15~15/15	0x00~0x0F	0x00~0x0F	125K	每计数 16 次溢出

PWM 的输出占空比和 TC1R 是不同的，占空比的范围是 0/255~255/255。



8.5.2 PWM 操作举例

➲ **例：设置 PWM1 的输出 PWM1OUT (P5.3)，其外部高速振荡器时钟 4MHz，PWM 输出占空比为 30/256。PWM 的输出频率是 1KHz。PWM 的时钟源来自外部振荡器，TC1 的速率是 Fcpu/4。TC1RATE2~TC1RATE1 =110。TC0C = TC0R = 30**

```
MOV     A,#01100000B
B0MOV   TC1M,A          ; 设置 TC1 的分频数为 Fcpu/4

MOV     A,#30           ; 设置 PWM 的占空比为 30/256
B0MOV   TC1C, A
B0MOV   TC1R, A

B0BCLR  FTC1OUT         ; 设置占空比范围为 0/255~255/255
B0BCLR  FALOAD1
B0BSET  FPWM1OUT        ; 使能 PWM1 输出到 P5.3 并禁止 P5.3 的输入输出功能
B0BSET  FTC1ENB         ;  TC1 开始计数
```

✸ **注：TC1R 是只写寄存器，不能执行 INCMS，DECMS 指令。**

➲ **例：调整 TC1R 的值**

```
MOV     A, #30H         ; 赋予一个立即数
B0MOV   TC1R, A

INCMS   BUF0            ; 从 BUF0 得到一个新的 TC1R 值
B0MOV   A, BUF0
B0MOV   TC1R, A
```

✸ **注：**

1.PWM1 占空比改变时，最好同时改变 TC1C 和 TC1R 的值，以避免 PWM1 信号受到干扰。
2.PWM 功能和中断可同时使用。

9 指令表

Field	Mnemonic		Description	C	DC	Z	Cycle
MOVE	MOV	A,M	A ← M	-	-	√	1
	MOV	M,A	M ← A	-	-	-	1
	B0MOV	A,M	A ← M (bank 0)	-	-	√	1
	B0MOV	M,A	M (bank 0) ← A	-	-	-	1
	MOV	A,I	A ← I	-	-	-	1
	B0MOV	M,I	M ← I, (M = only for Working registers R, Y, Z , RBANK & PFLAG)	-	-	-	1
	XCH	A,M	A ←→M	-	-	-	1+N
	B0XCH	A,M	A ← → M (bank 0)	-	-	-	1+N
	MOVC		R, A ← ROM [Y,Z]	-	-	-	2
ARITHMETIC	ADC	A,M	A ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1
	ADC	M,A	M ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1+N
	ADD	A,M	A ← A + M, if occur carry, then C=1, else C=0	√	√	√	1
	ADD	M,A	M ← A + M, if occur carry, then C=1, else C=0	√	√	√	1+N
	B0ADD	M,A	M (bank 0) ← M (bank 0) + A, if occur carry, then C=1, else C=0	√	√	√	1+N
	ADD	A,I	A ← A + I, if occur carry, then C=1, else C=0	√	√	√	1
	SBC	A,M	A ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1
	SBC	M,A	M ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1+N
	SUB	A,M	A ← A - M, if occur borrow, then C=0, else C=1	√	√	√	1
	SUB	M,A	M ← A - M, if occur borrow, then C=0, else C=1	√	√	√	1+N
	SUB	A,I	A ← A - I, if occur borrow, then C=0, else C=1	√	√	√	1
	DAA		To adjust ACC's data format from HEX to DEC.	√	-	-	1
LOGIC	AND	A,M	A ← A and M	-	-	√	1
	AND	M,A	M ← A and M	-	-	√	1+N
	AND	A,I	A ← A and I	-	-	√	1
	OR	A,M	A ← A or M	-	-	√	1
	OR	M,A	M ← A or M	-	-	√	1+N
	OR	A,I	A ← A or I	-	-	√	1
	XOR	A,M	A ← A xor M	-	-	√	1
	XOR	M,A	M ← A xor M	-	-	√	1+N
	XOR	A,I	A ← A xor I	-	-	√	1
PROCESS	SWAP	M	A (b3~b0, b7~b4) ← M(b7~b4, b3~b0)	-	-	-	1
	SWAPM	M	M(b3~b0, b7~b4) ← M(b7~b4, b3~b0)	-	-	-	1+N
	RRC	M	A ← RRC M	√	-	-	1
	RRCM	M	M ← RRC M	√	-	-	1+N
	RLC	M	A ← RLC M	√	-	-	1
	RLCM	M	M ← RLC M	√	-	-	1+N
	CLR	M	M ← 0	-	-	-	1
	BCLR	M.b	M.b ← 0	-	-	-	1+N
	BSET	M.b	M.b ← 1	-	-	-	1+N
	B0BCLR	M.b	M(bank 0).b ← 0	-	-	-	1+N
	B0BSET	M.b	M(bank 0).b ← 1	-	-	-	1+N
BRANCH	CMPRS	A,I	ZF,C ← A - I, If A = I, then skip next instruction	√	-	√	1 + S
	CMPRS	A,M	ZF,C ← A – M, If A = M, then skip next instruction	√	-	√	1 + S
	INCS	M	A ← M + 1, If A = 0, then skip next instruction	-	-	-	1+ S
	INCMS	M	M ← M + 1, If M = 0, then skip next instruction	-	-	-	1+N+S
	DECS	M	A ← M - 1, If A = 0, then skip next instruction	-	-	-	1+ S
	DECMS	M	M ← M - 1, If M = 0, then skip next instruction	-	-	-	1+N+S
	BTS0	M.b	If M.b = 0, then skip next instruction	-	-	-	1 + S
	BTS1	M.b	If M.b = 1, then skip next instruction	-	-	-	1 + S
	B0BTS0	M.b	If M(bank 0).b = 0, then skip next instruction	-	-	-	1 + S
	B0BTS1	M.b	If M(bank 0).b = 1, then skip next instruction	-	-	-	1 + S
	JMP	d	PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-	2
	CALL	d	Stack ← PC15~PC0, PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-	2
MISC	RET		PC ← Stack	-	-	-	2
	RETI		PC ← Stack, and to enable global interrupt	-	-	-	2
	PUSH		To push ACC and PFLAG into buffers.	-	-	-	1
	POP		To pop ACC and PFLAG from buffers.	√	√	√	1
	NOP		No operation	-	-	-	1

✸ 注：1. "M"代表寄存器和存储器。若"M"是系统寄存器，则"N"=0，否则"N"=1。
2. 若满足跳转指令的条件，则"S = 0"，否则"S = 1"。
3. OSCM 的任何一条读/写指令都不加特别的时钟周期。
4. "B0XCH"指令不支持特殊寄存器（0x80~0xFF）。
5. "B0M0V M, I"中的"I"不能是"0xE6"和"0xE7"。

10 电气特性

10.1 极限参数

电源电压(Vdd)..... - 0.3V ~ 6.0V
输入电压(Vin)..... Vss – 0.2V ~ Vdd + 0.2V
工作温度(Topr)
SN8P2604K，SN8P2604S，SN8P2604X..... -20(C ~ + 70(C
SN8P2604KD，SN8P2604SD，SN8P2604XD..... -40(C ~ + 85(C
存储温度(Tstor) ..... –30°C ~ + 125°C
功耗(Pc)..... 500mW

10.2 电气特性

（所有电压以 Vss, Vdd=5.0V 为参考值， fosc=3.579545MHz，环境温度为 25℃）

参数	符号	说明		最小值	标准值	最大值	单位
工作电压	Vdd	普通模式, Vpp = Vdd		2.4	5.0	5.5	V
		编程模式, Vpp = 12.3V		-	5.0	-	
OTP 编程电压	Vpp	OTP 编程电压		-	12.3	-	V
RAM 数据保持电压	Vdr			-	1.5	-	V
内部 POR	Vpor	Vdd 的上升速率，以确保内部的上电复位		-	0.05	-	V/ms
输入低电压	ViL1	施密特触发的输入端		Vss	-	0.2Vdd	V
输入高电压	ViH1	施密特触发的输入端		0.8Vdd	-	Vdd	V
复位引脚漏电流	Ilekg	Vin = Vdd		-	-	2	uA
I/O 口的上拉电阻	Rup	Vin = Vss , Vdd = 5V		-	100	-	K(
I/O 口的输入漏电流	Ilekg	禁止上拉电阻，Vin = Vdd		-	-	1	uA
所有端口的源电流	IoH	Vop = Vdd – 0.5V		-	12	-	mA
灌电流	IoL	Vop = Vss + 0.5V		-	15	-	
INTn 触发脉冲宽度	Tintn	INTn 中断请求脉冲宽度		2/fcpu	-	-	cycle
电源电流	Idd1	运行模式（无负载，Fcpu=Fosc/4 禁止低功耗功能)	Vdd= 5V 4Mhz	-	2.5	-	mA
			Vdd= 3V 4Mhz	-	1	-	mA
			Vdd= 5V 32768Hz	-	70	-	uA
			Vdd= 3V 32768Hz	-	15	-	uA
	Idd2	低速模式（内部 RC 模式)	Vdd= 5V 32Khz	-	25	-	uA
			Vdd= 3V 32Khz	-	5	-	uA
	Idd3	睡眠模式	Vdd= 5V	-	0.80	-	uA
			Vdd= 3V	-	0.70	-	uA
	Idd4	绿色模式（无负载，Fcpu=Fosc/4 禁止低功耗）	Vdd= 5V 4Mhz	-	0.60	-	mA
			Vdd= 3V 4Mhz	-	0.25	-	mA
			Vdd= 5V 32768Hz	-	70	-	uA
			Vdd= 3V 32768Hz	-	10	-	uA
			Vdd= 5V ILRC 32Khz	-	15	-	uZ
			Vdd= 3V ILRC 16Khz	-	3	-	uA
内部振荡器频率	Filrc	内部低速 RC(ILRC)	Vdd= 3V	-	16	-	Khz
LVD 侦测电平	VLVD	低电压侦测电平		-	1.8	1.8	V

11 应用注意事项

11.1 开发工具版本

11.1.1 ICE(在线仿真器)

- **S8KD-2 ICE**：S8KD-2 ICE 是用来进行 SN8P1XXX 系列的仿真，在仿真 SN8P2604 时会有所限制。请参照下面的 S8KD-2 ICE 仿真注意事项。
- **SN8ICE 2K**：全功能仿真 SN8P2604。

11.1.2 OTP 烧录器

- **Writer3.0**：支持 SN8P2604，但是没有脱机模式。
- **Easy Writer V1.0**：由 ICE 控制 OTP 编程，且没有升级硬件的烦恼。详见 easy writer 的用户手册。
- **MP-Easy Writer V1.0**：可脱机烧录，支持 SN8P2604 的大规模烧录。

11.1.3 SN8IDE

SONIX 8 位 MCU 完整的开发环境包括编译器，ICE 调试器和 OTP 烧录软件。

- **S8KD-2 ICE**：SN8IDE V1.99M 或以后的版本。
- **S8KE ICE**：SN8IDE V2.00 或以后的版本。
- **Writer3.0 和 Easy Writer**：SN8IDE V1.99M 或以后的版本。

11.2 编译选项(CODE OPTION)

11.2.1 FCPU 编译选项

SN8P2604 是一个 1T 系统 MCU，一条指令周期（Fcpu）等于系统时钟（Fosc）。对于不同的应用环境，用户可以由 Fcpu 编译选项选择不同的指令周期，选择范围为 Fosc/1~Fosc/8。针对不同的应用，编译器会自动的限制 Fcpu 的范围，如下所示：

Fcpu		Noise_Filter	
		Enable	Disable
Low_Power	Enable	**Fosc/4~Fosc/8**	**Fosc/4~Fosc/8**
	Disable	**Fosc/4~Fosc/8**	**Fosc/1~Fosc/8**

11.2.2 NOISE FILTER 编译选项

在 AC 等高干扰环境下应该使能 Noise_Filter，S8ASM 编译器强迫禁止 Low_Power 选项。使能 Noise_Filter 可以减少外部噪音对操作系统的影响。**若使能 Noise_Filter，Fcpu 会限制在 Fosc/4~Fosc/8 之间。**

11.2.3 WATCHDOG

SN8P2604 的 Watchdog 有 3 个选项：Enable、Disable 和 Always ON。

- **Watchdog Enable**：在程序执行的过程中，若看门狗溢出，则系统复位。看门狗定时器在绿色模式和省电（睡眠）模式下停止运行，系统唤醒后，看门狗定时器重新运行。
- **Watchdog Always On**：看门狗定时器在绿色模式和省电（睡眠）模式下仍然运行，若看门狗在这两种模式下溢出，系统复位。此选项的主要目是为了确保看门狗定时器能够在高干扰环境下保护系统有良好的性能。

11.3 中断向量(ORG 8)

当有中断发生时，系统跳到中断向量执行中断服务程序。中断向量（ORG 8）的第一条指令必须是“JMP”或“NOP”，如果第一条指令不是“JMP”或“NOP”，SN8ASM199L 和以后版本的编译器会发出报警信号。

➲ **例：中断服务程序位于 ORG 8 后。**

```
.CODE
                ORG     0               ; 0000H
                JMP     START           ; 跳到用户程序
                …                       ; 0004H ~ 0007H 系统保留

                ORG     8               ; 中断服务程序
                NOP                     ; ORG 8 处的第一条指令

                …
                RETI                    ; 中断返回
                …
START:                                  ; 用户程序的首地址
                .                       ; 用户程序
                .
                JMP     START           ; 用户程序结束
                …
                ENDP                    ; 程序结束
```

➲ **例：中断服务程序位于用户程序之后。**

```
.CODE
                ORG     0               ; 0000H
                JMP     START           ; 跳到用户程序
                …                       ; 0001H ~ 0007H 系统保留

                ORG     08
                JMP     MY_IRQ          ; 0008H，跳到中断服务程序

                ORG     10H
START:                                  ; 0010H，用户程序的首地址
                .                       ; 用户程序
                .
                .
                JMP     START           ; 用户程序结束
                …
MY_IRQ:                                 ; 中断服务程序首地址
                …

                RETI                    ; 中断返回
                …

                ENDP                    ; 程序结束
```

11.4 指令

11.4.1 B0MOV M, I

"B0MOV M, I"中的"M"指工作寄存器（Y、Z、R、PFLAG）(0x80~0x86)，"I"是立即数。

✷ **注："B0MOV M, I"中的"I"不能是"E6H"和"E7H"，若 I 等于 E6H 或 E7H，S8ASM V1.99K 和以后版本的编译器会报警并将错误信息显示在屏幕上。**

用户需要检查"B0MOV M, I"的值，这条指令常用在查表功能中设置表格的地址，在编译后可以很容易地从列表文件（.LST）中检查表格的地址。用户也可以使用 ORG 设置表格的首地址避开"E6H"和"E7H"。

➲ **例：通过 ORG 设置表格地址。**

```
        ORG     0x0100          ; 设置表格的地址从 0x0100 开始
Table:
        DW      0x9876
        …
```

11.4.2 B0XCH A, M

"B0XCH A, M"指令交换 A 和 M 的内容，仅支持用户自定义的 RAM。

✷ **注："M"不能是系统寄存器的地址（0x80~0xFF）**

➲ **例：保存 PFLAG**

Error case!

```
        B0XCH   A, PFLAG        ; PFLAG 是系统寄存器，"B0XCH A, M"不能访问 PFLAG
        B0MOV   PFLAGBUF, A
```

Correct case!

```
        B0MOV   A, PFLAG
        B0MOV   PFLAGBUF, A     ; PFLAGBUF 是用户自定义的 RAM
```

➲ **例：保存 ACC**

```
ACCBUF  DS      1               ; 定义 ACCBUF 保存 ACC
        ORG     0
        …
        ORG     10
        …
        B0XCH   A, ACCBUF
        …
```

11.5 S8KD-2 ICE 仿真

S8IDE 是 SONIX 8 位开发软件，包括有编译器/ICE /OTP Writer。S8KD-2 是 SONIX 的 8 位 ICE EV 芯片，SN8P2604 和 EV 芯片特性并不完全一样。SONIX 提供了宏指令用来解决这些差异，以保证在仿真时特性的一致性。S8ASM V1.99M 和以后的版本将支持上述的宏指令。

✸ **注：请使用 S8ASM V1.99K 或以后的版本来开发新项目。**

11.5.1 ICE_MODE

设置 ICE_MODE 是必须的，ICE_MODE=1 支持 S8KD-2 ICE 仿真，ICE_MODE=0 是实际芯片模式。

语法：ICE_MODE Val
Val：0=实际芯片，1=S8KD-2 ICE 仿真

➲ **例：“ICE_MODE EQU 1”用来进行 ICE 仿真。不要使用此时的 SN8 文件来烧录实际的 OTP 芯片。**

```
CHIP        SN8P2604
.DATA

ICE_MODE    EQU         1           ; 设置 ICE_MODE 为 ICE 仿真模式

            INCLUDESTD  SN8P2X_ICE.H
.CODE
            User program
            …
```

➲ **例：“ICE_MODE EQU 0”用来生成 OTP 烧录文档，在此模式中，S8KD-2 ICE 不能正确地仿真 SN8P270XA。**

```
CHIP        SN8P2604
.DATA

ICE_MODE    EQU         0           ; 设置 ICE_MODE 为实际芯片模式

            INCLUDESTD  SN8P2X_ICE.H
.CODE
            User program
            …
```

✸ **注：1. 设置 ICE_MODE = 0，ICE 仿真和校验所有的功能后，在烧录实际 OTP 芯片前要重新编译源代码。**
2. 请使用 ICE_MODE = 0 时的 CHECKSUM 值作为实际芯片的 CHECKSUM 值，而不要使用 ICE_MODE = 1 下的 CHECKSUM，这只是仿真时的 CHECKSUM 值。

11.5.2 指令周期

SN8P2604 和 EV 芯片的某些指令的周期是不同的，这些差异使二者的指令时序不完全一致。SN8IDE 编译器提供了一些宏指令以解决此问题，用户只需用内置的宏指令取代相应的指令。在仿真模式里，编译器会插入一些额外的代码使二者指令时序相同，因此，ICE 中可用的 ROM 的最大容量要大于实际芯片，而在实际芯片模式中，它们的容量是相同的。

SN8P2604				S8KD-2 EV CHIP				INSTRUCTION MACRO	DESCRIPTION
Field	Mnemonic		Cycle	Field	Mnemonic		Cycle		
	MOV	A,M	1		MOV	A,M	1	-	1. M = RAM, N = 0. M = system register, N = 1. 2. S8KD-2 ICE: Read OSCM = 1 cycle Write OSCM = 2 cycle SN8P2604: Read OSCM = 1 cycle Write OSCM = 1 cycle 3. PUSH,POP instructions are different between SN8P2604 and S8KD-2 ICE.
M	MOV	M,A	1	M	MOV	M,A	1	-	
O	B0MOV	A,M	1	O	B0MOV	A,M	1	-	
V	B0MOV	M,A	1	V	B0MOV	M,A	1	-	
E	MOV	A,I	1	E	MOV	A,I	1	-	
	B0MOV	M,I	1		B0MOV	M,I	1	-	
	XCH	**A,M**	**1+N**		**XCH**	**A,M**	**1**	**@XCH A,M**	
	B0XCH	**A,M**	**1+N**		**B0XCH**	**A,M**	**1**	**@B0XCH A,M**	
	MOVC		2		MOVC		2	-	
	ADC	A,M	1		ADC	A,M	1	-	
A	**ADC**	**M,A**	**1+A**	A	**ADC**	**M,A**	**1**	**@ADC M,A**	
R	ADD	A,M	1	R	ADD	A,M	1	-	
I	**ADD**	**M,A**	**1+N**	I	**ADD**	**M,A**	**1**	**@ADD M,A**	
T	**B0ADD**	**M,A**	**1+N**	T	**B0ADD**	**M,A**	**1**	**@B0ADD M,A**	
H	ADD	A,I	1	H	ADD	A,I	1	-	
M	SBC	A,M	1	M	SBC	A,M	1	-	
E	**SBC**	**M,A**	**1+N**	E	**SBC**	**M,A**	**1**	**@SBC M,A**	
T	SUB	A,M	1	T	SUB	A,M	1	-	
I	**SUB**	**M,A**	**1+N**	I	**SUB**	**M,A**	**1**	**@SUB M,A**	
C	SUB	A,I	1	C	SUB	A,I	1	-	
	AND	A,M	1		AND	A,M	1	-	
L	**AND**	**M,A**	**1+N**	L	**AND**	**M,A**	**1**	**@AND M,A**	
O	AND	A,I	1	O	AND	A,I	1	-	
G	OR	A,M	1	G	OR	A,M	1	-	
I	**OR**	**M,A**	**1+N**	I	**OR**	**M,A**	**1**	**@OR M,A**	
C	OR	A,I	1	C	OR	A,I	1	-	
	XOR	A,M	1		XOR	A,M	1	-	
	XOR	**M,A**	**1+N**		**XOR**	**M,A**	**1**	**@XOR M,A**	
	XOR	A,I	1		XOR	A,I	1	-	
	SWAP	M	1		SWAP	M	1	-	
P	**SWAPM**	**M**	**1+N**	P	**SWAPM**	**M**	**1**	**@SWAPM M**	
R	RRC	M	1	R	RRC	M	1	-	
O	**RRCM**	**M**	**1+N**	O	**RRCM**	**M**	**1**	**@RRCM M**	
C	RLC	M	1	C	RLC	M	1	-	
E	**RLCM**	**M**	**1+N**	E	**RLCM**	**M**	**1**	**@RLCM M**	
S	CLR	M	1	S	CLR	M	1	-	
S	**BCLR**	**M.b**	**1+N**	S	**BCLR**	**M.b**	**1**	**@BSET M.b**	
	BSET	**M.b**	**1+N**		**BSET**	**M.b**	**1**	**@BCLR M.b**	
	B0BCLR	**M.b**	**1+N**		**B0BCLR**	**M.b**	**1**	**@B0BSET M.b**	
	B0BSET	**M.b**	**1+N**		**B0BSET**	**M.b**	**1**	**@B0BCLR M.b**	
	CMPRS	A,I	1 + S		CMPRS	A,I	1 + S	-	
B	CMPRS	A,M	1 + S	B	CMPRS	A,M	1 + S	-	
R	INCS	M	1 + S	R	INCS	M	1 + S	-	
A	**INCMS**	**M**	**1+N+S**	A	**INCMS**	**M**	**1 + S**	**@INCMS M**	
N	DECS	M	1 + S	N	DECS	M	1 + S	-	
C	**DECMS**	**M**	**1+N+S**	C	**DECMS**	**M**	**1 + S**	**@DECMS M**	
H	BTS0	M.b	1 + S	H	BTS0	M.b	1 + S	-	
	BTS1	M.b	1 + S		BTS1	M.b	1 + S	-	
	B0BTS0	M.b	1 + S		B0BTS0	M.b	1 + S	-	
	B0BTS1	M.b	1 + S		B0BTS1	M.b	1 + S	-	
M	JMP	d	2	M	JMP	d	2	-	
I	CALL	d	2	I	CALL	d	2	-	
S	RET		2	S	RET		2	-	
C	RETI		2	C	RETI		2	-	
	NOP		1		NOP		1	-	
	PUSH		1		PUSH		1		
	POP		1		POP		1		

✷ 注：**S8KD-2 ICE 不能仿真 SN8P2604 的“PUSH”、“POP”指令。**

✷ 注：上表中的宏指令内置在**“SN8P2X_ICE.H”**，此文件必须包含在用户程序中。

➲ 例：在用户程序中包含 **SN8P2X_ICE.H**

```
CHIP        SN8P2604
.DATA
ICE_MODE    EQU             0

            INCLUDESTD      SN8P2X_ICE.H    ; SN8P2X_ICE.H 是一个标准宏文件，包含在“INCLUDESTD”

.CODE
            User program…
            …
```

➲ 例：由宏指令取代指令。

```
CHIP        SN8P2604
.DATA
ICE_MODE    EQU             0

            INCLUDESTD      SN8P2X_ICE.H

.CODE
            User program…

;           ADD             BUF1, A
            @ADD            BUF1, A         ;由 “@ADD M,A”代替“ADD M,A”
            …
;           AND             BUF1, A
            @AND            BUF1, A         ; 由“@AND M,A”代替“AND M,A”
            …
```

11.5.3 系统时钟

SONIX 2 系列的 8 位 MCU 具有多种可选择的系统时钟（Fosc/1~Fosc/8，Fosc/64，Fosc/128），但 ICE 固定于 Fosc/4，在 ICE 仿真模式下，用户必须确认 SN8P2604 和 ICE 的 Fcpu 频率相同，可以通过改变 ICE 的晶振频率使之和 SN8P2604 的系统时钟相匹配。

➲ **例：SN8P2604 系统时钟和 ICE 的关系表，SN8P2604 的时钟源频率为外部晶振 4MHz。**

SN8P2604 Fcpu Option	Fcpu Frequency	ICE Fcpu	ICE Crystal Frequency
Fosc/1	**4MHz**	Fosc/4	16MHz
Fosc/2	**2MHz**	Fosc/4	8MHz
Fosc/4	**1MHz**	Fosc/4	4MHz
Fosc/8	**0.5MHz**	Fosc/4	2MHz

✷ **注：对于不同速度的晶振，必须调整 ICE“HIGH CLK”选项。若 ICE 的晶振为 16MHz，则设置“HIGH CLK”选项为“X’TAL 12M”**

➲ **例：SN8P2604 系统时钟和 ICE 的关系表。SN8P2604 时钟源为内部 RC 晶振 16MHz。**

SN8P2604 Fcpu Option	Fcpu Frequency	ICE Fcpu	ICE Crystal Frequency
Fosc/1	**16MHz**	Fosc/4	-
Fosc/2	**8MHz**	Fosc/4	-
Fosc/4	**4MHz**	Fosc/4	16MHz
Fosc/8	**2MHz**	Fosc/4	8MHz

✷ **注：当 ICE 外部高速晶振等于 16MHz，并且 Fcpu = Fosc/1 和 Fosc/2 时不能仿真，对于这种应用，用户应该用 SN8P2604 的芯片验证功能。**

11.5.4 看门狗定时器

SN8P2604 的看门狗清零方法是设置 WDTR 寄存器为 0x5A，而 S8KD-2 ICE 则不是。SN8IDE 提供了一条宏指令”@RET_WDT”以便正确清看门狗。

➲ **例：通过设置 WDTR 为 0x5A 清看门狗。**

```
CHIP            SN8P2604
.DATA
ICE_MODE        EQU             0

                INCLUDESTD      SN8P2X_ICE.H

.CODE
                User program…

                MOV             A, #5Ah             ; 看门狗清零
                B0MOV           WDTR, A

                …
```

✷ **注：上述方法不能在 S8KD-2 ICE 中仿真，请使用下面的程序替代。**

➲ **例：通过“@RST_WDT”对看门狗清零。**

```
CHIP            SN8P2604
.DATA
ICE_MODE        EQU             0

                INCLUDESTD      SN8P2X_ICE.H

.CODE
                User program…

                @RST_WDT                            ; 用宏指令清看门狗

                …
```

11.5.5 P0 仿真

SN8P2604 的 P0.0 为双向 I/O 口，但是 ICE 的 P0.0 为单向输入口，PEDGE 的控制也是不同的。SN8IDE 提供宏指令控制 P0.0 的仿真，这些宏指令内置在编译器软件中。

11.5.5.1 @P00_MODE, @P01_MODE

语法：@P00_MODE Val
Val：0=设置 P0.0 为输入模式，1=设置 P0.0 为输出模式。

语法：@P01_MODE Val
Val：0=设置 P0.1 为输入模式，1=设置 P0.1 为输出模式。

➲ **例：设置 P0.0 为输入模式。**

```
@P00_MODE   0
```

✸ **注：若 P0.0/P0.1 设为输入模式，则 S8KD-2 ICE 的输入引脚为 P0.0/P0.1。**

➲ **例：设置 P0.0 为输出模式。**

```
@P00_MODE   1
```

✸ **注：若 P0.0/P0.1 为输出模式，则 S8KD-2 ICE 的输出引脚为 P6.0/P6.1。**

11.5.5.2 @P00_OUT, @P01_OUT

语法：@P00_OUT Val
Val：0=设置 P0.0 为低电平输出，1=设置 P0.0 为高电平输出。

语法：@P01_OUT Val
Val：0=设置 P0.1 为低电平输出，1=设置 P0.1 为高电平输出。

➲ **例：设置 P0.0 为高电平输出。**

```
@P00_OUT    1
```

➲ **例：设置 P0.0 为低电平输出。**

```
@P00_OUT    0
```

✸ **注：在 P0.0/P0.1 输出模式下，S8KD-2 ICE 的信号从 P6.0/P6.1 输出。**

11.5.5.3 PEDGE

PEDGE 寄存器的 P00G[1:0]的定义和 S8KD-2 ICE 不同，ICE 仿真和实际芯片的 PEDGE 功能也是不同的。

PEDGE		SN8P2604	S8KD-2 ICE
P00G1	P00G0		
0	0	保留	保留
0	1	上升沿	下降沿
1	0	下降沿	上升沿
1	1	双向	双向

SONIX 提供宏指令“@P00_EDGE”在 ICE 下仿真实际芯片的 PEDGE 功能，ICE_MODE 必须置 1 以便进行 ICE 仿真。ICE 仿真完毕后，ICE_MODE 置为 0 并重新编译获得 SN8 文件，以便实际芯片工作。

语法：@P00_EDGE Val
Val：1=上升沿，2=下降沿，3=改变电平（双向沿）

➲ **例：设置 P0.0 为上升沿中断触发和 ICE 仿真。**

```
CHIP        SN8P2604
.DATA
ICE_MODE    EQU         1           ; 设置 ICE 模式

            INCLUDESTD  SN8P2X_ICE.H

.CODE
            User program…

            @P00_EDGE   1           ; 设置 P0.0 上升沿触发
            B0BSET      FP00IEN
            …
```

➲ **例：设置 P0.0 为下降沿触发**

```
CHIP        SN8P2604
.DATA
ICE_MODE    EQU         0           ; 设置实际芯片模式

            INCLUDESTD  SN8P2X_ICE.H

.CODE
            User program…

            @P00_EDGE   2           ; 设置 P0.0 为下降沿触发
            B0BSET      FP00IEN
            …
```

11.5.6 PWM 占空比

SN8P2604 的 PWM 占空比有 ALOAD1、TC1OUT 控制。

PWM1OUT = 1					
ALOAD1	TC1OUT	TC1R 边界	PWM 占空比范围	PWM 最大频率 (Fcpu = 4M)	备注
0	0	00h to FFh	0/256 ~ 255/256	7.8125K	每计数 256 次溢出
0	1	00h to 3Fh	0/64 ~ 63/64	31.25K	每计数 64 次溢出
1	0	00h to 1Fh	0/32 ~ 31/32	62.5K	每计数 32 次溢出
1	1	00h to 0Fh	0/16 ~ 15/16	125K	每计数 16 次溢出

S8KD-2 ICE 不支持 PWM 占空比的设置功能，SONIX 提供 PWM 占空比设置的宏指令，用户可以利用宏指令仿真 PWM 功能且不会影响起他的功能。宏指令内置在编译软件中，用户必须设置 ICE_MODE 为 ICE 还是实际芯片。

语法：@PWM1_MAX_DUTY　　Max_Duty

Max_Duty	TC1 溢出边界	PWM 占空比范围	PWM 分辨率
256	FFh to 00h	0/256 ~ 255/256	8-bit
64	3Fh to 40h	0/64 ~ 63/64	6-bit
32	1Fh to 20h	0/32 ~ 31/32	5-bit
16	0Fh to 10h	0/16 ~ 15/16	4-bit

➲ **例：设置 PWM Max.Duty = 64，Duty = 2:1**

```
CHIP            SN8P2604
.DATA
ICE_MODE        EQU             0               ; 设置为实际芯片模式

                INCLUDESTD      SN8P2X_ICE.H

.CODE
                User program…

                @PWM1_MAX_DUTY  64              ; 设置 PWM1 最大占空比为 64

                MOV             A,#42           ; 42 = 63 (Max. TC0R) / 3 X 2
                B0MOV           TC0R,A
                B0BSET          FPWM0OUT
                B0BSET          FTC0ENB
                …
```

11.5.7 其它宏指令

一条宏指令包括很多条指令，直接用它跟在带有跳转功能的检测指令后会出错。

```
        BTS0        BUF.0
        @RST_WDT
        JMP         TEST_CODE
        …
TEST_CODE:
        …
```

BTS0 指令的跳转功能只跳过一条指令，@RST_WDT 是一条宏指令，由多条指令组成。上述程序的跳转功能是错误的，它不能正确地跳到“JMP TEST_CODE”，使用下面的程序就可以解决这个问题。

```
        BTS0        BUF.0
        JMP         CLR_WDT
        JMP         TEST_CODE
        …
CLR_WDT:
        @RST_WDT
        …
TEST_CODE:
        …
```

SN8IDE 提供给用户自定义的向前/向后的跳转指令以便更容易地的处理跳转功能，“Macro_Start”和“Macro_End”就是用户自定义的标签名，使用@@.Macro_Start 和@@.Macro_End，这些标签名就可以在程序中重复使用。

```
        BTS0        BUF.0
        JMP         @F.Macro_Start    ; 跳转到最近的用户自定义的@@.Macro_Start:
        JMP         @F.Macro_End      ; 跳转到最近的用户自定义的@@.Macro_End:
        …

@@.Macro_Start:
        @RST_WDT
@@.Macro_End:

TEST_CODE:
        …
```

✸ 注：仅 S8ASM V1.99K 和以后的版本支持用户自定义的向前/向后的跳转指令。

✸ 注：宏指令可能影响 ACC 和 PFLAG，用户必须检查宏指令。

12 OTP 烧录引脚

12.1 Easy Writer 转接板的引脚配置

Easy Writer JP1/JP2

VSS	2	1	VDD
CE	4	3	CLK/PGCLK
OE/ShiftDat	6	5	PGM/OTPCLK
D0	8	7	D1
D2	10	9	D3
D4	12	11	D5
D6	14	13	D7
VPP	16	15	VDD
RST	18	17	HLS
ALSB/PDB	20	19	-

JP1 for MP transition board
JP2 for Writer V3.0 transition board

Easy Writer JP3(Mapping to 48-pin text tool)

DIP1	1	48	DIP48
DIP2	2	47	DIP47
DIP3	3	46	DIP46
DIP4	4	45	DIP45
DIP5	5	44	DIP44
DIP6	6	43	DIP43
DIP7	7	42	DIP42
DIP8	8	41	DIP41
DIP9	9	40	DIP40
DIP10	10	39	DIP39
DIP11	11	38	DIP38
DIP12	12	37	DIP38
DIP13	13	36	DIP36
DIP14	14	35	DIP35
DIP15	15	34	DIP34
DIP16	16	33	DIP33
DIP17	17	32	DIP32
DIP18	18	31	DIP31
DIP19	19	30	DIP30
DIP20	20	29	DIP29
DIP21	21	28	DIP28
DIP22	22	27	DIP27
DIP23	23	26	DIP26
DIP24	24	25	DIP25

JP3 for MP transition board

12.2 Writer V3.0 和 Writer V2.5 转接板的引脚配置

GND	**1**	**2**	**VDD**
CE	**3**	**4**	**CLK**
OE	**5**	**6**	**PGM**
D0	**7**	**8**	**D1**
D2	**9**	**10**	**D3**
D4	**11**	**12**	**D5**
D6	**13**	**14**	**D7**
VPP	**15**	**16**	**VDD**
RST	**17**	**18**	**HLS**

Writer V2.5 JP1 Pin Assignment

GND	**2**	**1**	**VDD**
CE	**4**	**3**	**CLK**
OE	**6**	**5**	**PGM**
D0	**8**	**7**	**D1**
D2	**10**	**9**	**D3**
D4	**12**	**11**	**D5**
D6	**14**	**13**	**D7**
VPP	**16**	**15**	**VDD**
RST	**18**	**17**	**HLS**
	20	**19**	

Writer V3.0 JP1 Pin Assignment

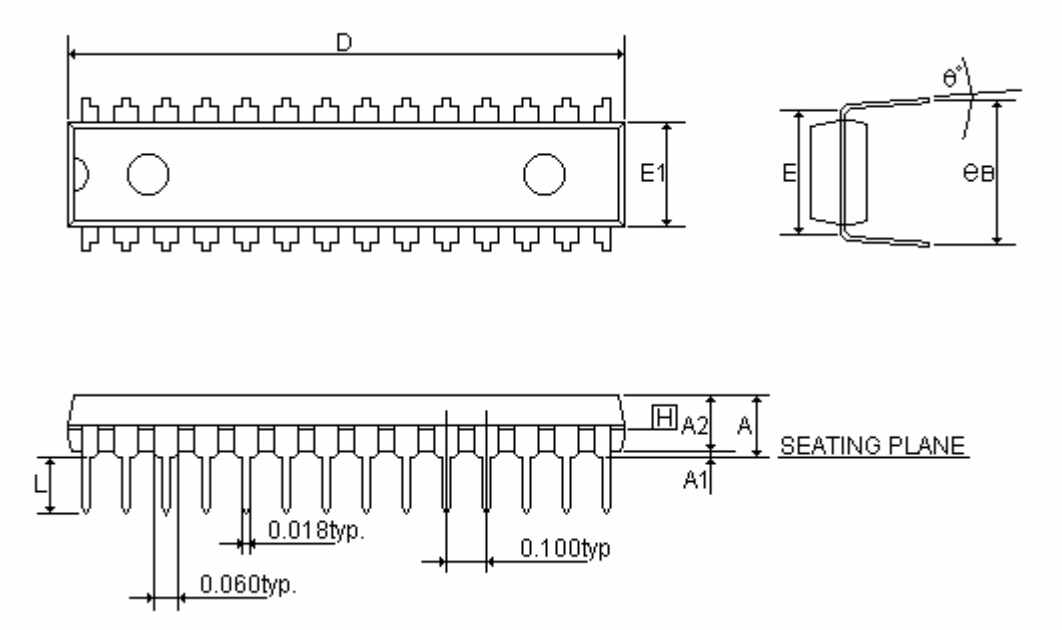
✸ 注：烧录 **SN8P2604** 时，**SONIX Writer2.5** 必须升级为 **Writer3.0**，有关 **SONIX Writer2.5** 的升级问题请联系 **SONIX** 的代理商。

12.3 SN8P2604 烧录引脚分配表

Programming Information of SN8P2600 Series					
Chip Name				SN8P2604	
EZ Writer / Writer V3.0		OTP IC / JP3 Pin Assignment			
Number	Pin			Number	Pin
1	VDD			2	VDD
2	GND			4	VSS
3	CLK			6	P5.0
4	CE			-	-
5	PGM			10	P1.0
6	OE			7	P5.1
7	D1			-	-
8	D0			-	-
9	D3			-	-
10	D2			-	-
11	D5			-	-
12	D4			-	-
13	D7			-	-
14	D6			-	-
15	VDD			-	-
16	VPP			28	RST
17	HLS			-	-
18	RST			-	-
19	-			-	-
20	ALSB/PDB			11	P1.1

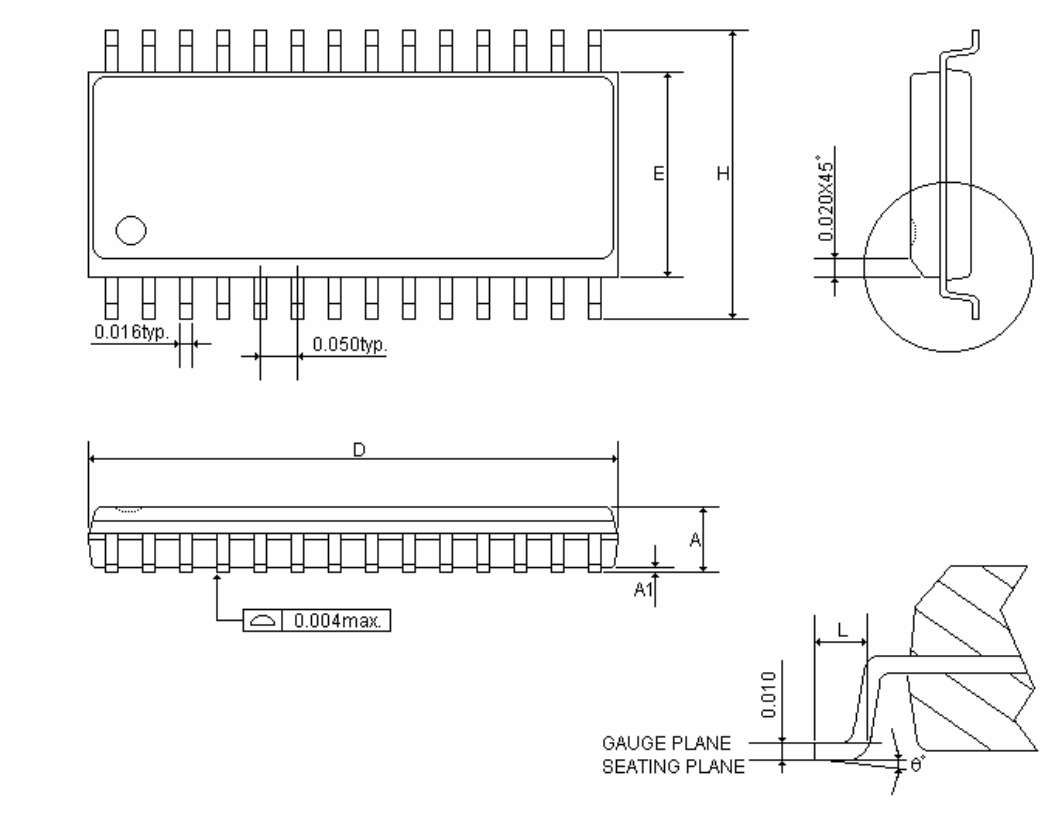
13 封装

13.1 SK-DIP 28 PIN



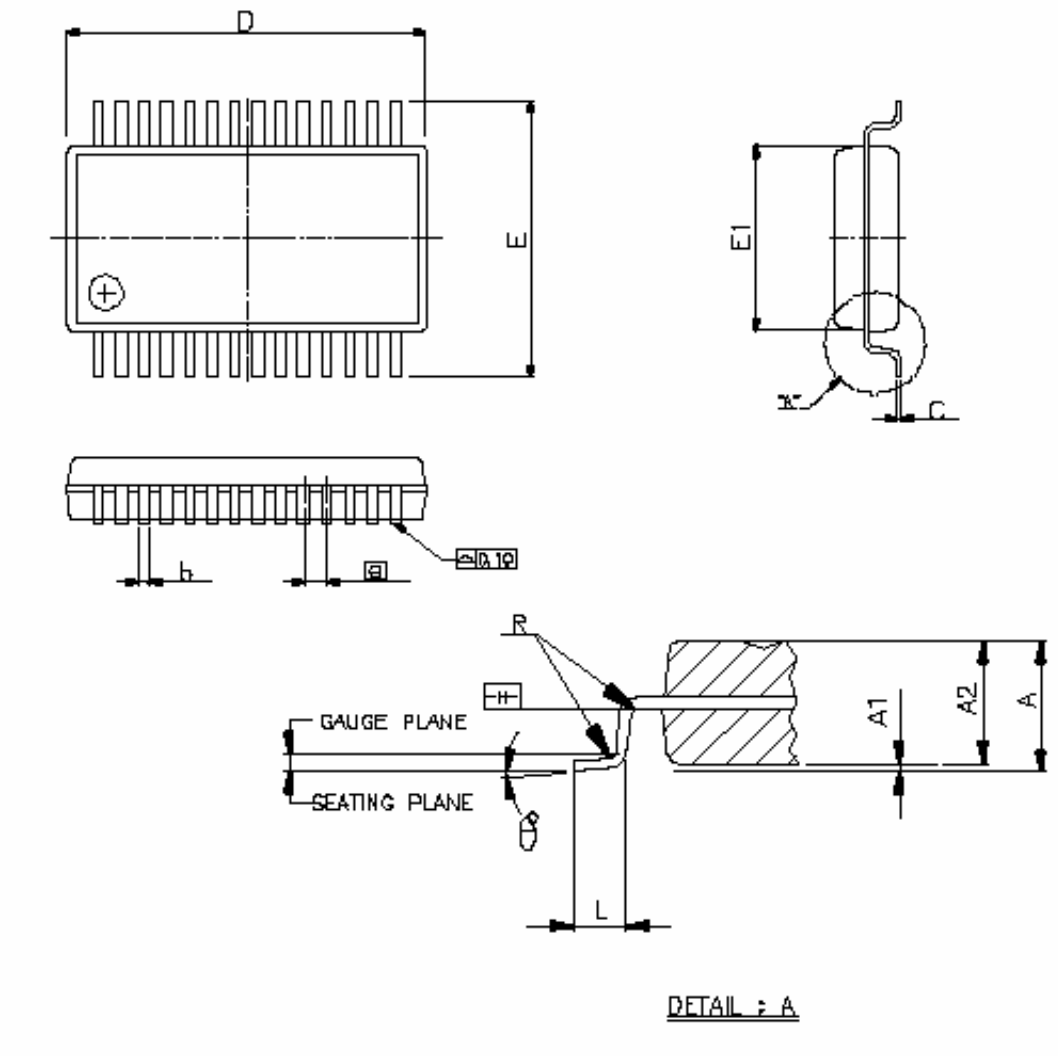
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			*(mm)*		
A	***-***	***-***	***0.210***	***-***	***-***	***5.334***
A1	***0.015***	***-***	***-***	***0.381***	***-***	***-***
A2	***0.114***	***0.130***	***0.135***	***2.896***	***3.302***	***3.429***
D	***1.390***	***1.390***	***1.400***	***35.306***	***35.306***	***35.560***
E		***0.310***			***7.874***	
E1	***0.283***	***0.288***	***0.293***	***7.188***	***7.315***	***7.442***
L	***0.115***	***0.130***	***0.150***	***2.921***	***3.302***	***3.810***
eB	***0.330***	***0.350***	***0.370***	***8.382***	***8.890***	***9.398***
θ°	***0°***	***7°***	***15°***	***0°***	***7°***	***15°***

13.2 SOP 28 PIN



SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	0.093	0.099	0.104	2.362	2.502	2.642
A1	0.004	0.008	0.012	0.102	0.203	0.305
D	0.697	0.705	0.713	17.704	17.907	18.110
E	0.291	0.295	0.299	7.391	7.493	7.595
H	0.394	0.407	0.419	10.008	10.325	10.643
L	0.016	0.033	0.050	0.406	0.838	1.270
θ°	0°	4°	8°	0°	4°	8°

13.3 SSOP 20 PIN



SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.08	-	-	2.13
A1	0.00	-	0.01	0.05	-	0.25
A2	0.06	0.07	0.07	1.63	1.75	1.88
B	0.01	-	0.01	0.22	-	0.38
C	0.00	-	0.01	0.09	-	0.20
D	0.39	0.40	0.41	9.90	10.20	10.50
E	0.29	0.31	0.32	7.40	7.80	8.20
E1	0.20	0.21	0.22	5.00	5.30	5.60
[e]	0.002BSC			0.065BSC		
L	0.02	0.04	0.04	0.63	0.90	1.03
R	0.00	-	-	0.09	-	-
θ°	0°	4°	8°	0°	4°	8°

SONIX 公司保留对以下所有产品在可靠性、功能和设计方面的改进做进一步说明的权利。SONIX 不承担由本手册所涉及的产品或电路的运用和使用所引起的任何责任。SONIX 的产品不是专门设计来应用于外科植入、生命维持和任何 SONIX 产品的故障会对个体造成伤害甚至死亡的领域。如果将 SONIX 的产品应用于上述领域，即使这些是由 SONIX 在产品设计和制造上的疏忽引起的，用户应赔偿所有费用、损失、合理的人身伤害或死亡所直接或间接产生的律师费用，并且用户保证 SONIX 及其雇员、子公司、分支机构和销售商与上述事宜无关。

Main Office:
Address: 9F, NO. 8, Hsien Cheng 5th St, Chupei City, Hsinchu, Taiwan R.O.C.
Tel: 886-3-551 0520
Fax: 886-3-551 0523

Taipei Office:
Address: 15F-2, NO. 171, Song Ted Road, Taipei, Taiwan R.O.C.
Tel: 886-2-2759 1980
Fax: 886-2-2759 8180

Hong Kong Office:
Address: Flat 3 9/F Energy Plaza 92 Granville Road, Tsimshatsui East Kowloon.
Tel: 852-2723 8086
Fax: 852-2723 9179

Technical Support by Email:
Sn8fae@sonix.com.tw

深圳技术支持中心：
地址：深圳市科技园南区 T2-B 栋 2 楼
电话：0755-26719666
传真：0755-26719786